HANDOUT 08: SHORTEST PATHS

This handout could help to prepare for the Written Assignment 08.

1.1 Dijkstra's Algorithm

(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).

1.1.1 Problem

We start with the graph shown in Figure below:

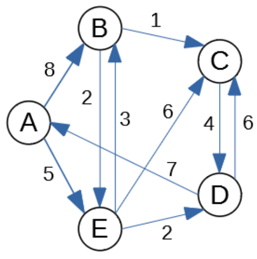


Fig. 1: Graph Diagram for Dijkstra's Algorithm

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.

1.1.2 Solution

(A) At every phase we select the minimum-distance vertex in the priority queue of vertices (not yet added to the set of finished vertices S). This becomes the current vertex v. After that we relax all the edges that go out from the current vertex v (if some distance decreases, we change the parent of this new vertex to become v).

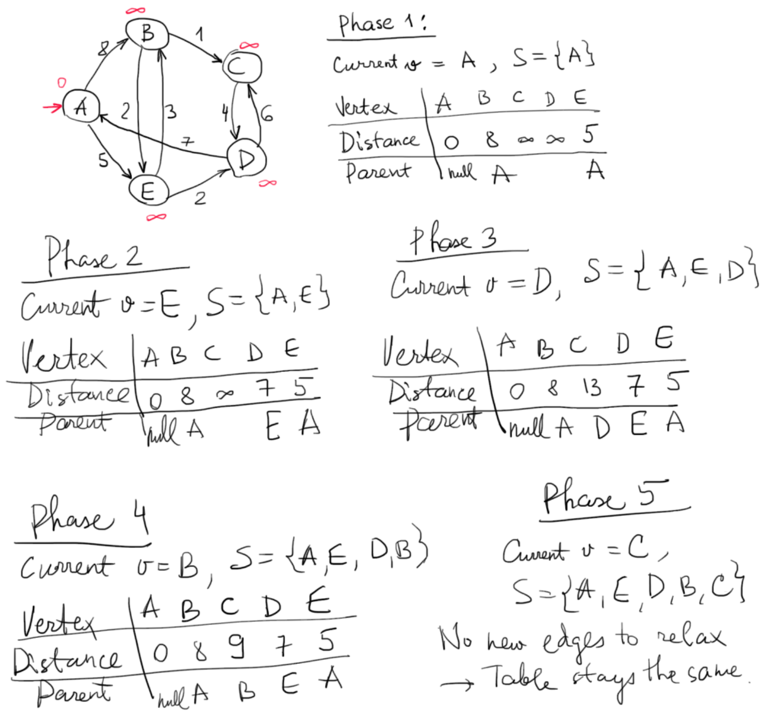


Fig. 2: **(A)** – Five Phases of Dijkstra's Algorithm

(B) The result of Dijkstra's algorithm can be summarized as shown below. For each vertex we specify the distance from A to that vertex (and also what is the shortest path to achieve it).

Vertex	Distance	Path
A	$d(A, A) = 0$	A
B	$d(A, B) = 8$	$A \to B$
C	$d(A, C) = 9$	$A \to B \to C$
D	$d(A, D) = 7$	$A \to E \to D$
E	$d(A, E) = 5$	$A \to E$

1.2 Bellman-Ford Algorithm

The Bellman-Ford algorithm solves the single source shortest paths problem in the case in which edge weights may be negative. It can work with directed graphs (and also undirected graphs; not discussed in this exercise). The algorithm initializes the distances to all the vertices u by $u.d = +\infty$. The only exception is the *source vertex*) which gets distance $s.d = 0$ (the distance to itself is 0).

After this initialization in a graph with n vertices it will perform $n-1$ identical iterations. In every iteration it considers all the edges in some order, and "relaxes" all the edges. See (Drozdek2013, p.402) and (Goodrich2011, p.640), where the edge relaxation procedure is described.

After that you can perform one last iteration with Bellman-Ford algorithm: If there are still relaxations that reduce distances even after n steps, this means that there is a negative loop in the original graph (and the shortest paths are not possible to compute as the distances can be reduced infinitely).

1.2.1 Problem

Consider the graph in Figure:

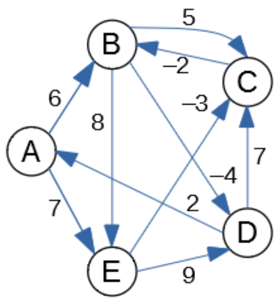


Fig. 3: Graph Diagram for Bellman Ford Algorithm

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:

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

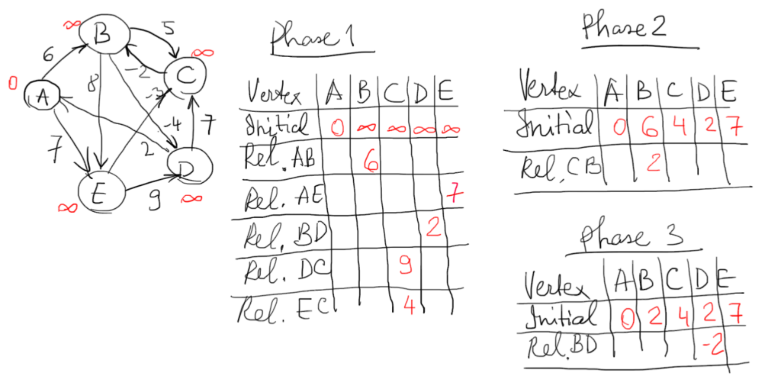
In fact, any order can work; the only thing that matters is that you consider all the edges. But alphabetical ordering of edges makes the solution deterministic.

(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 $(E, C), (C, B)$.

1.2.2 Solution

(A) In this case we only need to run three phases (not $n - 1 = 4$ phases), since all the distances become stable and do not change anymore after Phase 3. The tables show only those relaxed edges that lead to decreased distances.

(B) The result of Bellman-Ford's algorithm can be summarized as shown below. For each vertex we specify the distance from A to that vertex (and also what is the shortest path to achieve it).



Phase 1

Vertex	A	B	C	D	E
Initial	0	∞	∞	∞	∞
Rel. AB		6			
Rel. AE					7
Rel. BD				2	
Rel. DC			9		
Rel. EC			4		

Phase 2

Vertex	A	B	C	D	E
Initial	0	6	4	2	7
Rel. CB		2			

Phase 3

Vertex	A	B	C	D	E
Initial	0	2	4	2	7
Rel. BD				-2	

Fig. 4: **(A)** – Phases of Bellman-Ford's Algorithm

Vertex	Distance	Path
A	$d(A,A) = 0$	A
B	$d(A,B) = 2$	$A \to E \to C \to B$
C	$d(A,C) = 4$	$A \to E \to C$
D	$d(A,D) = -2$	$A \to E \to C \to B \to D$
E	$d(A,E) = 7$	$A \to E$