

# Lecture 22: 图的遍历

## 21.1 图的广度优先遍历

### 21.1.1 基本概念

**概念:** 广度优先遍历 (Breadth-First-Search, BFS)

- 遍历:
  - 首先访问起始顶点  $v$ , 接着由  $v$  出发, 依次访问  $v$  的各个未访问过的邻接顶点  $w_1, w_2 \dots, w_n$ , 然后依次访问  $w_1, w_2 \dots, w_n$  的所有未被访问过的邻接顶点;
  - 再从这些访问过的顶点出发, 访问它们所有未被访问过的邻接顶点, 直至图中所有顶点都被访问过为止;
- 描述:
  - 1. 找到与一个顶点相邻的所有顶点;
  - 2. 找到哪些顶点已经被访问过, 然后仅遍历未访问的结点;
  - 3. 需要一个辅助队列;
- 总结: 基于邻接表的 BFS

```
void BFS(Graph G, int i)
{
    visit(i);
    visited[i] = true; // 标记当前的i结点为true
    EnQueue(Q, i) // 队列i中进队i

    while(Q != null)
    {
        v = DeQueue(Q); // 如果当前队列不为空, 则将当前队列的第一个元素 v 出队
        // 对队列头元素的所有边进行访问
        for(p = G.vertices[v].firstVertex; p != null; p = p->nextVertex)
        {
            // 存储当前访问的结点所指向的边所存储的顶点的位置
            currentVertexPos = p->adjVertex;

            if(visited[currentVertexPos] == false)
            {
                visit(currentVertexPos);
                visited[currentVertexPos] = true;
                EnQueue(Q, currentVertexPos);
            }
        }
    }
}
```

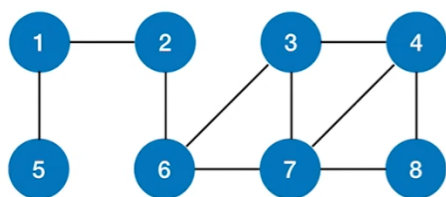
注意

- 同一个图的邻接矩阵表示方式唯一，因此广度优先遍历序列**唯一**；
- 同一个图的邻接表表示方式不唯一，因此广度优先遍历序列**不唯一**；

性能分析

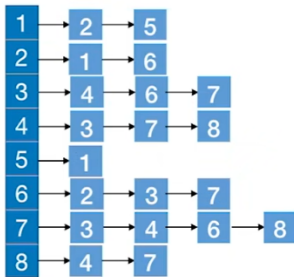
- 空间复杂度：  $O(|V|)$ 
  - 此空间为辅助队列所用，最大空间需求为：当前所有结点都连接在访问结点上；
- 时间复杂度：
  - 邻接矩阵：
    - 每个邻接点都需要  $O(|V|)$  的时间，一共有  $V$  个顶点
    - $O(|V|^2)$
  - 邻接表：
    - 访问  $V$  个顶点需要  $O(|V|)$  的时间，查找各个顶点的邻接点共需要  $O(|E|)$  的时间；
    - $O(|V| + |E|)$

概念图示



|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 7 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

邻接矩阵



邻接表

邻接矩阵存储的图：  
访问  $|V|$  个顶点需要  $O(|V|)$  的时间  
查找每个顶点的邻接点都需要  $O(|V|)$  的时间，而总共有  $|V|$  个顶点  
时间复杂度=  $O(|V|^2)$

邻接表存储的图：  
访问  $|V|$  个顶点需要  $O(|V|)$  的时间  
查找各个顶点的邻接点共需要  $O(|E|)$  的时间，  
时间复杂度=  $O(|V|+|E|)$

21.1.2 算法实现

```
bool visited[MAX_VERTEX_NUM]; //访问标记数组(初始为false)

//广度优先遍历
void BFS(Graph G,int v){ //从顶点v出发，广度优先遍历图G
    visit(v); //访问初始顶点v
    visited[v]=TRUE; //对v做已访问标记
    EnQueue(Q,v); //对顶点v入队列Q
    while(!isEmpty(Q)){
        DeQueue(Q,v); //顶点v出队列
        for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
            //检测v所有邻接点
            if(!visited[w]){ //w为v的尚未访问的邻接顶点
                visit(w); //访问顶点w
                visited[w]=TRUE; //对w做已访问标记
                EnQueue(Q,w); //顶点w入对列
            }
        }
    }
```

```

        }//if
    }//while
}

```

```

//对于出现不相邻结点的优化(非连通图)
void BFSTraverse(Graph G){ //对图G进行广度优先遍历
    for(i=1;i<G.vexnum;++i)
        visited[i]=FALSE; //访问标记数组初始化
    InitQueue(Q); //初始化辅助队列
    for(i=0;i<G.vexnum;++i) //从0号顶点开始遍历
        if(!visited[i]) //对每个连通分量调用一次BFS
            BFS(G,i); //vi为访问过，从vi开始BFS
}

```

**补充：**用 BFS 确定连通分量

- 对于无向图，调用 BFS 函数的次数=连通分量数

```

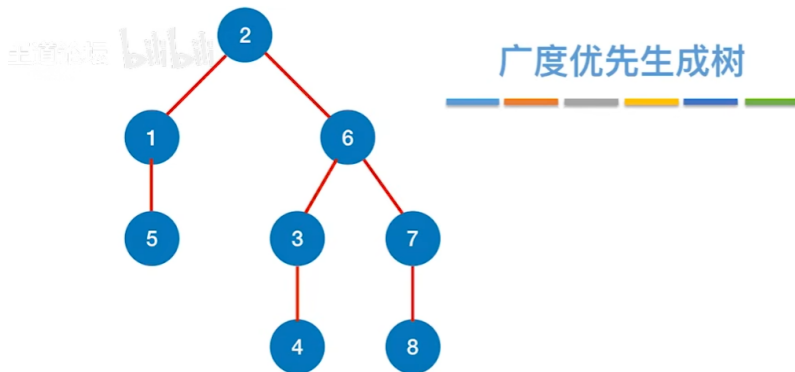
void BFSTraverse(Graph G)
{
    for(i=0;i<G.vexnum;++i)
    {
        visited[i]=FALSE;
    }
    InitQueue(Q);
    for(i=0;i<G.vexnum;++i)
        if(!visited[i])
            BFS(G,i);
}

```

### 21.1.3 广度优先生成树

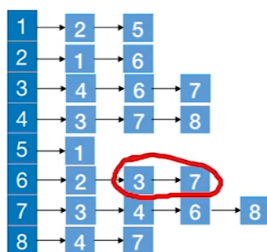
#### 广度优先生成树

- 概念：
  - 根据广度优先遍历的过程中访问各个结点所使用的边，得到的生成树；
- 图示：



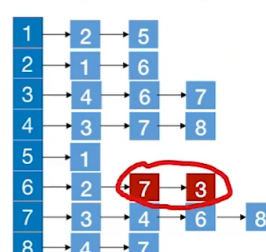
|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 7 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

邻接矩阵

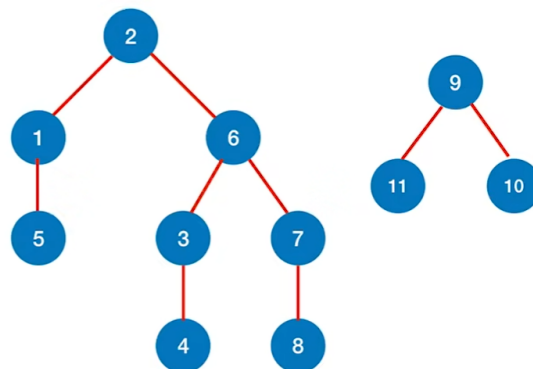
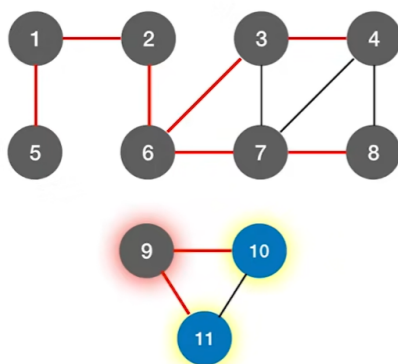


邻接表

广度优先生成树由广度优先遍历过程确定。由于邻接表的表示方式不唯一，因此基于邻接表的广度优先生成树也不唯一。



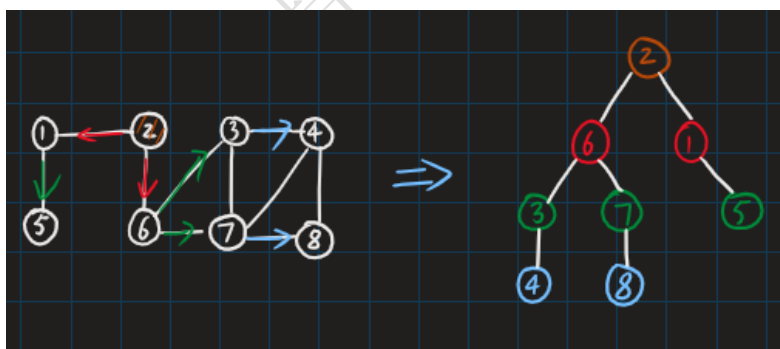
邻接表



对非连通图的广度优先遍历，可得到广度优先生成森林

总结：

- 假设从结点 2 开始



## 21.2 图的深度优先遍历

### 21.2.1 基本概念

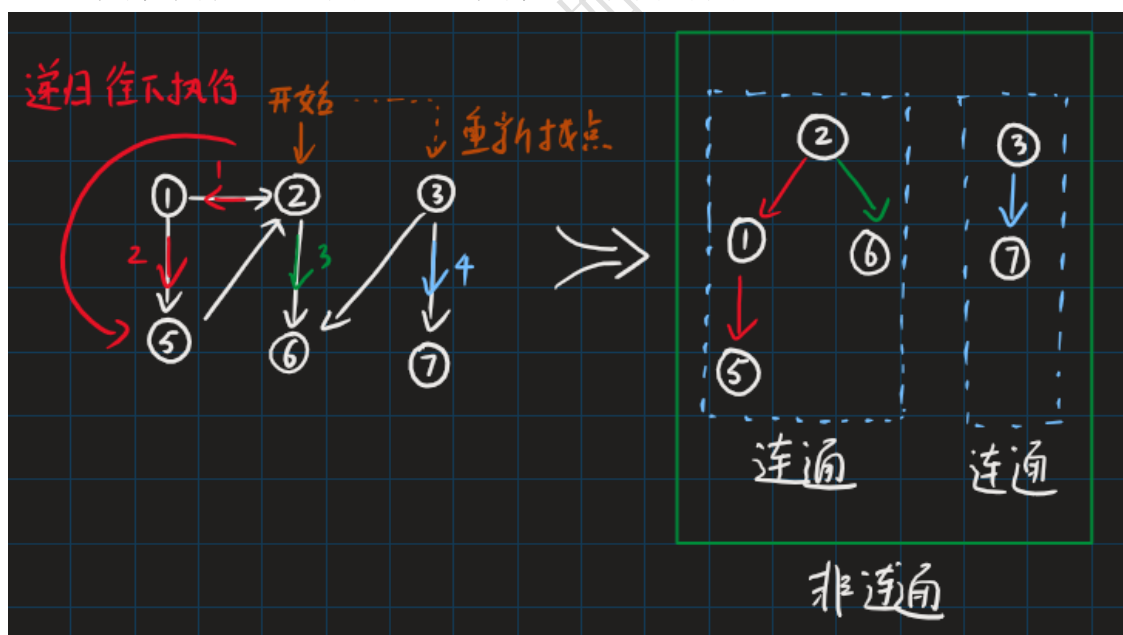
#### 概念

概念：

- 从当前结点的相邻结点中，选择一个，开始递归，一下捅到底；

过程：

- 首先访问图中某一起始顶点，然后由出发，访问与  $v$  邻接且未被访问的任意一个顶点  $w_1$ ，再访问与  $w_1$  邻接且未被访问的任意一个顶点  $w_2 \dots$  重复上述过程；
- 当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直至图中所有顶点均被访问过为止；
- 性能：
  - 空间复杂度： $O(|V|)$
  - 时间复杂度：
    - 邻接矩阵： $O(|V|^2)$
    - 邻接表： $O(|V| + |E|)$
- 对 无向图 进行BFS/DFS遍历
  - 调用 BFS/DFS 函数的次数 = 连通分量数 (几个非连通图)
  - 对连通图，只需要调用 1 次 BFS/DFS
- 对 有向图 进行BFS/DFS变异
  - 调用 BFS/DFS 函数的次数要具体问题具体分析
  - 若起始顶点到其他各顶点都有路径，则只需调用 1 次 BFS/DFS 函数
- 注意：
  - 邻接表的表示方式不唯一，所以遍历序列不唯一，生成树不唯一
- 总结：遍历过程
  - 从 2 开始，然后继续找点、从 3 开始，完成所有结点遍历



## 21.2.2 代码实现

### DFS 算法 - Final

```
bool visited[MAX_VERTEX_NUM]; //访问标记数组
void DFS(Graph G, int v){ //从顶点v出发，深度优先遍历图G
    visit(v); //访问顶点v
    visited[v]=TRUE; //投已访问标记
    for(w=FirstNeighbor(G, v); w>0; w=NextNeighbor(G, v, w))
```

```

        if(!visited[w]){    //w为u的尚未访问的邻接顶点
            DFS(G,w);
        }//if
    }
}

```

## 非连通图的处理办法

```

//非连通图的处理办法
void DFSTraverse(Graph G){    //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE;    //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v)    //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}

```

- 空间复杂度：
  - $O(|V|)$ ：来自**递归工作栈**
- 时间复杂度：
  - 访问结点的时间+访问所有边的时间
  - 邻接矩阵：
    - $O(|V| + |V|^2)$
  - 邻接表：
    - $O(|V| + |E|)$

### 21.2.3 深度优先生成树

- 同一个图的**邻接矩阵表示方式唯一**，因此**深度优先遍历序列唯一**，深度优先生成树也唯一
- 同一个图邻接表表示方式不唯一，因此深度优先遍历序列不唯一，深度优先生成树也不唯一
- 对有向图进行 BFS/DFS 遍历调用 BFS/DFS 函数的次数要具体问题具体分析
- 若起始顶点到其他各顶点都有路径，则只需调用 1 次 BFS/DFS 函数