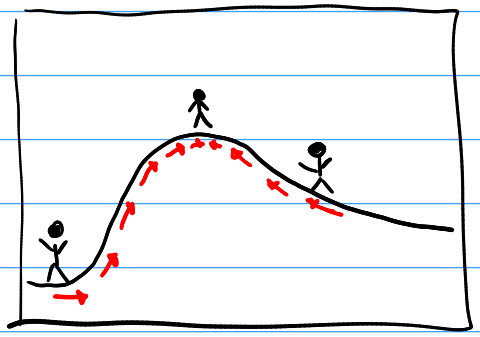
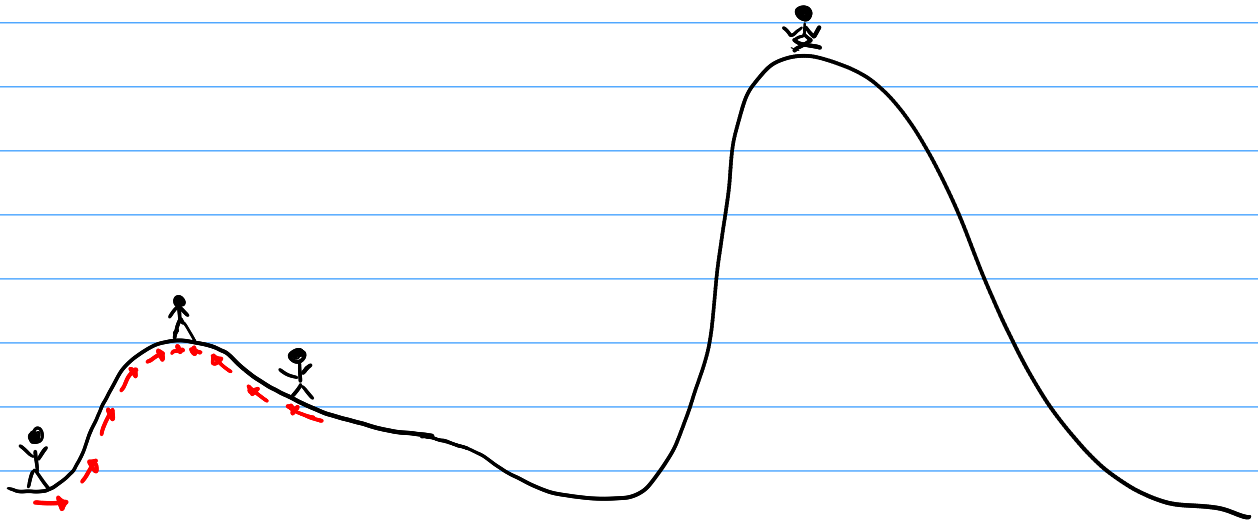
Algoritmi Greedy

- letteralmente: avidi
- ma anche: golosi, miopi ....



Una strategia per raggiungere il punto più alto di un territorio è quello di SALIRE ovvero quello di muoversi in modo ascendente lungo una PENDENZA ↗

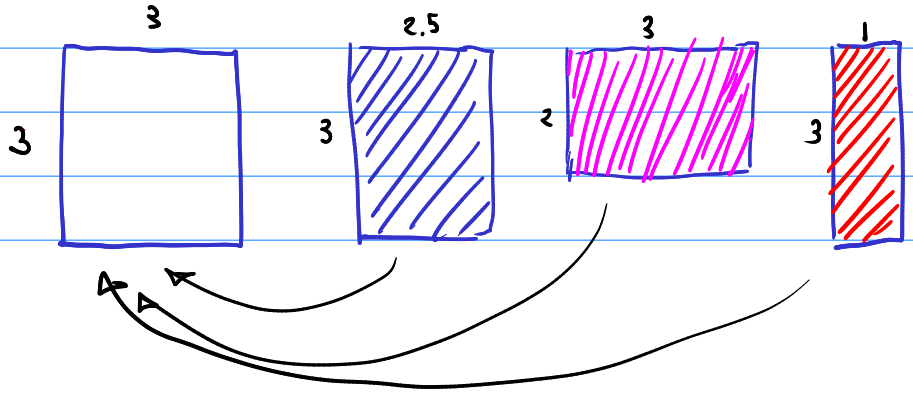
Ma le cose ovviamente non sono così semplici ...



a volte bisogna rinunciare a quello che sembra immediatamente vantaggioso ma che porta ad allontanarsi dai risultati migliori.

UNA SCELTA OTTIMALE NEL BREVE TERMINE PUÓ ALLONTANARE DAL RISUTATO OTTIMO FINALE

E.g.

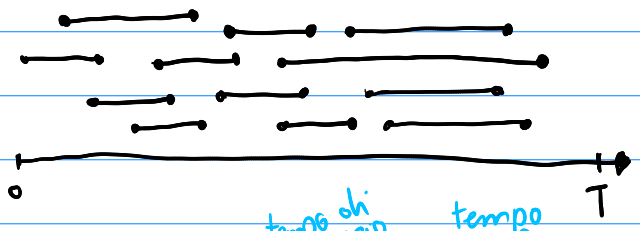


Non conviene mettere l'oggetto più grande nella scatola

- Esistono tuttavia dei problemi per cui ragionamenti "miopi" permettono di ottenere una soluzione ottima

- Perché utilizzarli? Perché i ragionamenti "miopi" sono più **facili** → semplici → efficienti

Caso di Studio: SELEZIONE DI ATTIVITÁ (cap 16.1)



Abbiamo una serie di ATTIVITÁ che, se eseguite, occupano una risorsa esclusiva in un determinato lasso di tempo (a_i, b_i)

tempo di inizio, tempo di fine

Input: $\{(a_1,b_1),(a_2,b_2)\dots(a_n,b_n)\}$ con $0 \le a_i < b_i \le T \quad \forall\, 0 \le i \le n$

- Vogliamo eseguire il maggior numero di attività
- Non possiamo eseguire due attività i cui intervalli si sovrappongano

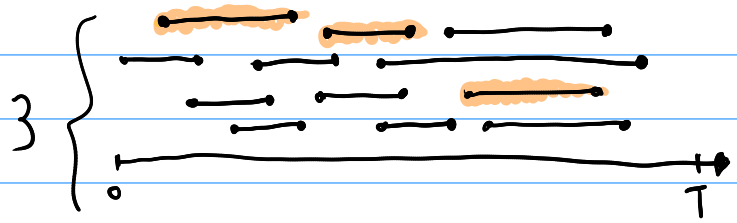
Goal:

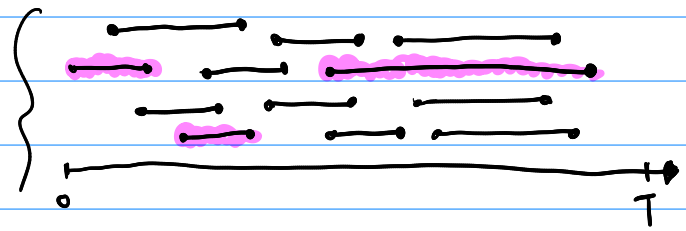
Trovare una sequenza

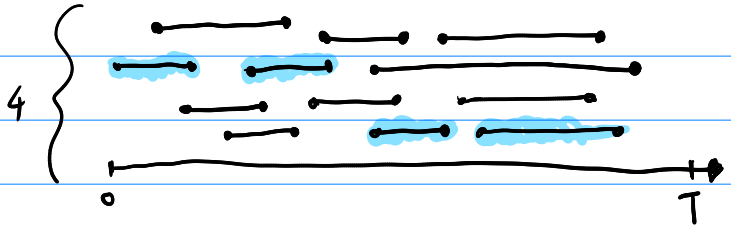
$(i_1, i_2, \dots, i_t)$

di lunghezza massima per cui

$a_{i_1} < b_{i_1} < a_{i_2} < b_{i_2} < \dots < a_{i_t} < b_{i_t}$







Def $P_{s,e}$ il sottoproblema ristretto a tutte le attività che hanno $s \le a_i, b_i \le e$

In pratica il problema originale, ossia tutti gli $0 \le a_i, b_i \le T$, è $P_{0,T}$

SOTTOSTRUTTURA OTTIMA (UNA CARATTERISTICA CHE VEDREMO ANCHE E SOPRATTUTTO PER LA PROGRAMMAZIONE DINAMICA)

- Se S è una soluzione del problema, in certi casi è possibile descrivere S come

$$S', S'', \dots \text{ unito ad altri elementi } e_1 \dots e_t$$

$S', S'', \dots$ sono SOLUZIONI OTTIME DI SOTTOPROBLEMI

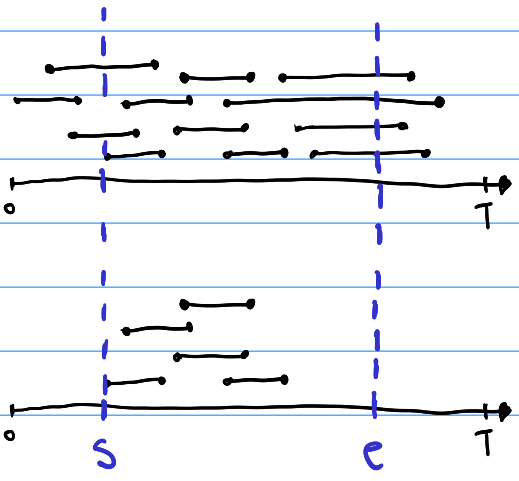
e allora

$$\text{costo}(S) = \text{costo}(S') + \text{costo}(S'') + \dots + \text{costo}(e_1, \dots, e_t)$$

questo permette una soluzione ricorsiva. Trovata la giusta decomposizione, si risolvono i sottoproblemi e si costruisce la soluzione S.

SELEZIONE DELLE ATTIVITÀ

- $P_{0,T}$ problema di selezione di attività in $\{(a_1,b_1), \dots, (a_n,b_n)\}$
- $P_{s,e}$ = selezione di attività $\{(a_i,b_i) \; s \le a_i < b_i \le e\}$



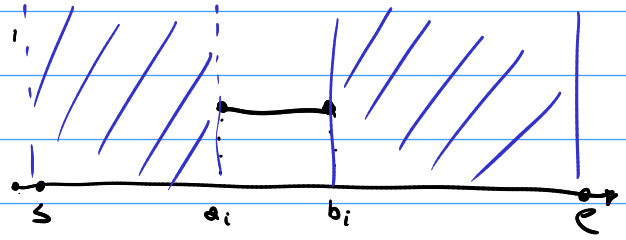
La **soluzione OTTIMA** $S_{s,e} \subseteq P_{s,e}$ ha due casi

(soluzione ↑ problema)

1. $S_{s,e}$ è vuoto (se e solo se $P_{s,e}$ è vuoto)
2. $(a_i,b_i) \in S_{s,e}$ e allora $S_{s,e} = S_{s,a_i} \cup \{(a_i,b_i)\} \cup S_{b_i,e}$

$S_{s,e}$ non vuota

$$S_{s,e} = S_{s,a_i} \cup \{(a_i, b_i)\} \cup S_{b_i,e} \quad \text{per qualche } (a_i, b_i)$$



poiché nessuna soluzione può aveve attività che si sovrappongono

- Possono S_{s,a_i} e $S_{b_i,e}$ essere soluzioni SUBOTTIMALI?

  NO! Perché se lo fossero potremo trovarne di migliori S'_{s,a_i} e $S'_{b_i,e}$ ottenendo $S'_{s,e} = S'_{s,a_i} \cup \{(a_i, b_c)\} \cup S'_{b_i,e}$ migliore di $S_{s,e}$

  e allora $S_{s,e}$ NON SAREBBE OTTIMA.

Tentativo di soluzione #1

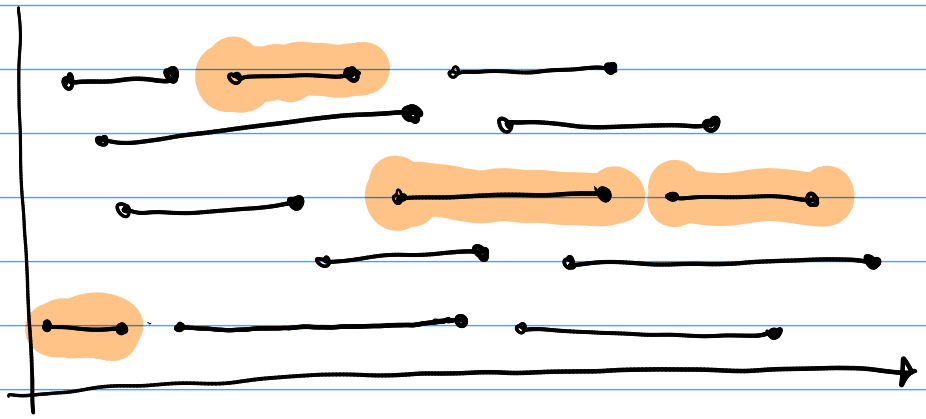
- Per trovare una soluzione ottima di $P_{s,e}$
  - Per ogni (a_i, b_i) risolvo P_{s,a_i} e $P_{b_i,e}$ → S_{s,a_i} e $S_{b_i,e}$
  - tengo la migliore tra le soluzioni $S_{s,a_i} \cup \{(a_i, b_i)\} \cup S_{b_i,e}$

ESERCIZIO Mostrare che la soluzione ottima viene trovata con questo metodo

POSSIAMO FARE DI MEGLIO: UNA SCELTA "GREEDY"

- L'idea precedente si basa sullo scegliere un'attività e poi costruirci attorno il resto della soluzione
- Alcune scelte potrebbero essere fallimentari COMPROMETTENDO DEFINITIVAMENTE LA SOLUZIONE FINALE OTTENUTA
- Per il problema della SELEZIONE DI ATTIVITÁ abbiamo una strategia "greedy"... (suspence...)

"Prendiamo l'attività, tra quelle a disposizione, che FINISCE PRIMA"
(e ripetiamo)



Perché la strategia greedy ci dà una soluzione ottima?

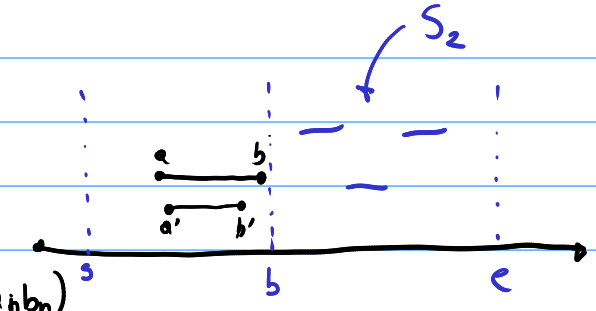
- Se S è una soluzione ottima di $P_{s,e}$ (assumiamo non vuoto), definiamo
  - (a,b) l'attività in S con b minimo
  - (a',b') l'attività in $P_{s,e}$ con b' minimo (quella che avremmo scelto con la strategia greedy)

allora $S = \{(a,b)\} \cup S_2$ (soluzione di $P_{b,e}$)

poiché $b' \leq b$ allora

$S' = \{(a',b')\} \cup S_2$ è una soluzione valida per $P_{s,e}$

$|S'| = 1 + |S_2| = |S|$ quindi S' è ottima



Algoritmo per "selezione di attività"

1. Ordiniamo le attività in maniera non decrescente rispetto alle b_i ottenendo $(a_1,b_1)(a_2,b_2)\dots(a_n,b_n)$ con $b_1 < b_2 < b_3 \dots < b_n$
2. $S := \emptyset$, end := 0
3. for i in 1...n:
   if $a_i \geq$ end:
   $S := S \cup \{(a_i,b_i)\}$
   end := b_i
4. return S

ESERCIZIO
Stimare la complessità dell'algoritmo

ESERCIZIO

L'algoritmo si basa sul fatto che un'attività valga l'altra. Cioè che sostituendo un'attività con un'altra la soluzione abbia la stessa VALORE.

- CONSIDERIAMO L'ESEMPIO DI UN NOLEGGIO DI UN B&B CON RELATIVI PERIODI DI PRENOTAZIONE.
- OGNI ATTIVITÀ (a_i, b_i) HA UN RICAVO v_i

Vero o FALSO? L'algoritmo massimizza anche il ricavo
- Vero? DIMOSTRATELO
- FALSO? Mostrate UN CONTRO ESEMPIO

ESERCIZIO 16.1-2 e 16.1-3

L'algoritmo si basa sulla strategia di prendere, tra tutte le attività compatibili con quelle già scelte, quella che:
- termina prima

e se invece sceglissimo l'attività
1. che dura di meno
2. che inizia più tardi
3. che inizia per prima
4. che si sovrappone al minor numero di altre attività compatibili

QUALI DI QUESTE STRATEGIE FUNZIONANO?

ALGORITMI GREEDY IN GENERALE

Un algoritmo greedy costruisce, tipicamente, una soluzione in modo incrementale.

- $S = \emptyset$ all'inizio e ad ogni passo $S := S \cup \{e\}$

Si mantiene l'<u>invariante</u> che "S è contenuto in una soluzione finale ottima".

> Abbiamo S soluzione parziale $\subseteq T$ soluzione ottima
> - scegliamo e secondo una strategia greedy localmente ottima
> - se $S \cup \{e\} \not\subseteq T$ allora deve esistere T' ottimo per cui $S \cup \{e\} \subseteq T$

- Non si torna mai indietro a modificare la soluzione parziale scelta ai passi precedenti

<u>Nota</u> il cap 16.4 mostra come la struttura degli algoritmi greedy sia intimamente collegata ai MATROIDI.

- Vedremo come utilizzare ALGORITMI GREEDY in vari ambiti. In particolare
  - calcolo di alberi di copertura minimi

Algoritmo di Prim — Algoritmo di Kruskal