컴퓨터알고리즘과실습 실습6

2020112046 이재원

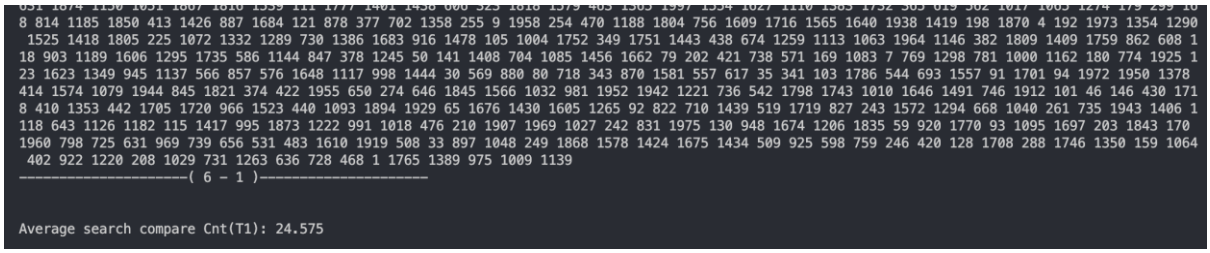
```
#include <algorithm>
#include <iostream>
#include <random>
#include <vector>

using namespace std;

typedef int itemType;
typedef int infoType;

class BST {
   private:
    struct node {
        int key;
        infoType info;
        struct node *left, *right;
        node(int k, infoType i, struct node *l, struct node *r) {
            key = k;
            info = i;
            left = l;
            right = r;
        }
    };
    struct node *head, *z;

   public:
    BST() {
        z = new node(0, 0, 0, 0);
        head = new node(0, 0, z, z);
    }
    ~BST(){};
    struct node *returnHead() {
        return head;
    }
    infoType BSTsearch(int v, int &count);
    itemType inorderSearch(struct node *x);
    void BSTinsert(int v, infoType info);
    int maxDepth(struct node *node) {
        if (node == z)
            return 0;
        else {
            int lDepth = maxDepth(node->left);
            int rDepth = maxDepth(node->right);
            if (lDepth > rDepth)
```

```
            return (lDepth + 1);
        else
            return (rDepth + 1);
    }
}
int printLevel(struct node *x, int value, int level) {
    if (x == z)
        return 0;
    if (x->key == value)
        return level;
    else {
        int downlevel = printLevel(x->left, value, level + 1);
        // left child node 쪽 subtree에서 value를 찾았다면 return
        if (downlevel != 0)
            return downlevel;
        // left child node 쪽 subtree에서 value를 못 찾았다면 right child node 쪽
subtree에서 검색
        // left child node 쪽 subtree에 없다면 무조건 right subtree에 존재함.
        // input value를 tree에 존재하는 value로 가정했기 때문.
        downlevel = printLevel(x->right, value, level + 1);
        return downlevel;
    }
}
void MaxMinLvl() {
    // print max Depth of tree
    cout << "Max Depth: " << maxDepth(head->right) << endl;
    // print min value's Level of tree
    cout << "Min Value's Level: " << printLevel(head->right, 0, 0) << endl;
    // print max value's Level of tree
    cout << "Max Value's Lelel: " << printLevel(head->right, 1999, 0) << endl;
}
};

infoType BST::BSTsearch(int v, int &count) {
    struct node *x = head->right;
    while (v != x->key) {
        // 키값이 같은 지와, 큰 지 작은 지 비교횟수 각각 1번씩으로 침.
        count += 2;
        if (v < x->key)
            x = x->left;
        else
            x = x->right;
    }
    return x->key;
}

void BST::BSTinsert(int v, infoType info) {
    struct node *p, *x;
    p = head;
    x = head->right;
    while (x != z) {
        p = x;
        if (v < x->key)
            x = x->left;
```

```
        else
            x = x->right;
    }
    x = new node(v, info, z, z);
    if (v < p->key)
        p->left = x;
    else
        p->right = x;
}

int inorderLevel = 0;
int inorderArr[2000];
int inorderIndex = 0;
// inorder traversal 결과는 inorderArr에 저장
itemType BST::inorderSearch(struct node *x) {
    if (!inorderLevel) x = head->right;
    inorderLevel++;
    if (x->left != z) inorderSearch(x->left);
    inorderArr[inorderIndex++] = x->key;
    if (x->right != z) inorderSearch(x->right);
}

int main() {
    BST T1;
    vector<int> a;
    int N = 2000;

    for (int i = 0; i < N; i++) {
        a.push_back(i);
    }

    // shuffle the vector
    random_device rd;
    default_random_engine rng(rd());
    shuffle(a.begin(), a.end(), rng);

    // insert
    for (int i = 0; i < N; i++) {
        cout << a[i] << " ";
        T1.BSTinsert(a[i], 1);
    }

    cout << "--------------------( 6 - 1 )---------------------" << endl;
    int searchCnt = 0;
    for (int i = 0; i < N; i++) {
        T1.BSTsearch(i, searchCnt);
    }

    cout << endl
         << endl;
    // shuffle할 때마다 다른 결과가 나오므로, 평균 비교 회수가 실행할 때마다 달라짐.
    cout << "Average search compare Cnt(T1): " << (double)searchCnt / (double)N << endl
         << endl;
```

```
    cout << "---------------------( 6 - 2 )----------------------" << endl;

    T1.inorderSearch(NULL);

    // print inorder
    for (int i = 0; i < N; i++) {
        cout << inorderArr[i] << " ";
    }

    BST T2;
    for (int i = 0; i < N; i++) {
        T2.BSTinsert(inorderArr[i], 1);
    }

    int searchCnt_T2 = 0;
    for (int i = 0; i < N; i++) {
        T2.BSTsearch(i, searchCnt_T2);
    }

    cout << endl
         << endl;
    // shuffle할 때마다 다른 결과가 나오므로, 평균 비교 회수가 실행할 때마다 달라짐.
    cout << "Average search compare Cnt (T2): " << (double)searchCnt_T2 / (double)N <<
endl
         << endl;

    cout << "---------------------( 6 - 3 )----------------------" << endl;
    T1.MaxMinLvl();

    return 0;
}
```

6-1



6-2, 6-3

```
2 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1
1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 170
21 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736
 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 17
780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 179
9 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1
1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 185
68 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883
 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 19
927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942
6 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1
1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999

Average search compare Cnt (T2): 1999

--------------------( 6 - 3 )--------------------
Max Depth: 28
Min Value's Level: 7
Max Value's Lelel: 7
```

```
// shuffle the vector
random_device rd;
default_random_engine rng(rd());
shuffle(a.begin(), a.end(), rng);
```

```
// insert
for (int i = 0; i < N; i++) {
cout << a[i] << " ";
T1.BSTinsert(a[i], 1);
}
```

=> vector를 이용하여 랜덤으로 shuffle하고 Tree에 insert해서, 실행시킬 때마다 tree가 다르게 생성된다.

따라서, 평균 비교 횟수, tree의 전체 level, 최대값과 최소값의 level은 계속 달라진다.

T1을 inorder로 traversal하여 insert한 T2는 skewed BST가 만들어지기 때문에,

0~1999까지 n일 때 각각 비교 횟수가 약 2n-1이므로, 총 비교 횟수는 Σ[n=0 to 1999] (2n-1).

따라서 평균 비교 횟수는, 약 1999회가 된다.