Report for Algorithms & Analysis Assignment 1

Chaithat Sukra (s3521070) (Working individually)

Data Scenarios

There are several factors, which may affect the time complexities of these three implementation structures including the size of multiple set, additional, removal, searching.

Searching is one of the most important thing in real-world development especially when the amount of data is very high. In order to see the clear difference between each data structures, first scenario is only focusing on searching by. For the second scenario, it will be a varying ratio of adding, removing, and searching because there will affect the time and it could help to give some recommendation.

Size of dataset

The aim of using various size of dataset is to find out the difference between using less amount and higher amount of data. So in this parameter, data generator class is created in this point. It will allow inserting fours integer index including starting data, adding data, removing data and searching data.

Generation of scenarios

In order to test time, Unix time command (System.nanotime) is the function, which can be found in lab3 for testing the benchmark of these scenarios. Each test cases will be run 5 times and use the average of them, which will be put in table.

Experiment and data generation

To generate data for testing, I have created a tester class called DataStructureTester. A sample call of using this class is shown on the following.

DataStructureTester x1 x2 x3 x4

string x1 // type of data structure you are going to test (sortedlinkedlist, linkedlist, bst)

int x2 // no of adding objects

int x3 // ratio of removing objects comparing to adding objects

int x4 // ratio of searching objects comparing to adding objects

In the construction, all type of objects will be created and the rest of function will be calculated according to the ratio of removing, and searching. Each test case will be tested for five times and record the average.

Scenario 1

In this scenario, the initial dataset will be created from the no of adding integer, which means there is no operation of removal as we will be able to find out the exact time complexities and performance of searching and adding from these three data structures.

The size of a multiple set is one of the main factors, which we can analyse. I will increase of adding integer from 1000->5000->10000->20000 so that we can see the difference between each data structures.

	LinkedList (ms)	SortedLinkedList (ms)	BST (ms)
1000 adding objects with 30% searching objects	D* linkedlist 1000 0 30 16.49907	D* sortedlinkedlist 1000 0 30 23.51514	D* bst 1000 0 30 5.349787
5000 adding objects with 30% searching objects	D* linkedlist 5000 0 30 106.69367	D* sortedlinkedlist 5000 0 30 118.975413	D* bst 5000 0 30 14.17699
10000 adding objects with 30% searching objects	D* linkedlist 10000 0 30 387.13866	D* sortedlinkedlist 10000 0 30 452.434857	D* bst 10000 0 30 23.13304
20000 adding objects with 30% searching objects	D* linkedlist 20000 0 30 1305.78966	D* sortedlinkedlist 20000 0 30 1771.97451	D* bst 20000 0 30 34.67902

Note D* = DataStructureTest

Evaluation

It can be seen that in the above table it takes approximately 5 milliseconds on 1000 objects and approximately 35 milliseconds on 20000 objects on Binary Search Tree but it takes much more on LinkedList and SortedLinkedList.

It is clear that Binary Search Tree consume less time even if I increase the amount of additional and searching object. This could confirm that the time complexity for searching of BST is better which has O(log(n) in or searching and inserting may become O(n) in average. On the other hand, even if LinkedList has O(1) in adding but it has O(n) in searching.

Scenario 2

In this scenario, the data scenarios will be varied between adding and removing versus searching. However, removing will be focused more on this scenario so I design to set as ratio as ten percentages for searching and thirty percentages for removing. As we can assume that this scenario will normally be found in many development environment.

	LinkedList	SortedLinkedList	BST
1000 adding objects with 10% searching objects and 30% removal objects	D* linkedlist 1000 30 10 14.03403	D* sortedlinkedlist 1000 30 10 14.66621	D* bst 1000 30 10 6.81608
5000 adding objects with 10% searching objects and 30% removal objects	D* linkedlist 5000 30 10 84.22322	D* sortedlinkedlist 5000 30 10 74.19187	D* bst 5000 30 10 16.31356
10000 adding objects with 10% searching objects and 30% removal objects	D* linkedlist 10000 30 10 302.87432	D* sortedlinkedlist 10000 30 10 280.43598	D* bst 10000 30 10 26.34922
20000 adding objects with 10% searching objects and 30% removal objects	D* linkedlist 20000 30 10 1113.39615	D* sortedlinkedlist 20000 30 10 1116.70537	D* bst 2000 30 10 44.55654

Note D* = DataStructureTest

Evaluation

As you can see, the consumption of time in this scenario, both linked and sort linked list are taking less comparing to the first scenario. It might be the cause of less searching operation. The more you increase your deleting operation, there is no affect to linked and sort linked list but binary search tree takes more time consumption.

The result is also going the same way as linked list has O(1) in insertion and deletion. There is not much difference between linked list and binary search tree in fewer amount of object.

Recommendation

In many many cases, I would recommend using Binary Search Tree according to the statistic and it is recommended that it is the most balance data structure. However there is a limitation. Even though the worst case can be less efficiency than linked list but it is not the main point. The implementation of Binary search tree is much more complex than linked lists. It also has issue regarding memory management. However, in fewer amounts of objects, linked list might be a better solution, as it’s a linear and less complex implementation.