算法设计与分析作业四

作者：吴润泽　　**学号：**181860109

Email：181860109@smail.nju.edu.cn

2020 年 4 月 14 日

目录

Chapter 4

problem 4.2

(1)

$\Rightarrow$ 如果 w 是 v 在 DFS 树中的后继结点，那么 $actice(w) \subseteq active(v)$: 当 $w \neq v$ 时，因为 w 是 v 的后继结点，所以 $v.discover < w.discover$, 并且 $v.finish > w.finish$。所以 $actice(w) \subset active(v)$。当 $w = v$ 时显然成立。$\Leftarrow$ 如果 $actice(w) \subseteq active(v)$，那么 w 是 v 在 DFS 树中的后继结点: 当 $w \neq v$ 时，因为 $active(w) \subseteq active(v)$，即 $v.discover < w.discover$，并且 $v.finish > w.finish$，即在遍历 v 的过程中将 w 遍历，即 w 是 v 的后继结点。

(2)

由 (1) 可知，w 不是 v 的后继结点 $\Leftrightarrow actice(w) \not\subset active(v)$。v 不是 w 的后继结点 $\Leftrightarrow actice(v) \not\subset active(w)$ 得证。

(3)

① $\Rightarrow$ 如果 vw 是 CE，那么 v 和 w 没有祖先和后继关系，由 (2) 可知 active(w) 和 active(v) 互不包含。同时 CE 说明在 v 指向 w 时，w 已经是黑色结点，w 已经遍历结束，所以 active(w) 在 active(v) 之前。

$\Leftarrow$ active(w) 在 active(v) 之前，w 先完成整个遍历过程，后才遍历到 v。且二者没有祖先后继关系，那么边 vw 即为 CE。

② $\Rightarrow$vw 是 DE，即 v 指向 w 时 w 为黑色，并且 $active(w) \subset active(v)$，若不存在第三个结点 x，满足 x 是 v 的后继，w 是 x 的后继，则 v 遍历到 w 时 w 一定为白色，边为 TE。即一定有 $active(w) \subset active(x) \subset active(v)$。

$\Leftarrow$ 如果存在结点 x，满足 $active(w) \subset active(x) \subset active(v)$，由 (1) 可知，x 是 v 的后继，w 是 x 的后继，且在遍历时 v 先走到 x，然后 x 走到 w，即 v 是 w 的祖先结点，因此 vw 是 DE。

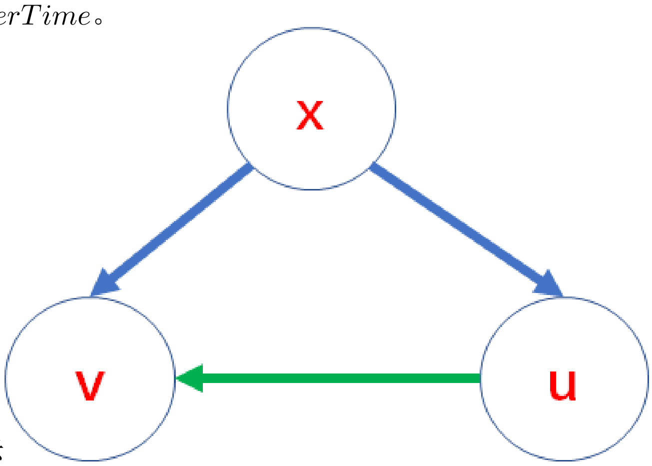
③ $\Rightarrow$vw 是 TE，即 v 指向 w 时 w 为白色，w 是 v 的后继，由 (1) 可知，$active(w) \subset active(v)$。若存在 x，满足 $active(w) \subset active(x) \subset active(v)$，则在遍历时 v 先走到 x，然后 x 走到 w，那么 v 是 w 的祖先而非父结点，与 vw 是 TE 矛盾。

$\Leftarrow$ 同理可得 w 是 v 的后继，且 v 直接指向 w，则 w 是白色，即 vw 是 TE。

④ vw 是 BE$\Leftrightarrow$v 是 w 的后继 $\Leftrightarrow active(v) \subset active(w)$ 得证。

problem 4.5

1. 不可能是 TE。如果是 TE，则有 $active(v) \subset actice(u)$，即 $v.finishTime > u.discoverTime$，故不成立。
2. 不可能是 BE。如果是 BE，则有 $active(u) \subset actice(v)$，即 $v.finishTime > u.discoverTime$，故不成立。
3. 不可能是 DE。如果是 DE，同样的 v 是 u 的后继结点，满足 $active(v) \subset actice(u)$，同 1. 不成立。
4. 可能是 CE。x 结点先遍历 v，然后从 v 返回 x，x 遍历 u，易知 $v.finishTime < u.discoverTime$。



如图所示

problem 4.7

在第一次 DFS 中，将结点压栈，同一强连通片的源头结点是最后一个压栈的。**(引理 4.4)** 若 l 是某个强连通片首结点，x 是另一个强连通片中的结点，并且存在 l 通向 x 的路径，则 x 比 l 先结束遍历，即 x 先进栈 l 后进栈。满足这些性质，才能保证第二次 DFS 中，按正确的顺序取出每个 SCC 的首结点。

无论是 DFS 还是 BFS 在一次遍历中都可以访问一个或者多个强连通片的所有结点。

如果第一次 DFS 换为 BFS，由于 BFS 中按层序遍历，同一连通片中出度不为 0 的点可能先入栈。在第二次 DFS 的时候，不能有正确的访问顺序。

如果第二次 DFS 换为 BFS，由于出栈的访问首结点顺序正确，BFS 同样可以正确地划分强连通片。

因此第一次必须为 DFS，第二次 DFS 和 BFS 都可以。

problem 4.8

充要条件: 对于无向连通图的 DFS 生成树的根结点 v，v 是割点，当且仅当 v 有两个及两个以上的子树。

证明: $\Rightarrow$ 如果 v 是割点，假设 v 只有一个子树，易知子树是连通的，将 v 删除，剩下的部分为 v 的子树仍然连通，这与 v 是割点相矛盾。因此 v 有两个及两个以上的子树。

$\Leftarrow$ 如果 v 有两个及两个以上的子树，因为图本身连通，则子树之间相连必然通过 v，即 v 是割点。

problem 4.9

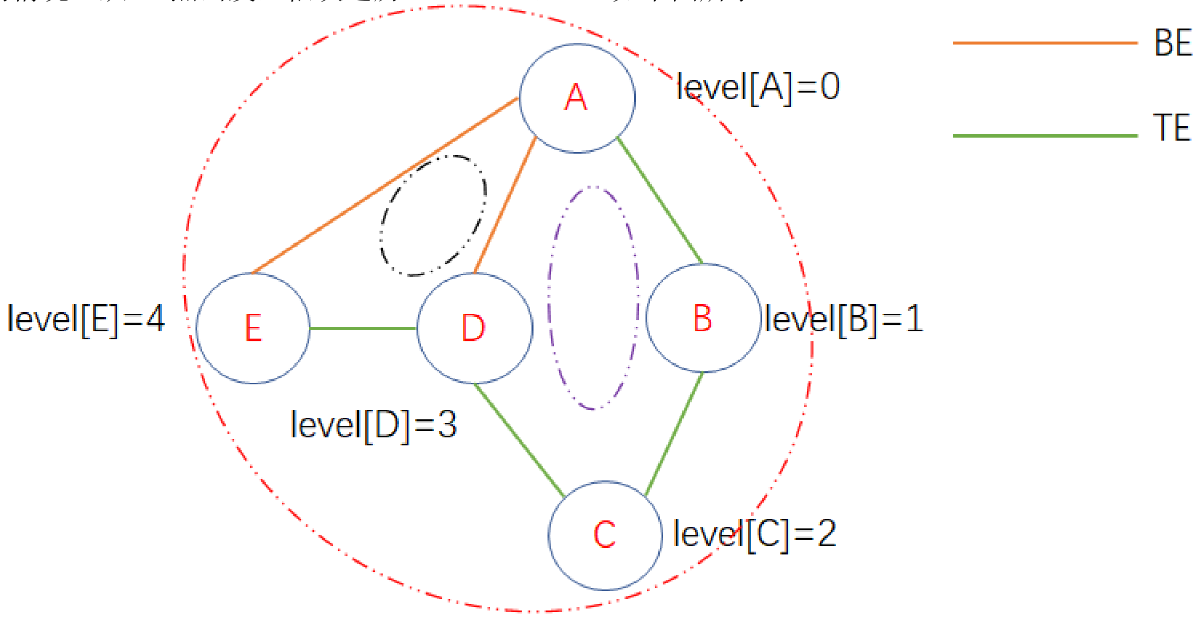
正确

证明: 当从 TE vw 回退时，如果以 w 为根的子树存在 BE 指向 v 的祖先，则 v 的祖先的 discoverTime 会被赋值给以 w 为根的子树中某个结点的 back 值，并最终会传递到 w.back，即必有 $w.back < v.discoverTime$；

否则，没有存在 BE 指向 v 的祖先，如果子树不存在 BE，则子树所有点 back 值均为初始值，即 $w.back > v.discoverTime$，如果子树中存在 BE，则子树中的所有结点的 back 值也将大于等于 $v.discoverTime$，因为 BE 只能指向 v 或者子树内部结点，故 back 值最小也不会低于 $v.discoverTime$，仍然满足 $w.back > v.discoverTime$，即仍能正确判断是否为割点。

problem 4.12

算法中，默认环为多条 TE 和 BE 构成，但存在两条 BE 和 TE 构成环的情况。从 A 点出发，依次遍历 B，C，D，E。如下图所示。



图中的 BE 为 AE 和 AD，根据题目给定算法，计算出外圈大环的大小为 5 和右侧环大小为 4，最终结果为 4。但图中最小环大小应为 3，为左侧小环。因此算法不正确。

problem 4.13

如果有向图没有环，则至少有一个点的入度为 0。即无向图 DFS 生成树必须存在环。

算法设计 1. 有向图中每个点的入度至少为 1，则有图的边数不小于图的点数，则无向图中必定有环（存在 BE）。

2. 利用 DFS 找到一条 BE（找到一个环），设边为 uv，u 为后继，v 为祖先。（如果没有找到 BE，说明存在一棵 DFS 生成树无环，则不能构造成功）

3. 以 v 为起点进行 DFS，将每条 TE xy，标记方向为 x->y，最后将 uv 边标记方向为 u->v，即满足了该环中每个点的入度都大于 0。

4. 找到图中仍为白色的点，进行 2、3 步，若没有算法结束。

5. 由于只进行了两次深度优先遍历，较易得算法时间复杂度为线性。

具体算法实现见AddDirection 算法

算法 1 *AddDirection* 算法

```
Function FindLoop (u)\\找到一个环路，并返回 BE 边祖先结点
    u.color := GRAY, res := None
    foreach neighbor v of u do
        if v.color = WHITE and v.vis = False then
            FindLoop(v)
        elif uv = BE then
            res := u, break
    return res
```

```
Function AddDir (u)\\给环路加方向
    u.vis = True\\标记其已经加边避免重复
    foreach neighbor v of u do
        if v.color = WHITE and v.vis = False then
            change uv dircetion to u → v
            AddDir(v)
        elif uv = BE then
            change uv dircetion to u → v
```

```
Function Main − AddDirection (V, E)\\wrapper 部分
    foreach point v in V do
        v.color := WHITE, v.vis := False
    foreach point v in V do
        if v.color = WHITE and v.vis = False then
            res := FindLoop(v)
            if res = None then return False
            else AddDir(res)
    return True
```

problem 4.14

算法设计 因为 G 是有向无环图，则 G 一定存在拓扑排序（**引理 4.2**）。

1. 入度最小的点（即拓扑排序的队首）a，如果图中存在哈密顿路径，则 a 的入度一定为 0。假设 a 的入度不为 0，结点 x 指向 a。由于 a 是拓扑排序队首，故一定存在 a 到达 x 的路径（因为有图连通）设为 $a \to \cdots \to x$，同时 x 指向 a，构成环路，与题干相矛盾。故 a 的入度一定为 0。
2. 显然，从 a 点出发，如果存在哈密顿路径，则一次 DFS 遍历所有结点。
3. 首先找到图 G 的入度为 0 的点 a，如果不存在或存在多个则不可能存在哈密顿通路。
4. 从 a 开始 DFS，每当递归回溯时（边为 uv），检查哈密顿路径 path 长度是否为结点个数。
5. 如果等于结点个数，说明存在路径 $a \to \cdots u \to v \to \cdots n$，算法结束。
6. 否则，将 $v.vis$ 置为假，将 v 从 path 中剔除（哈密顿通路上 u 不直接到达 v），从 u 的下一子结点继续 DFS。
7. 找最小入度结点为 $O(m+n)$，DFS 过程显然是 $O(m+n)$，因此时间复杂度是线性的。

具体算法实现见FindHamilton 算法

算法 2 *FindHamilton* 算法

```
Function FindHamilton (u, path, n)\\找到一个环路，并返回 BE 边祖先结点
    u.vis := True, path.push(u)
    foreach neighbor v of u do
        if v.vis = False then
            FindHamilton(v, path, n)
            if len(path) == n then return True
            v.vis := False, path.pop()\\没有找到，故将 vis 置为假，将 v 从 path 剔除
    return False\\u 所有子结点都没有找到哈密顿通路
```

Function $Main - FindHamilton$ (V, E)\\wrapper 部分

  foreach *point u in V* **do**

    $u.vis := False, u.cnt := 0$\\入度初始化为 0

    foreach *neighbor v of u* **do** $u.cnt + +$\\计算各点的入度

  $tar := None, cnt := 0$\\初始化要寻找的点

  foreach *point u in V* **do**\\找到唯一一个入度为 0 的点

    if $tar! = None$ **and** $cnt = 0$ **and** $u.cnt = 0$

      then return $False$\\存在不止一个入度为 0 的点，不可能存在哈密顿通路

    if $cnt \geq u.cnt$ **then** $tar := u, cnt := u.cnt$

  if $tar = None$ **or** $cnt! = 0$ **then return** $False$\\不存在入度为 0 的点

  return $FindHamilton(tar)$

problem 4.16

证明 有向无环图中必有入度为 0 的点。

设图有 N 个结点，假设每个点的入度均不为 0，必有 $A_2 \to A_1, A_3 \to A_2, \cdots, A_n \to A_{n-1}$，而对于 A_n 入度不为 0，故存在某个结点指向它，则出现环路，产生矛盾，得证。

算法设计

1. 有向无环图可以出现一个或者多个入度为 0 的结点，对这些结点的拓扑先后顺序没有要求。

2. 入度为 0 的结点的出边删去，必会出现入度为 0 的点（仍为无环图）。

3. 重复 **1，2** 步，直到算法结束。可知时间复杂度为线性。

由开始的**证明**可知，图中存在回路，会出现没有入度为 0 的点的情况，无法确定拓扑顺序。

算法实现

算法 3 $ZeroTopoLogical$ 算法

Function $InitQueue$ $(G(V, E), queue)$\\找到一个环路，并返回 BE 边祖先结点

  foreach *point u in V* **do**

    if $InEdge[u] = 0$ **then**

      $queue.push(u)$

  return $len(queue) > 0$\\没有找到入度为 0 的点，返回 False

```
Function Main − ZeroTopoLogical (G(V, E))\\找到一个环路，并返回 BE 边祖先结点
    queue.init(), topo := list(), count := 0
    /*queue 存放入度为 0 的结点的队列,topo 存放拓扑排序结果,count 为排好序的结点数*/
    if InitQueue(queue) = False then returnFalse
    while queue.empty()! = False do
        u = queue.pop(), topo[count + +] = u\\确定 u 为当前的逻辑起点
        foreach neighbor v of u do
            InEdge[v] − −
            if v.vis = False then queue.push(v)
    return count = len(V)\\所有结点应确定拓扑排序，否则存在环路
```

problem 4.17

(1)

易知只需要以顶点 s 为起点进行一次 DFS，判断是否遍历所有点即可。代价为 $O(m+n)$。

```
Function OneToAll (G(V, E), s)
    DFS(s)
    foreach point u in V do
        if u.color = WHITE then return False
    return True
```

(2)

利用课本上的划分强连通片的算法，将有向图 G 改造为 G 的收缩图 G↓，各点之间的方向转换为各连通片之间的方向，易知 G↓ 为有向无环图。

题目可以转换为一个连通片可以到达其他所有连通片。由于 G↓ 为有向无环图，必有入度为 0 的连通片。故如果图 G 存在到达所有顶点的点，其必在该连通片中。再利用(1)算法判断该连通片能否到达所有连通片即可。

强连通片划分与 DFS 判断是否可达算法时间复杂度均为 $O(m+n)$，因此时间复杂度为 $O(m+n)$，符合题意。

具体算法实现见SccOneToAll 算法

算法 4 SccOneToAll 算法

Function $SccOneToAll$ $(G(V,E))$

  $construct\ G\downarrow\ using\ Scc(G)$

  $point := None, CountZero := 0$\\\\记录入度为 0 的连通片个数

  foreach $point\ u\ in\ V\downarrow$ **do**

    if $InEdge[u] = 0$ **then**

      $point := u, CountZero := CountZero + 1$

  if $point = None$ **or** $CountZero > 1$ **then**

    return $False$\\\\没有或多个入度为 0 的点，返回 False

  return $OneToAll(G\downarrow, point)$

problem 4.18

(1)

同样利用课本上的划分强连通片的算法，将有向图 G 改造为 G 的收缩图 G↓。易知同一个连通片中的影响力值相同。易得影响力最低的连通片，一定是出度为 0 的连通片（一定存在，可能只有一个连通片），影响力是连通片中结点个数减 1。因此，找到所有出度为 0 的连通片，并找出结点个数最小的即可。

SCC 的划分和寻找出度为 0 的连通片并统计的时间复杂度为 $O(m+n)$

算法 5 MinImpact 算法

Function $MinImpact$ $(G(V,E))$

  $construct\ G\downarrow\ using\ Scc(G)$

  $MinPoint := None, MinCount := \infty$\\\\记录连通片的最小个数

  foreach $point\ u\ in\ V\downarrow$ **do**

    if $OutEdge[u] = 0$ **and** $MinImpact < number(u)$ **then**

      $MinPoint := u, MinCount := number(u)$

  return $(MinCount - 1, MinPoint)$\\\\结点个数减 1 为影响力, 以及对应的连通片

(2)

同样得到有向图 G 的收缩图 G↓，易知影响力最大的连通片，一定是入度为 0 的连通片（压缩图无环）。

则对所有的入度为 0 的连通片进行 DFS 遍历，遍历到的连通片即为其可到达的，DFS 结束后计算遍历到连通片的结点个数。取所有 DFS 得到结点个数最多的即为所求。

由于每次 DFS 后，都要查找所有访问过的连通片，因此时间复杂度为 $O(n(m+n))$。

算法实现

算法 6 MaxImpact 算法

Function $MinImpact$ $(G(V,E))$

 $construct\ G\downarrow\ using\ Scc(G)$

 $MaxPoint := None, CurCount := 0, MaxCount := 0$

 /*记录最大的 DFS 生成树的根以及结点个数*/

 foreach $point\ u\ in\ V\downarrow$ **do**

  if $InEdge[u] = 0$ **then**

   $DFS(G\downarrow, u), CurCount := 0$\\\\更新当前 DFS 树的结点个数

   foreach $point\ x\ in\ V\downarrow$ **do**

    if $x.vis = True$ **then**

     $CurCount := CurCount + number(x)$

     $x.vis := False$\\\\将访问置为 False，避免下次 DFS 错误计算

   if $MaxCount < CurCount$ **then**

    $MaxPoint := u, MaxCount := CurCount$

 return $(MaxCount - 1, MaxPoint)$\\\\返回最大影响力, 以及对应的连通片

problem 4.20

与problem 4.16 类似，每学期可以修所有入度为 0 的结点。然后将所有入度为 0 的结点的指出的边删去，假设本题有解即为有向无环图，一定再次出现入度为 0 的结点，重复上述过程即可修完所有课程，重复的次数即为所需要的最少学期数。可得算法时间复杂度是线性 $O(m+n)$。

算法实现

算法 3 $ZeroTopoLogical$ 算法

Function $InitQueue$ $(G(V,E), queue)$\\找到一个环路，并返回 BE 边祖先结点
    foreach $point\ u\ in\ V$ **do**
        if $InEdge[u] = 0$ **and** $u.vis = False$ **then**
            $queue.push(u)$
    return $len(queue) > 0$\\没有找到入度为 0 的点，返回 False

Function $Main - ZeroTopoLogical$ $(G(V,E))$\\找到一个环路，并返回 BE 边祖先结点
    $queue.init(), count := 0, TermCount := 0$
    /*queue 存放入度为 0 的结点的队列,count 为排好序的结点数,TermCount 为学期数*/
    while $count < len(V)$ **do**\\安排的课程数还少于总课程
        if $InitQueue(queue) = False$ **then return** $False$
        while $queue.empty()! = False$ **do**
            $u = queue.pop()$
            $u.vis = True$
            $count := count + 1$\\已经确定的课程安排数加 1
            foreach $neighbor\ v\ of\ u$ **do**
                $InEdge[v] - -$
        $TermCount := TermCount + 1$\\所有零入度课程均已安排，学期数加 1
    return $TermCount$\\最少学期数

problem 4.22

problem 4.23

Chapter 5

problem 5.1

problem 5.2

problem 5.4

problem 5.8

problem 5.9

problem 5.10