Estrutura de Dados II

29 de Outubro de 2018.

Relatório do Trabalho Prático.

1 Identificação do Relatório

Equipe: Vinicius Schnaider Zolet – RA: 1984136

Curso: Sistemas de Informação.

Departamento: DAINF.

Instituição: Universidade Tecnológica Federal do Paraná (UTFPR).

2 Introdução

Neste trabalho foi implementado os códigos, em C, para realizar tanto a operação de inserção quanto a de remoção em uma Árvore B e feita a criação de um arquivo Makefile.

Em vários aspectos, ela se assemelha com uma Árvore Binária de Busca, onde uma chave que será inserida irá para esquerda ou para a direita, se for menor ou maior, respectivamente, que as chaves que já estão inseridas. Ao mesmo tempo, tem a eficácia de uma Arvore AVL, uma vez que conforme as novas informações são inseridas a árvore irá se balancear.

Uma Árvore B se diferencia das ambas citadas pois ela apresenta mais de uma informação para cada nó da árvore, sendo T-1 o seu mínimo (com exceção da Raiz, que pode ter um mínimo de 1) e 2*T-1 o seu máximo de informações armazenadas. T é um número, maior ou igual que 2, definida pelo próprio programador/usuário na hora de gerar a Árvore B. Isso faz com que possamos acessar menos vezes o disco em relação as outras árvores de busca, assim ganhando bastante tempo de execução quando o número de informações presentes na árvore cresce.

Outra diferença, também, é que ela apenas cresce para cima. Seu crescimento se dá no momento que os nós folhas ficam cheio e se dividem, enviando o valor do meio do vetor para o nó pai. Eventualmente a raiz também estará cheia e irá expandir a Árvore B.

Tudo isso estará presente nas próximas sessões, com maiores detalhes.

3 Inserção

O trabalho se iniciou pela introdução, uma vez que é a parte mais simples entre ela e a remoção, além de já existir um pseudocódigo presente no livro de Cormem, tal qual utilizado pela docente da disciplina.

Alguns erros foram encontrados durante a implementação devido a necessidade de mudar algumas informações do pseudocódigo para a linguagem C, uma vez todo vetor de C é iniciado em 0 e Cormem inicia seu algoritmo em 1.

A parte que trouxe um pouco de dificuldade foi na hora de fazer a divisão de nós, pois aconteceram algumas confusões na hora de gerenciar qual variável receberia qual informação em determinados momentos. Sendo resolvido os erros de ordem, tudo se normalizou na inserção.

3.1 Funcionamento do Código

A inserção funciona de duas formas:

1. Inserir quando o nó se encontra cheio, ou seja, quando tem 2*T-1 chaves;

2. Inserir quando o nó não se encontra com 2*T-1 chaves;

Se for o primeiro caso, ele irá criar uma nova variável de árvore e irá dividir o nó atual, sendo o valor mediano do vetor se tornando o novo nó, que irá crescer para cima, e tudo que for menor que ele ficará armazenado em um filho à sua esquerda e tudo que for maior, à direita.

Em seguida ele passa a se tornar, automaticamente, o caso 2. Neste caso ele entra em uma função que irá procurar em qual filho ele irá ficar e, se for um nó folha, irá reorganizar o vetor para inserir a nova informação no local apropriado.

Toda vez que a raiz ficar cheia, ou seja, com 2*T-1 chaves, ela irá se dividir e criar uma nova raiz, fazendo assim com que a árvore cresça em profundidade. Essa é a única forma que essa árvore poderá crescer.

4. Remoção

A remoção foi um processo um pouco mais trabalhoso, uma vez que existem 6 casos diferentes de acordo com qual chave será removida e como está disposta a árvore naquele momento.

Neste momento houveram inúmeros problemas. Muitas das tentativas se mostraram erradas ou apenas problemáticas. Muitos dos problemas e dos erros se deram, enfim, devido ao erro com relação à onde iniciar um laço (de 0 para n ou de n para 0, se iria realmente até 0/n ou se pararia em T) ou por esquecer/não perceber a necessidade de colocar +1 ou -1 quando fosse descer para um filho do nó.

O maior problema foi na hora de descer para os filhos. Foi implementado uma função que verifica a necessidade do filho, a qual a chave está presente, ser preenchido caso tenha menos que T chaves. A função em si funcionou de forma simples, mas muitas vezes estava dando erro após isso, ao tentar descer para tal filho. Dentro desta função foi colocado uma função de “fundir”, para caso tanto o filho em si quanto o próximo (ou anterior, se ele fosse o último filho do vetor) tivessem menos que T chaves e quando essa função era realizada o código

entrava em conflito. Para a resolução foi necessário criar um auxiliar que marca como positivo (1) caso a posição atual da busca (index) seja igual a posição final do vetor (n) e esse auxiliar é utilizado após a função preencher, se ela for necessário, pois se isso fosse verdade e os 2 últimos filhos tivessem sido fundidos, agora a posição final do vetor seria (n-1) com relação a antes e index apontaria para algo que já não existe.

Após a implementação desse auxiliar, tudo se estabilizou e foi possível realizar todos os testes de remoção com sucesso.

4.1 Funcionamento do Código

O código inicia primeiramente verificando se a árvore não está vazia, para não dar nenhum problema de tentar tirar algo que não existe e passa, então, para a remoção. É feito uma busca do local aonde é para estar a chave a qual se deseja remover e retorna esse valor, sendo realizado então a verificação se na posição da qual foi encontrada o valor que se deseja remover é o mesmo que se encontra no vetor, levando então a duas situações:

1. Se coincidir, é verificado se é um nó folha ou não.
2. Se não coincidir, é verificado se é um nó folha e, se não for, continua o algoritmo.

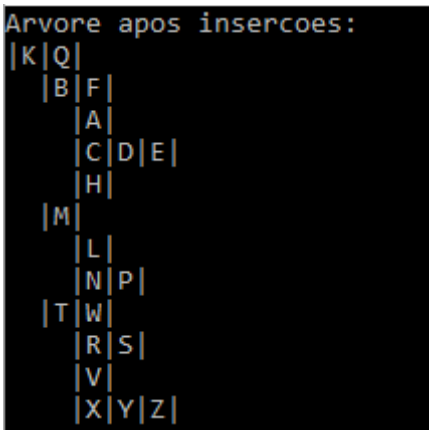
Se for a segunda situação e ele for um nó folha, quer dizer que a chave que se busca remover não se encontra no vetor. A continuação do código se dá na verificação da necessidade de preencher o filho, caso tenha menos que T chaves. Tal preenchimento pode ser feito tanto ao receber uma chave do seu filho que tenha T ou mais filhos (se for o da esquerda, pegará a maior chave; se for o da direita, pegará a menor chave) e irá substituir a chave “pai” do nó, inserindo a que estava nesse local no nó que continha menos de T filhos. Se nenhum dos dois filhos tiverem T ou mais filhos é feito a fusão dos dois e inserindo esta chave do nó pai no meio entre as chaves do filho da esquerda e das chaves do filho da direita. Sendo feito isso é chamado então, recursivamente, a função de remoção em seu filho dado pela posição de index ou index-1, caso tenha sido feito a fusão dos filhos.

Se for a primeira situação, ele iniciará uma função para remover tal chave, seja ele um nó folha (uma função) ou um nó interno (outra função). Se for um nó folha ele apenas irá eliminar tal chave. Se ele for um nó interno é necessário fazer algumas outras verificações.

Tais verificações são para ver se o filho da esquerda ou da direita tem pelo menos T chaves. Caso algum deles tenha é iniciado uma sequência de códigos que irá buscar a maior ou menor chave, dependendo qual dos filhos tem T ou mais chaves, sendo maior se for o da esquerda e menor se for o da direita. Se nenhum dos dois tiverem pelo menos T filhos é feito a fusão de ambos. É chamado então, recursivamente, a função remover.

5 Resultado dos testes e os casos de remoção

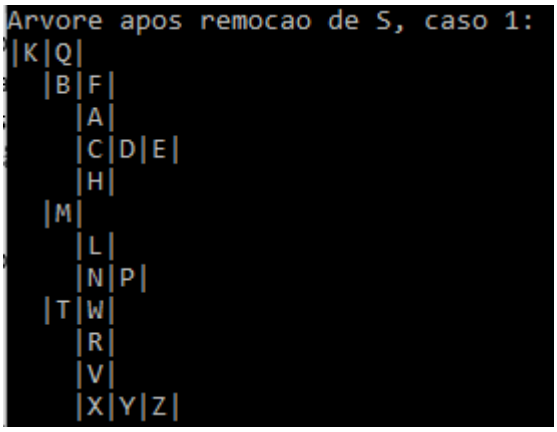
Os testes foram realizados de acordo com as orientações da professora, sendo feito a inserção das chaves e depois para pelo menos 4 casos de remoção (que no caso foram feitos 5).

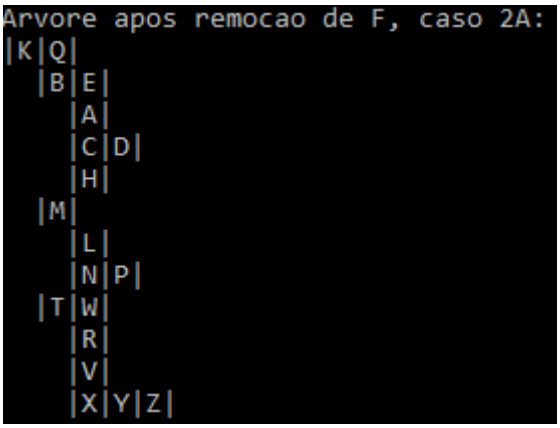


Após rodar o código, o resultado da inserção é o esperado. Todas as chaves estão como o desejado e está correto segundo a forma a qual a própria docente descreve como correto em suas orientações.

A remoção, como já dito, funcionam em 6 casos diferentes, tais casos são distribuídos como Caso 1, 3A e 3B, para nós folha, e Caso 2A, 2B, 2C, para nós internos. Os casos testados foram o 1, 2A , 2C, 3A e 3B, respectivamente.

O caso 1 é o caso mais simples e o resultado foi como esperado. Ele consiste em, basicamente, tirar a chave de um nó folha que contenha pelo menos T filhos. Ou seja, se remover tal chave o nó folha continuará tendo pelo menos T-1 filhos, o que é o mínimo e não irá afetar em nada a árvore. Nesse exemplo foi removido S, sem nenhuma alteração na estrutura da árvore como observado na imagem ao lado. (R < T < V)

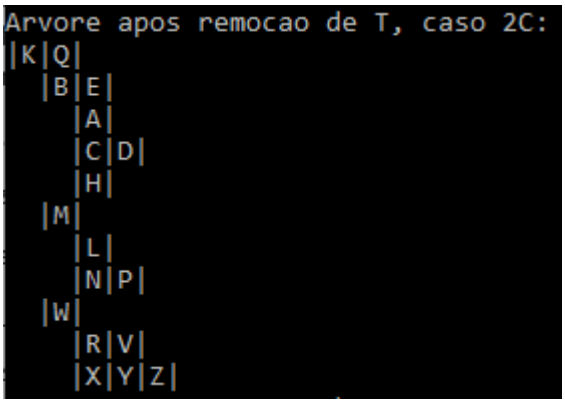


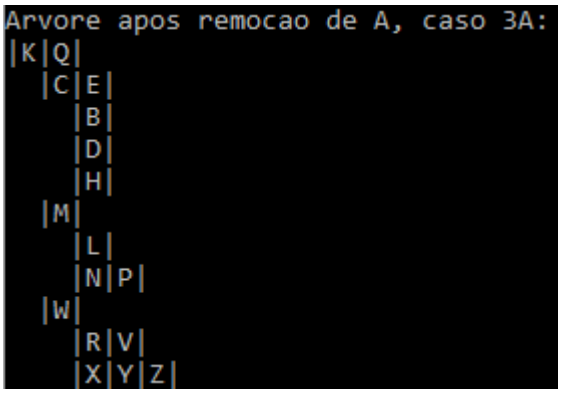


Os casos 2A, 2B e 2C consistem em remover chaves de nós internos, ou seja, nós que possuem filhos e não podem simplesmente ser removidos como no caso 1. O caso 2A consistem em remover uma chave de um nó interno quando seu filho da esquerda tem pelo menos T filhos. Sendo assim, é removido tal chave e a maior chave de seu filho da esquerda, seja do próximo filho ou do último, assume sua posição. Para esse exemplo foi removido a chave F. Podemos observar que, diferente do caso 1, houve algumas mudanças na Árvore. A chave E foi “promovida” para o antigo local de F, visando manter o balanço da Árvore. (C < D < E < H)

O caso 2B é basicamente o mesmo, mas o que muda é que o filho da esquerda não tem T ou mais chaves e o da direita sim. Nesse caso busca-se o menor ao invés do maior.

O caso 2C é necessário quando nem o filho da esquerda e nem o filho da direita tem pelo menos T chaves. Quando isso ocorre é feito a fusão de ambos os filhos, colocando as chaves do filho da direita no filho da esquerda e após a última chave nele presente. Depois é eliminado a chave a qual desejamos e as chaves que estavam no mesmo nó e após ela são movidos 1 posição para trás no vetor, mantendo assim o equilíbrio. Como mostra a imagem ao lado, foi removido T para este exemplo. Os seus antigos filhos, R e V, são fundidos em um único filho, T é removido e W, junto com seu filho, é movito 1 posição para trás. É mantido assim o equilíbrio. (R<V<W<X...)





O Caso 3A é referente quando a chave que queremos remover está presente em um nó com menos que T chaves e algum dos filhos adjacentes a sua posição, seja o próximo ou o anterior, contem pelo menos T filhos. Nesse caso a chave do nó pai descerá para filho que não possui T chaves (assumindo a primeira posição do vetor se for para o filho da direita ou a posição subsequente a ultima chave se ele for para a esquerda) e a chave do filho que possui T ou mais chaves subirá para assumir sua posição (sendo a ultima chave se vier do filho da esquerda ou a primeira chave se vier da esquerda). O nó é removido então normalmente. Como podemos observar na imagem, para a remoção de A, um nó com T-1 chaves, foi necessário que B fosse para esse nó e C assumisse sua posição. Assim, foi mantido o equilíbrio na árvore. (B<C<D<E<H)

O Caso 3B consistem em remover um nó folha que tenha menos de T chaves e nenhum de seus irmãos adjacentes tenha pelo menos T. Neste caso será feita uma fusão de nós, parecida com o caso 2C. Tal filho com menos de T chaves a qual possua a chave que será excluída será fundida com seu irmão anterior. Para tal, primeiro a chave é excluída. A chave que estava presente no nó pai irá descer para assumir a próxima posição livre no vetor filho da esquerda e as chaves do filho a qual a chave foi excluído será inserido em sequência. Se o nó for o primeiro, ele irá receber as chaves do seu próximo irmão depois do nó pai assumir seu lugar. Feito isso, as chaves do nó interno se movem em -1 posição do vetor, para manter o equilíbrio. Como observado na imagem, ao removermos a chave B a chave C, seu antigo “pai”, desce para seu local, a chave D é inserida logo após ela e o E é movido 1 posição a menos no vetor, mantendo assim o equilíbrio. (C<D<E<H)