7.3 图的遍历

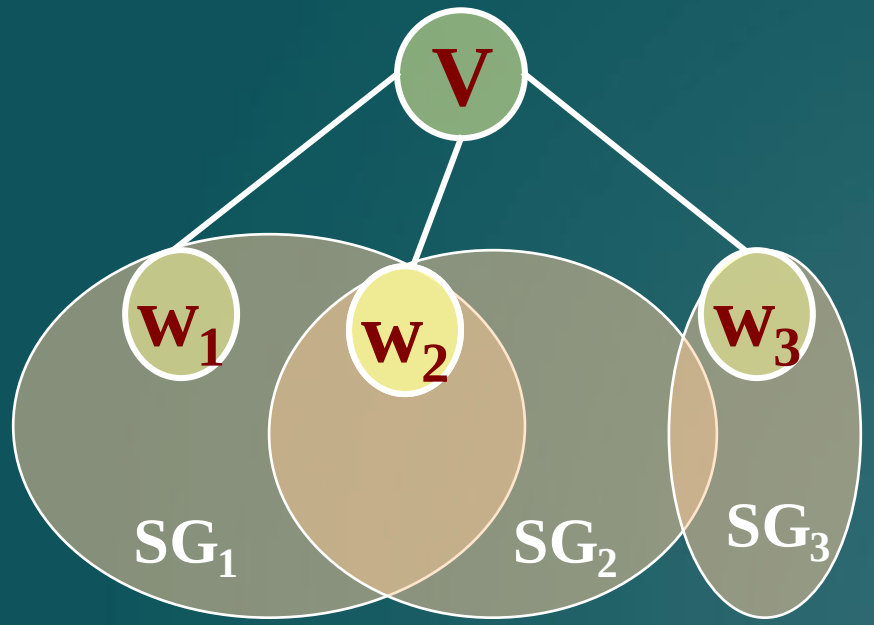
何谓图的遍历？

从图中某个顶点出发游历图，访遍图中其余顶点，并且使图中的每个顶点仅被访问一次的过程。

- **深度优先搜索（DFS）**
- **广度优先搜索（BFS）**

7.3.1 深度优先搜索（Depth_First Search）

从图中某个顶点V_0出发，访问此顶点，然后**依次从V_0的各个未被访问的邻接点出发深度优先搜索遍历图**，直至图中所有和V_0有路径相通的顶点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直到图中所有顶点都被访问到为止。



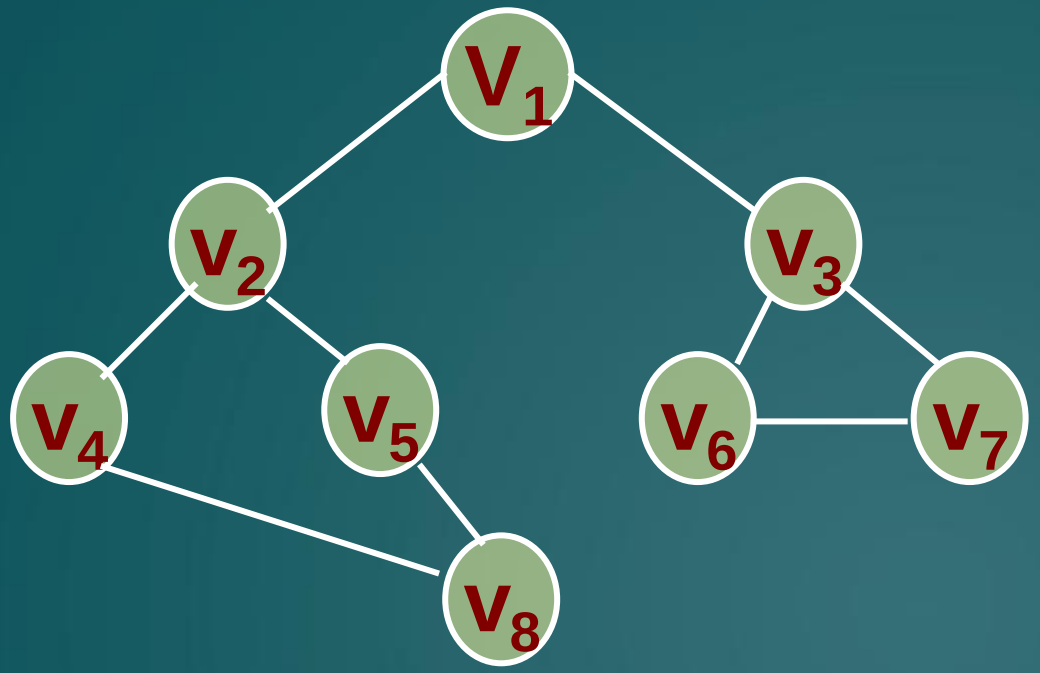
W_1、W_2和W_3 均为 V 的邻接点，SG_1、SG_2和 SG_3 分别为含顶点W_1、W_2和W_3的子图。

访问顶点 V ：

for (W1、W2、W3)

若该邻接点W未被访问，

则从它出发进行深度优先搜索遍历。



对上图，假设从v1开始进行深度优先遍历，则遍历顺序为：

$v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_8 \rightarrow v_5 \rightarrow v_3 \rightarrow v_6 \rightarrow v_7$

从上页的图解可见:

1. 深度优先搜索遍历连通图的过程类似于树的先根遍历；

2. 如何判别V的邻接点是否被访问？

解决的办法是：为每个顶点W设立一个 “访问标志 visited[w]”。其初值为 ‘false‘ ，一旦某个顶点被访问，则其相应的分量置为 ‘true‘ 。

```
void DFS(Graph G, int v) {
   // 从顶点v出发，深度优先遍历图G
    visited[v] = TRUE;   VisitFunc(v);//访问第v个顶点
    for(w=FirstAdjVex(G, v);  w>=0; w=NextAdjVex(G,v,w))
        if (!visited[w])  DFS(G, w);
      // 对v的尚未访问的邻接顶点w，递归调用DFS
} // DFS
```

算法 7.5

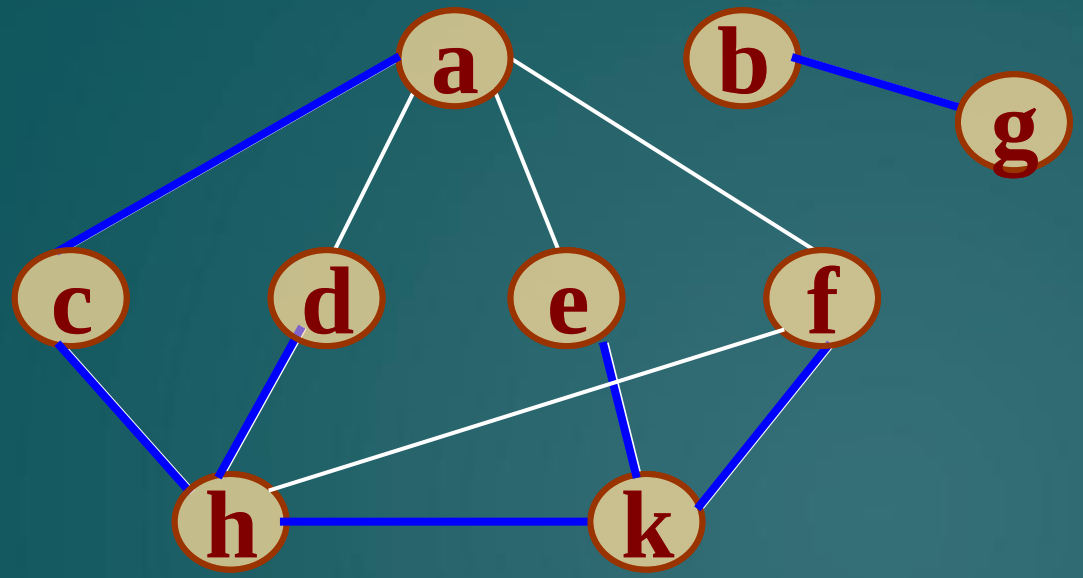
一般图的深度优先搜索遍历

首先将图中每个顶点的访问标志设为 FALSE，之后搜索图中每个顶点，如果未被访问，则以该顶点为起始点，进行深度优先搜索遍历，否则继续检查下一顶点。

```
void DFSTraverse(Graph G， Status (*Visit)(int v)) {
  // 对图 G 作深度优先遍历。
  VisitFunc = Visit;
  for (v=0; v<G.vexnum; ++v)
    visited[v] = FALSE; // 访问标志数组初始化
  for (v=0; v<G.vexnum; ++v)
    if (!visited[v])  DFS(G, v);
          // 对尚未访问的顶点调用DFS
}
```

算法 7.4

例如:



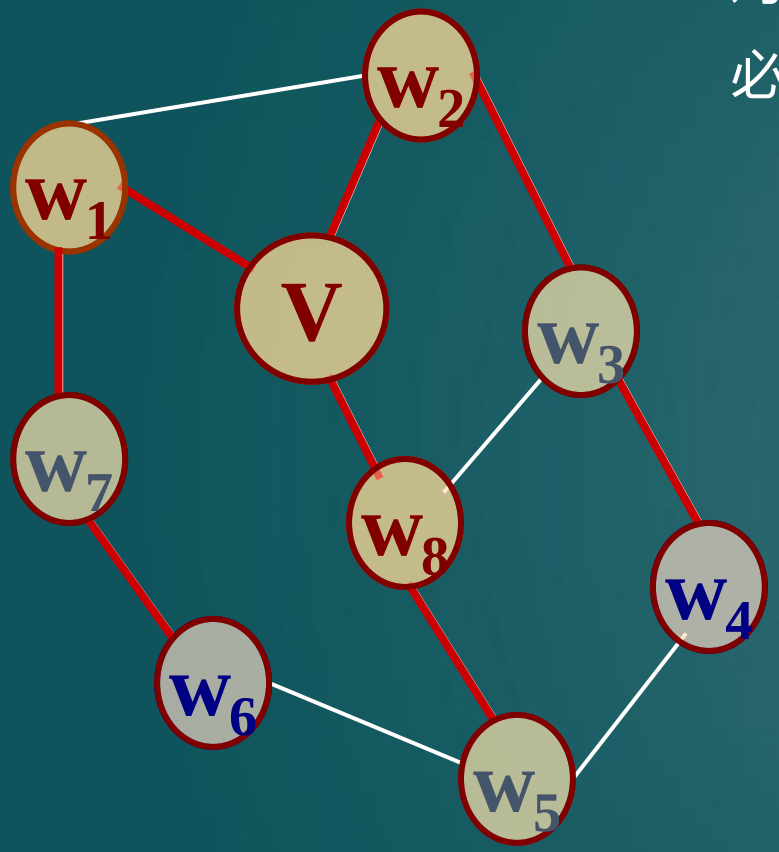
	0	1	2	3	4	5	6	7	8
访问标志:	T	T	T	T	T	T	T	T	T

访问次序:	a	c	h	d	k	f	e	b	g

7.3.2 广度优先搜索（Breadth_First Search）

从图中的某个顶点V_0出发，并在访问此顶点之后依次访问V_0的所有**未被访问**过的邻接点，之后**按这些顶点被访问的先后次序依次访问它们的邻接点**，直至图中所有和V_0有路径相通的顶点都被访问到。

若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。

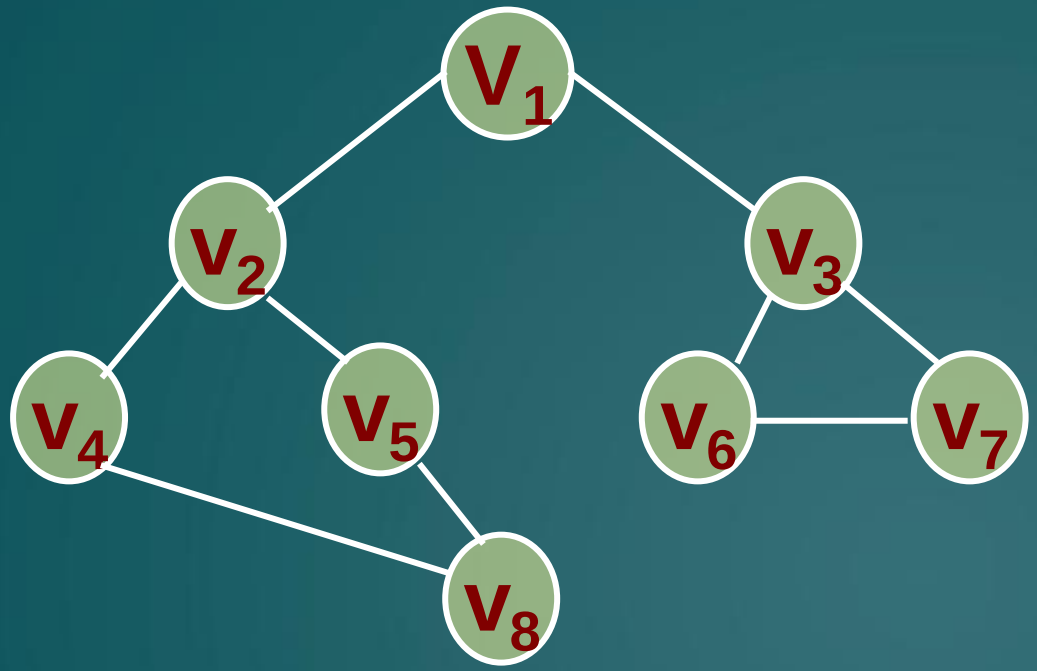


对连通图，从起始点V到其余各顶点必定存在路径。

其中，$V \rightarrow W_1$, $V \rightarrow W_2$, $V \rightarrow W_8$ 的路径长度为1；

$V \rightarrow W_7$, $V \rightarrow W_3$, $V \rightarrow W_5$ 的路径长度为2；

$V \rightarrow W_6$, $V \rightarrow W_4$ 的路径长度为3。



对上图，假设从v1开始进行广度优先遍历，则遍历顺序为：

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8$

```
void BFSTraverse(Graph G,  Status (*Visit)(int v)){
 //按广度优先非递归遍历图G，使用辅助队列Q和
//访问标志数组visited
  for (v=0; v<G.vexnum; ++v)
      visited[v] = FALSE;  //初始化访问标志
  InitQueue(Q);        // 置空的辅助队列Q
  for ( v=0;  v<G.vexnum;  ++v )
     if ( !visited[v]) {          // v 尚未访问
           …
    }
 } // BFSTraverse
```

算法 7.6

```
visited[v] = TRUE; Visit(v);    // 访问v
EnQueue(Q, v);                  // v入队列
while (!QueueEmpty(Q)) {
   DeQueue(Q, u);
                        // 队头元素出队并置为u
   for(w=FirstAdjVex(G, u); w>=0;
                        w=NextAdjVex(G,u,w))
      if ( ! visited[w]) {  //W为u的尚未访问的邻接顶点
         visited[w]=TRUE; Visit(w);
         EnQueue(Q, w); // 访问的顶点w入队列
      } // if
} // while
```