1 Written Question 1 (25 marks)

In the *depth-determination problem*, we maintain a forest \( F = \{T_i\} \) of rooted trees under three operations:

- MAKE-TREE\((v)\) creates a tree whose only node is \( v \).
- FIND-DEPTH\((v)\) returns the depth of node \( v \) within its tree.
- GRAFT\((r, v)\) makes node \( r \), which is assumed to be the root of a tree, become the child of node \( v \), which is assumed to be in a different tree than \( r \) but may or may not itself be a root.

(a) Suppose that we use a tree representation similar to the trees we used for the disjoint set data structures: \( p[v] \) is the parent of node \( v \), except \( p[v] = v \) if \( v \) is a root. If we implement GRAFT\((r, v)\) by setting \( p[r] = v \) and FIND-DEPTH\((v)\) by following the find path up to the root, returning a count of all nodes other than \( v \) encountered, show that the worst-case running time of a sequence of \( m \) MAKE-TREE, FIND-DEPTH, and GRAFT operations is \( \Theta(m^2) \).

By using something similar to the union-by-rank and path-compression heuristics, we can reduce the worst-case sequence complexity. Our implementation will use a tree \( S_i \) to represent each tree \( T_i \) from the disjoint forest \( F \). However, two corresponding trees \( S_i \) and \( T_i \) will not necessarily resemble each other. In fact, the implementation of \( S_i \) does not record the exact parent-child relationships but nevertheless allows us to determine any node’s depth in \( T_i \).

The key idea is to maintain in each node \( v \) a “pseudodistance” \( d[v] \), defined so that the sum of the pseudodistances along the path from \( v \) to the root of its set \( S_i \) equals the depth of \( v \) in \( T_i \). That is, if the path from \( v \) to its root in \( S_i \) is \( v_0, v_1, ..., v_k \), where \( v_0 = v \) and \( v_k \) is \( S_i \’s \) root, then the depth of \( v \) in \( T_i \) is \( \sum_{j=0}^{k} d[v_j] \).

For the remainder of this question, we will be implementing a data structure that uses this idea of “pseudodistance”.

(b) Give an implementation of MAKE-TREE.
(c) Show how to implement FIND-DEPTH. You may find it useful to look at the implementations of FIND-SET for the disjoint set ADT. Your implementation should perform path compression, and its running time should be linear in the length of the path to the root. Make sure that your implementation updates pseudodistances correctly.

(d) Show how to implement GRAFT(r, v), which combines the sets containing r and v. It may be useful to look at the implementation of UNION for the disjoint set ADT. Make sure that your implementation updates pseudodistances correctly. Note that the root of a set $S_i$ is not necessarily the root of the corresponding tree $T_i$.

(e) Give a tight bound on the worst-case running time of a sequence of $m$ MAKE-TREE, FIND-DEPTH, and GRAFT operations, $n$ of which are MAKE-TREE operations.

2 Written Question 2 (25 marks)

Let $G = (V, E)$ be an undirected, connected graph with weight function $w : E \to \mathbb{N}$ and suppose that $|E| \geq |V|$ and all edge weights are distinct.

Let $T$ be the set of all spanning trees of $G$ and let $T'$ be a minimum spanning tree of $G$. Then, a second-best minimum spanning tree is a spanning tree $T$ such that $w(T) = \min_{T'' \in T \setminus \{T'\}} \{w(T'')\}$.

(a) Show that the minimum spanning tree is unique but that the second-best minimum spanning tree need not be unique.

(b) Let $T$ be a minimum spanning tree of $G$. Prove that there exist edges $(u, v) \in T$ and $(x, y) \not\in T$ such that $T \setminus \{(u, v)\} \cup \{(x, y)\}$ is a second-best minimum spanning tree of $G$.

(c) Let $T$ be a spanning tree of $G$ and for any two vertices $u, v \in V$, let $\max[u, v]$ be an edge of maximum weight on the unique path between $u$ and $v$ in $T$. Describe an $O(n^2)$ algorithm that given $T$ computes $\max[u, v]$ for all $u, v \in V$.

(d) Give an efficient algorithm to compute the second-best minimum spanning tree of $G$.

3 Programming Question (25 marks)

Your task for this assignment will be to implement four distinct data structures for the Disjoint Set ADT. All four implementations will be tree based and will differ by the how they choose to union two sets together and whether they perform path compression. In DisjointSet.tar.gz you will find nine files:

- **DisjointSet.java**
  This interface contains the five functions that you are required to implement for each of the four data structures.

- **Element.java**
  This class contains the implementation of Element’s in the data structures and in the real world would have added fields to represent the data in an Element but in your application should not change much (if at all).
This class represents a node in the Disjoint Set that is not the representative of a set. It contains appropriate pointers to other nodes.

This class represents a node in the Disjoint Set that is the representative of a set and consequently has fields which reflect the values of the set.

This class gives a sample usage of the four classes. You may use this as a template for your test suite.

These four classes represent your four implementations.

All four data structures that you implement should implement the interface DisjointSet. The four data structures should be implemented as follows:

This data structure should perform union based on the weight. That is, the smaller set in terms of weight should be re-headed to the larger set in terms of weight. Weight here should always be the number of nodes in the set.

This data structure should perform union based on the rank. That is, the smaller set in terms of rank should be re-headed to the larger set in terms of rank. Rank should be implemented as described in lecture and in the lecture notes.

This data structure should be identical to that of UnionByWeight.java except that it uses path compression as described in lecture and in the lecture notes.

This data structure should be identical to that of UnionByPathCompression.java except that it uses path compression as described in lecture and in the lecture notes.

You can modify any of the files that you like and you are responsible for handing in all of your files both electronically and on paper stapled to the back of your assignment. The most important part of the assignment is that the DisjointSet.java does not change and that each of your four implementations work as they should.

In addition to this implementation, you are responsible for a report that does not exceed one page in length which includes the following:

• A description of your testing methodology including why you chose the tests that you did.

• A brief analysis of the differences between the four methods possibly including example inputs (i.e. sequences of operations) and some comments on the advantages and disadvantages of each of the four data structures.
4 Handing In

Handing in your assignment will take place in two parts:

- Drop box handin in BA 2220
  Your first page should be the cover sheet for this assignment found on the course web page under the assignments. Then, your answers to the written questions should follow. Next, you should include a copy of your report for the Programming Question. Finally, you should include a printout of the files that you modified.

- Electronic submission on cdf
  You should submit your code on the cdf systems using the `submit -c csc263h -a A4 files.ext` command.