

Ministério da Educação

Secretaria de Educação Profissional e Tecnológica Instituto Federal Catarinense

Campus Videira

**CRISTIANO RODRIGUES PIROLLI**

**RELATÓRIO FINAL DE ESTRUTURA DE DADOS I**

Videira

2024

**CRISTIANO RODRIGUES PIROLLI**

**RELATÓRIO FINAL DE ESTRUTURA DE DADOS I**

Relatório apresentado ao Curso de graduação em Ciência da Computação do Instituto Federal Catarinense – Campus Videira

Orientador: Diego Ricardo Krohl

Videira

2024

**SUMÁRIO**

1. **OBJETIVO** ..................................................................................................... 5
2. **ESTRUTURA GERAL** .................................................................................. 5
3. **FUNCIONALIDADES IMPLEMENTADAS** ............................................. 5
4. **DETALHES TÉCNICOS** .............................................................................. 6
5. **MELHORIAS E AJUSTES** .......................................................................... 6
6. **QUANDO IMPLEMENTAR TRATAMENTO COLISÃO** ...................... 6
7. **COMO FOI IMPLEMENTADO** ................................................................. 7
8. **AVALIAÇÃO DO HASHING UNIFORME** .............................................. 7
9. **ANÁLISE DA FREQUÊNCIA DE CADA BUCKET** ............................... 7
10. **TESTES REALIZADOS** ............................................................................. 8

**RESUMO**

Este relatório apresenta a implementação e a análise de uma tabela hash com encadeamento para a resolução de colisões com o propósito de armazenamento e busca de nomes. Neste sentido, foram tratadas questões como a necessidade do tratamento de colisões e, assim, o encadeamento como a estratégia mais adequada para as situações em que o mesmo índice hash é utilizado para operações de inserção. A tabela hash que foi utiliada e a função hash em si, foram projetadas para que a tabela se aproximasse do hashing hipotético uniforme. Uma análise das frequências nos intervalos indica que houve uma distribuição razoável, com baixo número de casos de sobrecarga.

1. **OBJETIVO**

O obetivo deste projeto, oi criar um programa em C que implementasse uma tabela hash para armaenar nomes, com funcionalidades avançadas de inserção, pesquisa, remoção, exibição e cálculo de estatísticas dos elementos em cada *bucket* da tabela.

1. **ESTRUTURA GERAL**

O programa foi desenvolvido com as seguintes estruturas principais:

1. Tabela Hash: A tabela hash utilia um número primo como tamanho e aplica uma função hash baseada no valor ASCII dos caracteres do nome. A função hash foi ajustada para evitar colisões ao usar o número primo 37.
2. Buckets: Cada bucket da tabela hash é uma lista duplamente encadeada que armazena os nomes que compartilham o mesmo índice hash.
3. Nó: Cada nó da lista, armazena um nome e referências para o próximo e o anterior, permitindo a remoção e inserção eficientes.
4. **FUNCIONALIDADES IMPLEMENTADAS**

* Carregamento de Dados: O programa lê nomes de um arquivo de texto (“nomes.txt”) e os insere na tabela hash, utiliando a função *insert*;
* Inserção de Nome: Usuário pode inserir um nome específico que será armazenado no bucket adequado de acordo com o hash calculado;
* Pesquisa de Nome: Usuário pode pesquisar um nome. O programa retorna o índice do bucket e a posição do nome caso ele seja encontrado;
* Remoção de Nome:Usuário pode remover um nome específico, e o programa atualiza o bucket removendo o *nó* correspondente;
* Exibição dos Buckets e Elementos: Com a opção de *exibir todos os buckets,* o programa lista cada bucket junto com os nomes armazenados no mesmo;
* Estatísticas dos Buckets: Foi implementada uma função avançada para exibir estatísticas, incluindo:
  + Número total de nomes armaenados;
  + Bucket com o maior e o menor número de nomes;
  + Média de nomes por bucket;
  + Diferença em percentual do maior e menor bucket, em relação à média;
  + Diferença entre os buckets com mais e menos elementos.
* Quantidade de Elementos e Exibição do Bucket Específico: Nessa opção, o usuário pode escolher um bucket específico para ver quantos elementos ele possui e listar os nomes contidos nele.

1. **DETALHES TÉCNICOS**
2. Função Hash: A função hash, utiliza multiplicação com um número primo para distribuir melhor os valore, minimizando colisões.
3. Lista Encadeada Dupla: Permite navegação e remoção de elementos no bucket de forma eficiente.
4. Módulo de Estatísticas: Calcula a média e o desvio do número de elementos por bucket, além de apresentar o maior e menor bucket em relação à média.
5. **MELHORIAS E AJUSTES**

Durante o desenvolvimento, foram feitos ajustes importantes:

* Adição de exibição de todos os nomes de um bucket específico;
* Implementação de cálculo estatístico mais detalhado, incluindo a porcentagem de desvio do maior e menor bucket;
* Integração de todas as funções em um menu com estrutura de fácil navegação e exibição organizada.

1. **QUANDO IMPLEMENTAR O TRATAMENTO DE COLISÃO?**

A colisão foi tratada sempre que ocorreu o mapeamento de múltiplos elementos para o mesmo índice, pois é inevitável em uma tabela hash com ocupação significativa ou tamanho fixo, como a que utilizamos (73 buckets). Isso ajudou a evitar perda de dados e permitiu que todos os elementos fossem armazenados com eficiência.

1. **COMO FOI IMPLEMENTADO?**

O tratamento de colisões foi implementado usando encadeamento externo, onde cada bucket é uma lista encadeada dupla. Quando múltiplos elementos caem no mesmo índice, eles são armazenados em uma lista ligada dentro desse bucket, preservando todos os elementos e permitindo operações de inserção e remoção com facilidade.

1. **AVALIAÇÃO EM RELAÇÃO À HIPÓTESE DO HASHING UNIFORME**

A hipótese do hashing uniforme sugere que os elementos devem ser distribuídos aleatoriamente pelos buckets, minimizando a ocorrência de colisões concentradas em alguns índices. Nesse código, a escolha da função hash com o número primo 37 e o tamanho da tabela como um número primo (73) ajudou a distribuir os elementos de forma mais uniforme. Embora não seja possível garantir distribuição uniforme devido à natureza dos dados, essa combinação aumentou as chances de uma dispersão mais balanceada dos elementos.

1. **ANÁLISE DA FREQUÊNCIA DE CADA BUCKET**

No código, a função *printStats* exibe a frequência de elementos em cada bucket, mostrando o bucket com mais elementos, o de menos elementos, a média e a diferença entre eles. Essa análise permitiu verificar a uniformidade da distribuição. A observação de que alguns buckets contêm mais elementos que outros, é comum e esperada, mas os dados indicam que a maioria dos buckets contém uma quantidade razoável de elementos, NÃO HAVENDO GRANDES CONCENTRAÇÕES.

1. **TESTES REALIZADOS**

Todos os testes foram realizados utiliando mód 73, **apenas alterando o valor hash**.

Teste com hash 31:

**Maior Bucket - 33:** *1462 elementos*

**Menor Bucket - 12:** *1302 elementos*

**Diferença:** *160 elementos*

Teste com hash 37: **CÓDIGO IMPLEMENTADO NESSES PARÂMETROS**

**Maior Bucket - 54:** *1462 elementos*

**Menor Bucket - 44:** *1314 elementos*

**Diferença:** *148 elementos*

Teste com hash 41:

**Maior Bucket - 17:** *1458 elementos*

**Menor Bucket - 67:** *1302 elementos*

**Diferença:** *156 elementos*

Teste com hash 53:

**Maior Bucket - 50:** *1467 elementos*

**Menor Bucket - 69:** *1322 elementos*

**Diferença:** *145 elementos*

Teste com hash 59:

**Maior Bucket - 37:** *1441 elementos*

**Menor Bucket - 6:** *1286 elementos*

**Diferença:** *155 elementos*

**LINK COM UM GRÁFICO MOSTRANDO A OCILAÇÃO ENTRE BUCKETS:**

https://docs.google.com/spreadsheets/d/1WzzEnkJpLwkoUx-KDGojyb3aOST-kFGRjb3lywtTRAc/edit?usp=sharing