

Algoritmi e Strutture Dati

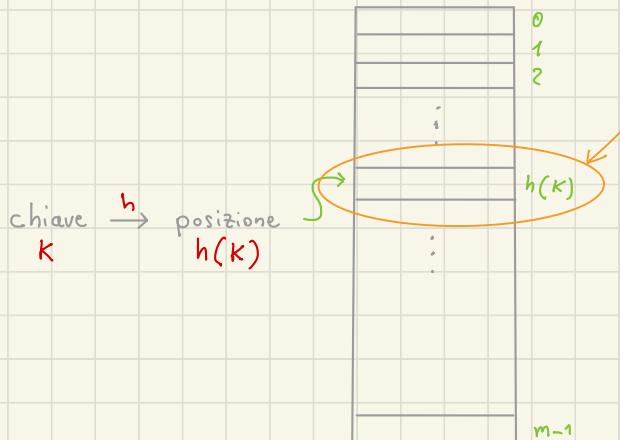
Lezione 31

3 dicembre 2025

GESTIONE DELLE COLLISIONI INTERNA

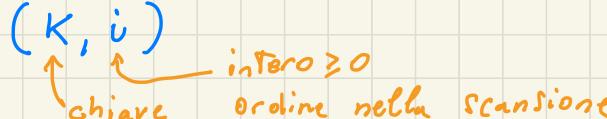
Indirizzamento aperto

inserimento di elemento di chiave K



se $h(K)$ è già occupata
si cerca un'altra posizione
Libera utilizzando una
strategia predefinita

INDIRIZZAMENTO APERTO

- Funzione ausiliaria $c(K, i)$


- La funzione ausiliaria deve soddisfare:

$$c(K, 0) = h(K)$$

$$\{c(K, 0), c(K, 1), \dots, c(K, m-1)\} = \{0, \dots, m-1\}$$

- Si considerano, in ordine, le posizioni:

$$c(K, 0), c(K, 1), c(K, 2), \dots$$

FUNZIONI AUSILIARIE : Esemp:

Scansione Lineare

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

Scansione quadratica

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

con c_1, c_2 opportuni (es. $c_1 = c_2 = 0.5$)

hashing doppio

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

$h' : U \rightarrow \{0, \dots, m-1\}$ è una seconda funzione hash

Inserimento (elemento e, chiave k)

$v[0..m-1]$ tabella

$i \leftarrow 0$

WHILE $i < m$ AND $v[c(k, i)]$ è occupata DO

$i \leftarrow i + 1$

IF $i < m$ THEN $v[c(k, i)] \leftarrow (e, k)$ // inserisci elemento

ELSE errore // tabella piena

Ricerca (chiave k) \rightarrow elemento

$v[0..m-1]$ tabella

$i \leftarrow 0$

WHILE $i < m$ AND $v[c(k, i)]$ è occupata DO

AND $v[c(k, i)].chiave \neq k$ DO

$i \leftarrow i + 1$

IF $i = m$ OR $v[c(k, i)]$ è libera THEN

RETURN null // non c'è

ELSE

RETURN $v[c(k, i)].elemento$ // trovato

Cancellazione (chiavi K) ?


cancellazione logica

Esempio:

$$f : \{a, \dots, z\} \rightarrow \{0, \dots, 15\}$$

x	f(x)
a	0
b	1
c	2
d	3
e	4
f	5
g	6
h	6
i	7
j	7
k	7
l	8
m	9
n	10
o	10
p	11
q	12
r	12
s	13
t	13
u	14
v	14
w	15
x	15
y	15
z	15

funzioni hash:

$$h(k) = f(\text{prima lettera di } k)$$

$g(k) = \text{più piccolo numero } p$
t.c. p è primo, $p \neq 2$ e
 $p \geq f(\text{terza lettera di } k)$

hashing doppio con funzione:

$$c(k, i) = (h(k) + i \cdot g(k)) \bmod 16$$

inserire nell'ordine:

salmone
tonno
rombo
aringa
cernia
aragosta
totano
ostrica
nasello
palombo

anguilla
merluzzo
tonno

0	ARANGA
1	
2	CERVIA
3	ARAGOSTA
4	NASELLO
5	
6	
7	OSTRICA
8	TONNO
9	
10	TOTANO
11	PALOMBO
12	ROMBO
13	SALMONE
14	
15	

COSTO OPERAZIONI (gestione interna)

I costi di ricerca, inserimento, cancellazione dipendono dal costo di scansione (posizioni da visitare)

CASO PEGGIORE:

si devono scandire tutte le chiavi presenti nella tabella

$\Theta(n)$ passi

COSTO OPERAZIONI (gestione interna)

I costi di ricerca, inserimento, cancellazione dipendono dal costo di scansione (posizioni da visitare)

CASO MEDIO: se la f₇ hash "sparpaglia" bene

n° di passi	scansione lineare	scansione quadratica hashing doppio
chiave trovata	$\frac{1}{2} + \frac{1}{2(1-\alpha)}$	$-\frac{1}{\alpha} \lg_e (1-\alpha)$
chiave non trovata	$\frac{1}{2} + \frac{1}{2(1-\alpha)^2}$	$\frac{1}{1-\alpha}$

n = # chiavi presenti

m = dim tabella

$\alpha = \frac{n}{m}$ FATTORE DI CARICO

COSTO OPERAZIONI (gestione interna - caso medio)

n° di passi	scansione lineare	scansione quadratica hashing doppio
chiave trovata	$\frac{1}{2} + \frac{1}{2(1-\alpha)}$	$-\frac{1}{\alpha} \lg_e (1-\alpha)$
chiave non trovata	$\frac{1}{2} + \frac{1}{2(1-\alpha)^2}$	$\frac{1}{1-\alpha}$

$$\frac{1}{1-\alpha}$$

$n = \#$ chiavi presenti

$m = \dim$ tabella

$$\alpha = \frac{n}{m} \quad \text{FATORE DI CARICO}$$

se $\alpha = \frac{l}{2}$

$$\frac{1}{1-\alpha} = 2$$

se $\alpha = \frac{3}{4}$

$$\frac{l}{1-\alpha} = 4$$

se $\alpha \rightarrow 1$

$$\frac{l}{1-\alpha} \rightarrow \infty$$

Il n° di passi di calcolo dipende da α , cioè da quanto la tabella è piena

$$n \approx 100$$

$$n = 10.000$$

$$m \approx 200$$

$$m = 20.000$$

REHASHING

Quando la tabella viene riempita oltre la soglia stabilita (es $\frac{1}{2}$) viene creata una tabella più grande (di solito il doppio) in cui vengono trasferiti tutti gli elementi:

Problemi:

- Funzione hash per la nuova tabella
- Costo in termini di tempo

FUNZIONE HASH

$f(k)$: funzione che restituisce un intero
in un range molto grande

$$h_m(k) = f(k) \bmod \underline{m}$$



fz. paritetica

rispetto a dim tabella



Se f è uniforme

anche h_m è uniforme

COSTO RE-HASHING

- Tavella inizialmente vuota
- Sequenza di operazioni fra cui N inserimenti
- Re-hashing ogni volta che α supera $\frac{1}{2}$,
raddoppiando la dimensione della Tavella

COSTO?

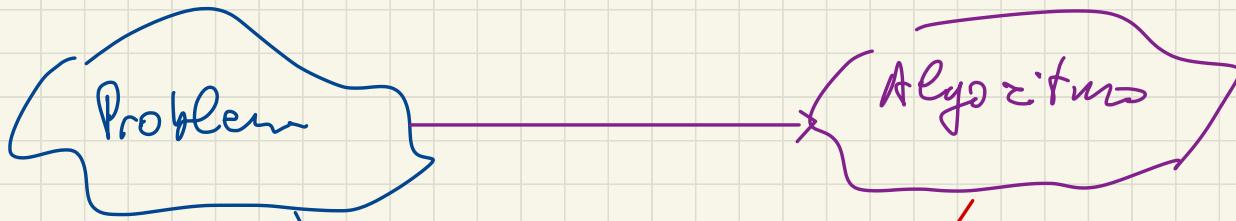
COSTO RE-HASHING

- Tabella inizialmente vuota
- Sequenza di operazioni fra cui N inserimenti
- Re-hashing ogni volta che α supera $\frac{1}{2}$, raddoppiando la dimensione della tabella

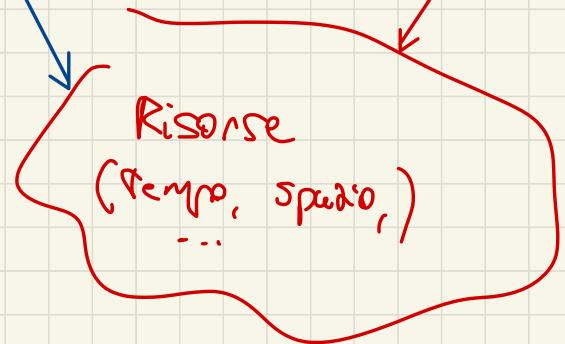
Si dimostra che :

- il numero complessivo di inserimenti dovuto alle operazioni di re-hashing è $O(N)$
- se ciascun inserimento costa $O(1)$, il costo ammontazzato dell'operazione di inserimento rimane $O(1)$

Complessità computazionale



STIMA DELLE
RISORSE
NECESSARIE e
SUFFICIENTI PER
RISOLVERE IL
PROBLEMA



RISORSE NECESSARIE E SUFFICIENTI

- r risorsa computazionale (tempo, spazio, ...)
- Π problemi risolvibili algoritmicamente

Quanta risorsa r è necessaria e sufficiente per risolvere Π ?

LIMITAZIONE SUPERIORE (upper bound)

$f: \mathbb{N} \rightarrow \mathbb{N}$ funzione

$f(n)$ risorsa r è SUFFICIENTE per risolvere Π
se \exists algoritmo A che risolve Π utilizzando
su OGNI INPUT di lunghezza n AL PIÙ $f(n)$
risorsa r (per ogni $n \geq 0$)

LIMITAZIONE INFERIORE (lower bound)

$g: \mathbb{N} \rightarrow \mathbb{N}$ funzione

$g(n)$ risorsa r è NECESSARIA per risolvere Π
se \nexists algoritmo A che risolve Π

ESISTE un input di lunghezza n (per ogni $n \geq 0$)
su cui A utilizza ALMENO $g(n)$ risorsa r

Esempio

$\Pi = \text{SORT}$

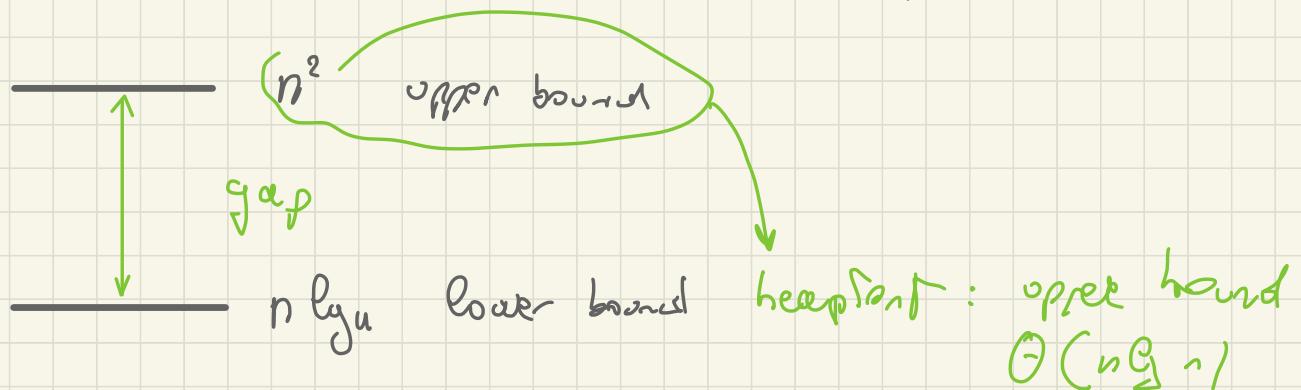
$r = \# \text{ confronti}$

• ALGORITMO inserzione, Sort

$\Theta(n^2)$ confronti

- Ogni algoritmo di ordinamento basato su confronti utilizza un n^2 di confronti ^{almeno} dell'ordine di $n \log n$ nel caso peggiore

$\Omega(n \log n)$



Esempio

Π = Prodotto matrici $n \times n$

r = # operazioni elementari (Tempo)

Upper bound

$$\Theta(n^3)$$

prodotto righe x colonna

$$\Theta(n^{2.81})$$

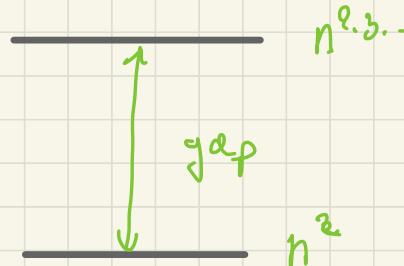
algoritmo di Strassen

$$\Theta(n^{2.3...})$$

algoritmo migliore noto

Lower bound

$$\Omega(n^2)$$



COMPLESSITÀ COMPUTAZIONALE

Classificazione dei problemi in base alle risorse utilizzate per la loro soluzione

CLASSE DI COMPLESSITÀ

Insieme di problemi che possono essere risolti utilizzando la stessa quantità di una determinata risorsa

Esempio

CLASSE P

Classe dei problemi risolvibili in
tempo polinomiale rispetto alla lunghezza dell'input

P

ordinamento

prod di matrici

Cennini minimi da un ente a più d'uno

:

Colorazione di grafi?

Zaino?

Combinatoria magica?

PROBLEMA

$$\Pi \subseteq I \times S$$

↗ ↙

universo delle possibili istanze (Input)

universo delle soluzioni

$(x, s) \in \Pi$ se e solo se s è una soluzione di Π su input x

Esempio

ALBERO RICOPRENTE

$I =$ grafi non orientati

$S =$ alberi

$(x, s) \in \Pi$ sse s è un albero ricoprente per il grafo x

PROBLEMA

$$\Pi \subseteq I \times S$$

universo delle
possibili istanze
(input)

universo delle soluzioni

$(x, s) \in \Pi$ sse s è una soluzione di Π su input x

problem
di ricerca

Esempio

ALBERO RICOPRENTE

$I =$ grafi non orientati

$S =$ alberi:

$(x, s) \in \Pi$ sse s è un albero ricoprente per il grafo x

PROBLEMA

$$\Pi \subseteq I \times S$$

universo delle
possibili istanze
(input)

universo delle soluzioni

$(x, s) \in \Pi$ sse s è una soluzione di Π su input x

Esempio

ALBERO RICOPRENTE MINIMO

$I =$ grafi non orientati pesati

$S =$ alberi:

$(x, s) \in \Pi$ sse s è un albero ricoprente per il grafo x

MINIMO: tra tutti gli alberi s t.c. $(x, s) \in \Pi$ cerchiamo

s^* t.c. $\text{peso}(s^*) \leq \text{peso}(s)$

problema di ottimizzazione

TIPOLOGIE DI PROBLEMI

RICERCA

Data $x \in I$ trovare $s \in S$ t.c. $(x,s) \in \Pi$

OPTIMIZZAZIONE

Data $x \in I$ trovare $s \in S$ t.c. $(x,s) \in \Pi$ che soddisfi un criterio di ottimalità fissato (es min / max)

DECISIONE $S = \{0,1\}$ risposta binaria

$(x, 1) \in \Pi$ istanze positive

$(x, 0) \in \Pi$ istanze negative

NO CONTRADDIZIONI

funzione
 $\Pi(x)=1$ ist. positiva
 $\Pi(x)=0$ " negativa

PROBLEMA

$$\Pi \subseteq I \times S$$

universo delle possibili istanze (input)

universo delle soluzioni

$(x,s) \in \Pi$ sse s è una soluzione di Π su input x

Problemi di decisione: esempi

- $I = \text{Grafi non orientati}$

Istanza $x \in I$

Quesione x è连通的?

- $I = \text{Grafi non orientati pesati } x \in \mathbb{N}$

Istanza $(x, k) \in I$

Quesione Esiste un albero ricoprente di x di peso $\leq k$?

OTTIMIZZAZIONE vs DECISIONE

Esempio: Albero ricoprente

OTTIMIZZAZIONE

Dato $G = \langle V, E, \omega \rangle$ grafo non orientato连通的 pesato
trovare un albero ricoprente di G di peso minimo

DECISIONE

Dato $G = \langle V, E, \omega \rangle$ grafo non orientato连通的 pesato
e un intero $K \in \mathbb{N}$, esiste un albero ricoprente di G
di peso $\leq K$?

CRICCA (clique)

Dizionario

1



cricca¹

/cric-ca/

sostantivo femminile

1. Gruppo d'intriganti, intenti a procurarsi reciproci favori; combriccola, camarilla.
 - ESTENS. •FAM.
Gruppo di amici molto uniti tra loro.
"se ne sta sempre con la sua c."
2. ARCAICO
Combinazione di tre carte; anche, nome di un gioco di carte.

Origine

Dal fr. *clique* 'combriccola' •sec. XV.



cricca²

/cric-ca/

sostantivo femminile

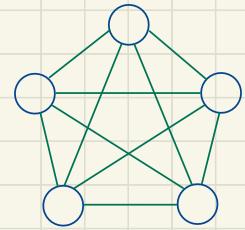
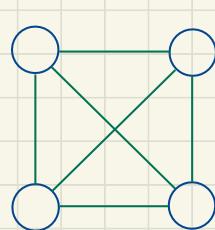
Piccola crepa sottile e profonda che si verifica nei laminati o nei getti metallici, dovuta a difetti di lavorazione.

Origine

Der. di *criccare* •1956.

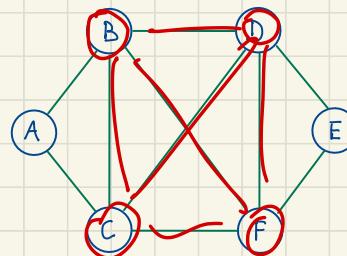
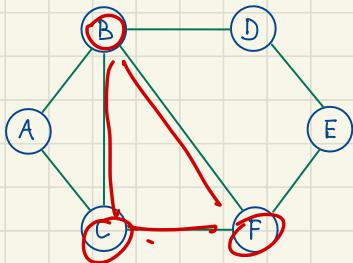
CRICCA (clique)

Grafo non orientato completo



CRICCA IN UN GRAFO

Sottografo completo



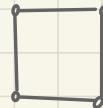
OTTIMIZZAZIONE

vs DECISIONE

Esempio: Clique

OTTIMIZZAZIONE

Dato $G = \langle V, E \rangle$ grafo non orientato trovare un sottografo completo con il max numero di vertici.



DECISIONE

Dato $G = \langle V, E \rangle$ grafo non orientato e un intero $K \in \mathbb{N}$

stabilire se G contiene una clique di K vertici

(di almeno K vertici)

PROBLEMI DI DECISIONE e CLASSI DI COMPLESSITA'

$s, t : \mathbb{N} \rightarrow \mathbb{N}$ funzioni

$\text{TIME}(t(n))$ = classe dei problemi di decisione

risolvibili da algoritmi che usano tempo $O(t(n))$

$\text{SPACE}(s(n))$ = classe dei problemi di decisione

risolvibili da algoritmi che usano spazio $O(s(n))$

CLASSE P

$$P = \bigcup_{c \geq 0} \text{TIME}(n^c)$$

classe dei problemi di
decisione riconoscibili
in tempo polinomiale
rispetto al problema
dell'input

ALTRÉ CLASSI

$$\text{PSPACE} = \bigcup_{c>0} \text{SPACE}(n^c) \quad \text{sono polinomiali}$$

$$\text{EXPTIME} = \bigcup_{c>0} \text{TIME}(2^{n^c})$$