Algorithms (CS2443) : Problem Set 3

Department of Computer Science and Engineering
IIT Hyderabad

Please read the following comment before you work on the problems.

This is a set of questions that you should solve to master the contents taught in the class. *You don't have to submit the solutions*! But you should still solve them to understand the material, and also because these problems or their variants have a devilish way of finding their way in exams/quizzes!

We suggest to try to solve them from scratch under exam conditions—by yourself, *without* your notes, *without* the internet, and if possible, even without a cheat sheet.

If you find yourself getting stuck on a problem, try to figure out *why* you're stuck. Do you understand the problem statement? Are you stuck on choosing the right high-level approach, or are you stuck on the details? Are you solving the right (recursive) subproblems?

Discussing problems with other people (in your study groups, with me, or on Google Classroom) and/or looking up old solutions can be *extremely* helpful, but **only after** you have (1) made a good-faith effort to solve the problem on your own, and (2) you have either a candidate solution or some idea about where you are getting stuck.

When you do discuss problems with other people, remember that your goal is not merely to "understand" the solution to any particular problem, but to become more comfortable with solving a certain type of problem on your own. Identify specific steps that you find problematic, read more about those steps, and focus your practice on those steps.

---

1. Consider the directed acyclic graph G show below (Figure 1). How many topological orderings does it have?

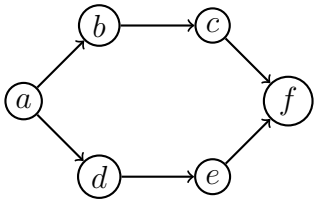

Figure 1: Directed acyclic graph G

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

3. The algorithm described in the class for computing a topological ordering of a DAG repeatedly finds a node with no incoming edges and deletes it. This will eventually produce a topological ordering, provided that the input graph really is a DAG.

   But suppose that we're given an arbitrary directed graph that may or may not be a DAG. Extend the topological ordering algorithm so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG; or (b) a directed cycle in G, thus establishing that G is not a DAG. The running time of your algorithm should be $O(m+n)$ for a directed graph with n nodes and m edges.

4. We have a connected undirected graph $G = (V, E)$, and a specific 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 breadth-first search tree rooted at u, and obtain the same tree T. Prove that $G = T$. (In other words, if T is both a depth-first search tree and a breadth-first search tree rooted at u, then G cannot contain any edges that do not belong to T.)

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

   Claim: Let G be an undirected graph on n nodes, where n is an even number. If every node of G has degree at least $n/2$, then G is connected.

6. Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist 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.

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

8. You're helping a group of ethnographers analyze some oral history data they've collected by interviewing members of a village to learn about the lives of people who've lived there over the past two hundred years.

   From these interviews, they've learned about a set of n people (all of them now deceased), whom we'll denote $P_1, P_2, \ldots, P_n$. They've also collected facts about when

these people lived relative to one another. Each fact has one of the following two forms:

(a) For some i and j, person P_i died before person P_j was born; or

(b) For some i and j, the life spans of P_i and P_j overlapped at least partially.

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

Give an efficient algorithm to do this: either it should report 'Consistent' if there could exist dates of birth and death for each of the n people so that all the facts hold true, or it should report 'Inconsistent' in case no such dates can exist—that is, the facts collected by the ethnographers are not internally consistent.

9. You're 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're 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 exchanged bits at time t_k. There are m triples total.

   We'll assume that the triples are presented to you in sorted order of time. For purposes of simplicity, we'll assume that each pair of computers communicates at most once during the interval you're observing.

   The security analysts you're working with would like to be able to answer questions of the following form: 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. Thus, for example, if C_i is infected by time t_k, and the trace data contains triples (C_i, C_j, t_k) and (C_j, C_q, t_r), where $t_k \leq t_r$, then C_q will become infected via C_j. (Note that it is okay for t_k to be equal to t_r; this would mean that C_j had open connections to both C_i and C_q at the same time, and so a virus could move from C_i to C_q.)

   For example, suppose $n = 4$, the trace data consists of the triples

   $$(C_1, C_2, 4), (C_2, C_4, 8), (C_3, C_4, 8), (C_1, C_4, 12),$$

   and the virus was inserted into computer C_1 at time 2. Then C_3 would be infected at time 8 by a sequence of three steps: first C_2 becomes infected at time 4, then C_4 gets

the virus from C_2 at time 8, and then C_3 gets the virus from C_4 at time 8. On the other hand, if the trace data were

$$(C_2, C_3, 8), (C_1, C_4, 12), (C_1, C_2, 14),$$

and again the virus was inserted into computer C_1 at time 2, then C_3 would not become infected during the period of observation: although C_2 becomes infected at time 14, we see that C_3 only communicates with C_2 before C_2 was infected. There is no sequence of communications moving forward in time by which the virus could get from C_1 to C_3 in this second example.

Design an algorithm that answers questions of this type: 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. The algorithm should run in time $O(m+n)$.

10. 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 $O(m + n)$ to decide whether e is contained in a minimum spanning tree of G.

11. Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

    Let G be an arbitrary connected, undirected graph with a distinct cost $c(e)$ on every edge e. Suppose e^* is the cheapest edge in G; that is, $c(e^*) < c(e)$ for every edge $e \neq e^*$. Then there is a minimum spanning tree T of G that contains the edge e^*.

12. Suppose you are given a connected graph G, with edge costs that are all distinct. Prove that G has a unique minimum spanning tree.

13. One of the basic motivations behind the Minimum Spanning Tree Problem is the goal of designing a spanning network for a set of nodes with minimum *total cost*. Here we explore another type of objective: designing a spanning network for which the *most expensive edge* is as cheap as possible.

    Specifically, let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of G; we define the *bottleneck edge* of T to be an edge of T with the greatest cost.

    A spanning tree T of G is a *minimum-bottleneck spanning tree* if there is no spanning tree T' of G with a cheaper bottleneck edge.

    (a) Is every minimum-bottleneck tree of G a minimum spanning tree of G? Prove or give a counterexample.

    (b) Is every minimum spanning tree of G a minimum-bottleneck tree of G? Prove or give a counterexample.

14. Let $G = (V, E)$ be an (undirected) graph with costs $c_e \geq 0$ 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.

    (a) Give an efficient algorithm to test if T remains the minimum-cost spanning tree with the new edge added to G (but not to the tree T). Make your algorithm run in time $O(|E|)$.

    (b) 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.

15. Suppose you are given a connected graph $G = (V, E)$, with a cost $c_e \geq 0$ on each edge e. In an earlier problem, we saw that when all edge costs are distinct, G has a unique minimum spanning tree. However, G may have many minimum spanning trees when the edge costs are not all distinct. Here we formulate the question: Can Kruskal's Algorithm be made to find all the minimum spanning trees of G?

    Recall that Kruskal's Algorithm sorted the edges in order of increasing cost, then greedily processed edges one by one, adding an edge e as long as it did not form a cycle. When some edges have the same cost, the phrase "in order of increasing cost" has to be specified a little more carefully: we'll say that an ordering of the edges is *valid* if the corresponding sequence of edge costs is nondecreasing. We'll say that a *valid execution* of Kruskal's Algorithm is one that begins with a valid ordering of the edges of G.

    For any graph G, and any minimum spanning tree T of G, is there a valid execution of Kruskal's Algorithm on G that produces T as output? Give a proof or a counterexample.

16. Your friends are planning an expedition to a small town deep in the Canadian north next winter break. They've researched all the travel options and have drawn up a directed graph whose nodes represent intermediate destinations and edges represent the roads between them.

    In the course of this, they've also learned that extreme weather causes roads in this part of the world to become quite slow in the winter and may cause large travel delays. They've found an excellent travel Web site that can accurately predict how fast they'll be able to travel along the roads; however, the speed of travel depends on the time of year. More precisely, the Web site answers queries of the following form: given an edge $e = (v, w)$ connecting two sites v and w, and given a proposed starting time t from location v, the site will return a value $f_e(t)$, the predicted arrival time at w. The Web site guarantees that $f_e(t) \geq t$ for all edges e and all times t (you can't travel backward in time), and that $f_e(t)$ is a monotone increasing function of t (that is, you do not arrive earlier by starting later). Other than that, the functions $f_e(t)$ may be arbitrary. For example, in areas where the travel time does not vary with the season, we would

have $f_e(t) = t + \ell_e$, where ℓ_e is the time needed to travel from the beginning to the end of edge e.

Your friends want to use the Web site to determine the fastest way to travel through the directed graph from their starting point to their intended destination. (You should assume that they start at time 0, and that all predictions made by the Web site are completely correct.) Give a polynomial time algorithm to do this, where we treat a single query to the Web site (based on a specific edge e and a time t) as taking a single computational step.

17. A group of network designers at the communications company CluNet find themselves facing the following problem. They have a connected graph $G = (V, E)$, in which the nodes represent sites that want to communicate. Each edge e is a communication link, with a given available bandwidth b_e.

    For each pair of nodes $u, v \in V$, they want to select a single $u - v$ path P on which this pair will communicate. The *bottleneck rate* $b(P)$ of this path P is the minimum bandwidth of any edge it contains; that is, $b(P) = \min_{e \in P} b_e$. The *best achievable bottleneck rate* for the pair u, v in G is simply the maximum, over all $u - v$ paths P in G, of the value $b(P)$.

    It's getting to be very complicated to keep track of a path for each pair of nodes, and so one of the network designers makes a bold suggestion: Maybe one can find a spanning tree T of G so that for every pair of nodes u, v, the unique $u - v$ path in the tree actually attains the best achievable bottleneck rate for u, v in G. (In other words, even if you could choose any $u - v$ path in the whole graph, you couldn't do better than the $u - v$ path in T.)

    This idea is roundly heckled in the offices of CluNet for a few days, and there's a natural reason for the skepticism: each pair of nodes might want a very different-looking path to maximize its bottleneck rate; why should there be a single tree that simultaneously makes everybody happy? But after some failed attempts to rule out the idea, people begin to suspect it could be possible.

    Show that such a tree exists, and give an efficient algorithm to find one. That is, give an algorithm constructing a spanning tree T in which, for each $u, v \in V$, the bottleneck rate of the $u - v$ path in T is equal to the best achievable bottleneck rate for the pair u, v in G.

18. Let us say that a graph $G = (V, E)$ is a *near-tree* if it is connected and has at most $n + 8$ edges, where $n = |V|$. Give an algorithm with running time $O(n)$ that takes a near-tree G with costs on its edges, and returns a minimum spanning tree of G. You may assume that all the edge costs are distinct.

19. Consider the Minimum Spanning Tree Problem on an undirected graph $G = (V, E)$, with a cost $c_e \geq 0$ on each edge, where the costs may not all be different. If the costs are not all distinct, there can in general be many distinct minimum-cost solutions.

Suppose we are given a spanning tree $T \subseteq E$ with the guarantee that for every $e \in T$, e belongs to some minimum-cost spanning tree in G. Can we conclude that T itself must be a minimum-cost spanning tree in G? Give a proof or a counterexample with explanation.

20. 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.

21. 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 $f_e : \mathbb{R} \to \mathbb{R}$. 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.

22. 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.

23. To assess how "well-connected" two nodes in a directed graph are, one can not only look at the length of the shortest path between them, but can also count the *number* of shortest paths.

    This turns out to be a problem that can be solved efficiently, subject to some restrictions on the edge costs. Suppose we are given a directed graph $G = (V, E)$, with costs on the edges; the costs may be positive or negative, but every cycle in the graph has strictly positive cost. We are also given two nodes $v, w \in V$. Give an efficient algorithm that computes the number of shortest $v - w$ paths in G. (The algorithm should not list all the paths; just the number suffices.)

24. Suppose you are given a directed graph $G = (V, E)$ with costs on the edges c_e for $e \in E$ and a sink t (costs may be negative). Assume that you also have finite values $d(v)$ for $v \in V$. Someone claims that, for each node $v \in V$, the quantity $d(v)$ is the cost of the minimum-cost path from node v to the sink t.

    (a) Give a linear-time algorithm (time $O(m)$ if the graph has m edges) that verifies whether this claim is correct.

    (b) Assume that the distances are correct, and $d(v)$ is finite for all $v \in V$. Now you need to compute distances to a different sink t'. Give an $O(m \log n)$ algorithm for computing distances $d'(v)$ for all nodes $v \in V$ to the sink node t'. (Hint: It is useful to consider a new cost function defined as follows: for edge $e = (v, w)$, let $c'_e = c_e - d(v) + d(w)$. Is there a relation between costs of paths for the two different costs c and c'?)

25. Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e, a designated source $s \in V$, and a designated sink $t \in V$. You are also given an integer maximum $s - t$ flow in G, defined by a flow value f_e on each edge e.

    Now suppose we pick a specific edge $e \in E$ and increase its capacity by one unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$, where m is the number of edges in G and n is the number of nodes.

26. Suppose you're a consultant for the Ergonomic Architecture Commission, and they come to you with the following problem.

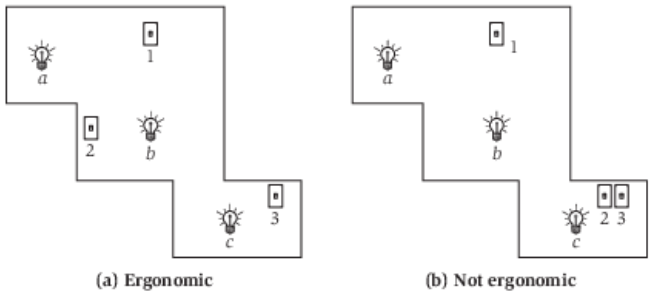

Figure 2: The floor plan in (a) is ergonomic, because we can wire switches to fixtures in such a way that each fixture is visible from the switch that controls it. (This can be done by wiring switch 1 to a, switch 2 to b, and switch 3 to c.) The floor plan in (b) is not ergonomic, because no such wiring is possible.

They're really concerned about designing houses that are "user friendly", and they've been having a lot of trouble with the setup of light fixtures and switches in newly designed houses. Consider, for example, a one-floor house with n light fixtures and n locations for light switches mounted in the wall. You'd like to be able to wire up one switch to control each light fixture, in such a way that a person at the switch can see the light fixture being controlled.

Sometimes this is possible and sometimes it isn't. Consider the two simple floor plans for houses in Figure 2. There are three light fixtures (labeled a, b, c) and three switches (labeled 1, 2, 3). It is possible to wire switches to fixtures in Figure 2 (a) so that every switch has a line of sight to the fixture, but this is not possible in Figure 2 (b).

Let's call a floor plan, together with n light fixture locations and n switch locations, ergonomic if it's possible to wire one switch to each fixture so that every fixture is visible from the switch that controls it. A floor plan will be represented by a set of m horizontal or vertical line segments in the plane (the walls), where the i-th wall has endpoints $(x_i, y_i), (x'_i, y'_i)$. Each of the n switches and each of the n fixtures is given by its coordinates in the plane. A fixture is visible from a switch if the line segment joining them does not cross any of the walls.

Give an algorithm to decide if a given floor plan is ergonomic. The running time should be polynomial in m and n. You may assume that you have a subroutine with $O(1)$ running time that takes two line segments as input and decides whether or not they cross in the plane.

27. Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible *base stations*. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There

are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways.

There is a *range parameter* r—a client can only be connected to a base station that is within distance r. There is also a *load parameter* L—no more than L clients can be connected to any single base station.

Your goal is to design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station, subject to the range and load conditions in the previous paragraph.

28. Statistically, the arrival of spring typically results in increased accidents and increased need for emergency medical treatment, which often requires blood transfusions. Consider the problem faced by a hospital that is trying to evaluate whether its blood supply is sufficient.

    The basic rule for blood donation is the following. A person's own blood supply has certain *antigens* present (we can think of antigens as a kind of molecular signature); and a person cannot receive blood with a particular antigen if their own blood does not have this antigen present. Concretely, this principle underpins the division of blood into four *types*: A, B, AB, and O. Blood of type A has the A antigen, blood of type B has the B antigen, blood of type AB has both, and blood of type O has neither. Thus, patients with type A can receive only blood types A or O in a transfusion, patients with type B can receive only B or O, patients with type O can receive only O, and patients with type AB can receive any of the four types.

    Let s_O, s_A, s_B, and s_{AB} denote the supply in whole units of the different blood types on hand. Assume that the hospital knows the projected demand for each blood type d_O, d_A, d_B, and d_{AB} for the coming week. Give a polynomial-time algorithm to evaluate if the blood on hand would suffice for the projected need.

29. Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time.

    Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now).

    At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to

collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most $\lceil n/k \rceil$ people.

Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

30. Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity $c(e)$ on each edge e, a source $s \in V$, and a sink $t \in V$. You are also given a maximum $s - t$ flow f in G, defined by a flow value $f(e)$ on each edge e. The flow f is *acyclic*: There is no cycle in G on which all edges carry positive flow. The flow f is also integer-valued.

    Now suppose we pick a specific edge $e^* \in E$ and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$, where m is the number of edges in G and n is the number of nodes.

31. Your friends have written a very fast piece of maximum-flow code based on repeatedly finding augmenting paths as taught in the class! However, after you've looked at a bit of output from it, you realize that it's not always finding a flow of maximum value. The bug turns out to be pretty easy to find; your friends hadn't really gotten into the whole backward-edge thing when writing the code, and so their implementation builds a variant of the residual graph that only includes the forward edges. In other words, it searches for $s - t$ paths in a graph $\tilde{G}_f$ consisting only of edges e for which $f(e) < c(e)$, and it terminates when there is no augmenting path consisting entirely of such edges. We'll call this the Forward-Edge-Only Algorithm. (Note that we do not try to prescribe how this algorithm chooses its forward-edge paths; it may choose them in any fashion it wants, provided that it terminates only when there are no forward-edge paths.)

    It's hard to convince your friends they need to reimplement the code. In addition to its blazing speed, they claim, in fact, that it never returns a flow whose value is less than a fixed fraction of optimal. Do you believe this? The crux of their claim can be made precise in the following statement.

    > *There is an absolute constant $b > 1$ (independent of the particular input flow network), so that on every instance of the Maximum-Flow Problem, the Forward-Edge-Only Algorithm is guaranteed to find a flow of value at least $1/b$ times the maximum-flow value (regardless of how it chooses its forward-edge paths).*

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

32. Consider the following problem. You are given a flow network with unit-capacity edges: It consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c(e) = 1$ for every $e \in E$. You are also given a parameter k.

The goal is to delete k edges so as to reduce the maximum $s-t$ flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum $s-t$ flow in $G' = (V, E \setminus F)$ is as small as possible subject to this.

Give a polynomial-time algorithm to solve this problem.

33. In a standard s-t Maximum-Flow Problem, we assume edges have capacities, and there is no limit on how much flow is allowed to pass through a node. In this problem, we consider the variant of the Maximum-Flow and Minimum-Cut problems with node capacities.

    Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative node capacities $\{c(v) \geq 0\}$ for each $v \in V$. Given a flow f in this graph, the flow though a node v is defined as $f^{in}(v)$. We say that a flow is feasible if it satisfies the usual flow-conservation constraints and the node-capacity constraints: $f^{in}(v) \leq c(v)$ for all nodes.

    Give a polynomial-time algorithm to find an s-t maximum flow in such a node-capacitated network. Define an s-t cut for node-capacitated networks, and show that the analogue of the Max-Flow Min-Cut Theorem holds true.