Universidade Federal de Uberlândia

Faculdade de Computação

Algoritmos e Estruturas de Dados II

Índice Remissivo

Código: https://replit.com/@belli0099/IndiceRemissivo#main.c

Repositório: https://github.com/IsaTedeschi/AED-Indice_Remissivo

Alunos: Isabelli Prudêncio Tedeschi **Matrícula**: 12011BCC018

Fábio José de Oliveira e Silva 12011BCC044

1. DESCRIÇÃO DO PROBLEMA

1.1 Descrição do problema geral

O objetivo do trabalho é criar um índice remisso, que lista os termos e tópicos que são abordados num documento, em ordem alfabética, juntamente com as páginas em que aparecem no livro. Além de apresentar as informações sobre a quantidade de palavras distintas, o total de palavras e o tempo que foi gasto para construir o programa.

1.2 Descrição do problema em detalhes

Para criar o índice remissivo, primeiramente lemos o arquivo texto, em que está contido o texto a ser organizado. Após a leitura do mesmo são desconsideradas diferenças entre letras maiúsculas e minúsculas, e se a palavra aparece mais de uma vez numa determinada linha, esta linha aparece apenas uma vez no índice.

É importante destacar que durante a leitura das palavras do texto, é contabilizado tanto a quantidade de palavras existentes quanto o número de termos distintos.

As palavras juntamente com as linhas em que aparecem são colocados num vetor e, um por um, esses índices são inseridos na árvore AVL, onde os índices são organizados por ordem alfabética.

Uma vez que os índices estão organizados em ordem alfabética e possuem as informações de quais linhas pertencem, a árvore é impressa em um outro arquivo texto, simultaneamente com as outras informações pedidas, ou seja, o número total de palavras, o número de palavras distintas e o tempo que demorou para ser executado esse processo.

E assim o programa se finaliza.

2. DESCRIÇÃO DAS ESTRUTURAS UTILIZADAS

2.1 Estrutura para guardar os termos do índice

A estrutura utilizada é identificada como Indice. Ela contém três informações, isto é, a palavra do texto (*char palavra*), um vetor com as linhas em que o termo aparece (*int linha[100]*) e o índice do vetor que a última informação do vetor linhas está, é uma variável de controle, para saber onde pode ser adicionado a próxima informação do vetor linhas (*int tam*).

```
struct Indice{
    char palavra[100];
    int tam;
    int linhas[100];
};
typedef struct Indice Indice;
```

2.2 Estrutura da árvore AVL

A estrutura de cada nó da árvore é caracterizada por seu valor, que é do tipo Índice, este é, um termo do arquivo texto e as linhas que ele aparece (*Indice info*); a altura de cada nó (*int altura*); uma estrutura que liga o nó ao nó da esquerda (*struct NO *esq*) e uma estrutura que liga ao nó da direita (*struct NO *dir*).

```
struct NO{
    Indice info;
    int altura;
    struct NO *esq;
    struct NO *dir;
};
```

3. CÓDIGO DAS PRINCIPAIS FUNÇÕES

3.1 Código do “main.c” - Leitura do arquivo texto

A leitura do arquivo é feita lendo cada palavra, linha por linha. Para cada palavra lida, primeiro ela é convertida para letras minúsculas, já que não tem distinção de mesma palavra com letras maiúsculas e minúsculas (essa mudança é feita na função *minúsculas()*).

Após a conversão, é feita uma verificação no vetor de palavras distintas, para analisar se ela já existe ou não. Caso ela não exista, ela é adicionada ao vetor e o número de palavras distintas aumenta, além de ser adicionado também a linha em que ela está. Porém, caso ela já exista, há dois casos, o termo ter aparecido duas vezes na mesma linha e o tempo existir mais de uma vez ao longo das linhas. Na primeira situação, o número de linhas não se repete, então nada é feito; se for a segunda situação, a palavra já existe, então só é adicionada a linha que ela está contida no vetor de linhas. Em qualquer um desses casos o número de palavras lidas aumenta.

```
while (fscanf(file, "%[^\n] ", linha) != EOF){
        novaPalavra = strtok(linha, " ,.;!?/");

        while (novaPalavra != NULL) {
            minusculo(novaPalavra, palavra);
            strcpy(aux.palavra, palavra);
            aux.tam = 0;
            int cont = 0;

            for(i=0;i<numDistintas;i++){
                if(strcmp(aux.palavra, palavrasDist[i].palavra)==0){
                    if(palavrasDist[i].tam != numLinhas){
                        palavrasDist[i].linhas[palavrasDist[i].tam] = numLinhas;
                        palavrasDist[i].tam++;
                    }
                    cont++;
                    break;
                }
            }
            if(cont == 0){
                aux.linhas[aux.tam] = numLinhas;

                palavrasDist[numDistintas] = aux;
                palavrasDist[numDistintas].tam++;
                numDistintas++;
            }

            numTotal++;
            novaPalavra = strtok(NULL," ,.;!?/");
        }


        numLinhas++;
}
```

3.2 Código da “arvore.c”

3.2.1 Inserção de elementos

Para inserir um novo valor v na árvore, devem ser considerados os casos: se a raiz é igual a NULL: criar o nó e inserir v; se v é menor do que a raiz, deve-se caminhar para a sub-árvore esquerda e reiniciar a inserção; se v é maior que a raiz: caminhar para a sub-árvore direita e reiniciar a inserção.

A aplicação recursiva do método permite alcançar uma posição nula na árvore, local em que o novo valor v será inserido.

Após a inserção de um novo valor na árvore AVL, deve-se: retroceder no caminho da inserção, verificando o fator de balanceamento de cada um dos nós visitados; executar, a depender do fator de balanceamento obtido para um nó (+2 ou -2), a rotação adequada para a reorganização da sua sub-árvore.

```
int insere_ArvAVL(ArvAVL *raiz, Indice valor){
    int r;
    if(*raiz == NULL){//árvore vazia ou nó folha
        struct NO *novo;
        novo = (struct NO*)malloc(sizeof(struct NO));
        if(novo == NULL)
            return 0;

        novo->info = valor;
        novo->altura = 0;
        novo->esq = NULL;
        novo->dir = NULL;
        *raiz = novo;
        return 1;
    }

    struct NO *atual = *raiz;
    if(strcmp(valor.palavra, atual->info.palavra)<0){
        r = insere_ArvAVL(&(atual->esq), valor);
        if(r == 1){
            if(fatorBalanceamento_NO(atual) >= 2){
                if(strcmp(valor.palavra, (*raiz)->esq->info.palavra)<0 ){
                    RotacaoDireita(raiz);
                }else{
                    RotacaoDuplaDireita(raiz);
                }
            }
        }
  }
```

```
else{
        if(strcmp(valor.palavra, atual->info.palavra)>0){
            r = insere_ArvAVL(&(atual->dir), valor);
            if(r == 1){
                if(fatorBalanceamento_NO(atual) >= 2){
                    if(strcmp((*raiz)->dir->info.palavra, valor.palavra)<0){
                        RotacaoEsquerda(raiz);
                    }else{
                        RotacaoDuplaEsquerda(raiz);
                    }
                }
            }
        }else{
            printf("Valor duplicado!!\n");
            return 0;
        }
    }

    atual->altura = maior(altura_NO(atual->esq),altura_NO(atual->dir)) + 1;

    return r;
}
```

3.2.2 Rotação

A rotação é uma forma de organizar a árvore para que ela fique balanceada.

Uma operação de rotação altera o balanceamento de uma árvore binária, garantindo a propriedade AVL e a sequência de percurso em ordem. Podem-se definir 4 tipos diferentes de rotação: rotação à esquerda, rotação à direita, rotação dupla à esquerda e rotação dupla à direita

```
void RotacaoDireita(ArvAVL *A){//LL
    struct NO *B;
    B = (*A)->esq;
    (*A)->esq = B->dir;
    B->dir = *A;
    (*A)->altura = maior(altura_NO((*A)->esq),altura_NO((*A)->dir)) + 1;
    B->altura = maior(altura_NO(B->esq),(*A)->altura) + 1;
    *A = B;
}

void RotacaoEsquerda(ArvAVL *A){//RR
    struct NO *B;
    B = (*A)->dir;
    (*A)->dir = B->esq;
    B->esq = (*A);
    (*A)->altura = maior(altura_NO((*A)->esq),altura_NO((*A)->dir)) + 1;
    B->altura = maior(altura_NO(B->dir),(*A)->altura) + 1;
    (*A) = B;
}

void RotacaoDuplaDireita(ArvAVL *A){//LR
    RotacaoEsquerda(&(*A)->esq);
    RotacaoDireita(A);
}

void RotacaoDuplaEsquerda(ArvAVL *A){//RL
```

3.2.3 Impressão da árvore

A impressão da árvore é feita de duas maneiras. A primeira é dentro do próprio programa, apresentando os resultados esperados, isto é, os termos com as linhas em que aparecem, o número total de palavras e o número de palavras distintas, além do tempo de execução do problema (função: *escreveIndiceConsole*). Na segunda forma, é criado um arquivo texto em que as informações pedidas são escritas e apresentadas (função: *escreveIndice*).

```
void escreveIndiceConsole(ArvAVL *raiz){
    if(raiz == NULL)
        return;
    if(*raiz != NULL){
        escreveIndiceConsole(&((*raiz)->esq));
        printf("\n%s: ",(*raiz)->info.palavra);
        for(int k=0;k<(*raiz)->info.tam;k++)
            printf("%d ", (*raiz)->info.linhas[k]);
        escreveIndiceConsole(&((*raiz)->dir));
    }
}

void escreveIndice(ArvAVL *raiz, FILE *file){
    if(raiz == NULL)
        return;
    if(*raiz != NULL){
        escreveIndice(&((*raiz)->esq), file);
        fprintf(file, "\n%s: ",(*raiz)->info.palavra);
        for(int k=0;k<(*raiz)->info.tam;k++)
            fprintf(file, "%d ", (*raiz)->info.linhas[k]);
        escreveIndice(&((*raiz)->dir), file);
    }
}
```

4. Exemplos rodados

4.1 Exemplo 1 – “O bom pirata”

```
Hoje eu cansei de ser bonzinho
hoje eu cansei da vida ingrata
e decidi me transformar
de herói a um bom pirata
Construí sozinho o meu navio
atraindo a todos os olhares
e tão só fui navegar
como autodidata dos mares
Usei todos os meus canhões
numa artilharia quase sem fim
pra derrubar sua fortaleza
e roubar você pra mim
Icei depressa as minhas velas
fugi das outras embarcações
e o infeliz que só te maltrata
eu joguei aos tubarões
Desenhei meu próprio mapa
só com a minha intuição
pro mais belo dos tesouros
a trilha para o seu coração
Ancorei na ilha deserta
pro início de uma eternidade
e brindarmos nossa união
com o rum da felicidade
```

```
Índice:

a: 4 6 18 20
ancorei: 21
aos: 16
artilharia: 10
as: 13
atraindo: 6
autodidata: 8
belo: 19
bom: 4
bonzinho: 1
brindarmos: 23
canhões: 9
cansei: 1 2
com: 18 24
como: 8
construí: 5
coração: 20
da: 2 24
das: 14
de: 1 4 22
decidi: 3
depressa: 13
derrubar: 11
desenhei: 17
deserta: 21
dos: 8 19
e: 3 7 12 15 23
embarcações: 14
eternidade: 22
eu: 1 2 16
felicidade: 24
fim: 10
fortaleza: 11
fugi: 14
fui: 7
herói: 4
hoje: 1 2
icei: 13
ilha: 21
infeliz: 15
```

```
infeliz: 15
ingrata: 2
intuição: 18
início: 22
joguei: 16
mais: 19
maltrata: 15
mapa: 17
mares: 8
me: 3
meu: 5 17
meus: 9
mim: 12
minha: 18
minhas: 13
na: 21
navegar: 7
navio: 5
nossa: 23
numa: 10
o: 5 15 20 24
olhares: 6
os: 6 9
outras: 14
para: 20
pirata: 4
pra: 11 12
pro: 19 22
próprio: 17
quase: 10
que: 15
roubar: 12
rum: 24
sem: 10
ser: 1
seu: 20
sozinho: 5
sua: 11
só: 7 15 18
te: 15
```

```
tesouros: 19
todos: 6 9
transformar: 3
trilha: 20
tubarões: 16
tão: 7
um: 4
uma: 22
união: 23
usei: 9
velas: 13
vida: 2
você: 12
```

```
Número total de palavras: 92
Número de palavras distintas: 118
Tempo de construção do índice usando árvore AVL: 0,007s
```

4.2 Exemplo 2 – “O Que o Sol Faz Com as Flores”

```
você liga e diz que sente a minha falta
encaro a porta da frente de casa
esperando uma batida
dias depois você liga e diz que precisa de mim
mas você não veio
no jardim cada dente-de-leão
revira os olhos de decepção
a grama decidiu que você é notícia velha
de que me importa
se você me ama
e sente minha falta
e precisa da minha presença
se não faz absolutamente nada
se eu não sou o amor da sua vida
com certeza serei a grande perda
```

```
Índice:

a: 1 2 8 15
absolutamente: 13
ama: 10
amor: 14
batida: 3
cada: 6
casa: 2
certeza: 15
com: 15
da: 2 12 14
de: 2 4 7 9
decepção: 7
decidiu: 8
dente-de-leão: 6
depois: 4
dias: 4
diz: 1 4
e: 1 4 11 12
encaro: 2
esperando: 3
eu: 14
falta: 1 11
faz: 13
frente: 2
grama: 8
grande: 15
importa: 9
jardim: 6
liga: 1 4
mas: 5
me: 9 10
mim: 4
minha: 1 11 12
nada: 13
no: 6
notícia: 8
não: 5 13 14
o: 14
olhos: 7
os: 7
```

```
o: 14
olhos: 7
os: 7
perda: 15
porta: 2
precisa: 4 12
presença: 12
que: 1 4 8 9
revira: 7
se: 10 13 14
sente: 1 11
serei: 15
sou: 14
sua: 14
uma: 3
veio: 5
velha: 8
vida: 14
você: 1 4 5 8 10
é: 8
```

```
Número total de palavras: 57
Número de palavras distintas: 87
Tempo de construção do índice usando árvore AVL: 0,004s
```

4.2 Exemplo 3 – “Amor é um fogo que arde sem se ver”

```
Amor é fogo que arde sem se ver;
É ferida que dói, e não se sente;
É um contentamento descontente;
É dor que desatina sem doer.

É um não querer mais que bem querer;
É um andar solitário entre a gente;
É nunca contentar-se de contente;
É um cuidar que se ganha em se perder.

É querer estar preso por vontade;
É servir a quem vence, o vencedor;
É ter com quem nos mata, lealdade.

Mas como causar pode seu favor
Nos corações humanos amizade,
Se tão contrário a si é o mesmo Amor?
```

```
Índice:

a: 6 10 14
amizade: 13
amor: 1 14
andar: 6
arde: 1
bem: 5
causar: 12
com: 11
como: 12
contentamento: 3
contentar-se: 7
contente: 7
contrário: 14
corações: 13
cuidar: 8
de: 7
desatina: 4
descontente: 3
doer: 4
dor: 4
dói: 2
e: 2
em: 8
entre: 6
estar: 9
favor: 12
ferida: 2
fogo: 1
ganha: 8
gente: 6
humanos: 13
lealdade: 11
mais: 5
mas: 12
mata: 11
mesmo: 14
nos: 11 13
nunca: 7
não: 2 5
o: 10 14
```

```
o: 10 14
perder: 8
pode: 12
por: 9
preso: 9
que: 1 2 4 5 8
quem: 10 11
querer: 5 5 9
se: 1 2 8 8 14
sem: 1 4
sente: 2
servir: 10
seu: 12
si: 14
solitário: 6
ter: 11
tão: 14
um: 3 5 6 8
vence: 10
vencedor: 10
ver: 1
vontade: 9
é: 1 2 3 4 5 6 7 8 9 10 11 14
```

```
Número total de palavras: 62
Número de palavras distintas: 94
Tempo de construção do índice usando árvore AVL: 0,009s
```