ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

a) Prim(G,c){
S è l'insieme dei nodi esplorati
foreach nodo u non in S, memorizzo il costo in a[u] e il predecessore in p[u]
Q è una coda a priorità del tipo (a[u], u) t.c. u non in S
insert(0,s) in Q
T ← Q //Albero vuoto
p[s] = null
foreach nodo u ∈ S, memorizzo (Q, ∞, u) in Q
while(Q non è vuota){
(a[u], u) ← ExtractMin(Q)
if(p[u] ≠ null){
T ← T ∪ {(p[u], u)};
}
Add u to S
foreach arco e=(u,v) incidente su u {
if(v non è in S e (e < a[v])){
ChangeKey(Q, (e, v)
p[v] = u;{{
}
return T
}

tempo di esecuzione:

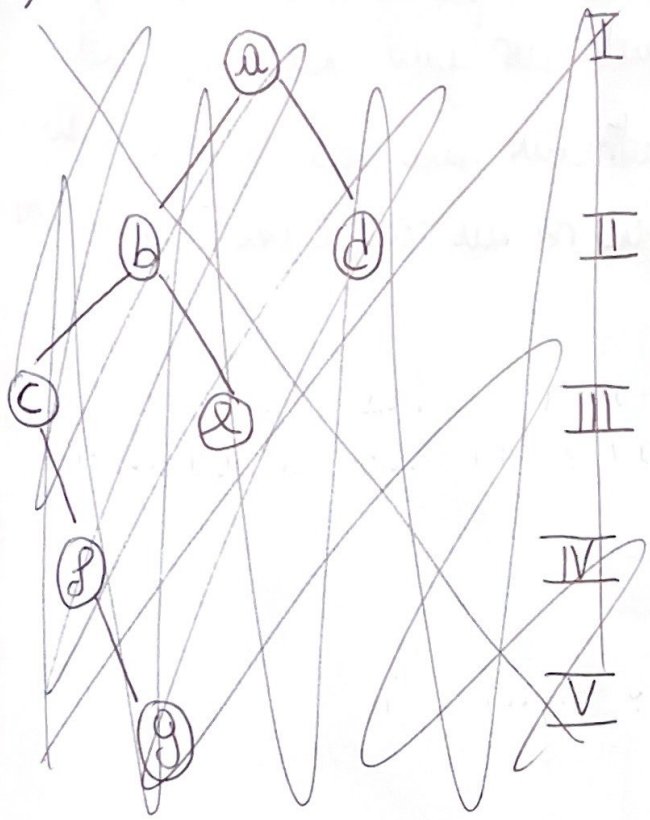
- Il primo for ha tempo per singola iterazione $O(\log n)$ in totale per n iterazioni ha tempo $O(n \log n)$.
- All'interno del while avremo tempo $O(n \log n)$, vediamo perché:

1) abbiamo l'ExtractMin che ha tempo per singola iterazione $O(\log n)$, per n iterazioni $O(n \log n)$
2) Il for più interno verrà eseguito $O(\sum_{v \in V} deg(v) \leq 2m) = O(m)$
3) Per effettuare il ChangeKey per singola iterazione $O(\log n)$ per n iterazioni ha tempo $O(n \log n)$

Il tempo totale è:

$O(n \log n) + O(n \log n) = O(n \log n)$ poiché il grafo è connesso quindi $m \geq n+1$.

b)



ERRATO, ~~#~~ pagina 5
sequenza corretta.

c) Un ordinamento topologico è un etichettatura dei nodi $v_i, \dots, v_j$, e andando a scegliere un nodo ~~precedente~~ all'interno del grafo G vi otteniamo un ciclo (v_i, v_j) con $j < i$

Dimostrazione: (per induzione)

Caso base: con $n = 1$ banalmente vero

Passo induttivo:

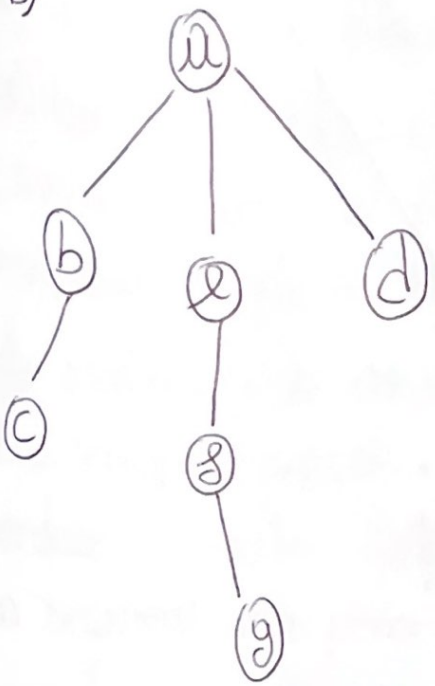
- Consideriamo G un Dag con $n+1 \geq n$ nodi.
- Possiamo dire che $G - \{v\}$ è ancora un DAG poiché all'interno di esso per definizione ha un nodo senza archi entranti.
- Andando ~~a~~ a cancellare per n volte i nodi che ~~sono~~ non

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

hanno archi entranti, arriveremo ad un punto in cui tutti gli archi puntavano sempre in avanti per ogni nodo selezionato quindi essi non avevano archi entranti ma solo archi uscenti. Quindi se G è un DAG esso allora ha un ordinamento topologico.

ESPOSITO VINCENZO 0512106070 Venerdì EspA

b)



I

II

III

IV

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

a) INPUT: n job ognuno dei quali inizia al tempo s_j e termina ad un determinato tempo f_j.

OBIETTIVO: trovare un sottoinsieme S di cardinalità massima di job a due a due compatibili.

La scelta greedy che viene effettuata è Earliest Finish time, cioè vengono ordinati i job in base ai tempi più piccoli in ordine crescente. La scelta greedy consiste nell'andare a scegliere ogni passo il job che ha tempo di fine più piccoli tra quelli non ancora esaminati.

b)

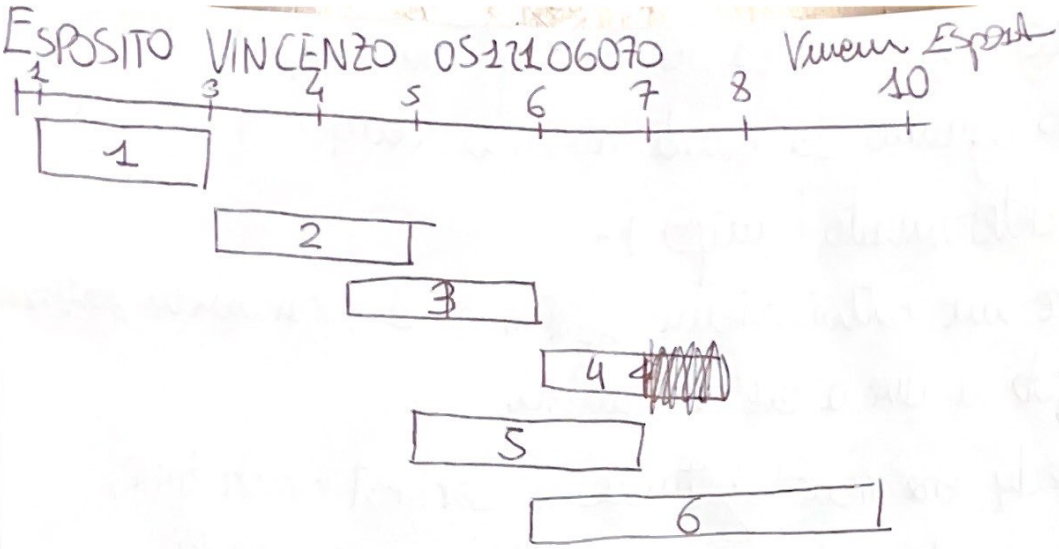
$s_1 = 1$ $f_1 = 3$
$s_2 = 4$ $f_2 = 6$
$s_3 = 3$ $f_3 = 5$
$s_4 = 6$ $f_4 = 7$
$s_5 = 6$ $f_5 = 10$
$s_6 = 5$ $f_6 = 7$

⇒ ordino i job in base ai tempi di fine

① $s_1 = 1$ $f_1 = 3$
② $s_3 = 3$ $f_3 = 5$
③ $s_2 = 4$ $f_2 = 6$
④ $s_4 = 6$ $f_4 = 7$
⑤ $s_6 = 5$ $f_6 = 7$
⑥ $s_5 = 6$ $f_5 = 10$

Esecuzione alle pagina successive

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito



Soluzione ottima {1, 2, 4}

a) A = ∅ *

① SJ > f si
A = {1}
f = 3

② SJ > f si
A = {1, 2}
f = 5

③ SJ > f no
A = {1, 2}
f = 5

④ SJ > f si
A = {1, 2, 4}
f = 7

⑤ SJ > f no
A = {1, 2, 4}
f = 7

⑥ SJ > f no
A = {1, 2, 4}
f = 7

* Esecuzione dell'algoritmo dopo aver ordinato i job

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

c)

Sort in base ai tempi di fine $f_1 \leq f_2 \leq \ldots \leq f_n$

```
f ← 0
A ← ∅
for J = 1 to n {
    if (S_J ≥ f) {
        A ← A ∪ {J}{
        f ← f_J{
{
return A{
```

Il tempo di esecuzione dell'algoritmo è:

- Per ordinare la sequenza in base ai tempi di fine l'algoritmo impiega $O(n \log n)$.
- Il for ha tempo $O(n)$

Il tempo totale dell'algoritmo è $O(n \log n)$.

ESPOSITO VINCENZO 0512106070 Vincenzo Esposto

a) INPUT: n job ognuno dei quali richiede tempo $w_i > 0$ e abbiamo un limite di utilizzo del processore W.

OBIETTIVO: trovare un sottoinsieme S t.c. $\sum_{i \in S} w_i$ sia quanto più grande possibile e che si rispetti il vincolo $\sum_{i \in S} w_i \leq W$.

OPT(i,w) = rappresenta il valore della soluzione ottima per il problema subset sum che ha in input 1,... i job con limite di utilizzo del processore w.

$$OPT(i,w) = \begin{cases} 0 & \text{se } i = 0 \\ OPT(i-1,w) & \text{se } w_i > w \\ \max\{OPT(i-1,w),\ w_i + OPT(i-1,w-w_i)\} & \text{altrimenti} \end{cases}$$

- Se $i = 0$ banalmente non abbiamo job.
- Se $w_i > w$, il tempo che richiede il job è più grande del tempo che ha il processore per utilizzare il job, quindi l'unica soluzione possibile è $OPT(i,w) = OPT(i-1,w)$.
- Nell'ultimo caso possono verificarsi due casistiche:

Caso 1 = il valore della soluzione ottima sarà dato dalla riga $i-1$ (precedente) andando a prendere il valore presente in esse.

Caso 2 = il valore della soluzione ottima sarà dato dal valore nella riga $i-1$ andando a calcolarci la somma del valore presente nella cella $w - w_i$ + w_i.

$OPT(i-1, w-w_i)$ in questo caso.

b) Input: $n, w_1, \ldots, w_n, W$

```
for i=0 to n
  M[0,w] = 0
for w=0 to n
  M[i,w] = w
SubsetSums(i,w){
  if (i==0){
    return null;{
  if (M[i,w] == null){
    return M[i,w]{
  if wi>w}
    M[i,w] = SubsetSums(i-1, w);{
  else{
    M[i,w]= max{(SubsetSums(i-1,w), SubsetSums(i-1, w-wi)+wi)}}
  return M[n,w];
}
```

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

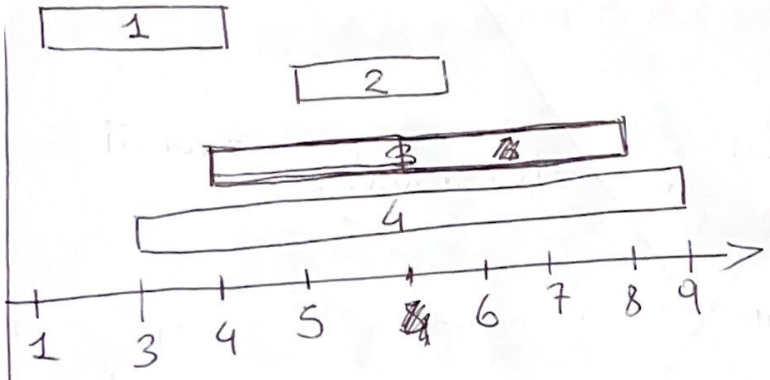
c) Ordino in base ai tempi di fine

① $s_2 = 1$ $f_2 = 4$ $w(2) = 3$

② $s_3 = 5$ $f_3 = 6$ $w(3) = 6$

③ $s_4 = 4$ $f_4 = 8$ $w(4) = 1$

④ $s_1 = 3$ $f_1 = 9$ $w(1) = 4$



$P(1) = 3$

$P(2) = 3+6 = 9$

$P(3) = 3+1 = 4$

$P(4) = 4$

Soluzione ottima = $\{1,2\}$

Valore della soluzione ottima = 9

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

a) • $n(\log n)^4 + 1000n^4 = O(n^4)$ VERA

• $n^{4/2} = \Omega(n^{3/4} n^{1/4})$ VERA

• $n^{1/10} = O(\log n)$ FALSA

• $n^3 + 1000n^2 + 100 = \Theta(n^3)$ VERA

• $\log(\log n) = \Omega((\log n)^{1/2})$ FALSA

b) Partendo dalla definizione di O

① $f(n)$ è $O(h(n))$ se e solo se $\exists$ due costanti $c' > 0$ e $n_0' \geq 0$ t.c.

$f(n) \leq c' \cdot h(n) \quad \forall n' \geq n_0'$

② $g(n)$ è $O(p(n))$ se e solo se $\exists$ due costanti $c'' > 0$ e $n_0'' \geq 0$ t.c.

$g(n) \leq c'' \cdot p(n) \quad \forall n'' \geq n_0''$

Partendo dalla ①, possiamo moltiplicare per $g(n)$ e dx e sx

$f(n) \cdot g(n) \leq c' h(n) \cdot g(n)$

La moltiplicazione può essere effettuata per il principio delle disequazioni, cioè aggiungendo ad entrambi i lati della disequazione una quantità il risultato non cambia

Otteniamo:

$f(n) \cdot g(n) \leq c' h(n) \cdot c'' p(n)$

• $f(n) g(n) = O(h(n) \cdot p(n))$ con $c = c' \cdot c''$ e $n = n' \cdot n''$ $\forall n \geq 0$

```
)For(i=1; i<n; i=i+1){
 For(j=1; j< 3^n; j=j*3){
    print(j){
 {
```

• Il for più esterno terminerà quando $i = n$, quindi asintoticamente tenderà a $O(n)$.

Il for più interno terminerà quando $j = 3^n$, possiamo riscrivere $j = 3^j$, quindi: ~~$3^n = 3^j$. Applico il logaritmo~~

$$3^n = 3^j \Longrightarrow n \log 3 = j \log 3$$

Il for più interno termine $j = n$, asintoticamente tendrà a $O(n)$

~~Il tempo totale del~~ Il tempo totale $= O(n) \cdot O(n) = O(n^2)$

ESPOSITO VINCENZO 0512106070 Vincenzo Esposito

a)
```
cerca(x, array){
  if(array == null){
    return -1{
  u = array[i]
  v = array[i+1]
  if(u == v)
```

```
cerca(x, array){
  if(x == null){
    return -1, {
  if(array == null){
    return -1; {
  if(array[0] == x){
    max = conta(x, array - array[0]) + 1,
  { else conta(x, array - array[0]);
  return max;
}
```

b)
$$T(n) \leq \begin{cases} c & \text{se } x = null \text{ oppure } array = null \\ T(n-1) + c & \text{altrimenti} \end{cases}$$

c)
$$T(n) \leq T(n-1) + c$$
$$\leq T(n-2) + ci$$
$$\leq \ldots \leq \text{Generalizzando}$$
$$\leq T(n-n) + cn$$
$$= O(n)$$