Trabalho prático 2 - Cuckoo Hashing

Allan Cedric G. B. Alves da Silva - GRR20190351

18 de fevereiro de 2021

1 Introdução

Este é um relatório realizado para o segundo trabalho da disciplina de **Algoritmos e Estruturas de Dados III - (CI1057) - ERE2, do curso de Ciência da Computação, UFPR**.

Em um primeiro momento será discorrido brevemente sobre a implementação da **tabela Hash** para a aplicação da técnica conhecida como **Cuckoo Hashing**. E no final, será tratado sobre o *output* esperado pelo trabalho.

2 Implementação da tabela Hash

A **tabela Hash** foi construída como um vetor de *structs*, do tipo *hash_table_t*. Cada uma dessas *structs* carrega um **estado(*state* - 'E' = Empty, 'F' = Filled, 'R' = Removed)**, um **ID tabular(*tableId*)** e **um ponteiro para uma célula de dados**, a qual neste caso vai ser uma chave simples em conjunto com o seu valor hash. A implementação como um todo foi realizada na linguagem ***C***.

```c
/* === Estrutura de uma célula de dados da tabela Hash === */
typedef struct data_t
{
    int key, hash;
} data_t;

/* === Estrutura de indexação (slot) da tabela Hash === */
typedef struct hash_table_t
{
    data_t *data;
    char state, *tableId;
} hash_table_t;
```

Figura 1: Estrutura de uma célula de dados e de um slot da tabela Hash (Arquivo *CuckooHash/table.h*)

A biblioteca da **tabela Hash** se encontra no diretório chamado *CuckooHash*, mais precisamente o conjunto de arquivos: *CuckooHash/table.h* e *CuckooHash/table.c*. A estrutura geral para a aplicação da técnica **Cuckoo Hashing** se encontra a seguir:

```c
/* === Estrutura que administra duas tabelas Hash === */
typedef struct cuckoo_hash_t
{
    hash_table_t *table_1, *table_2;
    int size_1, size_2;
} cuckoo_hash_t;
```

Figura 2: Estrutura que armazena duas tabelas Hash. Isso facilita a aplicação do **Cuckoo Hashing** (Arquivo *CuckooHash/table.h*)

3 Problema proposto

O objetivo principal deste trabalho é implementar a inserção, remoção e busca de chaves em tabelas Hash, no caso duas, com a técnica **Cuckoo Hashing**. Além de realizar um teste simples de impressão. O arquivo fonte de teste, *myht.c*, possui a implementação para a impressão 'emOrdem' das chaves da tabela Hash, com os respectivos ID's tabulares e valores hash (índices).

O *output* esperado foi realizado de forma extremamente simples, com apenas **o parsing das duas tabelas Hash para uma árvore AVL**. E com isso, foi feita a chamada *printCuckooHashTables(&cuckooHash, rootAVL)* no programa principal. A árvore AVL auxilia a tabela Hash na impressão 'emOrdem'.

```c
void hashTableToAVL(hash_table_t *table, int sizeTable, AVL **rootAVL)
{
    data_t *data;
    int i;
    for (i = 0; i < sizeTable; i++)
    {
        data = table[i].data;
        if (data)
            *rootAVL = insertNodeAVL(*rootAVL, data->key);
    }
}

void printCuckooHashTables(cuckoo_hash_t *ch, AVL *rootAVL)
{
    if (rootAVL)
    {
        printCuckooHashTables(ch, rootAVL->left);
        hash_table_t *dataToPrint = searchCuckooHash(ch, rootAVL->key);
        fprintf(stdout, "%i,%s,%i\n", dataToPrint->data->key, dataToPrint->tableId, dataToPrint->data->hash);
        printCuckooHashTables(ch, rootAVL->right);
    }
}
```

Figura 3: Algoritmo de parsing e do output esperado (Arquivo *myht.c*)