Report for Algorithms & Analysis Assignment 1

Shuo Wang (s3677615), Khulud Alyoubi(s3637767)

We certify that this is all our own original work. If we took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed to my submission. We will show we agree to this honor code by typing "Yes": Yes.

Experimental Setup

There are two stages for this process. Firstly, building the data structure for Sequential and Linked tree is generated. After that, the different size of random values will be generated, which will be used to evaluate our data structure by counting times (use System.*nanoTime*() to calculate function running time by recording before and after function, and then, round it to second using Math.pow(10,9)), and, based on different value size, the different average times will be tested.

1. Data generator

To generate random values, four classes have been implemented, including DataGenerator.java, CreateSmallSizeNodes.java, CreateNodeValues.java, and mergeFile.java. DataGenerator will be used to generate random values and set the start and the end limits of the required range (according to System.*currentTimeMillis*() as a random seed).

CreateSmallSizeNodes can generate random values that are smaller than four thousand. Firstly, it will generate random values and put it into an array, after that, values will be set into printout file each line with rules (arr[i], arr[i*2+1], arr[i*2+2]), which will be used to grow a tree.

With cooperation with CreateNodeValues and mergeFile, they can generate values randomly that are over 4000 nodes based on BSP_combined.txt. First of all, CreateNodeValues will scan the file into memory and count how many times does it appear and mark it in a boolean array. After that, the system will pick-up all values which appear just once into a new array. And then, according to the requirements, a new array with the rest of values (the number of new values + values appears once = total requirement) will be generated and then, it will generate a new file except BSP_Combined.txt. In the end, mergeFile can combine BSP_Combined.txt and new file to fulfill the requirements. For example, if the number of 10^4 values is needed, CreateNodeValues can generate the rest of 6000 nodes and the function mergeFile will combine BSP_combined.txt and the new file to generate a file with 10^4 values.

2. Experiment design

In order to meet the requirements with different complexity of the initial tree, different levels of the tree will be generated by the different depth of tree by 102 to 106 for the rest of three scenarios because the different order of magnitude is more meaningful to present the performance and magnify the difference in a different data structure. The time which will be used to evaluate the performance for each scenario will be counted. To minimize the impact of unrelated factors, for 102 to 104, the test will be done 100 times and the total time/100 will be seen as the final results and count, which will test the average cases. Because, with an increase in the number of values, consumption time will increase rapidly, 50 times and 3 times tests will be used in 105 and 106. Calculating the growth of the running time using this strategy is going to show the behavior of the algorithm in most cases.

In the first scenario, node splitting will be evaluated by the period. For node addition, in order to make the tree keep in growing, extra 100 nodes will be added in 103 to 106. the results for this period will be used to evaluate the performance for the sequential and linked tree. Because scenario 2 and scenario 3 will use the value size in ‘integral multiple', adding another 100 values will be used in the last part of all the codes in linkRepresentation.java and SeqRepresentation.java.
The second scenario,

Second scenario: After a scenario, one has been finished (except extra 100 nodes), an initialized tree (not changing) has been built. we randomly select an index from an array which stored where we just loaded from a file for generating the tree, and then, the time period of the function of FN, FP and FC in both sequential and linked will be tested the performance. Different sizes of values will be tested as well. (102~106)

Third scenario: Based on the first scenario (except extra 100 nodes), a tree has been initialized. Thus, the pre, in and post order methods will be evaluated in both data structure and estimate the time cost for each one in different value size. There are two folds named linkOrder and seqOrder that will show the results in this situation.

Evaluation

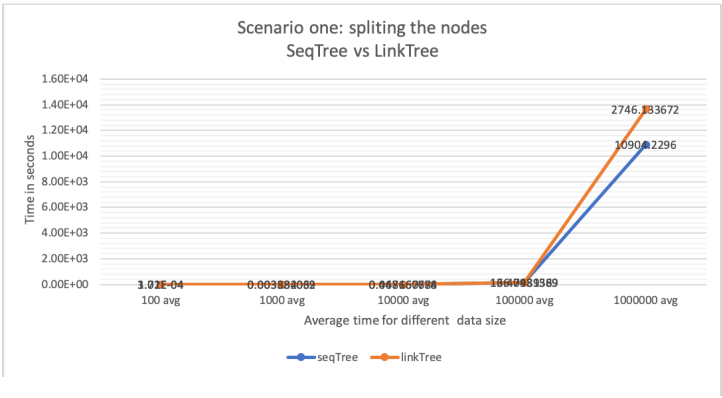
times to calculate average	100 times						50 times		3 times	
Initial tree	100_samples	100_samples	1000_samples	1000_samples	10000_samples	10000_samples	100000_samples	100000_samples	1000000_samples	1000000_samples
Data Structure	seqTree	linktree	seqTree	linktree	seqTree	linktree	seqTree	linktree	seqTree	linktree
SP	1.02E-04	3.71E-04	0.003384032	0.003832069	0.448660688	0.067167774	136.7939369	16.40481589	10904.2296	2746.133672
FN	2.66E-05	2.12E-05	1.77E-05	1.57E-05	2.16E-05	3.32E-05	0.0005334	4.82E-04	0.003430369	0.014034488
FP	1.80E-05	8.08E-06	1.51E-05	5.24E-06	3.73E-05	3.89E-05	0.00176	5.62E-04	0.009772509	0.011962969
FC	1.56E-05	7.17E-06	1.54E-05	5.74E-06	2.51E-05	3.38E-05	0.00098816	5.08E-04	0.009812452	0.013825073
TP	5.48E-05	4.69E-05	1.61E-04	2.21E-04	0.001087956	0.001272352	0.013578725	0.008516121	0.131598228	0.104354664
TI	2.93E-05	2.80E-05	1.09E-04	9.98E-05	9.29E-04	7.28E-04	0.011060357	0.006131223	0.120952577	0.072063431
TS	2.61E-05	3.40E-05	8.37E-05	9.94E-05	9.06E-04	7.39E-04	0.011148478	0.005611295	0.118618688	0.081381777
additional extra 100 nodes			3.56E-04	1.50E-04	0.005327893	0.001435433	0.143162387	0.042405672	1.451538389	0.278497013

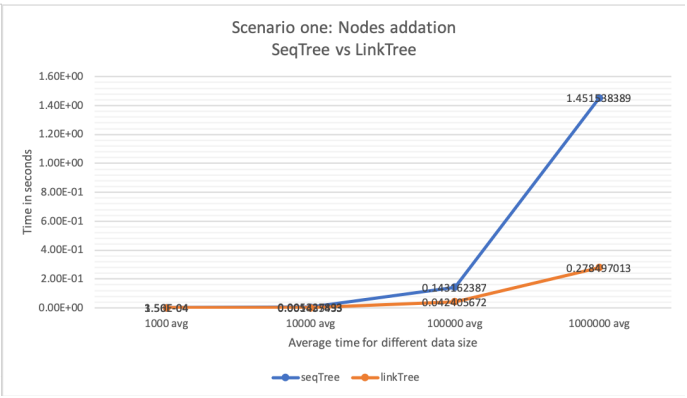
Above result is what we got.

Scenario 1

In this scenario, the node splitting is been tested by splitNode() function using random data, which is tested by different data sizes and calculating the average to compare them. According to both charts, we found that with the increase in the test size of seqtree and linktree there was no sufficient different (see Figure 1). We hypothesize the reason for this is that the storage allocation of building the structure is same, but we found that the cost of

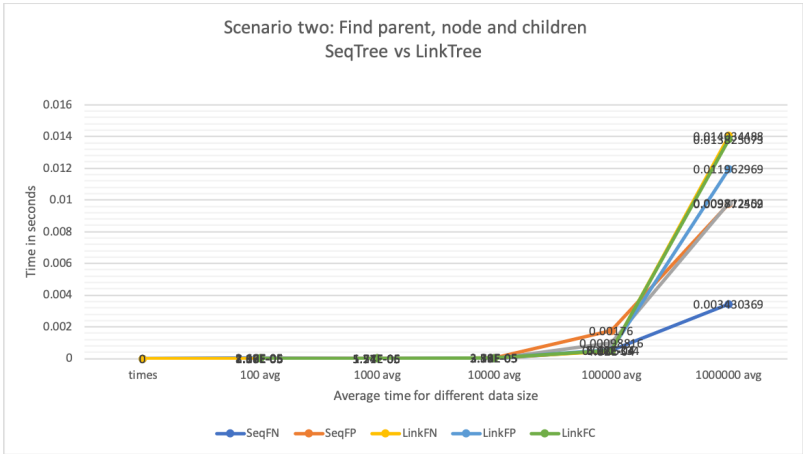
insertion in sequential tree required resizing the array for each splitting which effects on the time complexity compared to linktree which is better in terms of the insertion (Figure 2).





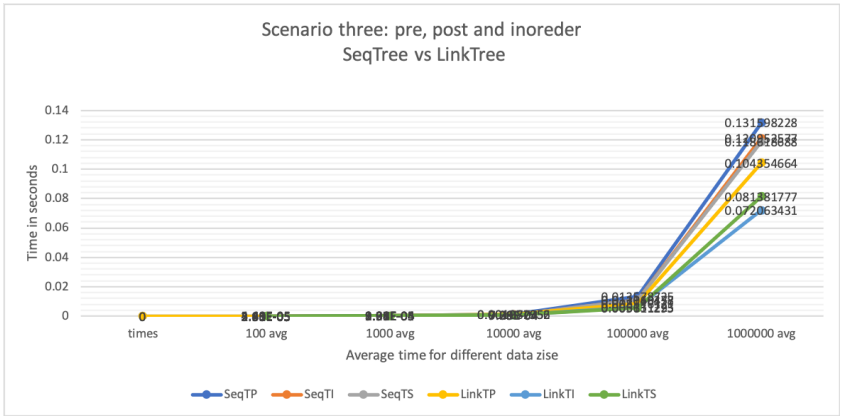
Scenario 2

For both sequential and linked the time required to search for a specific value is proportional to the size of the data. we analyze our results by using a random value for nodes and test the FN, FP and FC for each of them. we found that arrays are usually faster to iterate.



Scenario 3

As we see in the below figure we found that the linked response well to the functions of pre, post and inorder printing. We assume that the pointer of the left and right nodes makes the structure work faster for the traverses tree.



Recommendation

According to our results, we found that the linked representation response better to the scenario of ‘adding additional nodes’ (the situation is keep growing) in any part of the time.

Using sequential representation in case of searching for the particular value it doesn't cost time comparing to the linked structure.

linked representation works efficiently in the scenario of printing the whole value in the tree, which means it is better to access data in linked structure sequentially because each node has a pointer to the next rather than access it randomly.

Task C: Theoretical Complexity Analysis

C1. Complexities of tree operations

	O(1)	O(log(n))	O(n)	O(n!)
Find parent	No	No	Yes	No
Find a node	No	No	Yes	No
Print all nodes	No	No	Yes	No

The complexity of Both of them is O(n) in finding a parent, finding a node and printing all nodes.

1. Sequential Tree

In the implementation of SequentialReprensentation.java, all the data structures have been built in MyArray.java. (without using Node.class), all the operation is based on the relationship between the array index and a binary tree.

1.1 Finding parent:

Firstly, traverse the entire array to find the target node, and then based on the property with the array and binary tree to calculate the index of the parent. (targetIndex - 1)/2. The worst complexity is O(n) + O(constant), thus the complexity is O(n).

1.2 Finding a node:

To compare all the values of an array, it has to visit all the values to match the target value. The worst complexity happened in the last one each time, which is O(n).

1.3 Print all nodes:

Because values traversal needs to visit all the nodes one time, the complexity is O(n). (In seqOrder folder)

2. Linked Tree

The structure of a linked tree is built by LinkedRepresentaion.java and Node.java.

2.1 Finding parent:

As there is an instance variable named parent in Node.java, after finding the child of the target node, it is easy to find its parent. The first step is to find this node, thus the worst complexity should be the same as finding a node. It is O(n).

2.2 Finding a node:

This function was implemented by checking the left subtree and then, right subtree by recursion function, and each time, it needs to look for a tree from the root node. In the worst situation, a node is found in the last part. Thus, the complexity is O(n).

2.3 Print all nodes:

It needs to visit all nodes one time and then print it in print writer (In linkOrder folder). thus, the worst complexity is O(n).

C2. Guessing the age of an alien

1.1 The worst case is 14 billion questions because the astronaut asked age one by one.

1.2 Assume the age for an alien is between 1 to n. (n = 14 billion)

$C_{avg}(n) = C_{avg}(n) =$ =

$C_{avg}(n) = 7000000000.5$

Thus, in the average case, 7 billion and another 1 questions need to be asked.

1.3 Time(year) = seconds/60(s)/60(mins)/24(hours)/365.25(average years)

Time(worst case) = 443.63 years

Time(average case) = 221.82 years

2. The best strategy is to ask questions by ‘dichotomy', which asks aliens by dividing 2, and then call this method recursively according to results (yes or no). for example, Beginning from half value, 7 billion. If the answer is ‘yes', the 3.5 billion needs to be checked. Vice versa. And then based this number to create a binary tree. (similar to a binary search tree.)

$2^k = n$ (k = how many times need to ask, n = how many years)

$k = \log_2 n$, thus, in this situation, in the worst case, $\log_2 n$ needs to be asked.

Because the question is "are you at most X years?" and the answer is just yes or no, without other answers like "correct, that's my age". Thus, the number of asked questions has to arrive at the last one. Thus, there is no average case in this situation. (If it has the average case, the complexity for the average case is O(logn))

$$C(n) = C(n/2) + 1$$

$$C(1) = 0$$

$$n = 2^k$$

$C(2^k) = C(2^{k-1})+1$ $C(2^k) = k$ $C(n) = \log_2 n$

Thus, C(n) = 33.7, for this scenario, in the worst case and the average case, it needs to ask 34 questions.