Splay Trees and Dynamic Array Implementation using C++

Ishwar Choudhary
PES1201700189
CSE Department
PES UNIVERSITY
Bangalore,India
ishwarc404@gmail.com

***Abstract*—This paper discusses the significance of two data structures, Splay Trees and Dynamic Arrays and their implementation of using C++**

I. Introduction

Data structures are used to organize and manage data while enabling efficient access and modification.They are a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. **Splay tress** fall under the category of binary search trees with the additional property that recently accessed elements are quick to access again. They are self balancing and perform basic operations such as insertion, find and removal in amortized time. **Dynamic Arrays** are similar to generally implemented arrays but with a big improvement: Automatic resizing. They expand as you add more elements. So one doesn't need to determine the size ahead of time.

II. TIME COMPLEXITY

A. Splay Trees

Amortized complexity of find, insert, delete, and split is O(log n).Actual complexity of each splay tree operation is the same as that of the associated splay. Sufficient to show that the amortized complexity of the splay operation is O(log n).

B. Dynamic Array

The dynamic array introduces some important overhead in both time and space. If the dynamic array moves itself so that the entire array is contiguous (and so lookup is constant time), growing and moving the array will still take time. In the worst case asymptotically, inserting a new element takes O(n)O(n). However, it's important to look at the amortized analysis to see that the runtime is actually smaller; specifically, O(1)O(1).

III. Implementation

The following methods were employed to implement the two data structures using C++

A. Implementation: Splay Trees

- Through the use of vitual functions and abstract classes the implementation was structured.
- 5 different functions were deployed ; Insert, Remove,Find, Preorder and Postorder.
- The tree is build using a combination of nodes with two outgoing pointers. The basic Splay tree operations such as insert, remove, and print work very similarly to that of a Binary Search Tree with the exception of the additional splaying function.
- As the most recently updated/accessed element is given the highest preference, the "Splay" function is called which rearranges the tree and makes the recently accessed element the root of the entire Splay tree.
- The splaying function works based on any one of the 3 cases encountered ; **1)Zig-Zag situation:** X is (l child of a r child) : rotate same node to right or (r child of a l child) : rotate same node to left. **2)Zig Zig situation:** (l child of a l child) : rotate parent node to the right or (r child of a r child) : rotate parent node to the left then keep doing until it is the root **3)Zig situation:** when X is the child of the root. Rotate right or left based on the postition.

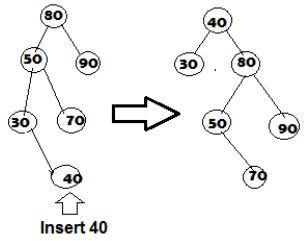

Fig. 1. An image of a insert operations on a Splay Tree

B. Implementation: Dynamic Array

- Through the use of vitual functions and abstract classes the implementation was structured.
- As the array size was a variable with constanly changing values, the array had to be allocated dynamically using the **new** keyword.

- While insertion operation, every time the array reaches it's maximum capacity, a new array with double the capacity is allocated and the contents of the previous array is copied onto the new array.
- The size of the dynamic array refers to the number of elements currently in the array whereas the capacity of the array denotes the total size of the array i.e the total number of elements the array can actually accomodate.
- While removal, once the capacity decreases after pop operation, according to the decrease decrease factor, we reduce the size of the array.

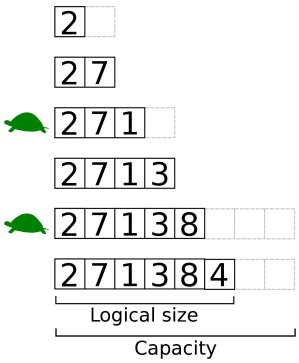

Fig. 2. An image of a insert operations on a Dynamic Array

IV. Applications

Data Structures form the building blocks of every program and have unimaginably infinite number of applications. Few of the applications of Splay Trees and Dynamic Array are as follows:

A. Applications: Splay Trees

- Good performance for a splay tree depends on the fact that it is self-optimizing, in that frequently accessed nodes will move nearer to the root where they can be accessed more quickly. Due to this fundamental property, they are extensively used in designing caching systems for computers.They are often used in applications where there is are memory constraints.

B. Applications: Dynamic Array

- Dynamic arrays benefit from many of the advantages of arrays, including good locality of reference and data cache utilization, compactness (low memory use), and random access. They usually have only a small fixed additional overhead for storing information about the size and capacity. This makes dynamic arrays an attractive tool for building cache-friendly data structures.

Acknowledgment

I would like to express my sincere gratitude to my supervisor to Prof.Channa Bankapur for providing his invaluable guidance and suggestions throughout the course.

References

1) http://ccf.ee.ntu.edu.tw/ yen/courses/ds17/chapter-4e.pdf
2) https://www.scribd.com/presentation/106689225/Application-of-Splay-Tree
3) https://www.sanfoundry.com/cpp-program-implement-splay-tree/