树形穷举

树和图的遍历

Youtube频道： **hwdong**

博客： **hwdong-net**.github.io

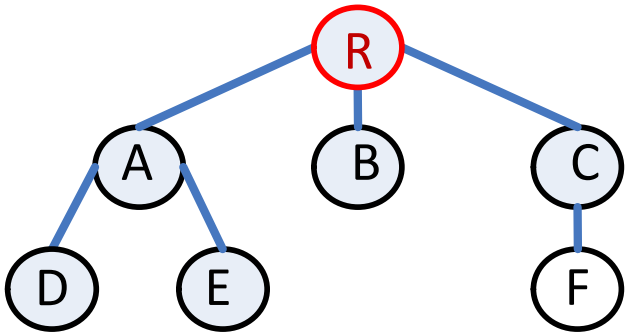
树的遍历

Youtube频道： **hwdong**

博客: **hwdong-net**.github.io

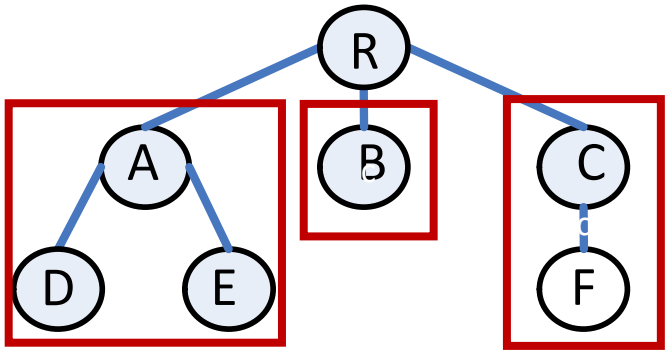
树

- 树是一种数据结构，它由节点和边组成，每个节点可以有零个或多个子节点，其中只有一个节点没有父节点，称为**根节点**。



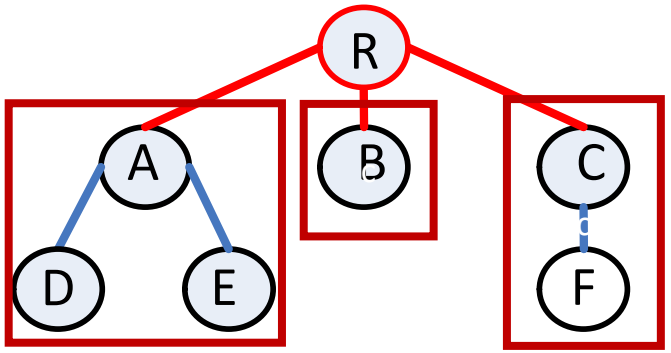
树

- 每个节点可以有零个或多子树。



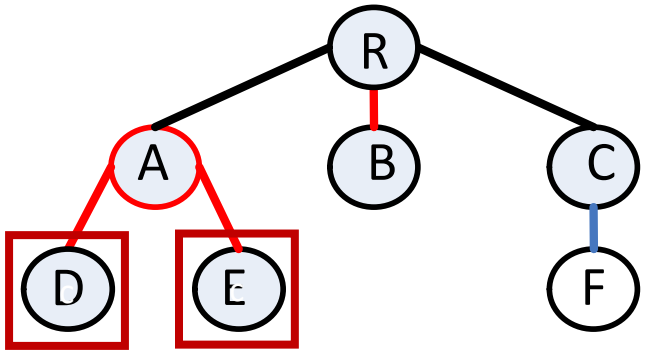
树

- 每个节点可以有零个或多子树。子树的根节点是这个节点的孩子



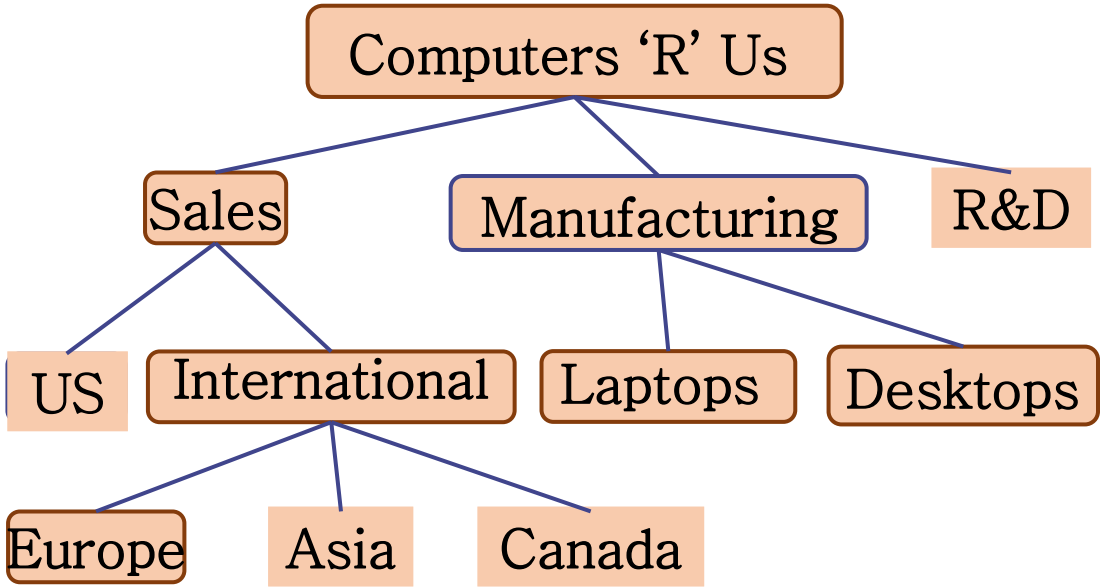
树

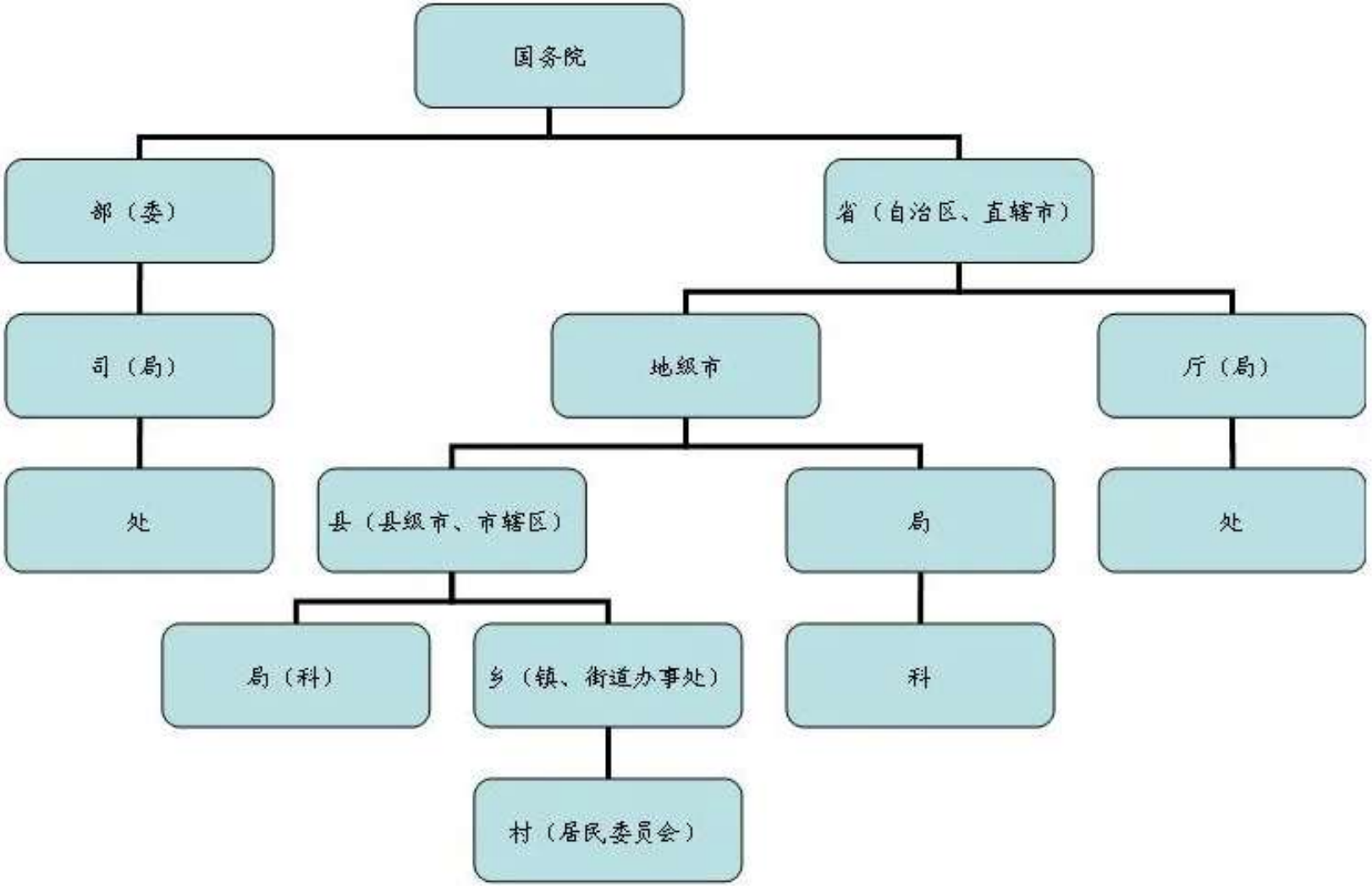
- 每个节点可以有零个或多子树。子树的根节点是这个节点的孩子



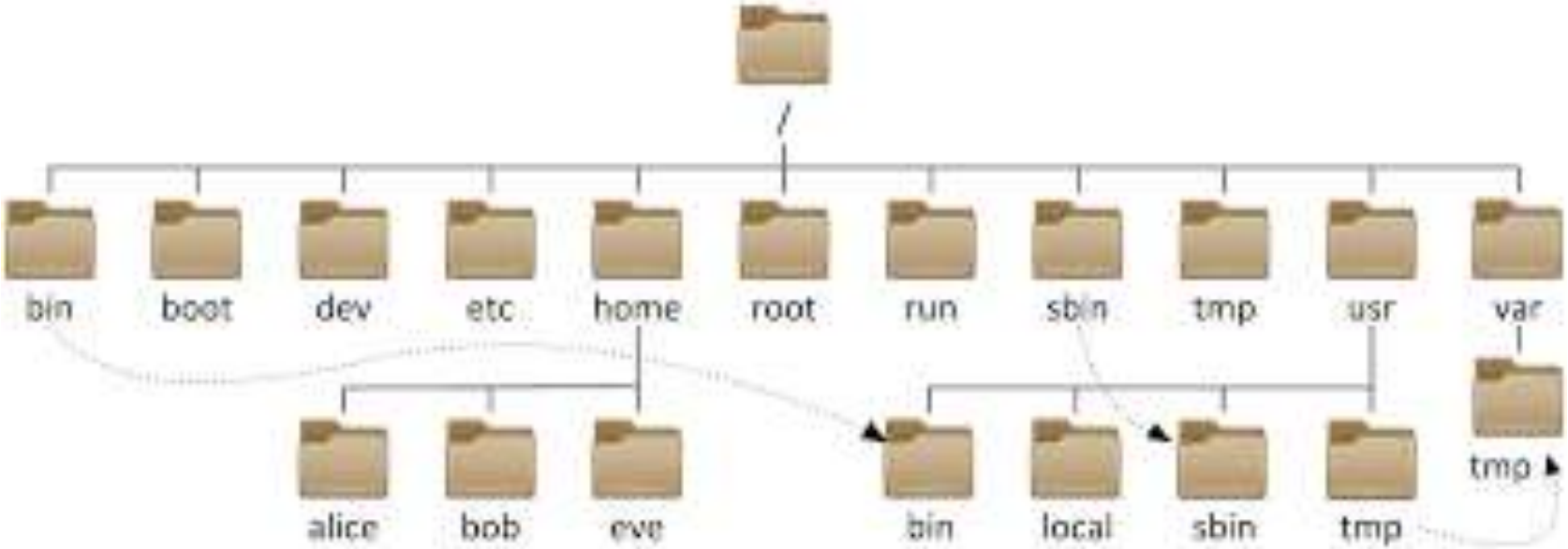
树的例子

- 组织结构

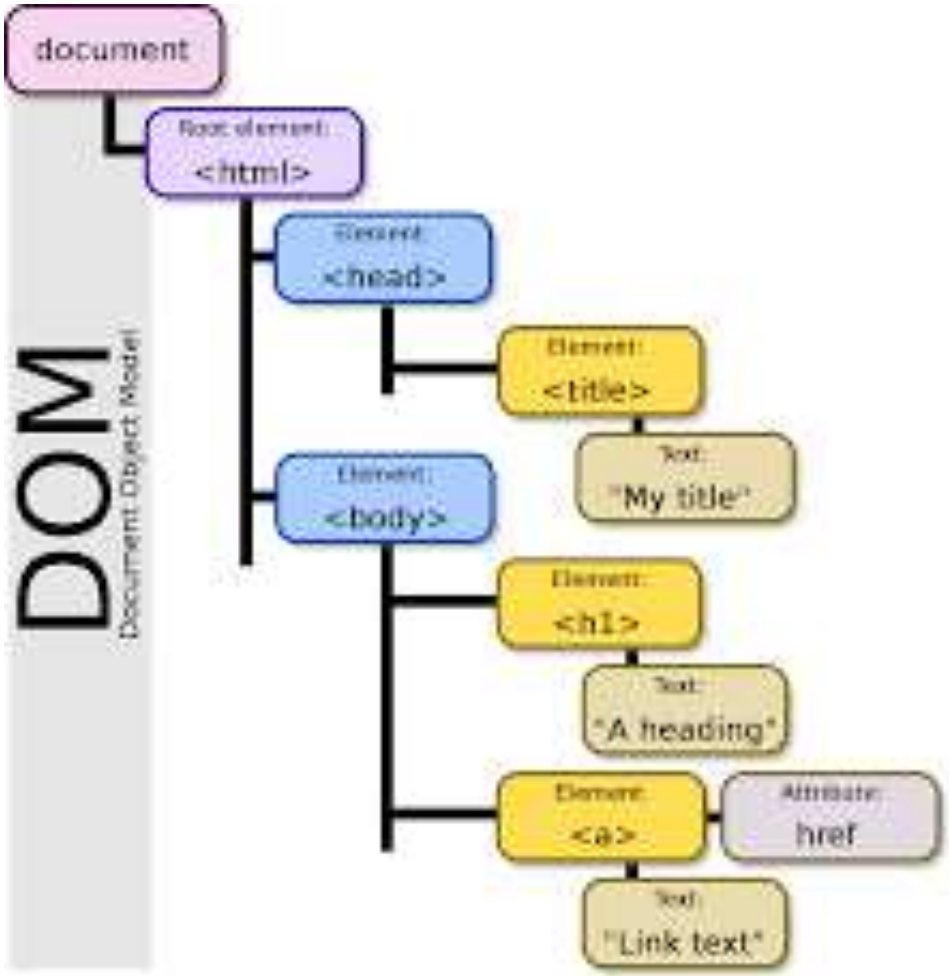

Computers ‘R’ Us
Sales
Manufacturing
R&D
US
International
Laptops
Desktops
Europe
Asia
Canada


国务院
部（委）
省（自治区、直辖市）
司（局）
地级市
厅（局）
处
县（县级市、市辖区）
局
处
局（科）
乡（镇、街道办事处）
科
村（居民委员会）

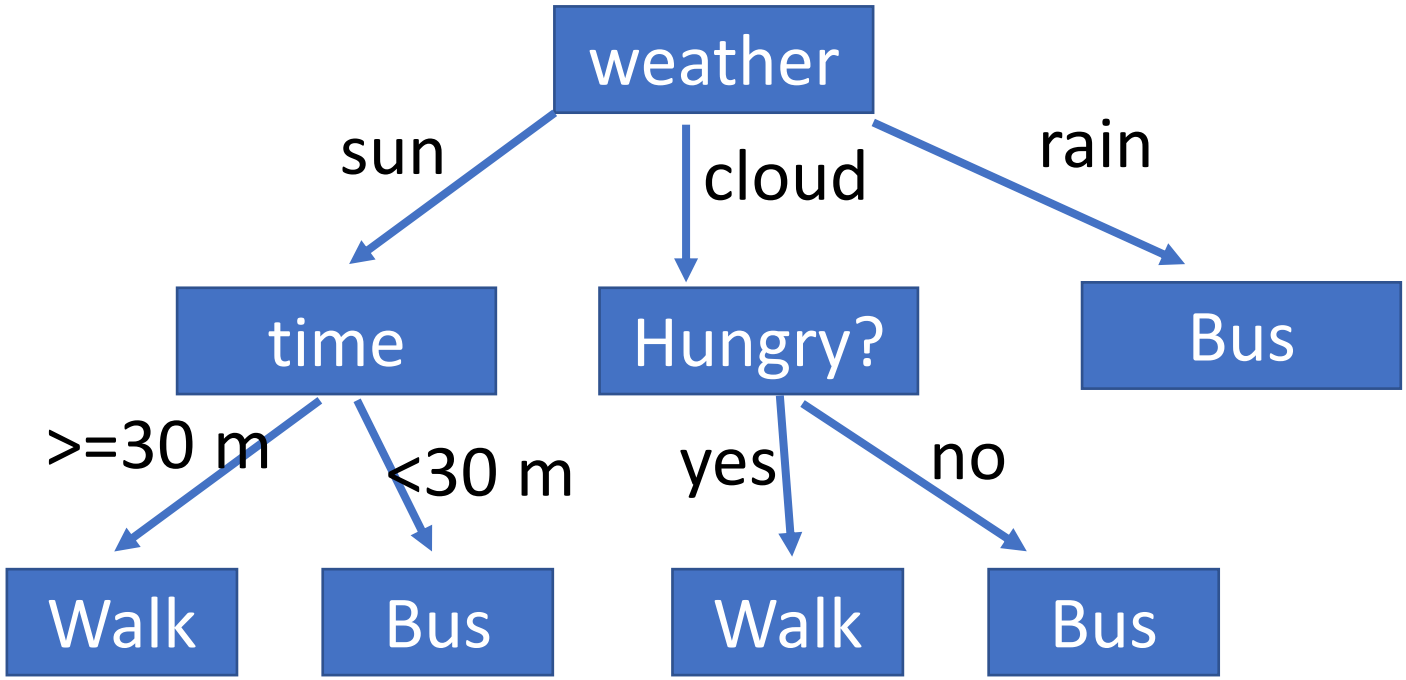
- 文件系统


/
bin
boot
dev
etc
home
root
run
sbin
tmp
usr
var
alice
bob
eve
bin
local
sbin
tmp
tmp

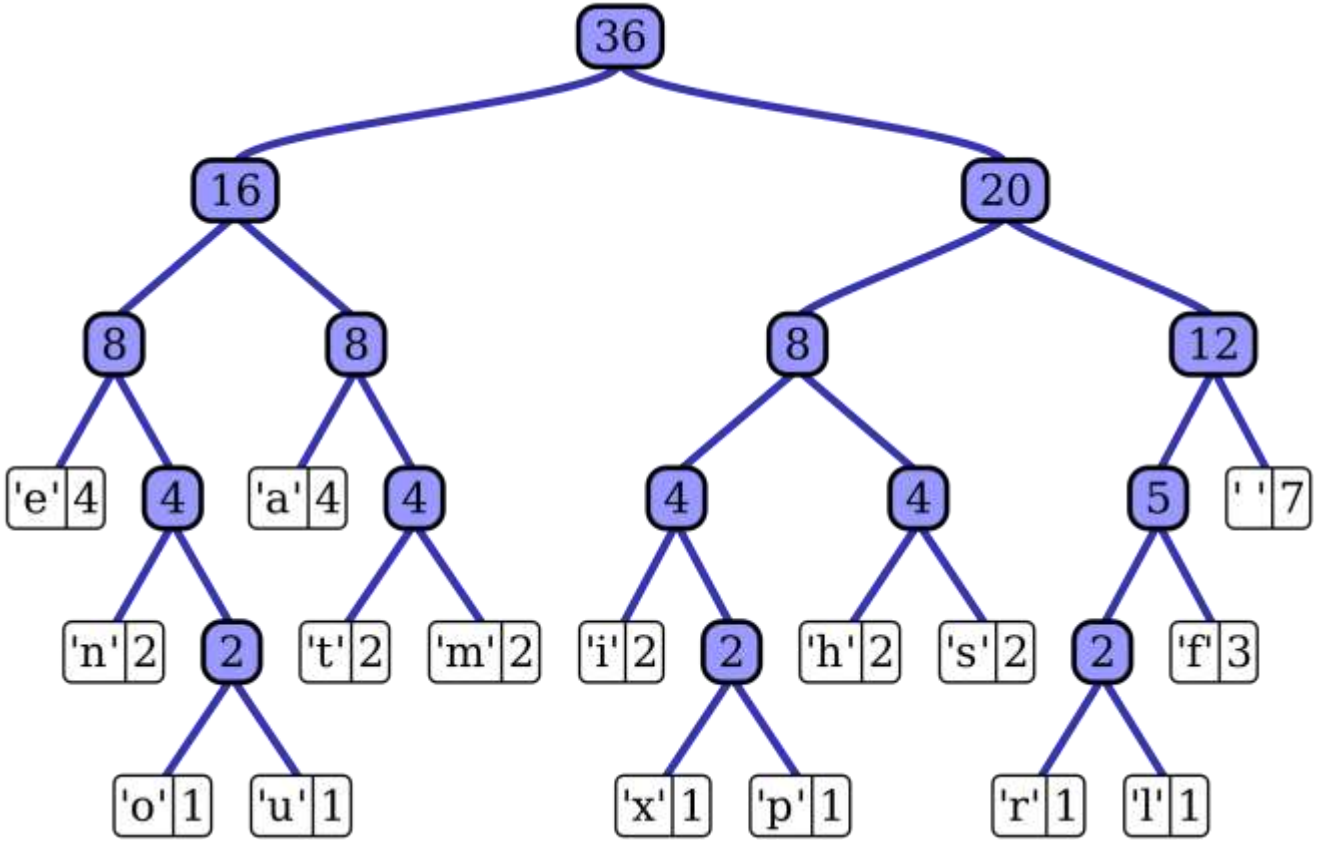
- HTML DOM ((Document Object Model))



- Decision tree（决策树）



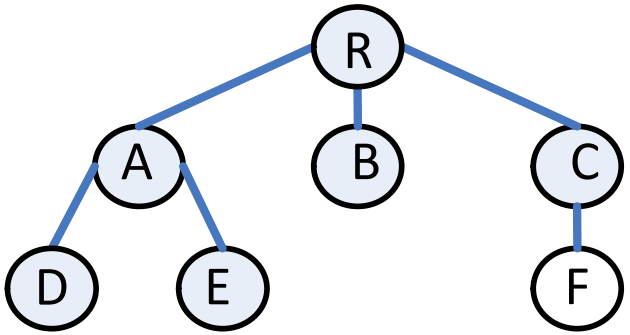
- Huffman coding（哈夫曼编码）



Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010

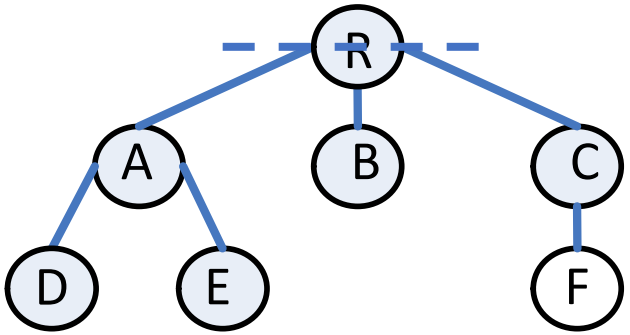
树的广度优先遍历

- 一层一层地访问每个节点。



树的广度优先遍历

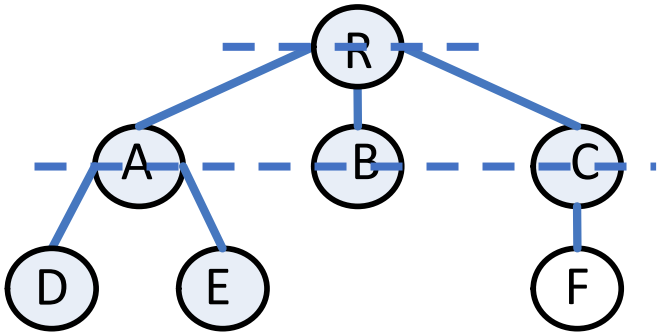
- 一层一层地访问每个节点。



R

树的广度优先遍历

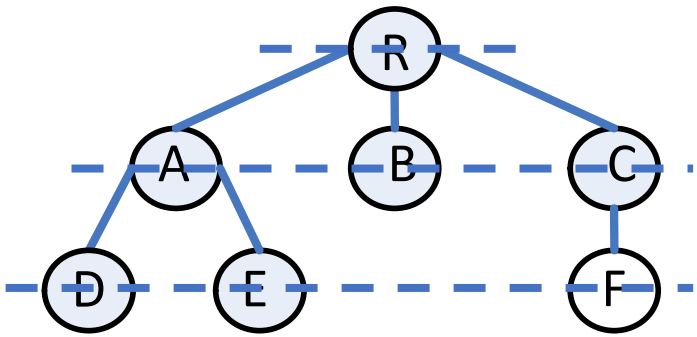
- 一层一层地访问每个节点。



R A B C

树的广度优先遍历

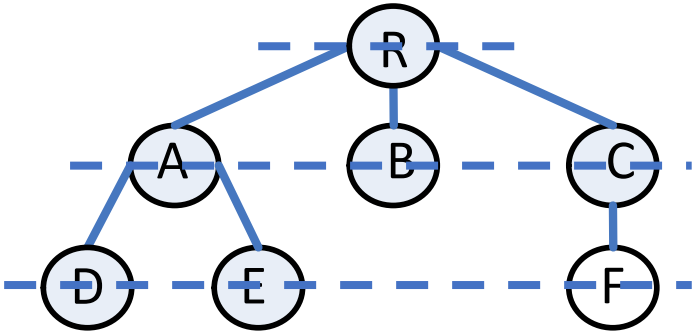
- 一层一层地访问每个节点。



R A B C D E F

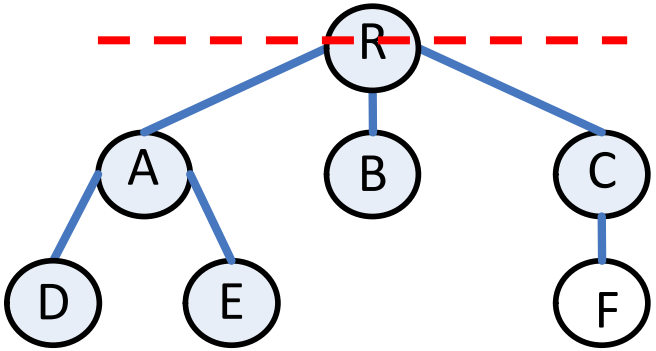
树的广度优先遍历

- 先访问的节点的孩子节点也先访问。A比C先访问，A的孩子D、E比C的孩子F先访问。



R A B C D E F

树的层次遍历

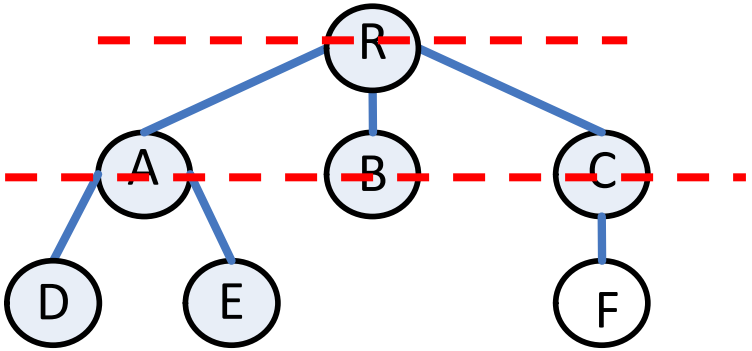


先进先出：

R

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历



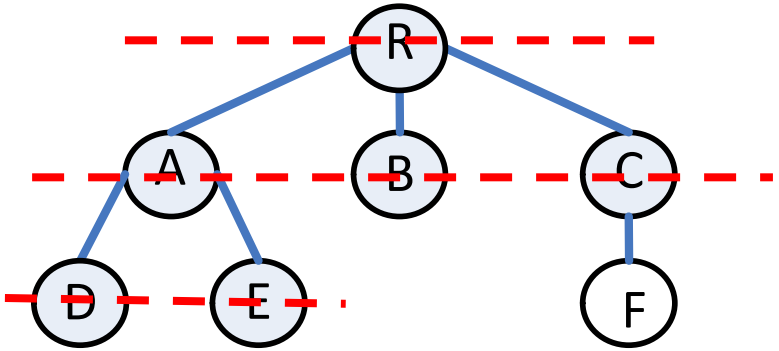
先进先出：

R

A B C

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历



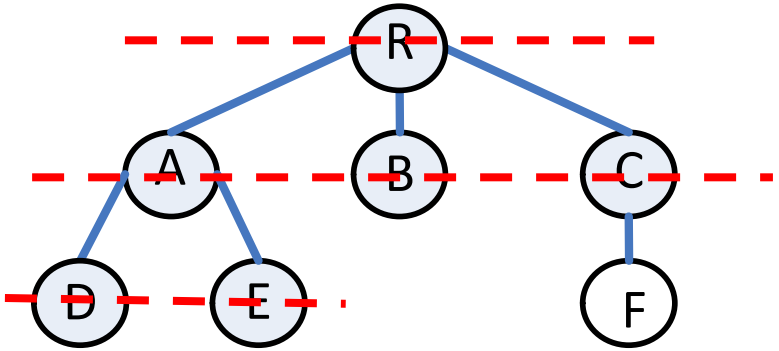
先进先出：

R A

B C D E

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历



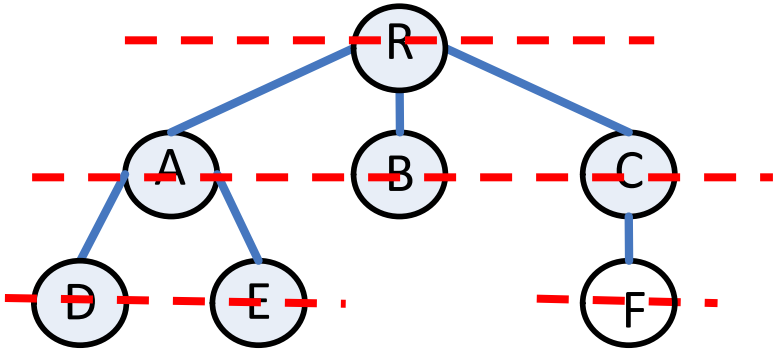
先进先出：

R A B

C D E

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历



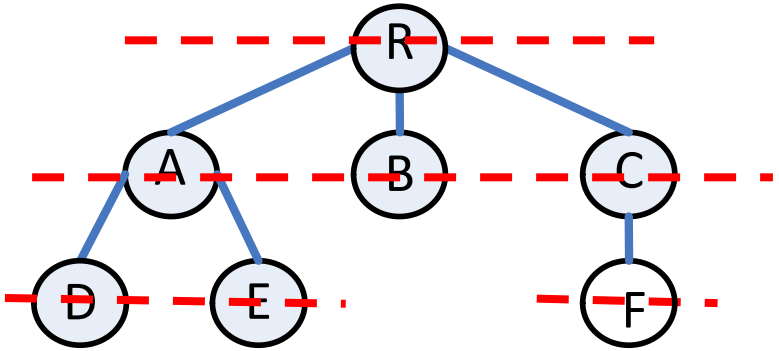
先进先出：

R A B C

D E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历

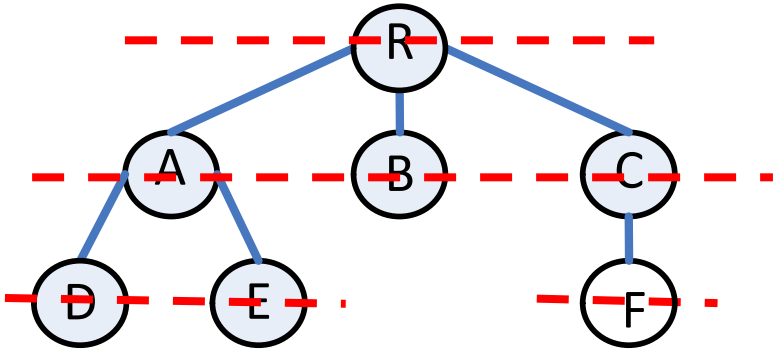


先进先出：

R A B C D　　E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历

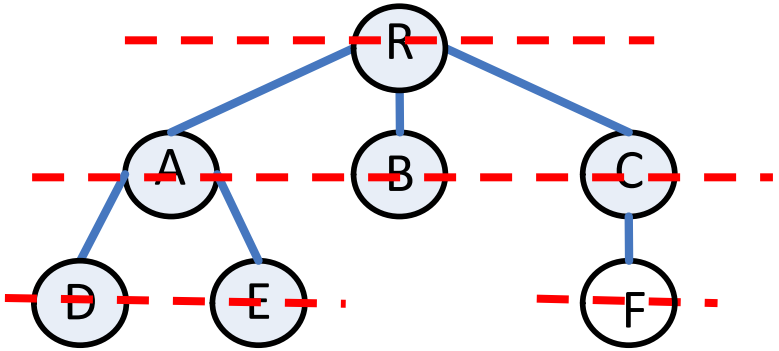


先进先出：

R A B C D E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的层次遍历



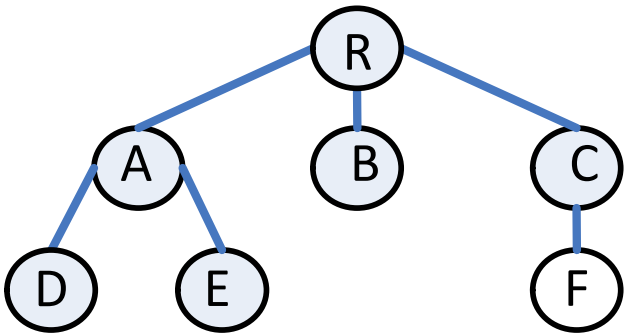
先进先出：

R A B C D E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

树的深度优先遍历

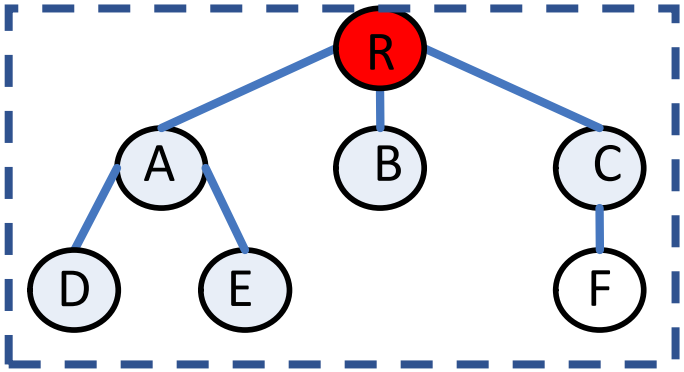
- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。




```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

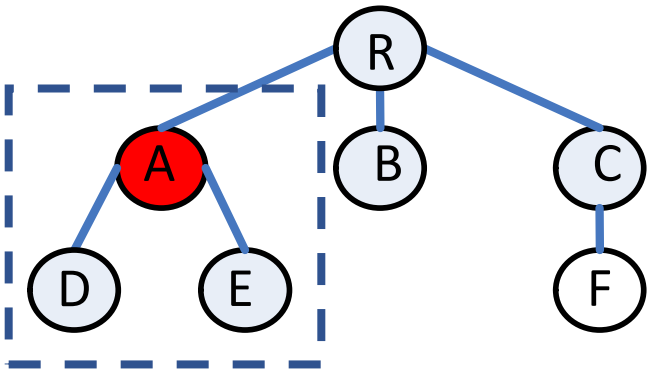



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

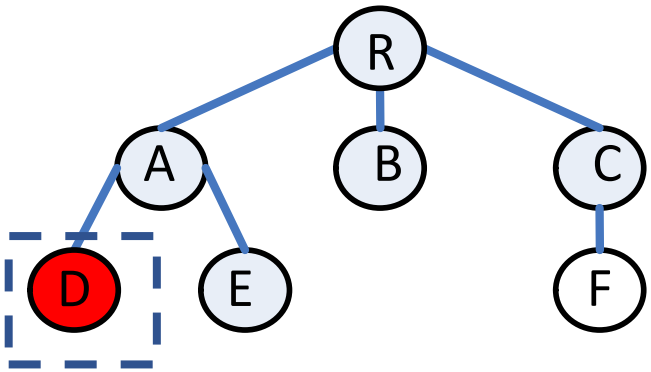



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

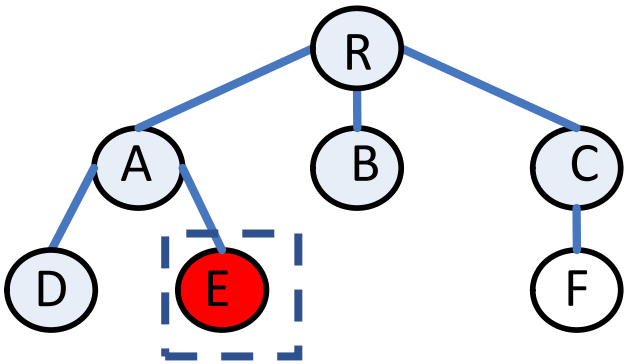



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

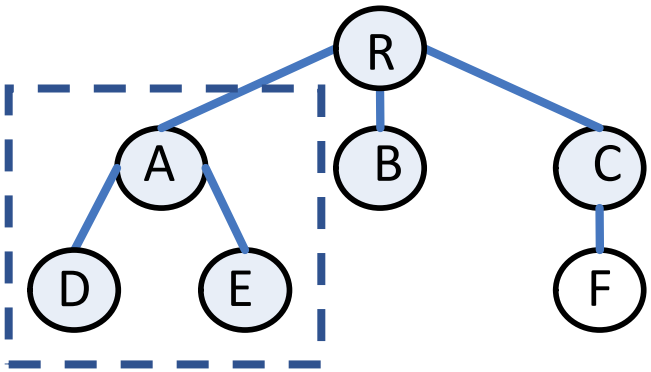



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

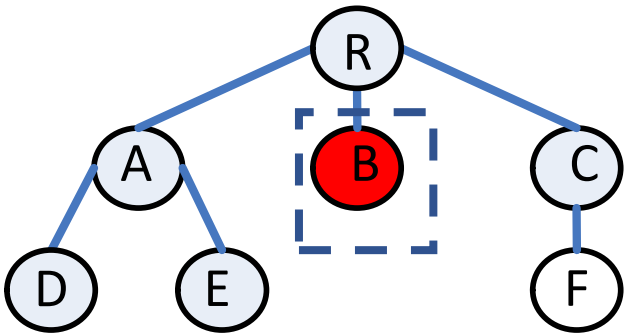



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

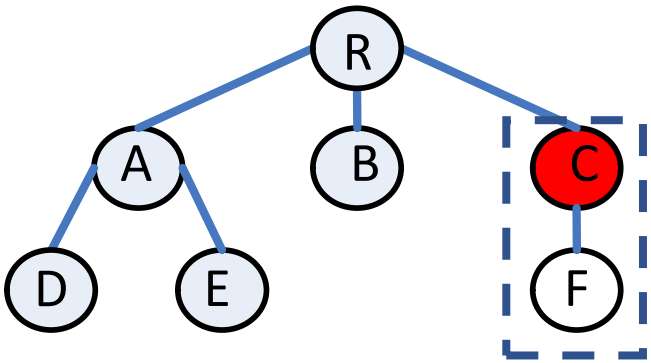



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E B

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

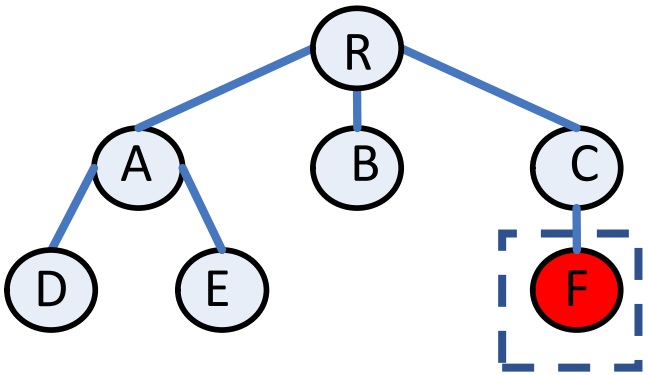



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E B C

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

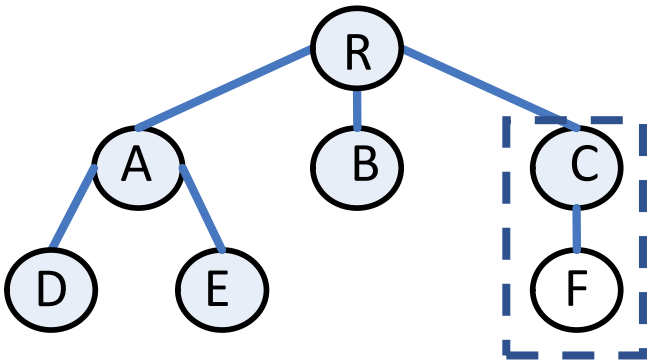



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E B C F

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

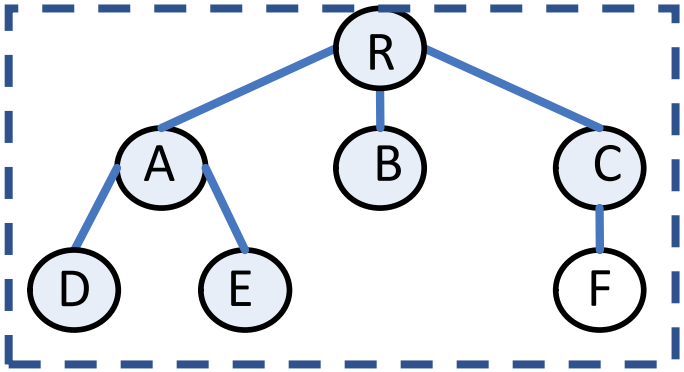



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E B C F

树的深度优先遍历

- 先序(根)深度优先遍历：先访问当前节点，再递归地遍历它的所有子树。

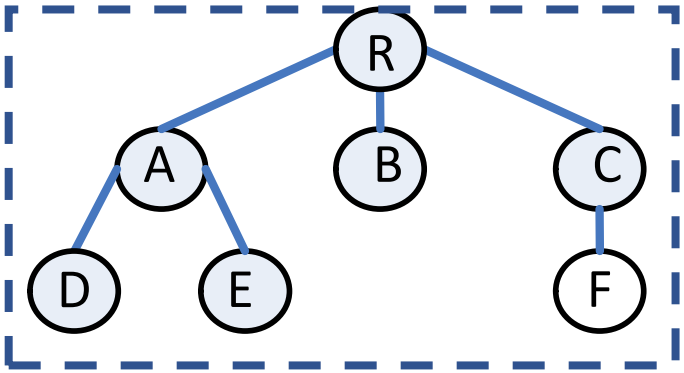



```
preorder(node)
  if node is not null
    visit(node)
    for each child in node.children
      preorder(child)
```

R A D E B C F

树的深度优先遍历

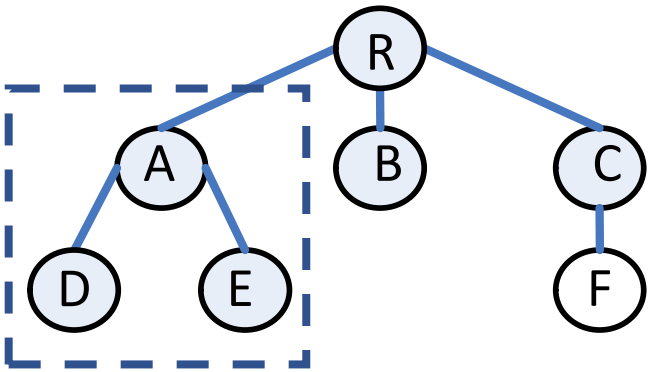
- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。




```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

树的深度优先遍历

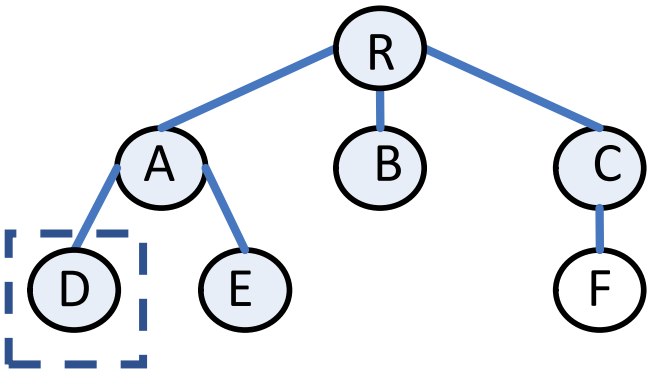
- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。




```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

树的深度优先遍历

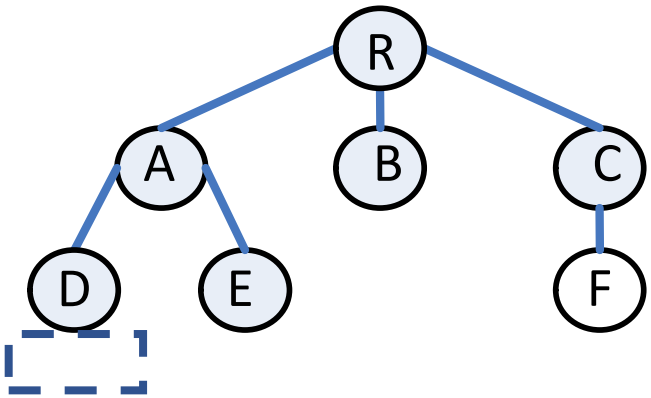
- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。




```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

树的深度优先遍历

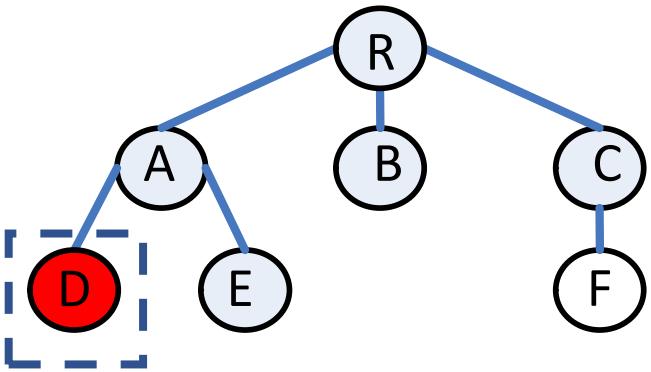
- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。




```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

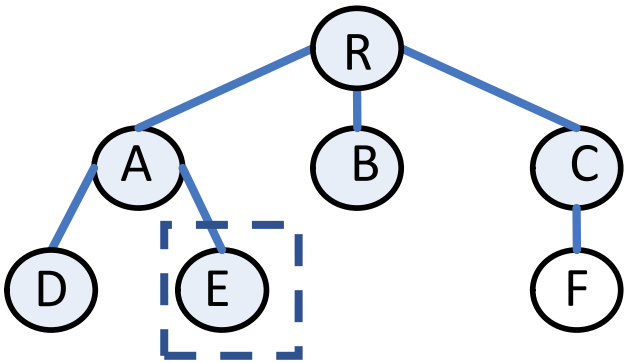



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

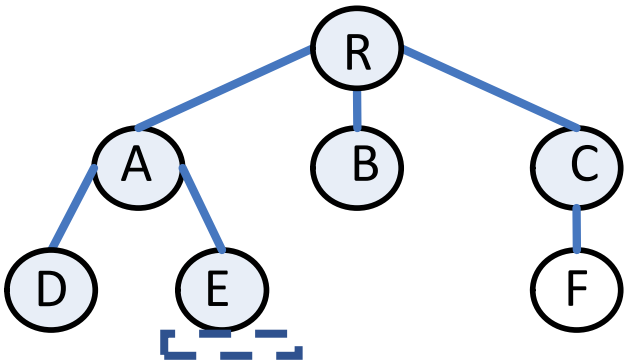



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

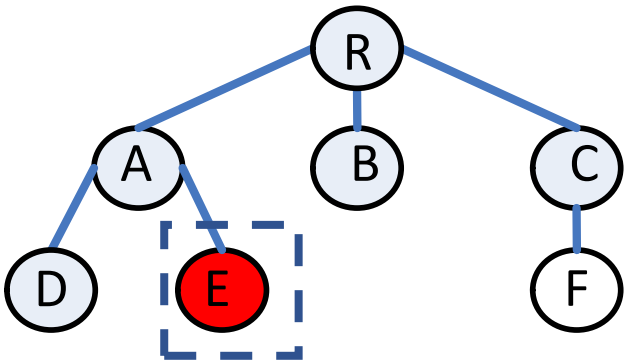



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

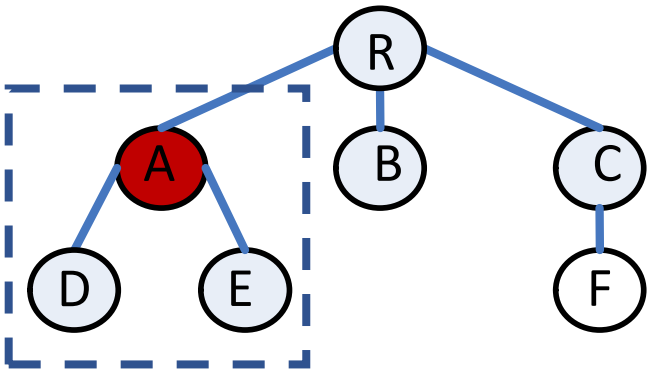



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

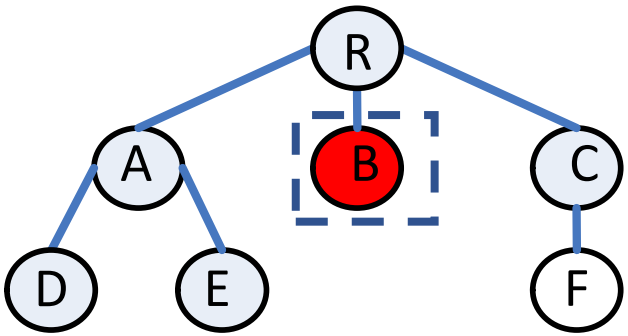



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E A

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

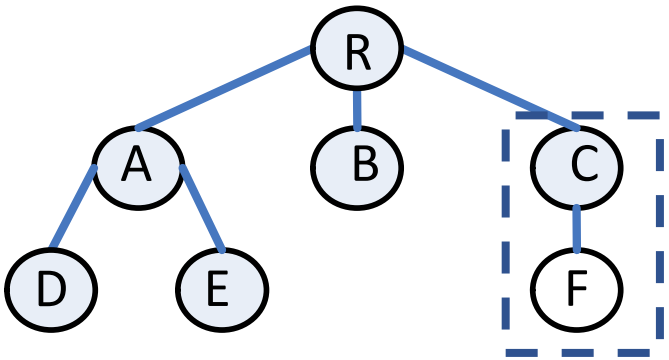



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E A B

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

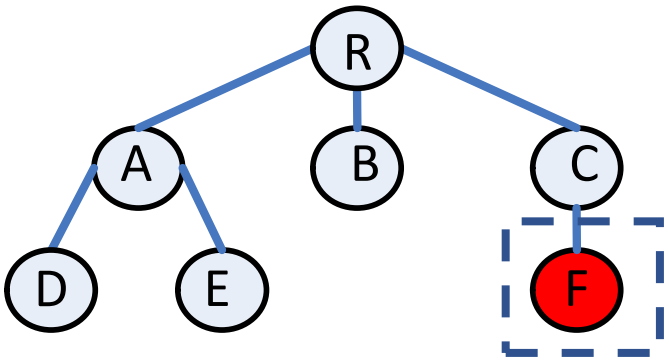



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E A B

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

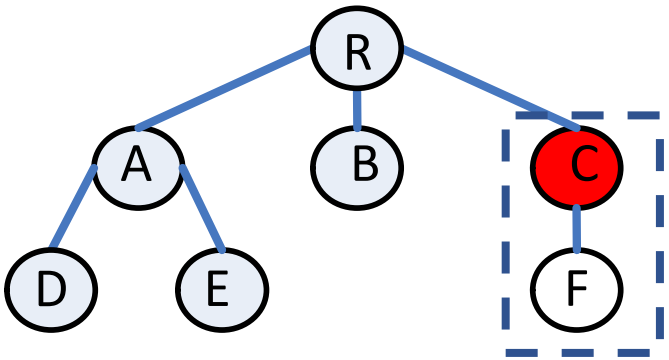



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E A B F

树的深度优先遍历

- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。

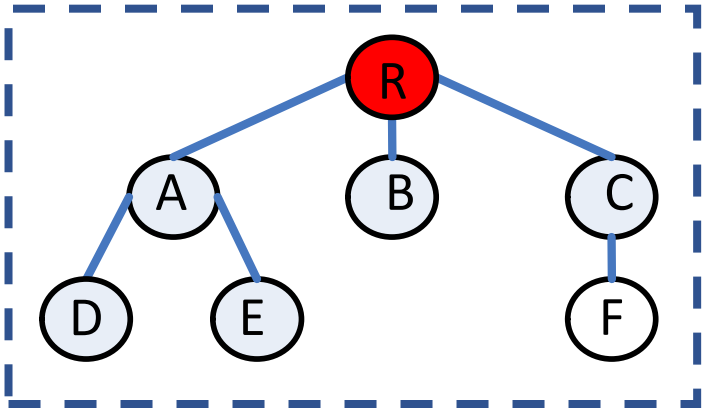



```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E A B F C

树的深度优先遍历

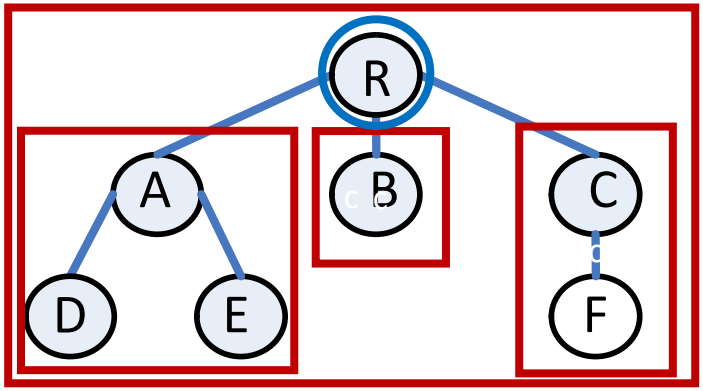
- 后序(根)深度优先遍历：先递归地遍历当前节点的所有子树，再访问当前节点。



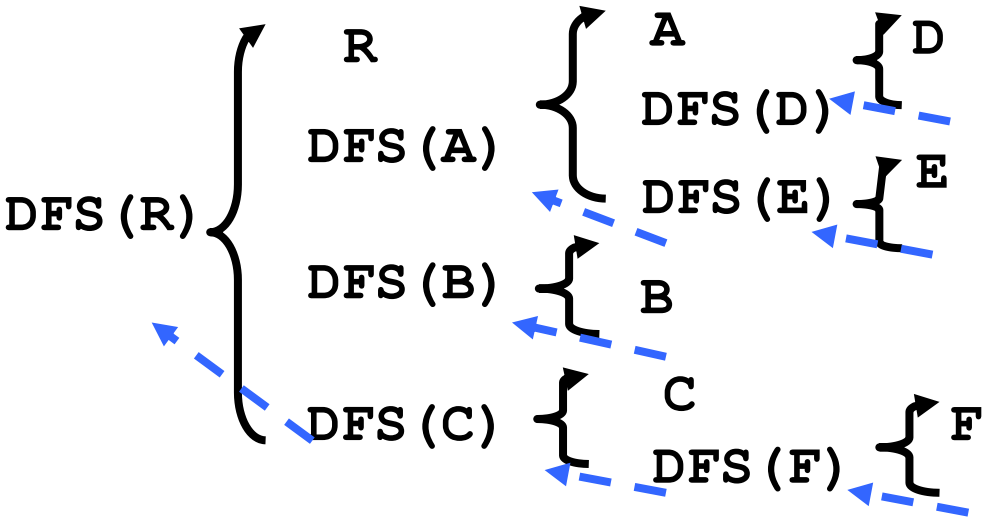

```
postorder(node)
  if node is not null
    for each child in node.children
      postorder(child)
    visit(node)
```

D E A B F C R

树的(先序)深度优先遍历




```
DFS(V){
    直接访问V;
    for(V的每个孩子W)
        DFS(W);
}
```



图

Youtube频道： **hwdong**

博客: **hwdong-net**.github.io

图的基本概念

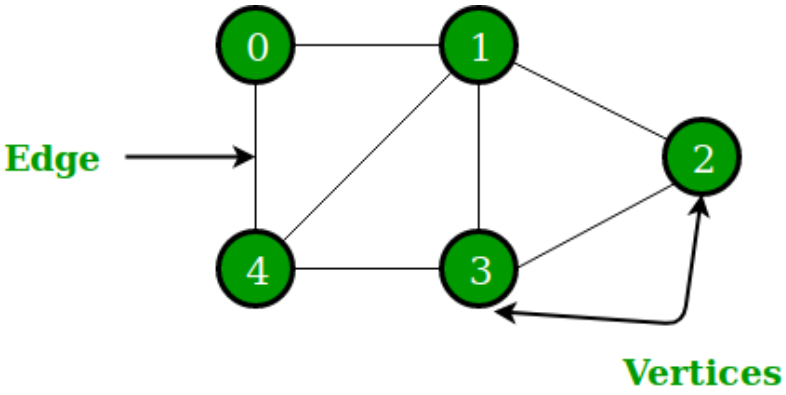
Graph

Youtube频道： **hwdong**

博客： **hwdong-net**.github.io

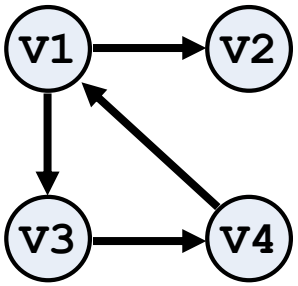
图

- 图G(V,E)是一个顶点集合V和边集合E。

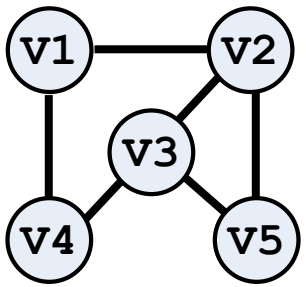


图

- 图G(V,E)是一个顶点集合V和边集合E。
- 无向图(undirected graph)中，边是一个无序顶点对:e=(u,v)
- 有向图(directed graph)中，边是一个有序顶点对:e=<u,v>



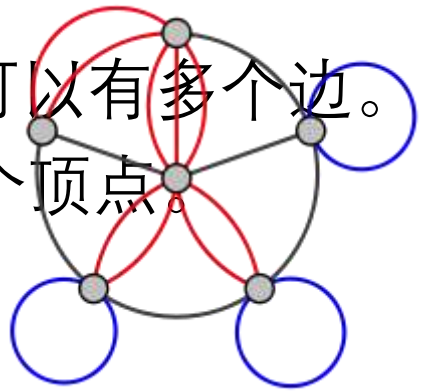
有向图



无向图

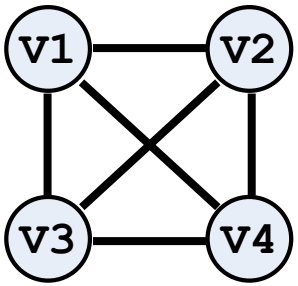
图

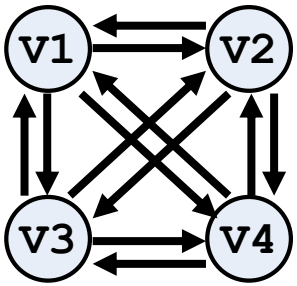
- 图G(V,E)是一个顶点集合V和边集合E。
- 无向图(undirected graph)中，边是一个无序顶点对:e=(u,v)
- 有向图(directed graph)中，边是一个有序顶点对:e=<u,v>
- 多边图(a multigraph): 2个顶点之间可以有多个边。
- self-loop：一个边的2个顶点是同一个顶点。
- 简单图：没有self-loop边的图。



完全图

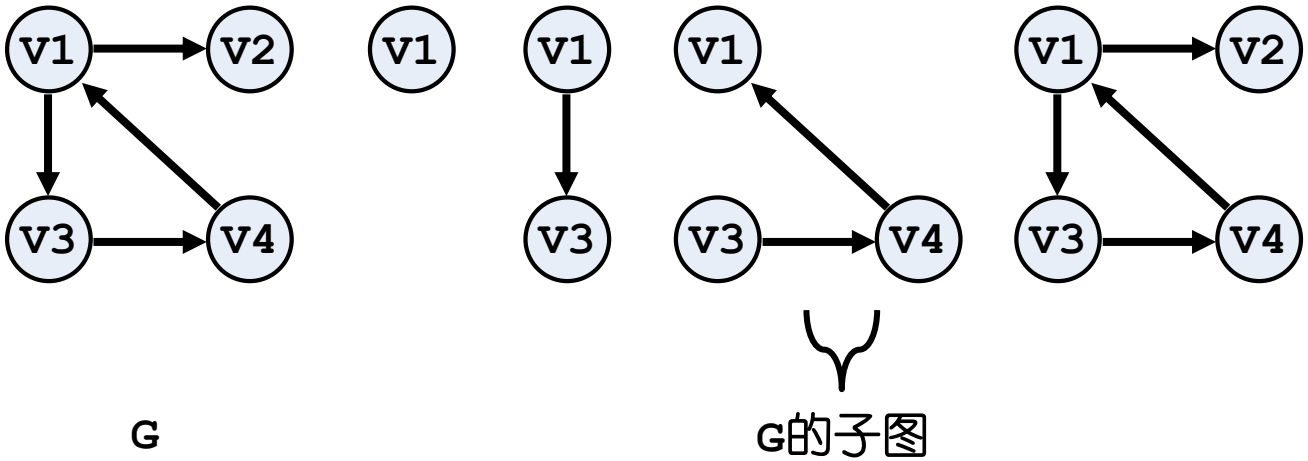
- 若无向图任意2个顶点之间都有一条边，则此图为完全无向图。
- 若有向图任意2个顶点u,v都有2条反向的弧<u,v>和<v,u>，则此图为完全有向图。
- 完全图其实就是边/弧的数量达到最大值





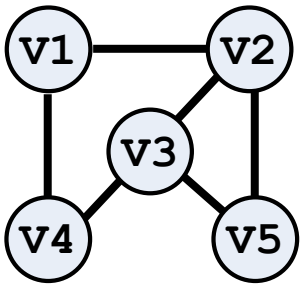
子图

- 有两个图G={V, E},G’={V’,E’}
- 如果V’⊆V, E’⊆E，则称G’为G的子图



邻接点、度

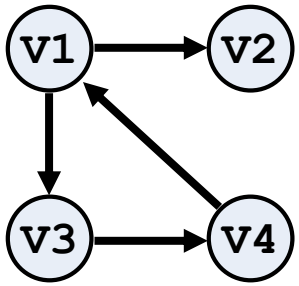
- u, v之间如果有一条边(u,v)或弧<u,v>,就称它们互为邻接点。
- 一个顶点的邻接点或边的个数，称为该顶点的度。



- TD(V1)=2
- TD(V2)=3
- TD(V3)=3
- TD(V4)=2
- TD(V5)=2

入度和出度

- 对有向图： 顶点的出度是以它为弧尾的弧的个数，而入度是以它为弧头的弧的个数 。度= 入度+出度



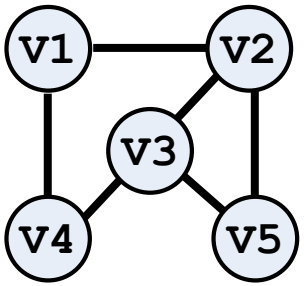
- ID(V1)=1
- OD(V1)=2
- TD(V1)=ID(v1)+OD(v1)=3

度和边数的关系

- 一个有n个顶点，e条边或弧的图满足：

$$e = \frac{1}{2}\sum_{i=1}^{n} TD(v_i)$$

- 即边（或弧）的总数 = 各个顶点的度的总数的一半



- TD(V1)=2
- TD(V2)=3
- TD(V3)=3
- TD(V4)=2
- TD(V5)=2

- e=6

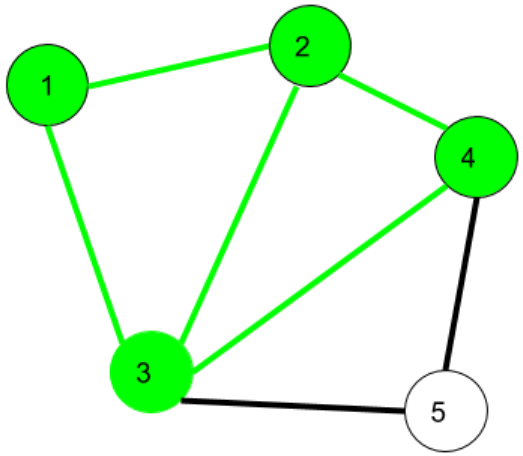
路径、简单路径、trail、回路，简单回路

- **Walk（路径）**：从一个顶点经过一系列边到达另外一个顶点，所经过的顶点和边的序列。
- **Trail**：没有重复边的路径
- **Path（简单路径）**：没有重复顶点和边的路径。
- 回路(**Circuit**): 没有重复边（可能有重复顶点）、起点终点相同的路径
- **Cycle(简单回路)**: 除起点终点相同、没有重复顶点和重复边的回路

Walk (路径)： 从一个顶点经过一系列边到达另外一个顶点，所经过的顶点和边的序列。

Vertices may repeat. Edges may repeat (Closed or Open)

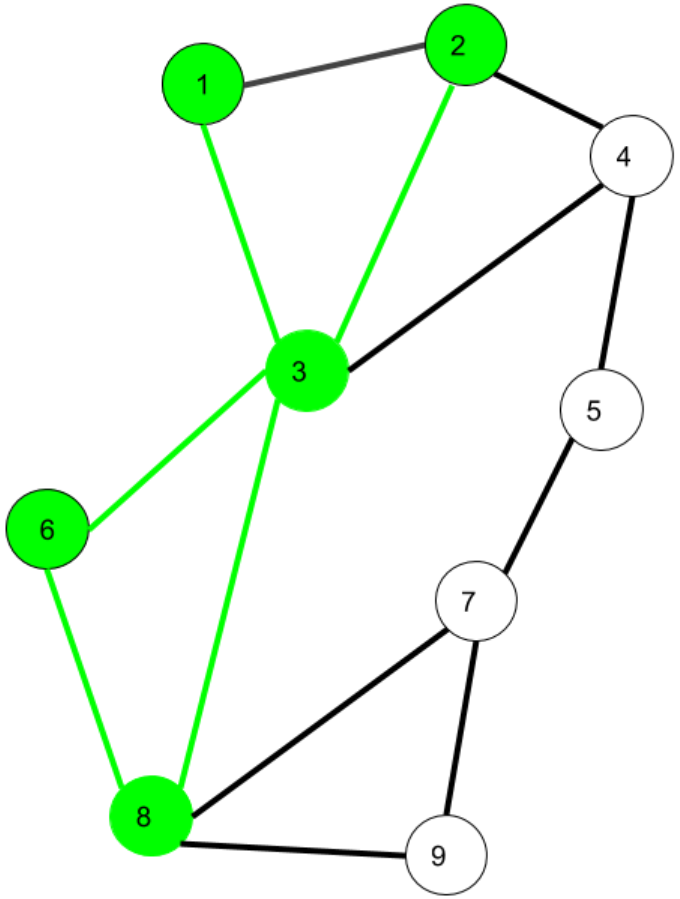
如： 1-2-3-4-2-1



Trail： Vertices may repeat. Edges cannot repeat (Open)

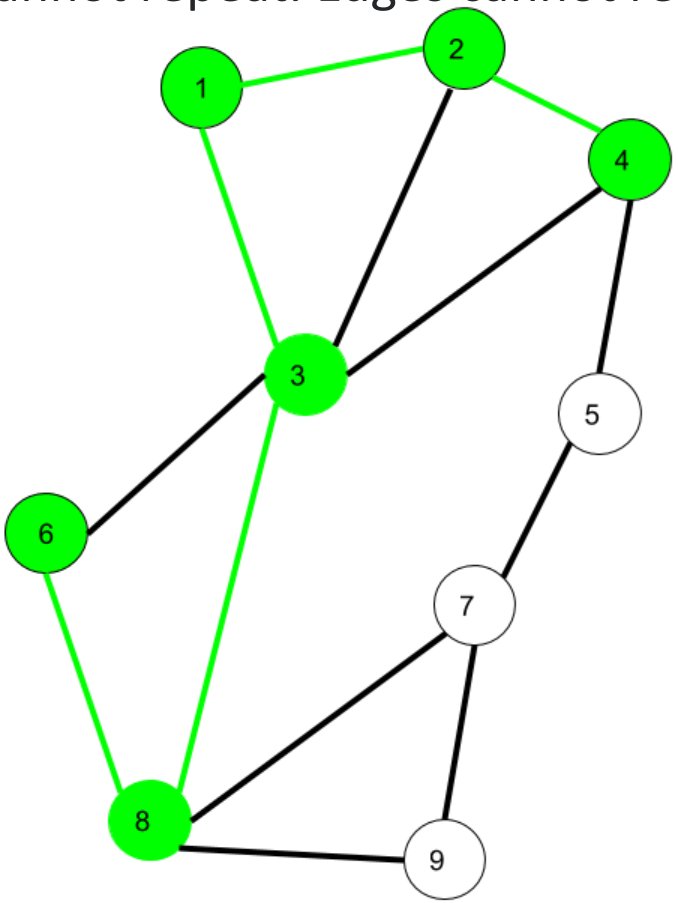
如： 1-3-8-6-3-2

1-3-8-6-3-2-1



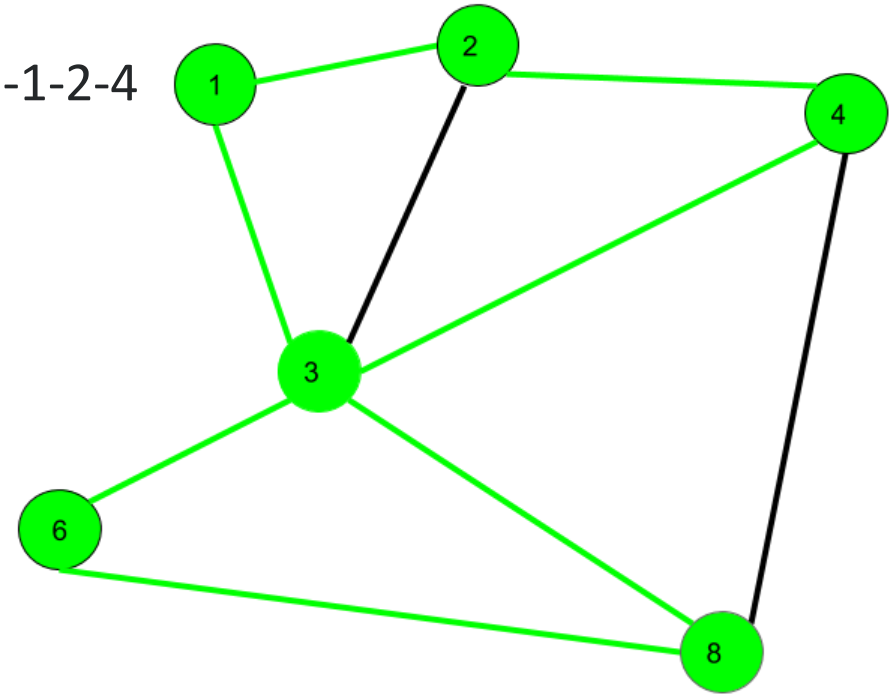
path : Vertices cannot repeat. Edges cannot repeat (Open)

如： 6-8-3-1-2-4



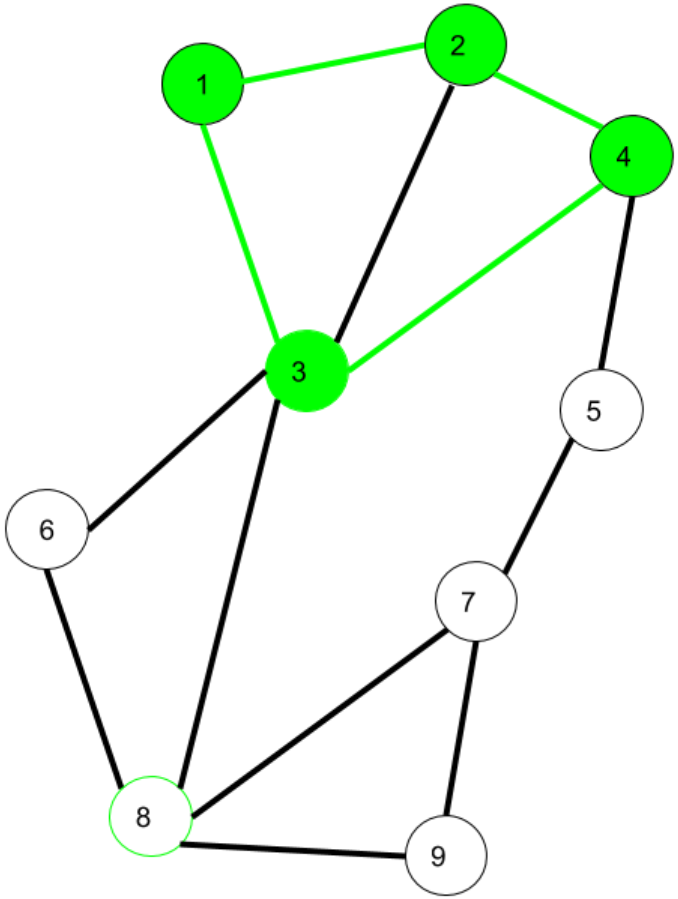
circuit : Vertices may repeat. Edges cannot repeat (Closed)

如： 6-8-3-1-2-4



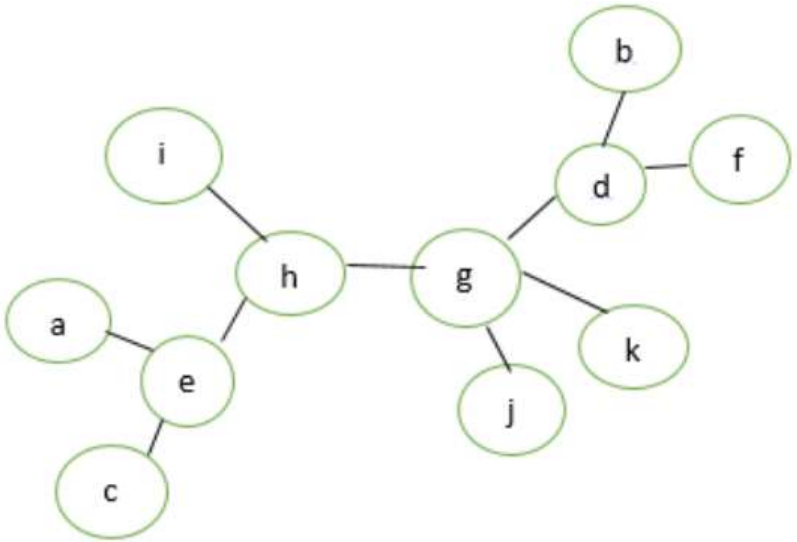
cycle : Vertices cannot repeat. Edges cannot repeat (Closed)

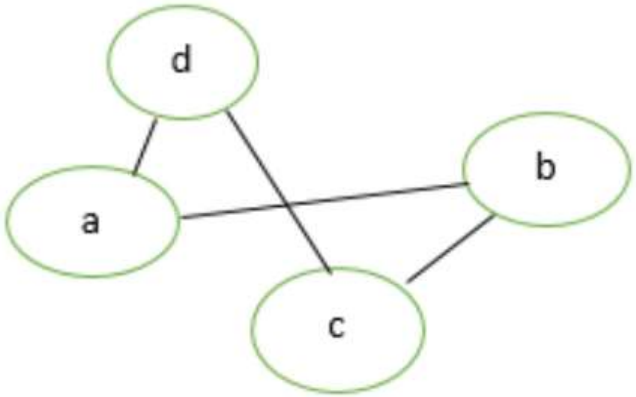
如： 1-2-4-3-1



树是一种特殊的图

- 没有（简单）回路（cycle）的连通图称为**树**。





Not a tree

图的遍历

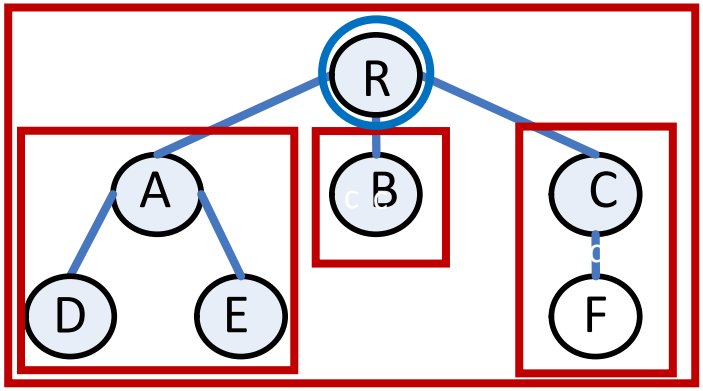
Youtube频道： **hwdong**

博客: **hwdong-net**.github.io

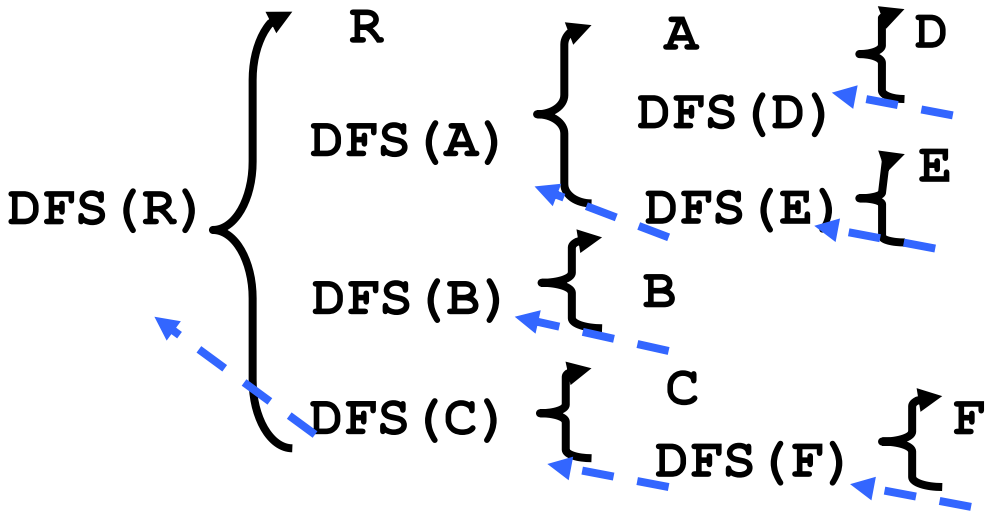
图的遍历

- 访问图中所有顶点，并且每个顶点仅被访问一次
- 类似于树的先根遍历和层次遍历,有图的**深度优先遍历**和**广度优先遍历**

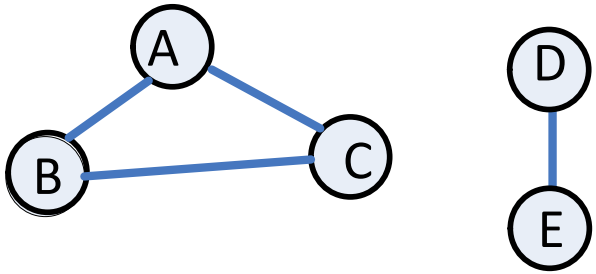
回顾树的先根遍历




```
DFS(V){
    直接访问V;
    for(V的每个孩子W)
        DFS(W);
}
```



树的遍历方法是否不修改就适用于图的遍历呢?




```
DFS(V){
    直接访问V;
    for(V的每个邻接点W)
        DFS(W);
}
```

问题 1：由于图存在环路，所以会导致无限循环

解决方法：设置顶点访问标志

问题 2：由于图不一定连通，从一个顶点出发的遍历只能访问其所在的连通分量中的所有顶点

解决方法：重复调用从一个顶点出发的DFS

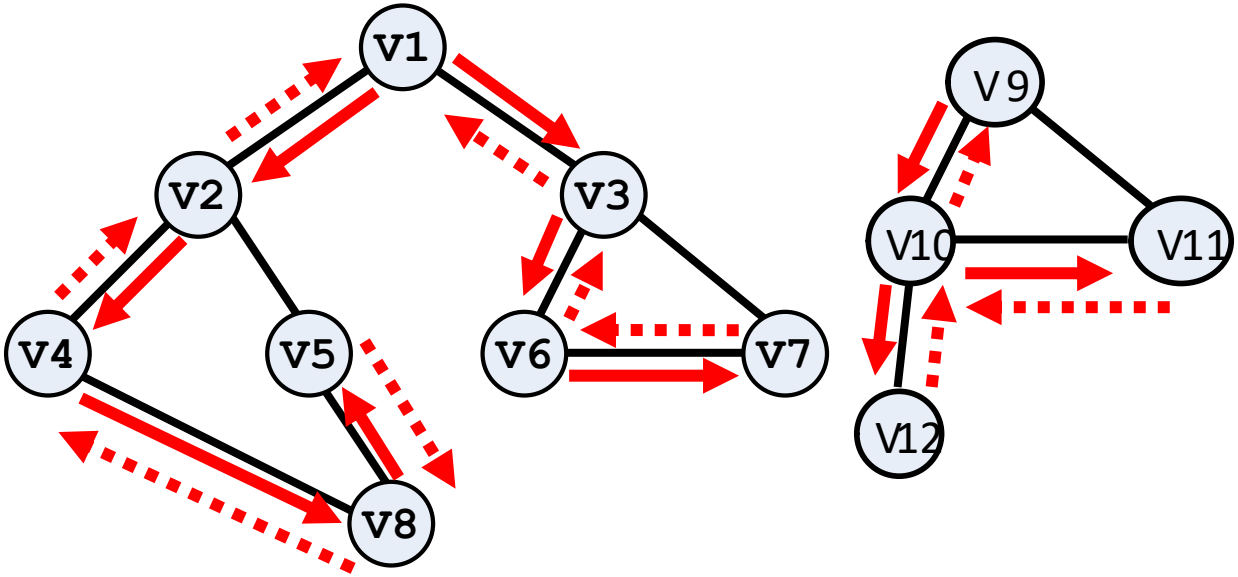
图的深度优先遍历

```
DFS(V){
    直接访问V;
    flag(v) = 1;
    for(V的每个邻接点W)
        if( !flag(w))
            DFS(W);
}
```

从一个顶点出发的深度优先遍历

```
DFS(G){
   for(每个v)
      flag[v]= 0;
   for(每个v)
      if(!flag(v))
         DFS(v);
}
```

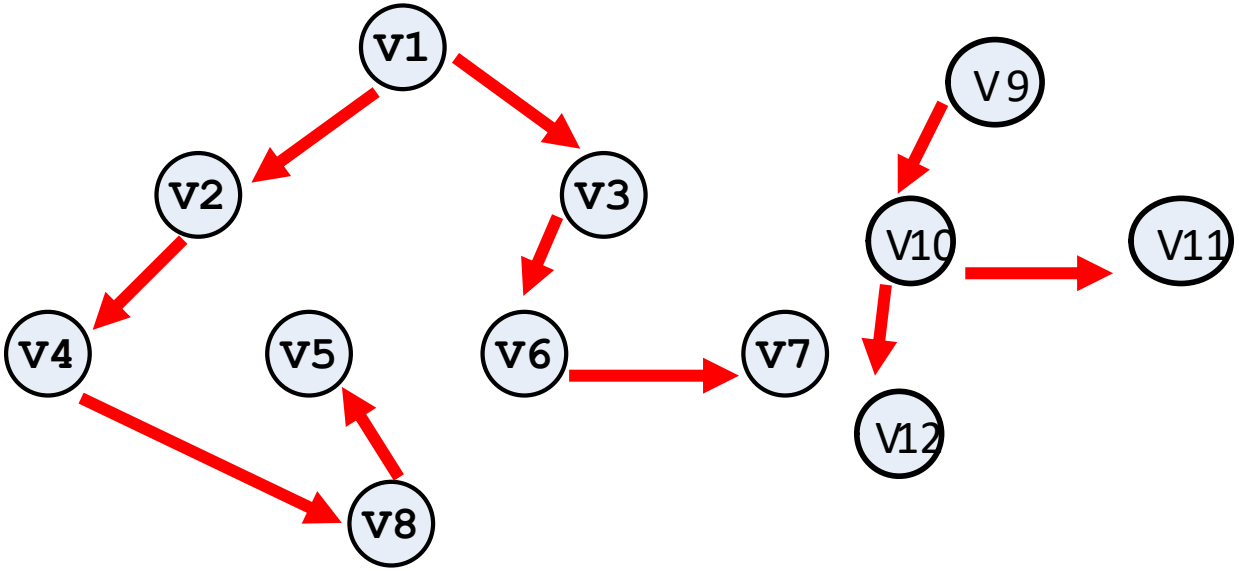
整个图的深度优先遍历



V1->V2->V4->V8->V5->V3->V6->V7

V9->V10->V12->V11

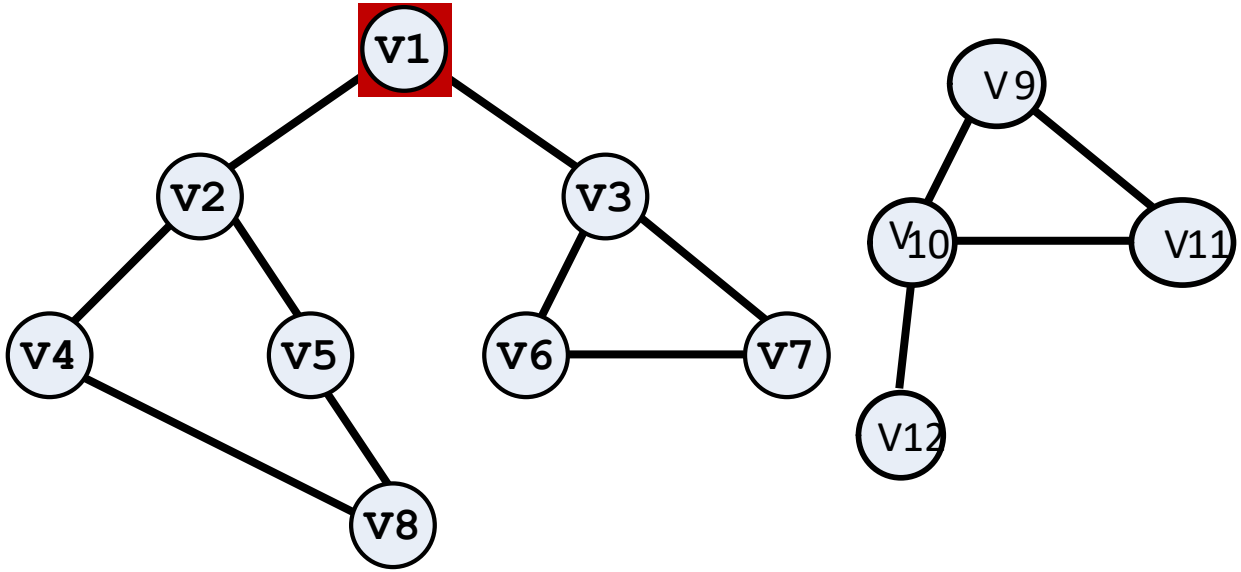
图的深度优先遍历是一个**树形穷举**过程



深度优先搜索树

DFS(V1)

访问V1做标记



V1

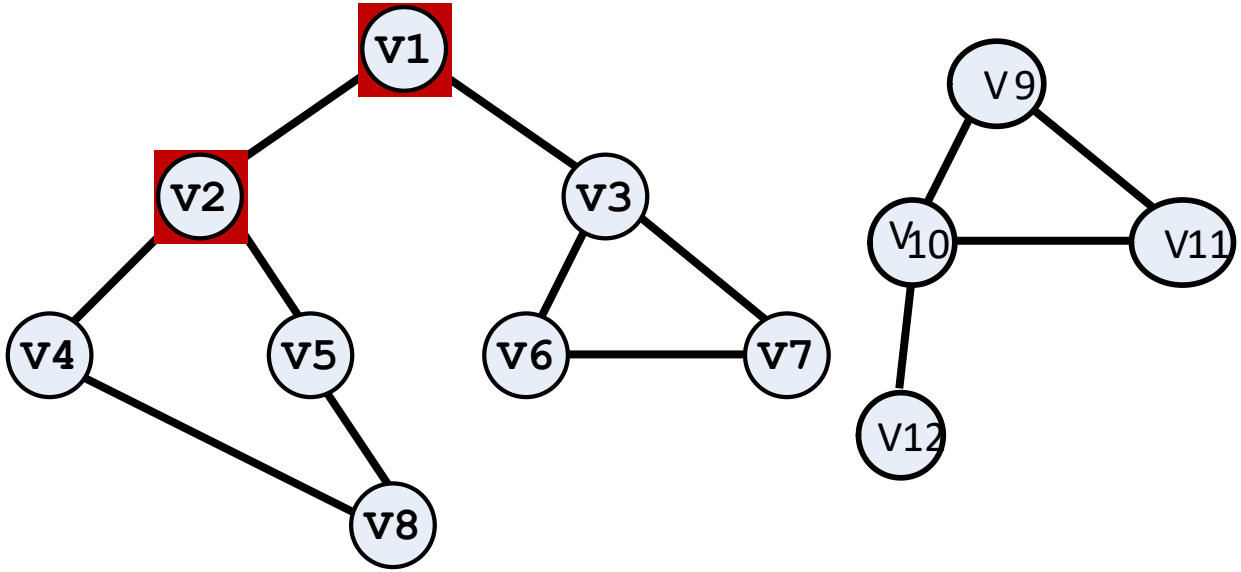
1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0

DFS(V1)

访问V1做标记

DFS(V2)

访问V2做标记



V1 V2

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	0	0	0	0	0	0	0	0	0

DFS(V1)

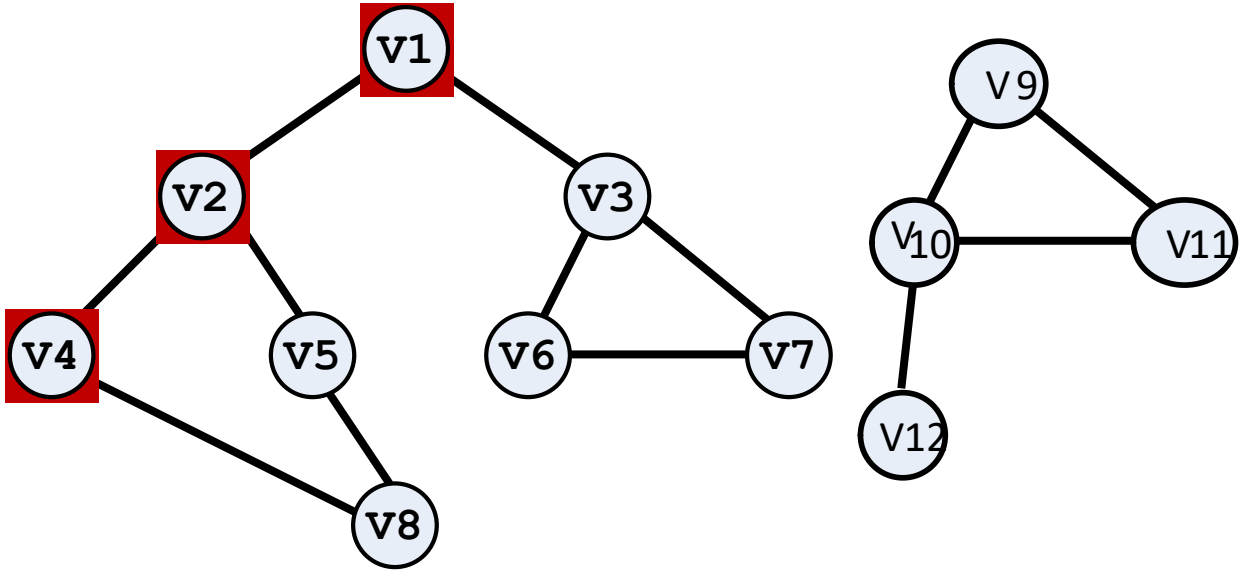
访问V1做标记

DFS(V2)

访问V2做标记

DFS(V4)

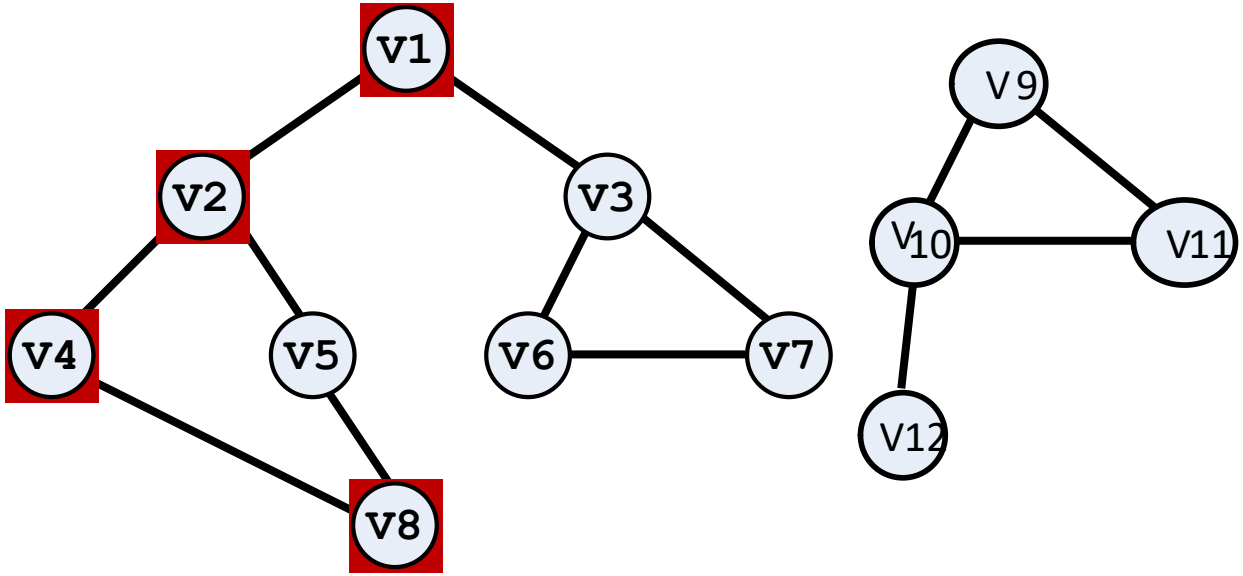
访问V4做标记



V1 V2 V4

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	0	0	0	0	0	0	0	0

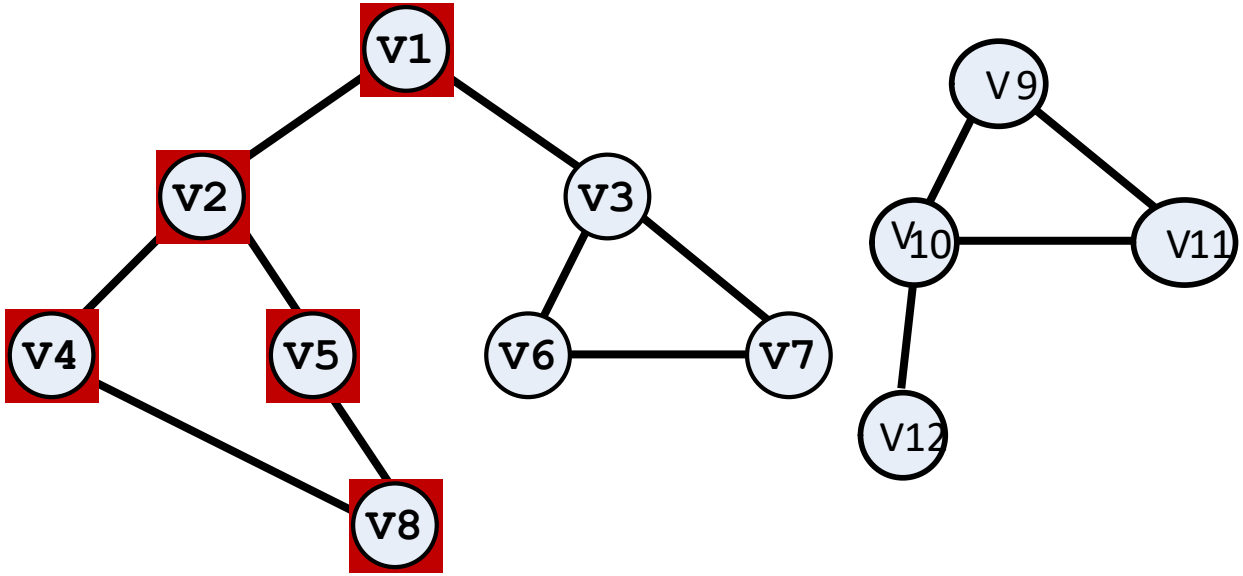
DFS(V1)
    访问V1做标记
    DFS(V2)
        访问V2做标记
        DFS(V4)
            访问V4做标记
            DFS(V8)
                访问V8做标记



V1 V2 V4 V8

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	0	0	0	1	0	0	0	0

DFS(V1)
    访问V1做标记
    DFS(V2)
        访问V2做标记
        DFS(V4)
            访问V4做标记
            DFS(V8)
                访问V8做标记
                DFS(V5)
                    访问V5做标记

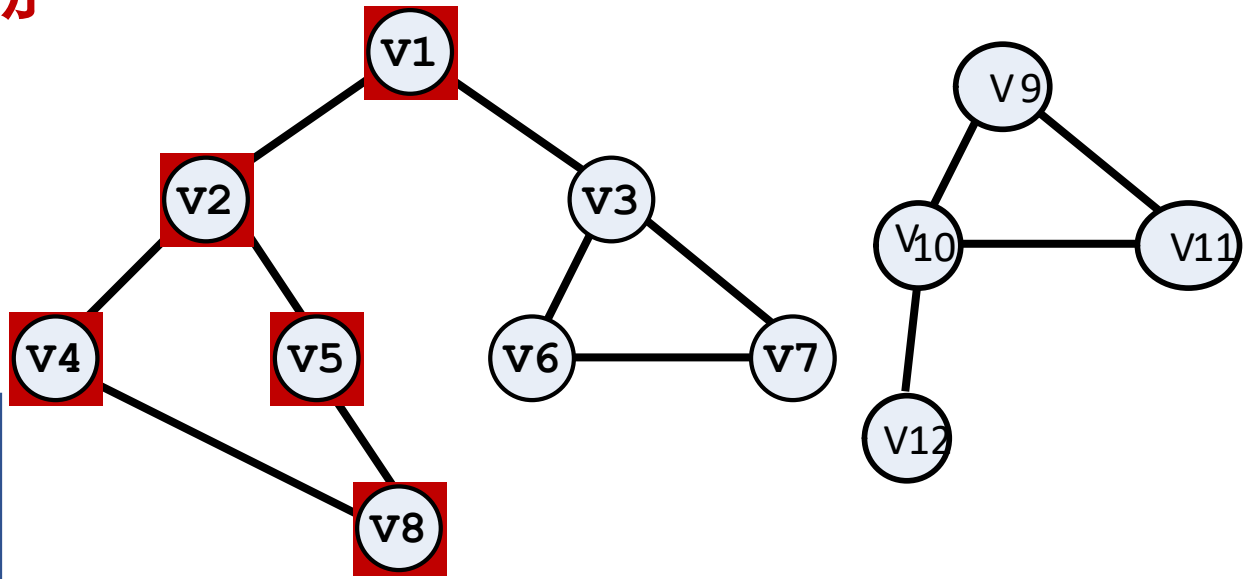


V1 V2 V4 V8 V5

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	1	0	0	1	0	0	0	0

回溯

DFS(V1)
- 访问V1做标记
- DFS(V2)
  - 访问V2做标记
  - DFS(V4)
    - 访问V4做标记
    - DFS(V8)
      - 访问V8做标记
      - DFS(V5)
        - 访问V5做标记

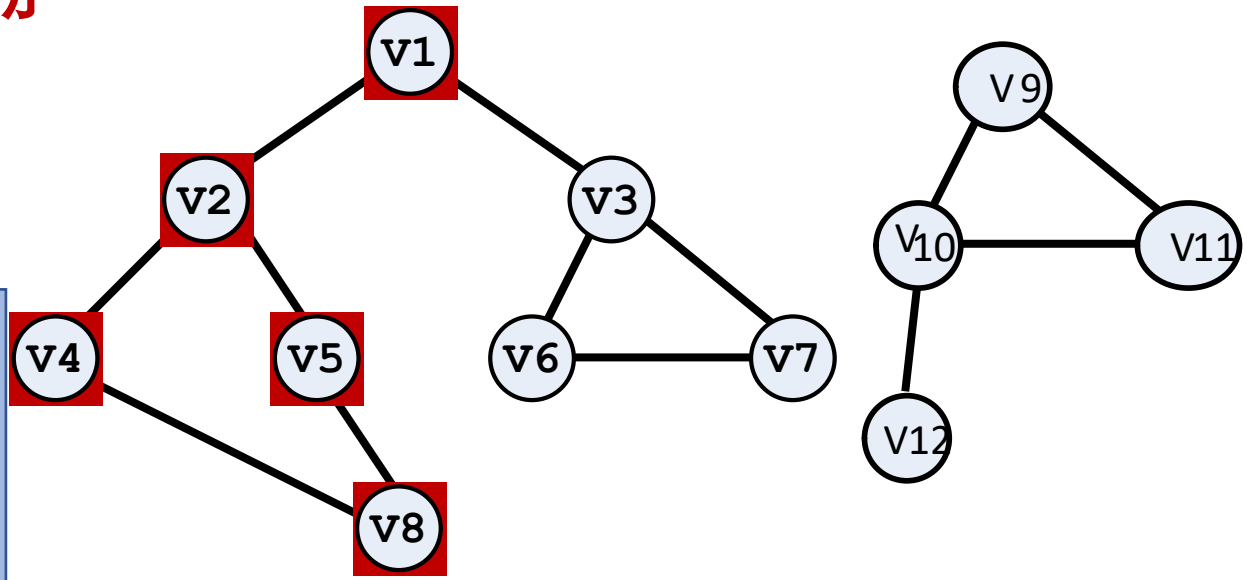


V1 V2 V4 V8 V5

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	1	0	0	1	0	0	0	0

回溯

DFS(V1)
访问V1做标记
DFS(V2)
访问V2做标记
DFS(V4)
访问V4做标记
DFS(V8)
访问V8做标记
DFS(V5)
访问V5做标记

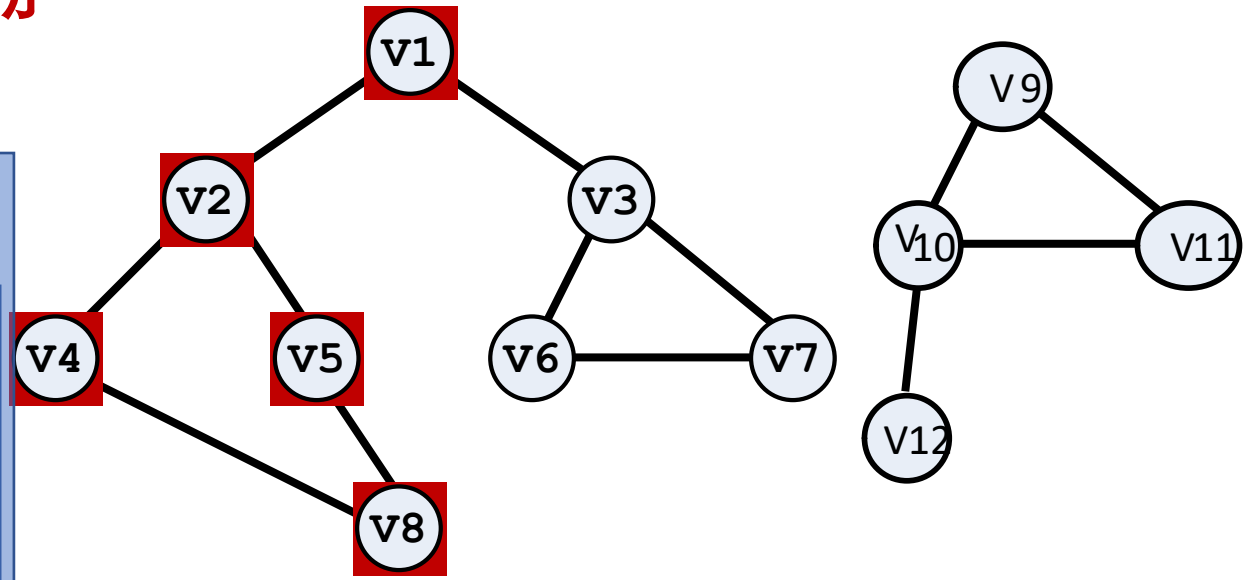


V1 V2 V4 V8 V5

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	1	0	0	1	0	0	0	0

回溯

DFS(V1)
访问V1做标记
DFS(V2)
访问V2做标记
DFS(V4)
访问V4做标记
DFS(V8)
访问V8做标记
DFS(V5)
访问V5做标记



V1 V2 V4 V8 V5

1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	1	1	0	0	1	0	0	0	0

DFS(V1)
访问V1做标记

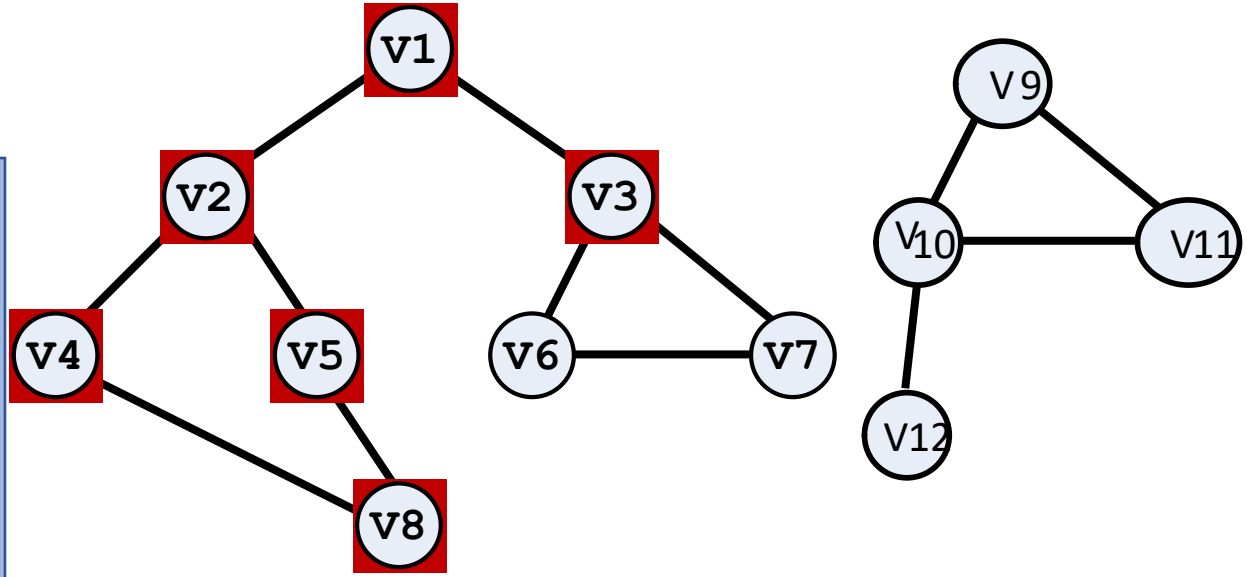
DFS(V2)
访问V2做标记

DFS(V4)
访问V4做标记

DFS(V8)
访问V8做标记

DFS(V5)
访问V5做标记

DFS(V3)
访问V3做标记



V1	V2	V4	V8	V5	V3						
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	0	0	1	0	0	0	0

DFS(V1)
访问V1做标记

DFS(V2)
访问V2做标记

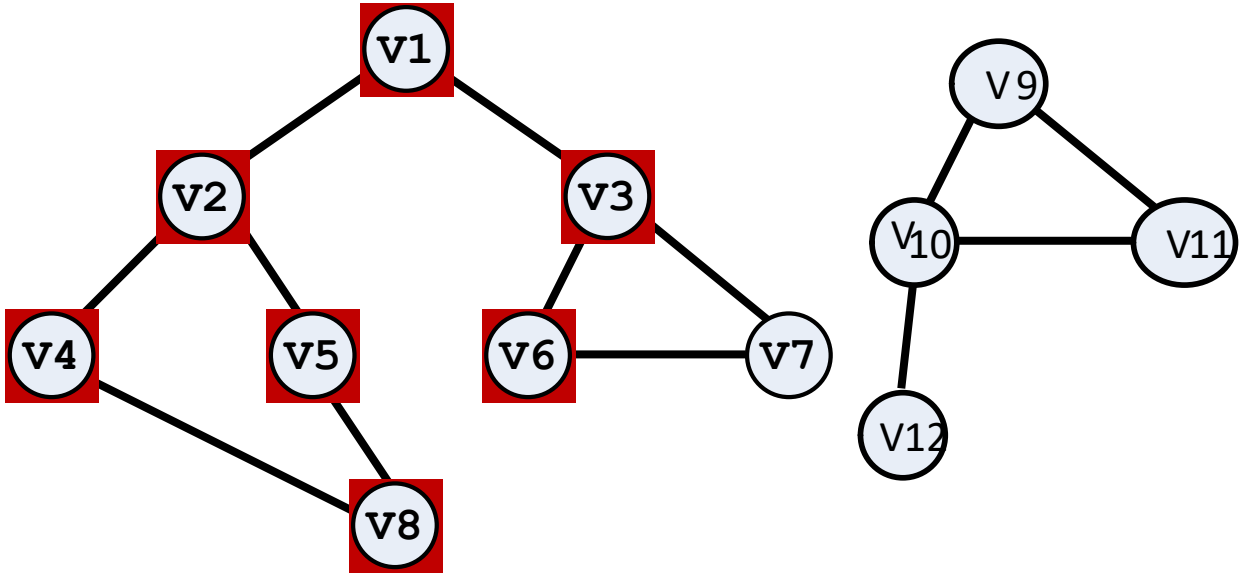
DFS(V4)
访问V4做标记

DFS(V8)
访问V8做标记

DFS(V5)
访问V5做标记

DFS(V3)
访问V3做标记

DFS(V6)
访问V6做标记



V1	V2	V4	V8	V5	V3	V6					
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	0	1	0	0	0	0

DFS(V1)
访问V1做标记

DFS(V2)
访问V2做标记

DFS(V4)
访问V4做标记

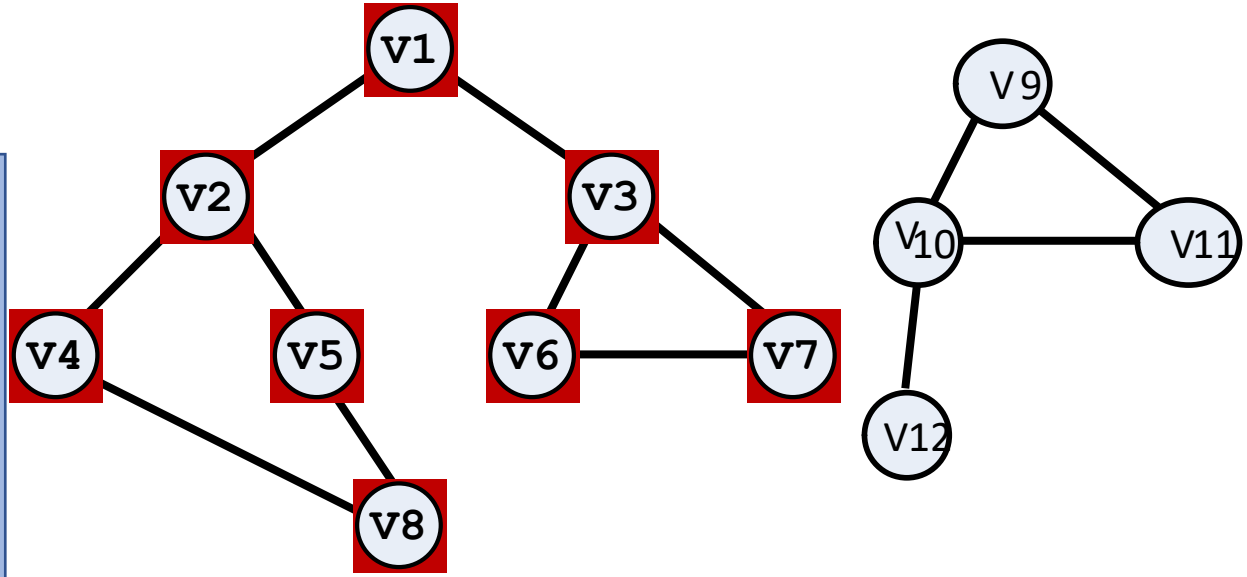
DFS(V8)
访问V8做标记

DFS(V5)
访问V5做标记

DFS(V3)
访问V3做标记

DFS(V6)
访问V6做标记

DFS(V7)
访问V7做标记



V1	V2	V4	V8	V5	V3	V6	V7				
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0

DFS(V1)
访问V1做标记

DFS(V2)
访问V2做标记

DFS(V4)
访问V4做标记

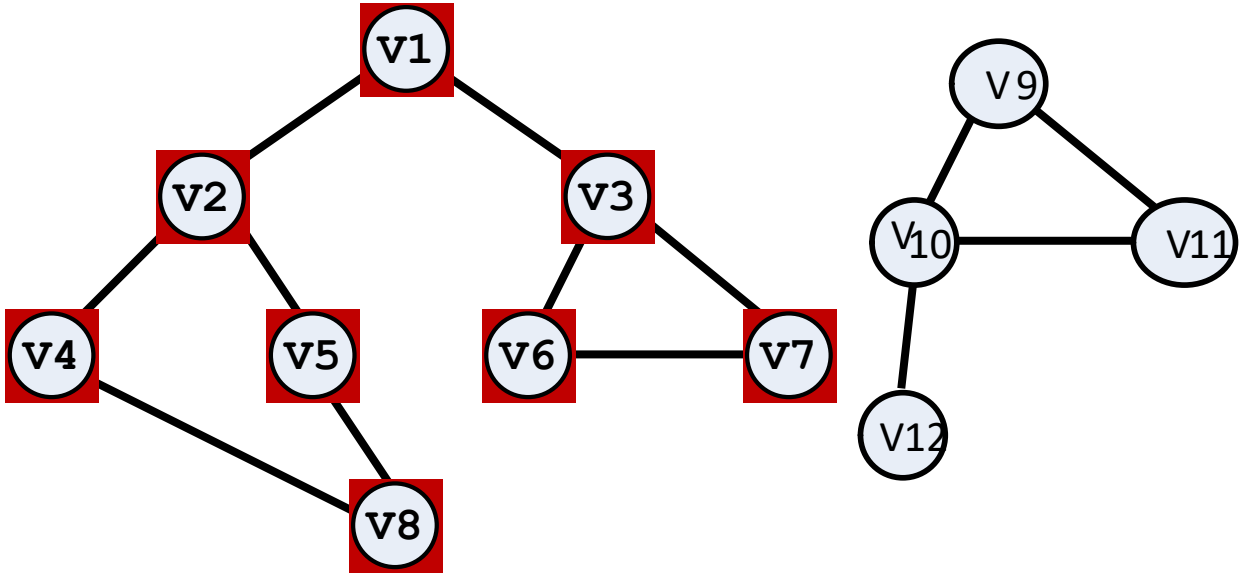
DFS(V8)
访问V8做标记

DFS(V5)
访问V5做标记

DFS(V3)
访问V3做标记

DFS(V6)
访问V6做标记

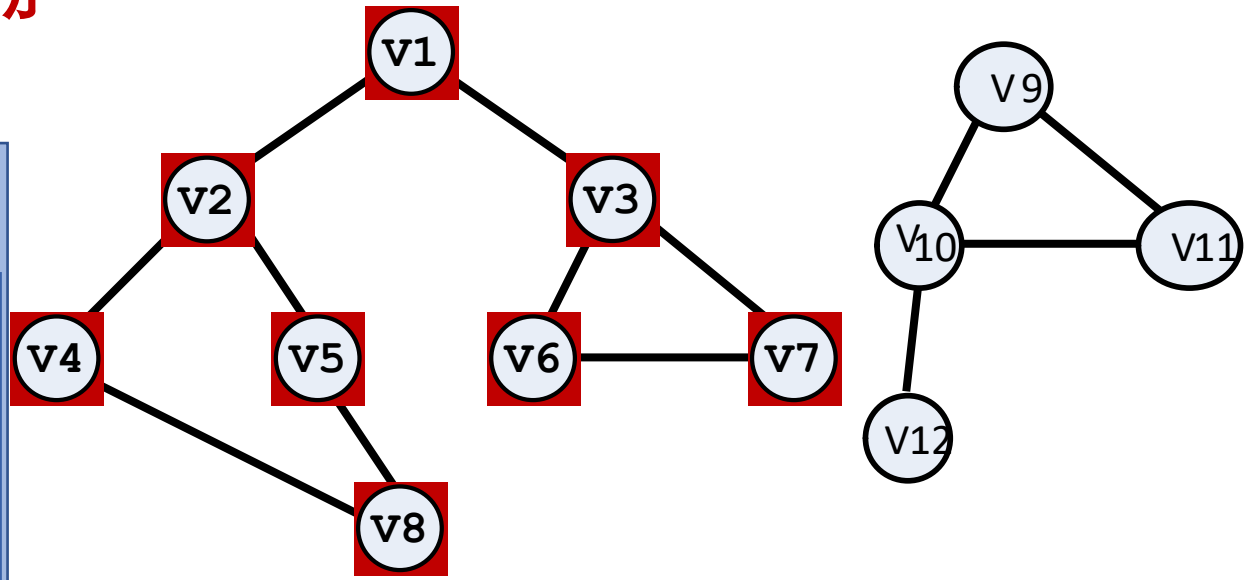
DFS(V7)
访问V7做标记



V1	V2	V4	V8	V5	V3	V6	V7				
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0

回溯

DFS(V1)
访问V1做标记
DFS(V2)
访问V2做标记
DFS(V4)
访问V4做标记
DFS(V8)
访问V8做标记
DFS(V5)
访问V5做标记



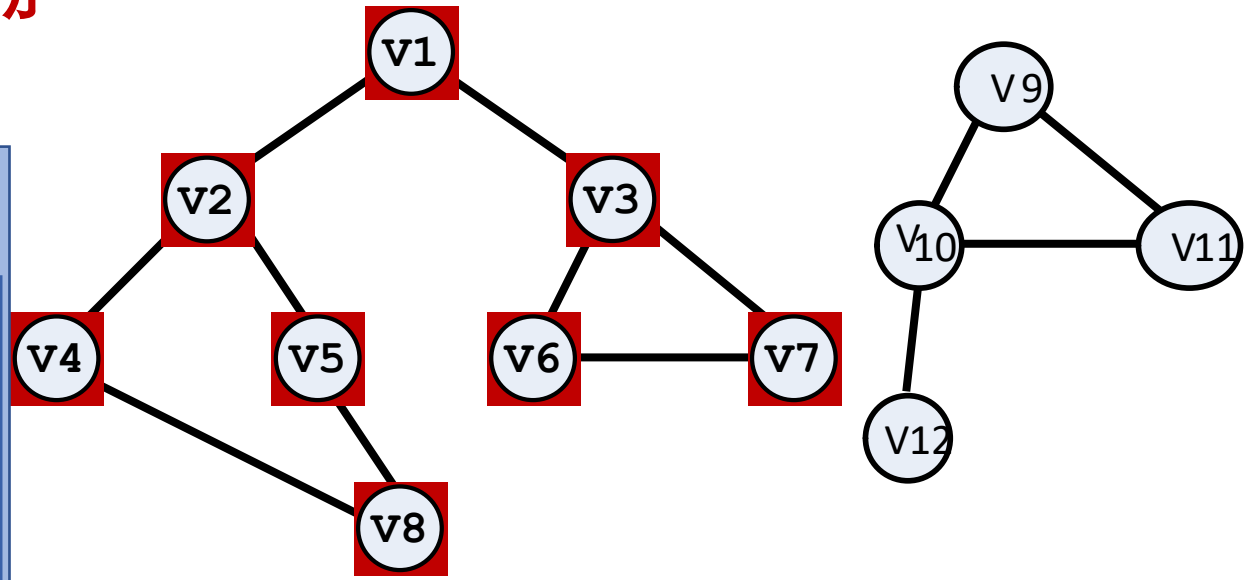
DFS(V3)
访问V3做标记
DFS(V6)
访问V6做标记
DFS(V7)
访问V7做标记

V1	V2	V4	V8	V5	V3	V6	V7				
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0

回溯

DFS(V1)
访问V1做标记
DFS(V2)
访问V2做标记
DFS(V4)
访问V4做标记
DFS(V8)
访问V8做标记
DFS(V5)
访问V5做标记

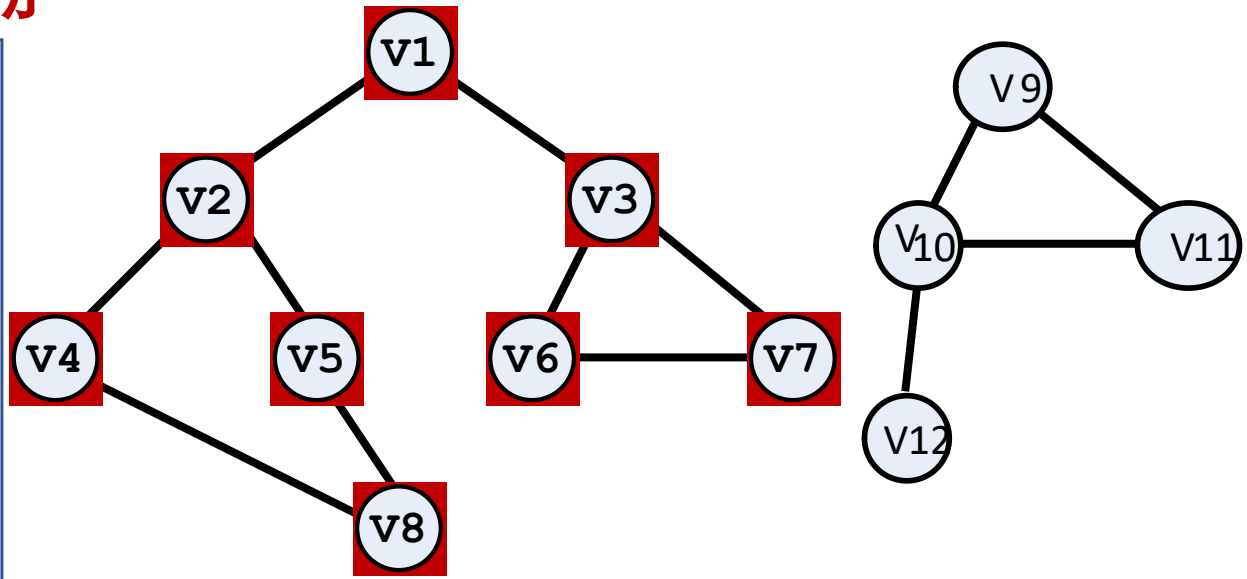
DFS(V3)
访问V3做标记
DFS(V6)
访问V6做标记
DFS(V7)
访问V7做标记



V1	V2	V4	V8	V5	V3	V6	V7				
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0

回溯

DFS(V1)
访问V1做标记
DFS(V2)
访问V2做标记
DFS(V4)
访问V4做标记
DFS(V8)
访问V8做标记
DFS(V5)
访问V5做标记
DFS(V3)
访问V3做标记
DFS(V6)
访问V6做标记
DFS(V7)
访问V7做标记



V1	V2	V4	V8	V5	V3	V6	V7				
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0

深度优先遍历

=

回溯法

时间复杂度

- 1）初始化访问标志 O(n)
- 2) 每个顶点的DFS,对每个顶点，要访问其邻接点：

对于邻接表，次数为顶点的度，而所有顶点的度之和为2e，加上顶点都访问一次，共（n+2e）

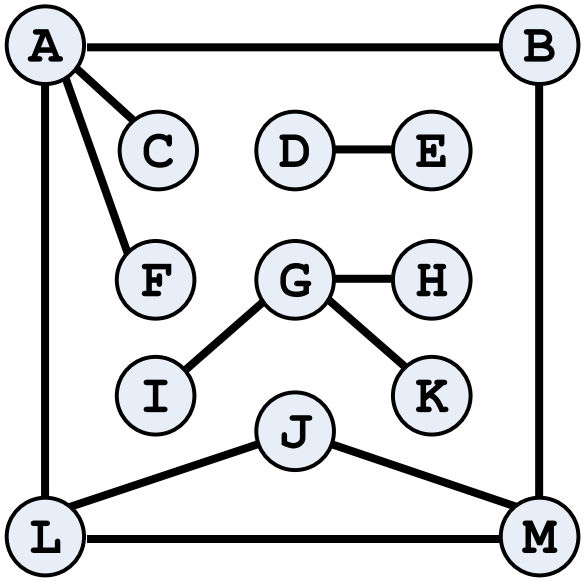
对于邻接矩阵，次数为n*n

```
DFS(G){
  for(每个v)
      flag = 0;
  for(每个v)
      if(!flag(v))
         DFS(v);
}
```

练习

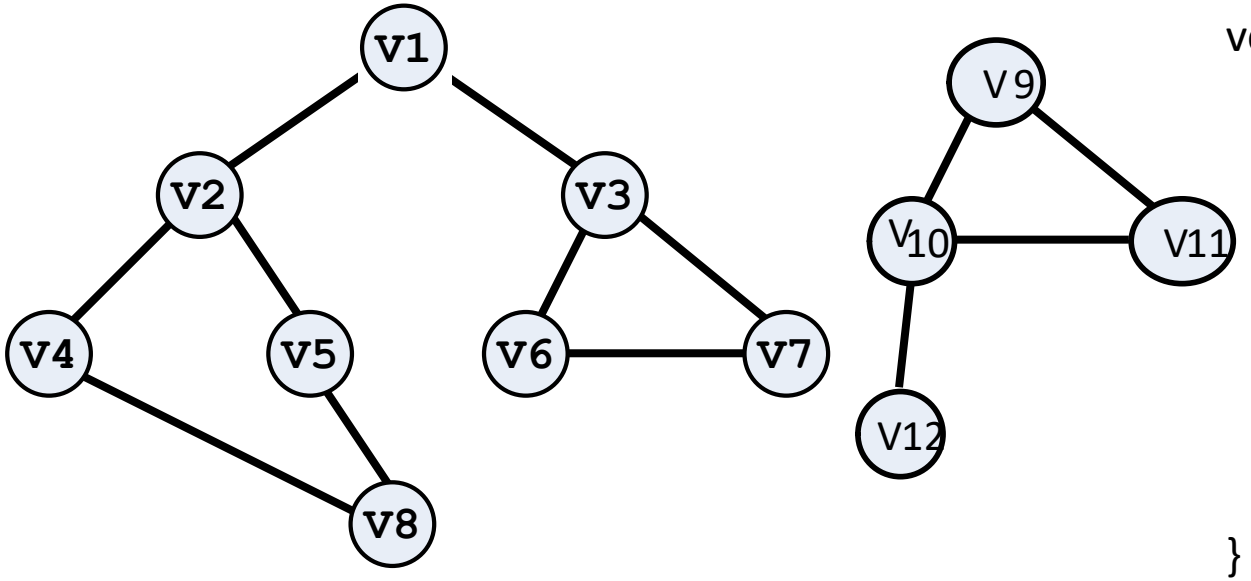
- 写出对下图进行深度优先遍历的结果

A B M J L C F
D E
G H I K



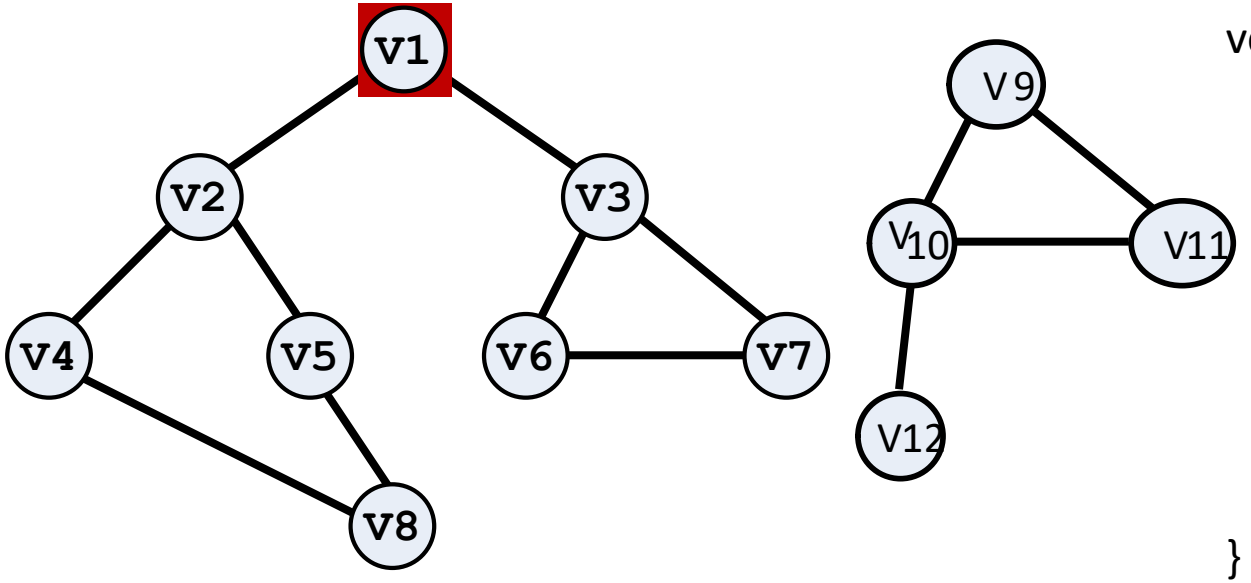
基于堆栈的深度优先遍历

```
void DFS(sV){
   S.push(sV);  mark(sV)     //入栈、设标记
    while(!S.empty()){
        v = S.top(); S.pop();
         visit(v);                          //访问它
         for(v的未访问邻接点w){
             S.push(w); mark(w);    //入栈、设标记
        }
    }
}
```




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

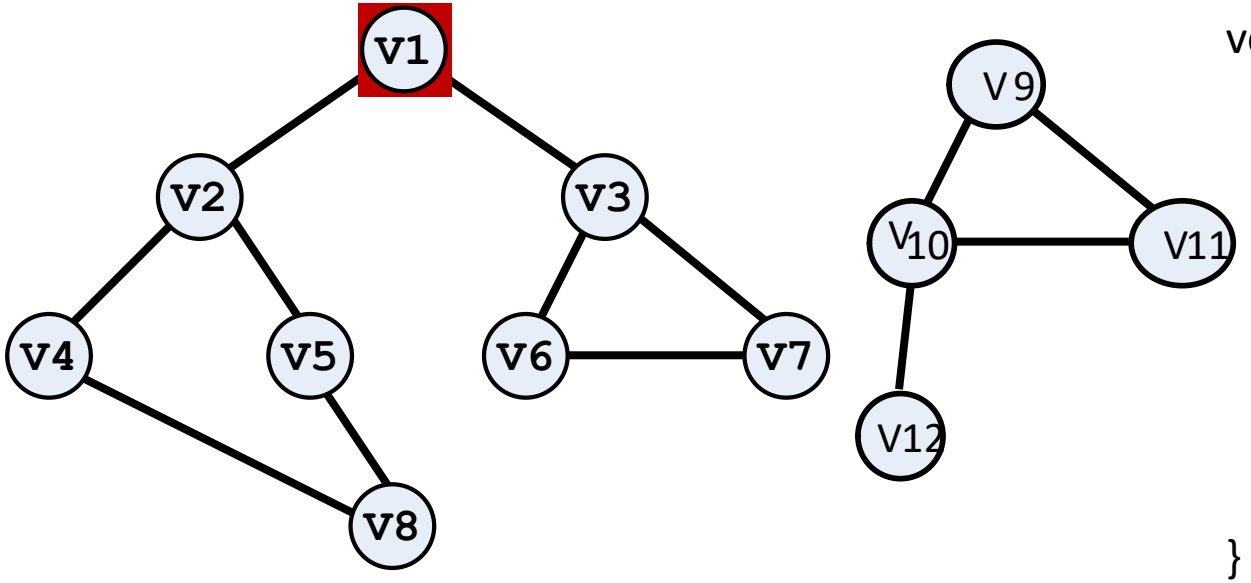
1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1

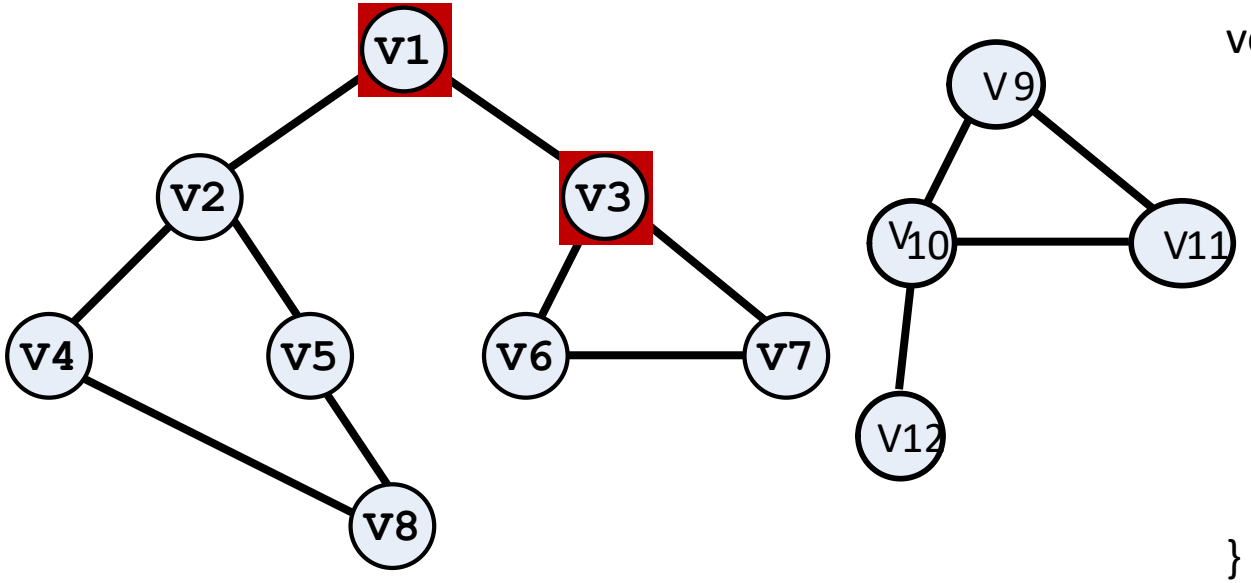
1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1

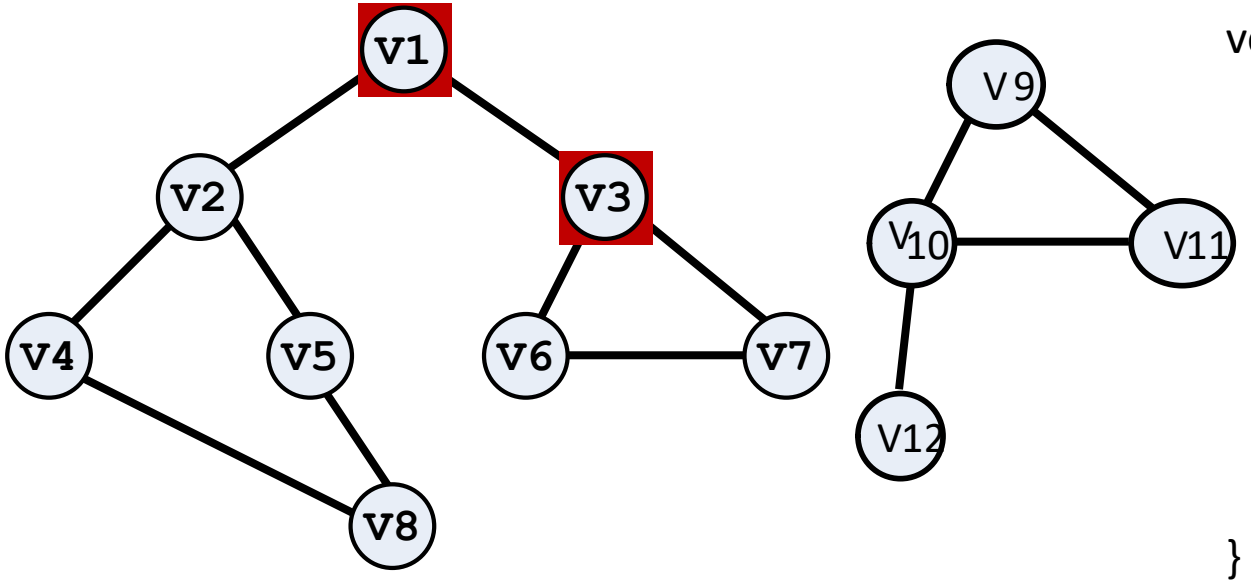
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	0	0	0	0	0	0	0	0	0



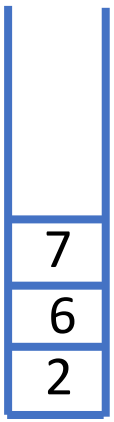

```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3

1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	0	0	0	0	0	0	0	0	0

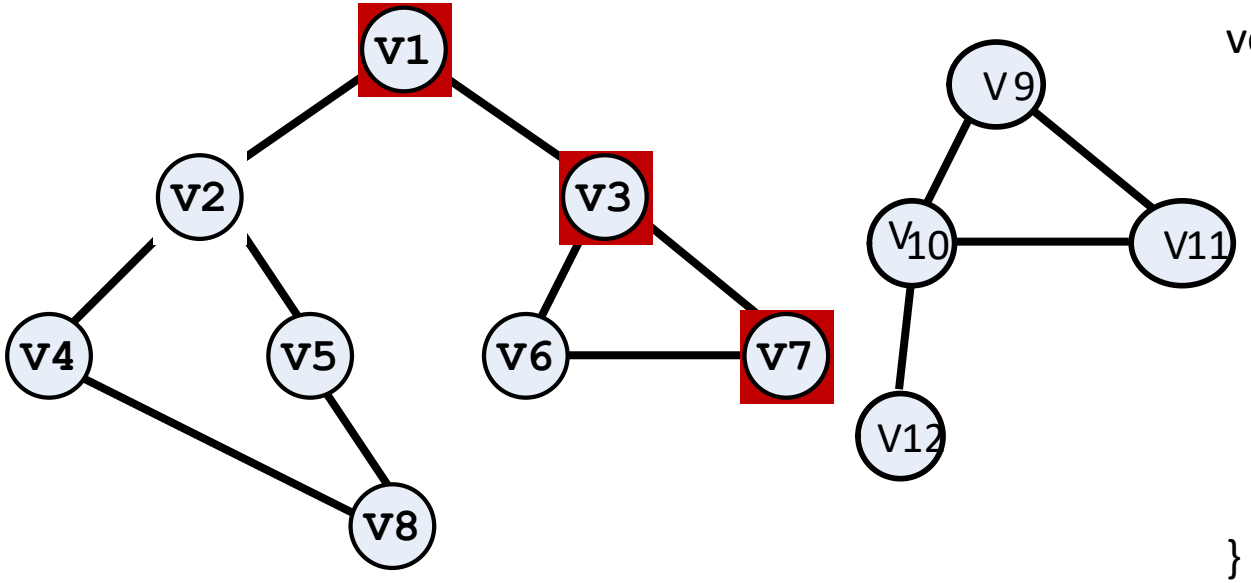



```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```



V1 V3

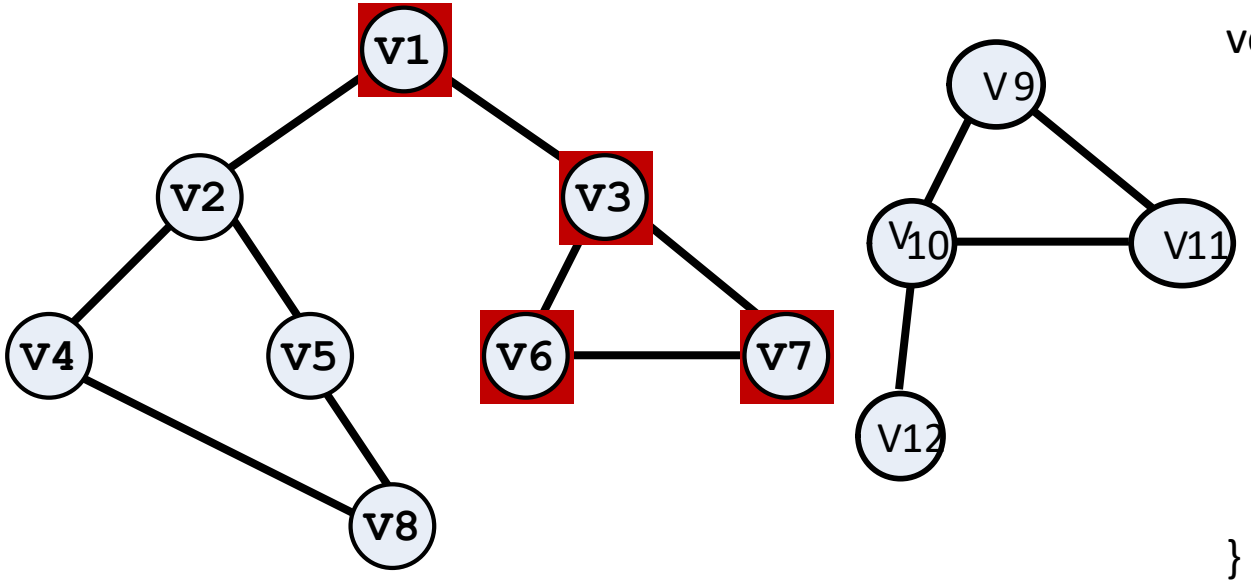
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	0	0	1	1	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7

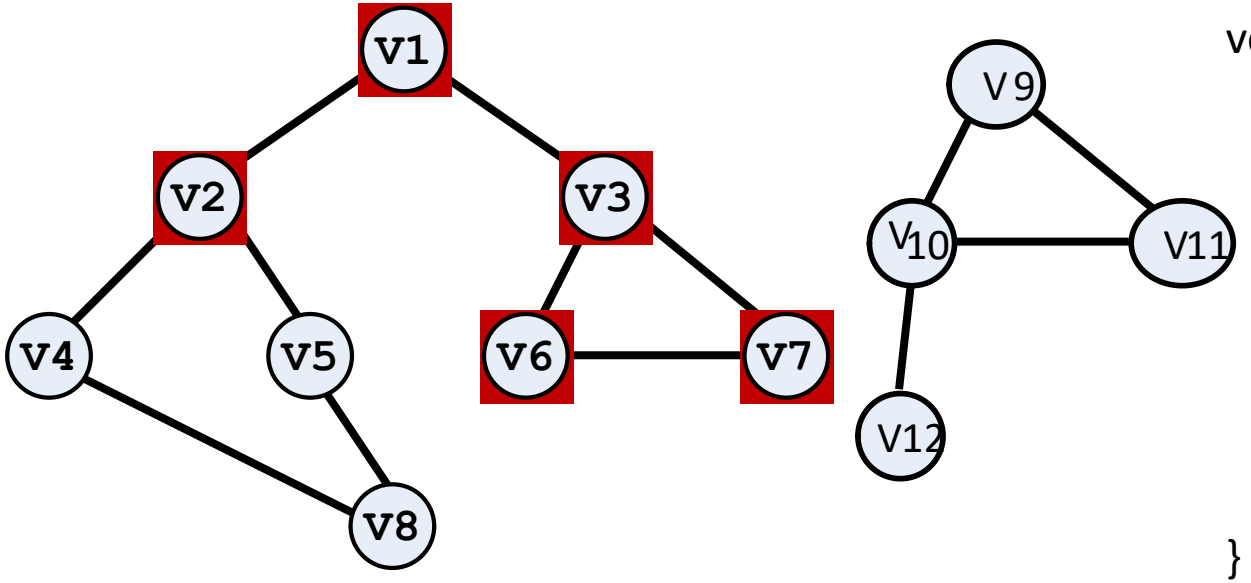
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	0	0	1	1	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7 V6

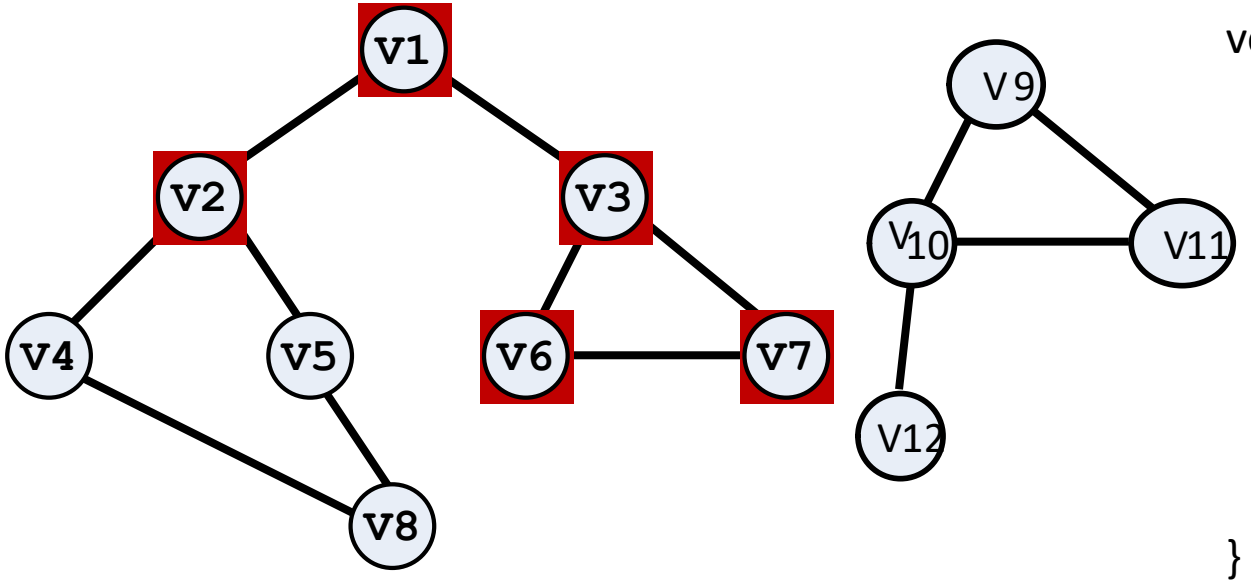
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	0	0	1	1	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7 V6 V2

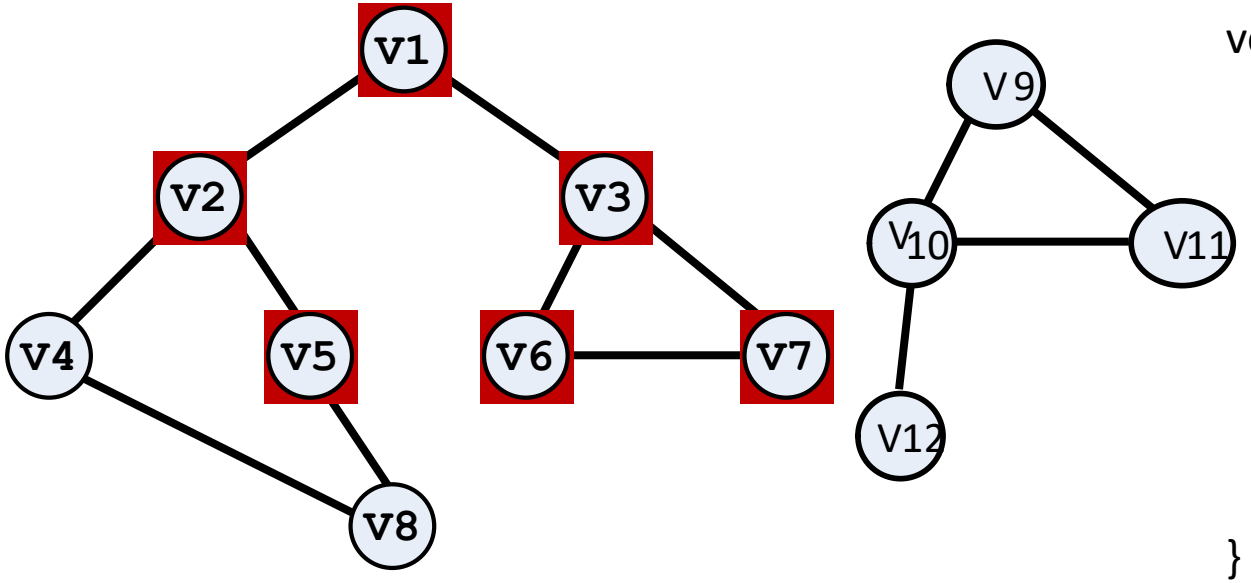
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	0	0	1	1	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7 V6 V2

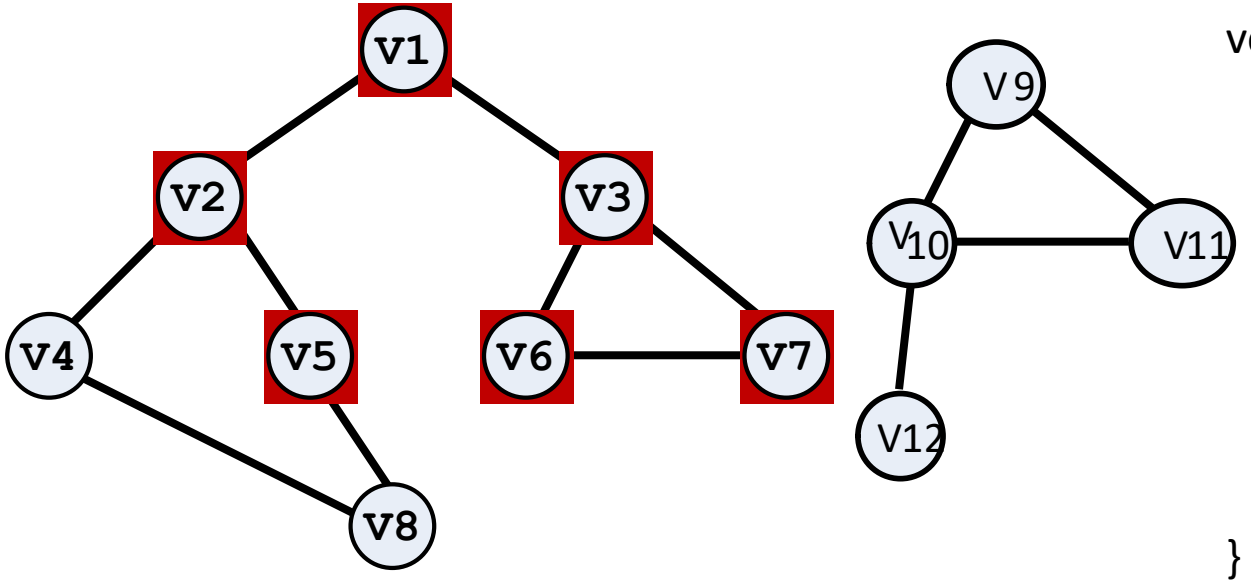
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7 V6 V2 V5

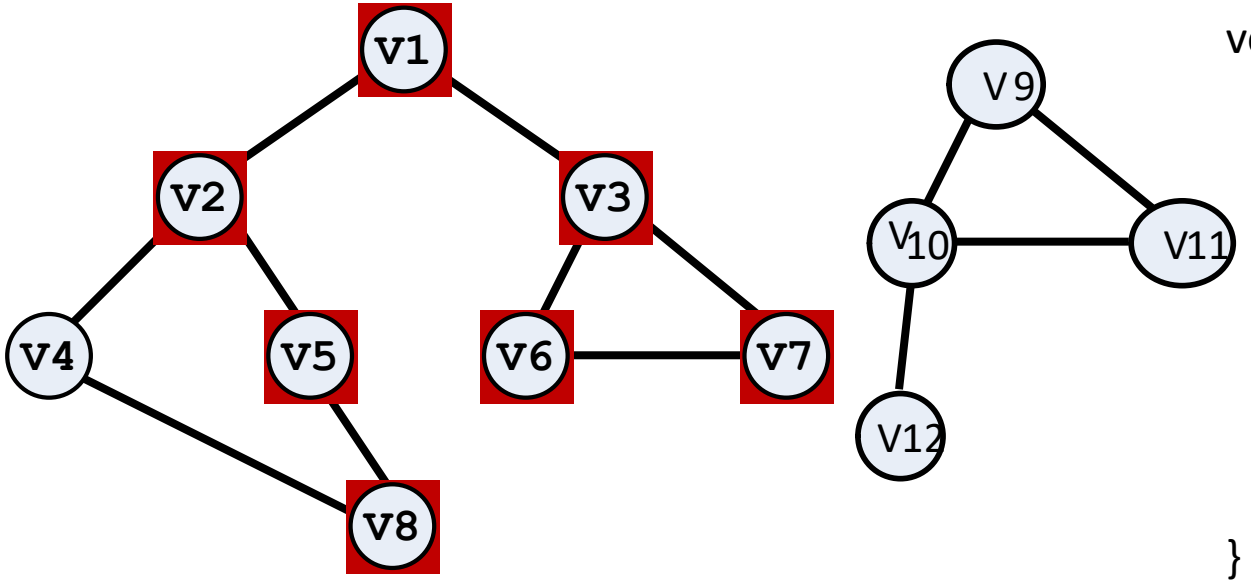
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	0	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7 V6 V2 V5

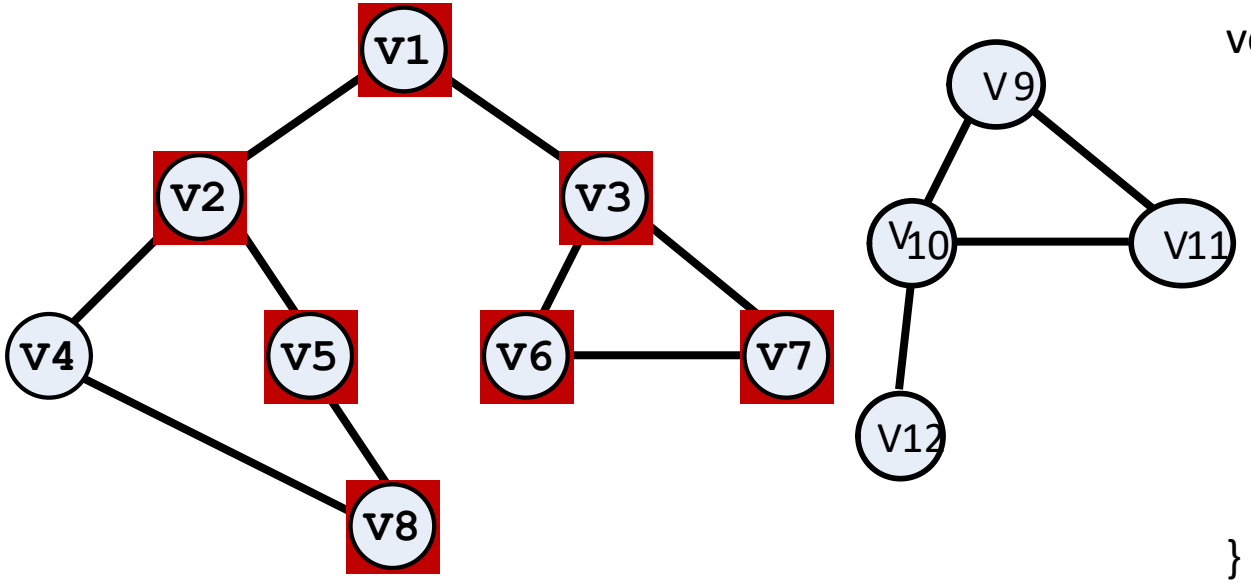
1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

V1 V3 V7 V6 V2 V5 V8

1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0




```
void DFS(sV){
    S.push(sV);  mark(sV)
    while(!S.empty()){
        v = S.top(); S.pop();
        visit(v);
        for(v的未访问邻接点w){
            S.push(w); mark(w);
        }
    }
}
```

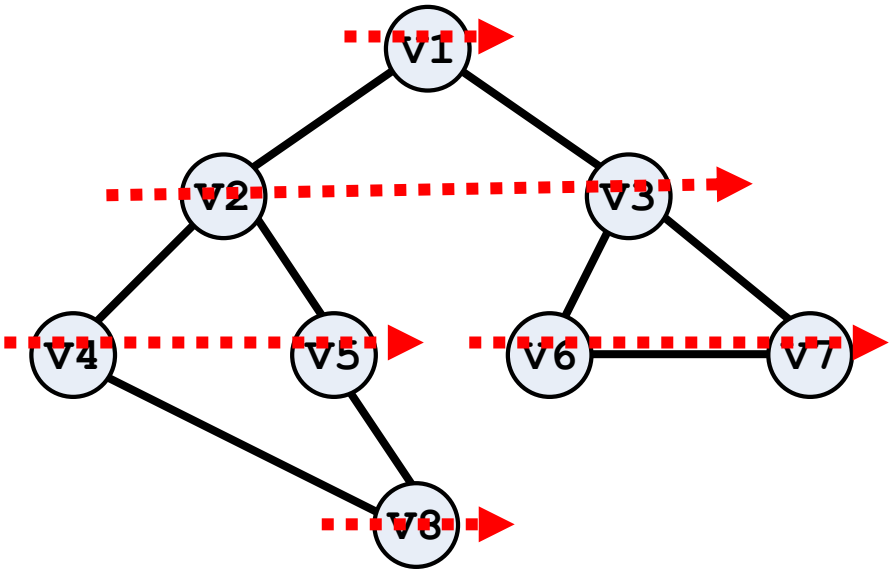
V1 V3 V7 V6 V2 V5 V8 V4

1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	0	0	0	0

图的广度优先遍历

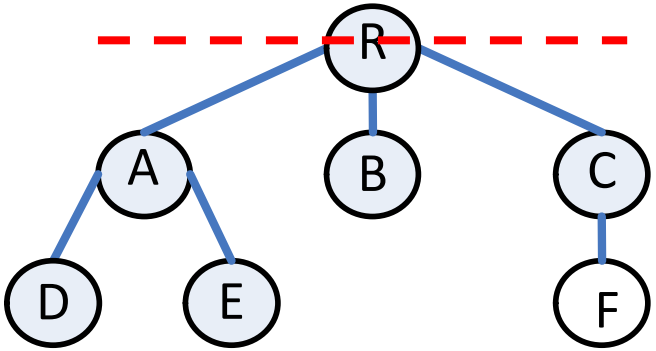
图的广度优先遍历

- 类似于树的广度优先(层次)遍历



V1->V2->V3->V4->V5->V6->V7->V8

回顾树的层次遍历

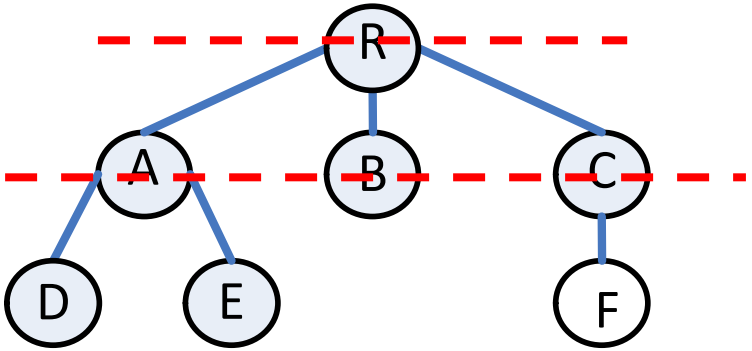


先进先出：

R

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历



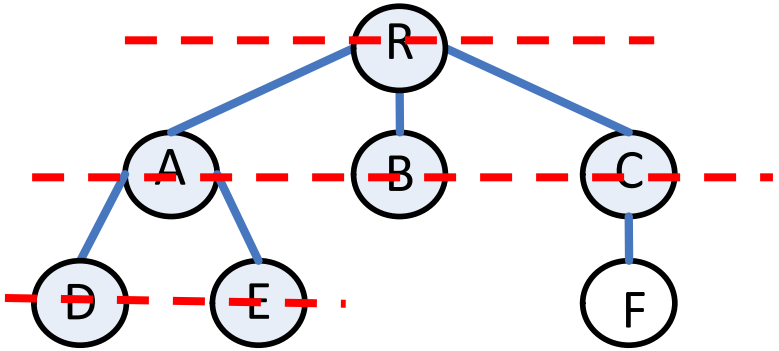
先进先出：

R

A B C

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历



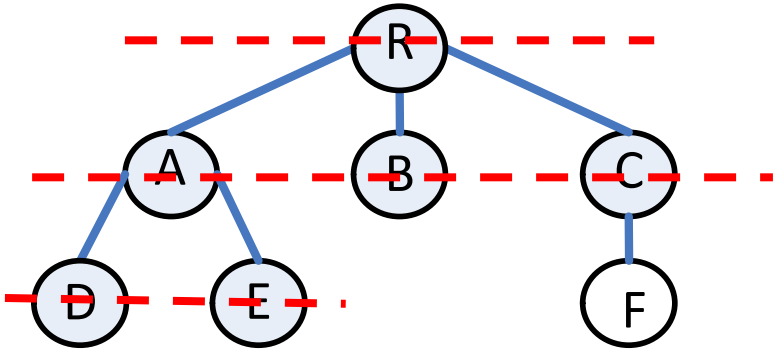
先进先出：

R A

B C D E

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历



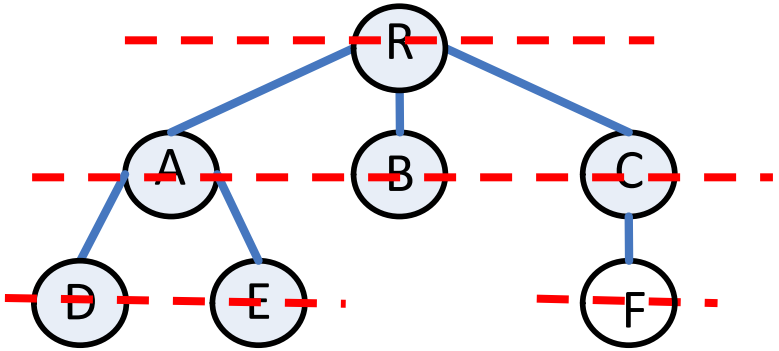
先进先出：

R A B

C D E

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历



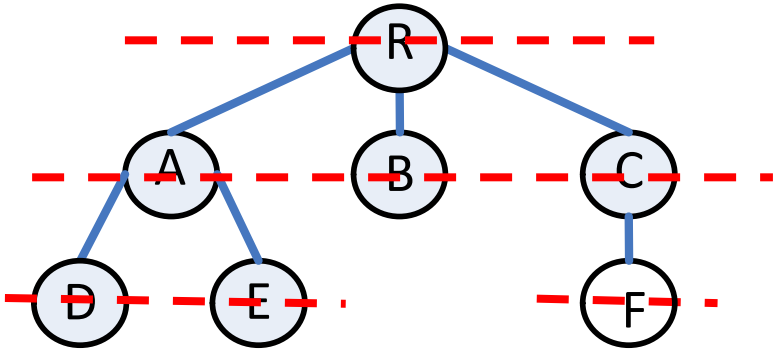
先进先出：

R A B C

D E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历

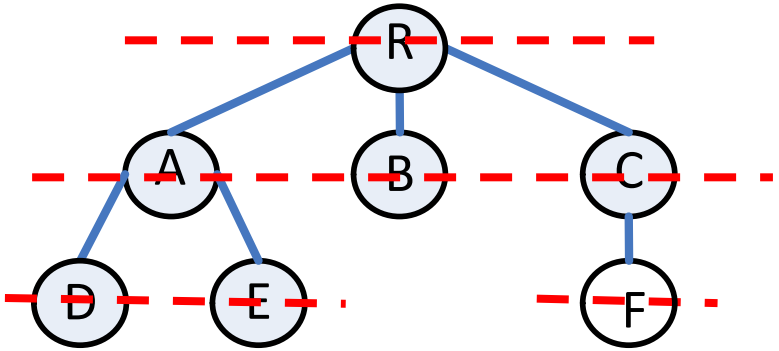


先进先出：

R A B C D　　E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历

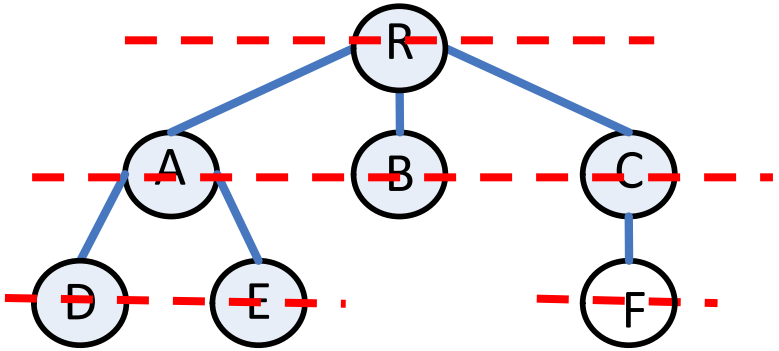


先进先出：

R A B C D E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

回顾树的层次遍历



先进先出：

R A B C D E F

```
BFS(R){
    R入队;
    while(队列不空){
        队头v出队并访问;
        v的孩子依次入队;
    }
}
```

BFS(v)

```
queue<int> Q;  //初始化队列
Q.push(v0); flag[v0] = 1; //入队并做标记
while( !Q.empty()){  //队列不空
   v = Q.front();Q.pop();    //出队一个顶点
   visit(v);                 //访问它
   for(v的未被访问邻接点w){
       Q.push(w); flag[w] = 1; //入队并做标记
   }
}
```

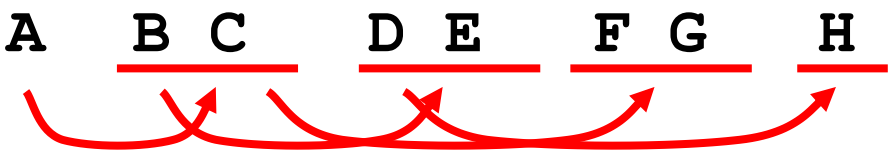
BFS(G)

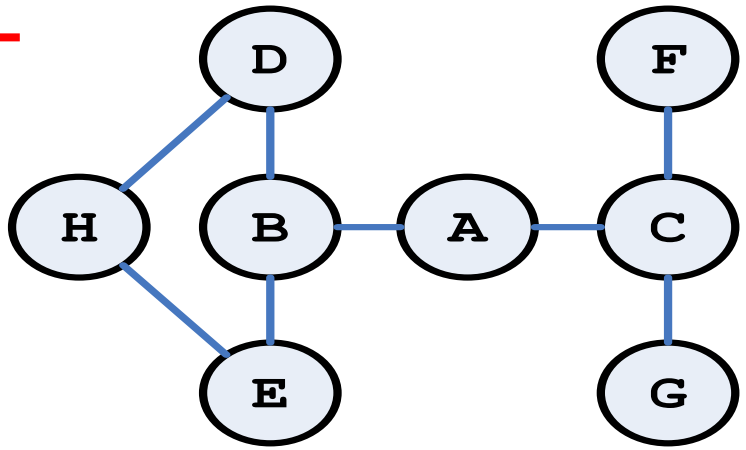
```
for(每个顶点v) marks[v] = false;
for(未访问的顶点v){
    BFS(v);
}
```

从顶点v出发的BFS

练习

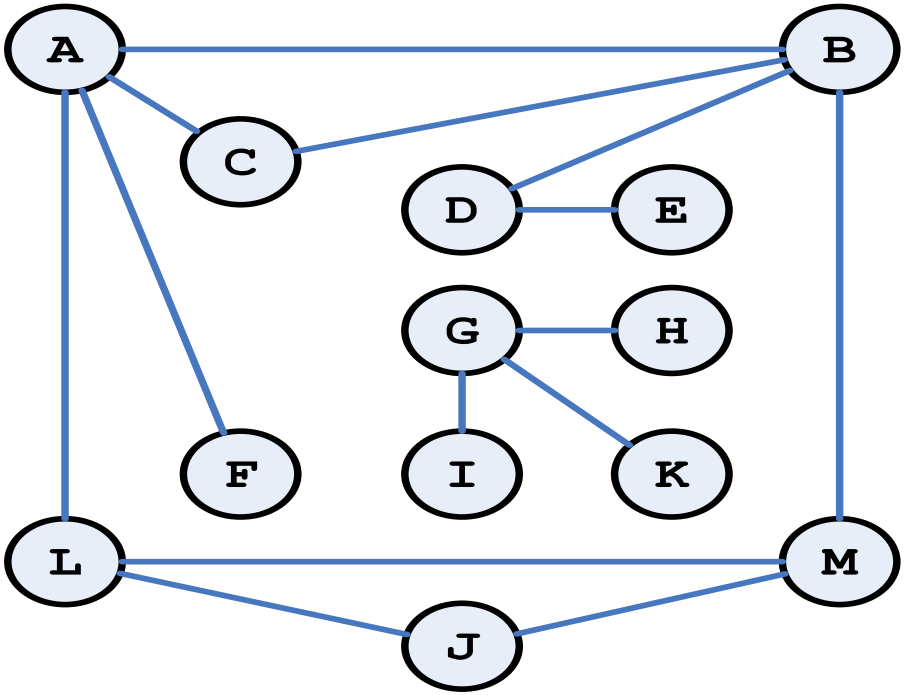
写出对下图进行广度优先遍历的结果





图的遍历：广度优先遍历

写出对下图进行深度优先遍历和广度优先遍历的结果



关注我

https://hwdong-net.github.io

Youtube频道:hwdong