

**COMP 5711: Advanced Algorithm**  
**Assignment 2**

**Amortized Analysis (CLRS Ch 17)**

Problem 17-2 (20 pts) The *logarithmic method* is a general technique to make a static data structure dynamic. Here we see how we apply it on a sorted array. We know that binary search in a sorted array takes logarithmic search time, but the time to insert or delete an element is linear in the size of the array. Such a data structure is usually said to be *static* as it is very expensive to do insertions and deletions.

Let  $n$  be the number of elements. Under the logarithmic method, instead of maintaining a single array containing all  $n$  elements, we look at the binary representation of  $n = \overline{b_k b_{k-1} \dots b_0}$ . Then we build a sorted array of size  $2^i$  if  $b_i = 1$ . Clearly, all the arrays have total size  $n$ .

- (a) Show that the query time now becomes  $O(\log^2 n)$ , i.e., a logarithmic factor slower than in the original array.
- (b) However, we can do insertions much faster. To insert a new element, thereby incrementing  $n$  by 1, we look at how the binary representation of  $n$  changes. Specifically, for some  $i$ ,  $b_i$  changes from 0 to 1 while  $b_j$  changes from 1 to 0 for every  $j = 0, 1, \dots, i-1$ . Therefore, we build a sorted array of size  $2^i$ , whose elements are all those from the arrays of size  $2^{i-1}, \dots, 2^0$ , plus the new element. These arrays are then deleted. First, show that this process can be done in time  $O(2^i)$ , i.e., linear in the number of elements involved.
- (c) Show that the amortized cost of an insertion is  $O(\log n)$ .

Problem 17-3 (20 pts) Most binary trees use rotation to restore balance (AVL-tree, red-black tree, splay tree). Another way for rebalancing a binary tree is *partial rebuilding*. For each node  $u$  in a binary tree, let  $size(u)$  denote the size (number of nodes) of the subtree below  $u$ , including  $u$  itself, and let  $u.left$  and  $u.right$  denote the left and right child of  $u$ , respectively. We say that  $u$  is  $\alpha$ -balanced (for some constant  $1/2 < \alpha < 1$ ) if  $size(u.left) \leq \alpha \cdot size(u)$  and  $size(u.right) \leq \alpha \cdot size(u)$ .

Insertions and deletions are done as in an ordinary binary tree, but without rotations. More precisely, an insertion simply adds a new leaf. To delete a node  $v$ , if  $v$  is a leaf or an internal node with only one child, we delete it directly; otherwise, we find the largest element in  $v$ 's left subtree, use it to replace  $v$ , and delete that element.

After an insertion or deletion, we find the highest node that is out of balance, and simply rebuild the whole subtree under that node.

- (a) Show that a subtree can be rebuilt in linear time (linear to the subtree size).
- (b) Show that the amortized cost of each insertion/deletion is  $O(\log n)$ , where  $n$  is the size of the whole tree.

Problem A (20 pts) Recall that in the dynamic table problem, we used the following strategy ( $\alpha$  is the load factor): When  $\alpha = 1$ , we rebuild a table doubling its original size; when  $\alpha = 1/4$ , we halve its size. Observe that using this strategy, we use at most  $4n$  space, where  $n$  is the number of elements stored in the table. This may not look good on very large data sets. Suppose your table is not allowed to use more than  $(1 + \epsilon)n$  space for some  $\epsilon \in (0, 1)$ , how would you modify the strategy? Your modified strategy should guarantee  $O(1/\epsilon)$  amortized cost per insertion/deletion. Note that any temporary storage used during a rebuilding is not counted. You should not treat  $\epsilon$  as a constant in your analysis.

### Randomized Algorithms (KT Ch 13)

Problem 1 (20 pts) Suppose we are given a graph  $G = (V, E)$ , and we want to cover each node with one of three colors. We say that an edge  $(u, v)$  is satisfied if the colors assigned to  $u$  and  $v$  are different. Consider a 3-coloring that maximizes the number of satisfied edges, and let  $c^*$  denote this number. Give a polynomial-time algorithm that produces a 3-coloring that satisfies at least  $\frac{2}{3}c^*$  edges with constant probability. [Hint:  $c^* \leq m$  where  $m$  is the number of edges in  $G$ .]

Problem 7 (20 pts) We have designed in class an  $(7/8)$ -approximation algorithm for the MAX-3SAT problem. now, consider the general MAX SAT problem, in which each clause can consists of any number of terms. We require that all variables in a single clause are distinct.

- (a) Suppose as in MAX-3SAT, we assign each variable independently to true or false with probability  $1/2$  each. Show that the expected number of clauses satisfied is at least  $k/2$ , where  $k$  is the total number of clauses. Give an example to show that there are MAX SAT instances such that no assignment satisfies more than  $k/2$  clauses.
- (b) For a clause with a single variable (e.g.,  $x_1$  or  $\overline{x_2}$ ), there is only one way to satisfy it. If there are two such clauses that are negation of each other, then this is a direct contradiction and cannot be both satisfied. Assume that in the input instance there are no such conflicting clauses. Modify the algorithm above to so that the expected number of satisfied clauses is at least  $0.6k$ .
- (c) Give a randomized algorithm for the general MAX SAT problem that satisfies at least  $0.6 \cdot \text{OPT}$  clauses in expectation, where  $\text{OPT}$  is the maximum number of clauses satisfied by an optimal assignment.