Pesquisa de Dados

Pesquisa/busca de dados

- Estratégias para pesquisa (busca) de um elemento específico em um conjunto de dados. (vetor, lista, etc.)
- Operação frequente
- Métodos mais conhecidos:
  - Busca sequencial/linear
  - Busca binária
  - Hashing (dispersão)

Pesquisa sequencial/linear

Ex.: buscar o valor 90 no vetor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14	21	5	45	12	3	86	98	46	53	24	2	1	15	90	47

Comparar cada elemento do vetor com o valor procurado

Pesquisa sequencial/linear

- Simples
- Lenta (quando comparada a outros métodos)
- Aplicável em situações específicas
- Complexidade:
  - Melhor caso: O(1)
  - Pior caso: O(n)
  - Caso médio: O(n/2)

Pesquisa sequencial/linear

Situação em que pesquisa linear é aceitável:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14	21	5	45	12	3	86	98	46	53	24	2	1	15	90	47

Dados ordenados por frequência de acesso.

Ex.: lista com reinserção por acesso (MRU)

Busca binária

Dados devem estar ordenados

Método de divisão-e-conquista:

Achar valor no ponto médio dos dados (VPM)
Se VPM == valor procurado, pesquisa retorna sucesso
Se VPM > valor procurado, repetir método para valores abaixo de VPM
Se VPM < valor procurado, repetir método para valores acima de VPM
Se não há mais valores a pesquisar, valor não existe nos dados

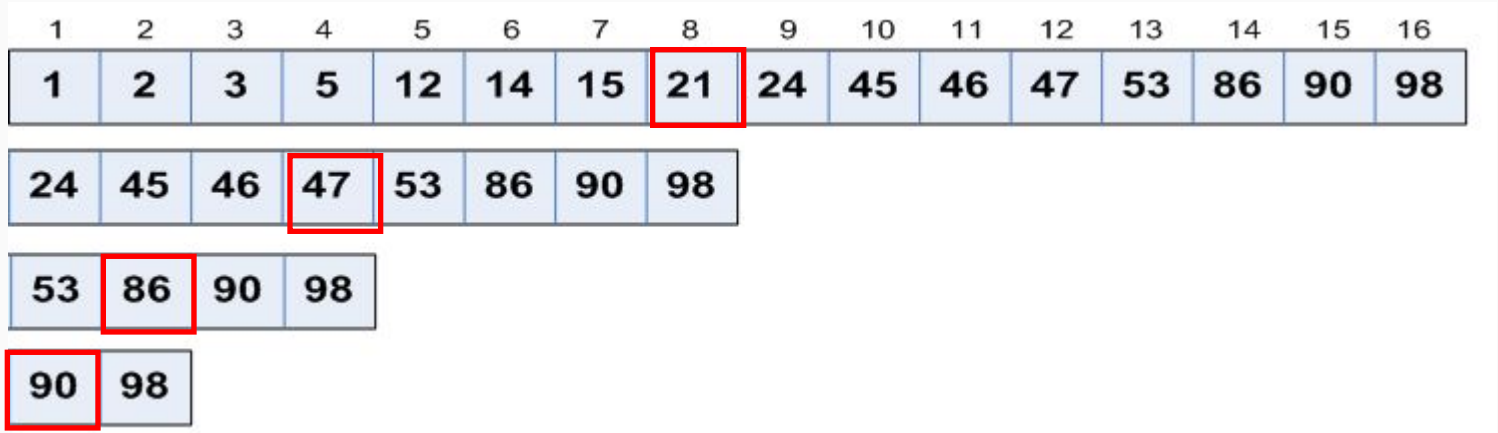
Busca binária

Ex.: buscar o valor 90 no vetor

14	21	5	45	12	3	86	98	46	53	24	2	1	15	90	47

Passo 1: ordenar o vetor

Passo 2: busca binária (para buscas sucessivas, não é necessário repetir o passo 1)



Busca binária

- Simples
- Rápida
- Complexidade:
  - Melhor caso: $O(1)$
  - Pior caso: $O(\log_2 n)$
  - Caso médio: $O(\log_2 n)$

- **Considerar tempo de ordenação!**

Busca interpolativa

Demanda saltos com intervalos pré-conhecidos (ou possíveis de serem “chutados”)

Tabela Hash/de dispersão (*Hash table*)

Estrutura de dados que implementa uma **matriz associativa**

- Mapeamento direto entre **chave** e **valor**
  (ex.: idade[“Maria da Silva”] = 90)

Tenta mesma funcionalidade/complexidade de acesso a vetor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14	21	5	45	12	3	86	98	46	53	24	2	1	15	90	47

Tabela Hash/de dispersão (*Hash table*)

Uso de uma função - Função *Hash* - que mapeia uma chave para um índice

Ex.: hash(“Maria da Silva”) → 14

Idealmente tem complexidade **O(1)**

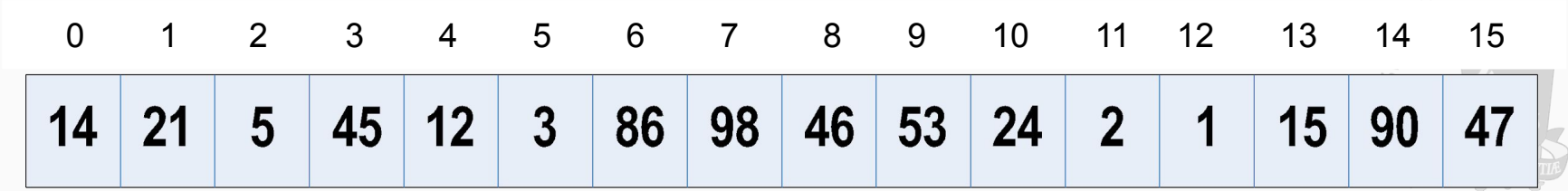


Tabela Hash

Ex.: criar uma tabela para todos os alunos da UFSM.

Chave → matrícula

Uma matrícula atual seria algo como: 202112345

Primeiro problema: tamanho da tabela (iremos usar uma turma de cada vez? Um curso? Todos os dados?)

Segundo problema: função hash/de mapeamento (idealmente, função **sem colisões** para os dados)

Tabela Hash

Exemplo comum de função Hash:

posição = chave % tamanho da tabela

Função imperfeita!

Tabela Hash

Suponha que queiramos armazenar as seguintes chaves:
C, H, A, V, E e S em um vetor de P = 7 posições (0..6) conforme a seguinte função

f(k) = k(código ASCII) % P

Tabela ASCII: C (67); H (72); A (65); V (86); E (69) e S (83).

f(‘C’) = 4; f(‘H’) = 2; f(‘A’) = 2; f(‘V’) = 2; f(‘E’) = 6; f(‘S’) = 6

0	
1	
2	H, A, V
3	
4	C
5	
6	E, S

Tabela Hash

Suponha que queiramos armazenar as seguintes chaves:
C, H, A, V, E e S em um vetor de P = 7 posições (0..6) conforme a seguinte função

f(k) = k(código ASCII) % P

Tabela ASCII: C (67); H (72); A (65); V (86); E (69) e S (83).

f(‘C’) = 4; f(‘H’) = 2; f(‘A’) = 2; f(‘V’) = 2; f(‘E’) = 6; f(‘S’) = 6

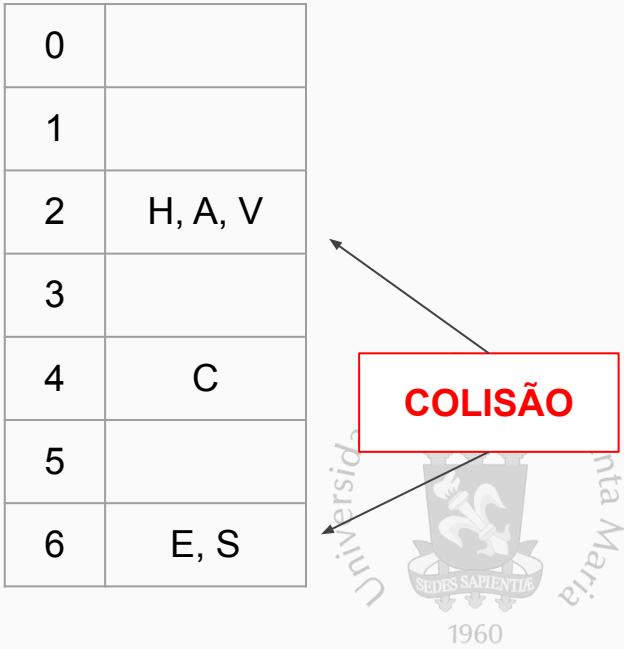


Tabela Hash

Resolução de colisão

- Encadeamento
- Endereçamento aberto (com tentativa linear, quadrática ou dispersão dupla)

Considerar o **paradoxo do aniversário**

Paradoxo do aniversário

Quantas pessoas precisam estar no mesmo lugar para duas terem aniversário no mesmo dia do ano?(desconsiderar anos bissextos)

Paradoxo do aniversário

Quantas pessoas precisam estar no mesmo lugar para duas terem aniversário no mesmo dia do ano?(desconsiderar anos bissextos)

366 → 100% de chances

100 → 99.99997%
70 → 99.9%
60 → 99.4%
30 → 70.6%
23 → 50.7%

Paradoxo do aniversário

Para uma tabela com um milhão de *slots*, se 2450 chaves forem inseridas, há uma probabilidade de **95%** de ocorrer uma colisão.
(e esta probabilidade apenas pode aumentar nas inclusões subsequentes)

Tabela Hash

Resolução de colisão

- Encadeamento
- Endereçamento aberto (com tentativa linear, quadrática ou dispersão dupla)

Considerar o **paradoxo do aniversário**

Tabela Hash

Encadeamento:

- Simples
- Potencial para degeneração

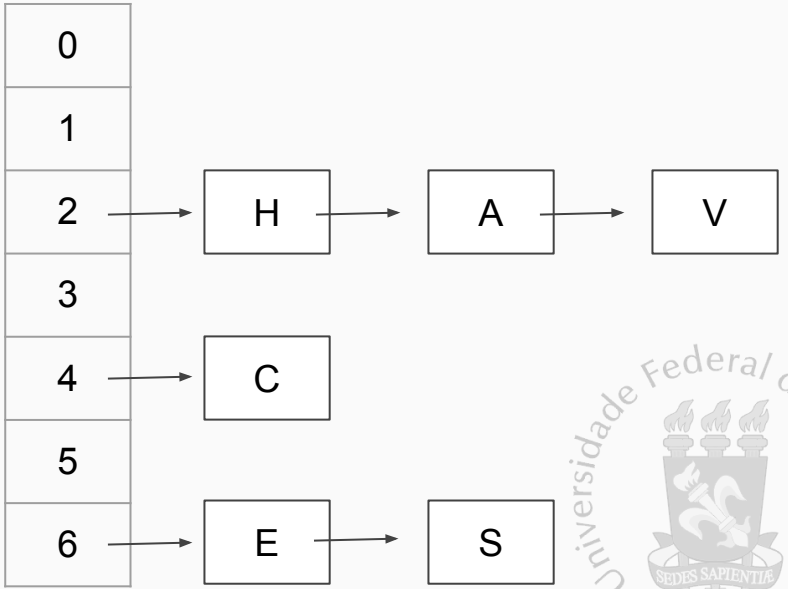


Tabela Hash

Endereçamento aberto

- Com tentativa linear (*linear probing*):Se houver colisão, somar valor fixo (em geral 1) ao índice calculado
- Simples
- Bom uso de caches em arquiteturas modernas
- Problemas:
  - Se elemento não está na tabela, será feita uma busca sequencial em toda ela
  - Potencial para ***clustering***

0	S
1	
2	H
3	A
4	C
5	V
6	E

Tabela Hash

Endereçamento aberto

- Com tentativa quadrática (*quadratic probing*)
  - Ex. de polinômio para cálculo de índice:

$$H + 1^2, H + 2^2, H + 3^2, H + 4^2, \ldots, H + k^2$$

  - reduz/elimina ***clustering***

- Com dispersão dupla (*double hash*)
  - um hash para o índice, outro para deslocamento

0	
1	
2	H
3	A
4	C
5	
6	V

Tabela Hash

Vantagens

- Velocidade
- Se pares (chave,valor) são conhecidos de antemão, é possível criar uma função hash perfeita, eliminando a necessidade de armazenamento da chave na tabela e obtendo-se uma complexidade O(1)

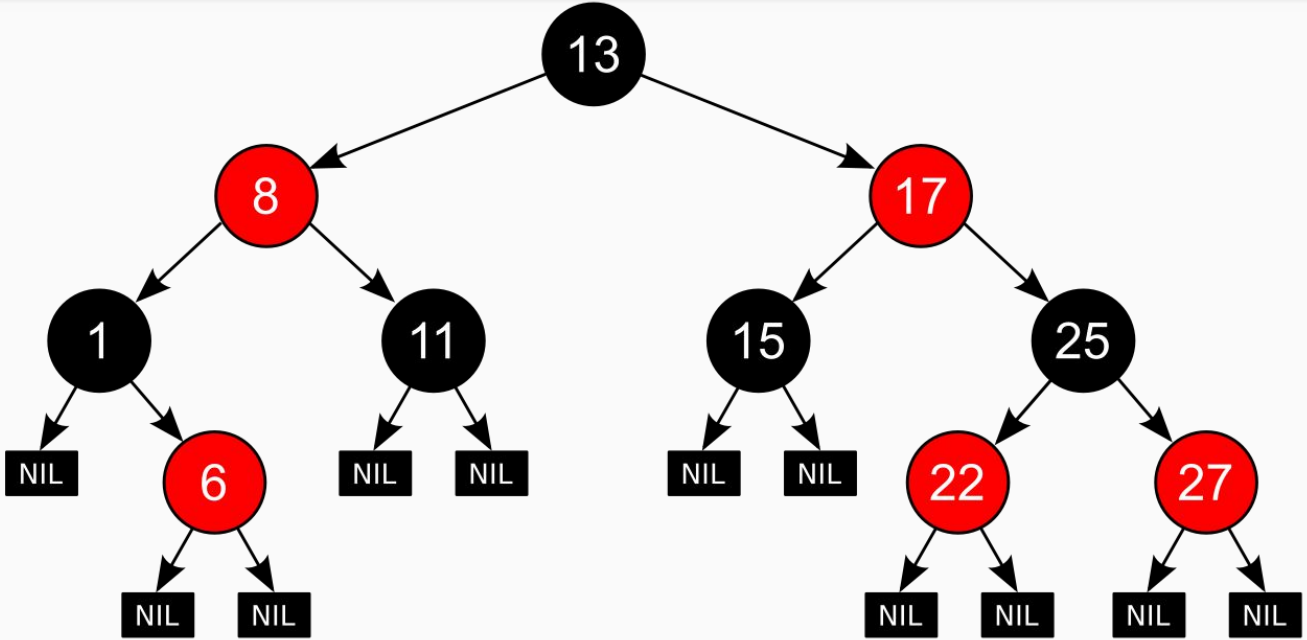
Desvantagens

- Custo da função hash pode ser alto, impossibilitando tabelas pequenas
- enumerar /listar em ordem dados armazenados em uma tabela hash é muito caro
- Custo de uma operação pode ser muito alto (*dynamic resizing*)
- Em geral, têm baixa localidade

Árvores de pesquisa

- Binárias
- Auto-balanceadas
  - AVL
  - Red-Black
  - B-Tree
  - B^+-Tree

Red-Black (Rubro-negra/preto-vermelha)



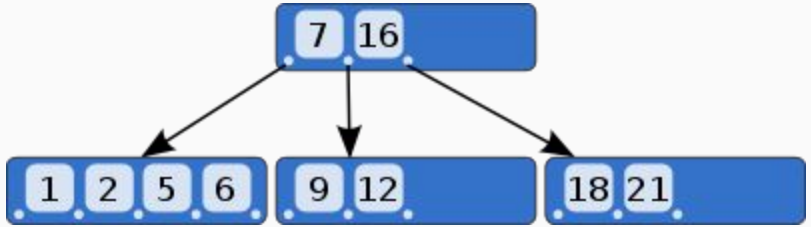
Red-Black (Rubro-negra/preto-vermelha)

Regras:

1. Cada nó é preto ou vermelho
2. A raiz é preta
3. Cada folha (NIL) é preta
4. Se um nó é vermelho, seus filhos são pretos
5. Para cada nó, todos os caminhos simples até seus descendentes folha contém o mesmo número de nós pretos

B-Tree (1979)

- Otimizada para leitura/escrita de grandes blocos (mem. externa)
- Comum em sistemas de arquivos ou BD
- Nós tem tamanho variável de chaves, usualmente entre *d* e *2d*
- Todas as folhas na mesma profundidade
- Rebalanceamentos menos frequentes
- Pesquisas, inserções e deleções em tempo logaritmico



B-Tree (1979)

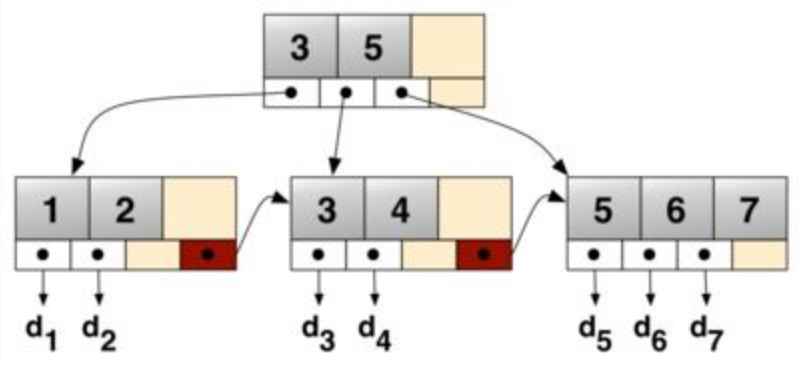
- De acordo com Knuth, uma B-Tree de ordem m deve satisfazer as seguintes propriedades:
  - Cada nó tem, no máximo, m filhos
  - Cada nó interno tem, no mínimo, m/2 filhos
  - A raiz tem, no mínimo, 2 filhos, exceto quando é folha
  - Todas as folhas estão no mesmo nível
  - Um nó não-folha com k filhos contém k-1 chaves

B-Tree

- Inserção
  - Se nó tem menos que *2d* elementos, insere no nó (em ordem)
  - Senão
    - Quebra nó em dois (split)
      - Escolhe mediana (entre elementos existentes e novo)
      - Mediana é novo pai (valor de separação)
      - Valor de separação é inserido no pai. Se inserção causar nova quebra, repetir recursivamente (pode gerar nova raiz)

B^+-Tree

Pode ser vista como uma B-Tree em que cada nó contém comente chaves (e não pares chave-valor) e há um nível adicional com folhas ligadas



B^+-Tree

- Originalmente usada para armazenar dados para recuperação eficiente
- Aplicações reais:
  - NTFS
  - ReiserFS
  - Microsoft SQL
  - JFS
  - Oracle
  - ...

B*-Tree

- Nós (exceto raiz) tem, no mínimo, ⅔ das chaves possíveis
- Split só ocorre quando irmãos cheios, e quebra dois nós em 3 (e não 1 em 2 como na B-Tree)
- Atrasa split ao máximo possível. Ao invés de realizar split quando um nó está cheio, realiza-se operação de *spill* (transbordo): chave que não cabe no nó é transportada para nó irmão. Somente quando os dois irmãos (direita e esquerda) estão cheios realiza-se split, quebrando o nó cheio e um dos irmãos em 3. Nesse caso uma chave é inserida no pai, podendo causar sucessivos splits até a raiz.