Instituto Federal de Educação, Ciência e Tecnologia Catarinense - Campus Videira
Bacharelado em Ciência da Computação - 2ª fase
Disciplina: Estrutura de Dados I
Professor: Manasses Ribeiro
Aluno: Henrique Pieri de Lima

RELATÓRIO DO PROJETO FINAL - ESTRUTURA DE DADOS I

A ideia do projeto basicamente é criar uma lista encadeada dupla (chamada de *ListaComSublista*) que em cada nodo contém outra lista encadeada dupla, uma sublista. Em cada nodo da lista "maior" (*ListaComSublistas*) contém o seu respectivo hash e a sublista, esta última possui nodos com os nomes inseridos. Ou seja, é um **hashing com encadeamento**.

Para a sublista foi utilizada a estrutura da lista encadeada dupla desenvolvida no decorrer do semestre, modificando o tipo de dado dos nodos para char* (string dinâmica). A lista maior foi feita com base na sublista, modificando os nodos para conterem o hash de cada um e a sublista citada, consequentemente alterando a lista para os novos tipos de nodos.

Então, o código vai criar as listas (formando todos os nodos da lista maior e suas respectivas sublistas, com base na quantidade de hashes determinada: **53**), ler o arquivo de texto com os nomes, separá-los em cada hash (inserindo na sublista do hash), ordenar cada sublista em ordem alfabética e imprimir outro arquivo de texto mostrando a quantidade de nomes em cada hash e os nomes separados e ordenados.

A função de hash utilizada foi um **hash modular**, que percorre cada caractere do nome, multiplicando o número primo 13 pelo hash que está sendo calculado (mudando a cada repetição), somando com o código ASCII do caractere e então atribuindo ao hash o resto do cálculo citado dividido pela quantidade de hashes:

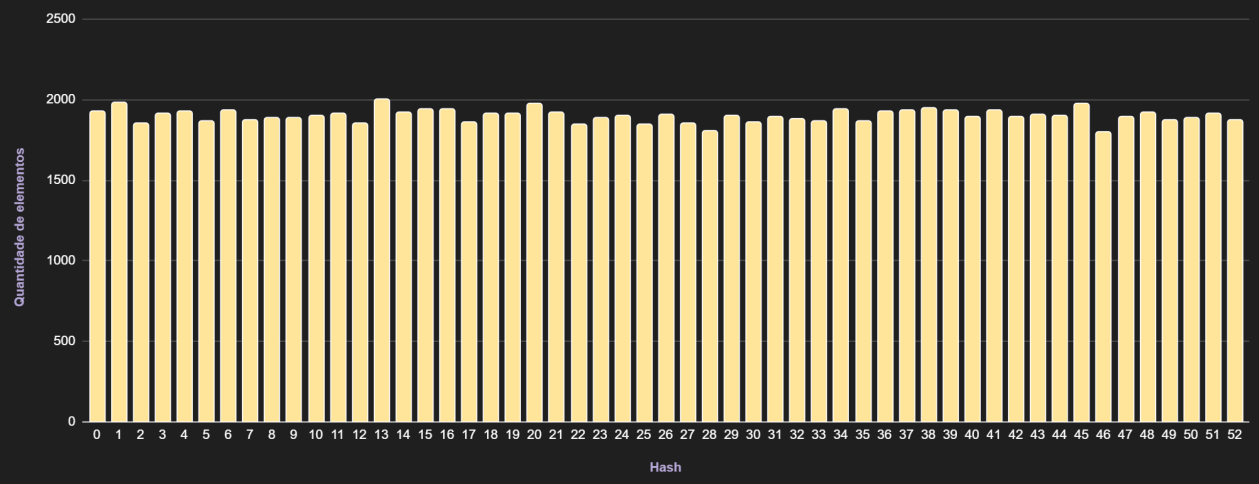
```c
int calculaHash(char* dado) {
    int hash = 0;
    for (int i = 0; i < strlen(dado); i++) {
        hash = (13 * hash + dado[i]) % M;
    }
    return hash;
}
```

A cada linha lida do arquivo de nomes, é chamada a função *inserirElementoHash*, que calcula o hash do nome, procura em qual nodo da lista maior

está o hash e insere o nome pelo *tail* da sublista deste nodo. Assim, é feito o **tratamento de colisões**, considerando que o espaço de cada lista não está delimitado, pois utiliza listas dinâmicas (está limitado apenas a memória física do computador em que o código é executado, mas teoricamente não existe um limite pré-determinado).

Após inserir cada nome em um hash, esses hashes serão ordenados pelo **quicksort**, sendo utilizado o método de partição de **Lomuto**, que tem como pivô o último elemento da lista, ou seja, o *tail* (é percorrido cada nodo da lista maior e ordenado sua respectiva sublista).

Por fim, é gerado o arquivo de texto final citado anteriormente e desalocado a memória das listas.



Gerando o **histograma** de frequência das chaves da tabela hash, foi obtido uma média de **1901,679245** e um desvio padrão de **40,36886347**. Analisando estes dados, foi parcialmente alcançada a hipótese do **hashing uniforme**, considerando que há um espalhamento satisfatório dos dados nos hashes.