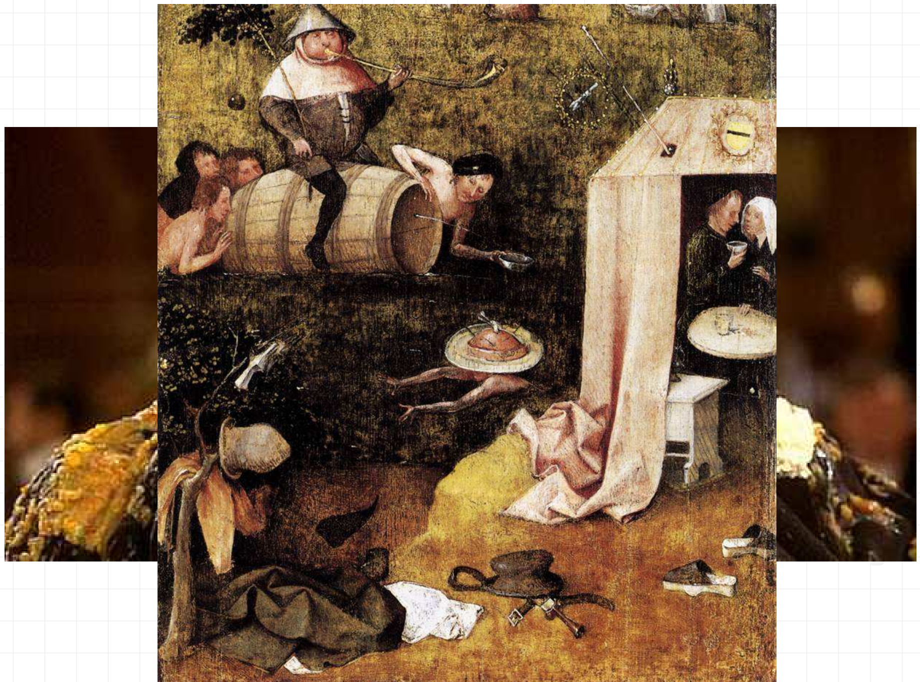


Algoritmi Greedy

Greedy

Cerchiamo la soluzione di un problema di ottimizzazione (cioè di un problema in cui alcune soluzioni sono preferibili ad altre)

La ricerca esaustiva è impraticabile.

Si cerca di costruire una soluzione, o di migliorarne una esistente, facendo una serie di aggiustamenti (*mosse*) successivi.

Ogni mossa è determinata solo sulla base di criteri locali.

Se si è fortunati, una sequenza di scelte localmente ottime può portare all'ottimo globale, in questo caso il problema ha la proprietà della scelta greedy (*greedy choice property*).

Problemi di questo tipo esistono, anche se sono rari. Ne vedremo uno anche parlando di alberi di copertura.

Selezione di attivitá

S={1,...,n} insieme di attività. Ogni attività ha un tempo di inizio s_i e un tempo di fine f_i.

Problema: *Selezionare* un sottoinsieme S' di attività in modo tale che:

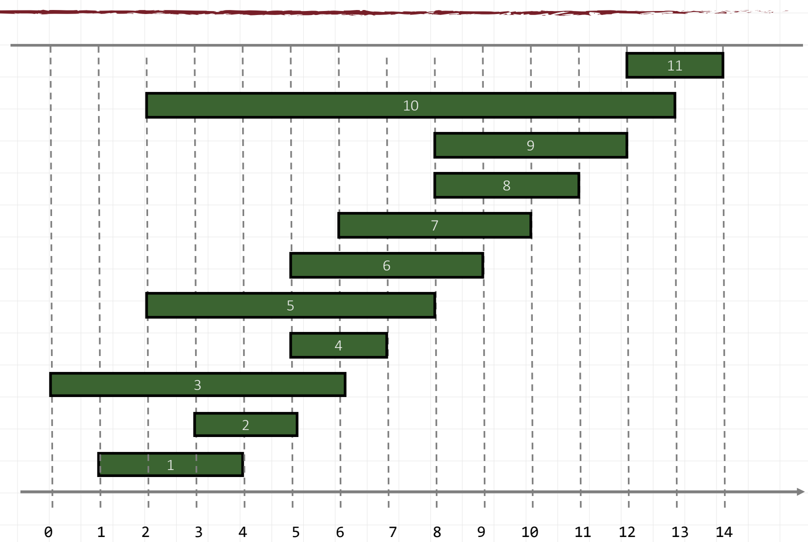
1. se *i* e *j* appartengono a *S'* allora: $s_i \geq f_j$ oppure $s_j \geq f_i$.
2. La cardinalità di S' è massimizzata.

Esempio, insieme S di attività:

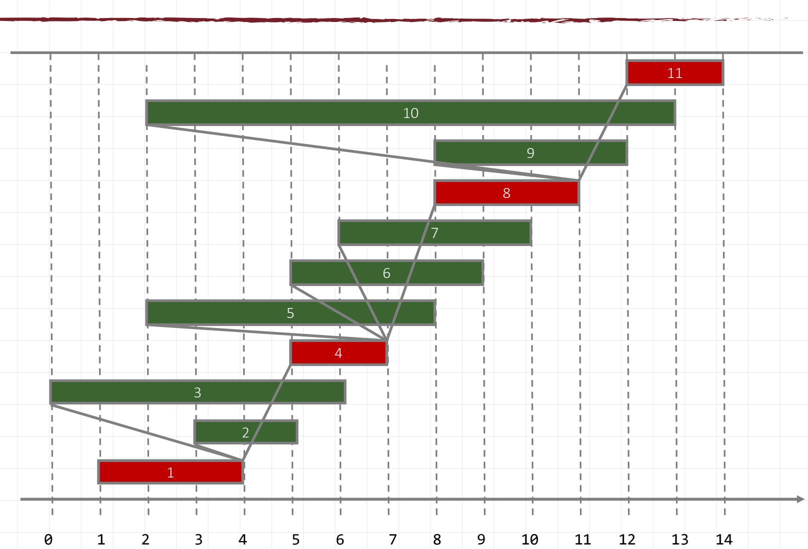
i	1	2	3	4	5	6
S_i	1	3	2	3	6	3
F_i	3	4	5	5	7	9

- a_1 e a_3 non sono compatibili
- $\{a_1, a_6\}$ è un insieme di attività mutuamente compatibili
- $\{a_1, a_4, a_5\}$ è un insieme massimale di attività compatibili
- $\{a_1, a_2, a_5\}$ è un insieme massimale di attività compatibili
- $\{a_1, a_4, a_5\}$ è una soluzione per S

Nota: l'insieme S è ordinato per tempi di fine crescenti



Attività ordinate in base a f_i



Pseudocodice

```
Greedy-activity-selector(s,f)
   n=length(s)
   A={1}
   j=1
   for i=2 to n
   do if s_i≥f_j
      then A=A∪{i}
           j=i
   return A
```

$\Theta(n)$

Dimostrazione di correttezza

Assumiamo che le attività siano ordinate per tempi di fine crescenti.

Dimostriamo che esiste una soluzione ottima che contiene l'attività 1 che è quella che termina per prima (con il tempo di fine più piccolo).

Supponiamo per assurdo che esista una soluzione S'' migliore di S' (S' contiene l'attività *1* ed è stata costruita con l'algoritmo greedy).

Assumiamo che sia *S'* che *S''* siano ordinate per tempi di fine crescenti.

$S'=\{1,.....\}$ e $S''=\{k,......\}$ dove $1\neq k$.

Sia $T=S''-\{k\}\cup\{1\}$. *T* quindi è la migliore soluzione possibile e contiene l'attività *1*.

Adesso eliminiamo da *S* tutte le attività che sono incompatibili con l'attività *1* e ripetiamo la dimostrazione da capo.

Scelte greedy

A volte gli algoritmi greedy non trovano la soluzione ottima.

La struttura del problema è tale per cui nessun algoritmo greedy può garantire di trovare sempre la soluzione ottima.

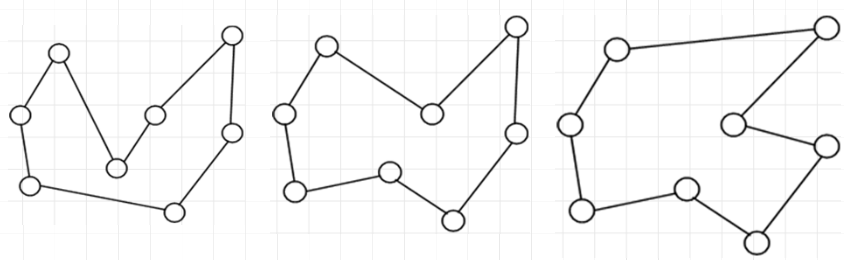
D'altro canto però algoritmi non greedy che garantiscono di trovarla hanno un costo computazionale proibitivo.

In questi casi spesso è necessario utilizzare algoritmi che *fanno il meglio che possono con le risorse a disposizione*: ***algoritmi euristici*** (molto da dire, in corsi futuri).

Esempio: problema del commesso viaggiatore. Dato un insieme di *n* città occorre trovare la strada più breve per visitarle tutte una ed una sola volta e tornare alla città di partenza.

TSP

Qual'è il giro più breve?



Non è sempre semplice capirlo …

TSP: storia

Il TSP come lo intendiamo oggi è stato studiato per primo negli anni 30 dal matematico Karl Menger a Vienna.

Problemi matematici correlati al TSP erano già stati introdotti nell'800 dal matematico irlandese Sir William Rowan Hamilton.

In figura il Gioco Icosiano di Hamilton, che chiede ai giocatori di completare un giro fra 20 pioli, usando un apposito spago millimetrato.

(http://www.math.princeton.edu/tsp/index.html)

Nel passato …

Ulysses 16, dati

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	509	501	312	1019	736	656	60	1039	726	2314	479	448	479	619	150
2	126	474	1526	1226	1133	532	1449	1122	2789	958	941	978	1127	542	
3	541	1516	1184	1084	536	1371	1045	2728	913	904	946	1115	499		
4	1157	980	919	271	1333	1029	2553	751	704	720	783	455			
5	478	583	996	858	855	1504	677	651	600	401	1033				
6	115	740	470	379	1581	271	289	261	308	687					
7	667	455	288	1661	177	216	207	343	592						
8	1066	759	2320	493	454	479	598	206							
9	328	1387	591	650	656	776	933								
10	1697	333	400	427	622	610									
11	1838	1868	1841	1789	2248										
12	68	105	336	417											
13	52	287	406												
14	237	449													
15	636														

1	Ithaca	38.24N	20.42E
2	Troy	39.57N	26.15E
3	Maronia	40.56N	25.32E
4	Malea	36.26N	23.12E
5	Djerba	33.48N	10.54E
6	Favignana	37.56N	12.19E
7	Ustica	38.42N	13.11E
8	Zakinthos	37.52N	20.44E
9	Bonifaccio	41.23N	9.10E
10	Circeo	41.17N	13.05E
11	Gibraltar	36.08N	5.21W
12	Stromboli	38.47N	15.13E
13	Messina	38.15N	15.35E
14	Taormina	37.51N	15.17E
15	Birzebbuga	35.49N	14.32E
16	Corfu	39.36N	19.56E

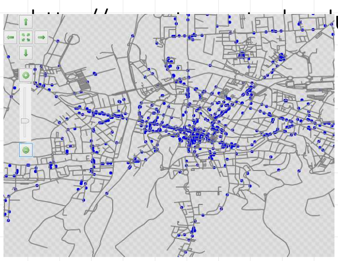
Attualmente

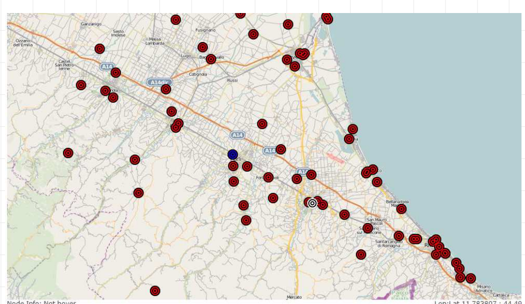
Viaggi di persone o di veicoli (consegne a clienti)

Dati: grafo stradale, alcuni nodi rappresentano clienti.

Trovare: giro più breve fra i clienti.

Precondizione: Problemi di cammino minimo ⇒ matrice delle distanze. Definizione di un overlay





Metodi di soluzione

n nodi $\Rightarrow$ $(n\text{-}1)!$ possibili tour

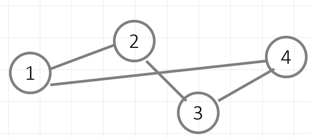
n	5	10	15	20
n!	120	3,628.800	1,31*10^12	2,43*10^18

Metodi di soluzione (alcuni):

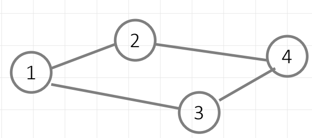
- Enumerazione completa
- IP: Branch and Bound (and Cut, and Price, …)
- Programmazione dinamica
- Algoritmi approssimati
- Euristiche (greedy)
- Metaeuristiche

Scelte greedy, TSP

Algoritmo greedy: parto dalla città 1 e poi procedo visitando la città più vicina non ancora visitata.
Quando tutte le città sono state visitate torno alla città 1.



Soluzione con algoritmo greedy



Soluzione ottima

Non sempre una soluzione di ottimo *globale* si ottiene facendo una serie di scelte *localmente ottime*.

Per il problema del commesso viaggiatore nessuna politica decisionale greedy fornisce una soluzione finale ottima.

Scelte greedy

Proprietà della scelta greedy.

Ad ogni passo l'algoritmo compie una scelta in base ad una certa politica ed alle scelte compiute fino a quel momento. Cosi facendo ci si riduce ad un sottoproblema di dimensioni più piccole. Ad ogni passo si calcola un pezzo della soluzione.

Sottostruttura ottima.

Per poter applicare con successo un algoritmo greedy è necessario (ma non sufficiente) che la soluzione ottima contenga le soluzioni ottime dei sottoproblemi.

Knapsack

O problema dello zaino, problema del ladro …

Problema dello zaino, 2 versioni

Knapsack 0-1: Un ladro durante una rapina si trova davanti a n oggetti. Ogni oggetto i ha un valore v_i e un peso w_i (numeri interi). Il ladro ha uno zaino che può contenere fino a W (numero intero) chilogrammi di refurtiva. Il ladro deve scegliere quali oggetti rubare per massimizzare il valore complessivo degli oggetti rubati.

Knapsack: In questo caso il ladro può anche prendere una parte frazionaria degli oggetti. Non è costretto a "prendere o lasciare" un oggetto, può decidere di prenderne un pezzo grande a suo piacimento.

Nota: Knapsack è una generalizzazione di Knapsack 0-1

Sottostruttura ottima

Entrambe le versioni del knapsack soddisfano tale proprietà.

Supponiamo infatti che il ladro possa rubare refurtiva avente peso W' non maggiore di W, di valore massimo V.

Se togliamo dallo zaino l'oggetto j otteniamo la soluzione ottima del sottoproblema in cui lo zaino può contenere al massimo $W\text{-}w_j$ chilogrammi ottenuti mettendo insieme oggetti da un insieme di $n\text{-}1$ (abbiamo eliminato l'oggetto j).

Knapsack, soluzione greedy

Idea:

il ladro prende la quantità più grande possibile dell'oggetto *i* tale per cui v_i/w_i (valore per unità di peso) è massimo. Dopodiché, se nello zaino c'è ancora posto, ripete l'operazione.

€ 60 | 10 | 6 € al chilo

€ 100 | 20 | 5 € al chilo

€ 120 | 30 | 4 € al chilo

Zaino da 50 chili

Soluzione ottima di knapsack con algoritmo greedy

10 | 20 | 20 = 2/3 di 30

€ 240

Knapsack 0-1, no greedy

€ 60 | 10 | 6 € al chilo

€ 100 | 20 | 5 € al chilo

€ 120 | 30 | 4 € al chilo

Zaino da 50 chili

Soluzione ottima di knapsack 0-1

30 | 20

€ 220

Soluzione di knapsack 0-1 scegliendo per primo l'oggetto da 6 € al chilo (valore per unità di peso massimo)

10 | 30

€ 180