universidade de aveiro
deti departamento de electrónica, telecomunicações e informática

Trabalho Prático 2
Árvores binárias ordenadas aleatoriamente

Licenciatura em Engenharia Informática
UC 40437 - Algoritmos e Estruturas de Dados
Ano letivo 2018/2019

Trabalho realizado por:
88808 - João Miguel Nunes de Medeiros e Vasconcelos
88886 - Tiago Carvalho Mendes

Aveiro, 31 de dezembro de 2018

Índice

Introdução

O presente documento destina-se a descrever detalhadamente a abordagem utilizada para resolver o problema proposto como segundo trabalho prático da unidade curricular de Algoritmos e Estruturas de Dados da Universidade de Aveiro, cujo tema era o estudo de Árvores binárias ordenadas aleatoriamente.

O objetivo deste trabalho é então estudar empiricamente árvores binárias aleatórias ordenadas. Especificamente, pretendemos para uma árvore com *n* nodes, calcular e perceber como evolui quando o *n* aumenta:

- A altura média de uma árvore;
- O número médio de folhas;
- O custo médio de procurar por um item de informação presente na árvore (hit);
- O custo médio de procurar por um item de informação que não esteja presente na árvore (miss);

Para suportar as nossas conclusões apresentamos gráficos e tabelas com os resultados.

Implementação - Código para o estudo das árvores binárias

```
//
// Tomás Oliveira e Silva, AED,  November 2018
//
// 88808 João Miguel Nunes de Medeiros e Vasconcelos
// 88886 Tiago Carvalho Mendes
// empirical study of random ordered binary trees
//

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "elapsed_time.h"


//
// each node of our ordered binary tree will store a long integer
//
// the root of the tree should be declared as follows (set initially to an empty tree):
//
//   tree_node *root = NULL;
//

typedef struct tree_node
{
 struct tree_node *left;   // pointer to the left branch (a sub-tree)
 struct tree_node *right;  // pointer to the right branch (a sub-tree)
 struct tree_node *parent; // pointer to the parent node (NULL for the root of the tree)
 long data;                // the data item (we use a long here)
}
tree_node;
```

```
//
// insert a node in the tree (it is assumed that the tree does not store repeated data)
//
// use it as follows (example):
//
//   insert_node(&root,&new_node);
//

static void insert_node(tree_node **link,tree_node *n)
{
 tree_node *parent;

 parent = NULL;
 while(*link != NULL)
 {
  if(n->data == (*link)->data)
  {
   fprintf(stderr,"insert_node: %ld is already in the tree\n",n->data);
   exit(1);
  }
  parent = *link;
  link = (n->data < (*link)->data) ? &((*link)->left) : &((*link)->right); // select branch
 }
 *link = n;
 n->parent = parent;
 n->left = n->right = NULL;
}

//
// count the number of leaves of the tree
//
// use if as follows (example):
//
//   int n_leaves = count_leaves(root);
//

static int count_leaves(tree_node *link)
{
 int leafcount;
 if(link==NULL){
  return 0;
 }
 else if(link->left==NULL && link->right==NULL){
  return 1;
 }
 else{
  leafcount=count_leaves(link->right)+count_leaves(link->left);
 }
 return leafcount;
}
```

```
//
// compute the height of the tree
//
// use if as follows (example):
//
//   int height = tree_height(root);
//

static int tree_height(tree_node *link)
{ int height,height_left,height_right;
 if (link==NULL){
   return 0;
 }
 else{
   height_left=tree_height(link->left);
   height_right=tree_height(link->right);
   if (height_left>=height_right)
     height=height_left+1;
   else
     height=height_right+1;
 }
  return height;
}


//
// recursive function used to search for the location of a data item
//
// use if as follows (example):
//
//   tree_node *node = search_tree(root,data);
//

static int search_counter;

tree_node *search_tree(tree_node *link,long data)
{
 search_counter++;
 if(link == NULL)
   return NULL;
 if(link->data == data)
   return link;
 return search_tree((data < link->data) ? link->left : link->right,data);
}
```

```
//
// assuming that each data item is searched for with equal probability, compute the average number
// of recursive function calls to the search_tree() function when
// 1) the search is successful (a hit)
// 2) the search is not successful (a miss)
//
// use them as follows (example):
//
//   double average_calls_on_hit = (double)count_function_calls_on_hit(root,0) / (double)number_of_nodes;
//   double average_calls_on_miss = (double)count_function_calls_on_miss(root,0) / (double)number_of_nodes;
//

static int count_function_calls_on_hit(tree_node *link,int level)
{ int count;
 if (link==NULL){

   return 0;
 }
 count = level+1;
 if(link->left!=NULL)
   count+=count_function_calls_on_hit(link->left,level+1);
 if(link->right!=NULL)
   count+=count_function_calls_on_hit(link->right,level+1);

 return count;

}

static int count_function_calls_on_miss(tree_node *link,int level)
{ int count=0;// tenho que inicializar
 if (link==NULL){
   count=level+1;
   return count;
 }
  count+=count_function_calls_on_miss(link->right,level+1);
 count+=count_function_calls_on_miss(link->left,level+1);

  return count;
}
```

```
//
// random permutation of the n numbers 1, 3, 5, ..., 2*n-1
//
// use if as follows (example):
//
//   int n = 100;
//   int a[n];
//   rand_perm(n,&a[0]);
//

static void rand_perm(int n,int *a)
{
 int i,j,k;

 for(i = 0;i < n;i++)
   a[i] = 2 * i + 1;
 for(i = n - 1;i > 0;i--)
 {
   j = (int)floor((double)(i + 1) * (double)rand() / (1.0 + (double)RAND_MAX)); // range 0..i
   k = a[i];
   a[i] = a[j];
   a[j] = k;
 }
}
  return count;
}
```

```
//
// main program
//

int main(int argc,char **argv)
{
 int details = (argc == 3 && argv[1][0] == '-' && argv[1][1] == 'a' && atoi(argv[2]) > 0) ? 1 : 0;
 int n_experiments = 1000; // TO DO: use more (1000000 should take 2 to 3 hours)
 FILE *file;                                  // location of minima and maxima
 file = fopen("Data.csv","w");
  fprintf(file,"%s,%s,%s,%s,%s\n","n","maximum tree height (mean)","number of leaves (mean)","calls om hit (mean)","calls on miss
(mean)");

 srandom(1u); // ensure reproducible results
 printf("                              data for %d random trees\n",n_experiments);
 printf("         maximum tree height          number of leaves         calls on hit    calls on miss\n");
 printf("       ----------------------  ------------------------------  --------------  --------------\n");
 printf("    n  min max    mean    std    min   max      mean     std     mean    std     mean    std\n");
 printf("------  --- --- -------- ------  ----- ----- ---------- --------  ------- ------  ------- ------\n");
 for(int n_log = 1 * 10;n_log <= 4 * 10;n_log++)
 {
  int n = (int)round(pow(10.0,(double)n_log / 10.0)); // the number of nodes of the tree
  int a[n];                                  // the nodes' data
  tree_node *root,nodes[n];                        // the root and the storage space for the nodes of the tree
  int h_height[n + 1];                       // for an histogram of the heights of the random trees
  int h_leaves[n + 1];                       // for an histogram of the number of leaves of the random trees
  double mean,std;                            // for mean and standard deviation computations
  double x,hit[2],miss[2];                     // for the average number of hits and misses
  int m,M;

  printf("%6d",n);

  fprintf(file,"%6d,",n);
  //
  // the example in the slides
  //
  if(n == 10)
  {
   root = NULL;
   nodes[0].data = 3l; insert_node(&root,&nodes[0]);
   nodes[1].data = 1l; insert_node(&root,&nodes[1]);
   nodes[2].data = 9l; insert_node(&root,&nodes[2]);
   nodes[3].data = 7l; insert_node(&root,&nodes[3]);
   nodes[4].data = 5l; insert_node(&root,&nodes[4]);
   if(count_leaves(root) != 2)
   {
    fprintf(stderr,"count_leaves() returned a wrong value\n");
    exit(1);
   }
   if(tree_height(root) != 4)
   {
    fprintf(stderr,"tree_height() returned a wrong value\n");
    exit(1);
   }
```

```
 search_counter = 0;
  for(int i = 1;i <= 9;i += 2)
    if(search_tree(root,(long)i) == NULL)
      return 1; // impossible if the program is correct
  if(count_function_calls_on_hit(root,0) != search_counter)
  {
    fprintf(stderr,"count_function_calls_of_hit() returned a wrong value\n");
    exit(1);
  }
  search_counter = 0;
  for(int i = 0;i <= 10;i += 2)
    if(search_tree(root,(long)i) != NULL)
      return 1; // impossible if the program is correct
  if(count_function_calls_on_miss(root,0) != search_counter)
  {
    printf("%d -- %d",count_function_calls_on_hit(root,0),search_counter);

    fprintf(stderr,"count_function_calls_of_miss() returned a wrong value\n");
    exit(1);
  }
}
//
// the experiments
//
for(int i = 0;i <= n;i++)
  h_height[i] = h_leaves[i] = 0;
hit[0] = hit[1] = miss[0] = miss[1] = 0.0;
for(int n_experiment = 0;n_experiment < n_experiments;n_experiment++)
{
  rand_perm(n,&a[0]);
  root = NULL;
  for(int i = 0;i < n;i++)
  {
    nodes[i].data = (long)a[i];
    insert_node(&root,&nodes[i]);
  }
  h_height[tree_height(root)]++;
  h_leaves[count_leaves(root)]++;
  x = (double)count_function_calls_on_hit(root,0) / (double)n; // there are n nodes
```

```
  hit[0] += x;
  hit[1] += x * x;
  x = (double)count_function_calls_on_miss(root,0) / (double)(n + 1); // there are n+1 NULL pointers
  miss[0] += x;
  miss[1] += x * x;
}
//
// output summary
//
mean = std = 0.0;
m = n + 1;
M = -1;
for(int i = 0;i <= n;i++)
  if(h_height[i] != 0)
  {
    mean += (double)i * (double)h_height[i];
    std += (double)i * (double)i * (double)h_height[i];
    if(i < m) m = i;
    if(i > M) M = i;
  }
mean /= (double)n_experiments;
std /= (double)n_experiments;
std = sqrt(std - mean * mean);
printf("  %3d %3d %7.4f %6.4f",m,M,mean,std);
fprintf(file,"%7.4f,",mean);
mean = std = 0.0;
m = n + 1;
M = -1;
for(int i = 0;i <= n;i++)
  if(h_leaves[i] != 0)
  {
    mean += (double)i * (double)h_leaves[i];
    std += (double)i * (double)i * (double)h_leaves[i];
    if(i < m) m = i;
    if(i > M) M = i;
  }
mean /= (double)n_experiments;
std /= (double)n_experiments;
std = sqrt(std - mean * mean);
```

```
printf("  %5d %5d %10.4f %8.4f",m,M,mean,std);
  fprintf(file,"%10.4f,",mean);

  mean = hit[0] / (double)n_experiments;
  std = hit[1] / (double)n_experiments;
  std = sqrt(std - mean * mean);
  printf("  %7.4f %6.4f",mean,std);
  fprintf(file,"%6.4f,",mean);
  mean = miss[0] / (double)n_experiments;
  std = miss[1] / (double)n_experiments;
  std = sqrt(std - mean * mean);
  printf("  %7.4f %6.4f",mean,std);
  fprintf(file,"%6.4f\n",mean);
  printf("\n");
  //
  // output the tree height data
  //
  if(details != 0 || n == atoi(argv[2]))
  {
    printf("      i frac height\n");
    printf("  ----- -----------\n");
    for(int i = 0;i <= n;i++)
      if(h_height[i] != 0)
        printf("  %5d %11.9f\n",i,(double)h_height[i] / (double)n_experiments);
    printf("  ----- -----------\n");
  }
  //
  // output the number of leaves data
  //
  if(details != 0 || n == atoi(argv[2]))
  {
    printf("      i frac leaves\n");
    printf("  ----- -----------\n");
    for(int i = 0;i <= n;i++)
      if(h_leaves[i] != 0)
        printf("  %5d %11.9f\n",i,(double)h_leaves[i] / (double)n_experiments);
    printf("  ----- -----------\n");
  }
  //
  // done
  //
  fflush(stdout);
 }
 printf("------  --- --- ------- ------  ----- ----- ---------- --------  ------- ------  ------- ------\n");
 printf("done in %.1f seconds\n",elapsed_time());
 return 0;
}
```

Descrição das funções desenvolvidas

Função tree_height

Para calcular a altura da árvore, se o node for nulo retornamos 0. Se não chamamos recursivamente a função com o node da esquerda e atribuimos o valor de retorno à variavel heigh_left e chamamos recursivamnete a função com o node da direita e atribuimos o valor de retorno à variavel height_right. Por fim se a altura da esquerda for maior que a altura da direita, retornamos a altura da esquerda mais um, para contabilizar o node atual, se não retornamos a altura da direita mais um.

Função search tree

Para procurar um dado na árvore, cada vez que a função é chamada incrementamos a variável search_counter. Se o node for nulo retornamos NULL, se o dado que estamos à procura for igual ao dado do node atual retornamos o node atual e por fim voltamos a chamar a própria função recursivamente com o dado a procurar e com o node da esquerda se o dado a procurar for menor que o dado do node atual e com o node da direita se o dado a procurar for maior que o dado do node atual.

Função count_function_calls_on_hit

Para calcular o custo de procurar um item presente na árvore, analisamos que o custo de procurar um certo node é igual ao nível onde esse node se encontra mais um. Logo criamos uma função recursiva que percorre a árvore e sempre que encontra um node nulo retorna 0 se não for nulo adiciona ao custo total o valor do nivel mais um.

Função count_function_calls_on_miss

Tal como anteriormente o custo de procurar um certo node é igual ao nível mais um. Logo criamos uma função recursiva que percorre a árvore e sempre que encontra um node nulo retorna 0 se não for nulo adiciona ao custo total o valor do nível mais um.

Código para representação dos gráficos

Para a representação dos gráficos e mais especificamente para o ajuste de curvas utilizámos código fornecido pelo professor nas aulas.

```
close all;
M = csvread('Data_100000.csv',1,0);
X=M(:,1);

%% Grafico maximum tree height (mean)
figure(1);
Y=M(:,4);
D=[log(X),1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');
title('Grafico maximum tree height ')
legend("Mean","Curve fitting")

%% Grafico maximum tree height (min)
figure(2);
Y=M(:,2);
D=[log(X),1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');
title('Grafico maximum tree height ')
legend("Minimo","Curve fitting")

%% Grafico maximum tree height (max)
figure(3);
Y=M(:,3);
D=[log(X),1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');
title('Grafico maximum tree height ')
legend("Maximo","Curve fitting")
```

```
%plot(X,Y_mod,'r');

%% Grafico number of leaves (mean)
figure(4);
Y=M(:,7);

D=[X,1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');

title('Grafico number of leaves ')
legend("Mean","Curve fitting","Min","Max")

%% Grafico number of leaves (min)
figure(5);
Y=M(:,5);

D=[X,1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');

title('Grafico number of leaves ')
legend("Min","Curve fitting")

%% Grafico number of leaves (max)
figure(6);
Y=M(:,6);

D=[X,1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');

title('Grafico number of leaves ')
legend("Max","Curve fitting")

%% Grafico calls on hit (mean)
figure(8);
Y=M(:,8);
D=[log(X),1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');
```

```
title('Grafico calls on hit (mean)')
legend("Mean","Curve fitting")

%% Grafico calls on miss (mean)
figure(9);
Y=M(:,9);
D=[log(X),1+0*X];
w=pinv(D)*Y
plot(X,Y,'b',X,D*w,'r');
title('Grafico calls on miss (mean)')
legend("Mean","Curve fitting")
```

Análise de resultados | Output

Para n=100000

Para 100000 árvores binárias aleatórias o calculo de todos os parametros demorou 508.9 segundos.

data for 100000 random trees

	maximum tree height				number of leaves				calls on hit		calls on miss	
n	min	max	mean	std	min	max	mean	std	mean	std	mean	std
10	4	10	5.6407	0.9126	1	5	3.6633	0.7005	3.4432	0.3915	5.0393	0.3559
13	4	12	6.3992	1.0127	2	7	4.6678	0.7881	3.8482	0.4299	5.5019	0.3992
16	5	13	7.0355	1.0851	2	8	5.6690	0.8680	4.1819	0.4553	5.8771	0.4285
20	5	14	7.7409	1.1614	3	10	6.9967	0.9658	4.5564	0.4833	6.2918	0.4603
25	6	16	8.4612	1.2434	4	13	8.6708	1.0806	4.9357	0.5063	6.7074	0.4868
32	6	17	9.2811	1.3153	6	16	11.0011	1.2129	5.3720	0.5285	7.1789	0.5125
40	7	19	10.0438	1.3716	8	19	13.6594	1.3468	5.7741	0.5438	7.6088	0.5305
50	7	18	10.8045	1.4285	10	23	17.0038	1.5055	6.1748	0.5565	8.0341	0.5456
63	8	20	11.6175	1.4824	14	28	21.3324	1.6872	6.6013	0.5671	8.4825	0.5582
79	8	22	12.4343	1.5432	17	34	26.6695	1.8866	7.0299	0.5821	8.9295	0.5748
100	9	23	13.2745	1.5792	23	42	33.6733	2.1202	7.4727	0.5899	9.3888	0.5841
126	10	24	14.1280	1.6297	32	53	42.3349	2.3811	7.9189	0.5999	9.8487	0.5952
158	11	26	14.9789	1.6744	41	64	53.0051	2.6503	8.3574	0.6062	10.2985	0.6024
200	11	28	15.8528	1.7152	53	79	67.0002	2.9928	8.8144	0.6157	10.7656	0.6127
251	12	28	16.7071	1.7445	69	98	84.0010	3.3481	9.2576	0.6219	11.2169	0.6194
316	13	30	17.5830	1.7839	87	121	105.6565	3.7487	9.7127	0.6294	11.6789	0.6274
398	14	29	18.4663	1.8148	114	150	132.9969	4.2098	10.1641	0.6296	12.1361	0.6280
501	14	29	19.3581	1.8481	146	191	167.3117	4.7287	10.6181	0.6324	12.5950	0.6312
631	15	32	20.2450	1.8635	186	233	210.6509	5.2991	11.0733	0.6363	13.0542	0.6352
794	15	32	21.1363	1.8937	238	290	264.9875	5.9423	11.5285	0.6382	13.5128	0.6374
1000	17	34	22.0328	1.9128	304	361	333.6831	6.6616	11.9867	0.6390	13.9737	0.6384
1259	17	34	22.9428	1.9352	385	454	420.0000	7.5071	12.4430	0.6410	14.4324	0.6405
1585	18	36	23.8439	1.9447	493	568	528.6599	8.3627	12.9010	0.6399	14.8923	0.6395
1995	19	36	24.7606	1.9786	627	705	665.3629	9.4171	13.3566	0.6420	15.3494	0.6417
2512	20	37	25.6774	1.9892	788	880	837.6539	10.5423	13.8205	0.6439	15.8146	0.6437
3162	21	40	26.5947	2.0108	1007	1104	1054.3305	11.8457	14.2801	0.6455	16.2753	0.6453
3981	22	39	27.5107	2.0296	1274	1396	1327.3027	13.3028	14.7367	0.6451	16.7327	0.6449
5012	23	41	28.4263	2.0453	1598	1735	1671.0460	14.9880	15.1981	0.6479	17.1949	0.6477
6310	24	45	29.3710	2.0650	2031	2183	2103.6471	16.7691	15.6564	0.6464	17.6538	0.6463
7943	24	44	30.3020	2.0762	2567	2730	2647.9810	18.7554	16.1194	0.6477	18.1173	0.6476
10000	25	43	31.2254	2.0887	3247	3426	3333.6365	21.0794	16.5789	0.6479	18.5771	0.6478

done in 508.9 seconds

Análise de resultados | Gráficos

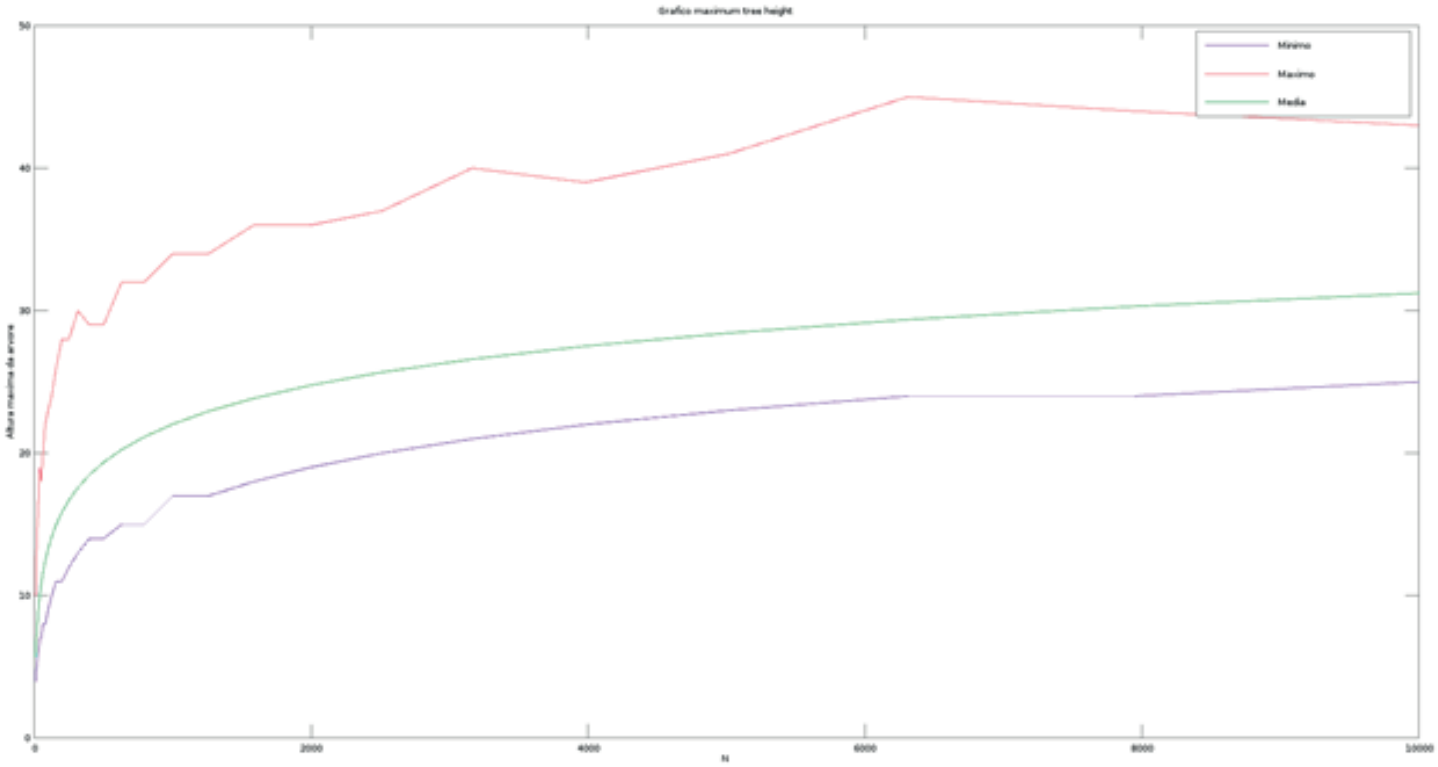

Gráfico de altura máxima

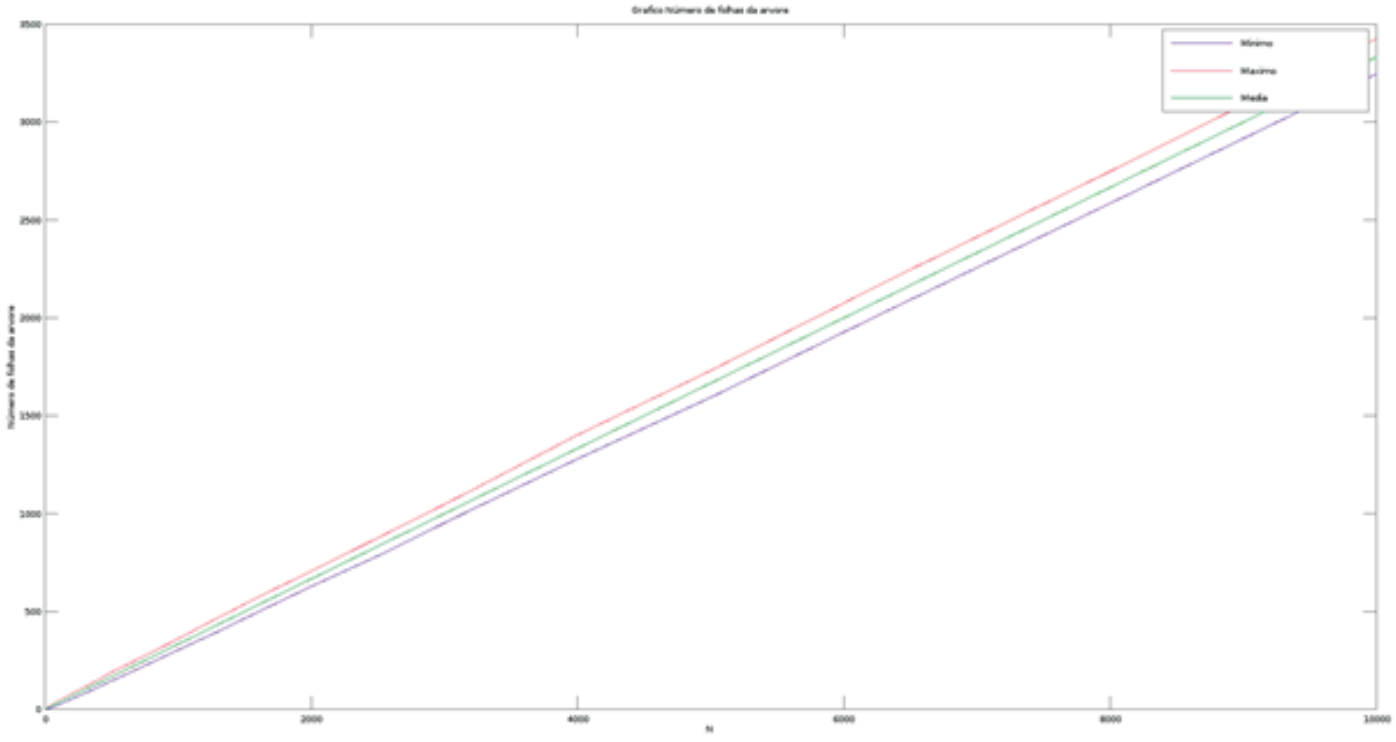

Gráfico de número de folhas

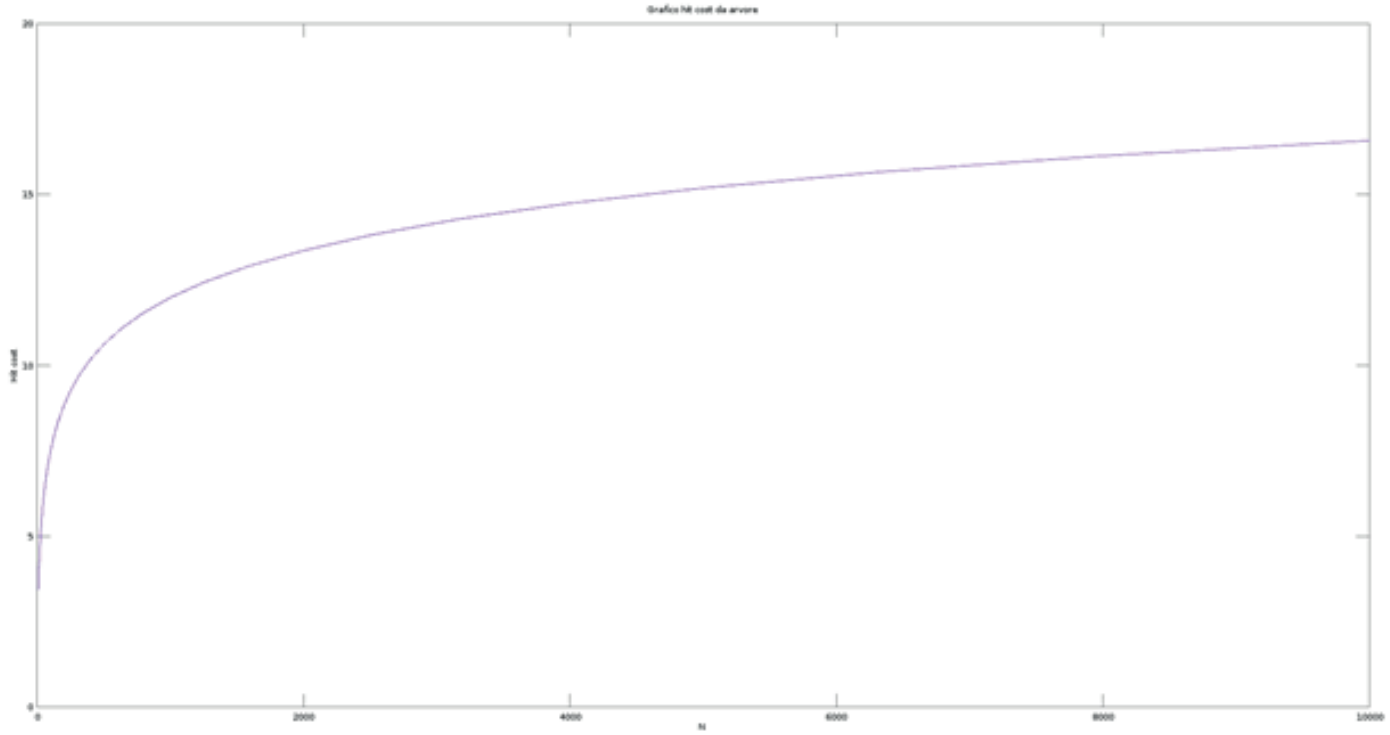

Gráfico Hit Cost

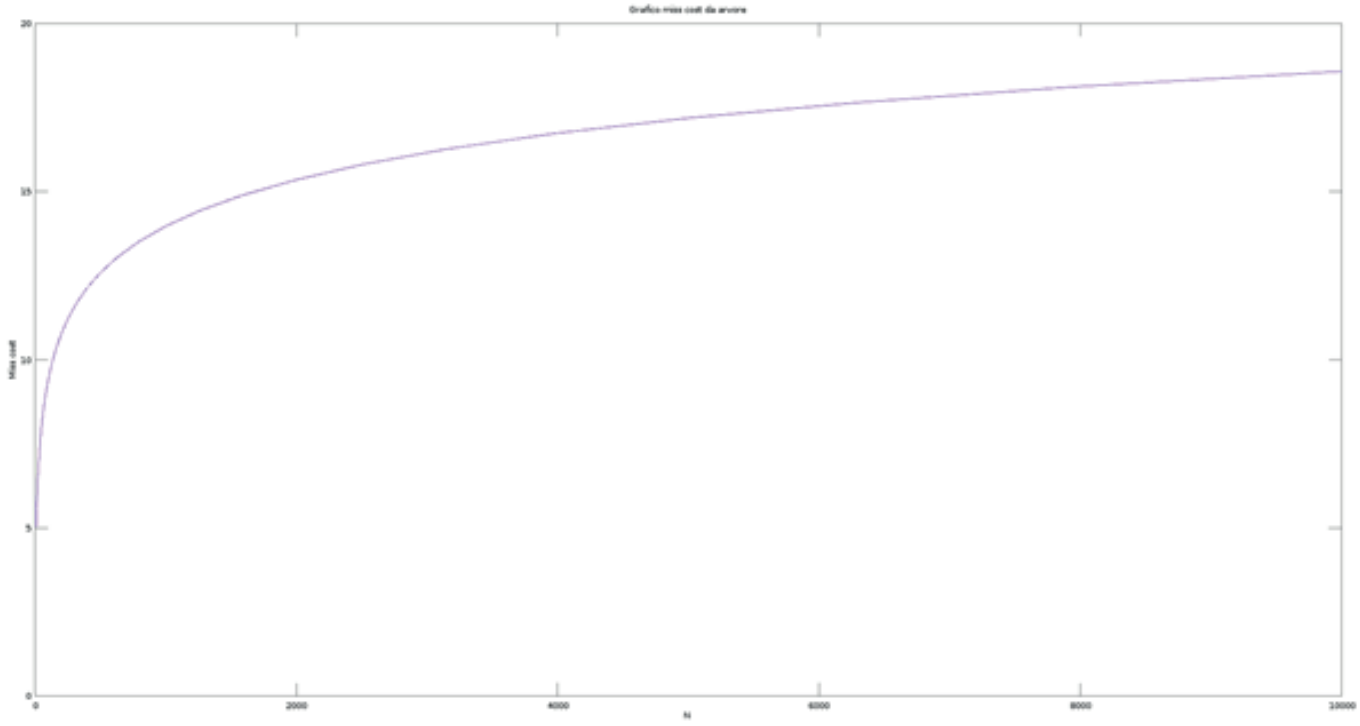

Gráfico de Miss Cost

Ajuste de curvas

Ajuste de Curvas é um método que consiste em encontrar uma curva que se ajuste da melhor forma uma série de pontos. Neste caso vamos apresentar ajuste de curvas na forma de Alog(n)+B e An+B para cada curva referida em cima.

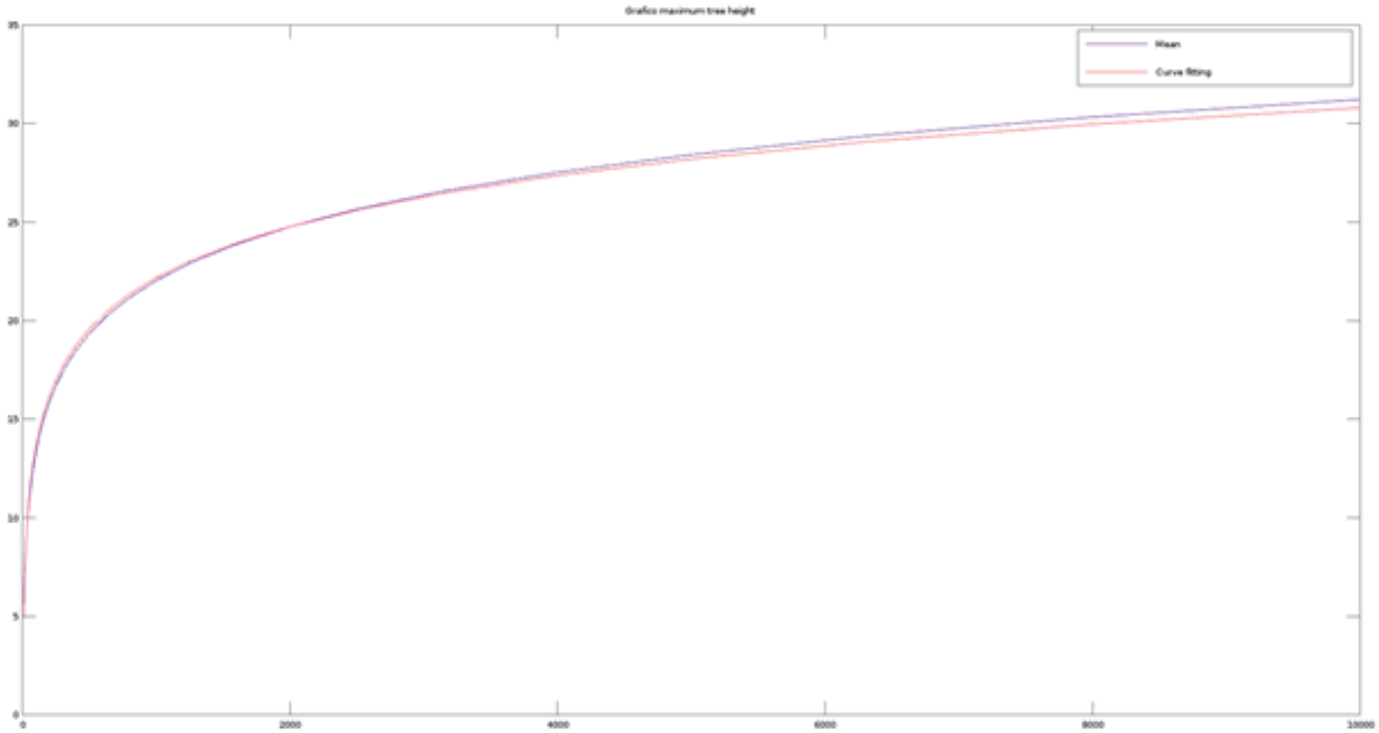

Gráfico de ajuste de curva á media da altura máxima de uma árvore com A=3.7468 e B=-3.7067

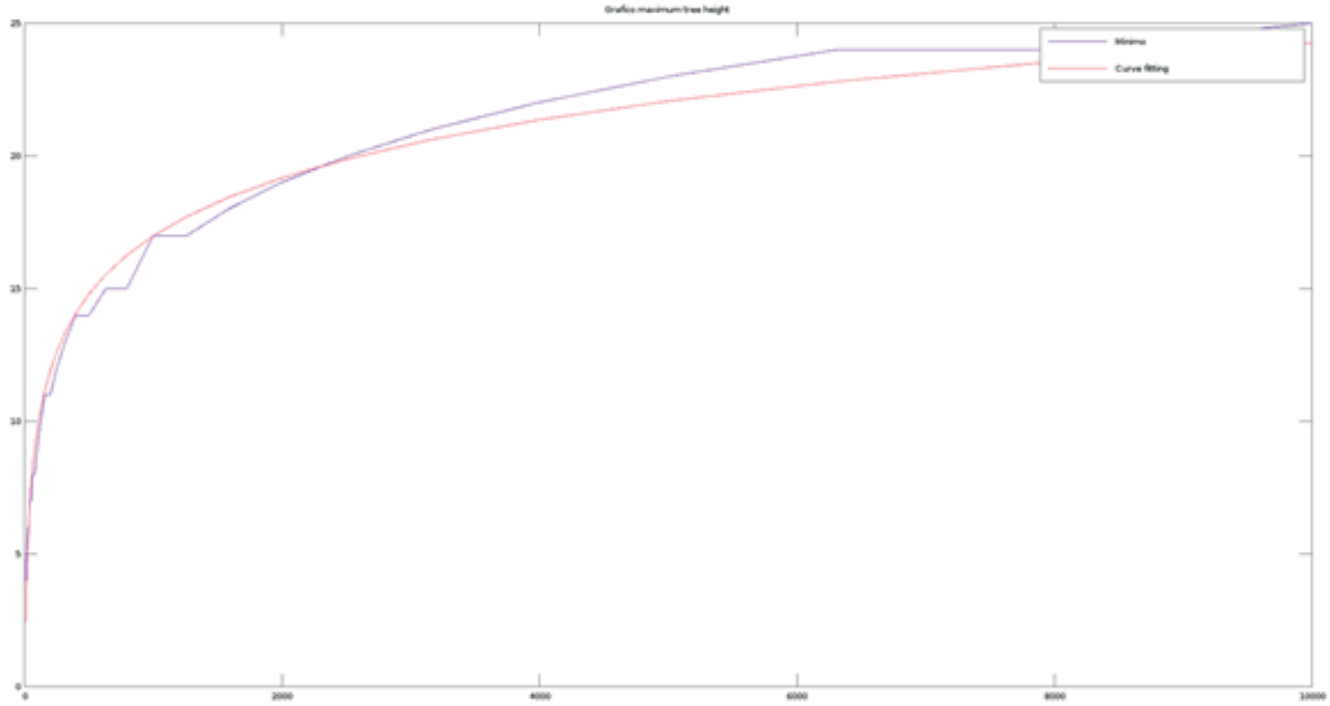

Gráfico de ajuste de curva ao minimo da altura máxima de uma árvore com A=3.1578 e B=-4.8272

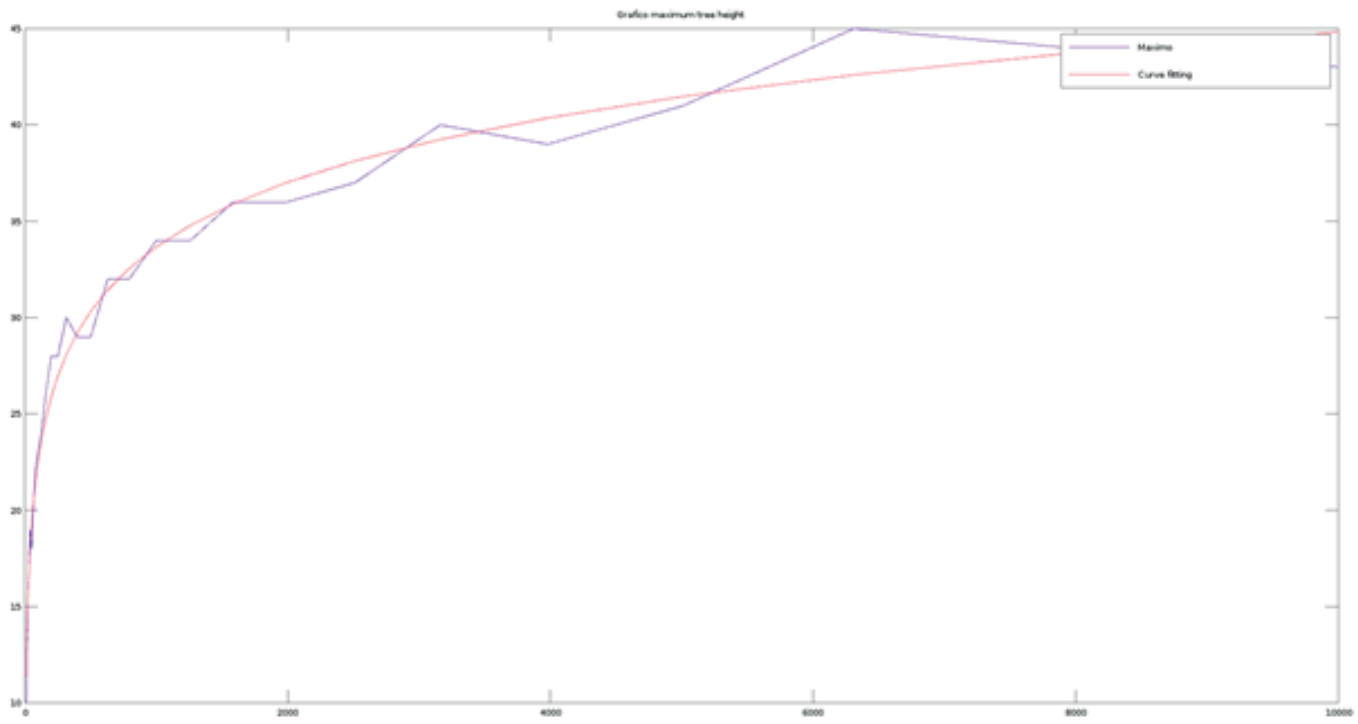

Gráfico de ajuste de curva ao máximo da altura máxima de uma árvore com A=4.8450 e B=0.1963

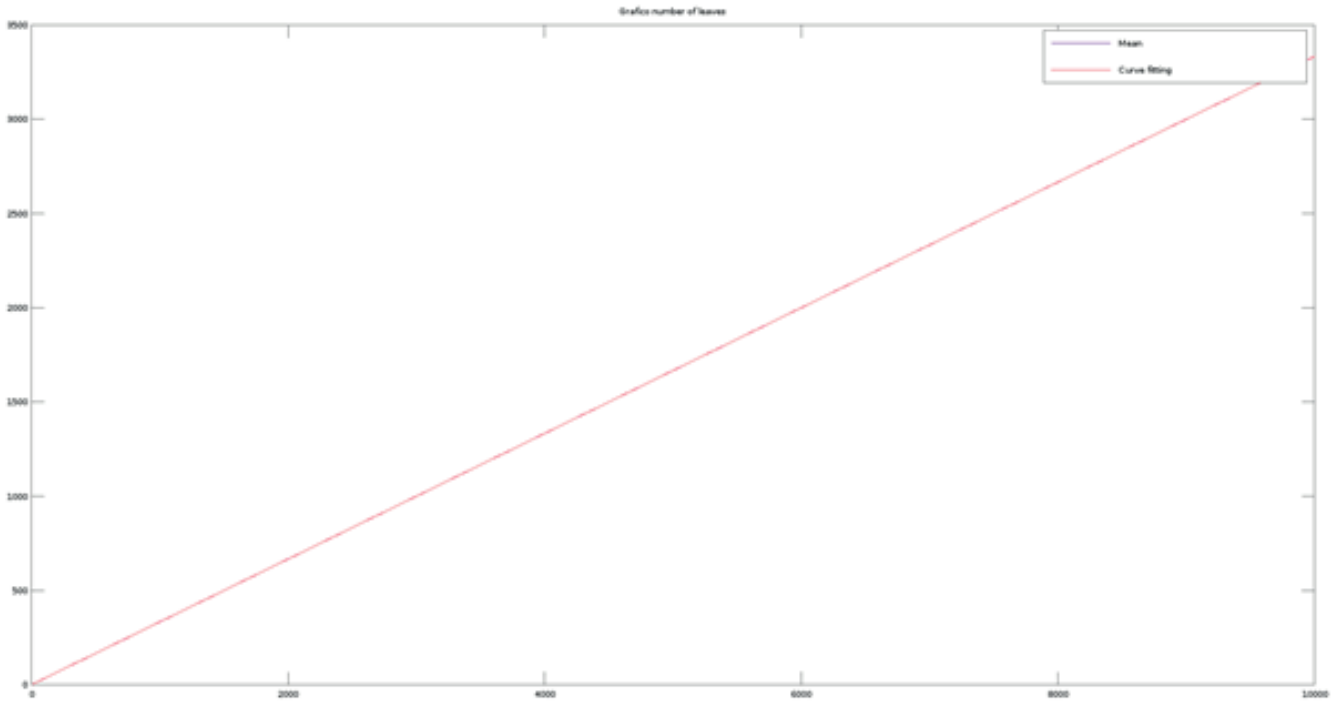

Gráfico de ajuste de curva á media do número de folhas de uma árvore com A=0.3333 e B=0.3336

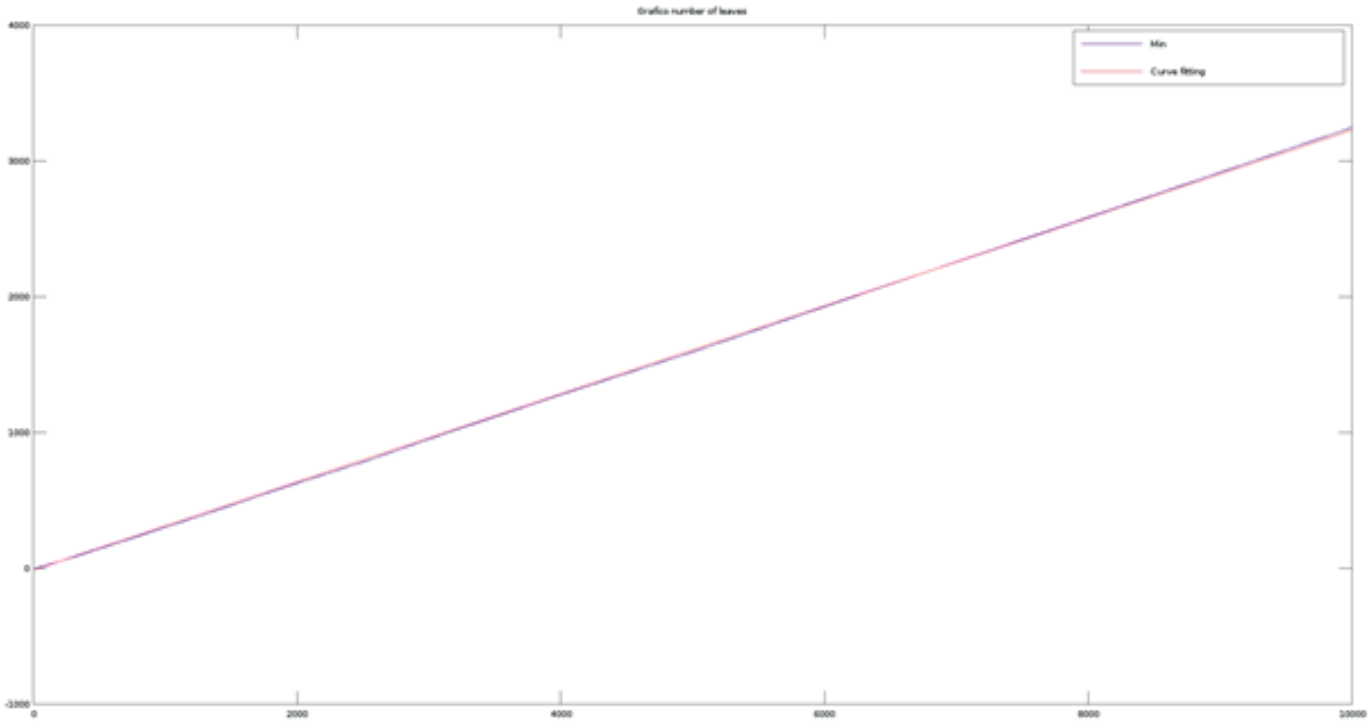

Gráfico de ajuste de curva ao minimo do número de folhas
de uma árvore com A=0.3240 e B=-11.8311

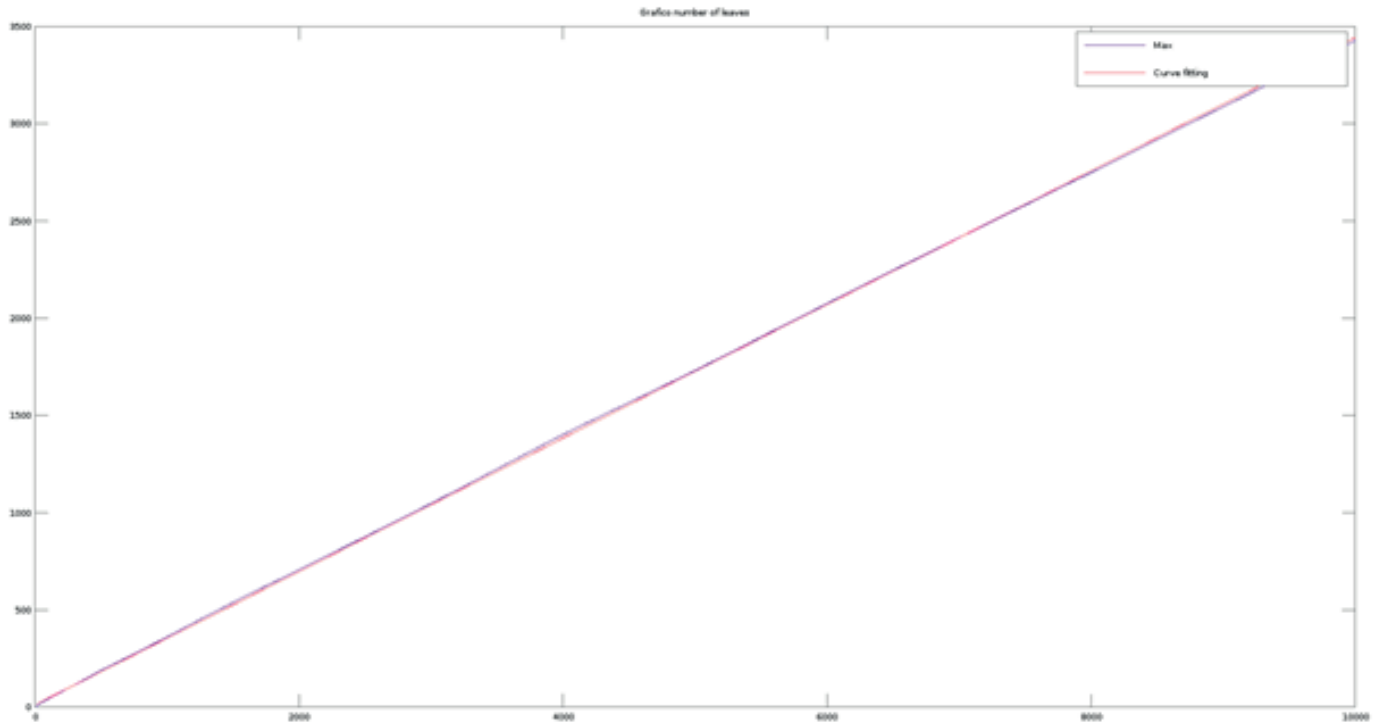

Gráfico de ajuste de curva ao máximo do número de folhas
de uma árvore com A=0.3432 e B=11.3479

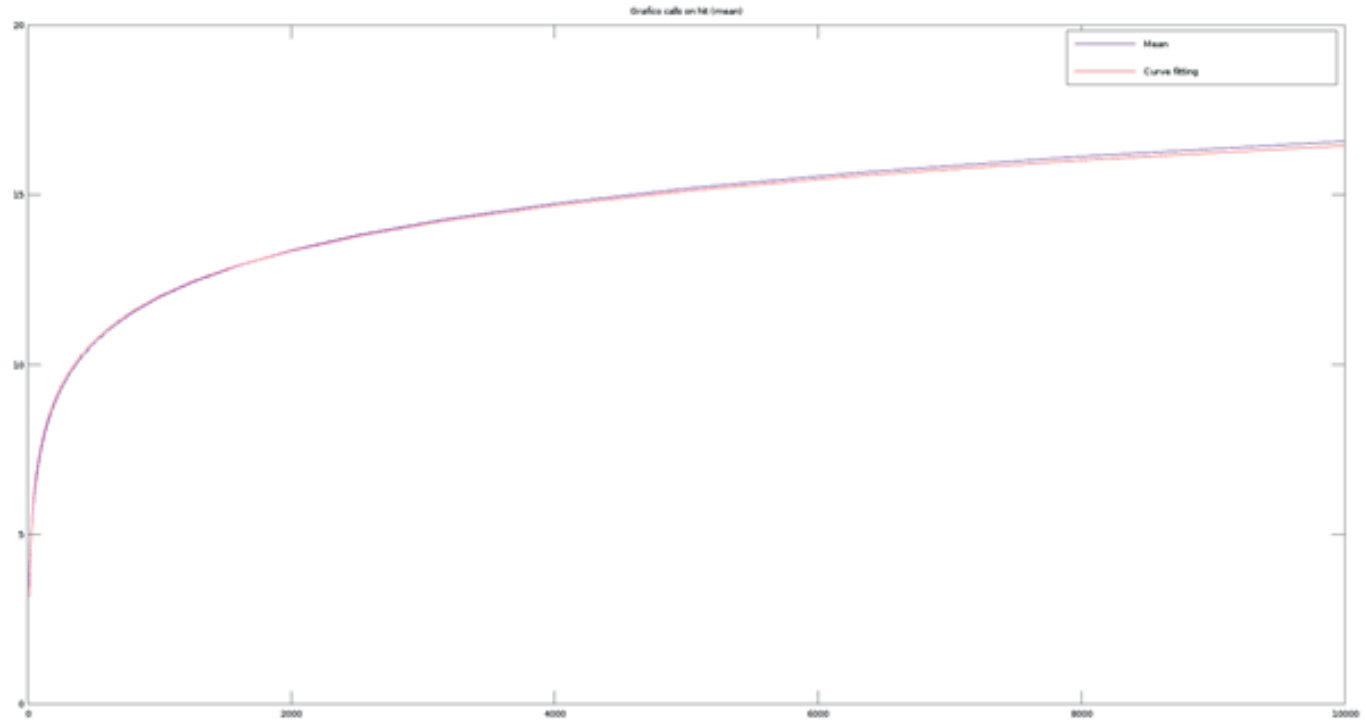

Gráfico de ajuste de curva á media de calls on hit de uma árvore com A=1.9263 e B=-1.2878

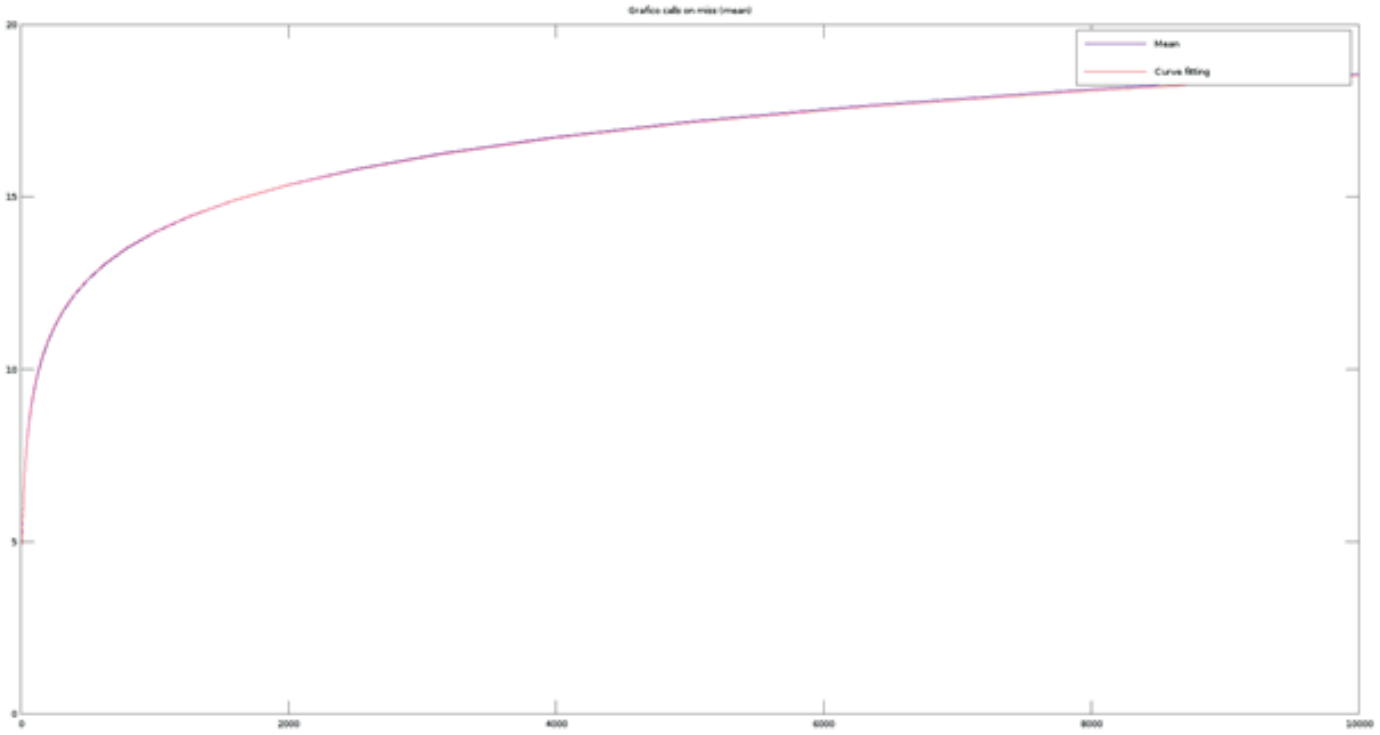

Gráfico de ajuste de curva á media de calls on miss de uma árvore com A=1.9733 e B=0.3527

Conclusões

Através da análise dos gráficos podemos retirar várias conclusões relativas a evolução de cada parâmetro à medida que o número de nodes aumenta.

Em relação à altura máxima de uma árvore binária podemos verificar que após o ajuste a uma curva, a altura mínima, média e máxima evoluem logaritimicamente , ou seja, no início a altura cresce muito rapidamente, mas acaba por estabilizar atingindo um valor máximo de 43 e um valor mínimo de 25.

Em relação ao número de folhas de uma árvore binária podemos verificar que após o ajuste a uma curva, a altura mínima, média e maxima evoluem linearmente , ou seja, no início o número de folhas cresce a um ritmo constante atingindo um valor máximo de 3247 e valor mínimo de 3426.

Em relação ao custo médio de procurar um item que está presente na árvore (hit cost) verificamos que evolui logariticamente, atingindo um valor máximo de 16.58.

Em relação ao custo médio de procurar um item que não está presente na árvore (miss cost) verificamos que evolui logariticamente, atingindo um valor máximo de 18.58.

Se inserirmos os números $1^2,3^2,..,(2n-1)^2$ em vez dos números 1,3,...,(2n-1) em ordem aleatória na árvore conseguimos mudar o custo de computação do cálculo do custo médio de procurar um dado que não está presente na árvore. Se estamos à procura dos números 0,1,...,2n(2n-1), podemos concluir que todos os números que não sejam quadrados de um número inteiro impar não vão estar presentes na árvore.

Logo, com uma verificação se o número é quadrado de um inteiro ímpar podemos assim concluir se é necessário ou não procurar pelo mesmo.

Após a realização deste trabalho prático, concluímos que os objetivos propostos foram alcançados com sucesso. Conseguimos implementar todas as funções pretendidas, obtendo resultados satisfatórios e de acordo com a previsão, como já apresentados anteriormente. Com este trabalho, ambos os alunos do grupo fortaleceram os seus conhecimentos em diversos conceitos de programação abordados na Unidade Curricular de Algoritmos e Estruturas de Dados. É de salientar ainda que o trabalho de equipa e a superação de dificuldades foram fatores importantíssimos no sucesso do trabalho, melhorando as competências interpessoais de ambos os elementos do grupo.

Referências

https://en.wikipedia.org/wiki/Curve_fitting?fbclid=IwAR1ak4_u-FXtSD1596hOCVZ4I91BuvQ5vPiyTbZ-kpHLqMvpyuFiv4c0wxuI

https://www.geeksforgeeks.org/write-a-c-program-to-find-the--maximum-depth-or-height-of-a-tree/?fbclid=IwAR096qClNKCS6cKVcJFS9cU58HofNYEiNFTsV0UnHqx4XLeMZKenhD3qm-Y

https://www.geeksforgeeks.org/write-a-c-program-to-get-count--of-leaf-nodes-in-a-binary-tree/?fbclid=IwAR2jeg-HBJzZLSaKc8EPkXYuIMMn2KbYvVhE1FptFhfehNneGw5wf396AiY

Slides da Unidade Curricular de Algoritmos e Estruturas de Dados da Universidade de Aveiro