Trabalho prático - Hashing

Bruno Fuchs Santos da Silva[1], Matheus Telles Batista[2]

,2[1]Departamento de Informática Universidade Federal do Paraná – UFPR Curitiba, Brasil

Resumo
Relatório contendo a sintetização da implementação de uma tabela hash com endereçamento aberto.

Keywords
hash, endereaçamento aberto, inserção, remoção

1. Introdução

1.1. O que é Hashing

A tabela hash, também conhecida como tabela de dispersão, é uma estrutura de dados no qual os termos de pesquisa se relacionam com dados em uma tabela com valores. Se desenvolveu principalmente para uma maior rapidez na busca de chaves e valores dentro da memória. Os valores salvos dentro da tabela são gerados com um algorítmo de disperção que pode variar dependendo do propósito do projeto (pesquisa de arquivos, criptografia, senhas).

Visto o problema de colisões ja descrito em aula, o trabalho presente foi desenvolvido com o *Cuckoo Hashing*.

1.2. Cuckoo Hashing

Contornando o problema de colisões, é usado este método no trabalho. Sendo k_i como a antiga chave e k_j como a chave que queremos inserir, a função de inserção toma essa forma:

$$insert(k_j) = \begin{cases} T_1[f_1(k_j)] = k_j, & \text{se } T1[f_1(k_i)] = \text{NULL} \\ T_2[f_2(k_i)] = k_i, & \text{caso contrário.} \\ T_1[f_1(k_j)] = k_j, & \text{caso contrário.} \end{cases}$$

O rehashing (tecnica utilizada após a chave fazer o caminho de T1 $\rightarrow T_2 \rightarrow T_1$ e ainda estar ocupado) não foi desenvolvido vide as regras para a montagem do algorítmo.

Sobre as especificações, as tabelas tem tamanho $m = 11$ e as seguintes funções de dispersão, para qualquer k sendo uma chave inteira:

$h_1(k) = k \mod m$, $h_2(k) = \lfloor m(0.9k - \lfloor 0.9k \rfloor) \rfloor$

2. Implementação

Abaixo está os detalhes da implementação de cada função e *struct*.

2.1. Structs

Seguem abaixo as *structs* necessárias para a aplicação: a primeira sendo o dicionário e a segunda a tabela hash que foi usada para salvar as chaves.

CI 057 - Algoritmos e Estrutura de Dados III.
bruno.fuchs@inf.ufpr.br (B. F. S. d. Silva); mtb21@ufpr.br (M. T. Batista)
https://github.com/AlmiranteFuchs (B. F. S. d. Silva); https://github.com/TilTelles (M. T. Batista)

```c
typedef struct
{
    int key;
    int value;
    char* table_name;
} dict;
```

```c
typedef struct
{
    int size;
    dict **table;
} hash_table;
```

2.2. Funções

2.3. Criação

Assinatura da função que cria a tabela com k ou *size* chaves vazias e ao lado uma criação de chave no dicionário:

```c
hash_table *new_hash_table(int size); | dict *new_dict(int value);
```

2.4. Inserção (Cuckoo insert)

A função de inserção é a descrita na seção seção 1.2, segue abaixo a assinatura da mesma no código:

```c
void insert_hash(hash_table *table, hash_table *table2, int value);
```

2.5. Remoção

A remoção ocorre ao chamar a função:

```c
void remove_hash(hash_table *table, hash_table *table2, int value);
```

Sendo o primeiro e segundo argumentos sendo as tabelas T1 e T2 e o valor procurado (chamaremos de k) passado como terceiro argumento.

A verificação ocorre aplicando as funções $f_1(k)$ e $f_2(k)$ e verificando se há nos respectivos índices o valor k. Se houver, ocorre a remoção.

2.6. Destruição

A destruição ocorre por meio da função abaixo que apenas precisa receber a tabela como argumento.

```c
void destroy_hash_table(hash_table *table);
```

2.7. Leitura e Saída

A main do programa recebe entrada padrão (stdin), cada linha é quebrada em duas partes, onde a primeira indica se é remoção ou inserção e a segunda é a chave que será removido ou inserido até não houver mais linhas.

```c
while ((read = getline(&line, &len, stdin)) != -1) { [...] }
```

A saída está indicada pela função *get_ordered_hashs(t1,t2)*, no qual é criada um vetor único com todos os valores de T_1 ordenados e logo após T_2 ordenados, sendo então T_1 e T_2 concatenados. Após isso é organizado as listas concatenadas pelo método de ordenação *quick_sort(ordered_hash, 0, c - 1)*.

Depois da função *quicksort()*, a função *get_hash(table, table2, ordered_hash[i])* obtém a chave $T_3[i]$ a tabela ordenada.

```c
get_ordered_hashs(t1, t2) {[...]
    quick_sort(ordered_hash, 0, c - 1);
    [...]
}
```

```c
for (i = 0; i < c; i++) {[...]
    dict *d = get_hash(t, t2, ordered_hash[i]);
    [...]
}
```