



**INSTITUTO FEDERAL CATARINENSE  
CAMPUS VIDEIRA**

Lucas Magalhães

**Implementação de uma tabela Hash**

Videira/SC

## Introdução

Neste relatório, descreve o método utilizado para resolver o problema de implementação de uma tabela hash com operações de inserção, busca, remoção e ordenação dos elementos armazenados. O código fornecido é escrito em linguagem C e usa uma estrutura de dados de tabela hash para armazenar uma lista encadeada de elementos em cada posição da tabela.

## Metodologia

### Função de Hash:

A função de hash utilizada é uma função simples baseada no algoritmo de hash chamado "Multiplicative Hashing". Essa função percorre cada caractere do nome, multiplicando o hash atual por 31 e adicionando o valor ASCII do caractere. Em seguida, retorna o hash calculado módulo o tamanho da tabela (TABLE\_SIZE).

```
26
27 v unsigned int funcaoHash(const char* chave) {
28     unsigned int hash = 0;
29     unsigned int len = strlen(chave);
30
31 v     for (unsigned int i = 0; i < len; i++) {
32         hash = hash * 31 + chave[i];
33     }
34
35     return hash % TABLE_SIZE;
36 }
37
```

### Inserção de Elementos:

A função de inserção recebe um nome como entrada e calcula a chave correspondente usando a função de hash. Em seguida, cria um novo nó para armazenar o nome e o insere na lista encadeada da posição da tabela correspondente. Se a posição estiver vazia, o novo nó é adicionado como o primeiro nó. Caso contrário, o novo nó é adicionado no início da lista encadeada e os ponteiros são ajustados corretamente.

### Busca de Elementos:

A função de busca recebe um nome como entrada e calcula a chave correspondente usando a função de hash. Em seguida, percorre a lista encadeada da posição da tabela correspondente e verifica se o nome buscado corresponde a algum dos nomes armazenados. Se encontrar um nome correspondente, retorna 1, caso contrário, retorna 0.

### Remoção de Elementos:

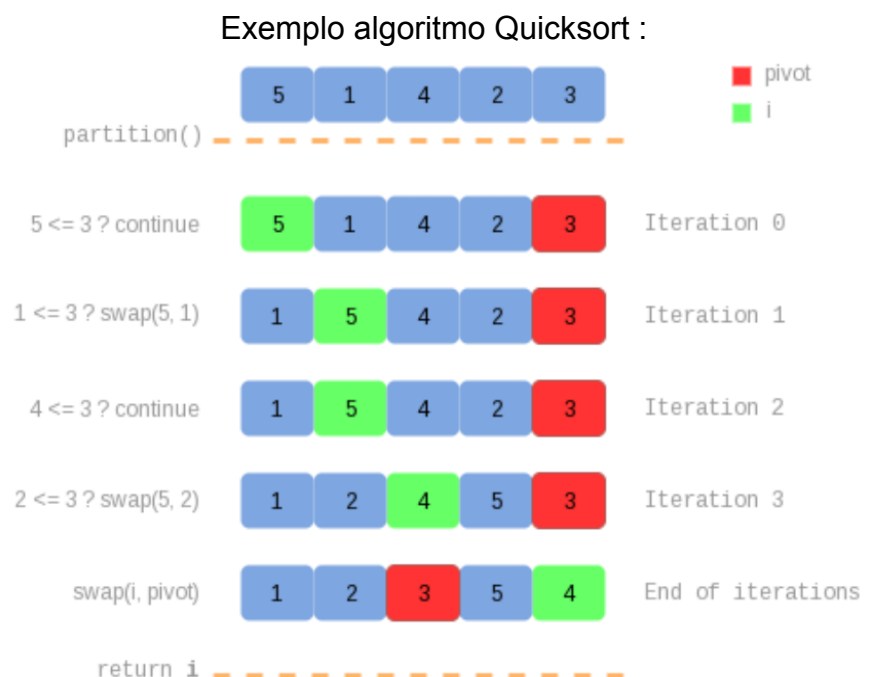
A função de remoção recebe um nome como entrada e calcula a chave correspondente usando a função de hash. Em seguida, percorre a lista encadeada da posição da tabela correspondente e procura o nome a ser removido. Se o nome for encontrado, os ponteiros da lista encadeada são ajustados para remover o nó correspondente. O espaço de memória ocupado pelo nó removido é liberado.

### Contagem de Elementos:

A função de contagem de elementos recebe uma chave como entrada e percorre a lista encadeada correspondente contando o número de elementos presentes. Retorna o total de elementos encontrados.

### Ordenação dos Elementos:

Para ordenar os elementos armazenados, o algoritmo de ordenação "Quicksort" é utilizado. Uma função de partição é implementada para escolher um pivô e rearranjar os elementos menores à esquerda e os maiores à direita do pivô. Em seguida, o Quicksort é aplicado recursivamente às sublistas da esquerda e da direita do pivô até que todos os elementos estejam ordenados.



### **Impressão dos Elementos:**

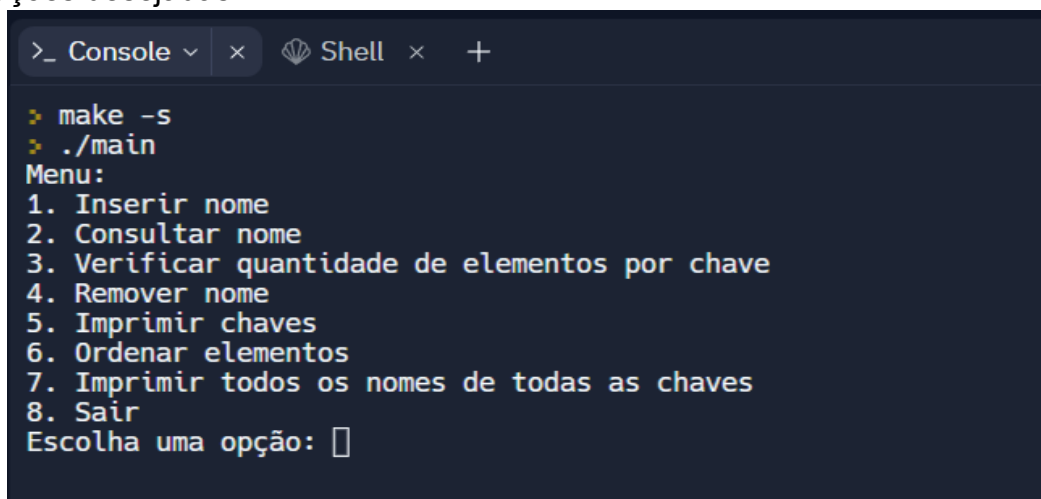
A função de impressão da tabela hash exibe a quantidade de elementos em cada posição da tabela. Percorre todas as posições da tabela e usa a função de contagem de elementos para obter o número de elementos presentes em cada posição.

### **Impressão de Todos os Nomes Ordenados:**

A função de impressão de todos os nomes ordenados coleta todos os nomes armazenados na tabela hash em um array. Em seguida, usa a função de ordenação "qsort" da biblioteca padrão do C para ordenar o array de nomes em ordem alfabética. Por fim, os nomes ordenados são impressos na saída padrão.

## **Resultados e Conclusões:**

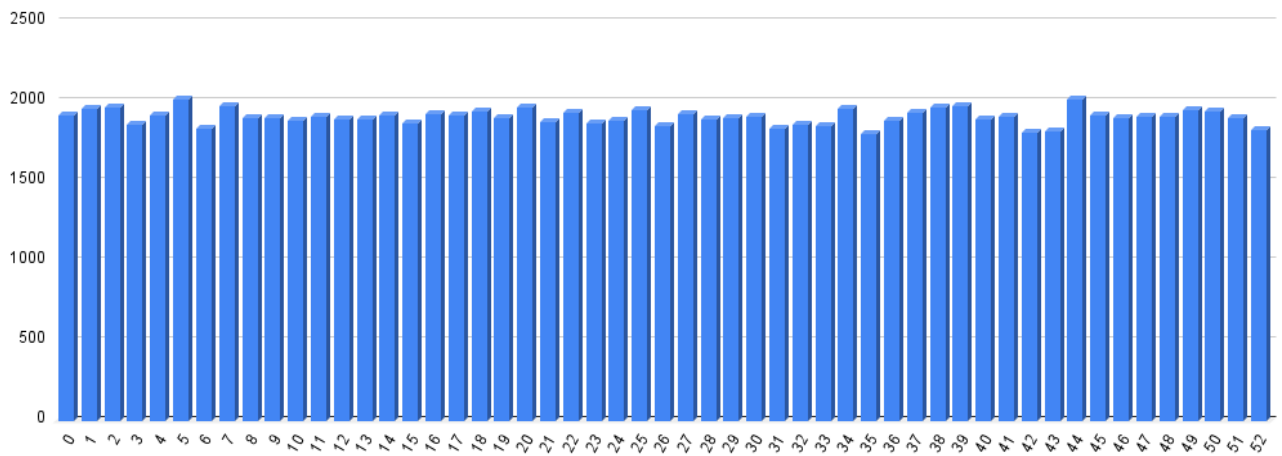
A implementação da tabela hash com as operações de inserção, busca, remoção, contagem e ordenação de elementos foi bem-sucedida. O código em C funciona conforme o esperado e realiza todas as operações de maneira correta. Durante a execução do programa, é fornecido um menu interativo para o usuário escolher as operações desejadas.



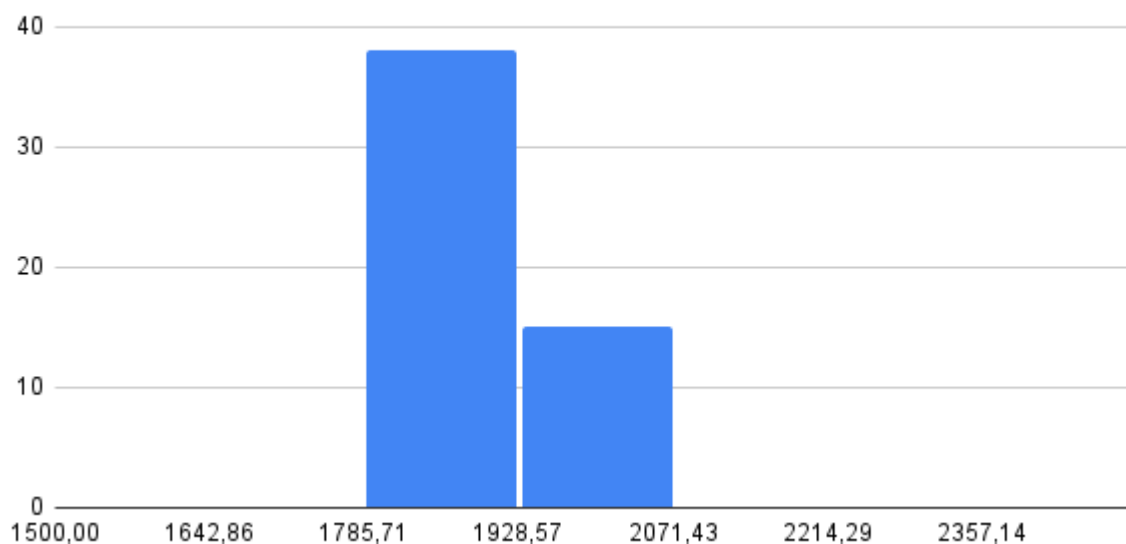
```
>_ Console x Shell x +
> make -s
> ./main
Menu:
1. Inserir nome
2. Consultar nome
3. Verificar quantidade de elementos por chave
4. Remover nome
5. Imprimir chaves
6. Ordenar elementos
7. Imprimir todos os nomes de todas as chaves
8. Sair
Escolha uma opção: [ ]
```

No geral, a implementação é eficiente em termos de tempo de execução para a maioria das operações, pois a estrutura de dados de tabela hash permite um acesso rápido e direto aos elementos.

## Aplicação do código com a base de dados com 100.788 nomes:



### Histograma



São ao todo 53 chaves para 100.788 nomes armazenados.

### Resultados:

A quantidade de elementos em cada chave varia, mas no geral, as diferenças não são muito grandes. A diferença máxima entre as chaves é de 212 elementos (Chave 44 com 2010 elementos e Chave 35 com 1800 elementos). Isso indica uma distribuição uniforme dos elementos na tabela hash.

A maioria das chaves possui uma quantidade de elementos próxima da média (por volta de 1900 a 1950 elementos). Poucas chaves têm uma quantidade significativamente maior ou menor.

### **A hipótese do hashing uniforme foi alcançada?**

A hipótese do hashing uniforme pode ser considerada alcançada com base na distribuição relativamente uniforme dos elementos nas chaves. Não há uma concentração excessiva de elementos em um número reduzido de chaves, em chaves com poucos ou nenhum elemento.

Em resumo, com base nos resultados obtidos a partir dos 100788 nomes distribuídos nas chaves da tabela hash, podemos afirmar que o código implementa um tratamento de colisão eficiente por meio do encadeamento. A tabela hash apresenta uma distribuição uniforme dos elementos nas chaves, assim a hipótese do hashing uniforme foi alcançada com sucesso.