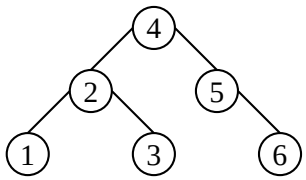
CS29003 Algorithms Laboratory

Assignment No: 7

Date: 10–March–2020

Binary Search Trees

Let $a_0, a_1, a_2, \ldots, a_{n-1}$ be distinct keys inserted in an initially empty binary search tree (BST) T in the given sequence. The standard BST insertion procedure is followed. In particular, no attempt is made to balance the tree. Any permutation of these keys, the inserion of which generates exactly the same BST is said to be *isogenic* to the given sequence. The following figure shows that there are exactly 20 isogenic sequences for the given BST. This assignment deals with counting and generating all sequences isogenic to a given sequence $A = (a_0, a_1, a_2, \ldots, a_{n-1})$ supplied by the user. Assume that a_i are distinct positive integers.



Left Subtree	Right Subtree
2,1,3	5,6
2,3,1	

Entire tree

4,2,1,3,5,6	4,2,3,1,5,6
4,2,1,5,3,6	4,2,3,5,1,6
4,2,5,1,3,6	4,2,5,3,1,6
4,5,2,1,3,6	4,5,2,3,1,6
4,2,1,5,6,3	4,2,3,5,6,1
4,2,5,1,6,3	4,2,5,3,6,1
4,5,2,1,6,3	4,5,2,3,6,1
4,2,5,6,1,3	4,2,5,6,3,1
4,5,2,6,1,3	4,5,2,6,3,1
4,5,6,2,1,3	4,5,6,2,3,1

Part 1: Counting isogenic sequences

For $n \leqslant 2$, the given sequence is the only sequence that can generate the same BST. So take $n \geqslant 3$. The first key a_0 in the sequence must be the root of the BST. Separate out two subsequences of $a_1, a_2, a_3, \ldots, a_{n-1}$. The first subsequence $A_L = (a_{i_1}, a_{i_2}, \ldots, a_{i_l})$ consists of keys smaller than a_0 (we must have $i_1 < i_2 < \cdots < i_l$). This subsequence generates the left subtree of the root. The second subsequence $A_R = (a_{j_1}, a_{j_2}, \ldots, a_{j_r})$ (with $j_1 < j_2 < \cdots < j_r$) consists of keys larger than a_0, and is responsible for generating the right subtree of the root. These subsequences are of size l and r, respectively. We have $l + r = n - 1$.

Recursively compute the counts $iso(A_L)$ and $iso(A_R)$ of sequences isogenic to A_L and A_R, respectively. The number of sequences isogenic to A is then given by the formula

$$iso(A) = iso(A_L) \times iso(A_R) \times \binom{l+r}{l} = iso(A_L) \times iso(A_R) \times \binom{n-1}{l}.$$

In order to see why, take any sequence B_L isogenic to A_L, and any sequence B_R isogenic to A_R. Construct any sequence C of length $l + r$, in which B_L and B_R are subsequences. Then, a_0 followed by C is a sequence isogenic to A. C is uniquely specified by the choice of l out of $l + r$ positions to be filled by the keys in B_L. The remaining r positions are filled by the keys in B_R.

Write a recursive function $countseq(A, n)$ to compute and return the count $iso(A)$ using the ideas mentioned above. The function must not create any BST, but play only with subsequences.

Part 2: Finding all isogenic sequences

Write a recursive function $findallseq(A, n)$ that, in addition to the count $iso(A)$, generates and returns all sequences isogenic to A. Again, do not create any BST, but follow the ideas given in Part 1. In order to construct the stitched sequences C from B_L and B_R, let a variable t run through integer values in the range $[0, 2^{l+r} - 1]$. Consider the $(l + r)$-bit binary expansion of t. If t contains exactly l zero bits and exactly r one bits, insert the keys from B_L at the zero-bit positions, and the keys from B_R at the one-bit positions. If t is not of this form, discard it.

Part 3: BST functions

Write the following functions.

> *BSTins*(T,x) to insert a key x in a BST T.
>
> *BSTcons*(A,n) to return a BST T generated by inserting in an empty tree the n keys in the array A (in the sequence specified by the array).
>
> *BSTprn*(T) to print the preorder and inorder listings of the keys in a BST T. You need two other functions for generating these two listings.
>
> *BSTsame*(T_1,T_2) to check whether two BST's T_1 and T_2 (may be assumed to be node-disjoint) are identical (same tree structure storing the same keys).
>
> *BSTfree*(T) to recursively free the memory allocated to the nodes of a BST T.

Part 4: Verifying isogenic property

In the *main* function, you generate a BST T from the sequence A supplied by the user. This tree will be used for comparing with other trees. Consider the list L of isogenic sequences, returned by *findallseq* of Part 2. For each such sequence, generate a fresh BST T', and verify whether T and T' are the same BST by calling *BSTsame*. After each check, call *BSTfree* on T'. Report how many sequences generated in Part 2 are actually not isogenic to A. If your implementation of Part 2 is correct, there will be no such non-isogenic sequence. Write a function *checkall*(T,L) to implement this check (pass other appropriate size parameters to this function).

The *main*() function

- The user enters $n, a_0, a_1, a_2, \ldots, a_{n-1}$. Store the keys a_i in an array A in the same order as supplied by the user. Do not construct any BST now.
- Call *countseq* on A, and report the count of sequences isogenic to A.
- Call *findallseq* on A to get a list L of all sequences isogenic to A. Print each sequence of L.
- Call *BSTcons* on A to generate a BST T from the keys stored in A. Print T by calling *BSTprn*.
- Call *checkall* to verify the correctness of your implementation of Part 2.

Submit a single C/C++ source file. Do not use global/static variables.

Sample output

```
6
661  615  695  573  675  652

+++ Sequence count
    Total number of sequences = 20

+++ All sequences
    Sequence 1     : 661  695  675  615  652  573
    Sequence 2     : 661  695  615  675  652  573
    Sequence 3     : 661  615  695  675  652  573
    Sequence 4     : 661  695  615  652  675  573
    Sequence 5     : 661  615  695  652  675  573
    Sequence 6     : 661  615  652  695  675  573
    Sequence 7     : 661  695  615  652  573  675
    Sequence 8     : 661  615  695  652  573  675
    Sequence 9     : 661  615  652  695  573  675
    Sequence 10    : 661  615  652  573  695  675
    Sequence 11    : 661  695  675  615  573  652
    Sequence 12    : 661  695  615  675  573  652
    Sequence 13    : 661  615  695  675  573  652
    Sequence 14    : 661  695  615  573  675  652
    Sequence 15    : 661  615  695  573  675  652
    Sequence 16    : 661  615  573  695  675  652
    Sequence 17    : 661  695  615  573  652  675
    Sequence 18    : 661  615  695  573  652  675
    Sequence 19    : 661  615  573  695  652  675
    Sequence 20    : 661  615  573  652  695  675

+++ BST constructed from input array
    Preorder       : 661  615  573  652  695  675
    Inorder        : 573  615  652  661  675  695

+++ Checking all sequences
    All trees match
```