Algoritmi e Strutture Dati 2

- Naturale continuazione del corso di Algoritmi 1
- Obiettivi:
  - studiare algoritmi e strutture dati fondamentali;
  - studiare le tecniche per risolvere algoritmicamente alcune classi rilevanti di problemi;
  - sviluppare una sensibilità per correttezza ed efficienza dei programmi

1

Programma

- Algoritmi golosi (greedy)
- Analisi ammortizzata della complessità
- Strutture dati avanzate per insiemi dinamici
  - B-alberi (memoria di massa)
  - Heap (con unione)
  - Insiemi disgiunti

2

Programma (cont.)

- Algoritmi su grafi
  - Visite (in ampiezza e profondità)
  - Ordinamento topologico
  - Componenti fortemente connesse e albero di connessione minimo
  - Cammini minimi
  - Reti di flusso

3

Docenti

- Paolo Baldan
  - Ricevimento: Giovedì 14:30-16:30
  - Email: baldan@math.unipd.it
- Brent Venable
  - Ricevimento: Mercoledì 14:30-16:30
  - Email: kvenable@math.unipd.it

4

Modalità d'esame

- Scritto
  Esercizi sui temi del corso
- Verifica Orale
  Opzionale (~ discussione dello scritto).

5

Materiale didattico

- Testo
  Cormen, Leiserson, Rivest, Stein.
  Introduzione agli Algoritmi e Strutture Dati
  McGraw-Hill
- Pagina del corso
  http://www.math.unipd.it/~baldan/Alg2/Alg2.html
  Slide, informazioni sul programma svolto, comunicazioni ecc.

6

Algoritmi golosi

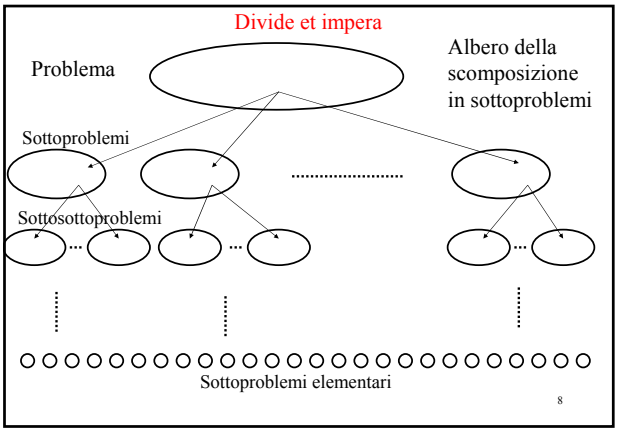
Tecniche di soluzione dei problemi in Algoritmi 1:

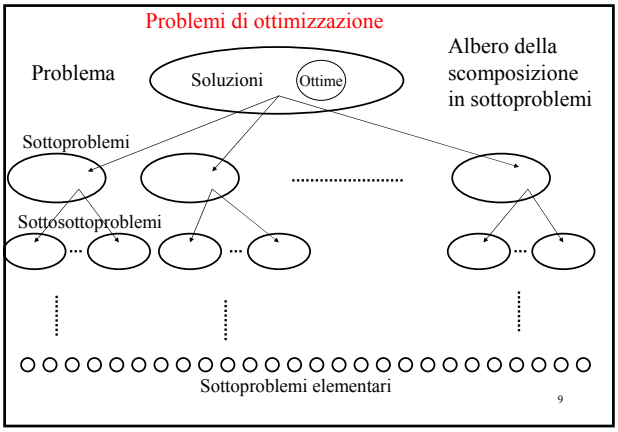
- Divide et impera
- Programmazione dinamica

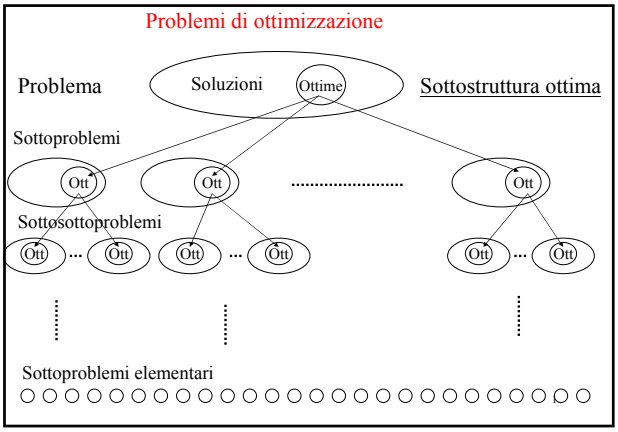
Nuova tecnica:

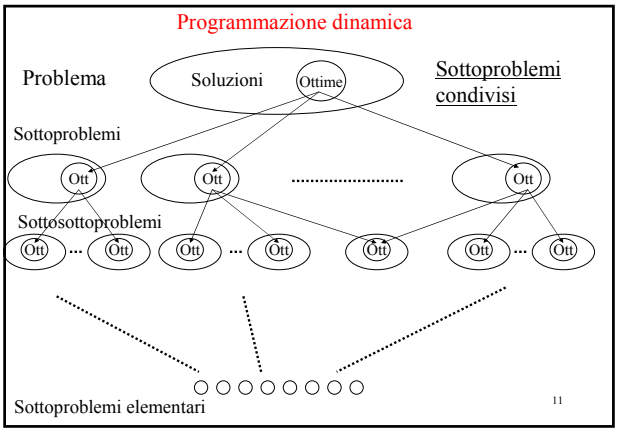
- Algoritmi golosi

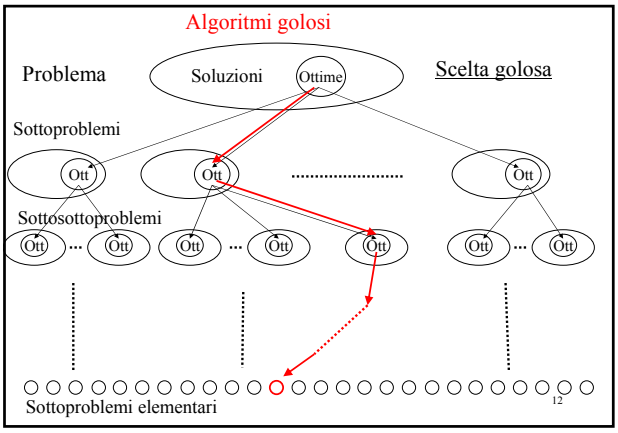
7











1) ogni volta si fa la scelta che sembra migliore localmente;

2) in questo modo per alcuni problemi si ottiene una soluzione globalmente ottima.

13

Problema della scelta delle attività

n *attività* $a_1,\ldots,a_n$ usano la stessa risorsa (es: lezioni da tenere in una stessa aula).

Ogni attività a_i ha un *tempo di inizio* s_i ed un *tempo di fine* f_i con $s_i < f_i$.

a_i occupa la risorsa nell'intervallo temporale $[s_i, f_i)$.

a_i ed a_j sono *compatibili* se $[s_i, f_i)$ ed $[s_j, f_j)$ sono disgiunti.

Problema: scegliere il massimo numero di attività compatibili.

14

Storiella Golosa

Personaggi:

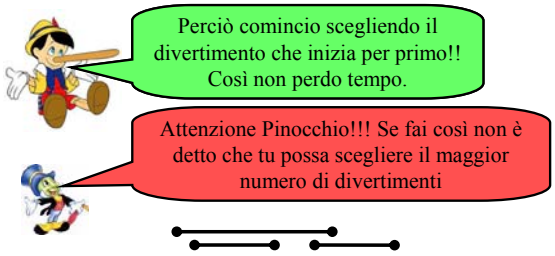
Pinocchio  L'algoritmo goloso

Il grillo  Controlla la correttezza.

La fatina  Conosce il futuro.

15

Pinocchio (l'algoritmo) arriva nella Città dei Balocchi dove può scegliere i divertimenti che preferisce.
Ogni divertimento ha un'ora di inizio ed una durata.



16

17

18

Io conosco una soluzione ottima S_{INV} ma non la mostro a nessuno.

Scelgo il divertimento “D” che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.

Insegnerò alla fatina come modificare la sua soluzione ottima in modo che contenga “D”.

19

Io conosco una nuova soluzione ottima S_{INV} che contiene “D”.

Allora scelgo il divertimento “D” che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.

Primo caso:

Ora so che la fatina conosce una soluzione ottima che contiene il divertimento “D”.

20

Io conosco una soluzione ottima S_{INV} ma non la mostro a nessuno.

Allora scelgo il divertimento “D” che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.

Secondo caso:

Cara fatina, se la tua soluzione non contiene il divertimento “D” metti “D” al posto del primo divertimento.

21

D_1 D_2 ..... D_m

D D_2 ..... D_m

22

Io conosco una nuova soluzione ottima S_{INV} che contiene “D”.

Allora scelgo il divertimento “D” che termina per primo!! Così quando ho finito mi rimane più tempo per gli altri.

Secondo caso:

So che la fatina conosce una soluzione ottima che contiene il divertimento “D”.

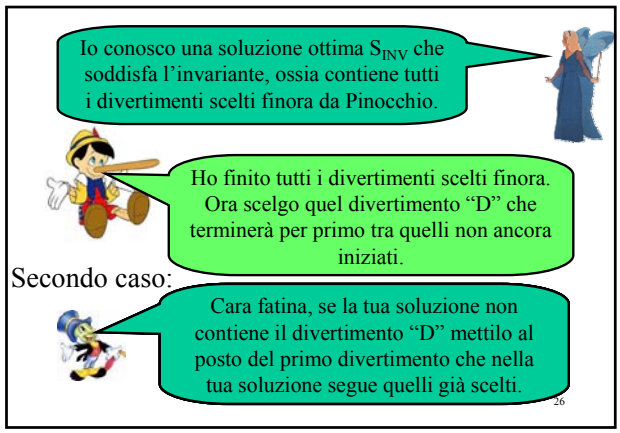
23

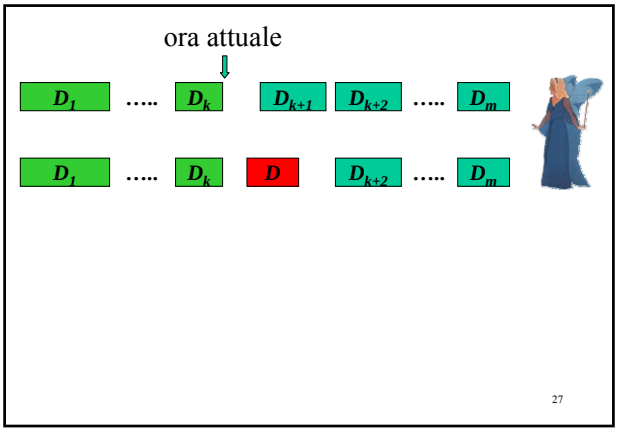
Io conosco una soluzione ottima S_{INV} che soddisfa l’invariante, ossia contiene tutti i divertimenti scelti finora da Pinocchio.

Ho finito tutti i divertimenti scelti finora. Ora scelgo quel divertimento “D” che terminerà per primo tra quelli non sono ancora iniziati.

Insegnerò alla fatina come modificare la sua soluzione ottima S_{INV} in modo che contenga anche “D”.

24





Problema della scelta delle attività

n *attività* $a_1,\ldots,a_n$ usano la stessa risorsa (es: lezioni da tenere in una stessa aula).

Ogni attività a_i ha un *tempo di inizio* s_i ed un *tempo di fine* f_i con $s_i < f_i$.

a_i occupa la risorsa nell'intervallo temporale $[s_i, f_i)$.

a_i ed a_j sono *compatibili* se $[s_i, f_i)$ ed $[s_j, f_j)$ sono disgiunti.

Problema: scegliere il massimo numero di attività compatibili.

30

Strategia che funziona

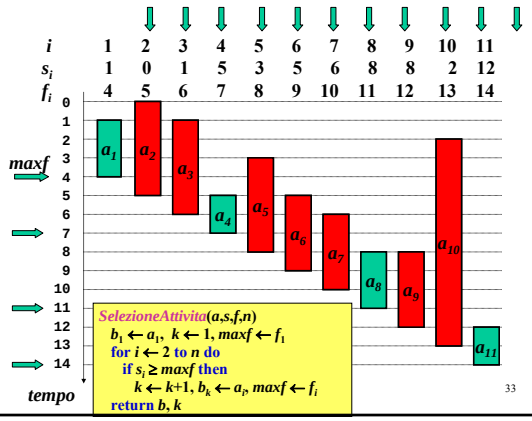
```
SelezioneAttivita(A)
 AttScelte ← ϕ,  AttComp ← A
 while AttComp ≠ ϕ do
   - scegli l'attività a ∈ AttComp che termina per prima
   - aggiungi a ad AttScelte
   - togli da AttComp tutte le attività incompatibili con a
 return AttScelte
```

31

Per implementarla supponiamo le attività $a_1,\ldots,a_n$ ordinate per tempo di fine non decrescente $f_1 \le \ldots \le f_n$. Altrimenti le possiamo ordinare in tempo $O(n \log n)$.

```
SelezioneAttivita(a, s, f, n)        ► f1 ≤ ... ≤ fn
  b1 ← a1,  k ← 1,  maxf ← f1
  for i ← 2 to n do
    if si ≥ maxf then
      k ← k+1, bk ← ai, maxf ← fi
  return b, k
```

32



33

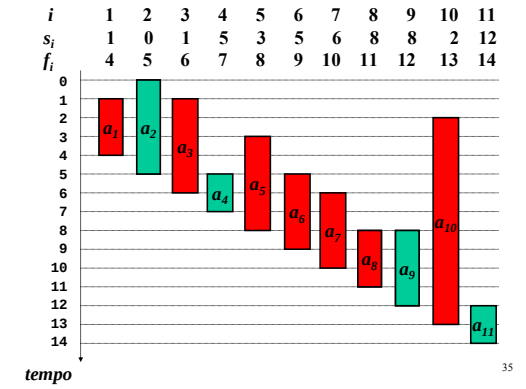
La soluzione trovata contiene quattro attività.

Due domande:

1) La soluzione trovata con l'algoritmo goloso è l'unica possibile che contiene quattro attività?

2) La soluzione trovata con l'algoritmo goloso è ottima o esistono anche soluzioni con più di quattro attività?

34



35

Cerchiamo di rispondere alla seconda domanda.

2) La soluzione trovata con l'algoritmo goloso è ottima o esistono anche soluzioni con più di quattro attività?

```
SelezioneAttivita(a, s, f, n)        ► f1 ≤ ... ≤ fn
  b1 ← a1,  k ← 1, maxf ← f1
  for i ← 2 to n do
    if si ≥ maxf then
      k ← k+1, bk ← ai, maxf ← fi
  return b, k
```

36

Invariante: Ad ogni stadio di esecuzione dell'algoritmo, la soluzione parziale corrente $b_1, \ldots, b_k$ si può estendere ad una soluzione ottima.

37

Base: L'algoritmo comincia con scegliere la prima attività a_1 (quella con tempo di fine minimo).

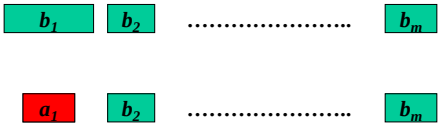
Siamo sicuri che questa scelta non possa compromettere il risultato?

In altre parole: esiste sempre una soluzione ottima che contiene a_1?

38

La risposta è affermativa.

Sia $b_1, \ldots, b_m$ una qualsiasi soluzione ottima (ne esiste certamente almeno una) che supponiamo ordinata per tempo di fine.



39

1. le attività $b_2, \ldots, b_m$ sono compatibili con b_1 e terminano dopo b_1.
2. Esse hanno quindi tempo di inizio maggiore o uguale al tempo di fine f di b_1.
3. Ma a_1 ha il tempo di fine f_1 minimo in assoluto e quindi termina prima di b_1.
4. Quindi anche a_1 è compatibile con $b_2, \ldots, b_m$.

Dunque $a_1, b_2, \ldots, b_m$ è una soluzione ottima che contiene a_1
(NB: a_1 è certamente $\neq$ da $b_2, \ldots, b_m$ … perche?)

40

maxf viene posto ad f_1 ed aggiornato ad f_i ogni volta che si seleziona una nuova attività a_i.

```
SelezioneAttivita(a, s, f, n)
  b_1 ← a_1,  k ← 1, maxf ← f_1
  for i ← 2 to n do
    if s_i ≥ maxf then
      k ← k+1, b_k ← a_i, maxf ← f_i
  return b, k
```

Siccome le attività sono ordinate per tempo di fine non decrescente, ***maxf*** è il massimo tempo finale delle attività precedentemente selezionate.

41

Con il test:

```
if s_i ≥ maxf then
  k ← k+1, b_k ← a_i, maxf ← f_i
```

l'algoritmo seleziona la prima attività a_i il cui tempo di inizio s_i è maggiore o uguale di ***maxf***.

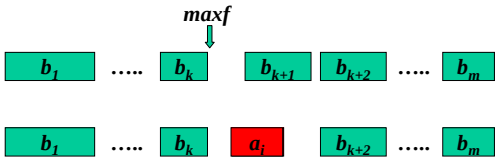
Siamo sicuri che questa scelta non comprometta il risultato?

In altre parole: esiste sempre una soluzione ottima che contiene a_i e le attività $b_1, \ldots, b_k$ finora scelte?

42

La risposta è ancora affermativa.

Assumiamo induttivamente che esista una soluzione ottima $b_1, ..., b_k, b_{k+1}, ..., b_m$ che estende le attività $b_1, ..., b_k$ finora scelte e supponiamo $b_1, ..., b_k$ e $b_{k+1}, ..., b_m$ ordinate per tempo di fine.



43

1. Le attività scartate finora iniziano prima e hanno tempo di fine maggiore o uguale ad una delle attività precedentemente scelte.
2. Esse sono quindi incompatibili con almeno una delle attività $b_1, ... ,b_k$ finora scelte.
3. $b_{k+1}, ... ,b_m$ sono invece compatibili con tutte le attività $b_1, ... ,b_k$ e quindi non sono tra quelle scartate precedentemente.
4. $b_{k+1}, ... ,b_m$ hanno sia tempo di fine che tempo di inizio maggiore o uguale di $maxf$.

44

5. L'attività a_i è quella con tempo di fine f_i minimo in assoluto tra quelle compatibili con $b_1, ... ,b_k$ e quindi $f_i \leq f$.
6. Siccome b_{k+1} è compatibile con $b_{k+2}, ... ,b_m$ i tempi di inizio di $b_{k+2}, ... ,b_m$ sono maggiori o uguali di f e quindi anche di f_i.
7. Dunque anche a_i è compatibile con $b_{k+2}, ... ,b_m$
8. Pertanto $b_1, ... ,b_k, a_i, b_{k+2}, ... ,b_m$ è una soluzione ottima contenente a_i e $b_1, ... ,b_k$.

45

Sappiamo quindi che durante tutta l'esecuzione dell'algoritmo vale l'invariante: esiste sempre una soluzione ottima contenente le attività $b_1, ... , b_k$ scelte fino a quel momento.

Quando l'algoritmo termina non ci sono altre attività compatibili con $b_1, ... , b_k$ e quindi le attività $b_1, ... , b_k$ sono una soluzione ottima.

46

L'algoritmo è *goloso* perchè ad ogni passo, tra tutte le attività compatibili con quelle già scelte, sceglie quella che termina prima.

Questa scelta è localmente ottima (golosa) perché è quella che lascia più tempo a disposizione per le successive attività.

47

Esercizio 1. Problema dello "zaino" frazionario:
Dati n tipi di merce $M_1,\ldots, M_n$ in quantità rispettive $q_1,\ldots, q_n$ e con costi unitari $c_1,\ldots, c_n$ si vuole riempire uno zaino di capacità Q in modo che il contenuto abbia costo massimo.
Mostrare che il seguente algoritmo risolve il problema:

```
RiempiZaino(q, c, n, Q)        ► c1 ≥ c2 ≥ ... ≥ cn
  Spazio ← Q , i ← 1
  while i ≤ n do
    if Spazio ≥ qi then zi ← qi, Spazio ← Spazio - qi
    else zi ← Spazio, Spazio ← 0
    i ← i + 1
  return z
```

48

Esercizio 2. Problema dello "zaino" 0-1:
Sono dati n tipi di oggetti $O_1,\ldots, O_n$ in numero illimitato. Un oggetto di tipo O_i occupa un volume v_i e costa c_i. Si vuole riempire uno zaino di capacità Q in modo che il contenuto abbia costo massimo. Mostrare che il seguente algoritmo **non** risolve il problema

```
RiempiZaino(v, c, n, Q)      ► c1/v1 ≥ c2/v2 ≥ ... ≥ cn/vn
  Spazio ← Q , i ← 1
  while i ≤ n do
     zi ← ⌊ Spazio / vi ⌋, Spazio ← Spazio – zi vi,
     i ← i + 1
  return z
```

49

Esercizio 3.
Siano $a_1,\ldots,a_n$ attività didattiche aventi tempi di inizio $s_1,\ldots,s_n$ e tempi di fine $f_1,\ldots, f_n$ e supponiamo di avere un insieme sufficiente di aule in cui svolgerle.
Trovare un algoritmo per programmare tutte le attività nel minimo numero possibile di aule.

50

Esercizio 4.
Siano $a_1,\ldots,a_n$ attività didattiche aventi tempi di inizio $s_1,\ldots,s_n$ e tempi di fine $f_1,\ldots, f_n$ e supponiamo di avere a disposizione m aule $A_1,\ldots,A_m$.
Trovare un algoritmo goloso per programmare il massimo numero di attività nelle m aule.

51