**Universidade Federal do Rio Grande do Sul**
**Instituto de Informática**
**INF01124 - Classificação e Pesquisa de Dados - Turma B**
**Prof. Renan Maffei**

# Laboratório 4 – Tabelas Hash

**Alexandre Lima – 273164** **Entrega: 06/11/2018**

## 1. Tabelas Hash - Endereçamento Aberto

### 1.1 Implementação da Tabela Hash:

O código foi escrito na linguagem Python, e a tabela hash foi implementada como um array (lista) de elementos do tipo **tuple**, sendo cada tuple composto por duas strings: um nome artístico (chave) e um nome real (dado satélite).

A função hash criada utiliza o número π como uma constante base para a geração do endereço. A ideia de utilizar esse número veio de um episódio da série Elementary, onde Sherlock Holmes descobre que a senha de um cofre é gerada a partir da constante matemática (temporada 1, episódio 10).

O endereço é gerado pela função:

```
def  computeHash(chave):
    sPi = str(math.pi)*4    # Concatena a string da constante Pi 4 vezes para chegar aos 50 caracteres
    end = 0
    for a in range(0,len(chave)):   # a indo de 0 a len(chave)
        end += ord(chave[a]) * ord(sPi[a])  # Multiplica o caracter a da chave pelo dígito a de Pi
    return end % M  # Retorna o resto da divisão da soma das multiplicações por M
```

O método de tratamento de colisões escolhido foi o **duplo hashing**. Não foi necessário alterar a implementação da tabela, mas o tamanho escolhido para a tabela deve ser um número primo, pois os incrementos (i) e o tamanho da tabela (M) não devem ter nenhum divisor inteiro comum, exceto a unidade, e a função de realeatorização escolhida foi semelhante à função hash, trocando a constante π pelo número de Euler ($e$):

```
def recomputeHash(chave): # Função de realeatorização: semelhante à função hash, mas trocando Pi por e
    sE = str(math.e)*4
    end = 0
    for a in range(0,len(chave)):
        end += ord(chave[a]) * ord(sE[a])
    return end % M
```

As funções de inserção e pesquisa estão disponíveis e comentadas nos códigos anexos.

1.2 Testando a inserção de dados:

O trecho de código responsável por inicializar a tabela hash, ler e inserir os dados em uma lista de tuples auxiliar (não é a tabela hash) é mostrado abaixo:

```
# Inicializa tabela hash #
HashTable = [None]*M  # HashTable é uma lista de tuples

# Abre arquivo do dataset e insere dados na tabela de entrada #
arquivo = open("dataset.txt",'r',1,"utf-8") # Codificação UTF-8
tabela_dados_aux = arquivo.readlines()
arquivo.close()

# Adiciona os dados do arquivo na tabela_dados #
tabela_dados = []
for linha in tabela_dados_aux:
    linha = linha[:-1]  # Remove \n do final da string
    tuple_aux = linha.partition(';')    # Particiona a linha num tuple (chave, ';', dado)
    tuple_aux = (tuple_aux[0],tuple_aux[2]) # Remove o ';'
    tabela_dados.append(tuple_aux) # Adiciona à tabela_dados

nums_1_2 = [0, 0, 0]   # Lista com o número de colisões, máximo de colisões em uma única inserção e
# a taxa de ocupação da tabela
```

E o trecho de código responsável por inserir esses dados na tabela hash e listar a tabela no terminal, junto com a taxa de ocupação da tabela, número médio de colisões e número máximo de colisões em uma única inserção:

```
insercoes = 0
for element in tabela_dados:    # Insere os dados na HashTable
    insertHash(HashTable, element, nums_1_2)
    insercoes += 1

i = 0
for a in HashTable: # Printa a HashTable
    print(i, a)
    i += 1

print("\nTaxa de ocupação da tabela: " + str(nums_1_2[TAXA_OCUPACAO]) + "%")
print("Número médio de colisões: " + str(nums_1_2[NUM_COLISOES]/float(insercoes)))
print("Número máximo de colisões em uma única inserção: " + str(nums_1_2[MAX_COLISOES]) + "\n\n")
```

Em relação a inserção se dados, segue tabela com os números resultantes das execuções:

| M (tamanho da tabela) | Taxa de ocupação da tabela | Número médio de colisões | Máximo de colisões em uma única inserção |
|---|---|---|---|
| 199 | 75,38% | 0,94 | 13 |
| 499 | 30,06% | 0,18 | 5 |
| 997 | 15,05% | 0,087 | 3 |

1.3 Testando a pesquisa de dados:

O código responsável por pesquisar os dados do arquivo “queries.txt” foi implementado da seguinte forma:

```
# Limpa tabela de entrada e abre arquivo queries #
tabela_dados_aux.clear()
arquivo = open("queries.txt",'r',1,"utf-8") # Codificação UTF-8
tabela_dados_aux = arquivo.readlines()
arquivo.close()

for nome_artistico in tabela_dados_aux:
    nome_artistico = nome_artistico[:-1]  # Remove \n do final da string
    searchHash(HashTable, nome_artistico)
```

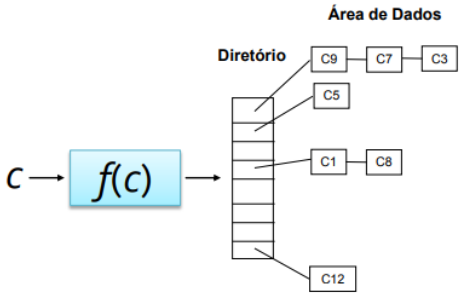
Semelhante ao código de inserção de dados, apresenta uma lista com os nomes artísticos e os nomes reais das pessoas listadas no arquivo “queries,txt”. O resultado da execução do programa é:

**Bob Dylan é Robert Allen Zimmerman**
**Freddie Mercury é Farrokh Bulsara**
**Chitãozinho é José Lima Sobrinho**
**Elton John é Reginald Kenneth Dwight**
**Lady Gaga é Stefani Joanne Angelina Germanotta**
**Leonardo é Emival Eterno Costa**
**Mano Lima não encontrado no dataset**
**Alicia Keys é Alicia Augello Cook**
**Raul Seixas é Raul Santos Seixas**
**Marilyn Manson é Brian Hugh Warner**
**Benito di Paula é Uday Vellozzo**
**Tina Turner é Annie Mae Bullock**
**Ringo Starr é Richard Starkey**
**Cazuza é Agenor de Miranda Araújo Neto**
**Lulu Santos é Luis Maurício Progana dos Santos**
**Mick Jagger é Michael Phillip Jagger**
**Ozzy Osbourne é John Michael Osbourne**
**Baitaca não encontrado no dataset**
**Cat Stevens é Steven Demetre Georgiou**
**Steven Tyler é Steven Victor Tallarico**

## 2. Tabelas Hash - Endereçamento Fechado

### 2.1 Implementação da Tabela Hash:

Na segunda parte do exercício, a hash foi implementada em um array (lista) de elementos, sendo cada elemento uma lista de tuples, onde os tuples são as estruturas que armazenam as chaves e os dados. Essa foi a implementação escolhida pois o método utilizado para resolução de conflitos foi a técnica de encadeamento (chaining):




Nessa implementação, os valores de M (tamanho da tabela) podem ser qualquer valor (não há a restrição de ser um número primo).

A técnica de encadeamento tornou o processo de inserção e pesquisa menos complexo se comparado com as funções de endereçamento aberto. Para inserção, somente inserir o elemento no final da lista correspondente ao seu endereço gerado pela chave, e para pesquisar, só percorrer a lista do endereço da chave dada. Ambas funções estão implementadas e comentadas nos códigos anexos.

A função hash permaneceu a mesma.

### 2.2 Testando a inserção de dados:

O código para inserir os dados é idêntico ao apresentado no método de endereçamento aberto, mudando somente a implementação da função insertHash.

A inserção de dados foi realizada gerando um endereço a partir da chave dada e inserindo na lista:

```
HashTable[end].append(element)
```

Neste caso, como não era necessário encontrar um novo endereço para tratar colisões, a ocupação da tabela ficou menor que o método de endereçamento

aberto, como apresenta a tabela abaixo com a taxa de ocupação da tabela e o número médio de colisões (não há número máximo de colisões em uma única inserção pois não ocorre mais de uma colisão). É apresentado também o número total de colisões:

| M (tamanho da tabela) | Taxa de ocupação da tabela | Número médio de colisões | Total de colisões |
|---|---|---|---|
| 200 | 51,0% | 0,32 | 48 |
| 500 | 26,40% | 0,12 | 18 |
| 1000 | 14,1% | 0,06 | 9 |

2.3 Testando a pesquisa de dados:

A chamada da função searchHash(...) e o resultado são idênticos aos apresentados no método de endereçamento aberto, mudando somente a implementação da função searchHash(...).