CSC265 W23: Assignment 5

Due: December 7, by midnight

Guidelines: (read fully!!)

- Your assignment solution must be submitted as a *typed* PDF document. Scanned handwritten solutions, solutions in any other format, or unreadable solutions will **not** be accepted or marked. You are encouraged to learn the LaTeX typesetting system and use it to type your solution. See the course website for LaTeX resources.
- Your submission should be no more than 7 pages long, in a single-column US Letter or A4 page format, using at least 10 pt font and 1 inch margins.
- To submit this assignment, use the MarkUs system, at URL `https://markus.teach.cs.toronto.edu/2023-09`
- This is a *group assignment*. This means that you can work on this assignment with *at most one other* student. You are *strongly encouraged* to work with a partner. Both partners in the group should work on and arrive at the solution together. Both partners receive the same mark on this assignment.
- Work on all problems together. For each problem, one of you should write the solution, and one should proof-read and revise it. The first page of your submission must list the *name*, *student ID*, and *UTOR email address* of both group members. It should also list, for each problem, which group member wrote the problem, and which group member proof-read and revised it.
- You **may not** consult any other resources except: your partner; your class notes; your textbook and assigned readings. *Consulting any other resource, or collaborating with students other than your group partner, is a violation of the academic integrity policy!*
- You may use any data structure, algorithm, or theorem previously studied in class, or in one of the prerequisites of this course, by just referring to it, and without describing it. This includes any data structure, algorithm, or theorem we covered in lecture, in a tutorial, or in any of the assigned readings. Be sure to give a *precise reference* for the data structure/algorithm/result you are using.
- Unless stated otherwise, you should justify all your answers using rigorous arguments. Your solution will be marked based both on its completeness and correctness, and also on the clarity and precision of your explanation.

Question 1. (8 marks)

In this question you are given a rooted tree T, where each edge e of the tree has non-negative integer weight $w(e)$. Suppose that the set of vertices of the tree is $V = \{1, \ldots, n\}$, and that the tree is represented by an array A, where, for any $u \in V$, $A[u]$ is a linked list of all the children of u, ordered from left to right. Together with each child v of u, in $A[u]$ we store the weight $w(e)$ of the edge $e = (u, v)$ between u and v. The root of the tree is the node 1.

In addition to T, you are also given a collection P of m pairs of vertices of T. You can assume that P is also stored in an array B, where, for each $u \in V$, $B[u]$ is a (potentially empty) linked list of all nodes v such that $\{u, v\} \in P$.

The goal in the question is to compute all distances between pairs of vertices in P. The distance $\delta(u, v)$ between two nodes u and v in V is defined to be the sum of the weights in the unique path between u and v in T.

Consider the following high-level description of an algorithm:

- For each $u \in V$ compute the distance $D[u]$ from 1 to u (stored in an array D).
- Start a post-order traversal of T at the root node 1. (I.e., nodes are traversed left to right, and a parent node is traversed after its children.)
  - Maintain throughout a collection $\mathcal{S}$ of sets of vertices, with the following invariant. Suppose u is the node last added to the traversal, and that it is connected to the root by a path $u_0 = u, u_1, \ldots, u_h = 1$. Then $\mathcal{S} = \{S_0, \ldots, S_h\}$, where the set S_i contains u_i and all descendents of u_i that precede u in the post-order traversal, but are not descendents of u_{i-1}. The set S_0 contains u and all descendents of u.
  - After a node u has been added to the traversal, check, for each pair $\{u, v\}$ in P, whether v precedes u in the traversal. If it does, then find which set S_i it belongs to inside $\mathcal{S}$, and use the corresponding u_i and D to determine $\delta(u, v)$.

Using data structures we have learned in class, give pseudocode that implements the algorithm above, and runs in worst-case time complexity $O((n+m)\log^*(n))$. Prove that your algorithm runs in the required time complexity, and that it correctly computes all distances $\delta(u, v)$ for all $\{u, v\} \in P$.

Question 2. (14 marks)

Let $G = (V, E)$ be an unweighted undirected connected graph with vertex set $V = \{1, \ldots, n\}$ and $|E| = m$ edged. Assume that G is represented by its *adjacency matrix* A. In this problem we want to compute the *distance matrix* of G: the $n \times n$ matrix D in which $D[u, v] = \delta(u, v)$ is the distance between nodes u and v, i.e. the number of edges in the shortest path betweeen u and v.

Part a. (2 marks)

Give an algorithm that computes D given the adjacency matrix A in worst-case running time $O(n^3)$. Show that your algorithm computes D correctly, and that it runs in the required running time.

Part b. (4 marks)

Define the *squared* graph $G^2 = (V, E^2)$ of G by: $(u, v) \in E^2 \Leftrightarrow 1 \leq \delta(u, v) \leq 2$. I.e. there is an edge between u and v, $u \neq v$, in G^2 if and only if there is a path of length at most 2 betwen u and v in G. Assume you are given the distance matrix D_{sq} of G^2, i.e. $D_{sq}[u, v]$ equals the number of edges in the shortest path between u and v in G^2. Assume further that you are also given the $n \times n$ matrix P defined by $P[u, v] = D[u, v] \bmod 2$, i.e. $P[u, v] = 1$ if $D[u, v]$ is odd, and $P[u, v] = 0$ otherwise. Give an algorithm ComputeD(D_{sq}, P) that computes the distance matrix of G from the matrices D_{sq} and P in worst-case running time $O(n^2)$. Justify the correctness and running time of your algorithm.

Hint: Can you give a simple formula for $D_{sq}[u, v]$ given $D[u, v]$?

Part c. (4 marks)

Define G^2, D_{sq}, and P as above and define a new $n \times n$ matrix T by

$$T[u, v] = \frac{1}{\deg(v)} \sum_{w:(w,v)\in E} D_{sq}[u, w].$$

Above, $\deg(v)$ is the degree of v in G, i.e. $\deg(v) = |\{w : (w, v) \in E\}|$ is the number of edges in E for which v is one of the endpoints. Give an algorithm ComputeP(D_{sq}, T) that computes the matrix P from D_{sq} and T in time $O(n^2)$. Justify the correctness and running time of your algorithm.

Hint: Suppose that you know that $P[u, v] = 0$, i.e., that $D[u, v]$ is even. What can you say about how $D_{sq}[u, v]$ and $D_{sq}[u, w]$ compare for any w such that $(v, w) \in E$? What about when $P[u, v] = 1$, i.e., $D[u, v]$ is odd? What do your observation imply about $T[u, v]$ in comparison to $D_{sq}[u, v]$?

Part d. (4 marks)

Assume that you are given the following two algorithms:

- Square(A): computes the adjacency matrix of G^2 given the adjacency matrix A of G;
- ComputeT(D_{sq}, A): computes the matrix T as defined above from D_{sq} and the adjacency matrix A of G.

Assume that both algorithms run in time $O(n^\omega)$, where $2 < \omega \leq 2.81$. Using Square and ComputeT, and the algorithms ComputeD and ComputeP that you designed in the previous two subproblems, give an algorithm that computes the distance matrix D of a graph G given by its adjacency matrix in time $O(n^\omega \log(n))$. Justify the correctness and running time of your algorithm

Remark: For those of you who are familiar with basic linear algebra, the algorithm ComputeT(D_{sq}, A) simply computes the matrix product $D_{sq}A$ and divides every entry in the v-th column of the resulting matrix by $\deg(v)$. Similarly, Square(A) can also be computed using a matrix product: compute $AA + A = A^2 + A$, replace every non-zero entry with 1, and set all diagonal entries to 0. There is an algorithm due to Strassen that computes the product of any two $n \times n$ matrices in time $O(n^{\log_2(7)})$: check chapter 4.2. of CLRS. Thus, both algorithms can be implemented in time $O(n^\omega)$ for $\omega \leq \log_2(7) < 2.81$.