Algorithms (2IL15): Homework Set B.1

Exercises for Lecture 5

B.1-1 (1 point) Consider Dijkstra’s algorithm, as given on slide 17 of Lecture 5. If \( d(v) \) changes when performing a Relax operation on edge \((u,v)\), we have to update \( Q \). To this end the statement Decrease-Key\((Q,v,d(v))\) is added as line 4 of Relax. Explain why we can be sure that \( v \) is still stored in \( Q \) when the Decrease-Key-operation is executed.

B.1-2 (1 point) Suppose we have a graph \( G = (V,E) \) such that all edge weights are either 1 or 2. Explain that in this case Dijkstra’s algorithm can be implemented in such a way that it runs in \( O(|V| + |E|) \) time.

B.1-3 (1+1 point) Let \( G = (V,E) \) be a weighted graph, where all edge weights are non-negative. Let \( s,t \) be two vertices in \( V \). We want to compute \( \delta(s,t) \), the distance from \( s \) to \( t \). We can do this by running Dijkstra\((G,s)\) (see slide 17 of Lecture 5) and then reporting \( d(t) \). However, we can also stop running Dijkstra’s algorithm as a soon as we know we have computed \( \delta(s,t) \).

(i) Explain how to modify the pseudocode of Dijkstra\((G,s)\) to achieve this.

(ii) Give an example of a graph with \( \Theta(|V|^2) \) edges, and vertices \( s \) and \( t \), such that this modified version of Dijkstra’s algorithm still relaxes all the edges before \( \delta(s,t) \) has been determined. Your example should be a generic example, that is, you should describe an example for arbitrarily large values of \( |V| \). Your example can use a directed graph or an undirected graph, but \( t \) must be reachable from \( s \).

B.1-4 (1+1 point) Let \( G = (V,E) \) be a weighted, directed graph with non-negative edge weights, and let \( A \subset V \) be a non-empty subset of vertices. Suppose we modify the procedure Initialize-Single-Source as follows.

Algorithm New-Initialize\((G,A)\)
1. for each vertex \( v \in V \) do \( d(v) \leftarrow \infty; \pi(v) \leftarrow \text{nil} \)
2. for each vertex \( v \in A \) do \( d(v) \leftarrow 0 \)

(Note that in 3rd edition of the book [CLRS] the notation \( v_i.d \) is used instead of \( d(v_i) \).)

Now suppose we run Dijkstra’s algorithm on \( G \), using New-Initialize\((G,A)\) instead of Initialize-Single-Source\((G,s)\).

(i) What does this version of the Dijkstra’s algorithm compute? (You don’t have to give a proof, just state what the algorithm computes.)

(ii) Consider the loop-invariant of the original Dijkstra’s algorithm as stated in the slides (slide no. 18). Give a similar invariant for the new version of Dijkstra’s algorithm, prove that the invariant holds initially, and prove that it is indeed maintained.

B.1-5 (1+1 point) Consider a graph \( G = (V,E) \) that is a complete binary tree consisting of vertices \( v_1,\ldots,v_n \), with \( n = 2^k - 1 \) for some \( k \geq 1 \), where each edge has weight 2; see Fig. 1 for an example. We run the Bellman-Ford algorithm on \( G \) with \( s = v_1 \). The Bellman-Ford algorithm consists of \( n-1 \) rounds, where in each round all edges are relaxed in an arbitrary order. Let’s assume that the order in each round is the same.

(i) Is there an order in which to relax the edges such that already after the first round of relaxations we have \( d(v_i) = \delta(v_1,v_i) \) for all \( i = 1,\ldots,n \)? If so, what is this order? If not, explain why this is not possible. (See the slides for the meaning of the notation \( d(v_i) \) and \( \delta(v_1,v_i) \). Note that in 3rd edition of the book [CLRS] the notation \( v_i.d \) is used instead of \( d(v_i) \).)
(ii) Give an order in which to relax the edges in each round that maximizes the number of rounds needed before \( d(v_i) = \delta(v_1, v_i) \) for all \( i = 1, \ldots, n \). Indicate what the value of \( d(v_i) \) is after the \( j \)-th round for \( 1 \leq i \leq n \) and \( 1 \leq j \leq n - 1 \) (if the edges are relaxed in the order you have defined).

Exercises for Lecture 6

B.1-6 (1 point) Let \( G = (V, E) \) be a directed graph with non-negative edge weights, and let \( T \subseteq V \) be a subset of the vertices. We want to compute for each vertex \( u \in V \) the minimum distance to any vertex in \( T \). We denote this minimum distance by \( \delta(u, T) \). Thus \( \delta(u, T) = \min_{v \in T} \delta(u, v) \), where \( \delta(u, v) \) denotes the distance from \( u \) to \( v \).

Explain how to modify \( G \) into a graph \( \tilde{G} \) such that the values \( \delta(u, T) \) (for all \( u \in V \)) can be computed by running Dijkstra’s algorithm once on \( \tilde{G} \). (It is not allowed to modify Dijkstra’s algorithm, or the routine \text{Initialize-Single-Source}; you are only allowed to modify the graph \( G \).)

B.1-7 (1 point) Suppose someone has an algorithm \text{Magic-APSP}(G) that solves the all-pairs shortest path problem in \( O(|V| + |E|) \) time on any given weighted, directed graph \( G = (V, E) \) in which every node has at most two incoming edges and two outgoing edges. (The edge weights can be negative, zero, and positive.) Explain how to use this algorithm to solve the all-pairs shortest path problem in \( O(|V| + |E|) \) time on any weighted, directed graph \( G \) (that is, even when \( G \) has nodes with more than two incoming or outgoing edges).

B.1-8 (2 points) The all-pairs shortest-paths problem can be solved by computing the matrix “product” \( W^{n-1} \). Explain how the single-source shortest path problem can be solved by computing the “product” of several matrices and a vector. \textit{Hint:} What is the meaning of the rows and columns in the matrix \( W^{n-1} \)?

B.1-9 (1 point) Johnson’s algorithm changes the edge weights in a certain way to ensure that they are all non-negative, so that Dijkstra’s algorithm can be used. Suppose that we change the edge weights in a different way, namely as follows: we compute the minimum weight among all the edges, \( w_{\text{min}} \), and then change the weight of each edge \((u, v)\) to \( \tilde{w}(u, v) = w(u, v) - w_{\text{min}} \). Does this lead to a correct algorithm? Explain your answer.