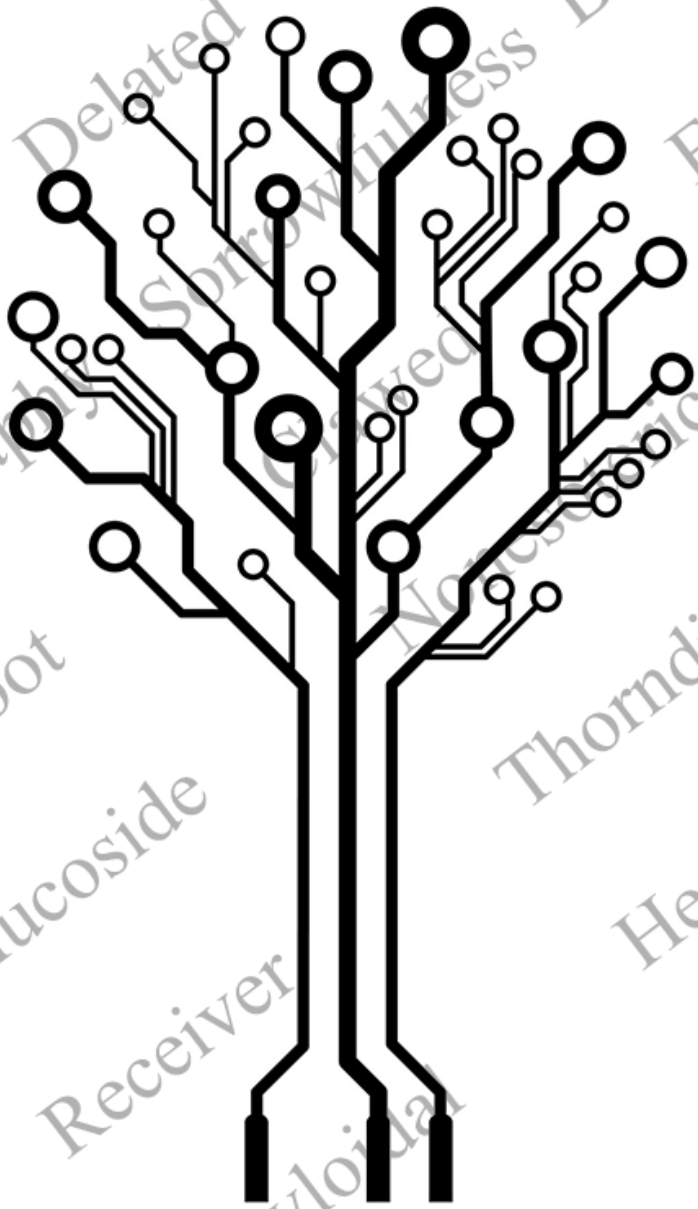


TEXT REPRESENTATION WITH BINARY TREES

MARMARA ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ

Author: Onurcan İŞLER
CSE2025 Project
Number:150120825

1.TREE A

Constructing the tree given in part a was easy. All we had to do is to insert each word to the tree. While inserting to the tree we considered the words as keys so that the tree in part a would be binary search tree.

1.1.ACCESS TIME CALCULATION OF TREE GIVEN IN PART A

To calculate the total access time of a binary search tree, we should add up the access time of each tree node in that tree. An access time of a tree node in a binary search tree is equals to the total number of comparison(s) we should make so as to reach a that tree node.

Suppose we have searched a word '$frequency$' times. Then total number of comparisons we made is found with this given formula:

$$accesstime = (depth + 1) . frequency$$

After constructing the binary search tree in part a by using the words given in 'input.txt' file, we calculated its total access time. We applied the given formula to each tree node in that tree and we add up each access time we had calculated. It gave us the total access time of the tree given in a.

Table 1: Access times of the tree given in part a.

Average Access Time (units)	375.28
Total Access Time (units)	18764

2.TREE C

In part c, we are asked to find a binary tree so as to minimize the total access time. So we should find the best binary tree which will have minimum total access time. **<u>It is an undeniable fact that the best binary tree having the minimum total access time is a binary search tree.</u>**

2.1.WHY THE TREE IN PART C IS A BINARY SEARCH TREE?

Let's think about the opposite situation. Suppose the tree in part c is not a binary search tree and we want to find a particular word in that tree. What would happen? We will have to traverse all of the nodes in that tree to find that particular word. This would take a lot of time. It would take so much time that the tree we found in part a would be faster than the tree we have found in part c.

2.2.CONSTRUCTING THE TREE IN PART C

Unfortunately being a binary search doesn't make our tree be the best binary tree. We should find an additional future that will make our tree be the best binary tree. What should we do? Should we balance that tree so as to minimize total access time? Actually balancing the tree doesn't make much difference. Even some cases, balancing the tree may increase the total access time of the tree since the most frequently searched words in that tree could be pushed to the deepest part of the tree.

To construct the tree in part c we should consider the which words is search how many times. In general, putting the most frequently searched words close to the root help us to decrease total access time. (Recall the formula given in first page. As depth increases the access time becomes smaller.)

$$accesstime = (depth + 1).frequency$$

Not that though while putting the most frequently searched words close to the root, we have to keep our tree as a binary search tree because this formula can be applied if and only if the tree is a binary search tree.

2.2.1.Formula to construct the best binary search tree

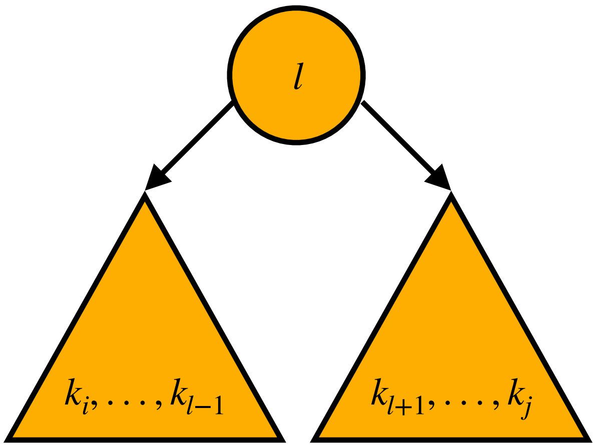
So, how do we write a program that will keep most frequently searched tree nodes near the root? It sounds scary right? Because we will have to know where each tree node should be placed. Applying brute force finding the each possible binary trees and checking the total access time will take $O(n2^n)$ time which is really big complexity. It may take several minutes for our program to be compiled when inputs are very large.

We should apply dynamic programming method. We separate the problem into smaller sub problems and we reach the solution by solving each of them. Suppose we have ordered keys $k_i, k_{i+1}, k_{i+2}, k_{i+3}, \ldots, k_j$ and we want to construct the best binary search tree with these keys. We start with to constructing the subtrees of that best binary search tree. Whenever we construct a subtree we ask ourselves which key should be the root of that subtree. Of course, the one with the biggest frequency.

We will have two tables. One is going to hold the total access time of each subtree and the other going to hold the root of each subtree having the minimum total access time.

int costTable[i][j] = minimum total access time of the best BST with the keys $k_i, k_{i+1}, k_{i+2}, \ldots, k_j$.
treeNode rootTable[i][j] = root of the subtree having a total access time of costTable[i][j].

Since $k_i <= k_j$ costTable and rootTable are sparse matrices. We basically ignore the elements at the index of (i, j) where $i > j$. Suppose we have selected l as a root of the keys $k_i, k_{i+1}, k_{i+2}, k_{i+3}, \ldots, k_j$ then we have a optimized tree as follows:



Graph 1: The structure of tree with the keys $k_i, k_{i+1}, k_{i+2}, k_{i+3}, ..., k_j$ and l as a root.

So total cost becomes;

$$costTable[i][j] = min_{i \le l \le j}(costTable[i][l-1] + costTable[l+1][j]) + \sum_{n=i}^{j} freq(k_n)$$

Where $costTable[i][j]$ is the total number of comparisons in an optimized binary search tree with the keys $k_i, k_{i+1}, k_{i+2}, k_{i+3}, ..., k_j$. Let's look at the costTable below:

$costTable[i][i-1]$	$costTable[i][i]$	...	$costTable[i][l-1]$	$costTable[i][j]$
				$costTable[i+1][j]$
				...
				$costTable[j][j]$
				$costTable[j-1][j]$

$when\ l = i : costTable[i][i-1] + costTable[i+1][j]$
$when\ l = i+1 : costTable[i][ii] + costTable[i+2][j]$
...

So we select k_l which makes $costTable[i][j]$ be minimum. The code can be found in my code file. Now let's see what happens when we use 'input.txt' file as our input.

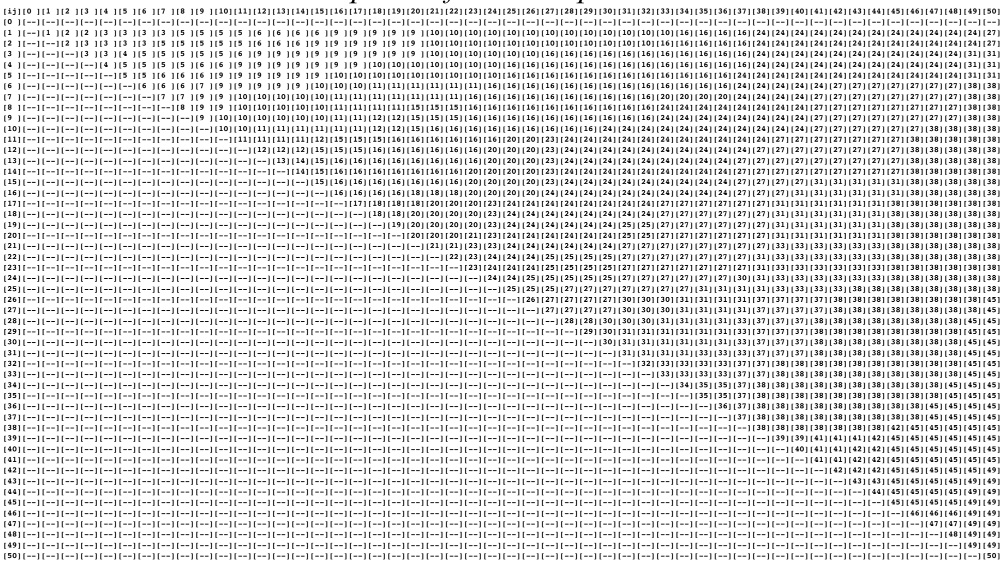
2.2.2.Constructing the tree using the data in 'input.txt' file

We have 50 keys in given input file. We order these keys using the binary search tree we have found in part a. So the keys become:

```
k1:algorithm
k2:Ankara
k3:bag
k4:board
k5:book
k6:bus
k7:car
k8:city
k9:class
k10:clock
k11:club
k12:compiler
k13:computer
k14:country
k15:department
k16:Dubai
k17:economics
k18:excel
k19:faculty
k20:game
k21:grade
k22:group
k23:head
k24:kitchen
k25:lab
k26:library
k27:meeting
k28:memory
k29:mouse
k30:name
k31:news
k32:New York
k33:pencil
k34:people
k35:plane
k36:population
k37:professor
k38:room
k39:society
k40:software
k41:sports
k42:student
k43:teacher
k44:team
k45:television
k46:text
k47:traffic
k48:university
k49:visit
k50:window
```

After ordering these keys we start constructing the costTable and rootTable. Let's observe the rootTable when we compile my code.

Table 2: rootTable when 'input.txt' file is compiled.



```
[ij][0 ][1 ][2 ][3 ][4 ][5 ][6 ][7 ][8 ][9 ][10][11][12][13][14][15][16][17][18][19][20][21][22][23][24][25][26][27][28][29][30][31][32][33][34][35][36][37][38][39][40][41][42][43][44][45][46][47][48][49][50]
[0 ][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--]
[1 ][--][1 ][2 ][2 ][3 ][3 ][3 ][3 ][5 ][5 ][5 ][5 ][6 ][6 ][6 ][6 ][9 ][9 ][9 ][9 ][9 ][10][10][10][10][10][10][10][10][10][10][10][10][10][16][16][16][16][24][24][24][24][24][24][24][24][24][24][24][24][27]
[2 ][--][--][2 ][3 ][3 ][3 ][3 ][5 ][5 ][5 ][5 ][5 ][6 ][6 ][6 ][9 ][9 ][9 ][9 ][9 ][9 ][10][10][10][10][10][10][10][10][10][10][10][10][16][16][16][16][16][24][24][24][24][24][24][24][24][24][24][24][24][27]
[3 ][--][--][--][3 ][3 ][4 ][5 ][5 ][5 ][5 ][5 ][6 ][9 ][9 ][9 ][9 ][9 ][9 ][9 ][9 ][9 ][10][10][10][10][10][10][16][16][16][16][16][16][16][16][16][16][16][24][24][24][24][24][24][24][24][24][24][24][31][31]
[4 ][--][--][--][--][4 ][5 ][5 ][5 ][5 ][6 ][6 ][9 ][9 ][9 ][9 ][9 ][9 ][9 ][10][10][10][10][10][10][10][16][16][16][16][16][16][16][16][16][16][16][16][16][24][24][24][24][24][24][24][24][24][24][24][31][31]
[5 ][--][--][--][--][--][5 ][5 ][6 ][6 ][6 ][9 ][9 ][9 ][9 ][9 ][9 ][10][10][10][10][10][10][10][10][10][16][16][16][16][16][16][16][16][16][16][16][16][24][24][24][24][24][24][24][24][24][24][24][24][31][31]
[6 ][--][--][--][--][--][--][6 ][6 ][6 ][7 ][9 ][9 ][9 ][9 ][9 ][10][10][10][11][11][11][11][11][11][16][16][16][16][16][16][16][16][16][16][16][16][16][24][24][24][24][24][27][27][27][27][27][27][27][38][38]
[7 ][--][--][--][--][--][--][--][7 ][7 ][9 ][9 ][10][10][10][10][10][11][11][11][11][11][11][11][16][16][16][16][16][16][16][16][16][16][20][20][20][20][24][24][24][24][27][27][27][27][27][27][27][27][38][38]
[8 ][--][--][--][--][--][--][--][--][8 ][9 ][9 ][10][10][10][10][10][11][11][11][11][15][15][15][16][16][16][16][16][16][16][16][16][16][24][24][24][24][24][24][24][24][27][27][27][27][27][27][27][27][38][38]
[9 ][--][--][--][--][--][--][--][--][--][9 ][10][10][10][10][10][10][11][11][12][12][15][15][15][16][16][16][16][16][16][16][16][16][16][24][24][24][24][24][24][24][24][27][27][27][27][27][27][27][27][38][38]
[10][--][--][--][--][--][--][--][--][--][--][10][10][11][11][11][11][11][11][12][12][15][16][16][16][16][16][16][16][16][16][24][24][24][24][24][24][24][24][24][24][24][27][27][27][27][27][27][38][38][38][38]
[11][--][--][--][--][--][--][--][--][--][--][--][11][11][11][11][12][15][15][15][16][16][16][16][16][16][20][20][23][24][24][24][24][24][24][24][24][24][24][24][27][27][27][27][27][27][27][38][38][38][38][38]
[12][--][--][--][--][--][--][--][--][--][--][--][--][12][12][12][15][15][15][16][16][16][16][16][16][20][20][20][23][24][24][24][24][24][24][24][24][24][24][27][27][27][27][27][27][27][27][38][38][38][38][38]
[13][--][--][--][--][--][--][--][--][--][--][--][--][--][13][14][15][16][16][16][16][16][16][16][16][20][20][20][23][24][24][24][24][24][24][24][24][24][27][27][27][27][27][27][27][27][27][38][38][38][38][38]
[14][--][--][--][--][--][--][--][--][--][--][--][--][--][--][14][15][16][16][16][16][16][16][16][20][20][20][20][23][24][24][24][24][24][24][24][24][24][27][27][27][27][27][27][27][27][27][38][38][38][38][38]
[15][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][15][16][16][16][16][16][16][16][20][20][20][20][23][24][24][24][24][24][24][24][24][24][27][27][27][27][31][31][31][31][31][38][38][38][38][38]
[16][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][16][16][16][16][18][18][18][20][20][20][20][24][24][24][24][24][24][24][24][24][24][27][27][27][31][31][31][31][31][31][38][38][38][38][38]
[17][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][17][18][18][18][20][20][20][23][24][24][24][24][24][24][24][24][27][27][27][27][27][27][31][31][31][31][31][31][38][38][38][38][38][38]
[18][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][18][18][20][20][20][20][23][24][24][24][24][24][24][24][24][27][27][27][27][27][27][31][31][31][31][31][31][38][38][38][38][38][38]
[19][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][19][20][20][20][20][23][24][24][24][24][24][24][25][25][27][27][27][27][27][27][31][31][31][31][31][31][38][38][38][38][38][38]
[20][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][20][20][20][21][23][24][24][24][24][24][24][25][25][27][27][27][27][27][27][31][31][31][31][31][31][38][38][38][38][38][38]
[21][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][21][21][23][23][24][24][24][24][24][24][27][27][27][27][27][27][27][27][33][33][33][33][33][33][38][38][38][38][38][38]
[22][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][22][23][24][24][24][25][25][25][25][27][27][27][27][27][27][27][31][33][33][33][33][33][33][38][38][38][38][38][38]
[23][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][23][24][24][24][25][25][25][25][27][27][27][27][27][27][27][31][33][33][33][33][33][33][38][38][38][38][38][38]
[24][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][24][24][25][25][25][25][25][27][27][27][27][27][27][30][31][33][33][33][33][33][33][38][38][38][38][38][38]
[25][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][25][25][25][27][27][27][27][27][27][27][31][31][31][31][33][33][33][33][38][38][38][38][38][38][38][38]
[26][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][26][27][27][27][27][30][30][30][31][31][31][31][37][37][37][37][38][38][38][38][38][38][38][38][45]
[27][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][27][27][27][27][30][30][30][31][31][31][31][37][37][37][37][38][38][38][38][38][38][38][38][45]
[28][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][28][28][30][30][30][31][31][31][31][33][37][37][37][38][38][38][38][38][38][38][38][45][45]
[29][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][29][30][31][31][31][31][31][31][33][37][37][37][38][38][38][38][38][38][38][38][45][45]
[30][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][30][31][31][31][31][31][31][33][37][37][37][38][38][38][38][38][38][38][38][45][45]
[31][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][31][31][31][31][33][33][33][37][37][37][38][38][38][38][38][38][38][38][45][45]
[32][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][32][33][33][33][33][37][37][38][38][38][38][38][38][38][38][38][38][45][45]
[33][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][33][33][33][33][37][37][38][38][38][38][38][38][38][38][38][38][45][45]
[34][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][34][35][35][37][38][38][38][38][38][38][38][38][38][38][45][45][45]
[35][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][35][35][37][38][38][38][38][38][38][38][38][38][45][45][45][45]
[36][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][36][37][38][38][38][38][38][38][38][38][38][45][45][45][45]
[37][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][37][38][38][38][38][38][38][38][38][38][45][45][45][45]
[38][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][38][38][38][38][38][38][38][42][45][45][45][45][45]
[39][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][39][39][41][41][41][42][45][45][45][45][45][45]
[40][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][40][41][41][42][42][45][45][45][45][45][45]
[41][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][41][41][42][42][45][45][45][45][45][45]
[42][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][42][42][42][45][45][45][45][45][49]
[43][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][43][43][45][45][45][45][49][49]
[44][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][44][45][45][45][45][49][49]
[45][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][45][45][45][45][49][49]
[46][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][46][46][46][49][49]
[47][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][47][47][49][49]
[48][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][48][49][49]
[49][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][49][49]
[50][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][--][50]
```

By observing this table we could see that the root of the keys $k_1, k_2, ..., k_{50}$ is $rootTable[1][50] = k_{27}$
So the root of the optimized binary search tree in part c is $k_{27} = meeting$

We use the same technique to find all other subtrees to construct the tree in part c.

For example;
The root of the keys $k_6, k_7, ..., k_{10}$ is $rootTable[6][10] = k_9 = class$

The root of the keys $k_{21}, k_{22}, ..., k_{30}$ is $rootTable[21][30] = k_{24} = kitchen$

The root of the keys k_{17}, k_{18}, k_{19} is $rootTable[17][19] = k_{18} = excel$

...

On the next page you can see how the tree given in part c looks like after being constructed.

Using this table we construct the tree in part c. After compiling 'input.txt' file, we will have this:
(Left child of a node is in upper row. Right child of a node starts with '\'. 'o' means the node is null.)

```
\---+[meeting]
    |---+[clock]
    |   |---+[book]
    |   |   |---+[bag]
    |   |   |   |---+[Ankara]
    |   |   |   |   |---+[algorithm]
    |   |   |   |   \---o
    |   |   |   |
    |   |   |   \---+[board]
    |   |   \---+[car]
    |   |       |---+[bus]
    |   |       \---+[class]
    |   |           |---+[city]
    |   |           |
    |   |           \---o
    |   \---+[game]
    |       |---+[Dubai]
    |       |   |---+[compiler]
    |       |   |   |---+[club]
    |       |   |   |
    |       |   |   \---+[department]
    |       |   |       |---+[country]
    |       |   |       |   |---+[computer]
    |       |   |       |   |
    |       |   |       |   \---o
    |       |   |       |
    |       |   |       \---o
    |       |   |
    |       |   \---+[excel]
    |       |       |---+[economics]
    |       |       |
    |       |       \---+[faculty]
    |       |
    |       \---+[kitchen]
    |           |---+[head]
    |           |   |---+[grade]
    |           |   |   |---o
    |           |   |   \---+[group]
    |           |   \---o
    |           |
    |           \---+[lab]
    |               |---o
    |               \---+[library]
    |
    \---+[television]
        |---+[room]
        |   |---+[pencil]
        |   |   |---+[name]
        |   |   |   |---+[memory]
        |   |   |   |   |---o
        |   |   |   |   \---+[mouse]
        |   |   |   |
        |   |   |   \---+[New York]
        |   |   |       |---+[news]
        |   |   |       \---o
        |   |   |
        |   |   \---+[professor]
        |   |       |---+[plane]
        |   |       |   |---+[people]
        |   |       |   \---+[population]
        |   |       \---o
        |   |
        |   \---+[student]
        |       |---+[sports]
        |       |   |---+[society]
        |       |   |   |---o
        |       |   |   \---+[software]
        |       |   \---o
        |       |
        |       \---+[teacher]
        |           |---o
        |           \---+[team]
        |
        \---+[visit]
            |---+[text]
            |   |---o
            |   \---+[traffic]
            |       |---o
            |       \---+[university]
            \---+[window]
```

2.3.ACCESS TIME CALCULATION OF TREE GIVEN IN PART C

Since the tree we have constructed is also binary search tree, we can use the formula below so as to calculate the access time of each tree node.

$$accesstime = (depth + 1) . frequency$$

<u>This given formula works if and only if the tree is a binary search tree.</u> If the tree we have is not a binary search tree, we will have to traverse all of the nodes to find a particular tree node. After calculation total access time of tree in part c we have:

Table 3: Access times of tree given in part c.

Average Access Time (units)	248.1
Total Access Time (units)	12405

3.DISCUSSION

1) We have seen that total access time of the tree given in part c is much smaller than the that of the tree in part a. But why is that?

As we have discussed in previous parts, we do not actually consider frequencies when building the tree in section a. As we have seen, inserting a word into greater levels increases the total comparisons we have to make to reach that particular word. The total access time of the tree in part a will be greater than that of the tree in part c, since ignoring the frequencies may cause words with larger frequencies to be inserted into larger deep levels and words with smaller frequencies to be inserted into smaller deep levels.

2) Why the tree given in part c was a binary search tree?

As we have learned in previous parts of this report, if a binary tree is not a binary search tree, we have to traverse all of the nodes in that tree to find a particular node. We have to overcome this because we are asked to find a binary tree which will provide smallest access time. So we used a binary search tree to construct the tree in part c.

3) Why don't we just order the words by frequency and insert them into the tree c?

That's the what a beginners programmer would do. <u>Putting all the most frequent words up may sound nice but we will be violating the property of binary search trees if we only consider the frequencies when inserting into the tree.</u> As we have discussed in second question, if the tree is not a binary search tree, then we will have to traverse all of the nodes to find a particular node in that tree. Which is really ugly situation for us.

4.REFERANCES

Thanks to these videos below, I have learned to construct the optimum binary search tree using dynamic programming method. These videos were created by a professor at the University of Nebraska-Lincoln.

https://youtu.be/PU4fgVbzIUs
https://youtu.be/Bn7b6Blj8hE
https://youtu.be/H69FkILSLsU
https://youtu.be/Xt8kuygspOk
https://youtu.be/CTUTPSXyBO8