Ali Mirabzadeh
305179067
CS 180 – HW7
11.
Consider a network that consists of N * N + 2 vertices - s and t are source and sink vertices and N * N vertices that are connected only with the direct vertical and horizontal neighbors. Let all edges have capacities 1 and have only left-to-right and down-to-up directions. The maximum flow value in this network is N (simply use only left-to-right edges when looking for an *s-to-t.* path). Now consider a path that starts from s and goes right and eventually up to get to t, found by the forward-only algorithm. If this path is considered first by the algorithm, it blocks all other paths that are considered by the algorithm, as s and t are then disconnected in the "forward-only" residual graph and the returned maximum flow is 1. Since N is arbitrary, the claim that the forward-only algorithm always finds a constant approximation solution is wrong.

Ali Mirabzadeh
305179067
CS 180 – HW7
14.

To solve this, I create a network flow problem. Assign unit capacity to all existing edges. Introduce source node s and connect it to each node in X with a unit capacity directed edge. Introduce sink node t and connect each node in S with a directed edge to t with capacity |X|. Then calculate max-flow. We argue that max-flow=|X| if and only if such routes exist.
First: If required routes exist, max-flow=|X|. Use |X| unit capacity edges from source to each of the nodes in X. From there, we have a path from each node in X to some node in S using unique edges. Thus the |X| units of flow can reach nodes in S from where they have |X| capacity paths to the sink.
Second: If max-flow=|X|, required routes exist. If max-flow=|X|, each node in X is receiving unit capacity flow from s, and since all this flow is reaching t and each edge between X and S is unit capacity, all of this flow must be using unique edges. Thus we have a unique path from every node in X to some node in S. Note that you may have edges between X and they might even be used. It is also not required to have |X| = |S|. Consider a small example of a graph with nodes X1, X2, X3, S1, S2 and edges (X1, X2),(X2, S1),(X2, S2),(X3, S2). The required routes exist in this graph. Also, there can be any number of interconnecting nodes between X and S.

Ali Mirabzadeh
305179067
CS 180 – HW7
17.
An algorithm that accomplishes this task using only 0(Klogn) pings. Here is restricted to O(K log n) pings. The run time should not be restricted, instead it should be polynomial time. The maximum flow R* -- value of arc capacity is 1. We have K-disjoint paths. The K - disjoint paths are: p1,p2,…pn Using foxd - fulkession algorith, these paths can be formed is 0(mk). The intruder has L arc. One are destroyed by each path. We cannot space two - paths for a single arc. The path has O(n) length each. Bisection - search ping algorithm is used to remove pi. If the vertex is formed ping it otherwise vertex-3-quetus along pi. Therefore, O(logn) remove edges, last vertex on the path is pinged and the first vertex which cannot be pinged is the removed edge. e* is the removed edge. then perform BFS search, the k-disjoint paths are return vertices that don’t exist

Ali Mirabzadeh
305179067
CS 180 – HW7
29.
I use Min-cut to solve this problem.
Let's create an undirected graph as graph has vertex for every SW aplications, I call them S1,S2,…,Sn. There is a special vertex t. If application i and j have an associated value xij, then we have an edge between Si and Sj of capacity xij. For every vertex Si, I not equal to, we have an edge (Si,t) of capacity b_{Si} . We can say that S1-t min-cut would give the desired answer.
Let X be such a cut so that S1 is in X. and t is not. Capacity of X is Total of xij – Total of b_{Si} + total b_S .
This is same as expense – benefit of moving application which are not in set X
Hence, finding a min-cut is the same as finding the set of applications for which expense – benefit is minimized.