Chapter 1

Rooted forests and trees

The notion of a rooted forest should be familiar to the reader. For completeness, we will give formal definitions.

Let \( N \) be a finite set. A rooted tree on \( N \) is defined by the pair \((N, p)\) where \( p \) is a function \( p : N \to N \cup \{\bot\} \) such that

- there is no positive integer \( k \) such that \( p^k(x) = x \) for some element \( x \in N \), and
- there is exactly one element \( x \in N \) such that \( p(x) = \bot \).

This element is called the root. The elements of \( N \) are called the nodes of the tree. They are also called the vertices of the tree.

For each nonroot node \( x \), \( p(x) \) is called the parent of \( x \) in the tree, and the ordered pair \( xp(x) \) is called the parent arc of \( x \) in the tree. An arc of the tree is the parent arc of some nonroot node. If the parent of \( x \) is \( y \) then \( x \) is a child of \( y \) and \( xy \) is the child arc of \( y \).

A rooted tree has arity \( k \) if every node has at most \( k \) children. A binary tree is a tree that has arity 2.

We say two nodes are adjacent if one is the parent of the other. We say the arc \( xp(x) \) is incident to the nodes \( x \) and \( p(x) \).

The ancestors of \( x \) are defined inductively: \( x \) is its own ancestor, and (if \( x \) is not the root) the ancestor’s of \( x \)’s parent are also ancestors of \( x \). If \( x \) is the ancestor of \( y \) then \( y \) is a descendant of \( x \). We say \( y \) is a proper ancestor of \( x \) (and \( x \) is a proper descendant of \( y \)) if \( y \) is an ancestor of \( x \) and \( y \neq x \). The depth of a node is the number of proper ancestors it has.

We say an arc \( xp(x) \) is an ancestor arc of \( y \) if \( x \) is an ancestor of \( y \). We say \( xp(x) \) is a descendant arc of \( y \) if \( p(x) \) is a descendant of \( y \).

A subtree of \((N, p)\) is a tree \((N', p')\) such that \( N' \) is a subset of \( N \). A rooted forest is a collection of disjoint rooted trees. That is, \((N, p)\) is a forest if there are trees \((N_1, p_1), \ldots, (N_k, p_k)\) such that

\[
N = N_1 \cup N_2 \cup \cdots \cup N_k
\]
and $p_i$ is the restriction of $p$ to $N_i$.

Deletion of an arc $\hat{x}p(\hat{x})$ from a rooted forest $(N, p)$ is an operation that yields the forest $(N, p')$ where

$$p'(x) = \begin{cases} \bot & \text{if } x = \hat{x} \\ p(x) & \text{otherwise} \end{cases}$$

If $T$ is a rooted forest and $e$ is an arc of $T$ then we use $T - \{e\}$ to denote the result of deleting $e$.

Deletion of a node $\hat{x}$ from a rooted forest $(N, p)$ is an operation that yields the forest $(N - \{\hat{x}\}, p')$ where

$$p'(x) = \begin{cases} \bot & \text{if } p(x) = \hat{x} \\ p(x) & \text{otherwise} \end{cases}$$

If $T$ is a rooted forest and $\hat{x}$ is a node of $T$ then we use $T - \{\hat{x}\}$ to denote the result of deleting $\hat{x}$.

More generally, if $S$ is a set of nodes or a set of arcs, $T - S$ denotes the forest obtained by deleting every element of $S$.

For a tree $T$ and a node $x$ of $T$, the subtree rooted at $x$ is the tree obtained from $T$ by deleting every node that is not a descendant of $x$.

For a forest $T$ and a node $x$ of $T$, the root-to-$x$ path is the sequence $x_0x_1\ldots x_k$ where $x_0$ is the root of $T$, $x_k$ is $x$, and $x_i$ is the parent of $x_{i+1}$ for $i = 0, \ldots, k - 1$. We denote this path by $T[x]$.

Ancestorhood defines a partial order among nodes of a forest. Given a set $S$ of nodes of a forest, a rootmost node of $S$ in the forest is a node $v$ such that no proper ancestor of $v$ is in $S$. A leafmost node of $S$ is a node $v$ such that no proper descendant of $v$ is in $S$.

Given two nodes $u$ and $v$ of a forest, we say $u$ is leafward of $v$ and $v$ is rootward of $u$ if $u$ is a descendant of $v$. A sequence $v_1, \ldots, v_k$ of nodes of the forest is a leafward path if $v_i$’s parent is $v_{i+1}$ for $i = 1, \ldots, k - 1$.

### 1.1 Rootward computations

Suppose $T$ is a rooted tree and $w(\cdot)$ is an assignment of weights to the nodes. There is a simple, linear-time algorithm to compute, for each node $u$, the total weight of all descendants of $u$:

\[
\text{def totalWeight}(u): \\
\quad \text{return } w(u) + \sum \{\text{totalWeight}(v) : v \text{ a child of } u\}
\]

This algorithmic schema, though simple, comes up again and again: in finding separators for trees (in the next section), in algorithms that exploit interdigitating trees in planar graphs (Section 4.5), in processing a breadth-first-search tree (Section 5.4), in dynamic-programming algorithms on trees (Section 14.1) and on graphs of bounded carvingwidth (Section 14.3.1) and bounded branchwidth (Section 14.5.1).
1.2 Separators for rooted trees

A separator for a tree is a vertex or edge whose deletion results in trees that are “small” in comparison to the original graph.

**Lemma 1.2.1 (Leafmost Heavy Vertex).** Let $T$ be a rooted tree. Let $\hat{w}(\cdot)$ be an assignment of weights to vertices such that the weight of each vertex is at least the sum of the weights of its children. Let $W$ be the weight of the root, and let $\alpha$ be a positive number less than 1. Then there is a linear-time algorithm to find a vertex $v_0$ such that $\hat{w}(v_0) > \alpha W$ and every child $v$ of $v_0$ satisfies $\hat{w}(v) \leq \alpha W$.

**Proof.** Call the procedure below on the root of $T$.

\[
\text{define } f(v):
\begin{align*}
1 & \quad \text{if some child } u \text{ of } v \text{ has } \hat{w}(u) > \alpha W, \\
2 & \quad \text{return } f(u) \\
3 & \quad \text{else return } v
\end{align*}
\]

By induction on the number of invocations, for every call $f(v)$, we have $\hat{w}(v) > \alpha W$. If $v$ is a leaf then the condition in Line 1 is not satisfied, so the procedure terminates. Let $v_0$ be the vertex returned by the procedure. Since the condition in Line 1 did not hold for $v_0$, every child $v$ of $v_0$ satisfies $\hat{w}(v) \leq \alpha W$. \qed

![Figure 1.1: A rooted tree. The grey vertex is a separator whose deletion results in a forest, each of whose trees weighs at most 1/2 the number of vertices. The dashed edge is a separator whose deletion results in a forest, each of whose trees weighs at most 2/3 of the edges.](image)

1.2.1 Vertex separator

**Lemma 1.2.2 (Tree Vertex Separator).** Let $T$ be a rooted tree, and let $w(\cdot)$ be an assignment of weights to vertices. Let $W$ be the sum of weights. There
is a linear-time algorithm to find a vertex $v_0$ such that every tree in the forest $T - \{v_0\}$ has total weight at most $W/2$.

Proof. For each vertex $u$, define $\hat{w}(u) = \sum \{w(v) : v \text{ a descendant of } u\}$. Then $\hat{w}(\text{root}) = W$. The values $\hat{w}(\cdot)$ can be computed using a rootward computation as in Section 1.1. Let $v_0$ be the vertex of the Leafmost-Heavy-Vertex Lemma with $\alpha = 1/2$. Let $v_1, \ldots, v_p$ be the children of $v_0$. For each child $v_i$, the subtree rooted at $v_i$ has weight at most $W/2$. Each such subtree is a tree of $T - \{v_0\}$. The remaining tree is $T - \{v : v \text{ is a descendant of } v_0\}$. Since the sum $\sum_{i} w(v) = \hat{w}(v_0)$ exceeds $W/2$, the weight of the remaining tree is less than $W/2$. \hfill \qed

1.3 Edge separators

Lemma 1.3.1 (Tree Edge Separator of Edge-Weight). Let $T$ be a tree of maximum degree three, and let $w(\cdot)$ be an assignment of weights to edges. There is a linear-time algorithm to find an edge $\hat{e}$ such that every tree in $T - \{\hat{e}\}$ has at most two-thirds of the weight.

Proof. Assume for notational simplicity that the total weight is 1. Choose a vertex of degree one as root. For each nonroot vertex $v$, define

$$\hat{w}(v) = \sum \{w(e) : e \text{ a descendant edge of } v\} \cup \{\text{parent edge of } v\}$$

Define $\hat{w}(\text{root}) = 1$. Let $v_0$ be the vertex of the Leafmost-Heavy-Vertex Lemma with $\alpha = 1/3$. Let $e_0$ be the parent edge of $v_0$. Then $T - \{e_0\}$ consists of two trees. One tree consists of all descendants of $v_0$, and the other consists of all nondescendants.

The weight of all edges among the nondescendants is $1 - \hat{w}(v_0)$, which is less than $1 - 1/3$ since $\hat{w}(v_0) > 1/3$. Let $v_1, \ldots, v_p$ be the children of $v_0$. (Note that $0 \leq p \leq 2$.) The weight of all edges among the descendants is $\sum_{i=1}^{p} \hat{w}(v_i)$. Since $\hat{w}(v_i) \leq 1/3$ for $i = 1, \ldots, p$ and $p \leq 2$, we infer $\sum_{i} \hat{w}(v_i) \leq 2/3$. \hfill \qed

The following example shows that the restriction on the arity of the trees in Lemma 1.3.1 cannot be discarded:

If the number of children is $k$ then removal of any edge leaves weight $(k - 1)/k$.

The following example shows that, for trees of degree at most three, the factor two-thirds in Lemma 1.3.1 cannot be improved upon.
For some separators, we need to impose a condition on the weight assignment. We say a weight assignment is \(\alpha\)-proper if no element is assigned more than an \(\alpha\) fraction of the total weight.

**Lemma 1.3.2 (Tree Edge Separator of Vertex Weight).** Let \(T\) be a tree of degree at most three, and let \(w(\cdot)\) be a \(\frac{1}{4}\)-proper assignment of weights to vertices such that each nonleaf vertex is assigned at most one-fourth of the weight. There is a linear-time algorithm to find an edge \(\hat{e}\) such that every tree in \(T - \{\hat{e}\}\) has at most three-fourths of the weight.

**Proof.** Assume the total weight is 1. Root \(T\) at a leaf. For each vertex \(v\), define
\[
\hat{w}(v) = \sum \{w(v') : v' \text{ a descendant of } v\}
\]

Let \(v\) be the vertex of the Leafmost-Heavy-Vertex Lemma with \(\alpha = \frac{3}{4}\). Let \(v_1, \ldots, v_p\) be the children of \(v\). Note that \(0 \leq p \leq 2\). Since \(w(v) \leq \frac{3}{4}\) but \(\hat{w}(v) > \frac{1}{4}\), we must have \(p > 0\), so \(w(v) \leq \frac{1}{4}\).

For \(1 \leq i \leq p\), let \(W_i\) be the weight of descendants of \(v_i\). Let \(\hat{i} = \text{maxarg}_{1 \leq i \leq p} W_i\). By choice of \(v\), \(W_{\hat{i}} \leq \frac{3}{4}\). By choice of \(\hat{i}\),
\[
W_{\hat{i}} \geq \frac{1}{2} \sum_{i=1}^{p} W_i = \frac{1}{2} \left( \frac{3}{4} - w(v_0) \right) \geq \frac{1}{2} \left( \frac{3}{4} - \frac{1}{4} \right) = \frac{W}{4}
\]

This shows that choosing \(\hat{e}\) to be the edge \(v_{\hat{i}}v\) satisfies the balance condition. \(\square\)

The following example shows that the factor three-fourths in Lemma 1.3.2 cannot be improved upon.

By changing our goal slightly, we can get a better-balanced separator.

**Lemma 1.3.3 (Tree edge separator of Vertex/Edge Weight).** Let \(T\) be a binary tree, and let \(w(\cdot)\) be an assignment of weights to the vertices and edges such that, for each vertex \(v\), the weight assigned to \(v\) is at most \(\frac{1}{3}(\text{degree}(v) - 1)\) times the total weight. There is a linear-time algorithm to find an edge \(e\) such that every tree in \(T - \{e\}\) has at most two-thirds of the weight.

**Problem 1.3.4.** Prove Lemma 1.3.3.
1.4 Recursive tree decomposition

Problem 1.4.1. A recursive edge-separator decomposition for an unrooted tree $T$ is a rooted tree $D$ such that

- the root $r$ of $D$ is labeled with an edge $e$ of $T$;
- for each connected component $K$ of $T - e$ (there are at most two), $r$ has a child in $D$ that is the root of a recursive edge-separator decomposition of $K$.

Show that there is an $O(n \log n)$ algorithm that, given a tree $T$ of maximum degree three and $n$ nodes, returns a recursive edge-separator decomposition of depth $O(\log n)$.

1.5 Data structure for sequences and rooted trees

In the Appendix, we describe data structures for representing sequences and rooted trees.

Problem 1.5.1. Show that the data structure for representing trees can be used to quickly find edge-separators in binary trees. Use this idea to give a fast algorithm that, given a tree of maximum degree three, returns a recursive edge-separator decomposition of depth $O(\log n)$. Note: A running time of $O(n)$ can be achieved.