**Assignment 3**

CS 362

Baker, Alex

# Problem 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Item\Size | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| (0,0) | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |
| (1,2) | 0/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 | 2/0 |
| (3,6) | 0/0 | 2/1 | 2/1 | 6/0 | 6/0 | 6/0 | 6/0 | 6/0 | 6/0 | 6/0 | 6/0 | 6/0 | 6/0 |
| (4,3) | 0/0 | 2/1 | 2/1 | 6/1 | 6/1 | 6/1 | 6/1 | 9/0 | 9/0 | 9/0 | 9/0 | 9/0 | 9/0 |
| (5,9) | 0/0 | 2/1 | 2/1 | 6/1 | 6/1 | 9/0 | 11/0 | 11/0 | 14/0 | 14/0 | 14/0 | 14/0 | 18/0 |
| (6,2) | 0/0 | 2/1 | 2/1 | 6/1 | 6/1 | 9/1 | 11/1 | 11/1 | 14/1 | 14/1 | 14/1 | 14/1 | 18/1 |

Items (3,6), (4,3), (5,9) are picked for a total value of 18.

# Problem 2

## Bellman-Ford

## Floyd-Washall

## Dijkstra’s

This will not run since there is a negative edge in the graph

# Problem 3

Bellman-Ford can be reduced to O(|V||E|) by looking for negative cycles last. This reduces the need for the third inner loop until the end. Instead, the vertices just need to be looped through once, and the edges will be looped through and relaxed for each iteration of the vertices. this will have a double nested for loop of vertices and edges, giving O(|V||E|). When these loops terminate, the final result can loop through all of the edges to find any negative loops, and would just add a constant |E| time, so it will still run in O(|V||E|) time.

# Problem 4

# Problem 5

## Algorithm

In order to determine if a Hamiltonian path exists for a Given DAG G, first find a topological sort of G. As shown in class this can be done in O(|V| + |E|). A Hamiltonian path exists if there is an edge between all consecutive vertices in the topological sort. This works because if there is not an edge joining to adjacent vertexes, then one of the vertexes will be skipped when traversing the topological sort. Since you cannot go backwards in a topological sort, one vertex would not be in the path, therefore making the path across the topological sort not Hamiltonian.

## Pseudo Code

G = input graph

T = TopologicalSort(G)

**For** every vertex v **in** T

**If** no edge exists between v[i] and v[i+1]

**Return** no Hamiltonian path exists

**Return** Hamiltonian path exists

## Time Complexity

Finding the topological sort of a graph G with |V| vertices and |E| edges can be done in O(|V| + |E|) as shown in class. To determine if all the vertices are touched when traversing the topological sort can be done in O(|V|). So the total running time of this algorithm is O(|V| + |E|).

# Problem 6

## Claim

If T is a MST of G, then if every edge in G in increased by a constant c, then T is still an MST of G

## Proof: Contradiction

Let T be a MST of G. If c is a constant which increases every edge in G, then for every edge in G: cost(e) + c > cost(e), which means c > 0. Let G’ be the graph where the constant c was added to every edge and let T’ be the MST of G’ and assume T != T’. If T != T’, then there must be a pair of edges a and b in G where a > b and in G’ b + c > a + c. However, this implies b > a, which is a contradiction.

# Problem 7

Determine if T is still an MST of G if an edge e is added to G.

## Algorithm

Let e be the edge added to G which connects nodes v and w with cost c. Let P be the minimum cost path from v to w and let m be the highest cost edge in P. If c > m, then T is still a MST. If c < m, then T is no longer an MST of G because there is a new edge e with cost c that is lower than the previous edge in T, and can therefore be replace with edge e.

## Pseudo Code

G = input graph

T = MST(G)

e = new edge from v to w with cost c

P = BFS(v,w)

m = largest cost edge in P

**If** c > m

**Return** true

**Else**

**Return** false

## Time Complexity

Finding the BFS of G can be done in O(|V| + |E|) as show in class, so this algorithm runs in O(|V| + |E|).

# Problem 8

This problem can view viewed as a weighted graph problem, where the n by n exchange table represents a graph, and the edge weights are the exchange rates and the vertices are different types of currencies. The shortest path between any two vertices can then be found which would give the best order to exchange money in to get the best value. In this case, a negative cycle in the shortest path would indicate an arbitrage. So to find an arbitrage on an n by n exchange table, a shortest path algorithm which can find negative cycles can determine if there is an arbitrage or not. In this case Bellman-Ford is capable of finding negative cycles and will run in O(|V||E|).

# Problem 9

Update a MST T after adding an edge e to G.

## Algorithm

This algorithm works similar to number 7. Let e be the edge added to G which connects nodes v and w with cost c. Let P be the minimum cost path from v to w and let f with cost m be the edge with highest cost in P. If c > m, then T is still a MST. If c < m, then T is no longer an MST, and can be fixed by removing f from T and adding e.

## Pseudo Code

G = input graph

T = MST(G)

e = new edge from v to w with cost c

P = BFS(v,w)

f = largest cost edge in P

**If** c > cost(f)

**Return** T

**Else**

**Return** T’ by swapping f with e

## Time Complexity

Finding the BFS of G can be done in O(|V| + |E|) as show in class, so this algorithm runs in O(|V| + |E|).