Algorithm Design and Analysis (Fall 2023)

Assignment 2

Deadline: Nov 27, 2023

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.

   Algorithm:

   Function explore(s){
   marked[s] ← true (hold marked to do dfs and avoid cycle)
   num[s]++ (the time s is visited)
   if num[s] = 2 (2 means s is visited by two different simple paths)
   output yes and end the algorithm
   for each (s,v) ∈ E
   if marked[v]=false
   explore[v]
   marked[s] ← false (make sure s can be visited again by another simple path)
   }

   The overall procedure is to first create two array named marked and num. Then initialize marked with false and num with 0. Next, apply explore(s) on the given vertex s. After the whole algorithm, if there is no output, output no.

   Correctness: First, the algorithm will not follow a cycle all the time because we use an array *marked* to make sure this circumstances won't happen. Therefore, if num[v] is equal to 1, there exist at least one simple path from s to v. If num[v] = 2, there are only two circumstances: 1. there is a vertex u that points to v satisfying num[u] = 2. 2. there are two vertices a, b (a, b can be the same vertex if there are multiple edges form a, b to v) that point to v satisfying num[a] = num[b] = 1. Meanwhile, the first circumstances won't happen because the above-mentioned algorithm ends when we find the first vertex v that satisfies num[v] = 2. Then, let's consider the second situation. Let num[v] = 2 and num[a] = num[b] = 1(a, b are two vertices that point directly to v). As mentioned above, num[a] = 1 means there exists a simple path from s to a, and so does b. Since (a,v), (b,v) are two distinct edges, $s \to a \to v$ and $s \to b \to v$ are two different simple paths.

Time Complexity: In the worst case: each edge is visited exactly once and each vertex is also visited exactly once. Therefore, the overall time complexity is $O(|V| + |E|)$ according to dfs algorithm mentioned in the class.

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.

   Suppose G isn't a tree, which means there is at least one cycle in G. Then there are two circumstances: 1. s is in a cycle. 2. s is not in a cycle.
   In the first situation, s has n children in DFS tree while s has more than n children in BFS tree. Because in BFS tree, all vertices adjacent to s is its first children. While in DFS tree, let a, b be two vertices that is adjacent to s and are in the same cycle. Then if we DFS a first, a will be s's first child and b won't be s's first child. Instead, b will be a's child because b has already be visited and be marked true when we DFS a since a, b are in a cycle and we can go from a to b without passing through s. Therefore, the DFS tree and the BFS tree rooted at s are different, which contradicts to the given condition.
   In the second situation, by applying DFS and BFS algorithm on s, we will find a vertex a that first appears in a cycle and a is a child of s. Then, as proved in the first situation, the DFS tree and the BFS tree rooted at a are different because we choose the first vertex appeared in a cycle and let it be the root. Therefore, the two tree are also different, which contradicts to the given condition.

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.

   Algorithm:

   1. Find the topological order of G.
   2. Initialize an array $dist$ with $dist[s] = 0$ and $dist[u] = INF$ for u other than s.
   3. For each u in topological order of G:
      for each edge $(u, v) \in E$:
      $dist[v] = min\{dist[v], dist[u] + w(u, v)\}$

   Correctness:

   Since G is a directed acyclic graph, there is a topological order of G. Then all the vertices can be devided into there parts: 1. vertices that are in front of s in topological order. 2. s. 3. vertices that are behind s in topological order. For the first part, $dist[u]$ will remains INF because there is no path from s to u and $dist[u]$ won't change before searching s. That corresponds to the truth that s can't reach vertices in the first part. For the second part, $dist[s]$ will remains 0 because there is no cycle from s to s. For the third part, let a, b be two vertices in topological order. We first search a and update all the dist of vertices that can be reached directly from a. Next, we search b and update all the dist of vertices that can be reached directly from b. Since a is in front of b in topological order and there is no path from b to a (acyclic graph), searching b won't influence $dist[a]$. Therefore, we start by searching s until all vertices are searched once. (search u means for each edge $(u, v) \in E : dist[v] = min\{dist[v], dist[u] + w(u, v)\}$) After the procedure, every $dist[u]$ can't be updated any more as we proved in the third part, which satisfies the end condition of the Bellman-Ford algorithm. So $dist[u]$ is the distance from s to u.

   Time Complexity:

   1. Finding and sorting the topological order of G takes $O(|V| + |S|)$ time.
   2. Initializing takes $O(|V|)$ time.
   3. Searching every vertex takes $O(|V| + |E|)$ time because we only pass every edge once.

   Therefore, the overall time complexity is $O(|V| + |E|)$.

   (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.

Proof:

Yes. $dist[u]$ means there exists a simple path from s to u whose length is $dist[u]$. After negation the weight of every edge, $dist[u]$ means there exists a simple path from s to u whose length is $-dist[u]$ in the origin graph because addition and subtraction are linear. According to (a), the above-mentioned algorithm finds the shortest distance from s to every vertex u, which means finding the smallest $dist[u]$. And the smaller $dist[u]$ is, the bigger $-dist[u]$ is. $dist[u]$ is the smallest means $-dist[u]$ is the biggest. Therefore, finding the smallest $dist[u]$ in the negated graph means finding the biggest $dist[u]$ in the origin graph where $dist_{ori}[u] = -dist[u]$.

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.

Algorithm:

1. Initialize:
   $T = \{s\}$.
   $B[v] \leftarrow false$ for all $v \in V$.
   $tdist[s] \leftarrow 0, tdist[v] \leftarrow INF$ for all v other than s.
   $tdist[v] \leftarrow w(s, v)$ for all $(s, v) \in E$.
   $check\_shortest[v] \leftarrow INF$ for all $v \in V$.
2. Explore:
   Find $v \notin T$ with smallest $tdist[v]$.
   $T \leftarrow T \cup \{v\}$
   if $check\_shortest[v] = tdist[v]$:
   $B[v] \leftarrow true$.
3. Update $tdist[u]$ and $B[u]$:
   if $tdist[u] = tdist[v] + w(v, u)$ for all $(v, u) \in E$:
   $check\_shortest[u] \leftarrow tdist[u]$.
   $tdist[u] \leftarrow min\{tdist[u], tdist[v] + w(v, u)\}$ for all $(v, u) \in E$.
   back to explore until T contains all $v \in V$.

Correctness:

The algorithm is based on Dijkstra algorithm and we have already proved the correctness of Dijkstra algorithm. $tdist[v]$ means the length of the shortest path from s to v for all $v \in T$. Notice that performing $check\_shortest[u] \leftarrow tdist[u]$ in the algorithm means there are at least two simple paths from s to u that the length of them are both $tdist[u]$. Now let's prove it: suppose there is only one simple path from s to u. Since s can reach every vertex in G, there are only one vertex v pointing at u or the number of simple paths will become 2. Then, before we check if $tdist[u] = tdist[v] + w(v, u)$, we have no chance to change $tdist[u]$ thus $tdist[u]$ remains INF and can't be equal to $tdist[v] + w(v, u)$, which can't satisfy the condition and therefore can't performe $check\_shortest[u] \leftarrow tdist[u]$. Contradiction! Finally, we check if $check\_shortest[u]$ is equal to the shortest distance from s to u, which is denoted by $tdist[u]$ when $u \in T$ to avoid the situation that the two simple paths are not shortest from s to u. If so, $B[u] \leftarrow true$. Therefore, every vertex v can be checked if there exists multiply shortest

paths from s to v.

Time Complexity:

To find $v \notin T$ with smallest $tdist[v]$ takes $O(log|V|)$ by using binary heap and takes $|V|$rounds. To check if $check\_shortest[v] = tdist[v]$ takes $O(|V|)$. To update key and check if $tdist[u] = tdist[v] + w(v,u)$ takes $O(log|V|)$ and $|E|$rounds. Therefore, the overall time complexity is $O(|V|log|V| + |E|log|V|) = O((|V| + |E|)log|V|)$.

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.

   About 7-8 hours(though it was divided into several days).

   1. 4
   2. 3
   3. 5
   4. 4

   No collaboration.