ESTRUTURA DE DADOS

Tabelas de Dispersão

Tabela Hash

Professor Mestre Igor de Moraes Sampaio
igor.sampaio@ifsp.edu.br

Motivação

- Sabemos que:
  - Busca sequencial executa em tempo O(n).
  - Busca binária executa em tempo O(log(n)). Busca binária exige vetor ordenado.
- Seria possível efetuar uma busca em tempo melhor do que O(log(n))?
- Quais restrições devem existir sobre os dados?

Motivação

- Tabelas de Hash (ou Tabelas Hash) permitem buscas em tempo constante, satisfeitas algumas restrições.

- Essa estrutura pode ter vários nomes como: dicionários, mapas, arrays associativos, e assim por diante.

- A princípio, a chave de busca pode ser de qualquer tipo.

Métodos

- retrieveItem(k): retorna uma entrada com chave igual a k, se ela existir. Caso contrário, retorna nulo.

- insertItem(k, v): insere uma entrada v na chave k se a chave não existir. Caso contrário, atualiza o valor associado a k.

- deleteItem(k): remove a chave k e o valor associado a ela.

Métodos

- size(): retorna o número de entradas.
- keySet(): retorna uma lista encadeada de todas as chaves armazenadas na tabela.
- values(): retorna uma coleção contendo
- todos os valores associados com as chaves armazenadas na tabela.
- entrySet(): retorna uma coleção contendo todas as entradas (chave-valor) da tabela.

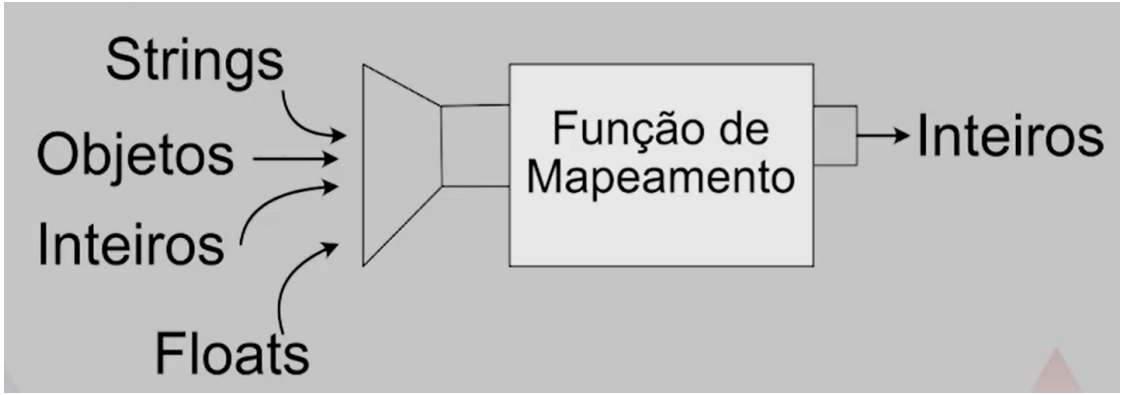
Implementação

- A própria chave deve ser usada para organizar os dados em memória.
- Cada entrada da estrutura é composta por um par "chave-valor" (k, v). A associação entre k e v define o mapeamento.
- A chave é um identificador único e deve ser vista como um "endereço" para seu valor.

Implementação

- A tabela pode ser organizada em memória como um vetor, dado que este permite acesso em tempo constante.

- As chaves podem ser de qualquer tipo de dados, mas para efetuarmos a busca no vetor, precisaremos de uma função que mapeie chaves em números inteiros.

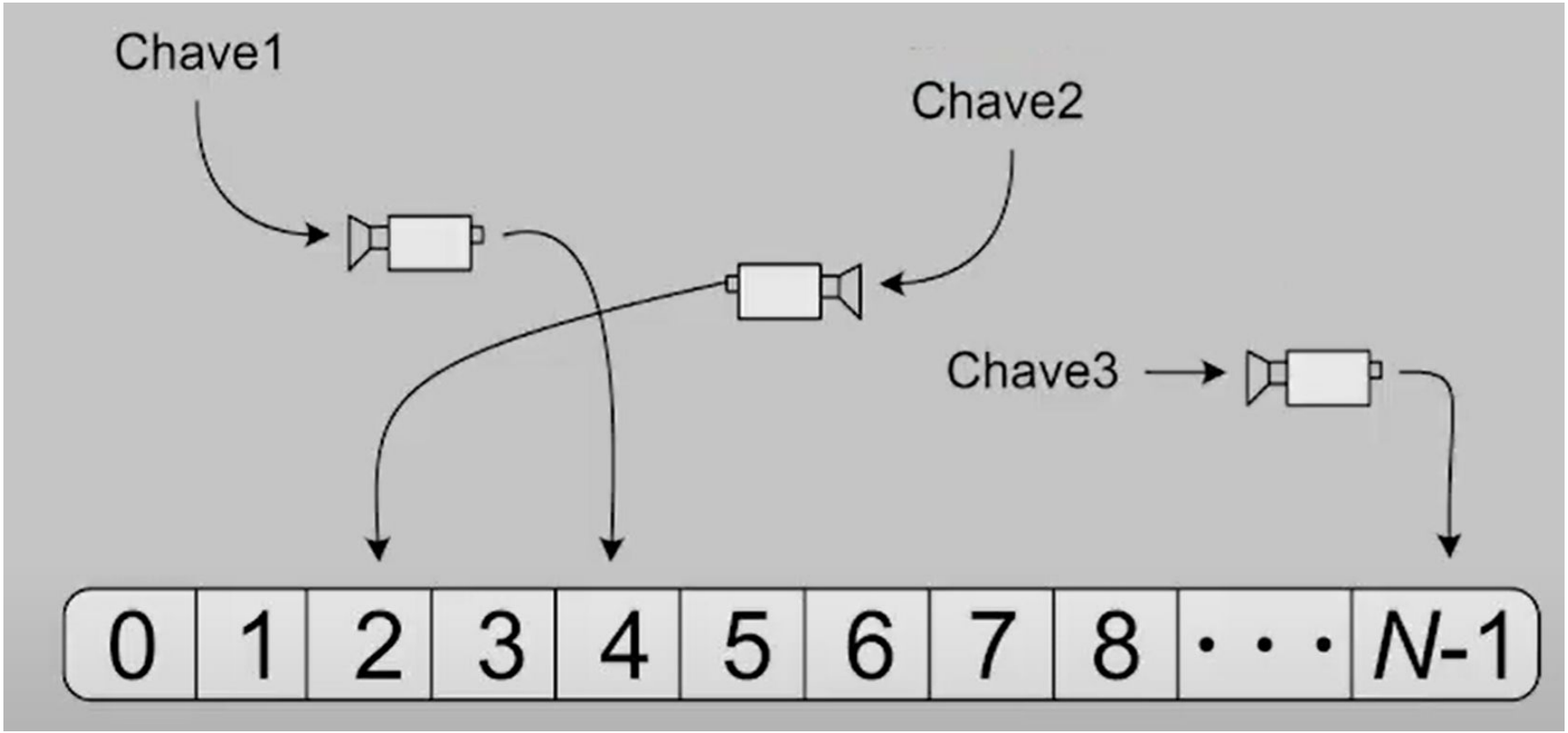


Implementação

- Seja h a função que faz o mapeamento (também chamada de função de espalhamento) e k a chave, o endereço de memória será dado por h(k).

- Se os valores retornados por h(k) forem bem distribuídos em um intervalo entre 0 e N-1, então precisamos de um vetor de capacidade N.

- Assumindo ausência de colisões, essa estrutura básica seria suficiente.


Chave1
Chave2
Chave3
0
1
2
3
4
5
6
7
8
• • •
N-1

Funções de Hash

- A função de hash h mapeia cada chave em um intervalo de 0 a N-1, onde N é a capacidade do arranjo.

- É possível tratar colisões, mas a melhor estratégia por enquanto é tentar evitá-las.

- Uma função de hash é boa se minimiza a ocorrência de colisões.

Funções de Hash

- A primeira tarefa da função será transformar chaves de tipos arbitrários em inteiros.

- Vamos assumir que queremos armazenar informações de funcionários de uma empresa e indexar essas informações pelo login único da pessoa.

- O login pode ser o primeiro nome da pessoa, mas se este já foi escolhido por alguém, então outro deve ser selecionado pelo funcionário.

Funções de Hash

- Uma função de hash pode primeiramente mapear os caracteres para inteiros.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

ulisses: u + l + i + e + 3*s =

117 + 108+ 105 + 101 + 3*115 =

776

Funções de Hash

- Podemos mapear qualquer login em inteiro

ulisses	776
danielle	830
amanda	610
cleópatra	1218

- O valor inteiro encontrado pode ser o índice da entrada em um vetor.

- Essa ideia ilustra uma implementação básica da tabela hash.

Funções de Hash

- Essa estratégia gera colisões:

orlando	751
adnalro	751
adriana	720
ariadna	720

- Uma função de hash melhor levaria em conta a posição dos caracteres Ci, na cadeia C= (C0,C1,C2,..., CK-1).

$$c_0 a^{k-1} + c_1 a^{k-2} + c_2 a^{k-3} + \ldots + c_{k-2} a^1 + c_{k-1}$$

- Para algum a diferente de 0 ou 1.

Funções de Hash

- Por exemplo, com a = 3, teríamos os seguintes valores para orlando e odnalro:

o	111 * 729 +	o	111 * 729 +
r	114 * 243 +	d	100 * 243 +
l	108 * 81 +	n	110 * 81 +
a	97 * 27 +	a	97 * 27 +
n	110 * 9 +	l	108 * 9 +
d	100 * 3 +	r	114 * 3 +
o	111 * 1 = 121389	o	111 * 1 = 118173

- Um valor de a alto (33, 37, 39 ou 41) tende a diminuir o número de colisões para algumas poucas.

Funções de Hash

- Um valor de a alto (33, 37, 39 ou 41) pode levar a um overflow do intervalo dos inteiros.

- A compressão dos valores pode fazer parte da função. O resto da divisão por N estabiliza os valores em um intervalo [0 .. N-1].

$$i \bmod N$$

- O tamanho do arranjo N pode aumentar ou diminuir o número de colisões:

    - Se usarmos N = 1000, teremos muito menos colisões do que com N = 100.

Funções de Hash

- Para ajudar o espalhamento das chaves, é interessante usar um número primo para N. Isso diminui a chance de ocorrer padrões na distribuição de dados.

- Por exemplo, se temos as chaves {200, 205, 210, 215, 220,..., 600).
  - Com N = 100, cada chave irá colidir com várias outras chaves.
  - Com N = 101 não teremos colisões.

Funções de Hash

- Nesse caso, é possível fazer com que cada endereço tenha espaço para mais de uma entrada.