CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2024**

Homework #4

Name(s): $FIRST_1$ $LAST_1$, $FIRST_2$ $LAST_2$, $FIRST_3$ $LAST_3$

Homework Policy

- If you leave a question completely blank, you will receive 20% of the grade for that question. This however does not apply to the extra credit questions.
- You may also consult all the materials used in this course (lecture notes, textbook, slides, etc.) while writing your solution, but no other resources are allowed.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "sort the array in $\Theta(n \log n)$ time using merge sort".
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question). See "Practice Homework" for an example.
- The extra credit problems are generally more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.
- **Groups:** You are allowed to form groups of size *two* or *three* students for solving each homework (you can also opt to do it alone if you prefer). The policy regarding groups is as follows:
  - You can pick different partners for different assignments (e.g., from HW1 to HW2) but for any single assignment (e.g., HW1), you have to use the same partners for all questions.
  - The members of each group only need to write down and submit a single assignment between them, and all of them will receive the same grade.
  - For submissions, only one member of the group submits the full solutions on Canvas and lists the name of their partners in the group. The other members of the group also need to submit a PDF on Canvas that contains only a single line, stating the name of their partner who has submitted the full solution on Canvas (Example: Say A, B, and C are in one group; A submits the whole assignment and writes down the names A, B, and C. B and C only submit a one-page PDF with a single line that says "See the solution of A").
  - You are allowed to discuss the questions with any of your classmates even if they are not in your group. **But each group must write their solutions independently.**

Problem 1. A **walk** in a directed graph $G = (V, E)$ from a vertex s to a vertex t, is a sequence of vertices $v_1, v_2, \ldots, v_k$ where $v_1 = s$ and $v_k = t$ such that for any $i < k$, (v_i, v_{i+1}) is an edge in G. The *length* of a walk is defined as the number of vertices inside it minus one, i.e., the number of edges (so the walk $v_1, v_2, \ldots, v_k$ has length $k - 1$).

Note that the only difference of a walk with a **path** we defined in the course is that a walk can contain the same vertex (or edge) more than once, while a path consists of only distinct vertices and edges.

Design and analyze an $O(n + m)$ time algorithm that given a directed graph $G = (V, E)$ and two vertices s and t, outputs *Yes* if there is a walk from s to t in G whose length is even, and *No* otherwise. **(25 points)**

Problem 2. We have an undirected graph $G = (V, E)$ and two vertices s and t. Additionally, we are given a list of *new* edges F which do *not* belong to the graph G. For any edge $f \in F$, we define the graph G_f as the graph obtained by adding the edge f to the original graph G.

Design an algorithm that in time $O(n + m + |F|)$ finds the edge $f \in F$ such that the distance from s to t in G_f is minimized among all choices of an edge from F. **(25 points)**

Problem 3. Let $G = (V, E)$ be a connected, undirected graph. We start with coins placed on arbitrarily chosen vertices of G, and each coin can move to an adjacent vertex at every step. The objective is to determine if it is possible to bring all coins to the same vertex, and if so, compute the minimum number of moves required.

(a) Suppose there are two coins placed on two given vertices $u, v \in V$, which may or may not be distinct. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex, or to report correctly that no such configuration is possible. The input to your algorithm consists of the graph $G = (V, E)$ and the initial vertices u and v.

(b) Now suppose there are three coins placed on three given vertices of G. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where all three coins are on the same vertex, or to report correctly that no such configuration is possible.

(c) Finally, suppose there are forty-two coins placed on forty-two given vertices of G. Describe and analyze an algorithm to determine whether it is possible to move all 42 coins to the same vertex. For full credit, your algorithm should run in $O(V + E)$ time.

(40points points)

Problem 4. Consider a directed graph G, where each edge is colored either red, white, or blue. A walk in G is called a French flag walk if its sequence of edge colors is red, white, blue, red, white, blue, and so on. More formally, a walk $v_0 \to v_1 \to \cdots \to v_k$ is a French flag walk if, for every integer i, the edge $v_i \to v_{i+1}$ is red if $i \bmod 3 = 0$, white if $i \bmod 3 = 1$, and blue if $i \bmod 3 = 2$.

Describe an algorithm to find all vertices in G that can be reached from a given vertex v through a French flag walk.

Problem 5. A vertex v in a connected undirected graph G is called a cut vertex if the subgraph $G - v$ (obtained by removing v from G) is disconnected.

Describe an algorithm that identifies every cut vertex in a given undirected graph in $O(V + E)$ time.

Problem 6. Suppose we are given a directed acyclic graph (DAG) $G = (V, E)$ whose nodes represent jobs, and whose edges represent precedence constraints. Specifically, each edge $u \to v$ indicates that job u must be completed before job v can begin. Each node v also has a weight $T(v)$ indicating the time required to execute job v.

(a) Describe an algorithm to determine the shortest interval of time in which all jobs in G can be completed.

(b) Suppose the first job starts at time 0. Describe an algorithm to determine, for each vertex v, the earliest time at which job v can begin.

(c) Now describe an algorithm to determine, for each vertex v, the latest time at which job v can begin without violating the precedence constraints or increasing the overall completion time (computed in part (a)), assuming that every job except v starts at its earliest start time (computed in part (b)).

Problem 7. Tarjan's algorithm is an efficient method for finding all strongly connected components (SCCs) of a directed graph $G = (V, E)$ in linear time. A strongly connected component of a directed graph is a maximal subgraph in which any two vertices are reachable from each other. Tarjan's algorithm leverages Depth-First Search (DFS) to identify these components, ensuring each node in the graph is part of exactly one SCC.

The main idea of Tarjan's algorithm is to perform a DFS on G, keeping track of each vertex's discovery time v.index (a unique integer assigned in the order of discovery) and a secondary value v.lowlink, which is the smallest discovery index reachable from v in the DFS subtree, including v itself. The key steps of the algorithm are as follows:

1. For each unvisited vertex v in G, initiate a DFS that assigns v.index and v.lowlink.
2. Push each visited vertex onto a stack, and keep it on the stack until its strongly connected component has been fully explored.
3. For each successor w of v:
   - If w has not yet been visited, recursively call the DFS on w. Update v.lowlink as $\min(v.\text{lowlink}, w.\text{lowlink})$.
   - If w is on the stack, update v.lowlink as $\min(v.\text{lowlink}, w.\text{index})$.
4. After visiting all descendants of v, if v.lowlink $= v$.index, then v is the root of a strongly connected component. Pop vertices from the stack until v is removed, forming one complete SCC.

Using the above description, prove the correctness of Tarjan's algorithm. In particular, demonstrate that:

- The algorithm correctly identifies each strongly connected component.
- Each vertex belongs to exactly one SCC.
- The invariant properties of v.index and v.lowlink are maintained throughout the execution of the algorithm.

Problem 8. Suppose we are given both an undirected graph G with weighted edges and a minimum spanning tree T of G.

(a) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is decreased.

(b) Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is increased.

Challenge Yourself. Suppose you are given an arbitrary directed graph G in which each edge is colored either red or blue, along with two special vertices s and t.

(a) Describe an algorithm that either computes a walk from s to t such that the pattern of red and blue edges along the walk is a palindrome, or correctly reports that no such walk exists.

(b) Describe an algorithm that either computes the shortest walk from s to t such that the pattern of red and blue edges along the walk is a palindrome, or correctly reports that no such walk exists.