程序设计实训 练习5

马的遍历

描述

马的遍历.pdf

有一个n*m的棋盘，在某个(x,y) 上有一个马，要求你计算出马到达棋盘上任意一个点最少要走几步。

输入

输入只有一行四个整数，分别为n，m，x，y

输出

一个n*m的矩阵，代表马到达某个点最少要走几步（不能到达则输出 -1）

为统一输出的格式，输出的矩阵中的每个元素占用的宽度总计为5且左对齐，除数字外的部分使用空格。

提示：可使用printf("%-5d", xxx)；来输出结果矩阵中的每个元素。

输入样例 1

```
3 3 1 1
```

输出样例 1

```
0    3    2
3    -1   1
2    1    4
```

```
int n, m, a, b;
struct w   {    int x, y; };
bool st[401][401];    //对走过的格子进行标记
int dis[401][401];    //到达某格需要的步数
 //马的八种走法
int dx[401] = {0, 1, 2, 2, 1, -1, -2, -2, -1};
int dy[401] = {0, -2, -1, 1, 2, 2, 1, -1, -2};
 //广度优先搜索
void bfs(int bx, int by)        {
    dis[bx][by] = 0; //0步到起点
    queue<w> q;     q.push({bx, by});
    st[bx][by] = 1; //起点
    while (q.size())    {
        w t = q.front();
        q.pop();
        for (int i = 1; i <= 8; i++) //进行八种走法的遍历        {
            int ix = t.x + dx[i], iy = t.y + dy[i];
            if (ix >= 1 && iy >= 1 && ix <= n && iy <= m) //在矩阵内          {
                if (!st[ix][iy])  //没走过的格              {
                    dis[ix][iy] = dis[t.x][t.y] + 1;
                    st[ix][iy] = 1;
                    q.push({ix, iy});
                }
            }
        }
    }
}
```

BY 162210107 蔡蕾

奇怪的电梯

描述

main.pdf

呵呵有一天我做了一个梦，梦见了一种很奇怪的电梯。大楼的每一层楼都可以停电梯，而且第 i 层楼（ 1<=i<=N）上有一个数字 Ki（ 0<=Ki<=N ）。

电梯只有四个按钮：开，关，上，下。上下的层数等于当前楼层上的那个数字。当然，如果不能满足要求，相应的按钮就会失灵。

例如：3 3 1 2 5 代表了 Ki（ K1=3，K2=3，……），从 1 楼开始。在1楼按“上”可以到 4 楼，按“下”是不起作用的，因为没有 -2 楼。那么，从 A 楼到 B 楼至少要按几次按钮呢?

输入

共二行。

第一行为三个用空格隔开的正整数，表示 N，A，B（1<=N<-200，1<=A,B<=N ）。

第二行为 个用空格隔开的非负整数，表示 。

输出

一行，即最少按键次数，若无法到达，则输出-1

输入样例 1

```
5 1 5
3 3 1 2 5
```

输出样例 1

```
3
```

```
int a[6];
int main(void)      {
    queue<pair<int, int>> q;
    int N, A, B;
    cin >> N >> A >> B;
    vector<int> c(N + 1);
    for (int i = 1; i <= N; i++)
        cin >> c[i];
    q.push(make_pair(A, 0));
    int flag = 0;
    while (q.size())    {
        pair<int, int> temp = q.front();
        q.pop();
        if (temp.first == B)       {
            cout << temp.second;
            flag = 1;
            break;
        }
        a[temp.first] = 1;
        if (c[temp.first] != 0 && temp.first + c[temp.first] <= N && a[temp.first + c[temp.first]] == 0)       // 向上
            q.push(make_pair(temp.first + c[temp.first], temp.second + 1));
        if (c[temp.first] != 0 && temp.first - c[temp.first] > 0 && a[temp.first - c[temp.first]] == 0)       // 向下
            q.push(make_pair(temp.first - c[temp.first], temp.second + 1));
    }
    if (flag == 0)
        cout << -1;
    return 0;
}
```

BY 162230217 陈梓鹏

填涂颜色

题目描述

由数字 0 组成的方阵中，有一任意形状闭合圈，闭合圈由数字 1 构成，围圈时只走上下左右 4 个方向。现要求把闭合圈内的所有空间都填写成 2。例如：6×6 的方阵（$n = 6$），涂色前和涂色后的方阵如下：

```
0 0 0 0 0 0
0 0 1 1 1 1
0 1 1 0 0 1
1 1 0 0 0 1
1 0 0 0 0 1
1 1 1 1 1 1
```

```
0 0 0 0 0 0
0 0 1 1 1 1
0 1 1 2 2 1
1 1 2 2 2 1
1 2 2 2 2 1
1 1 1 1 1 1
```

核心思想，使用BFS/DFS算法均可，从一个点出发，寻找相邻的0。
（过程与走迷宫类似，将1当作障碍，0作为通路，所有能走到的点都说明是相邻的）

判断0是否在1内部较为困难。转化为判断是否存在不在1的包围圈内的0。

初始把所有0改为2，然后沿着边界寻找没有被1包围住的2，以及与其相邻的所有2。将这些2修改为0。

其它方法：
直接寻找到某一个被1围住的0
由上向下由左向右找到的第一个1的右下方向遇到的第一个0，一定是被包围住的0。找到这个0之后，由该位置出发进行BFS/DFS找到与其联通的所有0，并全部修改为2.

填涂颜色

```
#define maxn 35
int a[maxn][maxn],n;
int dx[4][2]={{1,0},{0,1},{-1,0},{0,-1}};   //四个移动方向

void dfs(int x,int y){
        a[x][y]=0;
        for(int i=0;i<4;i++){
                int ux = x+dx[i][0],  uy = y+dx[i][1];
                if(ux>=0 && ux<=n+1 && uy>=0 && uy<=n+1 && a[ux][uy]==2)
                        dfs(ux,uy);

        }
}
```

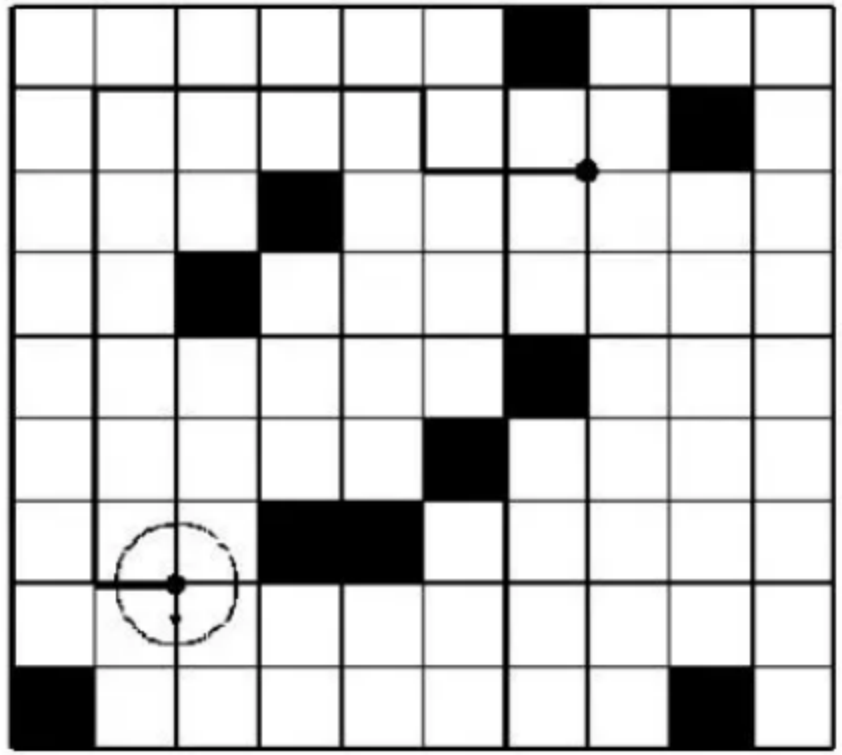
填涂颜色

```
int main()  {
        for(int i=0;i<35;i++)
            for(int j=0;j<35;j++)    a[i][j]=2;
        cin>>n;
        for(int i=1;i<=n;i++)   {
                for(int j=1;j<=n;j++)  {
                        int tmp;
                        cin>>tmp;
                        if(tmp==1)a[i][j]=1;            //读入数据，并将所有0改为2
                }
        }
        dfs(0,0);
        for(int i=1;i<=n;i++){
                for(int j=1;j<=n;j++)
                        cout<<a[i][j]<<" ";
                if(i!=n)cout<<endl;
        }
        return 0;
}
```

机器人搬重物

题目描述

机器人移动学会（`RMI`）现在正尝试用机器人搬运物品。机器人的形状是一个直径1.6米的球。在试验阶段，机器人被用于在一个储藏室中搬运货物。储藏室是一个 $N \times M$ 的网格，有些格子为不可移动的障碍。机器人的中心总是在格点上，当然，机器人必须在最短的时间内把物品搬运到指定的地方。机器人接受的指令有：向前移动1步（`Creep`）；向前移动2步（`Walk`）；向前移动3 步（`Run`）；向左转（`Left`）；向右转（`Right`）。每个指令所需要的时间为1 秒。请你计算一下机器人完成任务所需的最少时间。



输入格式

第一行为两个正整数$N, M(N, M \leq 50)$，下面N行是储藏室的构造，0表示无障碍，1表示有障碍，数字之间用一个空格隔开。接着一行有4个整数和1个大写字母，分别为起始点和目标点左上角网格的行与列，起始时的面对方向（东E，南S，西W，北N），数与数，数与字母之间均用一个空格隔开。终点的面向方向是任意的。

输出格式

一个整数，表示机器人完成任务所需的最少时间。如果无法到达，输出-1。

一、机器人是以所在点为中心的圆，障碍是方格，判断机器人是否与障碍碰撞

二、输入数据是以方格图形式给出，而不是按坐标点给出，需要将方格图转换为坐标点才更方便处理。比如统一取障碍方格的右上（也可选用其它位置坐标，确保统一即可）坐标进行记录，在判断碰撞的时候根据右上坐标点推导出方格其它三个坐标点一并进行判断。

三、机器人有三种前进方式（前进1步、前进2步、前进3步），和两种转向方式（左转、右转）。三种前进方式和两种转向都消耗一个单位的时间，在枚举状态时候需要一并考虑进去。
可以使用横/纵坐标前进方式数组+方向数组，简化枚举过程。

处理好以上内容后，即可按BFS进行搜索。

```
int tx[4][3]={{-1,-2,-3},{0,0,0},{1,2,3},{0,0,0},};
int ty[4][3]={{0,0,0},{-1,-2,-3},{0,0,0},{1,2,3},};    //对应四个方向移动1步、2步、3步的情况
int n,m,fx,fy,MAP[55][55],f[1000005][5];   // f 数组用来记录方向
void bfs() ;
int main()      {
        cin>>n>>m;
        for(int i=1;i<=n;i++)
                for(int j=1;j<=m;j++)    cin>>MAP[i][j];
        char c;
        cin>>f[1][1]>>f[1][2]>>fx>>fy>>c;
        MAP[f[1][1]][f[1][2]]=2;
        if(f[1][1]==fx&&f[1][2]==fy)    {
                cout<<0;
                return 0;
        }
        if(c=='N') f[1][3]=0;
        if(c=='W') f[1][3]=1;
        if(c=='S') f[1][3]=2;
        if(c=='E') f[1][3]=3;
        bfs();
        return 0;
}
```

```
void bfs()   {
        int head=0,tail=1;
        while(head<tail)   {
                head++;
                for(int i=0;i<=2;i++)  {
                        int xx=f[head][1]+tx[f[head][3]][i];
                        int yy=f[head][2]+ty[f[head][3]][i];
                        if(MAP[xx+1][yy]==1||MAP[xx][yy+1]==1||MAP[xx+1][yy+1]==1||MAP[xx][yy]==1) break;
                        if(xx>=1&&xx<n&&yy>=1&&yy<m&&MAP[xx][yy]==0)   {
                                MAP[xx][yy]=2;  tail++;
                                f[tail][1]=xx,f[tail][2]=yy,f[tail][3]=f[head][3],f[tail][4]=f[head][4]+1;
                                if(xx==fx&&yy==fy)              {
                                        cout<<f[tail][4]; return;
                                }
                        }
                }
                if( f[head][0]!=2)
                        for(int j=1;j<=3;j++)
                                if(j!=2)  {
                                        tail++;
                                f[tail][1]=f[head][1];  f[tail][2]=f[head][2];  f[tail][3]=(f[head][3]+j)%4;
                                f[tail][4]=f[head][4]+1; f[tail][0]=f[head][0]+1;
                                }
        }
        cout<<-1;
}
```

第一讲 程序设计语言回顾

一、数据类型
二、顺序、分支、循环结构
三、数组、字符串
四、函数、递归

五、黑盒测试
六、输入技巧

```
while (scanf(%d,&a)==1) { …… }
while (scanf(%d,&a)>0) { …… }
while (scanf(%d,&a)) { …… }

while (cin>>a) { …… }
while(c=getchar()) { …… }
```

七、算法时间复杂度

第二讲 排序和高精度计算

一、冒泡排序
交换相邻数据顺序实现排序

二、选择排序
选择剩余数据中最小（大）的数据实现排序

三、快速排序
确定一个关键字，将数据分为两部分，使其中一个部分的数据均比关键字小，另一部分的数据均比关键字大，再分别对这两部分重复以上操作，以达到整个序列有序。

四、桶排序
若待排序的值在一个明显有限范围内(整型)时，可设计有限个有序桶，待排序的值装入对应的桶（当然也可以装入若干个值），桶号就是待排序的值，顺序输出各桶的值，将得到有序的序列。

五、sort()函数
编写简单且效率高，可以通过自定义比较函数实现丰富的排序功能

六、高精度加法、减法、乘法
均为模拟竖式计算，注意数据输入、进位/借位、前导0等情况的处理

第三讲 基本算法思想

一、模拟算法

据题目描述，提取关键信息和要素，按题目要求完成代码编写。也可简单理解为“依葫芦画瓢”。处理过程可能较为繁琐，需要特别注意边界、特例和对题意的理解。

二、枚举算法

列举所有的可能情况；然后验证每个枚举结果是否是问题的解。

三、递推算法

数学思想，寻找相邻的数据项间的关系（即递推关系）。可能隐藏较为隐蔽，可通过小规模数据试算等寻找递推关系。

四、贪心算法

仅根据局部信息选择在当前范围内最优的解，不考虑全局状态。不一定总能通过一系列局部最优解得到全局最优解。

第三讲 基本算法思想

五、素数筛

通过在枢轴上划去已有数字的所有倍数，筛出给定范围内的全部素数

我们把2~n的数按顺序写出来：

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

从前往后看，找到第一个未被划掉的数，2，这说明它是质数。然后把2的倍数（不包括2）划掉：

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~

下一个未被划掉的数是3，它是质数，把3的倍数划掉：

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~

接下来应该是5，但是5已经超过 $\sqrt{16}$ 了，所以遍历结束，剩下未被划掉的都是素数：

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~

第四讲 深度优先搜索

一、形式

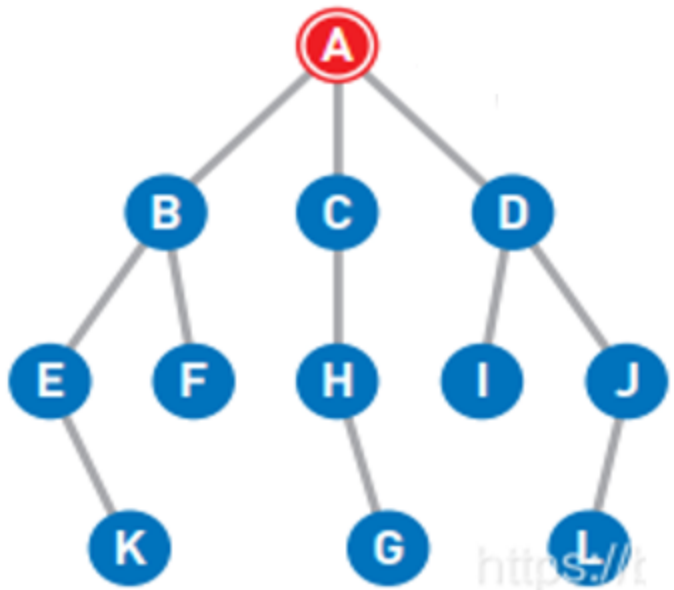
通过递归，实现任意多层循环嵌套，进而实现对问题可能解的枚举

二、效果

搜索的过程形成了一个树状的结构，并且总是沿着优先扩展树的深度的顺序进行枚举

三、优化

搜索算法的复杂度为O(n^m)，如果再当前节点位置已经可以判定不可能是问题的解，则不再向下一层节点扩展，称为剪枝



如果该树为深度优先搜索生成的搜索树
生成节点的顺序为：
A-B-E-K-F-C-H-G-D-I-J-L

第四讲 深度优先搜索

四、深度优先搜索的基本结构

深度优先算法框架[一]

```
int Search(int k)     {
    if  (到目的地||得到目标解)   输出解;
    else
        for (i=1;i<=算符种数;i++)
            if  (满足条件)          {
                保存结果;
                Search(k+1);
                恢复：保存结果之前的状态{回溯一步}
              }
}
```

深度优先搜索算法框架[二]

```
int Search(int k)    {
    for (i=1;i<=算符种数;i++)
       if (满足条件)       {
            保存结果
            if (到目的地||得到目标解) 输出解;
                else Search(k+1);
            恢复：保存结果之前的状态{回溯一步}
        }
}
```

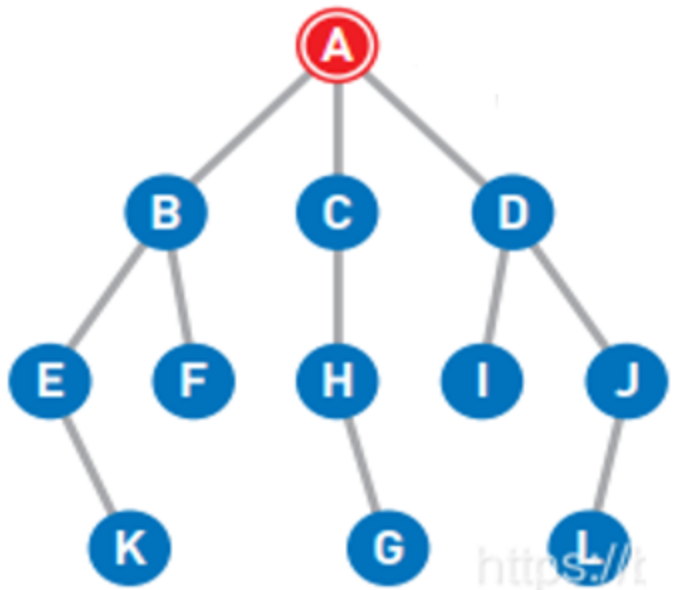
第五讲 广度优先搜索

一、广度优先搜索特点

在搜索的时候如果遇到分叉，不是选择某一个向下搜索（DFS），而是选择多个分支”同时”向下搜索。搜索的过程同样会生成树状的结构，但总是把处于同一层次的所有节点全部生成完成之后，才去扩展下一层次的节点。

二、广度优先搜索应用

适合处理各类求路径最短/最快到达等的问题。



深度优先搜索过程中生成节点的顺序为：
A-B-E-K-F-C-H-G-D-I-J-L

广度优先搜索过程中生成节点的顺序为：
A-B-C-D-E-F-H-I-J-K-G-L

第五讲 广度优先搜索

三、广度优先搜索的基本结构

```
int bfs()
{
   初始化，初始状态存入队列;
   队列首指针head=0; 尾指针tail=1;
   while(head<tail)                    //队列不为空
   {
       指针head后移一位，指向待扩展结点;
       for (int i=1;i<=max;++i)            //max为产生子结点的规则数
       {
           if (子结点符合条件)
           {
               tail指针增1，把新结点存入列尾;
               if (新结点与原已产生结点重复) 删去该结点（取消入队，tail减1）;
               else
                   if (新结点是目标结点) 输出并退出;
           }
       }
   }
}
```

第六讲 动态规划

一、动态规划适用条件

子问题重叠、无后效性、最优子结构性质

二、动态规划要素

三要素：状态、阶段、决策

三、动态规划设计

状态设计（子问题）：一般而言缩小原先问题规模即子问题

状态转移方程：当前问题的结果如何由子问题得到

初始状态：最初的状态

结果表示：最终目标的结果如何用已有状态表示

四、动态规划特点

不重复求解相同的子问题，实现对目标问题的高效求解

试图仅仅解决每个子问题一次，从而减少计算量：一旦某个给定子问题的解已经算出，则将其记忆化存储，以便下次需要同一个子问题解之时直接查表。

骗分导论

一、输出特定内容

在很多比赛中，可能会把样例数据也放到最终的测试数据集中。直接输出样例也许可以骗到分数。

部分题目可能会有无解的情况，并要求在无解情况下输出-1或NO ANSWER，直接输出无解的情况一定会至少通过一组测试数据。或者双方博弈的题目最终的输出结果也许是A WIN！ 或B WIN！，通过对题目分析输出可能性更高的一个。

第 k 小整数(knumber)

描述

现有 n 个正整数，n≤10000，要求出这 n 个正整数中的第 k 个最小整数（相同大小的整数只计算一次）， k≤1000，正整数均小于 30000。

输入

第一行为 n 和 k，第二行开始为 n 个正整数的值，整数间用空格隔开。

输出

第 k 个最小整数的值；若无解，则输出"NO RESULT"。

输入样例 1

```
10 3
1 3 3 7 2 5 1 2 4 6
```

输出样例 1

```
3
```

骗分导论

二、打表输出

数据规模较小、代码编写复杂但是可以比较简单的手算结果。（常用于较代码较复杂的搜索题）

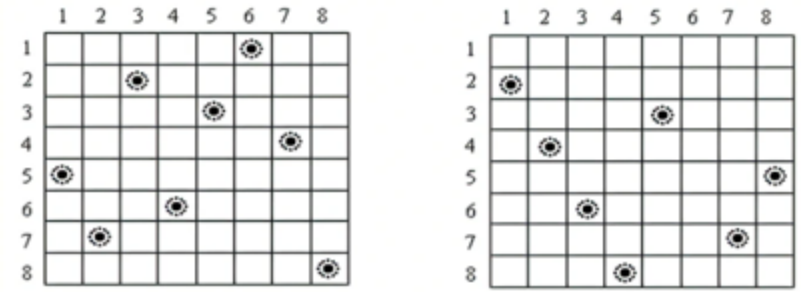
够想到的算法在数据规模较大时超时，通过本地运行代码获得给定范围内全部可能的解之后将较大数据打表输出。

题目求解过程中需要用到特定的数据集（比如一定范围内的所有素数），在代码运行过程中进行求解会严重影响程序运行的效率造成超时，通过打表直接将所需的数据存入数组中。

N 皇后问题(queen.cpp)

描述

在 N*N 的棋盘上放置 N 个皇后（n<50）而彼此不受攻击（即在棋盘的任一行，任一列和任一对角线上 不能放置 2 个皇后），编程求解所有的摆放方法。



八皇后的两组解

输入

输入：n

输出

每行输出一种方案，每种方案顺序输出皇后所在的列号，各个数之间有空格隔开。若无方案，则输出 no solute!

输入样例 1

```
4
```

输出样例 1

```
2 4 1 3
3 1 4 2
```

素数和

描述

给定一个区间范围[m,n]，输出该区间内所有素数的和

输入

本题有多组输入数据

每组数据一行，包含两个数字m,n。

0<=m,n<=10^5

输出

对每一组输入数据，输出[m,n]区间内所有素数的和

输入样例 1

```
2 3
1 10
10 20
```

输出样例 1

```
5
17
60
```

骗分导论

三、寻找边界或特殊情况

题目给出的数据可能存在特殊情况或边界，而当数据恰好是这些特殊情况时，可能会具有某些可以简单计算的性质。如果输入0或1，可能可以不需要复杂的运算直接给出对应的解。或者当输入数据具有某些特定关系时，题目可能可以使用简单的解法来进行计算。

NOIP2005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：$0, 1, \ldots, L$（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。

输入输出格式

输入格式：

第一行有1个正整数$L(1 \le L \le 10^9)$，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[illegible]微信号:noipnoi

$1 \le S \le T \le 10, 1 \le M \le 100$。

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。

当s == t 时，不需要使用DP，仅用贪心算法即可求解。

骗分导论

四、只拿小规模数据分数

题目的测试数据通常情况下都会覆盖给定范围内的小规模、中等规模、较大规模、大规模。

很多题目完整拿到所有分数比较困难，但是只针对数据规模比较小的情况拿下分数会比较容易。常用的求解小规模数据的方法有模拟、搜索等。

模拟通常适用于需要使用高级数据结构的题目。

搜索的本质依然是枚举，可以适用于几乎全部的动态规划、贪心题目，和大部分能看懂但是没有多少思路的题目。（万能搜索、搜索无敌）

矩阵运算（matrix）

【题目背景】

$\text{Softmax}(\frac{\mathbf{Q}\times\mathbf{K}^T}{\sqrt{d}})\times\mathbf{V}$ 是 Transformer 中注意力模块的核心算式，其中 $\mathbf{Q}$、$\mathbf{K}$ 和 $\mathbf{V}$ 均是 n 行 d 列的矩阵，$\mathbf{K}^T$ 表示矩阵 $\mathbf{K}$ 的转置，$\times$ 表示矩阵乘法。

【题目描述】

为了方便计算，顿顿同学将 Softmax 简化为了点乘一个大小为 n 的一维向量 $\mathbf{W}$：

$$(\mathbf{W}\cdot(\mathbf{Q}\times\mathbf{K}^T))\times\mathbf{V}$$

点乘即对应位相乘，记 $\mathbf{W}^{(i)}$ 为向量 $\mathbf{W}$ 的第 i 个元素，即将 $(\mathbf{Q}\times\mathbf{K}^T)$ 第 i 行中的每个元素都与 $\mathbf{W}^{(i)}$ 相乘。

现给出矩阵 $\mathbf{Q}$、$\mathbf{K}$ 和 $\mathbf{V}$ 和向量 $\mathbf{W}$，试计算顿顿按简化的算式计算的结果。

【输入格式】

从标准输入读入数据。

输入的第一行包含空格分隔的两个正整数 n 和 d，表示矩阵的大小。

接下来依次输入矩阵 $\mathbf{Q}$、$\mathbf{K}$ 和 $\mathbf{V}$。每个矩阵输入 n 行，每行包含空格分隔的 d 个整数，其中第 i 行的第 j 个数对应矩阵的第 i 行、第 j 列。

最后一行输入 n 个整数，表示向量 $\mathbf{W}$。

【输出格式】

输出到标准输出。

输出共 n 行，每行包含空格分隔的 d 个整数，表示计算的结果。

【子任务】

70% 的测试数据满足：$n\le 100$ 且 $d\le 10$；输入矩阵、向量中的元素均为整数，且绝对值均不超过 30。

全部的测试数据满足：$n\le 10^4$ 且 $d\le 20$；输入矩阵、向量中的元素均为整数，且绝对值均不超过 1000。

【提示】

请谨慎评估矩阵乘法运算后的数值范围，并使用适当数据类型存储矩阵中的整数。