Tree-Search Algorithms

March 30, 2010

1 Tree-Search

- Let G be a graph. If G has a spanning tree T, then G is connected.
- Let T be a tree of G. If $V(T) = V(G)$, then G is connected. If $V(T) \subsetneq V(G)$, then either $[T, G-T] = \emptyset$ or $[T, G-T] \neq \emptyset$. In the former case, G is disconnected; in the latter case, for any edge $e \in [T, G-T]$ with end-vertices $u \in T, v \in T^c$, the subgraph $T \cup e$ is again a tree of G.
- Using the above idea, one may generate a sequence of trees in G, starting with the trivial tree consisting of a single vertex v_0, and terminating either with a spanning tree of G or with a non-spanning tree T with $[T, G-T] = \emptyset$. The procedure is called a **tree-search**, and the resulting tree is called a **search tree**.
- Let (T, v_0) be a rooted tree of G. Let $P = v_0 v_1 \cdots v_l$ be the unique path in T from v_0 to a vertex $v (= v_l)$. Each vertex v_i of P, including v itself, is called an **ancestor** of v in T, and v is called a **descendant** of v_i in T. The vertex $u (= v_{l-1})$ is called the **predecessor** (or **parent**) of v, denoted $p(v)$, and v is called a **successor** (or **child**) of u.
- There are two typical tree-searches: **Breath-first search** (BFS) and **depth-first search** (DFS).

Theorem 1.1 (Breath-First Search Tree)**.** INPUT: a connected graph $G = (V, E)$ with a specified vertex v_0.

OUTPUT: a rooted tree (T, v_0) with the root v_0, a vertex sequence $P = v_0 v_1 \cdots v_n$ with $n = |V|$, an index function $\mathrm{ind} : V \to \mathbb{N}$, a parent function $p : V - \{v_0\} \to V$, and a level function $\ell : V \to \mathbb{N}$ such that $\ell(v) = d_G(v_0, v)$ for all $v \in V$.

STEP 1: *Start with a vertex sequence $Q := v_0$, a root tree (T, v_0) consisting of the single vertex v_0, $\ell(v_0, v_0) := 0$, and a vertex sequence $P := \emptyset$.*

STEP 2: *If $Q = \emptyset$,* STOP.
If $Q \neq \emptyset$, delete the initial vertex of Q, say, u, add u to the end of P, and go to STEP 3.

STEP 3: *If there are vertices $w_1, w_2, \ldots, w_k \in G-T$ adjacent with u by edges $e_1, e_2, \ldots, e_k$ respectively, add the edges $e_1, e_2, \ldots, e_k$ to T, the sequence $w_1 w_2 \cdots w_k$ to the end of Q, set* $\mathrm{ind}(w_i) := \mathrm{ind}(u) + i$, *$\ell(w_i) := \ell(u) + 1$, $p(w_i) := u$, where $1 \leq i \leq k$, and return to* STEP 2.
If there are no vertices of $G - T$ adjacent with u, return to STEP 2.

Proof. Initially, $Q = v_0 \neq \emptyset$. It is clear that in any step the subgraph T is a tree. The vertices of T can be ordered as a sequence PQ, where P is a subsequence having no vertices adjacent with a vertex of $G - T$. Since G is connected, if $V(T) = V$, then T is a spanning tree of G.

Now, if $V(T) \subsetneq V$, there are always vertices of T adjacent with some vertices of $G - T$. Let u be the first vertex in the sequence PQ that joins a vertex of $G - T$, and u is the initial vertex of Q; we then enter into STEP 3. (Thus the algorithm continues, and finally stops when T is a spanning tree.) Note that $w_i \notin Q$, $w_i \notin T$, and $\ell(v) = d_T(v_0, v) \geq d_G(v_0, v)$ for all $v \in T$.

Now, we need only to show that $\ell(w_i) = d_G(v_0, w_i)$. We may assume that $\ell(v) = d_G(v_0, v)$ for all $v \in T$. Since $\ell(u) = d_G(v_0, u)$, then $d_G(v_0, w_i) \leq \ell(u) + 1 = \ell(w_i)$. Let $P_i = v_0 v_1 \cdots v_d$ be a shortest path in G from v_0 to $v_d (= w_i)$, and let v_j be the last vertex of P_i such that $v_j \in T$, $v_{j+1} \notin T$. Of course, $j \leq d - 1$. We claim that $\ell(v_j) \geq \ell(u)$. (Otherwise, if $\ell(v_j) < \ell(u)$, then by Lemma 1.2, v_j enters Q before u. Subsequently, v_j leaves Q before u, and at the time that v_j leaves Q, the vertex v_{j+1} enters Q, i.e., $v_{j+1} \in T$ at the time that u leaves Q. This is a contradiction.) Thus

$$d \geq j + 1 = d_G(v_0, v_j) + 1 = \ell(v_j) + 1 \geq \ell(u) + 1 = \ell(w_i).$$

Therefore $d = d_G(v_0, w_i) = \ell(w_i)$. $\square$

Lemma 1.2. *Let T be a BFS-tree of a connected graph G. Let Q be the queuing vertex sequence and u, v be two vertices.*

(a) If $\ell(u) < \ell(v)$, then u enters Q before v.

(b) If u enters Q before v, then $\ell(u) \le \ell(v)$.

(c) If u, v are end-vertices of an edge $e \notin T$, then $|\ell(u) - \ell(v)| \le 1$.

Proof. (a) We proceed by induction on $\ell(u)$. When $\ell(u) = 0$, then $u = v_0$ and v_0 enters Q before every other vertex of V. Assume it is true when $\ell(u) < l$, and consider the case $\ell(u) = l \ge 1$. Let x, y be parents of u, v respectively in the rooted tree (T, v_0). Then $\ell(x) = \ell(u) - 1$ and $\ell(y) = \ell(v) - 1$. Clearly, $\ell(x) < \ell(y)$. By induction, the vertex x enters Q before y; subsequently, x leaves Q before y. Now, note that u enters Q right after x leaves Q, and at that time the vertex v did not yet enter Q, for it is not yet the turn for y to leave Q. Thus u enters Q before v.

(b) is equivalent to (a).

(c) If $\ell(u) = \ell(v)$, nothing is to be proved. If $\ell(u) \ne \ell(v)$, we may assume $\ell(u) < \ell(v)$. Then u enters Q before v. At the time when u leaves Q, if $v \notin Q$, then $\ell(v) = \ell(u) + 1$ by definition of ℓ; if $v \in Q$, let y be the parent of v, then y enters Q before u by definition of Q, hence $\ell(y) \le \ell(u)$; since $\ell(u) < \ell(v) = \ell(y) + 1$, we must have $\ell(u) = \ell(y)$; hence $\ell(v) = \ell(u) + 1$. $\square$

Theorem 1.3 (Depth-First Search-Tree Algorithm)**.** Input: a connected graph $G = (V, E)$ with a specified vertex v_0.

Output: a rooted tree (T, v_0), a closed walk $W = v_0 e_1 v_1 \cdots e_{2n-2} v_{2n-2}$ with $n = |V|$, a multi-valued index function $\mathrm{ind} : V \to \mathbb{N}$, a parent function $p : V - \{v_0\} \to V$, a degree function $d : V \to \mathbb{N}$, and two time functions $f : V \to \mathbb{N}$, $l : V \to \mathbb{N}$ such that $f(v) \le l(v)$ for all $v \in V$.

Step 1: *Initialize a vertex variable x and a rooted tree (T, v_0) consisting of a single vertex v_0; assign v_0 to the variable x; set $W := v_0$, $\mathrm{ind}(x) := 0$, $f(v_0) := 0$; then go to* Step 2.

Step 2: *If there are edges joining x to some vertices of $G - T$, select an edge e joining x to a vertex $w \in G - T$; add ew to T and ew to the end of W; set $f(w) := \mathrm{ind}(x) + 1$, $p(w) := x$; assign w to x and set $\mathrm{ind}(x) := f(w)$; then return to* Step 2.

If there is no edge joining x to any vertex of $G - T$, set $l(x) := \mathrm{ind}(x)$; then go to Step 3.

Step 3: *If $x = v_0$, set $d(x) := \#\{i \mid v_i = x \text{ in } W\} - 1$; then* Stop.

If $x \ne v_0$, set $d(x) := \#\{i \mid v_i = x \text{ in } W\}$; backtrack from x to its parent u through an edge e in T; add the word eu to the end of W, set $\mathrm{ind}(u) := \mathrm{ind}(x) + 1$; assign u to x and set $\mathrm{ind}(x) := \mathrm{ind}(u)$; then return to Step 2.

Proof. It is clear that at any stage the constructed subgraph T is always a tree and the algorithm stops eventually. When the algorithm reaches the stage $l(v_0)$, we have $[v_0, G - T(v_0)] = \emptyset$. Then $[T_{v_0}(v_0), G - T(v_0)] = \emptyset$ by Lemma 1.4. Since $T_{v_0}(v_0) = T_{v_0}$, we have $[T(v_0), G - T(v_0)] = \emptyset$. Since G is connected, this means that $T(v_0)$ is a spanning tree of G. $\square$

The time functions f, l can be given by the walk W as follows:

$$f(u) = \min\{i \mid v = v_i \in W\}, \quad l(v) = \max\{i \mid v = v_i \in W\}, \quad v \in V.$$

Lemma 1.4. *Let W be a walk resulted by DFS-Tree Algorithm, having the end vertex v assigned to the variable x. Let $T(x)$ denote the rooted tree produced by W at stage $l(x)$, i.e., $[x, G - T(x)] = \emptyset$. For each vertex u of $T(x)$, let $T_u(x)$ denote the rooted subtree of $T(x)$ at u as the root. Then $[T_x(x), G - T(x)] = \emptyset$.*

Proof. We proceed by induction on the number of vertices of $T_x(x)$. It is true when $T_x(x)$ contains only the vertex x, i.e., when x has no children in $T(x)$. Let $w_1, w_2, \ldots, w_k$ be all children of x in $T(x)$, been added to W in its current order. To have the vertex variable x at the vertex v, it must be backtracked from w_i to v in the order $w_1, w_2, \ldots, w_k$. This means that $[w_i, G - T(w_i)] = \emptyset$. By induction, $[T_{w_i}(w_i), G - T(w_i)] = \emptyset$. Note that $T_{w_i}(x) = T_{w_i}(w_i)$ and $T(w_i) \subseteq T(x)$. Thus

$$[T_x(x), G - T(x)] = \bigcup_{i=1}^{k} [T_{w_i}(x), G - T(x)] \subseteq \bigcup_{i=1}^{k} [T_{w_i}(w_i), G - T(w_i)] = \emptyset.$$

□

Proposition 1.5. *Let (T, v_0) be a DFS-tree of a connected graph G. Let u, v be two vertices.*

(a) The vertex v is a descendant of u if and only if $f(u) < f(v) \leq l(v) < l(u)$.

(b) If u, v are end-vertices of an edge $e \notin T$, then u is an ancestor or a descendant of v in T.

(c) If $f(u) < f(v)$, then either $l(v) < l(u)$ or $l(u) < f(v)$.

Proof. (a) Let v be a descendant of u, i.e., $v \in T_u$. Then u enters W before v; so $f(u) < f(v)$. When the vertex variable x is at v, to reach u again, x must be backtracked from v; so $l(v) < l(u)$. Conversely, if $f(u) < f(v) \leq l(v) < l(u)$. Then v enters W after u. Suppose v is not a descendant of u, i.e., $v \notin T_u$. Then v enters W after all vertices of T_u, i.e., after T_u is finished. So $l(u) < f(v)$; this is a contradiction.

(b) We may assume $f(u) < f(v)$, i.e., v enters W after u. Suppose v is not a descendant of u, i.e., $v \notin T_u$. Then v enters W after all vertices of T_u. At stage $l(u)$, the tree $T(u)$ does not contain v, i.e., $v \notin T(u)$. This means that $e \in [u, G - T(u)] \neq \emptyset$; the subtree T_u is not yet finished. This is a contradiction.

(c) Note that under the given condition $f(u) < f(v)$, $v \in T_u$ if and only if $l(v) < l(u)$. If $l(v) < l(u)$ is not true, i.e., $v \notin T_u$, then v enters W after T_u is finished; so $l(u) < f(v)$. □

Corollary 1.6. *Let (T, v_0) be a DFS-tree of a connected graph G.*

(a) Any leaf of T cannot be a cut vertex of G.

(b) The root v_0 is a cut vertex of G if and only if v_0 has at least two children in T.

(c) A vertex v is a cut vertex of G if and only if v has a child w in T such that there is no edge between a proper ancestor of v to a descendant of w.

Proof. Trivially follows from Proposition 1.5(a). □

2 Minimum-Weight Spanning Tree

- Let $G = (V, E)$ a graph together with a **weight function** $w : E \to \mathbb{R}$ is called a **weighted graph**, denoted (G, w). For each $e \in E$, the value $w(e)$ is called the **weight** of e. The **weight** of G is the value
$$w(G) = \sum_{e \in E} w(e).$$
- A **minimum-weight spanning tree** (MST) of a weighted graph (G, w) is a spanning tree whose weight is minimum among all spanning trees of G.

Theorem 2.1 (Prim's Algorithm)**.** INPUT: a connected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$.
OUTPUT: a minimum-weight spanning tree T of G.
STEP 1: *Choose a vertex v of G, initialize a tree T consisting of the single vertex v; and go to* STEP 2.
STEP 2: *If $V(T) = V$,* STOP.
STEP 3: *If $V(T) \neq V$, choose an edge e from the cut $[T, G - T]$ such that $w(e)$ is minimum in $[T, G - T]$, add e to T, and go to* STEP 2.

Proof. It is clear that in STEP 3 the subgraph $T \cup e$, constructed by adding the edge e from $[T, G - T]$ to the tree T, is still a tree. Finally, the trees T grow up to a spanning tree when the algorithm STOPS. We are left to show that the produced spanning tree is optimal. It is enough to show that at any stage the tree T is contained in an optimal spanning tree of G. We proceed by induction on the number of edges of T.

Initially, the tree $T := v_0$ is obviously contained in an optimal spanning tree of G. Assume that in STEP 3 the tree T is contained in an optimal spanning tree T^*. Note that $w(e) \leq w(x)$ for all $x \in [T, G - T]$. If $e \notin T^*$, then $T^* \cup e$ contains a cycle C_e and $e \in C_e$. Since C_e intersects the cut $[T, G - T]$, there is an edge $e' \in C_e \cap [T, G - T]$ other than e. Clearly, the spanning tree $T^{**} := (T^* \cup e) \backslash e'$ contains $T \cup e$, and
$$w(T^{**}) = w(T^*) + w(e) - w(e') \leq w(T^*).$$
The optimality of T^* implies that $w(T^{**}) = w(T^*)$. Hence T^{**} is an optimal spanning tree which contains $T \cup e$. □

Theorem 2.2 (Kruskal's Algorithm)**.** INPUT: a connected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$, $|V| = n$.

OUTPUT: a minimum-weight spanning tree T of G.

STEP 1: *Choose an edge e of G such that $w(e)$ is minimum in E, initialize a subgraph T consisting of the single edge e, and go to* STEP 2.

STEP 2: *If $|E(T)| = n - 1$,* STOP.

STEP 3: *If $|E(T)| < n - 1$, choose an edge e from $E - E(T)$ such that $w(e)$ is minimum in $E - E(T)$ and $T \cup e$ contains no cycle, add e to T, and go to* STEP 2.

Proof. It is enough to show that at any stage the subgraph T is contained in an optimal spanning tree of G. We proceed by induction on the number of edges of T.

Initially, the subgraph $T := e_0$ contains the single edge e_0 whose weight is minimum in E. Let T^* be an optimal spanning tree of G. If $e_0 \notin T^*$, then $T^* \cup e_0$ contains a cycle C_{e_0}. Select an edge e_1 from C_{e_0} other than e_0; the spanning tree $T^{**} := (T^* \cup e_0) \backslash e_1$ contains e_0. Since $w(e_0) \le w(x)$ for all $x \in E$, we have

$$w(T^{**}) = w(T^*) + w(e_0) - w(e_1) \le w(T^*).$$

The optimality of T^* implies that $w(T^{**}) = w(T^*)$. So T^{**} is an optimal spanning tree of G and contains T.

Assume that in STEP 3 the subgraph T is contained in an optimal spanning tree T^*. Note that $w(e) \le w(x)$ for all $x \in E - E(T)$. If $e \notin T^*$, then $T^* \cup e$ contains a unique cycle C_e and $e \in C_e$. Since $T \cup e$ contains no cycle, the cycle C_e cannot be contained in $T \cup e$. Thus there exists an edge $e' \in C_e$ such that $e' \notin T$ and $e' \ne e$. Set $T^{**} := (T^* \cup e) \backslash e'$; then T^{**} contains $T \cup e$ and

$$w(T^{**}) = w(T^*) + w(e) - w(e') \le w(T^*).$$

The optimality of T^* implies that $w(T^{**}) = w(T^*)$. Hence T^{**} is an optimal spanning tree of G and contains $T \cup e$. □

3 Branching-Search

Theorem 3.1 (Dijkstra's Algorithm, Directed Breadth-First Search)**.** INPUT: a digraph $D = (V, A)$ with a specified vertex v_0 and a positive weight function $w : E \to \mathbb{R}_+$.

OUTPUT: a v_0-branching (T, v_0) in D, a vertex sequence $P = v_0 v_1 \cdots v_n$ with $n = |V(T)|$, an index function $\text{ind} : V(T) \to \mathbb{N}$, a level function $\ell : V(T) \to \mathbb{N}$ such that $\ell(v) = d_G(v_0, v)$ for all $v \in V(T)$, and a parent function $p : V(T) - \{v_0\} \to V$.

STEP 1: *Start with a vertex sequence $Q := v_0$, a v_0-branching (T, v_0) consisting of the single vertex v_0, a vertex sequence $P := \emptyset$, $\text{ind}(v_0) := 0$, $\ell(v_0, v_0) := 0$; and go to* STEP 2.

STEP 2: *If $Q = \emptyset$,* STOP.

If $Q \ne \emptyset$, delete the initial vertex of Q, say, u, add u to the end of P, and go to STEP 3.

STEP 3: *If $(u, D - T)$ has arcs $a_1, a_2, \ldots, a_k$ with heads $w_1, w_2, \ldots, w_k$ in $D - T$ respectively, add $a_1, a_2, \ldots, a_k$ to T, the sequence $w_1 w_2 \cdots w_k$ to the end of Q; set $\text{ind}(w_i) := \text{ind}(u) + i$, $\ell(w_i) := \ell(u) + 1$, $p(w_i) := u$, where $1 \le i \le k$; and go to* STEP 2.

If $(u, D - T) = \emptyset$, go to STEP 2.

Proof. Similar to the proof of Theorem 1.1. □

Theorem 3.2 (Directed Depth-First Search)**.** INPUT: *a digraph $D = (V, A)$.*

OUTPUT: *a spanning branching forest F of D with a root set R, a closed walk $W = v_0 e_1 v_1 \cdots e_{2n-r-1} v_{2n-r-1}$, where $n = |V|$ and $r = |R|$, a multi-valued index function* $\text{ind} : V \to \mathbb{N}$*, a parent function $p : V - R \to V$, a degree function $d : V \to \mathbb{N}$, and two time functions $f : V \to \mathbb{N}$, $l : V \to \mathbb{N}$ such that $f(v) \le l(v)$ for all $v \in V$.*

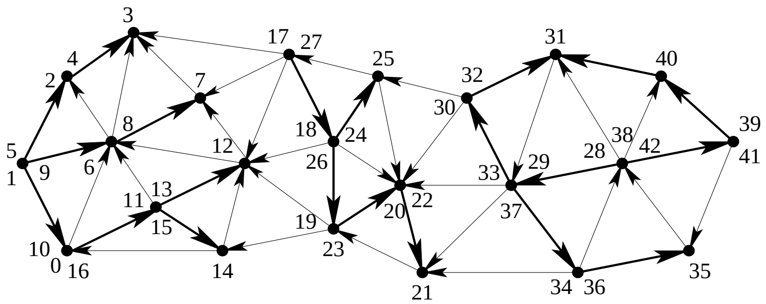
STEP 1: Initialize $F := \emptyset$, $R := \emptyset$, $W := \emptyset$, $x := \emptyset$; $\text{ind}(x) = -1$, $f(x) := -1$; then go to STEP 2.

STEP 2: *If $G - F = \emptyset$,* STOP.

If $G - F \ne \emptyset$, choose a vertex $u \in G - F$; add u to F, R, and to the end of W; set $f(u) := \text{ind}(x) + 1$; assign u to the vertex variable x and set $\text{ind}(x) := f(u)$; then go to STEP 3.

STEP 3: If $(x, G - F) \neq \emptyset$, select an arc a from x to $w \in G - F$; add a to F and the word aw to the end of W; set $p(w) := x$, $f(w) := \mathrm{ind}(x) + 1$; assign w to the vertex variable x, set $\mathrm{ind}(x) := f(w)$; then return to STEP 3.
If $(x, G - F) = \emptyset$, set $l(x) := \mathrm{ind}(v)$; then go to STEP 4.

STEP 4: If $x = u$, set $d(x) := \#\{i \mid v_i = x \text{ in } W\} - 1$; go to STEP 2.
If $x \neq u$, set $d(x) := \#\{i \mid v_i = x \text{ in } W\}$, backtrack from v to its parent $p(x)$ through an arc a in F, then add the word $a\,p(x)$ to the end of W; set $\mathrm{ind}(p(x)) := \mathrm{ind}(x) + 1$; assign $p(x)$ to the vertex variable x, set $\mathrm{ind}(x) := \mathrm{ind}(p(x))$; and then go to STEP 3.



Lemma 3.3. *Let W be a walk resulted by the Directed DFS-Branching Forest algorithm, with the end vertex v assigned to the variable x. Let $F(x)$ denote the branching forest produced by W at stage $l(x)$, i.e., the directed cut $(x, G - F(x))$ is empty. For each vertex u of $F(x)$, let $F_u(x)$ denote the branching of $F(x)$ at u as the root. Then the directed cut $(F_x(x), G - F(x))$ is also empty.*

Proof. We proceed by induction on the number of vertices of $F_x(x)$. It is true when $F_x(x)$ has the only vertex x, i.e., when x has no children in $F_x(x)$. Let x have children $w_1, w_2, \ldots, w_k$ in $F_x(x)$, been added to W in its current order. For the variable to reach the vertex v, it must be backtracked from w_i to v in the order $w_1, w_2, \ldots, w_k$. This means that the direct cuts $(w_i, G - F(w_i))$ are empty. By induction, the directed cuts $(F_{w_i}(w_i), G - F(w_i))$ are empty. Note that $F_{w_i}(x) = F_{w_i}(w_i)$ and $F(w_i) \subseteq F(x)$. Thus we have directed cut

$$\big(F_x(x), G - F(x)\big) = \bigcup_{i=1}^{k} \big(F_{w_i}(x), G - F(x)\big) \subseteq \bigcup_{i=1}^{k} \big(F_{w_i}(w_i), G - F(w_i)\big) = \emptyset.$$

□

- Let F be a branching spanning forest of a digraph D. An arc a with tail u and head v, written $a = (u, v)$, is called a **forward arc** if u is an ancestor of v in F, a **back arc** if u is a descendant of v in F, and a **cross arc** if u is neither an ancestor nor a descendant of v in F.
- Cross arcs can be happened inside a branching tree component of a DFS-branching forest F.
- The branching components of F can be linearly ordered as $T_1, T_2, \ldots, T_r$ so that $(T_i, T_j) = \emptyset$ for all $i < j$.

Proposition 3.4. *Let F be a DFS-branching forest of a digraph D. Let $u, v \in V(D)$, $F_u := F_u(u)$, and (x, y) be an arc in D. Then*

(a) $v \in F_u \Leftrightarrow f(u) < f(v) \leq l(v) < l(u)$.

(b) $F_u \cap F_v = \emptyset \Leftrightarrow f(u) \leq l(u) < f(v) \leq l(v)$ *or* $f(v) \leq l(v) < f(u) \leq l(u)$.

(c) (x, y) *is a forward arc* $\Leftrightarrow f(x) < f(y) \leq l(y) < l(x)$.

(d) (x, y) *is a back arc* $\Leftrightarrow f(y) < f(x) \leq l(x) < l(y)$.

(e) (x, y) *is a cross arc* $\Leftrightarrow f(y) \leq l(y) < f(x) \leq l(x)$.

Proof. (a) and (b) follow from Lemma 3.3; and (c), (d), (e) follow from (a) and (b). □

Proposition 3.5. *Let F be a DFS-branching forest of a digraph D. If C is a strong component of D, then $F \cap C$ is a spanning branching of C.*

Proof. Let x be a vertex of C such that $f(x)$ is smallest in C. Let F_x be the sub-branching of F generated by x. We first claim that $F_x \cap C$ is a branching with the root x. In fact, for each vertex $v \in F_x \cap C$, let P_{xv} be the unique directed path from x to v in F_x. Since C is a strong component of D, then $C \cup P_{xv}$ is also strong. Thus P_{xv} is contained in C; subsequently, P_{xv} is contained in $F_x \cap C$. So v is connected to x in $F_x \cap C$. This means that $F_x \cap C$ is a branching.

Now it suffices to show that $V(F_x \cap C) = V(C)$. Suppose $V(F_x \cap C) \neq V(C)$. Take a vertex $y \in V(C) - V(F_x \cap C)$; clearly, $y \in C$ and $y \notin F_x$. We claim that C has no arc from F_x to $D - F_x$. In fact, there is no arc in C from F_x to $F(x) - F_x$, where $F(x)$ is the branching forest generated by the Directed DFS Algorithm at time $l(x)$. (Otherwise, if $(u, v) \in C$ is an arc from F_x to $F(x) - F_x$, then the vertex v enters W before x; thus $v \in C$ and $f(v) < f(x)$; this is contradict to the choice of x.) By virtue of the Directed DFS Algorithm, there is no arc from F_x to $D - F(x)$; of course C has no arc from F_x to $D - F(x)$. It follows that C has no arc from F_x to $D - F_x$. Since $x, y \in C$, there is a directed path P in C from x to y. Since $x \in F_x$ and $y \notin F_x$, P has an arc from F_x to $D - F_x$; so is C. This is contradict to that C has no arc from F_x to $D - F_x$. Hence $V(F_x \cap C) = V(C)$. This means that $F_x \cap C = F \cap C$. $\square$

Remark. Let D be a digraph. (a) Applying Directed DFS to find a spanning forest F of D.

(b) Delete all cross edges of D between branching components of F and reverse the orientations of all remaining edges to produce a subdigraph $\tilde{D}$.

(c) For each branching component (T, v) of F with the root v, let $\tilde{D}(T)$ denote the subdigraph of $\tilde{D}$, induced by $V(T)$. Applying Directed BFS or DFS to find a branching $(\tilde{T}, v)$ rooted at v. Then the subdigraph $D(V(\tilde{T}))$ is a strong component of D containing the vertex v.

(d) Delete $\tilde{D}(V(\tilde{T}))$ from $\tilde{D}$, select another branching of $\tilde{D} - \tilde{D}(V(\tilde{T}))$; repeating the above procedure to find another strong component of D.