ADS Project-1
Binary Search Trees

Author Names

Date: 2019-03-03

Chapter 1: Introduction

We have learnt binary search tree, AVL tree and splay tree, which use three different ways to implement operations on trees. To compare which one is better, we may respectively insert distinct integers in increasing order and delete them in the same order, insert distinct integers in increasing order and delete them in the reverse order and insert distinct integers in random order and delete them in random order. Analyzing their time cost, we will get to know the differences.

Chapter 2: Data Structure / Algorithm Specification

Part one Binary search tree

```
/*BST Functions
 1.Insert an element
 2.Find the minimum element
 3.Delete the element
 4.Level-order Traverse
 */
struct BSTNode;
typedef BSTNode* BSTTree;
typedef BSTNode* PosBST;
BSTTree BSTInsert(int element, BSTTree T);
PosBST BSTFindMin(BSTTree T);
BSTTree BSTDelete(int element, BSTTree T);
void LevelOrder_BSTTree(BSTTree T,int number_Nodes);
struct BSTNode
{
    int element;
    BSTTree Left;
    BSTTree Right;
};
```

Use a fundamental BST data structure to represent the binary search tree. A BST node has three components: its element and two pointers pointing its left child and right child.

Create and return a one-node Tree. Recursively Insert when insertion number smaller. Recursively Insert when insertion number larger. Nothing has been done when x is already in the Tree. After deleting, find the right position of its parent and children respectively.

Part two AVL tree

```
/*AVL Functions
1.Both single and double rotation
2.Level order traversal
3.Get the height of tree
4.Get the Maximum of two numbers
5.AvlNode Definition
6.Delete Node
7.Level Order Traverse
 */
struct AvlNode;
typedef struct AvlNode *PosAvl;
typedef struct AvlNode *AvlTree;
AvlTree AVLInsert(int element,AvlTree T);//Insert an element into the Tree
AvlTree AvlFindMin(AvlTree T);
AvlTree AVLDelete(int element, AvlTree T);
PosAvl SingleRotateWithLeft(AvlTree T);
PosAvl DoubleRotateWithLeft(AvlTree T);
PosAvl SingleRotateWithRight(AvlTree T);
PosAvl DoubleRotateWithRight(AvlTree T);
void LevelOrder_AvlTree(AvlTree T, int number_Nodes);
static int HeightAvl(PosAvl P);
int Maximum(int a,int b);
//structs Define
struct AvlNode
{
    int element;//element
    AvlTree Left;//Pointer to the left root
    AvlTree Right;//Pointer to the right root
    int height;//the height of the Tree
};
```

AVL tree data structure is almost as same as the BST, despite that it has another component documenting the height of the node. Then insert one by one, and complete AVL tree through rotation.

pseudo-code:

```
AvlTree AVLInsert(int element,AvlTree T)
{
if(T == NULL)
```

```
4.  Situation One: Create One node Tree
5.      else if(element < T->element)
6.      {
7.          Insert recursively;
8.          if(HeightAvl(T -> Left) - HeightAvl(T->Right) == 2)
9.              adding the left Node can only make the left larger than right
10.             if(element < T->Left->element)
11.                 Single Rotate With Left;
12.             else
13.                 Double Rotate With Left;
14.     }
15.     else if(element > T->element)
16.     {
17.         T -> Right = AVLInsert(element, T->Right);
18.         if(HeightAvl(T->Right) - HeightAvl(T->Left) == 2)
19.             adding the right Node can only make the right larger than left
20.             if(element > T ->Right->element)
21.                 Single Rotate With Right;
22.             else
23.                 Double Rotate With Right;
24.     }
25.     Nothing has been done when the node is already in the Tree.
26. }
```

Delete the specific element in the AVLTree may well have two situations: deleting the left node may cause the height change beyond the range, then the situation may be similar to inner Left insert to the Left Child or similar to the situation that outer Right insert to the Right Child; deleting the right node may cause the height change beyond the range, which is quite similar to the situation that inner Left insert to the Left child, or similar to the situation that outer Right insert to the Left child. That is to say, after deleting a specific node in the tree, we can mend the AVL tree through rotating.

Part three Splay tree

```
1.  /*Splay Functions
2.   1.Insert an element
3.   2.Initialize the NULLNode
4.   3.Delete Functions
5.   4.Splay Function
6.   */
```

```
7.  typedef AvlNode SplayNode;
8.  typedef SplayNode* SplayTree;
9.  SplayTree SplayInsert(int element,SplayTree T);
10. SplayTree SplayDelete(int element,SplayTree T);
11. SplayTree Splay(int element,SplayTree T);
12. static AvlTree NULLNode = NULL;//Make the program run faster
13. SplayTree Initialize(void);
14. void LevelOrder_SplayTree(SplayTree T,int number_Nodes);
```

The data structure used in the splay tree is as same as the AVL tree. AvlNode is equivalence to SplayNode in this scenario. Splay is a operation that adjust the structure of the tree and reposition a specific element to the root of the tree using zig and zag which is similar to the rotation operation of the AVL tree.

After each node is inserted, splay it. When it comes to deleting, first find the element. If it does not have any child or have only one, delete it and splay its parent node. Or find the largest node of its left child and splay it. Then delete the element required.

Chapter 3: Testing Results

1. Necessary inputs for testing

We have already provided the input document (test data) in our document, which contains the size of data from 1000 to 10000, and you can copy these data to the demands. And our input format is here:

【The size of data------Input sequence------Delete sequence】

in these three document:

demand1.txt : increasing order

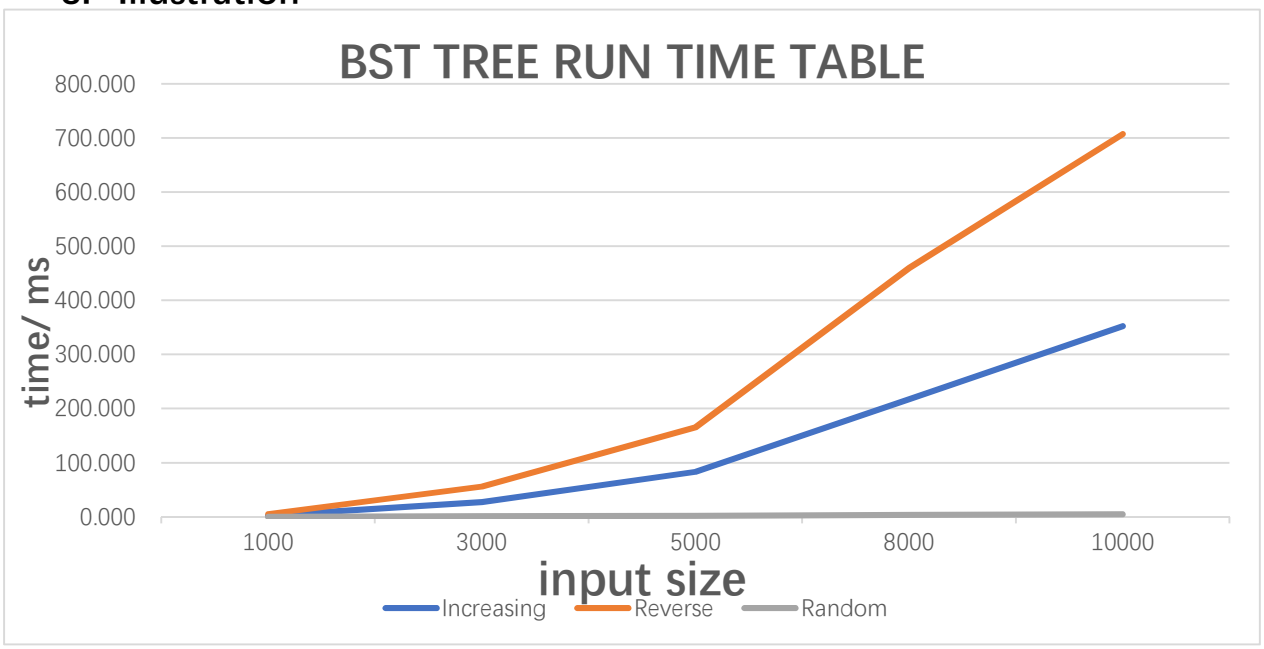
demand2.txt : reverse order

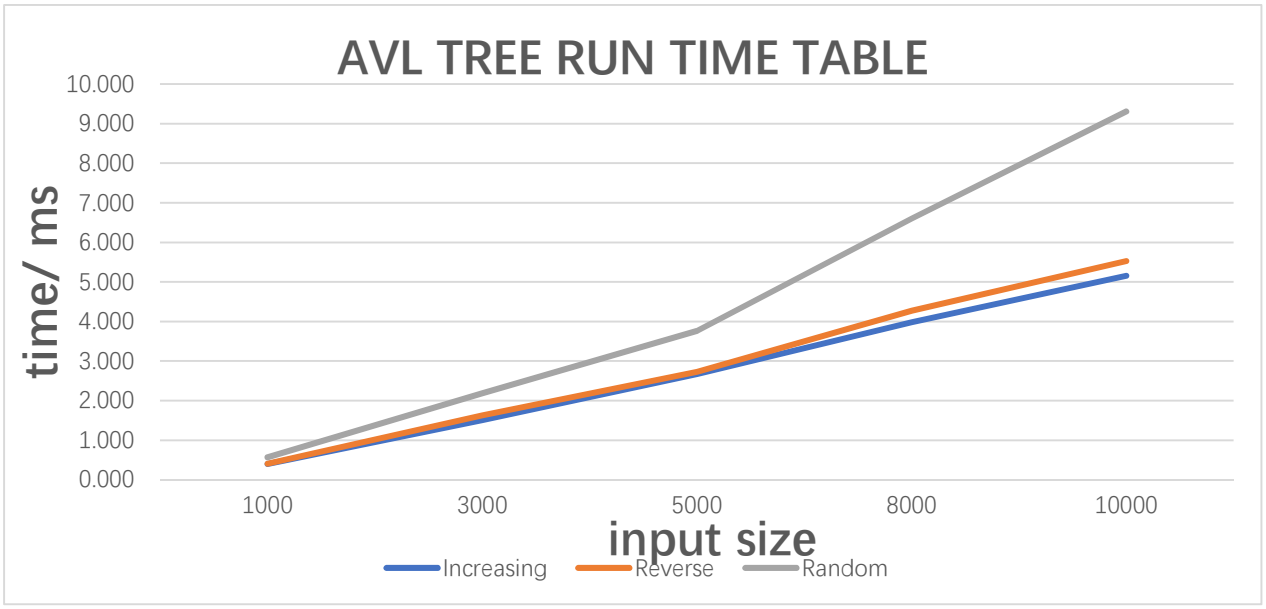
demand3.txt : random order

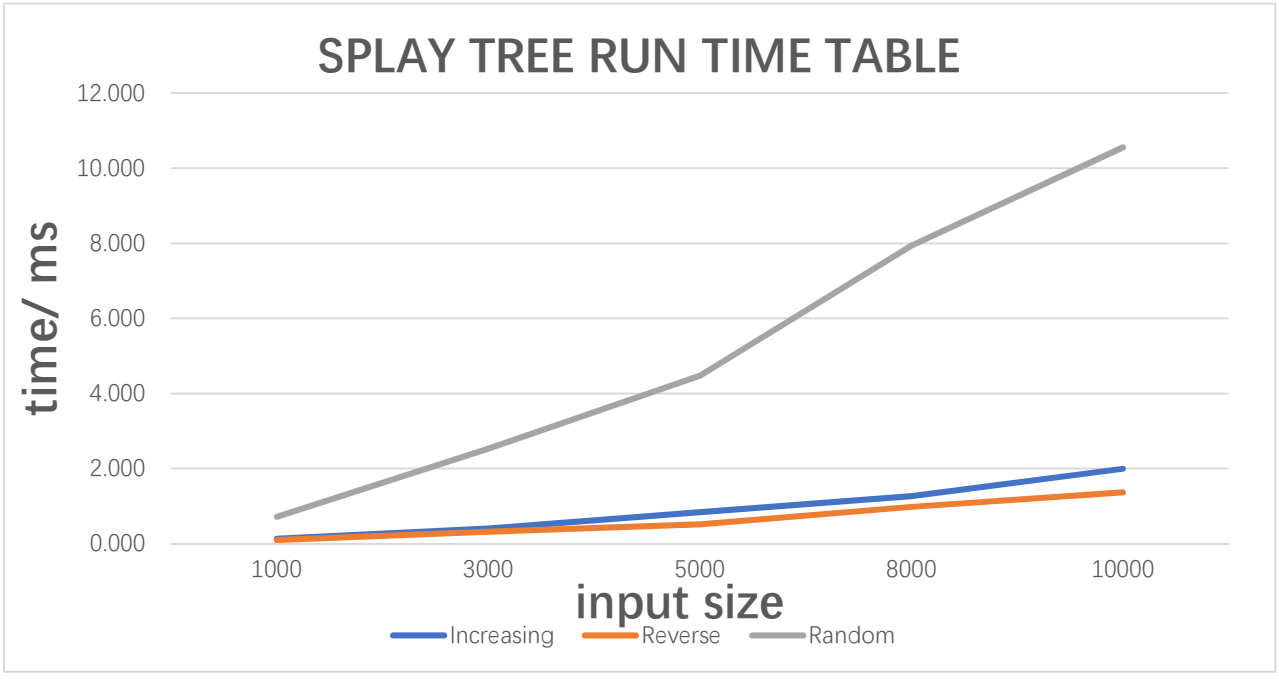
2. Run time table

BST TREE RUN TIME TABLE (**Unit: millisecond**)					
Size / **Order**	1000	3000	5000	8000	10000
Increasing	2.712	27.600	83.255	217.970	352.360
Reverse	5.654	56.434	165.630	460.020	707.500
Random	0.275	1.158	2.103	3.390	4.690
AVL TREE RUN TIME TABLE (**Unit: millisecond**)					
Size / **Order**	1000	3000	5000	8000	10000
Increasing	0.399	1.509	2.673	3.990	5.160
Reverse	0.407	1.635	2.730	4.280	5.530
Random	0.569	2.187	3.765	6.610	9.310
SPLAY TREE RUN TIME TABLE (**Unit: millisecond**)					
Size / **Order**	1000	3000	5000	8000	10000
Increasing	0.133	0.402	0.840	1.260	2.000
Reverse	0.099	0.317	0.510	0.980	1.370
Random	0.719	2.531	4.480	7.940	10.560

3. Illustration







Chapter 4: Analysis and Comments

By observing the time table and the illustration of the input size and the run time, we can easily find there these three kinds of tree have advantages of itself.

When we want to insert and delete the increasing order sequence and the reverse order sequence into the tree, we had better choose the splay tree because of its feature which can make it quickly insert and delete the data.

When we want to insert and delete the random order sequence into the tree, we had better choose the BST tree because of its feature which can make it quickly insert and delete the data.

Maybe you are considering that we don't involve AVL tree, indeed, when we have confirmed the sequence, we can choose other two kinds of tree, but most of time we input sequences which we can't confirm. The AVL tree are the medium of these three tree in all kinds of situation. And other two kinds of tree have their bad performance in some kinds of situation.

Author List

Programer:

Tester:

Reporter:

Declaration

We hereby declare that all the work done in this project titled " Binary Search Trees" is of our independent effort as a group.

Signatures