

INF1010 - ESTRUTURAS DE DADOS AVANÇADAS - 2022.1 - 3WB

Lab4 - Trabalho sobre Tabelas Hash

Nome: Eric Leão

Matrícula: 2110694

Nome: Marina Schuler Martins

Matrícula: 2110075

Relatório do Laboratório

(Enunciado) Deverá ser entregue ainda um relatório contendo o pseudocódigo dos algoritmos utilizados, os programas fonte, os tempos tomados para cada algoritmo e a geração do gráfico de tamanho da entrada X tempo de execução de cada um, para diferentes valores de entrada por exemplo: 128, 256 e 512 placas (complexidade prática).

Para implementarmos a **função hash** (endereçamento aberto com dispersão dupla) e tratamento de colisão com endereçamento interno, **utilizamos as funções hash1 e hash2**. Seguem seus respectivos pseudocódigos:

MAX definida como 1031

algoritmo hash1(key, c)

 aux ← 0

 k ← primeiro caractere de key

 Enquanto k diferente de 0

 aux ← aux + ((aux << 18) | ((aux >> 13) + k * 1037)) + c * c

 k ← proximo caractere de key

 Fim-enquanto

 Retorne aux % MAX

algoritmo hash2(key, c)

 aux ← 6791

 k ← primeiro caractere de key

 Enquanto k diferente de 0

 aux ← aux + ((aux << 19) ^ ((aux >> 15) + k * 1699)) + c * c

 k ← proximo caractere de key

 Fim-enquanto

 Retorne aux % MAX

1. Pseudocódigos dos algoritmos utilizados: Busca, inclusão e exclusão em vetor de 1031 posições.

a. Busca

```
algoritmo search_key (table, key) {  
  c ← 0  
  Enquanto 1 faça  
    Se c for ímpar faça  
      pos ← hash1(key, c)  
    Senão faça  
      pos ← hash2(key, c)  
    Fim-se  
    Se table[pos] for igual a key  
      retorne pos  
    Fim-se  
    c ← c+1  
  Fim-enquanto
```

b. Inclusão

```
algoritmo insert_key (table, key) {  
  c ← 0  
  Enquanto 1 faça  
    Se c for ímpar faça  
      pos ← hash1(key, c)  
    Senão faça  
      pos ← hash2(key, c)  
    Fim-se  
    Se table[pos] for 0  
      Inserir a chave na posição encontrada  
      Ir para Fim-enquanto  
    Fim-se  
    c ← c+1  
  Fim-enquanto  
  Retorne c
```

c. Exclusão

```
algoritmo delete_key (table, key) {  
  c ← 0  
  vazio ← '\0'  
  Enquanto 1 faça  
    Se c for ímpar faça  
      pos ← hash1(key, c)  
    Senão faça  
      pos ← hash2(key, c)
```

```

        Fim-se
        Se table[pos] for igual a key
            table[pos] ← vazio
        Retorna
    Fim-se
    c ← c+1
Fim-enquanto

```

2. Programa fonte com a implementação dos algoritmos

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#define MAX 1031
#define LEN 7

char ** create_hash(int row, int collumn);

void print_hash(char ** table);

unsigned int hash1(char* key, int c);

unsigned int hash2(char* key, int c);

int insert_key(char ** table, char* key);

void delete_key(char ** table, char* key);

int search_key(char ** table, char* key);

int main(void) {
    FILE * file = fopen("placas.txt", "r");
    char ** table = create_hash(MAX - 1, LEN);
    char *placa;
    int col = 0;
    int pos;
    clock_t start, stop;
    double total;
    if ((placa = (char*)malloc(sizeof(char*) * LEN)) == NULL) {
        fprintf(stderr, "erro alocar memoria.\n");
        exit(1);
    }
}

```

```

}
start = clock();
while (fscanf(file, "%s\n", placa) != EOF) {
    //printf("%s\n",placa);
    col += insert_key(table, placa);
}
stop = clock();
total = (double)(stop - start) / CLOCKS_PER_SEC;
printf("Tempo total para inserir = %f\n", total);
printf("Colisões totais = %d\n", col);
fclose(file);
//print_hash(table);
file = fopen("placas.txt", "r");
start = clock();
while (fscanf(file, "%s\n", placa) != EOF) {
    //printf("%s\n",placa);
    pos = search_key(table, placa);
    //printf("%s\n",table[pos]);
}
stop = clock();
total = (double)(stop - start) / CLOCKS_PER_SEC;
printf("Tempo total para buscar = %f\n", total);
fclose(file);
file = fopen("placas.txt", "r");
while (fscanf(file, "%s\n", placa) != EOF) {
    //printf("%s\n",placa);
    delete_key(table, placa);
    //printf("%s\n",table[pos]);
}
//print_hash(table);
fclose(file);
free(placa);
for (int w = 0; w <= MAX - 1; w++)
    free(table[w]);
free(table);
return 0;
}

char ** create_hash(int row, int collumn) {
    char ** hash;
    if ((hash = (char**)malloc(sizeof(char**) * row)) == NULL) {
        fprintf(stderr, "erro alocar memoria.\n");
        exit(1);
    }
}

```

```

for (int c = 0; c <= row; c++)
    if ((hash[c] = (char*)calloc(column, sizeof(char*))) == NULL) {
        fprintf(stderr, "erro alocar memoria.\n");
        exit(1);
    }
return hash;
}

```

```

unsigned int hash1(char* key, int c) {
    unsigned int aux = 0;
    unsigned int k;
    while ((k = *key++)) {
        aux += ((aux << 18) | ((aux >> 13) + k * 1037)) + c * c;
    }
    return aux % MAX;
}

```

```

unsigned int hash2(char* key, int c) {
    unsigned int aux = 6791;
    unsigned int k;
    while ((k = *key++)) {
        aux += ((aux << 19) ^ ((aux >> 15) + k * 1699)) + c * c;
    }
    return aux % MAX;
}

```

```

int insert_key(char ** table, char* key) {
    int c = 0;
    int pos;
    while (1) {
        if (c % 2) {
            pos = hash1(key, c);
        }
        else {
            pos = hash2(key, c);
        }
        if (*table[pos] == '\0') {
            strcpy(table[pos], key);
            break;
        }
        c++;
    }
    return c;
}

```

```

void delete_key(char ** table, char* key) {
    int c = 0;
    int pos;
    char vazio[] = "\0";
    while (1) {
        if (c % 2) {
            pos = hash1(key, c);
        }
        else {
            pos = hash2(key, c);
        }
        if (strcmp(table[pos], key) == 0) {
            strcpy(table[pos], vazio);
            return;
        }
        c++;
    }
}

```

```

int search_key(char ** table, char* key) {
    int c = 0;
    int pos;
    while (1) {
        if (c % 2) {
            pos = hash1(key, c);
        }
        else {
            pos = hash2(key, c);
        }
        if (strcmp(table[pos], key) == 0) {
            //printf("%s = %s\n", key, table[pos]);
            return pos;
        }
        c++;
    }
}

```

```

void print_hash(char ** table) {
    for (int c = 0; c < MAX; c++)
        printf("%s\n", table[c]);
}

```

3. Tempos tomados para cada algoritmo

Foi pedido que o programa previsse a geração de um **relatório** indicando o **total de colisões geradas** e os **tempos de inclusão e de busca** a todos os elementos do vetor (a busca sendo feita após o término da inclusão de todas as placas). Segue o relatório gerado ao inserirmos 1000 placas no vetor de 1031 posições, conforme os algoritmos acima:

```
> make -s
> ./main
Tempo total para inserir = 0.000196
Colisões totais = 2162
Tempo total para buscar = 0.000242
> 
```

4. Gráfico Tamanho da Entrada x Tempo de Execução, para os valores de entrada: 128, 256, 512 e 1000 placas

a. Inclusão



b. Busca

