

# ESTRUTURA DE DADOS I

## Aula 06

Prof. Sérgio Luis Antonello

FHO - Fundação Hermínio Ometto


31/03/2025

# Plano de Ensino

1. Unidade I – Métodos de ordenação em memória principal (objetivos d, e, f)
  - 1.1. Revisão de tipos de dados básicos em C, variáveis indexadas e recursividade
  - 1.2. Noções de complexidade computacional
  - 1.3. Conceitos e métodos de ordenação de dados
  - 1.4. Bubblesort, Insertsort e Selectsort
  - 1.5. Quicksort e Mergesort
  - 1.6. Shellsort e Radixsort
2. Unidade II – Métodos de pesquisa em memória principal (objetivos e, f)
  - 2.1. Pesquisa sequencial
  - 2.2. Pesquisa binária
  - 2.3. Hashing
3. Unidade III – Tipo abstrato de dados (TAD) (objetivo a)
  - 3.1. Revisão de registros, ponteiros e alocação dinâmica de memória
  - 3.2. Tipo abstrato de dados (TAD): conceitos e aplicações
4. Unidade IV – Estrutura de dados lineares (objetivos a, b, c)
  - 4.1. Lista Encadeada: conceitos e aplicações
  - 4.2. Pilha: conceitos e aplicações
  - 4.3. Fila: conceitos e aplicações

# Cronograma do Plano de Ensino

---

- 17/02 - Recursividade; Complexidade de tempo; Notação Big-Oh.
  - 24/02 - Métodos de ordenação: Bubble sort; Insert sort; Select sort.
  - 10/03 - Métodos de ordenação: Quick sort; Merge sort.
  - 17/03 - Métodos de ordenação: Shell sort; Radix sort.
  - 24/03 - Métodos de pesquisa: Sequencial; ~~Binária~~.
  - 31/03 - Métodos de pesquisa: Hashing.
  - 07/04 – Desenvolvimento do trabalho A1.
  - **14/04 - Prova 1**
- 

# Sumário

---

## ■ Primeiro momento (revisão)

- Busca sequencial
- Busca binária

## ■ Segundo momento

- Métodos de ordenação em memória primária
  - Hashing

## ■ Terceiro momento (síntese)

- Retome pontos importantes da aula

# 1. Primeiro momento: revisão

---

- Pesquisa Sequencial x Pesquisa Binária
  - Qual o objetivo de uma pesquisa?
  - Qual a diferença entre pesquisa sequencial e pesquisa binária? Qual a mais rápida?
  - Qual a condição básica dos dados para poder executar uma pesquisa binária?

# 1. Primeiro momento: revisão

## ➤ Pesquisa sequencial



## ➤ Pesquisa binária

1ª busca



$$\text{meio} = (0+9)/2 \rightarrow 4$$

2ª busca



$$\text{meio} = (5+9)/2 \rightarrow 7$$

3ª busca



$$\text{meio} = (5+6)/2 \rightarrow 5$$

# 1. Primeiro momento: revisão

---

## Correção do exercício 1 da aula 05

- 1) Cada registro do arquivo “**A05naoOrdenado.txt**”, contém o RA (de 10 a 99), nota A1 e nota A2 dos alunos de uma determinada turma de Algoritmos.  
Armazene-os em uma estrutura na memória principal e a partir de um RA informado pelo usuário, usando o método de **busca sequencial**, recuperar e apresentar os dados do referido aluno, incluindo sua nota final.

# 1. Primeiro momento: revisão

---

## Correção do exercício 2 da aula 05

- 2) Cada registro do arquivo “**A05ordenadoRA.txt**”, contém o RA (de 10 a 99), nota A1 e nota A2 dos alunos de uma determinada turma de Algoritmos. Os dados desse arquivo estão ordenados de modo crescente por RA.
- Armazene-os em uma estrutura na memória principal e a partir de um RA informado pelo usuário, usando o método de **busca binária**, recuperar e apresentar os dados do referido aluno, incluindo sua nota final.



# Motivação

O que acham sobre conhecer uma fermenta versátil que pode ser usada em diversas áreas?

## ➤ Bancos de Dados

- Indexação de dados
- Cache de dados

## ➤ Criptomoedas e Blockchain

- Geração de endereços
- Verificação de transações

## ➤ Redes de Computadores

- Roteamento de dados
- Detecção de duplicatas

## ➤ Segurança da Informação

- Armazenamento de senhas
- Verificação de Integridade de dados
- Assinaturas digitais

## ➤ Outras áreas

- Sistemas de busca
- Detecção de plágio
- Compressão de dados

## 2. Segundo momento

- Métodos de pesquisa em memória primária.
  - Pesquisa Sequencial.
  - Pesquisa Binária.
  - Hashing.



## 2. Segundo momento

Pesquisa Sequencial

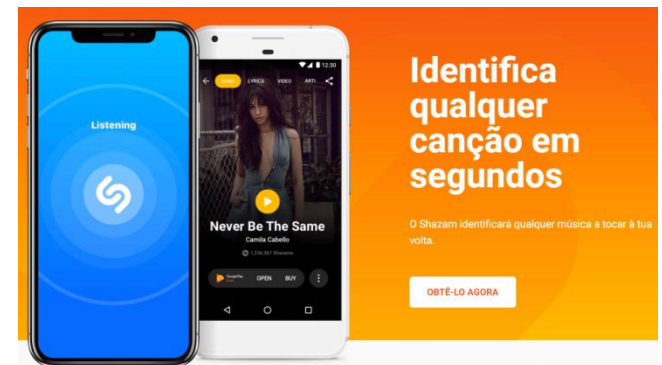
**Hashing**

Pesquisa Binária

# 3. Hashing



**HASHING**



# 3. Hashing: conceitos

---

- Hashing usa funções matemáticas para transformar dados em chaves únicas.
- Facilita a busca e comparação eficiente de dados.
- Faz uso de uma tabela de espalhamento (dispersão).

### 3. Hashing: conceitos

---

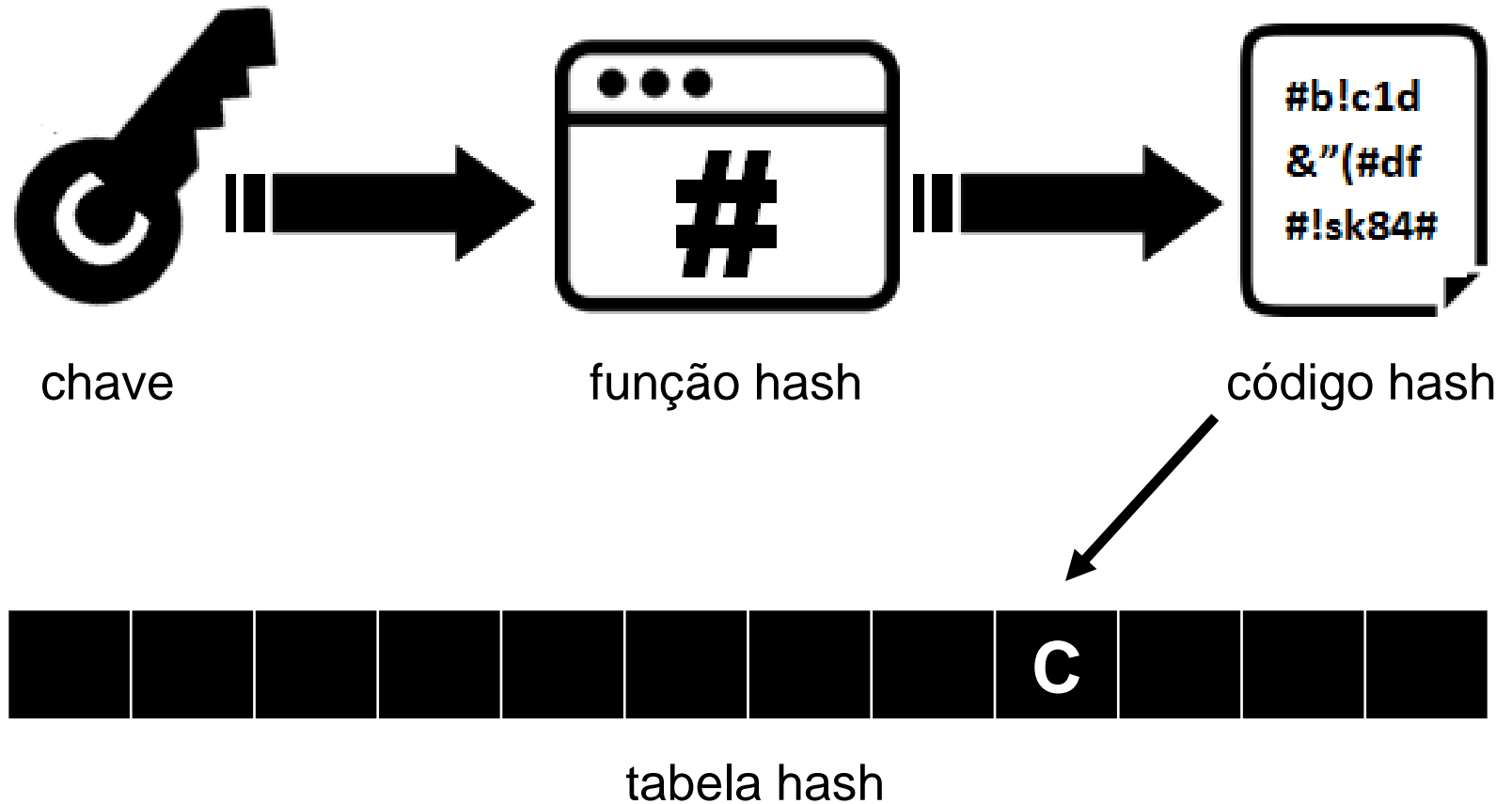
*“Em ciência da computação, uma tabela de dispersão (espalhamento) é uma estrutura de dados especial, que associa chaves de pesquisa a valores. Seu objetivo é, a partir de uma chave simples, fazer uma busca rápida e obter os dados desejados.”*

### 3. Hashing: ideia básica

---

- O elemento a ser armazenado, removido ou pesquisado é chamado de *chave* (*key*)
- A *chave* passa por uma *função de espalhamento* (*hash function*)
- A *hash function* gera um *código de espalhamento* (*hash code*)
- O *hash code* serve como endereço da chave na *tabela de espelhamento* (*hash table*)

### 3. Hashing: ideia básica





### 3. Hashing: ideia básica

---

**$h(ch) \rightarrow \text{endereço}$**

*função  
hash*



*chave*

*índice na  
tabela hash*

# 3. Hashing: ideia básica

Ch  $\rightarrow$  h(ch)  $\rightarrow$  endereço na tabela

Exemplo

ch = 51

h(ch)  $\rightarrow$  3

(acesso direto)

Tabela hash

0	12	end(12)
1	37	end(37)
2	2	end(2)
3	51	end(51)
...	...	...
M-1	x	end(x)

# 3. Hashing

- Há dois tipos de Hashing:
  - **aberto**: quando o número de *chaves* é potencialmente infinito.
  - **fechado**: quando o número de chaves é finito.
- A mesma *função hash* é usada para **inserção**, **remoção** e **busca** de chaves.
- Uma boa função hash deve ser **determinística**
  - ou seja: a função deve dar sempre o mesmo resultado quando aplicada à uma mesma chave.

### 3. Hashing: exemplo

- Exemplo de função hash (**h**): MOD

$$\mathbf{h(ch) = ch \% M}$$

- $M$  é o tamanho da tabela hash
- $\%$  é a operação *mod* (resto da divisão)

# 3. Hashing: exemplo

- A operação *mod* permite que o endereçamento na tabela de espalhamento varie no intervalo de 0 até  $M-1$ .
- Supondo  $M = 3$ , o resultado varia de 0 até 2:
  - $0 \% 3 = 0$
  - $1 \% 3 = 1$
  - $2 \% 3 = 2$
  - $3 \% 3 = 0$
  - $4 \% 3 = 1$
  - $5 \% 3 = 2$
  - $6 \% 3 = 0$
  - ...

### 3. Hashing: exemplo

---

- Exemplo: chaves = {1, 23, 10, 7, 19} e  $M = 5$

### 3. Hashing: exemplo

➤ Exemplo: chaves = {1, 23, 10, 7, 19} e  $M = 5$

➤  $h(1) = 1 \% 5 = 1$

➤  $h(23) = 23 \% 5 = 3$

➤  $h(10) = 10 \% 5 = 0$

➤  $h(7) = 7 \% 5 = 2$

➤  $h(19) = 19 \% 5 = 4$

### 3. Hashing: exemplo

➤ Exemplo: chaves = {1, 23, 10, 7, 19} e  $M = 5$

- $h(1) = 1 \% 5 = 1$
- $h(23) = 23 \% 5 = 3$
- $h(10) = 10 \% 5 = 0$
- $h(7) = 7 \% 5 = 2$
- $h(19) = 19 \% 5 = 4$

10	1	7	23	19
[0]	[1]	[2]	[3]	[4]



### 3. Hashing: exemplo

---

- Exemplo: chaves = {7, 11, 5, 20, 24, 9, 25} e  $M = 7$

### 3. Hashing: exemplo

➤ Exemplo: chaves = {7, 11, 5, 20, 24, 9, 25} e  $M = 7$

➤  $h(7) = 7 \% 7 = 0$

➤  $h(11) = 11 \% 7 = 4$

➤  $h(5) = 5 \% 7 = 5$

➤  $h(20) = 20 \% 7 = 6$

➤  $h(24) = 24 \% 7 = 3$

➤  $h(9) = 9 \% 7 = 2$

➤  $h(25) = 25 \% 7 = 4$

## 4. Hashing: colisões

➤ Exemplo: chaves = {7, 11, 5, 20, 24, 9, 25} e  $M = 7$

➤  $h(7) = 7 \% 7 = 0$

➤  $h(11) = 11 \% 7 = 4$

➤  $h(5) = 5 \% 7 = 5$

➤  $h(20) = 20 \% 7 = 6$

➤  $h(24) = 24 \% 7 = 3$

➤  $h(9) = 9 \% 7 = 2$

➤  $h(25) = 25 \% 7 = 4$



7		9	24	11	5	20
[0]	[1]	[2]	[3]	[4]	[5]	[6]

# 4. Hashing: colisões

---

- Uma “colisão” acontece quando a função hash gera um mesmo código hash (endereço) para duas ou mais chaves diferentes.
- Algumas estratégias para tratar colisões:
  - Hash fechado
    - Técnicas de *rehash* (*sondagem linear* ou *segunda função*)
  - Hash aberto
    - Encadeamento (listas ligadas)

# 5. Tratamento de Colisões: rehash

---

- **Técnicas de Rehash:** indicadas para quando o hashing é “fechado” (ou seja, quando a quantidade de dados a serem armazenados é limitada).
- Exemplos de rehash:
  - Sondagem linear
  - Segunda função hash

# 5. Tratamento de Colisões: rehash

---

- **Técnicas de Rehash:** Sondagem Linear
- Pesquisa-se pelo próximo endereço livre da tabela hash, de maneira incremental (de um em um).

## 5. Tratamento de Colisões: rehash

### ➤ Técnicas de Rehash: Sondagem Linear

➤ Exemplo: seja a tabela hash já parcialmente preenchida:

7	1	9	24		12	20
[0]	[1]	[2]	[3]	[4]	[5]	[6]

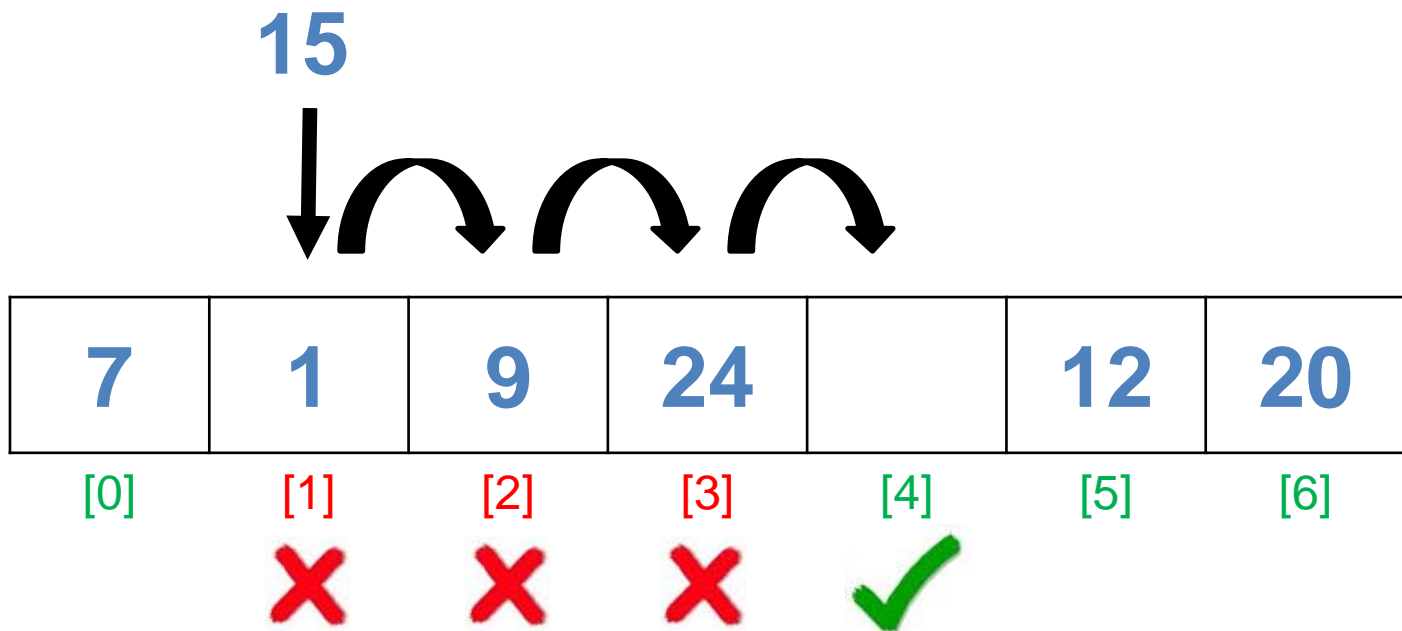
➤ Queremos adicionar a chave de valor **15**:

➤  $h(15) = 15 \% 7 = 1$



## 5. Tratamento de Colisões: rehash

- Técnicas de Rehash: **Sondagem Linear**
- Pesquisa-se pelo próximo endereço livre da tabela hash, de maneira incremental (de um em um).





## 5. Tratamento de Colisões: rehash

- Técnicas de Rehash: **Sondagem Linear**
- Pesquisa-se pelo próximo endereço livre da tabela hash, de maneira incremental.

$$\mathbf{rh(ch) = (h(ch) + i) \% M}$$

$$i = 1, 2, 3, \dots, M-1$$

# 5. Tratamento de Colisões: rehash

- **Técnicas de Rehash:** Sondagem Linear
- Pesquisa-se pelo próximo endereço livre da tabela hash, de maneira incremental.

Se o endereço  $h(ch)$  está ocupado, então:

$i = 1$

Enquanto endereço  $rh(ch)$  estiver ocupado e  
enquanto  $i < M$ , faça:

**$rh(ch) = (h(ch) + i) \% M$**

**$i = i + 1$**

Fim do enquanto

Fim do Se

# 5. Tratamento de Colisões: rehash

## ➤ Técnicas de Rehash: Sondagem Linear

- Pesquisa-se pelo próximo endereço livre da tabela hash, de maneira incremental.

## ➤ Questões:

- Como saber se um endereço da tabela está **desocupado**?
- Como o **desempenho** desta técnica se relaciona com o número de colisões? Qual é o pior caso?
- O que acontece quando, no caso de **busca** ou **remoção**, a chave procurada **não existe** na tabela?

## 5. Tratamento de Colisões: rehash

- Técnicas de Rehash: Segunda Função Hash
- Usa-se uma função hash secundária (auxiliar) para ajudar a calcular um novo possível endereço:

$$\mathbf{rh(ch) = (h(ch) + i * haux(ch)) \% M}$$

$$\mathbf{haux(ch) = 1 + ch \% (M - 1)}$$

# 5. Tratamento de Colisões: rehash

---

## ➤ **Técnicas de Rehash**

### ➤ **Vantagens:**

- Fácil de implementar

### ➤ **Desvantagens:**

- Se houver muitas colisões, a busca fica lenta.
- Inserções e remoções ficam mais difíceis.

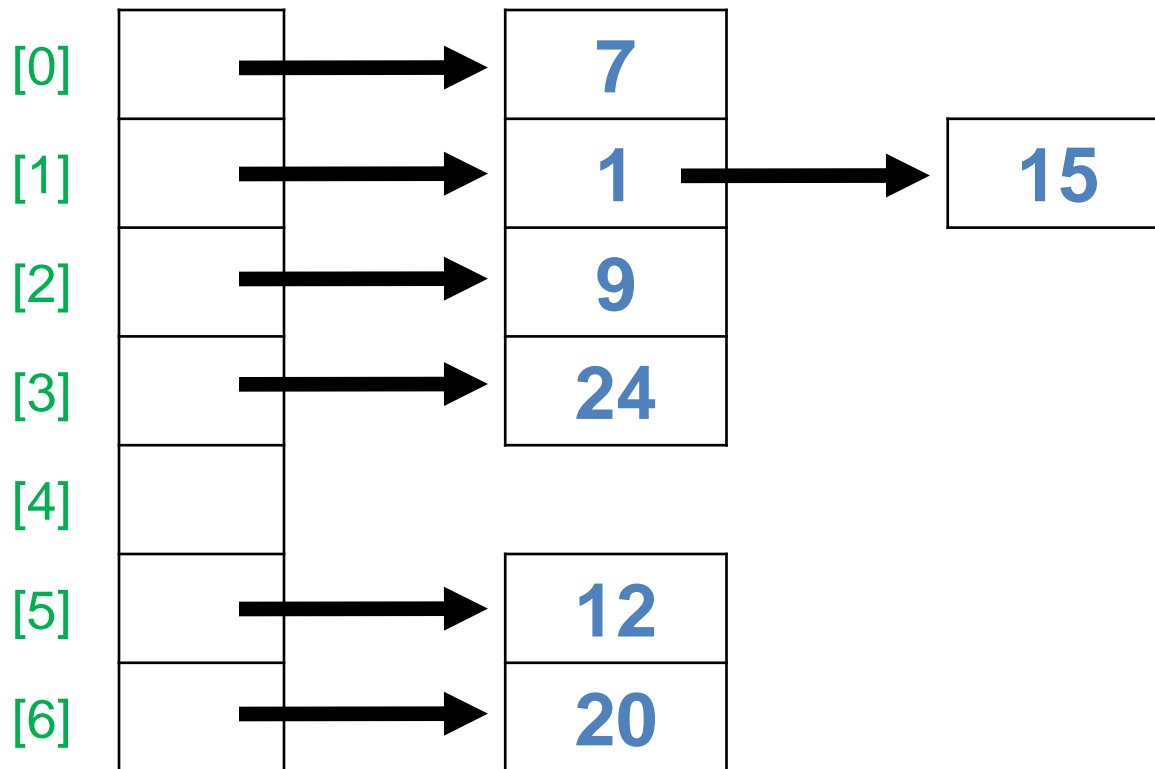
## 6. Tratamento de Colisões: Encadeamento

---

- **Encadeamento:** indicado para quando o hashing é “aberto” (ou seja, quando a quantidade de dados a serem armazenados é potencialmente ilimitada).
- As colisões são tratadas inserindo as chaves em **listas encadeadas** com início vinculado ao código hash (endereço) da chave.

## 6. Tratamento de Colisões: Encadeamento

- Exemplo: chaves = {24, 12, 1, 9, 15, 20, 7},  $M = 7$



# 6. Tratamento de Colisões: Encadeamento

---

## ➤ Encadeamento

### ➤ Vantagens:

- Tabela hash pode receber mais itens, mesmo quando todos os endereços estão ocupados.
- Permite percorrer a tabela por ordem de código hash.

### ➤ Desvantagens:

- Requer espaço extra para as listas.
- Se houver muitas colisões, implica em muito tempo para busca nas listas.



# 7. Hashing

---

## ➤ Principais Vantagens de Hashing:

- Busca tem potencial para ser quase instantânea:  $O(1)$ .

## ➤ Principais Desvantagens de Hashing:

- Dados não são armazenados sequencialmente (impossível percorrer a tabela de forma sequencial).
- Não se tem acesso direto ao elemento antecessor ou sucessor de um endereço qualquer.

# 7. Hashing

- Segredos para um bom Hashing:
  - Função hash deve provocar o melhor espalhamento possível (evitar colisões).
    - “Hashing Aceitável:” há poucas colisões!
    - “Hashing Perfeito”: não há colisões!
    - Recomenda-se que o tamanho da tabela hash seja um **número primo**.
  - Função hash deve ser fácil de computar
    - Recomenda-se o uso do operador ***mod***.
  - Deve-se adotar a estratégia de tratamento de colisões mais adequada para o conjunto de chaves a serem armazenadas.

# 7. Hashing

- **Hashing de Strings:** a dica é usar, como chave, a soma dos valores ASCII dos caracteres que formam a string.

- Ex: CAMILO, DIEGO, FABIANO, ORLANDO, SERGIO

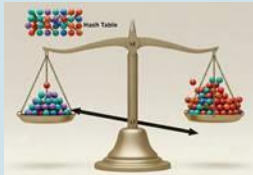
$$\begin{aligned}h(\text{CAMILO}) &= (67+65+77+73+76+79)\%5 &= 437\%5 &= 2 \\h(\text{DIEGO}) &= (68+73+69+71+79)\%5 &= 360\%5 &= 0 \\h(\text{FABIANO}) &= (70+65+66+73+65+78+79)\%5 &= 496\%5 &= 1 \\h(\text{ORLANDO}) &= (79+82+76+65+78+68+79)\%5 &= 527\%5 &= 2 \\h(\text{SERGIO}) &= (83+69+82+71+73+79)\%5 &= 457\%5 &= 2\end{aligned}$$

# 8. Hashing: Desafios e Limitações



## Colisões

Chaves diferentes podem gerar o mesmo hash.



## Distribuição

Má distribuição impacta a performance.



## Segurança

Funções fracas são vulneráveis à ataque.

# 9. Notação Big-oh

Algoritmo	Melhor Caso	Caso Médio	Pior Caso
Pesquisa Sequencial	$O(1)$	$O(n)$	$O(n)$
Pesquisa Binária	$O(1)$	$O(\log n)$	$O(\log n)$
Hashing (sem colisões)	$O(1)$	$O(1)$	$O(1)$

# 10. Exercícios

---

**Vamos  
Praticar!**



# 10. Exercícios

---

- 1) Construa um programa que receba os RAs de todos os alunos de uma determinada turma. Para cada um deles, gerar um *hash code* para armazenar em uma *hash table* de tamanho 97.

Em caso de colisão, simplesmente descarte o RA que está sendo gravado naquele momento.

Ao final do processo, exibir os RAs armazenados na tabela de espalhamento.

Os dados de entrada (75 no total) estão armazenados no arquivo “a06ex01Alunos.txt”.

# 10. Exercícios

## OBJETIVOS DE DESENVOLVIMENTO SUSTENTÁVEL

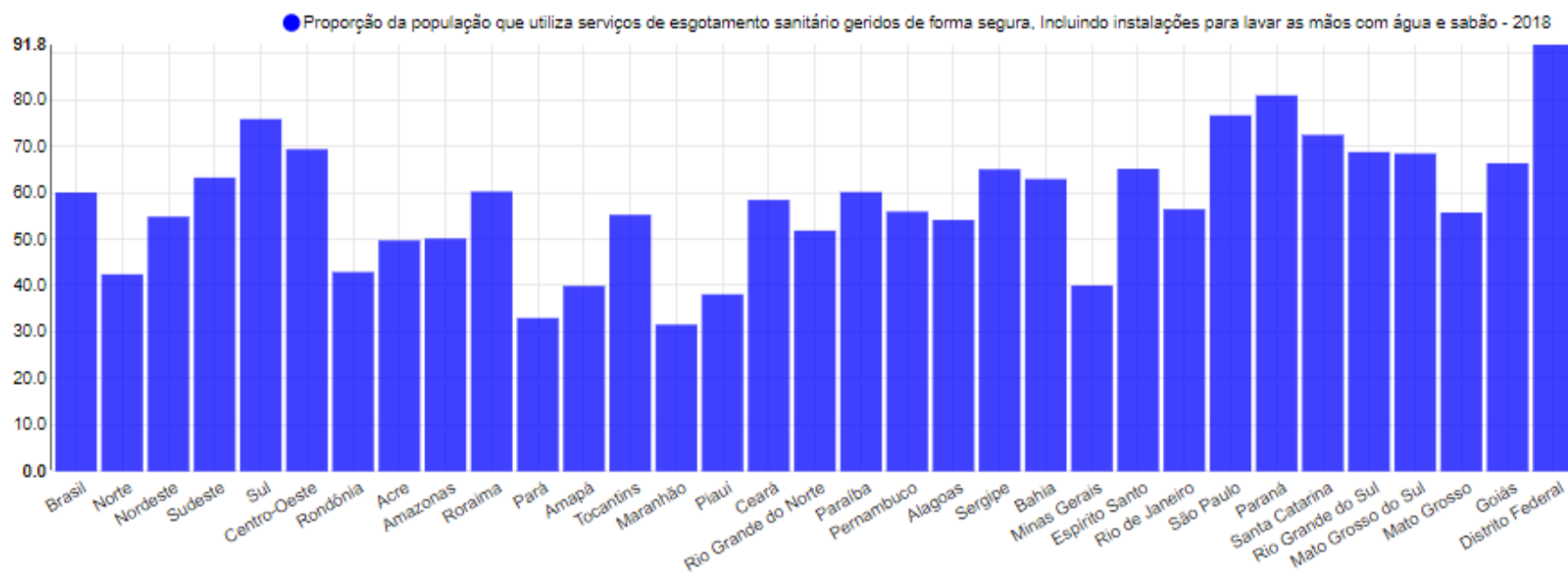




# 10. Exercícios



Indicador 6.2.1 - Proporção da população que utiliza (a) serviços de saneamento gerenciados de forma segura e (b) instalações para lavagem das mãos com água e sabão



# 10. Exercícios

Para explorar os dados disponíveis [clique aqui](#).

Indicador 6-2-1 - Proporção da população que utiliza (a) serviços de saneamento gerenciados de forma segura e (b) instalações para lavagem das mãos com água e sabão				
Brasil, Grande Região e Unidade da Federação	Proporção da população que utiliza serviços de esgotamento sanitário geridos de forma segura, Incluindo instalações para lavar as mãos com água e sabão			
	2017	2018		
Brasil	59,5	60,0		
Norte	42,4	42,4		
Nordeste	54,0	54,8		
Sudeste	63,1	63,2		
Sul	74,2	75,8		
Centro-Oeste	66,8	69,3		
Rondônia	43,1	42,9		
Acre	46,8	49,7		
Amazonas	50,9	50,1		
Roraima	54,0	60,2		
Pará	32,7	33,0		

# 10. Exercícios

- 2) A partir de dados obtidos do indicador ODS 6.2.1\*, pode-se tristemente observar que em 2018 no Brasil, apenas 60% da população utilizava serviços de saneamento gerenciados de forma segura e com instalações para lavagem das mãos com água e sabão.

Quem melhor dados apresenta é o DF com 91,8%, seguido do Paraná (80,9%), São Paulo (76,6%) e Santa Catarina (72,4%).

Os piores estados são Maranhão com apenas 31,6%, tendo logo acima Pará (33,0%), Piauí (38,1) e Amapá 39,9%.

O arquivo **A06ex02Estados.txt** apresenta em cada linha, as siglas e os nomes dos estados e do DF, que deve ser usados como chave para o seguinte processamento:

- leia cada nome de estado, aplique uma função Hash que gere o *hash code*. Pense em trabalhar com o valor ASCII de cada letra dos nomes.
- Utilize um vetor de 97 posições (tamanho M) como hash table e não se preocupe em fazer o tratamento de colisões.
- **No final do processamento, exiba quantas colisões ocorreram.**

\* Indicador 6.2.1 - Proporção da população que utiliza (a) serviços de saneamento gerenciados de forma segura e (b) instalações para lavagem das mãos com água e sabão.

# 11. Terceiro momento: síntese

---

- A busca sequencial percorre a estrutura, do início ao fim, comparando cada elemento ao conteúdo pesquisado.
- A busca binária é mais rápida que a busca sequencial.
- A busca binária exige que a estrutura esteja ordenada, já a busca sequencial pode ser feita em uma estrutura não ordenada.

# 11. Terceiro momento: síntese

---

- Hash é um método de busca que quando bem implementada é muito mais rápida que os outros métodos vistos. Esse tipo de busca tem potencial para ser  $O(1)$ .
- O conceito é transformar uma chave a ser pesquisada em um código de acesso à tabela de espalhamento.
- O hash perfeito é alcançado quando não ocorre colisões das chaves dentro da tabela de espalhamento.
- Quando ocorrem colisões, elas devem ser tratadas.