Estruturas de Dados em Árvores: Relatório com uma análise com os resultados obtidos de um algoritmo implementado com as árvores: A23 e ARN.

Dayan Ramos Gomes

Resumo:

Este relatório apresenta informações sobre as estruturas de dados em árvores utilizadas para resolução de um problema definido, sendo elas: Árvore 2-3 (A23) e a Árvore Red-Black ou Rubro-Negra (ARN), assim, foram desenvolvidos algoritmos para a resolução do problema com essas duas árvores. Com base nos resultados obtidos, uma análise comparativa para o tempo da operação de busca foi feita, onde mais detalhes foram mostrados a fim de determinar qual estrutura de dados em árvore é mais eficiente.

1. Introdução

Árvore é uma estrutura de dados que herda as características das topologias em árvore, conceitualmente diferente das listas encadeadas, onde, nas árvores os dados estão dispostos de forma hierárquica e o seu conceito e implementação está diretamente ligado à recursividade.

Este trabalho tem como objetivos demonstrar dois tipos de estruturas de dados em árvores mais complexas do que as já vistas e seu funcionamento e desempenho, e assim, podermos entender melhor as árvores e suas características. Fazendo uma análise parcial do problema e tentando identificar a quantidade de algumas coisas como: informações que vão ser armazenadas inicialmente, se os dados precisam estar bem distribuídos ou ordenados, entre outras, podemos definir um tipo de árvore a se utilizar. Sabendo disso, podemos prover uma melhor otimização tanto do hardware e software da nossa máquina e prover uma melhor resolução maximizando as possibilidades de problemas.

Assim, os testes realizados dos algoritmos foram feitos em um notebook da Acer, com as seguintes especificações técnicas:

- Modelo: Acer a515-51g-58vh
- Processador: Intel Core I5-7200U de 7ª G – 2 núcleos e 4 threads - (até 3.1 GHz) 3MB Cache
- Memória RAM: 7.9 GB DDR4 - 2133 MHz (2 x 4 GB)
- Placa de vídeo dedicada GeForce 940MX com 2GB GDDR5
- SSD SATA M.2 WD Green 240 GB – 545 MB/s leitura e escrita
- Sistema Operacional: Windows 10 de 64 bits

Desta forma, este documento está organizado da seguinte forma: A Seção 2 contém uma breve explicação sobre os conceitos dos tipos de árvores utilizados, o problema proposto e uma apresentação sobre os aspectos funcionais presentes nos programas desenvolvidos para resolver o problema; a Seção 3 apresenta um estudo explicativo e comparativo feito a partir da execução dos programas e uma explanação sobre o que são e para que servem; a Seção 4 apresenta a conclusão do trabalho.

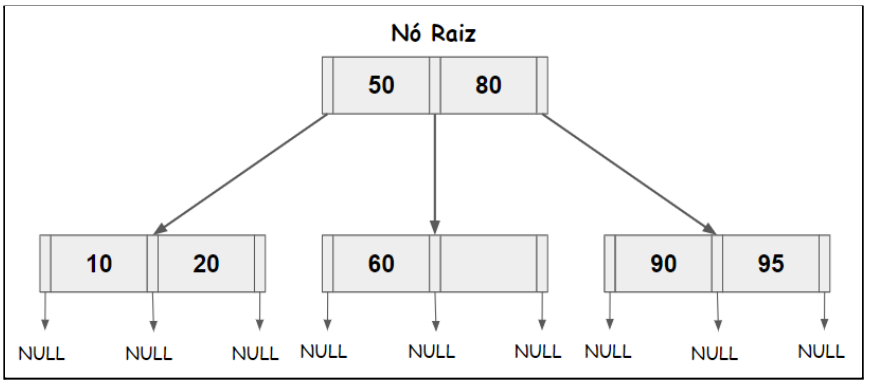
2. Seções Específicas

Para um melhor entendimento do relatório é importante conhecer as estruturas de dados em árvores utilizadas. Abaixo está uma breve explicação das principais características de cada árvore e logo depois o problema proposto com suas resoluções com as duas árvores.

2.1 Árvore 2-2 (A23)

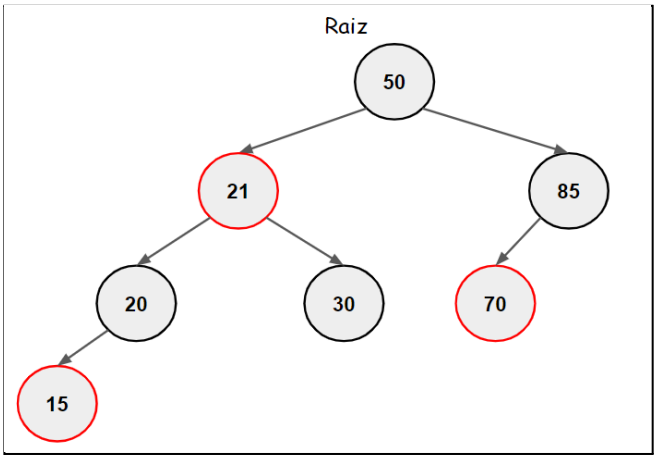
A árvore 2-3 é uma estrutura de árvore que pode armazenar até 2 informações em um único nó e que pode ter 2 ou 3 filhos, onde esses filhos são ponteiros para ou nó igual.

Sempre que houver uma informação no nó e esse nó não for um nó folha, haverá 2 filhos, um à esquerda e outro no centro. Já se houver 2 informações no nó e esse nó não for folha, ele deverá ter 3 filhos, um à esquerda, um no centro e outro à direita. Após encontrar o local de inserção deverá ser feito o balanceamento do mesmo de acordo com as informações. A principal função a ser desenvolvida é a de quebrar o nó, que é utilizado quando é inserido uma informação em um nó já cheio. Sempre promovendo um novo nó. A estrutura da árvore é apresentada abaixo.



2.2 Árvore Rubro-Negra caída para esquerda (ARN_CE)

A árvore rubro negra consiste em um tipo especial de árvore binária de busca onde cada nó tem um atributo cor, podendo ter valores RED e BLACK. Esse tipo de árvore rubro-negra utilizada é a caída para a esquerda, ou seja, é organizada de forma que os nós de cor vermelha fiquem sempre à esquerda de um nó pai (específico da árvore rubro negra caída à esquerda); o nó raiz sempre é preto; um novo nó é criado vermelho. Para que essas regras sejam cumpridas durante inserção é feito o balanceamento da árvore com base em 3 condições comparativos: Se o nó a direita é vermelho, se há 2 nós seguidos à esquerda da cor vermelha e se os dois filhos são da cor vermelha, onde para cada caso destes há uma modificação específica, sendo respectivamente o rotacionamento a esquerda, o rotacionamento a direita e a troca de cores.



Apesar da árvore rubro negra ser parecida com a árvore AVL, ela leva vantagem sobre a AVL, pois em sua função de balanceamento há apenas rotações simples diminuindo assim o tempo e custo de processamento na inserção.

2.3 Problema Proposto

Nessa única questão, é pedido para fazer uma loja de calçados em C, onde um sapato tem várias informações e essas informações têm que ser armazenadas em um arquivo txt. A partir disso, deve ser possível atualizar o arquivo txt quando um calçado chega ou sai da

loja, mostrar as informações de um calçado a partir de seu código e fazer um experimento para buscar trinta calçados mostrando o caminho percorrido e o tempo gasto.
A solução deste problema foi feita com as árvores já apresentadas (A23 e ARN_CE), e foi feita uma inserção de 3500 calçados, para uma melhor análise dos dados mostrados na próxima seção.

2.3.1 A23

Nesta implementação, inicialmente na main, a árvore 2-3 é criada e são inseridos nela informações sobre 3500 calçados que estão em um arquivo .txt chamado *calcados*. A seguir é feito um experimento que busca 30 calçados na árvore, onde há um laço de repetição que roda 30 vezes e gera códigos aleatórios de calçados e chama a função de busca, que mostra o caminho dessa busca na árvore, dizendo a direção que o algoritmo está tomando na árvores, podendo ser para esquerda (left), centro (center) ou direita (right), no fim, diz se o elemento foi encontrado ou não. O tempo para essas buscas também é coletado e mostrado ao fim das 30 buscas. Um menu é mostrado com as opções de inserir calçado, vender calçados, pesquisar calçado, checar estoque e uma função para salvar as alterações no arquivo .txt.

A seguir detalhamos o funcionamento de algumas funções:

- criaArv23(): cria a árvore e insere nela todos os calçados presentes no arquivo .txt chamado *calcados*;
- buscaTestes(): busca um calçado de acordo com seu código e retorna o nó em que ele está, caso exista;
- insereCalcado(): insere uma quantidade de calçado informada no calçado informado pelo código;
- vendeCalcado(): remove a quantidade de um calçado passado pelo código de acordo com a quantidade informada;
- buscaCalcado(): busca um calçado de acordo com o código passado e exibe todas as suas informações
- tabela(): mostra todos os calçados organizados como uma tabela;
- gravaArv23(): grava as alterações feitas no arquivo .txt chamado *calcados.*

2.3.2 ARN_CE

A implementação deste problema com a utilização da árvore rubro-negra caída para esquerda é, em geral, da mesma forma à árvore 2-3, onde vai haver mudanças somente em como os elementos são inseridos na árvore, pois diferente da árvore 2-3, vista na seção 2.1, na árvore rubro-negra, explicada na seção 2.2, ela não é totalmente balanceada, e na inserção de elementos ocorre rotações e trocas de cores dos nós, sem contar que nessa estrutura um nó só possui uma informação, ou seja, informações apenas sobre um calçado.

Assim, fora as características relacionadas com a forma com que as árvores são implementadas, o escopo geral do algoritmo para resolução deste problema ficou igual, reutilizando todas as características funcionais das funções explicadas anteriormente.

3. Resultados das Execuções dos Programas

Nesta seção será apresentado uma análise comparativa da busca dos dados resultantes da execução dos algoritmos implementados e utilizados na resolução do problema explicado na seção 2.3.

3.1 Árvore Rubro-Negra x Árvore 2-3

Os seguintes resultados mostrados, são resultados da inserção de 3500 (três mil e quinhentos) calçados inseridos, onde um teste de busca com 30 repetições foi feito para cada árvore mostrada (Seção 2.1 e 2.2).

Chamada da Função	**Resultado Mostrado**
buscaTestes(&tree, codigo, 0);	codigo: 984079 --> 0-center 1-center 2-right 3-right 4-right 5-right 6-right 7-left Encontrou: Sim

A tabela acima, mostra os parâmetros da chamada da função que busca os elementos para uma busca apenas, onde é passado a árvore, um código que é gerado aleatoriamente dentro do *for*, e um valor 0, que é um contador de quantos nós foram passados para chegar ao valor buscado. No resultado temos é mostrado o código a ser buscado antes de chamar a função, e na função, é mostrado as direções para qual filho da árvore as chamadas recursivas estão indo junto com um valor que conta essas passagens nesses nós, e por fim, após a função retornar esse nó em que o número está, caso este nó seja diferente de NULL, é mostrado uma mensagem dizendo se o número foi achado ou não.

ÁRVORE	Tempo médio para a Busca dos 30 Calçados (tempo em milissegundos)
Árvore 2-3	65.00 ms
Árvore Rubro-Negra	68.00 ms

O teste acima foi feito para o pior caso de busca nas duas árvores, ou seja, aquele caso em que toda a árvore é percorrida e o seu elemento não é encontrado. Podemos observar, que em relação a busca, o tempo calculado em milissegundos foi praticamente o mesmo, isso se deve porque, apesar da árvore 2-3 ter um custo maior para inserir devido a seus nós estarem totalmente balanceados sempre, ela possui 3 ponteiros para outros nós, logo isso pode gerar uma pequena elevação no tempo de busca, mas isso em casos com poucos elementos como esse caso testado. Uma função extra foi acionada para saber a profundidade do nó mais perto da raiz e dó nó mais distante da raiz dessa árvore, como ela é sempre balanceada, ambas as profundidades tiveram valor 9 nesse caso, com a inserção de 3500 calçados.

Observando agora a árvore rubro-negra, ela teve um tempo bem baixo também, e em alguns testes extras realizados, teve tempo de busca para 30 calçados até maior que a árvore 2-3, pois ela apesar mais ou menos balanceada, sendo caída para esquerda, para esse caso que houve a inserção de poucos elementos, ela se mostrou bastante eficiente, mas em casos com a presença de muitos elementos, sua busca já tem um custo mais alto em relação a árvore 2-3. Também foi feita uma função para saber a profundidade do nó mais perto da raiz, onde esse teve o valor 9, e o nó mais profundo em relação à raiz, teve valor 16, assim, podemos perceber e analisar, que em casos com mais elementos na árvore, essa diferença de profundidade se tornaria cada vez maior e, consequentemente, deixaria a busca por um elemento mais lenta.

4. Conclusão

A partir deste estudo, foi possível concluir, a partir das duas estruturas de dados em árvores mais complexas apresentadas, as características e rapidez das buscas dessas estruturas para o problema apresentado. E não apenas para este problema proposto, mas a partir de toda a explicação e apresentação dos resultados de eficiência dos algoritmos, já podemos ter uma boa base para determinar a melhor escolha de qual estrutura de dados usar para outros casos a partir do conhecimento sobre as propriedades de cada árvore.

Em relação a árvore 2-3, concluímos que é uma boa árvore para busca em grande banco de dados, ou seja, busca em uma árvores com muitos elementos, pois seus nós sempre estarão ordenados e isso facilitará a busca de um determinado elemento, sendo mais eficiente para estes casos, porém tem um custo de inserção maior.

Em relação a árvore rubro-negra, concluímos que para casos onde há uma menor quantidade de dados à ser inseridos, ela se torna uma boa opção, pois ela não é totalmente desbalanceada como a árvore binária, e tem assim, uma bom custo em relação ao tempo para a busca de elemento e também para a inserção, não sendo recomendada para casos onde há a presença de muitos elementos no banco de dados, pois a diferença entre o nó mais profundo e o menos profundo se torna cada vez maior.

Destarte, vimos as diferenças de cada estrutura de dados em árvore na solução de um problema e com essa análise parcial, foi aprendido a como implementar as árvores aplicadas a uma situação mais casual e a modificar seus elementos relacionando-os com arquivos .txt, assim, aprendendo cada vez mais sobre as diversas possibilidades que essas estruturas nos proporcionam.

5. Referências

VILLAS, Marcos V. et al. Estruturas de Dados . N.º 12. Local: Rio de Janeiro, Elsevier: Campus, 1993.

TENEMBAUM, Aaron M.; LANGSAM, Yedidyah; SOUZA, AUGENSTEIN, Moshe J. **Estruturas de Dados Usando C** . Local: São Paulo, MAKRON Books, 1995.