Problem 1 (10 points) Construct a sequence of operations such that when we are finished, the resulting Fibonacci heap is a path of length $2n$.

Solution

Starting from an empty Fibonacci heap $F$, the basic idea is to create, recursively, a Fibonacci heap that is a linear chain of $k - 1$ nodes and then add one more node to the chain.

More precisely, we start creating a Fibonacci Heap of height 1, having root key $m$. Then we add the elements $m - 1$ (a value less than the current minimum), $m + 1$ (a value larger than the current minimum) and $m - 2$ (an even smaller element that has to be deleted to force the consolidation) and delete $m - 2$. The consequent consolidate step will generate the required Fibonacci Heap.

Using the above algorithm, it is easy to construct a heap that is a path of length $2n$.

Problem 2 (20 points) Show that any sequence of $m$ MAKE-SET, FIND-SET, and UNION operations, where all UNION operations appear before any of the FIND-SET operations, takes only $O(m)$ time if both path compression and union by rank are used. What happens in the same situation if only the path compression heuristic is used?

Solution

We will prove the first part using the accounting method. Set amortized cost to:

- MAKE-SET = 2
- LINK = 1
- FIND-SET = 1

There are at most $m$ MAKE-SET. Since amortized cost is 2 and real cost is 1, we have a credit of 1. We leave this 1 at the node.

Next comes a sequence of link. Amortized cost = actual cost = 1 here. Basically, after all the link the credits at each node is not affected.

Finally comes a sequence of FIND-SET. If the node is at level (depth) = 1, the actual cost = 1.

This leaves the credit unaffected. If the node is at depth $k$, the actual cost is $k$. However, we now use the credits at the node. Note that after the FIND-SET, all nodes whose credit has been used one at depth 1 due to path-comparison. Any future FIND-SET on those nodes will cost $O(1)$, so we are safe.

For the second part, note that the argument does not use union by rank. The amortized cost remains the same if union by rank is not used.
Problem 3 (10 points) Let \( G = (V, E) \) be a weighted, directed graph that contains no negative weight cycles. Let \( s \in V \) be the source vertex, and let \( G \) be initialized by \textsc{initialize-single-source}(\( G, s \)). Prove that there is a sequence of \(|V| - 1\) relaxation steps that produces \( d[v] = \delta(s, v) \) for all \( v \in V \).

(Hint: We stated in class that there is a shortest path tree and that the relaxation rule is \( d[v] = \min\{d[v], d[u] + w(u, v)\} \))

Solution

Consider the shortest path tree \( G' \) of \( G \). \( G' \) represents a partial ordering of all vertices. To obtain a linear ordering, a topological sort can be performed on \( G' \). The topological sort will provide the desired result because \( G' \) is a tree and hence acyclic.

The nodes can be relaxed in the order specified by the topological sort. There are at most \(|V| - 1\) such relaxation steps, which occurs when \( G' \) has a single leaf node. It must then hold that \( d[v] = \delta(s, v) \) for all \( v \in V \), because each \( v \) has been relaxed along one of its shortest paths.

Problem 4 (15 points) Prove or give a counterexample. For any graph \( G \) with distinct positive weights associated with each cross edge and for any cut of \( G \), the minimum weight spanning tree of \( G \) contains exactly one edge belonging to the cut.

Solution

Any tree with distinct edge weights is a counterexample.

Problem 5 (10 points) Prove that if the edge weights of a graph \( G = (V, E) \) are different, then the minimum weight spanning tree is unique.

Solution

Suppose the graph has two MSTs. Let \( L_1 \) (resp. \( L_2 \)) be the list of edges of the first (resp. second) MST listed in order of increasing weight. Let \( L_{ij} \) represents the \( j \)th edge in \( L_i \). Then there must exist \( k \) such that the weight of \( L_{1k} \) is less than that of \( L_{2k} \), while \( L_{1j} \) is the same as \( L_{2j} \) for every \( j < k \). Then we add \( L_{1k} \) to the second MST, and some cycle appears. Now, we could delete \( L_{2j} \) for some \( j > k \) to make \( L_2 \) a new spanning tree, say \( L'_2 \). Clearly the weight of \( L'_2 \) is less than the weight of \( L_2 \), which is a contradiction.

Problem 6 (35 points) Let \( G = (V, E) \) be a graph with \(|V| = n, |E| = n + k\), where \( k << \log n \) and edge weights \( w_1, w_2, \ldots, w_{|E|} \). Give a fast algorithm to find a minimum weight spanning tree.

Solution

As long as there is a vertex of degree at most 2, we do the following: Pick an arbitrary vertex with degree at most 2. Choose the minimum weight edge going out of the vertex and contract this edge. Throughout the process, we keep track of the edges we contracted.

After the above process, we will have a graph with at most \( 2k \) vertices, where each vertex has degree at least 3. Finding the minimum spanning tree takes \( O(k^2) \) time.

Now we add the edges we contracted back into the graph. It is easy to see that the new graph won’t have any cycles (because of the contraction process we did initially) and will be a MST.

2