The arborescences of G' are in one-to-one correspondence with arborescences of G *that have exactly one edge entering the cycle* C; and these corresponding arborescences have the same cost with respect to $\{c'_e\}$, since C consists of 0-cost edges. (We say that an edge $e = (u, v)$ enters C if v belongs to C but u does not.) So to prove that our algorithm finds an optimal arborescence in G, we must prove that G has an optimal arborescence with exactly one edge entering C. We do this now.

(4.38) *Let C be a cycle in G consisting of edges of cost 0, such that $r \notin C$. Then there is an optimal arborescence rooted at r that has exactly one edge entering C.*

Proof. Consider an optimal arborescence T in G. Since r has a path in T to every node, there is at least one edge of T that enters C. If T enters C exactly once, then we are done. Otherwise, suppose that T enters C more than once. We show how to modify it to obtain an arborescence of no greater cost that enters C exactly once.

Let $e = (a, b)$ be an edge entering C that lies on as short a path as possible from r; this means in particular that no edges on the path from r to a can enter C. We delete all edges of T that enter C, except for the edge e. We add in all edges of C except for the one edge that enters b, the head of edge e. Let T' denote the resulting subgraph of G.

We claim that T' is also an arborescence. This will establish the result, since the cost of T' is clearly no greater than that of T: the only edges of T' that do not also belong to T have cost 0. So why is T' an arborescence? Observe that T' has exactly one edge entering each node $v \neq r$, and no edge entering r. So T' has exactly $n - 1$ edges; hence if we can show there is an r-v path in T' for each v, then T' must be connected in an undirected sense, and hence a tree. Thus it would satisfy our initial definition of an arborescence.

So consider any node $v \neq r$; we must show there is an r-v path in T'. If $v \in C$, we can use the fact that the path in T from r to e has been preserved in the construction of T'; thus we can reach v by first reaching e and then following the edges of the cycle C. Now suppose that $v \notin C$, and let P denote the r-v path in T. If P did not touch C, then it still exists in T'. Otherwise, let w be the last node in $P \cap C$, and let P' be the subpath of P from w to v. Observe that all the edges in P' still exist in T'. We have already argued that w is reachable from r in T', since it belongs to C. Concatenating this path to w with the subpath P' gives us a path to v as well. ∎

We can now put all the pieces together to argue that our algorithm is correct.

(4.39) *The algorithm finds an optimal arborescence rooted at r in G.*

Proof. The proof is by induction on the number of nodes in G. If the edges of F form an arborescence, then the algorithm returns an optimal arborescence by (4.36). Otherwise, we consider the problem with the modified costs $\{c'_e\}$, which is equivalent by (4.37). After contracting a 0-cost cycle C to obtain a smaller graph G', the algorithm produces an optimal arborescence in G' by the inductive hypothesis. Finally, by (4.38), there is an optimal arborescence in G that corresponds to the optimal arborescence computed for G'. ∎

Solved Exercises

Solved Exercise 1

Suppose that three of your friends, inspired by repeated viewings of the horror-movie phenomenon *The Blair Witch Project*, have decided to hike the Appalachian Trail this summer. They want to hike as much as possible per day but, for obvious reasons, not after dark. On a map they've identified a large set of good *stopping points* for camping, and they're considering the following system for deciding when to stop for the day. Each time they come to a potential stopping point, they determine whether they can make it to the next one before nightfall. If they can make it, then they keep hiking; otherwise, they stop.

Despite many significant drawbacks, they claim this system does have one good feature. "Given that we're only hiking in the daylight," they claim, "it minimizes the number of camping stops we have to make."

Is this true? The proposed system is a greedy algorithm, and we wish to determine whether it minimizes the number of stops needed.

To make this question precise, let's make the following set of simplifying assumptions. We'll model the Appalachian Trail as a long line segment of length L, and assume that your friends can hike d miles per day (independent of terrain, weather conditions, and so forth). We'll assume that the potential stopping points are located at distances $x_1, x_2, \ldots, x_n$ from the start of the trail. We'll also assume (very generously) that your friends are always correct when they estimate whether they can make it to the next stopping point before nightfall.

We'll say that a set of stopping points is *valid* if the distance between each adjacent pair is at most d, the first is at distance at most d from the start of the trail, and the last is at distance at most d from the end of the trail. Thus a set of stopping points is valid if one could camp only at these places and

still make it across the whole trail. We'll assume, naturally, that the full set of n stopping points is valid; otherwise, there would be no way to make it the whole way.

We can now state the question as follows. Is your friends' greedy algorithm—hiking as long as possible each day—*optimal*, in the sense that it finds a valid set whose size is as small as possible?

Solution Often a greedy algorithm looks correct when you first encounter it, so before succumbing too deeply to its intuitive appeal, it's useful to ask: why might it not work? What should we be worried about?

There's a natural concern with this algorithm: Might it not help to stop early on some day, so as to get better synchronized with camping opportunities on future days? But if you think about it, you start to wonder whether this could really happen. Could there really be an alternate solution that intentionally lags behind the greedy solution, and then puts on a burst of speed and passes the greedy solution? How could it pass it, given that the greedy solution travels as far as possible each day?

This last consideration starts to look like the outline of an argument based on the "staying ahead" principle from Section 4.1. Perhaps we can show that as long as the greedy camping strategy is ahead on a given day, no other solution can catch up and overtake it the next day.

We now turn this into a proof showing the algorithm is indeed optimal, identifying a natural sense in which the stopping points it chooses "stay ahead" of any other legal set of stopping points. Although we are following the style of proof from Section 4.1, it's worth noting an interesting contrast with the Interval Scheduling Problem: there we needed to prove that a greedy algorithm maximized a quantity of interest, whereas here we seek to minimize a certain quantity.

Let $R = \{x_{p_1}, \ldots, x_{p_k}\}$ denote the set of stopping points chosen by the greedy algorithm, and suppose by way of contradiction that there is a smaller valid set of stopping points; let's call this smaller set $S = \{x_{q_1}, \ldots, x_{q_m}\}$, with $m < k$.

To obtain a contradiction, we first show that the stopping point reached by the greedy algorithm on each day j is farther than the stopping point reached under the alternate solution. That is,

(4.40) *For each $j = 1, 2, \ldots, m$, we have $x_{p_j} \geq x_{q_j}$.*

Proof. We prove this by induction on j. The case $j = 1$ follows directly from the definition of the greedy algorithm: your friends travel as long as possible on the first day before stopping. Now let $j > 1$ and assume that the claim is true for all $i < j$. Then

$$x_{q_j} - x_{q_{j-1}} \leq d,$$

since S is a valid set of stopping points, and

$$x_{q_j} - x_{p_{j-1}} \leq x_{q_j} - x_{q_{j-1}}$$

since $x_{p_{j-1}} \geq x_{q_{j-1}}$ by the induction hypothesis. Combining these two inequalities, we have

$$x_{q_j} - x_{p_{j-1}} \leq d.$$

This means that your friends have the option of hiking all the way from $x_{p_{j-1}}$ to x_{q_j} in one day; and hence the location x_{p_j} at which they finally stop can only be farther along than x_{q_j}. (Note the similarity with the corresponding proof for the Interval Scheduling Problem: here too the greedy algorithm is staying ahead because, at each step, the choice made by the alternate solution is one of its valid options.) ■

Statement (4.40) implies in particular that $x_{q_m} \leq x_{p_m}$. Now, if $m < k$, then we must have $x_{p_m} < L - d$, for otherwise your friends would never have needed to stop at the location $x_{p_{m+1}}$. Combining these two inequalities, we have concluded that $x_{q_m} < L - d$; but this contradicts the assumption that S is a valid set of stopping points.

Consequently, we cannot have $m < k$, and so we have proved that the greedy algorithm produces a valid set of stopping points of minimum possible size.

Solved Exercise 2

Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month.

Each license is currently selling for a price of \$100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100 \cdot r_j^t$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the same price of \$100).