Análise comparativa de árvores binárias em uma aplicação de contagem de calorias

Estruturas de Dados - 23/2

Prof. Viviane Moreira
Guilherme Kohl (00579400) e Lucas Slongo Schiavo (00578225)

Contextualização

O objetivo do trabalho é comparar o desempenho de diferentes Árvores Binárias de Pesquisa (ABP's) em uma aplicação de contagem de calorias. Mais especificamente, uma nutricionista quer automatizar o processo de cálculo do consumo diário de calorias de seus pacientes.

Para isso, deve-se construir uma aplicação que insere os dados de uma tabela nutricional em uma árvore, acessa esses dados conforme uma outra tabela com os alimentos consumidos pelo paciente e produz um relatório contabilizando as calorias consumidas em cada alimento e o total do dia.

Neste caso, o programa recebe dois arquivos csv (tabela nutricional e alimentos consumidos) e retorna um arquivo txt (relatório).

Desenvolvimento do trabalho

Para essa aplicação, decidimos comparar o desempenho de uma ABP padrão com as árvores AVL e Splay. A ideia era averiguar as diferenças entre uma árvore sem balanceamento, uma que se auto balanceia e outra que efetua o balanceamento no acesso de nodos.

Após, planejamos os tipos de dados estruturados que seriam necessários para a aplicação:

```
typedef struct food{
    char name[50];
    int calories;
} FOOD;
```

Estrutura 'Food':
Um array de caracteres para o nome do alimento;
Um número inteiro para o número de calorias por 100g;

```
typedef struct nodeAVL{
    FOOD food;
    int height;
    struct nodeAVL *left;
    struct nodeAVL *right;
} nodeAVL;
```

Estrutura 'nodeAVL':
Uma estrutura 'food' para o alimento;
Um número inteiro para a altura do nodo;
Dois ponteiros 'nodeAVL' para os filhos do nodo;

```c
typedef struct nodeBST{
    FOOD node;
    struct nodeBST *left;
    struct nodeBST *right;
} nodeBST;
```

Estrutura ‘nodeBST’*:
Uma estrutura ‘food’ para o alimento;
Dois ponteiros ‘nodeBST’ para os filhos do nodo;

* notar que a estrutura ‘nodeBST’ pode ser usada para ABP e Splay

Com isso, partimos para a programação das funções que cada árvore exigia para seu funcionamento:

- AVL: função de criação de árvore, inserção, rotação simples à direita e à esquerda, cálculo de fator de balanceamento, consulta e cálculo de altura.
- ABP: função de criação de árvore, inserção, consulta e cálculo de altura.
- Splay: função de inserção, splaying, rotação simples à direita e à esquerda.

Observar que as funções de criação e de cálculo de altura de ABP podem ser usadas para a árvore Splay e que a função de consulta em Splay é o próprio splaying.

É importante mencionar que foram utilizadas variáveis globais para contar o número de comparações (comp) e rotações (rotations) durante os processos de inserção e consulta nas árvores:

```c
//GLOBAL VARIABLES:
int comp_AVL=0, comp_BST=0, comp_SPLAY=0;
int rotations_AVL=0, rotations_SPLAY=0;
```

```c
nodeAVL* consultAVL(nodeAVL* node, char* key){
    if (!key) return NULL;

    while (node){
        comp_AVL++;

        if (strcmp(node->food.name, key) == 0){
            return node;
        }
        if (strcmp(node->food.name, key) > 0){
            node = node->left;
        }
        else{
            node = node->right;
        }
    }
    return NULL;
}
```

A variável comp_AVL é atualizada a cada vez que um nodo é acessado na consulta. Esse processo é igual para a ABP e semelhante para a Splay.

```
int balance = AVL_getBalance(node);

if (balance > 1 && strcmp(newFood.name, node->left->food.name) < 0){
    rotations_AVL++;
    return AVL_rightRotate(node);
}

if (balance < -1 && strcmp(newFood.name, node->right->food.name) > 0){
    rotations_AVL++;
    return AVL_leftRotate(node);
}

if (balance > 1 && strcmp(newFood.name, node->left->food.name) > 0){
    rotations_AVL++;
    node->left = AVL_leftRotate(node->left);
    return AVL_rightRotate(node);
}

if (balance < -1 && strcmp(newFood.name, node->right->food.name) < 0){
    rotations_AVL++;
    node->right = AVL_rightRotate(node->right);
    return AVL_leftRotate(node);
}
```

A variável rotations_AVL é atualizada a cada vez que uma operação de rotação é efetuada. Esse processo é semelhante para a Splay, com a diferença que esta não possui fator de balanceamento.

* trecho da função de inserção da avl

Por fim, a função main utiliza a função ‘fgets’ para ler cada linha do arquivo csv e ‘strtok’ para separar a linha em nome do alimento e calorias. Após, atribui esses valores em uma estrutura ‘food’ e insere a mesma nas três árvores.

Um processo semelhante é usado na consulta dos alimentos. Para isso, utiliza-se novamente as funções ‘fgets’ e ‘strtok’, agora separando o nome do alimento de quantas gramas foram consumidas. Então, efetua-se uma busca nas árvores utilizando o nome do alimento como chave e calcula-se as calorias baseadas no número de gramas.

Análise comparativa

Com o propósito de comparar o desempenho das árvores, começamos criando bases de dados para diferentes propósitos. Para gerá-las, utilizamos a biblioteca Pandas da linguagem Python para criar e editar arquivos csv.

Para a análise, criamos quatro bases: em ordem alfabética e embaralhada, cada uma com uma versão de 1.000 e 10.000 itens. Os números representando as calorias por 100 gramas foram gerados aleatoriamente e os nomes dos alimentos são palavras de 3 caracteres seguindo a ordem “aaa”, “aab”, “aac”,

…, “aba”, … (posteriormente embaralhados quando este for o caso). Após, criamos os arquivos do consumo diário do paciente selecionando aleatoriamente 15 alimentos da base de dados.

Então, rodamos o programa chamando a base de dados e seu arquivo de consumo diário obtendo assim, quatro relatórios com informações de número de comparações e rotações. Após, montamos as seguintes tabelas com os dados obtidos:

Comparações	1000	10000
ABP Sorted	7366	86765
ABP Shuffled	178	275
AVL Sorted	131	192
AVL Shuffled	142	199
SPLAY Sorted	727	7205
SPLAY Shuffled	100	158

Rotações	1000	10000
AVL Sorted	990	9986
AVL Shuffled	433	4639
SPLAY Sorted	1439	14399
SPLAY Shuffled	13083	193879

A ABP, apesar de ter fácil implementação, tem o pior desempenho quando se trata de comparações para encontrar o nodo procurado em arquivos ordenados. Assim, quando utilizada com uma grande quantidade de dados, tem-se um custo computacional muito maior. Por outro lado, a AVL, por ser auto-balanceada, apresenta um número muito menor de comparações. Apesar da splay efetuar, em alguns casos, menos comparações do que a AVL, deve-se notar que o seu número de rotações é, por muito, superior.

Em um outro contexto de aplicação, onde um mesmo nodo da árvore devesse ser consultado mais de uma vez, a splay poderia vir a performar melhor, já que os nodos mais acessados estariam próximos à raiz. Por exemplo, digamos que o paciente seguisse uma dieta restrita e comesse, ao longo de um período de tempo, os mesmos alimentos e que o programa considerasse mais de um dia de dados, a splay, no longo prazo, realizaria cada vez menos comparações.

Com isso, ao analisar os dados obtidos é perceptível que a AVL teve o melhor desempenho, não variando muito conforme os tipos de ordenamento da base de dados e mantendo sempre uma baixa quantidade de comparações mesmo aumentando o número de nodos.