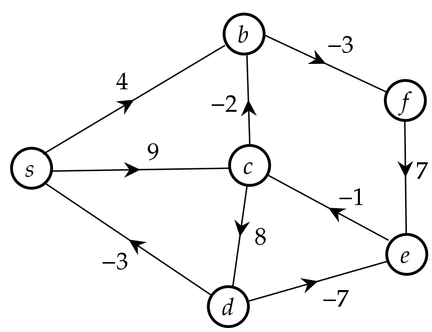
CMPSC 465 Data Structures & Algorithms

Fall 2024 Mingfu Shao and Debarati Das

HW 6

1. (0 pts.) (a). Run the Bellman-Ford algorithm on the graph given below, starting at the given source s. Whenever there is a choice of edges to update, always pick the one that is lexicographically/alphabetically first. Show the dist array after each round of updates.

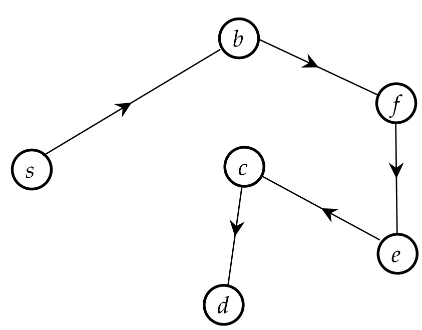


Solution: The initial dist array is given in the first row of the matrix given below; after i-th round it is given in the $(i+1)$-th row of the matrix, $1 \leq i \leq 5$.

$$
dist = \begin{array}{c}
\begin{matrix} s & b & c & d & e & f \end{matrix} \\
\begin{bmatrix}
0 & \infty & \infty & \infty & \infty & \infty \\
0 & 4 & 9 & 17 & 8 & 1 \\
0 & 4 & 7 & 17 & 8 & 1 \\
0 & 4 & 7 & 15 & 8 & 1 \\
0 & 4 & 7 & 15 & 8 & 1 \\
0 & 4 & 7 & 15 & 8 & 1
\end{bmatrix}
\end{array}
$$

(b). Draw the shortest-path tree.

Solution:



2. (0 pts.) You are given a directed graph $G = (V, E)$ and a vertex $s \in V$. Each edge e is assigned with a length $l(e)$, possibly with negative value. We know that there is no negative cycle in this graph, and that the only negative edges are the ones that leave the vertex s. That is, $l(s, v) < 0$ for all $(s, v) \in E$, and $l(u, v) > 0$ for all $u \neq s$ and $(u, v) \in E$. If we run Dijkstra's algorithm starting at s, will it fail on this graph? Prove your conclusion.

   Solution: No, it won't fail on this graph.

   Proof: The correctness of Dijkstra's algorithm depends on the claim that the next closest vertex, i.e., v^*_{k+1}, must be within one-edge extension of R_k i.e. $v^*_{k+1} = \arg\min_{v \notin R_k, u \in R_k, (u,v) \in E}(distance(s, u) + l(u, v))$. The proof of this statement only uses that, the edges leaving any $v \notin R_k$ have positive edge length. In other words, the proof only requires that the one-edge extension is always preferred than the two-edge extension (first edge being (u, v), second edge being (v, w) for some w) will always be longer since the second-edge has positive edge length. In our case, the second-edge always has positive length. This is because, although edges leaving s have negative edge length, s will always stay in R_k and consequently edges leaving s will always be part of one-edge extension.

3. (0 pts.) You are given a directed graph (V, E) with positive edge length $l(e)$ for any $e \in E$, and a positive weight $w(v)$ for any $v \in V$. Now we define the length of a path p from u to v as the sum of the lengths of all the edges in p plus the sum of the weights of all the vertices in p. You are also given a source $s \in V$ and it happens that $w(s) = 0$. Design an algorithm to find the length of the shortest path (in this new definition) from s to all vertices in V. Your algorithm should run in $O((|V| + |E|) \log |V|)$ time.

   Solution: We create a new graph G'. For each vertex $v \in V$, we add two vertices v_i and v_o to G', and add a new edge (v_i, v_o) with length being the weight of vertex v, i.e., $w(v)$. For each edge $e = (u, v)$ in G, we add an edge (u_o, v_i) to G' and its length keeps the same, i.e., $l(e)$. Notice that in G' nodes are not associated with weights and only edges are associated with lengths.

   Clearly, any path $p = s \to w \to \cdots \to u \to v$ in G corresponds to its counterpart $p' = s_i \to s_o \to w_i \to w_o \to \cdots \to u_i \to u_o \to v_i \to v_o$. And the length of p, under the new definition (i.e., sum of vertex weights and edge lengths), is exactly the length of p' in G', under the regular definition (i.e., sum of edge lengths). Therefore, we can run Dijkstra's algorithm on G', with s_i being the source vertex. The distance from s_i to v_o in G' will give exactly the length of the shortest path from s to v in G (under new definition).

   Graph G' contains $2|V|$ vertices and $(|V| + |E|)$ edges. The Dijkstra's algorithm, which dominates the running time, takes $O(2|V| + |V| + |E|) \cdot \log(2|V|) = O(|V| + |E|) \cdot \log(|V|)$ time.

4. (0 pts.) Shortest path algorithms can be applied in currency trading. Let $c_1, c_2, \cdots, c_n$ be various currencies. For any two currencies c_i and c_j, there is an exchange rate $r_{i,j}$; this means that you can purchase $r_{i,j}$ units of currency c_j in exchange for one unit of c_i. These exchange rates satisfy the condition that $r_{i,j} \cdot r_{j,i} < 1$, so that if you start with a unit of currency c_i, change it into currency c_j and then convert back to currency c_i, you end up with less than one unit of currency c_i (the difference is the cost of the transaction).

   (a). Give an efficient algorithm for the following problem: given a set of exchange rates $r_{i,j}$, and two currencies s and t, find the most advantageous sequence of currency exchanges for converting currency s into currency t. Toward this goal, you should represent the currencies and rates by a graph whose edge lengths are real numbers.

   Solution. We can transform the currency exchange problem into the shortest path problem. We build a directed graph $G = (V, E)$, where $V = \{c_1, c_2, \cdots, c_n\}$ represents all currencies, and E contains all pairs (c_i, c_j), $1 \leq i \neq j \leq n$. We assign length for edge $(c_i, c_j) \in E$ as $-\log(r_{i,j})$. We now show that computing the shortest path from s to t in G actually gives the optimal sequence of currency exchanges for converting s into t. In fact, there is one-to-one correspondence between a path from s to t in G and a sequence of currency exchange for converting s into t. Moreover, the length of such a path p equals to $\sum_{(c_i,c_j)\in p} -\log r_{i,j} = -\log \prod_{(c_i,c_j)\in p} r_{i,j}$. Hence, the shortest path in G gives the path maximizes $\prod_{(c_i,c_j)\in p} r_{i,j}$, which is exactly the maximized amount of currency t that can be coverted from a unit of currency s.

Based on the above analysis, the algorithm will be to compute the shorest path from s to t in G (using any of the single-source shortest path algorithm we introduced). The vertices along this optimal path gives the sequence of currencies following which we can get the maximized amount of currency t.

(b). The exchange rates are updated frequently. Occasionally the exchange rates satisfy the following property: there is a sequence of currencies $c_1, c_2, \cdots, c_k$ such that $r_{1,2} \cdot r_{2,3} \cdot r_{3,4} \cdots r_{k-1,k} \cdot r_{k,1} > 1$. This means that by starting with a unit of currency c_1 and then successively converting it to currencies c_2, c_3, $\cdots$, c_k, and finally back to c_1, you would end up with more than one unit of currency c_1. Give an efficient algorithm for detecting the presence of such an anomaly. Use the graph representation you found above.

Solution. We need to detect whether there is $r_{1,2} \cdot r_{2,3} \cdot r_{3,4} \cdots r_{k-1,k} \cdot r_{k,1} > 1$, which is equivalent to detect $-(\log(r_{1,2}) + \log(r_{2,3}) + \log(r_{3,4}) + \cdots + \log(r_{k-1,k}) + \log(r_{k,1})) < 0$. Since we assign length for edge $e = (c_i, c_j)$ as $-\log(r_{i,j})$, this exactly implies a negative cycle in G. In other words, such an anomaly exists if and only if G contains negative cycles. Therefore, we can use Bellman-Ford algorithm on G to identify whether G contains negative cycles, which also answers whether there exists anamaly.