Funzioni Hash

Sommario

- Tabelle ad indirizzamento diretto e hash
- Funzioni Hash
  - Requisiti
  - Metodo della divisione
  - Metodo della moltiplicazione
  - Funzione Hash Universale

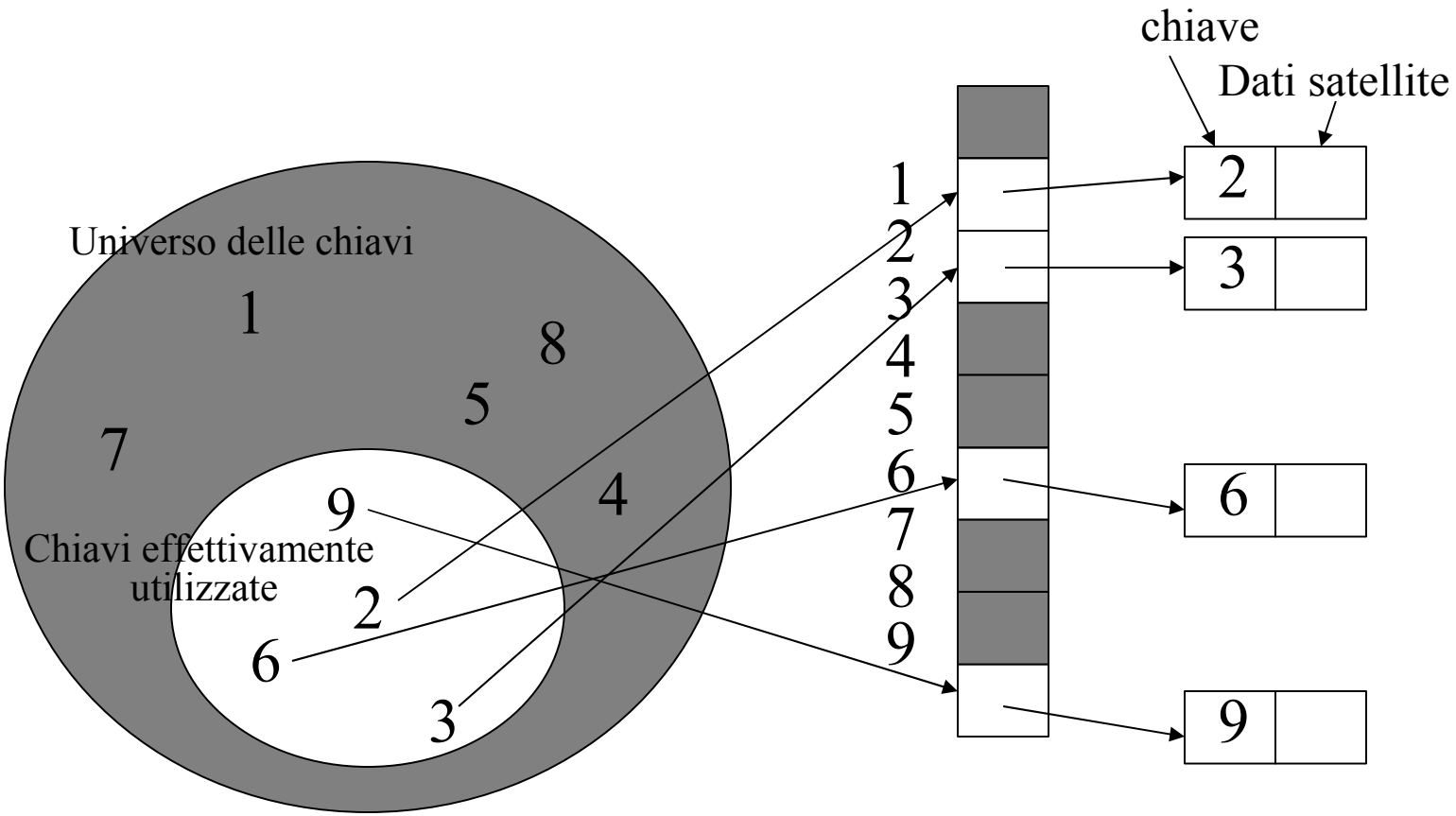
La ricerca

- Talvolta si richiede che un insieme dinamico fornisca solamente le operazioni di inserzione, cancellazione e ricerca
- Non è, ad esempio, necessario dover ordinare l'insieme dei dati o restituire l'elemento massimo, o il successore
- Queste strutture dati prendono il nome di *dizionari*

Tabelle ad indirizzamento diretto

- Se l'universo delle chiavi è piccolo allora è sufficiente utilizzare una *tabella ad indirizzamento diretto*
- Una tabella ad indirizzamento diretto corrisponde al concetto di array:
  - ad ogni chiave possibile corrisponde una posizione, o slot, nella tabella
  - una tabella restituiesce il dato memorizzato nello slot di posizione indicato tramite la chiave in tempo O(1)

Visualizzazione



Pseudocodice delle operazioni

```
Direct-Address-Search(T,k)
1  return T[k]

Direct-Address-Insert(T,x)
1  T[key[x]] ← x

Direct-Address-Delete(T,x)
1   T[key[x]] ← NIL
```

Memorizzazione

- E' possibile memorizzare i dati satellite direttamente nella tabella
- ..oppure memorizzare solo puntatori agli oggetti veri e propri

- Si deve distinguere l'assenza di un oggetto (oggetto NIL) dal caso particolare di un valore dell'oggetto stesso
  - Ad esempio se il dato è un intero positivo è possibile assegnare il codice -1 per indicare NIL

Universo grande delle chiavi

- Se l'universo delle possibili chiavi è molto grande non è possibile o conveniente utilizzare il metodo delle tabelle ad indirizzamento diretto
  - può non essere possibile a causa della limitatezza delle risorse di memoria
  - può non essere conveniente perché se il numero di chiavi effettivamente utilizzato è piccolo si hanno tabelle quasi vuote. Viene allocato spazio inutilizzato
- Le *tabelle hash* sono strutture dati che trattano il problema della ricerca permettono di mediare i requisiti di memoria ed efficienza nelle operazioni

Compromesso

- Nel caso della ricerca si deve attuare un compromesso fra spazio e tempo:
  - spazio: se le risorse di memoria sono sufficienti si può impiegare la chiave come indice (accesso diretto)
  - tempo: se il tempo di elaborazione non rappresenta un problema si potrebbero memorizzare solo le chiavi effettive in una lista ed utilizzare un metodo di ricerca sequenziale
- L'Hashing permette di impiegare una quantità ragionevole sia di memoria che di tempo operando un compromesso tra i casi precedenti

Importanza

- L'hashing è un problema classico dell'informatica
- Gli algoritmi usati sono stati studiati intensivamente da un punto di vista teorico e sperimentale
- Gli algoritmi si hashing sono largamente usati in un vasto insieme di applicazioni
  - Ad esempio nei compilatori dei linguaggi di programmazione si usano hash che hanno come chiavi le stringhe che corrispondono agli identificatori del linguaggio

Tabelle Hash

- Con il metodo di indirizzamento diretto un elemento con chiave k viene memorizzato nella tabella in posizione k
- Con il metodo hash un elemento con chiave k viene memorizzato nella tabella in posizione h(k)
- La funzione h(.) è detta *funzione hash*
- Lo scopo della funzione hash è di definire una corrispondenza tra l'universo U delle chiavi e le posizioni di una tabella hash T[0..m-1]

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

Tabelle Hash

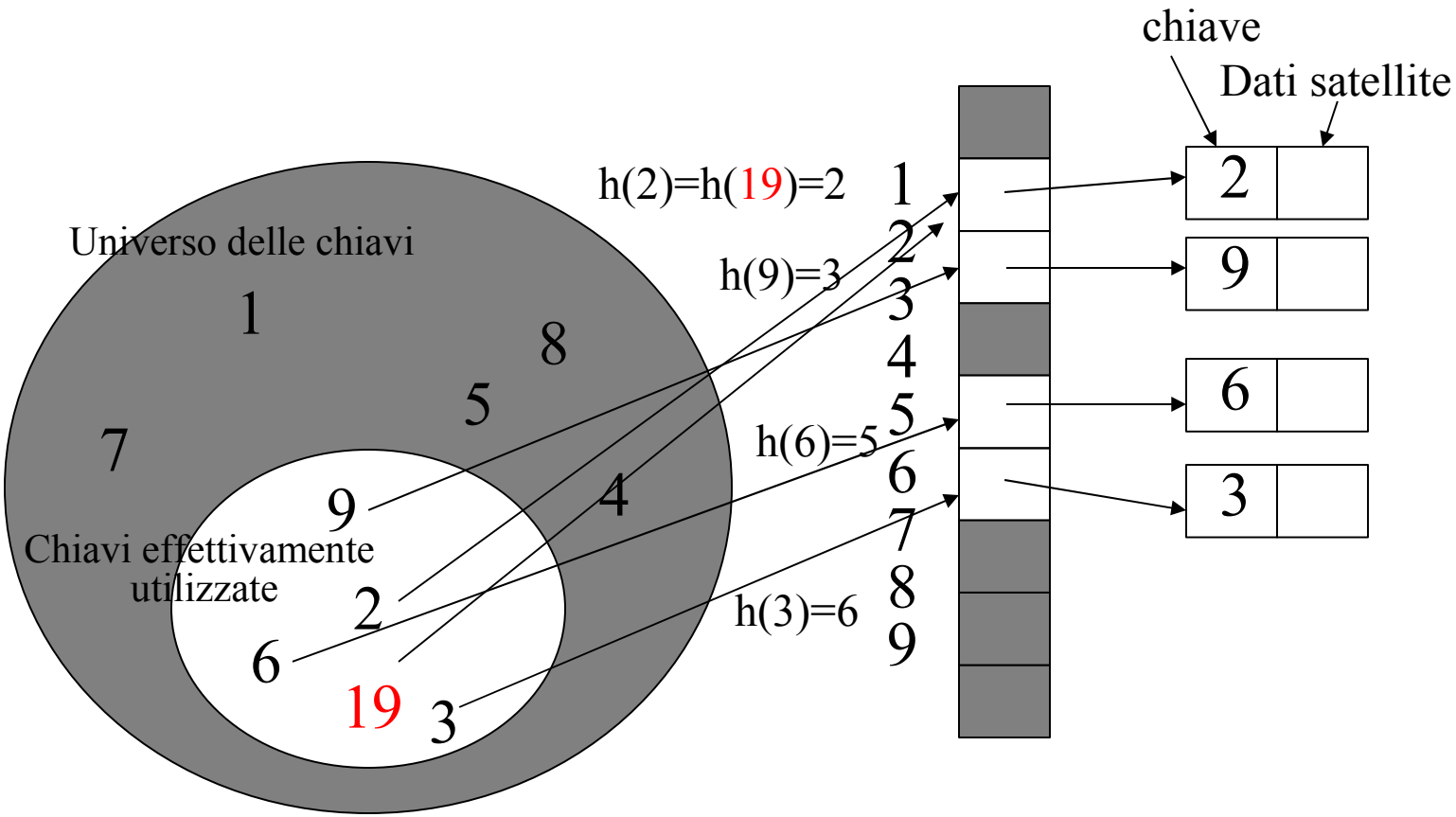
- Un elemento con chiave k ha posizione pari al *valore hash di k* denotato con h(k)
- Tramite l'uso di funzioni hash il range divariabilità degli indici passa da |U| a *m*
- Utilizzando delle dimensioni *m* comparabili con il numero di dati effettivi da gestire si riduce la dimensione della struttura dati garantendo al contempo tempi di esecuzione dell'ordine di O(1)

Tabelle Hash

- Necessariamente la funzione hash non può essere iniettiva, ovvero *due chiavi distinte possono produrre lo stesso valore hash*
- Quando questo accade si dice che si ha una *collisione*

- Vedremo due metodi per risolvere il problema delle collisioni

Visualizzazione



Funzioni Hash

- Quali sono le caratteristiche di una funzione hash?

  Criterio di uniformità semplice:

  il valore hash di una chiave k è uno dei valori 0..m-1 in modo equiprobabile

Funzioni Hash

- Formalmente: se si estrae in modo indipendente una chiave k dall'universo U con probabilità P(k) allora:

$$\sum_{k:h(k)=j} P(k) = 1/m \quad \text{per } j=0,1,\ldots,m-1$$

- Cioè se si partiziona l'universo U in m sottoinsiemi tali per cui nello stesso sottoinsieme consideriamo tutte le chiavi che sono mappate dalla funzione h in j, allora vi è la stessa probabilità di prendere un elemento da uno qualsiasi di questi sottoinsiemi

Funzioni Hash

- Tuttavia non sempre si conosce la distribuzione di probabilità delle chiavi P

- Esempio:
  - Se si ipotizza che le chiavi k siano numeri reali distribuiti in modo indipendente ed uniforme nell'intervallo [0,1], cioè $k \in [0,1]$, allora la funzione

$$h(k) = \lfloor k\, m \rfloor$$

  - soddisfa il criterio di uniformità semplice

Funzioni Hash

- Un altro requisito è che una “buona” funzione hash dovrebbe utilizzare tutte le cifre della chiave per produrre un valore hash
- In questo modo valgono le ipotesi sulla distribuzione dei valori delle chiavi nella loro interezza, altrimenti dovremmo considerare la distribuzione solo della parte di chiave utilizzata

Convertire le chiavi in numeri naturali

- In genere le funzioni hash assumono che l'universo delle chiavi sia un sottoinsieme dei numeri naturali
- Quando questo non è verificato si procede convertendo le chiavi in un numero naturale (anche se grande)
- Un metodo molto usato è quello di stabilire la conversione fra sequenze di simboli interpretati come numeri in sistemi di numerazione in base diversa

Conversione stringhe in naturali

- Per convertire una stringa in un numero naturale si considera la stringa come un numero in base 128
- Esistono cioè 128 simboli diversi per ogni cifra di una stringa
- E' possibile stabilire una conversione fra ogni simbolo e i numeri naturali (codifica ASCII ad esempio)

Tabella ASCII

Byte	Cod.	Char
00000000	0	Null
00000001	1	Start of heading
00000010	2	Start of text
00000011	3	End of text
00000100	4	End of transmit
00000101	5	Enquiry
00000110	6	Acknowledge
00000111	7	Audible bell
00001000	8	Backspace
00001001	9	Horizontal tab
00001010	10	Line feed
00001011	11	Vertical tab
00001100	12	Form Feed
00001101	13	Carriage return
00001110	14	Shift out
00001111	15	Shift in
00010000	16	Data link escape
00010001	17	Device control 1
00010010	18	Device control 2
00010011	19	Device control 3
00010100	20	Device control 4
00010101	21	Neg. acknowledge
00010110	22	Synchronous idle
00010111	23	End trans. block
00011000	24	Cancel
00011001	25	End of medium
00011010	26	Substitution
00011011	27	Escape
00011100	28	File separator
00011101	29	Group separator
00011110	30	Record Separator
00011111	31	Unit separator

Byte	Cod.	Char
00100000	32	Spc
00100001	33	!
00100010	34	"
00100011	35	#
00100100	36	$
00100101	37	%
00100110	38	&
00100111	39	'
00101000	40	(
00101001	41	)
00101010	42	*
00101011	43	+
00101100	44	,
00101101	45	-
00101110	46	.
00101111	47	/
00110000	48	0
00110001	49	1
00110010	50	2
00110011	51	3
00110100	52	4
00110101	53	5
00110110	54	6
00110111	55	7
00111000	56	8
00111001	57	9
00111010	58	:
00111011	59	;
00111100	60	<
00111101	61	=
00111110	62	>
00111111	63	?

Byte	Cod.	Char
01000000	64	@
01000001	65	A
01000010	66	B
01000011	67	C
01000100	68	D
01000101	69	E
01000110	70	F
01000111	71	G
01001000	72	H
01001001	73	I
01001010	74	J
01001011	75	K
01001100	76	L
01001101	77	M
01001110	78	N
01001111	79	O
01010000	80	P
01010001	81	Q
01010010	82	R
01010011	83	S
01010100	84	T
01010101	85	U
01010110	86	V
01010111	87	W
01011000	88	X
01011001	89	Y
01011010	90	Z
01011011	91	[
01011100	92	\
01011101	93	]
01011110	94	^
01011111	95	_

Byte	Cod.	Char
01100000	96	`
01100001	97	a
01100010	98	b
01100011	99	c
01100100	100	d
01100101	101	e
01100110	102	f
01100111	103	g
01101000	104	h
01101001	105	i
01101010	106	j
01101011	107	k
01101100	108	l
01101101	109	m
01101110	110	n
01101111	111	o
01110000	112	p
01110001	113	q
01110010	114	r
01110011	115	s
01110100	116	t
01110101	117	u
01110110	118	v
01110111	119	w
01111000	120	x
01111001	121	y
01111010	122	z
01111011	123	{
01111100	124	|
01111101	125	}
01111110	126	~
01111111	127	Del

Esempio

- La conversione viene fatta considerando il numero espresso in un sistema posizionale in base 128
- Es: per convertire la stringa “casa” si ha:
  - c a s a $_{(128)}$=
  - c_3 a_2 s_1 a_0 $_{(128)}$=
  - ‘c’ *128^3 + ‘a’ *128^2 + ‘s’ *128^1 + ‘a’ *128^0 =
  - 99 *128^3 + 97 *128^2 + 115 *128^1 + 97 *128^0 =
  - 99 *2097152 + 97 *16384 + 115 *128 + 97 *1 =$209222113_{(10)}$

Funzioni Hash

- Come realizzare una funzione hash?
  - Metodo della divisione
  - Metodo della moltiplicazione
  - Metodo della funzione hash universale (non lo vedremo)

Metodo di divisione

- La funzione hash è del tipo:

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

- Cioè il valore hash è il resto della divisione di k per m
- Caratteristiche:
  - il metodo è veloce
  - ma si deve fare attenzione ai valori di m

Metodo di divisione

- m deve essere diverso da 2^p per un qualche p
- Altrimenti fare il modulo in base m corrisponderebbe a considerare solo i p bit meno significativi della chiave
- In questo caso dovremmo garantire che la distribuzione dei p bit meno significativi sia uniforme
- Analoghe considerazioni per m pari a potenze del 10

- *Buoni valori sono numeri primi non troppo vicini a potenze del due*

Esempio

- Attenzione! Se si usasse m=100, allora:

k	h(k)
123	23
2323	23
128723	23

- Se si usasse $m=2^3$, allora:

k	h(k)
10110101	101
111101	101
100011101	101

Metodo di Moltiplicazione

- Il metodo di moltiplicazione per definire le funzioni hash opera in due passi:
  - si moltiplica la chiave per una costante A in [0,1] e si estrae la parte frazionaria del risultato
  - si moltiplica questo valore per m e si prende la parte intera
- Analiticamente si ha:

$$h(k) = \lfloor m(k\,A \bmod 1) \rfloor$$

Metodo di moltiplicazione

- Un vantaggio del metodo è che non ha valori critici di m
- Es:
  - A=(√5 -1)/2=0.61803…
  - sia k=123456 e m=10000
  - allora:

h(k)= ⌊10000(123456*0.61803.. mod 1)⌋

=⌊10000(76300.0041151.. mod 1)⌋

=⌊10000(0.0041151..)⌋

=⌊41.151..⌋

=41

Metodo di moltiplicazione

- Spesso si sceglie $m=2^p$ per un qualche p in modo da semplificare i calcoli
- Una implementazione veloce di una h per moltiplicazione è il seguente:
  - supponiamo che la chiave k sia un numero codificabile entro w bit dove w è la dimensione di una *parola* del calcolatore
  - si consideri l'intero anch'esso di w bit $\lfloor A\, 2^w \rfloor$
  - il prodotto $k * \lfloor A\, 2^w \rfloor$ sarà un numero intero di al più $2 \cdot w$ bit
  - consideriamo tale numero come $r_1 2^w + r_0$
  - r_1 è la parola *più significativa* del risultato e r_0 quella *meno significativa*
  - il valore hash cercato consiste nei p bit più significativi di r_0

Visualizzazione