

15-750 — Graduate Algorithms — Spring 2008

Miller and Sinop and Wu

Assignment 1 Due date: Friday, February 8th

Some Reminders:

- Read the Policies section on the course web site before you start working on this assignment. Collaboration *is* permitted for this assignment.
- You should refrain from using outside sources when solving these problems. For each problem, state whether you have seen it before. If you have questions, contact the course staff.
- We prefer that you type up your solutions (preferably using LaTeX). You may neatly hand-write your solutions, but if we have trouble reading them you will be required to type up future solutions.

1 A Dictionary using a Set of Arrays

[10 points]

The goal of this problem is to devise a data structure that stores a set of keys, and supports the operations search and insertion. However, we will not use trees or hash tables. The proposed structure instead consists of a collection of arrays, A_0, A_1, A_2, \dots , where A_i has size 2^i . Let's call i the rank of A_i . We maintain the invariant that our list of arrays is sorted by rank and there is at most one array of each rank. Each key is stored in exactly one array and the keys in each array are kept in sorted order. Also, while each individual array is in sorted order, there is no ordering relation between arrays (e.g., perhaps $A_0 = [5]$, $A_1 = [2, 6]$, and $A_3 = [1, 3, 4, 7, 8, 9, 10, 11]$).

Searching for an element x is implemented by going through the arrays in order from smallest to largest, and for each one, using binary search to determine if x is in it or not.

1. Suppose the structure has n keys. What is the worst-case running time per search operation.
2. Explain how to implement insertion in such a way that the amortized cost per insert is $O(\log n)$, while maintaining the invariants above. Assume that you start from the empty structure. and n is the total number of insertions done.
3. Explain why this scheme gets into trouble if we want to also support deletions (while maintaining the above invariants).

Supporting deletions

We can augment the above data structure to support deletions as follows. When we want to delete an element, we leave it in place, but simply mark it as deleted. We also increment a delete-count for the array that the element is in. Whenever the delete-count of an array A_i reaches 2^{i-1} (i.e., half

of its elements are deleted) we remove the deleted elements and then either (a) move what's left into A_{i-1} if A_{i-1} was empty, or (b) merge what's left with A_{i-1} back into A_i if A_{i-1} was full. So, we maintain the invariant that each array is either empty or full (but "full" may include elements marked as deleted). We also maintain the invariant that at most half the elements in any array are marked as deleted.

Prove that this method continues to support efficient searches and inserts, while also supporting deletions in amortized time equal to the search time (we need to find the element first) plus $O(1)$.

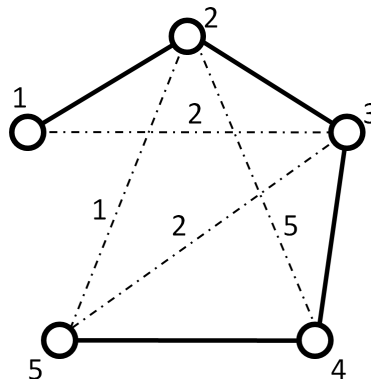
Be clear and careful: for instance, you need to argue that the deletions don't ruin the good time bounds for insertion. It may help to think of each insert operation as arriving with $O(\log n)$ insert-tokens (to be placed at appropriate points in the data structure) and to think of each delete operation as arriving with $O(1)$ delete-tokens (also to be placed at appropriate points in the data structure).

2 Removing Even Cycles from a Graph

[20 points]

Suppose you are given a weighted graph $G = (V, E, W)$ and a spanning tree $T \subseteq E$. Assume that all weights are positive. Your task is to find a set of edges $E' \subseteq E - T$ with *minimum cost*, such that when you remove these edges from the graph, there should be no *even*-cycles. Assume that the maximum number of edges incident to any node is bounded by a constant c (so the running time of your algorithm can be in $2^c, c!, etc$).

For example, consider the graph given below. Bold edges are the tree edges, and dashed edges are the ones which you are allowed to remove. In this case, the set of edges with minimum cost, whose removal leaves no even cycles, is $\{(1, 3), (3, 5), (2, 5)\}$ with cost 5.



- [10 points] First give a polynomial-time algorithm to solve this problem when T is a path (like in the above graph).
- [10 points] Now generalize your solution for the case where T is any spanning tree.

In both of these parts make sure you include a proof of correctness and a runtime analysis.

3 Computing Matrix Determinant

[5 points]

Given an $n \times n$ matrix M , describe a divide and conquer based algorithm which computes the determinant of M in $O(n^\omega)$ time, where $O(n^\omega)$ is the running time of matrix multiplication (current best known bound for ω is 2.376).

You can use the fact that matrix inversion and matrix multiplications are identical, in other words, you can find the inverse of a given $n \times n$ matrix M in time $O(n^\omega)$. To simplify the problem so that all need inverse exists you may assume that M is symmetric and positive definite. That is, $M = M^T$ and $x^T M x > 0$ for all vectors $x \neq 0$.

4 Fibonacci Heaps

[5 points]

In the original Fibonacci heap implementation, a node was cut when it lost two children. What might possibly go wrong if we do not cut a node *until* it lost three children? Can you give an example? Let S_i be the minimum number of vertices in a such a Fibonacci tree with rank i . What is the asymptotic size of S_i ?