

⌚ Dimostrazione: Induzione

$$k = 0 \implies 1 = \sum_{i=0}^0 a^i = \frac{a^1 - 1}{a - 1}$$

$$a^k + \sum_{i=0}^{k-1} a^i = a^k + \frac{a^k - 1}{a - 1} = \frac{a^{k+1} - a^k}{a - 1} \neq a^k - 1$$

$$\begin{aligned} a > 1 \\ \sum_{i=1}^{\infty} \frac{1}{a^i} &= k > 0 \\ \sum_{i=1}^{\infty} \frac{1}{i} &= \infty \end{aligned}$$

$$\sum_{i=1}^n \log(i) = \log(n!) = \Theta(n \log(n))$$

⌚ Dimostrazione

$$\log(n!) = \log(n \cdot (n-1) \cdots 2 \cdot 1) = \log(n) + \cdots + \log(1)$$

$$\begin{aligned} \log(n) + \cdots + \log(1) &\leq n \log(n) \\ \log(n) + \cdots + \log\left(\frac{n}{2} + 1\right) + \cdots &\geq \frac{n}{2} \log\left(\frac{n}{2}\right) = \Theta(n \log(n)) \\ \implies \log(n!) &= \Theta(n \log(n)) \end{aligned}$$

Ricerca della Somma in Due Vettori

⌚ Problema

$$\begin{aligned} \sum_{i=1}^n i^k &= \Theta(n^{k+1}) \\ \sum_{i=1}^n i^k &= 1^k + 2^k + \cdots + n^k \\ \sum_{i=1}^n i^k &\leq n^{k+1} \\ \left(\frac{n}{2}\right)^k + \cdots + n^k &\geq \left(\frac{n}{2}\right)^k \frac{n}{2} = \left(\frac{n}{2}\right)^{k+1} = \Theta(n^{k+1}) \\ \implies \sum_{i=1}^n i^k &= \Theta(n^{k+1}) \end{aligned}$$

$$\sum_{i=0}^k a^i = \frac{a^{k+1} - 1}{a - 1} = \Theta(a^k)$$

- Input:** 2 vettori V, W di dimensione n contenenti solo elementi $x > 0$ ed un valore $k > 0$
 - Output:** se esistono due indici i, j tale che $V[i] + W[j] = k$
- Un primo approccio, non che il più semplice, è quello di **brute force** controllando tutti i valori

BRUTE-FORCE(V, W, k)

```

1   n = V.length
2   for i = 1 to n
3       for j = 1 to n
4           if V[i] + W[j] = k
5               return TRUE
6   return FALSE

```

Parametri: V, W =vettori, k =somma da trovare

Return: se esistono due indici che danno la somma

① Costo Computazionale

$$\Theta(n^2)$$

Se prima ordino uno dei due vettori, posso utilizzare la ricerca binaria per cercare il valore necessario ad arrivare alla somma

$$T(n) = \Theta(n \log(n) + n \cdot \log(n)) = \Theta(n \log(n))$$

① Costo Computazionale

$$\Theta(n \log(n))$$

return NIL

```

2   return NIL
3
4   mid = floor((start + end) / 2)
5   if value = A[mid]
6       return mid
7   else if value < A[mid]
8       return BINARY-SEARCH(A, value, start, mid - 1)
9   else
10      return BINARY-SEARCH(A, value, mid + 1, end)

```

Parametri: A =vettore ordinato, $value$ =valore da cercare, $start$ =inizio del vettore, end =fine del vettore

Return: indice del valore nel vettore o NIL se non trovato

ORDER-AND-SEARCH(V, W, k)

```

1   SORT(W)
2
3   n = V.length
4   for i = 1 to n
5       j = BINARY-SEARCH(W, k - V[i])
6       if index != NIL
7           return TRUE
8
9   return FALSE

```

Parametri: V, W =vettori, k =somma da trovare

Return: se esistono due indici che danno la somma

Utilizzando un qualsiasi algoritmo di sort $\Theta(n \log(n))$ e la ricerca binaria $\Theta(\log(n))$ otteniamo

② Idea

$$T(n) = \Theta(n \log(n) + n \cdot \log(n)) = \Theta(n \log(n))$$

Ordinamento e Ricerca Binaria

BINARY-SEARCH($A, value, start, end$)

```

1   if start > end

```

Ipotesi Ordinati

Se ipotizziamo che i **vettori siano già entrambi ordinati**, possiamo ottimizzare ulteriormente l'algoritmo

💡 Idea

Se consideriamo una matrice costituita dalle somme di ogni valore dei due vettori, possiamo escludere a ogni passaggio porzioni intere di valori

Consideriamo la **matrice delle somme** possibili

$$V = (x_1, \dots, x_n) \quad W = (y_1, \dots, y_n)$$

$$\begin{pmatrix} x_1 + y_1 & \dots & x_n + y_1 \\ \dots & x_i + y_j & \dots \\ x_1 + y_n & \dots & x_n + y_n \end{pmatrix}$$

Per ogni elemento della matrice possiamo notare che, essendo i vettori ordinati

- Tutti i valori con **entrambi gli indici minori saranno minori**
- Tutti i valori con **almeno un indice maggiore saranno maggiori**

Prendiamo quindi il **valore centrale della matrice** e dividiamo quest'ultima in quattro quadranti

- Il primo quadrante (alto a sinistra) avrà solo valori minori del centrale
- I restanti tre possono avere valori maggiori, minori o anche uguali

Da qui possiamo applicare la stessa **logica della ricerca binaria** per vedere in quale quadrante ricade il valore k da trovare

MATRIX-BINARY-SEARCH(M, k, start_row, start_col, end_row, end_col)

```
1   if start_row > end_row or start_col > end_col
2     return FALSE
3
4   n = M.rows
5   mid = floor(n / 2)
6
7   if k = M[mid, mid]
8     return TRUE
9   else if k < M[mid, mid]
10    // primo (nw) quadrante
11    return MATRIX-BINARY-SEARCH(M, k,
12                                start_row, start_col,
13                                mid, mid)
14  else
15    // secondo (ne), terzo (se) e quarto (sw) quadrante
16  result = MATRIX-BINARY-SEARCH(M, k,
```

Parametri: $V, W = \text{vettori}$

Return: matrice M con le somme degli elementi di V e W ordinate

Matrix Binary-Search

```

17     start_row, mid + 1,
18     mid, end_col)
19         if result = TRUE
20             return TRUE
21
22     result = MATRIX-BINARY-SEARCH(M, k,
23
24     mid + 1, mid + 1,
25     end_row, end_col)
26         if result = TRUE
27             return TRUE
28
29     return MATRIX-BINARY-SEARCH(M, k,
30
31     mid + 1, start_col,
32     end_row, mid)

```

Parametri: M =matrice restituita da [BUILD-MATRIX\(\)](#), k =somma da trovare, $start_row, start_col=riga$ e $colonna$ iniziali della matrice, $end_row, end_col=riga$ e $colonna$ finali della matrice

Return: se esistono due indici che danno la somma

Andando ad analizzare il **caso pessimo**

$$T(n) = 1 + 3T\left(\frac{n}{2}\right)$$

L'albero della ricorsione sarà quindi

$$\sum_{i=0}^{\log_2(n)} 3^i = \Theta(3^{\log_2(n)}) = \Theta(3^{\log_3(n) \cdot \log_2(3)}) = \Theta(n^{\log_2(3)})$$

Ma siccome $\log_2(3) > 1$, **non abbiamo ottenuto un costo lineare** (o migliore)

ⓘ Costo Computazionale: MATRIX-BINARY-SEARCH

$$\Theta(n^{\log_2(3)})$$

Matrix Snake-Search

ⓘ Osservazione

Per ogni elemento della matrice possiamo notare che, essendo i vettori ordinati

- Ogni valore nella **stessa colonna** sarà maggiore o minore se in una **riga maggiore o minore**
- Ogni valore nella **stessa riga** sarà maggiore o minore se in una **colonna maggiore o minore**

Prendiamo quindi il valore $i = 1, j = n$ in quanto avrà

- Tutti gli elementi nella **stessa colonna minori** (in quanto ultimo)
- Tutti gli elementi nella **stessa riga maggiori** (in quanto primo)

Da qui possiamo muoverci nella matrice a seconda di k

- Se k è maggiore, ci muoviamo di **1 a destra**
- Se k è minore, ci muoviamo di **1 sopra**
- Se arriviamo ai bordi opposti della matrice, il **valore non è presente**

MATRIX-SNAKE-SEARCH(M, k, end_row, start_col)

```

1 if start_row > M.rows or start_col > M.columns
2 return FALSE
3
4 value = M[end_row, start_col]
5 if k == value
6   return TRUE
7 else if k > value
8   // 1 a destra
9   return MATRIX-SNAKE-SEARCH(M, k, end_row, start_col
+ 1)
10 else
11   // 1 sopra
12   return MATRIX-SNAKE-SEARCH(M, k, end_row - 1,
start_col)

```

Parametri: M =matrice restituita da **BUILD-MATRIX()**, k =somma da trovare, end_row =riga finale della matrice, $start_col$ =colonna iniziale della matrice

Return: se esistono due indici che danno la somma

Nel caso peggiore, arriveremo al bordo della matrice dopo n passi avanti e n passi in alto

$$T(n) = \Theta(2n) = \Theta(n)$$

① Costo Computazionale: MATRIX-SNAKE-SEARCH

$$\Theta(n)$$

BFS DANE AD OGNI LIVELLO ALTRENO IL COLORONE

SE INCONTRA UN NODO CON COLORI, CONTROLLA CHE

IL COLORONE SIA COMPATIBILE

② GRAFO PLANARE = NO IN CLASS

- SEMPRE IN COLORABILE

③ GRAFO COLORARE ESENTE A CLASH

- Se esistono due sottosinsiemi non vuoti V_1 e V_2 di V tali che $V_1 \cup V_2 = V$ e $V_1 \cap V_2 = \emptyset$. Inoltre deve valere anche che se $u, v \in V_1$ allora $(u, v) \notin E$ e se $u, v \in V_2$ allora $(u, v) \notin E$. Se V_1 e V_2 con queste proprietà non esistono l'algoritmo deve restituire NO .
- Calcolare il costo computazionale dell'algoritmo proposto
- Argomentare la sua correttezza.

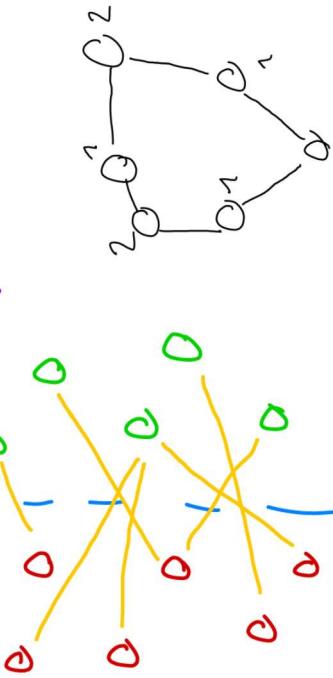
Traccia Soluzione esercizio 3.

- Il problema equivale a dire se un grafo è 2-colorabile. Basta usare una BFS opportunamente modificata.

GRAFO BIPARTITO

V_1 - V_2 - \Rightarrow 2 COLORABILE

\Leftarrow DUE CICLI DISPARI



2 colorabili = NO

NO, COLLEGATI CON STESSO colore

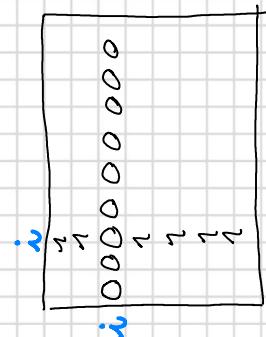


① Input: (M) grafo orientato come matrice

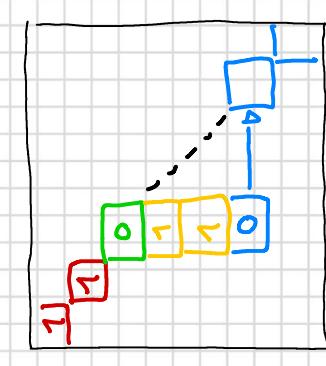
② Output: \exists posso in M

posso := $v \in V \setminus \{w\}$, $w \rightarrow v \in E$

ovvero tutti i nodi vanno a v ma non va in nessuno (v è unico)



non posso **UPPER BOUND:** $O(n^2)$



$$\sqrt{\sqrt{n}} = \left(\left(n^{\frac{1}{2^k}}\right)^{\frac{1}{2}}\right)^{\frac{1}{2}} = n^{\frac{1}{2^k}} \leq h$$

$$\log\left(n^{\frac{1}{2^k}}\right) \leq \log(h) \Leftrightarrow \frac{1}{2^k} \log(n) \leq \log(h)$$

$$\Leftrightarrow \log(n) \leq 2^k \log(h)$$

$$\Leftrightarrow \frac{\log(n)}{\log(h)} \leq 2^k \Leftrightarrow \log\left(\frac{\log(n)}{\log(h)}\right) \leq k \log(2)$$

• Controlliamo la diagonale: finché $\Rightarrow K \geq \log\left(\frac{\log(n)}{\log(h)}\right) / \log(2)$

- Se incontro solo v , è l'unico possibile posso
- Se incontro una v , posso scartare tutta la porzione superiore e riprendere la ricerca alla stessa altezza

COSTO: $\Theta(n)$

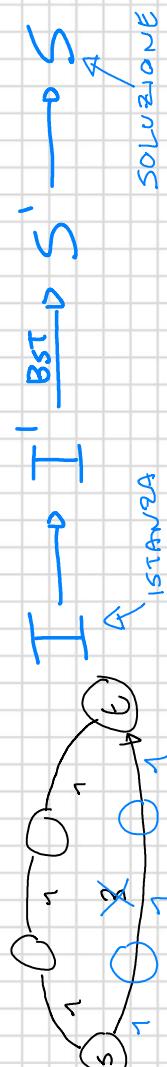
Output: cammino minimo su molti in tempo $\Theta(n+m)$

Input: $G = (V, E, \omega)$ grafo non orientato

Output: $\omega(e_v) = \omega_i$

$\omega_i \in \{1, 2, 3\}$

Spazzando ogni arco in equivalenti di peso 1, possiamo applicare una normale BFS



$I \rightarrow I'$ n nodi ed m archi

Siccome i valori dei pesi sono costanti, spezzare un arco sarà costante

(n)(m)

$I' \rightarrow S'$

Nel caso peggiore, per ogni arco spezziamo in 2 nodi e 3 archi per spezzarlo

$$\begin{aligned} n' &= h + 2m \\ m' &= 3m \end{aligned}$$

(h)(n+m)

$S' \rightarrow S$ Per lo stesso ragionamento, da trasformazione inversa avrà costo lineare

$H(n+m)$ TOTALE $\Theta(n+m)$

③ Input: $V = [1|2|1|3|5|2]_n$

Output: $P = [60|30|60|20|12|30]$

$\hookrightarrow x_i = \left(\prod_{j=1}^i v_j \right) / v_i$

SENZA USARE LA DIVISIONE

calcolo: "suffissi" e "prefissi"

$$\begin{aligned} s &= 1|2|1|6|30|60 \\ P &= 60|60|30|20|10|2 \end{aligned}$$

$$x_i = s_{i-1} \cdot P_{i+n}$$

INPUT $G(V, E, w)$, $s \in V$, $t \in V$ $w(v) \in \mathbb{Z}^+$

Peso dato $\xrightarrow{\text{2}} \xrightarrow{\text{0.2}} \xrightarrow{\text{D.o.}}$ $\xrightarrow{\text{D.o.}}$ $w = 24$
dal prodotto

OUTPUT Percorso più leggero

SOLUZIONE trasformando $w(v)$ in $\log(w(v))$
, il prodotto diventa somma e applichi
i normali algoritmi

$$\textcircled{1} T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + 1$$

$$T(n) \geq 2T\left(\frac{n}{3}\right) + 1 \quad \textcircled{4} \left(n \log_3 2 \right)$$

$$T(n) \leq 2T\left(\frac{n}{2}\right) + 1 \quad \textcircled{5} \left(n \right)$$



$$T(n) = T\left(\frac{n}{80}\right) + T\left(\frac{n}{80}\right) + T\left(\frac{n}{80}\right) + 1$$

$$T(n) \geq 3T\left(\frac{n}{80}\right) + 1 = \textcircled{6} \left(n \log_{80} 3 \right)$$

$$T(n) \leq 3T\left(\frac{n}{80}\right) + 1 = \textcircled{7} \left(n \log_{80} 3 \right)$$

1 $V = \{a_1, \dots, a_n\}$ numeri; $0 \leq a_i \leq m^2$

OBBIETTIVO: sort $\textcircled{4}(n)$

- counting sort $\rightarrow \textcircled{4}(n+k) = \textcircled{4}(n^2)$

Lunghezza $a = \log_2(a)$

$$\text{Lunghezza } n^2 = \log(n^2) = 2 \log_2(n)$$



$$\textcircled{4}(m \log n)$$

DIVIDIMO IN 2
NACCE CIFRE

counting sort $\Rightarrow k = \log m \Rightarrow \textcircled{4}(m)$

radix sort $\textcircled{4}(m)$

$$\text{radix sort } d = 2 \Rightarrow \textcircled{4}(m)$$

Vale in generale per $0 \leq a_i \leq m$ costante