MULTI-ORDERED TREES

ALGORITMOS E ESTRUTURAS DE DADOS 2021

Prof. Tomás Silva

Pedro Rasinhas – 103541 - 33%

Guilherme Antunes – 103600 – 33%

Daniel Ferreira – 102442 – 33%

ÍNDICE

Introdução

Como objetivo deste segundo trabalho prático pretende-se criar um programa que é capaz de gerar (aleatoriamente), guardar e organizar informação referente a diferentes *n* indivíduos, cada um com as seguintes características:

- **Nome;**
- ***Zip Code;***
- **Número de telefone;**
- **Número de Segurança Social (adicionado posteriormente, como trabalho 'extra');**

A estrutura de dados usada para guardar essa informação foi um conjunto de árvores binárias, onde cada nó dessas árvores continha a informação referente a cada pessoa.

Na sua forma mais simples, as árvores binárias são estruturas de dados caracterizadas por uma raiz e dois ponteiros, um para a subárvore da esquerda, e outro para a subárvore da direita.

No nosso caso, como foram usadas árvores binárias múltiplas, onde cada nó corresponde a uma pessoa, e, portanto, contém as suas características (Nome, *Zip Code,* Número de Telefone e Número de Segurança Social), tivemos que trabalhar com 2 *arrays* de ponteiros, [com 4 ponteiros em cada *array* (um para cada tipo de dados referente a cada pessoa)], um que aponta para o nó da esquerda, e outro para o nó da direita.

Sucintamente, o trabalho desenvolvido baseia-se em vários tipos de organização da informação (por ordem alfabética, por *Zip Code* (por ordem crescente), por número de telefone, ou por número de Segurança Social) e na pesquisa de qualquer tipo de dados referente a cada indivíduo: pesquisa através de um dado *Zip Code,* pesquisa através do nome completo ou do primeiro nome, pesquisa através do número de telefone e pesquisa através do número de Segurança Social.

Metodologia

Procedeu-se à implementação de várias funções com vista à construção da estrutura de dados e consequente tratamento e análise das referidas árvores.

Tree_insert **– Função recursiva que procede à criação e inserção dum novo nó, com a informação referente a uma pessoa, na árvore, sendo que, logicamente, na ausência de outros nós, é criada a árvore com apenas um nó.**

find **- função recursiva com o objetivo de procurar pessoas na árvore.**

Tree_depth **- função recursiva que calcula a profundidade da árvore.**

list **- função que imprime os dados de uma árvore por ordem crescente.**

search **- função que permite procurar uma pessoa por um determinado tipo de informação, que pode não estar completa.**

nodeCounter **- função que permite contar o número de nós em cada nível de uma árvore.**

Função *Tree_insert:*

A função *Tree_insert* trata-se do método recursivo utilizado para a inserção de um novo nó na árvore.

Verificada se a árvore não estiver vazia (caso contrário, é criada com o nó a inserir a assumir a função de raiz e conclui-se o processo de inserção desse nó), compara-se, recorrendo à função *compare_tree_nodes* (já fornecida), o valor do nó a inserir com um dado nó já inserido na árvore. De notar que esta comparação é feita utilizando até 4 níveis, começando no valor de Nome e indo até ao valor do número de Segurança Social, avançando para o próximo critério apenas se ocorrer um empate no critério atual.

Seguindo as regras estruturais duma árvore binária (valores menores à esquerda e maiores à direita) averigua-se a relação com o nó em análise e, caso ainda possua filhos do lado em que o nó a inserir seria colocado, continua o processo até encontrar um espaço vazio, onde irá proceder à inserção.

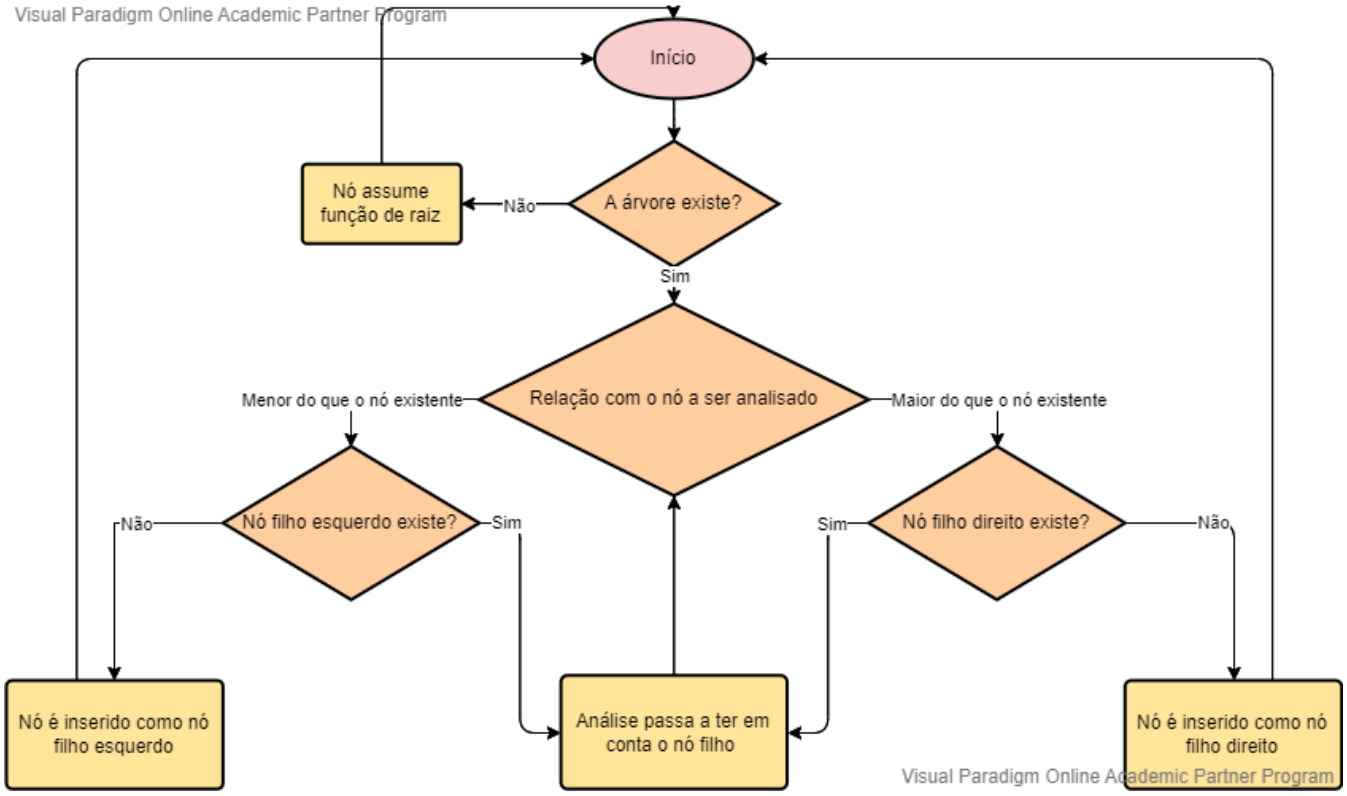

Figura 1 – Fluxo de execução da função *Tree_insert*

Função *find:*

A função *find* permite procurar na árvore uma pessoa passada como argumento. A função percorre a árvore, comparando um dado nó em análise com o que é desejado, através da função *compare_tree_nodes,* até que o nó em análise seja igual ao desejado.

Função *Tree_depth:*

A função recursiva *Tree_depth* permite o cálculo da profundidade máxima da árvore, ou seja, o número máximo de arestas entre a raiz e um nó. Para este efeito, a função recebe o nó raiz da árvore e o índice atual, assegura que o nó não é vazio, calcula as profundidades dos nós filhos (através duma repetição recursiva do processo acima descrito) e devolve a soma do maior valor encontrado com 1, visto que a distância entre a raiz e o nó mais profundo é igual à profundidade máxima dos nós filho da raiz somada à distância entre a raiz e esse nó filho: 1.

No exemplo abaixo, esta função encontraria todas as profundidades possíveis dos filhos (0, 1 e 2) e devolveria 3, visto que a maior profundidade é alcançada através da sequência 20 -> 8 -> 12 -> 14.

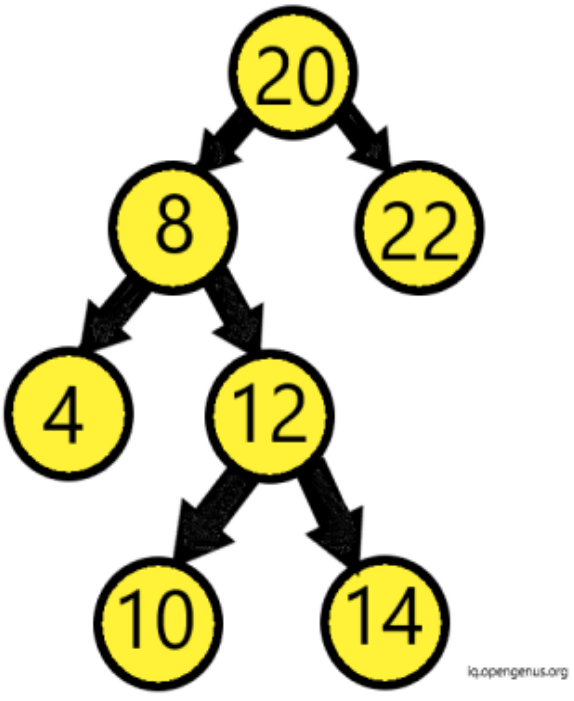

Figura 2 – Exemplo de uma árvore binária

Função *list:*

A função recursiva *list* permite imprimir a informação relativa aos nós presentes numa árvore. São percorridos e impressos os nós à esquerda da raiz, a raiz e por fim os nós à direita, conseguindo assim apresentar toda a informação.

Função *search:*

A função search funciona da mesma maneira que a função *list*, em termos de recursividade. Esta função permite imprimir todas as pessoas com um determinado nome, ou apenas com um determinado primeiro nome, com um certo *ZipCode,* com um certo número de telefone, e também com um determinado número de Segurança Social. Para este funcionamento, é passada à função como argumento uma string, que é lida no terminal, e uma *flag,* para indicar o tipo de dado segundo o qual queremos pesquisar. Sucintamente, é uma função *list* "melhorada", onde apenas vai ser impresso a informação que é relevante perante o tipo de dados e a *string* a pesquisar.

Função *nodeCounter:*

Esta função serve, sucintamente, para contar o número de nós que cada nível da árvore tem de uma maneira recursiva. Utiliza um vetor para guardar o número de nós encontrados em cada nível, incrementando na posição *depth* sempre que percorrer um nó nessa profundidade.

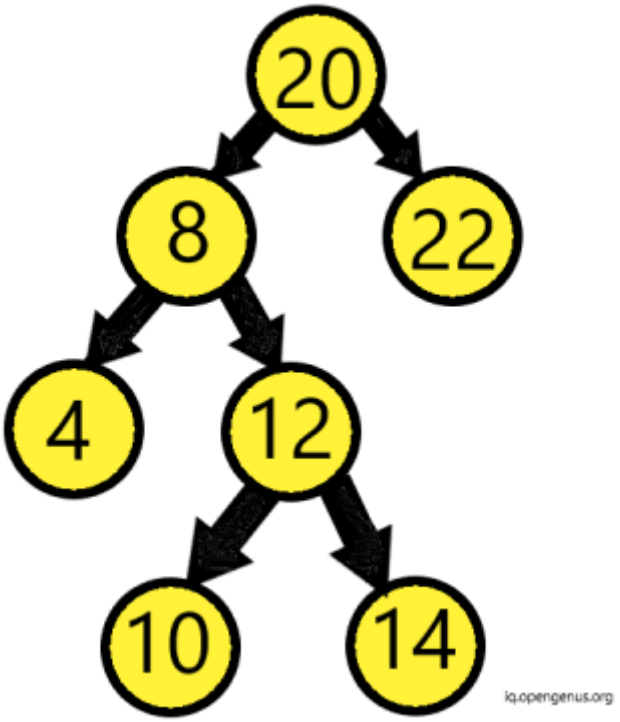

Figura 3 – Exemplo de uma árvore binária

Como exemplo ilustrativo, a árvore acima apresentada, tem 1 nó no nível zero (raíz), no nível um tem 2 nós, 2 nós no nível três e 2 no nível 4.

Resultados e algumas conclusões

Para se ter uma melhor noção de como o tempo de execução da criação da árvore, da pesquisa e da profundidade varia com o número de pessoas (e com o número mecanográfico do aluno), apresentamos, nas páginas seguintes, alguns gráficos e histogramas.

Optámos por colocar duas secções de quatro gráficos cada, uma com gráficos apenas para um dado número mecanográfico, e outra com uma ligeira variação nos números mecanográficos. Ambas as secções contêm os seguintes gráficos :

- **um a demonstrar como varia a profundidade máxima da árvore com o aumento do número de pessoas;**
- **um para denotar como o aumento do número de pessoas implica um aumento significativo no tempo de criação da árvore;**
- **um para demonstrar a variação do tempo de pesquisa, em relação ao número de pessoas;**
- **um para demonstrar como se distribuem os nós pelos vários níveis da árvore;**

Colocou-se apenas um gráfico de cada tipo, sem variar os números mecanográficos dos elementos do grupo, porque as variações que os números mecanográficos podiam trazer eram poucas ou até mesmo nulas, já que estes são valores muito próximos.

Decidimos apresentar também alguns (3) histogramas, variando os números mecanográficos de 10000 a 20000, para uma melhor compreensão da distribuição que cada índice (/para todos os índices, no caso do histograma que analisa, com a variação dos números mecanográficos, os tempos que as árvores demoram a ser formadas e o número de ocorrências) apresenta para:

- **a profundidade da árvore;**
- **o tempo de pesquisa da árvore;**
- **o tempo de criação da árvore;**

Notas:

Nos gráficos e histogramas seguintes "índice 0" refere-se ao nome, "índice 1" refere-se ao *zipCode,* "índice 2" refere-se ao número de telefone, e o "índice 3" refere-se ao número da Segurança Social.

A falta de pontos em alguns dos gráficos (Figura 6) deve-se ao facto de o tempo de execução ser arredondado para 0 segundos, e, por conseguinte, o *MatLab* não considera esses valores para o *plot*.

Gráficos (1)

- Max Tree Depth

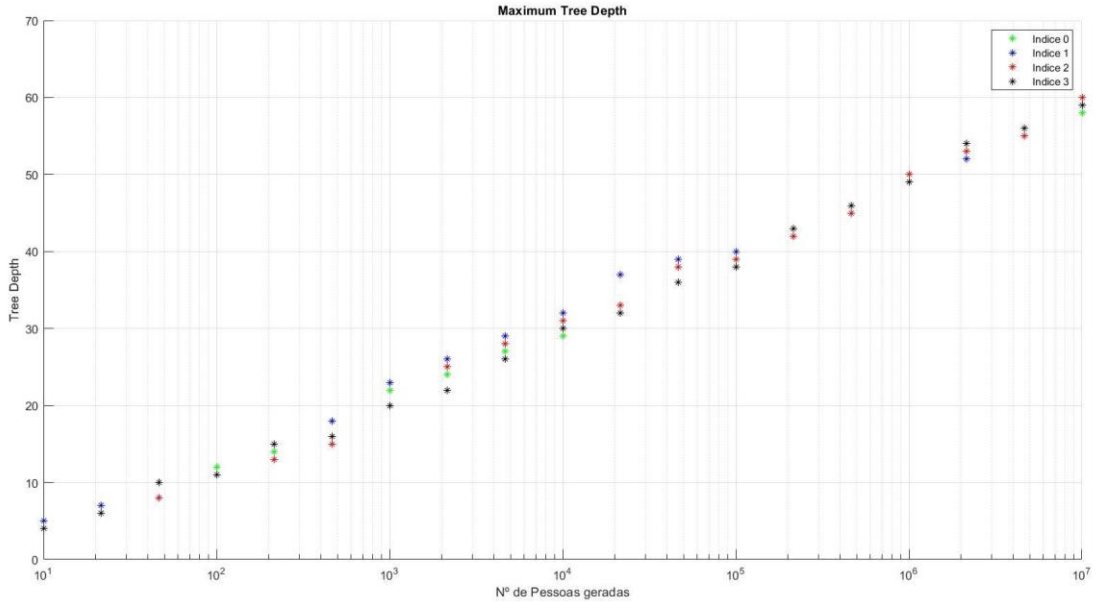

Figura 4 – Max Tree Depth (para o nº mec. 103541)

- Tree Search

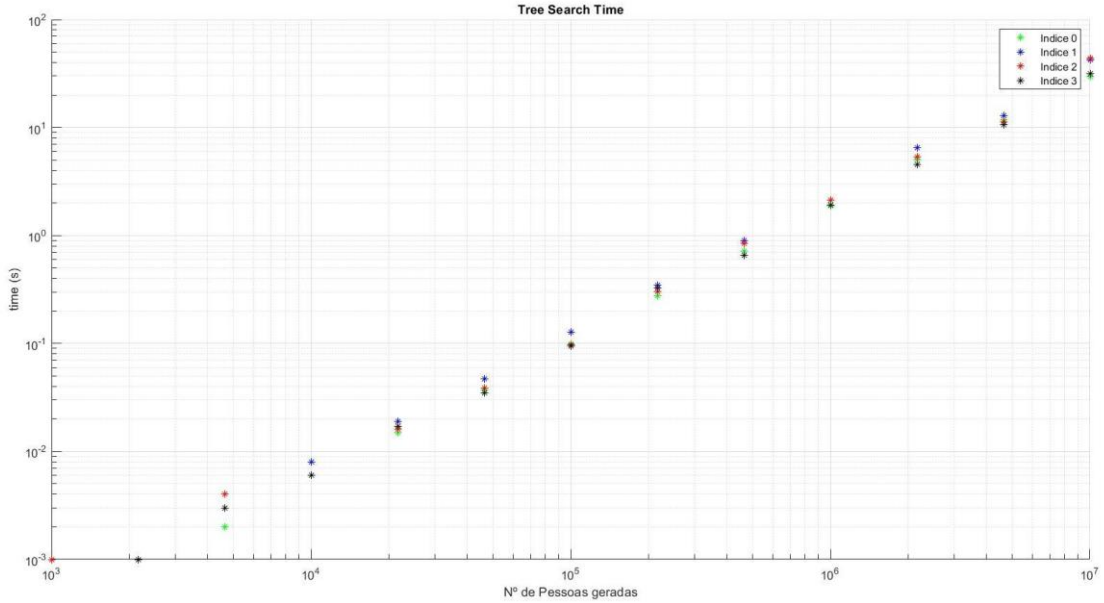

Figura 5 –Tree Search (para o nº mec. 103600)

- ## Tree Creation

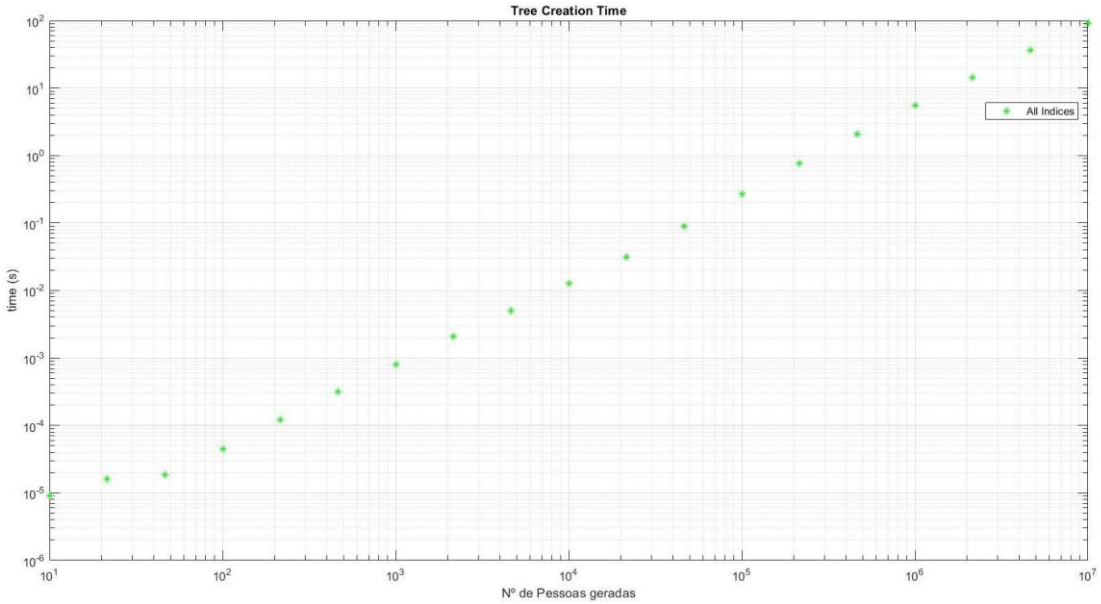

Figura 6 – Tree Creation (para o nº mec. 102442)

- ## Nodes by Level

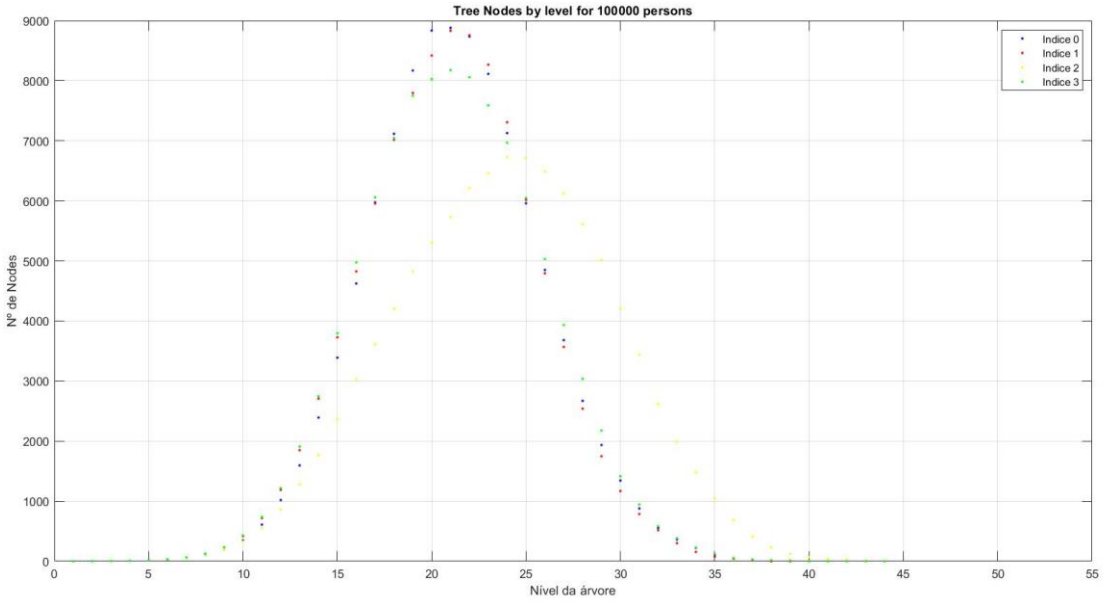

Figura 7 – Nodes by level (para o nº mec. 102442)

Gráficos (2)

[para vários números mecanográficos diferentes (10000 a 20000, de 2500 em 2500)]

- Max Tree Depth – Com variação no número Mecanográfico

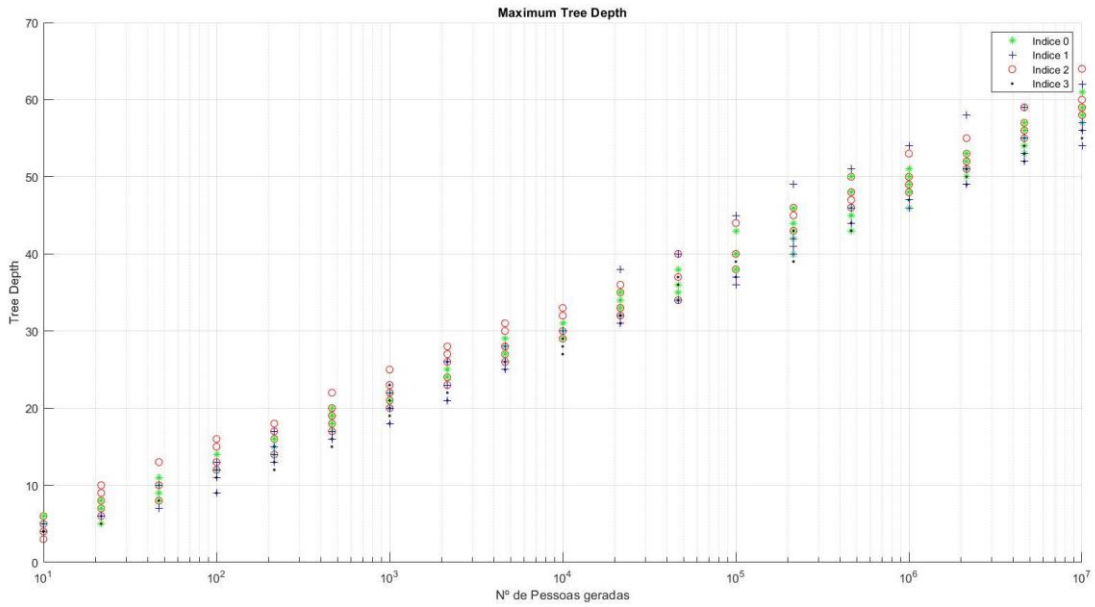

Figura 8 – Max Tree Depth (variando os nº mec)

- Tree Search – Com variação no número Mecanográfico

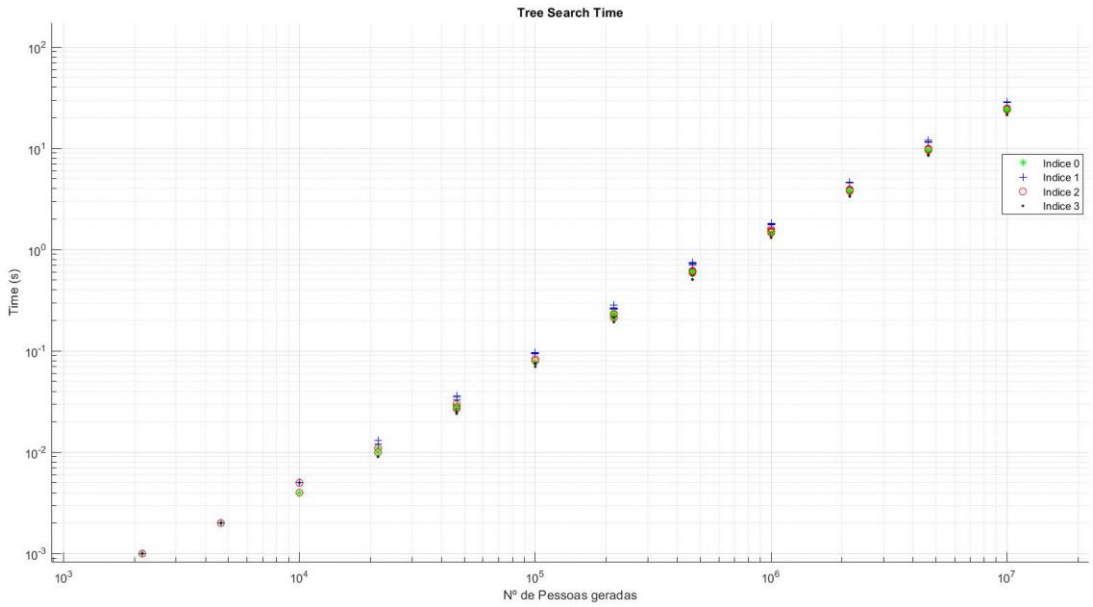

Figura 9 – Tree Search time (variando os nº mec)

- Tree Creation – Com variação no número Mecanográfico

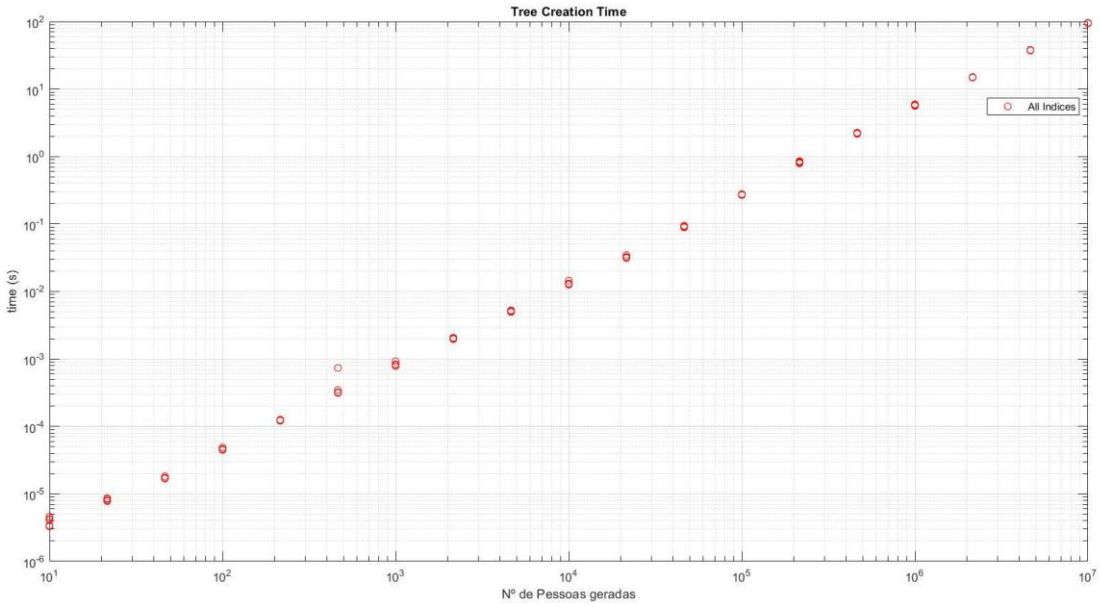

Figura 10 – Tree Creation time (variando os nº mec)

A análise dos 3 gráficos anteriores permite tirar dar conta de uma relação de proporcionalidade entre o número de nós estudado/criado e o tempo que a função em causa demora a concluir a sua execução. Atendendo ao facto de que a complexidade das funções de inserção, pesquisa e de cálculo da *tree depth* possuírem uma complexidade, em média, de O(log(n)), os resultados acima figurados eram expectáveis.

- Nodes by level – Com variação no número Mecanográfico

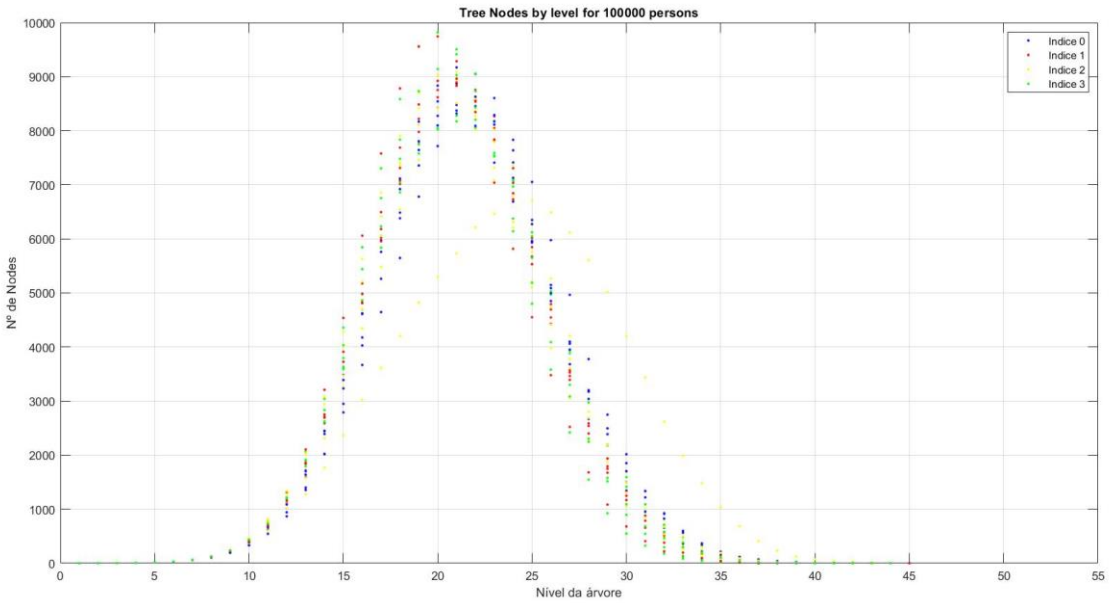

Figura 11 – Nodes by level (variando os nº mec)

Este gráfico permite concluir que o número de nós por nível em função do número de níveis segue uma distribuição Gaussiana, sendo que os níveis mais baixos (até, aproximadamente, 9) possuem um baixo nível de nós, já que o número máximo de nós por nível depende do nível (2^n nós no nível n). Já a curva descendente indica que os níveis não estão completamente preenchidos, havendo um ritmo de alargamento mais rápido da árvore em relação ao seu aprofundamento do que aquele verificado no início, já que os nós inseridos nestes níveis sofrem um número muito elevado de comparações, resultando na dispersão acrescida de dados. O afunilamento verificado a partir por volta do nível 22 deve-se ao início da escassez de pessoas a colocar, sendo que o número de nós total é o fator que dita a diminuição do número de nós por nível nesta secção da árvore.

Histogramas

- Máxima profundidade da árvore, variando os números mecanográficos

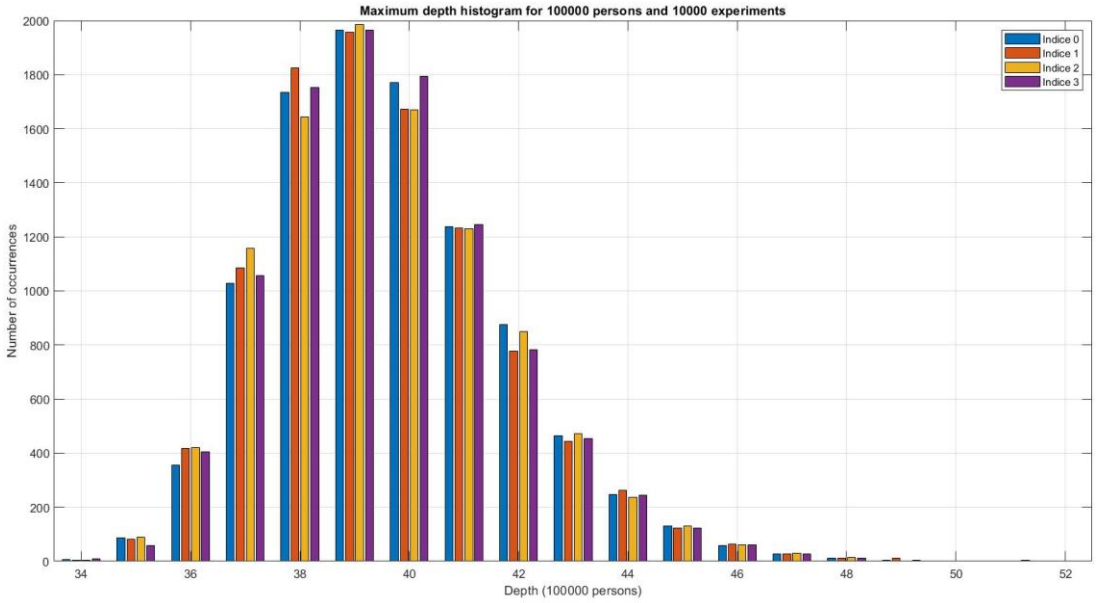

Figura 12 – Máxima profundidade para 100000 pessoas, variando os números mecanográficos

Os dados acima apresentados levam a inferir que a estrutura de dados escolhida, mesmo que não se enquadre na definição de árvore perfeita como já demonstrado na Figura 8, não possui grande variabilidade na sua profundidade máxima. Mesmo para níveis elevados de experiências, este valor enquadra-se no intervalo de [34, 48] para a esmagadora maioria dos casos, indicando que, independentemente do conjunto de dados gerado, as funções desenvolvidas têm um comportamento similar ao esperado.

- Tempo de procura da árvore, variando os números mecanográficos

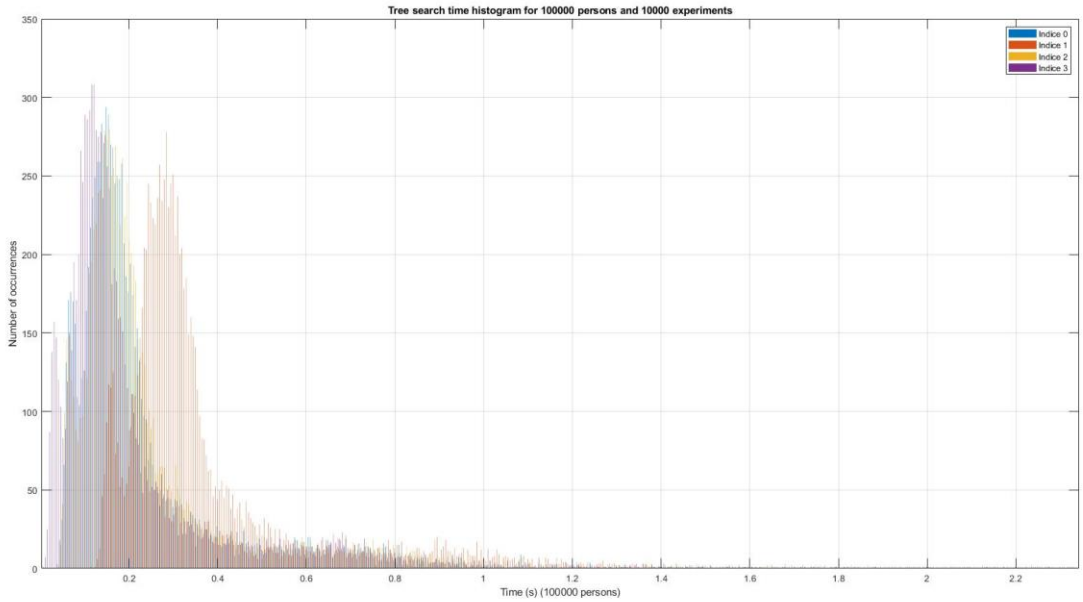

Figura 13 – Tempo de procura para 100000 pessoas, variando os números mecanográficos

- Tempo de criação da árvore, variando os números mecanográficos

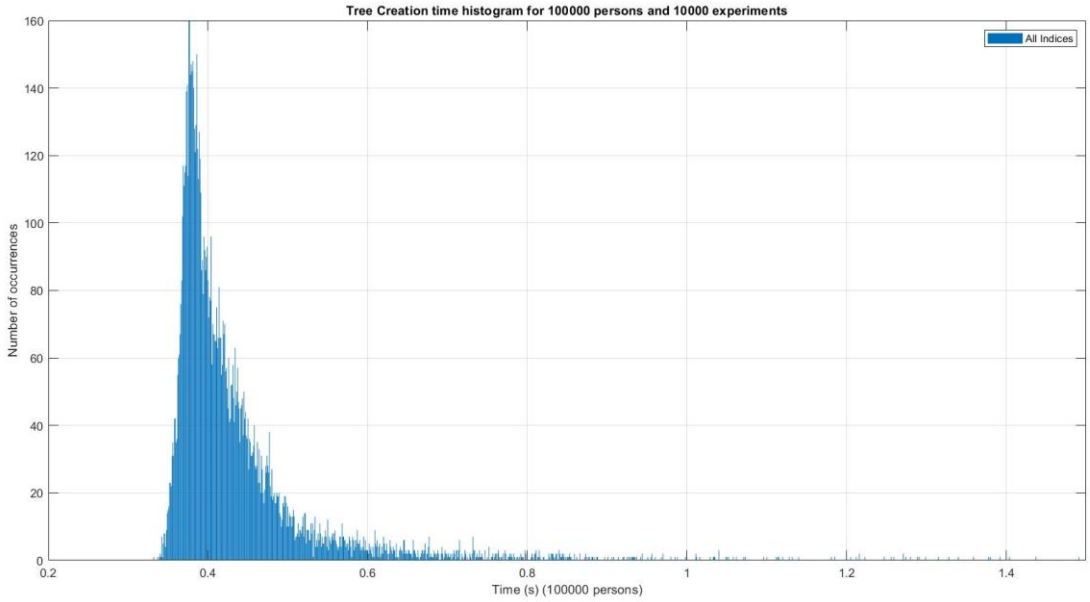

Figura 14 – Tempo de criação para 100000 pessoas, variando os números mecanográficos

Os dois gráficos acima permitem concluir que os tempos de criação e procura das árvores seguem uma distribuição similar, sendo que o valor da anterior figura mais frequentemente por volta dos 0.4s, indicando uma forte consistência no método utilizado. Já os tempos de pesquisa apresentam uma discrepância notória entre os mesmos. Em particular, o tempo no que concerne à sub-árvore dos *Zip Codes* é visivelmente mais elevado, em média, do que os restantes. Isto deve-se ao facto de existir muito menos dados nesta árvore (500) relativamente aos outros tipos de dados, levando a uma maior probabilidade de existirem valores repetidos. Por conseguinte, a função *compare_tree_nodes* (chamada pela função de pesquisa) irá apresentar um tempo mais elevado de execução, já que terá de

avaliar outras características dos nós (Nome, número de telefone, número de Segurança Social) para proceder ao desempate, aumentado assim o tempo total de pesquisa.

Demonstrações da função *search*

Execução com: ./multi_ordered_trees 103541 150 -list1 -search1

Segundo esta linha de comandos, é expectável serem mostradas 150 pessoas, ordenadas pelo *ZipCode,* e posteriormente ser pedida uma *string (ZipCode)* para voltar a mostrar apenas as pessoas com esse *zipCode.*

```
List of persons:

Nº          Name                Telephone Number    Social Sec. Number      Zip Code
1           Mark Glass          8025 107 046        037 2591 4648           10452 Bronx (Bronx county)
2           James Hobbs         8136 697 814        804 4200 8543           10462 Bronx (Bronx county)
3           Sherri Clayton      4980 574 606        481 6081 8592           10466 Bronx (Bronx county)
4           Louann Nunez        1669 010 969        015 1450 1786           10469 Bronx (Bronx county)
5           Elizabeth Morris    8049 586 262        584 3117 2172           11210 Brooklyn (Kings county)
6           Ignacio Ray         2114 441 014        615 2937 3868           11213 Brooklyn (Kings county)
7           Charles Mendoza     2436 506 888        417 8025 8894           11214 Brooklyn (Kings county)
8           Catherine Walsh     4384 214 161        141 8973 9763           11215 Brooklyn (Kings county)
9           Bernice Yoder       1421 782 160        295 9986 6536           11236 Brooklyn (Kings county)
10          Elroy Ayala         2232 573 413        706 2399 6743           11236 Brooklyn (Kings county)
11          Johnathan Johnson   3626 150 070        687 6374 4399           11236 Brooklyn (Kings county)
```

.....

```
140         Andrew Lane         2247 247 338        384 4593 1104           94538 Fremont (Alameda county)
141         Anna Jimenez        5454 777 714        361 3983 4212           94565 Pittsburg (Contra Costa county)
142         James Franklin      7983 266 970        207 5130 6090           94565 Pittsburg (Contra Costa county)
143         Steven Watson       5032 681 171        015 1535 3614           94587 Union City (Alameda county)
144         Dusty Walker        3544 143 202        653 8967 7935           95111 San Jose (Santa Clara county)
145         Wanda Coleman       9337 840 866        030 9772 5684           95112 San Jose (Santa Clara county)
146         John Solis          1758 423 811        763 4106 6011           95127 San Jose (Santa Clara county)
147         Greg Khan           4002 661 590        293 9717 7880           95206 Stockton (San Joaquin county)
148         Thomas Bryan        1898 270 095        614 2311 9515           95823 Sacramento (Sacramento county)
149         Robert Roberts      9492 864 626        802 9729 8981           95823 Sacramento (Sacramento county)
150         Floyd Martin        6738 013 801        168 8900 4614           99654 Wasilla (Matanuska-Susitna county)

A procurar por zip code...
Enter the zipCode you want to search for:
```

De seguida é inserida uma *string,* neste caso ilustrativo usamos "11236 Brooklyn (Kings county)".

```
List of persons with '11236 Brooklyn (Kings county)' (zipCode):

Nº          Name                Telephone Number    Social Sec. Number      Zip Code
1           Bernice Yoder       1421 782 160        295 9986 6536           11236 Brooklyn (Kings county)
2           Elroy Ayala         2232 573 413        706 2399 6743           11236 Brooklyn (Kings county)
3           Johnathan Johnson   3626 150 070        687 6374 4399           11236 Brooklyn (Kings county)
```

Execução com: ./multi_ordered_trees 103541 150 -list0 -search0

Segundo esta linha de comandos, é expectável serem mostradas 150 pessoas, ordenadas pelo nome, e posteriormente ser pedida uma *string (*nome) para voltar a mostrar apenas as pessoas com esse nome.

```
List of persons:

Nº          Name                Telephone Number    Social Sec. Number      Zip Code
1           Andrew Lane         2247 247 338        384 4593 1104           94538 Fremont (Alameda county)
2           Anna Hood           7477 301 311        290 5925 3361           92780 Tustin (Orange county)
3           Anna Jimenez        5454 777 714        361 3983 4212           94565 Pittsburg (Contra Costa county)
4           Anthony Wong        7280 610 853        163 4936 2049           75211 Dallas (Dallas county)
5           Barbara Gallegos    8727 928 908        760 5325 7798           79938 El Paso (El Paso county)
6           Barbara Mcfarland   8975 333 499        766 3489 1551           75211 Dallas (Dallas county)
7           Barbara Roman       5955 162 567        857 9119 8886           77479 Sugar Land (Fort Bend county)
8           Benita King         5696 412 376        400 3176 7869           75217 Dallas (Dallas county)
9           Bernice Yoder       1421 782 160        295 9986 6536           11236 Brooklyn (Kings county)
10          Blake Flores        5959 631 608        124 7590 6357           80015 Aurora (Arapahoe county)
11          Bonnie Fleming      4664 539 183        866 3899 3251           90660 Pico Rivera (Los Angeles county)
12          Bonny Holt          2152 553 932        662 4270 9066           92335 Fontana (San Bernardino county)
...
142         Tina Bauer          5491 053 233        790 2929 6545           77429 Cypress (Harris county)
143         Tina King           9343 538 322        350 8025 8524           78046 Laredo (Webb county)
144         Todd Galvan         1371 480 868        083 7767 8120           926 San Juan (San Juan county)
145         Tyrone Anderson     3325 037 308        923 1219 9054           85142 Queen Creek (Maricopa county)
146         Walter Davis        6127 425 747        806 3169 4777           29072 Lexington (Lexington county)
147         Wanda Coleman       9337 840 866        030 9772 5684           95112 San Jose (Santa Clara county)
148         William Gomez       1209 647 094        552 6509 9020           89121 Las Vegas (Clark county)
149         William Phillips    2248 720 093        675 5712 6085           94533 Fairfield (Solano county)
150         William Sullivan    4209 840 396        142 1740 7595           78660 Pflugerville (Travis county)

A procurar por nome (1st name or full name)...
Enter the name you want to search for:
```

De seguida é inserida uma *string,* neste caso ilustrativo usamos "William".

```
List of persons with 'William' (name):

Nº          Name                Telephone Number     Social Sec. Number       Zip Code
1           William Gomez       1209 647 094         552 6509 9020            89121 Las Vegas (Clark county)
2           William Phillips    2248 720 093         675 5712 6085            94533 Fairfield (Solano county)
3           William Sullivan    4209 840 396         142 1740 7595            78660 Pflugerville (Travis county)
```

É também possível pesquisar pelo nome completo ("William Gomez" no exemplo abaixo) .

```
A procurar por nome (1st name or full name)...
Enter the name you want to search for: William Gomez


List of persons with 'William Gomez' (name):

Nº          Name                Telephone Number     Social Sec. Number       Zip Code
1           William Gomez       1209 647 094         552 6509 9020            89121 Las Vegas (Clark county)
```

É possível pesquisar também pelo *Telephone Number* e pelo *Social Security Number* passando "-searchX" ao executar o programa, alterando X para 2 ou 3, respetivamente. O processo idêntico, sendo apresentado apenas a parte final (da pesquisa).

Phone Number:

```
A procurar por phone Number...
Enter the Phone Number you want to search for: 9962 562 674


List of persons with '9962 562 674' phoneNumber:

Nº          Name                Telephone Number     Social Sec. Number       Zip Code
1           Eddie Silva         9962 562 674         779 5089 6934            92677 Laguna Niguel (Orange county)
```

Social Security Number:

```
A procurar por Security Number...
Enter the Security Number you want to search for: 998 9746 8384


List of persons with '998 9746 8384' (securityNumber):

Nº          Name                Telephone Number     Social Sec. Number       Zip Code
1           Sharon Stewart      5080 471 818         998 9746 8384            92509 Riverside (Riverside county)
```

Demonstrações da função *nodeCounter*

Execução com: ./multi_ordered_trees 103541 150

Segundo esta linha de comandos, é expectável o programa apenas calcular os tempos e mostrar o número de nós por cada nível, para cada índice.

```
asd@LAPTOP-NFVK0SG0:/mnt/d/College/2_ano/1_SEMESTRE/AED/Trabalho2/A02/A02$ ./ml 103541 150
Tree creation time (150 persons): 1.626e-04s
Tree search time (150 persons, index 0): 4.930e-05s
Tree search time (150 persons, index 1): 4.400e-05s
Tree search time (150 persons, index 2): 4.170e-05s
Tree search time (150 persons, index 3): 5.730e-05s
Tree depth for index 0: 14 (done in 5.600e-06s)
Tree depth for index 1: 12 (done in 5.400e-06s)
Tree depth for index 2: 12 (done in 4.000e-06s)
Tree depth for index 3: 13 (done in 3.400e-06s)

Para o índice 0:
        Nível (0) -> 1 nodes!
        Nível (1) -> 2 nodes!
        Nível (2) -> 4 nodes!
        Nível (3) -> 6 nodes!
        Nível (4) -> 6 nodes!
        Nível (5) -> 9 nodes!
        Nível (6) -> 14 nodes!
        Nível (7) -> 20 nodes!
        Nível (8) -> 23 nodes!
        Nível (9) -> 19 nodes!
        Nível (10) -> 17 nodes!
        Nível (11) -> 18 nodes!
        Nível (12) -> 7 nodes!
        Nível (13) -> 3 nodes!
```

```
Para o índice 1:
        Nível (0) -> 1 nodes!
        Nível (1) -> 2 nodes!
        Nível (2) -> 4 nodes!
        Nível (3) -> 8 nodes!
        Nível (4) -> 13 nodes!
        Nível (5) -> 18 nodes!
        Nível (6) -> 21 nodes!
        Nível (7) -> 22 nodes!
        Nível (8) -> 24 nodes!
        Nível (9) -> 16 nodes!
        Nível (10) -> 11 nodes!
        Nível (11) -> 5 nodes!


Para o índice 2:
        Nível (0) -> 1 nodes!
        Nível (1) -> 2 nodes!
        Nível (2) -> 4 nodes!
        Nível (3) -> 8 nodes!
        Nível (4) -> 16 nodes!
        Nível (5) -> 21 nodes!
        Nível (6) -> 22 nodes!
        Nível (7) -> 26 nodes!
        Nível (8) -> 22 nodes!
        Nível (9) -> 15 nodes!
        Nível (10) -> 7 nodes!
        Nível (11) -> 5 nodes!


Para o índice 3:
        Nível (0) -> 1 nodes!
        Nível (1) -> 2 nodes!
        Nível (2) -> 4 nodes!
        Nível (3) -> 8 nodes!
        Nível (4) -> 13 nodes!
        Nível (5) -> 19 nodes!
        Nível (6) -> 23 nodes!
        Nível (7) -> 24 nodes!
        Nível (8) -> 23 nodes!
        Nível (9) -> 11 nodes!
        Nível (10) -> 9 nodes!
        Nível (11) -> 6 nodes!
        Nível (12) -> 5 nodes!
```

Conclusões Finais

Este trabalho permitiu evidenciar a importância da escolha do tipo de estrutura de dados no armazenamento de informação, os seus efeitos na eficiência das funções implementadas, bem como o efeito de diferentes tipos de dados na eficiência do seu armazenamento e posterior busca.
Para além disto, ficou evidente a utilidade de árvores binárias como um método de organização de informação.

Webgrafia

Figuras 2 e 3:
https://iq.opengenus.org/nodes-at-distance-k-from-given-node/

Código C (apenas as funções alteradas)

```c
void tree_insert(tree_node_t *node, int ind, tree_node_t **roots)
{
    //if roots are null (tree is empty), then the node is the root
    if (roots[ind] == NULL)
    {
        roots[ind] = node;
        return;
    }
    if (compare_tree_nodes(node, roots[ind], ind) < 0) //recurs down the tree if tree isn't empty
        tree_insert(node, ind, roots[ind]->left);
    else if (compare_tree_nodes(node, roots[ind], ind) > 0)
        tree_insert(node, ind, roots[ind]->right);
    return;
}

//
// tree search routine (place your code here)
//

tree_node_t *find(tree_node_t node, int ind, tree_node_t **roots)
{
    //if roots are null (tree is empty) || node to find is root (ig)
    if (roots[ind] == NULL || (compare_tree_nodes(&node, roots[ind], ind) == 0))
    {
        return roots[ind];
    }

    //left
    else if (compare_tree_nodes(&node, roots[ind], ind) < 0) //recurs down the tree if tree isn't empty
        return find(node, ind, roots[ind]->left);

    //right -> (compare_tree_nodes(&node, roots[ind], ind) > 0)
    else
        return find(node, ind, roots[ind]->right);
}
```

```c
int tree_depth(tree_node_t **roots, int idx)
{
    //tree non existent
    if (roots[idx] == NULL)
        return -1;
    else
    {
        //calcular rec para a esq e para a direita da root
        int left = tree_depth(roots[idx]->left, idx);
        int right = tree_depth(roots[idx]->right, idx);

        //ver qual é maior, e somar 1, para calcular a depth total
        if (left > right)
            return left + 1;
        else
            return right + 1;
    }
}

//
// list, i,e, traverse the tree (place your code here)
//

void list(tree_node_t **root, int ind, int *np)
{
    if (root[ind] != NULL)
    {
        list(root[ind]->left, ind, np);
        printf("%-13d%-20s %-20s %-25s %-10s\n", *np, root[ind]->name, root[ind]->telephone_number,
               root[ind]->social_number, root[ind]->zip_code);
        *np = *np + 1;
        list(root[ind]->right, ind, np);
    }
}
```

```c
void searchF(tree_node_t **root, int ind, int *np, char *search, int flag)
{
    if (root[ind] != NULL)
    {
        searchF(root[ind]->left, ind, np, search, flag);

        //NAME
        if (flag == 0)
        {
            char tempName[MAX_NAME_SIZE];
            strcpy(tempName, root[ind]->name);
            //guarda o nome completo na var tempName
            char *token = strtok(tempName, " ");
            int i = 0;
            while (token[i])
            {
                if (token[i] == '\n')
                {
                    token[i] = '\0';
                    break;
                }
                i++;
            }

            if ((strcmp(search, token)) == 0 || (strcmp(search, root[ind]->name)) == 0)
            {
                printf("%-13d%-20s %-20s %-25s %-10s\n", *np, root[ind]->name, root[ind]->telephone_number,
                       root[ind]->social_number, root[ind]->zip_code);
                *np = *np + 1;
            }
        }

        //ZIP CODE
        else if (flag == 1)
```

```
        //Phone Number
        else if (flag == 2)
        {
            if ((strcmp(search, root[ind]->telephone_number)) == 0)
            {
                printf("%-13d%-20s %-20s %-25s %-10s\n", *np, root[ind]->name, root[ind]->telephone_number,
                        root[ind]->social_number, root[ind]->zip_code);
                *np = *np + 1;
            }
        }

        //Security Number
        else if (flag == 3)
        {
            if ((strcmp(search, root[ind]->social_number)) == 0)
            {
                printf("%-13d%-20s %-20s %-25s %-10s\n", *np, root[ind]->name, root[ind]->telephone_number,
                        root[ind]->social_number, root[ind]->zip_code);
                *np = *np + 1;
            }
        }

        searchF(root[ind]->right, ind, np, search, flag);
    }
}
```

```
int nodeCounter(tree_node_t **root, int cind, int lvl, int idx) {

    if (root[idx] == NULL)
    {
        return 0;
    }
    if (cind == lvl)
        return 1;

    return nodeCounter(root[idx] -> left, cind + 1, lvl, idx) +
    nodeCounter(root[idx] -> right, cind + 1, lvl, idx);

}
```

```c
int main(int argc, char **argv)
{
    double dt;

    // process the command line arguments
    if (argc < 3)
    {
        fprintf(stderr, "Usage: %s student_number number_of_persons [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");
        fprintf(stderr, "  -search[N]           # search the tree contents, sorted by key index N (the default is index 0)\n");
        fprintf(stderr, "\t  N = 0 -> Name\n\t  N = 1 -> Zip Code\n\t  N = 2 -> Phone Number\n\t  N = 3 -> Security Number\n");
        // place a description of your own options here
        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_persons = atoi(argv[2]);
    if (n_persons < 3 || n_persons > 10000000)
    {
        fprintf(stderr, "Bad number of persons (%d) --- must be an integer belonging to [3,10000000]\n", n_persons);
        return 1;
    }

    // generate all data
    tree_node_t *persons = (tree_node_t *)calloc((size_t)n_persons, sizeof(tree_node_t));
    if (persons == NULL)
    {
        fprintf(stderr, "Output memory!\n");
        return 1;
    }
    aed_srandom(student_number);
    for (int i = 0; i < n_persons; i++)
    {
        random_name(&(persons[i].name[0]));
        random_zip_code(&(persons[i].zip_code[0]));
        random_telephone_number(&(persons[i].telephone_number[0]));
        random_social_number(&(persons[i].social_number[0]));
        for (int j = 0; j < 3; j++)
            persons[i].left[j] = persons[i].right[j] = NULL; // make sure the pointers are initially NULL
    }
```

```c
// create the ordered binary trees
dt = cpu_time();
tree_node_t *roots[4]; // four indices, four roots
for (int main_index = 0; main_index < 4; main_index++)
    roots[main_index] = NULL;
for (int i = 0; i < n_persons; i++)
    for (int main_index = 0; main_index < 4; main_index++)
        tree_insert(&(persons[i]), main_index, roots); // place your code here to insert &(persons[i]) in the tree with number main_index
dt = cpu_time() - dt;
printf("Tree creation time (%d persons): %.3es\n", n_persons, dt);

//

// search the tree
for (int main_index = 0; main_index < 4; main_index++)
{
    dt = cpu_time();
    for (int i = 0; i < n_persons; i++)
    {
        tree_node_t n = persons[i];                    // make a copy of the node data
        if (find(n, main_index, roots) != &(persons[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 persons, index %d): %.3es\n", n_persons, main_index, dt);
}

//

// compute the largest tree depdth
for (int main_index = 0; main_index < 4; main_index++)
{
    dt = cpu_time();

    int depth = tree_depth(roots, 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);
}

//
```

```c
//node counter
int currLvl = 0;
for (int main_index = 0; main_index < 4; main_index++)
{
    int nNodes = 0;
    int lvl = 0;

    int depth = tree_depth(roots, main_index);
    printf("\nPara o índice %d:\n",main_index);

    for (int i = 0; i < depth; i++) {
        nNodes = nodeCounter(roots, currLvl, lvl, main_index);
        //printf("lvl = %d -> %d\n", lvl, nNodes);

        printf("\tNível (%d) -> %d nodes!\n", lvl, nNodes);

        lvl ++;
    }
    printf("\n");
}
```

```c
// 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;
        int np = 1;
        printf("\n\nList of persons:\n\n");
        printf("%-13s %-20s %-20s %-25s %-10s\n", "Nº", "Name", "Telephone Number", "Social Sec. Number", "Zip Code");
        list(roots, main_index, &np); // place your code here to traverse, in order, the tree with number main_index
        printf("\n");
    }

    //search OPT
    if (strncmp(argv[i], "-search", 7) == 0)
    { // list all (optional)
        int main_index = atoi(&(argv[i][7]));
        if (main_index < 0)
            main_index = 0;
        if (main_index >= 3)
            main_index = 3;

        if (main_index == 0)
        {
            printf("A procurar por nome (1st name or full name)...\n");

            char nameSearch[MAX_NAME_SIZE];
            printf("Enter the name you want to search for: ");
            // read a string
            fgets(nameSearch, sizeof(nameSearch), stdin);
            int i = 0;
            while (nameSearch[i])
            {
                if (nameSearch[i] == '\n')
                {
                    nameSearch[i] = '\0';
                    break;
                }
                i++;
            }
            printf("\n\nList of persons with '%s' (name):\n\n", nameSearch);
            printf("%-13s %-20s %-20s %-25s %-10s\n", "Nº", "Name", "Telephone Number", "Social Sec. Number", "Zip Code");
            int np = 1;
            searchF(roots, main_index, &np, nameSearch, main_index);
        }
```

```c
if (main_index == 1)
{
    printf("A procurar por zip code...\n");

    char zipCodeSearch[MAX_ZIP_CODE_SIZE];
    printf("Enter the zipCode you want to search for: ");
    // read a string
    fgets(zipCodeSearch, sizeof(zipCodeSearch), stdin);
    int i = 0;
    while (zipCodeSearch[i])
    {
        if (zipCodeSearch[i] == '\n')
        {
            zipCodeSearch[i] = '\0';
            break;
        }
        i++;
    }
    printf("\n\nList of persons with '%s' (zipCode):\n\n", zipCodeSearch);
    printf("%-13s %-20s %-20s %-25s %-10s\n", "Nº", "Name", "Telephone Number", "Social Sec. Number", "Zip Code");
    int np = 1;
    searchF(roots, main_index, &np, zipCodeSearch, main_index);
}

if (main_index == 2)
{
    printf("A procurar por phone Number...\n");

    char phoneNumber[MAX_ZIP_CODE_SIZE];
    printf("Enter the Phone Number you want to search for: ");
    // read a string
    fgets(phoneNumber, sizeof(phoneNumber), stdin);
    int i = 0;
    while (phoneNumber[i])
    {
        if (phoneNumber[i] == '\n')
        {
            phoneNumber[i] = '\0';
            break;
        }
        i++;
    }
    printf("\n\nList of persons with '%s' phoneNumber:\n\n", phoneNumber);
    printf("%-13s %-20s %-20s %-25s %-10s\n", "Nº", "Name", "Telephone Number", "Social Sec. Number", "Zip Code");
    int np = 1;
    searchF(roots, main_index, &np, phoneNumber, main_index);
}
```

```c
            if (main_index == 3)
            {
                printf("A procurar por Security Number...\n");

                char securityNumber[MAX_ZIP_CODE_SIZE];
                printf("Enter the Security Number you want to search for: ");
                // read a string
                fgets(securityNumber, sizeof(securityNumber), stdin);
                int i = 0;
                while (securityNumber[i])
                {
                    if (securityNumber[i] == '\n')
                    {
                        securityNumber[i] = '\0';
                        break;
                    }
                    i++;
                }
                printf("\n\nList of persons with '%s' (securityNumber):\n\n", securityNumber);
                printf("%-13s %-20s %-20s %-25s %-10s\n", "Nº", "Name", "Telephone Number", "Social Sec. Number", "Zip Code");
                int np = 1;
                searchF(roots, main_index, &np, securityNumber, main_index);
            }
        }
        // place your own options here
    }

    //

    // clean up --- don't forget to test your program with valgrind, we don't want any memory leaks
    free(persons);

    return 0;
}
```

Alterações nos outros ficheiros

```c
void random_social_number(char social_number[MAX_SOCIAL_SECURITY_NUMBER + 1])
{
  int n1 = aed_random() % 1000;         //  000..999
  int n2 = 1000 + aed_random() % 9000;       //  1000..9999
  int n3 = 1000 + aed_random() % 9000;         //  1000..9999
  if (snprintf(social_number, MAX_SOCIAL_SECURITY_NUMBER + 1, "%03d %04d %04d", n1, n2, n3) >= MAX_SOCIAL_SECURITY_NUMBER + 1)
  {
    fprintf(stderr, "Social number too large (%03d) (%04d) (%04d)\n", n1, n2, n3);
    exit(1);
  }
}
```

C AED_2021_A02.h > MAX_SOCIAL_SECURITY_NUMBER

```c
//
// Global stuff for the 2021.A.02 project
//

#ifndef AED_2021_A02

// protect against multiple inclusions of this file
#define AED_2021_A02

#define MAX_NAME_SIZE              31
#define MAX_ZIP_CODE_SIZE          63
#define MAX_TELEPHONE_NUMBER_SIZE  15
#define MAX_SOCIAL_SECURITY_NUMBER 14

// from random_number.c
void aed_srandom(int seed);
int aed_random(void);

// from random_data.c
void random_name(char name[MAX_NAME_SIZE + 1]);
void random_zip_code(char zip_code[MAX_ZIP_CODE_SIZE + 1]);
void random_telephone_number(char telephone_number[MAX_TELEPHONE_NUMBER_SIZE + 1]);
void random_social_number(char social_number[MAX_SOCIAL_SECURITY_NUMBER + 1]);

// from elapsed_time.c
double cpu_time(void);

#endif
```

Código MATLAB

```
%% tree DEPTH -> 4 INDICES
%pessoas tdepth
clear;
clc;
close all;
%vetor de nº de pessoas -> escala log
n0 = logspace(1, 7, 19);
%valores0 = load("tdepth102442.txt");
valores0 = load("tdepth103541.txt");
%valores0 = load("tdepth103600.txt");
tdepth0 = valores0(1:4:end,2);
tdepth1 = valores0(2:4:end,2);
tdepth2 = valores0(3:4:end,2);
tdepth3 = valores0(4:4:end,2);

figure(1)
xlabel("Nº de Pessoas geradas");
ylabel("Tree Depth")
title('Maximum Tree Depth')
hold on;
semilogx(n0, tdepth0, '*', Color="Green")
hold on;
semilogx(n0, tdepth1,'*', Color="Blue");
hold on;
semilogx(n0, tdepth2, '*', Color="Red")
hold on;
semilogx(n0, tdepth3,'*', Color="Black");
legend("Indice 0");
legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");
ylim([0 70]);
set(gca, 'XScale', 'log');
grid on;
```

```
%% tree search -> 4 INDICES
%pessoas tdepth
clear;
clc;

valores0 = load("tsearch.txt");
% valores0 = load("tsearch102442.txt");
% valores0 = load("tsearch103600.txt");
n0 = logspace(1, 7, 19);
tsearch0 = valores0(1:4:end,2);
tsearch1 = valores0(2:4:end,2);
tsearch2 = valores0(3:4:end,2);
tsearch3 = valores0(4:4:end,2);

figure(2)
xlabel("Nº de Pessoas geradas");
ylabel("time (s)")
title('Tree Search Time')
hold on;
loglog(n0 ,tsearch0, '*', Color="Green")
hold on;
loglog(n0,tsearch1,'*', Color="Blue");
hold on;
loglog(n0,tsearch2, '*', Color="Red")
hold on;
loglog(n0,tsearch3,'*', Color="Black");
hold on;
legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");
set(gca, 'XScale', 'log', 'YScale', 'log');
grid on;
hold off;
```

```
%% tree creation -> 4 INDICES
%pessoas tdepth
clear;
clc;

valores0 = load("tcreation102442.txt");
% valores0 = load("tsearch102442.txt");
% valores0 = load("tsearch103600.txt");
n0 = logspace(1, 7, 19);
tsearch0 = valores0(1:end,2);

figure(2)
loglog(n0 ,tsearch0, '*', Color="Green")
legend("All Indices");
xlabel("Nº de Pessoas geradas");
ylabel("time (s)")
title('Tree Creation Time')
set(gca, 'XScale', 'log', 'YScale', 'log');
grid on;
hold off;
```

```
%% T DEPTH PARA VARIOS N MECS
clear;
clc;
close all;
%vetor de nº de pessoas -> escala log
n0 = logspace(1, 7, 19);
valores0 = load("tdepth10000.txt");
valores1 = load("tdepth12500.txt");
valores2 = load("tdepth15000.txt");
valores3 = load("tdepth17500.txt");
valores4 = load("tdepth20000.txt");

%valores 0
tdepth0_0 = valores0(1:4:end,2);
tdepth0_1 = valores0(2:4:end,2);
tdepth0_2 = valores0(3:4:end,2);
tdepth0_3 = valores0(4:4:end,2);

%valores 1
tdepth1_0 = valores1(1:4:end,2);
tdepth1_1 = valores1(2:4:end,2);
tdepth1_2 = valores1(3:4:end,2);
tdepth1_3 = valores1(4:4:end,2);

%valores 2
tdepth2_0 = valores2(1:4:end,2);
tdepth2_1 = valores2(2:4:end,2);
tdepth2_2 = valores2(3:4:end,2);
tdepth2_3 = valores2(4:4:end,2);

%valores 3
tdepth3_0 = valores3(1:4:end,2);
tdepth3_1 = valores3(2:4:end,2);
tdepth3_2 = valores3(3:4:end,2);
tdepth3_3 = valores3(4:4:end,2);
```

```
%valores 4
tdepth4_0 = valores4(1:4:end,2);
tdepth4_1 = valores4(2:4:end,2);
tdepth4_2 = valores4(3:4:end,2);
tdepth4_3 = valores4(4:4:end,2);

figure(1)
%valores 0
hold on;
semilogx(n0, tdepth0_0, '*', Color="Green")
hold on;
semilogx(n0, tdepth0_1,'+', Color="Blue");
hold on;
semilogx(n0, tdepth0_2, 'o', Color="Red")
hold on;
semilogx(n0, tdepth0_3,'.', Color="Black");
hold on

%valores 1
hold on;
semilogx(n0, tdepth1_0, '*', Color="Green")
hold on;
semilogx(n0, tdepth1_1,'+', Color="Blue");
hold on;
semilogx(n0, tdepth1_2, 'o', Color="Red")
hold on;
semilogx(n0, tdepth1_3,'.', Color="Black");

%valores 2
hold on;
semilogx(n0, tdepth2_0, '*', Color="Green")
hold on;
semilogx(n0, tdepth2_1,'+', Color="Blue");
hold on;
semilogx(n0, tdepth2_2, 'o', Color="Red")
hold on;
semilogx(n0, tdepth2_3,'.', Color="Black");
```

```
%valores 3
hold on;
semilogx(n0, tdepth3_0, '*', Color="Green")
hold on;
semilogx(n0, tdepth3_1,'+', Color="Blue");
hold on;
semilogx(n0, tdepth3_2, 'o', Color="Red")
hold on;
semilogx(n0, tdepth3_3,'.', Color="Black");

%valores 4
hold on;
semilogx(n0, tdepth4_0, '*', Color="Green")
hold on;
semilogx(n0, tdepth4_1,'+', Color="Blue");
hold on;
semilogx(n0, tdepth4_2, 'o', Color="Red")
hold on;
semilogx(n0, tdepth4_3,'.', Color="Black");


legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");

xlabel("Nº de Pessoas geradas");
ylabel("Tree Depth")
title('Maximum Tree Depth')
ylim([0 70]);
set(gca, 'XScale', 'log');
grid on;
hold off;
```

```
%% T CREATION PARA VARIOS N MECS
clear;
clc;
close all;
%vetor de nº de pessoas -> escala log
n0 = logspace(1, 7, 19);
valores0 = load("tcreation10000.txt");
valores1 = load("tcreation12500.txt");
valores2 = load("tcreation15000.txt");
valores3 = load("tcreation17500.txt");
valores4 = load("tcreation20000.txt");

t0 = valores0(1:end,2);
t1 = valores1(1:end,2);
t2 = valores2(1:end,2);
t3 = valores3(1:end,2);
t4 = valores4(1:end,2);

figure(2)
loglog(n0 ,t0, 'o', Color="Red"); hold on;
loglog(n0 ,t1, 'o', Color="Red"); hold on;
loglog(n0 ,t2, 'o', Color="Red"); hold on;
loglog(n0 ,t3, 'o', Color="Red"); hold on;
loglog(n0 ,t4, 'o', Color="Red"); hold on;

legend("All Indices");
xlabel("Nº de Pessoas geradas");
ylabel("time (s)")
title('Tree Creation Time')
set(gca, 'XScale', 'log', 'YScale', 'log');
grid on;
hold off;
```

```
%% T SEARCH PARA VARIOS N MECS
clear;
clc;
close all;
%vetor de nº de pessoas -> escala log
n0 = logspace(1, 7, 19);
valores0 = load("tsearc10000.txt");
valores1 = load("tsearc12500.txt");
valores2 = load("tsearc15000.txt");
valores3 = load("tsearc17500.txt");
valores4 = load("tsearc20000.txt");

%valores 0
t0_0 = valores0(1:4:end,2);
t0_1 = valores0(2:4:end,2);
t0_2 = valores0(3:4:end,2);
t0_3 = valores0(4:4:end,2);

%valores 1
t1_0 = valores1(1:4:end,2);
t1_1 = valores1(2:4:end,2);
t1_2 = valores1(3:4:end,2);
t1_3 = valores1(4:4:end,2);

%valores 2
t2_0 = valores2(1:4:end,2);
t2_1 = valores2(2:4:end,2);
t2_2 = valores2(3:4:end,2);
t2_3 = valores2(4:4:end,2);

%valores 3
t3_0 = valores3(1:4:end,2);
t3_1 = valores3(2:4:end,2);
t3_2 = valores3(3:4:end,2);
t3_3 = valores3(4:4:end,2);
```

```
%valores 4
t4_0 = valores4(1:4:end,2);
t4_1 = valores4(2:4:end,2);
t4_2 = valores4(3:4:end,2);
t4_3 = valores4(4:4:end,2);

figure(1)
%valores 0
hold on;
loglog(n0, t0_0, '*', Color="Green")
hold on;
loglog(n0, t0_1,'+', Color="Blue");
hold on;
loglog(n0, t0_2, 'o', Color="Red")
hold on;
loglog(n0, t0_3,'.', Color="Black");
hold on

%valores 1
hold on;
loglog(n0, t1_0, '*', Color="Green")
hold on;
loglog(n0, t1_1,'+', Color="Blue");
hold on;
loglog(n0, t1_2, 'o', Color="Red")
hold on;
loglog(n0, t1_3,'.', Color="Black");

%valores 2
hold on;
loglog(n0, t2_0, '*', Color="Green")
hold on;
loglog(n0, t2_1,'+', Color="Blue");
hold on;
loglog(n0, t2_2, 'o', Color="Red")
hold on;
loglog(n0, t2_3,'.', Color="Black");
```

```
%valores 3
hold on;
loglog(n0, t3_0, '*', Color="Green")
hold on;
loglog(n0, t3_1,'+', Color="Blue");
hold on;
loglog(n0, t3_2, 'o', Color="Red")
hold on;
loglog(n0, t3_3,'.', Color="Black");

%valores 4
hold on;
loglog(n0, t4_0, '*', Color="Green")
hold on;
loglog(n0, t4_1,'+', Color="Blue");
hold on;
loglog(n0, t4_2, 'o', Color="Red")
hold on;
loglog(n0, t4_3,'.', Color="Black");


legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");

xlabel("Nº de Pessoas geradas");
ylabel("Time (s)")
title('Tree Search Time')
ylim([0 70]);
set(gca, 'XScale', 'log', 'YScale', 'log');
grid on;
hold off;
```

```
%% Histogramas - DEPTH
clear;clc;close all;

valores = load("depthNMEC.txt");

N = length(valores);

x = unique(valores(:,2));
nOcc0 = zeros(length(x), 1);
nOcc1 = zeros(length(x), 1);
nOcc2 = zeros(length(x), 1);
nOcc3 = zeros(length(x), 1);

for i = 1 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i, 2)
            nOcc0(n) = nOcc0(n) + 1;
        end
    end
end

for i = 2 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i, 2)
            nOcc1(n) = nOcc1(n) + 1;
        end
    end
end

for i = 3 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i ,2)
            nOcc2(n) = nOcc2(n) + 1;
        end
    end
end

for i = 4 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i,2)
            nOcc3(n) = nOcc3(n) + 1;
        end
    end
```

```
%% Histogramas - search
clear;clc;close all;

valores = load("tsearchNMEC.txt");

N = length(valores);

x = unique(valores(:,2));
nOcc0 = zeros(length(x), 1);
nOcc1 = zeros(length(x), 1);
nOcc2 = zeros(length(x), 1);
nOcc3 = zeros(length(x), 1);

for i = 1 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i, 2)
            nOcc0(n) = nOcc0(n) + 1;
        end
    end
end

for i = 2 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i, 2)
            nOcc1(n) = nOcc1(n) + 1;
        end
    end
end

for i = 3 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i ,2)
            nOcc2(n) = nOcc2(n) + 1;
        end
    end
end

for i = 4 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i,2)
            nOcc3(n) = nOcc3(n) + 1;
        end
    end
```

```
for i = 4 : 4 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i,2)
            nOcc3(n) = nOcc3(n) + 1;
        end
    end
end

bar(x, [nOcc0 nOcc1 nOcc2 nOcc3], 'grouped')

legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");
ylabel("Number of occurrences")
xlabel("Depth (100000 persons)")
title("Maximum depth histogram for 100000 persons and 10000 experiments")
%set(gca, 'YScale', 'log');
grid on;
hold off;
```

```
bar(0.005:0.005:2.34, [nOcc0 nOcc1 nOcc2 nOcc3], 'grouped')

legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");
ylabel("Number of occurrences")
xlabel("Time (s) (100000 persons)")
title("Tree search time histogram for 100000 persons and 10000 experiments")

grid on;
%set(gca, 'XScale', 'log');
hold off;
```

```
%% Histogramas - CREATION
clear;clc;close all;

valores = load("tcreationNMEC.txt");

N = length(valores);

x = unique(valores(:,2));
nOcc = zeros(length(x), 1);

for i = 1 : length(valores)
    for n = 1 : length(x)
        if x(n) == valores(i, 2)
            nOcc(n) = nOcc(n) + 1;
        end
    end
end

bar(x, nOcc, 'grouped')
xlim([0.2 1.5])
legend("All Indices");
ylabel("Number of occurrences")
xlabel("Time (s) (100000 persons)")
title("Tree Creation time histogram for 100000 persons and 10000 experiments")
%set(gca, 'YScale', 'log');
grid on;
hold off;
```

```
%% NODES -> nivel e nº de nós
clear;
clc;
close all;

valores0 = load("tnodes10000.txt");
valores1 = load("tnodes12500.txt");
valores2 = load("tnodes15000.txt");
valores3 = load("tnodes17500.txt");
valores4 = load("tnodes20000.txt");

clc;
close all;
N = length(valores0);
N1 = length(valores1);
N2 = length(valores2);
N3 = length(valores3);
N4 = length(valores4);
%nºs de níveis
x = unique(valores0(:,2));

nOcc0 = zeros(length(x), 1);
nOcc1 = zeros(length(x), 1);
nOcc2 = zeros(length(x), 1);
nOcc3 = zeros(length(x), 1);
```

```
nOcc0 = zeros(length(x), 1);
nOcc1 = zeros(length(x), 1);
nOcc2 = zeros(length(x), 1);
nOcc3 = zeros(length(x), 1);

nOcc0_1 = zeros(length(x), 1);
nOcc1_1 = zeros(length(x), 1);
nOcc2_1 = zeros(length(x), 1);
nOcc3_1 = zeros(length(x), 1);

nOcc0_2 = zeros(length(x), 1);
nOcc1_2 = zeros(length(x), 1);
nOcc2_2 = zeros(length(x), 1);
nOcc3_2 = zeros(length(x), 1);

nOcc0_3 = zeros(length(x), 1);
nOcc1_3 = zeros(length(x), 1);
nOcc2_3 = zeros(length(x), 1);
nOcc3_3 = zeros(length(x), 1);

nOcc0_4 = zeros(length(x), 1);
nOcc1_4 = zeros(length(x), 1);
nOcc2_4 = zeros(length(x), 1);
nOcc3_4 = zeros(length(x), 1);
```

```
for i = 1 : N...

for i = 1 : N1...


for i = 1 : N2...


for i = 1 : N3...

for i = 1 : N4
    idx = valores4(i, 1);
    if (idx == 0)
        nOcc0_4(valores4(i, 2) + 1) = valores4(i, 3);
    end

    if (idx == 1)
        nOcc1_4(valores4(i, 2) + 1) = valores4(i, 3);
    end

    if (idx == 2)
        nOcc2_4(valores4(i, 2) + 1) = valores4(i, 3);
    end

    if (idx == 3)
        nOcc3_4(valores4(i, 2) + 1) = valores4(i, 3);
    end
end
```

```
%valores 0
plot(1:length(nOcc0), nOcc0, '.b'); hold on;
plot(1:length(nOcc1), nOcc1, '.r'); hold on;
plot(1:length(nOcc2), nOcc2, '.y'); hold on;
plot(1:length(nOcc3), nOcc3, '.g'); hold on;

%valores 1
plot(1:length(nOcc0_1), nOcc0_1, '.b'); hold on;
plot(1:length(nOcc1_1), nOcc1_1, '.r'); hold on;
plot(1:length(nOcc2_1), nOcc2_1, '.y'); hold on;
plot(1:length(nOcc3_1), nOcc3_1, '.g'); hold on;

%valores 2
plot(1:length(nOcc0_2), nOcc0_2, '.b'); hold on;
plot(1:length(nOcc1_2), nOcc1_2, '.r'); hold on;
plot(1:length(nOcc2_2), nOcc2_2, '.y'); hold on;
plot(1:length(nOcc3_2), nOcc3_2, '.g'); hold on;

%valores 3
plot(1:length(nOcc0_3), nOcc0_3, '.b'); hold on;
plot(1:length(nOcc1_3), nOcc1_3, '.r'); hold on;
plot(1:length(nOcc2_3), nOcc2_3, '.y'); hold on;
plot(1:length(nOcc3_3), nOcc3_3, '.g'); hold on;

%valores 4
plot(1:length(nOcc0_4), nOcc0_4, '.b'); hold on;
plot(1:length(nOcc1_4), nOcc1_4, '.r'); hold on;
plot(1:length(nOcc2_4), nOcc2_4, '.y'); hold on;
plot(1:length(nOcc3_4), nOcc3_4, '.g'); hold on;

legend("Indice 0", "Indice 1", "Indice 2", "Indice 3");

xlabel("Nível da árvore");
ylabel("Nº de Nodes")
title('Tree Nodes by level for 100000 persons')
xlim([0 55]);
grid on;
hold off;
```