COMP3821/9801

Week 1, Problem Sheet

The beautiful thing about learning is that no one can take it away from you.

Announcements.

- Welcome to **COMP3821/9801**!
- Use this time to mingle and talk with your fellow peers.
  - You will need to form groups of 2 to 4 for the project.
  - While we strongly discourage solo groups, please consult the lecturer if you are considering doing a solo project. We will not grade you differently to a group-based project.
- The first set of formatif tasks have been released.

General Tips.

- All of these problems can be done with COMP2521 or COMP9024 techniques. If you are struggling to do most of the problems, please consult your tutor and/or lecturer for advice.
- In this course, we assume that
  - Dijkstra's algorithm runs with time complexity $O(m \log n)$ on a *non-negatively* weighted graph with m edges and n vertices, and
  - BFS/DFS runs with time complexity $O(m + n)$.

Guided Problem

For the first 10 or so minutes of the tutorial, the tutor will solve the following problem. For the rest of the tutorial, work in small groups to solve the remaining tutorial exercises. Solutions will be posted at the end of the week.

Let $G = (V, E)$ be an undirected and unweighted graph, where each vertex is coloured either red, blue, or green. Given two vertices u and v, describe an $O(m+n)$ algorithm that computes the shortest path from u to v which never visits two consecutive vertices of the same colour.

As usual, m represents the number of edges and n represents the number of vertices in G.

Tutorial Exercises

Problem 1

Let $G = (V, E)$ be an undirected and unweighted graph. A subset $S \subseteq E$ of edges is called *edgy* and you can decide if a particular edge is edgy in $O(1)$ time by a function $f : E \to \{0, 1\}$. Given two vertices u and v, describe an $O(m + n)$ algorithm to decide if there exist a walk from u to v that only uses edgy edges.

Problem 2

Let $G = (V, E)$ be a directed and unweighted graph, where each edge is coloured either red, blue, or green. A *colourful walk* is a walk in G that does not contain two consecutive edges of the same colour.

Given a starting vertex s, describe an $O(m + n)$ algorithm to find all vertices in G that are reachable from s through a colourful walk.

Note. *G is not necessarily a DAG.*

Hint. *Reduce the problem to a graph reachability problem by constructing an appropriate graph.*

Problem 3

Let $G = (V, E, w)$ be a directed, acyclic, and weighted graph, where $w : E \to \mathbb{Z}$ denotes the weight of the edges which may be negative, zero, or positive. Given two vertices u and v, describe an $O(m+n)$ algorithm that decides if there exist a path from u to v that passes through more positively weighted edges than negatively weighted edges.

Hint. *There is an $O(m + n)$ algorithm for single-source shortest path on a directed and acyclic graph; you can simply refer to it.*

Problem 4

Let $G = (V, E)$ be a directed and weighted graph, where $|V| = n$ and $|E| = m$. Additionally, you know that *exactly* one edge in G has negative weight and you also know that there are no negative cycles. Given such a graph G and two vertices $s, t \in V$, describe an $O(m \log n)$ algorithm to return the shortest path from s to t.

Hint. *We cannot directly apply Dijkstra's algorithm to the input graph; how could we modify it so that running Dijkstra's algorithm is applicable?*

Problem 5

You and your best friend from high school have just reconnected, and have decided to find a time to catch up. The problem is, the both of you live in different suburbs that are too far away so it'll be inconvenient for either of you to travel to the other person's suburb. Therefore, to minimise the travel time, the both of you decide to meet at an intermediate suburb but you don't know which suburb.

Formally, you are given an undirected and weighted graph $G = (V, E, w)$ with positive edge weights $w(e) > 0$ denoting the travel time along edge e. Each edge is undirected; therefore, the edge $u \to v$ has the same travel time as the edge $v \to u$. You are also given two distinct vertices $u, v \in V$ denoting the suburbs that you and your friend currently reside in.

Describe an $O(m \log n)$ algorithm to find the intermediate suburb t so that you and your best friend can meet as soon as possible, assuming you both leave *right now*.