7: Space and Time Trade-Offs

Projeto de Análise de Algoritmos

Apresentado por:
Benedito Jaime
Daniel
Érick

Data:
4 de Julho de 2024

Trade-offs entre espaço e tempo

Situação em que se deve escolher entre usar mais espaço (memória) para reduzir o tempo de execução de um algoritmo ou usar menos espaço e, consequentemente, aumentar o tempo de execução.

Trade-offs entre espaço e tempo

ESPAÇO/MEMÓRIA TEMPO

ESPAÇO/MEMÓRIA TEMPO

Trade-offs entre espaço e tempo

↑ ESPAÇO/MEMÓRIA ↓ TEMPO

↓ ESPAÇO/MEMÓRIA ↑ TEMPO

Ordenação por Contagem

Ordenação por Contagem de Comparações

Para ordenar uma lista, podemos contar o número de elementos menores que cada elemento e usar esses números para determinar as posições corretas dos elementos na lista ordenada. Esse método tem eficiência quadrática devido ao número de comparações necessárias.

```
//Sorts an array by comparison counting
//Input: An array A[0..n − 1] of orderable elements
//Output: Array S[0..n − 1] of A's elements sorted in
nondecreasing order

for i ← 0 to n − 1 do Count[i]← 0
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] < A[j ]
        Count[j ]← Count[j ] + 1
        else Count[i]← Count[i] + 1
for i ← 0 to n − 1 do S[Count[i]]← A[i]
return S
```

Ordenação por Contagem

Ordenação por Contagem de Distribuição

Este método é útil quando os elementos a serem ordenados pertencem a um conjunto pequeno e conhecido de valores. Contamos a frequência de cada valor e usamos essas frequências para posicionar os elementos corretamente na lista ordenada. Este algoritmo é linear em eficiência, assumindo que o intervalo de valores é fixo.

```
//Sorts an array of integers from a limited range by distribution counting
//Input: An array A[0..n − 1] of integers between l and u (l ≤ u)
//Output: Array S[0..n − 1] of A's elements sorted in nondecreasing order

for j ← 0 to u − l do D[j ]← 0
for i ← 0 to n − 1 do D[A[i] − l]← D[A[i] − l] + 1
for j ← 1 to u − l do D[j ]← D[j − 1] + D[j ]
for i ← n − 1 downto 0 do
    • j ← A[i] − l
    • S[D[j ] − 1]← A[i]
    • D[j ]← D[j ] − 1
return S
```

Aprimoramento na correspondência de cadeia de caracteres

1 Redução de tempo

2 Pré-processamento do padrão

Algoritmos de Força-Bruta

Algoritmos de Boyler-Moore

Algoritmos de Horspool

Algoritmo de Força Bruta

Peso (Complexidade de espaço):

- Leve: Requer apenas espaço para armazenar o padrão e o texto, sem necessidade de estruturas adicionais.
- Complexidade de espaço: *O(1)*

Velocidade (Complexidade de tempo):

- Pior caso: $O(m \cdot n)$ onde m é o comprimento do padrão e n é o comprimento do texto.
- Melhor caso: O(n) quando o padrão é encontrado na primeira posição do texto.

Algoritmo Knuth-Morris-Pratt

Peso (Complexidade de Espaço):

- Moderado: Requer espaço adicional para armazenar a tabela de prefixos (tabela de falhas).
- Complexidade de espaço: O(m), onde m é o comprimento do padrão.

Velocidade (Complexidade de Tempo):

- Pior caso: O(m+n).
- Melhor caso: O(m+n).

Algoritmo Boyer-Moore

Peso (Complexidade de Espaço):

- Moderado a pesado: Requer espaço adicional para armazenar tabelas de heurísticas (Bad Character e Good Suffix).
- Complexidade de espaço: $O(m+\sigma)$

Velocidade (Complexidade de Tempo):

- Pior caso: $O(m \cdot n)$.
- Melhor caso: $O(n/m)$ em média.
- Na prática, muitas vezes muito rápido devido aos grandes saltos proporcionados pelas heurísticas.

Algoritmo Horspool

Peso (Complexidade de Espaço):

- Moderado: Requer uma tabela de deslocamento baseada nos caracteres do alfabeto.
- Complexidade de espaço: $O(\sigma)$, onde σ é o tamanho do alfabeto.

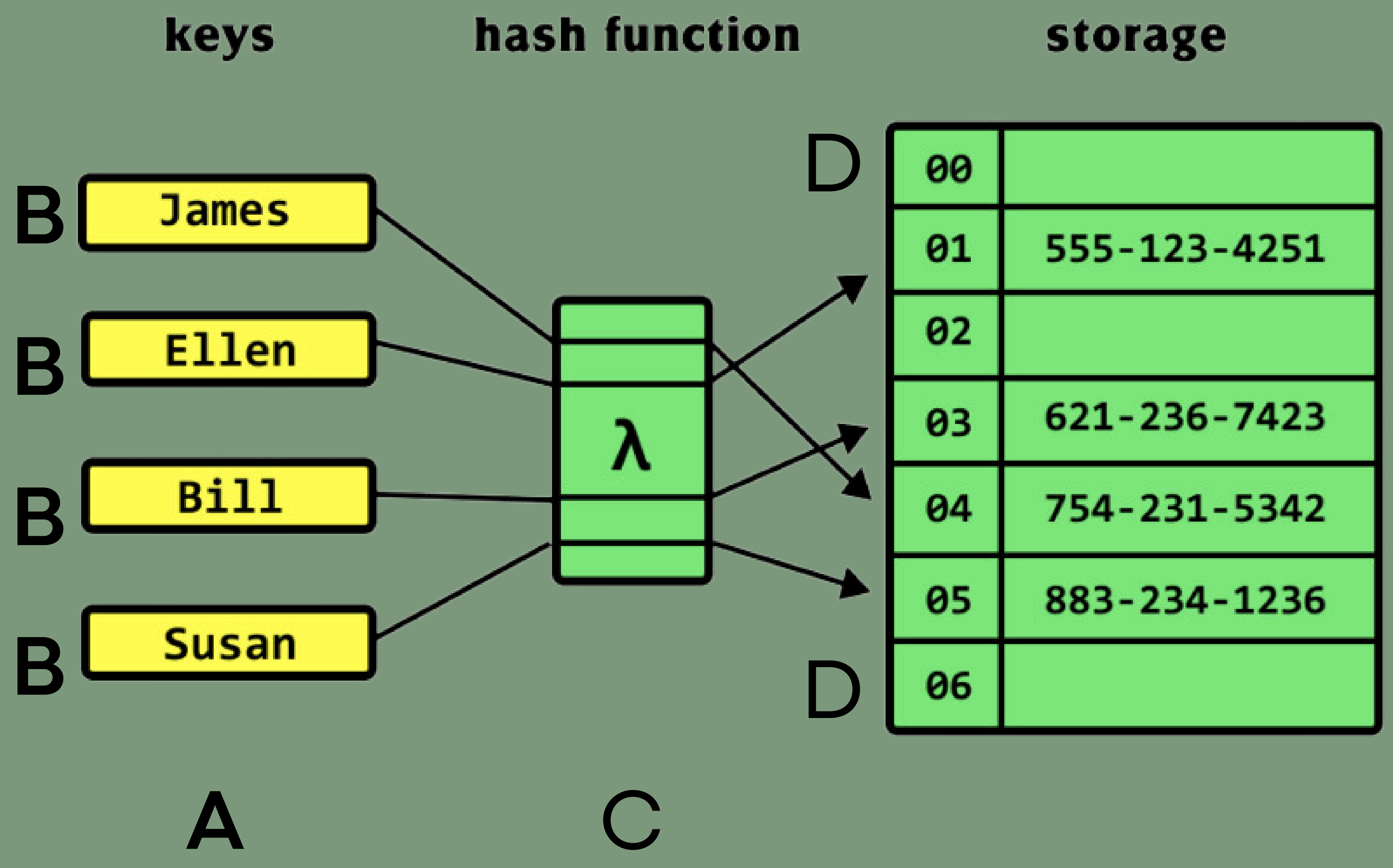
Velocidade (Complexidade de Tempo):

- Pior caso: $O(m \cdot n)$, mas é raro.
- Melhor caso: $O(n/m)$ em média, semelhante ao Boyer-Moore.
- Na prática, é eficiente para textos grandes e padrões que aparecem raramente, embora geralmente não tão eficiente quanto o Boyer-Moore completo.

HASHING

ABORDAGEM EFICIENTE PARA IMPLEMENTAR DICIONÁRIOS.

A: Tabela Hash
B: Chaves
C: Função Hash
D: Endereço Hash



HASHING - Colisões

FENÔMENO NO QUAL DUAS OU MAIS CHAVES SÃO “HASHEADAS” PARA A TABELA.

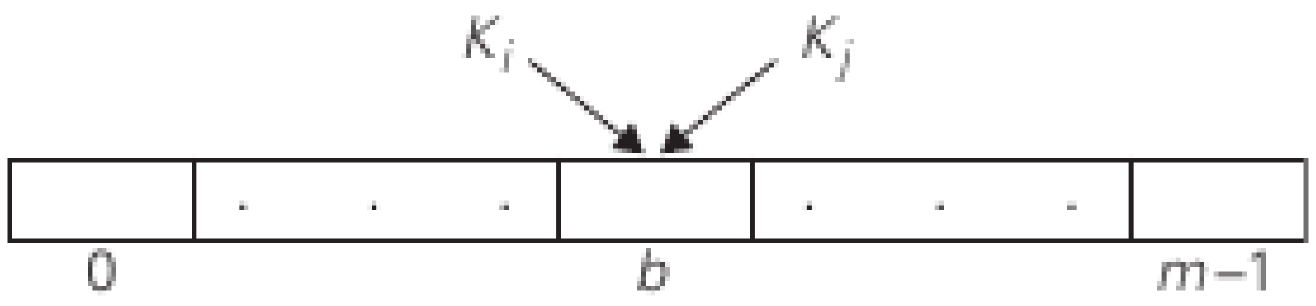
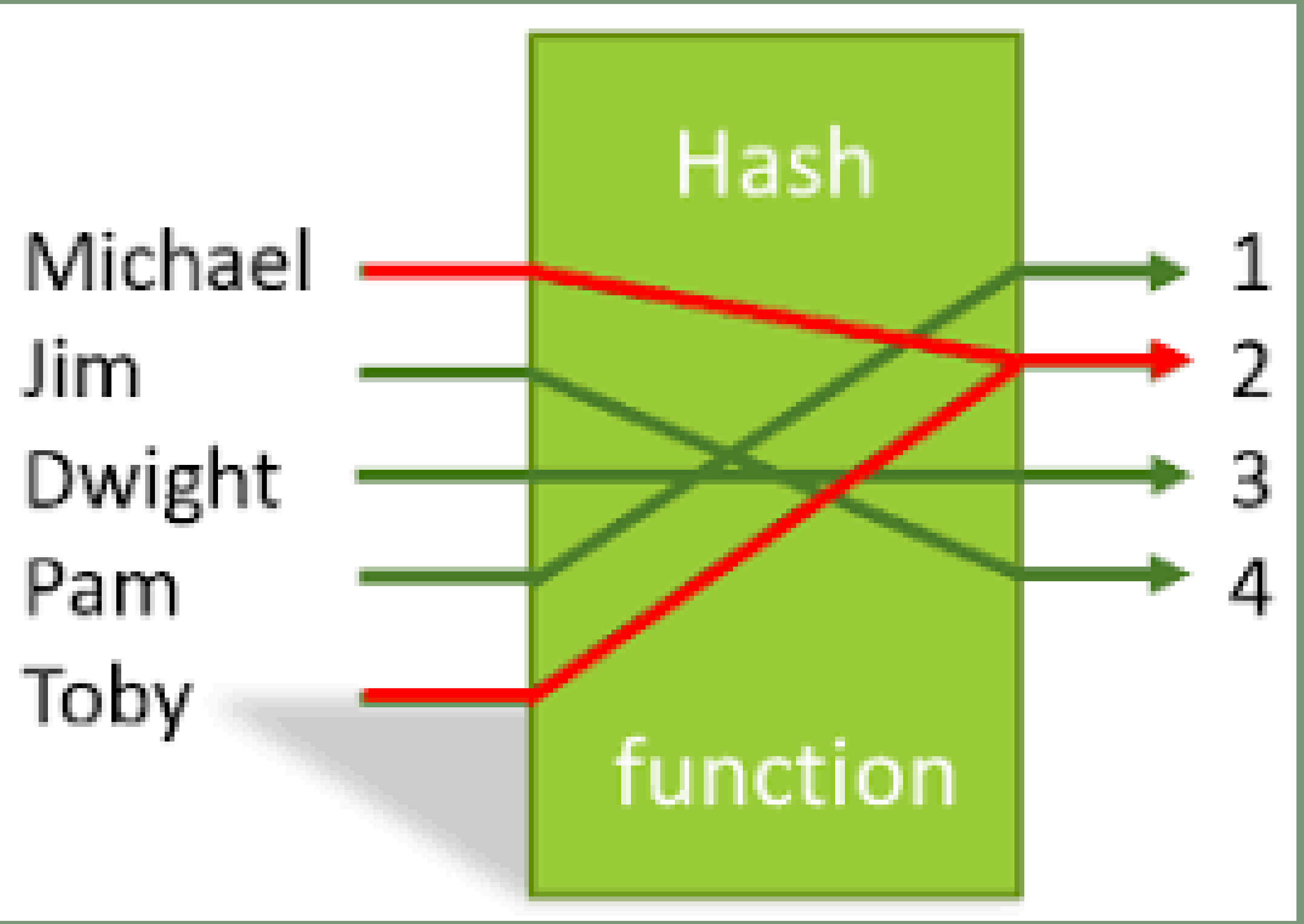


FIGURE 7.4 Collision of two keys in hashing: $h(K_i) = h(K_j)$.



Mecanismo de Resolução de Colisão

OPEN HASHING

A	(1) mod 13	1
FOOL	(6 + 15 + 15 + 12) mod 13	9
AND	(1 + 14 + 1) mod 13	6
HIS	(8 + 9 + 19) mod 13	10
MONEY	(13 + 15 + 14 +5 +25) mod 13	7
ARE	(1 + 18 + 5) mod 13	11
SOON	(19 + 15 + 15 + 14) mod 13	11
PARTED	(16 + 1 + 18 + 20 + 5 + 4) mod 13	12

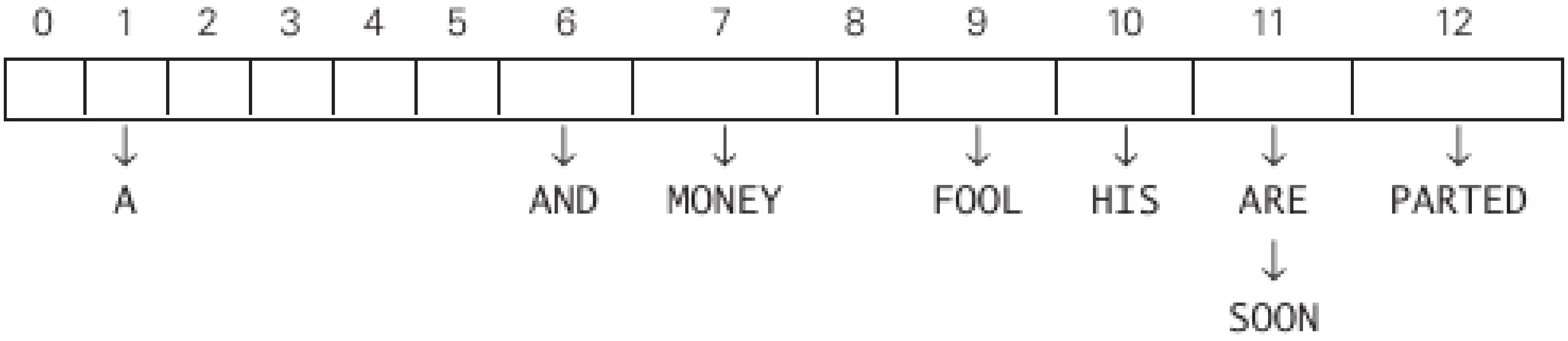
separate chaining

encadeamento separado

COLISÃO

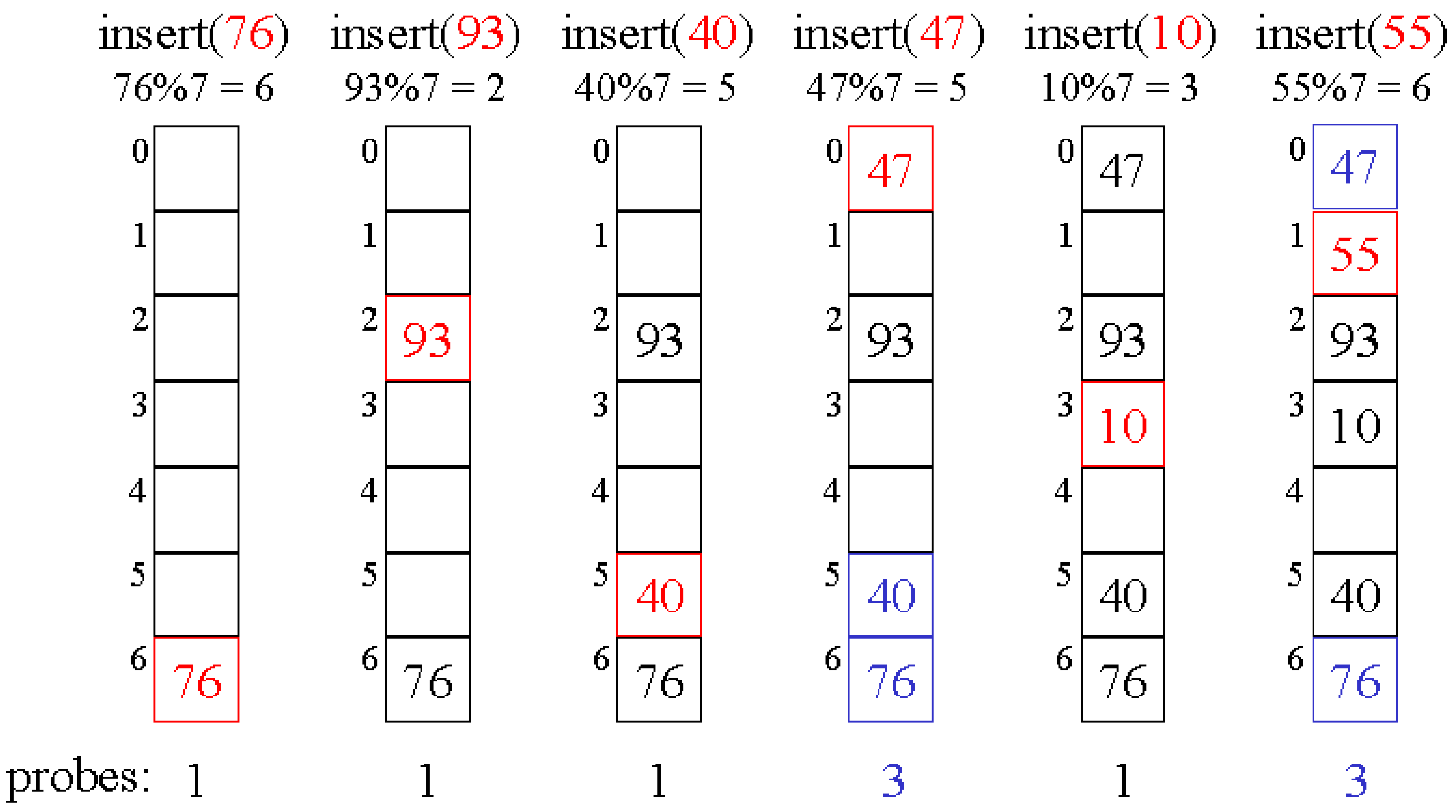
OPEN HASHING

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12



CLOSED HASHING - LINEAR PROBING

Linear Probing Example



CLOSED HASHING

open addressing
endereçamento aberto

SEM USO DE LISTAS ENCANDEADAS

TAMANHO DA TABELA DEVE SER QUASE TÃO LARGA QUANTO O NÚMERO DE CHAVES

LINEAR PROBING - SONDAGEM LINEAR

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

HASHING

Time complexity in big O notation		
Operation	Average	Worst case
Search	$\Theta(1)$	$O(n)$
Insert	$\Theta(1)$	$O(n)$
Delete	$\Theta(1)$	$O(n)$
Space complexity		
Space	$\Theta(n)$[1]	$O(n)$

ÁRVORES B

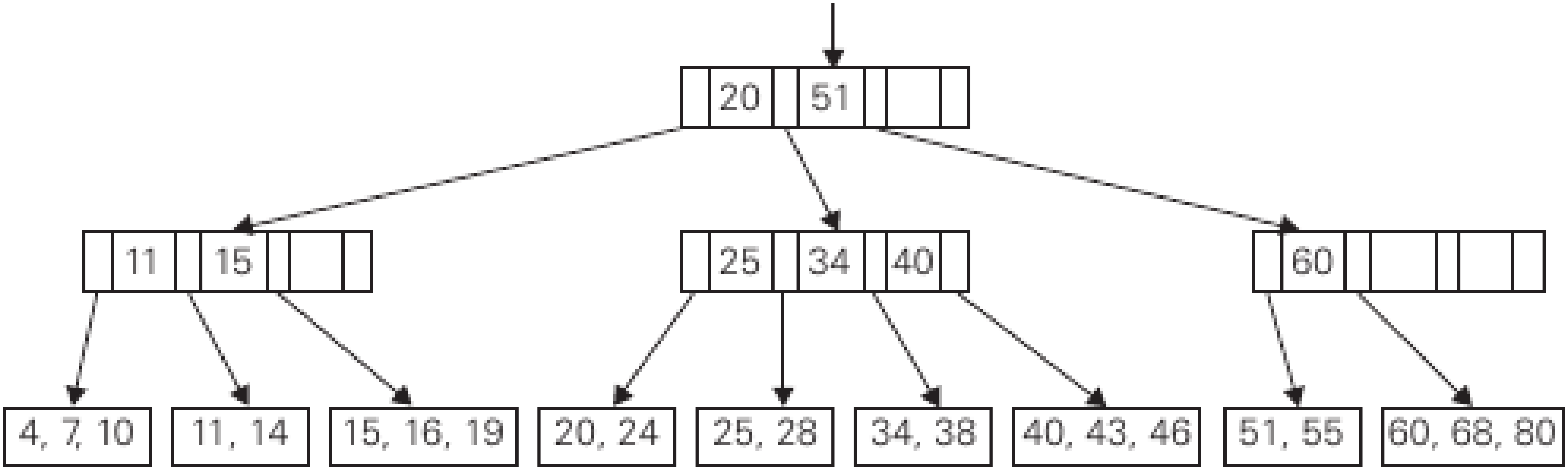
PRINCIPAL ALTERNATIVA PARA ARMAZENAMENTO DE GRANDES DICIONÁRIOS

ÁRVORE DE BUSCA BALANCEADA

MÚLTIPLAS CHAVES POR NÓ

Time complexity in big O notation		
Operation	**Average**	**Worst case**
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$
Space complexity		
Space	$O(n)$	$O(n)$

ÁRVORES B



20
51
11
15
25
34
40
60
4, 7, 10
11, 14
15, 16, 19
20, 24
25, 28
34, 38
40, 43, 46
51, 55
60, 68, 80

HASHING

ESPAÇO ADICIONAL PARA ARMAZENAR UMA TABELA HASH E RESOLVER COLISÕES

OPERAÇÕES RÁPIDAS EM TEMPO CONSTANTE NA MÉDIA

ÁRVORES B

ESPAÇO ADICIONAL PARA ARMAZENAR MÚLTIPLAS CHAVES POR NÓ E MANTER A ÁRVORE BALANCEADA

OPERAÇÕES EM TEMPO LOGARÍTMICO

HASHING

Time complexity in big O notation		
Operation	Average	Worst case
Search	$\Theta(1)$	$O(n)$
Insert	$\Theta(1)$	$O(n)$
Delete	$\Theta(1)$	$O(n)$
Space complexity		
Space	$\Theta(n)$[1]	$O(n)$

ÁRVORES B

Time complexity in big O notation		
Operation	Average	Worst case
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$
Space complexity		
Space	$O(n)$	$O(n)$