**CS 180** Homework 7

**Problem 1**

1. In a directed graph with negative edge weights but no negative cycles, there must be a node whose shortest path from the source consists of only one edge. If we assume that this node exists and try to prove the opposite, that the shortest path from the source to this node can be even shorter, that would entail that a negative cycle exists because you can just keep running through the cycle to infinitely shorten the length.
2. The Bellman-Ford algorithm only scans |V|-1 edges because the longest possible path without a cycle that can be formed would consist of |V|-1 edges. Therefore, on the |V|th iteration of the Bellman-Ford algorithm, it checks for any changes to the tables which would therefore indicate a cycle because the algorithm found a shorter path with |V| edges.

**Problem 2**

1. use the Gomory Hu algorithm to enumerate all minimum cuts and find the minimum cut
2. recursively enumerate all directed cuts in residual graph after contracting the strongly connected components of a graph. This would run in linear time per cut for an overall polynomial time. Then find the minimum cut.

**Problem 3**

In a graph G, convert any vertex v into two vertices, v1 and v2. We then put a directed edge from v1 to v2 with a capacity of weight(v). For every incoming node of vertex v, we point the node to v1 and then we point outgoing nodes of v to v2. We make all the original edges have infinite capacity. G’ will be the new graph and in it, a maximum flow s-t flow specifies a maximum s-t flow in G. And because all the original edges of G have infinite capacity in G’ then the minimum s-t cut in G’ will only have new edges. Therefore, the minimum cut of G’ can be considered as the set of nodes in G with a total capacity equal to the minimum edge capacity in G’. Then by the Max-Flow-Min-Cut theorem which is for edge capacities, the maximum s-t flow equals the minimum total weight of a set of nodes whose removal disconnects al s-t paths in G.

**Problem 4**

For a collection of *n* software applications, {1, … ,n} we can construct an undirected graph with *n* vertices labeled v1, … ,vn. If two applications *i* and *j* have an interaction expense *xij* then we create an edge between vi and vj of capacity xij. Then for every vertex vi where i is not 1 (because we can’t move the first application) we create an edge from vi to a dummy node d at the end and the edge capacity will be the benefit of porting the application bi. We can then treat v1 as the source node by giving it an edge to every other node except the dummy node and the dummy node d as the sink node. By finding a v1-d min-cut, we will have our solution. This min-cut will include v1 and not d and the capacity will be the expense subtracted with the benefit of porting applications not in the min-cut set. This is equivalent to finding the right applications to minimize the expression, expense - benefit, or better put, to maximize benefit - expense.