Algorithm Design and Analysis

Assignment 2

Deadline: April 15, 2024

1. (25 points) Let $G = (V, E)$ be an undirected connected graph.

   (a) Prove that you can sort the vertices $(v_1, \ldots, v_n)$ such that, for any $k \in \{1, \ldots, n\}$, the subgraph induced by the vertex set $V_k = \{v_1, \ldots, v_k\}$ is connected.

   (b) Does the claim hold for directed and strongly connected graphs? That is, given a directed graph $G = (V, E)$ that is strongly connected, can we sort the vertices $(v_1, \ldots, v_n)$ such that the subgraph induced by $V_k = \{v_1, \ldots, v_k\}$ is strongly connected for any $k \in \{1, \ldots, n\}$? If so, prove it; if not, provide a counterexample.

   Solutions:

   (a) We can refer to the idea of "breadth-first search"(BFS).First initialize a queue, choose a vertex and push it into the queue, then repeat the following steps until the queue is empty: pop the first vertex into the queue and record it, then push all vertexs(which is not recorded) connected to it in the queue. Because G is a connected graph, we can definitely traverse all vertexs. Similarly, due to the properties of connected graphs, we can ensure that the dequeue order is a answer that satisfies the problem.

   (b) This sorting method do not exists. Consider a counterexample: $V = \{v_1, v_2, v_3\}, E = \{v_1 -> v_2, v_2 -> v_3, v_3 -> v_1\}$, which is a directed and strongly connected graph. for $k = 2$, we cannot find two vertexs and edges, which can form a strongly connected graph.

2. (25 points) Let $G = (V, E)$ be a directed graph and s, t be two vertices. An (s, t)-*Hamiltonian path* is a path from s to t that visits each vertex exactly once. Design a polynomial time algorithm that, given a *directed acyclic graph* $G = (V, E)$ and two vertices s, t as inputs, decides if G contains an (s, t)-Hamiltonian path. Prove the correctness of your algorithm, and analyze its running time.

   Remark: We believe this problem does not admit polynomial-time algorithms on general graphs (that is not necessarily acyclic). In the last chapter of this course, we will prove that this problem is NP-complete.

   Solution:

   First, we need to check if there exists a path from s to t. If such a path does not exist, then G definitely does not contain an (s, t)-Hamiltonian path.

   If there exists a path from s to t, we can use a modified version of topological sorting and DFS to check for the existence of an (s, t)-Hamiltonian path. We can perform a depth-first search starting from s, checking during the search process if all vertices are visited exactly once.

   Time Complexity Analysis:

   Checking the existence of a path from s to t: $O(V + E)$.

   modified topological sorting algorithm: $O(V + E)$.

   If there is no path from s to t, then G definitely does not contain an (s, t)-Hamiltonian path, and the algorithm correctly identifies this.

   If there exists a path from s to t, we can find a path from s to t using depth-first search while ensuring all vertices are visited exactly once. Such a path is an (s, t)-Hamiltonian path. Therefore, the algorithm is correct.

3. (25 points) We have seen in the class that Dijkstra algorithm fails if the graph contains negatively-weighted edges. Consider the following variant of Dijkstra algorithm. Given a directed weighted graph $G = (V, E, w)$ where $w(u, v)$ may be negative, find an integer W such that $w(u, v) + W > 0$ for each edge $(u, v) \in E$, and define the new weight of (u, v) as $w'(u, v) = w(u, v) + W$. Now, $G' = (V, E, w')$ is a positively weighted graph where the weight of each edge is increased by W. Implement Dijkstra algorithm on $G' = (V, E, w')$, and a shortest path from s to every vertex u in G' is also a shortest path in G.

   (a) (10 points) Does this algorithm work for directed acyclic graphs? If so, prove it; if not, provide a counterexample.

   (b) (15 points) A *directed grid* is a directed weighted graphs $G = (V, E, w)$ where the set of vertices is given by $V = \{v_{ij} \mid i = 1, \ldots, m; j = 1, \ldots, n\}$ ($m, n \in \mathbb{Z}^+$ are two parameters) and the set of directed edges is given by

$$E = \left(\bigcup_{i=1,\ldots,m-1;j=1,\ldots,n} \{(v_{ij}, v_{(i+1)j})\} \right) \cup \left(\bigcup_{i=1,\ldots,m;j=1,\ldots,n-1} \{(v_{ij}, v_{i(j+1)})\} \right).$$

   That is, the vertices form a $m \times n$ grid. There is a *directed* edge *from* every vertex *to* the vertex "right above" it, and there is a *directed* edge *from* every vertex *to* the vertex "to the right of" it (unless the vertex is "on the boundary"). The weight of each edge is specified as input and can be negative.

   Does the algorithm work for directed grids? If so, prove it; if not, provide a counterexample.

Solutions:

   (a) This algorithm does not work. Consider a counterexample: $V = \{v_1, v_2, v_3, v_4\}, E = \{v_1 -> v_3 : 5, v_1 -> v_2 : 1, v_2 -> v_3 : 2, v_1 -> v_4 : -4\}$, Let $W = 5$, then the weights of all edges are positive in G'. Now the shortest past from v_1 to v_3 in G is $v_1 -> v_2 -> v_3$, but $v_1 -> v_3$ in G'. So the shortest path from s to t in G' may be not a shortest path in G.

   (b) This algorithm works. because the path can only be "up" or "right", refering to mathematical induction method, we can prove its correctness when $m, n = 1, 2$. If it holds when $m, n = a - 1, b - 1$, for $m, n = a, b - 1$, it becomes a subproblem of scale 1 again.

4. (25 points) Given a directed graph $G = (V, E)$ where each vertex can be viewed as a port. Consider that you are a salesman, and you plan to travel the graph. Whenever you reach a port v, it earns you a profit of p_v dollars, and it cost you c_{uv} if you travel from u to v. For any directed cycle in the graph, we can define a profit-to-cost ratio to be

$$r(C) = \frac{\sum_{(u,v)\in C} p_v}{\sum_{(u,v)\in C} c_{uv}}.$$

As a salesman, you want to design an algorithm to find the best cycle to travel with the largest profit-to-cost ratio. Let r^* be the maximum profit-to-cost ratio in the graph.

(a) If we guess a ratio r, can we determine whether $r^* > r$ or $r^* < r$ efficiently? Design an algorithm to determine this. Prove the correctness of your algorithm, and analyze its running time.

(b) Based on the guessing approach, given a desired accuracy $\epsilon > 0$, design an efficient algorithm to output a good-enough cycle, where $r(C) \geq r^* - \epsilon$. Justify the correctness and analyze the running time in terms of $|V|$, ϵ, and $R = \max_{(u,v)\in E}(p_u/c_{uv})$.

(Hint: You may want to construct another graph with appropriate edge weights, and use an appropriate shortest path algorithm.)

Solutions:

(a) Observe that if there exists a profit-to-cost ratio r^* greater than r. Thus, we can design an algorithm to check if there exists a cycle whose profit-to-cost ratio is greater than r^*. The algorithm proceeds as follows:

Use the Bellman-Ford or Floyd-Warshall algorithm to find all shortest paths in the graph.

Perform a topological sort on all vertices.

Starting from the source vertex, compute the maximum profit-to-cost ratio for each vertex in topological order. For each vertex v, update its maximum profit-to-cost ratio to max(current ratio, ratio of upstream vertex+adjustment value of the edge).

If a profit-to-cost ratio greater than r is found, return True; otherwise, return False.

The time complexity of the algorithm is $O(|V| \cdot |E|)$.

(b) For a given precision ϵ, we can employ binary search to find the maximum r such that there exists a cycle with a profit-to-cost ratio greater than or equal to $r - \epsilon$. Then, we can use the above algorithm to check if such a cycle exists.

The algorithm proceeds as follows:

Initialize the left and right boundaries of the binary search to 0 and R respectively, where $R = \max_{(u,v)\in E}(p_u/c_{uv})$. In each iteration, compute the current guess ratio r and use the above algorithm to check if there exists a cycle with a profit-to-cost

ratio greater than or equal to $r - \epsilon$. If such a cycle exists, update the left boundary to the current guess ratio; otherwise, update the right boundary to the current guess ratio. Repeat steps 2 and 3 until the difference between the left and right boundaries is less than ϵ. Return the left boundary as the final guessed ratio. The correctness of the algorithm is guaranteed by the properties of binary search. The time complexity of the algorithm is $O(|V| \cdot |E| \log R/\epsilon)$.

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.

   I spent about three hours completing this assignment.

   I think the difficulty score is 5. The first question is relatively easy, while the following questions are more difficult, especially 3 and 4.

   I used ChatGPT when answering, but due to the complexity of the problem, it did not provide me with much help in solving it. I think the answers of AI are not always correct, and questions that rely on searching for information cannot be called easy. Also, my English proficiency also influenced my answers to a certain extent