INSTITUTO FEDERAL DE ALAGOAS – IFAL / CAMPUS MACEIÓ
Curso Bacharelado em Sistemas de Informação
Disciplina de Estruturas de Dados

Prof. MSc. Ricardo Nunes
Ricardo (arroba) ifal.edu.br

Hashing (Espalhamento ou Dispersão)

Objetivos

1. Compreender a ideia de hashing como uma técnica de busca
2. Compreender e implementar o tipo abstrato de dados map

Conteúdo

- Conceito de hashing
- Funções de hashing
- Estudo sobre como lidar com colisões
- TAD: Map (mapa)

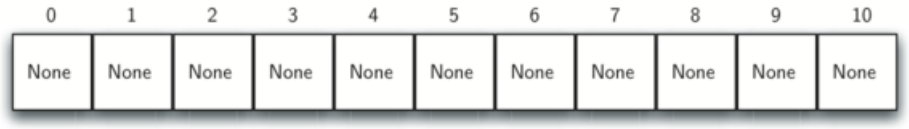
Hashing

- Busca sequencial: O(n)
- Busca binária: O(log n)
- Hashing: O(1)

▫ - Como?

Hashing

- Vetor (`list` em python) inicialmente com todas as posições vazias (None)



- função `hash`
- realiza o mapeamento entre o elemento e a posição que ele ocupará
- dado como entrada o valor, a função retorna a posição no intervalo dos slots (0,m-1)

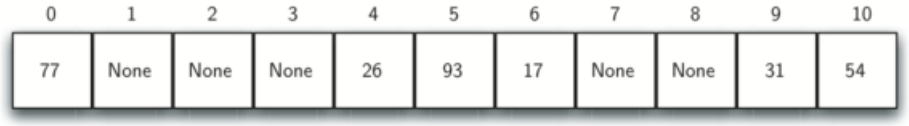
Hashing - Exemplo

- suponha os itens 54, 26, 93, 17, 77 e 31
- função hash: h(item)=item%11

Item	Valor do Hash
54	10
26	4
93	5
17	6
77	0
31	9

Hashing - Inserindo os valores

- Com o cálculo hash, insere os valores na lista



- fator de carga (FC)
- 6 dos 11 slots estão ocupados.
- FC = (número de itens)/(tamanho da tabela)

- agora o acesso se dá calculando o hash e acessando diretamente a posição

Funções de Hash

- função hash perfeita: mapeia cada item em um único slot

- Dada uma coleção de itens, não há como construir tal função :-(

- Não é preciso uma função hash perfeita para ganhar eficiência

- uma forma de ter uma função hash perfeita é aumentar o tamanho da tabela de armazenamento
- mas não é prático para coleções grandes
- exemplo: armazenar 25 alunos a partir do número do cpf (11 dígitos)

Meta

- Criar função hash que minimize o número de **colisões**
- Seja fácil de computar
- Distribui os itens uniformemente na tabela

folding method

- divida o item em pedaços de tamanhos iguais
- some os pedaços

- Exemplo:
  - Telefone: (82)7989.1507
  - dividindo em grupos de 2: (82,79,89,15,07)
  - some-os: 82 + 79 + 89 + 15 + 07 = 272
  - Se a tabela tem 11 slots, então 272 % 11 = 8
  - O telefone (82)9988.1507 deve ser armazenado na posição 8

- Alguns ‘folding methods’ têm um passo extra: inverter pedaços de maneira alternada
  - (82)7989.1507 = (82,79,89,15,07) = (82,97,89,51,07) = 326 % 11 = 7

mid-square method

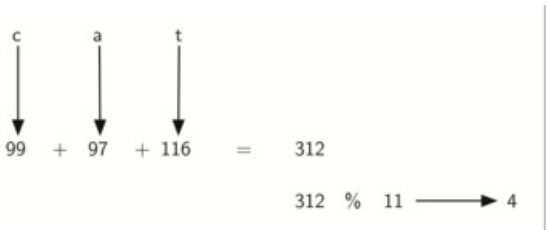
- Eleve o item ao quadrado
- Extraia algumas porções dos dígitos resultantes

- Exemplo:
  - Item: 44
  - 44^2 = 1936
  - extrair os dois dígitos do meio: 93
  - 93 % 11 = 5

Funções hash para String

- item: “cat”

```
>>> ord('c')
99
>>> ord('a')
97
>>> ord('t')
116
```



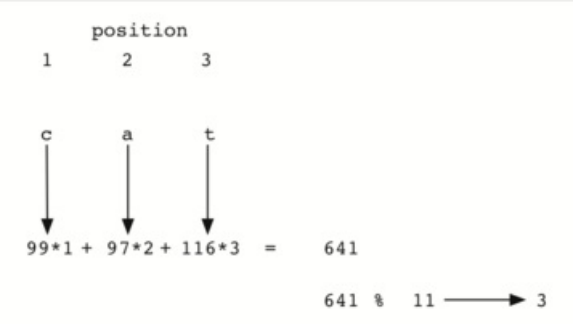

```
def hash(uma_string, tamanho_tabela):
    sum = 0
    for pos in range(len(uma_string)):
        sum = sum + ord(uma_string[pos])

    return sum %tamanho_tabela
```

Funções hash para String - Anagramas

- Anagramas terão o mesmo hash.

```
print(hash('cat',11)) #imprime 4
print(hash('tac',11)) #imprime 4
```

Solução?



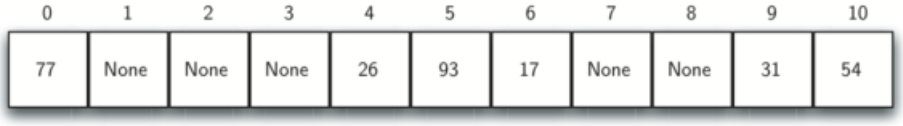
Nova função

```
print(hash('cat',11)) #imprime 3
print(hash('tac',11)) #imprime 2
```

Colisões

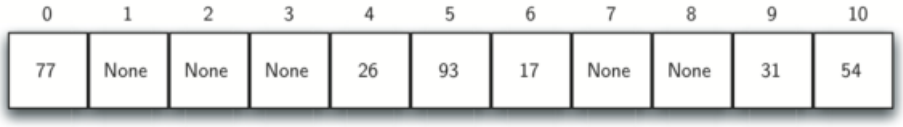
- Quando dois itens são associados ao mesmo slot
- Resolução de colisões: método para colocar o segundo item na tabela

Como colocar 44 na tabela?

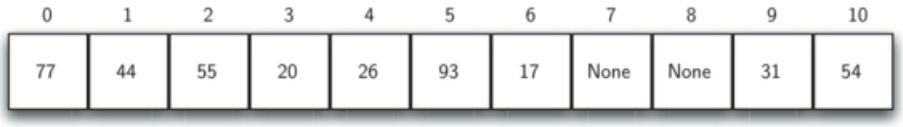


Colisões - Método 1: achar outro slot na tabela

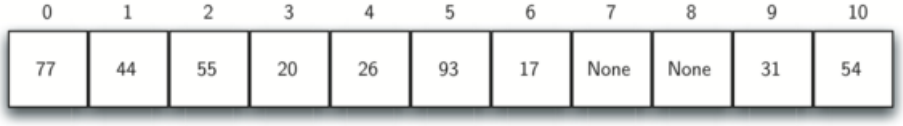
- Utilizar “open address” e “linear probing”



- Colocar 44 e 55 na tabela

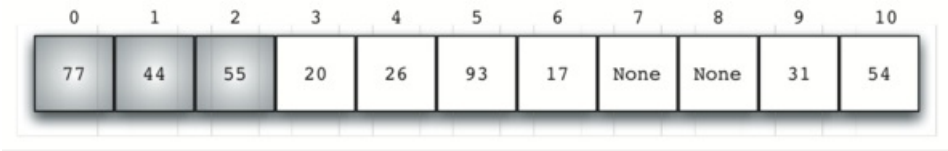


Colisões - Método 1: problemas

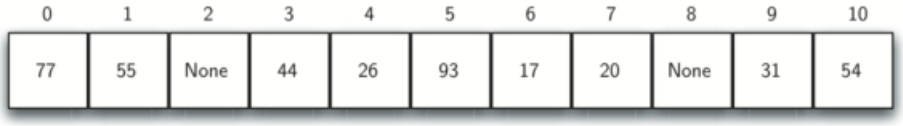


* Procura

- Clustering: muitos elementos com mesmo hash em posições vizinhas



- Lidando com clustering: pular alguns slots (rehashing)



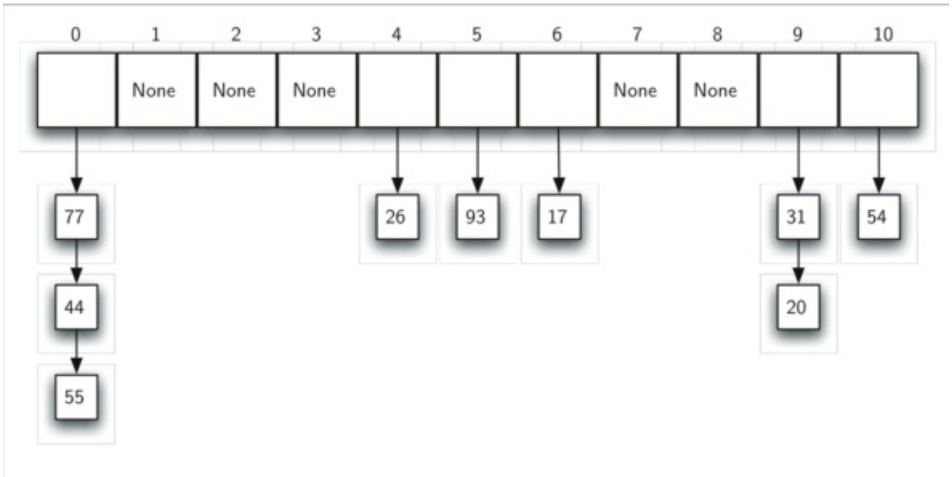
> newhashvalue=rehash(oldhashvalue)
> rehash(pos)=(pos+1)%sizeoftable.

Colisões - Método 2: quadradict probing

- Ao invés de pular as posições de forma constante para achar um novo slot (1, 3, 5, 7, 9, …), a função será incrementada da forma 1, 4, 9, 16

Colisões - Método 3: chaining

- Cada slot mantém uma referência para uma coleção (chain) de itens



Questões

Em uma tabela hash de tamanho 13 qual a posição do índice para as seguintes chaves: 27, 130 ? fh(v) = v%13

(A) 1, 10
(B) 13, 0
(C) 1, 0
(D) 2, 3

Suponha que você recebeu os seguintes conjunto de chaves para inserir em uma tabela hash que pode armazenar até 11 valores: 113, 117, 97, 100, 114, 108, 116, 105, 99. Qual das seguintes respostas demonstra o conteúdo da tabela hash após todas as chaves serem inseridas usando linear probing? Função hash: fh(v) = v%11

(A) 100, _____ , _____ , 113, 114, 105, 116, 117, 97, 108, 99
(B) 99, 100, _____ , 113, 114, _____ , 116, 117, 105, 97, 108
(C) 100, 113, 117, 97, 14, 108, 116, 105, 99, _____ , _____
(D) 117, 114, 108, 116, 105, 99, _____ , _____ , 97, 100, 113

TAD Map

- Uma instância do tipo Map suporta as seguintes funções/métodos:
- Map() Cria um novo mapa vazio. Retorna uma coleção mapa vazia
- put(key,val) adiciona um novo par chave-valor (key-value) ao mapa. Se a chave já existe no mapa então substitui o valor antigo com o novo
- get(key) Dada uma chave, retorna o valor armazenado no mapa ou None caso contrário
- del Remove o par chave-valor do mapa usando um comando da forma `map[key]`
- len() Retorna o número de pares chave-valor armazenados no mapa
- in Retorna True um comando na forma `key in map`, se a chave está no mapa, False caso contrário

TAD Map (construtor)

Construtor de um novo objeto HashTable:

```
class HashTable:
    def __init__(self):
        self._tamanho = 11 #
        self._slots = [None] * self._tamanho
        self._valores = [None] * self._tamanho
```

TAD Map (função hash)

```
    def hashfunction(self,chave,tamanho):
         return chave%tamanho

    def rehash(self,oldhash,tamanho):
        return (oldhash+1)%tamanho
```

TAD Map (função put)

```
    def put(self,chave,valor):
      valor_hash = self.hashfunction(chave,len(self._slots))

      if self._slots[valor_hash] == None:
        self._slots[valor_hash] = chave
        self._valores[valor_hash] = valor
      else:
        if self._slots[valor_hash] == chave:
          self._valores[valor_hash] = valor  #replace
        else:
          proximo_slot = self.rehash(valor_hash,len(self._slots))
          while self._slots[proximo_slot] != None and \
                        self._slots[proximo_slot] != chave:
            proximo_slot = self.rehash(proximo_slot,len(self._slots))

          if self._slots[proximo_slot] == None:
            self._slots[proximo_slot]=chave
            self._valores[proximo_slot]=valor
          else:
            self._valores[proximo_slot] = valor #replace
```

TAD Map (função get)

```
  def get(self,chave):
    slot_inicial = self.hashfunction(chave,len(self._slots))

    valor = None
    parar = False
    encontrou = False
    posicao = slot_inicial
    while self._slots[posicao] != None and  \
                    not encontrou and not parar:
      if self._slots[posicao] == chave:
        encontrou = True
        valor = self._valores[posicao]
      else:
        posicao=self.rehash(posicao,len(self._slots))
        if posicao == slot_inicial:
            parar = True
    return valor

  def __getitem__(self,chave):
      return self.get(chave)

  def __setitem__(self,chave,valor):
      self.put(chave,valor)
```

TAD Map (Exemplo 1)

```
from hashtable import *

H=HashTable()
H[54]="cat"
H[26]="dog"
H[93]="lion"
H[17]="tiger"
H[77]="bird"
H[31]="cow"
H[44]="goat"
H[55]="pig"
H[20]="chicken"
print(H._slots)
#[77, 44, 55, 20, 26, 93, 17, None, None, 31, 54]
print(H._valores)
#['bird', 'goat', 'pig', 'chicken', 'dog', 'lion','tiger', None, None, 'cow', 'cat']
print(H[17]) #'tiger'
H[20]='duck'
H[66]='fly'
print(H[20]) #'duck'
print(H._valores) #['bird', 'goat', 'pig', 'duck', 'dog', 'lion', 'tiger', 'fly', None, 'cow', 'cat']
print(H[99])#None
```

Análise de desempenho

- Inicialmente O(1)
- Considerando as colisões: O(1 + (FC)/ 2)
- fator de carga (FC)
- FC = (numero de itens)/(tamanho da tabela)

Visualize em https://visualgo.net/pt/hashtable

Para estudar

- Hashing
- Seções 5.5 do livro [5] https://panda.ime.usp.br/pythonds/static/pythonds_pt/05-OrdenacaoBusca/Hashing.html
- Capítulo sobre Hashing em qualquer livro sobre estruturas de dados

Referências

1. Tradução do livro *How to Think Like a Computer Scientist: Interactive Version*, de Brad Miller e David Ranum. link: https://panda.ime.usp.br/pensepy/static/pensepy/index.html
2. Allen Downey, Jeff Elkner and Chris Meyers. *Aprenda Computação com Python 3.0*. link: https://chevitarese.files.wordpress.com/2009/09/aprendacomputaocompython3k.pdf
3. SANTOS, A. C. *Algoritmo e Estrutura de Dados I*. 2014. Disponível em http://educapes.capes.gov.br/handle/capes/176522
4. SANTOS, A. C. *Algoritmo e Estrutura de Dados II*. 2014. Disponível em 2014. Disponível em https://educapes.capes.gov.br/handle/capes/176557
5. Tradução do livro [6] *Problem Solving with Algorithms and Data Structures using Python* de Brad Miller and David Ranum. link: https://panda.ime.usp.br/pythonds/static/pythonds_pt/index.html
6. Brad Miller and David Ranum. *Problem Solving with Algorithms and Data Structures using Python* link: https://runestone.academy/ns/books/published//pythonds/index.html
7. Caelum. Algoritmos e Estruturas Dados em Java. Disponível em https://www.caelum.com.br/download/caelum-algoritmos-estruturas-dados-java-cs14.pdf

That’s all Folks