Árvores para Análise Textual

Nayara Ribeiro Freire[1], Rafael Vinícius de Oliveira[1]

[1]Ciência da Computação – Instituto Federal do Sul de Minas Gerais (IFSULDEMINAS)
Caixa Postal 37903-070 – Passos – MG – Brazil

{nayara.freire,rafael.vinicius}@alunos.ifsuldeminas.edu.br

Resumo. *Esse projeto tem por objetivo realizar um estudo de caso sobre a estrutura de dados árvore binária de pesquisa, utilizando seus modelos com balanceamento (AVL) e sem balanceamento. Além da implementação de um vetor dinâmico para armazenar palavras lidas de um documento de texto. Os algoritmos foram testados em tempo e custo computacional. Para implementação foi utilizado a linguagem de programação C.*

1. Introdução

Esse trabalho busca realizar um estudo de caso sobre estruturas de dados de árvores e vetores dinamicamente alocados. Contextualizando o que são árvores binárias, o que é pesquisa binária. Com a apresentação dos métodos utilizados, funções implementadas e resultados obtidos com esse projeto. Ao final são relatados as dificuldades encontradas. O trabalho foi desenvolvido com a linguagem C, as funções podem ser calculadas com tempo e custo computacional.

2. Fundamentação Teórica

2.1. Pesquisa Binária

Xavier (2018) em seu livro aponta que pesquisa binária pode ser definida como, um sistema que identifica a posição central de um vetor, ou a mais próxima do centro. Verifica se o valor armazenado na posição do meio é maior ou menor que a chave utilizada como referência para a busca. Divide o vetor do lado que foi apontado como o detentor da chave, e verifica novamente a posição do central, recursivamente, até encontrar a chave.

2.2. Árvore Binária

De acordo com Ribeiro (201?), uma árvore binária é uma estrutura de dados organizada, em que possui um elemento raiz que determina a posição dos demais. Os menores ficam à esquerda da raiz enquanto os maiores ficam à direita, formando diversas subárvores, que podem ser utilizadas para uma pesquisa binária.

2.3. Árvore AVL

Segundo Galhardo *et. al* (2020), a árvore AVL pode ser considerada como uma árvore binária de busca balanceada, em que a diferença entre as subárvores direita e esquerda não podem ultrapassar 1 de tamanho. Com o balanceamento é garantido ao menos o tempo de execução $O\ (log\ n)$.

3. Material e Métodos

3.1 Hardware e Software Utilizados

Para a implementação das árvores o hardware utilizado é composto por: AMD Ryzen 5 3500U 2.10GHz, 12GB de Memória RAM DD4 dual channel 2400MHz, AMD Vega 8 Graphics 2GB com sistema operacional Windows 10 Home 20h2 (19042.1288). Para a implementação e execução dos algoritmos foi utilizado a linguagem C e o software Dev C++.

4. Experimentos Computacionais

A implementação se deu por partes, a primeira foi realizar a leitura do arquivo de texto na *main( )*, para isso foi utilizado o *fopen* com o método "r", método de leitura. Em seguida é implementado a função que inicializa o ponteiro, além da verificação se o arquivo que está sendo lido existe. O *while* percorre o arquivo lendo as informações, adicionando no contador à quantidade de palavras que há naquele documento, além de chamar a função letraMinuscula que tem dois objetivos, retirar os caracteres especiais e remover letras maiúsculas.

```
arq = fopen("texto.txt", "r");
int i=0, j = 0, qtdPalavras=0;
inicializa(&ponteiro);

if(arq == NULL){
      printf("Can't open file for reading.\n");
}
else{
      while (fscanf(arq, "%s", word) != EOF)        {
         qtdPalavras++;
         char *palavraCorrigida = letraMinuscula(word, strlen(word));
         insere(&ponteiro, palavraCorrigida);
      }
      fclose(arq);
}
```

A seguir é apresentado o código da função letraMinuscula, como dito anteriormente ela tem por função remover os caracteres especiais por exemplo pontuações, isso ocorre por meio do comando condicional *if* e um laço de repetição *for* passando a função *tolower* nas palavras. Função essa que realiza a transformação para minúscula. Após a verificação retorna a palavra para ser inserida na árvore.

```
//Função letraMinuscula remove os caracteres especiais e remove letras maiusculas
char* letraMinuscula(char *palavra, int tamanho){
         int i, j;
         char *aux = malloc(tamanho);
```

```
        if(strchr(palavra, '.') != NULL ||
                strchr(palavra, ',') != NULL ||
                strchr(palavra, '!') != NULL ||
                strchr(palavra, '?') != NULL ||
                strchr(palavra, ':') != NULL ||
                strchr(palavra, ';') != NULL ||
                strchr(palavra, '%') != NULL ||
                strchr(palavra, '&') != NULL ||
                strchr(palavra, '*') != NULL ||
                strchr(palavra, '$') != NULL ||
                strchr(palavra, '#') != NULL ||
                strchr(palavra, '@') != NULL){

                for(i=0; i<tamanho-1; i++){
                        aux[i] = tolower(palavra[i]);
                }

                printf("%s\n", aux);
                return aux;
        }
        else{
                for(i=0;i<tamanho;i++){
                        palavra[i] = tolower(palavra[i]);
                }
                return palavra;
        }
}
```

Após o retorno da palavra tratada, a função insere é chamada para que possa ser inserida a palavra na árvore. Nessa função é verificado a existência de palavras duplicadas, assim é possível identificar quais palavras mais se repetem, caso as palavras não tenham sido tratadas serão consideradas diferentes.

```
//Função insere para cadastrar os dados na árvore
void insere(TNo **ptr, char palavra[]){
        int inserida = contQtdRepeticao(*ptr, palavra);
        int i, j;

        if(inserida == 0){
                if(*ptr == NULL){
                        (*ptr) = (TNo *) malloc(sizeof(TNo));
                        (*ptr)->esq = NULL;
                        (*ptr)->dir = NULL;
                        strcpy((*ptr)->palavra, palavra);
                        (*ptr)->qtd = 1;
                }
                else{
                        if(strcmp(palavra, (*ptr)->palavra) < 0){
```

```
                        insere(&(*ptr)->esq, palavra);
                }
                else if(strcmp(palavra, (*ptr)->palavra) > 0){
                        insere(&(*ptr)->dir, palavra);
                }
                else{
                        printf("igual");
                }
        }
    }
}
```

Outra função implementada foi o *inordem* no qual apresenta os dados em ordem alfabética ficando visualmente mais fácil a interpretação.

```
//Função inOrdem para apresentação dos dados em ordem alfabética, além da quantidade de repetições
void inOrdem(TNo *ptr){
    if(ptr != NULL){
        inOrdem(ptr->esq);
        printf("\n**********************\n");
        printf("palavra: %s\n", ptr->palavra);
        printf("qtd: %d\n", ptr->qtd);
        inOrdem(ptr->dir);
    }
    contPalavra();
}
```

A função a seguir tem por objetivo apresentar a quantidade de palavras que se repetem, e quantas vezes as mesmas aparecem no vetor. Para isso foi utilizado comandos de condicionais *if* e *else if*. Ainda é utilizado recursão pois cada palavra é analisada individualmente.

```
void contQtdRepeticao(TNo *ptr, char palavra[]){
    if(ptr != NULL){
        if(strcmp(palavra, ptr->palavra) > 0){
            contQtdRepeticao(ptr->dir, palavra);
            //contPalavra();
        }
        else if(strcmp(palavra, ptr->palavra) < 0){
            contQtdRepeticao(ptr->esq, palavra);
            //contPalavra();
        }
        else if(strcmp(palavra, ptr->palavra) == 0){
            repeticoes++;
            ptr->qtd++;
            return 1;
```

```
                }
        }
        return 0;
}
```

O tempo e o custo computacional não foram calculados, todavia para realização dessas tarefas seriam utilizadas as seguintes funções:

```
//Função para calcular o tempo em milisegundos
struct timeval comeco, final;
gettimeofday(&comeco, NULL);
gettimeofday(&final, NULL);
printf("Tempo de processamento do vetor: %ld microsegundos\n\n", ((final.tv_sec - comeco.tv_sec)*1000000L+final.tv_usec) - comeco.tv_usec);
```

A função que seria utilizada para a busca binária seria composta pela função a seguir. Como houve problemas ao passar as informações da árvore para o vetor dinâmico, ela acaba funcionando para exemplos específicos.

```
//Função que seria utilizada para busca binária
void pesquisaBinaria(char **vetor, char chave){
        int direita = naoRepetidas - 1;
        int esquerda = 0;
        int encontrado = 0;
        int meio;
        while(esquerda <= direita && !encontrado){
                meio = (esquerda + direita)/2;
                if(strcmp((*vetor+meio), chave) == 0){
                        printf("Encontrado");
                }
                else if(chave < (*vetor+meio)){
                        direita = meio - 1;
                }
                else{
                        esquerda = meio + 1;
                }
        }
}
```

O problema encontrado se deve a função de inserção dos dados da árvore no vetor dinâmico, a estratégia abordada era copiar cada palavra da árvore ou seja cada nó assumiria uma posição no vetor. Para isso tentamos utilizar a seguinte função

```
//Função que seria utilizada para inserir os dados no vetor dinamicamente alocado
void insereVetor(TNo *ptr, char **vetor){
```

```
        if(ptr != NULL){
                char *word = malloc(naoRepetidas * sizeof(char));
                strcpy(word, ptr->palavra);
                insereVetor(ptr->esq, vetor);
                *(vetor+i) = word;
                free(word);
                i++;
                insereVetor(ptr->dir, vetor);
        }
}
```

Por fim para realizar a comparação da quantidade de comparações que foram realizadas, seria utilizado um contador que a cada iteração somaria um valor e seria apresentado no final.

5. Dificuldades Encontradas

A maior dificuldade encontrada foi a inserção dos dados da árvore em um vetor dinâmico para a realização da busca binária como dito anteriormente. A função consegue identificar qual elemento deve ser inserido e qual a posição que deve ser. Mas no momento de inserção não aponta para o mesmo local, isso foi identificado utilizando o *Replit* para debugar, tendo em vista que o Dev C++ "crashava" quando tentávamos debugar por ele. Como não foi possível corrigir esse problema, também não foi possível realizar a implementação das demais funções.

6. Referências

RIBEIRO, Alexandre. Um Objeto de Aprendizagem para o ensino de Árvores Binárias. 201?. Disponível em: cepein.femanet.com.br/BDigital/arqPIBIC/1211330130B503.pdf. Acesso em: 06 dez. 2021.

UMA COMPARAÇÃO DO CÁLCULO DA MEDIANA DE INTEIROS CONTIDOS EM ÁRVORES AVL E RUBRO-NEGRAS. Palmas: Sítio Novo, v. 4, n. 3, jul. 2020. Trimestral. Disponível em: sitionovo.ifto.edu.br/index.php/sitionovo/article/view/582. Acesso em: 06 dez. 2021.

XAVIER, Gley Fabiano Cardoso. Lógica de Programação. 13. ed. São Paulo: Senac Sp, 2018. 322 p.