ACM 常用算法模板

CWL

目录

说明:

　　本模板每一行代码我都使用过并 ac 过题目，但是难免有错误没测出来。请慎重使用。另外代码风格上，代码间保留空格是我大一暑假集训的时候养成的代码风格，另外常量大写等习惯后来才养成习惯。所以一些书写风格略有不规范。另外比如 LL 是 long long 的自定义类型。早期用”ll”小写的，可读性较差。因为工作量的原因。早期模板还未更正，但是正确性是验证过的

数据结构

hash

Hash 数值

```
struct Hash_table {
	static const int maxn = 1<<20;
	static const int maxm = 1<<22;
	static const int xor_mod = maxm-1;
	int first[maxm],sign;
	struct Node {
		int data,next;
	} edge[maxn];
	Hash_table() {
		clear();
	}
	void clear() {
		sign = 1;
		memset(first,0,sizeof(first));
	}
	int get_value(int val) {
		return (val<<1) & xor_mod;
	}
	bool find(int val) {
		int id = get_value(val);
		for(int i = first[id]; i; i = edge[i].next) {
			if(edge[i].data == val) {
				return 1;
			}
```

```
        }
        return 0;
    }
    void insert(int val) {
        if(find(val)) {
            return ;
        } else {
            int id = get_value(val);
            edge[sign].data = val;
            edge[sign].next = first[id];
            first[id] = sign ++;
        }
    }
} H;
```

Hash 字符串(线性探测)

```
struct Hash_map {
    static const int maxn = 1e4 + 7;//字符串的数量
    static const int MAXINT = 0x7FFFFFFF;
    struct Node {
        char str[1505];//hash 的单个字符串大小
    } mp[maxn << 2];
    inline void init() {
        memset(mp, 0, sizeof(mp));
    }
    inline int get_hash(char *str) {
        unsigned long long ans = 0;
        while(*str) {
            ans = (*str++) + (ans << 6) + (ans << 16) - ans;
        }
        return ans & MAXINT;
    }
    inline void Insert(char *str) {
        int key = get_hash(str) % (maxn << 2);
        while(mp[key].str[0]) {
            key ++;
        }
        strcpy(mp[key].str, str);
    }
    inline bool Find(char *str) {
        int key = get_hash(str) % (maxn << 2);
        bool ok = 0;
        while(1) {
            if(!mp[key].str[0]) {
                break;
            }
            if(!strcmp(mp[key].str, str)) {
                ok = 1;
                break;
            }
            key ++;
        }
        return ok;
    }
} T;
```

Hash 字符串(链式处理冲突)

```
struct Hash_map {
    static const int maxn = 1e4 + 7;
    static const int MAXINT = 0x7FFFFFFF;
    int first[maxn << 2], sign;
    struct Edge {
        int to, next;
        char str[1550];
    } edge[maxn];
    inline void init() {
        memset(first, -1, sizeof(first));
        sign = 0;
    }
    inline int get_hash(char *str) {
        unsigned long long ans = 0;
        while(*str) {
            ans = (*str++) + (ans << 6) + (ans << 16) - ans;
        }
        return ans & MAXINT;
    }
    inline void add_edge(int u, char *str) {
        strcpy(edge[sign].str, str);
        edge[sign].next = first[u];
        first[u] = sign ++;
    }
    inline void Insert(char *str) {
        int key = get_hash(str) % (maxn << 2);
        add_edge(key, str);
    }
    inline bool Find(char *str) {
```

```
		int key = get_hash(str) % (maxn << 2);
		for(int i = first[key]; ~i; i = edge[i].next) {
			if(!strcmp(str, edge[i].str)) {
				return 1;
			}
		}
		return 0;
	}
} T;
```

Hash 判断回文

```
//一个字符串，询问 l,r 区间是否为回文串
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int maxn = 1e5 + 7;
int len, flag;
struct Hash {
	LL mod, key[maxn], hs[maxn], fhs[maxn];
	static const int w = 26;
	inline void init(char s[]) {
		mod = 1e9 + rand();
		key[0] = 1;
		hs[0] = fhs[len + 1] = 0;
		for(int i = 1; i <= len; i ++ ) {
			key[i] = key[i - 1] * w % mod;
		}
		for(int i = 1; i <= len; i ++ ) {
			hs[i] = (hs[i - 1] * w + s[i] - 'a') % mod;
		}
		for(int i = len; i >= 1; i -- ) {
			fhs[i] = (fhs[i + 1] * w + s[i] - 'a') % mod;
		}
	}
	inline LL get(int l, int r) {
		return (hs[r] - hs[l - 1] * key[r - l + 1]) % mod;
	}
	inline LL getf(int l, int r) {
		return (fhs[l] - fhs[r + 1] * key[r - l + 1]) % mod;
	}
	inline bool ok(int l, int r) {
        return (get(l, r) - getf(l, r)) % mod == 0;
    }
} H[3];

char str[maxn];

int main() {
    srand(time(0));
    int m;
    while(~scanf("%d %d", &len, &m)) {
        scanf("%s", str + 1);
        H[0].init(str);
        H[1].init(str);
        while(m -- ) {
            int l, r;
            scanf("%d %d", &l, &r);
            if(H[0].ok(l, r) && H[1].ok(l, r)) {
                puts("Yes");
            } else {
                puts("No");
            }
        }
    }
    return 0;
}
```

字典树

//字典树也称前缀树,有时也可倒着插入，后缀树??其实不太准确，详见后缀自动机

指针版字典树

```
struct Trie_tree {
    struct Node {
        int cnt;
        struct Node *childs[26];
        Node() {
            cnt = 0;
            for(int i = 0; i < 26; i ++ ) {
                childs[i] = 0;
            }
        }
    } *root;
```

```
	void del(Node *now) {
		if(now == 0) {
			return ;
		}
		for(int i = 0; i < 26; i ++ ) {
			if(now->childs[i]) {
				del(now->childs[i]);
			}
		}
		delete now;
	}
	void init() {
		del(root);
		root = new Node();
	}
	inline void insert(char str[], int len) {
		Node *now = root;
		for (int i = 0; i < len; i++) {
			char ch = str[i];
			if (!(now->childs[ch - 'a'])) {
				now->childs[ch - 'a'] = new Node();
			}
			now = now->childs[ch - 'a'];
		}
		(now -> cnt) ++;
	}
	inline int match(char str[], int len) {
		Node *now = root;
		for (int i = 0; i < len; i++) {
			char ch = str[i];
			if (!(now->childs[ch - 'a'])) {
				return 0;
			}
			now = now->childs[ch - 'a'];
		}
		return now -> cnt ;
	}
} T;
```

数组版字典树

```
const int maxn = 3e5 + 7;
struct Trie_tree {
	struct Node {
		int cnt;
		int childs[26];
		inline void init() {
			memset(childs, 0, sizeof(childs));
			cnt = 0;
		}
	} tre[maxn];
	static const int root = 0;
	int tot;
	void init() {
		tre[0].init();
		tot = 1;
	}
	inline void insert(char str[], int len) {
		int now = 0;
		for (int i = 0; i < len; i++) {
			char ch = str[i];
			if (!tre[now].childs[ch - 'a']) {
				tre[tot].init();
				tre[now].childs[ch - 'a'] = tot++;
			}
			now = tre[now].childs[ch - 'a'];
		}
		tre[now].cnt ++;
	}
	inline int match(char str[], int len) {
		int now = 0;
		for (int i = 0; i < len; i++) {
			char ch = str[i];
			if (!tre[now].childs[ch - 'a']) {
				return 0;
			}
			now = tre[now].childs[ch - 'a'];
		}
		return tre[now].cnt;
	}
} T;
```

线段树

//线段树维护一切可区间合并的信息，线段树区间归并按照 home_w 学长的说法，我重载了+号。

区间更新区间求和

```
struct SegmentTree {
	struct Node {
		int l, r;
		LL sum, Lazy;
	} tree[maxn << 4];
	int num[maxn];
	inline void push_up(int rt) {
		tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
	}
	inline void push_down(int rt) {
		if(tree[rt].Lazy) {
			tree[rt << 1].Lazy += tree[rt].Lazy;
			tree[rt << 1 | 1].Lazy += tree[rt].Lazy;
			tree[rt << 1].sum += (tree[rt << 1].r - tree[rt << 1].l + 1) * tree[rt].Lazy;
			tree[rt << 1 | 1].sum += (tree[rt << 1 | 1].r - tree[rt << 1 | 1].l + 1) * tree[rt].Lazy;
			tree[rt].Lazy = 0;
		}
	}
	void build(int rt, int l, int r) {
		tree[rt].l = l;
		tree[rt].r = r;
		tree[rt].sum = 0;
		tree[rt].Lazy = 0;
		if(l == r) {
			tree[rt].sum = num[l];
			return ;
		}
		int mid = (l + r) >> 1;
		build(rt << 1, l, mid);
		build(rt << 1 | 1, mid + 1, r);
		push_up(rt);
	}
	void update(int rt, int l, int r, LL val) {
		if(l <= tree[rt].l && tree[rt].r <= r) {
			tree[rt].sum += (tree[rt].r - tree[rt].l + 1) * val;
			tree[rt].Lazy += val;
			return ;
		}
		push_down(rt);
		int mid = (tree[rt].l + tree[rt].r) >> 1;
		if(r <= mid) {
```

```
            update(rt << 1, l, r, val);
        } else if(l > mid) {
            update(rt << 1 | 1, l, r, val);
        } else {
            update(rt << 1, l, mid, val);
            update(rt << 1 | 1, mid + 1, r, val);
        }
        push_up(rt);
    }
    LL query(int rt, int l, int r) {
        if(l <= tree[rt].l && tree[rt].r <= r) {
            return tree[rt].sum;
        }
        push_down(rt);
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(r <= mid) {
            return query(rt << 1, l, r);
        } else if(l > mid) {
            return query(rt << 1 | 1, l, r);
        } else {
            return query(rt << 1, l, mid) + query(rt << 1 | 1, mid + 1, r);
        }
    }
} Seg;
```

单点更新区间求和

```
struct Segment_tree {
    struct Node {
        int sum,l,r;
    } t[maxn << 2];
    int num[maxn];
    void build(int rt,int l,int r) {
        t[rt].l = l;
        t[rt].r = r;
        if(l == r) {
            t[rt].sum = num[l];
            return ;
        } else {
            int mid = (l + r) >> 1;
            build(rt<<1,l,mid);
            build(rt<<1|1,mid+1,r);
            t[rt].sum = t[rt<<1].sum + t[rt<<1|1].sum;
        }
        }
        void update(int rt,int pos,int val) {
                if(t[rt].l == t[rt].r) {
                        t[rt].sum += val;
                        return ;
                }
                int mid = (t[rt].l + t[rt].r) >> 1;
                if(pos <= mid) {
                        update(rt<<1,pos,val);
                } else {
                        update(rt<<1|1,pos,val);
                }
                t[rt].sum = t[rt<<1].sum + t[rt<<1|1].sum;
        }
        int query(int rt,int l,int r) {
                if(l <= t[rt].l && t[rt].r <= r) {
                        return t[rt].sum;
                }
                int mid = (t[rt].l + t[rt].r) >> 1;
                if(r <= mid) {
                        return query(rt<<1,l,r);
                } else if(l > mid) {
                        return query(rt<<1|1,l,r);
                } else {
                        return query(rt<<1,l,mid) + query(rt<<1|1,mid+1,r);
                }
        }
} S;
```

区间加区间求和

```
struct Segment_tree {
        struct Node {
                int l,r;
                LL sum,lazy;
        } t[maxn << 2];
        int num[maxn];
        void build(int rt,int l,int r) {
                t[rt].l = l;
                t[rt].r = r;
                t[rt].lazy = 0;
                if(l == r) {
                        t[rt].sum = num[l];
                        return ;
                }
                int mid = (l + r) >> 1;
                build(rt<<1,l,mid);
                build(rt<<1|1,mid+1,r);
                t[rt].sum = t[rt<<1].sum + t[rt<<1|1].sum;
        }
        void push_down(int rt) {
                if(t[rt].lazy) {
                        t[rt<<1].lazy += t[rt].lazy;
                        t[rt<<1].sum += (t[rt<<1].r-t[rt<<1].l+1)*t[rt].lazy;
                        t[rt<<1|1].lazy += t[rt].lazy;
                        t[rt<<1|1].sum += (t[rt<<1|1].r-t[rt<<1|1].l+1)*t[rt].lazy;
                        t[rt].lazy = 0;
                }
        }
        void update(int rt,int l,int r,int x) {
                if(l <= t[rt].l && t[rt].r <= r) {
                        t[rt].lazy += x;
                        t[rt].sum += (r - l + 1) * x;
                        return ;
                }
                push_down(rt);
                int mid = (t[rt].l + t[rt].r) >> 1;
                if(r <= mid) {
                        update(rt<<1,l,r,x);
                } else if(l > mid) {
                        update(rt<<1|1,l,r,x);
                } else {
                        update(rt<<1,l,mid,x);
                        update(rt<<1|1,mid+1,r,x);
                }
                t[rt].sum = t[rt<<1].sum + t[rt<<1|1].sum;
        }
        LL query(int rt,int l,int r) {
                if(l <= t[rt].l&& t[rt].r <= r) {
                        return t[rt].sum;
                }
                push_down(rt);
                int mid = (t[rt].l + t[rt].r) >> 1;
                if(r <= mid) {
                        return query(rt<<1,l,r);
                } else if(l > mid) {
                        return query(rt<<1|1,l,r);
                } else {
                return query(rt<<1,l,mid) + query(rt<<1|1,mid+1,r);
            }
        }
} S;
```

线段树区间加，询问区间%7==0 的数

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6+7;
struct Segment_tree {
        struct Node {
                int l,r,mod[7],lazy;
        } tre[maxn << 2];
        int arr[maxn];
        inline void push_up(int rt) {
                for(int i = 0; i < 7; i ++ ) {
                        tre[rt].mod[i] = tre[rt<<1].mod[i] + tre[rt<<1|1].mod[i];
                }
        }
        inline void push_down(int rt) {
                if(tre[rt].lazy) {
                        tre[rt<<1].lazy += tre[rt].lazy;
                        tre[rt<<1|1].lazy += tre[rt].lazy;
                        update_mod(rt<<1,tre[rt].lazy);
                        update_mod(rt<<1|1,tre[rt].lazy);
                        tre[rt].lazy = 0;
                }
        }
        inline void update_mod(int rt,int val) {
                int tmp[7] = {0};
                for(int i = 0; i < 7; i ++ ) {
                        tmp[(i + val) % 7] = tre[rt].mod[i];
                }
                for(int i = 0; i < 7; i ++ ) {
                        tre[rt].mod[i] = tmp[i];
                }
        }
        void build(int rt,int l,int r) {
                tre[rt].l = l;
                tre[rt].r = r;
                tre[rt].lazy = 0;
                if(l == r) {
```

```
			for(int i = 0; i < 7; i ++ ) {
				tre[rt].mod[i] = 0;
			}
			tre[rt].mod[arr[l] % 7] ++;
			return ;
		}
		int mid = (l + r) >> 1;
		build(rt<<1,l,mid);
		build(rt<<1|1,mid+1,r);
		push_up(rt);
	}
	void update(int rt,int l,int r,int val) {
		push_down(rt);
		if(l <= tre[rt].l && tre[rt].r <= r) {
			update_mod(rt,val);
			tre[rt].lazy += val;
			return ;
		}
		int mid = (tre[rt].l + tre[rt].r) >> 1;
		if(r <= mid) {
			update(rt<<1,l,r,val);
		} else if(l > mid) {
			update(rt<<1|1,l,r,val);
		} else {
			update(rt<<1,l,mid,val);
			update(rt<<1|1,mid+1,r,val);
		}
		push_up(rt);
	}
	int query(int rt,int l,int r) {
		push_down(rt);
		if(l <= tre[rt].l && tre[rt].r <= r) {
			return tre[rt].mod[0];
		}
		int mid = (tre[rt].l + tre[rt].r) >> 1;
		if(r <= mid) {
			return query(rt<<1,l,r);
		} else if(l > mid) {
			return query(rt<<1|1,l,r);
		} else {
			return query(rt<<1,l,mid) + query(rt<<1|1,mid+1,r);
		}
	}
} S;
```

```
int n,q;
int main() {
	while(~scanf("%d",&n)) {
		for(int i = 1; i <= n; i ++ ) {
			scanf("%d",&S.arr[i]);
		}
		S.build(1,1,n);
		scanf("%d",&q);
		char op[6];
		int x,y,z;
		while(q--) {
			scanf("%s",op);
			if(op[0] == 'c') {
				scanf("%d %d",&x,&y);
				printf("%d\n",S.query(1,x,y));
			} else {
				scanf("%d %d %d",&x,&y,&z);
				S.update(1,x,y,z);
			}
		}
	}
	return 0;
}
```

线段树同时包含加法与乘法操作（两种标记相互影响）

```
/// (val*mul+add)*mul_lazy+add_lazy
///展开后子结点的 lazy 如下
/// val*[mul*mul_lazy]+[add*mul_lazy+add_lazy]
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 10;
typedef long long LL;
LL n, m, mod, opt, x, y, k, arr[MAXN];
struct Node {
	int l, r;
	LL val, add, mul, sz;
} tree[MAXN << 2];
void push_up(int rt) {
	tree[rt].val = (tree[rt << 1].val + tree[rt << 1 | 1].val) % mod;
}
void push_down(int rt) {
	int l = tree[rt].l, r = tree[rt].r, m = (l + r) >> 1;
	if(tree[rt].add != 0 || tree[rt].mul != 1) {
		tree[rt << 1].mul = (tree[rt << 1].mul * tree[rt].mul) % mod;
		tree[rt << 1].add = (tree[rt << 1].add * tree[rt].mul + tree[rt].add) % mod;
		tree[rt << 1].val = ((tree[rt << 1].val * tree[rt].mul) + (tree[rt].add * tree[rt << 1].sz % mod)) % mod;
		tree[rt << 1 | 1].mul = (tree[rt << 1 | 1].mul * tree[rt].mul) % mod;
		tree[rt << 1 | 1].add = (tree[rt << 1 | 1].add * tree[rt].mul + tree[rt].add) % mod;
		tree[rt << 1 | 1].val = ((tree[rt << 1 | 1].val * tree[rt].mul) % mod + (tree[rt].add * tree[rt << 1 | 1].sz) % mod) % mod;
		tree[rt].add = 0;
		tree[rt].mul = 1;
	}
}
void build(int rt, int l, int r) {
	tree[rt].l = l;
	tree[rt].r = r;
	tree[rt].add = 0;
	tree[rt].mul = 1;
	tree[rt].sz = r - l + 1;
	if(l == r) {
		tree[rt].val = arr[l];
		return ;
	}
	int mid = (l + r) >> 1;
	build(rt << 1, l, mid);
	build(rt << 1 | 1, mid + 1, r);
	push_up(rt);
}
void update_mul(int rt, int l, int r, LL x) {
	if(l <= tree[rt].l && tree[rt].r <= r) {
		tree[rt].val = (tree[rt].val * x) % mod;
		tree[rt].mul = (tree[rt].mul * x) % mod;
		tree[rt].add = (tree[rt].add * x) % mod;
		return ;
	}
	push_down(rt);
	int mid = (tree[rt].l + tree[rt].r) >> 1;
	if(r <= mid) {
		update_mul(rt << 1, l, r, x);
	} else if(l > mid) {
		update_mul(rt << 1 | 1, l, r, x);
	} else {
		update_mul(rt << 1, l, mid, x);
		update_mul(rt << 1 | 1, mid + 1, r, x);
	}
	push_up(rt);
}
void update_add(int rt, int l, int r, LL x) {
	if(l <= tree[rt].l && tree[rt].r <= r) {
		tree[rt].val = (tree[rt].val + (x * (tree[rt].r - tree[rt].l + 1)) % mod) % mod;
		tree[rt].add = (tree[rt].add + x) % mod;
		return ;
	}
	int mid = (tree[rt].l + tree[rt].r) >> 1;
	push_down(rt);
	if(r <= mid) {
		update_add(rt << 1, l, r, x);
	} else if(l > mid) {
		update_add(rt << 1 | 1, l, r, x);
	} else {
		update_add(rt << 1, l, mid, x);
		update_add(rt << 1 | 1, mid + 1, r, x);
	}
	push_up(rt);
}
LL query(int rt, int l, int r) {
	if(l <= tree[rt].l && tree[rt].r <= r) {
		return tree[rt].val;
	}
	int mid = (tree[rt].l + tree[rt].r) >> 1;
	push_down(rt);
	if(r <= mid) {
		return query(rt << 1, l, r);
	} else if(l > mid) {
		return query(rt << 1 | 1, l, r);
	} else {
		return (query(rt << 1, l, mid) % mod + query(rt << 1 | 1, mid + 1, r) % mod) % mod;
	}
}
int main() {
	scanf("%lld %lld %lld", &n, &m, &mod);
	for(int i = 1; i <= n; i++ ) {
		scanf("%lld", &arr[i]);
		arr[i] %= mod;
	}
	build(1, 1, n);
	for(int i = 1; i <= m; i++ ) {
		scanf("%lld", &opt);
```

```
        if(opt == 1) { ///mul
            scanf("%lld %lld %lld", &x, &y, &k);
            k %= mod;
            update_mul(1, x, y, k);
        } else if(opt == 2) { ///add
            scanf("%lld %lld %lld", &x, &y, &k);
            k %= mod;
            update_add(1, x, y, k);
        } else {
            scanf("%lld %lld", &x, &y);
            printf("%lld\n", query(1, x, y) % mod);
        }
    }
    return 0;
}
```

线段树各种操作

```
/**
有 n 个数和 5 种操作

add a b c：把区间[a,b]内的所有数都增加 c

set a b c：把区间[a,b]内的所有数都设为 c

sum a b：查询区间[a,b]的区间和

max a b：查询区间[a,b]的最大值

min a b：查询区间[a,b]的最小值

*/
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 7;
const long long INF    = 1LL << 62;
struct Segment_tree {
    struct Node {
        int l, r;
        long long sum, max, min, set_lazy, add_lazy;
    } tre[maxn << 2];
    long long arr[maxn];
    inline void push_up(int rt) {
        if(tre[rt].l == tre[rt].r) {
            return ;
        }
        tre[rt].sum = tre[rt<<1].sum + tre[rt<<1|1].sum;
        tre[rt].max = max(tre[rt<<1].max, tre[rt<<1|1].max);
        tre[rt].min = min(tre[rt<<1].min, tre[rt<<1|1].min);
    }
    inline void push_down(int rt) {
        if(tre[rt].set_lazy) { ///if set_lazy add_lazy = 0
            tre[rt<<1].set_lazy += tre[rt].set_lazy;
            tre[rt<<1].sum = (tre[rt<<1].r - tre[rt<<1].l + 1) * tre[rt].set_lazy;
            tre[rt<<1].max = tre[rt].set_lazy;
            tre[rt<<1].min = tre[rt].set_lazy;

            tre[rt<<1|1].set_lazy += tre[rt].set_lazy;
            tre[rt<<1|1].sum = (tre[rt<<1|1].r - tre[rt<<1|1].l + 1) * tre[rt].set_lazy;
            tre[rt<<1|1].max = tre[rt].set_lazy;
            tre[rt<<1|1].min = tre[rt].set_lazy;

            tre[rt].add_lazy = 0;
            tre[rt<<1].add_lazy = tre[rt<<1|1].add_lazy = 0;
            tre[rt].set_lazy = 0;
            return ;
        }
        if(tre[rt].add_lazy) {
            tre[rt<<1].add_lazy += tre[rt].add_lazy;
            tre[rt<<1].sum += (tre[rt<<1].r - tre[rt<<1].l + 1) * tre[rt].add_lazy;
            tre[rt<<1].max += tre[rt].add_lazy;
            tre[rt<<1].min += tre[rt].add_lazy;

            tre[rt<<1|1].add_lazy += tre[rt].add_lazy;
            tre[rt<<1|1].sum += (tre[rt<<1|1].r - tre[rt<<1|1].l + 1) * tre[rt].add_lazy;
            tre[rt<<1|1].max += tre[rt].add_lazy;
            tre[rt<<1|1].min += tre[rt].add_lazy;

            tre[rt].add_lazy = 0;
        }
    }
    void build(int rt,int l,int r) {
        tre[rt].l = l;
        tre[rt].r = r;
        tre[rt].set_lazy = 0;
        tre[rt].add_lazy = 0;
        if(l == r) {
            tre[rt].sum = tre[rt].max = tre[rt].min = arr[l];
            return ;
        }
        int mid = (l + r) >> 1;
        build(rt<<1,l,mid);
        build(rt<<1|1,mid+1,r);
        push_up(rt);
    }
    void update1(int rt,int l,int r,long long val) { ///add
        push_down(rt);
        if(l == tre[rt].l && tre[rt].r == r) {
            tre[rt].add_lazy = val;
            tre[rt].sum += (tre[rt].r - tre[rt].l + 1) * val;
            tre[rt].max += val;
            tre[rt].min += val;
            return ;
        }
        int mid = (tre[rt].l + tre[rt].r) >> 1;
        if(r <= mid) {
            update1(rt<<1,l,r,val);
        } else if(l > mid) {
            update1(rt<<1|1,l,r,val);
        } else {
            update1(rt<<1,l,mid,val);
            update1(rt<<1|1,mid+1,r,val);
        }
        push_up(rt);
    }
    void update2(int rt,int l,int r,long long val) { ///set
        push_down(rt);
        if(l == tre[rt].l && tre[rt].r == r) {
            tre[rt].set_lazy = val;
            tre[rt].sum = (tre[rt].r - tre[rt].l + 1) * val;
            tre[rt].max = val;
            tre[rt].min = val;
            tre[rt].add_lazy = 0;
            return ;
        }
        int mid = (tre[rt].l + tre[rt].r) >> 1;
        if(r <= mid) {
            update2(rt<<1,l,r,val);
        } else if(l > mid) {
            update2(rt<<1|1,l,r,val);
        } else {
            update2(rt<<1,l,mid,val);
            update2(rt<<1|1,mid+1,r,val);
        }
        push_up(rt);
```

```
}
long long query1(int rt,int l,int r) { ///sum
        push_down(rt);
        if(l == tre[rt].l && tre[rt].r == r) {
                return tre[rt].sum;
        }
        int mid = (tre[rt].l + tre[rt].r) >> 1;
        if(r <= mid) {
                return query1(rt<<1,l,r);
        } else if(l > mid) {
                return query1(rt<<1|1,l,r);
        } else {
                return query1(rt<<1,l,mid) + query1(rt<<1|1,mid+1,r);
        }
}
long long query2(int rt,int l,int r) { ///max
        push_down(rt);
        if(l == tre[rt].l && tre[rt].r == r) {
                return tre[rt].max;
        }
        int mid = (tre[rt].l + tre[rt].r) >> 1;
        if(r <= mid) {
                return query2(rt<<1,l,r);
        } else if(l > mid) {
                return query2(rt<<1|1,l,r);
        } else {
                return max(query2(rt<<1,l,mid), query2(rt<<1|1,mid+1,r));
        }
}
long long query3(int rt,int l,int r) { ///min
        push_down(rt);
        if(l == tre[rt].l && tre[rt].r == r) {
                return tre[rt].min;
        }
        int mid = (tre[rt].l + tre[rt].r) >> 1;
        if(r <= mid) {
                return query3(rt<<1,l,r);
        } else if(l > mid) {
                return query3(rt<<1|1,l,r);
        } else {
                return min(query3(rt<<1,l,mid), query3(rt<<1|1,mid+1,r));
        }
}
```

```
} S;
int n,q,x,y;
long long val;
string op;
int main() {
	while(~scanf("%d %d",&n,&q)) {
		for(int i = 1; i <= n; i ++ ) {
			scanf("%lld",&S.arr[i]);
		}
		S.build(1,1,n);
		while(q--) {
			cin >> op;
			if(op == "add") {
				scanf("%d %d %lld",&x,&y,&val);
				S.update1(1,x,y,val);
			} else if(op == "set") {
				scanf("%d %d %lld",&x,&y,&val);
				S.update2(1,x,y,val);
			} else if(op == "sum") {
				scanf("%d %d",&x,&y);
				printf("%lld\n",S.query1(1,x,y));
			} else if(op == "max") {
				scanf("%d %d",&x,&y);
				printf("%lld\n",S.query2(1,x,y));
			} else if(op == "min") {
				scanf("%d %d",&x,&y);
				printf("%lld\n",S.query3(1,x,y));
			}
		}
	}
	return 0;
}
```

HDU1542 线段树扫描线,矩形面积交

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2005 + 7;
int T, n;
struct Line {
	double lx, rx, y;
	int kind;
	inline void init(double l, double r, double yy, int k) {
		lx = l, rx = r, y = yy, kind = k;
```

```
        }
} line[maxn << 2];
struct Node {
        int l, r, status;
        double length;
} tree[maxn << 2];
vector<double>v;
inline int getid(double x) {
        return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
}
bool cmp1(Line a, Line b) {
        return a.y < b.y;
}
void build(int rt, int l, int r) {
        tree[rt].l = l, tree[rt].r = r;
        tree[rt].status = tree[rt].length = 0;
        if(l == r) {
                return ;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid + 1, r);
}
inline void push_up(int rt) {
        if(tree[rt].status) {
                tree[rt].length = v[ tree[rt].r ] - v[ tree[rt].l - 1 ];
        } else if(tree[rt].l == tree[rt].r) {
                tree[rt].length = 0;
        } else {
                tree[rt].length = tree[rt << 1].length + tree[rt << 1 | 1].length;
        }
}
void update(int rt, int l, int r, int x) {
        if(l <= tree[rt].l && tree[rt].r <= r) {
                tree[rt].status += x;
                push_up(rt);
                return ;
        }
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(r <= mid) {
                update(rt << 1, l, r, x);
        } else if(l > mid) {
                update(rt << 1 | 1, l, r, x);
        } else {
```

```
        update(rt << 1, l, mid, x);
        update(rt << 1 | 1, mid + 1, r, x);
    }
    push_up(rt);
}
int main() {
    int n, cas = 1;
    double x1, x2, y1, y2;
    while(~scanf("%d", &n) && n) {
        v.clear();
        for(int i = 1; i <= n; i ++ ) {
            scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
            line[i].init(x1, x2, y1, -1);
            line[i + n].init(x1, x2, y2, 1);
            v.push_back(x1), v.push_back(x2);
        }
        sort(line + 1, line + 1 + n * 2, cmp1);
        sort(v.begin(), v.end());
        build(1, 1, 2 * n);
        double ans = 0;
        for(int i = 1; i < 2 * n; i ++ ) {
            int l = getid(line[i].lx), r = getid(line[i].rx) - 1;
            update(1, l, r, line[i].kind);
            ans += (line[i + 1].y - line[i].y) * tree[1].length;
        }
        printf("Test case #%d\nTotal explored area: %.2f\n\n", cas ++, ans);
    }
    return 0;
}
```

线段树区间归并 01 区间最大子段和

```
//单点更新，询问区间最大连续子段和
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 500500;
const int INF = 0x7FFFFFFF;
struct Node {
    int l, r;
    int sum, dat, lmax, rmax;
} tree[MAXN << 2];
int follow, res;
int n, m, a[MAXN], x, y, k;
void push_up(int rt) {
    tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
    tree[rt].lmax = max(tree[rt << 1].lmax, tree[rt << 1].sum + tree[rt << 1 | 1].lmax);
    tree[rt].rmax = max(tree[rt << 1 | 1].rmax, tree[rt << 1 | 1].sum + tree[rt << 1].rmax);
    tree[rt].dat = max(tree[rt << 1].dat, tree[rt << 1 | 1].dat);
    tree[rt].dat = max(tree[rt].dat, tree[rt << 1].rmax + tree[rt << 1 | 1].lmax);
}
void build(int rt, int l, int r) {
    tree[rt].l = l;
    tree[rt].r = r;
    if(l == r) {
            tree[rt].sum = tree[rt].dat = tree[rt].lmax = tree[rt].rmax = a[l];
            return ;
    }
    int mid = (l + r) >> 1;
    build(rt << 1, l, mid);
    build(rt << 1 | 1, mid + 1, r);
    push_up(rt);
}
void update(int rt, int pos, int val) {
    if(tree[rt].l == tree[rt].r) {
            tree[rt].sum = tree[rt].lmax = tree[rt].rmax = tree[rt].dat = val;
            return ;
    }
    int mid = (tree[rt].l + tree[rt].r) >> 1;
    if(pos <= mid) {
            update(rt << 1, pos, val);
    } else {
            update(rt << 1 | 1, pos, val);
    }
    push_up(rt);
}
Node query(int rt, int l, int r) {
    if(l <= tree[rt].l && tree[rt].r <= r) {
            return tree[rt];
    }
    int mid = (tree[rt].l + tree[rt].r) >> 1;
    if(r <= mid) {
            return query(rt << 1, l, r);
    } else if(l > mid) {
            return query(rt << 1 | 1, l, r);
    } else {
            Node now, lNode, rNode;
```

```
            lNode = query(rt << 1, l, mid);
            rNode = query(rt << 1 | 1, mid + 1, r);
            now.sum = lNode.sum + rNode.sum;
            now.lmax = max(lNode.lmax, lNode.sum + rNode.lmax);
            now.rmax = max(rNode.rmax, rNode.sum + lNode.rmax);
            now.dat = max(lNode.dat, rNode.dat);
            now.dat = max(now.dat, lNode.rmax + rNode.lmax);
            return now;
        }
}
int main() {
        while(~scanf("%d %d", &n, &m)) {
            for(int i = 1; i <= n; i++) {
                scanf("%d", &a[i]);
            }
            build(1, 1, n);
            while(m--) {
                scanf("%d %d %d", &k, &x, &y);
                if(k == 1) {
                    if(x > y) {
                        swap(x, y);
                    }
                    printf("%d\n", query(1, x, y).dat);
                } else {
                    update(1, x, y);
                }
            }
        }
        return 0;
}
```

线段树区间归并 02(区间出现最多的数)

```
//出现次数最多的数的个数
#include <bits/stdc++.h>
const int MAXN = 100000 + 7;
int n, q, a[MAXN];
struct Node {
        int l, r;
        int l_tp, r_tp;
        int l_len, r_len;
        int len, tp;
        int get_len() {
            return r - l + 1;
```

```
	}
} tree[MAXN * 4];
Node operator + (const Node &a, const Node &b) {
	Node now;
	now.l = a.l, now.r = b.r;
	now.l_tp = a.l_tp, now.r_tp = b.r_tp;
	if(a.l_len == (a.r - a.l + 1) && a.r_tp == b.l_tp) {
		now.l_len = (a.r - a.l + 1) + b.l_len;
	} else {
		now.l_len = a.l_len;
	}
	if(b.r_len == (b.r - b.l + 1) && a.r_tp == b.l_tp) {
		now.r_len = (b.r - b.l + 1) + a.r_len;
	} else {
		now.r_len = b.r_len;
	}
	now.len = max(a.len, b.len);
	if(a.r_tp == b.l_tp) {
		now.len = max(now.len, a.r_len + b.l_len);
	}
	return now;
}
void build(int rt, int l, int r) {
	tree[rt].l = l;
	tree[rt].r = r;
	if(l == r) {
		tree[rt].tp = tree[rt].l_tp = tree[rt].r_tp = a[l];
		tree[rt].len = tree[rt].l_len = tree[rt].r_len = 1;
		return ;
	}
	int mid = (l + r) >> 1;
	build(rt << 1, l, mid);
	build(rt << 1 | 1, mid + 1, r);
	tree[rt] = tree[rt << 1] + tree[rt << 1 | 1];
}
int query(int rt, int l, int r) {
	if(l <= tree[rt].l && tree[rt].r <= r) {
		return tree[rt].len;
	}
	int mid = (tree[rt].r + tree[rt].l) >> 1;
	if(r <= mid) {
		return query(rt << 1, l, r);
	} else if(l > mid) {
		return query(rt << 1 | 1, l, r);
	} else {
		int res = max(query(rt << 1, l, mid), query(rt << 1 | 1, mid + 1, r));
		if(tree[rt << 1].r_tp == tree[rt << 1 | 1].l_tp) {
			res = max(res, min(tree[rt << 1].r_len, mid - l + 1) + min(tree[rt << 1 | 1].l_len, r - (mid + 1) + 1));
		}
		return res;
	}
}
int main() {
	while(~scanf("%d", &n), n) {
		scanf("%d", &q);
		for(int i = 1; i <= n; i++ ) {
			scanf("%d", &a[i]);
		}
		build(1, 1, n);
		while(q--) {
			int l, r;
			scanf("%d %d", &l, &r);
			printf("%d\n", query(1, l, r));
		}
	}
	return 0;
}
```

ZKW 单点更新线段树

```
//一种非递归的线段树，常数小，但是维护更复杂的信息苣蒻的我不会，只学了个单点更新
struct Segment_Tree {
	int tree[maxn << 2], tot;
	inline void push_up(int rt) {
		tree[rt] = tree[rt<<1] + tree[rt<<1|1];
	}
	inline void build(int n) {
		for(tot = 1; tot <= n; tot <<= 1);
		memset(tree, 0, sizeof(tree));
	}
	inline void build(int arr[], int n) {
		build(n);
		for(int i = 1; i <= n; i ++ ) tree[i + tot] = arr[i];
		for(int i = tot-1;	i; i -- ) push_up(i);
	}
	inline void update(int pos, int val) {
		for(tree[pos += tot] += val, pos >>= 1; pos; pos >>= 1) push_up(pos);
	}
	inline int query(int l, int r) {
		int ans = 0;
		for(l += tot - 1, r += tot + 1; l ^ r ^ 1; l >>= 1, r >>= 1) {
			if(~l&1) ans += tree[l^1];
			if(r&1)    ans += tree[r^1];
		}
		return ans;
	}
} S;
```

主席树

//又称可持久化线段树，可解决区间第 k 大的问题，也可维护保存历史版本的多可线段树,两种写法，一种是 b 站学的引用维护结点，一种是通过返回值，在算法竞赛进阶指南看的。

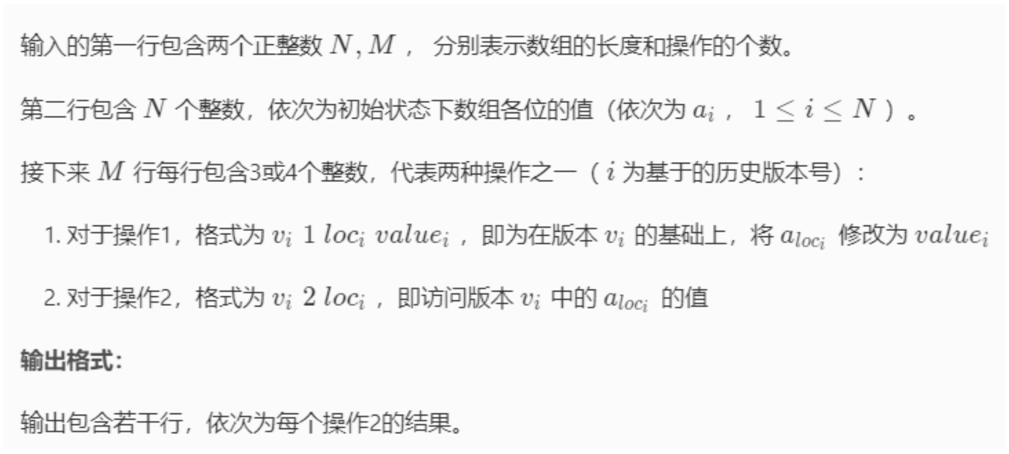
区间第 K 大 POJ 版本 1

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;
const int maxn = 2e5 + 7;
int n, m, cnt, root[maxn], a[maxn], x, y, k;
struct Node {
	int sum, l, r;
} tree[maxn * 40];
vector <int> v;
inline int get_id(int x) {
	return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
}
inline void update(int l,int r,int &x,int y,int pos) {
	tree[ ++cnt ] = tree[y], tree[cnt].sum ++, x = cnt;
	if(l == r) {
		return ;
	}
	int mid = (l + r) >> 1;
	if(pos <= mid) {
		update(l, mid, tree[x].l, tree[y].l, pos);
	} else {
        update(mid + 1, r, tree[x].r, tree[y].r, pos);
    }
}
inline int query(int l,int r,int x,int y,int k) {
    if(l == r) {
        return l;
    }
    int mid = (l + r) >> 1;
    int sum = tree[ tree[y].l ].sum - tree[ tree[x].l ].sum;
    if(sum >= k) {
        return query(l, mid, tree[x].l, tree[y].l, k);
    } else {
        return query(mid + 1, r, tree[x].r, tree[y].r, k - sum);
    }
}
int main() {
    while(~scanf("%d %d",&n, &m)) {
        v.clear();
        for(int i = 1; i <= n; i ++ ) {
            scanf("%d", &a[i]);
            v.push_back(a[i]);
        }
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(), v.end()), v.end());
        for(int i = 1; i <= n; i ++ ) {
            update(1, n, root[i], root[i - 1], get_id(a[i]));
        }
        for(int i = 1; i <= m; i ++ ) {
            scanf("%d %d %d",&x, &y, &k);
            printf("%d\n",v[query(1, n, root[x-1], root[y], k) - 1]);
        }
    }
    return 0;
}
//另一版本******************************************
#include <bits/stdc++.h>
using namespace std;
const int N = 100010;
const int INF = 0x7FFFFFFF;
struct Node {
    int l, r, sum;
} tree[N * 20];
int n, m, tot, root[N], a[N];
vector<int>v;
```

```
inline int getid(int x) {
        return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
}
int build(int l, int r) {
        int p = ++tot;
        tree[p].sum = 0;
        if(l == r) {
                return p;
        }
        int mid = (l + r) >> 1;
        tree[p].l = build(l, mid);
        tree[p].r = build(mid + 1, r);
        return p;
}
int insert(int now, int l, int r, int pos, int val) {
        int p = ++tot;
        tree[p] = tree[now];
        if(l == r) {
                tree[p].sum += val;
                return p;
        }
        int mid = (l + r) >> 1;
        if(pos <= mid) {
                tree[p].l = insert(tree[now].l, l, mid, pos, val);
        } else {
                tree[p].r = insert(tree[now].r, mid + 1, r, pos, val);
        }
        tree[p].sum = tree[ tree[p].l ].sum + tree[ tree[p].r ].sum;
        return p;
}
int query(int p, int q, int l, int r, int k) {
        if(l == r) {
                return l;
        }
        int mid = (l + r) >> 1;
        int cnt = tree[ tree[p].l ].sum - tree[ tree[q].l ].sum;
        if(k <= cnt) {
                return query(tree[p].l, tree[q].l, l, mid, k);
        } else {
                return query(tree[p].r, tree[q].r, mid + 1, r, k - cnt);
        }
}
int main() {
        cin >> n >> m;
```

```
    tot = 0;
    for(int i = 1; i <= n; i++ ) {
        cin >> a[i];
        v.push_back(a[i]);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    root[0] = build(1, n);
    for(int i = 1; i <= n; i++ ) {
        root[i] = insert(root[i-1], 1, n, getid(a[i]), 1);
    }
    for(int i = 1; i <= m; i++ ) {
        int l, r, k;
        cin >> l >> r >> k;
        int pos = query(root[r], root[l-1], 1, n, k);
        cout << v[pos - 1] << endl;
    }
    return 0;
}
```

单点回溯主席树



输入的第一行包含两个正整数 N, M， 分别表示数组的长度和操作的个数。

第二行包含 N 个整数，依次为初始状态下数组各位的值（依次为 a_i， $1 \leq i \leq N$）。

接下来 M 行每行包含3或4个整数，代表两种操作之一（i 为基于的历史版本号）：

1. 对于操作1，格式为 $v_i\ 1\ loc_i\ value_i$，即为在版本 v_i 的基础上，将 a_{loc_i} 修改为 $value_i$
2. 对于操作2，格式为 $v_i\ 2\ loc_i$，即访问版本 v_i 中的 a_{loc_i} 的值

输出格式：

输出包含若干行，依次为每个操作2的结果。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 7;
int n, m;
struct Node {
    int l, r, sum;
} tree[maxn * 40];
int root[maxn], tot, arr[maxn];
inline void init() {
    tot = 0;
```

```
}
void build(int &rt, int l, int r) {
        rt = ++tot;
        if(l == r) {
                tree[rt].sum = arr[l];
                return ;
        }
        int mid = (l + r) >> 1;
        build(tree[rt].l, l, mid);
        build(tree[rt].r, mid + 1, r);
}
void Insert(int &rt, int pre, int l, int r, int pos, int val)
{
        rt = ++tot, tree[rt] = tree[pre];
        if(l == r) {
                tree[rt].sum = val;
                return ;
        }
        int mid = (l + r) >> 1;
        if(pos <= mid) {
                Insert(tree[rt].l, tree[pre].l, l, mid, pos, val);
        } else {
                Insert(tree[rt].r, tree[rt].r, mid + 1, r, pos, val);
        }
}
int Query(int rt, int l, int r, int pos) {
        if(l == r) {
                return tree[rt].sum;
        }
        int mid = (l + r) >> 1;
        if(pos <= mid) {
                return Query(tree[rt].l, l, mid, pos);
        } else {
                return Query(tree[rt].r, mid + 1, r, pos);
        }
}
int main() {
        while(~scanf("%d %d", &n, &m)) {
                init();
                for(int i = 1; i <= n; i ++ ) {
                        scanf("%d", &arr[i]);
                }
                build(root[0], 1, n);
                for(int i = 1; i <= m; i ++ ) {
```

```
			int vi, opt, loc, val;
			scanf("%d %d", &vi, &opt);
			if(opt == 1) {
				scanf("%d %d", &loc, &val);
				Insert(root[i], root[vi], 1, n, loc, val);
			} else {
				scanf("%d", &loc);
				printf("%d\n", Query(root[vi], 1, n, loc));
				root[i] = root[vi];
			}
		}
	}
	return 0;
}
//*******************
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
const int N = 1000010;
const int INF = 0x7FFFFFFF;
struct Node {
	int l, r, sum;
} tree[N * 20];
int n, m, tot, root[N], a[N];
vector<int>v;
inline int getid(int x) {
	return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
}
int build(int l, int r) {
	int p = ++tot;
	tree[p].sum = 0;
	if(l == r) {
		tree[p].sum = a[l];
		return p;
	}
	int mid = (l + r) >> 1;
	tree[p].l = build(l, mid);
	tree[p].r = build(mid + 1, r);
	return p;
}
int insert(int now, int l, int r, int pos, int val) {
	int p = ++tot;
	tree[p] = tree[now];
	if(l == r) {
		tree[p].sum = val;
		return p;
	}
	int mid = (l + r) >> 1;
	if(pos <= mid) {
		tree[p].l = insert(tree[now].l, l, mid, pos, val);
	} else {
		tree[p].r = insert(tree[now].r, mid + 1, r, pos, val);
	}
	tree[p].sum = tree[ tree[p].l ].sum + tree[ tree[p].r ].sum;
	return p;
}
int query(int rt, int l, int r, int k) {
	if(l == r) {
		return tree[rt].sum;
	}
	int mid = (l + r) >> 1;
	if(k <= mid) {
		return query(tree[rt].l, l, mid, k);
	} else {
		return query(tree[rt].r, mid + 1, r, k);
	}
}
int main() {
	cin >> n >> m;
	tot = 0;
	for(int i = 1; i <= n; i++ ) {
		cin >> a[i];
		v.push_back(a[i]);
	}
	sort(v.begin(), v.end());
	v.erase(unique(v.begin(), v.end()), v.end());
	root[0] = build(1, n);
	for(int i = 1; i <= m; i++ ) {
		int vi, opt, loc, val;
		scanf("%d %d", &vi, &opt);
		if(opt == 1) {
			scanf("%d %d", &loc, &val);
			root[i] = insert(root[vi], 1, n, loc, val);
		} else {
			scanf("%d", &loc);
```

```
            printf("%d\n", query(root[vi], 1, n, loc));
            root[i] = root[vi];
        }
    }
    return 0;
}
```

主席树维护区间与 x 最接近的数字

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 7;
const int INF = 0x7FFFFFFF;
int n, m, cnt, root[MAXN], a[MAXN], x, y, k;
bool ok;
struct Node {
    int sum, l, r;
} T[MAXN * 20];
vector<int> v;
inline int get_id(int x) {
    return lower_bound(v.begin(),v.end(),x) - v.begin() + 1;
}
void update(int l, int r, int &x, int y, int pos) {
    T[ ++cnt ] = T[y], T[cnt].sum ++, x = cnt;
    if(l == r) {
        return ;
    }
    int mid = (l + r) >> 1;
    if(pos <= mid) {
        update(l, mid, T[x].l, T[y].l, pos);
    } else {
        update(mid + 1, r, T[x].r, T[y].r, pos);
    }
}
int query(int l, int r, int x, int y, int k) {
    if(l == r) {
        return l;
    }
    int mid = (l + r) >> 1;
    int sum = T[ T[y].l ].sum - T[ T[x].l ].sum;
    if(sum >= k) {
        return query(l, mid, T[x].l, T[y].l, k);
    } else {
        return query(mid + 1, r, T[x].r, T[y].r, k - sum);
        }

}
int cal(int l, int r, int x, int y, int pos) {
        if(l == r) {
                return T[y].sum - T[x].sum;
        }
        int mid = (l + r) >> 1;
        if(pos <= mid) {
                return cal(l, mid, T[x].l, T[y].l, pos);
        } else {
                return    T[T[y].l].sum - T[T[x].l].sum + cal(mid + 1, r, T[x].r, T[y].r, pos);
        }
}
int main() {
        scanf("%d %d", &n, &m);
        for(int i = 1; i <= n; i ++ ) {
                scanf("%d", &a[i]);
                v.push_back(a[i]);
        }
        sort(v.begin(), v.end());
        v.erase(unique(v.begin(),v.end()), v.end());
        for(int i = 1; i <= n; i ++ ) {
                update(1, n, root[i], root[i - 1], get_id(a[i]));
        }
        for(int i = 1; i <= m; i ++ ) {
                scanf("%d %d %d", &x, &y, &k);
                int kth = cal(1, n, root[x - 1], root[y], get_id(k + 1) - 1);
                int ans = INF;
                if(kth == 0) {
                        ans = abs(v[query(1, n, root[x - 1], root[y], 1) - 1] - k);
                } else if(kth == y - x + 1) {
                        ans = abs(v[query(1, n, root[x - 1], root[y], y - x + 1) - 1] - k);
                } else {
                        ans = min(ans, abs(k - v[query(1, n, root[x - 1], root[y], kth) - 1]));
                        ans = min(ans, abs(k - v[query(1, n, root[x - 1], root[y], kth + 1) - 1]));
                }
                printf("%d\n", ans);
        }
        return 0;
}
```

并查集

带权并查集

```
const int maxn = 100005;
int father[maxn],dist[maxn],size[maxn];
int n,m,x,y;
void init() {
	for(int i = 1; i <= n; i++)	{
		father[i] = i, dist[i] = 0, size[i] = 1;
	}
}
int get(int x) {
	if(father[x] == x) {
		return x;
	}
	int y = father[x];
	father[x] = get(y);
	dist[x] += dist[y];	// x 到根结点的距离等于 x 到之前父亲结点距离加上之前父亲结点
到根结点的距离
	return father[x];
}
void join(int a, int b) {
	a = get(a);
	b = get(b);
	if(a != b) {	// 判断两个元素是否属于同一集合
		father[b] = a;
		dist[a] = size[b];
		size[b] += size[a];
	}
}
```

按秩合并并查集

```
const int maxn = 1000000+7;
int n, m;
int father[maxn], rnk[maxn]; //节点的父亲、秩
void makeSet() { //初始化并查集
	for (int i = 0; i <= n + 1; i++)
		father[i] = i, rnk[i] = 0;
}
int findSet(int x) { //查找
```

```
	if (x != father[x])
		father[x] = findSet(father[x]);
	return father[x];
}
void unionSet(int x, int y) { //合并
	x = findSet(x), y = findSet(y);
	if (x == y)
		return;
	if (rnk[x] > rnk[y])
		father[y] = x;
	else
		father[x]	= y, rnk[y] += rnk[x] == rnk[y];
}
```

可持久化并查集

```
/**
 * BZOJ 3673  可持久化线段树维护启发式合并的可持久化并查集
n 个集合  m 个操作
操作：
1 a b  合并 a,b 所在集合
2 k  回到第 k 次操作之后的状态(查询算作操作)
3 a b  询问 a,b 是否属于同一集合，是则输出 1 否则输出 0
0<n,m<=2*10^4
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int INF = 0x3F3F3F3F;
const int INT_MAX = 0x7FFFFFFF;
const LL LLINF = 0x3F3F3F3F3F3F3F3F;
const double PI = acos(-1);
const int MAXN = 2e5 + 10;
const int MAXM = 1e6 + 10;
struct Node {
	int l, r, faz, rank;
} tree[MAXN * 20];
int n, m, root[MAXN], tot;
int build(int l, int r) {
	int now = ++tot;
	if(l == r) {
		tree[now].faz = l;
		tree[now].rank = 0;
```

```
        return now;
    }
    int mid = (l + r) >> 1;
    tree[now].l = build(l, mid);
    tree[now].r = build(mid + 1, r);
    return now;
}
int query(int rt, int l, int r, int x) {
    if(l == r) {
        return rt;
    }
    int mid = (l + r) >> 1;
    if(x <= mid) {
        return query(tree[rt].l, l, mid, x);
    } else {
        return query(tree[rt].r, mid + 1, r, x);
    }
}
int find_root(int rt, int x) {
    int y = query(rt, 1, n, x);
    if(x == tree[y].faz) {
        return y;
    }
    return find_root(rt, tree[y].faz);
}
int update(int rt, int l, int r, int pos, int val) {
    int now = ++tot;
    tree[now] = tree[rt];
    if(l == r) {
        tree[now].faz = val;
        tree[now].rank = tree[rt].rank;
        return now;
    }
    int mid = (l + r) >> 1;
    if(pos <= mid) {
        tree[now].l = update(tree[now].l, l, mid, pos, val);
    } else {
        tree[now].r = update(tree[now].r, mid + 1, r, pos, val);
    }
    return now;
}
void add(int now, int l, int r, int pos) {
    if(l == r) {
        tree[now].rank++;
```

```
            return ;
        }
        int mid = (l + r) >> 1;
        if(pos <= mid) {
            add(tree[now].l, l, mid, pos);
        } else {
            add(tree[now].r, mid + 1, r, pos);
        }
    }
    int main() {
        while(~scanf("%d %d", &n, &m)) {
            tot = 0;
            root[0] = build(1, n);
            for(int i = 1; i <= m; i++ ) {
                int opt;
                scanf("%d", &opt);
                if(opt == 1) {
                    int a, b;
                    scanf("%d %d", &a, &b);
                    root[i] = root[i - 1];
                    int x = find_root(root[i], a);
                    int y = find_root(root[i], b);
                    if(tree[x].faz == tree[y].faz) {
                        continue;
                    }
                    if(tree[x].rank > tree[y].rank) {
                        swap(x, y);
                    }
                    root[i] = update(root[i - 1], 1, n, tree[x].faz, tree[y].faz);
                    if(tree[x].rank == tree[y].rank) {
                        add(root[i], 1, n, tree[y].faz);
                    }
                } else if(opt == 2) {
                    int k;
                    scanf("%d", &k);
                    root[i] = root[k];
                } else if(opt == 3) {
                    int a, b;
                    scanf("%d %d", &a, &b);
                    root[i] = root[i - 1];
                    a = find_root(root[i], a);
                    b = find_root(root[i], b);
                    if(tree[a].faz == tree[b].faz) {
                        puts("1");
                        } else {
                                puts("0");
                        }
                }
        }
    }
    return 0;
}
```

树状数组

单点跟新区间求和

```
int lowbit(int x) {
    return x & -x;
}
void update(int pos,int val) {
    for(int i = pos; i <= n; i += lowbit(i) ) { //注意上界是否是 n
        c[i] += val;
    }
}
int query(int pos) { //注意 ans 视情况 long long
    int ans = 0;
    for(int i = pos; i > 0; i -= lowbit(i)) {
        ans += c[i];
    }
    return ans;
}
```

区间更新单点求和

板子如上，调用的时候 add(b,1); add(a-1,-1);

二维树状数组

```
//  对矩阵更新，查询当点
//  或者单点更新，查询矩阵
//对一个矩阵做加法，然后询问某点的奇偶，POJ 2155
#include <bits/stdc++.h>
#define MAX 1050
```

```
using namespace std;
int c[MAX][MAX];
int n,m;
int Lowbit(int x) {
        return x & (-x);
}
void Updata(int x,int y,int d) {
        int i,k;
        for(i=x; i<=n; i+=Lowbit(i)) {
                for(k=y; k<=n; k+=Lowbit(k)) {
                        c[i][k]+=d;
                }
        }
}
int Get(int x,int y) {
        int i,k,sum = 0;
        for(i=x; i>0; i-=Lowbit(i)) {
                for(k=y; k>0; k-=Lowbit(k)) {
                        sum += c[i][k];
                }
        }
        return sum;
}
int main() {
        int t;
        char op[5];
        int x,y,xx,yy;
        int first = 1;
        while(~scanf("%d",&t)) {
                while(t--) {
                        scanf("%d %d",&n,&m);
                        memset(c,0,sizeof(c));
                        while(m--) {
                                scanf("%s",op);
                                if(op[0] == 'C') {
                                        scanf("%d %d %d %d",&x,&y,&xx,&yy);
                                        Updata(x,y,1);
                                        Updata(x,yy+1,-1);
                                        Updata(xx+1,y,-1);
                                        Updata(xx+1,yy+1,1);
                                } else {
                                        scanf("%d %d",&x,&y);
                                        printf("%d\n",Get(x,y)%2);
                                }
```

```
            }
            if(t) {
                puts("");
            }
        }
    }
    return 0;
}
```

倍增

ST 表求极值

```
//求区间最大值
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 7;
int n, m, arr[maxn], st[maxn][15];
void RMQ() {
    for(int i = 1; i <= n; i ++ ) {
        st[i][0] = arr[i];
    }
    for(int j = 1; (1 << j) <= n; j ++ ) {
        for(int i = 1; i + (1 << j) - 1 <= n; i ++ ) {
            st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
        }
    }
}
int Query(int l, int r) {
    int k = 0;
    while((1 << (k + 1)) <= r - l + 1) {
        k ++;
    }
    return max(st[l][k], st[r - (1 << k) + 1][k]);
}
int main() {
    while(~scanf("%d %d", &n, &m)) {
        for(int i = 1; i <= n; i ++ ) {
            scanf("%d", &arr[i]);
        }
        RMQ();
        while(m--) {
            int l, r;
```

```
                scanf("%d %d", &l, &r);
                printf("%d\n", Query(l, r));
            }
        }
        return 0;
}
```

树链剖分

重链剖分-点权

```
///树剖  点权  路径更新，路径询问
///HDU 3966
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 50010;
typedef long long LL;
struct Edge {
        int to, w, next;
} edge[MAXN * 2];
int first[MAXN], sign, tot;
int dep[MAXN], siz[MAXN], faz[MAXN], id[MAXN], son[MAXN], top[MAXN], tid[MAXN];
int n, m, q;
long long a[MAXN];
struct SegmentTree {
        struct Node {
                int l, r;
                LL sum, Lazy;
        } tree[MAXN * 4];
        inline void push_up(int rt) {
                tree[rt].sum = tree[rt << 1].sum + tree[rt << 1 | 1].sum;
        }
        inline void push_down(int rt) {
                if(tree[rt].Lazy) {
                        tree[rt << 1].Lazy += tree[rt].Lazy;
                        tree[rt << 1 | 1].Lazy += tree[rt].Lazy;
                        tree[rt << 1].sum += (tree[rt << 1].r - tree[rt << 1].l + 1) * tree[rt].Lazy;
                        tree[rt << 1 | 1].sum += (tree[rt << 1 | 1].r - tree[rt << 1 | 1].l + 1) * tree[rt].Lazy;
                        tree[rt].Lazy = 0;
                }
        }
        void build(int rt, int l, int r) {
```

```
	tree[rt].l = l;
	tree[rt].r = r;
	tree[rt].sum = 0;
	tree[rt].Lazy = 0;
	if(l == r) {
		tree[rt].sum = a[ tid[l] ];
		return ;
	}
	int mid = (l + r) >> 1;
	build(rt << 1, l, mid);
	build(rt << 1 | 1, mid + 1, r);
	push_up(rt);
}
void update(int rt, int l, int r, LL val) {
	if(l <= tree[rt].l && tree[rt].r <= r) {
		tree[rt].sum += (tree[rt].r - tree[rt].l + 1) * val;
		tree[rt].Lazy += val;
		return ;
	}
	push_down(rt);
	int mid = (tree[rt].l + tree[rt].r) >> 1;
	if(r <= mid) {
		update(rt << 1, l, r, val);
	} else if(l > mid) {
		update(rt << 1 | 1, l, r, val);
	} else {
		update(rt << 1, l, mid, val);
		update(rt << 1 | 1, mid + 1, r, val);
	}
	push_up(rt);
}
LL query(int rt, int l, int r) {
	if(l <= tree[rt].l && tree[rt].r <= r) {
		return tree[rt].sum;
	}
	push_down(rt);
	int mid = (tree[rt].l + tree[rt].r) >> 1;
	if(r <= mid) {
		return query(rt << 1, l, r);
	} else if(l > mid) {
		return query(rt << 1 | 1, l, r);
	} else {
		return query(rt << 1, l, mid) + query(rt << 1 | 1, mid + 1, r);
	}
```

```
        }
} Seg;
void init() {
        sign = tot = 0;
        memset(first, -1, sizeof(first));
        memset(dep, 0, sizeof(dep));
        memset(siz, 0, sizeof(siz));
        memset(faz, 0, sizeof(faz));
        memset(id, 0, sizeof(id));
        memset(son, 0, sizeof(son));
        memset(top, 0, sizeof(top));
}
void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign++;
}
void dfs1(int now, int father, int depth) {
        siz[now] = 1;
        faz[now] = father;
        dep[now] = depth;
        son[now] = 0;
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to != father) {
                        dfs1(to, now, depth + 1);
                        siz[now] += siz[to];
                        if(son[now] == 0 || siz[ son[now] ] < siz[to]) {
                                son[now] = to;
                        }
                }
        }
}
void dfs2(int now, int topf) {
        top[now] = topf;
        id[now] = ++tot;
        tid[id[now]] = now;
        if(son[now]) {
                dfs2(son[now], topf);
        }
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to == faz[now] || to == son[now]) {
```

```
                continue;
            }
            dfs2(to, to);
        }
}
void cutting(int u, int v, int val) {
        int fu = top[u], fv = top[v];
        while(fu != fv) {
                if(dep[fu] < dep[fv]) {
                        swap(fu, fv);
                        swap(u, v);
                }
                Seg.update(1, id[fu], id[u], val);
                u = faz[fu];
                fu = top[u];
        }
        if(dep[u] > dep[v]) {
                swap(u,v);
        }
        Seg.update(1, id[u], id[v], val);
}
long long query(int u, int v) {
        long long sum = 0;
        int fu = top[u], fv = top[v];
        while(fu != fv) {
                if(dep[fu] < dep[fv]) {
                        swap(fu, fv);
                        swap(u, v);
                }
                sum = sum + Seg.query(1, id[fu], id[u]);
                u = faz[fu];
                fu = top[u];
        }
        if(dep[u] > dep[v]) {
                swap(u, v);
        }
        return sum = sum + Seg.query(1, id[u], id[v]);
}
int main() {
        while(~scanf("%d %d %d", &n, &m, &q)) {
                init();
                for(int i = 1; i <= n; i++ ) {
                        scanf("%d", &a[i]);
                }
                for(int i = 1; i <= m; i++ ) {
                        int u, v;
                        scanf("%d %d", &u, &v);
                        add_edge(u, v, 1);
                        add_edge(v, u, 1);
                }
                tot = 0;
                dfs1(1, 0, 0);
                dfs2(1, 1);
                Seg.build(1, 1, n);
                char opt[5];
                int x, y,z;
                for(int i = 1; i <= q; i++ ) {
                        scanf("%s", opt);
                        if(opt[0] == 'I') {
                            scanf("%d %d %d", &x, &y, &z);
                            cutting(x, y, z);
                            continue;
                        }
                        if(opt[0] == 'D') {
                            scanf("%d %d %d", &x, &y, &z);
                            cutting(x, y, -z);
                            continue;
                        }
                        if(opt[0] == 'Q') {
                            scanf("%d", &x);
                            printf("%I64d\n", query(x, x));
                            continue;
                        }
                }
        }
        return 0;
}
```

重链剖分-边权

```
///POJ 2763
///边权修改，边权求和
///题意大概是保姆在 x，某个点婴儿呼叫，保姆过去求代价。
///可以修改某个边权。把边权记录在更深的结点上
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
```

```
using namespace std;
const int maxn = 1e5 + 7;

int dep[maxn], siz[maxn], faz[maxn], id[maxn], son[maxn], val[maxn], top[maxn];

struct Edge {
        int to, w, next;
} edge[maxn * 2];

int first[maxn], sign, n, m, s, tot;

struct Node {
        int from, to, cost;
} input[maxn];

struct TreeNode {
        int l, r, val;
} tree[maxn << 2];

inline void init() {
        for(int i = 0; i <= n; i ++ ) {
                first[i] = -1;
        }
        sign = 0;
}
inline void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign ++;
}
void dfs1(int now, int father, int depth) {
        siz[now] = 1;
        faz[now] = father;
        dep[now] = depth;
        son[now] = 0;
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to != father) {
                        dfs1(to, now, depth + 1);
                        siz[now] += siz[to];
                        if(siz[ son[now] ] < siz[to]) {
                                son[now] = to;
                }
            }
        }
}
void dfs2(int now, int topf) {
        top[now] = topf;
        id[now] = ++tot;
        if(son[now]) {
                dfs2(son[now], topf);
        }
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to == faz[now] || to == son[now]) {
                        continue;
                }
                dfs2(to, to);
        }
}
void push_up(int rt) {
        tree[rt].val = tree[rt<<1].val + tree[rt<<1|1].val;
}
void build(int rt, int l, int r) {
        tree[rt].l = l, tree[rt].r = r;
        if(l == r) {
                tree[rt].val = val[l];
                return ;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l , mid);
        build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
}
void update(int rt, int pos, int val) {
        if(tree[rt].l == tree[rt].r) {
                tree[rt].val = val;
                return ;
        }
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(pos <= mid) {
                update(rt << 1, pos, val);
        } else {
                update(rt << 1 | 1, pos, val);
        }
        push_up(rt);
```

```
}
int query(int rt, int l, int r) {
        if(l <= tree[rt].l && tree[rt].r <= r) {
                return tree[rt].val;
        }
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(r <= mid) {
                return query(rt << 1, l, r);
        } else if(l > mid) {
                return query(rt << 1 | 1, l, r);
        } else {
                return query(rt << 1, l, mid) + query(rt << 1 | 1, mid + 1, r);
        }
}
int cutting(int x, int y) {
        int sum = 0;
        while(top[x] != top[y]) {
                if(dep[top[x]] < dep[top[y]]) {
                        swap(x,y);
                }
                sum += query(1, id[top[x]], id[x]);
                x = faz[top[x]];
        }
        if(dep[x] > dep[y]) {
                swap(x,y);
        }
        if(x != y) {
                sum += query(1, id[son[x]], id[y]);
        }
        return sum;
}
int main()
{
        while(~scanf("%d %d %d", &n, &m, &s)) {
                init();
                for(int i = 1; i <= n - 1; i ++ ) {
                        scanf("%d %d %d", &input[i].from, &input[i].to, &input[i].cost);
                        add_edge(input[i].from, input[i].to, input[i].cost);
                        add_edge(input[i].to, input[i].from, input[i].cost);
                }
                tot = 0;
                dfs1(1, 0, 1);
                dfs2(1, 1);
                for(int i = 1; i <= n - 1; i ++ ) {
```

```
            if(dep[ input[i].from ] < dep[ input[i].to ]) {
                swap(input[i].from, input[i].to);
            }
            val[ id[ input[i].from ] ] = input[i].cost;
        }
        build(1, 1, n);
        while(m -- ) {
            int opt, x, y;
            scanf("%d", &opt);
            if(opt == 0) {
                scanf("%d", &x);
                printf("%d\n", cutting(s, x));
                s = x;
            } else {
                scanf("%d %d", &x, &y);
                update(1, id[ input[x].from ], y);
            }
        }
    }
    return 0;
}
```

重链剖分-边权，边权取反

```
///POJ 3237  树剖边权。
///取反某条路径的权重，修改某条边的权重，询问某条路径的最大值
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 7;
const int INF = 0x7FFFFFF;

int dep[maxn], siz[maxn], faz[maxn], id[maxn], son[maxn], val[maxn], top[maxn];

struct Edge {
    int to, w, next;
} edge[maxn * 2];

int first[maxn], sign, n, m, s, tot;

struct Node {
    int from, to, cost;
```

```
} input[maxn];

struct TreeNode {
        int l, r, mx, mi, lazy;
} tree[maxn << 2];

inline void init() {
        memset(first, -1, sizeof(first));
        sign = 0;
}

inline void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign ++;
}

void dfs1(int now, int father, int depth) {
        siz[now] = 1;
        faz[now] = father;
        dep[now] = depth;
        son[now] = 0;
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to != father) {
                        dfs1(to, now, depth + 1);
                        siz[now] += siz[to];
                        if(siz[ son[now] ] < siz[to]) {
                                son[now] = to;
                        }
                }
        }
}

void dfs2(int now, int topf) {
        top[now] = topf;
        id[now] = ++tot;
        if(son[now]) {
                dfs2(son[now], topf);
        }
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to == faz[now] || to == son[now]) {
```

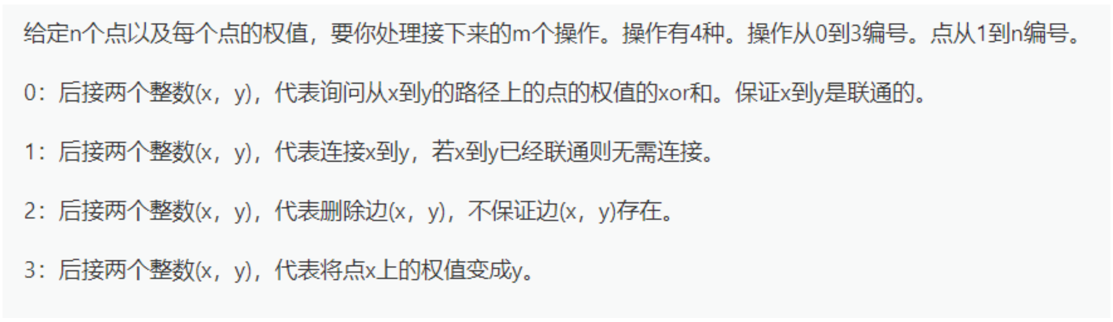
```
                continue;
            }
            dfs2(to, to);
        }
}

void push_up(int rt) {
        tree[rt].mx = max(tree[rt << 1].mx, tree[rt << 1 | 1].mx);
        tree[rt].mi = min(tree[rt << 1].mi, tree[rt << 1 | 1].mi);
}

void push_down(int rt) {
        if(tree[rt].lazy) {
            tree[rt].lazy ^= 1;
            tree[rt << 1].lazy ^= 1;
            tree[rt << 1 | 1].lazy ^= 1;
            swap(tree[rt << 1].mx, tree[rt << 1].mi);
            tree[rt << 1].mx *= -1;
            tree[rt << 1].mi *= -1;
            swap(tree[rt << 1 | 1].mx, tree[rt << 1 | 1].mi);
            tree[rt << 1 | 1].mx *= -1;
            tree[rt << 1 | 1].mi *= -1;
        }
}

void build(int rt, int l, int r) {
        tree[rt].l = l, tree[rt].r = r;
        tree[rt].lazy = 0;
        if(l == r) {
            tree[rt].mx = tree[rt].mi = val[l];
            return ;
        }
        int mid = (l + r) >> 1;
        build(rt << 1, l, mid);
        build(rt << 1 | 1, mid + 1, r);
        push_up(rt);
}

void update(int rt, int l, int r) { ///区间取反
        if(l <= tree[rt].l && tree[rt].r <= r) {
            tree[rt].lazy ^= 1;
            swap(tree[rt].mx, tree[rt].mi);
            tree[rt].mx *= -1;
            tree[rt].mi *= -1;
            return ;
        }
        push_down(rt);
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(r <= mid) {
            update(rt << 1, l, r);
        } else if(l > mid) {
            update(rt << 1 | 1, l, r);
        } else {
            update(rt << 1, l, mid);
            update(rt << 1 | 1, mid + 1, r);
        }
        push_up(rt);
}

void updatePos(int rt, int pos, int val) { ///单点修改
        if(tree[rt].l == tree[rt].r) {
            tree[rt].mx = tree[rt].mi = val;
            return ;
        }
        push_down(rt);
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(pos <= mid) {
            updatePos(rt << 1, pos, val);
        } else {
            updatePos(rt << 1 | 1, pos, val);
        }
        push_up(rt);
}

int query(int rt, int l, int r) {
        if(l <= tree[rt].l && tree[rt].r <= r) {
            return tree[rt].mx;
        }
        push_down(rt);
        int mid = (tree[rt].l + tree[rt].r) >> 1;
        if(r <= mid) {
            return query(rt << 1, l, r);
        } else if(l > mid) {
            return query(rt << 1 | 1, l, r);
        } else {
            return max(query(rt << 1, l, mid), query(rt << 1 | 1, mid + 1, r));
        }
}
```

```
void treeUpdate(int x, int y) {
        while(top[x] != top[y]) {
                if(dep[top[x]] < dep[top[y]]) {
                        swap(x,y);
                }
                update(1, id[top[x]], id[x]);
                x = faz[top[x]];
        }
        if(dep[x] > dep[y]) {
                swap(x,y);
        }
        if(x != y) {
                update(1, id[son[x]], id[y]);
        }
}

int treeQuery(int x, int y) {
        int ans = -INF;
        while(top[x] != top[y]) {
                if(dep[top[x]] < dep[top[y]]) {
                        swap(x,y);
                }
                ans = max(ans, query(1, id[top[x]], id[x]));
                x = faz[top[x]];
        }
        if(dep[x] > dep[y]) {
                swap(x,y);
        }
        if(x != y) {
                ans = max(ans, query(1, id[son[x]], id[y]));
        }
        return ans;
}

int cutting(int x, int y) {
        int sum = 0;
        while(top[x] != top[y]) {
                if(dep[top[x]] < dep[top[y]]) {
                        swap(x,y);
                }
                sum += query(1, id[top[x]], id[x]);
                x = faz[top[x]];
        }
        if(dep[x] > dep[y]) {
                swap(x,y);
        }
        if(x != y) {
                sum += query(1, id[son[x]], id[y]);
        }
        return sum;
}

int main() {
        int T, x, y;
        char opt[10];
        scanf("%d", &T);
        while(T--) {
                scanf("%d", &n);
                init();
                for(int i = 1; i <= n - 1; i++ ) {
                        scanf("%d %d %d", &input[i].from, &input[i].to, &input[i].cost);
                        add_edge(input[i].from, input[i].to, input[i].cost);
                        add_edge(input[i].to, input[i].from, input[i].cost);
                }
                tot = 0;
                dfs1(1, 0, 1);
                dfs2(1, 1);
                for(int i = 1; i <= n - 1; i++ ) {
                        if(dep[ input[i].from ] < dep[ input[i].to ]) {
                                swap(input[i].from, input[i].to);
                        }
                        val[ id[ input[i].from ] ] = input[i].cost;
                }
                build(1, 1, n);
                while(~scanf("%s", opt) && strcmp(opt, "DONE")) {
                        if(opt[0] == 'C') {
                                scanf("%d %d", &x, &y);
                                if(dep[ input[x].from ] < dep[ input[x].to ]) {
                                        swap(input[x].from, input[x].to);
                                }
                                updatePos(1, id[ input[x].from ], y);
                        }
                        if(opt[0] == 'N') {
                                scanf("%d %d", &x, &y);
                                treeUpdate(x, y);
                        }
                        if(opt[0] == 'Q') {
```

```
                        scanf("%d %d", &x, &y);
                        printf("%d\n", treeQuery(x, y));
                }
            }
        }
        return 0;
}
```

动态树问题

LCT 维护路径亦或合


给定n个点以及每个点的权值，要你处理接下来的m个操作。操作有4种。操作从0到3编号。点从1到n编号。

0：后接两个整数(x，y)，代表询问从x到y的路径上的点的权值的xor和。保证x到y是联通的。

1：后接两个整数(x，y)，代表连接x到y，若x到y已经联通则无需连接。

2：后接两个整数(x，y)，代表删除边(x，y)，不保证边(x，y)存在。

3：后接两个整数(x，y)，代表将点x上的权值变成y。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 3e5 + 10;
typedef long long LL;
#define fa(x) T[x].f
#define ls(x) T[x].ch[0]
#define rs(x) T[x].ch[1]
#define RG register int
int v[MAXN];
struct node {
        int f, ch[2], s;
        bool r;
} T[MAXN];
inline int read() {
        char c = getchar();
        int x = 0,f = 1;
        while(c < '0' || c > '9') {
                if(c == '-')
                        f = -1;
                c = getchar();
        }
        while(c >= '0' && c <= '9') {
```

```
        x = x * 10 + c - '0', c = getchar();
    }
    return x * f;
}
inline int ident(RG x) {
    return T[fa(x)].ch[0] == x ? 0 : 1;
}
inline void connect(RG x, RG fa, RG how) {
    T[x].f = fa;
    T[fa].ch[how] = x;
}
inline bool isRoot(RG x) { //若为 splay 中的根则返回 1  否则返回 0
    return ls( fa(x) ) != x && rs( fa(x) ) != x;
}
inline void push_up(RG x) {
    T[x].s = T[ls(x)].s ^ T[rs(x)].s ^ v[x]; //维护路径上的异或和
}
inline void push_down(RG x) {
    if(T[x].r) {
        swap(ls(x),rs(x));
        T[ls(x)].r ^= 1;
        T[rs(x)].r ^= 1;
        T[x].r = 0;
    }
}
inline void rotate(RG x) {
    int Y = T[x].f, R = T[Y].f, Yson = ident(x), Rson = ident(Y);
    int B = T[x].ch[Yson ^ 1];
    T[x].f = R;
    if(!isRoot(Y)) {
        connect(x, R, Rson);
    }
    connect(B, Y, Yson);
    connect(Y, x, Yson ^ 1);
    push_up(Y);
    push_up(x);
}
int st[MAXN];
inline void splay(RG x) {
    int y = x, top = 0;
    st[++top] = y;
    while(!isRoot(y)) {
        st[++top] = y = fa(y);
    }
```

```
        while(top) {
            push_down(st[top--]);
        }
        for(int y = fa(x); !isRoot(x); rotate(x), y = fa(x)) {
            if(!isRoot(y)) {
                rotate( ident(x) == ident(y) ? x : y );
            }
        }
}
inline void access(RG x) {
        for(int y = 0; x; x = fa(y = x)) {
            splay(x), rs(x) = y, push_up(x);
        }
}
inline void make_root(RG x) {
        access(x);
        splay(x);
        T[x].r ^= 1;
        push_down(x);
}
inline int find_root(RG x) {
        access(x);
        splay(x);
        push_down(x);
        while(ls(x)) {
            push_down(x = ls(x));
        }
        return x;
}
inline void split(RG x, RG y) {
        make_root(x);
        access(y);
        splay(y);
}
inline void link(RG x, RG y) {
        make_root(x);
        if(find_root(y) != x) {
            fa(x) = y;
        }
}
inline void cut(RG x, RG y) {
        make_root(x);
        if(find_root(y) == x && fa(x) == y && ls(y) == x && !rs(x)) {
            fa(x) = T[y].ch[0] = 0;
```

```
		push_up(y);
	}
}
int tab[MAXN][2], tot;
int main() {
	register int n = read(), m = read();
	register int i, u, v;
	for(i = 1; i <= n - 1; i++ ) {
		u = read(), v = read();
		link(u, v);
	}
	char opt[5];
	tot = 0;
	for(i = 1; i <= m; i++ ) {
		scanf("%s", opt);
		if(opt[0] == 'U') {
			u = read();
			link(tab[u][0], tab[u][1]);
		}
		if(opt[0] == 'C') {
			u = read(), v = read();
			tot++;
			tab[tot][0] = u, tab[tot][1] = v;
			cut(u, v);
		}
		if(opt[0] == 'Q') {
			u = read(), v = read();
			printf("%s\n", find_root(u) == find_root(v) ? "Yes" : "No");
		}
	}
	return 0;
}
```

莫队区间种类树

```
//这个莫队的 cmp 改成下面 dfs 序的莫队的 cmp 会快很多
//早期代码未优化!!!!  虽然也是对的
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6+7;
const int maxm = 2e5+7;
int n,m,arr[maxn],mark[maxn];
int ans,l,r,belong[maxn],output[maxn];
struct Node {
	int l,r,id,ans;
} qus[maxm];
bool cmp(Node a,Node b) {
	if(belong[a.l] == belong[b.l]) {
		return belong[a.r] < belong[b.r];
	}
	return belong[a.l] < belong[b.l];
}
int add(int val) {
	return mark[val] ++ == 0;
}
int cut(int val) {
	return -- mark[val] == 0;
}
int main() {
	while(~scanf("%d %d",&n,&m)) {
		int sz = sqrt(n);
		memset(mark,0,sizeof(mark));
		for(int i = 1; i <= n; i ++ ) {
			scanf("%d",&arr[i]);
			belong[i] = i / sz;
		}
		for(int i = 1; i <= m; i ++ ) {
			scanf("%d %d",&qus[i].l,&qus[i].r);
			qus[i].id = i;
		}
		ans = l = r = 0;
		sort(qus+1,qus+1+m,cmp);
		for(int i = 1; i <= m; i ++ ) {
			while(qus[i].l < l) {
				ans += add(arr[--l]);
			}
			while(qus[i].l > l) {
				ans -= cut(arr[l++]);
			}
			while(qus[i].r < r) {
				ans -= cut(arr[r--]);
			}
			while(qus[i].r > r) {
				ans += add(arr[++r]);
			}
			output[qus[i].id] = ans;
		}
		for(int i = 1; i <= m; i ++ ) {
			printf("%d\n",output[i]);
		}
	}

	return 0;
}
```

莫队+dfs 序列

```
//求子树种类数
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 10;
const int M = N * 4;
int n, m, first[N], sign, arr[N];
int lef[N], rig[N], indexs, x[N];
int sz;
int mark[N];
vector<int>v;
int getid(int x) {
	return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
}
int ans, l, r, belong[N], output[N], b[N];
struct Node {
	int l, r, id, ans;
} qus[N];
bool cmp(const Node &a, const Node &b) {
	if(belong[a.l] == belong[b.l]) {
		if(belong[a.l] & 1) {
			return a.r > b.r;
		}
		return a.r < b.r;
	}
	return a.l < b.l;
}
struct Edge {
	int to, w, next;
} edge[M * 4];
void init() {
	memset(first, -1, sizeof(first));
	sign = 0;
}
int add(int val) {
	return mark[val] ++ == 0;
```

```
}
int cut(int val) {
        return -- mark[val] == 0;
}
void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign++;
}
void dfs(int x, int faz) {
        lef[x] = ++indexs;
        for(int i = first[x]; ~i; i = edge[i].next) {
                if(edge[i].to == faz) {
                        continue;
                }
                dfs(edge[i].to, x);
        }
        rig[x] = indexs;
}
int main() {
        while(~scanf("%d %d", &n, &m)) {
                sz = sqrt(n);
                memset(mark,0,sizeof(mark));
                v.clear();
                for(int i = 1; i <= n; i++ ) {
                        scanf("%d", &arr[i]);
                        v.push_back(arr[i]);
                        belong[i] = i / sz;
                }
                sort(v.begin(), v.end());
                v.erase(unique(v.begin(), v.end()), v.end());
                for(int i = 1; i <= n; i++ ) {
                        arr[i] = getid(arr[i]);
                }
                init();
                for(int i = 1; i <= n - 1; i++ ) {
                        int u, v;
                        scanf("%d %d", &u, &v);
                        add_edge(u, v, 1);
                        add_edge(v, u, 1);
                }
                indexs = 0;
                dfs(1, -1);
```

```
		for(int i = 1; i <= n; i++ ) {
			b[ lef[i] ] = arr[i];
		}
		for(int i = 1; i <= n; i++ ) {
			arr[i] = b[i];
		}
		for(int i = 1; i <= m; i++ ) {
			int x;
			scanf("%d", &x);
			qus[i].id = i;
			qus[i].l = lef[x];
			qus[i].r = rig[x];
		}
		ans = l = r = 0;
		sort(qus + 1, qus + 1 + m, cmp);
		for(int i = 1; i <= m; i ++ ) {
			while(qus[i].l < l) {
				ans += add(arr[--l]);
			}
			while(qus[i].l > l) {
				ans -= cut(arr[l++]);
			}
			while(qus[i].r < r) {
				ans -= cut(arr[r--]);
			}
			while(qus[i].r > r) {
				ans += add(arr[++r]);
			}
			output[qus[i].id] = ans;
		}
		for(int i = 1; i <= m; i++ ) {
			printf("%d\n",output[i]);
		}
	}
	return 0;
}
```

分块

```
///弹飞绵羊为例
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 200000 + 10;
int n, a[MAXN], belong[MAXN], block_siz, block_num, lef[MAXN], rig[MAXN], q, step[MAXN],
dump[MAXN];
template<class T>
inline bool nextInt(T &n) {
        T x = 0, tmp = 1;
        char c = getchar();
        while((c < '0' || c > '9') && c != '-' && c != EOF) c = getchar();
        if(c == EOF) return false;
        if(c == '-') c = getchar(), tmp = -1;
        while(c >= '0' && c <= '9') x *= 10, x += (c - '0'), c = getchar();
        n = x*tmp; return true;
}
template<class T>
inline void Out(T n) {
        if(n < 0) { putchar('-'); n = -n; }
        int len = 0, data[20];
        while(n) { data[len++] = n%10; n /= 10; }
        if(!len) data[len++] = 0;
        while(len--) putchar(data[len]+48);
}
void update(int i) {
        for(int j = rig[i]; j >= lef[i]; j--) {
                int next_pos = j + a[j];
                if(next_pos <= rig[i]) {
                        step[j] = step[next_pos] + 1;
                        dump[j] = dump[next_pos];
                } else {
                        step[j] = 1;
                        dump[j] = next_pos;
                }
        }
}
void init() {
        block_siz = sqrt(n);
        block_num = n / block_siz;
        if(n % block_siz) {
                block_num++;
        }
        for(int i = 1; i <= block_num; i++ ) {
                lef[i] = (i - 1) * block_siz + 1;
                rig[i] = i * block_siz;
        }
        rig[block_num] = n;
        for(int i = 1; i <= n; i++ ) {
                belong[i] = (i - 1) / block_siz + 1;
```

```
	}
	for(int i = block_num; i >= 1; i-- ) {
		update(i);
	}
}
int main() {
	nextInt(n);
	for(int i = 1; i <= n; i++ ) {
		nextInt(a[i]);
	}
	init();
	nextInt(q);
	while(q--) {
		int opt, x, y;
		nextInt(opt);
		if(opt == 1) {
			nextInt(x);
			int ans = 0;
			x++;
			while(x <= n) {
				ans += step[x];
				x = dump[x];
			}
			Out(ans), putchar('\n');
		} else {
			nextInt(x), nextInt(y);
			x++;
			a[x] = y;
			update(belong[x]);
		}
	}
	return 0;
}
```

AC 自动机

AC 自动机是否补全 trie 图而有了两种写法。一种在建 fail 树中暴力会跳，某些情况下会被卡掉。但是保留了树的结构。另一种建 fail 树避免了会跳，但是破坏了原树的结构。看需求修改。以 HDU2222 为例。在一长串中查若干字符串出现次数。

不补全 trie 的暴力会跳的自动机

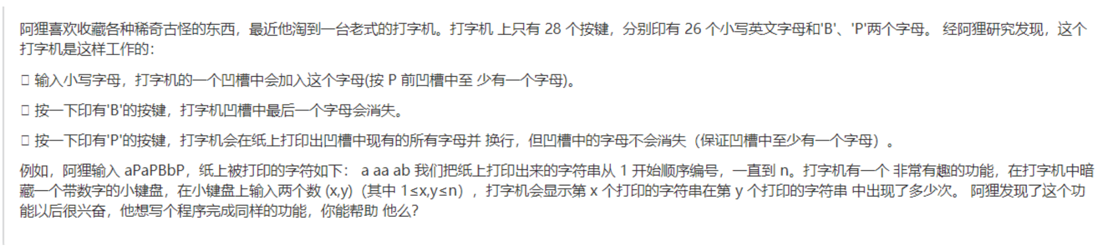
```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 7;
const int maxm = 5e5 + 7;
struct Aho {
    struct state {
        int childs[26];
        int fail, cnt;
        inline void init() {
            for(int i = 0; i < 26; i ++ ) {
                childs[i] = 0;
            }
            fail = cnt = 0;
        }
    } tree[maxm];
    int sign;
    inline void init() {
        sign = 1;
        tree[0].init();
    }
    inline void Insert(char str[]) {
        int len = strlen(str);
        int now = 0;
        for(int i = 0; i < len; i ++ ) {
            int id = str[i] - 'a';
            if(tree[now].childs[id] == 0) {
                tree[sign].init();
                tree[now].childs[id] = sign ++;
            }
            now = tree[now].childs[id];
        }
        tree[now].cnt ++;
    }
    inline void Build() {
        queue<int>que;
        tree[0].fail = -1;
        que.push(0);
        while(!que.empty()) {
            int now = que.front();
            que.pop();
            for(int i = 0; i < 26; i ++ ) {
                if(tree[now].childs[i]) {
                    if(now == 0) {
                        tree[ tree[now].childs[i] ].fail = 0;
                    } else {
                        int v = tree[now].fail;
                        while(v != -1) {
                            if(tree[v].childs[i]) {
                                tree[ tree[now].childs[i] ].fail = tree[v].childs[i];
                                break;
                            }
                            v = tree[v].fail;
                        }
                        if(v == -1) {
                            tree[ tree[now].childs[i] ].fail == 0;
                        }
                    }
                    que.push(tree[now].childs[i]);
                }
            }
        }
}
inline int Get(int now) {
        int ans = 0;
        while(now) {
            ans += tree[now].cnt;
            tree[now].cnt = 0;
            now = tree[now].fail;
        }
        return ans;
}
inline int Match(char str[]) {
        int len = strlen(str);
        int ans = 0, now = 0;
        for(int i = 0; i < len; i ++ ) {
            int id = str[i] - 'a';
            if(tree[now].childs[id]) {
                now = tree[now].childs[id];
            } else {
                int p = tree[now].fail;
                while(p != -1 && tree[p].childs[id] == 0) {
                    p = tree[p].fail;
                }
                if(p == -1) {
                    now = 0;
```

```
                        } else {
                                now = tree[p].childs[id];
                        }
                }
                if(tree[now].cnt) {
                        ans += Get(now);
                }
        }
        return ans;
    }
} aho;
int t, n;
char str[maxn];
int main() {
    while(~scanf("%d", &t)) {
        while(t--) {
            scanf("%d", &n);
            aho.init();
            for(int i=0; i<n; i++) {
                scanf("%s",str);
                aho.Insert(str);
            }
            aho.Build();
            scanf("%s",str);
            printf("%d\n",aho.Match(str));
        }
    }
    return 0;
}
```

补全 trie 图的 ac 自动机

```
///kuanbin  补全 trie 图的 ac 自动机
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e6 + 7;
struct ACauto {
    static const int MAXN = 1e6 + 7;
    int fail[MAXN], ch[MAXN][30], cnt[MAXN], tot, root;
    int new_node() {
        for(int i = 0; i < 26; i++ ) {
            ch[tot][i] = -1;
        }
        cnt[tot++] = 0;
    return tot - 1;
}
void init() {
    tot = 0;
    root = new_node();
}
void insert(char *str, int len) {
    int now = root;
    for(int i = 0; i < len; i++ ) {
        if(ch[now][ str[i] - 'a' ] == -1) {
            ch[now][ str[i] - 'a' ] = new_node();
        }
        now = ch[now][ str[i] - 'a' ];
    }
    cnt[now]++;
}
void get_fail() {
    queue<int>que;
    fail[root] = root;
    for(int i = 0; i < 26; i++ ) {
        if(ch[root][i] == -1) {
            ch[root][i] = root;
        } else {
            fail[ ch[root][i] ] = root;
            que.push(ch[root][i]);
        }
    }
    while(!que.empty()) {
        int now = que.front();
        que.pop();
        for(int i = 0; i < 26; i++ ) {
            if(ch[now][i] == -1) {
                ch[now][i] = ch[ fail[now] ][i];
            } else {
                fail[ ch[now][i] ] = ch[ fail[now] ][i];
                que.push(ch[now][i]);
            }
        }
    }
}
int query(char str[], int len) {
    int now = root;
    int res = 0;
    for(int i = 0; i < len; i++ ) {
            now = ch[now][ str[i] - 'a' ];
            int pos = now;
            while(pos != root) {
                res += cnt[pos];
                cnt[pos] = 0;
                pos = fail[pos];
            }
        }
        return res;
    }
};
ACauto ac;
char buf[MAXN];
int main() {
    int T, n;
    scanf("%d", &T);
    while(T--) {
        scanf("%d", &n);
        ac.init();
        for(int i = 1; i <= n; i++ ) {
            scanf("%s", buf);
            ac.insert(buf, strlen(buf));
        }
        ac.get_fail();
        scanf("%s", buf);
        printf("%d\n", ac.query(buf, strlen(buf)));
    }
    return 0;
}
```

Ac 自动机 fail 图的经典应用，阿狸的打字机



阿狸喜欢收藏各种稀奇古怪的东西，最近他淘到一台老式的打字机。打字机 上只有 28 个按键，分别印有 26 个小写英文字母和'B'、'P'两个字母。 经阿狸研究发现，这个打字机是这样工作的：

 输入小写字母，打字机的一个凹槽中会加入这个字母(按 P 前凹槽中至 少有一个字母)。

 按一下印有'B'的按键，打字机凹槽中最后一个字母会消失。

 按一下印有'P'的按键，打字机会在纸上打印出凹槽中现有的所有字母并 换行，但凹槽中的字母不会消失（保证凹槽中至少有一个字母）。

例如，阿狸输入 aPaPBbP，纸上被打印的字符如下： a aa ab 我们把纸上打印出来的字符串从 1 开始顺序编号，一直到 n。打字机有一个 非常有趣的功能，在打字机中暗藏一个带数字的小键盘，在小键盘上输入两个数 (x,y)（其中 1≤x,y≤n），打字机会显示第 x 个打印的字符串在第 y 个打印的字符串 中出现了多少次。 阿狸发现了这个功能以后很兴奋，他想写个程序完成同样的功能，你能帮助 他么?

```
//树状数组维护 ac 自动机上 fail 树 dfs 序
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e6 + 7;
struct BIT {
    static const int MAXN = 1e6 + 7;
```

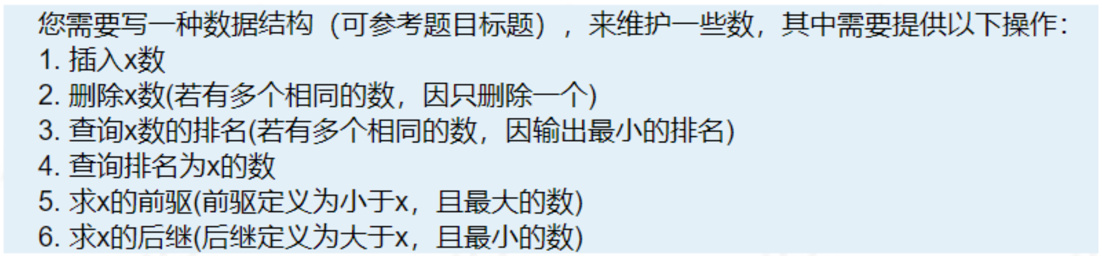
```
        int c[MAXN];
        inline void init() {
                memset(c, 0, sizeof(c));
        }
        inline int lowbit(int x) {
                return x & -x;
        }
        inline void add(int pos, int val) {
                for(int i = pos; i < MAXN; i += lowbit(i) ) {
                        c[i] += val;
                }
        }
        inline int sum(int pos) {
                int ans = 0;
                for(int i = pos; i > 0; i -= lowbit(i)) {
                        ans += c[i];
                }
                return ans;
        }
} bit;
char buf[MAXN];
int lef[MAXN], rig[MAXN], dfs_index, ans[MAXN], vis[MAXN];
vector<pair<int, int> > query[MAXN]; /// q[y] = <x, id>
struct Node {
        int to, w, next;
} edge[MAXN * 4];
int first[MAXN], sign;
void init() {
        memset(first, -1, sizeof(first));
        sign = 0;
}
void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign++;
}
int pos[MAXN];
struct ACauto {
        static const int MAXN = 1e6 + 7;
        int fail[MAXN], ch[MAXN][30], cnt[MAXN], tot, root, fa[MAXN];
        int new_node() {
                memset(ch[tot], -1, sizeof(ch[tot]));
                fail[tot] = -1;
                return tot++;
        }
        void init() {
                tot = 0;
                root = new_node();
        }
        void insert(char *str) {
                int now = root, indexs = 0;
                for(int i = 0; str[i]; i++ ) {
                        if(str[i] == 'P') {
                                pos[++indexs] = now;
                        } else if(str[i] == 'B') {
                                now = fa[now];
                        } else {
                                if(ch[now][ str[i] - 'a' ] == -1) {
                                        ch[now][ str[i] - 'a' ] = new_node();
                                        fa[ ch[now][ str[i] - 'a' ] ] = now;
                                }
                                now = ch[now][ str[i] - 'a' ];
                        }
                }
        }
        void get_fail() {
                queue<int>que;
                fail[root] = root;
                for(int i = 0; i < 26; i++ ) {
                        if(ch[root][i] == -1) {
                                ch[root][i] = root;
                        } else {
                                fail[ ch[root][i] ] = root;
                                que.push(ch[root][i]);
                        }
                }
                while(!que.empty()) {
                        int now = que.front();
                        que.pop();
                        for(int i = 0; i < 26; i++ ) {
                                if(ch[now][i] == -1) {
                                        ch[now][i] = ch[ fail[now] ][i];
                                } else {
                                        fail[ ch[now][i] ] = ch[ fail[now] ][i];
                                        que.push(ch[now][i]);
                                }
                        }
```

```
            }
            for(int i = 1; i < tot; i++ ) {
                add_edge(fail[i], i, 1);
            }
        }
        void cal(char *str) {
            int now = root, id = 0;
            bit.add(lef[0], 1);
            for(int i = 0; str[i]; i++ ) {
                if(str[i] == 'P') {
                    id++;
                    for(int j = 0; j < query[id].size(); j++ ) {
                        int x = pos[query[id][j].first];
                        ans[ query[id][j].second ] = bit.sum(rig[x]) - bit.sum(lef[x] - 1);
                    }
                } else if(str[i] == 'B') {
                    bit.add(lef[now], -1);
                    now = fa[now];
                } else {
                    now = ch[now][ str[i] - 'a' ];
                    bit.add(lef[now], 1);
                }
            }
        }
    } ac;
    void dfs1(int x) {
        lef[x] = ++dfs_index;
        for(int i = first[x]; ~i; i = edge[i].next) {
            dfs1(edge[i].to);
        }
        rig[x] = dfs_index;
    }
    int main() {
        bit.init();
        ac.init();
        init();
        scanf("%s", buf);
        ac.insert(buf);
        ac.get_fail();
        dfs1(0);
        int q;
        scanf("%d", &q);
        for(int i = 1; i <= q; i++ ) {
            int x, y;
            scanf("%d %d", &x, &y);
            query[y].push_back(make_pair(x, i));
        }
        ac.cal(buf);
        for(int i = 1; i <= q; i++ ) {
            printf("%d\n", ans[i]);
        }
        return 0;
    }
```

回文树/回文自动机

```
/// 求 回文串长度 * 它出现次数 的最大值
///BZOJ 3676 回文自动机
#include <bits/stdc++.h>
using namespace std;
struct Palindromic_Tree {
    ///num[i]节点 i 表示的最长回文串的最右端点为回文串结尾的回文串个数
    ///cnt[i]节点 i 表示的本质不同的串的个数,最后要 count 一遍才是正确的
    static const int MAXN = 1e6 + 10;
    static const int SZ = 26;
    int ch[MAXN][SZ], fail[MAXN], cnt[MAXN], num[MAXN], len[MAXN];
    int str[MAXN];
    int last, pos, tot; ///上一次字符所在结点,[字符数组]指针，[树]结点指针
    int new_node(int L) {
        memset(ch[tot], 0, sizeof(ch[tot]));
        cnt[tot] = num[tot] = 0;
        len[tot] = L;
        return tot++;
    }
    void init() {
        tot = last = pos = 0;
        new_node(0), new_node(-1);
        str[pos] = -1;
        fail[0] = 1;
    }
    int get_fail(int x) {
        while(str[pos - len[x] - 1] != str[pos]) {
            x = fail[x];
        }
        return x;
    }
    void add(int id) {
        str[++pos] = id;
        int cur = get_fail(last);
        if(!ch[cur][id]) {
            int now = new_node(len[cur] + 2);
            fail[now] = ch[ get_fail(fail[cur]) ][id];
            ch[cur][id] = now;
            num[now] = num[ fail[now] ] + 1;
        }
        last = ch[cur][id];
        cnt[last]++;
    }
    long long count() {
        long long ans = 0;
        for(int i = tot - 1; i >= 0; i-- ) {
            cnt[ fail[i] ] += cnt[i];
            ans = max(ans, (long long)cnt[i] * len[i]);
        }
        return ans;
    }
} pt;
const int MAXN = 1e6 + 10;
char str[MAXN];
int main() {
    int cas = 1;
    while(~scanf("%s", str)) {
        pt.init();
        for(int i = 0; str[i]; i++ ) {
            pt.add(str[i] - 'a');
        }
        printf("%lld\n", pt.count());
    }
    return 0;
}
```

平衡树



您需要写一种数据结构（可参考题目标题），来维护一些数，其中需要提供以下操作：
1. 插入x数
2. 删除x数(若有多个相同的数，因只删除一个)
3. 查询x数的排名(若有多个相同的数，因输出最小的排名)
4. 查询排名为x的数
5. 求x的前驱(前驱定义为小于x，且最大的数)
6. 求x的后继(后继定义为大于x，且最小的数)

Treap

```
//只有维护单点，维护区间的 fhq treap 暂时没学，因为有 splay 了，暂时够用
/**
 * BZOJ 3224 Treap 版本
 */
 #include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int INF = 0x3F3F3F3F;
const int INTMAX = 0x7FFFFFFF;
const LL LLINF = 0x3F3F3F3F3F3F3F3F;
const double PI = acos(-1);
const int MAXN = 1e6 + 10;
const int MAXM = 1e6 + 10;
struct Treap {
	struct Node {
		int val, cnt, size, random;
		int l, r;
	} tree[MAXN];
	int tot, root, n;
	void build() {
		tot = 0;
		newNode(-INT_MAX), newNode(INT_MAX);
		root = 1, tree[1].r = 2;
		update(root);
	}
	void update(int pos) {
		int l = tree[pos].l;
		int r = tree[pos].r;
		tree[pos].size = tree[pos].cnt + tree[l].size + tree[r].size;
	}
	int newNode(int val) {
		tot++;
		tree[tot].l = tree[tot].r = 0;
		tree[tot].cnt = tree[tot].size = 1;
		tree[tot].val = val;
		tree[tot].random = rand();
		return tot;
	}
	void zig(int &x) {
		int y = tree[x].l;
		tree[x].l = tree[y].r, tree[y].r = x, x = y;
```

```
        update(tree[x].r), update(x);
}
void zag(int &x) {
        int y = tree[x].r;
        tree[x].r = tree[y].l, tree[y].l = x, x = y;
        update(tree[x].l), update(x);
}
void insert(int &now, int val) {
        if(now == 0) {
                now = newNode(val);
                return ;
        }
        if(val == tree[now].val) {
                tree[now].cnt++;
                update(now);
                return ;
        }
        if(val < tree[now].val) {
                insert(tree[now].l, val);
                if(tree[now].random < tree[ tree[now].l ].random) {
                        zig(now);
                }
        } else {
                insert(tree[now].r, val);
                if(tree[now].random < tree[ tree[now].r ].random) {
                        zag(now);
                }
        }
        update(now);
}
int getRankByVal(int now, int val) {
        if(now == 0) {
                return 0;
        }
        if(val == tree[now].val) {
                return tree[ tree[now].l ].size + 1;
        }
        if(val < tree[now].val) {
                return getRankByVal(tree[now].l, val);
        }
        return getRankByVal(tree[now].r, val) + tree[ tree[now].l ].size + tree[now].cnt;
}
int getValByRank(int now, int rank) {
        if(now == 0) {
            return INF;
        }
        if(tree[ tree[now].l ].size >= rank) {
            return getValByRank(tree[now].l, rank);
        }
        if(tree[ tree[now].l ].size + tree[now].cnt >= rank) {
            return tree[now].val;
        }
        return getValByRank(tree[now].r, rank - tree[ tree[now].l ].size - tree[now].cnt);
    }
    int findNodeByVal(int val) {
        int now = root;
        while(1) {
            if(now == 0) {
                return -1;
            }
            if(val == tree[now].val) {
                return now;
            }
            if(val < tree[now].val) {
                now = tree[now].l;
            } else {
                val = tree[now].r;
            }
        }
    }
    int getPre(int val) {
        int ans = 1;
        int now = root;
        while(now) {
            if(val == tree[now].val) {
                if(tree[now].l > 0) {
                    now = tree[now].l;
                    while(tree[now].r > 0) {
                        now = tree[now].r;
                    }
                    ans = now;
                }
                break;
            }
            if(tree[now].val < val && tree[now].val > tree[ans].val) {
                ans = now;
            }
            if(val < tree[now].val) {
```

```
				now = tree[now].l;
			} else {
				now = tree[now].r;
			}
		}
		return tree[ans].val;
	}
	int getNext(int val) {
		int ans = 2;
		int now = root;
		while(now) {
			if(val == tree[now].val) {
				if(tree[now].r > 0) {
					now = tree[now].r;
					while(tree[now].l > 0) {
						now = tree[now].l;
					}
					ans = now;
				}
				break;
			}
			if(tree[now].val > val && tree[now].val < tree[ans].val) {
				ans = now;
			}
			if(val < tree[now].val) {
				now = tree[now].l;
			} else {
				now = tree[now].r;
			}
		}
		return tree[ans].val;
	}
	void remove(int &now, int val) {
		if(now == 0) {
			return ;
		}
		if(val == tree[now].val) {
			if(tree[now].cnt > 1) {
				tree[now].cnt--;
				update(now);
				return ;
			}
			if(tree[now].l || tree[now].r) {
				if(tree[now].r == 0 || tree[ tree[now].l ].random > tree[ tree[now].r ].random)
```

```
{
                    zig(now);
                    remove(tree[now].r, val);
                } else {
                    zag(now);
                    remove(tree[now].l, val);
                }
                update(now);
            } else {
                now = 0;
            }
            return ;
        }
        if(val < tree[now].val) {
            remove(tree[now].l, val);
        } else {
            remove(tree[now].r, val);
        }
        update(now);
    }
} cwl_treap;
int n, opt, x;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cwl_treap.build();
    cin >> n;
    for(int i = 1; i <= n; i++ ) {
        cin >> opt >> x;
        switch(opt) {
            case 1: cwl_treap.insert(cwl_treap.root, x); break;
            case 2: cwl_treap.remove(cwl_treap.root, x); break;
            case 3: cout << cwl_treap.getRankByVal(cwl_treap.root, x) - 1 << endl; break;
            case 4: cout << cwl_treap.getValByRank(cwl_treap.root, x + 1) << endl; break;
            case 5: cout << cwl_treap.getPre(x) << endl;    break;
            case 6: cout << cwl_treap.getNext(x) << endl; break;
        }
    }
    return 0;
}
```

Splay 单点

```
#include <bits/stdc++.h>
```

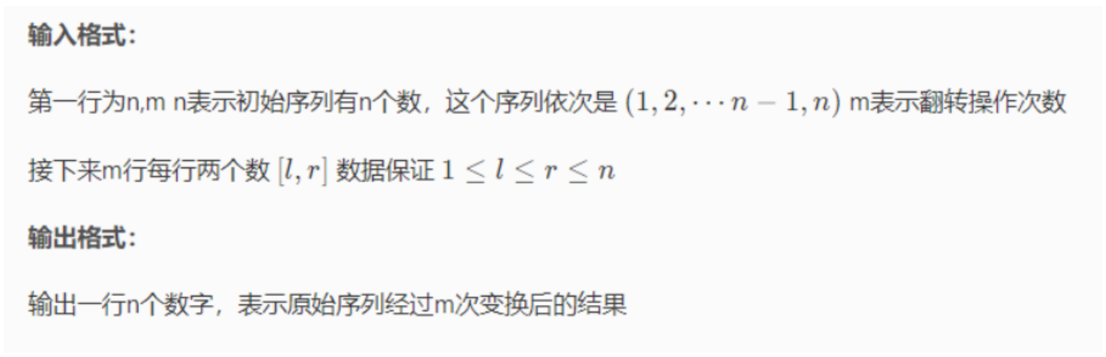
```
using namespace std;
typedef long long LL;
const int maxn = 5e5 + 7;
struct Splay_Tree {
        struct Node {
                int father, childs[2], key, cnt, _size;
                inline void init() {
                        father = childs[0] = childs[1] = key = cnt = _size = 0;
                }
                inline void init(int father, int lchild, int rchild, int key, int cnt, int sz) {
                        this -> father = father, childs[0] = lchild, childs[1] = rchild;
                        this -> key = key, this -> cnt = cnt, _size = sz;
                }
        } tre[maxn];
        int sign, root;
        inline void init() {
                sign = root = 0;
        }
        inline bool judge(int x) {
                return tre[ tre[x].father ].childs[1] == x;
        }
        inline void update(int x) {
                if(x) {
                        tre[x]._size = tre[x].cnt;
                        if(tre[x].childs[0]) {
                                tre[x]._size += tre[ tre[x].childs[0] ]._size;
                        }
                        if(tre[x].childs[1]) {
                                tre[x]._size += tre[ tre[x].childs[1] ]._size;
                        }
                }
        }
        inline void rotate(int x) {
                int y = tre[x].father, z = tre[y].father, k = judge(x);
                //tre[y].childs[k] = tre[x].childs[!k], tre[ tre[x].childs[!k] ].father = y;
                //tre[x].childs[!k] = y, tre[y].father = x;
                //tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;
                if(k == 0) { ///zig
                        tre[y].childs[0] = tre[x].childs[1], tre[ tre[x].childs[1] ].father = y;
                        tre[x].childs[1] = y, tre[y].father = x;
                } else { ///zag
                        tre[y].childs[1] = tre[x].childs[0], tre[ tre[x].childs[0] ].father = y;
                        tre[x].childs[0] = y, tre[y].father = x;
                }
                tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;

                update(y);
        }
        inline void splay(int x,int goal) {
                for(int father; (father = tre[x].father) != goal; rotate(x) ) {
                        if(tre[father].father != goal) {
                                rotate(judge(x) == judge(father) ? father : x);
                        }
                }
                root = x;
        }
        inline void insert_node(int x) {
                if(root == 0) {
                        tre[++sign].init(0, 0, 0, x, 1, 1);
                        root = sign;
                        return ;
                }
                int now = root, father = 0;
                while(1) {
                        if(tre[now].key == x) {
                                tre[now].cnt ++;
                                update(now), update(father);
                                splay(now, 0);
                                break;
                        }
                        father = now;
                        if(x > tre[now].key) {
                                now = tre[now].childs[1];
                        } else {
                                now = tre[now].childs[0];
                        }
                        if(now == 0) {
                                tre[++sign].init(father, 0, 0, x, 1, 1);
                                if(x > tre[father].key) {
                                        tre[father].childs[1] = sign;
                                } else {
                                        tre[father].childs[0] = sign;
                                }
                                update(father);
                                splay(sign, 0);
                                break;
                        }
                }
```

```
}
inline int pre() {
        int now = tre[root].childs[0];
        while(tre[now].childs[1]) {
                now = tre[now].childs[1];
        }
        return now;
}
inline int next() {
        int now = tre[root].childs[1];
        while(tre[now].childs[0]) {
                now = tre[now].childs[0];
        }
        return now;
}
inline int find_rank(int x) { ///  找 x 的排名
        int now = root, ans = 0;
        while(1) {
                if(x < tre[now].key) {
                        now = tre[now].childs[0];
                } else {
                        if(tre[now].childs[0]) {
                                ans += tre[ tre[now].childs[0] ]._size;
                        }
                        if(x == tre[now].key) {
                                splay(now, 0);
                                return ans + 1;
                        }
                        ans += tre[now].cnt;
                        now = tre[now].childs[1];
                }
        }
}
inline int find_rankx(int x) { ///  找排名为 x 的数字
        int now = root;
        while(1) {
                if(tre[now].childs[0] && x <= tre[ tre[now].childs[0] ]._size ) {
                        now = tre[now].childs[0];
                } else {
                        int lchild = tre[now].childs[0], sum = tre[now].cnt;
                        if(lchild) {
                                sum += tre[lchild]._size;
                        }
                        if(x <= sum) {
```

```
                return tre[now].key;
            }
            x -= sum;
            now = tre[now].childs[1];
        }
    }
}
inline void del(int x) {
    find_rank(x);
    if(tre[root].cnt > 1) {
        tre[root].cnt --;
        update(root);
        return ;
    }
    if(!tre[root].childs[0] && !tre[root].childs[1]) {
        tre[root].init();
        root = 0;
        return ;
    }
    if(!tre[root].childs[0]) {
        int old_root = root;
        root = tre[root].childs[1], tre[root].father = 0, tre[old_root].init();
        return ;
    }
    if(!tre[root].childs[1]) {
        int old_root = root;
        root = tre[root].childs[0], tre[root].father = 0, tre[old_root].init();
        return ;
    }
    int pre_node = pre(), old_root = root;
    splay(pre_node, 0);
    tre[root].childs[1] = tre[old_root].childs[1];
    tre[ tre[old_root].childs[1] ].father = root;
    tre[old_root].init();
    update(root);
}
inline bool find(int x) {
    int now = root;
    while(1) {
        if(now == 0) {
            return 0;
        }
        if(x == tre[now].key) {
            splay(now, 0);
```

```
                return 1;
            }
            if(x > tre[now].key) {
                now = tre[now].childs[1];
            } else {
                now = tre[now].childs[0];
            }
        }
    }
} S;

int n, opt, x;

int main() {
    while(~scanf("%d",&n)) {
        S.init();
        for(int i = 1; i <= n; i ++ ) {
            scanf("%d %d",&opt, &x);
            switch(opt) {
            case 1: S.insert_node(x); break;
            case 2: S.del(x); break;
            case 3: printf("%d\n",S.find_rank(x)); break;
            case 4: printf("%d\n",S.find_rankx(x)); break;
            case 5: S.insert_node(x); printf("%d\n",S.tre[S.pre()].key); S.del(x); break;
            case 6: S.insert_node(x); printf("%d\n",S.tre[S.next()].key); S.del(x); break;
            }
        }
    }
    return 0;
}
```

Splay 维护区间翻转



输入格式:

第一行为n,m n表示初始序列有n个数，这个序列依次是 $(1, 2, \cdots n-1, n)$ m表示翻转操作次数

接下来m行每行两个数 $[l, r]$ 数据保证 $1 \leq l \leq r \leq n$

输出格式:

输出一行n个数字，表示原始序列经过m次变换后的结果

```
#include <bits/stdc++.h>
using namespace std;
```

```
typedef long long LL;
const int maxn = 5e5 + 7;
const int inf = 1e9 + 7;
int n, m, arr[maxn];
struct Splay_Tree {
        struct Node {
                int father, childs[2], key, cnt, _size, rev;
                inline void init() {
                        father = childs[0] = childs[1] = key = cnt = _size = rev = 0;
                }
                inline void init(int father, int lchild, int rchild, int key, int cnt, int sz) {
                        this -> father = father, childs[0] = lchild, childs[1] = rchild;
                        this -> key = key, this -> cnt = cnt, _size = sz;
                        this -> rev = 0;
                }
        } tre[maxn];
        int sign, root;
        inline void init() {
                sign = root = 0;
        }
        inline bool judge(int x) {
                return tre[ tre[x].father ].childs[1] == x;
        }
        inline void update(int x) {
                if(x) {
                        tre[x]._size = tre[x].cnt;
                        if(tre[x].childs[0]) {
                                tre[x]._size += tre[ tre[x].childs[0] ]._size;
                        }
                        if(tre[x].childs[1]) {
                                tre[x]._size += tre[ tre[x].childs[1] ]._size;
                        }
                }
        }
        inline void rotate(int x) {
                int y = tre[x].father, z = tre[y].father, k = judge(x);
                pushdown(y), pushdown(x);
                //tre[y].childs[k] = tre[x].childs[!k], tre[ tre[x].childs[!k] ].father = y;
                //tre[x].childs[!k] = y, tre[y].father = x;
                //tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;
                if(k == 0) {///zig
                        tre[y].childs[0] = tre[x].childs[1], tre[ tre[x].childs[1] ].father = y;
                        tre[x].childs[1] = y, tre[y].father = x;
                } else {        ///zag
                        tre[y].childs[1] = tre[x].childs[0], tre[ tre[x].childs[0] ].father = y;
                        tre[x].childs[0] = y, tre[y].father = x;
                }
                tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;

                update(y);

        }
        inline void splay(int x,int goal) {
                for(int father; (father = tre[x].father) != goal; rotate(x) ) {
                        if(tre[father].father != goal) {
                                rotate(judge(x) == judge(father) ? father : x);
                        }
                }
                if(goal == 0) {
                        root = x;
                }
        }
        inline void pushdown(int x) {
                if(x && tre[x].rev) {
                        tre[ tre[x].childs[0] ].rev ^= 1;
                        tre[ tre[x].childs[1] ].rev ^= 1;
                        swap(tre[x].childs[0], tre[x].childs[1]);
                        tre[x].rev = 0;
                }
        }
        int build(int l, int r, int fa) {
                if(l > r) {
                        return 0;
                }
                int mid = (l + r) >> 1;
                int now = ++ sign;
                tre[now].init(fa, 0, 0, arr[mid], 1, 1);
                tre[now].childs[0] = build(l, mid - 1, now);
                tre[now].childs[1] = build(mid + 1, r, now);
                update(now);
                return now;
        }
        int find(int x) {
                int now = root;
                while(1) {
                        pushdown(now);
                        if(x <= tre[ tre[now].childs[0] ]._size) {
                                now = tre[now].childs[0];
                        } else {
                                x -= tre[ tre[now].childs[0] ]._size + 1;
                                if(!x) {
                                        return now;
                                }
                                now = tre[now].childs[1];
                        }
                }
        }
        void reverse(int x, int y) {
                int L = x - 1, R = y + 1, pos;
                L = find(L), R = find(R);
                splay(L, 0);
                splay(R, L);
                pos = tre[root].childs[1];
                pos = tre[pos].childs[0];
                tre[pos].rev ^= 1;
        }
        inline void dfs(int now) {
                pushdown(now);
                if(tre[now].childs[0]) {
                        dfs(tre[now].childs[0]);
                }
                if(tre[now].key != -inf && tre[now].key != inf) {
                        printf("%d ", tre[now].key);
                }
                if(tre[now].childs[1]) {
                        dfs(tre[now].childs[1]);
                }
        }
    } S;
    int main() {
        scanf("%d %d", &n, &m);
        S.init();
        arr[1] = -inf, arr[n + 2] = inf;
        for(int i = 1; i <= n; i ++ ) {
                arr[i + 1] = i;
        }
        S.root = S.build(1, n + 2, 0);
        for(int i = 1; i <= m; i ++ ) {
                int x, y;
                scanf("%d %d", &x, &y);
                S.reverse(x + 1, y + 1);
        }
        S.dfs(S.root);
        return 0;
    }
```

Splay 区间维护最值

```
/**
 * hdu 1754 I Hate It
 * 单点设置，区间求最大
 */
#include <bits/stdc++.h>

using namespace std;
const int MAXN = 2e6 + 10;
const int INF = -0x3F3F3F3F;

int n, m, arr[MAXN];

struct Splay_Tree {

    struct Node {
        int father, childs[2], key, cnt, _size, rev, max, max_lazy;
        inline void init() {
            father = childs[0] = childs[1] = key = cnt = _size = rev = 0;
            max = max_lazy = 0;
        }
        inline void init(int father, int lchild, int rchild, int key, int cnt, int sz) {
            init();
            this -> father = father, childs[0] = lchild, childs[1] = rchild;
            this -> key = key, this -> cnt = cnt, _size = sz;
            this -> rev = 0;
        }
    } tre[MAXN];

    int sign, root;

    inline void init() {
        sign = root = 0;
    }

    inline bool judge(int x) {
        return tre[ tre[x].father ].childs[1] == x;
    }
```

```
inline void update(int x) {
	if(x) {
		tre[x]._size = tre[x].cnt;
		tre[x].max = tre[x].key;
		if(tre[x].childs[0]) {
			tre[x]._size += tre[ tre[x].childs[0] ]._size;
			tre[x].max = max(tre[ tre[x].childs[0] ].max, tre[x].max);
		}
		if(tre[x].childs[1]) {
			tre[x]._size += tre[ tre[x].childs[1] ]._size;
			tre[x].max = max(tre[ tre[x].childs[1] ].max, tre[x].max);
		}
	}
}

inline void rotate(int x) {

	int y = tre[x].father, z = tre[y].father, k = judge(x);
	pushdown(y), pushdown(x);
	//tre[y].childs[k] = tre[x].childs[!k], tre[ tre[x].childs[!k] ].father = y;
	//tre[x].childs[!k] = y, tre[y].father = x;
	//tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;
	if(k == 0) {///zig
		tre[y].childs[0] = tre[x].childs[1], tre[ tre[x].childs[1] ].father = y;
		tre[x].childs[1] = y, tre[y].father = x;
	} else {	///zag
		tre[y].childs[1] = tre[x].childs[0], tre[ tre[x].childs[0] ].father = y;
		tre[x].childs[0] = y, tre[y].father = x;
	}
	tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;

	update(y);

}

inline void splay(int x,int goal) {
	for(int father; (father = tre[x].father) != goal; rotate(x) ) {
		if(tre[father].father != goal) {
			rotate(judge(x) == judge(father) ? father : x);
		}
	}
	if(goal == 0) {
		root = x;
```

```
            }
        }

        inline void pushdown(int x) {
            if(x && tre[x].rev) {
                tre[ tre[x].childs[0] ].rev ^= 1;
                tre[ tre[x].childs[1] ].rev ^= 1;
                swap(tre[x].childs[0], tre[x].childs[1]);
                tre[x].rev = 0;
            }
            if(x && tre[x].max_lazy) {
                tre[ tre[x].childs[0] ].max = max(tre[ tre[x].childs[0] ].max, tre[x].max_lazy);
                tre[ tre[x].childs[1] ].max = max(tre[ tre[x].childs[1] ].max, tre[x].max_lazy);
                tre[    tre[x].childs[0]    ].max_lazy    =    max(tre[    tre[x].childs[0]    ].max_lazy,
tre[x].max_lazy);
                tre[    tre[x].childs[1]    ].max_lazy    =    max(tre[    tre[x].childs[1]    ].max_lazy,
tre[x].max_lazy);
                tre[x].max_lazy = 0;
            }
        }

        int build(int l, int r, int fa) {
            if(l > r) {
                return 0;
            }
            int mid = (l + r) >> 1;
            int now = ++ sign;
            tre[now].init(fa, 0, 0, arr[mid], 1, 1);
            tre[now].childs[0] = build(l, mid - 1, now);
            tre[now].childs[1] = build(mid + 1, r, now);
            update(now);
            return now;
        }

        int find(int x) {
            int now = root;
            while(1) {
                pushdown(now);
                if(x <= tre[ tre[now].childs[0] ]._size) {
                    now = tre[now].childs[0];
                } else {
                    x -= tre[ tre[now].childs[0] ]._size + 1;
                    if(!x) {
                        return now;
                }
                now = tre[now].childs[1];
            }
        }
    }

    void reverse(int x, int y) {
        int L = x - 1, R = y + 1, pos;
        L = find(L), R = find(R);
        splay(L, 0);
        splay(R, L);
        pos = tre[root].childs[1];
        pos = tre[pos].childs[0];
        tre[pos].rev ^= 1;
    }

    inline void dfs(int now) {
        pushdown(now);
        if(tre[now].childs[0]) {
            dfs(tre[now].childs[0]);
        }
        if(tre[now].key != -INF && tre[now].key != INF) {
            printf("%d ", tre[now].key);
        }
        if(tre[now].childs[1]) {
            dfs(tre[now].childs[1]);
        }
    }

    int query_max(int l, int r) {
        int L = l - 1, R = r + 1, pos;
        L = find(L), R = find(R);
        splay(L, 0);
        splay(R, L);
        pos = tre[root].childs[1];
        pos = tre[pos ].childs[0];
        return tre[pos].max;
    }

    void update_max(int l, int r, int val) {
        int L = l - 1, R = r + 1, pos;
        L = find(L), R = find(R);
        splay(L, 0);
        splay(R, L);
```

```
            pos = tre[root].childs[1];
            pos = tre[pos ].childs[0];
            tre[pos].max = val;
            tre[pos].key = val;
            tre[pos].max_lazy = val;
        }

} S;

int main() {
        while(~scanf("%d %d", &n, &m)) {
                S.init();
                arr[1] = -INF, arr[n + 2] = INF;
                for(int i = 1; i <= n; i++ ) {
                        scanf("%d", &arr[i + 1]);
                }
                S.root = S.build(1, n + 2, 0);
                while(m--) {
                        char opt[5];
                        int x, y;
                        scanf("%s %d %d", opt, &x, &y);
                        if(opt[0] == 'Q') {
                                printf("%d\n", S.query_max(x + 1, y + 1));
                        } else {
                                S.update_max(x + 1, x + 1, y);
                        }
                }
        }
        return 0;
}
```

Splay 维护区间删除，区间添加

```
/**
  * hdu 3487
  *  区间删除，区间插入
  */
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2e6 + 10;
const int INF = -0x3F3F3F3F;
int n, m, arr[MAXN];
std::vector<int> vec;
struct Splay_Tree {
```

```
struct Node {
        int father, childs[2], key, cnt, _size, rev;
        inline void init() {
                father = childs[0] = childs[1] = key = cnt = _size = rev = 0;
        }
        inline void init(int father, int lchild, int rchild, int key, int cnt, int sz) {
                this -> father = father, childs[0] = lchild, childs[1] = rchild;
                this -> key = key, this -> cnt = cnt, _size = sz;
                this -> rev = 0;
        }
} tre[MAXN];
int sign, root;
inline void init() {
        sign = root = 0;
}
inline bool judge(int x) {
        return tre[ tre[x].father ].childs[1] == x;
}
inline void update(int x) {
        if(x) {
                tre[x]._size = tre[x].cnt;
                if(tre[x].childs[0]) {
                        tre[x]._size += tre[ tre[x].childs[0] ]._size;
                }
                if(tre[x].childs[1]) {
                        tre[x]._size += tre[ tre[x].childs[1] ]._size;
                }
        }
}
inline void rotate(int x) {
        int y = tre[x].father, z = tre[y].father, k = judge(x);
        pushdown(y), pushdown(x);
        //tre[y].childs[k] = tre[x].childs[!k], tre[ tre[x].childs[!k] ].father = y;
        //tre[x].childs[!k] = y, tre[y].father = x;
        //tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;
        if(k == 0) {///zig
                tre[y].childs[0] = tre[x].childs[1], tre[ tre[x].childs[1] ].father = y;
                tre[x].childs[1] = y, tre[y].father = x;
        } else {        ///zag
                tre[y].childs[1] = tre[x].childs[0], tre[ tre[x].childs[0] ].father = y;
                tre[x].childs[0] = y, tre[y].father = x;
        }
        tre[z].childs[ tre[z].childs[1] == y ] = x, tre[x].father = z;
        update(y);
}
inline void splay(int x,int goal) {
        for(int father; (father = tre[x].father) != goal; rotate(x) ) {
                if(tre[father].father != goal) {
                        rotate(judge(x) == judge(father) ? father : x);
                }
        }
        if(goal == 0) { root = x; }
}
inline void pushdown(int x) {
        if(x && tre[x].rev) {
                tre[ tre[x].childs[0] ].rev ^= 1;
                tre[ tre[x].childs[1] ].rev ^= 1;
                swap(tre[x].childs[0], tre[x].childs[1]);
                tre[x].rev = 0;
        }
}
int build(int l, int r, int fa) {
        if(l > r) { return 0; }
        int mid = (l + r) >> 1;
        int now = ++ sign;
        tre[now].init(fa, 0, 0, arr[mid], 1, 1);
        tre[now].childs[0] = build(l, mid - 1, now);
        tre[now].childs[1] = build(mid + 1, r, now);
        update(now);
        return now;
}
int find(int x) {
        int now = root;
        while(1) {
                pushdown(now);
                if(x <= tre[ tre[now].childs[0] ]._size) {
                        now = tre[now].childs[0];
                } else {
                        x -= tre[ tre[now].childs[0] ]._size + 1;
                        if(!x) {
                                return now;
                        }
                        now = tre[now].childs[1];
                }
        }
}
void reverse(int x, int y) {
        int L = x - 1, R = y + 1, pos;
        L = find(L), R = find(R);
        splay(L, 0);
        splay(R, L);

        int node = tre[R].childs[0];
        update(R);

        pos = tre[root].childs[1];
        pos = tre[pos].childs[0];
        tre[pos].rev ^= 1;
}
void dfs(int now) {
        pushdown(now);
        if(tre[now].childs[0]) { dfs(tre[now].childs[0]); }
        if(tre[now].key != -INF && tre[now].key != INF) {
                //printf("%d ", tre[now].key);
                vec.push_back(tre[now].key);
        }
        if(tre[now].childs[1]) { dfs(tre[now].childs[1]); }
}
void pre_order(int now) {
        if(!now) {
                return ;
        }
        printf("%d_", tre[now].key);
        pre_order(tre[now].childs[0]);
        pre_order(tre[now].childs[1]);
}
void mid_order(int now) {
        if(!now) {
                return ;
        }
        mid_order(tre[now].childs[0]);
        printf("%d_", tre[now].key);
        mid_order(tre[now].childs[1]);

}
void split(int x, int y, int z) {
        int L = x - 1, R = y + 1;
        L = find(L), R = find(R);
        splay(L, 0);
        splay(R, L);
        int node = tre[R].childs[0];
```

```
            tre[R].childs[0] = 0;
            z++;
            L = z - 1, R = z;
            L = find(L), R = find(R);
            splay(L, 0);
            splay(R, L);
            //pre_order(root);
            //puts("");
            //mid_order(root);
            //puts("");
            tre[R].childs[0] = node;
            tre[node].father = R;
            update(R);
        }
} S;

int main() {
        while(~scanf("%d %d", &n, &m)) {
            if(n == -1 && m == -1) {
                break;
            }
            S.init();
            arr[1] = -INF, arr[n + 2] = INF;
            for(int i = 1; i <= n; i++ ) {
                arr[i + 1] = i;
            }
            vec.clear();
            S.root = S.build(1, n + 2, 0);
            while(m--) {
                char opt[10];
                int x, y, z;
                scanf("%s", opt);
                if(opt[0] == 'C') {
                    scanf("%d %d %d", &x, &y, &z);
                    x++, y++, z++;
                    S.split(x, y, z);
                } else {
                    scanf("%d %d", &x, &y);
                    x++, y++;
                    S.reverse(x, y);
                }
            }
            S.dfs(S.root);
            for(int i = 0; i < vec.size(); i++ ) {
                if(i) {
                        printf(" ");
                }
                printf("%d", vec[i]);
        }
        puts("");
    }
    return 0;
}
```

后缀数组

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;
const int N = 1e6 + 10;
const int INF = 0x3f3f3f3f;
struct SuffixArray {
        ///记住数组从 1 开始
        /// SA 后缀数组, rak 排名, height
        ///记得调用 init 函数，交换指针优化
        static const int MAXN = 1e6 + 10;
        int N, M, sa[MAXN], tax[MAXN], Height[MAXN];
        int *rak, *tp;
        int s[MAXN];
        void init() {
                //if(N == 0) N = strlen(s + 1);
                rak = new int[MAXN];
                tp = new int[MAXN];
        }
        void Qsort() {
                for(int i = 0; i <= M; i++) tax[i] = 0;
                for(int i = 1; i <= N; i++) tax[rak[i]]++;
                for(int i = 1; i <= M; i++) tax[i] += tax[i - 1];
                for(int i = N; i >= 1; i--) sa[ tax[rak[tp[i]]]-- ] = tp[i];
        }
        void SuffixSort() {
                M = 300;
                for(int i = 1; i <= N; i++) rak[i] = s[i], tp[i] = i;
                Qsort();
```

```
		for(int w = 1, p = 0; p < N; M = p, w <<= 1) {
			p = 0;
			for(int i = 1; i <= w; i++) tp[++p] = N - w + i;
			for(int i = 1; i <= N; i++) if(sa[i] > w) tp[++p] = sa[i] - w;
			Qsort();
			std::swap(tp, rak);
			rak[sa[1]] = p = 1;
			for(int i = 2; i <= N; i++)
				rak[sa[i]] = (tp[sa[i - 1]] == tp[sa[i]] && tp[sa[i - 1] + w] == tp[sa[i] + w]) ? p :
++p;
		}
	}
	void GetHeight() {
		int j, k = 0;
		for(int i = 1; i <= N; i++) {
			if(k) k--;
			int j = sa[rak[i] - 1];
			while(s[i + k] == s[j + k]) k++;
			Height[rak[i]] = k;
		}
	}
	~SuffixArray() {
		delete [] rak;
		delete [] tp;
	}

} cwl;
bool check(int mid) {
	int maxx = -INF, minn = INF;
	for(int i = 2; i <= cwl.N; i++ ) {
		if(cwl.Height[i] >= mid) {
			maxx = max(maxx, cwl.sa[i]);
			minn = min(minn, cwl.sa[i]);
		} else {
			if(maxx - minn > mid) {
				return true;
			}
			maxx = cwl.sa[i], minn = cwl.sa[i];
		}
	}
	if(maxx - minn > mid) {
		return true;
	}
	return false;
```

```
}
int a[N];
int main() {
	while(~scanf("%d", &cwl.N) && cwl.N) {
		cwl.init();
		for(int i = 0; i < cwl.N; i++ ) {
			scanf("%d", &a[i]);
		}
		for(int i = 1; i <= cwl.N; i++ ) {
			cwl.s[i] = a[i] - a[i - 1] + 88;
		}
		cwl.SuffixSort();
		cwl.GetHeight();
		int l = 0, r = cwl.N;
		while(l + 1 < r) {
			int mid = (l + r) >> 1;
			if(check(mid)) {
				l = mid;
			} else {
				r = mid;
			}
		}
		if(r <= 4) {
			puts("0");
		} else {
			printf("%d\n", r);
		}
	}
	return 0;
}
```

后缀自动机

```
///cwl's 后缀自动机模板
///求出 S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值
///也可以 dfs parent 树求
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 2000000 + 10;
struct SAM {
	static const int MAXN = 2000000 + 10;
```

```
struct Node {
        int fa, ch[26], len;
        void init() { fa = len = 0, memset(ch, 0, sizeof(ch)); }
} node[MAXN];
int last, sign;
int siz[MAXN], t[MAXN], A[MAXN];
void init() { last = sign = 1; }
void extend(int c) {
        int f = last, p = ++sign;
        node[p].init(), last = p;
        node[p].len = node[f].len + 1;
        siz[p] = 1;
        while(f && !node[f].ch[c]) {
                node[f].ch[c] = p, f = node[f].fa;
        }
        if(f == 0) {
                node[p].fa = 1; return ;
        }
        int x = node[f].ch[c], y = ++sign;
        if(node[f].len + 1 == node[x].len) {
                node[p].fa = x;
                sign--;
                return ;
        }
        node[y] = node[x];
        node[y].len = node[f].len + 1;
        node[x].fa = node[p].fa = y;
        while(f && node[f].ch[c] == x) {
                node[f].ch[c] = y, f = node[f].fa;
        }
}
void radix() {
        LL ans = 0;
        memset(t, 0, sizeof(t));
        for(int i = 1; i <= sign; i++ ) { t[ node[i].len ]++; }
        for(int i = 1; i <= sign; i++ ) { t[i] += t[i - 1]; }
        for(int i = 1; i <= sign; i++ ) { A[ t[ node[i].len ]-- ] = i; }
        for(int i = sign; i >= 1; i-- ) {
                int now = A[i];
                siz[ node[now].fa ] += siz[now];
                if(siz[now] > 1) {
                        ans = max(ans, 1LL * siz[now] * node[now].len);
                }
        }
        printf("%lld\n", ans);
    }
} sa;
char s[N];
int L;
int main() {
    scanf("%s", s + 1);
    L = strlen(s + 1);
    sa.init();
    for(int i = 1; i <= L; i++ ) { sa.extend(s[i] - 'a'); }
    sa.radix();
    return 0;
}
```

字符串系列算法

模板 kmp

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 7;
char str[maxn], ttr[maxn];
int len1, len2, Next[maxn];
void getNext(char s[], int len) {
    int i = 0, j = -1;
    Next[i] = j;
    while(i < len) {
        if(j == -1 || s[i] == s[j]) {
            i ++, j ++;
            Next[i] = j;
        } else {
            j = Next[j];
        }
    }
}
int kmp(char str[], int len1, char ttr[], int len2) {
    int i = 0, j = 0;
    getNext(ttr, len2);
    while(i < len1 && j < len2) {
        if(j == -1 || str[i] == ttr[j]) {
            i ++, j ++;
```

```
		} else {
			j = Next[j];
		}
	}
	if(j == len2) {
		return i - j;
	}
	return -1;
}
int main() {
	while(~scanf("%s %s", str, ttr)) {
		len1 = strlen(str), len2 = strlen(ttr);
		printf("%d\n", kmp(str, len1, ttr, len2));
	}
	return 0;
}
```

扩展 kmp

```
扩展 KMP
next[i]:
P[i..m-1]  与  P[0..m-1]的最长公共前缀
ex[i]:
T[i..n-1]  与  P[0..m-1]的最长公共前缀
char T[maxn],P[maxn];
int next[maxn],ex[maxn];
void pre_exkmp(char P[]) {
	int m=strlen(P);
	next[0]=m;
	int j=0,k=1;
	while(j+1<m&&P[j]==P[j+1])
		j++;
	next[1]=j;
	for(int i=2; i<m; i++) {
		int p=next[k]+k-1;
		int L=next[i-k];
		if(i+L<p+1)
			next[i]=L;
		else {
			j=max(0,p-i+1);
			while(i+j<m&&P[i+j]==P[j])
				j++;
			next[i]=j;
			k=i;
```

```
        }
    }
}

void exkmp(char P[],char T[]) {
    int m=strlen(P),n=strlen(T);
    pre_exkmp(P);
    int j=0,k=0;
    while(j<n&&j<m&&P[j]==T[j])
        j++;
    ex[0]=j;
    for(int i=1; i<n; i++) {
        int p=ex[k]+k-1;
        int L=next[i-k];
        if(i+L<p+1)
            ex[i]=L;
        else {
            j=max(0,p-i+1);
            while(i+j<n&&j<m&&T[i+j]==P[j])
                j++;
            ex[i]=j;
            k=i;
        }
    }
}
```

Kmp 循环节

KMP 最小循环节、循环周期：

定理：假设 S 的长度为 len，则 S 存在最小循环节，循环节的长度 L 为 len-next[len]，子串为 S[0…len-next[len]-1]。

（1）如果 len 可以被 len - next[len]整除，则表明字符串 S 可以完全由循环节循环组成，循环周期 T=len/L。

（2）如果不能，说明还需要再添加几个字母才能补全。需要补的个数是循环个数 L-len%L=L-(len-L)%L=L-next[len]%L，L=len-next[len]。

```
for(int i = 1; i <= n; i ++ ) {
    int xlen = i - Next[i];
    if(i % xlen == 0 && i / xlen > 1) {
        printf("%d %d\n", i, i / xlen);
    }
}
```

马拉车算法

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 3e5;
char s[maxn], str[maxn];
int len1, len2, p[maxn], ans;
void init() {
	str[0] = '$';
	str[1] = '#';
	for(int i = 0; i < len1; i ++ ) {
		str[i * 2 + 2] = s[i];
		str[i * 2 + 3] = '#';
	}
	len2 = len1 * 2 + 2;
	str[len2] = '*';
}
void manachr() {
	int id = 0, mx = 0;
	for(int i = 1; i < len2; i ++ ) {
		if(mx > i) {
			p[i] = min(p[2 * id - i], mx - i);
		} else {
			p[i] = 1;
		}
		while(str[i + p[i]] == str[i - p[i]]) {
			p[i] ++;
		}
		if(p[i] + i > mx) {
			mx = p[i]+i;
			id = i;
		}
	}
}
int main() {
	while(~scanf("%s",s)) {
		len1 = strlen(s);
		init();
		manachr();
		ans=0;
		for(int i = 0; i < len2; i ++ ) {
			ans = max(ans,p[i]);
		}
		cout << ans-1 << endl;
	}
	return 0;
}
```

图论

生成树

最小生成树 Kruskal O(Elog2(E))

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 105;
const int maxm = maxn * maxn;
int n, m, pre[maxn];
struct Edge {
	int from, to, cost;
} edge[maxm];
void init() {
	for(int i = 1; i <= n; i ++ ) {
		pre[i] = i;
	}
}
int findx(int x) {
	return pre[x] == x ? x : pre[x] = findx(pre[x]);
}
void join(int x, int y) {
	int fx = findx(x), fy = findx(y);
	if(fx != fy) {
		pre[fx] = fy;
	}
}
bool same(int x, int y) {
	return findx(x) == findx(y);
}
bool cmp(const Edge &a, const Edge &b) {
	return a.cost < b.cost;
}
void kruskal() {
	int ans = 0, num = 0;
	sort(edge + 1, edge + 1 + m, cmp);
	for(int i = 1; i <= m; i ++ ) {
		if(!same(edge[i].from, edge[i].to)) {
			join(edge[i].from, edge[i].to);
			ans += edge[i].cost;
			num ++;
		}
	}
	if(num == n - 1) {
		printf("%d\n", ans);
	} else {
		puts("?");
	}
}
int main() {
	while(~scanf("%d %d", &m, &n) && m) {
		init();
		for(int i = 1; i <= m; i ++ ) {
			scanf("%d %d %d", &edge[i].from, &edge[i].to, &edge[i].cost);
		}
		kruskal();
	}
	return 0;
}
```

最小生成树 Prim O(nlog2(n))

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 105;
int n, m, first[maxn], sign;
struct Edge {
	int to, w, next;
} edge[maxn * maxn];
void init() {
	for(int i = 1; i <= n; i ++ ) {
		first[i] = -1;
	}
	sign = 0;
}
void add_edge(int u, int v, int w) {
	edge[sign].to = v;
	edge[sign].w = w;
```

```
        edge[sign].next = first[u];
        first[u] = sign ++;
}
struct Node {
        int to, cost;
        Node() {}
        Node(int tt, int cc):to(tt), cost(cc) {}
        friend bool operator < (const Node &a, const Node &b) {
                return a.cost > b.cost;
        }
};
int vis[maxn];
void prim() {
        int cnt = 0, ans = 0;
        for(int i = 1; i <= n; i ++ ) {
                vis[i] = 0;
        }
        priority_queue<Node>que;
        que.push(Node(1, 0));
        while(!que.empty()) {
                Node now = que.top();
                que.pop();
                if(!vis[now.to]) {
                        vis[now.to] = 1;
                        ans += now.cost;
                        cnt ++;
                        for(int i = first[now.to]; ~i; i = edge[i].next) {
                                int to = edge[i].to, cost = edge[i].w;
                                if(!vis[to]) {
                                        que.push(Node(to, cost));
                                }
                        }
                }
        }
        if(cnt == n) {
                printf("%d\n", ans);
        } else {
                puts("?");
        }
}
int main() {
        int u, v, w;
        while(~scanf("%d %d", &m, &n) && m) {
                init();
        for(int i = 1; i <= m; i ++ ) {
            scanf("%d %d %d", &u, &v, &w);
            add_edge(u, v, w);
            add_edge(v, u, w);
        }
        prim();
    }
    return 0;
}
```

最短路径

Floyd O(N^3)

```
        for(int k = 1; k <= n; k ++ ) {
            for(int i = 1; i <= n; i ++ ) {
                for(int j = 1; j <= n; j ++ ) {
                    mp[i][j] = min(mp[i][j],mp[i][k]+mp[k][j]);
                }
            }
        }
```

SPFA O(inf)

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 105;
const int maxm = maxn*maxn/2;
const int inf    = 0x3f3f3f3f;
struct Node {
    int to,w,next;
} edge[maxm];
int first[maxn],sign;
int n,m;
void init() {
    for(int i = 1; i <= n; i ++ ) {
        first[i] = 0;
    }
    sign = 1;
}
void add_edge(int u,int v,int w) {
    edge[sign].to = v;
```

```
        edge[sign].w    = w;
        edge[sign].next = first[u];
        first[u] = sign ++;
}
int spfa(int s) {
        int inq[maxn],dis[maxn];
        queue<int>que;
        for(int i = 1; i <= n; i ++ ) {
                inq[i] = 0;
                dis[i] = inf;
        }
        dis[s] = 0, inq[s] = 1;
        que.push(s);
        while(!que.empty()) {
                int now = que.front();
                que.pop();
                inq[now] = 0;
                for(int i = first[now]; i ; i = edge[i].next) {
                        int to = edge[i].to;
                        int ww = edge[i].w;
                        if(dis[to] > dis[now] + ww) {
                                dis[to] = dis[now] + ww;
                                if(!inq[to]) {
                                        que.push(to);
                                        inq[to] = 1;
                                }
                        }
                }
        }
        int ans = 0;
        for(int i = 2; i <= n; i ++ ) {
                ans = max(ans,dis[i]);
        }
        return ans;
}
int main() {
        char str[15];
        int num = 0;
        while(~scanf("%d",&n)) {
                init();
                for(int i = 2; i <= n; i ++ ) {
                        for(int j = 1; j < i; j ++ ) {
                                scanf("%s",str);
                                if(str[0] != 'x') {
```

```
                        sscanf(str,"%d",&num);
                        add_edge(i,j,num);
                        add_edge(j,i,num);
                    }
                }
            }
            printf("%d\n",spfa(1));
        }
        return 0;
}
```

差分约束

(1)、如果要求取最小值，那么求出最长路，那么将不等式全部化成 **xi – xj >= k** 的形式，这样建立 **j->i** 的边，权值为 **k** 的边，如果不等式组中有 **xi – xj > k**，因为一般题目都是对整形变量的约束，化为 **xi – xj >= k+1** 即可，如果 **xi – xj = k** 呢，那么可以变为如下两个：**xi – xj >= k, xi – xj <= k,**进一步变为 **xj – xi >= -k**，建立两条边即可。

(2)、如果求取的是最大值，那么求取最短路，将不等式全部化成 **xi – xj <= k** 的形式，这样建立 **j->i** 的边，权值为 **k** 的边，如果像上面的两种情况，那么同样地标准化就行了。

(3)、如果要判断差分约束系统是否存在解，一般都是判断环，选择求最短路或者最长路求解都行，只是不等式标准化时候不同，判环地话，用 **spfa** 即可，**n** 个点中如果同一个点入队超过 **n** 次，那么即存在环。

值得注意的一点是：建立的图可能不联通，我们只需要加入一个超级源点，比如说求取最长路时图不联通的话，我们只需要加入一个点 **S**，对其他的每个点建立一条权值为 **0** 的边图就联通了，然后从 **S** 点开始进行 **spfa** 判环。最短路类似。

3、 建好图之后直接 **spfa** 或 **bellman-ford** 求解，不能用 **dijstra** 算法，因为一般存在负边，注意初始化的问题。

Dijkstra+priority_queue O(nlog2(n))

```
#include <bits/stdc++.h>
using namespace std;
```

```
const int maxn = 105;
const int maxm = maxn*maxn/2;
const int inf    = 0x3f3f3f3f;
struct Node {
        int to,w,next;
} edge[maxm];
int first[maxn],sign;
int n,m;
void init() {
        for(int i = 1; i <= n; i ++ ) {
                first[i] = 0;
        }
        sign = 1;
}
void add_edge(int u,int v,int w) {
        edge[sign].to = v;
        edge[sign].w    = w;
        edge[sign].next = first[u];
        first[u] = sign ++;
}
struct Heap_Node {
        int to,cost;
        Heap_Node() {}
        Heap_Node(int f,int s):to(f),cost(s) {}
        friend bool operator < (Heap_Node a,Heap_Node b) {
                return a.cost > b.cost;
        }
};
int dijkstra(int s) {
        priority_queue<Heap_Node> heap;
        int dis[maxn], vis[maxn] = {0};
        heap.push(Heap_Node(s,0));
        while(!heap.empty()) {
                Heap_Node now = heap.top();
                heap.pop();
                if(!vis[now.to]) {
                        vis[now.to] = 1;
                        dis[now.to] = now.cost;
                        for(int i = first[now.to]; i; i = edge[i].next) {
                                int to = edge[i].to;
                                int ww = edge[i].w;
                                if(!vis[to]) {
                                        heap.push(Heap_Node(to,now.cost+ww));
                                }
```

```
                    }
                }
            }
        int ans = 0;
        for(int i = 1; i <= n; i ++ ) {
                ans = max(ans,dis[i]);
        }
        return ans;
}
int main() {
        char str[15];
        int num = 0;
        while(~scanf("%d",&n)) {
                init();
                for(int i = 2; i <= n; i ++ ) {
                        for(int j = 1; j < i; j ++ ) {
                                scanf("%s",str);
                                if(str[0] != 'x') {
                                        sscanf(str,"%d",&num);
                                        add_edge(i,j,num);
                                        add_edge(j,i,num);
                                }
                        }
                }
                printf("%d\n",dijkstra(1));
        }
        return 0;
}
```

次短路径

```
//HDU 1595
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1005;
const int inf    = 1<<29;
struct node {
        int to,w,next;
} edge[maxn*maxn];
int first[maxn],sign,n,m,pre[maxn];
void init() {
        for(int i=1; i<=n; i++) {
                first[i]=0;
        }
```

```
    sign=1;
}
void add_edge(int u,int v,int w) {
    edge[sign].w=w;
    edge[sign].to=v;
    edge[sign].next=first[u];
    first[u]=sign++;
}
int tmpu,tmpv,tmpw;
int spfa(int s,int t) {
    queue<int>q;
    int inq[maxn]= {0},dis[maxn],now;
    for(int i=1; i<=n; i++) {
        dis[i]=inf;
    }
    q.push(s);
    dis[s]=0;
    while(!q.empty()) {
        now=q.front();
        q.pop();
        inq[now]=0;
        for(int i=first[now]; i; i=edge[i].next) {
            int to=edge[i].to;
            if(now==tmpu&&to==tmpv||now==tmpv&&to==tmpu) {
                continue;
            }
            if(dis[to]>dis[now]+edge[i].w) {
                dis[to]=dis[now]+edge[i].w;
                pre[to]=now;
                if(!inq[to]) {
                    inq[to]=1;
                    q.push(to);
                }
            }
        }
    }
    if(dis[t]==inf) {
        return -1;
    } else {
        return dis[t];
    }
}
int main() {
    int u,v,w,ans;
```

```
	while(~scanf("%d%d",&n,&m)) {
		init();
		for(int i=1; i<=m; i++) {
			scanf("%d%d%d",&u,&v,&w);
			add_edge(u,v,w);
			add_edge(v,u,w);
		}
		memset(pre,-1,sizeof(pre));
		ans=spfa(1,n);

		stack<int>s;
		int pos=n;
		while(pre[pos]!=-1) {
			s.push(pos);
			pos=pre[pos];
		}
		s.push(1);
		vector<int>vec;
		while(!s.empty()) {
			vec.push_back(s.top());
			s.pop();
		}
		for(int i=0; i<vec.size()-1; i++) {
			tmpu=vec[i],tmpv=vec[i+1];
			tmpw=spfa(1,n);
			if(tmpw!=-1) {
				ans=max(ans,tmpw);
			}
		}
		printf("%d\n",ans);
	}
	return 0;
}
```

拓扑排序

拓扑模板

```
///POJ 2367
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
```

```
#include <queue>
using namespace std;
const int maxn = 105;
const int maxm = 1e4 + 7;
int n, m, first[maxn], sign;
int indegree[maxn];
struct Edge {
        int to, w, next;
} edge[maxm];
inline void init() {
        for(int i = 0; i <= n; i ++ ) {
                first[i] = -1;
        }
        sign = 0;
}
inline void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign ++;
}
inline void topusort() {
        queue<int>que;
        vector<int>vec;
        for(int i = 1; i <= n; i ++ ) {
                if(indegree[i] == 0) {
                        que.push(i);
                }
        }
        while(!que.empty()) {
                int now = que.front();
                que.pop();
                vec.push_back(now);
                for(int i = first[now]; ~i; i = edge[i].next) {
                        int to = edge[i].to;
                        indegree[to] --;
                        if(indegree[to] == 0) {
                                que.push(to);
                        }
                }
        }
        for(int i = 0; i < vec.size(); i ++ ) {
                if(i) {
                        printf("");
        }
        printf("%d", vec[i]);
    }
    puts("");
}
int main() {
    while(~scanf("%d", &n)) {
        init();
        memset(indegree, 0, sizeof(indegree));
        for(int i = 1; i <= n; i ++ ) {
            int num;
            while(scanf("%d", &num) && num) {
                add_edge(i, num, 1);
                indegree[num] ++;
            }
        }
        topusort();
    }
    return 0;
}
```

二分图

匈牙利算法模板

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 505;
const int maxm = 2005;
int vis[maxn],link[maxn];
int k,n,m,x,y,sum;
struct Edge {
    int to,w,next;
} edge[maxm];
int first[maxn], sign;
void init() {
    for(int i = 1; i <= n; i ++ ) {
        first[i] = 0;
    }
    sign = 1;
}
void add_edge(int u,int v,int w) {
    edge[sign].to = v;
    edge[sign].w    = w;
    edge[sign].next = first[u];
    first[u] = sign ++;
}
int Hungarian_algorithm(int x) {
    for(int i = first[x]; i; i = edge[i].next) {
        int to = edge[i].to;
        if(!vis[to]) {
            vis[to] = 1;
            if(!link[to] || Hungarian_algorithm(link[to])) {
                link[to] = x;
                return 1;
            }
        }
    }
    return 0;
}
int main() {
    while(~scanf("%d",&k) && k) {
        scanf("%d %d",&n,&m);
        memset(link,0,sizeof(link));
        init();
        for(int i = 1; i <= k; i ++ ) {
            scanf("%d %d",&x,&y);
            add_edge(x,y,1);
        }
        sum = 0;
        for(int i = 1; i <= n; i ++ ) {
            memset(vis,0,sizeof(vis));
            if(Hungarian_algorithm(i)) {
                sum ++;
            }
        }
        printf("%d\n",sum);
    }
    return 0;
}
```

二分图染色

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2005;
int col[maxn];
int t,n,m,x,y,ok;
vector<int>E[maxn];
bool bfs(int s) {
        queue<int>que;
        que.push(s);
        col[s] = 1;
        while(!que.empty()) {
                int now = que.front();
                que.pop();
                for(int i = 0; i < E[now].size() ; i ++ ) {
                        int to = E[now][i];
                        if(col[to] == -1) {
                                que.push(to);
                                col[to] = !col[now];
                        } else if(col[to] == col[now]) {
                                return 0;
                        }
                }
        }
        return 1;
}
int main() {
        int cas = 1;
        scanf("%d",&t);
        while(t--) {
                scanf("%d %d",&n,&m);
                for(int i = 1; i <= n; i ++ ) {
                        E[i].clear();
                }
                for(int i = 1; i <= m; i ++ ) {
                        scanf("%d %d",&x,&y);
                        E[x].push_back(y);
                        E[y].push_back(x);
                }
                memset(col,-1,sizeof(col));
                ok = 1;
                for(int i = 1; i <= n; i ++ ) {
                        if(col[i] == -1 && !bfs(i)) {
                                ok = 0;
                                break;
                        }
                }
                if(ok) {
                        printf("Scenario #%d:\n",cas++);
```

```
            puts("No suspicious bugs found!");
        } else {
            printf("Scenario #%d:\n",cas++);
            puts("Suspicious bugs found!");
        }
        puts("");
    }
    return 0;
}
```

欧拉回路

Ps:欧拉回路就是一笔画问题:

从某个点出发，不重复的经过所有边回到起点叫回路，经过所有边不用回到起点叫同路。

对于无向图，如果每个点的度都是偶数，则必存在欧拉回路。如果仅有两个点度是奇数，其他都是偶数，则存在欧拉通路。两个奇度的点风别是起点和终点。

对于有向图，如果所有点的入度等于出度。则必然存在欧拉回路。如果有一个点出度-入度==1，出度-入度==1，其他点出度==入度，则存在欧拉通路。这两个点是起点和终点。

下面代码主要是对路径的搜索

递归版本

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 7;
const int MAXM = 5e5 + 7;
int n, m, first[MAXN], sign, vis[MAXN];
struct Edge {
    int to, w, next;
} edge[MAXM * 4];
inline void init() {
    for(int i = 0; i <= n; i++ ) {
        first[i] = -1;
        vis[i] = 0;
    }
    sign = 0;
}
inline void add_edge(int u, int v, int w) {
    edge[sign].to = v;
    edge[sign].w = w;
    edge[sign].next = first[u];
    first[u] = sign++;
}
void dfs(int x) {
	for(int i = first[x]; ~i; i = edge[i].next) {
		int to = edge[i].to;
		if(!vis[i]) {
			vis[i] = 1;
			dfs(to);
		}
	}
	printf("%d\n", x);
}
int main() {
	while(~scanf("%d %d", &n, &m)) {
		init();
		for(int i = 1; i <= m; i++ ) {
			int u, v;
			scanf("%d %d", &u, &v);
			add_edge(u, v, 1);
			add_edge(v, u, 1);
		}
		dfs(1);
	}
	return 0;
}
```

非递归版本

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 7;
const int MAXM = 5e5 + 7;
int n, m, first[MAXN], sign, vis[MAXN];
struct Edge {
	int to, w, next;
} edge[MAXM * 4];
inline void init() {
	for(int i = 0; i <= n; i++ ) {
		first[i] = -1;
		vis[i] = 0;
	}
	sign = 0;
}
inline void add_edge(int u, int v, int w) {
	edge[sign].to = v;
```

```
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign++;
}
stack<int>st, ans;
void eulur(int start) {
        while(!st.empty()) {
                st.pop();
        }
        while(!ans.empty()) {
                ans.pop();
        }
        st.push(start);
        while(!st.empty()) {
                int x = st.top(), i = first[x];
                while(~i && vis[i]) {
                        i = edge[i].next;
                }
                if(~i) {
                        st.push(edge[i].to);
                        //vis[i] = vis[i ^ 1] = 1;
                        vis[i] = 1;
                        first[x] = edge[i].next;
                } else {
                        st.pop();
                        ans.push(x);
                }
        }
}
int main() {
        while(~scanf("%d %d", &n, &m)) {
                init();
                for(int i = 1; i <= m; i++ ) {
                        int u, v;
                        scanf("%d %d", &u, &v);
                        add_edge(u, v, 1);
                        add_edge(v, u, 1);
                }
                eulur(1);
                while(!ans.empty()) {
                        printf("%d\n", ans.top());
                        ans.pop();
                }
        }
```

```
    return 0;
}
```

网络流

EK 最大流

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 7;
const int INF = 0x3F3F3F3F;
typedef long long LL;
int n, m, first[MAXN], sign;
struct Edge {
    int to, flow, next;
} edge[MAXN << 2];
void init() {
    for(int i = 0; i <= n; i++ ) {
        first[i] = -1;
    }
    sign = 0;
}
void add_edge(int u, int v, int flow) {
    edge[sign].to = v, edge[sign].flow = flow, edge[sign].next = first[u];
    first[u] = sign++;
    edge[sign].to = u, edge[sign].flow = 0, edge[sign].next = first[v];
    first[v] = sign++;
}
int pre[MAXN], vis[MAXN];
bool bfs(int s, int t) {
    memset(vis, 0, sizeof(vis));
    memset(pre, -1, sizeof(pre));
    queue<int>que;
    vis[s] = 1;
    que.push(s);
    while(!que.empty()) {
        int now = que.front();
        que.pop();
        if(now == t) {
            return 1;
        }
        for(int i = first[now]; ~i; i = edge[i].next) {
            int to = edge[i].to, flow = edge[i].flow;
            if(!vis[to] && flow > 0) {
                vis[to] = 1;
                pre[to] = i;
                que.push(to);
            }
        }
    }
    return 0;
}
int edomon_krap(int s, int t) {
    int max_flow = 0, min_flow = INF;
    while(bfs(s, t)) {
        min_flow = INF;
        for(int i = t, pos; i != s; i = edge[pos^1].to) {
            pos = pre[i];
            min_flow = min(min_flow, edge[pos].flow);
        }
        for(int i = t, pos; i != s; i = edge[pos^1].to) {
            pos = pre[i];
            edge[pos].flow -= min_flow;
            edge[pos^1].flow += min_flow;
        }
        max_flow += min_flow;
    }
    return max_flow;
}
int main() {
    int T;
    scanf("%d", &T);
    for(int cas = 1; cas <= T; cas++ ) {
        scanf("%d %d", &n, &m);
        init();
        for(int i = 1; i <= m; i++ ) {
            int u, v, w;
            scanf("%d %d %d", &u, &v, &w);
            add_edge(u, v, w);
        }
        printf("Case %d: %d\n", cas, edomon_krap(1, n));
    }
    return 0;
}
```

当前弧优化的 dinic 最大流

```
#include <bits/stdc++.h>
using namespace std;
struct Dinic {
    static const int MAXN = 1e4 + 7;
    static const int MAXM = 1e5 + 7;
    static const int INF = 0x3f3f3f3f;
    int n, m, s, t;
    int first[MAXN], cur[MAXN], dist[MAXN], sign;
    struct Node {
        int to, flow, next;
    } edge[MAXM * 4];
    inline void init(int start, int vertex, int ss, int tt) {
        n = vertex, s = ss, t = tt;
        for(int i = start; i <= n; i++ ) {
            first[i] = -1;
        }
        sign = 0;
    }
    inline void addEdge(int u, int v, int flow) {
        edge[sign].to = v, edge[sign].flow = flow, edge[sign].next = first[u];
        first[u] = sign++;
    }
    inline void add_edge(int u, int v, int flow) {
        addEdge(u, v, flow);
        addEdge(v, u, 0);
    }
    inline int dinic() {
        int max_flow = 0;
        while(bfs(s, t)) {
            for(int i = 0; i <= n; i++ ) {
                cur[i] = first[i];
            }
            max_flow += dfs(s, INF);
        }
        return max_flow;
    }
    bool bfs(int s, int t) {
        memset(dist, -1, sizeof(dist));
        queue<int>que;
        que.push(s), dist[s] = 0;
        while(!que.empty()) {
```

```
                        int now = que.front();
                        que.pop();
                        if(now == t) {
                                return 1;
                        }
                        for(int i = first[now]; ~i; i = edge[i].next) {
                                int to = edge[i].to, flow = edge[i].flow;
                                if(dist[to] == -1 && flow > 0) {
                                        dist[to] = dist[now] + 1;
                                        que.push(to);
                                }
                        }
                }
                return 0;
        }
        int dfs(int now, int max_flow) {
                if(now == t) {
                        return max_flow;
                }
                for(int &i = cur[now]; ~i; i = edge[i].next) {
                        int to = edge[i].to, flow = edge[i].flow;
                        if(dist[to] == dist[now] + 1 && flow > 0) {
                                int next_flow = dfs(to, min(flow, max_flow));
                                if(next_flow > 0) {
                                        edge[i].flow -= next_flow;
                                        edge[i ^ 1].flow += next_flow;
                                        return next_flow;
                                }
                        }
                }
                return 0;
        }
        ///显示二分图匹配结果,n,m 分别是两个集合的大小
        void show(int n, int m) {
                for(int i = n + 1; i <= n + m; i++ ) {
                        for(int j = first[i]; ~j; j = edge[j].next) {
                                if(edge[j].flow == 1 && edge[j].to != t) {
                                        printf("%d %d\n", edge[j].to, i);
                                }
                        }
                }
        }
} cwl;
int main() {
        int n, m;
        int u, v;
        scanf("%d %d", &n, &m);
        cwl.init(0, n + m + 1, 0, n + m + 1);
        for(int i = 1; i <= n; i++ ) {
                cwl.add_edge(0, i, 1);
        }
        for(int i = n + 1; i <= n + m; i++ ) {
                cwl.add_edge(i, n + m + 1, 1);
        }
        while(~scanf("%d %d", &u, &v)) {
                if(u == -1 && v == -1) {
                        break;
                }
                cwl.add_edge(u, v, 1);
        }
        int ans = cwl.dinic();
        printf("%d\n", ans);
        if(ans) {
                cwl.show(n, m);
        } else {
                puts("No Solution!");
        }
        return 0;
}
```

比较玄学可能更快的 dinic，优化了 dfs

```
///网络流 24 题  圆桌问题
#include <bits/stdc++.h>
using namespace std;
const int N = 305;
int a[N], b[N];
struct Dinic {
    static const int MAXN = 500 + 7;
    static const int MAXM = MAXN * MAXN;
    static const int INF = 0x3f3f3f3f;
    int n, m, s, t;
    int first[MAXN], cur[MAXN], dist[MAXN], sign;
    struct Node {
        int to, flow, next;
    } edge[MAXM * 4];
    inline void init(int start, int vertex, int ss, int tt) {
        n = vertex, s = ss, t = tt;
        for(int i = start; i <= n; i++ ) {
            first[i] = -1;
        }
        sign = 0;
    }
    inline void addEdge(int u, int v, int flow) {
        edge[sign].to = v, edge[sign].flow = flow, edge[sign].next = first[u];
        first[u] = sign++;
    }
    inline void add_edge(int u, int v, int flow) {
        addEdge(u, v, flow);
        addEdge(v, u, 0);
    }
    inline int dinic() {
        int max_flow = 0;
        while(bfs(s, t)) {
            for(int i = 0; i <= n; i++ ) {
                cur[i] = first[i];
            }
            max_flow += dfs(s, INF);
        }
        return max_flow;
    }
    bool bfs(int s, int t) {
        memset(dist, -1, sizeof(dist));
        queue<int>que;
        que.push(s), dist[s] = 0;
        while(!que.empty()) {
            int now = que.front();
            que.pop();
            if(now == t) {
                return 1;
            }
            for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to, flow = edge[i].flow;
                if(dist[to] == -1 && flow > 0) {
                    dist[to] = dist[now] + 1;
                    que.push(to);
                }
            }
        }
        return 0;
    }
    int dfs(int now, int max_flow) {
        if(now == t) {
            return max_flow;
        }
        int ans = 0, next_flow = 0;
        for(int &i = cur[now]; ~i; i = edge[i].next) {
            int to = edge[i].to, flow = edge[i].flow;
            if(dist[to] == dist[now] + 1 && flow > 0) {
                next_flow = dfs(to, min(max_flow - ans, flow));
                ans += next_flow;
                edge[i].flow -= next_flow;
                edge[i ^ 1].flow += next_flow;
                if(ans == max_flow) {
                    return max_flow;
                }

            }
        }
        if(ans == 0) {
            return dist[now] = 0;
        }
        return ans;
    }
    void show(int n, int m) {
        puts("1");
        vector<pair<int, int> >vec;
        for(int i = n + 1; i <= n + m; i++ ) {
            for(int j = first[i]; ~j; j = edge[j].next) {
                if(edge[j].flow == 1 && edge[j].to != t) {
                    //printf("%d %d\n", edge[j].to, i);
                    vec.push_back(make_pair(edge[j].to, i - n));
                }
            }
        }
        vector<int>E[MAXN];
        for(int i = 0; i < vec.size(); i++ ) {
            E[vec[i].first].push_back(vec[i].second);
        }
        for(int i = 1; i <= n; i++ ) {
            for(int j = 0; j < E[i].size(); j++ ) {
                if(j) {
                    printf("");
                }
                printf("%d", E[i][j]);
            }
```

```
            puts("");
        }
    }
} cwl;
int main() {
    int n, m, sum = 0;
    scanf("%d %d", &n, &m);
    cwl.init(0, n + m + 1, 0, n + m + 1);
    for(int i = 1; i <= n; i++ ) {
        scanf("%d", &a[i]);
        sum += a[i];
    }
    for(int i = 1; i <= m; i++ ) {
        scanf("%d", &b[i]);
    }
    for(int i = 1; i <= n; i++ ) {
        cwl.add_edge(0, i, a[i]);
    }
    for(int i = n + 1; i <= n + m; i++ ) {
        cwl.add_edge(i, n + m + 1, b[i - n]);
    }
    for(int i = 1; i <= n; i++ ) {
        for(int j = 1; j <= m; j++ ) {
            cwl.add_edge(i, j + n, 1);
        }
    }
    int ans = cwl.dinic();
    if(ans == sum) {
        cwl.show(n, m);
    } else {
        puts("0");
    }
    return 0;
}
```

最小费用最大流

```
#include <bits/stdc++.h>
using namespace std;
struct MCMF {
    static const int MAXN = 5010;
    static const int MAXM = 50010;
    static const int INF = 0x7FFFFFFF;
static const int INF0X3F = 0x3f3f3f3f;
int n, m, first[MAXN], s, t, sign;
int dist[MAXN], inq[MAXN], pre[MAXN], incf[MAXN];
int max_flow, min_cost;
struct Edge {
        int to, cap, cost, next;
} edge[MAXM * 4];
void init(int l, int r, int ss, int tt) {
        for(int i = l; i <= r; i++ ) {
                first[i] = -1;
        }
        s = ss, t = tt, sign = 0;
        max_flow = min_cost = 0;
}
void add_edge(int u, int v, int cap, int cost) {
        edge[sign].to = v, edge[sign].cap = cap, edge[sign].cost = cost;
        edge[sign].next = first[u], first[u] = sign++;
        edge[sign].to = u, edge[sign].cap = 0, edge[sign].cost = -cost;
        edge[sign].next = first[v], first[v] = sign++;
}
bool spfa(int s, int t) {
        for(int i = 0; i < MAXN; i++ ) {
                dist[i] = INF, inq[i] = 0;
        }
        queue<int>que;
        que.push(s), inq[s] = 1, dist[s] = 0;
        incf[s] = INF0X3F;
        while(!que.empty()) {
                int now = que.front();
                que.pop();
                inq[now] = 0;
                for(int i = first[now]; ~i; i = edge[i].next) {
                        int to = edge[i].to, cap = edge[i].cap, cost = edge[i].cost;
                        if(cap > 0 && dist[to] > dist[now] + cost) {
                                dist[to] = dist[now] + cost;
                                incf[to] = min(incf[now], cap);
                                pre[to] = i;
                                if(!inq[to]) {
                                        que.push(to);
                                        inq[to] = 1;
                                }
                        }
                }
        }
```

```
		return dist[t] != INF;
	}
	void update(int s, int t) {
		int x = t;
		while(x != s) {
			int pos = pre[x];
			edge[pos].cap -= incf[t];
			edge[pos ^ 1].cap += incf[t];
			x = edge[pos ^ 1].to;
		}
		max_flow += incf[t];
		min_cost += dist[t] * incf[t];
	}
	void minCostMaxFlow(int s, int t) {
		while(spfa(s, t)) {
			update(s, t);
		}
	}
} cwl;
int main() {
	int n, m, s, t;
	while(~scanf("%d %d", &n, &m)) {
		s = 0, t = n + 1;
		cwl.init(s, t, s, t);
		for(int i = 1; i <= m; i++ ) {
			int u, v, cost;
			scanf("%d %d %d", &u, &v, &cost);
			u++, v++;
			cwl.add_edge(u, v, 1, cost);
		}
		cwl.add_edge(s, 1, 2, 0);
		cwl.add_edge(n, t, 2, 0);
		cwl.minCostMaxFlow(s, t);
		if(cwl.max_flow == 2) {
			printf("%d\n", cwl.min_cost);
		} else {
			puts("-1");
		}
	}
	return 0;
}
```

强连通/双联通问题

//缩环成点，两种联通分别对应有向图和无向图。调用后,color 一样的点属于一个环,num，cnt 一般是每个环的组成点有多少。

强连通分量

```
//题意是升序输出汇点
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 5005;
const int MAXM = MAXN * MAXN;
typedef long long LL;
int n, m, first[MAXN], sign;
int low[MAXN], dfn[MAXN], ins[MAXN], color[MAXN], num[MAXN], degree[MAXN];
struct Edge {
	int to, w, next;
} edge[MAXM];
int indx, scc;
inline void init() {
	for(int i = 0; i <= n; i++ ) {
		first[i] = -1;
	}
	sign = 0;
}
inline void add_edge(int u, int v, int w) {
	edge[sign].to = v;
	edge[sign].w = w;
	edge[sign].next = first[u];
	first[u] = sign++;
}
void tarjan(int x) {
	low[x] = dfn[x] = ++indx;
	ins[x] = 1;
	s.push(x);
	for(int i = first[x]; ~i; i = edge[i].next) {
		int to = edge[i].to;
		if(!dfn[to]) {
			tarjan(to);
			low[x] = min(low[to], low[x]);
		} else if(ins[to]) {
			low[x] = min(low[x], dfn[to]);
		}
```

```
        }
        if(low[x] == dfn[x]) {
                int now;
                scc++;
                do {
                        now = s.top();
                        s.pop();
                        ins[now] = 0;
                        color[now] = scc;
                        num[scc]++;

                } while(now != x);
        }
}
int main() {
        while(~scanf("%d", &n) && n) {
                scanf("%d", &m);
                init();
                init_tarjan();
                for(int i = 1; i <= m; i++ ) {
                        int u, v, w;
                        scanf("%d %d", &u, &v);
                        add_edge(u, v, 1);
                }
                for(int i = 1; i <= n; i++ ) {
                        if(!dfn[i]) {
                                tarjan(i);
                        }
                }
                for(int i = 1; i <= n; i++ ) {
                        for(int j = first[i]; ~j; j = edge[j].next) {
                                int to = edge[j].to;
                                if(color[i] != color[to]) {
                                        degree[ color[i] ]++;
                                }
                        }
                }
                set<int>s;
                for(int i = 1; i <= scc; i++ ) {
                        if(degree[i] == 0) {
                                s.insert(i);
                        }
                }
                vector<int>vec;
```

```
		for(int i = 1; i <= n; i++ ) {
			if(s.count(color[i])) {
				vec.push_back(i);
			}
		}
		sort(vec.begin(), vec.end());
		for(int i = 0; i < vec.size(); i++ ) {
			if(i) {
				printf("");
			}
			printf("%d", vec[i]);
		}
		puts("");
	}
	return 0;
}
```

双联通缩点与桥

```
/**
在树中至少添加多少条边能使图变为双连通图。
结论：添加边数=（树中度为 1 的节点数+1）/2

具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，
这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的
环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样
一对一对找完，恰好是(leaf+1)/2 次，把所有点收缩到了一起。*/
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 5e3 + 7;
const int MAXM = 1e4 + 7;
struct Edge {
	int to, w, next;
	int cut;
} edge[MAXM * 2];
int first[MAXN], dfn[MAXN], low[MAXN], belong[MAXN], degree[MAXN], ins[MAXM];
stack<int>st;
int bridge; ///桥的数目
int sign, n, m;
int indx, scc;
inline void init() {
	memset(first, -1, sizeof(first));
	sign = 0;
}
```

```
inline void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].cut = 0;
        edge[sign].next = first[u];
        first[u] = sign++;
}
void tarjan(int now, int pre) {
        dfn[now] = low[now] = ++indx;
        st.push(now);
        ins[now] = 1;
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(to == pre) {
                        continue;
                }
                if(!dfn[to]) {
                        tarjan(to, now);
                        low[now] = min(low[now], low[to]);
                        if(low[to] > dfn[now]) {
                                bridge++;
                                edge[i].cut = 1;
                                edge[i^1].cut = 1;
                        }
                } else if(ins[to]) {
                        low[now] = min(low[now], dfn[to]);
                }
        }
        if(dfn[now] == low[now]) {
                scc++;
                int top;
                do {
                        top = st.top();
                        st.pop();
                        ins[top] = 0;
                        belong[top] = scc;
                } while(top != now);
        }
}
int main() {
        int u, v;
        while(~scanf("%d %d", &n, &m)) {
                init();
                memset(dfn, 0, sizeof(dfn));
```

```
            memset(ins, 0, sizeof(ins));
            memset(belong, 0, sizeof(belong));
            memset(degree, 0, sizeof(degree));
            indx = scc = 0;
            while(!st.empty()) {
                st.pop();
            }
            for(int i = 1; i<= m; i++ ) {
                scanf("%d %d", &u, &v);
                add_edge(u, v, 1);
                add_edge(v, u, 1);
            }
            tarjan(1, -1);
            for(int i = 1; i <= n; i++ ) {
                for(int j = first[i]; ~j; j = edge[j].next) {
                    int from = i, to = edge[j].to;
                    if(edge[j].cut) {
                        degree[ belong[i] ]++;
                    }
                }
            }
            int ans = 0;
            for(int i = 1; i <= scc; i++ ) {
                if(degree[i] == 1) {
                    ans ++;
                }
            }
            ans = (ans + 1) / 2;
            printf("%d\n", ans);
        }
        return 0;
}
```

割点判定

```
/**
割点(割顶)模板：输出割点
分两种情况
1，找一个点作为根开始 tarjan,对于 dfs 树上如果根有两个以上的子树那么它是割点。
2，对于其他点,low[to] >= dfn[now]。
*/
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 7;
const int MAXM = 2e5 + 7;
int n, m, first[MAXN], sign, indx, scc;
int dfn[MAXN], low[MAXN], ins[MAXN], cut[MAXN];
//stack<int>stk;
struct Edge {
        int to, w, next;
} edge[MAXM];
inline void init() {
        memset(first, -1, sizeof(first));
        sign = 0;
}
inline void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign++;
}
inline void init_tarjan() {
        for(int i = 0; i <= n; i++ ) {
                dfn[i] = low[i] = ins[i] = cut[i] = 0;
        }
        indx = scc = 0;
        //while(!stk.empty()) { stk.pop(); }
}
void tarjan(int now, int faz) {
        low[now] = dfn[now] = ++indx;
        int child = 0;
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to;
                if(!dfn[to]) {
                        tarjan(to, faz);
                        low[now] = min(low[now], low[to]);
                        ///非根节点，如果 dfn[to] <= low[to]那么它是割点
                        if(low[to] >= dfn[now] && now != faz) {
                                cut[now] = 1;
                        }
                        if(now == faz) {
                                child++;
                        }
                }
                low[now] = min(low[now], dfn[to]);
        }
        if(child >= 2 && now == faz) { ///根节点有两个以上子树的是割点
                cut[now] = 1;
```

```
	}
}
int main() {
	while(~scanf("%d %d", &n, &m)) {
		init();
		init_tarjan();
		for(int i = 1; i <= m; i++ ) {
			int u, v;
			scanf("%d %d", &u, &v);
			add_edge(u, v, 1);
			add_edge(v, u, 1);
		}
		for(int i = 1; i <= n; i++ ) {
			if(!dfn[i]) {
				tarjan(i, i);
			}
		}
		vector<int>vec;
		for(int i = 1; i <= n; i++ ) {
			if(cut[i]) {
				vec.push_back(i);
			}
		}
		printf("%d\n", vec.size());
		for(int i = 0; i < vec.size(); i++ ) {
			printf("%d ", vec[i]);
		}
	}
	return 0;
}
```

Tarjan 求 LCA

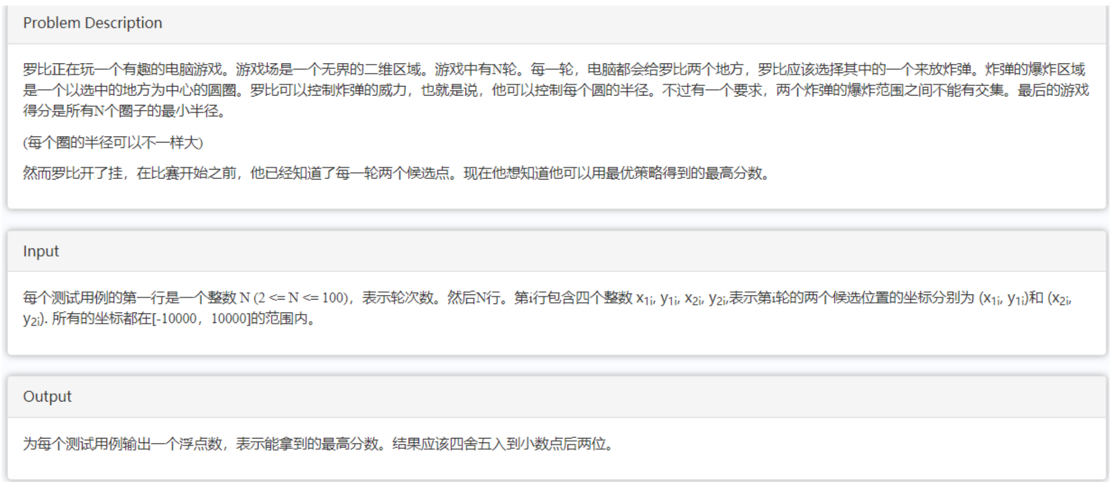
```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 5e4 + 7;
const int INF = 0x3f3f3f3f;
int n, m, first[MAXN], sign;
int pre[MAXN], ans[MAXN], vis[MAXN], dist[MAXN], lca[MAXN], indexs;
vector<pair<int, int> >query[MAXN]; ///y, id
struct Node {
	int to, w, next;
} edge[MAXN * 2];
inline void init() {
    for(int i = 0; i <= n; i++ ) {
        first[i] = -1;
        pre[i] = i;
        query[i].clear();
        ans[i] = INF;
        vis[i] = 0;
    }
    sign = 0;
}
inline void add_edge(int u, int v, int w) {
    edge[sign].to = v;
    edge[sign].w = w;
    edge[sign].next = first[u];
    first[u] = sign++;
}
int findx(int x) {
    return pre[x] == x ? x : pre[x] = findx(pre[x]);
}
inline void join(int x, int y) {
    int fx = findx(x), fy = findx(y);
    pre[fx] = fy;
}
inline bool same(int x, int y) {
    return findx(x) == findx(y);
}
void tarjan(int now) {
    vis[now] = 1;
    for(int i = first[now]; ~i; i = edge[i].next) {
        int to = edge[i].to;
        if(!vis[to]) {
            dist[to] = dist[now] + edge[i].w;
            tarjan(to);
            pre[to] = now;
        }
    }
    for(int i = 0; i < query[now].size(); i++ ) {
        int y = query[now][i].first, id = query[now][i].second;
        if(vis[y] == 2) {
            int lca = findx(y);
            ans[id] = min(ans[id], dist[now] + dist[y] - 2 * dist[lca]);
        }
    }
    vis[now] = 2;
}
```

```
int main() {
	int T;
	scanf("%d", &T);
	while(T--) {
		scanf("%d %d", &n, &m);
		init();
		for(int i = 1; i <= n - 1; i++ ) {
			int u, v, w;
			scanf("%d %d %d", &u, &v, &w);
			add_edge(u, v, w);
			add_edge(v, u, w);
		}
		for(int i = 1; i <= m; i++ ) {
			int x, y;
			scanf("%d %d", &x, &y);
			if(x == y) {
				ans[i] = 0;
				continue;
			}
			query[x].push_back(make_pair(y, i));
			query[y].push_back(make_pair(x, i));
			ans[i] = INF;
		}
		tarjan(1);
		for(int i = 1; i <= m; i++ ) {
			printf("%d\n", ans[i]);
		}
	}
	return 0;
}
```

2-SAT

二分 2-SAT

有 n 个问题，每个问题有两种答案，只能选一个。且 n 个问题直接也有各自的约束关系。比如问题 A 选了一个答案 x，问题 B 的答案 y 就不能选。问是否存在合法情况。约束看成边看是否一个问题的两个决策在同一强连通分量即可。存在则不合法。否则必有合法解。


Problem Description

罗比正在玩一个有趣的电脑游戏。游戏场是一个无界的二维区域。游戏中有N轮。每一轮，电脑都会给罗比两个地方，罗比应该选择其中的一个来放炸弹。炸弹的爆炸区域是一个以选中的地方为中心的圆圈。罗比可以控制炸弹的威力，也就是说，他可以控制每个圆的半径。不过有一个要求，两个炸弹的爆炸范围之间不能有交集。最后的游戏得分是所有N个圈子的最小半径。

(每个圈的半径可以不一样大)

然而罗比开了挂，在比赛开始之前，他已经知道了每一轮两个候选点。现在他想知道他可以用最优策略得到的最高分数。

Input

每个测试用例的第一行是一个整数 N (2 <= N <= 100)，表示轮次数。然后N行。第i行包含四个整数 x_{1i}, y_{1i}, x_{2i}, y_{2i},表示第i轮的两个候选位置的坐标分别为 (x_{1i}, y_{1i})和 (x_{2i}, y_{2i}). 所有的坐标都在[-10000，10000]的范围内。

Output

为每个测试用例输出一个浮点数，表示能拿到的最高分数。结果应该四舍五入到小数点后两位。

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2000;
const int INF = 0x3F3F3F3F;
const double EPS = 1e-6;
typedef long long LL;
int p, n, first[MAXN], sign;
int dfn[MAXN], low[MAXN], ins[MAXN], col[MAXN], indexs, scc;
stack<int>st;
struct Edge {
        int to, w, next;
} edge[MAXN * MAXN];
struct Point {
        double x, y;
} point[MAXN];
void init() {
        memset(first, -1, sizeof(first));
        sign = 0;
}
void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign++;
}
void init_tarjan() {
        memset(dfn, 0, sizeof(dfn));
        memset(low, 0, sizeof(low));
        memset(ins, 0, sizeof(ins));
        memset(col, 0, sizeof(col));
        indexs = scc = 0;
        while(!st.empty()) {
```

```
        st.pop();
    }
}
double dis(const Point &a, const Point &b) {
    return sqrt((a.x-b.x) * (a.x-b.x) + (a.y-b.y) * (a.y-b.y));
}
bool judge(const Point &a, const Point &b, double radius) {
    return dis(a, b) > radius * 2;
}
void tarjan(int x) {
    dfn[x] = low[x] = ++indexs;
    ins[x] = 1;
    st.push(x);
    for(int i = first[x]; ~i; i = edge[i].next) {
        int to = edge[i].to;
        if(!dfn[to]) {
            tarjan(to);
            low[x] = min(low[x], low[to]);
        } else if(ins[to]) {
            low[x] = min(low[x], dfn[to]);
        }
    }
    if(dfn[x] == low[x]) {
        int now = 0;
        scc++;
        do {
            now = st.top();
            st.pop();
            ins[now] = 0;
            col[now] = scc;
        } while(now != x);
    }
}
bool twoSAT(double radius) {
    init();
    for(int i = 1; i <= p; i++ ) {
        for(int j = i+1; j <= p; j++ ) {
            if(i == j) {
                continue;
            }
            int cnt1 = 0, cnt2 = 0;
            if(!judge(point[i], point[j], radius)) {
                add_edge(i, j + p, 1);
                add_edge(j, i + p, 1);
            }
            if(!judge(point[i], point[j + p], radius)) {
                add_edge(i, j, 1);
                add_edge(j + p, i + p, 1);
            }
            if(!judge(point[i + p], point[j], radius)) {
                add_edge(i + p, j + p, 1);
                add_edge(j, i, 1);
            }
            if(!judge(point[i + p], point[j + p], radius)) {
                add_edge(i + p, j, 1);
                add_edge(j + p, i, 1);
            }
        }
    }
    init_tarjan();
    for(int i = 1; i <= 2 * p; i++ ) {
        if(!dfn[i]) {
            tarjan(i);
        }
    }
    for(int i = 1; i <= p; i++ ) {
        if(col[i] == col[i + p]) {
            return false;
        }
    }
    return true;
}
int main() {
    while(~scanf("%d", &p)) {
        for(int i = 1; i <= p; i++ ) {
            scanf("%lf %lf", &point[i].x, &point[i].y);
            scanf("%lf %lf", &point[i + p].x, &point[i + p].y);
        }
        double l = 0, r = 1e9;
        while(l + EPS < r) {
            double mid = (l + r) / 2;
            if(twoSAT(mid)) {
                l = mid;
            } else {
                r = mid;
            }
        }
        printf("%.2f\n", r);
```

```
	}
	return 0;
}
```

2-SAT 记录结果

> 有n个布尔变量x_1~x_n，另有m个需要满足的条件，每个条件的形式都是"x_i为true/false或x_j为true/false"。比如"x_1为真或x_3为假"、"x_7为假或x_2为假"。2-SAT 问题的目标是给每个变量赋值使得所有条件得到满足。

```
///luogu 4782 2-sat + road
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2e6 + 10;
int n, m, first[MAXN], sign;
struct Edge {
	int to, w, next;
} edge[MAXN * 4];
int dfn[MAXN], low[MAXN], ins[MAXN], col[MAXN], indexs, scc;
stack<int>st;
void init() {
	memset(first, -1, sizeof(first));
	sign = 0;
	indexs = scc = 0;
	memset(dfn, 0, sizeof(dfn));
	memset(low, 0, sizeof(low));
	memset(ins, 0, sizeof(ins));
	memset(col, 0, sizeof(col));
	while(!st.empty()) {
		st.pop();
	}
}
void add_edge(int u, int v, int w) {
	edge[sign].to = v;
	edge[sign].w = w;
	edge[sign].next = first[u];
	first[u] = sign++;
}
void tarjan(int x) {
	low[x] = dfn[x] = ++indexs;
	st.push(x);
	ins[x] = 1;
	for(int i = first[x]; ~i; i = edge[i].next) {
		int to = edge[i].to;
		if(!dfn[to]) {
```

```
            tarjan(to);
            low[x] = min(low[x], low[to]);
        } else if(ins[to]) {
            low[x] = min(low[x], dfn[to]);
        }
    }
    if(low[x] == dfn[x]) {
        int top = 0;
        scc++;
        do {
            top = st.top();
            st.pop();
            ins[top] = 0;
            col[top] = scc;
        } while(top != x);
    }
}
bool two_sat() {
    for(int i = 1; i <= 2 * n; i++ ) {
        if(!dfn[i]) {
            tarjan(i);
        }
    }
    for(int i = 1; i <= n; i++ ) {
        if(col[i] == col[i + n]) {
            return false;
        }
    }
    return true;
}
int main() {
    scanf("%d %d", &n, &m);
    init();
    for(int i = 1; i <= m; i++ ) {
        int a, b, a_val, b_val;
        scanf("%d %d %d %d", &a, &a_val, &b, &b_val);
        int not_a = a_val ^ 1, not_b = b_val ^ 1;
        add_edge(a + not_a * n, b + b_val * n, 1);
        add_edge(b + not_b * n, a + a_val * n, 1);
    }
    if(two_sat()) {
        puts("POSSIBLE");
        for(int i = 1; i <= n; i++ ) {
            printf("%d ", col[i] > col[i + n]);
        }
    } else {
        puts("IMPOSSIBLE");
    }
    return 0;
}
```

带花树一般图匹配

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 555;
const int MAXM = MAXN * MAXN * 2;
int n, m, pre[MAXN], tim;
int first[MAXN], sign;
int match[MAXN], tp[MAXN], tic[MAXN];
queue<int>que;
struct Edge {
    int to, w, next;
} edge[MAXM * 4];
void init() {
    memset(first, -1, sizeof(first));
    sign = 0;
}
void add_edge(int u, int v, int w) {
    edge[sign].to = v;
    edge[sign].w = w;
    edge[sign].next = first[u];
    first[u] = sign++;
}
struct Disjoint_Set {
    int pre[MAXN];
    void init(int n) {
        for(int i = 1; i <= n; i++ ) {
            pre[i] = i;
        }
    }
    int findx(int x) {
        return pre[x] == x ? x : pre[x] = findx(pre[x]);
    }
    void join(int x, int y) {
        int fx = findx(x), fy = findx(y);
        if(fx != fy) {
            pre[fx] = fy;
        }
    }
    bool same(int x, int y) {
        return findx(x) == findx(y);
    }
} disjoint;
int lca(int x, int y) {
    for(tim++; ; swap(x, y)) {
        if(x) {
            x = disjoint.findx(x);
            if(tic[x] == tim) {
                return x;
            } else {
                tic[x] = tim, x = pre[ match[x] ];
            }
        }
    }
}

void shrink(int x, int y, int p) {
    while(disjoint.findx(x) != p) {
        pre[x] = y, y = match[x];
        if(tp[y] == 2)
            tp[y] = 1, que.push(y);
        if(disjoint.findx(x) == x)
            disjoint.join(x, p);
        if(disjoint.findx(y) == y)
            disjoint.join(y, p);
        x = pre[y];
    }
}
bool flower(int x) {
    disjoint.init(n);
    memset(tp, 0, sizeof(tp));
    memset(pre, 0, sizeof(pre));
    while(!que.empty()) {
        que.pop();
    }
    que.push(x);
    tp[x] = 1;
    while(!que.empty()) {
        int now = que.front();
        que.pop();
```

```
		for(int i = first[now]; ~i; i = edge[i].next) {
			int to = edge[i].to;
			if(disjoint.same(to, now) || tp[to] == 2) {
				continue;
			}
			if(!tp[to]) {
				tp[to] = 2, pre[to] = now;
				if(!match[to]) {
					for(int x = to, y, z; x; x = z) {
						z = match[ y = pre[x] ];
						match[x] = y;
						match[y] = x;
					}
					return true;
				}
				tp[match[to]] = 1;
				que.push(match[to]);
			} else if(tp[to] == 1) {
				int id = lca(now, to);
				shrink(now, to, id);
				shrink(to, now, id);
			}
		}
	}
	return false;
}
int main() {
	scanf("%d %d", &n, &m);
	init();
	for(int i = 1; i <= m; i++ ) {
		int u, v;
		scanf("%d %d", &u, &v);
		add_edge(u, v, 1), add_edge(v, u, 1);
	}
	int ans = 0;
	for(int i = 1; i <= n; i++ ) {
		ans += (!match[i] && flower(i));
	}
	printf("%d\n", ans);
	for(int i = 1; i <= n; i++ ) {
		printf("%d ", match[i]);
	}
	puts("");
	return 0;
```

```
}
```

树上倍增 LCA

倍增最短路 **HDU2586**

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 4e4 + 7;
int T, n, m, u, v, w;
int first[maxn], sign, st[maxn][21], level[maxn], dist[maxn];
struct Node {
        int to, w, next;
} edge[maxn * 2];
inline void init() {
        for(int i = 0; i <= n; i ++ ) {
                first[i] = -1;
        }
        sign = 0;
}
inline void add_edge(int u, int v, int w) {
        edge[sign].to = v;
        edge[sign].w = w;
        edge[sign].next = first[u];
        first[u] = sign ++;
}
void dfs(int now, int father) {
        level[now] = level[father] + 1;
        st[now][0] = father;
        for(int i = 1; (1 << i) <= level[now]; i ++ ) {
                st[now][i] = st[ st[now][i - 1] ][i - 1];
        }
        for(int i = first[now]; ~i; i = edge[i].next) {
                int to = edge[i].to, w = edge[i].w;
                if(to == father) {
                        continue;
                }
                dist[to] = dist[now] + w;
                dfs(to, now);
        }
}
int LCA(int x, int y) {
        if(level[x] > level[y]) {
```

```
        swap(x, y);
    }
    for(int i = 20; i >= 0; i -- ) {
        if(level[x] + (1 << i) <= level[y]) {
            y = st[y][i];
        }
    }
    if(x == y) {
        return x;
    }
    for(int i = 20; i >= 0; i -- ) {
        if(st[x][i] == st[y][i]) {
            continue;
        } else {
            x = st[x][i], y = st[y][i];
        }
    }
    return st[x][0];
}
int main() {
    scanf("%d", &T);
    while(T--) {
        scanf("%d %d", &n, &m);
        init();
        for(int i = 1; i <= n - 1; i ++ ) {
            scanf("%d %d %d", &u, &v, &w);
            add_edge(u, v, w);
            add_edge(v, u, w);
        }
        memset(level, 0, sizeof(level));
        memset(st, 0, sizeof(st));
        dist[1] = 0;
        dfs(1, 0);
        for(int i = 1; i <= m; i ++ ) {
            int u, v;
            scanf("%d %d", &u, &v);
            int lca = LCA(u, v);
            printf("%d\n", dist[u] + dist[v] - 2 * dist[lca]);
        }
    }
    return 0;
}
```

数论

快速傅里叶变换 FFT

```
///离散式快速傅里叶变换求 f(x), g(x)的卷积
/**
求多项式  f(x), g(x)的卷积。
1，分别对 f(x), g(x)  做傅里叶变换，使之从多项式系数表达式转换为点值表达式
2，对点值表达式做  逆傅里叶变换。求的多项式系数表达式，即答案卷积
*/
#include <bits/stdc++.h>
using namespace std;
const double PI = acos(-1.0);
struct FFT {
	static const int MAXN = 1e7 + 10;
	struct Complex {
		double x, y;
		Complex (double xx = 0, double yy = 0) { x = xx, y = yy; }
		friend Complex operator + (Complex a, Complex b) { return Complex(a.x + b.x, a.y + b.y); }
		friend Complex operator - (Complex a, Complex b) { return Complex(a.x - b.x, a.y - b.y); }
		friend Complex operator * (Complex a, Complex b) { return Complex(a.x * b.x - a.y * b.y ,
a.x * b.y + a.y * b.x); }

	} a[MAXN], b[MAXN];
	int n, m, rev[MAXN], limit, bit;
	void fft(Complex *arr, int type) {
		for(int i = 0; i < limit; i++ ) {
			if(i < rev[i]) {
				swap(arr[i], arr[ rev[i] ]);
			}
		}
		for(int mid = 1; mid < limit; mid <<= 1) { ///mid 区间半长
			Complex wn( cos(PI / mid), type * sin(PI / mid) );
			for(int r = mid << 1, l = 0; l < limit; l += r) { ///r 区间长度  [l, mid, r]
				Complex w(1, 0);
				for(int k = 0; k < mid; k++, w = w * wn) { ///玄学蝴蝶变换
					Complex x = arr[l + k], y = w * arr[l + mid + k];
					arr[l + k] = x + y;
					arr[l + mid + k] = x - y;
				}
			}
		}
```

```
        }
        void init() {
                scanf("%d %d", &n, &m);
                for(int i = 0; i <= n; i++ ) { scanf("%lf", &a[i].x); }
                for(int i = 0; i <= m; i++ ) { scanf("%lf", &b[i].x); }
                for(limit = 1, bit = 0; limit <= n + m; limit <<= 1) { bit++; }
                for(int i = 0; i < limit; i++ ) { ///某蒙蔽蝴蝶变换
                        rev[i] = (rev[i >> 1] >> 1) | ( (i & 1) << (bit - 1) );
                }
                fft(a, 1), fft(b, 1);
                for(int i = 0; i <= limit; i++) {
                        a[i] = a[i] * b[i];
                }
                fft(a, -1);
                for(int i = 0; i <= n+ m; i++ ) {
                        printf("%d ", (int)(a[i].x / limit + 0.5));
                }
        }
} cwl;
```

快速数论变换 NTT

```
///快速数论变换 NTT 求 f(x),g(x)的卷积
struct NTT {
        /// MOD  是质数
        /// G 是  MOD  的原根, GI 是原根的逆
        static const int MAXN = 3e6 + 10;
        static const int MOD = 998244353;
        static const int G = 3, Gi = 332748118;
        int n, m, limit, bit, rev[MAXN];
        long long a[MAXN], b[MAXN];
        inline LL pow_mod(LL a, LL b, LL c) {
                LL ans = 1;
                while(b) {
                        if(b & 1) {
                                ans = ans * a % c;
                        }
                        a = a * a % c;
                        b >>= 1;
                }
                return ans;
        }
        void ntt(LL *arr, int type) {
```

```
        for(int i = 0; i < limit; i++ ) {
            if(i < rev[i]) {
                swap(arr[i], arr[ rev[i] ]);
            }
        }
        for(int mid = 1; mid < limit; mid <<= 1) {
            LL wn = pow_mod(type == 1 ? G : Gi, (MOD - 1) / (mid << 1), MOD);
            for(int j = 0; j < limit; j += (mid << 1)) {
                LL w = 1;
                for(int k = 0; k < mid; k++, w = (w * wn) % MOD) {
                    int x = arr[j + k], y = w * arr[j + k + mid] % MOD;
                    arr[j + k] = (x + y) % MOD;
                    arr[j + k + mid] = (x - y + MOD) % MOD;
                }
            }
        }
    }
    void init() {
        scanf("%d %d", &n, &m);
        for(int i = 0; i <= n; i++ ) {
            scanf("%lld", &a[i]), a[i] = (a[i] + MOD) % MOD;
        }
        for(int i = 0; i <= m; i++ ) {
            scanf("%lld", &b[i]), b[i] = (b[i] + MOD) % MOD;
        }
        for(limit = 1, bit = 0; limit <= n + m; limit <<= 1) { bit++; }
        for(int i = 0; i < limit; i++ ) {
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << bit - 1);
        }
        ntt(a, 1), ntt(b, 1);
        for(int i = 0; i < limit; i++ ) {
            a[i] = (a[i] * b[i]) % MOD;
        }
        ntt(a, -1);
        LL inv = pow_mod(limit, MOD - 2, MOD);
        for(int i = 0; i <= n + m; i++ ) {
            printf("%lld ", (a[i] * inv) % MOD);
        }
    }
} cwl;
```

NTT 常用原根表（g 是 mod(r * 2 ^ k + 1)的原根）

r * 2 ^ k + 1	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22

2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

快速沃尔什变换 FWT

```
struct FWT {

    static const int MAXN = (1 << 17) + 5;
    static const int MOD = 998244353;

    void fwt_or(LL a[], int n, int opt) {
        for(int i = 1; i < n; i <<= 1) {
            for(int p = i << 1, j = 0; j < n; j += p) {
                for(int k = 0; k < i; k++ ) {
                    if(opt == 1) {
                        a[i + j + k] = (a[j + k] + a[i + j + k]) % MOD;
                    } else {
                        a[i + j + k] = (a[i + j + k] + MOD - a[j + k]) % MOD;
                    }
                }
            }
        }
    }

    void fwt_and(LL a[], int n, int opt) {
        for(int i = 1; i < n; i <<= 1) {
            for(int p = i << 1, j = 0; j < n; j += p) {
                for(int k = 0; k < i; k++ ) {
                    if(opt == 1) {
                        a[j + k] = (a[j + k] + a[i + j + k]) % MOD;
                    } else {
                        a[j + k] = (a[j + k] + MOD - a[i + j + k]) % MOD;
                    }
                }
            }
        }
    }

    void fwt_xor(LL a[], int n, int opt) {
        int inv2 = inv(2, MOD);
        for(int i = 1; i < n; i <<= 1) {
            for(int p = i << 1, j = 0; j < n; j += p) {
                for(int k = 0; k < i; k++ ) {
                    int x = a[j + k], y = a[i + j + k];
                    a[j + k] = (x + y) % MOD;
                    a[i + j + k] = (x + MOD - y) % MOD;
                    if(opt == -1) {
                        a[j + k] = 1LL * a[j + k] * inv2 % MOD;
                        a[i + j + k] = 1LL * a[i + j + k] * inv2 % MOD;
                    }
                }
            }
        }
    }
} cwl;
```

辛普森求积分

```
#include <bits/stdc++.h>
using namespace std;
```

```
int a,b,l,r;
double fun(double x) {
	return b*sqrt(1-x*x/a/a);
}
double Simpson(double l,double r) {
	return (fun(l)+fun(r)+4*fun((l+r)/2))/6*(r-l);
}
double asr(double l,double r,double eps,double last) {
	double mid = (l + r) / 2, a, b, ans;
	a = Simpson(l,mid);
	b = Simpson(mid,r);
	ans = a + b;
	if(fabs(last-ans) > eps) {
		ans = asr(l,mid,eps/2,a) + asr(mid,r,eps/2,b);
	}
	return ans;
}
int main() {
	int t,n,l,r;
	scanf("%d",&t);
	while(t--) {
		scanf("%d %d %d %d",&a,&b,&l,&r);
		printf("%.3f\n",2*asr(l,r,1e-5,Simpson(l,r)));
	}
	return 0;
}
```

卢卡斯定理求组合数

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
LL n,m,p;
LL quick_mod(LL a, LL b) {
	LL ans = 1;
	a %= p;
	while(b) {
		if(b & 1) {
			ans = ans * a % p;
		}
		b >>= 1;
		a = a * a % p;
	}
```

```
    return ans;
}
LL C(LL n, LL m) {
    if(m > n)
        return 0;
    LL ans = 1,a=1,b=1;
    for(int i=1; i<=m; i++) {
        a = a*(n + i - m) % p;
        b = b*i % p;
    }
    ans = (a * quick_mod(b, p-2) % p) % p;
    return ans;
}
LL Lucas(LL n, LL m) {
    if(m == 0)
        return 1;
    return C(n % p, m % p) * Lucas(n / p, m / p) % p;
}
int main() {
    int t;
    scanf("%d",&t);
    while(t--) {
        cin>>n>>m>>p;
        cout<<Lucas(n+m,m)<<endl;
    }
}
```

扩展欧几里得

```
LL gcd(LL a, LL b) {
    return b ? gcd(b, a % b) : a;
}
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y) {
    if(!b) {
        d = a, x = 1, y = 0;
    } else {
        exgcd(b, a%b, d, y, x);
        y -= x * (a / b);
    }
}
```

一元线性同余方程

```
///ax%b=c
#include <stdio.h>
typedef long long ll;
void exgcd(ll a, ll b, ll &d, ll &x, ll &y) {
	if (!b) {
		d = a, x = 1, y = 0;
	}
	else {
		exgcd(b, a%b, d, y, x);
		y -= x*(a/b);
	}
}
ll solve_eequation(ll a,ll mod,ll rem) { ///常系数，模，余数
	ll d,x,y,ans,x0,t;
	///标准化处理,保证所有数非负
	a=(a%mod+mod)%mod;
	rem=(rem%mod+mod)%mod;
	exgcd(a,mod,d,x,y);
	if(rem%d!=0)
		return -1;///无解
	else {
		x0=rem/d*x;///求特解 x0
		t=mod/d;///计算周期
		ans=(x0%t+t)%t;;///求最小非负解
		return ans;
	}
}
int main() {
	ll x,y,n,m,l,ans,a,b;
	while(scanf("%lld%lld%lld%lld%lld",&x,&y,&m,&n,&l)!=EOF) {
		b=y-x;///求余数
		a=m-n;///计算常系
		ans=solve_eequation(a,l,b);
		if(ans==-1)
			puts("Impossible");
		else
			printf("%lld\n",ans);

	}
}
```

快速幂

```
ll power_mod(ll a,ll b,ll c) {
	ll res=0;
	a=a%c;
	while(b>0) {
		if(b&1==1) {
			res=(res+a%c)%c;
		}
		a=(a%c+a%c)%c;
		b>>=1;
	}
	return res;
}
ll Power_Mod(ll a, ll b, ll c) {
	ll ans = 1;
	a=a%c;
	while(b>0) {
		if(b&1==1)
			ans=power_mod(ans,a,c);
		b>>=1;
		a=power_mod(a,a,c);
	}
	return ans;
}
```

矩阵快速幂模板

```
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
const int maxn = 110;
const int MOD = 1e9 + 7;
#define mod(x) ((x)%MOD)
const int n = 3;
struct mat {
	LL m[maxn][maxn];
	mat() {
		for(int i = 0; i < n; i ++ ) {
			for(int j = 0; j < n; j ++ ) {
				m[i][j] = 0;
			}
		}
```

```
        }
} unit;
mat operator * (mat a,mat b) {
        mat ret;
        LL x;
        for(int i = 0; i < n; i++ ) {
                for(int j = 0; j < n; j++ ) {
                        x = 0;
                        for(int k = 0; k < n ; k++ ) {
                                x += mod( a.m[i][k] * b.m[k][j] );
                        }
                        ret.m[i][j] = mod(x);
                }
        }
        return ret;
}
void init_unit() {
        for(int i = 0; i < maxn; i++) {
                unit.m[i][i] = 1;
        }
}
mat pow_mat(mat a, LL n) {
        mat ret = unit;
        while(n) {
                if(n & 1) {
                        ret = ret * a;
                }
                a = a * a;
                n >>= 1;
        }
        return ret;
}
int main() {
        LL x;
        int T;
        init_unit();
        scanf("%d", &T);
        while(T--) {
                scanf("%lld", &x);
                mat a, b;
                a.m[0][0] = a.m[0][1] = a.m[0][2] = 1;
                b.m[1][0] = b.m[2][1] = b.m[0][2] = b.m[2][2] = 1;
                a = a * pow_mat(b, x - 1);
                printf("%lld\n", a.m[0][0]);
        }
        return 0;
}
```

常见同余公式

定义：$a\%\text{mod}=q \Leftrightarrow a=\text{k}*\text{mod}+q,\text{其中}0<=q<\text{mod}$

公式 1(加法)：

$$(a+b)\%\text{mod}=((a\%\text{mod})+(b\%\text{mod}))\%\text{mod}$$

证明：

$$\text{设：}\ \text{a}=k_1*\text{mod}+q_1,b=k_2*\text{mod}+q_2$$
$$(a+b)\%\text{mod}=[(k_1+k_2)*\text{mod}+(q1+q2)]\%\text{mod}$$
$$=(q1+q2)\%\text{mod}=(a\%\text{mod}+b\%\text{mod})\%\text{mod}$$

公式 2(乘法)：

$$a*b\%\text{mod}=(a\%\text{mod})*(b\%\text{mod})\%\text{mod}$$

证明：

$$\text{设：}\ \text{a}=k_1*\text{mod}+q_1,b=k_2*\text{mod}+q_2$$
$$(a*b)\%\text{mod}=[(k_1*k_2)*\text{mod}^2+(k1+k2)*\text{mod}+q1*q2]\%\text{mod}$$
$$=q1*q2\%\text{mod}=(a\%\text{mod})*(b\%\text{mod})\%\text{mod}$$

公式 3(除法 1)：

$$\text{若a}\%b=0(\text{记作b}\,|\,a,\text{读作b整除a})，\text{则}(\text{a}/b)\%\text{mod}=[a\%(b*\text{mod})]/b$$

证明：

$$\text{设：}\ a/b=k*\text{mod}+q$$
$$\text{则}:a=k*(b*\text{mod})+b*q \Leftrightarrow a\%(b*\text{mod})=bq;$$
$$\therefore (a/b)\%\text{mod}=q=[a\%(b*\text{mod})]/b$$

适用范围：分母较小时，即 **b*mod<=1e18**

公式 4(幂运算 1)：

若p为质数,则

$a^b \% p = a^{(b-1)\%(p-1)+1} \% p$

或者$a^b \% p \begin{cases} a^{b\%(p-1)} & a\%p \neq 0 \\ 0 & a\%p = 0 \end{cases}$

证明：

由费马小定理得$a^{p-1}\%p = 1$,即$a^b\%p$以$p-1$为周期

设：$b = k(p-1)+q$

则$a^b\%p = a^{k(p-1)+q} = (a^q * \prod_{i=1}^{k} a^{p-1})\%p$

$= [(a^q\%p) * \prod_{i=1}^{k}(a^{p-1}\%p)]\%p$

$= [(a^q\%p) * \prod_{i=1}^{k} 1]\%p = (a^q\%p) = a^{b\%(p-1)}\%p$

适用范围：模为质数

公式 5(除法 2):

若p为质数，$b\%p \neq 0$且$a\%b = 0$

则$(a/b)\%p = a * b^{-1}\%p$

其中$b^{-1} = b^{p-2}\%p$,称为b在模p意义下的逆元

证明：

设$c = a/b = k*p+q$

则$a = c*b = b*k*p + b*q$

$\therefore a*b^{p-2} = b^{p-1}*k*p + b^{p-1}*q$

$\therefore a*b^{p-2}\%p = b^{p-1}*q\%p = (b^{p-1}\%p)*(q\%p)\%p = 1*(q\%p)\%p = (a/b)\%p$

适用范围：模为质数

公式 6(幂运算 2):

则$a^{b}\%c = a^{b\%\varphi(c)+\varphi(c)}\%c$

其中$\varphi(x)$,为欧拉函数。

若a与c互质则，$a^{\varphi(c)}\%c = 1$（欧拉定理）

特别的,当p为质数时，则$\varphi(p) = p-1$，即$a^{p-1}\%p = 1$（费马定理）

证明：略（因为需要讨论的情况比较多）

适用范围：模非质数

扩展欧几里得

```
typedef long long LL;
LL exgcd(LL a,LL b,LL &d,LL &x,LL &y) {
        if(!b) {
                d=a,x=1,y=0;
        } else {
                exgcd(b,a%b,d,y,x);
                y-=x*(a/b);
        }
}
```

欧拉函数

定义：表示区间 [1,n]中与 n 互质的数的个数

1. 欧拉函数是不完全积性函数 phi(nm) == phi(n) * phi(m) 只在 gcd(n, m) == 1 时成立
2. 若存在 N 的唯一分解 N = pai{ ai ^ pi } ps:pai 是那个连乘符号(囧).

则 phi(n) = n * pai{ 1 - 1 / pi }

3. 除了 n == 2 其他 phi(n)都是偶数
4. 设 n 为正数，sigma { phi(d) == n } (d | n)

求单个 phi(data)根据上述定义,对每一项(1 - 1 / pi) 进行通分，(pi-1)/pi

欧拉打表

//O(nlogn)

```
LL Euler[MAXN];
void pre_phi() {
	for(int i = 1; i < MAXN; i ++ ) {
		Euler[i] = i;
	}
	for(int i = 2; i < MAXN; i ++ ) {
		if(Euler[i] == i) {
			for(int j = i; j < MAXN; j += i) {
				Euler[j] = Euler[j] - Euler[j] / i;
				///the same as Euler[j] * (i - 1) / i
			}
		}
	}
}
//O(n)
const int maxn = 3e6 + 7;
bool vis[maxn];
int prime[maxn / 10];
int tot;
LL euler[maxn];
void init() {
	euler[1] = 1;
	for (int i = 2; i < maxn; i++) {
		if (!vis[i]) {
			prime[tot++] = i;
			euler[i] = i - 1;
			//素数 p 的欧拉函数等于 p - 1
		}
		for (int j = 0; prime[j] * i < maxn; j++) {
			vis[prime[j] * i] = 1;
			if (i % prime[j] == 0) {
				euler[i * prime[j]] = euler[i] * prime[j];
				break;
			} else {
				euler[i * prime[j]] = euler[i] * (prime[j] - 1);
			}
		}
	}
}
```

求单个数的欧拉值

```
inline int phi(int n) {
	int Euler = n;
	for(int i = 2; i * i <= n; i ++ ) {
		if(n % i == 0) {
			Euler = Euler / i * (i - 1);
			while(n % i == 0) {
				n /= i;
			}
		}
	}
	if(n > 1) {
		Euler = Euler / n * (n - 1);
	}
	return Euler;
}
```

素数筛

埃拉托斯特尼筛法

```
const int maxn = 1e7 + 7;
int flag[maxn], prime[maxn], tot;
void init() {
	int sq_n = sqrt(maxn);
	for(int i = 2; i <= sq_n; i ++ ) {
		if(!flag[i]) {
			for(int j = i * i; j <= maxn; j += i) {
				flag[j] = 1;
				prime[tot++] = j;
			}
		}
	}
}
```

欧拉筛素数

```
int flag[maxn], prime[maxn], tot;
void Euler_pre_solve() {
	tot = 0;
	for(int i = 2; i < maxn; i ++ ) {
		if(!flag[i]) {
			prime[tot++] = i;
		}
		for(int j = 0; prime[j] * i < maxn; j ++ ) {
```

```
                flag[prime[j] * i] = 1;
                if(i % prime[j] == 0) {
                    break;
                }
            }
        }
}
```

米勒罗宾

```
// 18 位素数：154590409516822759
// 19 位素数：2305843009213693951 (梅森素数)
// 19 位素数：4384957924686954497
LL prime[6] = {2, 3, 5, 233, 331};

LL qmul(LL x, LL y, LL mod) { // 乘法防止溢出， 如果 p * p 不爆 LL 的话可以直接乘； O(1)乘法或者转化成二进制加法
    return (x * y - (long long)(x / (long double)mod * y + 1e-3) *mod + mod) % mod;
    /*
    LL ret = 0;
    while(y) {
        if(y & 1)
            ret = (ret + x) % mod;
        x = x * 2 % mod;
        y >>= 1;
    }
    return ret;
    */
}

LL qpow(LL a, LL n, LL mod) {
    LL ret = 1;
    while(n) {
        if(n & 1) { ret = qmul(ret, a, mod); }
        a = qmul(a, a, mod);
        n >>= 1;
    }
    return ret;
}

bool Miller_Rabin(LL p) {
    if(p < 2) return 0;
    if(p != 2 && p % 2 == 0) return 0;
    LL s = p - 1;
    while(! (s & 1)) s >>= 1;
    for(int i = 0; i < 5; ++i) {
        if(p == prime[i]) return 1;
        LL t = s, m = qpow(prime[i], s, p);
        while(t != p - 1 && m != 1 && m != p - 1) {
            m = qmul(m, m, p);
            t <<= 1;
        }
        if(m != p - 1 && !(t & 1)) return 0;
    }
    return 1;
}
```

斯特灵树求 n 阶乘长度

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const double PI = acos(-1);
const double E   = exp(1.0);
int t;
ll n,res;
int main() {
    scanf("%d",&t);
    while (t--) {
        cin >> n;
        res =   0.5 * log10(2.0 * PI * n) + n * log10(n * 1.0 / E) + 1;
        cout << res << endl;
    }
    return 0;
}
```

求逆元

扩展欧几里得求逆元

```
void exgcd(LL a, LL b, LL &d, LL &x, LL &y) {
    if(!b) { d = a, x = 1, y = 0; }
    else {
        exgcd(b, a % b, d, y, x);
        y -= x * (a / b);
    }
}

LL inv(LL a, LL mod, LL x = 0, LL y = 0, LL d = 0) {
    exgcd(a, mod, d, x, y);
    return (x + mod) % mod;
}
```

快速幂求逆元

```
//快速幂求逆元
LL pow_mod(LL a, LL b, LL c) {
    LL res = 1;
    while(b) {
        if(b & 1) { res = res * a % c; }
        a = a * a % c;
        b = b >> 1;
    }
    return res;
}

LL inv(LL a, LL mod) {
    return pow_mod(a, mod - 2, mod);
}
```

线性逆元打表

```
const int MAXN = 3e6 + 10;

LL inv[MAXN];

void get_inv(LL n, LL mod) {
    inv[1] = 1;
    for(int i = 2; i <= n; i++ ) {
        inv[i] = (mod - mod / i) * inv[ mod % i ] % mod;
    }
}
```

几何

凸包格雷姆扫描

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 10005;
const double PI = acos(-1.0);
int n;
struct Node {
    double x, y;
    Node() {}
    Node(double xx, double yy):x(xx), y(yy) {}
} p[maxn], P[maxn];
int tot;
double ans, R;
inline double getDist(Node a, Node b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
inline double X(Node a, Node b, Node c) {
    return (b.x-a.x)*(c.y-a.y)-(c.x-a.x)*(b.y-a.y);
}
bool cmp(const Node &a, const Node &b) {
    double xj = X(p[0], a, b);
    if(xj > 0) {
        return 1;
    }
    if(xj < 0) {
        return 0;
    }
    return getDist(p[0], a) < getDist(p[0], b);
}
inline void GrahamScan() {
    for(int i = 0; i < n; i ++ ) {
        if(p[i].y < p[0].y) {
            swap(p[i], p[0]);
        } else if(p[i].y == p[0].y && p[i].x < p[0].x) {
            swap(p[i], p[0]);
        }
    }
    sort(p + 1, p + n, cmp);
    tot = 1;
```

```
	P[0] = p[0];
	P[1] = p[1];
	for(int i = 2; i < n; i ++ ) {
		while(tot > 0 && X(P[tot - 1], P[tot], p[i]) <= 0) {
			tot --;
		}
		tot++;
		P[tot] = p[i];
	}
}
int main() {
	while(~scanf("%d", &n)) {
		for(int i = 0; i < n; i ++ ) {
			scanf("%lf %lf", &p[i].x, &p[i].y);
		}
		GrahamScan();
		for(int i = 1; i <= tot; i ++ ) {
			P[i].x -= P[0].x;
			P[i].y -= P[0].y;
		}
		P[0].x = P[0].y = 0;
		double ans = 0;
		for(int i = 2; i <= tot; i ++ ) {
			double x1 = P[i].x;
			double y1 = P[i].y;
			double x2 = P[i - 1].x;
			double y2 = P[i - 1].y;
			ans += (x1 * y2 - x2 * y1) / 2.0;
		}
		printf("%.2f\n", fabs(ans));
	}

	return 0;
}
```

分治二维平面最近点对

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;
const double inf = 1e20;
const int maxn = 1e5 + 7;
int n;
struct Point {
        double x, y;
        friend double dis(const Point &a, const Point &b) {
                return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
        }
} p[maxn];
bool cmpxy(const Point &a, const Point &b) {
        if(a.x == b.x) {
                return a.y < b.y;
        }
        return a.x < b.x;
}
bool cmpy(const Point &a, const Point &b) {
        return a.y < b.y;
}
double solve(int l, int r) {
        double d = inf;
        if(l == r) {
                return d;
        }
        if(l + 1 == r) {
                return dis(p[l], p[r]);
        }
        int mid = (l + r) >> 1;
        double d1 = solve(l, mid);
        double d2 = solve(mid + 1, r);
        d = min(d1, d2);
        vector<Point>v;
        for(int i = l; i <= r; i ++ ) {
                if(fabs(p[mid].x - p[i].x) <= d) {
                        v.push_back(p[i]);
                }
        }
        sort(v.begin(), v.end(), cmpy);
        for(int i = 0; i < v.size(); i ++ ) {
                for(int j = i + 1; j < v.size() && v[j].y - v[i].y < d; j ++ ) {
                        d = min(d, dis(v[i], v[j]));
                }
        }
        return d;
}
int main() {
	while(~scanf("%d", &n) && n) {
		for(int i = 0; i < n; i ++ ) {
			scanf("%lf %lf", &p[i].x, &p[i].y);
		}
		sort(p, p + n, cmpxy);
		printf("%.2f\n", solve(0, n - 1) / 2.0);
	}
	return 0;
}
```

杂项

读入优化

```
template<class T>
inline bool nextInt(T &n) {
	T x = 0, tmp = 1; char c = getchar();
	while((c < '0' || c > '9') && c != '-' && c != EOF) c = getchar();
	if(c == EOF) return false;
	if(c == '-') c = getchar(), tmp = -1;
	while(c >= '0' && c <= '9') x *= 10, x += (c - '0'), c = getchar();
	n = x*tmp; return true;
}

template<class T>
inline void Out(T n) {
	if(n < 0) { putchar('-'); n = -n; }
	int len = 0, data[20];
	while(n) { data[len++] = n%10; n /= 10; }
	if(!len) data[len++] = 0;
	while(len--) putchar(data[len]+48);
}
void nextInt(int &x) {
	char c; do c=getchar();
	while (c<'0'||c>'9'); x=c-'0';
	while ('0'<=(c=getchar())&&c<='9') x=x*10+c-'0';
}

void Out(int a) {
	if(a>9) Out(a/10);
	putchar(a%10+'0');
}
```

PBDS

```
//红黑树
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <bits/stdc++.h>

using namespace __gnu_pbds;
using namespace std;
typedef tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update> rbtree;

/// int 类型
/// null_type 为映射类型, 低版本 g++为  null_mapped_type
/// less<int>, greater<int>  比较器
/// rb_tree_tag  和  splay_tree_tag  选择树的类型
/// tree_order_statistics_node_update  结点更新

/// insert, erase
/// order_of_key rank
/// find_by_order() kth
/// lower_bound()  前继，   >=x  最小的迭代器
/// upper_bound()  后继      >x    最小的迭代器
/// a.join(b) b 并入 a，前提是两颗树的取值范围不相交
/// a.split(v, b) key <= v 的属于 a，其他属于
///  注意，插入的元素会去重，如 set

//hash
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
cc_hash_table<int, bool> h1; //拉链法处理冲突
gp_hash_table<int, bool> h2; //探测法处理冲突

//可并堆，注意 c++stl 更快,pbds 注意是实习了合并功能
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <bits/stdc++.h>

using namespace __gnu_pbds;
using namespace std;

__gnu_pbds::priority_queue<int, less<int>, pairing_heap_tag> que;
```

Vimrc

```
set shiftwidth=4
set softtabstop=4
"set encoding=utf-8
set autoindent
set nu

"colorscheme desert "配色  // evening
set background=dark

syntax on

set vb "用闪烁代替响铃
set cursorline "突出当前行
set guifont=Courier_new:h12:b:cDEFAULT "设置字体
set autowrite

"inoremap ( ()<ESC>i
"inoremap [ []<ESC>i
inoremap { {}<ESC>i
"inoremap < <><ESC>i
"inoremap " ""<ESC>i
"inoremap ' ''<ESC>i

map <F9> :call CompileRunGcc()<CR>

func! CompileRunGcc()
        exec "w"
        if &filetype == 'cpp'
                exec "!g++ -g -Wall -std=c++11 -O2 -o %:r %"
                exec "! %:r"
        endif
        if &filetype == 'python'
                exec "!python %"
        endif
endfunc
```