Week 6: Tree-Searching Algorithm

October 21, 2020

1 Tree-Search

- Given a tree T of a graph G. If $V(T) = V(G)$, then G is connected.

  If $V(T) \subsetneq V(G)$, let $G\backslash T$ denote the subgraph obtained from G by deleting all vertices of T (any edge with an endpoint in T is also deleted.) Then either $[T, G\backslash T] = \emptyset$ or $[T, G\backslash T] \neq \emptyset$. In the former case, G is disconnected. In the latter case, for each edge $e = uv \in [T, G\backslash T]$ with $u \in T, v \in G\backslash T$, the subgraph $T \cup e$ is again a tree of G.

- Using the idea above, one may generate a sequence of trees in G, starting with a 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 $[T, G\backslash T] = \emptyset$. 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 $P = v_0 e_1 v_1 \cdots e_l v_l$ be the unique path in T from v_0 to a vertex $v = v_l$. Each vertex $u = v_i$ of P, including v itself, is called an **ancestor** of v; and v is called a **descendent** of u in T. The vertex v_{l-1} is called the **predecessor** (or **parent**) of v, denoted $p(v)$, and v is called a **successor** (or **child**) of v_{l-1}.

- There are two typical tree-searches: **Breadth-First Search** (BFS) and **Depth-First Search** (DFS).

Theorem 1.1 (Breadth-First Search). *Search for rooted spanning tree in connected graph.*
INPUT: a connected graph $G = (V, E)$ with specified vertex v_0.
OUTPUT: a rooted tree (T, v_0) with the root v_0 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 Initialize $T := v_0$ and $Q := v_0$ a queuing sequence of vertices; set $\ell(v_0) := 0$.

STEP 2 If $Q = \emptyset$, STOP.
       If $Q \neq \emptyset$, delete the initial vertex $u$ of $Q$, then go to STEP 3.

STEP 3 If $[u, V(T)^c] = \emptyset$, return to STEP 2.
       If $[u, V(T)^c] = \{e_i : 1 \leq i \leq k\}$ with $e_i = uw_i$, add $e_i w_i$ to $T$ and $w_i$ to the end of
       $Q$, set $\ell(w_i) := \ell(u) + 1$ for $i = 1, \ldots, k$, then return to STEP 2.
```

Proof. Initially, $Q = v_0 \neq \emptyset$. It is clear that at any stage the subgraph T is always a tree having root v_0. Since G is connected, if $V(T) = V$, then T is a spanning tree of G having root v_0.

If $V(T) \subsetneq V$, there are some vertices of T adjacent with some vertices of $V(T)^c$. Let u be the first vertex of Q that adjacent with vertices $w_1, \ldots, w_k$ of $V(T)^c$. Then the algorithm repeats between STEP 3 and STEP 2 at all vertices of Q before u. When u becomes the initial vertex of Q, we have $w_i \notin Q$, $w_i \notin T$, 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 $\ell(w_i) = d_G(v_0, w_i)$. By induction we may assume that $\ell(v) = d_G(v_0, v)$ for all $v \in T$. Since $\ell(u) = d_G(v_0, u)$, we have

$$d_G(v_0, w_i) \leq d_G(v_0, u) + d_G(u, w_i) = \ell(u) + 1 = \ell(w_i).$$

Let $P_i = v_0 e_1 v_1 \cdots e_l v_l$ be a shortest path in G from v_0 to v_l $(= w_i)$, and v_j the last vertex of P_i belonging to T, at the time when u leaves Q, i.e., $v_j \in T$ with $j \leq l-1$, $v_{j+1} \notin T, \ldots, v_l \notin T$. We claim that $\ell(v_j) \geq \ell(u)$. In fact, if $\ell(v_j) < \ell(u)$, then by Lemma 1.2, v_j enters Q before u; consequently, v_j leaves Q before u, and at the time that v_j leaves Q, the vertex v_{j+1} enters Q and T; of course, at the time when u leaves Q, we have $v_{j+1} \in T$ which is contradictory to $v_{j+1} \notin T$. Thus

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

We have seen that $\ell(w_i) = d_G(v_0, w_i)$. □

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

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

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

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

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 G. Assume that it is true when $\ell(u) < l$, and consider the case $\ell(u) = l \geq 1$. 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 x enters Q before y; consequently, x leaves Q before y. Note that u enters Q right after x leaves Q, and at that time v did not enter Q yet, for it is not yet the turn for y to leave Q. Thus u enters Q before v.

(b) is an equivalent version to (a).

(c) If $\ell(u) = \ell(v)$, nothing is to be proved. If $\ell(u) \neq \ell(v)$, 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; if $v \in Q$, let y be the parent of v, then $\ell(v) = \ell(y) + 1$, and by definition y enters Q before u, hence $\ell(y) \leq \ell(u)$ by (b). Now we have

$$\ell(u) < \ell(v) = \ell(y) + 1 \leq \ell(u) + 1,$$

which forces that $\ell(v) = \ell(u) + 1$. □

Theorem 1.3 (Depth-First Search-Tree Algorithm)**.** INPUT: a connected graph $G = (V, E)$ with specified vertex v_0.
OUTPUT: a rooted tree (T, v_0) with the root v_0, a closed walk $W = v_0 e_1 v_1 \cdots e_{2n-2} v_{2n-2}$ with $n = |V|$, a parent function $p : V \smallsetminus \{v_0\} \to V$, and two time functions $l, \ell : V \to \mathbb{N}$.

STEP 1 *Initialize a vertex variable x and a rooted tree (T, v_0) with $T := v_0$, assign v_0 to x, set $W := v_0$, $l(v_0) := 0$, $\operatorname{ind}(x) := 0$, then go to* STEP 2.

STEP 2 *If $[x, V(T)^c] \neq \emptyset$, select an edge $e = xw$ with $w \in V(T)^c$, add ew to T and to the end of W, set $p(w) := x$, $l(w) := \operatorname{ind}(x) + 1$, assign w to x and set $\operatorname{ind}(x) := l(w)$, then return to* STEP 2.

If $[x, V(T)^c] = \emptyset$, set $\ell(x) := \operatorname{ind}(x)$, then go to STEP 3.

STEP 3 *If $x = v_0$, then* STOP. *(A spanning rooted tree T is found.)*

If $x \neq v_0$, backtrack from x to its parent u through an edge $e = xu$ in T, add eu to the end of W, set $\operatorname{ind}(u) := \operatorname{ind}(x) + 1$, assign u to x and set $\operatorname{ind}(x) := \operatorname{ind}(u)$, 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 $\ell(v_0)$, the tree T is produced by the walk W, and $[v_0, V(T)^c] = \emptyset$. Then by Lemma 1.4, $[T_{v_0}, V(T)^c] = \emptyset$, i.e., $[T, V(T)^c] = \emptyset$. Since G is connected, it forces that $V(T) = V$, i.e., T is a spanning tree of G. □

The time functions l and ℓ are actually given by

$$l(v) = \min\{i : v = v_i \in W\}, \quad \ell(v) = \max\{i : v = v_i \in W\}$$

Lemma 1.4. *Let $W(k)$ be a walk resulted by the DFS-Tree Algorithm when a vertex v is assigned to x with $\operatorname{ind}(x) = k$. Let $T(k)$ denote the rooted tree produced by $W(k)$. For each vertex $u \in T(k)$, let $T_u(k)$ denote the rooted subtree of $T(k)$ having its root at u. If $[v, T(k)^c] = \emptyset$, then $[T_v(k), T(k)^c] = \emptyset$.*

Proof. We proceed by induction on the number of vertices of $T_v(k)$. It is trivially true if $T_v(k) = v$. If v has children $w_1, \ldots, w_j$ in $T(k)$, the children must have been added to $W(k)$ in its current order. To have v assigned to the vertex variable x, it must be backtracked from w_i to v in order $w_1, \ldots, w_j$. This means that $[w_i, T(k)^c] = \emptyset$ for $i = 1, \ldots, j$. Since $T_{w_i}(k)$ has less number of vertices than $T_v(k)$, by induction we have $[T_{w_i}(k), T(k)^c] = \emptyset$. Thus $[T_v(k), T(k)^c] = [v, T(k)^c] \cup \bigcup_{i=1}^{j} [T_{w_i}(k), T(k)^c] = \emptyset$. □

Proposition 1.5. *Let (T, v_0) be a rooted tree of a connected graph G resulted by the DFS-Tree Algorithm. Given two distinct vertices u and v.*

(a) The vertex v is a descendant of u iff $l(u) < l(v) \leq \ell(v) < \ell(u)$.

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

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

Proof. (a) It is trivial by definition that $l(v) \leq \ell(v)$. Let v be a descendant of u, i.e, $v \in T_u$. Let $u_0u_1 \ldots u_k$ be the unique shortest path from u to v in T_u. Then

$$l(v) = l(u_k) = l(u_0) + k = l(u) + k > l(u).$$

When the vertex variable x is at v, to reach u again, x must be backtracked from v. Thus by definition $\ell(v) < \ell(u)$. Conversely, let $l(u) < l(v) \leq \ell(v) < \ell(u)$. Clearly, $l(u) < l(v)$ implies that 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 finished. Thus we have $\ell(u) < l(v)$ by definition of l and ℓ, which is contradictory to $l(v) < \ell(u)$.

(b) We may assume that $l(u) < l(v)$, 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. Thus at the stage $\ell(u)$, we have $[u, T_u^c] = \emptyset$. However, $e = uv \in [u, T_u^c] \neq \emptyset$, which is a conradiction.

(c) Note that $v \in T_u$ iff $\ell(v) < \ell(u)$ under the given condition $l(u) < l(v)$ by (a). If $\ell(v) < \ell(u)$ is not valid, i.e., $v \notin T_u$, then v enters W after all vertices of T_u; thus $\ell(u) < l(v)$ by definition. □

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

(a) Each leaf of T is not a cut vertex of G.

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

(c) A vertex v other than v_0 and leaves of T is a cut vertex of G iff 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. (a) This is trivially true for any spanning tree T.

(b) For necessity, if v_0 has only one child, i.e., v_0 is a leaf, then v_0 cannot be a cut vertex by (a), a contradiction. For sufficiency, let w_1, w_2 be two children of v_0 in T. We may assume $\ell(w_1) < l(w_2)$. Then $w_2 \in T_{w_1}$ and $[T_{w_1}, T_{w_1}^c] = \emptyset$ by Lemma 1.4. Thus $G \smallsetminus v_0$ has at least components G_1, G_2 with $w_i \in G_1$ and $w_2 \in G_2$. Hence v_0 is a cut a vertex of G.

(c) Lemma 1.4 implies that there are no edges between any two rooted sub-trees whose roots have a common parent. At the time $l(v)$ in the DFS-Tree Algorithm, let $w_1, \ldots, w_k$ be the vertices such that $e_i = vw_i \in [v, T^c]$, $i = 1, \ldots, k$. Then v is not a cut vertex for G iff there are edges from some ancestors of v to each of the rooted sub-trees $T_{w_1}, \ldots, T_{w_k}$, i.e., to some descendants of each child w_i of v. □

2 Minimum Weight Spanning Tree

- A **weighted graph** is a graph $G = (V, E)$ together with a **weight function** $w : E \to \mathbb{R}$, denoted (G, w). For each edge $e \in E$, 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 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 u of G, initialize a tree $T := u$, then go to* STEP 2.

STEP 2 *If $V(T) = V$,* STOP.

If $V(T) \subsetneq V$, choose an edge $e = vw \in [T, T^c]$ such that

$$w(e) = \min \big\{ w(e') : e' \in [T, T^c] \big\},$$

add ew to T, then return to STEP 2.

Proof. It is clear that in STEP 2 the subgraph $T \cup e$, constructed by adding the edge e from $[T, T^c]$ to the tree T, is still a tree. Finally, the tree T grows up to a spanning tree when the algorithm stops. We are left to show that the produced tree is optimal. It is enough to show that at any stage the tree 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 2 the tree is contained in an optimal spanning tree T^*. Note that $w(e) \leq w(e')$ for all $e' \in [T, T^c]$ by STEP 2. If $e \in T^*$, then the optmal spanning tree T^* contains $T \cup e$ already. If $e \notin T^*$, then $T^* \cup e$ contains a unique cycle C_e and $e \in C_e$. Since C_e intersects the cut $[T, T^c]$, there exists an edge $e^* \in C_e \cap [T, T^c]$ other than e. Clearly, the spanning tree $T^{**} := (T^* \cup e) \smallsetminus e^*$ contains $T \cup e$, and

$$w(T^{**}) = w(T^*) + w(e) - w(e^*) \leq w(T^*).$$

The optimality of T^* implies that T^{**} is an optimal spanning tree containing $T \cup e$. □

Theorem 2.2 (Kruskal's Algorithm). INPUT: a connected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$. OUTPUT: a minimum-weight spanning tree T of G.

STEP 1 *Choose an edge e_0 of $E(G)$ such that $w(e_0) = \min\{w(e) : e \in G\}$, initialize a tree $T := e_0$, then go to* STEP 2.

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

STEP 3 *If $|E(T)| < |V| - 1$, choose an edge $e \in E \smallsetminus E(T)$ such that $T \cup e$ contains no cycles and*

$$w(e) = \min \big\{ w(e') : e' \in T^c,\ T \cup e' \text{ contains no cycle} \big\},$$

add e to T, then return to STEP 2.

Proof. It is enough to show that at any stage the subgraph T is always contained in an optimal spanning tree of G. We proceed by induction on the number of edges of T. Initially, $T := e_0$, and $w(e_0)$ is the minimum weight. Let T^* be an optimal spanning tree of G. If $e_0 \in T^*$, the optimal spanning tree T^* contains T already. If $e_0 \notin T^*$, then $T^* \cup e_0$ contains a unique cycle C_{e_0}. Select an edge e_1 from C_{e_0} other than e_0; the spanning tree $T^{**} := (T^* \cup e_0) \smallsetminus e_1$ contains e_0. Since $w(e_0) \leq w(e')$ for all $e' \in E$, we have

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

The optimality of T^* implies that 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 $T \cup e$ contains no cycles, and $w(e) \le w(e')$ for all $e' \in E \smallsetminus E(T)$ such that $T \cup e'$ contains no cycles. If $e \in T^*$, then the optimal spanning tree T^* contains $T \cup e$ already. If $e \notin T^*$, then $T^* \cup e$ contains a unique cycle C_e and $e \in C_e$. Since C_e intersects the cut $[T, T^c]$, there exists an edge $e^* \in C_e \cap [T, T^c]$ other than e. Clearly, the spanning tree $T^{**} := (T^* \cup e) \smallsetminus e^*$ contains $T \cup e$, and

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

The optimality of T^* implies that T^{**} is an optimal spanning tree containing $T \cup e$. □

3 Branching-Search

A **branching** is a rooted tree with an orientation such that every vertex but the root has in-degree 1. A branching with a root u is called a u-**branching**.

Theorem 3.1 (Breadth-First Search for Spanning Branching Forest).
Input: a digraph $D = (V, A)$.
Output: a spanning branching forest F in D with a root set R, a parent function $p : V \smallsetminus R \to V$, and a level function $\ell : V \to \mathbb{N}$ such that $\ell(v) = 0$ for all $v \in R$.

Step 1 *Initialize $F := \emptyset$, $R := \emptyset$, and a vertex queuing sequence $Q := \emptyset$; then go to* Step 2.

Step 2 *If $V(F) = V$,* Stop. *(A spanning branching forest F with root set R is found.)*

If $V(F) \subsetneq V$ and $Q = \emptyset$, choose a vertex $u \in V(F)^c$; add u to F, R, and Q; set $\ell(u) := 0$; then return to Step 2.

If $V(F) \subsetneq V$ and $Q \ne \emptyset$, delete the initial vertex u of Q; then go to Step 3.

Step 3 *If $(u, V(F)^c) = \emptyset$, then return to* Step 2

If $\big(u, V(F)^c\big) = \{a_1, \ldots, a_k\}$, where $a_i = uw_i$ with $w_i \in V(F)^c$; add $a_i w_i$ to F and w_i to the end of Q in the order $w_1, \ldots, w_k$; set $p(w_i) := u$, $\ell(w_i) := \ell(u) + 1$; then return to Step 2.

Proof. Similar to the proof of Theorem 1.2. □

Let $D = (V, A)$ be a digraph with a positive weigh function $w : A \to \mathbb{R}_+$. Fix a vertex v_0 and a vertex v reachable from v_0. Let $d_D(v_0, v)$ denote the shortest directed distance from v_0 to v in D. For each vertex subset $S \subset V$ that there exists a directed path from v_0 to a vertex of S, we denote by $d_D(v_0, S)$ the shortest directed distance from v_0 to S.

Lemma 3.2. *Let $D = (V, A)$ be a digraph with a positive weight function $w : A \to \mathbb{R}_+$. Given a vertex proper subset $S \subsetneq V$ with $v_0 \in S$. If S^c is reachable from v_0, then*

$$d_D(v_0, S^c) = \min\{d_D(v_0, u) + w(a) : a = uv \in (S, S^c)\}.$$

Moreover, if $P = v_0 a_1 v_1 \cdots v_{k-1} a_k v_k$ is a shortest directed path from v_0 to S^c, then $v_{k-1} \in S$ and $P_1 = v_0 e_1 v_1 \cdots e_{k-1} v_{k-1}$ is a shortest directed path from v_0 to v_{k-1}.

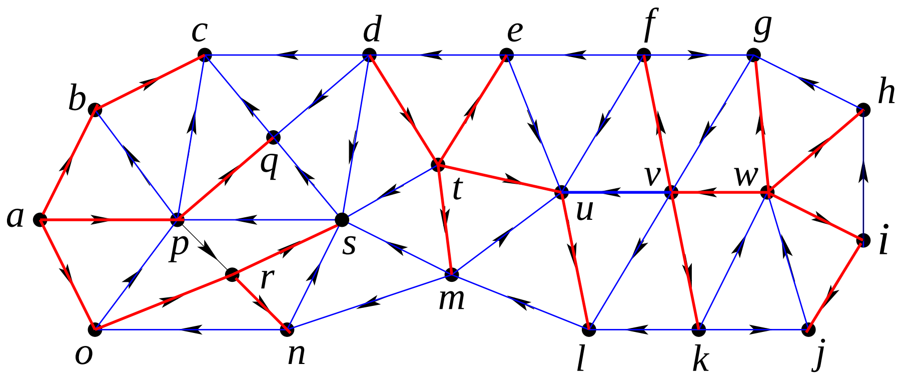

Figure 1: A BFS-branching forest F with the root set $\{a, d, v\}$

Proof. Suppose $v_{k-1} \notin S$, i.e., $v_{k-1} \in S^c$. Then P_1 is a shorter path than P from v_0 to S^c, which is contradictory to that P is a shortest directed path from v_0 to S^c. Suppose there is a directed path P_1' shorter than P_1 from v_0 to v_{k-1}. Then $P_1' a_k v_k$ is a directed path shorter than P from v_0 to S^c, which is a contradiction.

Since $d_D(v_0, S^c) = \sum_{i=1}^{k} w(a_i)$ and $d_D(v_0, v_{k-1}) = \sum_{i=1}^{k-1} w(a_i)$, we have $d_D(v_0, S^c) = d_D(v_0, v_{k-1}) + w(a_k)$. Since $v_{k-1} \in S$ and $a_k = v_{k-1}v_k \in (S, S^c)$, we see that

$$d_D(v_0, vS^c) \geq \min\{d_D(v_0, u) + w(a) : a = uv \in (S, S^c)\}.$$

For each $a = uv \in (S, S^c)$ and $d_D(v_0, u)$, there exists a directed path P' from v_0 to v_{k-1}. Then $P'av$ is a directed path from v_0 to S^c. Thus $d_D(v_0, S^c) \leq d_D(v_0, u) + w(a)$. It follows that $d_D(v_0, vS^c) \leq \min\{d_D(v_0, u) + w(a) : a = uv \in (S, S^c)\}$. □

Theorem 3.3 (Dijkstra's Algorithm for Shortest Path)**.** Input: a digraph $D = (V, A)$ with a specified vertex v_0 and a positive weight function $w : A \to \mathbb{R}_+$. Output: a branching (T, v_0) in D with root v_0, a parent function $p : V(T) \smallsetminus \{v_0\} \to V(T)$, and a level function $\ell : V(T) \to \mathbb{N}$ such that $\ell(v) = d_D(v_0, v)$ for all $v \in V(T)$.

Step 1 *Initialize a branching (T, v_0) with $T := v_0$; set $\ell(v_0) := 0$, $p(v_0) := \emptyset$; then go to* Step 2.

Step 2 *If $\big(V(T), V(T)^c\big) = \emptyset$,* Stop.

If $\big(V(T), V(T)^c\big) \neq \emptyset$, choose an arc $a = uv \in \big(V(T), V(T)^c\big)$ such that

$$\ell(u) + w(a) = \min\left\{\ell(u') + w(a') : a' = u'v' \in \big(V(T), V(T)^c\big)\right\};$$

add av to T; set $p(v) := u$, $\ell(v) := \ell(u) + w(a)$; then return to Step 2.

Proof. Lemma 3.2 implies that $d_D(v_0, V(T)^c) = \ell(u) + w(a)$, where $a = uv \in \big(V(T), V(T)^c\big)$. It follows that $d_D(v_0, v) = \ell(u) + w(a) = \ell(v)$. □

Theorem 3.4 (Depth-First Search for Spanning Branching Forest)**.** 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 parent function $V \smallsetminus R \to V$, and two time functions $l, \ell : V \to \mathbb{N}$ such that $l(v) \leq \ell(v)$ for all $v \in V$.

STEP 1 *Initialize* $F := \emptyset$, $R := \emptyset$, $W := \emptyset$, *a vertex variable* $x := \emptyset$; *set* $\text{ind}\,(x) := -1$ *and* $l(x) := -1$; *then go to* STEP 2.

STEP 2 *If* $V(F)^c = \emptyset$, STOP.

If $V(F)^c \neq \emptyset$, *choose a vertex* $u \in V(F)^c$; *add* u *to* F, R, *and to the end of* W; *set* $l(u) := \text{ind}\,(x) + 1$; *assign* u *to the vertex variable* x *and set* $\text{ind}\,(x) := l(u)$; *then go to* STEP 3.

STEP 3 *If* $\big(x, V(F)^c\big) \neq \emptyset$, *select an arc* $a = xw$ *with* $w \in V(F)^c$; *add* aw *to* F *and to the end of* W; *set* $p(w) := x$ *and* $l(w) := \text{ind}\,(x) + 1$; *assign* w *to* x *and set* $\text{ind}\,(x) := l(w)$; *then go to* STEP 3.

If $\big(x, V(F)^c\big) = \emptyset$, *set* $\ell(x) := \text{ind}\,(x)$; *then go to* STEP 4.

STEP 4 *If* $x = u$, *then go to* STEP 2.

If $x \neq u$, *backtrack from* x *to its parent* $p(x)$ *through an arc* a *in* F; *add* $ap(x)$ *to the end of* W; *set* $\text{ind}\,(p(x)) := \text{ind}\,(x) + 1$; *assign* $p(x)$ *to* x *and set* $\text{ind}\,(x) := \text{ind}\,(p(x)) + 1$; *then go to* STEP 3.

Proof. It is similar to that of Theorem 1.3. □

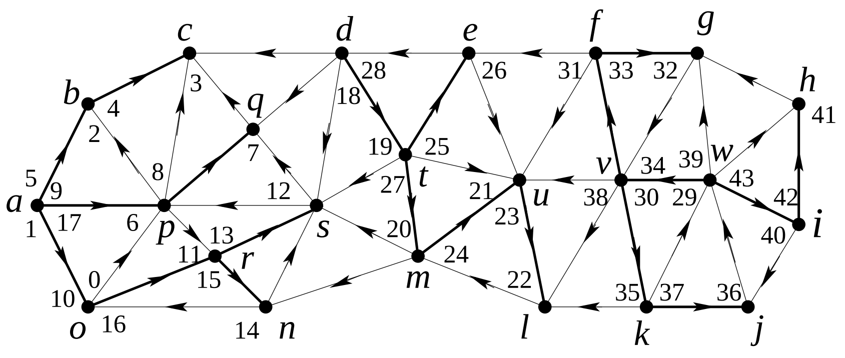

Figure 2: A DFS-branching forest F with the root set $\{a, d, v\}$

Lemma 3.5. *Let* W *be a walk resulted by the DFS-Branching Forest Algorithm, with the end vertex* v *assigned to to the vertex variable* x. *Let* $F(k)$ *denote the branching forest produced by* W *at stage* $\ell(v) = k$, *i.e.,* $(v, F(k)^c) = \emptyset$. *For each vertex* $u \in F(k)$, *let* $F_u(k)$ *denote the branching of* $F(k)$ *rooted at* u. *Then* $\big(F_v(k), F(k)^c\big) = \emptyset$.

Proof. We proceed by induction on the number of vertices of $F_v(k)$. It is trivially true when $F_v(k)$ contains the only vertex v, i.e., v has no children in $F_v(k)$. Let v have children $w_1, \ldots, w_j$ in $F_v(k)$, been added to W in its current order. For the vertex variable x to each the vertex v, it must be backtracked from w_i to v in order $w_1, \ldots, w_j$. This means that the directed cuts $\big(w_i, F(\ell(w_i))^c\big)$ are empty. By induction, the directed cuts $\big(F_{w_i}(\ell(w_i)), F(\ell(w_i))^c\big)$

are empty. Note that $F_{w_i}(\ell(v)) = F_{w_i}(\ell(w_i))$ and $F(\ell(w_i)) \subseteq F(\ell(v))$. We see that the directed cut

$$\begin{aligned}\big(F_v(\ell(v)), F(\ell(v))^c\big) &= \big(v, F(\ell(v))^c\big) \cup \bigcup_{i=1}^{j} \big(F_{w_i}(\ell(v)), F(\ell(v))^c\big) \\ &\subseteq \bigcup_{i=1}^{j} \big(F_{w_i}(\ell(w_i)), F(\ell(w_i))^c\big) = \emptyset.\end{aligned}$$

□

4 Finding Strong Components of Digraphs

Definition 4.1. 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

(a) **forward arc** if u is an ancestor of v in F, i.e., $l(u) < l(v)$ and $\ell(v) < \ell(u)$;

(b) **back arc** if u is a descendant of v in F, i.e., $l(v) < l(u)$ and $\ell(u) < \ell(v)$; and

(c) **cross arc** if u is neither ancestor nor a descendant of v in F, i.e., $\ell(v) < l(u)$.

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 $T_1, \ldots, T_k$ so that there are no directed edges from T_i to T_j, i.e., $(T_i, T_j) = \emptyset$, for all $i < j$.

Proposition 4.2. *Let F be a DBS-branching forest of a digraph D. Given two vertices $u, v \in V(D)$, let $F_u = F_u(\ell(u))$. Then*

(a) $v \in F_u$ iff $l(u) < l(v) \le \ell(v) < \ell(u)$.

(b) $F_u \cap F_v = \emptyset$ iff either $l(u) \le \ell(u) < l(v) \le \ell(v)$ or $l(v) \le \ell(v) < l(u) \le \ell(u)$.

Proof. Parts (a) and (b) follow from Lemma 3.5. □

Proposition 4.3. *Let F be a DBS-branching forest of a digraph D. If C is a strong component of D, then $F \cap C = (V(F) \cap V(C), A(F) \cap A(C))$ is a spanning branching of C.*

Proof. Let u be the first vertex of C entered W, i.e., $l(u)$ is smallest on C. Let F_u be the sub-branching of F rooted at u. We claim that $F_u \cap C$ is a branching with root u. In fact, for each vertex $v \in F_u \cap C$, let P_{uv} be the unique directed path from u to v in F_u. Since C is strongly connected, we see that $C \cup P_{uv}$ is also strongly connected. It follows that P_{uv} is contained in C, consequently, P_{uv} is contained in $F_u \cap C$. This means that $F_u \cap C$ is a branching rooted at u.

Next it suffices to show that $V(F_u \cap C) = V(C)$. Suppose $V(F_u \cap C) \subsetneq V(C)$. Take a vertex $w \in V(C) \smallsetminus V(F_u \cap C)$, i.e., $w \in V(C)$, $w \notin V(F_u)$. By the Directed DFS Algorithm, there is no arc from F_u to F_u^c. Since $u, w \in V(C)$, there exists a directed path P in C from u to w. Since $u \in V(F_u)$ and $w \notin V(F_u)$, the path P has an arc from F_u to F_u^c, which is a contradiction. □

Proposition 4.4. *Let D be a digraph. Applying Directed DFS to find a spanning forest F of D. For each strong component C of D we associate with C a unique vertex, the root of $F \cap C$. Let D' denote the digraph obtained from D by deleting all cross edges relative to F and reversing orientations of the remaining edges. Then the set of vertices reachable from a root u in D' induces a strong component of D.*

Proof. Given a branching T with a root u. For any two branching tree components T_x, T_y of T with $\ell(x) < l(y)$ there is no direct edge from T_x to T_y. If a vertex $x' \in T_x$ other than x is strongly connected to a vertex $y' \in T_y$ other than y, there must be a shortest directed path from x' to y' via a common ancestor of x' and y'. So deleting cross edges between tree components does not change strong connectedness.

For the branching T with the root u, let Let U be the set of vertices reachable from u in D'. Each directed path P' from u to a vertex $v \in U$ in D' can be transformed into a directed path P from v to u by reversing the arc of P'. So the sub-digraph $D(U)$ of D generated by U is a strong component of D.

Delete U from T to obtain sub-branchings $T_{u_1}, \ldots, T_{u_k}$. Let U_i be the set of vertices reachable from u_i in D. Then $D(U_i)$ are strong components of D. Continue this procedure, one obtain all strong components of D. □

Example 4.1. Consider the DFS-branchings T_a, T_d, T_w of D in Figure 2. For T_a, the set of vertices reachable from a in D' is $\{a\}$. For the tre components T_b, T_o, T_p, the sets of vertices reachable from b, o, p respectively in $D'(T_b), D'(T_o), D'(T_p)$ are $\{b\}, \{o, n, r\}$, $\{p, s\}$ respectively. The branching T_a induces strong components:

$$\{a\}, \quad \{b\}, \quad \{c\}, \quad \{p,\}, \quad \{q\}, \quad \{s\}, \quad \{o, n, r\}.$$

The branching T_d induces strong components: $\{d, e, t\}, \{m, l, u\}$. The branching T_w induces strong components: $\{w, j, k, v, f, g\}, \{g\}, \{h\}, \{i\}$.

Exercises

Ch6: 6.1.1; 6.1.4; 6.2.2; 6.2.3; 6.3.7; 6.3.12.