Homework Assignment #1
Due: March 19, 4:00 p.m.

Along with your solutions to this assignment, hand in a *separate* sheet of paper containing your student number and the following declaration: "I have read and understood the policy on academic honesty on the CSE4101 course web page." Sign this paper and date it. Without this declaration, your solutions will not be marked.

Recall Kruskal's algorithm for computing a minimum spanning tree of a connected, edge-weighted, undirected graph $G = (V, E)$. Suppose there are n nodes, numbered 1 to n and m edges in the input graph. Kruskal's algorithm builds a set of edges T that eventually form a spanning tree. Initially, $T = \emptyset$. The algorithm first sorts the edges of the graph by weight. For each graph edge (u, v), taken in the sorted order, the algorithm tests whether there is already a path in T between u and v. If not, it adds (u, v) to the set T.

Making Kruskal's algorithm run quickly requires a good data structure that allows the algorithm to quickly test whether there is already a path in T between u and v. This assignment will consider a few different implementations of that test.

1. One possible data structure for Kruskal's algorithm is just to store the edges in T as an adjacency list structure: we keep an array $neighbours[1..n]$, where $neighbours[u]$ stores a pointer to a linked list of all nodes v such that T contains an edge between u and v.

(a) How would you update the data structure in $O(1)$ time whenever an edge is added to T?

(b) How could you test whether there is a path from u to v using this data structure in $O(n)$ time?

(c) Give a good upper bound on the worst-case running time of Kruskal's algorithm using this data structure. State your answer in terms of n and m using big-O notation.

(d) For all n, give a graph G_n with $O(n)$ edges such that Kruskal's algorithm runs in $\Omega(n^2)$ time on G_n using this data structure.

2. In Question 1, updates to the data structure were fast, but tests were slow. In this question, we use add a data structure to make tests very fast but updates slower.

At any point in the execution of Kruskal's algorithm, the edges added to T so far create a set of connected components of nodes. Each connected component of (V, T) will have a name. (The names will just be numbers between 1 and n.) We shall use an array $C[1..n]$, where $C[v]$ stores the name of the component to which v belongs. Initially, T is empty, so there are n different connected components with one node each. So, initially, we set $C[v] = v$ for all v.

Thus, the new data structure consists of the adjacency lists (as in Question 1) and the new array C.

OVER. . .

(a) Using this data structure, how can you test if there is a path between two nodes in T in $O(1)$ time?

(b) When an edge is added to T, two connected components of (V, T) are merged to create one bigger connected component. Show that updating the data structure each time an edge is added to T can be accomplished in time proportional to the number of nodes in the newly merged component.

(c) For all n, give a graph G_n with $O(n)$ edges such that Kruskal's algorithm runs in $\Omega(n^2)$ time on G_n using this data structure.

(d) Finally, we consider adding one additional part to the data structure: an array $S[1..n]$, where $S[i]$ stores the number of nodes in the component whose name is i. Initially $S[i] = 1$ for all i. When an edge is added, and causes two components to merge, we can use information in S so that updating the data structure (adjacency list, C and S) can be accomplished in time proportional to the number of nodes in the *smaller* of the two components being merged. Explain how. Then, use an aggregate analysis to prove that this results in an overall running time of $O(m \log n)$ for Kruskal's algorithm.

Hint: For each v, think about how many times, during the entire execution, $C[v]$ must be changed.