





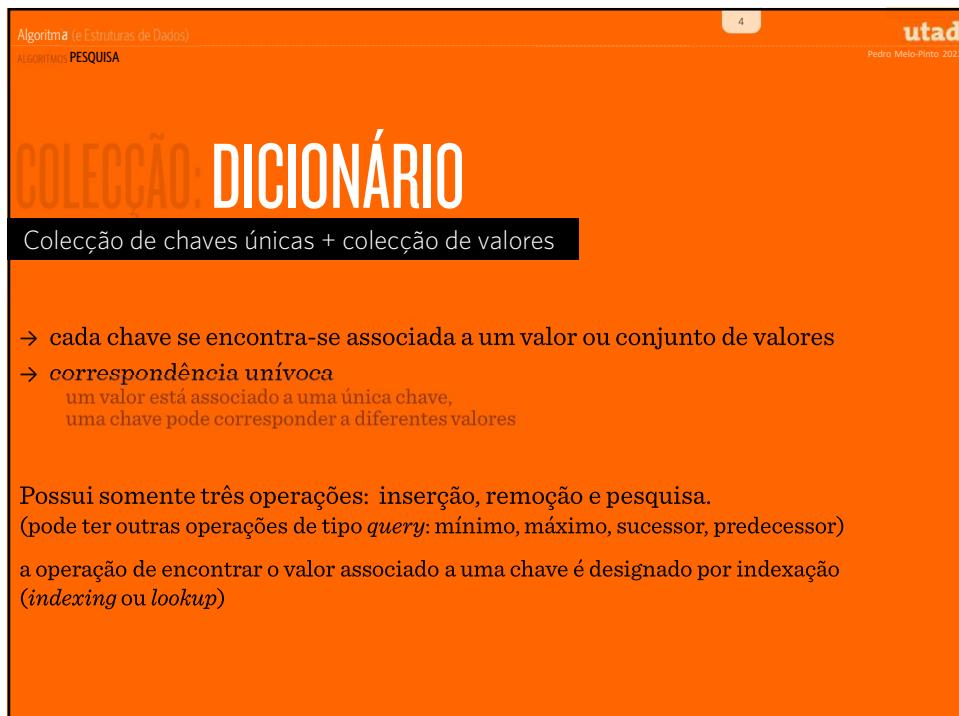
Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

3 utad
Pedro Melo-Pinto 2022

COLEÇÃO: DICIONÁRIO

Coleção de chaves únicas + coleção de valores

ARRAY ASSOCIATIVO, MAPEAMENTO, TABELA SIMBÓLICA



Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

4 utad
Pedro Melo-Pinto 2022

COLEÇÃO: DICIONÁRIO

Coleção de chaves únicas + coleção de valores

- cada chave se encontra-se associada a um valor ou conjunto de valores
- **correspondência unívoca**
 - um valor está associado a uma única chave,
 - uma chave pode corresponder a diferentes valores

Possui somente três operações: inserção, remoção e pesquisa.
(pode ter outras operações de tipo *query*: mínimo, máximo, sucessor, predecessor)

a operação de encontrar o valor associado a uma chave é designado por indexação
(*indexing* ou *lookup*)

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

5 utad
Pedro Melo-Pinto 2022

COLEÇÃO: DICIONÁRIO

Coleção de chaves únicas + coleção de valores

application	purpose of search	key	value
dictionary	find definition	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
financial account	process transactions	account number	transaction details
web search	find relevant web pages	keyword	list of page names
compiler	find properties of variables	variable name	type and value
routing table	route Internet packets	destination	best route
DNS	find IP address given URL	URL	IP address
reverse DNS	find URL given IP address	IP address	URL
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

6 utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

A diferença no acesso aos dados está no uso de uma tabela de indexação (*key-indexed table*).
Este endereçamento é feito através de um índice (geralmente de um *array*) que é função da chave de pesquisa.
Idealmente, este endereçamento (indexação) é directo.

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

03 TABELAS DE DISPERSÃO

Um array é uma tabela de endereçamento directo.
A cada posição da tabela (slot) corresponde uma chave (key) no universo U.
Permite o acesso a qualquer posição da tabela em O(1).

Funciona bem para universos U (de chaves) pequenos.
Se o conjunto de chaves realmente usadas é (muito) inferior ao número de slots a utilização de memória é ineficaz.

The diagram shows a circular set U (universe of keys) containing points labeled 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. A smaller circle K (actual keys) contains points 2, 3, 5, 8. Arrows point from each key in K to its corresponding slot index in a 10-slot array T . The array T has slots indexed from 0 to 9. To the right of the array, three operations are listed with their time complexities:

- directAddressInsert(T, x) $O(1)$
- directAddressRemove(T, x) $O(1)$
- directAddressSearch(T, k) $O(1)$

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

03 TABELAS DE DISPERSÃO

Um array é uma tabela de endereçamento directo.
A cada posição da tabela (slot) corresponde uma chave (key) no universo U.
Permite o acesso a qualquer posição da tabela em O(1).

Funciona bem para universos U (de chaves) pequenos.

The diagram shows a circular set U (universe of keys) containing points labeled 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. A smaller circle K (actual keys) contains points 2, 3, 5, 8. Arrows point from each key in K to its corresponding slot index in a 10-slot array T . The array T has slots indexed from 0 to 9. To the right of the array, a note states:

Se o universo U (de chaves) for muito grande, uma tabela T de tamanho $|U|$ é impraticável, ou até impossível.

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

9
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

Um *array* é uma tabela de endereçamento directo.
Permite o acesso a qualquer posição da tabela em $O(1)$.
Funciona bem para universos U (de chaves) pequenos.

Uma tabela de dispersão é uma generalização desta noção.
O índice (do array) é calculado com o uso de uma função a partir da chave.

A ideia é reduzir o número m de slots da tabela de forma a tornar praticável o espaço de memória necessário: $|m|$

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

10
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

No endereçamento directo um elemento com a chave i é guardado no lugar i da tabela T .

No endereçamento por *hashing*, um elemento com a chave i é guardado no lugar $h(i)$ da tabela de dispersão, ou seja, é utilizada uma função de dispersão h (*hash*) para endereçar o elemento de chave i na tabela T .

endereçamento directo endereçamento por hashing

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

11 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

No endereçamento directo um elemento com a chave i é guardado no lugar i da tabela T .

No endereçamento por *hashing*, um elemento com a chave i é guardado no lugar $h(i)$ da tabela de dispersão, ou seja, é utilizada uma função de dispersão h (*hash*) para endereçar o elemento de chave i na tabela T .

The diagram shows a mapping from a set of keys U (universe of keys) to an array T . A subset K (actual keys) is shown. The array T has indices 0 to $m-1$. The mapping is defined by a function h such that $h(k_i) = i$. For example, $k_1 \rightarrow h(k_1)$, $k_2 \rightarrow h(k_2)$, and $k_3 \rightarrow h(k_3)$. Below this, a specific mapping is shown for names to array indices:

chaves	funcção hash	array
Carlos Almeida	$h(k_1)$	01 919 999 999
Luisa Almeida	$h(k_2)$	05 936 666 999
Sandra Chaves	$h(k_3)$	14 935 555 000

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

12 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

No endereçamento directo um elemento com a chave i é guardado no lugar i da tabela T .

No endereçamento por *hashing*, um elemento com a chave i é guardado no lugar $h(i)$ da tabela de dispersão, ou seja, é utilizada uma função de dispersão h (*hash*) para endereçar o elemento de chave i na tabela T .

A função h faz corresponder (mapeia) o universo U (de chaves) numa tabela de dispersão T $[0, m - 1]$:

$$h : U \rightarrow \{0, 1, \dots, m - 1\}$$

(em vez de $|U|$ valores, lidamos somente com m)

The diagram shows a mapping from a set of keys U (universe of keys) to an array T . A subset K (actual keys) is shown. The array T has indices 0 to $m-1$. The mapping is defined by a function h such that $h(k_i) = i$. For example, $k_1 \rightarrow h(k_1)$, $k_2 \rightarrow h(k_2)$, $k_3 \rightarrow h(k_3)$, and $k_4 \rightarrow h(k_4)$.

Adaptado do Gómez T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

13 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

No endereçamento directo um elemento com a chave i é guardado no lugar i da tabela T .

No endereçamento por *hashing*, um elemento com a chave i é guardado no lugar $h(i)$ da tabela de dispersão, ou seja, é utilizada uma função de dispersão h (*hash*) para endereçar o elemento de chave i na tabela T .

As necessidades de memória podem ser reduzidas para $O(|m|)$ mantendo o tempo da pesquisa em $O(1)$ em média (enquanto que para endereçamento directo este tempo se mantém para o pior caso).

Adaptado de Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

14 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

No endereçamento por *hashing*, um elemento com a chave i é guardado no lugar $h(i)$ da tabela de dispersão, ou seja, é utilizada uma função de dispersão h (*hash*) para endereçar o elemento de chave i na tabela T .

Como $m < |U|$ vamos ter diferentes chaves a corresponder ao mesmo lugar na tabela de dispersão → colisões.

Adaptado de Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

15

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

No endereçamento por *hashing*, um elemento com a chave i é guardado no lugar $h(i)$ da tabela de dispersão, ou seja, é utilizada uma função de dispersão h (*hash*) para endereçar o elemento de chave i na tabela T .

Como $m < |U|$ vamos ter diferentes chaves a corresponder ao mesmo lugar na tabela de dispersão → **colisões**.

Exemplo:

Seja o universo U dos números naturais menores do que 1000000000 (como é o caso do NIF).
Seja a tabela de hash, cuja função h é a soma dos dígitos da chave original ($m = 81$).

$h(24277) = 2+4+2+7+7$	= 22
$h(997)$	= 25
$h(100421006)$	= 14
...	
$h(778)$	= 22
$h(142312531)$	= 22

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

16

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

Análise Geral

O método de endereçamento directo acede a qualquer elemento em tempo $O(1)$.
Para o caso do uso de tabelas de dispersão, temos:

Caso mais favorável
Não há colisões → $O(1)$

Caso pior
Todas as chaves colidem no mesmo lugar da tabela de dispersão. Ficamos com uma pesquisa sequencial → $O(n)$

Em média
Depende do número de colisões (e da técnica usada para as ultrapassar) → $O(1) + n$

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

17

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO

Análise Geral

O método de endereçamento directo acede a qualquer elemento em tempo $O(1)$. Para o caso do uso de tabelas de dispersão, temos:

- Caso mais favorável**
Não há colisões $\rightarrow O(1)$
- Caso pior**
Todas as chaves colidem no mesmo lugar da tabela de dispersão. Ficamos com uma pesquisa sequencial $\rightarrow O(n)$
- Em média**
Depende do número de colisões (e da técnica usada para as ultrapassar) $\rightarrow O(1) + n$

A eficácia do método depende do **número de colisões** existente.

O número de colisões depende da **função de dispersão** (e, claro, do **tamanho da tabela, comparativamente ao tamanho do universo U**).

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

18

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Uma boa função de dispersão satisfaz o seguinte pressuposto (*hashing uniforme simples*):
cada chave tem a mesma probabilidade de corresponder a cada elemento da tabela,
independentemente do que acontece às outras chaves.

We have three primary requirements in implementing a good hash function for a given data type:
 It should be *consistent* – equal keys must produce the same hash value.
 It should be *efficient to compute*.
 It should *uniformly distribute the keys*.

Sedgewick, R. and Wayne, K., 2011
Algorithms, 4th edition

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

19

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
 Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
 A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Uma boa função de dispersão satisfaz o seguinte pressuposto (*hashing uniforme simples*):
 cada chave tem a mesma probabilidade de corresponder a cada elemento da tabela,
 independentemente do que acontece às outras chaves.

Normalmente tal não acontece (não sabemos a distribuição das chaves e estas poderão não ser independentes)
 Recorre-se a heurísticas na criação de funções de dispersão que tenham bom desempenho.
 A informação qualitativa (quando existe) acerca da distribuição é útil neste processo (v. chaves alfanuméricas).
 Uma boa abordagem é a utilização de funções que se creem independentes de qualquer tipo de padrão existente
 (v. método da divisão).

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

20

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
 Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
 A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Método da multiplicação

- ➊ Multiplicar a chave k por uma constante a e ficar com a respectiva parte fraccionária
- ➋ Multiplicar este valor por m e obter o piso (*floor*) deste valor

Função de dispersão:

$$h(k) = \lfloor m(ak \bmod 1) \rfloor = \lfloor m(ak - \lfloor ak \rfloor) \rfloor$$

m não é crítico
 valor típico de $m : 2^p$, em que p é um valor inteiro

a função h resulta melhor com alguns valores de a do que com outros
 a melhor escolha depende dos dados em questão
 D. Knuth propõe $a \approx \frac{(\sqrt{5}-1)}{2} = 0.6180339887 \dots$

Knuth, D., 1973
 Sorting and Searching, volume 3 of The Art of Computer Programming.

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

21 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (array).

Método da multiplicação

Função de dispersão:

$$h(k) = \lfloor m(ak \bmod 1) \rfloor = \lfloor m(ak - \lfloor ak \rfloor) \rfloor$$

exemplo (1000 valores inteiros aleatórios entre 1 e 10000):

Four histograms illustrating the distribution of hash values for different values of 'a':

- $h(k) = \lfloor m(ak \bmod 1) \rfloor - a: 0,15 \quad m: 100$
- $h(k) = \lfloor m(ak \bmod 1) \rfloor - a: 0,33 \quad m: 100$
- $h(k) = \lfloor m(ak \bmod 1) \rfloor - a: 0,6180 \quad m: 100$
- $h(k) = \lfloor m(ak \bmod 1) \rfloor - a: 0,79 \quad m: 100$

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

22 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (array).

Método da divisão (modular hashing)

- 1 Escolher o tamanho m da tabela
- 2 Ficar com o resto da divisão de k por m

Função de dispersão:

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

Devem evitarse certos valores de m:
 se $m = 2^p$, $h(k)$ representa os p bits menos significativos de k (na base 2)
 um número primo não muito perto de um valor 2^p , é habitualmente uma boa escolha

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

23

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Método da divisão (*modular hashing*) – variante D. Knuth

- ➊ Escolher o tamanho m da tabela
- ➋ Ficar com o resto da divisão de k por m

Função de dispersão:

$$h(k) = k(k + 3) \bmod m$$

Devem evitarse certos valores de m:
 se $m = 2^p$, $h(k)$ representa os p bits menos significativos de k (na base 2)
 um número primo não muito perto de um valor 2^p , é habitualmente uma boa escolha

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

24

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Método da divisão (*modular hashing*)

- ➊ Escolher o tamanho m da tabela
- ➋ Ficar com o resto da divisão de k por m

Função de dispersão:

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

Devem evitarse certos valores de m:
 se $m = 2^p$, $h(k)$ representa os p bits menos significativos de k (na base 2)
 um número primo não muito perto de um valor 2^p , é habitualmente uma boa escolha

Funciona mal para muitos tipos de padrão existentes nos dados.

Algoritma (e Estruturas de Dados)
AUTORITMOS PESQUISA

25 utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (array).

Método da divisão (*modular hashing*)

Função de dispersão:
 $h(k) = ak \bmod m$ - a pode entender-se como uma generalização do $k+3$ proposto por D. Knuth

exemplo (1000 valores inteiros aleatórios entre 1 e 10000):

Algoritma (e Estruturas de Dados)
AUTORITMOS PESQUISA

26 utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (array).

Método da divisão (*modular hashing*)

Função de dispersão:
 $h(k) = ak \bmod m$ - a pode entender-se como uma generalização do $k+3$ proposto por D. Knuth

exemplo (1000 valores inteiros aleatórios múltiplos de 5 entre 1 e 10000):

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

27

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Floating-point numbers. If the keys are real numbers between 0 and 1, we might just multiply by M and round off to the nearest integer to get an index between 0 and $M-1$. Although this approach is intuitive, it is defective because it gives more weight to the most significant bits of the keys; the least significant bits play no role. One way to address this situation is to use modular hashing on the binary representation of the key (this is what Java does).

Sedgewick, R. and Wayne, K., 2011.
Algorithms, 4th edition

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

28

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (*array*).

Strings
As funções de dispersão por divisão funcionam bem com strings.

Função de dispersão:
 $h(k) = ak \bmod m$, em que m é o tamanho da tabela

Exemplo:
Cada string está representada por um inteiro, em octal (coluna 2) e decimal (coluna 3), resultante do somatório dos valores ASCII dos seus caracteres, utilizando a função de hash anterior, com valores de $m = 64$ e 31 respectivamente (colunas 4 e 5)

$\text{now} = 110 \times 128^2 + 111 \times 128^1 + 119 \times 128^0 = 1816567$

$1816567 \bmod 64 = 55$
 $1816567 \bmod 31 = 29$

		m	m	m
now	6733767	1816567	55	29
for	6333762	1685490	50	20
tip	7223260	1914096	48	1
ilk	6473153	1734251	43	18
din	6223256	1651949	45	21
tag	7230347	1913063	39	22
jot	6533764	1751028	52	24
sob	7173742	1898466	34	26
nob	6733742	1816564	34	8
sky	7172771	1897977	57	2
hat	6435564	1719050	52	16
act	6044645	1662021	37	3
bet	6131364	1618676	46	11
men	6671354	1798894	46	26
egg	6271747	1668071	39	23
few	6331367	1684215	55	16
jay	6530371	1749241	57	4
owl	6775754	1833964	44	4
joy	6533771	1751033	57	29
rap	7130360	1880304	48	30
gig	6372347	1701095	39	1
wee	7371347	1962275	37	22
was	7370363	1962227	51	20
cab	6170342	1634550	34	24
wad	7370344	1962212	36	5

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

29
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (array).

Strings
As funções de dispersão por divisão funcionam bem com strings.

Função de dispersão:
 $h(k) = ak \bmod m$, em que m é o tamanho da tabela

Exemplo:
(primeiras 1000 palavras diferentes do livro Moby Dick)

EX1: $m = 96$, $a = 128$
EX2: $m = 97$, $a = 128$
EX3: $m = 96$, $a = 127$

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

30
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO FUNÇÕES DE DISPERSÃO

Como $m < |U|$ é impossível evitar colisões.
Uma função de dispersão h bem escolhida, minimiza o número de colisões existente.
A função de dispersão h deve transformar qualquer chave num índice da tabela (array).

Strings
As funções de dispersão por divisão funcionam bem com strings.

Função de dispersão:
 $h(k) = ak \bmod m$, em que m é o tamanho da tabela

Exemplo:
(palavras diferentes do livro A Tale of Two Cities)

10 679 palavras diferentes
 $m = 97$

110 ≈ 10679 / 97

Hash value frequencies for words in Tale of Two Cities [10,679 keys, M = 97]
Sedgewick, R. and Wayne, K., 2011. Algorithms, 4th edition

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

31 utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Encadeamento (Separate Chaining)

Os elementos que colidem são colocados numa lista encadeada, cuja cabeça é o correspondente *slot* da tabela.
Se determinado *slot* não tiver elementos o seu valor é nulo.

O processo de pesquisa inclui a determinação do valor de *hash* e a pesquisa sequencial na lista encadeada correspondente.

Adaptado de Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

32 utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Encadeamento (Separate Chaining)

Os elementos que colidem são colocados numa lista encadeada, cuja cabeça é o correspondente *slot* da tabela.
Se determinado *slot* não tiver elementos o seu valor é nulo.

O processo de pesquisa inclui a determinação do valor de *hash* e a pesquisa sequencial na lista encadeada correspondente.

Adaptado de Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA
33
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ **Encadeamento** (*Separate Chaining*)

Os elementos que colidem são colocados numa lista encadeada, cuja cabeça é o correspondente *slot* da tabela.
Se determinado *slot* não tiver elementos o seu valor é nulo.

O processo de pesquisa inclui a determinação do valor de *hash* e a pesquisa sequencial na lista encadeada correspondente.

Análise Global:
se a distribuição das chaves for uniforme
eficácia média = função do número médio de elementos nas listas encadeadas
o seu desempenho degrada-se linearmente com este valor
uma tabela de dispersão com 1000 *slots* e 10000 chaves é 5-10 vezes mais lenta do que uma com 10000 *slots* e 10000 chaves

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA
34
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ **Encadeamento** (*Separate Chaining*)

Os elementos que colidem são colocados numa lista encadeada, cuja cabeça é o correspondente *slot* da tabela.
Se determinado *slot* não tiver elementos o seu valor é nulo.

O processo de pesquisa inclui a determinação do valor de *hash* e a pesquisa sequencial na lista encadeada correspondente.

Existem outras opções para guardar os elementos que colidem:

- *Listas ordenadas*
diminuem para aproximadamente metade o custo médio de procurar um elemento que não esteja no conjunto – caso pior
- *BSTs auto-equilibradas*

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

35 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Encadeamento (*Separate Chaining*)

Os elementos que colidem são colocados numa lista encadeada, cuja cabeça é o correspondente slot da tabela. Se determinado slot não tiver elementos o seu valor é nulo.

O processo de pesquisa inclui a determinação do valor de hash e a pesquisa sequencial na lista encadeada correspondente.

Análise:

chainedHashInsert(T, x) insert x at head of list $T[h(x.k)]$	$O(1)$
chainedHashRemove(T, x) delete x from the list $T[h(x.k)]$	$O(1)$
chainedHashSearch(T, k) search for an element with key k in list $T[h(k)]$	$O(n_m)$

n_m : número médio de elementos nas listas encadeadas

x, k é a chave de pesquisa de x

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

36 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Encadeamento (*Separate Chaining*)

Análise:
Factor de carga de uma tabela de dispersão T :
 $\alpha = n/m$,
em que n é o número de chaves/elementos na tabela e m é o número de slots

Caso-pior:
Todas as n chaves caem no mesmo slot.

A eficácia é $\Theta(n) + \text{tempo necessário para o cálculo da função de dispersão } h$

$O(1)$ normalmente

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

37

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Encadeamento (*Separate Chaining*)

Análise:

Factor de carga de uma tabela de dispersão T :
 $\alpha = n/m$,
em que n é o número de chaves/elementos na tabela e m é o número de slots

Caso-médio:

Depende do número de colisões.
Vamos assumir uma distribuição uniforme das chaves (*simple uniform hashing*).
O valor esperado do comprimento da lista encadeada j é:
 $E[n_j] = \alpha = n/m$

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

38

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Encadeamento (*Separate Chaining*)

Análise:

Factor de carga de uma tabela de dispersão T :
 $\alpha = n/m$,
em que n é o número de chaves/elementos na tabela e m é o número de slots

Caso-médio:

Depende do número de colisões.
O valor esperado do comprimento da lista encadeada j é:
 $E[n_j] = \alpha = n/m$

Theorem. In a hash table in which collisions are resolved by chaining, an **unsuccessful** search takes average-case time $\Theta(1 + \alpha)$, under the assumption of simple uniform hashing.

$\Theta(1)$ para cálculo de $h(k) + \Theta(\alpha)$ para percorrer a lista = $\Theta(1+\alpha)$

Theorem. In a hash table in which collisions are resolved by chaining, a **successful** search takes average-case time $\Theta(1 + \alpha - \frac{\alpha}{2n})$, under the assumption of simple uniform hashing.

$\Theta(1)$ para cálculo de $h(k) + \Theta(1 + \frac{\alpha}{2} - \frac{\alpha}{2n})$ para encontrar o elemento = $\Theta(1+\alpha - \frac{\alpha}{2n})$

Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009
Introduction to Algorithms, 3rd edition.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

39

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ **Encadeamento** (*Separate Chaining*)

Análise:

Factor de carga de uma tabela de dispersão T :

$$\alpha = n/m,$$

em que n é o número de chaves/elementos na tabela e m é o número de slots

Caso-médio:

Depende do número de colisões.

Para o caso de distribuição uniforme das chaves (*simple uniform hashing*) é $O(1) + \text{tempo de procurar uma chave na lista } T[h(k)] \text{ de comprimento } n_{h(k)}$

$$\Theta(1 + \alpha)$$

Se o número de slots m for proporcional ao número de elementos n , então $n=O(m)$ e

$$\alpha = \frac{n}{m} = \frac{O(m)}{m} = O(1)$$

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

40

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ **Reposiciónamento** (*Open Addressing*)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos slots da tabela.

Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T .

Para cada elemento que colide é procurada uma nova posição na tabela.

O tempo do processo de pesquisa é a soma do tempo de determinação do valor de hash e do tempo de pesquisa da nova posição do elemento na tabela (se necessário).

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

41

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela.

Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T .
Para cada elemento que colide é procurada uma nova posição na tabela.

array	chaves
00	Luisa Almeida
01	Carlos Almeida
02	Carlos Almeida
03	Paulo Gomes
04	Ricardo Silva
05	
...	...
13	Sandra Chaves
14	Sandra Chaves
15	
16	

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

42

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela.

Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T .
Para cada elemento que colide é procurada uma nova posição na tabela.

Algoritmos

<pre>function openHashInsert pass in: ARRAY INT T, INT k var: INT i,j i ← 0 repeat j ← h(k,i) if T[j] = NIL T[j] ← k return j else i ← i + 1 end if until i = m pass out: error "hash table overflow" end function</pre>	<pre>function openHashSearch pass in: ARRAY INT T, INT k var: INT i,j i ← 0 repeat j ← h(k,i) if T[j] = k return j else i ← i + 1 end if until T[j] = NIL or i = m pass out: NIL end function</pre>
--	---

Eliminar elementos é mais difícil, visto que não basta marcar o *slot* como NIL.
Poderemos, isso sim, marcá-lo como APAGADO, adaptando também openHashInsert(T,k)

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

43

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Linear

O intervalo i entre amostras é uniforme (em geral 1)

Função de dispersão $h(k)$ na exploração linear:
dada uma função de dispersão $h'(k)$

$$h(k,i) = (h'(k) + i) \bmod m,$$
 em que m é o tamanho da tabela e $i = 0, 1, 2, \dots, m-1.$

Primeiro explora-se o slot $T[h'(k)]$
Depois explora-se o slot $T[h'(k) + 1]$
E assim sucessivamente até ao slot $T[m - 1]$
Regressamos ao slot $T[0]$ e vamos até $T[h'(k) - 1]$

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

44

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Linear

O intervalo i entre amostras é uniforme (em geral 1)

Função de dispersão $h(k)$ na exploração linear:
dada uma função de dispersão $h'(k)$

$$h(k,i) = (h'(k) + i) \bmod m,$$
 em que m é o tamanho da tabela e $i = 0, 1, 2, \dots, m-1.$

Sofre de um problema designado por *clustering primário (primary clustering)*: A formação de longas sequências de *slots* ocupados → tempo médio de pesquisa aumenta como a probabilidade de determinado *slot* ser o resultado da função de dispersão é igual para todos os *slots*, as sequências maiores têm mais probabilidades de aumentarem ainda mais

Algoritma (e Estruturas de Dados)
AUTORITMOS PESQUISA

45

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Linear

$$h(k, i) = (h'(k) + i) \bmod m$$

Sofre de um problema designado por *clustering primário (primary clustering)*:

EXEMPLO ($\alpha = 0.7$)

sem clusters



com exploração linear



Algoritma (e Estruturas de Dados)
AUTORITMOS PESQUISA

46

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Quadrática

O intervalo i entre amostras cresce uniformemente

Função de dispersão $h(k)$ na exploração quadrática:
dada uma função de dispersão $h'(k)$

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m,$$

em que m é o tamanho da tabela, c_1 e $c_2 > 0$, e $i = 0, 1, 2, \dots, m-1$. (há quem proponha c_1 e $c_2 \neq 0$)

Primeiro explora-se o slot $T[h'(k)]$

As posições seguintes a explorar vão depender (quadraticamente) de i

- O método funciona melhor do que a exploração linear, mas há que escolher bem c_1 , c_2 e m
- Se duas chaves têm uma posição inicial idêntica na tabela, a sua sequência de exploração é a mesma (tal como na exploração linear a posição inicial define a sequência de exploração)
- Tipicamente “desperdiça” entradas da tabela, que acabam por nunca ser exploradas

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

47

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Quadrática

O intervalo i entre amostras cresce uniformemente

Função de dispersão $h(k)$ na exploração quadrática: dada uma função de dispersão $h'(k)$

$$h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m,$$

em que m é o tamanho da tabela, c_1 e $c_2 > 0$, e $i = 0, 1, 2, \dots, m-1$.

- O método funciona melhor do que a exploração linear, mas há que escolher bem c_1 , c_2 e m

Para $m = 2^n$, uma boa escolha das constantes é $c_1 = c_2 = 1/2$, que geram valores distintos de $h(k,i)$, $i \in [0, m-1]$. As sequências de exploração serão do tipo $h(k), h(k)+1, h(k)+3, h(k)+6, \dots$, com incrementos de 1, 2, 3, ... Se $m > 2$ for primo, grande parte das escolhas de c_1 e c_2 geram valores distintos de $h(k,i)$, $i \in [0, (m-1)/2]$.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

48

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Quadrática

O intervalo i entre amostras cresce uniformemente

Função de dispersão $h(k)$ na exploração quadrática: dada uma função de dispersão $h'(k)$

$$h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m,$$

em que m é o tamanho da tabela, c_1 e $c_2 > 0$, e $i = 0, 1, 2, \dots, m-1$.

- Se duas chaves têm uma posição inicial idêntica na tabela, a sua sequência de exploração é a mesma (tal como na exploração linear a posição inicial define a sequência de exploração)

Sofre de um problema menor de *clustering* (*secondary clustering*)

Algoritma (e Estruturas de Dados)
AUTORITMOS PESQUISA

49

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Exploração Quadrática

$$h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m$$

Sofre de um problema menor de *clustering* (*secondary clustering*):

EXEMPLO ($\alpha = 0.7$)

com exploração linear



com exploração quadrática



Algoritma (e Estruturas de Dados)
AUTORITMOS PESQUISA

50

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Dispersão Dupla
O intervalo i entre amostras é uniforme e calculado através de uma outra função de dispersão
Função de dispersão $h(k)$ na exploração por dispersão dupla:
dadas as funções de dispersão $h_1(k)$, $h_2(k)$

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$$
, em que m é o tamanho da tabela e $i = 0, 1, 2, \dots, m-1$.

Primeiro explora-se o slot $T[h_1(k)]$
Depois $T[h_1(k)+h_2(k)]$, $T[h_1(k)+2h_2(k)]$, $T[h_1(k)+3h_2(k)]$ e assim sucessivamente

- O método funciona melhor do que as explorações linear e quadrática
- As permutações possíveis das sequências aproximam-se de permutações aleatórias
- A primeira posição e/ou o offset (distância em relação à posição anterior) podem variar, o que evita a formação de clusters

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

51 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Dispersão Dupla

$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$

EXEMPLO $m = 13$; $h_1(k) = k \bmod 13$; $h_2(k) = 1 + (k \bmod 11)$; $h(k,i) = (h_1(k) + i h_2(k)) \bmod 13$

$h(72,0) = (7 + 0 \times 11) \bmod 13 = 7$
 $h(98,0) = (7 + 0 \times 11) \bmod 13 = 7$
 $h(69,0) = (7 + 0 \times 11) \bmod 13 = 7$

$h(98,1) = (7 + 1 \times 11) \bmod 13 = 5$
 $h(69,1) = (7 + 1 \times 11) \bmod 13 = 5$

$h(14,0) = (1 + 0 \times 11) \bmod 13 = 1$
 $h(14,1) = (1 + 1 \times 11) \bmod 13 = 5$
 $h(14,2) = (1 + 2 \times 11) \bmod 13 = 9$

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

52 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Dispersão Dupla

O intervalo i entre amostras é uniforme e calculado através de uma outra função de dispersão

Função de dispersão $h(k)$ na exploração por dispersão dupla:
dadas as funções de dispersão $h_1(k)$, $h_2(k)$
 $h(k,i) = (h_1(k) + i h_2(k)) \bmod m$, em que m é o tamanho da tabela e $i = 0, 1, 2, \dots, m-1$.

- A escolha de $h_2(k)$ é importante e nunca deve dar o valor zero
exemplo: $h_2(k) = k \bmod 6$, para $k = 81$
- A escolha de m é importante
Se não for primo corremos o risco de ficar sem localizações muito rapidamente
Se m e $h_2(k)$ forem primos entre si, conseguimos explorar toda a tabela de dispersão
chamamos números primos entre si ao conjunto de números
em que o único divisor comum a todos eles é o número 1

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

53 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Neste caso os elementos estão todos na tabela de dispersão T e a pesquisa é feita sistematicamente nos *slots* da tabela. Considera-se que se pode estimar à partida o número de elementos na tabela e que este é menor (desejavelmente folgado) em relação à dimensão de T . Para cada elemento que colide é procurada uma nova posição na tabela.

Exploração (probing):

④ Dispersão Dupla

O intervalo i entre amostras é uniforme e calculado através de uma outra função de dispersão

Função de dispersão $h(k)$ na exploração por dispersão dupla:
dadas as funções de dispersão $h_1(k)$, $h_2(k)$
 $h(k,i) = (h_1(k) + i h_2(k)) \bmod m$, em que m é o tamanho da tabela e $i = 0, 1, 2, \dots, m-1$.

- A escolha de $h_2(k)$ é importante e nunca deve dar o valor zero
- A escolha de m é importante

Se m e $h_2(k)$ forem primos entre si, conseguimos explorar toda a tabela de dispersão:

Escolher um m que seja uma potência de 2 e uma função h_2 cujo resultado seja sempre ímpar, ou Escolher um m primo e uma função h_2 cujo resultado seja sempre um inteiro positivo menor do que m por exemplo - m primo e $h_2(k) = k \bmod m$ e $h_2(k) = 1 + (k \bmod m')$ em que m' é um valor ligeiramente menor do que m

Algoritmia (e Estruturas de Dados)
ALGORITMOS PESQUISA

54 utad Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Repositionamento (Open Addressing)

Análise:

Factor de carga de uma tabela de dispersão T :
 $\alpha = n/m$,
em que n é o número de chaves/elementos na tabela e m é o número de slots

Assumindo uma distribuição uniforme das chaves (*uniform hashing*) not really true but double hashing can come close

n	m	alfa	1/(1-alfa)	1/alfa * ln 1/(1-alfa)
100	100000	0,001	1,0010	0,99950
1000	100000	0,010	1,0100	0,99503
2000	100000	0,020	1,0200	0,99013
5000	100000	0,050	1,0530	1,03286
10000	100000	0,100	1,1110	1,05261
50000	100000	0,500	2,0000	1,38629
99000	100000	0,990	100,0000	4,65169

Theorem. Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an **unsuccessful** search is at most $1/(1-\alpha)$, assuming uniform hashing.

Theorem. Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a **successful** search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Cormen T.H., Leiserson C.E., Rivest R.L., and Stein C., 2009
Introduction to Algorithms, 3rd edition.

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

55

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

④ Reposicionamento (*Open Addressing*)

Análise:

Factor de carga de uma tabela de dispersão T:

$$\alpha = n/m,$$

em que n é o número de chaves/elementos na tabela e m é o número de slots

Assumindo uma distribuição uniforme das chaves (*uniform hashing*)

Caso-médio:

Depende do número de colisões.

Complexidade algorítmica da pesquisa

$$O\left(\frac{1}{1-\alpha}\right)$$

ver teorema anterior para o caso de uma pesquisa infrutífera

n	m	alfa	1/(1-alfa)
100	100000	0,001	1,0010
1000	100000	0,010	1,0100
2000	100000	0,020	1,0200
5000	100000	0,050	1,0530
10000	100000	0,100	1,1110
50000	100000	0,500	2,0000
99000	100000	0,990	100,0000

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

56

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

Análise: Encadeamento vs Reposicionamento

Vantagens do Encadeamento:

- Resolução das colisões de forma simples e eficaz
- A tabela de dispersão pode conter mais elementos sem grande deterioração do desempenho (o factor de carga pode ser ≥ 1)
- O desempenho do *Encadeamento* piora muito mais lentamente do que o do *Reposicionamento*
- A eliminação de elementos é fácil
- O encadeamento é menos sensível às funções de dispersão e factores de carga/tamanho da tabela

Desvantagens do Encadeamento:

- Necessita de mais uma estrutura de dados para as listas de colisões
- O seu custo principal é a memória extra necessária para a implementação destas listas encadeadas
- Uso da *cache* pouco eficaz (as chaves ficam em listas encadeadas)
- Nalgumas linguagens, a manipulação de apontadores (subjacente ao uso de listas encadeadas) pode ser mais demorado e tornar o processo lento

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

57

utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

Análise: Encadeamento vs Reposicionamento

Vantagens do Repositionamento:

- Todos os elementos estão na tabela (não são necessárias estruturas de dados auxiliares)
- Mais eficiente quanto ao uso de *cache* (todos os elementos estão na tabela)
- Mais eficiente quanto ao uso de memória

Desvantagens do Repositionamento:

- Necessita de atenção na escolha adequada da dimensão da tabela de dispersão
- Necessita de atenção ao factor de carga e à possibilidade de *clustering*
- A implementação da saída de elementos é difícil e pode necessitar de info extra (ocupado, vazio, apagado) em cada *slot*

Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

58

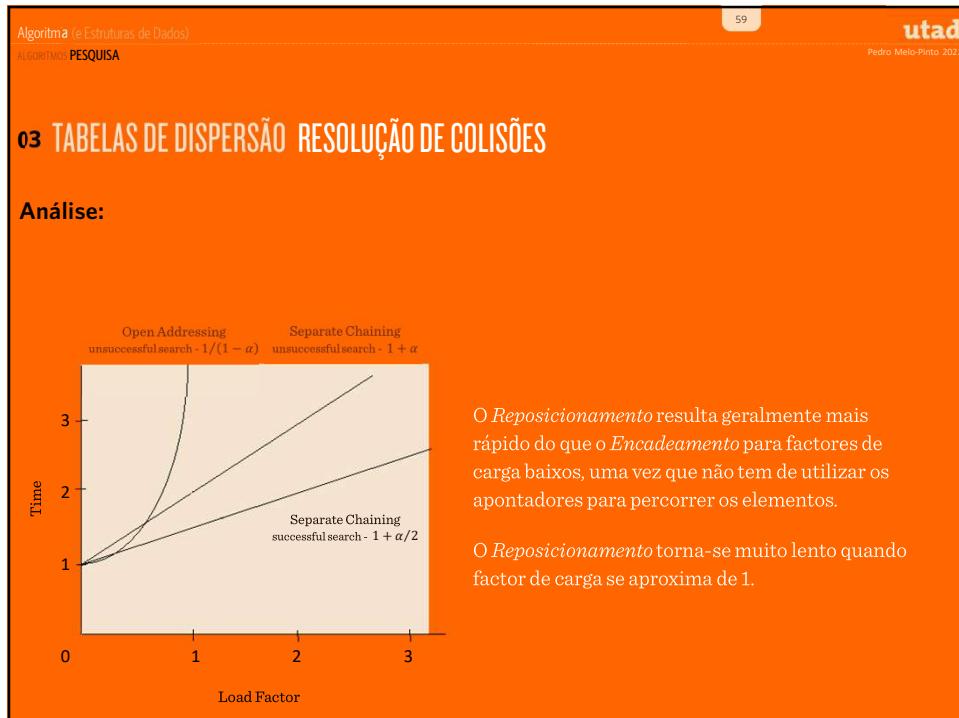
utad
Pedro Melo-Pinto 2022

03 TABELAS DE DISPERSÃO RESOLUÇÃO DE COLISÕES

Análise:

The figure consists of three side-by-side line graphs. All graphs have 'Load Factor' on the x-axis and 'Average Probe Length' on the y-axis.

- Linear Probing:** The x-axis ranges from 0 to 1.0, and the y-axis ranges from 0 to 16. Two curves are shown: an upper curve labeled 'Unsuccessful' and a lower curve labeled 'Successful'. Both curves start at (0,1) and increase rapidly as the load factor increases, with the unsuccessful probe length reaching approximately 15 at a load factor of about 0.8.
- Quadratic Probing:** The x-axis ranges from 0 to 1.0, and the y-axis ranges from 0 to 16. Two curves are shown: an upper curve labeled 'Unsuccessful' and a lower curve labeled 'Successful'. Both curves start at (0,1) and increase rapidly as the load factor increases, with the unsuccessful probe length reaching approximately 15 at a load factor of about 0.8.
- Separate Chaining:** The x-axis ranges from 0 to 5.0, and the y-axis ranges from 0 to 6. A single curve is shown, starting at (0,1) and increasing linearly with a slight upward bend as it approaches a load factor of 4.0, reaching a probe length of 6 at a load factor of approximately 4.5.



Algoritma (e Estruturas de Dados)
ALGORITMOS PESQUISA

60

utad
Pedro Melo-Pinto 2022

PESQUISA ANÁLISE COMPARATIVA

algorithm (data structure)	worst-case cost (after N inserts)	average-case cost (after N random inserts)	key interface	memory (bytes)		
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	$N/2$	N	<code>equals()</code>	$48N$
binary search (ordered array)	$\lg N$	N	$\lg N$	$N/2$	<code>compareTo()</code>	$16N$
binary tree search (BST)	N	N	$1.39 \lg N$	$1.39 \lg N$	<code>compareTo()</code>	$64N$
2-3 tree search (red-black BST)	$2 \lg N$	$2 \lg N$	$1.00 \lg N$	$1.00 \lg N$	<code>compareTo()</code>	$64N$
separate chaining ^f (array of lists)	$< \lg N$	$< \lg N$	$N/(2M)$	N/M	<code>equals()</code> <code>hashCode()</code>	$48N + 64M$
linear probing ^f (parallel arrays)	$c \lg N$	$c \lg N$	< 1.50	< 2.50	<code>equals()</code> <code>hashCode()</code>	between $32N$ and $128N$

^f with uniform and independent hash function

Asymptotic cost summary for symbol-table implementations

Sedgewick, R. and Wayne, K., 2013.
Algorithms, 4th edition.

61

utad
Pedro Melo-Pinto 2022

Leitura Adicional:

- ① Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., 2009.
Introduction to Algorithms 3rd Edition. **CAPÍTULOS 11-13**
- ② Sedgewick R. and Wayne, K., 2011
Algorithms 4th Edition. **CAPÍTULO 3**
- ③ Adrego da Rocha A., 2014
Estruturas de Dados e Algoritmos em C 3^a Edição. **CAPÍTULOS 4, 5.1-5.3**
- ④ Adrego da Rocha A., 2014
Análise da Complexidade de Algoritmos. **SUBCAPÍTULOS 2.1-2.2**