Homework 5 Solutions
CSI 503 Spring 2021

RAHUL BHENJALIA

May 3 2021

1 Problems

1. Run the Bellman-Ford algorithm on the directed graph of Figure 24.4 in CLRS using vertex y as the source. In the following table, show the values of d and π after each pass ($d(\pi)$).

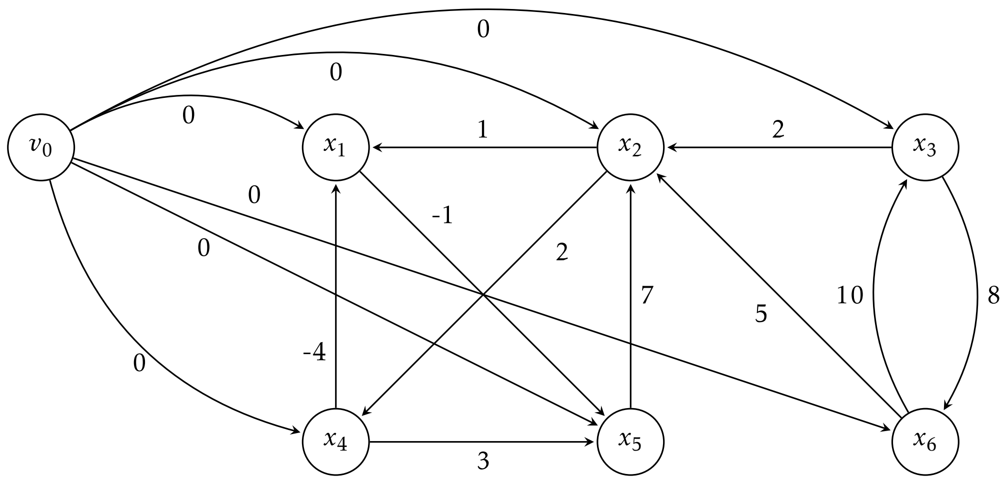
pass	s	t	x	y	z
initial	∞(NIL)	∞(NIL)	∞(NIL)	0	∞(NIL)
1	11	-5	-3	0	-9
2	-7	-5	-3	0	-9
3	-7	-5	-3	0	-9
4	-7	-5	-3	0	-9

Edge List: (y,x)(y,z)(x,t)(z,s)(z,x)(t,y)(t,z)(t,x)(s,y)(s,t)

2. Consider the following system of difference constraints:

$x_1 - x_2 \leq 1$
$x_1 - x_4 \leq -4$
$x_2 - x_3 \leq 2$
$x_2 - x_5 \leq 7$
$x_2 - x_6 \leq 5$
$x_3 - x_6 \leq 10$
$x_4 - x_2 \leq 2$
$x_5 - x_1 \leq -1$
$x_5 - x_4 \leq 3$
$x_6 - x_3 \leq 8$

(a) Form and draw the corresponding constraint graph.



(b) Use the Bellman-Ford algorithm to determine whether the system has a feasible solution or not. Show your calculated shortest paths.

pass	v_0	x_1	x_2	x_3	x_4	x_5	x_6
initial	0	∞	∞	∞	∞	∞	∞
1	0	-4	-3	0	0	-1	-8
2	0	-4	-3	0	0	-5	-8
3	0	-5	-3	0	-1	-5	-8
4	0	-5	-3	0	-1	-6	-8
5	0	-5	-3	0	-1	-6	-8
6	0	-5	-3	0	-1	-6	-8

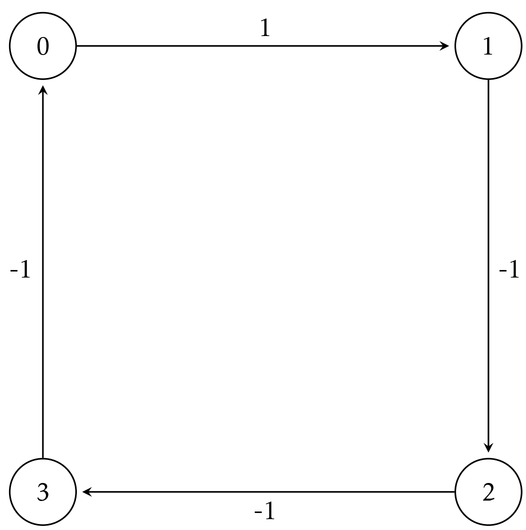
From the table we can conclude that the feasible solution for the problem is (-5,-3,0,-1,-6,-8)

3. We are interested in maintaining the transitive closure of a directed graph as the graph grows. Design an algorithm that can update the transitive closure of a graph in $O(V^2)$ time when an edge is inserted. You need to write the pseudocode for the following algorithm that takes current graph G, current transitive closure G*, and inserted edge (u, v) as inputs, and returns the updated transitive closure.
UpdateClosureAfterInsert(G, G*, u, v)
The algorithm below runs by exploring a pair vertices at a time so we can say that time complexity of the algorithm is $O(V^2)$
First the algorithm considers a pair of vertices in the graph G (a, b in this case)and make edges (u, a) and (v, b).
Then we create a transitive closure for the new graph that contains the new vertices (u, v).
Now, the only way the solution exists is if the transitive closure for the graph contains edges (u, a) and (v, b).

.
.
ALGORITHM:
UpdateClosureAfterInsert(G, G*, u, v)
```
1. //Input: A Graph G = (V,E)
2. for every pair (a,b) of vertices in V do
3. Let H = G (copy of original graph)
4.       createEdge-in-H – > (u,a) //append to H graph
5.       createEdge-in-H – > (b,v) //append to H graph
6.       result = CreateTransitiveClosure(H)
creating Transitive closure of new graph H which has added edges
8.       if result has edges(u,a) and(b,v) then do
9.             return result
10.       else do
11.             return "No Result"
```

4. The Floyd-Warshall algorithm fails (does not work correctly) if the input graph has a negative-weight cycle

   (a) Draw a small graph with a negative-weight cycle. Demonstrate the result of running Floyd-Warshall on it (show $D^{(k)}$ for k = 1, . . . , n).



We can apply the Floyed Warshall algorithm and construct the following table for the given negative weighted graph.

	0	**1**	2	**3**
0	-2	1	0	-1
1	-3	-2	-1	-2
2	-2	-1	-2	-1
3	-1	0	-1	-2

(b) In general, how can you use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle?

First, Since we used a very simple version of the graph, the data we have is little unusual than normal. As we can see in the diagonal of the table above, where distance is -2. So we can conclusively say that if the distance of or from any vertex is negative, we can end up with negative-weighted cycle.

5. We are given a directed graph G = (V, E) on which each edge (u, v) ∈ E has an associated value r(u, v), which is a real number in the range 0 ≤ r(u, v) ≤ 1 that represents the reliability of a communication channel from vertex u to vertex v. We interpret r(u, v) as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

(a) Provide the pseudocode of your algorithm.

We can modify the DIJKSTRA algorithm from the textbook to meet then requirements.

.

.

```
Most-Reliable-Path(u,v)
1. //Input: A directed Graph G = (V,E)
2. //Output: The most reliable path between two vertices.
3. w(u,v) //weight of those 2 vertices.
4. return (1/r(u,v))
5. Dijkstra(G,w,s)
6.      Initialize-Single-Source(G,s)
7.      S ← ∅
8.      Q ← V[G]
9.      While Q ≠ ∅
10.         do u ← EXTRACT - MIN(Q)
11.             S ← S ∪ {u}
12.             for each vertex v ∈ Adj[u]
13.                 do Relax(u,v,w)
```

(b) Show the correctness of your algorithm.

Let's consider we have an input graph G in the form (x,y,w), with x and y are vertices and w is the weight: [(2,1,1),(4,1,-4)(3,2,2)(5,2,7)(6,2,5)(6,3,10)(2,4,2)(1,5,-1)(4,5,3)(3,6,8)]

We have minimum path which is $(2- > 4- > 1- > 5)$ at weight -3

THe algorithm returns minimum weight on step 10, which proves that as we go through graph we will have minimum weight return.

(c) Analyze the time complexity of your algorithm.

Time complexity : O(E + VlogV)