队列与BFS

信智学院　陈琰宏

主要内容

01 队列的定义

02 队列的操作（增删改查等）

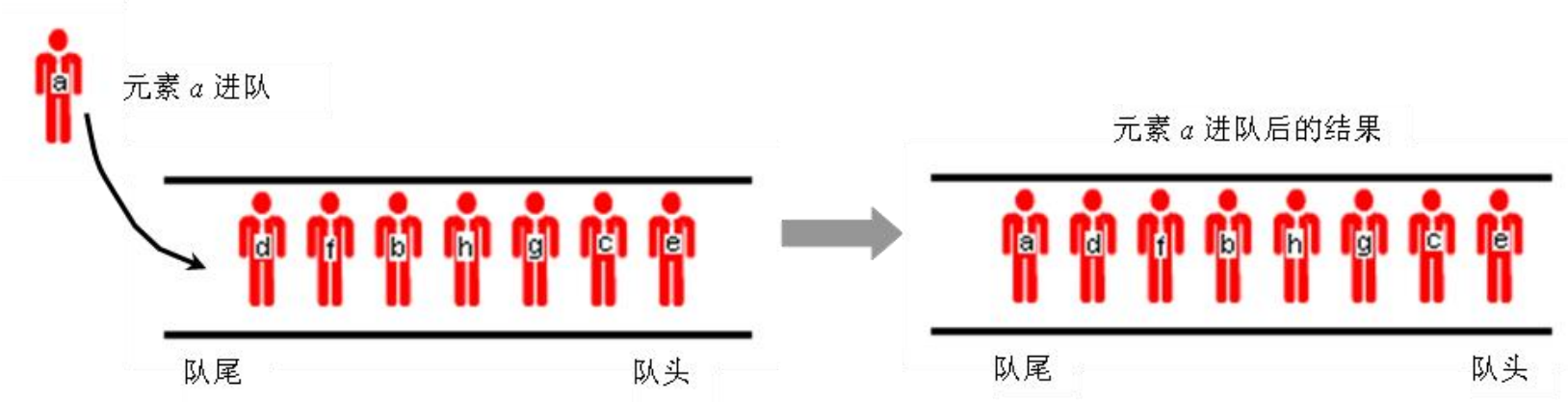
03 链式队列

04 队列的应用

05 广度优先搜索

3.1 队列的定义

队列（简称为队）是一种操作受限的线性表，其限制为仅允许在表的一端进行插入，而在表的另一端进行删除。把进行插入的一端称做队尾（rear），进行删除的一端称做队头或队首（front）。每个元素必然按照进入的次序出队，所以又把队列称为先进先出表。



3.1 队列的定义-基本操作

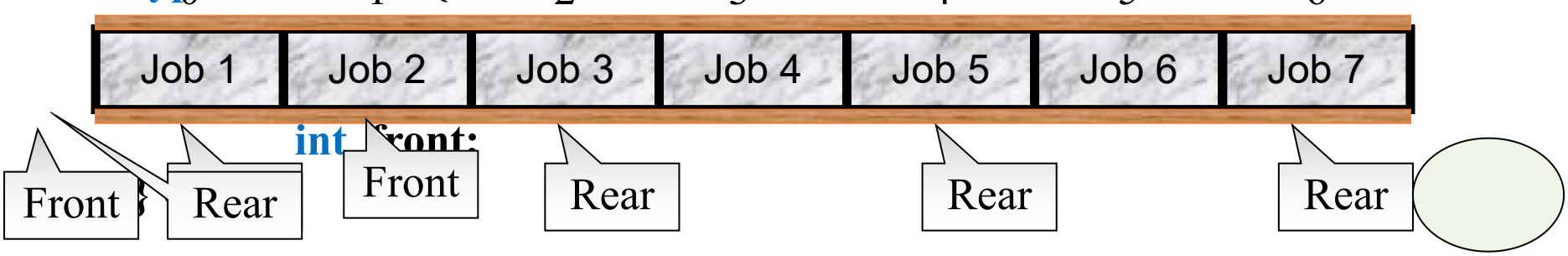
1、Queue CreatQueue(int MaxSize)：生成长度为MaxSize的空队列。
2、int IsFullQ(Queue Q, int MaxSize)：
判断队列Q是否已满，若是返回1(TRUE)；否则返回0(FALSE)。
3、void AddQ(Queue Q, ElementType item)：
若队列已经满了，返回已满信息；否则将数据元素item插入到队列Q中去。
4、int IsEmptyQ(Queue Q)：
判断队列Q是否为空，若是返回1(TRUE)；否则返回0(FALSE)。
5、ElementType DeleteQ(Queue Q)：
若队列为空信息，返回队列空信息；否则将队头数据元素从队列中删除并返回。
6、 ElementType FrontQ(Queue Q)：……

3.1 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。

【例】

```
#define MAXQSIZE 100  //最大队列储存数据元素的最大个数>
typedef struct {
    int front;
    int rear;
}
```



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		

3.1 队列的定义

- 初始时置front=rear=-1；
- 队空的条件为front==rear；
- 队满的条件为rear==MaxSize-1（rear指向数组最大下标时为队满）；
- 元素*e*进队的操作是先将队尾指针rear增1，然后将*e*放在队尾处；
- 出队操作是先将队头指针front增1，然后取出队头处的元素。

3.1 队列的定义-循环队列

在非循环队列中，元素进队时队尾指针rear增1，元素出队时队头指针front增1，当进队MaxSize个元素后，满足队满的条件即rear==MaxSize-1成立，此时即使出队若干元素，队满条件仍成立（实际上队列中有空位置），这是一种**假溢出**。

为了能够充分地使用数组中的存储空间，把数组的前端和后端连接起来，形成一个循环的顺序表，即把存储队列元素的表从逻辑上看成一个环，称为循环队列（也称为环形队列）。

3.1 队列的定义-循环队列

AddQ Job 1

AddQ Job 2

AddQ Job 3

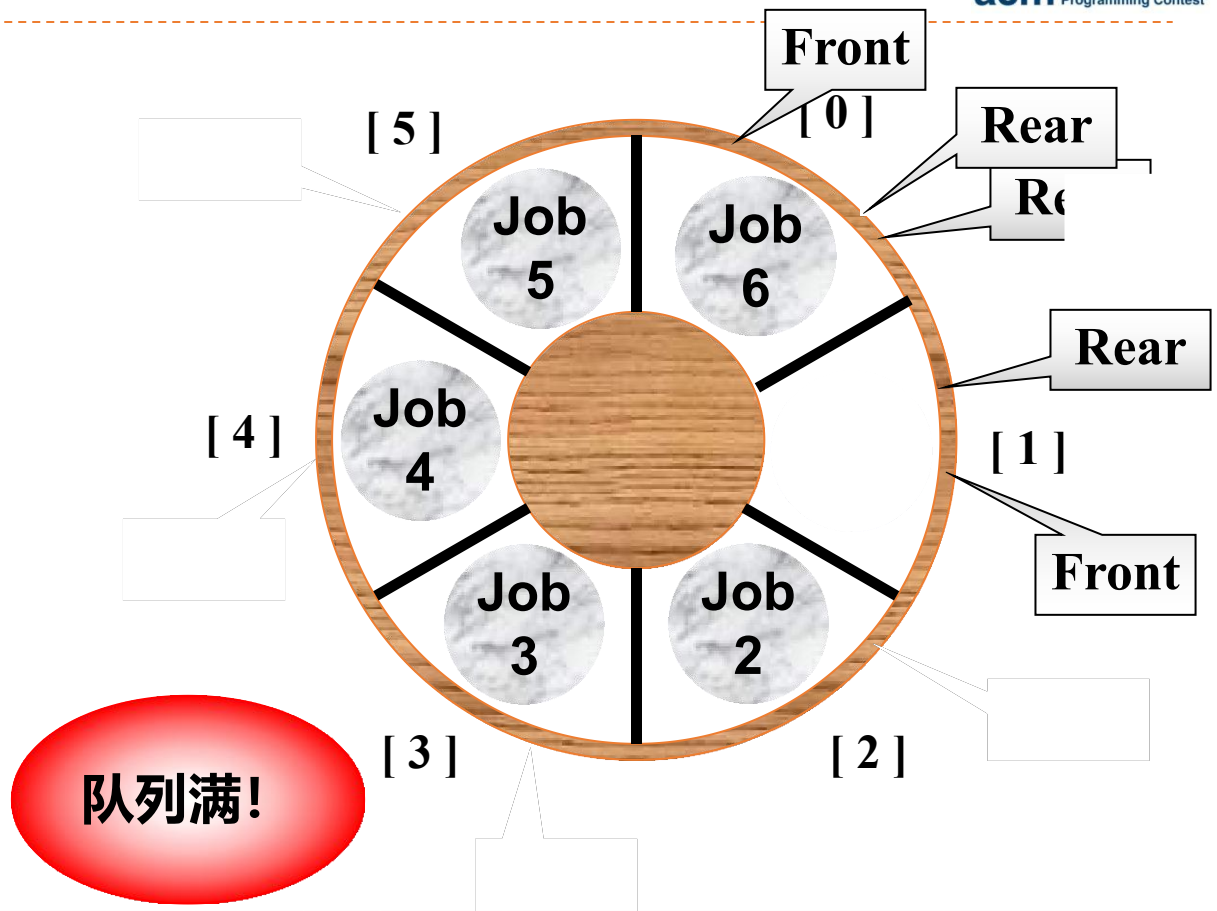
DeleteQ Job 1

AddQ Job 4

AddQ Job 5

AddQ Job 6

AddQ Job 7

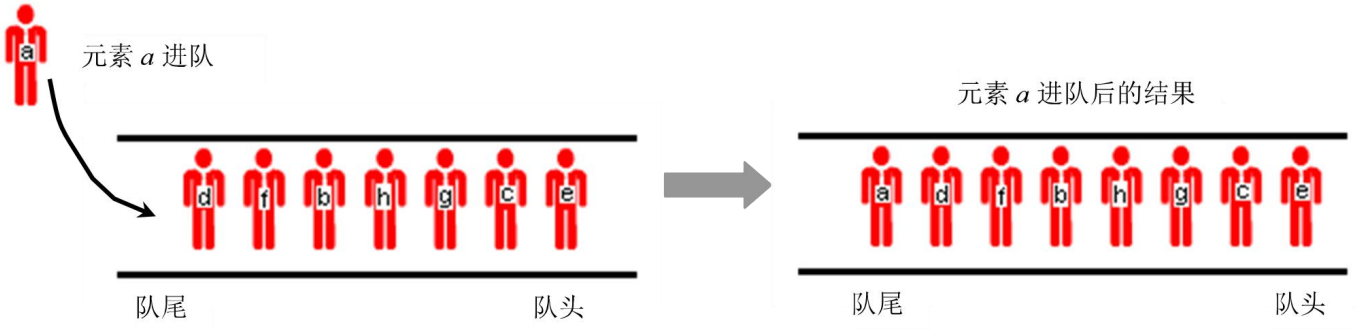


注：如果数据结构中增加一个 Size 域，用来区分队列“空”和“满”的话， 可以“节省”一个数据元素的存储单元。但是会带来算法描述的复杂性。

3.2 队列的操作-入队1

在进队运算中，当队列不满的条件下，先将队尾指针rear增1，然后元素*e*放到该位置处。对应的算法如下：

```
bool enQueue(string e)
{     if (rear==MaxSize-1)                    //队满上溢出
          return false;                       //返回false
      rear++;
      data[rear]=e;
      return true;
}
```



3.2 队列的操作-入队2

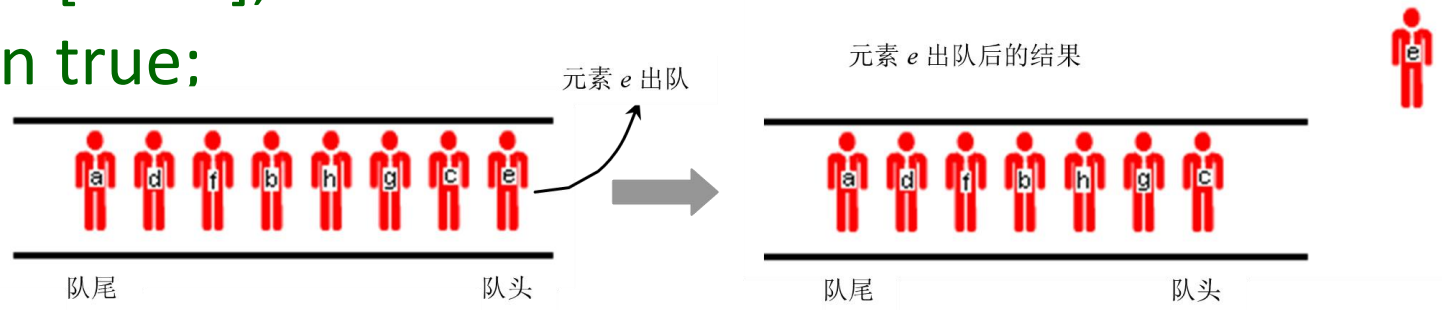
```
void AddQ( Queue *PtrQ, ElementType item)
{
    if ( (PtrQ->rear+1) % MaxSize == PtrQ->front ) {
        printf("队列满");
        return;
    }
    PtrQ->rear = (PtrQ->rear+1)% MaxSize;
    PtrQ->Data[PtrQ->rear] = item;
}
```

Front和rear指针的移动采用“加1取余”法，体现了顺序存储的“循环使用”。

3.2 队列的操作-出队1

在出队运算中，当队列不为空的条件下，将队首指针front增1，并将该位置的元素值赋给e。对应的算法如下：

```
public bool deQueue(ref string e)
{     if (front==rear)                          //队空下溢出
          return false;                    //返回false
       front++;
       e=data[front];
       return true;
}
```



3.2 队列的操作-出队2

```
ElementType DeleteQ ( Queue *PtrQ )
{
     if ( PtrQ->front == PtrQ->rear ) {
          printf("队列空");
          return ERROR;
     } else  {
          PtrQ->front = (PtrQ->front+1)% MaxSize;
          return  PtrQ->Data[PtrQ->front];
     }
}
```

3.2 队列的操作-判断为空

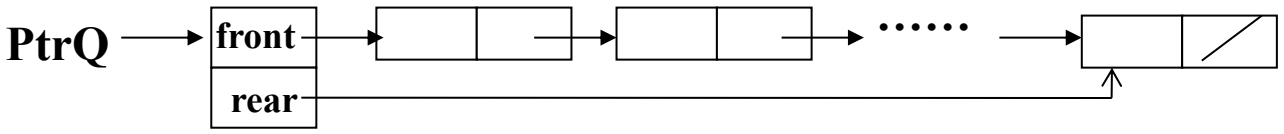
若队列满足front==rear条件，则返回true；否则返回false。对应的算法如下：

```
bool QueueEmpty()
{
    return(front==rear);
}
```

3.3 队列的链式存储实现

栈的链式存储结构实际上就是一个单链表，叫做链栈。插入和删除操作只能在链栈的栈顶进行；栈顶指针Top就是链表的头指针。

```
typedef struct Node{
    ElementType  Data;
    struct Node  *Next;
}QNode;
typedef  struct {        /* 链队列结构  */
   QNode  *rear;        /* 指向队尾结点 */
   QNode  *front;       /* 指向队头结点 */
} LinkQueue;
LinkQueue  *PtrQ ;
```



3.3 队列的链式存储实现

❖不带头结点的链式队列出队操作的一个示例：

```
ElementType DeleteQ ( LinkQueue  *PtrQ )
{   Qnode  *FrontCell;
    ElementType FrontElem;

    if  ( PtrQ->front == NULL) {
        printf("队列空");    return ERROR;
    }
    FrontCell = PtrQ->front;
    if ( PtrQ->front == PtrQ->rear)     /* 若队列只有一个元素 */
        PtrQ->front = PtrQ->rear = NULL;  /* 删除后队列置为空 */
    else
        PtrQ->front = PtrQ->front->Next;
    FrontElem = FrontCell->Data;
    free( FrontCell );                  /* 释放被删除结点空间  */
    return  FrontElem;
}
```

请学生完成入队操作的算法程序。

3.4.1 [7290] 队列的应用

1 [7290] 模拟队列

实现一个队列，队列初始为空，支持四种操作：

(1) “push x” – 向队尾插入一个数x;

(2) “pop” – 从队头弹出一个数;

(3) “empty” – 判断队列是否为空;

(4) “query” – 查询队头元素。

现在要对队列进行M个操作，其中的每个操作3和操作4都要输出相应的结果。

输入格式 第一行包含整数M，表示操作次数。 接下来M行，每行包含一个操作命令，操作命令为”push x”，”pop”，”empty”，”query”中的一种。

输出格式 对于每个”empty”和”query”操作都要输出一个查询结果，每个结果占一行。 其中，”empty”操作的查询结果为“YES”或“NO”，”query”操作的查询结果为一个整数，表示队头元素的值。

3.4.1 [7290] 队列的应用

样例输入

```
10
push 6
empty
query
pop
empty
push 3
push 4
pop
query
push 6
```

样例输出

```
NO
6
YES
4
```

3.4.1 [7290] –方法1

```
#include<iostream>
#include<cstring>
using namespace std;
int n,k,q=1,h=0,a[1000001];
string str;
void push(int x){
    h++;
    a[h]=x;
}
void pop(){
    q++;
}
bool empty(){
    if(q>h) return true;
    return false;
}
int top(){
    return a[q];
}
```

```
int main(){
    cin>>n;
    while(n--){
        cin>>str;
        if(str=="push"){
            cin>>k;
            push(k);
        }
        else if(str=="empty"){
            if(!empty()) cout<<"NO"<<endl;
            else cout<<"YES"<<endl;
        }
        else if(str=="pop") pop();
        else cout<<top()<<endl;
    }
    return 0;
}
```

3.4.1 [7290] –方法1

```
using namespace std;
int n,k,q=1,h=0,a[1000001];
string l;
void push(int x){
    _______1_______
}
void pop(){
    ______2_________
}
bool empty(){
    if(_______3_______) return true;
    return false;
}
int top(){
    return a[q];
}
```

```
21 int main(){
22     cin>>n;
23     while(n--){
24         cin>>str;
25         if(str=="push"){
26             cin>>k;
27             push(k);
28         }
29         else if(str=="empty"){
30             if(!empty()) cout<<"NO"<<endl;
31             else cout<<"YES"<<endl;
32         }
33         else if(str=="pop") pop();
34         else cout<<top()<<endl;
35     }
36     return 0;
37 }
```

3.4.1 [7290] 方法2

```
#include<iostream>
using namespace std;
int x,m;
string str;
struct queue{
    int que[100005],h,t;
    queue() { h=1 , t=0; }
    void push(int x) { que[++t] = x; }
    void pop() { h++; }
    bool empty() {  return t<h ? 1 : 0; }
    int front() { return que[h]; }
};
struct queue q;
int main(){
    scanf("%d",&m);
    while(m--) {
        cin>>str;
        if(str=="push") { cin>>x; q.push(x);}
        else if(str=="pop") q.pop();
        else if(str=="empty") q.empty() ? cout<<"YES\n" : cout<<"NO\n";
        else cout<<q.front()<<endl;
    }
    return 0;
}
```

3.4.1 [7290] 方法3

```cpp
#include<iostream>
using namespace std;
const int N=100010;
int q[N],hh=0,tt=-1;
int m;

int main(){
  cin>>m;
  while(m--){
      string op;
      int x;
      cin>>op;
      if(op=="push")
      {
          cin>>x;
          q[++tt]=x;
      }
      else if(op=="pop")hh++;
      else if(op=="empty")cout<<(hh>tt?"YES":"NO")<<endl;
      else cout<<q[hh]<<endl;
  }
  return 0;
}
```

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

3.4.1 [7290] 方法4

```cpp
#include <iostream>
#include <queue>
using namespace std;
int main(){
    queue <int> q;
    int m;
    cin >> m;
    while(m --){
        string op;
        int x;
        cin >> op;
        if(op =="push"){
            cin >> x;
            q.push(x);//入队
        }
        else if(op =="pop"){
            q.pop();//出队
        }
        else if(op =="query"){
            cout << q.front() << endl;//输出队首元素
        }
        else{
            if(q.empty())   cout << "YES" << endl;//队判空
            else cout << "NO" << endl;
        }
    }
    return 0;
}
```

3.4.2 [2882] 周末舞会

假设在周末舞会上，男士们和女士们进入舞厅时，各自排成一队。跳舞开始时，依次从男队和女队的队头上各出一人配成舞伴。规定每个舞曲能有一对跳舞者。若两队初始人数不相同，则较长的那一队中未配对者等待下一轮舞曲。现要求写一个程序，模拟上述舞伴配对问题。

输入

第一行两队的人数；

第二行舞曲的数目。

输出

配对情况。

样例输入

```
4 6
7
```

样例输出

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

3.4.2 [2882] 周末舞会

```cpp
#include <iostream>
#include <queue>//头文件
#include <algorithm>//算法文件
using namespace std;
int main()
{
    int n,m,k;
    while(~scanf("%d%d%d",&n,&m,&k))
    {
        queue<int>q1,q2;//定义一个队列
        //初始化，保持栈初始状态为空
        while(!q1.empty())q1.pop();
        while(!q2.empty())q2.pop();
        for(int i=1;i<=n;i++)
            q1.push(i);//往栈里 添加男生
        for(int i=1;i<=m;i++)
            q2.push(i);//往栈里 添加女生
        while(k--)
        {
            int x1=q1.front();//取出队头配对
            int x2=q2.front();
            q1.pop(),q2.pop();
            q1.push(x1),q2.push(x2);//配对后的男生女生重新排队
            printf("%d %d\n",x1,x2);
        }
    }
    return 0;
}
```

```
int main()
{
   int n,m,k;
   while(~scanf("%d%d%d",&n,&m,&k))
   {
      queue<int>q1,q2;//定义一个队列
      while(!q1.empty())q1.pop();    while(!q2.empty())q2.pop();
      for(int i=1;i<=n;i++)_______1_____________//往栈里 添加男生
      for(int i=1;i<=m;i++) _______1_____________//往栈里 添加女生
      while(k--)
      {
         int x1=_______2_____________//取出队头配对
         int x2=_______2_____________
         q1.pop(),q2.pop();
         q1.push(x1),q2.push(x2);//配对后的男生女生重新排队
         printf("%d %d\n",x1,x2);
      }
   }
```

3.4.3 [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

3.4.3 [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();
    }
}
```

3.4.3 [2884] 围圈报数

```
int main()
{
    _________1_________________
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        ______2____________________
    while (!student.empty())
    {
        for (int i = 1; i <= m-1; i++)
        {
            int x = _________3_________________
            student.pop();
            student.push(x);
        }
        cout << student.front() << " ";
        student.pop();
    }
}
```

3.5 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

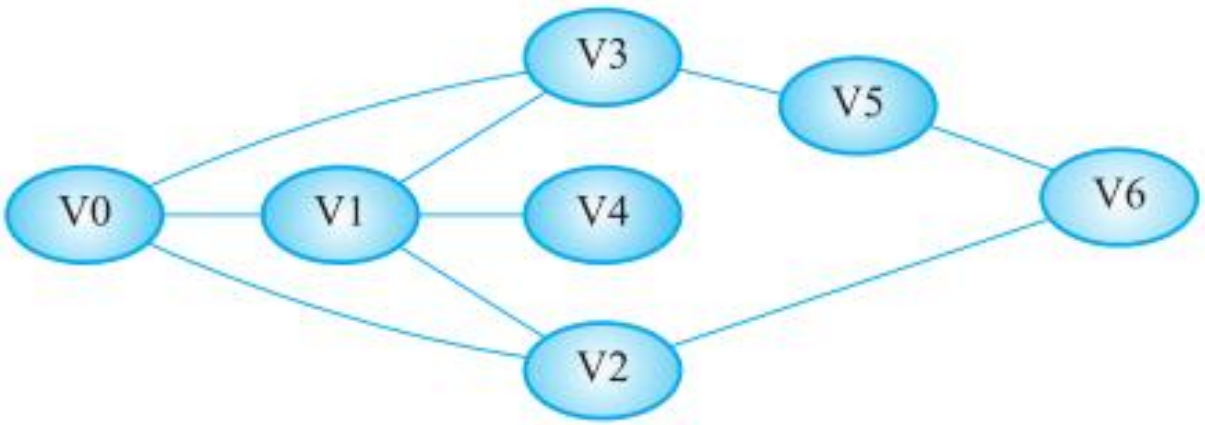
1.从图中的某一顶点V0开始，先访问V0；

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：

3.5 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
    }
}
```

```
void BFS(int curNode,int curDepth){
  while(front < rear) {
        front++;
        for(i = 0; i < m; i++) {
              newNode = Expend(q[front].node)
              if(!Exist(newNode)) {
                    q[rear++].node = newnode;
              }
              if(newNode == target) return ;
        }
    }
        return;
}
```

3.5.1 [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
```

3.5.1 [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。

3.5.1 [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

3.5.1 [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  }
```

3.5.1 [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 }
```

3.5.1 [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);
       }
    }
}
```

3.5.1 [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;
}
```

3.5.1 [2806] 走出迷宫-DFS栈模拟

```
#include<bits/stdc++.h>
using namespace std;
int n,t,m,min1,flag,f,qx,qy,qs;
char x[105][105];
int tx[4]={0,1,0,-1},ty[4]={1,0,-1,0};
struct node{
    int x,y,s;//x,y坐标 s步数
    int a[105][105];//记录x,y有没有走过
}S,T,now;
stack<node>q;
int main()
{
    cin>>n>>m;
    min1=1e9+7;
    for(int i=0;i<n;i++)cin>>x[i];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(x[i][j]=='#')S.a[i][j]=1;
            else if(x[i][j]=='S')
            {
                S.x=i,S.y=j,S.s=0;
                S.a[i][j]=1;
            }
            else if(x[i][j]=='T')
            {
                T.x=i,T.y=j;
            }
        }
    }//找到起点和终点
    q.push(S);//起点入栈
    flag=0;//记录是否到终点
    while(!q.empty())
    {
    cout<<min1<<endl;
}
```

3.5.1 [2806] 走出迷宫-DFS栈模拟

```
34      while(!q.empty())
35      {
36          now=q.top();
37          q.pop();
38          if(now.x==T.x&&now.y==T.y)
39          {//到达终点
40              flag=1;
41              min1=min(min1,now.s);
42          }
43          for(int i=0;i<4;i++)
44          {  //获得下一个点的坐标以及步数
45              qx=now.x+tx[i],qy=now.y+ty[i],qs=now.s+1;
46              if(qx>=0&&qy>=0&&qx<n&&qy<m&&now.a[qx][qy]==0)
47              {//如果能走
48                  now.a[qx][qy]=1;//记录走过
49                  now.x=qx,now.y=qy,now.s=qs;//更新now
50                  q.push(now);
51                  now.x-=tx[i],now.y-=ty[i],now.s--;//回退到上一个点
52                  now.a[qx][qy]=0;
53                  //回退到上一个点的目的是把其他能走的点入栈，起到保存现场的作用
54              }
55          }
56      }
```

3.5.2 [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。

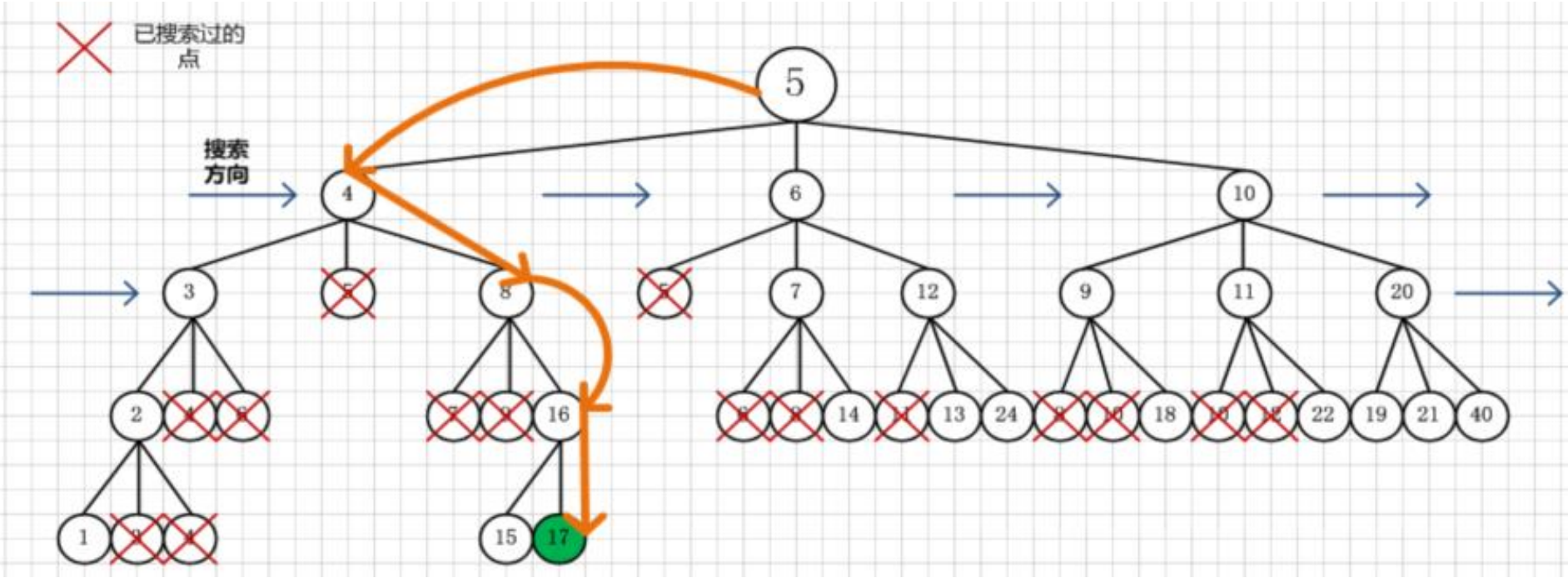
输出一个整数，农夫抓到牛所要花费的最小分钟数。

样例输入 5 17

样例输出 4

3.5.2 [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。

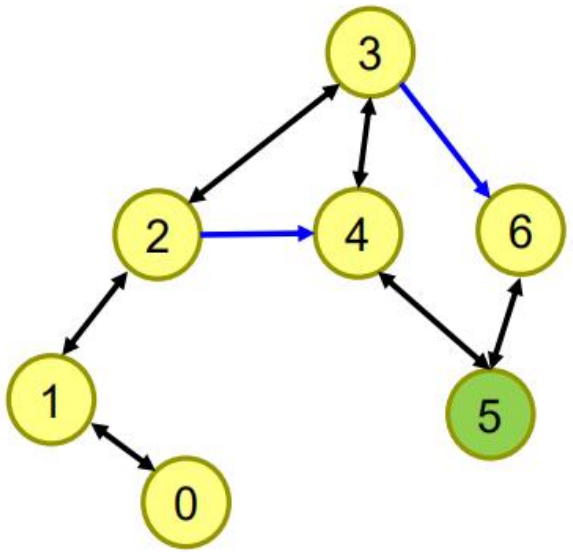


详细分析

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）。

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

详细分析

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。

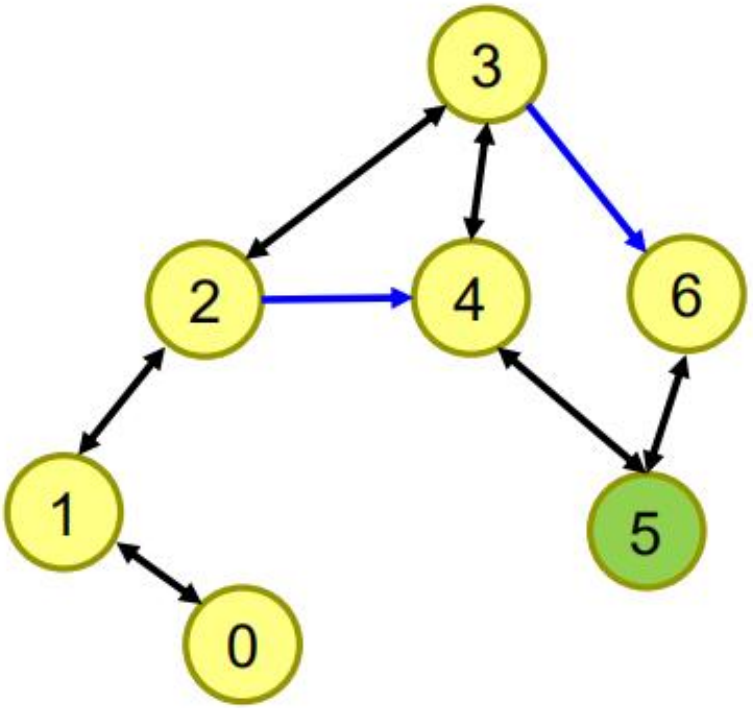
第1层：2, 4, 6

第2层：1, 5

第3层：0

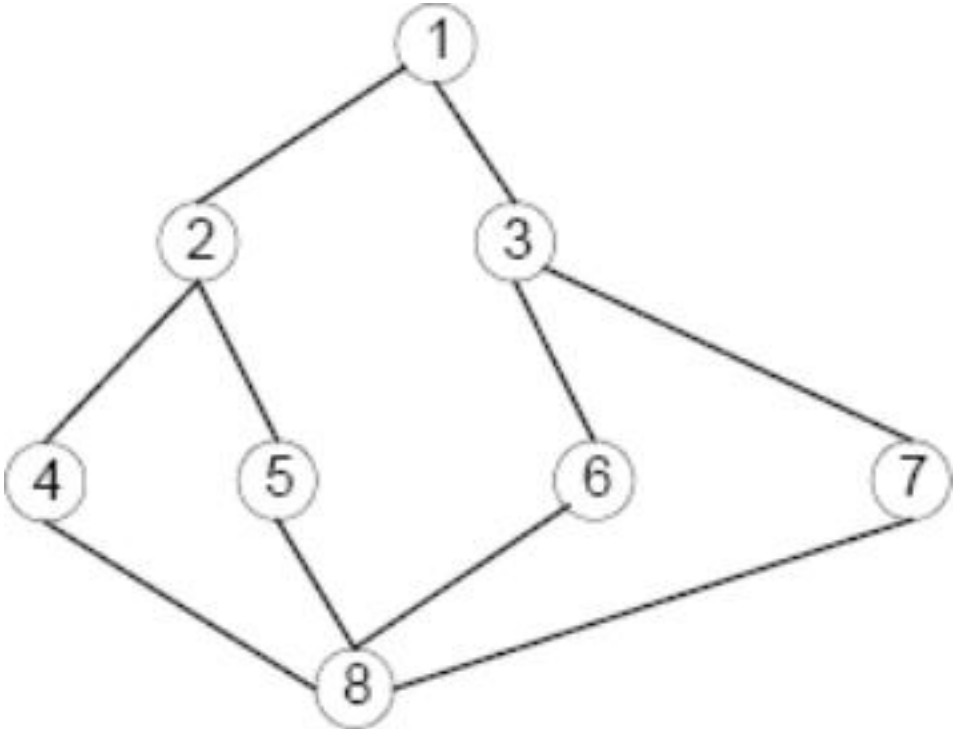
依层次顺序，从小到大扩展节点。

把层次低的点全部扩展出来后，才会扩展层次高的点。



详细分析

深搜 vs. 广搜



若要遍历所有节点：

- 深搜

1-2-4-8-5-6-3-7

- 广搜

1-2-3-4-5-6-7-8

广搜算法

(1) 把初始节点S0放入Open表中；

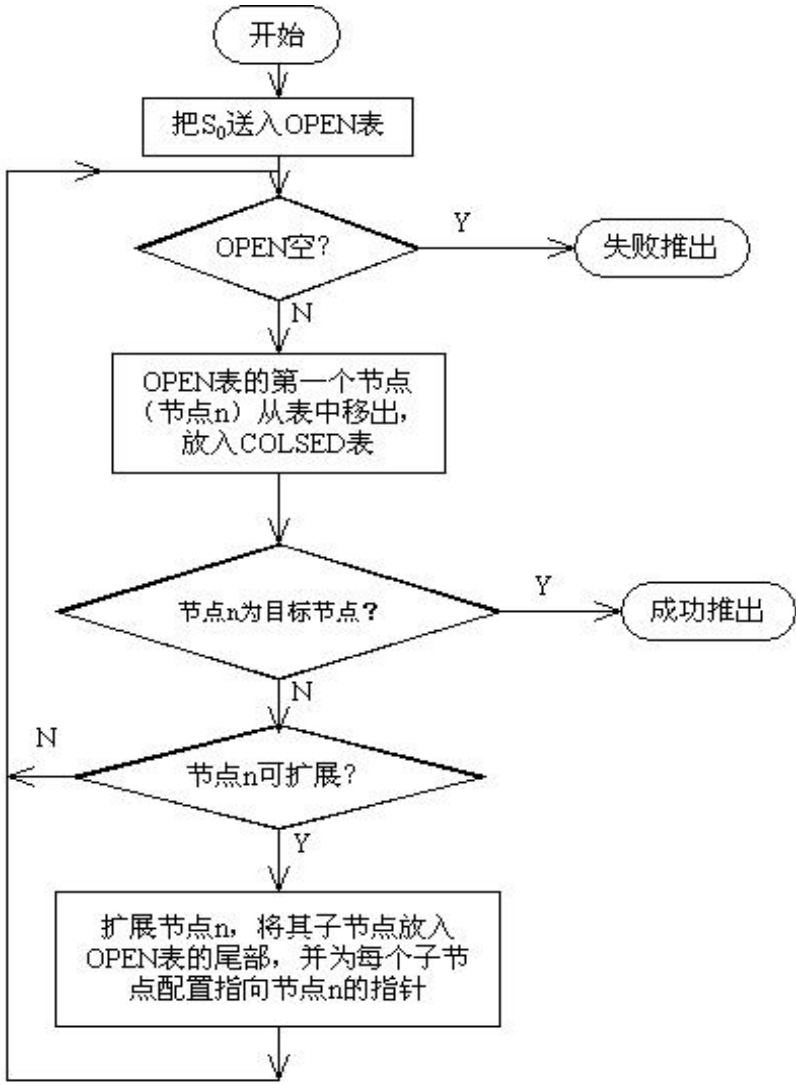
(2) 如果Open表为空，则问题无解，失败 退出；

(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；

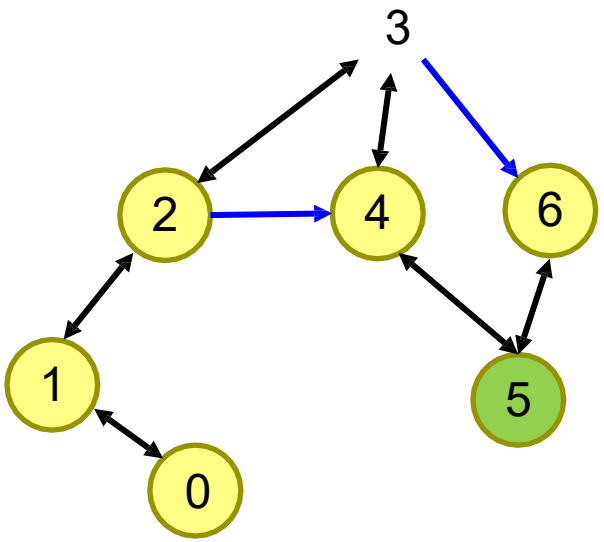
(5) 扩展节点n，将其不在Closed表和Open表中的子节点(判重）放入Open表的尾部，并为每一个子节点设置指向父节点的指针(或记录节点的层次），然后转第(2)步。



3.5.2 [2805]抓住那头牛-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？ 广度优先搜索队列变化过程：



3

Closed

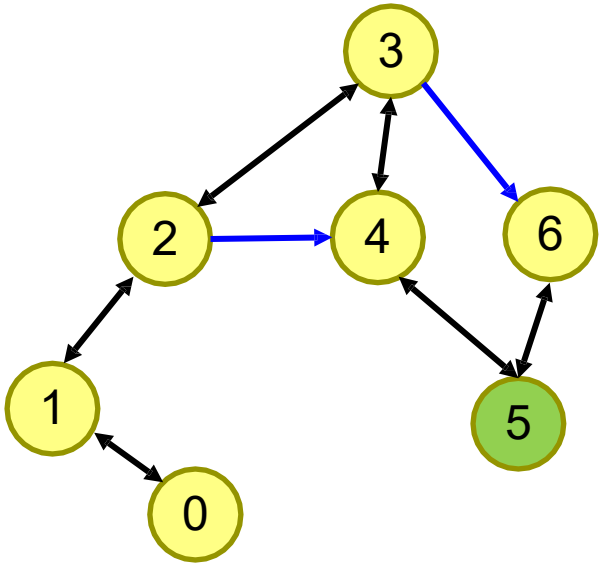
Open

3.5.2 [2805]抓住那头牛-分析

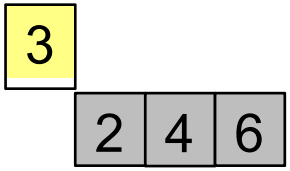
假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索队列变化过程：



Closed

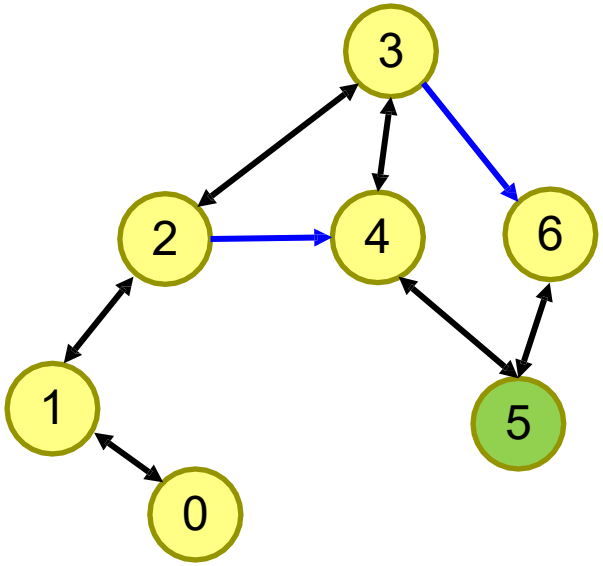
Open

3.5.2 [2805]抓住那头牛-分析

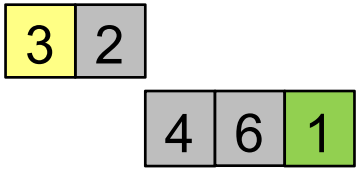
假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索队列变化过程：



Closed

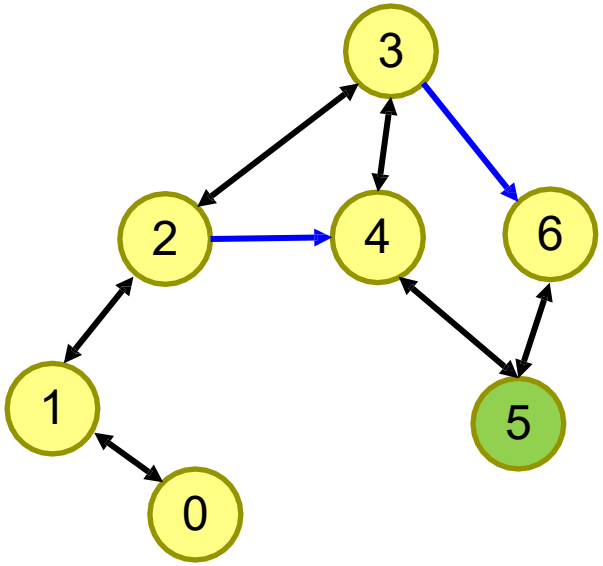
Open

3.5.2 [2805]抓住那头牛-分析

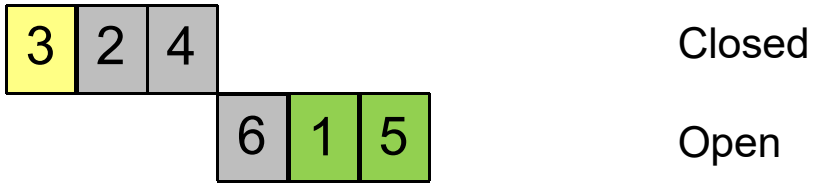
假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索队列变化过程：

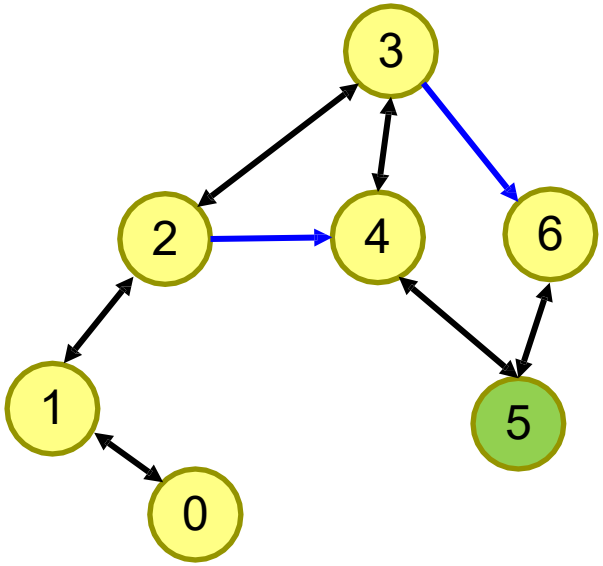


3.5.2 [2805]抓住那头牛-分析

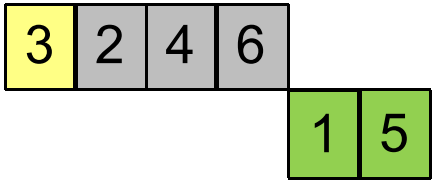
假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索队列变化过程：



Closed

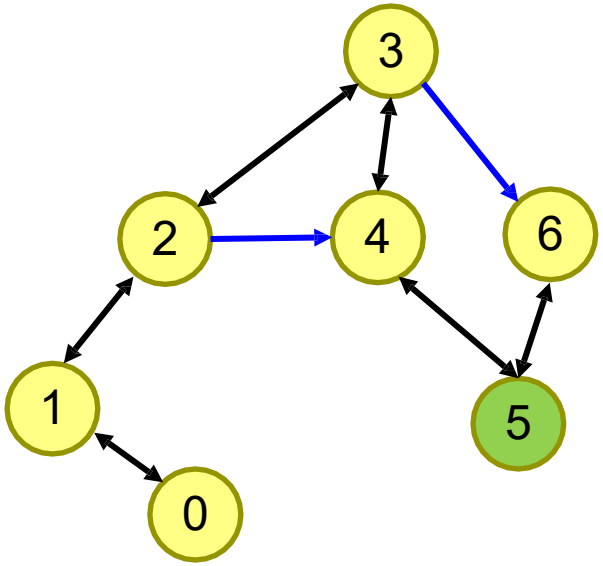
Open

3.5.2 [2805]抓住那头牛-分析

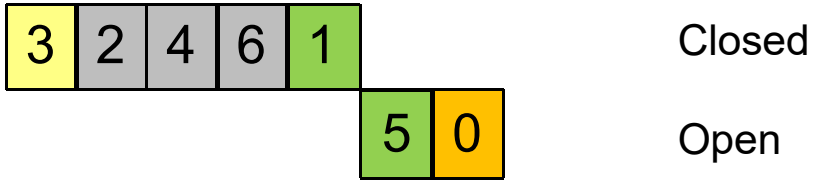
假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索队列变化过程：

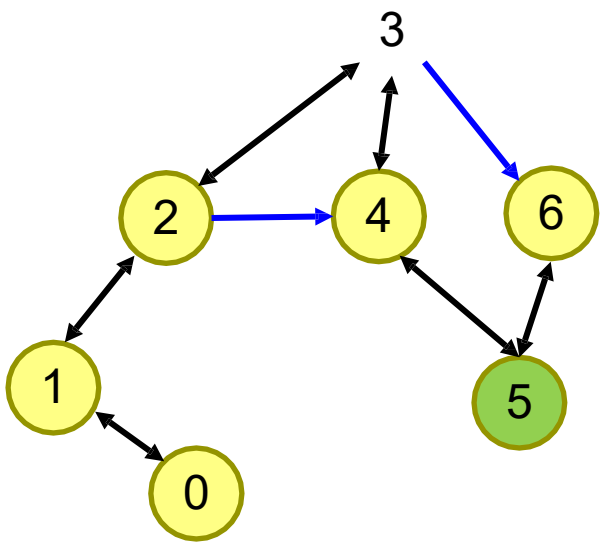


3.5.2 [2805]抓住那头牛-分析

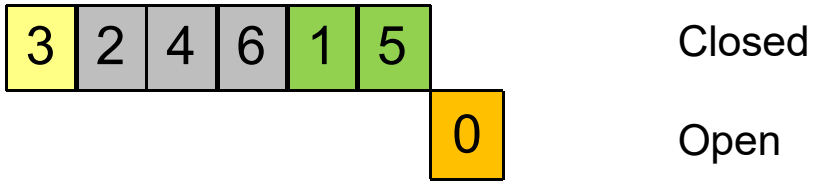
假设农夫起始位于点3，牛位于5

N=3,K=5，最右边是6。

如何搜索到一条走到5的路径？



广度优先搜索队列变化过程：



目标节点5出队列，问题解决！

3.5.2 [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
```

3.5.3 [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

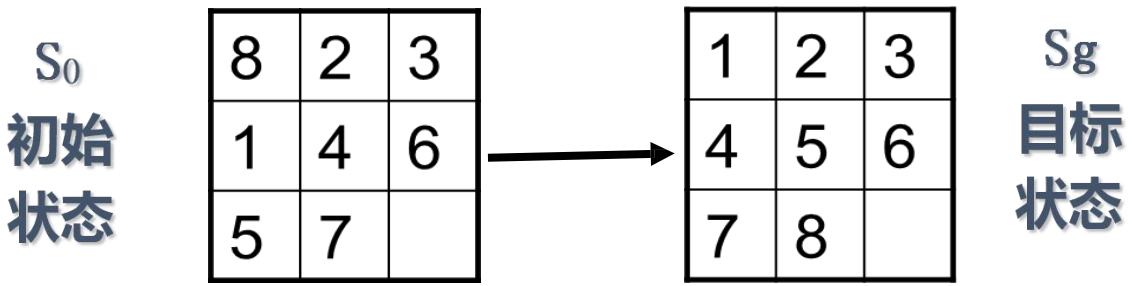
只有一行，该行只有一个数字，表示从初始状态到目标状态需要的最少移动次数(测试数据中无特殊无法到达目标状态数据)

样例输入

283104765

样例输出

4



3.5.3 [3378]八数码-分析

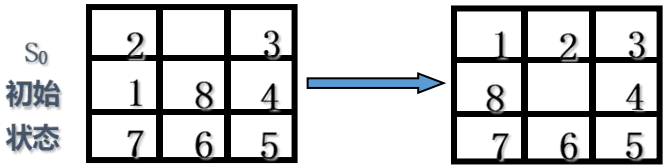
八数码的解决的方案由很多：

（1）非启发性的搜索：深搜、广搜，双向广搜，迭代加深的深搜；

（2）启发性搜索：A*，IDA*；（启发性算法有局部择优搜索和最好优先搜索，A*和IDA*都属于最好优先搜索，局部择优搜索有爬山算法，不过还没有研究）

（3）基于概率的随机搜索；遗传算法（比较有名的还有蚁群算法，模拟退火算法等智能搜索，不过没有研究）

DFS



规则：上下左右

0层

1层

2层

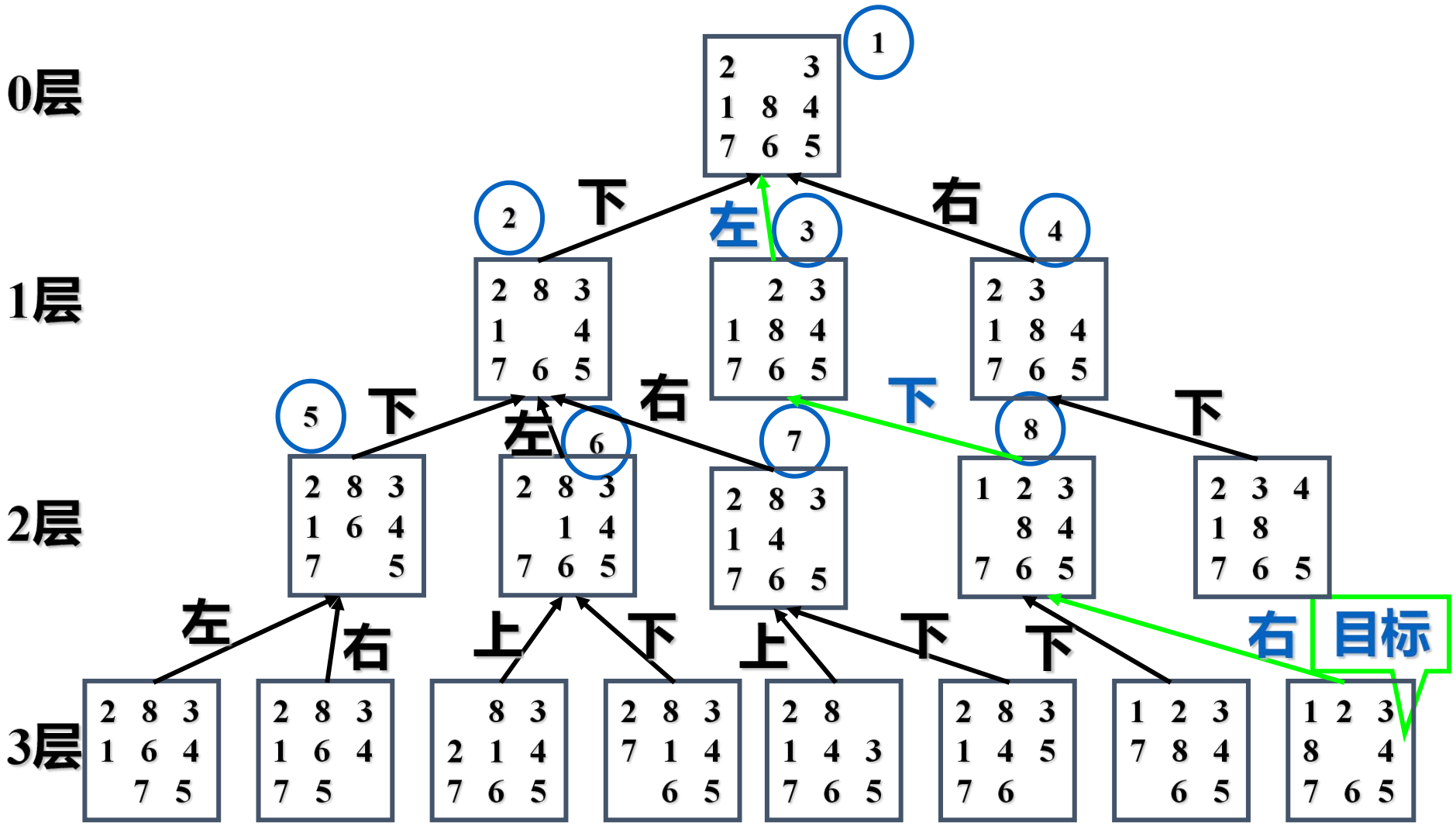
3层

4层

下 左 右

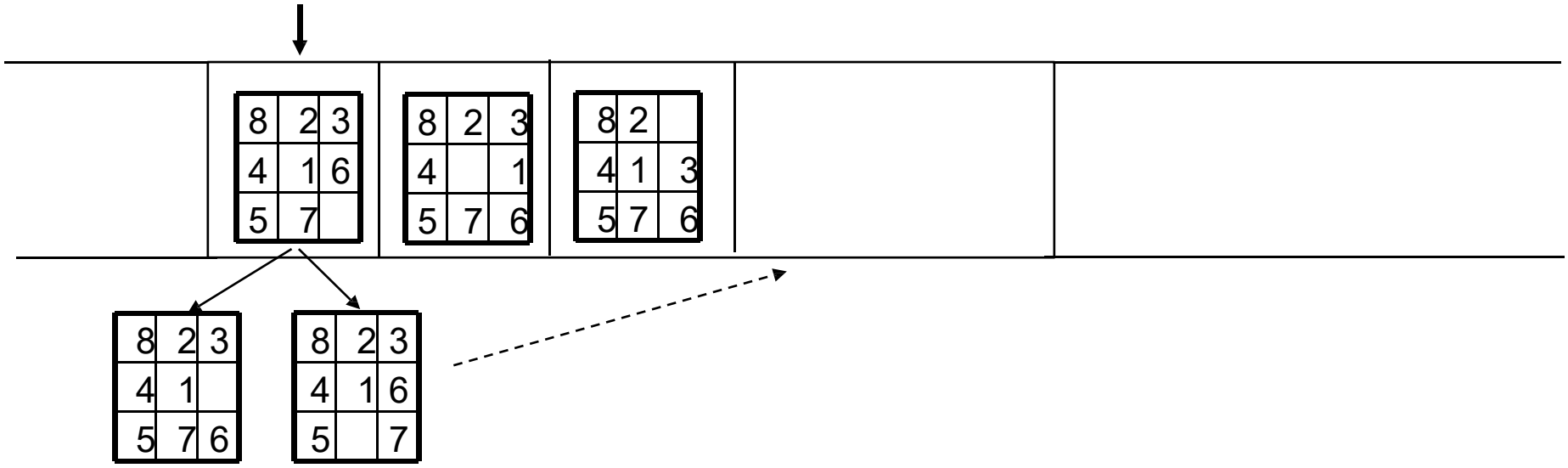
目标

BFS



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

待续

今天的课程结束啦……

下课了…
同学们再见！