

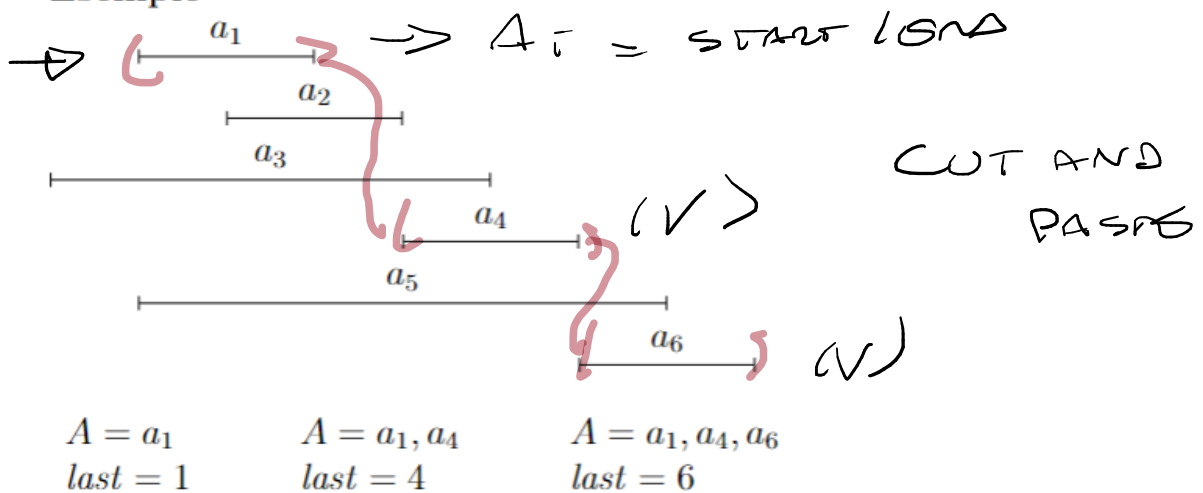
GREEDY → SCELTA INTELLIGENTE PROBLEMA

GREEDY-SEL(S, f)

```

1   $n = S.length$ 
→ 2   $A = \{a_1\}$  → GREEDY = STR. DAT. CHE SALVA LA SOL.
3   $last = 1$  // indice dell'ultima attività selezionata
4  for  $m = 2$  to  $n$ 
5      if  $s_m \geq f_{last}$  → POSSO AD
6          →  $A = A \cup \{a_m\}$  → ATTACCARE
7          →  $last = m$  → UN'ALTRA ATTIVITÀ
8  return  $A$  → COMPATIBILI
    
```

Esempio



ATTIVITÀ
COMPATIBILI

→ GRADO UNICO
→ POI SOLUZIONI POSSIBILI
→ NON SI POSSONO
SOVRAPPORRE?

ALGORITMO
GREEDY

→ SCELTA "OTTIMA"

↓
RAGIONE PER
CUT AND PASTE

LOGICA (DIMOSTRAZIONE):

$$A = \{a_1, \dots, a_m\}$$

$$A^* = A^* \cup \{a_i\}$$

SE COMPATIBILI

$$A^* = \text{SOL. OTTIMA}$$

$$A_i = f_{max} \geq A_{start}$$

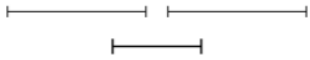
→ OTTIMO!

8.2.4 Scelte Greedy Alternative

Oltre alla scelta greedy vista precedentemente, ne esistono altre:

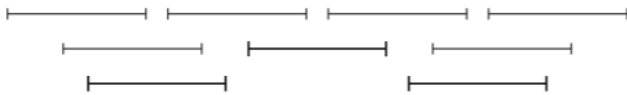
- Scegli l'attività di durata inferiore → non è ottima.

Controesempio:



- Scegli l'attività col minor numero di sovrapposizioni → non è ottima.

Controesempio:



- Scegli l'attività che inizia per prima → non è ottima.

Controesempio:



- Scegli l'attività che inizia per ultima → è ottima.

Controesempio 1



ALTRA STRA

GREEDY

NON

OTTIMO



ASSUMI SIA

SOPRA

Esercizio: matching sulla linea

Sia $S = \{s_1, s_2, \dots, s_m\}$ un insieme di punti ordinati sulla retta reale, rappresentanti dei server. Sia $C = \{c_1, c_2, \dots, c_n\}$ un insieme di punti ordinati sulla retta reale, rappresentanti dei client. Il costo di assegnare un client c_i ad un server s_j è $|c_i - s_j|$. Fornire un algoritmo greedy che assegna ogni client ad un server distinto e che minimizzi il costo totale (equiv. medio) dell'assegnamento.

GREEDY → SOL-ACC

$S = \{s_1, \dots, s_m\}$

$C = \{c_1, \dots, c_n\}$

$|C_i - S_j| = \text{Assegnamento}$

$c_1 - s_1, c_2 - s_2$

----- > ... < ...

$\left(\begin{matrix} S \rightarrow \dots \\ C \rightarrow \dots \end{matrix} \right) = \text{ordinamento}$

GREEDY-SEL(S, f)

```

1  n = S.length
2  A = {a_1}
3  last = 1 // indice dell'ultima attività selezionata
4  for m = 2 to n
5      if  $s_m \geq f_{last}$ 
6          A = A ∪ {a_m}
7          last = m
8  return A
    
```

GREEDY-CLIENT-SERVER(C, S)

```

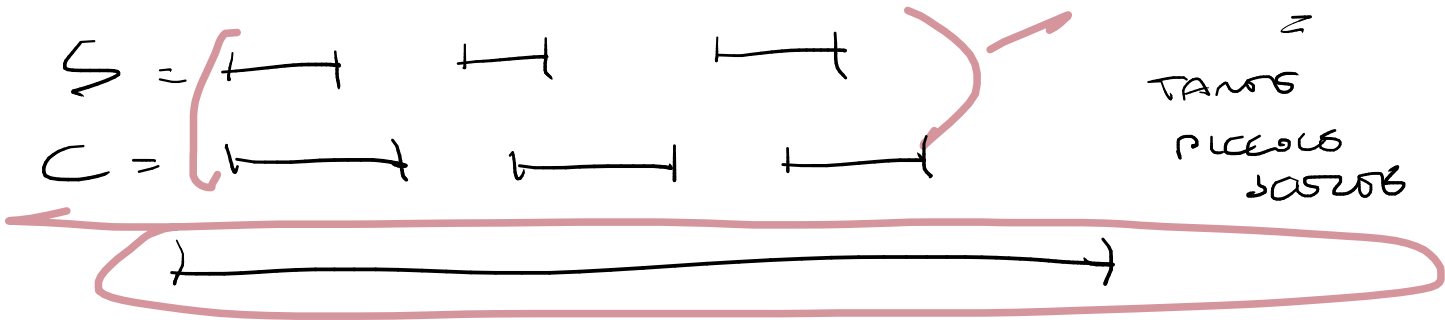
1. n = S.length
// Assunzione (client e server sono dello stesso numero)
2. SORT(C) → Ordina client
3. SORT(S) → Ordina server

4. OPT = {c_1, s_1} → Il pezzo iniziale è il primo elemento
5. last = 1

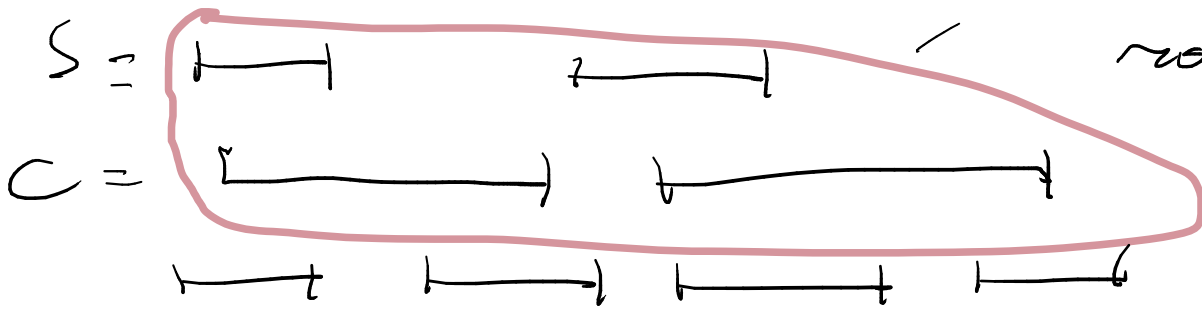
6. for m = 2 to n
7.     if {c_m - s_m} ≥ {c_last - s_last}
8.         OPT = OPT ∪ {c_m - s_m}
9.         last = m
    
```

10. return OPT

MINIMIZZARE
= TANG
PICCOLE
SOSTO



MASSIMIZZARE
= NON COSE,
PU
IN GRAD



GREEDY { MIN \rightarrow FOR - ...
MAX \rightarrow

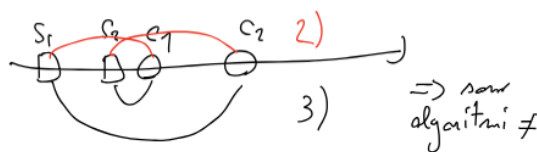
esempio: $n=4$

~~X~~ client al server + vicini, partendo dalla
coppia client-server con distanza minore

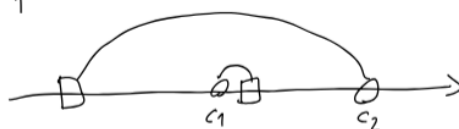
NON funziona: OPT

✓ 2) $C_1 - S_1, C_2 - S_2, \dots$
(equiv. $C_n - S_n, C_{n+1} - S_{n+1}, \dots$)

~~X~~ 3) C_1 al server + vicini; C_2 al server + vicini



NON funziona:




Esercizio 2 (9 punti) Supponiamo di avere un numero illimitato di monete di ciascuno dei seguenti valori: 50, 20, 1. Dato un numero intero positivo n , l'obiettivo è selezionare il più piccolo numero di monete tale che il loro valore totale sia n . Consideriamo l'algoritmo greedy che consiste nel selezionare ripetutamente la moneta di valore più grande possibile.

(a) Fornire un valore di n per cui l'algoritmo greedy *non* restituisce una soluzione ottima.

(b) Supponiamo ora che i valori delle monete siano 10, 5, 1. In questo caso l'algoritmo greedy restituisce sempre una soluzione ottima: dimostrare che ogni insieme ottimo M^* di monete di valore totale n contiene la scelta greedy.

m=60 \rightarrow 20 150 1] 2



$$\frac{60}{20} = 3$$

$(n=60)$ — 10 moedas de 1 €
e 1 moeda de 50 —→ 11 moedas

So to

↓
முதல்

3 months to 20

$$[3 < 17] \rightarrow$$

↳ GROSSI BUONA

① → WT - AND - PASTS → TAGUA - B

in colla

SOL \oplus - x - ortho complement
 $= \text{SOL}^x \cup d \text{ in } Z$

QUALCOSA DI BUONO

VALORI MULTIPLI

(b) Supponiamo ora che i valori delle monete siano (10, 5, 1). In questo caso l'algoritmo greedy restituisce sempre una soluzione ottima: dimostrare che ogni insieme ottimo M^* di monete di valore totale n contiene la scelta greedy.

$M^* = \text{SOL. OTTIMA}$

$n = 20 \rightarrow 10 \mid 5 \mid 1$
2 monete da 10

$x = \text{VAL. MAGGIORE TRA } \{10, 5, 1\}$

INSIEME DI
INQUI

$M \subseteq M^*$

$M + x \subseteq M^* \quad ? \quad \text{OTTIMA?}$

$M' = M^* \setminus M \cup x \rightarrow \text{ATTACCHI}$
CUT AND PASTE
UN POZZO
NON
OTTIMO
↓
ES. QUI:
MONETE DA
1

M' AVRÀ PIÙ MONETE
RISPETTO A M^*

M' NON OTTIMO \rightarrow INSIS 20 MONETE
A 500 MONETE!

(b) Sia M^* una soluzione ottima. Sia x il valore maggiore tra 10, 5, e 1 che sia non superiore a n . Se M^* contiene una moneta di valore x , la proprietà è dimostrata. Altrimenti, sia $M \subseteq M^*$ un insieme di (2 o più) monete di valore totale x (si osservi che tale insieme esiste sempre quando i valori delle monete sono 10, 5, 1); consideriamo $M' = M^* \setminus M \cup X$, dove X è l'insieme contenente una moneta di valore x . M' è un insieme di monete di valore totale n e di cardinalità inferiore a quella di M^* : assurdo, quindi questo secondo caso non può verificarsi, e quindi M^* contiene necessariamente una moneta di valore x .

Esercizio 2 (10 punti) Dato un insieme di n numeri reali positivi e distinti $S = \{a_1, a_2, \dots, a_n\}$, con $0 < a_i < a_j < 1$ per $1 \leq i < j \leq n$, un $(2,1)$ -boxing di S è una partizione $P = \{S_1, S_2, \dots, S_k\}$ di S in k

sottoinsiemi (cioè, $\bigcup_{j=1}^k S_j = S$ e $S_r \cap S_t = \emptyset, 1 \leq r \neq t \leq k$) che soddisfa inoltre i seguenti vincoli:

$$|S_j| \leq 2 \quad \text{e} \quad \sum_{a \in S_j} a \leq 1, \quad 1 \leq j \leq k.$$

In altre parole, ogni sottoinsieme contiene al più due valori la cui somma è al più uno. Dato S , si vuole determinare un $(2,1)$ -boxing che minimizza il numero di sottoinsiemi della partizione.

1. Scrivere il codice di un algoritmo greedy che restituisce un $(2,1)$ -boxing ottimo in tempo lineare. (Suggerimento: si creino i sottoinsiemi in modo opportuno basandosi sulla sequenza ordinata.)
2. Si enunci la proprietà di scelta greedy per l'algoritmo sviluppato al punto precedente e la si dimostri, cioè si dimostri che esiste sempre una soluzione ottima che contiene la scelta greedy.

$S \rightarrow$ INSIEMI \rightarrow SOLLO DL SOTTOINSIEMI
DL SOMMA 1

$S \rightarrow [0.4 / 0.3 / 0.5 / 0.7 / 0.1]$

(BIN-PACKING) \rightarrow \textcircled{P} 2 SOTTOINSIEMI
CHE SOMMANO
AD 1

OGGETTO IN OGNI SCATOLA

GREEDY \rightarrow ① ORDINE CRESCENTE

$S \rightarrow [0.1 / 0.3 / 0.4 / 0.5 / 0.7]$

Diagram illustrating the greedy algorithm steps: 0.1 is circled and labeled 'FIRST'. 0.7 is circled and labeled 'LAST'. Arrows show 0.3 and 0.4 being added to the set containing 0.1, and 0.5 being added to the set containing 0.7. A circled '1' at the bottom indicates the total sum constraint.

FIRST = MIN

$$\text{FIRST} + \text{LAST} = 1$$

LAST = MAX

$$SS < 1 \rightarrow \text{FIRST}++$$

$$SS \geq 1 \rightarrow \text{LAST}--$$

1. L'idea è provare ad accoppiare il numero più piccolo (a_1) con quello più grande (a_n). Se la loro somma è al massimo 1, allora $S_1 = \{a_1, a_n\}$, altrimenti $S_1 = \{a_n\}$. Poi si procede analogamente sul sottoproblema $S \setminus S_1$.

(2,1)-BOXING(S)

$n \leftarrow |S|$

$P \leftarrow \text{empty_set}$

$\text{first} \leftarrow 1$

$\text{last} \leftarrow n$

while (first <= last) Si considera un ciclo per cui "first" è <= "last" (perché scansioniamo gli estremi, come detto)

if (first < last) and $a_{\text{first}} + a_{\text{last}} \leq 1$ then

$P \leftarrow P \cup \{a_{\text{first}}, a_{\text{last}}\}$

$\text{first} \leftarrow \text{first} + 1$

else

$P \leftarrow P \cup \{a_{\text{last}}\}$

$\text{last} \leftarrow \text{last} - 1$

return P

Inizializziamo l'insieme, la partizione e gli estremi.

Esempio:

1 2 3 4 5 6 7

$P = (7, 6, 5, 4, 3, 2, 1)$

Se l'estremo inf. è < dell'estremo sup. non abbiamo ancora salvato nulla in P (non abbiamo estr. inf) allora salvo in P sia l'estremo inf che l'estremo sup e incremento first. Salvandolo una volta sola, so che la somma è sempre > 1.

Altrimenti, salvo solo l'estremo superiore migliore, la cui somma è sempre >= 1

Questo algoritmo scansiona ogni elemento una sola volta, quindi la sua complessità è lineare.

GREEDY-SEL(S, f)

1 $n = S.length$

2 $A = \{a_1\}$

3 $\text{last} = 1$ // indice dell'ultima attività selezionata

4 for $m = 2$ to n

if $s_m \geq f_{\text{last}}$

6 $A = A \cup \{a_m\}$

7 $\text{last} = m$

8 return A

ALG. ITERATIVO

CONTINUATO
ATTACCA

SOL.
OTTIMA!

2. La scelta greedy è $\{a_1, a_n\}$ se $n > 1$ e $a_1 + a_n \leq 1$, altrimenti $\{a_n\}$. Ora dimostriamo che esiste sempre una soluzione ottima che contiene la scelta greedy. I casi $n = 1$ e $a_1 + a_n > 1$ sono banali, visto che in questi casi ogni soluzione ammissibile deve contenere il sottoinsieme $\{a_n\}$. Quindi assumiamo che la scelta greedy sia $\{a_1, a_n\}$. Consideriamo una qualsiasi soluzione ottima dove $a_1 \in S_1$ e $a_n \in S_2$. Sostituiamo questi due sottoinsiemi con $S'_1 = \{a_1, a_n\}$ (cioè, la scelta greedy) e $S'_2 = S_1 \cup S_2 \setminus \{a_1, a_n\}$. $|S'_2| \leq 2$ e, se $|S'_2| = 2$, allora $S'_2 = \{a_s, a_t\}$ con $a_s \in S_1$ e $a_t \in S_2$. Siccome a_t era precedentemente accoppiato con a_n , a maggior ragione può essere accoppiato con $a_s < a_n$, quindi la nuova soluzione così creata è ammissibile e ancora ottima.

Spiegata in termini semplici: la scelta greedy consiste nel scegliere quello che sta agli estremi.

Esiste sempre una soluzione ottima in quanto il sottoinsieme che consideriamo è sempre formato da almeno due elementi (tolto il caso base) per cui tra questi ci siano i nostri estremi inf. e sup.

Andando a fare le differenze ottime (scegliendo di volta in volta gli estremi migliori), di sicuro

avremo ancora una differenza ottima, perché scegliamo gli estremi migliori (più piccolo per inf. e più grande per sup) in ogni momento.

(2)
=
DUE
GREEDY
CORRE
OTTIMO

2-1 BOXING → 4/5 APPROX...

GREEDY

HUFFMAN (TEORIA + ESERCIZI)

ALGORITMI

BALDAN/
XQUER

SCUOLAZZI → 1

→ BALDAN → 2

PROG.
DINAMICA