Centro Tecnológico

Departamento de Informática

Estrutura de Dados I

Trabalho Prático II

Indexador de Arquivos em linguagem C

Alunos: Ezequiel Schneider Reinholtz

Alunos: Henrique Coutinho Layber

Professor: Vinícius F. Soares Mota

Vitória, ES

10/07/2019

1 Introdução

Esse trabalho tem como objetivo construir e testar um indexador de arquivos feito unicamente na linguagem C. O indexador foi implementado em cinco estruturas de dados diferentes, são essas: Lista encadeada (Capitulo), Arvore Binária (Capitulo), Arvore AVL (Capitulo), Tabela Hash (Capitulo) e Arvore Trie (Capitulo), a fim de calcular o tempo de inserção e o tempo de busca de uma palavra em cada estrutura e estabelecer uma comparação entre as mesmas.

Os testes de tempo de busca e comparações estão descritos no Capítulo (falta) a implementação de escolha de uma palavra aleatória dentro de um arquivo está descrita no Capítulo (falta). O tempos de inserção e tempo de busca são calculados separadamente e não incluem o tempo gasto para a criação, impressão ou remoção da memória das estruturas. Exemplos de como usar referência: **??**) (in-line) ou (**??**).

2 Implementação

2.1 Lista Encadeada

2.1.1 Estrutura

A estrutura lista encadeada foi organizada da seguinte forma:

Listagem 1 – Struct da lista encadeada.

```
1  typedef struct Lista tCelula;
2  typedef struct Lista {
3      tPalavra *palavra;
4      tCelula *prox;
5  } tCelula;
6
7  typedef struct ListaSent{
8      int qtd;
9      tCelula *ini, *fim;
10 } tLista;
```

Na estrutura de `tCelula` temos um ponteiro de outra estrutura `tPalavra` para guardar todas as informações referentes a palavra inserida e um ponteiro `tCelula` para indicar a próxima célula na lista encadeada.

Na estrutura de `tLista` temos uma variavel `int` para guardar o tamanho da lista encadeada e dois ponteiros `tCelula`, um para indicar a célula inicial e outro para indicar a célula final.

2.1.2 Inserção

A implementação da inserção na lista encadeada foi feita da seguinte forma:

Listagem 2 – Implementação do código de inserção da lista encadeada.

```
1  char insere_Lista(tLista *l, char *str, int byte, char arq){
2      if(str == NULL || l == NULL) return 0;
3      //Procura a palavra na lista
4      for(tCelula *aux = l->ini; aux != NULL; aux = aux->prox)
5          if(strlen(aux->palavra->pal) == strlen(str))
6              if(strcasecmp(aux->palavra->pal, str) == 0){
7                  //palavra encontrada na lista
8                  adiciona_IndicePal(aux->palavra, byte, arq);
9                  return 1;
10             }
11     //Não foi encontrado a palavra na lista
12     //Será criada um novo nó com a palavra
13     tCelula *no = novo_no_Lista(str, byte, arq);
14     no->prox = l->ini;
15     l->ini = no;
16     if(l->fim == NULL)
17         l->fim = no;
18     l->qtd++;
19     return 2;
20 }
```

2.1.2.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `tLista` 'l', um ponteiro `char` 'str', uma variavel `int` 'byte' e uma variavel `char` 'arq'.

2.1.2.2 Saídas

Quando sua saída for igual a 2 significa que a palavra foi inserida corretamente na lista. Quando igual a 1 a palavra já pertence a lista então não precisa ser reinserida. Quando igual a 0 é por que ou a palavra ou a lista é nula.

2.1.2.3 Funcionalidade

Após feito os testes de sanidade a função entra em um `for` em busca da palavra, para verificar se a mesma já não está inserida na lista encadeada, se por acaso ela já foi inserida então o número de ocorrências da palavra aumenta, e isso é feito na função `adiciona_IndicePal(tPalavra *pal, int byte, char arq)`. Caso não encontre a palavra a função aloca mais uma célula com as informações da palavra e inclui o nó no fim da lista encadeada.

2.1.3 Busca palavra

A implementação da inserção na lista encadeada foi feita da seguinte forma:

2.1.3.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `tLista` 'l' e um ponteiro `char` 'pal'.

2.1.3.2 Saídas

Quando sua saída for igual a 1 significa que a palavra foi encontrada. Quando igual a 0 significa que a palavra não foi encontrada na lista.

2.1.3.3 Funcionalidade

Após feito os testes de sanidade a função entra em um `for` em busca da palavra, se ela encontra uma palavra igual a da palavra da entrada dentro da lista a função retorna o valor 1. Caso contrário a palavra não foi encontrada e o retorno é 0.

2.2 Árvore Binária

2.2.1 Estrutura

A estrutura árvore binaria foi organizada da seguinte forma:

Listagem 3 – Struct da Árvore Binária.

```
1 typedef struct NO tNo;
2 typedef tNo* ArvBin;
3 typedef struct NO{
4     tPalavra *palavra;
5     ArvBin esq, dir;
6 } tNo;
```

Na estrutura de `tNO` temos um ponteiro de outra estrutura `tPalavra` para guardar todas as informações referentes a palavra inserida e um ponteiro `tNO` para indicar qual o nó a esquerda e qual o nó à direita do nó em questão.

Inclusão de um `typedef tNO* ArvBin` que facilita a interpretação do código.

2.2.2 Inserção

A implementação da inserção na Árvore Binária foi feita da seguinte forma:

2.2.2.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `ArvBin` 'raiz', um ponteiro `char` 'palavra', uma variavel `int` 'byte' e uma variavel `char` 'arq'.

2.2.2.2 Saídas

2.2.2.3 Funcionalidade

2.2.3 Busca palavra

A implementação da inserção na Árvore Binária foi feita da seguinte forma:

2.2.3.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `ArvBin` 'raiz' e um ponteiro `char` 'palavra'.

2.2.3.2 Saídas

Quando sua saída for igual a 1 significa que a palavra foi encontrada. Quando igual a 0 significa que a palavra não foi encontrada na lista.

Listagem 4 – Implementação do código de inserção da árvore binária.

```
char insere_ArvBin(ArvBin* raiz, char* palavra, int byte, char arq){
    if(raiz == NULL || palavra == NULL) return 0;

    ArvBin aux = *raiz, anterior = NULL;
    while(aux != NULL && !strings_Iguais(aux->palavra->pal, palavra)){
        anterior = aux;
        if(strcasecmp((*raiz)->palavra->pal, palavra) > 0)
            aux = aux->esq;
        else
            aux = aux->dir;
    }
    //encontrei ou lugar vazio ou no pra adicionar
    if(aux != NULL){        //lugar preenchido, adicionar indice
        if(!adiciona_IndicePal(aux->palavra, byte, arq))
        puts("Deu ruim ao adicionar ocorrencia na palavra");
    }
    else{        //lugar vazio
        aux = (ArvBin) malloc(sizeof(tNo));
        if(aux == NULL) return 0;
        aux->palavra = cria_Palavra(palavra, arq, byte);
        aux->dir = aux->esq = NULL;
        if(anterior != NULL){
            if(strcasecmp((*raiz)->palavra->pal, palavra) > 0)
                anterior->esq = aux;
            else
                anterior->dir = aux;
        }
        if(*raiz == NULL)
            *raiz = aux;
    }
    return 1;
}
```

Listagem 5 – Implementação do código de busca da árvore binária.

```
char consulta_ArvBin(ArvBin *raiz, char* palavra){
    if(raiz == NULL)
        return 0;
    struct NO* atual = *raiz;
    int tam_menor = SelecionaMenorString(palavra, (*raiz)->palavra->pal);
    int compara = strncmp(palavra, atual->palavra->pal, tam_menor) > 0;
    while(atual != NULL){
        if(strings_Iguais(palavra, atual->palavra->pal)){
            return 1;
        }
        if(compara > 0)
            atual = atual->dir;
        else
            atual = atual->esq;
    }
    return 0;
}
```

2.2.3.3 Funcionalidade

Após feito os testes de sanidade a função cria um auxiliar e passa a referencia da raiz para ele. Primeiro a função compara se a palavra do nó atual é a desejada, se for retorna 1 e a palavra foi encontrada, caso contrário para selecionar se a palavra está ao nó a esquerda ou a direita inicia-se uma sequencia de comparações. Essas comparações são guardadas na variável `compara`, se `compara` for maior que 0 significa que a palavra da entrada é maior em relação a palavra do nó atual e então encaminha a consulta ao nó a direita, caso contrário encaminha ao nó a esquerda. Se por ventura não encontrar a palavra em nenhum nó a função retorna 0.

2.3 Árvore AVL

2.3.1 Estrutura

A estrutura árvore AVL foi organizada da seguinte forma:

Listagem 6 – Struct da Árvore AVL.

```
1 typedef struct NO tNo;
2 typedef tNo* ArvAVL;
3 typedef struct NO{
4     int altura;
5     tPalavra *palavra;
6     ArvAVL esq, dir;
7 } tNo;
```

Na estrutura de `tNO` temos um ponteiro de outra estrutura `tPalavra` para guardar todas as informações referentes a palavra inserida, um ponteiro `tNO` para indicar qual o nó a esquerda e qual o nó à direita do nó em questão e uma variável `int` para guardar a altura do nó atual.

Inclusão de um `typedef tNO* ArvAVL` que facilita a interpretação do código.

2.3.2 Inserção

A implementação da inserção na Árvore AVL foi feita da seguinte forma:

2.3.2.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `ArvAVL` 'raiz', um ponteiro `char` 'palavra', uma variavel `int` 'byte' e uma variavel `char` 'arq'.

Listagem 7 – Implementação do código de inserção da árvore AVL.

```
1  char insere_ArvAVL(ArvAVL *raiz, char* palavra, int byte, char arq){
2      if(raiz == NULL) return 0;
3      /* Verifica se Arvore Vazia ou se eh NO folha */
4      if(*raiz == NULL){  //criando novo nó
5          ArvAVL novo = (ArvAVL) malloc(sizeof(tNo));
6          if(novo == NULL)
7              return 0;
8
9          novo->palavra = cria_Palavra(palavra, arq, byte);
10         novo->altura = 0;
11         novo->esq = novo->dir = NULL;
12         *raiz = novo;
13         return 1;
14     }
15
16     /**/
17     int tam_menor = SelecionaMenorString(palavra, (*raiz)->palavra->pal);
18     if(!tam_menor)  return 0;
19     int compara = strncmp(palavra, (*raiz)->palavra->pal, tam_menor) > 0;
20     ArvAVL atual = *raiz;
21
22     if(compara > 0){
23         if(insere_ArvAVL(&(atual->esq), palavra, byte, arq)){
24             if(fatorBalanceamento_NO(atual) >= 2){
25                     if(strncmp(palavra, (*raiz)->esq->palavra->pal, tam_menor) >
                           0){
26                         RotacaoLL(raiz);
27                     }else{
28                         RotacaoLR(raiz);
29                 }
30             }
31         }
32     }else{
33         if(compara < 0){
34             if(insere_ArvAVL(&(atual->esq), palavra, byte, arq)){
35                 if(fatorBalanceamento_NO(atual) >= 2){
36                     if(strncmp(palavra, (*raiz)->esq->palavra->pal, tam_menor) <
                           0){
37                         RotacaoRR(raiz);
38                     }else{
39                         RotacaoRL(raiz);
40                     }
41                 }
42             }
43         }else{
44             return 0;
45         }
46     }
47
48     atual->altura = maior(altura_NO(atual->esq),altura_NO(atual->dir)) + 1;
49
50     return 1;
51 }
```

2.3.2.2 Saídas

2.3.2.3 Funcionalidade

2.3.3 Busca palavra

A implementação da inserção na Árvore AVL foi feita da seguinte forma:

Listagem 8 – Implementação do código de busca da árvore AVL.

```
1  char consulta_ArvAVL(ArvAVL *raiz, char* palavra){
2      if(raiz == NULL)
3          return 0;
4      struct NO* atual = *raiz;
5      int tam_menor = SelecionaMenorString(palavra, (*raiz)->palavra->pal);
6      int compara = strncmp(palavra, atual->palavra->pal, tam_menor) > 0;
7      while(atual != NULL){
8          if(strings_Iguais(palavra, atual->palavra->pal)){
9              return 1;
10         }
11         if(compara > 0)
12             atual = atual->dir;
13         else
14             atual = atual->esq;
15     }
16     return 0;
17 }
```

2.3.3.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `ArvAVL` 'raiz' e um ponteiro `char` 'palavra'.

2.3.3.2 Saídas

Quando sua saída for igual a 1 significa que a palavra foi encontrada. Quando igual a 0 significa que a palavra não foi encontrada na lista.

2.3.3.3 Funcionalidade

Após feito os testes de sanidade a função cria um auxiliar e passa a referencia da raiz para ele. Primeiro a função compara se a palavra do nó atual é a desejada, se for retorna 1 e a palavra foi encontrada, caso contrário para selecionar se a palavra está ao nó a esquerda ou a direita inicia-se uma sequencia de comparações. Essas comparações são guardadas na variável `compara`, se `compara` for maior que 0 significa que a palavra da entrada é maior em relação a palavra do nó atual e então encaminha a consulta ao nó a direita, caso contrário encaminha ao nó a esquerda. Se por ventura não encontrar a palavra em nenhum nó a função retorna 0.

2.4 Tabela Hash

2.4.1 Estrutura

A estrutura tabela Hash foi organizada da seguinte forma:

Listagem 9 – Struct da tabela Hash.

```
typedef struct Hash{
    ArvAVL *hash[TAMDAHASH];
    int colisoes, qtd;
    int *pesos;
} tHash;
```

Na estrutura de `tHash` temos um ponteiro de outra estrutura `tNO` onde para cada posição da tabela temos um raiz para uma Árvore AVL e então as informações da palavra são guardadas de forma semelhante à implementação da Árvore AVL, um ponteiro `int` de pesos, e duas variáveis `int`, 'colisões' para guardar o número de colisões da tabela e 'qtd' para guardar a quantidade total de inseridos.

2.4.2 Inserção

A implementação da inserção na tabela Hash foi feita da seguinte forma:

Listagem 10 – Implementação do código de inserção da tabela Hash.

```
char insere_Hash(tHash *hashtable, char *palavra, int byte, char arq){
    if(hashtable == NULL || palavra == NULL) return 0;
    if(hashtable->pesos == NULL || hashtable->hash == NULL) return 0;


    int insercao = hash(hashtable->pesos, palavra);
    // printf("atual = %d\n", hashtable->hash[insercao] == NULL);
    if(insercao < 0) insercao *= -1;
    if(*hashtable->hash[insercao] != NULL) hashtable->colisoes++;
    insere_ArvAVL(hashtable->hash[insercao], palavra, byte, arq);
    hashtable->qtd++;
    return 1;
}
```

2.4.2.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `tHash` 'hashtable', um ponteiro `char` 'palavra', uma variavel `int` 'byte' e uma variavel `char` 'arq'.

2.4.2.2 Saídas

2.4.2.3 Funcionalidade

2.4.2.4 Função de Hash

A função que trata o caso de colisões da tabela Hash foi feita da seguinte forma:

Listagem 11 – Implementação do código da função de Hash.

```
1 int hash(int *pesos, char *str){
2     if(str == NULL || pesos == NULL) return -1;
3     int result = 0;
4     int len = strlen(str);
5     for(int i = 0; i < len; i++)
6         result += pesos[i] * str[i];
7     return result % TAMDAHASH;
8 }
```

2.4.3 Busca palavra

A implementação da inserção na tabela Hash foi feita da seguinte forma:

Listagem 12 – Implementação do código de busca da tabela Hash.

```
1 char consulta_Hash(tHash *hashtable, char *str){
2     if(hashtable == NULL || str == NULL) return 0;
3     if(hashtable->pesos == NULL || hashtable->hash == NULL) return 0;
4
5     // printf("Buscando por %s\n", str);
6
7     int indice = hash(hashtable->pesos, str);
8     if(indice < 0 ) indice *= -1;
9     return consulta_ArvAVL(hashtable->hash[indice], str);
10 }
```

2.4.3.1 Entradas

Essa função recebe como entrada um ponteiro do tipo `tHash` 'raiz' e um ponteiro `char` 'str'.

2.4.3.2 Saídas

Quando sua saída for igual a 1 significa que a palavra foi encontrada. Quando igual a 0 significa que a palavra não foi encontrada na lista.

2.4.3.3 Funcionalidade

Após feito os testes de sanidade a função cria um auxiliar e passa a referencia da raiz para ele. Primeiro a função compara se a palavra do nó atual é a desejada, se for retorna 1 e a palavra foi encontrada, caso contrário para selecionar se a palavra está ao nó a esquerda ou a direita inicia-se uma sequencia de comparações. Essas comparações são guardadas na variável `compara`, se `compara` for maior que 0 significa que a palavra da entrada é maior em relação a palavra do nó atual e então encaminha a consulta ao nó a direita, caso contrário encaminha ao nó a esquerda. Se por ventura não encontrar a palavra em nenhum nó a função retorna 0.

2.5 Árvore Trie

2.5.1 Inserção

O pacote `listings`, incluído neste template, permite a inclusão de listagens de código. A Listagem 13 mostra um exemplo de listagem com especificação da linguagem utilizada no código. O pacote `listings` reconhece algumas linguagens[1] e faz "coloração" de código (na verdade, usa **negrito** e não cores) de acordo com a linguagem. O parâmetro `float=htpb` incluído em ambos os exemplos impede que a listagem seja quebrada em diferentes páginas.

Importante notar que não se deve incluir TODO o código do seu trabalho. Inclua apenas trechos que julgue necessário que sejam discutidos no relatório.

Listagem 13 – Exemplo de código C especificando linguagem utilizada.

```
1 #include<stdio.h>
2 #include<string.h>
3 #include<stdlib.h>
4
5 #include "arvore.h"
6
7 struct arv
8 {
9   char info;
10  struct arv* esq;
11  struct arv* dir;
12 };
```

[1] Veja a lista de linguagens suportadas em <http://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings#Supported_languages>.

3 Análise e resultados

3.1 Metodologia

Descrever a metodologia dos testes, como variou o tamanho dos arquivos, quantos arquivos foram utilizados, descrição do computador em que foram feitos os testes.

3.2 Resultados

Analisar os tamanhos dos arquivos compactados a partir dos experimentos realizados. Utilizar tabelas e gráficos para ilustrar o desempenho da sua implementação.

Exemplo de utilização de tabelas. A Tabela 1 apresenta um cronograma de execução de um PG fictício.

Tabela 1 – Cronograma de Atividades do primeiro semestre.

Atividade	Janeiro/99	Fevereiro/99	Março/99	Abril/99	Maio/99	Junho/99
1	X	X	X	X	X	X
2			X	X		
3			X	X	X	X
4						X
5					X	X
6						
7						

A Figura 1 exemplifica o uso de uma figura gráfica no texto.

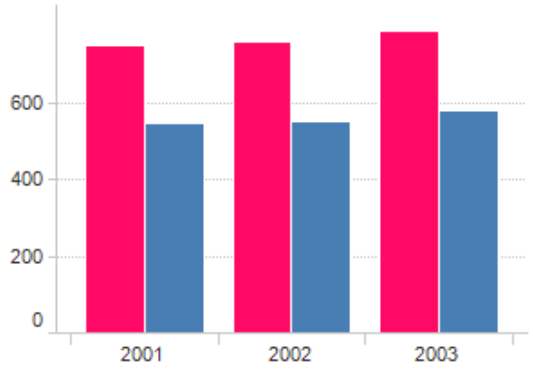

Figura 1 – Exemplo de inserção de figura

4 Conclusão

Discutir as conclusões e limitações do seu trabalho.