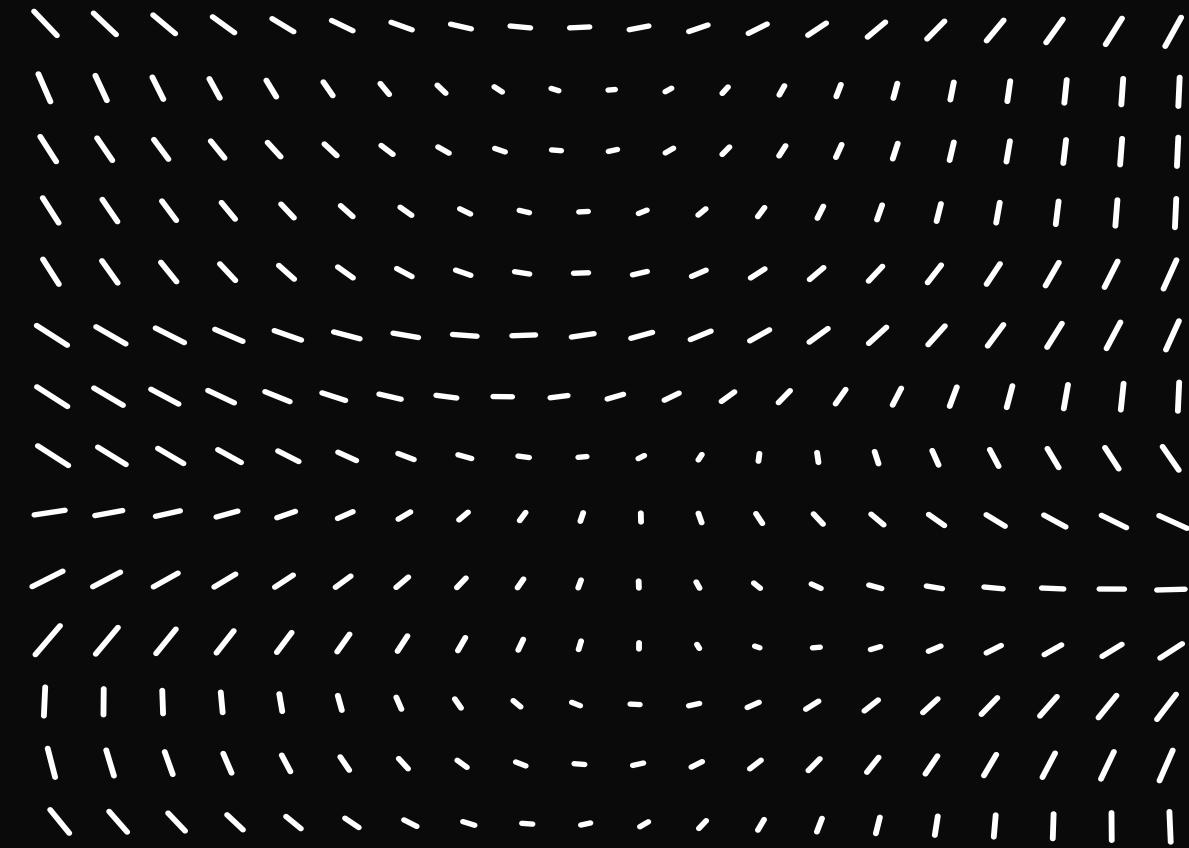


valentino.freschi@uniurb.it



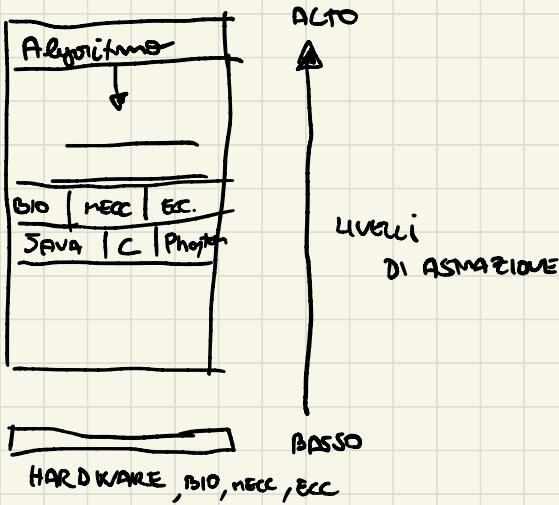
20/02/2023 (AUDIO 1)

Cos'è un algoritmo

- Sequenza di passi finiti interpretabili da un ESECUTORE (non necessariamente con computer)
- Azione linguisti di programmazione
- Non riduce un tempo finito, nonostante i passi siano finiti

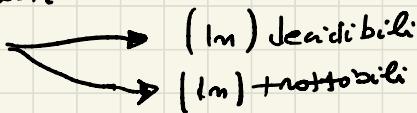
Dove da un matematico persiano Muhammed al-Kwarizmi.

ES. RICETTA DI CUCINA



21/02/2023 (AUDIO 2)

- (1) Algoritmo
- (2) Struttura dati
- (3) Problemi

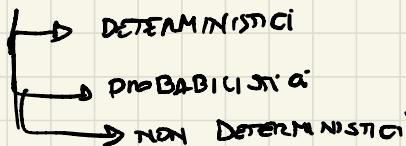


PSEUDO CODICE → ASSOMIGLIA AD UN CODICE SLEGATO DALLA SINTASSI

CLASSIFICAZIONE
(N BASE AL
NUMERO DI
PASSI)

SEQUENZIALI (UNO ALLA VOLTA)
PARALLELI (PIÙ PASSI PER VOLTA)

RISOLUZIONE DELLE SCELTE



I = insieme intenzioni dati input

S = insieme soluzioni

$P \subseteq I \times S$

→ APPARTENENZA O NO

$P = \text{problema}$ \rightarrow DECISIONE $S = \{0,1\}$
 \rightarrow RICERCA (GENERICA SOLUZIONE PER OGNI DATO DI INPUT)
 \rightarrow OTTIMIZZAZIONE / TROVARE LA SOLUT. OTTIMA RISP A UN CRITERIO)

Problema = cosa

Algoritmo = come

Dato P è un algoritmo, algoritmo è corretto se termina e da la soluzione corretta, si dice corretto se non termina nei tutti gli input

Gli algoritmi corretti su uno stesso problema si possono confrontare
in base:

RISORSE COMPUTAZIONALI

- TEMPO DI CALCOLO
- Spazio di memoria
- Bande

23/02/2023

(AUDIO 3)

CLASSI DI PROBLEMI

Possiamo risolvere ogni problema con un procedimento meccanico?

- TEORIA DELLA COMPLESSITÀ

• PROBLEMI DECIDIBILI E INDECIDIBILI

↳ EQ. DIOPANTINE & COEFF. INTEGI
↳ PROBLEMA DELLA PERMUTA



Termine (A, i)

1 Termine su i
0 Non termine su i

Paradosso (α)

While (Termine (A, A)) ;

Paradosso (Paradosso (α)) \rightarrow CONTRADDIZIONE LOGICA

P, NP, NPC

Problemi trattabili e intattabili

Lo risolviamo

Un modo efficiente

- Problemi intrinsecamente intattabili

Lo lo risolviamo sempre ampliando un insieme di possibili soluzioni esponenzialmente (la dimensione della soluzione)

- Problemi non intrinsecamente intattabili

• Problemi risolvibili in tempo polinomiale

(EFFICIENTE)

• Problemi risolvibili in tempo esponenziale

(NON EFFICIENTE)

$$m \rightarrow 2^m \quad (2 > 1) \quad \text{Esp.}$$

$$n^k \quad (\text{Efficiente})$$

P è l'insieme dei problemi di decisione risolvibili in tempo polinomiale da un algoritmo deterministico

NP è l'insieme dei problemi di decisione risolvibili in tempo polinomiale da un algoritmo non deterministico (infiniti nuovi computazionali)

NP è l'insieme dei problemi di decisione le cui soluzioni può essere verificate in tempo polinomiale da un algoritmo deterministico

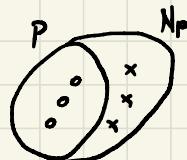
In che relazione stanno?

P è contenuto in NP $P \subseteq NP$

P vs NP (1/7 millennium problem)

Non c'è stato un grado di provarelo

Ipotesi:



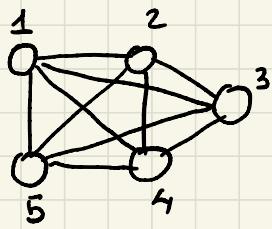
$$P \subseteq NP$$

\nwarrow
se fosse la stessa classe P ed NP potremmo trovare un algoritmo efficiente

P vs NP uno dei monimi problemi dell'informatica teoria

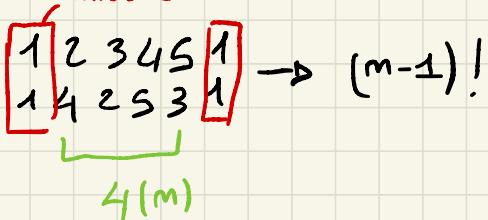
per gli algoritmi esponenziali c'è un modo per risolvere in maniera efficiente

TSP \rightarrow Traveling Salesman Problem (NP problem)



MIGLIOR ALGORITMO $\rightarrow m^2 \cdot 2m$

INIZIO E ARRIVO NOTI



MIGLIOR ALGORITMO EDotto $\rightarrow m^2 \cdot 2m \rightarrow$ ESPOENZIALE

↓
PROBLEMA DI CLASSE NP

27/02/2023 (AUDIO 4)

Algoritmi esponenziali (INEFFICIENTI)

Algoritmi polinomiali (EFFICIENTI)

Ci sono problemi che non mi permettono risolvere in modo efficiente?

Lo problema della soddisficiabilità (SAT)

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

$$x = 0$$

$$y = 0$$

$$z = 1$$

RIDUCIBILITÀ: (in tempo polinomiale)

$$\begin{array}{c} \text{problem} \\ \downarrow \\ P_1 \subseteq I_1 \times \{0,1\} \\ \downarrow \quad \text{INPUT} \\ P_2 \subseteq I_2 \times \{0,1\} \end{array}$$

- DATI E PROBLEMI DI DECISIONE

$$\cdot P_1 \rightarrow P_2 (P_1 \leq_p P_2)$$

$$P_1 \xrightarrow{\text{REDUCIBILE}} P_2 \quad o \quad P_1 \xleftarrow[\text{IN TEMPO POLINOMIALE}]{} P_2$$

\curvearrowright

NOTAZIONI UGUALI

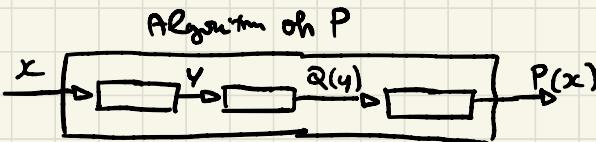
Se existe un algoritmo det. polinomial che calcola una funzione $f + c$:

$$f : I_1 \rightarrow I_2 \quad +.c. \quad \forall i \in I_1, \forall s \in \{0,1\}$$

$$(i, s) \in P_1 \Leftrightarrow (f(i), s) \in P_2$$

TRASFORMAZIONE PRESERVA LA SOMMATORIE

RIDOCIBILITÀ IN TEMPO POLYMONIAC



$$\Theta_A \rightarrow B \Rightarrow A \in P$$

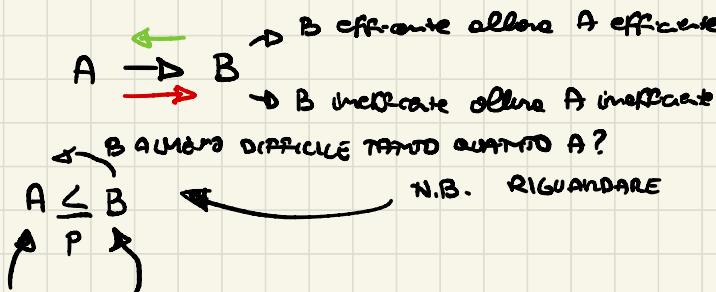
$$\exists B \in \mathbb{P}$$

① A non può essere risolto in tempo polinomiale se B
oltre meno B

② $A \rightarrow B$

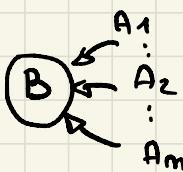
↓
Neanche B ammette un algoritmo efficiente

DIFFICOLTÀ



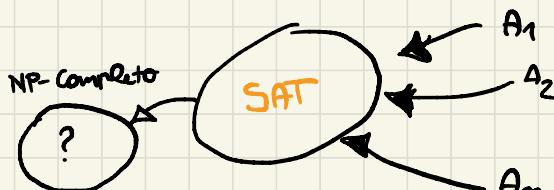
RIDURRE UN PROBLEMA IN UN'ALTRA

$P \in NP$ (con RIDUCIBILITÀ)



1971 → TEOREMA DI COOK

TUTTI I PROBLEMI DI CLASSE NP SONO RIDUCIBILI
IN TEMPO POLINOMIALE A DI SAT



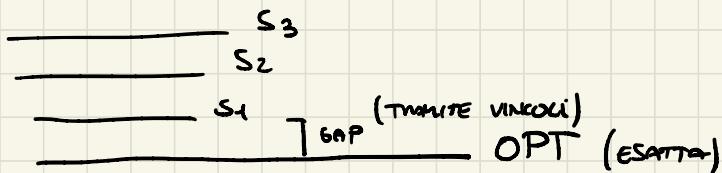
Concluso

Se esiste un algoritmo deterministico che risolve SAT in tempo
polinomiale, allora $P = NP$

IN REALTÀ SAT È SOLO UNO DEI TANTI PROBLEMI

NP - completeness un problema NP è detto NP-completo

Cosa fare se ci imbattiamo in un problema NP-complet?



Algoritmi di approssimazione

Algoritmi (montecarlo - los voyos) - probabilistici

28/02/2023 (AUDIO 4)

COMPLESSITÀ ASINTOTICA

- CONFRONTO ALGORITMI STESSA EFFICACIA \rightarrow PIÙ EFFICIENTE IN BASE AL $T(n)$ tempo di esecuzione
- Indipendenza da tecnologie (dipendenza da circuiti fatti)
- Modello di calcolo : macchine RAM (modello Von Neumann)
 - Operazioni elementari : sottrazione, confronto, logiche, trasformate,
 - Conta uniforme (nelle pratica non è così) controllo

- Nr. pari bene c'è l'istruzione di incremento

$$\begin{aligned} & x = 3; \quad \swarrow \text{si} \\ & x = \text{funzione}(a, b, c); \quad \begin{array}{l} \text{no} \\ \text{si} \end{array} \\ & x = (3 + a + b) / c; \quad \begin{array}{l} \text{no} \\ \text{no} \end{array} \\ & x = (3 + \text{funz}(a, b, c)) / 3; \quad \text{no} \end{aligned}$$

perché circano di mettere le funzioni tranne le funzioni di libreria standard

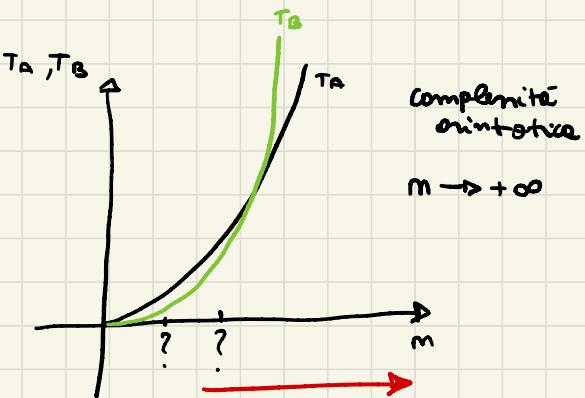
- Casi : (volontario) → det' istanze del dato input che **minimizza** il tempo di esecuzione $T(m)$ (limite sup.)
- ponino → la somma dei tempi di esecuzione parso per le probabilità di occorrenza (es. $0,1(T_1) + 0,1(T_2)$)
- medio → INPUT che **minimizza** $T(m)$ (limite inf.)
- ottimo → ponino

ponino

ottimo

$T_A(m), T_B(m), \dots, T_K(m) \Rightarrow$ confrontare le funzioni

Vogliamo capire quanto l'algoritmo cresce
→



Notazioni:

$O(f_n(m))$

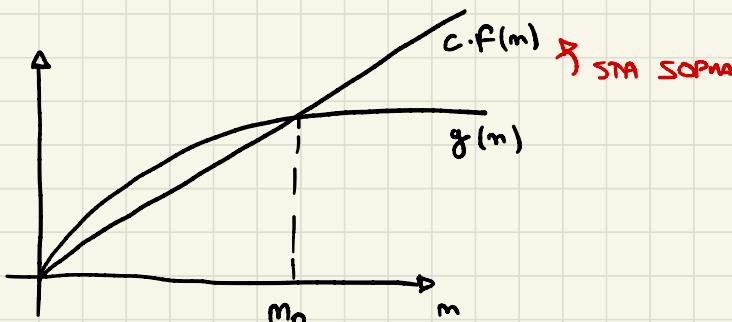
$\Omega(f_n(m))$

$\Theta(f_n(m))$

LIMITE ASINTOTICO SUPERIORE NOTAZIONE $O(\dots)$

tempo di esecuzione

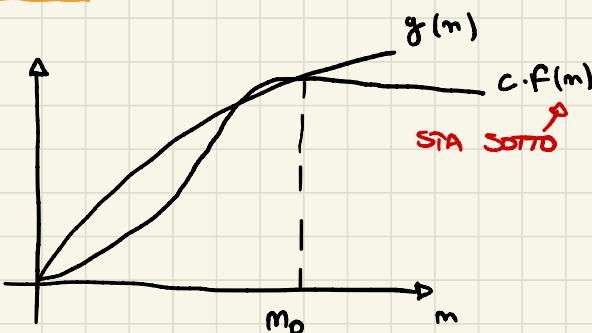
$$g(m) = O(f(m)) \Leftrightarrow \exists c, m_0 > 0, \text{ se } m > m_0, g(m) \leq c \cdot f(m)$$



LIMITE ASINTOTICO INFERIORE NOTAZIONE $\Omega(\dots)$

tempo di esecuzione

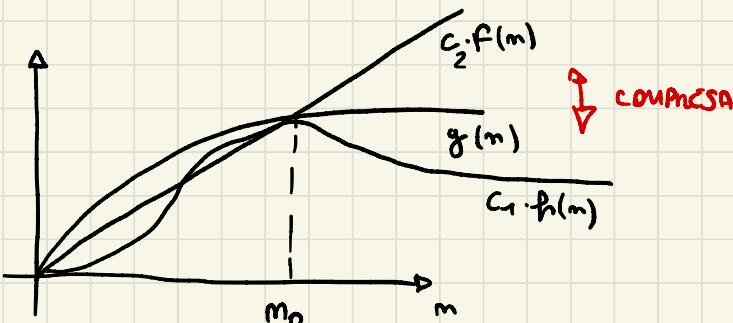
$$g(m) = \Omega(f(m)) \Leftrightarrow \exists c, m_0 > 0, \text{ se } m > m_0, g(m) \geq c \cdot f(m)$$



LIMITE ASINTOTICO STRETTO NOTAZIONE $\Theta(\dots)$

tempo di esecuzione

$$g(m) = \Theta(f(m)) \Leftrightarrow c_1, c_2, m_0 > 0, \text{ se } m > m_0, c_1 \cdot f(m) \leq g(m) \leq c_2 \cdot f(m)$$



COMPLESSITÀ LINEARE

$$T(m) = O(1)$$

costante

$$T(m) = O(\log(m))$$

logaritmica

$$T(m) = O(m)$$

lineare

$$T(m) = O(m \log m)$$

pseudo lineare

$$T(m) = O(m^2)$$

quadratica

$$T(m) = O(m^3)$$

cubica

$$T(m) = O(m^k), k > 0$$

polinomiale

$$T(m) = O(a^m), a > 1$$

esponenziale

21/03/2023

(AUDIO 6)

CALCOLO COMPLESSITÀ

Algoritmi non riconibili

- istruzioni di orologmetro (valutazione espressione) es: $x = 5$

- sequente di istruzioni S_1, S_2, \dots, S_m aiasano come

$$\xrightarrow{\text{Somma dei tempi}} T_i(m) = O(f_i(m))$$

ASSEGNAZIONE $x = e \rightarrow T(m) = O(1)$ Prova di chiamata a funzione

\uparrow \rightarrow se ho chiamata a funzione

$$\begin{aligned} T(m) &= 1 + T'(m) \\ &= O(f'(m)) \end{aligned}$$

$$T(m) = \begin{cases} 1 = O(1) \\ T'(m) + 1 = O(f'(m)) \end{cases}$$

SEQUENZA

$$T(m) = \sum_{i=1}^m T_i(m) = O(\max\{f_i(m) \mid 1 \leq i \leq m\})$$

$$\begin{aligned} S_1 &\rightarrow O(n) \\ S_2 &\rightarrow O(1) \rightarrow O(n^3) \\ S_3 &\rightarrow O(n^3) \end{aligned}$$

SELEZIONE

$$f_\beta(\beta) \quad S1 \text{ else } S2$$

$$\beta : T_0(m) = O(f_0(m))$$

$$S1 : T_1(m) = O(f_1(m))$$

$$S2 : T_2(m) = O(f_2(m))$$

$$T(m) = T_0(m) + \underset{\downarrow}{\text{minimo}} \left(T_1(m), T_2(m) \right) = O \left(\max \left\{ f_{T_0}(m), \underset{\text{espressione di grande}}{\text{O}} \left(f_{T_1}(m), f_{T_2}(m) \right) \right\} \right)$$

$\text{op} = \max(\text{costo puro})$, $\min(\text{costo effettivo})$

cicli while (β) \rightarrow espressione di grande S

$$i(m) = O(f_i(m))$$

$$\beta : T_1(m) = O(f_{T_1}(m))$$

$$S : T_2(m) = O(f_{T_2}(m))$$

$$T(m) = i(m) \cdot (T_1(m) + T_2(m)) + T_1(m)$$

ESEMPI ALGORITMI E CALCOLO COMPLESSITÀ:

CALCOLARE FATTORIALE DI UN NUMERO

$$m! \quad m \geq 1$$

calcolo - fatt+(m)

for ($fatt+=1$, $i=2$; ($i \leq m$); $i++$)

$fatt *= i$

return $|fatt|$

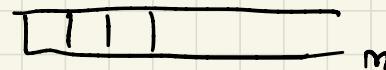
$$T(m) = 1 + (m-1) \left(1 + 1 + 1 \right) + 1 = 3m - 1 = O(m)$$

3 PASSI

- ① SE È CORRETTO?
- ② QUANTE RISORSE COMPUTAZIONALI SERVONO?
- ③ POSSIAMO FARLE DI MEGLIO?

CALCOLO DEL MASSIMO DI NUMERO DI INTERI

calcolo_max (e , m)



$$F_0 = 0$$

$$F_1 = 1 \quad f_2 = F_0 + F_1 = 1 \rightarrow$$

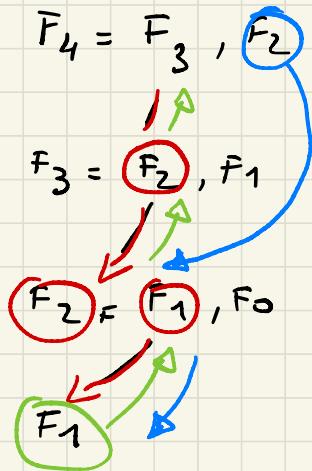
$$F_3 = F_2 + F_1 = 1 + 1 = 2$$

$$F_4 = F_3 + F_2 = 2 + 1 = 3$$

06/03/2023 | FIBONACCI |

ESEMPIO ALGORITMO ESPONENZIALE : (RICORSIVO)

SEQUENZA DI FIBONACCI (SUCCESSIONE)



Possibile dimostrare

$$F_m \approx 2^{0.694m}$$

Quanto impiega a fare la soluzione $T(m) = ?$

function fib(m)

if $m=0$: return(0)

if $m=1$: return(1)

return fib(m-1) + fib(m-2)

$$T(m) \leq 2, m \leq 1$$

$$T(m) = T(m-1) + T(m-2) + 3 \quad \text{for } m > 1$$

Relazioni di ricorrenza

N.B.

$$T(m) \geq F_m \rightarrow F_m \approx 2^{0.694m} \quad \text{ESPOENZIALE}$$

$T(m)$ esponenziale : brutte notizie

$$T(200) \geq F(200) \approx 2^{0.694 \cdot 200} = 138$$

- Prendiamo un NEL Fort Simulator ($\text{clock} = 40 \times 10^{12} \text{ sec}^{-1}$)
impiegherebbe 2^{92} sec

Calcolo-fib (m)

if $(m == 1) || (m == 2)$

$$\text{fib} = 1$$

else

for $\text{ultimo} = \text{penultimo} = 1, i = 3; (i \leq m); i++$

$$\{ \quad \text{fib} = \text{ultimo} + \text{penultimo}$$

$$\text{penultimo} = \text{ultimo}$$

$$\text{ultimo} = \text{fib}$$

}

return (fib)

$$T(m) = 1+1 = 2 = O(1) \quad \text{OTTIMO}$$

$$T(m) = 1+1 +$$

$$(m-2) \cdot (1+1+1+1+1) + 1 = 5m - 7 \quad O(m)$$

PESSIMO

ALGORITMI

RICORSIVI

- STRUTTURA GENERALE \rightarrow RICHIAMA SE STESSO

SOL_RISOLVI (PROBLEMA P)

SOLUZIONE s_1, \dots, s_m ;
if (semplice)

calcola direttamente le soluzioni di s_i di P
che

chiudi p in sottoproblemi dello stesso motivo
di P

$s_1 = \text{RISOLVI}(p_1), s_2, \dots, s_m = \text{RISOLVI}(p_m)$

combi per ottenere la soluzione s

return (s)

Complessità algorithmica ricorsiva

Non è definibile in forme chiuse perché spesso contiene il problema di tutti i problemi, ma in maniera induttiva tramite relazioni di ricorrenza.

- RELAZIONI DI RICORRENZA LINEARI

| FATTORINOMIA

↳ ORDINE COSTANTE k A COEFF. COSTANTI

- ↳ OMOGENEA
- ↳ NON OMOGENEA

$$T(m) = x_m$$

$$\begin{cases} x_m = \alpha_1 \cdot x_{m-1} + \alpha_2 \cdot x_{m-2} + \dots + \alpha_k \cdot x_{m-k}, & m \geq k \\ x_i = d; & 0 \leq i \leq k-1 \end{cases}$$

(si riesce a ottenere la forma chiusa)

$$\text{Se } k=1 \rightarrow x_m \begin{cases} d_0 = 0(1), \alpha_1 = 1 \\ d_0 \cdot \alpha_1^m = 0(\alpha_1^m), \alpha_1 > 1 \end{cases}$$

$$\begin{cases} x_m = \alpha_1 \cdot x_{m-1} \\ x_0 = d_0 \quad \rightarrow x_1 = \alpha_1 \cdot d_0 \\ x_2 = \alpha_1 \cdot x_1 = \alpha_1 (\alpha_1 \cdot d_0) \\ x_3 = \alpha_1 \cdot x_2 = \alpha_1 (\alpha_1 (\alpha_1 \cdot d_0)) \\ \alpha_1 < 1 \end{cases}$$

$$x_m = \sum_{j=1}^k c_j \cdot z_j^m = O\left(\max\{z_j^m \mid 1 \leq j \leq k\}\right)$$

$$x_m = c_1 \cdot z_1^m + c_2 \cdot z_2^m + \dots + c_k \cdot z_k^m$$

7/03/2023

(AUDIO 8)

$$z^k - \sum_{i=1}^k e_i \cdot z^{k-i} = 0$$

Polinomi di errore allo riconcavo

NON OMogenea

$$T(m) = x_m$$

$$\left\{ \begin{array}{l} x_m = a_1 \cdot x_{m-1} + a_2 \cdot x_{m-2} + \dots + a_k \cdot x_{m-k} + h \quad m \geq k \\ x_i = d_i \quad , \quad 0 \leq i \leq k-1 \end{array} \right.$$

$$y_m = x_m + h / \left(\sum_{j=1}^k a_j - 1 \right) \quad \text{come studio } y_m, \text{ che ha lo stesso ordine}$$

$$x_m = y_m - \frac{h}{\sum_{j=1}^k a_j - 1}$$

$$x_m = O(y_m)$$

ORDINE NON COSTANTE, VARIANO DI RECOMBINATION

$$T(m) = c \cdot T\left(\frac{m}{b}\right) + O(m^d) \quad c > 0, b > 1, \text{ and } d > 0$$

MASTER THEOREM (GOOGLE TRANSLATION)

$$T(m) = \begin{cases} O(m^d) & \text{if } d > \log_b c \\ O(m^d \cdot \log m) & \text{if } d = \log_b c \\ O(m \log_b c) & \text{if } d < \log_b c \end{cases} \quad] \text{ polinomiale.}$$

```
int calcolo_fibonacci_ricorsivo(int m)
```

{

```
    int fib;
```

```
    if ((m==1) || (m==2))
```

```
        fib = 1;
```

```
    else
```

```
        fib = calcolo_fibonacci_ricorsivo(m-1) + calcolo_fibonacci_ricorsivo(m-2);
```

```
    return (fib); /* next level */
```

}

$$\begin{cases} T(1) = 2 \\ T(2) = 2 \\ T(m) = m > 2 \end{cases}$$

↓

$$\begin{aligned} &= 1 + T(m-1) + T(m-2) + 1 \\ &= T(m-1) + T(m-2) + 2 \end{aligned}$$

$$x_m = \underset{1}{\alpha_1} \cdot \underset{1}{x_{m-1}} + \underset{1}{\alpha_2} \cdot \underset{1}{x_{m-2}} + 2$$

$$y_m = x_m + \frac{2}{1}$$

$$y_{m-2} = y_{m-1} - 2 + y_{m-2} - 2 + 2$$

$$x_m = y_m - \frac{2}{\sum_{j=1}^5 \alpha_j - 1}$$

$$y_m = y_{m-1} + y_{m-2}$$

$$z^k - \sum_{i=1}^k e_i \cdot z^{k-i} = 0$$

$$z^2 - z - 1 = 0$$

$$z_{1,2} = \frac{1 \pm \sqrt{5}}{2} \rightarrow \frac{1+\sqrt{5}}{2}$$

$$O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

$$\frac{F_n}{F_{n-1}} \xrightarrow{x \rightarrow \infty} \phi$$

int calculate_max_rec (int a[], int sx, int dx)

{
 int max, max1, max2;
 if (sx == dx)

 max = a[sx];
 else

 {
 max1 = calculate_max_rec (a, sx, (sx+dx)/2);
 max2 = calculate_max_rec (a, (sx+dx)/2+1, dx);

 if (max1 > max2)

 max = max1;

 else

 max = max2;

 }

} return(max);

$$T(1) = 2$$

$$T(m) = 1 + T\left(\frac{m}{2}\right) + 1 + T\left(\frac{m}{2}\right) + 1 + 2$$

$$= 2T\left(\frac{m}{2}\right) + 5 \rightarrow d=0 = O(1) \Rightarrow O(m^d)$$

$$\begin{aligned} a &= 2 \\ b &= 2 \end{aligned}$$

||

$$O(m^{\log_b a}) \Rightarrow O(m) \text{ LINEARE}$$

$$T(m) = a \cdot T\left(\frac{m}{b}\right) + O(m^d) \quad a > 0, b > 1, \text{ and } d \geq 0$$

MASTER THEOREM

$$T(m) = \begin{cases} O(m^d) & \text{IF } d > \log_b a \\ O(m^d \cdot \log m) & \text{IF } d = \log_b a \\ O(m^{\log_b a}) & \text{IF } d < \log_b a \end{cases} \text{ polynomial}$$

03/03/2023 (audio 7)

INT CALCOLA_FATT (int m)

{ int fatt;

if (m == 1)
fatt = 1;

else

fatt = calcola_fatt(m-1) * m;

$$\begin{aligned} m! &= 1 \cdot 2 \cdot 3 \cdot 4 \cdots m \\ m! &= m \cdot (m-1)! \end{aligned}$$

return(fatt);

}

$$\underline{T(1) = 1+1 = 2 = O(1)}$$

$$T(m) = 1 + T(m-1) + 1 \quad m > 2$$

$$\underline{T(m) = T(m-1) + 2} \quad \Delta$$

METODO DELLE SOSTITUZIONI

1) Ipotesi sul tempo di esecuzione

2) Dimostrazione per induzione

$$T(m) = 2 \cdot m - \text{IPOTESI INDUTRIS}$$

$$\begin{aligned}m &= 1 \\ T(m) &= 2\end{aligned}$$

OK!

$O(m)$

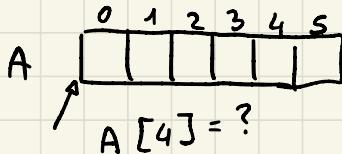
$m \geq 1 \quad T(m+1) ?$

$$T(m+1) = T(m) + 2$$

$$\begin{aligned}&\parallel \\ &= 2m + 2 \\ &= 2(m+1)\end{aligned}$$

ARRAY

- STRUTTURA OMogenea E STAtica
- ELEMENTI MEMORIZZATI CONSECUTIVAMENTE
- ACCESSO DIRETTO : LETTURA / SCRITTURA $O(1)$



$$x = A[4] = O(1)$$

$$A[4] = 3 \quad //$$

- VISITA

Dato A, attraversare tutti i suoi elementi esattamente 1 volta

- RICERCA

Dato A, X stabilire se x è contenuto in A, riportarne l'eventuale indice

- ORDINAMENTO (CHIAVE DI ORDINAMENTO)

determinare una permutazione

VISITA

void visita_array (int a [] , int m)

{

int i;

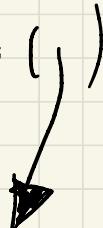
for (i = 0 ; (i < m) ; i ++)

elabora (a [i]);

}

$$T(m) = 1 + m(1 + d + 1) + 1 = (d+2)m + 2 = O(m)$$

ALGORITMO BINARIA O DICOTOMIA ()



13/03/2023 (AUDIO 9)

Se uno ordinamento per ricerca si fa meglio di $O(n)$
(diminuzione spazio di ricerca)

```
int ricerca_binaria_array ( int a [], int m, int valore )  
{
```

int sx, dx, mx;

for ($sx = 0, dx = m-1, mx = (sx+dx)/2$; ($sx <= dx$) && ($c[mx] != 0$))
 $; mx = (sx+dx)/2$)

If ($e[mx] > value$)

$$dx = mx - 1;$$

$$Sx = mx + 1 ;$$

```
return ((sx <= dx) ? mx : -1);
```

3

VALUTAZIONE COMPLESSITÀ

$$T(m) = O(1) \quad \text{caso ottimo}$$

$$T(m) = 1 + k \left(1 + 1 + 1 + 1 \right) + 1$$

$$= 4K+2 = 4 \log_2 m + 2 = O(\log m)$$

Quanto vale K?

① $\frac{m}{2}$

② $\frac{m}{2^2}$

③ $\frac{m}{2^K} = 1 \rightarrow K = \log_2 m$

ALGORITMI DI ORDINAMENTO

Contengono dati (ovvero chiavi) → chiavi primarie, univoca
Le sono composte: → chiavi secundarie, sottellisti
→ nome, cognome, ecc.

→ multivale

→ chiavi primarie, univoca

→ chiavi secundarie, sottellisti

→ multivale
monovale in
una singola
cella di un
array

STABILITÀ → Computero algoritmi stabili

Lo non altera ordine relativo ad altre chiavi **NON SI INCROCIANO**

Lo evita interazione con algoritmi di ordinamento precedenti, bontà su altre chiavi
SI INCROCIANO

1A|5B|4F|5B|6C|



STABILE

(SONO CORRETTI ESEMPI)

1A|4F|5B|5D|6C|



NON STABILE

H.B

1A|4F|5D|5B|6C|

-CLASSE DI COMPLESSITÀ

-STABILITÀ

-ORDINAMENTO SUL POSTO

→ CARATTERISTICHE PRINCIPALI ALGORITMI DI ORDINAMENTO

↳ Hanno bisogno di altre risorse per ordinare quel array (non sul posto)

↳ ESEGUE operazioni solo sull'array allocato (sul posto)

ALGORITMI DI CONFRONTO

- ↳ LIMITE INFERIORE $T(m) = \Omega(m \log m)$
- ↳ LIMITE SUPERIORE $T(m) = O(m^2)$

ESISTONO ANCHE ALGORITMI NUOVI DI CONFRONTO CHE DIPENDONO SULLE PROPRIETÀ DEGLI INTRATTI DATI ALLE CELLE DEGLI ARRARI (DATI DINAMICI) $T(m) = O(m)$

Poiché ordinare?

↳ RICERCA PIÙ EFFICIENTE

ESEMPIO APPLICATIVO

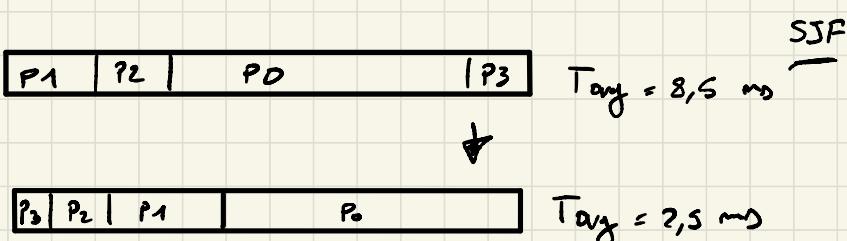
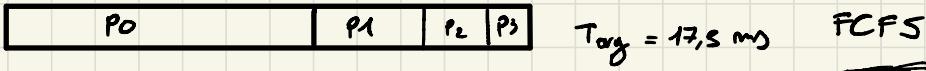
- ↳ Indice inverso dei motori di ricerca
- ↳ Hit = (parole, Frequenza, documenti) → TUPPA
- ↳ Ordine lexicografico nelle diverse parole

SCHEDE DI DEDICAZIONE DELLA CPU : SJF VS FCFS

SJF = SHORTEST JOB FIRST → FAI IL LAVORO PIÙ BREVE

FCFS = FIRST COME FIRST SERVE → PRIMO CHE ARRIVA IL PRIMO SERVITO

TEMPO MEDIO DI ATTESA



Efficiente

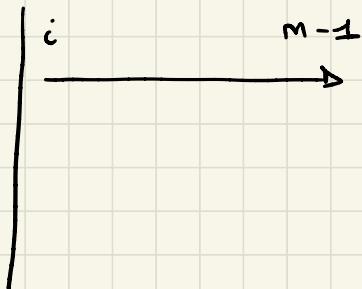
1° ALGORITMO DI ORDINAMENTO INSERT (ION) SORT

SCATTARE FOTO AL GENERICO PASSO GENERICO ierarca uomo le caratteristiche comuni:
La sequenza di destinazione ordinata.

SEQUENZA DI DESTINAZIONE

$e[0], \dots, e[i-1]$

SEQUENZA DI ORIGINE



Vocel insertsort (int e[], int m)

{

int value_ins, i, j;

for (i = 1; (i < m); i++)

{

for (value_ins = e[i], j = i - 1; ((j >= 0) && (e[j] > value_ins));
j--)

$e[j+1] = e[j];$

if (j + 1 != i)

$e[j+1] = value_ins;$

}

}

7 3 5 57 12
↑ ↑
↓ ↓

3 7 5 57 12
↓ ↓

14/03/2023 (AUDIO 10)

COMPLESSITÀ ALGORITMO

CASO OTTIMO (array già ordinato)

$$T(m) = 1 + (m-1) \left(1+1+0 \cdot (1+1+1) + 1+1+1+1 \right) + 1 \\ = 5m - 3 = O(m)$$

CASO PESSIMO (array inversamente ordinato)

$$T(m) = 1 + \sum_{i=1}^{m-1} (1+1+i(1+1+1)+1+1+1+1) + 1 \\ = 2 + 6(m-1) + 3 \sum_{i=1}^{m-1} i \\ = 2 + 6(m-1) + 3 \left(\frac{m(m-1)}{2} \right) \\ = 1,5m^2 + 4,5m - 4 = O(m^2)$$

ALGORITMO STABILE

ALGORITMO OPERA SUL POSTO

2° ALGORITMO DI ORDINAMENTO

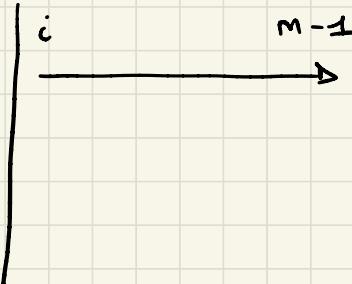
SELECT (ION) SORT

SCATTARE FOTO AL GENERICO PASSO GENERICO chiamato nome le caratteristiche comuni:
 La SEQUENZA DI DESTINAZIONE ORDINATA

SEQUENZA DI DESTINAZIONE

 $a[0], \dots, a[i-1]$

SEQUENZA DI ORIGINE



void selectsort (int a[], int m)

{

int valore_min, indice_valore_min, i, s;

for (i=0 ; (i < m-1) ; i++)

{
 for (valore_min = a[i], indice_valore_min = i, J = i+1 ;
 (J < m) ; J++)
 }

{
 if (a[s] < valore_min)
 {
 valore_min = a[s];
 indice_valore_min = s;
 }
 }

if (indice_value_min != i)

{

$$a[\text{indice_value_min}] = a[i];$$

$$a[i] = \text{value_min};$$

}

}

}

COMPLESSITÀ

$$h = \{1, 3\}$$

$$\begin{aligned} T(m) &= 1 + \sum_{i=1}^{m-1} (1+1+(m-i))(1+h+1)+1+h+1+1 \\ &= 2 + (h+4)(m-1) + (h+2) \cdot m (m-1) - (h+2) \sum_{i=1}^{m-1} i \\ &= (h+2)m^2 + 2m - (h+2) - (h+2) \frac{(m-1)m}{2} \\ &= (0,5h+1)m^2 + (0,5h+3)m - (h+2) = O(m^2) \end{aligned}$$

ALGORITMO

STABILE

ALGORITMO

OPEN SUL POSTO

3° ALGORITMO DI ORDINAMENTO

BUBBLE SORT

(DA NON SCEGLIERE)

(AUDIO 11)

void bubble sort (int a[], int n)

{

int temp, i, j;

for (i=1; (i < n); i++)

{

for (j=n-1; (j >= i); j--)

{

if (a[j] < a[j-1])

{

tmp = a[j-1];

a[j-1] = a[j];

a[j] = tmp;

}

}

{ }
{ }

[7|2|4|5|3|1]

[4|7|2|4|5|3]

2 7

[4|2|7|4|5|3]

[1|2|3|7|4|5]

4 7

[1|2|3|4|7|5]

5 7

[1|2|3|4|5|7]

COMPLESSITÀ ASINTOTICA

$n = \{1, 4\}$

$$\begin{aligned}
 T(n) &= 1 + \sum_{i=1}^{n-1} \left(1 + 1 + (n-i) (1+h+1) + 1 + 1 \right) + 1 \\
 &= 2 + 4(n-1) + (n-1)(h+2) \cdot n - (h+2) \sum_{i=2}^{n-1} i \\
 &= (h+2)n^2 + (2-h)n - 2n \frac{(n-1)}{2} = O(n^2)
 \end{aligned}$$

ALGORITMO

STABILE

ALGORITMO

OPERA SUL PUNTO

20/03/2023 (AUDIO 12)

MERGE SORT

- Dividi in 2 parti delle stesse dimensione (divide) $\begin{matrix} & \\ & \end{matrix}$ 1°
- Ri-applica lo stesso algoritmo $\begin{matrix} & \\ & \end{matrix}$ 2°
- Fondi ordinatamente le 2 parti (impara) $\begin{matrix} & \\ & \end{matrix}$ 3°

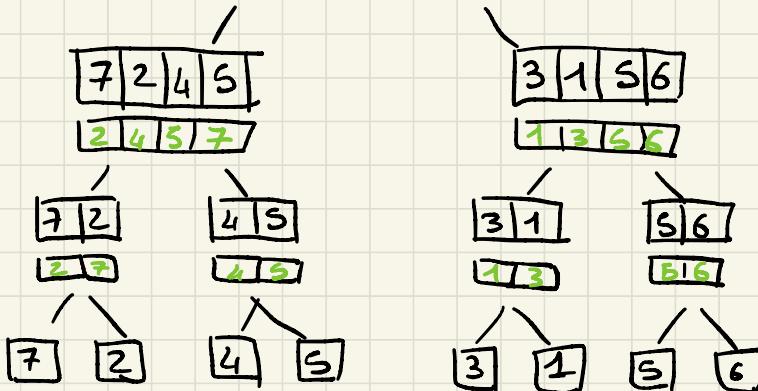
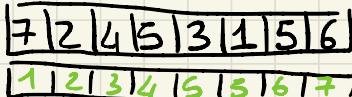
DIVIDI ET IMPERA \rightarrow ADATTO AGLI ALGORITMI RICORSIVI

FUSIONE CONNESSA

$$\square \square = \square$$

$$\square \square \square = \square \square \square$$

ALBERO RICORSIONI



```
void mergesort ( int a[], int sx, int dx ) {
```

```
    int mnx ;
```

```
    if ( sx < dx ) {
```

```
        mnx = (sx+dx)/2 ;
```

```
        mergesort ( a, sx, mnx ) ;
```

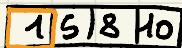
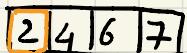
```
        mergesort ( a, mnx+1, dx ) ;
```

```
        fusioni ( a, sx, mnx, dx ) ;
```

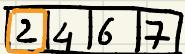
```
}
```



FUSIONE



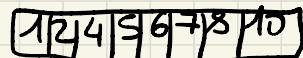
→



→



→



```
void fondu ( int a[], int sx, int mx, int dx ) {
```

```
    int *b, i, s, k;
```

```
    b = (int *) malloc( dx+1, sizeof(int) );
```

```
    for ( i=sx, j=mx+1, k=0; ((i<=mx)&&(j<=dx)); k++ )
```

```
        if ( a[i] <= a[j] ) {
```

```
            b[k] = a[i];
```

```
            i++;
```

```
}
```

```
        else {
```

```
            b[k] = a[j];
```

```
            j++;
```

```
}
```

```
    while ( i <= mx ) {
```

```
        b[k] = a[i];
```

```
        i++;
```

```
        k++;
```

```
}
```

```
    while ( j <= dx ) {
```

```
        b[k] = a[j];
```

```
        j++;
```

```
        k++;
```

```
}
```

```
    for ( k=sx ; ( k <= dx ) ; k++ )
```

```
        a[k] = b[k-sx];
```

```
    free(b);
```

```
3
```

$$T(1) = 1$$

$$T(m) = 2 + \left\lceil \frac{m}{2} \right\rceil + O(m^d)$$

$$\begin{aligned} a &= b = 2 \\ d &= 1 \end{aligned}$$

$$O(m^d \cdot \log m) \Rightarrow O(m \cdot \log m)$$

prende lineare

Th. esperto

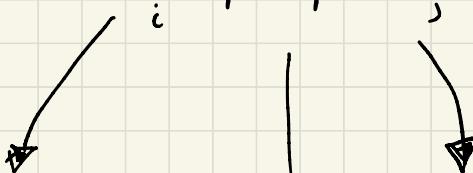
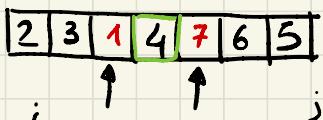
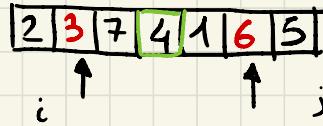
STABILE

NON SUL POSTO

Quick sort (audio 13)

scegliere un elemento dell'array (pivot)

PARTIZIONAMENTO



2, 3, 1

1, 2, 3
↑ $\boxed{1} \rightarrow$ vettore
indice

7, 6, 5

5, 6, 7

void quicksort (int a[], int sx, int dx) {
 int pivot, tmp, i, j;

for (pivot = a[(sx+dx)/2], i = sx, j = dx; (i < j);)

while (a[i] < pivot)

i++;

while (a[j] > pivot)

j--;

if (i <= j) {

if (i < j) {

tmp = a[i];

a[i] = a[j];

a[j] = tmp;

}

i++;
 j--;

}

if (sx < j)

quicksort (a, sx, j);

/* TRIPARTIZIONE */

O(m)

TRIPARTIZIONE

RICORSIONE

if ($i < dx$)

quick sort (a, i, dx);

}

partizionamento bilanciato

$T(1) = 11$

$$T(m) = T(m_1) + T(m_3) + O(m)$$

caso peggiore $O(m^2) = m + m-1 + m-2 + \dots + 1 =$

$$= \frac{m(m-1)}{2}$$

caso ottimale elemento mediano come pivot

$$T(m) = T\left(\frac{m}{2}\right) + T\left(\frac{m}{2}\right) + O(m^1)$$

th. master
 $\Rightarrow T(m \cdot \log m)$

caso medio $T(m) = O(m \cdot \log m)$

SCEGLIEMO IL CASO IN CUI IL PIÙ STABILE DIVENTA UN 90% E UN

$$90\% - 10\%$$

1°

$$m \cdot \frac{9}{10}$$

2°

$$m \cdot \left(\frac{9}{10}\right)^2$$

3°

$$m \cdot \left(\frac{9}{10}\right)^3$$

⋮

$$k \left[m \cdot \left(\frac{9}{10}\right)^k \right] = 1 \quad k = \log_{\frac{10}{9}} (m)$$

NON STABILE

SUL POSTO

HEAP SORT

$$h_1, h_2, h_3, \dots, h_m$$

rimuovo $\forall i : 1, \dots, \frac{m}{2} \rightarrow h_i \geq h_{2i}$



$h_i \geq h_{2i+1}$

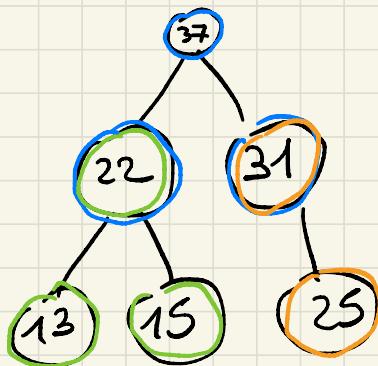
MAX-HEAP

MIN-HEAP

37, 22, 31, 13, 15, 25
h₁ h₂ h₃ h₄ h₅ h₆

→ 37 MAX

STRUTTURA AD ALBERO
GERARCHIA Relazioni



HEAP NEGLI ARRAY NON USIAMO L'INDICE 0 QUINDI ACCORDIAMO
ARMAI' DA m+1 Elementi ↗



1° cosa PERMUTA L'ARRAY IN UN HEAP

27/03/2023

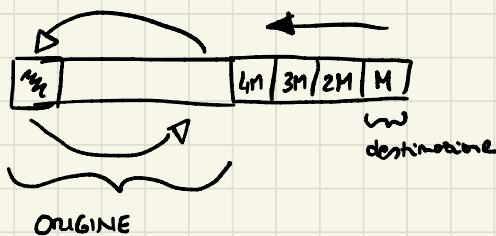
(AUDIO 14)

HEAP-SORT (ITERATIVO)

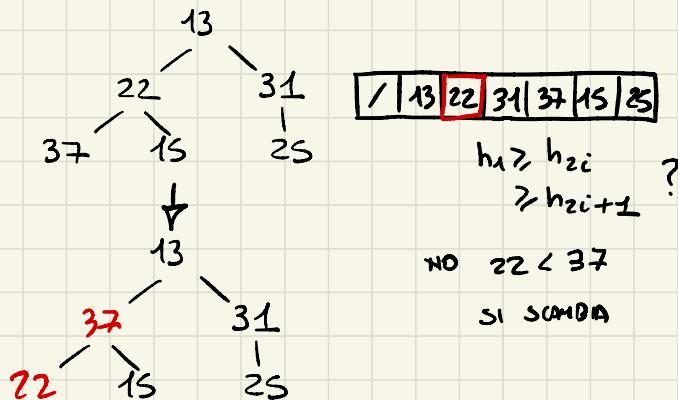
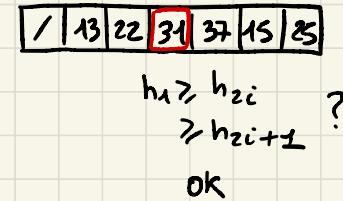
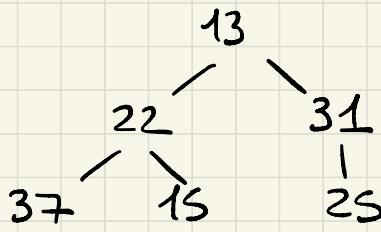
$$h_i \geq h_{2i} \rightarrow i = 1, \dots, \frac{M}{2}$$

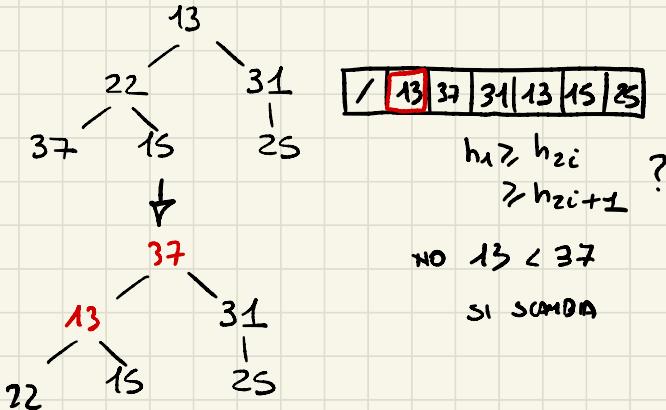
$$h_i \geq h_{2i+1}$$

- ① permutare l'array (i suoi elementi) \rightarrow nuova disposizione continua in heap
- ② $a[1], \dots, a[i-1]$ organizzate ad heap
- ③ somma con $a[i-1]$



COSTRUZIONE HEAP





```
void heap_sort (int a[], int m) {
```

```
    int tmp, dx, sx;
```

```
    for (sx = m/2; (sx >= 1); sx--)
```

```
        setaccia_heap (a, sx, m);
```

```
    for (dx = m; (dx > 1); dx--) {
```

```
        tmp = a[1];
```

```
        a[1] = a[dx];
```

```
        a[dx] = tmp;
```

```
        setaccia_heap (a, 1, dx-1);
```

```
}
```

```
}
```

void setmax_heap (int a[], int sx , int dx) {

int max_value , i , j ;

for (max_value = a [sx] , i = sx , j = 2 * i ; (j <= dx) ;) {

if ((j < dx) && (a [j + 1] > a [j]))

j++;

if (max_value < a [j]) {

a [i] = a [j] ;

i = j ;

j = 2 * i ;

3

else

j = dx + 1 ;

3

if (i != sx)

a [i] = max_value ;

3

$$T(n) \leq 1 + \frac{m}{2} \left(1 + \log_2 m + 1 \right) + 1 + 1 + (m-1) (1 + 3 + \log_2 m + 1) + 1 \rightarrow T(n) = 1.5 m \log_2 m + 6m + -\log_2 m - 1 = O(m \cdot \log m)$$

NON STABILE

OPERA SUL PUNTO

STRUTTURE DATI DINAMICHE

(AUDIO 15)

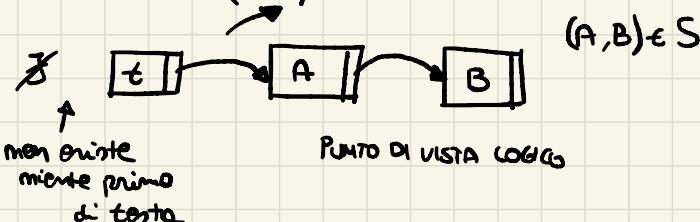
28.03.2023

LISTA → FORMALE $\rightarrow L = (E, t, S)$

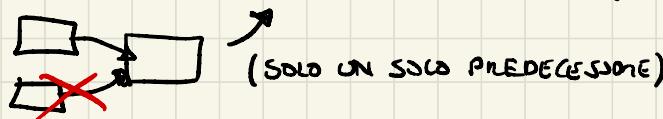
→ proprietà:

- E : insieme di elementi
- $t \in E$: testo
- $S \subseteq E \times E$ (relazione binaria)

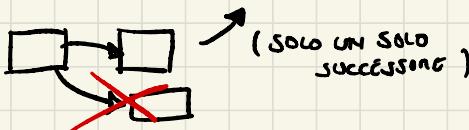
① $\forall e \in E \ (e, t) \notin S$



② $\forall e \in E, \text{ se } e \neq t \Rightarrow \exists \text{ unico ed solo } e' \in E : (e', e) \in S$



③ $\forall e \in E, \exists \text{ al più un } e' \in E : (e, e') \in S$



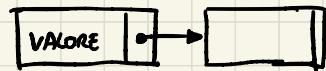
④ $\forall e \in E, \text{ se } e \neq t \Rightarrow e \text{ è raggiibile da } t$

(se conosco t riesco ad arrivare ad e tramite il concatenamento di 2 e 3)

IN C → typedef struct elem_liste {

```
int value;  
struct elem_liste * succ_P;
```

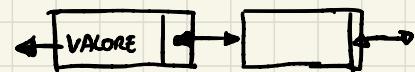
} elem_liste_t



typedef struct elem_liste_dc {

```
int value;  
struct elem_liste_dc * succ_P,  
* prec_P;
```

} elem_liste_dc_t



16/03/2023 (AUDIO 16)

LISTA → PROBLEMA DELLA VISITA

```
void visita_lista ( elem_liste_t * testo_p ) {  
    elem_liste_t * elem_p;  
    for ( elem_p = testo_p; ( elem_p != NULL ); elem_p =  
          elem_p -> succ_p )  
        elabora ( elem_p -> valore );  
}
```

$$T(m) = 1 + m(1 + d + 1) + 1 = (d+2)m + 2 = O(m)$$

PROBLEMA DELLA RICERCA

```
elem_liste_t * cerca_circa_lista ( elem_liste_t * testo_p, char valore ) {  
    elem_liste_t * elem_p;  
    for ( elem_p = testo_p; (( elem_p != NULL ) &&  
          ( elem_p -> valore != valore ));  
         elem_p = elem_p -> succ_p );  
    return ( elem_p );  
}
```

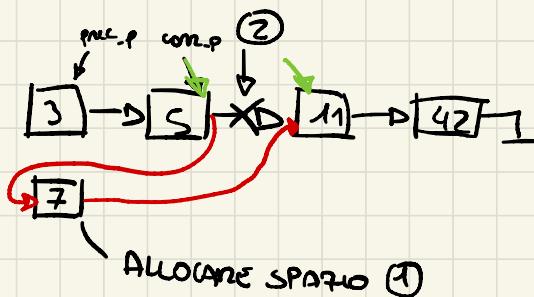
$$T(m) = 2 \quad \text{CASO OTTIMO}$$

$$T(m) = 1 + m(1+1) + 1 = 2m + 2 \quad O(m) \text{ CASO PESSIMO}$$

PROBLEMA DELL'INSERIMENTO

3, 5, 11, 42

↑ ??



ipotesi: nona duplicità

int inserisci_in_liste_ordinata (elem_lista_p ** testa_p , int value) {

int currento ;

elem_lista_p * corr_p , * prec_p , * muove_p ;

↓
cominciate

for (corr_p = prec_p = * testa_p ; ((corr_p != NULL) &&
(corr_p -> value < value)) ;

prec_p = corr_p , corr_p = corr_p -> succ_p);

if ((corr_p != NULL) && (corr_p -> value == value))

currento = 0 ;

else {

currento = 1 ;

muove_p = (elem_lista_t *) malloc (

size of (elem_lista_t));

muove_p -> value = value ;

$\text{muvole_p} \rightarrow \text{succ_p} = \text{curr_p};$

if ($\text{curr_p} == * \text{testo_p}$)

$* \text{testo_p} = \text{muvole_p};$

else

$\text{prec_p} \rightarrow \text{succ_p} = \text{muvole_p};$

}

return (l'menito)

}

$T(m) = 4$ CASO OTTIMO

$T(m) = 1 + n(1+1) + 1 + 7 = 2m + 9 = O(n)$

PROBLEMA DECLARA RIMOZIONE

int rimuovi_da_liste_ordinate (elam_list_t **testo_p, int value) {

int rimova;

elam_list_t *curr_p, *prec_p;

for ($\text{curr_p} = \text{prec_p} = * \text{testo_p}; (\text{curr_p} != \text{NULL}) \&\&$

$\text{curr_p} \rightarrow \text{value} < \text{value}$);

$\text{prec_p} = \text{curr_p}, \text{curr_p} = \text{curr_p} \rightarrow \text{succ_p};$

if ($(\text{curr_p} != \text{NULL}) \&\& (\text{curr_p} \rightarrow \text{value} > \text{value})$)

rimova = 0;

else

rimono = 1;

if (cor_P == * teste_P)

* teste_P = cor_P \rightarrow succ_P ;

else

prec_P \rightarrow succ_P = cor_P \rightarrow succ_P ;

free (cor_P) ;

}

Return (rimono);

}

$T(m) = 4 = O(1)$ CASO OTTIMO

$T(m) = 1 + m(1+1) + 1 + 5 = 2m + 7 = O(m)$ CASO PESSIMO

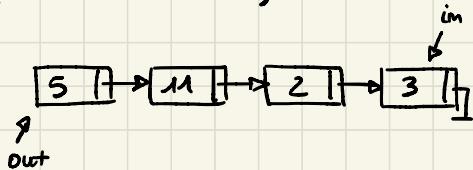
PILE E CODE

PARTICOLARI TIPI DI LISTE

CODE

FIFO

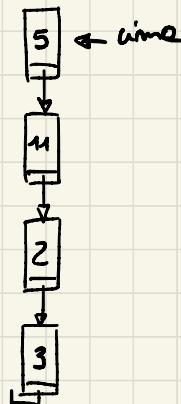
(First in First Out)



PILE

LIFO

(Last in first out)



INSERIMENTO CODE

```

void metti_in_code ( elem_linta_t **uscita_p, elem_linta_t **inizio_p,
                     int value) {
  
```

```

elem_linta_t * nuovo_p;
  
```

```

nuovo_p = ( elem_linta_t *) malloc ( sizeof ( elem_linta_t ) );
  
```

```

nuovo_p -> value = value;
  
```

```

nuovo_p -> succ_p = NULL;
  
```

```

if ( * inizio_p != NULL )
  
```

```

( * inizio_p ) -> succ_p = nuovo_p;
  
```

else

* uscita-p = mese-p;

* ingresso-p = mese-p;

}

ROTAZIONE CODE

elem-linta + * toglie-de-coda (elem-linta + ** uscita-p,
elem-linta + ** ingresso-p) {

elem-linta + * elem-p;

elem-p = * uscita-p;

if (* uscita-p != NULL) {

* uscita-p = (* uscita-p) -> succ-p;

if (* uscita-p == NULL)

* ingresso-p = NULL;

}

return (elem-p);

}

PILE INSERIMENTO

```
void metti_su_pila ( elem_lista_t **cima_p, int value ) {  
  
    elem_lista_t *nuova_p;  
  
    nuova_p = ( elem_lista_t * ) malloc( sizeof( elem_lista_t ) );  
    nuova_p -> value = value ;  
  
    nuova_p -> succ_p = *cima_p ;  
  
    *cima_p = nuova_p ;  
  
}
```

$$T(n) = \gamma = O(1)$$

PILE RIMOZIONE

```
elem_lista_t *tglv_de_pila ( elem_lista_t **cima_p ) {  
  
    elem_lista_t *elem_p;  
  
    elem_p = *cima_p ;  
  
    if ( *cima_p != NULL )  
        *cima_p = (*cima_p) -> succ_p ;  
  
    return ( elem_p );  
  
}
```

$$T(n) = O(1)$$

ALBERI → DEFINIZIONI E PROBLEMI



$$T = (N, r, B)$$

$\rightarrow N$ = insieme di nodi

r = (radice) appartenente a N

B = relazione binaria su N t.c.:
(Padre \rightarrow figlio)

① $\forall m \in N, (n, r) \notin B$ (radice unica per poche)

② $\forall m \in N$, se $m \neq r$ allora $\exists m' \in N$ tale che

$$(m', m) \in B$$

(Padre unico)

③ $\forall m \in N$, se $m \neq r$

Qualunque elemento dell'albero c'è raggiungibile
dalla radice

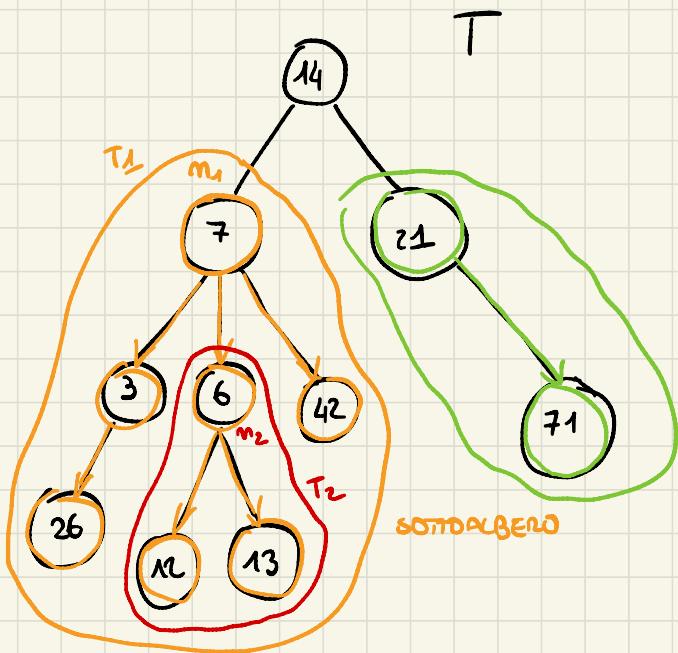
④ $\forall m \in N$, se $m \neq r$

17/04/2023 AUDIO (18)

LA LISTA È UN ALBERO PARTICOLARE SENZA DOPPIAZIONI

DATI $T = (N, r, B)$, $m \in N$

- SOTTO ALBERO $\rightarrow T' = (N', m, B')$



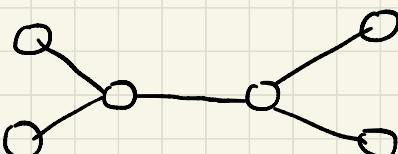
Abbiamo due sottoalberi T_1, T_2 generati da m_1, m_2



$N_1 \cap N_2 = \emptyset$ oppure $N_1 \subseteq N_2$ oppure $N_2 \subseteq N_1$

ALBERO LIBERO (SENZA RADICE)

- ANTRIFLESSIVA
- SIMMETRICA
- CONNESSIONE
- OCCLICITÀ



Grado di un albero :

$$d(T) = \max_{m \in N} |\{m' \in N \mid (m, m') \in B\}| \rightarrow \begin{array}{l} \text{massimo numero di figli} \\ \text{di un nodo} \end{array}$$

↓
degree

Altezza di un albero :

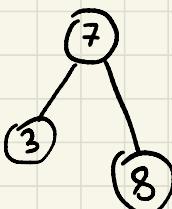
$$h(T) = \max \{i \in N \mid \exists m \in N, m \in \text{livello } i\} \rightarrow \begin{array}{l} \text{massimo numero di nodi} \\ \text{che si ottengono andando} \\ \text{dalla radice r alla foglia} \\ \text{più distante} \end{array}$$

Lunghezza di un albero :

$$b(T) = \max_{1 \leq i \leq h(T)} |\{m \in N \mid m \in \text{livello } i\}| \rightarrow \begin{array}{l} \text{max numero di nodi} \\ \text{nello} \\ \text{livello} \end{array}$$

$$\begin{matrix} d > 1 \\ h \end{matrix} \rightarrow m$$

$$m(d, h) = \sum_{i=0}^{h-1} d^i = \frac{d^h - 1}{d - 1}$$



ALBERI BINARI

$$B = B_{sx} \cup B_{dx}$$

$$B_{sx} \cap B_{dx} = \emptyset$$

$$\forall m, m_1, m_2 \in N \text{ se } (m, m_1) \in B_{sx} \text{ (risp. } B_{dx}) \text{ e } (m, m_2) \in B_{dx} \text{ (risp. } B_{sx}) \Rightarrow m_1 = m_2$$

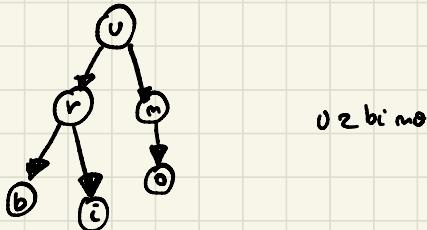
TRASFORMAZIONE FRATELLI - FIGLI

```
type def struct modo_albero_bin {  
    int value  
    struct modo_albero_bin *sx_p, *dx_p;  
}; modo_albero_bin_t
```

18/04/2023 (audio 13)

Problema vintre

- Anticipato
- Simmetrico
- Posticipato



```
void vintre_albero_bin_out (modo_albero_bin_t *modo_p) {
```

{

```
if (modo_p != NULL) {
```

```
    elabore (modo_p -> value);
```

```
    vintre_albero_bin_out (modo_p -> sx_p);
```

```
    vintre_albero_bin_out (modo_p -> dx_p);
```

}

```
void vintre_albero_bin_simm (modo_albero_bin_t *modo_p) {
```

```
if (modo_p != NULL) {
```

```
    vintre_albero_bin_simm (modo_p -> sx_p);
```

```
    elabore (modo_p -> value);
```

winita_olhos - bin - nim (modo_p - p cix - p);

}

3

void winita_olhos_bin_port (modo_olhos_bin_t *modo_p) {

if (modo_p != NULL) {

winita_olhos_bin_port (modo_p - p Sx_p);

winita_olhos - bin - port (modo_p - p Cx_p);

elolhos (modo_p -> value);

}

d = constante

$$T(m) = 1 + T(k) + T(m-k-1) + d$$

$$T(0) = 1$$

$$= T(k) + T(m-k-1) + (d+1)$$

$$T(m) = (d+2) \cdot m + 1 \quad \text{IP. INDUTIVA}$$

$$m=0$$

$$T(m) = 1$$

$$0 \leq m \leq m-1 \rightarrow T(m') = (d+2)m' + 1$$

$$T(m) = (d+2)k + 1 + (d+2)(m-k-1) + 1 + (d+1)$$

$$= (d+2)(m-1) + d + 3 =$$

$$= (d+2)m - d - 2 + d + 3$$

$$= (d+2)m + 1 \quad O(m)$$

modo_olbene_bin_t * cerca_in_olbene_bin_out (modo_olbene_bin_t * modo_p,
int value) {

modo_olbene_bin_t * modo_ris_p;

if ((modo_p == NULL) || (modo_p->value == value))
modo_ris_p = modo_p;

else {

modo_ris_p = cerca_in_olbene_bin_outc (modo_p->sk_p,
value);

if (modo_ris_p == NULL)

modo_ris_p = cerca_in_olbene_bin_outc (modo_p->dk_p,
value);

}

return (modo_ris_p);

3

$T(n) = O(n)$ per visita

$O(1)$

modo_olbene_bin_t * cerca_im_olbene_bin_rc (modo_olbene_bin_t * modo_p,
int value) {

modo_olbene_bin_t * modo_p

for (modo_p = racice_p; ((modo_p != NULL) && (modo_p->value != value));

modo_p = (value < modo_p->value) ?

modo_p->sk_p;
modo_p->dk_p;

return (mod-p);

3

Costruzione albero binari

① Generazione numeri casuali

13, 2, 8, 15, 20

Algoritmo di inserimento

28/04/2023

(AUDIO 20)

ALGORITMO ALBERO BINARIO → RIMOZIONE

- RIMOZIONE FOGLIA
- RIMUovere CON UN FIGLIO
- RIMUovere CON 2 FIGLII

INT Rimuovi_de_albero_bin_rec (mode_albero_bin_rec *root ,
int valore) {

```
int rimuono ;  
mode_albero_bin_rec * mode_p , * padre_p , * sest_p ;
```

```
if ( mode_p = poche_p = * root == NULL ;  
(( mode_p != NULL ) && ( mode_p -> valore != valore )) ;  
poche_p = mode_p , mode_p = ( valore < mode_p -> valore ) ?
```

mode_p -> dx_p ;
mode_p -> sx_p ;

```
if ( mode_p == NULL )
```

```
rimuono = 0 ;
```

```
else
```

```
{
```

```
rimuono = 1 ;
```

```
if ( mode_p -> sx_p == NULL )
```

```
{
```

```
if ( mode_p == * root == poche_p )
```

```
* root == mode_p -> dx_p ;
```

```
else
```

```
if ( valore < poche_p -> valore )
```

```
poche_p -> sx_p = mode_p -> dx_p ;
```

```
else
```

```
poche_p -> dx_p = mode_p -> dx_p ;
```

3

```
close
```

{

if (mode_p → dx_p == NULL)

{

if (mode_p == * node_p)

* node_p = mode_p → sx_p;

else

if (value < node_p → value)

node_p → sx_p = mode_p → sx_p;

else

node_p → dx_p = mode_p → dx_p;

}

else

{

sort_p = mode_p ;

for (node_p = sort_p , mode_p = sort_p → sx_p ;

(mode_p → dx_p != NULL);

node_p = mode_p , mode_p = mode_p → dx_p);

sort_p → value = mode_p → value ;

if (node_p == sort_p)

node_p → sx_p = mode_p → sx_p ;

else

node_p → dx_p = mode_p → sx_p ;

}

free (mode_p);

}

}

return (nimono);

}

COMPLESSITÀ

dipende dal caso für

- caso ottimo : $T(n) = O(1)$
- caso pessimo : $T(n) = O(n)$
- caso medie : $T(n) = O(\log n)$

BILANCIAMENTO

SI PUÒ FARE , PERO' COSTA MOLTO

2/OS (AUDIO 21)

ALBERI BINARI DI RICERCA ROSSO-NERO

Terme: $h \leq 2 \log_2(m+1)$

↳ CASO PEGGIOR

CASE' UN LEVA

ALBERI RN \rightarrow ric, ins, rim. $\rightarrow T(m) = O(\log m)$

TABELLE HASH (HASH "SPECIALE")

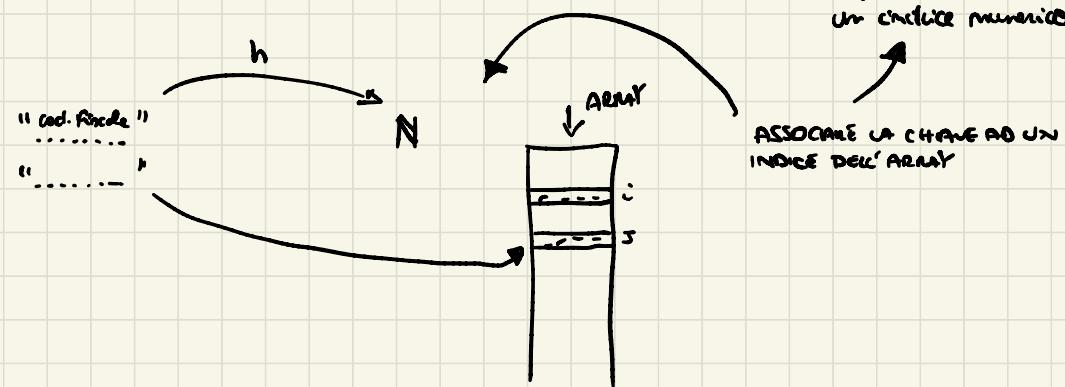
↳ METODI DATI IN UN ARRAY \rightarrow MANIERA DINAMICA \rightarrow DIZIONARIO

LISTA
ALBERI

↓
ARRAY ASSOCIAZIO

indice non più un intero

\hookrightarrow trasformare una chiave in un indice numerico



DIZIONARI

E:

↳ COSTITUITO DA DUE PARTI \rightarrow CHIAVE (GRUPPO DI CAMMINI) K_i

\rightarrow INFORMAZIONE ASSOCIAZIA ALLA CHIAVE I_i

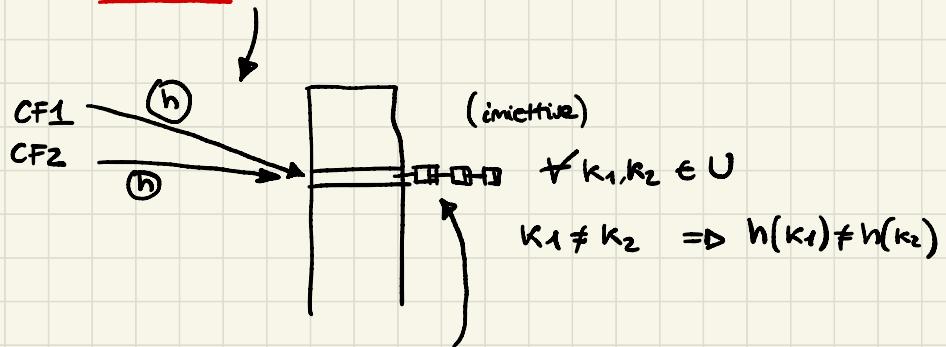
SCOPO CHIAVE ACCEDERE A TUTTE LE INFORMAZIONI ASSOCIAZIE A QUESTA CHIAVE

RIC. ELEMENTO E_i CON CHIAVE $K_i \rightarrow$ h TAN. FUNZIONE

↳ MEDIANANTE SI PUÒ ESEGUIRE LE OPERAZIONI RICERCA, RM, INSERIMENTO, MINIMA CON COSTANTE $O(1)$

PROGETTARE FUNZIONE HASH

- 1) h deve essere calcolabile in modo efficiente (FACILE DA ACCRESCERE)
- 2) Deve limitare le collisioni



Gli elementi che collisionano uniscono una lista che le unisce per contenere i vari elementi (LISTE DI COLLISIONE)

PRINCIPIO DI UNIFORMITÀ (RISULTANO TUTTE LE CHIAVI CON LA STESSA PROBABILITÀ E ANCHE CHE VENGONO ASSEGNAZIONI CON LA STESSA PROBABILITÀ NELL'INTERO DELLA)

$$T(m) = h + O(m) \quad \text{CASO PESSIMO}$$

$$T(m) = O(1) + T\left[\begin{array}{l} \text{* VERGONO SULLE} \\ \text{lunghezza lista } T(h|k) \end{array}\right] \rightarrow O(1+\alpha)$$

Th. 1 sento succoso

\downarrow

CASO MEDIO

Th. 2 com succoso m elementi

$O(1+\alpha)$ numero di righe tabella

FATT. DI CARICO

$\alpha = \frac{m}{n}$

α è collegato al Tempo medio

Se m è proporzionale ad n , allora $m = O(n)$

Quindi: $\lambda = \frac{m}{n} = \frac{O(n)}{n} = \frac{C \cdot \alpha n}{n} = O(1)$

Cosa ricava allora $\rightarrow O(1+\lambda) = O(1)$

RISOLUZIONE COLLISIONE CON INDIREZIONE APERTA



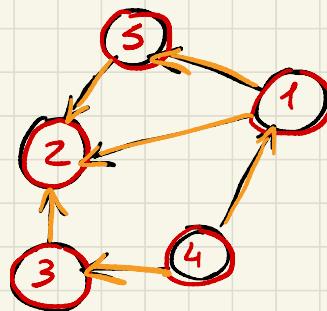
INDIREZIONE APERTA

05/05/2023

(AUDIO 23)

GRAFI

- UNA COPPIA $G = (V, E)$
- Diretti (ORIENTATO)
- INDIRETTI

V = insieme verticiE = relazione binaria su V

$$V = \{1, 2, 3, 4, 5\}$$

$$E = \begin{cases} (1, 5) \\ (1, 2) \\ (4, 1) \\ \vdots \end{cases}$$

Sia $G = (V, E)$ un grafo diretto

$G' = (V', E')$ è:

- un sottografo di G : $V' \subseteq V$
 $E' \subseteq E \cap (V' \times V')$

- un sottografo indotto di G : $V' \subseteq V$
 $E' = E \cap (V' \times V')$

- un grafo trasposto di G se: $V' = V$

$$E' = \{(v', v) \mid (v, v') \in E\}$$

GRADO USCENTE = $| \{ v' \in V : (v, v') \in E \} |$ do(v)

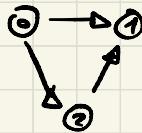
GRADO ENTRANTE = $| \{ v' \in V : (v', v) \in E \} |$ di(v)

GRADO VERTICE = do + di = d(v)

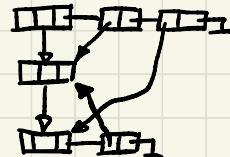
GRAFO PESATO (V, E, w) $w : E \rightarrow \mathbb{R}$ (ASSOCIA AD UN ARCO
UN NUMERO REALE)

AUDIO 24) 8/05/2023

MEMO RIZZARE UN GRAFO



GRAFO → RAPPRESENTAZIONE → A LISTA DI ADIACENZA



MATRICE DI ADIACENZA

0	1	1
0	0	0
0	1	0

$$O(|V|)$$

ACCESO AL VERTICE

$$O(|E| + |V|)$$

ACCESO ALL'ARCO

COMPLESSITÀ COMPUTAZIONALI

GRAFI HANNO DUE VARIABILI
PESCI E VERTICI

$$|E| \quad |V|$$

$$|E| \ll |V|^2 \quad (\text{grafo spesso})$$

↓
MASSIMO NUMERO DI ARCHI POSSIBILE

→ MATELLICE CONTENERE molti 0 e pochi 1

$$O(|V|^2)$$
 OPPORTUNA MEMORIA MATELLICE DI ADIACENZA

$$O(|V| + |E|)$$
 OCCUPAZIONE DI MEMORIA LISTA DI ADIACENZA

type def struct vertice-grafo {

int value;

struct vertice-grafo *vertice-succ-P;

struct enco-grafo *lince-anchi-P;

} vertice-grafo-t;

type def struct enco-grafo {

double peso;

struct vertice-grafo *vertice-adjacente-P;

struct enco-grafo *enco-nexacione-P;

} enco-grafo-t;

MATRICE AD ADJACENCI

DOUBLE GRADO [N-V][N-V];

ALBERO AKOMPENTE

MINIMO



09/05/2023 (AUDIO 25)

VISITA AMPIEZZA (BFS)

```
typedef enum { bianco, grigio, nero } colore_t;  
typedef struct vertice_grafo {  
    int valore;  
    struct vertice_grafo *vertice_succ_p;  
    struct arco_grafo *lato_anchi_p;  
    colore_t colore;  
    int distanza;  
    struct vertice_grafo *padre_p;  
} vertice_grafo_t;
```

}

```
void avvia_grafo_ampa (vertice_grafo_t *grafo_p)  
{  
    vertice_grafo_t *vertice_p;  
  
    for (vertice_p = grafo_p; (vertice_p != NULL); vertice_p =  
         vertice_p->vertice_succ_p)  
    {  
        vertice_p->colore = bianco;  
        vertice_p->distanza = -1; /* infinito */  
        vertice_p->padre_p = NULL;  
    }  
}
```

```
for (vertice_p = grafo_p; (vertice_p != NULL); vertice_p =  
     vertice_p->vertice_succ_p)  
if (vertice_p->colore == bianco)  
    visita_grafo_ampa(vertice_p);
```

}

void visito_grafo_omp (vertice_grafo_t * vertice_puntoare)

{ vertice_grafo_t * vertice_p;

orco_grafo_t * orco_p;

elem_liste_vertici_t * uscita_p; * ingresso_p;

vertice_puntoare_p \rightarrow colore = grigio;

vertice_puntoare_p \rightarrow distante = 0;

uscita_p = ingresso_p \Rightarrow = NULL;

metti_in_coda (& uscita_p, & ingresso_p, vertice_puntoare_p);

while (uscita_p != NULL) {

vertice_p = togli_de_coda (& uscita_p, & ingresso_p) \rightarrow colore;
chiudi (vertice_p \rightarrow colore);

for (orco_p = vertice_p \rightarrow liste_adiaci_p ; (orco_p != NULL));

orco_p =

orco_p = orco_p \rightarrow orco_nex;

if (orco_p \rightarrow vertice_adiacente_p \rightarrow colore == bianco)
{

orco_p \rightarrow vertice_adiacente_p \rightarrow colore = grigio;

orco_p \rightarrow vertice_adiacente_p \rightarrow distante = vertice_p \rightarrow
distante + 1;

orco_p \rightarrow vertice_adiacente_p \rightarrow padre_p = vertice_p;

metti_in_coda (& uscita_p, & ingresso_p, orco_p \rightarrow
vertice_adiacente_p);

}

vertice_p \rightarrow colore = marrone;

}

}

$$T(V, E) = O(|V| + |E|)$$

VISITA IN PROFONDITÀ (DFS)

→ AVVIENE SEMPRE PIÙ IN PROFONDITÀ

LABORATORIO 11/05/23

FILE GNAFI

g → numero di vertici

2 → numero di righe per quegli archi

1 8

1 2

3

2 8

2 3

2 1

(AUDIO 26) 15/05/23

VISITA IN PROFONDITÀ (DFS)

typedef struct vertice_grafos {

```
int valore;
struct vertice_grafos *vertice_nucc_p;
struct arce_grafos *lista_anchi_p;
colore_t colore;
int initia, fine;
struct vertice_grafos *pochi_p;
```

} vertice_grafos_t;

void avvia_visita_grafos_prof (vertice_grafos_t *grafos_t)

```
{  
    vertice_grafos_t *vertice_p;  
    int tempo;
```

```
    free(vertice_p = grafos_p; (vertice_p != NULL);  
        vertice_p = vertice_p -> vertice_succ_p)
```

}

```
    vertice_p -> colore = bianco;  
    vertice_p -> initia = vertice_p -> fine = -1;  
    vertice_p -> pochi_p = NULL;
```

}

```
free(vertice_p = grafos_p, tempo = 0; (vertice_p != NULL);  
    vertice_p = vertice_p -> vertice_succ_p)
```

if (vertice_p -> colore == bianco)

writta_grafos_prof (vertice_p, &tempo);

3

void viniti_grapf_prof (vertice_grapf_t * vertice_p, int *tempo)

{

onco_grapf_t * onco_p;

vertice_p -> colore = grigio;

vertice_p -> initia = ++(*tempo);

elaborazione (vertice_p -> valre);

for (onco_p = vertice_p -> lista_amiche_p; (onco_p != NULL);

onco_p = onco_p -> onco_nexx_p)

if (onco_p -> vertice_adiacente_p -> colore == bianco)

{

onco_p -> vertice_adiacente_p -> colore = vertice_p;

viniti_grapf_prof (onco_p -> vertice_adiacente_p, tempo);

}

vertice_p -> colore = marrone;

vertice_p -> finale = ++(*tempo);

3

$$T(EV) = T(|E| + |V|) = O(|E| + |V|)$$

$G = (V, E)$: determinare un ordinamento lineare dei vertici

t.c. se $(v, v') \in E \Rightarrow v$ precede v' nell'ordinamento

22/05/23 (AUDIO 27)

ORDINAMENTO TOPOLOGICO

Complessità di un algoritmo di ordinamento topologico $O(|V| + |E|)$

(1) ciclo for controllare tutti i vertici in bianco

(2) ciclo for si occupa di controllare che tutti i vertici vengano riluotati

↳ ciclo for da ordino_top_graf → lista recorsiva

PROBLEMA DEL PERCORSO PIÙ BREVE

con la condizione di non avere un ciclo con orchi negativi

↳ SOLO PERCORSI SEMPLICI

Principio di ottimalità dimostrato per esempio

typedef struct vertice_graf {

int value;

struct vertice_graf *vertice_suc_p;

struct arco_graf *listo_anchi_p;

double distanza_min;

struct vertice_graf *pochi_p;

} vertice_graf_t

```

void initialize (vertice_graf * graph, 
                 vertice_graf * vertice_p) { }

    vertice_graf * vertice_p;

    for (vertice_p = graph_p; (vertice_p != NULL); 
         vertice_p = vertice_p->vertice_succ_p) { }

        vertice_p->distante_min = INFINITO;
        vertice_p->path_p = NULL;

    }

    sorgente_p->distante_min = 0.0;

}

void minval (arco_graf * arco_p, vertice_graf * vertice_p) { }

    if (arco_p->vertice_arco_p->distante_min >
        vertice_p->distante_min + arco_p->peso) { }

        arco_p->vertice_arco_p->distante_min =
            vertice_p->distante_min + arco_p->peso;

        arco_p->vertice_arco_p->path_p = vertice_p;

    }

}

```

(AUDIO 28) 23/05/2023

ALGORITMO DI DAG BELLMAN-FORD

ALGORITMO DI DIJKSTRA

→ Complessità migliore

↳ non negativo (peso)

↳ $S = \{ \text{vertici passi finiti minimi} \}$

↳ code con priorità → complessità $O(|V| \lg |V| + |E|)$

Implementazione

Complessità
alg. di Dijkstra

Array

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

Heap binario

$$O(|V| + |E|) \cdot \lg |V|$$

d-ary heap più di binario

$$O(|V| \cdot d + |E|) \cdot \frac{\lg |V|}{\lg d}$$

Fibonacci heap

$$O(|V| \lg (|V| + |E|))$$

400 MPS \rightarrow

1 STRIDE AL SEGUNDO =

$$\frac{1}{400\cdot 000\ 000}$$

$m = 400000$ \swarrow

$$O(\log(m)) \quad \frac{\log(400000)}{400\ 000\ 000} = \text{Tempo}$$

$$O(x^2) \quad \frac{(400000)^2}{400\ 000\ 000} = \text{Tempo}$$

$$2^{20} \approx 10^6$$

$$V_{init} = O(n) \rightarrow n = 10^6$$

