AED

Algoritmos e Estrutura de Dados

Random Ordered Binary Trees

**Introdução**

Para a disciplina de Algoritmos e Estrutura de Dados foi-nos proposto o estudo de arvores binarias ordenadas com n elementos, nomeadamente a média da altura da arvore, o número medio de folhas, e o custo médio de encontrar elementos na arvore, quer estejam presentes ou não.

Para tal desenvolvemos funções na linguagem C que juntamente com um programa previamente oferecido pelo professor da cadeira, cria várias arvores binarias de tamanho crescente até 10000 elementos, calcula o seu tamanho, número de folhas e verifica se alguns números pertencem à arvore e devolve a media de tempo que cada número demorou a procurar.

Para criar as arvores o programa insere os números 1,3,5,7…, 2n-1 por ordem aleatória, sendo n o número de elementos da arvore.

**Count leaves**

Nas arvores binarias, as folhas são os nodes cujos ambos os ponteiros (pointeiro para a esquerda e ponteiro para a direita) são nulos

O número de folhas é calculado recursivamente através da função count\_leaves, quando esta é chamada verifica se o ponteiro para a esquerda ou o ponteiro para a direita do node onde está são não nulos, caso não sejam ela é invocada nos nodes seguintes. Quando chega a um node com ponteiros nulos, acrescenta 1 ao número de folhas e no final devolve o resultado.

Seguidamente o programa calcula o mínimo, o máximo, a media e a standard deviation/variância (std) de folhas das arvores geradas aleatoriamente

O número de folhas nunca é maior que o número de elementos

O número de folhas possíveis é entre 1, se todos os nodes da arvore tiverem apenas um node filho, e 2h-1 sendo h a altura da arvore

Nº folhas ∈ [1, 2h-1 ]

Numa arvore cheia(full), ou seja, que todos os nodes têm 0 ou 2 nodes filhos o número de folhas é igual ao número de nodes com 2 nodes filhos + 1 ou (n+1)/2

Dados para o número de folhas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | min | max | mean | std |
| 10 | 2 | 5 | 3.6540 | 0.6843 |
| 13 | 2 | 7 | 4.6940 | 0.7631 |
| 16 | 3 | 8 | 5.6700 | 0.8585 |
| 20 | 4 | 10 | 6.9570 | 0.9824 |
| 25 | 5 | 13 | 8.6460 | 1.1174 |
| 32 | 8 | 14 | 11.0430 | 1.2013 |
| 40 | 9 | 18 | 13.6440 | 1.3451 |
| 50 | 11 | 23 | 16.9760 | 1.5399 |
| 63 | 16 | 27 | 21.3560 | 1.6998 |
| 79 | 22 | 33 | 26.7660 | 1.8053 |
| 100 | 26 | 41 | 33.6570 | 2.1188 |
| 126 | 34 | 51 | 42.3280 | 2.3559 |
| 158 | 44 | 61 | 53.0230 | 2.7067 |
| 200 | 59 | 78 | 67.1120 | 3.0538 |
| 251 | 73 | 93 | 83.7730 | 3.3999 |
| 316 | 92 | 117 | 105.6710 | 3.6566 |
| 398 | 119 | 149 | 133.0730 | 4.3282 |
| 501 | 152 | 181 | 167.4630 | 4.8607 |
| 631 | 194 | 227 | 210.8870 | 5.4547 |
| 794 | 246 | 281 | 265.1700 | 5.8470 |
| 1000 | 308 | 355 | 333.6360 | 6.5544 |
| 1259 | 400 | 444 | 420.0940 | 7.4120 |
| 1585 | 508 | 555 | 529.0860 | 7.9671 |
| 1995 | 631 | 694 | 665.4450 | 9.4760 |
| 2512 | 807 | 871 | 838.3090 | 10.4556 |
| 3162 | 1012 | 1094 | 1054.1410 | 11.8696 |
| 3981 | 1280 | 1371 | 1326.9490 | 13.3436 |
| 5012 | 1623 | 1726 | 1670.1980 | 15.0302 |
| 6310 | 2054 | 2167 | 2104.0740 | 16.4703 |
| 7943 | 2583 | 2720 | 2647.9980 | 18.9578 |
| 10000 | 3267 | 3411 | 3332.4030 | 20.8589 |

Pelo gráfico podemos ver que o número medio de folhas cresce linearmente com o número de elementos no gráfico.

Porem os valores obtidos são apenas uma média e devido às propriedades dos gráficos, um gráfico pode ter mais nodes e mesmo assim ter menos folhas que um outro com menos nodes e quanto maior o número de nodes maior a probabilidade de tal acontecer, visto que a variância aumenta ao mesmo tempo que o nº de nodes aumenta

**Tree height**

A altura de uma arvore é o número de nível mais alto da arvore, um nível é contado sempre que cada node tem pelo menos um ponteiro não nulo, sendo a raiz considerada normalmente como nível 0.

Para o cálculo da altura das arvores é usada a função tree\_height que subdivide a arvore em várias arvores mais pequenas. Começando por calcular a altura da arvore da esquerda do node raiz e depois a altura da arvore da direita do node, retornando o valor mais alto. Sempre que um node é nulo a função retorna 0, pois não se pode contar o node não existente para a altura da arvore

O programa calcula também o mínimo, máximo, media e variância da altura

A altura máxima seria num caso degenerado em que cada node teria um filho e a altura seria igual a N

Para qualquer arvore binaria a altura mínima é log2 ( N )

Uma arvore binaria com L folhas tem pelo menos log2( L ) + 1 níveis

Dados para a altura das arvores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | min | max | mean | std |
| 10 | 4 | 9 | 5.6750 | 0.8885 |
| 13 | 4 | 11 | 6.3510 | 0.9828 |
| 16 | 5 | 12 | 7.0760 | 1.1438 |
| 20 | 5 | 12 | 7.7630 | 1.1759 |
| 25 | 6 | 13 | 8.4590 | 1.2338 |
| 32 | 6 | 16 | 9.2800 | 1.3075 |
| 40 | 7 | 15 | 10.0680 | 1.3976 |
| 50 | 8 | 17 | 10.7980 | 1.4047 |
| 63 | 9 | 17 | 11.5900 | 1.4470 |
| 79 | 9 | 20 | 12.4260 | 1.5371 |
| 100 | 10 | 20 | 13.2110 | 1.4834 |
| 126 | 11 | 21 | 14.1590 | 1.6192 |
| 158 | 11 | 23 | 15.0280 | 1.7231 |
| 200 | 12 | 24 | 15.9160 | 1.7941 |
| 251 | 13 | 25 | 16.6120 | 1.6809 |
| 316 | 13 | 25 | 17.5270 | 1.6880 |
| 398 | 14 | 25 | 18.4350 | 1.8253 |
| 501 | 15 | 29 | 19.3570 | 1.8203 |
| 631 | 16 | 30 | 20.2880 | 1.9378 |
| 794 | 16 | 29 | 21.0880 | 1.8773 |
| 1000 | 18 | 29 | 22.0120 | 1.8498 |
| 1259 | 18 | 31 | 22.9660 | 1.9165 |
| 1585 | 19 | 30 | 23.7820 | 1.8790 |
| 1995 | 20 | 36 | 24.7970 | 2.0287 |
| 2512 | 21 | 34 | 25.7570 | 2.0420 |
| 3162 | 21 | 37 | 26.6200 | 2.0469 |
| 3981 | 22 | 35 | 27.5410 | 2.0155 |
| 5012 | 24 | 37 | 28.4900 | 2.0596 |
| 6310 | 25 | 37 | 29.4390 | 2.1578 |
| 7943 | 26 | 40 | 30.2350 | 2.1203 |
| 10000 | 27 | 42 | 31.2180 | 2.0670 |

Vemos que a média da altura das arvores cresce com base na função logaritma, isto deve-se à arvore estar organizada e os números gerados aleatoriamente não distam muito entre si, o que faz com vários níveis da arvore estejam preenchidos.

A variância também não cresce não muito, comparativamente à variância do número de folhas.

**Functions calls on hit**

Para calcular o custo de encontrar um elemento pertencente à arvore implementamos a função functions calls on hit que calcula quantas vezes é invocada a função para procurar um elemento da arvore

Dados para function calls on hit

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | mean | std |  | n | mean | std |
| 10 | 1727,9079 | 995,3592 |  | 398 | 35910,2473 | 2931,1452 |
| 13 | 4573,8605 | 1106,3924 |  | 501 | 37868,7756 | 3072,8061 |
| 16 | 7368,0244 | 1212,9399 |  | 631 | 39835,3319 | 3200,0696 |
| 20 | 9862,6388 | 1319,483 |  | 794 | 41817,4164 | 3325,8279 |
| 25 | 12188,927 | 1428,3338 |  | 1000 | 43772,0618 | 3460,972 |
| 32 | 14136,2182 | 1547,9115 |  | 1259 | 45742,8441 | 3582,4391 |
| 40 | 16362,6321 | 1670,4442 |  | 1585 | 47727,7485 | 3718,8585 |
| 50 | 18487,089 | 1780,6916 |  | 1995 | 49736,0428 | 3863,6593 |
| 63 | 20422,8532 | 1906,3131 |  | 2512 | 51721,2077 | 3992,5758 |
| 79 | 22432,158 | 2028,914 |  | 3162 | 53750,0568 | 4134,6306 |
| 100 | 24217,6795 | 2152,2804 |  | 3981 | 55736,1395 | 4246,6413 |
| 126 | 26149,9565 | 2291,4393 |  | 5012 | 57717,8177 | 4390,8675 |
| 158 | 28199,9588 | 2416,1549 |  | 6310 | 59710,9548 | 4511,9743 |
| 200 | 29992,9779 | 2549,5497 |  | 7943 | 61708,6284 | 4653,1345 |
| 251 | 32029,2524 | 2664,9504 |  | 10000 | 63686,1239 | 4780,5411 |
| 316 | 33967,3905 | 2806,2587 |  |

Os valores da média são os valores da soma de média de 1000 arvores, para a media de cada arvore basta dividir o valor da média por 1000

Como a arvore está organizada a pesquisa por um elemento da arvore vai ser facilitada, pois na pesquisa apenas basta verificar se o valor do node onde nos encontramos é superior ou não ao valor pretendido, caso sim avançar para o node da esquerda(ou direita, dependendo de como a arvore está organizada), caso não avançar para o contrário e repetir até encontrar o valor

O tempo tal como a media de vezes que a função é invocada cresce com base na função logaritma, pelas mesmas razoes apresentadas para a média.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |