

本节内容

图的遍历

DFS

知识总览

图的遍历

广度优先遍历 (BFS)

与树的深度优先遍历之间的联系

算法实现

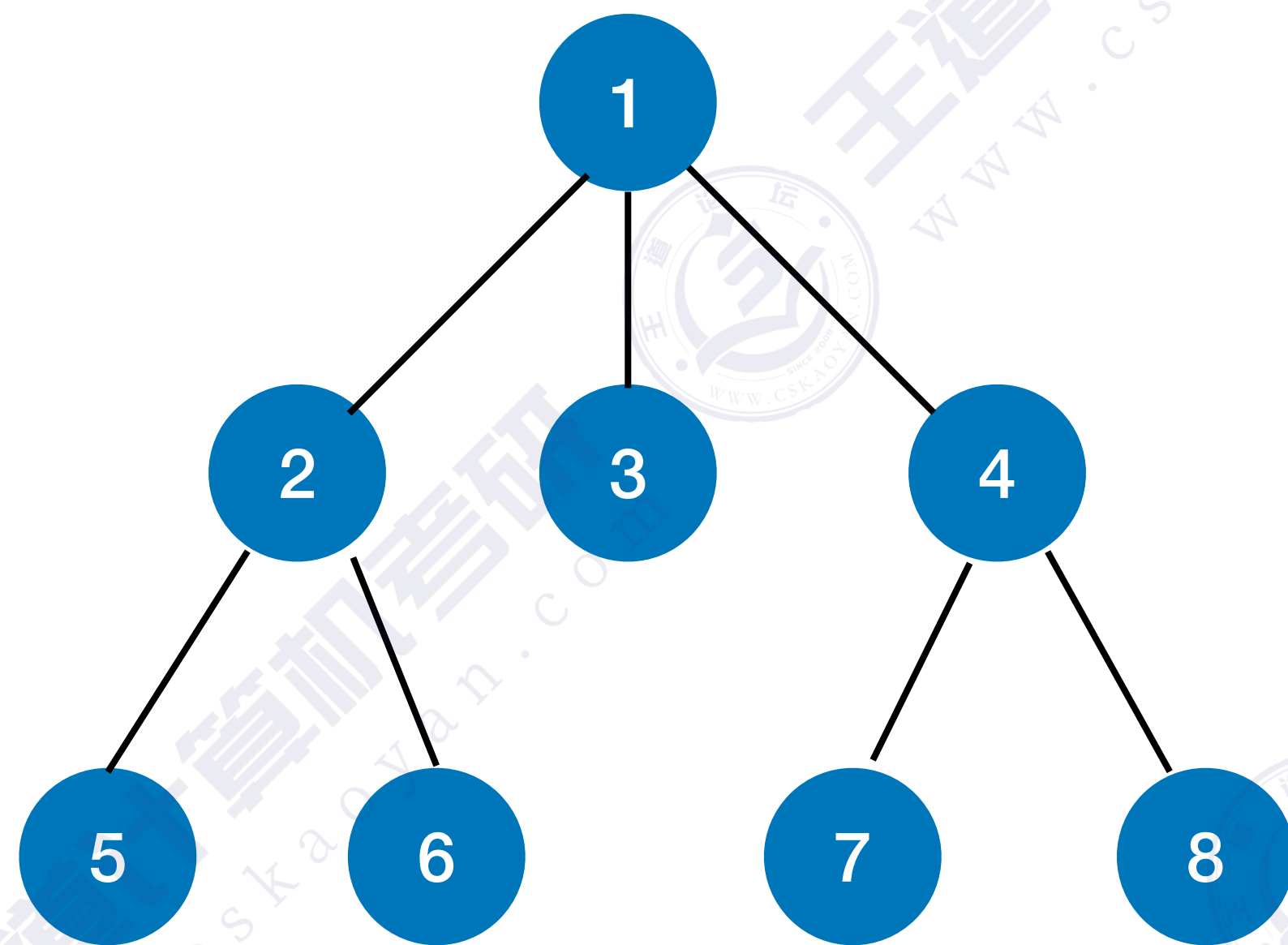
深度优先遍历 (DFS)

复杂度分析

深度优先生成树

图的遍历和图的连通性

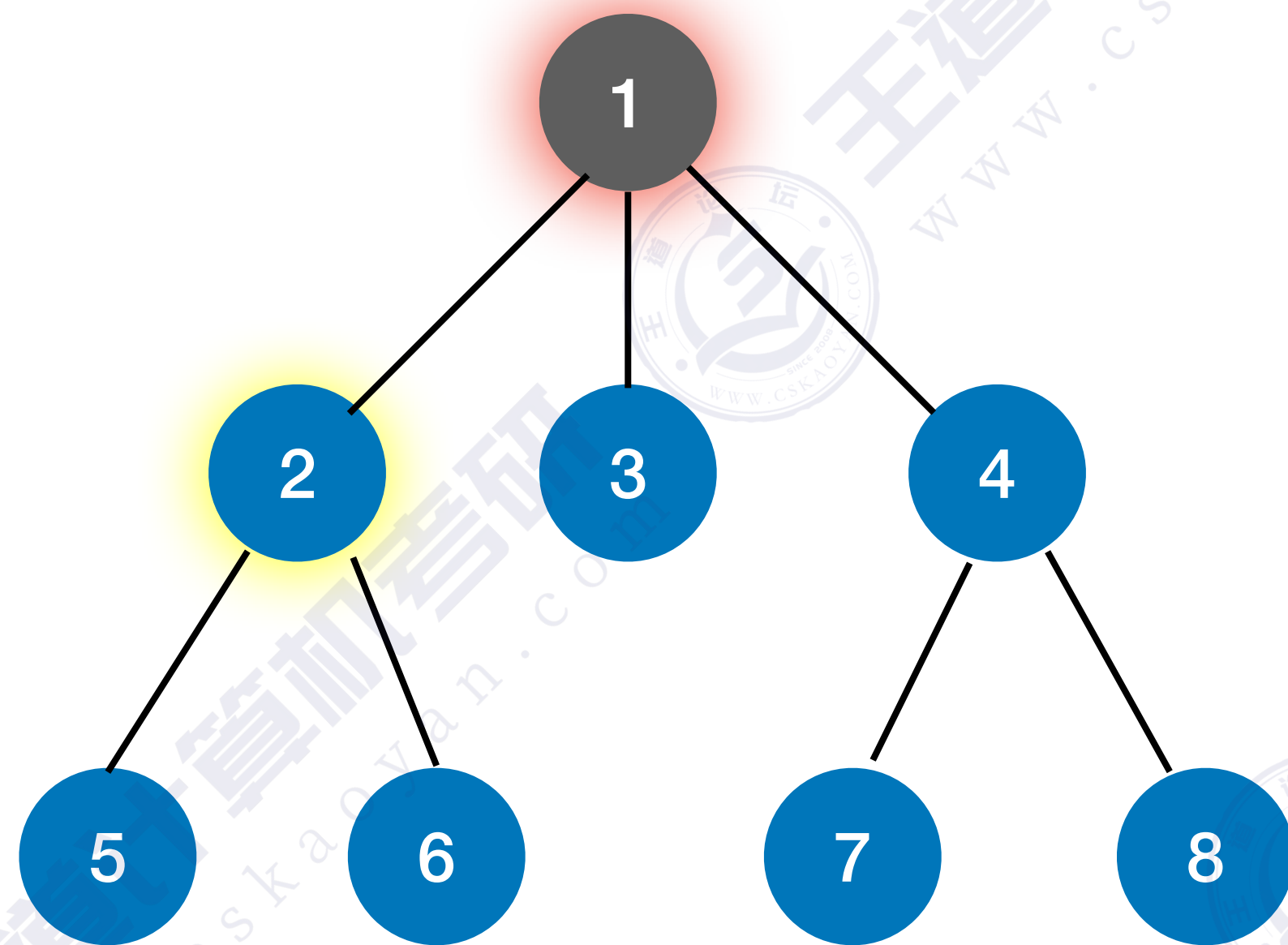
树的深度优先遍历



//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  
    }  
}
```

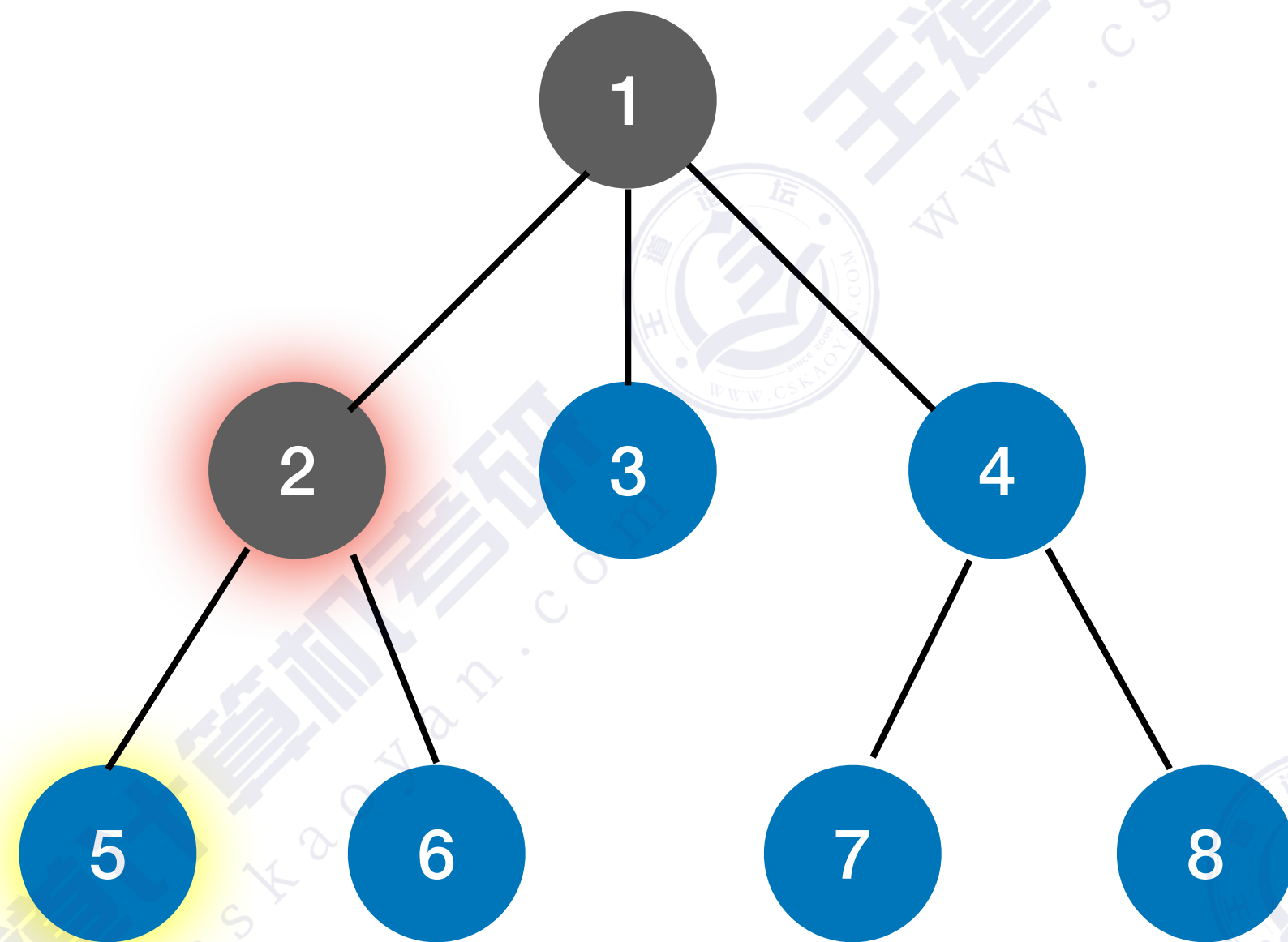

树的深度优先遍历



//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  
    }  
}
```

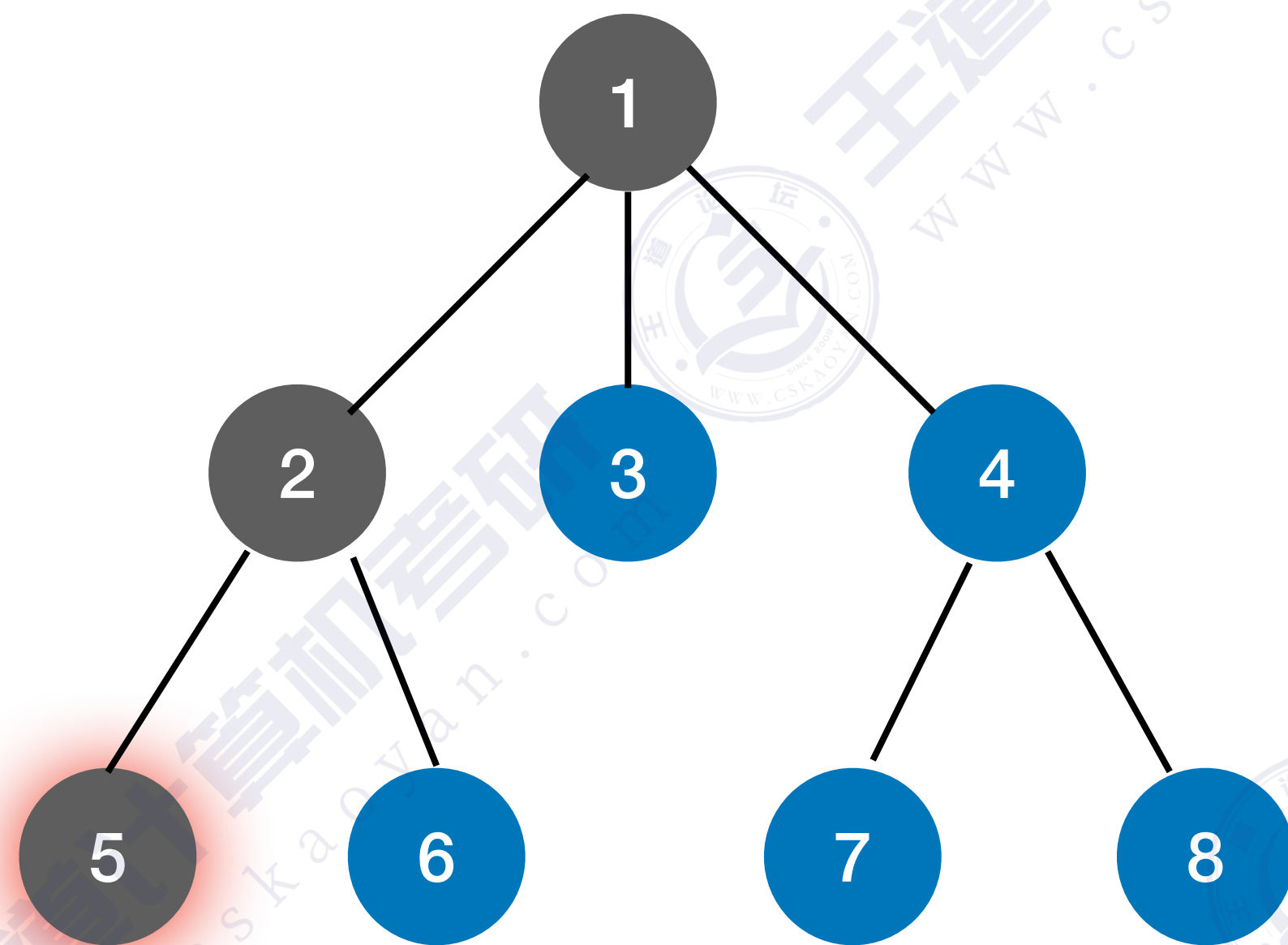
树的深度优先遍历



//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  
    }  
}
```

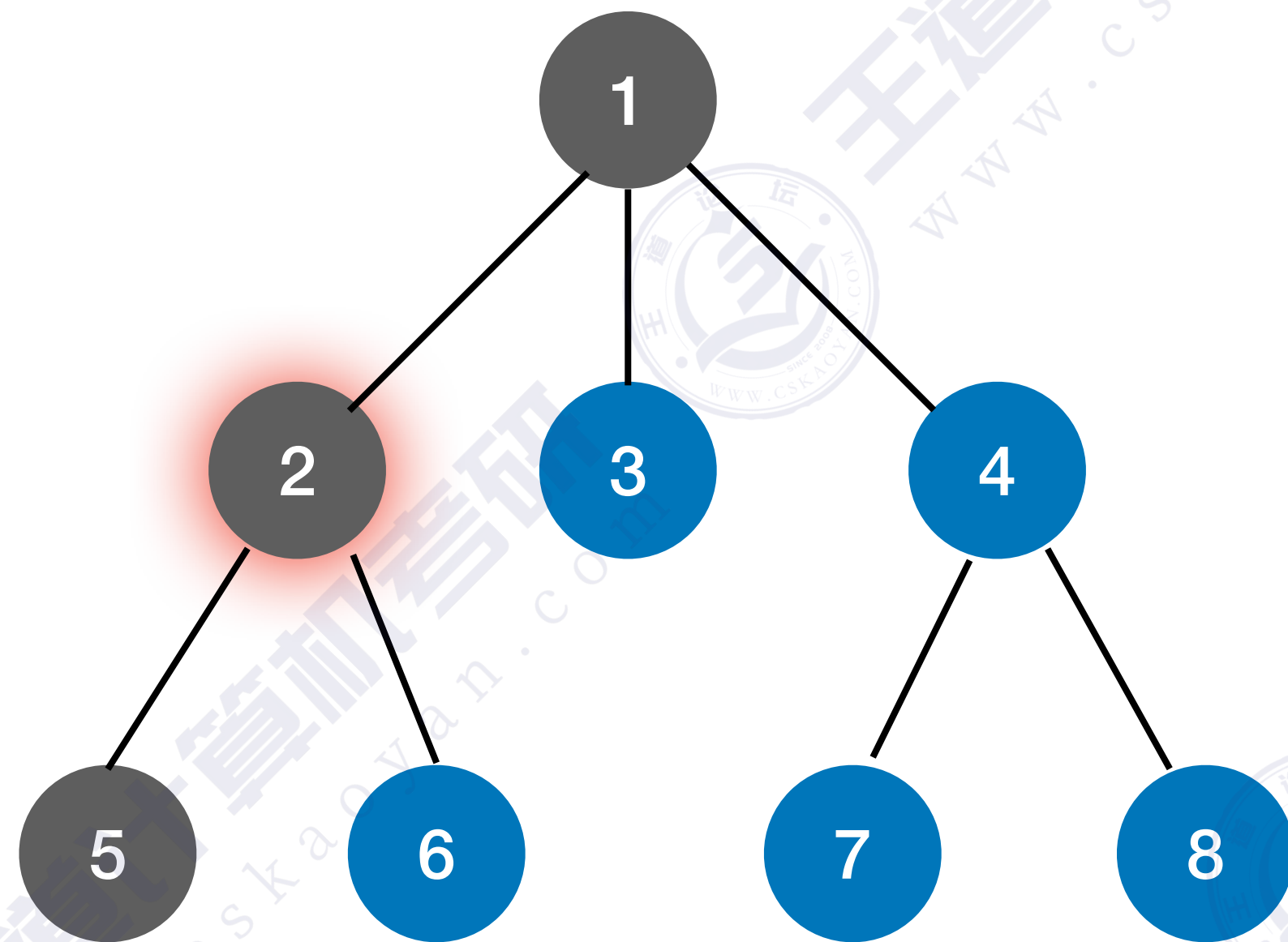
树的深度优先遍历



//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  
    }  
}
```

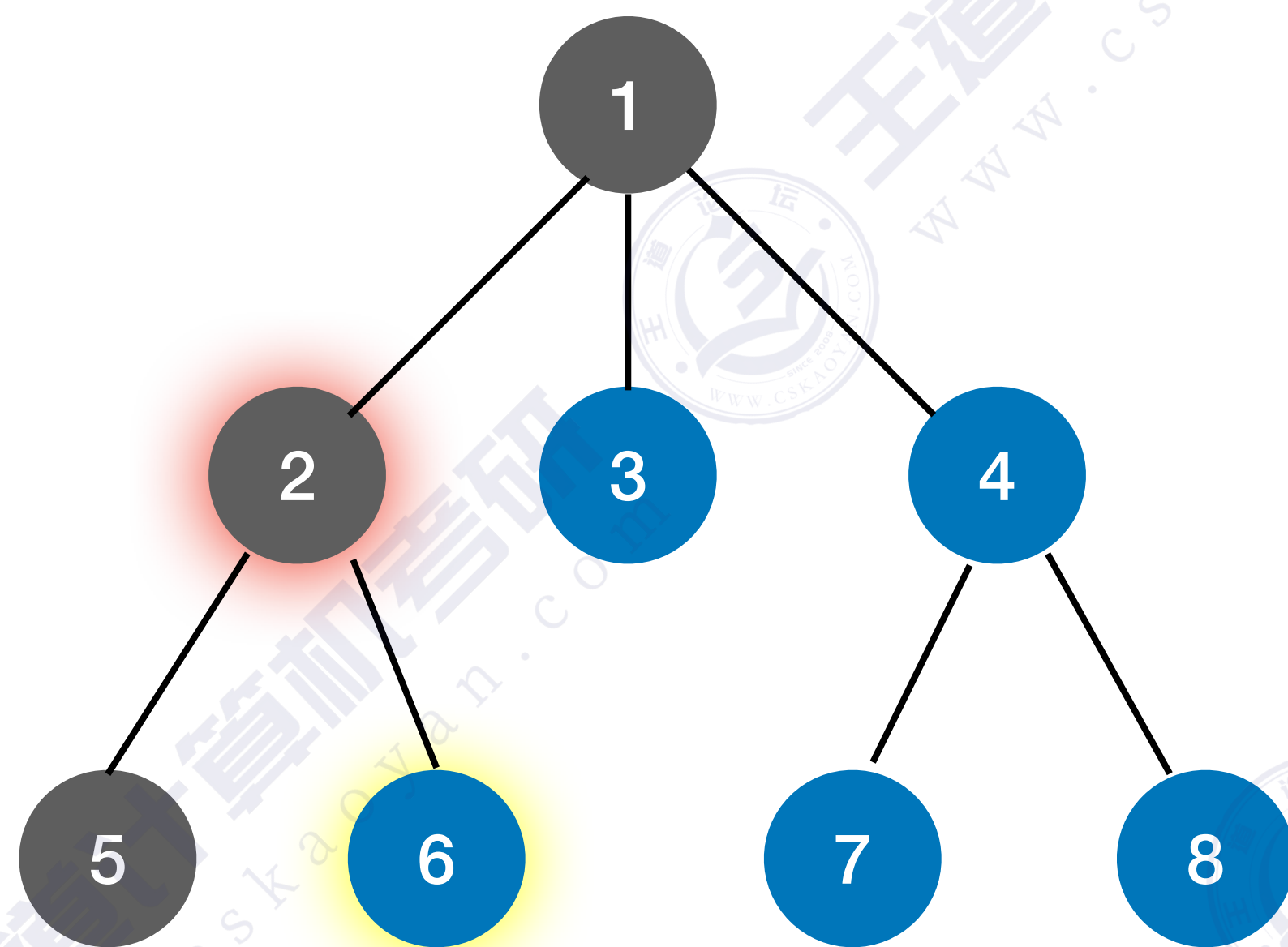

树的深度优先遍历



//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  
    }  
}
```

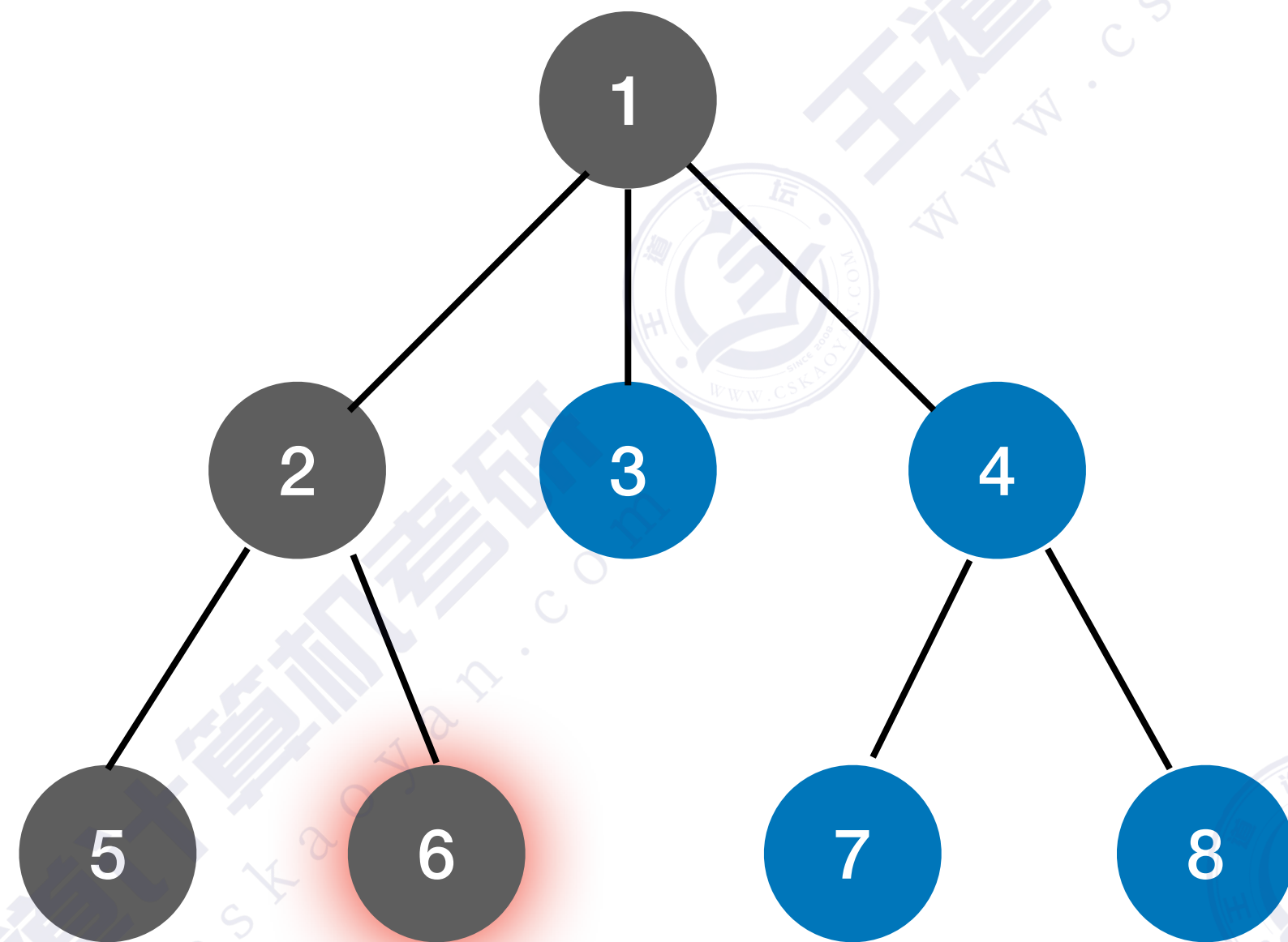
树的深度优先遍历



//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  
    }  
}
```

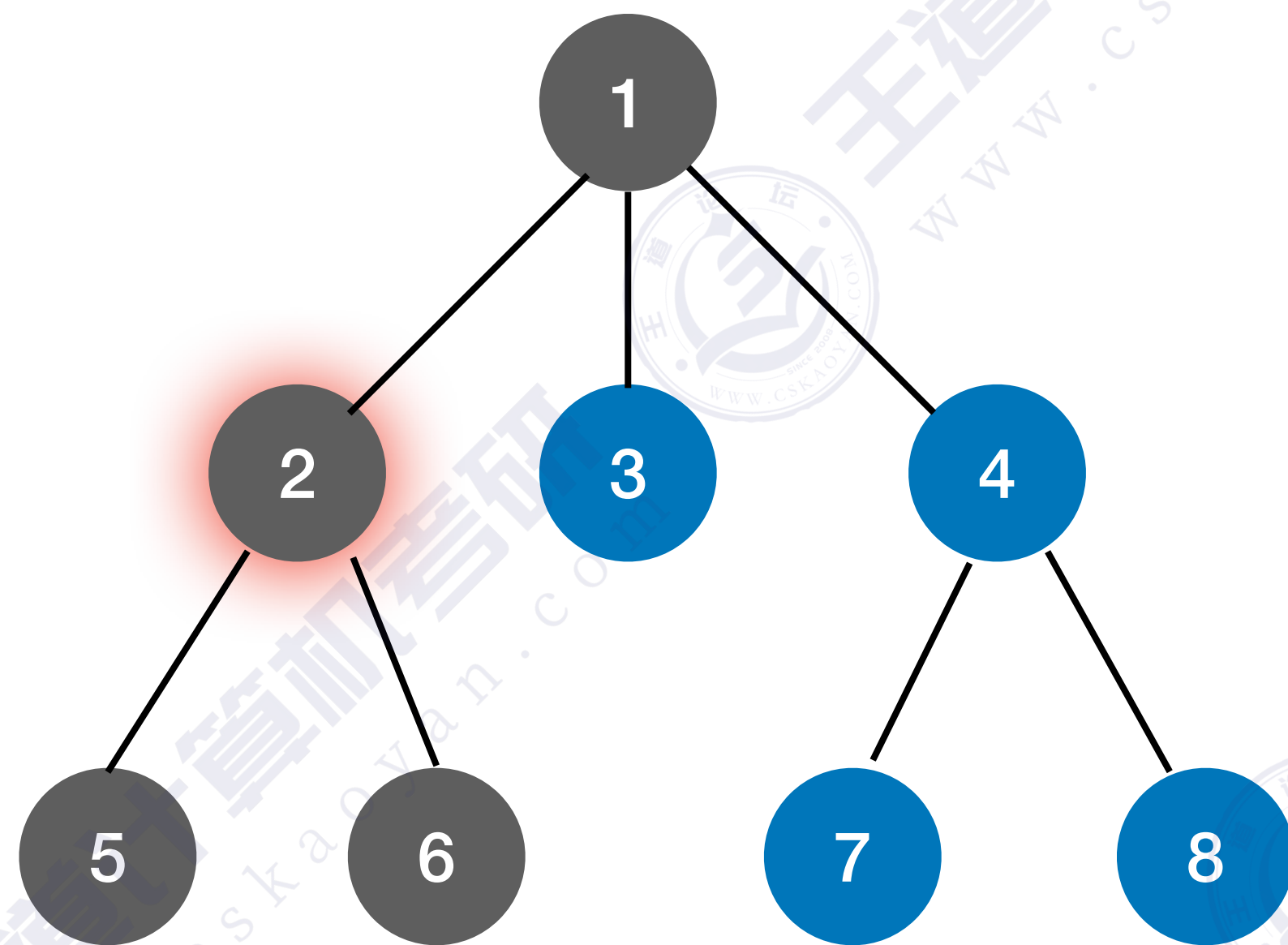

树的深度优先遍历



//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  
    }  
}
```

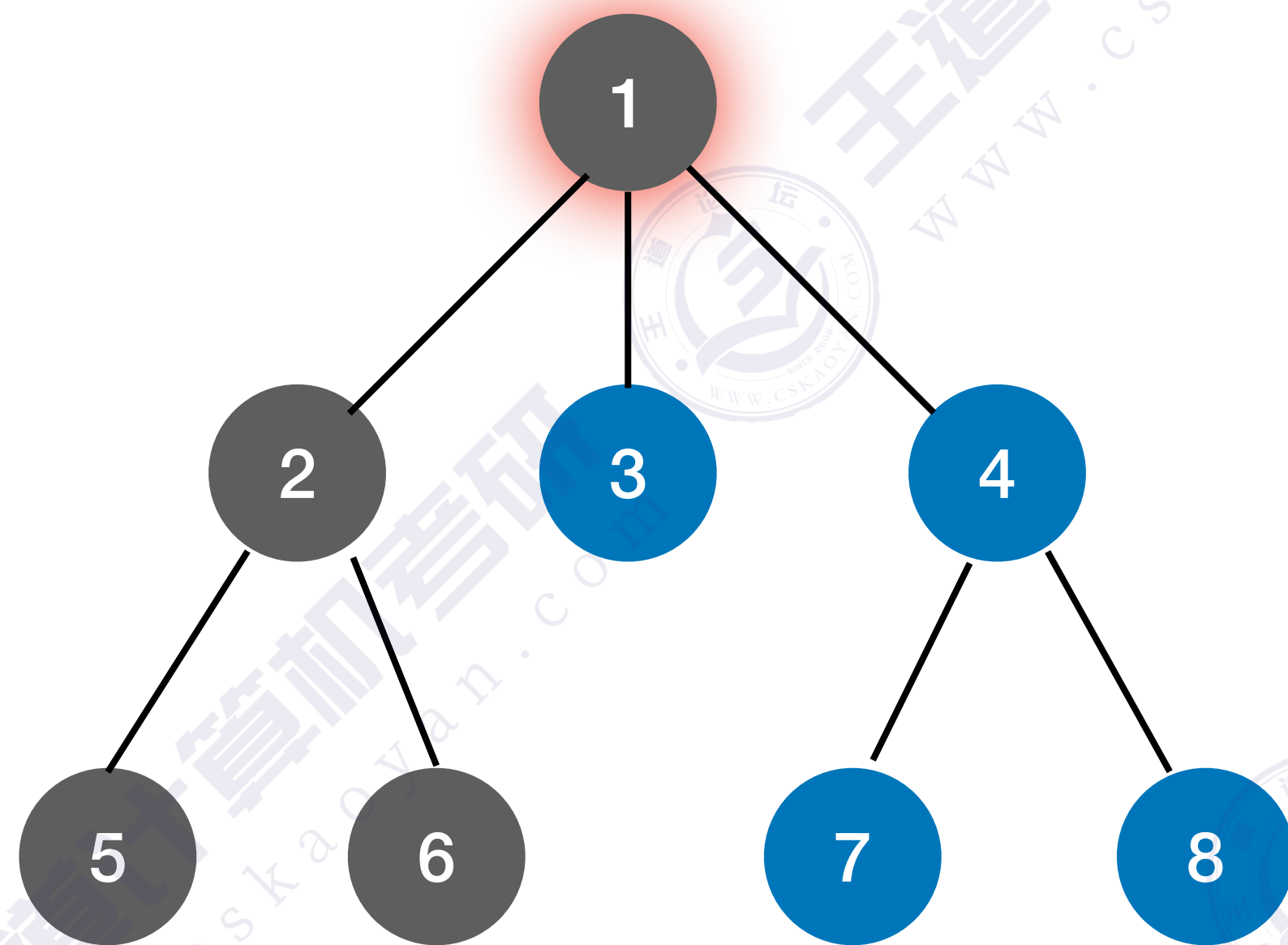
树的深度优先遍历



//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  
    }  
}
```

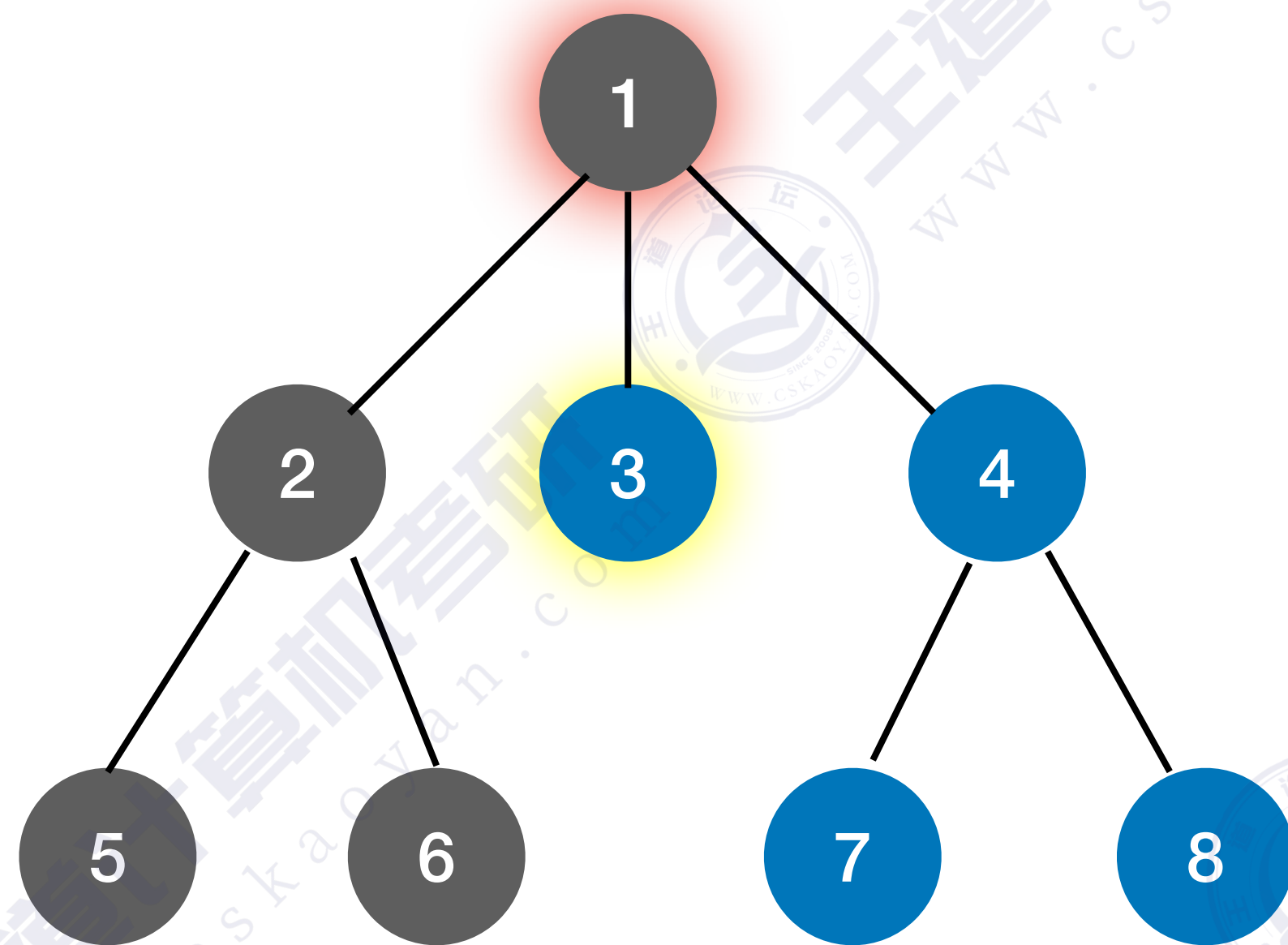
树的深度优先遍历



//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  
    }  
}
```

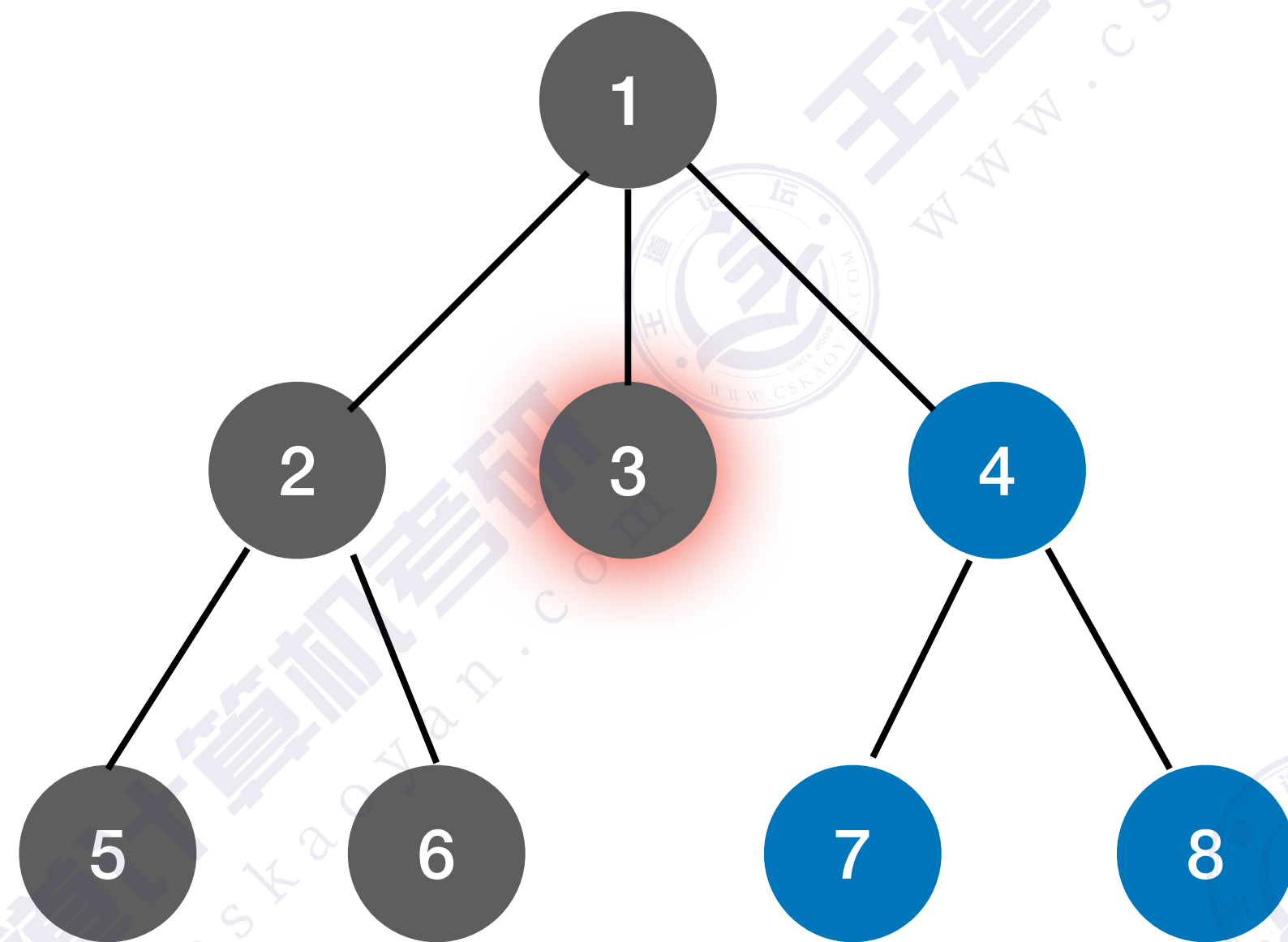

树的深度优先遍历



//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  
    }  
}
```

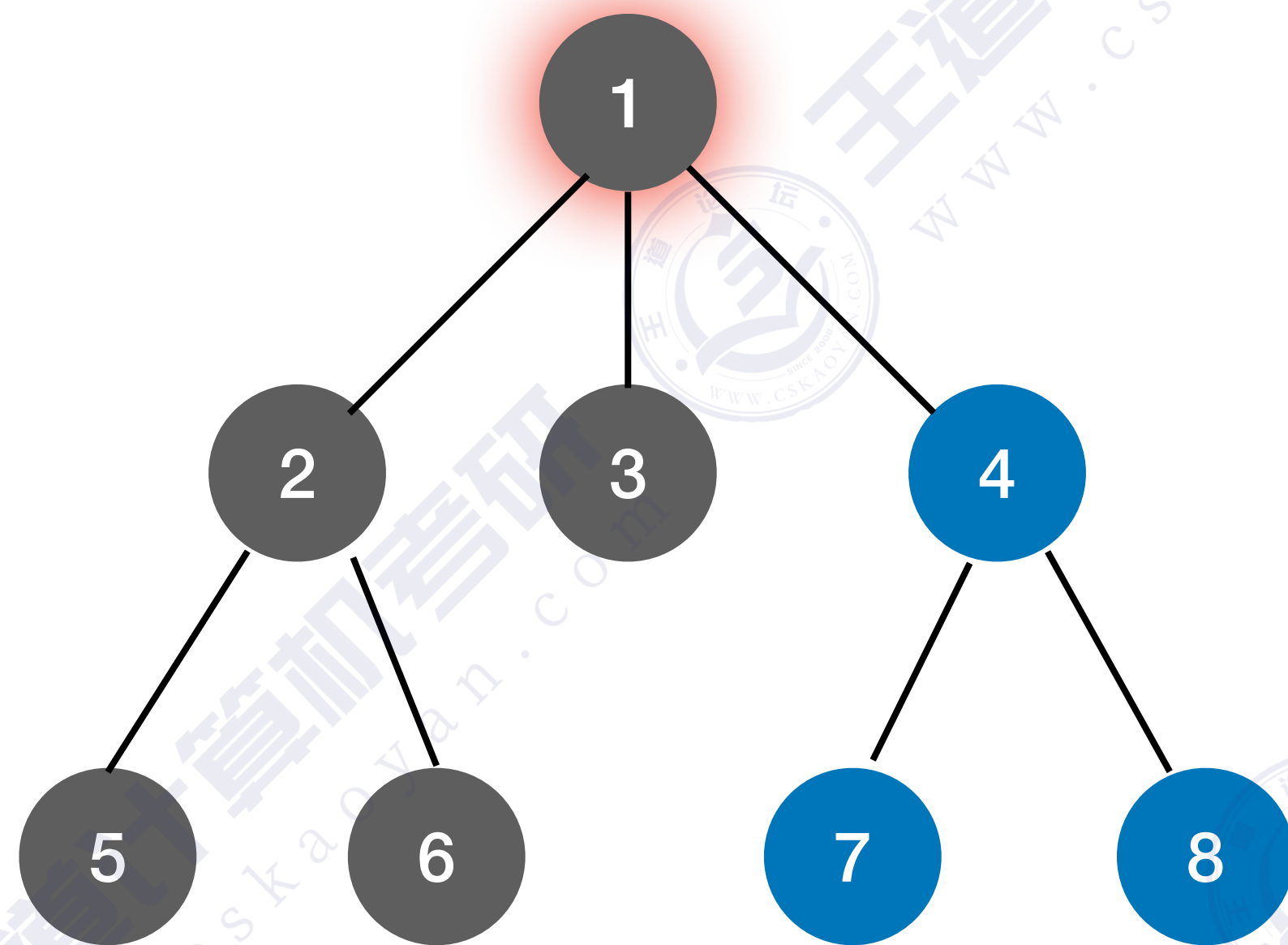
树的深度优先遍历



//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  
    }  
}
```

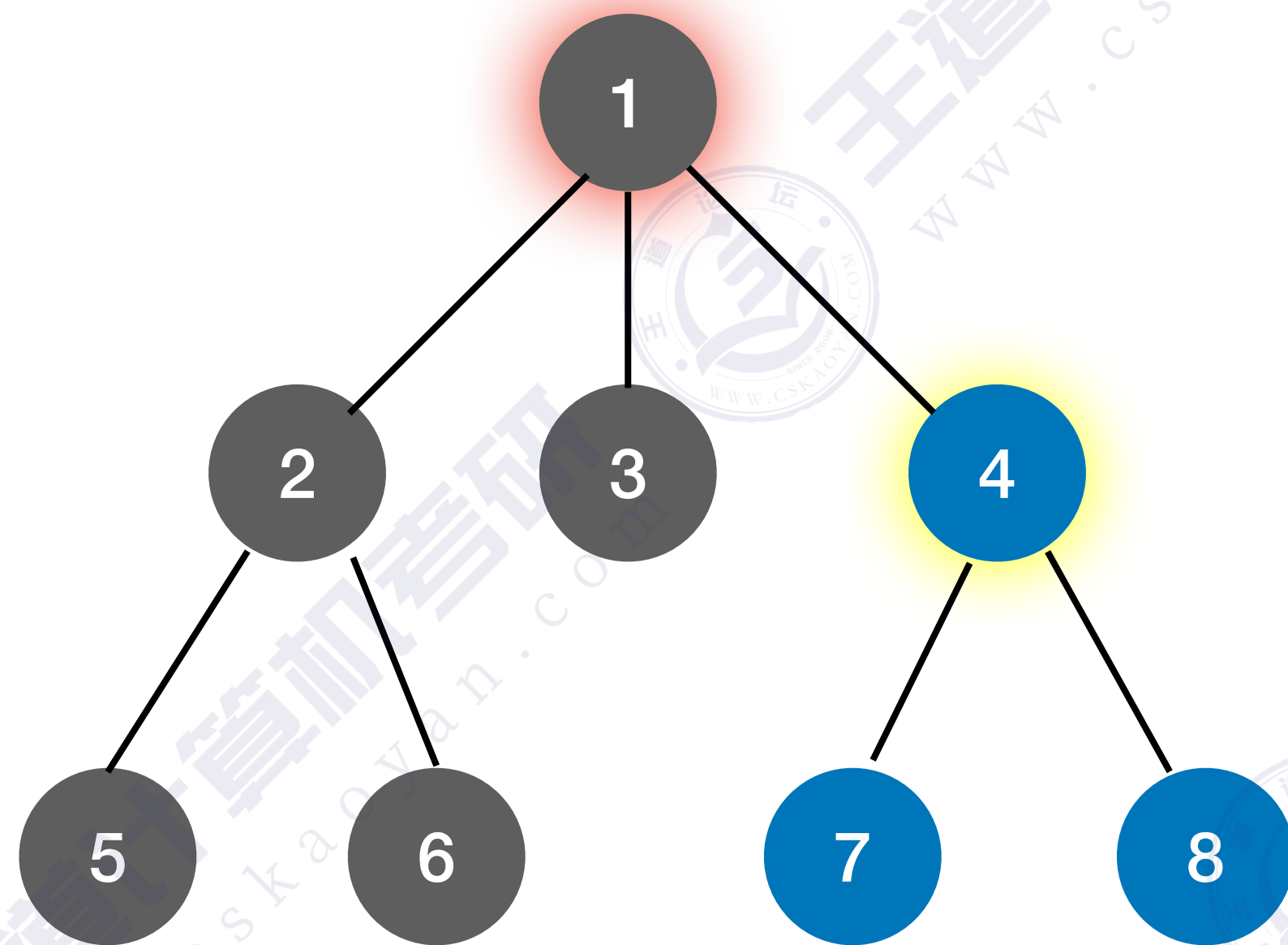
树的深度优先遍历



//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  
    }  
}
```

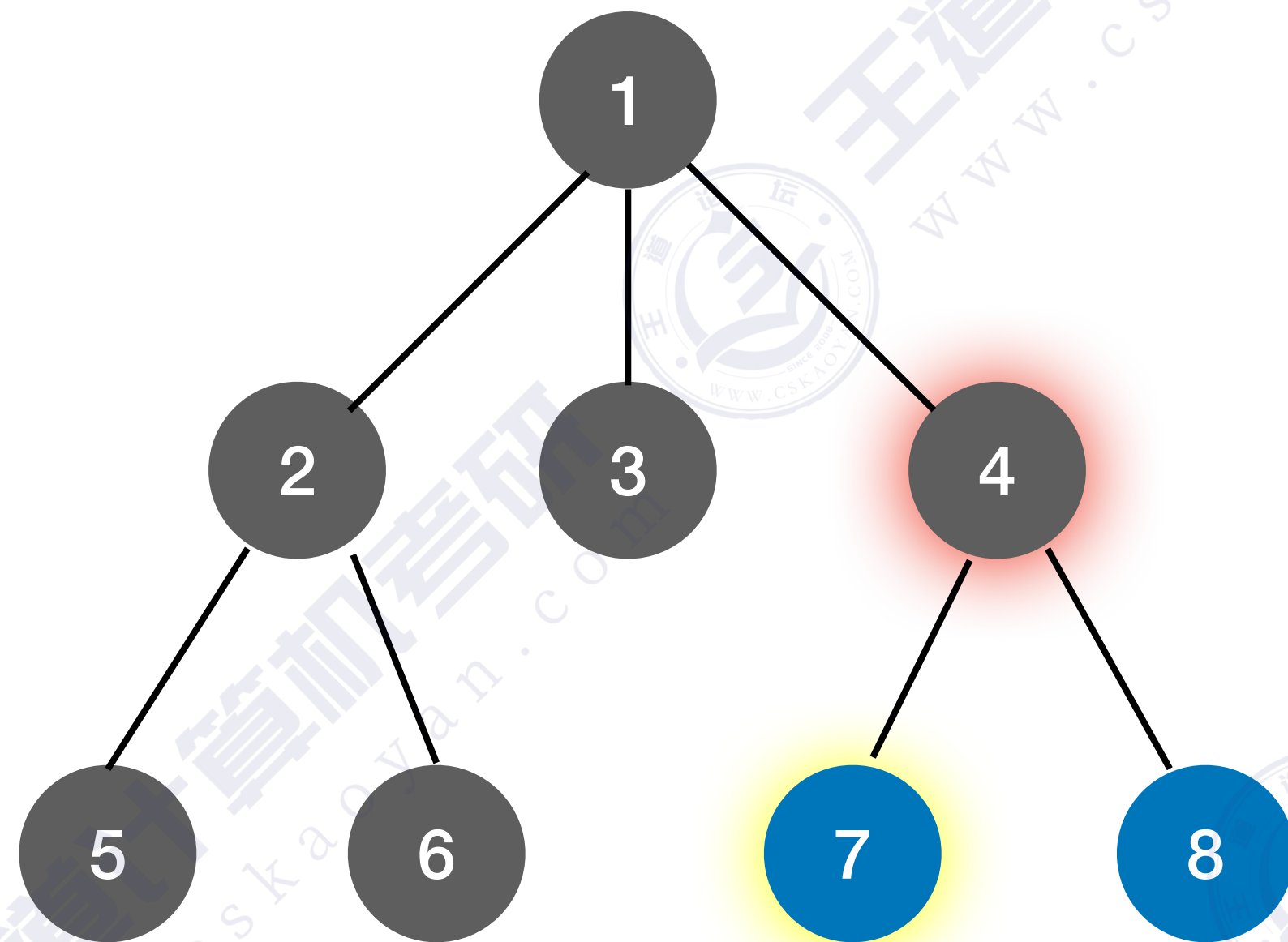

树的深度优先遍历



//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  
    }  
}
```

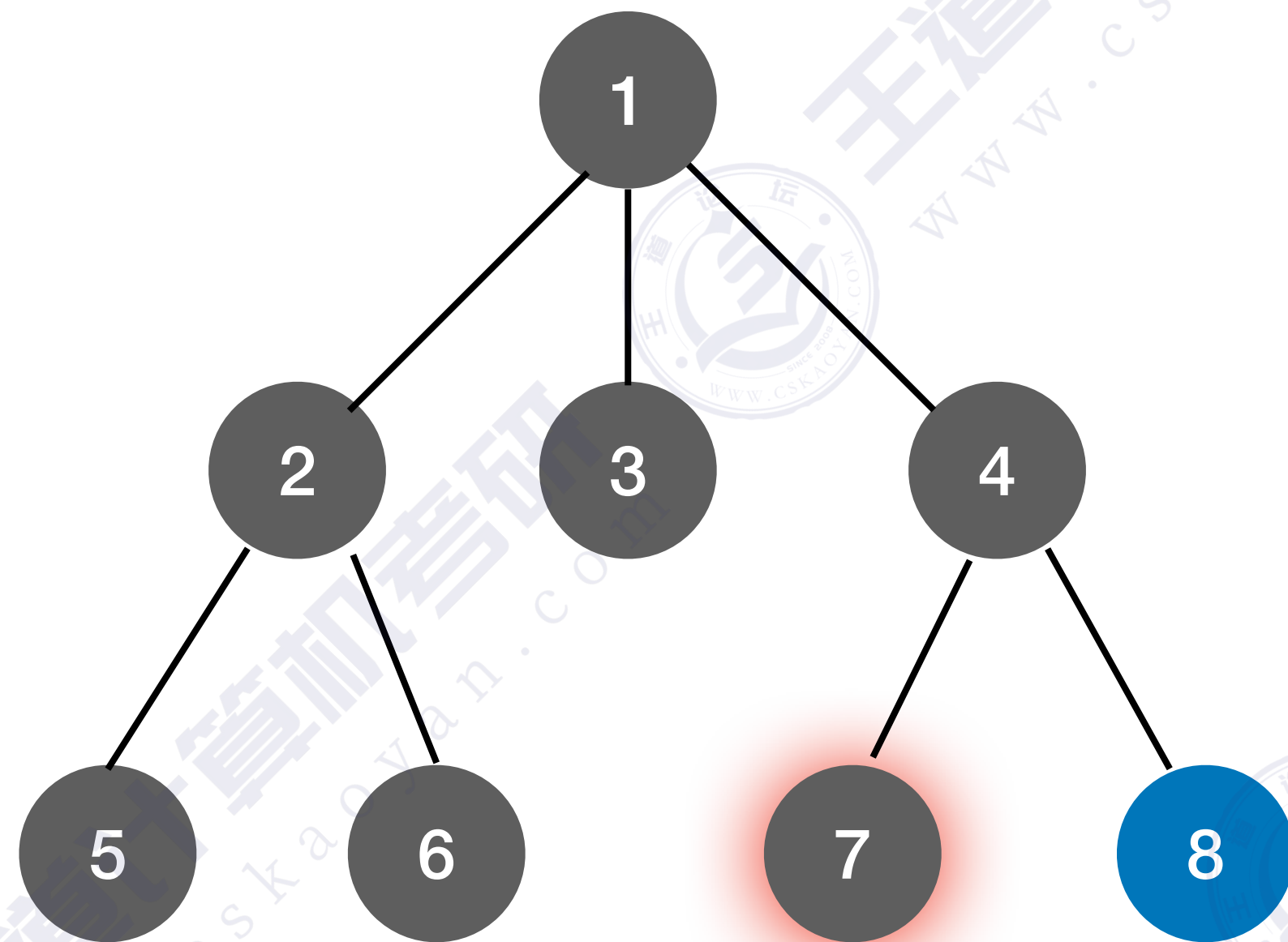
树的深度优先遍历



//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  
    }  
}
```

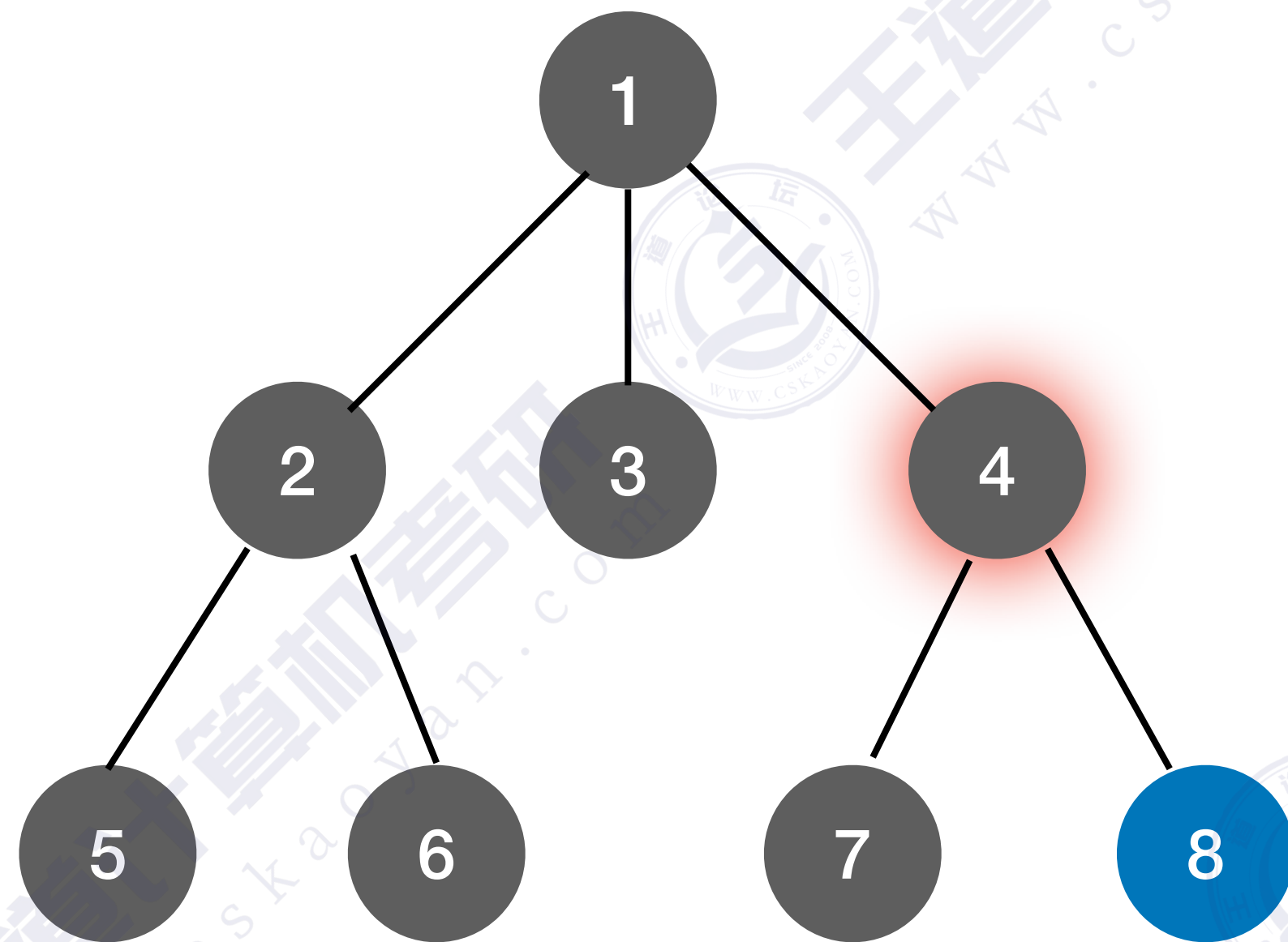
树的深度优先遍历



//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  
    }  
}
```

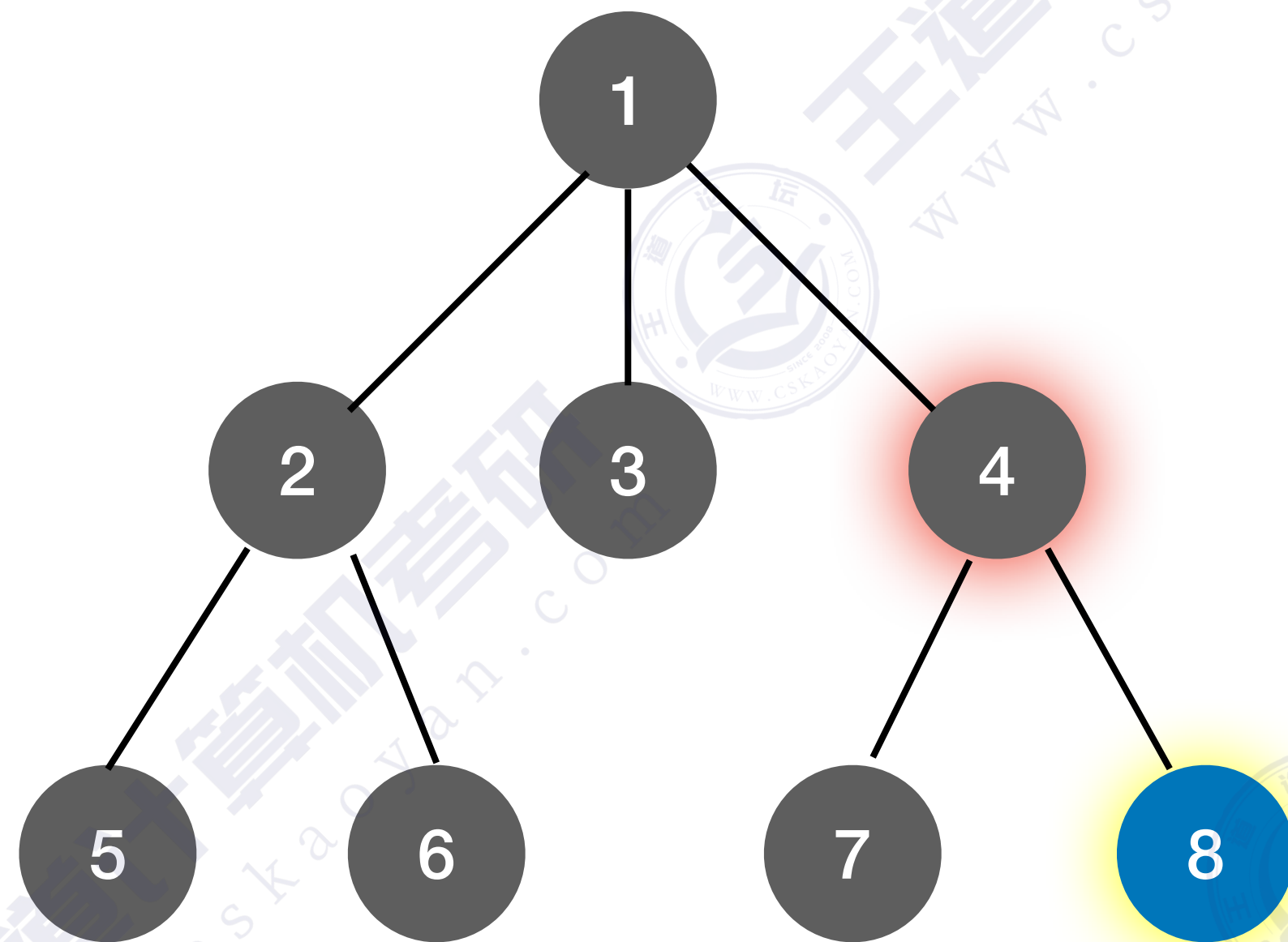

树的深度优先遍历



//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  
    }  
}
```

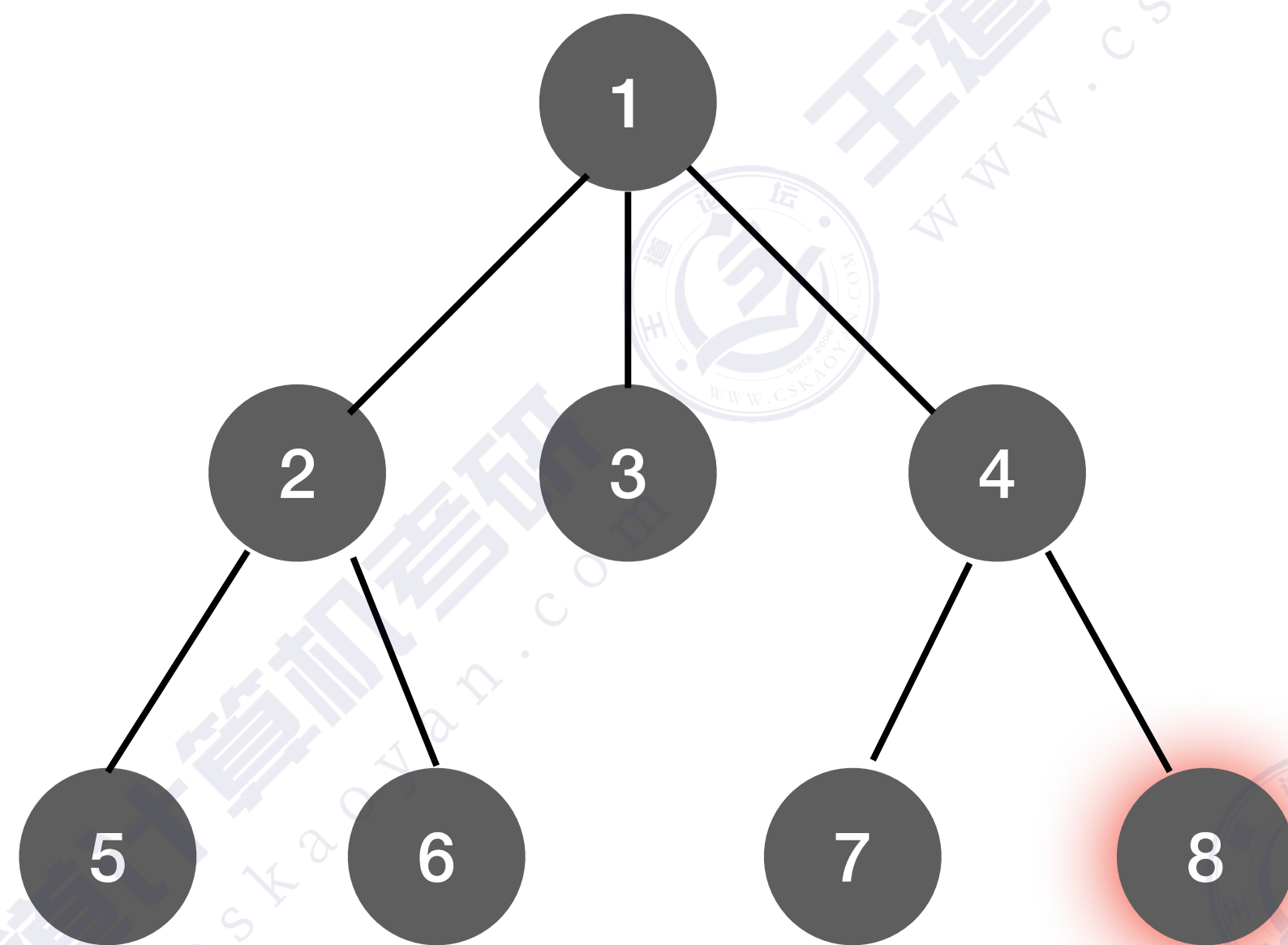
树的深度优先遍历



//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  
    }  
}
```

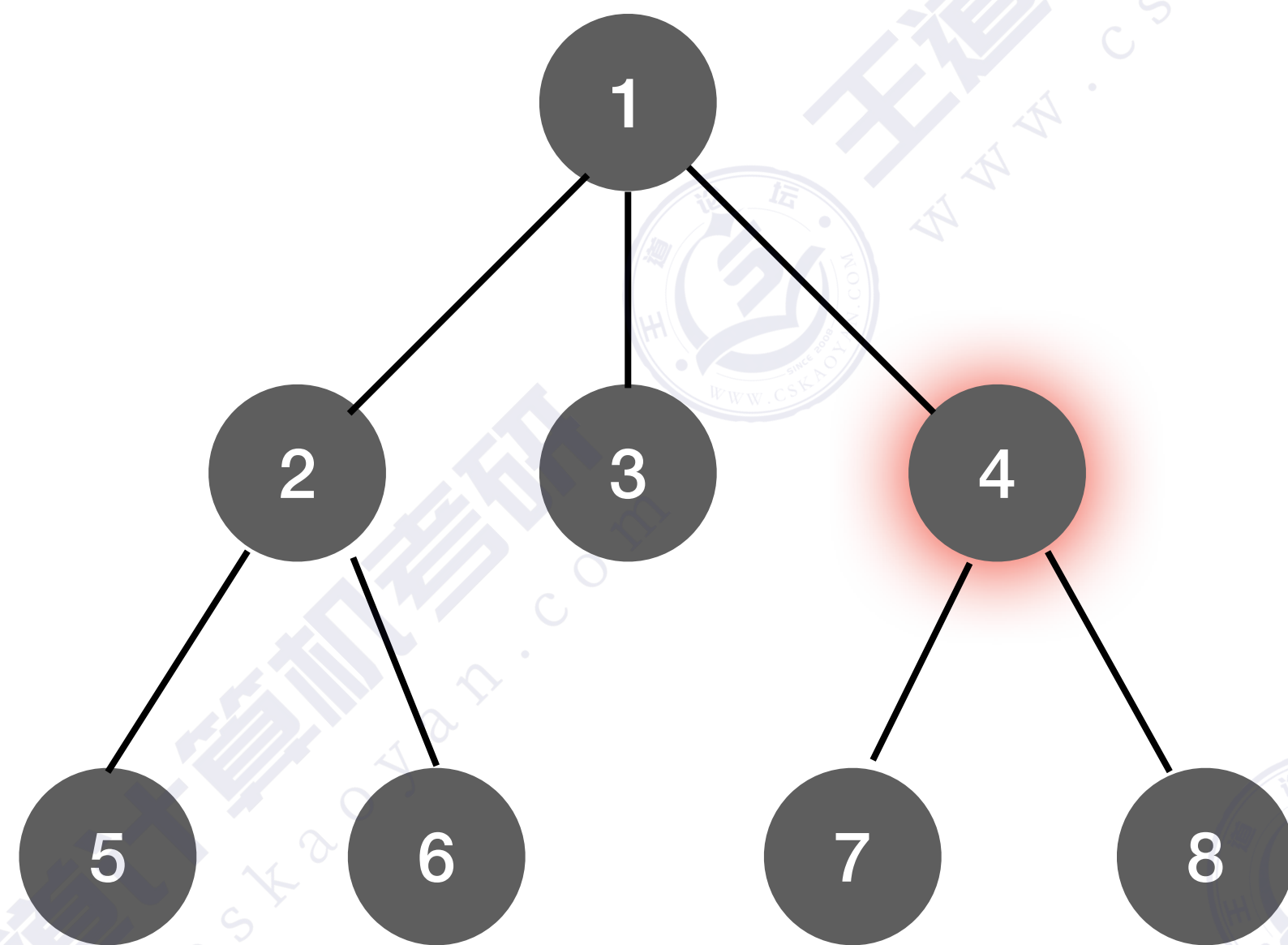
树的深度优先遍历



//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  
    }  
}
```

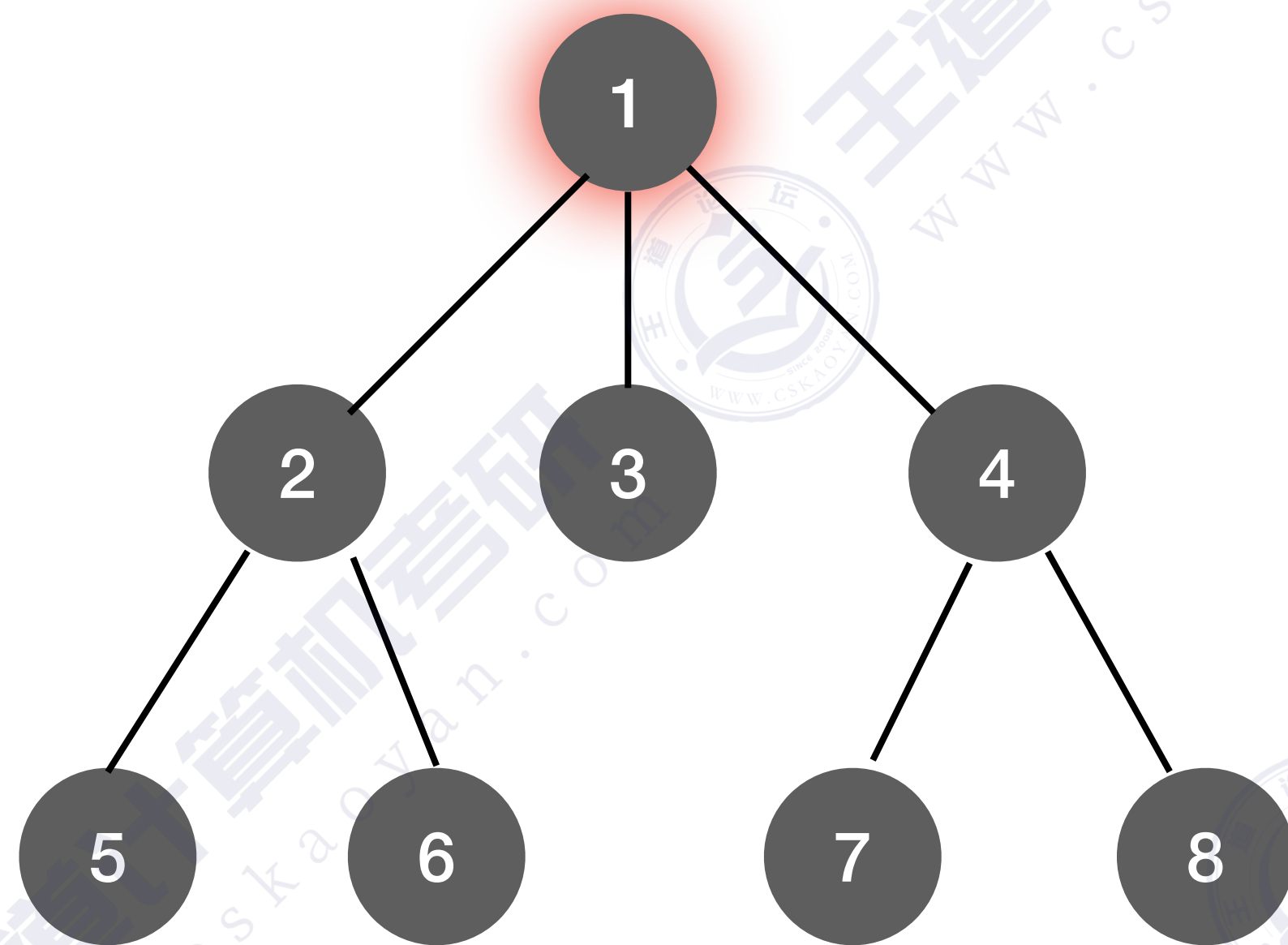

树的深度优先遍历



//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  
    }  
}
```

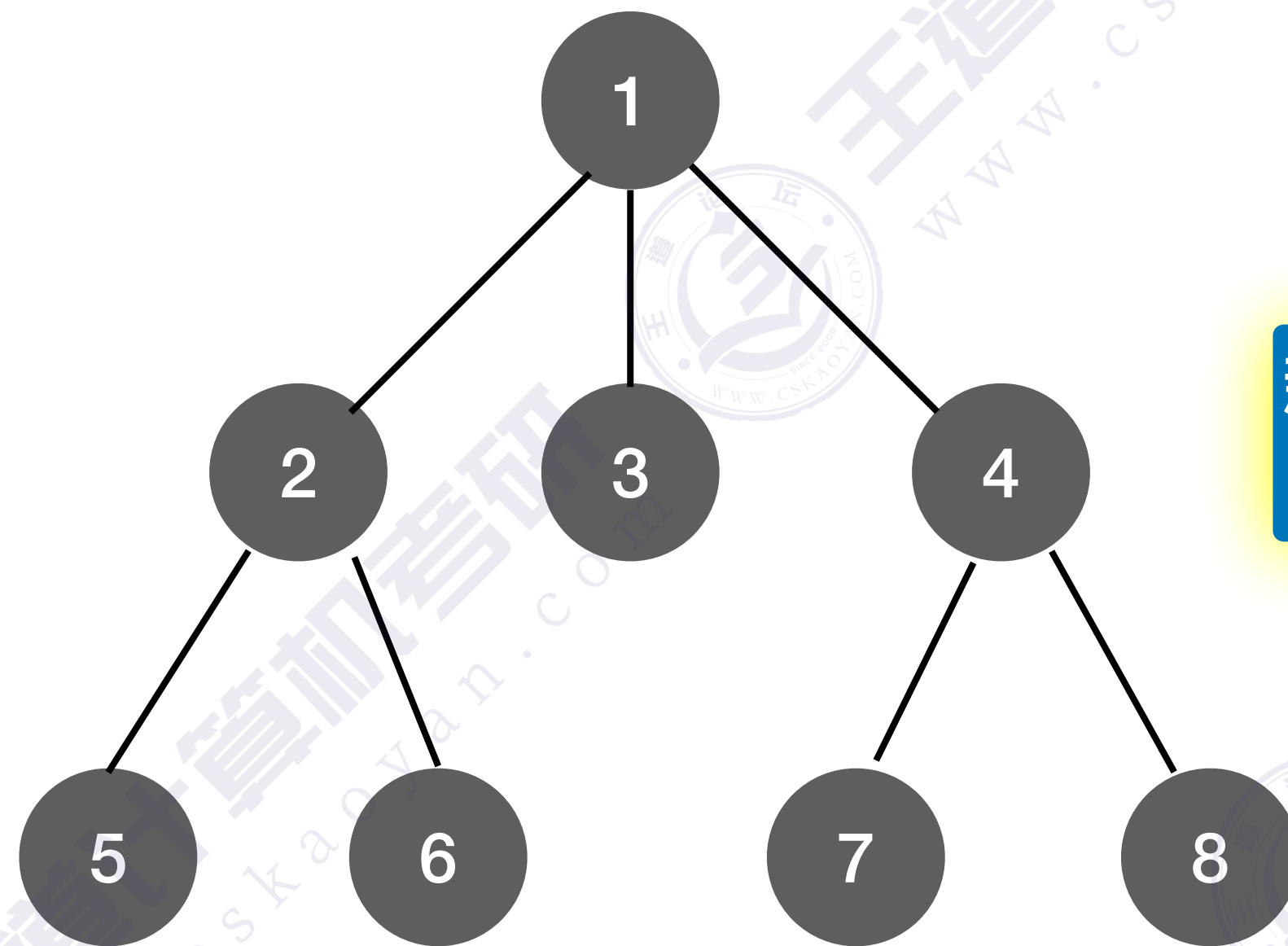
树的深度优先遍历



//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



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
```

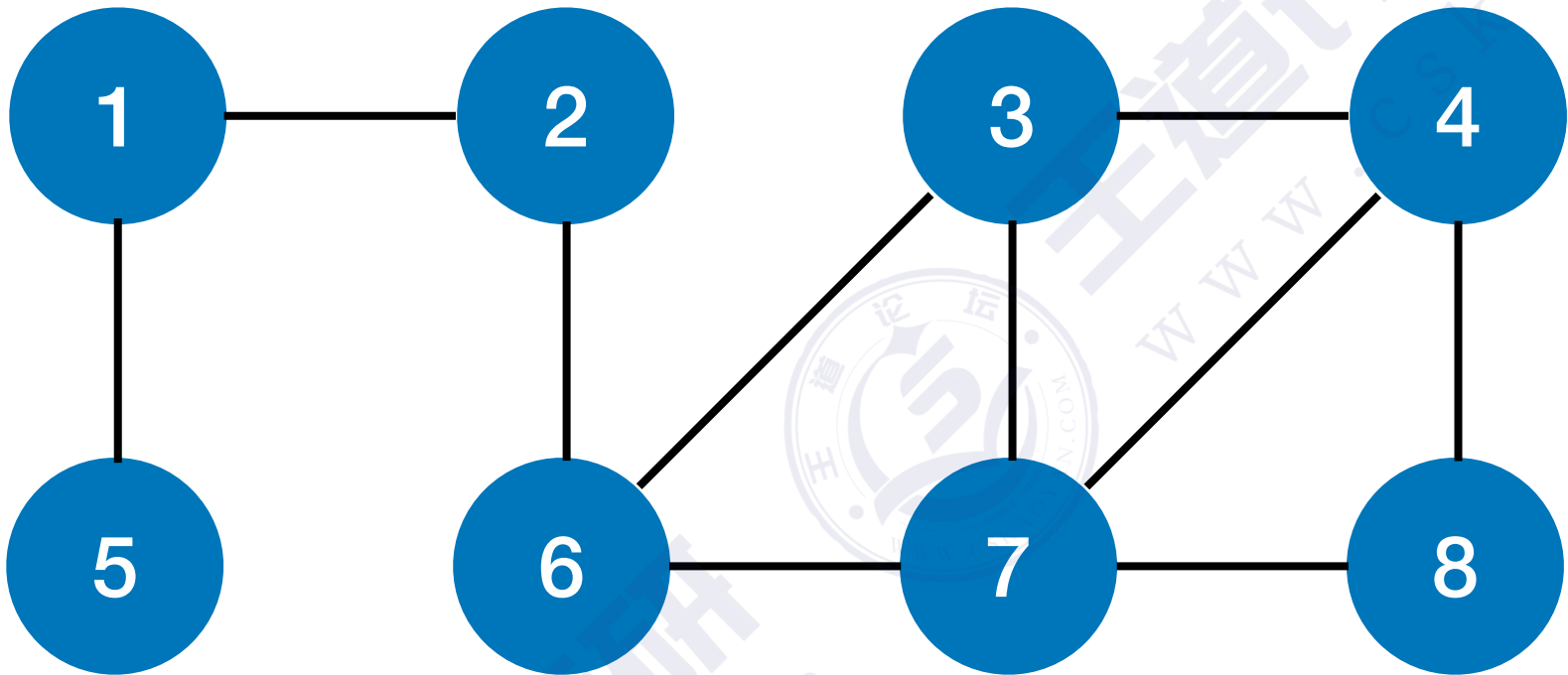
```
}
```

```
}
```

先根遍历序列：1，2，5，6，3，4，7，8

图的深度优先遍历

初始都为false



函数调用栈

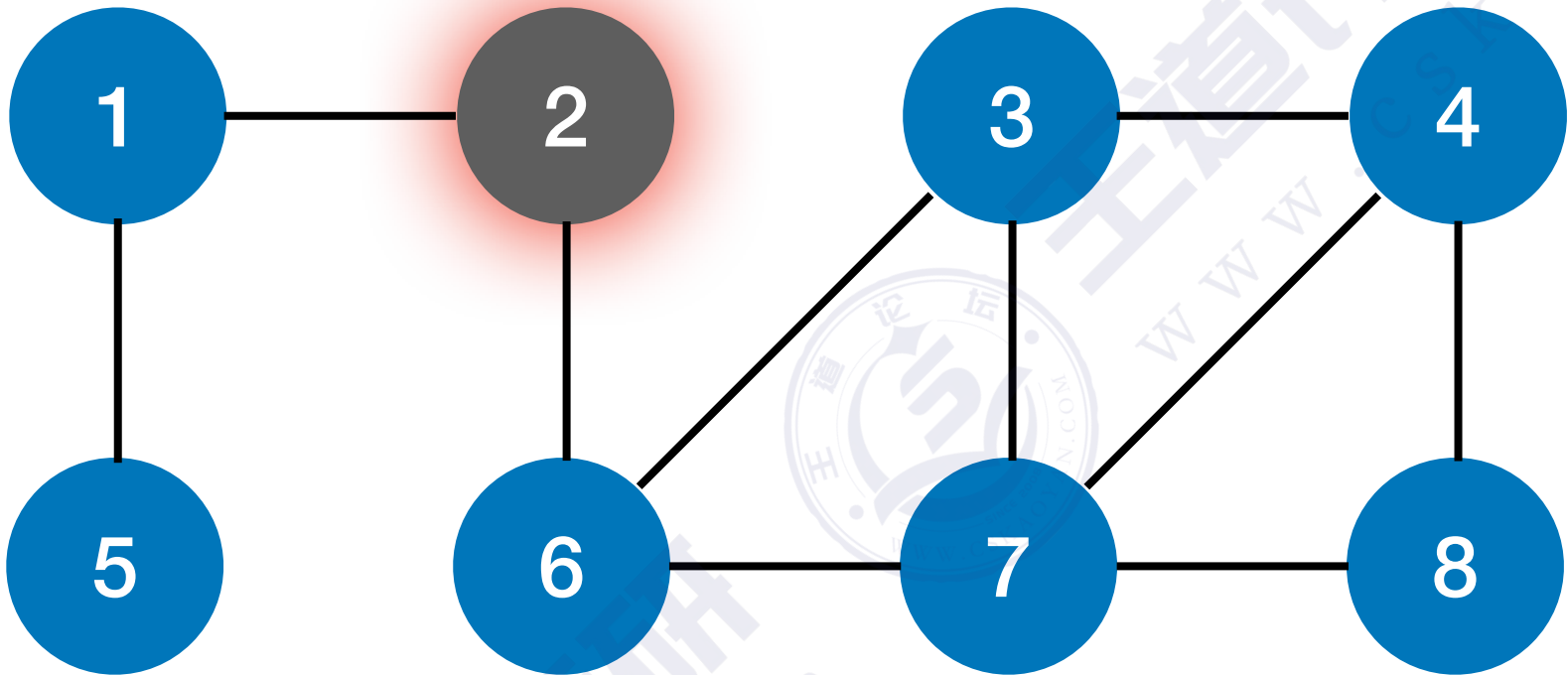
```
bool visited[MAX_VERTEX_NUM]; //访问标记数组

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

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

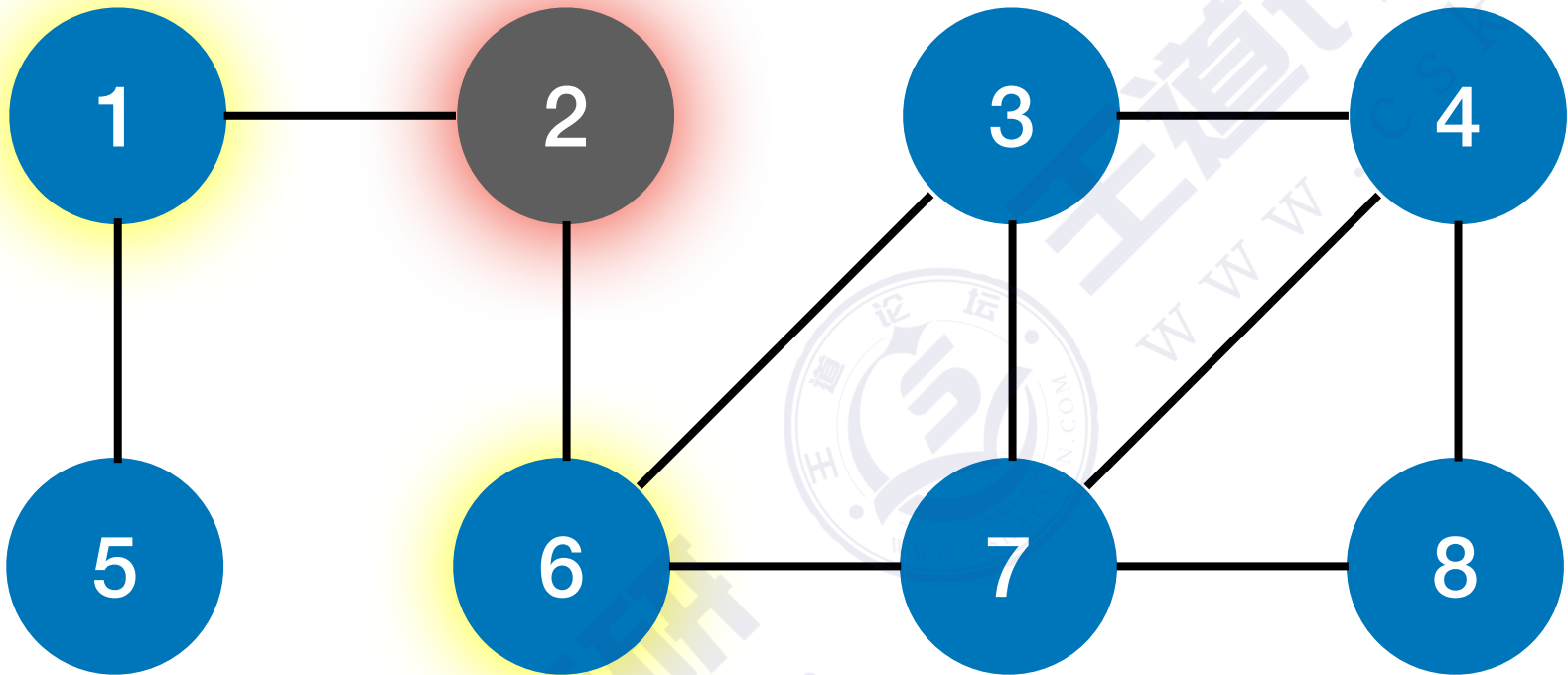
	1	2	3	4	5	6	7	8
visited	false	true	false	false	false	false	false	false



函数调用栈

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
    }
```

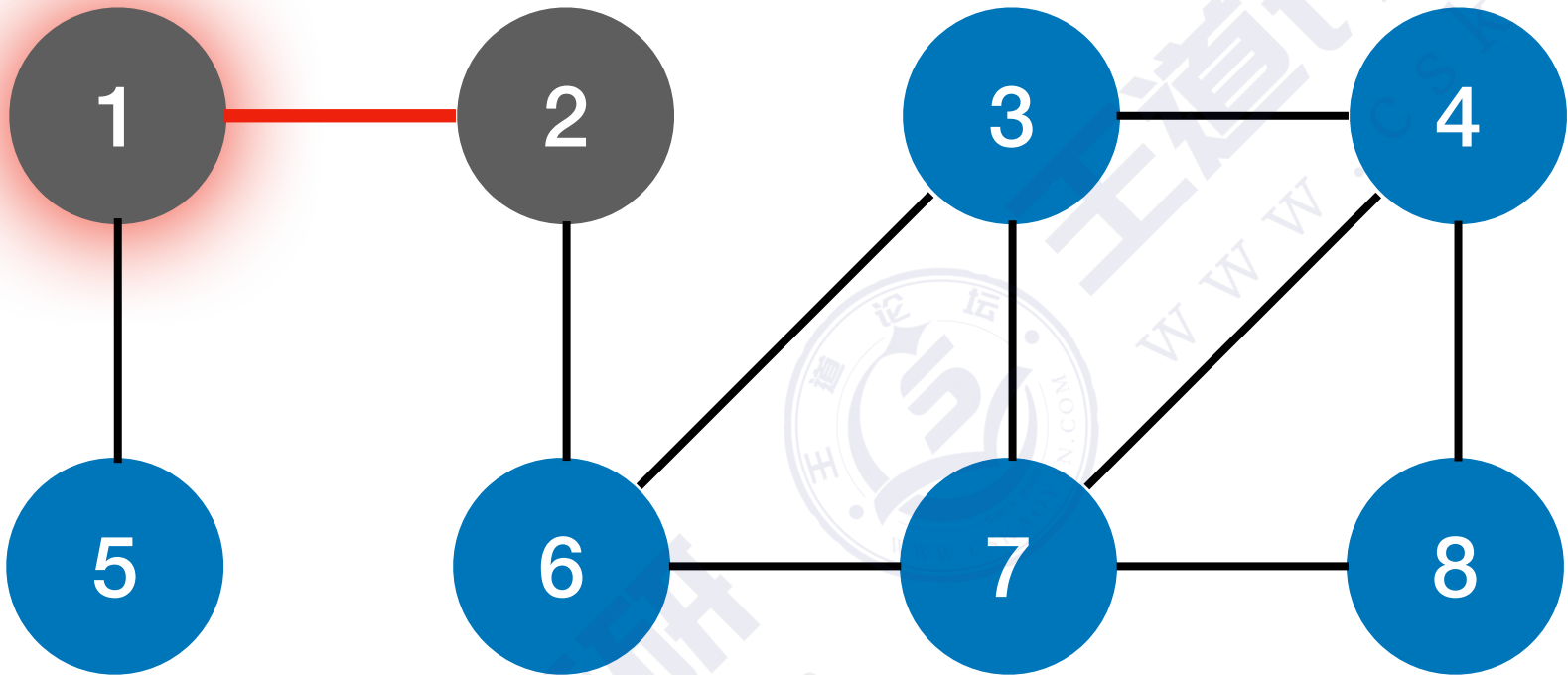
	1	2	3	4	5	6	7	8
visited	false	true	false	false	false	false	false	false



函数调用栈

图的深度优先遍历

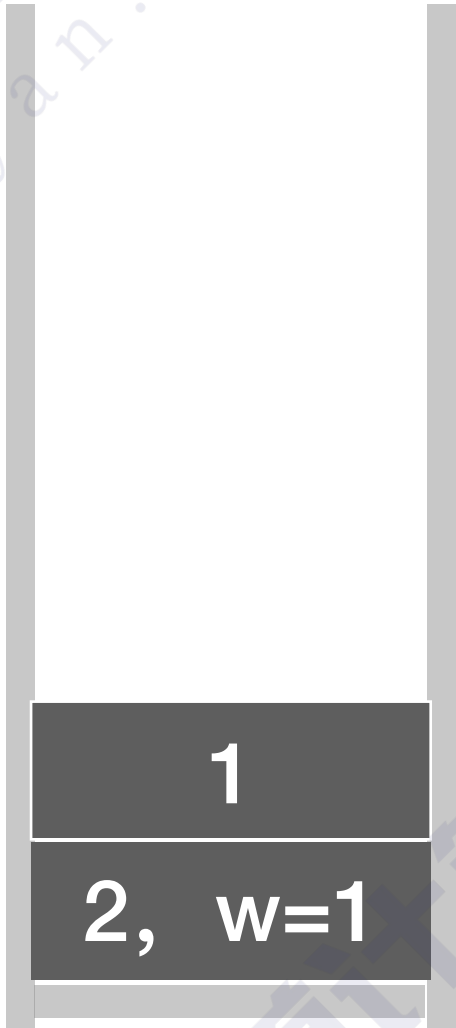
初始都为false



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

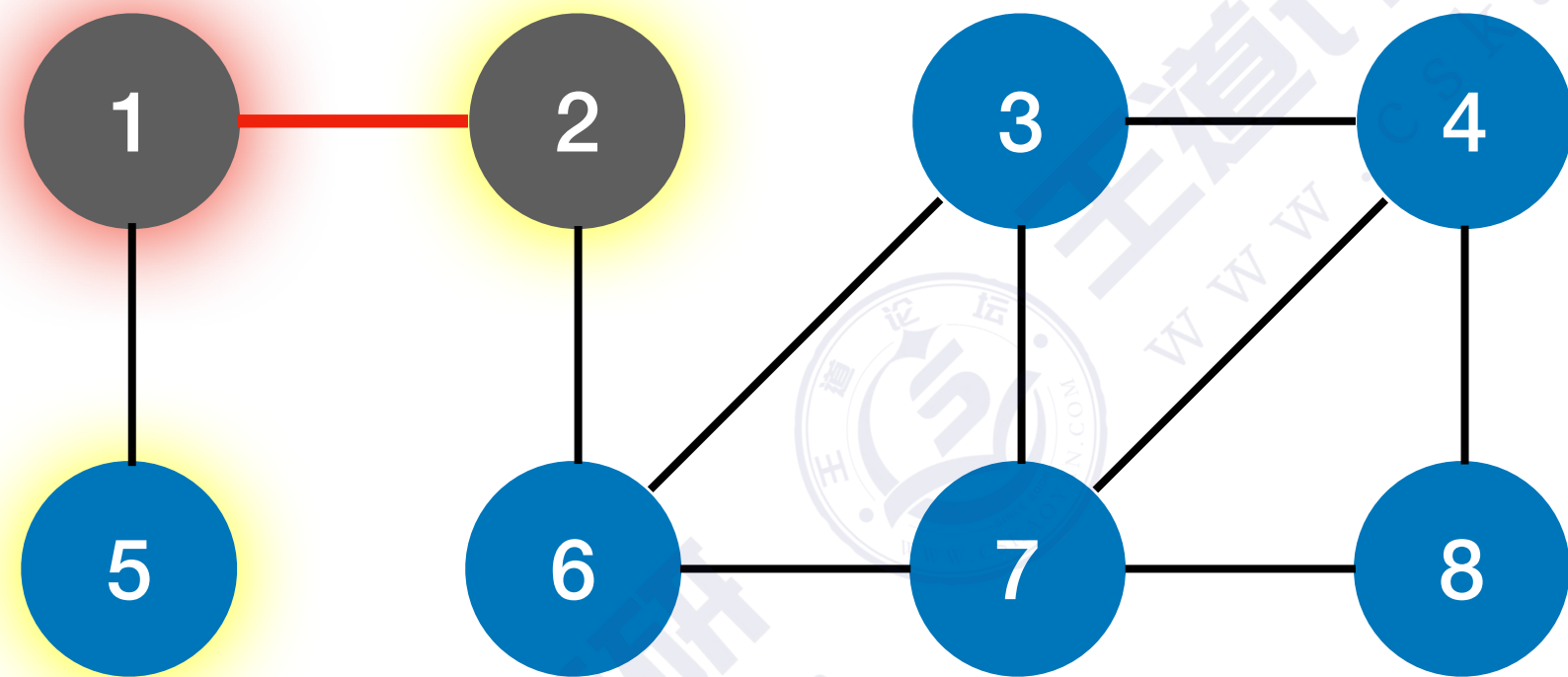
	1	2	3	4	5	6	7	8
visited	true	true	false	false	false	false	false	false



函数调用栈

图的深度优先遍历

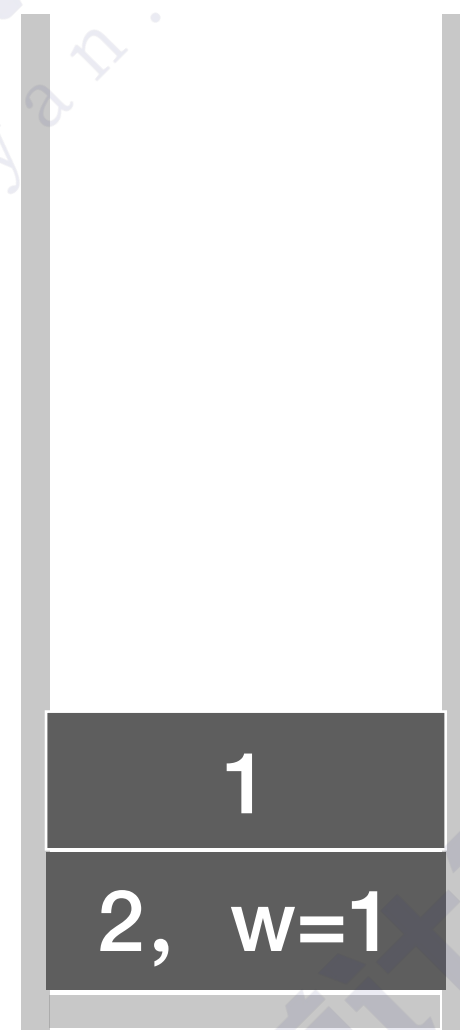
初始都为false



```
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
    }                             }
```

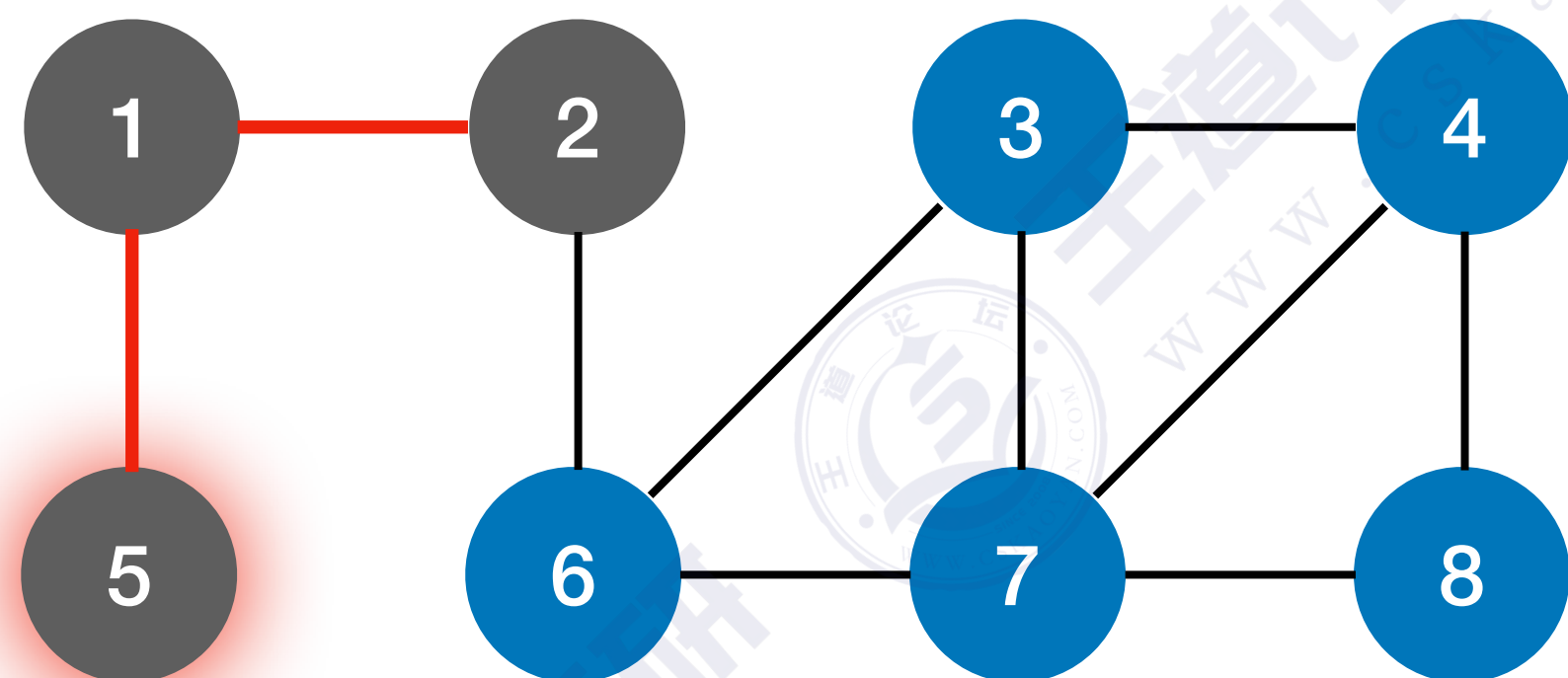
	1	2	3	4	5	6	7	8
visited	true	true	false	false	false	false	false	false



函数调用栈

图的深度优先遍历

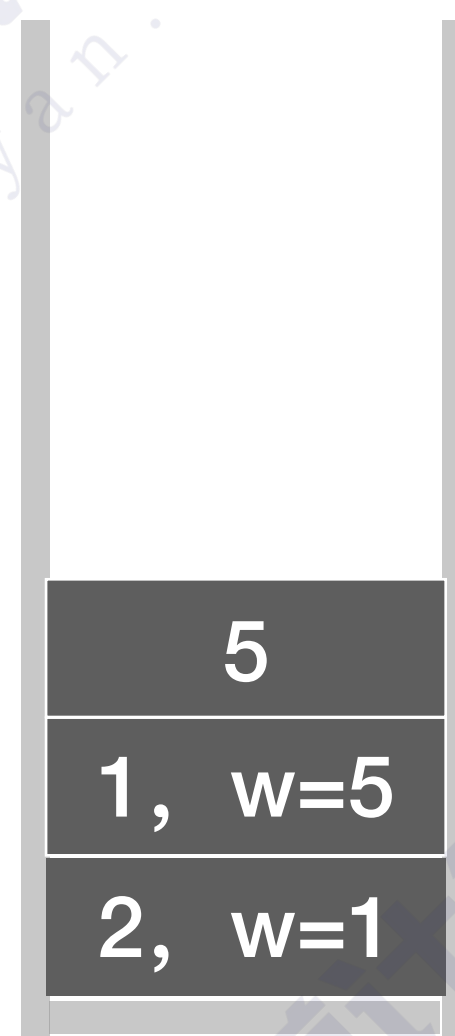
初始都为false



```
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
    }                             }
```

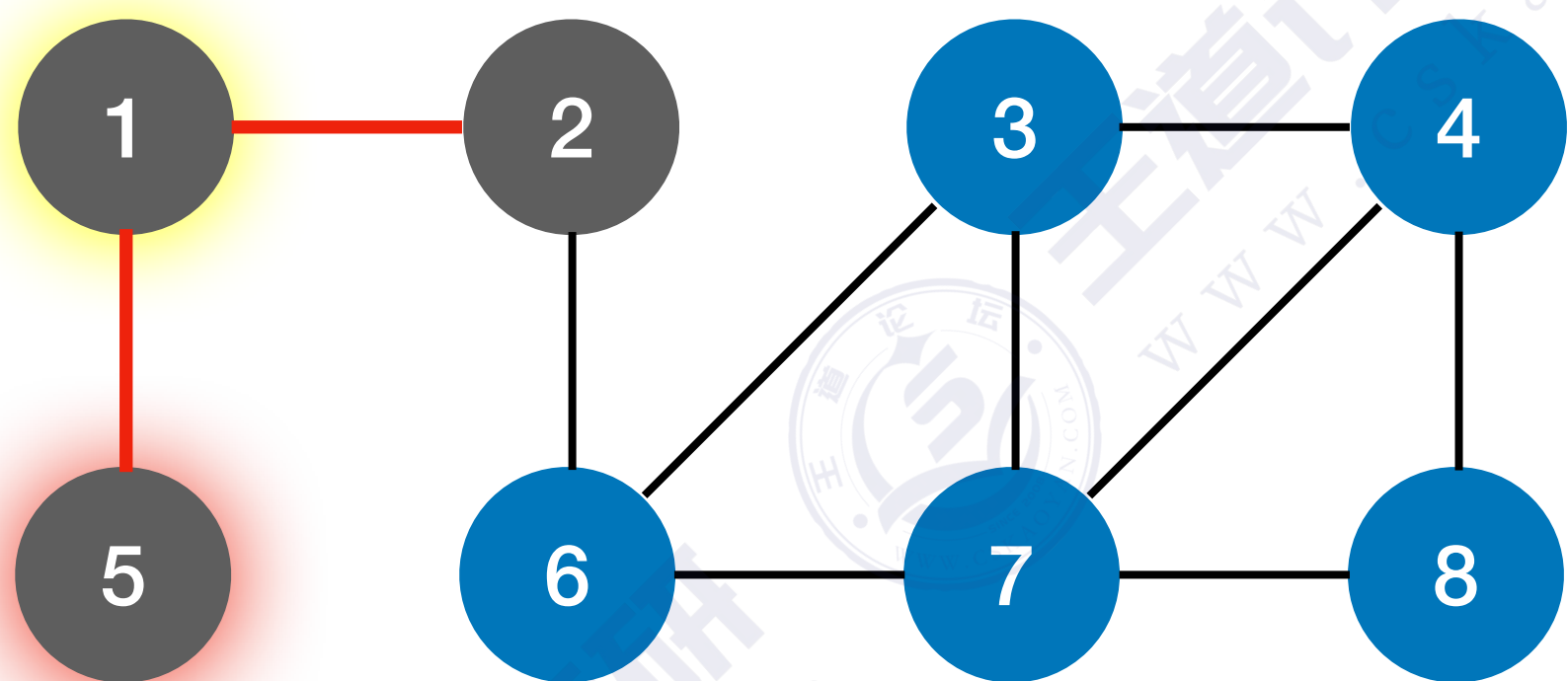
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	false	false	false



函数调用栈

图的深度优先遍历

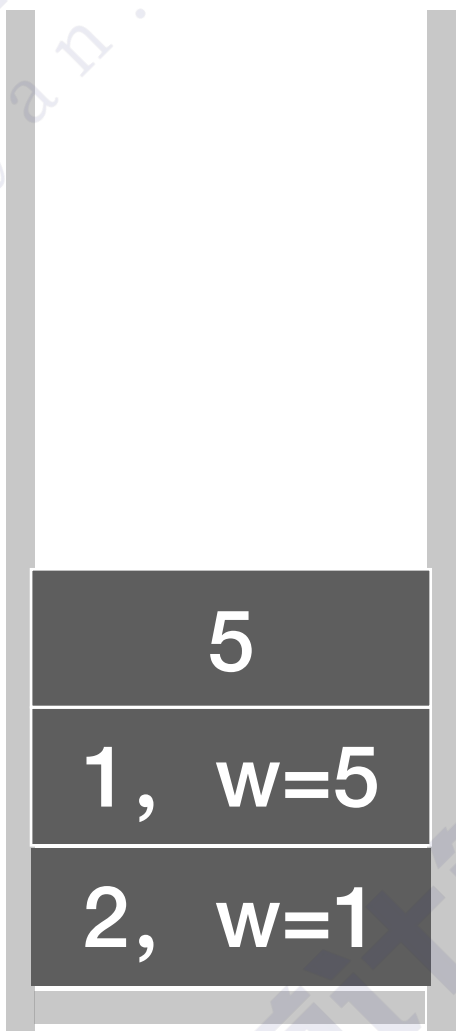
初始都为false



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

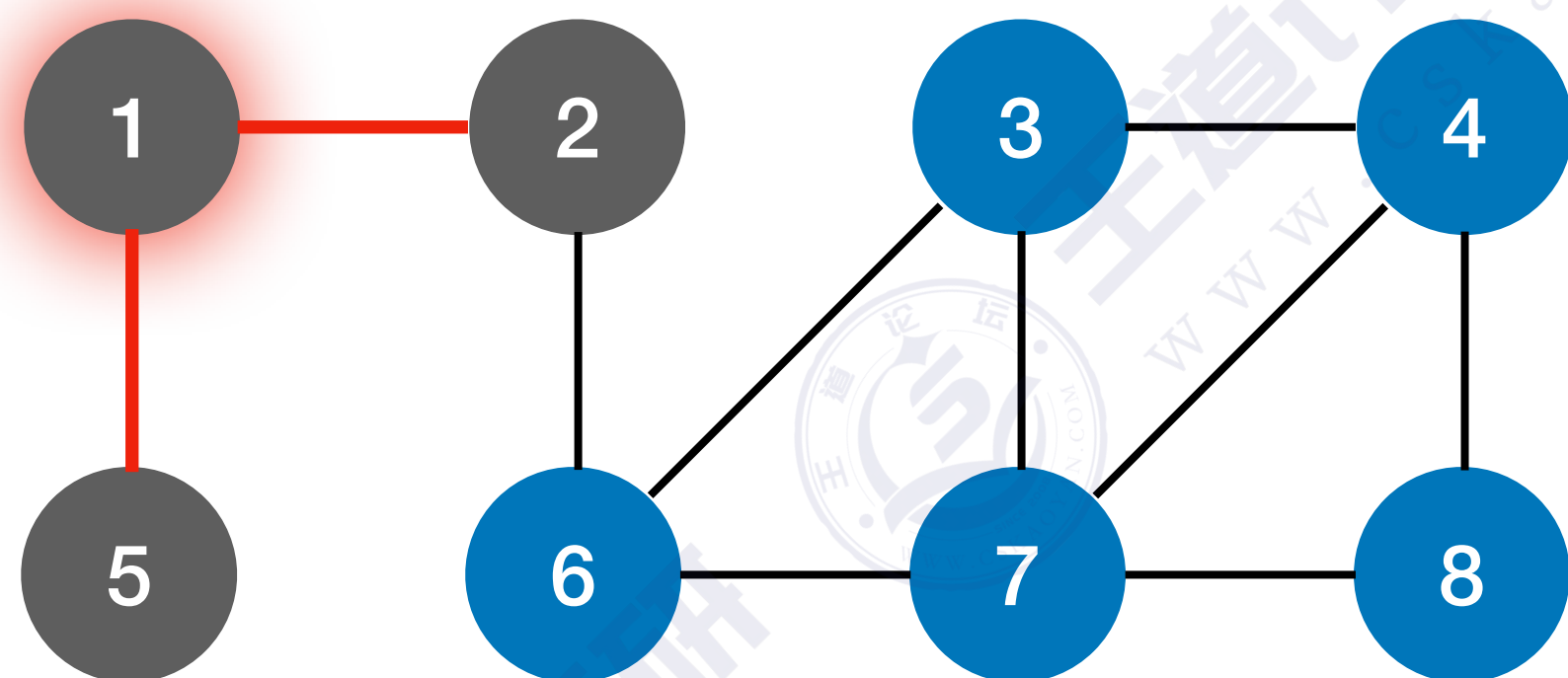
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	false	false	false



函数调用栈

图的深度优先遍历

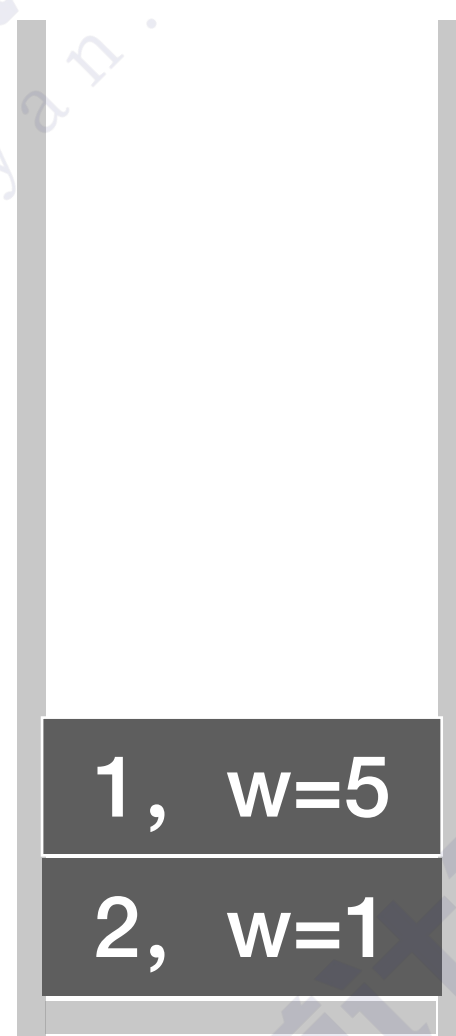
初始都为false



```
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
    }                             }
```

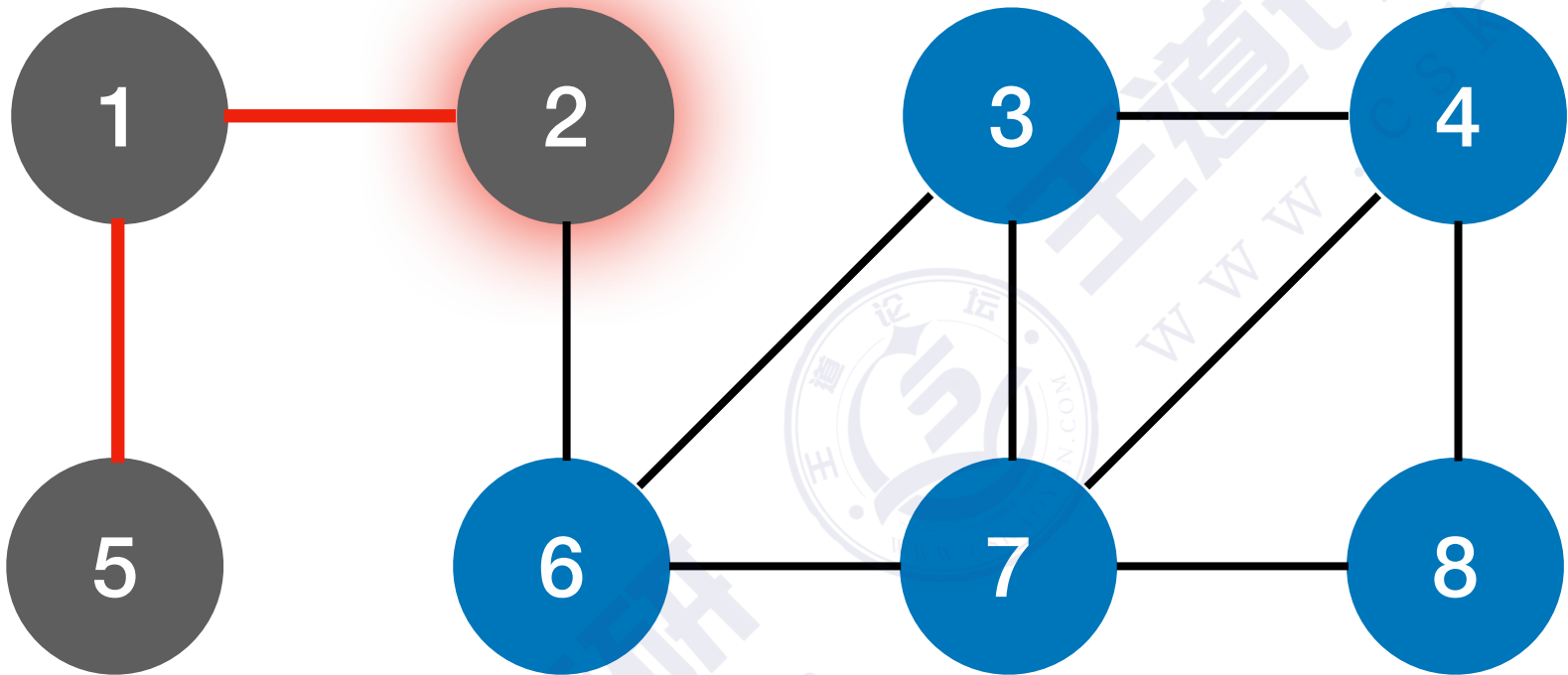
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	false	false	false



函数调用栈

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

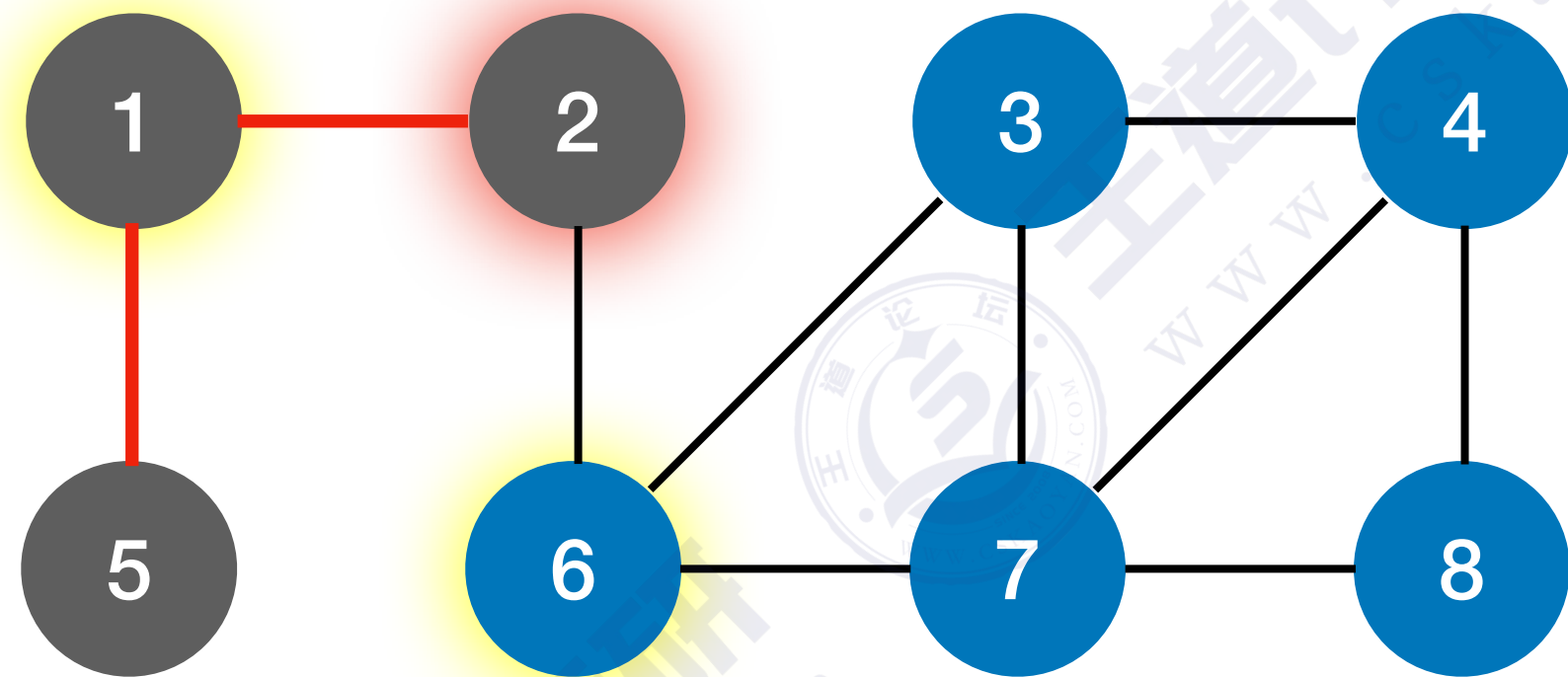
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	false	false	false



函数调用栈

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
    }                             }
```

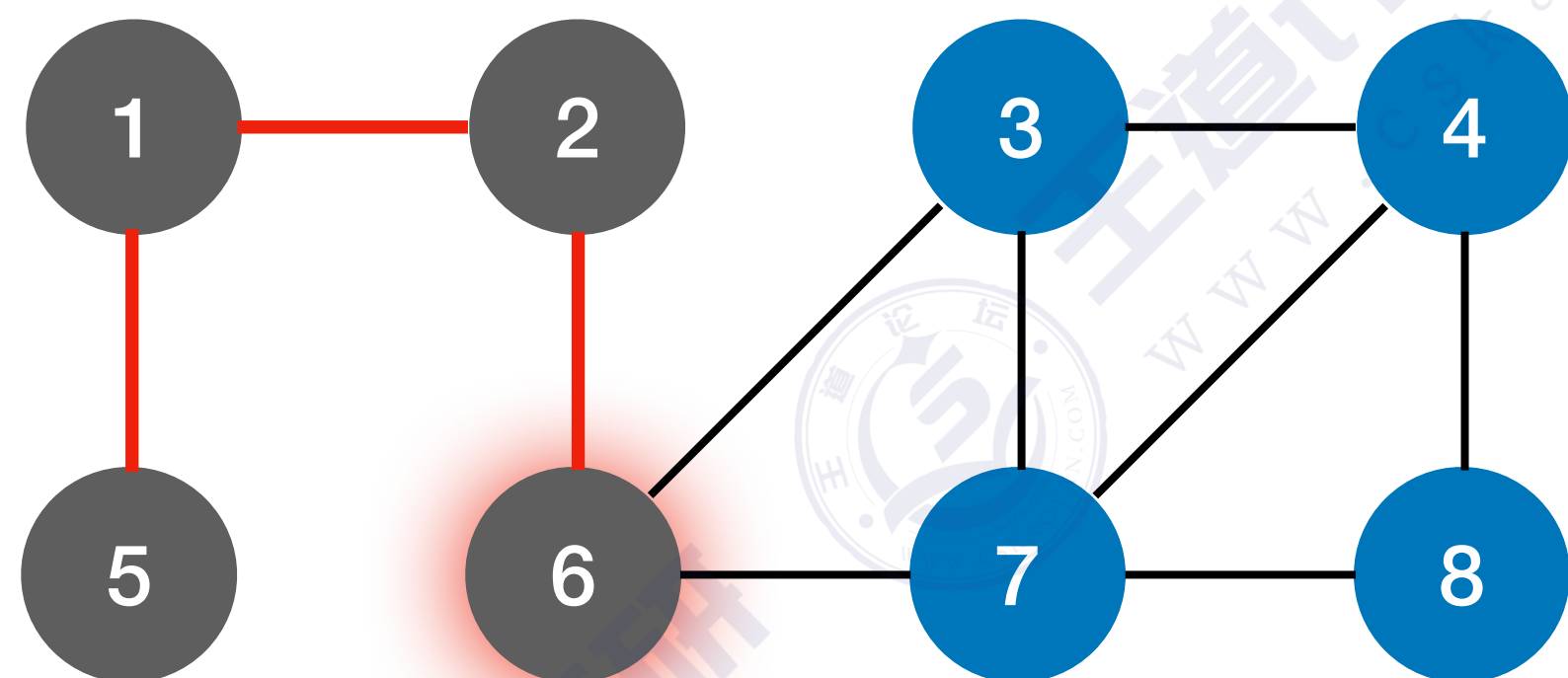
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	false	false	false

2, w=1

函数调用栈

图的深度优先遍历

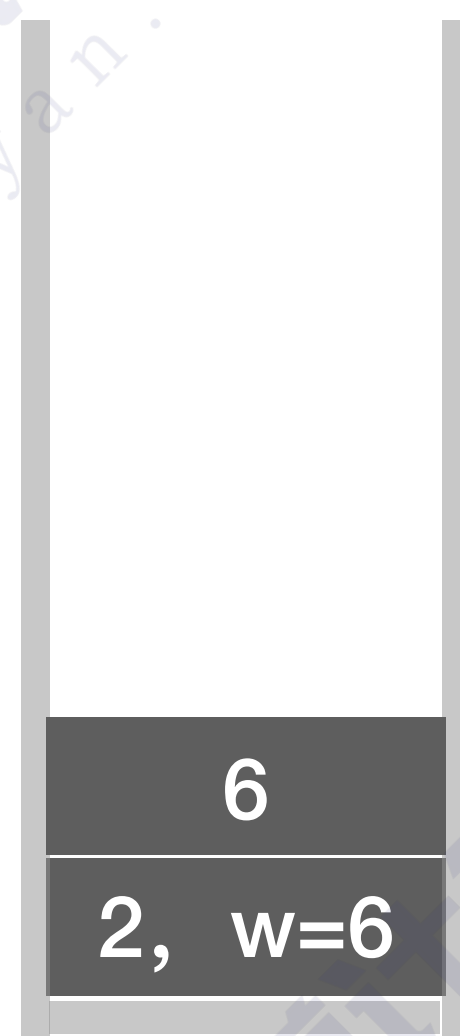
初始都为false



```
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
    }                             }
```

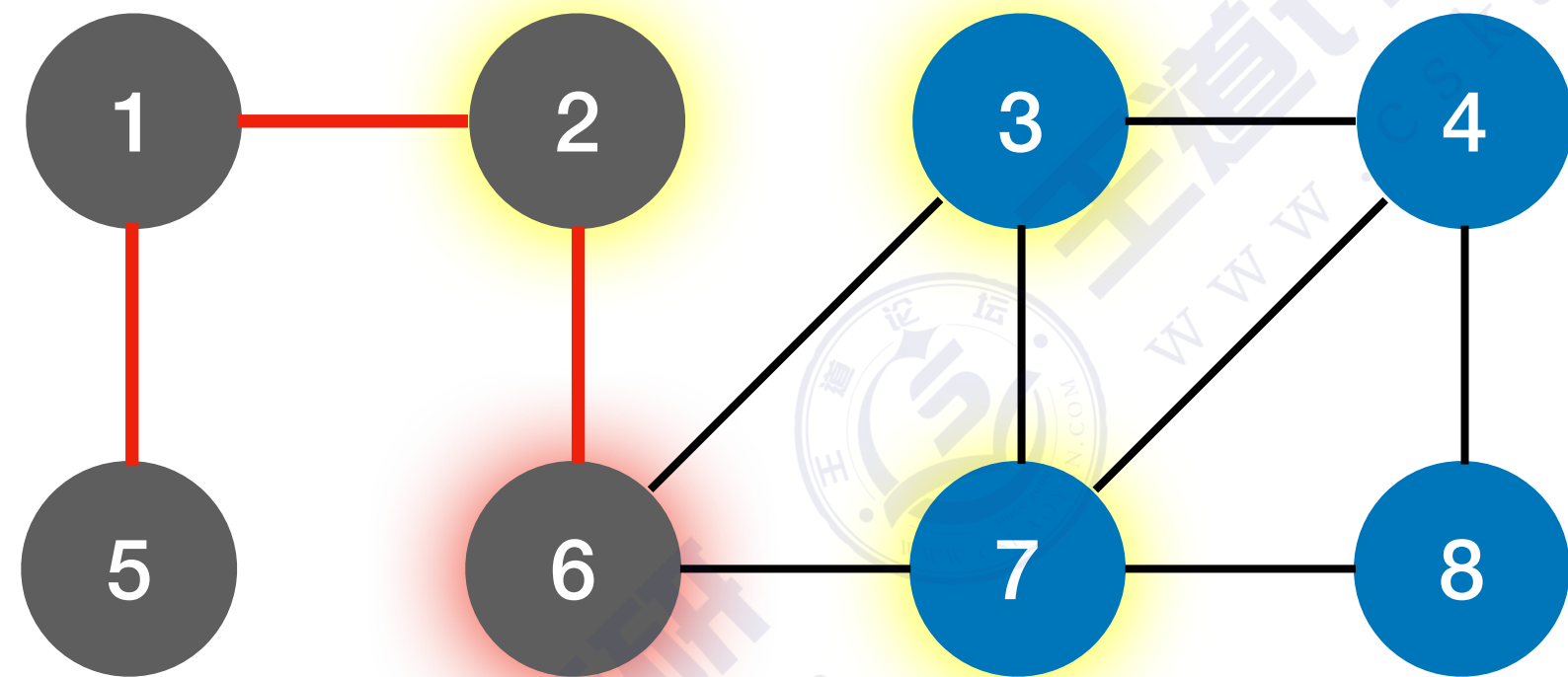
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	true	false	false



函数调用栈

图的深度优先遍历

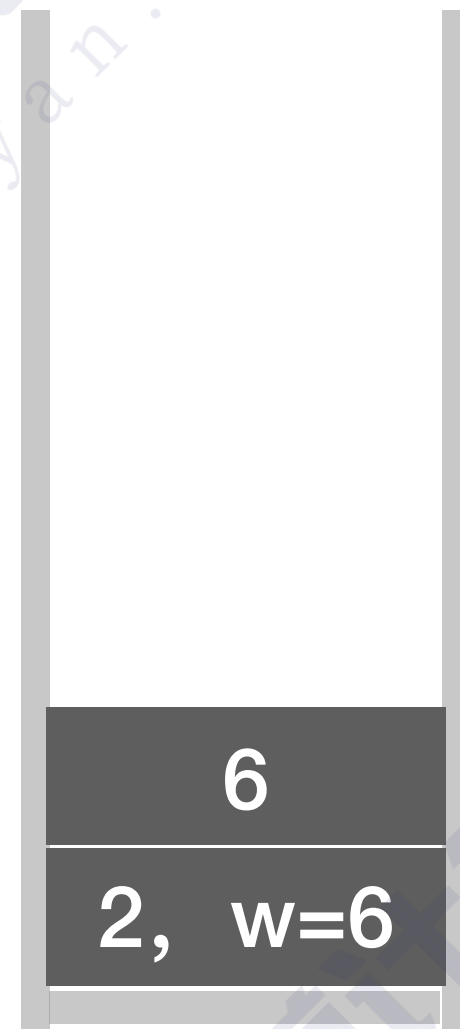
初始都为false



```
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
    }                             }
```

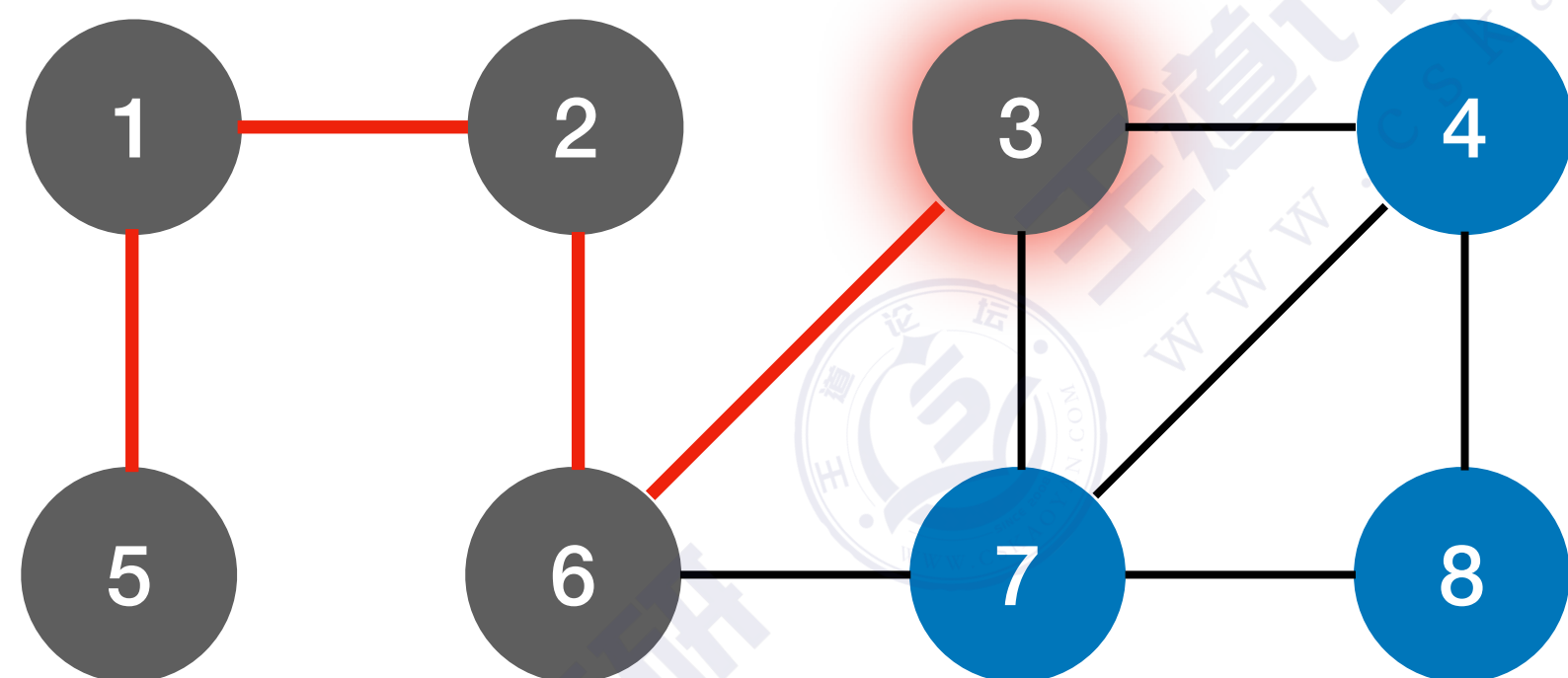
	1	2	3	4	5	6	7	8
visited	true	true	false	false	true	true	false	false



函数调用栈

图的深度优先遍历

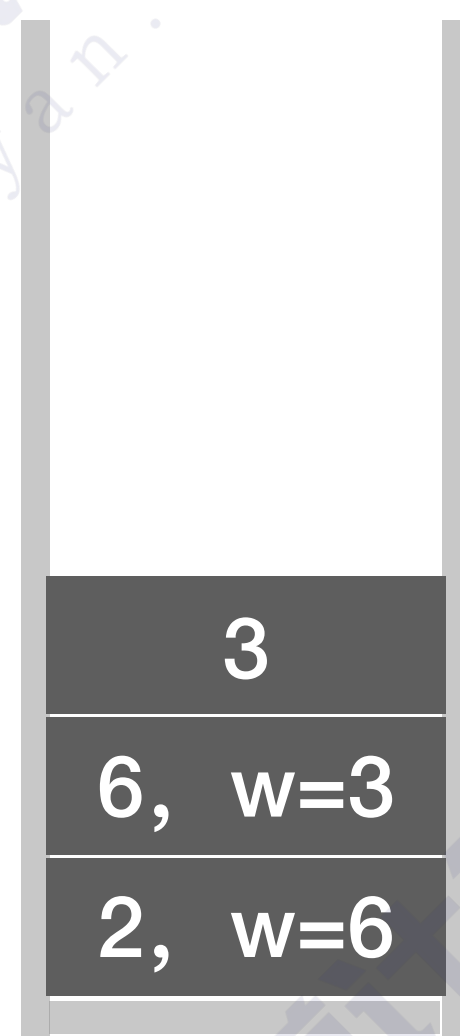
初始都为false



```
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
    }
}
```

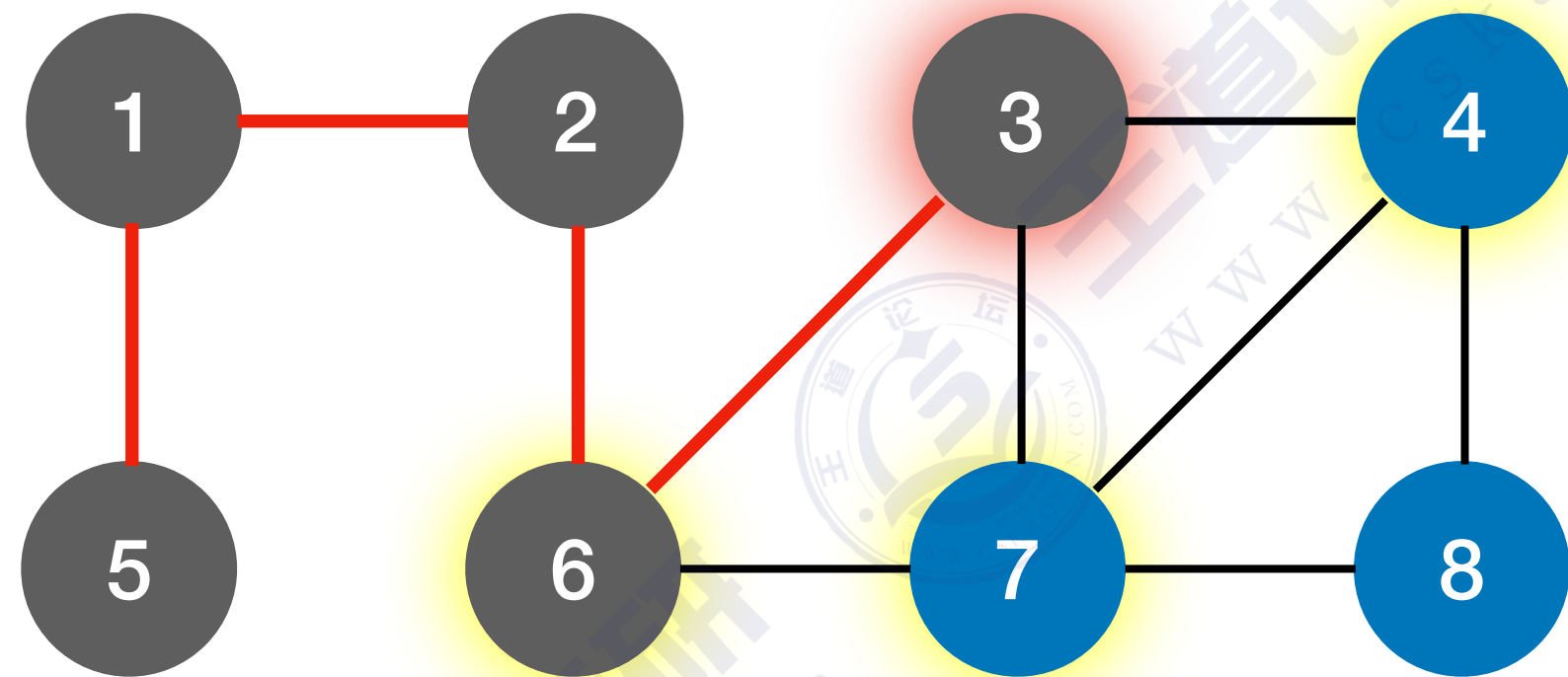
	1	2	3	4	5	6	7	8
visited	true	true	true	false	true	true	false	false



函数调用栈

图的深度优先遍历

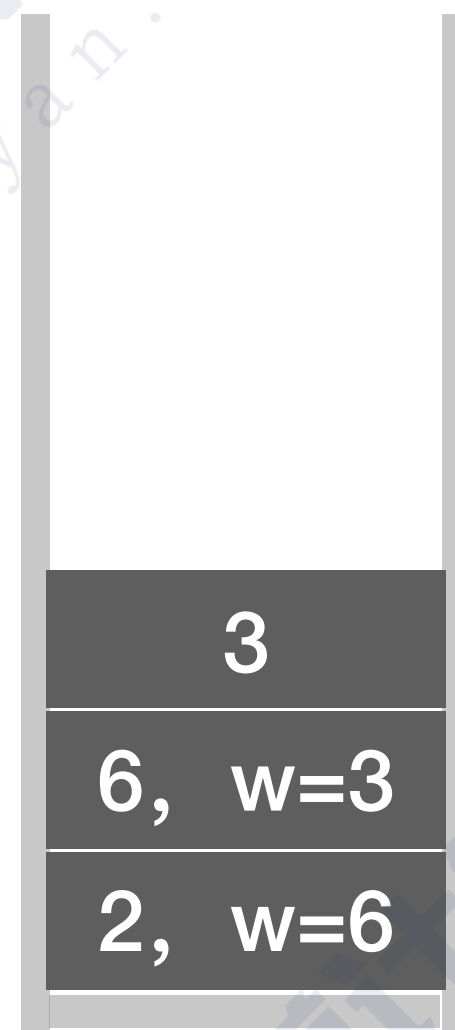
初始都为false



```
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
    }
}
```

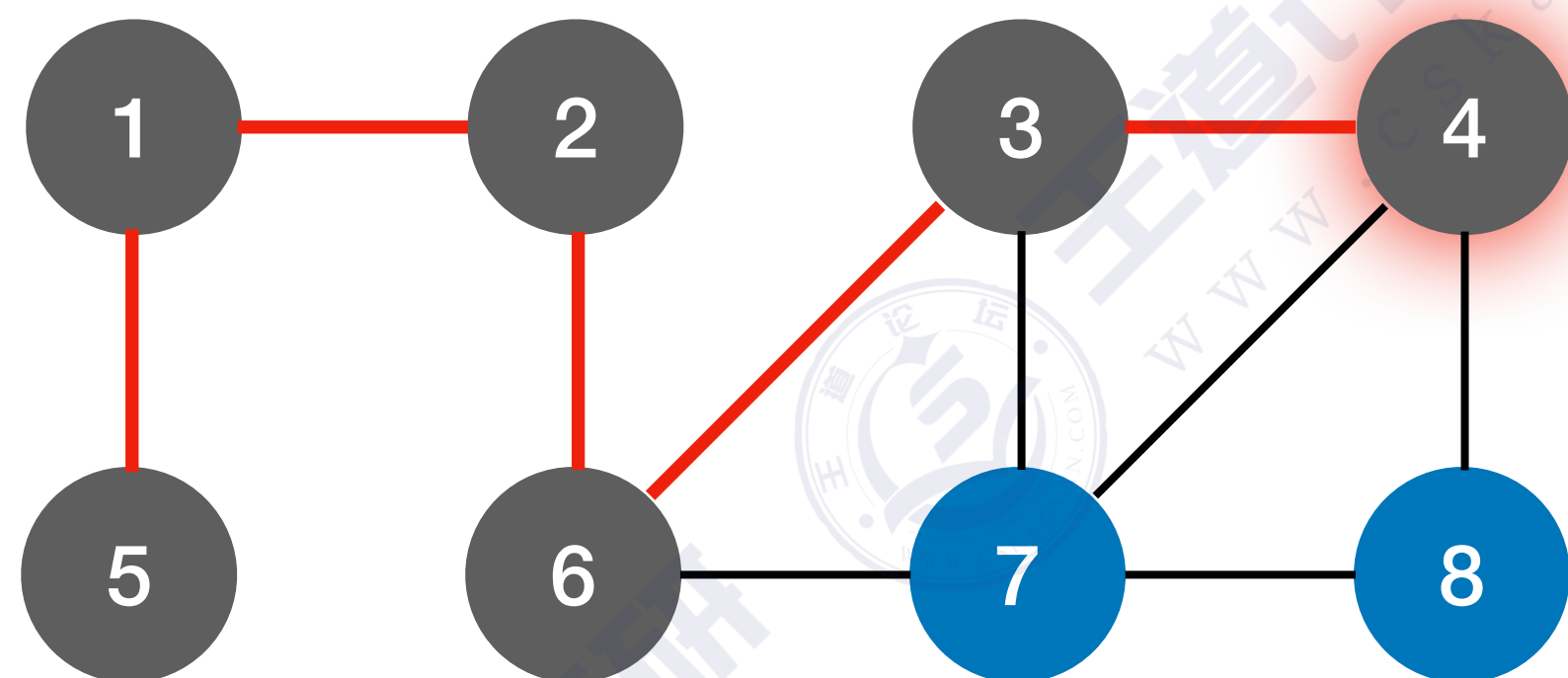
	1	2	3	4	5	6	7	8
visited	true	true	true	false	true	true	false	false



函数调用栈

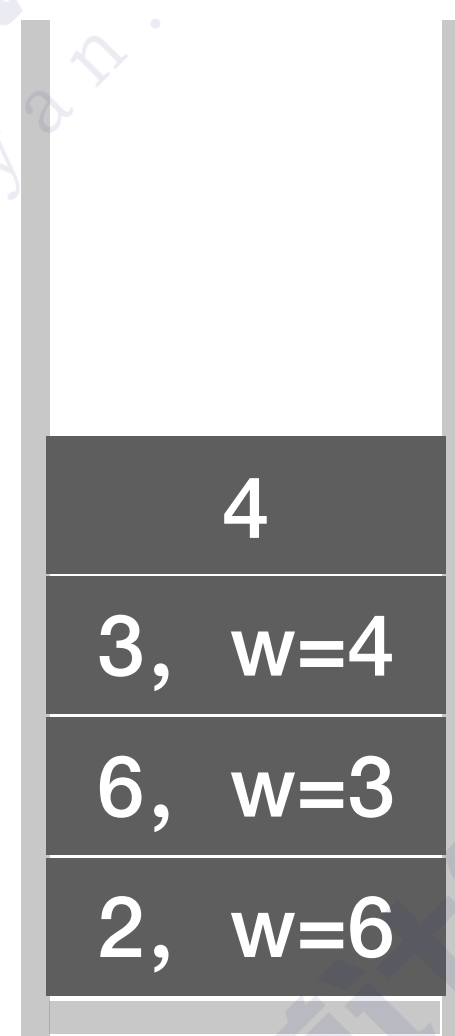
图的深度优先遍历

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
    }                             }
```

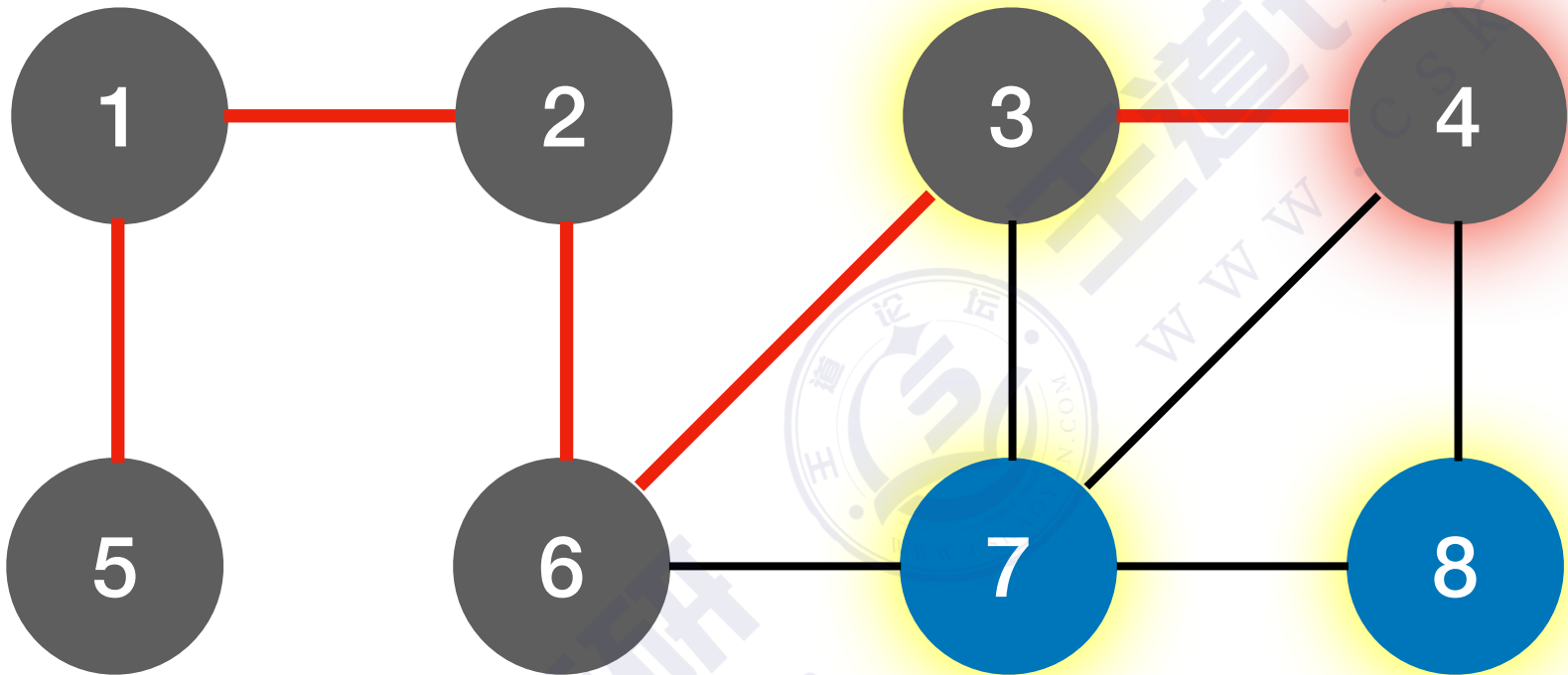


函数调用栈

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

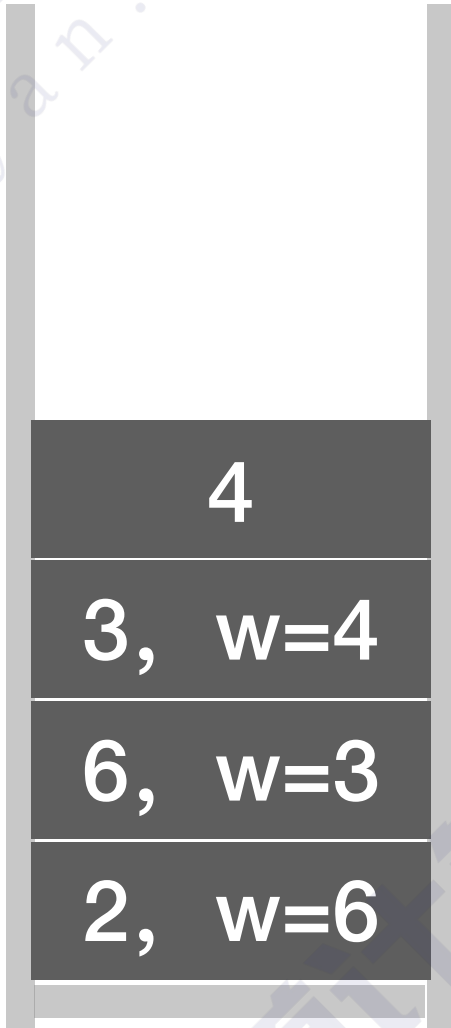
图的深度优先遍历

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

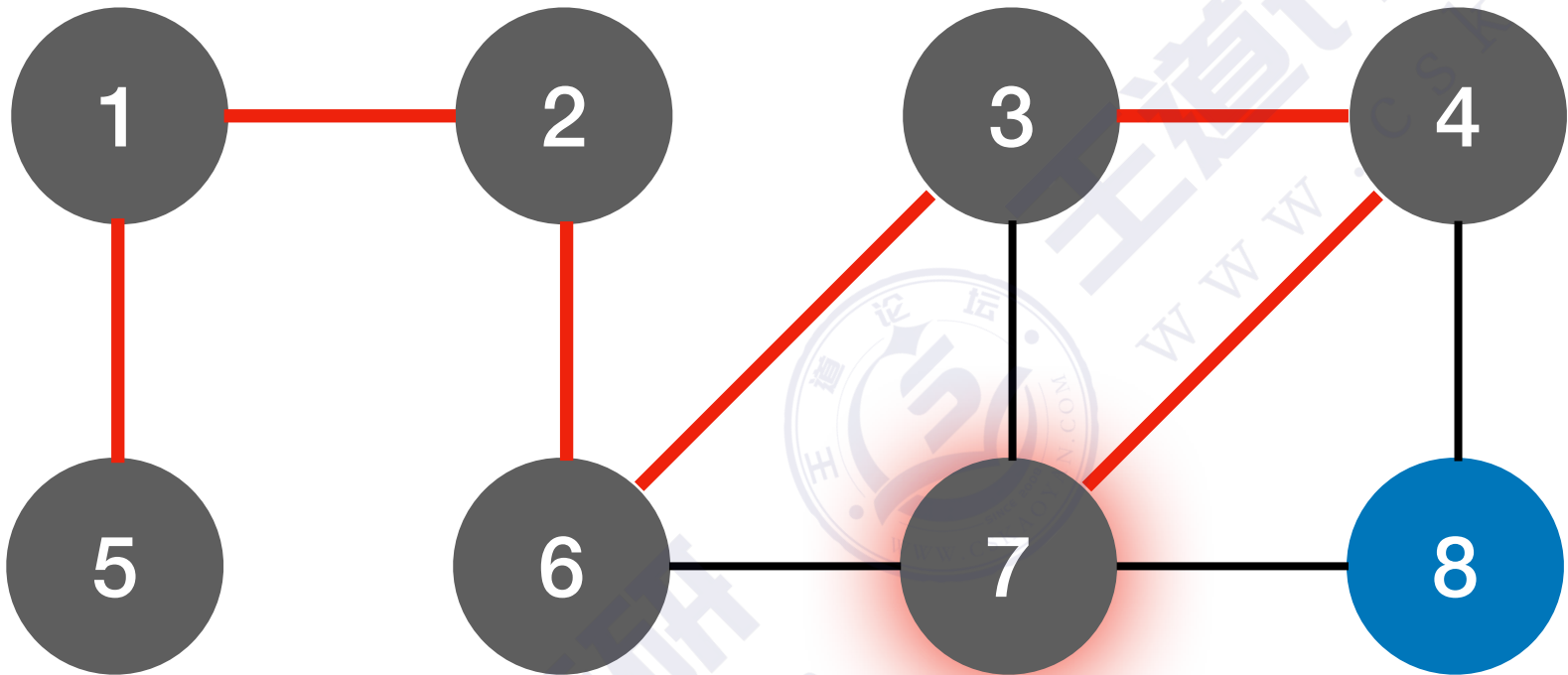


函数调用栈

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

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
    }
```

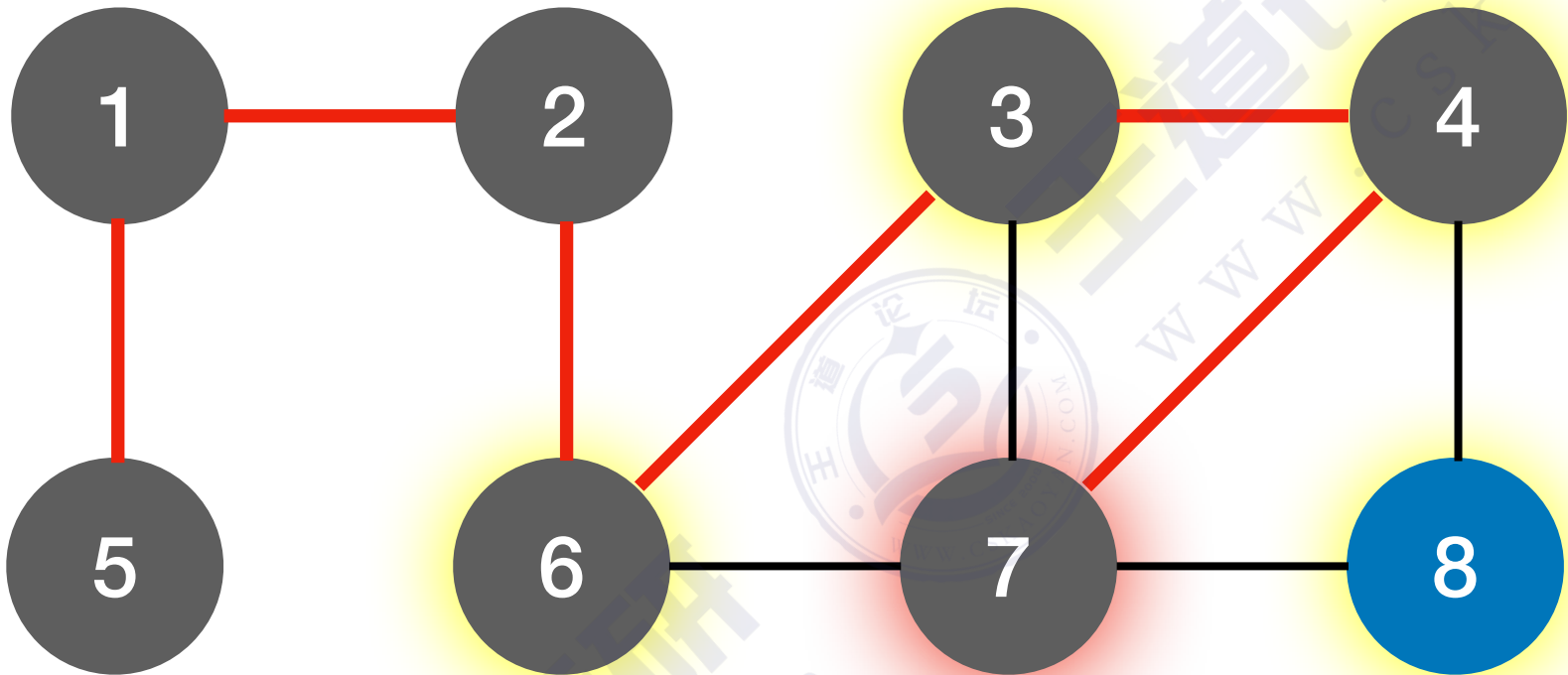
7
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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

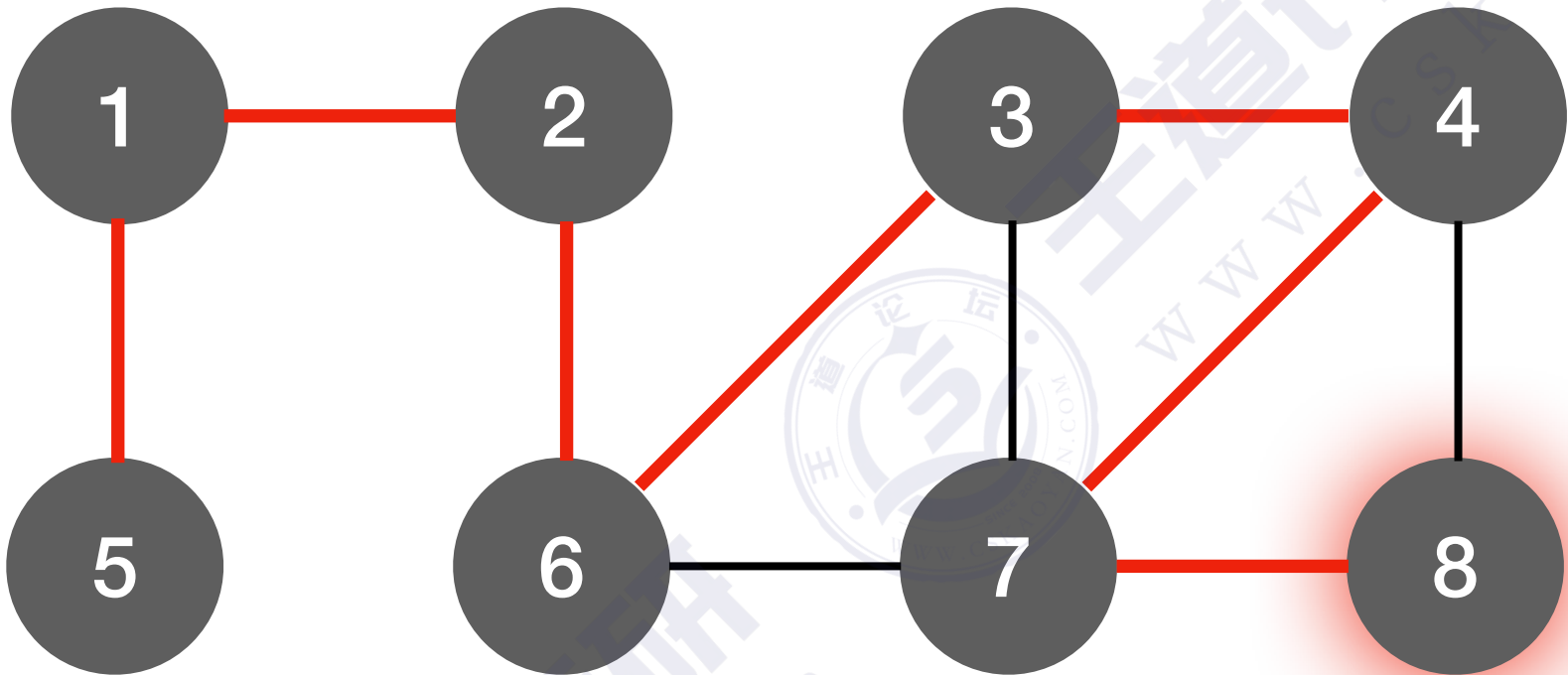
7
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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

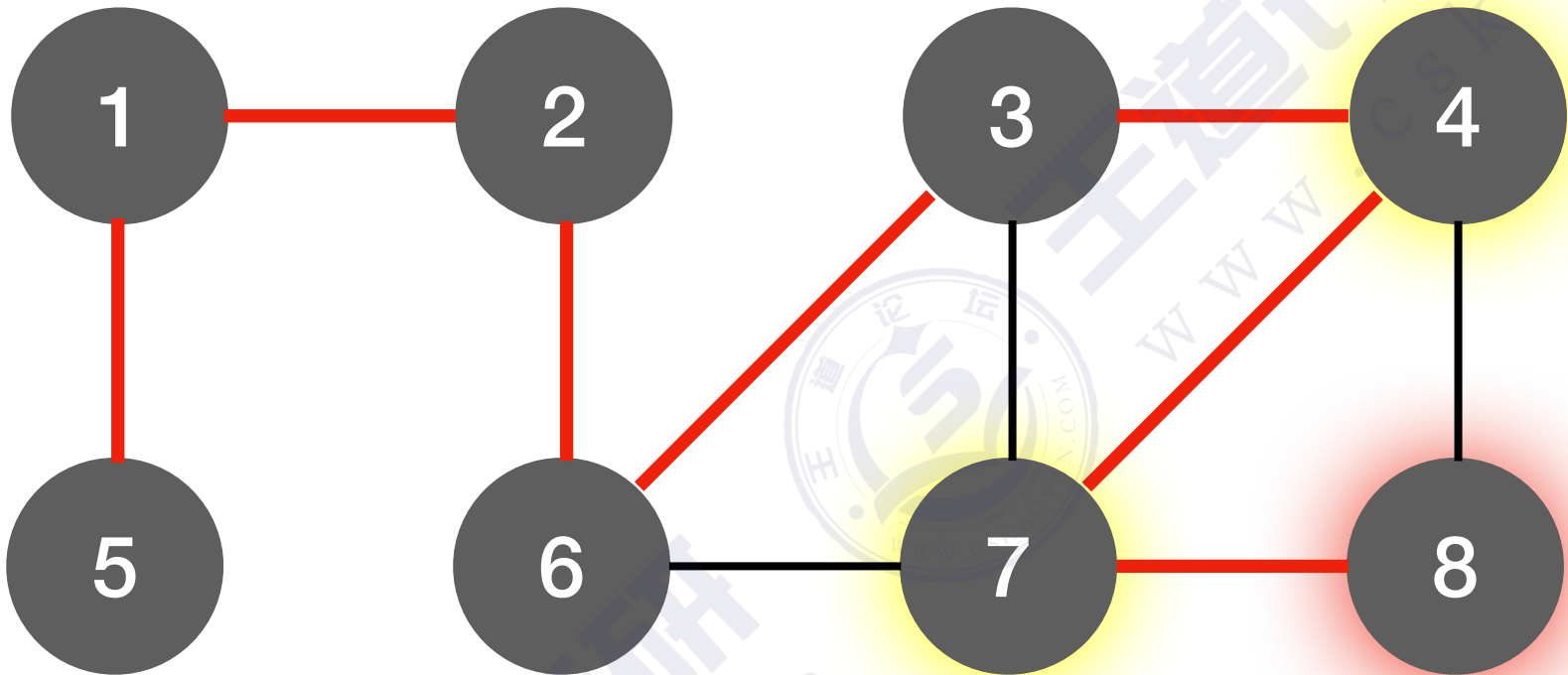
8
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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

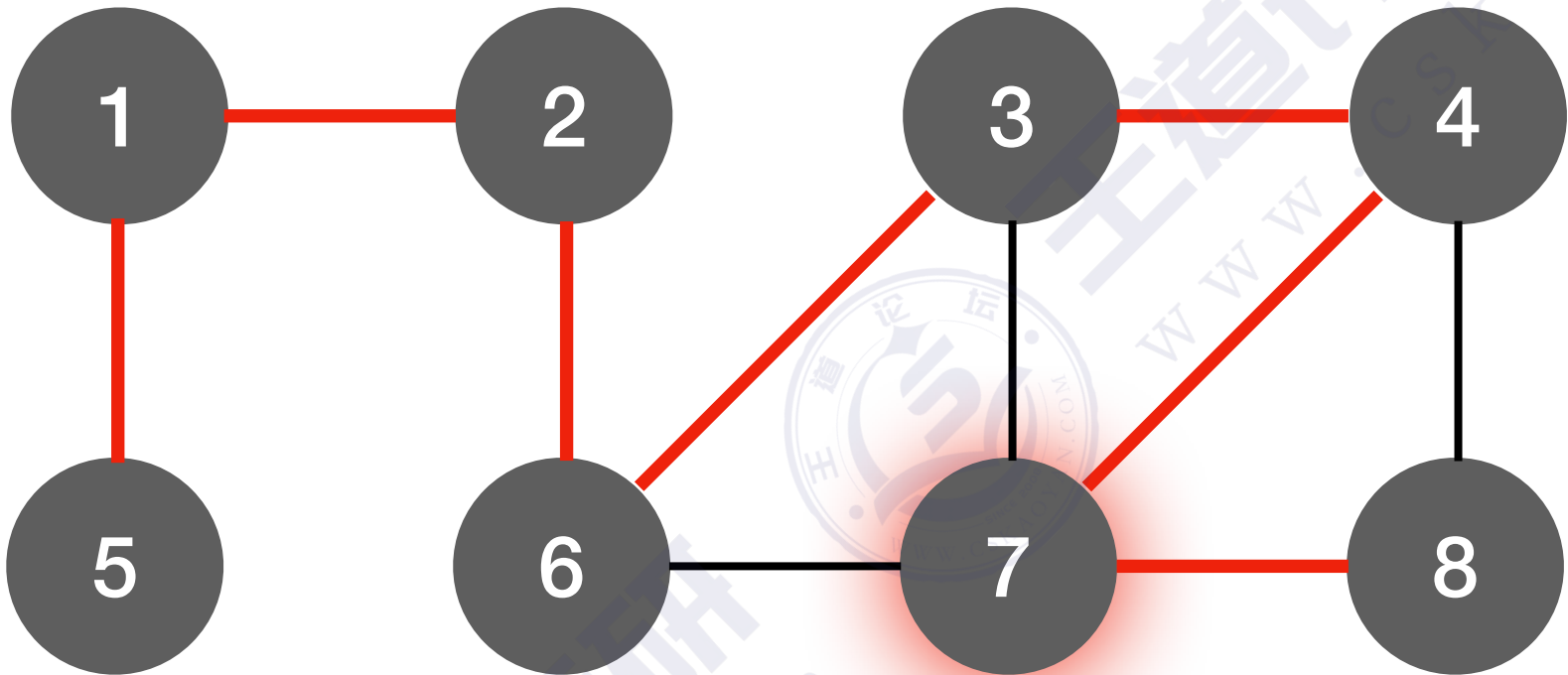
8
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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

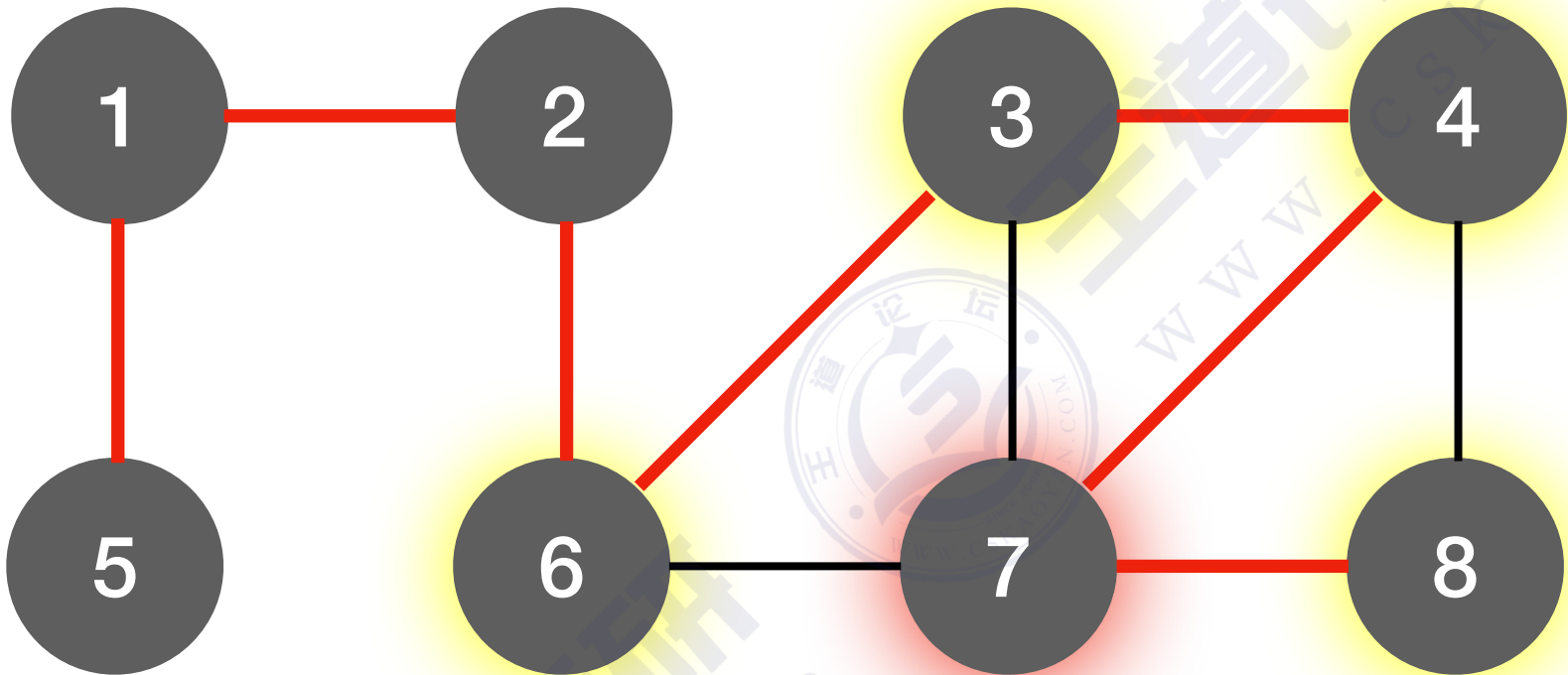
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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

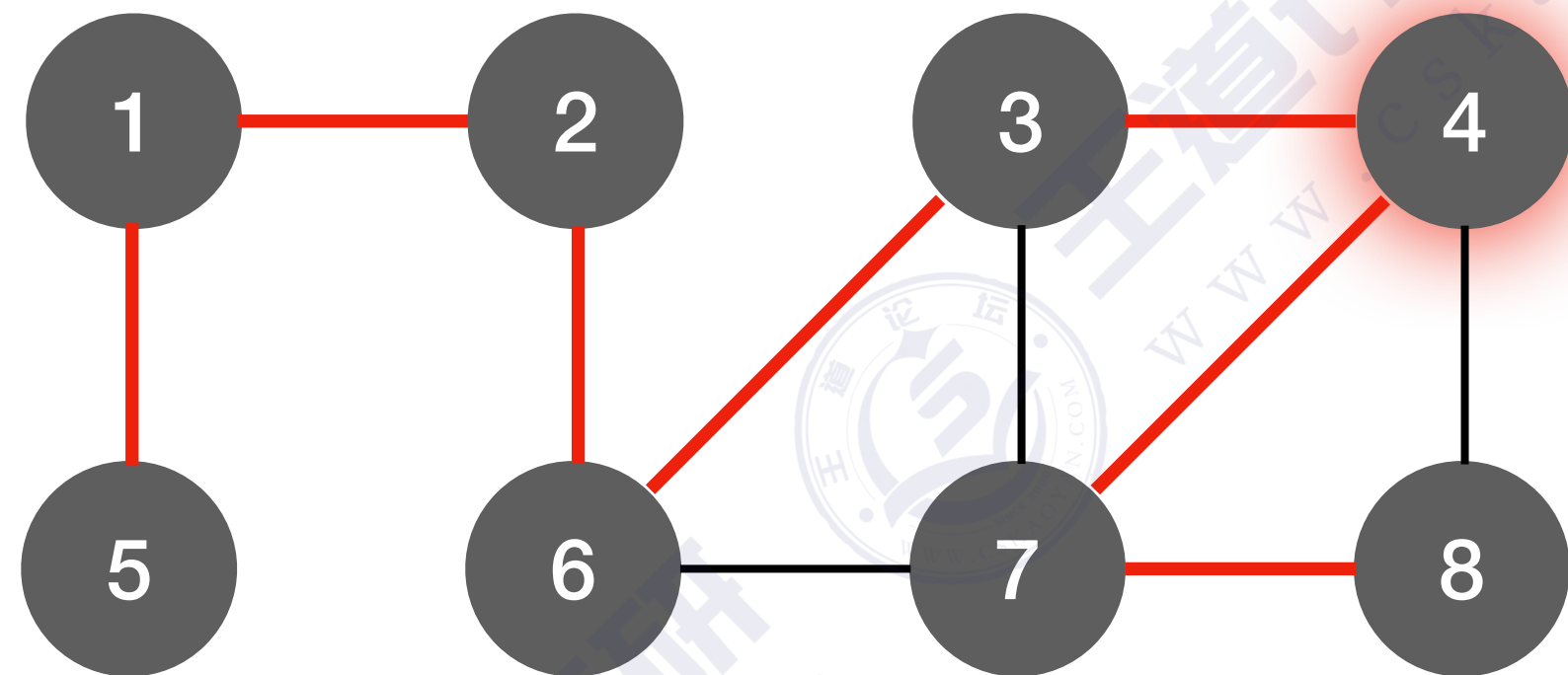
7, w=8
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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
    }
}
```

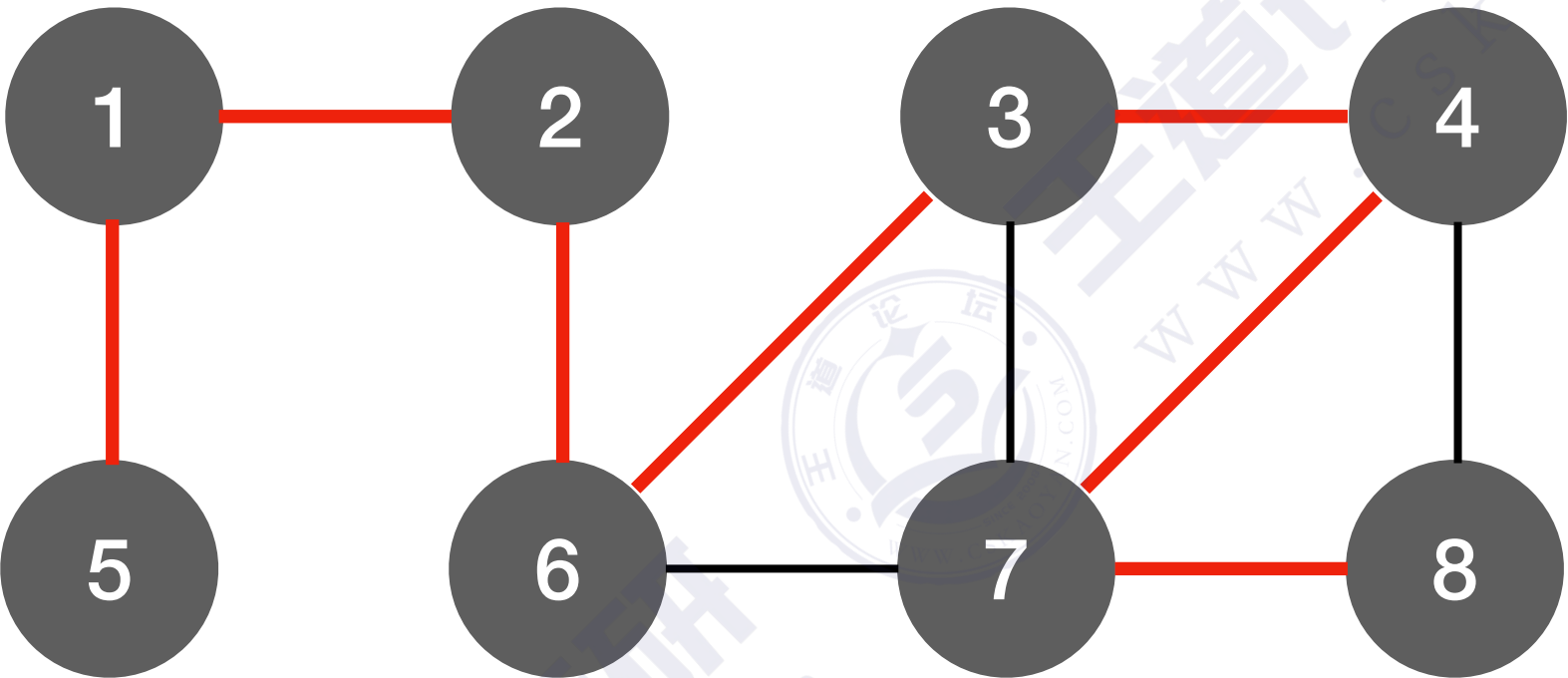
4, w=7
3, w=4
6, w=3
2, w=6

函数调用栈

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

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

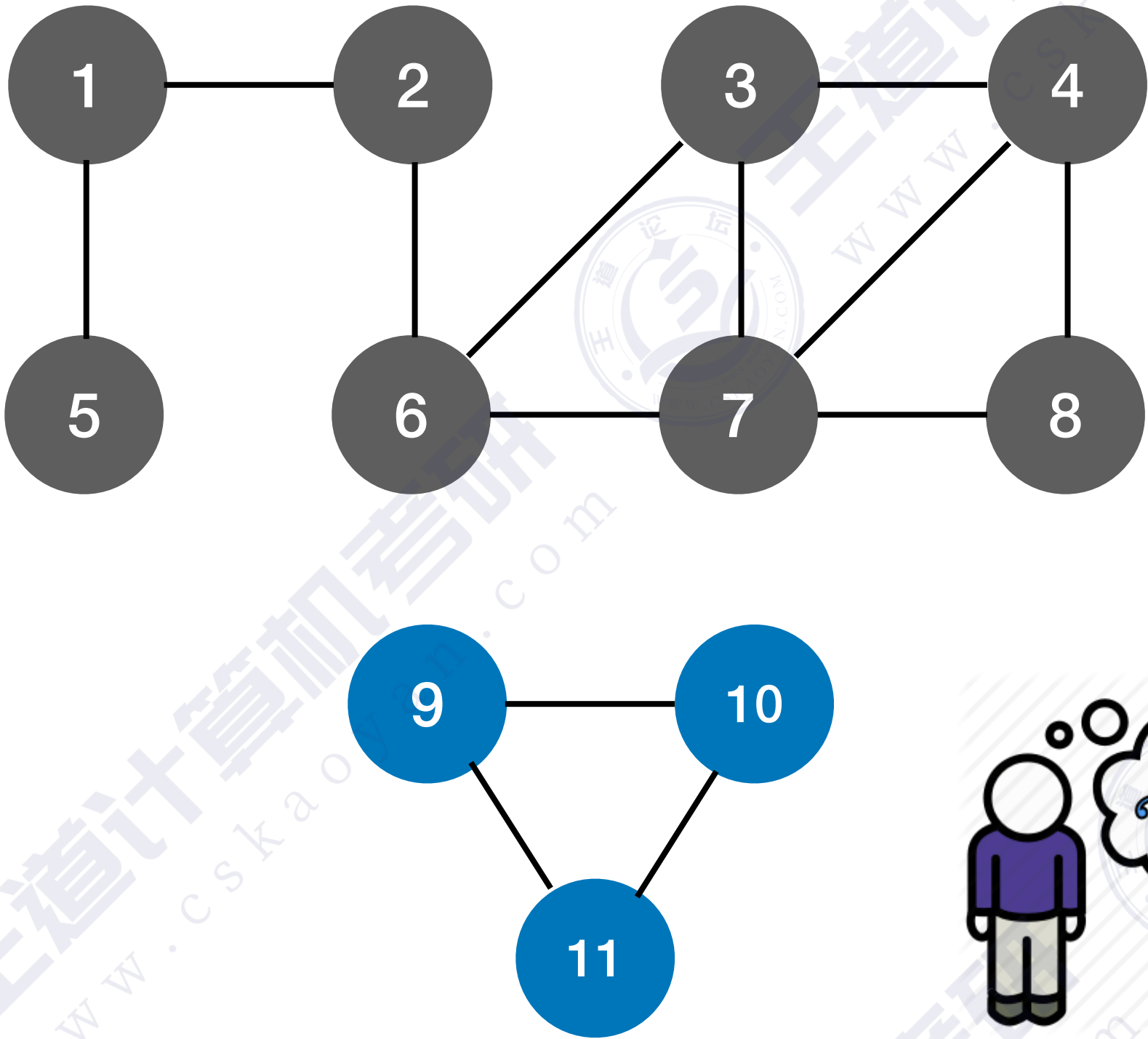
	1	2	3	4	5	6	7	8
visited	true	true	true	true	true	true	true	true

从2出发的深度遍历序列：2，1，5，6，3，4，7，8

函数调用栈

算法存在的问题

初始都为false



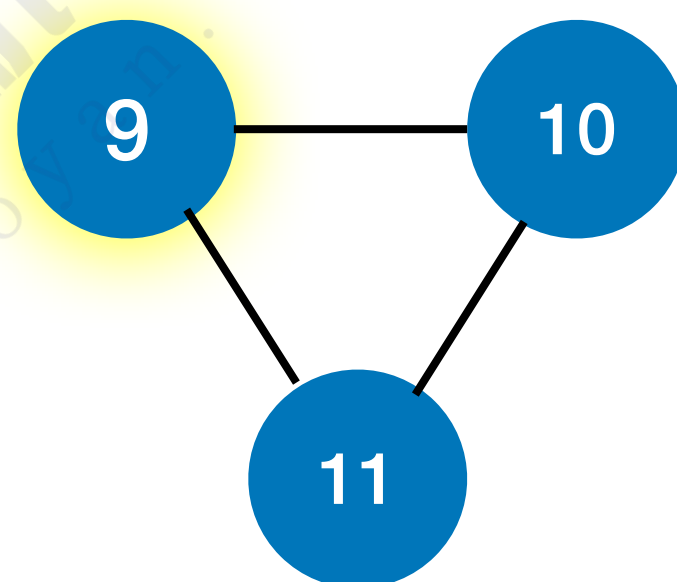
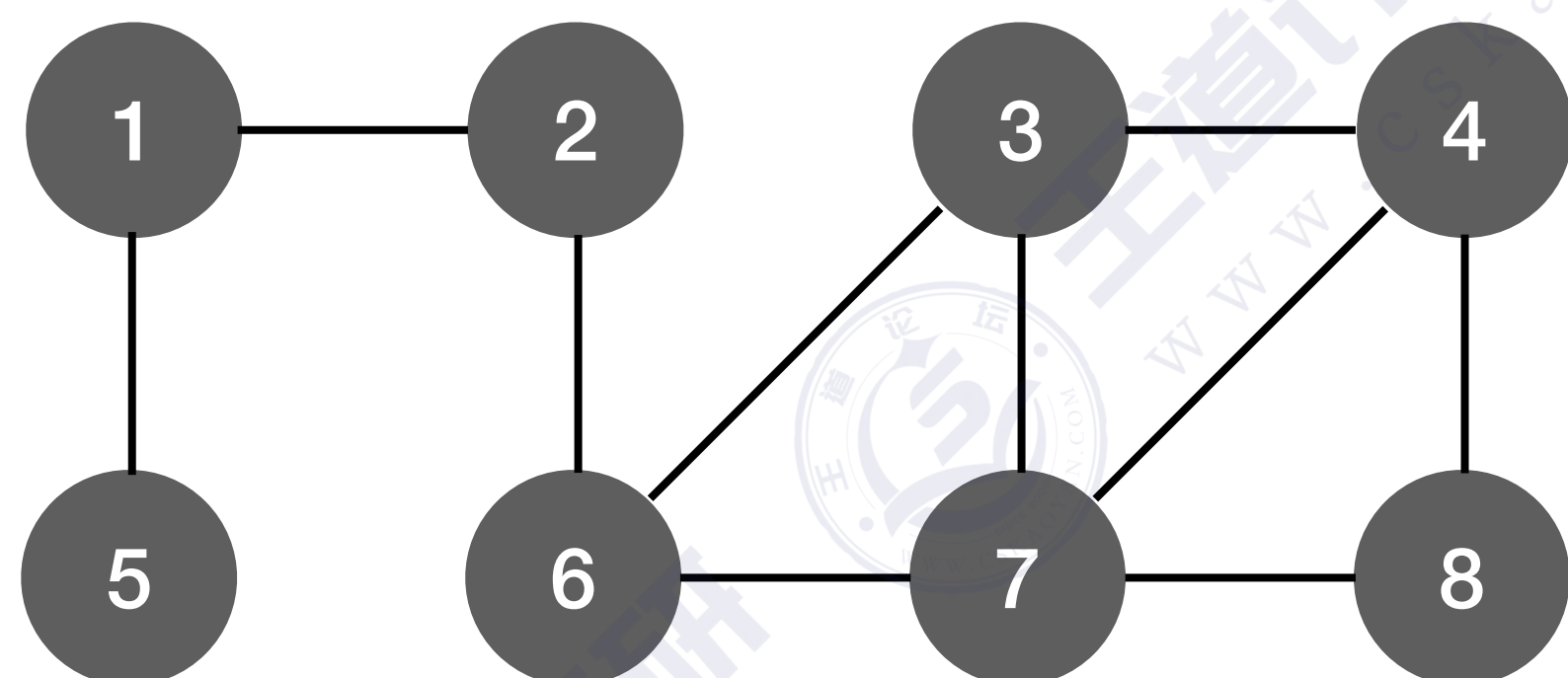
```
bool visited[MAX_VERTEX_NUM]; //访问标记数组

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

DFS算法 (Final版)



如果是非连通图，则无法遍历完所有结点

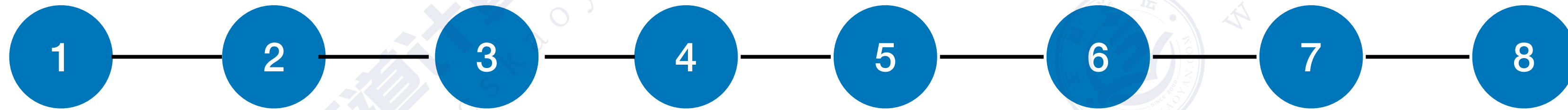
```
bool visited[MAX_VERTEX_NUM];    //访问标记数组

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)
        if(!visited[v])
            DFS(G,v);
}

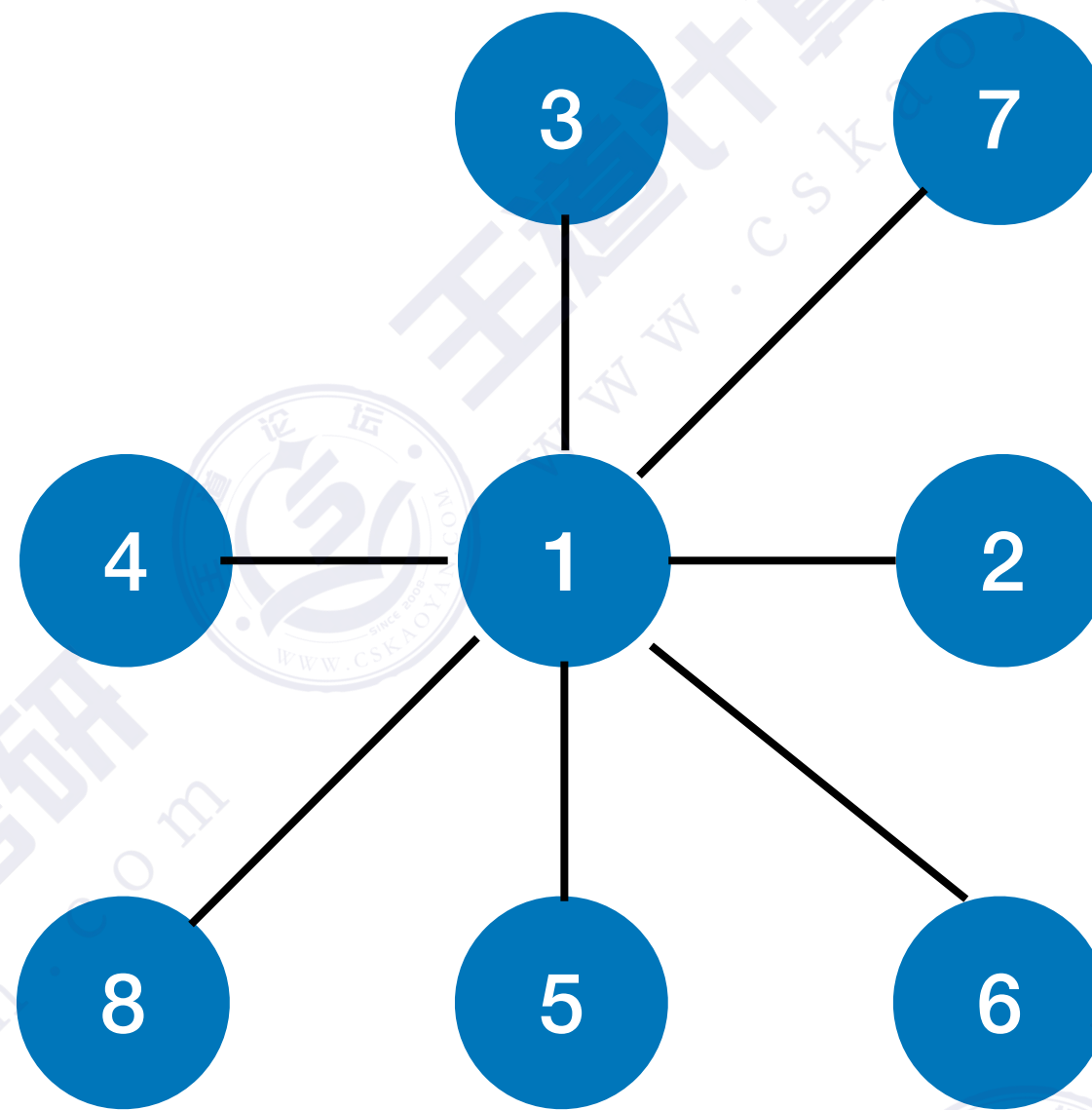
void DFS(Graph G,int v){         //从顶点v出发，深度优先遍历图G
    visit(v);                    //访问顶点v
    visited[v]=TRUE;             //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){        //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
}
```

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

复杂度分析

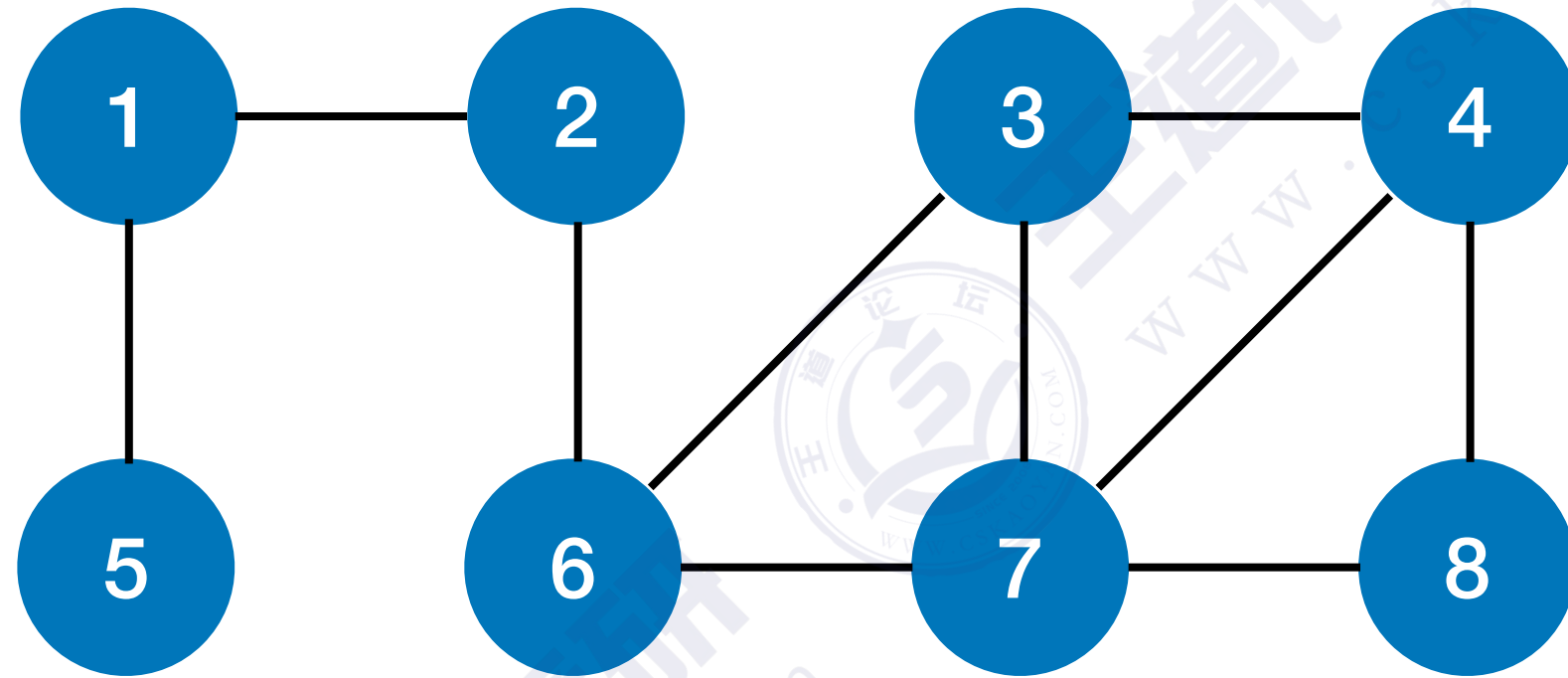


空间复杂度：来自函数调用栈，**最坏情况，递归深度为 $O(|V|)$**



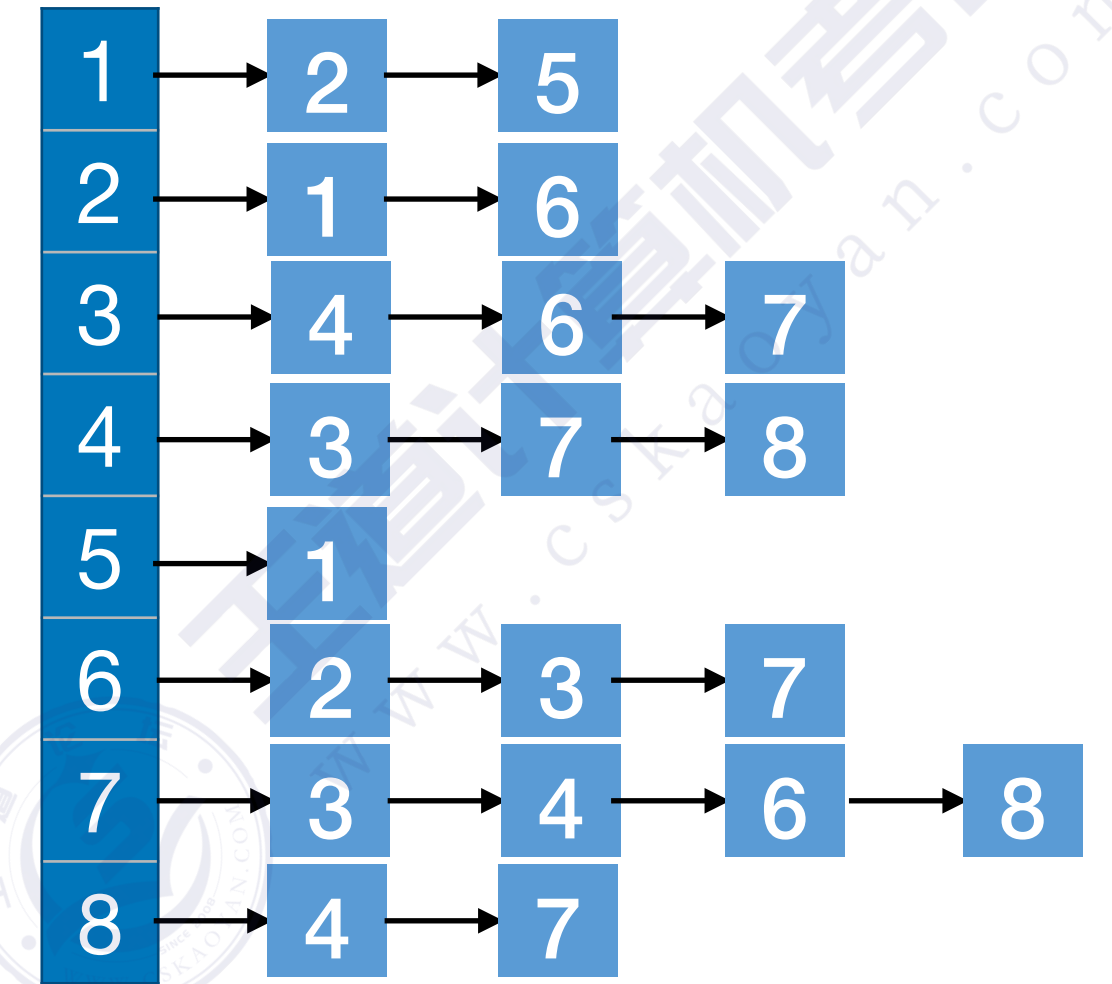
空间复杂度：最好情况， $O(1)$

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



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= $O(|V|^2)$

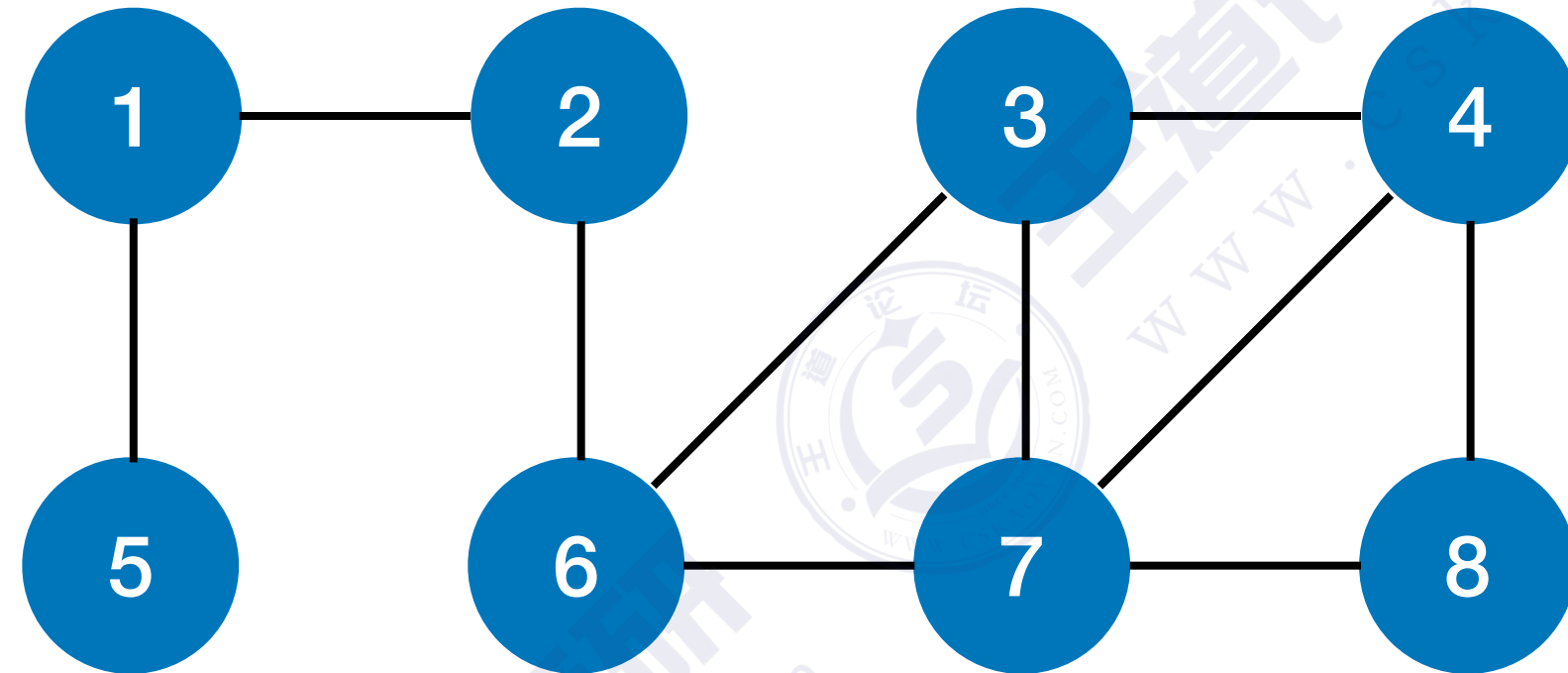
邻接表存储的图:

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,

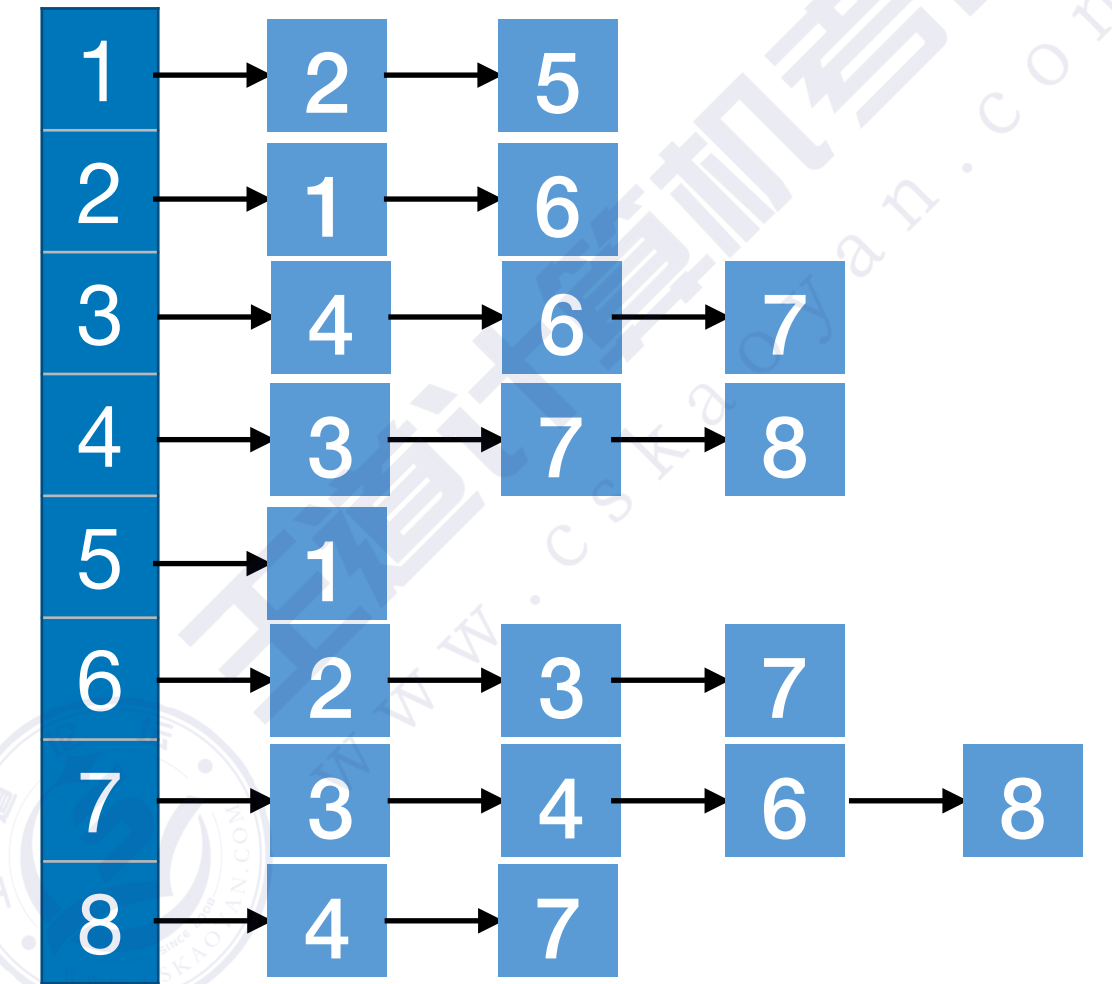
时间复杂度= $O(|V|+|E|)$

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	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



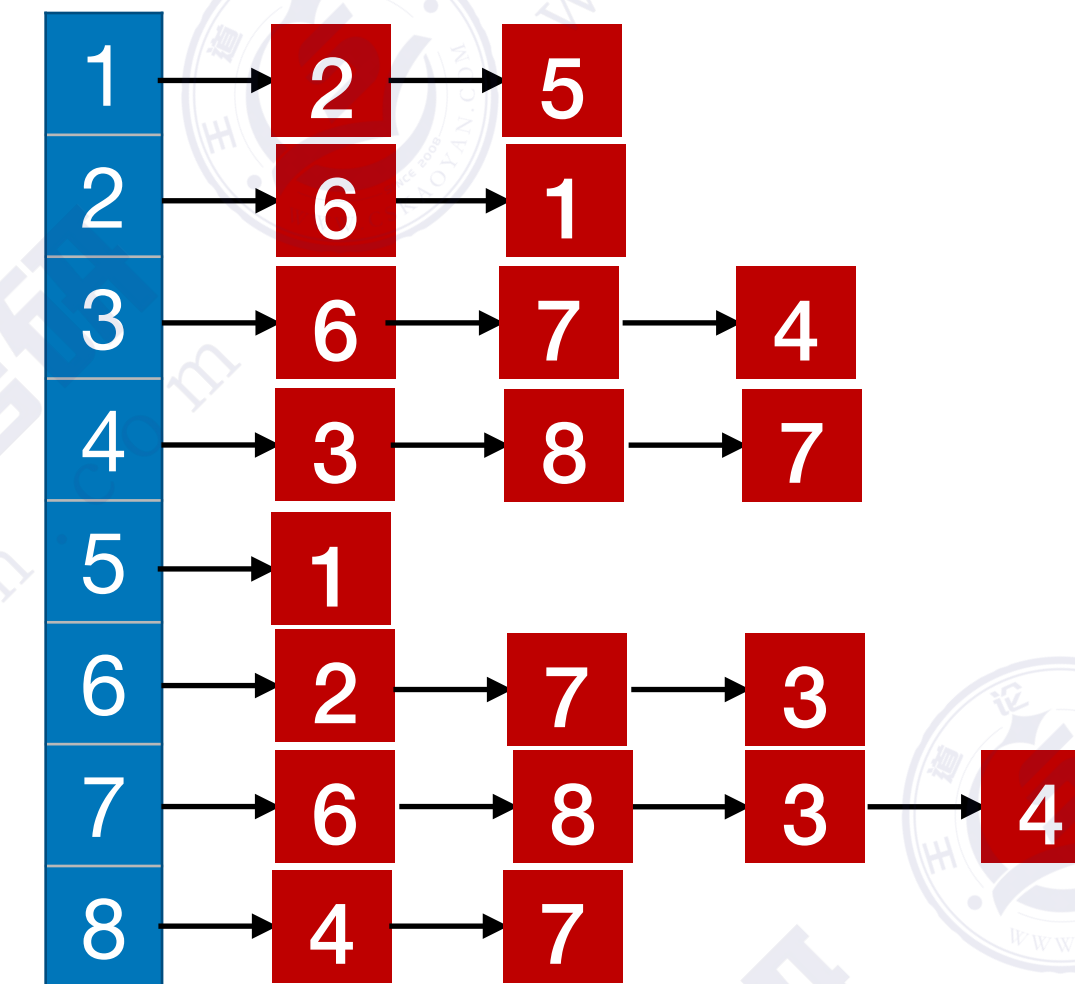
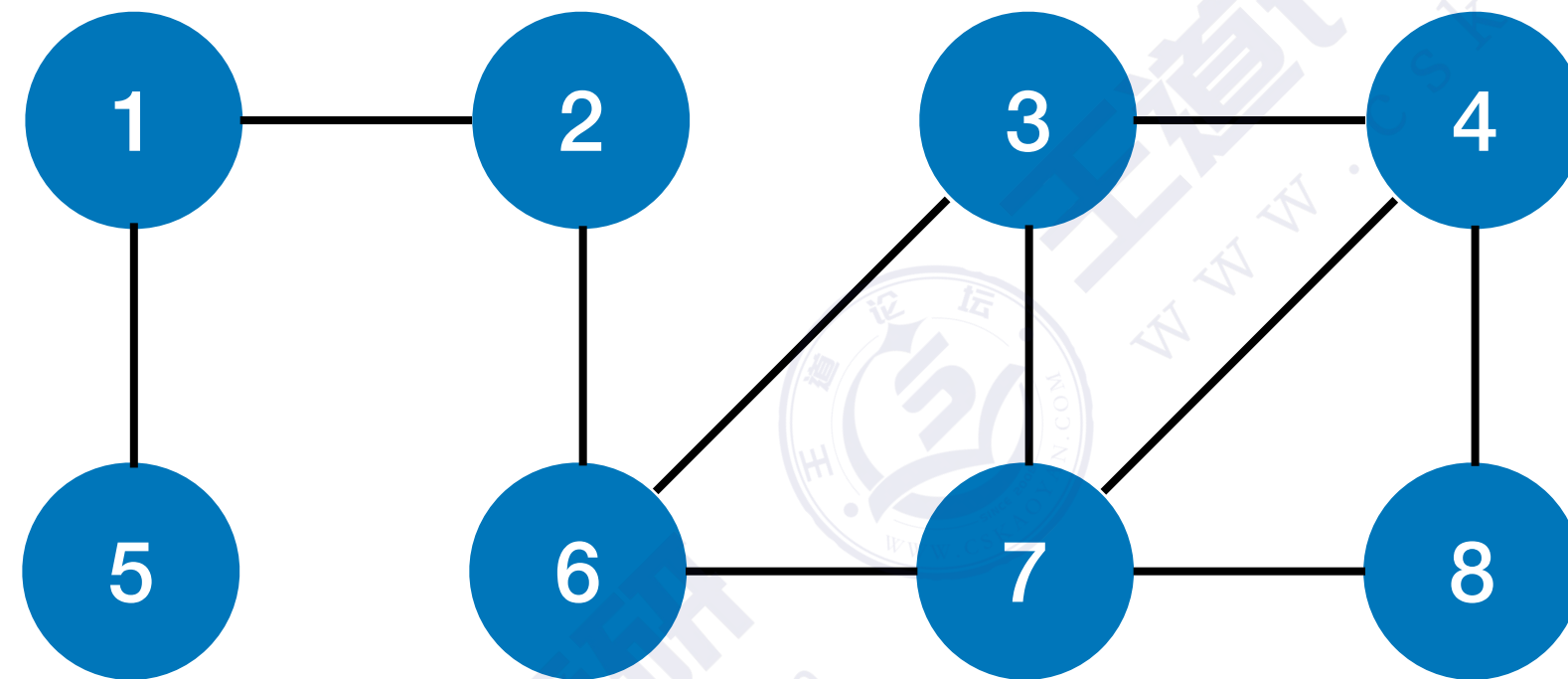
邻接表

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

深度优先遍历序列



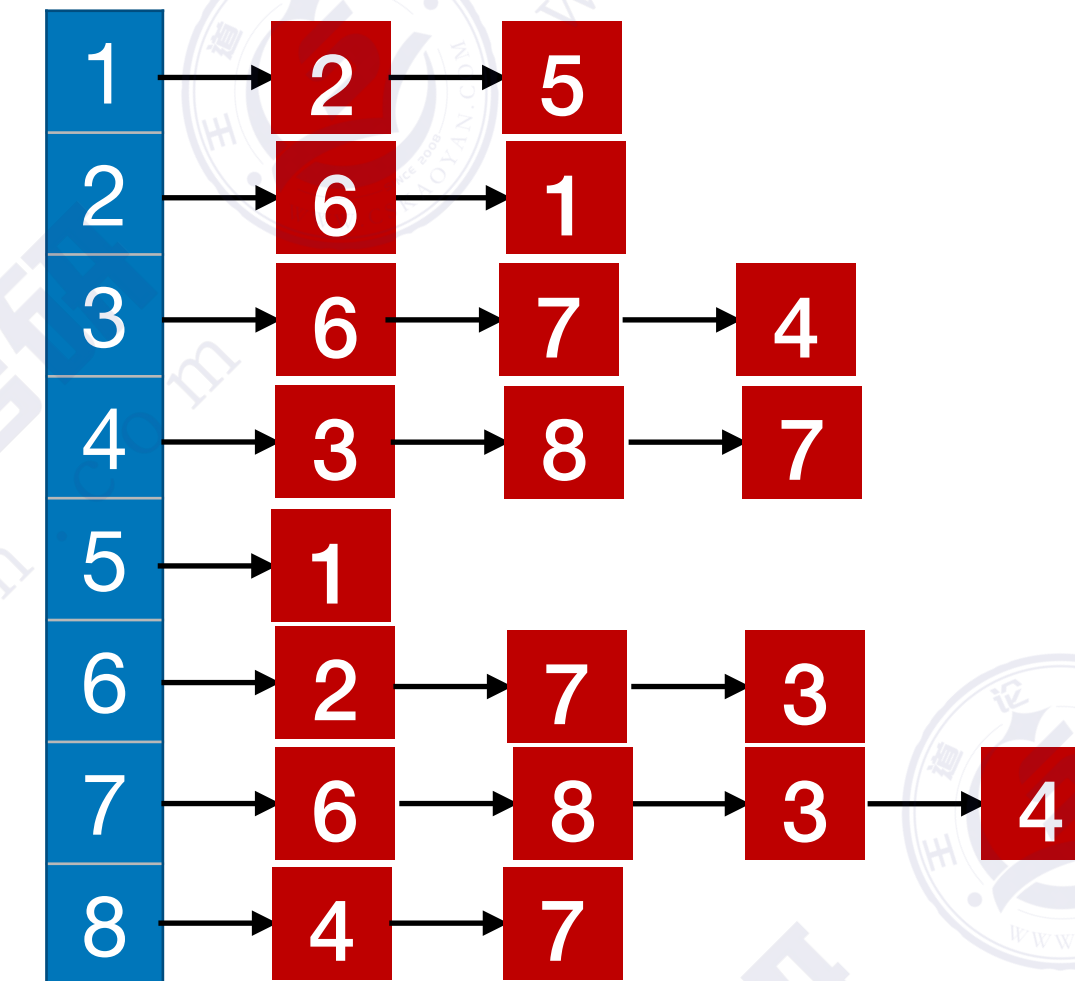
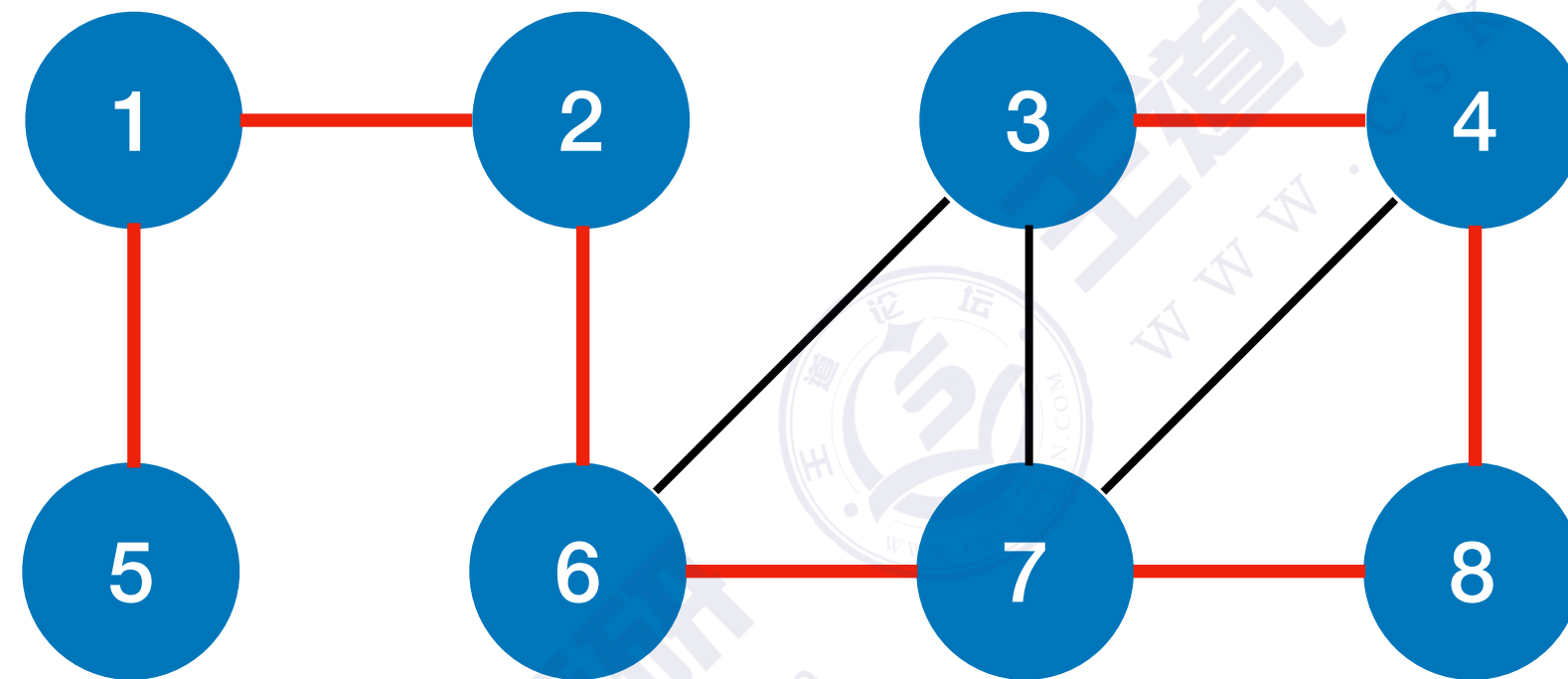
邻接表

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？

深度优先遍历序列



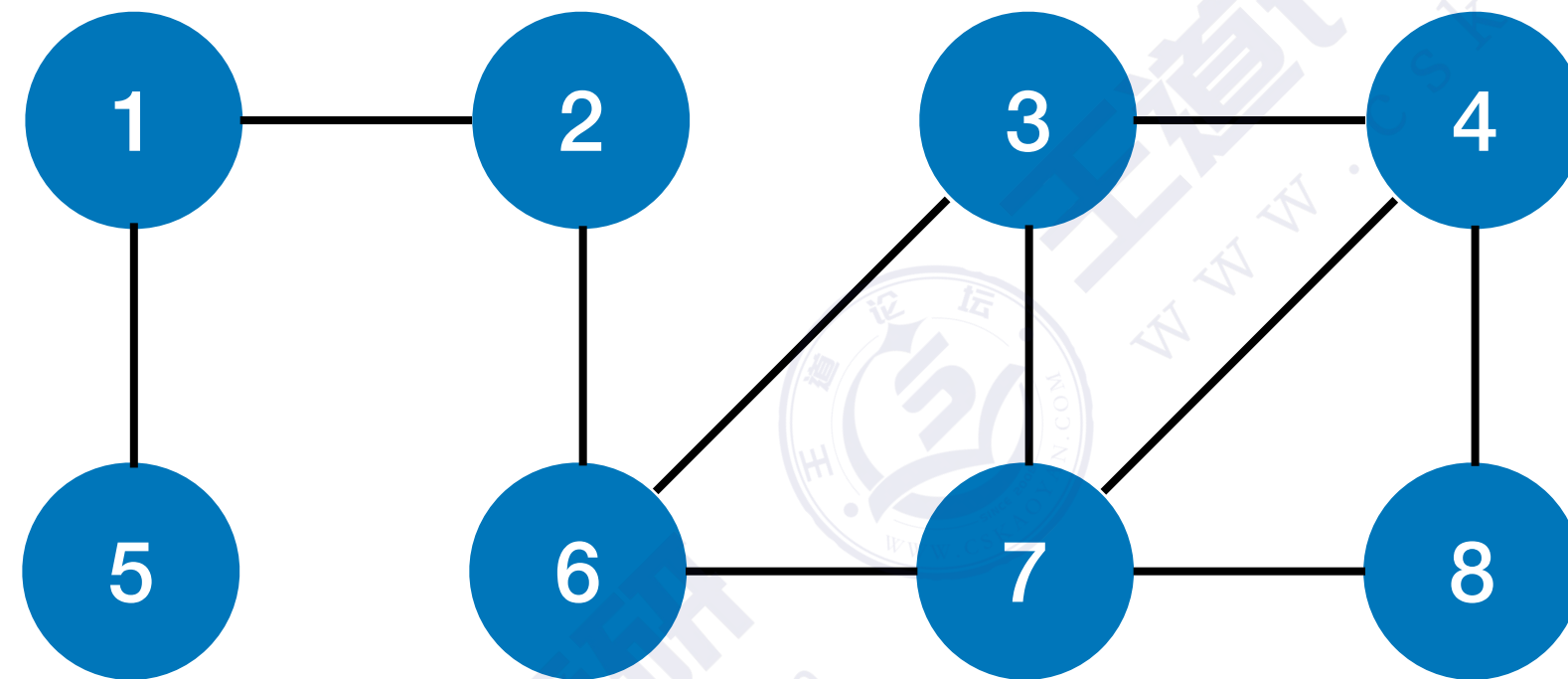
邻接表

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？

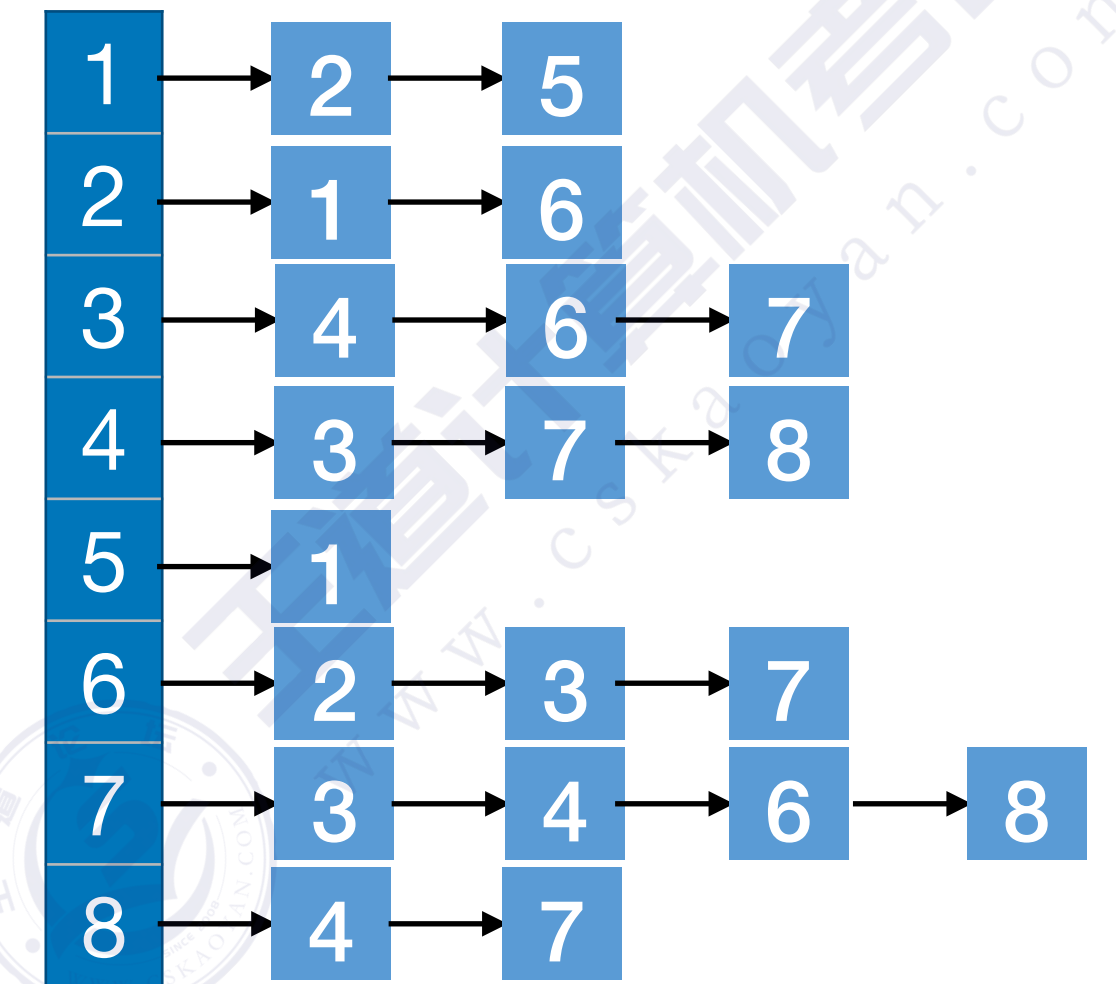
从1出发的深度优先遍历序列？

深度优先遍历序列

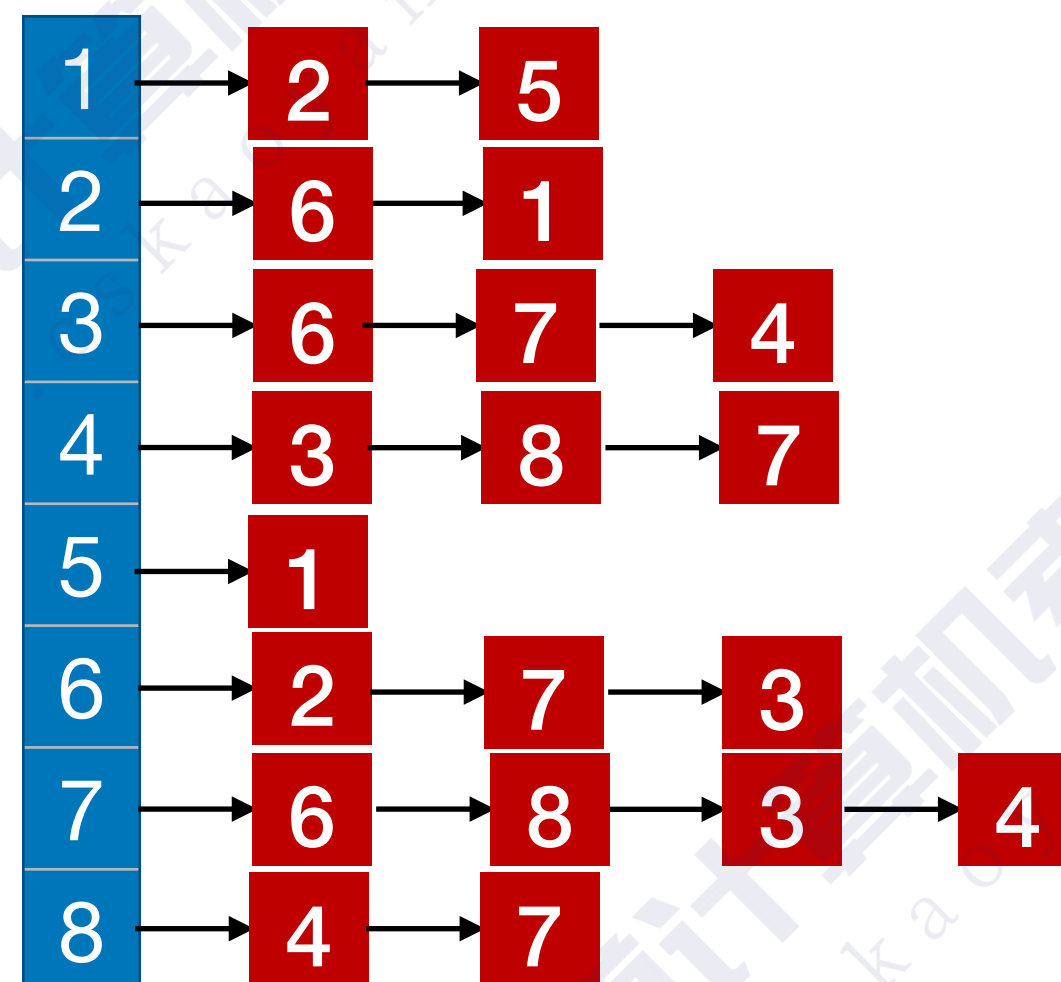


	1	2	3	4	5	6	7	8
1	0	1	0	0	1	0	0	0
2	1	0	0	0	0	1	0	0
3	0	0	0	1	0	1	1	0
4	0	0	1	0	0	0	1	1
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	1	0
7	0	0	1	1	0	1	0	1
8	0	0	0	1	0	0	1	0

邻接矩阵



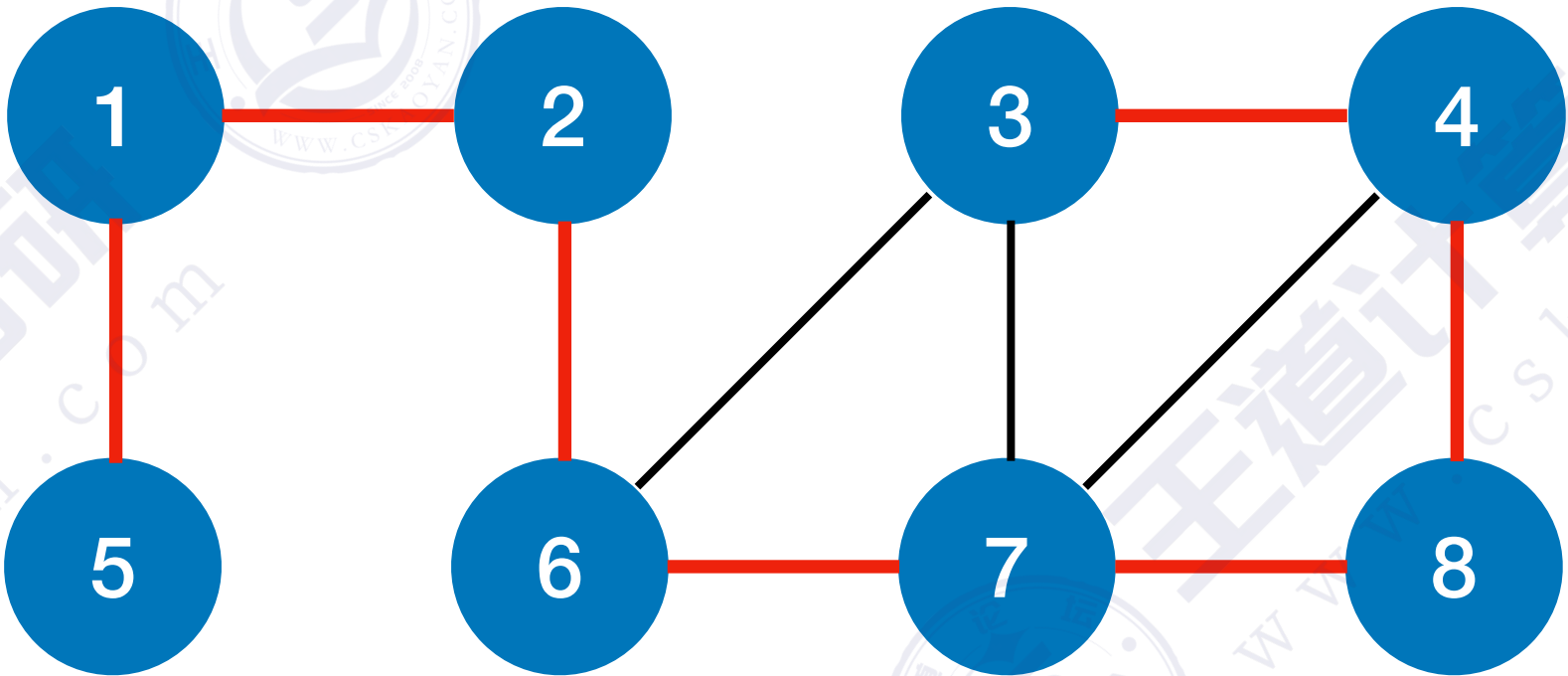
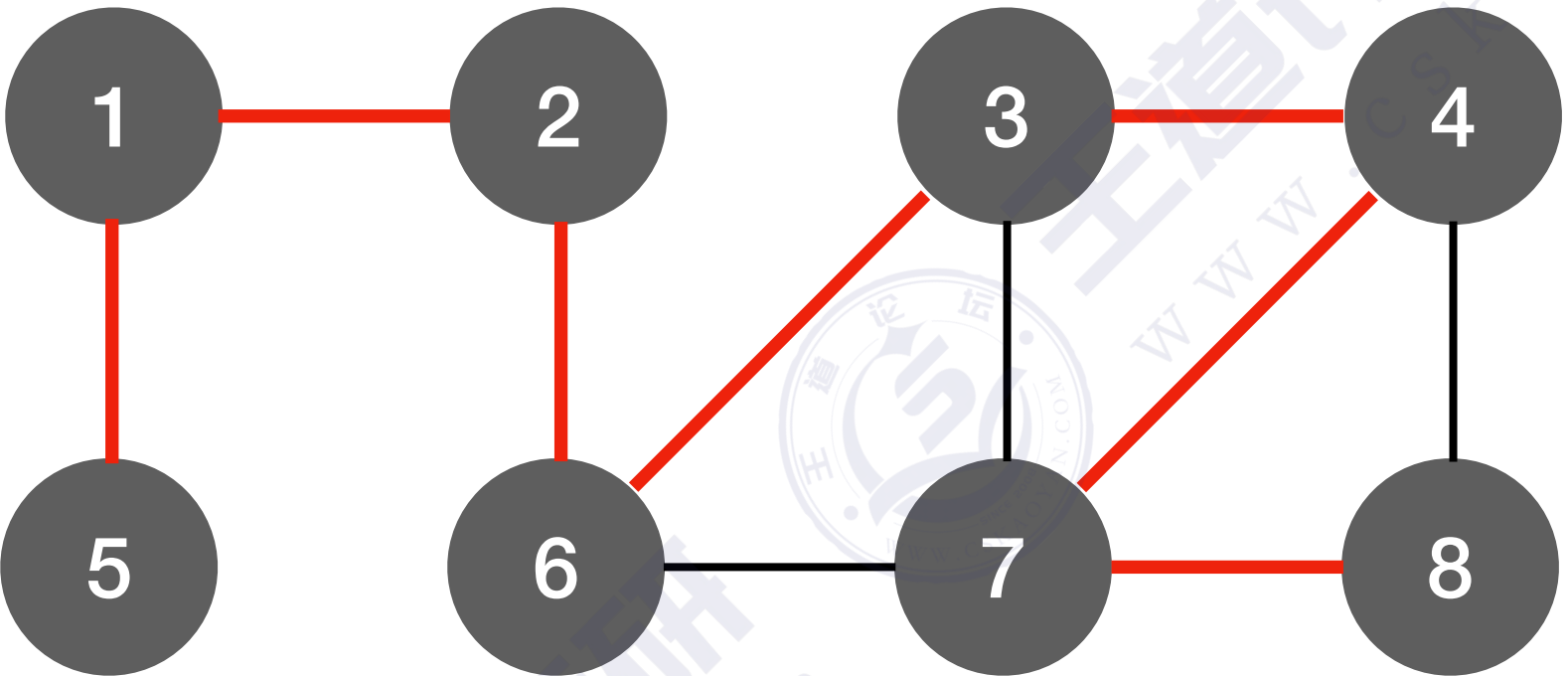
邻接表



邻接表

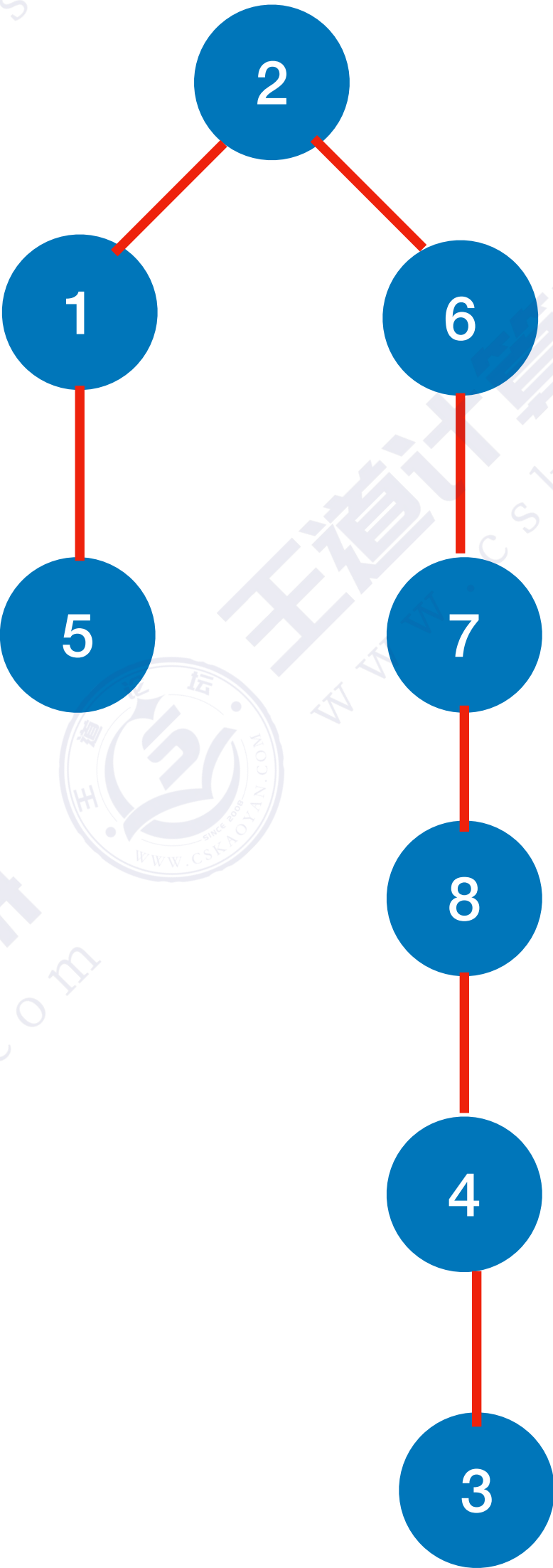
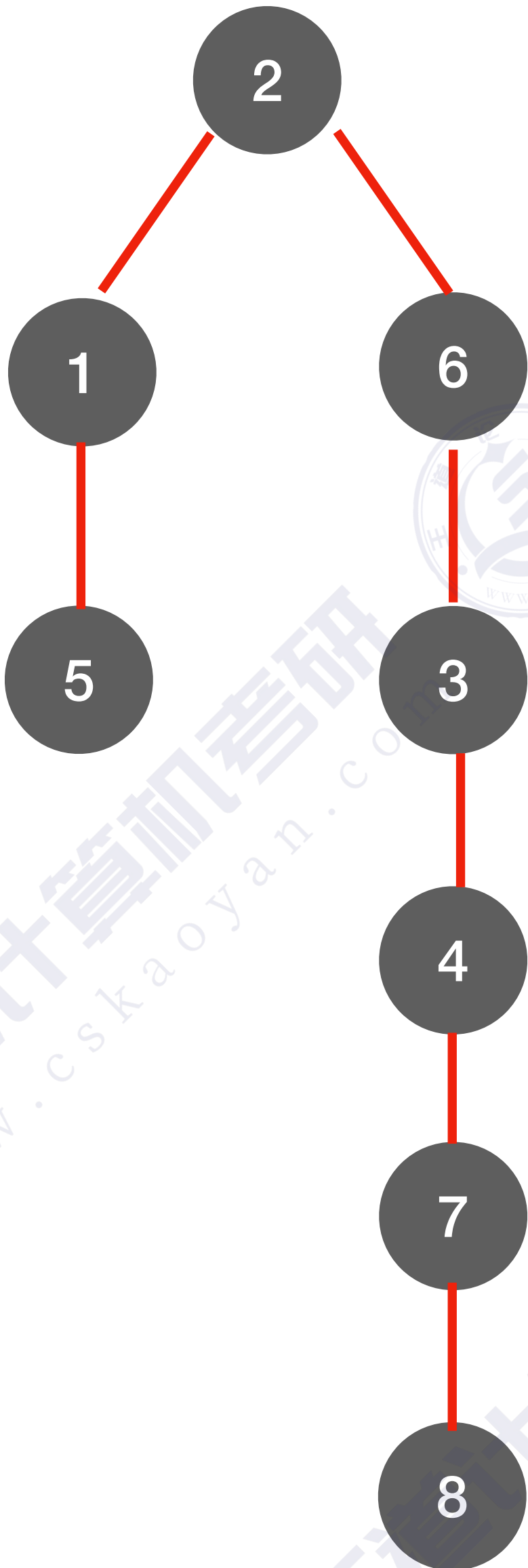
同一个图的邻接矩阵表示方式**唯一**，因此深度优先遍历序列**唯一**
同一个图邻接表表示方式**不唯一**，因此深度优先遍历序列**不唯一**

深度优先生成树

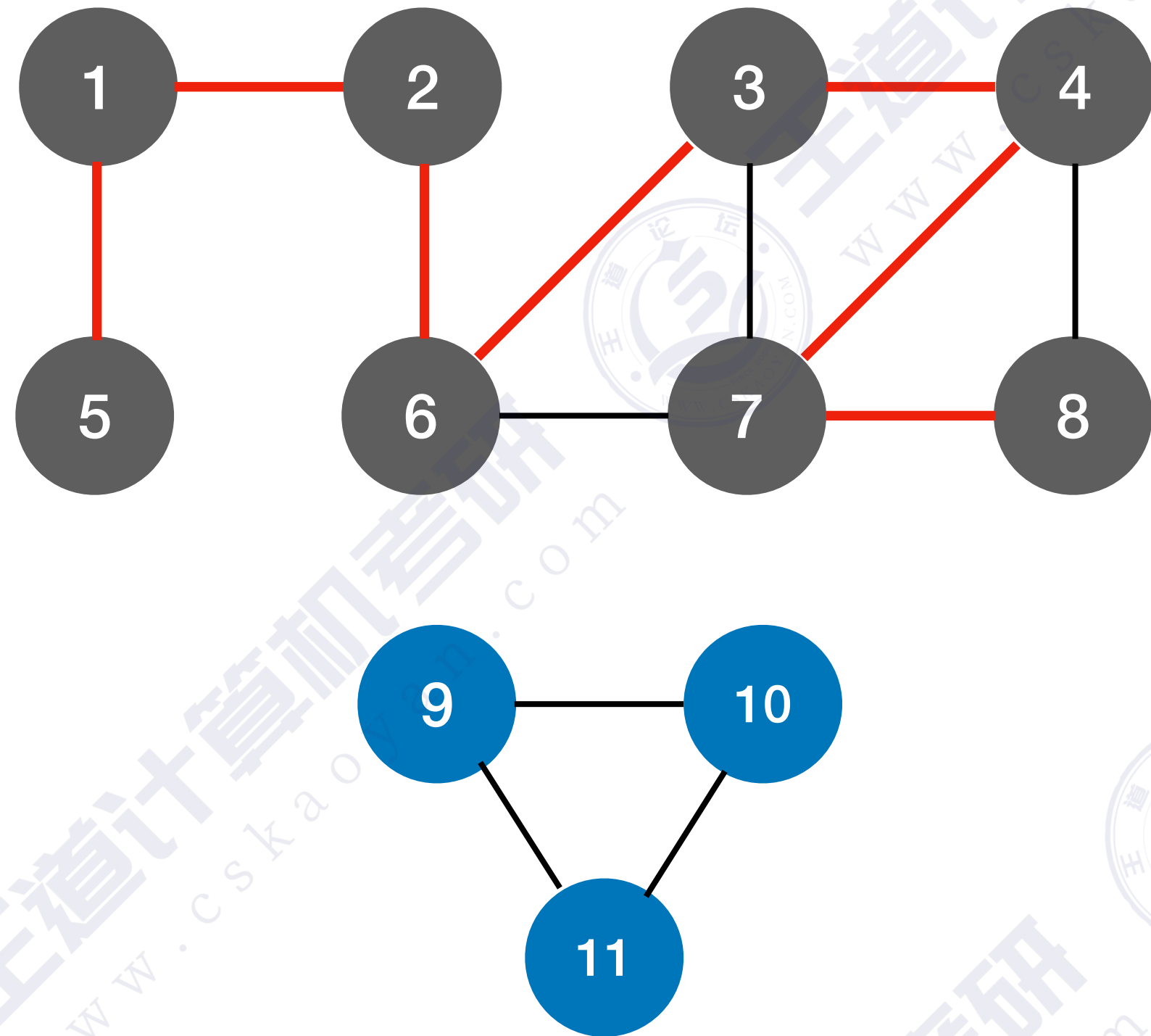


同一个图的邻接矩阵表示方式**唯一**，因此深度优先遍历序列**唯一**，深度优先生成树也**唯一**
同一个图邻接表表示方式**不唯一**，因此深度优先遍历序列**不唯一**，深度优先生成树也**不唯一**

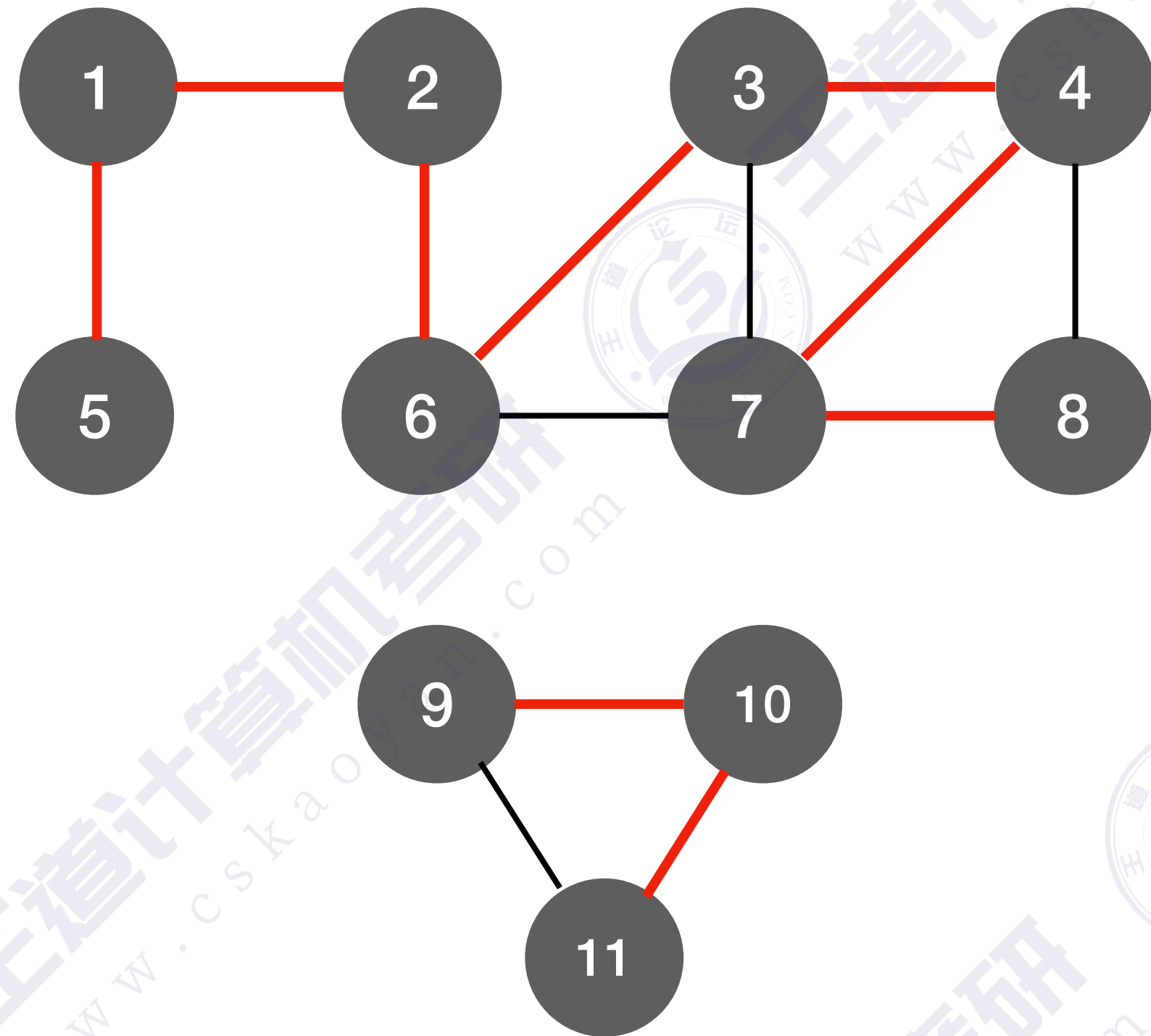
深度优先生成树



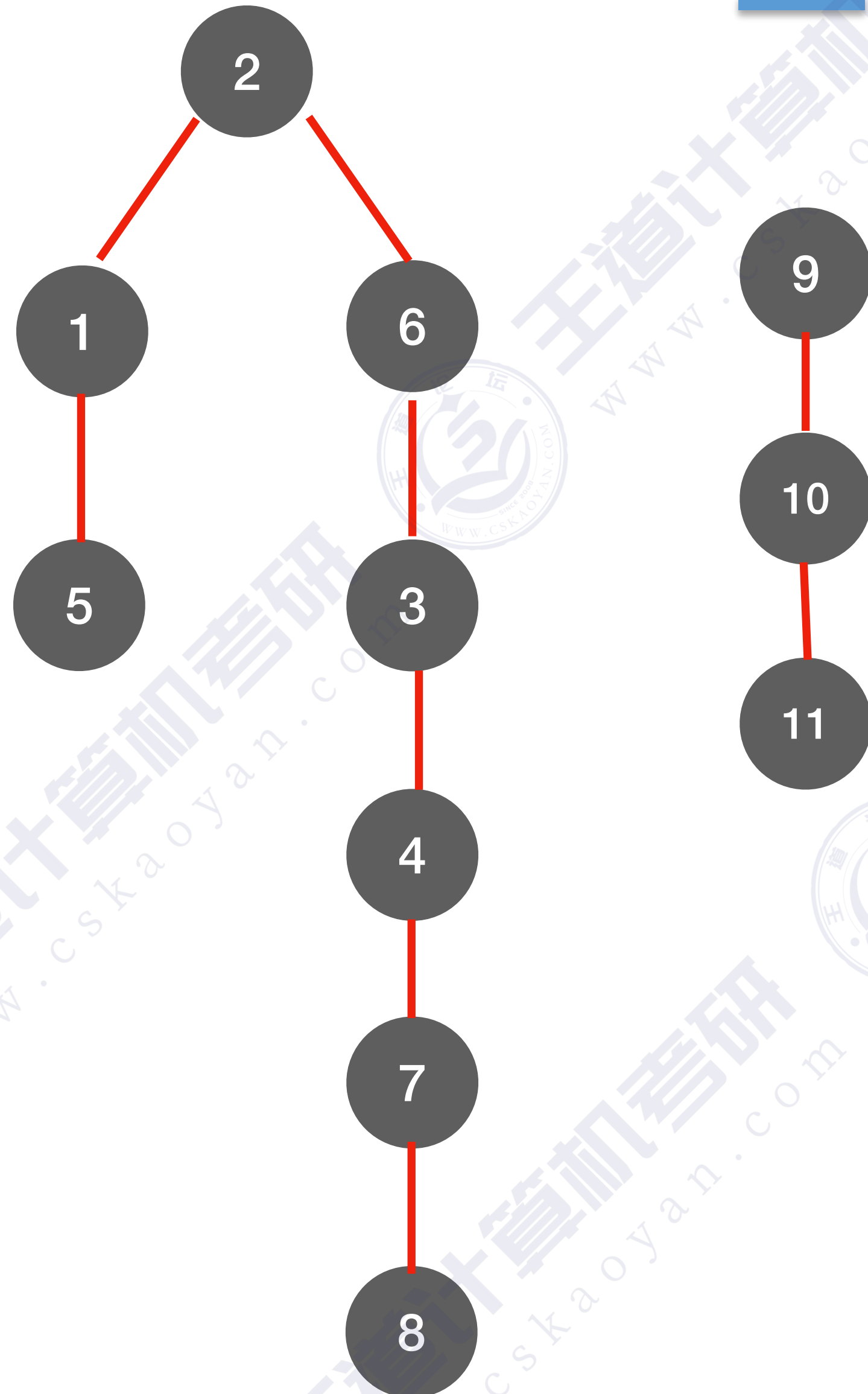
深度优先生成森林



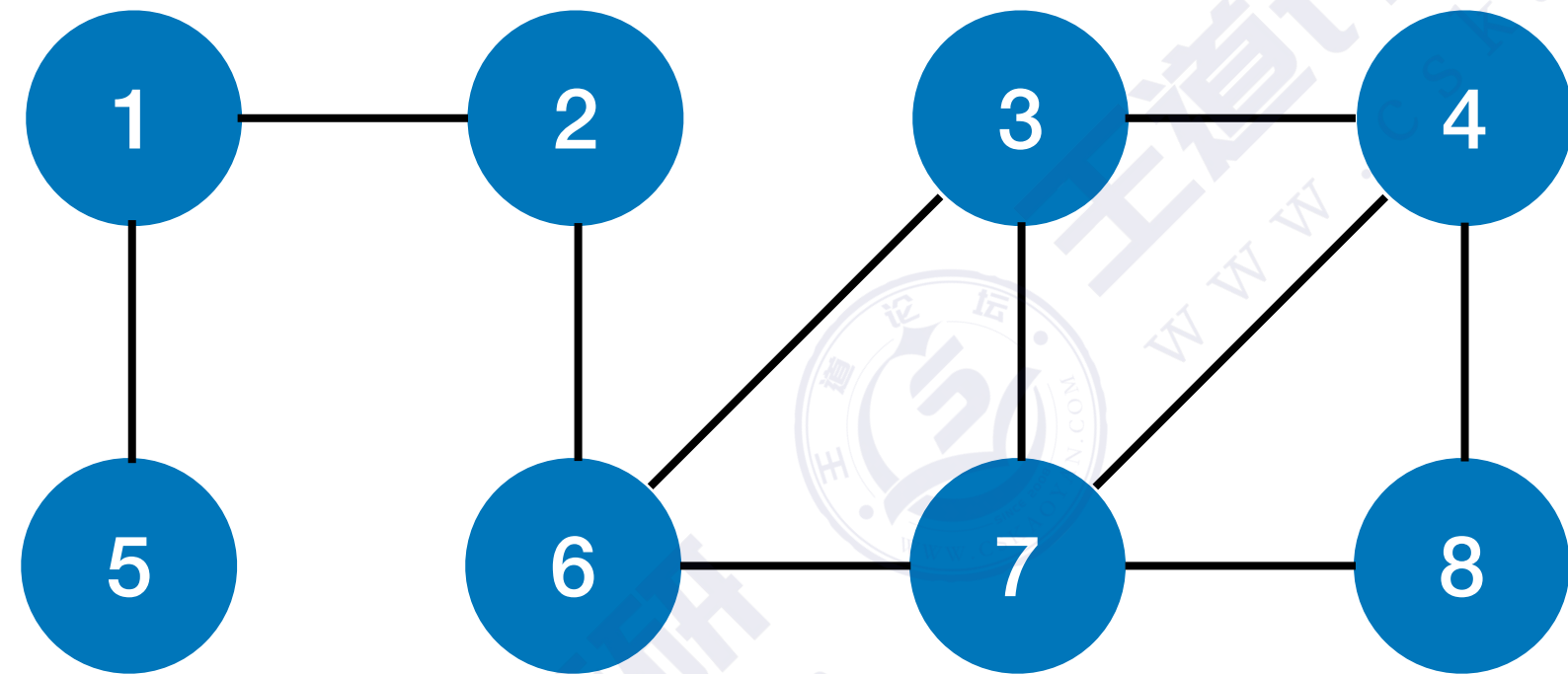
深度优先生成森林



深度优先生成森林

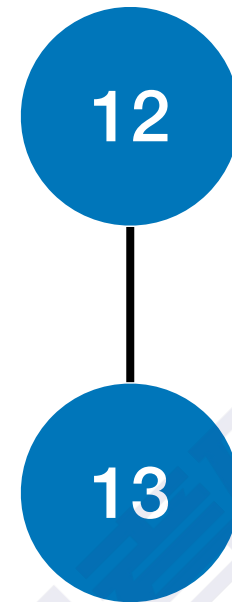
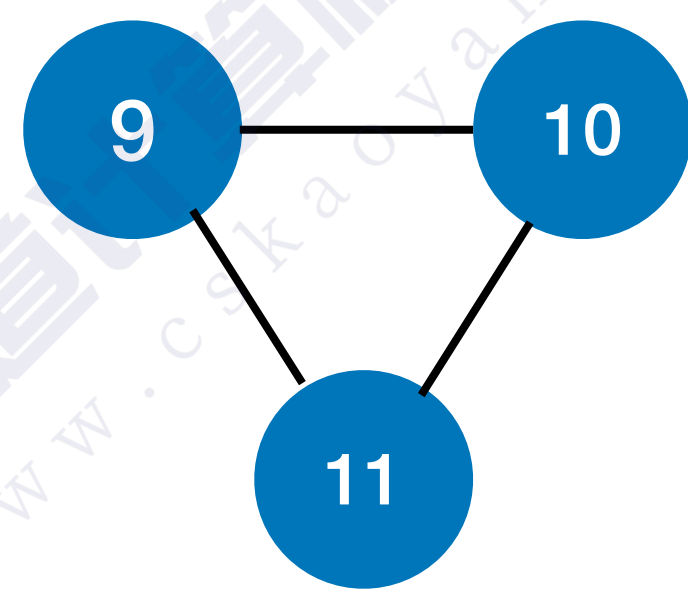


图的遍历与图的连通性

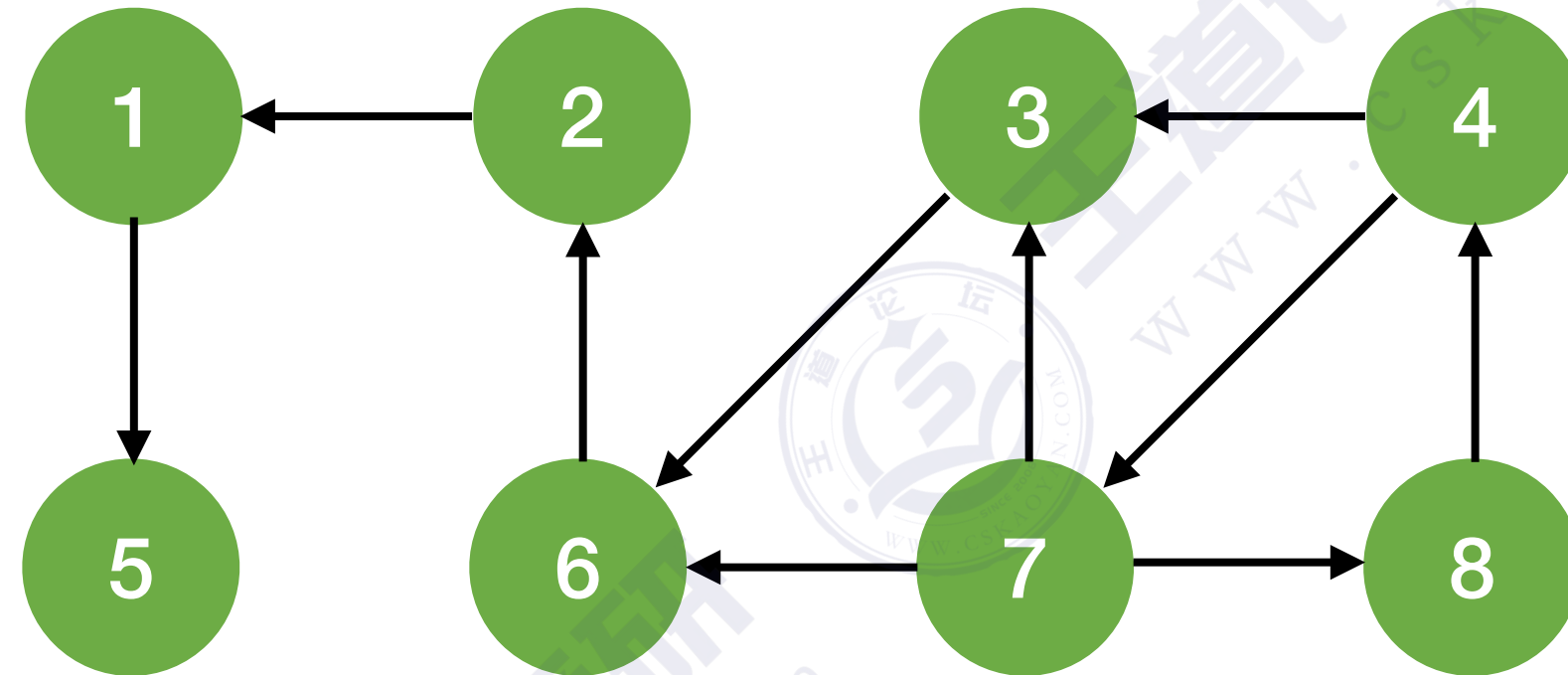


对**无向图**进行BFS/DFS遍历
调用BFS/DFS函数的次数=连通分量数

对于**连通图**，只需调用**1次** BFS/DFS



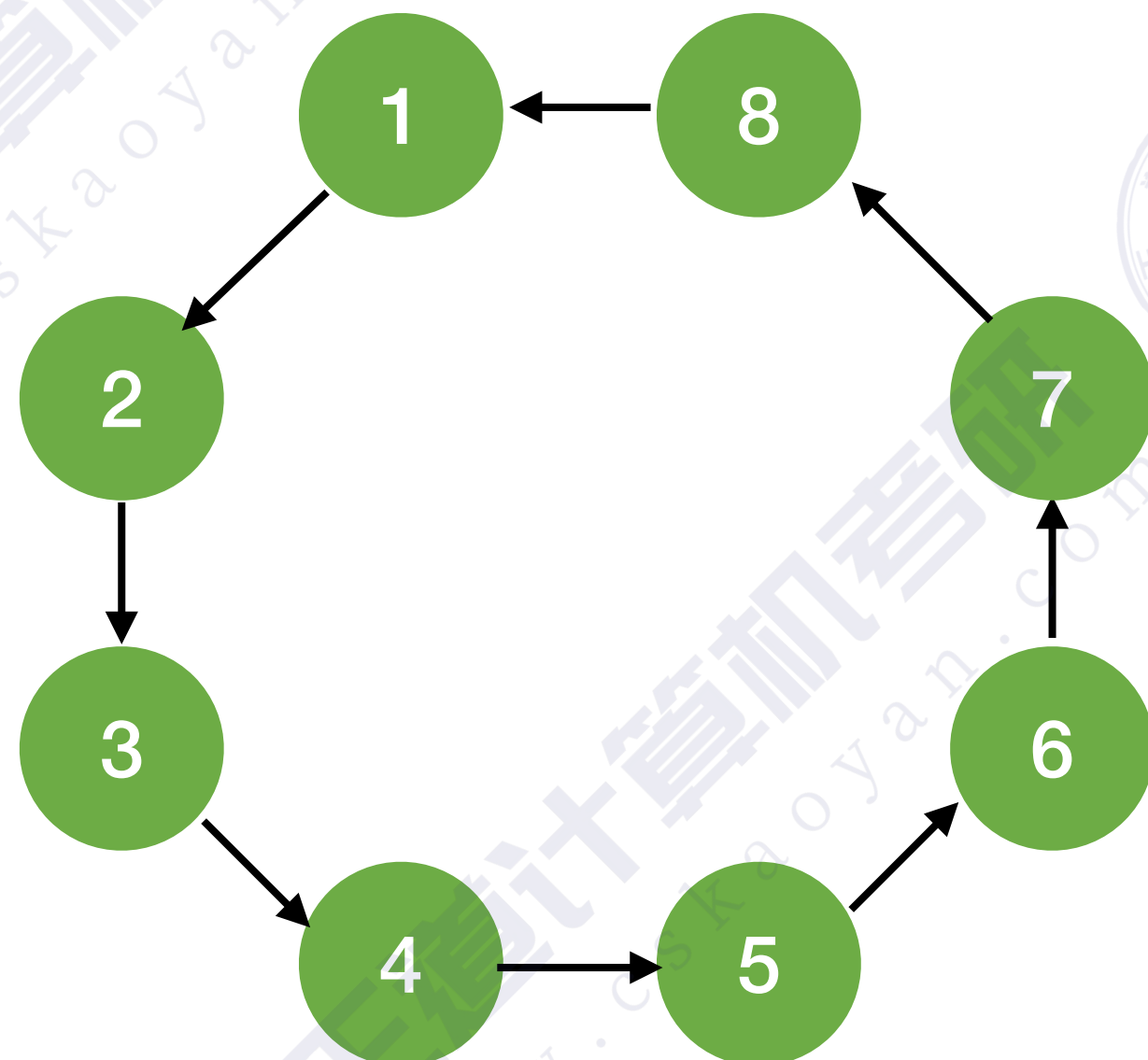
图的遍历与图的连通性



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



对于**强连通图**，从任一结点出发都只需调用1次 BFS/DFS

知识回顾与重要考点

图的DFS

算法要点

递归地深入探索未被访问过的邻接点（类似于树的先根遍历的实现）

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

复杂度分析

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： $O(|V|^2)$

邻接表： $O(|V|+|E|)$

深度优先生成树

由深度优先遍历确定的树

邻接表存储的图表示方式不唯一，深度优先遍历序列、生成树也不唯一

深度优先遍历非连通图可得深度优先生成森林

图的遍历和图的连通性

无向图

DFS/BFS 函数调用次数 = 连通分量数

若从起始顶点到其他顶点都有路径，则只需调用 1 次 DFS/BFS 函数

有向图

对于强连通图，从任一顶点出发都只需调用 1 次 DFS/BFS 函数