Tabela de dispersão

Prof. Marcel Hugo
Estruturas de Dados

Departamento de Sistemas e Computação
Universidade Regional de Blumenau – FURB

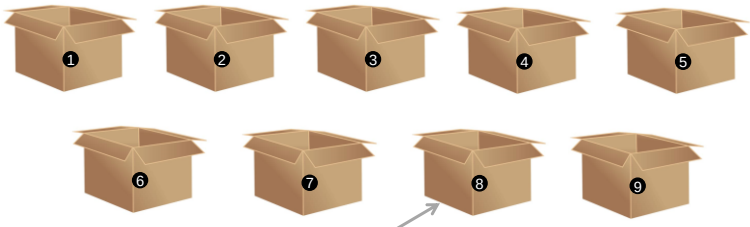
1

Definições

- Tabelas de dispersão oferecem inserção, busca e remoção muito rápidas:
  - Podem executar em *tempo constante*
- Também conhecidas como:
  - Mapas de dispersão
  - Tabelas de espalhamento
  - Tabela *hash* ou
  - Mapa *hash*

2

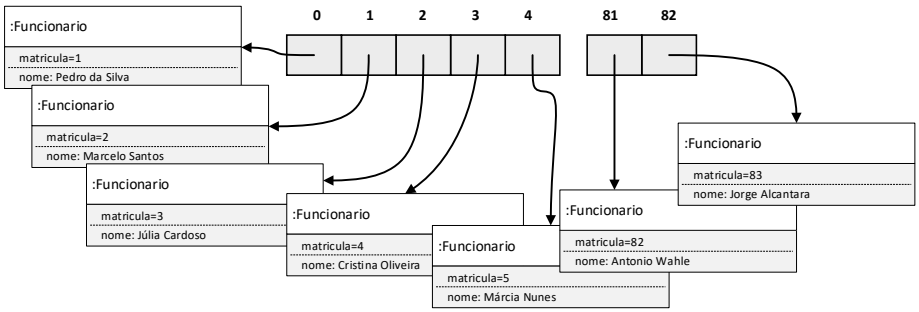
Motivação



Como encontrar rapidamente uma ficha, sem ter que acessar todas as caixas?

3

Busca em vetores



- Em que posição deve estar armazenado o funcionário de código *n*?
- Qual o custo para acessar o objeto que representa o funcionário de código *n*?

4

Usando vetores

- Para obter o registro de qualquer funcionário, basta ler o elemento cujo endereço é *n-1*
- Para incluir um novo funcionário, também é rápido, pois basta inserí-lo depois do último elemento ocupado:

```
Funcionario f = new Funcionario();
funcionarios[totalEmpregados++] = f;
```

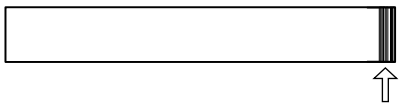
- Uso de vetores é atraente, porém:
  - Nem sempre as chaves de acesso são "bem organizadas"
  - Se houver exclusões de funcionários existirão intervalos perdidos

5

Uma tabela de dispersão simples

Sequencia	Matrícula	Nome
1	176489	Adriana Freitas
2	176203	Beatriz Souza
3	176770	Bruno Silva
4	181539	Douglas de Oliveira
5	181656	Eduardo Junior
6	181947	Fábio Costa
7	181766	Gabriel Xavier
8	185475	Gabriela Pereira
9	179264	Jonatan Becker
10	175416	Lucas Bocker ← menor valor
11	181616	Matheus Goias
12	182455	Miguel Radunz
13	176081	Paulo Wegner
14	185993	Renato Weber ← maior valor
15	176457	Ricardo Rocha
16	176278	Rodrigo Duarte

Com um vetor de [0..185992] teríamos:



dados concentrados aqui

Taxa de ocupação: $\frac{16}{185.993}$ = 0,008 %

6

Uma tabela de dispersão simples

Sequencia	Matrícula	Nome	Índice
1	176489	Adriana Freitas	1073
2	176203	Beatriz Souza	787
3	176770	Bruno Silva	1354
4	181539	Douglas de Oliveira	6123
5	181656	Eduardo Junior	6240
6	181947	Fábio Costa	6531
7	181766	Gabriel Xavier	6350
8	185475	Gabriela Pereira	10059
9	179264	Jonatan Becker	3848
10	175416	Lucas Bocker	**0**
11	181616	Matheus Goias	6200
12	182455	Miguel Radunz	7039
13	176081	Paulo Wegner	665
14	185993	Renato Weber	**10577**
15	176457	Ricardo Rocha	1041
16	176278	Rodrigo Duarte	862

⇧

Matricula - 175416

Tamanho do vetor: 10578 elementos (185994 – 175416)

Taxa de ocupação: 0,15%

7

Função de dispersão

- Ainda precisamos “compactar” a faixa de valores para utilizar uma faixa menor.
- *Função de dispersão* ou *transformação* ou *função de hash* é uma função matemática que converte um número compreendido numa faixa de valores grande para uma faixa de valores menor.

8

Função de Dispersão

- Uma função de transformação ou dispersão deve mapear chaves em inteiros dentro do intervalo [0...M − 1], onde M é o tamanho da tabela.

- Método mais usado: resto da divisão por M .

  h(K) = K % M (em linguagem Java)

  onde K é um inteiro correspondente à chave.

9

Compactação de valores

- Considerar que ao invés de um vetor de 10 mil elementos, tenhamos 32 elementos.
- Podemos "compactar" com a função:

  Matricula % 32

Mátricula	Índice	Aluno
176489	9	Adr...
176203	11	Bea...
176770	2	Bru...
181539	3	Dou...
181656	24	Edu...
181947	27	Fáb...
181766	6	Gab...
185475	3	Gab...
179264	0	Jon...
176081	17	Pau...
181616	16	Mat...
182455	23	Mig...
175416	24	Luc...
185993	9	Ren...
176457	9	Ric...
176278	22	Rod...

10

Tabela de dispersão

- Um vetor no qual os dados são inseridos usando uma *função de hash* é chamado de *Tabela de dispersão*.
- Cada objeto armazenado na tabela de dispersão deve possuir uma chave de busca que é única para cada objeto.
- Chave de busca ou chave de pesquisa é utilizada nas operações de inclusão, exclusão e pesquisa.

11

Tabela de dispersão

- Para pesquisar um elemento na tabela de dispersão:
  - Calcular o valor de hash da chave (índice)
  - Se a posição do índice estiver vazia, o elemento não foi adicionado
- Para inserir um elemento na tabela de dispersão
  - Calcular o valor de hash da chave (índice)
  - Inserir o elemento na posição do índice calculado
- Para excluir um elemento da tabela de dispersão
  - Calcular o valor de hash da chave (índice)
  - Excluir o elemento na posição do índice calculado

12

Tabela de dispersão

- Como a função de *hash* deriva o índice do vetor a partir de uma chave de busca, é esperado que a chave de busca seja imutável.

13

Funções de Transformação/Dispersão

- A função ideal é aquela que:
  - Seja simples de ser computada, por isso, rápida;
  - Para cada chave de entrada,

  qualquer uma das saídas possíveis é igualmente provável de ocorrer, ou seja, espalha adequadamente na tabela, evitando colisão.

- Colisão = duas ou mais chaves de busca são mapeadas para um mesmo índice da tabela *hash*.

14

Colisões

- Ao comprimir uma faixa de chaves com grande intervalo numa faixa de pequeno intervalo, não há garantias de que não haja duas chaves com o mesmo índice do vetor.
- Exemplo: tentar incluir um aluno cujo código de matrícula seja 18.000
  - Neste caso, 18000 % 32 = 16
  - A posição 16 já está ocupada
- Paradoxo do aniversário (Feller,1968, p. 33): em um grupo de 23 ou mais pessoas, juntas ao acaso, existe uma chance maior do que 50% de que 2 pessoas comemorem aniversário no mesmo dia.

15

Fator de carga

- É a proporção do número de dados armazenados no mapa sobre o tamanho total do vetor.

$$f = \frac{N}{M}$$

  - N = quantidade de dados armazenados
  - M = tamanho do vetor
- Para minimizar colisões mas não desperdiçar muito espaço, utiliza-se um fator de carga igual a 0,75:
  - Taxa de 50% traz melhores resultados
  - Taxa menor que 25% traz um gasto excessivo de memória

16

Colisões

- Ao calcular o valor de *hash* para um elemento e identificar que a posição do vetor já está ocupada, afirmamos que ocorreu uma colisão.
- Para reduzir as colisões deve-se criar um vetor cuja quantidade de elementos seja um número primo.
- Existem algumas formas de resolver a colisão
  - Endereçamento aberto
  - Endereçamento separado

17

Tratamento de colisões por endereçamento aberto

- No endereçamento aberto, quando um item de dados não pode ser colocado no índice calculado pela função de *hash*, outro local no vetor é procurado.
- Existem três métodos de endereçamento aberto:
  - Exploração linear
  - Exploração quadrática
  - Hash duplo

18

Exploração Linear

- Na inclusão de um novo elemento, ao identificar uma colisão buscar a próxima posição vazia no vetor
- Exemplo:
  - Inserir aluno com matrícula 190.019
  - Onde inserir: 190.019 % 32 = 3
  - Procurar próxima posição livre: 4

Índice	Matrícula	Nome
0	179264	Jo..
1		
2	176770	Br..
3	185475	Ga..
4		
5		
6	181766	Ga..
7		
8		
9	185993	Ri..
10		
11		
12		
13		
14		
15		
16		

19

Exploração linear

- O algoritmo de busca precisa ser adaptado para que, quando não encontrar a chave na posição calculada pela função de *hash*, seja explorada as posições seguintes, até encontrar o elemento ou então a posição estar vazia. Operações de exclusão também requerem ser tratadas para lidar com exclusão lógica.
- Esta técnica faz com que regiões preenchidas de forma consecutiva (denominadas de *clustering*) surjam com mais frequência, tornando a tabela menos dispersa.

20

Exploração Quadrática

- Já que a exploração linear favorece o surgimento de *clustering*, pode-se utilizar a exploração quadrática.
- Na exploração linear, o índice calculado pela função de hash é *x* e quando há colisão, procura-se nas posições adjacentes, isto é, x+1, x+2, x+3, etc.
- Na exploração quadrática, quando há colisão na posição *x*, procura-se nas posições $x+1^2$ + $x+2^2$ + $x+3^2$ + $x+4^2$, etc.

21

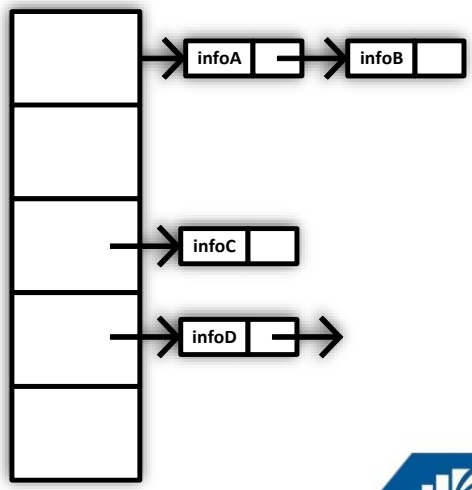
Hash duplo

- A exploração quadrática não evita o surgimento de *clustering*. Somente transfere para posições adiante.
- Isso ocorre porque as sequências de exploração são constantes (1, 4, 9, 16, etc)
- No *hash* duplo, mais uma função de *hash* é utilizada:
  - A segunda função de *hash* é utilizada para determinar o valor a ser acrescentado (isto é, o passo) para as sequências de exploração.
  - Deve resultar num número positivo
  - Exemplo: passo <- (chave % 7) + 1

22

Tratamento de colisões por Endereçamento separado

- Neste abordagem, o vetor não armazena os dados, ao invés disso, mantém listas encadeadas, para armazenar os dados.



23

Endereçamento separado

- A chave de um item de dados é convertida para o índice. O elemento é adicionado na lista encadeada daquele índice.
  - Não há necessidade de buscar células vazias no vetor primário.
- A pesquisa requer efetuar uma busca linear na lista encadeada do índice calculado pela função de dispersão
- Esquema mais simples de implementar do que os algoritmos de endereçamento aberto.

24

Desvantagens das tabelas de dispersão

- Algumas desvantagens:
  - Baseiam-se em vetores
    - Difíceis de expandir
  - Podem ter desempenho reduzido de forma significativa
    - Requer que o programador saiba antecipadamente a quantidade aproximada de dados.
  - Não há uma maneira conveniente de visitar os itens de uma tabela de dispersão de forma ordenada

25

Chaves de busca do tipo texto

- Pode-se atribuir um código para cada caractere. Exemplo:

Caractere	Valor
A	1
B	2
C	3
D	4
...	
S	19
...	
Z	26

Somar o código de caractere da palavra.

Exemplo: para a palavra “CASA” poderia ser atribuído:

C = 3
A = 1
S = 19
A = 1

3 + 1 + 19 + 1 = 24

Neste exemplo, está-se considerando que o conjunto de caracteres é composto por A..Z.

26

Chaves de busca do tipo texto

- Várias palavras podem produzir o mesmo código:

Palavra	Cálculo
ABOLIR	1 + 2 + 15 + 12 + 9 + 18
ACEITAR	1 + 3 + 5 + 9 + 20 + 1 + 18
CLARIDADE	3 + 12 + 1 + 18 + 9 + 4 + 1 + 4 + 5
DIGNIDADE	4 + 9 + 7 + 14 + 9 + 4 + 1 + 4 + 5
DUELO	4 + 21 + 5 + 12 + 15
FAZENDA	6 + 1 + 26 + 5 + 14 + 4 + 1
GELEIRA	7 + 5 + 12 + 5 + 9 + 18 + 1
HIGIENE	8 + 9 + 7 + 9 + 5 + 14 + 5
LARANJA	12 + 1 + 18 + 1 + 14 + 10 + 1
MARTE	13 + 1 + 18 + 20 + 5
OBEDECER	15 + 2 + 5 + 4 + 5 + 3 + 5 + 18
POETA	16 + 15 + 5 + 20 + 1

Alguns exemplos de palavras que produzem "57" como resultado

27

Chaves de busca do tipo texto

- Comparando com números...

Número	Representação
785	7 + 8 + 5
992	9 + 9 + 2
5555	5 + 5 + 5 + 5
82514	8 + 2 + 5 + 1 + 4
1225055	1 + 2 + 2 + 5 + 0 + 5 + 5

Diversos outros números produzem a mesma combinação

$7\ 8\ 5 = (7 * 10^2) + (8 * 10^1) + (5*10^0)$

28

Chaves de busca do tipo texto

C	A	S	A

Letra	Cálculo
C	$3 * 26^0$
A	$1 * 26^1$
S	$19 * 26^2$
A	$1 * 26^3$

Este esquema gera uma valor único para cada combinação de caracteres

Algoritmo: **calcularHash(String texto)**

```
int n ← size(texto) - 1;
h ← 0;

para c ← 0 até n faça
  caractere ← texto[c];
  codigo ← mapear(caractere);
  h ← h + (código * 26 ^ c);
fim-para;

retornar h
```

29

Chaves de busca do tipo texto

- Um sistema para obter um valor numérico a partir de um `caractere` baseia-se num *mapa de caracteres*.
- Um mapa de caracteres é um conjunto de caracteres que podem ser utilizados. Existem vários mapas:
  - US-ASCII (ISO/IEC 646): contém caracteres do alfabeto inglês
  - Latin-x (ISO/IEC 8859-x): contém caracteres ocidentais (inclusive latinos)
  - Unicode (ISO/10646): contém os caracteres de de todas as linguagens (atualmente capaz de representar aproximadamente 2 milhões de símbolos distintos)
- Num mapa de caracteres, cada caractere possui um código (code point)

30

Mapa de caracteres usado em Java

- Um fragmento do mapa de caracteres Unicode (hexa):

0 0030	@ 0040	P 0050	` 0060	p 0070
1 0031	A 0041	Q 0051	a 0061	q 0071
2 0032	B 0042	R 0052	b 0062	r 0072
3 0033	C 0043	S 0053	c 0063	s 0073
4 0034	D 0044	T 0054	d 0064	t 0074

Fonte: http://unicode.org/charts/PDF/U0000.pdf

31

Função de hash para string em Java

- Em Java, o *hashCode()* para string é calculado desta forma:

$s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + ... + s[n-1]*31^0$

- Exemplo:

  "CASA"

  $'C'*31^3 + 'A'*31^2 + 'S'*31^1 + 'A'*31^0$

  $67*31^3 + 65*31^2 + 83*31^1 + 65*31^0$

  2061100

32

Função de hash para String em Java

- Utiliza-se o número 31 por questão de otimização pois a multiplicação por este número pode ser resolvida com uma operação de deslocamento de bits e subtração:
  - 31 * i = (i << 5) – i

Exemplo: 31 * 31 = 961

Ou (31 << 5) – 31

= 992 – 31 = 961.

33