##邻接矩阵和邻接表版

```
//广度优先搜索邻接矩阵版
void BFS(int v, MGraph G){
    marked[v] = 1;//访问数组 1为访问了
    cout<<v<<endl;
    queue<int>que;
    que.push(v);
    while(!que.empty()){
        int node = que.front(); que.pop();
        for(int i = 0; i < G.n; i++){
            if(G.edge[node][i] != 0 && marked[i] == 0){
                cout<<i<<endl;
                marked[i] = 1;
                que.push(i);
            }
        }
    }
}

//广度优先搜索邻接表版
void BFSUn(int v, ALGraph al){
    cout<<v<<endl;
    marked[v] = 1;
    ArcNode* arc = NULL;
    queue<int> que;
    que.push(v);
    while(!que.empty()){
        arc = al.vertex[que.front()].firstedge; que.pop();
        while(arc != NULL){
            if(marked[arc->no]==0){//未被访问过
                cout<<arc->no<<endl;
                marked[arc->no] = 1;
                que.push(arc->no);
            }
            arc = arc->next;
        }
    }
}
```

# 完整的可运行的代码。邻接矩阵版

> 只测过一组数据

```
#include<iostream>
#include<algorithm>
#include<queue>
using namespace std;

#define N 8
//访问数组
```

```
int marked[N] = {0};
//定义邻接矩阵
struct MGraph{
    int vertex[N];
    int edge[N][N];
    int e,n;
};

//初始化邻接表信息
void init(MGraph &G,int vertex[],int edge[N][N]){
    for(int i=0;i<G.n;i++){
        G.vertex[i] = vertex[i];
    }
    for(int i=0;i<G.n;i++){
        for(int j=0;j<G.n;j++){
            G.edge[i][j] = edge[i][j];
        }
    }
}

//广度优先搜索 邻接表版
void BFS(int v, MGraph G){
    marked[v] = 1;
    cout<<v<<endl;
    queue<int>que;
    que.push(v);
    while(!que.empty()){
        int node = que.front(); que.pop();
        for(int i = 0; i < G.n; i++){
            if(G.edge[node][i] != 0 && marked[i] == 0){
                cout<<i<<endl;
                marked[i] = 1;
                que.push(i);
            }
        }
    }
}

int main(){
    int vertex[N] = {0,1,2,3,4,5,6,7};
    int edge[N][N] = {
        {0,1,0,1,1,0,0,0},
        {1,0,0,1,0,0,1,0},
        {0,0,0,0,0,0,1,0},
        {1,1,0,0,0,0,0,0},
        {1,0,0,0,0,0,0,1},
        {0,0,0,0,0,0,0,1},
        {0,1,1,0,0,0,0,1},
        {0,0,0,0,1,1,1,0}
    };
    MGraph G;
    G.n = N;
    init(G,vertex,edge);
    //DFS(0,G);
    BFS(0,G);
    return 0;
}
```

# 完整可运行的代码.邻接表版本.

```
#include<iostream>
#include<algorithm>
#include<queue>
using namespace std;

#define N 8
//访问数组
int marked[N] = {0};

//定义邻接边表
struct ArcNode{
    int no;//该边指向的顶点
    ArcNode *next;
};
//定义顶点表
struct VNode{
    int no;//顶点编号
    ArcNode *firstedge;
};
//复合结构体
struct ALGraph{
    VNode vertex[N];
    int e,n;
};

void init(ALGraph &al, int vertex[], int edge[N][N]){
    for(int i = 0; i < al.n; i++){
        al.vertex[i].no = vertex[i];//声明好邻接点
        al.vertex[i].firstedge = NULL;
    }
    //给边赋值
    for(int i=0;i<al.n;i++){
        ArcNode* temp = NULL;
        for(int j=0;j<al.n;j++){
            if(al.vertex[i].firstedge == NULL && edge[i][j] != 0){
                temp = (ArcNode*)malloc(sizeof(ArcNode));
                temp->no = j; temp->next = NULL;
                al.vertex[i].firstedge = temp;
            }else if(temp!=NULL && edge[i][j] != 0){
                ArcNode* arc = (ArcNode*)malloc(sizeof(ArcNode));
                arc->no = j; arc->next = NULL;
                temp->next = arc;
                temp = temp->next;
            }
        }
    }
}

void BFSUn(int v, ALGraph al){
    cout<<v<<endl;
    marked[v] = 1;
    ArcNode* arc = NULL;
    queue<int> que;
```

```
    que.push(v);
    while(!que.empty()){
        arc = al.vertex[que.front()].firstedge; que.pop();
        while(arc != NULL){
            if(marked[arc->no]==0){//未被访问过
                cout<<arc->no<<endl;
                marked[arc->no] = 1;
                que.push(arc->no);
            }
            arc = arc->next;
        }
    }
}

int main(){
    int vertex[N] = {0,1,2,3,4,5,6,7};
    int edge[N][N] = {
        {0,1,0,1,1,0,0,0},
        {1,0,0,1,0,0,1,0},
        {0,0,0,0,0,0,1,0},
        {1,1,0,0,0,0,0,0},
        {1,0,0,0,0,0,0,1},
        {0,0,0,0,0,0,0,1},
        {0,1,1,0,0,0,0,1},
        {0,0,0,0,1,1,1,0}
    };
    ALGraph G;
    G.n = N;
    init(G,vertex,edge);
    BFSUn(0,G);
    return 0;
}
```