Algorithm Design and Analysis (Fall 2023)

Assignment 2

Deadline: Nov 27, 2023

Yuqing Tu 522030910152

1. (25 points) Design a polynomial time algorithm that, given a directed unweighted graph $G = (V, E)$ and $s \in V$, outputs "yes" if there is a vertex $u \in V$ such that there are at least two different simple paths from s to u and outputs "no" otherwise. A simple path is a path that does not visit a vertex more than once. Two paths are different if they differ in at least one edge. Prove the correctness of your algorithm and analyze its time complexity.

   Solution:

Algorithm 1 *are_two_paths*(s, u)

```
Use DFS to find a path from s to u.
if don't find a path then
    return no
else
    Record this path as p_1.
    Use another way to run DFS to find a path from s to u ("another way" is explained below).
    Record this path as p_2.
    if p_1 == p_2 then
        return no
    else
        return yes
    end if
end if
```

"Another way":
Take the adjacency matrix adj as an example. When we run DFS, if we have visited the i_{th} node v_i, we will push every v_i's neighbors into stack from $adj[i][1]$ to $adj[i][n]$. And "another way" is that after we have visited the $i-th$ node v_i, we will push every v_i's neighbors into stack from $adj[i][n]$ to $adj[i][1]$, by reverse order.

Correctness proof:

1) If there is not a path from s to u, then this algorithm will output "no". In this case, the algorithm is correct.
2) If there is only one path from s to u, then p_1 must be as same as p_2, then this algorithm will output "no". In this case, the algorithm is correct.
3) If there are two different paths, $path_1$ and $path_2$, from s to u, suppose that $path_1$ is $s \to v_1 \to v_2 \to \cdots \to v_i \to x_1 \to \cdots \to u$ and $path_2$ is $s \to v_1 \to v_2 \to \cdots \to$

$v_i \to x_2 \to \cdots \to u$ (x_1 and x_2 are the first vertex of difference between the two paths). And suppose that x_1 is in $adj[i][j]$, x_2 is in $adj[i][k]$ and $j > k$, then when we run the first DFS, x_1 will come out of the stack before x_2, so $path_1$ will be find, $p_1 == path_1$. In a similar way, when we run the second DFS, $path_2$ will be find, $p_2 == path_2$. So p_1 is not as same as p_2, this algorithm will output "yes". In this case, the algorithm is correct.

4) If there are more than two different paths, the proof is similar to 3)'s proof.

Time complexity:
This algorithm is running twice DFS, so its time complexity is $O(|V| + |E|)$ if we use adjacency list to store the graph and is $O(|V|^2)$ if we use adjacency matrix to store the graph.

2. (25 points) Given an undirected connected graph $G = (V, E)$ and a vertex s, if the DFS tree and the BFS tree rooted at s are identical, prove that G is a tree.

Proof:

To prove this proposition, we just need to prove that if the DFS tree and the BFS tree rooted at s are identical, then G is acyclic.
Prove by contradiction. Suppose that the DFS tree and the BFS tree rooted at s are identical and G is cyclic, one of G's loop is $v_1 \to v_2 \to v_n \to v_1$.
Suppose that when we run DFS, the first vertex be found is v_i. Then v_{i-1} or v_{i+1} will be v_i's only child in this loop's vertices.
Suppose that when we run BFS, the first vertex be found is v_i. Then v_{i-1} and v_{i+1} will both be v_i's children in this loop's vertices.
So the DFS tree and the BFS tree rooted at s are not identical. There is a contradiction.
So if the DFS tree and the BFS tree rooted at s are identical, then G is acyclic. The proposition is proved.

3. (25 points) Consider a directed *acyclic* edge-weighted graph $G = (V, E, w)$ where edge weights can be negative and a vertex $s \in V$.

   (a) (20 points) Adapt the Bellman-Ford algorithm to find the distances of all vertices from s. Your algorithm must run in $O(|V| + |E|)$ time. Prove the correctness of your algorithm and analyze its time complexity.

   (b) (5 points) Suppose now we want to find the length of the longest path from s to each vertex $u \in V$ (the length is measured by the sum of the weights of the edges on the path, not by the number of the edges). Can we negate the weight of every edge and use the algorithm from the first part? If so, prove it; if not, provide a counterexample.

Solution:

(a) **Algorithm 2** $find\_distances(s, G)$

```
1: Use DFS to find the topological order of G
2: dist[s] ← 0, dist[x] ← ∞ for other vertices x in G
3: for each v, which start at s and go to the last vertex by topological order do
4:    for each (v, u) do
5:       if dist[u] > dist[v] + w(v, u) then
6:          dist[u] ← dist[v] + w(v, u)
7:       end if
8:    end for
9: end for
```

Correctness proof:

There is not any path from s to v for all v whose topological order is before s. So $dist[v] = \infty$ for these v. Then we do k rounds traversing from s to the last vertex by topological order and each round we will update distances if it is necessary. And by topological order to update distances is as same as traversing all vertices to update, because there is not a path from v to u for v's topological order is after u's.

So it satisfies Lemma 1 that we use to prove the correctness of Bellman-Ford. And there is no cycle in G, so we don't need Observation 2. Then we can prove that this algorithm is correct in a similar way to prove the correctness of Bellman-Ford.

Time complexity:

The time complexity of topological order is $O(|V| + |E|)$. Updating the distances can be divided into two parts: 1. traversing from s to the last vertex by topological order. Its time complexity is $O(|V|)$. 2. traversing at most all edges. Its time complexity is $O(|E|)$. So the total time complexity is $O(|V| + |E|) + O(|V|) + O(|E|) = O(|V| + |E|)$.

(b) Yes, we can.

Proof:

If s and u are connected, there must be a longest path from s to u and this longest path'length is finite because G is acyclic.

Prove by contradiction. Suppose that

1) the longest path from s to u is p and the length of p is l.
2) we negate the weight of every edge and this new graph is G', the shortest path from s to u is p' and the length of p' is l'.
3) p is not equal to p'.

p's length is $-l$ in G' and p''s length is $-l'$ in G because of 2). Then in G', because p' is the shortest path from s to u, so $l' < -l$. In G, $-l' > l$, which means p' is longer than p in G. So p is not the longest path in G. There is a contradiction. So p must be equal to p'. The algorithm's correctness is proved.

4. (25 points) It is possible that the shortest path from s to t in an edge-weighted graph is not unique. Given a directed edge-weighted graph $G = (V, E, w)$ with positive edge weights and $s \in V$, design an $O((|V| + |E|) \log |V|)$ time algorithm to output a Boolean array B, with vertices being the array indices, such that $B[u] =$ `true` if the shortest path from s to u is unique and $B[u] =$ `false` otherwise. You can assume that every vertex $u \in V$ is reachable from s. Prove the correctness of your algorithm.

Solution:

This algorithm is only a small modification of Dijkstra algorithm.

Algorithm 3 *is_unique*(s, G)

1: $T \leftarrow s$
2: $B[v] \leftarrow$ `true`, $map[v] \leftarrow$ `true` for all vertices v.
3: $tdist[s] \leftarrow 0$, $tdist[v] \leftarrow \infty$ for all vertices v other than s.
4: $tdist[v] \leftarrow w(s, v)$, $pre[v] \leftarrow s$ for all edges (s, v).
5: **while** $T \neq G$ **do**
6: Find $v \notin T$ with the smallest $tdist[v]$.
7: $T \leftarrow T \cup \{v\}$
8: $B[v] \leftarrow map[v]$
9: **if** $tdist[u] == tdist[v] + w(v, u)$ for some edges (v, u) **then**
10: $map[u] \leftarrow$ `false` for all these vertices u
11: **end if**
12: $tdist[u] \leftarrow \min\{tdist[u], tdist[v] + w(v, u)\}$ for all edges (v, u).
13: **if** $tdist[u]$ is update **then**
14: $pre[u] \leftarrow v$
15: $map[u] \leftarrow$ `true`
16: **end if**
17: **end while**

Correctness proof:
In one loop, the v that is newly added to T can't be as same as $pre[u]$ for all (v, u). If $tdist[u]$ is as same as $tdist[v] + w(u, v)$ for some edges (v, u), then there are at least two paths of equal length from s to u, and these paths' length is currently the shortest. In this case, $map[u]$ is `false`.
In next loop, if the v, which is newly added to T, has $map[v] =$ `false`, then this case means that there are at least two shortest paths from s to v. So $B[v]$ is `false`. And if $map[v]$ is `true`, then this case means that the shortest paths from s to v is unique. So $B[v]$ is `true`.
And the other vertices that has not been added to T are still have a chance to update,

if the vertex u is update, it is necessary to re-determine whether the shortest path from s to u is unique. So $map[u]$ will be `true`.

Time complexity:
This algorithm just adds five lines (8, 9, 10, 11, 15) on the basis of Dijkstra algorithm. This addition doubles the update time. If we use binary heap, the time complexity of update is still $O(|E| \log |V|)$. So the total time complexity is still $O((|V| + |E|) \log |V|)$.

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.

   It took me about 6 hours to finish the assignment. I spent about 2.5 hours thinking and discussing, and the rest of the time getting familiar with the use of Latex and search for relevant syntax. I'll give a 3 to the difficulty. My collaborator is Jiayang Li and his student ID is 522030910113.