UNIVERSIDADE FEDERAL DO PIAUÍ - CAMPUS SHNB

SISTEMAS DE INFORMAÇÃO

Estruturas de Dados II

Artur Pereira da Silva

Trabalho Prático

1 Introdução

A principal tarefa do hashing é distribuir as entradas de maneira uniforme em uma matriz, desse modo, cada elemento recebe uma chave convertida, desse modo é agilizado a busca de um dado. Para a implementação do algoritmo hashing, devem ser seguidas duas etapas principais:

1. Um elemento é convertido em inteiro usando a estrutura de dados hash. Tal elemento, poderá ser usado como índice para armazenar o elemento original que cai na tabela;
2. O elemento deve ser armazenado na tabela hash, onde poderá ser recuperado rapidamente usando a tabela Hash.

O hash é independente do tamanho matriz, onde, um elemento é convertido em inteiro usando a estrutura de dados hash. Tal elemento, poderá ser usado como índice para armazenar o elemento original que cai na tabela. Nesse contexto, a principal tarefa das técnicas de hash é espalhar os elementos de forma que tais elementos sejam rapidamente encontrados, independente dos elementos vazios na tabela, pois, apesar de haver mais gasto na memória, aumenta a eficiência dos algoritmos de busca.

Para o obtenção de bons resultados numa função de hash, é necessário desenvolver a função de acordo com os seguintes requisitos básicos:

1. Deve ser fácil de calcular e não deve se tornar um algoritmo em si;
2. Deve fornecer uma distribuição uniforme na tabela de hash e não deve resultar em clustering;
3. Deve-se evitar colisões quando os pares de elementos são mapeados para o mesmo valor de hash.

Desse modo, para calcular uma função hash é necessário efetivar a busca por meio de operações aritméticas que transformam a chave em endereços numa tabela. Numa função hashing é importante estimar o valor da função de transformação, a qual transforma a chave de pesquisa em um endereço da tabela.

Contudo nesse tipo de operações há alta probabilidade de ocorrer colisões. As colisões ocorrem quando elementos cujo valor da função de hash é o mesmo. Isso pode ocorrer, ainda que se obtenha uma

função de transformação que distribua os registros de transformação de maneira uniforme entre as entradas da tabela. Tendo em vista que em qualquer função de transformação podem ocorrer colisões, é necessário desenvolver um método que evite tais colisões.

Assim, será desenvolvido o cálculo do resultado do espalhamento de dados (ou hashing) em uma tabela de 101 compartimentos, onde, chaves dos compartimentos são strings com comprimento no máximo 15 letras.

2 Solução Proposta

Para o desenvolvimento da estrutura de dados hash será utilizada a linguagem programação C, no qual foram desenvolvidas funções que cumprem o objetivo do trabalho proposto. As principais operações para o desenvolvimento do trabalho são:

1. Buscar o índice do elemento definido pela chave: A busca foi implementada com base na definição dada pela chave;
2. Inserir uma nova chave: Para inserir uma chave foi definido a função int Hash(char * Chave), no qual foi determinado uma célula para o registro da chave;
3. Excluir uma chave da tabela: Marcando a posição na tabela como vazia, foram ignoradas chaves inexistentes na tabela.

   Segue abaixo os protótipos das principais funções implementadas e seus respectivos objetivos:

- int calculaHash(char * Chave): Nesta função é feito o cáculo da hash;
- int Colisoes(int mediaDeAcesso, hashTable * tabela, cahr * Chave): Nesta função é verificado se uma chave já possui ou nõa um registro a fim de evitar colisões;
- void Inserir(int mediaDeAcesso, hashTable * tabela, *Chave): A partir da função calculaHash(*Chave) foi determinado o local para a inserção da chave na tabela;
- void Remover(int mediaDeAcesso, hashTable * tabela, char * Chave): Nesta função é realizada a remoção de uma chave na tabela;
- void Listar(FILE * fileSaida, hashTable * tabela, const int Tamanho): nesta função é buscado uma chave na tabela e criado um arquivo referentes às chaves buscadas.

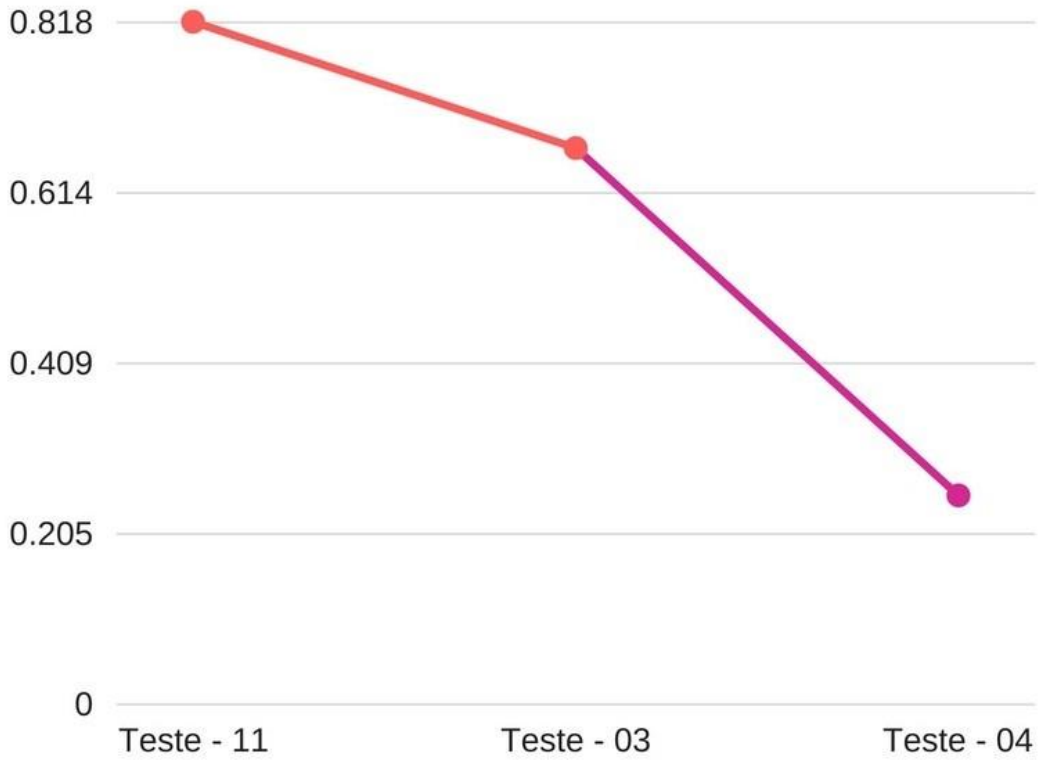
3 Análise de Desempenho

Foram desenvolvidos testes para a análise de desempenho, mediante a notação big O. Desconsiderando as colisões, a implementação da hash busca uma função de dispersão que seja de complexidade O(1). No pior caso sua complexidade é de O(n).

A maior vantagem em usar tabela *hash* é por causa do desempenho, pois, o tempo de busca na tabela *hash* é praticamente independente do número de chaves armazenadas na tabela, ou seja, tem complexidade temporal O(1) .

Para o teste foi feita contagem de toda função que faz determinada busca na tabela hash. Para cada acesso que a chave fosse encontrada na primeira tentativa, é somado 1 ao custo total, ou seja, O(1). Em outros casos, como por exemplo, na inserção, onde pode ocorrer colisões é contado o total de vezes que é necessário contar a quantidade de vezes que é recalculado a hash para que se possa encontrar um espaço livre para a inserção de um novo elemento.

O gráfico abaixo mostra a média no custo de acesso de todas as funções utilizadas, onde, o cálculo da média se dá pelo total de acessos divididos pelo numero de registros. No eixo Y do gráfico é dado a média do acesso em relação à quantidade de registros do eixo X.



4 Conclusão

O desenvolvimento deste trabalho se deu a partir de pesquisas, aulas do professor e exemplos disponibilizados na web, no qual foi possível desenvolver a atividade proposta de modo que resolva o problema. Assim, foi compreendido e desenvolvido a estrutura de dados hash, que assume a importante tarefa de espalhar dados, a fim de melhorar a busca de dados. No processo de desenvolvimento da tabela

hash foi na linguagem C por facilitar a manipulação de strings, acesso a memória e manipulação de estruturas de dados heterogêneas, facilitando assim, na construção.