COSE213 Data Structures (Fall 2021)

Assignment 4: Binary Search Trees

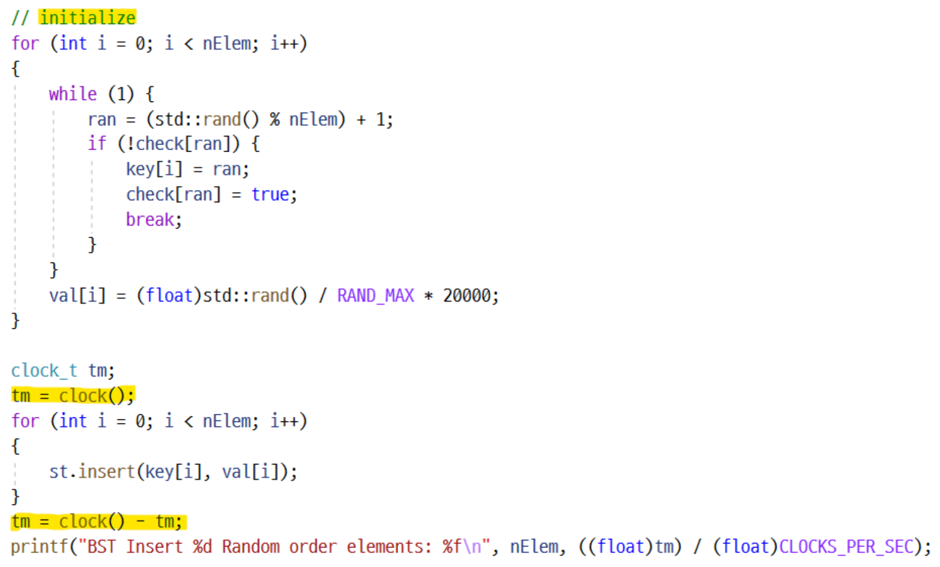
2021320322 데이터과학과 윤민서

This report is written to check the performance difference between Binary Search Trees and AVL Trees. To check the performance difference, the following experiments were conducted.

(a) insert operations only
(b) find operations only

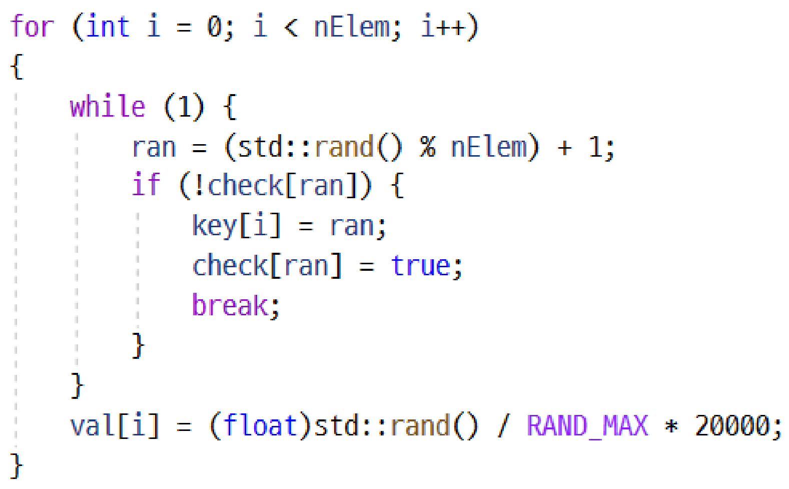
For this test, two types of test sets (keys are in skewed or random orders) with different sizes (1000, 10000, and 100000) are generated. The key type is unsigned int, and the value type is float, and random float number is assigned as a value for a given key. (a) and (b) experiments using the generated test sets were executed, and measured the running time using clock(). For experiments (b), this program was started with a search tree that was already filled with a test set from experiment (a).

For accurate timing measurement, random datasets were created before running the experiments. That means, the running time of insert/find operations was measured only and was not included the time for initializing the datasets.
Example code:



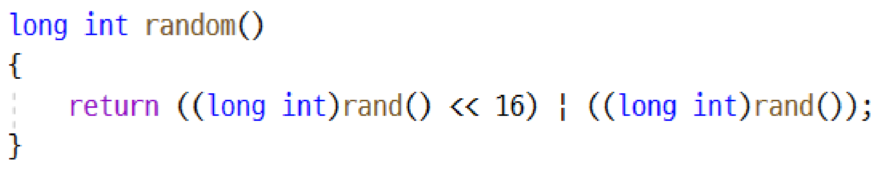
```
// initialize
for (int i = 0; i < nElem; i++)
{
    while (1) {
        ran = (std::rand() % nElem) + 1;
        if (!check[ran]) {
            key[i] = ran;
            check[ran] = true;
            break;
        }
    }
    val[i] = (float)std::rand() / RAND_MAX * 20000;
}

clock_t tm;
tm = clock();
for (int i = 0; i < nElem; i++)
{
    st.insert(key[i], val[i]);
}
tm = clock() - tm;
printf("BST Insert %d Random order elements: %f\n", nElem, ((float)tm) / (float)CLOCKS_PER_SEC);
```

In the cases of 1000 and 10000 numbers, random numbers of the key are generated according to the following code. The condition is also included to avoid duplication between key numbers.



```
for (int i = 0; i < nElem; i++)
{
    while (1) {
        ran = (std::rand() % nElem) + 1;
        if (!check[ran]) {
            key[i] = ran;
            check[ran] = true;
            break;
        }
    }
    val[i] = (float)std::rand() / RAND_MAX * 20000;
}
```

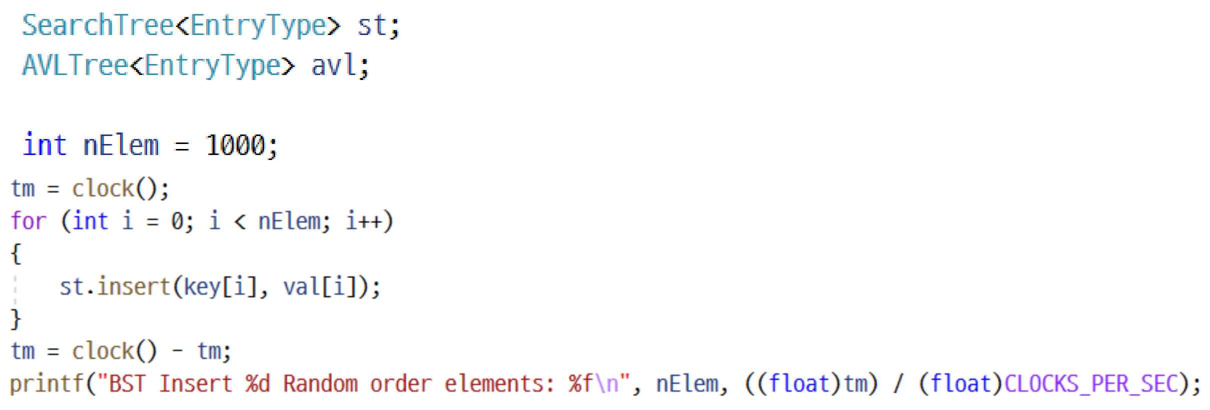
In the case 100000 numbers, one function is newly defined because the max number generated by rand() is 32767.



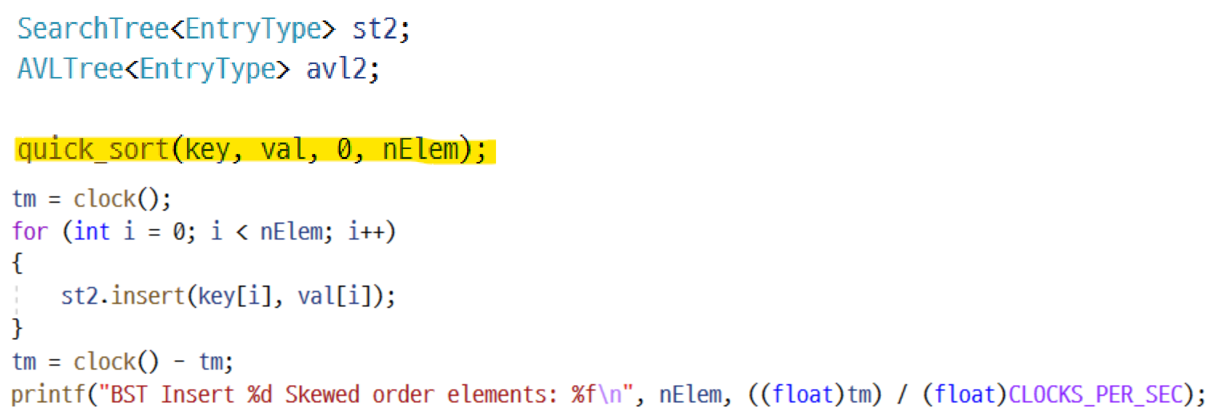
```
long int random()
{
    return ((long int)rand() << 16) | ((long int)rand());
}
```

For cases of the same size, the test was first conducted on a tree inserting a key array in which randomly generated numbers entered one after another. After that, the keys were sorted, and the test was conducted.

Example Code:

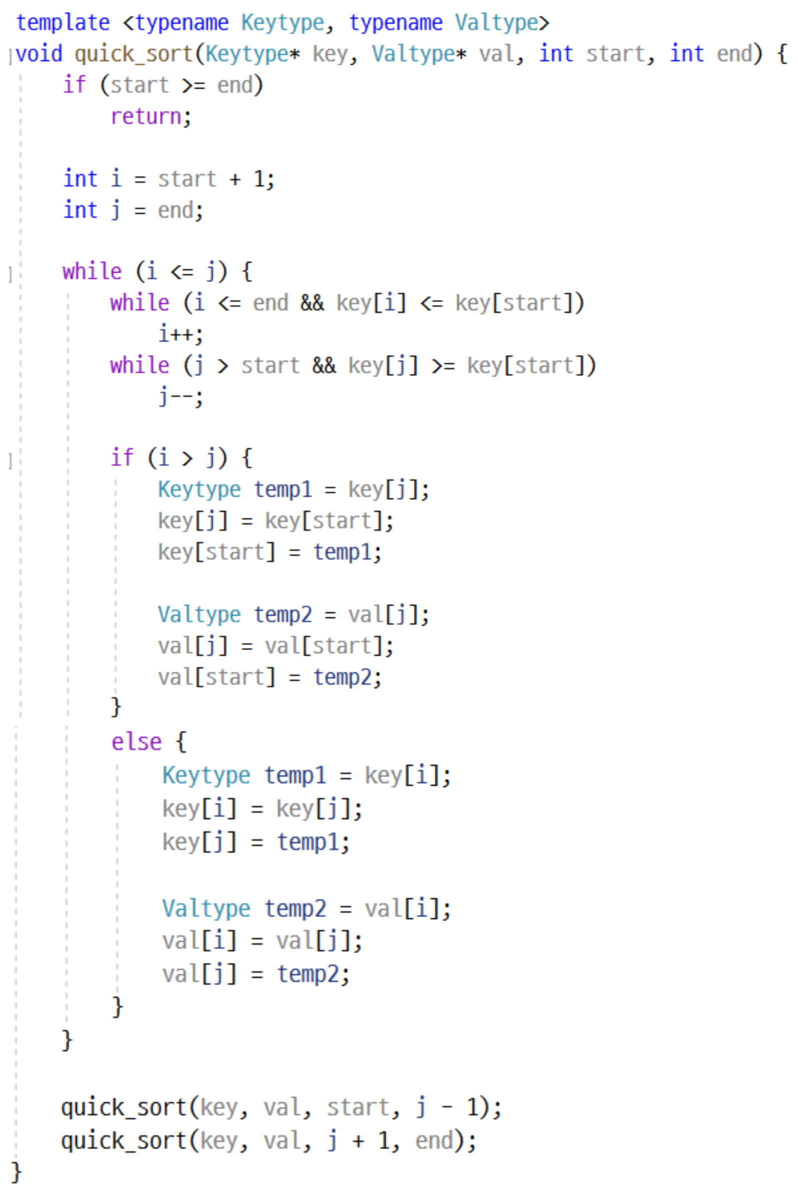


```
SearchTree<EntryType> st;
AVLTree<EntryType> avl;

int nElem = 1000;
tm = clock();
for (int i = 0; i < nElem; i++)
{
    st.insert(key[i], val[i]);
}
tm = clock() - tm;
printf("BST Insert %d Random order elements: %f\n", nElem, ((float)tm) / (float)CLOCKS_PER_SEC);
```



```
SearchTree<EntryType> st2;
AVLTree<EntryType> avl2;

quick_sort(key, val, 0, nElem);

tm = clock();
for (int i = 0; i < nElem; i++)
{
    st2.insert(key[i], val[i]);
}
tm = clock() - tm;
printf("BST Insert %d Skewed order elements: %f\n", nElem, ((float)tm) / (float)CLOCKS_PER_SEC);
```

The sorting method used is quick sort, and the code of the function is as follows.



```
template <typename Keytype, typename Valtype>
void quick_sort(Keytype* key, Valtype* val, int start, int end) {
    if (start >= end)
        return;

    int i = start + 1;
    int j = end;

    while (i <= j) {
        while (i <= end && key[i] <= key[start])
            i++;
        while (j > start && key[j] >= key[start])
            j--;

        if (i > j) {
            Keytype temp1 = key[j];
            key[j] = key[start];
            key[start] = temp1;

            Valtype temp2 = val[j];
            val[j] = val[start];
            val[start] = temp2;
        }
        else {
            Keytype temp1 = key[i];
            key[i] = key[j];
            key[j] = temp1;

            Valtype temp2 = val[i];
            val[i] = val[j];
            val[j] = temp2;
        }
    }

    quick_sort(key, val, start, j - 1);
    quick_sort(key, val, j + 1, end);
}
```

The test sequence is as follows.

1. 1,000 elements, random order
   a) The time it took to insert 1,000 randomly generated elements into the binary search tree.
   b) The time it took to find 1,000 randomly generated elements into the binary search tree.
   c) The time it took to insert 1,000 randomly generated elements into the AVL tree.
   d) The time it took to find 1,000 randomly generated elements into the AVL tree.

2. 1,000 elements, skewed order
   a) The time it took to insert 1,000 sorted elements into the binary search tree.
   b) The time it took to find 1,000 sorted elements into the binary search tree.
   c) The time it took to insert 1,000 sorted elements into the AVL tree.
   d) The time it took to find 1,000 sorted elements into the AVL tree.

3. 10,000 elements, random order
   a) The time it took to insert 10,000 randomly generated elements into the binary search tree.
   b) The time it took to find 10,000 randomly generated elements into the binary search tree.
   c) The time it took to insert 10,000 randomly generated elements into the AVL tree.
   d) The time it took to find 10,000 randomly generated elements into the AVL tree.

4. 10,000 elements, skewed order
   a) The time it took to insert 10,000 sorted elements into the binary search tree.
   b) The time it took to find 10,000 sorted elements into the binary search tree.
   c) The time it took to insert 10,000 sorted elements into the AVL tree.
   d) The time it took to find 10,000 sorted elements into the AVL tree.

5. 100,000 elements, random order
    a) The time it took to insert 100,000 randomly generated elements into the binary search tree.
    b) The time it took to find 100,000 randomly generated elements into the binary search tree.
    c) The time it took to insert 100,000 randomly generated elements into the AVL tree.
    d) The time it took to find 100,000 randomly generated elements into the AVL tree.

6. 100,000 elements, skewed order
    a) The time it took to insert 100,000 sorted elements into the binary search tree.
    b) The time it took to find 100,000 sorted elements into the binary search tree.
    c) The time it took to insert 100,000 sorted elements into the AVL tree.
    d) The time it took to find 100,000 sorted elements into the AVL tree.

The results are as follows.

The number of elements	Tree	Order	Operation	Time (second)
1000	BST	Random	Insert	0.001514
			Find	0.000291
	AVL		Insert	0.001239
			Find	0.000243
	BST	Skewed	Insert	0.007952
			Find	0.007726
	AVL		Insert	0.001392
			Find	0.000174
10000	BST	Random	Insert	0.004494
			Find	0.003315
	AVL		Insert	0.015151
			Find	0.002874
	BST	Skewed	Insert	0.711101
			Find	0.702526

	AVL		Insert	0.015910
			Find	0.002056
100000	BST	Random	Insert	0.058049
			Find	0.048139
	AVL		Insert	0.190916
			Find	0.036910
	BST	Skewed	Insert	82.601738
			Find	82.436920
	AVL		Insert	0.189726
			Find	0.025554

In general, the result above can be interpreted as follows. The direction of inequality is based on the performing time.

1. BST vs AVL Tree
   a) Random Insert: BST < AVL
   b) Random Find: BST >= AVL
   c) Skewed Insert: BST >> AVL
   d) Skewed Find: BST >> AVL
2. Types of the order
   a) BST Insert: Random << Skewed
   b) BST Find: Random << Skewed
   c) AVL Insert: Random = Skewed
   d) AVL Find: Random >= Skewed

The reasons for these results are as follows.

1. AVL Tree is balanced dynamically to keep the height of a binary search tree O(log n) for search, insertion and deletion. But BST is not balanced dynamically, so in the worst case, the height of BST is O(n) and this is so on search, insertion, and deletion.
2. Because AVL Tree manage strict height-balanced structure, up to two rotations can be accompanied during the insertion process. So insertion can be slower than lookup.

In random order, the number of elements determines which is more efficient between BST and AVL tree. However, AVL tree is overwhelmingly efficient when it comes to skewed order. Therefore, it is more efficient to use AVL tree if the given key values are generally well sorted. If how the given key values are sorted is not certain, or they are not sorted is, the use of BST also can be considered.