Aula 5

Estrutura de Dados

Prof. Vinicius Pozzobon Borin

1

Conversa Inicial

2

- **Investigadores uma nova estrutura de dados**
  - **Tabela *hash***
- **Tabelas *hash*, ou somente *hash*, são estruturas de dados que usufruem de características de *arrays*, mas que melhoram, e muito, o tempo de inserção e busca dos dados**

3

Tabelas *Hash*

4

O problema da quitanda do Seu Zé

Stokkete/Shutterstock

5

Como buscar no caderno?

Quitanda do Seu Zé

D S T Q Q S S

Abacaxi	5,42	Tomate	3,24
Acerola	4,50	Mamão	9,38
Banana	2,98	Alface	3,63
Cebola	5,12	Beterraba	1,19
Cenoura	11,76	Quiabo	10,40
Morango	6,80	Pepino	2,83
Laranja	1,97	Melancia	8,88
Pera	4,77	Couve	3,55
Vagem	0,99	Coco-verde	12,00
Figo	1,25	Uva	15,00

unister.com | 0800 702 0500

Fonte: Vinicius Pozzobon Borin

OLHANDO PRODUTO POR PRODUTO

BUSCA SEQUENCIAL O(n)

6

Quitanda do Seu Zé

__/__/__
DSTQQSS

Abacaxi	5,42	Tomate	3,24
Acerola	4,50	Mamão	9,38
Banana	2,98	Alface	3,63
Cebola	5,12	Beterraba	1,19
Cenoura	11,76	Quiabo	10,40
Morango	6,80	Pepino	2,83
Laranja	1,97	Melancia	8,88
Pera	4,77	Couve	3,55
Vagem	0,99	Coco-verde	12,00
Figo	1,25	Uva	15,00

uniter.com | 0800 702 0500

Fonte: Vinicius Pozzobon Borin

OLHANDO PRODUTO POR PRODUTO

BUSCA BINÁRIA O(logn)

7

E agora, Seu Zé?

- **Não queremos deixar o cliente esperando**
- **Como tornar essa busca muito mais rápida, praticamente instantânea?**

8

"A menina que tudo lembra"

- **Joana é muito boa em decorar nomes e números**
- **Ao perguntar, ela sempre responde instantaneamente**

9

- **O acesso instantâneo à informação faz com que não tenhamos dependência do tamanho do conjunto de dados**

	Busca sequencial	Busca binária	Busca pela Joana
Itens no caderno	O(n)	O(logn)	O(1)
100	10 s	1 s	Instantâneo (0,1 s)
1000	1.6 min	1 s	Instantâneo (0,1 s)
10000	16.6 min	2 s	Instantâneo (0,1 s)

Fonte: Vinicius Pozzobon Borin

10

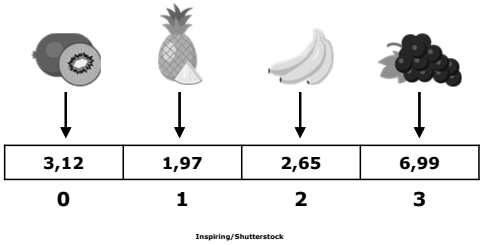
- **Como implementamos uma "Joana" em um programa?**

11

- **Como implementamos uma "Joana" em um programa?**
- **Acesso instantâneo, busca e inserção instantâneas**

12

Array com preços dos produtos



Inspiring/Shutterstock

13

Função *hash*

- Como você povoa um *array*?
- Provavelmente buscando a primeira posição livre, certo?

14

- E se pudermos calcular uma expressão matemática e/ou lógica que represente uma posição no *array*?

15

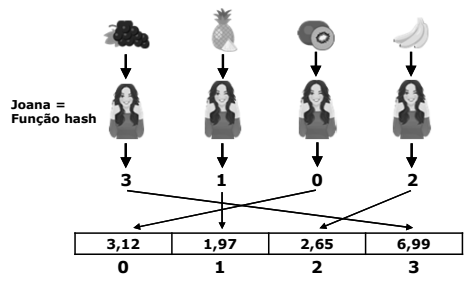
Mapeamento

Nomes dos produtos (*strings*)

⇩

Números (posições no *array*)

16



Imagens da Shutterstock: Inspiring; Volha Hlinskaya

17

Vantagens do emprego de uma função *hash*

- *Array*
  - Acesso ao dado na memória do *array*: O(1)
  - Busca de um dado no *array*: depende do algoritmo de busca, pesquisa sequencial O(n) e pesquisa binária O(logn)

18

- **Tabela *hash***
  - **Acesso ao dado na memória na tabela: O(1), pois é implementada como um *array* sequencial**
  - **Busca de um dado na tabela *hash*: O(1), pois acessa o índice por meio de uma função *hash***

19

- **Deixamos de precisar fazer uma varredura no *array* e tornamos o acesso ao dado independente do tamanho conjunto de dados**

20

Implementações em linguagens de programação

- **C++, container associativo map**
- **Java, classe HashMap**
- **Python, dicionário (dict)**

21

- **Vamos relembrar o dicionário em Python**

22

- **Não iremos trabalhar com estruturas de *hash* prontas na linguagem de programação, iremos aprender a construir uma do zero**

23

Aplicações de *hash*

- **Podemos manter um rastreamento de jogadas efetuadas por jogadores em jogos como xadrez, damas ou diversos outros jogos com alta quantidade de possibilidades**
- **Aplicações voltadas para segurança, como criptografia e autenticação de mensagens e assinatura digital, empregam *hashs***

24

- **A estrutura de dados-base que permite as populares criptomoedas, como bitcoin, operarem são *hashs*.**
  Elas trabalham com cadeias de *hashs* altamente complexas para manipular transações, oferecendo segurança e descentralização das operações

YellowJ/Shutterstock

25

Funções *Hash*

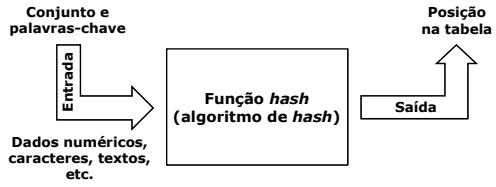
26

- **Também conhecida como**
  - **Algoritmo de *hash***
- **É uma expressão aritmética e/ou lógica específica para resolver uma determinada aplicação**

27

- **A função *hash* não apresenta uma fórmula definida**
- **Deve ser projetada levando-se em consideração o tamanho do conjunto de dados, seu comportamento e os tipos de dados-chave utilizados**

28



Fonte: Vinicius Pozzobon Borin

29

Uma boa função *hash*

- **Fácil de ser calculada. De nada valeria termos uma função com cálculos tão complexos e lentos que todo o tempo que seria ganho no acesso à informação com complexidade O(1), seria perdido calculando uma custosa função de *hash***
- **Capaz de distribuir palavras-chave o mais uniformemente possível dentro da estrutura do *array***

30

- Capaz de minimizar colisões. Os dados devem ser inseridos de uma forma que as colisões sejam as mínimas possíveis, reduzindo o tempo gasto resolvendo colisões e também reavendo os dados
- Capaz de resolver qualquer colisão que ocorrer

31

Método da divisão

- $h(k) = k\ MOD\ n$
  - Em que k é uma chave qualquer, n é o tamanho do array, e MOD representa o resto de uma divisão

32

Exemplo

$$h(k) = k\ MOD\ m$$

$k = 100$, $m = 12$ ⇨ $h(100) = 100\ MOD\ 12 = 4$

$k = 100$, $m = 15$ ⇨ $h(100) = 100\ MOD\ 15 = \mathbf{10}$

- Alterar o tamanho do vetor pode alterar todas as posições das chaves!

33

Exemplo

$$h(k) = k\ MOD\ m$$

- Oito dígitos telefônicos

$k = 99882233$

$k = 99 + 88 + 22 + 33$ ⇨ $h(242) = 242\ MOD\ 12 = \mathbf{2}$

$k = 242$

Caracteres alfanuméricos $h(k) = \left(\sum k_{ASCII\_Dec}\right) MOD\ m$

34

Hash universal

- Observe a situação a seguir para $h(k) = k\ MOD\ 6$

$k = 6$ ⇨ $h(6) = 6\ MOD\ 6 = 0$

$k = 12$ ⇨ $h(12) = 12\ MOD\ 6 = 0$

$k = 18$ ⇨ $h(18) = 18\ MOD\ 6 = 0$

$k = 24$ ⇨ $h(24) = 24\ MOD\ 6 = 0$

$k = 30$ ⇨ $h(30) = 30\ MOD\ 6 = 0$

⇩ Excesso de colisões!

Solução para minimizar colisões

Hashing universal

35

- O uso de uma única função *hash* pode resultar em uma situação em que todas as chaves precisam ser inseridas na mesma posição, gerando colisão e, consequentemente, piorando o desempenho do algoritmo

36

- **Para minimizar esse problema, adotamos um conjunto *H* de funções *hash*. Sorteamos uma função dentro da classe de funções disponíveis para fazer a inserção do dado**

37

Exemplo

Classe de funções *hash*

$$h_{a,b}(k) = ((ak + b)\ MOD\ p)\ MOD\ m$$

p é um número primo

$b = \{0, 1, 2 \ldots p - 1\}$

$a = b\ - \{0\}$

- **Podemos escolher um valor para p e m e variar a e b aleatoriamente para gerar funções diferentes com resultados diferentes**

38

Implementando uma Tabela *Hash*

39

Inserindo na tabela *hash*

Palavra-chave *k*	Função *hash* h(k)
Dois caracteres	$h(k) = (CHAR1_{ASCII} + CHAR2_{ASCII}) MOD\ m$
	$h(k) = (CHAR1_{ASCII} + CHAR2_{ASCII}) MOD\ 10$

0	1	2	3	4	5	6	7	8	9

$m = 10$

Fonte: Vinicius Pozzobon Borin

40

Palavra-chave *k*	Função *hash* h(k)	Resultado (posição)
PR	$h(PR) = (P_{ASCII} + R_{ASCII}) MOD\ 10$	$h(PR) = 2$
	$h(PR) = (80 + 82) MOD\ 10$	
	$h(PR) = 162\ MOD\ 10$	

0	1	2	3	4	5	6	7	8	9
		PR							

Fonte: Vinicius Pozzobon Borin

41

Palavra-chave *k*	Função *hash* h(k)	Resultado (posição)
SC	$h(SC) = (S_{ASCII} + C_{ASCII}) MOD\ 10$	$h(SC) = 0$
	$h(SC) = (83 + 67) MOD\ 10$	
	$h(SC) = 150\ MOD\ 10$	

0	1	2	3	4	5	6	7	8	9
SC		PR							

Fonte: Vinicius Pozzobon Borin

42

Palavra-chave *k*	Função *hash* h(k)	Resultado (posição)
RS	$\mathbf{h}(RS) = (R_{ASCII} + S_{ASCII}) MOD\ 10$ $\mathbf{h}(RS) = (82 + 83) MOD\ 10$ $\mathbf{h}(RS) = 165\ MOD\ 10$	$\mathbf{h}(RS) = 5$

0	1	2	3	4	5	6	7	8	9
SC		PR			RS				

Fonte: Vinicius Pozzobon Borin

43

0	1	2	3	4	5	6	7	8	9
SC	AL	PR	SP	RR	RS	RJ	-	DF	PE

Busca sequencial: seis iterações

Busca binária: três iterações

***Hash*: acesso imediato**

Fonte: Vinicius Pozzobon Borin

44

- **Vamos implementar em Python**

45

Colisões em Tabelas *Hash*

46

- **É impossível escrevermos uma função *hash* que seja livre de colisões**
- **Ocorrem quando uma chave precisa ser posicionada em uma posição em que já existe outra chave**

47

Exemplo

0	1	2	3	4	5	6	7	8	9
SC		PR			RS				

Colisão!

Inserindo a sigla *AM*:

$\mathbf{h}(AM) = 142\ MOD\ 10 = 2$

Fonte: Vinicius Pozzobon Borin

48

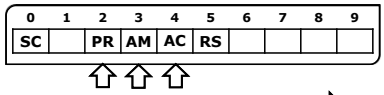
Endereçamento aberto

- Quando todas as posições são conhecidas e se tem em cada uma, no máximo, uma chave

49

Tentativa linear

- Quando uma colisão ocorre, busca-se a próxima posição livre



$h(AM) = 2$ ⇨ Colisão

$h(AM) = 3$ ⇨ Livre

$h(AC) = 2$ ⇨ Colisão

$h(AC) = 3$ ⇨ Colisão

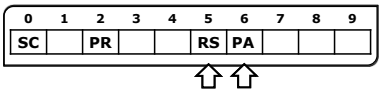
$h(AC) = 4$ ⇨ Livre

Fonte: Vinicius Pozzobon Borin

50

Tentativa quadrática

- Quando uma colisão ocorre, a próxima posição é calculada somando-se o valor da variável incremental *i*



$d = h(PA) = 5$ ⇨ Colisão

$d = h(PA) = (d + i) MOD\ 10$

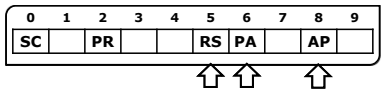
$i = 1$

$d = h\ (PA) = (5 + 1) MOD\ 10 = 6$ ⇨ Livre

Fonte: Vinicius Pozzobon Borin

51

- Quando uma colisão ocorre, a próxima posição é calculada somando-se o valor da variável incremental *i*



$d = h(AP) = 5$ ⇨ Colisão

$d = h(AP) = (d + i) MOD\ 10$

$i = 1$

$d = h\ (AP) = (5 + 1) MOD\ 10 = 6$ ⇨ Colisão

$i = 2$

$d = h\ (AP) = (6 + 2) MOD\ 10 = 8$ ⇨ Livre

Fonte: Vinicius Pozzobon Borin

52

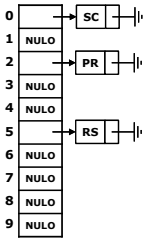
Endereçamento em cadeia

- Cada posição poderá conter diversas chaves encadeadas
- A forma de implementar é utilizando uma lista encadeada simples

53

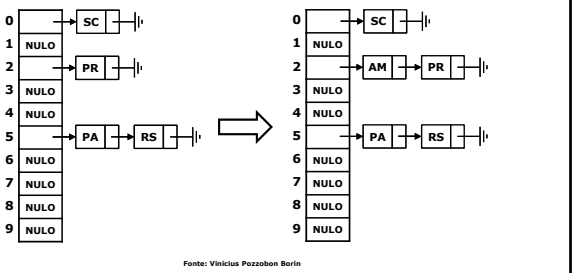
Resolvendo colisões com listas

- Uma lista encadeada de chaves é criada para cada posição do vetor
- Quando uma colisão ocorre, a nova chave é inserida no início dessa lista



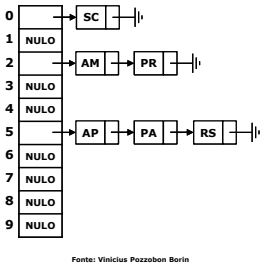
Fonte: Vinicius Pozzobon Borin

54



Fonte: Vinicius Pozzobon Borin

55



Fonte: Vinicius Pozzobon Borin

56

Implementando Colisões e Desempenho *Hash*

57

Endereçamento aberto e tentativa linear

- **Vamos implementar em Python**

58

Fator de carga

- **Ajuda a definir se o tamanho da tabela *hash* é suficiente para uma determinada aplicação**

$$\frac{Total\ de\ chaves}{Total\ de\ espaços}$$

59

- **26 estados brasileiros (+ DF)**
- **10 posições no *array***
- **Fator de carga: 2,7**

0	1	2	3	4	5	6	7	8	9
SC		PR	AM		RS				

Fonte: Vinicius Pozzobon Borin

60

- **Fator de carga acima de 1,0 indica a necessidade de redimensionar o *array***
- **Redimensionar o tempo todo tem um alto custo computacional**

61

Desempenho

	Tabela hash (caso médio)	Tabela hash (pior caso)	Array	Listas encadeadas
Busca	O(1)	O(n)	O(1)	O(n)
Inserção	O(1)	O(n)	O(n)	O(1)
Remoção	O(1)	O(n)	O(n)	O(1)

Fonte: Vinicius Pozzobon Borin

62

Referências

63

- **ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice, 2010.**
- **BHARGAVA, A. Y. Entendendo algoritmos. São Paulo: Novatec, 2017.**
- **CORMEN, T. H. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.**

64

- **DROZDEK, A. Estrutura de DADOS E ALGORITMOS Em C++. Tradução da 4. ed. norte-americana. São Paulo: Cengage Learning Brasil, 2018.**
- **FERRARI, R. *et al*. Estruturas de dados com jogos. Rio de Janeiro: Elsevier, 2014.**
- **KOFFMAN, E. B.; WOLFGANG, P. A. T. Objetos, abstração, estrutura de dados e projeto usando C++. São Paulo: Grupo GEN, 2008.**

65