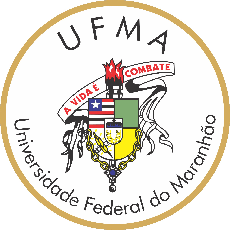
****

**UNIVERSIDADE FEDERAL DO MARANHÃO**

**Departamento de Informática**

**ESTRUTURA DE DADOS II**

**Atividade Prática 2**

**Membros da equipe:**

Felipe Rogerio Silva Teles

Rodrigo Pontes Oliveira Lima

**Professor da disciplina:**

João Dallyson Sousa De Almeida

São Luís do Maranhão

2022

**Tabela Hash**

1. **Introdução.**

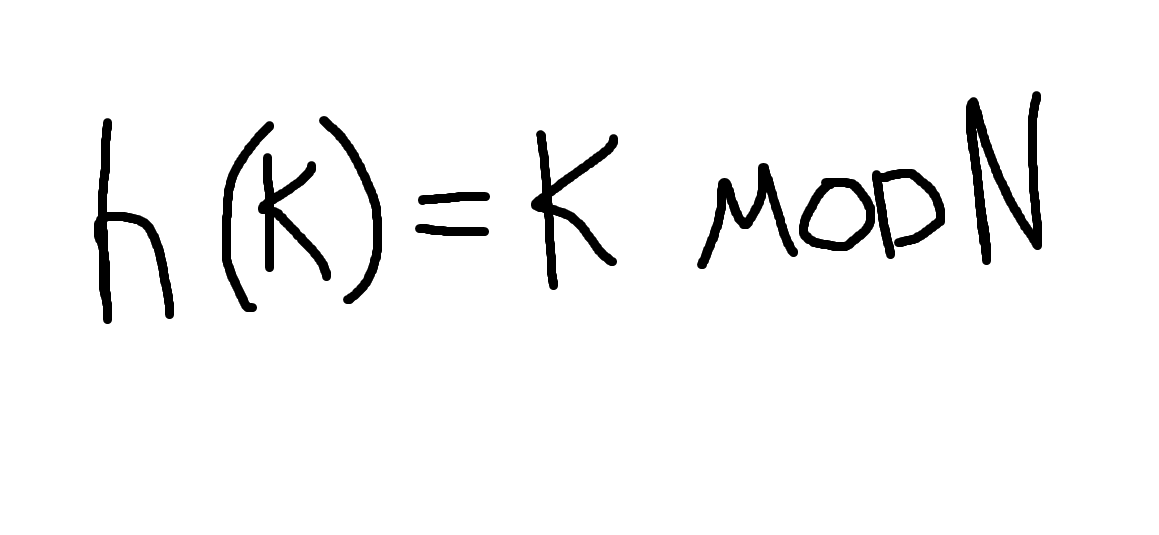
Uma tabela hash é uma estrutura de dados que visa diminuir o tempo de busca de um algoritmo. A chave é adicionada em um vetor e existe uma função que transforma essa chave em um valor inteiro. Esta função chama-se função de espalhamento ou função *Hash*.Basicamente atribuímos um inteiro a tabela que será a quantidade de termos que terá no array. Chamamos o inteiro de operador e o array de vetor por questões didáticas. Para a abstração da aplicação utilizamos o generic para permitir armazenar qualquer tipo de classe de no vetor e nas estruturas implementadas.

1. **Função Hash.**

Função responsável por transformar o valor de qualquer tipo que será inserido no vetor em uma chave do tipo inteiro. A chave criada será o índice no array reduzindo o custo da função de busca para O(1) no melhor caso.

Caso haja de durante a inserção já possuir um valor no índice então significa que ocorreu uma colisão.

A função Hash que utilizamos neste trabalho foi o resto da divisão da chave a ser inserida, que no caso é a matrícula do aluno, pela quantidade de termos da tabela.



1. **Colisão.**

Para o tratamento de colisões podem ser realizadas diversas táticas e elas variam entre vários tipo, neste trabalho usamos duas técnicas, sendo elas:

* 1. **Encadeamento Aberto**

Armazenando o valor colidido na próxima posição disponível dentro do array buscando em cada casa uma posição disponível.

* 1. **Encadeamento Externo**

O valor será armazenado em uma estrutura externa ao array principal.

* + 1. **Listas encadeada**
    2. **Árvore AVL**
    3. **Árvore Rubro-Negra**

1. **Resultados**

Para realizar a inserção de 100.000 itens na tabela Hash, primeiro inicializamos uma tabela com 100.000 posições. Após ser inicializada a tabela, realizamos a inserção de cada termo utilizando uma função chamada cria Aluno. Este método cria um objeto que chamamos de Aluno, que é basicamente uma classe com um inteiro para matrícula, uma string para o nome e uma lista de strings com as disciplinas matriculadas.

A inserção é feita de forma linear até ocorrer uma colisão, ou seja, ocorre em tempo de execução ***O(1)***. Após ocorrer uma colisão o termo pode ser inserido com o encadeamento aberto inserindo o novo valor em uma próxima posição disponível no array, neste caso o tempo de execução seria ***O(h)***sendo ***h*** o número de posições ocupadas até a próxima vazia. A colisão também pode ser tratada utilizando o encadeamento externo. Neste caso pode ocorrer de três formas e valor será inserido ou em uma lista encadeada ou em uma árvore AVL ou em uma árvore Rubro-Negra, o desempenho depende da posição a qual o valor ficará na estrutura escolhida. Utilizando como exemplo a inserção na lista encadeada: primeiro o tempo será combinado com o ***O(1)***, pois ele vai tentar inserir na primeira posição possível e depois vai inserir em uma lista nesta posição, em seguida ele vai inserir na lista que também tem o tempo de complexidade linear. Porém caso a inserção seja feita em uma das árvores o tempo seria de complexidade logarítmica.

Paralelo a isto a busca ocorre de forma similar, isto é, caso esteja na primeira posição possível terá o tempo ***O(1)***, entretanto caso não esteja primeiro ele vai buscar em uma das posições do array e caso esteja na última posição, configurando assim o pior caso, o tempo de busca vai ser ***O(100.000 - pos)***, sendo ***pos*** a primeira posição buscada. Caso o termo buscado esteja em uma estrutura fora do array então a busca será realizada na seguinte ordem: busca na lista encadeada, busca na árvore AVL e busca na árvore rubro-negra. Caso o termo esteja na lista encadeada então o tempo de busca será O(n) no pior caso, pois terá de percorrer cada termo da lista buscando a chave, somado ao tempo gasto para buscar no vetor. O tempo de busca em cada uma das árvores será ***O(log n)*** no pior caso, entretanto tem de se ater ao fato de que já foram realizadas tentativas de busca e primeiro será buscado na árvore AVL, então caso o termo buscado esteja na última posição possível da árvore rubro-negra, esta seria a pior posição possível e o tempo de execução será a soma de todos os tempos, portanto seria o maior valor concluindo assim que seria de complexidade ***O(n)***.