Summary

暑假开始准备转移博客，试了几个都不怎么满意（我还去试了下LineBlog 不知道那时候在想什么。。）
现在暂时转移至WordPress，不过还在完善中，预计。。算了不瞎预计的好。。

课上说最好做个代码集，嗯嗯 我也觉得挺有必要的
毕竟现在我连Floyd怎么写都忘了 无脑SPFA_(:з」∠)_

反正有用没用都稍微写一下，暂定是目录这些，有些还在找例题、整理代码什么的，所以还是空的。
GItHub上还欠了几题，之后会补上来。

我做的二级目录到博客园就被无视了，，将就看看吧
感觉实在简陋了些啊。。

    - k叉哈夫曼树（求合并n个数的最小代价）
    - 并查集（求图的连通性）
    - SPFA求负权环
    - SPFA求多源点最短路径（可直接作单源点用）
    - DIjkstra
    - Floyd
    - 字典树
    - 哈希表
    - 堆
    - 优先队列
    - 深搜
    - 广搜
    - 双向广搜
    - 红黑树
    - Prim
    - Kruskal
    - KMP
    - ac自动机
    - 快速幂
- 其他

题号以作业次数为准

STL

- stack
- queue
- priority_queue
- sort
- map
- set

---

stack

头文件

```
#include<stcak>
using namespace std;
```

声明

```
stack<数据类型> 变量名;
```

```
a.empty() 判断栈是否为空
a.pop() 移除栈顶元素
a.push(b) 将元素b压入栈中
a.size() 返回栈中元素个数
a.top() 返回栈顶元素
```

---

queue

头文件

```
#include<queue>
using namespace std;
```

声明

```
queue<数据类型> 变量名;
```

```
a.empty() 判断队列是否为空
a.pop() 将队头元素出队
a.push(b) 将元素b入队
a.size() 返回队列中元素个数
```

```
a.front() 返回队头元素
a.back() 返回队尾元素
```

priority_queue

头文件

```
#include<queue>
using namespace std;
```

声明

```
priority_queue<数据类型> 变量名;
```

```
a.empty() 判断队列是否为空
a.pop() 移除队头元素
a.push(b) 将元素b入队
a.size() 返回队列中元素个数
a.top() 返回队头元素

//默认从大到小
//从小到大&&多关键字
struct t
{
    int p, q;
};
priority_queue<t> a[n];
bool operator < (t x, t y)
{
    return x.p < y.p;
}
```

sort

头文件

```
#include<algorithm>
using namespace std;
```

```
//从小到大
int a[n];
sort(a,a+n);

//从大到小
int compare(int x, int y)
{
    return x > y;
}
sort(a, a + 3, compare);

//多关键字
struct t
{
    int p, q;
};
t a[n];
int compare(t x, t y)
{
    if (x.p == y.p) return x.q > y.q;
    else return x.p > y.p;
}
sort(a, a+n, compare);
```

功能函数

- MAX
- MIN
- 最大公约数

MAX

```
int max(int x, int y)
{
    return x > y ? x : y;
}
```

MIN

```
int min(int x, int y)
{
    return x < y ? x : y;
}
```

最大公约数

```
int gcd(int x, int y)
{
    if (y == 0) return x;
    else return gcd(y, x%y);
}
```

基础算法与数据结构

- 快速排序
- 归并排序
- 表达式求值（调度场算法）
- 线段树求区间和
- AVL树（不包含删除操作）
- k叉哈夫曼树（求合并n个数的最小代价）

- 并查集（求图的连通性）
- SPFA求负权环
- SPFA求多源点最短路径（可直接作单源点用）
- DIjkstra
- Floyd
- 字典树
- 哈希表
- 堆
- 优先队列
- 深搜
- 广搜
- 双向广搜
- 红黑树
- Prim
- Kruskal
- KMP
- ac自动机
- 快速幂

快速排序

```
#include<iostream>
using namespace std;

int i, j, k, n, m, s, t, a[1000];

void q(int l, int r)
{
    int i, j, x, t;
    i = l;
    j = r;
    x = a[(i + j) / 2];
    do
    {
        while (a[i] < x) i++;
```

```
        while (a[j] > x) j--;
        if (i <= j)
        {
            t = a[i];
            a[i] = a[j];
            a[j] = t;
            i++;
            j--;
        }
    } while (i <= j);
    if (j > l) q(l, j);
    if (i < r) q(i, r);
}

int main()
{
    cin >> n;
    for (i = 1; i <= n; i++)
        cin >> a[i];
    q(1, n);
    for (i = 1; i <= n; i++)
        cout << a[i] << ' ';
    return 0;
}
```

归并排序

2.1 nxd

> 给定 n 个数 a1,a2,...,an，求满足条件的(i,j)数量： i < j 且 a[i] < a[j]

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;

int a[200000], b[200000];
__int64 s;

```

```
void p(int l, int m, int r)
{
    int i = l;
    int j = m + 1;
    int k = l;
    while (i <= m && j <= r)
    {
        if (a[i] < a[j])
        {
            b[k++] = a[j++];
            s += m - i + 1;
        }
        else
        {
            b[k++] = a[i++];
        }
    }
    while (i <= m) b[k++] = a[i++];
    while (j <= r) b[k++] = a[j++];
    for (i = l; i <= r; i++)
        a[i] = b[i];
}

void q(int l, int r)
{
    if (l < r)
    {
        int m = (l + r) >> 1;
        q(l, m);
        q(m + 1, r);
        p(l, m, r);
    }
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i<n; i++)
        scanf("%d", &a[i]);
    s = 0;
    q(0, n - 1);
    printf("%I64d", s);
    return 0;
}
```

表达式求值（调度场算法）

3.2 calculator

```
#include<stdio.h>
#include<string.h>

int i, j, k, n, m, s, t, a[1000];
char b[2000], c[2000], d[2000];

int main()
{
    scanf("%s", &b);
    i = 0;
    j = 0;
    k = 0;
    n = strlen(b);
    //中缀转后缀
    while (i < n)
    {
        if ((b[i] >= '0') && (b[i] <= '9'))
        {
            while ((b[i] >= '0') && (b[i] <= '9'))
            {
                c[j++] = b[i++];
            }
            c[j++] = '!';
        }
        if ((b[i] == '+') || (b[i] == '-'))
        {
            while ((k > 0) && (d[k - 1] != '('))
            {
                c[j++] = d[k - 1];
                k--;
            }
            d[k++] = b[i];
        }
        if ((b[i] == '*') || (b[i] == '/'))
        {
            while ((k > 0) && (d[k - 1] != '(') && ((d[k - 1] == '*') ||
(d[k - 1] == '/')))
```

```
            {
                c[j++] = d[k - 1];
                k--;
            }
            d[k++] = b[i];
        }
        if (b[i] == '(')
        {
            d[k++] = b[i];
        }
        if (b[i] == ')')
        {
            while ((k > 0) && (d[k - 1] != '('))
            {
                c[j++] = d[k - 1];
                k--;
            }
            if (k > 0) k--;
        }
        i++;
    }
    while (k > 0)
    {
        c[j++] = d[k - 1];
        k--;
    }
    //计算后缀
    c[j] = '\0';
    i = 0;
    j = -1;
    while (c[i] != '\0')
    {
        if ((c[i] >= '0') && (c[i] <= '9'))
        {
            double x = 0;
            while ((c[i] >= '0') && (c[i] <= '9'))
            {
                x = 10 * x + c[i] - '0';
                i++;
            }
            j++;
            a[j] = x;
        }
        else
        {
```

```
                j--;
                switch (c[i])
                {
                case '+':
                {
                    a[j] += a[j + 1];
                    break;
                }
                case '-':
                {
                    a[j] -= a[j + 1];
                    break;
                }
                case '*':
                {
                    a[j] *= a[j + 1];
                    break;
                }
                case '/':
                {
                    a[j] /= a[j + 1];
                    break;
                }
                }
            }
            i++;
        }
        printf("%d", a[j]);
        return 0;
    }
```

线段树求区间和

5.2 bubble_sort

```
#include<stdio.h>

int i, j, k, n, m, s, t, a[300001], b[100001], c[100001];

int min(int x, int y)
```

```
{
    return x < y ? x : y;
}
int max(int x, int y)
{
    return x > y ? x : y;
}
int p(int l, int r)
{
    int s;
    s = 0;
    l += m - 1;
    r += m + 1;
    while ((l^r != 1) && (l != r))
    {
        if (l & 1 == 0) s += a[l ^ 1];
        if (r & 1 == 1) s += a[r ^ 1];
        l >>= 1;
        r >>= 1;
    }
    return s;
}

void q(int k)
{
    k >>= 1;
    while (k > 1)
    {
        a[k] = a[k << 1] + a[(k << 1) + 1];
        k >>= 1;
    }
}

int main()
{
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        scanf("%d", &b[i]);
    m = 1;
    while (m <= n) m <<= 1;
    for (i = m + 1; i <= m + n; i++)
        a[i] = 1;
    for (i = m - 1; i >= 1; i--)
        a[i] = a[i << 1] + a[(i << 1) + 1];
    for (i = 1; i <= n; i++)
```

```
        {
            t = p(1, b[i] - 1) + i;
            c[b[i]] = max(b[i], max(t, i)) - min(b[i], min(t, i));
            a[m + b[i]] = 0;
            q(m + b[i]);
        }
        printf("%d", c[1]);
        for (i = 2; i <= n; i++)
            printf(" %d", c[i]);
        return 0;
    }
```

AVL树（不包含删除操作）

8.1 wbhavl

```
#include<stdio.h>
#include<stdlib.h>

int i, j, k, n, m, s, t, a[100001];

struct node
{
    int dep;
    int val;
    node *p;
    node *l;
    node *r;
};

node* insert(node *tree, int value);
void updata(node *tree);
int depth(node *tree);
node* aaaavl(node *tree, node *newp);
int papa(node *tree);
node* leftSingle(node *tree);
node* rightSingle(node *tree);
node* leftDouble(node *tree);
node* rightDouble(node *tree);
int haha(node *tree, int pp);
```

```
25.
26.  node* insert(node *tree, int value)
27.  {
28.      node *newp, *nowp;
29.      newp = new node;
30.      newp->val = value;
31.      newp->p = NULL;
32.      newp->l = NULL;
33.      newp->r = NULL;
34.      if (tree == NULL)
35.      {
36.          newp->dep = 1;
37.          tree = newp;
38.      }
39.      else
40.      {
41.          nowp = tree;
42.          while (1 > 0)
43.          {
44.              if (newp->val <= nowp->val)
45.              {
46.                  if (nowp->l == NULL)
47.                  {
48.                      nowp->l = newp;
49.                      newp->p = nowp;
50.                      break;
51.                  }
52.                  else
53.                  {
54.                      nowp = nowp->l;
55.                      continue;
56.                  }
57.              }
58.              else
59.              {
60.                  if (nowp->r == NULL)
61.                  {
62.                      nowp->r = newp;
63.                      newp->p = nowp;
64.                      break;
65.                  }
66.                  else
67.                  {
68.                      nowp = nowp->r;
69.                      continue;
```

```
                }
            }
        }
        updata(newp);
        tree = aaaavl(tree, newp);
    }
    return tree;
}

void updata(node *tree)
{
    if (tree == NULL) return;
    else
    {
        int l, r;
        l = depth(tree->l);
        r = depth(tree->r);
        tree->dep = 1 + (l > r ? l : r);
    }
}

int depth(node *tree)
{
    if (tree == NULL) return 0;
    else return tree->dep;
}

node* aaaavl(node *tree, node *newp)
{
    int pa;
    while (newp != NULL)
    {
        updata(newp);
        pa = papa(newp);
        if ((pa < -1) || (pa > 1))
        {
            if (pa > 1)
            {
                if (papa(newp->r) > 0)
                {
                    newp = leftSingle(newp);
                }
                else
                {
                    newp = leftDouble(newp);
```

```
                }
            }
            if (pa < -1)
            {
                if (papa(newp->l) < 0)
                {
                    newp = rightSingle(newp);
                }
                else
                {
                    newp = rightDouble(newp);
                }
            }
            if (newp->p == NULL) tree = newp;
            break;
        }
        newp = newp->p;
    }
    return tree;
}

int papa(node *tree)
{
    if (tree == NULL) return 0;
    else return depth(tree->r) - depth(tree->l);
}

node* leftSingle(node *tree)
{
    node *newroot, *mature;
    mature = tree->p;
    newroot = tree->r;
    if (newroot->l != NULL)
    {
        newroot->l->p = tree;
    }
    tree->r = newroot->l;
    updata(tree);
    newroot->l = tree;
    newroot->p = mature;
    if (mature != NULL)
    {
        if (mature->l == tree)
        {
            mature->l = newroot;
```

```
160.            }
161.            else
162.            {
163.                mature->r = newroot;
164.            }
165.        }
166.        tree->p = newroot;
167.        updata(newroot);
168.        return newroot;
169.    }
170.
171.    node* rightSingle(node *tree)
172.    {
173.        node *newroot, *mature, *naive;
174.        mature = tree->p;
175.        newroot = tree->l;
176.        if (newroot->r != NULL)
177.        {
178.            newroot->r->p = tree;
179.        }
180.        tree->l = newroot->r;
181.        updata(tree);
182.        newroot->r = tree;
183.        newroot->p = mature;
184.        if (mature != NULL)
185.        {
186.            if (mature->l == tree)
187.            {
188.                mature->l = newroot;
189.            }
190.            else
191.            {
192.                mature->r = newroot;
193.            }
194.        }
195.        tree->p = newroot;
196.        updata(newroot);
197.        return newroot;
198.    }
199.
200.    node* leftDouble(node *tree)
201.    {
202.        rightSingle(tree->r);
203.        return leftSingle(tree);
204.    }
```

```
205.
206.  node* rightDouble(node *tree)
207.  {
208.      leftSingle(tree->l);
209.      return rightSingle(tree);
210.  }
211.
212.  int haha(node *tree, int pp)
213.  {
214.      node *nowp;
215.      int qq;
216.      qq = 1;
217.      nowp = tree;
218.      while (nowp)
219.      {
220.          if (nowp->val > pp)
221.          {
222.              nowp = nowp->l;
223.              qq++;
224.          }
225.          else
226.          {
227.              if (nowp->val < pp)
228.              {
229.                  nowp = nowp->r;
230.                  qq++;
231.              }
232.              else break;
233.          }
234.      }
235.      return qq;
236.  }
237.
238.  int main()
239.  {
240.      node *tree, *now;
241.      int val;
242.      tree = NULL;
243.      scanf("%d", &n);
244.      for (i = 0; i < n; i++)
245.      {
246.          scanf("%d", &a[i]);
247.          tree = insert(tree, a[i]);
248.      }
249.      printf("%d", haha(tree, a[0]));
```

```
    for (i = 1; i < n; i++)
        printf(" %d", haha(tree, a[i]));
    return 0;
}
```

---

k叉哈夫曼树（求合并n个数的最小代价）

也可用堆或优先队列

9.1 hbsz

```
#include<stdio.h>
#include<algorithm>
using namespace std;

int i, j, k, n, m, s, t, b[100002];
short int a[100002];

int main()
{
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    sort(a, a + n);
    t = 0;
    i = 0;
    j = 0;
    s = 0;
    while (n - i + t - j > 1)
    {
        m = 0;
        for (k = 0; k < 2; k++)
        {
            if (i == n)
            {
                m += b[j];
                j++;
            }
            else
                if (j == t)
```

```
                {
                    m += a[i];
                    i++;
                }
                else
                    if (a[i] < b[j])
                    {
                        m += a[i];
                        i++;
                    }
                    else
                    {
                        m += b[j];
                        j++;
                    }
            }
            s += m;
            b[t] = m;
            t++;
        }
        printf("%d", s);
        return 0;
    }
```

并查集（求图的连通性）

10.2 friends

```
#include<stdio.h>

struct node
{
    int x, y;
};

node e[50010];

int i, j, k, n, m, s, t, x, y, d, l, a[50010], b[50010], f[50010], c[50010], p[50010], q[50010];

```

```
int aaaa(int x)
{
    return f[x] == x ? x : f[x] = aaaa(f[x]);
}

void qqq(int x)
{
    int i, pp, qq;
    pp = aaaa(x);
    i = a[x];
    while (i != 0)
    {
        if (p[e[i].y])
        {
            qq = aaaa(e[i].y);
            if (pp != qq)
            {
                t--;
                f[qq] = pp;
            }
        }
        i = e[i].x;
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for (i = 0; i < n; i++)
    {
        f[i] = i;
    }
    l = 0;
    for (i = 0; i < m; i++)
    {
        scanf("%d%d", &x, &y);
        l++;
        e[l].x = a[x];
        a[x] = l;
        e[l].y = y;
        l++;
        e[l].x = a[y];
        a[y] = l;
        e[l].y = x;
    }
```

```
    scanf("%d", &d);
    for (i = 1; i <= d; i++)
    {
        scanf("%d", &b[i]);
        c[b[i]] = 1;
    }
    t = 0;
    for (i = 0; i < n; i++)
    {
        if (!c[i])
        {
            t++;
            qqq(i);
            p[i] = 1;
        }
    }
    q[d + 1] = t;
    for (i = d; i >= 1; i--)
    {
        t++;
        qqq(b[i]);
        p[b[i]] = 1;
        q[i] = t;
    }
    for (i = 1; i <= d + 1; i++)
    {
        printf("%d\n", q[i]);
    }
    return 0;
}
```

SPFA求负权环

11.1 CrazyScientist

```
#include<stdio.h>

int i, j, k, n, m, s, t, p, a[2010], b[80010][3], c[2010];
bool d[2010];

```

```
void q(int k)
{
    int i, j;
    d[k] = true;
    i = a[k];
    while (i != 0)
    {
        j = b[i][0];
        if (c[k] + b[i][1] < c[j])
        {
            c[j] = c[k] + b[i][1];
            if ((d[j] == true) || (p == 1))
            {
                p = 1;
                if (d[s] == true)
                {
                    t = 1;
                }
                break;
            }
            q(j);
        }
        i = b[i][2];
    }
    d[k] = false;
}

int main()
{
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++)
    {
        a[i] = 0;
        c[i] = 0;
        d[i] = false;
    }
    s = 0;
    for (i = 1; i <= m; i++)
    {
        scanf("%d%d%d", &j, &k, &t);
        s++;
        b[s][0] = k;
        b[s][1] = t;
        b[s][2] = a[j];
        a[j] = s;
```

```
        }
        scanf("%d", &s);
        t = 0;
        for (i = 1; i <= n; i++)
        {
            p = 0;
            q(i);
            if (t == 1) break;
        }
        if (t == 1)
            printf("EL PSY CONGROO");
        else
            printf("ttt");
        return 0;
    }
```

SPFA求多源点最短路径（可直接作单源点用）

11.2 FuYihao

```
#include<stdio.h>
#include<string.h>

int i, j, k, n, m, s, t, q, a[410][410] = { 0 }, b[410][410] = { 0 }, c[410], d[200010], e[410][410];
bool f[410];

void sasasa(int k)
{
    int i, j, h, t;
    if (k > 1)
    {
        j = 1;
        for (i = 2; i < k; i++)
            if (e[i][k] < e[j][k]) j = i;
        for (i = 1; i <= n; i++)
            e[k][i] = e[j][k] + e[j][i];
    }
    e[k][k] = 0;
    f[k] = true;
```

```
20.      d[1] = k;
21.      h = 0;
22.      t = 1;
23.      while (h < t)
24.      {
25.          h++;
26.          j = d[h];
27.          f[j] = false;
28.          for (i = 1; i <= n; i++)
29.          {
30.              if (e[k][i] > e[k][j] + a[j][i])
31.              {
32.                  e[k][i] = e[k][j] + a[j][i];
33.                  if (f[i] == false)
34.                  {
35.                      t++;
36.                      d[t] = i;
37.                      f[i] = true;
38.                  }
39.              }
40.          }
41.      }
42.  }
43.
44.  int main()
45.  {
46.      memset(a, 0x3f, sizeof(a));
47.      memset(e, 0x3f, sizeof(e));
48.      scanf("%d%d", &n, &m);
49.      for (i = 0; i < m; i++)
50.      {
51.          scanf("%d%d%d", &j, &k, &t);
52.          if ((a[j][k] != 0) && (t > a[j][k])) continue;
53.          a[j][k] = t;
54.          a[k][j] = t;
55.      }
56.      scanf("%d", &q);
57.      for (i = 1; i <= n; i++)
58.      {
59.          memset(f, 0, sizeof(f));
60.          sasasa(i);
61.      }
62.      while (q--)
63.      {
64.          scanf("%d%d", &j, &k);
```

```
            if (e[j][k] != 0x3f3f3f3f)
            {
                if (q > 0) printf("%d\n", e[j][k]);
                else printf("%d", e[j][k]);
            }
            else
            {
                if (q > 0) printf("-1\n");
                else printf("-1");
            }
        }
        return 0;
    }
```

Dijkstra