Eyu Chen

CS124

Oct 20

Project3 Report

In this project, we still use the dataset used from the previous projects, which is a data file of uvm fall courses with 5842 records. This time using three tree structures to store the objects, which are binary search tree, AVL tree and splay tree. The 3 structures have their different abilities to store and manage data.

Data

The data file used is about all the fall courses in University of Vermont in 2021, the source is from the Office of Register https://serval.uvm.edu/~rgweb/batch/curr_enroll_fall.html, and was modified a little bit. There are 9 attributes in the .csv file and then become the fields for the class, some naming follows abbreviations used in CSV file:

- rowId: the row ID for ordering
- subject: the subject that the course belongs to
- title: the title of the course
- CRN: the registration number for this course
- college: the college code that the class belongs to
- maxEnrollment: the maximum enrollment of the course
- currEnrollment: the current enrollment of this course by now
- instructor: the professor's name of the course, in the format of FirstName, LastName
- email: the uvm email address of the professor

The original dataset is ordered by title over alphabet, and a column named rowId was added in the first place from 1 to 5422, and now the dataset is ordered by this.

Citation

- The methods to generate random sequences such as shuffle function are from http://www.cplusplus.com/reference/algorithm/random_shuffle/
- The methods to set timer are from https://en.cppreference.com/w/cpp/chrono/duration

Questions

1. Search 0, 101, 102 in a binary search tree
   When searching 0, the depth stops at 0, which is the root; and for 101 and 102, it stops at 99 which is the leaf level. When search for the numbers that are not in the binary search tree, the algorithm will give a reminder of not finding the values and return the depth it arrived finally. As we add values into a binary tree in order of 1-100, which means that the tree will root at 1 and go straight to the right until the leaf of 100, because every new value is greater than the last one and would be the right child of the last node.

2. Comparison between BST in random and in order
   A common BST has no ability to rebalance the nodes, thus when inserting in order, the number of depths equals to the number of objects. When the insertion is in random, the maximum depth is much smaller than that of the previous one as the nodes in this tree possibly would have their left children, which makes the tree shorter.

   We also have a BST storing Course objects in order, the result is as the same as what we had in the first integer BST. The reason is also the same, the BST does not rebalance itself after insertion.

3. Comparison between AVL trees and BST
   AVL tree has ability to rebalance itself after each insertion to guarantee that no node in this tree will have 2 subtrees that the difference between their depths is greater than 1. From the depth values returned by find method of AVL tree class, it's clearly that the maximum depth in 2 AVL trees are both less than 10 and the value is around 6 or 7. The reason is that the ability to rebalance allows AVL tree to minimize the maximum depth of the structure, while BST structure will be easier to gain bigger value of depth.

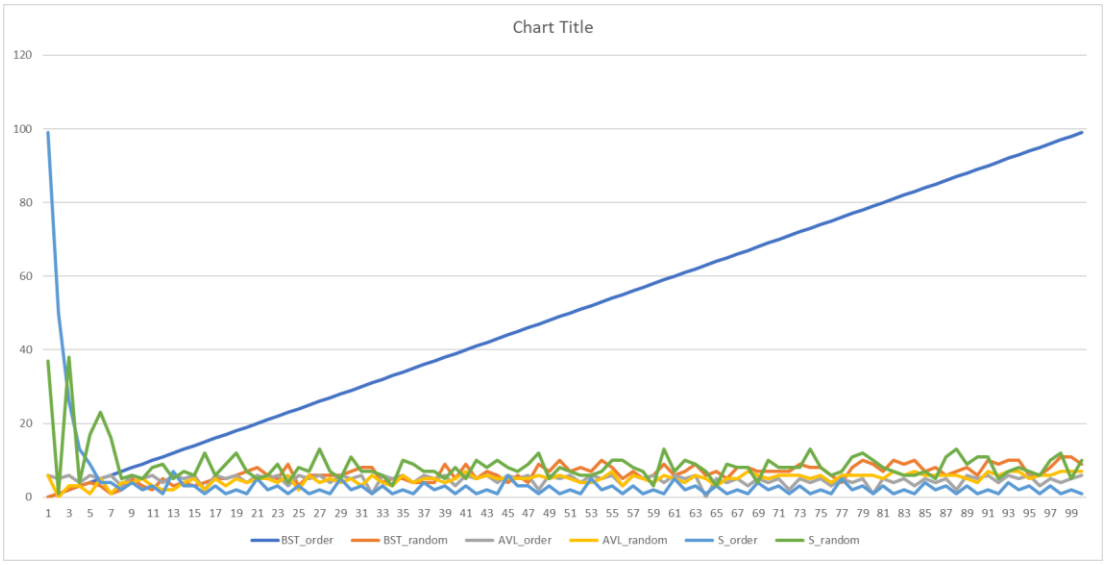
   For the AVL tree of Course objects, it is clearly that the depth values of AVL tree are much less than those of BST due to the rebalance operation, and unlike the linear relationship in BST, the distribution of depth values in AVL tree is more like an upside-down complete binary search tree whose root is at the bottom.

4. Comparison among splay tree and other 2 types of trees
   The goal of a splay tree is to splay the element that is visited recently to the root, but it rebalances not after insertion but search operation. Thus, a splay tree that was inserted in order will likely look like the first BST. In the splay tree with insertion in order, the depths returned by find method in splay class show that the depth of each node is getting smaller in general, which implies that for every integer, as the last one is visited and splayed, its nearby values might move to a lower depth and be close to the root. For the random insertion tree, the values are obviously larger than those of the other one, however, the graph shows that both have a similar trend when searched number increases.

   The comparison among 3 trees of Course object also indicates the difference among the three structures. Like the trend in integer tree, the depth values in Course splay tree are decreasing to a nearly stable value. Then we search objects in a random sequence and search for 5 times for each object, record the values returned by every find operation – in the first search, the returned depths also gain a similar trend like others, but it clearly that a few of them have been splayed to the root, however from the second iteration, their depths all turn into 0, which is the root, which satisfies the goal of splay tree.

Graph Analysis

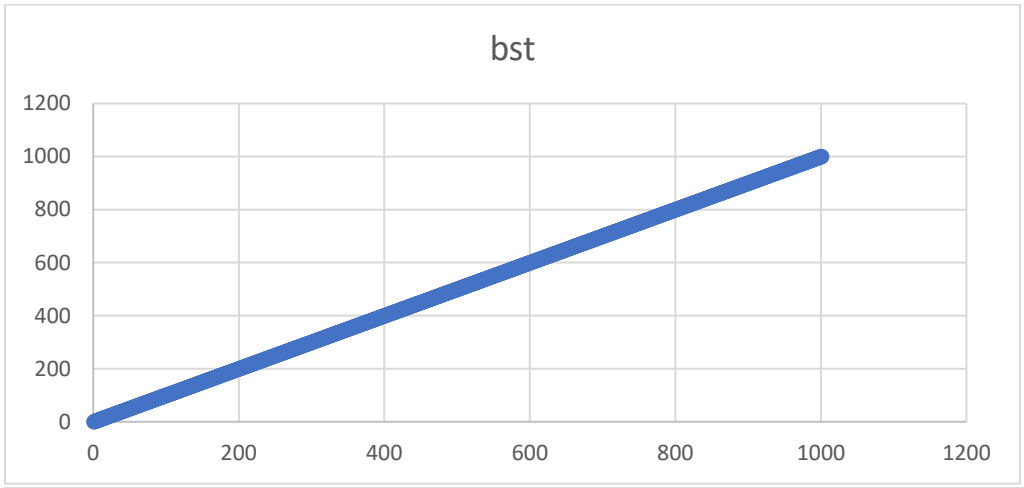
1. Trees of integers

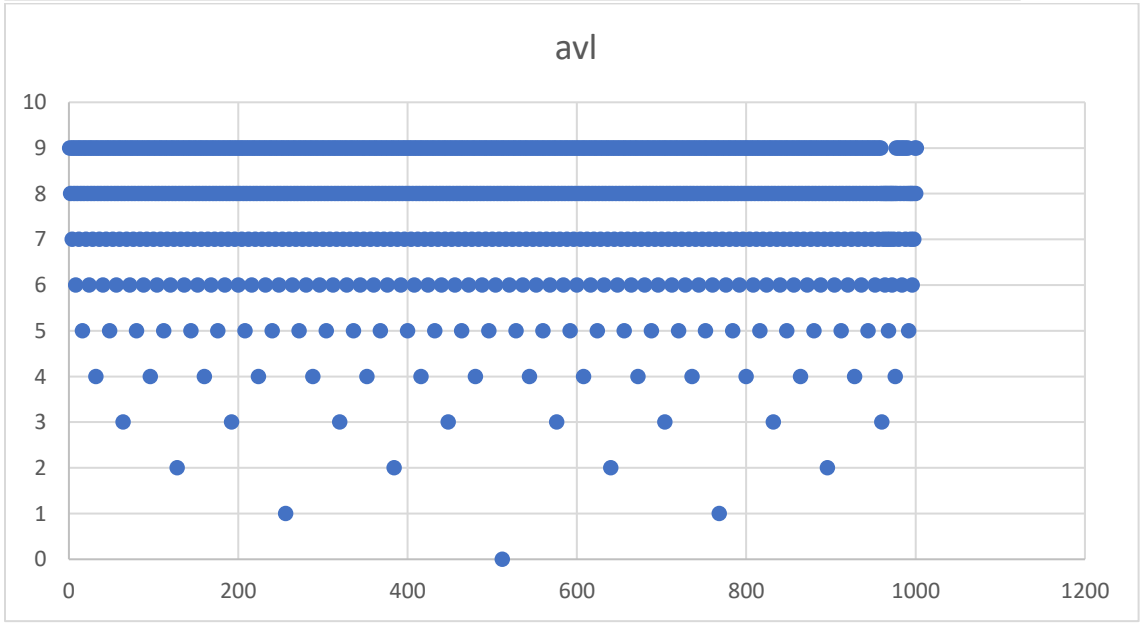


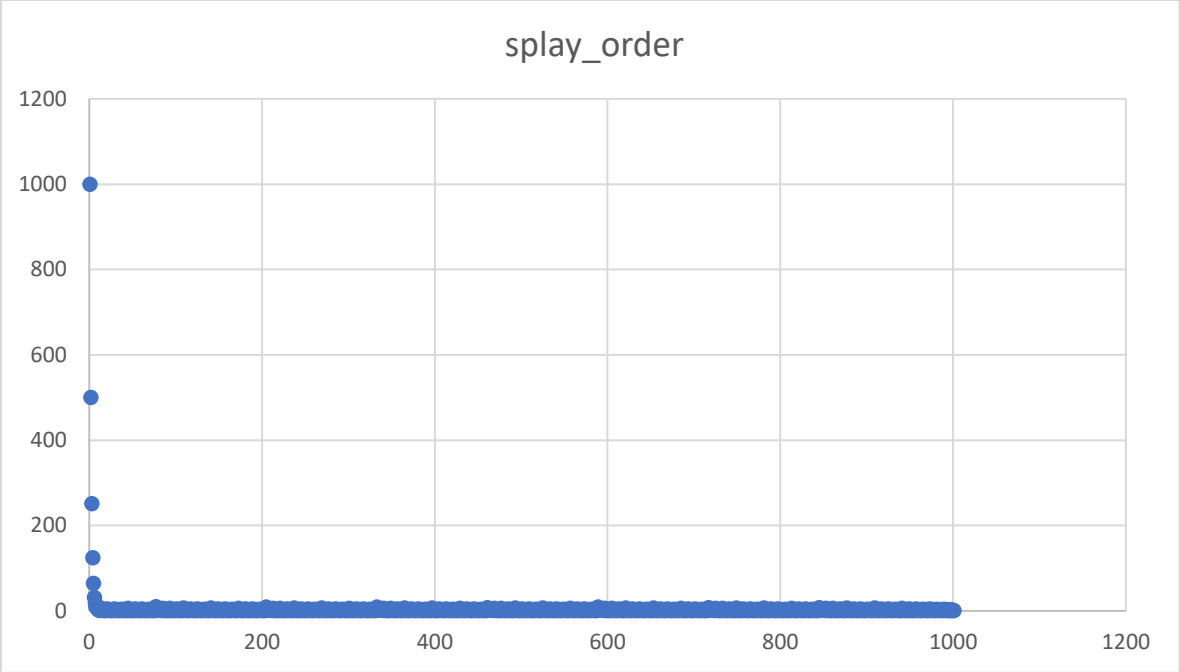
   - Binary Search Tree: the first tree inserts integers in order and the depth is a sharp line because every new value is the right child of the last one, which means that in this tree every node only has right child. When inserting in random, the tree will possibly have the left subtrees and the maximum depth decreases.

- AVL Tree: in a avl tree, the difference of 2 subtrees of a node is less than or equal to 1, thus no matter the insertion is in order or in random, the rebalance operation will make the tree a nearly complete BST, and the line of depth values is nearly flat compared to the others.
- Splay Tree: a splay tree will splay the node visited before to the root, and like BST, it is fine if the structure of the tree turns into an extreme shape. In the lines of splay tree, it is clearly that the depth decreases quickly when searching the first few elements and becomes stable after that. The reason is that splay operation will move the visited nodes to the top, and in the tree with insertion in order, the nearby nodes, which is also nearby in values, of the visited node might move to a lower level in splaying; but in the tree with insertion in random, this rule may not work.

2. Trees of Course objects







- These 3 graphs imply the similar conclusion to the integer ones. All of them are with insertion in order, and BST gives a linear relationship, avl tree is plotted as an upside-down complete BST, while the graph of splay tree shows that it moves the nodes to the root or lower level.

Complexity of Search method

- Binary Search Tree
  - Time Complexity: O(log n) or O(n)
  - Auxiliary Complexity: O(1)
  - Space Complexity: O(1)
- AVL Tree
  - Time complexity: $O(\log_2 n)$
  - Auxiliary complexity: O(1)
  - Space complexity: O(1)
- Splay Tree
  - Time complexity: O(log n)
  - Auxiliary complexity: O(1)
  - Space complexity: O(1)

Timer

In this project, we use methods and functions from chrono to set timer and count the duration of search operation of 3 types of integer trees. The code should return the duration in microseconds and show message in the command line.