

# C++编程

# 队列

queue

## 先进先出

### **FIFO**

first-in-first-out

请预测  
输出结果

```
1 #include<iostream>
2 #include<queue>
3 using namespace std;
4 int main(){
5     queue<int> q;
6     q.push(3);
7     q.push(1);
8     q.push(4);
9     cout<<q.size()<<endl;
10    cout<<q.front()<<endl;
11    q.pop();
12    cout<<q.size()<<endl;
13    cout<<q.front()<<endl;
14    q.pop();
15    cout<<q.size()<<endl;
16    cout<<q.front()<<endl;
17    q.pop();
18    cout<<q.size()<<endl;
19    return 0;
20 }
```

3		
3	1	
3	1	4

	1	4
--	---	---

		4
--	--	---

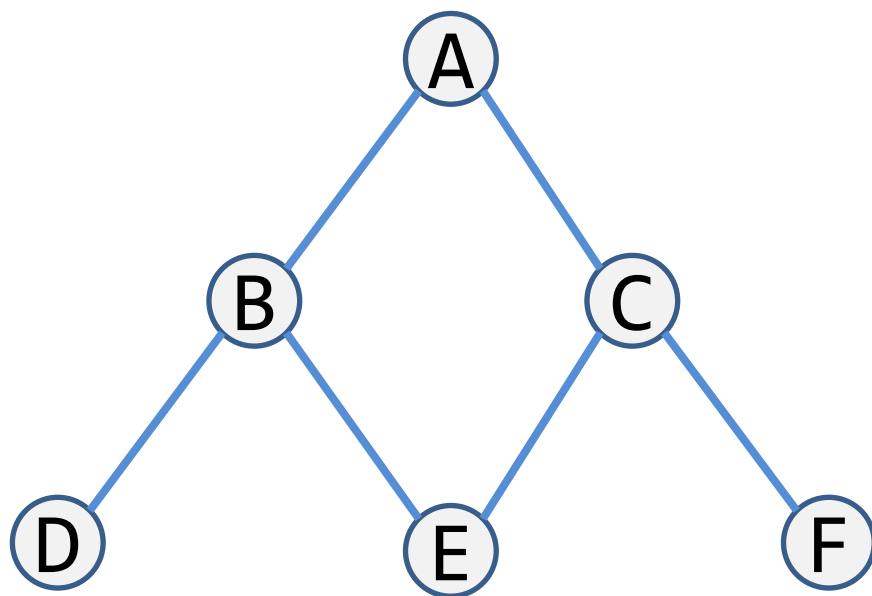
--	--	--

# 广度优先搜索

## Breadth-first Search

# 藏宝图 BFS

你拿到一张地下藏宝图，标有若干藏宝洞。你从A洞放了一把火，火势蔓延开，这几个洞被火烧的顺序会是怎样的呢？



**ABCDEF**

广度优先搜索

**breadth-first search**

简称BFS

# Flood Fill

Flood fill, also called seed fill, is an algorithm that determines the area **connected** to a given node in a multi-dimensional array

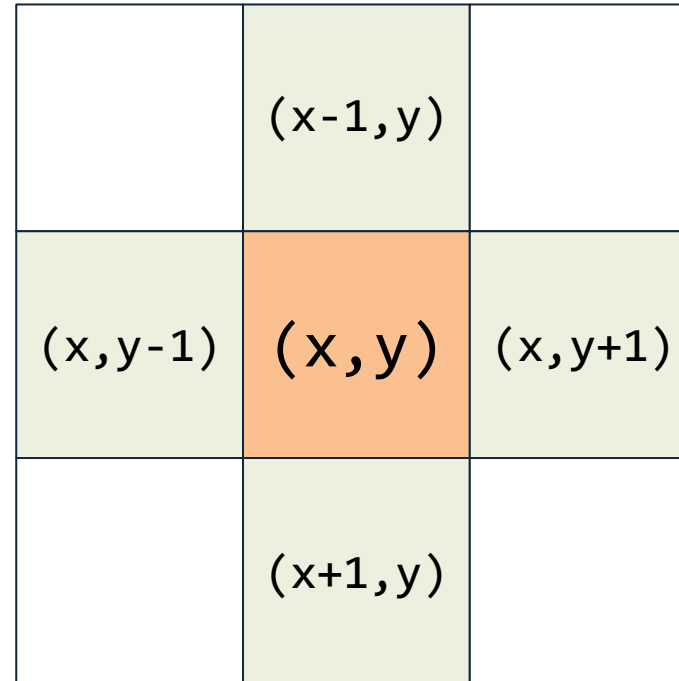
It can be implemented by DFS or BFS.

1	2	4	6	8	12	16
3			9			19
5			13			23
7	10	14	17	20	24	27
11			21			29
15			25			31
18	22	26	28	30	32	33

**BFS**访问顺序

四方向

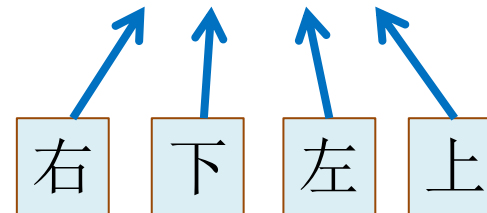
# 连通方向: 四方向



x代表行号  
y代表列号

dx代表行号的变化  
dy代表列号的变化

```
int dx[4]={0,1,0,-1};  
int dy[4]={1,0,-1,0};
```



# 连通方向: 八方向

$(x-1, y-1)$	$(x-1, y)$	$(x-1, y+1)$
$(x, y-1)$	$(x, y)$	$(x, y+1)$
$(x+1, y-1)$	$(x+1, y)$	$(x+1, y+1)$

依次是哪  
八个方向

```
int dx[8]={1,1,1,0,0,-1,-1,-1};  
int dy[8]={1,0,-1,1,-1,1,0,-1};
```



# 发洪水

在 $n*m$ 格的地图上发洪水了，水从第 $a$ 行第 $b$ 列涌入。'-'代表空地，'+'代表墙体。如果两块空地是八向连通的，那么他们同时被淹或者同时不被淹。输入 $n,m,a,b$ 和原始地图。输出被淹后的地图，'@'代表被淹的格子。 $n,m \leq 100$

输入样例

8 8 8 1

---+-----

--++-----

-+++-----

+++-----

-----+++

-----+++

-----++--

- - - - + - - -

输出样例

---+@@@@

--++@@@@

-+++@@@@

+++@@@@@

@@@@@++++

@@@@@++++-

@@@@@++--

@@@@@+---

输入样例

4 5 4 5

---++

-+-+--

-++++

-----

输出样例

@@@++

@+@+-

@++++

@@@@@



# 发洪水：DFS递归实现

递归方式实现深度优先搜索算法

不断沿某个方向往更深处探索

直到无法再深入时退回，再尝试其他方向

```
8 void dfs(int x,int y){
9     d[x][y]='@';
10    for(int k=0;k<8;k++){
11        int nx=x+dx[k],ny=y+dy[k];
12        if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&d[nx][ny]!='-'){
13            dfs(nx,ny);
14        }
15 }
```

从(x,y)格开始深度优先搜索

淹没(x,y), 修改地图中(x,y)符号为 '@'

依次循环查看8个方向

走到新位置(nx,ny)

若(nx,ny)没越界且(nx,ny)符号为 '-'

从(nx,ny)格继续深度优先搜索

# 发洪水：BFS队列实现

BFS用队列**queue**维护搜索顺序

头文件<queue>

对每一格，优先访问它的所有邻居，都进入队列

先进先出FIFO，从队首格子继续拓展

```
20 queue<Node> q;
21 d[a][b]='@';
22 q.push((Node){a,b});
23 while(!q.empty()){
24     Node now=q.front(); q.pop();
25     for(int k=0;k<8;k++){
26         int nx=now.x+dx[k],ny=now.y+dy[k];
27         if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&d[nx][ny]=='-'){
28             d[nx][ny]='@';
29             q.push((Node){nx,ny});
30         }
31     }
32     print();
33 }
```

(a,b)格被淹, 入队

当队列不为空不断循环

队首格子存入now, 出队

依次循环查看8个方向

走到新位置(nx,ny)

若(nx,ny)没越界且(nx,ny)符号为'-'

(nx,ny)格被淹, 入队

快快编程  
kkcoding.net

# 发洪水：DFS栈实现

DFS用栈stack维护搜索顺序

头文件<stack>

对每一格，优先访问它的所有邻居，都进入栈  
后进先出LIFO，从栈顶格子继续拓展

```
20 stack<Node> s;  
21 d[a][b]='@';  
22 s.push((Node){a,b});  
23 while(!s.empty()){  
24     Node now=s.top(); s.pop();  
25     for(int k=0;k<8;k++){  
26         int nx=now.x+dx[k],ny=now.y+dy[k];  
27         if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&d[nx][ny]=='-'){  
28             d[nx][ny]='@';  
29             s.push((Node){nx,ny});  
30         }  
31     }  
32     print();  
33 }
```

(a,b)格被淹, 入栈

当栈不为空不断循环

栈顶格子存入now,出栈

依次循环查看8个方向

走到新位置(nx,ny)

若(nx,ny)没越界且(nx,ny)符号为'-'

(nx,ny)格被淹, 入栈

快快编程  
kkcoding.net

# BFS 对比 DFS

BFS
队列 queue


DFS
栈 stack

DFS递归版代码简单  
因为递归利用了内存的栈分配  
不需要自己维护栈

BFS队列版代码多几行  
因为需要自己维护队列

# 发洪水： 错误代码1

```
20 queue<Node> q;  
21 d[a][b]='@';  
22 q.push((Node){a,b});  
23 while(!q.empty()){  
24     Node now=q.top();  
25     for(int k=0;k<8;k++){  
26         int nx=now.x+dx[k],ny=now.y+dy[k];  
27         if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&d[nx][ny]=='-'){  
28             d[nx][ny]='@';  
29             q.push((Node){nx,ny});  
30         }  
31     }  
32     print();  
33 }
```



# 发洪水： 错误代码2

```
21 queue<Node> q;  
22 vst[a][b]=1; ←  
23 q.push((Node){a,b});  
24 while(!q.empty()){  
25     Node now=q.front(); q.pop();  
26     for(int k=0;k<8;k++){  
27         int nx=now.x+dx[k],ny=now.y+dy[k];  
28         if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&!vst[nx][ny]){  
29             vst[nx][ny]=1; ←  
30             q.push((Node){nx,ny});  
31         }  
32     }  
33     print();  
34 }
```

# 发洪水： 错误代码3

```
20 queue<Node> q;  
21 d[a][b]='@';  
22 q.push((Node){a,b});  
23 while(!q.empty()){  
24     Node now=q.front(); q.pop();  
25     for(int k=0;k<8;k++){  
26         int nx=now.x+dx[k],ny=now.y+dy[k];  
27         if(d[nx][ny]=='-'){ ←  
28             d[nx][ny]='@';  
29             q.push((Node){nx,ny});  
30         }  
31     }  
32     print();  
33 }
```

因为缓冲带避免了越界

推荐： 额外判断边界+缓冲带



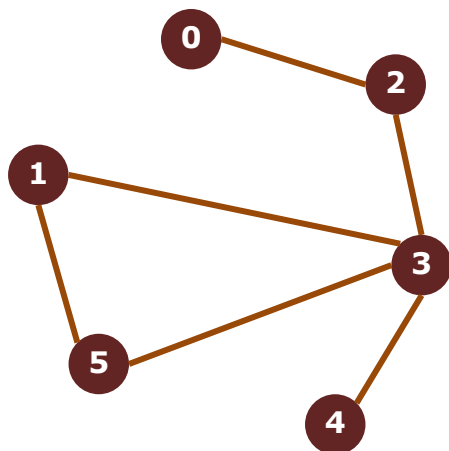
# 攀亲戚

你发现自己和古代一个皇帝长得很像：都有两个眼睛一个鼻子，你想知道皇帝是不是你的远方亲戚，你是不是皇亲国戚。目前你能掌握的信息有 $m$ 条，关于 $n$ 个人：第 $i$ 条信息包含两个人的编号 $a_i, b_i$ ，表示 $a_i$ 和 $b_i$ 是亲戚。你的编号是0，皇帝的编号是1，最大编号为 $n-1$ ，请问能否通过信息推理出你和皇帝是不是亲戚？备注：亲戚关系具有传递性。输入 $m$ 和 $n$ ，均不超过1000。

输入样例：

```
6 6
0 2
3 4
5 1
2 3
3 5
1 3
```

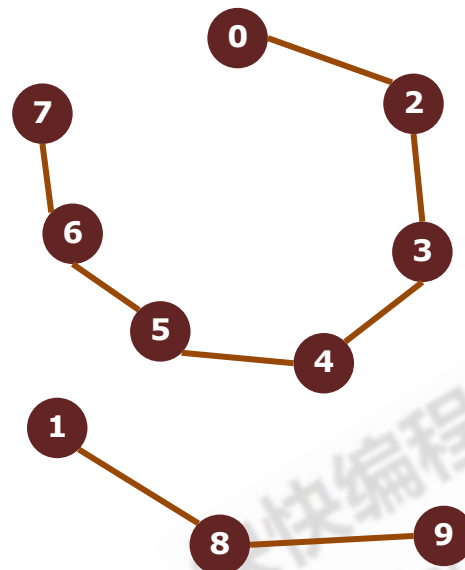
输出样例：  
Yes



输入样例：

```
8 10
0 2
4 5
6 7
1 8
8 9
2 3
5 6
3 4
```

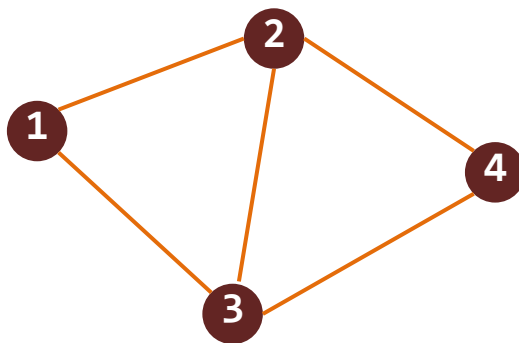
输出样例：  
No



# 无向无权图的储存

邻接矩阵  
adjacency matrix

## 邻接矩阵



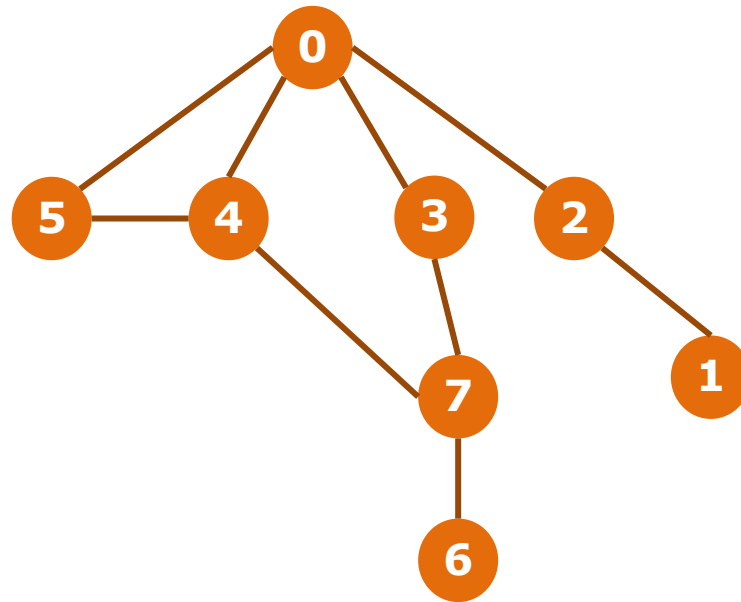
	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

```
int d[N][N];
```

$d[a][b]$ 代表  
节点a和b  
是否是邻居

# 攀亲戚 DFS顺序

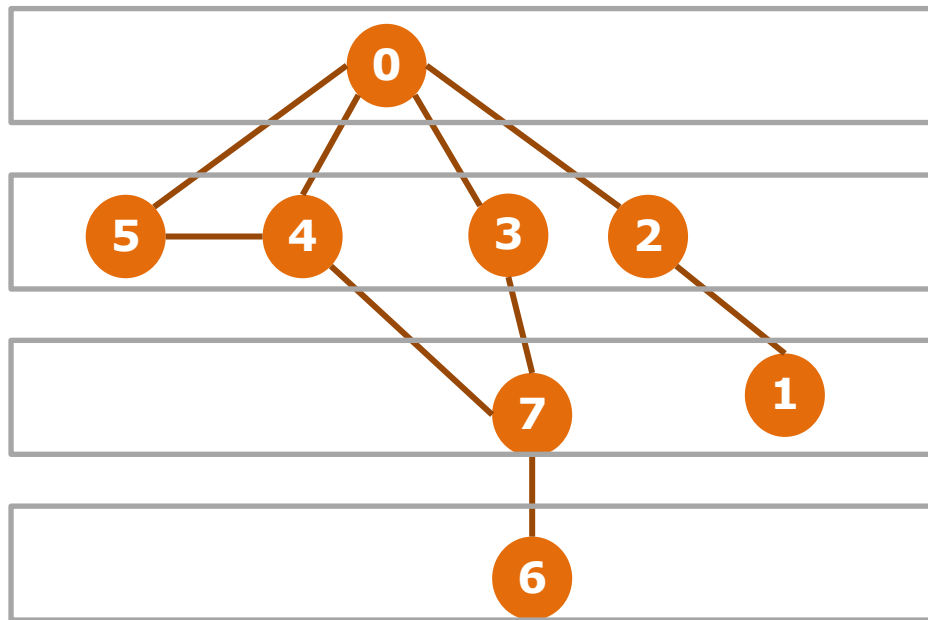
从0号节点开始访问



DFS访问顺序为0,5,4,7,6,3,2,1

# 攀亲戚 BFS顺序

从0号节点开始访问



BFS访问顺序为0,5,4,3,2,7,1,6

# 完善程序

```
1  #include<bits/stdc++.h>
2  #define N 1009
3  using namespace std;
4  int m,n,a,b;
5  bool d[N][N],vst[N];
6  void bfs(int x){
19 int main(){
20     freopen("relation.in","r",stdin);
21     freopen("relation.out","w",stdout);
22     cin>>m>>n;
23     for(int i=1;i<=m;i++){
24         cin>>a>>b;
25         d[a][b]=1;
26     }
27     bfs(1);
28     if( ) cout<<"Yes"<<endl;
29     else cout<<"No"<<endl;
30     return 0;
31 }
```

# 完善程序

```
4 int m,n,a,b;
5 bool d[N][N],vst[N];
6 void bfs(int x){
7     queue<int> q;
8     // 
9     q.push(x);
10    while(!q.empty()){
11        int now=q.pop();
12        for(int k=1;k<=n-1;k++){
13            if(d[now][k]==0){
14                vst[k]=1;
15                q.push(k);
16            }
17        }
18    }
```

# 现场挑战

## 530

快快编程  
kkcoding.net





棋盘格里两人同时移动  
每秒可以移动一格  
求最少几秒相遇

固定一人不懂  
只让另一人移动

起点和终点给定  
求最短路长度  
结果再除以2

单源单汇最短路问题

每一步耗时都是1

无权图最短路问题  
用BFS求解



# 无权图最短路问题 用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

## 从J到R求最短路

BFS时第一次访问  
就能确定最短路结果

1	2	3	4	5
0		4		6
1	2	3		
2		4	5	6
3	4	5		

dst[x][y]代表从起点到  
(x,y)的最短路长度

```
31 cin>>n>>m;
32 for(int i=1;i<=n;i++)
33     for(int j=1;j<=m;j++){
34         cin>>d[i][j];
35         if(d[i][j]=='R')xR=i,yR=j;
36         else if(d[i][j]=='J')xJ=i,yJ=j;
37     }
38 int len=bfs();
39 if(len==-1)cout<<"forever"<<endl;
40 else cout<<fixed<<setprecision(1)<<len/2.0<<endl;
```

```

9 int bfs(){
10     queue<Node> q;
11     vst[xR][yR]=1;
12     dst[xR][yR]=0;
13     [ ]
14     while(!q.empty()){
15         Node now=q.front(); q.pop();
16         for(int k=0;k<4;k++){
17             int nx=now.x+dx[k],ny=now.y+dy[k];
18             if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&d[nx][ny]!='#'&&!vst[nx][ny]){
19                 vst[nx][ny]=1;
20                 dst[nx][ny]=[ ];
21                 q.push((Node){nx,ny});
22                 if(nx==xJ&&ny==yJ)return [ ];
23             }
24         }
25     }
26     return -1;
27 }

```

# 快快编程作业

467

768

530

拓展题

487,531,470