BFS

信智学院　陈琰宏

主要内容

01

02

03

04

05

[2884] 围圈报数

1 [2806] 走出迷宫

2 [2805]抓住那头牛

3 [1190] 武士风度的牛

4 [2879] 细胞

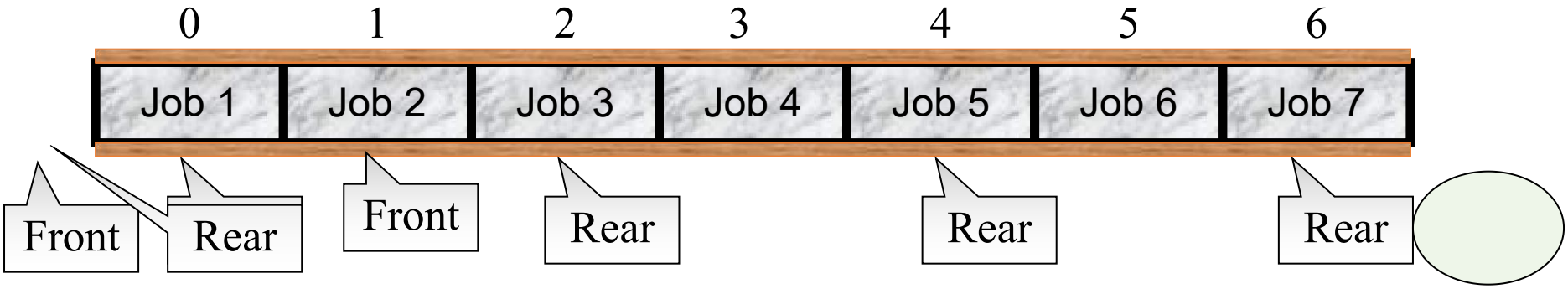
5 [2800] 迷宫问题

6 [2809] Knight Moves

7 [7330] 加工零件

1 队列的定义-顺序存储

队列的顺序存储结构通常由一个一维数组和一个记录队列头元素位置的变量front以及一个记录队列尾元素位置的变量rear组成。



AddQ Job 1	AddQ Job 2	AddQ Job 3	DeleteQ Job 1
AddQ Job 4	AddQ Job 5	AddQ Job 6	DeleteQ Job 2
AddQ Job 7	AddQ Job 8		

2 STL queue

包含头文件：#include <queue>
使用命名空间：using namespace std;
定义栈的方法：

```
queue<char> S1;    //队列中的结点为字符
queue<int> S2;     //队列中的结点为整型数据
queue<pos> S3; //队列中的结点为自定义结构体pos变量
```

queue的成员函数：
back()返回最后一个元素
empty()如果队列空则返回真
front()返回第一个元素
pop()删除第一个元素
push()在末尾加入一个元素
size()返回队列中元素的个数

[2884] 围圈报数

有 n 个人依次围成一圈，从第 1 个人开始报数，数到第m个人出列，然后从出列的下一个人开始报数，数到第m个人又出列，…，如此反复到所有的人全部出列为止。设 n 个人的编号分别为1，2，…，n，打印出列的顺序。

输入

n和m。

输出

出列的顺序。

样例输入

4 17

样例输出

1 3 4 2

[2884] 围圈报数

```cpp
#include<bits/stdc++.h>
using namespace std;
int main()
{
    queue<int>student;
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        student.push(i);
    while (!student.empty())
    {
        for (int i = 1; i <= m-1; i++)
        {
            int x = student.front();
            student.pop();
            student.push(x);
        }
        cout << student.front() << " ";
        student.pop();
    }
}
```

3 广度优先搜索

广度优先搜索算法，又称宽度优先搜索。广度优先算法的核心思想是：**从初始节点开始，应用算符生成第一层节点，检查目标节点是否在这些后继节点中，若没有，再用产生式规则将所有第一层的节点逐一扩展，得到第二层节点，并逐一检查第二层节点中是否包含目标节点。**若没有，再用算符逐一扩展第二层的所有节点……，如此依次扩展，检查下去，直到发现目标节点为止。即

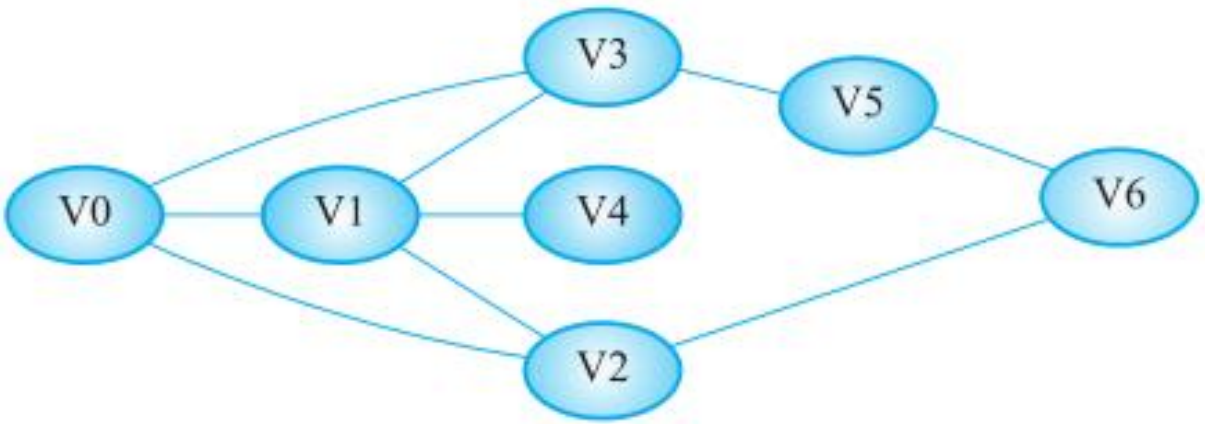
1.从图中的某一顶点V0开始，先访问V0；

2.访问所有与V0相邻接的顶点V1，V2，……，Vt；

3.依次访问与V1，V2，……，Vt相邻接的所有未曾访问过的顶点；

4.循此以往，直至所有的顶点都被访问过为止。

3.5 广度优先搜索



如图所示，从顶点 V0 开始进行广度优先搜索，得到的一个序列为 V0，V1，V2，V3，V4，V6，V5。

广度优先搜索是一种“盲目”搜索，所有结点的拓展都遵循“先进先出”的原则，所以采用“队列”来存储这些状态。宽度优先搜索的算法框架如下：

3 广度优先搜索

```
void BFS{
    while (front <rear){
    // 当队列非空时做，front 和 rear 分别表示队列的头指针和尾指针
        if (找到目标状态)
            做相应处理(如退出循环输出解、输出当前解、比较解的优劣);
            else{
                    拓展头结点 ;
                    if( 拓展出的新结点没出现过 ){
                        rear++;
                       将新结点插到队尾 ;
                    }
            }
         front++;// 取下一个结点
    }
}
```

1 [2806] 走出迷宫

当你站在一个迷宫里的时候，往往会被错综复杂的道路弄得失去方向感，如果你能得到迷宫地图，事情就会变得非常简单。
假设你已经得到了一个n*m的迷宫的图纸，请你找出从起点到出口的最短路。
输入　第一行是两个整数n和m(1≤n,m≤100)，表示迷宫的行数和列数。
接下来n行，每行一个长为m的字符串，表示整个迷宫的布局。
字符‘.’表示空地，‘#’表示墙，‘S’表示起点，‘T’表示出口。
输出　输出从起点到出口最少需要走的步数。
样例输入

```
3 3
S#T
.#.
...
```

样例输出

```
6
```

1 [2806] 走出迷宫

搜索从(xi,yi)到(xe,ye)路径的过程是：首先将(xi,yi)进队，在队列qu不为空时循环：出队一次（由于不是循环队列，该出队元素仍在队列中），称该出队的方块为当前方块，qu.front为该方块在队列中的下标位置。如果当前方块是出口，则按入口到出口的次序输出该路径并结束。　否则，按顺时针方向找出当前方块的4个方位中可走的相邻方块（对应的迷宫a数组值为0），将这些可走的相邻方块均插入到队列qu中，其pre设置为本搜索路径中上一方块在qu中的下标值，也就是当前方块的qu.front值，并将相邻方块对应的a数组元素值置为-1，以避免回过来重复搜索。如此队列为空，表示未找到出口，即不存在路径。

1 [2806] 走出迷宫-设计算法

下标	i	j	pre	下标	i	j	pre
0	1	1	-1	21	1	6	18
1	1	2	0	22	6	5	20
2	2	1	0	23	5	5	22
3	2	2	1	24	7	5	22
4	3	1	2	25	4	5	23
5	3	2	3	26	5	6	23
6	4	1	4	27	8	5	24
7	3	3	5	28	4	6	25
8	5	1	6	29	5	7	26
9	3	4	7	30	8	6	27
10	5	2	8	31	8	4	27
11	6	1	8	32	4	7	28
12	2	4	9	33	5	8	29
13	5	3	10	34	6	7	29
14	7	1	11	35	8	7	30
15	1	4	12	36	8	3	31
16	2	5	12	37	3	7	32
17	6	3	13	38	4	8	32
18	1	5	15	39	6	8	33
19	2	6	16	40	8	8	35
20	6	4	17				

1 1	1 2	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

1 [2806] 走出迷宫-BFS

```
4   #include<queue>
5   using namespace std;
6   int n,m,vis[110][110];////用来标记有没有走过（有没有在队列中)
7   int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向 上右下左
8   char a[110][110];
9   struct node{
10      int r,c,step;//row ,column,step
11  };
12  void bfs(int sr,int sc,int er,int ec){
38  int main(){
39      int i,j,sr,sc,er,ec;
40      scanf("%d%d",&n,&m);
41      for(i=0;i<n;i++)scanf("%s",a[i]);//读入字符数组 a[i]表示每行的首地址
42
43      for(i=0;i<n;i++)
44          for(j=0;j<m;j++)//求出入口和出口的位置
45              if(a[i][j]=='S')sr=i,sc=j;
46              else if(a[i][j]=='T')er=i,ec=j,a[i][j]='.';
47      bfs(sr,sc,er,ec);//把出口和入口作为参数
48      return 0;
49  }
```

1 [2806] 走出迷宫

```
12  void bfs(int sr,int sc,int er,int ec){
13      int k;
14      memset(vis,0,sizeof(vis));
15      queue<node> qe;//定义队列
16      node q,t;//q表示当前队头位置，t表示下一个位置
17      q.r=sr,q.c=sc,q.step=0;//把起点入队
18      vis[sr][sc]=1;//记录已经走过
19      qe.push(q);//入队
20      while(!qe.empty()){
21          q=qe.front();//取出队头元素
22          qe.pop();
23          if(er==q.r&&ec==q.c){//到出口了
24              printf("%d",q.step);
25              break;
26          }
27          for(k=0;k<4;k++){//继续搜索4个方向
28             t.r=q.r+dir[k][0],t.c=q.c+dir[k][1];
29              //如果没有出迷宫、并且该位置可走、没走过
30              if(0<=t.r&&t.r<n&&0<=t.c&&t.c<m&&vis[t.r][t.c]==0&&a[t.r][t.c]=='.'){
31                  vis[t.r][t.c]=1;//走到这个位置
32                  t.step=q.step+1;
33                 qe.push(t);
34              }
35          }
36      }
37  }
```

1 [2806] 走出迷宫

```
void bfs(int sr,int sc,int er,int ec){
   int k;  memset(vis,0,sizeof(vis));
   queue<node> qe;   node q,t;//q表示当前队头位置，t表示下一个位置
   ________1_____________;//
    _______2_____________;//
   qe.push(q);//入队
   while(!qe.empty()){
       ______3_____________;//
      if(er==q.r&&ec==q.c){//到出口了
         printf("%d",q.step);  break;
      }
      for(k=0;k<4;k++){//继续搜索4个方向
         ________4____________;
          //如果没有出迷宫、并且该位置可走、没走过
         if(0<=t.r&&t.r<n&&0<=t.c&&t.c<m&&vis[t.r][t.c]==0&&a[t.r][t.c]=='.'){
            vis[t.r][t.c]=1;//走到这个位置
            ______5_____________;
           qe.push(t);
         }
      }
   }
```

1 [2806] 走出迷宫-DFS

```
#include<iostream>
using namespace std;
int n,m,ans,sx,sy,ex,ey;
char a[40][40];
int dir[4][2]={-1,0,1,0,0,-1,0,1};
void dfs(int x,int y,int min)
{
```

```
int main()
{
    cin>>n>>m;
    ans=1000;
       for(int i=0;i<n;i++)
       {
           for(int j=0;j<m;j++)
           {
               cin>>a[i][j];
               if(a[i][j]=='S')
                   sx=i,sy=j;
               if(a[i][j]=='T')
                   ex=i,ey=j;
           }
       }
       dfs(sx,sy,0);
       cout<<ans<<endl;
    return 0;
}
```

```
void dfs(int x,int y,int min)
{
    if(x==ex&&y==ey)
    {
        if(min<ans)ans=min;
        return;
    }
    if(x<0||x>=n||y<0||y>=m)return;
        for(int i=0;i<4;i++)
        {
            int dx=x+dir[i][0];
            int dy=y+dir[i][1];
            if(a[dx][dy]=='.'||a[dx][dy]=='T')
            {
                a[x][y]='#';
                dfs(dx,dy,min+1);
                a[x][y]='.';
            }
        }
        return;
}
```

2 [2805]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。

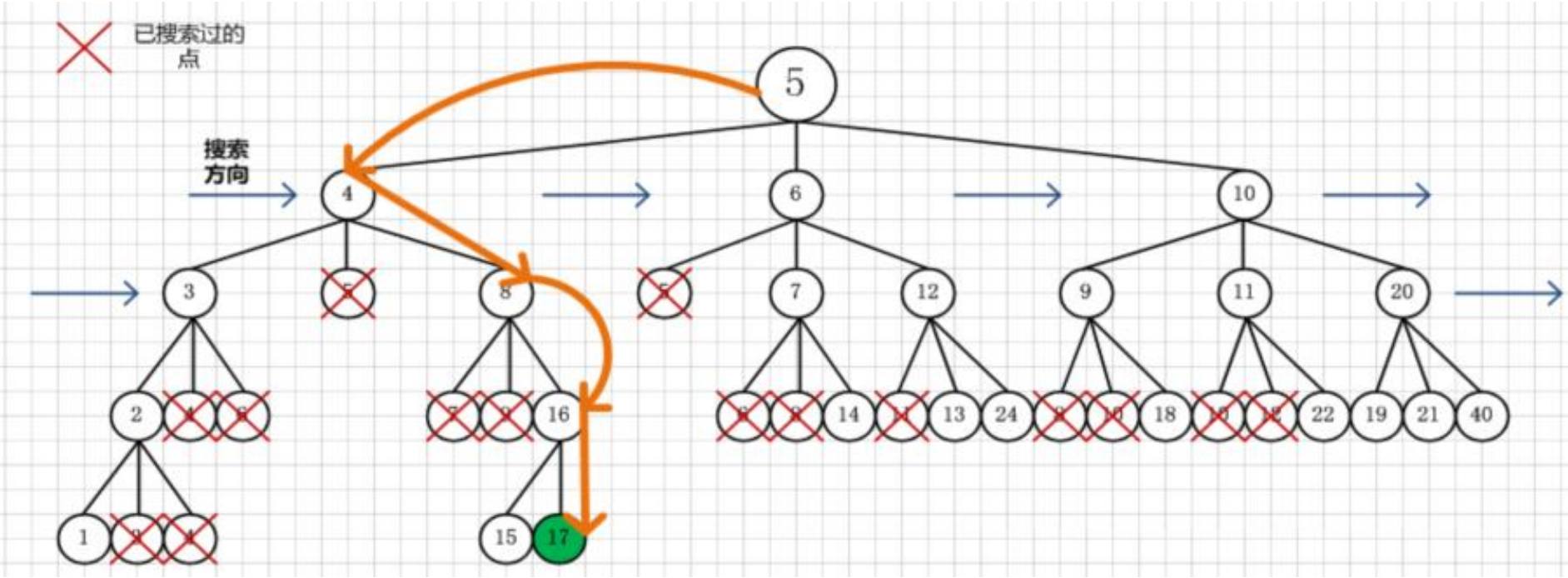
输出一个整数，农夫抓到牛所要花费的最小分钟数。

样例输入 5 17

样例输出 4

2 [2805]抓住那头牛-分析

从5扩展开去，每次扩展 x-1, x+1, 2x, 扩展的点跟目标点进行配对。



广搜算法

(1) 把初始节点S0放入Open表中；

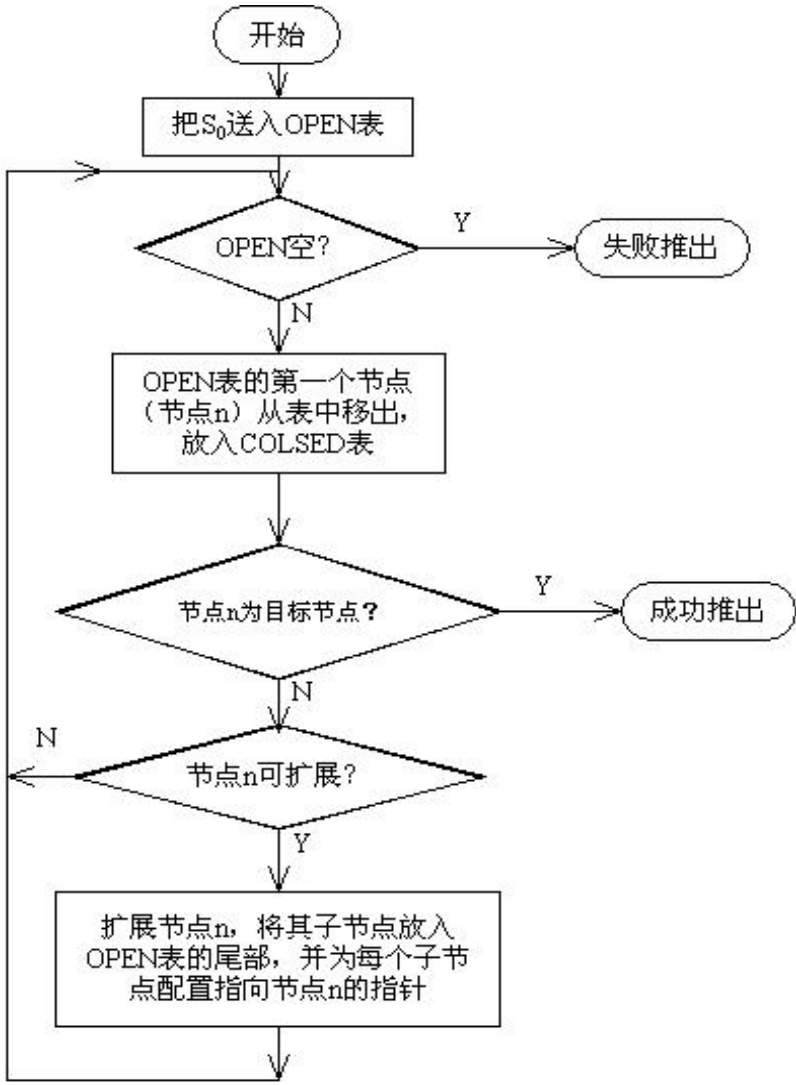
(2) 如果Open表为空，则问题无解，失败 退出；

(3) 把Open表的第一个节点取出放入Closed表，并记该节点为n；

(4) 考察节点n是否为目标节点。若是， 则得到问题的解，成功退出；

(4) 若节点n不可扩展，则转第(2)步；

(5) 扩展节点n，将其不在Closed表和Open表中的子节点(判重）放入Open表的尾部，并为每一个子节点设置指向父节点的指针(或记录节点的层次），然后转第(2)步。



2 [2805]抓住那头牛-代码

```
1  #include <iostream>
2  #include<cstring>
3  #include <queue>
4  #include <algorithm>
5  #define max_n 100001
6  using namespace std;
7
8  int n,k,cur,res;
9  int closed[max_n],d[3];//closed数组储存最小分钟数，且标记已经访问
10 queue<int> open;
11
```

```
45 int main() {
46     cin >> n >> k;
47     bfs();
48     return 0;
49 }
50
```

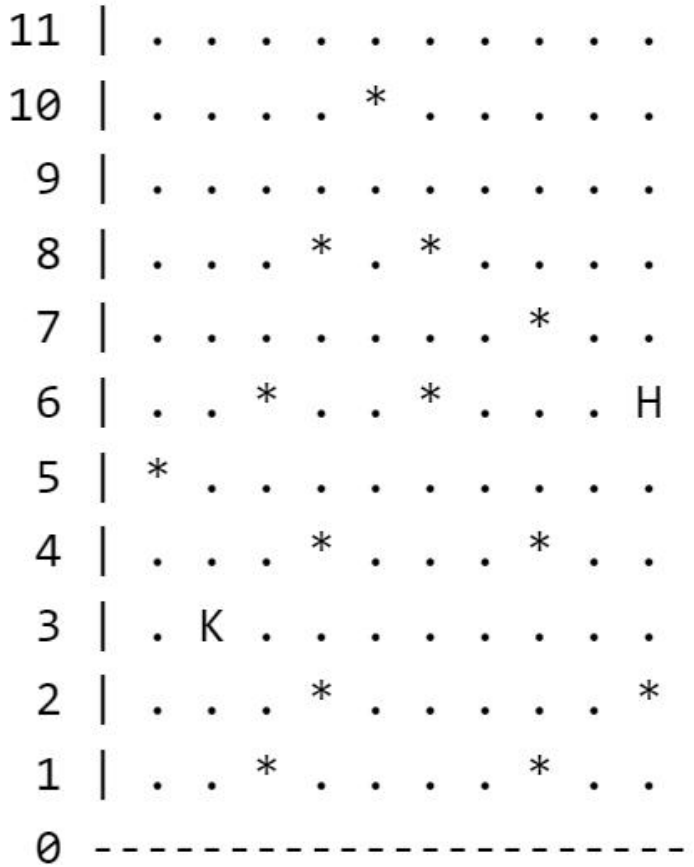
```
12  void bfs(){
13
14      //初始化closed数组
15      memset(closed,-1,sizeof(closed));
16      open.push(n);//将起点加入open队列
17      closed[n] = 0;// 从位置n开始，把起始点设置为0
18      while (!open.empty()){
19
20          //若当前值为终点，则退出循环
21          cur = open.front();
22          open.pop();
23          if (cur == k) break;
24          d[0] = cur - 1;//三种选择
25          d[1] = cur + 1;
26          d[2] = cur * 2;
27          //对三种选择bfs
28          for (int i = 0;i < 3;i++){
29              //注意数据不能越界且不能被访问
30              if (d[i] >= 0 && d[i] < max_n && closed[d[i]] == -1){
31                  open.push(d[i]);
32                  closed[d[i]] = closed[cur] + 1;
33              }
34          }
35      }
36      res = closed[k];
37      cout << res << endl;
38  }
39
```

3 [1190] 武士风度的牛

农民John有很多牛，他想交易其中一头被Don称为The Knight的牛。
这头牛有一个独一无二的超能力，在农场里像Knight一样地跳（就是我们熟悉的象棋中马的走法）。
虽然这头神奇的牛不能跳到树上和石头上，但是它可以在牧场上随意跳，我们把牧场用一个x，y的坐标图来表示。

这头神奇的牛像其它牛一样喜欢吃草，给你一张地图，上面标注了The Knight的开始位置，树、灌木、石头以及其它障碍的位置，除此之外还有一捆草。

现在你的任务是，确定The Knight要想吃到草，至少需要跳多少次。The Knight的位置用’ K’ 来标记，障碍的位置用’ *’ 来标记，草的位置用’ H’ 来标记。



The Knight的位置用’K’来标记，障碍的位置用’*’来标记，草的位置用’H’来标记。

```cpp
#include<bits/stdc++.h>
using namespace std;//方向数组
const int N=155;
const int dir[8][2]={{2,1},{-2,1},{2,-1},{-2,-1},
                     {-1,2},{1,2},{-1,-2},{1,-2}};
struct node{
    int x,y;
};
int n,m,sx,sy,ex,ey;
char Map[N][N];//地图
int dis[N][N];//记录到达 (i,j)点的最短距离
```

```
38 int main()
39 {
40     memset(dis,0,sizeof(dis));
41     scanf("%d %d",&m,&n);
42     for(int i=1;i<=n;i++)
43         for(int j=1;j<=m;j++){
44             cin>>Map[i][j];
45             if(Map[i][j]=='K'){//记录起点
46                 sx=i,sy=j;
47                 dis[i][j]=0;
48             }
49             if(Map[i][j]=='H')//记录终点
50                 ex=i,ey=j;
51         }
52     printf("%d\n",BFS());
53 }
```

```
13 bool check(int x,int y)//判断当前状态是否合法
14 {
15     if(x<1||x>n||y<1||y>m||dis[x][y]||Map[x][y]=='*')
16         return false;
17     return true;
18 }
19 int BFS()
20 {
21     queue<node> q;
22     q.push(node{sx,sy});
23     while(!q.empty()){
24         node now=q.front();
25         q.pop();
26         if(now.x==ex && now.y==ey)
27             return dis[now.x][now.y];
28 
29         for(int i=0;i<8;i++){
30             int dx=now.x+dir[i][0],dy=now.y+dir[i][1];
31             if(!check(dx,dy))
32                 continue;
33             q.push(node{dx,dy});
34             dis[dx][dy]=dis[now.x][now.y]+1;//距离加一
35         }
36     }
37 }
```

4 [2879] 细胞

一矩形阵列由数字0到9组成,数字1到9代表细胞,细胞的定义为沿细胞数字上下左右还是细胞数字则为同一细胞,求给定矩形阵列的细胞个数。如:
阵列
4 10
0234500067
1034560500
2045600671
0000000089
有4个细胞。

输入格式
第一行为矩阵的行n和列m; n<300,m<300
下面为一个n*m的矩阵。

输出格式
细胞个数。

4 [2879] 细胞

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

char a[300][300];  //地图
int vis[300][300];  //记录路径 有没有到过
int dx[4]={0,0,1,-1};  //偏
int dy[4]={1,-1,0,0};
int n,m,cnt=0;  //cnt数量
//结构体
struct point{
    int x,y;
};

int main(){
    cin>>n>>m;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>a[i][j];
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(vis[i][j]==0 && a[i][j]!='0'){
                cnt++;//只要符合要求而且没去过
                bfs(i,j);  //广搜
            }
        }
    }
    cout<<cnt;  //输出
    return 0;
}
```

```
queue <point> q; //bfs队列
void bfs(int x,int y){
    point p;
    p.x=x;p.y=y;
    q.push(p);
    while(!q.empty()){  //当队列不为空
        point tmp=q.front();
        q.pop();  //记得pop弹出
        for(int i=0;i<4;i++){
            point np=tmp;
            np.x+=dx[i];  //偏移量
            np.y+=dy[i];
            if(np.x<n && np.x>=0 &&
            np.y<m && np.y>=0 &&
            vis[np.x][np.y]==0 &&
            a[np.x][np.y]!='0'){
                //假如点坐标合法而且没去过
                q.push(np);
                vis[np.x][np.y]=vis[tmp.x][tmp.y]+1;  //记录步数
            }
        }
    }
}
```

4 [2879] 细胞

dfs 是否可以?

```cpp
#include<bits/stdc++.h>
using namespace std;
char c[300][300];
int n,m;
void dfs(int x,int y){
    if(x<0||y<0||x>n-1||y>m-1||c[x][y]=='0')return;
    c[x][y]='0';
    dfs(x-1,y);
    dfs(x+1,y);
    dfs(x,y-1);
    dfs(x,y+1);
}
```

```cpp
int main(){
    cin>>n>>m;
    int sum=0;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>c[i][j];
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(c[i][j]!='0'){
                sum++;
                dfs(i,j);
            }
        }
    }
    cout<<sum;
    return 0;
}
```

5 [2800] Dungeon Master迷宫问题

这 题是一个三维的迷宫题目，其中用 ‘.’ 表示空地， ‘#’ 表示障碍物， ‘S’ 表示起点， ‘E’ 表示终点，求从起点到终点的最小移动次数，解法和二维的类似，只是在行动时除了东南西北移动外还多了上下。可以上下左右前后移动，每次都只能移到相邻的空位，每次需要花费一分钟，求从起点到终点最少要多久。

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,m,kk;
int vis[100][100][100];
char s[100][100][100];
int dir[][3]={{1,0,0},{-1,0,0},{0,1,0},
                    {0,-1,0},{0,0,1},{0,0,-1}};
struct node{
    int a,b,c,step;
};

int check(node t ){
    if(0<=t.a&&t.a<n&&
       0<=t.b&&t.b<m&&
       0<=t.c&&t.c<kk&&
       s[t.a][t.b][t.c]=='.'&&
       vis[t.a][t.b][t.c]==0)
           return 1;
    else
          return 0;

}
```

```
23 void bfs(int sa,int sb,int sc,int ea,int eb,int ec){
24     int k,flag=0;
25     memset(vis,0,sizeof(vis));
26     queue<node> qe;
27     node q,t;
28     q.a=sa,q.b=sb,q.c=sc,q.step=0;vis[sa][sb][sc]=1;
29     qe.push(q);
30     while(!qe.empty()){
31         q=qe.front();
32         qe.pop();
33         if(q.a==ea&&q.b==eb&&q.c==ec){
34             printf("Escaped in %d minute(s).\n",q.step),flag=1;
35             break;
36         }
37         for(k=0;k<6;k++){
38             t.a=q.a+dir[k][0],t.b=q.b+dir[k][1],t.c=q.c+dir[k][2];
39             if(check(t)){
40                 vis[t.a][t.b][t.c]=1;
41                 t.step=q.step+1;
42                 qe.push(t);
43             }
44         }
45     }
46     if(flag==0)printf("Trapped!\n");
47 }
```

```
48 int main(){
49
50     int i,j,k,sa,sb,sc,ea,eb,ec;
51     scanf("%d%d%d",&n,&m,&kk);
52         for(i=0;i<n;i++)
53             for(j=0;j<m;j++)
54                 scanf("%s",s[i][j]);
55         for(i=0;i<n;i++)
56             for(j=0;j<m;j++)
57                 for(k=0;k<kk;k++){
58                     if(s[i][j][k]=='S')sa=i,sb=j,sc=k;
59                     if(s[i][j][k]=='E')ea=i,eb=j,ec=k,s[i][j][k]='.';
60                 }
61         bfs(sa,sb,sc,ea,eb,ec);
62     return 0;
63 }
```

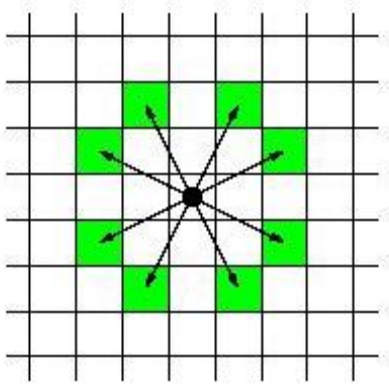
6 [2809] Knight Moves

【问题描述】

编写一个程序计算一个骑士从一点到另一点所需的最小步数。骑士可以移动到的位置如下图。

【输入格式】

第一行给出骑士的数量n。对于每一个骑士都有3行，第一行一个整数L表示棋盘的大小(4≤L≤300)，整个棋盘大小为L*L，第二行和第三行分别包含一对整数(x, y)表示骑士的起始点和终点。



黑点表示骑士，绿色格子表示可跳到的位置

【输出格式】

对每一个骑士，输出一行一个整数表示需要移动的最小步数。

基础广搜，按照模板直接写

```
 5  int vis[301][301];
 6  int len;
 7  int dirx[8]={-2,-2,-1,-1,1,1,2,2};//定义8个方向
 8  int diry[8]={-1,1,2,-2,2,-2,1,-1};
 9  int xe,ye;
10  struct Node{
11      int x,y,step;//定义每一个位置
12  };
13  void bfs(int x,int y){
39
40  int main(){
41      int n,xb,yb;
42      cin>>n;
43      while(n--){
44          cin>>len>>xb>>yb;
45          memset(vis,0,sizeof(vis));
46          cin>>xe>>ye;
47          bfs(xb,yb);
48      }
```

```
13  void bfs(int x,int y){
14      queue<Node>q;
15      Node start;
16      start.x=x,start.y=y,start.step=0;
17      vis[x][y]=1;//初始状态
18      q.push(start);//初始点入队
19      while(!q.empty()){
20          start=q.front();
21          q.pop();//取出队头元素
22          if(start.x==xe&&start.y==ye){
23              cout<<start.step<<endl;
24              return;
25          }
26          for(int i=0;i<8;i++){
36
37      }
38  }
```

```
26          for(int i=0;i<8;i++){
27              int xi=start.x+dirx[i];//取出8个方向
28              int yi=start.y+diry[i];
29              if(!vis[xi][yi]&&xi>=0&&xi<len&&yi>=0&&yi<len){
30                  Node s;//合法的点入队
31                  s.x=xi,s.y=yi,s.step=start.step+1;
32                  q.push(s);
33                  vis[xi][yi]=1;
34              }
35          }
36
37      }
38  }
```

7 [7330] 加工零件

凯凯的工厂正在有条不紊地生产一种神奇的零件，神奇的零件的生产过程自然也很神奇。工厂里有 n 位工人，工人们从 1~ n 编号。某些工人之间存在双向的零件传送带。保证每两名工人之间最多只存在一条传送带。

如果 x 号工人想生产一个被加工到第 L (L>1) 阶段的零件，则所有与 x 号工人有传送带直接相连的工人，都需要生产一个被加工到第 L - 1 阶段的零件（但 x 号工人自己无需生产第 L - 1 阶段的零件）。

如果 x 号工人想生产一个被加工到第 1 阶段的零件，则所有与 x 号工人有传送带直接相连的工人，都需要为 x 号工人提供一个原材料。

[7330] 加工零件

轩轩是 1 号工人。现在给出 q 张工单，第 i 张工单表示编号为 a_i 的工人想生产一个第 L_i 阶段的零件。轩轩想知道对于每张工单，他是否需要给别人提供原材料。他知道聪明的你一定可以帮他计算出来！

输入

第一行三个正整数 n，m 和 q，分别表示工人的数目、传送带的数目和工单的数目。

接下来 m 行，每行两个正整数 u 和 v，表示编号为 u 和 v 的工人之间存在一条零件传输带。保证 u≠v。

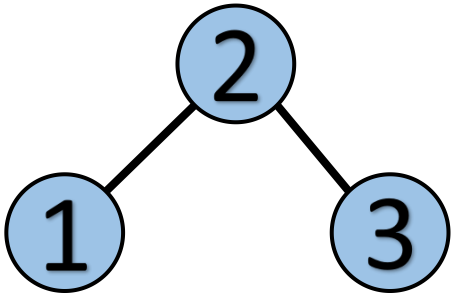
接下来 q 行，每行两个正整数 a 和 L，表示编号为 a 的工人想生产一个第 L 阶段的零件。

输出

共q行，每行一个字符串“Yes”或者“No”。如果按照第 i 张工单生产，需要编号为 1 的轩轩提供原材料，则在第 i 行输出“Yes”否则输出“No”。注意不含引号。

$1 \le n,m,q \le 10^5$，$1 \le L \le 10^9$

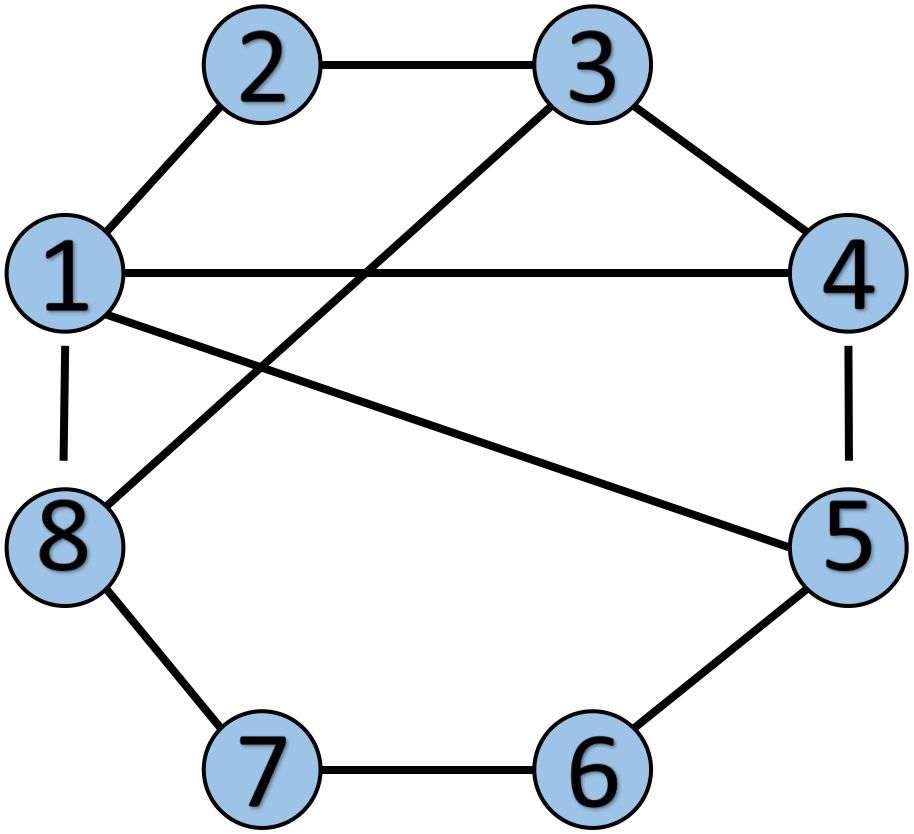
分析样例



编号为 a 的工人想生产一个第 L 阶段的零件。

a	L	out
1	1	No
2	1	Yes
3	1	No
1	2	Yes
2	2	No
3	2	Yes

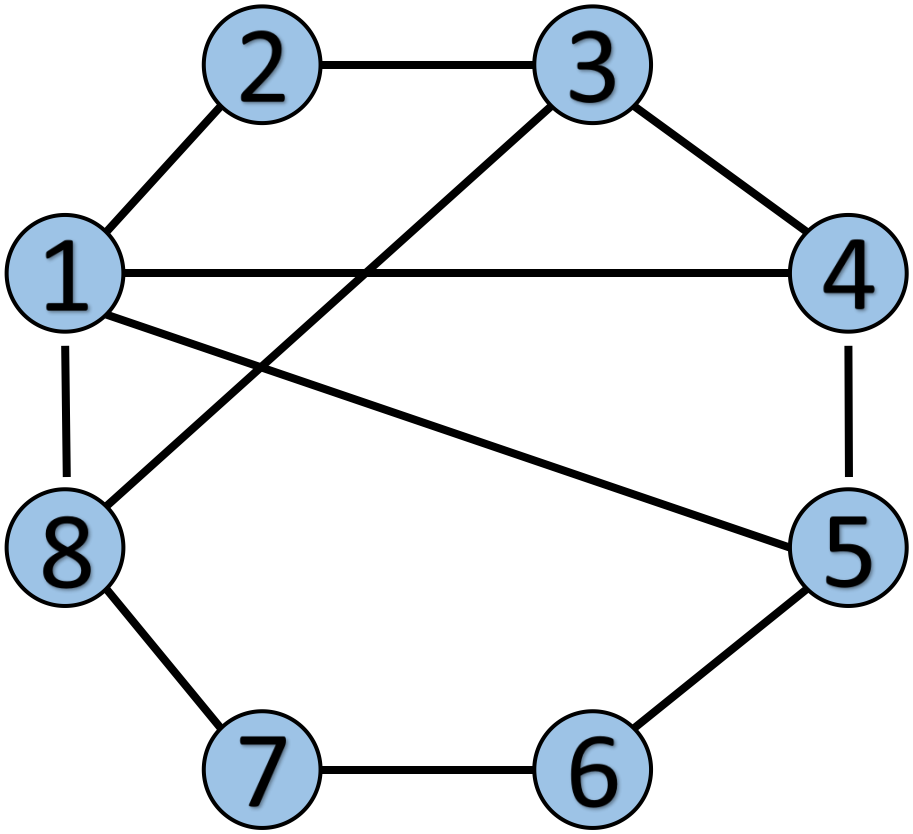
举例理解

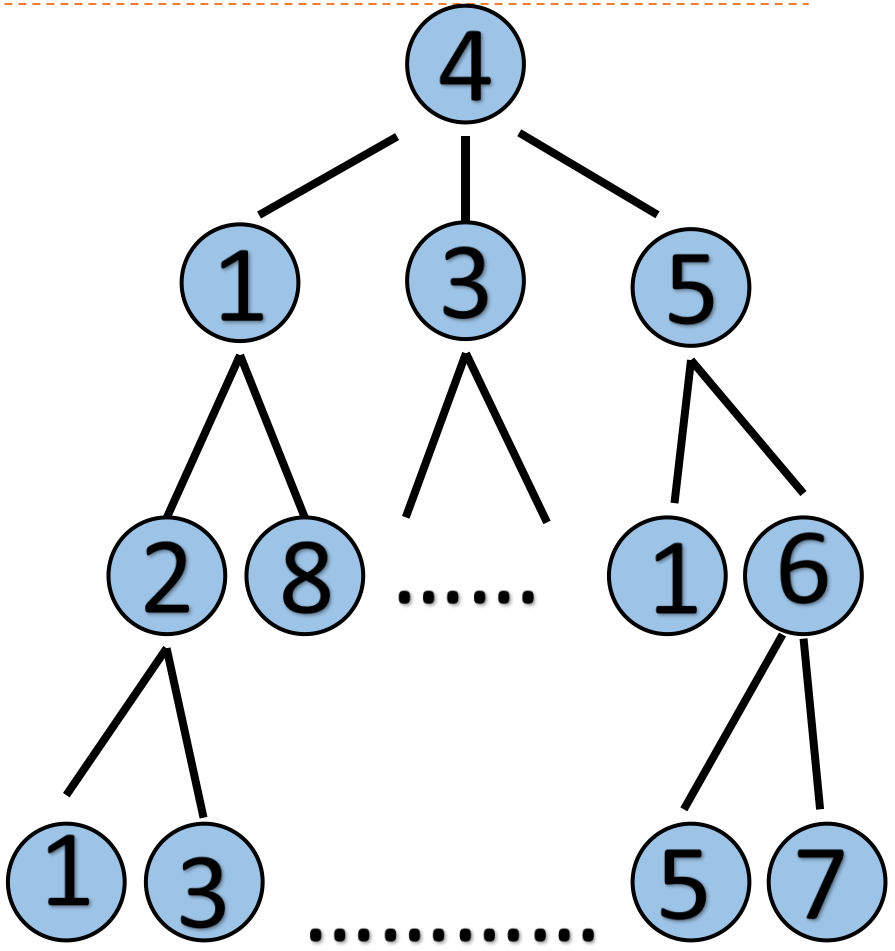


a=4　　L=2

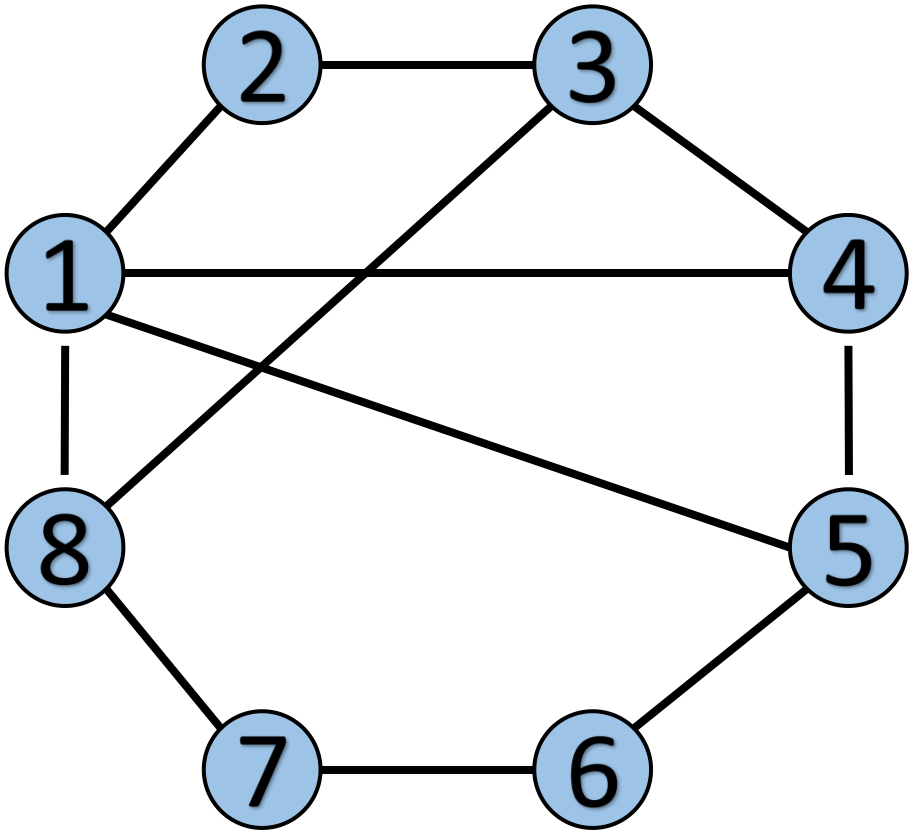
Out：Yes

需要关心全局么?





问题退化为



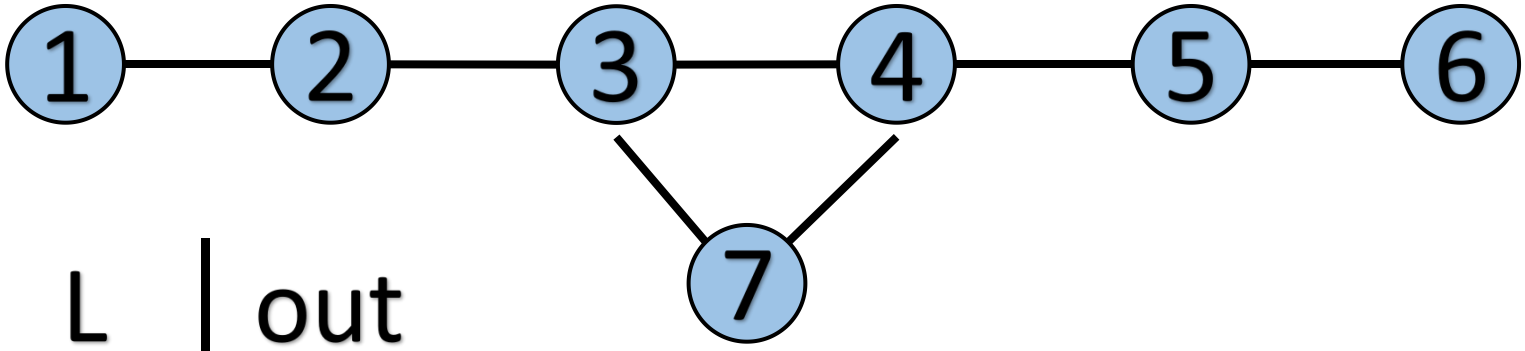
- 从 a点开始；
- 恰好行走 L步；
- 是否能恰好走到 1点。

又等价于____？

等价嘛？

问题解决了么?

dis[]: 0 1 2 3 4 5

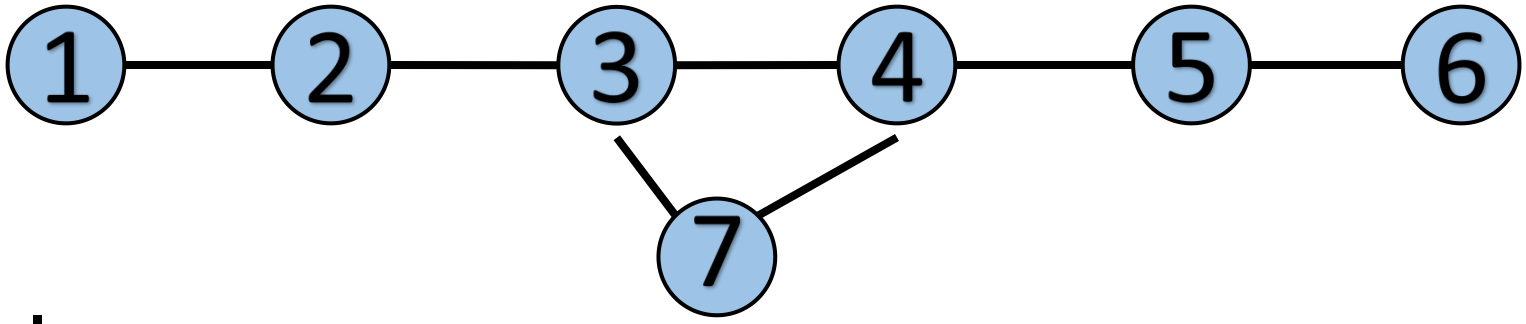


a	L	out
5	3	No
5	8	Yes
5	9	?

兵来将挡，水来土掩

dis[][0]: 0 -6 2 4 4 6

dis[][1]: -7 1 -5 3 5 5



a	L	out
5	9	Yes

代码——BFS部分

由于我们可以在某一条边上玩折返跑，因此只要有从 1 号点出发到 a 距离小于等于 L 且奇偶性与 L 相等的路径
那么就有恰好经过 L 条边抵达 a 号点的方案

定义 d[j][0] 表示经过偶数条边
d[j][1]为经过奇数条边抵达 j 号的最短距离

用 bfs 跑一遍最短路即可，每个点最多入队两次

```cpp
int main(){
    int n,m,q;
    cin>>n>>m>>q;
    for(int i=1;i<=m;i++){
        int a,b;
        scanf("%d %d",&a,&b);
        ve[a].push_back(b);//建图 邻接表加边
        ve[b].push_back(a);
    }
    bfs();
    for(int i=1;i<=q;i++){
        int a,b;
        scanf("%d %d",&a,&b);
        if(b%2==1){
            if(d[a][1]<=b)printf("Yes\n");
            else printf("No\n");
        }
        else if(b%2==0){
            if(d[a][0]<=b)printf("Yes\n");
            else printf("No\n");
        }
    }

    return 0;
}
```

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

const int N=1e5;
const int INF=0x3f3f3f3f;
int d[N][2];
vector<int> ve[N];// 邻接表
```

```
 9 void bfs(){// 预处理（BFS）
10     queue<int>q;
11     q.push(1);
12     memset(d,INF,sizeof d);// 初距离始化为正无穷
13     d[1][0]=0;// 初始化队列起点
14     while(!q.empty()){
15         int u=q.front();
16         q.pop();
17         for(int i=0;i<ve[u].size();i++){//遍历邻接表表邻边
18             int v=ve[u][i];// 取出邻边编号
19             if(d[v][0]>d[u][1]+1||d[v][1]>d[u][0]+1){
20                 if(d[v][0]>d[u][1]+1)// // 如果走一步的长度大于dist[j][type]+
21                     d[v][0]=d[u][1]+1;
22                 if(d[v][1]>d[u][0]+1)/// 注：由于走一步就会变一下奇偶性，
23                     d[v][1]=d[u][0]+1;
24                 q.push(v);//  刷新并将点更新到队列中
25             }
26         }
27     }
28 }
```

2801
2803
2804
2806
2808
2809

3.5.3 [3378]八数码

在3×3的棋盘上，摆有八个棋子，每个棋子上标有1至8的某一数字。棋盘中留有一个空格，空格用0来表示。空格周围的棋子可以移到空格中。要求解的问题是：给出一种初始布局（初始状态）和目标布局（为了使题目简单,设目标状态为123804765），找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。

输入

输入初试状态，一行九个数字，空格用0表示

输出

只有一行，该行只有一个数字，表示从初始状态到目标状态需要的最少移动次数(测试数据中无特殊无法到达目标状态数据)

样例输入

283104765

样例输出

4

3.5.3 [3378]八数码

在3×3的棋盘上，摆有八个棋子，每个棋子上标有1至8的某一数字。棋盘中留有一个空格，空格用0来表示。空格周围的棋子可以移到空格中。要求解的问题是：给出一种初始布局（初始状态）和目标布局（为了使题目简单,设目标状态为123804765），找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。

输入

输入初试状态，一行九个数字，空格用0表示

输出

只有一行，该行只有一个数字，表示从初始状态到目标状态需要的最少移动次数(测试数据中无特殊无法到达目标状态数据)

样例输入

283104765

样例输出

4

3.5.3 [3378]八数码

在3×3的棋盘上，摆有八个棋子，每个棋子上标有1至8的某一数字。棋盘中留有一个空格，空格用0来表示。空格周围的棋子可以移到空格中。要求解的问题是：给出一种初始布局（初始状态）和目标布局（为了使题目简单,设目标状态为123804765），找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。

输入

输入初试状态，一行九个数字，空格用0表示

输出

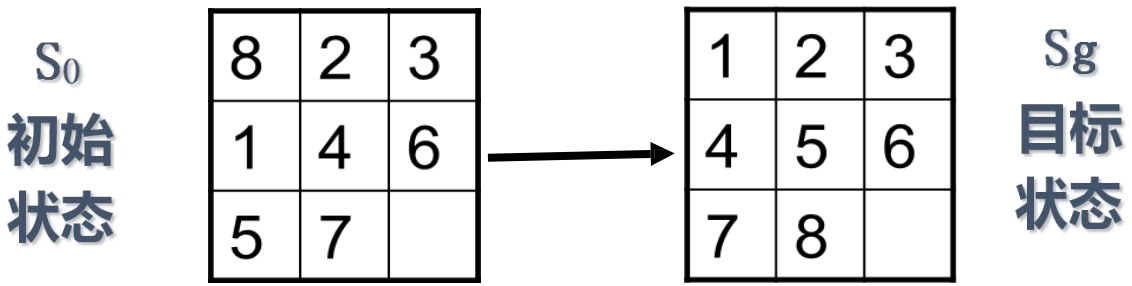
只有一行，该行只有一个数字，表示从初始状态到目标状态需要的最少移动次数(测试数据中无特殊无法到达目标状态数据)

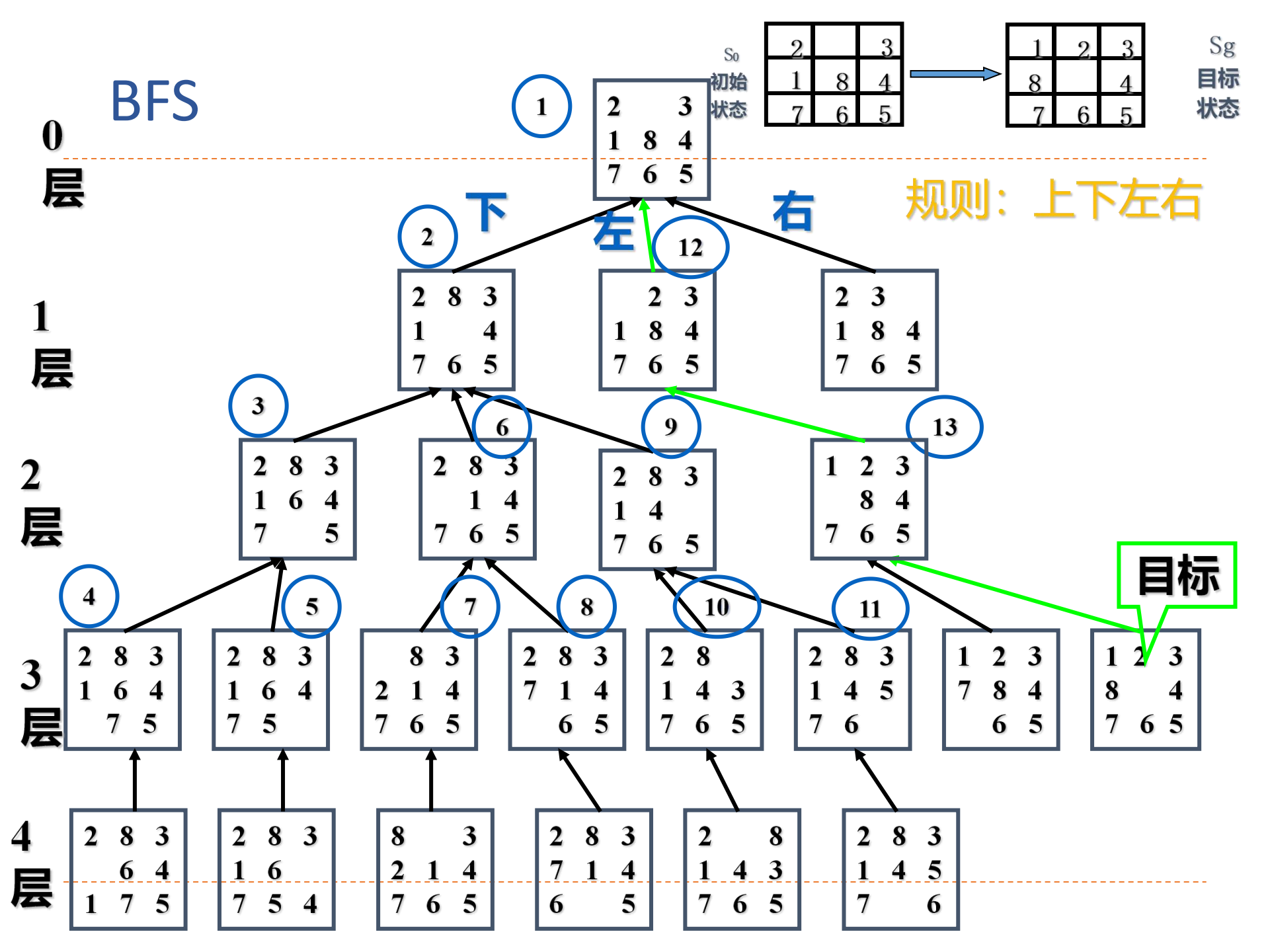
样例输入

283104765

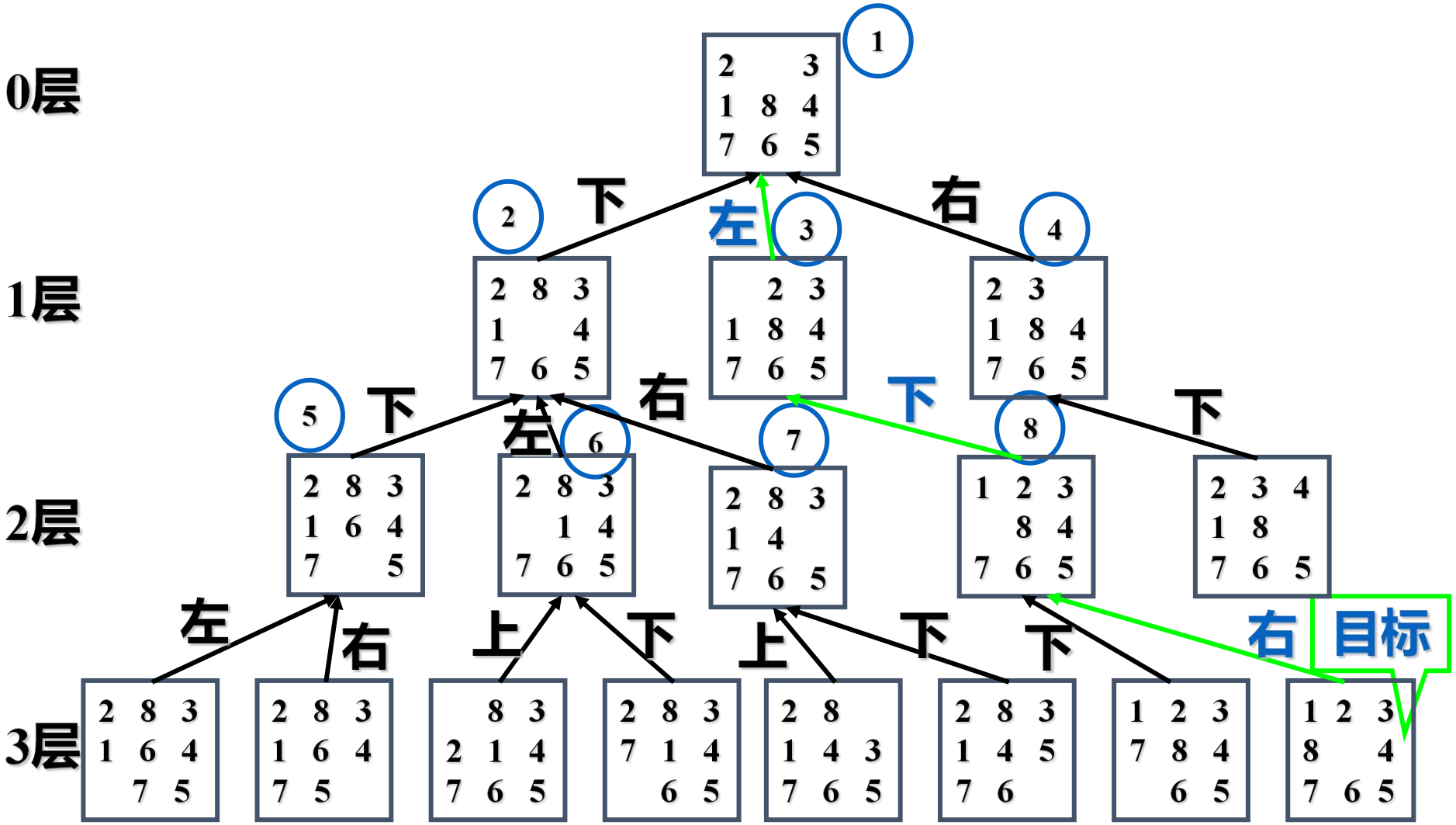
样例输出

4



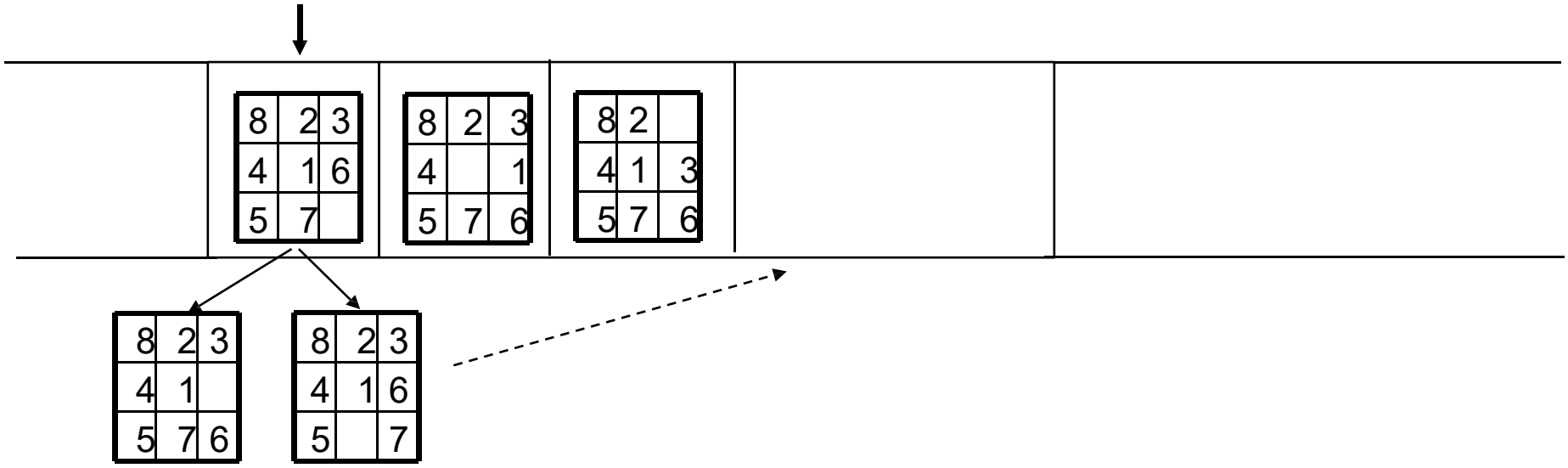

BFS
S0
初始状态
Sg
目标状态
规则：上下左右
下
左
右
目标
0层
1层
2层
3层
4层

BFS


0层
1层
2层
3层
1
2 3
1 8 4
7 6 5
下
左
右
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 3
1 8 4
7 6 5
下
左
右
下
下
5
2 8 3
1 6 4
7 5
6
2 8 3
1 4
7 6 5
7
2 8 3
1 4
7 6 5
8
1 2 3
8 4
7 6 5
2 3 4
1 8
7 6 5
左
右
上
下
上
下
下
右
目标
2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 8
1 4 3
7 6 5
2 8 3
1 4 5
7 6
1 2 3
7 8 4
6 5
1 2 3
8 4
7 6 5

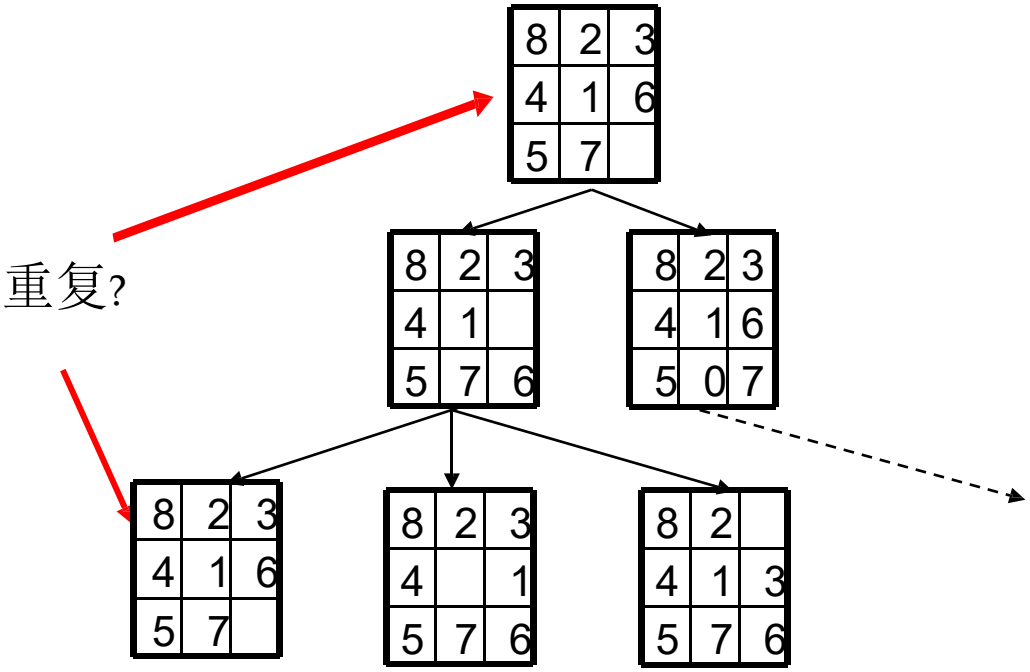
3.5.3 [3378]八数码-分析

- 用队列保存待扩展的节点
- 从队首队取出节点，扩展出的新节点放入队尾， 直到队首出现目标节点（问题的解）



3.5.3 [3378]八数码-分析

关键问题：判重



3.5.3 [3378]八数码-分析

关键问题：判重

- 状态(节点）数目巨大，如何存储？
- 怎样才能较快判断一个状态是否重复？

3.5.3 [3378]八数码-分析

用合理的编码表示“状态”，减小存储代价

- 方案一：

8	2	3
4	1	6
5	7	

每个状态用一个字符串存储, 要9个字节,

太浪费了！！！

3.5.3 [3378]八数码-分析

用合理的编码表示“状态”，减小存储代价

- 方案二：

8	2	3
4	1	6
5	7	

- 每个状态对应于一个9位数，则该9位数最大为876,543,210，小 于2^{31}，则int 就能表示一个状态。
- 判重需要一个标志位序列，每个状态对应于标志位序列中的1位，标志位为0表示该状态尚未扩展，为1则说明已经扩展过了
- 标志位序列可以用字符数组a存放。a的每个元素存放8个状态 的标志位。最多需要876,543,210位，因此a数组需要876,543,210 /8 + 1个元素，即 109,567,902 字节
- 如果某个状态对应于数x ,则其标志位就是a[x/8]的第x%8位
- 空间要求还是太大!!!!

3.5.3 [3378]八数码-分析

用合理的编码表示“状态”，减小存储代价

- 方案三：

8	2	3
4	1	6
5	7	

  - 将每个状态的字符串形式看作一个9位九进制数，则该9位数最大为$876543210_{(9)}$, 即 $381367044_{(10)}$ 需要的标志位数目也降为$381367044_{(10)}$ 比特,即47,670,881字节。
  - 如果某个状态对应于数x ,则其标志位就是a[x/8]的第x%8位
  - 空间要求还是有点大！！！！

用合理的编码表示“状态”，减小存储代价

- 方案三：

8	2	3
4	1	6
5	7	

- 状态数目一共只有9!个，即$362880_{(10)}$个，怎么会需要$876543210_{(9)}$即$381367044_{(10)}$个标志位呢？

用合理的编码表示“状态”，减小存储代价

- 方案三：

8	2	3
4	1	6
5	7	

- 状态数目一共只有9! 个，即$362880_{(10)}$个，怎么会需要 $876543210_{(9)}$ 即 $381367044_{(10)}$ 个标志位呢？
- 如果某个状态对应于数x ,则其标志位就是a[x/8]的第x%8位
- 因为有浪费！例如，$666666666_{(9)}$ 根本不对应于任何状态，也为其准备了标志位！

用合理的编码表示“状态”，减小存储代价

- 方案四：

8	2	3
4	1	6
5	7	

- 把每个状态都看做’0’-’8’的一个排列，以此排列在全部排列中的位置作为其序号。状态用其排列序号来表示
- 012345678是第0个排列，876543210是第9!-1个
- 状态总数即排列总数：9!=362880
- 判重用的标志数组a只需要362,880比特即可。
- 如果某个状态的序号是x ,则其标志位就是a[x/8]的第x%8位

用合理的编码表示“状态”，减小存储代价

- 方案四：

8	2	3
4	1	6
5	7	

- 在进行状态间转移，即一个状态通过某个移动变化到另一个状态时，需要先把int形式的状态（排列序号），转变成字符串形式的状态，然后在字符串形式的状态上进行移动，得到字符串形式的新状态，再把新状态转换成int形式（排列序号）。

用合理的编码表示“状态”，减小存储代价

- 方案四：

8	2	3
4	1	6
5	7	

- 需要编写给定排列（字符串形式）求序号的函数
- 需要编写给定序号，求该序号的排列（字符串形式）的函数

给定排列求序号：

整数 1,2…k的一个排列：
a1 a2 a3 …ak
求其序号
基本思想：算出有多少个排列比给定排列小。
先算1到a1-1放在第1位，会有多少个排列：(a1-1)*((k-1)!)
再算a1不变，1到a2-1 放在第2位(左边出现过的不能再用），会有多少个排列：(a2-1)*((k-2)!)
再算a1，a2不变，1到a3-1 放在第3位，会有多少个排列
….全加起来。 时间复杂度：$O(n^2)$

3241
1,2放在第一位，有 2*3! = 12 种
3在第一位，1放在第2位，有 2! = 2种
32？ 1放在第3位，有 1种
=>前面共 12+2+1 = 15种。所以 3241是第16个排列

给定序号n求排列：

1234的排列的第9号

第一位假定是1，共有3!种，没有到达9, 所以第一位至少是2

第一位是2, 一共能数到 3!+3!号，>= 9, 所以第一位是2

第二位是1，21??, 一共能数到 3!+2! = 8 不到9，所以第二位至少是 3

第二位是3，23??, 一共能数到 3!+2!+2! >= 9, 因此第二位是3

第三位是1，一共能数到3!+2!+1 = 9, 所以第三位是1，第四位是 4

答案：2314

时间复杂度：$O(n^2)$

- 时间与空间的权衡
  - 对于状态数较小的问题，可以用最直接的方式编码以空间换时间
  - 对于状态数太大的问题，需要利用好的编码方法以时间换空间
  - 具体问题具体分析

今天的课程结束啦……

下课了…
同学们再见！