Contagem de calorias usando Árvores Binárias de Pesquisa

Ian Kersz Amaral[1] e Nathan Alonso Guimarães[1]

[1] Engenharia de Computação, Universidade Federal do Rio Grande do Sul, Brasil

O objetivo deste trabalho é comparar o desempenho de 4 diferentes Árvores Binárias de Pesquisa em uma aplicação de contagem de calorias. Uma aplicação de teste com 1000 comidas foi desenvolvida e utilizada para medir a performance de cada uma. Com base nela, foi possível concluir que, para um número mais elevado de dados, as árvores RN e AVL são as mais eficazes para essa aplicação, devido às suas rotações que visam balancear as árvores, diminuindo as comparações.

INTRODUÇÃO

Utilizando os 4 tipos de árvores vistas em sala de aula, ABP, AVL, Rubro-Negra e Splay, foi construída uma aplicação para inserir nodos nelas a partir de um arquivo com 1000 tipos de comidas, seus nomes e suas calorias por 100 gramas. Por conseguinte, um segundo arquivo contendo a lista de comidas ingeridas pelo indivíduo e sua quantidade em gramas foi lida e um arquivo de saída foi gerado, contendo um sumário das calorias por comida e das calorias totais. Junto no arquivo de saída, são geradas as estatísticas de performance de cada árvore, contendo a quantidade de nodos, altura, número de rotações e de comparações.

ESTRUTURAS DE DADOS E FUNÇÕES UTILIZADAS

Para facilitar a manipulação de tantas informações simultâneas sobre o trabalho (arquivos, estatísticas das árvores, as próprias árvores em si, etc.), foi feito o uso de diversas *structs* e funções autorais. O código foi todo fatorado em diversos arquivos, evitando assim que a *main.c* ficasse abrupta, desproporcional e fácil de se perder. Ao fim, ele foi dividido em:

1. *files.c*, cujas funções lidam com a leitura do arquivo de entrada e escrita do arquivo de saída;

```c
int getFoodFromFile(Food *food, FILE *file)
{
    if (food == NULL || file == NULL)
    {
        return -1;  // Retorna -1 se algum parametro for nulo
    }

    char temp[1000]; // Variavel para percorrer o arquivo
    if (fgets(temp, 1000, file) == NULL)
    {
        food = NULL;        // Inicializa o alimento como nulo
        return 1;           // Retorna 1 se chegar no final do arquivo
    }
    if (strchr(temp, ';') == NULL)
    {
        food = NULL;        // Inicializa o alimento como nulo
        return 1;           // Retorna 1 se chegar no final do arquivo
    }

    strtok(temp, ";");      // Remove o ; do nome do alimento

    // Copia o nome do alimento para o struct
    strcpy(food->name, temp);

    // Converte o nome do alimento para minusculo
    for (int i = 0; i < strlen(food->name); i++) food->name[i] = tolower(
food->name[i]);

    // Pega a quantidade de calorias por 100 gramas do alimento
    food->calories = atoi(strtok(NULL, ";"));

    return 0; // Retorna 0 se nao houver erros
}
```

Imagem 1 - Exemplo de função na *files.c* para leitura das tabelas.

2. *types.c,* que armazena structs que serão usadas durante o código como controle das estatísticas das árvores;

```c
// Structs para as estatisticas de saida
typedef struct TreeStats
{
    char name[100];  // Nome da arvire
    int counters[4];
  // 0 numNodes, 1 height, 2 rotations, 3 comparasions
} TreeStats;
```

```c
TreeStats initCounter(char name[])
{
    TreeStats stats;
    strcpy(stats.name, name);
    stats.counters[0] = 0;
    stats.counters[1] = 0;
    stats.counters[2] = 0;
    stats.counters[3] = 0;
    return stats;
}
```

Imagens 2 e 3 - Struct que vai cuidar das estatísticas das árvores e a função que a inicializa.

3. *trees.c*, o qual apresenta funções e propriedades divididas por todas as árvores;

```c
Tree *consulta(Tree *root, char *key, int *comp)
{
    while (root != NULL)
    {
        (*comp)++;
        if (!strcmp(root->info.name, key))
        {
            (*comp)++;
            return root;
        }
        else
        {
            (*comp)++;
            if (strcmp(root->info.name, key) > 0)
                root = root->L;
            else
                root = root->R;
        }
    }
    return NULL;
}
```

Imagem 4 - Função consulta, fornecida no enunciado do trabalho, se encontra aqui, pois é a mesma para todas as árvores.

4. *ABP.c*, *AVL.c*, *RN.c* e *Splay.c*, que cria as funções respectivas às suas árvores.

```c
typedef struct ABP
{
    Typeinfo info;
    ABP *L;
    ABP *R;
} ABP;
```

```c
typedef struct AVL
{
    Typeinfo info;
    struct AVL *L;
    struct AVL *R;
    int k;
}AVL;
```

```c
typedef struct RN
{
    Typeinfo info;
    struct RN *L;
    struct RN *R;
    struct RN *P;
    int red;
  //1 se nodo é vermelho, 0 se é preto
} RN;
```

```c
typedef struct Splay
{
    Typeinfo info;
    struct Splay *L;
    struct Splay *R;
} Splay;

```

Imagens 5 a 8 - Structs representantes de cada árvore.

Assim, fez-se uma *main* muito mais clara e objetiva, que tem como papel apenas fazer a chamada das funções dos outros arquivos. Nela, as árvores serão criadas, preenchidas, consultadas e, antes de destruídas, passarão suas estatísticas para o arquivo de saída.

RESULTADOS E DISCUSSÃO DAS ÁRVORES

Com a parte lógica do código pronta, foi possível testar a aplicação (e consequentemente, o desempenho das árvores). Devido a quantidade de dados passada, uma comparação em base do tempo não é muito útil, já que, utilizando um computador de performance média, todo o programa, incluindo a inserção e busca em todas as árvores, dura 7 milisegundos. Portanto, para compararmos as árvores, será utilizado somente o número de comparações durante a busca, a altura final da árvore, e, onde aplicável, o número de rotações realizadas pela árvore. O número total de nós nas árvores sempre será o mesmo, visto que o conjunto de dados tem 1000 comidas e todas devem ser inseridas corretamente. Assim, testamos dois casos específicos:

No primeiro, as árvores foram montadas tendo como base o arquivo *1000Shuffled.csv*, que continha todas as comidas que podem ser fornecidas pelo usuário e suas calorias por 100 gramas, organizadas aleatoriamente. O *day1.csv*, que continha as comidas ingeridas pelo usuário durante certo dia, foi usado como parâmetro para as pesquisas dentro das árvores. Passando a terceira informação necessária para o programa, o nome do arquivo de saída, como *saida_day1_shuffled.txt*, foi possível obter tanto as informações úteis para a consulta do paciente, quanto às estatísticas de performance de cada árvore durante a execução do programa.

Com isso em mente, os seguintes dados foram obtidos para cada árvore analisada:

	ABP	AVL	Rubro-negra	Splay
Altura	22	12	12	22
Rotações	0	449	380	5490
Comparações	144	114	117	144

Tabela 1 - Saída do código quando a entrada são os dados embaralhados.

Quando comparamos qual tipo de árvore teria mais eficiência da aplicação, podemos rapidamente descartar a árvore Splay, tendo em vista que ela foi criada para solucionar problemas onde poucos nós são

consultados muitas vezes, baseado em sua popularidade. Após isso, todas as árvores têm performance em par umas com as outras, isso se deve ao tamanho do conjunto de dados que é relativamente pequeno.

Apesar da ABP ter uma altura significantemente maior que as duas outras árvores, foi observado que seu número de comparações apresentado não é muito maior, tendo em conta que na inserção, as outras árvores tiveram que fazer um número considerável de rotações para obter suas alturas menores. Caso mais comidas fossem acessadas para consulta, seria possível ver que a AVL e a árvore RN apresentariam uma performance muito mais rápida.

É possível também estimar que a árvore RN terá uma performance total melhor que a AVL nessa aplicação com mais dados de consulta, visto que, ela apresenta um número significante de rotações a menos com um número de comparações a mais bem pequeno.

Já para o segundo caso de teste, foi usado o arquivo *1000Sorted.csv*, que difere do usado anteriormente pois contém os dados ordenados. Ou seja, as inserções de novos nodos nas árvores serão feitas sempre com valores “maiores” (seguindo a ordem alfabética, nesse caso) que o anterior. Novamente foi usado o arquivo *day1.csv* para percorrer a árvore. Como saída, obtivemos o arquivo *saida_day1_sorted.txt*, com as seguintes estatísticas atualizadas:

	ABP	**AVL**	**Rubro-negra**	**Splay**
Altura	867	10	17	69
Rotações	0	986	979	1030
Comparações	3219	109	102	1780

Tabela 2 - Saída do código quando a entrada são os dados ordenados.

Para este caso, é visível que a árvore Splay deixa de ser a pior, levando a ABP a tomar seu antigo lugar. Como os dados são praticamente ordenados (se fossem 100% ordenados, a ABP deveria ter altura 1000), os novos nodos são sempre inseridos na direita, gerando uma sub-árvore direita enorme, com poucas ramificações à esquerda. O que deveria ser uma árvore mais balanceada se torna praticamente uma lista simplesmente encadeada, com uma ocupação de espaço na memória muito maior. Isso justifica também o valor alto na saída das comparações. Por exemplo, caso se quisesse visitar o nodo que armazena as informações do alimento “Zucchini”, último no arquivo ordenado, teria-se que percorrer a árvore inteira, gerando algo próximo das 1000 comparações (seriam exatamente 1000 no caso perfeitamente ordenado).

Este é o caso de teste perfeito para pôr em prática a funcionalidade das rotações na AVL e Rubro-Negra. Elas existem justamente para reorganizar a árvore de forma a mantê-la o mais balanceada possível, cada uma com seus critérios únicos para assim o fazerem. Apesar de terem feito números próximos a 1 rotação por nodo na árvore, reduziu-se muito o número de comparações necessárias para acessar os dados, quando comparadas à ABP.

Faz-se interessante a comparação de que, nas árvores RN e AVL, o número de comparações foi ligeiramente menor com a entrada ordenada, se mantendo próximo de uma razão 1:1 com o arquivo embaralhado, o que indica que elas têm comportamentos parecidos e independentes de como são os dados de entrada. Já para a ABP e Splay, a situação se torna um pouco mais caótica, com um número de comparações estrondosamente maior com o arquivo ordenado, o que nitidamente as torna mais

devagar e menos eficientes para estes casos.

	ABP	AVL	Rubro-negra	Splay
Comparações	22,35x	0,96x	0,87x	12,36x

Tabela 3 - Razão entre as comparações da Tabela 1 e 2.

A Splay aparece agora numa posição mediana. Apresentou uma considerável piora no desempenho, pois novamente não está sendo usada com o propósito original dela (tal qual AVL e RN estão), mas ainda sim se coloca à frente da ABP, tendo aquela feito quase metade das comparações feitas por esta.

CONCLUSÕES

A partir da análise dos dados apresentados previamente, conseguimos chegar a conclusão que a adoção da árvore RN ou AVL seria o mais indicado para esse tipo de aplicação. Isso se baseia na noção de que, diferente desses casos de testes apresentados, será analisado um conjunto de dados bem maior, ou até mesmo se vários dias juntos, sem necessitar a reconstrução da árvore, Esse comportamento pode ser obtido com mudanças mínimas no código, dando uma grande vantagem para essas duas árvores.

Com isso em mente, para um ambiente onde uma nutricionista acompanha diversos pacientes em dieta, como descrito, é altamente desencorajado utilizar ambas as árvores Splay ou ABP, pois elas irão causar o maior número de comparações a serem realizadas.

Mesmo que um programador utilize a ABP com um conjunto de dados bem maior, a potência dos computadores atuais concebivelmente irá acabar a análise mais rápido do que o ser humano possa clicar no botão de finalizar a tarefa. Isso se baseia na noção que, a comparação das quatro árvores ao mesmo tempo, com a inclusão da árvore Splay, a qual é muito ineficaz nesse projeto, sempre ocasionou em tempos muito pequenos de execução.

	Sorted	Shuffled
Comparações	3629	6370

Tabela 4 - Conjunto de dados obtidos a partir da implementação de uma LSE nas entradas analisadas anteriormente.

Pode-se notar pela Tabela 4 que apesar de as árvores divergirem entre si no desempenho, todas elas ainda se fazem mais rápidas e eficientes do que se fosse utilizada uma Lista Simplesmente Encadeada para realizar a mesma tarefa de ordenamento e busca. A maior diferença se dá nos dados embaralhados, em que a LSE faz um trabalho 44 vezes maior do que o pior dos casos nas árvores.

Nenhum dos resultados obtidos durante esse relatório foram surpreendentes, tendo em vista os diferentes propósitos pelos quais essas 4 implementações de árvores foram criadas. A ABP tem o foco em inserções muito rápidas mas com uma consulta mais lenta, enquanto a AVL sacrifica parte da performance de inserção para potencialmente ganhar muita eficiente durante a consulta, já a árvore RN existe como um meio termo entre essas, realizando uma inserção bem mais rápida que a AVL e ao mesmo tempo não tendo uma consulta tão lenta quando a ABP. A árvore Splay por outro lado foi criada para tarefas de “popularidade”, onde poucos nodos muito populares são acessados múltiplas vezes, algo que não se aplica bem ao ambiente criado para esse relatório.