CIT 596 Online **Summer 2020**

Module 11–12

Name: Solutions

Questions requiring written answers.

1. Suppose you have a flow network G with integer capacities, and an integer maximum flow f . Suppose that, for some edge e, we increase the capacity of e by one. Describe an $O(m)$ time algorithm to find a maximum flow in the modified graph.

   Solution:

   Construct the residual graph using the given maximum flow f. That is, subtract f from the capacities of the respective edges in G and add reverse edges as appropriate. Now, increase the capacity of e by one and use BFS to check if there exists a path from the source to target. If so, then the new maximum flow is one higher than the old maximum flow. Otherwise, it is the same.

2. (Problem 9, Chapter 7, Page 419 of text) 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.

   Solution:

   Algorithm: We set up a network flow graph to solve the problem. First, create the source s and sink t nodes. Then create nodes p_i corresponding to the patients and insert the edges (s, p_i) for $(1 \leq i \leq n)$, each with capacity 1. Create nodes h_j corresponding to hospitals and add the edges (h_j, t) for $(1 \leq j \leq k)$, each with capacity $\lceil n/k \rceil$. Then, add all of the edges (p_i, h_k) that satisfy the condition that the ith person is within a half-hour's driving time of hospital k, each with capacity 1. Use the Ford-Fulkerson method to find the max flow, and return YES if the max flow has value equal to n. Return NO otherwise.

   Correctness: We argue that the task of sending all patients to hospitals in a balanced way is possible iff the max-flow has size n.
   ($\Longrightarrow$) We are given a way of sending all patients to hospitals in a balanced way. Then one could construct a flow in the following way: if person i is sent to hospital j, then add the path flow along the path $s \to p_i \to h_j \to t$. The total flow has value n, since the source contains one edge of capacity 1 for each of the n people. The "balanced" load ensures that the $\lceil n/k \rceil$ capacities are not violated.
   ($\Longleftarrow$) We are given a flow of value n. From this flow, we know to send person i to hospital j if the flow along the edge (p_i, h_j) is 1. The capacity constraints ensures that the load is balanced.

   Runtime: We create a graph with $n + k + 2$ nodes and up to $O(nk)$ edges, which can be done efficiently. Since the maximum possible flow is n, Ford-Fulkerson runs for at most n iterations, each of which involves a dfs that takes $O(nk)$ time. Therefore the entire algorithm takes $O(n^2k)$ time.

3. In a flow network with n nodes and m edges, given a flow f from s to t, prove that f can be decomposed into flows on at most m paths. A stronger claim would be that the flow can be decomposed into a flow on at most n paths. Is this claim true? Prove or give a counter-example.

   Solution:

   The proof of this claim is constructive and the algorithm for finding the decomposition is described below:

   - Find a path P from s to t where f has non-zero flow on each edge on the path.
   - Let $e*$ be the edge on P that has the least flow, $f(e*)$. Output path P with $f(e*)$ units of flow.
   - For all edges e inP replace $f(e)$ by $f(e) - f(e*)$. We now have a new flow f with a smaller flow value
   - Recursively find a path decomposition of this flow.

   Since each iteration zeroes out the flow on at least 1 edge, there can be at most m paths found by this procedure.

   However, a similar claim does not hold on the vertices. Imagine s is connected to k vertices and t is connected to a different k vertices. Each vertex connected to s has an edge to all vertices connected to t with capacity 1. All other edges have infinite capacity. There are k^2 augmenting paths, but there are only $2k + 2$ vertices.

4. A graph is called *d-regular* if all vertices in the graph have degree d. Prove that a d-regular bipartite graph (for $d \geq 1$) has a perfect matching. Furthermore, show that a d-regular bipartite graph is the disjoint union of d perfect matchings. **Hint:** The min-cut in an appropriate flow network can be useful in answering this question. Show Halls theorem

   Solution:

   Approach 1:
   Let $G = (V, E)$ be a d-regular bipartite graph and let $V = X \cup Y$ be the partition such that each edge has one endpoint in X and the other in Y. First of all, note that since each vertex in X has degree d, there must be $d|X|$ edges leaving X. These edges must enter Y and exactly d edges enter each vertex in Y. Thus $|Y| = \frac{d|X|}{d} = |X|$. Let us let n denote $|X|$.

   Construct the flow network G' we construct to solve the maximum matching problem with edges of capacity 1 from a source s to each vertex in X, edges in G being directed from X to Y and having capacity ∞, and capacity 1 edges from Y to a sink t.

   Suppose for contradiction that G does not have a perfect matching. Then the max flow in G' must have value less than n, and hence the minimum (s, t)-cut must have capacity less than n. Suppose this min-cut is a partition (A, B) where $s \in A$ and $t \in B$. A and B can each contain some vertices in X and some vertices in Y. Note that if there is an edge (u, v) such that $u \in A \cap X$ and $v \in B \cap Y$, then the capacity of this edge (which is ∞) would have contributed to the capacity of the cut, making that also ∞. Since this cut supposedly has capacity less than n, there can be no such edge. Thus the contributions to the capacity of the cut come from two sources: 1) Edges from $A \cap Y$ to t, each of capacity 1, for a total capacity of $|A \cap Y|$ and 2) Edges from s to $B \cap X$ of total capacity $|B \cap X|$.

   Thus the capacity of this cut is: $|A \cap Y| + |B \cap X| = n + |A \cap Y| - |A \cap X|$. This will be less than n only if $|A \cap Y| < |A \cap X|$. But every one of the $d|A \cap X|$ edges leaving $A \cap X$ must have an endpoint in $A \cap Y$, as we have argued previously, for the cut to have finite capacity. This means that $|A \cap Y| \geq |A \cap X|$, a contradiction.

   This proves that there are no cuts of capacity less than n, and therefore that G has a perfect matching.

   Approach 2:
   Another way to prove this is to simply show that this graph exhibits the Hall's Theorem property. If it does, then this graph has a perfect matching.

If consider any set $S \in X$, with x vertices, we know that it has $d * x$ out edges. Let $T \in Y$ consist of all neighbors for each vertex in S. Say that T has y neighbors. We know that each of these neighbors have a in degree of d, and some of them are due to vertices in S. Thus, we can conclude that $d * x \leq d * y$. This means $x \leq y$ so $|S| \leq |T|$. This concludes that the Hall's Theorem property exists, concluding the proof.

For the last part of the question, if we are given a d-regular bipartite graph, once we 'strip off' the perfect matching that is guaranteed to exist by the above argument, we are left with a $(d-1)$-regular bipartite graph, and we can repeat the argument all over again.

5. Show the following problem is in NP:

   Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are they *isomorphic*?

   Recall that two graphs are said to be isomorphic if there exists a bijection f from V_1 to V_2 such that for any two vertices $u, v \in G_1$, $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$.

   Solution:

   The certificate is the bijective mapping from vertices in V_1 to vertices in V_2. For each edge in (u, v) in E_1, check that the edge $(f(u), f(v))$ exists in E_2, and for each edge in E_2, check that the corresponding edge exists in E_1. If so, the graphs are isomorphic. Otherwise, they are not.

6. Consider the following two decision problems: In the SINGLE-SOURCE-DISTANT-VERTEX problem you are given a weighted, directed graph G, a source vertex s and a number L, and asked if there is a vertex u whose distance from s is at least L. In the ALL-PAIRS-DISTANT-VERTEX problem, you are just given a weighted, directed graph G and a number L, and asked if there are two vertices u and v such that the distance from u to v is at least L.

   Given an algorithm that solves SINGLE-SOURCE-DISTANT-VERTEX, show how you can solve ALL-PAIRS-DISTANT-VERTEX with at most n calls to the algorithm.

   How about the other direction? Given an algorithm for solving ALL-PAIRS-DISTANT-VERTEX can you see a way of using it to solve SINGLE-SOURCE-DISTANT-VERTEX efficiently? If you are not able to, that's okay. An explanation of your attempt and the difficulty you ran into will suffice.

   Solution:

   Given an algorithm that solves SINGLE-SOURCE-DISTANT-VERTEX, simply run the algorithm on each of the n vertices. If for any call it is found that there is a vertex whose distance from the source is at least L, then return YES. Otherwise, return NO.

   Now let's make an attempt at the other direction and see the problem that arises. Given an algorithm for the ALL-PAIRS-DISTANT-VERTEX problem, we want to know if we can solve SINGLE-SOURCE-DISTANT-VERTEX for a given graph G, vertex s, and number L. Let G' be the graph obtained from G by removing all incoming edges to s. First, using the algorithm and binary search, we find the largest L' for which the all-pairs-distant-vertex problem on (G', L') produces a YES-answer. In other words, there are some two vertices in G' whose distance is L' and all other distances are less than or equal to L'. Now we add L' to the weights of each of the outgoing edges from s to get a new graph G'' and query the all-pairs algorithm on $(G'', L + L')$. If the algorithm returns NO, then in G there was no vertex that was at distance at least L from s, because such a vertex would be at distance at least $L + L'$ in G''. If the algorithm returns YES, the only possibility is that there is a vertex whose distance from s in G is at least L, since all pairs not involving s will have distance at most L'.

   The one problem with this solution is that when we remove the incoming edges to s, we may cause some vertex v to become unreachable from some other vertex u, making their distance infinity. This can also be fixed by some additional nodes and edges, and it is possible to produce a correct but pretty complicated solution to this problem.