Pearson New International Edition

Algorithm Design

Jon Kleinberg Éva Tardos

First Edition

ALWAYS LEARNING

PEARSON

Pearson New International Edition

Algorithm Design

Jon Kleinberg   Éva Tardos

First Edition

PEARSON

leaf v, and if u is the parent of v in T, then $h(u) \geq h(v)$. We place each point in P at a distinct leaf in T. Now, for any pair of points p_i and p_j, their distance $\tau(p_i, p_j)$ is defined as follows. We determine the least common ancestor v in T of the leaves containing p_i and p_j, and define $\tau(p_i, p_j) = h_v$.

We say that a hierarchical metric τ is *consistent* with our distance function d if, for all pairs i, j, we have $\tau(p_i, p_j) \leq d(p_i, p_j)$.

Give a polynomial-time algorithm that takes the distance function d and produces a hierarchical metric τ with the following properties.

(i) τ is consistent with d, and

(ii) if τ' is any other hierarchical metric consistent with d, then $\tau'(p_i, p_j) \leq \tau(p_i, p_j)$ for each pair of points p_i and p_j.

26. One of the first things you learn in calculus is how to minimize a differentiable function such as $y = ax^2 + bx + c$, where $a > 0$. The Minimum Spanning Tree Problem, on the other hand, is a minimization problem of a very different flavor: there are now just a finite number of possibilities for how the minimum might be achieved—rather than a continuum of possibilities—and we are interested in how to perform the computation without having to exhaust this (huge) finite number of possibilities.

One can ask what happens when these two minimization issues are brought together, and the following question is an example of this. Suppose we have a connected graph $G = (V, E)$. Each edge e now has a *time-varying edge cost* given by a function $f_e : \mathbf{R} \to \mathbf{R}$. Thus, at time t, it has cost $f_e(t)$. We'll assume that all these functions are positive over their entire range. Observe that the set of edges constituting the minimum spanning tree of G may change over time. Also, of course, the cost of the minimum spanning tree of G becomes a function of the time t; we'll denote this function $c_G(t)$. A natural problem then becomes: find a value of t at which $c_G(t)$ is minimized.

Suppose each function f_e is a polynomial of degree 2: $f_e(t) = a_e t^2 + b_e t + c_e$, where $a_e > 0$. Give an algorithm that takes the graph G and the values $\{(a_e, b_e, c_e) : e \in E\}$ and returns a value of the time t at which the minimum spanning tree has minimum cost. Your algorithm should run in time polynomial in the number of nodes and edges of the graph G. You may assume that arithmetic operations on the numbers $\{(a_e, b_e, c_e)\}$ can be done in constant time per operation.

27. In trying to understand the combinatorial structure of spanning trees, we can consider the space of *all* possible spanning trees of a given graph and study the properties of this space. This is a strategy that has been applied to many similar problems as well.

Here is one way to do this. Let G be a connected graph, and T and T' two different spanning trees of G. We say that T and T' are *neighbors* if T contains exactly one edge that is not in T', and T' contains exactly one edge that is not in T.

Now, from any graph G, we can build a (large) graph $\mathcal{H}$ as follows. The nodes of $\mathcal{H}$ are the spanning trees of G, and there is an edge between two nodes of $\mathcal{H}$ if the corresponding spanning trees are neighbors.

Is it true that, for any connected graph G, the resulting graph $\mathcal{H}$ is connected? Give a proof that $\mathcal{H}$ is always connected, or provide an example (with explanation) of a connected graph G for which $\mathcal{H}$ is not connected.

28. Suppose you're a consultant for the networking company CluNet, and they have the following problem. The network that they're currently working on is modeled by a connected graph $G = (V, E)$ with n nodes. Each edge e is a fiber-optic cable that is owned by one of two companies—creatively named X and Y—and leased to CluNet.

Their plan is to choose a spanning tree T of G and upgrade the links corresponding to the edges of T. Their business relations people have already concluded an agreement with companies X and Y stipulating a number k so that in the tree T that is chosen, k of the edges will be owned by X and $n - k - 1$ of the edges will be owned by Y.

CluNet management now faces the following problem. It is not at all clear to them whether there even *exists* a spanning tree T meeting these conditions, or how to find one if it exists. So this is the problem they put to you: Give a polynomial-time algorithm that takes G, with each edge labeled X or Y, and either (i) returns a spanning tree with exactly k edges labeled X, or (ii) reports correctly that no such tree exists.

29. Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, show how to decide in polynomial time whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \ldots, d_n$. (That is, if $V = \{v_1, v_2, \ldots, v_n\}$, then the degree of v_i should be exactly d_i.) G should not contain multiple edges between the same pair of nodes, or "loop" edges with both endpoints equal to the same node.

30. Let $G = (V, E)$ be a graph with n nodes in which each pair of nodes is joined by an edge. There is a positive weight w_{ij} on each edge (i, j); and we will assume these weights satisfy the *triangle inequality* $w_{ik} \leq w_{ij} + w_{jk}$. For a subset $V' \subseteq V$, we will use $G[V']$ to denote the subgraph (with edge weights) induced on the nodes in V'.

We are given a set $X \subseteq V$ of k *terminals* that must be connected by edges. We say that a *Steiner tree* on X is a set Z so that $X \subseteq Z \subseteq V$, together with a spanning subtree T of $G[Z]$. The *weight* of the Steiner tree is the weight of the tree T.

Show that the problem of finding a minimum-weight Steiner tree on X can be solved in time $O(n^{O(k)})$.

31. Let's go back to the original motivation for the Minimum Spanning Tree Problem. We are given a connected, undirected graph $G = (V, E)$ with positive edge lengths $\{\ell_e\}$, and we want to find a spanning subgraph of it. Now suppose we are willing to settle for a subgraph $H = (V, F)$ that is "denser" than a tree, and we are interested in guaranteeing that, for each pair of vertices $u, v \in V$, the length of the shortest u-v path in H is not much longer than the length of the shortest u-v path in G. By the *length* of a path P here, we mean the sum of ℓ_e over all edges e in P.

Here's a variant of Kruskal's Algorithm designed to produce such a subgraph.

- First we sort all the edges in order of increasing length. (You may assume all edge lengths are distinct.)
- We then construct a subgraph $H = (V, F)$ by considering each edge in order.
- When we come to edge $e = (u, v)$, we add e to the subgraph H if there is currently no u-v path in H. (This is what Kruskal's Algorithm would do as well.) On the other hand, if there is a u-v path in H, we let d_{uv} denote the length of the shortest such path; again, length is with respect to the values $\{\ell_e\}$. We add e to H if $3\ell_e < d_{uv}$.

In other words, we add an edge even when u and v are already in the same connected component, provided that the addition of the edge reduces their shortest-path distance by a sufficient amount.

Let $H = (V, F)$ be the subgraph of G returned by the algorithm.

(a) Prove that for every pair of nodes $u, v \in V$, the length of the shortest u-v path in H is at most three times the length of the shortest u-v path in G.

(b) Despite its ability to approximately preserve shortest-path distances, the subgraph H produced by the algorithm cannot be too dense. Let $f(n)$ denote the maximum number of edges that can possibly be produced as the output of this algorithm, over all n-node input graphs with edge lengths. Prove that

$$\lim_{n\to\infty} \frac{f(n)}{n^2} = 0.$$

32. Consider a directed graph $G = (V, E)$ with a root $r \in V$ and nonnegative costs on the edges. In this problem we consider variants of the minimum-cost arborescence algorithm.

(a) The algorithm discussed in Section 4.9 works as follows. We modify the costs, consider the subgraph of zero-cost edges, look for a directed cycle in this subgraph, and contract it (if one exists). Argue briefly that instead of looking for cycles, we can instead identify and contract strong components of this subgraph.

(b) In the course of the algorithm, we defined y_v to be the minimum cost of an edge entering v, and we modified the costs of all edges e entering node v to be $c'_e = c_e - y_v$. Suppose we instead use the following modified cost: $c''_e = \max(0, c_e - 2y_v)$. This new change is likely to turn more edges to 0 cost. Suppose now we find an arborescence T of 0 cost. Prove that this T has cost at most twice the cost of the minimum-cost arborescence in the original graph.

(c) Assume you do not find an arborescence of 0 cost. Contract all 0-cost strong components and recursively apply the same procedure on the resulting graph until an arborescence is found. Prove that this T has cost at most twice the cost of the minimum-cost arborescence in the original graph.

33. Suppose you are given a directed graph $G = (V, E)$ in which each edge has a cost of either 0 or 1. Also suppose that G has a node r such that there is a path from r to every other node in G. You are also given an integer k. Give a polynomial-time algorithm that either constructs an arborescence rooted at r of cost *exactly* k, or reports (correctly) that no such arborescence exists.

Notes and Further Reading

Due to their conceptual cleanness and intuitive appeal, greedy algorithms have a long history and many applications throughout computer science. In this chapter we focused on cases in which greedy algorithms find the optimal solution. Greedy algorithms are also often used as simple heuristics even when they are not guaranteed to find the optimal solution. In Chapter 11 we will discuss greedy algorithms that find near-optimal approximate solutions.

As discussed in Chapter 1, Interval Scheduling can be viewed as a special case of the Independent Set Problem on a graph that represents the overlaps among a collection of intervals. Graphs arising this way are called *interval graphs*, and they have been extensively studied; see, for example, the book by Golumbic (1980). Not just Independent Set but many hard computational problems become much more tractable when restricted to the special case of interval graphs.

Interval Scheduling and the problem of scheduling to minimize the maximum lateness are two of a range of basic scheduling problems for which a simple greedy algorithm can be shown to produce an optimal solution. A wealth of related problems can be found in the survey by Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993).

The optimal algorithm for caching and its analysis are due to Belady (1966). As we mentioned in the text, under real operating conditions caching algorithms must make eviction decisions in real time without knowledge of future requests. We will discuss such caching strategies in Chapter 13.

The algorithm for shortest paths in a graph with nonnegative edge lengths is due to Dijkstra (1959). Surveys of approaches to the Minimum Spanning Tree Problem, together with historical background, can be found in the reviews by Graham and Hell (1985) and Nesetril (1997).

The single-link algorithm is one of the most widely used approaches to the general problem of clustering; the books by Anderberg (1973), Duda, Hart, and Stork (2001), and Jain and Dubes (1981) survey a variety of clustering techniques.

The algorithm for optimal prefix codes is due to Huffman (1952); the earlier approaches mentioned in the text appear in the books by Fano (1949) and Shannon and Weaver (1949). General overviews of the area of data compression can be found in the book by Bell, Cleary, and Witten (1990) and the survey by Lelewer and Hirschberg (1987). More generally, this topic belongs to the area of *information theory*, which is concerned with the representation and encoding of digital information. One of the founding works in this field is the book by Shannon and Weaver (1949), and the more recent textbook by Cover and Thomas (1991) provides detailed coverage of the subject.

The algorithm for finding minimum-cost arborescences is generally credited to Chu and Liu (1965) and to Edmonds (1967) independently. As discussed in the chapter, this multi-phase approach stretches our notion of what constitutes a greedy algorithm. It is also important from the perspective of linear programming, since in that context it can be viewed as a fundamental application of the *pricing method*, or the *primal-dual* technique, for designing algorithms. The book by Nemhauser and Wolsey (1988) develops these connections to linear programming. We will discuss this method in Chapter 11 in the context of approximation algorithms.

More generally, as we discussed at the outset of the chapter, it is hard to find a precise definition of what constitutes a greedy algorithm. In the search for such a definition, it is not even clear that one can apply the analogue of U.S. Supreme Court Justice Potter Stewart's famous test for obscenity—"I know it when I see it"—since one finds disagreements within the research community on what constitutes the boundary, even intuitively, between greedy and nongreedy algorithms. There has been research aimed at formalizing classes of greedy algorithms: the theory of *matroids* is one very influential example (Edmonds 1971; Lawler 2001); and the paper of Borodin, Nielsen, and Rackoff (2002) formalizes notions of greedy and "greedy-type" algorithms, as well as providing a comparison to other formal work on this question.

Notes on the Exercises Exercise 24 is based on results of M. Edahiro, T. Chao, Y. Hsu, J. Ho, K. Boese, and A. Kahng; Exercise 31 is based on a result of Ingo Althofer, Gautam Das, David Dobkin, and Deborah Joseph.