Instituto Federal Catarinense Campus Videira

Bacharelado em ciências da computação - 3ª Fase

Aluno: Wellington Eliel Santos de Oliveira

Relatório final: projeto de Estrutura de dados 1

Neste trabalho, fomos orientados a fazer um algoritmo em C que fosse capaz de importar os nomes de um arquivo txt e organizá-los em uma tabela Hash e depois organizá-los em ordem alfabética de uma maneira com que as colisões sejam evitadas.

A maneira de evitar colisões foi resolvida logo no planejamento do projeto, onde escolhemos inserir os elementos em uma lista encadeada dupla, e fazer o mesmo com a tabela hash, desta maneira não tem limite de itens para cada chave hash, logo não existirá nenhuma colisão nos “buckets” do hash.

Utilizei o número de chaves hash (“M”)de 53, que nos foi orientado, e também utilizei um vetor de 16 letras para cada nome, além de importar as bibliotecas necessárias.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define M 53
#define QTDCARACTERES 16
```

Na sequência do código implementei as Structs de nodo e lista tradicionais de uma lista encadeada dupla, com a alteração de um inteiro para posição e um inteiro para tabela, que será útil para fazermos a busca desse nodo, além de fazer a prototipação de todas as funções que vou utilizar neste algoritmo.

```c
typedef struct sNodo {
  char nome[QTDCARACTERES];
  int posicao;
  int tabela;
  struct sNodo *next;
  struct sNodo *prev;
} Nodo;

typedef struct sLista {
  struct sNodo *head;
  struct sNodo *tail;
  int size;
} Lista;

Nodo *alocaMemoriaNodo();
Lista *alocaMemoriaLista();
Nodo *criaNodo(char *);
Lista *criaLista();
int insereElementoNaLista(Lista *, Nodo *, char *, int, int);
void percorreListaHeadTail(Lista *, int);
int hash(char *nome);
int ascii(char letra);
Nodo *encontraElementoNaLista(Lista tabelaHash[], char *nome);
int removeElementoNaLista (Lista *, Nodo *);
void readArquivoTXT(Lista *tabelaHash);

int main() {
```

Na função main basicamente chamamos as funções que utilizamos para criar a tabela hash e para importar os nomes do arquivo, além de chamar as funções de printar os nomes e das funções de buscar e remover algum nome específico.

```
33
34  int main() {
35    Lista tabelaHash[M];
36    for (int i = 0; i < M; i++) {
37      tabelaHash[i] = *criaLista();
38    }
39    readArquivoTXT(tabelaHash);
40    for (int i = 0; i < M; i++) {
41      printf("Lista %d: ", i);
42      percorreListaHeadTail(&tabelaHash[i], i);
43      printf("\n");
44    }
45
46
47    Nodo *elemento= encontraElementoNaLista (tabelaHash, "JERMINIA");
48    if(elemento != NULL){
49      printf("\nO elemento %s está na lista, na tabela hash %i e na posição %i \n\n",
       elemento->nome, elemento->tabela, elemento->posicao);
50    } else{
51      printf("\nO elemento não está na lista\n");
52    }
53
54    for(int i=0; i<M; i++){
55      printf("A tabela hash %i tem %i elementos\n", i, tabelaHash[i].size);
56    }
57
58    Nodo *nodoremove = encontraElementoNaLista (tabelaHash, "JERMINIA");
59    if (nodoremove != NULL){
60        removeElementoNaLista (tabelaHash, nodoremove);
61      }
62
63    return 0;
64  }
```

A seguir a função para ler o arquivo e inserir na lista, também aproveito para direcionar a posição de cada nome dentro do nodo do mesmo.

```
65
66  void readArquivoTXT(Lista *tabelaHash) {
67    FILE *file = fopen("nomes.txt", "r");
68    char nome[QTDCARACTERES];
69    int posicao = 0;
70    while (fgets(nome, QTDCARACTERES, file) != NULL) {
71      nome[strcspn(nome, "\n")] = '\0';
72      int index = hash(nome);
73      insereElementoNaLista(&tabelaHash[index], tabelaHash[index].tail, nome, posicao,
    index);
74      posicao++;
75    }
76    fclose(file);
77  }
```

Na sequência temos a função hash, onde nos foi orientado utilizar o método ascii, que retorna um valor decimal para cada letra do nome, que vai ser utilizado pela função hash para definir em qual “Bucket” o nome será incluido.

```c
int ascii(char letra) {
  return letra;
}

int hash(char *nome) {
  int x = 0;
  for (int i = 0; i < strlen(nome); i++) {
    x = x + ascii(nome[i]);
  }
  x = x % M;
  return x;
}
```

Finalizando o código temos as funções de inserir, deletar, percorrer e pesquisar o nodo, que funcionam basicamente da mesma maneira de uma lista encadeada dupla, onde a principal diferença é na pesquisa, onde calculamos o valor hash do nome e buscamos diretamente pela tabela hash dele, assim facilitando e deixando a pesquisa mais rápida, além de algumas alterações na inserção para inserirmos a posição e a tabela hash no nodo e na impressão para imprimirmos essas duas variáveis.

```c
Nodo *encontraElementoNaLista(Lista tabelaHash[], char *nome) {
  int index = hash(nome);
  Nodo *nodo = tabelaHash[index].head;
```

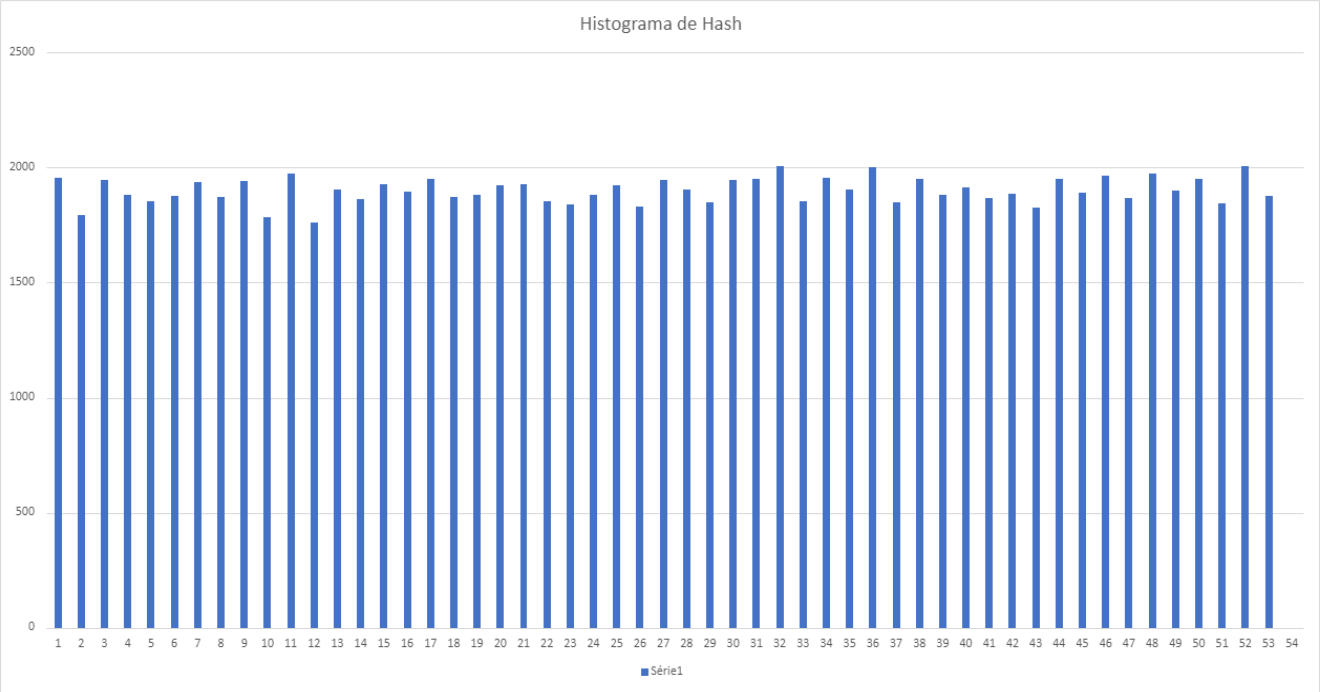
```c
  novo->posicao = posicao;
  novo->tabela = tabela;
  lista->size++;
  return lista->size;
}
```

```c

void percorreListaHeadTail(Lista *lista, int hash) {
  Nodo *nodo = lista->head;
  if (nodo != NULL) {
    while (nodo != NULL) {
      printf("Hash %d - Elemento %d do hash: %s\n", hash, nodo->posicao, nodo->nome);
      nodo = nodo->next;

    }
    printf("Fim da lista\n");
  } else {
    printf("Lista está vazia\n");
  }
}
```

Para finalizar, vamos analisar o histograma desta distribuição hash:



Como podemos analisar, a distribuição dos nomes está bem equilibrada, onde o menor valor é de 1761 nomes enquanto o maior valor é de 2008, uma variação de menos de 15%, obviamente não é um histograma perfeito, porém é um dos mais próximos da perfeição que podemos chegar