RUI ALBERTO ECKE TAVARES

UM ALGORITMO PARA PAGINAÇÃO DE ÁRVORES BINÁRIAS DE PESQUISA UTILIZANDO EMPACOTAMENTO UNIDIMENSIONAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, Curso de Mestrado em Informática, Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Elias P. Duarte Jr.

CURITIBA

2003

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluno Rui Alberto Ecke Tavares, avaliamos o trabalho intitulado, "*Um Algoritmo para Paginação de Árvores Binárias de Pesquisa Utilizando Empacotamento* ", cuja defesa foi realizada no dia 07 de julho de 2003, às dez horas, no Auditório da Informática da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 07 de julho de 2003.

Prof. Dr. Elias Procópio Duarte Jr
DINF/UFPR (Orientador)

Prof. Dr. Nivio Ziviani
UFMG

Profª. Dra. Cristina Duarte Murta
DINF/UFPR

Agradecimentos

À minha esposa Daniela e à pequena Julia pela compreensão, carinho e apoio constantes.

Ao orientador Prof. Dr. Elias Procópio Duarte Jr., pelo incentivo, apoio, paciência e horas de dedicação que foram vitais para a conclusão deste trabalho.

Aos demais professores do curso de Mestrado em Informática da Universidade Federal do Paraná, pelos ensinamentos.

Ao corpo diretivo da Universidade Tuiuti do Paraná, pelo estímulo e apoio à capacitação docente, em especial aos amigos e colegas de trabalho Prof. Francisco Carlos Sardo e Profª Denize Carneio de Campos.

Aos *designers* gráficos Fabrício Pires dos Santos e Neilor Pereira Stockler Jr., pelo apoio na elaboração de parte das ilustrações.

A minha irmã Úrsula e ao Prof. Dr. Camilo Ferronato pelo auxílio nas traduções para a língua inglesa do resumo desta dissertação e dos artigos de pesquisa desenvolvidos.

Obrigado ao Prof. Dr. Nívio Ziviani, pelas sugestões de aperfeiçoamento e pela participação na banca examinadora. Também à Profª. Drª. Cristina Duarte Murta e ao Prof. Dr. André Luiz Pires Guedes, pelos pareceres para a melhoria desta dissertação.

Por fim, agradeço a todos aqueles que confiaram em mim.

Sumário

Lista de Ilustrações

Lista de Tabelas

Resumo

As árvores binárias são estruturas de dados utilizadas tradicionalmente para a realização de pesquisa de forma eficiente sobre um conjunto de dados. Uma árvore pode atingir grandes dimensões, bem como ser utilizada para armazenar dados em memória secundária ou distribuídos pelos nodos de uma rede de computadores. Nestes casos, é necessário definir uma estratégia eficiente para o acesso aos dados da árvore, que são organizados em páginas. Uma página é utilizada para a transferência de dados em blocos da memória secundária para a primária, além do acesso remoto em redes de computadores, por intermédio de pacotes que possuem tamanho máximo pré-fixado. Este trabalho apresenta um algoritmo para a paginação de árvores binárias de pesquisa aplicável quando o conjunto de informações é estático, as freqüências de acesso não são conhecidas e o armazenamento é remoto ou secundário. O algoritmo visa reduzir o tempo de pesquisa aos dados armazenados na árvore binária em termos do número de páginas visitadas e do aumento da taxa de preenchimento das páginas utilizadas. Uma versão alternativa do algoritmo que visa reduzir a distância internodal nas páginas é apresentada. Observou-se que o algoritmo proposto constrói a paginação ótima quando possível, isto é, quando a árvore é completa e o número de nodos é múltiplo do tamanho da página. Além disso, propõe-se uma política eficiente para o preenchimento das páginas de uma árvore binária degenerada tendo por base a aplicação de empacotamento unidimensional na franja da árvore. A complexidade computacional do algoritmo, que depende do empacotamento unidimensional a ser utilizado, é discutida e apresentada. O algoritmo foi implementado e resultados experimentais quanto ao número de páginas visitadas e à taxa de preenchimento das páginas utilizadas, comparativos com a paginação seqüencial, valores ótimos teóricos e as árvores B, são descritos e analisados. Comparando o algoritmo proposto com as árvores B, enquanto o número de paginas visitadas por pesquisa é similar em ambas as abordagens, a taxa de preenchimento das páginas produzida pelo algoritmo proposto é mais de 30% superior à taxa obtida pelas árvores B.

Abstract

Binary trees are data structures traditionally used to search efficiently on large amounts of data. If the tree is too large to be completely stored in main memory it must be allocated in pages which are transferred in blocks from secondary to main memory. Analogously, if the tree is stored in a remote computer connected to a network, it is also necessary to allocate the tree in pages which are sent through the network in packets. This work presents a new algorithm for paging binary trees applicable when the information set is static, the access frequencies are not known and the storage is remote or secondary. The algorithm aims at reducing the number of pages visited for searching, and at decreasing the unused space in each page as well as reducing the total number of pages required to store a tree. An alternative version of the algorithm which aims at reducing the distance between nodes stored in a given page is presented. The algorithm builds the best possible paging when it is possible, and presents an efficient strategy for allocating degenerated trees based on one-dimensional packing. The complexity of the algorithm is presented, which depends on the complexity of one-dimensional packing. Experimental results are reported and compared with other approaches, including sequential paging and B-trees. The comparison with B-trees shows that while the number of pages accessed in a search is similar in both approaches, the page filling percentage of the proposed algorithm is over 30% larger than that obtained with B-trees.

Capítulo 1

Introdução

Busca ou pesquisa é o procedimento de recuperação de informações de acordo com um valor dado na entrada do processo. Existem muitos métodos de pesquisa e diversos fatores são levados em consideração para se adotar um ou outro método [1, 2, 3]. As árvores binárias de pesquisa são uma das mais flexíveis e completas técnicas para organização de grandes quantidades de dados [4]. A sua importância prática decorre do fato de que realizam com grande eficiência o processo de pesquisa e com eficiência razoável as demais operações sobre arquivos: processamento aleatório e seqüencial, inserção, modificação e remoção de registros e reestruturação de arquivos.

Uma árvore pode atingir grandes dimensões, bem como ser utilizada para armazenar dados em memória secundária [5, 6, 7] ou distribuídos pelos nodos de uma rede de computadores [8]. Nestes casos, é necessário definir uma estratégia eficiente para o acesso aos dados da árvore, que são organizados em páginas. Uma página é utilizada para a transferência de dados em blocos da memória secundária para a primária, além do acesso remoto em redes de computadores, por intermédio de pacotes que possuem tamanho máximo pré-fixado [8].

Os algoritmos para tratamento da paginação de estruturas de dados são utilizados em diversos sistemas de informação [9]. Com isso, os critérios para alocação dos dados nas páginas são essenciais para a eficiência destes sistemas. Neste trabalho apresenta-se um algoritmo para realizar a paginação de árvores binárias de pesquisa baseado em empacotamento unidimensional [10, 11, 12]. O algoritmo visa reduzir o número de páginas visitadas em buscas e, ao mesmo tempo, aumentar a taxa de

preenchimento das páginas utilizadas, sendo aplicável quando o conjunto de informações é estático, as freqüências de acesso não são conhecidas e o armazenamento é remoto ou secundário.

Uma outra abordagem para solucionar o problema tratado neste trabalho são as árvores B, que se configuram como árvores balanceadas de múltiplos caminhos utilizadas como método de acesso para bases de dados que não podem ser armazenadas em memória principal [2, 7, 13].

Considerando os experimentos realizados, o algoritmo proposto neste trabalho apresenta uma taxa de preenchimento das páginas 30% superior à taxa obtida pelas árvores B, com desempenho similar quanto ao número de paginas visitadas.

1.1 Paginação de Árvores Binárias de Pesquisa

Uma árvore binária é um conjunto finito de elementos que pode ser vazio ou dividido em 3 subconjuntos disjuntos [2, 5, 6, 7]. O primeiro subconjunto contém um único elemento chamado raiz da árvore. Os outros dois subconjuntos são, eles mesmos, árvores binárias, chamados de subárvore esquerda e subárvore direita da árvore original. As subárvores (esquerda ou direita) podem ser vazias. Cada elemento de uma árvore binária é chamado de nodo da árvore.

Se o conjunto de nodos de uma árvore binária de pesquisa é muito grande para ser todo mantido na memória principal, então ele deve ser alocado em memória secundária. De modo análogo, a árvore pode estar armazenada em uma máquina remota conectada em rede de computadores. Durante a realização de uma pesquisa a árvore estará apenas parcialmente armazenada na memória principal da máquina que executa a pesquisa. Para viabilizar a transferência de dados os nodos da árvore são agrupados em blocos ou pacotes de tamanho fixo, chamados páginas. Cada página é composta de células, cada nodo da árvore é armazenado em uma célula. Como o tempo necessário para acessar ou visitar uma página é predominantemente o tempo de transferência da página, o desempenho do algoritmo de manipulação da árvore é estritamente relacionado ao número de páginas transferidas.

1.2 O Algoritmo Proposto

O algoritmo proposto é um paginador de árvores binárias que, na versão implementada, parte de uma paginação seqüencial existente e realiza a sua otimização, de forma a reduzir o número de páginas visitadas em pesquisas e aumentar a taxa de preenchimento das páginas.

O algoritmo proposto atinge o ideal de paginação, a ser definido no capítulo 2, quando a árvore binária é completa e o número de nodos é múltiplo do tamanho da página. Além disso, estabelece uma política eficiente de preenchimento de páginas, para a hipótese das árvores incompletas ou degeneradas.

O algoritmo inicia pela raiz da árvore a ser paginada alocando, sempre que possível, subárvores em páginas. Pode ser considerado guloso no sentido que procura armazenar em uma mesma página subárvores que a preencham completamente.

Quando sobram subárvores que não preenchem completamente uma página, ditas constituintes da franja da árvore, o algoritmo aloca essas subárvores em páginas utilizando empacotamento unidimensional [10, 11, 12]. Desta forma, cada página é preenchida visando a otimização dos acessos durante uma pesquisa na árvore binária.

O empacotamento unidimensional, em questão, consiste em alocar um conjunto de subárvores num conjunto de páginas, conhecendo o espaço disponível nas páginas, de modo a minimizar o número de páginas utilizadas. Por exemplo, na figura 1.1 a subárvore *s1* formada pelos nodos 3, 5, 7 e 12; a subárvore *s2* formada pelos nodos 36, 38 e 41; a subárvore *s3* formada pelos nodos 46, 49, 53 e 57; a subárvore *s4* formada pelo nodo 73; a subárvore *s5* formada pelos nodos 83, 85 e 87 e a subárvore *s6* formada pelos nodos 93, 95, 97 e 98 devem ser alocadas no menor número possível de páginas. Observa-se que estamos diante de subárvores com tamanhos 4, 3, 4, 1, 3 e 4, respectivamente, e que devem ser alocadas em páginas de tamanho 7.

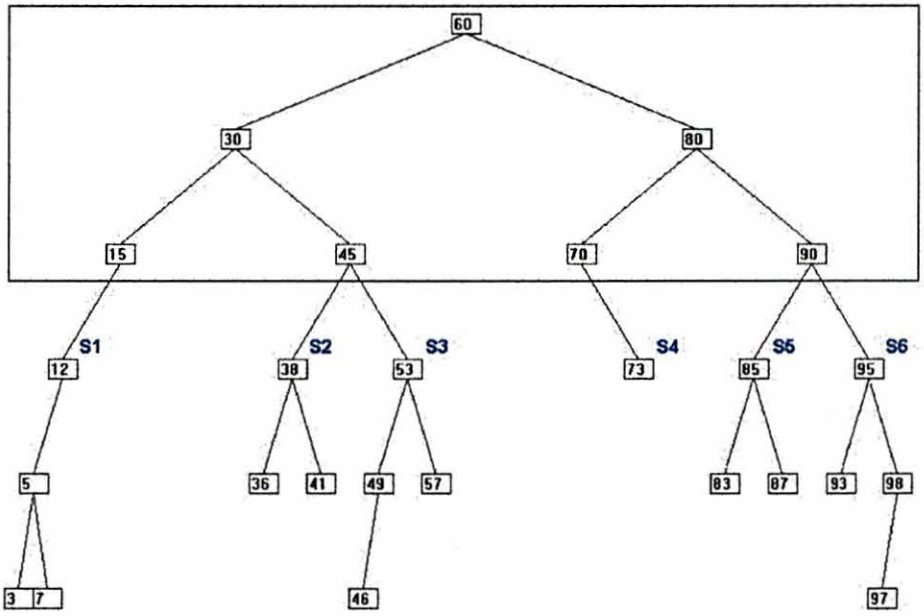

Figura 1.1 Exemplo de aplicação de empacotamento unidimensional.

Uma solução ótima, que é obtida pelo algoritmo, produz uma alta taxa de preenchimento das páginas, alocando as subárvores *s1* e *s2* na página 1, as subárvores *s3* e *s5* na página 2 e as subárvores *s4* e *s6* na página 3, conforme se observa na figura 1.2.

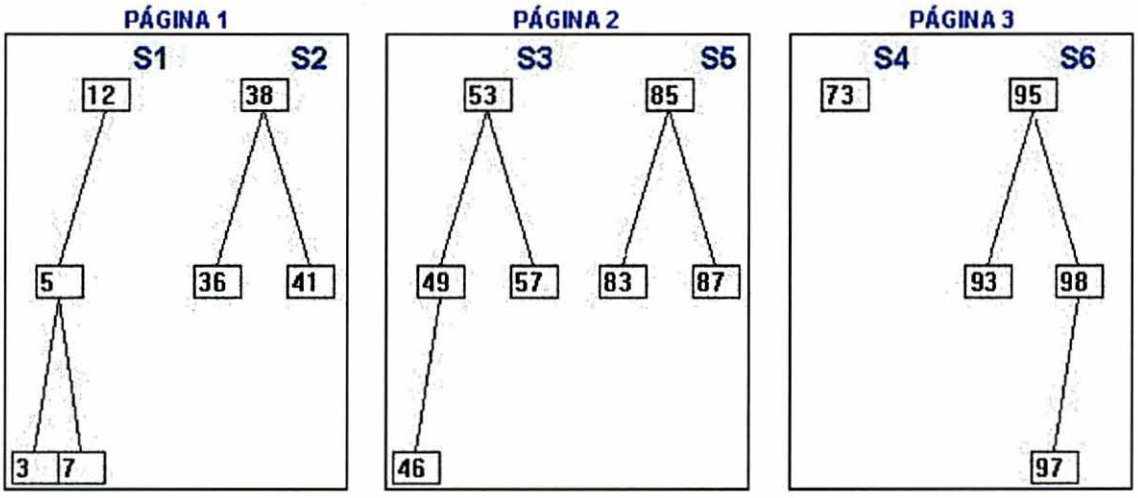

Figura 1.2 Paginação utilizando empacotamento unidimensional.

Para ilustrar o ganho do uso do empacotamento unidimensional, considere uma abordagem que preenche completamente as páginas com as subárvores de menor tamanho, para posteriormente alocar as demais subárvores. Dessa forma, seria obtida como resposta a alocação das subárvores *s2*, *s4* e *s5* na página 1 e das subárvores *s1*,

s3 e *s6* nas páginas 2, 3 e 4, produzindo uma quantidade maior de páginas, conforme se observa da figura 1.3.

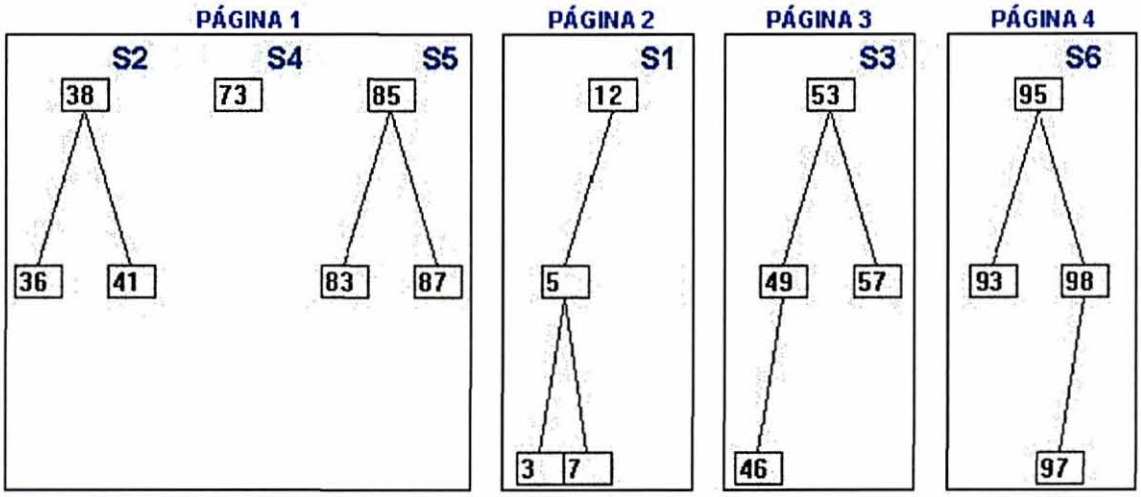

Figura 1.3 Paginação produzida por uma abordagem alternativa.

A especificação do algoritmo é apresentada, incluindo as estruturas de dados utilizadas para a manipulação de nodos e subárvores. A complexidade do algoritmo, que depende do empacotamento unidimensional a ser utilizado, é apresentada e discutida.

Uma versão do algoritmo foi implementada e resultados experimentais comparativos com a paginação seqüencial, com árvores B e com valores ótimos teóricos são apresentados. Uma medição empírica do desempenho do algoritmo quanto ao número de páginas visitadas em pesquisas, à distância internodal e à taxa de preenchimento das páginas é realizada. Uma versão alternativa do algoritmo que visa reduzir a distância internodal nas páginas é apresentada.

1.3 Trabalhos Relacionados

Intenso trabalho de investigação tem sido desenvolvido sobre algoritmos e estruturas de dados para tratamento de grandes quantidades de dados em memória externa [9, 14, 15], incluindo o problema da alocação de árvores de pesquisa.

O método de alocação seqüencial para a paginação dos nodos de uma árvore binária pode ser considerado como uma estratégia simples que serve de base para

outras abordagens [16]. Além desse método também foi proposto o método de paginação agrupada no qual os nodos de uma árvore são alocados nas páginas na ordem de entrada das chaves, procurando alocar os nodos de uma página de forma que mantenham uma relação de proximidade dentro da estrutura lógica da árvore [16]. Esse método foi aperfeiçoado e constatou-se que o mesmo produz resultados próximos do ótimo, porém com o prejuízo de que o número de páginas pode crescer indefinidamente [3, 17].

As árvores B, árvores balanceadas de múltiplos caminhos, configuram-se como uma das mais importantes técnicas de organização e manutenção de arquivos, sendo utilizadas como método de acesso para bases de dados que não podem ser armazenadas em memória principal [7, 13].

Em [18] são definidos critérios para a paginação de árvores binárias tendo em vista a organização de estruturas de arquivos. Uma técnica de alocação dinâmica de páginas para árvores binárias de pesquisa tendo por fundamento o conceito de divisão das páginas que ocorre nas árvores B e a alocação agrupada balanceada foi apresentada em [19]. Outra proposta para armazenamento secundário de árvores binárias de pesquisa na qual são utilizadas memórias magnéticas de bolha foi apresentado em [20]. Uma análise da quantidade de páginas acessadas em pesquisas na alocação seqüencial e uma variação do método de alocação agrupada de árvores binárias em armazenamento secundário são descritos em [21].

Outra abordagem para alocação agrupada de árvores binárias de pesquisa que busca o equilíbrio na quantidade de nodos nas páginas foi proposto em [22]. Uma abordagem para paginação de uma árvore binária parcialmente paginada utilizando balanceamento externo é apresentada em [23]. Uma primeira versão do algoritmo apresentado neste trabalho foi descrita em [24]. Por fim, entre os trabalhos relacionados destacam-se aqueles que visam o desenvolvimento de técnicas que utilizam da melhor forma possível as tecnologias de discos óticos [25, 26, 27].

1.4 Organização deste Trabalho

O presente trabalho está dividido em cinco capítulos, incluindo esta introdução. No capítulo 2 as árvores binárias e sua paginação são descritas. No capítulo 3 é descrita a funcionalidade do algoritmo, bem como a sua especificação formal e análise de complexidade. No capítulo 4 são descritas as métricas utilizadas para avaliar o desempenho do algoritmo e apresentados resultados experimentais obtidos com a versão implementada do algoritmo. No capítulo 5 apresentam-se as conclusões e as sugestões de trabalhos futuros. Nos anexos 1, 2 e 3 encontra-se o código fonte utilizado nas simulações. No anexo 4 encontram-se os resultados obtidos nos experimentos.

Capítulo 2

Árvores Binárias e sua Paginação

Considerando um conjunto específico de dados, o problema de encontrar unidades que atendam a um determinado requisito é conhecido como busca ou pesquisa [28]. A árvore binária é uma estrutura de dados que permite a pesquisa eficiente sobre um grande conjunto de dados [2, 5, 6, 7]. Este capítulo apresenta uma visão geral dos métodos básicos de pesquisa, além de uma introdução às árvores binárias. Nem sempre é possível manter toda a árvore em memória principal. Nestes casos a árvore é organizada em páginas, cada uma das quais contendo uma quantidade máxima de dados. Ao realizar uma pesquisa, as páginas vão sendo adequadamente movidas da memória secundária para a memória primária. Este capítulo examina os métodos básicos de organização de arquivos e apresenta o conceito de paginação de árvores binárias.

2.1 Métodos de Pesquisa

Considere um conjunto específico de dados armazenados em registros, cada um dos quais consistindo de uma série de campos de tipos pré-definidos. Um destes campos denominado *chave*, identifica de forma única cada registro do conjunto. Define-se *busca* ou *pesquisa* como o processo de recuperação de registros cuja chave tenha valor igual a um valor dado na entrada do processo.

O método de pesquisa mais simples é denominado *pesquisa seqüencial* [2]. Este método funciona da seguinte forma: a partir do primeiro registro, a chave procurada é

comparada com o campo correspondente de cada registro, seqüencialmente. O método termina com ou sem sucesso. Quando tem sucesso, é localizado um registro contendo o valor pesquisado; caso contrário nenhum registro apresenta tal valor. Considerando um total de n registros, no pior caso são efetuadas $O(n)$ comparações.

Caso os registros estejam armazenados ordenados pela chave de pesquisa, a *pesquisa binária* é um método mais eficiente que a pesquisa seqüencial [2]. Considere que os registros estão armazenados em uma tabela. A localização de um registro se faz comparando a chave de pesquisa com o registro do meio da tabela. Se a chave é menor, então o registro procurado está na metade inferior da tabela; se a chave é maior, está na metade superior. Considera-se apenas a metade da tabela desejada e repete-se o processo até que a chave seja encontrada, ou fique apenas um registro cuja chave é diferente da procurada, significando uma pesquisa sem sucesso. Considerando um total de n registros, no pior caso são efetuadas $O(\log n)$ comparações. Todos os logaritmos deste trabalho são base 2.

Outro método de pesquisa consiste na transformação de chave ou *hashing* [2]. Este método tem por base o fato de que os registros armazenados em uma tabela são diretamente endereçáveis a partir de uma transformação aritmética sobre a chave de pesquisa. Possui duas etapas principais: computar o valor da função de transformação (função *hashing*), a qual transforma a chave de pesquisa em um endereço da tabela; e resolver as colisões que ocorrem [2]. Considerando um total de n registros, no pior caso são efetuadas $O(n)$ comparações, quando há colisão de todos os registros.

Este trabalho trata das árvores de pesquisa. As árvores de pesquisa são estruturas de dados que possuem um amplo conjunto de operações, tendo aplicações diversas, podendo, por exemplo, ser utilizadas tanto como dicionário quanto como uma fila de prioridades [2]. As árvores são eficientes para armazenar informações, pois satisfazem a combinação de requisitos, tais como: acesso direto e seqüencial; facilidade de inserção e retirada de registros; adequada utilização da memória, entre outros [2]. As operações de pesquisa em árvores têm tempo proporcional à sua altura, o que viabiliza a sua utilização nas mais diversas aplicações. Considerando uma árvore de pesquisa com n nodos, no pior caso $O(\log n)$ comparações são realizadas. Dentre as

árvores de pesquisa destaca-se a árvore binária que tem suas propriedades detalhadas na próxima seção deste capítulo.

2.2 Árvores Binárias de Pesquisa

O objetivo desta seção é definir a estrutura de dados árvore binária e suas propriedades para com isso compreender o escopo de aplicação do algoritmo que será apresentado no capítulo seguinte.

2.2.1 Definições Preliminares

Uma árvore binária é um conjunto finito de elementos que pode ser vazio ou dividido em 3 subconjuntos disjuntos [2, 5, 6, 7]. O primeiro subconjunto contém um único elemento chamado raiz da árvore. Os outros dois subconjuntos são, eles mesmos, árvores binárias, chamados de subárvore esquerda e subárvore direita da árvore original. As subárvores (esquerda ou direita) podem ser vazias. Cada elemento de uma árvore binária é chamado de nodo da árvore.

As árvores binárias podem ser definidas recursivamente:

i) A árvore binária T_0 de zero nodos é uma árvore binária.

ii) Uma árvore binária T_n de $n >= 1$ nodos é ordenada em uma tripla (T_{esq}, R, T_{dir}), onde R é o nodo simples dito raiz de T_n. T_{esq} e T_{dir} são as árvores binárias de *esq* e *dir* nodos, respectivamente ditas subárvores esquerda e direita da raiz ($esq >= 0$, $dir >= 0$ e $esq + dir = n - 1$).

Se A é a raiz de uma árvore binária e B é a raiz da sua subárvore esquerda ou direita, então A se diz pai de B e B é denominado o filho esquerdo ou direito de A.

A figura 2.1 ilustra um método convencional para descrever árvores binárias. As árvores binárias podem também ser representadas de diferentes formas como: conjuntos aninhados, parênteses aninhados, denteação e grafos [28].

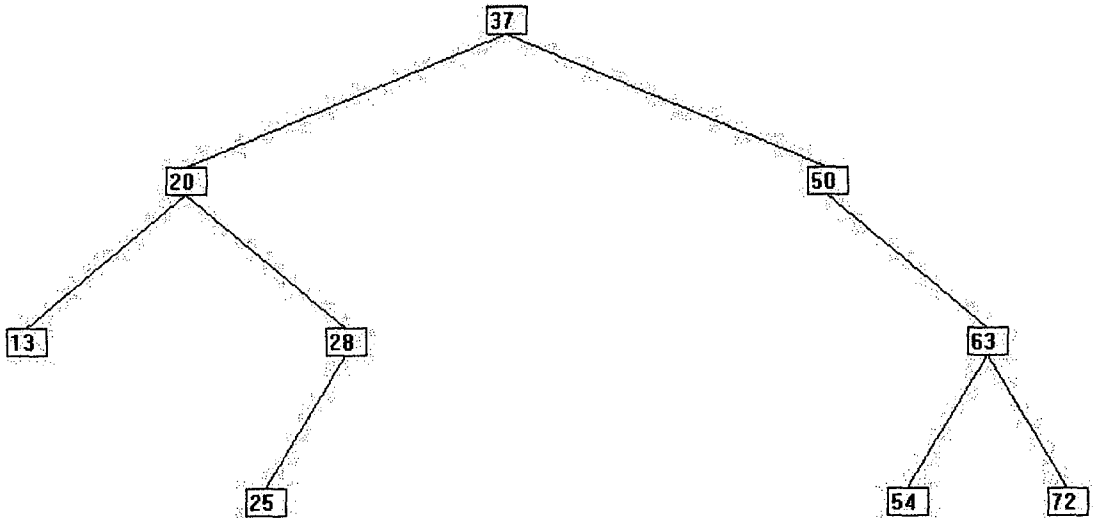

Figura 2.1 Uma árvore binária.

Ressalta-se que universalmente a estrutura de dados árvore possui raiz no topo e as folhas na base. A raiz se dirige para as folhas através de uma descida, ao passo que o caminhamento das folhas para a raiz é uma escalada. Na figura 2.1 tem-se uma árvore com 9 nodos, cuja raiz tem como chave de pesquisa 37. A subárvore esquerda tem por raiz 20 e a subárvore direita tem raiz 50. Isto é indicado pelas duas arestas que saem de 37; para 20 na esquerda e para 50 na direita. A ausência de uma aresta indica uma subárvore vazia. Por exemplo, a subárvore esquerda com raiz 50 e a subárvore direita com raiz 28, são ambas vazias. As subárvores com raiz 13, 25, 54 e 72 têm as subárvores esquerda e direita vazias. Um nodo que não possui filhos (como 13, 25, 54 e 72) é chamado de folha.

Um nodo *A* é um ancestral do nodo *B* (por conseguinte *B* é um descendente de *A*) se *A* é ou o pai de *B* ou o pai de algum ancestral de *B*. Por exemplo, na árvore da figura 2.1, 37 é um ancestral de 25, 54 é um descendente de 50, 28 não é nem um ancestral nem um descendente de 50. Um nodo *B* é um descendente esquerdo do nodo *A* se *B* é ou filho esquerdo de *A* ou um descendente do filho esquerdo de *A*. Um descendente direito pode ser similarmente definido. Dois nodos são irmãos se eles são filhos esquerdo e direito do mesmo pai.

Se todo nodo não folha de uma árvore binária possuir subárvores esquerda e direita não vazias, a árvore é denominada árvore estritamente binária. Assim a árvore

da figura 2.2 é estritamente binária, enquanto a árvore da figura 2.1 não é (porque os nodos 50 e 28 têm somente um filho cada). Uma árvore estritamente binária com n folhas sempre contém $2n-1$ nodos.

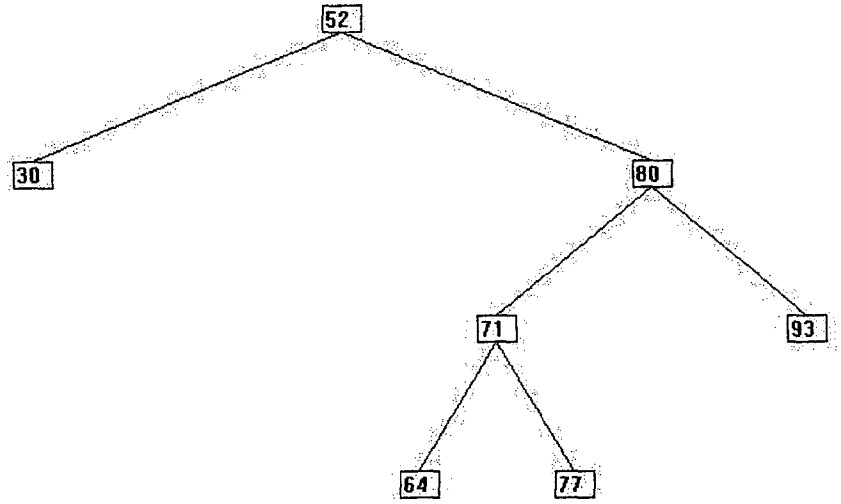

Figura 2.2 Árvore estritamente binária.

O nível de um nodo em uma árvore binária é definido como se segue: a raiz da árvore tem nível 0, e o nível de qualquer outro nodo na árvore é um a mais do que o nível do seu pai. Por exemplo, na árvore binária da figura 2.2, o nodo 71 está no nível 2 e o nodo 77 está no nível 3. A profundidade ou altura de uma árvore binária é o nível máximo de qualquer folha na árvore. Isto é igual ao comprimento do mais longo caminho da raiz a qualquer folha. Assim a profundidade da árvore da figura 2.3 é 3. Uma árvore binária completa de profundidade d é a árvore binária completa cujas folhas estão no nível d. A figura 2.3 ilustra uma árvore binária completa de profundidade 3. Uma árvore é dita balanceada se e somente se para qualquer nó, a altura de suas subárvores diferem de, no máximo, uma unidade. As árvores que satisfazem a esta condição são denominadas árvores *AVL* [29].

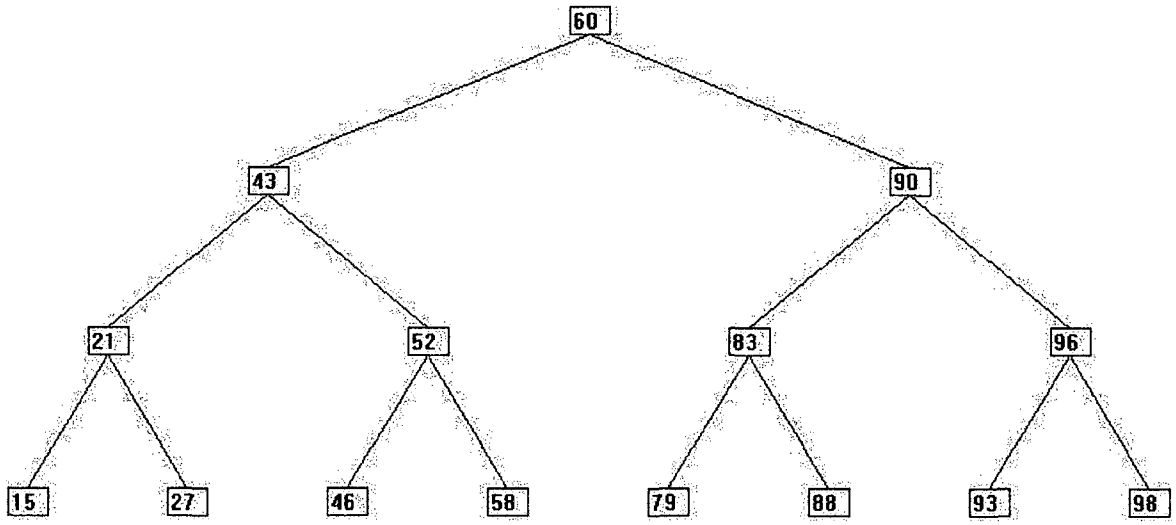

Figura 2.3 Uma árvore binária completa.

2.2.2 Pesquisa em Árvores Binárias

Neste trabalho as árvores binárias são utilizadas para realização de pesquisa. Uma árvore binária de pesquisa com N nodos com chaves $X_1,...,X_N$ é a árvore binária T_N em que cada um dos nodos é rotulado com distintas chaves escolhidas entre $X_1,...,X_N$ tais que para cada nodo i é satisfeita a seguinte propriedade: todas as chaves na subárvore esquerda de i são menores que a chave rotulada i, e todas as chaves na subárvore direita de i são maiores. A pesquisa a uma chave j deverá ser realizada da seguinte forma: i) se não existe raiz, então j não está na árvore e a pesquisa termina sem sucesso; ii) se j é igual a chave da raiz, então a pesquisa termina com sucesso; iii) se j precede a chave da raiz, então a pesquisa prossegue recursivamente analisando-se a subárvore esquerda da raiz de modo semelhante; iv) se j segue a chave da raiz, então a pesquisa prossegue recursivamente pelo exame da subárvore direita da raiz.

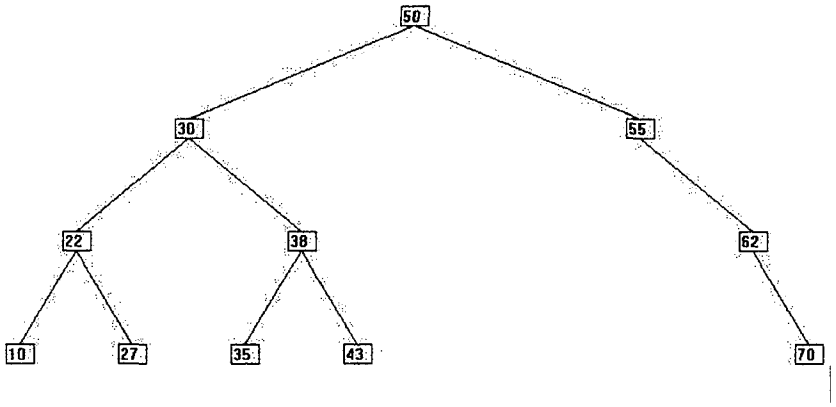

Figura 2.4 Exemplo de uma árvore binária de pesquisa.

Com base na figura 2.4 observa-se que os elementos da subárvore direita da raiz 50 são numericamente posteriores, ao passo que os elementos da subárvore esquerda da raiz 50 são numericamente precedentes a 50. Para se pesquisar a chave 35, ter-se-á sucesso e serão consultados, respectivamente os nodos contendo as chaves 50, 30, 38 e 35. Para se pesquisar a chave 83, teremos fracasso e serão consultados 50, 55, 62 e 70. Salienta-se que a pesquisa depende do modo com que os nodos da árvore estão organizados. Dessa forma, as árvores balanceadas têm em média um caminho de pesquisa menor do que uma árvore desbalanceada.

2.3 Métodos de Paginação de Árvores

Se o conjunto de chaves de uma árvore binária de pesquisa é muito amplo para ser todo mantido na memória principal, ele deve ser alocado em memória secundária. Durante a realização de uma pesquisa a árvore estará apenas parcialmente armazenada na memória principal. Como a transferência de dados entre a memória principal e a secundária é realizada em blocos de tamanho fixo chamados páginas, a árvore também deve ser dividida em páginas. Cada página é composta de células, cada nodo da árvore é armazenado em uma célula. Como o tempo necessário para acessar ou visitar uma página é predominantemente o tempo para transferir uma página da memória secundária para a memória principal, o desempenho do algoritmo de manipulação da árvore é estritamente relacionado ao número de páginas transferidas.

De forma análoga, se a árvore está armazenada em uma máquina remota de uma rede de computadores, a transferência de nodos para a máquina na qual a pesquisa é realizada também implica no agrupamento dos nodos em pacotes, equivalentes às páginas, cujo tamanho máximo corresponde ao MTU (*Maximum Transfer Unit*) [8] da rede física utilizada; por exemplo, no caso da rede Ethernet, 1500 *bytes*.

Durante o caminhamento pela árvore verifica-se se a página a ser visitada está na memória principal. Se estiver, o acesso é realizado, caso contrário, verifica-se a existência de espaço para mais uma página. Em caso afirmativo a alocação apropriada é efetuada e o acesso é realizado; se não há espaço, uma política de substituição de páginas, previamente determinada, deve ser executada. Na figura 2.5 exemplifica-se a paginação de uma árvore binária, onde o endereço de um nodo é composto pela tupla (número da página, posição na página). Na página 1 estão armazenados os nodos 4, 2, e 6. Na página 2 estão armazenados os nodos 1, 3 e 5. Na página 3 apenas o nodo 7 está armazenado.

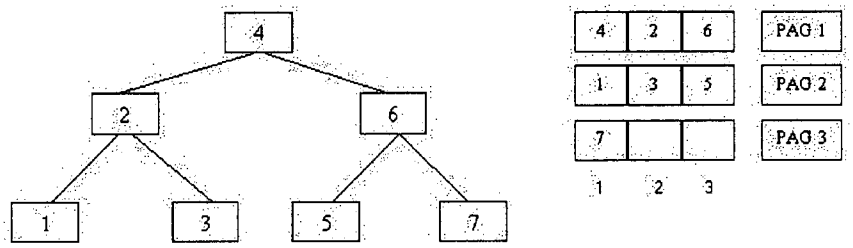

Figura 2.5 Exemplo de uma paginação de árvore binária.

Diversas são as políticas para a substituição de páginas, entre elas destacam-se FIFO (*First-In-First-Out*), LFU (Least-*Frequently-Used*), LRU (*Least-Recently-Used*), MFU (*Most Frequently Used*), NRU (*Not-Recently-Used*), entre outras [30, 31, 32].

Adotada uma política para substituição de páginas, o objetivo é reduzir a quantidade de transferências de páginas da memória secundaria para a memória primária, ou entre computadores de uma rede.

A árvore deve, então, ser paginada, de forma que seja minimizado o número de acessos a páginas. A melhor alocação de páginas é obtida agrupando-se os nodos da

árvore binária como na figura 2.6, na qual os retângulos representam as páginas onde os nodos estão alocados. Com tal disposição dos nodos nas páginas o número de páginas visitadas em cada pesquisa é o mínimo possível [19].

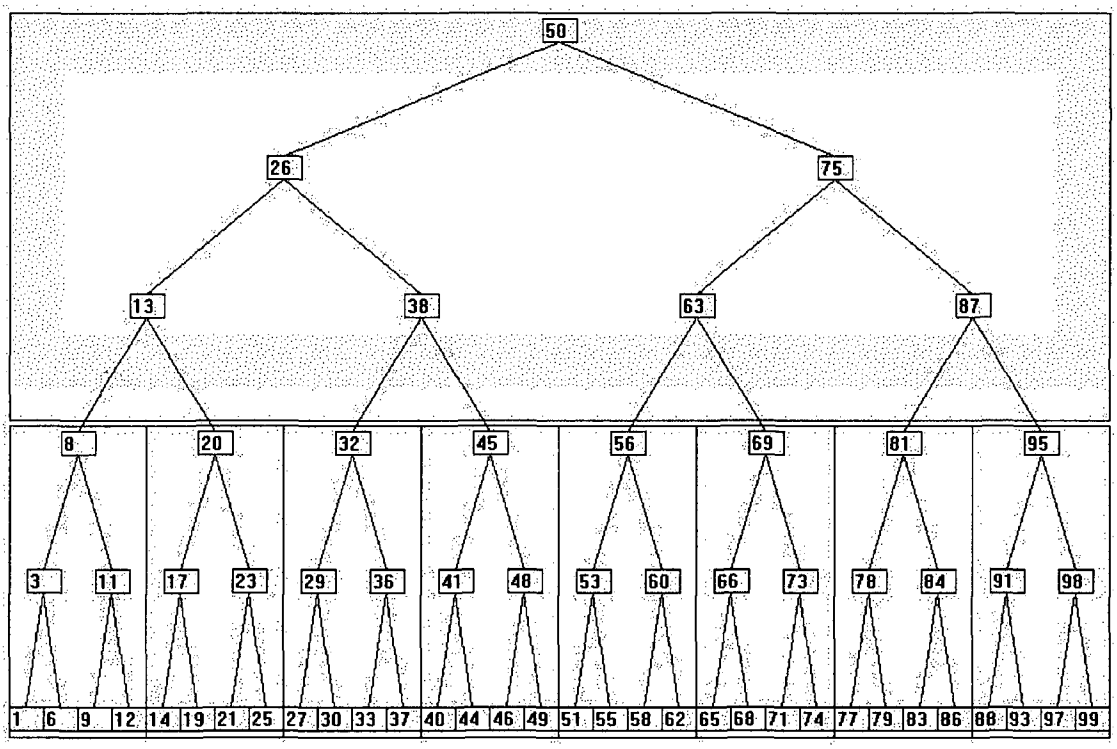

Figura 2.6 A alocação ideal de páginas.

A forma mais simples de alocar os nodos em páginas é dita *paginação seqüencial*, isto é, à medida que novos nodos são inseridos na árvore eles são alocados seqüencialmente em uma página. Quando o espaço desta página se esgota, uma nova página é criada para os próximos nodos a serem inseridos.

O desempenho da pesquisa em uma árvore armazenada com a paginação seqüencial é inferior ao desempenho de uma pesquisa realizada em uma árvore armazenada com a paginação ideal. O exemplo a seguir ilustra esta diferença.

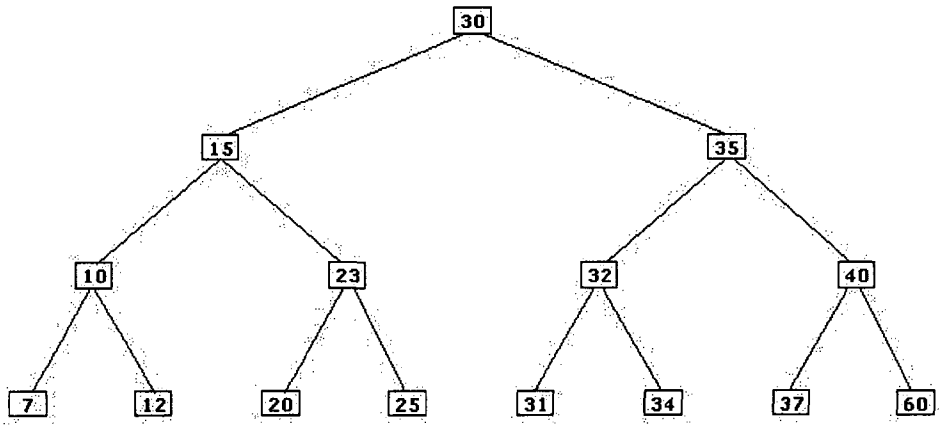

Figura 2.7 Árvore binária completa de 4 níveis.

A figura 2.7 nos mostra a árvore obtida com a inserção dos nodos com chaves numéricas: 30, 35, 40, 60, 37, 32, 15, 23, 25, 10, 31, 34, 7, 12 e 20. Considerando o tamanho de cada página como sendo 3, ou seja, uma página abrange dois níveis de uma árvore, a paginação seqüencial fornece a seguinte estrutura de páginas: página 1, nodos 30, 35 e 40; página 2, nodos 60, 37 e 32; página 3, nodos 15, 23 e 25; página 4, nodos 10, 31 e 34 e página 5, nodos 7, 12 e 20, conforme ilustrado na figura 2.8.

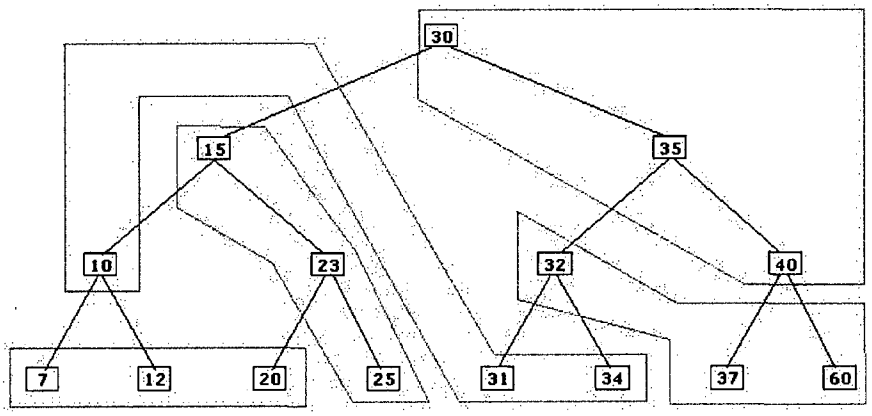

Figura 2.8 Exemplo de paginação obtida seqüencialmente.

Para a referida árvore a paginação ótima seria: página 1, nodos 30, 15 e 35; página 2, nodos 7, 10 e 12; página 3, nodos 23, 20 e 25; página 4, nodos 32, 31 e 34 e página 5, nodos 40, 37 e 60, conforme ilustrado na figura 2.9.

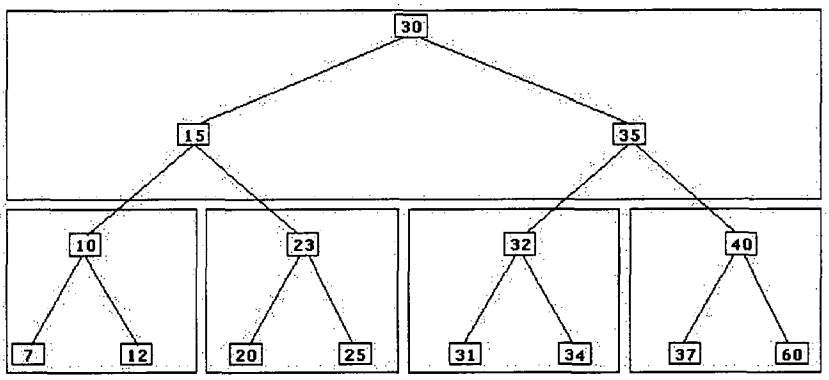

Figura 2.9 Esquema ótimo de paginação.

Considerando, por exemplo, a pesquisa da chave 7, tem-se para o primeiro esquema de páginas, quatro páginas visitadas; enquanto que a pesquisa da mesma chave utilizando a paginação ideal será atingida com a consulta de apenas duas páginas. Considerando todas as chaves observa-se que na paginação seqüencial o número médio de páginas visitadas é 2,33, enquanto na paginação ideal é 1,8.

2.4 Árvores B

As árvores B consistem em uma das mais importantes e tradicionais técnicas de organização e manutenção de arquivos [2, 13]. Dessa forma, configuram-se como árvores balanceadas de múltiplos caminhos freqüentemente utilizadas como método de acesso para bases de dados que não podem ser armazenadas em memória principal [7, 13]. Resultados de experimentos comparando o desempenho do algoritmo proposto com essa estratégia são apresentados no capítulo 4.

As árvores B podem ser chamadas de árvores de pesquisa *n*-árias, pois possuem mais de dois descendentes por nodo. Convencionalmente os nodos podem ser denominados páginas que armazenam informações. As árvores B possuem tamanhos variados, sendo referenciadas por sua ordem. Dessa forma, numa árvore B de ordem m cada página contém no mínimo m registros (e $m+1$ descendentes) e no máximo $2m$ registros (e $2m+1$ descendentes), exceto a página raiz que pode conter entre 1 e $2m$ registros. Outra característica é que todas as páginas folhas aparecem no mesmo nível.

A figura 2.10 apresenta a forma geral de uma página de um árvore B de ordem *m*. Note que cada página possui no mínimo *m* e no máximo *2m* chaves e no mínimo *m+1* e no máximo *2m+1* apontadores para páginas descendentes.

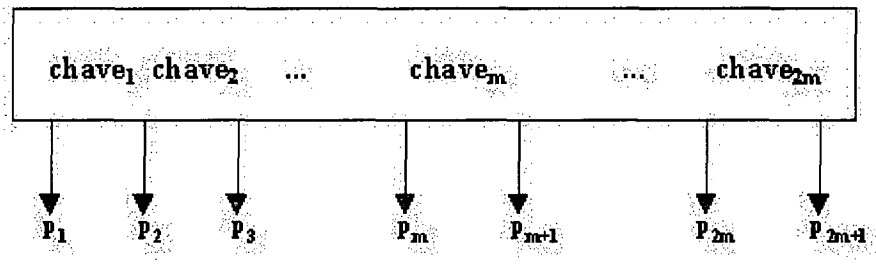

Figura 2.10 Página de uma árvore B de ordem *m*.

A figura 2.11 ilustra uma árvore B de ordem 2 que possui 3 níveis. Todas as páginas contém 2, 3 ou 4 registros, exceto a raiz que pode conter um único registro. Observa-se que as árvores B podem ser consideradas uma extensão natural da organização de uma árvore binária. Em uma página, as chaves aparecem em ordem crescente da esquerda para a direita. Considerando uma chave específica, um apontador à sua esquerda conduz a uma subárvore cujas chaves são todas menores e um apontador à direita conduz a uma subárvore com chaves maiores. Por exemplo, na figura 2.11 o apontador à direita da chave 20 e à esquerda da chave 35 refere-se uma página descendente cujas chaves estão entre 20 e 35.

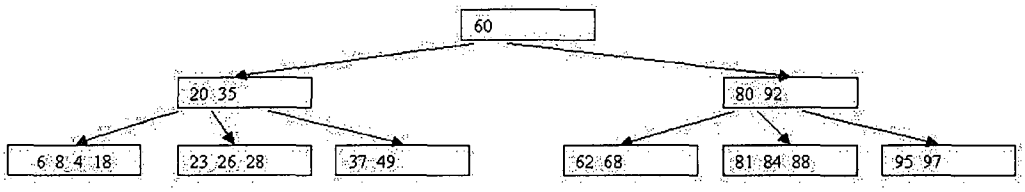

Figura 2.11 Exemplo de uma árvore B.

As principais características das árvores B são manter equilibrado o crescimento da árvore e permitir inserções e retiradas de forma balanceada, não existindo necessidade de procedimentos de reorganização da árvore.

Considerando características particulares de aplicações foram propostas diversas variações dessa estrutura de dados: árvores B^+, árvores B^*, entre outras [7].

2.5 Arquivos Estáticos e Dinâmicos

No estudo da organização de arquivos algumas considerações são necessárias, como por exemplo se um arquivo é estático ou dinâmico [19]. O arquivo será estático se o conjunto de registros é fornecido e raramente modificado. Será dinâmico se existirem inserções e remoções freqüentes de registros. Para o primeiro caso, é interessante procurar algoritmos eficientes para construção de árvores que minimizem o tempo médio da pesquisa. No outro caso é importante manter, sempre que possível, a árvore balanceada.

Outro ponto que deve ser salientado refere-se ao conhecimento ou não das probabilidades ou freqüências de acesso a todos os registros. Em geral, quando conhecidas, as freqüências de acesso obedecem a uma distribuição estatística não uniforme. Não sendo conhecidas as probabilidades, pressupõe-se que os registros são acessados com igual freqüência. É possível, através de uma abordagem intermediária, coletar dados sobre as freqüências de acesso enquanto o arquivo está em uso e com isso reestruturar a árvore em determinados momentos.

Outra questão relativa à organização de arquivos consiste em verificar se o conjunto de informações é mantido em memória principal de acesso aleatório ou se somente uma pequena parte dele pode ser mantida na memória principal em determinado instante, enquanto o restante deve ser mantido em armazenamento secundário.

O algoritmo proposto neste trabalho, apresentado no próximo capítulo, refere-se ao caso em que o arquivo é estático, as freqüências de acesso não são conhecidas e o armazenamento é secundário, pois o conjunto de registros não pode ser mantido integralmente em memória principal.

Capítulo 3

O Algoritmo Proposto

Este capítulo apresenta o algoritmo proposto para paginação de árvores binárias de pesquisa. O algoritmo é aplicável quando o conjunto de informações a serem tratadas é estático, as freqüências de acesso não são conhecidas e o armazenamento é remoto ou secundário. Além da descrição de funcionamento e especificação formal apresenta a análise de complexidade do algoritmo.

3.1 Funcionalidade do Algoritmo

Ao se paginarem árvores binárias de pesquisa existe um objetivo único, claro e muito bem definido a ser atingido. Este objetivo é expresso como o ideal de paginação [19] e foi ilustrado na figura 2.7.

O algoritmo proposto neste trabalho é um paginador de árvores binárias que visa aumentar o grau de parentesco dos nodos que serão armazenados numa mesma página. O algoritmo atinge o ideal de paginação quando a árvore binária é completa e o número de nodos é múltiplo do tamanho da página. Além disso, estabelece uma política eficiente de preenchimento de páginas, para a hipótese das árvores degeneradas.

Antes de descrever o algoritmo é necessário definir a noção de nodo *patriarca*. Um patriarca é um nodo raiz de uma subárvore que deverá ser alocado em uma nova página, juntamente com seus descendentes, tantos quantos couberem na página. Na figura 2.9 os nodos 30, 10, 23, 32 e 40 são patriarcas das páginas.

Considere que uma página armazena até x níveis ou gerações com relação a um nodo de uma árvore binária, onde x é um número natural. O algoritmo inicia pela raiz da árvore a ser paginada, considerando esta raiz o patriarca da primeira página. Em seguida, são armazenadas x-1 gerações do patriarca nesta página. Todos os nodos da geração seguinte não podem ser armazenados aí, pois não há espaço disponível. Estes nodos serão, cada um, o patriarca de uma nova página. Em cada uma destas páginas são alocadas x-1 gerações do patriarca, e assim sucessivamente, até que todos os nodos tenham sido armazenados adequadamente.

Quando a árvore é incompleta ou quando o número de nodos não é múltiplo de 2^x-1, sobram espaços em uma ou mais páginas. Por exemplo, a figura 3.1 apresenta uma árvore incompleta com 9 nodos. Neste trabalho a franja da árvore é definida como o conjunto de subárvores que não foram alocadas em nenhuma página e cujo número de nodos é menor que o tamanho da página. Toda subárvore da franja tem pelo menos uma folha. Na figura 3.1 a franja da árvore é composta pela subárvore *s1* de tamanho 2 contendo os nodos 10 e 12 e pela subárvore *s2* de tamanho 1 contendo o nodo 40.

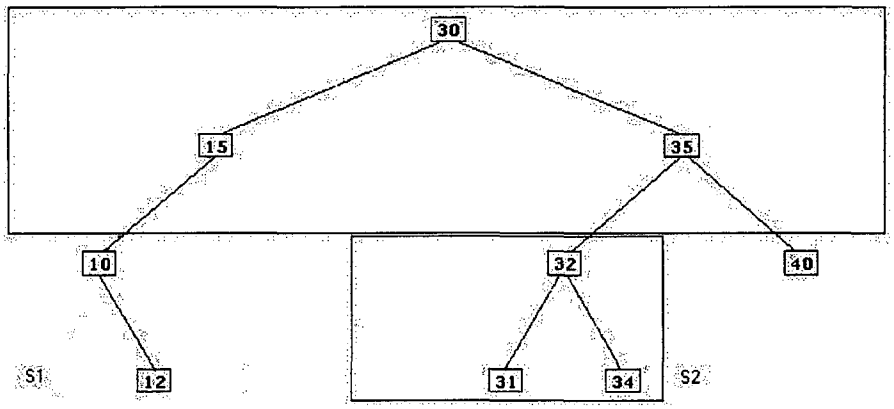

Figura 3.1 Árvore degenerada com franja em destaque.

O algoritmo adota uma política para armazenar as subárvores pertencentes à franja, aplicando um algoritmo de empacotamento unidimensional para alocar as subárvores em páginas.

O objetivo desta política é minimizar o número de páginas necessárias, procurando assegurar a proximidade dos nodos alocados em uma mesma página. Com isso, cada subárvore da franja é integralmente alocada em uma única página, pois,

desta forma, cada página é preenchida visando a otimização do número de páginas acessadas durante uma pesquisa na árvore binária. O problema do empacotamento unidimensional é descrito na próxima seção.

3.2 O Problema do Empacotamento Unidimensional

Para alocar as subárvores da franja estamos diante de um problema denominado empacotamento unidimensional [10, 11, 12].

O problema do empacotamento unidimensional pode ser definido como: dadas uma constante C e uma lista finita de itens $L = \{p_1, p_2,..., p_n\}$, onde cada item p, está associado a um valor $w(p_i)$ satisfazendo $0 < w(p_i) < C$, deseja-se encontrar o menor inteiro m tal que L possa ser particionada em m listas $L_1, L_2,...,L_m$ onde cada lista L_i, satisfaz $w(L_i) = \sum_{p_j \in L_i} w(p_j) \leq C$, $i = 1,...,m$. Num típico problema de empacotamento unidimensional deseja-se particionar uma lista de itens em sublistas de maneira a minimizar o número de partições respeitando a capacidade de cada sublista.

Neste trabalho deseja-se alocar um conjunto de subárvores (sem restrição de precedência) em um conjunto de páginas, conhecendo o espaço disponível nas páginas, de modo a minimizar o número de páginas utilizadas. As subárvores, com tamanhos $s_1, s_2,..., s_n$ devem ser alocadas em páginas de tamanho C. Um algoritmo de empacotamento unidimensional encontra a alocação que minimiza o número de páginas de tamanho C.

O problema de empacotamento unidimensional encontra-se entre os problemas clássicos de Otimização Combinatória, sendo considerado *NP*-difícil no sentido forte [12]. A única forma garantida de encontrar a solução exata para esses tipos de problemas seria a enumeração total das alternativas [33]. Essa forma de solucionar esses problemas pode ser considerada apenas para pequenas instâncias do problema, pois o tamanho do espaço de solução dos problemas combinatórios cresce de forma exponencial [34].

Para resolver o problema do empacotamento unidimensional existem diversos alternativas: algoritmos exatos, algoritmos aproximados, métodos de programação linear e não linear, algoritmos estocásticos, algoritmos analógicos, meta-heurísticas e combinação de algoritmos [10, 11]. Dentre as possíveis alternativas para a implementação do algoritmo proposto destaca-se a utilização de algoritmos aproximados ou heurísticos [10, 11, 12], ou seja, onde não existe a garantia de se encontrar a solução ótima, porém o tempo de execução é polinomial. Com isso, busca-se resolver o problema baseado em regras empíricas, cuja aplicação costuma ser dependente do tipo de problema. A maior parte da literatura sobre problemas de empacotamento unidimensional concentra-se nos algoritmos aproximados e seus respectivos desempenhos. Dentre os algoritmos aproximados destacam-se [10, 11, 12]: NF (*Next Fit*), FF (*First FIT*), BF (*Best Fit*), NFD (*Next Fit Decreasing*), FFD (*First Fit Decreasing*), BFD (*Best Fit Decreasing*), H_M (*Harmonic M*) e VL (*Vega and Lueker Algorithm*).

Na figura 3.2 apresentam-se exemplos de paginações da franja. Considere que as subárvores têm tamanhos $s_1 = 8$, $s_2 = 8$, $s_3 = 3$, $s_4 = 8$, $s_5 = 6$, $s_6 = 8$ e $s_7 = 6$, devem ser alocadas em páginas de tamanho 15. No primeiro exemplo de paginação, cinco páginas são necessárias, enquanto no empacotamento ideal apenas quatro páginas são necessárias.

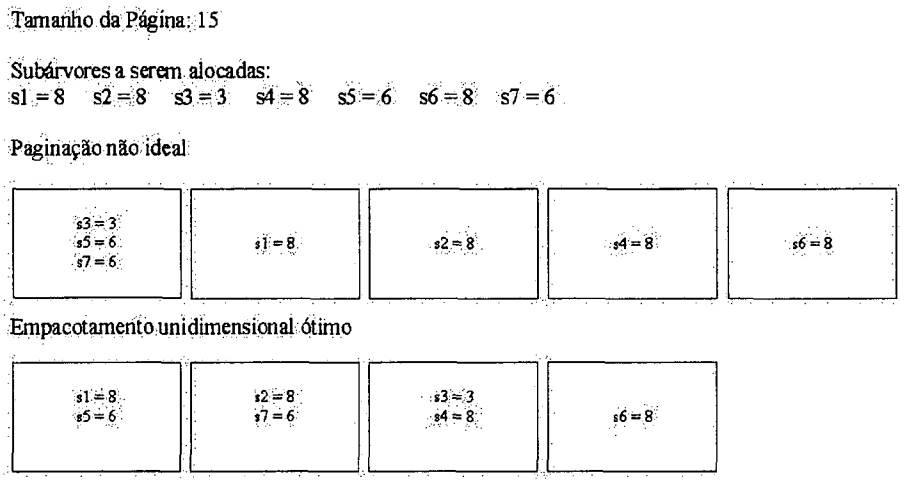

Figura 3.2 Exemplos de empacotamento unidimensional.

Na implementação realizada neste trabalho, descrita no capítulo 4, usou-se um algoritmo de aproximação guloso para solucionar o empacotamento das subárvores existentes na franja. Este algoritmo é bastante simples se comparado com outras soluções aproximadas disponíveis na literatura, entretanto foi escolhido tendo em vista o fato de que resolve o problema do empacotamento de maneira adequada considerando os resultados obtidos para o conjunto de experimentos executados.

3.3 Especificação do Algoritmo

Dentre os objetivos a serem alcançados na paginação de árvores binárias destacam-se a otimização do tempo de pesquisa e a minimização do espaço de armazenamento. Quando se insere em uma página o nodo patriarca e suas (x-1) gerações, busca-se otimizar o tempo de pesquisa. Ao se preencher uma página que está incompleta, objetiva-se tanto a minimização do espaço de armazenamento, quanto a otimização do tempo de acesso, no sentido que se mantém em uma mesma página todos os elementos de uma subárvore.

O algoritmo proposto atinge o ideal de paginação quando tal ideal existe para a árvore em processamento, nos demais casos uma política para tratamento de árvores degeneradas é adotada.

O algoritmo utiliza estruturas de dados que são descritas a seguir. A estrutura de cada nodo é ilustrada na figura 3.3. Um nodo é um registro que, no campo dados, armazena qualquer tipo de dados, definido pelo usuário de acordo com a aplicação. O campo chave permite a identificação única do nodo. O nodo deve conter também os endereços de seus filhos da esquerda e da direita. Um endereço é constituído por dois campos: página em que se encontra e posição dentro desta. No caso de não existir um dos filhos convenciona-se um valor nulo correspondente. Além disso, o registro armazena o tamanho da subárvore de que é patriarca.

Dados	
Chave	
Tamanho da subárvore	
Página filho esquerda	Página filho direita
Posição filho esquerda	Posição filho direita

Figura 3.3 A estrutura de um nodo.

Considerando uma árvore binária completa com $2^x - 1$ nodos, esta árvore possui x níveis ou gerações, com relação à raiz da árvore. De forma análoga, subárvores completas ou incompletas também possuem gerações de descendentes. Por exemplo, a figura 3.4 ilustra uma árvore binária incompleta, os nodos 12, 26 e 42 pertencem à segunda geração com relação ao nodo 35, que é a raiz da árvore. Por sua vez, os nodos 12 e 26 pertencem à primeira geração com relação à subárvore com raiz no nodo 20.

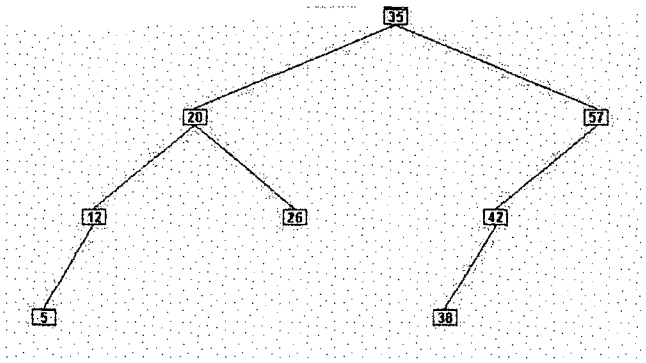

Figura 3.4 Uma árvore binária e suas gerações.

Para armazenar os nodos de uma árvore binária é necessário que uma página seja formada por $2^x - 1$ elementos, sendo x um número inteiro positivo, que indica o número de gerações de uma subárvore. Na figura 3.5, a página armazena 7 nodos, ou seja, $x = 3$ gerações. Na paginação obtida após o processamento, o ideal é que em cada página o pai aponte para o seu filho que está na mesma página, conforme ilustrado na figura 3.5.

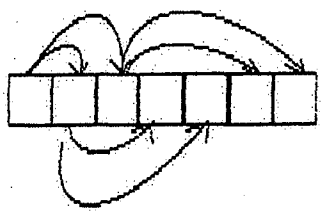

Figura 3.5 A estrutura de uma página ideal.

Em uma página ideal, o primeiro nodo, dito patriarca, tem exatamente suas próximas *(x-1)* gerações na sua página, de forma que a página tem um total de *x* gerações. Para preencher as outras páginas cada um dos elementos da geração *(x+1)*-ésima será tratado exatamente como um novo patriarca.

O algoritmo proposto utiliza duas estruturas de dados chamadas *SQ* e *FL*, descritas a seguir. A estrutura de dados *SQ*, iniciais de *Stack* e *Queue*, comporta-se ora como uma pilha, ora como uma fila. A estrutura de dados denominada *FL* (*Fringe List*) é uma lista encadeada que armazena as raízes das subárvores pertencentes à franja da árvore.

Considere que uma lista linear é um conjunto $A=(a_1, a_2,..., a_n)$ com $n >= 0$ elementos. Sua principal propriedade estrutural envolve as posições relativas dos elementos em uma dimensão. Assumindo $n >= 1$, a_1 é o primeiro elemento da lista e a_n é o último elemento da lista. Temos que a_i, com $1 <= i <= n$, referencia um elemento pertencente à i-ésima posição do conjunto. A lista linear é denominada nula ou vazia quando tem $n=0$ elementos. Além disso, sobre a lista linear são definidas operações que permitem a inserção e a retirada de elementos.

A estrutura de dados *Fringe List* pode ser considerada uma lista linear $FL=(a_1, a_2,..., a_n)$ sobre a qual são possíveis as seguintes operações:

- criar(*FL*), que cria *FL* como uma estrutura de dados vazia;
- tamanho(*FL*), que devolve o comprimento *n* de *FL;*
- inserir (*x*, *i*, *FL*), que insere o valor *x* na posição *i*, $1 <= i <= n+1$, alterando a numeração dos elementos de $i,i+1,...,n$ para $i+1,i+2,...,n+1$, devolvendo a estrutura *FL* resultante;
- remover (*i*, *FL*), que retira o elemento da posição *i* de *FL*, devolvendo-o juntamente com a estrutura *FL* resultante;
- ler (*i*, *FL*), que devolve o valor da posição *i;*
- vazia(*FL*), que devolve a condição verdadeira se *FL* for vazia, e em caso contrário, devolve a condição falsa;

Por sua vez, a estrutura de dados *Stack Queue* pode ser considerada uma lista linear $SQ=(a_1, a_2,..., a_n)$ na qual são possíveis inserções e retiradas em uma

extremidade, chamada *traseira* ou *topo* e retiradas em outra extremidade, dita *frontal*. Sobre a estrutura *SQ* podem ser realizadas diversas operações:

- criar(*SQ*), que cria *SQ* como uma estrutura de dados vazia;
- enfileirar(*x*, *SQ*), que acrescenta o elemento *x* na traseira da estrutura *SQ*, devolvendo a estrutura *SQ* resultante;
- desenfileirar(*SQ*), que retira o elemento frontal de *SQ*, devolvendo-o juntamente com a estrutura *SQ* resultante;
- desempilhar (*SQ*), que retira o elemento do topo de *SQ*, devolvendo-o juntamente com a estrutura *SQ* resultante;
- vazia(*SQ*), que devolve a condição verdadeira se *SQ* for vazia, e em caso contrário, devolve a condição falsa;

Considerando as propriedades e operações que podem ser realizadas sobre as estruturas de dados *SQ* e *FL*, a seguir descreve-se o comportamento do algoritmo. Inicialmente, a raiz da árvore é enfileirada na *SQ*. Toda vez que se inicia o preenchimento de uma nova página, desenfileira-se o primeiro elemento da *SQ*, que se transforma no patriarca da página a ser preenchida. Todos os elementos das próximas *x-1* gerações do patriarca são então armazenados na página. Sobrando espaço na página, os elementos da geração seguinte que sejam patriarcas de subárvores maiores ou iguais ao espaço disponível na página são enfileirados na *SQ*. Os demais elementos são inseridos na *FL*, pois correspondem às subárvores da franja. Não sobrando espaços na página os elementos da geração seguinte que sejam patriarcas de subárvores maiores que o tamanho da página são enfileirados na *SQ*, os demais inseridos na *FL*.

Caso a página tenha espaço disponível, o algoritmo procede da seguinte maneira: o último elemento inserido na *SQ* é desempilhado e armazenado na página. Os seus filhos são enfileirados na *SQ*. Se ainda houver espaço, e a *SQ* não estiver vazia, mais uma vez o último elemento inserido na *SQ* é desempilhado e armazenado na página, e seus filhos são enfileirados na *SQ*. O processo se repete até que a página esteja preenchida, ou a *SQ* esteja vazia.

Quando a página já está completa, isto é, sem espaços sobrando, o algoritmo inicia o preenchimento de uma nova página, desenfileirando seu patriarca da *SQ*, como

descrito acima. Quando não há mais elementos na *SQ*, resta ao algoritmo proceder à alocação das subárvores pertencentes à franja da árvore. Na franja da árvore existem grupos de nodos que ainda devem ser paginados e espaços sobrando em páginas, dessa forma objetiva-se alocá-los mantendo-os juntos e otimizando o uso de espaços disponíveis. Para isso propõe-se a utilização de algoritmos de empacotamento unidimensional descritos anteriormente. O algoritmo termina quando todos os nodos da árvore estão armazenados em páginas, situação refletida por *FL* vazia.

O algoritmo proposto é especificado em alto nível na figura 3.6.

Algoritmo de Paginação de Árvores Binárias

Considere o tamanho da página 2^x-1
início
criar(*SQ*);
enfileirar(raiz da árvore, *SQ*);
repetir
criar uma nova página;
patriarca ← desenfileirar(*SQ*);
alocar na página corrente o patriarca e seus descendentes dos próximos $x-1$ níveis;
se (página está completamente preenchida)
então
para todo nodo pertencente ao nível subsequente:
se (nodo é raiz de subárvore maior ou igual ao tamanho da página)
então enfileirar(nodo, *SQ*);
senão inserir(nodo, *FL*);
senão /* existe espaço disponível na página corrente */
para todo nodo pertencente ao nível subsequente:
se (nodo é raiz de subárvore maior ou igual ao espaço disponível na página)
então enfileirar(nodo, *SQ*);
senão inserir(nodo, *FL*);
enquanto (existe espaço disponível na página corrente) ***e*** (***não*** vazia(*SQ*)) ***faça***
nodo ← desempilhar(*SQ*);
alocar nodo na página corrente;
para todo filho do nodo alocado:
enfileirar(filho, *SQ*);
fim enquanto
até vazia(*SQ*);
se (**não** vazia(*FL*))
então
aplicar empacotamento unidimensional para paginar as subárvores da franja;
fim.

Figura 3.6 O algoritmo proposto.

3.4 Exemplos de Execução

Nesta seção iremos observar alguns exemplos práticos do algoritmo proposto. A figura 3.7 mostra a árvore binária obtida pela entrada das chaves: 25, 30, 33, 20, 27, 12, 32, 10 e 7.

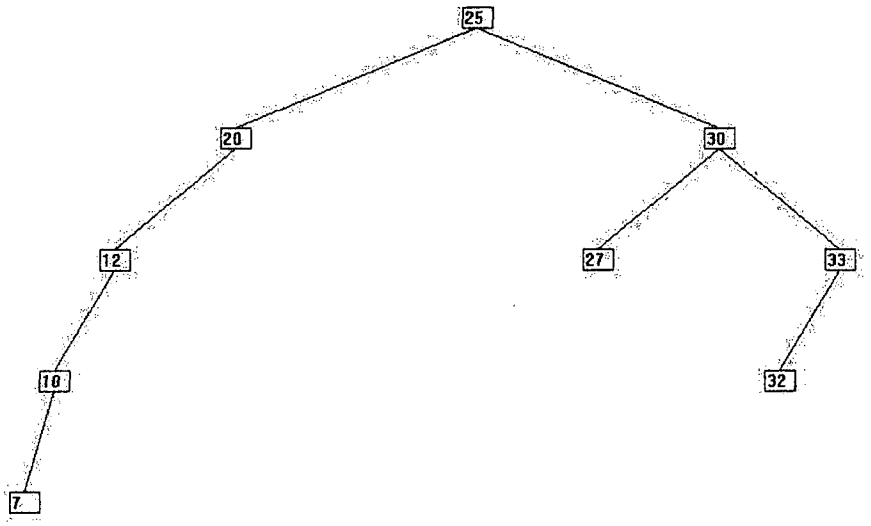

Figura 3.7 Árvore binária degenerada de 5 níveis.

A paginação seqüencial obtida, para $x = 2$ (níveis por página) e tamanho da página 3, é descrita na tabela 3.1:

Página	Posição	Chave	Esq	Dir
0	0	25	1-0	0-1
0	1	30	1-1	0-2
0	2	33	2-0	Nulo
1	0	20	1-2	Nulo
1	1	27	Nulo	Nulo
1	2	12	2-1	Nulo
2	0	32	Nulo	Nulo
2	1	10	2-2	Nulo
2	2	7	Nulo	Nulo

Tabela 3.1 Exemplo de paginação seqüencial de árvore binária.

Nesse exemplo, o algoritmo de paginação atua da seguinte forma: inicialmente a raiz é inserida na *SQ*. Cria-se a página 0. Desenfileira-se o nodo 25 de SQ e o mesmo é armazenado na página junto com suas (x-1) gerações, ou seja, os nodos 20 e 30. O nodo 12, elemento da próxima geração, é enfileirado na *SQ*, enquanto os nodos 27 e 33 são inseridos na *FL* Como não existe mais espaço nessa página, a página 1 é criada. Desenfileira-se o nodo 12 de SQ e o mesmo é armazenado na página junto com suas (x-1) gerações, ou seja, o nodo 10. O nodo 7, único elemento da próxima geração, é inserido na *SQ*. Como a página ainda não está completa, utilizaremos *SQ* com o papel de uma pilha, onde potencialmente seus últimos elementos guardam proximidade de parentesco com os elementos que estão na página incompleta. Dessa forma, desempilha-se o nodo 7 de *SQ* e armazena-se na página. Como *SQ* é vazia o algoritmo prossegue para alocar as subárvores existentes na *FL*. Como não existe mais espaço nessa página, a página 2 é criada. Aplicando-se um algoritmo de empacotamento unidimensional a subárvore formada pelo nodo 27 e a subárvore formada pelos nodos 33 e 32 são retiradas de *FL* e alocadas na página. O algoritmo encerra a sua execução, pois *FL* é vazia. Ao final obtém-se a paginação descrita na tabela 3.2:

Página	Posição	Chave	Esq	Dir
0	0	25	0-1	0-2
0	1	20	1-0	Nulo
0	2	30	2-0	2-1
1	0	12	1-1	Nulo
1	1	10	1-2	Nulo
1	2	7	Nulo	Nulo
2	0	27	Nulo	Nulo
2	1	33	2-2	Nulo
2	2	32	Nulo	Nulo

Tabela 3.2 Exemplo de paginação após o processamento do algoritmo.

Como explicado no capitulo 1, quanto menor o número de páginas acessadas para manipular a árvore melhor. Considerando o exemplo da figura 3.7, o número total de páginas visitadas na busca de todos os elementos da árvore é de 17, enquanto que

usando o algoritmo proposto este número fica reduzido para 15. O ganho aumenta com o tamanho da árvore, até 50% como será mostrado no capítulo 4, que contém resultados experimentais exaustivos de comparação das estratégias de paginação.

3.5 Complexidade do Algoritmo de Paginação Proposto

A análise da complexidade do algoritmo proposto será realizada levando-se em consideração os dois trechos principais. O primeiro aloca em páginas toda a árvore com exceção da franja, enquanto o segundo realiza o empacotamento unidimensional nas subárvores da franja. Na análise apresentada será considerada a ordem de tempo de execução e não o valor exato da função de complexidade.

Analisando-se a fase inicial do algoritmo, considera-se f uma função de complexidade tal que $f(n)$ é o número de registros acessados no arquivo, isto é, o número de vezes que um nodo da árvore é consultado. Considere que o pior caso corresponde ao maior tempo de execução sobre todas as entradas de tamanho n. Considere, ainda, que f é uma função de complexidade baseada na análise de pior caso, com isso o custo de aplicar o algoritmo nunca é maior do que $f(n)$.

Observa-se que a função de complexidade f da fase inicial do algoritmo é $f(n) = 4n$. Tal linearidade pode ser aferida analisando-se os procedimentos realizados. Para calcular o tamanho de todas as subárvores são necessários n acessos aos elementos. Para alocar todos os nodos da árvore são necessários $2n$ acessos, dos quais n acessos para a alocação do nodo em si e n acessos para se estabelecer a vinculação entre nodo pai e nodo filho. Finalmente, para processar, durante a fase inicial do algoritmo, as estruturas de dados SQ e FL são necessários no máximo n acessos a elementos

A fase final do algoritmo depende fundamentalmente do algoritmo para empacotamento unidimensional aplicado às subárvores da franja. Considere g a função de complexidade da fase final do algoritmo tal que $g(n)$ descreve o número de acessos a nodos para alocação das subárvores da franja. A literatura reporta algoritmos aproximados com funções de complexidade quadrática, ou seja, $g(n) = kn^2$, onde k é uma constante.

Tomando-se o conceito de dominação assintótica [2, 35] têm-se que para o trecho inicial do algoritmo $O(f(n)) = O(n)$ e para o trecho final é $O(g(n)) = O(n^2)$. Com isso, demonstra-se que a complexidade do algoritmo é quadrática, pois $O(f(n)) + O(g(n))$ é $O(n^2)$.

3.6 Uma Versão Alternativa do Algoritmo

Um versão alternativa do algoritmo proposto é apresentada, consistindo em uma simplificação do mesmo e possuindo complexidade linear [24]. Tal versão alternativa considera preponderantemente a necessidade de proximidade de parentesco dos nodos existentes em uma mesma página, ou seja, objetiva-se diminuir a distância internodal existente nas páginas. Para essa versão a diminuição do número de páginas visitadas em pesquisas é um objetivo secundário. Em aplicações que necessitem de proximidade de vizinhança dos nodos da árvore tal versão apresenta bons resultados.

Inicialmente, a raiz da árvore é enfileirada na *SQ*. Toda vez que se inicia o preenchimento de uma nova página, desenfileira-se o primeiro elemento da *SQ*, que se transforma no patriarca da página a ser preenchida. Todos os elementos das x-1 gerações do patriarca são então armazenados na página. Sobrando espaço na página, os elementos da geração seguinte são enfileirados na *SQ*. Caso a página tenha espaço disponível, o algoritmo procede da seguinte maneira: o último elemento inserido na *SQ* é desempilhado e armazenado na página. Os seus filhos são enfileirados na *SQ*. Se ainda houver espaço, e a *SQ* não estiver vazia, mais uma vez o último elemento inserido na *SQ* é desempilhado e armazenado na página, e seus filhos são enfileirados na *SQ*. O processo se repete até que a página esteja preenchida, ou a *SQ* esteja vazia.

Quando a página já está completa, isto é, sem espaços sobrando, o algoritmo inicia o preenchimento de uma nova página, desenfileirando seu patriarca da *SQ*, como descrito acima. O algoritmo termina quando todos os nodos da árvore estão armazenados em páginas, situação refletida por *SQ* vazia.

A especificação da versão alternativa do algoritmo para a paginação de uma árvore binária de pesquisa é descrita em alto nível na figura 3.8. Nessa versão o

algoritmo não utiliza a estrutura *Fringe List* e apenas aloca os nodos com auxílio da *Stack-Queue*.

Versão Alternativa do Algoritmo de Paginação de Árvores Binárias

Considere o tamanho da página 2^x-*1*
início
criar(*SQ*);
enfileirar(raiz da árvore, *SQ*);
repetir
criar uma nova página;
patriarca ← desenfileirar(*SQ*);
alocar na página corrente o patriarca e seus descendentes dos próximos *x-1* níveis;
enquanto (existe espaço disponível na página corrente) ***e*** (***não*** vazia(*SQ*)) ***faça***
nodo ← desempilhar(*SQ*);
alocar nodo na página corrente;
para todo filho do nodo alocado:
enfileirar(filho, *SQ*);
fim enquanto
até vazia(*SQ*);
fim.

Figura 3.8 Versão alternativa do algoritmo.

Capítulo 4

Implementação e Resultados Experimentais

Este capítulo discute a implementação do algoritmo proposto e apresenta os resultados experimentais obtidos. São descritas as métricas utilizadas para avaliar o desempenho do algoritmo, bem como são apresentados os resultados empiricamente obtidos. Comparações com outras estratégias também são apresentadas, incluindo paginação seqüencial, paginação ótima teórica e árvores B.

4.1 A Implementação Realizada

O algoritmo proposto foi codificado na linguagem Pascal [37]. O código fonte dos programas utilizados encontra-se nos anexos 1, 2 e 3. O algoritmo proposto foi implementado sob a forma de um pós-processador de paginação. Inicialmente é gerada uma árvore paginada seqüencialmente, em seguida o algoritmo é aplicado produzindo uma paginação próxima da ótima. Com isso, foi possível comparar a paginação obtida pelo algoritmo com o resultado produzido pela paginação seqüencial. Além disso, os resultados obtidos também foram comparados com valores ótimos teóricos, considerando a possibilidade de realizar o balanceamento da árvore e posterior alocação em páginas. Além disso, realizaram-se experimentos comparativos com as árvores B.

Para analisar o desempenho do algoritmo, foram realizados experimentos com seqüências de chaves geradas aleatoriamente. Os experimentos foram divididos em função do tamanho das páginas. Tendo em vista que o objetivo é paginar árvores

binárias, consideraram-se como parâmetros para o tamanho da página os valores 3, 7 e 15, múltiplos de subárvores completas de 1, 2 e 3 níveis respectivamente.

As árvores utilizadas nos experimentos foram geradas aleatoriamente, possuindo de 10 a 2000 nodos em intervalos de 10 nodos. Foram realizados 100 experimentos para cada tamanho de árvore. Ao todo foram geradas 20000 árvores utilizando funções de geração de número aleatórios do próprio ambiente de programação.

A tabela 4.1 apresenta as quantidades de nodos das árvores e o número de árvores utilizadas nos experimentos.

Tamanho da Página	Quantidade de nodos na árvore	Número de árvores testadas
3	10 a 400	4000
7	10 a 900	9000
15	10 a 2000	40000

Tabela 4.1 Indicadores da medição empírica realizada.

4.2 As Métricas Utilizadas

Para analisar o desempenho do algoritmo proposto, foram utilizadas três métricas: número de páginas visitadas, distância internodal e quantidade de espaços não utilizados nas páginas.

O número de páginas visitadas consiste na quantidade de páginas que são acessadas para a pesquisa a um nodo qualquer da árvore. A métrica utilizada calcula a quantidade de páginas visitadas para pesquisar *todos* os nodos da árvore. Por exemplo na figura 4.1 o número total de páginas visitadas é 17, pois para pesquisar os nodos 25, 30 e 33, somente uma página é visitada, para pesquisar os nodos 12, 20, 27 e 32, duas páginas são visitadas e para pesquisar os nodos 7 e 10 três páginas são visitadas.

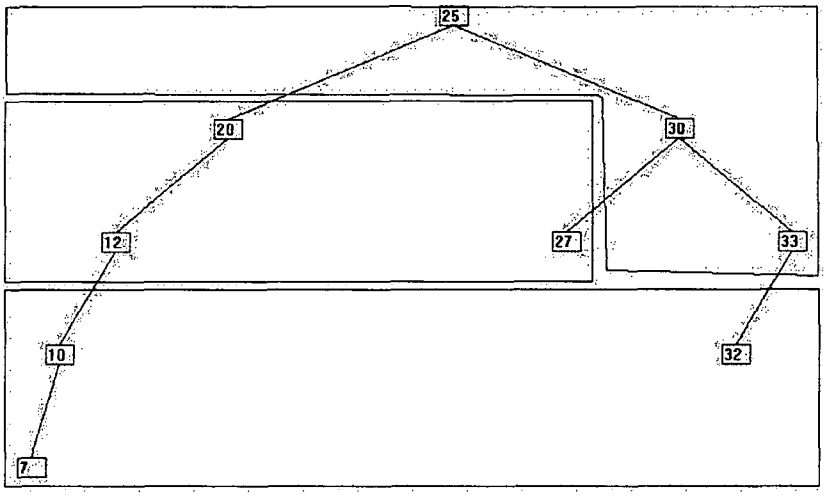

Figura 4.1 Árvore exemplo para as métricas utilizadas.

A distância internodal consiste na distância entre dois nodos de uma árvore. Com isso, avalia-se o grau de proximidade dos nodos dentro de uma mesma página, atributo relevante para aplicações que exijam que nodos próximos sejam armazenados em uma mesma página. A métrica utilizada consistiu em mensurar todas as distâncias internodais existentes em uma página e computar a soma do resultado obtido em todas as páginas. Por exemplo, na figura 4.1 a distância internodal total é 26, sendo formada pelas distâncias: 4 (página formada pelos nodos 25, 30 e 33, distâncias 25-30, 25-33 e 30-33), 8 (página formada pelos nodos 20, 12 e 27) e 14 (página formada pelos nodos 10, 7 e 32).

A quantidade de espaços não utilizados produzidos nas páginas é outra métrica utilizada e consiste em aferir quantas posições das páginas não estão sendo ocupadas por nenhum nodo da árvore.

4.3 Ganho Médio no Número de Páginas Visitadas

Na primeira medição empírica realizada foi considerada a métrica do número de páginas visitadas. O desempenho do algoritmo foi comparado à paginação seqüencial, utilizada para representação do pior caso ou limite superior.

A tabela 4.2 apresenta os ganhos médios do algoritmo e de sua versão alternativa em relação à paginação seqüencial quanto ao número de páginas visitadas.

Tamanho da Página	Ganho Médio do Algoritmo	Ganho Médio da Versão Alternativa do Algoritmo
3	32,24 %	29,73 %
7	43,64 %	41,13 %
15	49,64 %	47,33 %

Tabela 4.2 Ganho médio quanto ao número de páginas visitadas.

O gráfico da figura 4.2 ilustra a evolução do desempenho do algoritmo em árvores com 10 a 400 nodos, quanto à métrica páginas visitadas, sendo considerado o tamanho da página 3 nodos. Nesse experimento foi observado o ganho médio de 32,24% do algoritmo proposto em relação à paginação seqüencial.

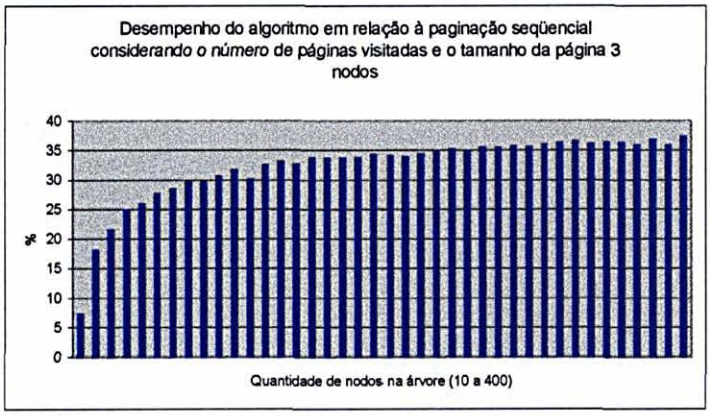

Figura 4.2 Ganho médio de páginas visitadas com tamanho de página 3 nodos.

O gráfico da figura 4.3 ilustra a evolução do desempenho da versão alternativa do algoritmo em árvores com 10 a 400 nodos, quanto à métrica páginas visitadas, sendo considerado o tamanho da página 3 nodos. Nesse experimento foi observado o ganho médio de 29,73% da versão alternativa do algoritmo proposto em relação à paginação seqüencial.

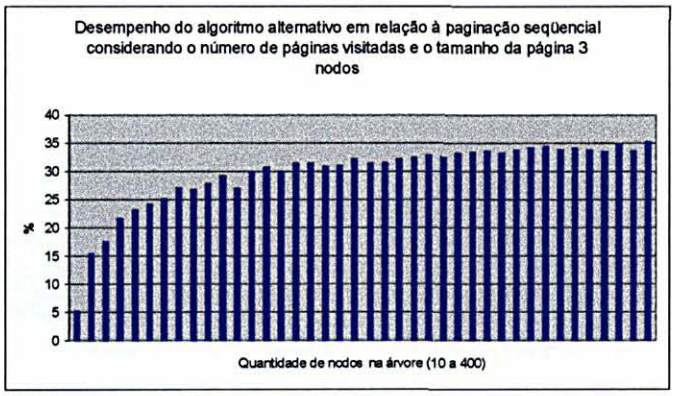

Figura 4.3 Ganho médio de páginas visitadas da versão alternativa do algoritmo com tamanho de página 3 nodos.

O gráfico da figura 4.4 ilustra a evolução do desempenho do algoritmo em árvores com 10 a 900 nodos, quanto à métrica páginas visitadas, sendo considerado o tamanho da página 7 nodos. Nesse experimento foi observado o ganho médio de 43,64% do algoritmo proposto em relação à paginação seqüencial.

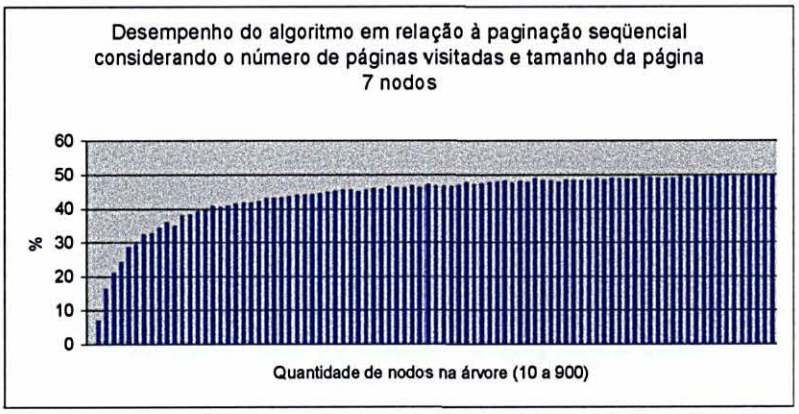

Figura 4.4 Ganho médio de páginas visitadas com tamanho de página 7 nodos.

O gráfico da figura 4.5 ilustra a evolução do desempenho da versão alternativa do algoritmo em árvores com 10 a 900 nodos, quanto à métrica páginas visitadas, sendo considerado o tamanho da página 7 nodos. Nesse experimento foi observado o ganho médio de 41,13% da versão alternativa do algoritmo em relação à paginação seqüencial.

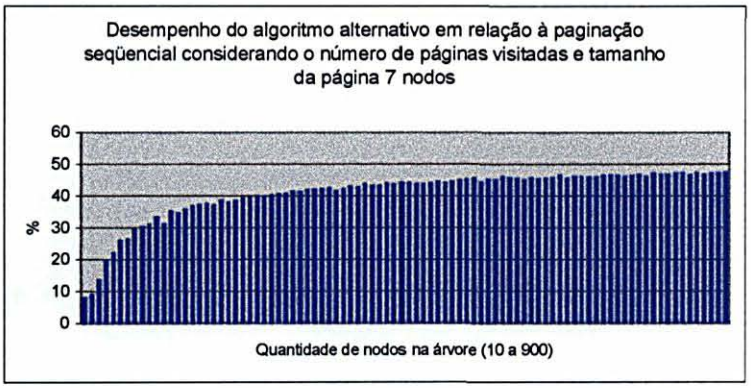

Figura 4.5 Ganho médio de páginas visitadas da versão alternativa do algoritmo com tamanho de página 7 nodos.

O gráfico da figura 4.6 ilustra a evolução do desempenho do algoritmo em árvores com 10 a 2000 nodos, quanto à métrica páginas visitadas, sendo considerado o tamanho da página 15 nodos. Nesse experimento foi observado o ganho médio de 49,67% do algoritmo proposto em relação à paginação seqüencial.

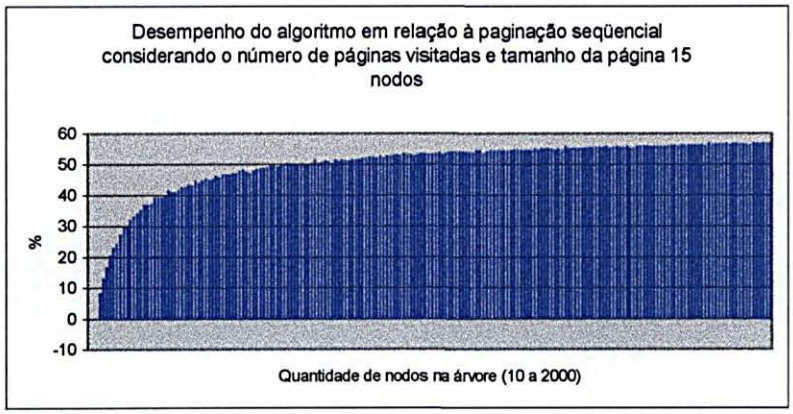

Figura 4.6 Ganho médio de páginas visitadas com tamanho de página 15 nodos.

O gráfico da figura 4.7 ilustra a evolução do desempenho da versão alternativa do algoritmo em árvores com 10 a 2000 nodos, quanto à métrica páginas visitadas, sendo considerado o tamanho da página 15 nodos. Nesse experimento foi observado o ganho médio de 47,33% da versão alternativa do algoritmo proposto em relação à paginação seqüencial.

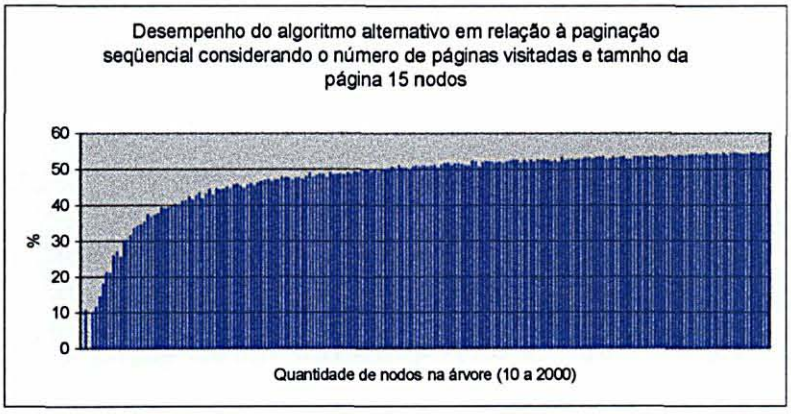

Figura 4.7 Ganho médio de páginas visitadas da versão alternativa do algoritmo com tamanho de página 15 nodos.

Pelos resultados apresentados pode-se concluir que o algoritmo produz menor número de páginas visitadas em comparação aos resultados tanto da versão alternativa quanto da paginação seqüencial. A diferença constatada nos resultados das versões do algoritmo proposto se referem ao ganho obtido com o uso do empacotamento unidimensional para a paginação da franja.

4.4 Ganho Médio na Distância Internodal

Na segunda medição empírica realizada foi considerada a métrica da distância internodal. Novamente o desempenho do algoritmo foi comparado à paginação seqüencial, utilizada como representação do pior caso ou limite superior.

A tabela 4.3 apresenta os ganhos médios do algoritmo e de sua versão alternativa em relação à paginação seqüencial quanto à distância internodal.

Tamanho da Página	Ganho Médio do Algoritmo	Ganho Médio da Versão Alternativa do Algoritmo
3	65,63 %	59,42 %
7	53,92 %	35,91 %
15	42,89 %	53,19 %

Tabela 4.3 Ganho médio quanto à distância internodal.

O gráfico da figura 4.8 ilustra a evolução do desempenho do algoritmo em árvores com 10 a 400 nodos, quanto à métrica distância internodal, sendo considerado o tamanho da página 3 nodos. Nesse experimento foi observado o ganho médio de 65,63% do algoritmo proposto em relação à paginação seqüencial.

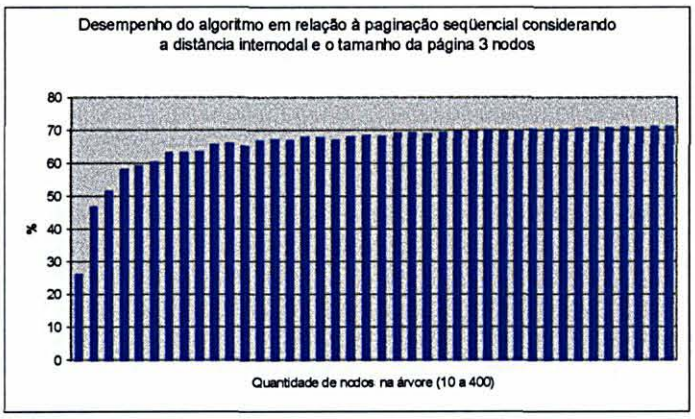

Figura 4.8 Ganho médio de distância internodal com tamanho de página 3 nodos.

O gráfico da figura 4.9 ilustra a evolução do desempenho da versão alternativa do algoritmo em árvores com 10 a 400 nodos, quanto à métrica distância internodal, sendo considerado o tamanho da página 3 nodos. Nesse experimento foi observado o ganho médio de 59,42% da versão alternativa do algoritmo em relação à paginação seqüencial.

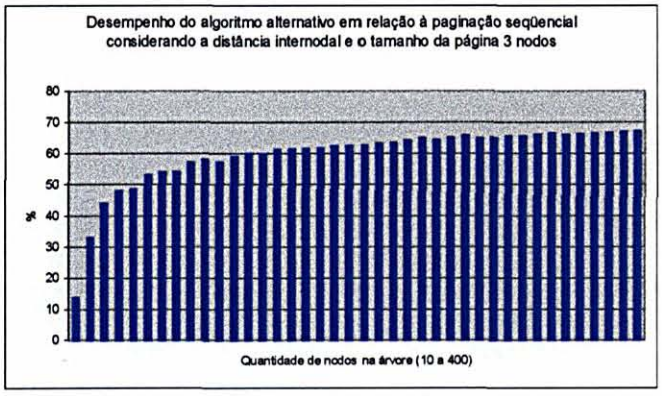

Figura 4.9 Ganho médio de distância internodal da versão alternativa do algoritmo com tamanho de página 3 nodos.

O gráfico da figura 4.10 ilustra a evolução do desempenho do algoritmo em árvores com 10 a 900 nodos, quanto à métrica distância internodal, sendo considerado o tamanho da página 7 nodos. Nesse experimento foi observado o ganho médio de 53,92% do algoritmo proposto em relação à paginação seqüencial.

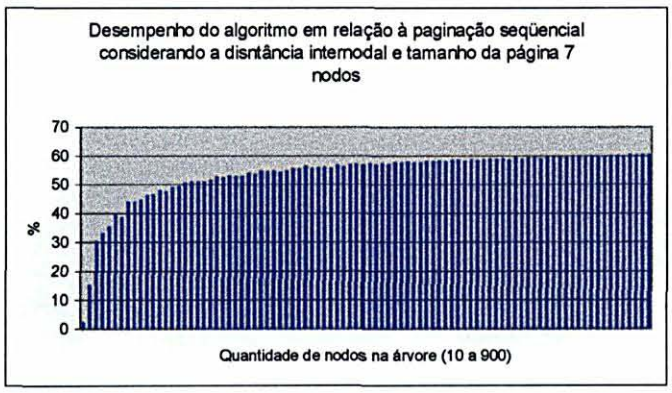

Figura 4.10 Ganho médio de distância internodal com tamanho de página 7 nodos.

O gráfico da figura 4.11 ilustra a evolução do desempenho da versão alternativa do algoritmo em árvores com 10 a 900 nodos, quanto à métrica distância internodal, sendo considerado o tamanho da página 7 nodos. Nesse experimento foi observado o ganho médio de 55,91% da versão alternativa do algoritmo em relação à paginação seqüencial.

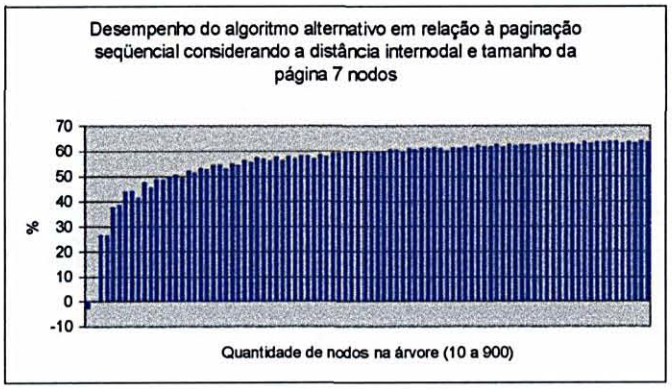

Figura 4.11 Ganho médio de distância internodal da versão alternativa do algoritmo com tamanho de página 7 nodos.

O gráfico da figura 4.12 ilustra a evolução do desempenho do algoritmo em árvores com 10 a 2000 nodos, quanto à métrica distância internodal, sendo considerado o tamanho da página 15 nodos. Nesse experimento foi observado o ganho médio de 42,89% do algoritmo proposto em relação à paginação seqüencial.

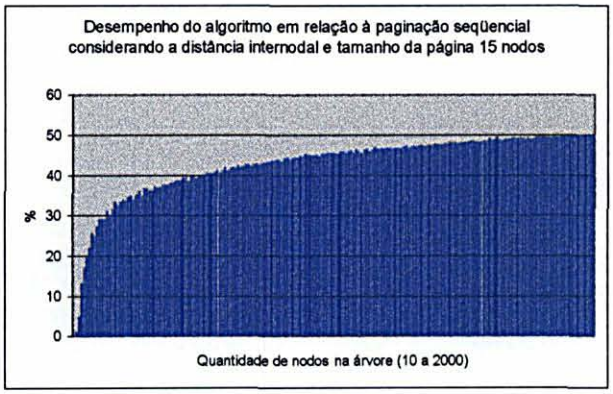

Figura 4.12 Ganho médio de distância internodal com tamanho de página 15 nodos.

O gráfico da figura 4.13 ilustra a evolução do desempenho da versão alternativa do algoritmo em árvores com 10 a 2000 nodos, quanto à métrica distância internodal, sendo considerado o tamanho da página 15 nodos. Nesse experimento foi observado o ganho médio de 53,19% da versão alternativa do algoritmo em relação à paginação seqüencial.

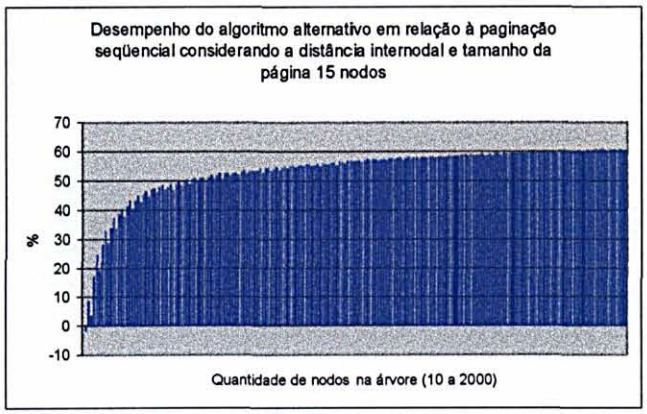

Figura 4.13 Ganho médio de distância internodal da versão alternativa do algoritmo com tamanho de página 15 nodos.

Pelos resultados apresentados pode-se concluir que a versão alternativa do algoritmo produz páginas com menor distância internodal, o que é explicado pelo fato da paginação da franja da árvore ser feita seqüencialmente enquanto que no algoritmo proposto o resultado do empacotamento unidimensional pode levar à inclusão de subárvores distantes em uma mesma página.

4.5 Comparação da Distância Internodal com Valores Ótimos Teóricos e com a Paginação Seqüencial

Outra medição empírica permitiu a comparação do desempenho do algoritmo e de diferentes estratégias de paginação com um valor ótimo teórico, utilizado como representação do melhor caso ou *limite inferior*. O valor ótimo teórico é obtido considerando a existência de um algoritmo que realize o balanceamento e ajuste da árvore no menor número de páginas possível. Portanto, o valor ótimo teórico considera que a árvore a ser paginada não possui qualquer degeneração. Esses valores teóricos são obtidos com simples cálculo matemático em função da quantidade de nodos e do tamanho da página. Por exemplo, para uma árvore com 530 nodos e tamanho da página 7, têm-se que a distância internodal mínima é 3618, pois são produzidas 75 páginas com subárvores completas de 7 nodos e uma página com uma subárvore completa de 5 nodos. Como a distância internodal existente numa subárvore completa de 7 nodos é 48 e numa subárvore completa de 5 nodos é 18, temos que a distância internodal ótima teórica para árvores com essa dimensão é $75*48 + 18=3618$. De modo análogo, calcula-se o número de páginas visitadas ótimo teórico. Considerando os níveis da árvore paginada, por exemplo, para uma árvore com 530 nodos têm-se no primeiro nível 1 página completa, logo, para 7 nodos apenas uma página é visitada; no segundo nível 8 páginas completas, logo, para 56 nodos duas páginas são visitadas; no terceiro nível 64 páginas completas, logo, para 448 nodos três páginas são visitadas; e 2 páginas completas e 1 página incompleta no quarto nível, com isso, para 19 nodos quatro páginas são visitadas. Daí têm-se que o número de páginas visitadas ótimo

teórico para uma árvore com 530 nodos em páginas com 7 nodos é *7*1 + 56*2 + 448*3 + 19*4=1539.*

Na tabela 4.4 observam-se as quantidades médias de distância internodal obtidas nas diferentes estratégias.

Estratégia	Quantidades médias de distância internodal por tamanho de página		
	3 nodos	7 nodos	15 nodos
Paginação Seqüencial	2582,85	21381,17	131997,14
Algoritmo Proposto	791,96	8930,26	69755,07
Ótimo Teórico	272,33	3108,59	25249,96

Tabela 4.4 Quantidades médias de distância internodal.

Na figura 4.14 é apresentado o gráfico que descreve a evolução do comportamento do algoritmo em árvores com 10 a 400 nodos, sendo considerado o tamanho de página 3 nodos. A distância internodal produzida pelo algoritmo foi comparada com a produzida pela paginação seqüencial e com o valor teórico.

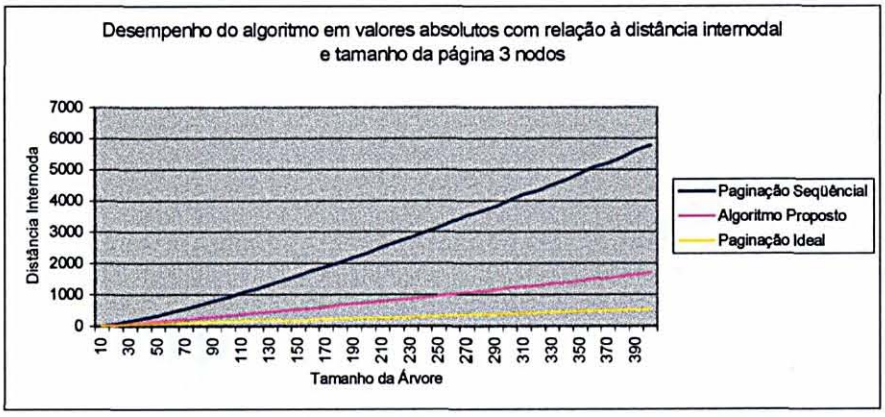

Figura 4.14 Distância internodal com tamanho de página 3 nodos.

Na figura 4.15 é apresentado o gráfico que descreve a evolução do comportamento do algoritmo em árvores com 10 a 900 nodos, sendo considerado o tamanho de página 7 nodos. A distância internodal produzida pelo algoritmo foi comparada com a produzida pela paginação seqüencial e com o valor teórico.

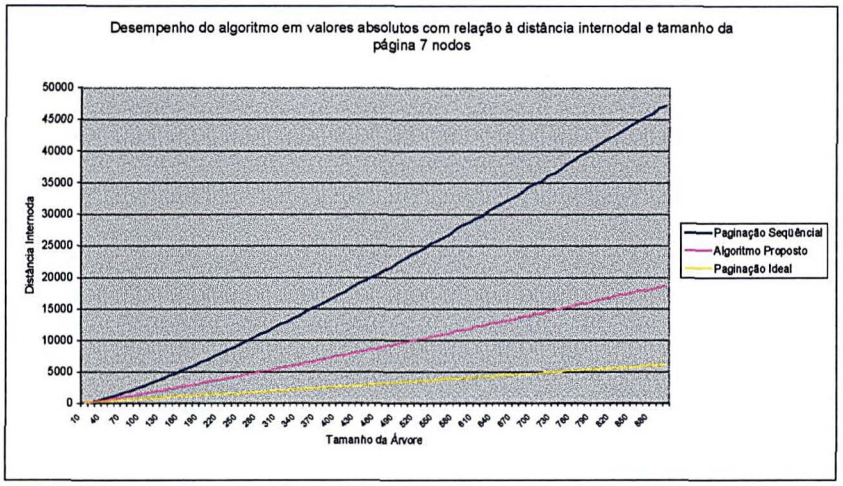

Figura 4.15 Distância internodal com tamanho de página 7 nodos.

Na figura 4.16 é apresentado o gráfico que descreve a evolução do comportamento do algoritmo em árvores com 10 a 2000 nodos, sendo considerado o tamanho de página 15 nodos. A distância internodal produzida pelo algoritmo foi comparada com a produzida pela paginação seqüencial e com o valor teórico.

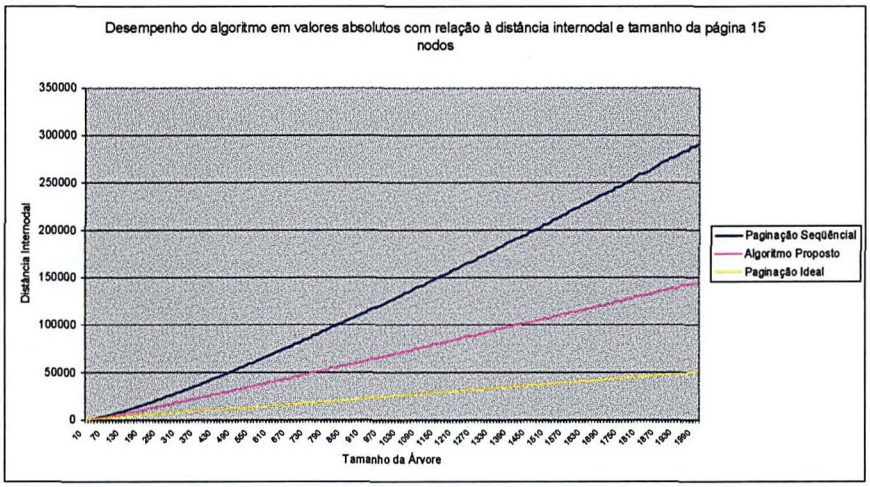

Figura 4.16 Distância internodal com tamanho de página 15 nodos.

Considerando o conjunto dos experimentos realizados, pode-se concluir que quanto à distância internodal o algoritmo tem desempenho 30% inferior ao ótimo teórico.

4.6 Comparação do Número de Páginas Visitadas com Valores Ótimos Teóricos, com a Paginação Seqüencial e com a Árvore B

No que se refere ao número de páginas visitadas o algoritmo pode ser comparado à paginação seqüencial, às árvores B e ao valor ótimo teórico. Para fins de comparação utilizou-se a implementação de árvores B descrita em [2], com as adaptações necessárias para utilização das mesmas árvores de teste e para os cálculos das métricas.

Para avaliar comparativamente as estratégias de paginação foram considerados os mesmos tamanhos de página (3, 7 e 15 nodos) no processamento da paginação seqüencial e no cálculo do valor ótimo teórico. Quanto às árvores B adotou-se como parâmetro de comparação árvores cujas ordens e respectivos tamanhos de páginas fossem, tanto quanto possível, os mais próximos do tamanho de página avaliado. Dessa forma, para o tamanho da página 3 nodos, foi realizada a comparação com árvores B de ordem 2 (4 nodos). Para o tamanho da página 7 nodos, foi realizada a comparação com árvores B de ordem 3 (6 nodos) e de ordem 4 (8 nodos). Para o tamanho da página 15 nodos, foi realizada a comparação com árvores B de ordem 7 (14 nodos) e de ordem 8 (16 nodos).

Na tabela 4.5 observam-se as quantidades médias de páginas visitadas obtidas nas diferentes estratégias.

Estratégia	Quantidades médias de páginas visitadas por tamanho de página		
	3 nodos	7 nodos	15 nodos
Paginação Seqüencial	1495,99	3313,30	7410,69
Algoritmo Proposto	969,69	1726,17	3365,55
Árvore B (1° caso)	-	1705,67	3195,93
Árvore B (2° caso)	841,04	1644,62	2914,78
Ótimo Teórico	771,45	1383,10	2760,65

Tabela 4.5 Quantidades médias de páginas visitadas.

Na figura 4.17 é apresentado o gráfico que descreve a evolução do comportamento do algoritmo em árvores com 10 a 400 nodos, sendo considerado o

tamanho de página 3 nodos. O algoritmo foi comparado, quanto ao número de páginas visitadas, com a paginação seqüencial, com o valor teórico e com árvores B de ordem 2, ou seja, que possuem páginas com capacidade de armazenar até 4 nodos. Pode-se observar que as árvores B e o algoritmo proposto tem desempenho, respectivamente, 9,60% e 27,36% inferior ao ótimo teórico.

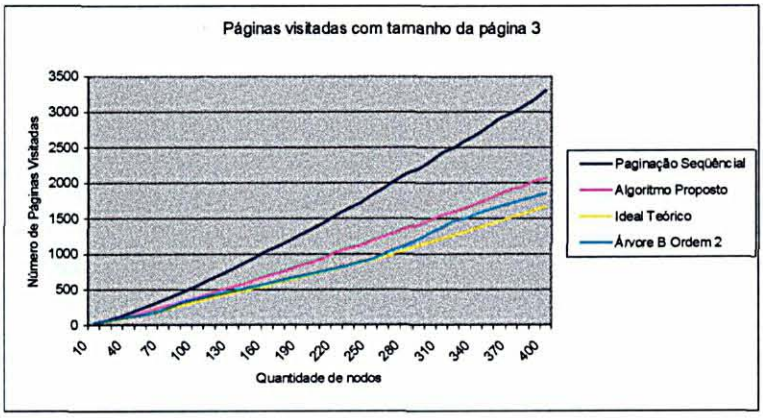

Figura 4.17 Número de páginas visitadas com tamanho de página 3 nodos.

Na figura 4.18 é apresentado o gráfico que descreve a evolução do comportamento do algoritmo em árvores com 10 a 900 nodos, sendo considerado o tamanho de página 7 nodos. O algoritmo foi comparado, quanto ao número de páginas visitadas, com a paginação seqüencial, com o valor teórico e com árvores B de ordem 3 e 4, ou seja, que possuem páginas com capacidade para armazenar até 6 e 8 nodos, respectivamente. Pode-se observar que as árvores B e o algoritmo proposto tem desempenho, respectivamente, 15,13% e 17,77% inferior ao ótimo teórico.

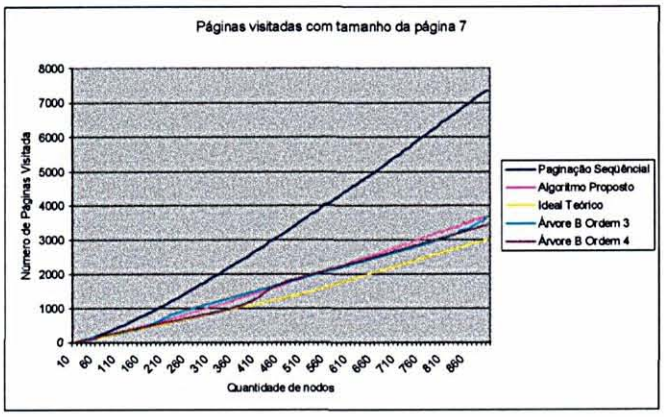

Figura 4.18 Número de páginas visitadas com tamanho de página 7 nodos.

Na figura 4.19 é apresentado o gráfico que descreve a evolução do comportamento do algoritmo em árvores com 10 a 2000 nodos, sendo considerado o tamanho de página 15 nodos. O algoritmo foi comparado, quanto ao número de páginas visitadas, com a paginação seqüencial, com o valor teórico e com árvores B de ordem 7 e 8, ou seja, que possuem páginas com capacidade para armazenar até 14 e 16 nodos, respectivamente. Pode-se observar que as árvores B e o algoritmo proposto tem desempenho, respectivamente, 6,34% e 13,01% inferior ao ótimo teórico.

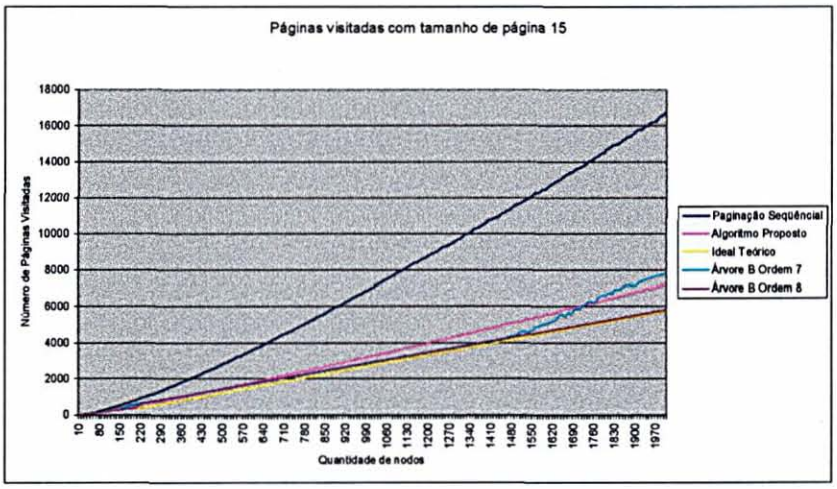

Figura 4.19 Número de páginas visitadas com tamanho de página 15 nodos.

Considerando os resultados apresentados, o algoritmo proposto continua, nesta métrica, entre a paginação seqüencial e o valor ótimo teórico, sendo muito mais próximo do ótimo teórico do que da paginação seqüencial.

Considerando o conjunto dos experimentos, no que se refere à métrica número de páginas visitadas, o algoritmo proposto apresentou-se 10% inferior às árvores B. Entretanto, uma diferença significativa foi obtida na quantidade de espaços requeridos pelas duas estratégias, conforme descrito na próxima seção.

4.7 Análise dos Espaços Não Utilizados nas Páginas

Outro resultado experimental refere-se aos espaços não utilizados nas páginas obtidos pelo processamento do algoritmo. Na tabela 4.6 é possível avaliar a quantidade

de espaços não utilizados produzidos na paginação seqüencial, que é mínima, no algoritmo proposto e nas árvores B. A unidade de medida utilizada na comparação é o tamanho do nodo da árvore. Observa-se que as árvores B, por serem estruturas de dados de múltiplos caminhos, apresentam uma quantidade muito grande de espaços não utilizados.

Estratégia	Quantidade total de espaços não utilizados produzidos nos experimentos		
	3 nodos	7 nodos	15 nodos
Paginação Seqüencial	4.100	27.300	100.500
Algoritmo Proposto	4.235	27.327	100.632
Árvores B (1° caso)	-	1.934.922	9.315.736
Árvores B (2° caso)	395.944	1.936.144	9.296.000

Tabela 4.6 Quantidade total de espaços não utilizados.

A figura 4.20 ilustra graficamente a comparação das quantidades totais de espaços não utilizados produzidos nas diferentes estratégias de paginação. Nos experimentos realizados observou-se que o algoritmo produz uma quantidade de espaços muito próxima da seqüencial que é ótima. Considerando a necessidade de transferência integral da árvore, por exemplo, em uma rede de computadores, o algoritmo apresenta um desempenho muito superior ao das árvores B, em termos de banda ocupada.

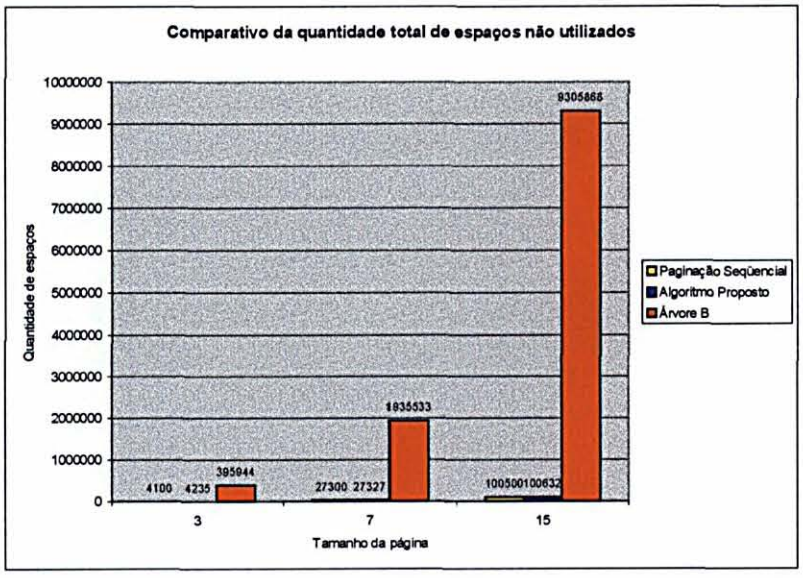

Figura 4.20 Comparativo da quantidade total de espaços não utilizados.

Na tabela 4.7 é possível avaliar as taxas de preenchimento de páginas observadas nos experimentos realizados na paginação seqüencial, no algoritmo proposto e nas árvores B.

Estratégia	Taxas de preenchimento de páginas		
	3 nodos	7 nodos	15 nodos
Paginação Seqüencial	98,8177 %	98,4188 %	98,6878 %
Algoritmo Proposto	98,7736 %	98,4166 %	98,6800 %
Árvores B (1° caso)	-	67,5282 %	67,8817 %
Árvores B (2° caso)	67,1446 %	67,3045 %	67,7530 %

Tabela 4.7 Taxas de preenchimento de páginas.

A figura 4.21 ilustra graficamente a comparação das taxas de preenchimento de páginas obtidas nas diferentes estratégias de paginação. Nos experimentos realizados observou-se que as árvore B apresentam a taxa média de preenchimento de 67,52% em árvores aleatórias, confirmando resultados de outros trabalhos [19]. Por sua vez, o algoritmo proposto apresentou uma taxa média de preenchimento de 98,62%, muito próxima da ótima que é obtida na paginação seqüencial.

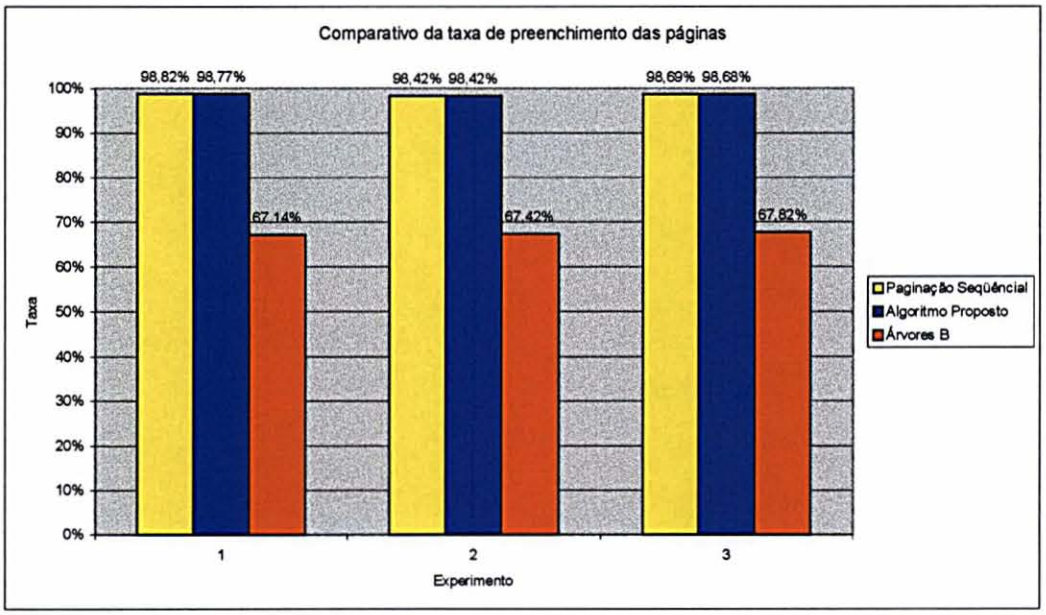

Figura 4.21 Comparativo da taxa de preenchimento das páginas.

No anexo 4 encontram-se os valores obtidos nos experimentos que foram utilizados para a apresentação das tabelas e gráficos deste capítulo.

Capítulo 5

Conclusão

Neste trabalho foi descrito um algoritmo que realiza a paginação de árvores binárias de pesquisa. O algoritmo constrói a paginação ótima quando isso é possível e propõe uma política eficiente para o preenchimento das páginas de uma árvore binária degenerada, baseada na aplicação de empacotamento unidimensional na franja da árvore. A complexidade do algoritmo foi apresentada e discutida. O algoritmo foi implementado e resultados experimentais foram apresentados. Observou-se que o algoritmo produz um esquema de paginação até aproximadamente 50% superior à paginação seqüencial e com resultados próximos aos obtidos com a utilização de árvores B, no que se refere ao número de páginas visitadas em pesquisas. Por outro lado, no que se refere à taxa de preenchimento das páginas o algoritmo apresentou uma taxa média de preenchimento de 98,62%, enquanto as árvores B chegam até 67,52%. Uma versão alternativa do algoritmo que visa reduzir a distância internodal nas páginas foi também apresentada.

Como trabalhos futuros deve-se estudar o comportamento do algoritmo em aplicações reais e não apenas obtidas com dados aleatórios. Também deverá ser investigado o comportamento do algoritmo em implementações que envolvam outros algoritmos de aproximação para alocação das subárvores pertencentes à franja. Além disso, sugere-se a realização de experimentos envolvendo comparações com variações da árvore B. Por fim, deve ser estudado o comportamento do algoritmo diante de inserções e retiradas freqüentes de nodos, bem como do acesso concorrente aos dados.

Referências

[1] SEDGEWICK. R. **Algorithms in C.** Addison-Wesley, 1990, 702 p.

[2] ZIVIANI, N. **Projeto de algoritmos com implementações em Pascal e C.** São Paulo: Pioneira, 1996, 267 p.

[3] KNUTH. D. E. **The Art of Computer Programming.** v. 3: Sorting and Searching. Addison-Wesley, 1973, 780 p.

[4] CORMEN, T. H. ; LEISERSON, C. E. ; RIVEST, R. L. **Introduction to Algorithms.** Cambrige: The MIT Press, 1996, 1028 p.

[5] HOROWITZ, E. ; SAHIN, S. **Fundamentos de estrutura de dados.** Rio de Janeiro: Campus, 1987, 494 p.

[6] TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estruturas de dados usando C.** São Paulo: Prentice Hall, 1995, 884 p.

[7] GONNET, G. H. ; BAEZA-YATES, R. **Handbook of Algorithms and Data Structures: in Pascal and C.** Addison-Wesley, 1991, 424 p.

[8] TANENBAUM, A. S. **Computer Networks.** New Jersey: Prentice Hall PTR, 2002, 912 p.

[9] FRAKES, W. B. ; BAEZA-YATES, R. **Information Retrieval Data Structures and Algorithms.** Prentice Hall, 1992, 464 p.

[10] BERNARDI, R. **Aplicando a técnica de times assíncronos na otimização de problemas de empacotamento unidimensional.** Dissertação de Mestrado, São Paulo: Universidade de São Paulo, 2001, 73 p.

[11] MIYAZAWA, F. K. **Algoritmos de aproximação para problemas de empacotamento.** Tese de Doutorado, São Paulo: Universidade de São Paulo, 1997, 208 p.

[12] MIYAZAWA, F. K. ; WAKABAYASHI, Y. **Algoritmos de aproximação para problemas de empacotamento.** Anais do Congresso da Sociedade Brasileira de Computação, Belo Horizonte, SBC, p. 335-351, 1998.

[13] BAYER. R. ; MCCREIGHT, E. Organization and Maintenance of Large Ordered Indexes. **Acta Informatica**, v. 1, p. 173-189, 1972.

[14] VITTER, J. S. External Memory Algorithms and Data Structures: Dealing with Massive Data. **ACM Computing Surveys**, v. 33, n. 2, p. 209-271, 2001.

[15] BAEZA-YATES, R. ; RIBEIRO-NETO, B. **Modern Information Retrieval.** Addison-Wesley, 1999, 513 p.

[16] MUNTZ, R.; UZGALIS. R. **Dynamic Storage Allocation for Binary Search Trees in Two Level Memory.** Proceedings of Princeton Conference on Information Sciences and Systems, Princeton, New Jersey, v. 4, p. 345-349, 1970.

[17] HILL, E. Jr A Comparative Study of Very Large Data Bases. **Lectures Notes in Computer Science.** n. 59, Springer Verlag, Berlim, 1978, em [21].

[18] NIEVERGELT. J. Binary Search Trees and File Organization. **Computing Surveys**, v. 6, p. 195-207, 1974.

[19] CESARINI, F. ; SODA, G. Binary Trees Paging. **Information Systems**, v. 7, n. 4, p. 337-344, 1982.

[20] WRIGHT, W. E. Binary Search Trees in Secondary Memory. **Acta Informatica**, v. 15, p. 3-17, 1981.

[21] SPRUGNOLI, R. On The Allocation of Binary Trees in Secondary Storage. **BIT Numerical Mathematics.** v. 21, n. 3, p. 305-316, 1981.

[22] GONÇALVES, C. C. **A Árvore PATRICIA como método de acesso para bancos de dados não estruturados.** Dissertação de Mestrado, Belo Horizonte: Universidade Federal de Minas Gerais, 1989, 115 p.

[23] HENRICH, A. ; SIX, H.W. ; WIDMAYER, P. **Paging Binary Trees with External Balancing.** Proceedings of the 15th International Workshop on Graph-theoretic Concepts in Computer Science, Castle Rolduc, The Netherlands, Springer-Verlag, p. 260-276, 1990.

[24] TAVARES, R. A. E. **Um algoritmo para paginação de árvore binárias de pesquisa.** Monografia de Especialização, Curitiba: Universidade Federal do Paraná, 1993, 51 p.

[25] BARBOSA, E. F. **Estruturas de dados e alocação de arquivos em discos CD-ROM.** Dissertação de Mestrado, Belo Horizonte: Universidade Federal de Minas Gerais, 1990, 139 p.

[26] BARBOSA, E. F. ; ZIVIANI, N. **Data Structures and Access Methods for Read-Only Optical Disks.** BAEZA-YATES, R. ; MANBER, U. (Editors) In: Computer Science: Research and Applications, New York: Plenum Publishing Corp., p. 189-207, 1992.

[27] BARBOSA, E. F. **Métodos eficientes de busca em texto armazenado em memória secundária.** Tese de Doutorado, Belo Horizonte: Universidade Federal de Minas Gerais, 1995, 169 p.

[28] WIRTH, N. **Algoritmos e estruturas de dados.** Rio de Janeiro: LTC, 2001, 272 p.

[29] ADEL'SON-VEL'SKII, G. M. ; LANDIS E. M. An Algorithm for The Organization of Information. **Doklady Akademia Nauk USSR**, v. 146, n. 2, p. 263-266, Tradução para o Inglês em Soviet Math. Doklay 3, p. 1259-1263, 1962.

[30] TANENBAUM, A. S. ; WOODHULL, A. S. **Sistemas operacionais: projeto e implementação.** Porto Alegre: Bookman, 2000, 759 p.

[31] MACHADO, F. B. ; MAIA, L. P. **Arquitetura de sistemas operacionais.** Rio de Janeiro: LTC, 2002, 311 p.

[32] ALBERS, S. **Online Algorithms: A Study of Graph-Theoretic Concepts.** Proceedings of the 25th International Workshop on Graph-theoretic Concepts in Computer Science, Ascona, Switzerland, Springer-Verlag, p. 10-26, 1999.

[33] PARKER, R. G. ; RARDIN, R. L. **Discrete Optimization.** Computer Science and Scientific Computing, Academic Press, 1988, 472 p.

[34] GAREY, M. R. ; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness.** W. H. Freeman and Company, 1979, 338 p.

[35] GREENLAW, R. ; HOOVER, H. J. **Fundamentals of the Theory of Computation: Principles and Practice.** San Francisco: Morgan Kaufmann Publishers, 1998, 336 p.

[36] GOTTFRIED, B. S. **Schaum's Outline of Programming with Pascal.** McGraw-Hill, 1994, 428 p.

Anexo 1

Programa Gerador de Árvores Aleatórias

```
(* ************************************************************************
   UNIVERSIDADE FEDERAL DO PARANA
   MESTRADO EM INFORMATICA
   ORIENTADOR: PROF. DR. ELIAS P. DUARTE JR

   TRABALHO DE DISSERTACAO DE MESTRADO
   PROGRAMA GERADOR DE SEQUENCIA DE CHAVES DE ARVORES ALEATORIAS

   DESENVOLVIDO POR RUI ALBERTO ECKE TAVARES
   ************************************************************************
*)

Program Gera_Arquivo;

Const
  Max_Num = 20000;

Type
  Ptr = ^Nodo;
  Nodo = Record
           Info : Integer;
           Prox : Ptr;
         End;

Var
  Ind, Ensaio  : Integer;
  Lista        : Ptr;


(* Funcao que verifica se a lista de numeros esta vazia ou nao *)
Function Vazia (Lista : Ptr) : Boolean;
Begin
  If (Lista^.Prox = Nil)
    Then
      Vazia := True
    Else
      Vazia := False;
End;

(* Procedimento que cria uma lista de numeros *)
Procedure Cria_lista;
Begin
  New(Lista);
  Lista^.Prox := Nil;
End;

(* Procedimento que libera o espaco alocado por uma lista de numeros *)
Procedure Finaliza_lista;
Var
  P1,P2 : Ptr;
Begin
  P1 := Lista;
  P2 := Lista;
  While (Not Vazia(P1)) Do
    Begin
      P1 := P1^.Prox;
      Dispose(P2);
      P2 := P1;
    End;
  Dispose(P1);
```

```
  Lista := Nil;
End;

(* Funcao que verifica se o numero esta na lista ou nao *)
Function Existe (Lista : Ptr; Val : Integer) : Boolean;
Var
  Achou : Boolean;
  P     : Ptr;
Begin
  Achou := False;
  P     := Lista;
  While (Not Achou) And (Not Vazia(P)) Do
    If (P^.Prox^.Info = Val)
      Then
        Achou := True
      Else
         P := P^.Prox;
  Existe := Achou;
End;

(* Procedimento que insere um numero na lista *)
Procedure Insere (Lista : Ptr; Val : Integer);
Var
  P : Ptr;
Begin
  P := Lista;
  While (Not Vazia(P)) Do
    P := P^.Prox;
  New(P^.Prox);
  P^.Prox^.Info := Val;
  P^.Prox^.Prox := Nil;
End;

(* Funcao que verifica se o numero ja foi gerado anteriormente ou nao *)
Function Novo (Num : Integer) : Boolean;
Begin
  If Not Existe(Lista, Num)
    Then
      Begin
        Insere(Lista, Num);
        Novo := True;
      End
    Else
      Novo := False;
End;

(* Procedimento que cria um arquivo com dados aleatorios *)
Procedure Criar_Arquivo_Aleatorio (Ensaio : Integer; Tam :  Integer);
Var
  Arq               : Text;
  K,Num             : Integer;
  Nome_arq, s1, s2  : String;
Begin
  Randomize;
  Cria_lista;
  str(Tam,s1);
  str(Ensaio,s2);
  Nome_Arq := 'a' + s1 + 'e'+ s2 + '.txt';
  Writeln(Nome_Arq);
  Assign(Arq,Nome_Arq);
```

```
  Rewrite(Arq);
  For k := 1 to Tam Do
    Begin
      Repeat
        Num := Trunc(Random(Max_Num));
      Until Novo(Num);
      Writeln(Arq,Num);
      Write(Arq,Char(trunc(Random(23)+65)));
      If k < Tam
        then
          Writeln(Arq);
    End;
  Close(Arq);
  Finaliza_lista;
End;

Begin
  For Ind := 1 To 200 do
    For Ensaio := 1 to 100 do
      Criar_Arquivo_Aleatorio(Ensaio, Ind*10);
End.
```

Anexo 2

Programa de Teste do Algoritmo

```
(* ************************************************************************
   UNIVERSIDADE FEDERAL DO PARANA
   MESTRADO EM INFORMATICA
   ORIENTADOR: PROF. DR. ELIAS P. DUARTE JR

   TRABALHO DE DISSERTACAO DE MESTRADO
   - ALGORITMO PARA PAGINACAO DE ARVORES BINARIAS DE PESQUISA
   - PROGRAMA DE TESTE DO ALGORITMO PROPOSTO

   DESENVOLVIDO POR RUI ALBERTO ECKE TAVARES
   ************************************************************************
*)

Program Testador_Paginacao (Input,Output);

{$M 65500,0,655360}

Const
  Max_Tam_Pag  = 15;
  Max_Pag      = 140;
  Esq          =  1;
  Dir          =  2;
  Nulo         = -1;

Type
  Nodo = Record
           Chave   : Integer;
           Info    : Char;
           Pag_Dir : Integer;
           Pos_Dir : Integer;
           Pag_Esq : Integer;
           Pos_Esq : Integer;
         End;
  Vet_result = Array[0..99] of Real;

Conj = Record
         Pag, Pos, Lado : Integer;
       End;

Page = Array[0..Max_Tam_Pag] Of Nodo;

Var
  Pag,Pos       : Integer;               (* Pagina e Posicao Atual      *)
  NN            : Integer;               (* Numero de Niveis por Pagina *)
  Tam_Pag       : Integer;               (* Tamanho da Pagina           *)
  Sq            : Array[0..500] of Nodo; (* Estrutura de Dados Stack-Queue
*)
  Fl            : Array[0..500] of Nodo; (* Estrutura de Dados Fringe List
*)
  Sqaux         : Array[0..500] of Conj; (* Estrutura Auxiliar da SQ *)
  Flaux         : Array[0..500] of Conj; (* Estrutura Auxiliar da FL *)
  Tot_Sq        : Integer;               (* Quantificador da SQ *)
  Tot_Fl        : Integer;               (* Quantificador da FL *)
  Vetor         : Array[0..Max_Pag,0..Max_Tam_Pag] Of Nodo;
                                         (* Armazenador das Paginacoes *)

(* Insere elemento na Stack_Queue *)
Procedure Insere_SQ (Val : Nodo ; I,J,L : Integer);
Begin
  Tot_Sq := Tot_Sq + 1;
```

```
  Sq[Tot_Sq] := Val;
  Sqaux[Tot_Sq].Pag  := I;
  Sqaux[Tot_Sq].Pos  := J;
  Sqaux[Tot_Sq].Lado := L;
End;

(* Retira elemento da Stack_Queue *)
Procedure Retira_SQ (Var Val : Nodo;  Var I : Integer;
                     Var J : Integer; Var L : Integer);
Var
  Ind : Integer;
Begin
  Val := Sq[1];
  I   := Sqaux[1].Pag;
  J   := Sqaux[1].Pos;
  L   := Sqaux[1].Lado;
  For Ind := 1 to Tot_Sq-1 Do
    Begin
      Sq[Ind]    := Sq[Ind+1];
      Sqaux[Ind] := Sqaux[Ind+1];
    End;
  Tot_Sq := Tot_Sq - 1;
End;

(* Remove elemento da Stack_Queue *)
Procedure Remove_SQ (I :  Integer);
Var
  Ind : Integer;
Begin
  For Ind := I to Tot_Sq-1 Do
    Begin
      Sq[Ind]    := Sq[Ind+1];
      Sqaux[Ind] := Sqaux[Ind+1];
    End;
  Tot_Sq := Tot_Sq - 1;
End;

(* Desempilha elemento da Stack_Queue *)
Procedure Desemp_SQ (Var Val : Nodo;  Var I :  Integer;
                     Var J : Integer; Var L : Integer);
Begin
  Val := Sq[Tot_Sq];
  I   := Sqaux[Tot_Sq].Pag;
  J   := Sqaux[Tot_Sq].Pos;
  L   := Sqaux[Tot_Sq].Lado;
  Tot_Sq := Tot_Sq - 1;
End;

(* Indica se Stack_Queue esta vazia ou nao *)
Function Vazia_SQ : Boolean;
Begin
 If (Tot_Sq = 0)
   Then
     Vazia_Sq := True
   Else
     Vazia_Sq := False;
End;

(* Insere elemento na Fringe List *)
Procedure Insere_FL (Val : Nodo;I,J,L : Integer);
```

```
Begin
  Tot_FL := Tot_FL + 1;
  FL[Tot_FL] := Val;
  Flaux[Tot_Fl].Pag  := I;
  Flaux[Tot_Fl].Pos  := J;
  Flaux[Tot_Fl].Lado := L;
End;

(* Retira elemento da Fringe List *)
Procedure Retira_FL (Ind : Integer;Var Val : Nodo;  Var I :  Integer;
                     Var J : Integer; Var L : Integer);
Var
  C : Integer;
Begin
  Val := Fl[Ind];
  I   := Flaux[Ind].Pag;
  J   := Flaux[Ind].Pos;
  L   := Flaux[Ind].Lado;
  For C := Ind to Tot_Fl-1 Do
    Begin
      Fl[C]    := Fl[C+1];
      Flaux[C] := Flaux[C+1];
    End;
  Tot_Fl := Tot_Fl - 1;
End;

(* Indica se a Fringe List esta vazia ou nao *)
Function Vazia_FL : Boolean;
Begin
 If (Tot_FL = 0)
   Then
     Vazia_FL := True
   Else
     Vazia_FL := False;
End;

(* Indica se o caracter eh alfanumerico *)
Function Alfa (Ch : Char): Boolean;
Begin
  If ( (Ch >= 'a') And (Ch <= 'z') Or
       (Ch >= 'A') And (Ch <= 'Z') Or
       (Ch >= '0') And (Ch <= '9') )
    Then
      Alfa := True
    Else
      Alfa := False;
End;

(* Funcao que calcula a elevado a b *)
Function pot(a,b : longint) : longint;
begin
  if (b = 0)
    then
      pot := 1
    else
      pot := a * pot(a,b-1);
end;


(* Desloca a pag e pos para a proxima posicao *)
```

```
Procedure Prox (Var Pag : Integer; Var Pos : Integer);
Begin
  Pos := Pos + 1;
  If (Pos = Tam_Pag)
    Then
      Begin
        Pag := Pag + 1;
        Pos := 0;
      End;
End;

(* Indica se o nodo REG eh folha ou nao pelo lado LADO *)
Function Eh_Folha (Reg : Nodo; Lado : Integer) : Boolean;
Var
  Eh : Boolean;
Begin
  Eh := False;
  If (Lado = Esq)
    Then
      If (Reg.Pag_Esq = Nulo) And (Reg.Pos_Esq = Nulo)
        Then
          Eh := True
        Else
          Begin
          End
    Else
      If (Lado = Dir)
         Then
           If (Reg.Pag_Dir = Nulo) And (Reg.Pos_Dir = Nulo)
            Then
              Eh := True
            Else
              Begin
              End
         Else
           Writeln('ERRO DE PASSAGEM DE FUNCOES');
  Eh_Folha := Eh;
End;

(* Procedimento que grava a paginacao sequencial *)
Procedure Gravar (N : Integer);
Var
  X,I,J : Integer;
  Dest  : File Of Nodo;
Begin
  If (N > 0)
    Then
      Begin
        Assign(Dest,'dest.bin');
        Rewrite(Dest);
        If Ioresult <> 0
          Then
            Begin
              Writeln('ERRO DE ABERTURA DE ARQUIVO');
              Exit;
            End;
        I := 0;
        J := 0;
        For X:= 0 To N Do
          Begin
```

```
            Write(Dest,Vetor[i,j]);
            Prox(I,J);
          End;
        Close(Dest);
      End
    Else
      Writeln('NAO CONSTAM INFORMACOES NO SISTEMA');
End;

(* Funcao que retorna o espaco livre de uma pagina *)
Function Livre (I : Integer) : Integer;
Begin
  If I < Pag
    Then
      Livre := 1
    Else
      Livre := Tam_Pag - Pos + 1;
End;

(* Funcao que indica se os valores de i e j sao validos *)
Function Ok (I : Integer; J : Integer) : Boolean ;
Begin
  If ((I*Tam_Pag + J) < (Pag*Tam_Pag + Pos))
    Then
      Ok := True
    Else
      Ok := False;
End;

(* Funcao que le um nodo do arquivo indicado em local *)
Function Ler_Nodo (I : Integer; J : Integer;
                   Var Resp : Nodo; local : String) : Boolean;
Var
   Arq : File Of Nodo;
   P   : Nodo;
   Ok  : Boolean;
Begin
  Assign(Arq,Local);
  Reset(Arq);
  If Ioresult <> 0
    Then
      Begin
        Writeln('ERRO DE ABERTURA DE ARQUIVO');
        Exit;
      End;
  Ok := False;
  If (I <= Pag) And (J <= Tam_Pag)
    Then
      If (I*Tam_Pag + J <= Pag*Tam_Pag + Pos)
        Then
          Begin
            Seek(Arq,I*Tam_Pag + J);
            Read(Arq,P);
            Ok := True;
          End;
       Close(Arq);
   Resp := P;
   Ler_Nodo := Ok;
End;
```

```
(* Funcao que escreve um nodo no arquivo indicado em local *)
Function Gravar_Nodo (I: Integer; J : Integer;
                      Resp : Nodo; Local : String) : Boolean;
Var
  Arq : File Of Nodo;
Begin
  Assign(Arq,Local);
  Reset(Arq);
  If Ioresult <> 0
    Then
      Begin
        Writeln('ERRO DE ABERTURA DE ARQUIVO');
        Exit;
      End;
  Seek(Arq,I*Tam_Pag + J);
  Write(Arq,Resp);
  Close(Arq);
  Gravar_Nodo := True;
End;

(* Procedimento que altera o conteudo de um nodo no arquivo destino *)
Procedure Alterar_Nodo(I,J,L,Pai,Posic : Integer);
Var
  P : Nodo;
Begin
  If Ler_Nodo(I,J,P,'dest.otm') And (L In [1,2])
    Then
      Begin
        If L = 1
          Then
            Begin
              P.Pag_Esq := Pai;
              P.Pos_Esq := Posic;
            End
          Else
            Begin
              P.Pag_Dir := Pai;
              P.Pos_Dir := Posic;
            End;
        If Not Gravar_Nodo(I,J,P,'dest.otm')
          Then
            Writeln('ERRO DE OTIMIZACAO');
      End
    Else
      Writeln('ERRO DE OTIMIZACAO');
End;

(* Procedimento que le uma pagina do arquivo indicado em local *)
Procedure Ler_Pagina (I : Integer; Var Aux : Page;
                      Local : String);
Var
  J   : Integer;
  Ok  : Boolean;
Begin
  If I <= Pag
    Then
      For J := 0 To Tam_Pag - Livre(I) Do
        Ok := Ler_Nodo(I,J,Aux[J],Local);
End;
```

```
(* Procedimento que grava uma pagina no arquivo indicado em local *)
Procedure Gravar_Pagina (I : Integer; Aux : Page; Local: String);
Var
 J  : Integer;
 Ok : Boolean;

Begin
 If I <= Pag
   Then
     For J := 0 to Tam_Pag - 1 Do
       Ok := Gravar_Nodo(I,J,Aux[J],Local);
End;


(* Procedimento que cria um arquivo *)
Procedure Criar_Arquivo (Str : String);
Var
  Arq : File Of Nodo;
Begin
  Assign(Arq,Str);
  Rewrite(Arq);
  Close(Arq);
End;

(* Procedimento que localiza um filho esquerdo ou direito *)
Procedure Encontrar (Var P : Nodo; Lado : Integer);
Var
  I,J : Integer;
Begin
  If Lado = Esq
    Then
      Begin
        I := P.Pag_Esq;
        J := P.Pos_Esq;
        P := Vetor[I,J];
      End;
  If Lado = Dir
    Then
      Begin
        I := P.Pag_Dir;
        J := P.Pos_Dir;
        P := Vetor[I,J];
      End;
End;

(* Funcao que localiza e calcula o tamanho de uma subarvore dado um nodo *)
Function Tam_Sub_Arv (N : Nodo) :  Integer;
Begin
  If ((N.Pag_Dir = Nulo) And
      (N.Pos_Dir = Nulo) And
      (N.Pag_Esq = Nulo) And
      (N.Pag_Esq = Nulo))
    Then
      Tam_Sub_Arv := 1
    Else
      Begin
        If ((N.Pag_Dir <> Nulo) And
            (N.Pos_Dir <> Nulo) And
            (N.Pag_Esq <> Nulo) And
            (N.Pos_Esq <> Nulo))
```

```
        Then
        Tam_Sub_Arv := Tam_Sub_Arv(Vetor[N.Pag_Dir,N.Pos_Dir])
                     + Tam_Sub_Arv(Vetor[N.Pag_Esq,N.Pos_Esq]) + 1
        Else
         If ((N.Pag_Dir <> Nulo) And
             (N.Pos_Dir <> Nulo))
           Then
             Tam_Sub_Arv := Tam_Sub_Arv(Vetor[N.Pag_Dir,N.Pos_Dir]) + 1
           Else
             Tam_Sub_Arv := Tam_Sub_Arv(Vetor[N.Pag_Esq,N.Pos_Esq]) + 1;
      End;
End;

(* Procedimento que inicializa uma pagina a ser otimizada *)
Procedure Inic_Page (Var Aux : Page);
Var
  Ind : Integer;
Begin
    For Ind := 0 to Tam_Pag - 1 do
      With Aux[Ind] Do
       Begin
           Chave   := -1;
           Info    := '-';
           Pag_Dir := -1;
           Pos_Dir := -1;
           Pag_Esq := -1;
           Pos_Esq := -1;
       End;
End;

(* Procedimento que Otimiza a Paginacao Sequencial *)
Procedure Otimizar;
Var
Patriarca, Tmp          : Nodo;
Aux                     : Page;
Cont, K, Ind            : Integer;
Page_Num, Np, Esp_Disp  : Integer;
G, H, T                 : Integer;

(* Procedimento que vincula nodos pai e filho *)
Procedure Marcar_Pai (Var Aux : Page; I,J,Lado,X,Y : Integer);
Var
  Arq : Text;
Begin
  If (I <> -1) And (J <> -1) And (Lado <> -1)
    Then
       Begin
         If I = X
           Then
             If Lado = 1
               Then
                 Begin
                   Aux[J].Pag_Esq := X;
                   Aux[J].Pos_Esq := Y;
                 End
               Else
                 Begin
                   Aux[J].Pag_Dir := X;
                   Aux[J].Pos_Dir := Y;
                 End
```

```
          Else
            Alterar_Nodo(I, J, Lado, X, Y);
       End;
End;

(* Procedimento que aloca um nodo numa pagina *)
Procedure Aloca (Var Aux : Page; Item: Nodo;
                 Contador : Integer; I,J,Lado : Integer);
Var
Fesq, Fdir : Nodo;
X, Y       : Integer;
Begin
  Np := Np + 1;
  Aux[Np] := Item;
  (* Salva a Pagina e posicao do pai *)
  X := Page_Num;
  Y := Np;
  Marcar_Pai(Aux,I,J,Lado,X,Y);    (* Marca o pai *)
  If Contador > 0
    Then
      Begin
        If Item.Pag_Esq <> Nulo
          Then
            Begin
              Fesq := Item;
              Encontrar(Fesq, Esq);
              Aloca(Aux,Fesq,Contador -1,X,Y, Esq);
            End
          Else
            Begin
              Aux[Np].Pag_Esq := Nulo;
              Aux[Np].Pos_Esq := Nulo;
            End;
         If Item.Pag_Dir <> Nulo
           Then
             Begin
               Fdir := Item;
               Encontrar(Fdir, Dir);
               Aloca(Aux,Fdir,Contador-1,X,Y,Dir);
             End
           Else
             Begin
               Aux[Np].Pag_Dir := Nulo;
               Aux[Np].Pos_Dir := Nulo;
             End;
      End
    Else
      If (Contador = 0)
        Then
          Begin
            If Item.Pag_Esq <> Nulo
              Then
                Begin
                  Fesq := Item;
                  Encontrar(Fesq,Esq);
                  Insere_Sq(Fesq,X,Y,Esq);
                End;
            If Item.Pag_Dir <> Nulo
              Then
                Begin
```

```
                Fdir := Item;
                Encontrar(Fdir,Dir);
                Insere_Sq(Fdir,X,Y,Dir);
              End
          End;
End;

(* Funcao que encontra a melhor subarvore a ser alocada *)
Function Find_Best(Var Best : Integer;Disp : Integer) : Boolean;
Var
  I,Tam,TamBest : Integer;
  Achou         : Boolean;
Begin
  Achou := False;
  I := 1;
  While (I <= Tot_FL) And (Not Achou) Do
    Begin
     If (Disp - Tam_Sub_Arv(FL[I]) >= 0)
      Then
        Begin
          Achou := True;
          Best := I;
          TamBest := Tam_Sub_Arv(FL[Best]);
        End;
      I := I + 1;
    End;
  If Achou
    Then
       While (I <= Tot_FL) Do
         Begin
           Tam := Tam_Sub_Arv(FL[I]);
           If (Disp - Tam) >= 0
            Then
               If (Disp - Tam) < (Disp - TamBest)
                 Then
                   Begin
                     Best := I;
                     TamBest := Tam_Sub_Arv(FL[Best]);
                   End;
           I := I + 1;
         End;
  Find_Best := Achou;
End;

(* Procedimento que armazena uma subarvore numa pagina *)
Procedure Armazena_Sub_Arv(Var Aux : Page; N : Nodo;
G,H,T : Integer; Page_Num : Integer; Var K : Integer);
Var
  Tmp : Nodo;
  J   : Integer;
Begin
  Aux[k] := N;
  Marcar_Pai(Aux,G,H,T,Page_Num,K);
  J := K;
  K := K  + 1;
  If (Aux[J].Pag_Esq <> Nulo)
    Then
      Begin
        Tmp := Aux[J];
        Encontrar(Tmp,Esq);
```

```
        Armazena_Sub_Arv(Aux,Tmp,Page_Num,J,Esq,Page_Num,K);
      End;
  If Aux[J].Pag_Dir <> Nulo
     Then
      Begin
        Tmp := Aux[J];
        Encontrar(Tmp,Dir);
        Armazena_Sub_Arv(Aux,Tmp,Page_Num,J,Dir,Page_Num,K);
      End;
End;

Begin
  Criar_Arquivo('dest.otm');
  Tot_SQ := 0;
  (* Criar SQ *)
  Insere_Sq(Vetor[0,0],-1,-1,-1);
  (* Alocar raiz da arvore na SQ *)
  Page_Num := -1;
  Repeat
    Page_Num := Page_Num + 1;              (* Cria nova pagina *)
    Inic_Page(Aux);

    Retira_Sq(Patriarca,G,H,T);  (* Patriarca <- SQ[1] *)
    Np := -1;                    (* Posicao na pagina *)
    Cont := NN;                   (* Numero de Niveis *)

    Aloca(Aux,Patriarca,cont-1,G,H,T);
    (* Aloca X-1 geracoes na pagina *)
    (* Inserir a geracao X+1 em SQ *)

    K := Np+1;

    (* Percorre SQ e decide quais elementos deverao ser alocado na FL *)
    If K < Tam_Pag
     Then
       Esp_Disp := Tam_Pag - K
     Else
       Esp_Disp := Tam_Pag;
     Ind := 1;
     While Ind <= Tot_Sq Do
     Begin
       If Tam_Sub_Arv(Sq[ind]) < Esp_Disp
         Then
           Begin

Insere_FL(Sq[ind],Sqaux[ind].Pag,Sqaux[ind].Pos,Sqaux[ind].Lado);
             Remove_SQ(Ind);
           End
         Else
           Ind := Ind + 1;
     End;
    While (K < Tam_Pag) And (Not Vazia_Sq) And (Ok(Page_Num,Np)) Do
      Begin
        Desemp_Sq(Aux[K],G,H,T);     (* Desempilha de SQ *)

        (* Insere patriarcas dos descendentes do Nodo recem
           colocado na Pagina em SQ *)
        If Aux[K].Pag_Esq <> Nulo
          Then
            Begin
```

```
              Tmp := Aux[K];
              Encontrar(Tmp,Esq);
              Insere_Sq(Tmp,Page_Num,K,Esq);
            End;
        If Aux[K].Pag_Dir <> Nulo
          Then
            Begin
              Tmp := Aux[K];
              Encontrar(Tmp,Dir);
              Insere_Sq(Tmp,Page_Num,K,Dir);
            End;
        Marcar_Pai(Aux,G,H,T,Page_Num,K);
        K := K  + 1;
      End;
      (* Grava a Pagina otimizada *)
      if (Not Vazia_Sq)
        Then
          Gravar_Pagina(Page_Num,Aux,'dest.otm')
        Else
          If K = Tam_Pag
            Then
              Begin
                Gravar_Pagina(Page_Num,Aux,'dest.otm');
                K := 0;
                Page_Num := Page_Num + 1;     (* Cria nova pagina *)
                Inic_Page(Aux);
              End;
    Until Vazia_Sq;
    Repeat
      If K = Tam_Pag
        Then
          Begin
            Page_Num := Page_Num + 1;     (* Cria nova pagina *)
            Inic_Page(Aux);
            Esp_Disp := Tam_Pag;
            K := 0;
          End
        Else
          Esp_Disp := Tam_Pag - K;
      If Not Find_Best(Ind,Esp_Disp)
        Then
          Begin
            Gravar_Pagina(Page_Num,Aux,'dest.otm');
            K := Tam_Pag;
          End
        Else
          Begin
            Retira_Fl(Ind,Tmp,G,H,T);
            Armazena_Sub_Arv(Aux,Tmp,G,H,T,Page_Num,K);
            If K = Tam_Pag
              Then
                Gravar_Pagina(Page_Num,Aux,'dest.otm');
          End
    Until Vazia_Fl;
    If K < Tam_Pag
      Then
        Gravar_Pagina(Page_Num,Aux,'dest.otm');
End;

(* Procedimento que monta a arvore *)
```

```
Procedure Montar ( Key : Integer; Ch : Char);
Var
  No               : Nodo;
  Aux              : Nodo;
  Ok, Erro         : Boolean;
  C_Pag, C_Pos     : Integer;
  Aux_Pag, Aux_Pos : Integer;
Begin
  No.Chave   := Key;
  No.Info    := Ch;
  No.Pag_Dir := Nulo;
  No.Pos_Dir := Nulo;
  No.Pag_Esq := Nulo;
  No.Pos_Esq := Nulo;
  Vetor[Pag][Pos] := No;

  Ok    := False;
  Erro  := False;
  C_Pag := 0;
  C_Pos := 0;
  If Not ((Pag = 0) And (Pos = 0))
    Then
      Begin
        While ( (Not Ok) And ( Not Erro) ) Do
          Begin
            If (Vetor[C_Pag][C_Pos].Chave < Key)
              Then
                If (Eh_Folha(Vetor[C_Pag][C_Pos],Dir))
                  Then
                    Begin
                      Vetor[C_Pag][C_Pos].Pag_Dir := Pag;
                      Vetor[C_Pag][C_Pos].Pos_Dir := Pos;
                      Ok := True;
                    End
                  Else
                    Begin
                      Aux_Pag := Vetor[C_Pag][C_Pos].Pag_Dir;
                      Aux_Pos := Vetor[C_Pag][C_Pos].Pos_Dir;
                      C_Pag := Aux_Pag;
                      C_Pos := Aux_Pos;
                    End
              Else
                If (Vetor[C_Pag][C_Pos].Chave > Key)
                 Then
                   If (Eh_Folha(Vetor[C_Pag][C_Pos],Esq))
                     Then
                       Begin
                         Vetor[C_Pag][C_Pos].Pag_Esq := Pag;
                         Vetor[C_Pag][C_Pos].Pos_Esq := Pos;
                         Ok := True;
                       End
                     Else
                       Begin
                         Aux_Pag := Vetor[C_Pag][C_Pos].Pag_Esq;
                         Aux_Pos := Vetor[C_Pag][C_Pos].Pos_Esq;
                         C_Pag   := Aux_Pag;
                         C_Pos   := Aux_Pos;
                       End
                 Else
                   Begin
```

```
                    Erro := True
                  End;
          End;
      End;
  If (Erro)
   Then
     Writeln('ERRO DE INSERCAO')
   Else
     Prox(Pag,Pos);
End;

(* Procedimento que le a arvore a ser paginada *)
Procedure Ler_Arvore (Str : String);
Var
  Arq   : Text;
  Chave : Integer;
  Info  : Char;
Begin
  If Str <> ''
    Then
      Begin
        Assign(Arq,Str);
        Reset(Arq);
        While Not Eof(Arq) Do
          Begin
            Readln(Arq,Chave);
            Readln(Arq,Info);
            Montar(Chave,Info);
          End;
        Close(Arq);
      End;
End;

(* Calcula a distancia entre dois nodos *)
Function Dist (PagRaiz,PosRaiz,Pag1,Pos1,Pag2,Pos2 : Integer) : Real;
Begin
  If (PagRaiz <> Nulo) And (PosRaiz <> Nulo)
  Then
  If (Pag1 = Pag2) AND (Pos1 = Pos2)
  Then
      Dist := 0
  Else
  If (Pag1 = PagRaiz) AND (Pos1 = PosRaiz)
  Then
      If (Vetor[Pag2,Pos2].chave > Vetor[PagRaiz,PosRaiz].chave)
        Then
          Dist := Dist(Vetor[PagRaiz,PosRaiz].Pag_Dir,
                       Vetor[PagRaiz,PosRaiz].Pos_Dir,
                       Vetor[PagRaiz,PosRaiz].Pag_Dir,
                       Vetor[PagRaiz,PosRaiz].Pos_Dir,Pag2,Pos2) + 1
        Else
          Dist := Dist(Vetor[PagRaiz,PosRaiz].Pag_Esq,
                       Vetor[PagRaiz,PosRaiz].Pos_Esq,
                       Vetor[PagRaiz,PosRaiz].Pag_Esq,
                       Vetor[PagRaiz,PosRaiz].Pos_Esq,Pag2,Pos2) + 1
  Else
  If (Pag2 = PagRaiz) AND (Pos2 = PosRaiz)
  Then
      If (Vetor[Pag1,Pos1].chave > Vetor[PagRaiz,PosRaiz].chave)
        Then
```

```
          Dist := Dist(Vetor[PagRaiz,PosRaiz].Pag_Dir,
                       Vetor[PagRaiz,PosRaiz].Pos_Dir,
                       Pag1,Pos1,
                       Vetor[PagRaiz,PosRaiz].Pag_Dir,
                       Vetor[PagRaiz,PosRaiz].Pos_Dir) + 1
        Else
          Dist := Dist(Vetor[PagRaiz,PosRaiz].Pag_Esq,
                       Vetor[PagRaiz,PosRaiz].Pos_Esq,
                       Pag1,Pos1,
                       Vetor[PagRaiz,PosRaiz].Pag_Esq,
                       Vetor[PagRaiz,PosRaiz].Pos_Esq) + 1
  Else
      Begin
        If ((Vetor[Pag1,Pos1].chave > Vetor[PagRaiz,PosRaiz].chave) And
            (Vetor[Pag2,Pos2].chave < Vetor[PagRaiz,PosRaiz].chave) Or
            (Vetor[Pag1,Pos1].chave < Vetor[PagRaiz,PosRaiz].chave) And
            (Vetor[Pag2,Pos2].chave > Vetor[PagRaiz,PosRaiz].chave))
          Then
             Dist := Dist(PagRaiz,PosRaiz,PagRaiz,PosRaiz,Pag1,Pos1) +
                     Dist(PagRaiz,PosRaiz,PagRaiz,PosRaiz,Pag2,Pos2)
          Else
           If (Vetor[Pag1,Pos1].chave > Vetor[PagRaiz,PosRaiz].chave)
            Then
             Dist :=
Dist(Vetor[PagRaiz,PosRaiz].Pag_Dir,Vetor[PagRaiz,PosRaiz].Pos_Dir,
               Pag1,Pos1,Pag2,Pos2)
            Else
             Dist :=
Dist(Vetor[PagRaiz,PosRaiz].Pag_Esq,Vetor[PagRaiz,PosRaiz].Pos_Esq,
               Pag1,Pos1,Pag2,Pos2)
      End
   Else
     Dist := 0;
End;

(* Calcula a quantidadede das distancia internodais de todas as paginas *)
Function Metrica1 : Real;
Var
  I,J,K  : Integer;
  Soma   : Real;
Begin
  Soma := 0.0;
  For I := 0 To Pag Do
    For K := 0 to Tam_Pag-1 Do
      For J := K+1 to Tam_Pag-1 Do
        If (vetor[i,j].chave <> -1) and (vetor[i,k].chave <> -1)
        Then
          If (I <> Pag)
            Then
              Soma := Soma + Dist(0,0,i,k,i,j)
            Else
              If (K < Pos) And (J < Pos)
                Then
                  Soma := Soma + Dist(0,0,i,k,i,j);
  Metrica1 := Soma;
End;

(* Calcula o numero de paginas visitadas para localizar um nodo *)
Function Num_Pag (PagRaiz,PosRaiz,Info : Integer) : Real;
Begin
```

```
  If (PagRaiz <> Nulo) And (PosRaiz <> Nulo)
  Then
  If (Vetor[PagRaiz,PosRaiz].chave = Info)
    Then
      Num_Pag := 1
    Else
      If (Vetor[PagRaiz,PosRaiz].chave < Info)
        Then
          If (Vetor[PagRaiz,PosRaiz].Pag_Dir <> PagRaiz)
          Then
          Num_Pag :=
Num_Pag(Vetor[PagRaiz,PosRaiz].Pag_Dir,Vetor[PagRaiz,PosRaiz].Pos_Dir,
                  Info) + 1
          Else
          Num_Pag :=
Num_Pag(Vetor[PagRaiz,PosRaiz].Pag_Dir,Vetor[PagRaiz,PosRaiz].Pos_Dir,
                  Info)
        Else
          If (Vetor[PagRaiz,PosRaiz].Pag_Esq <> PagRaiz)
          Then
          Num_Pag :=
Num_Pag(Vetor[PagRaiz,PosRaiz].Pag_Esq,Vetor[PagRaiz,PosRaiz].Pos_Esq,
                  Info) + 1
          Else
          Num_Pag :=
Num_Pag(Vetor[PagRaiz,PosRaiz].Pag_Esq,Vetor[PagRaiz,PosRaiz].Pos_Esq,
                  Info)
   Else
     Num_Pag := 0;
End;

(* Calcula a quantidadede de paginas visitadas para localizacao dos nodos
*)
Function Metrica2 : Real;
Var
  I,J    : Integer;
  Soma   : Real;
Begin
  Soma := 0.0;
  For I := 0 To Pag Do
    For J := 0 to Tam_Pag-1 Do
     Begin
         if (I <> Pag)
          Then
            Begin
              If (vetor[i,j].chave <> -1)
                Then
                  Soma := Soma + Num_Pag(0,0,vetor[i,j].chave);
            End
          Else
            If (J < Pos)
              Then
                Begin
                  If (vetor[i,j].chave <> -1)
                    Then
                      Soma := Soma + Num_Pag(0,0,vetor[i,j].chave);
                End;
      End;
  Metrica2 := Soma;
End;
```

```
(* Calcula a somatoria da distancia internodal minima dados o numero de
nodos
    e tamanho da pagina, com rearranjo da arvore *)
Function Otima_dist_internodal (tamarv : integer;tampag : integer):longint;
var
  soma : longint;
  val : array [0..15] of longint;
begin
   val[0] := 0;    val[1] := 0;    val[2] := 1;    val[3] := 4;
   val[4] := 10;   val[5] := 18;   val[6] := 32;   val[7] := 48;
   val[8] := 71;   val[9] := 102;  val[10] := 133; val[11] := 170;
   val[12] := 217; val[13] := 266; val[14] := 321; val[15] := 378;
   soma := 0;
   if not (tampag in [3, 7, 15])
     then
       soma := 999999
     else
         soma := (tamarv div tampag) * val[tampag] + val[tamarv mod
tampag];
   otima_dist_internodal:= soma;
end;

(* Funcao que calcula a somatoria das paginas visitadas minima dados
   o numero de nodos e tamanho da pagina, com rearranjo da arvore *)
Function Otima_pagvisit (tamarv : integer;tampag : integer):longint;
var
  soma,i,aux : longint;
begin
   soma := 0;
   if not (tampag in [3, 7, 15])
     then
       soma := maxint
     else
       begin
         case tampag of
         3 : i := 4;
         7 : i := 8;
         15 : i := 16;
         end;
         soma := 0;
         aux := 1;
         while (tamarv >= tampag*pot(i,aux-1)) do
         begin
           soma := soma + aux*tampag*pot(i,aux-1);
           tamarv := tamarv - tampag*pot(i,aux-1);
           aux := aux + 1;
         end;
         soma := soma + tamarv*aux;
       end;
   otima_pagvisit:= soma;
end;

(* Le a arvore otimizada de um arquivo em armazena em vetor *)
Procedure Ler_Arvore_Otimizada;
Var
  I,J     : Integer;
  P       : Nodo;
  Arq     : File Of Nodo;
Begin
```

```
  I := 0;
  J := 0;
  Assign(Arq,'dest.otm');
  Reset(Arq);
  While Not Eof(Arq) Do
    Begin
      Read(Arq,P);
      Vetor[I,J] := P;
      Prox(I,J);
    End;
  Close(Arq);
  Pag := I;
  Pos := J;
End;

(* Calcula espacos em branco num dado esquema de paginacao *)
Function Calcula_Espacos(Str : String) : Longint;
Var
  Arq     : File Of Nodo;
  P       : Nodo;
  I,J     : Integer;
  Soma    : Longint;
Begin
  I := 0;
  J := 0;
  Soma := 0;
  Assign(Arq,Str);
  Reset(Arq);
  While Not Eof(Arq) Do
    Begin
      Read(Arq,P);
      If (P.Chave = -1)
        Then
          Soma := Soma + 1;
      Prox(I,J);
    End;
  Close(Arq);
  Calcula_Espacos := Soma;
End;

(* Procedimento que inicializa o sistema *)
Procedure Inicializar;
Begin
  Pag := 0;
  Pos := 0;
  Tot_Sq := 0;
  Tot_Fl := 0;
End;

(* Procedimento que gera os testes armazenando sob a forma de dados brutos
*)
Procedure Gera_testes;
Var
  M1a,M1b,M2a,M2b,M3a,M3b             : Real;
  I,J,Tam,N_ensaio                    : Integer;
  Nomearq,Nensaio                     : String;
  Arqdest1,Arqdest2,Arqdest3,Arqdest4 : Text;
Begin
  Repeat
    Writeln('DIGITE O NUMERO DE ENSAIOS : ');
```

```
    Readln(N_ensaio);
  Until (N_ensaio In [1..100]);
  case NN of
    2 : Tam := 40;
    3 : Tam := 90;
    4 : Tam := 200;
  end;
  Assign(Arqdest1,'Alg_1.txt');
  (* Distancia internodal - Pag. Sequencial e Algoritmo Proposto *)
  Assign(Arqdest2,'Alg_2.txt');
  (* Numero de paginas visitadas - Pag. Sequencial e Algoritmo Proposto *)
  Assign(Arqdest3,'Alg_3.txt');
  (* Espacos em branco - Pag. Sequencial e Algoritmo Proposto *)
  Assign(Arqdest4,'Val_Otm.txt');
  (* Ideais teoricos - Distancia Internodal, Paginas Visitadas e Espacos *)
  Rewrite(Arqdest1);  Rewrite(Arqdest2);
  Rewrite(Arqdest3);  Rewrite(Arqdest4);
  For I := 1 to Tam do
  Begin
    For J := 0 to N_Ensaio-1 do
    Begin
      Inicializar;
      Str(I*10,nomearq);
      Str(J,nensaio);
      nomearq := 'a' + nomearq + 'e'+ nensaio + '.txt';
      Ler_Arvore(nomearq);
      M1a := Metrica1;
      M2a := Metrica2;
      Gravar(Pag*Tam_Pag + Pos);
      Otimizar;
      Ler_Arvore_Otimizada;
      If (I*10 mod Tam_Pag = 0)
        Then
          M3a := 0
        Else
          M3a := Tam_Pag - (I*10 mod Tam_Pag);
      M3b := Calcula_Espacos('dest.otm');
      M1b := Metrica1;
      M2b := Metrica2;
      Writeln(Arqdest1,M1a:10:0,M1b:10:0);
      Writeln(Arqdest2,M2a:10:0,M2b:10:0);
      Writeln(Arqdest3,M3a:10:0,M3b:10:0);
    End;
    Writeln(Arqdest4,I*10:5,Otima_dist_internodal(I*10,Tam_Pag):10,
            Otima_pagvisit(I*10,Tam_Pag):10,M3a:10:0);
    Writeln(I*10:5);
  End;
  Close(Arqdest1);  Close(Arqdest2);
  Close(Arqdest3);  Close(Arqdest4);
End;

Begin
  Repeat
    Writeln('DIGITE O NUMERO DE NIVEIS POR PAGINA (X) : ');
    Readln(NN);
  Until (NN In [2..4]);
  Tam_Pag := Pot(2,NN) - 1;
  Gera_testes;
End.
```

Anexo 3

Programa de Teste das Árvores B

```
(* ************************************************************************
   UNIVERSIDADE FEDERAL DO PARANA
   MESTRADO EM INFORMATICA
   ORIENTADOR: PROF. DR. ELIAS P. DUARTE JR

   TRABALHO DE DISSERTACAO DE MESTRADO
   PROGRAMA DE TESTE DAS ARVORES B - ADAPTADO DE (ZIVIANI, 1996)

   DESENVOLVIDO POR RUI ALBERTO ECKE TAVARES
   ************************************************************************
*)

{$M 65500,0,655360}

Const
  nn = 4;                    (* Numero de Niveis da Pagina *)
  m  = 8;                    (* Ordem m da Arvore B        *)
  mm = 16;                   (* Valor de 2*m               *)

(*
   Valores utilizados nos testes:
   nn = 2; m = 2; mm = 4;
   nn = 3; m = 3; mm = 6;
   nn = 3; m = 4; mm = 8;
   nn = 4; m = 7; mm = 14;
   nn = 4; m = 8; mm = 16;
*)

Type
  Registro = Record
               Chave : Integer;
               Info  : Char;
             End;
  Apontador = ^Pagina;
  Pagina = Record
             n : 0..mm;
             r : array[1..mm] of Registro;
             p : array[0..mm] of Apontador;
           End;

Var
  Arv    : Apontador;
  RegAux : Registro;

(* Procedimento que inicializa a Arvore B *)
Procedure Inicializa (Var Ap : Apontador);
Begin
  Ap := Nil;
End;

(* Procedimento que realiza uma pesquisa na Arvore B *)
Procedure Pesquisa (Var x : Registro; Var Ap : Apontador);
Var
  I    : Integer;
Begin
  If Ap = Nil
    Then
      Writeln('Registro nao esta presente na arvore')
    Else
      With Ap^ Do
```

```
        Begin
          I := 1;
          While (i < n) and (x.chave > r[i].chave) do
            i := i + 1;
          If x.chave = r[i].chave
            Then
              x := r[i]
            Else
              If x.chave < r[i].chave
                Then
                  Pesquisa(x,p[i-1])
                Else
                  Pesquisa(x,p[i])
         End;
End;

(* Funcao que conta a quantidade de paginas produzidas pela Arvore B *)
Function Pesq_Cont_Pag (x : Integer; Var Ap : Apontador) : Longint;
Var
  I    : Integer;
Begin
  If Ap = Nil
    Then
      Begin
        Pesq_Cont_Pag := 0;
        Writeln('Registro nao esta presente na arvore');
      End
    Else
      With Ap^ Do
        Begin
          I := 1;
          While (i < n) and (x > r[i].chave) do
            i := i + 1;
          If x = r[i].chave
            Then
              Pesq_Cont_Pag := 1
            Else
              If x < r[i].chave
                Then
                  Pesq_Cont_Pag := Pesq_Cont_Pag(x,p[i-1]) + 1
                Else
                  Pesq_Cont_Pag := Pesq_Cont_Pag(x,p[i]) + 1;
         End;
End;

(* Procedimento que mostra em ordem o conteudo da Arvore B *)
Procedure Mostra_Em_Ordem (Var Ap : Apontador);
Var
  I    : Integer;
Begin
  If Ap <> Nil
    Then
      Begin
        I := 1;
        While (i <= Ap^.n) do
          Begin
            If (Ap^.p[i-1] <> Nil)
              Then
                Mostra_Em_Ordem(Ap^.p[i-1]);
            Writeln(Ap^.r[i].chave);
```

```
            i := i + 1;
          End;
        Mostra_Em_Ordem(Ap^.p[i-1]);
      End;
End;

(* Procedimento que libera as posicoes de memoria ocupadas pela Arvore B *)
Procedure Libera (Var Ap : Apontador);
Var
  I    : Integer;
Begin
  If Ap <> Nil
    Then
      Begin
        I := 1;
        While (i <= Ap^.n) do
          Begin
            If (Ap^.p[i-1] <> Nil)
              Then
                Libera(Ap^.p[i-1]);
            i := i + 1;
          End;
        Libera(Ap^.p[i-1]);
        Dispose(Ap);
      End;
End;

(* Procedimento que finaliza a Arvore B *)
Procedure Finaliza;
Begin
  Libera(Arv);
End;

(* Conjunto de procedimentos que realiza a insercao na Arvore B *)
Procedure Inserenapagina (Ap : Apontador; Reg : Registro;
                          ApDir : Apontador);
Var
  Naoachouposicao : Boolean;
  k               : Integer;
Begin
  With Ap^ Do
    Begin
      k := n;
      Naoachouposicao := k > 0;
      While Naoachouposicao Do
        If Reg.chave < r[k].chave
          Then
            Begin
              r[k+1] := r[k];
              p[k+1] := p[k];
              k := k - 1;
              If k < 1
                Then
                  Naoachouposicao := False;
            End
          Else
            Naoachouposicao := False;
      r[k+1] := Reg;
      p[k+1] := ApDir;
      n := n + 1;
```

```
    End;
End;

Procedure Insere (Reg : Registro; Var Ap : Apontador);
Var
  Cresceu    : Boolean;
  RegRetorno : Registro;
  ApRetorno  : Apontador;
  ApTemp     : Apontador;

Procedure Ins (Reg : Registro; Ap : Apontador; Var Cresceu : Boolean;
               Var RegRetorno : Registro; Var ApRetorno : Apontador);
Var
  i,j    : Integer;
  ApTemp : Apontador;
Begin
  If Ap = Nil
    Then
      Begin
        Cresceu := True;
        RegRetorno := Reg;
        ApRetorno := Nil;
      End
    Else
      With Ap^ Do
       Begin
          i := 1;
          While (i < n) and (Reg.chave > r[i].chave) Do
            i := i + 1;
          If Reg.chave = r[i].chave
            Then
              Begin
                Writeln('Erro: Registro ja esta presente');
                Cresceu := False;
              End
            Else
              Begin
                If Reg.chave < r[i].chave
                  Then
                    Ins(Reg,p[i-1], Cresceu, RegRetorno, ApRetorno)
                  Else
                    Ins(Reg,p[i], Cresceu, RegRetorno, ApRetorno);
                If Cresceu
                  Then
                    If n < mm
                      Then
                        Begin  (* Pagina tem espaco *)
                          Inserenapagina(Ap, RegRetorno, ApRetorno);
                          Cresceu := False;
                        End
                      Else
                        Begin  (* Overflow: pagina tem que ser dividida *)
                          New(ApTemp);
                          ApTemp^.n := 0;
                          ApTemp^.p[0] := Nil;
                          If i <= m+1
                            Then
                              Begin
                                Inserenapagina(ApTemp, r[mm], p[mm]);
                                n := n - 1;
```

```
                                Inserenapagina(Ap, RegRetorno, ApRetorno)
                              End
                            Else
                              Inserenapagina(ApTemp, RegRetorno,
ApRetorno);
                          For j := m+2 to mm do
                            Inserenapagina (ApTemp, r[j], p[j]);
                          n := m;
                          ApTemp^.p[0] := p[m+1];
                          RegRetorno := r[m+1];
                          ApRetorno := ApTemp;
                        End;
          End;
        End;
End;

Begin
  Ins(Reg, Ap, Cresceu, RegRetorno, ApRetorno);
  If Cresceu   (* Arvore cresce na altura pela raiz *)
    Then
      Begin
        new(ApTemp);
        ApTemp^.n := 1;
        ApTemp^.r[1] := RegRetorno;
        ApTemp^.p[1] := ApRetorno;
        ApTemp^.p[0] := Ap;
        Ap := ApTemp;
      End;
End;

(* Procedimento que le a arvore a ser paginada *)
Procedure Ler_Arvore (Str : String);
Var
  Arq    : Text;
  Chave  : Integer;
  Info   : Char;
  RegAux : Registro;
Begin
  If Str <> ''
    Then
      Begin
        Assign(Arq,Str);
        Reset(Arq);
        While Not Eof(Arq) Do
          Begin
            Readln(Arq,Chave);
            Readln(Arq,Info);
            RegAux.Chave := Chave;
            RegAux.Info  := Info;
            Insere(RegAux, Arv);
          End;
        Close(Arq);
      End;
End;

(* Procedimento que percorre a Arvore B, computando as paginas visitadas *)
Procedure Percorre1 (Var Soma : Real;Var Ap : Apontador);
Var
  I    : Integer;
Begin
```

```
  If Ap <> Nil
    Then
      Begin
        I := 1;
        While (i <= Ap^.n) do
          Begin
            If (Ap^.p[i-1] <> Nil)
              Then
                Percorre1(Soma,Ap^.p[i-1]);
            Soma := Soma  + Pesq_Cont_Pag(Ap^.r[i].chave,Arv);
            i := i + 1;
          End;
        Percorre1(Soma,Ap^.p[i-1]);
      End;
End;

(* Funcao que retorna o numero de paginas visitadas na Arvore B *)
Function Calc_Pag_Visit : Real;
Var
  Soma : Real;
Begin
  Soma := 0;
  Percorre1(Soma,Arv);
  Calc_Pag_Visit := Soma;
End;

(* Procedimento que percorre a Arvore B, computando espacos em branco *)
Procedure Percorre2 (Var Soma : Real;Var Ap : Apontador);
Var
  I    : Integer;
Begin
  If Ap <> Nil
    Then
      Begin
        I := 1;
        While (i <= Ap^.n) do
          Begin
            If (Ap^.p[i-1] <> Nil)
              Then
                Percorre2(Soma,Ap^.p[i-1]);
            i := i + 1;
          End;
        Percorre2(Soma,Ap^.p[i-1]);
        Soma := Soma + mm - Ap^.n;
      End;
End;

(* Funcao que retorna o numero de espacos nas paginas da Arvore B *)
Function Calc_Espacos : Real;
Var
  Soma : Real;
Begin
  Soma := 0;
  Percorre2(Soma,Arv);
  Calc_Espacos := Soma;
End;

(* Procedimento que percorre a Arvore B, computando o numero de paginas *)
Procedure Percorre3 (Var Soma : Real;Var Ap : Apontador);
Var
```

```
  I    : Integer;
Begin
  If Ap <> Nil
    Then
      Begin
        I := 1;
        While (i <= Ap^.n) do
          Begin
            If (Ap^.p[i-1] <> Nil)
              Then
                Percorre3(Soma,Ap^.p[i-1]);
            i := i + 1;
          End;
        Percorre3(Soma,Ap^.p[i-1]);
        Soma := Soma + 1;
      End;
End;

(* Funcao que retorna o numero de paginas da Arvore B *)
Function Calc_Paginas : Real;
Var
  Soma : Real;
Begin
  Soma := 0;
  Percorre3(Soma,Arv);
  Calc_Paginas := Soma;
End;

(* Procedimento que coordena a realizacao dos testes do sistema *)
Procedure Gera_testes;
Var
  Media,M,NPag                          : Real;
  I,J,Tam,N_ensaio           : Integer;
  Nomearq,Nensaio            : String;
  Arqdest1,Arqdest2,Arqdest3 : Text;
Begin
  Repeat
    Writeln('DIGITE O NUMERO DE ENSAIOS : ');
    Readln(N_ensaio);
  Until (N_ensaio In [1..100]);
  case NN of
    2 : Tam := 40;
    3 : Tam := 90;
    4 : Tam := 200;
  end;
  Assign(Arqdest1,'Arv_B_1.txt');
  (* Numero de paginas visitadas *)
  Assign(Arqdest2,'Arv_B_2.txt');
  (* Espacos em branco *)
  Assign(Arqdest3,'Arv_B_3.txt');
  (* Quantidade de paginas x tamanho da pagina *)
  Rewrite(Arqdest1);
  Rewrite(Arqdest2);
  Rewrite(Arqdest3);
  For I := 1 to Tam do
  Begin
    For J := 0 to N_Ensaio-1 do
    Begin
      Inicializa(Arv);
      Str(I*10,nomearq);
```

```
      Str(J,nensaio);
      nomearq := 'a' + nomearq + 'e'+ nensaio + '.txt';
      Ler_Arvore(nomearq);
      Writeln(Arqdest1,Calc_Pag_Visit:10:0);
      Writeln(Arqdest2,Calc_Espacos:10:0);
      Writeln(Arqdest3,Calc_Paginas*mm:10:0);
      Finaliza;
    End;
    Writeln(I*10:5);
  End;
  Close(Arqdest1);
  Close(Arqdest2);
  Close(Arqdest3);
End;

Begin
  Gera_Testes;
End.
```

Anexo 4

Resultados Experimentais Obtidos

Tabela. Resultados para o número de páginas visitadas com tamanho de página 3 nodos.

Qtde. Nodos	Algoritmo	Seqüencial	Ideal	Arvore B Ordem 2
10	18,75	20,31	17,00	18,00
20	47,55	58,35	42,00	47,65
30	83,11	106,58	72,00	81,59
40	119,96	160,45	102,00	107,81
50	160,42	217,59	132,00	134,68
60	202,60	280,85	162,00	161,28
70	248,80	348,71	199,00	202,33
80	290,16	414,03	239,00	272,03
90	341,09	486,87	279,00	326,99
100	384,06	555,30	319,00	366,19
110	434,91	637,85	359,00	402,97
120	493,30	707,26	399,00	439,39
130	530,56	787,42	439,00	475,73
140	583,36	874,92	479,00	512,45
150	639,04	951,73	519,00	548,52
160	687,07	1040,26	559,00	584,43
170	735,22	1110,83	599,00	620,96
180	789,51	1193,51	639,00	657,39
190	850,08	1285,84	679,00	693,80
200	892,47	1362,82	719,00	729,72
210	955,69	1451,87	759,00	766,49
220	1027,52	1558,71	799,00	802,91
230	1076,96	1645,44	839,00	839,56
240	1116,20	1716,90	879,00	883,01
250	1184,76	1833,89	919,00	924,68
260	1249,24	1921,79	964,00	997,63
270	1304,50	2028,93	1014,00	1063,04
280	1370,72	2129,54	1064,00	1119,15
290	1413,04	2201,31	1114,00	1211,16
300	1479,14	2301,79	1164,00	1303,18
310	1552,07	2430,17	1214,00	1377,25
320	1588,25	2496,28	1264,00	1463,24
330	1645,88	2598,24	1314,00	1499,50
340	1714,38	2686,24	1364,00	1565,55
350	1772,44	2787,29	1414,00	1621,16
360	1852,17	2911,21	1464,00	1671,27
370	1910,67	2984,69	1514,00	1717,35
380	1941,82	3078,20	1564,00	1763,69
390	2033,19	3174,66	1614,00	1810,30
400	2066,98	3300,96	1664,00	1857,39

Tabela. Resultados para o número de páginas visitadas com tamanho de página 7 nodos.

Qtde. Nodos	Algoritmo	Seqüencial	Ideal	Arvore B Ordem 3	Arvore B Ordem 4
10	13,00	13,00	13,00	19,00	19,00
20	34,50	37,17	33,00	36,90	37,66
30	58,98	70,69	53,00	54,87	56,27
40	85,81	109,18	73,00	105,67	74,68
50	114,94	152,00	93,00	140,09	93,10
60	142,87	200,86	113,00	167,97	142,72
70	176,42	250,62	140,00	195,21	198,26
80	203,10	301,10	170,00	222,91	227,35
90	240,62	358,41	200,00	250,32	256,07
100	270,44	412,85	230,00	278,18	284,41
110	303,87	475,00	260,00	305,88	312,65
120	345,11	532,87	290,00	333,51	340,57
130	370,13	599,99	320,00	361,57	368,17
140	407,49	662,47	350,00	388,68	396,37
150	444,23	733,35	380,00	416,42	424,75
160	477,90	794,31	410,00	448,81	453,08
170	511,08	865,08	440,00	495,38	481,53
180	548,32	925,04	470,00	549,36	509,90
190	590,47	1000,23	500,00	614,07	537,89
200	621,33	1065,32	530,00	689,88	566,41
210	660,72	1135,81	560,00	757,75	594,47
220	710,59	1224,12	590,00	822,07	622,86
230	742,66	1290,37	620,00	860,00	650,88
240	772,87	1361,25	650,00	902,13	678,98
250	821,00	1446,29	680,00	941,52	706,88
260	864,63	1527,96	710,00	979,36	735,19
270	902,22	1603,09	740,00	1017,34	763,10
280	945,00	1691,85	770,00	1055,04	791,54
290	975,57	1748,92	800,00	1092,35	819,97
300	1023,04	1838,06	830,00	1130,12	847,71
310	1072,58	1929,89	860,00	1167,41	876,26
320	1095,47	1991,34	890,00	1205,54	904,33
330	1135,51	2076,20	920,00	1242,53	932,91
340	1180,21	2165,99	950,00	1280,52	964,54
350	1221,70	2248,10	980,00	1318,15	1000,01
360	1275,88	2326,95	1010,00	1355,58	1049,82
370	1315,69	2414,48	1040,00	1392,90	1090,56
380	1338,91	2479,12	1070,00	1430,45	1150,02
390	1401,50	2581,18	1100,00	1467,77	1184,45
400	1421,57	2672,21	1130,00	1505,58	1282,09
410	1472,46	2742,32	1160,00	1542,67	1351,06
420	1516,45	2821,83	1190,00	1580,47	1463,41
430	1568,00	2953,24	1220,00	1618,28	1554,08
440	1616,94	3021,20	1250,00	1655,76	1638,38
450	1631,71	3092,94	1280,00	1693,47	1666,07
460	1685,57	3181,37	1310,00	1730,82	1735,58
470	1739,45	3270,94	1340,00	1768,35	1792,07
480	1793,88	3359,38	1370,00	1805,69	1835,12

490	1827,66	3458,65	1400,00	1843,57	1867,99
500	1862,06	3566,81	1430,00	1881,41	1910,99
510	1921,21	3644,73	1460,00	1919,00	1949,61
520	1961,83	3746,53	1499,00	1956,48	1987,39
530	1987,99	3811,28	1539,00	1993,32	2025,52
540	2033,90	3917,92	1579,00	2031,51	2064,09
550	2083,97	4037,92	1619,00	2069,08	2101,94
560	2125,46	4068,13	1659,00	2106,36	2140,41
570	2162,22	4176,46	1699,00	2144,37	2178,37
580	2228,33	4279,89	1739,00	2182,14	2216,81
590	2243,95	4390,00	1779,00	2219,46	2255,25
600	2291,16	4452,70	1819,00	2257,81	2293,40
610	2360,42	4571,94	1859,00	2294,48	2331,81
620	2405,32	4630,52	1899,00	2332,19	2369,96
630	2441,70	4758,33	1939,00	2369,74	2408,16
640	2501,88	4854,73	1979,00	2407,23	2446,35
650	2546,80	4927,28	2019,00	2445,24	2484,44
660	2585,34	5034,32	2059,00	2482,78	2522,66
670	2611,50	5126,86	2099,00	2520,92	2561,24
680	2688,53	5231,54	2139,00	2557,28	2599,18
690	2722,79	5342,61	2179,00	2595,47	2637,44
700	2768,27	5426,64	2219,00	2632,69	2675,24
710	2816,04	5500,64	2259,00	2670,64	2713,35
720	2879,10	5623,85	2299,00	2707,40	2752,22
730	2883,45	5693,69	2339,00	2744,72	2790,56
740	2960,27	5824,85	2379,00	2782,66	2828,55
750	3008,62	5918,72	2419,00	2819,92	2866,64
760	3067,08	6015,38	2459,00	2866,08	2904,44
770	3104,20	6104,64	2499,00	2902,80	2943,09
780	3140,41	6216,96	2539,00	2949,02	2980,36
790	3203,12	6286,87	2579,00	3010,28	3019,74
800	3228,72	6426,99	2619,00	3023,59	3057,32
810	3275,00	6500,38	2659,00	3054,26	3094,89
820	3335,20	6593,10	2699,00	3124,49	3133,12
830	3350,52	6676,33	2739,00	3170,85	3171,82
840	3392,68	6787,32	2779,00	3267,69	3209,37
850	3490,42	6892,02	2819,00	3306,67	3247,70
860	3502,12	6987,84	2859,00	3327,78	3286,13
870	3579,91	7099,84	2899,00	3410,44	3324,65
880	3610,06	7186,89	2939,00	3503,08	3362,40
890	3638,95	7292,93	2979,00	3550,86	3400,19
900	3664,30	7356,04	3019,00	3680,41	3437,99

Tabela. Resultados para o número de páginas visitadas com tamanho de página 15 nodos.

Qtde. Nodos	Algoritmo	Seqüencial	Ideal	Arvore B Ordem 7	Arvore B Ordem 8
10	10,00	10,00	10,00	10,00	10,00
20	25,06	25,00	25,00	39,00	39,00
30	45,01	45,00	45,00	58,00	58,62
40	66,60	72,66	65,00	77,14	77,39
50	89,30	103,00	85,00	96,28	96,85
60	113,10	136,07	105,00	115,34	116,03
70	137,33	172,90	125,00	134,30	135,07
80	161,95	211,02	145,00	153,40	154,13
90	191,06	253,12	165,00	172,42	173,31
100	214,78	295,99	185,00	191,50	192,66
110	241,90	344,12	205,00	210,52	211,83
120	273,24	388,83	225,00	229,70	230,92
130	295,76	436,27	245,00	248,72	250,06
140	324,59	486,40	265,00	267,70	269,25
150	354,18	538,27	285,00	300,16	288,47
160	382,83	591,30	305,00	370,86	307,61
170	406,12	644,36	325,00	434,84	326,86
180	437,11	695,19	345,00	517,48	345,92
190	469,38	749,61	365,00	550,39	370,83
200	492,56	811,29	385,00	581,20	412,29
210	523,95	865,42	405,00	610,16	499,57
220	564,85	929,75	425,00	639,26	599,85
230	592,13	984,18	445,00	668,13	652,69
240	612,39	1048,57	465,00	697,18	699,88
250	650,27	1104,76	485,00	726,59	729,11
260	687,19	1167,27	510,00	755,47	758,28
270	718,15	1232,89	540,00	784,21	787,33
280	745,94	1301,32	570,00	813,44	816,64
290	776,74	1359,71	600,00	842,32	845,53
300	810,25	1433,03	630,00	871,25	874,97
310	851,21	1495,92	660,00	900,48	903,91
320	863,09	1560,99	690,00	928,72	933,24
330	901,16	1609,08	720,00	957,58	962,24
340	936,77	1694,87	750,00	986,45	991,37
350	963,96	1768,96	780,00	1015,44	1020,46
360	1008,65	1828,72	810,00	1044,46	1050,11
370	1039,21	1896,89	840,00	1073,17	1078,72
380	1058,32	1970,66	870,00	1102,35	1107,84
390	1103,34	2036,25	900,00	1131,13	1137,07
400	1120,94	2110,97	930,00	1160,28	1165,92
410	1159,52	2182,97	960,00	1189,45	1194,81
420	1195,26	2246,18	990,00	1218,30	1223,78
430	1234,60	2332,29	1020,00	1247,29	1252,83
440	1274,35	2407,99	1050,00	1276,45	1281,72
450	1288,82	2470,33	1080,00	1305,32	1310,68
460	1318,96	2553,44	1110,00	1334,62	1340,06
470	1361,87	2605,58	1140,00	1363,65	1369,21
480	1410,94	2679,69	1170,00	1392,72	1398,16

490	1440,34	2761,59	1200,00	1421,53	1426,90
500	1467,89	2856,16	1230,00	1450,54	1455,95
510	1500,58	2922,63	1260,00	1479,84	1485,38
520	1538,96	3005,30	1290,00	1508,72	1514,51
530	1565,17	3068,57	1320,00	1537,49	1543,68
540	1596,88	3165,44	1350,00	1566,66	1572,81
550	1639,01	3247,34	1380,00	1595,65	1602,30
560	1667,71	3292,16	1410,00	1624,77	1631,29
570	1701,19	3377,60	1440,00	1653,73	1660,49
580	1744,38	3462,61	1470,00	1682,57	1689,63
590	1758,81	3550,03	1500,00	1711,16	1718,72
600	1795,27	3604,43	1530,00	1740,93	1747,87
610	1851,55	3704,24	1560,00	1769,57	1777,17
620	1882,29	3742,93	1590,00	1798,04	1805,85
630	1911,62	3841,68	1620,00	1826,86	1835,24
640	1960,43	3914,81	1650,00	1855,80	1864,80
650	2002,65	3999,40	1680,00	1884,54	1893,48
660	2026,22	4102,46	1710,00	1914,13	1922,86
670	2035,74	4200,16	1740,00	1942,83	1951,93
680	2105,72	4239,96	1770,00	1971,23	1980,91
690	2137,76	4336,73	1800,00	2000,19	2010,36
700	2166,45	4432,97	1830,00	2029,11	2039,08
710	2201,91	4501,19	1860,00	2057,95	2068,61
720	2258,35	4572,69	1890,00	2087,02	2097,80
730	2251,82	4652,61	1920,00	2116,60	2126,26
740	2320,51	4738,21	1950,00	2145,12	2155,61
750	2348,30	4836,14	1980,00	2173,87	2185,34
760	2393,61	4916,67	2010,00	2203,26	2213,73
770	2431,21	4985,79	2040,00	2231,56	2243,00
780	2462,01	5090,64	2070,00	2260,79	2271,89
790	2498,08	5147,47	2100,00	2289,94	2300,44
800	2528,73	5249,61	2130,00	2319,17	2329,93
810	2556,92	5333,11	2160,00	2348,05	2358,63
820	2601,18	5434,00	2190,00	2376,76	2387,80
830	2610,75	5497,55	2220,00	2405,91	2416,37
840	2648,13	5563,27	2250,00	2434,26	2445,99
850	2722,12	5686,51	2280,00	2463,86	2475,16
860	2725,90	5765,83	2310,00	2493,64	2503,58
870	2796,21	5847,89	2340,00	2522,27	2533,15
880	2809,67	5943,51	2370,00	2551,10	2562,13
890	2841,96	6021,01	2400,00	2579,81	2590,97
900	2851,87	6079,42	2430,00	2609,17	2619,87
910	2891,88	6180,36	2460,00	2637,36	2649,17
920	2952,69	6267,95	2490,00	2666,85	2678,09
930	2957,91	6362,50	2520,00	2695,71	2707,32
940	3025,31	6469,20	2550,00	2724,87	2736,17
950	3068,31	6554,78	2580,00	2754,34	2765,10
960	3133,72	6643,34	2610,00	2783,51	2794,26
970	3131,08	6704,27	2640,00	2811,36	2823,40
980	3166,39	6820,85	2670,00	2841,24	2852,55
990	3197,93	6884,75	2700,00	2870,27	2881,71

1000	3255,11	6987,85	2730,00	2899,36	2910,68
1010	3287,94	7052,78	2760,00	2928,06	2939,57
1020	3323,09	7166,92	2790,00	2957,17	2969,37
1030	3351,39	7263,60	2820,00	2986,70	2998,62
1040	3427,31	7334,78	2850,00	3015,46	3027,25
1050	3445,27	7451,80	2880,00	3044,00	3056,58
1060	3465,08	7527,96	2910,00	3073,62	3085,34
1070	3488,66	7603,14	2940,00	3102,08	3114,56
1080	3550,54	7714,33	2970,00	3131,06	3143,91
1090	3591,81	7833,04	3000,00	3160,26	3173,27
1100	3615,30	7898,88	3030,00	3189,08	3202,36
1110	3667,32	7988,80	3060,00	3218,16	3231,65
1120	3723,22	8064,28	3090,00	3246,79	3260,61
1130	3759,12	8162,27	3120,00	3276,01	3289,49
1140	3750,33	8275,91	3150,00	3304,80	3318,43
1150	3817,09	8395,83	3180,00	3334,36	3347,64
1160	3912,25	8421,48	3210,00	3362,74	3377,02
1170	3905,41	8528,37	3240,00	3391,77	3406,49
1180	3903,46	8624,16	3270,00	3420,90	3435,23
1190	3941,54	8697,20	3300,00	3449,63	3464,27
1200	3992,40	8800,67	3330,00	3479,11	3493,51
1210	4046,42	8917,50	3360,00	3507,52	3522,80
1220	4071,22	8977,59	3390,00	3536,59	3552,20
1230	4123,10	9043,48	3420,00	3565,47	3581,60
1240	4163,88	9194,42	3450,00	3593,90	3610,44
1250	4182,00	9276,56	3480,00	3623,33	3639,26
1260	4208,20	9337,64	3510,00	3652,09	3668,69
1270	4251,74	9420,07	3540,00	3680,96	3697,54
1280	4320,23	9523,22	3570,00	3709,88	3726,82
1290	4351,67	9639,53	3600,00	3738,91	3756,47
1300	4407,38	9704,28	3630,00	3767,44	3785,45
1310	4400,92	9823,18	3660,00	3796,84	3813,71
1320	4500,74	9958,43	3690,00	3825,67	3844,10
1330	4490,02	10026,64	3720,00	3853,80	3872,44
1340	4520,52	10079,07	3750,00	3883,54	3901,51
1350	4588,79	10255,67	3780,00	3912,58	3930,78
1360	4626,32	10312,37	3810,00	3941,78	3960,07
1370	4701,86	10389,57	3840,00	3970,40	3989,10
1380	4701,78	10493,06	3870,00	4012,79	4017,89
1390	4791,19	10604,77	3900,00	4027,78	4047,45
1400	4743,43	10749,91	3930,00	4056,63	4076,97
1410	4837,82	10802,99	3960,00	4085,36	4105,38
1420	4855,66	10868,57	3990,00	4129,25	4134,18
1430	4879,98	10941,78	4020,00	4157,59	4163,63
1440	4929,08	11039,83	4050,00	4172,80	4192,21
1450	4997,10	11161,08	4080,00	4216,52	4221,80
1460	5033,79	11285,76	4110,00	4274,45	4251,00
1470	5032,77	11353,49	4140,00	4332,58	4279,84
1480	5100,20	11493,77	4170,00	4347,42	4308,42
1490	5129,00	11581,69	4200,00	4392,10	4338,01
1500	5159,11	11664,32	4230,00	4481,27	4367,29

1510	5175,90	11718,89	4260,00	4601,35	4396,68
1520	5208,62	11832,60	4290,00	4647,33	4425,20
1530	5320,56	11941,37	4320,00	4555,29	4454,34
1540	5314,36	12003,00	4350,00	4662,54	4483,31
1550	5354,35	12139,67	4380,00	4739,51	4512,93
1560	5440,75	12278,20	4410,00	4862,86	4541,06
1570	5447,39	12304,46	4440,00	4925,71	4570,18
1580	5461,24	12431,02	4470,00	5003,97	4599,72
1590	5570,88	12476,84	4500,00	5035,97	4628,51
1600	5624,80	12619,36	4530,00	5067,63	4657,13
1610	5585,64	12718,81	4560,00	5163,67	4686,81
1620	5629,77	12827,55	4590,00	5228,52	4715,30
1630	5684,88	12927,75	4620,00	5277,38	4744,41
1640	5721,19	13026,72	4650,00	5538,46	4773,42
1650	5792,38	13141,88	4680,00	5555,70	4803,05
1660	5843,43	13247,15	4710,00	5423,88	4831,64
1670	5859,12	13337,44	4740,00	5706,74	4861,54
1680	5915,50	13423,62	4770,00	5540,19	4889,79
1690	5967,96	13523,38	4800,00	5825,80	4918,98
1700	5976,20	13628,11	4830,00	5894,48	4948,18
1710	5990,76	13685,21	4860,00	5809,66	4977,26
1720	6029,26	13774,91	4890,00	6084,05	5006,11
1730	6132,88	13894,91	4920,00	6085,09	5034,69
1740	6129,35	14044,78	4950,00	6276,59	5063,67
1750	6136,70	14061,54	4980,00	6295,20	5092,84
1760	6212,48	14218,60	5010,00	6208,39	5122,20
1770	6219,73	14287,18	5040,00	6403,20	5151,10
1780	6277,29	14452,68	5070,00	6581,21	5180,52
1790	6318,59	14508,85	5100,00	6618,42	5209,17
1800	6377,10	14671,54	5130,00	6601,45	5238,40
1810	6450,91	14749,50	5160,00	6764,66	5267,88
1820	6415,03	14897,98	5190,00	6711,32	5296,40
1830	6496,52	14938,87	5220,00	6931,10	5325,69
1840	6508,38	15015,44	5250,00	6896,35	5354,10
1850	6547,35	15103,89	5280,00	6933,39	5384,48
1860	6585,45	15236,42	5310,00	7100,47	5412,97
1870	6622,58	15362,31	5340,00	7157,21	5441,62
1880	6700,53	15401,23	5370,00	7177,01	5471,31
1890	6704,90	15530,42	5400,00	7121,24	5500,47
1900	6782,72	15709,36	5430,00	7120,94	5529,69
1910	6833,55	15801,27	5460,00	7330,13	5557,95
1920	6828,02	15776,71	5490,00	7348,89	5587,36
1930	6918,99	15924,30	5520,00	7445,44	5616,44
1940	6948,96	16017,49	5550,00	7484,22	5645,31
1950	6935,26	16159,20	5580,00	7521,95	5674,98
1960	6989,29	16211,32	5610,00	7600,19	5703,77
1970	7032,85	16247,09	5640,00	7658,63	5732,92
1980	7075,71	16467,01	5670,00	7697,43	5761,82
1990	7133,21	16551,67	5700,00	7736,29	5790,89
2000	7176,10	16705,39	5730,00	7795,17	5820,55

Tabela. Resultados dos espaços não preenchidos com tamanho de página 3 nodos.

Qtde. Nodos	Algoritmo	Seqüencial	Arvore B Ordem 2
10	200	200	600
20	106	100	1200
30	9	0	1564
40	202	200	2076
50	104	100	2328
60	29	0	2688
70	202	200	3232
80	102	100	4040
90	17	0	4712
100	208	200	5124
110	106	100	5412
120	15	0	5844
130	200	200	6308
140	102	100	6620
150	3	0	7192
160	204	200	7828
170	100	100	8216
180	0	0	8644
190	200	200	9080
200	102	100	9712
210	3	0	10004
220	200	200	10436
230	104	100	10776
240	0	0	11288
250	200	200	11748
260	100	100	12384
270	0	0	12820
280	200	200	13280
290	100	100	13828
300	0	0	14608
310	200	200	15220
320	100	100	15716
330	0	0	15960
340	208	200	16324
350	104	100	17132
360	3	0	17492
370	202	200	18060
380	100	100	18524
390	0	0	18880
400	200	200	19044

Tabela. Resultados dos espaços não preenchidos com tamanho de página 7 nodos.

Qtde. Nodos	Algoritmo	Seqüencial	Arvore B Ordem 3	Arvore B Ordem 4
10	400	400	800	1400
20	100	100	1060	1472
30	503	500	1278	1584
40	200	200	2462	1856
50	600	600	2746	2120
60	304	300	3018	3288
70	0	0	3674	4224
80	400	400	4054	4520
90	106	100	4608	4544
100	500	500	4892	4872
110	200	200	5272	5280
120	600	600	5694	5944
130	300	300	5858	6864
140	14	0	6592	7304
150	400	400	6948	7600
160	100	100	7412	7936
170	500	500	7936	8176
180	200	200	8592	8480
190	600	600	9074	9088
200	300	300	9880	9272
210	0	0	10494	9824
220	400	400	11180	10112
230	100	100	11242	10696
240	500	500	11676	11216
250	200	200	12488	11896
260	600	600	12784	12248
270	300	300	12996	12920
280	0	0	13376	13168
290	400	400	13990	13424
300	100	100	14328	14232
310	500	500	14954	14392
320	200	200	15076	14936
330	600	600	15882	15072
340	300	300	16088	15496
350	0	0	16510	15816
360	400	400	17052	16536
370	100	100	17660	16568
380	500	500	18130	17344
390	200	200	18738	17528
400	600	600	19052	18632
410	300	300	19798	19088
420	0	0	20118	19960
430	400	400	20432	20528
440	100	100	20944	21216
450	500	500	21318	22048
460	200	200	21908	22296
470	600	600	22390	22776
480	300	300	22986	23104

490	0	0	23258	23880
500	400	400	23554	24408
510	100	100	24000	24512
520	500	500	24512	25288
530	200	200	25408	25784
540	600	600	25494	25928
550	300	300	25952	26648
560	0	0	26584	26872
570	400	400	26778	27504
580	100	100	27116	27752
590	500	500	27724	28000
600	200	200	27714	28480
610	600	600	28712	28752
620	300	300	29086	29232
630	0	0	29556	29672
640	400	400	30062	30120
650	100	100	30256	30648
660	500	500	30732	31072
670	200	200	30848	31208
680	600	600	32032	31856
690	300	300	32118	32248
700	0	0	32786	33008
710	400	400	33016	33520
720	100	100	33960	33424
730	500	500	34568	33752
740	200	200	34804	34360
750	600	600	35448	34888
760	300	300	35318	35648
770	0	0	36346	35728
780	400	400	36360	36912
790	100	100	36962	36408
800	500	500	37858	37344
810	200	200	37710	38288
820	600	600	38336	38704
830	300	300	38806	38744
840	0	0	39384	39704
850	400	400	39776	40040
860	100	100	40558	40296
870	500	500	40752	40480
880	200	200	40844	41280
890	600	600	41842	42048
900	300	300	42552	42808

Tabela. Resultados dos espaços não preenchidos com tamanho de página 15 nodos.

Qtde. Nodos	Algoritmo	Seqüencial	Arvore B Ordem 7	Arvore B Ordem 8
10	500	500	400	600
20	1000	1000	2200	2800
30	15	0	2600	2408
40	500	500	2804	3376
50	1000	1000	3008	3240
60	30	0	3324	3552
70	500	500	3780	4088
80	1000	1000	4040	4592
90	15	0	4412	4904
100	505	500	4700	4944
110	1000	1000	5072	5272
120	30	0	5220	5728
130	500	500	5592	6104
140	1007	1000	6020	6400
150	30	0	6602	6648
160	500	500	8010	7024
170	1000	1000	8634	7224
180	0	0	10126	7728
190	500	500	9980	8040
200	1000	1000	10520	8560
210	0	0	10976	10184
220	500	500	11236	11856
230	1000	1000	11818	11928
240	0	0	12148	12992
250	500	500	11974	13224
260	1000	1000	12542	13552
270	0	0	13306	14072
280	500	500	13384	14176
290	1000	1000	13952	14952
300	0	0	14450	14848
310	500	500	14528	15544
320	1000	1000	15992	15616
330	0	0	16588	16216
340	500	500	17170	16608
350	1000	1000	17584	17064
360	0	0	17956	16624
370	500	500	18762	17848
380	1000	1000	18910	18256
390	0	0	19618	18488
400	500	500	19808	19328
410	1000	1000	19970	20104
420	0	0	20580	20752
430	500	500	20994	21272
440	1000	1000	21170	22048
450	0	0	21752	22712
460	500	500	21732	22704
470	1000	1000	22090	23064
480	0	0	22392	23744

490	500	500	23058	24760
500	1000	1000	23444	25280
510	0	0	23424	25192
520	500	500	23992	25584
530	1000	1000	24714	25912
540	0	0	24876	26304
550	500	500	25290	26120
560	1000	1000	25522	26736
570	0	0	25978	27016
580	500	500	26602	27392
590	1000	1000	27576	27848
600	0	0	26898	28208
610	500	500	27802	28328
620	1000	1000	28944	29440
630	0	0	29596	29416
640	500	500	30080	29120
650	1000	1000	30844	30232
660	0	0	30418	30224
670	500	500	31238	30712
680	1000	1000	32478	31344
690	0	0	32934	31224
700	500	500	33446	32272
710	1000	1000	34070	32024
720	0	0	34372	32320
730	500	500	33960	33784
740	1000	1000	35032	33824
750	0	0	35782	33256
760	500	500	35636	34832
770	1000	1000	37016	35000
780	0	0	37094	35776
790	500	500	37284	37096
800	1000	1000	37362	36912
810	0	0	37930	37992
820	500	500	38736	38320
830	1000	1000	38926	39608
840	0	0	40236	39216
850	500	500	39796	39544
860	1000	1000	39104	41072
870	0	0	40022	40760
880	500	500	40660	41392
890	1000	1000	41466	42248
900	0	0	41362	43008
910	500	500	42896	43128
920	1000	1000	42610	43856
930	0	0	43206	44088
940	500	500	43382	44928
950	1000	1000	43124	45640
960	0	0	43286	45984
970	500	500	45296	46360
980	1000	1000	44464	46720
990	0	0	44822	47064

1000	500	500	45096	47712
1010	1000	1000	45916	48488
1020	0	0	46162	47808
1030	500	500	45820	48008
1040	1000	1000	46556	49200
1050	0	0	47600	49272
1060	500	500	47132	50256
1070	1000	1000	48288	50504
1080	0	0	48716	50544
1090	500	500	48836	50568
1100	1000	1000	49488	51024
1110	0	0	49776	51160
1120	500	500	50694	51824
1130	1000	1000	50786	52616
1140	0	0	51480	53312
1150	500	500	51096	53576
1160	1000	1000	52364	53568
1170	0	0	52722	53416
1180	500	500	52940	54432
1190	1000	1000	53718	54968
1200	0	0	53446	55184
1210	500	500	54672	55320
1220	1000	1000	54974	55280
1230	0	0	55542	55240
1240	500	500	56740	56096
1250	1000	1000	56538	56984
1260	0	0	57274	56896
1270	500	500	57856	57736
1280	1000	1000	58368	57888
1290	0	0	58726	57448
1300	500	500	59784	58080
1310	1000	1000	59624	59864
1320	0	0	60262	58240
1330	500	500	61880	59896
1340	1000	1000	61244	60384
1350	0	0	61588	60552
1360	500	500	61708	60688
1370	1000	1000	62640	61240
1380	0	0	63628	62176
1390	500	500	64308	61880
1400	1000	1000	64918	61648
1410	0	0	65696	63192
1420	500	500	65144	64112
1430	1000	1000	66608	63992
1440	0	0	66280	65264
1450	500	500	66386	64920
1460	1000	1000	67332	65200
1470	0	0	68558	66056
1480	500	500	68748	67328
1490	1000	1000	68630	66984
1500	0	0	69548	67136

1510	500	500	70620	67112
1520	1000	1000	70642	68480
1530	0	0	71266	68856
1540	500	500	71106	69504
1550	1000	1000	71310	69112
1560	0	0	72984	71104
1570	500	500	73062	71512
1580	1000	1000	74260	71248
1590	0	0	74240	72184
1600	500	500	74696	73392
1610	1000	1000	75236	72904
1620	0	0	75174	74320
1630	500	500	75224	74744
1640	1000	1000	77108	75328
1650	0	0	77578	74920
1660	500	500	77166	76176
1670	1000	1000	78252	75336
1680	0	0	77294	77136
1690	500	500	79150	77432
1700	1000	1000	79326	77712
1710	0	0	79376	78184
1720	500	500	80546	79024
1730	1000	1000	80582	80296
1740	0	0	81528	80928
1750	500	500	81900	81256
1760	1000	1000	81656	81280
1770	0	0	81888	82040
1780	500	500	83226	81968
1790	1000	1000	83332	83128
1800	0	0	83648	83360
1810	500	500	84412	83192
1820	1000	1000	84378	84560
1830	0	0	85086	84696
1840	500	500	84100	86240
1850	1000	1000	85144	84632
1860	0	0	86510	86048
1870	500	500	87148	87208
1880	1000	1000	87114	86704
1890	0	0	86702	87048
1900	500	500	87074	87296
1910	1000	1000	87894	89080
1920	0	0	88896	89024
1930	500	500	88848	89496
1940	1000	1000	88996	90304
1950	0	0	90614	89832
1960	500	500	90426	90768
1970	1000	1000	90924	91128
1980	0	0	91464	91888
1990	500	500	91920	92376
2000	1000	1000	92362	91920

Tabela. Resultados para a distância internodal com tamanho de página 3 nodos.

Qtde. Nodos	Seqüencial	Algoritmo	Alg. Alternativo	Ideal
10	22,65	15,98	18,90	12,00
20	80,46	42,18	52,95	25,00
30	164,90	79,29	90,68	40,00
40	242,27	101,38	125,68	52,00
50	337,41	137,29	172,43	65,00
60	459,98	182,29	215,75	80,00
70	561,68	205,78	255,43	92,00
80	671,97	247,43	309,06	105,00
90	803,17	294,75	345,02	120,00
100	912,30	313,20	383,33	132,00
110	1061,30	359,03	451,83	145,00
120	1183,25	412,13	482,53	160,00
130	1321,22	439,32	534,72	172,00
140	1461,29	480,82	589,96	185,00
150	1608,02	534,05	627,15	200,00
160	1747,85	564,28	676,97	212,00
170	1897,97	611,61	730,45	225,00
180	2031,32	672,82	781,93	240,00
190	2199,52	706,06	824,75	252,00
200	2342,00	736,64	876,97	265,00
210	2515,50	806,49	938,62	280,00
220	2680,77	828,68	987,88	292,00
230	2826,57	869,54	1035,65	305,00
240	2998,85	930,18	1068,95	320,00
250	3142,30	966,02	1108,30	332,00
260	3337,26	1012,25	1188,61	345,00
270	3531,03	1074,00	1234,80	360,00
280	3664,85	1097,54	1255,07	372,00
290	3812,65	1157,01	1350,36	385,00
300	4006,55	1211,44	1401,70	400,00
310	4218,58	1251,76	1452,55	412,00
320	4325,00	1283,06	1489,13	425,00
330	4498,35	1348,52	1521,82	440,00
340	4669,15	1379,16	1558,28	452,00
350	4872,63	1427,83	1653,63	465,00
360	5080,98	1494,05	1715,82	480,00
370	5196,42	1518,42	1736,32	492,00
380	5385,79	1572,70	1799,15	505,00
390	5638,40	1636,26	1861,12	520,00
400	5762,20	1677,20	1896,23	532,00

Tabela. Resultados para a distância internodal com tamanho de página 7 nodos.

Qtde. Nodos	Seqüencial	Algoritmo	Alg. Alternativo	Ideal
10	63,33	61,68	65,10	52,00
20	243,85	205,71	243,94	128,00
30	455,14	315,85	332,80	193,00
40	712,05	475,22	528,74	258,00
50	1028,78	659,92	633,85	336,00
60	1308,59	791,15	807,39	394,00
70	1718,72	1042,40	970,17	480,00
80	1989,03	1102,25	1105,56	532,00
90	2386,13	1328,25	1379,69	608,00
100	2715,77	1500,57	1424,28	673,00
110	3165,35	1690,78	1713,22	738,00
120	3524,62	1879,45	1793,00	816,00
130	3910,12	2026,78	2012,20	874,00
140	4429,08	2305,93	2234,90	960,00
150	4749,37	2412,16	2343,68	1012,00
160	5243,92	2645,62	2654,58	1088,00
170	5651,92	2774,21	2690,11	1153,00
180	6039,55	2951,70	2937,23	1218,00
190	6625,85	3240,84	3097,07	1296,00
200	6980,04	3406,78	3301,50	1354,00
210	7535,45	3652,14	3432,87	1440,00
220	7986,92	3763,56	3628,91	1492,00
230	8465,71	4027,67	3973,50	1568,00
240	8918,10	4164,43	3980,06	1633,00
250	9416,83	4421,14	4303,12	1698,00
260	10005,75	4703,38	4332,67	1776,00
270	10500,09	4797,56	4656,64	1834,00
280	11058,68	5109,42	4691,92	1920,00
290	11403,50	5146,74	4911,79	1972,00
300	11977,25	5443,63	5292,79	2048,00
310	12596,76	5673,99	5310,56	2113,00
320	12946,07	5873,76	5648,29	2178,00
330	13470,93	6090,94	5665,70	2256,00
340	13976,78	6209,14	5954,55	2314,00
350	14652,95	6518,86	6139,20	2400,00
360	15172,47	6606,42	6306,96	2452,00
370	15612,08	6918,71	6685,46	2528,00
380	16180,44	7105,53	6657,11	2593,00
390	16777,80	7327,84	7017,05	2658,00
400	17318,45	7658,66	7096,27	2736,00
410	17803,30	7684,66	7222,21	2794,00
420	18470,20	8070,18	7364,15	2880,00
430	19148,80	8204,54	7677,95	2932,00
440	19633,95	8392,54	7938,00	3008,00
450	20033,33	8652,36	8038,50	3073,00
460	20526,93	8717,54	8309,10	3138,00
470	21196,22	9106,10	8489,45	3216,00
480	21595,02	9177,98	8652,12	3274,00

490	22383,30	9581,58	8778,58	3360,00
500	23019,87	9655,00	9043,34	3412,00
510	23493,73	9951,53	9471,99	3488,00
520	24106,93	10096,96	9381,30	3553,00
530	24617,42	10338,52	9688,56	3618,00
540	25351,97	10667,10	9869,12	3696,00
550	25788,24	10790,67	10043,14	3754,00
560	26387,32	10992,30	10158,63	3840,00
570	26903,50	11174,26	10444,39	3892,00
580	27838,96	11666,03	10992,35	3968,00
590	28385,34	11688,24	10985,05	4033,00
600	28767,72	11871,84	11170,76	4098,00
610	29472,50	12308,04	11199,15	4176,00
620	29869,41	12265,96	11519,06	4234,00
630	30682,00	12729,10	11591,45	4320,00
640	31268,53	12840,12	12025,91	4372,00
650	31809,81	13032,75	12127,46	4448,00
660	32373,24	13230,17	12138,49	4513,00
670	32946,73	13386,10	12542,10	4578,00
680	33684,35	13850,32	12603,25	4656,00
690	34378,25	13879,54	13026,94	4714,00
700	34991,00	14316,00	13048,80	4800,00
710	35294,75	14275,90	13270,03	4852,00
720	36199,34	14681,94	13786,33	4928,00
730	36586,58	14882,17	13707,41	4993,00
740	37182,15	15028,02	13952,21	5058,00
750	38054,73	15338,50	13989,93	5136,00
760	38618,02	15572,65	14396,44	5194,00
770	39403,35	15927,50	14617,83	5280,00
780	39857,68	15894,40	14796,21	5332,00
790	40510,09	16183,94	15172,35	5408,00
800	41116,25	16482,51	14934,37	5473,00
810	41838,73	16729,48	15454,59	5538,00
820	42491,98	16944,82	15387,75	5616,00
830	43286,13	17235,68	15679,78	5674,00
840	43594,95	17414,98	15793,90	5760,00
850	44348,02	17645,76	16036,31	5812,00
860	44907,21	17929,14	16512,81	5888,00
870	45491,81	17941,07	16461,85	5953,00
880	45932,70	18168,24	16908,72	6018,00
890	46822,32	18478,82	16909,05	6096,00
900	47160,29	18619,21	17167,11	6154,00

Tabela. Resultados para a distância internodal com tamanho de página 15 nodos.

Qtde. Nodos	Seqüencial	Algoritmo	Alg. Alternativo	Ideal
10	138,81	139,19	138,81	133,00
20	474,87	468,63	483,08	396,00
30	1176,55	1116,98	1080,18	756,00
40	1569,30	1363,35	1517,34	889,00
50	2237,00	1848,25	1861,44	1152,00
60	3220,57	2582,91	2414,77	1512,00
70	3744,31	2906,36	3007,68	1645,00
80	4548,95	3374,16	3253,98	1908,00
90	5687,35	4267,30	3791,98	2268,00
100	6212,15	4510,76	4387,09	2401,00
110	7244,73	5099,61	4753,24	2664,00
120	8356,68	5915,39	5194,22	3024,00
130	9003,85	6394,23	6004,14	3157,00
140	9992,13	6884,31	6101,33	3420,00
150	11316,80	7901,60	6722,95	3780,00
160	11975,40	8219,69	7455,19	3913,00
170	12972,45	8605,05	7593,99	4176,00
180	14285,05	9655,82	8150,20	4536,00
190	15178,03	10119,30	8973,85	4669,00
200	16154,97	10665,98	9168,34	4932,00
210	17610,68	11714,24	9684,37	5292,00
220	18491,82	12086,55	10538,40	5425,00
230	19509,37	12714,10	10685,55	5688,00
240	20990,47	13818,74	11274,85	6048,00
250	21736,38	14087,92	12007,10	6181,00
260	23092,45	14767,78	12247,74	6444,00
270	24755,75	16119,30	12995,53	6804,00
280	25368,71	16029,06	13657,69	6937,00
290	26460,55	16763,34	13783,00	7200,00
300	28110,68	17930,10	14421,72	7560,00
310	29209,57	18598,52	15473,31	7693,00
320	29982,10	18769,74	15583,39	7956,00
330	31540,05	19971,54	16013,35	8316,00
340	32409,84	20412,69	17219,26	8449,00
350	33795,60	21100,18	17182,49	8712,00
360	35638,27	22173,38	17708,55	9072,00
370	36097,20	22527,16	18577,64	9205,00
380	37524,65	23177,82	18782,61	9468,00
390	39427,13	24272,10	19440,07	9828,00
400	40090,61	24580,80	20517,39	9961,00
410	41364,68	25305,68	20446,61	10224,00
420	43178,70	26423,36	21121,25	10584,00
430	44492,64	26846,83	22377,57	10717,00
440	45504,38	27239,98	22094,75	10980,00
450	47003,23	28795,64	22879,60	11340,00
460	47673,21	28487,38	23838,16	11473,00
470	49103,90	29504,60	23936,04	11736,00
480	50655,02	30782,92	24345,60	12096,00

490	51784,63	31098,60	25424,81	12229,00
500	53460,35	32181,24	25660,11	12492,00
510	54974,55	32975,80	26064,85	12852,00
520	56048,41	33112,76	27708,49	12985,00
530	57195,12	33960,18	27157,49	13248,00
540	59357,90	35276,74	28112,00	13608,00
550	59914,99	35403,64	28941,32	13741,00
560	61103,35	35709,02	28977,87	14004,00
570	63085,90	37350,50	29921,50	14364,00
580	64553,79	37764,70	30819,26	14497,00
590	65967,92	38178,52	30919,31	14760,00
600	67322,30	39457,60	31131,20	15120,00
610	68505,95	39753,86	32582,56	15253,00
620	69538,58	40151,48	32392,20	15516,00
630	71589,35	41648,76	33408,75	15876,00
640	72742,80	41704,10	34060,99	16009,00
650	73879,15	42620,60	34312,89	16272,00
660	75875,58	43715,54	34463,02	16632,00
670	76651,64	43708,89	36107,39	16765,00
680	78314,02	44639,36	35760,34	17028,00
690	80507,33	46318,22	36491,65	17388,00
700	81159,35	46355,32	37675,40	17521,00
710	82134,58	46840,62	37795,48	17784,00
720	84560,00	48465,14	38307,82	18144,00
730	85108,37	48424,21	39408,70	18277,00
740	86526,83	49390,72	39443,86	18540,00
750	88975,60	50215,12	39912,88	18900,00
760	89853,37	50688,77	41212,71	19033,00
770	91432,30	51455,78	41008,23	19296,00
780	93346,87	52447,28	41756,28	19656,00
790	94157,35	52660,98	42486,51	19789,00
800	95619,55	53795,16	42699,68	20052,00
810	97932,45	55285,04	43665,07	20412,00
820	98802,70	54824,63	44364,48	20545,00
830	100733,40	55693,52	44561,54	20808,00
840	101716,70	57123,64	44963,68	21168,00
850	103195,41	57337,37	46050,48	21301,00
860	104411,67	57829,92	46560,19	21564,00
870	106461,65	59076,46	46853,52	21924,00
880	106914,21	58908,42	48314,20	22057,00
890	108918,80	60114,20	48026,40	22320,00
900	110361,85	60223,44	48943,45	22680,00
910	111783,36	61398,91	49477,15	22813,00
920	113068,35	62160,10	49640,39	23076,00
930	115447,15	63444,72	50486,30	23436,00
940	117276,26	64019,02	52245,27	23569,00
950	117592,47	64857,42	51115,02	23832,00
960	120098,33	65353,90	52061,93	24192,00
970	120565,34	65675,90	52933,41	24325,00
980	121931,35	66492,80	52514,04	24588,00
990	124376,90	67639,72	53865,65	24948,00

1000	125453,80	68277,31	54347,85	25081,00
1010	127029,48	69013,48	54959,45	25344,00
1020	128842,93	70250,82	55336,25	25704,00
1030	129633,74	70261,40	56567,39	25837,00
1040	132423,33	71215,28	56817,90	26100,00
1050	134332,78	72935,62	57223,57	26460,00
1060	134037,58	72601,40	58208,96	26593,00
1070	136211,58	73313,50	58394,75	26856,00
1080	139156,78	75478,20	59172,48	27216,00
1090	140173,05	74913,23	60701,77	27349,00
1100	141257,80	76075,30	60301,09	27612,00
1110	142179,40	77331,14	60783,77	27972,00
1120	144429,35	77439,32	61811,87	28105,00
1130	146124,25	77935,40	62025,65	28368,00
1140	148068,10	79710,66	62404,82	28728,00
1150	149613,60	79938,98	63660,87	28861,00
1160	151494,67	79781,24	64074,64	29124,00
1170	152456,65	81162,80	63925,10	29484,00
1180	153718,65	81814,15	65730,90	29617,00
1190	154619,50	82473,28	65470,14	29880,00
1200	157440,90	83737,88	65690,53	30240,00
1210	159057,64	84219,84	67318,58	30373,00
1220	159939,95	84734,52	66922,92	30636,00
1230	162188,93	86084,76	67733,13	30996,00
1240	163584,62	86943,02	69906,04	31129,00
1250	165947,82	88092,40	69098,23	31392,00
1260	166060,62	88247,00	69724,85	31752,00
1270	167479,06	88410,53	70392,50	31885,00
1280	169333,67	88854,08	70605,61	32148,00
1290	171961,65	91466,04	71073,77	32508,00
1300	172939,78	91121,09	72457,21	32641,00
1310	174008,80	91431,96	72746,76	32904,00
1320	177321,05	93350,70	73408,10	33264,00
1330	177334,25	93223,25	73978,42	33397,00
1340	178598,12	93644,62	74329,30	33660,00
1350	181577,22	95574,66	74427,00	34020,00
1360	182558,03	95797,40	75745,11	34153,00
1370	184068,30	96370,98	76068,99	34416,00
1380	187024,50	98222,18	77236,52	34776,00
1390	188877,03	98010,55	77288,04	34909,00
1400	190210,53	98898,80	77689,34	35172,00
1410	191397,62	100284,50	78221,92	35532,00
1420	192000,74	100395,64	79136,42	35665,00
1430	195135,67	101229,28	79728,08	35928,00
1440	194656,45	101418,50	80327,72	36288,00
1450	197152,40	102245,71	81198,11	36421,00
1460	198677,62	103124,82	81838,64	36684,00
1470	200689,10	103709,04	81597,33	37044,00
1480	201740,35	104875,13	83093,38	37177,00
1490	203041,10	105400,76	83598,63	37440,00
1500	206277,85	106968,84	83749,83	37800,00

1510	205847,99	106518,32	84788,59	37933,00
1520	208359,25	107564,88	84318,71	38196,00
1530	211411,95	108496,70	85662,73	38556,00
1540	211502,65	108919,77	87220,75	38689,00
1550	213588,08	110093,54	86434,12	38952,00
1560	216358,40	111838,36	87974,15	39312,00
1570	215449,75	111388,72	88611,26	39445,00
1580	219114,82	112752,08	88637,61	39708,00
1590	220858,92	114000,86	89508,33	40068,00
1600	222697,11	113307,43	90317,44	40201,00
1610	223422,10	114249,20	89968,60	40464,00
1620	225919,45	115478,42	91219,22	40824,00
1630	227105,51	115013,29	91983,80	40957,00
1640	228856,70	117093,76	91407,40	41220,00
1650	229974,87	118241,68	91936,63	41580,00
1660	232405,80	118413,69	93250,04	41713,00
1670	234426,43	119558,00	93528,67	41976,00
1680	236628,05	120444,48	94461,22	42336,00
1690	237636,54	120266,04	95771,38	42469,00
1700	238009,50	120954,82	95669,48	42732,00
1710	240949,78	122397,76	95870,95	43092,00
1720	241062,93	122009,42	97413,14	43225,00
1730	244942,97	124231,68	97159,95	43488,00
1740	246021,95	125618,58	98059,65	43848,00
1750	246164,46	124265,85	99274,84	43981,00
1760	249500,20	126937,58	99005,77	44244,00
1770	250516,80	127657,64	99799,80	44604,00
1780	251890,31	127322,40	100514,80	44737,00
1790	254637,17	127726,88	100501,59	45000,00
1800	257693,62	129976,82	102126,45	45360,00
1810	260263,23	130357,49	103015,34	45493,00
1820	259764,52	131134,64	103194,00	45756,00
1830	260956,95	131740,90	103186,42	46116,00
1840	262644,75	132664,60	105602,71	46249,00
1850	264077,60	132434,52	105246,10	46512,00
1860	266716,35	134451,44	104865,12	46872,00
1870	268890,12	135294,59	107208,46	47005,00
1880	270899,05	135993,22	107005,24	47268,00
1890	273242,72	136198,98	107107,23	47628,00
1900	273452,26	136932,69	107835,35	47761,00
1910	276391,23	138206,98	107958,96	48024,00
1920	276418,00	138438,86	108587,65	48384,00
1930	278391,39	138215,19	110584,33	48517,00
1940	279715,47	140030,02	110405,92	48780,00
1950	282507,72	141066,48	111071,00	49140,00
1960	283047,71	141527,81	111416,30	49273,00
1970	283826,95	142227,54	112335,90	49536,00
1980	288850,43	143493,36	113175,43	49896,00
1990	287503,91	143410,22	113279,21	50029,00
2000	290246,40	144475,98	114211,15	50292,00