Q1. Given an undirected graph G, describe a linear time algorithm to find the number of distinct shortest paths between two given vertices u and v. Note that two shortest paths are distinct if they have at least one edge that is different.

We perform a BFS on the graph starting from u, and create a variable $\texttt{num\_paths}(x)$ for the number of paths from u to x, for all vertices x. If $x_1, x_2, \ldots x_k$ are vertices at depth l in the BFS tree and x is a vertex at depth $l+1$ such that $(x_1, x), \ldots, (x_k, x) \in E$ then we want to set $\texttt{num\_paths}(x) = \texttt{num\_paths}(x_1) + \ldots + \texttt{num\_paths}(x_k)$. The easiest way to do this is to start with $\texttt{num}_\texttt{p}\texttt{aths}(x) = 0$ for all vertices $x \neq u$ and $\texttt{num\_paths}(u) = 1$. We then update $\texttt{num\_paths}(y) = \texttt{num\_paths}(y) + \texttt{num\_paths}(x)$, for each edge (x, y) that goes down one level in the tree. Since, we only modify BFS to do one extra operation per edge, this takes linear time. The pseudocode is as follows

```
function count_paths(G, u, v)
    for all x ∈ V:
        dist(x) = ∞
        num_paths(x) = 0

    dist(u) = 0
    num_paths(u) = 1
    Q = [u]
    while Q is not empty:
        x = eject(Q)
        for all edges (x, y) ∈ E
            if dist(y) = dist(x) + 1:

                num_paths(y) = num_paths(y) + num_paths(x)
            if dist(y) = ∞:
                inject(Q,y)
                dist(y) = dist(x) + 1
                num_paths(y) = num_paths(x)
```

Q2. Given a weighted directed graph with positive weights, given a $O(|V|^3)$ algorithm to find the length of the shortest cycle or report that the graph is acyclic.

Define matrix D so that D_{ij} is the length of the shortest path from vertex i to vertex j in the input graph. Row i of the matrix can be computed by a run of Dijkstra's algorithm in time $O(|V|^2)$. So we can calculate all of D in time $O(|V|^3)$. For any pair of vertices u, v we know that there is a cycle of length $D_{uv} + D_{vu}$ consisting of the two shortest paths between u and v and that this cycle is the shortest among cycles containing u and v. This shows that it suffices to compute the minimum $D_{uv} + D_{vu}$ over all pairs of vertices u, v to find the length of the shortest cycle. This last operation takes time $O(|V|^2)$, so the overall running time is $O(|V|^3)$.

Note that for the Dijkstra's algorithm, we make use of the array implementation of the priority queue, since that gives worst case time $O(|V|^2)$.

Q3. Given a directed weighted graph G, with positive weights on the edges, let us also add positive weights on the nodes. Let l(x,y) denote the weight of an edge (x,y), and let w(x) denote the weight of a vertex x. Define the cost of a path as the sum of the weights of all the edges and vertices on that path. Give an efficient algorithm to find all the smallest cost path (as defined above) from a source vertex to all other vertices. Analyze and report the running time of your algorithm.

There are two approaches: one is a *reduction*; the other is a direct modification of Dijkstra's algorithm.

METHOD I: The idea is to use a *reduction*: on input (G, l, c, s), we construct a graph $G' = (V', E')$ where G only has edge weights (no node weights), so that the shortest path from s to t in G is essentially the same as that in G', with some minor modifications. We can then compute shortest paths in G' using Dijkstra's algorithm.

The reduction works by taking every vertex v of G and splitting it into two vertices v_i and v_o. All edges coming into v now come into v_i, while all edges going out of v now go out of v_o. Finally, we add an edge from v_i to v_o of weight $c(v)$.

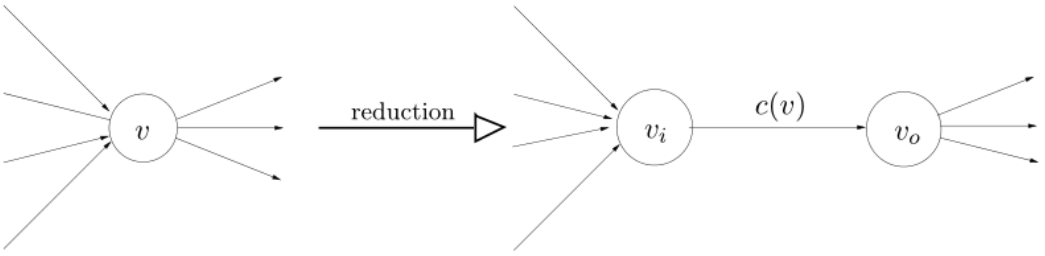


Figure 5: Reduction in 4.19.

Consider now any path in G and notice that it can be converted to an edge-weighted path of the same weight in G' by replacing the visit to vertex v with the traversal of edge (v_o, v_i). Conversely,

consider a path in G': every other edge visited is of the form (v_i, v_o) and corresponds to a vertex v of G. Replacing these edges with the corresponding vertices we obtain a path in G of the same weight as the path in G'. The time required to perform this reduction is $O(|V| + |E|)$. G' has $|V| + |E|$ edges and $2|V|$ vertices, so running Dijkstra takes time $O(|V|^2)$ and the total running time is $O(|V|^2)$.

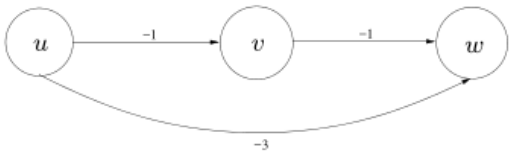
METHOD II: We make the following modifications to Dijkstra's algorithm to take into account node weights:

- In the initialization phase, $\texttt{dist(s)} = \texttt{w(s)}$.
- In the update phase, we use $\texttt{dist(u)} + \texttt{l(u, v)} + \texttt{w(v)}$ instead of $\texttt{dist(u)} + \texttt{l(u, v)}$.

Analysis of correctness and running time are exactly the same as in Dijkstra's algorithm.

Q4. Given a directed graph G with possibly negative edge weights. Consider the following algorithm to find the shortest paths from a source vertex to all other vertices. Pick some large constant c, and add it to the weight of each edge, so that there are no negative weights. Now, just run Dijkstra's algorithm to find all the shortest paths. Is this method correct? If yes, reason why. If not, give a counterexample.

The weighted graph in Figure 4 is a counterexample: According to the algorithm proposed



we can add c=4 to all the edges, so uv is 3, vw is 3 and uw=1. Now the shortest path from u-w is 1, but in the original graph it was u-v-w with a cost of -2.