Tecniche avanzate di programmazione

Algoritmi greedy

Prof. Luca Campion

I.S. "Antonio Scarpa" - Motta di livenza (TV)

A.S. 2023/2024

Indice

Problemi di ottimizzazione

Problemi di ottimizzazione

Gli algoritmi greedy sono procedure utilizzate per risolvere problemi di ottimizzazione.

Definizione

Un *problema di ottimizzazione* ha la forma:

$$\begin{aligned} \max / \min \quad & f(x) \\ & x \in K \end{aligned}$$

Questi problemi hanno spesso diverse soluzioni *ammissibili* (punti di K), ma poche di queste sono *ottime* (punti di K che massimizzano o minimizzano f).

Problemi di ottimizzazione

Gli algoritmi greedy sono procedure utilizzate per risolvere problemi di ottimizzazione.

Definizione

Un *problema di ottimizzazione* ha la forma:

$$\begin{aligned} \max / \min \quad & f(x) \\ & x \in K \end{aligned}$$

Questi problemi hanno spesso diverse soluzioni *ammissibili* (punti di K), ma poche di queste sono *ottime* (punti di K che massimizzano o minimizzano f).

Problemi di ottimizzazione

Gli algoritmi greedy sono procedure utilizzate per risolvere problemi di ottimizzazione.

Definizione

Un *problema di ottimizzazione* ha la forma:

$$\begin{aligned} \max / \min \quad & f(x) \\ & x \in K \end{aligned}$$

Questi problemi hanno spesso diverse soluzioni *ammissibili* (punti di K), ma poche di queste sono *ottime* (punti di K che massimizzano o minimizzano f).

Problemi di ottimizzazione

L'idea che sta alla base degli algoritmi greedy è quella di *prendere la decisione localmente migliore*.

Consideriamo il seguente problema:

Problema 1 (Vetta massima)

Un alpinista si trova in una valle e vuole raggiungere la vetta più alta. A causa della nebbia riesce a vedere solo le vette adiacenti.

Questo è un problema di ottimizzazione: i punti ammissibili dell'insieme K sono le vette e le valli del profilo montuoso, mentre la funzione obiettivo f è quella che assenga alla vetta o valle l'altezza.

Problemi di ottimizzazione

L'idea che sta alla base degli algoritmi greedy è quella di *prendere la decisione localmente migliore*.
Consideriamo il seguente problema:

Problema 1 (Vetta massima)

Un alpinista si trova in una valle e vuole raggiungere la vetta più alta. A causa della nebbia riesce a vedere solo le vette adiacenti.

Questo è un problema di ottimizzazione: i punti ammissibili dell'insieme K sono le vette e le valli del profilo montuoso, mentre la funzione obiettivo f è quella che assenga alla vetta o valle l'altezza.

Problemi di ottimizzazione

L'idea che sta alla base degli algoritmi greedy è quella di *prendere la decisione localmente migliore*.
Consideriamo il seguente problema:

Problema 1 (Vetta massima)

Un alpinista si trova in una valle e vuole raggiungere la vetta più alta. A causa della nebbia riesce a vedere solo le vette adiacenti.

Questo è un problema di ottimizzazione: i punti ammissibili dell'insieme K sono le vette e le valli del profilo montuoso, mentre la funzione obiettivo f è quella che assenga alla vetta o valle l'altezza.

Problemi di ottimizzazione

L'idea che sta alla base degli algoritmi greedy è quella di *prendere la decisione localmente migliore*.
Consideriamo il seguente problema:

Problema 1 (Vetta massima)

Un alpinista si trova in una valle e vuole raggiungere la vetta più alta. A causa della nebbia riesce a vedere solo le vette adiacenti.

Questo è un problema di ottimizzazione: i punti ammissibili dell'insieme K sono le vette e le valli del profilo montuoso, mentre la funzione obiettivo f è quella che assenga alla vetta o valle l'altezza.

Problemi di ottimizzazione

Un algoritmo greedy per risolvere questo problema potrebbe essere:

Algoritmo 1: Algoritmo greedy per il problema della vetta di altezza massima

while *Ci sono vette vicine più alte del punto corrente* **do**
 sali sulla vetta più alta che riesci a vedere

Problemi di ottimizzazione

Un algoritmo greedy per risolvere questo problema potrebbe essere:

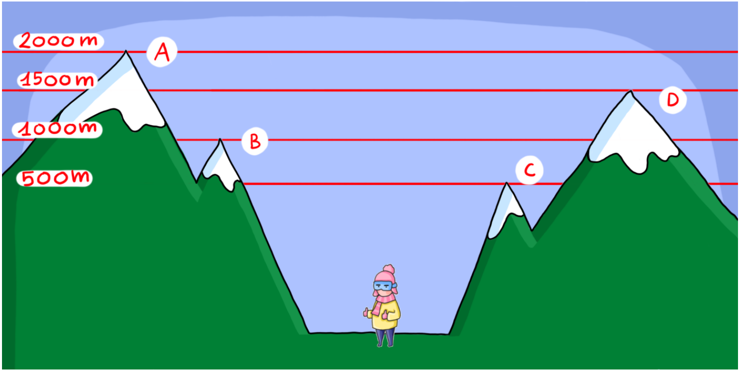
Algoritmo 1: Algoritmo greedy per il problema della vetta di altezza massima

while *Ci sono vette vicine più alte del punto corrente* **do**
  sali sulla vetta più alta che riesci a vedere

Problemi di ottimizzazione

Per alcune istanze del problema questo algoritmo porta alla soluzione ottima. Ad esempio, nella figura

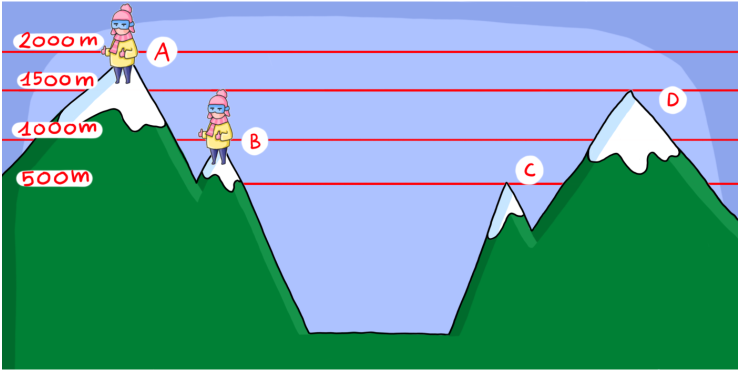
Figura: Istanza del problema della vetta massima.



Problemi di ottimizzazione

l'algoritmo 1 produce la soluzione ottima del problema, infatti l'alpinista passa per i punti B e A.

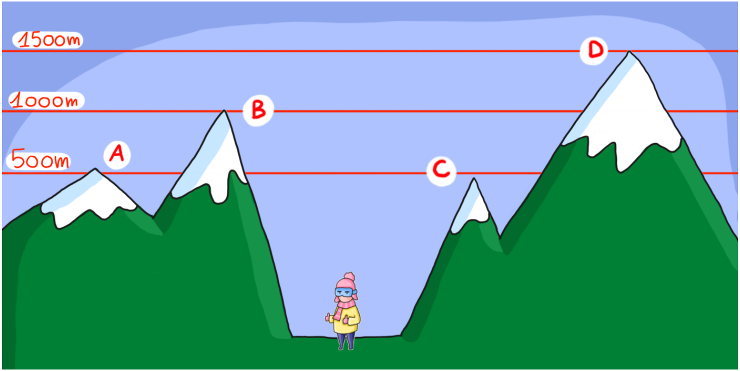
Figura: Istanza del problema della vetta massima.



Problemi di ottimizzazione

Se però consideriamo l'istanza in figura

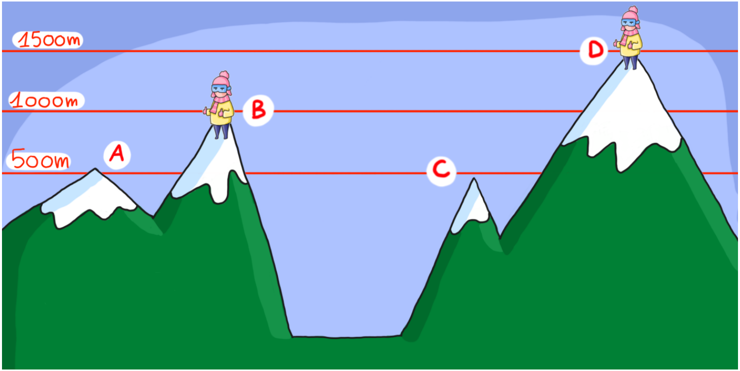
Figura: Istanza del problema della vetta massima.



Problemi di ottimizzazione

l'algoritmo 1 produce una soluzione ammissibile, ma non ottima, infatti l'alpinista passa al B e si ferma più in basso del punto D.

Figura: Istanza del problema della vetta massima.



Problemi di ottimizzazione

Da questo esempio notiamo che:

Osservazione

Gli algoritmi greedy non sempre producono una soluzione ottima, ma producono sempre una soluzione ammissibile, in generale buona, in un tempo molto veloce.

Osservazione

Se gli algoritmi greedy raggiungono una soluzione ottima, la raggiungono molto velocemente

Problemi di ottimizzazione

Da questo esempio notiamo che:

Osservazione

Gli algoritmi greedy non sempre producono una soluzione ottima, ma producono sempre una soluzione ammissibile, in generale buona, in un tempo molto veloce.

Osservazione

Se gli algoritmi greedy raggiungono una soluzione ottima, la raggiungono molto velocemente

Problemi di ottimizzazione

Il problema 1 ammette un algoritmo che produce una soluzione ottima, ma in un tempo maggiore.

Algoritmo 2: Algoritmo ingenuo per il problema della vetta di altezza massima

foreach *vette nel profilo montuoso* **do**
    passo alla vetta successiva ricordandomi la più alta su cui sono passato
ritorno indietro alla più alta visitata.

Problemi di ottimizzazione

Il problema 1 ammette un algoritmo che produce una soluzione ottima, ma in un tempo maggiore.

Algoritmo 2: Algoritmo ingenuo per il problema della vetta di altezza massima

foreach *vette nel profilo montuoso* **do**
    passo alla vetta successiva ricordandomi la più alta su cui sono passato

ritorno indietro alla più alta visitata.

Problemi di ottimizzazione

Ipotizzando di partire dal centro di un profilo montuoso di n vette, con l'algoritmo ingenuo servono, nel caso pessimo in cui la vetta massima sia all'estrema destra, $\frac{n}{2}$ passi per arrivare all'estremo destro, n passi per attraversare il profilo montuoso arrivando all'estremo sinistro, e altri n passi per ritornare all'estremo destro. Il numero totale di passi effettuati dall'alpinista è

$$\frac{n}{2} + n + n = \frac{5}{2}n$$

Con l'algoritmo 1, dato che non si retrocede, si arriva ad una soluzione ammissibile non ulteriormente migliorabile in $\frac{n}{2}$ passi. L'algoritmo greedy è quindi 5 volte più veloce dell'algoritmo ingenuo.

Problemi di ottimizzazione

Ipotizzando di partire dal centro di un profilo montuoso di n vette, con l'algoritmo ingenuo servono, nel caso pessimo in cui la vetta massima sia all'estrema destra, $\frac{n}{2}$ passi per arrivare all'estremo destro, n passi per attraversare il profilo montuoso arrivando all'estremo sinistro, e altri n passi per ritornare all'estremo destro. Il numero totale di passi effettuati dall'alpinista è

$$\frac{n}{2} + n + n = \frac{5}{2}n$$

Con l'algoritmo 1, dato che non si retrocede, si arriva ad una soluzione ammissibile non ulteriormente migliorabile in $\frac{n}{2}$ passi. L'algoritmo greedy è quindi 5 volte più veloce dell'algoritmo ingenuo.

Problemi di ottimizzazione

Ipotizzando di partire dal centro di un profilo montuoso di n vette, con l'algoritmo ingenuo servono, nel caso pessimo in cui la vetta massima sia all'estrema destra, $\frac{n}{2}$ passi per arrivare all'estremo destro, n passi per attraversare il profilo montuoso arrivando all'estremo sinistro, e altri n passi per ritornare all'estremo destro. Il numero totale di passi effettuati dall'alpinista è

$$\frac{n}{2} + n + n = \frac{5}{2}n$$

Con l'algoritmo 1, dato che non si retrocede, si arriva ad una soluzione ammmissibile non ulteriormente migliorabile in $\frac{n}{2}$ passi. L'algoritmo greedy è quindi 5 volte più veloce dell'algoritmo ingenuo.

Problemi di ottimizzazione

Ipotizzando di partire dal centro di un profilo montuoso di n vette, con l'algoritmo ingenuo servono, nel caso pessimo in cui la vetta massima sia all'estrema destra, $\frac{n}{2}$ passi per arrivare all'estremo destro, n passi per attraversare il profilo montuoso arrivando all'estremo sinistro, e altri n passi per ritornare all'estremo destro. Il numero totale di passi effettuati dall'alpinista è

$$\frac{n}{2} + n + n = \frac{5}{2}n$$

Con l'algoritmo 1, dato che non si retrocede, si arriva ad una soluzione ammmissibile non ulteriormente migliorabile in $\frac{n}{2}$ passi. L'algoritmo greedy è quindi 5 volte più veloce dell'algoritmo ingenuo.

Problemi di ottimizzazione

Ipotizzando di partire dal centro di un profilo montuoso di n vette, con l'algoritmo ingenuo servono, nel caso pessimo in cui la vetta massima sia all'estrema destra, $\frac{n}{2}$ passi per arrivare all'estremo destro, n passi per attraversare il profilo montuoso arrivando all'estremo sinistro, e altri n passi per ritornare all'estremo destro. Il numero totale di passi effettuati dall'alpinista è

$$\frac{n}{2} + n + n = \frac{5}{2}n$$

Con l'algoritmo 1, dato che non si retrocede, si arriva ad una soluzione ammmissibile non ulteriormente migliorabile in $\frac{n}{2}$ passi. L'algoritmo greedy è quindi 5 volte più veloce dell'algoritmo ingenuo.

Algoritmo del resto

Algoritmo del resto

Uno dei problemi più semplici che si risolve con un metodo greedy è il *problema del resto*.

Problema 2 (Resto)

Dato un prezzo e una quantità pagata, determinare il resto utilizzando il numero minimo di monete o banconote.

Algoritmo del resto

Uno dei problemi più semplici che si risolve con un metodo greedy è il *problema del resto*.

Problema 2 (Resto)

Dato un prezzo e una quantità pagata, determinare il resto utilizzando il numero minimo di monete o banconote.

Algoritmo del resto

L'insieme dell soluzioni ammissibili K è l'insieme di tutte le combinazioni di tagli il cui valore è uguale al resto, mentre la funzione obiettivo f è quella che associa ad ogni insieme di quantità di tagli la sua cardinalità. Un approccio greedy per risolvere questo problema è quello di partire dal taglio più grande e calcolare quante banconote o monete ci stanno nella soluzione ottima. Con il resto rimanente si procede in modo analogo.

Algoritmo del resto

L'insieme dell soluzioni ammissibili K è l'insieme di tutte le combinazioni di tagli il cui valore è uguale al resto, mentre la funzione obiettivo f è quella che associa ad ogni insieme di quantità di tagli la sua cardinalità. Un approccio greedy per risolvere questo problema è quello di partire dal taglio più grande e calcolare quante banconote o monete ci stanno nella soluzione ottima. Con il resto rimanente si procede in modo analogo.

Algoritmo del resto

Algoritmo 3: Algoritmo greedy per il problema del resto

foreach *taglio di monete/banconote ordinate in ordine decrescente* **do**
    calcolo quante monete/banconote ci stanno nel resto

Il problema 2 ammette un algoritmo greedy che produce una soluzione ottima.

Algoritmo del resto

Algoritmo 3: Algoritmo greedy per il problema del resto

foreach *taglio di monete/banconote ordinate in ordine decrescente* **do**
    calcolo quante monete/banconote ci stanno nel resto

Il problema 2 ammette un algoritmo greedy che produce una soluzione ottima.

Algoritmo del Knapsack

Algoritmo del Knapsack

Problema 3 (Knapsack)

Dato uno zaino di peso massimo $P > 0$ e un insieme $T = \{1, \ldots, n\}$ di *tipi* di oggetti di cui si conoscono i pesi (positivi) $p_1, \ldots, p_n$ e i valori (positivi) $v_1, \ldots, v_n$, vogliamo riempire lo zaino non superando il peso massimo e massimizzando il valore degli elementi inseriti. Gli oggetti sono presenti con disponibilità infinita.

Se gli oggetti sono di molti tipi e la capacità dello zaino è molto grande, le possibilità diventano troppe.

Algoritmo del Knapsack

Problema 3 (Knapsack)

Dato uno zaino di peso massimo $P > 0$ e un insieme $T = \{1, \ldots, n\}$ di *tipi* di oggetti di cui si conoscono i pesi (positivi) $p_1, \ldots, p_n$ e i valori (positivi) $v_1, \ldots, v_n$, vogliamo riempire lo zaino non superando il peso massimo e massimizzando il valore degli elementi inseriti. Gli oggetti sono presenti con disponibilità infinita.

Se gli oggetti sono di molti tipi e la capacità dello zaino è molto grande, le possibilità diventano troppe.

Algoritmo del Knapsack

Problema 3 (Knapsack)

Dato uno zaino di peso massimo $P > 0$ e un insieme $T = \{1, \dots, n\}$ di *tipi* di oggetti di cui si conoscono i pesi (positivi) $p_1, \dots, p_n$ e i valori (positivi) $v_1, \dots, v_n$, vogliamo riempire lo zaino non superando il peso massimo e massimizzando il valore degli elementi inseriti. Gli oggetti sono presenti con disponibilità infinita.

Se gli oggetti sono di molti tipi e la capacità dello zaino è molto grande, le possibilità diventano troppe.

Algoritmo del Knapsack

Si può procedere in diversi modi per risolvere questo problema:

1. Inserisco nello zaino gli elementi in ordine decrescente di valore (punto sulla qualità);
2. Inserisco nello zaino gli elementi in ordine crescente di peso (punto sulla quantità);
3. Inserisco nello zaino gli elementi in ordine decrescente di rapporto valore-peso (cerco un compromesso);

Algoritmo del Knapsack

Si può procedere in diversi modi per risolvere questo problema:

1. Inserisco nello zaino gli elementi in ordine decrescente di valore (punto sulla qualità);
2. Inserisco nello zaino gli elementi in ordine crescente di peso (punto sulla quantità);
3. Inserisco nello zaino gli elementi in ordine decrescente di rapporto valore-peso (cerco un compromesso);

Algoritmo del Knapsack

Si può procedere in diversi modi per risolvere questo problema:

1. Inserisco nello zaino gli elementi in ordine decrescente di valore (punto sulla qualità);
2. Inserisco nello zaino gli elementi in ordine crescente di peso (punto sulla quantità);
3. Inserisco nello zaino gli elementi in ordine decrescente di rapporto valore-peso (cerco un compromesso);

Algoritmo del Knapsack

Il primo metodo trova facilmente un controesempio. Ipotiziamo di avere a disposizione uno zaino di capacità 10 e due tipologie di oggetti i cui pesi e valori sono rispettivamente:

$$p = (8, 3) \quad v = (4, 2) \tag{1}$$

La soluzione ottenuta dal primo algoritmo è quella che inserisce nello zaino solo il primo elemento di peso 8 e valore 4, anche se la soluzione ottima è quella che inserisce nello zaino tre copie dell'elemento 2, raggiungendo un valore totale di 6.

Algoritmo del Knapsack

Il primo metodo trova facilmente un controesempio. Ipotiziamo di avere a disposizione uno zaino di capacità 10 e due tipologie di oggetti i cui pesi e valori sono rispettivamente:

$$p = (8, 3) \quad v = (4, 2) \tag{1}$$

La soluzione ottenuta dal primo algoritmo è quella che inserisce nello zaino solo il primo elemento di peso 8 e valore 4, anche se la soluzione ottima è quella che inserisce nello zaino tre copie dell'elemento 2, raggiungendo un valore totale di 6.

Algoritmo del Knapsack

Il primo metodo trova facilmente un controesempio. Ipotiziamo di avere a disposizione uno zaino di capacità 10 e due tipologie di oggetti i cui pesi e valori sono rispettivamente:

$$p = (8, 3) \quad v = (4, 2) \tag{1}$$

La soluzione ottenuta dal primo algoritmo è quella che inserisce nello zaino solo il primo elemento di peso 8 e valore 4, anche se la soluzione ottima è quella che inserisce nello zaino tre copie dell'elemento 2, raggiungendo un valore totale di 6.

Algoritmo del Knapsack

Il secondo metodo risolve correttamente l'istanza precedente, ma fallisce se analiziamo gli oggetti di peso e valore:

$$p = (8, 3) \quad v = (16, 3) \tag{2}$$

La soluzione ottenuta dal secondo algoritmo è quella che inserisce nello zaino tre copie dell'oggetto due, totalizzando un valore di 9, mentre la soluzione ottima è quella che inserisce nello zaino il primo oggetto, raggiungendo un valore di 16.

Algoritmo del Knapsack

Il secondo metodo risolve correttamente l'istanza precedente, ma fallisce se analiziamo gli oggetti di peso e valore:

$$p = (8, 3) \quad v = (16, 3) \tag{2}$$

La soluzione ottenuta dal secondo algoritmo è quella che inserisce nello zaino tre copie dell'oggetto due, totalizzando un valore di 9, mentre la soluzione ottima è quella che inserisce nello zaino il primo oggetto, raggiungendo un valore di 16.

Algoritmo del Knapsack

Il terzo metodo notiamo che funziona in tutti e due i casi.
Nell'esempio 1 i rapporti valore-peso sono:

$$\frac{v}{p} = \left(\frac{4}{8}, \frac{2}{3}\right)$$

e dato che il maggiore è $0.\overline{6}$, vengono inserite nello zaino tre copie dell'elemento 2. Nell'esempio 2 i rapporti valore-peso sono:

$$\frac{v}{p} = \left(\frac{16}{8}, \frac{3}{3}\right)$$

e dato che il maggiore è 2, vengono inserite nello zaino tre copie dell'elemento 2.

Algoritmo del Knapsack

Il terzo metodo notiamo che funziona in tutti e due i casi. Nell'esempio 1 i rapporti valore-peso sono:

$$\frac{v}{p} = \left(\frac{4}{8}, \frac{2}{3}\right)$$

e dato che il maggiore è $0.\overline{6}$, vengono inserite nello zaino tre copie dell'elemento 2. Nell'esempio 2 i rapporti valore-peso sono:

$$\frac{v}{p} = \left(\frac{16}{8}, \frac{3}{3}\right)$$

e dato che il maggiore è 2, vengono inserite nello zaino tre copie dell'elemento 2.

Algoritmo del Knapsack

Il terzo metodo notiamo che funziona in tutti e due i casi. Nell'esempio 1 i rapporti valore-peso sono:

$$\frac{v}{p} = \left(\frac{4}{8}, \frac{2}{3}\right)$$

e dato che il maggiore è $0.\overline{6}$, vengono inserite nello zaino tre copie dell'elemento 2. Nell'esempio 2 i rapporti valore-peso sono:

$$\frac{v}{p} = \left(\frac{16}{8}, \frac{3}{3}\right)$$

e dato che il maggiore è 2, vengono inserite nello zaino tre copie dell'elemento 2.

Algoritmo del Knapsack

È stato dimostrato nel 1990 da Martello e Toth che questo metodo arriva ad una soluzione ottima. (Silvano Martello, Paolo Toth, Knapsack Problems: Algorithms and Computer Implementations).
Questo è uno dei problemi più importanti dell'ottimizzazione matematica in quanto ammette diverse varianti con moltissime applicazioni pratiche.

Algoritmo del Knapsack

È stato dimostrato nel 1990 da Martello e Toth che questo metodo arriva ad una soluzione ottima. (Silvano Martello, Paolo Toth, Knapsack Problems: Algorithms and Computer Implementations). Questo è uno dei problemi più importanti dell'ottimizzazione matematica in quanto ammette diverse varianti con moltissime applicazioni pratiche.

Algoritmo del Knapsack

Possiamo quindi formulare un algoritmo greedy funzionante per risolvere il problema 3

Algoritmo 4: Algoritmo greedy per il problema del knapsack

foreach $i \in T$ **do**
  calcolo $A_i = \frac{v_i}{p_i}$

foreach A_i *ordinati in modo decrescente* **do**
  se l'elemento i ci sta nello zaino lo inserisco.

Schedulazione delle attività

Schedulazione delle attività

Un altro problema che si può risolvere con un approccio greedy è quello della schedulazione delle attività.

Problema 4 (Schedulazione delle attività)

Dato un insieme di attività $T = \{1, \ldots, n\}$ con dei relativi istanti di inizio $S = \{s_1, \ldots, s_n\}$ e di fine $F = \{f_1, \ldots, f_n\}$ con $s_i < f_i$, determinare un insieme di cardinalità massima di attività compatibili. Due attività i e j si dicono compatibili se $f_i \leq s_j$ oppure $f_j \leq s_i$.

Schedulazione delle attività

Un altro problema che si può risolvere con un approccio greedy è quello della schedulazione delle attività.

Problema 4 (Schedulazione delle attività)

Dato un insieme di attività $T = \{1, \ldots, n\}$ con dei relativi istanti di inizio $S = \{s_1, \ldots, s_n\}$ e di fine $F = \{f_1, \ldots, f_n\}$ con $s_i < f_i$, determinare un insieme di cardinalità massima di attività compatibili. Due attività i e j si dicono compatibili se $f_i \leq s_j$ oppure $f_j \leq s_i$.

Schedulazione delle attività

Un altro problema che si può risolvere con un approccio greedy è quello della schedulazione delle attività.

Problema 4 (Schedulazione delle attività)

Dato un insieme di attività $T = \{1, \ldots, n\}$ con dei relativi istanti di inizio $S = \{s_1, \ldots, s_n\}$ e di fine $F = \{f_1, \ldots, f_n\}$ con $s_i < f_i$, determinare un insieme di cardinalità massima di attività compatibili. Due attività i e j si dicono compatibili se $f_i \leq s_j$ oppure $f_j \leq s_i$.

Schedulazione delle attività

I criteri greedy con cui scegliere le attività possono essere:

- *Earliest start time*: prendo le attività in ordine crescente rispetto gli istanti iniziali;
- *Shortest interval*: prendo le attività più brevi;
- *Fewest conflicts*: prendo le attività in ordine crescente del numero di conflitti che generano;
- *Earliest finish time*: prendo le attività in ordine crescente rispetto gli istanti finali;

Schedulazione delle attività

I criteri greedy con cui scegliere le attività possono essere:

- *Earliest start time*: prendo le attività in ordine crescente rispetto gli istanti iniziali;
- *Shortest interval*: prendo le attività più brevi;
- *Fewest conflicts*: prendo le attività in ordine crescente del numero di conflitti che generano;
- *Earliest finish time*: prendo le attività in ordine crescente rispetto gli istanti finali;

Schedulazione delle attività

I criteri greedy con cui scegliere le attività possono essere:

- *Earliest start time*: prendo le attività in ordine crescente rispetto gli istanti iniziali;
- *Shortest interval*: prendo le attività più brevi;
- *Fewest conflicts*: prendo le attività in ordine crescente del numero di conflitti che generano;
- *Earliest finish time*: prendo le attività in ordine crescente rispetto gli istanti finali;

Schedulazione delle attività

I criteri greedy con cui scegliere le attività possono essere:

- *Earliest start time*: prendo le attività in ordine crescente rispetto gli istanti iniziali;
- *Shortest interval*: prendo le attività più brevi;
- *Fewest conflicts*: prendo le attività in ordine crescente del numero di conflitti che generano;
- *Earliest finish time*: prendo le attività in ordine crescente rispetto gli istanti finali;

Schedulazione delle attività

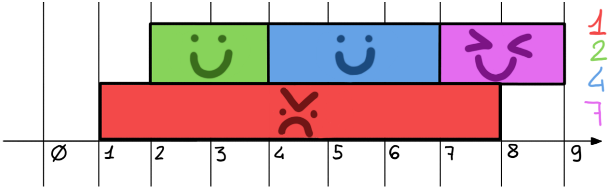
I criteri greedy con cui scegliere le attività possono essere:

- *Earliest start time*: prendo le attività in ordine crescente rispetto gli istanti iniziali;
- *Shortest interval*: prendo le attività più brevi;
- *Fewest conflicts*: prendo le attività in ordine crescente del numero di conflitti che generano;
- *Earliest finish time*: prendo le attività in ordine crescente rispetto gli istanti finali;

Schedulazione delle attività

Con il metodo *Earliest start time* può capitare che un'attività che comincia per prima finisca dopo molte altre, come in figura

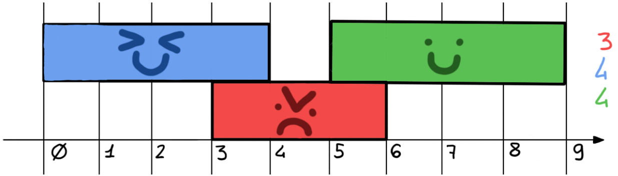
Figura: Earliest start time.



Schedulazione delle attività

Con il metodo *Shortest interval* può capitare che un'attività che un'attività che dura meno di altre due le renda non compatibili, come in figura

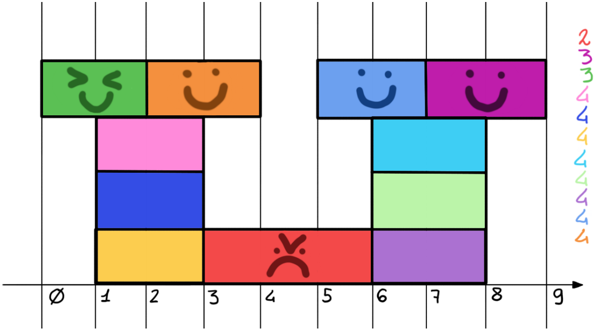
Figura: Shortest interval.



Schedulazione delle attività

Con il metodo *Fewest conflicts* può capitare che un'attività che un'attività che genera pochi conflitti renda inutilizzabili molte attività che ne generano di più.

Figura: Fewest conflicts.



Schedulazione delle attività

La soluzione migliore è adottare una politica *earliest finish time*.
Possiamo quindi definire un algoritmo greedy per risolvere li problema 4.

Algoritmo 5: Algoritmo greedy per il problema della schedulazione delle attività

Ordino le attività per ordine crescente di istante finale
foreach $i \in T$ **do**
  se l'attività i è compatibile con quelle nella soluzione la aggiungo

Schedulazione delle attività

La soluzione migliore è adottare una politica *earliest finish time*. Possiamo quindi definire un algoritmo greedy per risolvere li problema 4.

Algoritmo 5: Algoritmo greedy per il problema della schedulazione delle attività

Ordino le attività per ordine crescente di istante finale
foreach $i \in T$ **do**
    se l'attività i è compatibile con quelle nella soluzione la aggiungo

Schedulazione delle attività

La soluzione migliore è adottare una politica *earliest finish time*. Possiamo quindi definire un algoritmo greedy per risolvere li problema 4.

Algoritmo 5: Algoritmo greedy per il problema della schedulazione delle attività

Ordino le attività per ordine crescente di istante finale
foreach $i \in T$ **do**
  se l'attività i è compatibile con quelle nella soluzione la aggiungo