Instituto Federal Catarinense - Campus Videira

Turma - Ciência da computação – 3º semestre 2021

Aluno: Pedro Henrique Fagundes

Trabalho Final-Estrutura de Dados.

Implementação de uma Tabela de Espalhamento (Tabela Hash)

Este trabalho procura implementar uma tabela de espalhamento, mais conhecido como tabela hash, feita especificamente com listas duplamente encadeadas em linguagem C, onde nela será manipulada 100.788 dados.

Oque é uma tabela hash? Resumidamente é uma estrutura de dados que associa várias “chaves” de pesquisa a valores, seu objetivo é a partir de uma chave fazer uma busca rápida e obter um dado/valor desejado. As tabelas hash são tipicamente utilizadas para grandes volumes de dados.

O Algoritmo

Como dito o algoritmo foi implementado em C, usando programação estruturada. Sendo feita com listas duplamente encadeadas, foi necessário criar a estrutura base para que as listas executassem, sendo assim, foi criada uma lista que guardava todas as chaves, e a partir dessas chaves, outras listas onde ficariam os dados, desse jeito tendo uma tabela encadeada onde um elemento apontará para o outro com seus respectivos ponteiros.

Além disso esses 100 mil nomes serão lidos de um arquivo de texto através de uma função FILE, tendo a opção de adicionar ou remover nomes se desejar.

Funções hash

Para que a tabela hash funciona como desejado, é necessários alguns cálculos que vão deixar a tabela mais harmônica, assim ganhando tempo de execução, pesquisa. Na tabela está sendo adicionado nomes, e na programação cada letra tem um valor único chamado de ASCII, partindo disso, para saber em que chave cada nome irá ser adicionado, é feito uma conta.

mod = (31 * mod + (int) nome [contador] % QuantidadeDeChaves;

A partir de uma estrutura de repetição, cada letra do nome é transformada em seu valor ASCII e são somadas entre si, essa soma é divida pela quantidade de chaves, o resultado será o mod dessa divisão.

Tratamento de colisão

A tabela hash sofre de um problema inevitável chamado colisão, onde dois dados vão para a mesma chave, oque se pode fazer é tratar esse problema, e um jeito é usando lista encadeadas ou duplamente encadeadas, onde se houver colisão a lista automaticamente aumenta e adiciona o novo dado.

Um ponto importante que deve ser analisado é a quantidade de dados em cada chave, por isso é necessário analisar a quantidade de dados que vão ser inseridos e a quantidade de chaves que vai ter. A comunidade cientifica diz que por um motivo desconhecido um número primo de quantidade de chaves tem melhor desempenho em deixar a tabela hash uniforme, mas nos testes em laboratório a melhor quantidade de chaves foi 100 em questão com a quantidade de nomes. Resultados são mostrados nas tabelas em “EXPERIÊNCIAS”, mais a frente.

Algoritmo de ordenação

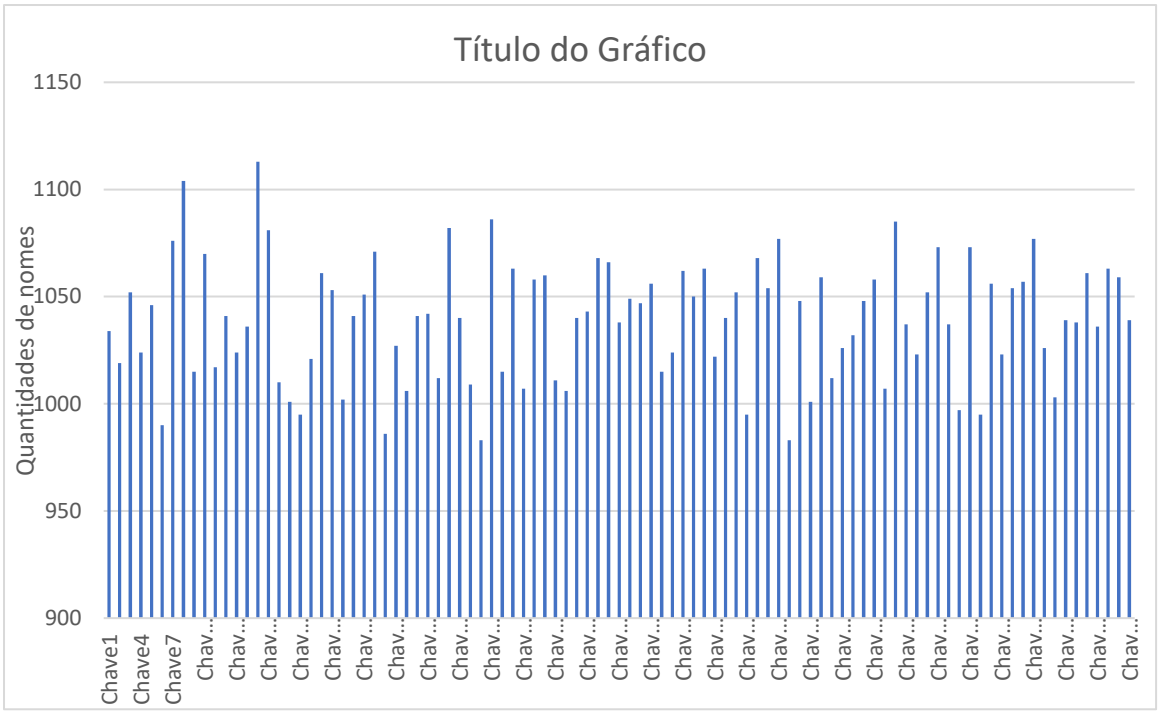
Logo após a inserção, cada lista de nomes é ordenada com a estrutura de ordenação quicksort. Um algoritmo de ordenação como o próprio nome já diz, lê uma quantidade x de dados e os ordena de uma respectiva forma, nesse caso que os dados são nomes, será ordenado em ordem alfabética, existe varios algoritimos de ordenação, como “bubble sort”, “Tim sort”, “Insertion sort”, sendo um melhor que o outro importando a quantidade e o tipo de dado. Nesse caso usamos quicksort.

Primeiramente o programador deve escolher um dado da lista para ser o pivô, dele o algoritmo reorganiza os dados em duas listas, a lista a esquerda do pivô fica com os dados menores que o pivô, e a lista da direita fica com os dados maiores que o pivô. Com a tabela divida e parcialmente ordenada, fica mais facil de ordenar as duas pares separadamente, e com isso a lista inteira fica ordenada, desse jeito a pesquisa à algum nome fica maia rápida, sendo o pior caso $\Theta(n_2)$.

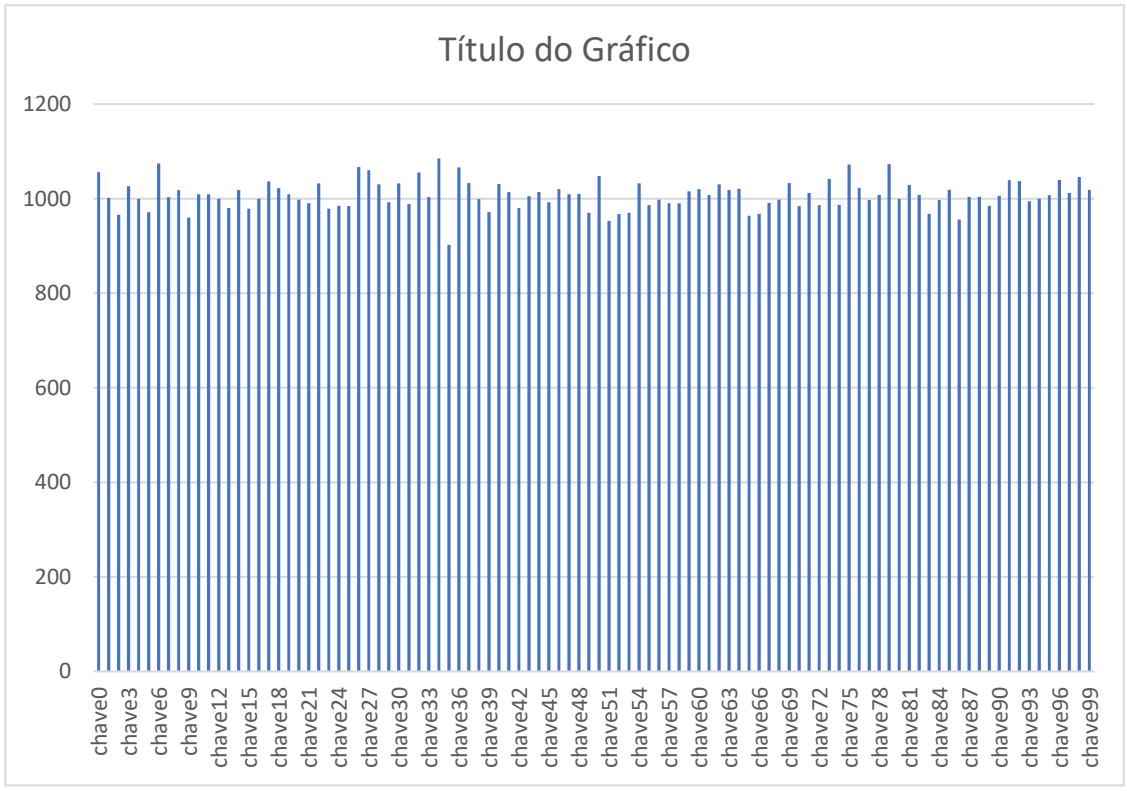
EXPERIÊNCIAS

Resultados da tabela hash:

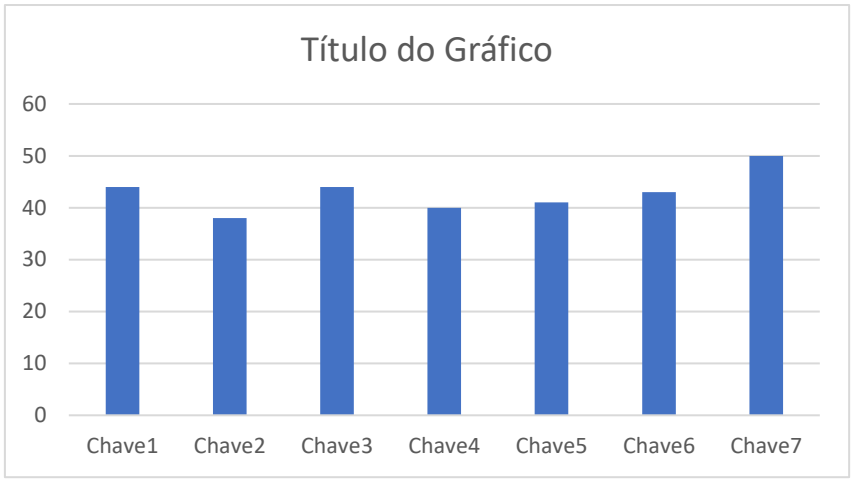
Teste com 97 chaves e 100.788 nomes:



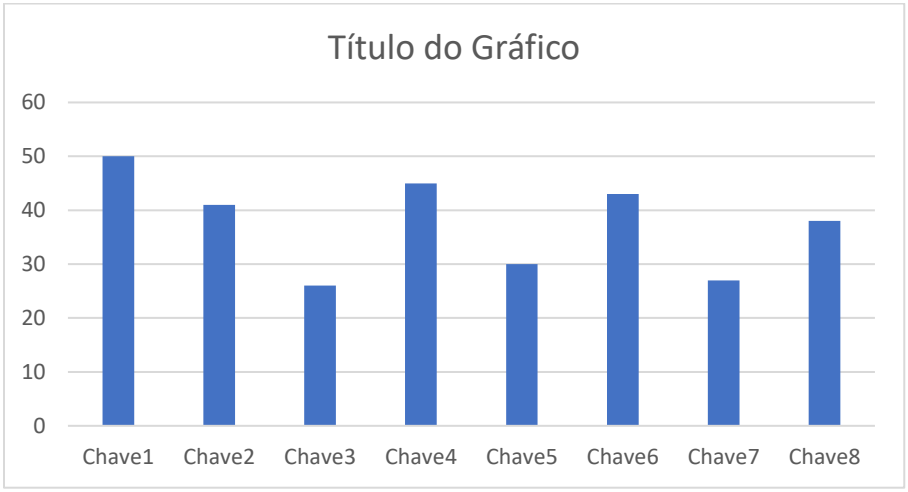
Teste com 100 chaves e 100.788 nomes:



Teste com 7 chaves e 100 nomes:



Teste com 8 chaves e 100 nomes



Link para o algoritmo completo:

➔ **https://github.com/Pedrohfzip/TrabalhoFinalED/tree/main**

Referências

https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/st-hash.html

https://pt.khanacademy.org/computing/computer-science/algorithms/quicksort/a/overview-of-quicksort

https://pt.wikipedia.org/wiki/Ficheiro:Sorting_quicksort_anim.gif

https://pt.wikipedia.org/wiki/Tabela_de_dispers%C3%A3o