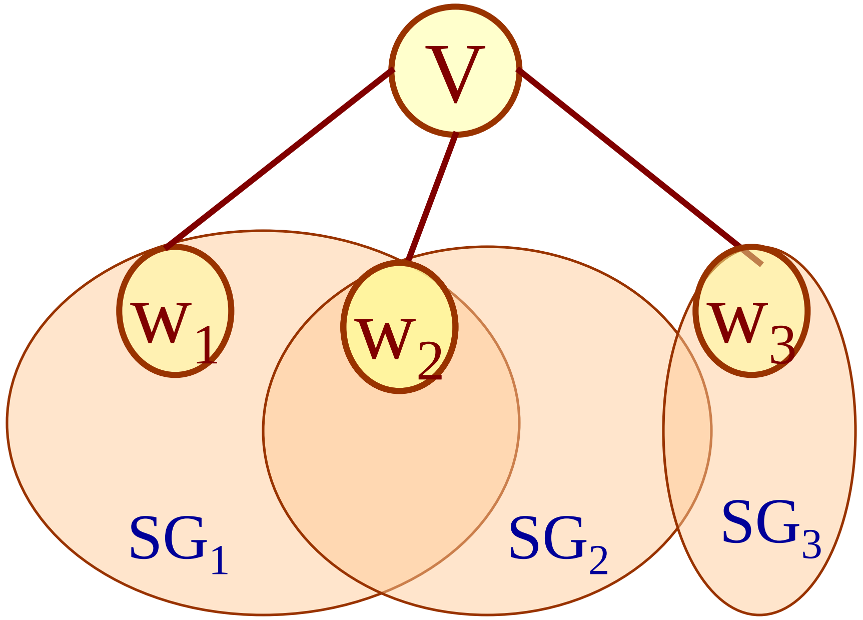
图的遍历

图的遍历

- *Depth-first traversal(深度优先遍历)* of a graph is roughly **analogous to preorder traversal of an ordered tree**. Suppose that the traversal has just visited a vertex *v*, and let *w*1*;w*2……*wk* be the vertices adjacent to *v*. Then we shall next visit *w*1 and keep *w*2……*wk* waiting. After visiting *w*1 , we traverse all the vertices to which it is adjacent before returning to traverse *w*2*;* ……*;wk* .
- *Breadth-first traversal(广度或宽度优先遍历)* of a graph is roughly analogous to level-by-level traversal*(层序遍历)* of an ordered tree. If the traversal has just visited a vertex *v*, then it next visits *all* the vertices adjacent to *v*, putting the vertices adjacent to these in a waiting list to be traversed after all vertices adjacent to *v* have been visited.

连通图的深度优先搜索遍历

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。



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。

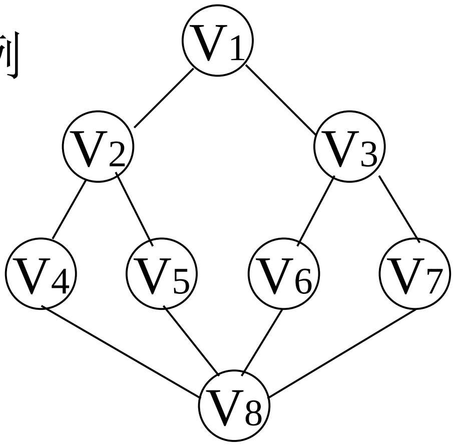
访问顶点 V：

for (W_1、W_2、W_3)

若该邻接点W未被访问，

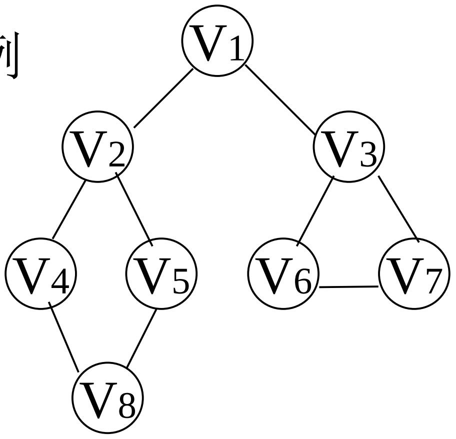
则从它出发进行深度优先搜索遍历。

例



深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V6 ⇒V3 ⇒V7

例



深度遍历：V1⇒ V2 ⇒V4 ⇒ V8 ⇒V5 ⇒V3 ⇒V6 ⇒V7

连通图的深度优先搜索遍历

1. 从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：为每个顶点设立一个 “访问标志visited[0..n-1]”。

连通图的深度优先搜索遍历

❖ Depth-First算法

```
template <int max_size>
void Digraph<max_size> :: traverse(Vertex &v, bool
        visited[], void (*visit)(Vertex &)) const
/* Pre: v is a vertex of the Digraph .
Post: The depth-first traversal, using function*visit ,
        has been completed for v and for all vertices that
        can be reached from v .
Uses: traverse recursively. */
```

连通图的深度优先搜索遍历

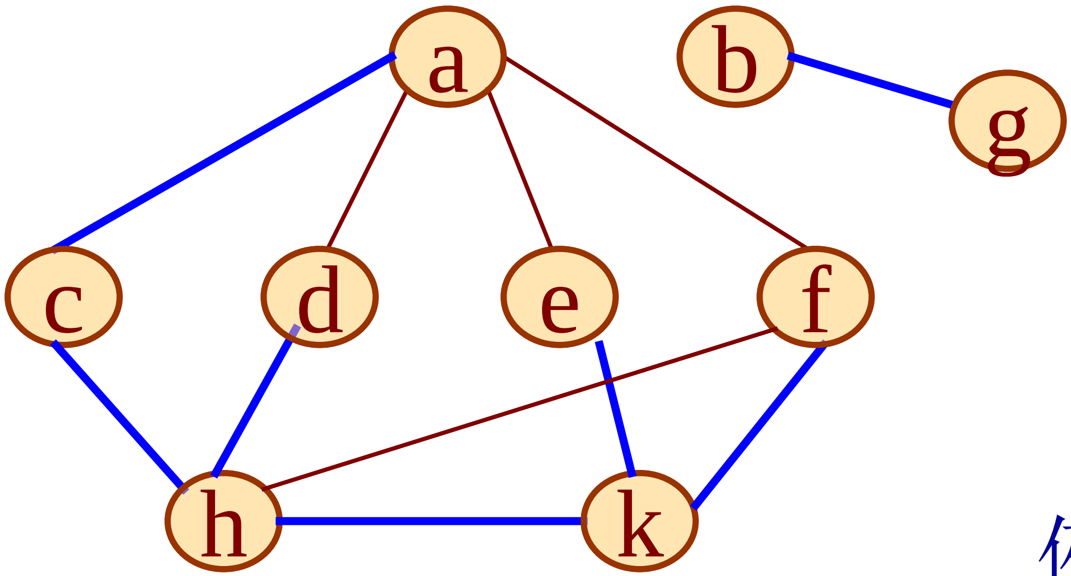
❖ Depth-First算法（续）

```
{
    Vertex w;
    visited[v] = true;
    (*visit)(v);
    for (all w adjacent to v)
        if (!visited[w])
            traverse(w, visited, visit);
}
```

养天地正氣 法古今完人

非连通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。



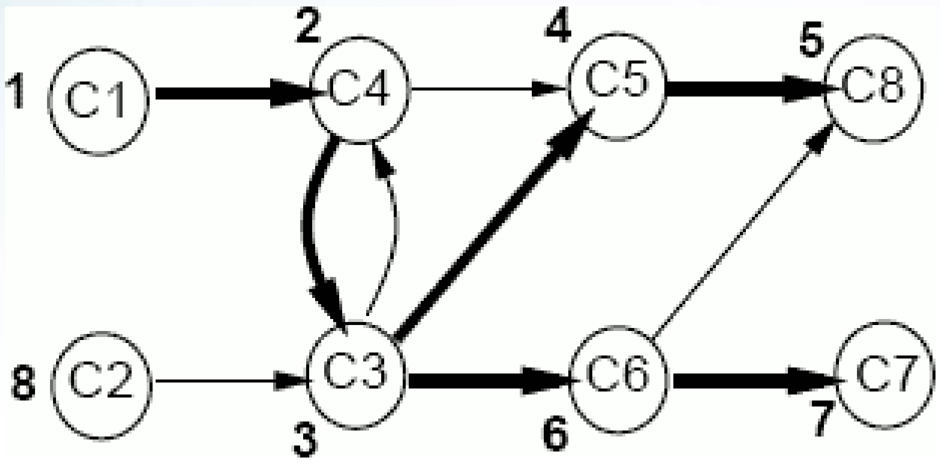
例如:

	0a	1b	2c	3d	4e	5f	6g	7h	8k
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

养天地正气 法古今完人

有向图的深度优先搜索遍历



非连通图的深度优先搜索遍历

❖ Depth-First 算法：

```
template <int max_size>
void Digraph<max_size> :: depth_first(void (*visit)(Vertex &))
        const
/* Post: The function *visit has been performed at each vertex of
        the Digraph in depth-first order.
Uses: Method traverse to produce the recursive depth-first order. */
{
bool visited[max_size];
Vertex v;
for (all v in G) visited[v] = false;
for (all v in G)
         if (!visited[v])
                  traverse(v, visited, visit);
}
```