

# Relatório: Trabalho 2 - Cuckoo Hashing

Pedro Folloni Pessler  
*Departamento de Informática*  
*Universidade Federal do Paraná - UFPR*  
Curitiba, Brasil  
pfp22@inf.ufpr.br

## Resumo

Este relatório documenta o software desenvolvido para o segundo trabalho prático da disciplina CI1057 - Algoritmos e Estruturas de Dados 3, do Departamento de Informática da UFPR. O programa é uma implementação em linguagem C da busca, inclusão e exclusão de valores em uma tabela hash de endereçamento aberto, simulando o algoritmo de Cuckoo Hash.

## 1 Estruturas de dados utilizadas

A tabela hash foi implementada usando as seguintes estruturas, definidas na biblioteca `libchash.h`:

- Entrada da tabela: contém uma chave de busca, uma variável que especifica se a entrada está vazia ou não, e uma que especifica se a chave foi excluída.

```
struct Entrada {  
    int chave, vazia, delet;  
};
```

- Tabela Hash: contém dois vetores de Entradas, um para cada tabela do Cuckoo Hash (OBS: `TABLESIZE` é um macro, definido nessa implementação como 11).

```
struct Hash {  
    struct Entrada t1[TABLESIZE], t2[TABLESIZE];  
};
```

## 2 Bibliotecas desenvolvidas

A biblioteca desenvolvida para o trabalho foi a `libchash.hc`, que define as structs acima, além das seguintes funções (considere, para as funções a seguir,  $m = \text{TABLESIZE}$ ):

- `struct Hash cria_tabela()`; – Cria uma tabela hash e marca todas as entradas como vazias.
- `int h1(int k)`; – Função hash para a tabela 1, dada por  $h_1(k) = k \bmod m$ .
- `int h2(int k)`; – Função hash para a tabela 2, dada por  $h_2(k) = \lfloor m(0,9k - \lfloor 0,9k \rfloor) \rfloor$ .

- `int busca_chash(struct Hash *hash, int k);` – Busca uma chave na tabela. Se a posição  $h_1(k)$  na tabela 1 estiver vazia, checa a posição  $h_2(k)$  da tabela 2 (veja detalhes sobre esse comportamento na seção 3, adiante). Se a chave não estiver em nenhuma das tabelas, retorna -1. Se estiver na tabela 1, retorna a posição (um inteiro  $i \in [0..10]$ ). Se estiver na tabela 2, retorna a posição mais  $m$  (um inteiro  $j \in [11..21]$ ). Esses valores foram escolhidos para facilitar a interpretação do resultado em um possível uso da função de busca.
- `int insere_chash(struct Hash *hash, int k);` – Insere uma chave na tabela. Se a inserção ocorrer normalmente, retorna 0. Se houver tentativa de inserir um valor duplicado, não faz nada e retorna 1. Se o valor não puder ser inserido (colisão na tabela 2), não faz nada e retorna 2.
- `int exclui_chash(struct Hash *hash, int k);` – Exclui um valor da tabela hash, seja da tabela 1 ou da tabela 2. Na prática, a posição na tabela é marcada como vazia. Se a exclusão ocorrer normalmente, retorna 0. Se a chave  $k$  não estiver na tabela, não faz nada e retorna 1.
- `void imprime_chash(struct Hash *hash);` – Imprime as chaves armazenadas na tabela em ordem crescente, juntamente com a tabela na qual a chave  $k$  se encontra (T1 ou T2), seguido pela posição da chave na tabela (inteiro  $i \in [0..10]$ ). A saída segue o formato `k,T[1|2],i`.  
OBS: Para realizar a impressão em ordem, foi usada a função `qsort`, da biblioteca padrão da linguagem C. Para tanto, foi desenvolvida uma nova função, definida com a palavra-chave `static` (ou seja, que só pode ser acessada pelo código da `libchash.c`):
- `static int compara_chaves(const void *a, const void *b);` – Compara duas estruturas `Entrada`, por meio do campo `chave`: retorna um valor menor que, igual a, ou maior que 0 se a primeira chave for, respectivamente, menor que, igual à, ou maior que a segunda.