Homework 3 Solution:

Problem 1 Solution:

Knapsack Size = 12.

The items are: (6, 2), (3, 6), (4, 3), (1, 2), (8, 17), (5, 9).

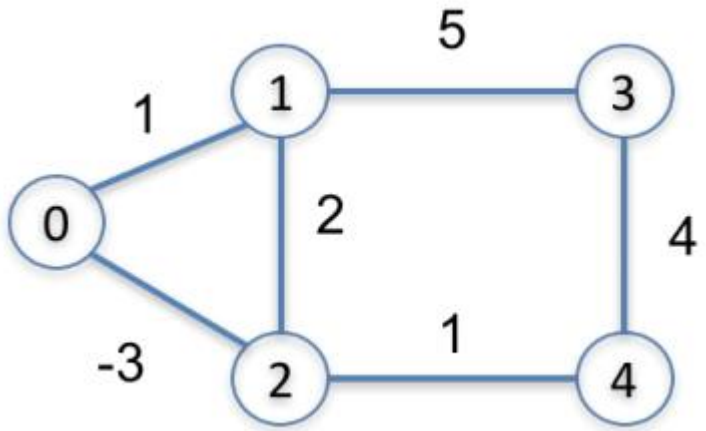
Weights { 6, 3, 4, 1, 8, 5 };

Values { 2, 6, 3, 2, 17, 9 };

Optimal set of items {(3, 6), (1, 2), (8, 17)}

	1	2	3	4	5	6	7	8	9	10	11	12
1	0|0	0|0	0|0	0|0	0|0	2|1	2|1	2|1	2|1	2|1	2|1	2|1
2	0|0	0|0	6|1	6|1	6|1	6|1	6|1	6|1	8|1	8|1	8|1	8|1
3	0|0	0|0	6|0	6|0	6|0	6|0	9|1	9|1	9|1	9|1	9|1	9|1
4	2|1	2|1	6|0	8|1	8|1	8|1	9|0	11|1	11|1	11|1	11|1	11|1
5	2|0	2|0	6|0	8|0	8|0	8|0	9|0	17|1	19|1	19|1	23|1	25|1
6	2|0	2|0	6|0	8|0	9|1	11|1	11|1	17|0	19|0	19|0	23|0	25|0

Problem 2 Solution:



Dijkstra

1. Source node 0, distance[0,Inf, Inf, Inf, Inf]
2. Min node 0, using node 0 update node 1 and node 2: distance[0,1, -3, Inf, Inf]
3. Min node 2, using node 2 update node 1 and node 4: distance[-6,-1, -3, Inf, -2]
4. Min node 4, using node 4 update node 3: distance[-3,-1, -3, 2, -2]

Bellman-Ford

Iteration 1: d[-6 -1 -3 2 -2]

Iteration 2: d[-12 -7 -9 -4 -8]

Iteration 3: d[-18 -13 -15 -10 -14]

Iteration 4: d[-24 -19 -21 -16 -20]

Floyd-Warshall

Considering 0:

	0	1	2	3	4
0	0	1	-3	inf	inf
1	1	0	-2	5	inf
2	-3	-2	-6	inf	1
3	inf	5	inf	0	4
4	inf	inf	1	4	0

Considering 1:

	0	1	2	3	4
0	0	1	-3	6	inf
1	1	0	-2	5	inf
2	-3	-2	-6	3	1
3	6	5	3	0	4
4	inf	inf	1	4	0

Considering 2:

	0	1	2	3	4
0	-6	-5	-9	-6	-8
1	-5	-4	-8	-5	-7
2	-9	-8	-12	-9	-11
3	-6	-5	-9	-18	-20
4	-8	-7	-11	-20	-22

Considering 3:

	0	1	2	3	4
0	-12	-11	-15	-24	-44
1	-11	-10	-14	-23	-43
2	-15	-14	-18	-27	-47
3	-24	-23	-27	-36	-56
4	-44	-43	-47	-56	-112

Considering 4:

	0	1	2	3	4
0	-88	-87	-91	-100	-156
1	-87	-86	-90	-99	-155
2	-91	-90	-94	-103	-159
3	-100	-99	-103	-112	-168
4	-156	-155	-159	-168	-224

Problem 3 Solution:

In here we consider adjacency edge list, instead of considering adjacency matrix.

Bellman-Ford Algorithm

```
for(i = 1 to n)
        Distance[i] = infinity;
Distance[source] = 0;


for(i=1 to n-1)
        for each edge (u, v) with weight w in edges:
                if distance[u] + w < distance[v]:
                        distance[v] := distance[u] + w
                        predecessor[v] := u
```

If we use adjacency matrix it will search all vertices for each edge. In that case, $E=V^2$. So the running time will be $O(V^3)$.

But if we use adjacency list it will update the distance array considering each edge. So the running time will be O(VE).

Problem 4 Solution:

For G(V,E) undirected graph, each vertex has an even degree. For this, we can find an eulerian circuit form of G(V,E). We can modify depth-first-search algo to direct the edges of G such that, the outdegree is equal to the indegree.

Algorithm:
Input: Undirected graph G(V,E) that each vertex has an even degree.
Output: Make undirected graph G(V,E) to directed graph where the outdegree is equal to the indegree.

Steps:

1. Maintain three array degree[], indegree[] and outdegree.
2. Initialize indegree[], outdegree with 0.
3. Initialize degree[] with number of edges corresponds to that edges. degree[v] contains number of degree of v vertex.
4. Pich any vertex v as source and call Modified_DFS()

Modified_DFS(v)

```
If(deg[v] ==0)
        return;
for each u in adj[v]
        if(indegree[v]≥outdegree[v])
        {
                outdegree[v] = outdegree[v]+1;
                degree[v] = degree[v] -1;
                adj[v][u] = 1; adj[u][v] =0;
                degree[u] = degree[u] -1;
                indegree[u] = indegree[u] +1;
                Modified_DFS(u);

        }
```

Running of Modified_DFS() is O(|V|+ |E|)

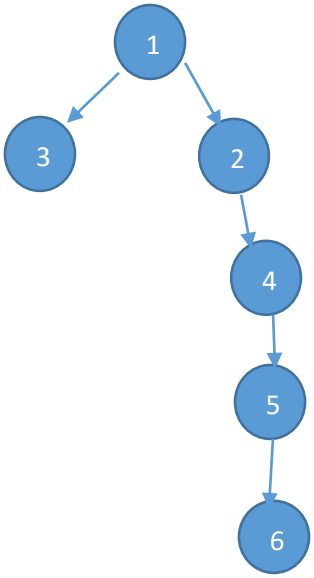
Problem 5 Solution:

A Hamiltonian Path is a simple path that includes all the vertices of the graph. As G(V,E) is a directed acyclic graph we can run topological sort and then check if it is unique or not. If the graph has unique topological orderning then G(V,E) has Hamiltonian path.

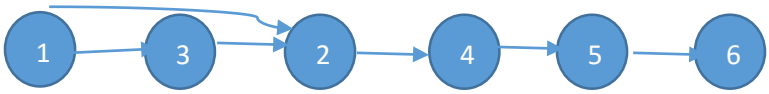
Algorithm:
Steps:
1. Run topological sort
2. Check Uniqueness of the topological sort.

Explanation:

For the following graph:



We get the following topological ordering



In the following ordering, if consecutive nodes are connected with edges then we can say the graph has unique topological order and exhibits Hamiltonian path. Otherwise that graph doesn't exhibit Hamiltonian path.

Problem 6 Solution:

Yes, T is still the MST. By adding constant c to every edge in the graph G(V,E) every MST's cost will be increased by (E-1)*c, where c is the constant. Though shortest path of G(V,E) may vary.

Problem 7 Solution:

Here, G(V,E) be an undirected weighted graph and T be an MST of G.
Now, we add a new vertex v with some weighted edges to T. If more than one edge from v connecting multiple vertices of T, then T becomes a graph consisting cycles. We need to eliminate the maximum valued edges from the cycles and make the graph into MST.

For this, we will take the newly added vertex v as source and run DFS() to look for a cycle. During DFS visit we will track the edge which has the maximum value. When we detect the cycle during DFS() visit we will mark the edge which has the maximum value for future removal. After DFS() visit completion we will remove all the marked edges from the graph and the resultant tree will be the MST.

Runnung Time: O(|V|+|E|)

Problem 8 Solution:

We can consider each currency as a node and each transaction between two currencies as a edge. Our goal is to find a cycle of transactions in the graph so that the product of the edge weights greater than initial value. If X be the edge weights, we can write it as …

$X_1.X_2.X_3. . . X_n > 1$
$=> 1/X_1. 1/X_2. 1/X_3. . .1/X_n < 1$
$=> \ln(1/X_1. 1/X_2. 1/X_3. . .1/X_n) < 0$
$=> \ln(1/X_1) + \ln(1/X_2) + \ln(1/X_3) + ……. + \ln(1/X_n) < 0$

Now, considering this, if we can detect a negative cycle from the graph, we can say there is an arbitrage. We can use Floyd-Warshall algorithm to detect the negative cycle

Algorithm:
Input: G(V,E) a digraph
Output: Detect negative cycle and discover arbitrage
Steps:

1. Initialize W[i][j] = inf where i not equal to j
2. Initialize W[i][j] = 0 where i == j
3. Initialize W[i][j] = ln(1/C[i[][j]), where C[i][j] is edge cost G(V,E)
4. for(i = 0; i < n; i++)

```
        for(j = 0; j < n; j++)
                for(k = 0; k < n; k++)
                {
                        if(W[i][j]>W[i][k]+ W[k][j])
                                W[i][j] = W[i][k]+ W[k][j]

                }
for(i=0;i<n;i++)
{
        if(W[i][i]<0)
        {
                return “Arbitrage”;
                break;
        }
}
return “No Arbitrage”;
```

Problem 9 Solution:

Removal of an edge:

If the deleted edge is not in the MST then the existing MST is the current MST of the updated graph. If the deleted edge is in the MST then removal of the edge will create two new trees T1 and T2. We have to find the minimum weight edge that connects these two components.

Algorithm:

Steps:

1. Run DFS() and find all the vertices V1 of T1.
2. Run DFS() and find all the vertices V2 of T2 .
3. Now in the graph G(V,E), run a linear search on the vertices V1 and find the minimum weigthed edge e(u,v) where u ϵ V1 and v ϵ V2.

Removal of a node:

Run Kruskal algorithm and reorganize the tree from the graph.