**Implementation and Analysis of Dynamic Arrays and Splay Trees**

**Rahul.M.Koushik**

**PES1201700121**

***Abstract-*This report analyzes two of the most commonly used Data Structures, Dynamic Arrays and Splay Trees. Explains the Amortized cost of each operation on both data structure.**

1. **INTRODUCTION**
2. A **Dynamic array** automatically grows when we try to make an insertion and there is no more space left for the new item. Usually the area doubles in size.
3. A **splay tree** is a self-balancing binary search **tree** with the additional property that recently accessed elements are quick to access again. It performs basic operations such as insertion, look-up and removal in O(log n) amortized time.
4. **IMPLEMENTATION**

***A.Dynamic Arrays***

The capacity of the array is fixed initially. As elements are inserted into the array, the size of the array is updated. When the size equals the capacity, the array is expanded. The new capacity of the array is equal to the product of the increase factor and the old capacity. When elements are deleted from

the array, the size of the array is updated. When the size of the array is equal to the product of the decrease factor and capacity, the array is contracted. It is contracted by a factor of the product of increase factor and decrease factor.

1. **Splay Trees**

Splay trees are binary search trees with good balance properties when amortized over a sequence of operations. When a node x is accessed, we perform a sequence of splay steps to move x to the root of the tree. There are 6 types of splay steps, each consisting of 1 or 2 rotations

1. **AMORTIZED ANALYSIS**
2. ***Dynamic Arrays***
3. ***Aggregate Method***

The idea of the aggregate method is to compute the total cost of n operations, then divide the total cost by n. This is the analysis technique used to analyze DFS, Dynamic Array and various other problems. 3.1 Analysis Let ti be the running time of the ith add operation. Total Running Time = n ∑ i=1 ti = n ∑ i=1,i6=2 k+1 ti + ∑ k,2 k+1≤n t2 k+1 = n ∑ i=1 1+ ∑ k,2 k+1≤n 2 k+1 ≤ n+ (2+4+8+...+2 l+1 ) = n+2 l+2 −2 ≤ n+4n = 5n where l is the largest number such that 2l +1 ≤ n. The last inequality is because 2 l +1 ≤ n 2 ·(2 l +1) ≤ 2n 2 l+1 ≤ 2n So the amortized time is equal to the total time divided by n. Then the amortized time for ”Add” operation is 5, which is equal to O(1)

1. ***Charging Method.***

Between two heavy operations 2k+1th operation and 2k+1+1th operation, we have 2k+1+1−(2 k+1)−1 = 2 k −1 light operations. The cost for the 2k+1 +1th operation is 2k+2 . So the average time we need to save is 2 k+2 2 k −1 ≈ 4 What we should to is to save 4 units of time for each light operation. Then when (2 k+1 + 1)th operation happens, I have saved (2 k −1)· 4 = 2 k+2 −4 units of time in my account. So the additional time I need to pay now is just 4. In summary, • For Light operations: Pay 1, Save 4 (In total, I cost 1+4 =5) • For Heavy operations: Use all savings, Pay 4 In total, in each step, I won’t pay more than 5. So the amortized cost for ”Add” operation is O(1)

1. ***Splay Trees***

For amortized analysis, we define the following for each node x:

1. The lr splay step: This is performed when x is a right child and x’s parent is a left child. The splay step consists of first a left rotation on y and then a right rotation on z. The rl splay step, for x being a left child and x’s parent being a right child, is analogous.
2. The r splay step: This is performed when x is the left child of the root y. The splay step consists of a right rotation on the root. The l splay step, for x being the right child of the root, is analogous. • a constant weight w(x) > 0 (for the analysis, this can be arbitrary) • weight sum s(x) = y∈subtree(x) w(y) (where subtree(x) is the subtree rooted at x, including x) • rank r(x) = log s(x) We use r(x) as the potential of a node. The potential function after i operations is thus φ(i) = x∈tree r(x).The amortized cost of a splay step on node x is ≤ 3(r (x) − r(x)) + 1, where r is rank before the splay step and r is rank after the splay step. Furthermore, the amortized cost of the rr, ll, lr, and rl splay steps is ≤ 3(r (x) − r(x)).Thus the amortized cost of the rr splay step is ≤ 3(r (x) − r(x)). The same inequality must hold for the ll splay step; the inequality also holds for the lr (and rl) splay steps. The +1 in the lemma applies for the r and l cases.

Analysis of Splay Tree Operations Find For the find operation, we perform a normal BST find followed by a splay operation on the node found (or the leaf node last encountered, if the key was not found). We can charge the cost of going down the tree to the splay operation. Thus the amortized cost of find is O(log n). Insert For the insert operation, we perform a normal BST insert followed by a splay operation on the node inserted. Assume node x is inserted at depth k. Denote the parent of x as y1, y1’s parent as y2, and so on (the root of the tree is yk). Then the change in potential due to the insertion of x is (r is rank before the insertion and r is rank after the insertion, s is weight sum before the insertion): k ∆φ = (r (yj ) − r(yj )) j=1 k = (log(s(yj ) + 1) − log(s(yj )) j=1 = k j=1 log s(yj)+1 s(yj ) \_x0005\_ ⎛ ⎞ k s(yj )+1 = log ⎝ ⎠ (note that s(yj )+1 ≤ s(yj+1)) s(yj) j=1 s(y2) s(y3) s(yk) s(yk) + 1 \_x0005\_ ≤ log · ··· · s(y1) s(y2) s(yk−1) s(yk) s(yk) + 1 \_x0005\_ = log s(yk) ≤ log n The amortized cost of the splay operation is also O(log n), and thus the amortized cost of insert is O(log n).

**Analysis of Splay Tree Operations**

Find For the find operation, we perform a normal BST find followed by a splay operation on the node found (or the leaf node last encountered, if the key was not found). We can charge the cost of going down the tree to the splay operation. Thus the amortized cost of find is O(log n).

Insert For the insert operation, we perform a normal BST insert followed by a splay operation on the node inserted. Assume node x is inserted at depth k. Denote the parent of x as y1, y1’s parent as y2, and so on (the root of the tree is yk). Then the change in potential due to the insertion of x is (r is rank before the insertion and r is rank after the insertion, s is weight sum before the insertion): k ∆φ ≤ log n .The amortized cost of the splay operation is also O(log n), and thus the amortized cost of insert is O(log n).

1. **CONCLUSIONS**

The dynamic array performs all operations in constant time amortized. The splay tree performs all operations in logarithmic time amortized.

1. **REFERENCES**
2. <https://www2.cs.duke.edu/courses/fall17/compsci330/lecture17note.pdf>
3. <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-854j-advanced-algorithms-fall-2005/lecture-notes/dzhang_splaytree.pdf>
4. <https://www.geeksforgeeks.org/>