Algoritmos e Estruturas de Dados

Multi-ordered Trees

Licenciatura em Engenharia Informática

Leonardo dos Santos Flórido - 103360 - 50%
Gabriel Hall Abreu - 102851 - 50%

Índice

Introdução

No âmbito desta unidade curricular pretendemos com este projeto aprofundar os nossos conhecimentos sobre a linguagem C, aproveitando também para pôr em prática novos métodos necessários para a resolução deste problema.

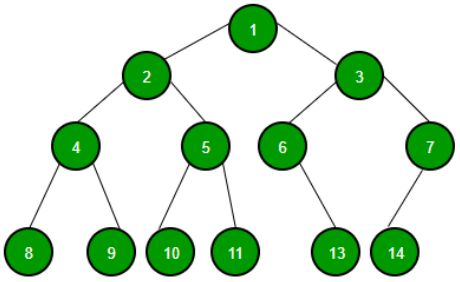
Para tal, será necessário compreender a estrutura de dados da **árvore binária** e os seus respectivos métodos.

Em particular, neste trabalho será necessário compreender como guardar e processar dados diferentes de forma que se tenha acesso aos mesmos usando uma de várias possíveis chaves.

Binary Trees

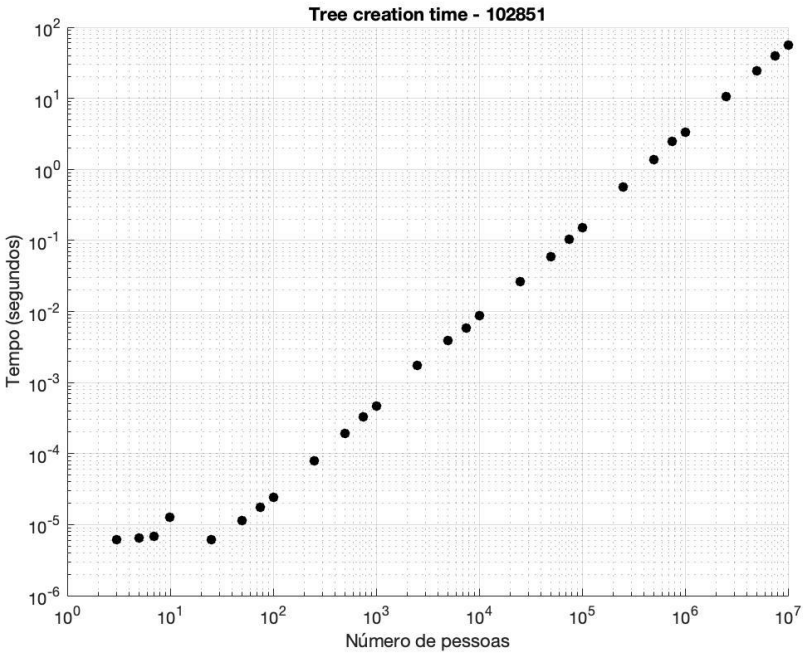
Uma **árvore binária** é uma estrutura dinâmica composta de *nodes*. Cada *node* de informação contém:

- A própria informação (item de dados).
- Um ponteiro para o *node* à esquerda; numa árvore binária ordenada os itens de dados estão guardados neste lado são todos eles mais pequenos do que os dados guardados no *node*.
- Um ponteiro para o *node* à direita; numa árvore binária ordenada os itens de dados estão guardados neste lado são todos eles maiores do que os dados guardados no *node*.
- Opcionalmente, um ponteiro para o *node* pai.

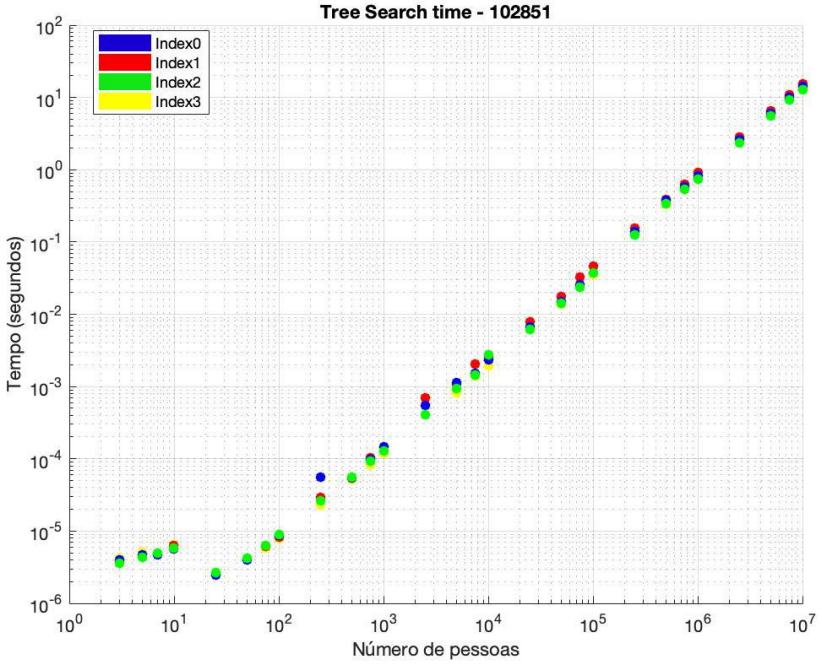


Gráficos

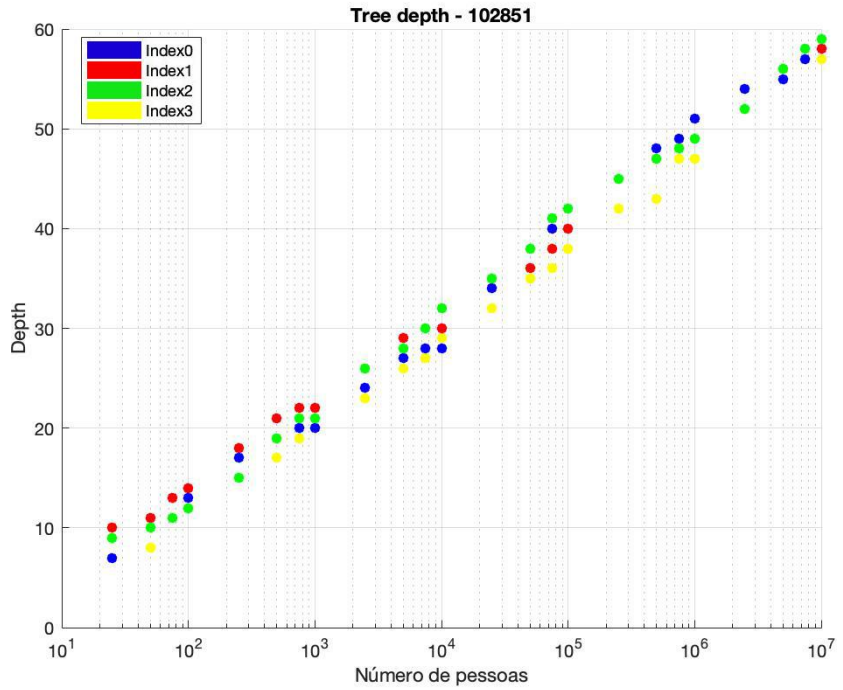
102851 - Relação Tree Creation Time / Number of people (escala: log/log)



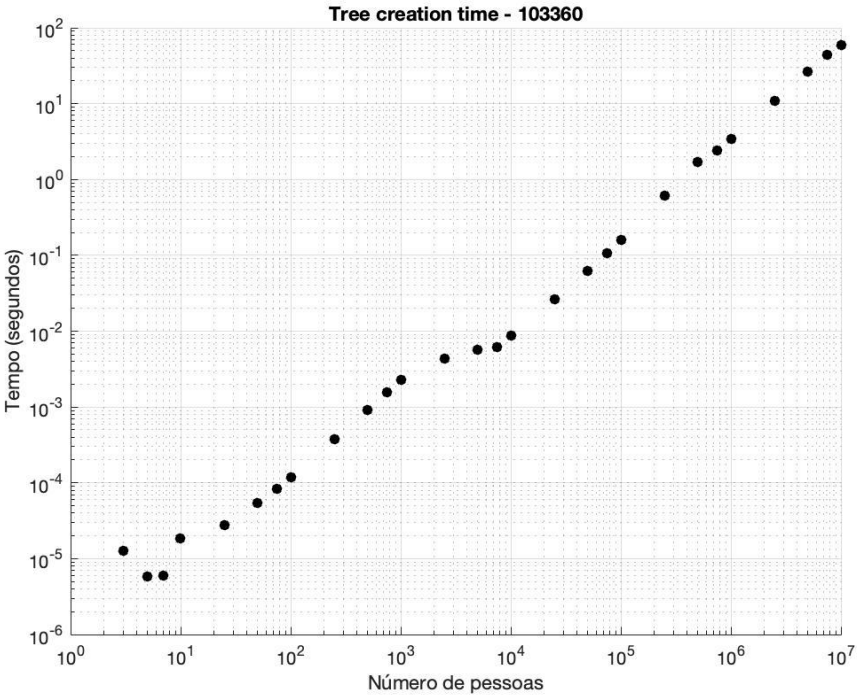
102851 - Relação Tree Search Time / Number of people (escala: log/log)



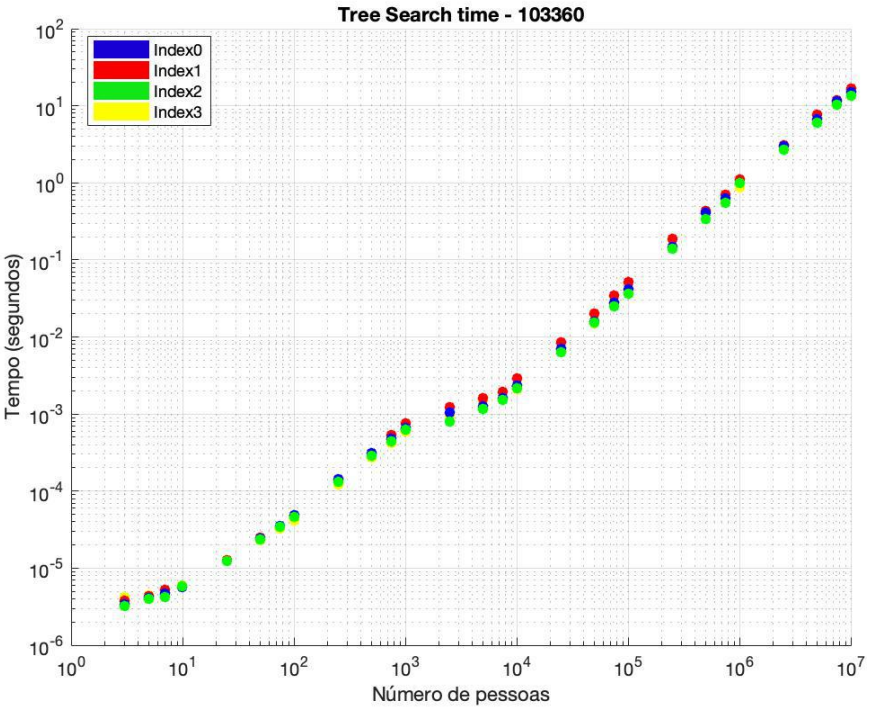
102851 - Relação Tree Depth / Number of people
(escala: linear/log)



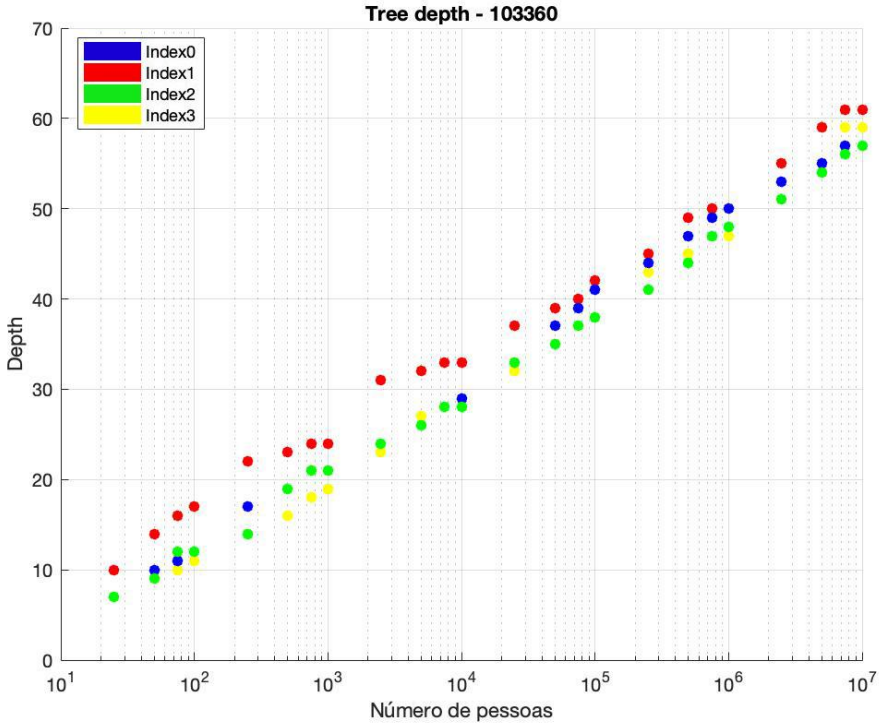
103360 - Relação Tree Creation Time / Number of people
(escala: log/log)



103360 - Relação Tree Search Time / Number of people
(escala: log/log)



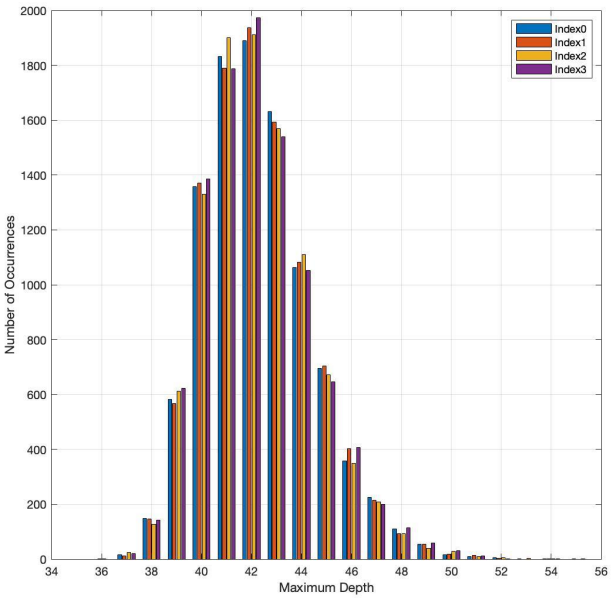
103360 - Relação Tree Depth / Number of people
(escala: linear/log)



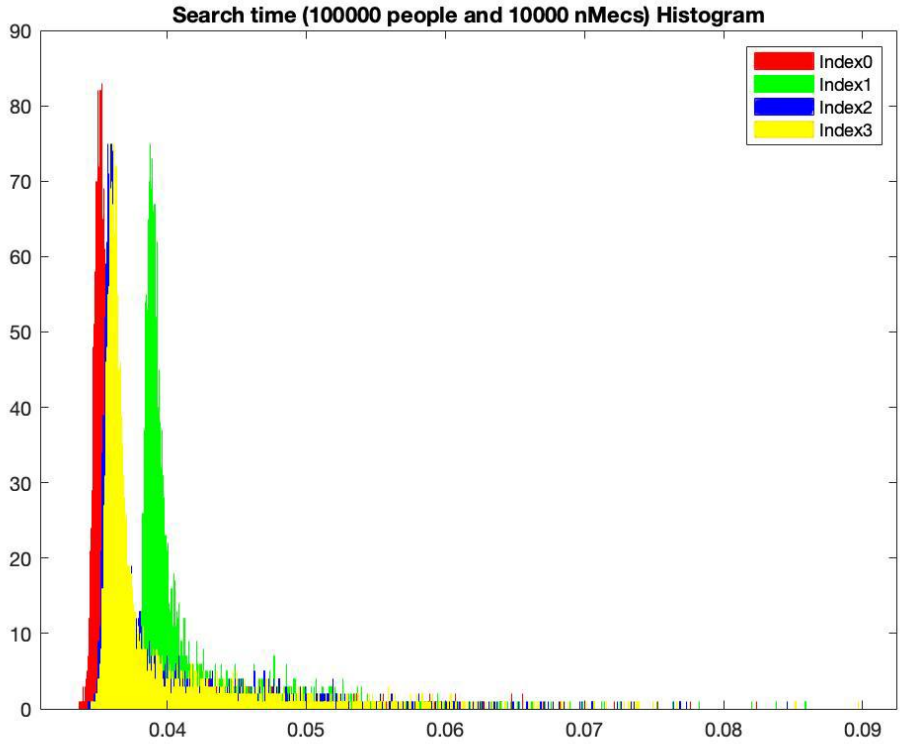
Histogramas

Para cada histograma foram geradas 100000 pessoas e 10000 nMecs.

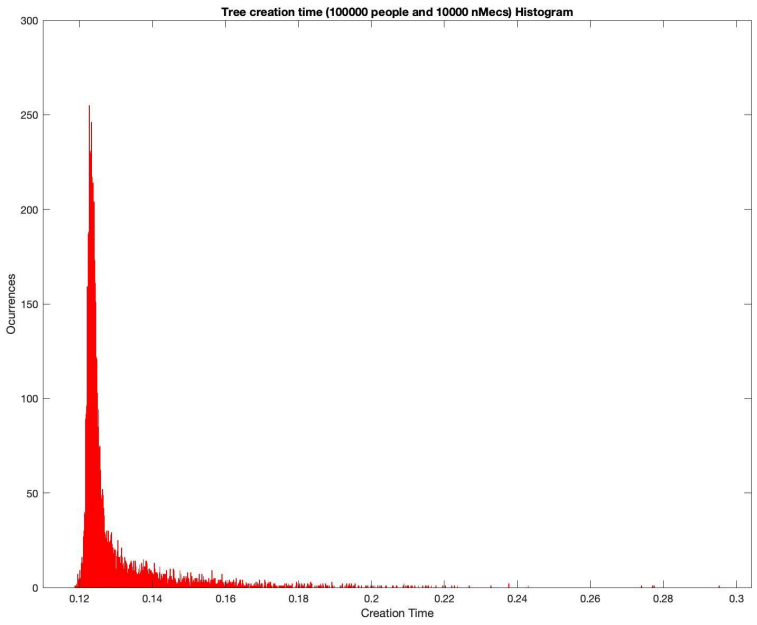
Histograma - número de ocorrências / tree depth



Histograma - número de ocorrências / tree search time



Histograma - número de ocorrências / tree creation time



Influência nas árvores com zip codes limitados

"Does that influence much the execution times for the construction and searches for the tree ordered by the zip codes?"

Resposta:

São disponibilizados 500 zip codes diferentes, ao criar uma árvore (*root*) com mais de 500 pessoas, vão existir pessoas com o mesmo zip code.

Na criação da árvore (*root*) ordenada por zip code, que vai ser feita pela a função *'tree_insert()'* recorrendo a função *'compare_tree_nodes()'*, a função *'compare_tree_nodes()'* em certos casos vai comparar pessoas com o mesmo zip code. Para saber a ordem certa de inserção dessas pessoas na árvore (*root*), terá de comparar o próximo parâmetro das pessoas (no nosso caso vai comparar os *cc*). Com isto, poderão existir casos em que a função terá de fazer 2 comparações para saber a ordem correta para inserir na árvore (*root*), o que aumenta o tempo de criação da árvore. O mesmo raciocínio aplica-se ao *search time* (search time é o tempo que demora a função '*find()*' a correr). A função '*find()*' recorre também à função *'compare_tree_nodes()'*, como em alguns casos têm de fazer 2 comparações, o *search time* vai aumentar.

Código implementado

‘*random_cc()*’

Dentro do ficheiro random_data.c desenvolvemos a função ‘*random_cc()*’ que devolve um número de cartão de cidadão aleatório a ser utilizado posteriormente como quarto campo da estrutura *tree_node_t*.

```
1937 // void random_cc(char cc[MAX_CC_SIZE + 1]) ------------------------------------------ generate a random cc number and place it in the cc[] array
1938 void random_cc(char cc[MAX_CC + 1])
1939 {
1940   int n1 = aed_random() % 9000; // 0000..9999
1941   int n2 = aed_random() % 9000; // 0000..9999
1942   if(snprintf(cc,MAX_CC + 1,"%04d%04d",n1,n2) >= MAX_CC + 1)
1943   {
1944     fprintf(stderr,"CC too large (%04d%04d)\n",n1,n2);
1945     exit(1);
1946   }
1947 }
```

Em termos de estrutura, temos *tree_node_t*, no qual define os *nodes* das *Binary Trees*. Acrescentamos o *cc* à estrutura, na qual se define o número de cartão de cidadão.

```
1  //
2  // AED, January 2022
3  //
4  // Solution of the second practical assignement (multi-ordered tree)
5  //
6  // Place your student numbers and names here
7  //
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include "AED_2021_A02.h"
13
14 //
15 // the custom tree node structure
16 //
17 // we want to maintain three ordered trees (using the same nodes!), so we need four left and four right pointers
18 // so, when inserting a new node we need to do it four times (one for each index), so we will end upo with 4 three roots
19 //
20
21 typedef struct tree_node_s
22 {
23     char name[MAX_NAME_SIZE + 1];                       // index 0 data item
24     char zip_code[MAX_ZIP_CODE_SIZE + 1];               // index 1 data item
25     char telephone_number[MAX_TELEPHONE_NUMBER_SIZE + 1]; // index 2 data item
26     char cc[MAX_CC + 1];                                // index 3 data item
27     struct tree_node_s *left[4];                        // left pointers (one for each index) ---- left means smaller
28     struct tree_node_s *right[4];                       // right pointers (one for each index) --- right means larger
29 } tree_node_t;
```

'compare_tree_nodes()'

Usámos a função *'compare_tree_nodes()'* fornecida pelo o docente para equiparar os *nodes*, o que será útil para criar as *Binary Trees* de uma forma ordenada. Como acrescentamos um quarto parâmetro, foi necessário alterar a função.

```
31  //
32  // the node comparison function (do not change this)
33  //
34
35  int compare_tree_nodes(tree_node_t *node1, tree_node_t *node2, int main_idx)
36  {
37      int i, c;
38
39      for (i = 0; i < 4; i++)
40      {
41          if (main_idx == 0)
42              c = strcmp(node1->name, node2->name);
43          else if (main_idx == 1)
44              c = strcmp(node1->zip_code, node2->zip_code);
45          else if (main_idx == 2)
46              c = strcmp(node1->telephone_number, node2->telephone_number);
47          else if (main_idx == 3)
48              c = strcmp(node1->cc, node2->cc);
49          if (c != 0)
50              return c;                                   // different on this index, so return
51          main_idx = (main_idx == 3) ? 0 : main_idx + 1; // advance to the next index
52      }
53      return 0;
54  }
```

'tree_insert()'

Seguidamente temos a função *'tree_insert()'*, adaptada da contida nas Lecture notes, que passando como argumentos os *roots*, o *node* e o *main_index* vai inserir os *nodes* nos roots de forma ordenada recorrendo a função *'compare_tree_nodes()'*.

```
56  //
57  // tree insertion routine (place your code here)
58  //
59
60  void tree_insert(tree_node_t **roots, tree_node_t *node, int main_index)
61  {
62      /* If the tree is empty, the tree will be to equal the node and will not return any value */
63      if (roots[main_index] == NULL)
64      {
65          roots[main_index] = node;
66          return;
67      }
68      /* Otherwise, recur down the tree */
69      if (compare_tree_nodes(roots[main_index], node, main_index) > 0)
70          tree_insert(roots[main_index]->left, node, main_index);
71      else if (compare_tree_nodes(roots[main_index], node, main_index) < 0)
72          tree_insert(roots[main_index]->right, node, main_index);
73      else
74      {
75          fprintf(stderr, "ERROR generating tree, two equal people!\n");
76          fprintf(stderr, "Please chose another student number.\n");
77          exit(1);
78      }
79  }
```

'*find()*'

A função '*find()*', recorrendo à função '*compare_tree_nodes()*', percorre toda a árvore (*root*) e será posteriormente utilizada para calcular o '*tree search time*' (o tempo de execução da função será o '*tree search time*').

```
81  //
82  // tree search routine (place your code here)
83  //
84
85  tree_node_t *find(tree_node_t *root, tree_node_t node, int main_index)
86  {
87      if (root == NULL)
88          return NULL;
89
90      if (compare_tree_nodes(root, &node, main_index) > 0)
91          return find(root->left[main_index], node, main_index);
92      else if (compare_tree_nodes(root, &node, main_index) < 0)
93          return find(root->right[main_index], node, main_index);
94      else
95          return root;
96  }
```

'*tree_depth()*'

A função '*tree_depth()*' calcula a profundidade máxima da árvore (*root*). Se a profundidade do lado esquerdo da *root* for maior do que a do lado direito será retornada a profundidade do lado esquerdo, caso contrário será retornada a profundidade do lado direito.

```
98  //
99  // tree depth
100 //
101
102 int tree_depth(tree_node_t *root, int main_index)
103 {
104     if (root == NULL)
105         return 0;
106
107     int l = tree_depth(root->left[main_index], main_index);
108     int r = tree_depth(root->right[main_index], main_index);
109
110     if (l > r)
111         return l + 1;
112
113     return r + 1;
114 }
```

‘*list()*’

A função ‘*list()*’ lista ordenadamente todos os *nodes* pertencentes à árvore (*root*) de acordo com o index escolhido pelo utilizador.

```
116   //
117   // list, i,e, traverse the tree (place your code here)
118   //
119
120   int c1 = 1; // global variable
121   void list(tree_node_t *root, int main_index)
122   {
123       if (root != NULL)
124       {
125           list(root->left[main_index], main_index);
126           printf("Person #%d\n", c1++);
127           printf("    name ---------------- %s\n", root->name);
128           printf("    zip code ------------ %s\n", root->zip_code);
129           printf("    telephone number --- %s\n", root->telephone_number);
130           printf("    cc ------------------ %s\n", root->cc);
131           list(root->right[main_index], main_index);
132       }
133   }
```

‘*findZipCode()*’

A função ‘*findZipCode()*’, dado um *zip code*, percorre toda a árvore (*root*) e quando encontra um *node* com o *zip code* desejado lista-o. Isto acontece até não existirem mais *nodes* a comparar.

```
135   //
136   // list the people with a given zip code
137   //
138
139   int c2 = 1; // global variable
140   void findZipCode(tree_node_t *root, char *zip_code)
141   {
142       if (root != NULL)
143       {
144           if (strcmp(root->zip_code, zip_code) == 0)
145           {
146               findZipCode(root->left[1], zip_code);
147               printf("Person #%d\n", c2++);
148               printf("    name ---------------- %s\n", root->name);
149               printf("    zip code ------------ %s\n", root->zip_code);
150               printf("    telephone number --- %s\n", root->telephone_number);
151               printf("    cc ------------------ %s\n", root->cc);
152               findZipCode(root->right[1], zip_code);
153           }
154           else
155           {
156               findZipCode(root->left[1], zip_code);
157               findZipCode(root->right[1], zip_code);
158           }
159       }
160   }
```

'main()'

```
//
// main program
//

int main(int argc, char **argv)
{
    double dt;
    // process the command line arguments
    if (argc < 3)
    {
        fprintf(stderr, "Usage: %s student_number number_of_people [options ...]\n", argv[0]);
        fprintf(stderr, "Recognized options:\n");
        fprintf(stderr, "  -list[N]              # list the tree contents, sorted by key index N (the default is index 0)\n");
        // place a description of your own options here
        fprintf(stderr, "  -find 'zip code'      # list the people with a given zip code\n");
        return 1;
    }
    int student_number = atoi(argv[1]);
    if (student_number < 1 || student_number >= 1000000)
    {
        fprintf(stderr, "Bad student number (%d) --- must be an integer belonging to [1,1000000[\n", student_number);
        return 1;
    }
    int n_people = atoi(argv[2]);
    if (n_people < 3 || n_people > 10000000)
    {
        fprintf(stderr, "Bad number of people (%d) --- must be an integer belonging to [3,10000000]\n", n_people);
        return 1;
    }
    // generate all data
    tree_node_t *people = (tree_node_t *)calloc((size_t)n_people, sizeof(tree_node_t));
    if (people == NULL)
    {
        fprintf(stderr, "Output memory!\n");
        return 1;
    }
    aed_srandom(student_number);
    for (int i = 0; i < n_people; i++)
    {
        random_name(&(people[i].name[0]));
        random_zip_code(&(people[i].zip_code[0]));
        random_telephone_number(&(people[i].telephone_number[0]));
        random_cc(&(people[i].cc[0]));
        for (int j = 0; j < 4; j++)
            people[i].left[j] = people[i].right[j] = NULL; // make sure the pointers are initially NULL
    }
    // create the ordered binary trees
    dt = cpu_time();
    tree_node_t *roots[4]; // three indices, three roots
    for (int main_index = 0; main_index < 4; main_index++)
        roots[main_index] = NULL;
    for (int i = 0; i < n_people; i++)
        for (int main_index = 0; main_index < 4; main_index++)
            tree_insert(roots, &people[i], main_index); // place your code here to insert &(people[i]) in the tree with number main_index
    dt = cpu_time() - dt;
    printf("Tree creation time (%d people): %.3es\n", n_people, dt);
    // search the tree
    for (int main_index = 0; main_index < 4; main_index++)
    {
        dt = cpu_time();
        for (int i = 0; i < n_people; i++)
        {
            tree_node_t n = people[i];                                  // make a copy of the node data
            if (find(roots[main_index], n, main_index) != &(people[i])) // place your code here to find a given person, searching for it using the tree with number main_index
            {
                fprintf(stderr, "person %d not found using index %d\n", i, main_index);
                return 1;
            }
        }
        dt = cpu_time() - dt;
        printf("Tree search time (%d people, index %d): %.3es\n", n_people, main_index, dt);
    }
    for (int main_index = 0; main_index < 4; main_index++)
    {
        dt = cpu_time();
        int depth = tree_depth(roots[main_index], main_index); // place your code here to compute the depth of the tree with number main_index
        dt = cpu_time() - dt;
        printf("Tree depth for index %d: %d (done in %.3es)\n", main_index, depth, dt);
    }
    // process the command line optional arguments
    for (int i = 3; i < argc; i++)
    {
        if (strncmp(argv[i], "-list", 5) == 0)
        { // list all (optional)
            int main_index = atoi(&(argv[i][5]));
            if (main_index < 0)
                main_index = 0;
            if (main_index > 3)
                main_index = 3;
            printf("List of people:\n");
            list(roots[main_index], main_index); // place your code here to traverse, in order, the tree with number main_index
        }
```

```
254         // place your own options here
255         else if (strcmp(argv[i], "-find") == 0)
256         {
257             printf("List of people with the zip code: %s\n", argv[i + 1]);
258             findZipCode(roots[1], argv[i + 1]);
259         }
260     }
261     // clean up --- don't forget to test your program with valgrind, we don't want any memory leaks
262     free(people);
263     return 0;
264 }
```

Output

- *Student_number* = 103360 :

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 103360 10000000
Tree creation time (10000000 people): 5.682e+01s
Tree search time (10000000 people, index 0): 1.366e+01s
Tree search time (10000000 people, index 1): 1.576e+01s
Tree search time (10000000 people, index 2): 1.509e+01s
Tree search time (10000000 people, index 3): 1.387e+01s
Tree depth for index 0: 59 (done in 5.332e-01s)
Tree depth for index 1: 61 (done in 5.414e-01s)
Tree depth for index 2: 57 (done in 5.934e-01s)
Tree depth for index 3: 57 (done in 5.988e-01s)
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 103360 4 -list0
Tree creation time (4 people): 1.535e-06s
Tree search time (4 people, index 0): 8.300e-07s
Tree search time (4 people, index 1): 7.660e-07s
Tree search time (4 people, index 2): 7.030e-07s
Tree search time (4 people, index 3): 7.400e-07s
Tree depth for index 0: 3 (done in 5.210e-07s)
Tree depth for index 1: 4 (done in 4.800e-07s)
Tree depth for index 2: 3 (done in 4.510e-07s)
Tree depth for index 3: 3 (done in 4.810e-07s)
List of people:
Person #1
    name --------------- Ethelyn Barry
    zip code ----------- 95828 Sacramento (Sacramento county)
    telephone number --- 9030 741 502
    cc ----------------- 76297810
Person #2
    name --------------- Lisa Hernandez
    zip code ----------- 11368 Corona (Queens county)
    telephone number --- 8327 821 401
    cc ----------------- 53708478
Person #3
    name --------------- Mark Jenkins
    zip code ----------- 60629 Chicago (Cook county)
    telephone number --- 4438 898 422
    cc ----------------- 32264209
Person #4
    name --------------- Michael Ramirez
    zip code ----------- 33027 Hollywood (Broward county)
    telephone number --- 9658 733 399
    cc ----------------- 34477417
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 103360 4 -list1
Tree creation time (4 people): 6.297e-06s
Tree search time (4 people, index 0): 4.281e-06s
Tree search time (4 people, index 1): 3.495e-06s
Tree search time (4 people, index 2): 3.235e-06s
Tree search time (4 people, index 3): 3.467e-06s
Tree depth for index 0: 3 (done in 2.440e-06s)
Tree depth for index 1: 4 (done in 2.369e-06s)
Tree depth for index 2: 3 (done in 2.208e-06s)
Tree depth for index 3: 3 (done in 2.233e-06s)
List of people:
Person #1
    name --------------- Lisa Hernandez
    zip code ----------- 11368 Corona (Queens county)
    telephone number --- 8327 821 401
    cc ----------------- 53708478
Person #2
    name --------------- Michael Ramirez
    zip code ----------- 33027 Hollywood (Broward county)
    telephone number --- 9658 733 399
    cc ----------------- 34477417
Person #3
    name --------------- Mark Jenkins
    zip code ----------- 60629 Chicago (Cook county)
    telephone number --- 4438 898 422
    cc ----------------- 32264209
Person #4
    name --------------- Ethelyn Barry
    zip code ----------- 95828 Sacramento (Sacramento county)
    telephone number --- 9030 741 502
    cc ----------------- 76297810
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 103360 4 -list2
Tree creation time (4 people): 6.310e-06s
Tree search time (4 people, index 0): 5.813e-06s
Tree search time (4 people, index 1): 5.055e-06s
Tree search time (4 people, index 2): 4.614e-06s
Tree search time (4 people, index 3): 4.899e-06s
Tree depth for index 0: 3 (done in 3.150e-06s)
Tree depth for index 1: 4 (done in 3.043e-06s)
Tree depth for index 2: 3 (done in 6.882e-06s)
Tree depth for index 3: 3 (done in 2.491e-06s)
List of people:
Person #1
    name ---------- Mark Jenkins
    zip code ---------- 60629 Chicago (Cook county)
    telephone number --- 4438 898 422
    cc ---------- 32264209
Person #2
    name ---------- Lisa Hernandez
    zip code ---------- 11368 Corona (Queens county)
    telephone number --- 8327 821 401
    cc ---------- 53708478
Person #3
    name ---------- Ethelyn Barry
    zip code ---------- 95828 Sacramento (Sacramento county)
    telephone number --- 9030 741 502
    cc ---------- 76297810
Person #4
    name ---------- Michael Ramirez
    zip code ---------- 33027 Hollywood (Broward county)
    telephone number --- 9658 733 399
    cc ---------- 34477417
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 103360 4 -list3
Tree creation time (4 people): 6.001e-06s
Tree search time (4 people, index 0): 4.489e-06s
Tree search time (4 people, index 1): 3.823e-06s
Tree search time (4 people, index 2): 3.683e-06s
Tree search time (4 people, index 3): 3.620e-06s
Tree depth for index 0: 3 (done in 2.450e-06s)
Tree depth for index 1: 4 (done in 2.281e-06s)
Tree depth for index 2: 3 (done in 2.193e-06s)
Tree depth for index 3: 3 (done in 2.297e-06s)
List of people:
Person #1
    name ---------- Mark Jenkins
    zip code ---------- 60629 Chicago (Cook county)
    telephone number --- 4438 898 422
    cc ---------- 32264209
Person #2
    name ---------- Michael Ramirez
    zip code ---------- 33027 Hollywood (Broward county)
    telephone number --- 9658 733 399
    cc ---------- 34477417
Person #3
    name ---------- Lisa Hernandez
    zip code ---------- 11368 Corona (Queens county)
    telephone number --- 8327 821 401
    cc ---------- 53708478
Person #4
    name ---------- Ethelyn Barry
    zip code ---------- 95828 Sacramento (Sacramento county)
    telephone number --- 9030 741 502
    cc ---------- 76297810
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 103360 1500 -find "11206 Brooklyn (Kings county)"
Tree creation time (1500 people): 9.251e-04s
Tree search time (1500 people, index 0): 3.924e-04s
Tree search time (1500 people, index 1): 4.129e-04s
Tree search time (1500 people, index 2): 4.279e-04s
Tree search time (1500 people, index 3): 3.925e-04s
Tree depth for index 0: 21 (done in 4.496e-05s)
Tree depth for index 1: 27 (done in 2.960e-05s)
Tree depth for index 2: 23 (done in 4.530e-05s)
Tree depth for index 3: 22 (done in 3.720e-05s)
List of people with the zip code: 11206 Brooklyn (Kings county)
Person #1
    name ---------- Betty Le
    zip code ---------- 11206 Brooklyn (Kings county)
    telephone number --- 4569 063 154
    cc ---------- 72406663
Person #2
    name ---------- Jacalyn Kim
    zip code ---------- 11206 Brooklyn (Kings county)
    telephone number --- 5122 296 196
    cc ---------- 31927034
Person #3
    name ---------- Bobbie Cole
    zip code ---------- 11206 Brooklyn (Kings county)
    telephone number --- 5833 122 918
    cc ---------- 76578885
Person #4
    name ---------- Andrew Murray
    zip code ---------- 11206 Brooklyn (Kings county)
    telephone number --- 7475 441 241
    cc ---------- 01606069
Person #5
    name ---------- Gene Ahmed
    zip code ---------- 11206 Brooklyn (Kings county)
    telephone number --- 7505 344 933
    cc ---------- 07875628
```

- *Student_number* = 102851 :

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 102851 10000000
Tree creation time (10000000 people): 5.949e+01s
Tree search time (10000000 people, index 0): 1.478e+01s
Tree search time (10000000 people, index 1): 1.797e+01s
Tree search time (10000000 people, index 2): 1.619e+01s
Tree search time (10000000 people, index 3): 1.458e+01s
Tree depth for index 0: 57 (done in 5.584e-01s)
Tree depth for index 1: 58 (done in 5.327e-01s)
Tree depth for index 2: 59 (done in 6.338e-01s)
Tree depth for index 3: 59 (done in 6.124e-01s)
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 102851 4 -list0
Tree creation time (4 people): 1.633e-06s
Tree search time (4 people, index 0): 9.430e-07s
Tree search time (4 people, index 1): 7.860e-07s
Tree search time (4 people, index 2): 7.530e-07s
Tree search time (4 people, index 3): 7.360e-07s
Tree depth for index 0: 4 (done in 5.440e-07s)
Tree depth for index 1: 4 (done in 6.000e-07s)
Tree depth for index 2: 4 (done in 5.020e-07s)
Tree depth for index 3: 3 (done in 8.220e-07s)
List of people:
Person #1
    name ---------- Clair Rodriguez
    zip code ---------- 92804 Anaheim (Orange county)
    telephone number --- 1963 704 614
    cc ---------- 07000733
Person #2
    name ---------- Mary Paul
    zip code ---------- 37211 Nashville (Davidson county)
    telephone number --- 3146 767 353
    cc ---------- 78085756
Person #3
    name ---------- Rita Hensley
    zip code ---------- 89031 North Las Vegas (Clark county)
    telephone number --- 9413 320 621
    cc ---------- 87162912
Person #4
    name ---------- Shannon Christian
    zip code ---------- 90201 Bell (Los Angeles county)
    telephone number --- 6189 678 649
    cc ---------- 48368067
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 102851 4 -list1
Tree creation time (4 people): 1.493e-05s
Tree search time (4 people, index 0): 4.491e-06s
Tree search time (4 people, index 1): 3.892e-06s
Tree search time (4 people, index 2): 3.802e-06s
Tree search time (4 people, index 3): 3.759e-06s
Tree depth for index 0: 4 (done in 2.730e-06s)
Tree depth for index 1: 4 (done in 2.605e-06s)
Tree depth for index 2: 4 (done in 2.586e-06s)
Tree depth for index 3: 3 (done in 2.491e-06s)
List of people:
Person #1
    name ---------- Mary Paul
    zip code ---------- 37211 Nashville (Davidson county)
    telephone number --- 3146 767 353
    cc ---------- 78085756
Person #2
    name ---------- Rita Hensley
    zip code ---------- 89031 North Las Vegas (Clark county)
    telephone number --- 9413 320 621
    cc ---------- 87162912
Person #3
    name ---------- Shannon Christian
    zip code ---------- 90201 Bell (Los Angeles county)
    telephone number --- 6189 678 649
    cc ---------- 48368067
Person #4
    name ---------- Clair Rodriguez
    zip code ---------- 92804 Anaheim (Orange county)
    telephone number --- 1963 704 614
    cc ---------- 07000733
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 102851 4 -list2
Tree creation time (4 people): 7.051e-06s
Tree search time (4 people, index 0): 5.183e-06s
Tree search time (4 people, index 1): 4.269e-06s
Tree search time (4 people, index 2): 4.242e-06s
Tree search time (4 people, index 3): 4.103e-06s
Tree depth for index 0: 4 (done in 2.684e-06s)
Tree depth for index 1: 4 (done in 2.605e-06s)
Tree depth for index 2: 4 (done in 2.576e-06s)
Tree depth for index 3: 3 (done in 2.526e-06s)
List of people:
Person #1
    name --------------- Clair Rodriguez
    zip code ----------- 92804 Anaheim (Orange county)
    telephone number --- 1963 704 614
    cc ----------------- 07000733
Person #2
    name --------------- Mary Paul
    zip code ----------- 37211 Nashville (Davidson county)
    telephone number --- 3146 767 353
    cc ----------------- 78085756
Person #3
    name --------------- Shannon Christian
    zip code ----------- 90201 Bell (Los Angeles county)
    telephone number --- 6189 678 649
    cc ----------------- 48368067
Person #4
    name --------------- Rita Hensley
    zip code ----------- 89031 North Las Vegas (Clark county)
    telephone number --- 9413 320 621
    cc ----------------- 87162912
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 102851 4 -list3
Tree creation time (4 people): 4.714e-06s
Tree search time (4 people, index 0): 3.076e-06s
Tree search time (4 people, index 1): 2.650e-06s
Tree search time (4 people, index 2): 2.525e-06s
Tree search time (4 people, index 3): 2.560e-06s
Tree depth for index 0: 4 (done in 1.797e-06s)
Tree depth for index 1: 4 (done in 1.773e-06s)
Tree depth for index 2: 4 (done in 1.723e-06s)
Tree depth for index 3: 3 (done in 1.696e-06s)
List of people:
Person #1
    name --------------- Clair Rodriguez
    zip code ----------- 92804 Anaheim (Orange county)
    telephone number --- 1963 704 614
    cc ----------------- 07000733
Person #2
    name --------------- Shannon Christian
    zip code ----------- 90201 Bell (Los Angeles county)
    telephone number --- 6189 678 649
    cc ----------------- 48368067
Person #3
    name --------------- Mary Paul
    zip code ----------- 37211 Nashville (Davidson county)
    telephone number --- 3146 767 353
    cc ----------------- 78085756
Person #4
    name --------------- Rita Hensley
    zip code ----------- 89031 North Las Vegas (Clark county)
    telephone number --- 9413 320 621
    cc ----------------- 87162912
```

```
leonardodsf@leonardodsf-Creator-15M-A9SD:~/Documents/GitHub/LEI/AED/Multi-ordered_trees$ ./multi_ordered_tree 102851 1500 -find "11206 Brooklyn (Kings county)"
Tree creation time (1500 people): 8.409e-04s
Tree search time (1500 people, index 0): 2.212e-04s
Tree search time (1500 people, index 1): 2.709e-04s
Tree search time (1500 people, index 2): 2.261e-04s
Tree search time (1500 people, index 3): 2.098e-04s
Tree depth for index 0: 21 (done in 1.729e-05s)
Tree depth for index 1: 23 (done in 1.666e-05s)
Tree depth for index 2: 23 (done in 1.760e-05s)
Tree depth for index 3: 24 (done in 1.737e-05s)
List of people with the zip code: 11206 Brooklyn (Kings county)
Person #1
    name --------------- Anthony Johnson
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 1784 026 128
    cc ----------------- 43307230
Person #2
    name --------------- Matthew Daniel
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 3623 063 360
    cc ----------------- 81437014
Person #3
    name --------------- Sheila Jordan
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 4239 644 982
    cc ----------------- 85172369
Person #4
    name --------------- John Oliver
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 7637 016 372
    cc ----------------- 82112637
Person #5
    name --------------- Donna Dawson
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 8022 739 696
    cc ----------------- 72691211
Person #6
    name --------------- Albert Alexander
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 8441 932 530
    cc ----------------- 16667200
Person #7
    name --------------- Marie Watson
    zip code ----------- 11206 Brooklyn (Kings county)
    telephone number --- 9776 101 715
    cc ----------------- 10567965
```

Código MATLAB

histogram_generator.m

```
A1 = load ("statistics_data/depthIndex0.txt");
B1 = load ("statistics_data/depthIndex1.txt");
C1 = load ("statistics_data/depthIndex2.txt");
D1 = load ("statistics_data/depthIndex3.txt");
binc1=(35:56);
o0 = hist(A1,binc1);
o1 = hist(B1,binc1);
o2 = hist(C1,binc1);
o3 = hist(D1,binc1);
figure(1)
x = 35:1:56;
y = [o0;o1;o2;o3];
bar(x,y)
set(gca,'xscale','log');
set(gca,'yscale','log');
xlim([35 56])
grid on;
xlabel('Maximum Depth')
ylabel('Number of Occurrences')
legend('Index0','Index1','Index2', 'Index3')
A2 = readmatrix("statistics_data/creationTime.txt");
figure(2);
histogram(A2,'BinWidth',0.0001,EdgeColor="r")
title("Tree creation time (100000 people and 10000 nMecs) Histogram");
xlabel("Creation Time");
ylabel("Ocurrences");

A3 = load("statistics_data/searchTime0.txt");
B3 = load("statistics_data/searchTime1.txt");
C3 = load("statistics_data/searchTime2.txt");
D3 = load("statistics_data/searchTime3.txt");
figure(3)
binc3 = (0.048:0.0001:0.090);
histogram(A3,'BinWidth',0.00001,EdgeColor="r")
hold on;
histogram(B3,'BinWidth',0.00001,EdgeColor="g")
hold on;
histogram(C3,'BinWidth',0.00001,EdgeColor="b")
hold on;
histogram(D3,'BinWidth',0.00001,EdgeColor="y")
title("Search time (100000 people and 10000 nMecs) Histogram");
legend('Index0','Index1','Index2', 'Index3')
```

graph_generator.m

```
%% Creation Time
C1 = readmatrix("102851/data_creation_time.txt");
C2 = readmatrix("103360/data_creation_time.txt");
figure(1);
scatter(C1(1:28,1),C1(1:28,2),'black','filled');
grid on;
title("Tree creation time - 102851");
xlabel("Número de pessoas");
ylabel("Tempo (segundos)");
set(gca,'xscale','log');
set(gca,'yscale','log');
figure(2);
scatter(C2(1:28,1),C2(1:28,2),'black','filled');
grid on;
title("Tree creation time - 103360");
xlabel("Número de pessoas");
ylabel("Tempo (segundos)");
set(gca,'xscale','log');
set(gca,'yscale','log');
```

```
%% Search Time
S1 = readmatrix("102851/data_search_time0.txt");
S2 = readmatrix("102851/data_search_time1.txt");
S3 = readmatrix("102851/data_search_time2.txt");
S4 = readmatrix("102851/data_search_time3.txt");
figure(3);
scatter(S1(:,1),S1(:,2),'yellow','filled')
hold on;
scatter(S2(:,1),S2(:,2),'red','filled');
hold on;
scatter(S3(:,1),S3(:,2),'blue','filled')
hold on;
scatter(S4(:,1),S4(:,2),'green','filled')
set(gca,'xscale','log');
set(gca,'yscale','log');
grid on;
title("Tree Search time - 102851");
xlabel("Número de pessoas");
ylabel("Tempo (segundos)");
S5 = readmatrix("103360/data_search_time0.txt");
S6 = readmatrix("103360/data_search_time1.txt");
S7 = readmatrix("103360/data_search_time2.txt");
S8 = readmatrix("103360/data_search_time3.txt");
figure(4);
scatter(S5(:,1),S5(:,2),'yellow','filled')
hold on;
scatter(S6(:,1),S6(:,2),'red','filled');
hold on;
scatter(S7(:,1),S7(:,2),'blue','filled')
hold on;
scatter(S8(:,1),S8(:,2),'green','filled')
set(gca,'xscale','log');
set(gca,'yscale','log');
grid on;
title("Tree Search time - 103360");
xlabel("Número de pessoas");
ylabel("Tempo (segundos)");
```

```
%% Tree depth
S1 = readmatrix("102851/treedepth0.txt");
S2 = readmatrix("102851/treedepth1.txt");
S3 = readmatrix("102851/treedepth2.txt");
S4 = readmatrix("102851/treedepth3.txt");
figure(3);
scatter(S1(:,1),S1(:,2),'yellow','filled')
hold on;
scatter(S2(:,1),S2(:,2),'red','filled');
hold on;
scatter(S3(:,1),S3(:,2),'blue','filled')
hold on;
scatter(S4(:,1),S4(:,2),'green','filled')
set(gca,'xscale','log');
grid on;
title("Tree depth - 102851");
xlabel("Número de pessoas");
ylabel("Depth");
S5 = readmatrix("103360/treedepth0.txt");
S6 = readmatrix("103360/treedepth1.txt");
S7 = readmatrix("103360/treedepth2.txt");
S8 = readmatrix("103360/treedepth3.txt");
figure(4);
scatter(S5(:,1),S5(:,2),'yellow','filled')
hold on;
scatter(S6(:,1),S6(:,2),'red','filled');
hold on;
scatter(S7(:,1),S7(:,2),'blue','filled')
hold on;
scatter(S8(:,1),S8(:,2),'green','filled')
set(gca,'xscale','log');
grid on;
title("Tree depth - 103360");
xlabel("Número de pessoas");
ylabel("Depth");
```

Conclusão

Em suma, este trabalho ajudou-nos a uma melhor percepção de métodos de criação de código e estruturas de dados, bem como a implementação das Binary Trees. Enriquecemos também os nossos conhecimentos da linguagem de programação C, na gestão de tempo e no trabalho em equipa.

Bibliografia

- Lecture notes da disciplina
- https://www.geeksforgeeks.org/tree-sort/