UFAM

UNIVERSIDADE FEDERAL DO AMAZONAS

FT - FACULDADE DE TECNOLOGIA

FT05 – ENGENHARIA DA COMPUTAÇÃO

JOSIAS BEN FERREIRA CAVALCANTE

Trabalho 1: Árvore AVL e Problema

Manaus-AM

2019

2175359 – Josias Ben Ferreira Cavalcante

Árvores AVL e Solução de Problema

Trabalho de aproveitamento para a disciplina de Algoritmos e Estruturas de Dados II, ministrado pelo Professor Edson Nascimento, para o curso de Engenharia da Computação, na Universidade Federal do Amazonas.

Manaus-AM

2019

INTRODUÇÃO

Uma Árvore AVL é uma árvore binária de busca balaceada, ou seja, uma árvore balanceada são árvores que minimizam o número de comparações efetuadas no pior caso para uma busca com chaves de probabilidades de ocorrencias identicas.

Contudo, para garantir essa propriedade em aplicações dinâmicas, é preciso reconstruir a árvore para seu estado ideal a cada operação sobre seus nós, para ser alcaçado um custo de algoritmo com o tempo de pesquisa tendendo a O(log n).

OBJETIVO

Solucionar o problema proposto em sala de aula, fazendo aplicação da Árvore AVL e seus resursos de minimização de espaço de busca.

PROBLEMA

Considerando o que foi discutido em sala de aula, sobre o serviço de agendamento de atendimento de uma hipotética clínica de psicologia, onde teremos uma serviço colaborativo.

Deve-se construir um programa que faça a marcação de sessões terapêuticas, usando para registrar essas marcações árvores AVL.

Para gerar os pedidos, deve-se perguntar inicialmente o número de pedidos, e com esse número gerar aleatoriamente o dia e hora de todos os pedidos. Lembre-se que pode haver pedidos que não poderão ser atendidos por falta de disponibilidade de consultório.

É indicado que se crie um arquivo de log para guardar todas as solicitações na ordem que ocorreram.

Levar em consideração os seguintes aspectos:

A) Considere que a clínica possui 4 consultórios, com disponibilidade para atendimento, de 2a. a 6a. feira, em três turnos:

* Turno matutino: das 9h às 12h, sendo o último horário de marcação às 11h. Portanto 3h de atendimento.

* Turno vespertino: das 13h às 18h, sendo o último horário de marcação às 17h. Portanto 5h de atendimento.

* Turno noturno: das 19h às 22h, sendo o último horário de marcação às 21h. Portanto 3h de atendimento.

Assim, cada consultório possui a disponibilidade máxima de 10h de atendimento por dia.

B) Considere ainda que não há limites de terapeutas. Apenas que cada terapeuta possa atender no máximo 3h por dia, e 10h por semana.

C) Considere que o paciente pode realizar o pedido de atendimento por horário (dia e hora).

D) Ao ser confirmado o horário (dia e hora) do atendimento, aquele horário, naquele consultório passa a ficar bloqueado para outros atendimentos por um tempo que o terapeuta julgar necessário.

E) Ao ocorrer o atendimento no horário e consultório previsto, o terapeuta ira registrar o atendimento, e confirmar o próximo atendimento do referido paciente atendido, normalmente para a semana subsequente, no mesmo horário e consultório.

F) O sistema deverá apresentar relatórios de marcação de atendimentos para o paciente, para o terapeuta e para o administrativo. Podendo estes serem diários, semanais e mensais.

FUNÇÕES PRINCIPAIS

A estrutura da AVL usada para a solução do problema foi a seguinte:

```
typedef struct dados{
    int consultorio;
    int dia;
    int hora;
    int paciente;
    int psicologo;
} Dados;

typedef struct arvoreavl{
    Dados *DATA;
    int chave;
    int alt; // será a altura de determinado nó;
    struct arvoreavl *filhoDir, *filhoEsq;
} AVL;
```

A função insereAVL funciona da seguinte forma:

É composta de uma repetição recursiva, retorna um valor inteiro (0/1) que servirá de flag futura para verificar se a inserção foi feita com sucesso.

Primeiro passo da inserção é uma verificação se a raiz é nula (se a arvore é nula ou se chegou ao final de uma folha) e insere o nó alí.

Caso contrário, procura-se o lado do nó a se seguir a partir de comparações com a chave passada e chave da raiz, e entra numa recursão a qual procura inserir o elemento até que haja sucesso ou fracasso. Ao fim da inserção recursiva, busca-se verificar o fator de balanceamento dos nós voltando da recursão. Houver necessidade de realizar rotações.

A função removeAVL funciona da seguinte forma:

Assim como a de inserção, a função de remoção retorna uma flag representando o sucesso ou o fracasso da remoção. Primeiro verifica-se se a raiz passada como parâmetro é nula; se sim, retorne 0 pois não há nada a ser removido do nada.

Faz comparação de chaves entre a raiz e a chave passada a fim de determinar a direção do andar da função, buscando o elemento a ser removido. Encontrado o elemento a ser removido tratar do caso de ele ser folha, pai de um ou dois filhos. Ao fim da remoção, verificar fatores de balanceamento voltando da recursão, e aplicando rotações caso necessário.

A função buscaNo funciona da seguinte forma:

Inicialmente feita para Árvores Binárias de Busca, a função buscaNo foi reaproveitada para a Árvore AVL pois funciona igualmente. A função recebe como parâmetro uma raiz e uma chave a ser buscada. Primeiro verifica-se se a raiz é nula; se sim, retorna NULL. Depois verifica-se se a chave passada é igual a chave da raiz; se sim, encontrou o nó buscado e o retorna. Depois verifica se a chave é menor ou maior que a chave da raiz, e entra em recursão passando na função o filho direito ou esquerdo, e a chave buscada.

IDEIAS PARA A SOLUÇÃO DO PROBLEMA

A chave a ser usada para a arvore seria uma composição de dia+hora+sala. Por exemplo, deseja-se inserir uma reserva para dia 2, às 15 horas, na sala 3; a chave deste nó seria (2 * 1000) + (15 * 10) + 3 = 2153.

A ideia geral para o funcionamento do programa era criar um vetor estático do tipo AVL* para cada semana (total de 4 semanas), e a partir daí criar um menu de tela sugerindo as seguintes opções:

1) **Simular inserção de agendamentos aleatórios**: Onde é perguntado do usuário a semana e a quantidade de dados aleatórios a serem gerados. Enquanto a arvore da semana em questão não ficar cheia (240 nós) ou findados a quantidade de dados aleatórios a serem criados, utilizarei uma função padrão do stdlib.h chamada rand(), para criar variáveis de valores aleatórios para o dia da semana (1~5), a hora (9~20) e a sala do consultório (1~4). Tais valores seriam passados para uma variável do tipo Dados *dados e este posteriormente seria passado para a variável DATA contida dentro do tipo AVL. O no tipo AVL resultante iria ser inserido na arvore referente a semana selecionada, observando sempre a flag resultante da função de inserção e fazendo um contador de sucessos de inserção. O resultado deste contador seria exibido ao fim das inserções aleatórias.

2) **Exibir uma determinada reserva:** Faria uso da função de busca. Perguntaria do usuário a semana, o dia, a hora e a sala marcada. Tomaria conhecimento da chave do nó procurado e iniciaria a busca na arvore referente a semana digitada pelo usuário. Encontrado a reserva, exibiria sem conteúdo.

3) **Exibir situação de um determinado horário/dia:** Faria uso de uma função semelhante a função exibirAVL, adaptada para exibir os nós que satisfazerem a condição da busca: Se eu busco as reservas de determinado dia, a função só retornará as informações daquelas que possuem o primeiro numero da chave igual ao dia buscado; o semelhante acontece para buscar o horário.

4) **Remover N reservas:** O funcionamento seria semelhante ao da primeira opção, no entanto, seria perguntado do usuário qual semana remover dados e a quantidades de chaves aleatórias seriam geradas. A função de remoção seria usada sob a arvore da semana e uma chave gerada aleatoriamente. O retorno da função é uma flag (0/1), onde uma variável contadora armazenaria a quantidade de 1’s (sucessos de remoção) e exibiria tal valor ao fim das remoções. O fim das remoções seria imediato caso a arvore ficasse vazia.

5) **Exibir a situação das reservas num período de uma semana:** Fazendo uso da função ExibeAVL, perguntaria do usuário qual arvore (semana) exibir, e exibiria a situação (dia, hora, sala, medico, paciente) de cada agendamento em ordem crescente à sua chave.

6) **Exit:** Opção que finaliza o programa.

CONCLUSÃO E ESCLARECIMENTOS

A implementação de uma árvore AVL ajuda na redução do espaço de busca, eliminando a ocorrência de árvores que tendem a uma única direção ou que tem aparência de “zigue-zague” (“Árvore troncha” nas palavras do professor), diminuindo assim a diferença de altura entre os nós.

O código de solução do problema não foi finalizado; encontra-se incompleto.

A causa disto se deve a má administração do tempo trabalhado no código, também em decorrência da repentina desistência da disciplina (trancamento) por parte de um componente da equipe, tornando dificultosa a realização de todo o trabalho por apenas uma pessoa.

Referências

- Árvore AVL – Wikipédia A enciclopédia livre - https://pt.wikipedia.org/wiki/Árvore_AVL acesso em: 27/04/2019 20:06.