
Tecniche Algoritmiche: tecnica greedy (o golosa)

Algoritmi Greedy

Idea: per trovare una soluzione **globalmente ottima**, scegli ripetutamente soluzioni **ottime localmente**

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Un esempio

Input: lista di interi che rappresentano il valore di monete disponibili e un intero che rappresenta un resto.

Output: una collezione di **cardinalità minima** di monete la cui somma sia uguale al resto.

Input: monete disponibili: 25, 10, 5 e 1;
resto da formare: 87

Output: 25+25+25+10+1+1

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Osservazione:

gli algoritmi greedy non sempre funzionano !

Input: monete disponibili: 1,4 e 6
resto da formare: 9

Greedy output: 6 + 1 + 1 + 1

Output ottimale: 4 + 4 + 1

Le scelte ottime a breve non garantiscono di raggiungere una soluzione ottima finale.

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Algoritmi Greedy - Schema generale 1

Le appetibilità degli oggetti sono note fin dall'inizio e non vengono modificate

Greedy1 ($\{a_1, a_2, \dots, a_n\}$)

$S \leftarrow$ insieme vuoto

ordina gli a_i in ordine **non crescente di appetibilità**

for ogni a_i nell'ordine **do**

if a_i può essere aggiunto a S

then $S \leftarrow S \cup \{a_i\}$

return S

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Una applicazione classica per lo schema 1

Il problema della selezione di attività

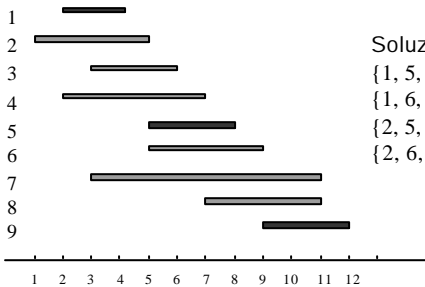
Input: $S = \{1, 2, \dots, n\}$ insieme di attività che devono usare una risorsa in modo esclusivo. Ogni attività i è caratterizzata da un tempo di inizio e da un tempo di fine: $[s_i, f_i)$ ($s_i \leq f_i$).

Output: insieme che contiene il massimo numero di attività mutuamente compatibili.

Oggetti: le attività

Appetibilità: tempo di fine

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)



A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Greedy-Activity-Selector (n, s, f)

$A \leftarrow$ insieme vuoto

“ordina le attività in ordine non decrescente rispetto ai tempi di fine”

$A \leftarrow \{1\}$

$j \leftarrow 1$

for $i \leftarrow 2$ **to** n **do**

if $s_i \geq f_j$

then $A \leftarrow A \dot{\cup} \{i\}$

$j \leftarrow i$

return A

Complessità: $O(n \lg n)$

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Ricordiamo che: gli algoritmi greedy non sempre funzionano !

Domanda. L'algoritmo *Greedy-Activity-Selector* seleziona sempre un numero massimo di attività?

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Greedy-Activity-Selector (n, s, f)

```
A ← insieme vuoto
“ordina le attività in ordine non decrescente
rispetto ai tempi di fine”
A ← {1}
j ← 1
{A e' una soluzione ottima rispetto alle attività esaminate}
for i ← 2 to n do
  if  $s_i \geq f_j$ 
    then A ← A ∪ {i}
    j ← i
return A
{A e' una soluzione ottima}
```

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Invariante dettagliato durante il ciclo:
esiste un E, insieme massimo di attività compatibili in $\{1, \dots, n\}$, tale che:

- A è un sottoinsieme di E
- l'appetibilità degli elementi di E-A non è superiore a quella degli elementi di A
- j è l'ultima attività aggiunta ad A

Dopo il ciclo:
A è un insieme massimo di attività compatibili

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

L'invariante viene reso vero dall'inizializzazione:

- Siano $\langle i_1, i_2, \dots, i_k \rangle$ le attività (in ordine crescente di fine attività) di un insieme massimo di attività compatibili E.
- Poiché $f_1 \leq f_{i_1} \leq s_{i_2}$ anche $\langle 1, i_2, \dots, i_k \rangle$ è un insieme massimo di attività compatibili.
- L'appetibilità degli elementi di E-{1} non è superiore a quella degli elementi di {1}, e $j = 1$ è l'ultima attività scelta.

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Supponiamo che

- $A = \langle 1, h_2, \dots, h_r \rangle$ sia un sottoinsieme di $E = \langle 1, h_2, \dots, h_r, i_{r+1}, \dots, i_k \rangle$ (in ordine crescente di fine attività),
- sia $j = h_r$ l'ultima attività scelta.

Allora

- $f_{i_{r+1}} \leq s_{i_{r+2}}$
- se i è l'attività selezionata nel corpo del while, $f_i \leq f_{i_{r+1}}$

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Perciò

- $\langle 1, h_2, \dots, h_r, i \rangle$ è un sottoinsieme di un insieme massimo di attività compatibili: $\langle 1, h_2, \dots, h_r, i, i_{r+2}, \dots, i_k \rangle$
- l'appetibilità degli elementi di $E - \{1, h_2, \dots, h_r, i\}$ è non superiore a quella degli elementi di $\{1, h_2, \dots, h_r, i\}$

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Algoritmi Greedy - Schema generale 2

Se le appetibilità degli oggetti possono essere modificate dalle scelte già fatte.

Greedy2 ($\{a_1, a_2, \dots, a_n\}$)

$S \leftarrow$ insieme vuoto

valuta le appetibilità degli a_i

while ci sono elementi da scegliere **do**

 scegli l' a_i più appetibile

if a_i può essere aggiunto a S

then $S \leftarrow S \cup \{a_i\}$

 aggiorna le appetibilità degli a_i

return S

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Una applicazione classica per lo schema 2

Codici di Huffman (tecnica efficiente per la compressione dei dati)

Esempio:

- Si consideri un alfabeto di 6 caratteri.
- In un codice a lunghezza fissa servono 3 bit per la loro rappresentazione.
- Quindi un file di dati di 100.000 caratteri richiede 300.000 bit.

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Una applicazione classica per lo schema 2

Un codice a lunghezza variabile

- codifica i simboli che occorrono più di frequente con meno bit
- può permettere un risparmio anche del 25%.

Per esempio:

- caratteri: a,b,c,d,e,f
- frequenza: 0.45,0.13,0.12,0.16,0.09,0.05
- codice l. fissa: 000,001,010,011,100,101
- codice l. variabile: 0,101,100,111,1101,1100

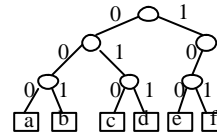
A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Codici prefissi:

nessuna parola del codice è prefisso di un'altra.

Con codici prefissi è facile la decodifica in quanto non ci sono ambiguità.

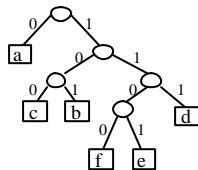
Un codice binario può essere rappresentato in modo efficiente con un albero binario in cui le foglie rappresentano i caratteri.



A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Un codice "ottimo":

- permette la codifica più possibile compressa
- può essere rappresentato con un albero binario in cui
 - ogni nodo interno ha sempre due figli e
 - i caratteri più frequenti compaiono ai livelli alti dell'albero



A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

• Se C è l'alfabeto, l'albero di un codice prefisso ottimo per C ha $|C| - 1$ nodi interni.

• A partire dall'albero è facile calcolare il numero di bit necessari a codificare un file.

• Indichiamo

• con $f(c)$ la frequenza del carattere "c" nel file

• con $d_T(c)$ il livello della foglia etichettata "c" nell'albero T .

• Il numero di bit è allora:

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

L'algoritmo di Huffman

Input: un insieme C di caratteri e una funzione f che indica la frequenza di ogni carattere in un testo.

Output: un codice binario ottimo per la compressione del testo.

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

L'algoritmo di Huffman

- considerare i caratteri in ordine non decrescente di frequenza
- per ogni carattere creare un albero formato solo da una foglia
- a partire dai due alberi che hanno frequenza minore costruire un nuovo albero che ha come frequenza la somma delle frequenze degli alberi fusi
- ripetere la fusione finché si ottiene un unico albero

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Si può individuare in tale descrizione una strategia greedy:

Oggetti: gli alberi
Appetibilità: frequenza

Poiché gli oggetti vengono modificati e con essi le loro appetibilità, applichiamo lo schema greedy, con qualche modifica per l'eventuale aggiornamento delle appetibilità.

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Huffman ($\{c_1, c_2, \dots, c_n\}$)
valuta le appetibilità dei c_i

while l'albero non è unico do

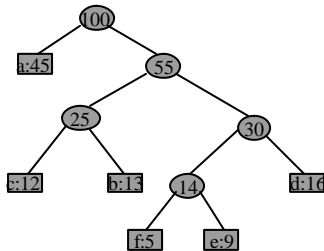
 scegli i due alberi più **appetibili**
 crea l'albero fusione dei due
 aggiorna le appetibilità degli alberi

return l'albero ottenuto

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Esempio:

- caratteri: a,b,c,d,e,f
- frequenza: 0.45,0.13,0.12,0.16,0.09,0.05



A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Implementazione

Consideriamo i seguenti oggetti e metodi dati:

- C: un insieme di $|C|$ caratteri
 - ogni carattere c e' inizialmente rappresentato come una coppia (carattere, frequenza)
 - $f[c]$ denota la frequenza di c

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Implementazione

- **Allocate-Node()**: un funzione che restituisce un nuovo nodo di una struttura ad albero
 - le cui foglie conterranno coppie (carattere, frequenza) e
 - i cui nodi interni conterranno la somma delle frequenze contenute nelle foglie del sottoalbero corrispondente
 - $f[z]$ denota la frequenza contenuta nel nodo (foglia o nodo interno) z ,
 - $left[z]$ e $right[z]$ denotano il figlio sinistro e destro di z rispettivamente

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Implementazione

- Q è una coda con priorità che
 - memorizza (radici di) alberi la cui priorità è data dal valore di f
 - sarà utilizzata per identificare i due alberi con la frequenza più piccola, da fondere assieme

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Implementazione

Huffman (C, f)

$n \leftarrow |C|$

$Q \leftarrow C$

{per ottenere un unico albero
sono necessarie n-1 fusioni}

for $i \leftarrow 1$ **to** $n - 1$ **do**

$z \leftarrow \text{Allocate-Node} ()$

{due alberi più

$x \leftarrow \text{left}[z] \leftarrow \text{Extract-min}(Q)$ appetibili e crea

$y \leftarrow \text{right}[z] \leftarrow \text{Extract-min}(Q)$ l'albero fusione}

$f[z] \leftarrow f[x] + f[y]$ {aggiorna le appetibilità
degli alberi}

Insert (Q, z)

return $\text{Extract-min}(Q)$

{restituisce l'albero ottenuto}

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Implementazione

Complessita' in tempo:

$O(n \lg n)$ se la coda di priorità è implementata con un heap binario.

$O(n^2)$ se la coda fosse realizzata con una struttura con inserzione e/o estrazione del minimo lineari

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)

Conclusione

- **Tecnica greedy**: varie applicazioni:
 - problema del sequenziamento
 - calcolo dei codici di Huffman
 - calcolo del minimo albero ricoprente (algoritmi di Prim, Kruskal, Boruvka)
 - cammini minimi a sorgente singola (algoritmo di Dijkstra)
- Abbiamo visto due schemi.
- Importante dimostrare che funzionano.

A. Horvath Alg&Lab 0809 (da F. Damiani - Alg. & Sper. 06/07 e M. Zacchi - Alg. & Lab. 03/04)