Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciência da Computação

PROJETO E ANÁLISE DE ALGORITMOS

Primeiro trabalho prático – disponível em:
http://www.dcc.ufmg.br/~anísio/PAATP1/docs
Código fonte disponível em:
http://www.dcc.ufmg.br/~anísio/sc

Anísio Mendes Lacerda

Professor - Nivio Ziviani

Belo Horizonte
4 de abril de 2006

Sumário

1 Avaliação de somas

Este exercício consite na análise de um conjunto de somas, como segue.

1. $\sum_{i=1}^{n} i2^{i-1}$

   Resposta: Para chegar à identidade proposta, podemos fazer o seguinte desenvolvimento:

$$\sum_{i=1}^{n} i2^{i-1} = \sum_{i=1}^{n} i\frac{2^i}{2} = \frac{1}{2}\sum_{i=1}^{n} i2^i \tag{1}$$

   Uma vez que $\sum_{i=1}^{n} i2^i$ é da forma: $\sum_{i=1}^{n} ia^i$, iremos resolver a fórmula geral e aplicar o resultado para o somatório dado:

   $S_n = \sum_{i=1}^{n} ia^i$

   Expandindo o somatório temos:

   $S_n = 1a^1 + 2a^2 + 3a^3 + \ldots + na^n$

   Multiplicando-se S_n por a resulta em:

   $aS_n = 1a^2 + 2a^3 + 3a^4 + \ldots + na^{n+1}$

   Fazendo $S_n - aS_n$:

$$\begin{aligned} S_n - aS_n &= a - a^2 + 2a^2 + \ldots + a^n - na^{n+1} \\ &= a + a^2 + \ldots + a^n - na^{n+1} \\ &= \sum_{i=1}^{n} a^i - na^{n+1} \end{aligned}$$

   Colocando-se S_n em evidência temos:

$$\begin{aligned} S_n\,(1-a) &= \sum_{i=1}^{n} a^i - na^{n+1} \\ &= a\tfrac{a^n-1}{a-1} - na^{n+1} \\ &= \tfrac{a^{n+1}-a}{a-1} \\ &= \tfrac{a-a^{n+1}}{1-a} \\ &= \tfrac{a-a^{n+1}-na^{n+1}(1-a)}{1-a} \\ &= \tfrac{a-a^{n+1}-na^{n+1}+na^{n+2}}{1-a} \end{aligned}$$

Enfim temos S_n dado por:

$$S(n) = \begin{cases} \frac{na^{n+2} - a^{n+1}(1+n)+a}{(1-a)^2} & \text{se } a \neq b, \\ \\ \frac{n(n+1)}{2} & \text{se } a = b \end{cases}$$

Para o somatório em pedido temos $a = 2$, logo substituindo em 1 temos:

$$\begin{aligned} \sum_{i=1}^{n} i2^{i-1} &= \frac{1}{2}\sum_{i=1}^{n} i2^{i} \\ &= \frac{1}{2}\frac{n2^{n+2} - 2^{n+1}(n+1)+2}{(1-2)^2} \\ &= \frac{1}{2}\frac{n\ 2^n\ 2^2 - 2^n\ 2(n+1)+2}{1} \\ &= \frac{1}{2}n\ 2^n\ 2^2 - 2^n\ 2\ n - 2^n\ 2 + 2 \\ &= \frac{1}{2}n\ 2^n\ 2 - 2^n\ 2 + 2 \\ &= \frac{1}{2}2\left(2^n\left(n-1\right)+1\right) \\ &= 2^n\left(n-1\right)+1 \end{aligned}$$

Portanto:

$$\boxed{S(n) = 2^n\left(n-1\right)+1}$$

2. $\sum_{i=1}^{n} \frac{1}{i}$

Resposta: O desenvolvimento apresentado a seguir visa estabelecer funções para os limites inferior e superior do somatório. Para isto, será usada uma técnica conhecida como *aproximação por integrais*: duas funções descrevem os limites inferior e superior de cada termo do somatório e são integradas a seguir.

Como o objetivo é trabalhar com integrais, deve-se arranjar uma maneira de representar a função dos termos do somatório de maneira contínua no espaço das abscissas. Assim, define-se a função $s\left(x\right)$:

$$s\left(x\right) = \frac{1}{\lfloor x+1 \rfloor}$$

Além disso, define-se duas funções, $f\left(x\right)$ e $g\left(x\right)$, que delimitam $s\left(x\right)$:

$$f\left(x\right) = \frac{1}{x+1}$$

$$g(n) = \begin{cases} 1 & \text{se } 0 \leq x < 1 \\ \\ \frac{1}{x} & \text{se } x \geq 1 \end{cases}$$

Todas são funções monotonicamente decrescentes. Além disso, é fácil perceber que a seguinte relação entre as funções é verdadeira, para $x \geq 0$:

$$f\left(x\right) \leq s\left(x\right) \leq g\left(x\right) \tag{2}$$

A Figura (1) mostra de maneira clara a corretude dessas relações de desigualdade.

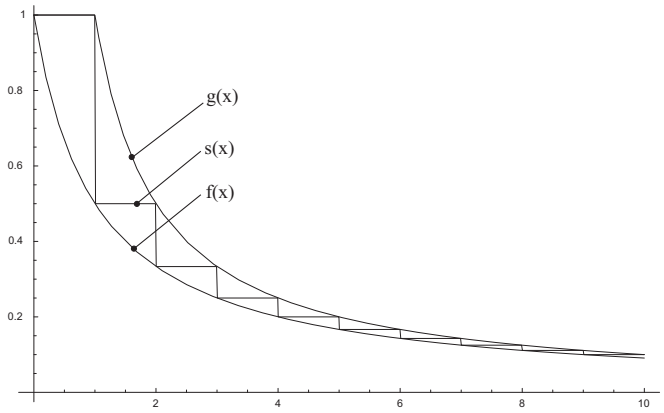

Figura 1: Gráfico das três funções, mostrando s(x) limitada por f(x) e g(x).

A inequação (2) prevalece após integrar cada termo em relação a um intervalo definido:

$$\int_0^n f(x)dx \leq \int_0^n s(x)dx \leq \int_0^n g(x)dx$$

Desenvolvendo a integral de $f(x)$ e $g(x)$:

$$\int_0^n \frac{1}{x+1}dx \leq \int_0^n s(x)dx \leq \int_0^1 1 + \int_1^n \frac{1}{x}dx$$

$$\ln(x+1)\Big|_0^n \leq \int_0^n s(x)dx \leq 1 + \ln x\Big|_1^n$$

$$\ln(n+1) \leq \int_0^n s(x)dx \leq \ln n + 1 \tag{3}$$

Agora sobre a integral de $s(x)$: É simples perceber que, para qualquer i inteiro não-negativo,

$$\begin{aligned}\int_i^{i+1} s(x)dx &= \int_i^{i+1} \frac{1}{\lfloor x+1 \rfloor} dx \\ &= \int_i^{i+1} \frac{1}{i+1} dx \\ &= \frac{1}{i+1}\end{aligned}$$

Portanto, para qualquer n inteiro positivo, pode-se decompor a integral de $s(x)$ em um somatório de integrais, visto a seguir:

$$\begin{aligned}\int_0^n s(x)dx &= \sum_{i=0}^{n-1} \int_i^{i+1} s(x)dx \\ &= \sum_{i=0}^{n-1} \frac{1}{i+1} dx \\ &= \frac{1}{i} \qquad (4)\end{aligned}$$

que é o somatório em questão no enunciado. Substituindo 4 em 3, temos:

$$\boxed{\ln(n+1) \leq \textstyle\sum_{i=1}^{n} \frac{1}{i} \leq \ln n + 1}$$

2 Relações de divisão e conquista

Sejam a, b e c constantes não negativas. Prove que a solução para

$$\begin{cases} T(1) = c & n = 1 \\ T(n) = aT(n/b) + cn & n > 1 \end{cases}$$

para n uma potência de b é

$$T(n) = \begin{cases} O(n), & \text{se } a < b, \\ O(n \log n), & \text{se } a = b, \\ O(n^{\log_b a}, & \text{se } a > b. \end{cases}$$

Resposta: Em algoritmos de divisão e conquista o problema é dividido em subproblemas menores, cada subproblema é resolvido recursivamente, e um algoritmo de combinação é utilizado para resolver o problema original. Assumindo que existem a subproblemas, cada um de tamanho $\frac{1}{b}$ do problema original e que o algoritmo utilizado para combinar as soluções dos subproblemas executa em tempo cn^k, para algumas constantes a, b, c e k. O tempo de execução $T(n)$ do algoritmo então satisfaz a relação proposta em.

Por simplicidade iremos assumir que $n = b^m$, tal que $\frac{n}{b}$ é sempre inteiro (b é inteiro positivo).

Expandindo T(n) temos:

$$\begin{aligned} T(n) &= aT\left(\frac{n}{b}\right) + cn \\ aT\left(\frac{n}{b}\right) &= a^2T\left(\frac{n}{b^2}\right) + ac\frac{n}{b} \\ a^2T\left(\frac{n}{b^2}\right) &= a^2T\left(\frac{n}{b^3}\right) + a^2c\frac{n}{b^2} \\ \vdots \quad &\vdots \quad \vdots \\ a^{m-1}T\left(\frac{n}{b^{m-1}}\right) &= a^mT\left(\frac{n}{b^m}\right) + a^{m-1}c\frac{n}{b^{m-1}} \end{aligned}$$

$$\begin{aligned} T(n) &= a^mc + \left(a^0cb^m + a^1cb^{m-1} + \ldots + a^{m-1}cb^1\right) \\ &= a^mcb^0 + a^{m-1}cb^1 + \ldots + a^1cb^{m-1} + a^0cb^m \\ &= c\sum_{i=0}^{m} a^{m-i}b^i \\ &= c\sum_{i=0}^{m} a^m\frac{b^i}{a^i} \end{aligned}$$

Enfim,

$$T(n) = a^m c \sum_{i=0}^{m} \frac{b}{a}^i \quad (5)$$

Uma vez que a equação 5 trata-se de uma série geométrica existem três casos a serem analisados:

(a) $\frac{b^k}{a} > 1$

(b) $\frac{b^k}{a} = 1$

(c) $\frac{b^k}{a} < 1$

A seguir iremos analisar cada caso separadamente:

Caso 1: $a > b^k$

Neste caso o fator da série geométrica é menor que 1, logo a série converge para uma constante, mesmo se m tender para infinito. Então, $T(n) = O(a^m)$. Dado que $m = log_b n$, temos $a^m = a^{log_b n} = n^{log_b a}$. Enfim:

$$\boxed{T(n) = O\left(n^{log_b a}\right)}$$

Caso 2: $a = b^k$

Neste caso o fator da série geométrica é igual a 1, então $T(n) = O(a^m m)$. Uma vez que $a = b^k$ implica que $log_b a = k$ and $m = O(logn)$. Enfim:

$$\boxed{T(n) = O\left(n^k logn\right)}$$

Caso 3: $a < b^k$

Neste caso, o fator da série geométrica é maior que 1. Utilizando a equação 14 e denotando $\frac{b^k}{a}$ por F (F é uma constante). Dado que o primeiro elemento da série é a^m, temos:

$$T(n) = a^m \frac{F^{m+1}-1}{F-1} = \mathrm{O}(a^m F^m) = O\left(\left(b^k\right)^m\right) = O\left((b^m)^k\right) = O\left(n^k\right)$$

$$\boxed{T(n) = O\left(n^k\right)}$$

3 Avaliação das relações de recorrência

(a) $\begin{cases} T(n) = 2T(n/4) + n, \text{ para } n > 1 \\ T(1) = 27 \end{cases}$

Resposta: Expandindo $T(n)$ e a seguir somando-se ambos os lados das igualdades, temos:

$$\begin{array}{ccc} T(n) & = & 2T\left(\frac{n}{4}\right) + n \\ 2T(\frac{n}{4}) & = & 2^2T\left(\frac{n}{4^2}\right) + \frac{2n}{4} \\ 2^2T\left(\frac{n}{4^2}\right) & = & 2^3T\left(\frac{n}{4^3}\right) + \frac{2^2n}{4^2} \\ 2^3T\left(\frac{n}{4^3}\right) & = & 2^4T\left(\frac{n}{4^4}\right) + \frac{2^3n}{4^3} \\ \vdots & \vdots & \vdots \\ 2^{k-1}T\left(\frac{n}{4^{k-1}}\right) & = & 2^kT\left(\frac{n}{4^k}\right) + \frac{2^{k-1}n}{4^{k-1}} \\ \hline T(n) & = & 2^kT\left(\frac{n}{4^k}\right) + \sum_{j=0}^{k-1}\left(\frac{2}{4}\right)^j n \end{array}$$

Podemos considerar n uma potência de 4 sem perda de generalidade. Logo, $n = 4^k, log_4 n = k$, então tem-se:

$$\begin{aligned} T(n) &= 2^k\ 27 + 4^k \sum_{j=0}^{k-1}\left(\frac{1}{2}\right)^j n \\ &= 2^k\ 27 + 4^k\left(\frac{1-\left(\frac{1}{2}\right)^k}{1-\left(\frac{1}{2}\right)}\right) \\ &= 2^k\ 27 + 2^{2k}\ 2\left(1 - 2^{-k}\right) \\ &= 27\ 2^k + 2^{2k}\ 2 - 2\ 2^k \\ &= 25\ 2^k + 2\ 2^{2k} \\ &= 25\ 2^{log_4 n} + 2\ 2^{log_4 n^2} \\ &= 25\ n^{log_4 2} + 2n^{log_4 4} \\ &= 25\ n^{0.5} + 2n \end{aligned}$$

Logo,

$$\boxed{T(n) = 25\ \sqrt[2]{n} + 2n}$$

(b) $\begin{cases} T(n) = 7T(n/2) + n^2, \text{ para } n > 1 \\ T(1) = 0 \end{cases}$

Resposta: Novamente expandindo $T(n)$ e somando-se ambos os lados das equações, temos:

$$\begin{array}{rcl}
T(n) & = & 7T(n) + n^2 \\
7T\left(\frac{n}{2}\right) & = & 7^2T\left(\frac{n}{2^2}\right) + 7\left(\frac{n}{2}\right)^2 \\
7^2T\left(\frac{n}{2^2}\right) & = & 7^3T\left(\frac{n}{2^3}\right) + 7^2\left(\frac{n}{2^2}\right)^2 \\
7^3T\left(\frac{n}{2^3}\right) & = & 7^4T\left(\frac{n}{2^4}\right) + 7^3\left(\frac{n}{2^3}\right)^2 \\
\vdots & \vdots & \vdots \\
7^{k-1}T\left(\frac{n}{2^{k-1}}\right) & = & 7^kT\left(\frac{n}{2^k}\right) + 7^{k-1}\left(\frac{n}{2^{k-1}}\right)^2
\end{array}$$

$$\begin{array}{rcl}
T(n) & = & 7^kT\left(\frac{n}{2^k}\right) + \sum_{j=0}^{k-1} 7^j\left(\frac{n}{2^j}\right)^2 \\
& = & \sum_{j=0}^{k-1} 7^j\left(\frac{2^{2k}}{2^{2j}}\right) \\
& = & 2^{2k}\sum_{j=0}^{k-1}\left(\frac{7}{4}\right)^j \\
& = & 2^{2k}\left(\frac{\left(\frac{7}{4}\right)^k - 1}{\left(\frac{7}{4}\right) - 1}\right) \\
& = & 2^{2k}\left(\frac{\left(\frac{7}{4}\right)^k - 1}{\frac{3}{4}}\right) \\
& = & \frac{4}{3}4^k\left(\left(\frac{7}{4}\right)^k - 1\right) \\
& = & \frac{4^{k+1}}{3}\ \frac{7^k}{4^k} - \frac{4}{3}4^k \\
& = & \frac{4}{3}7^k - \frac{4}{3}4^k \\
& = & \frac{4}{3}7^{log_2 n} - \frac{4}{3}4^{log_2 n} \\
& = & \frac{4}{3}n^{log_2 7} - \frac{4}{3}n^{log_2 4} \\
& = & \frac{4}{3}n^{log_2 7} - \frac{4}{3}n^2
\end{array}$$

Enfim,

$$\boxed{T(n) \approx \frac{4}{3}n^{2.81} - \frac{4}{3}n^2}$$

(c) $\begin{cases} T(n) = T(\sqrt{n}) + \log n, \text{ para } n \geq 1 \\ T(1) = 1 \end{cases}$

Resposta: Expandindo $T(n)$ tomando-se o $\lim_{k\to\infty}$, temos:

$$\begin{aligned}
T(n) &= T(\sqrt{n}) + logn \\
&= T\left(n^{\frac{1}{2}}\right) + logn \\
&= T\left(n^{\frac{1}{2^2}}\right) + logn^{\frac{1}{2}} + logn \\
&= T\left(n^{\frac{1}{2^3}}\right) + logn^{\frac{1}{2^2}} + logn^{\frac{1}{2}} + logn \\
\vdots \quad &\quad \vdots \quad \vdots \\
T(n) &= T\left(n^{\frac{1}{2^k}}\right) + logn^{\frac{1}{2^{k-1}}} + \ldots + logn^{\frac{1}{2}} + logn \\
\hline
T(n) &= 1 + \sum_{j=0}^{\infty} logn^{\frac{1}{2^j}} \\
&= 1 + \sum_{j=0}^{\infty} \frac{1}{2^j}\ logn \\
&= 1 + logn \sum_{j=0}^{\infty} \frac{1}{2^j} \\
&= 1 + logn \left(\frac{1}{1-\frac{1}{2}}\right)
\end{aligned}$$

Enfim,

$$\boxed{T(n) = 1 + 2\ logn}$$

(d) $\begin{cases} T(n) = T(n/9) + T(8n/9) + n, \quad \text{para } n \geq 1 \end{cases}$

Resposta: Iremos utilizar o método da árvore de recursão [3] para resolver a recorrência dada. Em uma árvore de recursividade, cada nodo representa o custo de um único subproblema no conjunto de chamadas recursivas. Logo, somamos os custos de cada nível da árvore para obtermos o custo por nível, e enfim, somamos os custos de todos os níveis para determinar o custo total de todos os níveis da recursividade.

A Figura 2 mostra a árvore de recursão para a recorrência $T(n) = T(n/9)+T(8n/9)+n$. Por conveniência assumimos que n é uma potência de 9. A Parte (a) da figura apresenta $T(n)$, que é expandida na Parte (b) em uma árvore de recorrência equivalente. O termo n representa o custo do primeiro nível da recursividade, e os dois termos a seguir representam

o custo de problemas de tamanho $\frac{n}{9}$ e $\frac{8n}{9}$. Na Parte (c) temos as folhas da Parte (b) expandidas. O processo de expansão de cada folha continua como determinado pela recorrência.

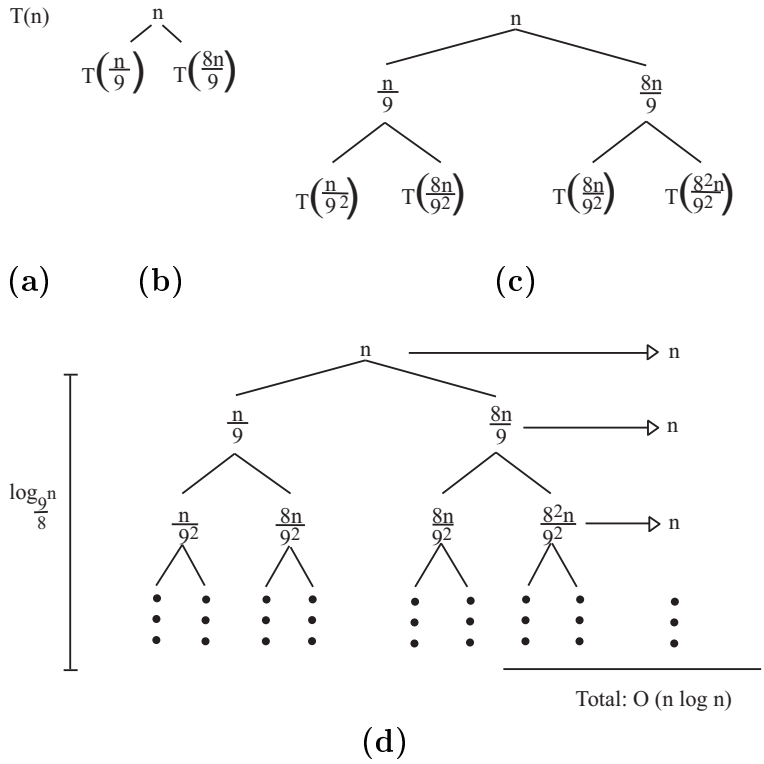

Figura 2: Construção da árvore de recursão da questão **3d**.

Uma vez que os tamanhos dos subproblemas diminuem a medida que expandimos a árvore a partir da raiz, eventualmente uma condição de borda é alcançada. A fim de determinar quando alcançaremos a borda, temos que o custo de um nodo no nível i é n. O caminho mais longo a partir da raiz até uma folha é $n\frac{8}{9}n(\,)^2n \ldots 1$. Dado que $(\frac{8}{9})n = 1$ se $k = \log_{\frac{9}{8}} n$, a altura da árvore é $\log_{\frac{9}{8}} n$.

Como mencionado acima temos que o custo da recursividade é dado pelo número de níveis multiplicado pelo custo de cada nível, ou $O(n \log_{\frac{9}{8}}) = O(n \log n)$.

Enfim:

$$\boxed{T\,(n) = O(n \log n)}$$

4 Limite inferior

Apresente o limite inferior para intercalar dois conjuntos de inteiros $A(1:m)$ e $B(1:n)$ onde os itens em A e os itens em B estão ordenados. Assuma que todos os números $m+n$ são distintos.

Esta prova é baseada na prova proposta em [6]. Uma vez que os elementos de $A(1:m)$ e de $B(1:n)$ estão ordenados e são distintos, temos:

$$A(1) < A(2) < \ldots < A(m)$$
$$B(1) < B(2) < \ldots < B(n)$$

Após o término do processo de ordenação é possível que os dois conjuntos estejam intercalados de várias maneiras. Para ser mais preciso temos que os $A\acute{}s$ podem aparecer entre os $B\acute{}s$ em $\binom{m+n}{m}$ formas distintas, preservando a ordenação dos $A\acute{}s$ e dos $B\acute{}s$. Logo, como demonstrado em **??**, podemos utilizar árvores de comparação como representação das comparações feitas pelo algoritmo de intercalação, logo existirão $\binom{m+n}{m}$ nodos externos e enfim, pelo menos

$$\left\lceil \log \binom{m+n}{m} \right\rceil$$

comparações são necessárias para qualquer algoritmo de intercalação. [6] propões um procedimento para intercalação que leva $m+n-1$ comparações. Tomando-se $M(m,n)$ como o número mínimo de comparações necessárias para intercalar m itens com n items, então temos a inequação:

$$\left\lceil \log \binom{m+n}{m} \right\rceil \leq M(m,n) \leq m+n-1$$

Quando m e n são iguais o limite inferior dado pelo modelo de árvore de comparações [9] é de fato muito baixo e o número de comparações para o algoritmo convencional [6] pode ser demonstrado ser ótimo.

Teorema 1 $M(m,n) = 2m-1$, *para* $m \geq 1$

Dado os possíveis algoritmos para intercalar os conjuntos $A(1) < A(2) < \ldots < A(m)$ e $B(1) < B(2) < \ldots < B(n)$ conhecemos um em particular que utiliza $2m-1$ comparações. Se pudermos mostrar que $M(m,n) \geq 2m-1$ então a prova está concluída. Consideremos qualquer algoritmo baseado em comparação para resolver o problema de intercalação e uma instância para a qual o resultado final é $B(1)A(1)B(2)A(2)\ldots B(m)A(m)$, isto é, onde os $B\acute{}s$ e os $A\acute{}s$ estão ordenados. Qualquer algoritmo de intercalação de realizar cada uma das $2m-1$ comparações $B(1):A(1), A(1):B(2), B(2):A(2), \ldots, B(m):A(m)$ enquanto intercala as entradas dadas. Para ver isto suponhamos que uma comparação do tipo $A(i):B(i)$ não é feita para algum i. Então o algoritmo não pode distiguir entre a ordenação anterior e a ordenação dada por

$B(1)A(1)\ldots A(i-1)A(i)B(i)B(i+1)\ldots B(m)A(m)$. Logo, é possível que o algoritmo não realize a intercalação dos $A´s$ e $B´s$ de maneira apropriada. Caso uma comparação do tipo $A(i) : B(i+1)$ não seja feita, então o algoritmo não será capaz de distinguir entre os casos $B(1)A(1)B(2)\ldots B(m)A(m)$ e $B(1)A(1)B(2)A(2)\ldots A(i-1)B(i)B(i+1)A(i)A(i+1)\ldots B(m)A(m)$. Enfim, qualquer algoritmo deve realizar todas as $2m-1$ comparações para produzir o resultado final.

5 Estudo do algoritmo árvore B ([12])

(a) Obtenha analiticamente o valor da altura h de uma árvore B de ordem m para o melhor caso e o pior caso.

Resposta:

Esta prova é baseada na prova dada por [1]. Consideremos a altura h da árvore B sendo igual ao número de páginas num caminho da árvore. Os valores de h variam no intervalo $[1 \ldots \infty]$.

Sejam P_{min} e P_{max} o número mínimo e o número máximo de páginas numa árvore B de ordem m.

Uma vez que, excetuando-se a raiz, as folhas têm pelo menos $m+1$ filhos, temos:

$$\begin{aligned} P_{min} &= 1 + 2((m+1)^0 + (m+1)^1 + (m+1)^2 + \ldots + (m+1)^{h-2}) \\ &= 1 + 2\sum_{j=0}^{h-2}(m+1)^j \\ &= 1 + \frac{2}{m}((m+1)^{h-1} - 1) \end{aligned}$$

Para determinarmos o número máximo de páginas numa árvore B de altura h, devemos supor que todas as páginas da árvore estejam completas, ou seja, todas as páginas contêm $2m$ chaves. Logo, cada página possuirá $2m+1$ filhos. Então:

$$\begin{aligned} P_{max} &= 1 + (2m+1)^1 + (2m+1)^2 + \ldots + (2m+1)^{h-1} \\ &= \sum_{j=0}^{h-1}(2m+1)^j \\ &= \frac{1}{2m}((2m+1)^h - 1) \end{aligned}$$

De acordo com os resultados anteriores temos que:

$$P_{min} = 1 + \frac{2}{m}((m+1)^{h-1} - 1) \tag{6}$$

$$P_{max} = \frac{1}{2m}((2m+1)^h - 1) \tag{7}$$

Dadas as equações 6 e 7, iremos derivar o resultado de h para um conjunto de N chaves. Sejam N_{min} e N_{max} o número mínimo e máximo de chaves em uma árvore B de altura h.

$$\begin{aligned} N_{min} &= 1 + m\frac{\left(2(m+)^{h-1}-1\right)}{m} &= 2(m+1)^{h-1} - 1 \\ N_{max} &= 2m\left((2m+1)^h - 1\right) &= (2m+1)^h - 1 \end{aligned}$$

Isolando-se h em ambas as equações temos:

$$\log_{N+1} \leq h \leq 1 + log_{m+1}\left(\frac{N+1}{2}\right)$$

(b) Obtenha empiricamente os resultados a seguir. Utilize valores distintos de $m = 1$, $m = 50$ e $m = 100$. Utilize arquivos de diferentes tamanhos com chaves geradas randomicamente para cada valor de m. Use o programa *Permut.c* para obter uma permutação randômica de n valores (Ziviani, 2004, pag. 427). Repita cada experimento algumas vezes e obtenha a média para cada medida de complexidade.

- A altura esperada h de uma árvore B de ordem m randômica com n chaves, para os valores de m indicados acima. Assuma que a altura esperada é aproximadamente $\log_x n + 1$. Determine empiricamento o valor de x.
- A probabilidade de que a página segura mais profunda esteja no primeiro nível da árvore.
- O valor da taxa de utilização de memória.

(c) Procure trabalhos relacionados mais recentes na literatura com resultados analíticos e descreva suscintamente esses trabalhos. Como os resultados experimentais que você obteve comparam com os resultados analíticos de Ziviani (2004) e/ou encontrados na literatura?

Resposta dos itens (b) e (c):

Foram realizados experimentos a fim de estudar características do algoritmo árvore B proposto em [1].

A seguir iremos descrever e discutir cada experimento realizado:

Experimento 1

Conforme discutido em [12], a altura de uma árvore B não é conhecida analiticamente, uma vez que não existem trabalhos conhecidos na literatura apresentando tal resultado. Porém, a partir do cálculo analítico do

número esperado de páginas para os quatro primeiros níveis, contados a partir das páginas folha em direção à páginas raiz de uma **árvore 2-3** (ou árvore B de ordem $m = 1$), em [4] foi proposta a seguinte conjectura: a altura esperada de uma árvore 2-3 **randômica** com N chaves é dada por:

$$\overline{h}(N) \approx \log_{\frac{7}{3}}(N+1) \tag{8}$$

Conforme mencionado acima, neste experimento iremos determinar a base x do logaritmo em:

$$h(N) = \log_x(N+1)$$

Com o intuito de determinar o valor de x determinamos a altura média de árvores randômicas, isto é, executamos o programa *Permut.c* [12] para obter uma permutação randômica de N valores. Estes valores foram inseridos numa estrutura de árvore B e a altura determinada. A média das alturas foi calculada sobre 33 execuções, ou seja, 33 sequências aleatórias de N valores. Os resultados estão apresentados nas Tabelas 3, 4 e 5, para valores de $M = 1$, $M = 50$ e $M = 100$, respectivamente. Os valores de $< Altura >$, $<\text{Taxa}_{com\_ap} >$ e $<\text{Taxa}_{sem\_ap} >$ referem-se a valores médios, com relação a 33 execuções.

Expandindo $h(N)$ temos:

$$\begin{aligned} h(N) &= \log_x(N+1) \\ &= \tfrac{\log(N+1)}{\log x} \\ &= \tfrac{1}{\log x}(\log(N+1)) \end{aligned}$$

Para determinar $\frac{1}{\log x}$ iremos utilizar um modelo de regressao linear simples. Sendo $h(N)$ dado por:

$$h(N) = a(\log(N+1))$$

Temos:

$$a = \frac{1}{\log x} \quad \text{e} \quad x = 10^{\frac{1}{a}}$$

Na Tabela 2c apresentamos os resultados de a e x encontrados para diferentes valores de M. Conforme discussão apresentada anteriormente 8, o valor de x para $M = 1$ eh aproximadamente $\frac{7}{3}(\approx 2.33)$. Podemos ver que nosso resultado para este valor de M apresentado na Tabela 2c é muito próximo ao valor dado pela conjectura [4]. Além disso, os valores do coeficiente de determinação R^2 são muito próximos de 1. Segundo [7] quanto maior o coeficiente de determinação melhor é a regressão. Caso um modelo de regressão seja perfeito o coeficiente de determinação é 1. Por exemplo, na Tabela 2c, para $M = 50$ temos o coeficiente de determinação k^2 aproximadamente igual a 0.98, o que significa que a regressão

M	a	x	$x_{esperado}$	R^2
1	2.747	2.312	$\ln 2 \approx 2.333$	0.9998
50	0.651	34.36	–	0.9879
100	0.565	58.87	–	0.9845

Tabela 1: Valores do modelo de regressão linear.

explica 98% da variação do valor de a.

De acordo com o Gráfico 3 as funções utilizando a base do logaritmo conjecturada em [4] e utilizando-se a base do logaritmo obtida pelo nosso modelo de regressão linear simples, não diferem significativamente para os valores de N calculados.

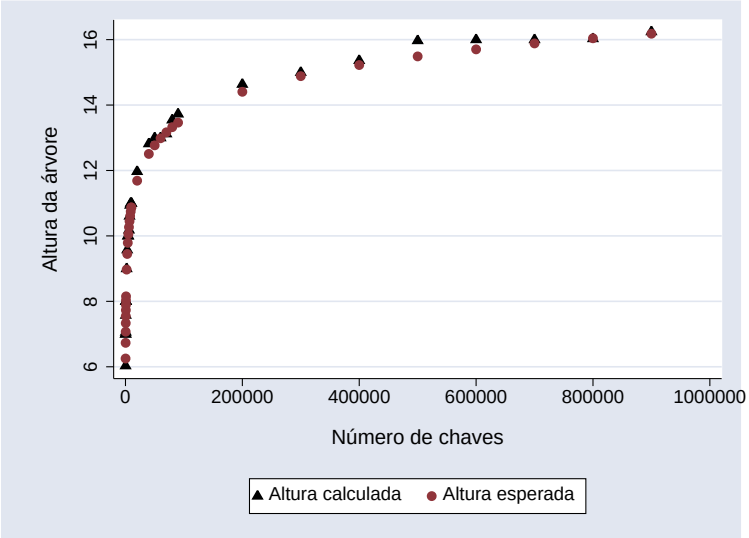

Figura 3: Altura da **árvore 2-3**.

No Gráfico 4 analisamos o comportamento da altura h de uma árvore em relação ao número de chaves para diferentes valores de x encontrados, ou seja, para diferentes valores de M. Notamos que mesmo para valores relativamente pequenos de N os valores da altura média das árvores de ordem $M = 1$ são superiores a altura média de árvores de ordem maior, isto é, $M = 50$ e $M = 100$

Experimento 2

Neste experimento, assim como no anterior, para cada valor de M analisamos 33 árvores randômicas. Segundo [12] uma página é chamada segura quando se sabe que não existe possibilidade de modificações na estrutura da árvore, como consequência de uma operação de inserção ou de remoção de chaves naquela árvore. Cabe ressaltar que neste trabalho

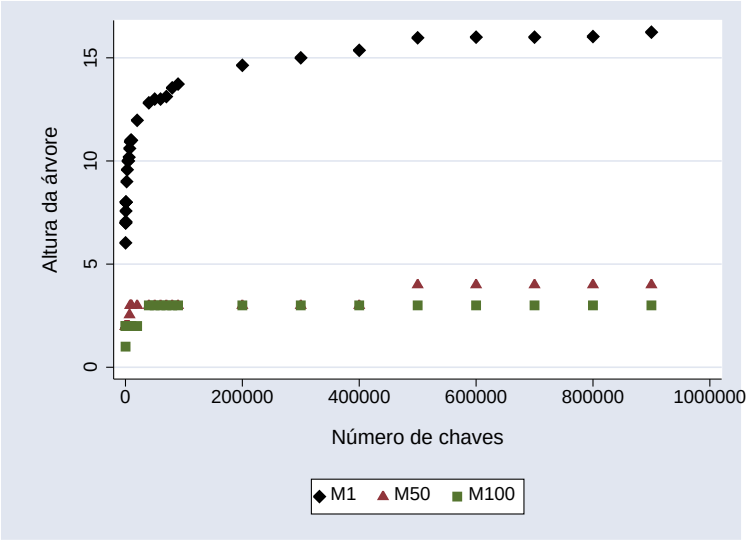

Figura 4: Altura da árvore B.

consideramos apenas as operações de inserção para a verificar se uma página é segura. Uma página segura é a mais profunda de um caminho de inserção se não existir outra página segura abaixo dela. Para determinar a página segura mais profunda inserimos $N-1$ chaves permutadas na árvore, no momento de inserção da N-ésima chave caminhamos na estrutura da árvore até encontrarmos a página na qual a chave N deve ser inserida, caso o número de chaves nesta página seja menor que $2M$, a página é considerada segura para inserção (uma vez que estrutura da árvore não será alterada), caso contrário, a página não é considerada segura, obviamente.

Segundo [12], para uma **árvore 2-3** (árvore B de ordem $m = 1$), temos que a probabilidade Pr_1 de que a página segura mais profunda esteja no primeiro nível é dada por:

$$Pr_1 = \frac{4}{7} \approx 0.57 \tag{9}$$

De acordo com a Tabela 3, encontramos o valor de Pr_1 dado por:

$$0.42 \leq Pr_1 \leq 0.69$$

Sendo assim, os valores encontrados estão próximos ao valor esperado.

Já para uma árvore B de ordem M qualquer, temos que a probabilidade Pr_M de que a página segura mais profunda esteja no primeiro nível é:

$$Pr_{Mesp} = 1 - \frac{1}{(2\ln 2)M} + O(m^{-2}) \tag{10}$$

Podemos observar o mesmo comportamento na Tabela 4 e na Tabela 5, para valores de $M = 50$ e $M = 100$, respectivamente. A Tabela 2 apresenta os intervalos de valores encontrados para Pr_M e os valores esperados

dessa grandeza. Os valores da probabilidade Pr_{Mesp} esperada são aproximados. Então podemos concluir que a medida que a ordem da árvore aumenta o valor da probabilidade da página segura mais profunda estar no primeiro nível aumenta.

M	Pr_M	Pr_{Mesp}
1	$[0.42, 0.69]$	0.57
50	$[0.94, 1.00]$	0.98
100	$[0.94, 1.00]$	0.99

Tabela 2: Probabilidade da página segura mais profunda estar no primeiro nível.

Experimento 3

Este experimento consiste em medir a taxa de utilização de memória para os diferentes valores de M e diferentes conjuntos de chaves randômicas n. Consideramos duas formas diferentes para o cálculo da taxa de utilização de memória. Dada a estrutura de dados árvore B proposta por [1] e disponível em [12]. A primeira suposição consiste em considerar que cada página da árvore é constituída por um número de chaves e por um número de apontadores, valores estes que dependem da ordem M da árvore B considerada. Optamos então por realizar dois experimentos: no primeiro utilizamos ambos elementos da página a fim de calcular a taxa de utilização da memória, já no segundo experimento, optamos por desconsiderar os apontadores da página. A seguir, discutimos cada experimento.

i. **Considerando** apontadores

A taxa de utilização de memória é calculada como:

$$\text{Taxa} = \frac{\text{Quantidade utilizada}}{\text{Quantidade alocada}} \tag{11}$$

No caso em que consideramos os apontadores temos:

$$\text{Quantidade utilizada} = P + n$$

onde, P é o número de páginas da árvore e n o número de chaves.

$$\begin{aligned} \text{Quantidade alocada} &= (P \times (a + n)) + 1 \\ &= (P \times (2M + 1 + 2M)) + 1 \\ &= (P \times (4M + 1)) + 1 \end{aligned}$$

onde, a é o número de apontadores e M a ordem da árvore.

Enfim,

$$Taxa_{com\_ap} = \frac{P + n}{P \times (4M + 1) + 1} \tag{12}$$

Segundo [12], a utilização de memória é cerca de $\ln 2$ para o algoritmo original proposto por [1]. Porém, de acordo com as Tabelas 3, 4 e 5, os resultados de $Taxa_{com\_ap}$, em geral, são inferiores. O Gráfico 5 permite visualizarmos o comportamento da taxa de utilização de memória para valores diferentes de N, para cada valor de M pedido.

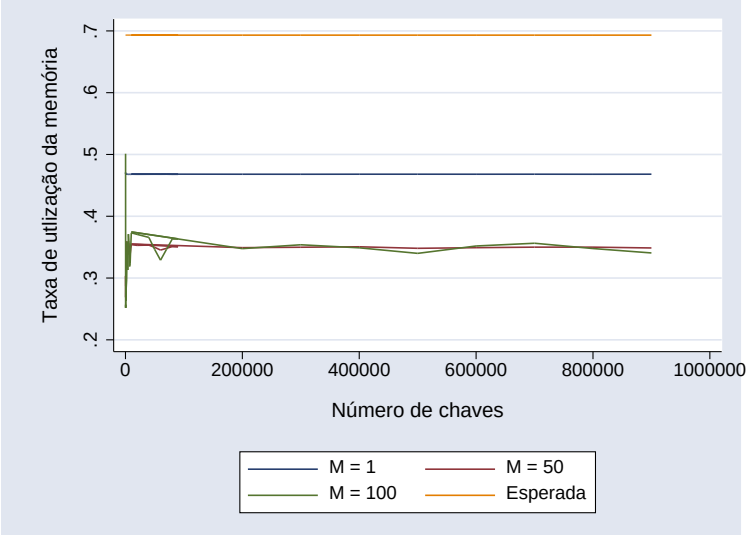

Figura 5: Taxa de utilização de memória considerando apontadores.

Devido ao fato de os valores de taxa não convergirem para o valor esperado, mesmo para grandes valores de N e M, motivou o segundo experimento, no qual desconsideramos os apontadores. Nossa hipótese é o que o resultado relatado possa estar desconsiderando os apontadores. A seguir, segue a discussão deste novo experimento.

ii. **Desconsiderando** apontadores

Dada a equação 11, temos:

$$\text{Quantidade utilizada} = n$$

onde, n o número de chaves.

$$\text{Quantidade alocada} = (P \times 2M)$$

onde, P é o número de páginas e M a ordem da árvore.

Enfim,

$$Taxa_{sem\_ap} = \left(\frac{n}{P \times 2M}\right) + 1 \tag{13}$$

Conforme vemos nas Tabelas 3, 4 e 5 é possível que nossa hipótese esteja correta, uma vez que os valores apresentados estão próximos a $\ln 2 \approx 0.69$.

Além disso, novamente analisando o gráfico da taxa de utilização de memória desconsiderando apontadoresr̃efgraf:taxa:sem:ap. Podemos notar que o valor da taxa de utilização de memória para todos os valores de M estão mais próximos do esperado em relação ao gráfico anterior 5. Além disso, vemos que quanto maior o valor de M melhor é a taxa de utilização de memória. Por fim, gostaríamos de chamar a atenção para o que parece ser um dado incorreto no gráfico, para um valor baixo de N, temos que a taxa de utilização da memória foi de 100%, este ponto refere-se a uma árvore de ordem $M = 100$ com 200 chaves inseridas.

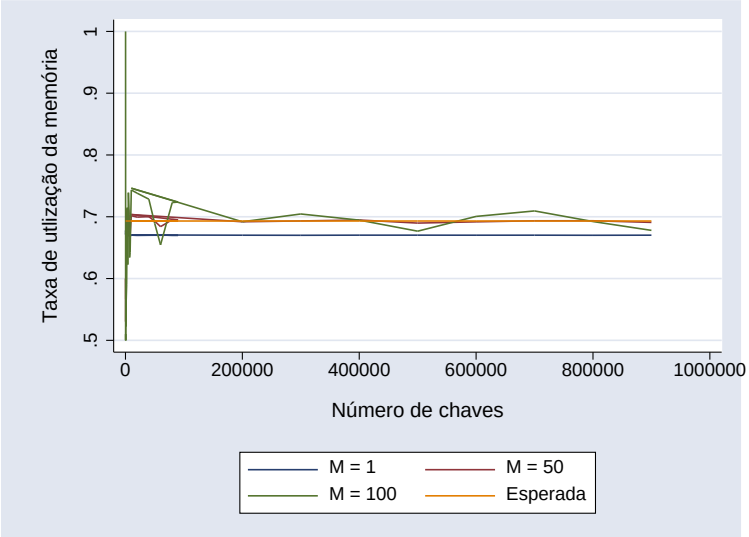

Figura 6: Taxa de utilização de memória desconsiderando apontadores.

Todo o estudo sobre árvores B começões com o artigo [1], no qual os autores propõe a estrutura de dados árvore B e descrevem suas principais características. Em [2] os autores citam o enorme sucesso da estrutura de dados árvore B para organização de arquivos como um todo. Além disso os autores discutem variações da estrutura proposta, especialmente as árvores B+. Em [5] os autores propõe um estudo sobre como árvores B se comportam em conjuntos não bem distribuído de chaves (*skewed distribution*). Os autores propõem variar o tamanho dos regitros da árvore durante a inserção de chaves. Somente a inserção é analisada neste trabalho. Um dos resultados

analíticos do trabalho é:

$$I(b, g, h, K) = 2 + \sum_{i=1}^{\infty} kbH^i g^{i-1} p(bH^i g^{i-1})$$

onde:

b = tamanho inicial do bloco
g = fator de crescimento dos registros durante em uma cópia
h = fator de crescimento dos registros com ausência de cópia
K = número máximo de registros numa sequência

Segundo [10] árvores B que possuem a propriedade *lazy parent split*(lps) são variantes de árvores B, para as quais a divisão do nodo pai é postergarda até um futuro acesso a um nodo mais profundo. Esta estratégia permite que o número de divisões durante uma inserção seja diminuído, assim como o número de *locks*. Logo em uma situação de concorrência serão observadas melhorias significativas de performance. Dentre outros resultados significativos, os autores provam que o número esperado de chaves numa árvore 23 com lps é

$$\overline{f}(N) = (N+1)\frac{514.927}{615.307} + O(N^{-6.5})$$

Em [11] os autores provam que o número médio de nodos (em árvores somente com inserção) é aproximadamente 69%. Já em [8] os autores discutem a taxa de utilização tanto em inserções como em modificações, sendo que uma modificação é uma retirada seguida de inserção. O trabalho proposto em [8] é uma generalização do trabalho proposto em [11].

M = 1				
Numero de chaves	<Altura>	Probabilidade	<Taxa$_{com_ap}$>	<Taxa$_{sem_ap}$ >
200	6.030303	0.636364	0.471151	0.677878
300	7.000000	0.606061	0.468638	0.671596
400	7.000000	0.484848	0.469072	0.672679
500	7.060606	0.696970	0.468442	0.671106
600	7.575758	0.545455	0.468132	0.670329
700	8.000000	0.424242	0.466931	0.667328
800	8.000000	0.484848	0.467673	0.669183
900	8.000000	0.636364	0.468182	0.670455
1000	8.000000	0.545455	0.468290	0.670724
2000	9.000000	0.575758	0.468644	0.671611
3000	9.575758	0.454545	0.468231	0.670577
4000	10.000000	0.575758	0.467641	0.669102
5000	10.000000	0.424242	0.467723	0.669308
6000	10.181818	0.636364	0.468119	0.670298
7000	10.606061	0.454545	0.467947	0.669867
8000	10.939394	0.666667	0.467846	0.669616
9000	11.000000	0.696970	0.467691	0.669227
10000	11.000000	0.454545	0.468103	0.670257
20000	11.969697	0.484848	0.467947	0.669869
40000	12.818182	0.484848	0.468028	0.670069
50000	13.000000	0.606061	0.468055	0.670138
60000	13.000000	0.454545	0.468017	0.670042
70000	13.121212	0.606061	0.468095	0.670237
80000	13.545455	0.424242	0.467977	0.669942
90000	13.727273	0.575758	0.468008	0.670021
10000	11.000000	0.484848	0.468246	0.670615
200000	14.636364	0.575758	0.468072	0.670179
300000	15.000000	0.606061	0.468017	0.670042
400000	15.363636	0.606061	0.468107	0.670267
500000	15.969697	0.545455	0.468104	0.670260
600000	16.000000	0.545455	0.468070	0.670175
700000	16.000000	0.515152	0.468090	0.670225
800000	16.030303	0.515152	0.468028	0.670069
900000	16.242424	0.636364	0.468090	0.670225
1000000	16.757576	0.484848	0.468072	0.670179

Tabela 3: Resultados para $M = 1$.

M = 50				
Numero de chaves	<Altura>	Probabilidade	<Taxa$_{com_ap}$>	<Taxa$_{sem_ap}$>
200	2.000000	0.939394	0.268807	0.530303
300	2.000000	1.000000	0.303483	0.600000
400	2.000000	1.000000	0.300396	0.593795
500	2.000000	0.969697	0.286905	0.566679
600	2.000000	1.000000	0.331625	0.656566
700	2.000000	0.969697	0.359736	0.713070
800	2.000000	0.939394	0.327493	0.648261
900	2.000000	0.939394	0.306413	0.605890
1000	2.000000	0.969697	0.311856	0.616830
2000	2.000000	1.000000	0.327300	0.647873
3000	2.000000	1.000000	0.357849	0.709277
4000	2.000000	1.000000	0.337542	0.668460
5000	2.000000	0.969697	0.359356	0.712306
6000	2.000000	1.000000	0.355176	0.703905
7000	2.545455	1.000000	0.339815	0.673029
8000	3.000000	1.000000	0.338438	0.670260
9000	3.000000	0.969697	0.347429	0.688333
10000	3.000000	0.969697	0.354086	0.701712
20000	3.000000	0.939394	0.352948	0.699426
40000	3.000000	1.000000	0.353348	0.700229
50000	3.000000	1.000000	0.349739	0.692974
60000	3.000000	1.000000	0.345313	0.684079
70000	3.000000	1.000000	0.348035	0.689550
80000	3.000000	1.000000	0.351022	0.695555
90000	3.000000	0.939394	0.350729	0.694965
10000	3.000000	0.939394	0.355051	0.703652
200000	3.000000	0.969697	0.349208	0.691907
300000	3.000000	1.000000	0.349881	0.693260
400000	3.000000	1.000000	0.350621	0.694748
500000	4.000000	1.000000	0.348159	0.689800
600000	4.000000	0.969697	0.349167	0.691825
700000	4.000000	0.939394	0.350021	0.693541
800000	4.000000	1.000000	0.350035	0.693570
900000	4.000000	1.000000	0.348714	0.690914
1000000	4.000000	1.000000	0.347854	0.689186

Tabela 4: Resultados para $M = 50$.

M = 100				
Numero de chaves	<Altura>	Probabilidade	<Taxa$_{com_ap}$>	<Taxa$_{sem_ap}$>
200	1.000000	1.000000	0.501247	1.000000
300	2.000000	1.000000	0.251870	0.500000
400	2.000000	0.969697	0.256908	0.510101
500	2.000000	1.000000	0.251870	0.500000
600	2.000000	1.000000	0.301746	0.600000
700	2.000000	1.000000	0.344568	0.685859
800	2.000000	0.969697	0.301530	0.599567
900	2.000000	1.000000	0.263071	0.522457
1000	2.000000	1.000000	0.279579	0.555556
2000	2.000000	1.000000	0.296989	0.590463
3000	2.000000	0.969697	0.358987	0.714770
4000	2.000000	1.000000	0.313047	0.622660
5000	2.000000	1.000000	0.371090	0.739036
6000	2.000000	1.000000	0.357816	0.712421
7000	2.000000	0.969697	0.318767	0.634127
8000	2.000000	1.000000	0.322983	0.642581
9000	2.000000	1.000000	0.350330	0.697412
10000	2.000000	1.000000	0.373239	0.743344
20000	2.000000	1.000000	0.370670	0.738194
40000	3.000000	1.000000	0.365824	0.728478
50000	3.000000	0.969697	0.346294	0.689320
60000	3.000000	0.969697	0.328960	0.654564
70000	3.000000	1.000000	0.346134	0.688999
80000	3.000000	1.000000	0.363115	0.723045
90000	3.000000	1.000000	0.363499	0.723816
10000	2.000000	1.000000	0.374690	0.746254
200000	3.000000	1.000000	0.347629	0.691996
300000	3.000000	1.000000	0.353877	0.704523
400000	3.000000	1.000000	0.348939	0.694622
500000	3.000000	1.000000	0.339992	0.676684
600000	3.000000	1.000000	0.351920	0.700600
700000	3.000000	0.969697	0.356336	0.709455
800000	3.000000	0.969697	0.347775	0.692289
900000	3.000000	1.000000	0.340645	0.677993
1000000	3.000000	0.939394	0.340425	0.677552

Tabela 5: Resultados para $M = 1000$.

Referências

[1] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Inf.*, 1:173–189, 1972.

[2] Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2001.

[4] Bernhard Eisenbarth, Nivio Ziviani, Gaston H. Gonnet, Kurt Mehlhorn, and Derick Wood. The theory of fringe analysis and its application to 2-3 trees and b-trees. *Information and Control*, 55(1-3):125–174, 1982.

[5] Christos Faloutsos and H. V. Jagadish. On b-tree indices for skewed distributions. In *VLDB*, pages 363–374, 1992.

[6] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[7] Raj Jain. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.

[8] Theodore Johnson and Dennis Shasha. Utilization of b-trees with inserts, deletes and modifies. In *PODS*, pages 235–246, 1989.

[9] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Publishing Company, 1973.

[10] Antigoni Manousaka and Yannis Manolopoulos. Fringe analysis of 2-3 trees with lazy parent split.

[11] Andrew Yao. On random 2-3 trees. *Acta Informática*, pages 159–170, 1978.

[12] Nivio Ziviani. *Projeto de Algoritmos com implementação em Pascal e C*, volume Único. Thomson, 2004.

Anexo 1 – Somatórios úteis

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1} - 1}{a - 1}, \quad \text{para } a \neq 1 \tag{14}$$

$$\sum_{i=0}^{k} a^i = \frac{1 - a^{k+1}}{1 - a}, \quad \text{para } -1 < a < 1 \tag{15}$$

Anexo 2 – Código fonte do programa utilizado

(a) **Script para geração das tabelas de resultados**

```
#!/bin/bash

echo
"############################################################
##############"
echo "#        SCRIPT DE TESTES DO ALGORITMO ARVORE B
(ZIVIANI,2004)        #" echo
"############################################################
##############"

echo "1 - Gerando arquivos de chaves aleatorias:" echo

data_out_ms="data_out_ms.txt"; command_rm_old_data="/bin/rm
"$data_out_ms; eval $command_rm_old_data;

set_numeric="LC_NUMERIC=C"; eval $set_numeric;
set_numeric="export
LC_NUMERIC"; eval $set_numeric;

out_m_1="output_tex_file_m1" command_reset="/bin/rm "$out_m_1".tex"
eval $command_reset #out_m_50="output_tex_file_m50"
#command_reset="/bin/rm "$out_m_50".tex" #eval $command_reset
#out_m_100="output_tex_file_m100" #command_reset="/bin/rm
"$out_m_100".tex" #eval $command_reset

m_order=1; each_n=33; i=1; echo "M: " $m_order " Execucoes: "
$each_n; echo "M: "$m_order >> $data_out_ms;

#echo "<Numero de chaves> <Altura> <Probalidade> <Taxa>" #v_keys="10
20 30 40 50 60 70 80 90 100"; v_keys="200 300 400 500 600 700 800
900 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000 20000 40000
50000 60000 70000 80000 90000 10000 200000 300000 400000 500000
```

```
600000 700000 800000 900000 1000000"; #v_keys="200 400 600 800 1000
2000 4000 8000 10000 20000 30000 40000 50000 100000 1000000";
#v_keys="200 400 600";

echo "\documentclass[12pt]{article}" >> $out_m_1".tex" echo
"\begin{document}" >> $out_m_1".tex" echo "\begin{center}" >>
$out_m_1".tex" echo "\begin{tabular}{|c|c|c|c|c|} \hline" >>
$out_m_1".tex" echo "\multicolumn{5}{|c|}{M = 1}\\\\ \hline \hline"
>> $out_m_1".tex" echo "Numero de chaves & Altura & Probabilidade &
Taxa\$_{com\_ap}\$ & Taxa\$_{sem\_ap}\$\\\\ \hline" >>
$out_m_1".tex"

for key in $v_keys ; do
    echo "Number of keys: "$key;
    i=1;
    temp_file="tmp_m"$m_order"_n"$key"_file.txt";
    eval "touch "$temp_file;
    while [ $i -le $each_n ]; do
        result_keys=`./rand_gen $key 1`;
        temp_file_keys="temp_file_keys.txt";
        echo $result_keys > $temp_file_keys
        stats=`./b_tree $m_order $key $temp_file_keys`;
        #echo $stats;
        eval "/bin/rm "$temp_file_keys;
        echo $stats >> $temp_file;
        i=$(( $i + 1 ));
    done
    command2="awk -f gen_stats.awk "$temp_file;
    result=`eval $command2`;
    echo $result >> $data_out_ms;
    for i in 1 2 3 4 5; do
        v_feat[$i]=`echo "$result" | cut -d" " \-f$i`;
        #echo ${v_feat[$i]};
    done
    line=`echo ${v_feat[1]}" & "${v_feat[2]}" & "${v_feat[3]}"
     & "${v_feat[4]}" & "${v_feat[5]}`;
    echo $line"\\\\ \hline" >>  $out_m_1".tex";
    command1="/bin/rm "$temp_file;
    eval $command1;
done

echo "\end{tabular}" >> $out_m_1".tex"; echo "\end{center}" >>
$out_m_1".tex"; echo #echo "\end{document}" >> $out_m_1".tex";
#command_latex="latex "$out_m_1; #eval $command_latex;
#command_latex="dvips "$out_m_1".dvi"; #eval $command_latex;
#command_latex="gv "$out_m_1".ps"; #eval $command_latex;
```

```
m_order=50; #each_n=33; i=1; echo "M: " $m_order " Execucoes: "
$each_n; echo "M: "$m_order >> $data_out_ms; #echo "<Numero de
chaves> <Altura> <Probalidade> <Taxa>" #v_keys="10 20 30 40 50 60 70
80 90 100"; #v_keys="200 300 400 500 600 700 800 900 1000 2000 3000
4000 5000 6000 7000 8000 9000 10000 20000 40000 50000 60000 70000
80000 90000 100000 1000000"; #v_keys="200 300 400 500";

#echo "\documentclass[12pt]{article}" >> $out_m_1".tex" #echo
"\begin{document}" >> $out_m_1".tex" echo "\begin{center}" >>
$out_m_1".tex" echo "\begin{tabular}{|c|c|c|c|c|} \hline" >>
$out_m_1".tex" echo "\multicolumn{5}{|c|}{M = "$m_order"}\\\\ \hline
\hline" >> $out_m_1".tex" echo "Numero de chaves & Altura &
Probabilidade & Taxa\$_{com\_ap}\$ & Taxa\$_{sem\_ap}\$ \\\\ \hline"
>> $out_m_1".tex"

for key in $v_keys ; do
    echo "Number of keys: "$key;
    i=1;
    temp_file="tmp_m"$m_order"_n"$key"_file.txt";
    eval "touch "$temp_file;
    while [ $i -le $each_n ]; do
        result_keys=`./rand_gen $key 1`;
        temp_file_keys="temp_file_keys.txt";
        echo $result_keys > $temp_file_keys
        stats=`./b_tree $m_order $key $temp_file_keys`;
        #echo $stats;
        eval "/bin/rm "$temp_file_keys;
        echo $stats >> $temp_file;
        i=$(( $i + 1 ));
    done
    command2="awk -f gen_stats.awk "$temp_file;
    result=`eval $command2`;
    echo $result >> $data_out_ms;
    for i in 1 2 3 4 5; do
        v_feat[$i]=`echo "$result" | cut -d" " \-f$i`;
        #echo ${v_feat[$i]};
    done
    line=`echo ${v_feat[1]}" & "${v_feat[2]}" & "${v_feat[3]}"
     & "${v_feat[4]}" & "${v_feat[5]}`;
    echo $line"\\\\ \hline" >>  $out_m_1".tex";
    command1="/bin/rm "$temp_file;
    eval $command1;
done

echo "\end{tabular}" >> $out_m_1".tex"; echo "\end{center}" >>
$out_m_1".tex"; echo #echo "\end{document}" >> $out_m_1".tex";
```

```
#command_latex="latex "$out_m_1; #eval $command_latex;
#command_latex="dvips "$out_m_1".dvi"; #eval $command_latex;
#command_latex="gv "$out_m_1".ps"; #eval $command_latex;



m_order=100; #each_n=33; i=1; echo "M: " $m_order " Execucoes: "
$each_n; echo "M: "$m_order >> $data_out_ms; #echo "<Numero de
chaves> <Altura> <Probalidade> <Taxa>" #v_keys="10 20 30 40 50 60 70
80 90 100"; #v_keys="200 300 400 500 600 700 800 900 1000 2000 3000
4000 5000 6000 7000 8000 9000 10000 20000 40000 50000 60000 70000
80000 90000 100000 1000000"; #v_keys="200 300 400 500 1000";

#echo "\documentclass[12pt]{article}" >> $out_m_1".tex" #echo
"\begin{document}" >> $out_m_1".tex" echo "\begin{center}" >>
$out_m_1".tex" echo "\begin{tabular}{|c|c|c|c|c|} \hline" >>
$out_m_1".tex" echo "\multicolumn{5}{|c|}{M = "$m_order"}\\\\ \hline
\hline" >> $out_m_1".tex" echo "Numero de chaves & Altura &
Probabilidade & Taxa\$_{com\_ap}\$ & Taxa\$_{sem\_ap}\$ \\\\ \hline"
>> $out_m_1".tex"

for key in $v_keys ; do
    echo "Number of keys: "$key;
    i=1;
    temp_file="tmp_m"$m_order"_n"$key"_file.txt";
    eval "touch "$temp_file;
    while [ $i -le $each_n ]; do
        result_keys=`./rand_gen $key 1`;
        temp_file_keys="temp_file_keys.txt";
        echo $result_keys > $temp_file_keys
        stats=`./b_tree $m_order $key $temp_file_keys`;
        #echo $stats;
        eval "/bin/rm "$temp_file_keys;
        echo $stats >> $temp_file;
        i=$(( $i + 1 ));
    done
    command2="awk -f gen_stats.awk "$temp_file;
    result=`eval $command2`;
    echo $result >> $data_out_ms;
    for i in 1 2 3 4 5; do
        v_feat[$i]=`echo "$result" | cut -d" " \-f$i`;
        #echo ${v_feat[$i]};
    done
    line=`echo ${v_feat[1]}" & "${v_feat[2]}" & "
    ${v_feat[3]}" & "${v_feat[4]}" & "${v_feat[5]}`;
    echo $line"\\\\ \hline" >>  $out_m_1".tex";
    command1="/bin/rm "$temp_file;
```

```
    eval $command1;
done

echo "\end{tabular}" >> $out_m_1".tex"; echo "\end{center}" >>
$out_m_1".tex"; echo "\end{document}" >> $out_m_1".tex";
command_latex="latex "$out_m_1; eval $command_latex;
command_latex="dvips "$out_m_1".dvi"; eval $command_latex;
#command_latex="gv "$out_m_1".ps"; #eval $command_latex;
```

(b) **Comandos para geração do Gráfico 1**

```
fs[x_] := 1/Floor[x + 1]
ff[x_] := 1/(x + 1)
fg[x_ /; 0<=x<1] := 1
fg[x_ /; x > 1] := 1/x
Plot[{fg[x], ff[x], fs[x]}, {x, 0, 10}]
```

(c) **Código *Permut.c* alterado**

```
/*
    Este programa foi alterado para receber o numero
    de sequencias a serem geradas e o valor de n
*/ #include <stdlib.h> #include <sys/time.h>

double rand0a1() {
    double resultado = (double) rand()/ RAND_MAX;
    /* Dividir pelo maior inteiro */
    if (resultado > 1.0)
        resultado= 1.0;
    return resultado;
}

void Permut(long* A, int n) { /* Obtem permutacao randomica dos
numeros entre 1 e n */
    int  i, j, b;

    for (i = n; i >= 1; i--) {
        j = (long)(i * rand0a1()  + 1);
        b = A[i-1];
        A[i-1] = A[j-1];
        A[j-1] = b;
    }
}

int main(int argc, char *argv[]) {
    int num_seq = 0;
    int k = 0;
    int n = 0, i = 0;
    long* A;
```

```
    struct timeval semente;
    gettimeofday(&semente,NULL); // utilizar o
    tempo como semente para a funcao srand()
    srand((int)(semente.tv_sec +
    1000000*semente.tv_usec));

    if(argc <= 2) {
        printf("Usage: ./rand_gen <numbers of keys>
         <number of sequences>\n");
        exit(0);
    }
    n = atoi(argv[1]);  //numero de chaves a
    serem permutadas
    num_seq = atoi(argv[2]);    //numero de sequencias
     a serem geradas
    A = (long *)malloc(sizeof(long) * n);
    if (A == NULL) {
        printf("ERRO: Nao foi possivel alocar memoria
         para variavel A\n");
        exit(0);
    }
    //printf("N: %i\tNumero de sequencias: %i\n",
    n, num_seq);
    for(k = 0; k < num_seq; k++) {
        for (i = 1; i <= n; i++)
            A[i-1] = i;
        Permut(A, n);
        for (i = 1; i <= n; i++)
            printf("%d ", A[i-1]);
        putchar('\n');
    }
    return 0;
}
```

(d) **Código *Árvore B* alterado**

```
#include<stdlib.h> #include<stdio.h> #include <string.h>
//#define m               2
//#define mm              (m * 2)
#define FALSE           0 #define TRUE            1

typedef long TipoChave;

typedef struct Registro {
    TipoChave Chave;
    /*outros componentes*/
} Registro;

typedef struct Pagina* Apontador;
```

```
typedef struct Pagina {
    short n;
    Registro *r;
    Apontador *p;
    //Registro r[mm];
    //Apontador p[mm + 1];
} Pagina;


void Inicializa(Apontador *Dicionario) {
    *Dicionario = NULL;
}

void Pesquisa(Registro *x, Apontador Ap) {
    long i = 1;
    if (Ap == NULL) {
        printf("Registro nao esta presente na arvore\n");
        return;
    }
    while (i < Ap->n && x->Chave > Ap->r[i-1].Chave)
        i++;
    if (x->Chave == Ap->r[i-1].Chave) {
        *x = Ap->r[i-1];
        return;
    }
    if (x->Chave < Ap->r[i-1].Chave)
        Pesquisa(x, Ap->p[i-1]);
    else
        Pesquisa(x, Ap->p[i]);
}

void InsereNaPagina(Apontador Ap, Registro Reg,
Apontador ApDir) {
    short NaoAchouPosicao;
    int k;

    k = Ap->n;
    NaoAchouPosicao = (k > 0);
    while (NaoAchouPosicao) {
        if (Reg.Chave >= Ap->r[k-1].Chave) {
            NaoAchouPosicao = FALSE;
            break;
        }
        Ap->r[k] = Ap->r[k-1];
        Ap->p[k+1] = Ap->p[k];
        k--;
```

```
        if (k < 1)
            NaoAchouPosicao = FALSE;
    }
    Ap->r[k] = Reg;
    Ap->p[k+1] = ApDir;
    Ap->n++;
}

void Ins(Registro Reg, Apontador Ap, short *Cresceu, Registro
*RegRetorno,  Apontador *ApRetorno, int m_order, int *prob) {
    long i = 1;
    long j;
    Apontador ApTemp;

    if (Ap == NULL) {
        *Cresceu = TRUE;
        (*RegRetorno) = Reg;
        (*ApRetorno) = NULL;
        return;
    }

    //Anisio's modification
    if(Ap->n < (2*m_order)) {
        *prob = 1;
    }
    else {
        *prob = 0;
        //printf("N: %i\tm_order: %i\t%i\t%i\n",
        Ap->n, m_order, (*prob), Reg.Chave);
    }

    //terminate here

    while (i < Ap->n && Reg.Chave > Ap->r[i-1].Chave)
        i++;
    if (Reg.Chave == Ap->r[i-1].Chave) {
        printf(" Erro: Registro ja esta presente\n");
        *Cresceu = FALSE;
        return;
    }
    if (Reg.Chave < Ap->r[i-1].Chave)
        i--;
    Ins(Reg, Ap->p[i], Cresceu, RegRetorno,
    ApRetorno, m_order, prob);
    if (!*Cresceu)
        return;
    if (Ap->n < (2*m_order)){   /* Pagina tem espaco */
```

```
        InsereNaPagina(Ap, *RegRetorno, *ApRetorno);
        *Cresceu = FALSE;
        return;
    }
    /* Overflow: Pagina tem que ser dividida */
    ApTemp = (Apontador)malloc(sizeof(Pagina));
    ApTemp->r = (Registro *)malloc(sizeof(Registro)
    *2*m_order);
    ApTemp->p = (Apontador *)malloc(sizeof(Pagina)
    *((2*m_order)+1));
    ApTemp->n = 0;
    ApTemp->p[0] = NULL;
    if (i < m_order+1){
        InsereNaPagina(ApTemp, Ap->r[(2*m_order)-1],
         Ap->p[(2*m_order)]);
        Ap->n--;
        InsereNaPagina(Ap, *RegRetorno, *ApRetorno);
    }
    else
        InsereNaPagina(ApTemp, *RegRetorno, *ApRetorno);
    for (j = m_order + 2; j <= (2*m_order); j++)
        InsereNaPagina(ApTemp, Ap->r[j-1], Ap->p[j]);
    Ap->n = m_order;
    ApTemp->p[0] = Ap->p[m_order+1];
    *RegRetorno = Ap->r[m_order];
    *ApRetorno = ApTemp;
}

void Insere(Registro Reg, Apontador *Ap,
int m_order, int *prob) {
    short Cresceu;
    Registro RegRetorno;
    Pagina *ApRetorno, *ApTemp;

    Ins(Reg, *Ap, &Cresceu, &RegRetorno, &ApRetorno,
     m_order, prob);
    if (Cresceu){   /* Arvore cresce na altura
    pela raiz */
        ApTemp = (Pagina *)malloc(sizeof(Pagina));
        ApTemp->r = (Registro *)malloc
        (sizeof(Registro)*2*m_order);
        ApTemp->p = (Apontador *)malloc
        (sizeof(Pagina)*((2*m_order)+1));
        ApTemp->n = 1;
        ApTemp->r[0] = RegRetorno;
        ApTemp->p[1] = ApRetorno;
        ApTemp->p[0] = *Ap;
```

```
        *Ap = ApTemp;
    }
}

void Reconstitui(Apontador ApPag,
Apontador ApPai, int PosPai, short
*Diminuiu, int m_order) {
    Pagina *Aux;
    long DispAux, j;
    if (PosPai < ApPai->n) {
    /* Aux = Pagina a direita de ApPag */
        Aux = ApPai->p[PosPai+1];
        DispAux = (Aux->n - m_order + 1) / 2;
        ApPag->r[ApPag->n] = ApPai->r[PosPai];
        ApPag->p[ApPag->n + 1] = Aux->p[0];
        ApPag->n++;
        if (DispAux > 0){
        /* Existe folga: transfere de Aux
        para ApPag */
            for (j = 1; j < DispAux; j++)
                InsereNaPagina(ApPag, Aux->r[j-1],
                Aux->p[j]);
            ApPai->r[PosPai] = Aux->r[DispAux-1];
            Aux->n -= DispAux;
            for (j = 0; j < Aux->n; j++)
                Aux->r[j] = Aux->r[j + DispAux];
            for (j = 0; j <= Aux->n; j++)
                Aux->p[j] = Aux->p[j + DispAux];
            *Diminuiu = FALSE;
        }
        else
        { /* Fusao: intercala Aux em
        ApPag e libera Aux */
            for (j = 1; j <= m_order; j++)
        InsereNaPagina(ApPag, Aux->r[j-1],
        Aux->p[j]);
        free(Aux);
            for (j = PosPai + 1; j < ApPai->n;
            j++)
        { ApPai->r[j-1] = ApPai->r[j];
        ApPai->p[j] = ApPai->p[j+1]; }
        ApPai->n--;
        if (ApPai->n >= m_order)
        *Diminuiu = FALSE;
        }
    }
    else {  /* Aux = Pagina a esquerda de ApPag */
```

```
        Aux = ApPai->p[PosPai-1];
        DispAux = (Aux->n - m_order + 1) / 2;
        for (j = ApPag->n; j >= 1; j--)
            ApPag->r[j] = ApPag->r[j-1];
        ApPag->r[0] = ApPai->r[PosPai-1];
        for (j = ApPag->n; j >= 0; j--)
            ApPag->p[j+1] = ApPag->p[j];
        ApPag->n++;
        if (DispAux > 0) {  /* Existe folga:
        transfere de Aux para ApPag */
            for (j = 1; j < DispAux; j++)
                InsereNaPagina(ApPag, Aux->r[Aux->n - j],
                Aux->p[Aux->n - j + 1]);
            ApPag->p[0] = Aux->p[Aux->n - DispAux + 1];
            ApPai->r[PosPai-1] = Aux->r[Aux->n - DispAux];
            Aux->n -= DispAux;
            *Diminuiu = FALSE;
        }
        else {  /* Fusao: intercala ApPag
        em Aux e libera ApPag */
            for (j = 1; j <= m_order; j++)
                InsereNaPagina(Aux, ApPag->r[j-1], ApPag->p[j]);
            free(ApPag);
            ApPai->n--;
            if (ApPai->n >= m_order)  *Diminuiu = FALSE;
        }
    }
}

void Antecessor(Apontador Ap, int Ind,
Apontador ApPai, short
*Diminuiu, int m_order) {
    if (ApPai->p[ApPai->n] != NULL) {
        Antecessor(Ap, Ind, ApPai->p[ApPai->n],
        Diminuiu, m_order);
        if (*Diminuiu)
        Reconstitui(ApPai->p[ApPai->n], ApPai,
        (long)ApPai->n, Diminuiu, m_order);
        return;
    }
    Ap->r[Ind-1] = ApPai->r[ApPai->n - 1];
    ApPai->n--; *Diminuiu = (ApPai->n < m_order);
}

void Ret(TipoChave Ch, Apontador *Ap, short
*Diminuiu, int m_order)
{
```

```
    long j, Ind = 1;
    Apontador Pag;

    if (*Ap == NULL) {
        printf("Erro: registro nao esta na arvore\n");
        *Diminuiu = FALSE;
        return;
    }
    Pag = *Ap;
    while (Ind < Pag->n && Ch > Pag->r[Ind-1].Chave)
        Ind++;
    if (Ch == Pag->r[Ind-1].Chave) {
        if (Pag->p[Ind-1] == NULL) {/* Pagina folha */
            Pag->n--;
            *Diminuiu = (Pag->n < m_order);
            for (j = Ind; j <= Pag->n; j++) {
                Pag->r[j-1] = Pag->r[j];
                Pag->p[j] = Pag->p[j+1];
            }
        return;
        }
        /* Pagina nao e folha: trocar
        com antecessor */
        Antecessor(*Ap, Ind, Pag->p[Ind-1],
        Diminuiu, m_order);
        if (*Diminuiu)
        Reconstitui(Pag->p[Ind-1], *Ap, Ind - 1,
        Diminuiu, m_order);
        return;
    }
    if (Ch > Pag->r[Ind-1].Chave)
        Ind++;
    Ret(Ch, &Pag->p[Ind-1], Diminuiu, m_order);
    if (*Diminuiu)
        Reconstitui(Pag->p[Ind-1], *Ap, Ind - 1,
        Diminuiu, m_order);
}

void Retira(TipoChave Ch, Apontador *Ap,
int m_order) {
    short Diminuiu;
    Apontador Aux;

    Ret(Ch, Ap, &Diminuiu, m_order);
    if (Diminuiu && (*Ap)->n == 0) {
    /* Arvore diminui na altura */
        Aux = *Ap;
```

```
        *Ap = Aux->p[0];
        free(Aux);
    }
}

void ImprimeI(Apontador p, int nivel) {
    long i;

    if (p == NULL)
        return;
    printf("Nivel %d : ", nivel);
    for (i = 0; i < p->n; i++)
        printf("%ld ",(long)p->r[i].Chave);
    putchar('\n');
    nivel++;
    for (i = 0; i <= p->n; i++)
        ImprimeI(p->p[i], nivel);
}

void Imprime(Apontador p) {
    int  n = 0;

    ImprimeI(p, n);
}

void CompProbPsmp2(Apontador p,
float *total, float *used, int
m_order, int nivel, int height) {
    long i;

    //if(p == NULL)
        //return;

    if ((nivel + 1) == height) {
        (*total) = (*total) + 1;
        if ((p->n) < (2 * m_order)) {
            (*used) = (*used) + 1;
        }
        return;
    }
    nivel++;
    for (i = 0; i <= p->n; i++)
        CompProbPsmp2(p->p[i], total,
        used, m_order, nivel, height);
}

void CompUtilMem(Apontador p,
```

```
float *total, float *used, int
m_order) {
    long i;

    if(p == NULL)
        return;

    (*total)++;
    (*used) = (*used) + (p->n);

    for (i = 0; i <= p->n; i++)
        CompUtilMem(p->p[i], total,
        used, m_order);
}

void CompProbPsmp(Apontador p,
int height, int nivel, int m_order,
int *prob) {
    long i;

    if(p == NULL)
        return;
    if((nivel + 1) == height) {
        if((p->n) < (2 * m_order))
        {    //Segura insercao
            (*prob) = 1;
        }
    }
    nivel++;
    for(i = 0; i <= p->n; i++) {
        CompProbPsmp(p->p[i], height,
        nivel, m_order, prob);
    }
}

void CompHight(Apontador p, int *height) {
    if(p == NULL)
        return;
    (*height)++;
    CompHight(p->p[0], height);
}

void TestaI(Apontador p, int pai, short
direita) {
    int i;
    int antecessor = 0;
    if (p == NULL)
```

```
    return;
    if (p->r[0].Chave > pai && direita ==
    FALSE) {
        printf("Erro: filho %12ld maior que
        pai %d\n", p->r[0].Chave, pai);
        return;
    }
    for (i = 0; i < p->n; i++) {
        if (p->r[i].Chave <= antecessor) {
            printf("Erro: irmao %ld maior que
            irmao a esquerda %d\n",
            (long)p->r[i].Chave, antecessor);
            return;
        }
            antecessor = p->r[i].Chave;
    }
    for (i = 0; i < p->n; i++)
        TestaI(p->p[i], p->r[i].Chave, FALSE);
    TestaI(p->p[p->n], p->r[i].Chave, TRUE);
}

void Testa(Apontador p) {
    int i;

    if (p == NULL)
        return;
    for (i = 0; i < p->n; i++)
        TestaI(p->p[i], p->r[i].Chave, FALSE);
    TestaI(p->p[p->n], p->r[i].Chave, TRUE);
}

void get_keys(FILE *fp_input, int n_keys, int *keys) {
    int temp = 0, k = 0;

    for(k = 0; k < n_keys; k++) {
        fscanf(fp_input, "%i", &temp);
        keys[k] = temp;
    }
}

int main(int argc, char *argv[]) {
    //Registro x;
    Pagina *D;
    Pagina *D_new;
    Registro x_new;
    int m_order = 0, n_keys = 0, k = 0;
    int height = 0, prob = 0, nivel = 0;
```

```
float perc_prob = 0.0;
float perc_used = 0.0, perc_total = 0.0;
int s = 0;
float total = 0.0, used = 0.0,
taxa_util = 0.0, taxa_util_wo_apont = 0.0;
char input_file_name[50];
FILE *fp_input;
int *keys = NULL;

if(argc <= 3) {
    printf("Usage: ./b_tree <m_order>
    <n_keys> <input file>\n");
    exit(0);
}

Inicializa(&D);
Inicializa(&D_new);

m_order = atoi(argv[1]);
n_keys = atoi(argv[2]);
strcpy(input_file_name, argv[3]);
//printf("file: %s\n", input_file_name);
fp_input = fopen(input_file_name, "r");
if (fp_input == NULL) {
    printf("ERRO: Nao foi possivel
    abrir o arquivo.\n");
    exit(0);

}

keys = (int *)malloc(sizeof(int) * (n_keys));
get_keys(fp_input, n_keys, keys);

//printf("Ordem da arvore: %i
\tNumero de chaves: %i\n", m_order, n_keys);
//printf("keys: ");
for(k = 0; k < n_keys; k++) {
    //printf("%i ", keys[k]);
    x_new.Chave = keys[k];
    Insere(x_new, &D_new, m_order, &prob);
}
//printf("\n");

//Imprime(D_new);
CompHight(D_new, &height);
CompUtilMem(D_new, &total, &used, m_order);
//printf("m: %i\tn: %i\theight: %i\tprob: %i
```

```
    \tused: %.0f\ttotal: %.0f\n",
    m_order, n_keys, height, prob, used, total);
    //CompProbPsmp2(D_new, &perc_total, &perc_used,
    m_order, nivel, height);
    //printf("m: %i\tn: %i\theight: %i\tprob: %i\t
    used: %.0f\ttotal: %.0f\n",
    m_order, n_keys, height, prob, used, total);

    //CompProbPsmp(D_new, height, nivel,
    m_order, &prob);
    //printf("m: %i\tn: %i\theight: %i\tprob:
    %i\ttotal: %.0f\tused: %.0f\n",
    m_order, n_keys, height, prob, total,
    used);
    //printf("numero de nodos: %.1f\tnumero
    de chaves: %.1f\n", total, used);

    taxa_util = ((used) + (total)) /
    (((4 * m_order) + 1) * (total));
    taxa_util_wo_apont = (used) /
    (2 * m_order * total);
    //perc_prob = (perc_used) / (perc_total);
    printf("%i %i %i %i %.8f %0.8f\n",
    m_order, n_keys, height, prob, taxa_util,
    taxa_util_wo_apont);

    return 0;
}
```