广搜

Breadth First Search(BFS)广度优先算法

抓住那头牛(百练习4001)

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

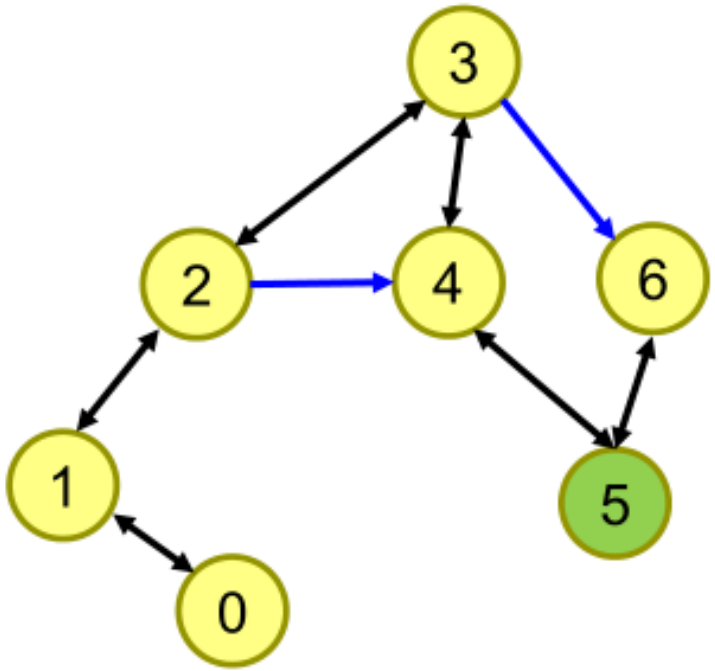
假设牛没有意识到农夫的行动，站在原地不动。农夫最少要花多少时间才能抓住牛？

分析

假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



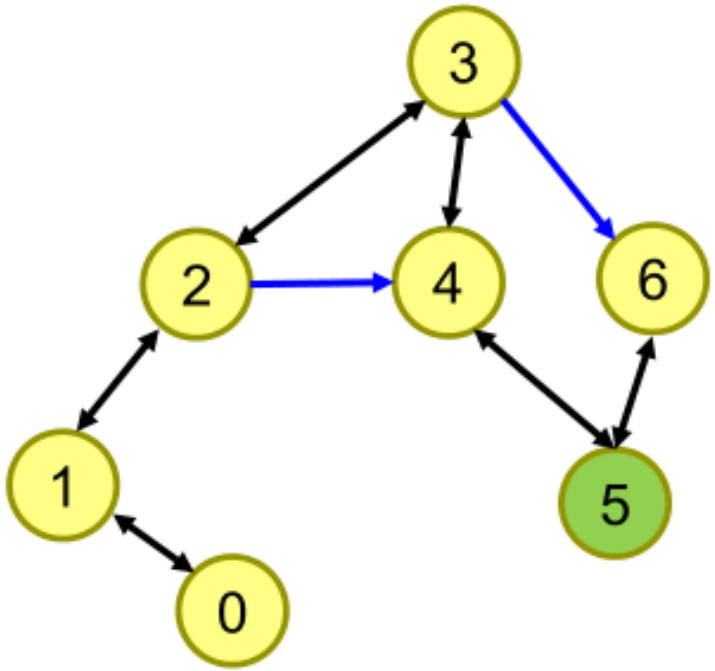
策略1）深度优先搜索：从起点出发，随机挑一个方向，能往前走就往前走(扩展），走不动了则回溯。**不能走已经走过的点(要判重）**。

分析

假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



运气好的话：
3->4->5　或　3->6->5
问题解决！

运气不太好的话：
3->2->4->5
运气最坏的话：
3->2->1->0->4->5

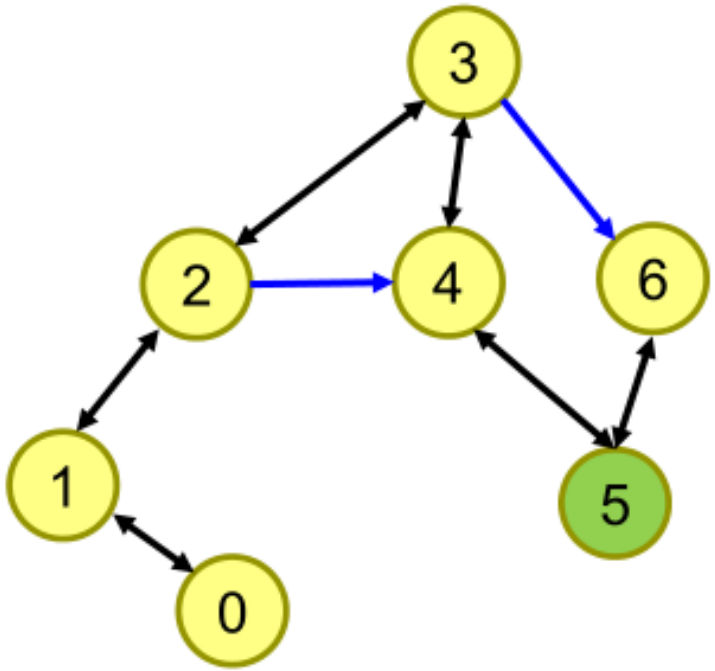
要想求最优(短）解，则要遍历所有走法。可以用各种手段优化，比如，若已经找到路径长度为n的解，则**所有长度大于n的走法就不必尝试。**
运算过程中需要存储路径上的节点，数量较少。
用栈存节点。

分析

假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



策略2）广度优先搜索：

给节点分层。起点是第0层。从起点最少需n步就能到达的点属于第n层。

第1层：2,4,6
第2层：1,5
第3层：0

依层次顺序，从小到大扩展节点。

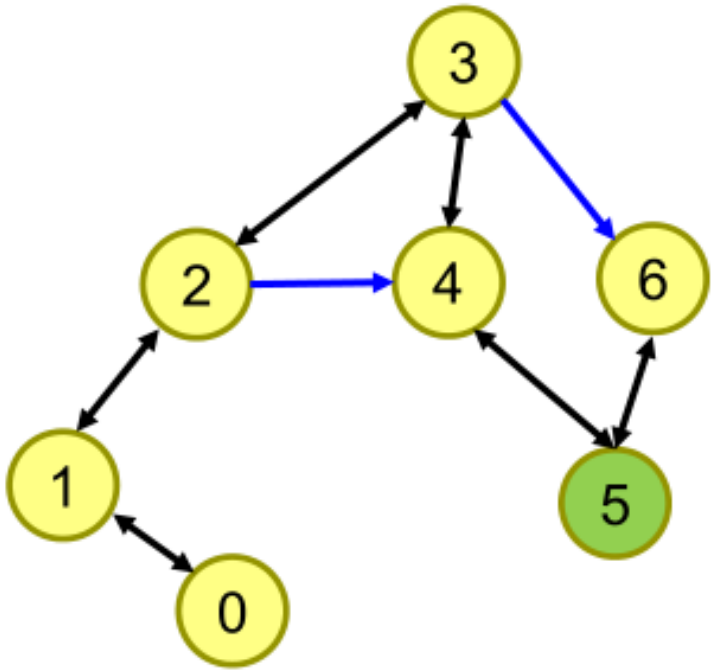
把层次低的点全部扩展出来后，才会扩展层次高的点。

分析

假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



策略2）广度优先搜索：

给节点分层。起点是第0层。从起点最少需n步就能到达的点属于第n层。

扩展时，不能扩展出已经走过的节点(要判重)。

可确保找到最优解，但是因扩展出来的节点较多，且多数节点都需要保存，因此需要的存储空间较大。

用队列存节点。

广度优先搜索算法

广度优先搜索算法如下：（用QUEUE）

(1) 把初始节点S0放入Open表中；

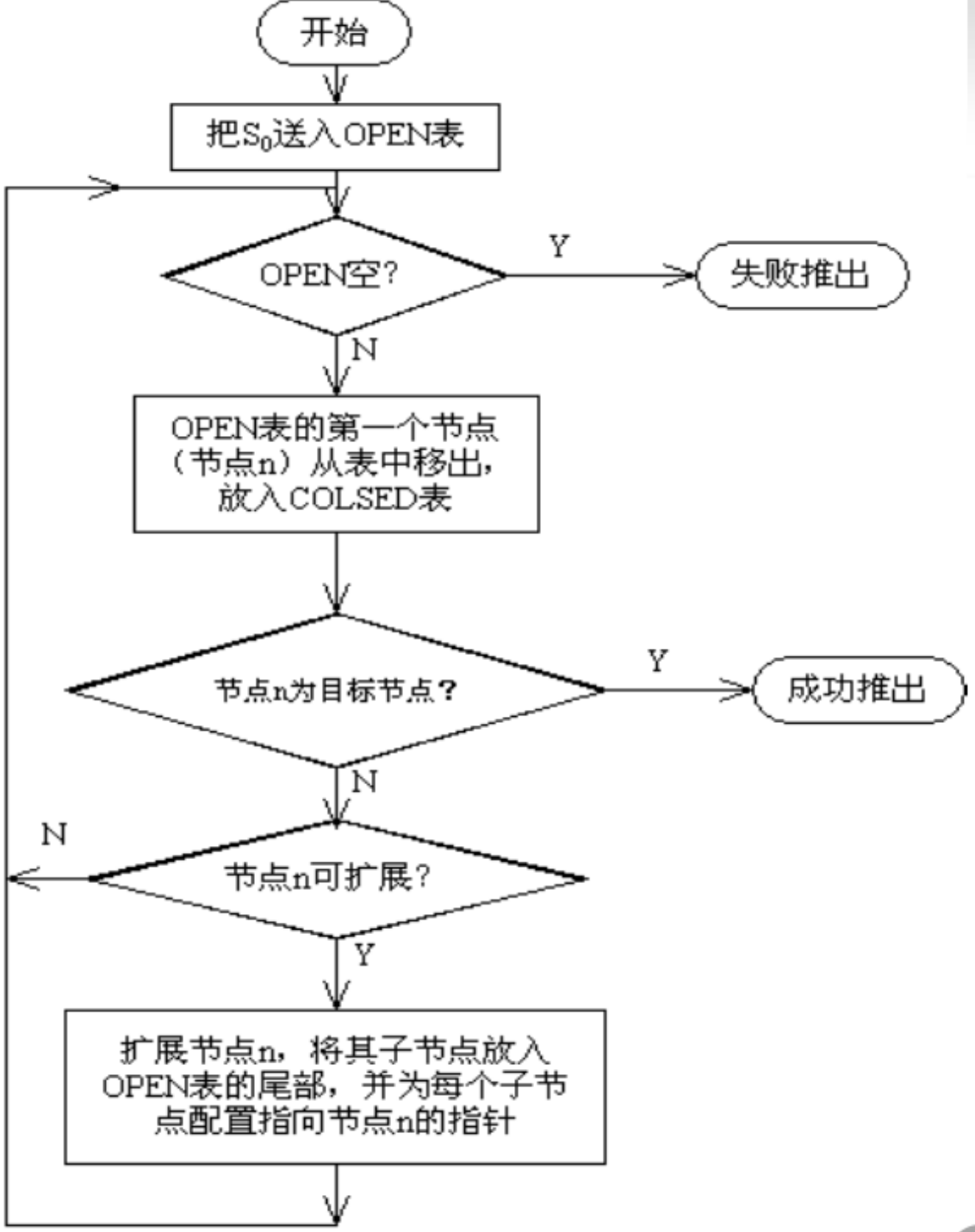
(2) 如果Open表为空，则问题无解，失败退出；

(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；

(5) 若节点n不可扩展，则转第(2)步；

(6) 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。

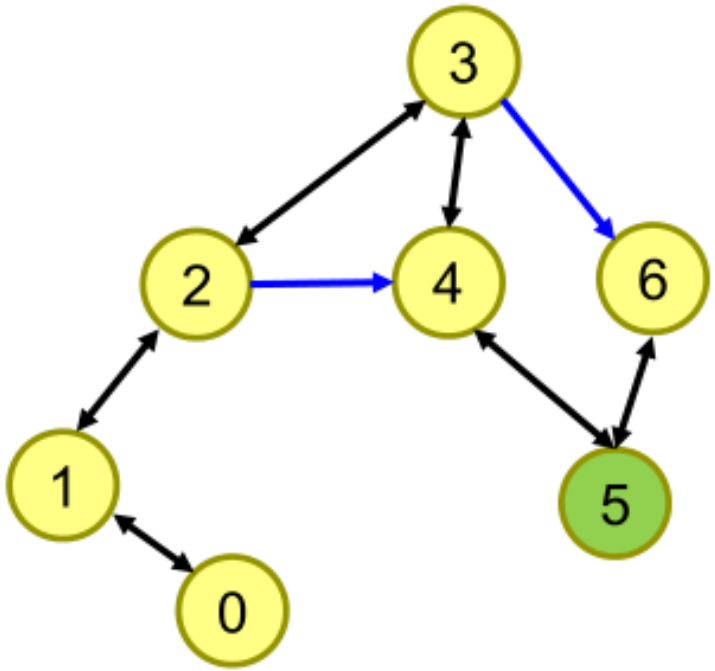


广度优先搜索算法

假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索的代码框架

```
BFS() {
        初始化队列
        while(队列不为空且未找到目标节点) {
                取队首节点扩展，并将扩展出的非重复节点放入队尾；
                必要时要记住每个节点的父节点；
        }
}
```

广度优先算法描述2

```
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；
do
 {
   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；
         }
     }
}while(head<tail);                          //队列为空
}
```

```
#include <iostream>
#include <cstring>
#include <queue>
using namespace std;
int N,K;
const int MAXN = 100000;
bool visited[MAXN+10];
//判重标记,visited[i] = true表示i已经扩展过
struct Step {
        int x; //位置
        int steps; //到达x所需的步数
        Step(int xx,int s){x = xx;steps = s;}
};
queue<Step> q; //队列,即Open表
int main() {
        cin >> N >> K;
        memset(visited,0,sizeof(visited));
        q.push(Step(N,0));
        visited[N] = 1;
        while(!q.empty()) {
                Step s = q.front();
                if( s.x == K ) { //找到目标
                        cout << s.steps <<endl;
                        return 0;
                } else {if( s.x - 1 >= 0 && !visited[s.x-1] ) {
                                q.push(Step(s.x-1,s.steps+1));
                                visited[s.x-1] = 1;}
                        if( s.x + 1 <= MAXN && !visited[s.x+1] ) {
                                q.push(Step(s.x+1,s.steps+1));
                                visited[s.x+1] = 1;
                        }
                        if( s.x * 2 <= MAXN &&!visited[s.x*2] ) {
                                q.push(Step(s.x*2,s.steps+1));
                                visited[s.x*2] = 1;
                        }
                        q.pop();
        }}
        return 0;
}
```

鸣人和佐助(百练6044)

已知一张地图（以二维矩阵的形式表示）以及佐助+和鸣人@的位置。地图上的每个位置都可以走到，只不过有些位置上有大蛇丸的手下(#)，需要先打败大蛇丸的手下才能到这些位置。

鸣人有一定数量的查克拉，每一个单位的查克拉可以打败一个大蛇丸的手下。假设鸣人可以往上下左右四个方向移动，每移动一个距离需要花费1个单位时间，打败大蛇丸的手下不需要时间。如果鸣人查克拉消耗完了，则只可以走到没有大蛇丸手下的位置，不可以再移动到有大蛇丸手下的位置。

佐助在此期间不移动，大蛇丸的手下也不移动。请问，鸣人要追上佐助最少需要花费多少时间？

```
4 4 1
#@##
**##
###+
****
```

分析

状态定义为：

（r,c,k)，鸣人所在的行，列和查克拉数量

如果队头节点扩展出来的节点是有大蛇手下的节点，则其 k 值比队头的k要减掉 1。如果队头节点的查克拉数量为0,则不能扩展出有大蛇手下的节点。

```
4 4 1
#@##
**##
###+
****
```

求钥匙的鸣人

不再有大蛇丸的手下。

但是佐助被关在一个格子里，需要集齐k种钥匙才能打开格子里的门门救出他。

K种钥匙散落在迷宫里。有的格子里放有一把钥匙。一个格子最多放一把钥匙。走到放钥匙的格子，即得到钥匙。

鸣人最少要走多少步才能救出佐助。

求钥匙的鸣人

状态：

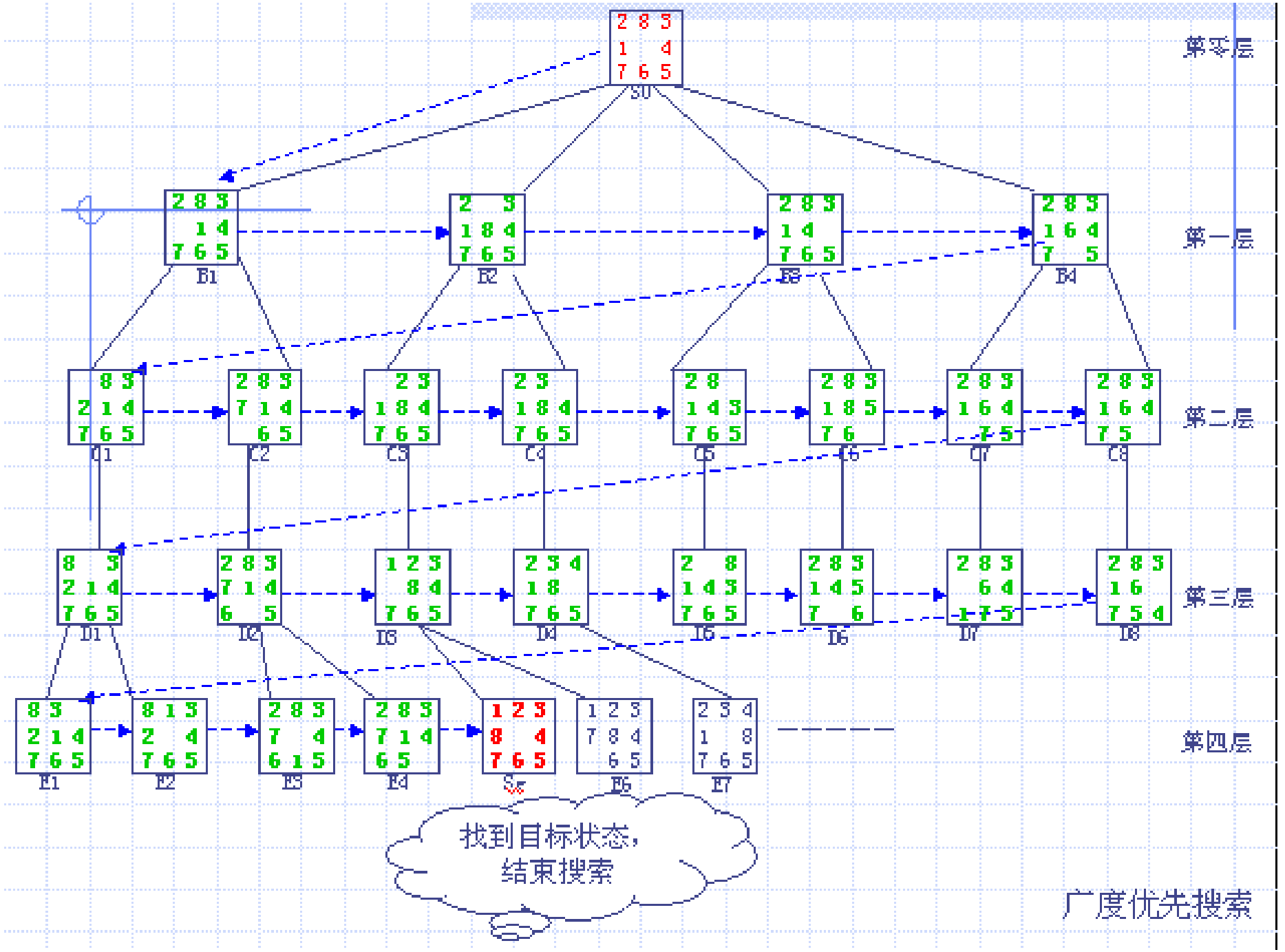
（r,c,keys)： 鸣人的行，列，已经拥有的钥匙种数

目标状态 (x,y,K) (x,y)是佐助呆的地方

如果队头节点扩展出来的节点上面有不曾拥有的某种钥匙，则该节点的 keys比队头节点的 keys要加1

广度优先搜索

在深度优先搜索算法中，是深度越大的结点越先得到扩展。如果在搜索中把算法改为按结点的层次进行搜索，本层的结点没有搜索处理完时，不能对下层结点进行处理，即深度越小的结点越先得到扩展，也就是说先产生的结点先得以扩展处理，这种搜索算法称为广度优先搜索法。


第零层
第一层
第二层
第三层
第四层
找到目标状态，
结束搜索
广度优先搜索

迷宫

如下图所示，给出一个N*M的迷宫图和一个入口、一个出口。

编一个程序，打印一条从迷宫入口到出口的路径。这里黑色方块的单元表示走不通（用-1表示），白色方块的单元表示可以走（用0表示）。只能往上、下、左、右四个方向走。如果无路则输出“no way.”。

输入1：	输出1：	输入2：	输出2：
8 5 -1 -1 -1 -1 -1 0 0 0 0 -1 -1 -1 -1 0 -1 -1 0 0 0 -1 -1 0 0 -1 -1 -1 0 0 0 -1 -1 -1 -1 0 -1 -1 0 0 0 -1 2 1 8 4	2,1 2,2 2,3 2,4 3,4 4,4 4,3 5,3 6,3	8 5 -1 -1 -1 -1 -1 0 0 0 0 -1 -1 -1 -1 0 -1 -1 0 0 0 -1 -1 0 0 -1 -1 -1 0 0 0 -1 -1 -1 -1 -1 -1 -1 0 0 0 -1 2 1 8 4	no way.

入口→	0	-1	0	0	0	0	0	0	-1	
	0	0	0	0	-1	0	0	0	-1	
	-1	0	0	0	0	0	-1	-1	-1	
	0	0	-1	-1	0	0	0	0	0	→出口
	0	0	0	0	0	0	0	-1	-1	

分析

只要输出一条路径即可，所以是一个经典的回溯算法问题，本例给出了回溯（深搜）程序和广搜程序。实现见参考程序。

入口→	0	-1	0	0	0	0	0	0	-1	
	0	0	0	0	-1	0	0	0	-1	
	-1	0	0	0	0	0	-1	-1	-1	
	0	0	-1	-1	0	0	0	0	0	→出口
	0	0	0	0	0	0	0	-1	-1	

【深搜参考程序】

```
#include <iostream>
using namespace std;
int n,m,desx,desy,soux,souy,totstep,a[51],b[51],map[51][51];
bool f;
int move(int x, int y,int step)
{
 map[x][y]=step;                    //走一步，作标记，把步数记下来
 a[step]=x;  b[step]=y;             //记路径
 if ((x==desx)&&(y==desy))
 {
  f=1;
  totstep=step;
 }
  else
  {
    if ((y!=m)&&(map[x][y+1]==0))  move(x,y+1,step+1);       //向右
    if ((!f)&&(x!=n)&&(map[x+1][y]==0))  move(x+1,y,step+1); //往下
    if ((!f)&&(y!=1)&&(map[x][y-1]==0))  move(x,y-1,step+1);  //往左
    if ((!f)&&(x!=1)&&(map[x-1][y]==0))  move(x-1,y,step+1);  //往上
  }
}
```

```
int main()
{
  int i,j;
  cin>>n>>m;                          //n行m列的迷宫
  for (i=1;i<=n;i++)                  //读入迷宫，0表示通，-1表示不通
   for (j=1;j<=m;j++)
    cin>>map[i][j];
  cout<<"input the enter:";
  cin>>soux>>souy;                    //入口
  cout<<"input the exit:";
  cin>>desx>>desy;                    //出口
  f=0;                                //f=0表示无解；f=1表示找到了一个解
  move(soux,souy,1);
  if (f)
  {
    for (i=1;i<=totstep;i++)          //输出直迷宫的路径
     cout<<a[i]<<","<<b[i]<<endl;
  }
else  cout<<"no way."<<endl;
return 0;
}
```

【广搜参考程序】

```
#include <iostream>
using namespace std;
int u[5]={0,0,1,0,-1},
    w[5]={0,1,0,-1,0};
int n,m,i,j,desx,desy,soux,souy,head,tail,x,y,a[51],b[51],pre[51],map[51][51];
bool f;
int print(int d)
{
  if (pre[d]!=0) print (pre[d]);          //递归输出路径
  cout<<a[d]<<","<<b[d]<<endl;
}
int main()
{
  int i,j;
  cin>>n>>m;                              //n行m列的迷宫
  for (i=1;i<=n;i++)                      //读入迷宫，0表示通，-1表示不通
   for (j=1;j<=m;j++)
    cin>>map[i][j];
  cout<<"input the enter:";
  cin>>soux>>souy;                        //入口
  cout<<"input the exit:";
  cin>>desx>>desy;                        //出口
  head=0;
  tail=1;
```

```
 f=0;
  map[soux][souy]=-1;
  a[tail]=soux;  b[tail]=souy; pre[tail]=0;
  while (head!=tail)                          //队列不为空
  {
    head++;
    for (i=1;i<=4;i++)                        //4个方向
    {
      x=a[head]+u[i];  y=b[head]+w[i];
      if ((x>0)&&(x<=n)&&(y>0)&&(y<=m)&&(map[x][y]==0))
      {                                       //本方向上可以走
        tail++;
        a[tail]=x;  b[tail]=y;  pre[tail]=head;
        map[x][y]=-1;
        if ((x==desx)&&(y==desy))     //扩展出的结点为目标结点
        {
          f=1;
          print(tail);
          break;
        }
      }
    }
    if (f) break;
  }
  if (!f) cout<<"no way."<<endl;
  return 0;
}
```

最近练习

栈： 计算后缀表达式；中缀表达式转后缀表达式。
队列：舞伴 ，blah 数集，连通块（DFS、BFS）
深搜：数的划分，数的拆分，跳马，八皇后，迷宫（方案数、路径）
广搜：抓住那头牛，迷宫（最短路径、路径长度）

有空的同学：
深搜：单词接龙P1019
广搜：填涂颜色P1162