算法实验报告

姓名：王敬博
学号：U201314936
班级：ACM1301
指导老师：王多强

1.最近点对问题的算法实现

1.1 题目描述

测试文件
Input file: in.dat
格式：第一行为一个整数，表示测试用例的组数，其后跟相应组数的测试用例。
每组测试用例包括：
首行：一个整数，表示本组测试用例包含的点数，其后跟相应点数的行
其后：每行两个整数，表示该点的x,y坐标
Output file：out.dat
格式：每行输出一个测试用例的答案，即本组测试用例中相距最近的两个点，用点的坐标表示：四个整数。前两个整数表示第一个点，后两个整数表示第二个点。若有多对相距最近的点，依次罗列。

1.2 算法设计

距离描述：已知平面上分布着点集P中的n个点$p_1,p_2,...p_n$，点i的坐标记为(x_i,y_i)，$1\le i\le n$。两点之间的距离取其欧式距离：（两个点位于同一位置时，距离为0）

$$d(p_i,p_j)=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$$

分析：若采取暴力解决的方法，全搜索，计算每一对点的距离，然后比较大小，从中找出最小者，该方法的时间复杂度为:
(1) 计算点间距离，需要计算n*(n-1)/2对点间的距离，O(n^2)
(2) 找最小距离， O(n^2),因为需要n*(n-1)/2－1次比较
总的时间复杂度O(n^2)

1.2.1 分治法设计

利用分治法“设计”一个具有O(nlogn)时间复杂度的算法求解最近点对问题。

设计策略：

1）首先将所有的点按照x坐标排序。排序过程需要O(nlogn)的时间，不会从整体上增加时间复杂度的数量级。

2）划分：由于点已经按x坐标排序，所以空间上可以“想象”画一条垂线，作为分割线，将平面上的点集分成左、右两半SL和SR。

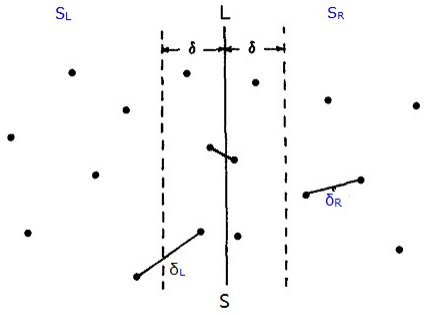
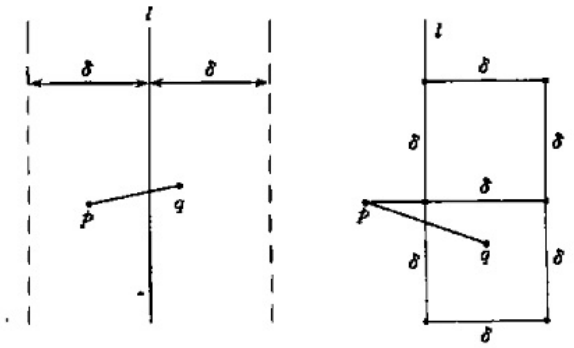


Figure 1.

最近的一对点或者在S_L中，或者在S_R中，或者一个在S_L中而另一个在S_R中（跨越分割线）。

建立一个递归过程求d_L和d_R，并在此基础上计算d_C。而且，要使得d_C的计算至多只能花O(n)的时间。

这样，递归过程将由两个大致相等的一半大小的递归调用和O(n)附加工作组成，总的时间就可以控制在O(nlogn)以内。



以L为中心，δ为半径划分一个长带，最小点对还有可能存在于SL和SR的交界处，如上图中的虚线带，p点和q点分别位于SL和SR的虚线范围内，在这个范围内，p点和q点之间的距离才会小于δ，最小点对计算才有意义。

对于SL虚框范围内的p点，在SR虚框中与p点距离小于δ的顶多只有六个点，就是上图中的2个正方形的6的顶点。这个可以反推证明，如果右边这2个正方形内有7个点与p点距离小于δ，例如q点，则q点与下面正方形的四个顶点距离小于δ，则和δ为SL和SR中的最小点对距离相矛盾。因此对于SL虚框中的p点，不需求出p点和右边虚线框内所有点距离，只需计算SR中与p点y坐标距离最近的6个点，就可以求出最近点对，节省了比较次数。

这样，对于每个p_i，有最多7个点p_j要考虑，所以对每个p_i的计算时间可看作是O(1)的。则，计算比δ 好的d_C的时间就为O(n)，即使在最坏的情况下。

于是得：最近点对求解过程由两个一半大小的递归调用加上合并两个结果的线性附加工作组成。

1.2.2 算法流程

(1)做分割线（垂线）将点集分为PL和PR两部分

(2)递归地在PL中的找具有dL的点对

(3)递归地在PR中的找具有dR的点对

(4)在跨越分割线的点对中找具有dC的点对

(5)返回min(dL,dR,dC)对应的点对。

计算d_C的方法：

设点按它们的y坐标排序，从p_i开始向远处搜索某个p_j时，若与p_j的y坐标相差大于δ，那么这样的p_i以及更远的点都可以终止计算，转而继续处理p_{i+1}

for i=1 to numPointsInStrip do

for j=i+1 to numPointsInStrip do

if p_i *and* p_j *'s y-coordinates differ by more than* δ

break;

else if distance(p_i, p_j) < δ

δ = *dist*(p_i, p_j);

1.3 源代码

```
//
//  main.cpp
//  FileInputOutput
//
//  Created by bobobo on 11/8/15.
//  Copyright © 2015 bobobo. All rights reserved.
//
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <vector>
#include <iostream>
#include <fstream>
#include <cstdlib>
```

```
//#include "stdafx.h"
int n ;  //n组测试数据
const int MaxSize = 100000 + 10; //最大点数
const double infinity = 1e20;       //无穷大距离
struct Point
{
    int x, y;
    Point(int  a = 0, int b = 0) : x(a), y(b) {}
} v[MaxSize], tem[MaxSize];     //点集, 中间点集
struct PointPair
{
    int x1;
    int y1;
    int x2;
    int y2;
}vP[MaxSize];
using namespace std;

bool CompareX (const Point &p1, const Point &p2) { return p1.x < p2.x; }
bool CompareY (const Point &p1, const Point &p2) { return p1.y < p2.y; }

//计算欧几里得距离
inline double calDis(const Point &p1, const Point &p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y));
}

int findVertex(Point arr[],int n, int x,int y)
{
    int i = 0;
    int index = -1;
    for (i = 0; i < n; i++) {
        if((x == arr[i].x)&&(y == arr[i].y))
        {
            index = i;
            break;
        }
    }
```

```
    return index;
}
//寻找下标在[st, ed]之间的最近点对
double closestPair(int st, int ed, int &p1, int &p2)
{
    double dis = infinity, temdis;
    //int lenArray[MaxSize];
    if (st >= ed) return dis;
    //int m = ed - st;
    int mid = st + ((ed - st) >> 1);
    int t1 ,t2;
    if ((temdis = closestPair(st, mid, t1, t2)) < dis)  //左边
        dis = temdis, p1 = t1, p2 = t2;
    if ((temdis = closestPair(mid+1, ed, t1, t2)) < dis)//右边
        dis = temdis, p1 = t1, p2 = t2;

    //寻找距离中间x坐标小于dis的点，并按Y坐标排序
    int len = 0;
    for (int i = st; i <= ed; i++)
        if (fabs(v[i].x - v[mid].x) < dis)
        {
            tem[len++] = v[i];
            //lenArray[len++] = i;
        }
    sort(tem, tem + len, CompareY);

    //考虑每个点附近至多8个点，由距离控制
    for (int i = 0; i < len; i++)
        for (int j = i + 1; ((j < len )&& ((tem[j].y - tem[i].y) <= dis)); j++)
            if ((temdis = calDis(tem[i], tem[j])) < dis)
            {
                dis = temdis;
                p1 = findVertex(v, n, tem[i].x, tem[i].y);
                p2 = findVertex(v, n, tem[j].x, tem[j].y);
            }
    return dis;
}
```

```
int main()
{
    ifstream fin("/users/bobobo/Documents/AlgSevenProblems/FileInputOutput/in.dat",ios::inlios::binary);
    if(!fin)
    {
        cerr<<"open error!"<<endl;
        abort( );
    }
    fin.read((char *)&n, sizeof(n));
    ofstream fout("out.dat",ios::binary);
    if(!fout)
    {
        cerr<<"open error!"<<endl;
        abort( );//退出程序
    }
    int j;
    for(int i = 0 ; i < n ; i++)
    {
        fin.read((char *)&j, sizeof(j));//std::getline(fin,lineStr);
        //int j = std::stoi(lineStr);    //j=>这一组的测试数据的个数
        for (int k = 0; k < j; k++) {
            fin.read((char *)&v[k], sizeof(v[k]));
        }
        sort(v, v + j, CompareX);
        int k1, k2;
        double dis = closestPair(0, j - 1, k1, k2);
        vP[i].x1 = v[k1].x;
        vP[i].y1 = v[k1].y;
        vP[i].x2 = v[k2].x;
        vP[i].y2 = v[k2].y;
        fout.write((char *)&vP[i], sizeof(vP[i]));  //写进n组数据作为最小值
    }

}
```

1.4 测试分析

显示in.dat数据

```
The Number of The Group:100
the Number of the datas:6
the x:4the y :8
the x:0the y :8
the x:1the y :8
the x:9the y :1
the x:3the y :6
the x:3the y :4
the Number of the datas:1
the x:2the y :7
the Number of the datas:4
the x:3the y :8
the x:3the y :8
the x:7the y :6
the x:6the y :7
the Number of the datas:6
the x:7the y :2
the x:1the y :6
the x:7the y :2
the x:9the y :8
the x:0the y :7
the x:0the y :9
the Number of the datas:4
the x:2the y :6
the x:1the y :5
the x:6the y :9
the x:6the y :2
the Number of the datas:9
the x:1the y :6
the x:5the y :9
the x:4the y :6
the x:1the y :4
the x:2the y :0
the x:7the y :5
the x:6the y :9
the x:8the y :8
the x:6the y :4
```

显示计算后放入out.dat中的数据

将每组对应的两对坐标分别打印 x,y,x,y,即输出数据中每四行为一组

```
The x is:0
The y is:8
The x is:1
The y is:8
The x is:2
The y is:7
The x is:1
The y is:8
The x is:3
The y is:8
The x is:3
The y is:8
The x is:7
The y is:2
The x is:7
The y is:2
The x is:1
The y is:5
The x is:2
The y is:6
The x is:5
The y is:9
The x is:6
The y is:9
The x is:8
The y is:7
The x is:9
The y is:9
```

每组对应的最小点对都是符合要求的。

1.5 技术总结

如果每次递归都要对点的y坐标进行排序，则这又要有O(nlogn)的附加工作。总的时间为

$$T(n) = 2T(n/2) + n\log n$$

可得，整个算法的时间复杂度就为$O(n\log^2 n)$。

解决方案：改进对点的坐标进行排序的处理方法。

策略：设置两个表，**P表：按x坐标对点排序得到的表；Q表：按y坐标对点排序得到的表。**

这两个表可以在预处理阶段花费O(nlogn)时间得到。

再记，P_L和Q_L是传递给左半部分递归调用的参数表，P_R和Q_R是传递给右半部分递归调用的参数表。

在使用上述策略将P分为两个部分之后，复制Q到Q_L和Q_R中，然后删除Q_L和Q_R中不在各自范围内的点。这一操作花费O(n)时间即可完成。而此时Q_L和Q_R均已按y坐标排好序。

然后：将P_L、P_R和上面的Q_L、Q_R带入递归过程进行处理，P_L, P_R是按照x坐标排序的点集，Q_L、Q_R是按照y坐标排序的点集。

最后：当递归调用返回时，扫描本级的Q表，删除其x坐标不在带内的所有点。此时Q中就只含有带中的点，而且这些点已是按照y坐标排好序了的。这一处理需要O(n)的时间。再进一步，对每个p_i，寻找近邻中$\Delta y \le \delta$的p_j即可。

综上所述，所有附加工作的总时间为O(n)，则整个算法的**计算时间为**

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ &= O(n\log n) \end{aligned}$$

2.大整数的算术运算

2.1　题目描述

利用分治法设计一个计算两个n位的大整数相乘的算法，要求计算时间低于$O(n^2)$。
大整数：位数很多的整数，普通的计算机不能直接处理，如： 983497597213080234579102349857034 5

对大整数的算术运算，显然常规程序语言是无法直接表示的。编程实现大整数的加、减、乘运算，需考虑操作数为0、负数、任意位等各种情况
测试文件：Input file in.dat
格式：第一行为一个整数，表示测试用例的组数，其后跟相应组数的测试用例。每个测试用例一行，包含3个整数（长整数数字串），前两个是待测试的操作数，第3个整数表示操作类型（1：加法，2：减法，3：乘法）。
Output file：out.dat
格式：每组测试用例输出一行，最后不要加空行.

2.2 算法设计

2.2.1 算法描述

大整数的加法可以采取循环求解，计算出较大的整数位数，循环相加大整数每一位和进位，直至最高位，减法采取同样的原理。
乘法的两个乘数比较大，最后结果超过了整型甚至长整型的最大范围，此时如果需要得到精确结果，就不能常规的使用乘号直接计算了，就需要采用分治的思想，将乘数“分割”，将大整数计算转换为小整数计算。

分治的规律找到了，接下来就是具体实现的思想了。
依然是递归思想（这里为了简化，就不递归到两位数相乘了，4位数相乘，计算机还是能够得到精确值的）：
1. 如果两个整数M和N的长度都小于等于4位数，则直接返回M*N的结果的字符串形式。
2. 如果如果M、N长度不一致，补齐M高位0（不妨设N>M），使都为N位整数。
3. N/2取整，得到整数的分割位数。将M、N拆分成m1、m2，n1，n2。
4. 将m1、n1；m2、n1；m1、n2；m2、n2递归调用第1步，分别得到结果AC(高位)、BC(中位)、AD(中位)、BD(低位)。
5. 判断BD位是否有进位bd，并截取bd得到保留位BD'；判断BC+AD+bd是否有进位abcd，并截取进位得到保留位ABCD'；判断AC+abcd是否有进位ac，并截取进位得到保留位AC'。
6. 返回最终大整数相乘的结果：ac AC' ABCD' BD'。

2.2.2 伪代码描述

伪代码：
```
function MULT(X，Y，n); {X和Y为2个小于2n的整数，返回结果为X和Y的乘积XY}
begin
S:=SIGN(X)*SIGN(Y); {S为X和Y的符号乘积}
X:=ABS(X);
Y:=ABS(Y); {X和Y分别取绝对值}
if n=1 then
```

```
if (X=1)and(Y=1) then return(S)
else return(0)
else begin
A:=X的左边n/2位;
B:=X的右边n/2位;
C:=Y的左边n/2位;
D:=Y的右边n/2位;
ml:=MULT(A,C,n/2);
m2:=MULT(A-B,D-C,n/2);
m3:=MULT(B,D,n/2);
S:=S*(m1*2n+(m1+m2+m3)*2n/2+m3);
return(S);
end;
end;
```

2.3 源代码

```
//
//  main.cpp
//  FileInputOutput
//
//  Created by bobobo on 11/8/15.
//  Copyright © 2015 bobobo. All rights reserved.
//
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <vector>
#include <iostream>
#include <fstream>
#include <cstdlib>
//#include "stdafx.h"
int n ;  //n组测试数据
const int MaxSize = 100000 + 10; //最大点数
const double infinity = 1e20;       //无穷大距离
struct Point
{
    int x, y;
    Point(int  a = 0, int b = 0) : x(a), y(b) {}
} v[MaxSize], tem[MaxSize];     //点集, 中间点集
struct PointPair
{
    int x1;
    int y1;
    int x2;
    int y2;
}vP[MaxSize];
using namespace std;

bool CompareX (const Point &p1, const Point &p2) { return p1.x < p2.x; }
bool CompareY (const Point &p1, const Point &p2) { return p1.y < p2.y; }

//计算欧几里得距离
inline double calDis(const Point &p1, const Point &p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y));
}
```

```
int findVertex(Point arr[],int n, int x,int y)
{
    int i = 0;
    int index = -1;
    for (i = 0; i < n; i++) {
        if((x == arr[i].x)&&(y == arr[i].y))
        {
            index = i;
            break;
        }
    }
    return index;
}
//寻找下标在[st, ed]之间的最近点对
double closestPair(int st, int ed, int &p1, int &p2)
{
    double dis = infinity, temdis;
    //int lenArray[MaxSize];
    if (st >= ed) return dis;
    //int m = ed - st;
    int mid = st + ((ed - st) >> 1);
    int t1 ,t2;
    if ((temdis = closestPair(st, mid, t1, t2)) < dis)  //左边
        dis = temdis, p1 = t1, p2 = t2;
    if ((temdis = closestPair(mid+1, ed, t1, t2)) < dis)//右边
        dis = temdis, p1 = t1, p2 = t2;

    //寻找距离中间x坐标小于dis的点，并按Y坐标排序
    int len = 0;
    for (int i = st; i <= ed; i++)
        if (fabs(v[i].x - v[mid].x) < dis)
        {
            tem[len++] = v[i];
            //lenArray[len++] = i;
        }
    sort(tem, tem + len, CompareY);

    //考虑每个点附近至多8个点，由距离控制
    for (int i = 0; i < len; i++)
        for (int j = i + 1; ((j < len )&& ((tem[j].y - tem[i].y) <= dis)); j++)
            if ((temdis = calDis(tem[i], tem[j])) < dis)
            {
                dis = temdis;
                p1 = findVertex(v, n, tem[i].x, tem[i].y);
                p2 = findVertex(v, n, tem[j].x, tem[j].y);
            }
    return dis;
}


int main()
{
    ifstream fin("/users/bobobo/Documents/AlgSevenProblems/FileInputOutput/in.dat",ios::inlios::binary);
    if(!fin)
    {
        cerr<<"open error!"<<endl;
        abort( );
    }
```

```
    fin.read((char *)&n, sizeof(n));
    ofstream fout("out.dat",ios::binary);
    if(!fout)
    {
        cerr<<"open error!"<<endl;
        abort( );//退出程序
    }
    int j;
    for(int i = 0 ; i < n ; i++)
    {
        fin.read((char *)&j, sizeof(j));//std::getline(fin,lineStr);
        //int j = std::stoi(lineStr);    //j=>这一组的测试数据的个数
        for (int k = 0; k < j; k++) {
            fin.read((char *)&v[k], sizeof(v[k]));
        }
        sort(v, v + j, CompareX);
        int k1, k2;
        double dis = closestPair(0, j - 1, k1, k2);
        vP[i].x1 = v[k1].x;
        vP[i].y1 = v[k1].y;
        vP[i].x2 = v[k2].x;
        vP[i].y2 = v[k2].y;
        fout.write((char *)&vP[i], sizeof(vP[i]));  //写进n组数据作为最小值
    }

}
```

2.4 测试分析

```
    while(cin>>a>>b>>op){
        //cout<<bigInt_mutiply(a, b)<<endl;
        switch (op) {
            case 1:
                cout<<add(a, b)<<endl;
                break;
            case 2:
                cout<<sub(a, b)<<endl;
                break;
            case 3:
                cout<<bigInt_mutiply(a, b)<<endl;
                break;
            default:
                cout<<"The Operator input error!"<<endl;
                break;
        }
    }
    //system("PAUSE");
    return 0;
}
```

BigInteger

3:乘 2:减 1:加

```
424239 4632492 3
1965283773588
6294932642934 439289 3
2765294665781833926
623794923 438473224324 1
439097019247
46829349324382 328402730883 2
46500946593499
647297493 424923382948 2
-424276085450
```

```
424239 4632492 3
1965283773588
6294932642934 439289 3
2765294665781833926
623794923 438473224324 1
439097019247
46829349324382 328402730883 2
46500946593499
647297493 424923382943 2
-424276085450
6439748933 4389473820 1
10829222753
4726493243284239 4639478320 1
4726497882762559
37723489264923 43843203820 2
37679646061103
437438347390940 6437434393 2
437431909956547
438430473 4734032094304 2
-4733593663831
748340324320 47384703223 3
35459884177706769283360
6439439292 4382974320302483270432 3
28223897053982804096792482614144
```

经测试，该算法能较好的实现大整数的加法 减法和乘法

2.5 技术总结

两个n位整数a，b相乘，一般有一下几种方法：

1.用小学的竖式运算，要用到两层for循环，时间复杂度是O（n^2）

2.用分治法，把整数拆分成两部分，一般是对称拆分

a = a1* 10^(n/2)+a0　b = b1* 10^(n/2)+b0
a*b = a1*b1*10^n +(a1*b0*+a0*b1)*10^(n/2)+a0*b0

拆分后需要对n/2位数做四次乘法运算，时间复杂度可表示为

n =1　T(n) = O(1)

n>1　T(n) = 4T(n/2) +O(n)

计算以后时间复杂度还是O（n^2）

3.还需要减少乘法的次数

(1)　a1*b0*+a0*b1 = (a1+a0)*(b1+b0)-a1*b1-a0*b0

(2)　a1*b0*+a0*b1 = (a1-a0)*(b0-b1)+a1*b1+a0*b0

简化后需要对n/2位数做三次乘法运算，时间复杂度可表示为

n =1　T(n) = O(1)

n>1　T(n) = 3T(n/2) +O(n)

计算以后时间复杂度是O(n^log3)=O(n^1.59),(1)中两数相加可能会得到m+1位的结果，增加了复杂度，所以选第二种方案。
对于位数不相同的大整数乘法，用上面的第2种方法比较简单。
a*b = a1*b1*10^n +(a1*b0*+a0*b1)*10^(n/2)+a0*b0，其中的几个乘法又可以考虑用拆分的方法递归求解。计算乘积的递归函数终止条件是两个乘数的位数都为1，这时可直接返回相乘结果；

3.Wooden Sticks

3.1题目描述

Description

There is a pile of n wooden sticks. The length and weight of each stick are known in advance. The sticks are to be processed by a woodworking machine in one by one fashion. It needs some time, called setup time, for the machine to prepare processing a stick. The setup times are associated with cleaning operations and changing tools and shapes in the machine. The setup times of the woodworking machine are given as follows:
(a) The setup time for the first wooden stick is 1 minute.
(b) Right after processing a stick of length l and weight w , the machine will need no setup time for a stick of length l' and weight w' if l <= l' and w <= w'. Otherwise, it will need 1 minute for setup.
You are to find the minimum setup time to process a given pile of n wooden sticks. For example, if you have five sticks whose pairs of length and weight are (9 , 4) , (2 , 5) , (1 , 2) , (5 , 3) , and (4 , 1) , then the minimum setup time should be 2 minutes since there is a sequence of pairs (4 , 1) , (5 , 3) , (9 , 4) , (1 , 2) , (2 , 5) .
Input

The input consists of T test cases. The number of test cases (T) is given in the first line of the input file. Each test case consists of two lines: The first line has an integer n , 1 <= n <= 5000 , that represents the number of wooden sticks in the test case, and the second line contains 2n positive integers l1 , w1 , l2 , w2 ,..., ln , wn , each of magnitude at most 10000 , where li and wi are the length and weight of the i th wooden stick, respectively. The 2n integers are delimited by one or more spaces.
Output

The output should contain the minimum setup time in minutes, one per line.

Sample Input

```
3
5
4 9 5 2 2 1 3 5 1 4
3
2 2 1 1 2 2
3
1 3 2 2 3 1
```

Sample Output

```
2
1
3
```

3.2 算法设计

道题的要求其实是将所有stick分为x个不下降子序列(Ai <= Ai+1)，然后问题归结于求x的最小值。

x的最小值其实等于按l递增排序后stick按w最长下降子序列的长度L，证明如下：

若x < L，先从stick中取出最长下降子序列L，取走的元素留下一个大小相同的“空穴”，然后将剩下的元素和空穴分成x个不下降子序列。接着把最长下降子序列L中的L个元素放回这L个空穴里。由于x < L，所以根据鸽笼原理，必然有两个或两个以上的下降子序列L中的元素(b > a)被按顺序放到同一个不下降子序列(a <= b)，产生矛盾（两者本应该是等效的),问题归结于求解最长下降子序列的长度L.

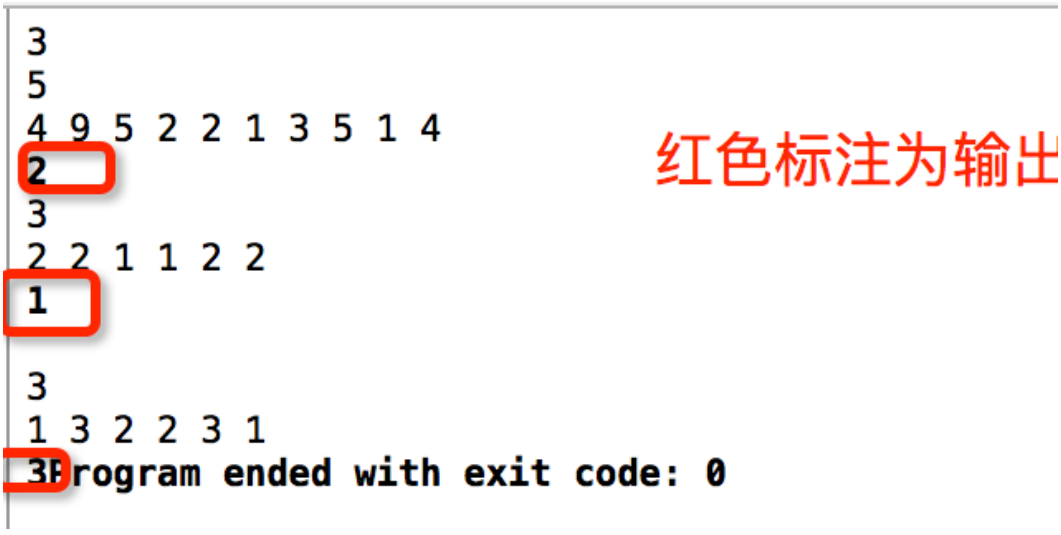
对于双关键字排序。讨论pair的second值，有如下结论：
某序列的最少上升子序列数就是该序列的最长下降子序列的长度。
如序列2 1 5 3 4，其至少可分成2个上升子序列{2,5}和{1,3,4}，但其最长下降子序列的长度也是2。
因而求出给定输入的最长下降子序列即可。

3.3 源代码

```
//
//  main.cpp
//  StickenLast
//
//  Created by bobobo on 11/14/15.
//  Copyright © 2015 bobobo. All rights reserved.
//

#include <iostream>
#include <cstdio>
#include <memory.h>
#include <algorithm>
using namespace std;

#define MAXN 10050

struct Node{
    int x,y;
}node[MAXN];


bool cmp(const Node& a,const Node& b)
{
    if(a.y == b.y)
        return a.x<b.x;
    return a.y<b.y;
}

int bsearch1(int c[],int n,Node a)
{
    int l = 1,r = n;
    while (l <= r) {
        int mid = (l+r)/2;
        if((c[mid] > a.x)&&(c[mid + 1] <= a.x))
            return mid+1;
        else if (c[mid] < a.x)
            r = mid - 1;
        else
            l = mid + 1;
    }
    return -1;
}
int LDS(Node a[],int n)
{
    int i,j,size = 1;
    int *c = new int[n+1];
    int *dp = new int [n+1];
    c[1] = a[0].x;
    dp[1] = 1;
    for (i = 1; i < n; i++) {
```

```
        if (a[i].x >= c[1])
            j = 1;
        else if(a[i].x < c[size])
            j = ++size;
        else
            j = bsearch1(c,size,*a);
        c[j] = a[i].x;
        dp[i] = j;
    }
    return size;
}

int main()
{
    int Num;
    scanf("%d",&Num);
    int n;
    while (Num--) {
        scanf("%d",&n);
        for (int i = 0; i < n; i++) {
            scanf("%d %d",&node[i].x,&node[i].y);
        }
        sort(node,node+n,cmp);
        int res = LDS(node, n);
        cout<<res;
    }
}
```

3.4 测试分析



3.5 技术总结

首先可以贪心，先按长度 l排序，如果l相同，按宽度w排序。

从i=0开始，每次把接下来的i+1 - n-1 的没有标记并且长度和宽度大于等于i这根木棍的长度和宽度标记起来。核心点在于如何求解最长下降子序列，可以使用动态规划法来求解，其实求解最长下降子序列和最长上升子序列是一种思路，以最长上升子序列为例：

先回顾经典的O(n^2)的动态规划算法，设A[t]表示序列中的第t个数，F[t]表示从1到t这一段中以t结尾的最长上升子序列的长度，初始时设 F[t] = 0(t = 1, 2, ..., len(A))。则有动态规划方程：F[t] = max{1, F[j] + 1} (j = 1, 2, ..., t - 1, 且A[j] < A[t])。

现在，我们仔细考虑计算F[t]时的情况。假设有两个元素A[x]和A[y]，满足

(1)x < y < t (2)A[x] < A[y] < A[t] (3)F[x] = F[y]

此时，选择F[x]和选择F[y]都可以得到同样的F[t]值，那么，在最长上升子序列的这个位置中，应该选择A[x]还是应该选择A[y]呢?

很明显，选择A[x]比选择A[y]要好。因为由于条件(2)，在A[x+1] ... A[t-1]这一段中，如果存在A[z]，A[x] < A[z] < a[y]，则与选择A[y]相比，将会得到更长的上升子序列。

再根据条件(3)，我们会得到一个启示：根据F[]的值进行分类。对于F[]的每一个取值k，我们只需要保留满足F[t] = k的所有A[t]中的最小值。设D[k]记录这个值，即D[k] = min{A[t]} (F[t] = k)。

注意到D[]的两个特点：

(1) D[k]的值是在整个计算过程中是单调不上升的。

(2) D[]的值是有序的，即D[1] < D[2] < D[3] < ... < D[n]。

利用D[]，我们可以得到另外一种计算最长上升子序列长度的方法。设当前已经求出的最长上升子序列长度为len。先判断A[t]与D[len]。若 A[t] > D[len]，则将A[t]接在D[len]后将得到一个更长的上升子序列，len = len + 1， D[len] = A[t]；否则，在D[1]..D[len]中，找到最大的j，满足D[j] < A[t]。令k = j + 1，则有D[j] < A[t] <= D[k]，将A[t]接在D[j]后将得到一个更长的上升子序列，同时更新D[k] = A[t]。最后，len即为所要求的最长上升子序列的长度。

在上述算法中，若使用朴素的顺序查找在D[1]..D[len]查找，由于共有 O(n)个元素需要计算，每次计算时的复杂度是O(n)，则整个算法的时间复杂度为O(n^2)，与原来的算法相比没有任何进步。但是由于D[]的特点 (2)，我们在D[]中查找时，可以使用二分查找高效地完成，则整个算法的时间复杂度下降为O(nlogn)，有了非常显著的提高。

4.Gone Fishing

4.1题目描述

Description

John is going on a fishing trip. He has h hours available (1 <= h <= 16), and there are n lakes in the area (2 <= n <= 25) all reachable along a single, one-way road. John starts at lake 1, but he can finish at any lake he wants. He can only travel from one lake to the next one, but he does not have to stop at any lake unless he wishes to. For each i = 1,...,n - 1, the number of 5-minute intervals it takes to travel from lake i to lake i + 1 is denoted ti (0 < ti <=192). For example, t3 = 4 means that it takes 20 minutes to travel from lake 3 to lake 4. To help plan his fishing trip, John has gathered some information about the lakes. For each lake i, the number of fish expected to be caught in the initial 5 minutes, denoted fi(fi >= 0), is known. Each 5 minutes of fishing decreases the number of fish expected to be caught in the next 5-minute interval by a constant rate of di (di >= 0). If the number of fish expected to be caught in an interval is less than or equal to di , there will be no more fish left in the lake in the next interval. To simplify the planning, John assumes that no one else will be fishing at the lakes to affect the number of fish he expects to catch.
Write a program to help John plan his fishing trip to maximize the number of fish expected to be caught. The number of minutes spent at each lake must be a multiple of 5.
Input

You will be given a number of cases in the input. Each case starts with a line containing n. This is followed by a line containing h. Next, there is a line of n integers specifying fi (1 <= i <=n), then a line of n integers di (1 <=i <=n), and finally, a line of n - 1 integers ti (1 <=i <=n - 1). Input is terminated by a case in which n = 0.
Output

For each test case, print the number of minutes spent at each lake, separated by commas, for the plan achieving the maximum number of fish expected to be caught (you should print the entire plan on one line even if it exceeds 80 characters). This is followed by a line containing the number of fish expected. If multiple plans exist, choose the one that spends as long as possible at lake 1, even if no fish are expected to be caught in some intervals. If there is still a tie, choose the one that spends as long as possible at lake 2, and so on. Insert a blank line between cases.
Sample Input

```
2
1
10 1
2 5
2
4
4
10 15 20 17
0 3 4 3
1 2 3
4
4
10 15 50 30
0 3 4 3
1 2 3
0
```

Sample Output

```
45, 5
Number of fish expected: 31
```

240, 0, 0, 0
Number of fish expected: 480

115, 10, 50, 35
Number of fish expected: 724

4.2 算法设计

由于每个湖都必须经过，且只经过一次，所以john花在路中的总时间是确定的。
在这个条件下，可以想成john学会了“瞬间移动”，即：他可以在任何时间，移动到任何他想去的湖，而移动的过程无需时间。
于是，john只需在每个5分钟的开始“瞬间移动”到当前5分钟中能钓到最多的鱼的湖中，且只钓5分钟的鱼。
这样可以保证john钓到尽可能多的鱼。
只要枚举john的行程是从第一个湖到第k个湖（1<=k<=n）,比较出最大的钓鱼数，就是题目所求的最佳方案。
需要注意的是要枚举从第一个湖走到最后一个湖的每一种情况，对结果相等这种情况做特殊处理。
思路：采用枚举加贪心的方法，枚举John在每一个湖停止钓鱼的最优解。先减去路上的耗时,然后优先钓鱼多的湖，直到时间耗尽。

4.3 源代码

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int MAXM=30;
int n,h;
int f[MAXM],d[MAXM],t[MAXM];
int each_time[MAXM];
int ans_time[MAXM];
int tf[MAXM];
int ans;
//初始化
void init()
{
    memset(each_time,0,sizeof(each_time));
    memset(ans_time,0,sizeof(ans_time));
    ans=0;
}
//输入
bool input()
{
    init();
    scanf("%d",&n);
    if(0==n) return false;
    scanf("%d",&h);
    for(int i=0;i<n;++i)
    {
        scanf("%d",f+i);
    }
    for(int i=0;i<n;++i)
    {
        scanf("%d",d+i);
    }
```

```
    for(int i=0;i<n-1;++i)
    {
        scanf("%d",t+i);
    }
    return true;
}

void solve(int rem_time,int q)
{
    int tans=0;
    memset(each_time,0,sizeof(each_time));

    memcpy(tf,f,sizeof(f));
    for(int i=0;i<rem_time;++i)//找鱼的数量最多的湖
    {
        int pos=0,max_fish=tf[0];
        if(q>0)
            for(int j=1;j<=q;++j)
            {
                if(max_fish<tf[j])
                {
                    max_fish=tf[j];
                    pos=j;
                }
            }
        ++each_time[pos];
        tans+=tf[pos];
        tf[pos]-=d[pos];
        if(tf[pos]<0) tf[pos]=0;

    }
    if(tans>ans)
    {
        ans=tans;
        memcpy(ans_time,each_time,sizeof(each_time));
    }
    else if(tans==ans)//细节，答案相等时做的处理。
    {
        int i;
        for(i=0;i<n;++i)
        {
            if(each_time[i]!=ans_time[i]) break;
        }
        if(each_time[i]>ans_time[i])
        {
            memcpy(ans_time,each_time,sizeof(each_time));
        }
    }
}

void print()
{
    printf("%d",ans_time[0]*5);
    for(int i=1;i<n;i++)
    {
        printf(", %d",ans_time[i]*5);
    }
    printf("\nNumber of fish expected: %d\n\n",ans);
}
int main()
```

```
{
    // freopen("in.txt","r",stdin);
    // freopen("out2.txt","w",stdout);
    while(input())
    {
        int rem_time=h*60;
        rem_time/=5;
        solve(rem_time,0);
        for(int i=0;i<n-1;++i)//枚举每个湖
        {
            rem_time-=t[i];
            solve(rem_time,i+1);
        }
        print();

    }

    return 0;
}
```

4.4 分析测试

```
4
4
10 15 30 50
0 3 4 3
1 2 3
105, 10, 25, 70
Number of fish expected: 774
```

```
2
1
10 1
2 5
2
45, 5
Number of fish expected: 31
```

```
4
4
10 15 20 17
0 3 4 3
1 2 3
240 0 0 0
Number of Fish Expected:480
```

4.5 技术总结

于是从第一个湖出发的，而且所有的湖都是一字排开的，所以只需枚举他走过的湖泊数X即可。即先假设他从湖1走到湖X，则路上总共花了T= T1 + T2 + T3 + ... + Tx。在这个前提下，就可以认为他有能力在1~X之间的任何两个湖之间“瞬移”，即在任一时刻可以任选一个1~X中的湖钓鱼。（想一想为什么？其实这跟汽车加油的道理是一样的，在每个湖的钓鱼顺序可以不是依次来的，你可能认为总时间肯定比这个花得多，其实不是的，顺序其实是不影响结果的。因为假如我要先去湖1钓5分钟，接着去湖2钓5分钟，再接着回来湖1钓5分钟，这个过程其实相当于先在湖1钓5+5=10分钟，然后再去湖2钓5分钟）。因此只

需一直贪心的选择当前能钓到鱼最多的湖即可。还有就是贪心选择的时候，若有相同的湖时，优先选择编号较小的湖。

5.Floyd-Warshall

5.1题目描述

每对结点之间的最短路径(Floyd-Warshall算法）

补充ALL-PATHS算法，增加前驱矩阵(Chp.25.2)，使得在求出结点间的最短路径长度矩阵A后，能够推导出每对结点间的最短路径

测试文件：

Input file：in.dat

格式：第一行为一个整数，表示测试用例的组数，其后跟相应组数的测试用例

每组测试用例包括：

首行：一个整数，表示本组测试用例包含的结点数n，其后跟n行

其后：每行n个整数，表示结点间邻接关系及边的长度（邻接成本矩阵）

边的长度<100，100即表示结点间没有边。

Output file：out.dat

格式：第一行为一个整数，表示测试用例的组数，其后跟相应组数的测试用例

每组测试用例输出包括：

首行：一个整数，表示本组测试用例包含的结点数n，其后跟$n+n^2$行

其后：开始的n行，每行n个整数，表示结点间最短路径的长度（A矩阵）。路径的程度<32767，32767即表示结点间没有可达的路径。其后n^2行，顺次输出结点对(1,1)、(1,2)、…、(1,n)，(2,1)、(2,2)、…、(2,n)、…(n,1)、…、(n,n)之间的最短路径结点序列，结点间用空格隔开。(i,i)输出i，若(i,j)之间没有路径，输出NULL。

5.2 算法设计

求矩阵A，A中的任一元素A(i,j)代表结点i到结点j的最短路径成本。利用动态规划策略求解，将求解成本矩阵G中每对结点之间的最短路径问题转化成一个多阶段决策过程。

动态规划求解策略的条件是满足的。

(1)最优性原理对于每对结点之间的最短路径问题成立

对G的一条由i到j的最短路径（假设该路径中不包含环），设k是该路径上的一个中间结点：

i,…,k,…j

由i到k的最短路径　k是中间结点　由k到i的最短路径,则,由i到k和k到j的两条子路径将分别是由i到k和由k到j的最短路径。（反证，否则i,...,k,…,j也将不是由i到j的最短路径） 故，最优性原理对于该问题成立.

(2)多阶段决策过程

约定：对所有n个结点从1到n依次编号。

设 k是由i到j的最短路径上编号最高的中间结点：

i,...,k,…,j

k是编号最高的中间结点

则由i到k的子路径上将不会有比编号k-1更大的中间结点；同理，由k到j的子路径上也将不会有比编号k-1更大的中间结点。

构造一个“多阶段决策过程”：对由i到j的最短路径，去寻求具有最大编号的中间结点k。

(3) 递推关系式

分析：对于 $A^n(i,j)$，路径可以经过结点n，也可以不经过结点n。

若该路径经过结点n，则

$$A^n(i,j) = A^{n-1}(i,n) + A^{n-1}(n,j)$$

若该路径不经过结点n，则

$$A^n(i,j) = A^{n-1}(i,j)$$ 故可得，

$$A^n(i,j) = \min\{A^{n-1}(i,j) , A^{n-1}(i,n) + A^{n-1}(n,j)\}$$

(4)算法描述 每对结点之间的最短路径长度

```
procedure ALL-PATHS(COST,A,n)
  //COST(n,n)是n结点图的成本邻接矩阵；A(i,j)是结点vi到vj的最短路径的成本；COST(i,i)=0,1≤i≤n//
  integer i,j,k,n; real COST(n,n),A(n,n)
  for i←1 to n do
    for j←1 to n do
      A(i,j) ←COST(i,j)    //用COST(i,j)对A0赋初值//
    repeat
  repeat
  for k←1 to n do          //k控制Ak (i,j) //
    for i←1 to n do
      for j←1 to n do
        A (i,j) ← min{A (i,j) ， A (i,k) + A (k,j)}
      repeat
    repeat
  repeat
end ALL-PATHS
```

5.3 源代码

```
//
//  main.cpp
//  FlyodAlgorithm
//
//  Created by bobobo on 11/13/15.
//  Copyright © 2015 bobobo. All rights reserved.
//

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

#define MaxVertice 100
```

```
#define MaxEdge 50
#define oo 65535   //表示两点间距离为无穷大

typedef int PointPath [MaxVertice][MaxVertice];
typedef int ShortPathTable [MaxVertice][MaxVertice];

typedef struct
{
    int vertice[MaxVertice];
    int lngTable[MaxVertice][MaxVertice];
    int numVertice,numEdge;
} MatrixGraph;

void createMatrixMap(MatrixGraph *m)
{
    int i ,j,n;
    cout<<"Please input the Vertice Number"<<endl;
    scanf("%d",&n);
    m->numVertice = n;
    for ( i = 0; i < n; i++) {
        m->vertice[i] = i;
    }
    //初始化
    for (i = 0; i < n; i++)
        for (j = 0;  j < n; j++) {
            if (i == j) {
                m->lngTable[i][j] = 0;
            }else
            {
                m->lngTable[i][j] = oo;
            }
        }
    //输入数据
    for (int i = 0;  i < n; i++)
        for (int j = 0; j < n; j++) {
            if (i == j) {
                m->lngTable[i][j] = 0;
            }else
            {
                cout<<"input the length of distance of("<<i<<","<<j<<")"<<endl;
                cin>>m->lngTable[i][j];
            }
        }

}

void Flyod(MatrixGraph m1,PointPath d, ShortPathTable p)
{
    for (int i = 0; i < m1.numVertice; i++)
        for (int j = 0; j < m1.numVertice; j++) {
            d[i][j] = m1.lngTable[i][j];
            p[i][j] = j;
        }
    for (int k = 0; k < m1.numVertice; k++)
        for (int i = 0; i < m1.numVertice; i++)
            for(int j = 0;j < m1.numVertice;j++)
            {
                if (d[i][j] > (d[i][k] + d[k][j])) {
                    d[i][j] = d[i][k] + d[k][j];
```

```
                p[i][j] = p[i][k];  //路径设置经过下标为k的端点
            }
        }

}
int main(int argc, const char * argv[]) {
    // insert code here...
    ifstream fin("/users/bobobo/Documents/OcTrial/FlyodAlgorithm/FlyodAlgorithm/in.dat",ios::binaryl
ios::in);
    if(!fin)
    {
        cerr<<"open error!"<<endl;
        abort();
    }
    int n ;
    fin.read((char *)&n, sizeof(n));
    MatrixGraph m[n];
    PointPath d[n];
    ShortPathTable p[n];
    //有n组测试项
    //fin.read((char *)&n, sizeof(n));
    ofstream fout("out.dat",ios::binarylios::out);

    fout.write((char *)&n, sizeof(n));
    for (int i = 0; i < n; i++) {
        int j;
        fin.read((char *)&j, sizeof(j));
        //本组的测试点数
        fout.write((char *)&j,sizeof(j));
        m[i].numVertice = j;
        for (int a = 0; a < j; a++) {
            fin.read((char *)&(m[i].lngTable[a]), sizeof(m[i].lngTable[a]));
        }
        for (int b = 0 ;b < j; b++) {
            m[i].lngTable[b][b] = 0;
        }
            Flyod(m[i], d[i], p[i]);
            //n 行数据 每行也有n个数据 代表最短距离 n  * n矩阵 每个节点值代表图中的两点间最短距离
            for (int e = 0;  e < j; e++) {
                fout.write((char *)&(d[i][e]), sizeof(d[i][e])); //写入一行数据
            }
        for (int x = 0; x < j; x++) {
            fout.write((char *)&(p[i][x]), sizeof(p[i][x]));//写入一行数据
        }
            // 前驱矩阵代表其中
     }
    fin.close();
    fout.close();
    return 0;
}
```

5.4 分析测试

输入数据如下图所示

```
The Number of The Group:100
the Number of the datas:6
the x:4the y :8
the x:0the y :8
the x:1the y :8
the x:9the y :1
the x:3the y :6
the x:3the y :4
the Number of the datas:1
the x:2the y :7
the Number of the datas:4
the x:3the y :8
the x:3the y :8
the x:7the y :6
the x:6the y :7
the Number of the datas:6
the x:7the y :2
the x:1the y :6
the x:7the y :2
the x:9the y :8
the x:0the y :7
the x:0the y :9
the Number of the datas:4
the x:2the y :6
the x:1the y :5
the x:6the y :9
the x:6the y :2
the Number of the datas:9
the x:1the y :6
the x:5the y :9
the x:4the y :6
the x:1the y :4
the x:2the y :0
the x:7the y :5
the x:6the y :9
the x:8the y :8
the x:6the y :4
```

输出数据如下图第一组输入

```
the number of test group is:6
 the test group - 0-points-5
0 4 5 5 7
1 0 6 4 3
5 7 0 7 8
1 5 5 0 2
2 6 3 7 0
(0,0):0
(0,1):NULL
(0,2):NULL
(0,3):NULL
(0,4):1 4
(1,0):NULL
(1,1):1
(1,2):0 2
(1,3):NULL
(1,4):NULL
(2,0):NULL
(2,1):NULL
(2,2):2
(2,3):NULL
(2,4):NULL
(3,0):NULL
(3,1):0 1
(3,2):4 2
(3,3):3
(3,4):NULL
(4,0):NULL
(4,1):0 1
(4,2):NULL
(4,3):0 3
(4,4):4
```

算法的输入和输出是正确的，求得了最小距离并且输出了每个最短距离的前驱矩阵，如上图，即为给出最短路径上的各个点。

5.5 技术总结

在第k-1到第k次的迭代过程中，A的第k行、第k列元素不变

$A^k(i,k) = A^{k-1}(i,k)$

$A^k(k,j) = A^{k-1}(k,j)$

(1)i>k且k>j,则有

$A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^k(k,j)\}$ $A^k(k,j) \rightarrow A^{k-1}(k,j)$

(2)k>i且j>k,则有

$A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^k(i,k) + A^{k-1}(k,j)\}$　$A^k(i,k)$ -> $A^{k-1}(i,k)$

(3)k<i且k<j,则有

$A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^k(i,k) + A^k(k,j)\}$

$A^k(i,k)$ -> $A^{k-1}(i,k)$

$A^k(k,j)$ -> $A^{k-1}(i,k)$ $A^{k-1}(k,j)$

$A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^k(i,k) + A^{k-1}(k,j)\}$

$A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^k(k,j)\}$

$A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^k(i,k) + A^k(k,j)\}$

$\equiv A^k(i,j) \leftarrow \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$

∴ 在算法的计算过程中取消了A的上标，并保证了每次计算的$A^k(i,j)$即为

$$\min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$$

(4)性能分析： 计算时间

$\Theta(n^3)$

```
for k←1 to n do                    迭代n次
  for i←1 to n do                  迭代n次
    for j←1 to n do                迭代n次
      A (i,j) ← min{A (i,j) ， A (i,k) + A (k,j)}
    repeat
  repeat
repeat
```

6.Corn Fields

6.1题目描述

Description

Farmer John has purchased a lush new rectangular pasture composed of M by N (1 ≤ M ≤ 12; 1 ≤ N ≤ 12) square parcels. He wants to grow some yummy corn for the cows on a number of squares. Regrettably, some of the squares are infertile and can't be planted. Canny FJ knows that the cows dislike eating close to each other, so when choosing which squares to plant, he avoids choosing squares that are adjacent; no two chosen squares share an edge. He has not yet made the final choice as to which squares to plant.

Being a very open-minded man, Farmer John wants to consider all possible options for how to choose the squares for planting. He is so open-minded that he considers choosing no squares as a valid option! Please help Farmer John determine the number of ways he can choose the squares to plant.

Input

Line 1: Two space-separated integers: M and N
Lines 2..M+1: Line i+1 describes row i of the pasture with N space-separated integers indicating whether a square is fertile (1 for fertile, 0 for infertile)
Output

Line 1: One integer: the number of ways that FJ can choose the squares modulo 100,000,000.
Sample Input

2 3
1 1 1
0 1 0
Sample Output

9
Hint

Number the squares as follows:
1 2 3
  4

There are four ways to plant only on one squares (1, 2, 3, or 4), three ways to plant on two squares (13, 14, or 34), 1 way to plant on three squares (134), and one way to plant on no squares. 4+3+1+1=9.

6.2 算法设计

运用状态压缩方法，这道题由于列数较小、所以将行压缩成二进制来看、首先处理第一行、先判断同一行中不能有相邻的1出现、然后判断1出现的位置要与题目中的不冲突、接下来就是转移了、从上一行转移到这一行、首先判断上下不能有1相邻、然后就是将上一行的状态转移到当前行、上一行的所有符合条件的状态的总的方案数就是当前行该状态的方案数。
解法->设d[i][j]表示第i行状态为j的情况下，能放的牛的数量。d[i][j] += d[i-1][k]，其中k表示能转移到j的状态，num(j)表示状态为j的行所放的牛的数量。当然，还要判断一下状态j是不是能放在第i行。

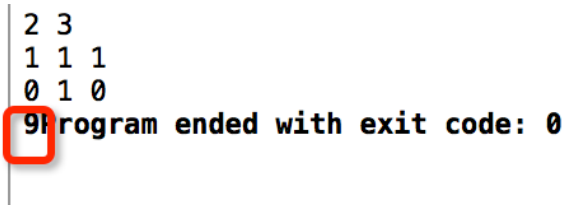
6.3 源代码

```
//
//  main.cpp
//  CornFields
```

```
//
//  Created by bobobo on 11/14/15.
//  Copyright © 2015 bobobo. All rights reserved.
//

#include <iostream>
#include <cstdio>
#include <queue>
#include <algorithm>
#include <cstring>
using namespace std;

int n,m; //m  行 n  列
int row[12];//12行 12种状态 1000011110110101诸如此类 1可以种植 0不可以
int nState,state[1000];
int dp[14][1000];
#define MOD 100000000

void init()
{
    int k = 1 << n;
    nState = 0;
    for (int i = 0; i < k; i++) {
        if ((i&(i << 1)) == 0) {
            //相邻之间不能同时为1
            state[nState++] = i;
        }
    }
}
int main(int argc, const char * argv[]) {
    int k;
    scanf("%d %d",&m,&n);
    init();

    //输入每一行的草地状态
    for (int i = 0; i < m; i++) {
        row[i] = 0;
        for (int j = n - 1; j >= 0; j--) {
            scanf("%d",&k);
            row[i] += k << j;
        }
    }
    //满足第0行草地 的状态
    for (int i = 0; i < nState; i++) {
        dp[0][i] = ((row[0] & state[i]) == state[i])?1:0;
    }

    // 求dp[i]，如果state[j]和第i-1行的状态state[k]不冲突，则dp[i][j]=Σ(dp[i-1][k])
    for(int  i = 1;i < m;i++)
        for (int j = 0; j < nState; j++) {
            if ((row[i] & state[j]) != state[j])
                continue;
            for (int l = 0; l < nState; l++) {
                if ((dp[i - 1][l]) && ((state[l] & state[j]) == 0)) {
                    dp[i][j] = (dp[i - 1][l] + dp[i][j])%MOD;
                }
            }
        }
```

```
    int result = 0;
    for (int j = 0; j < nState; j++) {
        if (dp[m - 1][j]) {
            result += dp[m - 1][j];
            result = result % MOD;
        }
    }
    cout<<result;
    return 0;
}
```

6.4 测试分析



根据OJ平台上给定的输入：
2 3
1 1 1
0 1 0
可以得到相应的输出 -> 9.

6.5 技术总结

输入m行n列的数字，其中为1或者是0

1表示土壤肥沃可以种植草地，0则不可以。

在种草地的区域可以放牛，但相邻的两 块区域不允许同时放牛，问有多少种放牛的方法？ （不放牛也算一种情况）

分析
由m,n<=12,可用状态压缩

对于第i行，可以放草的格子置为0，不可以种草的格子设置为1，整一行的状态存入graph[i]中

对于每一行，放牛的格为1，不放牛的格为0，整行用一个二进制数表示 dp[i][j]表示第i行放牛状态为j时有多少种方法，易知：

首先j必须合法，即左右相邻两位不同时出现1，
不能在不能种草的地方放牛，即j&graph[i]==0
dp[i][j] = SUM(dp[i-1][k]),其中k&j==0,即上下相邻位置不放牛
由此，可以求出所有的dp[i][j]，那么放牛的种类共有 = SUM(dp[n-1][j])最后一行所有状态的放牛种类之和.

7.Paid Roads

7.1题目描述

Description

A network of m roads connects N cities (numbered from 1 to N). There may be more than one road connecting one city with another. Some of the roads are paid. There are two ways to pay for travel on a paid road i from city ai to city bi:

in advance, in a city ci (which may or may not be the same as ai);
after the travel, in the city bi.
The payment is Pi in the first case and Ri in the second case.

Write a program to find a minimal-cost route from the city 1 to the city N.

Input

The first line of the input contains the values of N and m. Each of the following m lines describes one road by specifying the values of ai, bi, ci, Pi, Ri ($1 \le i \le m$). Adjacent values on the same line are separated by one or more spaces. All values are integers, $1 \le m$, $N \le 10$, $0 \le Pi$, $Ri \le 100$, $Pi \le Ri$ ($1 \le i \le m$).

Output

The first and only line of the file must contain the minimal possible cost of a trip from the city 1 to the city N. If the trip is not possible for any reason, the line must contain the word ‘impossible’.

Sample Input

4 5
1 2 1 10 10
2 3 1 30 50
3 4 3 80 80
2 1 2 10 10
1 3 2 10 50
Sample Output

110

7.2 算法设计

有n座城市和m（1<=n，m<=10）条路。 现在要从城市1到城市n。 有些路是要收费的，从a城市到b城市，如果之前到过c城市，那么只要付P的钱，如果没有去过就付R的钱。 求的是最少要花多少钱。

注意：路径是有向的。
解题思路：

DFS。 这题当有了思路后，做起来是没有难度的，但是思维推算能力要求很高。

这题难点在于“城市与城市之间可能存在多条路径”：

1、 输入数据时可能会出现多条从城市a到城市b的路径信息，但是费用有所差别；

2、 对于从城市a到城市b 的同一条路径，允许重复走。

重复走同一条路径只是单纯增加费用而已，为什么不能标记所有路径，每条路只允许走一次，这样费用不是更少么?
先来看一组数据：

6 5

1 2 1 10 10

2 3 4 10 100

2 4 2 15 15

4 1 1 12 12

3 6 6 10 10

如果每条路只允许走一次，那么方案只有1个：

共135元

但这组数据的正确答案是67元。 为什么？ 正确的方案如下：

共67元

显然1－2重复走了一次，目的是为了先到达城市4，从而使得2－3这段路的费用从100缩减到10元。

但是问题马上又来了。 如果同一条路允许重复走，那么就不能标记了，但一旦不标记，失去了搜索的限制条件，DFS就无法结束，不是陷入死循环了?

这种思路“对一半，错一半”，“对”是对在“重复走会增加费用”，“错”是错在“重复走的对象不是某一条路，而是某一个环路”。 在同一个环路重复走才会真正增加费用。 但是标记环路是很麻烦的，那么能不能根据某一条路或某一个城市重复走过的次数来判断当前所走的方案已经出现了环路？ 答案是可以的。

上述的例子已经验证过了，同一条路可以重复走，但是不能无限重复走，重复的次数是有限的。 那么应该重复多少次才合理？ 这与m值有关。 题目的m值范围为<=10，那么当人一个城市被到达的次数若>3次（不包括3），所走的方案必然出现了环路。

7.3 源代码

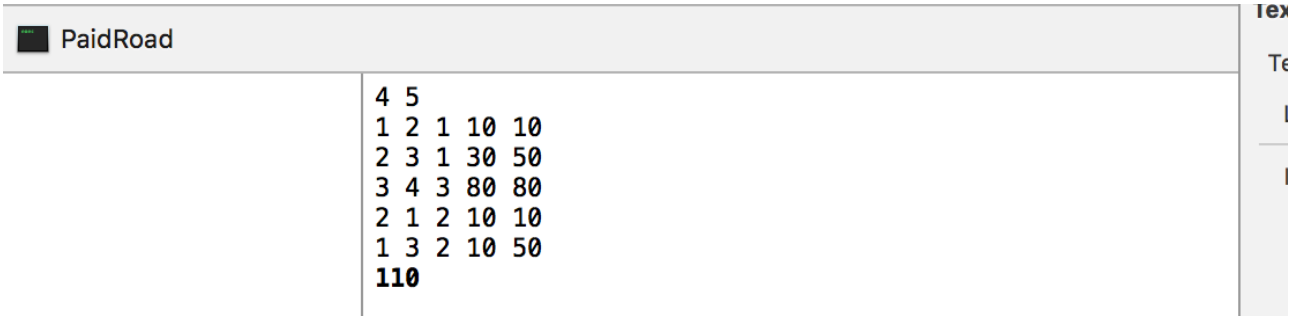
```
//  main.cpp
//  PaidPaidRoad
//
//  Created by bobobo on 11/14/15.
//  Copyright © 2015 bobobo. All rights reserved.
//

#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int inf=(unsigned)((1 << 31) - 1);
//int map[15][15];
int vis[15];
int n,m;
```

```
int ans;
struct node
{
    int a;
    int b;
    int c;
    int p;
    int r;
}PaidRoad[15];
void dfs(int u,int v)
{
    if(v>ans)
        return;
    if(u==n)
    {
        if(v<ans)
        {
            ans=v;
            // printf("  %d\n",ans);
        }
        return;
    }
    for(int i=1;i<=m;i++)
    {
        if(PaidRoad[i].a==u&&vis[PaidRoad[i].b]<=3)
        {
            int b=PaidRoad[i].b;
            vis[b]++;
            if(vis[PaidRoad[i].c])
            {
                dfs(b,v+PaidRoad[i].p);
            }
            else
            {
                dfs(b,v+PaidRoad[i].r);
            }
            vis[b]--;
        }
    }
    return;
}
int main()
{
    while(~scanf("%d%d",&n,&m))
    {
        memset(vis,0,sizeof(vis));
        ans=inf;
        for(int i=1;i<=m;i++)
        {
            scanf("%d%d%d%d%
%d",&PaidRoad[i].a,&PaidRoad[i].b,&PaidRoad[i].c,&PaidRoad[i].p,&PaidRoad[i].r);
        }
        vis[1]=1;
        dfs(1,0);
        if(ans==inf)
            printf("impossible\n");
        else
            printf("%d\n",ans);
    }
    return 0;
```

}

7.4 测试分析

输入按照OJ平台给出 输出也符合要求



7.5 技术总结

因为可以来回走，所以不能用单纯的最短路，可以用二维SPFA，状态压缩一下，第二维来记录状态，表示到过这个点的第几个状态。也可以用DFS，因为最多十个点，所以如果走某一个点走过三遍说明就是真的只增加费用了，也就是真正的在走环路了，DFS分析。

OJ截图

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
14910893	bobowangwangwang	3411	Accepted	132K	0MS	C++	1329B	2015-11-14 21:37:31
14910438	bobowangwangwang	3254	Accepted	184K	0MS	C++	1547B	2015-11-14 18:54:38
14910396	bobowangwangwang	1042	Accepted	132K	47MS	C++	2150B	2015-11-14 18:36:30
14904874	bobowangwangwang	1065	Accepted	212K	16MS	C++	1010B	2015-11-12 20:05:09