《算法设计与分析》

第二章 图与遍历算法

马丙鹏

2023年09月18日

中国科学院大学
University of Chinese Academy of Sciences

第二章 图与遍历算法

- 2.1 图的基本概念和性质
- 2.2 图的遍历算法
- 2.3 双联通图与网络可靠性
- 2.4 对策树

2.2 图的遍历算法

■ 1. 检索与遍历

□检索：

- ➢以某种方法检查给定的数据对象，找出满足某些给定性质的结点的过程称为检索

□遍历：

- ➢当检索过程必须检索到数据对象的每一个结点时，则该检索过程称为遍历

□访问结点：

- ➢当算法对一个结点的信息段进行处理时，称该结点被访问。

2.2 图的遍历算法

■ 2. 二元树遍历

□遍历次序

- 在二元树的遍历中，以D、L、R分别代表访问结点的信息段、访问左子树、访问右子树。则可能的顺序有：

- **LDR**：中根次序遍历(中根遍历)
- **LRD**：后根次序遍历(后根遍历)
- **DLR**：先根次序遍历(先根遍历)

（先左后右）

- **RDL**：逆中根次序遍历
- **RLD**：逆后根次序遍历
- **DRL**：逆先根次序遍历

（先右后左）

2.2 图的遍历算法

■ 2. 二元树遍历

□中根次序遍历

算法2.1 中根次序遍历的递归表示

```
procedure INORDER(T)
//T是一棵二元树。T的每个结点有三个信息
 段:LCHILD, DATA, RCHILD//
   if T≠0 then
      call INORDER(LCHILD(T))
      call VISIT(T)
      call INORDER(RCHILD(T))
   endif
end INORDER
```

2.2 图的遍历算法

■ 2. 二元树遍历

□先根次序遍历

算法2.2 先根次序遍历的递归表示

```
procedure PREORDER(T)
//T是一棵二元树。T的每个结点有三个信息段:
    LCHILD, DATA, RCHILD//
  if T≠0 then
    call VISIT(T)
    call PREORDER(LCHILD(T))
    call PREORDER(RCHILD(T))
  endif
end PREORDER
```

2.2 图的遍历算法

■ 2. 二元树遍历

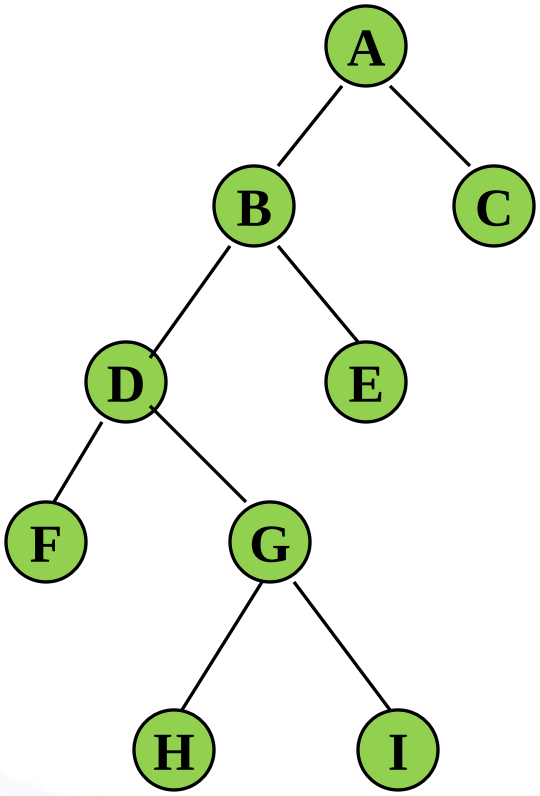
□后根次序遍历

算法2.3 后根次序遍历的递归表示

```
procedure POSTORDER(T)
//T是一棵二元树。T的每个结点有三个信息段:
   LCHILD, DATA, RCHILD//
  if T≠0 then
     call POSTORDER(LCHILD(T))
     call POSTORDER(RCHILD(T))
     call VISIT(T)
  endif
end PREORDER
```

2.2 图的遍历算法

■ 2. 二元树遍历



左图中：

中根次序遍历的输出是：

FDHGIBEAC

先根次序遍历的输出是：

ABDFGHIEC

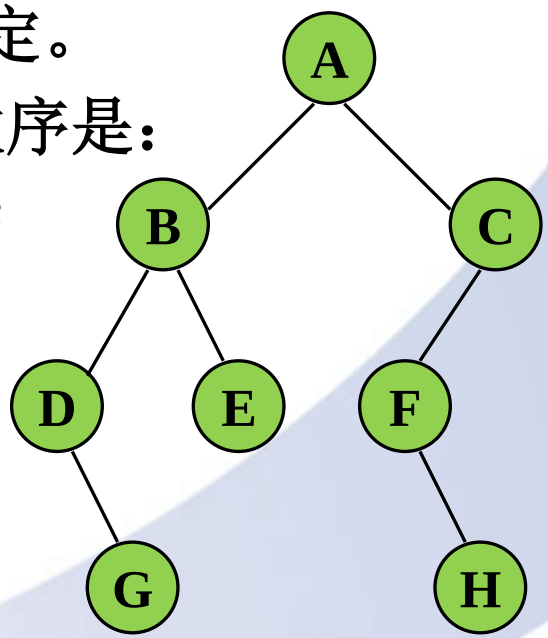
后根次序遍历的输出是：

FHIGDEBCA

2.2 图的遍历算法

■ 2. 二元树遍历

- □一棵二元树可由中根遍历序列+先根遍历序列、或中根遍历序列+后根遍历序列唯一确定。但不能由先根遍历序列+后根遍历序列唯一确定。
- □如已知一棵二元树的中根遍历次序是：**DGBEAFHC**，先根遍历次序是：**ABDGECFH**
  则这棵二元树唯一确定如下：



以A为根：左：DGBE，右：FHC

以B为根：左：DG，右：E

以D为根：左Φ，右：G

以G为根：左Φ，右：Φ

2.2 图的遍历算法

定理2.1 当输入的树T有$n\geq 0$个结点时，设t(n)和s(n)分别表示这些遍历算法中的任意一个算法所需要的最大时间和空间。如果访问一个结点所需要的时间和空间是$\Theta(1)$，则$t(n)=\Theta(n)$，$s(n)=\Theta(n)$。

证明：

时间：由于已知访问一个结点所需要的时间是$\Theta(1)$，故可用常数c_1限界。

设T的左子树中的结点数是n_1，则t(n)有：

$$t(n)=\max_{n1}\{t(n_1)+t(n-n_1-1)+c_1\}\ n\geq 1$$

其中，$t(0)\leq c_1$。

归纳法证明$t(n)\leq c_2 n+c_1$，其中c_2是一使得$c_2\geq 2c_1$的常数。

2.2 图的遍历算法

1)当n=0时，成立

2)假定当n=0, 1, …, m-1时均成立。则当n=m时有，

设T是一棵有n个结点的树，T左子树结点数为n_1,则

$t(n)=max_{n1}\{t(n_1) + t(n-n_1-1) + c_1\}$

$\leq max_{n1}\{c_2n_1 + c_1 + c_2(n-n_1-1) + c_1 + c_1\}$

$=max_{n1}\{c_2n + 3c_1 - c_2\}$

$\leq c_2n + c_1$

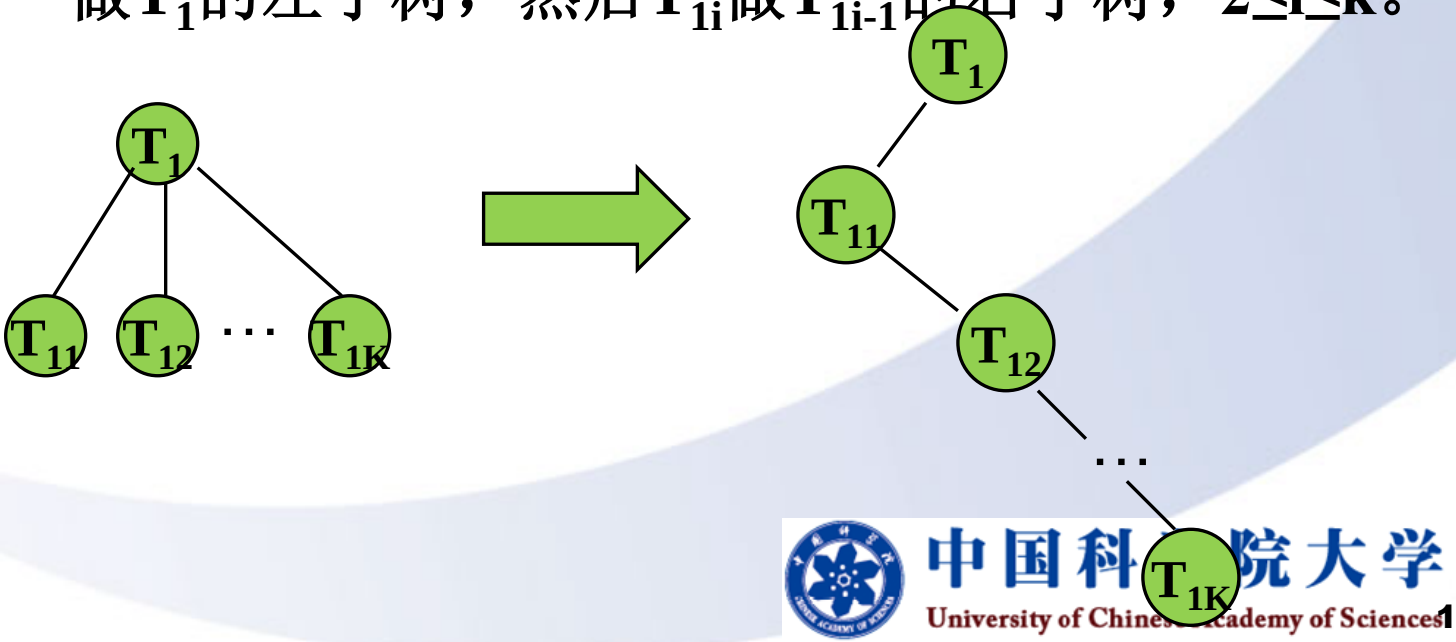
同理，存在c'_2和c'_1有$t(n)\geq c'_2n + c'_1$。所以$t(n)=\Theta(n)$

空间：若T的深度为d，则所需空间为$\Theta(d)$，$d\leq n$，所以$s(n)=\Theta(n)$。

2.2 图的遍历算法

■ 3. 树的遍历

- □树的子树顺序：无序→有序
- □树转换成二元树方法：
  - ➢ 设有一棵树T(它的根是T_1)，人为安排它的子树有序且设为$T_{11},T_{12},\ldots,T_{1K}$。用$T_1$做二元树的根，$T_{11}$做$T_1$的左子树，然后$T_{1i}$做$T_{1i-1}$的右子树，$2\le i\le k$。



2.2 图的遍历算法

■ 3. 树的遍历

- 设F是一个森林
- 树的先根次序遍历
  - 若F为空，则返回
  - 访问F的第一棵树的根
  - 按树先根次序遍历F的第一棵树的子树
  - 按树先根次序遍历F的其它树
- 树的中根次序遍历
  - 若F为空，则返回
  - 按树中根次序遍历F的第一棵树的子树
  - 访问F的第一棵树的根
  - 按树中根次序遍历F的其它树

2.2 图的遍历算法

■ 3. 树的遍历

□树的后根次序遍历

- ➢若F为空，则返回
- ➢按树后根次序遍历F的第一棵树的子树
- ➢按树后根次序遍历F的其它树
- ➢访问F的第一棵树的根

□森林F的遍历：设T是由森林F转换成的二元树，则：

- ➢T的先根次序遍历相当于按树先根次序遍历访问F
- ➢T的中根次序遍历相当于按树中根次序遍历访问F
- ➢对T的后根次序遍历无类似的自然对应

2.2 图的遍历算法

■ 4. 图的宽度优先检索

① 从结点v开始，给v标上已到达(或访问)标记——此时称结点v还没有被检测，而当算法访问了邻接于某结点的所有结点时，称该结点被检测了。

② 访问邻接于v且尚未被访问的所有结点——这些结点是新的未被检测的结点。将这些结点依次放置到一未检测结点表(队列Q)中(末端插入)。

③ 标记v已被检测。

④ 若未检测结点表为空，则算法终止；否则

⑤ 从未检测结点表的表头取一结点作为下一个待检测结点，重复上述过程。

算法终止时，Q为空。

算法2.6 宽度优先检索算法

```
procedure BFS(v)
 //宽度优先检索G，它从结点v开始。所有已访问结点被标记为VISITED(i)=1。//
    VISITED(v)←1; u←v   //VISITED(n)是一个标示数组，初始值
                                VISITED(i)=0, 1≤i≤n //
    将Q初始化为空          //Q是未检测结点的队列//
    loop
      for 邻接于u的所有结点w do
        if VISITED(w)=0 then     //w未被检测//
            call ADDQ(w, Q)      //ADDQ将w加入到队列Q的末端//
            VISITED(w)←1         //同时标示w已被访问//
        endif
      repeat
      if Q 为空 then return endif
      call DELETEQ(u, Q)    //DELETEQ取出队列Q的表头，并赋给变量u//
    repeat
end BFS
```

2.2 图的遍历算法

■ 4. 图的宽度优先检索

例：

检测结点1:

Visited(1) =1、Visited(2)=1、 Visited(3)=1

队列状态：

2	3	

检测结点2(结点2出队列):

Visited(4) =1、Visited(5)=1

队列状态：

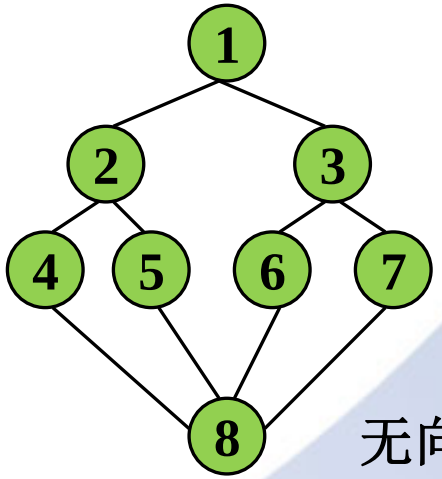
3	4	5	

检测结点3(结点3出队列):

Visited(6) =1、Visited(7)=1

队列状态：

4	5	6	7	



无向图G

2.2 图的遍历算法

■ 4. 图的宽度优先检索

检测结点4(结点4出队列): Visited(8) =1

队列状态：

5	6	7	8	

检测结点5(结点5出队列):

队列状态：

6	7	8	

检测结点6(结点6出队列):

队列状态：

7	8	

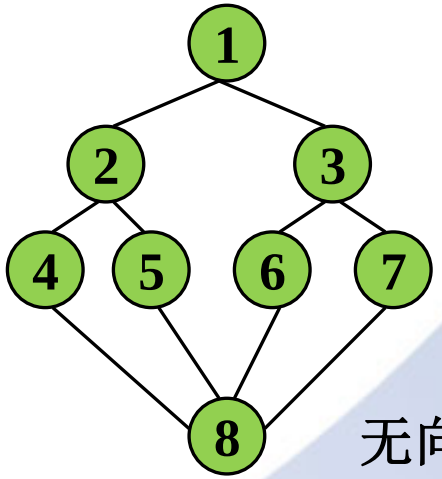
检测结点7(结点7出队列):

队列状态：

8	

检测结点8(结点8出队列):

队列状态：



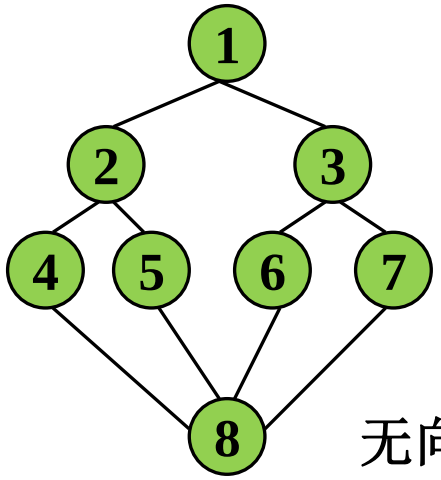
无向图G

2.2 图的遍历算法

■ 4. 图的宽度优先检索

□BFS的结点访问序列：

1 2 3 4 5 6 7 8



无向图G

2.2 图的遍历算法

■ 4. 图的宽度优先检索

定理2.2 算法BFS可以访问由v可到达的所有结点

证明：设G=(V, E)是一个(有向或无向)图，v∈V。

归纳法证明定理结论正确。

记d(v, w)是由v到达某一可到达结点w(w∈V)的最短路径长度。

⑴ 若d(v, w)≤1，则显然这样的所有w都将被访问。

⑵ 假设对所有d(v, w)≤r的结点都可被访问。则当d(v, w)=r+1时有：

设w是V中d(v, w)=r+1的一个结点

2.2 图的遍历算法

■ 4. 图的宽度优先检索

设u是从v到w的最短路径上紧挨着w的前一个结点。则有

$d(v, u)=r$。

所以，u可通过BFS被访问到。

假设$u \neq v$, 且$r \geq 1$。根据BFS的处理规则，

u将在被访问之前的某个时刻被放到未被检测结点队列Q上，而在另一时刻u将从队列Q中移出。

此时，所有邻接于u且尚未被访问的结点将被访问。

若结点w在这之前未被访问，则此刻将被访问到。

由上，定理得证。

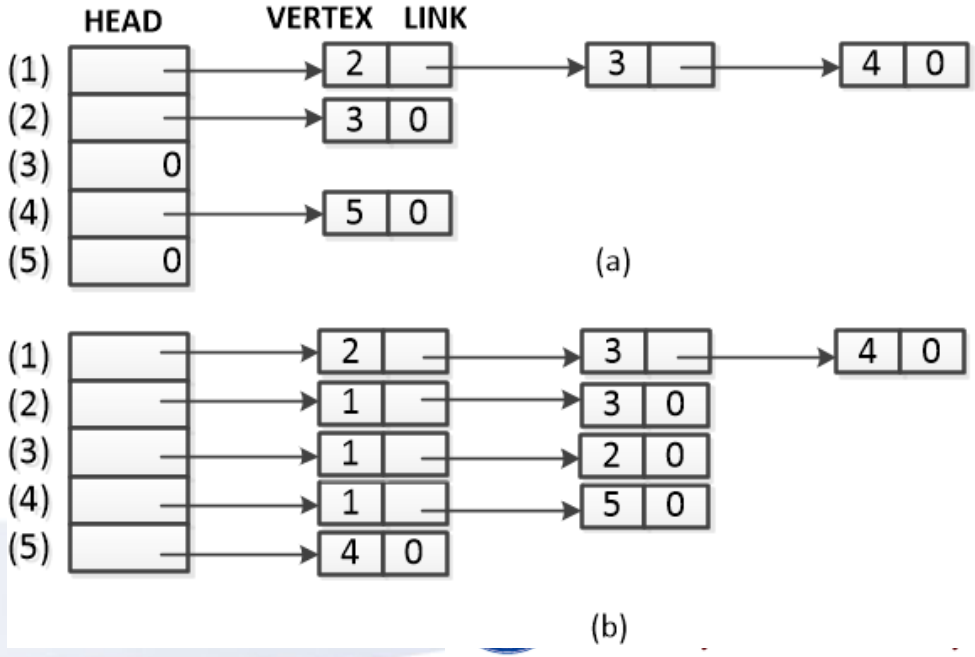
2.2 图的遍历算法

■ 4. 图的宽度优先检索

定理2.3 设t(n, e)和s(n, e)是算法BFS在任一具有n个结点和e条边的图G上所花的最大时间和最大附加空间。

- 若G由邻接表表示，则$t(n, e)=\Theta(n+e)$和$s(n, e)=\Theta(n)$。
- 若G由邻接矩阵表示，则$t(n, e)=\Theta(n^2)$和$s(n, e)=\Theta(n)$。

	1	2	3	4	5
(1)	0	1	1	1	0
(2)	0	0	1	0	0
(3)	0	0	0	0	0
(4)	0	0	0	0	1
(5)	0	0	0	0	0



2.2 图的遍历算法

■ 4. 图的宽度优先检索

空间分析：根据算法的处理规则，结点v不会放到队列Q中。结点w，w∈V且w≠v,仅在VISITED(w)=0时由ADDQ(w, Q)加入队列，并置VISITED(w)=1，所以每个结点(除v)至多只有一次被放入队列Q中。

至多有n-1个这样的结点考虑，故总共至多做n-1次结点加入队列的操作。需要的队列空间至多是n-1。所以s(n, e)=O(n)(其余变量所需的空间为O(1))

当G是一个具有v与其余的n-1个结点相连的图，则邻接于v的全部n-1个结点都将在“同一时刻”被放在队列上(Q至少应有Ω(n)的空间)。

同时，数组VISITED(n)本身需要Θ(n) 的空间。

所以s(n, e)=Θ(n)——这一结论与使用邻接表或邻接矩阵无关。

2.2 图的遍历算法

■ 4. 图的宽度优先检索

□时间分析: G采用邻接表表示

- ➢判断邻接于u的结点将在d(u)时间内完成：
  - ✓若G是无向图，则d(u)是u的度；
  - ✓若G是有向图，则d(u)是u的出度。
- ➢所有结点的处理时间：O(Σd(u))=O(e)。
  注：嵌套循环中对G中的每一个结点至多考虑一次。
- ➢VISITED数组的初始化时间：O(n)
- ➢算法总时间：O(n+e)。

2.2 图的遍历算法

■ 4. 图的宽度优先检索

- □时间分析: G采用邻接矩阵表示
  - ➢判断邻接于u的所有结点需要的时间：$\Theta(n)$
  - ➢所有结点的处理时间：$O(n)$
  - ➢算法总时间：$O(n^2)$
- □时间分析: 如果G是一个由v可到达所有结点的图，则将检测到V中的所有结点，所以上两种情况所需的总时间至少应是$\Omega(n+e)$和$\Omega(n^2)$。
  所以，$t(n, e)=\Theta(n+e)$ 使用邻接表表示
  或， $t(n, e)=\Theta(n^2)$ 使用邻接矩阵表示

2.2 图的遍历算法

■ 4. 图的宽度优先遍历

算法2.7图的宽度优先遍历算法

```
procedure BFT(G, n)
    //G的宽度优先遍历//
    int VISITED(n)
    for i←1 to n do VISITED(i)←0 repeat
    for i←1 to n do  //反复调用BFS//
        if VISITED(i)=0 then call BFS(i)  endif
    repeat
end BFT
```

注：若G是无向连通图或强连通有向图，则一次调用BFS即可完成对T的遍历。

否则，需要多次调用。

2.2 图的遍历算法

■ 4. 图的宽度优先遍历

□图遍历算法的应用

- ➢判定图G的连通性：
  - ✓若调用BFS的次数多于1次，则G为非连通的。
- ➢生成图G的连通分图：
  - ✓一次调用BFS中可以访问到的所有结点及连接这些结点的边构成一个连通分图。

2.2 图的遍历算法

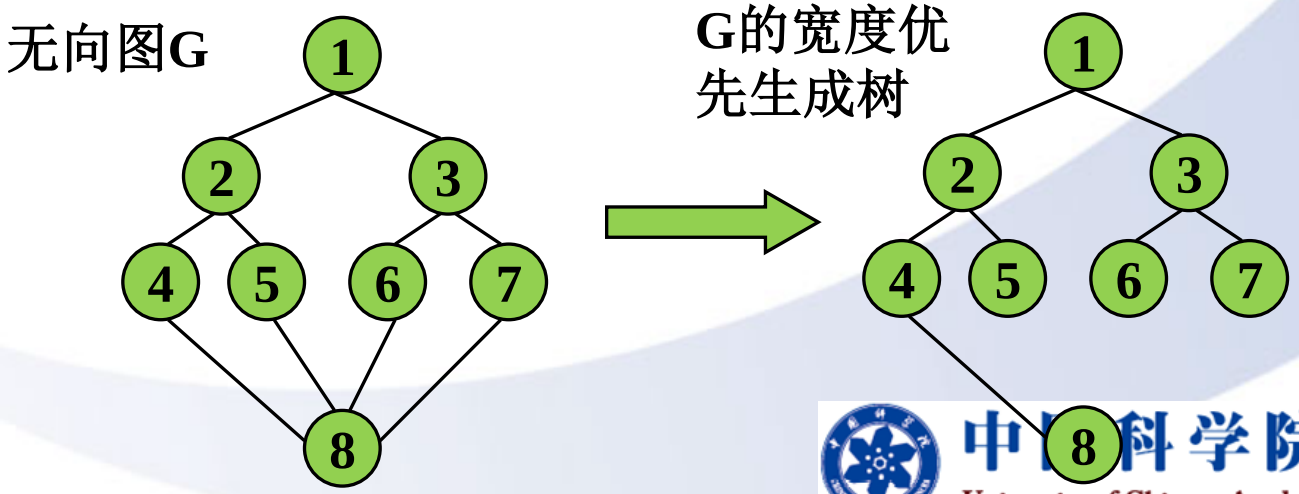
■ 4. 图的宽度优先遍历

□图遍历算法的应用

- 宽度优先生成树

向前边：BFS中由u达到未访问结点w的边(u, w)称为向前边。记T是BFS中所处理的所有向前边集合。

宽度优先生成树：若G是连通图，则BFS终止时，T构成一棵生成树。



2.2 图的遍历算法

■ 4. 图的宽度优先遍历

□图遍历算法的应用

➢宽度优先生成树

定理2.4 修改算法BFS，在第1行和第6行分别增加语句T←Φ和T←T∪{(u, w)}。修改后的算法称为BFS*。若v是无向图中任一结点，调用BFS*，算法终止时，T中的边组成G的一棵生成树。

```
procedure BFS*(v)
    VISITED(v)←1;u←v
    T←Φ
    将Q初始化为空
    loop
      for 邻接于u的所有结点w do
        if VISITED(w)=0 then     //w未被检测//
          T←T∪{(u, w)}
          call ADDQ(w, Q)        //ADDQ将w加入到队列Q的末端//
          VISITED(w)←1           //同时标示w已被访问//
        endif
      repeat
      if Q 为空 then return endif
      call DELETEQ(u, Q)  //DELETEQ取出队列Q的表头，并赋给变量u//
    repeat
end BFS*
```

2.2 图的遍历算法

证明：

若G是n个结点的连通图，则这n个结点都要被访问。除起始点v以外，其它n-1个结点都将被放且仅将被放到队列Q上一次，从而T将正好包含n-1条边，且这些边是各不相同的。即T是关于n个结点n-1边的无向图。

同时，对于连通图G，T将包含由起始结点v到其它结点的路径，所以T是连通的。

则T是G的一棵生成树。

注：对于n个结点且正好有n-1条边的连通图是一棵树。

2.2 图的遍历算法

■ 5. 图的深度优先检索

- 从结点v开始，首先给v标上已到达(或访问)标记，同时中止对v的检测，
- 开始对邻接于v且尚未被访问的结点u检测，
- 在这样的u均被检测后，再恢复对v的检测，
- 当所有可到达的结点全部被检测完毕后，算法终止。

2.2 图的遍历算法

■ 5. 图的深度优先检索

算法2.8 图的深度优先检索

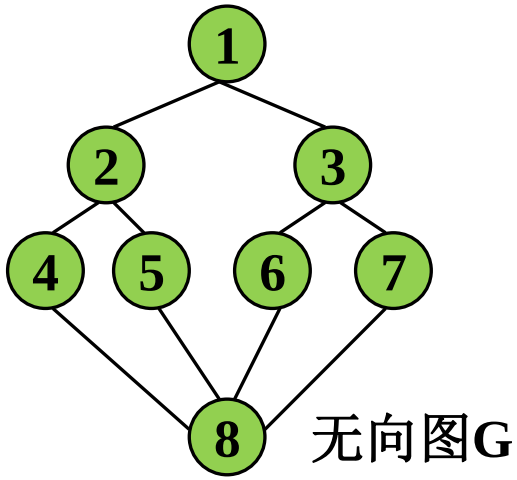
```
procedure DFS(v)
//已知一个n结点的无向(或有向)图G=(V,E)
      以及初值已置为零的数组 VISITED(1:n)。
      这个算法访问由v可以到达的所有结点。//
   VISITED(v)←1
   for 邻接于v的每个结点w do
       if VISITED(w)=0 then
           call DFS(w)
        endif
   repeat
end DFS
```

2.2 图的遍历算法

■ 5. 图的深度优先检索

例：DFS结点访问序列：

1→2 →4 →8 →5 →6 →3 →7



无向图G

2.2 图的遍历算法

■ 5. 图的深度优先检索

□性质：

① DFS可以访问由v可到达的所有结点

② 如果t(n, e)和s(n, e)表示DFS对一n结点e条边的图所花的最大时间和最大附加空间，则

✓$s(n, e)=\Theta(n)$

✓$t(n, e)= \Theta(n+e)$　G采用邻接表表示，或

✓$t(n, e)= \Theta(n^2)$　G采用邻接矩阵表示

2.2 图的遍历算法

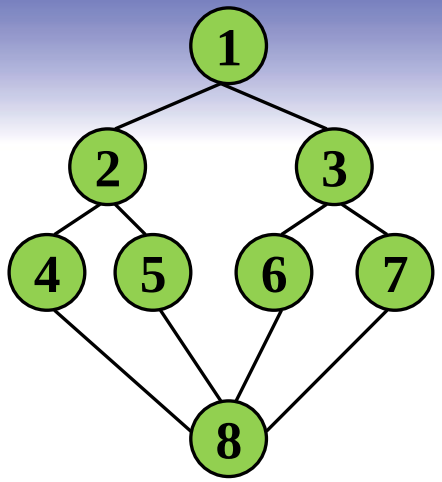
- 5. 图的深度优先检索
  - 深度优先遍历算法DFT
  - 反复调用DFS,直到所有结点均被检测到。
  - 应用：
    - 判定图G的连通性
    - 连通分图
    - 深度优先生成树

2.2 图的遍历算法

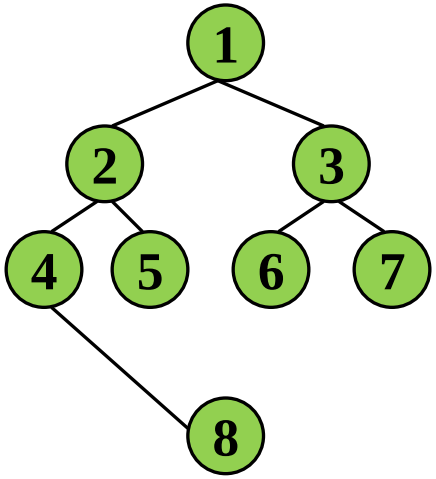
■ 5. 图的深度优先检索

深度优先生成树算法

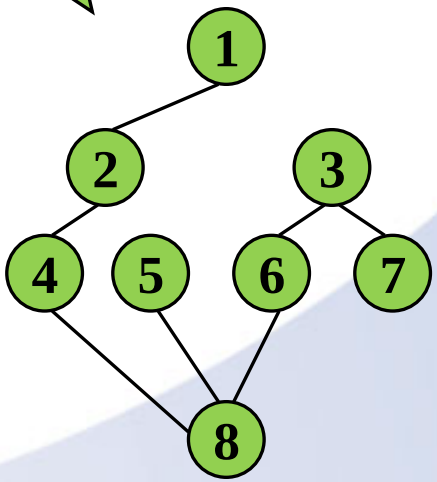
```
procedure DFS*(v)
    T←Φ
    VISITED(v)←1
    for 邻接于v的每个结点w do
        if VISITED(w)＝0 then
            T←T∪{(u, w)}
            call DFS(w)
        endif
    repeat
end DFS*
```



无向图G



G的宽度优先生成树



G的深度优先生成树

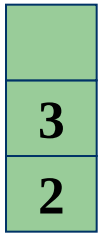
2.2 图的遍历算法

■ 6. 图的D_Search检索

- □改造BFS算法，用栈来保存未被检测的结点，则得到的新检索算法称为深度检索(D_Search)算法。
- □ 注：结点被压入栈中后将以相反的次序出栈，并进行新的检测。

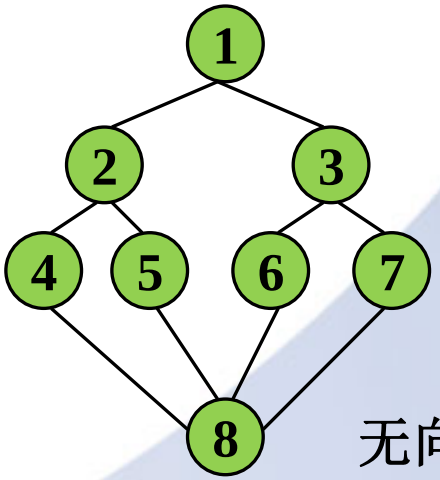
例：D_Search的检索过程

检测结点1:
visited(1) =1、Visited(2)=1、 Visited(3)=1
栈状态：



检测结点3(结点3出栈):
visited(6) =1、Visited(7)=1
栈状态：

检测结点7(结点7出栈):
visited(8) =1
栈状态：



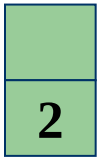
无向图G

检测结点8(结点8出栈):
visited(4) =1、 visited(5) =1
栈状态:

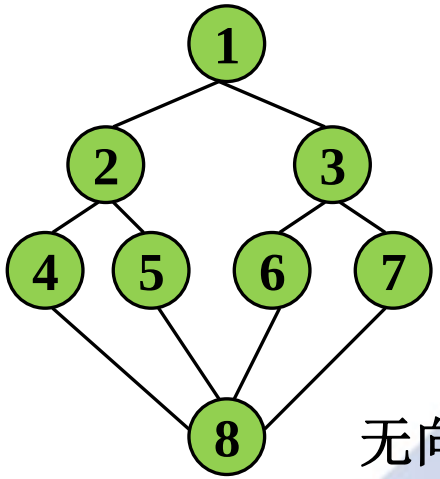
检测结点5(结点5出栈):
栈状态:

检测结点4(结点4出栈):
栈状态:

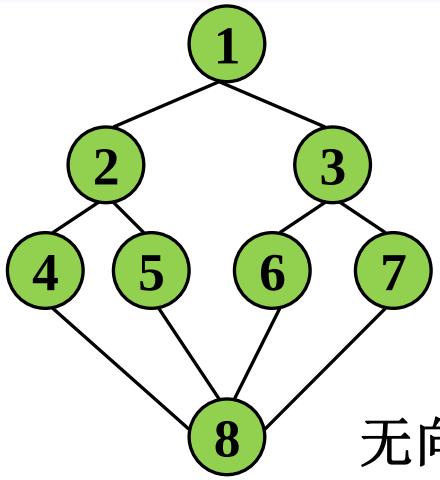
检测结点6(结点6出栈):
栈状态:



检测结点2(结点2出栈):
栈状态:



无向图G



无向图G

D_Search的结点检测序列：

1 3 7 8 5 4 6 2

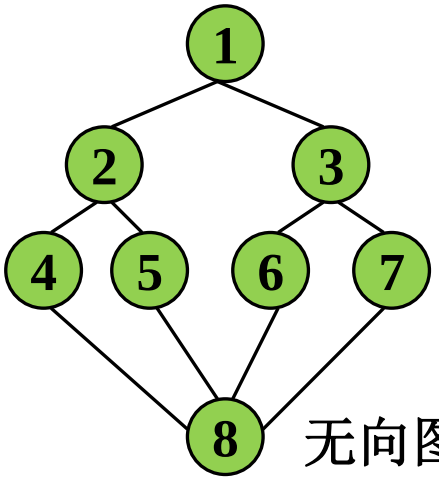
D_Search的结点访问序列是什么？

2.2 图的遍历算法

- **7. BFS、DFS、D_Search算法比较**
  - **BFS**：使用队列保存未被检测的结点。结点按照宽度优先的次序被访问和进、出队列。
  - **DFS**：使用栈保存未被检测的结点，结点按照深度优先的次序被访问并依次被压入栈中，并以相反的次序出栈进行新的检测。
  - **D_Search**：使用栈保存未被检测的结点，结点按照宽度优先的次序被访问并被依次压入栈中，然后以相反的次序出栈进行新的检测。新的检测结点是最新被访问但未被检测的结点。

2.2 图的遍历算法

■ 7. BFS、DFS、D_Search算法比较



无向图G

BFS检测序列： 1 2 3 4 5 6 7 8

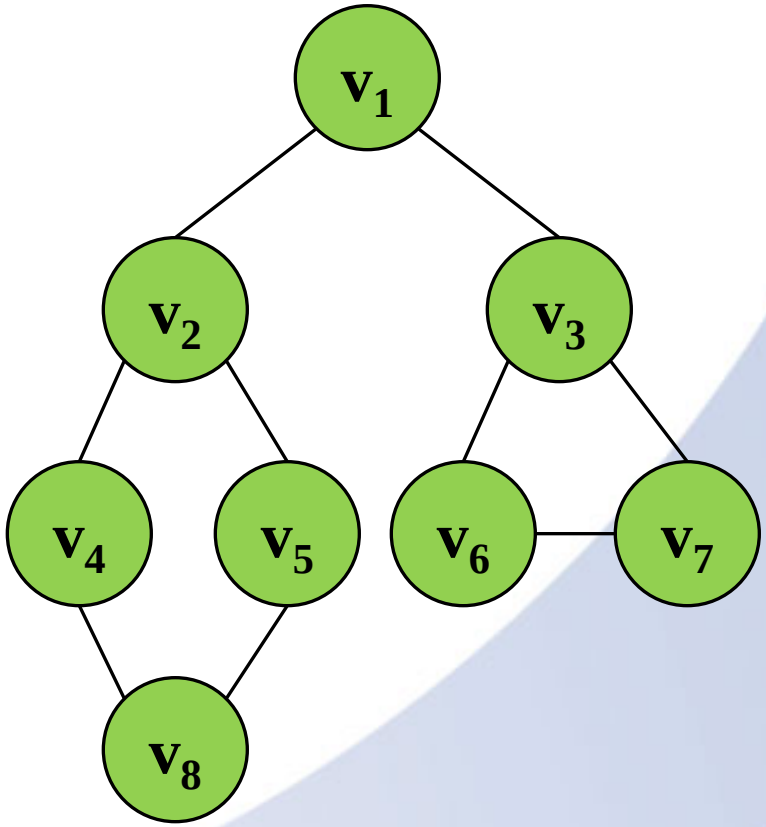
DFS访问序列： 1 2 4 8 5 6 3 7

D_Search检测序列： 1 3 7 8 5 4 6 2

D_Search访问序列： 1 2 3 6 7 8 4 5

作业-课后练习2

- 描述
  - 以V_1为起点，写出BFS的结点检测顺序
  - 以V_2为起点，写出DFS的结点访问顺序
  - 以V_8为起点，写出D_Search的结点检测顺序和结点访问顺序



End