PA3 Report III

John J. McCarthy

November 8, 2021

Question 1

Copy the `update_search_times` into the below listing (also copy in any recursive helper functions):

```
void helper_update(Node& root, int search_time) {
  if (&root == nullptr)
    return;
  root.search_time = search_time;
  helper_update(*root.left, search_time + 1);
  helper_update(*root.right, search_time + 1);
}
void BSTree::update_search_times()
{
    helper_update(*root, 1);
}
```

Analyze the time complexity of your `update_search_times` function. Give an asymptotic upper bound for the runtime on a tree with n nodes. Provide a mathematically sound justification for this bound. (Clarification: This question does **not** ask about the search complexity of the tree. Rather, it asks about the time complexity of the function which recalculates the auxiliary search costs.)

$$f(n) \in O(n) \tag{1}$$

The work done at each node is $O(1)$ and there are a total of n nodes. Hence, the work done in total is $O(n)$.

Question 2

In this question, you will analyze the average search cost of a binary tree. For the purposes of this assignment, the search cost is the number of distinct nodes the algorithm makes comparisons against during the search process. This means the search cost of an individual node is $d + 1$ where d is the depth of a node.

2.1 Linear Binary Tree

Find a formula for the average search cost of a linear binary tree in terms of the number of nodes in the tree n. Show every step of your derivation. (Note: You may wish to familiarize yourself with the closed form for an arithmetic series.)

$$g_l(n) = (\sum_{k=1}^{n} k) \div n \tag{2}$$

$$g_l(n) = \frac{n+1}{2} \tag{3}$$

2.2 Perfect Binary Tree

Find a formula for the average search cost of a perfect binary tree in terms of the number of nodes in the tree n. Show every step of your derivation. (Note: You may wish to familiarize yourself with a geometric series.)

$$g_p(n) = (\sum_{k=1}^{log_2(n+1)} k * 2^{k-1}) \div n \tag{4}$$

Using the summation of a finite geometric series formula:

$$g_p(n) = -(1 - 2^{log_2(n+1)}) \times (log_2(n+1))! \tag{5}$$

2.3 Experimental Verification

When you first downloaded the assignment, you were provided with serialized trees in the `data-files` directory. Each file beginning with numbers 1, 2, 3, $\cdots$, 12 contain $2^1 - 1$, $2^2 - 1$, $2^3 - 1$, $\cdots$, $2^{12} - 1$ integers respectively. The files with the p suffix contain integers which make 12 **perfect binary trees** where all the leaves have the same depth. Thus, the file named `3p` contains a perfect binary tree with $2^3 - 1 = 7$ nodes. The files with the r suffix contain randomly ordered integers which make 12 **random binary trees**. The files with the l suffix contain integers in increasing order which make 12 **linear binary trees.**

Create a plot with the following series:

- The average search time for a linear binary tree (found experimentally)
- The average search time for a perfect binary tree (found experimentally)
- The average search time for the random binary trees (found experimentally)
- The average search time for the linear tree you calculated in 2.1
- The average search time for a perfect binary tree you calculated in 2.2

Things to keep in mind:

- If you found the correct formulas in 2.1 and 2.2, they should provide the same values you calculated experimentally. You may wish to plot your theoretical formula as a continuous function so we can differentiate it from the series.
- It's important that your final plot is able to be easily interpreted and clearly shows the relative growth. You may wish to log the Y-axis.

Programmatically calculating search times for 2.3

The following code will print the depth, the average search cost of the linear binary trees, perfect binary trees, and then random binary trees in CSV form. (In that order.) The values can written to a file with `./run-trees > avg_costs.csv`. The resulting file can be imported into a spreadsheet application or read using Python.

```
#include <iostream>
#include <string>
#include <fstream>
#include "BSTree.h"

BSTree read_file(std::string file_name) {
    BSTree new_tree;
    std::ifstream infile(file_name);
    if (infile.fail())
        throw "Error while attempting to read file: " + file_name;
    infile >> new_tree;
    return new_tree;
}

int main() {
    try {
        std::string file;
        for(int d = 0; d < 12; d++) {
            file = "data-files/" + std::to_string(d + 1);

            BSTree l = read_file(file + "l");
            std::cout << d << ", " <<  l.get_average_search_time();

            BSTree p = read_file(file + "p");
            std::cout << ", " << p.get_average_search_time();

            BSTree r = read_file(file + "r");
            std::cout << ", " << r.get_average_search_time() << std::endl;
        }
    } catch (std::string& error) {
        std::cout << error << std::endl;
    }
}
```