Homework 7

Math 182, Winter 2025

When designing an algorithm, you must include the following:

1. A short (1-2 sentence) sketch of the main idea of the algorithm.
2. The algorithm itself, written in pseudo-code.
3. The runtime of the algorithm.
4. A proof that the algorithm is correct.
5. A proof that the runtime is correct.

If you wish to sort a list, assume that you can do so in $O(n \log n)$ time; you do not need to write out the details of the sorting algorithm. If you wish to use an algorithm we have discussed in class (eg BFS/DFS, Interval Scheduling, Dijkstra's Algorithm, Ford-Fulkerson, etc) you may do so without elaboration; if you need to modify the algorithm, you should provide sufficient details for the modification in your pseudocode. As mentioned in class, the runtime of the Ford-Fulkerson algorithm depend on implementation. You can use the Edmonds-Karp $O(|V||E|^2)$ bound without justification.

Question 1:

Suppose (G, w) is a network such that $w(e) = 1$ for all edges of G. Let k be a positive integer with $k \leq |E|$. Design an efficient algorithm to select k edges so that when these edges are removed the maximum flow in the new graph is as small as possible.

Question 2:

Recall our initial flawed attempt to write a greedy algorithm to solve the max flow problem:

1. Find an $s - t$ path.
2. Push as much flow through as possible.
3. Reduce the edge-weights along the path by the amount of flow we used, deleting an edge when the weight hits zero.
4. Repeat until there are no more $s - t$ paths.

We showed that this will not always produce an optimal graph. But sometimes greedy algorithms that aren't correct will nonetheless always produce a good approximation.

Prove or disprove the following claim: There is a fixed constant $A > 1$ (independent of the network and the method used to determine the path) so that the greedy algorithm will always produce a flow with value at least 1/A times the actual max-flow value.

Question 3:

Consider a set of computing clients in a town who need to be connected to one of several possible base stations. Clients can only be connected to base stations within a certain distance of them, and the base stations have a maximum capacity.

Formally, we have a list of n clients, with the location of the i-th client given by coordinates (x_i, y_i). Similarly, we have k base stations, with the coordinate of the j-th station given by (x'_j, y'_j). We are also given two parameters: a range parameter r and a load parameter l. Clients must connect only to a single base station, and can only connect to a station if the distance between the client and the base station is at most r. (This is a midwestern town, so there is no elevation changes to worry about; you

can compute the distance using the standard 2D distance formula.) Each base station can connect to at most l clients.

Design an efficient algorithm that takes as input the coordinates of each client and base station and the load and range parameters, and determines whether every client can be connected to a base station.

Question 4:

Suppose you work for the LA department of water and power managing the water mains for the city of LA. Each day, water is drawn from the Lake Mathews reservoir[1] in Riverside and delivered through a network of pipes and pumping stations to a storage tank in Westwood to supply water for UCLA's campus[2]. You are in charge of figuring out how to get as much water as possible from the reservoir to the storage tank each day.

Usually, the amount of water delivered each day is the same as the previous day. However, occasionally a new pipe is added or an old pipe is replaced with a larger pipe and more water can be sent than the previous day. You would like to come up with an algorithm to efficiently update the amount of water that can be delivered in such situations without having to redo the entire calculation from scratch.

You decide to formalize the problem as follows. You are given a network $G = (V, E, s, t, w)$ representing the pipes and pumping stations throughout the city (s represents the reservoir, t the storage tank in Westwood and the edge weights represent the pipe capacities). You are also given a maximum flow f on G (representing the amount of water sent through each pipe yesterday). You need to design an efficient algorithm to solve the following problem: given an edge $e \in E$, find a maximum flow on the network that is formed by increasing the weight of $w(e)$ by 1. For full credit, your solution should run in $\Theta(|V| + |E|)$ time. You may assume that all weights and flow values are integers.

Hint: Use the residual graph for yesterday's flow as a starting point.

Question 5:

Optional: Given an undirected graph $G = (V, E)$ and a set $A \subseteq V$, define the *set of neighbors* of A, written $N(A)$, to be the set of vertices which are the neighbor of at least one vertex in A. In other words,

$$N(A) = \{u \mid \text{for some } v \in A,\ (v, u) \in E\}.$$

Define a perfect matching of G to be a matching which touches every vertex—i.e. a matching $M \subseteq E$ such that for all $v \in V$, there is some $e \in M$ such that v is an endpoint of e.

Prove the following statement: if $G = (V, E)$ is a bipartite graph with bipartition (V_1, V_2) then G has a perfect matching if and only if $|V_1| = |V_2|$ and for every subset $A \subseteq V_1$, $|N(A)| \geq |A|$.

Hint: The forward direction is straightforward. For the other direction, use the max-flow/min-cut theorem.

[1] Fed by the Colorado River Aqueduct, which diverts water from the Colorado river 240 miles across the Mojave desert to supply water to Southern California.

[2] Actually I'm pretty sure the water at UCLA does not come from the Lake Mathews reservoir, but from one of the many other reservoirs in LA.