Chapter 34

Exercises - Network Flow

By Sariel Har-Peled, September 25, 2014[1] Version: 1.02

This chapter include problems that are realted to network flow.

34.1 Network Flow

34.1.1 The good, the bad, and the middle.

(10 PTS.)

Suppose you're looking at a flow network G with source s and sink t, and you want to be able to express something like the following intuitive notion: Some nodes are clearly on the "source side" of the main bottlenecks; some nodes are clearly on the "sink side" of the main bottlenecks; and some nodes are in the middle. However, G can have many minimum cuts, so we have to be careful in how we try making this idea precise.

Here's one way to divide the nodes of G into three categories of this sort.

- We say a node v is *upstream* if, for all minimum s-t cuts (A, B), we have $v \in A$ – that is, v lies on the source side of every minimum cut.
- We say a node v is *downstream* if, for all minimum s-t cuts (A, B), we have $v \in B$ – that is, v lies on the sink side of every minimum cut.
- We say a node v is *central* if it is neither upstream nor downstream; there is at least one minimum s-t cut (A, B) for which $v \in A$, and at least one minimum s-t cut (A', B') for which $v \in B'$.

Give an algorithm that takes a flow network G and classifies each of its nodes as being upstream, downstream, or central. The running time of your algorithm should be within a constant factor of the time required to compute a *single* maximum flow.

[1]This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

34.1.2 Ad hoc networks

(20 PTS.)

Ad hoc networks are made up of low-powered wireless devices, have been proposed for situations like natural disasters in which the coordinators of a rescue effort might want to monitor conditions in a hard-to-reach area. The idea is that a large collection of these wireless devices could be dropped into such an area from an airplane and then configured into a functioning network.

Note that we're talking about (a) relatively inexpensive devices that are (b) being dropped from an airplane into (c) dangerous territory; and for the combination of reasons (a), (b), and (c), it becomes necessary to include provisions for dealing with the failure of a reasonable number of the nodes.

We'd like it to be the case that if one of the devices v detects that it is in danger of failing, it should transmit a representation of its current state to some other device in the network. Each device has a limited transmitting range – say it can communicate with other devices that lie within d meters of it. Moreover, since we don't want it to try transmitting its state to a device that has already failed, we should include some redundancy: A device v should have a set of k other devices that it can potentially contact, each within d meters of it. We'll call this a *back-up set* for device v.

1. Suppose you're given a set of n wireless devices, with positions represented by an (x, y) coordinate pair for each. Design an algorithm that determines whether it is possible to choose a back-up set for each device (i.e., k other devices, each within d meters), with the further property that, for some parameter b, no device appears in the back-up set of more than b other devices. The algorithm should output the back-up sets themselves, provided they can be found.

2. The idea that, for each pair of devices v and w, there's a strict dichotomy between being "in range" or "out of range" is a simplified abstraction. More accurately, there's a power decay function $f(\cdot)$ that specifies, for a pair of devices at distance δ, the signal strength $f(\delta)$ that they'll be able to achieve on their wireless connection. (We'll assume that $f(\delta)$ decreases with increasing δ.)

   We might want to build this into our notion of back-up sets as follows: among the k devices in the back-up set of v, there should be at least one that can be reached with very high signal strength, at least one other that can be reached with moderately high signal strength, and so forth. More concretely, we have values $p_1 \geq p_2 \geq \cdots \geq p_k$, so that if the back-up set for v consists of devices at distances $d_1 \leq d_2 \leq \cdots \leq d_k$, then we should have $f(d_j) \geq p_j$ for each j.

   Give an algorithm that determines whether it is possible to choose a back-up set for each device subject to this more detailed condition, still requiring that no device should appear in the back-up set of more than b other devices. Again, the algorithm should output the back-up sets themselves, provided they can be found.

34.1.3 Minimum Flow

(10 PTS.)

Give a polynomial-time algorithm for the following minimization analogue of the Maximum-Flow Problem. You are given a directed graph $G = (V, E)$, with a source $s \in V$ and sink $t \in V$, and numbers (capacities) $\ell(v, w)$ for each edge $(v, w) \in E$. We define a flow f, and the value of a flow, as usual, requiring that all nodes except s and t satisfy flow conservation. However, the given numbers are lower bounds on edge flow – that is, they require that $f(v, w) \geq \ell(v, w)$ for every edge $(v, w) \in E$, and there is no upper bound on flow values on edges.

1. Give a polynomial-time algorithm that finds a feasible flow of minimum possible values.

2. Prove an analogue of the Max-Flow Min-Cut Theorem for this problem (i.e., does min-flow = max-cut?).

34.1.4 Prove infeasibility.

You are trying to solve a circulation problem, but it is not feasible. The problem has demands, but no capacity limits on the edges. More formally, there is a graph $G = (V, E)$, and demands d_v for each node v (satisfying $\sum_{v \in V} d_v = 0$), and the problem is to decide if there is a flow f such that $f(e) \geq 0$ and $f^{in}(v) - f^{out}(v) = d_v$ for all nodes $v \in V$. Note that this problem can be solved via the circulation algorithm from Section 7.7 by setting $c_e = +\infty$ for all edges $e \in E$. (Alternately, it is enough to set c_e to be an extremely large number for each edge – say, larger than the total of all positive demands d_v in the graph.)

You want to fix up the graph to make the problem feasible, so it would be very useful to know why the problem is not feasible as it stands now. On a closer look, you see that there is a subset U of nodes such that there is no edge into U, and yet $\sum_{v \in U} d_v > 0$. You quickly realize that the existence of such a set immediately implies that the flow cannot exist: The set U has a positive total demand, and so needs incoming flow, and yet U has no edges into it. In trying to evaluate how far the problem is from being solvable, you wonder how big the demand of a set with no incoming edges can be.

Give a polynomial-time algorithm to find a subset $S \subset V$ of nodes such that there is no edge into S and for which $\sum_{v \in S} d_v$ is as large as possible subject to this condition.

34.1.5 Cellphones and services.

Consider an assignment problem where we have a set of n stations that can provide service, and there is a set of k requests for service. Say, for example, that the stations are cell towers and the requests are cell phones. Each request can be served by a given set of stations. The problem so far can be represented by a bipartite graph G: one side is the stations, the other the customers, and there is an edge (x, y) between customer x and station y if customer x can be served from station y. Assume that each station can serve at most one customer. Using a max-flow computation, we can decide whether or not all customers can be served, or can get an assignment of a subset of customers to stations maximizing the number of served customers.

Here we consider a version of the problem with an addition complication: Each customer offers a different amount of money for the service. Let U be the set of customers, and assume that customer $x \in U$ is willing to pay $v_x \geq 0$ for being served. Now the goal is to find a subset $X \subset U$ maximizing $\sum_{x \in X} v_x$ such that there is an assignment of the customers in X to stations.

Consider the following greedy approach. We process customers in order of decreasing value (breaking ties arbitrarily). When considering customer x the algorithm will either "promise" service to x or reject x in the following greedy fashion. Let X be the set of customers that so far have been promised service. We add x to the set X if and only if there is a way to assign $X \cup \{x\}$ to servers, and we reject x otherwise. Note that rejected customers will not be considered later. (This is viewed as an advantage: If we need to reject a high-paying customer, at least we can tell him/her early.) However, we do not assign accepting customers to servers in a greedy fashion: we only fix the assignment after the set of accepted customers is fixed. Does this greedy approach produce an optimal set of customers? Prove that it does, or provide a counterexample.

34.1.6 Follow the stars

(20 PTS.)

Some friends of yours have grown tired of the game "Six Degrees of Kevin Bacon" (after all, they ask, isn't it just breadth-first search?) and decide to invent a game with a little more punch, algorithmically speaking. Here's how it works.

You start with a set X of n actresses and a set Y of n actors, and two players P_0 and P_1. Player P_0 names an actress $x_1 \in X$, player P_1 names an actor y_1 who has appeared in a movie with x_1, player P_0 names an actress x_2 who has appeared in a movie with y_1, and so on. Thus, P_0 and P_1 collectively generate a sequence $x_1, y_1, x_2, y_2, \ldots$ such that each actor/actress in the sequence has costarred with the actress/actor immediately preceding. A player P_i ($i = 0, 1$) loses when it is P_i's turn to move, and he/she cannot name a member of his/her set who hasn't been named before.

Suppose you are given a specific pair of such sets X and Y, with complete information on who has appeared in a movie with whom. A *strategy* for P_i, in our setting, is an algorithm that takes a current sequence $x_1, y_1, x_2, y_2, \ldots$ and generates a legal next move for P_i (assuming it's P_i's turn to move). Give a polynomial-time algorithm that decides which of the two players can force a win, in a particular instance of this game.

34.1.7 Flooding

(10 pts.)

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.

34.1.8 Capacitation, yeh, yeh, yeh

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 a maximum s-t flow in G, defined by a flow value f_e on each edge e. The flow $\{f_e\}$ is *acyclic*: There is no cycle in G on which all edges carry positive flow.

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.

34.1.9 Fast Friends

(20 pts.)

Your friends have written a very fast piece of maximum-flow code based on repeatedly finding augmenting paths as in the course lecture notes. 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 forwards edges*. In other words, it searches for s-t paths in a graph $\tilde{G}_f$ consisting only of edges of 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 ot prescribe how this algorithms 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.

34.1.10 Even More Capacitation

(10 PTS.)

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 through 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.

34.1.11 Matrices

(10 PTS.)

Let M be an $n \times n$ matrix with each entry equal to either 0 or 1. Let m_{ij} denote the entry in row i and column j. A *diagonal entry* is one of the form m_{ii} for some i.

Swapping rows i and j of the matrix M denotes the following action: we swap the values of m_{ik} and m_{jk}, for $k = 1, \ldots, n$. Swapping two columns is defined analogously.

We say that M is ***rearrangeable*** if it is possible to swap some of the pairs of rows and some of the pairs of columns (in nay sequence) so that after all the swapping, all the diagonal entries of M are equal to 1.

1. (2 PTS.) Give an example of a matrix M that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1.

2. (8 PTS.) Give a polynomial-time algorithm that determines whether a matrix M with 0-1 entries is rearrangeable.

34.1.12 Unique Cut

(10 PTS.)

Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative edge capacities $\{c_e\}$. Give a polynomial-time algorithm to decide whether G has a *unique* minimum s-t cut (i.e., an s-t of capacity strictly less than that of all other s-t cuts).

34.1.13 Transitivity

(10 PTS.)

Given a graph $G = (V, E)$, and a natural number k, we can define a relation $\xrightarrow{G,k}$ on pairs of vertices of G as follows. If $x, y \in V$, we say that $x \xrightarrow{G,k} y$ if there exist k mutually edge-disjoint paths from x to y in G.

Is it true that for every G and every $k \geq 0$, the relation $\xrightarrow{G,k}$ is transitive? That is, is it always the case that if $x \xrightarrow{G,k} y$ and $y \xrightarrow{G,k} z$, then we have $x \xrightarrow{G,k} z$? Give a proof or a counterexample.

34.1.14 Census Rounding

(20 PTS.)

You are consulting for an environmental statistics firm. They collect statistics and publish the collected data in a book. The statistics are about populations of different regions in the world and are recorded in multiples of one million. Examples of such statistics would look like the following table.

Country	A	B	C	Total
grown-up men	11.998	9.083	2.919	24.000
grown-up women	12.983	10.872	3.145	27.000
children	1.019	2.045	0.936	4.000
total	26.000	22.000	7.000	55.000

We will assume here for simplicity that our data is such that all row and column sums are integers. The Census Rounding Problem is to round all data to integers without changing any row or column sum. Each fractional number can be rounded either up or down. For example, a good rounding for our table data would be as follows.

Country	A	B	C	Total
grown-up men	11.000	10.000	3.000	24.000
grown-up women	13.000	10.000	4.000	27.000
children	1.000	2.000	0.000	4.000
total	26.000	22.000	7.000	55.000

1. (5 PTS.) Consider first the special case when all data are between 0 and 1. So you have a matrix of fractional numbers between 0 and 1, and your problem is to round each fraction that is between 0 and 1 to either 0 or 1 without changing the row or column sums. Use a flow computation to check if the desired rounding is possible.

2. (5 PTS.) Consider the Census Rounding Problem as defined above, where row and column sums are integers, and you want to round each fractional number α to either $\lfloor \alpha \rfloor$ or $\lceil \alpha \rceil$. Use a flow computation to check if the desired rounding is possible.

3. (10 PTS.) Prove that the rounding we are looking for in (a) and (b) always exists.

34.1.15 Edge Connectivity

(20 PTS.)

The *edge connectivity* of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how the edge connectivity of an undirected graph $G = (V, E)$ can be determined by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.

34.1.16 Maximum Flow By Scaling

(20 PTS.)

Let $G = (V, E)$ be a flow network with source s, sink t, and an integer capacity $c(u, v)$ on each edge $(u, v) \in E$. Let $C = max_{(u,v)\in E} c(u, v)$.

1. (2 PTS.) Argue that a minimum cut of G has capacity at most $C|E|$.
2. (5 PTS.) For a given number K, show that an augmenting path of capacity at least K can be found in $O(E)$ time, if such a path exists.

   The following modification of FORD-FULKERSON-METHOD can be used to compute a maximum flow in G.

```
MAX-FLOW-BY-SCALING($G, s, t$)
1   $C \leftarrow max_{(u,v)\in E} c(u, v)$
2   initialize flow $f$ to 0
3   $K \leftarrow 2^{\lfloor \lg C \rfloor}$
4   while $K \geq 1$ do {
5       while (there exists an augmenting path $p$ of
                capacity at least $K$) do {
6           augment flow $f$ along $p$
        }
7       $K \leftarrow K/2$
    }

8   return $f$
```

3. (3 PTS.) Argue that MAX-FLOW-BY-SCALING returns a maximum flow.
4. (4 PTS.) Show that the capacity of a minimum cut of the residual graph G_f is at most $2K|E|$ each time line 4 is executed.
5. (4 PTS.) Argue that the inner **while** loop of lines 5-6 is executed $O(E)$ times for each value of K.
6. (2 PTS.) Conclude that MAX-FLOW-BY-SCALING can be implemented so that it runs in $O(E^2 \lg C)$ time.

34.1.17 Perfect Matching

(20 PTS.)

1. (10 PTS.) A *perfect matching* is a matching in which every vertex is matched. Let $G = (V, E)$ be an undirected bipartite graph with vertex partition $V = L \cup R$, where $|L| = |R|$. For any $X \subseteq V$, define the *neighborhood* of X as

$$N(X) = \left\{ y \in V \,\middle|\, (x, y) \in E \text{ for some } x \in X \right\},$$

   that is, the set of vertices adjacent to some member of X. Prove *Hall's theorem*: there exists a perfect matching in G if and only if $|A| \leq |N(A)|$ for every subset $A \subseteq L$.

2. (10 PTS.) We say that a bipartite graph $G = (V, E)$, where $V = L \cup R$, is *d-regular* if every vertex $v \in V$ has degree exactly d. Every d-regular bipartite graph has $|L| = |R|$. Prove that every d-regular bipartite graph has a matching of cardinality $|L|$ by arguing that a minimum cut of the corresponding flow network has capacity $|L|$.

34.1.18 Number of augmenting paths

1. (10 PTS.) Show that a maximum flow in a network $G = (V, E)$ can always be found by a sequence of at most $|E|$ augmenting paths. [Hint: Determine the paths after finding the maximum flow.]

2. (10 PTS.) Suppose that a flow network $G = (V, E)$ has symmetric edges, that is, $(u, v) \in E$ if and only $(v, u) \in E$. Show that the Edmonds-Karp algorithm terminates after at most $|V||E|/4$ iterations. [Hint: For any edge (u,v), consider how both $\delta(s, u)$ and $\delta(v, t)$ change between times at which (u, v) is critical.]

34.1.19 Minimum Cut Festival

(20 PTS.)

1. Given a multigraph $G(V, E)$, show that an edge can be selected uniform at random from E in time $O(n)$, given access to a source of random bits.

2. For any $\alpha \geq 1$, define an α approximate cut in a multigraph G as any cut whose cardinality is within a multiplicative factor α of the cardinality of the min-cut in G. Determine the probability that a single iteration of the randomized algorithm for cuts will produce as output some α-approximate cut in G.

3. Using the analysis of the randomized min-cut algorithm, show that the number of distinct min-cuts in a multigraph G cannot exceed $n(n-1)/2$, where n is the number of vertices in G.

4. Formulate and prove a similar result of the number of α -approximate cuts in a multigraph G.

34.1.20 Independence Matrix

(10 PTS.)

Consider a $0-1$ matrix H with n_1 rows and n_2 columns. We refer to a row or a column of the matrix H as a line. We say that a set of 1's in the matrix H is *independent* if no two of them appear in the same line. We also say that a set of lines in the matrix is a *cover* of H if they include (i.e., "cover") all the 1's in the matrix. Using the max-flow min-cut theorem on an appropriately defined network, show that the maximum number of independent 1's equals the minimum number of lines in the cover.

34.1.21 Scalar Flow Product

(10 PTS.)

Let f be a flow in a network, and let α be a real number. The *scalar flow product*, denoted by αf, is a function from $V \times V$ to $\mathbb{R}$ defined by

$$(\alpha f)(u, v) = \alpha \cdot f(u, v).$$

Prove that the flows in a network form a *convex set*. That is, show that if f_1 and f_2 are flows, then so is $\alpha f_1 + (1-\alpha) f_2$ for all α in the range $0 \leq \alpha \leq 1$.

34.1.22 Go to school!

Professor Adam has two children who, unfortunately, dislike each other. The problem is so severe that not only they refuse to walk to school together, but in fact each one refuses to walk on any block that the other child has stepped on that day. The children have no problem with their paths crossing at a corner. Fortunately both the professor's house and the school are on corners, but beyond that he is not sure if it is going to be possible to send both of his children to the same school. The professor has a map of his town. Show how to formulate the problem of determining if both his children can go to the same school as a maximum-flow problem.

34.1.23 The Hopcroft-Karp Bipartite Matching Algorithm

(20 pts.)

In this problem, we describe a faster algorithm, due to Hopcroft and Karp, for finding a maximum matching in a bipartite graph. The algorithm runs in $O(\sqrt{V}E)$ time. Given an undirected, bipartite graph $G = (V, E)$, where $V = L \cup R$ and all edges have exactly one endpoint in L, let M be a matching in G. We say that a simple path P in G is an *augmenting path* with respect to M if it starts at an unmatched vertex in L, ends at an unmatched vertex in R, and its edges belong alternatively to M and $E - M$. (This definition of an augmenting path is related to, but different from, an augmenting path in a flow network.) In this problem, we treat a path as a sequence of edges, rather than as a sequence of vertices. A shortest augmenting path with respect to a matching M is an augmenting path with a minimum number of edges.

Given two sets A and B, the *symmetric difference* $A \oplus B$ is defined as $(A - B) \cup (B - A)$, that is, the elements that are in exactly one of the two sets.

1. (4 pts.) Show that if M is a matching and P is an augmenting path with respect to M, then the symmetric difference $M \oplus P$ is a matching and $|M \oplus P| = |M| + 1$. Show that if $P_1, P_2, ..., P_k$ are vertex-disjoint augmenting paths with respect to M, then the symmetric difference $M \oplus (P_1 \cup P_2 \cup ... \cup P_k)$ is a matching with cardinality $|M| + k$.

   The general structure of our algorithm is the following:

```
Hopcroft-Karp(G)
1  $M \leftarrow \emptyset$
2  repeat
3      let $\mathcal{P} \leftarrow \{P_1, P_2, ..., P_k\}$ be a maximum set of
              vertex-disjoint shortest augmenting paths
              with respect to $M$
4      $M \leftarrow M \oplus (P_1 \cup P_2 \cup \ldots \cup P_k)$
5  until $\mathcal{P} = \emptyset$

6  return $M$
```

   The remainder of this problem asks you to analyze the number of iterations in the algorithm (that is, the number of iterations in the **repeat** loop) and to describe an implementation of line 3.

2. (4 pts.) Given two matchings M and M^* in G, show that every vertex in the graph $G' = (V, M \oplus M^*)$ has degree at most 2. Conclude that G' is a disjoint union of simple paths or cycles. Argue that edges in each such simple path or cycle belong alternatively to M or M^*. Prove that if $|M| \leq |M^*|$, then $M \oplus M^*$ contains at least $|M^*| - |M|$ vertex-disjoint augmenting paths with respect to M.

Let l be the length of a shortest augmenting path with respect to a matching M, and let P_1, P_2, ..., P_k be a maximum set of vertex-disjoint augmenting paths of length l with respect to M. Let $M' = M \oplus (P_1 \cup P_2 \cup ... \cup P_k)$, and suppose that P is a shortest augmenting path with respect to M'.

3. (2 pts.) Show that if P is vertex-disjoint from P_1, P_2, ..., P_k, then P has more than l edges.

4. (2 pts.) Now suppose P is not vertex-disjoint from P_1, P_2, ..., P_k. Let A be the set of edges $(M \oplus M') \oplus P$. Show that $A = (P_1 \cup P_2 \cup ... \cup P_k) \oplus P$ and that $|A| \geq (k + 1)l$. Conclude that P has more than l edges.

5. (2 pts.) Prove that if a shortest augmenting path for M has length l, the size of the maximum matching is at most $|M| + |V|/l$.

6. (2 pts.) Show that the number of **repeat** loop iterations in the algorithm is at most $2\sqrt{V}$. [Hint: By how much can M grow after iteration number $\sqrt{V}$?]

7. (4 pts.) Give an algorithm that runs in $O(E)$ time to find a maximum set of vertex-disjoint shortest augmenting paths P_1, P_2, ..., P_k for a given matching M. Conclude that the total running time of Hopcroft-Karp is $O(\sqrt{V}E)$.

34.2 Min Cost Flow

34.2.1 Streaming TV.

(20 pts.)

You are given a directed graph G, a source vertex s (i.e., a server in the internet), and a set T of vertices (i.e., consumers computers). We would like to broadcast as many TV programs from the server to the customers simultaneously. A single broadcast is a path from the server to one of the customers. The constraint is that no edge or vertex (except from the server) can have two streams going through them.

(A) (10 pts.) Provide a polynomial time algorithm that computes the largest number of paths that can be streamed from the server.

(B) (10 pts.) Let k be the number of paths computed in (A). Present an algorithm, that in polynomial time, computes a set of k such paths (one end point in the server, the other endpoint is in T) with minimum number of edges.

34.2.2 Transportation Problem.

(20 pts.)

Let G be a digraph with n vertices and m edges.

In the transportation problem, you are given a set X of x vertices in a graph G, for every vertex $v \in X$ there is a quantity $q_x > 0$ of material available at v. Similarly, there is a set of vertices Y, with associated capacities c_y with each vertex $y \in Y$. Furthermore, every edge of G has an associated distance with it.

The work involved in transporting α units of material on an edge e of length ℓ is $\alpha * \ell$. The problem is to move all the material available in X to the vertices of Y, without violating the capacity constraints of the vertices, while minimizing the overall work involved.

Provide a polynomial time algorithm for this problem. How fast is your algorithm?

34.2.3 Edge Connectivity

(20 PTS.)

The *edge connectivity* of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how the edge connectivity of an undirected graph $G = (V, E)$ can be determined by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.

34.2.4 Perfect Matching

(20 PTS.)

1. (10 PTS.) A *perfect matching* is a matching in which every vertex is matched. Let $G = (V, E)$ be an undirected bipartite graph with vertex partition $V = L \cup R$, where $|L| = |R|$. For any $X \subseteq V$, define the *neighborhood* of X as
$$N(X) = \left\{ y \in V \;\middle|\; (x, y) \in E \text{ for some } x \in X \right\},$$
that is, the set of vertices adjacent to some member of X. Prove *Hall's theorem*: there exists a perfect matching in G if and only if $|A| \le |N(A)|$ for every subset $A \subseteq L$.

2. (10 PTS.) We say that a bipartite graph $G = (V, E)$, where $V = L \cup R$, is *d-regular* if every vertex $v \in V$ has degree exactly d. Every d-regular bipartite graph has $|L| = |R|$. Prove that every d-regular bipartite graph has a matching of cardinality $|L|$ by arguing that a minimum cut of the corresponding flow network has capacity $|L|$.

34.2.5 Max flow by augmenting

1. (10 PTS.) Show that a maximum flow in a network $G = (V, E)$ can always be found by a sequence of at most $|E|$ augmenting paths. [Hint: Determine the paths after finding the maximum flow.]

2. (10 PTS.) Suppose that a flow network $G = (V, E)$ has symmetric edges, that is, $(u, v) \in E$ if and only $(v, u) \in E$. Show that the Edmonds-Karp algorithm terminates after at most $|V||E|/4$ iterations. [Hint: For any edge (u, v), consider how both $\delta(s, u)$ and $\delta(v, t)$ change between times at which (u, v) is critical.]

34.2.6 And now for something completely different.

(10 PTS.)

Prove that the following problems are NPC or provide a polynomial time algorithm to solve them:

1. Given a directly graph G, and two vertices $u, v \in V(G)$, find the maximum number of edge disjoint paths between u and v.

2. Given a directly graph G, and two vertices $u, v \in V(G)$, find the maximum number of *vertex* disjoint paths between u and v (the paths are disjoint in their vertices, except of course, for the vertices u and v).

34.2.7 Minimum Cut

(10 PTS.)

Present a *deterministic* algorithm, such that given an undirected graph G, it computes the minimum cut in G. How fast is your algorithm? How does your algorithm compares with the randomized algorithm shown in class?