Assignment 02
Shortest Paths

Please submit your solutions in PDF format. The PDF must be typed, NOT handwritten. Solution for each problem must start on a new page. The solutions should be concise; complicated and half-witted solutions might receive low marks, even when they are correct. Solutions should be submitted on the course website.

Problem 1: Collaborators [2 points]

List the name of the collaborators for this assignment. If you did not collaborate with anyone, write "None" (without the quotes).

Solution:

Bakhtiar Hasan

Problem 2: Dijkstra Practice [12 points]

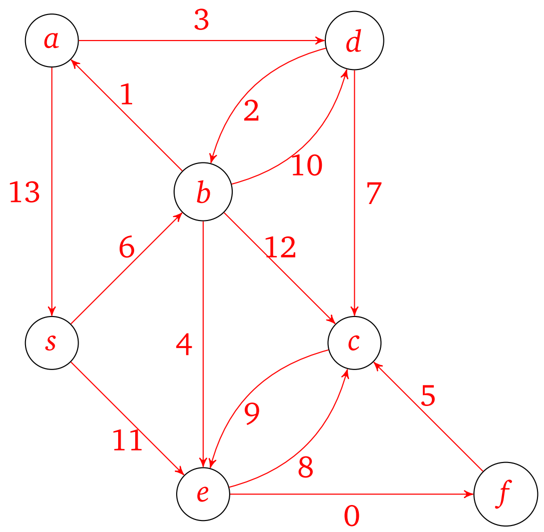
Consider the following representation of a graph:

Vertex 1	Vertex 2	W
a	s	13
a	d	3
b	a	1
b	d	10
b	e	4
b	c	12
c	e	9
d	b	2
d	c	7
e	f	0
e	c	8
f	c	5
s	b	6
s	e	11

Each row of the table denotes that there is a directed edge from Vertex 1 to Vertex 2 with weight W. s is consider as the source node.

(a) [4 points] Draw the weighted graph.

Solution:



(b) [4 points] Assume that we run Dijkstra's Algorithm to find the shortest path here. Write down the weight of the shortest path from s to all nodes.

Solution:

The $\delta(s, v)$ values are as follows:

Vertex	a	b	c	d	e	f	s
$\delta(s, v)$	7	6	15	10	10	10	0

(c) [4 points] List the order in which the vertices are removed from the Dijkstra queue.

Solution:

The nodes are processed in the increasing order of $\delta(s, v)$, but there are some flexibility in the order that d, e, and f are processed: a tie arises between d and e, and if e is processed first then another tie arises between d and f. The three possible processing orders are $sba\{def, edf, efd\}c$.

Problem 3: Vanilla-BFS [4 points]

Without modifying BFS, can we use it to find the shortest path in a weighted graph? Justify your answer.

Solution:

We can use BFS in weighted graphs, with a little change in the graph. If the graph weights are positive integers, you can replace an edge with weight n with n edges with weight 1 and $(n-1)$ middle nodes. The complexity of this algorithm depends on the maximum weight of the edges. If the weight is quite large, the algorithm becomes infeasible.

Problem 4: Longest Paths [12 points]

A path in a graph is simple if it visits any vertex at most once. Your friend Starney Binson comes up to you with this ingenious idea of finding the longest simple path in a graph in polynomial time. He wants to do the following:

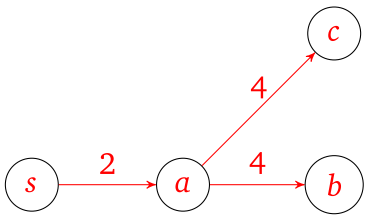
1. Negate all the edge weights of the graph
2. Run Bellman-Ford to find the shortest path
3. Negate all the shortest distances found

He claims that the distances will represent the length of the longest simple path in that graph.

(a) [4 points] Draw an example graph where Starney's Algorithm will work.

Solution:

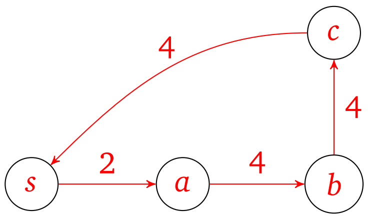
Any simple graph will do.



(b) [8 points] Argue whether Starney's Algorithm will work for all graphs or not. If not, provide a counterexample.

Solution:

Bellman-Ford will not necessarily compute the longest paths in the original graph, since there might be a negative-weight cycle reachable from the source, and the algorithm will abort. Finding a longest simple path in a graph is actually an NP-hard problem, meaning that it is unlikely that a polynomial time algorithms exists to solve this problem;



Problem 5: Weight! That's possible? [10 points]

Remember PERT charts from System Analysis and Design Course? No? Alright! Suppose a system analyst is trying to set up a realistic schedule for the data gathering and proposal phases of the systems analysis and design life cycle. The systems analyst looks over the situation and lists activities that need to be accomplished along the way. The list shows that some activities must precede other activities. That means if task u depends on task v, v must be completed before starting u. The time estimates are also determined for each activity. Given the list of activities including their duration and dependencies, describe an algorithm to compute the shortest time to complete all the activities.

Solution:

Construct a graph with a vertex for each activity, and a directed edge from activity u to activity v if u must be completed prior to v, weighted by the **negation** of the time required to complete activity u. Further, add vertices s and t, with a directed edge from s to every other vertex in the graph and directed edge from every vertex in the graph to t, each with weight 0. If there are n activities and m dependencies, this graph has a size $O(n+m)$. Use depth-first search to check whether this graph contains any cycles; if it does, then no activity on the cycle can be first, so no schedule can be created. Otherwise, finding the shortest path from s to t will correspond to the longest time to complete any sequence of activities, with all other activities being performed in parallel. Since the graph is acyclic, relax edges in topological sort order to compute the path of minimum weight in $O(n+m)$ linear time, which is faster than other methods.