UNIVERSIDADE FEDERAL DE MATO GROSSO - UFMT
CAMPUS ARAGUAIA

MARIA LUISE BRITTO DA SILVA
TAINÁ ISABELA MONTEIRO DA SILVA

BARRA DO GARÇAS
2021

Maria Luise Britto da Silva
Tainá Isabela Monteiro Da Silva

Trabalho escrito apresentado como requisito parcial para obtenção de nota na disciplina de Estrutura de Dados II em Bacharel em Ciência da Computação do Universidade Federal de Mato Grosso - UFMT, Campus Araguaia.

Docente: Profº Dr. Robson Lopes

Barra do Garças
2021

SUMÁRIO

1 Introdução

O trabalho em questão foi composto por 4 etapas são elas:

- Etapa 1: Elaborar um algoritmo em linguagem C que gerasse um arquivo com um conjunto de valores e suas respectivas chaves, para serem inseridos em uma tabela Hash. Este algoritmo deve receber como parâmetro o número de elementos a ser gerado, o tipo das entradas e o nome do arquivo resultante. Como no Exemplo:

  ```
  ./algoritmo

  1000

  1

  saida.txt
  ```

  Onde algoritmo é o nome do programa, 1000 é o número de elementos a ser gerado, 1 é o tipo de arquivo e *saida.txt* o nome do arquivo resultante.

  No arquivo resultante, cada linha deve conter uma chave inteira com valor entre 0 e 1023, e um valor composto por 3 letras. Estes valores serão definidos aleatoriamente, porém é necessário que não existam chaves com valores repetidos.

  São dois tipos de arquivos de entrada possíveis:

  Tipo 1 - Todas as chaves geradas têm valor par.

  Tipo 2 - Sem restrição.

- Etapa 2:

  Implementar um algoritmo que receba um arquivo da etapa 1 e aplique em uma tabela Hash de tamanho 100 e que trate as colisões com encadeamento externo testando duas funções Hash distintas de tratamento de colisões.

- Etapa 3:

  Com o algoritmo da etapa 01:

  1. Gerar 5 arquivos do tipo 1 com 50 números gerados, com 100 números gerados e 150 números gerados. Um total de 15 arquivo do tipo 1;
  2. Gerar 5 arquivos do tipo 2 com 50 números gerados, com 100 números gerados e 150 números gerados. Um total de 15 arquivo do tipo 2;

Desta forma, ao final da etapa 3 teremos 30 arquivos.

Após isso executar os 30 arquivos no algorítimo da etapa 2 e para cada execução, contabilizar o número de colisões que ocorreram em cada uma das funções Hash.

- Etapa 4:

  Ao final, elaborar um relatório apresentando o trabalho, os algoritmos desenvolvidos, as funções Hash e os resultados obtidos. O arquivo deve conter : Capa, introdução, desenvolvimento, resultados e considerações finais. No qual é este atual relatório.

2 Desenvolvimento

Para a primeira etapa foi elaborado um algoritmo baseado nas aulas ministradas pelo docente, com as modificações de um vetor de *char* para a valor de 3 letras, além de uma função para randomizar o vetor de letras, as funções nativas para criação de um arquivo .txt:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
typedef struct aluno{
    int matricula;
    char cr[3];
    struct aluno *prox;
} TAluno;

typedef TAluno *Hash;

int hash(int mat, int tam){
    return mat % tam;
}

void inicializa(Hash *tab, int tam) {
    int i;
    for (i = 0; i < tam; i++) {
        tab[i] = NULL;
    }
}

TAluno *aloca(int mat, char cr[]){
    TAluno *novo = (TAluno *)malloc(sizeof(TAluno));
    novo->matricula = mat;
    novo->cr[0] = cr;
    novo->prox = NULL;
    return novo;
}

TAluno *busca(Hash *tab, int m, int mat){
    int h = hash(mat, m);
```

```
    TAluno *p = tab[h];
    while (p != NULL && p->matricula != mat){
        p = p->prox;
    }
    return p;
}

float excluir(Hash *tab, int m, int mat){
    int h = hash(mat, m);
    if (tab[h] == NULL){
        return -1;
    }
    TAluno *p = tab[h];
    TAluno *ant = NULL;
    float cr = -1;
    while (p != NULL && p->matricula != mat){
        ant = p;
        p = p->prox;
    }
    if (p == NULL){
        return -1;
    }
    if (ant == NULL){
        tab[h] = p->prox;
    }
    else{
        ant->prox = p->prox;
    }
    cr = p->cr[0];
    free(p);
    return cr;
}

void insere(Hash *tab, int m, int mat, char cr){
    int h = hash(mat, m);
    TAluno *p = tab[h];
    TAluno *ant = NULL;
    while (p != NULL && p->matricula != mat){
        ant = p;
```

```
            p = p->prox;
        }
        TAluno *novo = aloca(mat, cr);

        if (!ant){
            tab[h] = novo;
        }
        else{
            ant->prox = novo;
        }
    }

    void imprime(Hash *tab, int m){
        int i;
        for (i = 0; i < m; i++){
            printf("%d: ", i);
            if (tab[i]){
                TAluno *p = tab[i];
                while (p){
                printf("\t%d\t%c\t%p\t\n ", p->matricula, p->cr, p->prox);
                p = p->prox;
                }
            }
            else{
                printf("NULL\n");
            }
        }
    }

    void libera(Hash *tab, int m){
        int i;
        for (i = 0; i < m; i++)
        {
            if (tab[i]){
                TAluno *p = tab[i];
                TAluno *q;
                while (p){
                    q = p;
                    p = p->prox;
```

```
                free(q);
            }
        }
    }
}
char randomLetter(char vetor[]){
    char letra = 97 + (char)(rand() % 26);
    return letra;
}
int main(){
    srand(time(NULL));
    int m;
    scanf("%d", &m);
    Hash *tab[m];
    int tipo;
    scanf("%d", &tipo);
    char nome[100];
    scanf("%s", nome);
    strcat(nome, "");
    inicializa(tab, m);
    FILE *arq;
    arq = fopen(nome, "w");
    int i;
    int mat;
    char cr[3];
    for (i = 0; i < m; i++){
        if (tipo == 1){
            mat = rand() % 1023;
            if (mat % 2 == 0){
                mat = mat;
            }
            else{
                mat = mat + 1;
            }
            cr[0] = randomLetter(cr);
            cr[1] = randomLetter(cr);
            cr[2] = randomLetter(cr);
            insere(tab, m, mat, cr);
        }
```

```
        if (tipo == 2){
            mat = rand() % 1023;
            cr[0] = randomLetter(cr);
            cr[1] = randomLetter(cr);
            cr[2] = randomLetter(cr);
            insere(tab, m, mat, cr);
        }
        fprintf(arq, "%d\t%c%c%c\n", mat, cr[0], cr[1], cr[2]);
    }
    fclose(arq);
    libera(tab, m);
    return 0;
}
```

Para a segunda etapa, foi elaborado outros dois, um com as funções necessárias para tratamento de colisões afim de se contabilizar ao final essas colisões e chegar à um resultado e outro no qual seria o responsável por receber o arquivo, inseri-lo na tabela de tamanho 100, e mostrar o contador de colisões.

Algoritmo de inserção e contagem:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <time.h>
#include <sys/times.h>
#include <sys/types.h>
#include <unistd.h>
#include"funcoes.c"
int main(int argc, char *argv[]){
        clock_t tt;
        struct tms time;
    int tics_per_second;
    tics_per_second = sysconf(_SC_CLK_TCK);
    FILE *arquivo;
    int m = 100, chave;
        int colisao;
        char ltr[5];
        Hash *tab[m];
        inicializa(tab, m);
        arquivo = fopen ("Nome␣do␣arquivo", "r");
```

```
        if( arquivo == NULL){
                printf("IMPOSSIVEL␣ABRIR␣O␣ARQUIVO");
                return 0;
        }
        while(EOF != fscanf(arquivo, "%d", &chave)
        && fgets(ltr, 5, arquivo)){
                for(int i = 0; i < 4; i++)
                        ltr[i] = ltr[i+1];
                insere(tab, m, chave, ltr);
        }
        fclose(arquivo);
        imprime(tab, m, &colisao);
        printf("numero␣de␣colis es:␣%d\n", colisao);
        return 0;
}
```

Algoritmo das funções:

```
#ifndef tabHash_c
#define tabHash_c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct aleatorio{
        int chave;
        char let[4];
        struct aluno *prox;
}ale;
typedef ale* Hash;
int hash(int chav, int tam){
        return chav % tam;
}
int hashMult(int chav, int tam){
        float w = 64, a = 19;
        int h = chav * (a/w);
        if(h >= tam)
                return tam -1;
        else
                return h;
}
```

```
void inicializa(Hash *tab, int m){
        int i;
        for(i = 0; i < m; i++)
                tab[i] = NULL;
}
ale* aloca(int chav, char *vet ){
        ale *novo = (ale*) malloc(sizeof(ale));
        novo->chave = chav;
        strcpy(novo->let, vet);
        novo->prox = NULL;
        return novo;
}
void insere(Hash *tab, int m, int chav, char *vet){
        int h = hashMult(chav, m);
        ale *p = tab[h];
        ale *ant = NULL;
        while((p != NULL) && (p->chave != chav)){
                ant = p;
                p = p->prox;
        }
        if(p){
                strcpy(p->let, vet);
                return;
        }
        ale *novo = aloca(chav, vet);
        if(!ant)
                tab[h] = novo;
        else
                ant->prox = novo;

        puts(novo->let);
}
void imprime(Hash *tab, int m, int *colisao){
        int i, coli = 0;
        for(i = 0; i < m; i++)  {
                printf("%d", i);
                if(tab[i]){
                        ale *p = tab[i];
                        printf("\n");
```

```
                        while(p){
                        if(p->prox != NULL)
                                        coli += 1;
                printf("\t%d\t%s\t%p\n", p->chave, p->let, p->prox);
                        p = p->prox;
                        }
                }
                else
                        printf("\n\tNULL\n");

        }
        *colisao = coli;
}
ale* busca(Hash *tab, int m, int chav){
        int h = hashMult(chav, m);
        ale *p = tab[h];
        while((p) && (p->chave != chav))
                p = p->prox;
        return p;
}
void excluir(Hash *tab, int m, int chav){
        int h = hashMult(chav, m);
        if(tab[h] == NULL)
                return;

        ale *p = tab[h];
        ale *ant = NULL;
        while((p != NULL) && (p->chave != chav)){
                ant = p;
                p = p->prox;
        }
        if(p == NULL){
                printf("Chave n o encontrada\n");
                return;
        }
        if( ant == NULL)
                tab[h] = p->prox;
        else
                ant->prox = p->prox;
```

```
        free(p);

}
void libera(Hash *tab, int m){
        int i;
        for(i = 0; i < m; i++){
                if(tab[i]){
                        ale *p = tab[i];
                        ale *q;
                        while(p){
                                q = p;
                                p = p->prox;
                                free(q);
                        }
                }
        }
}
#endif
```

Para a terceira etapa foram criados cerca de 30 arquivos .txt para serem analisadas a quantidade de colisões. Para não haver confusões e erros com os arquivos foi acertado entre as discentes que a organização se daria pelo título sendo [quantidade de elementos]tipo[nº do tipo]-[nºde criação] como nos exemplos a seguir:

- 50tipo1-1.txt
- 100tipo2-4.txt

3 Resultados

Os resultados obtidos foram colocados em gráficos separados por quantidade de elementos gerados, afim de uma melhor análise a respeito da quantidade de colisões por tipo de arquivo. Podemos observar com as figuras abaixo que existe uma média de colisões correlacionada com o a quantidade de elementos presente na lista.

Além disso foi possível calcular a média de colisões de cada tipo de arquivo e sua quantidade de elementos:

- 50 elementos do tipo 1: Média de 30 colisões
- 50 elementos do tipo 2: Média de 29 colisões
- 100 elementos do tipo 1: Média de 62 colisões
- 100 elementos do tipo 2: Média de 67 colisões
- 150 elementos do tipo 1: Média de 86 colisões
- 150 elementos do tipo 2: Média de 102 colisões

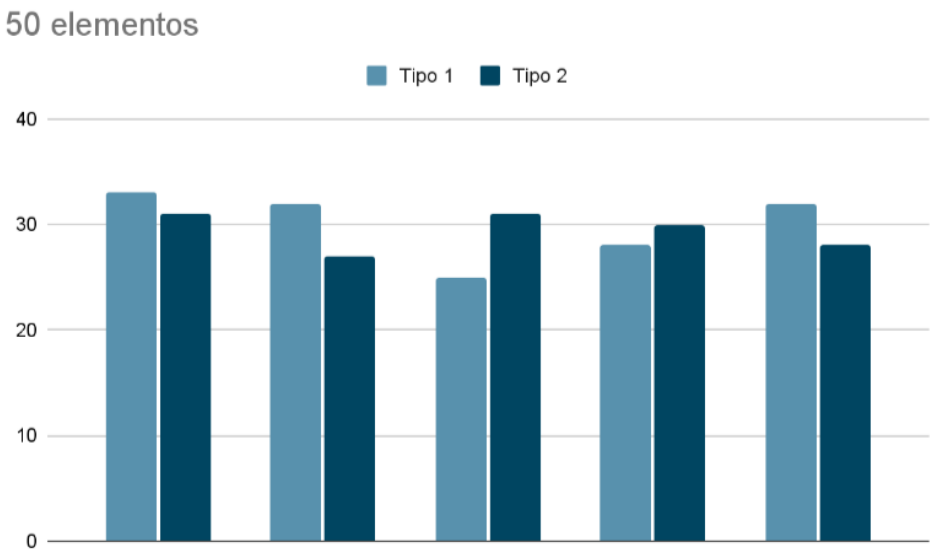

Figura 1 – Colisões dos arquivos de 50 Elementos

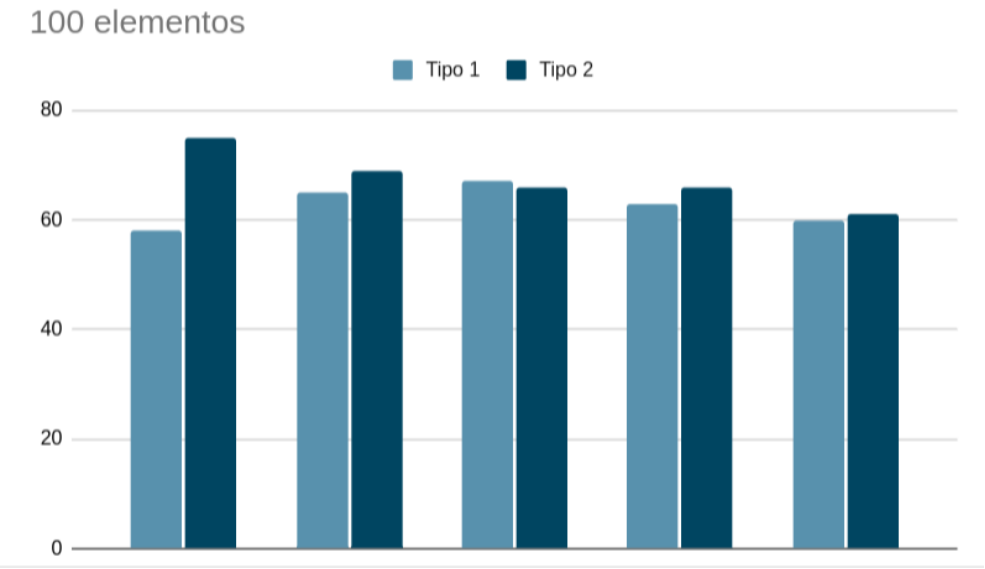

Figura 2 – Colisões dos arquivos de 100 Elementos

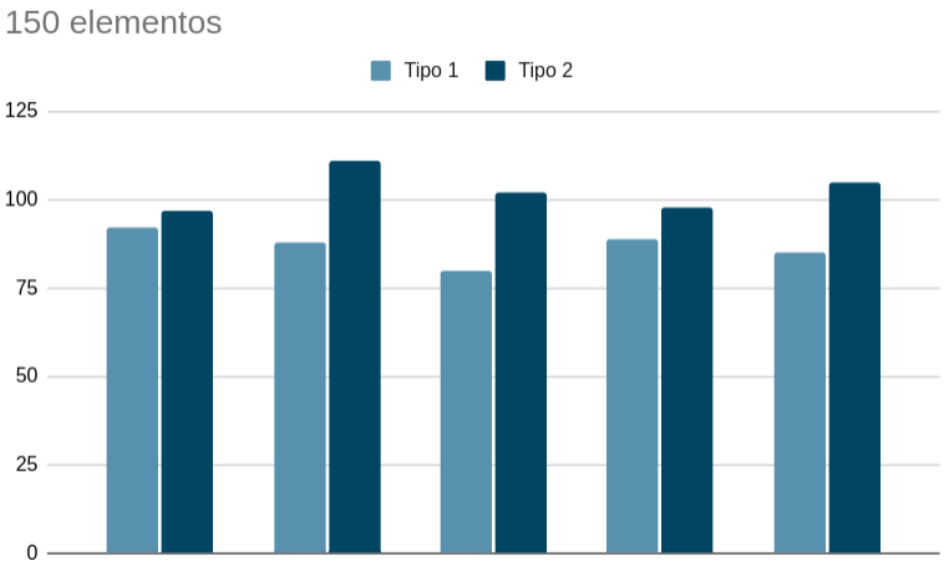

Figura 3 – Colisões dos arquivos de 150 Elementos

4 Considerações Finais

Podemos concluir a partir deste relatório aprendemos primeiramente a manipulação de arquivos com fim de realizar tabelas Hash tendo como funções base de inserção, exclusão, impressão, além de aspectos importantíssimos como tratamento de colisões dessas tabelas, encadeamento dinâmico, contagem e a importância do conceito de chaves utilizando a linguagem c.