Estruturação de arquivo utilizando Tabela Hash

Vinicius Gazolla Boneto

Instituto Federal Catarinense (IFC) - Campus Videira

Rodovia SC 135, km 125, S/n – Campo Experimental, Videira – SC, 89560-000

vineboneto@gmail.com

Introdução

O presente texto tem como objetivo descrever a lógica referente ao atual código. Onde este tem como proposta estruturar um arquivo que contém 100,788 nomes que foram cadastrados em determinado ano. Para realizar sua implementação foi utilizado as seguintes estruturas de dados: tabela hash, lista encadeada dupla, o método de ordenação quick sort, além de métodos relacionados a consulta, inserção e remoção de elementos.

Trabalho elaborado para disciplina de estrutura de dados I do curso ciência da computação com objetivo de aplicar o conteúdo abordado durante o semestre e obtenção a nota.

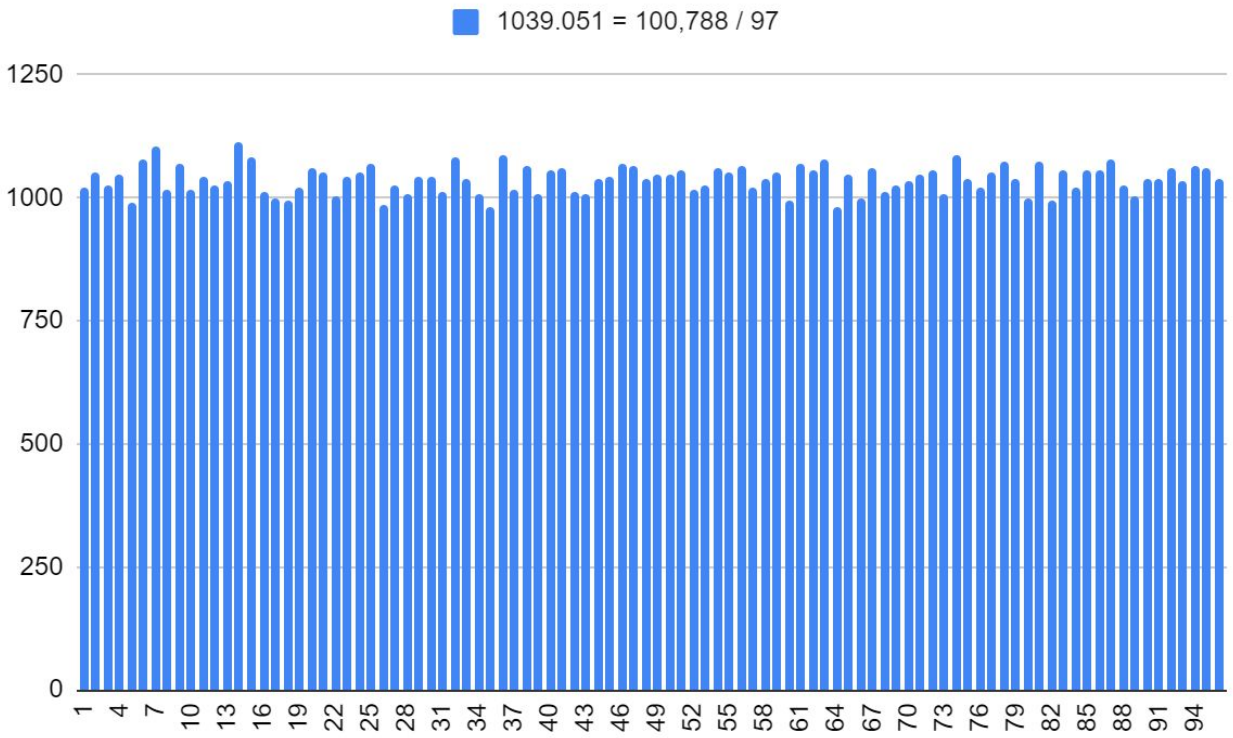
Metodologia

Para estruturar o determinado arquivo compreende-se que um dos maiores obstáculos é a sua grande quantidade de dados, onde está pode dificultar na realização de testes e afins. Para isso foi utilizado um arquivo similar com uma quantidade inferior que possuía apenas 100 nomes. As estrutura de dados utilizadas foram: tabela hash, lista encadeada dupla e quick sort.

Para representar a tabela hash foi utilizada um tipo abstrato de dado onde este continha um outro tipo abstrato de dado, uma lista encadeada dupla.

Na presente tabela hash foi determinado que as suas chaves seriam um número inteiro e positivo. Dessa forma é possível transformar a tabela hash em um vetor de tamanho M, onde M representa a quantidade total de chaves e o index do vetor representa a respectiva chave sendo lida, sendo assim, M tem que ser preferencialmente ímpar e primo, pois possibilita um espalhamento melhor dos valores lidos entre as chaves da tabela hash. Nesse caso devido ao fato do arquivo possuir

mais de 100,000 nomes, foi definido M = 97, resultando em aproximadamente 1039,051 nomes por chave como mostra no presente histograma.



Função Hashing

Na tabela hash a função *hashing* é de vital importância, pois ela é responsável por determinar em que chave o valor será inserido. Devido ao fato da chave ser um número inteiro e positivo, optou-se por utilizar uma função hashing modular, onde está retorna o resto da divisão por M. Como é demonstrado na equação matemática.

$$h = \left(\sum_{h=0}^{L} (h \cdot 31 + i) \ mod \ M \right)$$

$h = variavei\ acumuladora$

$M = quantidade\ de\ chaves$

$L = tamanho\ total\ do\ vetor\ de\ caracters\ lidos\ da\ linha$

$i = letra\ lida\ e\ convertida\ para\ tabela\ ASCII$

Dessa forma a função *hashing* realiza a somatória de todos os caracteres presentes no nome o que resultará em um número randômico melhor e o módulo por M garante que o resultado não ultrapassará a quantidade total de chaves determinadas.

Tratamento de colisões

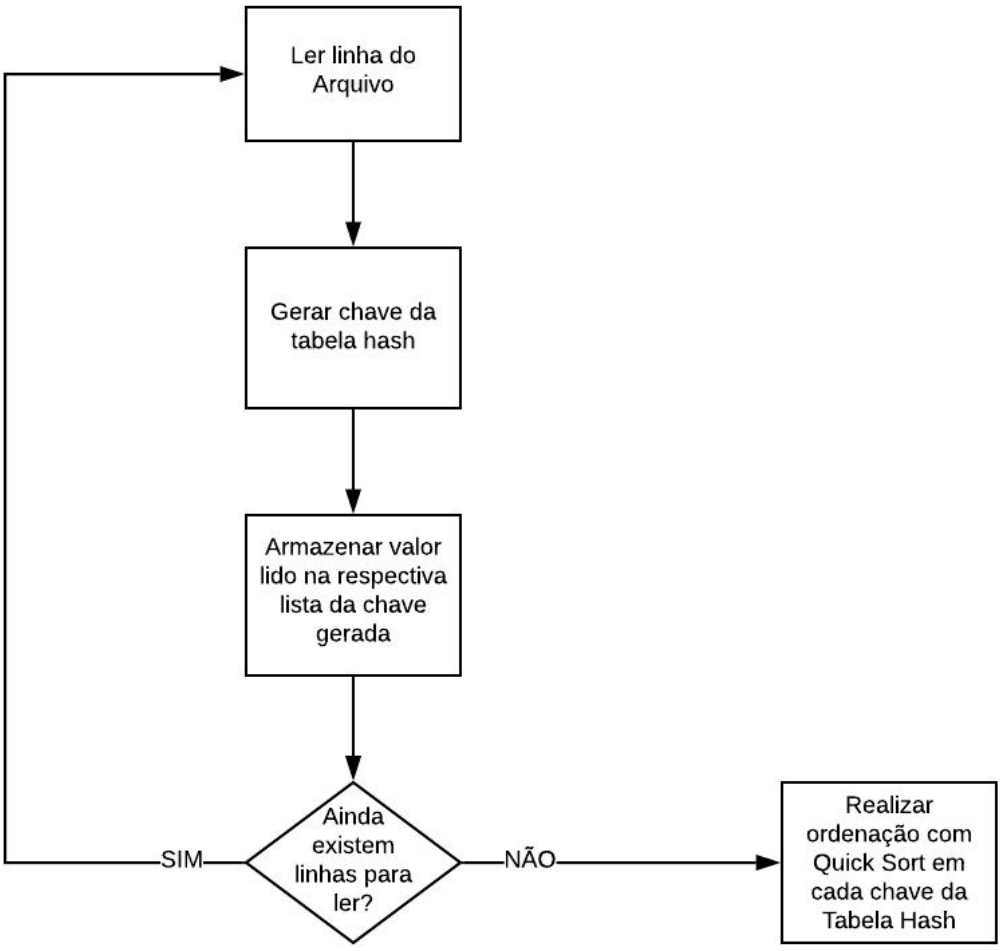
Para tratar as colisões que poderiam ocorrer na tabela hash utilizou-se a técnica hashing com encadeamento. A mesma caracteriza-se por implementar uma lista encadeada seja ela simples ou dupla em cada chave da tabela hash, dessa forma a mesma não irá possuir um limitador lógico como tamanho, podendo ser inserido quantos elemento forem necessários em cada chave de forma dinámica.

Dessa forma é possível concluir que a código hash gerado com a função hashing gerou um espalhamento eficiente como demonstrado no histograma e a implementação da lista encadeada dupla resolveu todos os problemas relacionados a possíveis colisões que poderiam ocorrer.

Ordenação

Após ocorrer a hashing dos nomes lidos do arquivo foi realizada a ordenação de cada respectiva chave da tabela hash em ordem alfabética, a estrutura de dados utilizada foi o Quicksort com lista encadeada dupla, devido ao fato desta estrutura apresentar um dos melhores desempenhos na ordenação de dados, por outro lado devido ao fato da alta utilização de recursividade onde a mesma utiliza de muita memória, foi estabelecido que cada chave iria possuir aproximadamente 1000 elemento como já explicado, para demandar de pilhas menores. Dessa forma a quantidade passos realizadas no sistema foi de aproximadamente 3000 por chave, conforme sua complexidade que é definida por $n\,Log n$. Onde n representa o número total de elementos.

Segue Fluxograma explicando a lógica do presente código.



Conclusão

O código conseguiu abordar todos os conteúdos abordados durante o semestre.

A tabela hash se mostrou eficiente em separar um arquivo grande em grupos de arquivos menores que possuem algo em comum, sendo ele uma chave única que é determinada pela função de espalhamento. Separar um arquivo grande em grupos menores dá uma grande vantagem na hora de se realizar pesquisas e ordenações, além de diminuir a complexidade do tempo que será realizada em cada um dos processos.

Com o fim do trabalho realizado, pode-se concluir que o mesmo contribuiu para o meu desenvolvimento na área de ciência da computação, onde se obteve um conhecimento avantajado sobre estrutura de dados, como trabalhar com elas e onde aplicá-las. O que será de grande importância em diversos ramos da computação.