Campus de Frutal

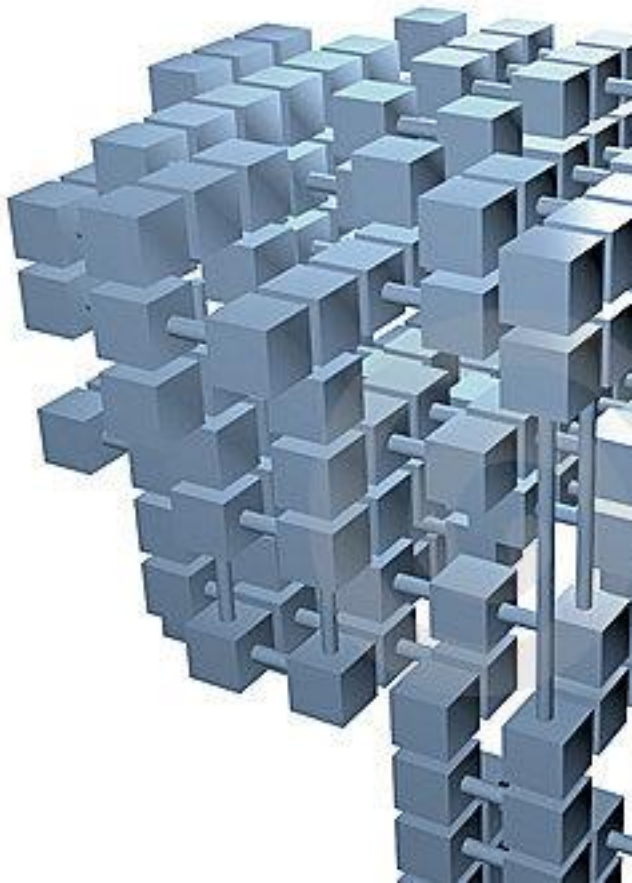
UEMG UNIVERSIDADE DO ESTADO DE MINAS GERAIS

Sistemas de Informação

Estrutura de Dados

II

Hashing

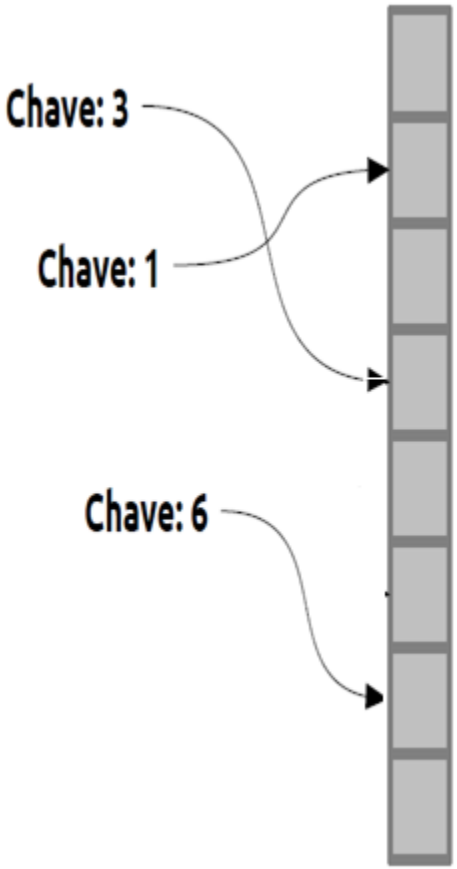


Prof. Ivan José dos Reis Filho

ivanfilhoreis@gmail.com

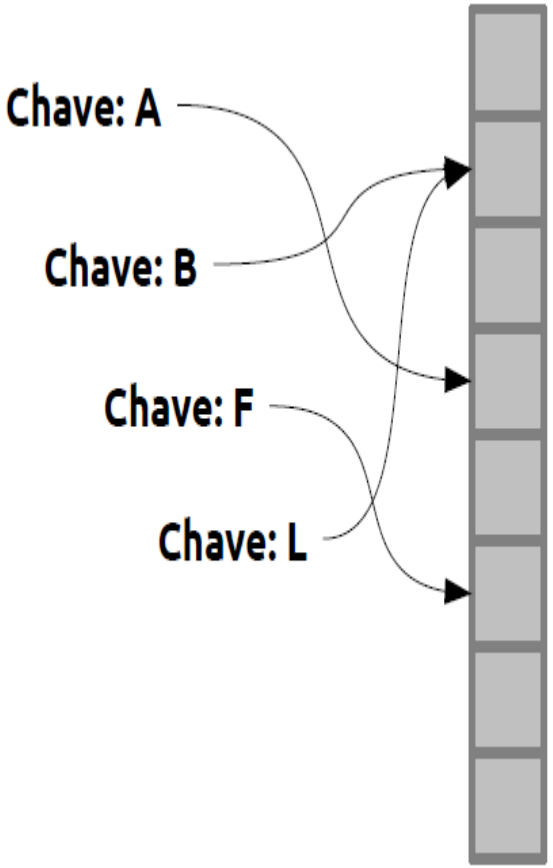
Hashing

- Armazenamento de dados em tabelas *indexadas pela chave*
- Função *Hash: transforma a chave em uma posição na tabela*
  - Exemplo:
    - Chave ***i na posição i do vetor***
- Ideal:
  - Cada chave é mapeada para uma posição única do vetor
  - Todas as posições do vetor possuem chaves mapeadas



Hashing

- Exige um conhecimento prévio dos dados (chaves)
  - Números inteiros
  - Intervalos
  - Distribuição das chaves
- Função *hash: f(D) → [0, M-1] ∈ N,* onde D é o domínio das chaves e M é o número de posições na tabela (vetor)
- Em geral: espaço ideal é proibitivo
- Balanceamento: espaço *x tempo*
- Colisões: chaves diferentes levam a uma mesma posição



Hashing

- Balanceamento: espaço *x tempo*
  - Sem limitação de espaço: função trivial
  - Sem limitação de tempo: colisão trivial (sequencial)
- Boa função hash:
  - Cálculo rápido
  - Distribuição uniforme pela tabela
- Exemplos:
  - Telefones de Frutal:
    - Utilizar os últimos dígitos é melhor do que os dígitos do prefixo
  - RAs dos alunos
    - Últimos digitos, sem o dígito verificador (bom? ruim?)
- Cada chave exige uma abordagem diferente

Hashing

- Abordagens comuns:
  - Módulo
- ***chave % M***
- Uso dos *k dígitos menos significativos*
  - Reais
- ***round(chave * M)***
- Uso dos dígitos mais significativos
  - Strings
- ***polinômio com o caracteres (ascii)***
  - ...

Colisões

- Chaves distintas podem levar a uma mesma posição
- Tratamentos mais comuns:
  - Listas encadeadas
  - Sequencial: linear, quadrático, …
  - *Hashing duplo, triplo, ...*

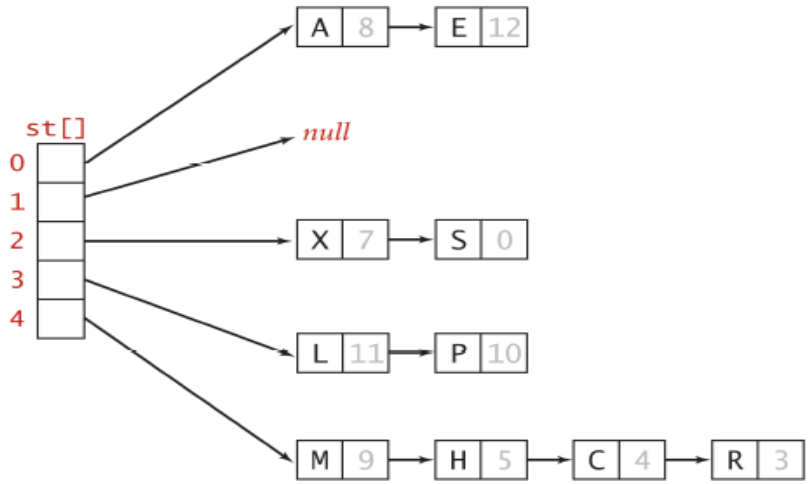
Colisões

- Listas encadeadas:

Use an array of $M < N$ linked lists. [H. P. Luhn, IBM 1953]

- Hash: map key to integer i between 0 and $M - 1$.
- Insert: put at front of i^{th} chain (if not already there).
- Search: need to search only i^{th} chain.

key	hash	value
S	2	0
E	0	1
A	0	2
R	4	3
C	4	4
H	4	5
E	0	6
X	2	7
A	0	8
M	4	9
P	3	10
L	3	11
E	0	12



Colisões

- Sequencial: linear, quadrático, ...

Linear probing demo

Hash. Map key to integer i between 0 and $M - 1$.

Insert. Put at table index i if free; if not try $i + 1$, $i + 2$, etc.

Search. Search table index i; if occupied but no match, try $i + 1$, $i + 2$, etc.

Note. Array size M must be greater than N.

Colisões

- Múltiplas funções *hash*
  - *Hashing duplo, triplo, ...*
- Auxilia no espalhamento dos dados
- Maior dificuldade na remoção

Fator de Carga

- Proporção entre uso e espaço da função *hash*
- Fator de carga baixo
  - Maior probabilidade de não ocorrer colisão
- Auxílio para técnicas de *rehash*
  - Aumento do espaço da função *hash*

Hashing

- No pior caso, buscas, inserções e remoções em uma tabela *hashing são* ***O(n)***
  - Quando todas as chaves colidem
- O fator de carga afeta o desempenho
- O tempo de execução esperado é ***O(1)***

Hashing x Árvore Balanceadas

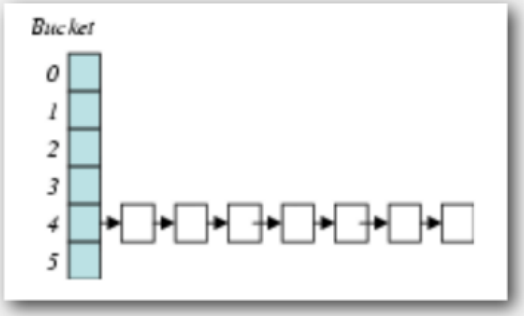
- Hashing:
  - Implementação simples
- Estrutura simplificada
  - Muito rápido (para conjuntos simples/conhecidos)
    - Constante
- Árvores balanceadas
  - Performance garantida
    - logN
  - Mais fácil adaptar para qualquer tipo de dados
    - função de comparação

War Story: algorithmic complexity attacks

Q. Is the uniform hashing assumption important in practice?

A. Obvious situations: aircraft control, nuclear reactor, pacemaker.

A. Surprising situations: denial-of-service attacks.



malicious adversary learns your hash function (e.g., by reading Java API) and causes a big pile-up in single slot that grinds performance to a halt

Real-world exploits. [Crosby-Wallach 2003]

- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem.
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.

BucketSort

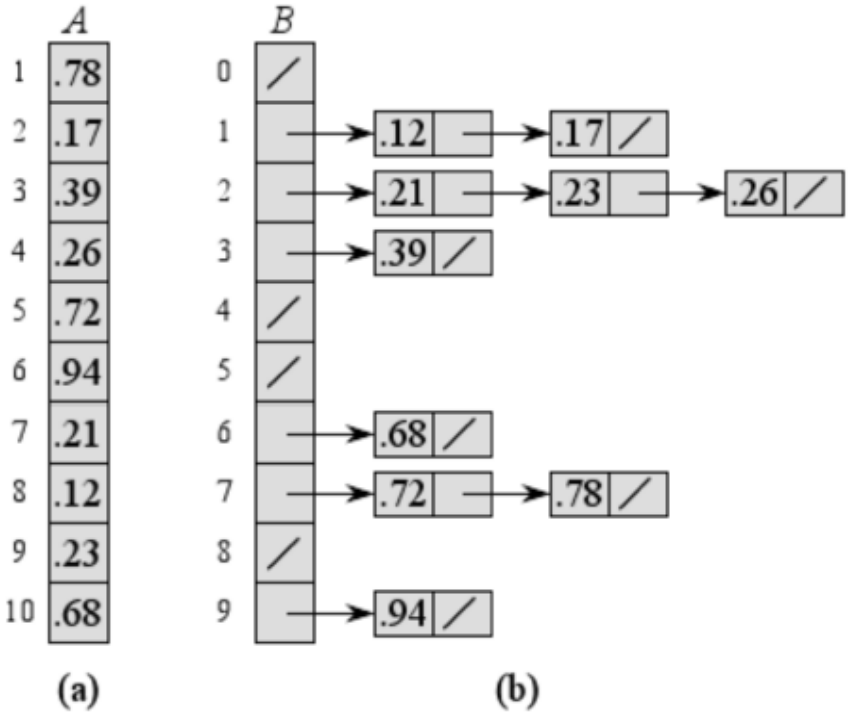
Algorithm bucketSort(S):

Input: Sequence S of entries with integer keys in the range $[0, N-1]$

Output: Sequence S sorted in nondecreasing order of the keys

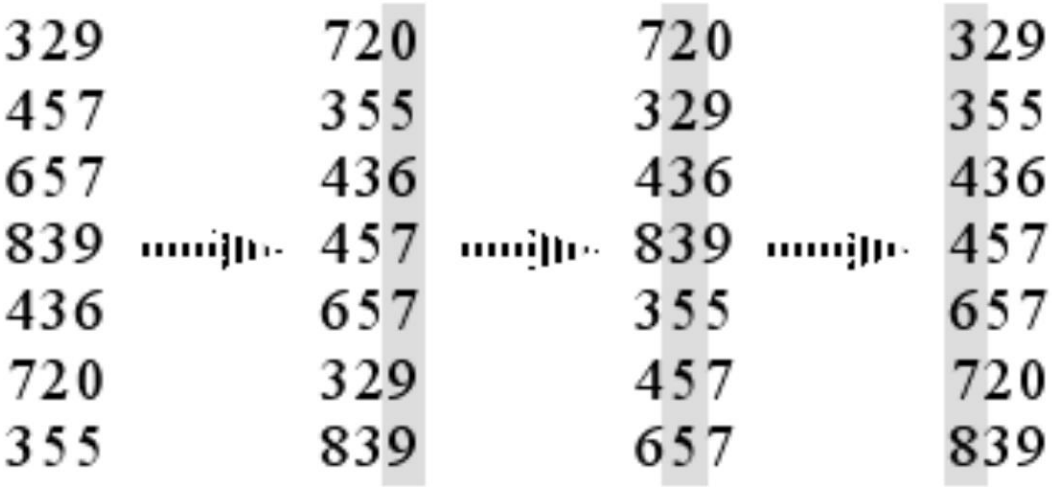
let B be an array of N sequences, each of which is initially empty
for each entry e in S **do**
  $k \leftarrow e$.key()
  remove e from S and insert it at the end bucket (sequence) $B[k]$
for $i \leftarrow 0$ to $N-1$ **do**
  for each entry e in sequence $B[i]$ **do**
    remove e from $B[i]$ and insert it at the end of S

BucketSort




```
BUCKET-SORT(A)
1  n = A.length
2  let B[0..n − 1] be a new array
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```

RadixSort

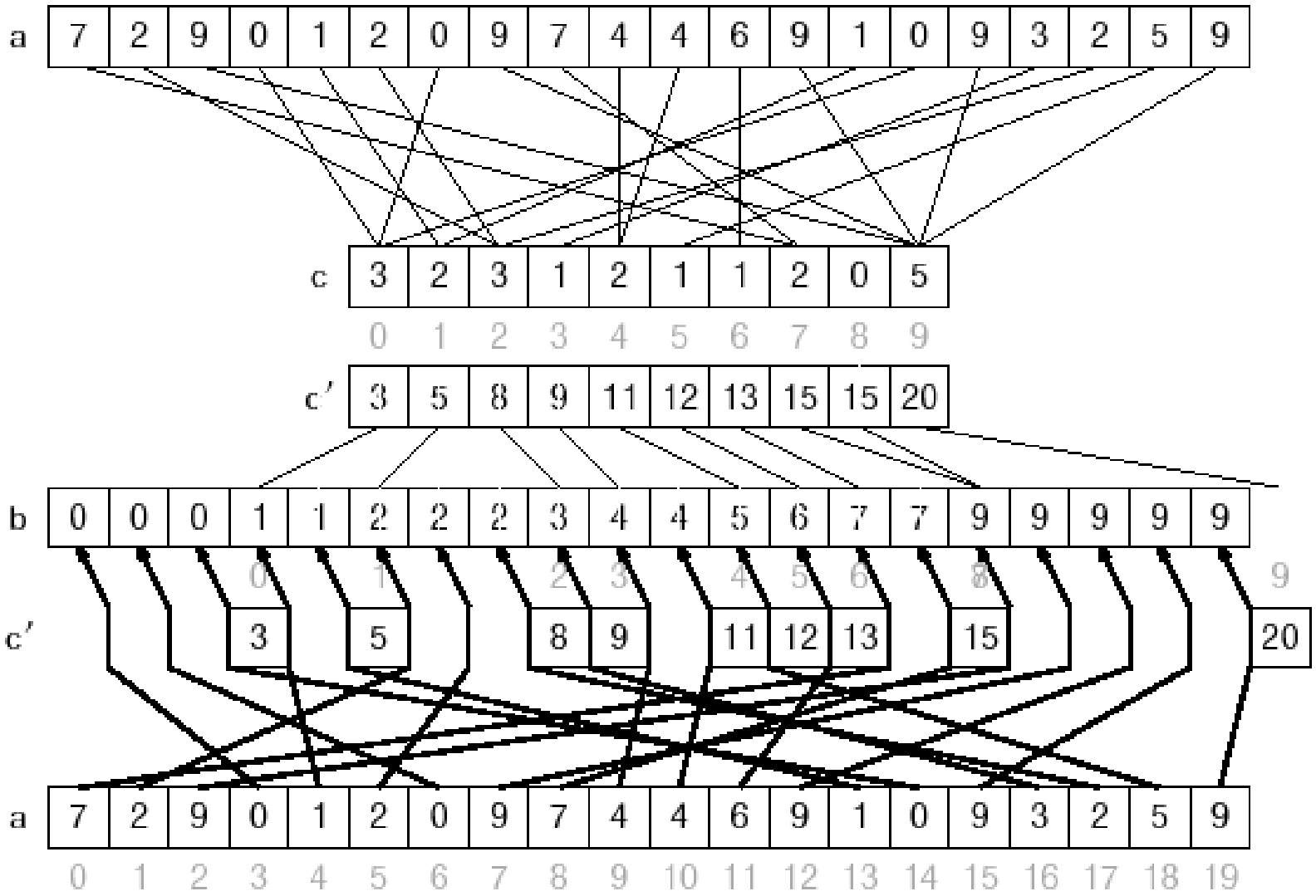


RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d

2 use a stable sort to sort array A on digit i

CountingSort



CountingSort

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Utilidade

- Jogos
- Banco de Dados
- Dicionários Eletrônicos

Referências

- M. T. GOODRICH et al
  - Estruturas de Dados e Algoritmos em Java
  - Data Structures and Algorithms in Java
  - Data Structures and Algorithms in C++
  - Lectures Slides
- H. T. CORMEN et al
  - Introduction to Algorithms
  - Algoritmos - Teoria e Prática
- R. Sedgewick et al
  - An Introduction of the Analysis of Algorithms
  - Algorithms in C++
  - Lectures Slides