ACM / ICPC程序设计

简单算法

作者：屠添翼

模拟算法及题目

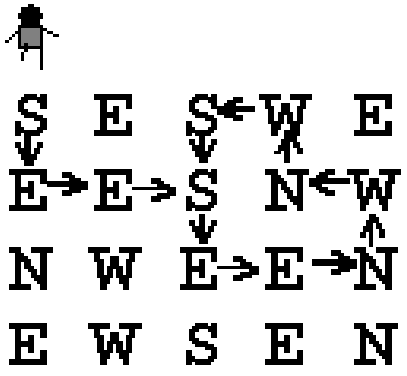
枚举算法及题目

贪心算法及题目

综合实例

一个简单的模拟题

Guidl:
10 step(s) to exit



Grid2

3 step(s) befor a
loop of 8 step(s)

- 题目（zju1708：Robot Motion）

  以矩阵形式给定一张地图和机器人的初始位置。矩阵上每一点的字母代表在这一点机器人的移动方向。如果机器人按图中信息能走出的话输出需要的步数。如果机器人进入某个循环则输出循环前所走的步数和循环的长度。

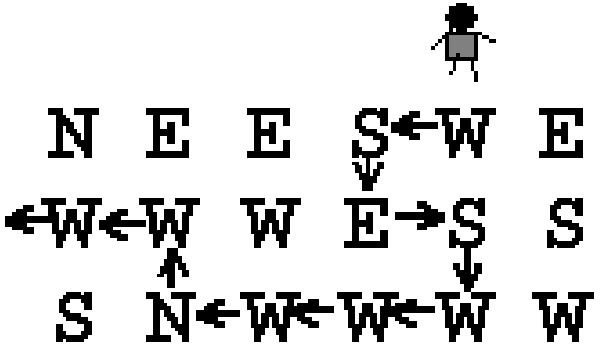
Sample input and output:

- Sample Input

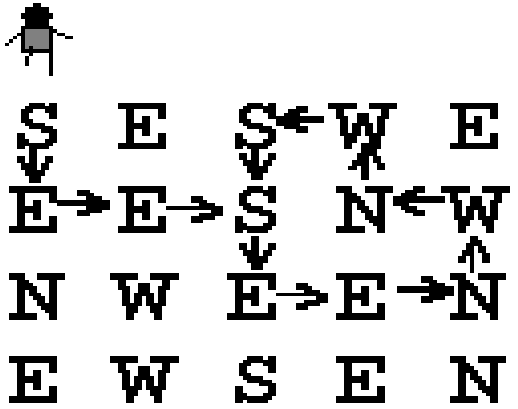
```
3 6 5
NEESWE
WWWESS
SNWWWW
4 5 1
SESWE
EESNW
NWEEN
EWSEN
0 0 0
```

- Sample Output

```
10 step(s) to exit
 3 step(s) before a loop of 8 step(s)
```



Guid1:
10 step(s) to exit



Grid 2

3 step(s) befor a loop of 8 step(s)

问题分析

- 我们有什么信息？
- 地图，起点
- 我们要什么？
- 移动结束（走出边界或遇到以前走过的点）时，该次移动的序号。（如果是遇到以前走过的点，还需知道该点的序号）
- 设计数据结构

  主要信息：

  int data[100][100]; //存放地图

  int place; //当前序号

  int m,n; //矩阵大小

  int begin; //起始位置

- 方法：

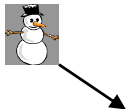
  按照地图及初态移动机器人直到满足结束条件：下一个要经过的点（row，line）满足（row<0 || row>=m || line< 0 || line>=n) 或 data[row][line] >0.用place ++标记刚走过的点。

  如果走出地图输出：place –1 step(s) to exit 。

  否则输出：data[row][line]-1 step(s) before a loop of place-data[row] [line] step(s)

运行演示

N	E	E	S	W	E
W	W	W	E	S	S
S	N	W	W	W	W



-1	-4	-4	-2	-3	-4
-3	-3	-3	-4	-2	-2
-2	-1	-3	-3	-3	-3

place = 1

row = 0

col = 4

3 6 5
NEESWE
WWWESS
SNWWWW

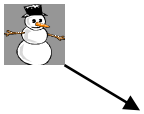
N	-1
S	-2
W	-3
E	-4

运行演示

place = 2

row = 0

col = 3



-1	-4	-4	-2	-3	-4
-3	-4	-4	-2	-3	-4
-2	-1	-3	-3	-3	-3

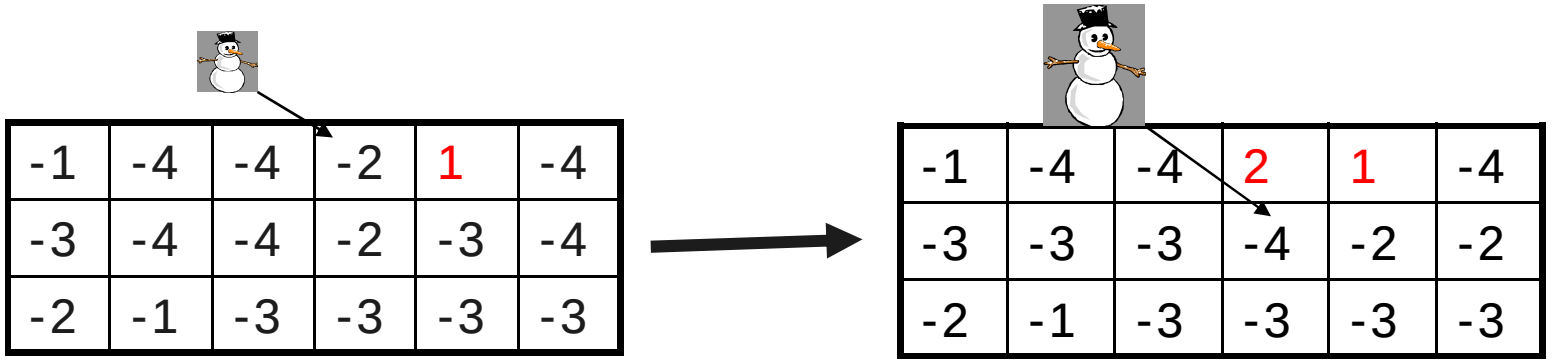
-1	-4	-4	-2	1	-4
-3	-3	-3	-4	-2	-2
-2	-1	-3	-3	-3	-3

place = 3

row = 1

col = 3

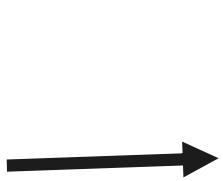
Guidl:
10 step(s) to exit



-1	-4	-4	-2	1	-4
-3	-4	-4	-2	-3	-4
-2	-1	-3	-3	-3	-3

-1	-4	-4	2	1	-4
-3	-3	-3	-4	-2	-2
-2	-1	-3	-3	-3	-3

place = 3
row = 1
col = 3



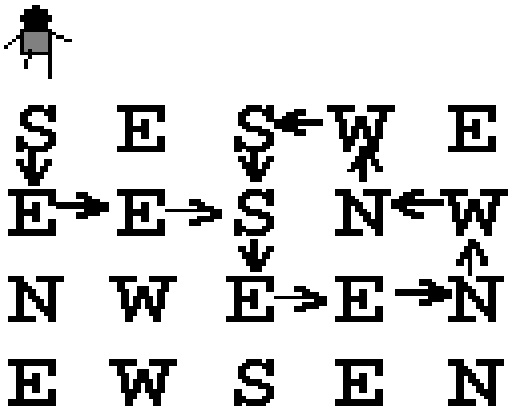
place = 11
row = 1
col = -1

-1	-4	-4	2	1	-4
10	9	-3	3	4	-2
-2	8	7	6	5	-3

满足结束条件：**col < 0**
输出：**10 step(s) to exit**

同理Grid2结束状态为：

1	-4	11	10	-4
2	3	4	9	8
-1	-3	5	6	7
-4	-3	-2	-4	-1



Grid 2

```
3 step(s) befor a
loop of 8 step(s)
```

满足结束条件2：data[row][col] > 0

输出：

3 step(s) before a loop of 8 step(s)

data[row][col] -1

place-data[row] [col]

place = 12

row = 1

line = 2

Robot Motion 程序

```
#include <iostream>
using namespace std;
int data[100][100]; // 地图
int place; // 当前位置
int m,n,begin; // 地图大小，初始位置
void move(int row,int col)；//按要求移动机器人
int main(){
    return 0;
}
```

Robot Motion 程序(续)

```
int main(){
    char ch;
    while( (cin>>m>>n>>begin) && m){ //输入数据
        place=1;
        for(int i=0;i<m;i++)
            for(int j=0;j<n;j++){
                cin>>ch;
                switch (ch){ //把字符转换成整数以便于统一处理
                        case 'N': data[i][j]=-1; break;
                        case 'S': data[i][j]=-2; break;
                        case 'W': data[i][j]=-3; break;
                        case 'E': data[i][j]=-4; break;
                }
            }
        move(0,begin-1);
    } //end of while
    return 0;
} //end of main
```

Robot Motion 程序(续)

```
void move(int row,int col)
{   bool notExit=true;
    while( notExit && (data[row][col]<0) ) {
        int d = data[row][col];
        data[row][col]= place;
        switch (d) {
            case -1: row--;  break;
            case -2: row++;  break;
            case -3: col--;  break;
            case -4: col++;  break;
        }
        place++;
        notExit = (row>-1)&&(row<m)&&(col>-1)&&(col<n);
    } //end of while
    if(!notExit) cout<<place-1<<" step(s) to exit"<<endl;
    else cout<<data[row][line]-1<<" step(s) before a loop of "
            <<place-data[row][line]<<" step(s)"<<endl;
} //end of move
```

- 结论：一个简洁高效的算法在很大程度上依赖于数据结构的建立。这也往往是做好一个模拟题的重点。

枚举算法及举例

- 核心思想：通过列举所有情况然后逐一判断，从而得到结果。从本质上看，就是把问题用一定的数据结构描述，然后用某种方式遍历它，以寻求问题的解。

一个简单的例子

- Zero Sum （USACO 36）

问题描述：

给定一个整数N（3=<N<=9)。在序列1...N中插入‘+’，‘-’，‘’，构造表达式。按字典序输出所有使得表达式为0的情况。

Sample Input: 7
Sample Output 1+2-3+4-5-6+7 1+2-3-4+5+6-7 1-2 3+4+5+6+7 1-2 3-4 5+6 7 1-2+3+4-5+6-7 1-2-3-4-5+6+7

问题分析

- 在1..N中插入‘+’，‘–’，‘’一共有多少中插法？

  考虑最坏情况N＝9 时，有

  3^8 ＝ 6561 种情况

  这个解空间比较小，这就意味着我们可以列举所有情况看它是否满足“使得表达式为0”

  选择变量：

  string digital = "112233445566778 89";

  int n;

  思路：输入**n**后把**digital**中**2*n – 1** 后剔除，并在奇数位置按字典序枚举各种插入情况。对每种情况如果表达式值为**0**，则输出之。

Zero Sum 的枚举法实现

```
#include <fstream>
#include <string>
#include <sstream>
using namespace std;
ifstream fin("zerosum.in");
Ofstream fout("zerosum.out");
string digital = "11223344556677889";
int n;   //问题规模

int check(); // 检查表达式是否为0
void proc(int t); //确定第t处应插入什么字符
int main()
{
    fin>>n; //输入问题规模
    digital.resize(2*n -1); //剔除2*n-1后边的元素
    proc(0);
     return 0;
}
```

Zero Sum 的枚举法实现

```
void proc(int t){
    if(t == n - 1){ //已经插入完毕
          //如果和为0则输出
          if (check() == 0)
                fout<<digital<<endl;
          return ;
    }
    //确定要插入字母在digital中的位置
    int i = 2 * t + 1;
    digital[i] = ‘ ’; //插入‘ ’  (依字母序)
    proc(t + 1);  //在下一个位置插入
    digital[i] = '+';
    proc(t + 1);
    digital[i] = '-';
    proc(t + 1);
}
```

Zero Sum 的枚举法实现

```
int check()
{
    /*去处插入字符后digital中的‘’，
      并存入teamp中*/
    string teamp;
    for(int i = 0; i < digital.size();
    i++)
        if(digital[i] != ' ')
            teamp += digital[i];
    /*把teamp作为输入流
      需要#include  <sstream>*/
    istringstream ssin(teamp);
    int sum;
    ssin >> sum;
    char ch;
    while(ssin>>ch)
    {
        int t;
        ssin>>t;
        if(ch == '-')
            sum -= t;
        else
            sum += t;
    }
    return sum;
}
```

枚举的特点

- 枚举的特点－－简明但低效

  只要有合适的搜索规则，就能设计出枚举算法，就可以解决几乎所有的问题。这是一种普遍适用的解题策略。

  枚举算法的普遍性换来的代价是低效，由于它要考虑所有可能的情况，如果问题的规模很大，情况很多，算法难免耗时过长。如果对算法进行优化，精简枚举范围，就有可能在可接受的时间内得出结果。

重点提示

- 虽然枚举在理论上可以解决绝大部分优化问题，然而当问题规模较大时其速度是相当差的。
- 然而枚举法的用处比我们想象的要大。在问题毫无头绪的情况下，枚举往往可以为我们打开一个缺口。

贪心算法及应用

- 贪心法的基本思想是每次选择一个局部最优策略来实施，而不考虑对今后的影响，一般来说，其时间复杂度较低。对很多题目，用贪心法并不能得到最优解，该方法的另一个难点是比较难以证明其最优性。但是，如果能证明当前策略至少不比其他策略差，就可以继续做下去。

- 贪心算法总是作出在当前看来最好的选择，也就是说贪心算法并不从整体最优上加以考虑，它所作出的选择只是在某种意义下的局部最优选择。
- 当然，我们希望从局部的最优选择能得到的最终结果是问题的整体最优解。

贪心法的基本要素

- 贪心选择性质

  所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。

- 最优子结构性质

  当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。

Mixing Milk（usaco76）

- 问题描述

  某商人欲从M个农民那里购买N公斤牛奶。给定整数M，N及若干个农民拥有牛奶的价格和总量。问他最少需要多少钱可以购买N公斤牛奶。

- INPUT FORMAT

  Line 1: Two integers, N and M.
  The first value, N, (0 <= N <= 2,000,000) is the amount of milk that Merry Milk Makers' want per day. The second, M, (0 <= M <= 5,000) is the number of farmers that they may buy from.

  Lines 2 through M+1: The next M lines each contain two integers, Pi and Ai.
  Pi (0 <= Pi <= 1,000) is price in cents that farmer i charges.
  Ai (0 <= Ai <= 2,000,000) is the amount of milk that farmer i can sell to Merry Milk Makers per day.

- Sample Input

```
100 5
5 20
9 40
3 10
8 80
6 30
```

- Output

```
630
```

问题分析

- 用贪心法的关键是找到最优的度量标准
- 既然本题要求花钱最小，我们就可以选择牛奶的价格为度量标准。

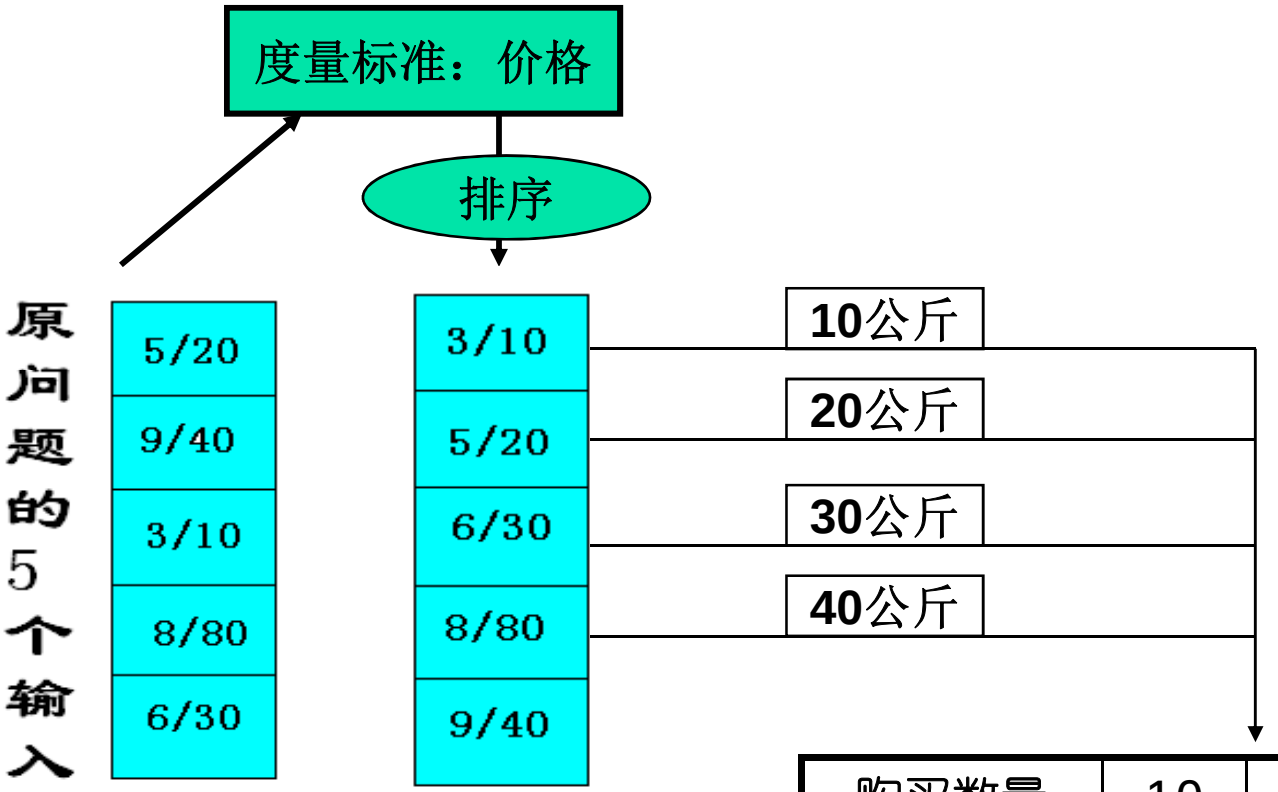
思考：该问题的最优子结构性质和贪心选择性质？

- 解题方法：

对每个农民按其牛奶价格进行排序，每次选择价格最低的。如果还没有买够就向该农民购买。

Mixing Milk 实例

问题：N ＝ 100，M ＝ 5，price ＝ {5,9,3,8,6}, num = {20,40,10,80,30}



购买数量	10	20	30	40
购买价格	3	5	6	8
已经花钱	30	130	310	630

实现代码

```
#include<fstream>
#include<vector>
#include<algorithm>
using namespace std;
ifstream fin("milk.in");
ofstream fout("milk.out");
struct IntPair
{
    IntPair(int a = 0, int b = 0)
    {
        first = a, second = b;
    }
    int first, second;
};
//比较谓词
bool small(IntPair i1,IntPair i2){
        return i1.first<i2.first;
}
int cost(vector<IntPair> &v,int M);
int main(){
    int M,N,P,A;
    vector<IntPair> v;
    v.reserve(5000);
    fin>> N >> M;
    for(int i = 0; i<N; i++){
        fin>>P>>A;
        v.push_back(IntPair(P,A));
    }
    fout<<cost(v,M)<<endl;
    return 0;
}
```

实现代码『续』

```
int cost(vector<IntPair> &v,int M) {
    sort(v.begin(), v.end(), small);  //按价格排序
    int result=0,i=0,aready=0;
    while(v[i].second+aready < M){ //若尚未买够，继续买
        aready += v[i].second;
        result += v[i].second*v[i].first;
        i++;
    }
    return result + v[i].first * (M - aready);
} // end of cost
```