WORKSHEET, WEEK 09: SHORTEST PATHS AND MSTS

9.1 Shortest Paths: Dijkstra

(Drozdek2013, p.400) and (Goodrich2011, p.640) both define Dijkstra's algorithm. See also https://bit.ly/2JSXqMU. It is an efficient algorithm; it requires $O((m+n)\log_2 n)$ time, if we use priority queues; here $m = |E|$ is the number of edges and $n = |V|$ is the number of vertices in a graph.

In this exercise you do not need to implement a priority queue; assume that you can always pick the vertex with the smallest distance and add it to the set S of visited vertexes (those having distances already computed).

9.1.1 Problem

We start with the graph shown in Figure below:

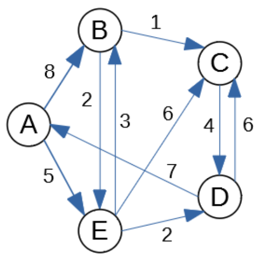


Fig. 1: Graph Diagram for Dijkstra's Algorithm

Vertex A will be your source vertex. (You can assume that the distance from A to itself is 0; initially all the other distances are infinite, but then Dijkstra's algorithm relaxes them).

(A) Run the Dijkstra's algorithm: At every phase write the current vertex v; the set of finished vertices and also a table showing the new distances to all A, B, C, D, E (and their parents) after the relaxations from v are performed. At the end of every phase highlight which vertex (among those not yet finished) has the minimum distance. This will become the current vertex in the next phase.

(B) After the algorithm finishes, summarize the answer: For each of the five vertices tell what is its minimum distance from the source. Also show what is the shortest path how to achieve that minimum distance.

9.2 Shortest Paths: Bellman-Ford

Let $G(V, E)$ be a directed graph. Let $w : E \to \mathbf{Z}$ be a function assigning integer weights to all the graph's edges and let $s \in V$ be the source vertex. Every vertex $v \in V$ stores $v.d$ – the current estimate of the distance from the source. A vertex also stores $v.p$ – its "parent" (the last vertex on the shortest path before reaching v). Bellman-Ford algorithm to find the minimum distance from s to all the other vertices is given by the following pseudocode:

```
BellmanFord($G, w, s$):
    for each vertex $v \in V$:        (initialize vertices to run shortest paths)
        $v.d = \infty$
        $v.p =$ Null
    $s.d = 0$        (the distance from source vertex to itself is 0)
    for $i = 1$ to $|V| - 1$        (repeat $|V| - 1$ times)
        for each edge $(u, v) \in E$
            if $v.d > u.d + w(u, v)$:        (relax an edge, if necessary)
                $v.d = u.d + w(u, v)$
                $v.p = u$
```

Question 2: Consider the graph in Figure:

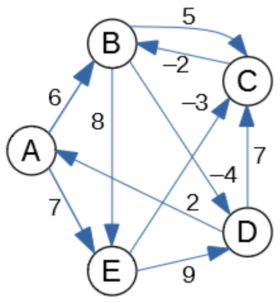


Fig. 2: Graph Diagram for Bellman Ford Algorithm

Let us pick vertex B as the *source vertex* for Bellman-Ford algorithm. (You could pick the source vertex differently, but then all the distance computations would be different as well.)

(A) Create a table showing all the changes to all the distances to A, B, C, D, E as the relaxations are performed. In a single iteration the same distance can be relaxed/improved multiple times (and you can use distances computed in the current phase to relax further edges). The table should display all $n - 1$ iterations (where $n = 5$ is the number of vertices). (*Sometimes it is worth running one more iteration to find possible negative loops*).

Note: Please make sure to release the edges in the alphabetical/lexicographical order: Regardless of which is your source, in every iteration the edges are always relaxed in this order:

$$AB, AE, BD, BE, CB, DA, DC, EC, ED.$$

In fact, any order can work; the only thing that matters is that you consider all the edges. But alphabetical ordering of edges makes the solution deterministic.

(B) Summarize the result: For each of the 5 vertices tell what is its minimum distance from the source. Also tell what is the shortest path how to get there. For example, if your source is E then you could claim that the shortest path $E \rightsquigarrow B$ is of length -5 and it consists of two edges $(E, C), (C, B)$.

Question 3: In this task the input graph is shown in the figure below.

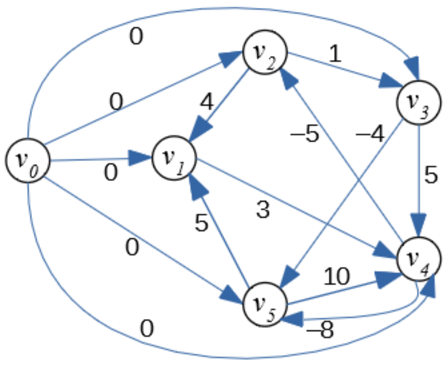


Fig. 3: A directed graph for Bellman-Ford Algorithm

(A) In your graph use the vertex $s = v_0$ as the *source vertex* for Bellman-Ford algorithm. Create a table showing the changes to all the distances to the vertices of the given graph every time a successful edge relaxing happens and some distance is reduced. You should run $n - 1$ phases of the Bellman-Ford algorithm (where n is the number of vertices). You can also stop earlier, if no further edge relaxations can happen.

Note: Please make sure to release the edges in the lexicographical order. For example, in a single phase the edge (v_1, v_4) is relaxed before the edge (v_2, v_1), since v_1 precedes v_2.

(B) Summarize the result: For each vertex tell what is its minimum distance from the source. Also tell what is the shortest path how to get there.

(C) Does the input graph contain negative cycles? Justify your answer.

9.3 Minimum Spanning Trees

(Goodrich2011, p.651) defines Prim's algorithm. It finds a minimum spanning tree in an undirected graph with given edge weights. See also https://bit.ly/2VLz3DK. It is an efficient algorithm; it requires $O((m + n) \log_2 n)$ time, if we use priority queues. In this exercise you do not need to implement a priority queue; assume that you can always compute the minimums in your head and grow the MST accordingly.

9.3.1 Problem

Question 4 (Prim's algorithm): Prim's algorithm for the graph shown in Figure:

(A) Vertex A will be your source vertex. It is the first vertex added to the MST vertice set S. At every step you find the lightest edge that connects some vertex in S to some vertex not in S. Add this new vertex to a graph and remember the edge you added. Show how the Prim's MST (Minimum Spanning Tree grows) one edge at a time.

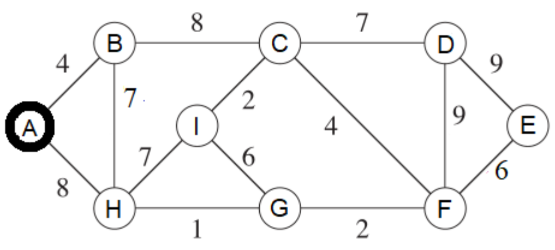


Fig. 4: Graph Diagram for Prim's Algorithm.

Note: In cases when there is a choice between multiple lightest edges of the same weight, pick the edge (v, w) with $v \in S$ and $w \notin S$ such that (v, w) lexicographically precedes any other lightest edge.

(B) Redraw the graph, highlight the edges selected for MST (make them bold or color them differently). Add up the total weight of the obtained MST and write this in your answer (it should be the minimum value among all the possible spanning trees in this graph).

Question 5 (Prim's algorithm): Denote the last three digits of your Student ID by a, b, c. Student ID often looks like this: `201RDBabc`, where a, b, c are digits. Compute three more digits x, y, z:

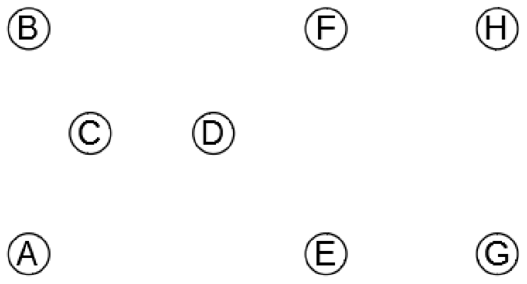
$$\begin{cases} x = (b + 4) \bmod 10 \\ y = (c + 4) \bmod 10 \\ z = (a + b + c) \bmod 10 \end{cases}$$

In this task the input graph $G = (V, E)$ is given by its adjacency matrix:

$$M_G = \begin{pmatrix} 0 & 0 & 5 & 8 & y & 0 & 0 & 0 \\ 0 & 0 & 3 & 7 & 0 & z & 0 & 0 \\ 5 & 3 & 0 & 3 & 0 & 0 & 0 & 0 \\ 8 & 7 & 3 & 0 & 1 & 7 & 0 & 0 \\ y & 0 & 0 & 1 & 0 & 6 & 9 & 6 \\ 0 & z & 0 & 7 & 6 & 0 & x & 2 \\ 0 & 0 & 0 & 0 & 9 & x & 0 & 7 \\ 0 & 0 & 0 & 0 & 6 & 2 & 7 & 0 \end{pmatrix}.$$

(A) Draw the graph as a diagram with nodes and edges. Replace x, y, z with values calculated from your Student ID. Label the vertices with letters A, B, C, D, E, F, G, H (they correspond to the consecutive rows and columns in the matrix).

If you wish, you can use the following layout (edges are not shown, but the vertice positions allow to draw the edges without much intersection). But you can use any other layout as well.



(B) Run Prim's algorithm to find MST using $r = A$ as the root. If you do not have time to redraw the graph many times, just show the table with $v.key$ values after each phase. (No need to show $v.p$, as the parents do not change and they are easy to find once you have the final rooted tree drawn.) The top of the table would look like this (it shows Phase 0 – the initial state before any edges have been added).

Phase	A	B	C	D	E	F	G	H
0 (initial state)	0	∞	∞	∞	∞	∞	∞	∞

(C) Summarize the result: Draw the MST obtained as the result of Prim's algorithm, find its total weight.

Question 6: Run Kruskal's algorithm on the same graph as in the Question 4.

(A) After each step when there is an edge connecting two sets of vertices, write that edge and show the partition where that edge connects two previously disjoined pieces in the forest of trees.

Note: If there are multiple lightest edges that can be used to connect two disjoined pieces, pick edge (v, w) which lexicographically precedes any other.

(B) Redraw the given graph (show the order how you added the edges in parentheses). Also compute the total weight of this MST.