Assignment 4

CS 514 – Algorithms

Submitted By: Aman Pandita

Onid: panditaa@oregonstate.edu

2. (10 points) Suppose that the only negative edges are those that leave the starting node s. Does Dijkstra's algorithm find the shortest path from s to every other node in this case? Justify your answer carefully.

Dijkstra's algorithm works based on the idea that once a node is visited, you won't find a shorter path to it. This works because the algorithm assumes that all the edges have non-negative weights.

From a queue of vertices sorted by their distances from the starting point, it constantly picks the vertex with the shortest distance.
Even if some edges departing from the starting node have negative weights, Dijkstra's method can surprisingly still find the shortest routes to all other nodes in a network starting from "s".

Justification of the answer above:

Initialization:

The initialization of Dijkstra's algorithm sets the distance to start vertex ***'s'*** as 0 and to all other vertices as infinity. The lengths between the vertices that they leave behind will update appropriately since the edges that leave ***'s'*** have negative weights.

Priority Queue:

Priority will be given to the start vertex s that are connected via negative edges. This is because, the priority queue always returns the vertex with the least tentative distance.

Path Relaxation:

The algorithm will not discover shorter paths to vertices directly connected to s after initializing with negative weights for those vertices because all other edges have non-negative weights.

Propagation of Negative Weights:

The negativity does not spread throughout the graph because negative weights are only connected to edges that depart from the start vertex s. The algorithm operates in a graph with non-negative edges once it has moved past s's immediate neighbors, in line with Dijkstra's hypothesis.

Optimality:

The negative weights are advantageous to the immediate neighbors of the initial vertex's'. In order to guarantee that the shortest paths discovered are ideal, the algorithm determines the shortest paths for vertices that are farther away from their neighbors using the non-negative edges and the initially determined negative weights.

Conclusion:

In conclusion, ***"YES",*** Dijkstra's algorithm is still able to find the shortest paths from the start vertex, 's', to all other vertices in this specific scenario even though there are negative edges leaving the starting vertex. This is because the negativity doesn't spread throughout the graph, and the algorithm operates consistently after handling "s" immediate neighbors.

3. (20 points) Give an $O\left(n^3\right)$ algorithm that takes a directed graph as input and returns the length of the shortest cycle in the graph where n is the number of nodes.

```python
def shortest_cycle(graph):
    n = len(graph)
    min_cycle_length = float('inf')

    # Run Floyd-Warshall to find shortest paths between all pairs of vertices
    dist = [[float('inf')] * n for _ in range(n)]
    for vertex in graph:
        for neighbor in graph[vertex]:
            dist[vertex][neighbor] = 1  # or the weight of the edge if the graph is weighted

    for k in range(n):
        for i in range(n):
            for j in range(n):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

    # Check for cycles after all shortest paths are found
    for i in range(n):
        if dist[i][i] < min_cycle_length:
            min_cycle_length = dist[i][i]

    return -1 if min_cycle_length == float('inf') else min_cycle_length
```

Finding the length of the shortest cycle in a ***directed graph*** *(because of $O(n^3)$ time complexity)* is the goal of the algorithm, which is a modification of the ***"Floyd-Warshall algorithm"***. My version of the Floyd-Warshall algorithm does more than just finding the shortest paths between all pairs of points.
It checks for the shortest cycle as well, which is not available in the regular version. To check for cycles, I check if the diagonal of the distance matrix ***(dist[i][i])*** is smaller than infinity, assuming we're working with non-negative weights. This algorithm runs in ***O(n^3)*** time.
A brief explanation of the algorithm is below:

1. Initialize a distance matrix ***"dist"*** to store the shortest distances between all pairs of vertices. The distance between a vertex and itself is 0, and the distance between two vertices that are not directly connected is infinity.
2. The algorithm uses a triple nested loop to iteratively update the distance matrix. For each pair of vertices ***(i, j)***, it checks whether a shorter path exists through an intermediate vertex ***k***, updating the distances to ensure the shortest paths are stored.
3. Instead of looking for negative cycles ***(dist[i][i] < 0)***, the algorithm specifically looks for any cycles by checking the diagonal of the distance matrix ***(dist[i][i])***. If a vertex has a distance less than infinity to itself, a cycle has been detected.
4. The algorithm keeps track of the shortest cycle length found during the process, and this length is returned as the output. If no cycle is found, it returns ***-1***.

4. (10 points) You are given a strongly connected directed graph G = (V, E) with positive edge weights. Give an efficient algorithm for finding the shortest paths between all pairs of nodes with the restriction that they all must pass through node A.

We can use the below idea for the requirement above:

1. Running Dijkstra's algorithm from node A to find the shortest path distances from A to all other nodes in ***G***. Store these distances in an array ***B***.
2. Reverse the edges of ***G*** to create a new graph ***G_R***.
3. Run Dijkstra's algorithm from node ***A*** in ***G_R*** to find the shortest path distances from all other nodes to ***A***. Store these distances in an array ***A***.
4. For each pair of nodes ***s*** and ***t***, the shortest path from ***s*** to ***t*** that passes through ***A*** has length ***A[s] + B[t]***.

This algorithm takes ***O (|V| + |E|) log |V|*** time.

Note:

Assuming that the lengths of the shortest path are not stored directly which could lead to $O(n^2)$ time complexity.

References:

1. Stack Overflow, https://stackoverflow.com
2. Geek for Geeks, https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/#