Algorithm Design and Analysis (Spring 2023)
Assignment 2
Deadline: April 10, 2023

1. (25 points) Given a directed weighted graph $G = (V, E, w)$ where edges can be negatively weighted and a vertex s, consider the execution of Bellman-Ford algorithm. Recall that the algorithm starts by initializing $\text{dist}(s) = 0$ and $\text{dist}(u) = \infty$ for each $u \neq s$; and, at each iteration, the algorithm updates $\text{dist}(v) \leftarrow \min\{\text{dist}(v), \text{dist}(u) + w(u, v)\}$ for each edge (u, v). We assume all vertices are reachable from s.

   (a) (10 points) In the class, we have proved that G contains a negatively weighted cycle if $\text{dist}(u)$ is updated for some $u \in V$ at the $|V|$-th iteration. In this sub-question, you are to complete the correctness proof of Bellman-Ford algorithm by proving the converse of this statement: if G contains a negatively weighted cycle, then there exists $u \in V$ such that $\text{dist}(u)$ is updated at the $|V|$-th iteration.

   We prove its converse-negative proposition: If there is no distance updated at the $|V|$-th iteration, then G doesn't contain a negatively weighted cycle.

   If no distance is updated in $|V|$-th iteration, then for every cycle $v_1, v_2 ... v_k$, $dist(v_{i+1\%k}) \leq dist(v_i) + w(v_i, v_{i+1\%k})$, in which $i = 1, ..., k$. Summing them up, we have $0 \leq \sum_{i=1}^{k} w(v_i, v_{i+1})$, which indicates there's no negatively weighted cycle in the graph.

   Some other proof:

   If there is no distance updated at $|V|$-round, then there's no distance update in following rounds. Thus the distance of every vertex is a finite number, which contradicts to the fact by going through the negative cycle, the shortest distance from s to vertices on or after that cycle can decrease infinitely.

   (b) (5 points) Give a counterexample to disprove the following claim: for a vertex t, if there is a path from s to t that contains a negatively weighted cycle, then $\text{dist}(t)$ must be updated at the $|V|$-th iteration.

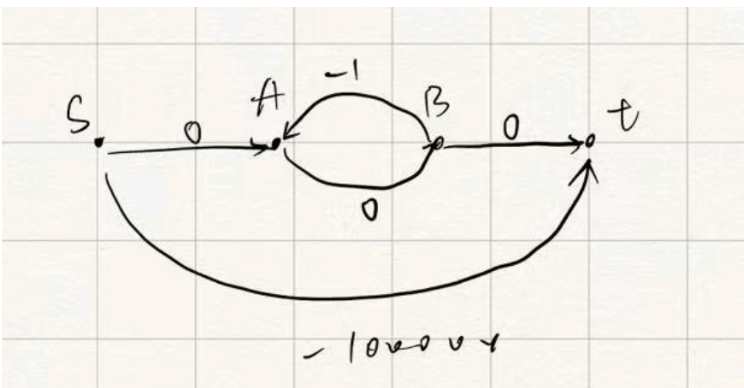

Figure 1: $\text{dist}(t)$ doesn't update in $|V|$-th iteration

(c) (10 points) Adapt Bellman-Ford algorithm to solve the following problem. Give a vertex s and a vertex t, decide if there is an s-t path that contains a negatively weighted cycle. You need to prove the correctness of your algorithm. Your algorithm must run in time $O(|V| \cdot |E|)$.

Algorithm:

1) First we apply Bellman-Ford Algorithm to the graph, and record all vertices that are updated in the $|V|$-th iteration. We use S to represent the set.

2) Then we construct a reverse graph $G' = (V, E')$, where $E' = \{(v, u)|(u, v) \in E\}$. We DFS the graph G' from vertex t. If it can reach any of the vertex in set S, then there is an s-t path that contains a negatively weighted cycle.

Correctness Proof:

The correctness proof contains two parts.

First, if there's a negative cycle on $s - t$ path, it can be searched by DFS from t in step 2. This can be achieved since the vertices set S contains at least one vertex of each negative cycle as a corollary of (1a).

Then, we claim that if we can reach a vertex in S through DFS from t in step 2, there's a negative cycle on $s - t$ path. This is obviously true for$v \in S$ that are on a negative cycle, so we only need to analyze those $v \in S$ but are not on a negative cycle. Because they are updated in $|V|$-th iteration, it implies there's a path from s to it containing a negative cycle. Thus, if we can reach a vertex in S through DFS from t, it implies an $s - t$ path containing negative cycle.

Time Complexity analysis:

It takes $O(|V| \cdot |E|)$ to do Bellman-Ford in step 1. In step 2, it takes $O(|E|)$ to construct G' and $O(|V| + |E|)$ to do DFS. Therefore, the time complexity is $O(|V| \cdot |E|)$ in conclusion.

Algorithm 1 Every-vertex Path

$G' = (V', E') \leftarrow$ the SCC graph of G.
Get the topological order $SCC[1], \cdots, SCC[h]$ of the SCC graph. ▷ Using DFS or other approaches.
if $s \in SCC[1]$, $t \in SCC[h]$ and edge $(SCC[i], SCC[i+1])$ exists for any i. **then**
    return Yes.
else
    return No.

2. (25 points) Given a directed graph $G = (V, E)$, a starting vertex $s \in V$ and a destination vertex $t \in V$. Design a polynomial time algorithm to decide if you can walk from s to t such that every vertex is visited. You are allowed to visit a vertex or an edge more than once. Prove the correctness of your algorithm and analyze its time complexity.

   See Algorithm 1.

   Correctness analysis: By the property of SCC, the problem can be converted to finding a path containing every SCC start from $SCC(s)$ and end at $SCC(t)$. Because the SCC graph is a DAG, we can find the topological order $SCC[1], \cdots, SCC[h]$ of these SCC and check whether SCC containing s rank 1 and SCC containing t rank h, which guarantees the path starts at $SCC(s)$ and ends at $SCC(t)$. Then we need to check whether there's a total order of all the SCC, which means we can walk from $SCC[1]$ to $SCC[h]$ through all the SCC. We achieve this by checking whether edge exists between each adjacent SCC.

   Time complexity: Finding SCC: $O(|V|+|E|)$. DFS: $O(|V|+|E|)$. Check the adjacent SCC: $O(|V|)$. Total: $O(|V|+|E|)$.

3. (25 points) Let $G = (V, E)$ be a graph and s be a vertex such that there is a path from s to each $u \in V$. We say G is a *good graph* if there exists a tree $T = (V, E')$ that share the same vertex set V with G such that T is both a depth-first search tree and a breadth-first search tree.

   (a) (10 points) If G is an undirected graph, prove that G is a good graph if and only if G is a tree.

   Sufficiency: If G is a tree, then its BFS tree and DFS tree are both itself.

   Necessity: Assume G is not a tree. And There exists a cycle $u_1 \leftrightarrow u_2 \leftrightarrow u_3 \leftrightarrow \cdots \leftrightarrow u_k \leftrightarrow u_1$. Without loss of generality, assume u_1 is firstly explored in BFS. Then we have parents of u_2 and u_k are both u_1 in BFS tree. However, that is impossible in DFS tree.

   (b) (10 points) If G is a good directed acyclic graph, prove or disprove that the vertex set V can be sorting in an array $\mathcal{L}$ such that $\mathcal{L}$ is both an ascending order of the distances from s and a topological order.

   The statement is false. An counterexample:

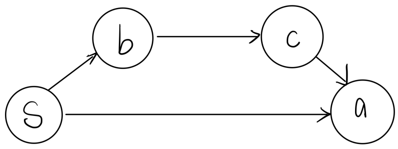

Figure 2: Counterexample for good directed acyclic graph

   (c) (5 points) Prove or disprove the converse of (b). That is, if G is a directed acyclic graph where the vertex set V can be sorting in an array $\mathcal{L}$ such that $\mathcal{L}$ is both an ascending order of the distances from s and a topological order, then G is a good graph.

   The statement is false. An counterexample:

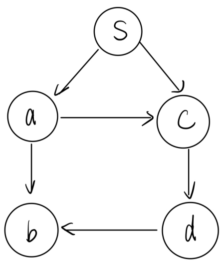

Figure 3: Counterexample for array $\mathcal{L}$

4. (25 + 5 bonus points) Given an undirected simple graph $G = (V, E)$ and a vertex s, you need to find a cycle with the minimum length that contains s. A cycle cannot contain an edge more than once.

   (a) (25 points) If G is unweighted, design an algorithm for this problem that runs in time $O(|V| + |E|)$.

   Consider *unweighted, undirected* graph G, we can use *BFS* to find the shortest path from s to other vertexes, which takes $O(|V| + |E|)$ time.

   Then, consider edges that are not in the *BFS tree*, we can find the minimum of $l_{s-v_1} + l_{s-v_2} + 1_{(v_1-v_2)}$. in $O(|E|)$.

   Now, to see that $s - v_1$, $s - v_2$ forms a cycle containing s (in other word, *least common ancestors* of v_1, v_2 should be s, otherwise, s in not in the cycle), we 'need to' check LCA. However, considering the requirement of time complexity and property of the problem, we find that we only need to check if v_1 and v_2 are on the different sub-trees rooted at 1st-level children of s.

   Thus,

   Algorithm:

   1. Run BFS rooted at s, meanwhile recording the level of each vertex(distance), BFS tree and subtree that each vertex is in(to see if s is in the cycle);
   2. For all edges $e = (v_1, v_2)$ that is not in the BFS tree, check if subtree of v_1 is different from subtree of v_2;
   3. For all edges that satisfies 2., find the minimum of $l_{s-v_1^*} + l_{s-v_2^*}$, return $s - v_1 - v_2 - s$.

   Sketch of Correctness:

   1. correctness of BFS to find shortest path on undirected, unweighted graph;
   2. correctness of checking if s is exactly in the cycle.

   Time Complexity:

   1. $O(|V| + |E|)$, 2. $O(|E|)$, 3. $O(|E|)$.

   $\Longrightarrow$ Total: $O(|V| + |E|)$.

   (b) (5 bonus points) If G is edge-weighted such that the weights are positive, design an algorithm for this problem that runs in time $O(|V|^2)$.

   Change BFS in Question4(a) to *Dijkstra* is enough for algorithm and correctness.

   As for Time Complexity, you can think the following 2 questions:

   1. $O(|E|) \leq O(|V|^2)$ holds for simple graphs. Of course, you can assume that there doesn't exists multi-edges; however, if there exists multi-edges and self-loops, how do we reduce it to simple graph?

2. Here Dijkstra without *heap optimization* is enough for $O(|V|^2)$, some of you use heap for $O(|V| \log |V|)$ time; however, does the correctness still holds?

For both parts, you need to prove the correctness of your algorithms and analyze their time complexities.