Problem Set 7

Due dates: Electronic submission of the pdf file of this homework is due on **3/22/2024 before 11:59pm** on canvas. The homework must be typeset with LaTeX to receive any credit. All answers must be formulated in your own words.

Watch out for additional material that will appear on Thursday! Deadline is on Friday, as usual.
Name: Jawahar Sai Nathani

Resources. (All people, books, articles, web pages, etc. that have been consulted when producing your answers to this homework)

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: Jawahar Sai Nathani

Read the chapters on "Elementary Graph Algorithms" and "Single-Source Shortest Paths" in our textbook before attempting to answer these questions.

Problem 1 (20 points)**.** Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(V)$ time, independent of the number $|E|$ of edges.

Solution.

- Begin by marking all vertices as unvisited. Proceed to iterate through all vertices, conducting a Depth-First Search (DFS) traversal for any unvisited vertices encountered.
- In case a visited node is encountered that is not the parent, indicating a cycle, appropriate action can be taken. Utilizing the parent information allows for the identification of back edges.
- The 'cycleExists' function iterates through all V nodes at worst, independent of the number of edges, as DFS explores each path at most once. Consequently, its time complexity is bound by the number of vertices, resulting in a time complexity of $O(V)$.

Algorithm 1 Detect Cycle in Graph

```
function cycleExists($graph, n$)
    $visited \leftarrow$ array of size $n$ initialized with False
    for $i \leftarrow 0$ to $n - 1$ do
        if $\neg visited[i]$ then
            if DFS($graph, i, visited, -1$) then
                return True
            end if
        end if
    end for
    return False
end function

function DFS($graph, u, visited, parent$)
    $visited[u] \leftarrow$ True
    for $v$ in $graph[u]$ do
        if $\neg visited[v]$ then
            if DFS($graph, v, visited, u$) then
                return True
            end if
        else if $v \neq parent$ then
            return True
        end if
    end for
    return False
end function

cycleExists($graph, n$)
```

Problem 2 (20 points)**.** Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

Solution. We can incorporate this optimization of the Bellman-Ford algorithm by keeping track of whether vertex v has been relaxed or not. If v has been relaxed, we pause to observe whether v has been updated (indicating it has been relaxed again). If v has not been updated, we halt the process.

Additional explanation:

Given that the maximum number of edges on any shortest path from the source is m, the path-relaxation property assures us that after m iterations of the Bellman-Ford algorithm, every vertex v will have reached its shortest-path weight in v.d. As per the upper-bound property, after m iterations, no d values will undergo further alteration. Consequently, no d values will change in the $(m + 1)$st iteration. Since we don't have foreknowledge of m, we can't precisely iterate the algorithm m times and then halt it. However, by allowing the algorithm to terminate when no further changes occur, it will stop after $m + 1$ iterations.

Problem 3 (20 points)**.** Suppose that we change line 6 of Dijkstra's algorithm in our textbook to the following.

6 **while** $|Q| > 1$

This change causes the while loop to execute $|V| - 1$ times instead of $|V|$ times. Is this proposed algorithm correct? Explain. [Use the version of Dijkstra's algorithm from the textbook]

Solution. No, Modifying the while loop condition from "while($Q \neq \varnothing$)" to "while($|Q| > 1$)" does not preserve Dijkstra's algorithm's key property, which ensures that the vertex extracted from the priority queue always has the guaranteed shortest distance from the source so far.

By changing the condition to "$|Q| > 1$", the algorithm would terminate when there is only one vertex left in the priority queue. This adjustment could potentially result in missing the shortest paths from the source to other vertices.

The original while loop condition in Dijkstra's algorithm ensures that the algorithm continues until the priority queue is empty, guaranteeing that all vertices are processed and their shortest distances updated accordingly:

Consider an example where the shortest path to all vertices passes through the last vertex in the queue. If we use the updated condition, we might miss updating these shortest distances because the algorithm would terminate prematurely.

Therefore, it is essential to maintain the original while loop condition "while($Q \neq \varnothing$)" to ensure the correctness and effectiveness of Dijkstra's algorithm.

Problem 4 (40 points)**.** Help Professor Charlie Eppes find the most likely escape routes of thieves that robbed a bookstore on Texas Avenue in College Station. The map will be published on Thursday evening. In preparation, you might want to implement Dijkstra's single-source shortest path algorithm, so that you can join the manhunt on Thursday evening. Include your implementation of Dijkstra's algorithm and explain all details of your choice of the min-priority queue.

[Edge weight 1 means very desirable street, weight 2 means less desirable street]

Solution. There are a total of 22 waypoints. The robbers were last spotted at waypoint 1. It's more likely that the robbers will pass through waypoints 6, 8, 9, 15, 16, and 22. Hence, in the context of Dijkstra's algorithm, the starting node is 1, and our objective is to determine the shortest distances from node 1 to the previously mentioned 6 waypoints. I am using Heap as a priority queue to extract the minimum distance vertex.

Algorithm 2 Dijkstra's Algorithm

```
function DIJKSTRA($graph, source$)
    $n \leftarrow$ length of $graph$
    $queue \leftarrow [(0, source)]$
    $parent \leftarrow [-1] * n$
    $dist \leftarrow [\infty] * n$
    $dist[source] \leftarrow 0$
    while $queue$ is not empty do
        $d, u \leftarrow heapq.heappop(queue)$
        if $d > dist[u]$ then
            continue
        end if
        for $v$ in range($n$) do
            if $graph[u][v] \neq 0$ then
                $alt \leftarrow dist[u] + graph[u][v]$
                if $alt < dist[v]$ then
                    $dist[v] \leftarrow alt$
                    $parent[v] \leftarrow u$
                    $heapq.heappush(queue, (alt, v))$
                end if
            end if
        end for
    end while
    return $dist, parent$
end function
```

In the preceding algorithm, I utilized an adjacency matrix to depict the graph. I'll opt for an alternative notation to illustrate the graph: [u, v, (weight)].

$$
\begin{gathered}
[1, 2, (1)], [1, 11, (1)] \\
[2, 3, (1)], [2, 21, (1)] \\
[3, 4, (1)], [3, 8, (2)] \\
[4, 5, (1)] \\
[5, 6, (1)], [5, 7, (1)] \\
[6, 7, (1)] \\
[7, 8, (1)] \\
[8, 9, (1)] \\
[9, 10, (1)], [9, 19, (1)] \\
[10, 11, (1)], [10, 18, (2)] \\
[11, 12, (2)], [11, 17, (1)] \\
[12, 13, (1)] \\
[13, 14, (2)], [13, 21, (1)] \\
[14, 15, (1)], [14, 16, (1)], [14, 20, (1)] \\
[16, 17, (2)] \\
[17, 18, (2)] \\
[18, 19, (2)] \\
[20, 21, (2)], [20, 22, (1)] \\
[21, 22, (2)]
\end{gathered}
$$

Node 6: Distance = 6, path = (1 - 2 - 3 - 4 - 5 - 6)
Node 8: Distance = 4, path = (1 - 2 - 3 - 8)
Node 9: Distance = 3, path = (1 - 11 - 10 - 9)
Node 15: Distance = 6, path = (1 - 11 - 17 - 16 - 14 - 15)
Node 16: Distance = 4, path = (1 - 11 - 17 - 16)
Node 22: Distance = 4, path = (1 - 2 - 21 - 22)

Since the distance to Node 9 is the lowest, robbers will most likely choose it.

Discussions on canvas are always encouraged, especially to clarify concepts that were introduced in the lecture. However, discussions of homework problems on ecampus should not contain spoilers. It is okay to ask for clarifications concerning homework questions if needed. Make sure that you write the solutions in your own words.

Checklist:
✓ Did you add your name?
✓ Did you disclose all resources that you have used?
(This includes all people, books, websites, etc. that you have consulted)
✓ Did you sign that you followed the Aggie honor code?
✓ Did you solve all problems?
✓ Did you submit the pdf file of your homework?