Estrutura de Dados I

Algoritmos de busca

Vinicios Barretos Batista

1º de Junho de 2018

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Introdução

O objetivo deste trabalho é comparar a eficiência de algoritmos de busca em diferentes estruturas de dados.

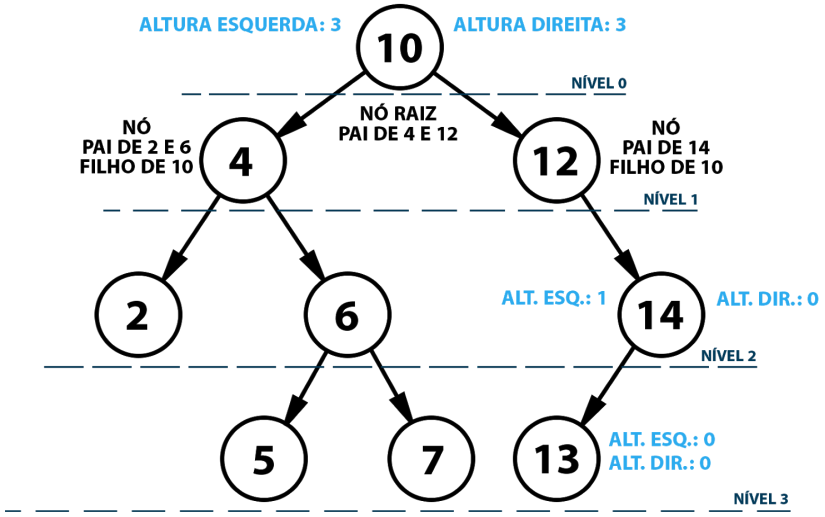
Para a realização do experimento foi utilizado 3 diferentes tipos de estruturas:

1) Árvore Binária convencional
2) Árvore Binária Balanceada com o método AVL
3) Vetor ordenado

1. Árvore Binária

Árvore Binária é uma estrutura de dados não linear na qual um elemento, chamado nó, é vinculado a no máximo dois elementos sucessores, referidos como nó filho esquerdo e nó filho direito. A cada elemento adicionado é verificado se este é maior ou menor do que o nó pai(anterior), caso seja maior o elemento é adicionado no ramo da direita, caso contrário, no ramo da esquerda.

Nó raiz é o nome do primeiro nó, o que dá origem a árvore. Nó pai é um nó que tem 1 ou 2 filhos. Nível de um nó é sua distância do nó raiz. A altura de um nó é o nível do nó mais distante subtraído do nível atual, considerando todo o ramo e podendo ser calculado tanto do lado esquerdo quanto do direito. Nó folha é um nó que não possui filhos.



1.1 Busca

Para buscar um elemento deve-se começar examinando a raiz. Se o valor for igual ao da raiz, este existe na árvore. Caso seja menor, deve-se iniciar a busca no ramo da esquerda, e assim recursivamente em todos os nós subsequentes.

Se o valor for maior do que a raiz, deve-se buscar no ramo da direita até alcançar o nó folha, encontrando ou não o valor desejado.

1.2 Inserção

Ao introduzir um nó novo na árvore, seu valor é primeiro comparado com o valor da raiz. Se seu valor for menor do que a raiz, este então é comparado com o valor do filho da esquerda da raiz. Se seu valor for maior, é comparado com o filho da direita da raiz. Este processo continua até que o novo nó seja comparado com um nó folha, e então adiciona-se o filho na direita caso o número seja maior ou na esquerda, caso seja menor.

1.3 Percurso

Existem 3 formas de se percorrer uma árvore binária, em ordem, pré ordem ou pós ordem.

Em ordem: percorre o ramo esquerdo, visita a raiz e percorre o ramo direito.
Pré ordem: visita a raiz, percorre o ramo esquerdo e percorre o ramo direito.
Pós ordem: percorre o ramo esquerdo, percorre o ramo direito e visita a raiz.

Uma característica interessante é que ao percorrer a árvore em ordem obtém-se os valores em ordem crescente.

2. Árvore Binária Balanceada com o método AVL

Este tipo de árvore binária é balanceada, ou seja, é uma árvore que segue todas as propriedades da árvore binária e em cada nó apresenta diferença de altura entre os ramos direito e esquerdo sendo -1, 0 ou 1 apenas.

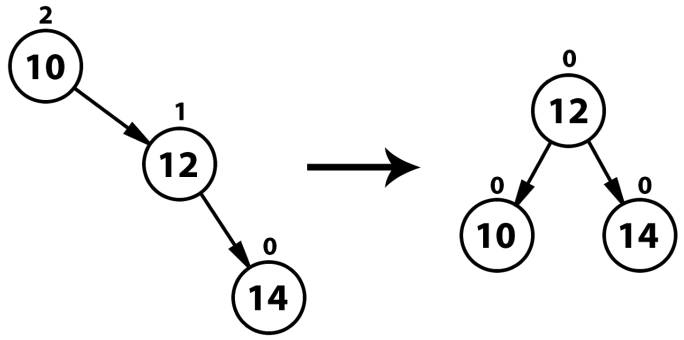
A diferença de altura, também chamada de peso de um nó, se dá subtraindo a altura do ramo direito com a altura do ramo esquerdo. Caso o peso seja maior que 1 ou menor que -1, a árvore está desbalanceada e será preciso realizar uma rotação. Para cada caso há uma rotação específica como segue na tabela 1.

Peso do nó desbalanceado	Peso do filho do nó desbalanceado	Tipo de rotação
2	1	Simples Esquerda
	0	Simples Esquerda
	-1	Dupla com filho para direita e pai para esquerda
-2	1	Dupla com filho para esquerda e pai para direita
	0	Simples Direita
	-1	Simples Direita

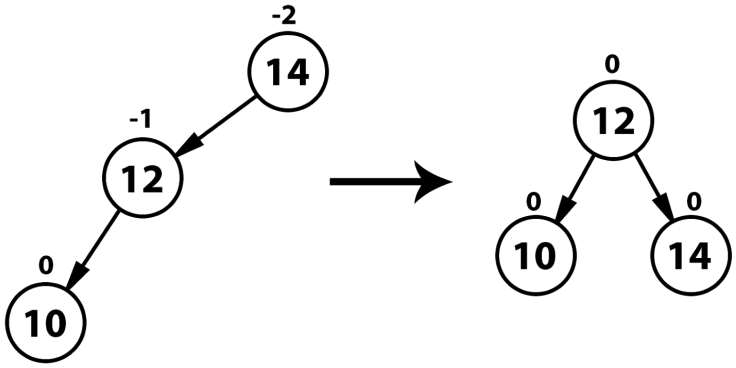
Tabela 1 - Tipos de rotação

2.1 Tipos de Rotação

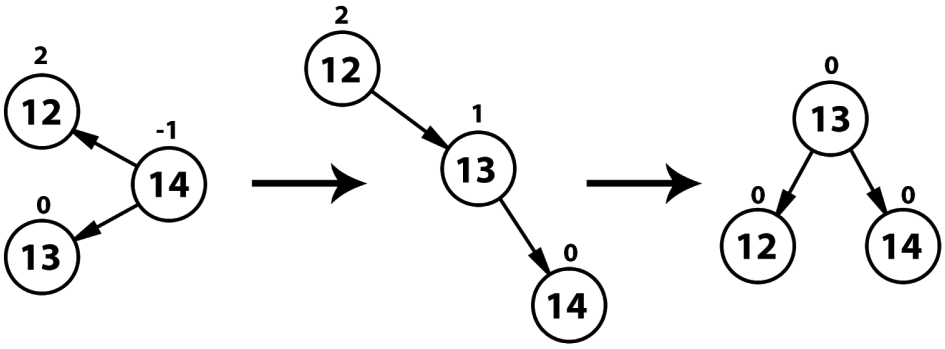
Simples Esquerda:



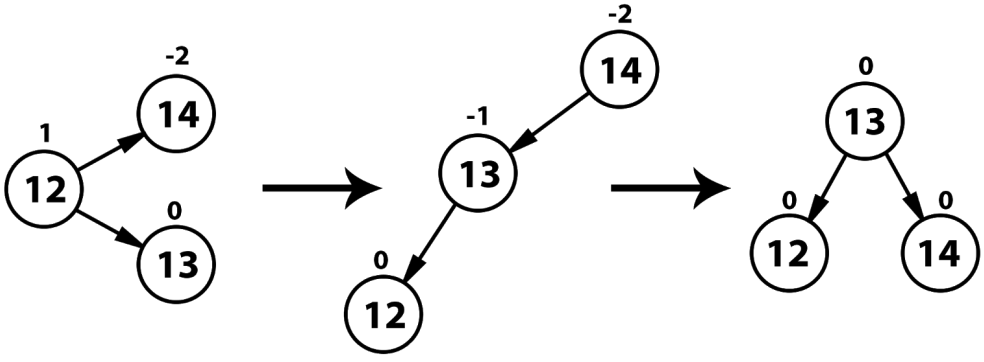
Simples Direita:



Dupla com filho para direita e pai para esquerda:



Dupla com filho para esquerda e pai para direita:



2.2 Inserção

O processo de inserção utiliza-se de uma função muito parecida com o da árvore binária. Algumas modificações são realizadas para que a cada inserção se atualize as alturas de cada nó e caso necessário, realize o balanceamento.

O balanceamento consiste em identificar o nó desbalanceado e fazer as rotações necessárias.

Cumprido estes passos a árvore estará balanceada e pronta para receber um novo número.

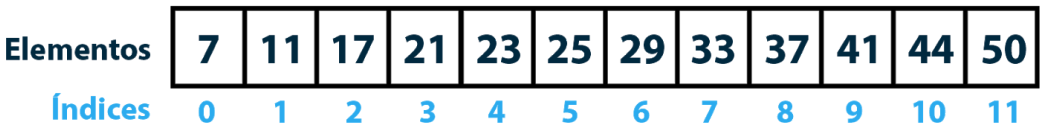
2.3 Busca

O processo de busca é exatamente o mesmo da árvore binária convencional.

3. Vetor ordenado

O vetor é uma estrutura de dados linear, ou seja, os números podem ser acessados através de índices numéricos. Este tipo de estrutura permite armazenar uma determinada quantidade de valores do mesmo tipo.

Para se obter um vetor ordenado pode-se utilizar algum algoritmo de ordenação ou então já realizar a inserção dos elementos de forma ordenada. Neste trabalho será utilizado o algoritmo de ordenação Quicksort por apresentar um bom desempenho, conforme será demonstrado a seguir.



3.1 Inserção

A inserção acontece de forma não ordenada, e depois o vetor é ordenado pelo algoritmo Quicksort. Este método funciona da seguinte maneira:

1. É selecionado um elemento pivô, que neste caso é o que ocupa a posição central do vetor
2. Particiona o vetor em 2 partes, a esquerda do pivô e a sua direita
3. No lado esquerdo procura-se um elemento maior do que o pivô, enquanto no lado direito um elemento menor
4. Caso encontre e os elementos estejam dentro de suas partições, estes são trocados de posição
5. Esse processo é repetido sucessivamente até que toda a parte esquerda ou direita tenha sido percorrida
6. De modo recursivo, os passos anteriores são repetidos até que se obtenha o vetor ordenado

3.2 Busca

Basicamente, é possível realizar a busca em um vetor ordenado de duas formas distintas, sendo elas a Busca Sequencial e a Busca Binária.

A busca sequencial consiste em percorrer o vetor até que o elemento seja encontrado, ou até que se atinja um elemento maior do que o desejado, indicando assim que não há possibilidade do elemento existir a partir deste ponto.

Já a busca binária, por outro lado, se baseia no método de divisões sucessivas do vetor, até que o valor desejado seja encontrado. O primeiro passo é verificar se elemento a ser buscado é igual, menor ou maior que o elemento do meio do vetor. Se este for igual, então o elemento foi encontrado. Caso o valor seja menor, a busca é realizada somente na metade “menor” do vetor, ou seja, o lado esquerdo, enquanto o lado maior é descartado. Caso o valor a ser buscado seja maior que o valor do elemento do meio, a busca é realizada somente na metade “maior” do vetor, ou seja, a partir do elemento do meio, enquanto o lado menor é descartado.

Experimentos e Discussão

A realização do experimento consiste na obtenção dos seguintes dados:

1. Tempo de criação da estrutura
2. Quantidade de iterações para criação da estrutura
3. Tempo para realizar a busca
4. Quantidade de iterações para realizar a busca

As funções de inserção, rotação, balanceamento e busca foram modificadas para que fossem capazes de contar a quantidade de iterações a cada passo recursivo ou por meio de laços de repetição.

Para se obter os dados 1 e 2 foi utilizado a seguinte estratégia:

1. Definir a quantidade de elementos das estruturas
2. Definir o alcance dos números a serem gerados
3. Inicializar as estruturas
    a. Gera um número aleatório até o alcance definido
    b. Verifica se este número já foi adicionado à estrutura
    c. Caso positivo, é gerado um novo número até que este seja único e possa ser adicionado
    d. Começa a contagem de tempo e de iterações
    e. Adiciona o elemento à estrutura
    f. Encerra a contagem de tempo e de iterações
    g. Guarda estes dados da contagem e iterações em um vetor para serem somados com os da próxima inserção
    h. Os passos d, e, f, g se repetem para cada estrutura
    i. Todos os passos anteriores são realizados sucessivamente até atingir a quantidade de elementos definida
    j. Começar a contagem de tempo e de iterações
    k. O vetor é ordenado utilizando o algoritmo Quicksort
    l. Encerra a contagem de tempo e de iterações
    m. Soma estes dados com os de inserção do vetor
4. Obtenção do tempo total e a quantidade total de iterações para a criação de cada estrutura

É importante ressaltar que o tempo gasto para gerar um número único não é contabilizado, uma vez que isto trata-se de uma característica do experimento e não das estruturas em si.

A ideia de usar um alcance definido nos números gerados é para poder escolher, em termos probabilísticos, a chance de ter um número encontrado durante a busca. Isso deve-se, pois o número a ser buscado também é um número aleatório gerado com alcance definido.

Deste modo, ao gerar um conjunto de dados de 300.000 números e definir o alcance em 600.000 a probabilidade do número gerado ser encontrado é de 50%.

Contudo, o alcance nunca deve ser menor do que a quantidade de elementos, pois dessa forma não seria possível gerar as estruturas com a devida quantidade. Além disso, vale ressaltar que quanto mais próximo o alcance da quantidade de elementos, maior poderá ser o tempo gasto gerando os dados, uma vez que a probabilidade de se adicionar um número que já existe nas estruturas é ainda maior, e este teria de gerar um novo número até que fosse único.

Para verificar se o número já foi adicionado, foi utilizado a busca em uma das estruturas, sendo a árvore AVL a escolhida, uma vez que neste ponto o vetor ainda não foi ordenado e além disso, este tipo de árvore tem um excelente desempenho para busca conforme será descrito posteriormente.

Já, para obter os dados 3 e 4 foi utilizado a seguinte estratégia:

1. Obtenção do número a ser pesquisado
2. Começa a contagem de tempo e de iterações
3. Procura o elemento na estrutura
4. Encerra a contagem de tempo e de iterações
5. Verifica se o elemento buscado foi encontrado
6. Guarda estes dados de contagem e busca em um vetor
7. O processo se repete para cada método de busca

Os experimentos foram realizados em um ambiente UNIX, utilizando linguagem C++ e compilador g++.

Para exibir os dados foi utilizado a biblioteca <iomanip> e seus recursos de impressão em forma de tabela, enquanto para gerar os números aleatórios foi utilizado a biblioteca <time.h> e a função srand.

Os experimentos realizados seguem todos o padrões descritos anteriormente e apresentaram os seguintes resultados:

Números: 100.000 | Alcance: 200.000					
Criação			Busca		
	Tempo	Iterações	Tempo	Iterações	Encontrado
Árvore AVL	0,145067s	1.764.677	0,000001s	17	Sim
Árvore Convencional	0,143580s	2.088.437	0,000002s	26	Sim
Vetor Ordenado Busca Sequencial	0,096504s	1.958.942	0,000103s	41.369	Sim
Vetor Ordenado Busca Binária	0,096504s	1.958.942	0,000001s	16	Sim

Tabela 2 - Inserindo 100.000 números

Números: 500.000 | Alcance: 1.000.000					
Criação			Busca		
	Tempo	Iterações	Tempo	Iterações	Encontrado
Árvore AVL	0,751823s	10.016.651	0,000004s	22	Não
Árvore Convencional	0,808160s	11.984.097	0,000002s	32	Não
Vetor Ordenado Busca Sequencial	0,487136s	10.772.758	0,000494s	199.567	Não
Vetor Ordenado Busca Binária	0,487136s	10.772.758	0,000001s	19	Não

Tabela 3 - Inserindo 500.000 números

Números: 2.000.000 | Alcance: 2.000.000					
Criação			Busca		
	Tempo	Iterações	Tempo	Iterações	Encontrado
Árvore AVL	3,232825s	44.117.119	0,000005s	21	Sim
Árvore Convencional	3,658556s	52.466.244	0,000001s	23	Sim
Vetor Ordenado Busca Sequencial	2,052958s	48.357.765	0,000566s	229.135	Sim
Vetor Ordenado Busca Binária	2,052958s	48.357.765	0,000003s	19	Sim

Tabela 4 - Inserindo 2.000.000 números

Números: 4.000.000 | Alcance: 8.000.000					
Criação			Busca		
	Tempo	Iterações	Tempo	Iterações	Encontrado
Árvore AVL	6,675805s	92.185.623	0,000004s	21	Sim
Árvore Convencional	8,450834s	119.117.747	0,000003s	23	Sim
Vetor Ordenado Busca Sequencial	4,080466s	100.496.859	0,010460s	3.979.501	Sim
Vetor Ordenado Busca Binária	4,080466s	100.496.859	0,000001s	20	Sim

Tabela 5 - Inserindo 4.000.000 números

A partir desses dados é possível afirmar que a busca na árvore balanceada pelo método AVL e a busca binária em um vetor ordenado foram os métodos nos quais tiveram melhor desempenho, especialmente levando em conta o número de iterações da criação e busca. Este número é muito importante pois cada iteração representa um ciclo de máquina dentro do processador, e mesmo que mínimo, cada ciclo leva um certo tempo para ser completado.

Já a busca sequencial em um vetor ordenado foi a que obteve pior resultado em todos os testes, tanto no quesito tempo quanto iterações.

Ao se comparar a árvore binária convencional com a árvore binária balanceada a diferença, embora exista, não é tão grande para os dados apresentados. Contudo, essa diferença, principalmente no quesito iterações poderia ser muito maior caso a entrada fosse diferente. Este fenômeno será explicado com maior detalhes na próxima sessão.

Complexidade de Algoritmos

Para se medir a complexidade de algoritmos é utilizado a notação Big-O (O-grande). Este termo caracteriza funções de acordo com suas taxas de crescimento.

Quando informado a complexidade de um algoritmo é importante que seja mostrado o pior caso e o caso médio, para que se analise em qual situação o problema a ser resolvido se enquadraria.

Com base no livro Algoritmos: Teoria e Prática de Charles Eric Leiserson, Ronald Rivest e Thomas H. Cormen, é possível descrever as estruturas aqui apresentadas e sua complexidade da seguinte forma:

Busca				
	Árvore AVL	Árvore Binária	Busca Sequencial	Busca Binária
Caso Médio	O(log n)	O(log n)	O(n)	O(log n)
Pior Caso	O(log n)	O(n)	O(n)	O(log n)

Tabela 6 - Complexidade dos algoritmos de busca

*Para os dados da tabela 6 considerar log na base 2.

Com os resultados exibidos nas tabelas 2, 3, 4 e 5 foi possível observar que a diferença entre as iterações de busca para a árvore AVL, árvore binária convencional e a busca binária em um vetor ordenado permaneceu sempre próximas uma das outras. Isto ocorreu pois a amostra de dados utilizada não atuou de forma tão negativa às estruturas.

Contudo, em uma situação hipotética onde se deseja buscar um número em uma amostra com 8.000.000 elementos e tendo como análise o pior caso, obteríamos os seguintes dados:

1. **Árvore AVL:** 23 iterações
2. **Árvore Convencional:** 8.000.000 iterações
3. **Vetor Ordenado Busca Sequencial:** 8.000.000 iterações
4. **Vetor Ordenado Busca Binária:** 23 iterações

A ideia de utilizar números aleatórios é para que se tenha uma amostra de caso médio, entretanto, é possível gerar estruturas como sendo o pior caso. Para isso, por exemplo, basta realizar a inserção do número 0 a n, onde n é a quantidade desejada e procurar por n.

A fim de ainda realizar um último experimento, foi obtido os seguintes dados com uma amostra de 12.000.000 elementos:

Números: 12.000.000 | Alcance: 24.000.000					
	Criação		Busca		
	Tempo	Iterações	Tempo	Iterações	Encontrado
Árvore AVL	20,362730s	295.847.600	0,000005s	21	Sim
Árvore Convencional	26,198605s	372.691.878	0,000003s	30	Sim
Vetor Ordenado Busca Sequencial	12,388724s	323.242.375	0,012477s	5.053.377	Sim
Vetor Ordenado Busca Binária	12,388724s	323.242.375	0,000003s	24	Sim

Tabela 7 - Inserindo 12.000.000 números

É possível analisar que, conforme a amostra de dados cresce, a diferença de tempo para criação das estruturas torna-se ainda mais relevante. Sendo um fator decisivo na escolha da estrutura.

Conclusão

A partir dos dados anteriores é possível afirmar que a busca binária em um vetor ordenado é a melhor opção, pois esta foi a que apresentou menor tempo de criação em todos os casos aqui apresentados, e também um desempenho de busca melhor do que as outras estruturas em quase todos os testes.

A complexidade de busca da busca binária é a mesma que da árvore AVL, portanto, compreende-se que o fator decisivo para sua escolha é o tempo de criação da estrutura.

Contudo, a necessidade de se ordenar o vetor faz com que, dependendo da situação, seja mais vantajoso o uso da árvore AVL. Portanto, essa escolha deve ser tomada com base no projeto e suas prioridades.

Referências

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. MIT Press, 2009. p. 333.

[2] https://en.wikipedia.org/wiki/Binary_search_algorithm
Última visualização: 1º de Junho de 2018

[3] https://en.wikipedia.org/wiki/Big_O_notation
Última visualização: 1º de Junho de 2018

[4] http://bigocheatsheet.com/
Última visualização: 1º de Junho de 2018