程序设计实训

南京大学智能科学与技术学院 史桀绮

课程形式

预计11.05最后一课

南雍楼

每周二课上布置本周练习题，周六截止

联系方式

jayceesjq@gmail.com

南雍楼

大作业

- 五人一组完成，结课提交源代码，10.22前提交实验报告和一份6分钟以内的介绍视频，主要解释实现的重要功能，并附上相关的代码片段，展示完整的编译-运行、试玩流程，视频在11.03前可以修改，但需要提交以便分组报告。10.15前麻烦大家把组队名单提交到cms-lab，或发送至老师邮箱。

- 完成的题目选题(可作为参考):
  - 扫雷小游戏
  - 贪吃蛇小游戏
  - 2048
  - 自选

- 电脑端程序，使用C++完成，需要运用面向对象的编程方法

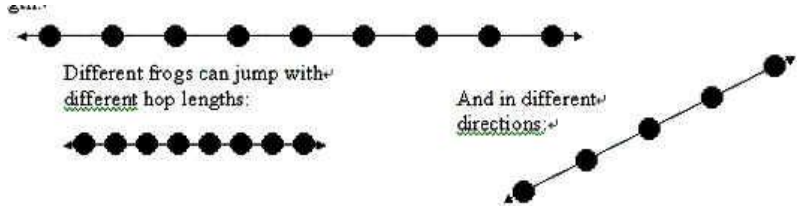
- 最后三节课会在课上播放大家的视频，每组附3分钟提问-答疑时间，并在教学平台上互相提交评分，作为大作业得分的20%

算法分类复习—搜索

穷尽搜索法

恼人的青蛙

在韩国，有一种小的青蛙。每到晚上，这种青蛙会跳越稻田，从而踩踏稻子。农民在早上看到被踩踏的稻子，希望找到造成最大损害的那只青蛙经过的路径。每只青蛙总是沿着一条直线跳越稻田，而且每次跳跃的距离都相同。



如下图所示，稻田里的稻子组成一个栅格，每棵稻子位于一个格点上。而青蛙总是从稻田的一侧跳进稻田，然后沿着某条直线穿越稻田，从另一侧跳出去

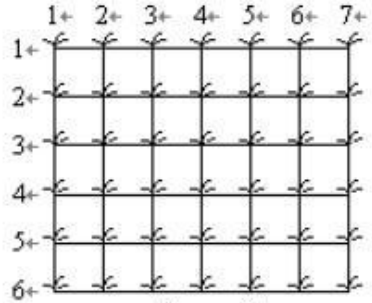


Figure-1

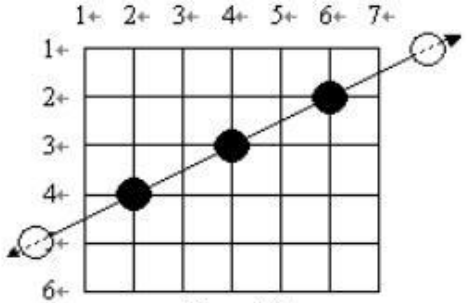


Figure-2

恼人的青蛙

如下图所示，可能会有多只青蛙从稻田穿越。青蛙的每一跳都恰好踩在一棵水稻上，将这棵水稻拍倒。有些水稻可能被多只青蛙踩踏。根据图4，农民能够构造出青蛙穿越稻田时的行走路径。农民只关心那些在穿越稻田时至少踩踏了3棵水稻的青蛙。因此，每条青蛙行走路径上至少包括3棵被踩踏的水稻。而在一条青蛙行走路径的直线上，也可能会有些被踩踏的水稻不属于该行走路径。

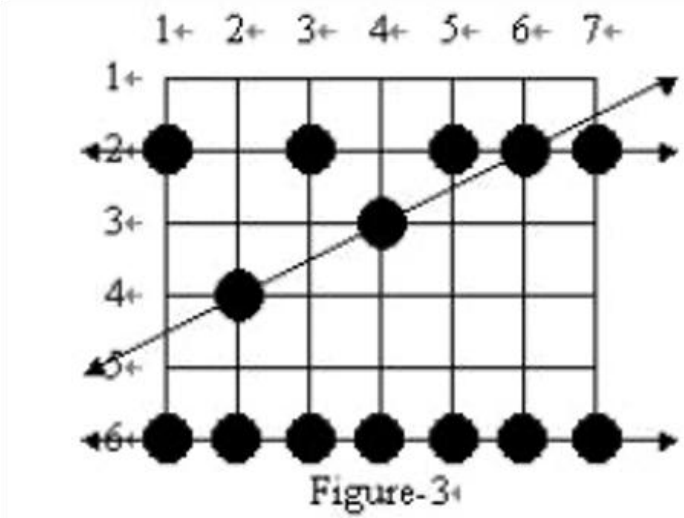


Figure-3

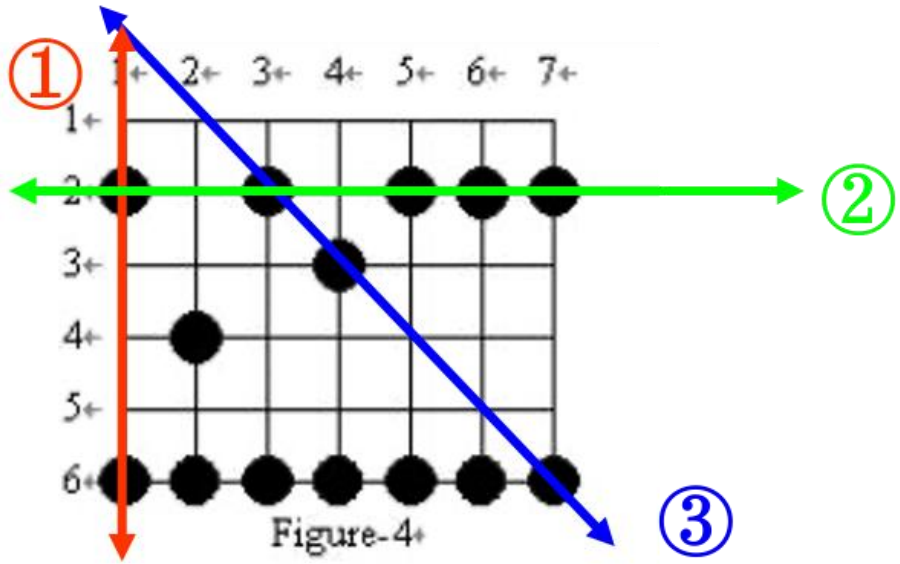


Figure-4

恼人的青蛙

①不是一条行走路径：只有两棵被踩踏的水稻；
②是一条行走路径，但不包括（2，6）上的水稻；
③不是一条行走路径：虽然有3棵被踩踏的水稻，但这三棵水稻之间的距离间隔不相等。
请你写一个程序，确定：在一条青蛙行走路径中，最多有多少颗水稻被踩踏。
例如，图4的答案是7，因为第6行上全部水稻恰好构成一条青蛙行走路径。

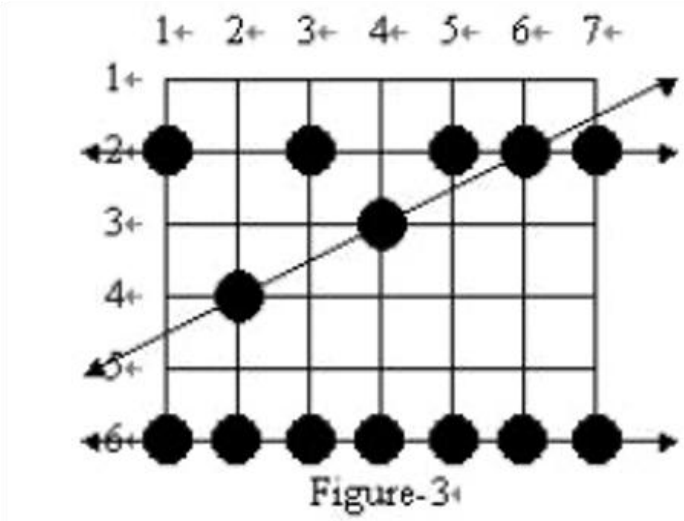


Figure-3

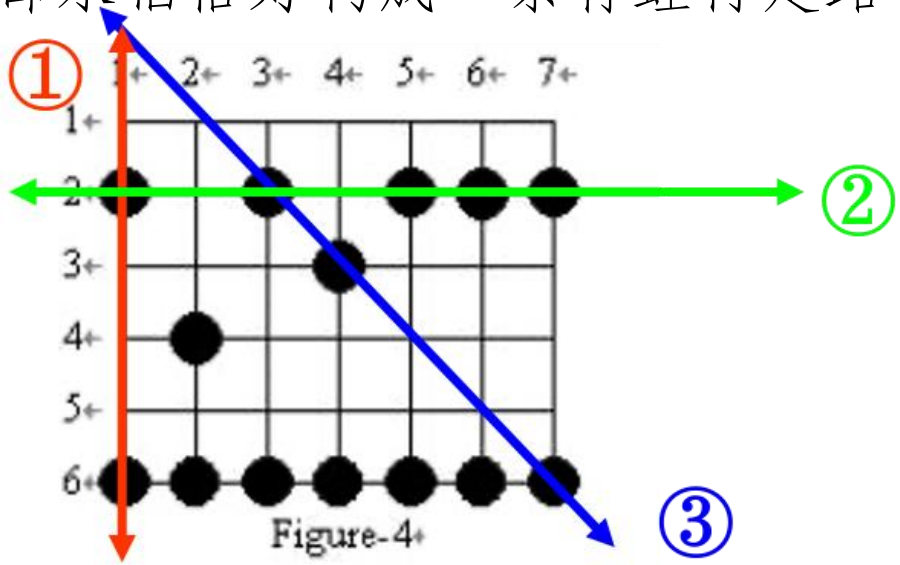


Figure-4

恼人的青蛙

输入

第一行是两个整数R、C，分别表示稻田中水稻的行数和列数，1≤R、C≤5000。
第二行是一个整数N，表示被踩踏的水稻数量， 3≤N≤5000。
在剩下的N行中，每行有两个整数，分别是一颗被踩踏水稻的行号(1~R)和列号(1~C)，两个整数用一个空格隔开。而且，每棵被踩踏水稻只被列出一次。

输出

一个整数。
如果在稻田中存在青蛙行走路径，则输出包含最多水稻的青蛙行走路径中的水稻数量，否则输出0。

恼人的青蛙

需要获得最长路径，显然需要枚举出所有可能的路径。

恼人的青蛙

需要获得最长路径，显然需要枚举出所有可能的路径。

一条直线由两个点决定，因此，简化为枚举每个可能路径的前两个点，并判断是否能组成一条完整的路径，以及其长度。

恼人的青蛙

需要获得最长路径，显然需要枚举出所有可能的路径。

一条直线由两个点决定，因此，简化为枚举每个可能路径的前两个点，并判断是否能组成一条完整的路径，以及其长度。剪枝操作包括：

1. 判断前两点的合法性（能否从外面一步跳到第一个点）

2. 对所有点进行排序，步长小的优先考虑。

3. 每一步间距太长，最多也跳不到当前的最大值的方法可以直接否定。

穷尽搜索法(就是穷举法)。

恼人的青蛙

```
void count(int start, int next){
    int dx = p[next].x - p[start].x;
    int dy = p[next].y - p[start].y;
    if(!outside(p[start].x - dx, p[start].y - dy)) return;
    if(outside(p[start].x + ans * dx, p[start].y + ans * dy)) return;
    int k = 2;
    int x1 = p[next].x + dx;
    int y1 = p[next].y + dy;
    while(!outside(x1, y1) && flags[x1][y1]){
        k++; x1 += dx; y1 += dy;
    }
    if(outside(x1, y1) && k > ans) ans = k;
}
```

木棒

乔治拿来一组等长的木棒，将它们随机地裁断，使得每一节木棍的长度都不超过50个长度单位。然后他又想把这些木棍恢复到为裁截前的状态，但忘记了初始时有多少木棒以及木棒的初始长度。请你设计一个程序，帮助乔治计算木棒的可能最小长度。每一节木棍的长度都用大于零的整数表示。

输入

输入包含多组数据，每组数据包括两行。第一行是一个不超过64的整数，表示砍断之后共有多少节木棍。第二行是截断以后，所得到的各节木棍的长度。在最后一组数据之后，是一个零。

输出

为每组数据，分别输出原始木棒的可能最小长度，每组数据占一行。

木棒

样例输入

9

5 2 1 5 2 1 5 2 1

4

1 2 3 4

0

样例输出

6

5

木棒

问题抽象：有一些数，把这些数组合到一起，让每组数据总和相等，求总和的最小值。

木棒

问题抽象：有一些数，把这些数组合到一起，让每组数据总和相等，求总和的最小值。

将问题转换为：对每一个固定的数S，求是否有一种方法将数字进行组合，使得每一组的和都等于S。

木棒

问题抽象：有一些数，把这些数组合到一起，让每组数据总和相等，求总和的最小值。

将问题转换为：对每一个固定的数S，求是否有一种方法将数字进行组合，使得每一组的和都等于S。

枚举S，对每个S搜索组合方法。

木棒

枚举范围:

1. 最长的一根木棍的长度

2. 所有木棍长度的和。

剪枝方法:

1.枚举范围进一步缩小为所有木棒长度和的因子。
2. 从小到大枚举长度，这样第一个长度就是最小的。
3. 短的木棍可以更灵活组合，所以dfs前对输入的所有木棍按长度从大到小排序，先拼长棍，这样短木棍可以更加灵活地拼接 。
4. dfs时二分查找需要的剩下的木棒

木棒

```
void dfs(int tot,int last,int rest){              //tot代表现在有几组
    if(rest==0){                                  //当前的木棒拼成了想要的长度
        if(tot==group){                           //所有木棒都拼成了
            cout<<sum;
            flag =True;
            return;}
        for(int k=1;k<=cnt;k++){                  //找一个还没用的最长的木棒
            if(!vis[k])break;
        }
        vis[k]=1;
        dfs(tot+1,k,target-a[k]);
        vis[k]=0;
    }
```

木棒

```
    int l=last+1,r=cnt,mid;
    while(l<r){
        mid=(l+r)/2;
        if(a[mid]<=rest) r=mid;
        else l=mid+1;}          //二分查找第一个小于等于rest的位置
    for(int i=l;i<=cnt;i++){ //注意木棒是从大到小排序
        if(vis[i])continue;
        vis[i]=1;
        dfs(tot,i,rest-a[i]);
        vis[i]=0;                   //这个位置还可以剪枝，跳过部分i
    }
}
```

深度/广度优先搜索

抓住那头牛

农夫知道一头牛的位置，想要抓住它。农夫和牛都位于数轴上，农夫起始位于点N(0<=N<=100000)，牛位于点K(0<=K<=100000)。农夫有两种移动方式：

1、从X移动到X-1或X+1，每次移动花费一分钟
2、从X移动到2*X，每次移动花费一分钟

假设牛没有意识到农夫的行动，站在原地不动。农夫最少要花多少时间才能抓住牛？

输入
两个整数，N和K
输出
一个整数，农夫抓到牛所要花费的最小分钟数

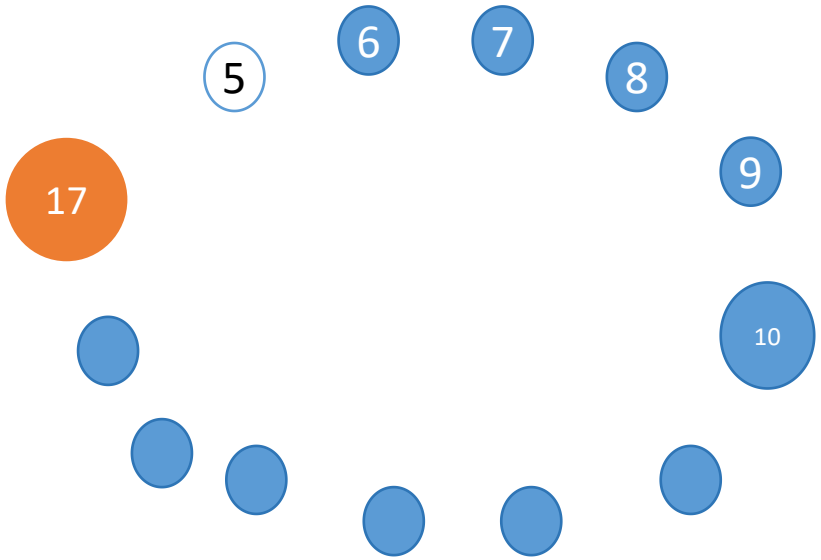
抓住那头牛

样例输入
5 17

样例输出
4

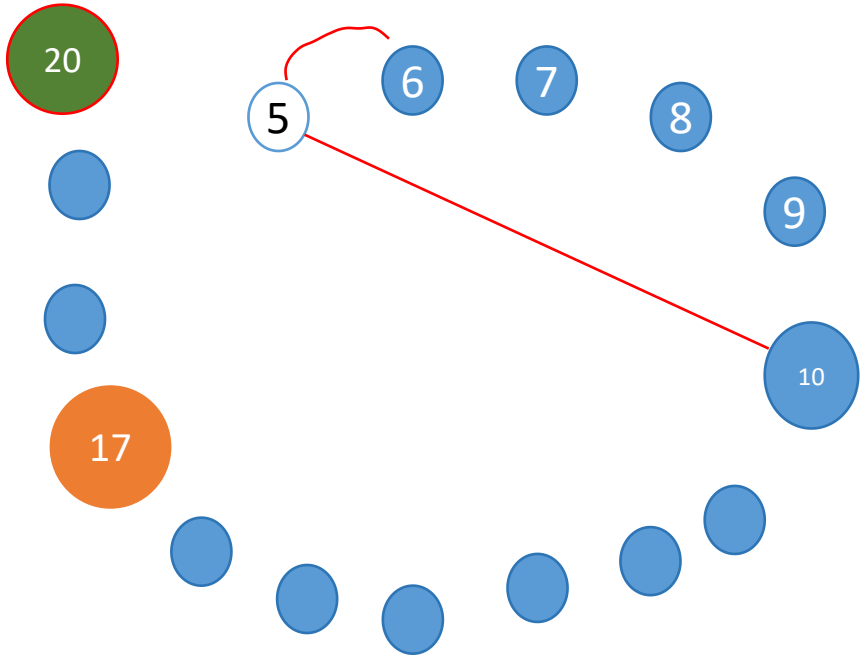
抓住那头牛

举一个简单的例子:



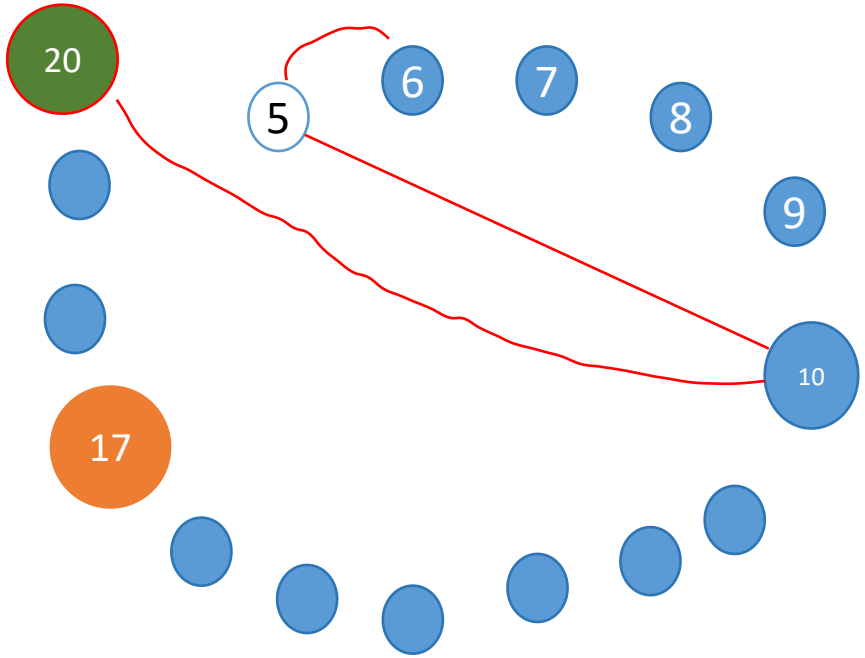
抓住那头牛

举一个简单的例子：



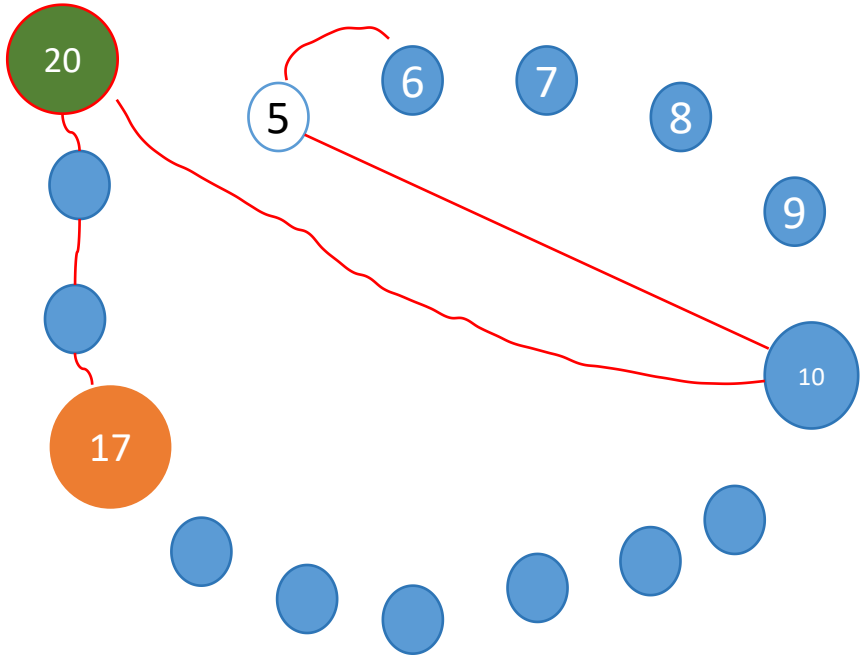
抓住那头牛

举一个简单的例子：



抓住那头牛

举一个简单的例子：



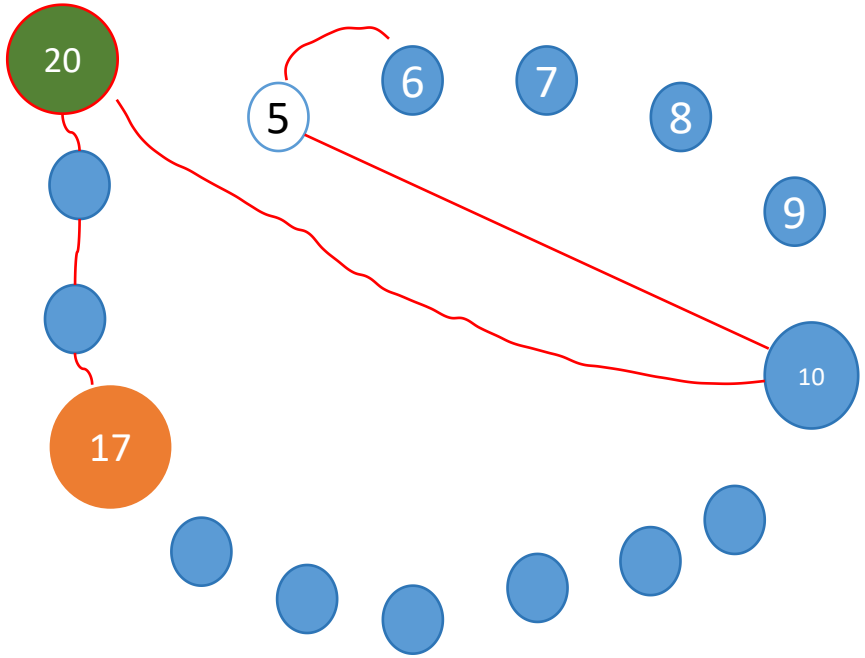
策略一：

每次只有三种走路方法，+1, -1, 或者*2，随机选一个往前走，直到走到目标位置为止。

深度优先搜索。计算过程中需要保存的内容比较少，但是由于方向随机，需要进行剪枝节省时间，并且不保证能够随机到最优解。

抓住那头牛

举一个简单的例子：



策略二：

给节点分层，从起点开始，一步能够到达的点为第一层，两步到达的是第二层……

按照层次顺序从小到大扩展节点，在拜访完所有低层次节点后才摆放高层次的，保证第一次遇到目标点时一定是最短路。

广度优先搜索。一定能够找到最优解，但需要较大的空间用于存储。
5 -> (6, 10) -> (7, 9, 11, 20) -> ... ->17

单词序列

给出两个单词（开始单词和结束单词）以及一个词典。找出从开始单词转换到结束单词，所需要的最短转换序列。转换的规则如下：

1、每次只能改变一个字母

2、转换过程中出现的单词(除开始单词和结束单词)必须存在于词典中

例如：开始单词为：hit，结束单词为：cog，词典为：[hot,dot,dog,lot,log,mot]，那么一种可能的最短变换是： hit -> hot -> dot -> dog -> cog,所以返回的结果是序列的长度5。

注意：

1、如果不能找到这种变换，则输出0；

2、词典中所有单词长度一样；

3、所有的单词都由小写字母构成；

4、开始单词和结束单词可以不在词典中。

单词序列

输入

共两行，第一行为开始单词和结束单词（两个单词不同），以空格分开。第二行为若干的单词（各不相同），以空格分隔开来，表示词典。单词长度不超过5,单词个数不超过30。

输出

输出转换序列的长度。

样例输入

hit cog
hot dot dog lot log

样例输出

5

单词序列

变化的最短序列，明显的广度优先搜索题目。注意本体和动态规划题目的区别。

1. 设置used数组，代表已经访问过某个单词。

2. 设函数CompareStr(string s1, string s2)判断s1与s2两个等长的字符串不同的字符个数。如果两字符串只有1个字符不同，那么经过一次变换就能从s1变为s2。

3. 每一步保存当前单词，以及从初始单词转为当前单词的序列长。

单词序列

```
que.push(Node(st, 1));
while(!que.empty()){
        Node u = que.front();
        que.pop();
        if(CompareStr(u.str, target) == 1)
                return u.len+1;
        for(int i = 0; i < num_str; ++i){
                if(vis[i] == false && CompareStr(u.str, w[i]) == 1){
                        vis[i] = true;
                        que.push(Node(w[i], u.len+1));
                }
        }
}
```

迷宫问题

定义一个二维数组，它表示一个迷宫，其中的1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左上角到右下角的最短路线。

输入

一个5 × 5的二维数组，表示一个迷宫。数据保证有唯一解。

输出

左上角到右下角的最短路径，格式如样例所示。

迷宫问题

样例输入

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

样例输出

```
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
(2, 4)
(3, 4)
(4, 4)
```

解题思路

最典型的广度优先搜索题目。

先将起始位置入队列，每次从队列拿出一个元素，扩展其相邻的4 个元素，判重后加入队列，直到队头元素为终点为止。

队列里的元素记录了指向父节点（上一步）的指针。队列元素：

```
struct{
    int r,c
    int f; 父节点在队列中的下标
};
```

城堡

有一座城堡被划分为m*n（m<=50，n<=50）个方形模块。每个这样的模块可以有零到四面墙。请编写一个计算程序，计算

1.这座城堡有多少个房间

2.最大的房间有多大

输入

第一行包含南北方向上的模块数量和东西方向上的模块数量。在接下来的几行中，每个模块都由一个数字（0<=p<=15）描述。这个数字是1（=西面的墙）、2（=北面的墙），4（=东面的墙）和8（=南面的墙）的总和。内壁被定义了两次；模块1,1中南面的墙在模块2.1中也表示为北面的墙。城堡至少有两个房间。

输出

两个数字，首先是房间的数量，然后是最大房间的面积（以模块计算）。

城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

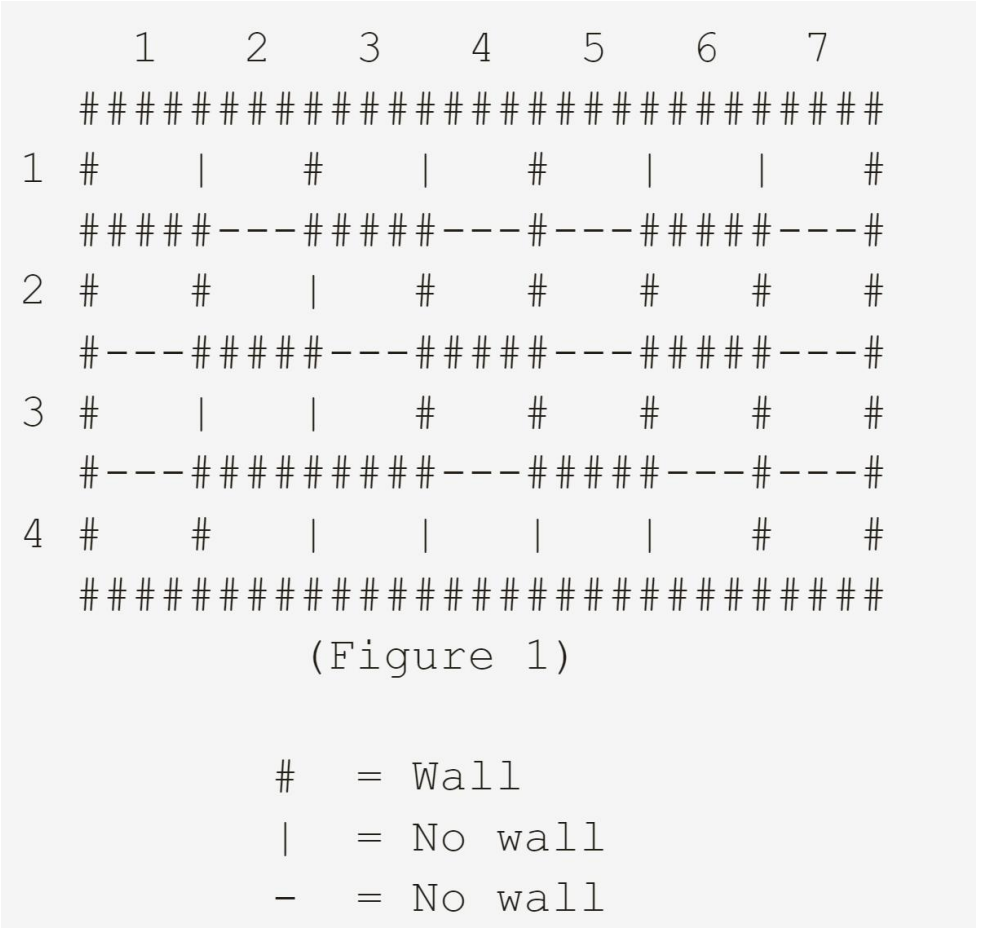
1 10 12 7 13 7 5

13 11 10 8 10 12 13

样例输出

5

9



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

1 10 12 7 13 7 5

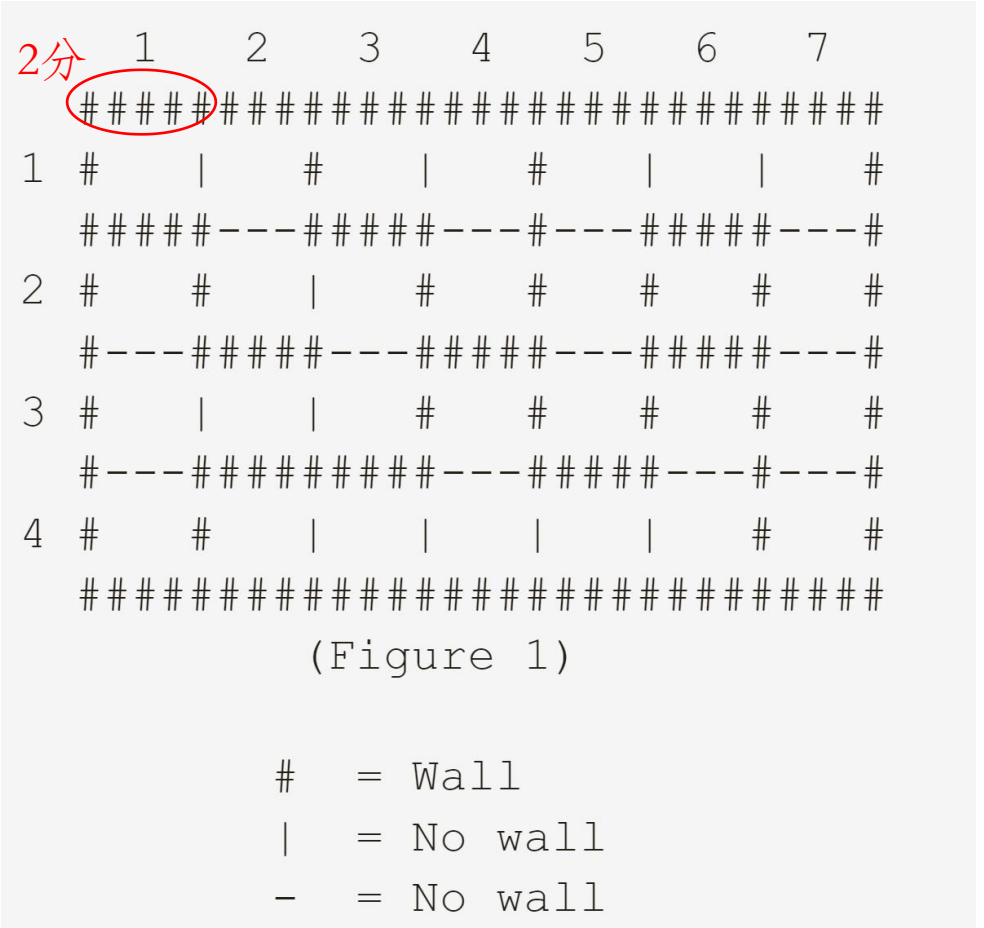
13 11 10 8 10 12 13

样例输出

5

9

上面的墙，2分



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

1 10 12 7 13 7 5

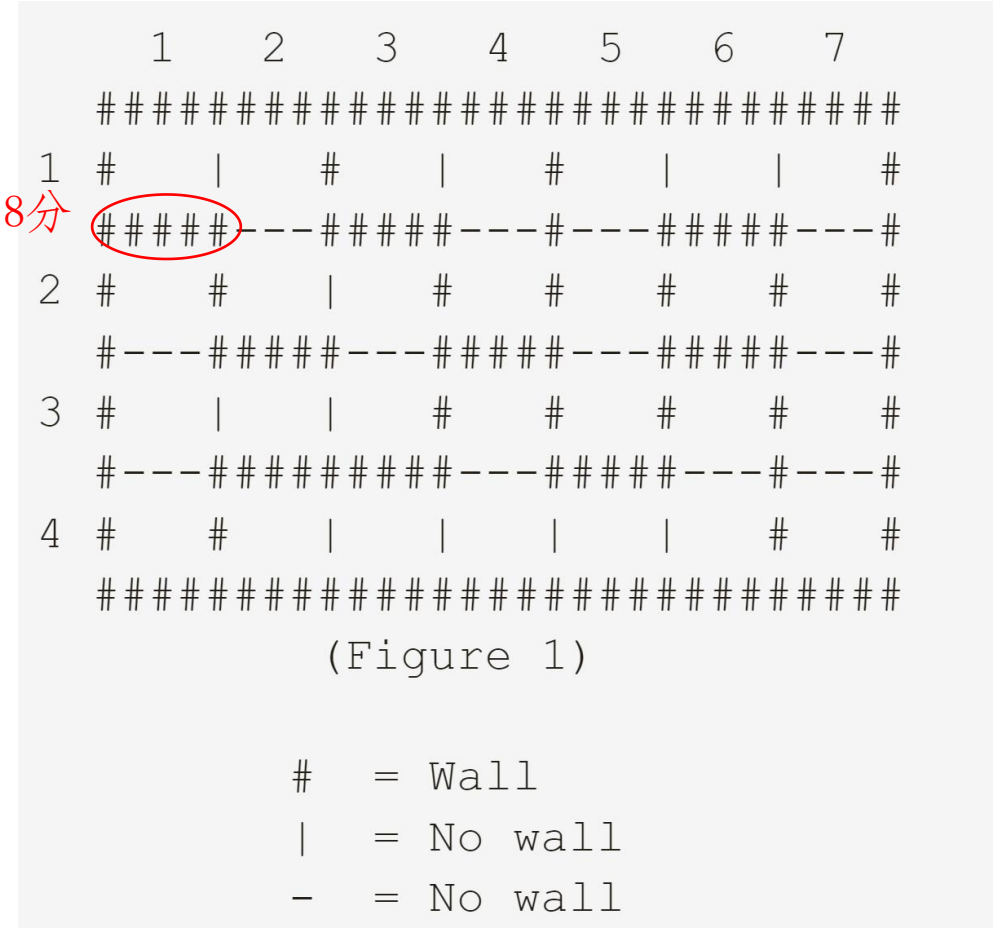
13 11 10 8 10 12 13

样例输出

5

9

下面的墙，8分



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

1 10 12 7 13 7 5

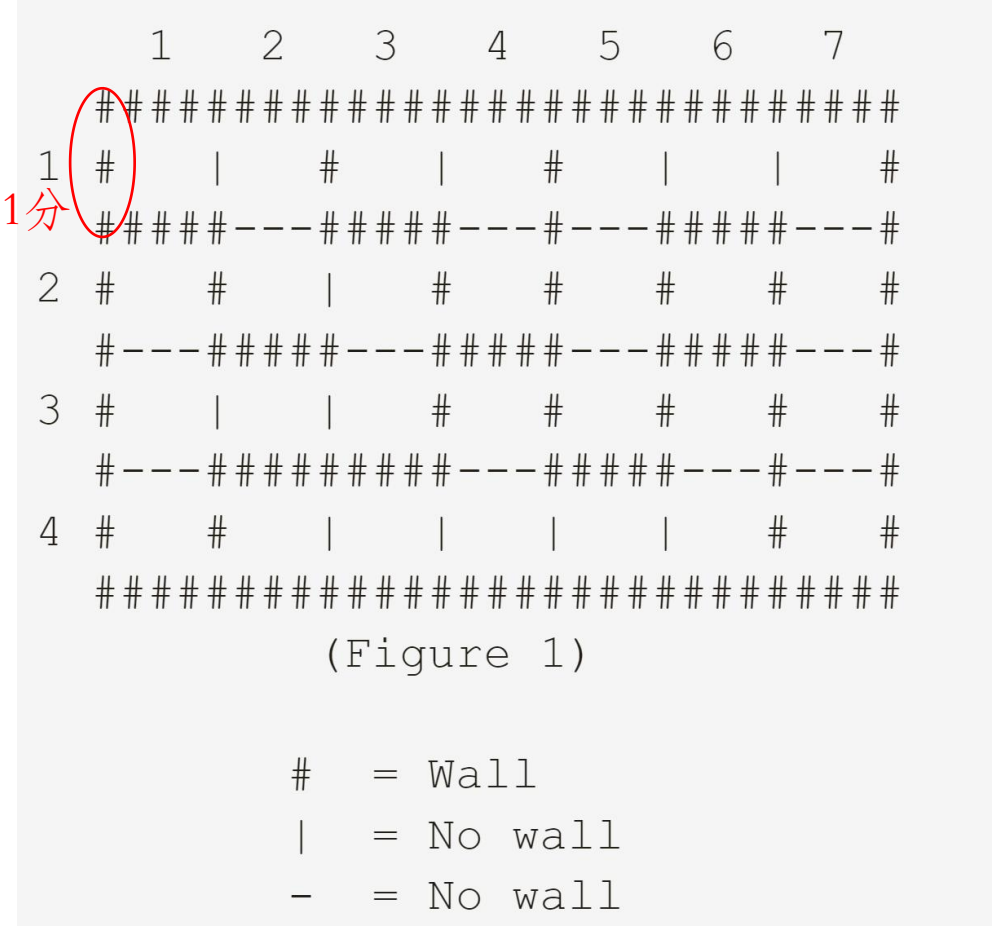
13 11 10 8 10 12 13

样例输出

5

9

左面的墙，1分



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

1 10 12 7 13 7 5

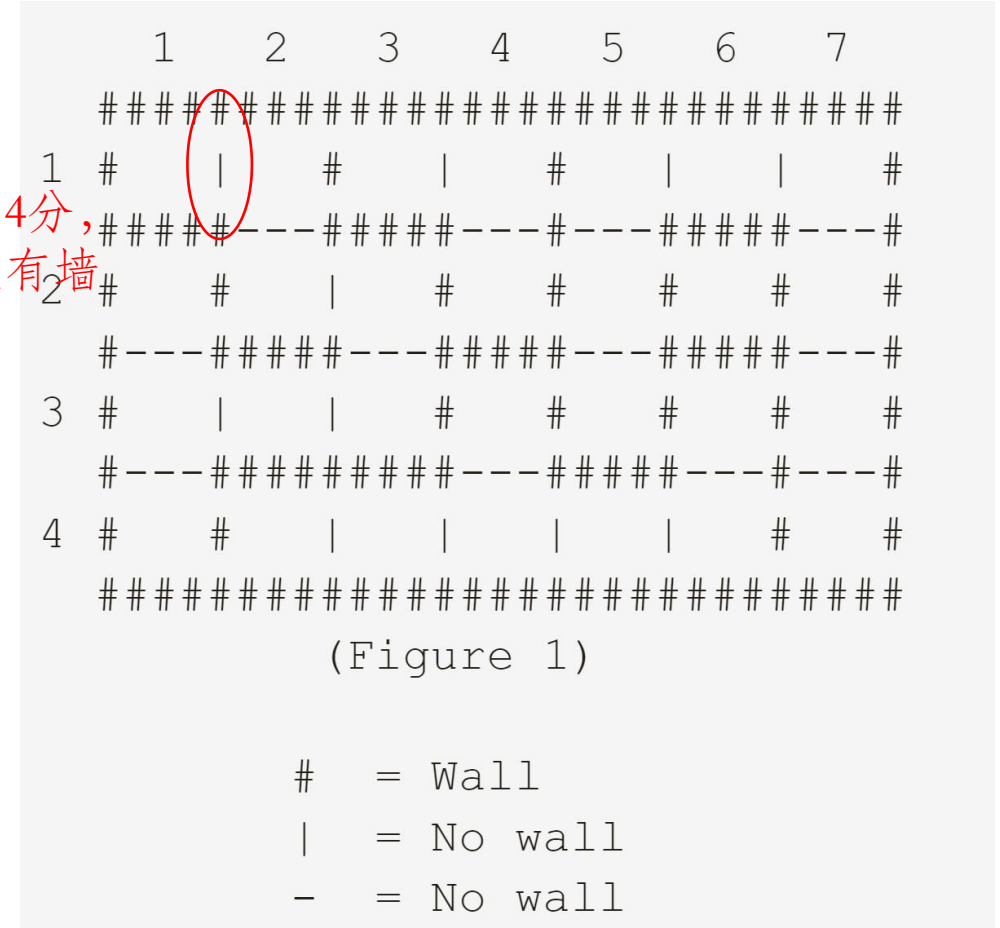
13 11 10 8 10 12 13

样例输出

5

9

右面的墙，4分，
但是这里没有墙



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

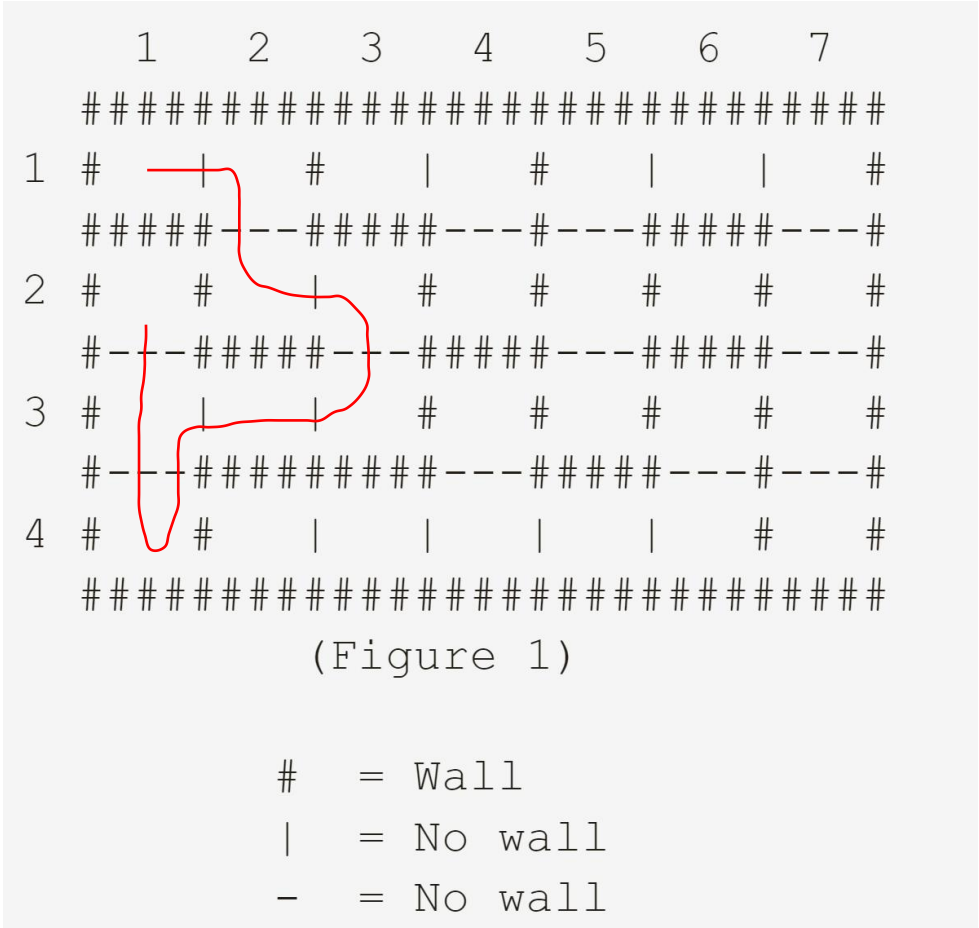
1 10 12 7 13 7 5

13 11 10 8 10 12 13

样例输出

5

9



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

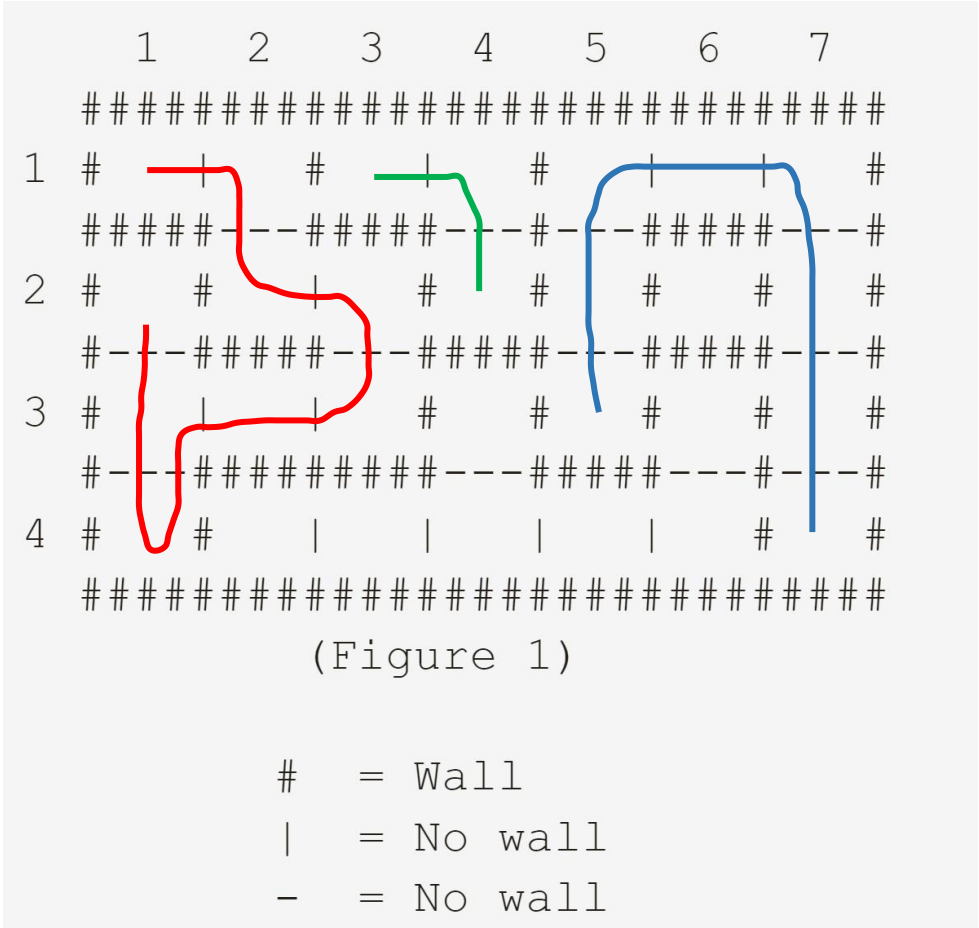
1 10 12 7 13 7 5

13 11 10 8 10 12 13

样例输出

5

9



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

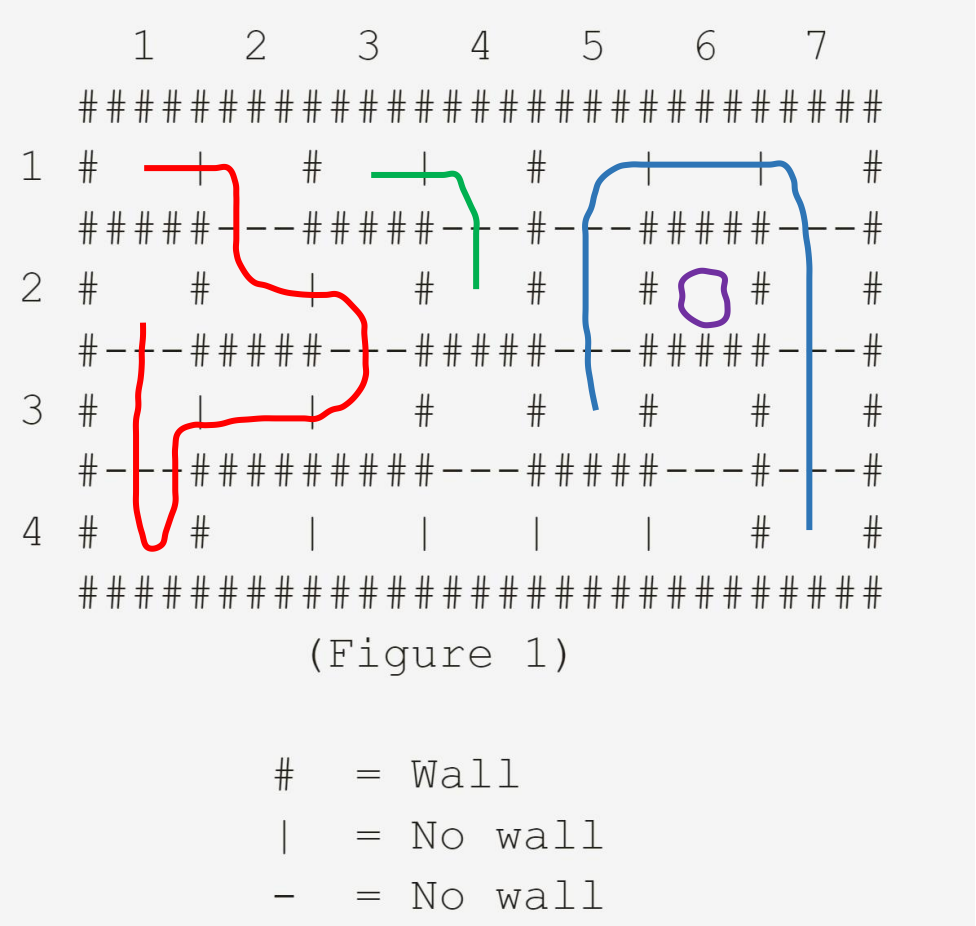
1 10 12 7 13 7 5

13 11 10 8 10 12 13

样例输出

5

9



城堡

样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

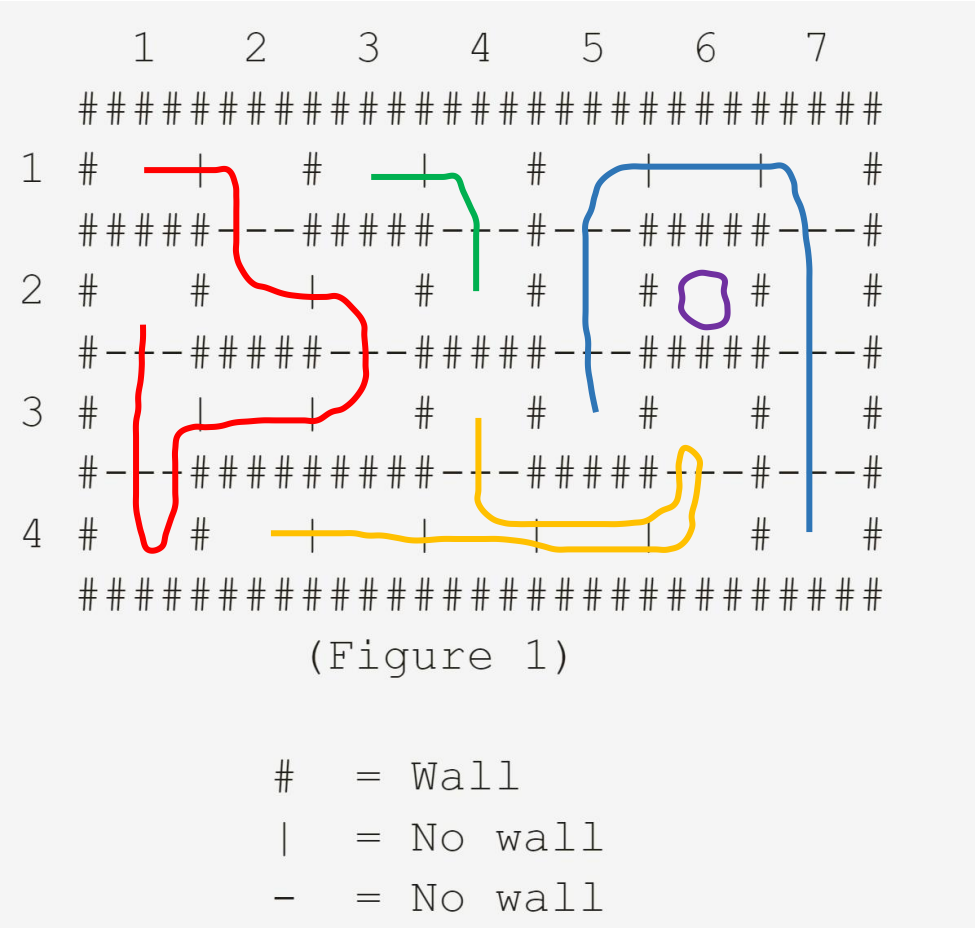
1 10 12 7 13 7 5

13 11 10 8 10 12 13

样例输出

5

9



解题思路

想要达到目的，需要知道哪些地方有墙，哪些地方没有。那么问题就是如何将一个方块上的数分离开来，从而得知哪些地方有墙呢?

即如何将输入的数字转换为二维的方格?

解题思路

想要达到目的，需要知道哪些地方有墙，哪些地方没有。那么问题就是如何将一个方块上的数分离开来，从而得知哪些地方有墙呢?

即如何将输入的数字转换为二维的方格?

墙的值分别为1,2,4,8，等于$2^0, 2^1, 2^2, 2^3$。因此，将某一个方块转为二进制，判断($2^0, 2^1, 2^2, 2^3$)对应位是否是1，就能够得到某一面墙是否存在。

```
for(int a = m; a > 0; a /= 2)
    wall[i] = a%2;
```

解题思路

获得墙是否连通后，转化为求解连通块问题。

1. 遍历整个地图，从每个位置分别开始一次搜索。如果成功进行一次搜索，那么存在一个连通块，也就是房间。

2. 每搜索到一个位置，根据该位置的值来获得该位置周围墙的信息。

3. 如果某一方向上没有墙，而且该方向上的下一个位置在地图内且没访问过，那么就从下一个位置继续进行搜索。搜索时，要同时统计该连通块中格子的个数。在搜索结束后，更新连通块格子的最大值。

4. 最后输出连通块个数，即连通块中格子数的最大值。

解题思路：深度优先搜索

```
int dir[4][2] = {{0,-1},{-1,0},{0,1},{1,0}};        //0西1北2东3南
int max_area = 0;
void dfs(int curr_x, int curr_y){
    int val = m[sx][sy];
    for(int i = 0; i < 4; i++){
        int next_x = curr_x+ dir[i][0], next= curr_y+ dir[i][1];      //向第i个方向走一步
        if(next_x>= 1 && next_x<= m && next_y >= 1 && next_y <= n
            && val  % 2 == 0 && used[next_x][next_y] == false){
           used[next_x][next_y] = true; max_area++;
           dfs(next_x, next_y);
        }
        val  /= 2;
    }
```

解题思路：广度优先搜索

```
int bfs(int curr_x, int curr_y){
    int max_area= 1;
    queue<Node> que;
    used[curr_x][curr_y] = true;
    que.push(Node(curr_x, curr_y));
    while(!que.empty()){
        Node u = que.front();
        que.pop();
        int val = m[u.x][u.y];//mp[u.x][u.y]为题解中的m值
        for(int i = 0; i < 4; i++){
            int next_x = u.x + dir[i][0], next_y = u.y + dir[i][1];
```

拯救公主

多灾多难的公主又被大魔王抓走啦！国王派遣了第一勇士阿福去拯救她。

身为超级厉害的术士，同时也是阿福的好伙伴，你决定祝他一臂之力。你为阿福提供了一张大魔王根据地的地图，上面标记了阿福和公主所在的位置，以及一些不能够踏入的禁区。你还贴心地为阿福制造了一些传送门，通过一个传送门可以瞬间转移到任意一个传送门，当然阿福也可以选择不通过传送门瞬移。传送门的位置也被标记在了地图上。此外，你还查探到公主所在的地方被设下了结界，需要集齐K种宝石才能打开。当然，你在地图上也标记出了不同宝石所在的位置。

你希望阿福能够带着公主早日凯旋。于是在阿福出发之前，你还需要为阿福计算出他最快救出公主的时间。

拯救公主

地图用一个R×C的字符矩阵来表示。字符S表示阿福所在的位置，字符E表示公主所在的位置，字符#表示不能踏入的禁区，字符$表示传送门，字符.表示该位置安全，数字字符0至4表示了宝石的类型。阿福每次可以从当前的位置走到他上下左右四个方向上的任意一个位置，但不能走出地图边界。阿福每走一步需要花费1个单位时间，从一个传送门到达另一个传送门不需要花费时间。当阿福走到宝石所在的位置时，就视为得到了该宝石，不需要花费额外时间。

拯救公主

输入

第一行是一个正整数T（1 <= T <= 10），表示一共有T组数据。

每一组数据的第一行包含了三个用空格分开的正整数R、C（2 <= R, C <= 200）和K，表示地图是一个R×C的矩阵，而阿福需要集齐K种宝石才能够打开拘禁公主的结界。

接下来的R行描述了地图的具体内容，每一行包含了C个字符。字符含义如题目描述中所述。保证有且仅有一个S和E。$的数量不超过10个。宝石的类型在数字0至4范围内，即不会超过5种宝石。

输出

对于每一组数据，输出阿福救出公主所花费的最少单位时间。若阿福无法救出公主，则输出“oop!”（只输出引号里面的内容，不输出引号）。每组数据的输出结果占一行。

拯救公主

样例输入

1

7 8 2

.....

..S..#0.

.##..1..

.0#.....

...1#...

...##E..

...1....

.	.	.	.	.	.	.	.
.	.	阿福	.	.	禁区	0	.
.	禁区	禁区	.	.	1	.	.
.	0	禁区	.	.	.	.	.
.	.	.	1	禁区	.	.	.
.	.	.	禁区	禁区	公主	.	.
.	.	.	1	.	.	.	.

样例输出

11

拯救公主

升级版本的迷宫问题，还是广度优先搜索，但是如何修改模版?

拯救公主

升级版本的迷宫问题，还是广度优先搜索，但是如何修改模版?

1. 多组数据。避免定义全局变量，全都在for循环内定义。

2. 设置额外的列表存储传送门。在迷宫中加入了传送门，需要在搜索时考虑传送和不传送的情况。

3. 修改剪枝方法。used[i][j]变为三维数组，记录宝石的种类。不可以使用宝石数量，因为可能同一种有多个宝石。考虑状态压缩，用二进制表示获得的宝石种类。

4. 添加棋盘属性。除了是否联通外，需要添加是否有宝石以及宝石种类。

5. 添加自身属性。包括结界是否存在，已经获得了多少种宝石。

拯救公主

升级版本的迷宫问题，还是广度优先搜索，但是如何修改模版?

修改广度优先搜索的搜索策略。

1) 判断是否超界或是否为墙

2) 判断该位置是不是宝石。如果是，则修改获得宝石的总类别，用walk三维数组判断是否走了重复的路。若不是，则判断得到的宝石数量是否达标。

3) 判断是不是传送门。如果是传送门，则遍历传送门列表，将除它本身的传送门存入队列。

4) 如果找到终点，则判断结界是否为false。若是，则输出，作找到路径的标记并且结束数据；若结界为true，不要退出本次循环，将终点当做道路进行下一次移动。

拯救公主

```
struct Path{
    int x,y,tot;                          //坐标，步数
    bool jewel[7],enchantment;      //找到的宝石类别，结界
};

while(!que.empty()){
            for(int i=0;i<4;i++){
                start=que.front();
                start.tot++; start.x+=dir[i][0]; start.y+=dir[i][1];
                if(start.x<0 || start.x>=r || start.y<0 || start.y>=c || maze[start.x][start.y]=='#')
continue;
                if('0'<=maze[start.x][start.y] && maze[start.x][start.y]<='4') //如果为宝石
                    start.jewel[maze[start.x][start.y]-'0']=true;
```

拯救公主

```
int get_jewel=0,ss=0;
for(int i=0;i<5;i++)
    if(start.jewel[i])
        get_jewel+=pow(2,i),ss++;          //二进制表示找到的宝石种类
if(used[start.x][start.y][get_jewel]) continue;//到达过该位置，无更多的宝石
used[start.x][start.y][get_jewel]=true;
if(ss>=needkind) start.enchantment=false;
if(maze[start.x][start.y]=='$'){
    door=start;
    for(int i=0;i<door_num;i++) //遍历所有传送门
        if(door_list[i][0]!=start.x || door_list[i][1]!=start.y){
            door.x=door_list[i][0];door.y=door_list[i][1];
            que.push(door);
```

拯救公主

```
                }
            }
            if(maze[start.x][start.y]=='E' && !start.enchantment){
                //如果是找到终点并且没有结界
                printf("%d\n",start.tot);
                finish=true;
            }
            if(finish) break;
            que.push(start); }
        if(finish) break;
        que.pop();}
    if(!finish) printf("oop!\n"); //没有找到
}
```

寻找NEMO

Nemo 是个顽皮的小孩，一天他一个人跑到深海里去玩，可是他迷路了，于是他向父亲 Marlin 发送了求救信号。

通过查找地图 Marlin 发现那片海像一个有着墙和门的迷宫，所有的墙都是平行于 X 轴或 Y 轴的，墙的厚度可以忽略不计。所有的门都开在墙上并且长度为1。

Marlin 只能穿过有门的墙，因为穿过墙是有危险的 (门旁可能会藏有巨毒的水母)。

Marlin 想穿过尽量少的门找到 Nemo。
图-1 显示了一个迷宫的样例及 Marlin 找到 Nemo的路线。

寻找NEMO

我们假设 Marlin 的初始位置在 (0, 0). 给定 Nemo 的位置和墙及门的位置情况,请你写一个程序计算 Marlin 要找到 Nemo最少要穿过多少道门.

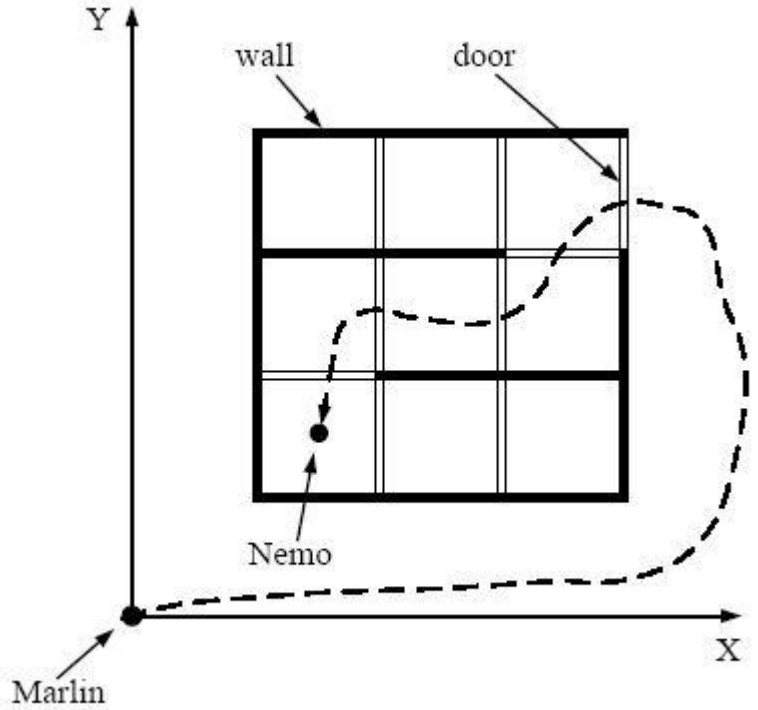


Figure-1. Labyrinth and Path

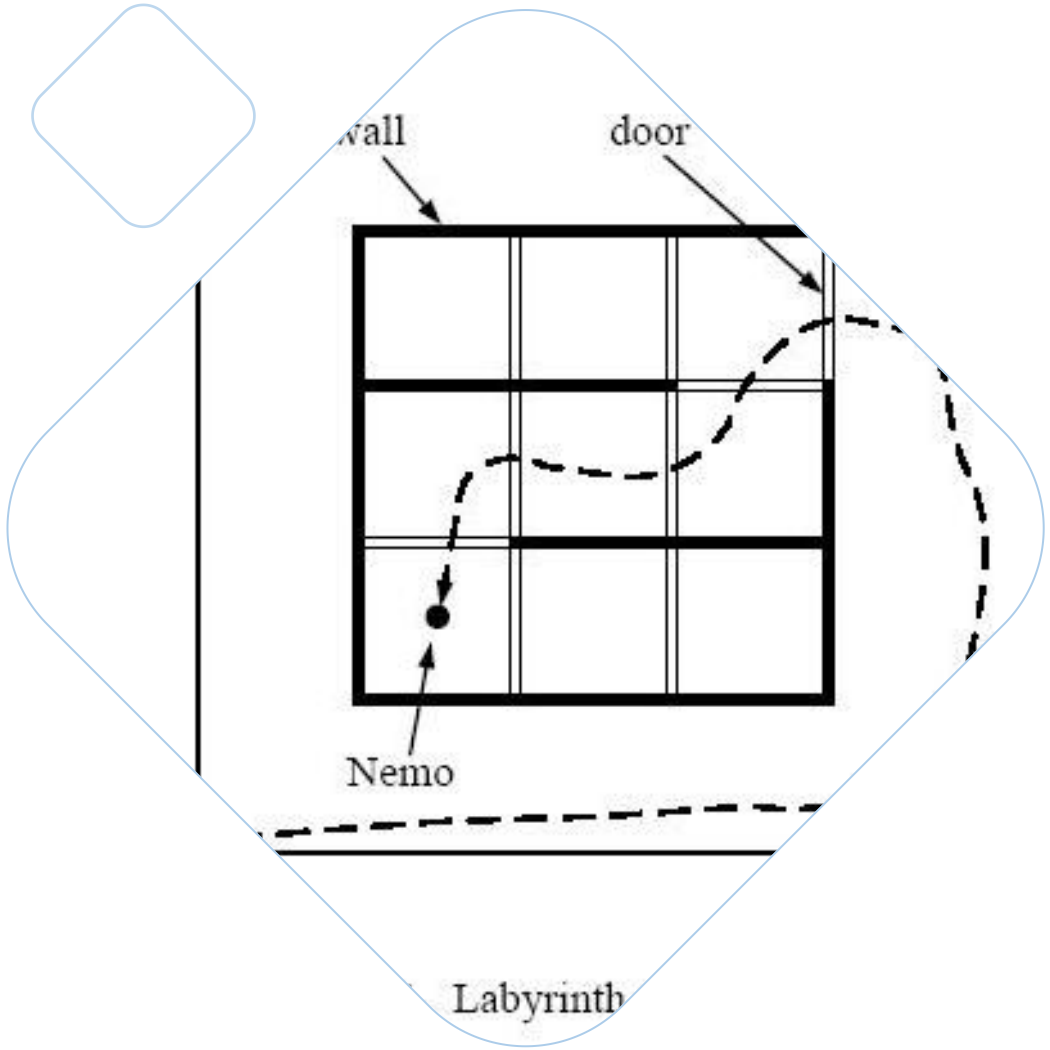
寻找NEMO

Presentations are communication tools that can be used as demonstrations.

Presentations are communication tools that can be used as demonstrations.

Presentations are communication tools that can be used as demonstrations.

Presentations are communication tools that can be used as demonstrations.



寻找NEMO

输入

输入有多组测试数据. 每组测试数据以两个非零整数 M 和 N 开始. M 表示迷宫中墙的数目, N 表示门的数目. 接下来有 M 行, 每行包含四个整数描述一堵墙 (x y d t)。(x, y) 表示墙的左下角, d 是墙的方向，0 表示它与 X轴平行, 1表示它与 Y轴平行, t 表示墙的长度。墙的两个顶点坐标在[1,199]。
接下来有 N 行,用来描述门的情况(x y d)。x, y, d 与门的描述含义相同，因为门的长度是 1, t 被省略了。
每组测试数据的最后一行包含两个正的浮点数 (f1, f2) 给出了 Nemo 的位置。它不在墙和门上。

输出

对于每组测试数据，输出一行，该行包含 Marlin 找到 Nemo需要穿过的最少的门数。如果他不可能找到 Nemo, 输出 -1.

寻找NEMO

样例输入

```
8 9
1 1 1 3
2 1 1 3
3 1 1 3
4 1 1 3
1 1 0 3
1 2 0 3
1 3 0 3
1 4 0 3
2 1 1
2 2 1
2 3 1
3 1 1
3 2 1
3 3 1
1 2 0
3 3 0
4 3 1
1.5 1.5
4 0
1 1 0 1
1 1 1 1
2 1 1 1
1 2 0 1
1.5 1.7
-1 -1
```

样例输出

5

-1

解题思路

典型的最短路问题，常用广度优先搜索。

1. 输入为线段形式，转换为二维数组wall[x][y][4]。

2. 坐标转化。将坐标由以角点为中心，转换为以方格为单位。

解题思路

典型的最短路问题，常用广度优先搜索。

1. 输入为线段形式，转换为二维数组wall[x][y][4]。

2. 坐标转化。将坐标由以角点为中心，转换为以方格为单位。

一道墙1 1 1 3，左下角在(1,1)，与Y轴平行，长度为3，则其位于(1,1)到(1,4)的位置。对于方格来说，可以直接对应到在(0, 1-3)和(1, 1-3)之间。因此，需要对(0,1-3)和(1,1-3)的数据都进行处理。对于浮点数，只需要直接取整。

```
int dir[4][2] = {{0,-1},{-1,0},{0,1},{1,0}};
```

对于一道墙(x, y, t, d)，t＝1时，需要处理wall[x][y:y+d][1], wall[x-1][y:y+d][3]

t＝0时，需要处理wall[x:x+d][y][0], wall[x:x+d][y-1][2]

解题思路

3. 广度优先搜索。

由于Nemo的位置不确定，但出发点(0,0)固定，所以可以将Nemo的位置作为出发点，反过来寻找怎么走出迷宫。

不太最短路，而是找最少的门。需要修改剪枝方法。

walked[i][j]：是否走过这个房间

解题思路

3. 广度优先搜索。

由于Nemo的位置不确定，但出发点(0,0)固定，所以可以将Nemo的位置作为出发点，反过来寻找怎么走出迷宫。

不太最短路，而是找最少的门。需要修改剪枝方法。

walked[i][j]：是否走过这个房间

walked[i][j]: 从nemo位置到达[i][j]最少需要穿过几道门

- 到达坐标(0,0)后不能退出搜索，而是要走到无法继续走了为止，保证walked的每个元素都是最小值

解题思路

4.搜索边界。

遍历200×200个格子可能导致TE，如何缩短时间？

解题思路

4.搜索边界。

遍历200×200个格子可能导致TE，如何缩短时间？

人为设置边界。在输入时找到迷宫在x,y轴出现的最远的位置，加1(在边缘留一条路)并储存到全局变量中表示边界。

注意，Nemo出现的位置可能比迷宫出现的位置要远，则边界是Nemo的x,y轴加1。最后输出walked(0,0)，即从Nemo位置到达(0,0)最少穿过的门数量。

解题思路

```
while(!Path.empty()){
    for(int i=0;i<4;i++){
        start = Path.front();
        Maze next = start;
        next.x += dir[i][0]; next.y += dir[i][1];
        if(walls[start .x][start .y].wall[i]==2) next.tot++;
         //这里wall不可以使用bool，因为有墙/门/空白三种状态
        if(next.x<0 || next.x>=r || next.y<0 || next.y>=c ||
                (walked[next.x][next.y]<=next.tot && walked[next.x][next.y]!=-1) ||
                walls[start.x][start.y].Around[i]==1) continue;
        walked[next.x][next.y]=next.tot;
        Path.push(next);}
    Path.pop();
}
```

解题思路

本题也可以使用深度优先搜索。

解题思路

本题也可以使用深度优先搜索。

由于不是完全的求最短路，可以将广度优先搜索换一个形式，用一个全局变量ans储存从Nemo到(0,0)最少穿过的门数量。

搜索中每次到达(0,0)就更新一次最小值，最后输出ans或者walked(0,0)。

解题思路

```
void flag(int x,int y,int tot)
{
    if(x==(int)Nemo_x && y==(int)Nemo_y){
        ans=min(ans,tot);
        return;
    }
    if(walked[x][y]<=tot && walked[x][y]!=-1) return;
    walked[x][y]=tot;
    for(int i=0;i<4;i++){
        int next_x=x+dir[i][0],next_y=y+dir[i][1];
        if(next_x<0 || next_x>=r || next_y<0 || next_y>=c) continue;
        if(walls[x][y].Around[i]==2) flag(next_x,next_y,tot+1); //有门，穿过去
        if(walls[x][y].Around[i]==0) flag(next_x,next_y,tot);    //没有门也没有墙，直接走
```

单词接龙

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
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（只含有大写或小写字母，长度不超过20），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
只需输出以此字母开头的最长的“龙”的长度。

单词接龙

样例输入
5
at
touch
cheat
choose
tact
a
样例输出
23
提示
连成的“龙”为atoucheatactactouchoose

解题思路

1. 确定两单词重合部分的“头尾”:

将后一个字符串的第i个字符与前一个字符串的最后一个字符对比，得到重合部分的开始

向前遍历一遍核对是否每一个字母都对应相同，确定两个字符串是否能够相连

2. 将现在所连的总单词长度求出，进入下一轮搜索

解题思路

注意题目中的几个限制条件。

1. 每个单词都最多在“龙”中出现两次。

2. 每两个单词相连时，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。

3. 每相邻的两部分不能存在包含关系，例如at和atide间不能相连。但此处存在问题。

解题思路

```
void check(int x,int m){   //被接单词和接龙总长
    ans=max(ans,m);        //找最长
    for(int y=1;y<=n;y++){                    //遍历所有单词
        if(!used[y]) continue;
        for(int i=0;i<s[x].length();i++)      //遍历候选者所有字符
            if(s[x][i]==s[y][0]){             //有字符和第一个字符一样就开始
                int iy=1; bool flag=1;
                for(int ix=i+1;ix<s[x].length()&&iy<s[y].length();ix++,iy++)
                    if(s[x][ix]!=s[y][iy]){ flag=0; break; } //有字符不一样，直接退出
                    if(flag){
                        k[y]--;//用了一次
                        check(y,m+s[y].length()-iy);//只记长度
                        k[y]++;//回溯
            }
```

碎纸机

你现在负责设计一种新式的碎纸机。一般的碎纸机会把纸切成小片，变得难以阅读。而你设计的新式的碎纸机有以下的特点：

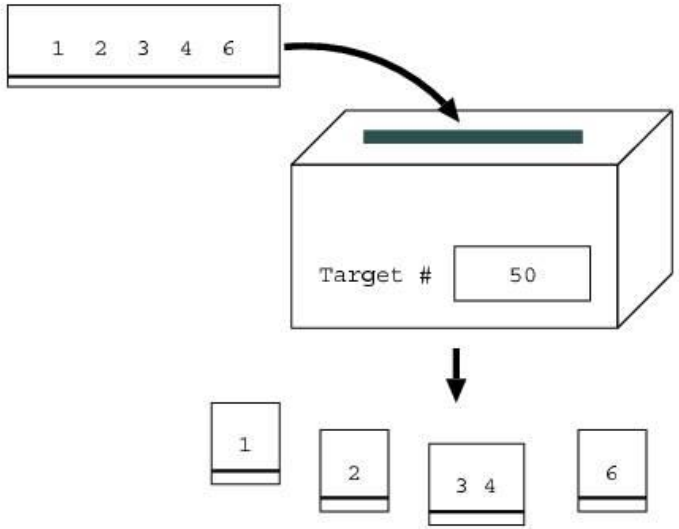
1.每次切割之前，先要给定碎纸机一个目标数，而且在每张被送入碎纸机的纸片上也需要包含一个数。

2.碎纸机切出的每个纸片上都包括一个数。

3.要求切出的每个纸片上的数的和要不大于目标数而且与目标数最接近。

碎纸机

举一个例子，如下图，假设目标数是50，输入纸片上的数是12346。碎纸机会把纸片切成4块，分别包含1，2，34和6。这样这些数的和是43 (= 1 + 2 + 34 + 6)，这是所有的分割方式中，不超过50，而又最接近50的分割方式。又比如，分割成1，23，4和6是不正确的，因为这样的总和是34 (= 1 + 23 + 4 + 6)，比刚才得到的结果43小。分割成12，34和6也是不正确的，因为这时的总和是52 (= 12 + 34 + 6)，超过了50



碎纸机

还有三个特别的规则：
1.如果目标数和输入纸片上的数相同，那么纸片不进行切割。
2.如果不论怎样切割，分割得到的纸片上数的和都大于目标数，那么打印机显示错误信息。
3.如果有多种不同的切割方式可以得到相同的最优结果。那么打印机显示拒绝服务信息。比如，如果目标数是15，输入纸片上的数是111，那么有两种不同的方式可以得到最优解，分别是切割成1和11或者切割成11和1，在这种情况下，打印机会显示拒绝服务信息。

为了设计这样的一个碎纸机，你需要先写一个简单的程序模拟这个打印机的工作。给定两个数，第一个是目标数，第二个是输入纸片上的数，你需要给出碎纸机对纸片的分割方式。

碎纸机

输入

输入包括多组数据，每一组包括一行。每行上包括两个正整数，分别表示目标数和输入纸片上的数。已知输入保证：两个数都不会以0开头，而且两个数至多都只包含6个数字。输入的最后一行包括两个0，这行表示输入的结束。

输出

对每一组输入数据，输出相应的输出。有三种不同的输出结果：

sum part1 part2 ...

rejected

error

每一个partj是切割得到的纸片上的一个数。partj的顺序和输入纸片上原始数中数字出现次序一致。

如果不论怎样切割，分割得到的纸片上数的和都大于目标数，那么打印“error”。

如果有多种不同的切割方式可以得到相同的最优结果，那么打印“rejected”。

碎纸机

样例输入

```
50 12346
376 144139
927438 927438
18 3312
9 3142
25 1299
111 33333
103 862150
6 1104
0 0
```

样例输出

```
43 1 2 34 6
283 144 139
927438 927438
18 3 3 12
error
21 1 2 9 9
rejected
103 86 2 15 0
rejected
```

碎纸机

本题数据范围非常小，因此可以在区间内枚举断点。

碎纸机

本题数据范围非常小，因此可以在区间内枚举断点。

深度优先搜索：dfs(int start,int end, int cnt)

start代表区间起点，end代表区间的中点，cnt代表这个区间内断点的数量。结束条件有两个:

1. 当拆分出来的数的总和＞n，不满足拆分要求，直接返回。

2. 当没有断点可以枚举的时候，计算整个数分解完之后的结果之和，并判断是否大于当前最大值。

碎纸机

```
void dfs(int dq,int sum,int cnt)  {
    if(dq>=m){                                  //情况2，切割完了
            vis[sum]++;                         //记录是否有多种不同的切割方式
            if(sum>ans){
                    ans=sum;                    //记录最大的值
                    num=cnt;                    //记录切割次数
                    for(int i=0;i<cnt;i++)
                            step[i]=now[i];         //切割方法
            }
            return;
    }
```

碎纸机

```
    int t=0;
    for(int i=dq;i<m;i++){                  //从当前位置搜索
            t=t*10+str[i]-'0';              //当前切割的子串
            if(sum+t>n)                     //情况1，大于目标值
                    return;
            now[cnt]=t;                     //记录切割方法
            dfs(i+1,sum+t,cnt+1);           //递归搜索
    }
}
```

生日蛋糕

7月17日是Mr.W的生日，ACM-THU为此要制作一个体积为Nπ的M层生日蛋糕，每层都是一个圆柱体。设从下往上数第i(1 <= i <= M)层蛋糕是半径为Ri, 高度为Hi的圆柱。当i < M时，要求Ri > Ri+1且Hi > Hi+1。
由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积Q最小。令Q = Sπ。请编程对给出的N和M，找出蛋糕的制作方案（适当的Ri和Hi的值），使S最小。
除Q外，以上所有数据皆为正整数

输入
有两行，第一行为N（N <= 10000），表示待制作的蛋糕的体积为Nπ；第二行为M(M <= 20)，表示蛋糕的层数为M。
输出
仅一行，是一个正整数S（若无解则S = 0）。

生日蛋糕

样例输入
100
2

样例输出
68

提示
圆柱公式
体积V = πR2H
侧面积A' = 2πRH
底面积A = πR2

生日蛋糕

总结题意:

给定M个圆柱体的总体积Nπ，要求编程求出使表面积最小的圆柱体设计方案。注意:
1. 上下两个底面的表面积完全取决于底层蛋糕的半径，上层蛋糕只有侧面积参与了表面积的贡献。
2. 从上向下，第i层的最小宽高都是i(必须一层比一层大)，可以提前计算出最小剩余体积用于剪枝。

思路:

利用DFS从底层开始向上枚举每一层可能的高度和半径，并剪枝来减少多余操作。

生日蛋糕

搜索范围：
1. 底层蛋糕的最大可能半径N,即只有一层、高度为1的情况。
2. 底层蛋糕的最大可能高度N，即只有一层，底层半径为1。

剪枝：
1. 当前体积加上剩余的最小体积仍然大于N，则直接返回。如剩余体积1，但是需要堆叠两层蛋糕。
2. 当前体积加上剩下的最大体积仍小于N，则直接返回。如当前半径和高度都是2，剩余体积5，只有一层。
3. 高度或半径已无法安排，高度半径都为整数且下一层大于上一层。
4. 已经建好的表面积大于当前最佳策略。
5. 剩下的部分全部用来做成一个圆柱体（此时表面积最小）再加上当前表面积仍大于当前最佳策略。

生日蛋糕

```
int MaxVforNRH(int n, int r, int h) {
    int v = 0;
    for (int i = 0; i < n; i++) v += (r - i)*(r - i)*(h - i);
    return v;}
void dfs(int curr, int sumS, int sumV, int r, int h){//用curr层圆柱，底层的半径高度为rh
  if(curr==0){
    if(sumV==N && best>sumS)
      best = sumS;
    return;}
  if((sumV+leftMinV[curr]>N) || (sumS+leftMinS[curr]>best)) return;//最小体积偏大
  if (sumV + MaxVforNRH(curr, r, h) < N) return;       //剩下最大体积不够
  if((2*(N-sumV)/r + sumS) >=best)      //剩下只做一个圆柱体仍然大于最优策略
    return;
```

生日蛋糕

```
    for(int i=r-1; i>=curr; i--){
      if(curr==M)              //在底层
         sumS = i*i;
      for(int j=h-1; j>=curr; j--){
         dfs(curr-1, sumS+i*j*2, sumV+i*i*j, i, j);
      }
   }
}
```

这里还有剪枝空间，h可以用leftminV来求，和h-1取min