# EC504 — Fall 2021 — Homework 5 Solutions

**Reading Assignment: CLRS Chapters 25, 26 and Appendix D.1**

1. (20 pts) Answer True or False to each of the questions below, and explain briefly why you think the answer is correct..

   (a) In the Bellman-Ford algorithm, the maximum number of times that the distance label of a given node $i$ can be reduced is less than or equal to $n-1$, where $n$ is the number of nodes in the graph.

   **Solution:** This answer depends a bit on the implementation, so it could be True or False. In the implementation where distance labels after k steps correspond to the shortest distance with k or less hops, this is true. For other implementations where you change distance labels immediately after scanning an arc, this is may not be true.

   (b) Bellman-Ford algorithm can be used for finding the longest path in a graph.

   **Solution:** False in general, but true provided there are no positive weight cycles.

   (c) In Floyd's algorithm for finding all-pairs shortest paths, after each major outer loop $k$, upper bounds are computed on shortest distances $D(i,j)$ between each pair of nodes $i,j$. These upper bounds correspond to the shortest distance among all paths between nodes $i$ and $j$ which use only intermediate nodes in $\{1,\ldots,k\}$.

   **Solution:** True. That is the proof of Floyd's algorithm.

   (d) Consider a directed, weighted graph where every arc in the graph has weight 100. For this graph, executing Dijkstra's algorithm to find a shortest path tree starting from a given node is equivalent to performing breadth first search.

   **Solution:** True. When the distances are the same on every arc, Dijkstra's algorithm scans the nodes in breadth first search order, since distance and number of hops are equivalent.

   (e) Consider a directed, capacitated graph, with origin node O and destination node D. The maximum flow which can be sent from O to D is equal to the minimum of the following two quantities: the sum of the capacities of the arcs leaving O, and the sum of the capacities of the arcs entering D.

   **Solution:** False. Removing all of the arcs from the origin is a cut. Removing all of the arcs of the destination is a cut. So this is an upper bound but the may be cut that this is smaller than either of these.

   (f) In Dijkstra's algorithm, the temporary distance labels D assigned to nodes which have not yet been scanned can be interpreted as the minimum distance among all paths with intermediate nodes restricted to the nodes already scanned.

   **Solution:** True. That is the property which is used to prove Dijkstra's algorithm is correct.

(g) The worst case complexity of the fastest minimum spanning tree algorithm is $O(Nlog(N))$, where $N$ is the number of nodes in the graph.

**Solution:** False. Every arc must be examined so the agorithm and only be bounded by $O(A)$ where in genera $A$ may grow as faxt asa $N^2$ for a dense graph.

(h) Consider an undirected graph where, for every pair of nodes $i, j$, there exists a unique simple path between them. This graph must be a tree.

**Solution:** True. Consider the path from one node (say 0) and all others. These path form a tree with $|N| - 1$. There can to be any other links since this would imply loops and a non-unique path. So this is a tree.

(i) Consider a minimum spanning tree T in a connected undirected graph (N, A) which has no two arcs with equal weights. Then, an arc i,j in T must be either the minimum weight arc connected to node i or the minimum weight arc connected to node j.

**Solution:** True. By contradiction, suppose there were a smaller arc our of i (or j) and you replace this arc by the (i,j) arc. This would again be a tree but with a reduced value so the orginal T was not mininum spanning.

(j) The following three algorithms are examples of greedy algorithms: Dijkstra's shortest path algorithm, Kruskal's minimum spanning tree algorithm, Prim's minimum spanning tree algorithm.

**Solution:** True. They are greedy algorithms, because they select candidates based on a local metric, and once a candidate is selected, there is no backtracking.

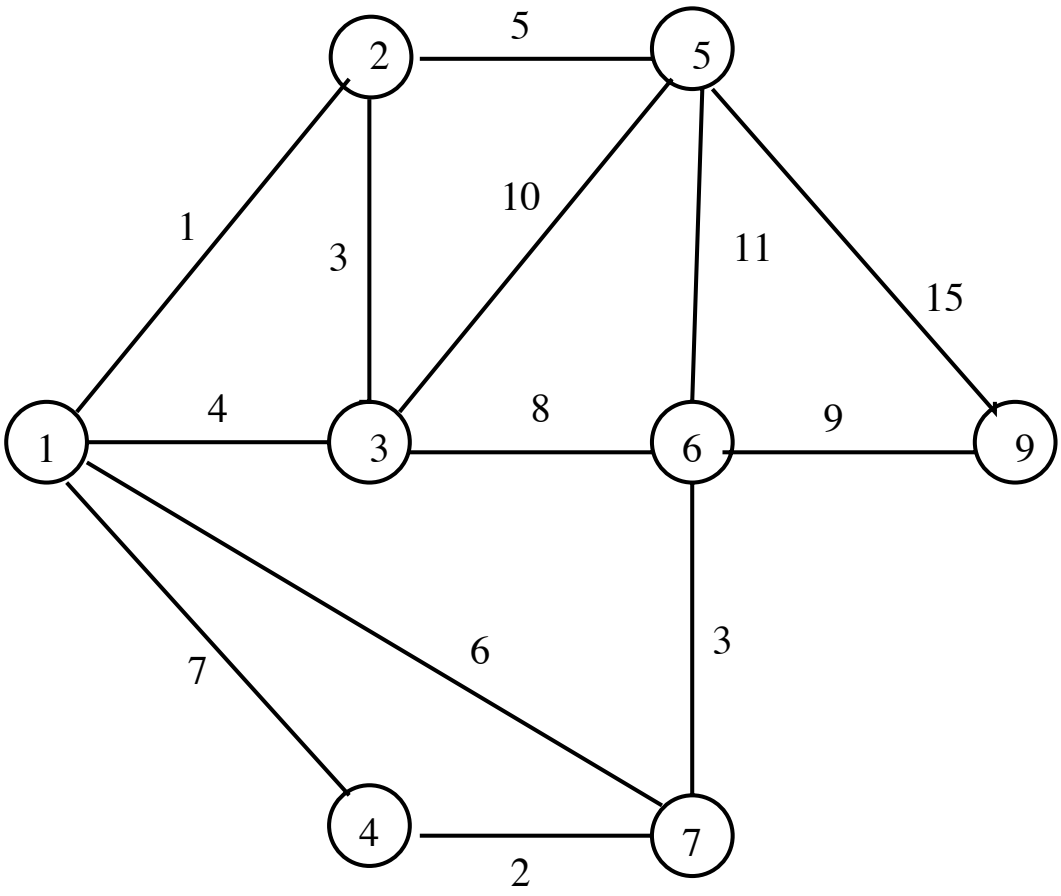



Figure 1:

2. (15 pts) Consider the undirected weighted graph in Figure 1. Show the distance estimates computed by each step of the Bellman-Ford algorithm for finding a shortest path tree in the graph, starting from node 1 to all other nodes. **For each cycle in the outer that adds an arc record in a table new distance $d(j)$ and the predecessor $p(j)$ at each node.**

**Solution:**

If you used the old graph, you need only say there is NO solution to due to negative cycles!

For the new graph: Since there is no 8-th node connecet to the graph its distance is alway inf (or just leave it out of the list!) A fake -1 node is the convention a node with no precedence set.

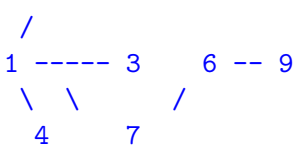
Follow Bellman Ford Algorithm on the slides. The min at each cycle is the min of using zero, one, one or two, one or two or three etc .. Note that if there is tie, the old path is used. There are lots of ties in this diagram!

Initial Distances: D(1) = 0; D(2) = inf; D(3) = 0; D(4) = inf; D(5) = inf;

```
                    D(6) = inf; D(7) = inf;  D(9) = inf;
Initial precedencd: P(1) = -1; P(2) = -1; P(3) = -1; P(4) = -1; P(5) = -1;
                    P(6) = -1; P(7) = -1;  P(9) = -1;

Add one arc:        D(1) = 0; D(2) = 1; D(3) = 4; D(4) = 7; D(5) = inf;
                    D(6) = inf; D(7) = 6;  D(9) = inf;
                    P(1) = -1; P(2) = 1; P(3) = 1; P(4) = 1; P(5) = -1;
                    P(6) = -1; P(7) = 1;  P(9) = -1;

two or less arcs    D(1) = 0; D(2) = 1; D(3) = 4; D(4) = 7; D(5) = 6;
                    D(6) = 9; D(7) = 6;  D(9) = inf;
                    P(1) = -1; P(2) = 1; P(3) = 1; P(4) = 1; P(5) = 2;
                    P(6) = 7; P(7) = 1;  P(9) = -1;

three or less arcs  D(1) = 0; D(2) = 1; D(3) = 4; D(4) = 7; D(5) = 6;
                    D(6) = 9; D(7) = 6;  D(9) = 18;
                    P(1) = -1; P(2) = 1; P(3) = 1; P(4) = 1; P(5) = 2;
                    P(6) = 7; P(7) = 1;  P(9) = 6;
```

There are NO Changes for adding more arc. (I know this was a rather simple example.)

Shortest path tree:



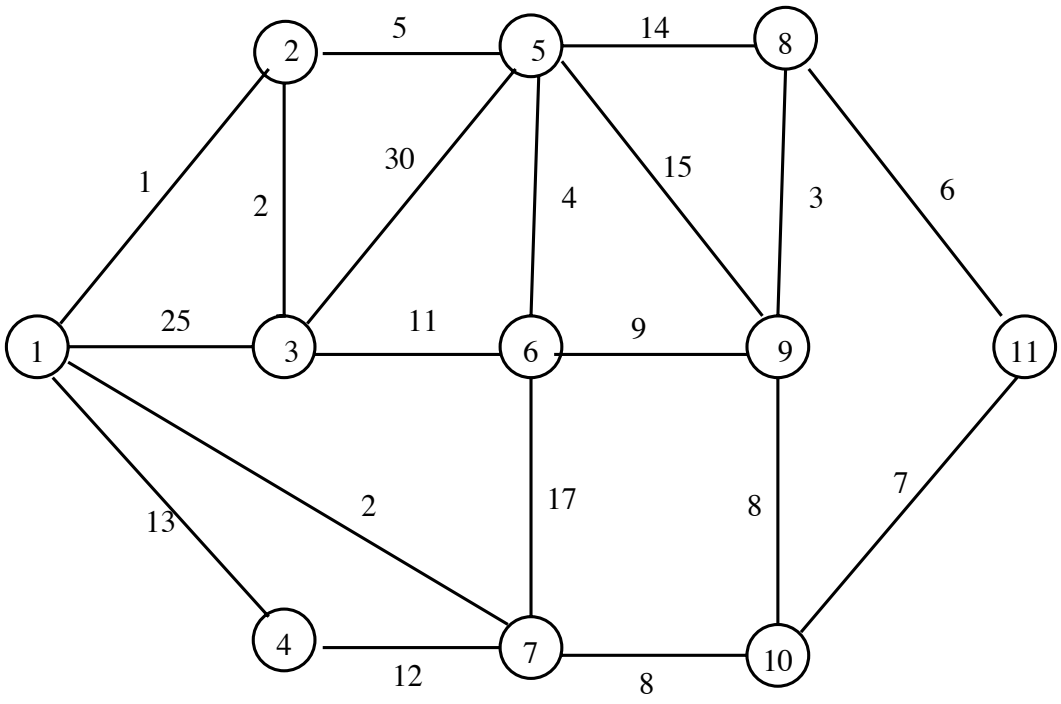



Figure 2:

3. (15 pts) Consider the weighted, undirected graph in Figure 2. Assume that the arcs can be traveled in both directions. Illustrate the steps of Dijkstra's algorithm for finding a shortest paths from node 1 to all others.

**Solution:** As before, keep track of distances which change using brackets, and delete nodes already scanned from distance list

```
Initial Distances: D(0)=0; D(1)=inf; D(2) = inf; D(3)=inf; D(4)=inf; D(5) = inf;
                   D(6) = inf; D(7) = inf; D(8) = inf; D(9) = inf; D(D) = inf
Scan node 0: D(1)=[3]; D(2) = [4]; D(3)=inf; D(4)=[7]; D(5) = inf;
                   D(6) = inf; D(7) = inf; D(8) = inf; D(9) = inf; D(D) = inf

Scan node 1: D(2) = 4; D(3)=[6]; D(4)=[6]; D(5) = inf;
                   D(6) = inf; D(7) = [9]; D(8) = inf; D(9) = inf; D(D) = inf
Scan node 2: D(3)=6; D(4)=6; D(5) = [7];
                   D(6) = inf; D(7) = 9; D(8) = inf; D(9) = inf; D(D) = inf
Scan node 3: D(4)=6; D(5) = 7;
                   D(6) = inf; D(7) = 9; D(8) = inf; D(9) = inf; D(D) = inf
Scan node 4: D(5) = 7;
                   D(6) = [8]; D(7) = 9; D(8) = [14]; D(9) = inf; D(D) = inf
Scan node 5: D(6) = 8; D(7) = 9; D(8) = 14; D(9) = [11]; D(D) = inf
Scan node 6: D(7) = 9; D(8) = [13]; D(9) = 11; D(D) = inf
Scan node 7: D(8) = 13; D(9) = 11; D(D) = inf
```

```
Scan node 8: D(9) = 11; D(D) = [17]
Scan node 9: D(D) = [16]

Shortest path in reverse:  D--9--5--2--0, length 16.
```

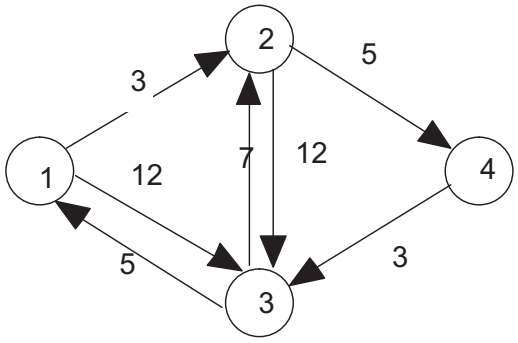



Figure 3:

4. (15 pts) Consider the graph in Figure 3. This is a directed graph. Use the Floyd-Warshall algorithm to find the shortest distance for all pairs of nodes. Show your work as a sequence of 4 by 4 tables.

   **Solution:** Here is the initial table, obtained from the arcs in the graph. the following tables expand the algorithm.

```
D(i,j)     j=1    j=2     j=3    j=4
i=1        0       3       12    inf
i=2        inf     0       12     5
i=3        5       7        0    inf
i=4        inf     inf      3     0

iteration: k = 1 as intermediate node
D(i,j)     j=1    j=2     j=3    j=4
i=1        0       3       12    inf
i=2        inf     0       12     5
i=3        5       7        0    inf
i=4        inf     inf      3     0

iteration: k = 2 as intermediate node
D(i,j)     j=1    j=2     j=3    j=4
i=1        0       3       12    [8]
i=2        inf     0       12     5
i=3        5       7        0    [12]
i=4        inf     inf      3     0

iteration: k = 3 as intermediate node
D(i,j)     j=1    j=2     j=3    j=4
i=1        0       3       12     8
i=2        [17]    0       12     5
i=3        5       7        0    12
i=4        [8]     [10]     3     0

iteration: k = 4 as intermediate node
```

| D(i,j) | j=1 | j=2 | j=3 | j=4 |
|---|---|---|---|---|
| i=1 | 0 | 3 | [11] | 8 |
| i=2 | [13] | 0 | [8] | 5 |
| i=3 | 5 | 7 | 0 | 12 |
| i=4 | 8 | 10 | 3 | 0 |

5. (15 pts) Explain how you modify Dijkstra's shortest path algorithms on a directed graph with non-negative edge weights to count the number of shortest paths from a given origin $n$ to a destination node $d$.

**Solution:** Note the following step in Dijkstra's algorithm for a directed graph $G = (V, E)$ with non-negative weight function $w()$: Let $D(k)$ denote the estimated distance to node $k$, and assume we are scanning node $n$ (that is, node $n$ is the node that we have pulled out of the priority queue that has the shortest distance estimate $D(n)$. The scan step is:

For each edge $(n, k) \in E$, if $D(n) + w(n, k) < D(k)$, then $D(k) = D(n) + w(n, k), Parent(k) = n$.

We modify Dijkstra's algorithm as follows: We initialize the number of shortest paths to each node as $Paths(k) = 0$ for $k \in V$. For the origin $o$, we set $Paths(o) = 1$.

Then, we modify the above step in Dijkstra's algorithm as follows:

For each $(n, k) \in E$,

- if $D(n) + w(n, k) < D(k)$, then $D(k) = D(n) + w(n, k), Parent(k) = n, Paths(k) = Paths(n)$
- else if $D(n) + w(n, k) == D(k)$, $Paths(k) = Paths(k) + Paths(n)$

What this does is, when we find a path whose distance estimate to node k is equal to the distance to node $k$, we increment the number of shortest paths by the number of shortest paths to this new parent. If we find a shorter distance estimate, we initialize the number of shortest paths found to the number of shortest paths to the parent. This has the same complexity and optimality properties of Dijkstra's algorithm.

6. (15 pts)In city streets, the length of an arc often depends on the time of day. Suppose you have a directed graph of streets connecting nodes that represent intermediate destinations, and you are given the travel time on the arc as a function of the time at which you start to travel that arc. Thus, for arc $e$, you are given $d_e(t)$, the time it takes to travel arc $e$ if you start at time $t$. These travel times must satisfy an interesting ordering property: You can't arrive earlier if you started later. That is, if $s < t$, then $s + d_e(s) <= t + d_e(t)$. Suppose that you start at time 0 at the origin node 1. Describe an algorithm for computing the minimum time path to all nodes when travel times on arcs are time dependent and satisfy the ordering property.

   **Solution:** Turns out Dijkstra's algorithm is exactly right for this problem. Dijkstra's algorithm scans nodes in the order of arrival time. The travel time on the arcs for the node that are being scanned can be computed because we know the time at which that node is reached. The only difference is that the travel times on arcs are computed as a function of the travel time to the origin node of the arc, when that node is permanently labeled by the algorithm.

7. (15 pts) In this problem, you are to implement an algorithm for the solution of shortest path problems using any variation you choose of the Bellman Ford and Dijkstra's algorithms for sparse graphs. You must implement both a variation of Bellman Ford and a variation of Dijkstra. You can choose to implement Dijkstra with either a priority heap, or without one. The objective of the project is to time these algorithms for solution on two large test graphs. The shell program `Shortest_paths.cpp` has timers already in it. The two test graphs are stored as `smallGraph.txt` and `Graph1.txt` with the progam on GitHub. The graphs are very sparse.

   Using the code provided or your own implementations, answer the questions below. Note that the nodes in the data (and in the code) will be indexed from 1 to number of nodes. The `Shortest_paths.cpp` is a fully functioning version of all code required with timers and output. You can use this or modify this as you see fit to report results for part (a) below.

   (a) The first part is to report the times required by each of the two algorithms to compute a full shortest path tree with origin node 5. Be very patient with the Bellman-Ford algorithm. It will converge some day on ACS, so you should bring along some reading material or use scripts. Print out as output the computation times and the shortest distances and the shortest paths from node 5 to nodes 573, 1021, 5783 and 9875 for the larger of the two graphs `Graph1.txt`for each of the algorithms. A hard copy of these result should be turned in with your source code on the SCC.

   (b) For extra credit, progam the $O(N^3)$ all to all distance finder. Sort the distances and plot the all distances to estimate the size *width* of graph in terms of the average distance in the distance matrx and the maximun distance. This program for the large graph will take a while so you should submit in batch queue.

   Comment: To debug and test your program you may want to make a new input file with very few node (4 or 5) that you can easily solve on scrap of paper. Trace the program with lots of output.