Problem 1 (15 points).

You are given a network $G = (V,E)$ with positive integral capacity $c(e)$ on $e \in E$, a source $s \in V$, and a sink $t \in V$. Let f be an integral maximum s-t flow and G, and let G_f be the residual graph w.r.t. f. Define $S_1 = \{v \in V \mid s \text{ can reach } v \text{ in } G_f\}$ and $T_1 = V \setminus S_1$; $T_2 = \{v \in V \mid v \text{ can reach } t \text{ in } G_f\}$ and $S_2 = V \setminus T_2$.

1. Prove that (S_2, T_2) is one minimum s-t cut of G.

   Solution: The proof is symmetric to the proof for the correctness of Ford-Fulkerson's algorithm.

   We first prove that (S_2, T_2) is an s-t cut, i.e., $s \in S_2$ and $t \in T_2$. Suppose coversely that $s \notin S_2$. Then we have $s \in T_2$ and therefore there exists an s-t path p in the residual graph $G(f)$. This is contradicting to the fact that f is a maximum flow.

   Now we prove that (S_2, T_2) is an minimum s-t cut. Consider these edges from S_2 to T_2 and those edges from T_2 to S_2. Formally, define $E(S_2,T_2) = \{(u,v) \in E \mid u \in S_2, v \in T_2\}$, and $E(T_2,S_2) = \{(u,v) \in E \mid u \in T_2, v \in S_2\}$. For any edge $e = (u,v) \in E(S_2,T_2)$, we must have that e is saturated in f, i.e., $f = c(e)$. This is because otherwise (i.e., $f(e) < c(e)$), there will be an edge from u to v in the residual graph G_f, and therefore u can reach t, which contradicts to the definition of $E(S_2,T_2)$. Also, for any edge $e = (u,v) \in E(T,S)$, we must have that $f^*(e) = 0$. This is because otherwise (i.e., $f(e) > 0$), there will be an edge from v to u in G_f, and therefore v can reach t, which contradicts to the definition of $E(T_2,S_2)$. Then we compute the value of flow f: we proved that $|f| = \sum_{e \in E(S_2,T_2)} f(e) - \sum_{e \in E(T_2,S_2)} f(e)$. By using the above results, we have $|f| = \sum_{e \in E(S_2,T_2)} c(e) - 0 = \sum_{e \in E(S_2,T_2)} c(e)$, which is exactly the capacity of cut (S_2,T_2). As we know $c(S_2,T_2) \geq |f|$ for any cut (S_2,T_2) and any flow f, such an equation of $|f| = c(S_2,T_2)$ implies that (S_2,T_2) is a minimum s-t cut.

2. Let $\mathcal{C}$ be the set of *all* minimum s-t cut of G. Design a polynomial-time algorithm to compute the intersection of s-side (i.e., S) of all minimum s-t cut (S,T) in $\mathcal{C}$, formally $\cap_{C=(S,T)\in\mathcal{C}} S$; design a polynomial-time algorithm to compute the union of s-side (i.e., S) of all minimum s-t cut (S,T) in $\mathcal{C}$, formally $\cup_{C=(S,T)\in\mathcal{C}} S$. Show the running time of your algorithm and prove that your algorithm is correct. *Hint:* use S_i and/or T_i, $i = 1,2$, in your algorithm.

   Solution. The algorithm to compute $\cap_{C=(S,T)\in\mathcal{C}} S$ and $\cup_{C\in\mathcal{C}} S$ is quite simple: $\cap_{C=(S,T)\in\mathcal{C}} S = S_1$ and $\cup_{C=(S,T)\in\mathcal{C}} S = S_2$. The running time of this algorithm is dominated by the running time of the max-flow algorithm.

   We first prove that $\cap_{C=(S,T)\in\mathcal{C}} S = S_1$, and the other part can be proved symmetrically. First, we show that if $v \in \cap_{C=(S,T)\in\mathcal{C}} S$ then we have $v \in S_1$. In fact, $(S_1, V - S_1)$ is a minimum s-t cut (proved in class). Thus, we have that $v \in S_1$, as v belongs to the s-side of *every* minimum s-t cut. Second, we prove that if $v \in S_1$ then we have $v \in \cap_{C=(S,T)\in\mathcal{C}} S$. We prove this by contradiction. Assume that $v \notin \cap_{C=(S,T)\in\mathcal{C}} S$. Then there exists one minimum s-t cut $C' = (S',T')$ such that $v \in T'$. Since $v \in S_1$, there must be one path in G_f from s to v. This path must go through some cut-edge e of C' since v is in the t-side of C'. This means that e is not saturated by f, which is a contradiction with the fact that C' is one minimum s-t cut.

3. Design a polynomial-time algorithm to decide if a given network has a unique minimum s-t cut. Show the running time of your algorithm and prove that your algorithm is correct. *Hint:* use S_i and/or T_i, $i = 1,2$, in your algorithm.

   Solution. According to above result, a given network has a unique minimum s-t cut if and only if $S_1 = S_2$, or equivalently $T_1 = T_2$.

Problem 2 (10 points).

You are given a directed graph $G = (V,E)$ with unit integral capacity $c(e) = 1$ on $e \in E$, a source $s \in V$,

and a sink $t \in V$. You are also given an integral maximum *s-t* flow f of G. Now we would like to pick k edges from E and remove them from G such that the maximum flow of the new graph is reduced as much as possible. In other words, find a set of edges F ($F \subset E$, $|F| = k$) such that the max-flow of $G' = (V, E - F)$ is as small as possible. Design one polynomial-time algorithm to identify such k edges. You need to show run time complexity analysis and correctness of your algorithm. A correct but high-complexity algorithm will earn half of the total points of this problem.

Solution: We can find a min-cut (e.g. like in problem 1) and remove k edges from its $|f|$ cut edges. We can reduce the max-flow by k to $|f| - k$, if $k < |f|$, and 0 otherwise.

Complexity: Given a max-flow f we can find a min-cut and associated min-cut edges using the algorithm introduced in problem 1, which runs in $O(|E|)$. Removing k edges takes $O(|E|)$ time. Hence the total complexity is $O(|E|)$.

Correctness: The value of the original max-flow is $|f|$. Since all edges have unit capacity, apparently any *s-t* min-cut of G has $|f|$ cut edges. After removing k edges from this min-cut, the resulting new cut has $(|f| - k)$ cut edges. We show that this new cut is still a min-cut of the new graph G'. This is because *all* min-cut in G have $|f|$ cut edges. Any min-cut of G' can be augmented as a min-cut of G by adding the removed k edges. This leads to that the min-cut of G' contains exactly $(|f| - k)$ edges.

Problem 3 (10 points).

You are given a network $G = (V, E)$ with positive integral weight $w(e)$ on $e \in E$, a source $s \in V$, and a sink $t \in V$. Now you need to find k different paths from s to t, and one edge can only be on one of the k paths. The bottleneck of a path is defined as the largest weight of the edges on this path. Let L be the maximum bottleneck among the k paths from s to t. Design an algorithm to find k paths so as to minimize L. *Hint:* Read 7.6 [KT] and you may directly use any algorithm introduced there.

Solution: The problem of deciding if an (unweighted) graph contain k edge-disjoint paths from s to t can be solved efficiently by reducing to the max-flow problem (see 7.6 [KT]). We will use it as a subroutine for solving this problem.

Now, we want to find k different paths from s to t that minimizes L. We first sort all edges according to their weights. We then use *binary search* over all edge-weights to get the minimum L. For any given edge-index i, we only keep the edges that the weight of it is smaller or equal to $w(e_i)$ and remove all other edges, and decide if k paths exist using above subroutine. If it does, we can keep decrease i as $i/2$ in binary search, otherwise we need to increase i to $2i$ in binary search. The running time is $O(\log(|E|)) \cdot T(maxflow)$, if we use preflow-push algorithm, the overall running time would be $O(\log(|E|) \cdot |V|^2 \cdot |E|)$.

Problem 4 (10 points).

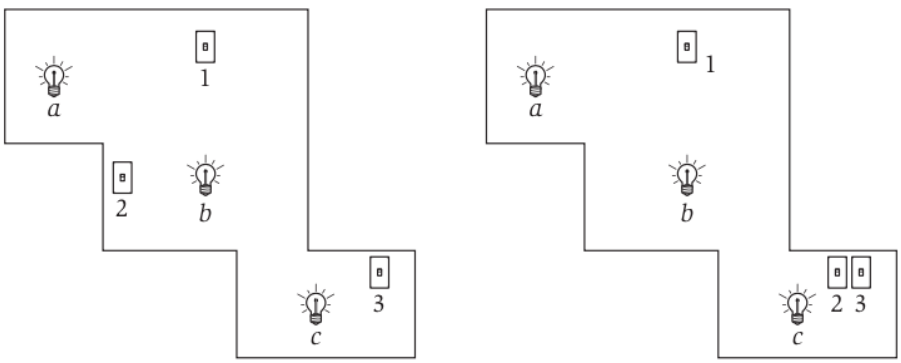
Given n booked taxi rides, each of the rides contains the source address, the destination address and the departure time. We define an address as the street and avenue number (x, y). And it takes $|x_1 - x_2| + |y_1 - y_2|$ time to drive from (x_1, y_1) to (x_2, y_2). A cab may carry out a booked ride if it is its first ride of the day, or if it can get to the source address of the new ride from its latest, at least one minute before the new ride's scheduled departure. Design a polynomial-time algorithm, to find the minimum number of cabs required to carry out all the booked taxi rides.

Solution: We can reduce this problem into a bipartite max-matching problem. Initially, we can assume we need n taxis to finish all the rides. Once we find one taxi can reach one departure address in time after finish current ride, the number of taxis needed can be decreased by one. According to this idea, we can build a

bipartite graph $G = (X \cup Y, E)$, where each vertex $x_i \in X$ corresponds to a destination, each vertex $y_j \in Y$ corresponds to a source address, and each edge $e = (x_i, y_j) \in E$ if one taxi can take ride j after ride i, which can be determined in constant time by calculating their distances. The max-matching of G corresponds to the minimum number of cabs needed, formally, let m be the size of the max-matching of G, then the minimum number of cabs needed is $n - m$.

Problem 5 (10 points).

Let's call a floor plan, together with n light fixture locations and n switch locations, ergonomic if it's possible to wire one switch to one fixture (i.e., to pair them) so that every fixture is visible from the switch that controls it. A floor plan will be represented by a set of m horizontal or vertical line segments in the plane (the walls), where the i-th wall has endpoints $(x_i, y_i), (x'_i, y'_i)$. Each of the n switches and each of the n fixtures is given by its coordinates in the plane. A fixture is visible from a switch if the line segment joining them does not cross any of the walls. Design an algorithm to decide if a given floor plan is ergonomic. The running time should be polynomial in m and n. You may assume that you have a subroutine with O(1) running time that takes two line segments as input and decides whether or not they cross in the plane, i.e. whether a fixture is visible from a switch (visibility not blocked by a wall).



(a) Ergonomic **(b) Not ergonomic**

Figure 1: The floor plan in (a) is ergonomic, because we can wire switches to fixtures in such a way that each fixture is visible from the switch that controls it. (This can be done by wiring switch 1 to a, switch 2 to b, and switch 3 to c.) The floor plan in (b) is not ergonomic, because no such wiring is possible.

Solution: We can convert this problem to a bipartite matching problem. Make a graph $G = (X \cup Y, E)$. Each vertex $x_i \in X$ corresponds to a fixture. Each vertex $y_j \in Y$ corresponds to a switch. Draw an edge $e = (x_i, y_j)$ in E if line segment (x_i, y_j) is not intersecting with any of the m line segments, i.e. fixture x_i is not blocked by any wall from switch y_j. If we can find a max-matching of G and the size of this matching is n, this floor plan is ergonomic, otherwise no.

Complexity: Making X and Y takes $O(n)$ time. Deciding whether (x_i, y_j) is intersecting with any of the m line segments takes $O(m)$ time. We have n^2 possible combinations between n fixtures and n switches, so making E takes $O(n^2 m)$ time in total. Finding max-matching can be done in polynomial time.

Correctness: In the max-matching, each picked edge (x_i, y_j) represent the writing between x_j and y_j. If we have a max-matching of size n, we can wire those n fixture and n switches according to matching so that the floor plan is ergonomic. On the other hand, if the floor plan is ergonomic, we can have a one-to-one wiring between the n fixtures and n switches. It means we can pick n edges from the bipartite graph according to the wiring plan. Apparently those n edges is a matching of G because the wiring plan is one-to-one. Since

size of X and Y are both n, maximum possible size of max-matching of G is n. In conclusion, there exists a max-matching of G with size n if and only if the floor plan is ergonomic.

Problem 6 (10 points).

There are m mice and n holes on the ground of a warehouse; each is represented with a distinct 2D (x,y) coordinates. A cat is coming and all mice are running to try to hide in a hole. Each hole can hide at most k mice. All mice run at the same velocity v. A mouse is eventually safe if it reaches a hole (with at most k mice including itself) in s seconds. Design a polynomial-time algorithm (in m and n) to find the maximum number of mice that can be safe.

Solution. We build a network $G = (V,E)$, where $V = \{M_1, M_2, \cdots, M_m\} \cup \{H_1, H_2, \cdots, H_n\} \cup \{s,t\}$. $\{M_i\}$ correspond to the m mice and $\{H_j\}$ correspond to the n holes. In the network we add edge (M_i, H_j) if mouse-i can reach hole-j in s seconds, i.e., if the distance between mouse-i and hole-j is within $s \cdot v$; the capacities of such edges are always 1. We add edge (s, M_m) with capacity 1, and add edge (H_j, t) with capacity of k to guarantee that each hole can hide up to k mice. We then calculate an (integral) flow f of this network; $|f|$ gives the maximum number of mice that can be safe.

The running time of above algorithm consists of building the network which is $O(mn)$ and calculating the max-flow which is $O(|v|^2|E|) = O((m+n)^2 mn)$. The total running time is $O((m+n)^2 mn)$.

The correctness of above is straightforward. There is a one-to-one correspondence between "hiding" plan and integral flow of the network. Therefore, a plan in which maximized number of mice can be safe correspond to the flow with maximized value.

Problem 7 (10 points).

A company has n branches in one city, and it plans to move some of them to another city. The expenses for operating the i-th branch is a_i per year if it stays in the original city, and is b_i per year if it is moved to the new city. The company also needs to pay c_{ij} per year for traveling if the i-th and j-th branches are not in the same city. Design a polynomial time algorithm to decide which branches should be moved to the new city such that the total expenses (including operating and traveling expenses) per year is minimized.

Solution. We build a directed network $G = (V,E)$, where $V = \{X_1, X_2, \cdots, X_n\} \cup \{s,t\}$, where $\{X_i\}$ correspond to the n branches. We add the following edges to the network. For every branch X_i, we add edge (s, X_i) with capacity of b_i. For every branch X_i, we add directed edge (X_i, t) with capacity of a_i. For every pair of branches (X_i, X_j), we add two edges (X_i, X_j) and (X_j, X_i) with identical capacity of c_{ij}.

We run any max-flow algorithm to get the maximum-flow and minimum s-t cut of G. Let (S^*, T^*) be the minimum s-t cut of G. We return $T^* \setminus \{t\}$, i.e., move the corresponding branches in $T^* \setminus \{t\}$ to another city.

We now prove that the above algorithm is optimal. Let (S,T) be an arbitrary s-t cut of G. Its capacity is $c(S,T) = \sum_{X_i \in S} a_i + \sum_{X_j \in T} b_j + \sum_{X_i \in S} \sum_{X_j \in T} c_{ij}$. This is exactly the total cost of moving branches in $T \setminus \{t\}$ to another city. Hence, we have that the capacity of minimum s-t cut gives the minimized total costs. This prove that $T^* \setminus \{t\}$ is the optimal solution.

Problem 8 (5 points).

You are given an integer k, a set X and a collection $\mathcal{C}$ of subsets of X (e.g., each element of $\mathcal{C}$ is a subset of X). Design a polynomial time (in terms of $|X|$ and $|\mathcal{C}|$) algorithm to decide whether there exists a subset $\mathcal{S} \subset \mathcal{C}$ such that $|\mathcal{S}| \geq k \cdot |\cup_{A \in \mathcal{S}} A|$.

Solution. This problem can be reduced to the project selection problem. We setup a directed graph by adding one vertex for each element in X, one vertex for each subset in $\mathcal{C}$, and adding one edge from $x \in X$ to $A \in \mathcal{C}$ if $x \in A$. Notice that the existence of a sub-collection $\mathcal{S}$ satisfying $|\mathcal{S}| \geq k \cdot |\cup_{A \in \mathcal{S}} A|$ is equivalent to $\max_{\mathcal{S}}(|\mathcal{S}| - k \cdot |\cup_{A \in \mathcal{S}} A|) \geq 0$. Thus, we need to compute $\max_{\mathcal{S}}(|\mathcal{S}| - k \cdot |\cup_{A \in \mathcal{S}} A|)$. For any $A \in \mathcal{C}$ we set its profit $p(A) = 1$, and for $x \in X$ we set its profit $p(x) = -k$. Clearly, computing $\max_{\mathcal{S}}(|\mathcal{S}| - k \cdot |\cup_{A \in \mathcal{S}} A|)$ is equivalent to finding a vertex subset of the directed graph such that all parents of each vertex in the subset are also in this subset and that the sum of the profit in the subset is maximized.