Midterm

Instructions

• This is a close-book/note exam. The exam length will be one hour and 20 minutes.

• There are four questions. Please complete any three.

• Collaboration is not permitted in this exam.

• Use of electronic devices is not permitted in this exam.

• No question will be answered by the teaching staff unless it is typo-related.

• Sign the honor code statement on the front of this cover sheet.

• GOOD LUCK!

Honor Code

In recognition of and in the spirit of the Honor Code, I certify that I will neither give nor receive unpermitted aid on this examination and that I will report, to the best of my ability, all Honor Code violations observed by me.

Name: ________________________________________________

Signature: ____________________________________________
1. Maximal Matching

A matching $M$ of a graph $G$ is called maximal if no more edges can be added to it.

a) Give an example of a graph $G$ and a maximal matching such that the maximal matching is strictly smaller than the maximum matching of $G$.

Solution: Consider the following example. Let $G = (V, E)$ be a path on four nodes. So $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$. Then $M = \{(v_2, v_3)\}$ is a maximal matching because adding either of the other two edges results in the set no longer being a matching. However, the maximum matching of $G$ is $M^* = \{(v_1, v_2), (v_3, v_4)\}$, strictly larger than $|M|$.

b) Prove that the size of a maximal matching is at least $\frac{1}{2}$ of the size of maximum matching.

Solution: Let $M^*$ be the maximum matching of a graph $G = (V, E)$. Suppose $|M^*| = k$ and let $M$ be any maximal matching of $G$. Now, for a contradiction, suppose $|M| < k/2$. The number of vertices covered by $M^*$ is $2k$ and $M$ covers $< k$ vertices. That is, $M$ does not cover at least one end point of all $k$ edges in $M^*$. So there exists an edge $e = (u, v) \in M^*$ such that neither $u$ nor $v$ are covered by edges in $M$. But that means $M \cup \{e\}$ is still a matching, contradicting the fact that its maximal. So we have shown that for all maximal matchings $M$, $|M| \geq |M^*|/2$.

2. Escape Problem

An $n \times n$ grid is an undirected graph consisting of $n$ rows and $n$ columns of vertices, as shown below. We denote the vertex in the $i$th row and the $j$th column by $(i, j)$. All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the points $(i, j)$ for which $i = 1, i = n, j = 1,$ or $j = n$.

Given $m \leq n^2$ starting points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ in the grid, the escape problem is to determine whether or not there are $m$ edge-disjoint paths from the starting points to any $m$ different points on the boundary. For example, the grid in Figure 1 has an escape.

Use maximum flow to give an efficient algorithm to solve the escape problem.

Solution: To use max-flow algorithm, we need to add a source node $s$ and a destination node $t$ to the grid. We connect $s$ to each of the $m$ starting points and draw a link from every boundary point to $t$. We replace grid edges with two directed edges and give a capacity of one to all edges in the graph. Because all capacities are integral, the max flow will be integral, so we may find feasible edge-disjoint paths. If the size of maximum-flow in the resulting graph is $m$ then there exists $m$ edge-disjoint paths from each of the $m$ starting points to a boundary point.
Note that to ensure correctness, we need to consider the case when both directed edges between a pair of grid nodes have flow in the output max flow. In this case, the output flow is not edge disjoint. However, we may remove the flow on both these edges without changing any feasibility and without affecting the size of the flow. Therefore, these cases do not affect the size of the maximum flow. To actually find the paths, we first need to remove all these bi-directional flows between grid nodes.

To find the $m$ edge-disjoint paths, we start from $s$ and follow the edges with positive flow to reach $t$, call this path $P$. $P$ is clearly an escape path from one of the starting points to a boundary point. We remove edges of $P$ from the flow network, it is easy to observe that the remaining flow is still feasible. We can do this $m - 1$ more times to retrieve all the $m$ escape paths.

3. Ride Assignment

After designing a new logo and brand, a giant ride hail company realizes that they have no idea how to actually assign drivers to potential passengers. The CEO hears you are taking MS&E 212 and may be able to help them with this assignment problem, so the company contacts you for advice.

The CEO describes the problem as follows: consider the city map as a network with roads as edges and intersections as nodes. At any given time we have a set of passengers requesting rides from our service and a set of drivers who may pick up these passengers. The company currently just randomly assigns a car in the area to each person. However, they have been receiving some complaints about wait-time. Therefore, the CEO wishes to find a new method of assignment to minimize wait-times for all the passengers. He tasks you to develop this method with the following
stipulations and assumptions:

- Suppose at a certain time in the city, there are \( n \) people requesting a pick up and \( m \) drivers in the area. We assume \( m \geq n \)
- All people and drivers are located at intersections, the nodes of the map network
- You are given the travel time for every road in the network
- Your goal is to find an assignment of passengers to drivers minimizing the total waiting time

Provide a suitable algorithm to solve this problem.

**Hint:** Consider using the original map network to create a new network with people and cars as nodes to help find the optimal assignment.

**Solution:** There were a couple ways of going about this problem, both involving minimum cost flows.

One way dealt with first finding the shortest distance between all car and people pairs. One could do this via minimum cost flow, as described in class; start with the original road map network and originally give all nodes a balance of zero, then per a particular pair of car and person, increment the balance of the node where the car is by one and decrease the balance of the node where the person is by one. Then with the travel times as edge (road) costs and arbitrarily large integral capacities on all edges (could work with unconstrained or with even capacity of 1 as well), solve the min-cost flow problem to get the shortest distance. Then repeat this process for all pairs. Note that some people suggested Dijkstra’s to get all pairs shortest path, which would also work.

Once all shortest paths are calculated, create a new bipartite network. One side is all cars and one side is all people. The network is complete - edge \((c_i, p_j)\) \((1 \leq i \leq m, 1 \leq j \leq n)\) exists from each car to each person with cost equal to the shortest path between them. Now we just need to find the minimum cost maximum matching on the graph to give us our assignment. We can do this with min-cost flow as well, we can create a source node \(s\) with balance \(n\) and edges \((s, c_i)\) for all \(1 \leq i \leq m\) with cost 0 and capacity 1. Also, we create a sink node \(t\) with balance \(-n\). Create edges \((p_j, t)\) for \(1 \leq j \leq n\) with cost 0 and capacity 1. Give all other edges capacity 1 and all other nodes balance 0. Now, solving this min-cost flow problem outputs an integral solution because all capacities and balances are integral and thus a valid optimal assignment.

Note also, that a min-cost flow could have been used directly on the original map network. We give every node a balance equal to: (number of cars at node) - (number of people at node). All edges have cost equal to travel time and capacity set to an arbitrarily high integer value (or leave this portion unconstrained). We also need to create a sink node to take care of any extra cars we may have so that balance can be satisfied and we can get feasible solutions. Create node \(t\) and create an
edge from every node in the network to $t$ with cost 0 and capacity 1 (or any larger integer). Give a balance of $-(m - n)$ to $t$. Now solve for the min-cost flow on this network. Again, solving this min-cost flow problem outputs an integral solution because all capacities and balances are integral. But now we need to do a little more work to analyze the flow and grab the optimal assignment (not described here).

4. Unique Minimum $s - t$ Cut

Given a network $G(V, E, s, t)$, describe a algorithm to determine whether $G$ has a unique minimum $s - t$ cut.

**Solution:** First compute a minimum $s$-$t$ cut $C$, and define its volume by $|C|$. Let $e_1, e_2, \ldots, e_k$ be the edges in $C$. For each $e_i$, try increasing the capacity of $e_i$ by 1 and compute a minimum cut in the new graph. Let the new minimum cut be $C_i$, and denote its volume (in the new graph) as $|C_i|$. If $|C| = |C_i|$ for some $i$, then clearly $C_i$ is also a minimum cut in the original graph and $C \neq C_i$, so the minimum cut is not unique. Conversely, if there is a different minimum cut $C'$ in the original graph, there will be some $e_i \in C$ that is not in $C'$, so increasing the capacity of that edge will not change the volume of $C'$, thus $|C| = |C_i|$. In conclusion, the graph has a unique minimum cut if and only if $|C| < |C_i|$ for all $i$. The algorithm takes at most $m + 1$ computing of minimum cuts, and therefore runs in polynomial time.