Due in class: Feb 16.

In general, Homework problems are to be done by yourself (unless specified) with no help from another person. If you use any sources other than the textbooks for the class, you are to cite them. In any case, the writeup should be your own and not simply taken from another source.

(1) The following operations manipulate piles of “rocks”. All rocks are identical, except that each rock has a “grade”, which is an integer from 1 to \( n \).

- \textbf{CreateRock} creates a new rock of grade 1.
- \textbf{Promote(g)} changes one rock from grade \( g \) to grade \( g + 1 \).
- \textbf{Merge(g,h)} removes one rock of grade \( g \) and one rock of grade \( h \), and creates a rock of grade \( g + h \).
- \textbf{Delete(g)} removes a rock of grade \( g \).
- \( x = \text{MaxGrade()} \) returns the highest grade of any rock. It does not change any rocks or grades.

Implement this abstract datatype using only an integer Max and an array \( \text{Rocks}[1 \ldots n] \), where \( \text{Rocks}[g] \) is the number of rocks of grade \( g \). You may assume that \( \text{Rocks} \) is initially all zero, and that no operations that would cause errors are ever done. You are to show how to implement these operations so that all except \text{Delete} run in constant real time, and give a potential function proof that they all run in constant amortized time.

(2) Problem dealing with scheduling of unit jobs with deadlines and profits. Prove that this satisfies the requirement of being a matroid with the definition of independent set as a set of jobs that can feasibly be scheduled. In addition you should argue that the entire algorithm can be implemented efficiently using the Disjoint-Set Data Structure.

(3) Show that for any positive \( n \), there is a sequence of F-heap operations that creates an F-heap consisting of one tree that is a linear chain of \( n \) nodes.

(4) (Dynamic Tables with Insertion and Deletion) Use a potential function to show that the amortized cost of \text{DELETE} as well as \text{INSERT} is bounded by a constant. This generalizes the discussion from class where we only dealt with INSERTIONS and no deletions. The key is to also achieve the property that our space utilization exceeds the current number of elements by only a constant factor.

(5) (You may do this problem with another student if you wish).

You are given a graph \( G = (V, E) \) and a rooted spanning tree of \( G \) (called \( T \)). For every non-tree edge \( e = (u, v) \) we wish to compute the Nearest Common Ancestor \( NCA(u, v) \) in \( T \). A common ancestor of \( u \) and \( v \) is a node that is an ancestor of both. A nearest common ancestor is a common ancestor that is furthest from the root. A node can be its own ancestor as well. Design an algorithm that takes \( O(|E| \log |V|) \) time in the worst case. Can you do better?