BINARY TREES

Algoritmos e Estruturas de dados

universidade
de aveiro

Tomás Oliveira – 89016
Rui Coelho – 86182
João Carvalho - 89059

Índice

Binary Tree

Uma binary tree e uma estrutura de dados dinâmica composta por nodes, cada node contem (no exemplo fornecido na aula P07):

- A própria informação, um long que contem os dados desse node
- Um ponteiro para a esquerda, que aponta para o ramo do lado esquerdo (nas arvores binarias sorted, este ramo se existir o seu valor e mais pequeno que o do node atual)
- Um ponteiro para a direita, que aponta para o ramo do lado direito (nas arvores binarias sorted, este ramo se existir o seu valor e maior que o do node atual)
- Um ponteiro para o node pai, no caso da root, este valor e NULL

Baseando no código fornecido, seria apenas preciso desenvolver 4 funções:

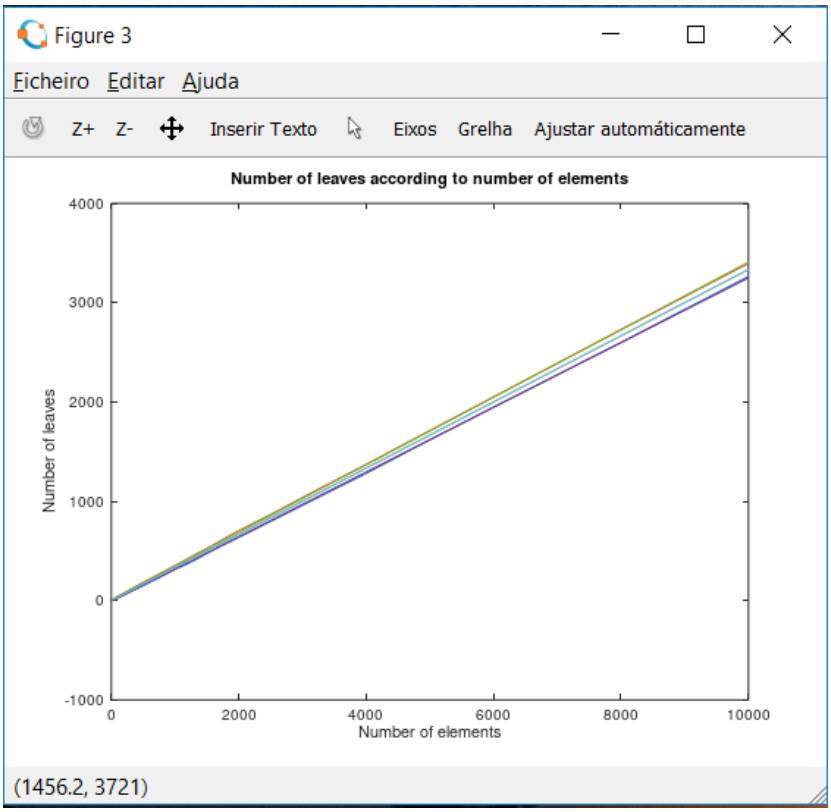
- count_leaves – que devolve o numero de folhas que essa arvore têm.
- tree_height – que devolve a altura dela.
- count_function_calls_on_hit – que nas sucessivas chamadas a funçao devolve o numero de sucessos (acertamos num node).
- count_function_calls_on_miss – oposta a funçao supracitada, devolve o numero de vezes que falhamos (acertamos num ponteiro nulo.

Count Leaves

A ideia geral desta funçao na nossa implementação e percorrer a arvore pela esquerda e depois pela direita, caso o node onde nos situamos nao tenha ponteiro nem para a esquerda nem direita (o que significa que e uma folha), nos retornamos 1 e voltamos atras para continuar a percorrer a arvore, caso o ponteiro seja nulo nos retornamos 0.

Esta função devolve um inteiro correspondente ao numero de folhas de uma dada árvore.

```c
static int count_leaves(tree_node *link)
{
  //caso node dado seja nulo
  if(link == NULL)
    return 0;
  //se nao tiver ponteiro para a esquerda nem direita e pq e uma folha
  if (link->left == NULL && link->right == NULL)
    return 1;
  //caso nao seja folha, incrementamos o n leaves primeiro a esquerda e dps a direita
  //como ja damos return 0 caso seja nulo, podemos sempre chamar o ponteiro para a esquerda
  return count_leaves(link->left) + count_leaves(link->right);
}
```



Tree Height

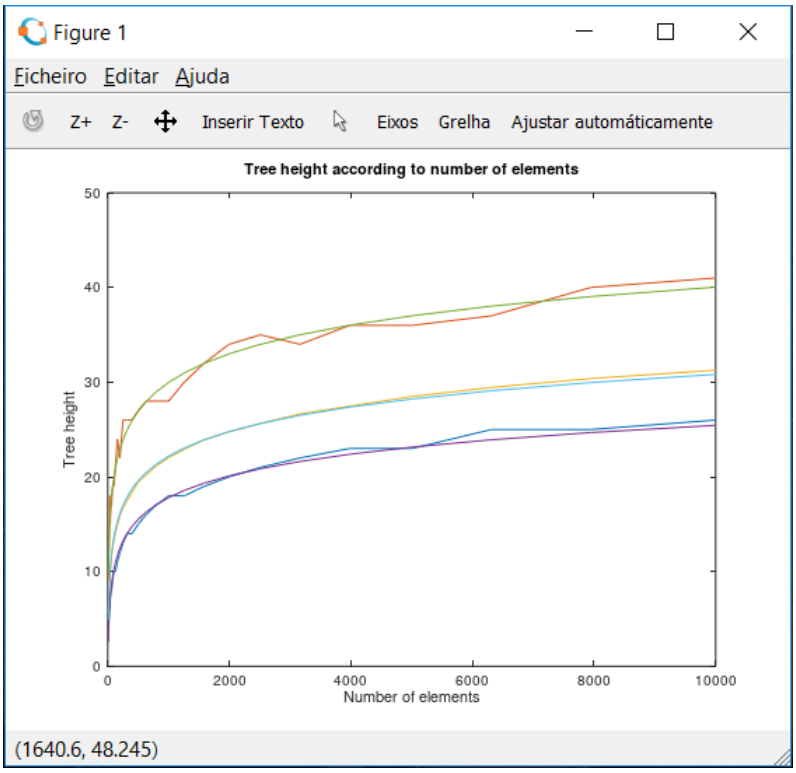
Esta função pretende calcular a altura de uma árvore e para isso e preciso atingirmos o node mais “profundo” da arvore, para tal vamos percorrer a árvore primeiro para a esquerda e depois para a direita, vamos para cada node verificar se a subárvore da esquerda e maior ou menor que a direita, e devolvemos o maior para podermos comparar com as subárvores vizinhas.

Numa primeira versão fazíamos varias validações para não irmos para nodes nulos, como aqui representado:

```
//caso nao haja ponteiro para esquerda nem direita chegamos a uma folha, entao damos return a 1, que e esse nivel
if (link->left == NULL && link->right == NULL)
  return height;

//caso so haja node para a direita
else if(link->left == NULL)
  return height += tree_height(link->right);

//caso so haja node para a esquerda
else if(link->right == NULL)
  return height += tree_height(link->left);
```

Posteriormente, devido a insatisfação do grupo de como o código estava escrito (criticismo que aprendemos na UC), apercebemo-nos que apenas precisamos de verificar se e nulo no momento que entramos no pointer, por isso retornamos 0 se for nulo, como aqui representado:

```
static int tree_height(tree_node *link)
{
  int height = 1;

  if(link == NULL)
    return 0;
  return height += (tree_height(link->left) >= tree_height(link->right)) ? tree_height(link->left) : tree_height(link->right);

}
```



Calls on Hit

Baseando na funçao search_counter fornecida pelo professor, que devolve o numero de chamadas a funçao:

```
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;
  //Se for mais pequeno que o data do node atual ir par a esquerda, senao vamos para a direita
  return search_tree((data < link->data) ? link->left : link->right,data);
}
```

Para vermos o numero de calls on hit, precisamos do nível onde esse node se encontra, logo basta definirmos o nível inicial e para o total retornamos a soma da chamada da função para a esquerda e para a direita, sendo que incrementamos o nível a 1 sempre que o fazemos, como aqui demonstrado:

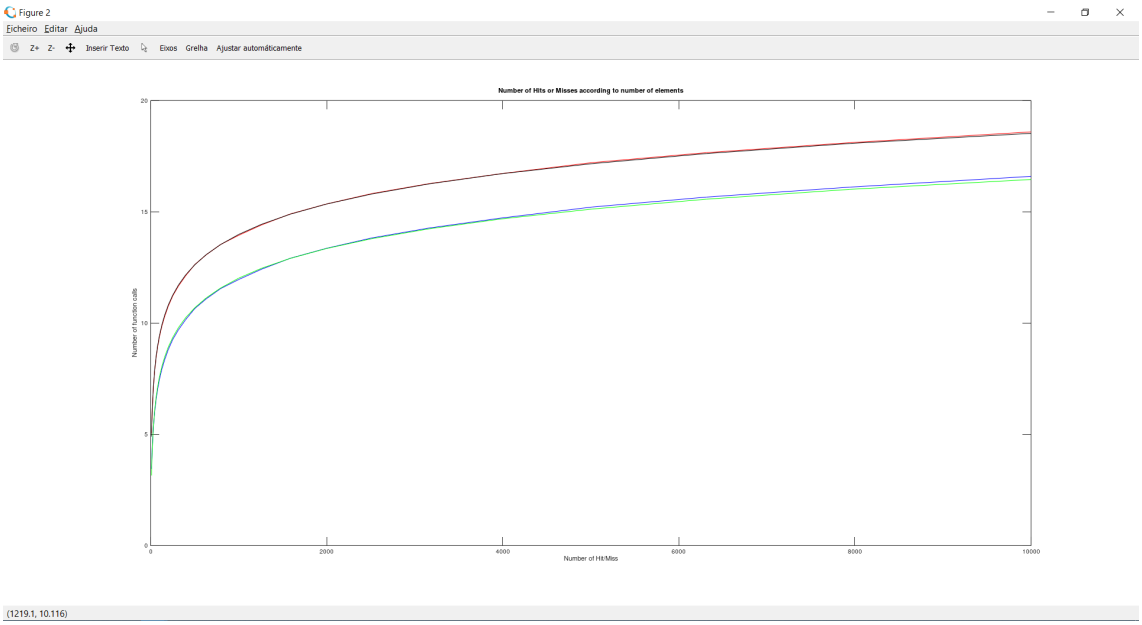
```
static int count_function_calls_on_hit(tree_node *link,int level)
{
  int count;
  // TO DO: delete the next line of code and place your code here
  // if link != null, existe node, o count incrementa o level + 1 = height
  //search_counter tem numero de vezes que func e chamada, root e nivel 0
  if (link == NULL)
    return 0;
  count=level+1;
  return (count += (count_function_calls_on_hit(link->left,level+1) + count_function_calls_on_hit(link->right,level+1)));
}
```

*o gráfico para o calls on hit e partilhado com o calls on miss (próxima pagina)

Calls on Miss

Para o calls on miss fazemos algo semelhante ao que fizemos em cima, mas vamos retornar apenas quando o pointer e nulo, o que significa que foi um miss. Para isso quando não e nulo retornamos a chamada das funçoes para os seus ramos para procurar por nulos:

```
static int count_function_calls_on_miss(tree_node *link,int level)
{
  // TO DO: delete the next line of code and place your code here
  int count=0;
  // TO DO: delete the next line of code and place your code here
  // if link != null, existe node, o count incrementa o level + 1 = height
  //search_counter tem numero de vezes que func e chamada, root e nivel 0
  if (link == NULL)
    return level+1;
  else
    return (count += (count_function_calls_on_miss(link->left,level+1) + count_function_calls_on_miss(link->right,level+1)));
}
```



Legenda: n,callsOnHit -> Azul

n,callsOnMiss-> Vermelho

n,intv*BinaryTreeHits -> Verde

n,intv*BinaryTreeMiss -> Preto

Resultados Obtidos

OUTPUT PARA 100000 nodes

data for 1000 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	9	5.6660	0.9276	2	5	3.6550	0.6663	3.4496	0.3929	5.0451	0.3572
13	4	11	6.4760	1.0166	2	7	4.6090	0.7925	3.8766	0.4382	5.5283	0.4069
16	5	11	6.9840	1.0815	3	8	5.6690	0.8351	4.1732	0.4582	5.8689	0.4313
20	5	12	7.7730	1.2106	4	10	7.0200	0.9755	4.5858	0.5130	6.3198	0.4886
25	6	14	8.4380	1.2273	6	12	8.6610	1.0909	4.9137	0.4921	6.6862	0.4731
32	6	18	9.3130	1.3293	7	15	10.9980	1.2075	5.3568	0.5149	7.1642	0.4993
40	7	15	10.0490	1.3866	10	18	13.7370	1.3667	5.7827	0.5531	7.6172	0.5396
50	8	17	10.8290	1.4669	12	22	16.9810	1.5280	6.1656	0.5472	8.0251	0.5365
63	8	17	11.6550	1.4415	16	26	21.2870	1.7374	6.6122	0.5616	8.4932	0.5528
79	9	19	12.4280	1.5234	20	32	26.6840	1.9251	7.0242	0.5741	8.9239	0.5669
100	10	19	13.2690	1.5462	26	40	33.6090	2.0890	7.4765	0.5943	9.3926	0.5884
126	10	21	14.0990	1.6520	34	50	42.4260	2.4001	7.9116	0.6199	9.8414	0.6150
158	11	24	14.8660	1.6613	45	61	52.8920	2.6496	8.3390	0.6266	10.2803	0.6227
200	12	22	15.9220	1.7105	56	75	66.9240	3.0173	8.8078	0.6067	10.7590	0.6037
251	13	26	16.7430	1.7723	73	94	83.8800	3.4845	9.2634	0.6333	11.2226	0.6308
316	14	26	17.4770	1.8252	94	118	105.4940	3.8108	9.7014	0.6461	11.6677	0.6441
398	14	26	18.3540	1.8299	119	147	133.0570	4.2128	10.1573	0.6345	12.1293	0.6329
501	15	27	19.4450	1.9050	151	184	167.4320	4.6257	10.6527	0.6454	12.6295	0.6441
631	16	28	20.2120	1.8818	195	227	210.6580	5.2072	11.0874	0.6595	13.0683	0.6584
794	17	28	21.1270	1.8727	245	283	265.0850	5.7329	11.5475	0.6390	13.5317	0.6382
1000	18	28	22.0320	1.9321	314	352	333.4960	6.7206	11.9567	0.6155	13.9437	0.6149
1259	18	30	22.9100	1.9854	390	443	420.3390	7.2932	12.4196	0.6390	14.4090	0.6385
1585	19	32	23.9210	1.9034	504	556	529.2560	8.6191	12.9113	0.6535	14.9025	0.6531
1995	20	34	24.7630	1.9972	635	702	665.1490	9.3587	13.3573	0.6293	15.3501	0.6290
2512	21	35	25.6390	2.0007	801	870	837.5910	10.2995	13.8255	0.6370	15.8196	0.6368

3162	22	34	26.6500	1.9818	1012	1093	1054.6180	11.2465	14.2711	0.6357	16.2663	0.6355
3981	23	36	27.4670	2.0719	1275	1364	1327.6370	13.0179	14.7190	0.6509	16.7151	0.6507
5012	23	36	28.4960	2.1133	1622	1716	1671.1720	14.4025	15.2136	0.6837	17.2104	0.6836
6310	25	37	29.4380	2.0504	2049	2156	2103.9750	16.5256	15.6602	0.6573	17.6576	0.6572
7943	25	40	30.3810	2.1568	2580	2708	2647.0270	19.0916	16.1093	0.6480	18.1071	0.6479
10000	26	41	31.2600	2.0848	3264	3395	3333.4050	21.0672	16.5935	0.6298	18.5917	0.6298
12589	28	41	32.1130	2.0981	4123	4293	4197.9490	23.3098	17.0225	0.6441	19.0211	0.6441
15849	28	42	33.1390	2.0952	5201	5361	5283.8170	26.1192	17.5018	0.6746	19.5007	0.6746
19953	29	42	33.9520	2.0109	6552	6766	6650.1380	29.7318	17.9617	0.6345	19.9608	0.6344
25119	30	47	34.9060	2.1178	8264	8478	8373.7560	34.6661	18.4154	0.6708	20.4146	0.6708
31623	31	44	35.8550	2.1932	10427	10652	10541.4810	36.8547	18.8874	0.6488	20.8868	0.6488
39811	32	49	36.8320	2.2068	13141	13390	13272.9340	40.7183	19.3695	0.6761	21.3689	0.6760
50119	33	46	37.7060	2.1207	16548	16859	16706.7250	48.5113	19.8277	0.6361	21.8273	0.6361
63096	33	48	38.8280	2.1271	20861	21174	21034.7400	53.5762	20.2796	0.6390	22.2793	0.6390
79433	34	48	39.6850	2.1637	26272	26655	26474.7520	57.8774	20.7219	0.6542	22.7216	0.6541
100000	36	50	40.5690	2.1409	33125	33525	33336.8960	66.8484	21.1564	0.6184	23.1562	0.6184

------ --- --- ------- ------ ----- ----- ---------- -------- ------- ------ ------- ------

done in 153.5 seconds

Código Matlab/Octave

```
%% Load file that we made while running binary_tree
InfoCols=load('info.data');

%% Read N nodes
n=InfoCols(:,1);

% Read Tree Height values
TreeHeightMin=InfoCols(:,2);
TreeHeightMax=InfoCols(:,3);
TreeHeightAvg=InfoCols(:,4);

% Read Tree Leaves values
TreeLeavesMin=InfoCols(:,6);
TreeLeavesMax=InfoCols(:,7);
TreeLeavesAvg=InfoCols(:,8);

% Read calls on hit and miss respectively
callsOnHit=InfoCols(:,10);
callsOnMiss=InfoCols(:,12);

clear InfoCols;
```

```
%Tree Height Figure
intv = [log10(n),1+0*n];
frm = pinv(intv);
BinTreeHeightMin= frm * TreeHeightMin;
BinTreeHeightMax= frm * TreeHeightMax;
BinTreeHeightAvg= frm * TreeHeightAvg;

figure(1)
plot(n,TreeHeightMin);
hold on
plot(n,TreeHeightMax);
hold on
plot(n,TreeHeightAvg);
hold on
plot(n,intv*BinTreeHeightMin);
hold on
plot(n,intv*BinTreeHeightMax);
hold on
plot(n,intv*BinTreeHeightAvg);
hold off

title('Tree height according to number of elements')
xlabel('Number of elements')
ylabel('Tree height')
```

```
%% Number of leaves

intv = [n,1+0*n];
frm = pinv(intv);
BinTreeLeaveMin= frm * TreeLeavesMin;
BinTreeLeavesMax= frm * TreeLeavesMax;
BinTreeLeavesAvg= frm * TreeLeavesAvg;

figure(3)
plot(n,TreeLeavesMin);
hold on
plot(n,TreeLeavesMax);
hold on
plot(n,TreeLeavesAvg);
hold on
plot(n,intv*BinTreeLeaveMin);
hold on
plot(n,intv*BinTreeLeavesMax);
hold on
plot(n,intv*BinTreeLeavesAvg);
hold off

title('Number of leaves according to number of elements')
xlabel('Number of elements')
ylabel('Number of leaves')
```

```
%% Calls on Hit and on Miss

BinTreeHits= frm * callsOnHit;
BinTreeMiss= frm * callsOnMiss;

figure(2)
plot(n,callsOnHit);
hold on
plot(n,callsOnMiss);
hold on
plot(n,intv*BinTreeHits);
hold on
plot(n,intv*BinTreeMiss);
hold off

title('Number of Hits or Misses according to number of elements')
xlabel('Number of Hit/Miss')
ylabel('Number of function calls')
```

Conclusões

Aa

Bibliografia

1. https://elearning.ua.pt/pluginfile.php/214852/mod_resource/content/48/AED.pdf, Tomás Oliveira e Silva, Universidade de Aveiro, Portugal

2. https://www.geeksforgeeks.org/calculate-depth-full-binary-tree-preorder/