

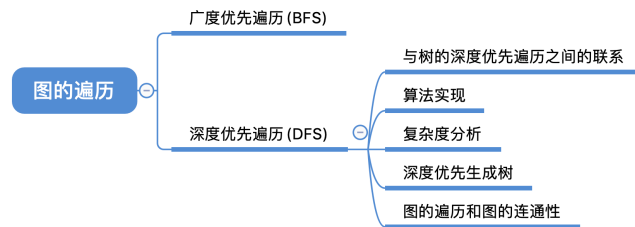
本节内容

图的遍历

DFS

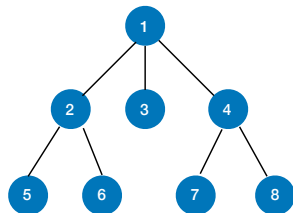
王道考研/CSKAOYAN.COM

知识总览



王道考研/CSKAOYAN.COM

树的深度优先遍历

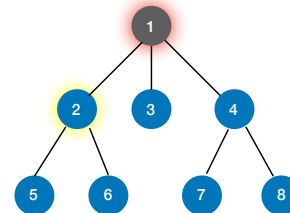


```
// 树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); // 访问根节点
        while(R还有下一个子树T)
            PreOrder(T); // 先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

王道考研/CSKAOYAN.COM

树的深度优先遍历

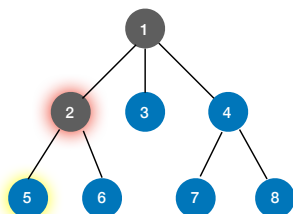


```
// 树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); // 访问根节点
        while(R还有下一个子树T)
            PreOrder(T); // 先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

王道考研/CSKAOYAN.COM

树的深度优先遍历

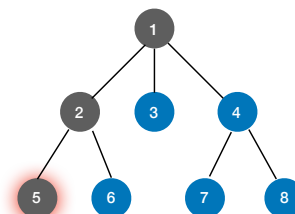


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

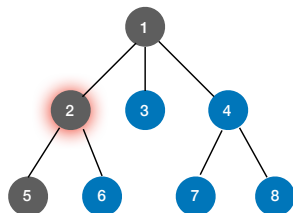


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

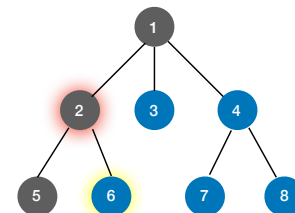


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

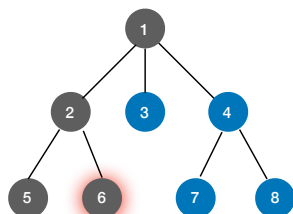


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

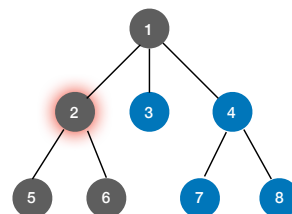
树的深度优先遍历



```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

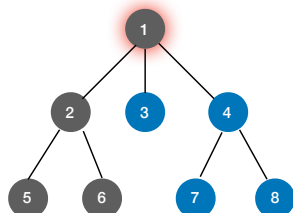
树的深度优先遍历



```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

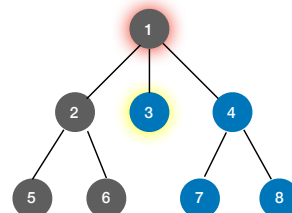
树的深度优先遍历



```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

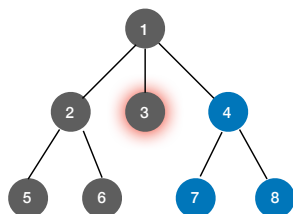
树的深度优先遍历



```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

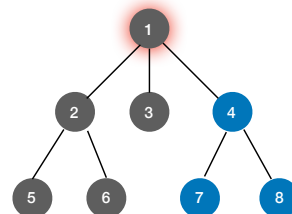


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下一层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

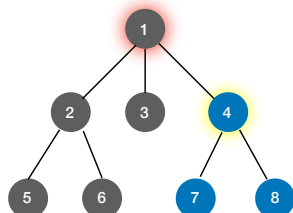


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下一层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

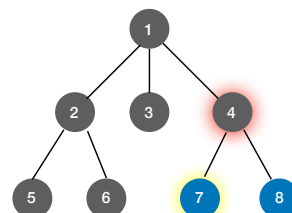


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下一层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

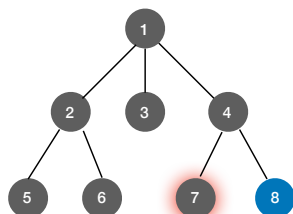


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下一层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

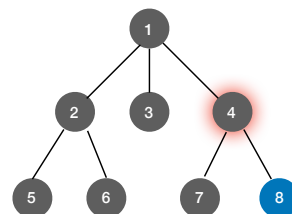


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

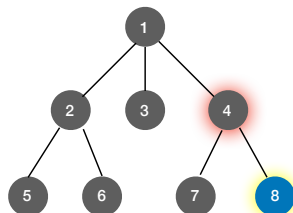


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

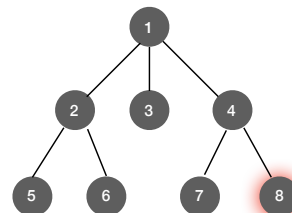


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

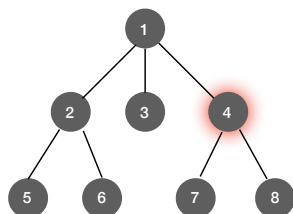


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

王道考研/CSKAQYAN.COM

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

树的深度优先遍历

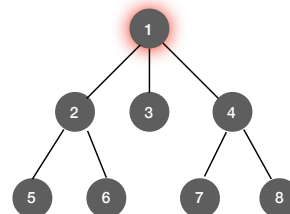


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

王道考研/CSKAQYAN.COM

树的深度优先遍历

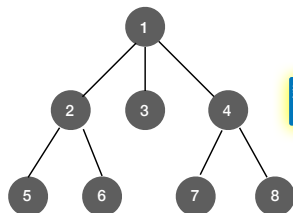


```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

王道考研/CSKAQYAN.COM

树的深度优先遍历



新找到的相邻结点一定
是没有访问过的

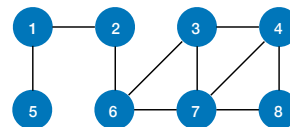
```
//树的先根遍历
void PreOrder(TreeNode *R){
    if (R!=NULL){
        visit(R); //访问根节点
        while(R还有下一个子树T)
            PreOrder(T); //先根遍历下一棵子树
    }
}
```

树的深度优先遍历（先根、后根）：
从根节点出发，能往更深处走就尽量往深处走。每当访问一个结点的时候，要检查是否还有与当前结点相邻的且没有被访问过的结点，如果有的话就往下层钻。
图的深度优先遍历类似于树的先根遍历。

先根遍历序列：1, 2, 5, 6, 3, 4, 7, 8

王道考研/CSKAQYAN.COM

图的深度优先遍历



函数调用栈

```
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; //设已访问标记
    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
    }
}
```

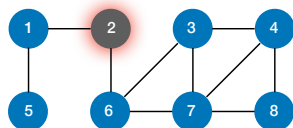
	1	2	3	4	5	6	7	8
visited	false	false	false	false	false	false	false	false

初始都为false

王道考研/CSKAQYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

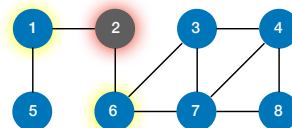


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

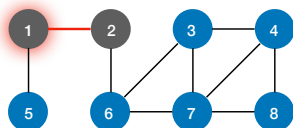


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

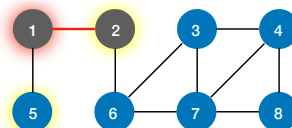


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

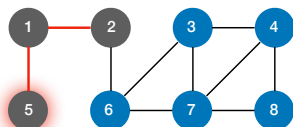


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

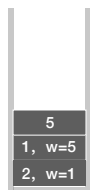
初始都为false



```
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; //设已访问标记
    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
}
```

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

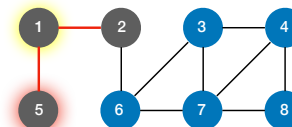


函数调用栈

王道考研/CSKAQYAN.COM

图的深度优先遍历

初始都为false



```
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; //设已访问标记
    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
}
```

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

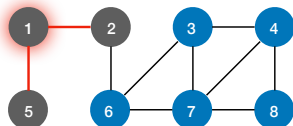


函数调用栈

王道考研/CSKAQYAN.COM

图的深度优先遍历

初始都为false



```
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; //设已访问标记
    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
}
```

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

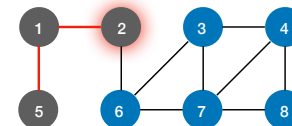


函数调用栈

王道考研/CSKAQYAN.COM

图的深度优先遍历

初始都为false



```
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; //设已访问标记
    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
}
```

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

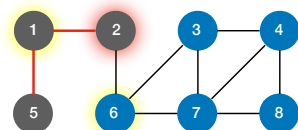


函数调用栈

王道考研/CSKAQYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

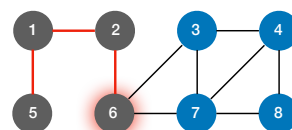
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	false	false	false

函数调用栈
2, w=1

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

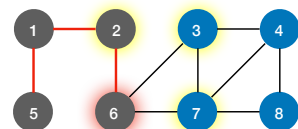
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	true	false	false

函数调用栈
6
2, w=6

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

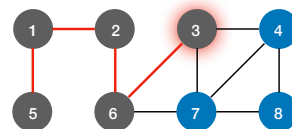
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	true	false	false

函数调用栈
6
2, w=6

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

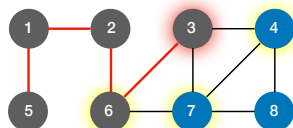
	1	2	3	4	5	6	7	8
visited	true	true	true	false	true	true	false	false

函数调用栈
3
6, w=3
2, w=6

王道考研/CSKAOYAN.COM

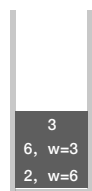
图的深度优先遍历

初始都为false



```
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;          //设已访问标记
    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
}
```

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

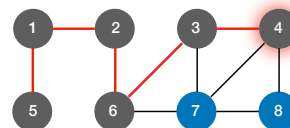


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;          //设已访问标记
    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
}
```

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

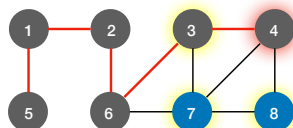


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;          //设已访问标记
    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
}
```

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

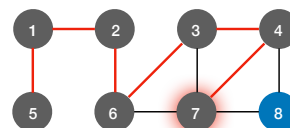


函数调用栈

王道考研/CSKAOYAN.COM

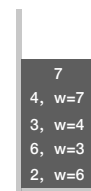
图的深度优先遍历

初始都为false



```
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;          //设已访问标记
    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
}
```

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

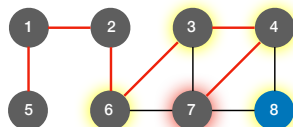


函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

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

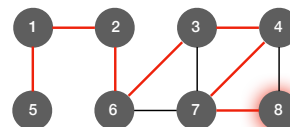
7
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

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

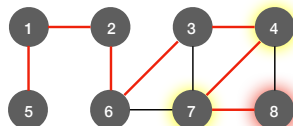
8
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

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

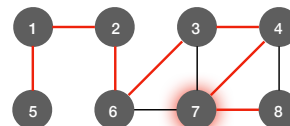
8
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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);
        }
}
```

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

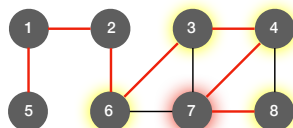
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

函数调用栈

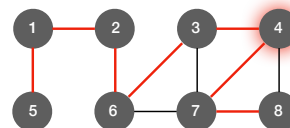
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

函数调用栈

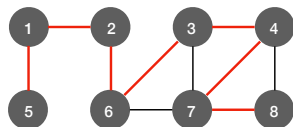
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

王道考研/CSKAOYAN.COM

图的深度优先遍历

初始都为false



```
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;           //设已访问标记
    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
}
```

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

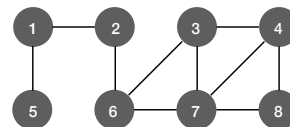
从2出发的深度遍历序列: 2, 1, 5, 6, 3, 4, 7, 8

函数调用栈

王道考研/CSKAOYAN.COM

算法存在的问题

初始都为false



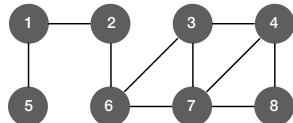
```
bool visited[MAX_VERTEX_NUM]; //访问标记数组
void DFS(Graph G,int v){      //从顶点v出发，深度优先遍历图G
    visit(v);                  //访问顶点v
    visited[v]=TRUE;           //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){      //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
}
```

	1	2	3	4	5	6	7	8	9	10	11
visited	true	true	true	true	true	true	true	true	false	false	false

如果是非连通图，则无法遍历完所有结点

王道考研/CSKAOYAN.COM

DFS算法 (Final版)



如果是非连通图，则无法遍历完所有结点

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

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; //设已访问标记
    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
}
```

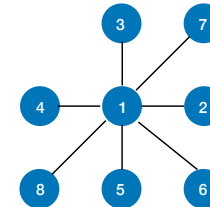
	1	2	3	4	5	6	7	8	9	10	11
visited	true	true	true	true	true	true	true	true	false	false	false

王道考研/CSKAOYAN.COM

复杂度分析



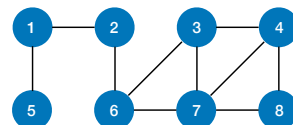
空间复杂度：来自函数调用栈，最坏情况，递归深度为 $O(|V|)$



空间复杂度：最好情况， $O(1)$

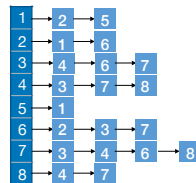
王道考研/CSKAOYAN.COM

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	1	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

时间复杂度=访问各结点所需时间+探索各条边所需时间

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：

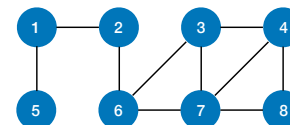
访问 $|V|$ 个顶点需要 $O(|V|)$ 的时间

查找各个顶点的邻接点共需要 $O(|E|)$ 的时间，

时间复杂度= $O(|V|+|E|)$

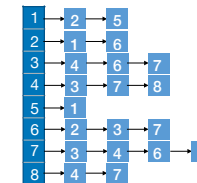
王道考研/CSKAOYAN.COM

深度优先遍历序列



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

从2出发的深度优先遍历序列：2, 1, 5, 6, 3, 4, 7, 8

从3出发的深度优先遍历序列：3, 4, 7, 6, 2, 1, 5, 8

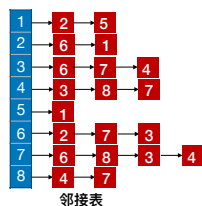
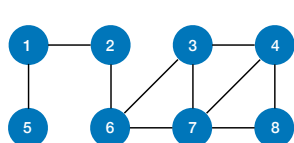
从1出发的深度优先遍历序列：1, 2, 6, 3, 4, 7, 8, 5

在邻接表中出现的顺序是可变的，因此如果用这种数据结构存储树，那么可能会有不同的遍历序列

王道考研/CSKAOYAN.COM

深度优先遍历序列

在邻接表中出现的顺序是可变的，因此如果用这种数据结构存储树，那么可能会有不同的遍历序列



从2出发的深度优先遍历序列：2, 6, 7, 8, 4, 3, 1, 5

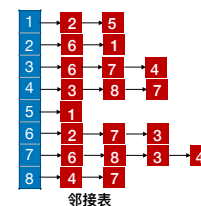
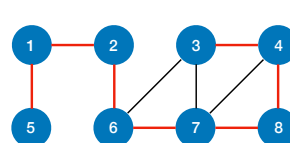
从3出发的深度优先遍历序列？

从1出发的深度优先遍历序列？

王道考研/CSKAQYAN.COM

深度优先遍历序列

在邻接表中出现的顺序是可变的，因此如果用这种数据结构存储树，那么可能会有不同的遍历序列



从2出发的深度优先遍历序列：2, 6, 7, 8, 4, 3, 1, 5

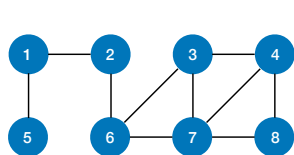
从3出发的深度优先遍历序列？

从1出发的深度优先遍历序列？

王道考研/CSKAQYAN.COM

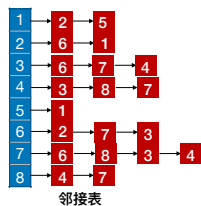
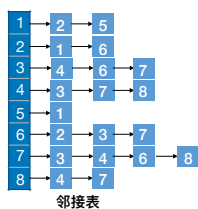
深度优先遍历序列

在邻接表中出现的顺序是可变的，因此如果用这种数据结构存储树，那么可能会有不同的遍历序列



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

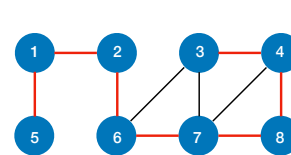
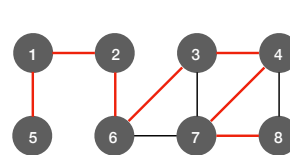
邻接矩阵



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
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

王道考研/CSKAQYAN.COM

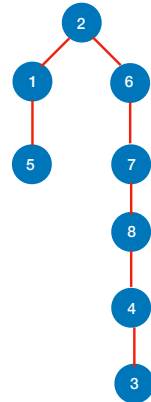
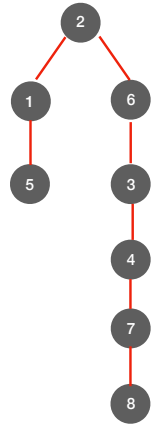
深度优先生成树



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

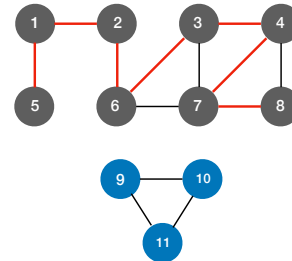
王道考研/CSKAQYAN.COM

深度优先生成树



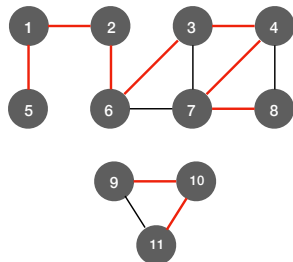
王道考研/CSKAOYAN.COM

深度优先生成森林



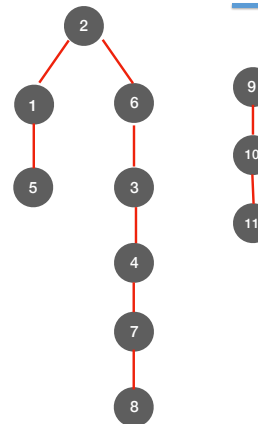
王道考研/CSKAOYAN.COM

深度优先生成森林



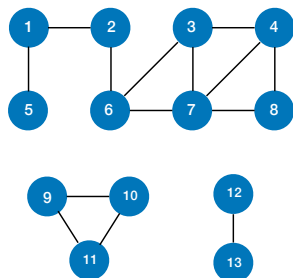
王道考研/CSKAOYAN.COM

深度优先生成森林



王道考研/CSKAOYAN.COM

图的遍历与图的连通性



对**无向图**进行BFS/DFS遍历
调用BFS/DFS函数的次数=连通分量数

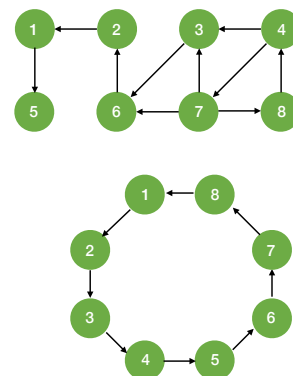
对于**连通图**，只需调用1次 BFS/DFS

通分量数

对于有向图，如果是个强连通图，通过一次BFS或DFS遍历，所有顶点都能被访问到。
如果不是强连通图，那么要看顶点是否能找到路径，如果从某点BFS或DFS就可以遍历所有顶点。

王道考研/CSKAOYAN.COM

图的遍历与图的连通性



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

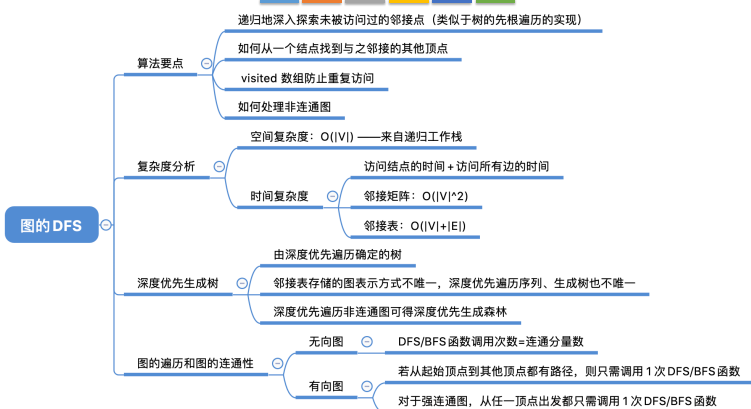
通分量数

对于有向图，如果是个强连通图，通过一次BFS或DFS遍历，所有顶点都能被访问到。
如果不是强连通图，那么要看顶点是否能找到路径，如果从某点BFS或DFS就可以遍历所有顶点。

对于**强连通图**，从任一结点出发都只需调用1次 BFS/DFS

王道考研/CSKAOYAN.COM

知识回顾与重要考点



王道考研/CSKAOYAN.COM