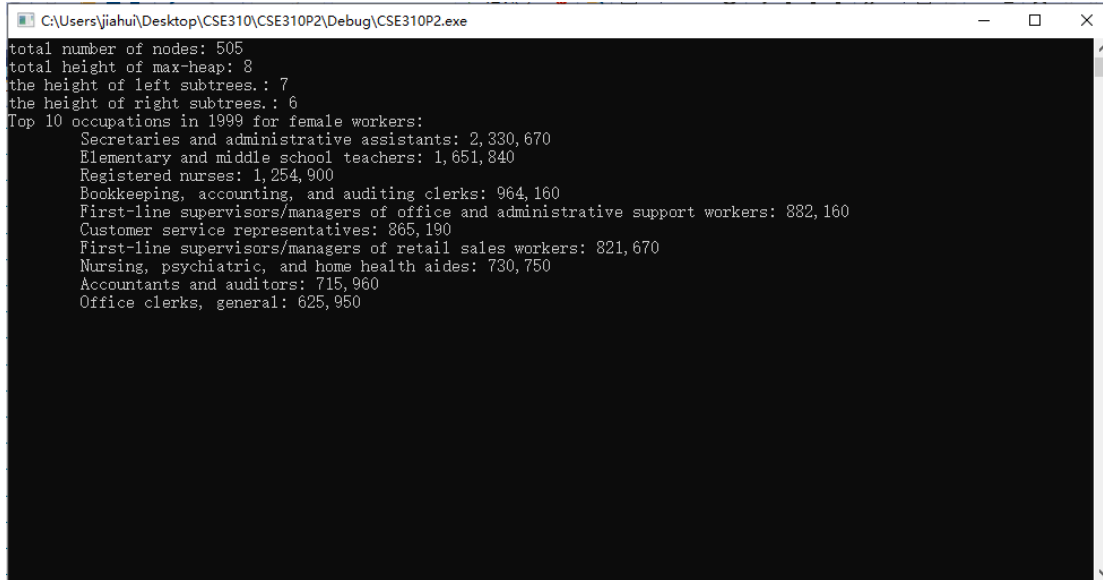


CSE310 P2 Milestone

Jiahui Li 1220317163

Max-heap: For a max-heap constructed to answer find max queries:

1. Print a count of the total number of nodes in the max-heap.



```
C:\Users\jiahui\Desktop\CSE310\CSE310P2\Debug\CSE310P2.exe
total number of nodes: 505
total height of max-heap: 8
the height of left subtrees.: 7
the height of right subtrees.: 6
Top 10 occupations in 1999 for female workers:
  Secretaries and administrative assistants: 2,330,670
  Elementary and middle school teachers: 1,651,840
  Registered nurses: 1,254,900
  Bookkeeping, accounting, and auditing clerks: 964,160
  First-line supervisors/managers of office and administrative support workers: 882,160
  Customer service representatives: 865,190
  First-line supervisors/managers of retail sales workers: 821,670
  Nursing, psychiatric, and home health aides: 730,750
  Accountants and auditors: 715,960
  Office clerks, general: 625,950
```

```
else if (type == "female") {
    BuildMaxHeap(soc, odaSize, type);

    //REPORT(MAX-HEAP)
    int nodeNum = BuildMaxHeap(soc, odaSize, type);
    cout << "total number of nodes: " << nodeNum << endl;
    cout << "total height of max-heap: " << floor(log2(nodeNum)) << endl;
    cout << "the height of left subtrees.: " << floor(log2(nodeNum)) - 1 << endl;
    cout << "the height of right subtrees.: " << floor(log2(nodeNum)) - 2 << endl;

    cout << "Top " << maxTimes << " occupations in " << 1999 << " for female workers:" << endl;
    for (int a = 0; a < maxTimes; a++) {
        cout << '\t' << soc[0].occupation << ": " << addComma(soc[0].female) << endl;
        MaxHeapDelete(soc, odaSize, type);
        odaSize = odaSize - 1;
    }
}
```

2. Print the total height of the max-heap, and the height of its left and right subtrees. Is it "height balanced"?

```
C:\Users\jiahui\Desktop\CSE310\CSE310P2\Debug\CSE310P2.exe
total number of nodes: 505
total height of max-heap: 8
the height of left subtrees.: 7
the height of right subtrees.: 6
Top 10 occupations in 1999 for female workers:
  Secretaries and administrative assistants: 2,330,670
  Elementary and middle school teachers: 1,651,840
  Registered nurses: 1,254,900
  Bookkeeping, accounting, and auditing clerks: 964,160
  First-line supervisors/managers of office and administrative support workers: 882,160
  Customer service representatives: 865,190
  First-line supervisors/managers of retail sales workers: 821,670
  Nursing, psychiatric, and home health aides: 730,750
  Accountants and auditors: 715,960
  Office clerks, general: 625,950
```

```
else if (type == "female") {
    BuildMaxHeap(soc, odaSize, type);

    //REPORT(MAX-HEAP)
    int nodeNum = BuildMaxHeap(soc, odaSize, type);
    cout << "total number of nodes: " << nodeNum << endl;
    cout << "total height of max-heap: " << floor(log2(nodeNum)) << endl;
    cout << "the height of left subtrees.: " << floor(log2(nodeNum)) - 1 << endl;
    cout << "the height of right subtrees.: " << floor(log2(nodeNum)) - 2 << endl;

    cout << "Top " << maxTimes << " occupations in " << 1999 << " for female workers:" << endl;
    for (int a = 0; a < maxTimes; a++) {
        cout << '\t' << soc[0].occupation << ": " << addComma(soc[0].female) << endl;
        MaxHeapDelete(soc, odaSize, type);
        odaSize = odaSize - 1;
    }
}
```

It is not "height balanced".

3. Is the max-heap an efficient implementation of the find max query? What is another implementation to compute the answer to a find max query? Would it be more or less efficient (does it depend on the value of n)?

Yes, the max-heap is an efficient implementation of the find max query.

Quicksort.

It would be more efficient if the value of n is large.

Binary search tree: For the BST constructed to answer range occupation queries:

1. Print a count of the total number of nodes in the BST.

```
C:\Users\jiahui\Desktop\CSE310\CSE310P2\Debug\CSE310P2.exe
the total number of nodes: 505
the height of the root of the BST: 20
the height of its left subtree: 4
the height of its right subtree: 9
```

```
int GetBTHeight(bst* root) {
    if (root == nullptr)
        return 0;
    return max(GetBTHeight(root->left), GetBTHeight(root->right)) + 1;
}

int leftSub(bst* tree) {
    if (tree == nullptr)
        return 0;
    return leftSub(tree->left) + 1;
}

int rightSub(bst* tree) {
    if (tree == nullptr)
        return 0;
    return rightSub(tree->right) + 1;
}

//REPORT(BST)
cout << "the total number of nodes: " << odaSize << endl;
cout << "the height of the root of the BST: " << GetBTHeight(root) << endl;
cout << "the height of its left subtree: " << leftSub(root) << endl;
cout << "the height of its right subtree: " << rightSub(root) << endl;
```

2. Print the height of the root of the BST, and the height of its left and its right subtree. Is it "height balanced"?

```
C:\Users\jiahui\Desktop\CSE310\CSE310P2\Debug\CSE310P2.exe
the total number of nodes: 505
the height of the root of the BST: 20
the height of its left subtree: 4
the height of its right subtree: 9
```

```
int GetBTHeight(bst* root) {
    if (root == nullptr)
        return 0;
    return max(GetBTHeight(root->left), GetBTHeight(root->right)) + 1;
}

int leftSub(bst* tree) {
    if (tree == nullptr)
        return 0;
    return leftSub(tree->left) + 1;
}

int rightSub(bst* tree) {
    if (tree == nullptr)
        return 0;
    return rightSub(tree->right) + 1;
}

//REPORT(BST)
cout << "the total number of nodes: " << odaSize << endl;
cout << "the height of the root of the BST: " << GetBTHeight(root) << endl;
cout << "the height of its left subtree: " << leftSub(root) << endl;
cout << "the height of its right subtree: " << rightSub(root) << endl;
```

It is not "height balanced".

- Is the BST an efficient implementation of the range occupation query? What is another implementation to compute the answer to a range occupation query? Would it be more or less efficient (does it depend on the value of n)?

Yes, the BST is an efficient implementation of the find max query.

AVL tree.

AVL tree would be more efficient, its search performance is $O(\log n)$ in the worst case.

Hash table: For the hash table constructed to answer find occupation queries:

1. Print a table that lists for each chain length l , $0 \leq l \leq l_{max}$, the number of chains of length l , up to the maximum chain length l_{max} that your hash table contains.

$l = 1$:

```
chain[0,2,3,4,12,15,16,20,22,26,30,31,32,33,40,41,42,43,44,45,49,50,51,56,62,63,66,72,81,82,86,
,87,92,96,97,100,101,103,106,111,112,116,118,121,123,126,127,131,132,133,138,142,143,144,
,148,156,157,158,162,163,164,165,168,169,178,182,183,185,188,189,190,195,197,202,206,208,
,211,216,217,221,227,228,229,230,231,232,233,238,244,248,249,253,257,264,266,267,270,277,
,278,283,284,287,288,292,294,298,303,304,306,308,309,313,314,317,318,319,322,324,327,329,
,337,338,347,349,357,358,359,362,366,368,369,378,387,388,393,397,399,400,402,403,404,405,
,408,409,414,417,418,424,425,426,427,432,434,438,444,457,458,473,479,483,487,489,492,493,
,499,509,515,519,520,521,529,530,539,554,555,575,590,594,605,608,620,630,633,635,640,645,
,655,665,690,704,714,715,721,722,724,725,733,735,744,759,770,771,772,780,781,782,783,785,
,786,787,788,789,791,792,799,800,802,806,809,811,812,819,821,830,832,836,841,846,852,860,
,869,880,905,911,916,917,918,921,922,923,924,925,926,932,934,936,940,941,942,943,945,946,
,948,955,956,960,965,967,968,971,974,975,981,989,991,993,997,998,1000,1002,1005,1006,100
7,1010,1012,1015,1016,1017,1020,1021,1022,1023,1024,1025,1026,1027,1030,1031,1033,103
7,1039,1043,1047,1049,1053,1056,1063,1066,1070,1071,1072,1076,1077,1078,1082,1083,109
0,1093,1094,1102,1104,1106,1111,1113,1116,1117,1126,1128,1132,1133,1137,1143,1148,115
3,1157,1161,1168,1178,1188,1190,1194,1197,1198,1200,1208,1209,1210,1218,1221,1231,123
3,1234,1237,1241,1244,1247,1251,1254,1257,1260,1264,1267,1268,1276,1277,1282,1283,128
7,1288,1289,1290,1291,1292,1300,1310,1313,1317,1319,1321,1324,1328,1357,1363,1364,136
8,1371,1372,1373,1374,1376,1377,1379,1387,1393,1408,1418,1424,1428,1433,1435,1438,143
9,1443,1445,1446,1448,1449,1454,1455,1458,1464,1466,1468,1473,1475,1478,1479,1482,148
4,1485,1489,1493,1494,1499,1503,1504,1505,1508,1509,1518,1519,1520]
```

$l = 2$:

```
chain[6,11,52,73,76,91,166,167,201,218,234,247,274,398,407,437,447,600,610,669,745,784,79
0,915,966,985,986,1038,1073,1074,1075,1092,1103,1121,1122,1192,1203,1213,1228,1318,13
25,1365,1375,1378,1398,1404,1413,1465,1514]
```

$l = 3$:

```
chain[236,996]
```

2. Compute and print the load factor for the hash table. Do you consider the hash function to

be a "good" one for the SOC codes?

m = 1523. Yes.

3. Is the hash-table an efficient implementation of the find occupation query? What is another implementation to compute the answer to a find occupation query? Do you think it would it be more or less efficient? Why or why not?

Yes.

Array.

No. The hash table combines arrays and linked lists, so the overall performance is better.