Un problema di programmazione dei lavori

Un lavoro unitario è un lavoro che richiede esattamente una unità di tempo per essere eseguito.

Dato un insieme S di lavori unitari, una programmazione per S è semplicemente un ordine in cui eseguirli.

L'esecuzione del primo lavoro inizia all'istante 0 e termina all'istante 1, quella del secondo inizia all'istante 1 e termina all'istante 2, ecc.

1

Nel problema della programmazione di lavori unitari con scadenza e penalità sono dati:

- n lavori unitari $a_1, a_2, \dots, a_n$;
- Le scadenze $d_1, d_2, \dots, d_n$ entro cui deve essere terminato ciascuno di tali lavori ($1 \leq d_i \leq n$)
- Le penalità $w_1, w_2, \dots, w_n$ da pagare per i lavori che non sono terminati entro la scadenza.

Si chiede una programmazione che minimizzi la somma delle penalità da pagare.

2

Data una programmazione diciamo che i lavori che terminano entro la scadenza sono in orario mentre gli altri sono in ritardo.

a) Data una programmazione qualsiasi ne esiste una equivalente in cui i lavori in orario vengono eseguiti prima di quelli in ritardo.

Se x in ritardo precede y in orario, scambiandoli di posto x rimane in ritardo ed y rimane in orario.

3

b) Data una programmazione qualsiasi ne esiste una equivalente in cui i lavori in orario sono ordinati per scadenza.

Se x in orario precede y in orario e la scadenza di x è successiva alla scadenza di y , scambiandoli di posto essi rimangono entrambi in orario.

Non è quindi limitativo considerare soltanto programmazioni in forma canonica in cui i lavori in orario sono ordinati per scadenza e vengono tutti prima dei lavori in ritardo.

4

c) In una programmazione ottima l'ordine dei lavori in ritardo è irrilevante.

Due lavori in ritardo scambiati restano in ritardo altrimenti la programmazione non sarebbe ottima. Considerando indistinguibili programmazioni ottime che differiscano solo per l'ordine dei lavori in ritardo (quindi equivalenti) ...

d) Una programmazione canonica ottima è determinata dall'insieme A dei lavori in orario.

Basta elencare i lavori in A in ordine di scadenza e aggiungere gli altri lavori in un ordine qualsiasi.

5

Un insieme di lavori A si dice indipendente se esiste almeno una programmazione in cui tutti i lavori in A sono in orario.

Questo permette di riformulare il problema della programmazione dei lavori ...

6

Programmazione dei lavori

Dato un insieme di lavori unitari con scadenze e penalità determinarne un sottoinsieme A tale che

- A indipendente

- A ha costo $w(A)$ massimo

$$w(A) = \sum_{x \in A} w(x)$$

Come si verifica l'indipendenza di un insieme A ?

7

Lemma: Per $t = 0, 1, 2, \dots, n$ sia $N_t(A)$ il numero di lavori in A con scadenza minore o uguale a t .

Le seguenti asserzioni sono equivalenti:

- A. A è indipendente;
- B. $N_t(A) \leq t$ per ogni $t = 0, 1, 2, \dots, n$;
- C. Se i lavori in A sono eseguiti in ordine di scadenza nessuno di essi risulta in ritardo.

8

(A $\Rightarrow$ B) Se $N_t(A) > t$ per qualche t allora non vi è modo di ordinare tali $N_t(A)$ lavori in modo che nessuno sia in ritardo.

(B $\Rightarrow$ C) Supponiamo che valga B e ordiniamo i lavori in A per ordine di scadenza.

Sia $t = d_i$ la scadenza dell' i -esimo lavoro.

Siccome $N_t(A) \leq t$ allora $i \leq t$ e il lavoro i -esimo è in orario.

(C $\Rightarrow$ A) Se vale C, in una programmazione che consiste dei lavori in A ordinati per scadenza nessuno è in ritardo. Quindi A è indipendente.

9

.START

10

Riassunto: Programmazione di lavori unitari con scadenza e penalità

- n lavori unitari $a_1, a_2, \dots, a_n$;
- Le scadenze $d_1, d_2, \dots, d_n$ entro cui deve essere terminato ciascuno di tali lavori ($1 \leq d_i \leq n$)
- Le penalità $w_1, w_2, \dots, w_n$ da pagare per i lavori che non sono terminati entro la scadenza.

Trovare una programmazione che minimizzi la somma delle penalità da pagare.

11

Proprietà

- Per ogni programmazione ne esiste una equivalente (stesso costo) in forma canonica: i lavori in orario sono ordinati per scadenza e vengono tutti prima dei lavori in ritardo.
- Se la programmazione è ottima l'ordine dei lavori in ritardo è irrilevante

12

Proprietà (cont.)

Un insieme di lavori A si dice *indipendente* se esiste almeno una programmazione in cui tutti i lavori in A sono in orario.

Una programmazione ottima è determinata dal sottoinsieme A di lavori in orario.

13

Programmazione dei lavori

Dato un insieme di lavori unitari con scadenze e penalità determinarne un sottoinsieme A tale che

- A indipendente

- A ha costo $w(A)$ massimo

$$w(A) = \sum_{x \in A} w(x)$$

Come si verifica l'indipendenza di un insieme A ?

14

Lemma: Per $t = 0, 1, 2, \dots, n$ sia $N_t(A)$ il numero di lavori in A con scadenza minore o uguale a t .

Le seguenti asserzioni sono equivalenti:

- A. A è indipendente;
- B. $N_t(A) \leq t$ per ogni $t = 0, 1, 2, \dots, n$;
- C. Se i lavori in A sono eseguiti in ordine di scadenza nessuno di essi risulta in ritardo.

15

Test di indipendenza:

```
IndipTest( $I, k, d$ ) ▷ PRE:  $I[1..k]$  sono gli indici di  $k$  lavori  
▷ scelti tra  $a_1, a_2, \dots, a_n$   
for  $t \leftarrow 0$  to  $n$  do  $N_t \leftarrow 0$   
for  $j \leftarrow 1$  to  $k$  do  
   $d \leftarrow d_{I[j]}$   
   $N_d \leftarrow N_d + 1$   
▷  $N_t =$  numero di lavori in  $I[1..k]$  con scadenza =  $t$   
for  $t \leftarrow 1$  to  $n$  do  
   $N_t \leftarrow N_t + N_{t-1}$   
  ▷  $N_t =$  numero di lavori in  $I[1..k]$  con scadenza  $\leq t$   
  if  $N_t > t$  then return false  
return true
```

16

Test di indipendenza:

```
IndipTest( $I, k, d$ ) ▷ PRE:  $I[1..k]$  sono gli indici di  $k$  lavori  
▷ scelti tra  $a_1, a_2, \dots, a_n$   
for  $t \leftarrow 0$  to  $k$  do  $N_t \leftarrow 0$   
for  $j \leftarrow 1$  to  $k$  do  
   $d \leftarrow d_{I[j]}$   
  if  $d \leq k$  then  $N_d \leftarrow N_d + 1$   
▷  $N_t =$  numero di lavori in  $I[1..k]$  con scadenza =  $t$   
for  $t \leftarrow 1$  to  $k$  do  
   $N_t \leftarrow N_t + N_{t-1}$   
  ▷  $N_t =$  numero di lavori in  $I[1..k]$  con scadenza  $\leq t$   
  if  $N_t > t$  then return false  
return true
```

17

Alcune proprietà degli insiemi indipendenti:

1. L'insieme vuoto ϕ è indipendente.
2. Se $A \subseteq B$ e B è indipendente allora anche A è indipendente.
3. Se A e B sono indipendenti e A contiene meno lavori di B allora esiste un lavoro $x \in B \setminus A$ tale che $A' = A \cup x$ è indipendente.

18

3. Se A e B sono indipendenti e A contiene meno lavori di B allora esiste un lavoro $x \in B \setminus A$ tale che $A' = A \cup x$ è indipendente.

Sia k il massimo t tale che $N_t(B) \leq N_t(A)$.

Siccome $N_0(B) = N_0(A) = 0$ e $N_n(B) = |B|$ è maggiore di $N_n(A) = |A|$ tale k esiste e $k < n$.

Siccome $N_k(B) \leq N_k(A)$ ed $N_{k+1}(B) > N_{k+1}(A)$ allora B ha più lavori con scadenza $k+1$ di A . Sia x un lavoro in $B \setminus A$ con scadenza $k+1$.

19

Mostriamo che $A' = A \cup x$ è indipendente.

Per $0 \leq t \leq k$ abbiamo

$$N_t(A') = N_t(A) \leq t.$$

Per $k+1 \leq t \leq n$ abbiamo

$$N_t(A') = N_t(A) + 1 \leq N_t(B) \leq t.$$

20

Le proprietà suggeriscono l'algoritmo:

1. L'insieme vuoto ϕ è indipendente.
2. Se $A \subseteq B$ e B è indipendente allora anche A è indipendente.
3. Se A e B sono indipendenti e A contiene meno lavori di B allora esiste un lavoro $x \in B \setminus A$ tale che $A' = A \cup x$ è indipendente.

21

Algoritmo goloso:

```

SelezLavori(d,w,n)  ▷ PRE:  $w_1 \geq w_2 \geq \dots \geq w_n$ 
   $k \leftarrow 0, h \leftarrow n+1$ 
  ▷  $I[1..k]$  indipendenti e  $I[h..n]$  incompatibili con i lavori in  $I[1..k]$ 
  for  $i \leftarrow 1$  to  $n$  do
     $I[k+1] \leftarrow i$           ▷ provo ad aggiungere  $i$  a  $I[1..k]$ 
    if  $IndipTest(I, k+1, d)$  then ▷ se  $I[1..k+1]$  è indipendente OK
       $k \leftarrow k+1$ 
    else                       ▷ altrimenti metto il lavoro  $i$  in coda
       $I[h-1] \leftarrow i, h \leftarrow h-1$ 
  Sort( $I, k, d$ )             ▷ ordina  $I[1..k]$  in ordine di scadenza
  return  $I$                  ▷ POST:  $I[1..n]$  è una programmazione ottima.

```

22

Nella parte iniziale $I[1..k]$ dell'array vengono messi gli indici dei lavori selezionati mentre nella parte finale $I[h..n]$ vengono messi gli indici dei lavori scartati.

Indichiamo con A l'insieme dei lavori selezionati e messi in $I[1..k]$ e con D quelli scartati e messi in $I[h..n]$.

23

Dimostriamo la proprietà della scelta golosa, ossia che le scelte locali effettuate non pregiudicano la possibilità di arrivare ad una soluzione ottima.

Invariante del ciclo: Esiste sempre una programmazione ottima con insieme indipendente B che estende A ovvero tale che

- i lavori dell'insieme A sono in orario ($A \subseteq B$) e
- i lavori in D sono in ritardo ($D \cap B = \phi$)

24

La dimostrazione è per induzione su i e si basa unicamente sulle seguenti tre proprietà degli insiemi indipendenti:

1. L'insieme vuoto ϕ è indipendente.
2. Se $A \subseteq B$ e B è indipendente allora anche A è indipendente.
3. Se A e B sono indipendenti e $|A| < |B|$ allora esiste $x \in B \setminus A$ tale che $A' = A \cup x$ è indipendente.

25

($i = 0$) Banalmente vera, dato che all'inizio $A = \phi$ e $D = \phi$.

($i > 0$) Per ipotesi induttiva, prima di eseguire il passo i -esimo, esiste una programmazione ottima con insieme indipendente B , tale che

- i) $A \subseteq B$
- ii) $D \cap B = \phi$

Mostriamo che questo rimane vero anche dopo l'esecuzione del passo i -esimo.

26

Sia x il lavoro i -esimo.

Caso 1: l'insieme $A \cup x$ non è indipendente.

Se l'insieme $A \cup x$ non è indipendente allora per la Proprietà 2 **non** vale

$$A \cup x \subseteq B$$

Siccome $A \subseteq B$, deve essere

$$x \notin B$$

ovvero il lavoro x sarà in ritardo anche in B .

Quindi aggiungendo x a D l'invariante rimane vero.

27

Caso 2: l'insieme $A \cup x$ è indipendente.

Se $A \cup x$ è indipendente x viene aggiunto ad A .

Mostriamo che esiste una programmazione ottima B' dove i lavori in $A \cup x$ sono in orario e i lavori in D sono in ritardo.

Se x è in orario in B ($x \in B$) siamo a posto.

Se x è in ritardo in B ($x \notin B$), allora $|A \cup x| \leq |B|$.

Altrimenti siccome $A \subseteq B$ si avrebbe $A = B$, quindi $B \cup x = A \cup x$ indipendente e B non sarebbe ottima.

28

Per la Proprietà 3 possiamo aggiungere elementi di B ad $A \cup x$ fino ad ottenere un insieme indipendente $A' \cup x$ tale che $|A' \cup x| = |B|$.

Dunque $A' \cup x = B \setminus y \cup x$ per $y \in B \setminus A'$.

Ma $w(x) \geq w(y)$ per come è stato scelto x e dato che $y \notin A \cup D$. Quindi

$$w(A' \cup x) = w(B) - w(y) + w(x) \geq w(B).$$

Ma B è una programmazione ottima e quindi anche l'insieme $A' \cup x$ rappresenta una programmazione ottima: $w(A' \cup x) = w(B)$.

29

Dunque $B' = A' \cup x$ è ottima e soddisfa:

i) Tutti i lavori in $A \cup x$ sono in orario in B' .
Per costruzione $A \cup x \subseteq A' \cup x = B'$.

ii) Tutti i lavori in D sono in ritardo in B'
Se $z \in D$ allora $A \cup z$ non è indipendente. Quindi non può essere sottoinsieme di B' (Proprietà 2).

Siccome $A \subseteq B'$ allora $z \notin B'$ (ovvero il lavoro z è in ritardo in B').

Dunque tutti i lavori in D sono in ritardo anche in B' e quindi l'invariante resta vero anche nel Caso 2.

30

Quando termina il ciclo, vale l'invariante, ovvero esiste una programmazione ottima B tale che

- i) $A \subseteq B$
- ii) $D \cap B = \emptyset$

Ovvero l'insieme B

- i) contiene l'insieme A dei lavori selezionati e messi in $I[1,k]$
- ii) non contiene nessuno dei lavori dell'insieme D messi in $I[k+1,n]$

Se ne conclude che $A=B$. Pertanto A è ottima.

31

Un metodo generale

La tecnica usata per risolvere il problema della programmazione di un insieme S di lavori unitari con scadenza e penalità si può generalizzare ad una vasta gamma di problemi.

Per risolvere il problema abbiamo costruito un sottoinsieme indipendente A di lavori tale che la somma delle penalità dei lavori in A sia massima (ovvero somma delle penalità dei lavori in $S \setminus A$ sia minima).

32

Abbiamo costruito tale insieme A scegliendo di volta in volta, con la tecnica golosa, il lavoro con penalità massima tra tutti quelli che si potevano aggiungere ad A mantenendo l'indipendenza. Lo schema dell'algoritmo è:

Selezione(a,w,n)

▷ **PRE:** $w_1 \geq w_2 \geq \dots \geq w_n$

$A \leftarrow \emptyset$

for $i \leftarrow 1$ to n do

if " $A \cup a_i$ è indipendente" then

$A \leftarrow A \cup a_i$

▷ **POST:** A è un insieme indipendente di costo massimo.

33

Per dimostrare che in questo modo si ottiene proprio un insieme indipendente di costo massimo ci siamo basati sulle seguenti proprietà dei sottoinsiemi indipendenti:

1. L'insieme vuoto $\emptyset$ è indipendente.
2. Se $A \subseteq B$ e B è indipendente allora anche A è indipendente.
3. Se A e B sono indipendenti e $|A| < |B|$ allora esiste $x \in B \setminus A$ tale che $A' = A \cup x$ è indipendente.

34

Quindi !!!!!!!!!!!!!!!!!!!!!!!:

Dato un qualsiasi insieme S di n oggetti con pesi $w_1, w_2, \dots, w_n$ ed una qualsiasi famiglia $\mathcal{M}$ di sottoinsiemi di S che soddisfino tali tre proprietà (e che chiameremo insiemi *indipendenti*), l'algoritmo generale trova un sottoinsieme $A \in \mathcal{M}$ di peso massimo.

Una famiglia $\mathcal{M}$ di sottoinsiemi che soddisfa le proprietà (1)-(3) è nota con il nome di *matroide*.

35

La correttezza dell'algoritmo si prova con:

- **Scelta golosa**

Sia x un elemento di peso massimo in S tale che $\{x\}$ è indipendente (se esiste). Allora esiste una soluzione ottima A tale che $x \in A$.

- **Sottostruttura ottima**

Sia x come sopra e

$$S' = \{ x \in S : \{ x, y \} \text{ indep.} \}$$

$$\mathcal{M}' = \{ B \subseteq S \setminus x : B \cup x \text{ indep.} \}$$

Sia A una soluzione ottima per $\mathcal{M}'$. Allora $A \cup x$ è ottima per $\mathcal{M}$.

36

Talvolta i matroidi sono definiti con le proprietà:

- a. Esiste almeno un insieme indipendente;
- b. Se B è indipendente ed $A \subseteq B$ allora anche A è indipendente;
- c. Per ogni sottoinsieme R di S , se A e B sono due sottoinsiemi di R indipendenti e massimali in R allora $|A| = |B|$.

invece che con le 1, 2 e 3.

Esercizio 7: dimostrare che le due definizioni di matroide sono equivalenti.

37

Esercizio 8.

Sia S l'insieme delle righe di una matrice.

Dimostrare che la famiglia $\mathcal{M}$ degli insiemi di righe che sono linearmente indipendenti è un matroide.

Assumendo che a ciascuna riga della matrice sia associato un peso, scrivere un algoritmo che calcola un insieme di righe linearmente indipendenti di peso massimo.

38

Esercizio 9.

Una ditta, al fine di ridurre le spese telefoniche, intende collegare con linea privata i centri calcolo delle sue molte sedi. A tal fine si è procurata i preventivi per un insieme S di connessioni dirette tra due sedi. Sia w_{ij} il risparmio previsto con la connessione diretta tra le sedi x_i, y_j . Una connessione diretta tra sedi già collegate da una catena di connessioni dirette è chiaramente inutile.

Dimostrare che la famiglia $\mathcal{M}$ dei sottoinsiemi aciclici di S è un matroide.

Sviluppare un algoritmo che calcola un sottoinsieme aciclico di connessioni che massimizza il risparmio.

39

Esercizio 10.

Un cassiere vuole dare un resto di n centesimi di euro usando il minimo numero di monete.

- a) Descrivere un algoritmo goloso per fare ciò con tagli da 1¢, 2¢, 5¢, 10¢, 20¢, 50¢, 1€ e 2€.
- b) Dimostrare che l'algoritmo goloso funziona anche con monete di tagli 1, c , c^2 , ..., c^k dove $c > 1$ e $k \geq 0$.
- c) Trovare un insieme di tagli di monete per i quali l'algoritmo goloso non funziona.

40