Implementation and Analysis of Dynamic Array

Hitesh Kumar
PES1201701511
PES UNIVERSITY
Bangalore, India
hiteshkumarhk.info@gmail.com

***Abstract*—Dynamic arrays are data structures whose initial size is not known at compilation. This data structure can be used when the data that we are dealing with is increasing or decreasing at run time but we want our code to use minimum memory. The size of the array is adjusted whenever the user wants to insert or delete on the data structure to have minimum memory footprint to the program.**

I. Introduction

Dynamic Table is a data structure with a random access, variable-size list data structure that allows elements to be added or removed. While Insertion and Deletion we accordingly expand and contract the capacity of the table based on parameters called Increase factor C and Minimum Load factor L. The aim of this data structure is to reduce the time complexity compared to the regular array.

II. Implementation

The dynamic table has been implemented using the operations APPEND, POP, GET, COUNT. For expansion and contraction of the table is done based on Load Factor. Let N be the number of elements at any given time in the array. And S be the size of the memory allocated to the array.

A. APPEND

The Append operation inserts an element at the end of the array. When the size of the array is equal to the capacity of the array the capacity of the array increased by the previously defined increase factor and the elements from the actual array is copied into a temporary array, the size of the actual array is changed using the new capacity and the elements are copied back into it.

B. POP

Managing the memory after deleting an element from the array is very crucial. We need an optimal fraction to decrease the size of the array when a pop happens. We might half the size of the array whenever we pop after compressing till half the size. But this wont be efficient if alternate push and pop are performed. So through amortized analysis, whenever while popping, if we only one fourth of the array is full, we half the capacity of our dynamic array. So if we have allocated 8000 bytes and we are using only 2000 bytes, we half the capacity to 4000 bytes.

C. GET

Takes an index as the parameter and returns the element present at the index in the array.

D. COUNT

Returns the number of elements in the current array.

III. AMORTIZED ANALYSIS

An elementary insertion costs 1 unit. Let cost be represented as C. If, number of elements = capacity , double the table size. Copy all existing items, then insert the new item. According to Cursory Analysis: O(n2) time for n operations. but we know that , we dont always expand the array size, hence cost is i if i-1 is exact power of 2, else cost is 1 . Total actual cost of n insert operations = summation(ci) n + summation(log(n- 1)*2i) is always less than or equal to 3n. Therefore, Aggregate Method derives O(n) is an upper bound for the actual cost of n operations. That is, amortized cost of an operation as 3 O(1). So basically This algorithm works in linear amount of time.

IV. REFERENCES

https://www.cse.cuhk.edu.hk/taoyf/course/comp3506/lec/dyn-array.pdf
https://www.geeksforgeeks.org/analysis-algorithm-set-5-amortized-analysis-introduction/