Universidade Federal de Alagoas - UFAL
Departamento de Tecnologia da Informação - TCI
Ciência da Computação

Árvores Binárias e AVL

Felipe Barros Pontes
Gustavo Márcio de Morais Cunha
Márcio de Medeiros Ribeiro

Trabalho requisitado pelo Professor
Cid Cavalcanti afim de obtenção de nota na
disciplina de Banco de Dados 1

Maceió - Alagoas

Introdução

Árvores são Estruturas de Dados que combinam as vantagens de duas outras estruturas: Uma Lista Encadeada e um Array Ordenado. A Árvore pode fazer inserções e deleções tão rápido quanto uma Lista Encadeada, e também pode fazer buscas tão rápidas quanto um Array Ordenado.

Inserção e Deleção em Arrays Ordenados → A inserção e deleção em Arrays Ordenados é lenta porque é necessário que todos os elementos que são maiores que o elemento a ser inserido sejam movidos adiante, até que exista um espaço para ser inserido o elemento em sua posição correta. De modo análogo para a operação de deleção, onde todos os elementos maiores que o elemento eliminado tem que ser movidos para trás em uma posição.

Buscas em Listas Encadeadas → As buscas em Listas Encadeadas tem que ser feitas do inicio da lista, e então, visitar todos os elementos a procura do elemento desejado. Esse processo é lento, pois é necessário, em média, de N/2 comparações (considerando **N** o número de elementos da lista).

Definições

Nó → Entes da árvore que contém além dos dados que se quer armazenar ponteiros pra outros entes;

Raiz → É o primeiro Nó da Árvore, que se encontra sempre no topo da mesma;

Folha → É o Nó que não possui filhos;

Pai → É o Nó imediatamente superior. Todos os elementos, exceto a Raiz, possuem um Pai;

Filho → São os Nós imediatamente inferiores;

Sub-árvore → Qualquer Nó pode ser Raiz. Uma Subárvore é qualquer árvore cuja Raiz é um Nó de uma árvore maior ou igual a ela;

Níveis → É a referência a quantidade de gerações do nó comparados a Raiz da árvore;

Árvore Binária → São árvores que possuem apenas dois filhos, um a esquerda e outro a direita.

Algoritmos

Inserção → O Algoritmo de Inserção em Árvores Binárias consiste em, dado um elemento, encaixá-lo na árvore obedecendo a seguinte regra: a esquerda de cada nó ficam os elementos de valor menor que o valor do nó, e a direita de cada nó ficam os elementos de valor maior que o valor do nó**.**

Busca → O Algoritmo de Busca em Árvores Binárias consiste em, dado um elemento, procurar se ele existe na árvore e retornar um ponteiro para o mesmo. Caso o elemento não exista, o algoritmo retorna *NULL*.

Remoção → O Algoritmo de Remoção é um algoritmo extenso e complexo. Para remover um Nó que não possui filhos é simples: apenas elimine-o da árvore.

Se o Nó possui um filho, remove-se o Nó e insere a subárvore no Nó pai do Nó eliminado.

Se o Nó possui dois filhos, é necessário encontrar o "inorder sucessor" do Nó a ser eliminado. Ou seja, é necessário encontrar o menor elemento da subárvore a direita do nó a ser deletado. Esse Nó será o substituto do Nó eliminado.

Travessias

"InOrder" → O percurso "InOrder" fará com que todos os nós sejam visitados na ordem ascendente, baseada em seus valores chaves.

"PreOrder" → A motivação da travessia "PreOrder" e "PostOrder" ainda é muito obscura, mas sabe-se que é útil para programas que analisam expressões algébricas. A travessia "PreOrder" origina uma expressão prefixa.

"PostOrder" → A travessia "PostOrder" origina uma expressão pósfixa, que também é utilizada para o cálculo de expressões algébricas.

Árvores Balanceadas e AVL

A inserção aleatória de nós em uma árvore binária pode degenerá-la, tornando-a bastante semelhante a outras estruturas de dados, tais como: array ordenados e listas encadeadas. Assim, o uso de uma árvore binária torna-se discutível.

Embora existam ordens de inserção de nós que conservam o balanceamento de uma árvore, na prática isso é impossível de se prever ou até alterar. Para esse propósito existem algoritmos que se propõem a balancear uma árvore.

A vantagem de uma árvore balanceada sobre uma degenerada está na maior eficiência nas suas operações de busca, pois, sendo a altura da balanceada bem menor, o número necessário de comparações diminui sensivelmente. Por exemplo, numa árvore degenerada de 10.000 nós, são necessárias, em média, 5.000 comparações, numa busca; numa árvore balanceada, com o mesmo número de nós, essa média baixa para 14.

Uma árvore binária de busca é considerada balanceada quando, para cada nó, as alturas de suas sub-árvores esquerda e direita diferem de, no máximo, 1. Essa diferença é chamada *fator de balanceamento*.

Fator de balanceamento de um nó é a diferença entre a altura da sub-árvore esquerda em relação à sub-árvore direita.

FB(p) = altura(sub-árvore esquerda p) - altura(sub-árvore direita p)

Em uma árvore binária balanceada todos os Fatores de Balanceamento de todos os nós assumem os valores -1, 0 e 1. Para quaisquer valores fora dessa faixa teremos uma árvore binária degenerada.

Seja um nó qualquer da árvore, apontado por **n**:

se FatBal(n) = 0, as duas sub-árvores têm a mesma altura;
se FatBal(n) = -1, a sub-árvore esquerda é mais alta que a direita em 1;
se FatBal(n) = +1, a sub-árvore direita é mais alta que a esquerda em 1.

AVL

As árvores AVL, cujo nome foi obtido a partir das iniciais de seus criadores russos – Adel'son-Vel'skii e Landis, são uma implementação de um algoritmo para a criação de árvores binárias balanceadas.

Uma vez que foi detectado o desbalanceamento da árvore após uma inserção, e que foi determinado o nodo pivô (onde o valor absoluto de FB é maior que 1), passamos para a realização do procedimento de balanceamento da árvore. Este procedimento consiste primeiramente de determinar em qual dos seguintes tipos de casos nos enquadramos:

Tipo I : Se a subárvore esquerda é maior que a subárvore direita (FB > 1)
E a subárvore esquerda desta subárvore esquerda é maior que a subárvore direita dela
Então realizar uma **rotação simples para a direita**;

Tipo II : Se a subárvore esquerda é maior que a subárvore direita (FB > 1)
E a subárvore esquerda desta subárvore esq. é menor ou igual que a subárvore direita
Então realizar uma **rotação dupla para a direita**;

Tipo III: Se a subárvore esquerda é menor que a subárvore direita (FB < -1)

E a subárvore direita desta subárvore direita é maior que a subárvore esquerda dela
Então realizar uma **rotação simples para a esquerda**;

Tipo IV: Se a subárvore esquerda é menor que a subárvore direita (FB < -1)
E a subárvore direita desta subárvore direita é menor que a subárvore esquerda dela
Então realizar uma **rotação dupla para a esquerda**;

Aplicações das Árvores Binárias

As árvores binárias são uma das estruturas de dados mais importantes que se têm conhecimento no mundo da computação. Suas aplicações são muitas e, o mais interessante, é que elas são aplicáveis a diversas áreas do conhecimento computacional.

Neste trabalho são mostradas duas aplicações em áreas totalmente distintas. A semelhança entre elas é que, sem dúvida alguma, as árvores binárias tiveram participação extremamente importante à resolução dos problemas abaixo citados.

A primeira aplicação refere-se a comunicação de pacotes em uma rede. Tais pacotes, além de poderem ser enviados em redundância, podem não ser ordenados para formar a mensagem completa e consistente.

A segunda aplicação diz respeito ao contexto de compressão e descompressão de arquivos utilizando o algoritmo de David Huffman. Em tal algoritmo, a presença das árvores binárias é fundamental na desenvoltura do mesmo.

1) Redes de comunicação de dados

A maioria dos protocolos de comunicação fragmenta as mensagens em pacotes que são numerados e enviados através da rede. Não há nenhuma garantia de que os pacotes chegarão ao seu destino na ordem em que foram enviados pelo remetente. Além disso, pode haver (por causa da perda de mensagens de confirmação) a duplicação de alguns pacotes que chegam ao destinatário. Afim de reconstituir a mensagem original, o destinatário precisa descartar as réplicas de pacotes recebidos e ordená-los. O uso de árvores binárias possibilita a implementação de um algoritmo eficiente para resolver ambos os problemas.

Como funciona o algoritmo para ordenar e evitar redundâncias?

O primeiro número da lista é colocado num nó que é a raiz da árvore, com as sub-árvores esquerda e direita vazias. Cada número sucessivo é comparado com o número da raiz. Se for igual, achou uma réplica (descarta-se o número). Se é menor, examinamos a sub-árvore esquerda; se é maior, examinamos a sub-árvore direita. Se a sub-árvore é vazia, o número não está duplicado e é colocado em um novo nó naquela posição da árvore. Se a sub-árvore não for vazia, comparamos o número com o conteúdo da raiz da sub-árvore e o processo é então repetido para a sub-árvore. Uma vez construída a árvore, basta percorrê-la em ordem simétrica (veja seção seguinte) para obter os seus elementos em ordem crescente.

2) Codificação de Huffman

O algoritmo de compressão de Huffman (criado por **David Huffman** em 1952) é um algoritmo utilizado para compressão e descompressão de arquivos. Importantes programas como **Winzip** e **WinRAR** utilizam este algoritmo. O funcionamento básico do mesmo se dá pela associação de pequenos códigos a freqüentes caracteres e códigos maiores a caracteres não muito freqüentes.

Entende-se por código como uma seqüência de 0 e 1 que pode representar de maneira única o caracter, ou seja, que não pode ser repetido.

Um arquivo é uma coleção de caracteres. Como dito anteriormente, há caracteres muito freqüentes e pouco freqüentes. O número de bits requeridos para representar cada caracter depende do número de caracteres que têm de ser representados.

Usando um bit pode-se representar dois caracteres ao passo que usando dois bits pode-se representar quatro caracteres. Generalizando, tem-se que o número de caracteres é igual a dois elevado ao número de bits.

Uma desvantagem do código ASCII no contexto de representação de caracteres por bits, é que o mesmo utiliza uma codificação com número fixo de caracteres. Isso pode gerar muitos bits que podem ser desprezíveis na hora do armazenamento no arquivo. Existe, nesse contexto, uma outra abordagem: a codificação com número variável de bits. Como o próprio nome diz, os caracteres não precisam ser representados com um número fixo de bits e, desta maneira, pode-se diminuir significativamente o número de bits na representação de alguns caracteres. Esta é a abordagem utilizada pelo algoritmo de Huffman.

Como dito anteriormente, o algoritmo de Huffman está intimamente ligado a uma árvore binária. A seguir, tem-se como se dá a construção da mesma.

Construção da Árvore Binária

Primeiramente, um Array é considerado afim de armazenar a freqüência com que os caracteres aparecem no texto do arquivo. Ao percorrer todo o arquivo, estas informações são armazenadas no Array supracitado e é a partir daí que a construção da árvore binária ocorre de fato.

O próximo passo é verificar as duas menores freqüências dentro do Array. Elas serão consideradas folhas da árvore (bem como o próprio caracter que possui tal freqüência) e a soma de suas freqüências será considerada seus pais. Essa soma é armazenada em posições finais do Array em questão. É interessante salientar que o algoritmo passa a desprezar as folhas criadas, o que é um fator positivo no ponto de vista da eficiência do mesmo.

O passo acima é repetido considerando os outros elementos do Array bem como a soma de freqüências criada anteriormente.

Compressão do Arquivo

Um fator importantíssimo dentro desta árvore binária é que filhos (de qualquer sub-árvore) da esquerda possuem o bit 0 ao passo que os da direita possuem o bit 1 como informações. Estes bits são de fundamental importância para a criação da referência que haverá entre cadeia de bits e os caracteres propriamente ditos.

Como cada caracter está contido em uma folha, para descobrir o código em bits que foi gerado para o mesmo basta percorrer a árvore até tal folha. A cadeia de bits de acordo com os lados esquerdo e direito determina que ela será referência para o caracter encontrado na folha, após o percurso até a mesma.

A compressão começa quando o arquivo é percorrido novamente e os caracteres são substituídos pelos bits correspondentes que já se encontram na árvore binária.

Após o percurso do arquivo, uma tabela é criada. Ela possui cadeias de bits e seus caracteres correspondentes. Esta tabela é muito importante para a descompressão do arquivo pois será através da mesma que haverá o mapeamento entre bits e caracter.

Descompressão

Através dos bits os quais se encontram na tabela de bits, a árvore binária é refeita. A cadeia de bits é percorrida e, à medida que uma sub-cadeia é encontrada na tabela de caracteres e códigos, a mesma é substituída pelo caracter correspondente, dando ao arquivo seus caracteres originais.

Anexo

Figura 1. Rotação simples para a direita.

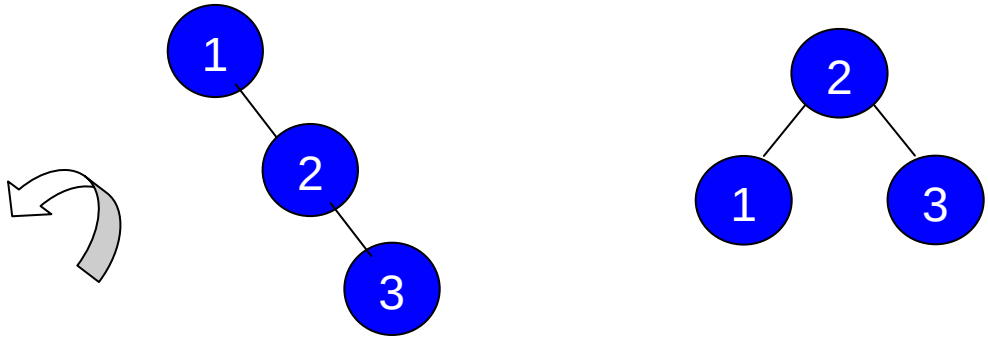

Figura 2. Rotação simples para a esquerda.

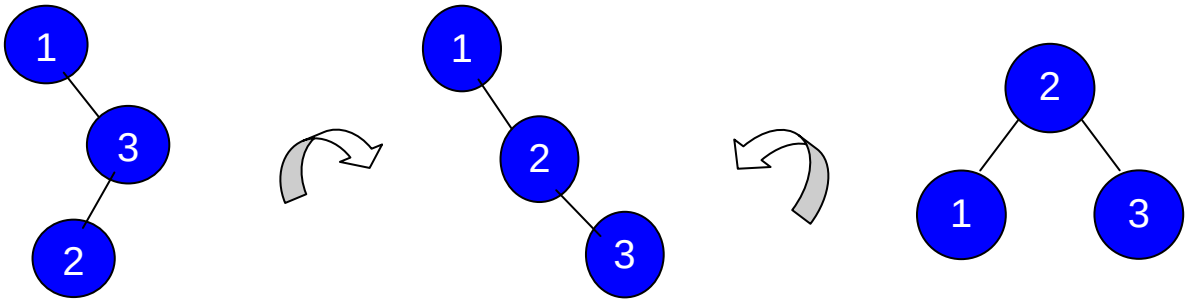

Figura 3. Rotação dupla para a direita.

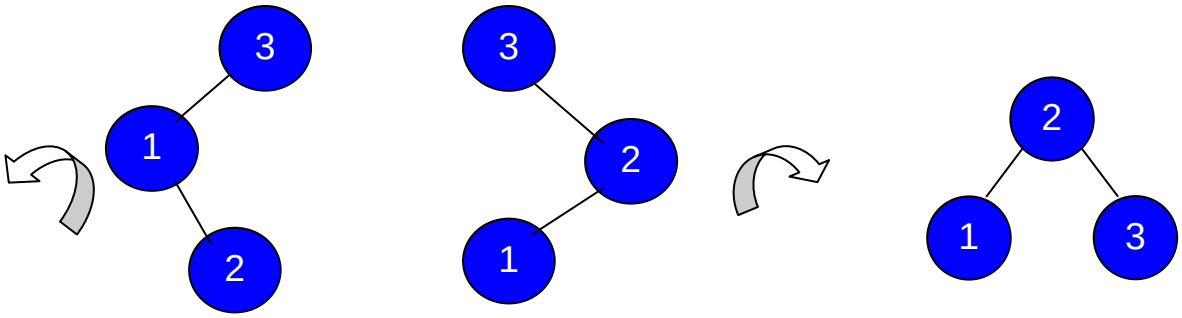

Figura 4. Rotação dupla para a esquerda.

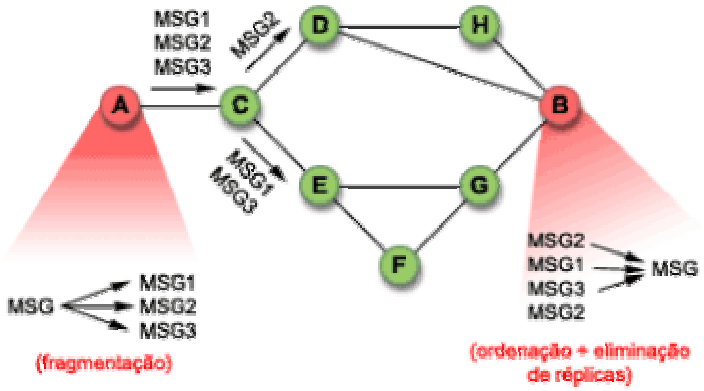

Figura 5. Rede de comunicação de dados que utiliza árvore binária para ordenar os pacotes de rede bem como evitar redundâncias decorrentes do reenvios de pacotes.

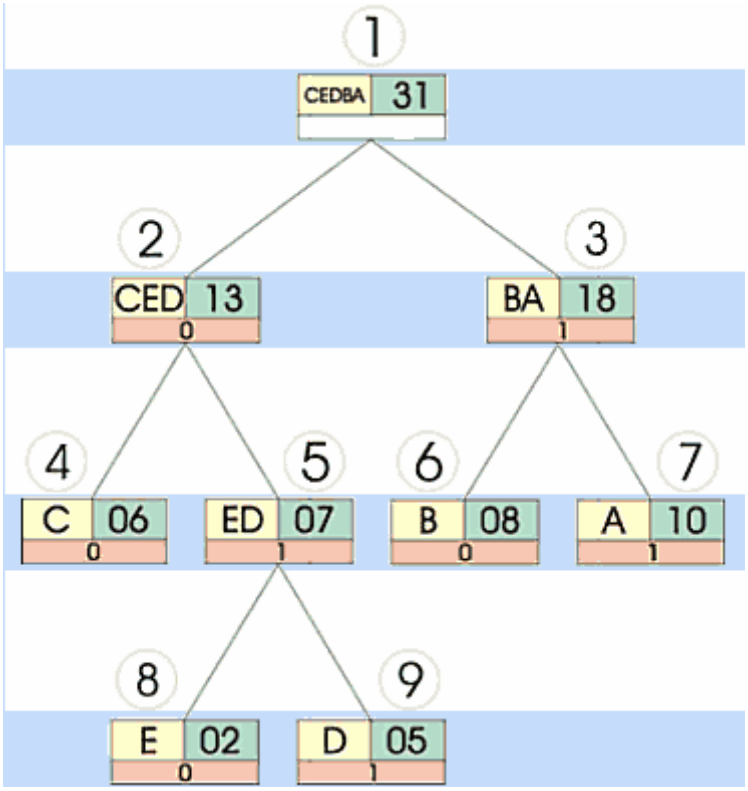

Figura 6. Árvore binária gerada pelo algoritmo de Huffman.

Bibliografia

- **http://w3.ualg.pt/~hshah/ped/**
- **http://www.geocities.com/grupotrees/AVL/avl.htm**
- **http://inf.unisinos.br/~anibal/prog2avl.rtf**
- **http://www.inf.unisinos.br/~osorio/lab2/lab2-a09b.pdf**
- **http://www.howtodothings.com/showarticle.asp?article=313**
- **http://www.ppgia.pucpr.br/~laplima/aulas/materia/arvbin_m.html**