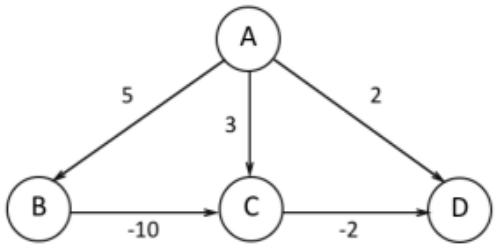
CSC4120 Spring 2024 - Written Homework 7

Yohandi 120040025
Andrew Nathanael 120040007

April 3, 2024

Problem 1. Dijkstra with negative weights



Consider the following directed graph with negative edges: $V = \{A, B, C, D\}$, $w_{AB} = 5$, $w_{AC} = 3$, $w_{AD} = 2$, $w_{BC} = -10$, $w_{CD} = -2$, starting node = A. Show the resulting iteration table (one row per iteration $0, 1, 2, \ldots$), with the value of the shortest distance for each node and the set of nodes in Q. What is the problem and why it happens?

Iteration	Dist(A)	Dist(B)	Dist(C)	Dist(D)	Nodes in Q
0	0	∞	∞	∞	$\{A\}$
1	0	5	3	2	$\{B, C, D\}$
2	0	5	3	1	$\{B, C\}$
3	0	5	3	1	$\{B\}$
4	0	5	-5	1	$\{\}$

The value of the shortest distance from A to D in the graph should be -7, passing through B and C; however, the algorithm returns 1. The issue is due to Dijkstra's algorithm that assumes all edges have positive weights; hence, node re-visitation is unnecessary as the shortest path to that node is assumed to be finalized (this is correct when all edges have positive weights). However, the negative weights introduce the possibility of finding a shorter path to already finalized nodes, leading to incorrect shortest path calculations.

Problem 2.

In the below modified Dijkstra's algorithm each vertex v will be pushed back to the priority queue Q each time it is successfully relaxed, i.e., $v.d > u.d + w(u, v)$, unless we are certain that it was there already, i.e., $v.d = \infty$. This guarantees that each time a vertex v is found to have a shorter path than before, the algorithm will re-examine the shortest paths through its neighbours (this will happen when v will be eventually enter S and this can happen even if it was in S previously). One can check that this algorithm can solve the single-source shortest path in a graph with negative weighted edges as long as there are no negative weight cycle.

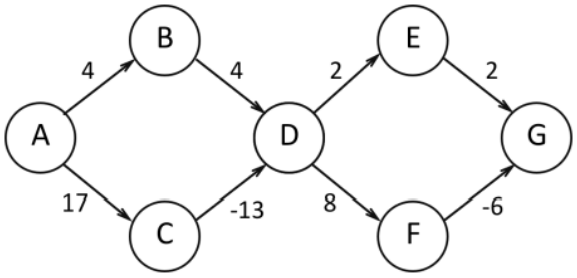
Use the modified Dijkstra's algorithm to answer the following questions: In this scenario, traversal order refers to the order of nodes extracted by EXTRACT-MIN(Q).

```
DIJKSTRA (G,w,s)

1    INITIALIZE-SINGLE-SOURCE (G,s)
2    S = ∅
3    Q = G.V
4    while Q ≠ ∅
5        u = EXTRACT-MIN (Q)
6        S = S ∪ {u}
7        for each vertex v ∈ G.Adj[u]
8            RELAX (u,v,w)
```

```
MODIFIED_DIJKSTRA (G,w,s)

1    INITIALIZE-SINGLE-SOURCE (G,s)
2    S = ∅
3    Q = G.V
4    while Q ≠ ∅
5        u = EXTRACT-MIN (Q)
6        S = S ∪ {u}
7        for each vertex v ∈ G.Adj[u]
8            MODIFIED_RELAX (Q,u,v,w)
```

```
RELAX (u,v,w)

1    if v.d > u.d + w(u,v)
2        v.d = u.d + w(u,v)
3        v.π = u
```

```
MODIFIED_RELAX (Q,u,v,w)

1    if v.d > u.d + w(u,v)
2        if v.d ≠ ∞ :
3            push(Q,v)
4        v.d = u.d + w(u,v)
5        v.π = u
```

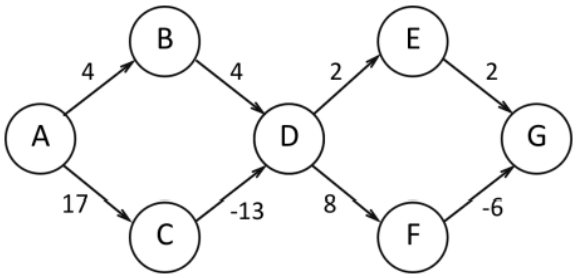


(a) List the traversal order of the first seven vertices, starting from A.
(b) List the next six.

Iteration	Dist(A)	Dist(B)	Dist(C)	Dist(D)	Dist(E)	Dist(F)	Dist(G)
at A	0	∞	∞	∞	∞	∞	∞
at B	0	4	17	∞	∞	∞	∞
at D	0	4	17	8	∞	∞	∞
at E	0	4	17	8	10	16	∞
at G	0	4	17	8	10	16	12
at F	0	4	17	8	10	16	12
at G	0	4	17	8	10	16	10
at C	0	4	17	8	10	16	10
at D	0	4	17	4	10	16	10
at E	0	4	17	4	6	12	10
at G	0	4	17	4	6	12	8
at F	0	4	17	4	6	12	8
at G	0	4	17	4	6	12	6

(a) A B D E G F G
(b) C D E G F G

Problem 3.

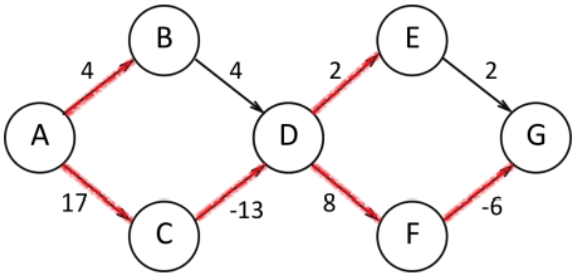


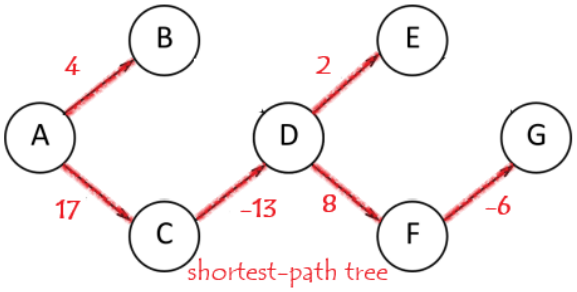
Use Bellman-Ford algorithm to answer the following for the above graph.

Show the iteration table and resulting shortest-path tree, assuming the order of edges relaxed at each iteration is A-B, B-D, A-C, C-D, D-E, E-G, D-F, F-G.

Iteration	Dist(A)	Dist(B)	Dist(C)	Dist(D)	Dist(E)	Dist(F)	Dist(G)
0	0	∞	∞	∞	∞	∞	∞
1	0	4	17	∞	∞	∞	∞
2	0	4	17	4	∞	∞	∞
3	0	4	17	4	6	12	∞
4	0	4	17	4	6	12	6
5	0	4	17	4	6	12	6

Predecessor array: [NIL, A, A, C, D, D, F]





Problem 4.

Consider a graph G in which its vertices V represent cities, and the edges E represent roads that connect cities. A delivery company is planning a route from a particular city u to another city v. This company knows the probability $p(e) \in [0, 1]$ that a given road $e \in E$ is closed (blocked). A route connecting two cities is a sequence of roads and it is closed if any of its roads is closed. Alternatively, a route is open if all its roads are open.

(a) Use Dijkstra's algorithm to find a route with minimum probability of being closed.

Assume that road closures are independent.

(b) Make appropriate changes to Dijkstra's algorithm to compute paths with the largest product of edge weights and use this algorithm to solve the problem.

Readers are assumed to have the knowledge of Dijkstra's algorithm.

(a) Consider any route r in $\mathcal{R}(u, v)$ as a sequence of roads $e_1, e_2, \ldots, e_k$ such that $e_i \in E, \forall 1 \leq i \leq k$ and $e_1 = (u, p_{12}), e_2 = (p_{12}, p_{23}), \ldots, e_k = (p_{(k-1)k}, v)$. Then, we aim to find the following:

$$\begin{aligned}
\arg \max_{r \in \mathcal{R}(u,v)} \prod_{i=1}^{k} (1 - p(e_i)) &= \arg \max_{r \in \mathcal{R}(u,v)} \ln \left(\prod_{i=1}^{k} (1 - p(e_i)) \right) \\
&= \arg \max_{r \in \mathcal{R}(u,v)} \sum_{i=1}^{k} \ln(1 - p(e_i)) \\
&= \arg \min_{r \in \mathcal{R}(u,v)} \sum_{i=1}^{k} \underbrace{-\ln(1 - p(e_i))}_{w(e_i)}
\end{aligned}$$

. Note that $1 - p(e_i), \forall i \in \{1, \ldots, k\}$ denotes the road e_i being "opened"; we aim to maximize the probability of any route r being "opened" (equivalent to our task). After the transformation, the Dijkstra's algorithm can now be applied to the task as the form is already in:

$$\arg \min_{r \in \mathcal{R}(u,v)} \sum_{i=1}^{k} w(e_i)$$

. We run Dijkstra on the weight-modified graph with source u and sink v. Note that $w(e_i) = -\ln(1 - p(e_i)) \geq 0$ as $\ln(1 - p(e_i)) \leq 0$ for $p(e_i) \in [0, 1]$; hence, the algorithm runs correctly.

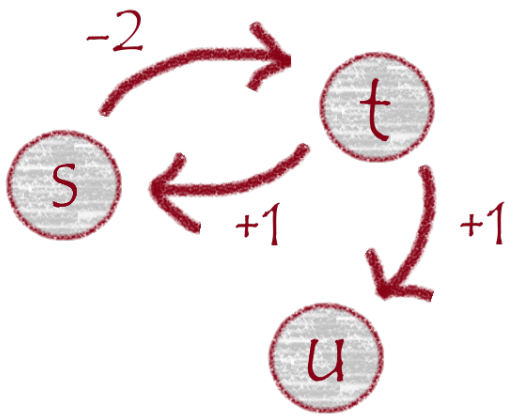
(b) The following changes can be made to Dijkstra's algorithm to compute the largest product of edge weights.

```
function Relax(u, v, w):
    if v.d > u.d + w(u, v):
        v.d = u.d + w(u, v)
        v.p = u
```

```
function Product-Relax(u, v, w):
    if v.d > u.d * w(u, v):
        v.d = u.d * w(u, v)
        v.p = u
```

Problem 5.

Consider a directed graph in which the only edges with negative weights are those that leave the initial node s and all other edges are positive. Can Dijkstra's algorithm, started at s, fail on such a graph? Prove your answer.



The above graph contains a negative weight cycle; hence, Dijkstra's algorithm may fail as it never revisits nodes to update their distances based on the newly encountered negative path.

Problem 6.

You are given a directed graph with (possibly negative) weighted edges, in which the shortest path between any two vertices is guaranteed to have at most k edges. Give an algorithm that finds the shortest path between two vertices u and v in $O(k \cdot |E|)$ time.

Since the term "shortest path exists within k edges" is used, we will assume there is no negative weight cycle.

```
function Modified-Bellman-Ford(graph, u, v, k):
    dist = array of size |V| filled with infinity
    dist[u] = 0

    for i from 1 to k:
        tmp = dist
        for each edge (x, y) in graph:
            if dist[x] + weight(x, y) < tmp[y]:
                tmp[y] = dist[x] + weight(x, y)
        dist = tmp

    return dist[v] # if dist[v] is infinity, then there is no path exists
   within k edges
```

The above is almost exactly the same as the Bellman-Ford algorithm; however, instead of looping until $|V|$ (the number of vertices), we stop until k. This works as it iteratively relaxes all the edges in the graph up to k times; consequently, each relaxation step can only use paths with one additional edge compared to the previous step, which methodically builds up the shortest path from the starting vertex u to every other vertex with at most k edges.

Problem 7.

We consider the case of speeding up the search for the shortest path to destination t using modified edge weight based on the potential function method. Prove that the potential computed using landmarks is always consistent (same as 'admissible', 'feasible'). In particular, prove that property for the case of using a single landmark and then for the case of using multiple landmarks.

Consider a graph $G = (V, E)$ and a single landmark l. The potential function for a vertex u is defined based on the triangle inequality and shortest distances to and from the landmark. The potential, $\pi(u)$, defined as $\pi(u) = \mathrm{d}(u, l) - \mathrm{d}(t, l)$, where $\mathrm{d}(x, y)$ denotes the shortest path distance between vertices x and y, and t is the target vertex.

For the heuristic to be consistent, every edge $(u, v) \in E$ must satisfy:

$$\begin{aligned} & \mathrm{d}(u, v) + \pi(v) - \pi(u) \geq 0 \\ \Rightarrow\ & \mathrm{d}(u, v) + (\mathrm{d}(v, l) - \mathrm{d}(t, l)) - (\mathrm{d}(u, l) - \mathrm{d}(t, l)) \geq 0 \\ \Rightarrow\ & \mathrm{d}(u, v) + \mathrm{d}(v, l) \geq \mathrm{d}(u, l) \end{aligned}$$

, confirming that the potential is consistent (due to triangle inequality).

Denote L as the set of landmarks. When multiple landmarks are used, the potential for a vertex u can be taken as the maximum of the differences between the distances from u to all landmarks and from the target t to all landmarks:

$$\pi(u) = \max_{l \in L} (\mathrm{d}(u, l) - \mathrm{d}(t, l))$$

For the heuristic to be consistent, every edge $(u, v) \in E$ must satisfy:

$$\mathrm{d}(u, v) + \pi(v) - \pi(u) \geq 0$$

Since the triangle inequality holds for each landmark individually and the maximum of a set of consistent estimations remains consistent (because each estimation respects the triangle inequality), the potential computed using multiple landmarks is also consistent.

Problem 8.

You are given a directed graph $G = (V, E)$ with (possibly negative) weighted edges, along with a specific node $s \in V$ and a tree $T = (V, E')$, $E' \subseteq E$. Give an algorithm that checks whether T is a shortest-path tree for G with starting point s. Your algorithm should run in linear time.

Let $d_T(u, v)$ be the distance between u and v along the edges of T. Then, $d_T(u, v)$ can be computed in linear time using DFS starting from s. In the search, we store and maintain the distances s to all other nodes v in T, in $\mathcal{O}(|V| + |E'|) = O(|V|)$ (as $|E'| = |V| - 1$).

For each edge $(u, v) \in E - E'$ (not in the tree), we check if $w(u, v) + d_T(s, u) \geq d_T(s, v)$, to make sure that no edge outside T returns a shorter path from s to any v. If such a path exists, then the current tree is not a shortest-path tree (as it contradicts the definition of shortest-path tree); otherwise, T correctly represents all shortest paths from s. This runs in $\mathcal{O}(|V| + |E'|) = O(|V|)$.

Formal proof

To prove that the test will succeed for all $e \in E - E'$ if and only if T is a correct shortest-path tree from s, let's consider the properties of shortest-path trees and the behavior of the Bellman-Ford algorithm. A shortest-path tree T rooted at s correctly represents the shortest paths from s to all other vertices in V if for any path from s to v, the path length in T is equal to the shortest path length in G. The distance $d_T(s, v)$ represents the length of the shortest path from s to v within the tree T.

For T to be a shortest-path tree, the following must hold:

- For all edges $(u, v) \in E'$, the distances satisfy the property of being part of the shortest path, i.e., $d_T(s, v) = d_T(s, u) + w(u, v)$.
- For all edges $(u, v) \in E - E'$, adding the edge to T should not provide a shorter path from s to v, i.e., $d_T(s, v) \leq d_T(s, u) + w(u, v)$.

($\Leftarrow$) Assume T is a correct shortest-path tree. For any edge $(u, v) \in E - E'$, if adding this edge would provide a shorter path from s to v, then T would not have been a shortest-path tree, to begin with, contradicting our assumption. Thus, $d_T(s, v) \leq d_T(s, u) + w(u, v)$ must hold for T to represent all shortest paths correctly.

($\Rightarrow$) Suppose for every edge $(u, v) \in E - E'$, the inequality $d_T(s, v) \leq d_T(s, u) + w(u, v)$ holds. This implies there's no edge outside T that could create a shorter path from s to any vertex v, satisfying the shortest-path criterion for all vertices. Thus, T must be a shortest-path tree.

The Bellman-Ford algorithm iteratively relaxes edges, reducing the estimated distances to each vertex until no more updates are possible, so that the discovery of shortest paths in a graph with potentially negative weights (assuming no negative cycles). Now, if we consider the first update (u, v) that causes the distance of v to drop below $d_T(s, v)$:

- Before the update, $d_T(s, v)$ represents the shortest distance from s to v in T.
- The update implies there exists a shorter path from s to v via u with length $d_T(s, u) + w(u, v) < d_T(s, v)$.

This edge (u, v) directly contradicts the requirement for T being a shortest-path tree, as it shows a path outside of T providing a shorter route to v. Therefore, this edge would fail the test $d_T(s, v) \leq d_T(s, u) + w(u, v)$, indicating T cannot be a shortest-path tree if any such edge exists. Hence, the test $d_T(s, v) \leq d_T(s, u) + w(u, v)$ for all $e \in E - E'$ is both necessary and sufficient to verify if T is a correct shortest-path tree from s. The first update in a Bellman-Ford run that violates this condition proves that T does not contain all shortest paths from s, thereby failing to be a shortest-path tree.