

**MINISTÉRIO DA EDUCAÇÃO  
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA  
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
CATARINENSE  
CÂMPUS VIDEIRA**

**Bacharelado em Ciência da Computação**

# **Relatório do Trabalho Final da Disciplina de Estrutura de Dados**

**Henrique Martinelli Pinheiro**

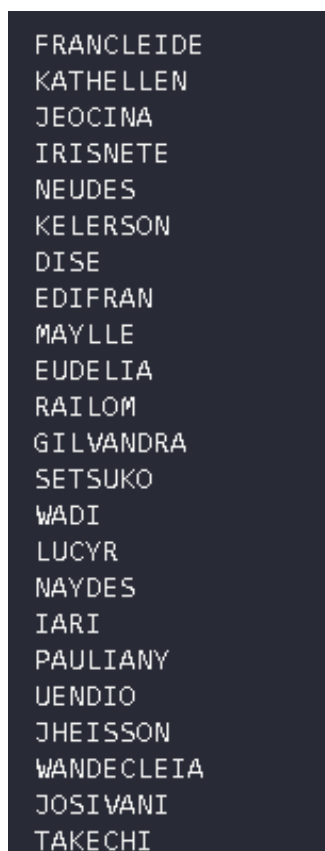
**Videira, Julho de 2022**

## 1. Introdução

O trabalho foi desenvolvido em C, usando o arquivo com cem mil nomes, que foram carregados pelo programa, inseridos em uma tabela hash implementada com lista encadeada dupla, sendo implementadas funções de alocação e liberação de memória, inserção dos nomes em um lista, inserção dessa lista em outra lista para gerar a tabela hash, além de funções para gerar o hash, imprimir valores, encontrar uma chave, elemento ou nome específico dentro da tabela hash, uma função de ordenação, baseada no quicksort, dentre outras.

## 2. Fluxo do Programa

O programa inicia alocando memória para a lista, alocando memória usando malloc para os índices(chaves) que vão guardar os nomes, e inserindo os índices na lista, sendo sua chave definida usando o tamanho da lista geral como referência. Depois disso, o arquivo é carregado usando o comando FILE, os nomes são lidos, é feito o hash deles, então é encontrada a chave correspondente e o nome inserido na chave. Nesse ponto os nomes não estão ordenados como pode ser visto na figura 1, isso só ocorrerá quando o usuário selecionar a opção correspondente no menu.



```
FRANCLEIDE
KATHELLEN
JEOCINA
IRISNETE
NEUDES
KELERSON
DISE
EDIFRAN
MAYLLE
EUDELIA
RAILOM
GILVANDRA
SETSUKO
WADI
LUCYR
MAYDES
IARI
PAULIANY
UENDIO
JHEISSON
WANDECLEIA
JOSIVANI
TAKECHI
```

Figura 1.

O método de ordenação exigido na descrição da atividade foi o quicksort, que foi implementado no trabalho, utilizando pivô no fim da lista

Em seguida um menu é mostrado para o usuário com várias opções, são elas: imprimir toda uma chave, informar e inserir um novo nome na lista, imprimir o tamanho de todas as chaves, encontrar a posição de um nome dentro da sua chave e remover um nome da lista, e ordenar toda a lista. O menu fica se repetindo até que o usuário digite 0 e finalize o programa. É importante lembrar que, sempre que o usuário insere um novo nome na lista, o índice no qual o nome foi inserido é reordenado.

### **3. Hash**

O algoritmo foi implementado utilizando lista encadeada dupla para evitar colisões, já que ela é dinâmica nunca vai ficar cheia, podendo sempre ser expandida e um elemento nunca vai acabar ocupando a mesma posição que outro. Sempre que possível é ideal implementar lista dupla para evitar colisão, pois ela não é tão diferente da lista simples com uma complexidade semelhante e ainda pode evitar muitos problemas.

A função hash utilizada no desenvolvimento do trabalho foi uma função hash modular, de forma que primeiro os caracteres do nome são convertidos para o padrão ASCII, somados, multiplicados por 31, um número primo, e por último, é feito o resto da divisão por 53 que é o número de chaves.

Os motivos de uma função modular ter sido escolhida são o fato dela ser simples de entender e implementar, ter sido usada como exemplo durante a aula, além de ter apresentado um resultado satisfatório em relação ao hash uniforme.

### **4. Ordenação**

A ordenação foi feita usando o método quicksort, que foi exigido na descrição do trabalho.

O quicksort trabalha com a ideia de dividir e conquistar. No trabalho, o método escolhido utiliza como pivô o último elemento da lista.

Primeiro ele escolhe um elemento de referência, o pivô, e passa a comparar a partir do começo da lista, e se um elemento menor estiver no final da lista eles são trocados, assim passando para o próximo elemento. Quando o elemento comparado for o mesmo que o pivô,

significa que uma repetição terminou e então uma estrutura recursiva é chamada para continuar a ordenação.

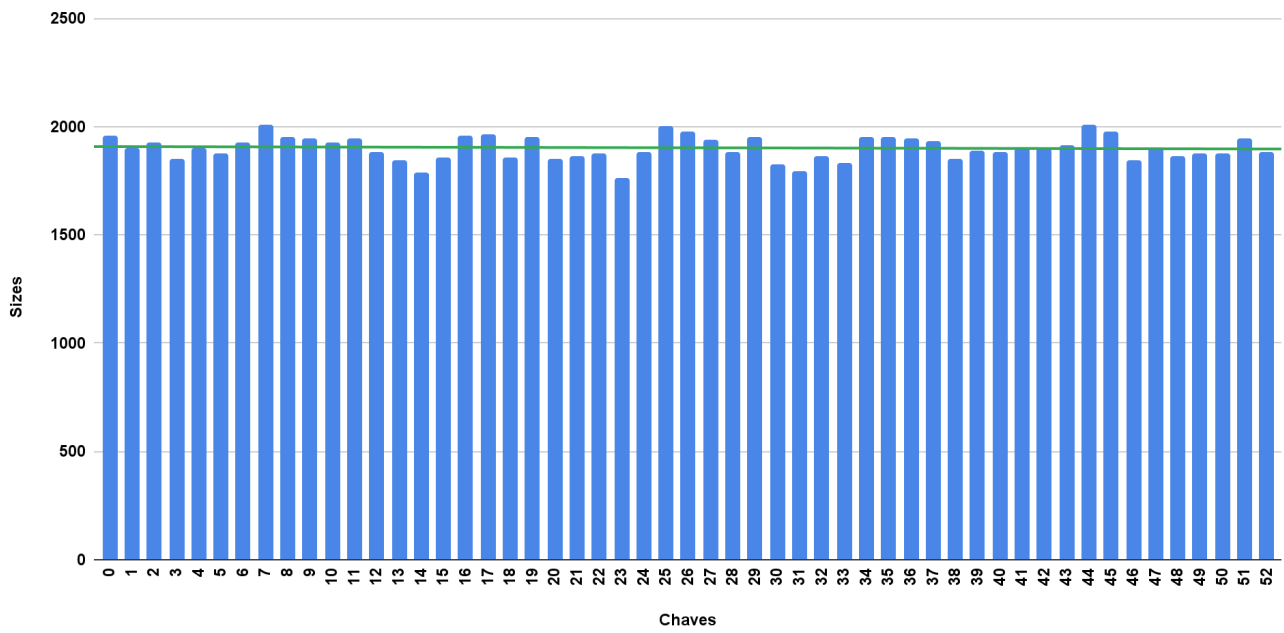
## 5. Hash Uniforme

O hash uniforme é o conceito de que as funções hashing distribuem de maneira uniforme os valores entre as chaves, mas na prática nenhuma função faz isso, pois, tomando como exemplo a função utilizada neste trabalho, ele leva em conta o valor do padrão ASCII dos caracteres, dessa forma, a menos que os nomes inseridos fossem previamente escolhidos, é impossível o hash ser uniforme, que é o que acontece na vida real. O que se busca é se aproximar do hash uniforme, de forma que uma chave não fique com muito mais elementos do que outra.

Neste trabalho, os 100.788 nomes foram divididos em 53 chaves, sendo que a média deveria ser 1901 nomes em cada chave. A chave com mais nomes, teve 2008, 107 nomes a mais que a média, representando aproximadamente 0.10% do total, enquanto a chave com menos nomes teve 1761, 140 nomes a menos que a média, o que representa aproximadamente 0.14% do total de nomes, com um desvio padrão de aproximadamente 55,47. Considerando que a maioria das chaves ficou com uma quantidade de nomes na casa dos 1800 ou 1900, pode-se considerar que a ideia de se aproximar do hash uniforme foi satisfeita até certo ponto, já que os nomes ficaram relativamente próximos da média.

Abaixo é possível visualizar um gráfico com a quantidade de nomes por chave. O gráfico também pode ser acessado pelo link: [https://docs.google.com/spreadsheets/d/1vnorbvdnW4-jTrNqfD-rT6Ze4RUUp\\_Wo2dGVjeMAC\\_4Q/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1vnorbvdnW4-jTrNqfD-rT6Ze4RUUp_Wo2dGVjeMAC_4Q/edit?usp=sharing).

## Quantidade de Nomes por Chave



A coluna da esquerda representa a quantidade de nomes por chave, enquanto a linha horizontal abaixo das barras representa o número das chaves hash.