Exercises

Problem

Suppose you are given a connected graph G, with edge costs that you may assume are all distinct. G has n vertices and m edges. A particular edge e of G is specified. Give an algorithm with running time O(n + m) to decide whether e is contained in the minimum spanning tree of G.

▲ロト ▲帰 ト ▲ ヨ ト ▲ ヨ ト ・ ヨ ・ の Q ()

Use the cut property and the Cycle property. Both properties are essentially talking about how *e* related to the set of edges that are *cheaper* than *e*. The Cut property can be viewed as asking: Is there some set $S \subseteq V$ so that in order to get from S to $V \setminus S$ without using *e*, we need to use an edge that is more expensive than *e*? The Cycle property says that an alternative route between two endpoints of *e* that uses cheaper edges.

Theorem

Edge e = (v, w) does not belong to a minimum spanning tree of G if and only if v and w can be joined by a path consisting entirely of edges that are cheaper than e.

Given this fact, we form a graph G' by deleting from G all edges of weight greater that w_e (as well as deleting e itself). We then use one of the connectivity algorithm (BFS or DFS) to determine whether there is a path from v to w in G'. Above theorem says that e belongs to a minimum spanning tree if and only if there is no such path.

Consider techniques for coordinating groups of mobile robots. Each robot has a radio transmitter that it uses to communicate with a base station, and your friend find that if the robots get too close to one another, then there are problems with *interference* among the transmitters. So a natural problem arises:

Problem

How to plan the motion of the robots in such a way that each robot gets to its intended destination, but in the process the robots do not come close enough together to cause interference problems.

We can model the problem abstractly as follows. Suppose that we have an undirected graph G = (V, E), representing the floor plan of a building, and there are two robots initially located at nodes a and b in the graph. The robot at node a wants to travel to node c along a path in G, and the robot at node b wants to travel to node d. This is accomplished by means of a *schedule*: at each time step, the schedule specifies that one of the robots moves across a single edge, from one node to a neighboring node; at the end of this schedule, the robot from a should be sitting on c, and the robot from b should be sitting on d. A *schedule* is *interference-free* if there is no point at which the two robots occupy nodes that are at a distance $\leq r$ from one another in the graph, for a given parameter r. We will assume that the two starting nodes a and b are at a distance greater than r, and so are the two ending nodes c and d.

Problem

Given a polynomial-time algorithm that decides whether there exists an interference-free schedule by which each robot can get to its destination.

Problem

Given an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be O(n + m) for a graph with n nodes and m edges.

Inspired by the example of that great Cornellian, Vladimir Nabokov, some of your friends have become amateur lepidopterist (they study bufferflies). Often when they return from a trip with specimens of bufferflies, it is very difficult for them to tell how many distinct species they have caught — thanks to the fact that many species look very similar to one another.

One day they return with n bufferflies, and they believe that each belongs to one of two different species, which we will call A and B for purposes of this discussion. They would like to divide the n specimens into two groups — those that belong to A and those that belong to B — but it is very hard for them to directly label any one specimen. So they decide to adopt the following approach.

For each pair of specimens i and j, they study them carefully side by side. If they are confident enough in their judgement, then they label the pair (i, j)either "same" or "different". They also have the option of rendering no judgement on a given pair, in which case we will call the pair "ambiguous". We will declare m judgement "consistent" if it is possible to label each specimen either A or B in such a way that for each pair (i, j) labeled "same", it is the case the i and j have the same label; and for each pair (i, j) labeled "different", it is the case that i and j have different labels.

Problem

Given an algorithm with running time O(m + n) that determines whether the m judgements are consistent.

We have a connected graph G = (V, E), and a specified vertex $u \in V$. Suppose we compute a depth-first search tree rooted at u, and obtain a tree T that includes all nodes of G. Suppose we then compute a breath-first search tree rooted at u, and obtain the same tree T.

Theorem

Prove that G = T.

(In other words, if T is both a depth-first search tree and a breath-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)

< □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > <

Some friends of yours work on wireless networks, and they are currently studying the prosperities of n mobile devices. As the devices move around, they define a graph at any point in time as follows: there is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

They would like it to be the case the network of devices is connected at all times, and so they have constrained the motion of the devices to satisfy the following property: at all times, each device *i* is within 500 meters of at least $\frac{n}{2}$ of the other devices. (We will assume *n* is an even number.) What they would like to know is: Does this property by itself guarantees that the network will remain connected?

Here is a concrete way to formulate the question as a claim about graphs:

Problem

Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least $\frac{n}{2}$, then G is connected.

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Problem What is the answer if we change $\frac{n}{2}$ to $\frac{n}{2} - 1$?

Definition (Distance, dist(u, v).)

The distance between two nodes u and v in a graph G = (V, E) is the minimum number of edges in a path joining them.

Definition (Diameter, diam(G).)

The maximum distance between any pair of nodes in G.

Definition (Average pairwise distance in G, apd(G).)

apd(G) is the average, over all pairs of two distinct nodes u and v, of the distance between u and v.

$$\mathsf{apd}(\mathsf{G}) = rac{\sum_{\{u, \ v\} \in V} \mathsf{dist}(u, \ v)}{\binom{n}{2}}.$$

Problem

There exists a positive natural number c so that for all connected graphs G, it is the case that

$$\frac{diam(G)}{apd(G)} \leq c.$$

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Suppose that an *n*-node undirected graph G = (V, E) contains two nodes *s* and *t* such that the distance (the number of hops on the path s - t) between *s* and *t* is strictly greater than $\frac{n}{2}$.

Problem

Show that there must exists some node v, not equal to either s or t, such that deleting v from G destroys all s - t paths.

(In other words, the graph obtained from G by deleting v contains no path from s to t.) Give an algorithm with running time O(m + n) to find such a node v.

A number of art museums around the country have been featuring by an artist named Mark Lombardi (1951 - 2000), consisting of a set of intricately rendered celestial-like graphs. Building on a great deal of research, these graphs encode the relationship among people involved in major political scandals over the past several decades: the nodes correspond to participants; and each edge indicates some type of relationship between a pair of participants. And so, if you peer closely enough at the drawings, you can trace out ominous-looking paths form a high-ranking US government officials, to a former business parter, to a bank of Switzerland, to a shadowy arms dealer. Such picture form striking example of *social networks*, which have nodes representing people and organizations, and edges representing relationships of various kinds. And the short paths that abound in these networks have attracted considerable attention recently, as people ponder what they mean. In the case of Mark Lombardi's graphs, they hint at the shorter set of steps that can carry you form the reputable to the disreputable.

< □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > <

Of course, a single, spurious short path between node v and w in such a network may be more coincidental than anything else; a large number of short paths between v and w can be much more convincing. So in addition to the problem of computing a single shortest v - w path in a graph G, social networks researchers have looked at the problem of determining the number of shortest v - w paths. This turns out to be problem that can be solved efficiently.

Problem

Suppose we are given an undirected graph G = (V, E), and we identify two nodes v and w in G. Given an algorithm that computes the number of shortest v - w paths in G. (This algorithm should not list all the paths; just the number suffices.) The running time of your algorithm should be O(n + m) for a graph with n nodes and m edges.

You are helping some security analysts monitor a collection of networked computers, tracking the spread of an online virus. There are *n* computers in the system, labeled C_1, C_2, \ldots, C_n , and as input you are given a collection of *trace data* indicating the times at which pairs of computers communicated. Thus the data is a sequence of ordered triples (C_i, C_j, t_k) ; such a triple indicates that C_i and C_j exchanges bits at time t_k . There are *m* triples total. We will assume that the triples are presented to you in sorted order of time. For purposes of simplicity, we will assume that each pair of computers communication at most once during the interval you are observing. The security analysts you are working with would like to be able to answer questions of the following form:

Problem

If the virus was inserted into computer C_a at time x, could it possibly have infected computer C_b by time y?

The mechanics of infection are simple: if an infected computer C_i communicates with an uninfected computer C_j at time t_k (in other words, if one of the triples (C_i, C_j, t_k) or (C_j, C_i, t_k) appears in the trace data), then computer C_j becomes infected as well, starting at time t_k . Infection can thus spread from one machine to another across a sequence of communications, provided that no step in this sequence involves a move backward in time.

Problem

Design an algorithm: given a collection of trace data, the algorithm should decide whether a virus introduced at computer C_a at time x could have infected computer C_b by time y. That algorithm should run in time O(m + n).

You are helping a group of ethnographers analyze some oral history data they have collected by interviewing members of a village to learn about the lives of people who have lived there over the past two hundred years. From these interviews, they have learnt about a set of *n* people (all of them now decreased), whom we will denote P_1, P_2, \ldots, P_n . They have also collected facts about when these people lived relative to one another. Each fact has one of the following two forms:

1. For some i and j, person P_i died before person P_j was born; or

2. for some *i* and *j*, the life spans of P_i and P_j overlapped at least partially.

Naturally, they are not sure that all these facts are correct; memories are not so good, and a lot of this was passed down by words of mouth. So what they have like you to determine is whether the data they have collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they have learnt simultaneously hold.

Problem

Given an efficient algorithm to do this: either it should produce proposed dates of birth and death for each of the n people so that all the facts hold true, or it should report (correctly) that no such data can exist – that is, the facts collected by the ethnographers are not internally consistent.

Problem

Let G = (V, E) be an undirected graph with costs $c_e \ge 0$ on the edges $e \in E$. Assume you are given a minimum-cost spanning tree T in G. Now assume that a new edge is added to G, connecting two nodes v, $w \in V$ with cost c.

- Give an efficient algorithm to test if T remains the minimum-cost spanning tree with the new edge added to G (but not the tree T). Make your algorithm run in time O(|E|). Can you do it in O(|V|) time?
- 2. Suppose T is no longer the minimum-cost spanning tree. Give a linear-time algorithm (time O(|E|)) to update the tree T to the new minimum-cost spanning tree.

< □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > < □ > <

Problem

Compute a minimum-cost arborescence of a directed graph. Let G = (V, E) be a directed graph in which $r \in V$ is a root. An arborescence (with respect to r) is essentially a directed spanning tree rooted at r. Specifically, it is a subgraph T = (V, F) such that T is a spanning tree of G if we ignore the directions of edges; and there is a path in T from r to each other node $v \in V$ if we take the directions of edges into account.

A subgraph T = (V, F) of G is an arborescence with respect to root r if and only if T has no cycles, and for each node $v \neq r$, there is exactly one edge in F that enters v.

▲□▶ ▲□▶ ▲□▶ ▲□▶ □ のQ@

Proof.

A subgraph T = (V, F) of G is an arborescence with respect to root r if and only if T has no cycles, and for each node $v \neq r$, there is exactly one edge in F that enters v.

Proof.

1. \Rightarrow . If T is an arborescence with root r, then indeed every other node v has exactly one edge entering it: this is simply the last edge on the unique r - v path.

A subgraph T = (V, F) of G is an arborescence with respect to root r if and only if T has no cycles, and for each node $v \neq r$, there is exactly one edge in F that enters v.

Proof.

- 1. \Rightarrow . If *T* is an arborescence with root *r*, then indeed every other node *v* has exactly one edge entering it: this is simply the last edge on the unique r v path.
- 2. \Leftarrow . Suppose *T* has no cycles, and each node $v \neq r$ has exactly one entering edge.

In order to establish that T is an arborescence, we need to only show that there is a directed path from r to each other node v: We start at v and repeatedly follow edges in the backward direction. Since T has no cycles, we can never return to a node we have previously visited, and thus this process must terminate. But r is the only node without incoming edges, and so the process must in fact terminate by reaching r; the sequence of nodes thus visited yields a path (in the reverse direction) from r to v.

- 1. Can we apply the ideas directly we developed for the minimum-spanning tree to this setting?
- 2. Must the minimum-cost arborescence contain the cheapest edge in the whole graph?
- 3. Can we safely delete the most expensive edge on a cycle, confident that it cannot be in the optimal arborescence?

◆□▶ ◆□▶ ◆三▶ ◆三▶ 三三 のへぐ

- For each node v ≠ r, select the cheapest edge entering v (breaking ties arbitrarily), and let F* be this set of n 1 edges. Now, consider the subgraph (V, F*). Since the optimal arborescence needs to have exactly one edge entering each node v ≠ r, and (V, F*) represents the cheapest possible way of making these choices.
- 2. If (V, F^*) is an arborescence, we are done (why?). If (V, F^*) is not an arborescence, it must contain a cycle C.
- Every arborescence contains exactly one edge entering each node v ≠ r; so if we pick some node v and subtract a uniform quantity from the cost of every edge entering v, then the total cost of every arborescence changes by exactly the same amount.

This means, essentially, that the actual cost of the cheapest edge entering v is not important; what matters is the cost of all other edges entering v relative to this.

4. Let y_v denote the minimum cost of any edge entering v. For each edge e = (u, v), with cost $c_e \ge 0$, we define its modified cost c'_e to be $c_e - y_v$, all the modified costs are still nonnegative.

Theorem

T is an optimal arborescence in G subject to costs $\{c_e\}$ if and only if it is an optimal arborescence subject to the modified costs $\{c'_e\}$.

Proof.

The total difference $\sum_{e \in T} c_e - \sum_{e \in T} c'_e = \sum_{v \neq r} y_v$.

All the edges in set F* (for each node v ≠ r, select the cheapest edge entering v — breaking ties arbitrarily, and let F* be this set of n - 1 edges) have cost 0 under the modified costs.
If (V, F*) contains a cycle C, and all edges in C have cost 0. This

suggest that we can afford to use as many edges from C as we want (consistent with producing an arborescence), since including edges from C does not raise the cost.

- We contract C into a single supernode, obtaining a smaller graph G' = (V', E'), V' contains the nodes V \ C, plus a single node c* representing C. We transform each edge e ∈ E to an edge e' ∈ E' by replacing each end of e that belongs to C with a new node c*. This can result in G' having parallel edges. We delete self-loops from E'.
- 3. We recursively find an optimal arborescence in this smaller graph G', subject to the cost $\{c'_e\}$. The arborescence returned by this recursive call can be converted into an arborescence of G by including all but one edge on the cycle C.

- 1. For each node $v \neq r$
- 2. Let y_v be the minimum cost of an edge entering node v
- 3. Modify the costs of all edges e entering v to $c'_e = c_e y_v$.
- 4. Choose one 0-cost edge entering each $v \neq r$, obtaining a set F^*

5. If F^* forms an arborescence, then return it

- 6. Else there is a directed cycle $C \subseteq F^*$
- 7. Contract C to a single supernode, yielding a graph G' = (V', E')
- 8. Recursively find an optimal arborescence (V', F') in G' with costs $\{c'_e\}$
- 9. Extend (V', F') to an arborescence (V, F) in G by adding all but one edge of C

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.

Let T be an optimal arborescence 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, we are done. Otherwise, we show how to modify T 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*. 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*. *T'* is also an arborescence (why?). (1) *T'* has exactly one edge entering each node $v \neq r$, and no edge entering *r*.

So T' has exactly n-1 edges.

(2) Consider any node $v \neq r$, we show there is a r - v path in T'. If $v \in C$, the path in T from r to e is preserved in T'. If $v \notin C$, 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 sub-path of P from w to v. Observe that all the edges in P' still exist in T'. We know w is reachable from r in T', since $w \in C$. Concatenating this path to w with the sub-path P' gives us a path to v as well.

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.

T' is an arborescence.

Proof.

T' has exactly one edge entering each node $v \neq r$, and no edge entering r. So T' has exactly n-1 edges.

Consider any node $v \neq r$, we show there is a r - v path in T'.

If $v \in C$, the path in T from r to e is preserved in T'.

If $v \notin C$, 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 sub-path of *P* from *w* to *v*. Observe that all the edges in *P'* still exist in *T'*. *w* is reachable from *r* in *T'*, since $w \in C$. Concatenating this path to *w* with the sub-path *P'* gives us a path to *v* as well.

Theorem

The algorithm finds an optimal arborescence rooted at r in G.

Proof.

Mathematical induction...

Cayley's Theorem

Definition (Labeled Tree.)

A *labeled tree* is a tree in which each node is labeled with a distinct integer. (Without loss of generality, we let them be 1, 2, ..., n.)



From a labeled tree with at least 2 nodes we can remove the leaf with the lowest number, together with the edge connecting it to the rest of the tree; the remaining graph is again a labeled tree. Hence this action can be repeated until the tree has been reduced to a single node; that remaining node is the one with the maximum label number.

For instance, each time we have written the number of the leaf being removed in the upper line:

 $2 \hspace{0.1in} 3 \hspace{0.1in} 4 \hspace{0.1in} 0 \hspace{0.1in} 5 \hspace{0.1in} 6 \hspace{0.1in} 1 \hspace{1.5in} \text{the node removed in order}$

1 7 0 7 1 1 7 the other end of the edge when one of its nodes is removed

▲□▶ ▲圖▶ ▲臣▶ ▲臣▶ ―臣 … のへで

- I_1 . The right-most value of the bottom line (i.e., node 7) is always the highest node number.
- I_2 . The top line is always a permutation of the remaining node numbers (all nodes except node 7).
- 13. The number of times that a value occurs in such a scheme equals the degree of the corresponding node (at that time when a node is removed).

When we remove from such a scheme the top line and the right-most column in both top and bottom lines — in our example

$$1 7 0 7 1 1$$
 (1)

would be the result — each value has been removed once (on account of I_1 and I_2). On account of I_3 , we have (note nodes 2, 3, 4, 5, 6 are leaves)

Lemma

The number of times that a value occurs in such a sequence (i.e., the bottom line) is one less than the degree of the corresponding nodes. Hence the leaves are the nodes whose number is missing in the sequence.

Theorem

The left-most element of the top line is the minimum index value from the set that is missing in the sequence. (All other leaves have higher index values but they are less than the maximum one.)

Because for a tree of n nodes the sequence (as in Formula (1)) is n-2elements long, at least 2 values are missing; therefore that minimum missing value never equals the maximum node number (in our example, $2 \neq 7$), i.e., the construction is possible for any sequence of n-2 node numbers. The next value in the top line is found by the same argument after reducing the set of node numbers by removing from it the value just filled in (in our example, 2), and reducing the sequence by the removal of its left-most element. By repeating the argument, the first n-2 elements of the top line can be reconstructed. Finally, of the right-most column the bottom element is reconstructible on account of I_1 and its top element by I_2 . Hence, for a set of *n* labeled nodes, there is a one-to-one correspondence between the trees connecting these nodes and the sequences (i.e., the bottom line) of n-2 node numbers. Because the number of such sequences is obviously equal to

 n^{n-2} ,

this value also equals the number of possible trees connecting n labeled nodes.

(日) (同) (三) (三) (三) (○) (○)