

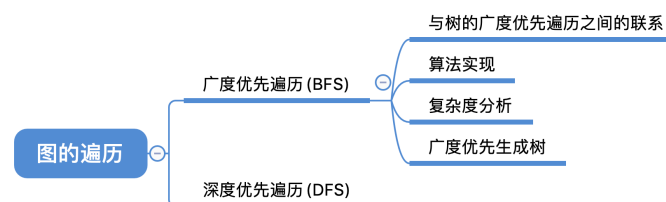
本节内容

图的遍历

BFS

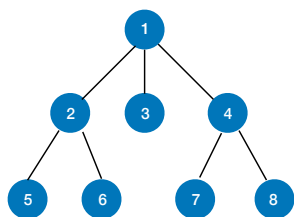
王道考研/CSKAOYAN.COM

知识总览



王道考研/CSKAOYAN.COM

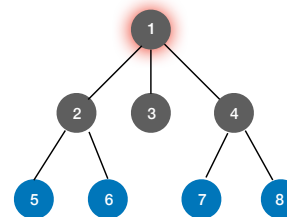
树的广度优先遍历



树的广度优先遍历：
通过根节点，可以找到下一层的结点
2、3、4。通过234又可以找到再下一层的
结点5678

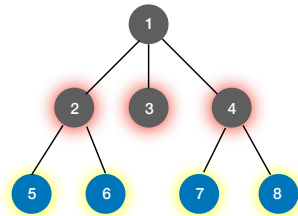
王道考研/CSKAOYAN.COM

树的广度优先遍历

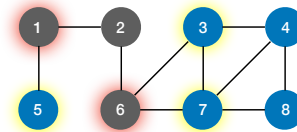


王道考研/CSKAOYAN.COM

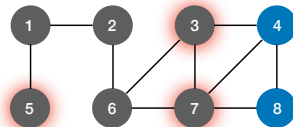
树的广度优先遍历



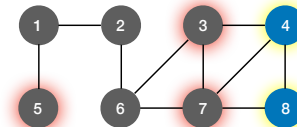
图的广度优先遍历



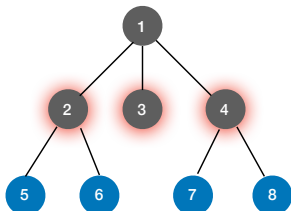
图的广度优先遍历



图的广度优先遍历



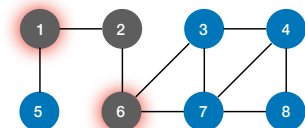
树 vs 图



不存在“回路”，搜索相邻的结点时，不可能搜到已经访问过的结点

树的广度优先遍历（层序遍历）：

- ①若树非空，则根节点入队
- ②若队列非空，队头元素出队并访问，同时将该元素的孩子依次入队
- ③重复②直到队列为空



搜索相邻的顶点时，有可能搜到已经访问过的顶点

王道考研/CSKAOYAN.COM

代码实现

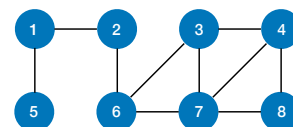
广度优先遍历（Breadth-First-Search, BFS）要点：

1. 找到与一个顶点相邻的所有顶点
2. 标记哪些顶点被访问过
3. 需要一个辅助队列

• **FirstNeighbor(G, x)**：求图G中顶点x的第一个邻接点，若有则返回顶点号。若x没有邻接点或图中不存在x，则返回-1。

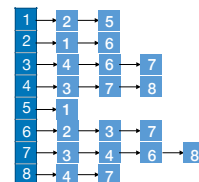
• **NextNeighbor(G, x, y)**：假设图G中顶点y是顶点x的一个邻接点，返回除y之外顶点x的下一个邻接点的顶点号，若y是x的最后一个邻接点，则返回-1。

bool visited[MAX_VERTEX_NUM]; // 访问标记数组



	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

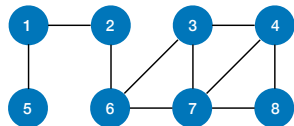
邻接矩阵



邻接表

王道考研/CSKAOYAN.COM

代码实现



```

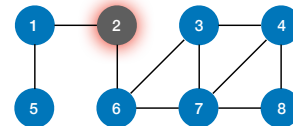
bool visited[MAX_VERTEX_NUM]; // 访问标记数组

// 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); // 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入队列
            }
        }
    }
}
    
```

	1	2	3	4	5	6	7	8
visited	false	false	false	false	false	false	false	false

王道考研/CSKAOYAN.COM

代码实现



```

bool visited[MAX_VERTEX_NUM]; // 访问标记数组

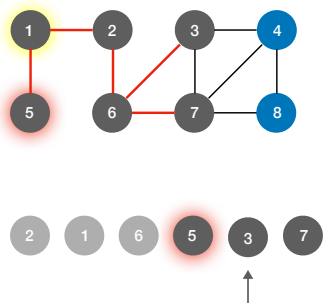
// 广度优先遍历
void BFS(Graph G, int v) { // 从顶点v出发，广度优先遍历图G
    visit(v); // 访问初始顶点v
    visited[v] = TRUE; // 对v做已访问标记
    Enqueue(Q, v); // 顶点v入队列Q
    while(!IsEmpty(Q)) {
        Dequeue(Q, v); // 顶点v出队列
        for(w = FirstNeighbor(G, v); w >= 0; w = NextNeighbor(G, v, w)) {
            // 检测v所有邻接点
            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入队列
            }
        }
    }
}
    
```

	1	2	3	4	5	6	7	8
visited	false	true	false	false	false	false	false	false

王道考研/CSKAOYAN.COM

代码实现

初始都为false



```
bool visited[MAX_VERTEX_NUM]; //访问标记数组

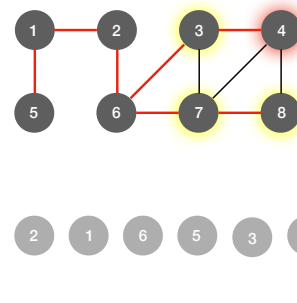
//广度优先遍历
void BFS(Graph G,int v){ //从顶点v出发, 广度优先遍历图G
    visit(v); //访问初始顶点v
    visited[v]=TRUE; //对v做已访问标记
    Enqueue(Q,v); //顶点v入队列Q
    while(!isEmpty(Q)){
        Dequeue(Q,v); //顶点v出队列
        for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
            //检测v所有邻接点
            if(!visited[w]){ //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入队列
            }
    }
}
```

	1	2	3	4	5	6	7	8
visited	true	true	true	false	true	true	false	false

王道考研/CSKAOYAN.COM

代码实现

初始都为false



```
bool visited[MAX_VERTEX_NUM]; //访问标记数组

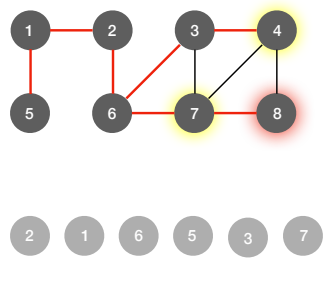
//广度优先遍历
void BFS(Graph G,int v){ //从顶点v出发, 广度优先遍历图G
    visit(v); //访问初始顶点v
    visited[v]=TRUE; //对v做已访问标记
    Enqueue(Q,v); //顶点v入队列Q
    while(!isEmpty(Q)){
        Dequeue(Q,v); //顶点v出队列
        for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
            //检测v所有邻接点
            if(!visited[w]){ //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入队列
            }
    }
}
```

	1	2	3	4	5	6	7	8
visited	true	true	true	true	true	true	true	true

王道考研/CSKAOYAN.COM

代码实现

初始都为false



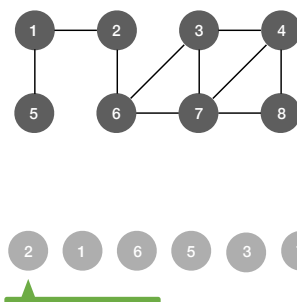
```
bool visited[MAX_VERTEX_NUM]; //访问标记数组

//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); //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入队列
            }
    }
}
```

	1	2	3	4	5	6	7	8
visited	true	true	true	true	true	true	true	true

王道考研/CSKAOYAN.COM

广度优先遍历序列



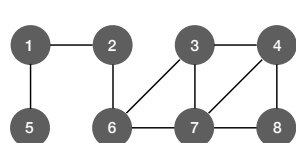
从顶点1出发得到的广度优先遍历序列:
1, 2, 5, 6, 3, 7, 4, 8

从顶点3出发得到的广度优先遍历序列:
3, 4, 6, 7, 8, 2, 1, 5

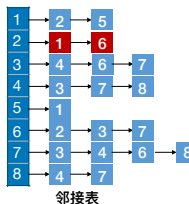
从顶点2出发得到的广度优先遍历序列

王道考研/CSKAOYAN.COM

遍历序列的可变性



	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



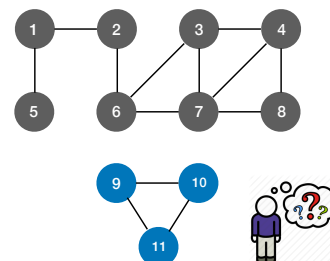
从顶点2出发得到的广度优先遍历序列

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
同一个图邻接表表示方式不唯一，因此广度优先遍历序列不唯一

广度优先遍历序列：
2, 6, 1, ...

王道考研/CSKAQYAN.COM

算法存在的问题



如果是非连通图，则无法遍历完所有结点

初始都为false

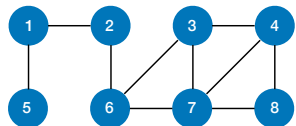
```
bool visited[MAX_VERTEX_NUM]; // 访问标记数组

// 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); // 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入队列
            }
        }
    }
}
```

visited: true true true true true true true true true false false false

王道考研/CSKAQYAN.COM

BFS算法 (Final版)



如果是非连通图，则无法遍历完所有结点

visited: false false false false false false false false false false false

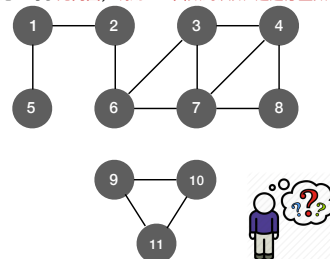
```
bool visited[MAX_VERTEX_NUM]; // 访问标记数组
void BFSTraverse(Graph G) { // 对图G进行广度优先遍历
    for (i = 0; i < G.vexnum; ++i)
        visited[i] = FALSE; // 访问标记数组初始化
    InitQueue(Q); // 初始化辅助队列Q
    for (i = 0; i < G.vexnum; ++i) // 从0号顶点开始遍历
        if (!visited[i]) // 对每个连通分量调用一次BFS
            BFS(G, i); // vi未访问过，从vi开始BFS
}

// 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); // 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入队列
            }
        }
    }
}
```

王道考研/CSKAQYAN.COM

BFS算法 (Final版)

结论：对于无向图，调用BFS函数的次数=连通分量数



如果是非连通图，则无法遍历完所有结点

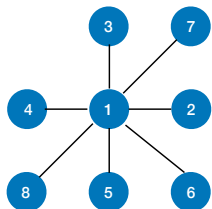
visited: true true true true true true true true true true true

```
bool visited[MAX_VERTEX_NUM]; // 访问标记数组
void BFSTraverse(Graph G) { // 对图G进行广度优先遍历
    for (i = 0; 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Q
    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未访问过，从vi开始BFS
}

// 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); // 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入队列
            }
        }
    }
}
```

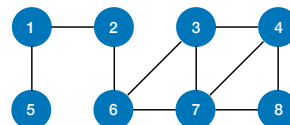
王道考研/CSKAQYAN.COM

复杂度分析



空间复杂度：最坏情况，辅助队列大小为 $O(|V|)$

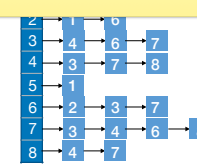
复杂度分析



	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

其实对于无向图来说，应该是 $O(2|E|)$ 的时间



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 = $O(|V|^2)$

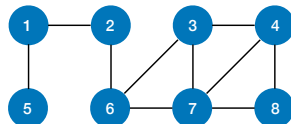
邻接表存储的图：

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，

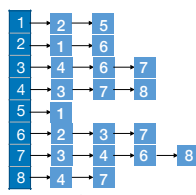
时间复杂度 = $O(|V| + |E|)$

广度优先生成树



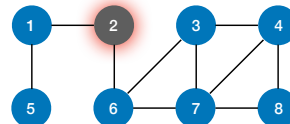
	1	2	3	4	5	6	7	8
1	0	1	0	0	1	0	0	0
2	1	0	0	0	0	1	0	0
3	0	0	0	1	0	1	1	0
4	0	0	1	0	0	0	1	1
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	1	0
7	0	0	1	1	0	1	0	1
8	0	0	0	1	0	0	1	0

邻接矩阵



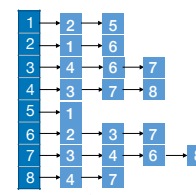
邻接表

广度优先生成树



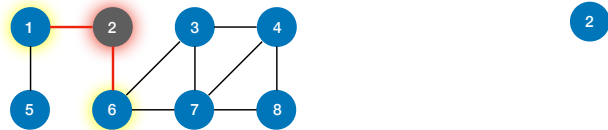
	1	2	3	4	5	6	7	8
1	0	1	0	0	1	0	0	0
2	1	0	0	0	0	1	0	0
3	0	0	0	1	0	1	1	0
4	0	0	1	0	0	0	1	1
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	1	0
7	0	0	1	1	0	1	0	1
8	0	0	0	1	0	0	1	0

邻接矩阵



邻接表

广度优先生成树



	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

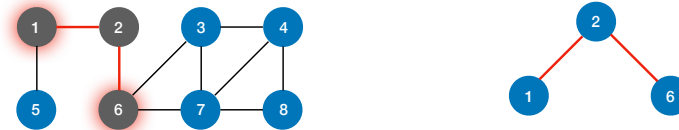
邻接矩阵

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

邻接表

王道考研/CSKAOYAN.COM

广度优先生成树



	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	1
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

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

邻接表

王道考研/CSKAOYAN.COM

广度优先生成树



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

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

邻接表

王道考研/CSKAOYAN.COM

广度优先生成树



	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	1
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

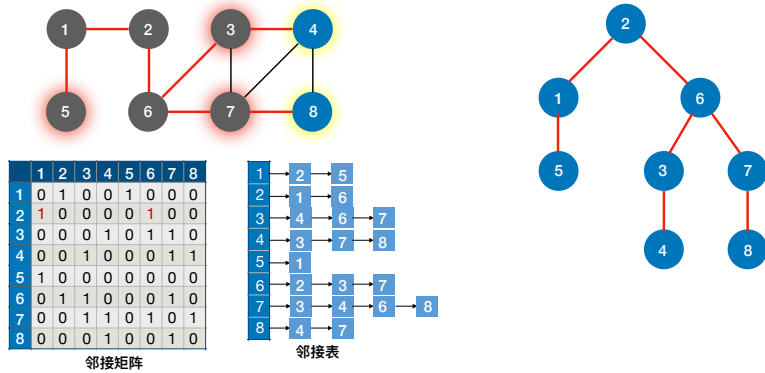
邻接矩阵

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

邻接表

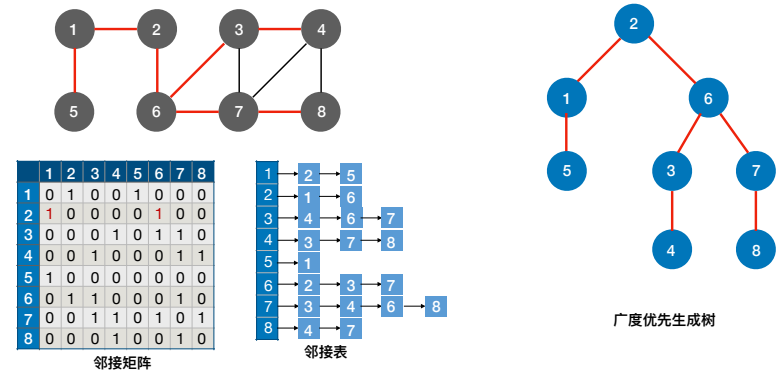
王道考研/CSKAOYAN.COM

广度优先生成树



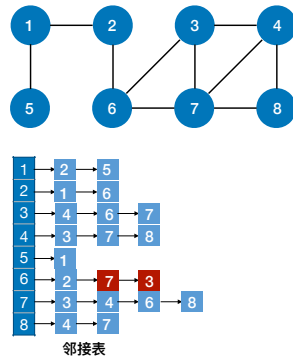
王道考研/CSKAOYAN.COM

广度优先生成树



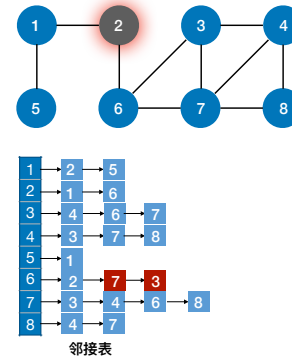
王道考研/CSKAOYAN.COM

广度优先生成树



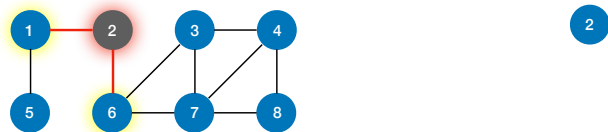
王道考研/CSKAOYAN.COM

广度优先生成树



王道考研/CSKAOYAN.COM

广度优先生成树

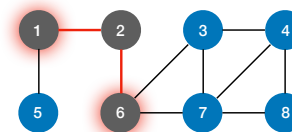


1	2	5
2	1	6
3	4	6
4	3	7
5	1	
6	2	7
7	3	4
8	4	7

邻接表

王道考研/CSKAOYAN.COM

广度优先生成树

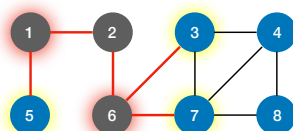


1	2	5
2	1	6
3	4	6
4	3	7
5	1	
6	2	7
7	3	4
8	4	7

邻接表

王道考研/CSKAOYAN.COM

广度优先生成树

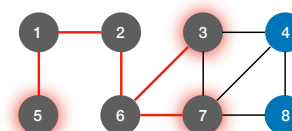


1	2	5
2	1	6
3	4	6
4	3	7
5	1	
6	2	7
7	3	4
8	4	7

邻接表

王道考研/CSKAOYAN.COM

广度优先生成树

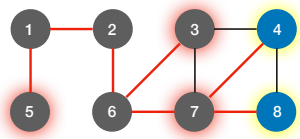


1	2	5
2	1	6
3	4	6
4	3	7
5	1	
6	2	7
7	3	4
8	4	7

邻接表

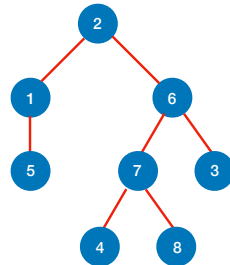
王道考研/CSKAOYAN.COM

广度优先生成树

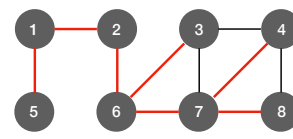


1	2	5
2	1	6
3	4	6
4	3	7
5	1	7
6	2	7
7	3	4
8	4	7

邻接表

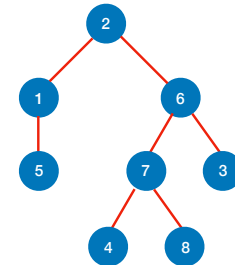


广度优先生成树

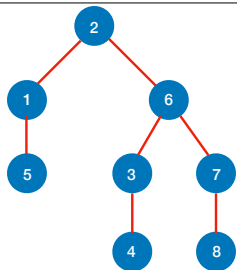


1	2	5
2	1	6
3	4	6
4	3	7
5	1	7
6	2	7
7	3	4
8	4	7

邻接表



广度优先生成树



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

邻接矩阵

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

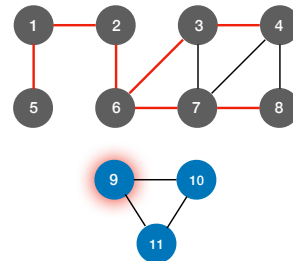
邻接表

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。

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

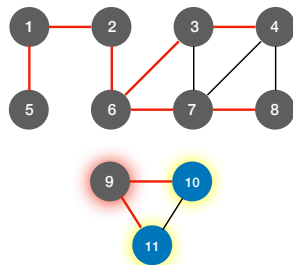
邻接表

广度优先生成森林

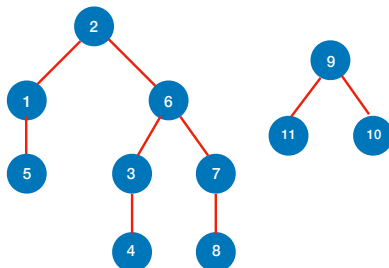


对非连通图的广度优先遍历，可得到广度优先生成森林

广度优先生成森林



对非连通图的广度优先遍历，可得到广度优先生成森林

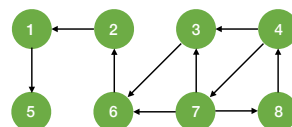


王道考研/CSKAOYAN.COM

练习：有向图的BFS过程

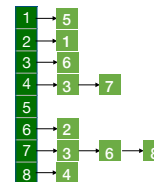
思考：

1. 从1出发，需要调用几次BFS函数？
2. 从7出发，需要调用几次BFS函数？



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

邻接矩阵



邻接表

```
bool visited[MAX_VERTEX_NUM]; // 访问标记数组
void BFSTraverse(Graph G) { // 对图G进行广度优先遍历
    for(i=0; 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Q
    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); // v1未访问过，从v1开始BFS
}

// 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); // 顶点v入队列Q
    while(!IsEmpty(Q)) { // 顶点v出队列
        Dequeue(Q, v); // 检测v所有邻接点
        for(w=FirstNeighbor(G, v); w!=0; w=NextNeighbor(G, v, w))
            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入队列
            }
    }
}
```

王道考研/CSKAOYAN.COM

知识回顾与重要考点

BFS求单元最短路径问题放在6.4.2讲解

图的BFS

类似于树的层序遍历（广度优先遍历）

需要一个辅助队列

如何从一个结点找到与之邻接的其他顶点

visited 数组防止重复访问

如何处理非连通图

空间复杂度：O(|V|) —— 辅助队列

访问结点的时间 + 访问所有边的时间

邻接矩阵：O(|V|^2)

邻接表：O(|V|+|E|)

由广度优先遍历确定的树

邻接表存储的图表示方式不唯一，遍历序列、生成树也不唯一

遍历非连通图可得广度优先生成森林

王道考研/CSKAOYAN.COM