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 forest on $N$ is defined by a pair $(N,p)$ where $p$ is a function $p : N \rightarrow N \cup \{\perp\}$ such that there is no positive integer $k$ such that $p^k(x) = x$ for some element $x \in N$.

The elements of $N$ are nodes of the forest (also called vertices but we sometimes prefer node over vertex to emphasize these are elements of a rooted forest). A node $x$ such that $p(x) = \perp$ is a root. Each forest has at least one root; it is a tree if it contains only one root.

For each nonroot node $x$, $p(x)$ is called the parent of $x$ in the tree and $x$ is called a child of $p(x)$, and the ordered pair $xp(x)$ is called the parent edge of $x$ and a child edge of $p(x)$. The edges of the forest are the parent edges of its nodes. A node with no children is a leaf.

A rooted tree has arity $k$ if every node has at most $k$ children. A binary tree is a rooted tree that has arity two.

We say two nodes are adjacent if one is the parent of the other. We say the edge $xp(x)$ 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 ancestors 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 edge $xp(x)$ is an ancestor edge of $y$ if $x$ is an ancestor of $y$. We say $xp(x)$ is a descendant edge of $y$ if $p(x)$ is a descendant of $y$.

A subforest/subtree of $(N,p)$ is a forest/tree $(N',p')$ such that $N'$ is a subset of $N$, and $p'$ is the restriction of $p$ to $N'$.

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

$$p'(x) = \begin{cases} \perp & \text{if } x = \hat{x} \\ p(x) & \text{otherwise} \end{cases}$$
If $F$ is a rooted forest and $e$ is an edge of $F$ then we use $F - \{e\}$ 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 $F$ is a rooted forest and $\hat{x}$ is a node of $F$ then we use $F - \{\hat{x}\}$ to denote the result of deleting $\hat{x}$.

More generally, if $S$ is a set of nodes or a set of edges, $F - S$ denotes the forest obtained from $F$ 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 $x_0, x_1, \ldots, x_k$ 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 $u$:

```python
def totalWeight(u):
    return w(u) + \sum \{ totalWeight(v) : v \text{ a child of } u \}
```

We call this a rootward computation since the order of nodes for which it computes results is consistent with the rootward partial order: children before parents. 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 carving width (Section 14.3.1) and bounded branchwidth (Section 14.5.1).

Note that totalWeight($u$) must iterate through the children of $u$, whereas the formal definition of a forest provides only a way to go from a node to its parent. For this algorithm to be efficient, therefore, it should be proceeded by another rootward computation in which a table is constructed that maps
1.2 Separators for rooted trees

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

**Lemma 1.2.1** (Leafmost Heavy Node). Let $T$ be a rooted tree. Let $\hat{w}(\cdot)$ be an assignment of weights to vertices such that the weight of each node 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 exists a node $v_0$ 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 $T$.

```plaintext
define LeafmostHeavyNode(v):
  1 if some child $u$ of $v$ has $\hat{w}(u) > \alpha W$,
  2 return LeafmostHeavyNode(u)
  3 else return $v$
```

By induction on the number of invocations, for every call LeafmostHeavyNode($v$), 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 node returned by the procedure. Since the condition in Line 1 did not hold for $v_0$, every child $v$ of $v_0$ satisfies $\hat{w}(v) \leq \alpha W$. \hfill \Box

1.2.1 Node separator

**Lemma 1.2.2** (Tree Node 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 $\hat{w}(u) = \sum \{w(v) : v a descendant of u\}$. 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 is a descendant of $v_0\}$. Since the sum $\sum w(v)$ is a descendant of $v_0$, $w(v) = \hat{w}(v_0)$ exceeds $W/2$, the weight of the remaining tree is less than $W/2$. \hfill \Box
Figure 1.1: A rooted tree. Suppose the vertices are each assigned weight 1. The gray node is a separator whose deletion results in a forest, each of whose trees has weight at most half the total weight. The dashed edge is a separator whose deletion results in a forest, each of whose trees has weight at most three-fourths of the total weight.

1.3 Edge separators

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

Proof. Assume for notational simplicity that the total weight is 1. 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 node of the Leafmost-Heavy-Node Lemma with $\alpha = 1/3$. Since $T$ is binary, $v_0$ is not the root of $T$. 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 $\sum_{i} \hat{w}(v_i) \leq 2/3$. \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 results in remaining weight \( (k - 1)/k \).

The following example shows that, for binary trees, 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 Node Weight). Let \( T \) be a binary tree, and let \( w(\cdot) \) be a \( \frac{3}{4} \)-proper assignment of weights to vertices such that each nonleaf node is assigned at most one-fourth of the weight. There is 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. For each node \( v \), define

\[
\hat{w}(v) = \sum \{w(v') : v' \text{ a descendant of } v\}
\]

Let \( v \) be the node of the Leafmost-Heavy-Node Lemma with \( \alpha = 3/4 \). Let \( v_1, \ldots, v_p \) be the children of \( v \). Note that \( p \leq 2 \). Since \( w(v) \leq \frac{3}{4} \) but \( \hat{w}(v) > \frac{3}{4} \), 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 \( v \), \( W_{\hat{i}} \leq \frac{1}{4} \). By choice of \( \hat{i} \),

\[
W_{\hat{i}} \geq \frac{1}{2} \sum_{i=1}^{p} W_i > \frac{1}{2} \left( \frac{3}{4} - w(v) \right) \geq \frac{1}{2} \left( \frac{3}{4} - \frac{1}{4} \right) = W/4
\]

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

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

**Problem 1.1.** Prove Lemma 1.3.3.
1.4 Computation time for finding separators

For Lemmas 1.3.1 through 1.3.2, the weight assignment \( \hat{w}(\cdot) \) can be obtained from \( w(\cdot) \) via a rootward computation, and the linear-time implementation of \textsc{LeafmostHeavyNode} can be employed.

1.4.1 Recursive tree decomposition

In the Appendix, we describe data structures for representing sequences and rooted trees. These can be used to preprocess a rooted tree so as to find recursive separators.

**Problem 1.2.** A recursive edge-separator decomposition for a rooted 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 binary tree \( T \) with \( n \) nodes, returns a recursive edge-separator decomposition of depth \( O(\log n) \).

**Problem 1.3.** 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.