Funções Hash - Estrutura de Dados

Emannuel Levi de Assis Bezerra

Janeiro de 2025

1 Implementação das Funções de Hash

1.1 Somatório

Esse é de longe o algoritmo mais simples de implementar dos apresentados. Nele você recebe uma string, converte cada caractere para o seu respectivo código ASCII e então retorna a soma dos valores. Pode ser representado da seguinte forma:

```
def somatorio(string: str):
    return sum(map(ord, string))
```

1.2 Dispersão Polinomial

Esse método é um pouco mais complexo mas ainda assim tem uma simples implementação. Nele você recebe uma string, converte cada caractere para seu código ASCII e múltiplica por um numero pré-definido p elevado à iteração atual i. Pode ser representado da seguinte forma:

```
def dispersao_polinomial(string: str):
    return sum(ord(string[i])*(97**i) for i in range(len(string)))
```

1.3 Dispersão de Deslocamento

Esse é o mais compelxo dos apresentados, mesmo assim dá para representar ele de uma forma simples. Nele, você recebe uma string, converte cada caractere para o seu respectivo código ASCII e então faz um XOR nos valores deslocados em uma posição. Ele pode ser representado da seguinte forma:

```
def dispersao_de_deslocamento(string: str):
    return reduce(xor, map(ord, string)) ^ ord(string[0])
```

2 Implementação dos Métodos de Compressão

2.1 Divisão

Esse é o mais simples de todos, nele você apenas faz $h \mod n$ sendo h o número do Hash obtido pela função e n o tamanho da HashTable. Pode ser representado pelo simples algoritmo:

```
def divisao(k):
    return k%32
```

2.2 Dobra

Esse é o método mais complexo, nele você tem que separar grupos de dois números do Hash e ir somando com outros dois numeros até o numero se transformado numa versão reduzida. Segue implementação:

```
def dobra(k):
    arr = list(map(int, str(k)))
    while len(arr) > 2:
        if len(arr) == 3:
            arr = [(arr[1]+arr[2])%10, arr[0]]
            break
        num1 = (arr[0] + arr[3])%10
        num2 = (arr[1] + arr[2])%10
        arr = [num2, num1, *arr[4:]]

    return divisao(arr[0]*10 + arr[1]) if len(arr) > 1 else arr[0]
```

2.3 MAD (Multiplicação, Adição, Divisão)

Esse método é um pouco mais simples que o da dobra, nele você escolhe 3 números arbitrários e aplica soma multiplicação e divisão. Segue implementação:

```
def mad(k):
    return divisao((257*k+1)%1009)
```

3 Resultados

Os números de colisões obtidos para diferentes combinações de métodos de *hash* e compressão são apresentados na Tabela 1. Há de se reiterar que o resultado depende dos números árbitrários que foram escolhidos, podendo variar bastante.

Table 1: Número de colisões por método de hash e compressão

Método de Hash	Divisão	Dobra	MAD
Somatório	8	12	9
Dispersão Polinomial	8	12	14
Dispersão de Deslocamento	11	17	13

É perceptível então que o método do somatório aliado à divisão apresentou um resultado melhor no nosso caso.
Link para o código: https://github.com/dabzr/data-structures/blob/main/hash.py