Homework #7
Introduction to Algorithms/Algorithms 1
601.433/633
Spring 2018

Due on: Thursday, April 26, 5.00pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: Gradescope.
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

May 11, 2018

Problem 1 (15 points)

Theorem 0.1 (Menger'27). *Let $k \in N$. In a directed graph G with vertices s and t, there are k edge-disjoint s-t paths (no two paths share an edge) if and only if t is still reachable from s after removing any $k-1$ edges.*

Please deduce Menger's Theorem from the Max-Flow Min Cut Theorem. (Hint: You may refer to Theorem 26.10 (Integrality Theorem) in the CLRS 3rd.)

Proof. We need to prove both directions of the theorem statement:
(1). If there are k edge-disjoint s-t paths, then t is still reachable from s after removing any possible $k-1$ edges.
(2). If t is still reachable from s after removing any possible $k-1$ edges, there are k edge-disjoint s-t paths.

Now let's prove the (1). Because each path is edge-disjoint, you can try to remove one edge on each path to maximum the possibility to disconnect s-t. Given that there are k edge-disjoint s-t paths, if we remove exactly one edge for each edge-disjoint path, $k-1$ paths are disconnected. But there is still one edge-disjoint path existing to keep s-t connected.

Now let's prove the (2). Assume the graph $G = (V, E)$ is a flow network with unit capacity. Use Ford-Fulkerson algorithm to construct a max flow on this flow network. By the Integrality Theorem, for all vertices $u, v \in V$, the value of $f(u, v)$ is an integer, which can be 0 or 1 in our case. Let's denote the the max flow as f and the min-cut(s, t) as $c(s, t)$. If we want to remove any $k-1$ edges from G, the most effective way is to remove edges of a min cut. Suppose we remove $k-1$ edges of the min-cut and s-t is still connected; there is at least one edge (of flow 1) remaining on the min-cut to make s-t connected. By Max-flow Min-cut Theorem, $|f| \geq k$. Since the edges of the flow network have unit capacity, for each edge-disjoint s-t path, there is at most flow 1 moving from s to t. So the value of max-flow $|f|$ is equal to the number of edge-disjoint s-t paths. Therefore, there are k edge-disjoint s-t paths.

□

Problem 2 (15 points)

The *support* of a flow is the set of arcs on which the flow function is positive. Show that there always exists a maximum flow whose support has no directed cycle.

Proof. First, given a flow network $G = (V, E)$, denote its maximum flow as f and assume that the support of f has a directed cycle. Repeat the following steps until there is at least one edge in the cycle whose flow is zero:

- On the edges of this cycle, find the edge with minimum value of flow and denote the this value as m.
- Reduce the flow of each edge in the cycle by m.

In this way, the new flow is still a max flow since the flow in the cycle will not go out of the cycle. Thus, the zero-flow edges does not belong to the support of max flow and then the cycle breaks.

□

Problem 3 (20 points)

You are given an undirected graph $G = (V, E)$ and vertices $s, t \in V$. Here we define that paths p_1 and p_2 are *totally different* if p_1 and p_2 have no common edges. Design an algorithm that counts the maximum number of *totally different* paths from s to t. Prove correctness and provide running time analysis.

Answer:

Algorithm:

- Convert undirected graph to a directed graph $G = (V, E)$ by replacing each edge with two opposite edges. Assign each $e \in E$ a capacity of 1.
- Run the Ford-Fulkerson/Edmonds-Karp algorithm and return the max-flow

Proof of Correctness:

Claim: Each iteration of the Edmonds-Karp algorithm that increases the flow does the following:

- increases the total flow by 1.
- selects 1 "totally different" path as the augmenting path.

Initialization. The first iteration of the algorithm will find any path from s to t. Because this is the first path we have seen, it is trivially "totally different" from the paths we have seen before. Moreover, because every edge in the path has a capacity of 1, the flow through this path is 1, and the total flow is increased by 1.

Maintenance. Assume the algorithm selects some new path P. Because P is a valid augmenting path, each edge in P must have a nonzero residual capacity. Because the total capacity of each edge in P is 1, and each edge in P has a nonzero residual capacity, this means that each edge in P must have a residual capacity of 1. Because each edge of P has a residual capacity of 1, this means that there is currently no flow going through any edge of P. This means that no edge of P was a part of any previously used augmenting path, and thus P is "totally different" from all paths we have seen before. Moreover, because of the same logic above, the flow through P is 1 and the total flow is increased by 1.

Termination. The algorithm terminates when it cannot find an augmenting path. If there is no augmenting path, that means there is no path from s to t such that each edge has a residual capacity of 1. This means that there is no path from s to t such that each edge has not been used by a prior augmenting path. This means that there is no path from s to t such that the path is "totally different" from all previous augmenting paths. Thus, the algorithm terminates when there are no remaining "totally different" paths.

Claim: If the Edmonds-Karp algorithm from step 2 returns some value k, then there are k "totally different" paths in the graph.

Proof: Assume for the sake of contradiction that there are actually $k' \neq k$ "totally different" paths in the graph.

- Case 1: $k' < k$. Our algorithm found k "totally different" paths, so $k' \geq k$. This is a contradiction.
- Case 2: $k' > k$. In this case, we could run the Edmonds-Karp algorithm again, and select the k' "totally different" paths as our augmenting paths. This would result in a total flow through the network of $k' > k$, which contradicts the fact that Edmonds-Karp gives us the max-flow.

Running Time: The runtime of our algorithm is equivalent to that of Edmonds-Karp, $O(|V||E|^2)$.