P1002 过河卒

棋盘上A点有一个过河卒，需要走到目标B点。卒行走的规则：可以向下、或者向右。同时在棋盘上C点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。棋盘用坐标表示，A点(0,0)、B点(n,m)(n, m为不超过20的整数)，同样马的位置坐标是需要给出的。

现在要求你计算出卒从A点能够到达B点的路径的条数，假设马的位置是固定不动的，并不是卒走一步马走一步。

示例:

```
输入: 6 6 3 3
输出:6
```

思路

动态规划，得出状态转移方程即可。

$f[i][j]=max(f[i-1][j]+f[i][j-1],f[i][j])$

Solution

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}
const int fx[]={0,-2,-1,1,2,2,1,-1,-2};
const int fy[]={0,1,2,2,1,-1,-2,-2,-1};
//马可以走到的位置
int bx,by,mx,my;
ull ans;
ull f[30][30];//f[i][j]代表从A点到(i,j)会经过的线路数
bool s[30][30];//判断这个点有没有马盯着
int main(){
    bx = read();
    by = read();
    mx = read();
```

```
    my = read();
    ++bx; ++by; ++mx; ++my;
    //坐标+1以防越界
    f[1][1]=1;//初始化
    s[mx][my]=1;//标记马的位置
    for(int i=1;i<=8;i++)
        s[ mx + fx[i] ][ my + fy[i] ]=1;
    for(int i=1;i<=bx;i++){
        for(int j=1;j<=by;j++){
            if(s[i][j])continue;
            f[i][j]=max( f[i][j] , f[i-1][j] + f[i][j-1] );
            //状态转移方程
        }
    }
    printf("%llu\n",f[bx][by]);
    return 0;
}
```

P1003 铺地毯

为了准备一个独特的颁奖典礼，组织者在会场的一片矩形区域（可看做是平面直角坐标系的第一象限）铺上一些矩形地毯。一共有n张地毯，编号从1到n。现在将这些地毯按照编号从小到大的顺序平行于坐标轴先后铺设，后铺的地毯覆盖在前面已经铺好的地毯之上。

地毯铺设完成后，组织者想知道覆盖地面某个点的最上面的那张地毯的编号。注意：在矩形地毯边界和四个顶点上的点也算被地毯覆盖。

输入共n+2行

第一行，一个整数n，表示总共有n张地毯

接下来的n行中，第i+1行表示编号i的地毯的信息，包含四个正整数a,b,g,k，每两个整数之间用一个空格隔开，分别表示铺设地毯的左下角的坐标(a,b)以及地毯在xx轴和yy轴方向的长度

第n+2行包含两个正整数x和y，表示所求的地面的点的坐标(x,y)

输出格式：

输出共1行，一个整数，表示所求的地毯的编号；若此处没有被地毯覆盖则输出-1

示例:

```
输入：
3
1 0 2 3
0 2 3 3
2 1 3 3
2 2
输出:3
输入：
3
```

```
1 0 2 3
0 2 3 3
2 1 3 3
4 5
输出:-1
```

思路

简单，比较点值即可。

Solution

```cpp
#include<iostream>
#include<cstring>
#include<cstdio>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}

int main(){
    int n = read();
    int a[n+1];
    int b[n+1];
    int g[n+1];
    int k[n+1];
    for (int i = 1; i <= n; i++) {
        a[i] = read();
        b[i] = read();
        g[i] = read();
        k[i] = read();
    }

    int x = read();
    int y = read();
    int result = -1;

    for (int i = 1; i <= n; i++) {
        if(a[i] <=x && a[i] + g[i] >= x && b[i] <=y && b[i] + k[i] >= y) {
            result = i;
        }
    }
```

```
    cout<<result;
    return 0;
}
```

P1004 方格取数

设有N×N的方格图(N≤9)，我们将其中的某些方格中填入正整数，而其他的方格中则放入数字0。如下图所示（见样例）：

A 0 0 0 0 0 0 0 0 0 0 13 0 0 6 0 0 0 0 0 0 7 0 0 0 0 0 0 14 0 0 0 0 0 21 0 0 0 4 0 0 0 0 15 0 0 0 0 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 B

某人从图的左上角的A点出发，可以向下行走，也可以向右走，直到到达右下角的B点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字0）。此人从A点到B点共走两次，试找出2条这样的路径，使得取得的数之和为最大。

输入格式：

输入的第一行为一个整数N（表示N×N的方格图），接下来的每行有三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的0表示输入结束。

输出格式：

只需输出一个整数，表示2条路径上取得的最大的和。

示例:

```
输入：
8
2 3 13
2 6  6
3 5  7
4 4 14
5 2 21
5 6  4
6 3 15
7 2 14
0 0  0
输出：
67
```

思路

这题，是四维动规的模板题。

我们考虑两个人同时走，就相当于数字三角形。

用fijkl表示第一个人走到(i,j)，第二个人走到(k,l)的最优解

状态转移方程为：

$ f[i][j][k][l]=\max (f[i-1][j][k-1][l], f[i-1][j][k][l-1], f[i][j-1][k], f[i][j-1]]+a[i][j]+a[k][l] $

不过要判断i=k&&j=l的情况。

Solution

```
#include<iostream>
    #include<cstdio>
    #include<algorithm>
    using namespace std;
    int f[12][12][12][12],a[12][12],n,x,y,z;
    int main() {
        cin>>n>>x>>y>>z;
        while(x!=0||y!=0||z!=0){
            a[x][y]=z;
            cin>>x>>y>>z;
        }
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                for(int k=1;k<=n;k++){
                    for(int l=1;l<=n;l++){
                        f[i][j][k][l]=max(max(f[i-1][j][k-1][l],f[i-1][j]
[k][l-1]),max(f[i][j-1][k-1][l],f[i][j-1][k][l-1]))+a[i][j]+a[k][l];
                        if(i==k&&l==j)f[i][j][k][l]-=a[i][j];
                    }
                }
            }
        }
        cout<<f[n][n][n][n];
        return 0;
}
```

P1005 矩阵取数游戏

帅帅经常跟同学玩一个矩阵取数游戏：对于一个给定的n×m的矩阵，矩阵中的每个元素ai,j均为非负整数。游戏规则如下：

- 每次取数时须从每行各取走一个元素，共n个。经过m次后取完矩阵内所有元素
- 每次取走的各个元素只能是该元素所在行的行首或行尾；
- 每次取数都有一个得分值，为每行取数的得分之和，每行取数的得分 = 被取走的元素值×2^i，其中i表示第i次取数（从1开始编号）；
- 游戏结束总得分为m次取数得分之和。

帅帅想请你帮忙写一个程序，对于任意矩阵，可以求出取数后的最大得分。

输入格式：

输入文件包括n+1行：

第1行为两个用空格隔开的整数n和m。

第2~n+1行为n×m矩阵，其中每行有m个用单个空格隔开的非负整数。

输出格式：

输出文件仅包含1行，为一个整数，即输入矩阵取数后的最大得分。

示例:

```
输入：
2 3
1 2 3
3 4 2
输出：
82
```

思路

- 求nn行最大得分和，每一行取数又不会影响到其他行，那么只要确保每一行得分最大，管好自家孩子就行了。（这个在动规中叫**最优子结构**）
- 每次取数是在边缘取，那么每次取数完剩下来的元素一定是在一个完整的一个区间中，又是求最优解，**区间DP**应运而生。

DP流程

(每次DP仅针对第TT行)

状态

- 我们用fi,jfi,j表示区间变为[i,j]时，获得的最大分数。

转移

- 当区间变为[i,j]时，上一次取数的时候区间一定是[i−1,j]或[i,j+1]，从这两个状态转移即可。在第m−j+i−1次(这个请自行模拟)取走了Ai−1,j或Ai,j+1即：

  $ f_{i, j}=\max \left\{f_{i-1, j}+A_{i-1, j} \cdot 2^{m-j+i-1}, f_{i, j+1}+A_{i, j+1} \cdot 2^{m-j+i-1}\right\} $

终值（答案）

- 啊这个终值超级讨厌，状态不明确的话还真想不出来。
- **因为题目中说要取完，但是空区间是DP不出来的，然后就得手动模拟每个长度为11的区间。即：**

  $ A n s=m a x_{i \leq m}\left\{f_{i, i}+A_{i, i} \cdot 2^{m}\right\} $

- **高精度**

Solution

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

using namespace std;

const int MAXN = 85, Mod = 10000; //高精四位压缩大法好
int n, m;
int ar[MAXN];

struct HP {
    int p[505], len;
    HP() {
        memset(p, 0, sizeof p);
        len = 0;
    } //这是构造函数，用于直接创建一个高精度变量
    void print() {
        printf("%d", p[len]);
        for (int i = len - 1; i > 0; i--) {
            if (p[i] == 0) {
                printf("0000");
                continue;
            }
            for (int k = 10; k * p[i] < Mod; k *= 10)
                printf("0");
            printf("%d", p[i]);
        }
    } //四位压缩的输出
} f[MAXN][MAXN], base[MAXN], ans;

HP operator + (const HP &a, const HP &b) {
    HP c; c.len = max(a.len, b.len); int x = 0;
    for (int i = 1; i <= c.len; i++) {
        c.p[i] = a.p[i] + b.p[i] + x;
        x = c.p[i] / Mod;
        c.p[i] %= Mod;
    }
    if (x > 0)
        c.p[++c.len] = x;
    return c;
} //高精+高精

HP operator * (const HP &a, const int &b) {
    HP c; c.len = a.len; int x = 0;
    for (int i = 1; i <= c.len; i++) {
```

```
        c.p[i] = a.p[i] * b + x;
        x = c.p[i] / Mod;
        c.p[i] %= Mod;
    }
    while (x > 0)
        c.p[++c.len] = x % Mod, x /= Mod;
    return c;
} //高精*单精

HP max(const HP &a, const HP &b) {
    if (a.len > b.len)
        return a;
    else if (a.len < b.len)
        return b;
    for (int i = a.len; i > 0; i--)
        if (a.p[i] > b.p[i])
            return a;
        else if (a.p[i] < b.p[i])
            return b;
    return a;
} //比较取最大值

void BaseTwo() {
    base[0].p[1] = 1, base[0].len = 1;
    for (int i = 1; i <= m + 2; i++){ //这里是m! m! m! 我TM写成n调了n年...
        base[i] = base[i - 1] * 2;
    }
} //预处理出2的幂

int main(void) {
    scanf("%d%d", &n, &m);
    BaseTwo();
    while (n--) {
        memset(f, 0, sizeof f);
        for (int i = 1; i <= m; i++)
            scanf("%d", &ar[i]);
        for (int i = 1; i <= m; i++)
            for (int j = m; j >= i; j--) { //因为终值是小区间，DP自然就从大区间
开始
                f[i][j] = max(f[i][j], f[i - 1][j] + base[m - j + i - 1] *
ar[i - 1]);
                f[i][j] = max(f[i][j], f[i][j + 1] + base[m - j + i - 1] *
ar[j + 1]);
            } //用结构体重载运算符写起来比较自然
        HP Max;
        for (int i = 1; i <= m; i++)
            Max = max(Max, f[i][i] + base[m] * ar[i]);
        ans = ans + Max; //记录到总答案中
    }
```

```
    ans.print(); //输出
    return 0;
}
```

P1006 传纸条

小渊和小轩是好朋友也是同班同学，他们在一起总有谈不完的话题。一次素质拓展活动中，班上同学安排做成一个m行n列的矩阵，而小渊和小轩被安排在矩阵对角线的两端，因此，他们就无法直接交谈了。幸运的是，他们可以通过传纸条来进行交流。纸条要经由许多同学传到对方手里，小渊坐在矩阵的左上角，坐标(1,1)，小轩坐在矩阵的右下角，坐标(m,n)。从小渊传到小轩的纸条只可以向下或者向右传递，从小轩传给小渊的纸条只可以向上或者向左传递。

在活动进行中，小渊希望给小轩传递一张纸条，同时希望小轩给他回复。班里每个同学都可以帮他们传递，但只会帮他们一次，也就是说如果此人在小渊递给小轩纸条的时候帮忙，那么在小轩递给小渊的时候就不会再帮忙。反之亦然。

还有一件事情需要注意，全班每个同学愿意帮忙的好感度有高有低（注意：小渊和小轩的好心程度没有定义，输入时用0表示），可以用一个0-100的自然数来表示，数越大表示越好心。小渊和小轩希望尽可能找好心程度高的同学来帮忙传纸条，即找到来回两条传递路径，使得这2条路径上同学的好心程度之和最大。现在，请你帮助小渊和小轩找到这样的2条路径。

输入格式：

输入文件，第一行有2个用空格隔开的整数m和n，表示班里有m行n列。

接下来的m行是一个m×n的矩阵，矩阵中第i行j列的整数表示坐在第i行j列的学生的好心程度。每行的n个整数之间用空格隔开。

输出格式：

输出文件共一行，包含一个整数，表示来回2条路上参与传递纸条的学生的好心程度之和的最大值。

示例:

```
输入：
3 3
0 3 9
2 8 5
5 7 0
输出：
34
```

思路

- 同7.23日的1004题

Solution

```
#include <iostream>
#define maxn 55
```

```
using namespace std;
int f[maxn][maxn][maxn][maxn],a[maxn][maxn];
int n,m;
int max_ele(int a,int b,int c,int d){
    if (b>a)
        a = b;
    if (c>a)
        a = c;
    if (d>a)
        a = d;
    return a;
}
int main(){
    cin >> n >> m;
    for (int i=1;i<=n;i++)
        for (int j=1;j<=m;j++)
                cin >> a[i][j];
    for (int i=1;i<=n;i++)
        for (int j=1;j<=m;j++)
            for (int k=1;k<=n;k++)
                for (int l=j+1;l<=m;l++)
                    f[i][j][k][l]=max_ele(f[i][j-1][k-1][l],f[i-1][j][k][l-
1],f[i][j-1][k][l-1],f[i-1][j][k-1][l])+a[i][j]+a[k][l];
    cout << f[n][m-1][n-1][m] << endl;
    return 0;
}
```

P1007 独木桥

战争已经进入到紧要时间。你是运输小队长，正在率领运输部队向前线运送物资。运输任务像做题一样的无聊。你希望找些刺激，于是命令你的士兵们到前方的一座独木桥上欣赏风景，而你留在桥下欣赏士兵们。士兵们十分愤怒，因为这座独木桥十分狭窄，只能容纳1个人通过。假如有2个人相向而行在桥上相遇，那么他们2个人将无妨绕过对方，只能有1个人回头下桥，让另一个人先通过。但是，可以有多个人同时呆在同一个位置。

突然，你收到从指挥部发来的信息，敌军的轰炸机正朝着你所在的独木桥飞来！为了安全，你的部队必须撤下独木桥。独木桥的长度为L，士兵们只能呆在坐标为整数的地方。所有士兵的速度都为1，但一个士兵某一时刻来到了坐标为0或L+1的位置，他就离开了独木桥。

每个士兵都有一个初始面对的方向，他们会以匀速朝着这个方向行走，中途不会自己改变方向。但是，如果两个士兵面对面相遇，他们无法彼此通过对方，于是就分别转身，继续行走。转身不需要任何的时间。

由于先前的愤怒，你已不能控制你的士兵。甚至，你连每个士兵初始面对的方向都不知道。因此，你想要知道你的部队最少需要多少时间就可能全部撤离独木桥。另外，总部也在安排阻拦敌人的进攻，因此你还需要知道你的部队最多需要多少时间才能全部撤离独木桥。

输入格式：

第一行：一个整数L，表示独木桥的长度。桥上的坐标为1...L

第二行：一个整数N，表示初始时留在桥上的士兵数目

第三行：有N个整数，分别表示每个士兵的初始坐标。

输出格式：

只有一行，输出2个整数，分别表示部队撤离独木桥的最小时间和最大时间。2个整数由一个空格符分开。

示例：

```
输入：
4
2
1 3
输出：
2 4
```

思路

- 首先自行脑补一下，假装你正在20000米高空的轰炸机上用高倍~~显微镜~~望远镜默默欣赏士兵离开，你会发现什么东西？一堆花花绿绿的迷彩服在移动。（不是鬼片！不是鬼片！不是鬼片！重要的事情说三遍）

  那么当两个士兵撞在一起时，从你的视角看会发生什么？当然他们认为他们都掉头了，但因为你在特高的地方，你会认为他们“穿过”了对方。换言之，这与他们相互穿过并没有任何区别。

  然后我们就可以把士兵分开了。比方说有一个士兵在位置3，开始时向右，那么一定有一个士兵在两秒后在位置5。虽然这两个家伙可能不是同一个人，但由于士兵都是相同的，我们可以认为他们相同。

  那么我们就可以把所有士兵分开。首先，我们把他们一个个读进去。然后，对于每一个士兵，他有向左和向右两种选择。设士兵在位置p，如果向左，需要p时间单位；向右，需要L-p+1个。分别取max和min，更新答案即可。

Solution

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}
int main(){
```

```
    int l = read();
    int n = read();
    int max_result = 0;
    int min_result = 0;
    int solider;
    for (int i = 0; i < n; i++) {
        solider = read();
        max_result = max(max_result, max(l-solider+1,solider));
        min_result = max(min_result, min(l-solider+1,solider));
    }
    cout << min_result << " " << max_result;
    return 0;
}
```

P1008 三连击

本题为提交答案题，您可以写程序或手算在本机上算出答案后，直接提交答案文本，也可提交答案生成程序。

将1,2,…,9共9个数分成3组，分别组成3个三位数，且使这3个三位数构成1:2:3的比例，试求出所有满足条件的

3个三位数。

输入格式：

无。

输出格式：

若干行，每行3个数字。按照每行第1个数字升序排列。

示例:

```
无
```

思路

- 简单枚举即可

Solution

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
```

```
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}


int main(){
    cout << 192 << " " << 384 << " " << 576 << endl;
    cout << 219 << " " << 438 << " " << 657 << endl;
    cout << 273 << " " << 546 << " " << 819 << endl;
    cout << 327 << " " << 654 << " " << 981 << endl;

    return 0;
}
```

P1009 阶乘之和

用高精度计算出S=1!+2!+3!+...+n!(n≤50)其中“!”表示阶乘，例如：5!=5×4×3×2×1。

输入格式：

一个正整数N

输出格式：

一个正整数S，表示计算结果。

示例:

```
无
```

思路

- 高精度问题。
- python打表也可。

Solution

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int a[2000];
int b[2000];
int c[2000];
int sum[2000];
void pplus(int *a,int *c)
{
    int jw=0;
```

```
    for(int i=1;i<=1000;i++)
    {
        c[i]+=a[i]+jw;
        jw=c[i]/10;
        c[i]%=10;
    }
}
void cheng(int *a,int c)
{
    int jw=0;
    for(int i=1;i<=1000;i++)
    {
        a[i]=a[i]*c+jw;
        jw=a[i]/10;
        a[i]%=10;
    }
}
int main()
{
    int n;
    cin>>n;
    a[1]=1;
    for(int i=1;i<=n;i++)
    {
        cheng(a,i);
        pplus(a,c);
    }
    bool flag=0;
    for(int i=1000;i>=1;i--)
    {
        if(c[i]!=0) flag=1;
        if(flag) cout<<c[i];
    }
}
```

P1010 幂次方

任何一个正整数都可以用2的幂次方表示。例如

$ 137=2^{7}+2^{3}+2^{0} $

同时约定方次用括号来表示，即a^b可表示为a(b)。

由此可知，137可表示为：$$2(7)+2(3)+2(0)$$

进一步，$$2(2(2)+2+2(0))+2(2+2(0))+2(0)$$

所以1315可以表示为$$2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)$$

输入格式：

一个正整数N

输出格式：

符合约定的n的0,2表示(在表示中不能有空格)

示例:

```
无
```

思路

- 二进制
- 递归

Solution

```
#include<iostream>
#include<cstdio>
using namespace std;
void mici(int n){
    if(n==1) return;                    //递归终止条件
    if(n==0) { printf("0"); return ; }    //递归终止条件
    for(int i=16, mask=0x00008000, first=1; i>=1; i--){
        if(mask&n){                       //位运算进行分解
            if(!first) printf("+");       //第一次不输出加号
            printf("2");
            if(i!=2) printf("(");        //2^1时不需要括号
            mici(i-1);                    //递归幂次
            if(i!=2) printf(")");        //2^1时不需要括号
            if(first) first=0;            //控制加号的变量
        }
        mask>>=1;                         //掩码右移一位
    }
}
int main(){
    int n;
    scanf("%d", &n);
    mici(n);
    return 0;
}
```

P1011 车站

火车从始发站（称为第1站）开出，在始发站上车的人数为a，然后到达第2站，在第2站有人上、下车，但上、下车的人数相同，因此在第2站开出时（即在到达第3站之前）车上的人数保持为a人。从第3站起（包括第3站）上、下车的人数有一定规律：上车的人数都是前两站上车人数之和，而下车人数等于上一站上车人数，一直到终点站的前一站（第n−1站），都满足此规律。现给出的条件是：共

有N个车站，始发站上车的人数为a，最后一站下车的人数是m（全部下车）。试问x站开出时车上的人数是多少?

输入格式：

a(≤20)，n(≤20)，m(≤2000)，和x(≤20)，

输出格式：

从x站开出时车上的人数。

示例:

```
5 7 32 4
13
```

思路

- 斐波那契数列
- 在做之前，我们先找找规律：
  1. 第一站：上车 a人；车上有 a 人；
  2. 第二站：假设上车 u 人，则下车 u 人；车上仍然是a人；
  3. 第三站：上车人数等于前两站上车人数之和：a+u 人，下车人数等于上次上车人数 u 人；净上车人数为 a 人；车上有 2a 人；
  4. 第四站：上车人数 =a+2u，下车人数 =a+u；净上车人数 =u；车上有多少人呢？就是 2a+u；
  5. 第五站：上车人数 =2a+3u，下车人数 =a+2u，净上车人数 =a+u；车上有 3a+2u人；
  6. 第六站：上车人数 =3a+5u，下车 2a+3u 人，净上车人数 =a+2u；车上有 4a+4u 人.....

  这里不必在列下去了，发现规律了吗?

  将第三站净上车人数看作x1，第四站看作x2，第五站为x3，第六站为x4，有 x1+x2=x3, x2+x3=x4...x1+x2=x3, x2+x3=x4...**斐波那契数列**
- 对a和u的系数进行存储

Solution

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}
```

```cpp
int main(){
    int a = read();
    int n = read();
    int m = read();
    int x = read();

    int A[n+1];
    int U[n+1];

    int sum_A = 0;
    int sum_U = 0;

    int u = 0;
    A[1] = 0; U[1] = 0;
    A[2] = 0; U[2] = 0;
    A[3] = 1; U[3] = 0;
    A[4] = 0; U[4] = 1;

    if(n >= 5) {
        for (int i = 5; i <= n; i++) {
            A[i] = A[i-1] + A[i-2];
            U[i] = U[i-1] + U[i-2];
        }

        for (int i = 1; i <= n - 1; i++) {
            sum_A += A[i];
            sum_U += U[i];
        }


        u = (m  - sum_A*a - a)/sum_U;
    }


    sum_A = 0;
    sum_U = 0;
    for (int i = 1; i <= x ; i++) {
        sum_A += A[i];
        sum_U += U[i];
    }

    int result = a*sum_A + u*sum_U + a;

    cout<<result;
    return 0;
}
```

P1012 拼数

设有n个正整数(n≤20)，将它们联接成一排，组成一个最大的多位整数。

例如：n=3时，3个整数13,312,343联接成的最大整数为：34331213

又如：n=4时，4个整数7,13,4,246联接成的最大整数为：7424613

输入格式：

第一行，一个正整数n。

第二行，n个正整数。

输出格式：

一个正整数，表示最大的整数

示例:

```
无
```

思路

- 按照string来做

Solution

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}



int main(){
    int n = read();
    string a[n+1];
    for (int i = 1 ; i <= n; i++) {
        cin >> a[i];
    }
    for(int i=1;i<n;i++) //排序
```

```
    {
        for(int j=i+1;j<=n;j++)
        {
            if(a[j]+a[i]>a[i]+a[j]) //string类型可以直接比较大小
            {
                swap(a[j],a[i]);  //交换a[i]与a[j],同样可以用swap(a[i],a[j]);
                /*
                    举个例子:如果是12,34，由于3412>1234，所以肯定要交换一下，直接用
swap就可以了
                */
            }
        }
    } //排序结束
    for(int i=1;i<=n;i++) cout<<a[i]; //由于已经排好序了，直接输出即可
    return 0;
}
```

P1013 进制位

著名科学家卢斯为了检查学生对进位制的理解，他给出了如下的一张加法表，表中的字母代表数字。例如：

```
+    L    K    V    E
L    L    K    V    E
K    K    V    E    KL
V    V    E    KL    KK
E    E    KL    KK     KV
```

其含义为：

$ \begin{array}{l}{L+L=L, L+K=K, L+V=V, L+E=E} \\ {K+L=K, K+K=V, K+V=E, K+E=K L} \\ {\ldots \ldots E+E=K V}\end{array} $

根据这些规则可推导出：

输入格式：$$L=0, \quad K=1, \quad V=2, \quad E=3$$

同时可以确定该表表示的是4进制加法。

n(n≤9)表示行数。

以下n行，每行包括n个字符串，每个字串间用空格隔开。（字串仅有一个为‘+’号，其它都由大写字母组成）

输出格式：

- 各个字母表示什么数，格式如：L=0，K=1，……按给出的字母顺序。
- 加法运算是几进制的。
- 若不可能组成加法表，则应输出“ERROR!”

示例:

无

思路

- 一定是n-1进制
- 每一行有几个二位数，这个数就是几

Solution

```
#include<bits/stdc++.h>
#define fu(i,q,w) for(register int i=q;i<=w;i++)
#define fd(i,q,w) for(register int i=q;i>=w;i--)
using namespace std;
typedef long long ll;
inline int read(){
    int ret=0,f=1;char c;
    while((c=getchar())<'0'||c>'9')if(c=='-')f=-1;
    while(c>='0'&&c<='9')ret=ret*10+(c-'0'),c=getchar();
    return ret*f;
}
char word[10];//记录字母
char check[10];//检查重复
string numx,numy;//储存输入数据、检查重复
map<char,int> two;//一行中两位数个数
map<char,int> tone;//存字母在两位数个位出现几次
int n;
void in(){
    n=read();
    cin>>numx;//"+"特判输入
    fu(i,1,n-1){cin>>numx,word[i]=numx[0];}// 第一行存表头的每个字母
    fu(i,1,n-1)//从第二行开始
    fu(j,1,n){cin>>numx;

        if(j!=1&&j!=2)//表头不算
        if(numx==numy){printf("ERROR!");exit(0);}//发现重复输入一定不对
        numy=numx; //前后比，不要全行比
        if(numx.size()==2){//统计两位数个数
            two[word[i]]++;tone[numx[1]]++;
        }
    }
}
void solve(){
    fu(i,1,n-1)
    if(two[word[i]]!=n-2-tone[word[i]]){printf("ERROR!");exit(0);}
    //比较两种算法的结果是否相同
    fu(i,1,n-1)
    cout<<word[i]<<'='<<two[word[i]]<<' ';
    printf("\n");
```

```
    printf("%d",n-1);

    }
int main(){
    in();
    solve();
    return 0;

}
```

P1014 Cantor表

题目描述：

现代数学的著名证明之一是Georg Cantor证明了有理数是可枚举的。他是用下面这一张表来证明这一命题的：

$ \begin{array}{l}{1 / 1,1 / 2,1 / 3,1 / 4,1 / 5, \ldots} \\ {2 / 1,2 / 2,2 / 3,2 / 4, \ldots} \\ {3 / 1,3 / 2,3 / 3, \ldots} \\ {4 / 1,4 / 2, \ldots} \\ {5 / 1, \ldots}\end{array} $

我们以Z字形给上表的每一项编号。第一项是1/1，然后是1/2，2/1.....

输入格式：

整数N（1≤N≤10000000）

输出格式：

表中的第N项

示例:

```
无
```

思路

- 简单

Solution

```
#include<cstdio>
int main()
{
    int n;scanf("%d",&n);
    int t=1,ans=0;//t是表示下一次跳到下一次的距离，ans是表示第几层
    while(1)
    {
        if(n>t){n-=t;ans++;t++;}//printf("%d\n",ans);
        else if(n==t&&ans%2==0){printf("1/%d",ans+1);break;}
        //如果在n==t，并且为偶数层，就在第一行 第ans+1个
```

```
        else if(n==t&&ans%2!=0){printf("%d/1",ans+1);break;}
        //如果在n==t，并且为奇数层，就在第ans+1行 第一个
        else if(n<t&&ans%2!=0){printf("%d/%d",ans+n-t+1,t-n+1);break;}
        //如果在n<t，并且为奇数层，t-n+1表示该层最后一个往后走n-1步，ans+n-t+1示该层最后一个往上走t-1步
        else if(n<t&&ans%2==0){printf("%d/%d",t-n+1,ans+n-t+1);break;}
        // 如果在n<t，并且为偶数层，t-n+1表示该层最后一个往上走n-1步，ans+n-t+1示该层最后一个往后走t-1步
    }
    return 0;
}
```

P1015 回文数

题目描述：

若一个数（首位不为零）从左向右读与从右向左读都一样，我们就将其称之为回文数。

例如：给定一个十进制数56，将56加65（即把56从右向左读），得到121是一个回文数。

又如：对于十进制数87：

STEP1：87+78=165

STEP2：165+561=726

STEP3：726+627=1353

STEP4：1353+3531=4884

在这里的一步是指进行了一次N进制的加法，上例最少用了4步得到回文数4884。

写一个程序，给定一个N(2≤N≤10,N=16)进制数M(100位之内)，求最少经过几步可以得到回文数。如果在30

步以内（包含30步）不可能得到回文数，则输出Impossible!

输入格式：

两行，分别是N，M

输出格式：

STEP=ans

示例:

```
无
```

思路

- 高精度加法 + 回文数
- 读入数字，并将这个数字的每一位存入 x 数组（反着存，即高位在后，低位在前，从低位到高位）。如果是**十六进制**（原来没有看到N=16导致百思不得其解），将ABCDEF分别转换成

10,11,12,13,14,15再存入数组，这样方便接下来的高精度计算

- while循环。循环条件：不是回文数且步数不大于30，将这个数（x数组）的长度赋给变量 l，并将 x [i] 和 x [l-i+1] 相加并将值赋给 sum [i]，并加以进位判断等
- 判断回文（返回布尔值的函数find，返回true即为回文，返回false即为非回文）
-

Solution

```
#include <bits/stdc++.h>
using namespace std;
int n, q[1000001], l, w[1000001], ans;
string s;
void init()
{
    int j = 0;
    for(int i = s.length() - 1; i >= 0 ; i--)
    {
        if(s[i] >= '0' && s[i] <= '9')
        {
            q[++j] = s[i] - '0';
        }
        else
        {
            q[++j] = s[i] - 'A' + 10;
        }
    }
}
void add(int a[], int b[])
{
    for(int i = 1; i <= l; i++)
    {
        a[i] += b[i];
        a[i + 1] += a[i] / n;
        a[i] %= n;
    }
    if(a[l + 1] > 0)
    {
        l++;
    }
}
bool f(int a[])
{
    int ln = l;
    int i = 1;
    int j = l;
    while(ln--)
    {
        if(ln < l / 2)
        {
```

```
            break;
        }
        if(a[i] != a[j])
        {
            return false;
        }
        i++;
        j--;
    }
    return true;
}
void turn(int a[])
{
    int j = 0;
    for(int i = l; i >= 1; i--)
    {
        w[++j] = a[i];
    }
}
int main()
{
    cin>>n>>s;
    init();
    l = s.length();
    while(!f(q))
    {
        turn(q);
        add(q, w);
        ans++;
        if(ans > 30)
        {
            break;
        }
    }
    if(ans > 30)
    {
        printf("Impossible!");
    }
    else
    {
        printf("STEP=%d", ans);
    }
    return 0;
}
```

P1016 旅行家的预算

题目描述：

一个旅行家想驾驶汽车以最少的费用从一个城市到另一个城市（假设出发时油箱是空的）。给定两个城市之间的距离D1、汽车油箱的容量C（以升为单位）、每升汽油能行驶的距离D2、出发点每升汽油价格P和沿途油站数N（N可以为零），油站i离出发点的距离Di、每升汽油价格Pi（i=1,2,...,N）。计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出“No Solution”。

输入格式：

第一行，D1，C，D2，P，N。

接下来有N行。

第i+1行，两个数字，油站i离出发点的距离Di和每升汽油价格Pi。

输出格式：

所需最小费用，计算结果四舍五入至小数点后两位。如果无法到达目的地，则输出“No Solution”。

示例:

```
275.6 11.9 27.4 2.8 2
102.0 2.9
220.0 2.2

26.95
```

思路

- 先确定每个节点间的距离，start为0节点，end为n+1节点。
- 若存在节点间的距离d[i]-d[i-1]大于D*c，则No Solution；其余情况，必有解。
- 在每个节点，我们有三种选择：
    - 如果指定的goal还未完成，继续行驶
    - 若到达了goal，则向前找第一个邮价比自己低的油站（end处油站油价为0）
        - 能够开往（distance<=D*c）,则将goal定为它
        - 不够开往（distance>D*c）,将油加满，将goal定为在加满情况下所能开到的最远油站。

Solution

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<string>
#include<cmath>
#include<algorithm>
using namespace std;
double d[600]={0},p[600];
double s,c,D,p1;
int n;
double money=0,oil=0;
```

```
double locin(int goal,int now)
{
    if(now==n+1)
    {
        printf("%.2lf",money);
        return 0;
    }
    if(now<goal)
    {
        oil-=(d[now+1]-d[now])/D;
        locin(goal,now+1);
        return 0;
    }
    if(now==goal)
    {
        int k=now+1;
        while(p[k]>p[now])k++;
        if(d[k]-d[now]<=D*c)
        {
            if(oil<(d[k]-d[now])/D)
            {
            money+=((d[k]-d[now])/D-oil)*p[now];
            oil=(d[k]-d[now])/D;
            }
            locin(k,now);
            return 0;
        }
        else
        {
            int w=now+1;
            while(d[w]-d[now]<=D*c)w++;
            w--;
            money+=(c-oil)*p[now];
            oil=c;
            locin(w,now);
            return 0;
        }
    }
}
int main()
{
    cin>>s>>c>>D>>p1>>n;
    p[0]=p1;
    for(int i=1;i<=n;++i)
    {
    cin>>d[i]>>p[i];
    if(d[i]-d[i-1]>D*c)
    {
        cout<<"No Solution";
```

```
            return 0;
        }
        }
        d[n+1]=s;p[n+1]=0;
        if(d[n+1]-d[n]>D*c)
        {
            cout<<"No Solution";
            return 0;
        }
        locin(0,0);
        return 0;
    }
```

P1017 进制转换

题目描述：

我们可以用这样的方式来表示一个十进制数: 将每个阿拉伯数字乘以一个以该数字所处位置的(值减1)为指数,以10

为底数的幂之和的形式。例如:123可表示为$$1 \times 10^{2}+2 \times 10^{1}+3 \times 10^{0}$$这样的形式。

与之相似的,对二进制数来说,也可表示成每个二进制数码乘以一个以该数字所处位置的(值−1)为指数,以2为底数的幂之和的形式。一般说来,任何一个正整数R或一个负整数−R都可以被选来作为一个数制系统的基数。如果是以R或

−R为基数,则需要用到的数码为0,1,....R−1。例如,当R=7时,所需用到的数码是0,1,2,3,4,5和6,这与其是R或−R无关。如果作为基数的数绝对值超过10,则为了表示这些数码,通常使用英文字母来表示那些大于9的数码。例如对16进制数来说,用A表示10,用B表示11,用C表示12,用D表示13,用E表示14,用F表示15。

在负进制数中是用−R作为基数,例如−15(十进制)相当于110001(−2进制),并且它可以被表示为2的幂级数的和数:

$ 110001=1 \times(-2)^{5}+1 \times(-2)^{4}+0 \times(-2)^{3}+0 \times(-2)^{2}+0 \times(-2)^{1}+1 \times(-2)^{0} $

设计一个程序,读入一个十进制数和一个负进制数的基数, 并将此十进制数转换为此负进制下的数:

−R∈−2,−3,−4,...,−20

输入格式：

输入的每行有两个输入数据。

第一个是十进制数N(−32768≤N≤32767)

第二个是负进制数的基数−R。

输出格式：

结果显示在屏幕上，相对于输入，应输出此负进制数及其基数，若此基数超过10，则参照16进制的方式处理。

示例:

```
30000 -2
30000=11011010101110000(base-2)
```

思路

- 负整数取模后是正整数取模的相反数，例如-15对-2取模得到-1，因此我们要用它减去模数，即为1.
- 负数的短除法

Solution

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

void zhuan(int n,int r)
{
    if(n==0) return ;
    int m=n%r;//m为余数

    if(m<0) m-=r,n+=r;//如果余数小于0，转化为正数

    //将余数转化为ascll码方便输出，省略了一个数组
    if(m>=10) m='A'+m-10;
    else m+='0';

    zhuan(n/r,r);

    printf("%c",m);//注意，因为结果为余数倒序，输出要写在递归后面，不然会顺序输出
    return ;
}
int main()
{
    //freopen("in.txt","r",stdin);
    int n,r;
    string ans="";
    cin>>n>>r;
    cout<<n<<"=";
    zhuan(n,r);
    printf("(base%d)",r);
    return 0;
}
```

P1018 乘积最大

题目描述：

今年是国际数学联盟确定的“2000——世界数学年”，又恰逢我国著名数学家华罗庚先生诞辰90周年。在华罗庚先生的家乡江苏金坛，组织了一场别开生面的数学智力竞赛的活动，你的一个好朋友XZ也有幸得以参加。活动中，主持人给所有参加活动的选手出了这样一道题目：

设有一个长度为N的数字串，要求选手使用K个乘号将它分成K+1个部分，找出一种分法，使得这K+1个部分的乘积能够为最大。同时，为了帮助选手能够正确理解题意，主持人还举了如下的一个例子：

有一个数字串：312， 当N=3,K=1时会有以下两种分法：

1、3×12=36 2、31×2=62

这时，符合题目要求的结果是: 31×2=62

现在，请你帮助你的好朋友XZXZ设计一个程序，求得正确的答案。

输入格式：

程序的输入共有两行：

第一行共有2个自然数N,K（6≤N≤40,1≤K≤6）

第二行是一个长度为N的数字串。

输出格式：

结果显示在屏幕上，相对于输入，应输出所求得的最大乘积（一个自然数）。

示例:

```
4  2
1231

62
```

思路

- dp[k][i]表示前i个数，加入k个乘号时的最大值
- 我们需要枚举最后一个乘号是在哪里放的
- 转移时，直接计算一下最后一个乘号之前的数字加入k-1个乘号时的最大值，再乘以最后一个乘号之后的数字

Solution

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

```
const int maxn=100;
int n,K;
string s;
int a[maxn];
struct lxt
{
    int len;
    int ans[maxn];
}dp[maxn/10][maxn];

lxt cal(lxt x,int l,int r)
{
    lxt Ans,y;
    memset(Ans.ans,0,sizeof(Ans.ans));
    memset(y.ans,0,sizeof(y.ans));
    y.len=r-l+1;
    for(int i=r;i>=l;--i) y.ans[r-i+1]=a[i];
    int l1=x.len,l2=y.len,ll;
    for(int i=1;i<=l1;++i)
      for(int j=1;j<=l2;++j)
          Ans.ans[i+j-1]+=x.ans[i]*y.ans[j];
    ll=l1+l2-1;
    for(int i=1;i<=ll;++i)
    {
        Ans.ans[i+1]+=Ans.ans[i]/10;
        Ans.ans[i]=Ans.ans[i]%10;
    }
    if(Ans.ans[ll+1]) ll++;
    Ans.len=ll;
    return Ans;
}
lxt cmp(lxt x,lxt y)
{
    int lx=x.len,ly=y.len;
    if(lx<ly) return y;
    if(lx>ly) return x;
    for(int i=lx;i>=1;--i)
    {
        if(x.ans[i]>y.ans[i]) return x;
        if(x.ans[i]<y.ans[i]) return y;
    }
    return x;
}
int main()
{
    scanf("%d%d",&n,&K);
    cin>>s;
    for(int i=1;i<=n;++i) a[i]=s[i-1]-'0';
```

```
    for(int i=1;i<=n;++i)
      for(int j=i;j>=1;--j)
        dp[0][i].ans[++dp[0][i].len]=a[j];
    for(int i=2;i<=n;++i)
      for(int k=1;k<=min(K,i-1);++k)
        for(int j=k;j<i;++j)
          dp[k][i]=cmp(dp[k][i],cal(dp[k-1][j],j+1,i));
    for(int i=dp[K][n].len;i>=1;--i)
      printf("%d",dp[K][n].ans[i]);
    return 0;
}
```

P1019 单词接龙

题目描述：

单词接龙是一个与我们经常玩的成语接龙相类似的游戏，现在我们已知一组单词，且给定一个开头的字母，要求出以这个字母开头的最长的“龙”（每个单词都最多在“龙”中出现两次），在两个单词相连时，其重合部分合为一部分，例如beast和astonish，如果接成一条龙则变为beastonish，另外相邻的两部分不能存在包含关系，例如at和

atide间不能相连。

输入格式：

输入的第一行为一个单独的整数n(n≤20)表示单词数，以下n 行每行有一个单词，输入的最后一行为一个单个字符，表示“龙”开头的字母。你可以假定以此字母开头的“龙”一定存在.

输出格式：

只需输出以此字母开头的最长的“龙”的长度

示例:

```
5
at
touch
cheat
choose
tact
a

23
```

思路

- 两个单词合并时，合并部分取的是**最小重叠部分**
- **相邻的两部分不能存在包含关系**就是说如果存在包含关系，就不能标记为使用过。
- 每个单词最多出现两次.

- 首先是预处理，用yc[i][j]来存储 第i个单词 后连接 第j个单词 的 最小重叠部分（mt函数）
- 后来预处理完了之后就是深搜:先从第一个到最后一个单词看一看哪个单词是指定字母为开头的，作为深搜的第一个单词，同时标记使用过一次（vis[i]++）

  然后继续搜吧。

Solution

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#define maxn 100
using namespace std;
int n;
int ans = 0;
string word[maxn];//字符串数组，用来存储单词
string beginn;//用来存储开头字符
int used[maxn];//这个就是用来记录dfs时候每一个单词被使用了几次的数组

bool check(string s,string m,int k){//重点一，check函数判断接口可行性，k代表接口长度，以下同
    int lens = s.length();
    for (int i=0;i<k;i++){
        if(s[lens-k+i]!=m[i])//我讲过了
            return false;
    }
    return true;
}

void add(string &s,string m,int k){//拼接操作，因为要把m接到s上，所以对于s串不可以传参，因为我们要试图改变这个串
    int lenm = m.length();
    for (int i=k;i<lenm;i++)
        s+=m[i];//C++字符串类型特性操作，支持+=符号直接拼接
}

void dfs(string now){//这只是一个看似平淡无奇的dfs
    int x = now.length();
    ans = max(ans,x);//每次拼接之后更新一下答案
    for (int i=1;i<=n;i++){
        if (used[i]>=2)//如果有一个单词用完了，那这个单词就不能选了
            continue;
        int maxk = word[i].length();
        for (int j=1;j<=maxk;j++){//枚举接口长度
            if (check(now,word[i],j)){
                string temp = now;//重点二，使用字符串副本进行拼接
                add(temp,word[i],j);
```

```
                if (temp==now)//拼完之后如果发现长度没增加，也就是和原串一样，那这次拼接没有意义，剪掉
                    continue;
                used[i]++;
                dfs(temp);
                used[i]--;//这只是一个看似平淡无奇的回溯
            }
        }
    }

}

int main(){
    cin >> n;
    for (int i=1;i<=n;i++)
        cin >> word[i];
    cin >> beginn;
    dfs(beginn);

    cout << ans << endl;
    return 0;

}
```

P1020 导弹拦截

题目描述：

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是≤50000的正整数），计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

输入格式：

1行，若干个整数（个数≤100000）

输出格式：

2行，每行一个整数，第一个数字表示这套系统最多能拦截多少导弹，第二个数字表示如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

示例:

```
389 207 155 300 299 170 158 65

6
2
```

思路

- 对于问一直接用O（n*logn）的方法求最长不升子序列即可
- 对于问二求整个数列的最长上升子序列即可

Solution

```
#include<algorithm>
#include<cstdio>
#include<cstring>
using namespace std;
int n=0,a[100001],f[100001],d[100001],ans=1,t=0;
int main() {
    while(~scanf("%d",&a[++n]));//读入数据方法
    n--;//n是导弹数，由于某些原因要--
    for(int i=1; i<=n; i++) {
        f[i]=1;
        for(int j=t; j>0; j--)
            if(a[i]<=a[d[j]]) {
                f[i]=f[d[j]]+1;
                break;
            }
        t=max(t,f[i]);
        d[f[i]]=i;//简单的维护过程
        ans=max(ans,f[i]);
    }
    printf("%d\n",ans);
    ans=1;
    t=0;
    for(int i=1; i<=n; i++) {
        f[i]=1;
        for(int j=t; j>0; j--)
            if(a[i]>a[d[j]]) {
                f[i]=f[d[j]]+1;
                break;
            }
        t=max(t,f[i]);
        d[f[i]]=i;
        ans=max(ans,f[i]);
    }
    printf("%d",ans);
}
```

P1021 邮票面值设计

题目描述：

给定一个信封，最多只允许粘贴N张邮票，计算在给定K（N+K≤15）种邮票的情况下（假定所有的邮票数量都足够），如何设计邮票的面值，能得到最大值MAX，使在1至MAX之间的每一个邮资值都能得到。

例如，N=3，K=2，如果面值分别为1分、4分，则在1分～6分之间的每一个邮资值都能得到（当然还有8分、9分和12分）；如果面值分别为1分、3分，则在1分～7分之间的每一个邮资值都能得到。可以验证当N=3，K=2时，7

分就是可以得到的连续的邮资最大值，所以MAX=7，面值分别为1分、3分。

输入格式：

2个整数，代表N，K。

输出格式：

2行。第一行若干个数字，表示选择的面值，从小到大排序。

第二行，输出“MAX=S”，S表示最大的面值。

示例:

```
3 2

1 3
MAX=7
```

思路

- 必须要有1（满足连续性要求）
- 取数，单调递增地取
- 确定上下界，上界是上一个数+1

  以n=3,k=3为例:第一个面值肯定为1，但是第二个面值只能是

  但是第二个面值只能是2，3，4，因为面值为1的最多贴3张

  贴满的最大值为3，要保证数字连续,那么第二个数字最大是4

  所以我们可以得到规律，如果邮票张数为n，种类为k,那么从小到大的顺序，邮票a[i]的下一种面值的取值范围必然是f[i]+1到f[i]*n+1
- 深搜+动态规划

  如果已知邮票的不同面值，可以用动态规划求出这些不同面值的邮票能组合出的最大连续数：

  设dp[i]表示已知面值的邮票组合出面值为i所需要的最小邮票数，我们把已知的q种不同的邮票面值存在num中，则有状态转移方程：dp[i]=min(dp[i-f[j]]+1)

  然后随着搜索不断枚举面值集合，同时更新最大值

Solution

```
 #include<cstdio>
 #include<cstring>
 #include<algorithm>
 using namespace std;
const int maxn=51;
const int INF=2147483647;
int n,k,f[maxn],b[maxn],ans,dp[maxn*100];
void dfs(int x)
{
    if(x==k+1)
    {
        memset(dp,0,sizeof(dp));
        int i=0;
        while(dp[i]<=n)
        {
            i++;
            dp[i]=INF;
            for(int j=1;j<=k&&i-f[j]>=0;j++)
            {
                dp[i]=min(dp[i],dp[i-f[j]]+1);
            }
        }
        if(i-1>ans)
        {
            for(int j=1;j<=k;j++)
            {
                b[j]=f[j];
                ans=i-1;
            }
        }
        return;
    }
    for(int i=f[x-1]+1;i<=f[x-1]*n+1;i++)
    {
        f[x]=i;
        dfs(x+1);
    }
}
int main()
{
    scanf("%d%d",&n,&k);
    f[1]=1;
    dfs(2);
    for(int i=1;i<=k;i++)printf("%d ",b[i]);
    printf("\n");
    printf("MAX=%d\n",ans);
    return 0;
```

```
}
```

P1022 计算器的改良

题目描述：

NCL是一家专门从事计算器改良与升级的实验室，最近该实验室收到了某公司所委托的一个任务：需要在该公司某型号的计算器上加上解一元一次方程的功能。实验室将这个任务交给了一个刚进入的新手ZL先生。

为了很好的完成这个任务，ZL先生首先研究了一些一元一次方程的实例：

$ \begin{array}{l}{4+3 x=8} \\ {6 a-5+1=2-2 a} \\ {-5+12 y=0}\end{array} $

ZL先生被主管告之，在计算器上键入的一个一元一次方程中，只包含整数、小写字母及+、-、=这三个数学符号（当然，符号“-”既可作减号，也可作负号）。方程中并没有括号，也没有除号，方程中的字母表示未知数。

你可假设对键入的方程的正确性的判断是由另一个程序员在做，或者说可认为键入的一元一次方程均为合法的，且有唯一实数解。

输入格式：

一个一元一次方程。

输出格式：

解方程的结果(精确至小数点后三位)。

示例:

```
6a-5+1=2-2a
a=0.750
```

思路

- 1.去分母：在方程两边都乘以各分母的最小公倍数；
- 2.去括号：先去小括号，再去中括号，最后去大括号；
- 3.**移项**：把含有未知数的项都移到方程的一边，其他项都移到方程的另一边；
- 4.**合并同类项**：把方程化成ax=b(a≠0)的形式；
- 5.**系数化成1**：在方程两边都除以未知数的系数a，得到方程的解

Solution

```
#include<cstdio>
#include<cstring>
using namespace std;
double num=0,x=0;//num是数字之和，x为系数之和
char c,p;//c被用作读入每一个字符，p用来储存未知数是啥
int a[100],l=1,mid,pd;//a[]储存每一个出现的数,l记录数组长度,mid分开等号两边,pd为判断此数字是正是负
```

```
int main()
{
    memset(a,0,sizeof(a));//数组清0，因为第一个有可能是又前置符号的数，这样不好解决判断问题，索性全为0，最后加起来也没啥影响
    pd=1;//默认开头第一个数为正
    while(c!='=')//读入等号左边
    {
        c=getchar();
        if(c=='-')
            ++l,pd=-1;//只要判断是-数，切换到下一个数,设置这个数为负数
        if(c=='+')
            ++l,pd=1;//判断为正，切换下一个数，设置这个数为正数
        if(c>='0'&&c<='9')//读入的是数字
        {
            if(!a[l])//这个数字字符是数字中的第一个
                a[l]=(c-'0')*pd;//赋值（记得乘上判断的数）
            else
                a[l]=a[l]*10+(c-'0')*pd;//数字中已经有数了，为了加上这个数，向前移一位再塞进去 （记得乘上判断的数）
        }
        if(c>='a'&&c<='z')//发现未知数!
        {
            p=c;
            if(a[l]!=0)
                x+=a[l],a[l]=0;//如果有前面的系数，则存入系数集合，把塞在数字数组中的系数去掉
            else
                x+=pd;//判断特殊情况如-x/+x
            --l; //减去这一位数
        }
    }
    mid=l,++l,pd=1;//存储mid，数组位数进一位，pd=1与上面同理
    while(c!='\n')//同上
    {
        c=getchar();
        if(c=='-')
            ++l,pd=-1;
        if(c=='+')
            ++l,pd=1;
        if(c>='0'&&c<='9')
        {
            if(!a[l])
                a[l]=(c-'0')*pd;
            else
                a[l]=a[l]*10+(c-'0')*pd;
        }
        if(c>='a'&&c<='z')//这里有点不一样，因为未知数要放在等号左边所以这里要减去系数
        {
```

```
                p=c;
                if(a[l]!=0)
                    x-=a[l],a[l]=0;
                else
                    x-=pd;
                --l;
            }
        }
        for(int i=1;i<=l;++i)//叠加数字
        {
            if(i<=mid)//在等号左边要减去
                num-=a[i];
            else//在等号右边的要加上
                num+=a[i];
        }
        if(!(num/x))//这里要加个特判断，因为会出现-0，虽然-0和0等效，但评测机并不吃这一套
            printf("%c=0.000",p);
        else
            printf("%c=%.3lf",p,num/x);//输出
        return 0;
    }
```

P1023 税收与补贴问题

题目描述：

每样商品的价格越低，其销量就会相应增大。现已知某种商品的成本及其在若干价位上的销量（产品不会低于成本销售），并假设相邻价位间销量的变化是线性的且在价格高于给定的最高价位后，销量以某固定数值递减。（我们假设价格及销售量都是整数）

对于某些特殊商品，不可能完全由市场去调节其价格。这时候就需要政府以税收或补贴的方式来控制。（所谓税收或补贴就是对于每个产品收取或给予生产厂家固定金额的货币）

你是某家咨询公司的项目经理，现在你已经知道政府对某种商品的预期价格，以及在各种价位上的销售情况。要求你确定政府对此商品是应收税还是补贴的最少金额（也为整数），才能使商家在这样一种政府预期的价格上，获取相对其他价位上的最大总利润。

总利润=单位商品利润× 销量

单位商品利润=单位商品价格 - 单位商品成本 （- 税金 or + 补贴）

输入格式：

输入的第一行为政府对某种商品的预期价，第二行有两个整数，第一个整数为商品成本，第二个整数为以成本价销售时的销售量，以下若干行每行都有两个整数，第一个为某价位时的单价，第二个为此时的销量，以一行-1，-1

表示所有已知价位及对应的销量输入完毕，输入的最后一行为一个单独的整数表示在已知的最高单价外每升高一块钱将减少的销量。

输出格式：

输出有两种情况：若在政府预期价上能得到最大总利润，则输出一个单独的整数，数的正负表示是补贴还是收税，数的大小表示补贴或收税的金额最小值。若有多解，取绝对值最小的输出。

如在政府预期价上不能得到最大总利润，则输出“NO SOLUTION”。

示例:

```
31
28 130
30 120
31 110
-1  -1
15


4
```

思路

- 把用户输入的价格和销量通通存进数组里。
- 如果发现有的价格和销量用户没输入则按线性的规则自己写入。
- 在用户输入的最大价格后面按用户输入的递减数量把最后所有销量不为零的价格补充完整。
- 现在，你已经有了一个存有所有销量大于等于零的价格-销量表。
- 从1/-1开始枚举所有可能的补贴或税收，并看看在每种税收或补贴下总利润最高的价格是不是政府预期价，如果是则打印。（因为从最小的开始找的）

Solution

```
#include <iostream>
#include <cmath>
using namespace std;
int a[100010][3];//用于存放价格和销量的数组
int main()
{
    int i=1,j=1,k,expect,down,max,temp,cha,xl,num,s,price,p;
    cin>>expect;//读入预期价
    while(cin>>a[i][1]>>a[i][2]&&a[i][1]!=-1&&a[i] [2]!=-1)//如果输入的两个数不是-1，-1
    {
        i++;//循环变量i++
        if(i>2&&a[i-1][1]-a[i-2][1]>1)//如果两个价格之间差大于一
        {
            i--;//回到上一个读入的销量
            cha=(a[i-1][2]-a[i][2])/(a[i][1]-a[i-1][1]);//求出每次销量减少多少：销量差/价格差
            temp=a[i][1];//记录下价格
            for(j=a[i-1][1]+1;j<=temp;j++)//按价格递增顺序依次写入
            {
                a[i][1]=j;//写入价格
                a[i][2]=a[i-1][2]-cha;//按销量差写入销量
```

```
                i++;
            }
        }
    }
    cin>>down;//输入超过最大价格之后每次销量降低多少
    i--;//因为上面的while循环最后有i++所以用i--抵消……
    xl=a[i][2];//记录目前的销量
    while(xl>0)
    {
        if(xl-down<0)break;//如销量小于零则退出
        else//否则
        {
            xl-=down;//销量每次减掉down
            i++;//循环变量++
            a[i][1]=a[i-1][1]+1;//每次价格+1
            a[i][2]=xl;//销量就是xl
        }
    }
    for(j=1;j<=10000;j++)//该遍历了，因为收税相当于补贴*-1所以记录一下符号即可
    {
        max=-99999;//用于存储最大的总利润
        for(k=1;k<=i;k++)//每次扫一遍每一种价格
        {
            num=(a[k][1]-a[1][1]+j)*a[k][2];//套公式算出总利润
            if(num>=max)//如果总利润比目前最大的大
            {
                max=num;//更新max
                price=a[k][1];//记录下价格
                p=1;//记录下符号
            }
        }
        if(price==expect){cout<<j*p;return 0;}//如果价格就是政府预期价则打印出来，因为本身就是从小到大遍历所以不用求绝对值最小的
        max=-99999;//后面是收税，原理同上
        for(k=1;k<=i;k++)
        {
            num=(a[k][1]-a[1][1]-j)*a[k][2];
            if(num>=max)
            {
               max=num;
                price=a[k][1];
                p=-1;
            }
        }
        if(price==expect){cout<<j*p;return 0;}
    }
    //前面有了return 0;这儿就不用了。
}
```

P1024 一元三次方程求解

题目描述：

有形如：$$a x^{3}+b x^{2}+c x^{1}+d x^{0}=0$$ 这样的一个一元三次方程。给出该方程中各项的系数(a,b,c,d均为实数)，并约定该方程存在三个不同实根(根的范围在-100至100之间)，且根与根之差的绝对值≥1。要求由小到大依次在同一行输出这三个实根(根与根之间留有空格)，并精确到小数点后2位。

提示：记方程f(x)=0，若存在2个数x1和x2，且x1<x2，f(x1)×f(x2)<0，则在(x1,x2)之间一定有一个根。

输入格式：

一行，4个实数A,B,C,D。

输出格式：

一行，3个实根，并精确到小数点后2位。

示例:

```
1 -5 -4 20

-2.00 2.00 5.00
```

思路

- 分治、二分搜索
- 三个答案都在[-100,100]范围内，两个根的差的绝对值>=1,保证了每一个大小为1的区间里至多有1个解，也就是说当区间的两个端点的函数值异号时区间内一定有一个解，同号时一定没有解。那么我们可以枚举互相不重叠的每一个长度为1的区间，在区间内进行二分查找。

Solution

```
#include<cstdio>
double a,b,c,d;
double fc(double x)
{
    return a*x*x*x+b*x*x+c*x+d;
}
int main()
{
    double l,r,m,x1,x2;
    int s=0,i;
    scanf("%lf%lf%lf%lf",&a,&b,&c,&d);  //输入
    for (i=-100;i<100;i++)
    {
        l=i;
        r=i+1;
```

```
            x1=fc(l);
            x2=fc(r);
            if(!x1)
            {
                printf("%.2lf ",l);
                s++;
            }       //判断左端点，是零点直接输出。

                            //不能判断右端点，会重复。
            if(x1*x2<0)                                     //区间内有根。
            {
                while(r-l>=0.001)                           //二分控制精度。
                {
                    m=(l+r)/2;  //middle
                    if(fc(m)*fc(r)<=0)
                       l=m;
                    else
                       r=m;    //计算中点处函数值缩小区间。
                }
                printf("%.2lf ",r);
                //输出右端点。
                s++;
            }
            if (s==3)
                break;
                //找到三个就退出大概会省一点时间
        }
        return 0;
    }
```

P1025 数的划分

题目描述：

将整数n分成k份，且每份不能为空，任意两个方案不相同(不考虑顺序)。

例如：n=7，k=3，下面三种分法被认为是相同的。

$ \begin{array}{l}{1,1,5 ;} \\ {1,5,1} \\ {5,1,1}\end{array} $

问有多少种不同的分法。

输入格式：

$ n, k(6<n \leq 200,2 \leq k \leq 6) $

输出格式：

1个整数，即不同的分法。

示例:

```
7 3

4
```

思路

- 递推的方法来求解
- f[i][j]=f[i-1][j-1]+f[i-j][j]**(i>=j),**f[i][j]=0**(i<j).记得**f[i][i]=1

Solution

```
#include<iostream>
#include "string.h"
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}

#define N 100

int n,k,f[N][N];
int main()
{
    n=read(),k=read();
    for(int i=1;i<=n;i++)f[i][1]=1;//如上1。
    for(int i=1;i<=n;i++)
        for(int j=2;j<=k;j++)//1的情况已经处理，从2开始，否则会访问0。
            if(i>=j)f[i][j]=f[i-1][j-1]+f[i-j][j];//小细节：只有n>=k时才能有数
分。
    printf("%d\n",f[n][k]);//输出从n中分出k份的方案数
    return 0;
}
```

P1026 统计单词个数

题目描述：

给出一个长度不超过200的由小写英文字母组成的字母串(约定;该字串以每行20个字母的方式输入，且保证每行一定为20个)。要求将此字母串分成k份(1<k≤40)，且每份中包含的单词个数加起来总数最大(每份中包含的单词可以部分重叠。当选用一个单词之后，其第一个字母不能再用。例如字符串this中可包含this和is，选用this之后就不能包含th)。

单词在给出的一个不超过6个单词的字典中。

要求输出最大的个数。

输入格式：

每组的第一行有2个正整数(p,k)

p表示字串的行数,k表示分为k个部分。

接下来的p行，每行均有20个字符。

再接下来有1个正整数s，表示字典中单词个数。(1≤s≤6)

接下来的s行，每行均有1个单词。

输出格式：

1个整数，分别对应每组测试数据的相应结果。

示例:

```
1 3
thisisabookyouareaoh
4
is
a
ok
sab

7（this/isabookyoua/reaoh）
```

思路

- 实质问题就是分组的时候，切不同的点会造成不一样的结果，选取最优的切点
- 一开始，不分开和一定是最大的，你只要依次遍历每个点，看能否找到符合的单词，找到一个就行了，因为找到一个单词只影响第一个字母。从i和i+1之间分开意味着原来可能连成单词的被拆开，所以只需要找到每次分开使得失去单词数最小的切开点就行了。
- 用d[i]表示拆分成i个，能得到最大的和 用t[i]表示i到i+1中间是否已被切过 0表示没有，1表示已经被切开过
- 首先很明显d[1] = 整个字符串包含的所有单词和
- 然后从2-k贪心，对每个点遍历找出从这个点切开，导致单词强行分开的点的数目最少的 即 d[i] = d[i-1]+minum(此为最佳切点，失去的单词数目最少)
- 最后d[k]就是答案

Solution

```c
#include<stdio.h>
#define min(a,b) ((a)<(b)?(a):(b))
#define LEN 20
char c[201] = {0}, word[6][200] = {0};
int d[41] = {0}, t[41] = {0};
int main()
{
    int p, k, s, i, j, m, n, f, g, minum;
    scanf("%d%d", &p, &k);
    for(i = 0; i < p; i++)
        scanf("%s", c+i*LEN+1);
    scanf("%d", &s);
    for(i = 0; i < s; i++)
        scanf("%s", word[i]);
    for(i = 1; i <= p*LEN; i++)
        for(j = 0; j < s; j++)
        {
            for(m = 0; word[j][m] != 0; m++)
                if(word[j][m] != c[i+m])
                    break;
            if(word[j][m] == 0)
            {
                d[1]++;
                break;
            }
        }
    for(i = 2; i <= k; i++)
    {
        minum = 99999;
        for(j = 1; j <= p*LEN-1; j++)
            if(!t[j])
            {
                f = 0;
                for(m = 0; m < s; m++)
                    for(n = 0; word[m][n+1] != 0; n++)
                        if(word[m][n] == c[j] && word[m][n+1] == c[j+1])
                        {
                            f++;
                            break;
                        }
                if(minum > f)
                {
                    minum = f;
                    g = j;
                }
            }
        d[i] = d[i-1]-minum;
        t[g] = 1;
    }
```

```
    printf("%d\n", d[k]);
    return 0;
}
```

P1027 Car的旅行路线

题目描述：

又到暑假了，住在城市A的Car想和朋友一起去城市B旅游。她知道每个城市都有4个飞机场，分别位于一个矩形的4个顶点上，同一个城市中2个机场之间有1条笔直的高速铁路，第I个城市中高速铁路了的单位里程价格为Ti，任意两个不同城市的机场之间均有航线，所有航线单位里程的价格均为t。

那么Car应如何安排到城市B的路线才能尽可能的节省花费呢?她发现这并不是一个简单的问题，于是她来向你请教。

找出一条从城市A到B的旅游路线，出发和到达城市中的机场可以任意选取，要求总的花费最少。

输入格式：

第一行为一个正整数n(0≤n≤10)，表示有n组测试数据。

每组的第一行有4个正整数s,t,A,B。S(0<S≤100)表示城市的个数，t表示飞机单位里程的价格，A，B分别为城市A，B的序号，(1≤A，B≤S)。

接下来有S行，其中第I行均有7个正整数xi1,yi1,xi2,yi2,xi3,yi3,Ti，这当中的(xi1,yi1)，(xi2,yi2)，(xi3,yi3)分别是第i个城市中任意3个机场的坐标，Ti为第i个城市高速铁路单位里程的价格。

输出格式：

共有n行，每行1个数据对应测试数据。 保留一位小数。

示例:

```
1
3 10 1 3
1 1 1 3 3 1 30
2 5 7 4 5 2 1
8 6 8 8 11 6 3

47.5
```

思路

- 本题我们不妨把每个城市的4个机场看做四个点。那样这图就有4×s 个点。
- 根据题目描述，我们又知道：每一个机场都与另外每一个机场互通，差别只是在是否是同一个城市：
- 如果是，那么只能走高速铁路；
- 如果不是，那么只能走航道。用一个判断来计算这条路的花费即可。
- 最后跑最短路，答案为到达城市的4个机场的花费的最小值。

Solution

```
#include <cstdio>
#include <cmath>
#include <cstring>
#include <queue>
using namespace std;

struct data {
    int x,y; //(x,y)
    int city; // 所在城市
};

const int maxn=100;
int s,t,A,B;
int T[maxn+1];
double dis[maxn<<2|1];
data a[maxn<<2|1];

int pingfang(int x) { return x*x; }

//两点间距离公式
double juli(int x1, int y1, int x2, int y2) { return sqrt(pingfang(x1-
y1)+pingfang(x2-y2)); }

//求矩形的第四个点的函数
void get_4th(int x1, int y1, int x2, int y2, int x3, int y3, int i) {
    //已知A(x1,y1),B(x2,y2),C(x3,y3)，求D(x4,y4)
    //ab表示AB^2,ac表示AC^2,BC表示BC^2
    int ab=pingfang(x1-x2)+pingfang(y1-y2),
        ac=pingfang(x1-x3)+pingfang(y1-y3),
        bc=pingfang(x2-x3)+pingfang(y2-y3);
    int x4,y4;
    //用勾股定理的逆定理，判断谁是直角边
    //再根据矩形对边平行的性质，算出第四个点的坐标
    if (ab+ac==bc) x4=x2+x3-x1, y4=y2+y3-y1;
    if (ab+bc==ac) x4=x1+x3-x2, y4=y1+y3-y2;
    if (ac+bc==ab) x4=x1+x2-x3, y4=y1+y2-y3;
    a[i+3].x=x4;
    a[i+3].y=y4;
}

//初始化函数如题意所述
void init() {
    memset(a,0,sizeof(a));
    scanf("%d%d%d%d",&s,&t,&A,&B);
    //对每个城市的机场进行处理
    for (int i=1; i<=4*s; i+=4) {
```

```
 scanf("%d%d%d%d%d%d%d",&a[i].x,&a[i].y,&a[i+1].x,&a[i+1].y,&a[i+2].x,&a[i+
2].y,&T[i/4+1]);
        a[i].city=a[i+1].city=a[i+2].city=a[i+3].city=i/4+1;
        //调用求出第四个点坐标的函数
        get_4th(a[i].x,a[i].y,a[i+1].x,a[i+1].y,a[i+2].x,a[i+2].y,i);
    }
}

//最短路spfa
void spfa() {
    //队内有没有该元素（用于加速）
    bool mark[maxn<<2|1];
    queue <int> q;
    for (int i=1; i<=4*s; i++) dis[i]=99999999.99999;
    //可以从出发地任意一个机场出发，所以初始化都入队，并且花费均为0
    for (int i=A*4-3;i<=A*4;i++)
        dis[i]=0, q.push(i), mark[i]=true;

    //bfs
    while (!q.empty()) {
        int x=q.front(); q.pop(); mark[x]=false;
        //这个机场与其余所有机场都有通路
        for (int i=1; i<=4*s; i++) {
            if (i==x) continue;
            //花费先赋值为两点间的距离
            double cost=juli(a[x].x,a[i].x,a[x].y,a[i].y);
            //如果两机场在同一城市，则走该城市的高速铁路
            if (a[i].city==a[x].city) cost*=T[a[i].city];
            //否则坐飞机
            else cost*=t;
            //如果花费更少则更新
            if (dis[x]+cost<dis[i]) {
                dis[i]=dis[x]+cost;
                if (!mark[i])
                    mark[i]=true, q.push(i);
            }
        }
    }
}

int main() {
    int n;
    scanf("%d",&n);

    //有多组数据
    while (n--) {
        init();
        spfa();
```

```
        //答案是到达地四个机场中花费最少的那个
        //用“打擂台”的方法求出最小值
        double ans=dis[B*4];
        for (int i=B*4-3; i<B*4; i++)
            if (dis[i]<ans) ans=dis[i];
        printf("%.1lf",ans);
    }
}
```

P1028 数的计算

题目描述：

我们要求找出具有下列性质数的个数(包含输入的自然数n):

先输入一个自然数nn(n≤1000),然后对此自然数按照如下方法进行处理:

1. 不作任何处理;
2. 在它的左边加上一个自然数,但该自然数不能超过原数的一半;
3. 加上数后,继续按此规则进行处理,直到不能再加自然数为止.

输入格式：

1个自然数n(n≤1000)

输出格式：

1个整数，表示具有该性质数的个数。

示例:

```
6 6(6, 16, 26, 126, 36, 136)
```

思路

- 直接递归 TLE
- 故而采用递推的方法：
- f[1]=1 f[2]=2=f[1]+1 f[3]=2=f[1]+1 f[4]=4=f[1]+f[2]+1 f[5]=4=f[1]+f[2]+1

Solution

```
#include<cstdio>
using namespace std;
int main(){
    int n,cnt=1,i,f[1010];
    f[0]=f[1]=1;
    scanf("%d",&n);
    for(i=2;i<=n;i++){
        if(i%2==0){
            f[i]=f[i-1]+f[i/2];
```

```
            }else{
                f[i]=f[i-1];
            }
        }
        printf("%d\n",f[n]);
    }
```

P1029 最大公约数和最小公倍数问题

题目描述：

1. 输入2个正整数x0,y0(2≤x0<100000,2≤y0<=1000000),求出满足下列条件的P,Q的个数

   条件:

   1. P,Q是正整数
   2. 要求P,Q以x0为最大公约数,以y0为最小公倍数.

   试求:满足条件的所有可能的2个正整数的个数.

输入格式：

2个正整数x0,y0

输出格式：

1个数，表示求出满足条件的P,Q

的个数示例:

```
3 60

4 (3,60; 15,12; 12,15; 60,3)
```

思路

- gcd(a,b)为数a与数b的最大公约数
- lcm(a,b)为数a与数b的最小公倍数
- 设s=lmc(p,q)/gcd(p,q),问题化为求解 xy=s 且 gcd(x,y)=1 的正整数解数。因式分解就可以得到。

Solution

```
#include<iostream>
#include "string.h"
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
```

```
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}

int main()
{
   int x0 = read();
   int y0 = read();

   int mul = x0 * y0;

   int result = 0;

   for (int i = x0; i <= y0 ; i++) {
       for(int j = x0; j <= y0 ; j++) {
            if( (i % x0 == 0) && (j % x0 == 0) && (y0 % i == 0) && (y0 % j
== 0)) {
                if(i * j == mul && __gcd(i, j) == x0) {
                    result += 1;
                }
            }
       }
   }

   cout << result << endl;


    return 0;
}
```

P1030 求先序排列

题目描述：

给出一棵二叉树的中序与后序排列。求出它的先序排列。（约定树结点用不同的大写字母表示，长度≤8）。

输入格式：

2行，均为大写字母组成的字符串，表示一棵二叉树的中序与后序排列。

输出格式：

1行，表示一棵二叉树的先序。

的个数示例:

```
BADC
BDCA

ABCD
```

思路

- 一点基本常识，给你一个后序遍历，那么最后一个就是根
- step1:找到根并输出
- step2:将中序，后序各分为左右两棵子树；
- step3:递归，重复step1,2；

Solution

```
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
void beford(string in,string after){
    if (in.size()>0){
        char ch=after[after.size()-1];
        cout<<ch;//找根输出
        int k=in.find(ch);
        beford(in.substr(0,k),after.substr(0,k));
        beford(in.substr(k+1),after.substr(k,in.size()-k-1));//递归左右子树；
    }
}
int main(){
    string inord,aftord;
    cin>>inord;cin>>aftord;//读入
    beford(inord,aftord);cout<<endl;
    return 0;
}
```

P1031 均分纸牌

题目描述：

有N堆纸牌，编号分别为1,2,…,N。每堆上有若干张，但纸牌总数必为N的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为1堆上取的纸牌，只能移到编号为2的堆上；在编号为N的堆上取的纸牌，只能移到编号为N−1的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。

例如N=4，4堆纸牌数分别为：

①9②8③17④6

移动33次可达到目的：

从 ③ 取4张牌放到 ④ （9,8,13,10）-> 从 ③ 取3张牌放到 ②（9,11,10,10）-> 从 ② 取1张牌放到 ①（10,10,10,10）。

输入格式：

两行

第一行为：N（N 堆纸牌，1≤N≤100）

第二行为：A1,A2,…,An （N堆纸牌，每堆纸牌初始数，1≤Ai≤10000）

输出格式：

一行：即所有堆均达到相等时的最少移动次数。

的个数示例:

```
4
9 8 17 6

3
```

思路

- 算平均数。
- 求每堆纸牌与平均数的关系（多1记为1，少1记为-1）
- 当q[y]（第y堆纸牌与平均数的关系）不等于0时，q[y+1]=q[y+1]+q[y],移动次数加1

Solution

```
#include <iostream>
using namespace std;
int main()
{
int a,p=0,js=0; cin >>a;int q[a];
for (int y=0;y<a;y++){cin >>q[y]; p+=q[y];} p/=a;
for (int y=0;y<a;y++)q[y]-=p;
for (int y=0;y<a;y++) {if (q[y]==0)continue; q[y+1]+=q[y]; js++; }
cout <<js;
return 0;
}
```

P1032 字串变换

题目描述：

已知有两个字串A,B及一组字串变换的规则（至多6个规则）：

A1 ->B1

A2 -> B2

规则的含义为：在 A中的子串 A1 可以变换为B1，A2 可以变换为 B2 ...。

例如：A=`abcd`，B＝`xyz`，

变换规则为：

```
abc→xu，ud→y，y→yz
```

则此时，A可以经过一系列的变换变为B，其变换的过程为：

`abcd`→`xud`→`xy`→`xyz`。

共进行了33次变换，使得A变换为B。

输入格式：

输入格式如下：

A B A1 B1 A2 B2 |-> 变换规则

... ... /

所有字符串长度的上限为20。

输出格式：

输出至屏幕。格式如下：

若在10步（包含10步）以内能将A变换为B，则输出最少的变换步数；否则输出"NO ANSWER!"

实例：

```
abcd xyz
abc xu
ud y
y yz

3
```

思路

- KMP字符串模式匹配
- 广度优先搜索

Solution

```
#include <iostream>
#include <cctype>
#include <cmath>
#include <ctime>
#include <climits>
```

```
#include <cstring>
#include <string>
#include <cstdio>
#include <cstdlib>
#include <iomanip>
#include <algorithm>
#include <sstream>
#include <queue>
#include <map>
#define debug cout << "debug"<<endl

using namespace std;
#define il inline
#define re register
typedef long long ll;

string a,b;

struct Node {//用于queue中存放，一个是字串，一个是搜索的“深度”
    string data;
    int step;
    Node(string _data,int _step):data(_data),step(_step) {}
    Node() {}
};
queue<Node>q;
string change[10];//改成哪个
string diff[10];//改哪个
/*即
搜索diff[i]
改成change[i]
*/

int nxt[10][10000];//kmp的next数组
map<string,bool>mp;//用于判重，避免重复搜索
il void get_next(int x)//找next，具体的可以翻翻网上的Blog。
{
    re int i,j=0;
    for (i=2; i<diff[x].length(); i++) {
        while (j&&diff[x][i]!=diff[x][j+1]) j=nxt[x][j];
        if (diff[x][j+1]==diff[x][i]) j++;
        nxt[x][i]=j;
    }
}

il void KMP(string a,int x,int step)//寻找匹配的串，顺便修改并添加到queue中
{
    string z=a;
    a=" "+a;//神奇的操作，。。。
    re int i,j=0;
```

```
    for (i=1; i<a.length(); i++) {
        while (j>0&&diff[x][j+1]!=a[i]) j=nxt[x][j];
        if (diff[x][j+1]==a[i]) j++;
        if (j==diff[x].length()-1) {//找到了~
            re int t= i-diff[x].length()+1;//记录位置
            string
tmp=z.substr(0,t)+change[x]+z.substr(t+diff[x].length()-1);//修改（就不用
replace，（真香））
            q.push(Node(tmp,step+1));
            j=nxt[x][j];//继续找
/*
第一次交由于脑子不好，找了一遍就return了。
*/
        }
    }
    return;
}

int cn=0;
int main()
{
    //freopen("in.txt","r",stdin);
    cin >> a >> b;
    string t1,t2;
    while (cin >>t1>>t2) {
        change[++cn]=t2;
        diff[cn]=" "+t1;//继续神奇的操作
        get_next(cn);
    }
    q.push(Node(a,0));
    while (!q.empty()) {
        Node now=q.front();
        q.pop();
        string x=now.data;
        if (mp[x]) continue;//map判重
        mp[x]=1;//标记
        if (now.step>10) {//找不到（因为bfs是按照step:1,2,3...来找的，所以一旦到
了STEP11时一定无解了）
            puts("NO ANSWER!");
            exit(0);
        }
        if (x==b) {//找到，由于搜索有序，step一定是最小的
            cout << now.step<<endl;
            exit(0);
        }
        for (re int i=1; i<=cn; i++) {//枚举所有模式串，匹配文本串
            KMP(x,i,now.step);
        }
    }
```

```
    puts("NO ANSWER!");//最后由于map的判重，可能导致queue为空，于是到达这里的数据肯定是无解的
    exit(0);
}
```

P1033 自由落体

题目描述：

在高为H的天花板上有n个小球，体积不计，位置分别为0,1,2,...,n−1。在地面上有一个小车（长为L，高为K，距原点距离为S1）。已知小球下落距离计算公式为d=0.5×g×(t2)，其中g=10，t为下落时间。地面上的小车以速度V前进。

小车与所有小球同时开始运动，当小球距小车的距离≤0.0001(感谢Silver_N修正) 时，即认为小球被小车接受（小球落到地面后不能被接受）。

请你计算出小车能接受到多少个小球。

输入格式：

H,S1,V,L,K,n(l≤H,S1,V,L,K,n≤100000)

输出格式：

小车能接受到的小球个数。

实例：

```
5.0 9.0 5.0 2.5 1.8 5


1
```

思路

- 这道题需要物理基础
- 算出球落到车顶和车底的区间
- 通过区间算答案

Solution

```
#include<map>
#include<cmath>
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<iostream>
#include<algorithm>
using namespace std;
double h,s1,v,l,k;
int n;
```

```
int main()
{
    scanf("%lf%lf%lf%lf%lf%d",&h,&s1,&v,&l,&k,&n);
    double time_oncar=sqrt((h-k)/5);
    double carend=(s1+l)-v*time_oncar+0.0001;
    double time_under=sqrt(h/5);
    double start=s1-v*time_under-0.0001;
    int s,e,ans;
    s=start;
    if(double(s)!=start)s++;
    e=carend;
    ans=min(n-1,e)-max(0,s)+1;
    printf("%d",max(ans,0));
}
```

P1034 矩形覆盖

题目描述：

在平面上有n个点(n≤50)，每个点用一对整数坐标表示。例如：当n=4 时，4个点的坐标分另为：p1(1,1)，p2(2,2)，p3(3,6)，P4(0,7)，见图一。

这些点可以用k个矩形(1≤k≤4)全部覆盖，矩形的边平行于坐标轴。当 k=2 时，可用如图二的两个矩形 s1,s2 覆盖，s1,s2 面积和为4。问题是当n个点坐标和k给出后，怎样才能使得覆盖所有点的k个矩形的面积之和为最小呢?

约定：覆盖一个点的矩形面积为0；覆盖平行于坐标轴直线上点的矩形面积也为0。各个矩形必须完全分开（边线与顶点也都不能重合）。

输入格式：

$ \begin{array}{l}{n k} \\ {x_{1} y_{1}} \\ {x_{2} y_{2}} \\ {\ldots \ldots} \\ {x_{n} y_{n}\left(0 \leq x_{i}, y_{i} \leq 500\right)}\end{array} $

输出格式：

输出至屏幕。格式为：

1个整数，即满足条件的最小的矩形面积之和。

实例：

```
4 2
1 1
2 2
3 6
0 7

4
```

思路

- 暴力搜索
- 暴力枚举每个点被哪个矩形覆盖

Solution

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

int getint(){
    int x=0,f=1; char ch=getchar();
    while(ch>'9'||ch<'0'){if(ch=='-')f=-f; ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0'; ch=getchar();}
    return f*x;
}
const int MAXN=100;
const int inf=0x3f3f3f3f;
struct POINT{int X,Y;}point[MAXN];
struct REG{int x1,y1,x2,y2;}reg[10];
int n,k,ans=inf;
int insert(int p,int r){//矩形r从原来状态变到覆盖点p所增大的面积
    if(reg[r].x1<0){
        reg[r].x1=reg[r].x2=point[p].X;
        reg[r].y1=reg[r].y2=point[p].Y;
        return 0;
    }
    int ls=(reg[r].x2-reg[r].x1)*(reg[r].y2-reg[r].y1);//原面积
    /*以下是分类讨论*/
    if(point[p].X<reg[r].x1){
        reg[r].x1=point[p].X;
        if(point[p].Y<reg[r].y1) reg[r].y1=point[p].Y;
        if(reg[r].y2<point[p].Y) reg[r].y2=point[p].Y;
        return (reg[r].x2-reg[r].x1)*(reg[r].y2-reg[r].y1)-ls;
    }
    if(reg[r].x1<=point[p].X&&point[p].X<=reg[r].x2){
        if(point[p].Y>reg[r].y2){
            reg[r].y2=point[p].Y;
            return (reg[r].x2-reg[r].x1)*(reg[r].y2-reg[r].y1)-ls;
        }
        if(point[p].Y<reg[r].y1){
            reg[r].y1=point[p].Y;
            return (reg[r].x2-reg[r].x1)*(reg[r].y2-reg[r].y1)-ls;
        }
        return 0;
    }
    if(reg[r].x2<point[p].X){
        reg[r].x2=point[p].X;
        if(point[p].Y<reg[r].y1) reg[r].y1=point[p].Y;
        if(reg[r].y2<point[p].Y) reg[r].y2=point[p].Y;
        return (reg[r].x2-reg[r].x1)*(reg[r].y2-reg[r].y1)-ls;
```

```
    }
    /*以上是分类讨论*/
}
inline bool check(){//是否有矩形重叠
    for(int i=1;i<=k;++i)
        for(int j=1;j<=k;++j){
            if(i==j||reg[i].x1<0||reg[j].x1<0) continue;
            if(reg[i].x1<=reg[j].x1&&reg[j].x1<=reg[i].x2&&
               reg[i].y1<=reg[j].y1&&reg[j].y1<=reg[i].y2) return true;
            if(reg[i].x1<=reg[j].x1&&reg[j].x1<=reg[i].x2&&
               reg[i].y1<=reg[j].y2&&reg[j].y2<=reg[i].y2) return true;
            if(reg[i].x1<=reg[j].x2&&reg[j].x2<=reg[i].x2&&
               reg[i].y1<=reg[j].y1&&reg[j].y1<=reg[i].y2) return true;
            if(reg[i].x1<=reg[j].x2&&reg[j].x2<=reg[i].x2&&
               reg[i].y1<=reg[j].y2&&reg[j].y2<=reg[i].y2) return true;
        }
    return false;
}
void dfs(int used,int sum){
    if(check()||sum>=ans) return;//check判断矩形是否有重叠，最优化剪枝
    if(used>n){
        ans=min(ans,sum); return;
    }
    for(int i=1;i<=k;++i){//暴力枚举当前点(点 used )被哪个矩形(矩形 i )覆盖
        int lx1=reg[i].x1,lx2=reg[i].x2,ly1=reg[i].y1,ly2=reg[i].y2;
        dfs(used+1,sum+insert(used,i));
        reg[i].x1=lx1,reg[i].x2=lx2,reg[i].y1=ly1,reg[i].y2=ly2;
    }
}
int main(){
    n=getint(),k=getint();
    for(int i=1;i<=n;++i) point[i].X=getint(),point[i].Y=getint();//read in
    for(int i=1;i<=k;++i)
reg[i].x1=reg[i].x2=reg[i].y1=reg[i].y2=-1;//initialize
    dfs(1,0);
    cout<<ans<<endl;
}
```

P1035 级数求和

题目描述：

已知：$$S_{n}=1+1 / 2+1 / 3+\ldots+1 / n_{\circ}$$显然对于任意一个整数K，当n足够大的时候，Sn大于K。

现给出一个整数K（1≤k≤15），要求计算出一个最小的n；使得Sn>K。

输入格式：

一个正整数K

输出格式：

一个正整数N

实例：

```
1


2
```

思路

- 较为简单

Solution

```
#include<iostream>
#include "string.h"
#include<cstdio>
#include<cmath>
#include<algorithm>
#define ull unsigned long long
using namespace std;
inline int read(){//快速读入
    char c=getchar();int num=0;bool b=0;
    for(;c<'0'||c>'9';b=(c=='-'?1:0),c=getchar());
    for(;c>='0'&&c<='9';num=(num<<3)+(num<<1)+(c^'0'),c=getchar());
    return b?-num:num;
}


int main()
{
    int k = read();

    double sum = 0;

    double n = 1;
    while(sum <= k) {
        double x = 1/n;
        sum += x;

        n += 1;
    }

    cout<<n - 1;


    return 0;
```

```
}
```

P1036 选数

题目描述：

已知n 个整数x1,x2,...,xn，以及1个整数k(k<n)。从n个整数中任选k个整数相加，可分别得到一系列的和。例如n=4,k=3,4个整数分别为3,7,12,19时，可得全部的组合与它们的和为：

$ \begin{array}{l}{3+7+12=22} \\ {3+7+19=29} \\ {7+12+19=38} \\ {3+12+19=34}\end{array} $

现在，要求你计算出和为素数共有多少种。

例如上例，只有一种的和为素数：3+7+19=29。

输入格式：

$ \begin{array}{l}{n, k(1 \leq n \leq 20, k<n)} \\ {x_{1}, x_{2}, \ldots, x_{n}\left(1 \leq x_{i} \leq 5000000\right)}\end{array} $

输出格式：

1个整数（满足条件的种数）

实例：

```
4 3
3 7 12 19

1
```

思路

- 这道题的关键在于如何列出所有的选数组合，那么自然想到递归。我们将n个数存入数组num[]中，选数过程可以看作是从下标0～n-1中选择不重复的k个填满k个空位。函数dfs()中sum记录当前已选数的累加和，left记录当前剩余空位数，p指出当前空位可以从num[]中哪一位下标开始选。由于组合是无序的，且不能选择重复的数，所以要遵循从前往后选数的规则，即后一个空位只能选择num[]中下标比前一个空位所选数下标大的数。另一方面，还必须保证后面所有的空位都至少有一个选择，所以当前空位所选数的下标最大只能是n-left。因此p～n-left都是当前可能的选择，遍历所有可能递归调用dfs()进行下一个空位的选择即可。直到left为0，说明已无空位，判定当前的sum是否为素数然后退出。

Solution

```c
#include <stdio.h>

int n, k, cnt;
int num[25];

int isPrime(int n)
```

```
{
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
            return 0;
    return 1;
}

void dfs(int sum, int p, int left)
{
    if (!left)
    {
        if (isPrime(sum))
            cnt++;
        return;
    }
    for (int i = p; i <= n - left; i++)
        dfs(sum + num[i], i + 1, left - 1);
}

int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i++)
        scanf("%d", &num[i]);
    dfs(0, 0, k);
    printf("%d\n", cnt);
    return 0;
}
```

附

递归取数（排列）

从n个数中选取k个数

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
vector<int> a;
vector<int> b;
void fun(int a)
{
        cout << a << " ";
}
void show(vector<int> a)
{
        for_each(a.begin(),a.end(),fun);
        cout << endl;
```

```
}

//这个函数的功能是求从数组a的n个数中选出k个数放到数组b之中
void nchoosek(int n,int k)
{
        int i;
        if(k==0)
        {
                show(b);
        }
        else
        {
                //这个地方i是下标
                for(i=k-1;i<=n-1;i++)
                {
                        //确定倒数第一个数
                        b[k-1]=a[i];
                        //大问题转化为小问题
                        //这个地方的i是小问题n的规模
                        //二者的意义是不一样的，切记
                        nchoosek(i,k-1);
                }
        }
}

void nchoosek_wrap(int n,int k)
{
        sort(a.begin(),a.end());
        nchoosek(n,k);
}

int main()
{
        int n,k;
        cin >> n >> k;
    a.resize(n);
    b.resize(k);
        for(int i=0;i<n;i++)
        {
                cin >> a[i];
        }
        nchoosek_wrap(n,k);
        return 0;
}
```

P1037 产生数

题目描述：

给出一个整数n 和k 个规则。

求出：

经过任意次的变换（0次或多次），能产生出多少个不同整数。

仅要求输出个数。

输入格式：

$ \begin{array}{l}{n, k(1 \leq n \leq 20, k<n)} \\ {x_{1}, x_{2}, \ldots, x_{n}\left(1 \leq x_{i} \leq 5000000\right)}\end{array} $

输出格式：

1个整数（满足条件的个数）

实例：

```
234 2
2 5
3 6

4
```

思路

- ans=每个位上可以取的数字的种数

  所以我们要做的就是统计每种数字可以变换成哪些数字

  同时注意统计自己（因为可以不变）

  然而两次都忽略了当x --- > y后，就可以进行y的变换了，

  所以由于这个特性，单纯统计就不好计算了，

  所以我们采用深搜，并且判断走过的不能走，以免陷入死循环，

  同时由于ans会很大，所以要用高精

Solution

```
#include <iostream>
#include <string>
using namespace std;
string str;
int k,vis[10][10],f[10],num[101];
inline void floyd() {  //弗洛伊德
  for (int k = 0;k <= 9;k++)
    for (int i = 0;i <= 9;i++)
      for (int j = 0;j <= 9;j++) vis[i][j] = vis[i][j] || (vis[i][k] &&
vis[k][j]);
}
int main (){
```

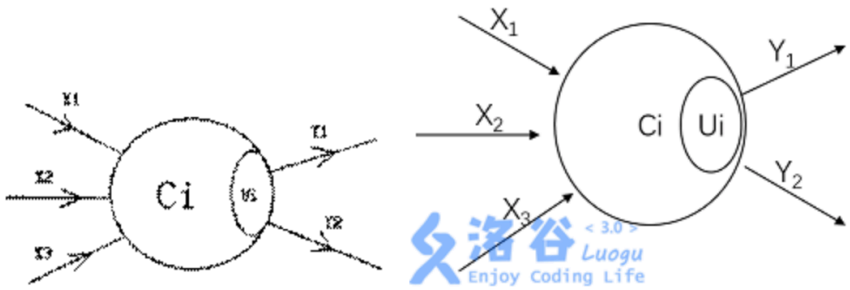
```
    ios::sync_with_stdio(false);
    cin >> str >> k;
    while (k--) {
      int a,b;
      cin >> a >> b;
      vis[a][b] = true;  //a可以变成b
    }
    for (int i = 0;i <= 9;i++) vis[i][i] = true;  //自己可以变成自己
    floyd();
    for (int i = 0;i <= 9;i++)
      for (int j = 0;j <= 9;j++)
        if (vis[i][j]) f[i]++;  //求出i可以变成多少种数字
    int len = 2; num[1] = 1;
    for (int i = 0;i < (int)str.length();i++) {  //高精度
      for (int j = 1;j <= 100;j++) num[j] *= f[str[i]-'0'];
      for (int j = 1;j <= 100;j++)
        if (num[j] >= 10) {  //进位
          num[j+1] += num[j]/10;
          num[j] %= 10;
        }
      while (num[len]) len++;  //求出长度
    }
    for (int i = len-1;i >= 1;i--) cout << num[i];  //输出
    return 0;
}
```

P1038 神经网络

题目描述：

人工神经网络（ArtificialNeuralNetworkArtificialNeuralNetwork）是一种新兴的具有自我学习能力的计算系统，在模式识别、函数逼近及贷款风险评估等诸多领域有广泛的应用。对神经网络的研究一直是当今的热门方向，兰兰同学在自学了一本神经网络的入门书籍后，提出了一个简化模型，他希望你能帮助他用程序检验这个神经网络模型的实用性。

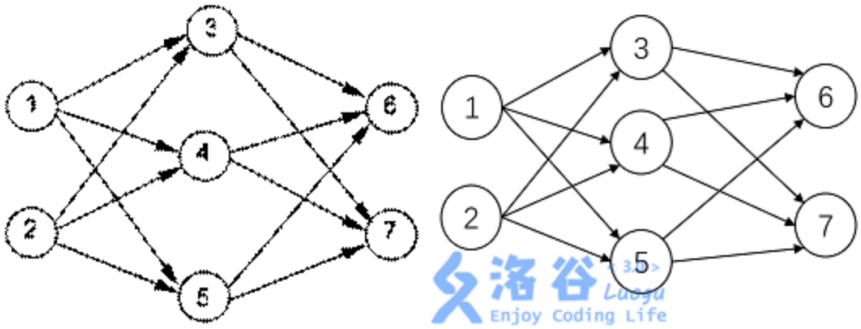
在兰兰的模型中，神经网络就是一张有向图，图中的节点称为神经元，而且两个神经元之间至多有一条边相连，下图是一个神经元的例子：



神经元〔编号为1）

图中，$X_1 - X_3$是信息输入渠道，$Y_1 - Y_2$是信息输出渠道，C_1表示神经元目前的状态，U_i是阈值，可视为神经元的一个内在参数。

神经元按一定的顺序排列，构成整个神经网络。在兰兰的模型之中，神经网络中的神经元分为几层；称为输入层、输出层，和若干个中间层。每层神经元只向下一层的神经元输出信息，只从上一层神经元接受信息。下图是一个简单的三层神经网络的例子。



兰兰规定，C_i服从公式：（其中n是网络中所有神经元的数目）

$$C_i = \sum_{(j,i)\in E} W_{ji}C_j - U_i$$

$$C_i = \sum_{(j,i)\in E} W_{ji}C_j - U_i$$

公式中的W_{ji}（可能为负值）表示连接j号神经元和i号神经元的边的权值。当 C_i大于0时，该神经元处于兴奋状态，否则就处于平静状态。当神经元处于兴奋状态时，下一秒它会向其他神经元传送信号，信号的强度为C_i。

如此．在输入层神经元被激发之后，整个网络系统就在信息传输的推动下进行运作。现在，给定一个神经网络，及当前输入层神经元的状态（C_i），要求你的程序运算出最后网络输出层的状态。

输入格式

输入文件第一行是两个整数$n(1 \leq n \leq 100)$和p。接下来n行，每行2个整数，第$i+1$行是神经元i最初状态和其阈值（U_i），非输入层的神经元开始时状态必然为0。再下面P行，每行由2个整数i,j及1个整数W_{ij}，表示连接神经元i,j的边权值为W_{ij}。

输出格式

输出文件包含若干行，每行有2个整数，分别对应一个神经元的编号，及其最后的状态，2个整数间以空格分隔。仅输出最后状态大于0的输出层神经元状态，并且按照编号由小到大顺序输出。

若输出层的神经元最后状态均为 0，则输出“NULL”。

输入输出样例

输入 #1 复制

```
5 6
1 0
1 0
0 1
0 1
0 1
1 3 1
1 4 1
1 5 1
2 3 1
2 4 1
2 5 1
```

输出 #1 复制

```
3 1
4 1
5 1
```

思路

-

Solution

```
#include<iostream>
#include<cstdio>

#define _ 0

using namespace std;

struct edge
```

```
{
    int v,w,ne;
}a[5000];

int n,p,tmp;
int c[110],u[110],h[110],in[110],out[110];
bool v[110];

int main()
{
    scanf("%d%d",&n,&p);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d",&c[i],&u[i]);
        if(c[i]>0)u[i]=-1;
    }
    for(int s,d,w,i=1;i<=p;i++)
    {
        scanf("%d%d%d",&s,&d,&w);
        a[++tmp]=(edge){d,w,h[s]};
        h[s]=tmp;
        in[d]++;
        out[s]++;
    }
    for(int tmp=1;tmp<=n;tmp++)
    {
        for(int i=1;i<=n;i++)
        {
            if(in[i]==0&&v[i]==0)
            {
                v[i]=1;
                if(u[i]!=-1)c[i]-=u[i];
                for(int j=h[i];j>0;j=a[j].ne)
                {
                    if(c[i]>0)
                    {
                        c[a[j].v]+=c[i]*a[j].w;
                    }
                    in[a[j].v]--;
                }
                break;
            }
        }
    }
    bool flag=0;
    for(int tmp=1;tmp<=n;tmp++)
    {
        if(out[tmp]==0&&c[tmp]>0)flag=1;
    }
```

```
    if(flag)
    {
        for(int tmp=1;tmp<=n;tmp++)
        {
            if(out[tmp]==0&&c[tmp]>0)
            {
                printf("%d %d\n",tmp,c[tmp]);

            }
        }
    }
    else
    {
        printf("NULL");
    }
    return ~~(0^_^0);
}
```

P1039 侦探推理

题目描述：

明明同学最近迷上了侦探漫画《柯南》并沉醉于推理游戏之中，于是他召集了一群同学玩推理游戏。游戏的内容是这样的，明明的同学们先商量好由其中的一个人充当罪犯（在明明不知情的情况下），明明的任务就是找出这个罪犯。接着，明明逐个询问每一个同学，被询问者可能会说：

证词中出现的其他话，都不列入逻辑推理的内容。

明明所知道的是，他的同学中有NN个人始终说假话，其余的人始终说真。

现在，明明需要你帮助他从他同学的话中推断出谁是真正的凶手，请记住，凶手只有一个！

输入格式：

输入由若干行组成，第一行有三个整数，M(1≤M≤20)、N(1≤N≤M)和P(1≤P≤100)；M是参加游戏的明明的同学数，N是其中始终说谎的人数，P是证言的总数。

接下来M行，每行是明明的一个同学的名字（英文字母组成，没有空格，全部大写）。

往后有P行，每行开始是某个同学的名字，紧跟着一个冒号和一个空格，后面是一句证词，符合前表中所列格式。证词每行不会超过250个字符。

输入中不会出现连续的两个空格，而且每行开头和结尾也没有空格。

输出格式：

如果你的程序能确定谁是罪犯，则输出他的名字；如果程序判断出不止一个人可能是罪犯，则输出"Cannot Determine"；如果程序判断出没有人可能成为罪犯，则输出 "Impossible"。

输入输出样例：

```
3 1 5
MIKE
CHARLES
KATE
MIKE: I am guilty.
MIKE: Today is Sunday.
CHARLES: MIKE is guilty.
KATE: I am guilty.
KATE: How are you??、

MIKE
```

思路

- 根据题意可知可以利用枚举法完成。枚举罪犯和今天是星期几。满足N个人始终说谎话和M-N个人始终说真话的条件，就可以确定罪犯。
- 首先利用字符串操作将证言转化成计算机可表示的信息。
- 说话者的内容：

```
"Today is Monday.",    //0
"Today is Tuesday.",   //1
"Today is Wednesday.", //2
"Today is Thursday.",  //3
"Today is Friday.",    //4
"Today is Saturday.",  //5
"Today is Sunday.",    //6
"I am guilty.",        //7
"I am not guilty.",    //8
"is guilty.",          //9
"is not guilty.",      //10
```

Solution

```
#include <bits/stdc++.h>
using namespace std ;
const int M = 20 ;
const int P = 100 ;
const int L = 300 ;
char Says[30][300]=
{ //一共11中说话类型
    "Today is Monday.",    //0
    "Today is Tuesday.",   //1
    "Today is Wednesday.", //2
    "Today is Thursday.",  //3
    "Today is Friday.",    //4
```

```
    "Today is Saturday.",  //5
    "Today is Sunday.",    //6
    "I am guilty.",        //7
    "I am not guilty.",    //8
    "is guilty.",          //9
    "is not guilty.",      //10
} ;
char talk[P],name[M][L],who[P][L],guilty[P][L],sayswhat[P][L] ;
bool liar[M] ;
vector<pair<int,int> > vec[M] ;
int m,n,p ;
bool same(char a[],char b[]) { //判断两个字符串是否相同
    if (strlen(a)!=strlen(b)) return false ;
    for (int i=0;i<strlen(a);i++) if (a[i]!=b[i]) return false ;
    return true;
}
int search(char s[],char b){
    for (int i=0;i<strlen(s);i++) if (s[i]==b) return i ;
}
bool lie(int i,int j,int d,int x){
    if (vec[i][j].first>=0 && vec[i][j].first<=6){
        if (vec[i][j].first!=d) return true ;
    }
    else if (vec[i][j].first==7){
        if (i!=x) return true  ;
    }
    else if (vec[i][j].first==8){
        if (i==x) return true ;
    }
    else if (vec[i][j].first==9){
        if (vec[i][j].second!=x) return true  ;
    }
    else if (vec[i][j].first==10){
        if (vec[i][j].second==x) return true ;
    }
    return false ;
}
bool judge(int d,int x){
    int nosay=0 ;
    memset(liar,false,sizeof(liar)) ;//所有人都不是liar
    for (int i=1;i<=m;i++){
        if (vec[i].empty()) nosay++ ;
        else
        {
            liar[i]=lie(i,0,d,x) ;
            for (int j=1;j<vec[i].size();j++){
                bool bf=lie(i,j,d,x);
                if (bf!=liar[i]) return false ;
                liar[i]=bf ;
```

```
            }
        }
    }
    int liesum=0;
    for (int i=1;i<=m;i++) if (liar[i]) liesum++ ;
    if (liesum+nosay==n || liesum==n) return true ;
    else return false ;
}
int main(){
    scanf("%d%d%d\n",&m,&n,&p) ;
    for (int i=1;i<=m;i++) {
        gets(name[i]);//他的名字
        name[i][strlen(name[i])-1]='\0' ;
    }
    for (int i=1;i<=p;i++){
        int id ;
        gets(talk) ;
        talk[strlen(talk)-1]='\0' ;
        int pos;
        pos=search(talk,':') ;//先查询，每个人的名字
        strncpy(who[i],talk,pos) ;
        for (int j=1;j<=m;j++) //说话者的编号
        if (same(who[i],name[j])){
            id=j ;
            break ;
        }
        strncpy(sayswhat[i],talk+pos+2,strlen(talk)) ;
        int f=false ;
        for (int j=0;j<=8;j++){ //他说的话的种类
            if (same(sayswhat[i],Says[j])){
                if (j>=0 && j<=6) vec[id].push_back(make_pair(j,0)) ;//星期
                else if (j==7 || j==8) vec[id].push_back(make_pair(j,0))
;//我的信息
                f=true ;
                break ;
            }
        }
        if (!f)
        for (int j=9;j<=10;j++){
            char *pos ;
            pos=strstr(sayswhat[i],Says[j]) ;
            if (pos!=NULL){
                strncpy(guilty[i],sayswhat[i],pos-sayswhat[i]-1) ;
                int ID ;
                for (int k=1;k<=m;k++) //说话者的编号
                if (same(guilty[i],name[k])){
                    ID=k ;
                    break ;
                }
```

```
                vec[id].push_back(make_pair(j,ID)) ;//别人的信息
            }
        }
    }
    int ans=-1 ;
    for (int j=1;j<=m;j++){
        bool f=false;
        for (int i=0;i<=6;i++){
            bool ok=judge(i,j) ;
            if (ok){
                if (ans==-1 && !f) {
                    ans=j ;
                    f=true;
                    break;
                }
                else if (ans!=-1) {
                    printf("Cannot Determine\n") ;
                    return 0 ;
                }
            }
        }
    }
    if (ans==-1) printf("Impossible\n") ;
    else printf("%s\n",name[ans]) ;
}
```

P1040 加分二叉树

题目描述：

设一个n个节点的二叉树tree的中序遍历为（1,2,3,...,n），其中数字1,2,3,...,n为节点编号。每个节点都有一个分数（均为正整数），记第i个节点的分数为di,tree及它的每个子树都有一个加分，任一棵子树subtree（也包含tree本身）的加分计算方法如下：

subtree的左子树的加分×subtree的右子树的加分＋subtree的根的分数。

若某个子树为空，规定其加分为1，叶子的加分就是叶节点本身的分数。不考虑它的空子树。

试求一棵符合中序遍历为（1,2,3,...,n）且加分最高的二叉树tree。要求输出；

（1）tree的最高加分

（2）tree的前序遍历

输入格式：

第1行：

1个整数n(n<30)，为节点个数。

第2行：

n个用空格隔开的整数，为每个节点的分数（分数<100）。

输出格式：

第1行：

1个整数，为最高加分（Ans ≤4,000,000,000）。

第2行：n个用空格隔开的整数，为该树的前序遍历。

输入输出样例：

```
5
5 7 1 2 10

145
3 1 2 4 5
```

思路

- 前序遍历：先访问根再访问左子树再访问右子树，简称：根左右（性质：遍历出的序列的第一个节点是整个tree的根）
- 中序遍历：左根右（性质：遍历出的序列的中间某个节点是整个树的根）
- 后序遍历：左右根（性质：遍历出来的序列的末尾一定是整个tree的根）
- 其实就是让你在整个序列中选定一个根节点，求出此节点左子树的最大分数，与右子树的最大分数。说白了就是让你枚举子树的根。 既然知道了是枚举子树的根，那做法很显然了

Solution

```
#include <bits/stdc++.h>//万能头文件
#define ll long long
//中序遍历具有的性质：在序列中必有且只有一个节点是根，使得此节点左边的序列是左子树的中序遍历，右边序列是右子树的中序遍历
using namespace std;
int n;
int score[32];//没记错的话score这个单词意思应该是分数，用它来表示每个节点的分数
ll f[32][32];//用于保存一个区间内最大分数
int root[32][32];//保存根，题目要求输出
ll ans=0;
ll dfs(int l,int r){
    if(f[l][r]!=-1)return f[l][r];
    if(l>r||r<l){
        return 1;
    }//注意好递归边界！！！  dfs到l>r或r<l,意味着子树为空，题目要求返回1
    ll sum=0;
    for(int i=l;i<=r;i++){
        if(dfs(l,i-1)*dfs(i+1,r)+score[i]>sum){
            sum=dfs(l,i-1)*dfs(i+1,r)+score[i];//要求[l,r]最大分数，先求[l,i-1],与[i+1,r]的最大分数
            root[l][r]=i;//记录根节点
        }
```

```
    }
    f[l][r]=sum;
    return f[l][r];
}
void print(int l,int r){
    if(l>r||r<l)return ;//注意，空子树不需要输出，空格都不用
    if(l==r){
        cout<<l<<" ";
        return ;
    }
    cout<<root[l][r]<<" ";
    print(l,root[l][r]-1);
    print(root[l][r]+1,r);
}//递归输出前序遍历
int main(){
    memset(f,-1,sizeof(f));//初始化为-1，因为n可能为0
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>score[i];
        f[i][i]=score[i];//区间[i,i]表示i是叶子节点，题目要求保留原来的分数
    }
    cout<<dfs(1,n)<<endl;
    print(1,n);

}
```

P1041 传染病控制

题目描述：

近来，一种新的传染病肆虐全球。蓬莱国也发现了零星感染者，为防止该病在蓬莱国大范围流行，该国政府决定不惜一切代价控制传染病的蔓延。不幸的是，由于人们尚未完全认识这种传染病，难以准确判别病毒携带者，更没有研制出疫苗以保护易感人群。于是，蓬莱国的疾病控制中心决定采取切断传播途径的方法控制疾病传播。经WHO

（世界卫生组织）以及全球各国科研部门的努力，这种新兴传染病的传播途径和控制方法已经研究清楚，剩下的任务就是由你协助蓬莱国疾控中心制定一个有效的控制办法。

研究表明，这种传染病的传播具有两种很特殊的性质；

第一是它的传播途径是树型的，一个人X只可能被某个特定的人Y感染，只要Y不得病，或者是XY之间的传播途径被切断，则X就不会得病。

第二是，这种疾病的传播有周期性，在一个疾病传播周期之内，传染病将只会感染一代患者，而不会再传播给下一代。

这些性质大大减轻了蓬莱国疾病防控的压力，并且他们已经得到了国内部分易感人群的潜在传播途径图（一棵树）。但是，麻烦还没有结束。由于蓬莱国疾控中心人手不够，同时也缺乏强大的技术，以致他们在一个疾病传播周期内，只能设法切断一条传播途径，而没有被控制的传播途径就会引起更多的易感人群被感染（也就是与当前已经被感染的人有传播途径相连，且连接途径没有被切断的人

群）。当不可能有健康人被感染时，疾病就中止传播。所以，蓬莱国疾控中心要制定出一个切断传播途径的顺序，以使尽量少的人被感染。

你的程序要针对给定的树，找出合适的切断顺序。

输入格式：

输入格式：

第一行是两个整数

n(1≤n≤300)和p。

接下来p行，每一行有2个整数i和j，表示节点i和j间有边相连。（意即，第i人和第j人之间有传播途径相连）。其中节点1是已经被感染的患者。

输出格式：

1行，总共被感染的人数。

输入输出样例：

```
7 6
1 2
1 3
2 4
2 5
3 6
3 7


3
```

思路

- 首先，有一个明显的贪心：当前传染到第i代，一定要切断第i与第i+1代的关系，深度再深就不够优了，比如样例中第一次要切断1和3但不能且3和6。于是搜索的框架也就有了：暴力枚举每一层切断谁，到最后一层统计答案。注意一个细节就是要是父亲安全了，孩子也一定安全，故也要打上tag。

Solution

```
#include<bits/stdc++.h>
using namespace std;
#define pb push_back
int getint(){
    int x=0,f=1; char ch=getchar();
    while(ch>'9'||ch<'0'){if(ch=='-')f=-f; ch=getchar();}
    while(ch>='0'&&ch<='9'){x=x*10+ch-'0'; ch=getchar();}
    return f*x;
}
const int MAXN=555;
```

```
const int inf=0x3f3f3f3f;
vector <int> g[MAXN],stor[MAXN];
int ans=inf,n,p,maxdep,fa[MAXN],tag[MAXN],deep[MAXN];
void build(){//build up the tree
    queue <int> q;
    q.push(1),stor[1].pb(1),deep[1]=1;
    while(!q.empty()){
        int cur=q.front();
        maxdep=max(maxdep,deep[cur]);
        q.pop();
        for(int i=0;i<g[cur].size();++i)
            if(g[cur][i]!=fa[cur]){
                deep[g[cur][i]]=deep[cur]+1;
                fa[g[cur][i]]=cur;
                stor[deep[g[cur][i]]].pb(g[cur][i]);
                q.push(g[cur][i]);
            }
    }
}
void dfs(int st,int sum){
    if(sum>ans) return;
    if(st>maxdep){
        ans=min(ans,sum);
        return;
    }
    int tmp=0;
    for(int i=0;i<stor[st].size();++i)//push tag if fa has tag
        if(tag[fa[stor[st][i]]])
            tmp++,tag[stor[st][i]]=st;
    if(tmp==stor[st].size()){
        ans=min(ans,sum);
        return;
    }
    for(int i=0;i<stor[st].size();++i){//try protect each node
        if(tag[stor[st][i]]) continue;
        tag[stor[st][i]]=true;
        dfs(st+1,sum+stor[st].size()-tmp-1);
        tag[stor[st][i]]=false;//注意post时还原pre时的操作
    }
    for(int i=1;i<=n;++i)//注意post时还原pre时的操作
        if(tag[i]==st) tag[i]=0;
}
int main(){
    n=getint(),p=getint();
    for(int i=1;i<=p;++i){//read in
        int f=getint(),s=getint();
        g[f].pb(s),g[s].pb(f);
    }
    build();dfs(2,1);
```

```
    cout<<ans<<endl;
    return 0;
}
```

P1042 乒乓球

题目描述：

华华通过以下方式进行分析，首先将比赛每个球的胜负列成一张表，然后分别计算在11分制和21分制下，双方的比赛结果（截至记录末尾）。

比如现在有这么一份记录，（其中`W`表示华华获得一分，`L`表示华华对手获得一分）：

WWWWWWWWWWWWWWWWWWWWWWWLW

在11分制下，此时比赛的结果是华华第一局11比0获胜，第二局11比0获胜，正在进行第三局，当前比分1比1。而在21分制下，此时比赛结果是华华第一局21比0获胜，正在进行第二局，比分2比1。如果一局比赛刚开始，则此时比分为0比0。直到分差大于或者等于2，才一局结束。

你的程序就是要对于一系列比赛信息的输入（WL形式），输出正确的结果。

输入格式：

输入格式：

每个输入文件包含若干行字符串，字符串有大写的W、L和E组成。其中E表示比赛信息结束，程序应该忽略E之后的所有内容。

输出格式：

输出由两部分组成，每部分有若干行，每一行对应一局比赛的比分（按比赛信息输入顺序）。其中第一部分是11分制下的结果，第二部分是21分制下的结果，两部分之间由一个空行分隔。

输入输出样例：

```
WWWWWWWWWWWWWWWWWWWWW
WWLWE

11:0
11:0
1:1

21:0
2:1
```

思路

- 先判断11分
- 再判断21分

Solution

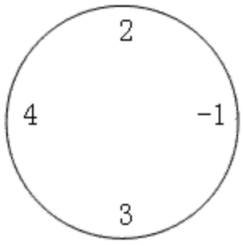
```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#define maxn 1000010
using namespace std;//以上不解释。
char s[maxn];//定义输入的字符串。
long long hh,ds,sum;//定义华华的分值与对手的分值与判断循环次数的辅助变量sum。
int main()
{
    for(int i=1;;i++)
    {
    cin>>s[i];
    if(s[i]=='E')break;
    sum++;
    }//按照题目要求输入。
    for(int i=1;i<=sum;i++)
    {
        if(s[i]=='W')hh++;//如果当前字符是W,那么将华华的分值加一。
        if(s[i]=='L')ds++;//同理对手分值加一。
        if((hh>=11&&hh-ds>=2)||(ds>=11&&ds-hh>=2))//如果华华或者对手有一个达到了11分且华华对手分差大于等于2，执行输出并清零。
        {
            cout<<hh<<":"<<ds<<endl;//输出。
            hh=0;//将华华分值清零。
            ds=0;//将对手分值清零。
        }
    }
        cout<<hh<<":"<<ds<<endl<<endl;//处理余分问题。
        hh=0;ds=0;//清零。
        for(int i=1;i<=sum;i++)//以下为判断二十一分值下胜负结果。
        {
        if(s[i]=='W')hh++;//加分。
        if(s[i]=='L')ds++;//加分。
        if((hh>=21&&hh-ds>=2)||(ds>=21&&ds-hh>=2))//与上同理判断。
        {
            cout<<hh<<":"<<ds<<endl;//输出。
            hh=0;//清零。
            ds=0;//清零。
        }
        }
    cout<<hh<<":"<<ds;//处理余数问题。
    return 0;//我爱return!!
}
```

P1043 数字游戏

题目描述：

丁丁最近沉迷于一个数字游戏之中。这个游戏看似简单，但丁丁在研究了许多天之后却发觉原来在简单的规则下想要赢得这个游戏并不那么容易。游戏是这样的，在你面前有一圈整数（一共n个），你要按顺序将其分为m个部分，各部分内的数字相加，相加所得的m个结果对10取模后再相乘，最终得到一个数k。游戏的要求是使你所得的k最大或者最小。

例如，对于下面这圈数字（$n=4, m=2$）：



要求最小值时，$((2-1) \bmod 10) \times ((4+3) \bmod 10) = 1 \times 7 = 7$，要求最大值时，为 $((2+4+3) \bmod 10) \times (-1 \bmod 10) = 9 \times 9 = 81$。特别值得注意的是，无论是负数还是正数，对10取模的结果均为非负值。

丁丁请你编写程序帮他赢得这个游戏。

输入格式：

输入文件第一行有两个整数，n(1≤n≤50)和m(1≤m≤9)。以下n行每行有个整数，其绝对值≤104，按顺序给出圈中的数字，首尾相接。

输出格式：

输出文件有2行，各包含1个非负整数。第1行是你程序得到的最小值，第2行是最大值

输入输出样例：

```
4 2
4
3
-1
2

7
81
```

思路

- 动态规划
- 破环成链。没有太多的技巧性，具体而言就是把数据存储两遍，使得环形的数据可以链式展开，便于我们去DP。但最后一定要记得扫一遍答案，取F[i][i+N-1]，i：1->N中的最大/小值。
- 前缀和。这个东西并不是在所有情况下都适用，但使用起来真的很方便，可以把O(n)的复杂度优化为O(1)。不过只适用于需要把数据直接相加的地方，比如说这道题。
- 初始化。这里实际上包括两点，一方面是在某些特殊情况下需要初始化，初始化为某特定值（比如本题只分成1段的时候）。另一方面也就是数组初始化，求最大值的时候根本不用管（因为初始默认为0），在求最小值的时候把数组全部赋初值为极大值就好啦。
- 状态表达。一般来说可以用F[i][j]表示在区间[i,j]中怎么怎么样，但由于本题还加了一个分为几段

的状态，就把数组直接加一维就好了。

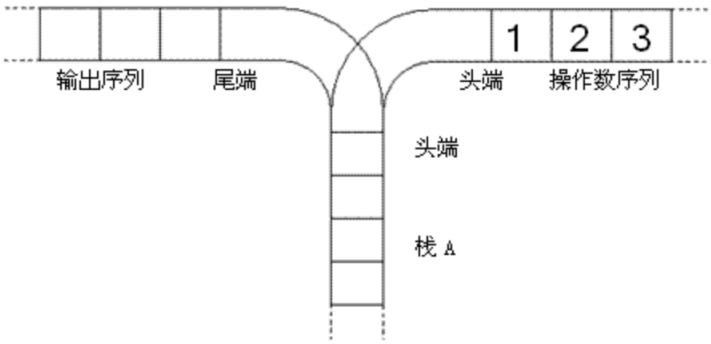
Solution

```
#include<cstdio>
#include<algorithm>
#include<cstring>
#define oo 2147483647//是个好习惯，使程序显得有条理一点
using namespace std;
int B[101][101][11],S[101][101][11];//区间[l,r]内分成i段的最大/小值
int n,m;
int a[101];//a存放前缀和
int mod(int a)//写成函数方便一点
{
    return ((a%10)+10)%10;
}
int main()
{
    scanf("%d %d",&n,&m);
    for (int i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        a[i+n]=a[i];
    }
    for (int i=1;i<=2*n;i++)
      a[i]+=a[i-1];//前缀和
    for (int l=1;l<=2*n;l++)
      for (int r=l;r<=2*n;r++)
        B[l][r][1]=S[l][r][1]=mod(a[r]-a[l-1]);//初始化不分段的状态
    for (int i=2;i<=m;i++)
      for (int l=1;l<=2*n;l++)
        for (int r=l+i-1;r<=2*n;r++)
          S[l][r][i]=oo;//求最小值时记得把数组初始化为极大值
    for (int i=2;i<=m;i++)//枚举分段数
      for (int l=1;l<=2*n;l++)//枚举左端点
        for (int r=l+i-1;r<=2*n;r++)//枚举右端点
        {
            for (int k=l+i-2;k<r;k++)//枚举区间断点 注意范围
            {
                S[l][r][i]=min (S[l][r][i],S[l][k][i-1]*mod(a[r]-a[k]));
                B[l][r][i]=max (B[l][r][i],B[l][k][i-1]*mod(a[r]-a[k]));
          }
        }
    int Max=0,Min=oo;//答案初始化
    for (int i=1;i<=n;i++)
    {
        Max=max(Max,B[i][i+n-1][m]);//从前往后扫一遍
        Min=min(Min,S[i][i+n-1][m]);
    }
```

```
    printf("%d\n%d",Min,Max);
    return 0;
}
```

P1044 栈(Catalan数)

题目描述：



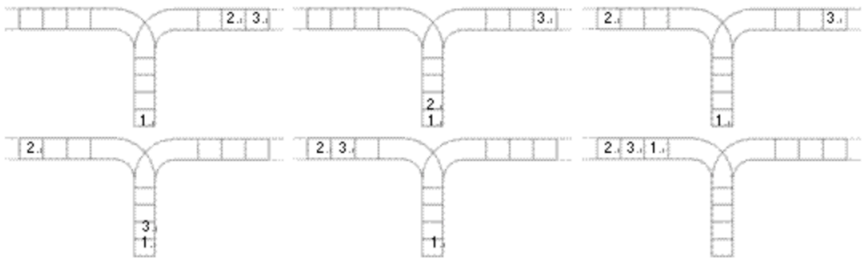
宁宁考虑的是这样一个问题：一个操作数序列，$1,2,...,n$（图示为1到3的情况），栈A的深度大于n。

现在可以进行两种操作，

1. 将一个数，从操作数序列的头端移到栈的头端（对应数据结构栈的$push$操作）

2. 将一个数，从栈的头端移到输出序列的尾端（对应数据结构栈的pop操作）

使用这两种操作，由一个操作数序列就可以得到一系列的输出序列，下图所示为由123生成序列231的过程。



（原始状态如上图所示）

你的程序将对给定的n，计算并输出由操作数序列$1,2,...,n$经过操作可能得到的输出序列的总数。

输入格式：

输入文件只含一个整数n(1≤n≤18)

输出格式：

输出文件只有

1行，即可能输出序列的总数目

输入输出样例：

```
3 5
```

思路

- 卡特兰数
- h(0)=h(1)=1

  h(n)= h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)*h(0) (n>=2)

Solution

```
#include<cstdio>
#define siz 20
using namespace std;
int n;
int c[siz*2][siz];
int main(){

    scanf("%d",&n);
    for(int i=1;i<=2*n;i++) c[i][1]=c[i][i]=1;
    for(int i=3;i<=2*n;i++)
     for(int j=2;j<i;j++)
      c[i][j]=c[i-1][j]+c[i-1][j-1];
    printf("%d",c[2*n][n]-c[2*n][n-1]);
    return 0;
}
```

P1045 麦森数（高精度快速幂）

题目描述：

形如2^P−1的素数称为麦森数，这时P一定也是个素数。但反过来不一定，即如果P是个素数，2P−1不一定也是素数。到1998年底，人们已找到了37个麦森数。最大的一个是P=3021377，它有909526位。麦森数有许多重要应用，它与完全数密切相关。

任务：从文件中输入P（1000<P<3100000），计算2P−1的位数和最后500位数字（用十进制高精度数表示）

输入格式：

文件中只包含一个整数P（1000<P<3100000）

输出格式：

第一行：十进制高精度数2^P−1的位数。

第2-11行：十进制高精度数2^P−1的最后500位数字。（每行输出50位，共输出10行，不足500位时高位补0）

不必验证2^P−1与P是否为素数。

输入输出样例：

```
1279

386
00000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
00000000000000104079321946643990819252403273640855
38615262247266704805319112350403608059673360298012
23944173232418484242161395428100779138356624832346
49081399066056773207629241295093892203457731833496
61583550472959420547689811211693677147548478866962
50138443826029173234888531116082853841658502825560
46662248318909188018470682222031405210266984354887
32958028878050869736186900714720710555703168729087
```

思路

- 高精度 快速幂

Solution

```
#include<cstdio>
#include<cmath>
#include<cstring>
using namespace std;
int f[1001],p,res[1001],sav[1001];//乘法要开两倍长度
void result_1()
{
    memset(sav,0,sizeof(sav));
    for(register int i=1;i<=500;i+=1)
        for(register int j=1;j<=500;j+=1)
            sav[i+j-1]+=res[i]*f[j];//先计算每一位上的值（不进位）
    for(register int i=1;i<=500;i+=1)
    {
        sav[i+1]+=sav[i]/10;//单独处理进位问题，不容易出错
        sav[i]%=10;
    }
    memcpy(res,sav,sizeof(res));//cstring库里的赋值函数，把sav的值赋给res
}
void result_2()//只是在result_1的基础上进行了细微的修改
{
    memset(sav,0,sizeof(sav));
    for(register int i=1;i<=500;i+=1)
        for(register int j=1;j<=500;j+=1)
            sav[i+j-1]+=f[i]*f[j];
    for(register int i=1;i<=500;i+=1)
    {
```

```
            sav[i+1]+=sav[i]/10;
            sav[i]%=10;
        }
        memcpy(f,sav,sizeof(f));
    }
    int main()
    {
        scanf("%d",&p);
        printf("%d\n",(int)(log10(2)*p+1));
        res[1]=1;
        f[1]=2;//高精度赋初值
        while(p!=0)//快速幂模板
        {
            if(p%2==1)result_1();
            p/=2;
            result_2();
        }
        res[1]-=1;
        for(register int i=500;i>=1;i-=1)//注意输出格式，50个换一行，第一个不用
            if(i!=500&&i%50==0)printf("\n%d",res[i]);
            else printf("%d",res[i]);
        return 0;
    }
```

P1046 陶陶摘苹果

题目描述：

陶陶家的院子里有一棵苹果树，每到秋天树上就会结出10个苹果。苹果成熟的时候，陶陶就会跑去摘苹果。陶陶有个30厘米高的板凳，当她不能直接用手摘到苹果的时候，就会踩到板凳上再试试。

现在已知10个苹果到地面的高度，以及陶陶把手伸直的时候能够达到的最大高度，请帮陶陶算一下她能够摘到的苹果的数目。假设她碰到苹果，苹果就会掉下来。

输入格式：

输入包括两行数据。第一行包含10个100到200之间（包括100和200）的整数（以厘米为单位）分别表示10个苹果到地面的高度，两个相邻的整数之间用一个空格隔开。第二行只包括一个100到120之间（包含100和120）的整数（以厘米为单位），表示陶陶把手伸直的时候能够达到的最大高度。

输出格式：

输出包括一行，这一行只包含一个整数，表示陶陶能够摘到的苹果的数目。

输入输出样例：

```
100 200 150 140 129 134 167 198 200 111
110

5
```

思路

- 入门级难度

Solution

```
#include <iostream>
using namespace std;
int main() {
    int a[10];
    int max;
    for(int i = 0; i < 10; i++) {
        cin >>a[i];
    }
    cin >> max;
    int res = 0;
    for(int i = 0; i < 10; i++) {
        if(a[i] <= max + 30) {
            res++;
        }
    }
    cout << res;
    return 0;
}
```

P1047 校门外的树

题目描述：

某校大门外长度为L的马路上有一排树，每两棵相邻的树之间的间隔都是1米。我们可以把马路看成一个数轴，马路的一端在数轴0的位置，另一端在L的位置；数轴上的每个整数点，即0,1,2,...,L，都种有一棵树。

由于马路上有一些区域要用来建地铁。这些区域用它们在数轴上的起始点和终止点表示。已知任一区域的起始点和终止点的坐标都是整数，区域之间可能有重合的部分。现在要把这些区域中的树（包括区域端点处的两棵树）移走。你的任务是计算将这些树都移走后，马路上还有多少棵树。

输入格式：

第一行有2个整数L(1≤L≤10000)和M(1≤M≤100)，L代表马路的长度，M代表区域的数目，L和M之间用一个空格隔开。接下来的M行每行包含2个不同的整数，用一个空格隔开，表示一个区域的起始点和终止点的坐标。

输出格式：

1个整数，表示马路上剩余的树的数目。

输入输出样例：

```
500 3
150 300
100 200
470 471

298
```

思路

- 入门级难度

Solution

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
    int L, n;
    cin >>L >>n;
    bool flag[L+1];
    int x,y;
    memset(flag,0,sizeof(flag));//初始化该数组
    for(int i =0; i < n; i++) {
        cin >>x >> y;
        for(int j = x; j <= y; j++) {
            flag[j] = 1;
        }
    }
    int res = 0;
    for(int i = 0; i <= L; i++) {
        if(flag[i] == 0) {
            res += 1;
        }
    }
    cout << res;
    return 0;
}
```

P1048 采药(01背包模版)

题目描述：

辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗?

输入格式:

第一行有2个整数T(1≤T≤1000)和M(1≤M≤100，用一个空格隔开，T代表总共能够用来采药的时间，M代表山洞里的草药的数目。接下来的M行每行包括两个在1到100之间（包括1和100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出格式:

1个整数，表示在规定的时间内可以采到的草药的最大总价值。

输入输出样例:

```
70 3
71 100
69 1
1 2

3
```

思路

- 最基本的01背包问题

Solution

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cstdio>
const int maxn = 1010;
const int maxw = 1010;
int n, w;
int c[maxn], v[maxn];//c为费用，v为价值
int dp[maxn][maxw];
using namespace std;
int main()
{
    scanf("%d %d", &w, &n);//w为最大费用，n为数量
    for(int i = 1;i <= n;i++)
    {
        scanf("%d %d", &c[i], &v[i]);//输入，注意这里的下标是从1开始的
    }
    memset(dp, 0, sizeof(dp));//若不涉及多组输入，这一步其实可以省略
    //如果下标从0开始，下面也需要稍作修改
    for(int i = 1;i <= n;i++)
    {
        for(int j = 0;j <= w;j++)
        {
```

```
            if(c[i] > j)
                dp[i][j] = dp[i - 1][j];//状态转移，情况①
            else
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - c[i]] + v[i]);//
情况②
        }
    }
    printf("%d\n", dp[n][w]);
    return 0;
}
```

P1049 装箱问题

题目描述：

有一个箱子容量为V（正整数，0≤V≤20000），同时有n个物品（0<n≤30，每个物品有一个体积（正整数）。

要求n个物品中，任取若干个装入箱内，使箱子的剩余空间为最小。

输入格式：

1个整数，表示箱子容量

1个整数，表示有n个物品

接下来n行，分别表示这n个物品的各自体积

输出格式：

1个整数，表示箱子剩余空间。

输入输出样例：

```
24
6
8
3
12
7
9
7


0
```

思路

- 最基本的01背包问题，要善于套用模版

Solution

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

const int maxn = 40;
const int maxw = 20010;
int n, w;
int c[maxn], v[maxn];//c为费用，v为价值
int dp[maxn][maxw];

using namespace std;

int main()
{
    scanf("%d %d", &w, &n);//w为最大费用，n为数量
    for(int i = 1;i <= n;i++)
    {
        scanf("%d", &c[i]);//输入，注意这里的下标是从1开始的
        v[i] = c[i];
    }
    memset(dp, 0, sizeof(dp));//若不涉及多组输入，这一步其实可以省略
    //如果下标从0开始，下面也需要稍作修改
    for(int i = 1;i <= n;i++)
    {
        for(int j = 0;j <= w;j++)
        {
            if(c[i] > j)
                dp[i][j] = dp[i - 1][j];//状态转移，情况①
            else
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - c[i]] + v[i]);//
情况②
        }
    }
    printf("%d\n", w - dp[n][w]);
    return 0;
}
```

P1050 循环

题目描述：

乐乐是一个聪明而又勤奋好学的孩子。他总喜欢探求事物的规律。一天，他突然对数的正整数次幂产生了兴趣。

众所周知，2的正整数次幂最后一位数总是不断的在重复2,4,8,6,2,4,8,6...我们说2的正整数次幂最后一位的循环长度是4（实际上4的倍数都可以说是循环长度，但我们只考虑最小的循环长度）。类似的，其余的数字的正整数次幂最后一位数也有类似的循环现象：

```
    循环     循环长度
2   2,4,8,6   4
3   3,9,7,1   4
4   4,6       2
5   5         1
6   6         1
7   7,9,3,1   4
8   8,4,2,6   4
9   9,1       2
```

这时乐乐的问题就出来了：是不是只有最后一位才有这样的循环呢？对于一个整数n的正整数次幂来说，它的后k位是否会发生循环？如果循环的话，循环长度是多少呢？

注意：

1. 如果n的某个正整数次幂的位数不足k，那么不足的高位看做是0。

2. 如果循环长度是L，那么说明对于任意的正整数a,n的a次幂和a+L次幂的最后k位都相同。

输入格式：

一行，包含2个整数n(1≤n<10^100)和k(1≤k≤100)，n和k之间用一个空格隔开，表示要求n的正整数次幂的最后k位的循环长度。

输出格式：

一个整数，表示循环长度。如果循环不存在，输出-1。

输入输出样例：

```
32 2


4
```

思路

- 难度较大，高精度
- 真正需要计算的部分就是后k位了，我们可以从尾来分析→既后1位，后2位，后3位，后4位.....递推去找

Solution

```
#include<cstdio>
#include<cstring>
int ans[101];
int a[101],n=0,k;
int i,j,rec[101],newn[101];
char ch[101];
bool judge(int k) //计算是否循环
{
```

```
    int s[102];
    memset(s,0,sizeof(s));
    for (int i=1; i<=n; i++)
      for (int j=1; j<=newn[0]; j++)
        if (i+j-1 > k) break;
        else
        {
           s[i+j-1]+=a[i]*newn[j];
           s[i+j]+=s[i+j-1]/10;
           s[i+j-1]%=10;
        }
    if (s[k] == a[k]) return 1;
    return 0;
}
void mul() //累乘
{
    int s[102];
    memset(s,0,sizeof(s));
    for (int i=1; i<=rec[0]; i++)
      for (int j=1; j<=newn[0]; j++)
        if (i+j-1 <= n)
        {
           s[i+j-1]+=rec[i]*newn[j];
           s[i+j]+=s[i+j-1]/10;
           s[i+j-1]%=10;
        }
    if (rec[0]+newn[0]-1 >= n) s[0]=n;
    else if (s[rec[0]+newn[0]] > 0) s[0]=rec[0]+newn[0];
    else s[0]=rec[0]+newn[0]-1;
    for (int i=0; i<=s[0]; i++)
      newn[i] = s[i];
}
void calc(int x) //单精*高精
{
    for (int i=1; i<=ans[0]; i++)
      ans[i]*=x;
    for (int i=1; i<=ans[0]; i++)
    {
        ans[i+1]+=ans[i]/10;
        ans[i]%=10;
    }
    while (ans[ans[0]+1] > 10)
    {
        ans[0]++;
        ans[ans[0]+1]+=ans[ans[0]]/10;
        ans[ans[0]]%=10;
    }
    if (ans[ans[0]+1] > 0) ans[0]++;
}
```

```
int main()
{
    scanf("%c",&ch[++n]);
    while (ch[n] != ' ')
      scanf("%c",&ch[++n]);
    scanf("%d",&k);
    n--;
    for (i=1; i<=n; i++)
      a[i]=ch[n+1-i]-'0';
    if (n > k) n=k;
    if (n < k)
    {
      for (i=n+1; i<=k; i++)
        a[i] = 0;
      n = k;
    } //init
    for (i=1; i<=n; i++)
      rec[i]=a[i]; //last one 加数
    rec[0]=n;
    for (i=1; i<=n; i++)
      newn[i]=a[i];
    newn[0]=n; //rec^x (累乘器)
    ans[0]=1; ans[1]=1;
    for (i=1; i<=k; i++)
    {
        if (judge(i)) continue; //第i位相等
        else for (j=2; j<=11; j++)
        {
            mul(); //累乘器
            if (judge(i)) break;
        }
        if (j > 11) //无解
        {
            printf("-1");
            return 0;
        }
        calc(j); //ans=ans*j;
        for (j=newn[0]; j>=0; j--)
          rec[j]=newn[j];
    }
    for (i=ans[0]; i>=1; i--)
      printf("%d",ans[i]);
    return 0;
}
```

P1051 谁拿了最多奖学金

题目描述：

某校的惯例是在每学期的期末考试之后发放奖学金。发放的奖学金共有五种，获取的条件各自不同：

院士奖学金，每人8000元，期末平均成绩高于80分（>80），并且在本学期内发表1篇或1篇以上论文的学生均可获得；

五四奖学金，每人4000元，期末平均成绩高于85分（>85），并且班级评议成绩高于80分（>80）的学生均可获得；

成绩优秀奖，每人2000元，期末平均成绩高于90分（>90）的学生均可获得；

西部奖学金，每人1000元，期末平均成绩高于85分（>85）的西部省份学生均可获得；

班级贡献奖，每人850元，班级评议成绩高于80分（>80）的学生干部均可获得；

只要符合条件就可以得奖，每项奖学金的获奖人数没有限制，每名学生也可以同时获得多项奖学金。例如姚林的期末平均成绩是87分，班级评议成绩82分，同时他还是一位学生干部，那么他可以同时获得五四奖学金和班级贡献奖，奖金总数是48504850元。

现在给出若干学生的相关数据，请计算哪些同学获得的奖金总数最高（假设总有同学能满足获得奖学金的条件）。

输入格式：

第一行是1个整数N(1≤N≤100)，表示学生的总数。

接下来的N行每行是一位学生的数据，从左向右依次是姓名，期末平均成绩，班级评议成绩，是否是学生干部，是否是西部省份学生，以及发表的论文数。姓名是由大小写英文字母组成的长度不超过20的字符串（不含空格）；期末平均成绩和班级评议成绩都是0到100之间的整数（包括0和100）；是否是学生干部和是否是西部省份学生分别用1个字符表示，Y表示是，N表示不是；发表的论文数是0到10的整数（包括0和10）。每两个相邻数据项之间用一个空格分隔。

输出格式：

包括3行。

第1行是获得最多奖金的学生的姓名。

第2行是这名学生获得的奖金总数。如果有两位或两位以上的学生获得的奖金最多，输出他们之中在输入文件中出现最早的学生的姓名。

第3行是这N个学生获得的奖学金的总数。

输入输出样例：

```
4
YaoLin 87 82 Y N 0
ChenRuiyi 88 78 N Y 1
LiXin 92 88 N N 0
ZhangQin 83 87 Y N 1

ChenRuiyi
9000
28700
```

思路

- 思路简单，可以设置学生结构体

Solution

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n,score1,score2,sum=0,max=0,total=0,x,i;
    char a,b;
    string name,maxn;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>name>>score1>>score2>>a>>b>>x;
        if(score1>80 && x>0)//判断是否获得院士奖学金
            sum+=8000;
        if(score1>85 && score2>80)//判断是否获得五四奖学金
            sum+=4000;
        if(score1>90)//判断是否获得成绩优秀奖
            sum+=2000;
        if(score1>85 && b=='Y')//判断是否获得西部奖学金
            sum+=1000;
        if(score2>80 && a=='Y')//判断是否获得班级贡献奖
            sum+=850;
        total+=sum;//累加奖学金
        if(sum>max)//找出最牛学生
            maxn=name,max=sum;//sum的用处
        sum=0;
    }
    cout<<maxn<<endl<<max<<endl<<total;
    return 0;
}
```

P1052 过河

题目描述：

在河上有一座独木桥，一只青蛙想沿着独木桥从河的一侧跳到另一侧。在桥上有一些石子，青蛙很讨厌踩在这些石子上。由于桥的长度和青蛙一次跳过的距离都是正整数，我们可以把独木桥上青蛙可能到达的点看成数轴上的一串整点：0,1,...,L（其中L是桥的长度）。坐标为0的点表示桥的起点，坐标为L的点表示桥的终点。青蛙从桥的起点开始，不停的向终点方向跳跃。一次跳跃的距离是S到T之间的任意正整数（包括S,T）。当青蛙跳到或跳过坐标为L的点时，就算青蛙已经跳出了独木桥。

题目给出独木桥的长度L，青蛙跳跃的距离范围S,T，桥上石子的位置。你的任务是确定青蛙要想过河，最少需要踩到的石子数。

输入格式：

第一行有1个正整数L(1≤L≤10^9)，表示独木桥的长度。

第二行有3个正整数S,T,M，分别表示青蛙一次跳跃的最小距离，最大距离及桥上石子的个数，其中1≤S≤T≤10,1≤M≤100。

第三行有M个不同的正整数分别表示这M个石子在数轴上的位置（数据保证桥的起点和终点处没有石子）。所有相邻的整数之间用一个空格隔开。

输出格式：

一个整数，表示青蛙过河最少需要踩到的石子数。

输入输出样例：

```
10
2 3 5
2 3 5 6 7

2
```

思路

- 首先设f[i]为在i点上的最少踩石子数则在前面(i-s)到(i-t)的点都可以改变i点的值,因此我们可以取f[i-s]-f[i-t]之中的最小值，另外如果有石头就加上1，如果没有就不加值，这里我们直接用flag[i]表示该点有无石头(有则为1，无则为0)。
- 因此我们可以写出状态转移方程式:

  $ f[i]=\min (f[i-j]+\operatorname{flag}[i] | s<=j<=t) $
- 路径压缩：

  $ f[i]=f[i-1]+(i \bmod s=0) $

Solution

```
#include<iostream>
#include<algorithm>
using namespace std;
int a[105],d[105],stone[350000];
int f[350000]; //压缩路径后f[]就不必开10^9那么大了
int main()
{
    int l,s,t,m;
    cin>>l>>s>>t>>m;
    for (int i=1;i<=m;i++)
        cin>>a[i];
    sort(a+1,a+m+1); //输入石子坐标可能无序
    for (int i=1;i<=m;i++)
        d[i]=(a[i]-a[i-1])%2520; //要对1~10的最小公倍数取余，压缩路径的核心
```

```
    for (int i=1;i<=m;i++)
    {
        a[i]=a[i-1]+d[i];
        stone[a[i]]=1; //此处有石子，标记
    }
    l=a[m]; //压缩路径后的总长度
    for (int i=0;i<=l+t;i++) f[i]=m; //f[i]表示到位置i最少能踩到的石子数
    f[0]=0;
    //以上是初始化，接下来是动归
    for (int i=1;i<=l+t;i++)
        for (int j=s;j<=t;j++)
        {
            if (i-j>=0)
                f[i]=min(f[i],f[i-j]); //状态转移方程
            f[i]+=stone[i];
        }
    int ans=m;
    for (int i=l;i<l+t;i++) ans=min(ans,f[i]);
    cout<<ans<<endl;
    return 0;
}
```