

AI CORPO PARTE IN CORSO

CLASSIFICARE PROBLEMI IN BASE ALLA COMPLICAZIONE

- Facili \rightarrow Problemi che SAPPIAMO risolvibili in modo EFFICIENTE con un ALGORITMO
- Difficili \rightarrow si possono risolvere MA CON UN ALGORITMO che non è efficiente
EFFICIENTE
 - \hookrightarrow Dificalissimi \rightarrow se riuscissi a risolverli CON ALGORITMI EFFICIENTI
Avrai tutti gli algoritmi difficili sarebbero facili
- Piuttosto difficili \rightarrow li SO risolvibili CON UN ALGORITMO NON EFFICIENTE
E posso dimostrare che non esiste UN ALGORITMO
EFFICIENTE
- Impossibili \rightarrow posso dimostrare che non esiste UN ALGORITMO che lo risolve

PROBLEMA COMPUTAZIONALE

È dato da UN INPUT: come sono dati i dati su cui opera
 E UN OUTPUT: come sono dati i risultati
 E una loro RELAZIONE tra INPUT e OUTPUT

ESEMPIO: IL PROBLEMA DELL'ORDINAMENTO DI INTERI

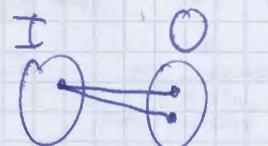
INPUT: sequenza di n interi: A

OUTPUT: una permutazione π dei primi n naturali

$$\text{t.c. } A[\pi(1)] \leq A[\pi(2)] \leq \dots \leq A[\pi(n)]$$

$$\pi \in I \times O$$

in senso delle relazioni



ALGORITMO

È UNA PROCEDURA COMPOSTA DA UN NUMERO FINITO DI OPERAZIONI CONCRETE E GENERALI t.c. SE APPLICATA A UN'ISTANZA CALCOLA L'OUTPUT

\downarrow
ASSEGNAZIONE DI VALORI ALL'INPUT

UN ALGORITMO È CORRETTO SE CALCOLA L'OUTPUT PER TUTTI GLI POSSIBILI ESTATE DI UN PROBLEMA IN UN TEMPO FINITO

Problema dell'arco minimo

INPUT: $G = (V, E)$ pesato sui archi $w: E \rightarrow \mathbb{R}$

pesi sugli archi

OUTPUT: un arco $e \in E$ t.c. $w(e)$ è minimo

Complessità: $O(|E|)$ oppure

$O(|V|^2)$ oppure $O(|V| + |E|)$

con matrice

con liste
adiacenza

è semplice

→
Rappresentazioni
dei grafici

Problema raggiungibilità

INPUT: $G = (V, E)$ e 2 vertici $s, t \in V$

OUTPUT: Vero se \exists $s \xrightarrow{e_1} s \xrightarrow{e_2} \dots \xrightarrow{e_n} t$, falso altrimenti

cammino da s a t

Complessità: $O(2^{|E|})$ → PUNTO 1 È UNO STATO $O(2^{|E|} \cdot n)$

↓
ESPOSIONE

PUNTO 2 È SEMPRE?
USANDO DEI TAVOLI

Problema TSP - TRAVELLING SALESMAN PROBLEM

INPUT: $G = (V, E)$ completo e pesato suili archi

Output: Un ciclo che passa per tutti i vertici una sola volta

↓
Passa per tutti i vertici
T. C. il peso è minimo
stesso vertice

Complessità: $O(|V|!)$ è difficilissimo

CON DEI TAVOLI LO PUOI RISOLVERE A GRADUATORIA MA RIMANENDO

ALGORITMO EFFICIENTE \rightarrow COMPLESSITÀ ALGOARITMICA POLINOMIALE

ALGORITMO NON EFFICIENTE \rightarrow COMPLESSITÀ EXPONENZIALE

Esempio:

con $n=40$

RISULTATO \downarrow

$n=50$

$O(n)$

0,00004 secondi 0,00005 secondi

$O(2^n)$

12 giorni 36 anni

$O(n^3)$

0,125 secondi

PROBLEMA TRATTABILE Risolvibile con algoritmo a complessità polinomiale

PROBLEMA INTRATTABILE Risolvibile ma complessità troppo alta

- dimostrabilmente intrattabili;
- forse intrattabili
- probabilmente intrattabili → difficili

Problemi indecidibili → impossibili → posso dimostrare che non esiste un algoritmo che lo risolve

GS.

INPUT: insieme V di n elementi

OUTPUT: funzione totale i sottoinsiemi di V

→ 2^n output quindi è dimostrabilmente intrattabile

Problema primality

INPUT: un numero k

OUTPUT: true se k è primo, false altrimenti

ALGORITMO: provo tutti i numeri fino a \sqrt{k} e vedo se il resto per la divisione è 0

(Complessità computazionale: $O(\sqrt{k})$) Sembra efficiente ma non lo è

$$\text{Dimensione input in bit: } n = \lceil \log_2 k \rceil$$

$$k = 2^n$$

$$\text{Complessità: } O(\sqrt{2^n})$$

vediamo è un algoritmo esponenziale

però la rappresentazione in bit non è ragionevole perché

probabilmente l'algoritmo non è efficiente

Se pensi la rappresentazione come numero di stringhe è $O(\sqrt{n})$ e quindi è efficiente

Quindi il problema è trattabile (però)

Macchina di Turing



Tesi di Church-Turing

Un qualsiasi algoritmo può essere rappresentato tramite una macchina di Turing

RISOLVE GLI STESSI PROBLEMI MA L'ESECUZIONE PUÒ ESSERE DIVERSA

$$M = (Q, \Sigma, q_0, \delta)$$

Macchina di Turing ↓ Stato attuale ↓ punzione di transizione
 ↓ ↓ ↓ ↓
 Stato iniziale

Q = insieme finito di stati

Σ = alfabeto su cui opera, finito, che contiene i simboli di controllo:

▷ □

q_0 = stato iniziale, $q_0 \in Q$

$$\delta = Q \times \Sigma \rightarrow Q \times \Sigma \times \{+, -, -\}$$

↓ ↓ ↓
 Stato-simbolo Stato-simbolo-movimento
 Attuaci AVANZA SCRIVI UN SIMBOLO SPOSTA LA TESTINA
 ↓
 NUOVO STATO

per dare concerenza, scrivere Hello world

INPUT: stringa x (sequenza di simboli)

Output: la stringa "Hello"

$$M = (Q, \Sigma, q_0, \delta)$$

$$\Sigma = \{ H, e, l, o, >, \sqcup \}$$

↓
BLANK

Si parte da scacchi a sinistra, nella q₀
e puoi vedi man mano

$Q = \{ q_0 \}$

$$(q_0, \sqcup) \rightarrow (q_1, H, \rightarrow)$$

$$(q_1, \sqcup) \rightarrow (q_2, e, \rightarrow)$$

$$(q_2, \sqcup) \rightarrow (q_3, l, \rightarrow)$$

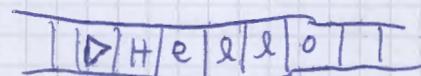
$$(q_3, \sqcup) \rightarrow (q_4, l, \rightarrow)$$

$$(q_4, \sqcup) \rightarrow (q_5, o, \rightarrow)$$

$$(q_5, l) \rightarrow (q_5, l, \rightarrow)$$

$$(q_5, e) \rightarrow (q_5, e, \rightarrow)$$

$$(q_5, h) \rightarrow (q_5, h, \rightarrow)$$



↓
corrisponde all'inizio dell'input

CANCELLAZIONE $(q_6, \frac{H}{e}) \rightarrow (q_6, \sqcup, \rightarrow)$

$$(q_6, \sqcup) \rightarrow (q_7, \sqcup, \rightarrow)$$

$$(q_7, \sqcup) \rightarrow (q_7, \sqcup, \rightarrow)$$

$$(q_7, >) \rightarrow (q_0, >, \rightarrow)$$

$$M = (Q, \Sigma, q_0, \delta)$$

$$Q = \{ q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7 \}$$

$$\Sigma = \{ H, e, l, o, >, \sqcup \}$$

CONFIGURAZIONE

STATO GEOSALE DELL'ATMOSFERA DI TUTTO IN UN CERTO MOMENTO

$(q_0, \text{simbolo sotto la testina}, \text{stazione A} \rightarrow \text{risp. A} \text{ alla testina}, \text{stazione}$

\downarrow
simbolo
sotto
la testina)

grado che c'è sul nastro

se l'input è vuoto



CONFIGURAZIONE INIZIALE: $(q_0, \sqcup, \triangleright, \text{uu...})$

\downarrow
OPPURE ε

stazione vuota



$(q_0, \sqcup, \triangleright, \varepsilon) \vdash (q_1, \sqcup, \triangleright H, \varepsilon)$

CONFIGURAZIONE SUCCESSIVA

$\vdash (q_0, \sqcup, \triangleright, \varepsilon) \vdash (q_1, \sqcup, \triangleright H, \varepsilon)$

$\vdash (q_1, \sqcup, \triangleright H e, \varepsilon) \vdash (q_2, \sqcup, \triangleright H e l, \varepsilon)$

$\vdash (q_2, \sqcup, \triangleright H e l l, \varepsilon) \vdash (q_3, \sqcup, \triangleright H e l l o, \circ)$

$\vdash (q_3, \sqcup, \triangleright H e l l o, \circ) \vdash (q_4, \sqcup, \triangleright H e l l o, \circ)$

$\vdash (q_4, \sqcup, \triangleright H e l l o, \circ) \vdash (q_5, \sqcup, \triangleright H e l l o, \circ)$

configurazione finale è spagnola

PER LE MACHINES DI TURING CI SONO DUE TIPI D'ARANCHE L'UNA SONO SIA FINITO

PROBLEMA: INCREMENTO

INPUT: UN NUMERO x RAPPRESENTATO COME STRINGA BINARIA

OUTPUT: IL NUMERO $x+1$ RAPPRESENTATO COME STRINGA BINARIA

D | 1 | 0 | 0 | 1 |

$$(Init, \%) \rightarrow (Init, 0, \rightarrow)$$

$$(Init, \sqcup) \rightarrow (Inc, \sqcup, \star)$$

$$(Inc, 0) \rightarrow (Half, \wedge, -)$$

$$(Inc, 1) \rightarrow (Inc, 0, \star)$$

$$(Inc, \Delta) \rightarrow (Uno, \Delta, \rightarrow)$$

$$(Uno, 0) \rightarrow (Zero, 1, \rightarrow)$$

$$(Zero, 0) \rightarrow (Zero, 0, \rightarrow)$$

$$(Zero, \sqcup) \rightarrow (H, 0, -)$$

$$M = (Q, \Sigma, Init, f)$$

$$Q = \{Init, Inc, Uno, Zero, H\}$$

$$\Sigma = \{0, 1, \Delta, \sqcup\}$$

$$(Init, 1, \Delta, 00) \vdash (Init, 0, \Delta 1, 0) \vdash (Init, 0, \Delta 10, \varepsilon) \vdash$$

$$\vdash (Init, \sqcup, \Delta 10, \varepsilon) \vdash (Inc, 0, \Delta 10, \varepsilon) \vdash (H, 1, \Delta 10, \varepsilon)$$

Programa para achar o

Input: $x \in \{a, b\}^*$ Output: γ se x é palíndromo

DAGBA

$$(Init, a) \rightarrow (LA, \sqcup, \rightarrow)$$

$$\xrightarrow{\text{escrever } a} (LA, \overset{a}{\sqcap}, \rightarrow)$$

$$\xrightarrow{\text{convergir}} (LA, \sqcup) \rightarrow (CA, \sqcup, \leftarrow)$$

$$(CA, a) \rightarrow (BACK, \sqcup, \leftarrow)$$

$$(BACK, \overset{a}{\sqcap}, \leftarrow) \rightarrow (BACK, \overset{a}{\sqcap}, \leftarrow)$$

$$(BACK, \sqcup) \rightarrow (Init, \sqcup, \rightarrow)$$

$$(Init, b) \rightarrow (LB, \sqcup, \rightarrow)$$

$$(LB, \overset{b}{\sqcap}, \rightarrow) \rightarrow (LB, \overset{b}{\sqcap}, \rightarrow)$$

$$(LB, \sqcup) \rightarrow (CB, \sqcup, \leftarrow)$$

$$(CB, b) \rightarrow (BACK, \sqcup, \leftarrow)$$

$$(Init, \sqcup) \rightarrow (OK, \sqcup, \leftarrow)$$

$$(OK, \sqcup) \rightarrow (OK, \sqcup, \leftarrow)$$

$$(OK, \triangleright) \rightarrow (WN, \triangleright, \rightarrow)$$

$$(WN, \sqcup) \rightarrow (WN, \sqcup, \rightarrow)$$

$$(CA, b) \rightarrow (NO, \sqcup, +)$$

$$(NO, \frac{a}{b}) \rightarrow (NO, \sqcup, +)$$

$$(NO, \triangleright) \rightarrow (NO, \triangleright, +)$$

$$(NO, \sqcup) \rightarrow (NO, \sqcup, -)$$

CHECKS

$$(CB, a) \rightarrow (NO, \sqcup, +)$$

$$(CA, \sqcup) \rightarrow (OK, \sqcup, +)$$

$$(CB, \sqcup) \rightarrow (OK, \sqcup, +)$$

Grammatica

INPUT $x \in \{a, b, c\}^*$ OUTPUT $x = a^n b^m c^n ?$ → può essere scritta in questo modo!

$$\Sigma = \{a, b, c, \triangleright, \sqcup, *, /, \emptyset, \perp\}$$

$$(ini, \sqcup) \rightarrow (x_B, \sqcup, -)$$

$$(ini, A) \rightarrow (LA, a, \rightarrow)$$

$$(ini, \frac{b}{c}) \rightarrow (NO, \sqcup, -)$$

$$(LA, a) \rightarrow (LA, a, \rightarrow)$$

$$(LA, b) \rightarrow (LB, b, \rightarrow)$$

$$(LA, c) \rightarrow (NO, \sqcup, -)$$

$$(LA, \sqcup) \rightarrow (NO, \sqcup, -)$$

$$(LB, b) \rightarrow (LB, b, \rightarrow)$$

$$(LB, c) \rightarrow (LC, c, \rightarrow)$$

$$(LB, \frac{a}{b}) \rightarrow (NO, \sqcup, -)$$

$$(LC, c) \rightarrow (LC, c, +)$$

$$(LC, \sqcup) \rightarrow (CHECK, \sqcup, +)$$

$$(LC, \emptyset) \rightarrow (NO, \sqcup, -)$$

$$(CHECK, \frac{a}{b}) \rightarrow (CHECK, \frac{a}{b}, +)$$

$$(CHECK, \triangleright) \rightarrow (CHECK, \triangleright, -)$$

$$(CHECK', \emptyset) \rightarrow (CHECK', \emptyset, -)$$

$$(CHECK', \emptyset) \rightarrow (FINAL, \emptyset, +)$$

$$(CB, a) \rightarrow (CB, a, \rightarrow)$$

$$(CB, b) \rightarrow (CB, b, \rightarrow)$$

$$(CB, c) \rightarrow (CC, c, \rightarrow)$$

$$(CB, \emptyset) \rightarrow (NO, \sqcup, -)$$

$$(CC, b) \rightarrow (CC, b, \rightarrow)$$

$$(CC, \emptyset) \rightarrow (CC, \emptyset, -)$$

$$(CC, c) \rightarrow (CHECK, \sqcup, +)$$

$$(CHECK, \frac{a}{b}) \rightarrow (CHECK, \frac{a}{b}, +)$$

$$(CC, \sqcup) \rightarrow (NO, \sqcup, -)$$

+C - L4

$$(\text{FINAL}, \frac{k}{x}) \rightarrow (\text{FINAL}, \frac{k}{x}, \rightarrow)$$

$$(\text{FINAL}, \cup) \rightarrow (\text{XH}, \cup, -)$$

$$(\text{FINAL}, \frac{b}{c}) \rightarrow (\text{NO}, \cup, -)$$

$$t_M(x) = \# \text{config M attivata su input } x$$

TOP-UP MACHINE

TEMPO DI CALCOLO
DELLA CONVERGENZA?

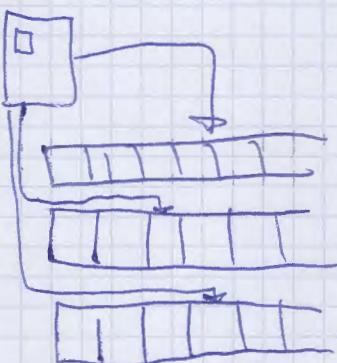
$$T_M(h) = \max_{x \in \Sigma^*} \{t_M(x) \mid |x| = h\}$$

↓
ACKERMANN

TEMPO DI CALCOLO MASSIMO MA
GLI INPUT DI UNA CERTA LUNGHEZZA H

$$T_M(h) = O(h^2) \quad \text{nel caso precedente, poiché in genere della macchina}$$

Macchina di Turing multivettore



CON K NASTRI FINITI E COSTANTI

$$M = (Q, \Sigma, \phi_0, \delta) \quad \text{COMPRENDE ANCHE I NASTRI}$$

$$\delta: Q \times \Sigma^k \rightarrow Q \times \Sigma^k \times \{\leftarrow, -, \rightarrow\}^k$$

$$T_{M'}(h) = O(h) \quad \text{COMPLESSITÀ INERENTE DALLA MACCHINA}$$

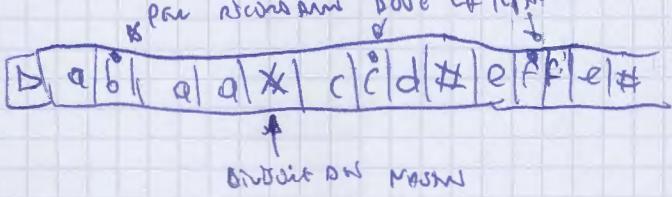
Teorema

MACCHINA DI TURING CON K NASTRI

SIA M UNA K-TM AVERE ESISSE UNA M' A NASTRI SEMPRE

EQUIVALENTE

* per numero dove la testina



$$\Sigma \rightarrow \Sigma'$$

$$\{a, b, c, d, e, f, \Delta, \cup\} \rightarrow \{a, b, c, d, e, f, \Delta, \cup, \square, \# \}$$

\square è DOPPIO
di una costante
e ANIMA COSTANTE

$$\delta(q, b, c, f) = (q', o, c, e, \rightarrow, *, -)$$

COME RICCRE A SINISTRA
QUINDI È UNA SINGOLA MAPPATURA

UNA FUNZIONE
 $T_n(n)$ DI COMPLESSITÀ, ALLORA $T_{n+1}(n) = O((T_n(n))^2)$

AUMENTA LA COMPLESSITÀ PER OGNI UNA SUB MAGGIORI?
 PERSO SIGNIFICHI QUESTO

$$Q \rightarrow Q' \quad |Q'| = O(|Q| \cdot |\Sigma|^k)$$

$$\Sigma \rightarrow \Sigma' \quad |\Sigma'| = O(|\Sigma|)$$

DATA M CON Σ T.C. $|\Sigma| \geq 4$ ESISTE UNA M' EQUIVALENTE

$$\text{T.C. } |\Sigma'| = 4$$

$$\forall x \in \Sigma' \quad M'(x) = M(f(x))$$



a b c d a b c d ..

$$\begin{aligned} a &= 00 \\ b &= 01 \\ c &= 10 \\ d &= 11 \end{aligned}$$

$$f: \Sigma^* \rightarrow \{0,1\}^*$$

TRANSFORMA VALORES DI M ENTRANTI CON $0 \& 1$



0 0 1 0 0 0 ..

a c a

$\sigma \in \Sigma$ RAPPRESENTATO CON $\log |\Sigma|$ BIT
= k

$$t_n(x)$$

$$\begin{aligned} t_{n'}(f(x)) &= O(k \cdot t_n(x)) \\ &\downarrow \text{costante} = (\log |\Sigma|) \\ &= t_{n'}(f(x)) = O(t_n(x)) \end{aligned}$$

Adattivamente aumenta lo stesso tempo perché l'aggiunta logaritmica è costante

$$T_m(n)$$

$$T_{n'}(h) = O(T_m(h))$$

TC-L5

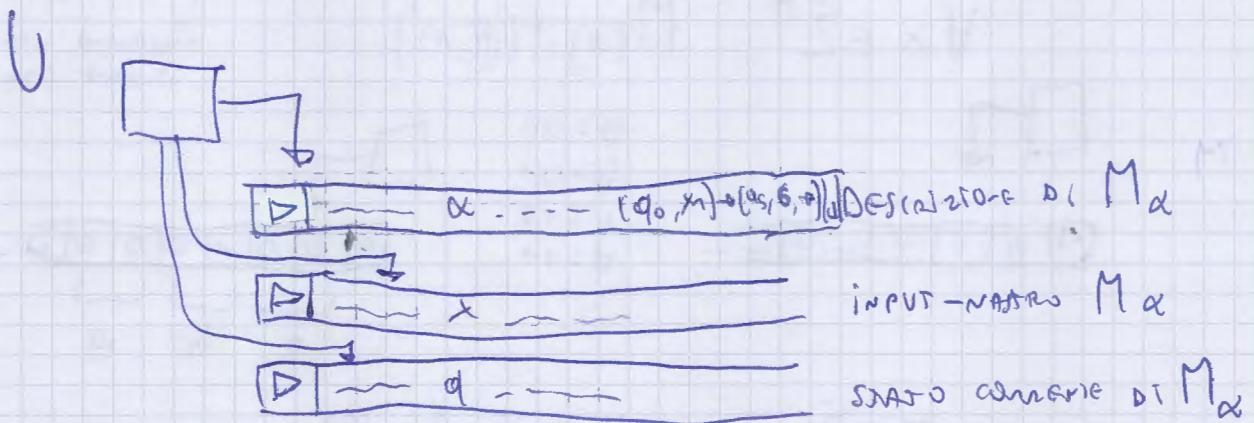
MACCHINA DI TURING UNIVERSALE

ESISTE UNA TM U t.c. $\forall \alpha, x \in \Sigma^*$,

$$U(\alpha, x) = M_\alpha(x)$$

IL RISULTATO È OTENUTO DALLA MACCHINA DI TURING
 α SUL DATO x

M_α È LA TM RAPPRESENTATA DA α



$$f: \Sigma^* \rightarrow \Sigma^*$$

$$f: \Sigma^* \rightarrow \{0, 1\}$$

$L = \{x \in \Sigma^* \mid f_D(x) = 1\}$

L = {ε, abc, aaabbcc, ...}

$$f(x) = y$$

$$f_D(x, y) = \begin{cases} 1 & \text{se } f(x) = y \\ 0 & \text{altrimenti} \end{cases}$$

LINGUAGGIO DECIDIBILE O RICONOSCIBILE SE ESISTE UNA TM
 M t.c. $\forall x \in \Sigma^*$ $M(x) = 1$ se $x \in L$ e $M(x) = 0$ se $x \notin L$

M decide L

L è SEMI DECIDIBILE o RICORSIVAMENTE ENUMERABILE

SE ESISTE TM M t.c. $\forall x \in \Sigma^*$

(esistenza solo ac. regresso)

$$M(x) = 1 \text{ se } x \in L$$

M ACCETTA L

$$\begin{cases} M(x) = 0 \\ \text{ } \end{cases} \text{ se } x \notin L$$

SE LA SISTEMA NON APPARTE AL
LIMITESSO SOLO REGRESSIVO O ORARIO
POSSONO TUTTI TERMINARE

↳ LOOP INFINTO

AVERA E' ANNULLATA



① L è REGOLARE $\rightarrow L$ è RICORSIVAMENTE ENUMERABILE

② L è REGOLARE $\rightarrow \bar{L}$ è REGOLARE

DIMOSTRAZIONE?

$$\begin{array}{c} M \text{ decide } L \quad M(x) \xrightarrow{\text{ }} \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases} \end{array}$$

$$\Rightarrow \exists M' \quad M'(x) \xrightarrow{\text{ }} \begin{cases} 1 & \text{se } M(x)=0 \\ 0 & \text{se } M(x)=1 \end{cases}$$

$\Rightarrow M'$ decide \bar{L}

E' chiaro: \bar{L} è REGOLARE

$$\bar{L} = \{x \in \Sigma^* | M(x)=0\}$$

PROPRIETÀ

$$\textcircled{1} \quad L_{\text{REC}} \Rightarrow L_{\text{RE}} \rightarrow \text{UN Linguaggio} \xrightarrow{\text{Ricorsivo è anche}} L_{\text{REC}} \Leftrightarrow \exists M \text{ decide } L$$

$$\textcircled{2} \quad L_{\text{REC}} \Rightarrow \overline{L}_{\text{REC}}$$

Il suo complemento è composto da quattro le stringhe che non sono in L
sono solo stringhe ricorsive

SE G.DIAMO UNA SIMBOLICA
PER L'INGRESSO ALLORA LA
MACHINA DARA' YES \rightarrow SUL NASTRO
se invece LA SIMBOLICA
NON APPARISCE
dice NO.
E ovviamente
TERMINA SEMPRE

$$L_{\text{R.G.}} \Leftrightarrow \exists M \text{ accetta } L$$

$$x \in L \Rightarrow M(x) = Y$$

SE G.DIAMO UNA SIMBOLICA
per l'ingresso ALLORA TERRA CON YES
Quindi SE LA SIMBOLICA NON APPARISCE
AL LINGUAGGIO PUÒ RESTARE NO
O NON TERMINARE

$$\textcircled{3} \quad L_{\text{REC}} \Leftrightarrow L_{\text{RE}} \wedge \overline{L}_{\text{R.E.}}$$

$$L_{\text{RE}} \Rightarrow \exists M^1 \text{ accetta } L$$

$$\overline{L}_{\text{R.E.}} \Rightarrow \exists M^2 \text{ accetta } \overline{L}$$

$$x \in L \Rightarrow M^1(x) = Y$$

$$x \in \overline{L} \Rightarrow x \notin L \Rightarrow M^2(x) = Y$$

M decide L eseguendo alternativamente $M^1 \circ M^2$

insieme delle m.s.

$$|A| < |\mathcal{P}(A)|$$

$$A = \{\emptyset\} \quad \mathcal{P}(A) = \{\emptyset, \{\emptyset\}\}$$

numero di macchine di Turing

Rappresento la TM
come stringhe

per iniziare dimostrare il verso

$$\#TM \leq |\beta^*|$$

$$\beta = \{0, 1\}$$

$$f_D : \beta^* \rightarrow \beta$$

$$L_D = \{x \in \beta^* | f_D(x) = 1\} \subset \beta^*$$

$$L_D \in \mathcal{P}(\beta^*)$$

Insieme di tutti i problemi risolvibili
e quindi tutti i possibili problemi di decisione

$$\# \text{problem di decisione} = |\mathcal{P}(\beta^*)|$$

$$\#TM \leq |\beta^*| \leq |\mathcal{P}(\beta^*)| = \# \text{problem di decisione}$$

quindi $TM \leq \# \text{problem di decisione}$

Quindi esistono problemi insolvibili, ovvero che non hanno una TM

INPUT: una TM M , una stringa x

Output: $M(x) \neq \perp$? $\Leftrightarrow M$ su input x termina?

Teorema:

L_H NON è deciso

$$L_H = \{ \underbrace{M \# x} \mid M(x) \neq \perp \}$$

qualsiasi macchina codificata come stringa accetta data
stringa al input con lo simbolo in mezzo

DIM per assurdo

L_H decisivo $\Rightarrow \exists M_H$ deciso L_H

$M_H(M, x) \xrightarrow{X} N \quad \text{se } M(x) \neq \perp$

$C(c) \xrightarrow{Y} N \quad \text{se } M_H(M, M) = N$

$\xrightarrow{\perp} Y \quad \text{se } M_H(N, N) = Y$

$C(c)$

① $C(c) = Y \Rightarrow M_H(c, c) = N \Rightarrow C(c) = \perp$

② $C(c) = \perp \Rightarrow M_H(c, c) = Y \Rightarrow C(c) \neq \perp$

Teorema

L_H è RE

DIM $\exists M_A$ accacc L_H

$M_A(M, x) = Y \Leftrightarrow M(x) \neq \perp$

$\exists V \quad V(M, x) = M(x)$

$M_A(M, x) \xrightarrow{Y} N \quad \text{se } V(N, x) = Y \vee N$

$\xrightarrow{\perp} N \quad \text{se } V(N, x) = \perp$

TC-L6

Teorica

L_H non è R.E.

DIM (per assurso)

E_H è RE
 $L_H \subseteq E_H$ $\Rightarrow L_H$ è ric

(OMP(H))⁻
macchina n
piano n

$T_m(n)$

↓
Funzione
COMPLESSA
COMBINATORIALE
(fattoriale)

$DTime(f(n)) = \{L \subseteq \Sigma^* \mid \exists M \text{ decide } L \text{ in tempo } O(f(n))\}$

ESEMPIO

Immagini di problemi

Decisione (che propono)

risorse con un

algoritmo in tempo

$O(n)$

$\leftarrow DTime(n)$

$DTime(n^2) \supseteq DTime(n)$

$DTime(2^n) \supseteq DTime(n)$

$P = \bigcup_{c \in N} DTime(n^c)$

↓
Immagini dei problemi che possono
essere risolti in tempo polinomiale

$Exptime = \bigcup_{c \in N} DTime(2^{n^c})$

↓
Immagini dei problemi che sono esponenziali

$P \subseteq Exptime$

Competenza in termini di spazio

S piacere



$$s_m(x) = \#\text{CELLA DEL MASTRO (H)} \text{ SONO USATE}$$

NELL'COMBINAZIONE $M(x) - \{x\}$

MASSO LA LUNGHEZZA DI X

S grande



$$S_m(h) = \max_{x \in \Sigma^*} \{ s_m(x) \mid |x|=h \}$$



FUNZIONE DENA COMBINAZIONE
SPAZIALE DI UNA TM
NEL CASO PEGGIOR

$$S_m(h) \leq T_m(h) + h$$

in vicinanza non varia

SE M USA SPAZIO LIMITATO K LA MACCHINA POSSBNE NON TERMINARE MAI $\exists M'$ EQUIVALENTE A M CHE TERMINA PER TUTTI GLI INPUT

STATO

Configurazione
originale $Q \times \text{POSSIBILI STATE} \times \text{CONTENUTO NERVO}$

$$\text{NUMERO CONFIGURAZIONI DISTINTE} = O(|Q| \times k \times |\Sigma|^k)$$



TOGLIE LE POSIZIONI
STRUTTURE SOTTRATTI
IL LUNGHEZZA CARATTERI

Se raggiungo questo numero massimo
di combinazioni, purtroppo il risultato
più una TM, entrerà in loop perché
tornerei su una configurazione già
vista, quindi posso fermarmi

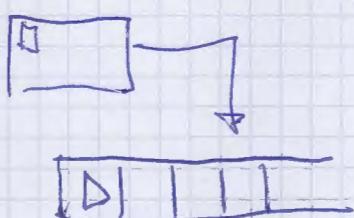
MACHINE DI TURING NON DETERMINISTICHE (NTM)

$$\text{NTM} \quad N = (\Sigma, Q, q_0, \delta)$$

$$\delta: Q \times \Sigma \Rightarrow P(Q \times \Sigma \times \{\leftarrow, -, \rightarrow\})$$

↓
Diversità

↓
Infinito delle
PARI

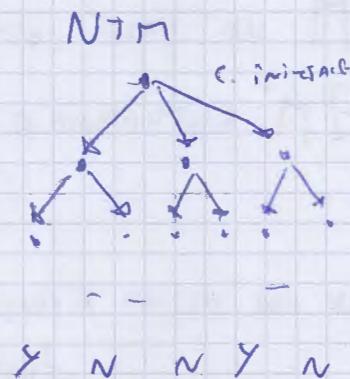
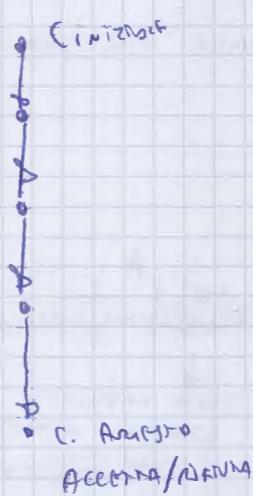


può fare più mosse diverse \rightarrow esplora tutte le opzioni in contemporanea

$$(q_0, a) \xrightarrow{\quad} (q_1, b, \rightarrow)$$

$$(q_0, a) \xrightarrow{\quad} (q_0, c, \rightarrow)$$

DIM



(NTM) N ACCETTA x SE ALMENO UN RAMO TERMINA CON YES

(NTM) N RIFIUTA x SE TUTTI I RAMI AI COMBINAZIONI TERMINANO CON NO

(NTM) N ACCETTA $L \subseteq \Sigma^*$ se $\forall x \in \Sigma^* \quad x \in L \iff N(x) = Y$

(NTM) N DECIDE $L \subseteq \Sigma^*$ se ① tutti i ramo di N su x terminano $A(x)$

$$\begin{aligned} ② \quad & x \in L \Rightarrow N(x) = Y \\ & x \notin L \Rightarrow N(x) = N \end{aligned}$$

TC-C 7

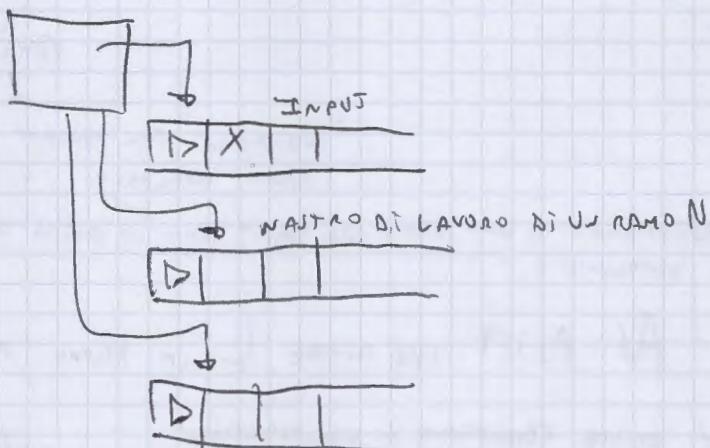
Possiamo dimostrare che le NTM non sono più poteri delle PTM
i problemi risolvibili sono gli stessi

Teorema

Sia N una NTM, esiste M PTM equivalente

$$\forall x \in \Sigma^* \quad N(x) = M(x)$$

M PTM 3 nastri



$f_N(x)$ è l'altezza dell'asta, ovvero la profondità delle regole più profonda
 $\# \text{pote} \leq b^{T_N(n)}$ $d \geq 1$
 $\# \text{nodi} \leq b^{T_N(n)}$

$$T_M(n) \leq T_N(n) \cdot 2^{b^{T_N(n)}}$$

$$T_N(n) = O(2^{O(T_N(n))}) \rightarrow \text{equa a 3 nastri}$$

$$T_{M_1}(n) = O(2^{O(T_N(n))}) \rightarrow \text{trasformazione a griglia con nastro unidimensionale}$$

TC-17

$$NTIME = \{f(n) \mid L \text{ è deciso da una NTM in tempo } O(f(n))\}$$

CLASSI DEI LINGUAGGI DI DECISIONE ACCORDO IN TEMPO POLINOMIALE?

TEMPO POLINOMIALE

$$NP = \bigcup_{c \geq 0} NTIME(n^c)$$

$$P \subseteq NP \subseteq EXP TIME$$

$$L \in NP \iff \exists c > 0 \text{ e una DTM } V \text{ (che lavora in tempo polinomiale s.t. } \forall x \in L \text{ esiste } c \in \mathbb{N} \text{ per qualche } c \in \mathbb{N}^{|\alpha|^c} \text{ tale che } V(x, c) = 1 \text{)}$$

$$\text{COPPIE} \downarrow \quad \text{LAVORA } V \uparrow \\ \text{polinomiale s.t. } \forall x \in L \text{ esiste } c \in \mathbb{N} \text{ per qualche } c \in \mathbb{N}^{|\alpha|^c} \text{ tale che } V(x, c) = 1$$

LAVORA V

\uparrow

yes

"Individua" che trovare il
caso corretto

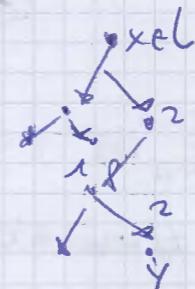
prova

caso corretto

Il VERIFICATORE è una semiparziale di una NTM che viene fatta le azioni del verificatore
 \rightarrow macchina di Turing polinomiale

$L \in NP$ perché esiste N NTM che decide L in tempo polinomiale

$\Rightarrow L \in NP$ perché esiste certificato e verificatore
certificato polinomiali



Questo può essere un certificato

$L \in NP$ perché \exists certificato e verificatore

$\Rightarrow L \in NP$ perché \exists NTM N che decide L in tempo polinomiale

Esempio: $(a \vee b \vee c) \wedge (\bar{a} \vee b \vee \bar{c})$ SAT $\in NP$?

Certificato: assegnamento di valori alle variabili input

Problema: \Rightarrow risultato
varo della
formula
esiste?

$$(1 \vee 1 \vee 0) \wedge (1 \vee 0 \vee 1) = 1 \text{ vero}$$

P = linguaggi periti per DTM in tempo polinomiale

NP = linguaggi verificati per DTM in tempo polinomiale

perché NTM non è finitamente realizzabile

SAT

Forma normale congiuntiva

INPUT: Espressione booleana in CNF



$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee z) \wedge (\bar{x} \vee \bar{z})$$

OUTPUT: Espressione è soddisfacibile?

IDEA: Criterio è l'assegnamento dei valori ai variabili

① → ARGOMENTARE Criterio polinomiale → OK È IDEA LUNGHEZZA
AVVISO

② → VERIFICAONE IN TEMPO POLINOMIALE → OK } È UN ALGORITMO

→ SAT ∈ NP

{ È UNA TM CHE DETERMINA
CHE RISOLVE SAT IN
TEMPO POLINOMIALE

dTSP

INPUT: $G = (V, E)$, $w: E \rightarrow \mathbb{R}^+$ G completo, $k \in \mathbb{R}^+$ OUTPUT: ESISTE UN CICLO CHE PUO PASSARE I VERTICI NEL SOLA VOLTA
CON COSTO $\leq k$?

dTSP ∈ NP?

IDEA: Criterio è il ciclo

① Criterio è di dimensione polinomiale nella dimensione dell'input → OK

② ∃ verificaione polinomiale → OK

$$\text{coP} = \{L | \bar{L} \in P\}$$

I linguaggi di cui posso decidere la non appartenenza al una
stringa su una macchina polinomiale non deterministica

$$\text{teo } P = \text{coP}$$

dim $L \in P \Rightarrow M$ decide \bar{L} in tempo.

$$\begin{aligned} \forall x \in \Sigma^* \quad x \in L &\Rightarrow M(x) = Y \\ x \notin L &\Rightarrow M(x) = N \end{aligned}$$

M' decide \bar{L} in tempo polinomiale

$$\begin{aligned} \forall x \in \Sigma^* \quad x \in \bar{L} &\Rightarrow M'(x) = Y \\ x \notin \bar{L} &\Rightarrow M'(x) = N \\ x \in L & \end{aligned}$$

M' ottiene da M scambiando Y/N

$$\text{coNP} = \{L | \bar{L} \in NP\}$$

insieme dei linguaggi la cui non appartenenza
può essere decisa in tempo polinomiale in
maniera non deterministica

b
non sono sicuri se questa posse essere
non polinomiale solo

$$P \subseteq NP$$

$$P \subseteq \text{coNP}$$

$$P \subseteq NP \cap \text{coNP}$$

↓
intersezione

$$NP \stackrel{?}{=} \text{coNP}$$

non lo sappiamo

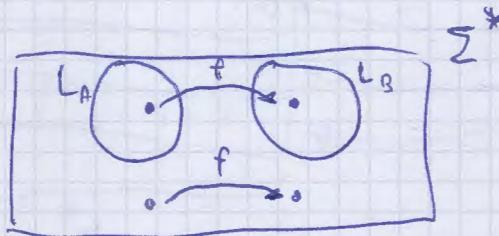
Riduzioni polinomiali

$f: \Sigma^* \rightarrow \Sigma^*$ (Accade in teoremi polinomiali da DTM)

$$\text{t.c. } \forall x \in \Sigma^* \quad x \in L_A \iff f(x) \in L_B$$

Sf. f si dice **riduzione polinomiale** da un linguaggio L_A a un linguaggio L_B
più facile (TF)

$L_A \leq_p L_B$ L_A è riducibile polinomialmente a L_B



f trasforma elementi per linguaggio A in elementi per linguaggio B
e elementi che non sono di A in elementi che non sono di B

NON SO SE È MOLTO BUON DEFINIZIONE DI \subseteq → MOLTI UGLIOLI HA DATO

TEO.

$$L_A \leq_p L_B \wedge L_B \in P \Rightarrow L_A \in P$$

TEO.

$$L_A \leq_p L_B \wedge L_B \leq_p L_C \Rightarrow L_A \leq_p L_C \quad \text{TRANSITIVITÀ}$$

$f \qquad g \qquad g(f)$
 $f \circ g$

DGP

$$L \text{ è NP-HARD se } \forall L' \in \text{NP. } L' \leq_p L$$

SE TUTTI I LINGUAGGI IN NP SI RIDUCANO
POLINOMIALMENTE A L

LINGUAGGI PIÙ COMPLICATI
HANNO QUANTO PIÙ GESI
MASSI LINGUAGGI IN NP → NON PIÙ FACILI

PRF

$$L \text{ è NP-completo se } \begin{cases} \textcircled{1} L \in \text{NP} \\ \textcircled{2} L \text{ è NP-HARD} \end{cases}$$

SOPRATTUTTO OGNI NP-HARD CHE SONO ANCHE NP

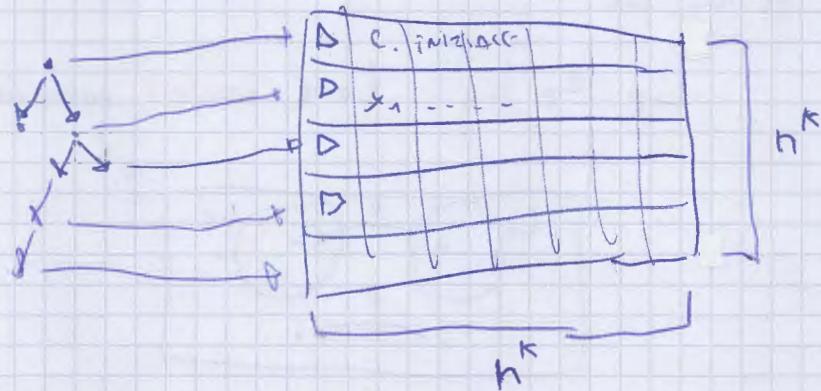
TC-C8

TEO.

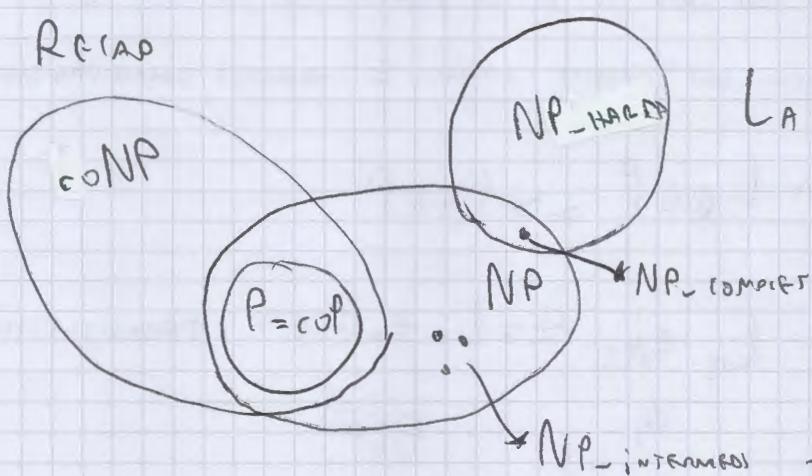
se $L_A \leq_p L_B \wedge L_A \in NP\text{-HARD} \Rightarrow L_B \in NP\text{-HARD}$

Teoria di Cook-Levin

TEO SAT \in NP-HARD



RCIAO



$L_A \in NP\text{-HARD} \wedge L_A \in P \Rightarrow P = NP$

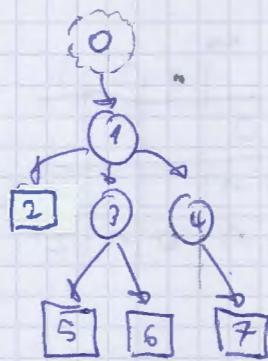
SLIDE 7

- è la concentrazione tra simboli

$$R(0) \rightarrow 0$$

$$R(1) \rightarrow 1 \cdot R(0) = 10$$

$$R(2) \rightarrow 1 \cdot R(1) = 110$$



PENSIAMO DA $B = \emptyset$ AGGIUNGIAMO UN ULTERIORE BIT VETTORE CHE È
A RIPETIZIONE QUANTE VOLTE È IL SUO GRADO CON UNO ZERO FINALE
NUMERO DI FIGLI POSSIBILI
③ HA GRADO 3 QUINDI 1110

$$B = [1110011010000]$$

AGGIUNGONO CO: ① (0) INIZIALE

OGNI A RAPPRESENTA UN NUOVO E OGNI 0 UN NUOVO FIGLIO

$$B = [101110011010000]$$

— SLIDE 8

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{matrix} \quad i=7 \rightarrow \text{RANK}(7)=4 \\ \text{SELECT}(4)=2$$

CALCOLA IN $\rightarrow O(h)$ -TIME

SULL'ULTIMA CANCELLA NELLA

$\xrightarrow{\text{SOMMA}} S = \underbrace{A A A}_{A_3} \underbrace{B B}_{B_2} \underbrace{A A A A}_{A_4} \underbrace{C}_{C_1} \underbrace{B B C C C}_{B_2 C_3} \rightarrow$ ZERI CON UNA CANCELLAZIONE

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{matrix}$$

PRIMA IPA

$$0111233445 \rightarrow$$
 SOMMA I° SALVO IL RANK DI OGNI CELLA DEL VETTORE

CONTINUO DAL SLIDE 9

VETTORE F \rightarrow FIRER
 $f \rightarrow$ SCALAR

TC-G1
T₀T₁C₀

	1	2	3
000			
001			
011	0	1	2

→
since g

For n=0 and 12?

Outline

1 Strutture dati succinte

2 LOUDS

3 Wavelet Tree

4 Altre strutture dati succinte

5 Succinct data Structures Library

6 Conclusioni

7 Bibliografia e riferimenti

Motivazione

L'aumentare della potenza di calcolo e delle strumentazioni tecniche hanno portato alla sempre maggior produzione di dati.

Esempi

- database
- information retrieval
- bioinformatica etc...
-

A cosa siamo interessati

- gestione di tali dati
- indicizzazione e interrogazione
- compressione

Una veloce catalogazione

Possiamo catalogare (in modo non esaustivo) le strutture dati in base alla quantità di bit necessari a memorizzare (tramite una certa rappresentazione in memoria) dei dati [1].

Z = numero di bit (teoricamente) ottimale per memorizzare un certo dato

■ **strutture dati implicite:** $Z + O(1)$ bit (es. $Z + 14$ bit)

■ **strutture dati succinte:** $Z + o(Z)$ bit (es. $Z + \log Z$ o $Z + \sqrt{Z}$ bit)

■ **strutture dati compatte:** $O(Z)$ bit (es. $5Z$ bit)

Si ricorda che, avendo noi $x_0 = +\infty$

$$\text{se } \lim_{x \rightarrow x_0} \frac{f(x)}{g(x)} = 0 \implies f(x) = o_{x_0}(g(x))$$

Strutture dati succinte

strutture dati succinte: $\mathcal{Z} + o(\mathcal{Z})$ bit

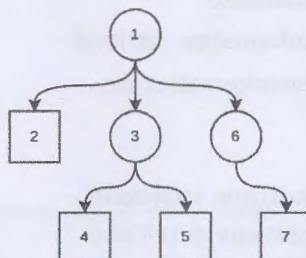
- **spazio** richiesto "vicino" al bound teorico ($o(\mathcal{Z})$ bit aggiuntivi)
- **tempo** richiesto per le operazioni "comparabile" a quello che si avrebbe senza vincoli sullo spazio

Rappresentazione di un albero

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi

Rappresentazione "raw"

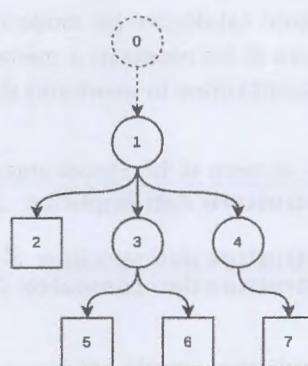
- una rappresentazione a pointer richiede $\mathcal{O}(n \log n)$ -space
- infatti servono $\mathcal{O}(nw)$ bit con $w \geq \log_2 n$ dimensione in bit di un pointer



LOUDS

Rappresentazione Level Order Unary Degree Sequence (LOUDS), Jacobson - 1989 [2]

- rappresentazione basata sulla rappresentazione in *left-to-right level-order* di un albero (*ordered-tree*)
- si considera nella rappresentazione un nodo detto *super-root*, qui indicato con 0
- si memorizza la sequenza D dei *degree* dei nodi



$$D = <1, 3, 0, 2, 1, 0, 0, 0>$$

LOUDS

Rappresentazione Level Order

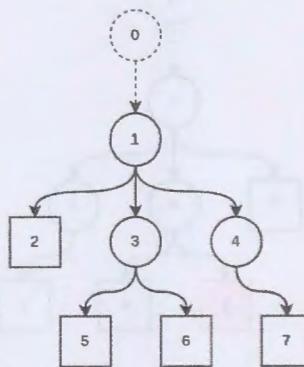
Unary Degree Sequence

(LOUDS), Jacobson - 1989 [2]

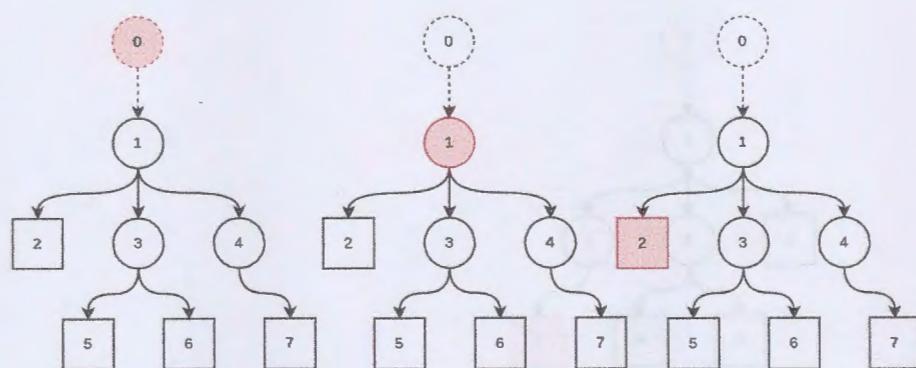
- memorizziamo D con un *prefix code binario (unary code)*:

$$R(i) = \begin{cases} 0 & \text{se } i = 0 \\ 1 \cdot R(i-1) & \text{altrimenti} \end{cases}$$

- per ogni nodo visitato si aggiunge ad un **bitvector B** (un vettore costruito su alfabeto $\Sigma = \{0, 1\}$) i simboli della sequenza $1^d 0$, con d grado del nodo
- si può risalire all'albero associato senza ambiguità (la *super-root* garantisce almeno un 1 per ogni nodo)



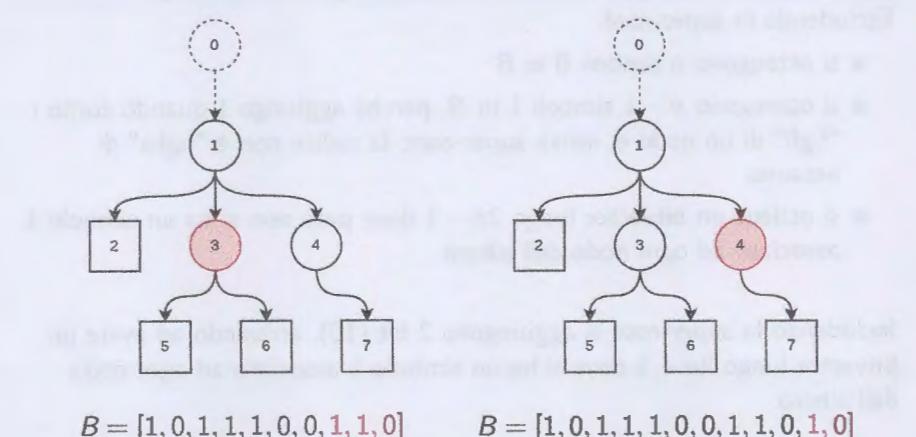
7 / 19



$B = [1, 0]$

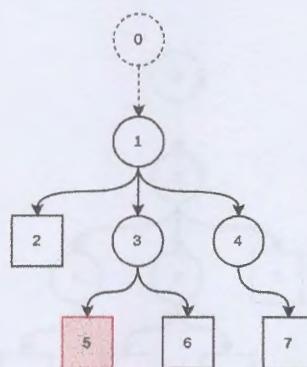
$B = [1, 0, 1, 1, 1, 0]$

$B = [1, 0, 1, 1, 1, 0, 0]$

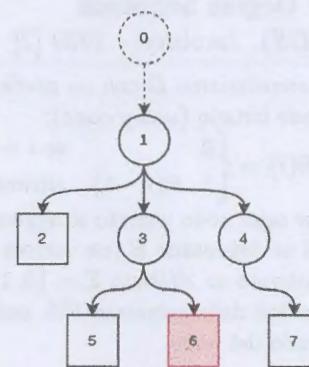


$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0]$

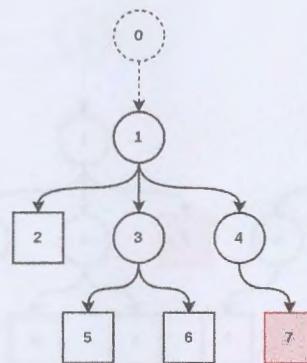
$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0]$



$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, \textcolor{brown}{0}]$$



$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, \textcolor{brown}{0}, 0]$$



$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$

LOUDS

Bitvector B

Escludendo la *super-root*:

- si ottengono n simboli 0 in B
- si ottengono $n - 1$ simboli 1 in B , perché aggiungo 1 quando conto i "figli" di un nodo e, senza *super-root*, la radice non è "figlia" di nessuno
- si ottiene un bitvector lungo $2n - 1$ dove però non si ha un simbolo 1 associato ad ogni nodo dell'albero

Includendo la *super-root* si aggiungono 2 bit (10), arrivando ad avere un bitvector lungo $2n + 1$ dove si ha un simbolo 1 associato ad ogni nodo dell'albero

Con questa rappresentazione posso indicare il nodo m con l'indice relativo del corrispondente bit 1 (es. il nodo 3 corrisponde al terzo bit pari a 1).

LOUDS

Per poter capire come funzionano le operazioni base su questa rappresentazione **succinta** di un albero dobbiamo prima introdurre (*senza entrare nei dettagli*) il funzionamento dei **bitvector**.

Bitvector I

Definizione

Si definisce un **bitvector** B come un array di lunghezza n , popolato da elementi binari $[1, 3, 4]$. Formalmente, si ha:

$$B[i] \in \{0, 1\}, \forall i \text{ t.c. } 1 \leq i \leq n$$

In alternativa si potrebbe avere, come formalismo:

$$B[i] \in \{\perp, \top\}, \forall i \text{ t.c. } 1 \leq i \leq n$$

8 / 19

Bitvector I

Funzione rank

Dato un bitvector B , lungo n , e data una certa posizione i del bitvector, la funzione *rank* restituisce il numero di valori uguali a 1 presenti fino a quella data posizione (*nota: in alcune definizioni si esclude la posizione i*). Più formalmente, si ha che:

$$\text{rank}_B(i) = \sum_{k=1}^{k \leq i} B[k], \forall i \text{ t.c. } 1 \leq i \leq n$$

Funzione select

Dato un bitvector B , lungo n , e dato un valore intero i , la funzione *select* calcola l'indice dell' i -esimo valore pari a 1 nel bitvector B . Più formalmente, si ha che:

$$\text{select}_B(i) = \min\{j \leq n \mid \text{rank}_B(j) = i\}, \forall i \text{ t.c. } 1 < i \leq \text{rank}_B(n)$$

$$\text{rank}_B(\text{select}_B(i)) = i$$

Bitvector I

Esempio

Ipotizziamo di avere il seguente bitvector B , di lunghezza $n = 14$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	1	0	1	0	1	0	1	0	0	1	0

Si ha che, per esempio:

$$\text{rank}(7) = 3$$

$$\text{select}(5) = 10$$

È possibile ottenere una struttura dati succinta, usando $\mathcal{O}(n)$ bit aggiuntivi, che permetta di effettuare $\text{rank}(i)$ (si darà un'intuizione del metodo) e $\text{select}(i)$ in $\mathcal{O}(1)$ -time [4, 3, 5, 6, 7].

Bitvector II, funzione rank

Memorizzare tutti i valori $\text{rank}(i)$ necessiterebbe $\mathcal{O}(n \log n)$ bit (se $B[i] = 1, \forall i = 0..n - 1$)

- memorizziamo ogni ℓ -esimo valore $\text{rank}(i)$
- a query time scorriamo i restanti $\ell - 1$ bit
- salviamo questi valori in un vettore $\text{first } F[0..n/\ell]$ (/ indica qui la divisione intera) t.c:
 - $F[0] = 0$
 - $F[i/\ell] = \text{rank}(i)$ se $i \bmod \ell = 0$
- se $\ell = \left(\left\lceil \left(\frac{\log n}{2}\right)\right\rceil\right)^2$ si ha un'ordine di $\frac{n \log n}{\log^2 n} = \frac{n}{\log n}$ bit per l'array F
- otteniamo $\text{rank}(i) = F[i/\ell] + \mathcal{C}(B[\ell \cdot (i/\ell) + 1..i])$, con $\mathcal{C}(B')$ funzione che conta i simboli $\sigma = 1$ in B'
- con questa soluzione abbiamo $\text{rank}(i)$ in $\mathcal{O}(\log^2 n)$ -time

9 / 19

Bitvector II, funzione rank

Possiamo tenere in memoria più informazioni

- in ogni blocco di lunghezza ℓ indotto da F memorizziamo ogni k posizioni il numero di simboli $\sigma = 1$ a partire dall'inizio del blocco (escludo la posizione iniziale il cui valore rank è di fatto già salvata in F e quindi si "resetta" il conteggio)
- otteniamo un vettore $\text{second } S[0..\ell/k]$ con
 $S[i/k] = \text{rank}_{B[\ell \cdot (i/\ell) + 1..i]}(i - \ell \cdot (i/\ell) + 1)$ quando $i \bmod k = 0$
- questo richiede $\mathcal{O}\left(\left(\frac{n}{k}\right) \log \ell\right)$ bit aggiuntivi
- usando $k = \lceil \frac{\log n}{2} \rceil$ si ottiene uno spazio di $\mathcal{O}\left(\frac{n \log \log n}{\log n}\right)$ bit
- otteniamo $\text{rank}(i) = F[i/\ell] + S[i/k] + \mathcal{C}(B[k \cdot (i/k) + 1..i])$,
- con questa soluzione abbiamo $\text{rank}(i)$ in $\mathcal{O}(\log n)$ -time

Bitvector II, funzione rank

Four Russians technique per ottenere $rank(i)$ in $\mathcal{O}(1)$ -time [7]

- si osserva che ci sono meno di \sqrt{n} bitvector di lunghezza $k - 1 = \lceil \frac{\log n}{2} \rceil - 1 < \frac{\log n}{2}$
- salviamo una *lookup table* (che garantisce tempo di accesso in $\mathcal{O}(1)$ -time) *third T*, di dimensioni $2^{k-1} \times k - 1$
- *T* salva i valori $rank(i')$ per tutte le posizioni $k - 1 < \frac{\log n}{2}$ in tutte le possibili $2^{k-1} < \sqrt{n}$ configurazioni indotte dai blocchi definiti per *S*
- in altri termini salva tutti i possibili valori di rank (uno per ogni posizione) di tutte le possibili configurazioni che può avere un bitvector di lunghezza $k - 1$
- tali valori $rank(i')$ sono interi di $\log(k) \leq \log \log n$ bit, quindi *T* richiede $\mathcal{O}(\sqrt{n} \log n \log \log n)$ bit

Bitvector II, funzione rank

Riscriviamo $C(B[k \cdot (i/k) + 1..i])$

- si definisce $c_i = B[k \cdot (i/k) + 1..k \cdot (i/k + 1) - 1]$ come il bitvector di lunghezza $k - 1$ che "copre" il $(k + 1)$ -esimo blocco ($k + 1$ perché indicizziamo qui da 0)
- usando il modulo posso sapere la posizione relativa di i in questo bitvector e accedere alla lookup table dove la riga è indicizzata da c_i e la colonna dalla posizione relativa in $\mathcal{O}(1)$ - time

$$rank(B) = \begin{cases} F[i/l] & \text{se } i \bmod l = 0 \\ F[i/l] + S[i/k] & \text{se } i \bmod l \neq 0 \wedge i \bmod k = 0 \\ F[i/l] + S[i/k] + T[c_i][(i \bmod k) - 1] & \text{se } i \bmod l \neq 0 \wedge i \bmod k \neq 0 \end{cases}$$

Bitvector II, funzione rank

tempo

- random access a *F* in $\mathcal{O}(1)$ -time (array)
- random access a *S* in $\mathcal{O}(1)$ -time (array)
- random access a *T* in $\mathcal{O}(1)$ -time (lookup table)

Spazio aggiuntivo

- *F* richiede $\frac{n}{\log n}$ bit quindi $o(n)$ bit
- *S* richiede $\mathcal{O}\left(\frac{n \log \log n}{\log n}\right)$ bit quindi $o(n)$ bit
- *T* richiede $\mathcal{O}(\sqrt{n} \log n \log \log n)$ bit quindi $o(n)$ bit

Assumendo un modello RAM con $w = \Omega(\log n)$ come size di una word

- $rank(i)$ in $\mathcal{O}(1)$ -time
- $rank(i)$ usa $n + o(n)$ bit come spazio (n per *b* e $o(n)$ per *F*, *S* e *T*)
- costruzione della struttura in $\mathcal{O}(n)$ -time

Bitvector II, funzione rank

E $select(i)$?

Si può dimostrare che, con un procedimento abbastanza analogo (ma un po' più complesso) a quello visto per la funzione $rank$, è possibile costruire una struttura che necessita di $o(n)$ aggiuntivi e che permette di effettuare $select(i)$ in $\mathcal{O}(1)$ -time.

Si rimanda a [3] per una spiegazione esaustiva del metodo.

Bitvector III, esempio costruzione struttura rank

Toy example, si impone:

- $\ell = 9$
- $k = 3$

input

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	1	0	...

10 / 19

Bitvector III, esempio costruzione struttura rank

Toy example, si impone:

- $\ell = 9$
- $k = 3$

Costruisco F

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	1	0	...	
F	0	-	-	-	-	-	-	-	-	4	-	-	-	-	...	

Bitvector III, esempio costruzione struttura rank

Toy example, si impone:

- $\ell = 9$
- $k = 3$

Costruisco S

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	0	1	0	...
F	0	-	-	-	-	-	-	-	-	4	-	-	-	-	-	...
S	0	-	-	1	-	-	3	-	-	0	-	-	1	-	-	...

Bitvector III, esempio costruzione struttura rank

Toy example, si impone:

- $\ell = 9$
- $k = 3$

Costruisco T

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	0	1	0	...
F	0	-	-	-	-	-	-	-	-	4	-	-	-	-	-	...
S	0	-	-	1	-	-	3	-	-	0	-	-	1	-	-	...

T	0	1
00	0	0
01	0	1
10	1	1
11	1	2

Bitvector IV, esempio rank

Overview della struttura con $\ell = 9$ e $k = 3$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	0	1	0	...
F	0	-	-	-	-	-	-	-	-	4	-	-	-	-	-	...
S	0	-	-	1	-	-	3	-	-	0	-	-	1	-	-	...

F, S e T

$$F = [0, 4, \dots]$$

$$S = [0, 1, 3, 0, 1, \dots]$$

$rank(i)$ con $i = 9$

T	0	1
00	0	0
01	0	1
10	1	1
11	1	2

- $i \bmod \ell = 9 \bmod 9 = 0$

$$\text{■ } rank(9) = F[9/9] = F[1] = 4$$

Bitvector IV, esempio rank

Overview della struttura con $\ell = 9$ e $k = 3$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	1	0	...	
F	0	-	-	-	-	-	-	-	-	4	-	-	-	-	...	
S	0	-	-	1	-	-	3	-	-	0	-	-	1	-	...	

F, S e T

$$F = [0, 4, \dots]$$

$$S = [0, 1, 3, 0, 1, \dots]$$

T	0	1
00	0	0
01	0	1
10	1	1
11	1	2

rank(i) con $i = 6$

■ $i \bmod \ell = 6 \bmod 9 = 6 \neq 0$

■ $i \bmod k = 6 \bmod 3 = 0$

■ $\text{rank}(6) = F[6/9] + S[6/3] = F[0] + S[2] = 0 + 3 = 3$

Bitvector IV, esempio rank

Overview della struttura con $\ell = 9$ e $k = 3$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
B	1	0	0	1	0	1	0	1	0	1	0	0	1	0	...	
F	0	-	-	-	-	-	-	-	-	4	-	-	-	-	...	
S	0	-	-	1	-	-	3	-	-	0	-	-	1	-	...	

F, S e T

$$F = [0, 4, \dots]$$

$$S = [0, 1, 3, 0, 1, \dots]$$

T	0	1
00	0	0
01	0	1
10	1	1
11	1	2

rank(i) con $i = 11$

■ $i \bmod \ell = 11 \bmod 9 = 2 \neq 0$

■ $i \bmod k = 11 \bmod 3 = 2 \neq 0$

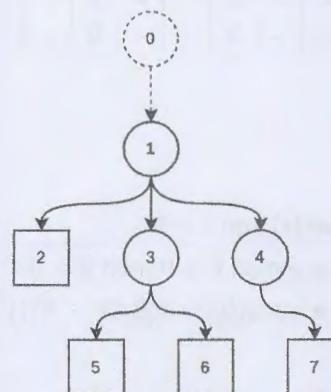
■ $c_{11} = B[3 \cdot (11/3) + 1..3 \cdot (11/3 + 1) - 1] = B[3 \cdot 3 + 1..3 \cdot 4 - 1] = B[10..11] = 10$

■ $\text{rank}(11) = F[11/9] + S[11/3] + T[10][(11 \bmod 2) - 1] = F[1] + S[3] + T[10][1] = 4 + 0 + 1 = 5$

LOUDS... parte 2

Bitvector B per la rappresentazione LOUDS di un albero

$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$



LOUDS... parte 2

Operazioni

- $\text{rank}_\sigma(i)$ che ritorna il numero di occorrenze di un simbolo $\sigma \in \{0, 1\}$. Per come abbiamo descritto $\text{rank}(i)$ ipotizziamo di costruire un secondo bitvector ottenuto col complemento bitwise di ogni valore di B , in modo da effettuare la $\text{rank}(i)$ sui simboli $\sigma = 0$ di B
- $\text{select}_\sigma(i)$ che ritorna la posizione dell' i -esimo simbolo $\sigma \in \{0, 1\}$. Per come abbiamo descritto $\text{select}(i)$ ipotizziamo di costruire un secondo bitvector ottenuto col complemento bitwise di ogni valore di B , in modo da effettuare la $\text{select}(i)$ sui simboli $\sigma = 0$ di B
- $\text{rank}_0(i) = i - \text{rank}_1(i)$

LOUDS... parte 2

Operazioni

- $\text{is_leaf}(v) = \top$ sse $\text{select}_0(v) = \text{select}_0(v+1) - 1$ in quanto per costruzione una foglia aggiunge solo uno 0 al bitvector B , quindi con $\text{select}_0(v)$ troviamo la posizione dello 0 posto in B dal nodo antecedente a v nella visita (quello antecedente perché abbiamo il 10 della *super-root*) e con $\text{select}_0(v+1)$ la posizione dello 0 relativo a v . Se sono in due posizioni adiacenti significa che v ha aggiunto solo uno 0 e quindi è una foglia

- $\text{is_leaf}(2) = \top \Leftarrow 6 = \text{select}_0(2) = \text{select}_0(2) - 1 = 7 - 1 = 6$
- $\text{is_leaf}(6) = \top \Leftarrow 13 = \text{select}_0(6) = \text{select}_0(7) - 1 = 14 - 1 = 13$
- $\text{is_leaf}(4) = \perp \Leftarrow 10 = \text{select}_0(4) \neq \text{select}_0(5) - 1 = 12 - 1 = 11$

LOUDS... parte 2

Operazioni

- $\text{first_child}(v) = \text{rank}_1(\text{select}_0(\text{rank}_1(\text{select}_1(v))) + 1) = \text{rank}_1(\text{select}_0(v) + 1)$:
 - $k = \text{select}_0(v) + 1$ restituisce la posizione k su B del primo "figlio" di v . In altri termini il v -esimo 0 mi dice che ho finito di "visitare" la sottosequenza di bitvector costruita per il nodo $v - 1$ e al bit successivo inizia la sequenza del bitvector per v
 - $m = \text{rank}_1(k)$ restituisce il numero di nodo dell'albero in posizione k di B , quindi la label del primo "figlio" di v
 - imponiamo che $\text{first_child}(v) = -1$ se $\text{is_leaf}(v) = \top$

$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$

- $\text{first_child}(1) = \text{rank}_1(\text{select}_0(1) + 1) = \text{rank}_1(2 + 1) = 2$
- $\text{first_child}(3) = \text{rank}_1(\text{select}_0(3) + 1) = \text{rank}_1(7 + 1) = 5$
- $\text{first_child}(4) = \text{rank}_1(\text{select}_0(4) + 1) = \text{rank}_1(10 + 1) = 7$

LOUDS... parte 2

Operazioni

- $\text{last_child}(v) = \text{rank}_1(\text{select}_0(\text{rank}_1(\text{select}_1(v)) + 1) - 1) = \text{rank}_1(\text{select}_0(v+1) - 1)$:
- $k = \text{select}_0(v+1)$ restituisce la posizione k su B dello 0 inserito in visita level-order del nodo con label v . In altri termini, il $(v+1)$ -esimo 0 mi dice che ho finito di "visitare" la sottosequenza di bitvector costruita per il nodo v
- $w = k - 1$ restituisce l'indice dell'ultimo 1 inserito in visita level-order del nodo con label j , quindi l'indice su B dell'ultimo "figlio" di c . In altri termini, con la precedente operazione si raggiunge lo 0 di $1^d 0$ e col -1 l'ultimo 1 di 1^d
- $m = \text{rank}_1(w)$ restituisce il numero di nodo dell'albero in posizione w di B , quindi la label dell'ultimo "figlio" di v
- **imponiamo che $\text{last_child}(v) = -1$ se $\text{is_leaf}(v) = \top$**

$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$

- $\text{last_child}(1) = \text{rank}_1(\text{select}_0(1+1) - 1) = \text{rank}_1(6-1) = 4$
- $\text{last_child}(3) = \text{rank}_1(\text{select}_0(3+1) - 1) = \text{rank}_1(10-1) = 6$
- $\text{last_child}(4) = \text{rank}_1(\text{select}_0(4+1) - 1) = \text{rank}_1(12-1) = 7$

LOUDS... parte 2

Operazioni

- $\text{parent}(v) = \text{rank}_1(\text{select}_1(\text{rank}_0(\text{select}_1(v)))) = \text{rank}_0(\text{select}_1(v))$:
- $i = \text{select}_1(v)$ restituisce la posizione i del nodo v nel bitvector B (identificando in quale $1^d 0$ è stato aggiunto)
- $j = \text{rank}_0(i)$ restituisce il numero di sequenze che sono state aggiunte al bitvector B fino a quella relativa al "genitore" del nodo in posizione i . Il numero di tali sequenze è l'indice del nodo "genitore" per definizione di vista level order e conseguente etichettatura dei nodi
- **imponiamo che $\text{parent}(v) = -1$ se $v = 1$ (non considero la super-root)**

$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$

- $\text{parent}(2) = \text{rank}_0(\text{select}_1(2)) = \text{rank}_0(2) = 1$
- $\text{parent}(6) = \text{rank}_0(\text{select}_1(6)) = \text{rank}_0(9) = 3$
- $\text{parent}(7) = \text{rank}_0(\text{select}_1(7)) = \text{rank}_0(11) = 4$

LOUDS... parte 2

Operazioni

- $\text{degree}(v) = \text{last_child}(v) - \text{first_child}(v) + 1$,
- **imponiamo $\text{degree}(v) = 0$ se $\text{last_child}(v) = \text{first_child}(v) = -1$**
- $\text{nth_child}(v, nth) = \text{rank}_1(\text{select}_1(\text{first_child}(v)) + nth - 1)$,
- **imponiamo $\text{nth_child}(v, nth) = -1$ se $\text{degree}(v) < nth$**

$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$

- $\text{degree}(1) = \text{last_child}(1) - \text{first_child}(1) + 1 = 4 - 2 + 1 = 3$
- $\text{degree}(4) = \text{last_child}(4) - \text{first_child}(7) + 1 = 7 - 7 + 1 = 1$

$$B = [1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]$$

- $\text{nth_child}(1, 2) = \text{rank}_1(\text{select}_1(\text{first_child}(1)) + 2 - 1) = \text{rank}_1(\text{select}_1(2) + 2 - 1) == \text{rank}_1(3 + 2 - 1) = \text{rank}_1(4) = 3$
- $\text{nth_child}(3, 2) = \text{rank}_1(\text{select}_1(\text{first_child}(3)) + 2 - 1) = \text{rank}_1(\text{select}_1(5) + 2 - 1) == \text{rank}_1(8 + 2 - 1) = \text{rank}_1(9) = 6$

VOGLIAMO GENERALIZZARE RANK SELECT SLIDE 13 (1/2)

 $\begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \text{ACC} & G & T & C & C & A & T \end{matrix}$

Ci interessano lezioni del NPO che è la seconda T

possiamo usare un bitvector per ogni carattere

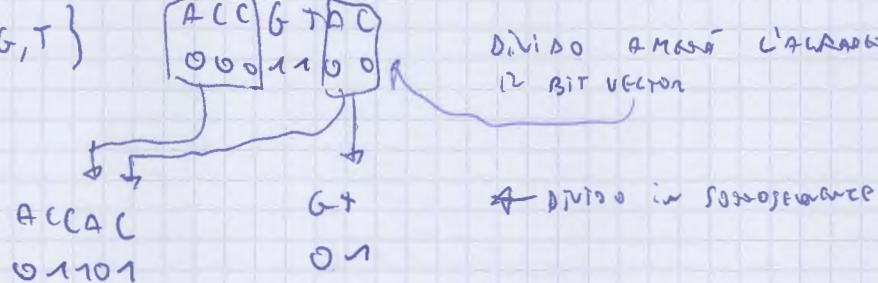


$B_c = 011001100$

 $S = \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \text{ACC} & G & T & C & C & A & T \end{matrix}$

— SLIDE 13 (3)

$$\Sigma = \{A, C | G, T\}$$

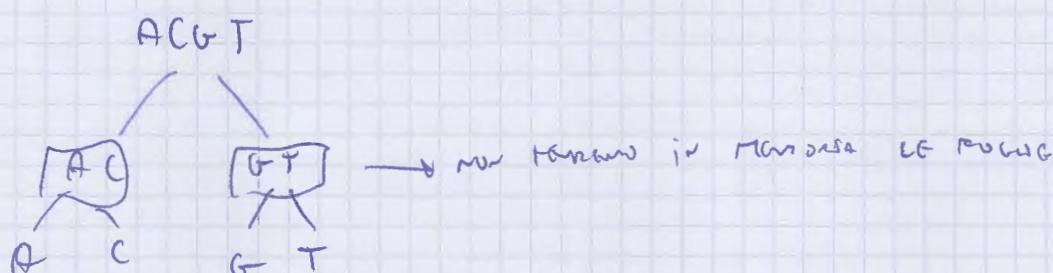


diviso ampiamente e ricci
12 bit vector

→ diviso in sottovettori

$$\Sigma_1 = \{A, C\} \quad \Sigma_{11} = \{G, T\} \quad \text{→ sotto-alfabeti}$$

servono solo pochi punti per il rango ma non siano



— SLIDE 13 (4)

 $\begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ \text{ACC} & G & T & C \end{matrix}$

bitvector → 000110

RANK SOLO BSI



4° campo della sottovettore del blocco di 6 bit

 $\begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ \text{ACC} & G & T & C \end{matrix}$

0111 → RANK 3 non sovrapponeccc ed è quindi il terzo C

In EJERCICIO HABEMOS QUE USAR UN ALGORITMO DE KMP, CON UN ESPACIO
MUY LIMITADO EN MEMORIA

SWAP 13

SEQUENCIA T, $\Sigma = \{A, C, G, T\}$, $n = |T| = 16$, $s = |\Sigma| = 4$

A C C G T A A G T T T T A G C T T

INGRESO SÓLO DE LOS NUEVOS SÍMBOLOS EN MEMORIA

CLASIFICA RANK DE LA S

NÚMERO 15 EN POSICIÓN 6 ROBA EXPRESA

Wavelet Tree

Una generalizzazione

Abbiamo visto come *rank* e *select* possono essere usati per interrogare un bitvector, che ha un alfabeto fisso di grandezza 2. Siamo ora interessati a generalizzare tali query ad un alfabeto di grandezza arbitraria. Per praticità assumiamo $\Sigma[1..s]$ ordinato: $\Sigma[i] \prec \Sigma[j] \iff i < j..$

Dato un generico alfabeto Σ e una sequenza $T[1..n] \in \Sigma^*$

- $rank_{\sigma, T}(i)$ conta tutte le occorrenze del simbolo $\sigma \in \Sigma$ fino all'indice i in T , $i \leq |T|$
- $select_{\sigma, T}(i)$ ritorna la posizione dell' i -esima occorrenza del simbolo $\sigma \in \Sigma$ in T , $i \leq |T|$
- $rank_{\sigma, T}(select_{\sigma, T}(i)) = i$, $\forall \sigma \in \Sigma \wedge \forall i = 1..n$
- se T è inferibile dal contesto scriviamo $rank_\sigma(i)$ e $select_\sigma(i)$

Quando con la professoressa Rizzi parlerete di Burrows-Wheeler Transform, FM-index e LF-function ricordate i wavelet tree.

13 / 19

Wavelet Tree

Una rappresentazione "naïve"

- considero come rappresentazione di T un insieme di $|\Sigma| = s$ stringhe binarie $B_\sigma[1..n]$, $\forall \sigma \in \Sigma$
- si ha che $B_\sigma[i] = 1 \iff T[i] = \sigma$ e $B_\sigma[i] = 0 \iff T[i] \neq \sigma$

Si ottiene

- $rank_{\sigma, T}(i) = rank_{1, B_\sigma}(i)$
- se per ogni bitvector B_σ abbiamo calcolato la struttura a supporto vista precedentemente si ha $rank_{\sigma, T}(i)$ in:
 - $\mathcal{O}(1)$ -time
 - $s \cdot (n + o(n))$ bit aggiuntivi

Possiamo ottenere una rappresentazione più efficiente in memoria senza "sacrificare troppo" i tempi di query?

Wavelet Tree

Wavelet tree [3, 4, 8, 9, 10]

- consideriamo un **albero binario perfettamente bilanciato** dove ogni nodo corrisponde ad un sottoinsieme di Σ
- i due "figli" di ogni nodo partizionano il corrispondente sottoinsieme di Σ in due
- a ogni nodo v corrisponde una sequenza R_v , che è sottosequenza di T , che è la sottosequenza della sequenza che etichetta il nodo "genitore" di v corrispondente al sottoinsieme di Σ associato a v . Alla *root* corrisponde la sequenza $R_v = T$
- ogni nodo v si associa un bitvector B_v che indica a quale dei due "figli" di v ogni simbolo di R_v appartiene (ad esempio con $1 \leq j \leq |R_v|$ se $B_v[j] = 0$ allora $R_v[j]$ appartiene al "figlio di sinistra" mentre se $B_v[j] = 1$ allora $R_v[j]$ appartiene al "figlio di destra"). A breve si vedrà meglio come determinare il "figlio"
- le foglie dell'albero sono "virtualmente" etichettate con i singoli caratteri dell'alfabeto (uno per foglia) ma ci basta la funzione $rank_1$ (che può esprimere anche $rank_0$) sui bitvector che etichettano i "genitori" delle foglie per recuperare l'informazione

Wavelet Tree

Calcolo di $rank_{\sigma, T}(i)$

- si inizia il calcolo nel nodo *root* e si determina a quale dei due figli appartiene σ , tramite l'ordinamento dell'alfabeto. Nella radice $B_v[j] = 0$ se $R_v[j] = \Sigma [\lceil \frac{s}{2} \rceil] \vee R_v[j] \prec \Sigma [\lceil \frac{s}{2} \rceil]$ e $B_v[j] = 1$ altrimenti:
 - se $\sigma = \Sigma [\lceil \frac{s}{2} \rceil] \vee \sigma \prec \Sigma [\lceil \frac{s}{2} \rceil]$ allora si prosegue verso il "figlio" di sinistra, aggiornando i con $i = rank_{0, B_v}(i)$
 - altrimenti usiamo il "figlio" di destra, aggiornando i con $i = rank_{1, B_v}(i)$
- nel nuovo v si procede nello stesso modo, considerando che ora $\Sigma = \Sigma [1.. \lceil \frac{s}{2} \rceil]$ se si è andati a sinistra e $\Sigma = \Sigma [\lceil \frac{s}{2} \rceil + 1..s]$ se a destra
- si prosegue fino ad una foglia e $rank_{\sigma, T}(i) = i'$, dove i' è l'ultimo valore di i che si ottiene nei vari step

l'albero ha altezza $\lceil \log s \rceil$ quindi $rank_{\sigma, T}(i)$ può essere calcolato in $\mathcal{O}(\log s)$ -time

Wavelet Tree

Random access: accesso a $T[i]$, $1 \leq i \leq |T|$.

Calcolo di $access_{\sigma, T}(i)$

- in questo caso la scelta del figlio dipende unicamente da $B_v[i]$:
 - se $B_v[i] = 0$ prosegui verso il "figlio" di sinistra, con $i = rank_{0, B_v}(i)$
 - se $B_v[i] = 1$ prosegui verso il "figlio" di destra, con $i = rank_{1, B_v}(i)$
- $T[i]$ è il simbolo che etichetta la foglia raggiunta alla fine della visita
- dato che il wavelet tree di una sequenza T garantisce random access alla sequenza stessa possiamo sostituirla col suo wavelet tree

l'albero ha altezza $\lceil \log s \rceil$ quindi $access_{\sigma, T}(i)$ può essere calcolato in $\mathcal{O}(\log s)$ -time

Analogamente ai bitvector si può dimostrare che anche $select_{\sigma, T}(i)$ può essere calcolato in $\mathcal{O}(\log s)$ -time

Wavelet Tree

Costruzione wavelet tree

- si può costruire un wavelet tree livello per livello a partire dalla *root*:
 - se etichetta la root con $R_v = T$ e un bitvector B_v tale che $\forall 1 \leq i \leq |T|$:
 - ▶ $B_v[i] = 0 \iff T[i] = \Sigma [\lceil \frac{s}{2} \rceil] \vee T[i] \prec \Sigma [\lceil \frac{s}{2} \rceil]$
 - ▶ $B_v[i] = 1$ altrimenti
 - si estrae la sottosequenza T' corrispondente ai valori 0 di B_v e la si usa per etichettare il "figlio" di sinistra v_1 (che è relativo all'alfabeto $\Sigma = \Sigma [1.. \lceil \frac{s}{2} \rceil]$) mentre quella corrispondente ai valori 1 la si usa per etichettare il "figlio" di destra v_2 (che è relativo all'alfabeto $\Sigma = \Sigma [\lceil \frac{s}{2} \rceil + 1..s]$) per entrambe
 - posso quindi cancellare T e costruire i bitvector B_{v_1} e B_{v_2} , con le rispettive strutture per la funzione *rank* e continuare ricorsivamente fino al raggiungimento delle foglie (quando si raggiungono alfabeti di cardinalità 1)

La costruzione di un wavelet tree richiede $\mathcal{O}(n \log s)$ -time

Wavelet Tree

Spazio richiesto

- in ogni momento della costruzione del wavelet tree abbiamo un bound in spazio pari a $3n \log s + \mathcal{O}(s \log n)$, dato da:
 - 3 sottosequenze di T , quella del "genitore" e quelle dei due "figli"
 - tutti i bitvector finora computati che formano il wavelet tree
 - i puntatori che memorizzano la struttura ad albero
- un ulteriore miglioramento in spazio si può ottenere concatenando tutti i bitvector in un unico bitvector con una sola struttura a supporto della funzione *rank*. In tal caso la struttura ad albero si può inferire dai partizionamenti dell'alfabeto e dai bitvector. Questa variante è chiamata **levelwise wavelet tree** [11, 12] (esempio più avanti).

Un wavelet tree per una sequenza lunga n costruita su alfabeto di cardinalità s occupa, avendo $\mathcal{O}(s \log n)$ per la topologia dell'albero:

$$n \log s + o(n \log s) + \mathcal{O}(s \log n) \text{ bit}$$

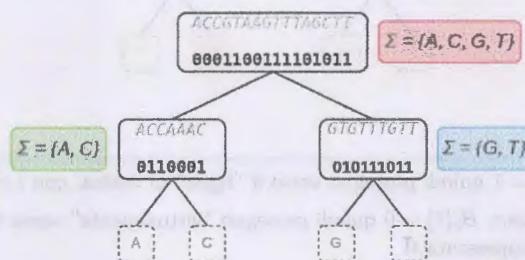
Un levelwise wavelet tree richiede:

$$n \log s + o(n \log s) \text{ bit}$$

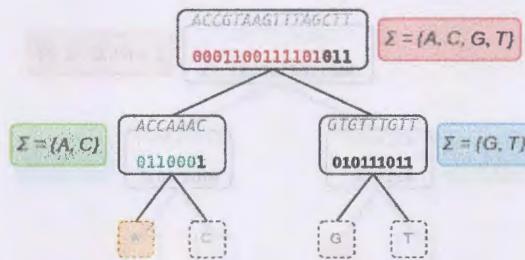
Wavelet Tree

Sequenza $T, \Sigma = \{A, C, G, T\}, n = |T| = 16, s = |\Sigma| = 4$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A C C G T A A G T T T T A G C T T



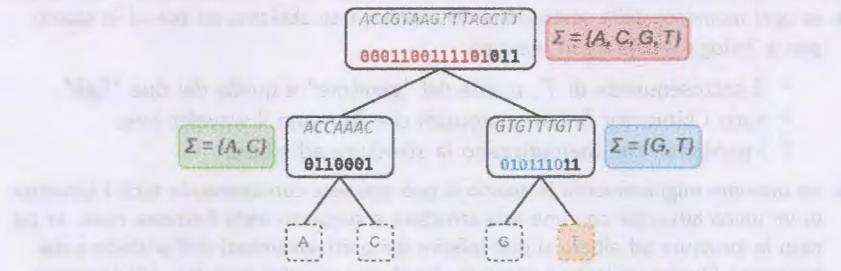
Wavelet Tree



$rank_A(13)$

- *root*: "A" è nella prima metà dell'alfabeto $\Sigma = \{A, C, G, T\}$ quindi, nella radice: $i = rank_0(13) = 6$
- "figlio" di sinistra: "A" è nella prima metà dell'alfabeto $\Sigma = \{A, C\}$ quindi, nella "figlio" di sinistra: $i = rank_0(6) = 4$
- *foglia*: $rank_A(13) = 4$

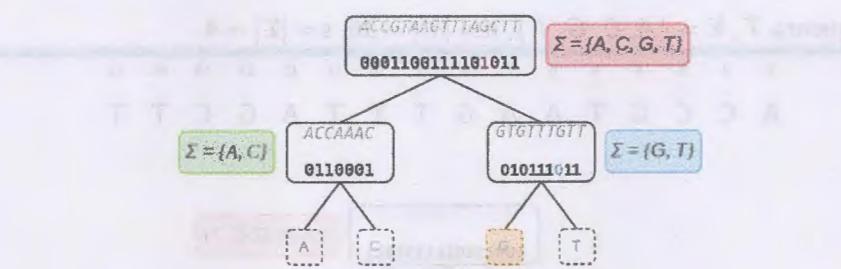
Wavelet Tree



$rank_T(13)$

- *root*: "T" è nella seconda metà dell'alfabeto $\Sigma = \{A, C, G, T\}$ quindi, nella radice: $i = rank_1(13) = 7$
- "figlio" di sinistra: "T" è nella seconda metà dell'alfabeto $\Sigma = \{G, T\}$ quindi, "figlio" di destra: $i = rank_1(7) = 4$
- *foglia*: $rank_T(13) = 4$

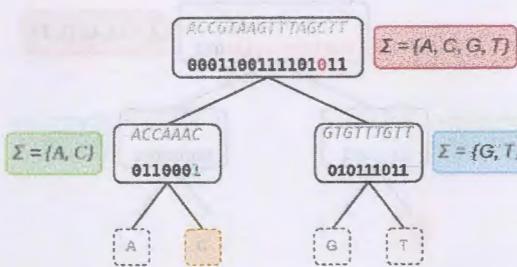
Wavelet Tree



$access(13)$

- *root*: $B_v[13] = 1$ quindi proseguo verso il "figlio" di destra, con $i = rank_1(13) = 7$
- "figlio" di destra: $B_v[7] = 0$ quindi proseguo "virtualmente" verso la foglia di sinistra che rappresenta G
- *foglia*: $access(13) = G$

Wavelet Tree



$access(14)$

- *root*: $B_v[14] = 0$ quindi proseguo verso il "figlio" di sinistra, con $i = rank_0(14) = 7$
- "figlio" di sinistra: $B_v[7] = 1$ quindi proseguo "virtualmente" verso la foglia di destra che rappresenta C
- *foglia*: $access(14) = C$

Altre strutture dati succinte

~~BIT VECTOR ...~~
~~WAVELET TREE ...~~

Un elenco non esaustivo

- parentesi bilanciate [13, 2, 14, 15]
- strutture dati succinte che supportano le **range minimum queries** [16]
- **wavelet matrix** [17]
- etc...

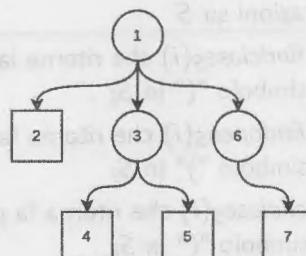
14 / 19

Parentesi bilanciate

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi

Rappresentazione "raw"

- una rappresentazione a pointer richiede $\mathcal{O}(n \log n)$ -space
- infatti servono $\mathcal{O}(nw)$ bit con $w \geq \log_2 n$ dimensione in bit di un pointer



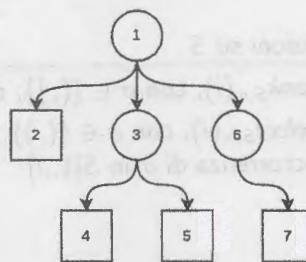
15 / 19

Parentesi bilanciate

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi

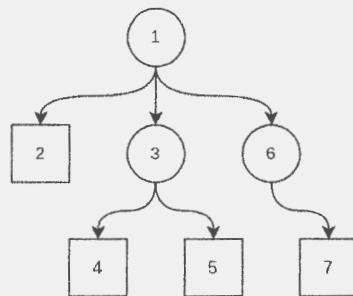
Rappresentazione a **parentesi bilanciate**, proposta da Jacobson nel 1989 [2]

- rappresentazione tramite parentesi bilanciate
- si costruisce a partire dalla DFS dell'albero (preorder):
 - "(" quando si raggiunge un nodo per la prima volta
 - ")" quando si è terminata la visita del sottoalbero



Parentesi bilanciate

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi



S	\parallel	(()		(()		()		())				
n	\parallel	1		2		2		3		4		4		5		5		3		6		7		6		1

Parentesi bilanciate

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi

Operazioni su S

- $findclose_S(i)$ che ritorna la posizione del simbolo ")" corrispondente al simbolo "(" in S_i ;
- $findopen_S(i)$ che ritorna la posizione del simbolo "(" corrispondente al simbolo ")" in S_i ;
- $enclose_S(i)$ che ritorna la posizione del simbolo "(" che racchiude il simbolo "(" in S_i ;

Parentesi bilanciate

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi

Operazioni su S

- $rank_{S,\sigma}(i)$, con $\sigma \in \{(), ()\}$, che ritorna la frequenza di σ in $S[1..i]$
- $select_{S,\sigma}(i)$, con $\sigma \in \{(), ()\}$, che ritorna la posizione dell' i -esima occorrenza di σ in $S[1..i]$

Parentesi bilanciate

Consideriamo la rappresentazione in memoria di un albero etichettato con n nodi

Operazioni su S

- $\text{parent}(v) = \text{encloses}_S(v)$ che restituisce il "genitore" del nodo v
- $\text{first_child}(v) = v + 1$, per costruzione di S
- $\text{sibling}(v) = \text{findcloses}_S(v) + 1$, infatti il "fratello" si ha subito dopo la ")" relativa a v
- $\text{last_child} = \text{findopens}_S(\text{findcloses}_S(v) - 1)$, avendo che $\text{findcloses}_S(v) - 1$ restituisce la ")" dell'ultimo "figlio" di v
- $\text{subtree_size}(v) = \frac{\text{findcloses}_S(v) - v + 1}{2}$, perché trevo il numero di simboli su S che rappresentano il sottoalbero (entrambe le parentesi quindi divido per 2)
- depth di un nodo, *lowest common ancestor* tra due nodi, etc...

Parentesi bilanciate

Si può dimostrare che

$\text{findcloses}(i)$, $\text{findopens}(i)$, $\text{encloses}(i)$ possono essere calcolate in $O(\log n)$ -time con $o(n)$ bit aggiuntivi [13, 14, 15].

Range Minimum Queries

RICERCA DEL VALORE MINIMO IN UN INTERVALLO SPECIFICO, NELLA
SEQUENZA DI NUMERI A

Definizione

Dato un array $A[1..n]$ di numeri n elementi da un universo totalmente ordinato la **Range Minimum Query** $\text{RMQ}_A(i,j)$, con $1 \leq i \leq j \leq n$, restituisce la posizione k di un elemento minimo in $A[i..j]$:

$$\text{RMQ}_A(i,j) = \underset{i \leq k \leq j}{\operatorname{argmin}} \{A[k]\}$$

Le RMQ vengono usate i vari ambiti tra cui l'indicizzazione di testi, pattern matching, la compressione e il retrieval di documenti etc..

Si può dimostrare che

Si può costruire una struttura dati succinta che richiede $2n + o(n)$ bit in memoria e che risponde in $O(1)$ -time [16].

Wavelet matrix

Motivazione

Le **wavelet matrix** nascono con l'idea di migliorare i *levelwise wavelet tree* nella gestione di larghi alfabeti:

- tempi di access dimezzati
- tempi di *rank* e *select* leggermente ridotti

Costruzione

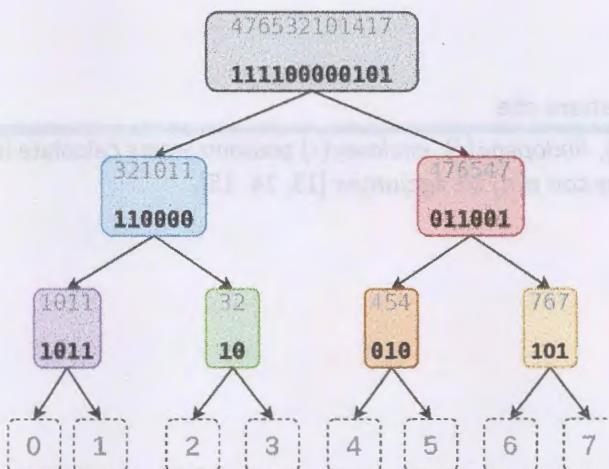
L'idea è che i bit di un nodo "figlio" non sono più "allineati" al "genitore" ma si assume che passando da un livello all'altro, **tutti** gli zero vanno da una parte e gli uni dall'altra. Salvando ad ogni livello ℓ il numero totale di simboli $\sigma = 0 z_\ell$, richiedendo in totale $\mathcal{O}(\log n \log s)$ bit, si ottiene lo stesso comportamento di un levelwise wavelet tree [17].

Una wavelet matrix richiede $n \log s + o(n \log s)$ bit, può essere costruita in $\mathcal{O}(n \log s)$ e risponde alle stesse query di un (levelwise) wavelet tree in $\mathcal{O}(\log s)$

17 / 19

Wavelet matrix

Un esempio con $T = 476532101417$, avendo $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7\}$ [17]



Wavelet matrix

Un esempio con $T = 476532101417$, avendo $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7\}$ [17]

4	7	6	5	3	2	1	0	1	4	1	7
1	1	1	1	0	0	0	0	0	1	0	1
3	2	1	0	1	1	4	7	6	5	4	7
1	1	0	0	0	0	0	1	1	0	0	1
1	0	1	1	3	2	4	5	4	7	6	7
1	0	1	1	1	0	0	1	0	1	0	1
0	1	1	1	2	3	4	4	5	6	7	7

Wavelet matrix

Un esempio con $T = 476532101417$, avendo $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7\}$ [17]

4	7	6	5	3	2	1	0	1	4	1	7	
1	1	1	1	0	0	0	0	0	1	0	1	
3	2	1	0	1	1	4 7 6 5 4 7						
1	1	0	0	0	0	0 1 1 0 0 1						
1	0	1	1	4	5	4	3 2 7 6 7					
1	0	1	1	0	1	0	1 0 1 0 1					
0	4	4	2	6	1	1	1	5	3	7	7	

Le linee verticali sono una per livello e rappresentano i valori z_ℓ .

Conclusioni

Per concludere:

- abbiamo "grattato la superficie" del mondo delle strutture dati succinte
- abbiamo scoperto i **bitvector succinti** la principale di esse, alla base di tutte le altre
- abbiamo introdotto una loro estensione ad alfabeto arbitrario e un loro uso per rappresentare *alberi*
- abbiamo introdotto alcune altre strutture dati succinte che potrete approfondire autonomamente

~~STRUTTURA AD ALBERO D'AVVISTAMENTO, CASO DI~~

SLOG 3

Se T è un albero binario c'è qualcosa, nessuno aggiunge

nil / nil / nil / vvvvvv / nil / valore / nil / nil

c'è tanto spazio

SLOG 4

per risolvendo le collisioni posso, aggiungere più funzioni di HASH

Esempio: funzione di HASH? Massimo di piano X se ha lo stesso output al X

Se trovo x
vado qua
Se trovo y
vado qua
...

→	$h(x)$	$h(x')$	—	—
→	$h(y)$	$h(w)$	—	—
→	$h(u)$		—	—

Così ho i risultati al tempo scorso in questo modo

Risultato di questo è avere valori (in modo) in appena una riga

cerco scommesse in $\mathcal{O}(\alpha)$
la lista

SLOG 6

LG HASH
Sono usate
per strutturare
dati che permesso
operazioni:

INSERT
DELETE
SEARCH
COUNT
 \downarrow
 $x \in S ? \rightarrow$ MEMBERSHIP PROBLEM

INPUT \hookrightarrow è il sovraccarico di INPUT che vogliano

or. dizionario intero

HEAP ✓
TREAP
AVL
RAST

Falsi positivi: se $x \notin S$ la struttura dice che è membro, con una certa probabilità
Dentro

in $A[i]$ ci possono essere collisioni

per questo è necessaria

SLIDE 7

LA FUNZIONE DI HASH DOVEVAESSERE UNA FUNKTION

BLOOM FILTER

3 FUNZIONI DI HASH: h_1, h_2, h_3 l FUNZIONI

PISTONE DI
PARTECIPANTE
OSSIA UN
VALORE

$$\rightarrow S = [x_1, \dots, x_n]$$

USCISSE SARE PREPROCESSATO PIÙ S

COSTRUISSO UN BITVECTOR $A = [00000000]$

$$\text{ES. } A = \begin{bmatrix} & h_2(x_1) \\ 1 & 0 & 1 & 0 & 1 & 1 & \dots & 1 \end{bmatrix}$$

\downarrow \downarrow \downarrow \downarrow
 $h_1(x_1)$ $h_2(x_1)$ $h_3(x_1)$ $h_3(x_2)$
 $h_3(x_1)$

QUINDI SE VOGLIO VERIFICARE SE x CAPOGGIA LA FUNZIONE DI HASH

SE UNO È 0 ALLORA SI CONSIDERA NON APPARTENENTE

SE TUTTI SONO 1 ALLORA PUÒ APPARTENERE, MA STANCIAMENTE POSSIBILE
C'È UNA LEZIONE SU QUESTO

ALLA FINE AVREMO AL MASSIMO IL BIT A 1

IL BLOOM FILTER È UNO VETTORE

IN POCHE PAROLE COSE IN BLOOM FILTER

PERMETTE DI FAIRE PRESEZIONI CHE USOGLIANO SOLO SKIPPING

SLIDE 8 ABBIANO IL PROBLEMA CHE I BLOOM FILTER SOTTOVENDO

IN MEDIO MOSTRA L'AUMENTO DEL RAZZO POIJNVI ALL'NUMERO DI CELLE
FUNZIONE DI HASH

Strutture Dati Probabilistiche Hashing-Based

Outline

1 Introduzione hashing

2 Bloom filter

3 Count-min sketch

4 Conclusioni

Ripasso...

Indirizzamento diretto

- usa una struttura dati semplice (es. array, matrici etc...) T
- insieme universo \mathcal{U} , $|\mathcal{U}| = u$, "piccolo"
- ogni posizione di T è una chiave dell'universo \mathcal{U} , ogni posizione ha poi i puntatori ai vari elementi (o direttamente l'elemento)
- se per una chiave $k \in \mathcal{U}$ non ci sono elementi: $T[k] = \text{nil}$
- accesso, selezione di un elemento e inserimento di un elemento in $\mathcal{O}(1)$ -time, data la chiave k , l'input x e la posizione $k[x]$ (o la posizione è direttamente x)

Problemi

- se universo \mathcal{U} è "grande" si possono avere problemi di memoria a memorizzare T
- dato l'insieme $K \subseteq \mathcal{U}$ di chiavi utilizzate ($T[k] \neq \text{nil}$) si ha $|K| \ll |\mathcal{U}|$

Soluzione: hash tables

- si richiede $\Theta(|K|)$ -space
- operazioni in $\Theta(1)$ -time

Funzioni di hash

Se con l'accesso diretto l'elemento con chiave k era memorizzato in posizione k , con l'**hash** è memorizzato in $h(k)$ [1]

Funzione di hash [1]

- **h funzione di hash** t.c: $h : \mathcal{U} \rightarrow \{0, \dots, m-1\}$
- dominio: insieme universo \mathcal{U} , $|\mathcal{U}| = u$
- codominio: insieme di posizioni $\{0, \dots, m-1\}$ di una tabella hash T
- la tabella di hash è indicizzata $T[0..m-1]$
- si assume che $h(k)$ è computato in $\mathcal{O}(1)$ -time

Problemi

- le funzioni di hash possono avere input "scomodi". Si possono avere **collisioni**: $h(k') = h(k'')$, $k' \neq k''$

Funzioni di hash

Famiglia di funzioni hash

- una **famiglia di funzioni hash** è un insieme \mathcal{H} di funzioni hash con lo stesso dominio e codominio
- la scelta di $h \in \mathcal{H}$ può essere fatta con un sampling uniforme su \mathcal{H}
- riduco le possibilità di collisione
- \mathcal{H} è detta **universale** sse, con $h : \mathcal{U} \rightarrow \{0, \dots, m-1\}$ scelta casualmente da \mathcal{H} , $\mathcal{P}[h(x) = h(y)] \leq \frac{1}{m}$ (non si ha alta probabilità di collisioni)

Hash table con collisioni risolte per concatenazione [1]

- si mettono tutti gli elementi che collidono nella stessa posizione dell'hash table in una lista concatenata
- chiamiamo α il **fattore di carico**, ovvero il numero medio di elementi in queste liste concatenate
- se tutte le n chiavi "mappano" in una sola lista ho il caso peggiore: accesso in $\Theta(n)$ -time ma nella realtà è difficile che accada quindi accesso in $\Theta(1 + \alpha)$
- nel caso migliore, con il numero di posizioni nella hash table proporzionale al numero di elementi della tabella (quindi $\alpha \rightarrow 1$), ho accesso in tempo $\Theta(1)$

4

Membership problem

Vogliamo risolvere il problema detto **membership problem**

Ambiti d'uso

- **backend/database**: check che una mail/password sia stata già scelta da un altro utente
- **sicurezza**: check della qualità di una password
- etc...

Input

- insieme universo \mathcal{U} , $|\mathcal{U}| = u$
- per praticità assumiamo valori interi con ogni elemento che occupa $w = \log u$ bit
- insieme $S \subseteq \mathcal{U}$, $|S| = n$
- un elemento $y \in \mathcal{U}$

Output

- \top se $y \in S$, \perp altrimenti

Membership problem

Una prima soluzione

- costruisco un'hash table per \mathcal{S} con collisioni risolte per concatenazione
 - risposta in $\Theta(1)$ -time
 - struttura in $\mathcal{O}(n \log u)$ bit
- possiamo ridurre lo spazio in memoria ?
- no, se vogliamo il 100% di veri positivi e veri negativi
 - si, se assumiamo di poter ammettere falsi positivi **ma comunque non falsi negativi**

Membership problem

Ammettiamo falsi positivi **ma non falsi negativi**: approximate membership

- se $y \in \mathcal{S}$ voglio sempre ottenere \top , quindi ho **sempre** l'informazione corretta in merito al fatto che un elemento y sia in \mathcal{S}
- se $y \notin \mathcal{S}$ voglio che ottenere \perp con probabilità $\mathcal{P} \geq 1 - \delta$, con $\delta \in \mathbb{R}^+ \wedge \delta \rightarrow 0$. Si assume quindi di avere errori sui falsi positivi (ottengo \top e non \perp) con probabilità $\mathcal{P} \leq \delta$

Membership problem

Creiamo una struttura con n/δ bit, per \mathcal{S} , $|\mathcal{S}| = n$

- insieme universo \mathcal{U}
 - \mathcal{H} famiglia universale di funzioni hash per \mathcal{U} con $h_j : \mathcal{U} \rightarrow \{0, \dots, m-1\}$, con $m = n/\delta$
 - prendendo casualmente una funzione di hash $h \in \mathcal{H}$, popoliamo un array A , $|A| = m = n/\delta$, t.c.
- $$A[i] = \begin{cases} 1 & \text{se } \exists k \in \mathcal{S} \text{ t.c. } h(k) = i \\ 0 & \text{altrimenti} \end{cases}$$
- A è un bitvector e richiede n/δ bit

Query

- dato $x \in \mathcal{S}$ la funzione ritorna \top se $A[h(x)] = 1$, \perp altrimenti
- query in $\mathcal{O}(1)$

Membership problem

Creiamo una struttura con n/δ bit, per \mathcal{S} , $|\mathcal{S}| = n$

- insieme universo \mathcal{U}
- \mathcal{H} famiglia universale di funzioni hash per \mathcal{U} con $h_j : \mathcal{U} \rightarrow \{0, \dots, m-1\}$, con $m = n/\delta$
- prendendo casualmente una funzione di hash $h \in \mathcal{H}$, popoliamo un array A , $|A| = m = n/\delta$, t.c:

$$A[i] = \begin{cases} 1 & \text{se } \exists k \in \mathcal{S} \text{ t.c. } h(k) = i \\ 0 & \text{altrimenti} \end{cases}$$

- A è un bitvector e richiede n/δ bit

Considerazioni

- se $x \in \mathcal{S}$ si ottiene sempre 1
- se $x \notin \mathcal{S}$ si ottiene lo stesso T sse $\exists k \in \mathcal{S}$ t.c. $h(k) = h(x)$ \mathcal{H}
- famiglia universale quindi $P[h(k) = h(x)] \leq 1/m = \delta/n$
- probabilità che esista almeno una tale chiave k è ($P(A \cup B) \leq P(A) + P(B)$):

$$\sum_{k \in \mathcal{S}} P[h(k) = h(x)] \leq n/m = (\delta \cdot n)/n = \delta$$

Bloom filter

Miglioriamo ulteriormente il risultato appena ottenuto (n/δ bit).

Funzione hash ideale (*risultato solo teorico*)

Data una funzione di hash $h : \mathcal{U} \rightarrow \{0, 1, \dots, m-1\}$, per praticità $\mathcal{U} \subseteq \mathbb{N}$, questa è **ideale** sse, $\forall k \in \mathcal{U}$, $h(k)$ vale indipendentemente un valore uniformemente distribuito su $[0..m-1]$. Quindi, $\forall k \in \mathcal{U}$, $h(k)$ vale un qualsiasi intero tra 0 e $m-1$ con la stessa probabilità e tale valore non dipende dal valore di hash delle altre chiavi.

Bloom filter [2]

- risposta in $\mathcal{O}(1)$ -time
- spazio ulteriormente ridotto a $\mathcal{O}(n \log \frac{1}{\delta})$ bit
- probabilità non nulla di avere falsi positivi

7 / 18

Bloom filter

Struttura dati per \mathcal{S} , $|\mathcal{S}| = n$

- si considera un bitvector A , $|A| = m$
- si considera una famiglia di ℓ funzioni hash ideali $\mathcal{H} = \{h_0, \dots, h_{\ell-1}\}$
- $h : \mathcal{U} \rightarrow \{0, 1, \dots, m-1\}$, $\forall h \in \mathcal{H}$

Preprocessing / caricamento del bitvector A

- $\forall k \in \mathcal{S}$ e $\forall h_i \in \mathcal{H}$: $A[h_i(k)] = 1$
- quindi $\forall k \in \mathcal{S}$ abbiamo fino a ℓ bit pari a 1 in A
- "fino a" perché alcune $h_i, h_j \in \mathcal{H}$ potrei avere $h_i(k) = h_j(k)$ e se già $A[h_i(k)] = 1$ non avrò ulteriori modifiche in posizione $h_i(k) = h_j(k)$

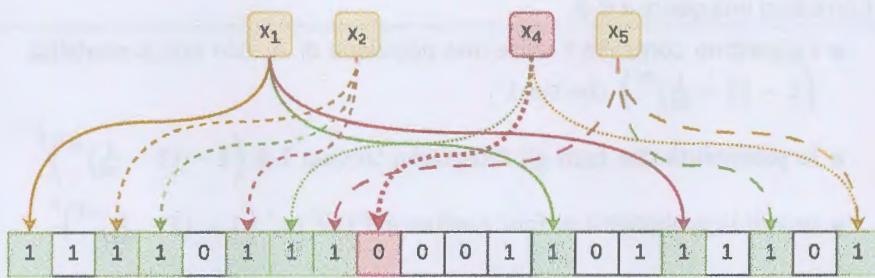
Si (caso) (Gere Ju b3 Appunti)

A è denominato **Bloom filter** di \mathcal{S} .

Query per l'approximate membership problem

Dato $x \in \mathcal{U}$ si ha che $x \in \mathcal{S}$ sse $A[h_i(x)] = 1, \forall h_i \in \mathcal{H}$

Bloom filter, un esempio



Analisi statistica dei Bloom filter

Alcune domande:

- quanto vale m ?
- che probabilità si ha di avere falsi positivi ($x \notin S$ ma $A[h_i(x)] = 1, \forall h_i \in \mathcal{H}$)?

Capendo quanto vale probabilità di avere falsi positivi, calcolando δ , si può derivare quanto vale m .

9 / 18

Analisi statistica dei Bloom filter

Qualche considerazione statistica

- in fase di preprocessing applico ℓ funzioni di hash ad ogni $k \in S$
- usando ℓ funzioni di hash ideali, la probabilità che il bit i -esimo non sia "influenzato" da k è: $\mathcal{P}(h(k) \neq i, \forall h \in \mathcal{H}) = (1 - \frac{1}{m})^\ell$
- alla fine del preprocessing $A[j] = 0$ sse $h(k) \neq j, \forall h \in \mathcal{H} \wedge \forall k \in S$, quindi: $\mathcal{P}(A[j] = 0) = (1 - \frac{1}{m})^{n\ell}$
- alla fine del preprocessing $A[j] = 1$ sse $\exists h \in \mathcal{H} \wedge \exists k \in S$, t.c. $h(k) = j$, quindi: $\mathcal{P}(A[j] = 1) = 1 - (1 - \frac{1}{m})^{n\ell}$
- quindi ci si aspetta che il numero atteso di bit pari a 1 in A sia $n \cdot (1 - (1 - \frac{1}{m})^{n\ell})$

Considero una query $x \notin S$

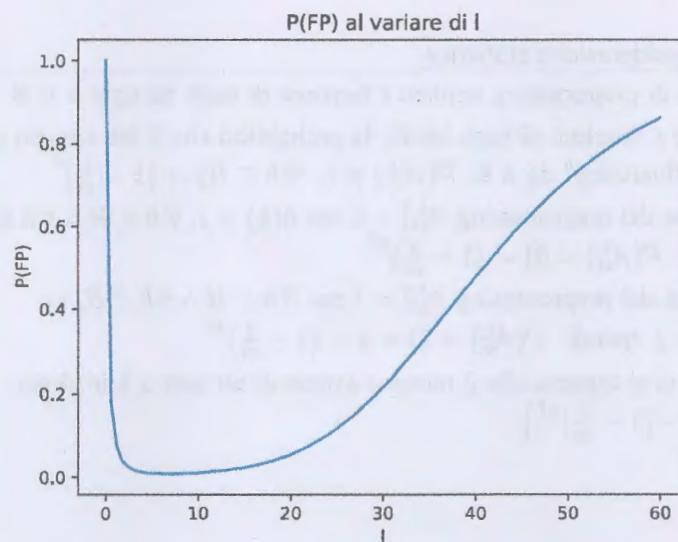
- l'algoritmo controlla ℓ volte una posizione di A , con una probabilità $\left(1 - \left(1 - \frac{1}{m}\right)^{n\ell}\right)$ che sia 1
- la probabilità che tutti gli ℓ tentativi trovino 1 è $\left(1 - \left(1 - \frac{1}{m}\right)^{n\ell}\right)^\ell$
- quindi la probabilità di falsi positivi è $P(FP) = \left(1 - \left(1 - \frac{1}{m}\right)^{n\ell}\right)^\ell$
- $P(FP) = \left(1 - \left(1 - \frac{1}{m}\right)^{n\ell}\right)^\ell = (1 - P(A[j] = 0))^\ell = P(A[j] = 1)^\ell$

Proseguiamo...

- $P(A[j] = 0) = \left(1 - \frac{1}{m}\right)^{n\ell} = e^{\ln\left[\left(1 - \frac{1}{m}\right)^{n\ell}\right]} = e^{\ln\left(1 - \frac{1}{m}\right) \cdot n\ell}$
- se $m \rightarrow \infty$ quindi $\frac{1}{m} \rightarrow 0$: $\ln\left(1 - \frac{1}{m}\right) \approx -\frac{1}{m} \rightarrow e^{\ln\left(1 - \frac{1}{m}\right) \cdot n\ell} \approx e^{-\frac{n\ell}{m}}$
- $P(A[j] = 0) = \left(1 - \frac{1}{m}\right)^{n\ell} \approx e^{-\frac{n\ell}{m}}$
- $P(FP) = (1 - P(A[j] = 0))^\ell \approx \left(1 - e^{-\frac{n\ell}{m}}\right)^\ell$

Ragioniamo su $\frac{n}{m}$ e ℓ

- $\frac{n}{m}$ è di fatto il numero medio di "elementi" che mappa su una posizione di A
- più ℓ è grande e più si "satura" il Bloom filter, avendo che si tende ad avere A riempito solo di 1
- più ℓ è piccolo e più si aumenta la probabilità di avere falsi positivi



Analisi statistica dei Bloom filter

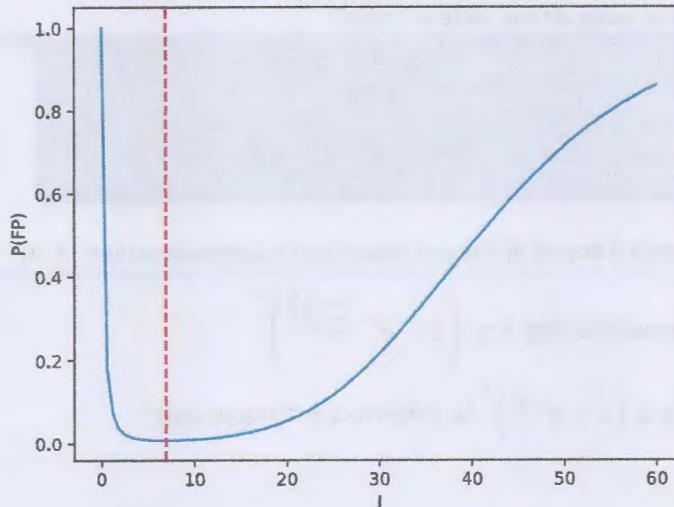
Vogliamo il minimo: deriviamo

- $\left(1 - e^{-\frac{n\ell}{m}}\right)^\ell$ è difficile da derivare ma sappiamo che, essendo il logaritmo una funzione monotona crescente, ogni x che minimizza $f(x)$ minimizza anche $\log(f(x))$ o $\ln(f(x))$
- $\ln\left(\left(1 - e^{-\frac{n\ell}{m}}\right)^\ell\right) = \ell \cdot \ln\left(\left(1 - e^{-\frac{n\ell}{m}}\right)\right)$
- $\frac{d\ell \cdot \ln\left(\left(1 - e^{-\frac{n\ell}{m}}\right)\right)}{d\ell} = \dots = \left(1 - e^{-\frac{n\ell}{m}}\right) + \frac{n\ell}{m} + \frac{e^{-\frac{n\ell}{m}}}{1 - e^{-\frac{n\ell}{m}}}$
- facendo i conti la derivata è nulla se $\ell = \ln(2) \cdot \frac{m}{n}$
- $e^{-\frac{n\ell}{m}} = \frac{1}{2}$ e $\frac{n\ell}{m} = \ln 2$

- la miglior scelta della cardinalità di \mathcal{H} è $\ell = \ln(2) \cdot \frac{m}{n}$
- $\mathcal{P}(A[j] = 0) \approx e^{-\frac{n\ell}{m}} = e^{-\frac{n \cdot \ln(2) \cdot \frac{m}{n}}{m}} = e^{-\ln 2} = \frac{1}{2} \Rightarrow \mathcal{P}(\text{FP}) = \left(\frac{1}{2}\right)^\ell = 0.6185^{\frac{m}{n}}$

Analisi statistica dei Bloom filter

P(FP) al variare di ℓ



Analisi statistica dei Bloom filter

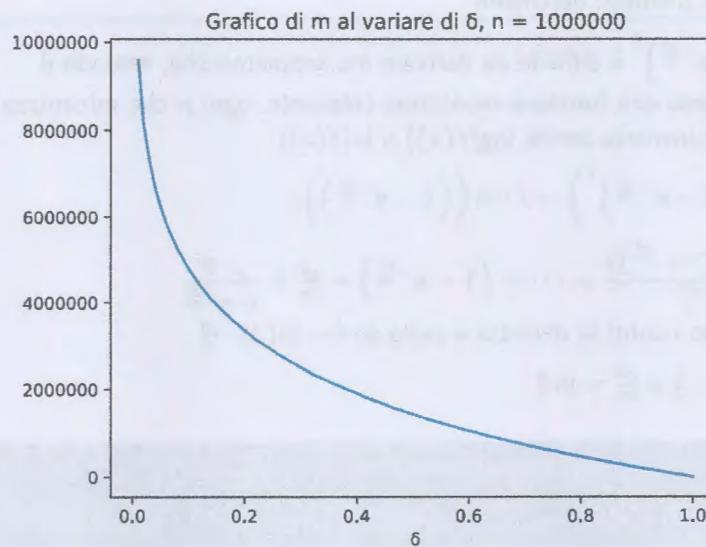
Manca solo da stabilire m in base alla massima probabilità d'errore δ che vogliamo

- $\mathcal{P}(\text{FP}) = \left(1 - \left(1 - \frac{1}{m}\right)^{n\ell}\right)^\ell \approx \left(1 - e^{-\frac{n\ell}{m}}\right)^\ell$
- con $\ell = \ln(2) \cdot \frac{m}{n}$: $\mathcal{P}(\text{FP}) \approx \left(1 - e^{-\frac{n\ell}{m}}\right)^\ell = \left(1 - e^{-\frac{n \cdot \ln(2) \cdot \frac{m}{n}}{m}}\right)^{\ln(2) \cdot \frac{m}{n}} = \left(\frac{1}{2}\right)^{\ln(2) \cdot \frac{m}{n}}$
- $\left(\frac{1}{2}\right)^{\ln(2) \cdot \frac{m}{n}} = \delta \rightarrow \ln\left(\frac{1}{2}\right)^{\ln(2) \cdot \frac{m}{n}} = \ln \delta \rightarrow \ln\left(\frac{1}{2}\right) \cdot \ln(2) \cdot \frac{m}{n} = \ln \delta \rightarrow -\ln^2(2) \cdot \frac{m}{n} \leq \ln \delta$

La dimensione m dell'array A deve essere $m = -\frac{n \ln \delta}{\ln^2(2)}$, con $n \neq 0$ per avere massima probabilità di falsi positivi pari a δ

Se $m = 8n$ e $\ell = 5$ si ha $\mathcal{P}(\text{FP}) = 0.024$

Analisi statistica dei Bloom filter



Analisi statistica dei Bloom filter

Quanto detto nelle ultime slide è "falso"

- approssimazione per $m \rightarrow \infty$
- si assume totale indipendenza tra il fatto che $A[h_i(x)] = 1$ e $A[h_j(x)] = 1$
- si assume che $\ell = \ln(2) \cdot \frac{m}{n} \in \mathbb{N}$

Si è dimostrato il bound di δ senza assunzioni e approssimazioni [3, 4]

- si è dimostrato che $\delta \leq \left(1 - e^{-\frac{\ell \cdot (n + \frac{1}{2})}{m-1}}\right)^\ell$
- rispetto a $\left(1 - e^{-\frac{n\ell}{m}}\right)^\ell$ la differenza è "trascutabile"

Counting Bloom filter

ESTENSIONE BLOOM FILTER, CONTA IL NUMERO DI OCCURRENZE DI CIASUN ELEMENTO.
PER LA VERIFICA, COMUNQUE CHE TUTTI I COUNTERS SIANO > 0

Counting Bloom filter [5, 6]

- una generalizzazione dei Bloom filter
- si tiene conto anche di quante funzioni di hash mappano in una certa posizione
- dato un elemento qualsiasi si verifica la presenza tramite il counting Bloom filter tramite una threshold ϑ
- in fase di preprocessing $\forall k \in \mathcal{S}$ e $\forall h_i \in \mathcal{H}$: $[A[h_i(k)]]+ = 1$
- in fase di query, dato $x \in \mathcal{U}$:
 - se $\exists h_i \in \mathcal{H}$ t.c. $[A[h_i(x)]] = 0$ o $[A[h_i(x)]] \leq \vartheta$ allora $x \notin \mathcal{S}$
 - se $\forall h_i \in \mathcal{H}$ $[A[h_i(x)]] > 0$ o $[A[h_i(x)]] > \vartheta$ allora "probabilmente" $x \in \mathcal{S}$, avendo $P(FP) \neq 0$
 - $[A[h_i(x)]]$ è una sovrastima del numero di elementi x in \mathcal{S}

\downarrow
TIGRE TRACIA DEL
NUERO PI
OCCURENZE
DEGLI ELEMENTI!

Shark [7]

- AlgoLab/shark
- un uso combinato di Bloom filter e bitvector succinti

Ocean Read Atlas (*ORA*) [8, 9]

Dato un gene lo si ricerca, tramite (**counting**) **Bloom filter**, in una collezione di geni geo-taggati (di specie marine) collezionati dalla spedizione della *Tara Oceans* negli oceani di tutto il mondo: <https://ocean-read-atlas.mio.osupytheas.fr/>.

Al cuore c'è lrobidou/fimpera [10], un miglioramento "ad hoc" dei counting Bloom filter.

Una libreria in C++

- mavam/libbf (circa 350 star su GitHub)

8.2.2.4 COUNT-MIN Sketch

Il numero di elementi in c'è massimo dei valori è a seconda

BLOOM FILTER

 $h_1 \rightarrow h_2 \rightarrow h_3$

Funzione H che funziona su un bitvector



bitmask

dovete essere molto lento \rightarrow il bitvector

Altamente veniva saturato

	C ₁	C ₂	C ₃	C _m
h ₁	1	0	--	0
h ₂	0	1	--	0
h ₃	0	0	--	0

$$h \rightarrow V \rightarrow \{0, \dots, m\}$$

mi appreso che nel caso minimo se

Una facciamo una matrice con una kapa per riga
quindi le righe dipendono da k perché δ (la probabilità)
le colonne sono date da m

Come per il Bloom Filter
ma m è calcolata
diversamente, è molto
più piccolo in questo
caso

esempio che entra 3 righe



$$\begin{aligned} h_1(k) &= C_1 \rightarrow 3 \quad \text{che collisioni} \\ h_2(k) &= C_2 \rightarrow 3 \quad \text{possono portare} \\ h_3(k) &= C_3 \rightarrow 3 \quad \text{ad avere 4} \quad \text{qui} \end{aligned}$$



$$h_1(k') = C_1$$

 $h_1(k')$ contiene solo

(vino) per contante davanti non appare k

calcolo quale hash è primo il minimo \rightarrow $\hat{k} \leftarrow \text{COUNT}(k) = 3$

I output sarà spesso una sovrapposizione perché abbiamo ancora le collisioni

Sulla stessa funzione di hash

Heavy Hitters Problem

Si prosegue dall'idea di Counting Bloom filter

- spesso non si vuole solo verificare o meno la presenza di un elemento in un set di dati
- spesso si vuole identificare solo "il più importante", ammettendo anche che possano essere più di uno (ma "pochi")
- questi elementi sono detti *heavy hitters* quindi si parla di **heavy hitters problem**

Aggiungiamo una *problematica*

- consideriamo dati davvero grossi, nell'ordine del 10^{18} (*trillioni*) o anche 10^{24} (*quadrillioni*)
- un *bloom filter* (non *counting* quindi non risponderemmo nemmeno al problema) richiede m bit per ogni elemento, **infattibile** (*private a fare i conti*) [11]

12 / 18

Heavy Hitters Problem

Esempi d'uso

- identificazione di prodotti popolari
- identificazione di frasi "importanti" sui social network
- identificazione di "trend"
- identificazione di sottosequenze over-esprese in un genoma
- etc...

Stream di dati

Per essere precisi

- con *stream di dati* si intende una sequenza di dati passati uno ad uno alla struttura dati
- quindi, data una sequenza $S = s_0, s_1, \dots, s_{n-1}$, prima si considera s_0 , poi s_1 etc... fino a s_{n-1}
- si costruisce quindi una struttura dati che può essere interrogata con nuovi valori $x \in \mathcal{U}$

Query per $x \in \mathcal{U}$

- quante volte appare x nello stream: $|\{i \in \{0, 1, \dots, n-1\} | s_i = x\}|$
- quanti elementi distinti si hanno nello stream: $|\{s_i | i \in \{0, 1, \dots, n-1\}\}|$

13 / 18

Count-min sketch

Count-min sketch [12]

- una delle più importanti strutture dati usate per grandi stream di dati
- vedremo che richiede davvero poco spazio in memoria
- concettualmente molto simile ad un Bloom filter

- insieme universo \mathcal{U}
- uno stream \mathcal{S} lungo n costruito da \mathcal{U}
- due parametri d'errore δ e ε
- $\mathcal{H}, |\mathcal{H}| = \ell = \lceil \ln \frac{1}{\delta} \rceil$, famiglia di funzioni hash universale per \mathcal{U}
- si impone che $h_i : \mathcal{U} \rightarrow \{0, \dots, m\}$, $\forall h_i \in \mathcal{H}$, con $m = \lceil \frac{\varepsilon}{\delta} - 1 \rceil$

14/18

Count-min sketch

La struttura dati

- matrice bidimensionale T con ℓ righe (una per ogni $h_i \in \mathcal{H}$), e m colonne
- si hanno quindi ℓ hash table indipendenti con m entry ciascuna
- T viene inizializzata con tutti gli elementi a 0

Preprocessing / caricamento di T

- si considerano in ordine tutti gli $x_j \in \mathcal{S}$, con $j = 0, 1, \dots, n-1$
- sappiamo che ogni h_i ha di fatto come codominio l'insieme degli indici di colonna
- "inserire" x_j in T : si incrementa di 1 $T_{h_i}[h_i(x_j)]$, $\forall h_i \in \mathcal{H}$
- il caricamento si effettua in $\mathcal{O}(n)$ -time

Count-min sketch

Query per $q \in \mathcal{U}$

- si applica ogni funzione di hash a q
- si tiene traccia di ogni $T_{h_i}[h_i(q)]$, $\forall h_i \in \mathcal{H}$
- si restituisce il minimo tra tali valori, che è una stima (una frequenza approssimata \hat{a}_q) di quante volte occorre q in \mathcal{S} :
$$\hat{a}_q = \min_{h_i \in \mathcal{H}} T_{h_i}[h_i(q)]$$
- una query si effettua in $\mathcal{O}(\ell)$ -time

Si dimostra che

- data a_q , frequenza reale di q in \mathcal{S} , si ha che $a_q \leq \hat{a}_q$
- $\hat{a}_q \leq a_q + \varepsilon \cdot n$ con probabilità almeno $1 - \delta$

Spazio richiesto dalla tabella T per count-min sketch

Si ha una matrice di dimensione $\ell \times m = \lceil \ln \frac{1}{\delta} \rceil \times \lceil \frac{\varepsilon}{\delta} - 1 \rceil$ con valori che richiedono $\log n$ bit: $(\lceil \ln \frac{1}{\delta} \rceil \cdot \lceil \frac{\varepsilon}{\delta} - 1 \rceil \cdot \log n)$ bit

Count-min sketch, un esempio

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	0	0	0	0	0	0	0	0	0	0
h_2	0	0	0	0	0	0	0	0	0	0	0
h_3	0	0	0	0	0	0	0	0	0	0	0
h_4	0	0	0	0	0	0	0	0	0	0	0

15 / 18

Count-min sketch, un esempio

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(4) = \xi_2 \quad h_2(4) = \xi_1 \quad h_3(4) = \xi_7 \quad h_4(4) = \xi_{11}$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	0	0	0	0	0	0
h_2	1	0	0	0	0	0	0	0	0	0	0
h_3	0	0	0	1	0	0	0	0	0	0	0
h_4	0	0	0	0	0	0	0	0	0	0	1

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(2) = \xi_6 \quad h_2(2) = \xi_2 \quad h_3(2) = \xi_8 \quad h_4(2) = \xi_1$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	1	0	0	0	0	0
h_2	1	1	0	0	0	0	0	0	0	0	0
h_3	0	0	0	1	0	0	0	1	0	0	0
h_4	1	0	0	0	0	0	0	0	0	0	1

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(6) = \xi_8 \quad h_2(6) = \xi_{10} \quad h_3(6) = \xi_5 \quad h_4(6) = \xi_9$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	1	0	1	0	0	0
h_2	1	1	0	0	0	0	0	0	0	1	0
h_3	0	0	0	1	1	0	0	1	0	0	0
h_4	1	0	0	0	0	0	0	0	1	0	1

$$\mathcal{S} = \{4, 2, 6, 2, 7, 1, 6\}$$

$$h_1(2) = \xi_6 \quad h_2(2) = \xi_2 \quad h_3(2) = \xi_8 \quad h_3(2) = \xi_1$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	2	0	1	0	0	0
h_2	1	2	0	0	0	0	0	0	0	1	0
h_3	0	0	0	1	1	0	0	2	0	0	0
h_4	2	0	0	0	0	0	0	0	1	0	1

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(2) = \xi_6 \quad h_2(2) = \xi_2 \quad h_3(2) = \xi_8 \quad h_3(2) = \xi_1$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	3	0	1	0	0	0
h_2	1	3	0	0	0	0	0	0	0	1	0
h_3	0	0	0	1	1	0	0	3	0	0	0
h_4	3	0	0	0	0	0	0	0	1	0	1

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(7) = \xi_6 \quad h_2(7) = \xi_4 \quad h_3(7) = \xi_8 \quad h_3(7) = \xi_9$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	4	0	1	0	0	0
h_2	1	3	0	1	0	0	0	0	0	1	0
h_3	0	0	0	1	1	0	0	4	0	0	0
h_4	3	0	0	0	0	0	0	0	2	0	1

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(1) = \xi_8 \quad h_2(1) = \xi_2 \quad h_3(1) = \xi_5 \quad h_3(1) = \xi_1$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	4	0	2	0	0	0
h_2	1	4	0	1	0	0	0	0	0	1	0
h_3	0	0	0	1	2	0	0	4	0	0	0
h_4	4	0	0	0	0	0	0	0	2	0	1

Count-min sketch, un esempio

$$\mathcal{S} = \{4, 2, 6, 2, 2, 7, 1, 6\}$$

$$h_1(6) = \xi_8 \quad h_2(6) = \xi_{10} \quad h_3(6) = \xi_5 \quad h_3(6) = \xi_9$$

T	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}
h_1	0	1	0	0	0	4	0	3	0	0	0
h_2	1	4	0	1	0	0	0	0	0	2	0
h_3	0	0	0	1	3	0	0	4	0	0	0
h_4	4	0	0	0	0	0	0	0	3	0	1

Count-min sketch, un esempio

- alcune frequenze sono sovrastimate
- se avessi $q = 2$ otterrei $\hat{a}_2 = \min_{h_i \in \mathcal{H}} T_{h_i}[h_i(2)] = \min\{4, 4, 4, 4\} = 4$
- se avessi $q = 6$ otterrei $\hat{a}_6 = \min_{h_i \in \mathcal{H}} T_{h_i}[h_i(6)] = \min\{3, 2, 3, 3\} = 2$
- potrei avere $q \notin \mathcal{S}$ e ottenere come minimo un valore $x \neq 0$

15 / 18

Analisi statistica di Count-min sketch

Se con i soli algoritmi di caricamento e query di T si dimostra che $a_q \leq \hat{a}_q$ (a causa delle collisioni si ottengono sovrastime ma mai sottostime della frequenza), più complesso è dimostrare $\hat{a}_q \leq a_q + \varepsilon \cdot n$ con probabilità almeno $1 - \delta$.

Ingredienti per la dimostrazione [12]

- usiamo una variabile $\mathcal{I}_{i,j,k}$:

$$\mathcal{I}_{i,j,k} = \begin{cases} 1 & \text{se } (i \neq k) \wedge (h_j(i) = h_j(k)) \\ 0 & \text{altrimenti} \end{cases}$$

- si ha per definizione che $\mathbb{E}[\mathcal{I}_{i,j,k}] = \mathcal{P}(h_j(i) = h_j(k)) \leq \frac{1}{|\mathcal{H}|} = \frac{\varepsilon}{e}$

Analisi statistica di Count-min sketch

Linearity of expectation, con \mathcal{X} variabile casuale e $a, b \in \mathbb{R}$ scalari

$$\mathbb{E}[a \cdot \mathcal{X} + b] = a \cdot \mathbb{E}[\mathcal{X}] + b$$

Markov's inequality, con \mathcal{X} variabile casuale non negativa e $a \in \mathbb{R}$ scalare

$$\mathcal{P}(\mathcal{X} \geq a) \leq \frac{\mathbb{E}[\mathcal{X}]}{a}$$

Analisi statistica di Count–min sketch

Continuiamo:

- si definisce una variabile casuale non negativa $\mathcal{X}_{i,j}$, legata alla funzione di hash h_j
- nel dettaglio: $\mathcal{X}_{i,j} = \sum_{k=1}^n \mathcal{I}_{i,j,k} \cdot a_k$ (notare che: $a_k \geq 0 \implies \mathcal{X}_{i,j} \geq 0$)
- per costruzione $T_{h_j}[h_j(i)] = a_i + \mathcal{X}_{i,j}$, ovvero la cella di T relativa all'elemento query i contiene la somma tra la frequenza esatta di i , data da a_i , e la somma di tutte le frequenze di altri elementi diversi da i che hanno una collisione tramite h_j , data da $\mathcal{X}_{i,j}$
- possiamo quindi garantire che $T_{h_j}[h_j(i)] \geq a_i$

Statisticamente si arriva a dire che:

$$\mathbb{E}[\mathcal{X}_{i,j}] = \mathbb{E} \left[\sum_{k=1}^n \mathcal{I}_{i,j,k} \cdot a_k \right] \leq \sum_{k=1}^n a_k \cdot \mathbb{E}[\mathcal{I}_{i,j,k}] \leq \frac{\varepsilon}{e} \cdot n$$

Analisi statistica di Count–min sketch

Infine:

$$\begin{aligned} \mathcal{P}(\hat{a}_i > a_i + \varepsilon \cdot n) &= \mathcal{P}(T_{h_j}[h_j(i)], \forall j) > a_i + \varepsilon \cdot n \\ &= \mathcal{P}(a_i + \mathcal{X}_{i,j} > a_i + \varepsilon \cdot n, \forall j) \\ &= \mathcal{P}(\mathcal{X}_{i,j} > e \cdot \mathbb{E}[\mathcal{X}_{i,j}], \forall j) \\ &< \left(\frac{\mathbb{E}[\mathcal{X}_{i,j}]}{e \cdot \mathbb{E}[\mathcal{X}_{i,j}]} = e^{-1}, \forall j \right) < e^{-\ell} = e^{-\lceil \ln \frac{1}{\delta} \rceil} \leq \delta \end{aligned}$$

"Ribalto" la probabilità richiesta

$$\mathcal{P}(\hat{a}_i \leq a_i + \varepsilon \cdot n) \geq 1 - \delta$$

Conclusioni

- abbiamo recuperato il concetto di funzione di hash e di hash table
- abbiamo affrontato il *membership problem* tramite **Bloom filter**
- abbiamo esteso il concetto di *Bloom filter* a quello di **counting Bloom filter**
- abbiamo analizzato l'*heavy hitters problem* tramite count–min sketch

Ci siamo 3-SAT: \exists cluster di dimensione k

\hookrightarrow dato $G = (V, E)$ non connesso

$C, |C|=k$, cluster sse $\forall i, j \in C \quad (i, j) \in E \rightarrow$ ogni nodo è collegato

Dobbiamo dimostrare che è NP-completo



C'è una MT che determina se S è in NP
MT in NSUUF, O(G)
O(G) in tempo
polinomiale

S è in NP-HARD



$$S \subseteq V$$

Ci serve capire se posso dare una soluzione S , vogliare polinomiale
con una DTM

Dove vogliamo che S sia una cluster di dimensione k
vado a controllare le cardinalità di S

$$|S|=k$$

Ogni nodo deve avere connettività

$$\forall (i, j) \in S \quad (i, j) \in E$$

Io posso vedere in termini polinomiali? perciò quanti nodi vuoi
uscare ho da ogni nodo, al massimo n bisogna avere numero
di un grande complesso:

$$\frac{k(k-1)}{2} = \text{gradi - binomio}$$

$$\rightarrow \frac{|V|(|V|-1)}{2}$$

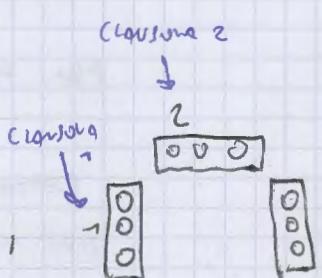
$$\hookrightarrow O(|V|)^2$$

Autro argomento di cui da fare controllo per convincere in termini polinomiali

$$3\text{-SAT} \leq_p C_f$$

\downarrow
in clausole e in literali

1) A generare x_i nego v_i t.c. $\text{label}(x_i) = v_i$



2) Controlla due nodi v_i : i) sono in "clausole" diverse

grado 3

ii) $\text{label}(v_i) \neq \text{label}(v_j)$

TG-ES

\rightarrow se $S \rightarrow T$ 3-SAT $\rightarrow C_d$ se la soluzione è valida per 3-SAT

Alzioni delle regole valgono anche per C_d

Siano $x_1 \dots x_n$ variabili; F (formula di 3-SAT)

$v_1 \dots v_n$ gli assegnamenti per cui $F = T$

$$\text{Formula SAT: } (\underset{\text{OR}}{\vee} \underset{\text{AND}}{\vee}) \wedge (\underset{\text{OR}}{\vee} \underset{\text{AND}}{\vee})$$

per ogni v_i decit m clausule almeno un esponente sia vero

m è la stessa cosa del precedente, dovetti chiamare diverso



un'altra più esempio è una clausa

da ogni gadget scelgo un valore T

$$V \subseteq V' \quad |V'| = m$$

CLASSE = GADGET
VARIABILE = L'ESPRESSO

ora facciamo l'opposto

3-SAT $\nrightarrow C_d$

$$U, \quad |U| = m$$

che si chiama

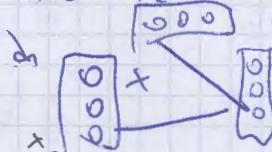
se questa non è una soluzione per 3-SAT

allora per avere questo esito
bisogna che tutti i valori siano

per assurdo $(x_i, \neg x_i) \in E$

ma allora non può essere
una clausa

~~ossia assurdo ed $O(k^2)$~~



$$x_1 = \neg x_2$$

e
assurdo

3-SAT $\leq_p C_d$

$\rightarrow C_d \in \text{NP-HARD}$

TC-ES
ES. 2 02/23

$L \in \text{PSPACE}$

a) Considerare Halting problem in bounded time

I: DTM, stringa $x \in \Sigma^*$ $n \in \mathbb{N}$

O: $\begin{cases} y \in \Gamma \text{ termina in } n \text{ passi} \\ N \text{ autorizzata} \end{cases}$

DTM \neq non un contatore

+

NUO facciamo con bounded space

SARÀ TUTTO IL COMBINATORIO SU UNA TM E VARI A
COMBINAZIONE SE SHAI CHE' MENO

ES 2 GRADO

Tutto dimostra che $\Pi \in NP \rightarrow \Pi \in co-NP$
se provato

Dimostrazione
per tutto

$\vdash NDTM = Y \uparrow N \downarrow NDTM$

OUTPUT
YES
NO

? → POSSIAMO CALCOLARE SE
IL TUTTO HA RETURNA
PER QUESTO DOBBIANO

recursiva accettante
che invierte
l'ordine

A leggere abbiamo visto che SAT non è in coNP

A leggere

SAT risponde YES quando esiste una combinazione TRUE
NO quando non ne esiste NESSUNO

Almeno

Quindi l'inverso non va bene

Possibile la NDTM cerca ALMENO UNA TRUE PER RETURNA

TRUE MA SE NEL CIRCUITO COME LO HA LA TM ~~NON~~ SE cerca
LO XQS MA NON È CORRETTO POSSIBILE DOVREBBE CONCERNIRE TUTTI

SAT \notin coNP \rightarrow sempre Y

→ nella seconda accettante

TCSES

ES 3 GRUPO

TIZIO COSTRUISE UNA RIDUZIONE DA SAT A HAMILTON PROBLEM

$$\sum_{N} f_N$$

$\forall N'$ INPUT: $N \in \mathbb{N} \rightarrow$ tutti i raw Y si N restano pari
tutti i raw N si N vanno in loop

ISTANZA $I \ni N$ in $\langle N, I \rangle \ni N'$

SAT E NP-COMPLETO
Hamilton problem E NP-HARD \rightarrow STA COSA HA SENSO?

Ci sono
lo yes

$\overbrace{\text{è polinomiale?}}^{\text{O}}$, se l'unica cosa polinomiale \checkmark la risata è la copia dell'input
 $\overbrace{\text{è riduzione?}}^{\text{O}}$ se perche' sua cosa ha senso, è un'applicazione corrente per la
dimostrazione di riduzione

Quindi arriviamo a dire che Hamilton problem E NP-HARD

ES 3 LUTTO

vertex cover \downarrow INPUT
 $I: G(V, E) \rightarrow$ non vuoto $k \in \mathbb{N}$
 $O: \exists C \subseteq V, |C|=k, \forall (i, j) \in E$
 $i \in C \vee j \in C$

EXISTS $i: \text{ARRAY A } \lg N$

$O: \exists x \in \text{A } N \text{ ACCIDENTI}$

TIZIO RIDUZIONE

A