AoA HW6

Name: Ayush Jain
UNI: aj2672

Problem 1

Exercise 25.2-6
If any of the diagonal element in the output of the Floyd-Warshall algorithm is zero, it means that the graph contains a negative-weight cycle.

In the output matrix of the algorithm, each element d_{ij} is the weight of the least weight path from the vertex i to j. In the absence of any negative-weight cycles, d_{ij} should be equal to 0 for i=j. However, when a negative-weight cycle exists, zero will be replaced by the weight of this cycle and hence tells us the existence of a negative-weight cycle.

Exercise 25.1-10

```
MINIMUM-NEGATIVE-WEIGHT-CYCLE(W)
  1. n = W.rows
  2. $L^1$ = W
  3. for m = 2 to n - 1
  4.     let $L^m$ be a new $n \times n$ matrix
  5.     $L^m$ = EXTEND-SHORTEST-PATHS($L^{m-1}$, W)
  6.     for i=0 to n-1
  7.         if $L^m[i][i] < 0$
  8.             return m
  9. return 0
```

```
EXTEND-SHORTEST-PATHS(L, W)
  1. n = L.rows
  2. let $L' = (l'_{ij})$ be a new $n \times n$ matrix
  3. for i = 1 to n
  4.     for j = 1 to n
  5.         $l'_{ij} = \infty$
  6.         for k = 1 to n
  7.             $l'_{ij} = min(l'_{ij}, l'_{ik} + w_{kj})$
  8. return L.
```

The above algorithm gives the minimum negative weight cycle for a graph. This is similar to the all pairs shortest paths given in the book. We consecutively find L^m from L^{m-1} at each step with the weight function. The little addition is that at every step we check the diagonal elements of the L^m matrix to see if any of the diagonal element is negative. Since, a cycle essentially means that the path starts and ends at the same vertex, we need to check the distance of the vertex from itself.

Ideally it should be zero, unless there is a negative-weight cycle in the matrix, in which case it will become negative. Since, we need to only find the length of the minimum-length negative-weight cycle, we stop at the point and return m at that point.

The proof of the algorithm comes from the all-pair-shortest-paths. After every execution, each element l_{ij} in the L^m matrix stores the minimum path length formed with m or less edges between vertex i and j. For all the diagonal elements, this means the distance between the element i and itself. This will be zero initially. However, if any negative cycle exists than this will hold the value of the cycle. And the value of m for which l_{ij} becomes negative is the number of edges in the cycle.

The time complexity is same as the shortest path problem being $O(n^4)$. The difference is the loops of line 6-8 in the MINIMUM-NEGATIVE-WEIGHT-CYCLE algorithm, which is O(n) as compared to the EXTEND-SHORTEST-PATHS which is $O(n^3)$. The total complexity is still $n(O(n)+O(n^3))$.

Problem 2

Exercise 26.1-2

MAXIMUM-FLOW-ALGORITHM(G, s, t)

```
1. initialize flow f to 0
2. while there exists an augmenting path p in the residual network Gf
3.         augment flow f along p
4. return f
```

Above is the basic outline of the maximum flow algorithm. We find an augmenting path from the source to the sink and then increase the flow along p with the minimum capacity that exists on that path. We can obviously not go beyond the minimum capacity of any edge.

Now, if we were to attach a supersource connected to each of the sources with edges of infinite capacity and a supersink connected to the sinks with edges of infinite capacity, the problem would not change. The solution is still guided by the paths that connect the sources with the sinks and the capacities on those paths. Moreover, we can now treat the different sources and sinks as normal nodes, with inflow and outflow, as the sources and sinks for the domain of solving our problem has changed. But, as above stated the solution and the flow function on different paths will not change as that is still guided by the different paths connecting the edges and their weights.

Exercise 26.2-10

We have a graph G=(V,E). Now, it is given that we already know the maximum flow network which means that we know the edges and respective their flows that form the maximum flow. Since, the graph has E edges, the maximum flow can also have a maximum of E edges. We start with the graph and try to figure out the augmenting paths that form the maximum flow. Here we will take an approach opposite to what we have for finding the maximum flow.

We follow the following steps:

FIND-MAXIMUM-FLOW-PATHS

```
1. while there exists path p from source to sink with positive flow
2.         find the minimum capacity edge on the path and subtract that amount from all edges on the path.
3.         add the path to list of paths.
```

Thus, we start with the maximum flow network and each step find a path from the source to the sink, whose capacity is non-zero. We reduce the flow of the edges on that path by the capacity of that path. This runs as long as there are no more paths p from source to sink which have non-zero capacity.
It can be seen that in every step, we are reducing the capacity of at least one edge, the minimum capacity edge on the path, to zero. Since, there are can be at most E edges there can be only at most E augmenting paths.

Problem 3

Problem 26-2 a.
We can find the minimum path cover by making a new graph $G' = (V', E')$, such that

$$V' = \{x_0, x_1, ...x_n\} \cup \{y_0, y_1, y_2, ...y_n\}$$

$$E' = \{(x_0, x_i) : i \in V\} \cup \{(y_0, y_i) : i \in V\} \cup (x_i, y_j) : (i, j) \in E$$

Now, the above graph is such that x_0 is the source and y_0 is the sink. Each x_i is connected to the source, each y_i is connected to the sink and each path (i, j) in E gives a path (x_i, y_j) in E'. Thus any vertex x_i in G' can have an incoming edge from x_0 and outgoing edges to different y_j. Similarly, any vertex y_j can have an outgoing edge to y_0 and incoming edges from various x_i. The flow of any edge in our graph G' is equal to 1.

Now, if we perform a maximum flow algorithm in our graph G', we will have any vertex x_i and y_j in a maximum of one path from x_0 to y_0. This is because the edge x_0 to x_i and the edge y_j to y_0 both have a flow of 1. Thus we will get various paths of the form $x_0- > x_i- > y_j- > y_0$, where x_i and y_j will be unique in each path. Also, since we are trying to maximize the flow, the solution will have maximum number of such paths and hence maximum number of vertices without any repetition. It will have one such path corresponding to each edge $(i, j) \in E$ of graph G.

Thus, using the solution of maximum flow of G' we can achieve a minimum path cover of graph G. We iterate over the augmenting paths $x_0- > x_i- > y_j- > y_0$ of G', and add the edge (i,j) in our minimum path cover of G. We are ensuring that any vertex i occurs only once and that the most number of vertices are covered which are the two necessary conditions of the minimum path algorithm.

The time complexity is defined by the time taken by the maximum flow algorithm $O(VE^2)$(using Edward Karp algorithm). The setting up of G' from G takes $O(V + E)$ and constructing the final solution takes at max $O(E)$ time, if all edges are covered. Thus the running time is defined by the time for the maximum flow algorithm.

Problem 26-2 b.
The algorithm will not work if there are cycles. Consider four edges (1,2),(2,3),(3,1) and (4,1). Now these four points make a triangle(1,2,3 together) and an edge (4,1) from an external point to the triangle. Now, the maximum cover for the above points is given by (4− > 1− > 2− > 3), but our algorithm can chose any of (4− > 1− > 2− > 3) or (3− > 1− > 2− > 3), without any guarantee.

Problem 4

a)
Lets say that the two graphs are given by $G = (V, E)$ and $G' = (V', E')$. And a function F, which maps the vertices from G to G'. Given these parameters, we can verify that the solution is correct in O(E) using the following algorithms.
Verify-graph-isomorphism(G, G', F)

```
1. for edge $(u, v) \in E$
2.     find $(u', v') \in G'$ corresponding to $(u, v) \in G$ using F
3.     if edge (u',v') does not exist in graph $G'$
4.         return false
5.     remove edge $(u', v')$ from list $E'$
6. if (E' is not empty)
7.     return false
8. return true
```

Hence, we go through the edges in G and check if a similar edge exists in G' using the mapping function F. If we come across an edge that exists in G but not in G' or vice versa, then we reject the solution. Since, we go over all the edges only once, this solution is linear in complexity with respect to the number of edges. Hence, it can be seen that the problem is NP.

b)
Given a path p, starting and ending vertices x and y and the graph G, we can verify that p is a Hamiltonian path in O(V) time. We start from x and go over the path as sequence of vertices given by p one by one. At the end of the traversal, if we are left with any vertex v not covered by the hamiltonian path, then we reject the path otherwise we accept it. This is linear with respect to the number of vertices V in the graph and hence the problem is NP.

Problem 5

Exercise 34.5-2
It is easy to see that the 0-1 integer-programming problem is NP. Given matrices A, x and b, we can compute the product Ax and verify if each element of the product is less than the corresponding element of b or not. We know that matrix multiplication is polynomial. Given, A is $m \times n$ and x is a $n \times 1$ matrix, the multiplication Ax is $O(mn)$.

In order to prove that this problem is NP-hard, we prove that 3-CNF-SAT $\leq$ 0-1 INTEGER PROGRAMMING. That is we take the 3-CNF-SAT problem and reduce it to an instance of the 0-1 INTEGER PROGRAMMING problem.

Lets assume that the we have a 3-CNF-SAT problem with n variables, for example $a \vee \sim b \vee c$. The algorithm that we follow to convert into the 0-1 integer-programming problem is to replace a variable a by a, its negation $\sim a$ by (1-a), such that the above problem $a \vee (\sim b) \vee c = 1$ reduces to $a + (1 - b) + c \geq 1$ or $-a + b - c \leq 0$, which is of the form $Ax \leq b$.

It can be seen that our function to translate a 3-CNF-SAT problem to a 0-1 integer linear programming problem is a simple mathematical transformation. Any solution of the first form guarantees a solution of the second and vice versa. Also, the translation is simple and polynomial. Thus, the proof:

$\Rightarrow$ Assume that X is a yes instance of 3-CNF-SAT problem. That is, for the values of variables in the circuit diagram, we can derive the value 1. Hence, the same values satisfy the given matrix form of 0-1 integer programming problem as well. Hence, X is a yes for 0-1 integer programming.

$\Leftarrow$ Similarly, if X is an instance of 0-1 integer programming, it is also an instance of the transformed 3-CNF-SAT problem.

Hence, this proves that the 0-1 integer programming problem is NP-complete.

Problem 6

Problem 34-2 a.
This problem can be solved in polynomial time. Lets say we have 'a' coins of x denomination and 'b' coins of y denomination. Then it follows that:

$$ax + by = V$$

Now, we need to find the values a' and b' so that,

$$a'x + b'y = V/2$$

$$(a - a')x + (b - b')y = V/2$$

We can easily iterate over values from 1 to a and 1 to b and check for each pair of values to see which satisfies our equation. This is clearly of the order of O(mn) and hence it is a polynomial solution.

Problem 34-2 b.
Since, the coins are in the powers of 2^n, this can also be solved in polynomial time. We can sort the coins by their denomination. Now we start with the highest denomination coin, lets say denomination is x, and move down the array giving coins of x alternatively to Bonnie and Clyde as long as they last. If we have even number of coins of x, they will be equally divided and we can move to the next highest denomination in a similar way. If, on the other hand we had odd number of coins of x and we gave the last x coin to Bonnie, then we need to give the next set of coins to Clyde, until their combined value is x. Now, Bonnie and Clyde have same amount of money and hence, we can continue doing the same with the rest of the money in different denominations.

The sorting will take $O(n \log n)$ time plus the division requires going through the entire stack once and hence is $O(n)$. Hence, the algorithm is polynomial in n.

Problem 34-2 c. This problem is NP-complete.

It is easy to see that the problem is NP. Given a solution of the division of checks we can calculate the sum for both the division and see if the sum is equal or not. Using this we can accept or reject the solution. For, the verification we need to go over each stack only once and calculate the sum. Hence, the verification is $O(n)$.

Now, we prove that this problem is NP-complete. We will prove this using SUBSET-SUM problem and show that the SUBSET-SUM problem can be reduced into our Bonnie and Clyde problem.

Lets recall the SUBSET-SUM problem. We have a set of numbers X, and we need to find a subset such that they sum to a bound S. Lets denote the sum of all numbers in X by T. We can transform this problem to the Bonnie and Clyde by considering cheques of value equal to each number in X. Then we add another cheque, whose value is equal to:

2S - T, if 2S > T
T - 2S, if 2S < T

It is important to note that in the first case, where 2S > T, the total sum of cheques combined is equal to 2S. In the second case, the total sum is equal to 2T-2S. Thus the division of cheques has to be S and T-S for the first and second case respectively.

Now it can be seen that the transformation is polynomial, since we have added only one check in the transformation, rest is all conversion from set of numbers to cheques. We now go on to prove that a solution of SUBSET-SUM is a solution of Bonnie and Clyde and vice versa.

$\Rightarrow$ Lets say that we have a solution of the subset-sum problem, that is we have a subset whose sum is equal to S
If lets say we consider the first case, where 2S > T, then a solution to the subset problem corresponds to the solution of the Bonnie and Clyde problem. One set of cheques is given by the solution of subset-sum problem, while the rest of the set(along with the added cheque) coresponds to the other division.
If we consider the second case, then we have a subset which sums to S, then the rest of the set of the numbers must sum to T-S, which is required for the equal division of cheques in the Bonnie and Clyde problem. The other division is the solution of the SUBSET-SUM plus the extra cheque added by us during the transformation.

$\Leftarrow$ If we have a solution to the Bonnie and Clyde problem that is we are able to divide the cheques in equal proportions:
Lets say we have the first case where, the total value of all cheques is 2S. This means that we have two divisions of total value S each. Then the division of the solution which does not have our added cheque is also the solution to the SUBSET-SUM problem.
If we have the second case, where the total value of cheques is 2T-2S, then the solution means two divisions of T-S each. If we consider the division which has our added cheque and remove the added cheque then its value will be T-S - (T-2S) = S. Thus this is the solution to our subset-sum problem.

Hence, it can be seen that SUBSET-SUM can be reduced to our Bonnie and Clyde problem in polynomial time and hence, our given problem is NP-complete.

Problem 34-2 d. This is also NP-complete.
It is easy to see that the problem is NP. Given a solution of the division of checks we can calculate the sum for both the division and see what is the difference between the two sums. Using this we can accept or reject the solution based on whether the difference is less than or greater than 100. For this we need to go over each division only once and calculate the sum. Hence, the verification is $O(n)$.

We prove this using the above problem of part c. We can convert the above problem of equal distribution to this problem by multiplying each value by 1000. If there exists a solution to equal distribution, then there exists a solution here. This is because, if there is even a difference of 1 in the above problem of equal distribution problem, then the difference here is 1000 and hence can not be a solution of this problem.

Since, the conversion is simply multiplication by 1000, it is polynomial time. Also, each solution to the the problem of equal distribution means a solution to our problem. Hence, this is NP-complete.