Relatório de funções hash

As funções hash são responsáveis por gerar um código de busca ágil para encontrar um valor ou objeto em uma **tabela hash**.

Uma tabela hash é essencialmente um array de listas variáveis, das quais para definir quais valores serão colocados em cada índice do array é a **função hash**.

Uma função hash pode ser a mais simples, como retornar o resto da divisão da string pelo tamanho do array, como também pode ser bastante complexa, envolvendo até algoritmos de criptografia.

Para este trabalho foram comparadas 2 funções hash para inserir uma sequência de nomes em uma tabela:

- retorno do resto da divisão entre o tamanho da string menos 1 e a capacidade da tabela.
- para cada caractere, busca o valor **char** dele e adiciona numa variável. Então retorna o resto da divisão entre essa variável e a capacidade da tabela.

Resultados da primeira tabela

A primeira tabela ficou com a seguinte formatação:

```
Distribuicao das chaves por posicao:
Posicao[0] - 0 elementos
Posicao[1] - 7 elementos
Posicao[2] - 125 elementos
Posicao[3] - 534 elementos
Posicao[4] - 1210 elementos
Posicao[5] - 1250 elementos
Posicao[6] - 942 elementos
Posicao[7] - 545 elementos
Posicao[8] - 245 elementos
Posicao[9] - 83 elementos
Posicao[10] - 16 elementos
Posicao[11] - 6 elementos
Posicao[12] - 2 elementos
Posicao[13] - 1 elementos
Posicao[14] - 1 elementos
Posicao[15] - 0 elementos
Posicao[16] - 0 elementos
Posicao[17] - 0 elementos
Posicao[18] - 0 elementos
Posicao[19] - 0 elementos
Posicao[20] - 0 elementos
Posicao[21] - 0 elementos
Posicao[22] - 0 elementos
Posicao[23] - 0 elementos
Posicao[24] - 0 elementos
Posicao[25] - 0 elementos
Posicao[26] - 0 elementos
Posicao[27] - 0 elementos
Posicao[28] - 0 elementos
Posicao[29] - 0 elementos
Posicao[30] - 0 elementos
Posicao[31] - 0 elementos
```

Como é uma função baseada apenas no tamanho dos nomes inseridos, ocorreu que quase todos os nomes ficaram nas posições de índice 3, 4, 5, 6 e 7, por mais que a tabela tivesse 32 índices possíveis.

Esse não é o melhor aproveitamento dessa estrutura, visto que não há uma distribuição nos elementos do array que tende à uniformidade. Essa tabela possui uma variação da quantidade de elementos que vai de 0 a 1250, sendo que metade dos índices possui 0 elementos e 3 índices possuem mais de 900 elementos.

Resultados da segunda tabela

A segunda tabela ficou com a seguinte formatação

```
Distribuicao das chaves por posicao:
Posicao[0] - 159 elementos
Posicao[1] - 171 elementos
Posicao[2] - 159 elementos
Posicao[3] - 146 elementos
Posicao[4] - 145 elementos
Posicao[5] - 149 elementos
Posicao[6] - 166 elementos
Posicao[7] - 158 elementos
Posicao[8] - 164 elementos
Posicao[9] - 144 elementos
Posicao[10] - 153 elementos
Posicao[11] - 146 elementos
Posicao[12] - 174 elementos
Posicao[13] - 149 elementos
Posicao[14] - 182 elementos
Posicao[15] - 146 elementos
Posicao[16] - 139 elementos
Posicao[17] - 145 elementos
Posicao[18] - 162 elementos
Posicao[19] - 160 elementos
Posicao[20] - 148 elementos
Posicao[21] - 150 elementos
Posicao[22] - 157 elementos
Posicao[23] - 158 elementos
Posicao[24] - 146 elementos
Posicao[25] - 149 elementos
Posicao[26] - 156 elementos
Posicao[27] - 147 elementos
Posicao[28] - 159 elementos
Posicao[29] - 156 elementos
Posicao[30] - 176 elementos
Posicao[31] - 148 elementos
```

Como é possível notar, há uma distribuição quase uniforme da quantidade de elementos em cada índice do array, com a lista com menos elementos contendo apenas 139 e a maior lista contendo 182 elementos.

Esse é um exemplo de bom proveito da tabela hash, visto que para buscar uma string em uma estrutura de 5000 elementos é necessário fazer 182 iterações na pior das hipóteses.

Clusterização

Nota-se que houve uma clusterização na tabela 1, visto que todos os elementos estão contidos em metade dos índices do array. Isso caracteriza uma clusterização, que pode ser notada quando calculamos o **fator de carga** da tabela.

Fator de carga

O fator de carga é essencialmente uma média de quantos elementos estão em cada índice da tabela. Como as duas tabelas possuem o mesmo tamanho de array e a mesma quantia de elementos, o fator de carga acabou por ser o mesmo: 155,2188.

Porém, se calcularmos o fator de carga considerando apenas a quantia de colunas que possuem algum elemento, os resultados desse experimento mudam:

A tabela 2 se mantém, visto que todas as colunas possuem elementos.

A tabela 1 possui apenas 14 colunas que possuem elementos, o que faz com que o cálculo do fator de carga seja:

4967/14 = 354,785714286.

Ou seja, um fator de carga que é mais que o dobro da segunda tabela.

Isso mostra que, para uma busca nessa tabela, a quantidade de passos necessários para encontrar o elemento tende a ser muito maior que para a outra tabela, pois é como se essa tabela fosse menor que a outra.

Colisões

Em uma tabela hash, é comum haver colisões, já que normalmente há mais elementos do que o número de colunas.

Contando que cada colisão seja quando há uma tentativa de inserir um elemento em uma coluna que já há elementos, o número de colisões é o seguinte:

```
Numero de Colisoes:
    Tabela 1: 4953
    Tabela 2: 4935
    Diferenca: 18
```

Nota-se que houve praticamente a mesma quantidade de colisões entre as duas tabelas, o que é esperado, já que são quase 5000 elementos em uma tabela com 32 listas. Porém há uma diferença de 18 colisões entre as tabelas.

Essa diferença ocorre porque na tabela 1 há 18 índices do array que não há elementos, logo, não há colisões.

Tempo de busca, inserção e remoção

```
Tempo de Insercao:
    Tabela 1: 19.331588 ms
    Tabela 2: 5.134927 ms

Tempo de Busca:
    Tabela 1: 1.423833 ms
    Tabela 2: 0.267316 ms

Tempo de Remocao:
    Tabela 1: 1.320067 ms
    Tabela 2: 0.317332 ms
```

Nota-se uma diferença no tempo de inserção, busca e remoção das duas tabelas.

Isso ocorre justamente por conta da clusterização, já que para inserir, buscar e remover um elemento da tabela 1, tende a ser necessário percorrer muito mais elementos de uma lista do que na tabela 2.

Comparação entre os dois métodos.

As comparações diretas são:

```
Numero de Colisoes:
    Tabela 1: 4953
    Tabela 2: 4935
    Diferenca: 18

Fator de Carga:
    Tabela 1: 155,2188
    Tabela 2: 155,2188

Tempo de Insercao:
    Tabela 1: 19.331588 ms
    Tabela 2: 5.134927 ms

Tempo de Busca:
    Tabela 1: 1.423833 ms
    Tabela 2: 0.267316 ms

Tempo de Remocao:
    Tabela 1: 1.320067 ms
    Tabela 2: 0.317332 ms
```

A primeira grande diferença está no tempo de inserção, em que a tabela 1 possui um tempo muito maior do que a tabela 2. Isso ocorre porque a inserção em uma lista encadeada precisa percorrer toda a lista, e como na tabela 1 a há clusterização, as listas ficaram muito grandes, de maneira que 3 delas representam mais de 50% dos elementos da estrutura.

Devido ao fato de as listas ficarem muito maiores, são necessários muito mais passos para se chegar ao elemento final da lista, aumentando drasticamente o tempo de inserção, remoção e busca.

Por outro lado, devido ao fato de a tabela 2 ter um algoritmo que permite distribuição mais uniforme, a quantidade de elementos da maior lista da estrutura acaba sendo muito menor que na tabela 1.

Assim, para se inserir um elemento novo são necessários muito menos passos em média do que para a tabela 1.

Também, o tempo de remoção da tabela 1 novamente é muito maior que o tempo da tabela 2. Isso ocorre pelo mesmo motivo da inserção, visto que é necessário muito mais iterações para remoção.

Aprendizados

Embora a tabela hash 2 possua um algoritmo de hash mais complexo que o da tabela 1, é um algoritmo que faz muito mais sentido para o caso, pois proporciona distribuição mais uniforme dos elementos.

Isso não significa que o algoritmo de hash da tabela 1 está errado, apenas que não é o mais apropriado para o caso de uso.

Uma distribuição mais uniforme da tabela hash é muito vantajosa, já que proporciona uma média de menor número de passos a serem seguidos para se chegar a um elemento.