Gli algoritmi

Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

Algoritmo

Sequenza finita di istruzioni che:

- risolvono un problema
- soddisfano i seguenti criteri:
  - ricevono valori in ingresso
  - producono valori in uscita
  - sono chiare, non ambigue ed eseguibili
  - terminano dopo un numero finito di passi
- operano su strutture dati

Algoritmo: da al-Huarizmi, matematico persiano del IX secolo d.C.

Algoritmo vs. procedura

Se, per ogni possibile valore di ingresso, la terminazione è garantita in un numero finito di passi, allora

algoritmo

altrimenti

procedura

La congettura di Collatz (1937)

Data la funzione f(n) così definita:

$$f(n) = \begin{cases} n/2 \text{ per } n \text{ pari} \\ 3n + 1 \text{ per } n \text{ dispari} \end{cases}$$

dato un qualsiasi numero naturale n, la funzione f(n) convergerà ad 1 in un numero finito di passi?

Non siamo in grado di rispondere!

n = 11

11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

converge in 15 passi

n = 27

Converge in 111 passi

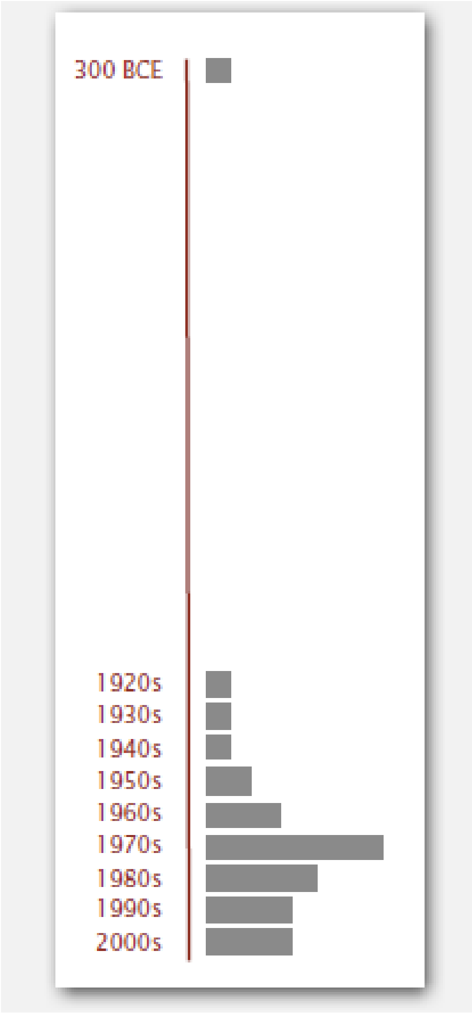
Non si può garantire che con n qualsiasi converga.

01collatz.c

```c
#include <stdio.h>
main() {
  int n;
  printf("Input natural number:  ");
  scanf("%d", &n);

  while (n > 1) {
    printf("%d   ", n);
    if ((n%2) ==1)
      n = 3*n + 1;
    else
      n = n/2;
  }
  printf("%d \n", n);
}
```

Gli algoritmi nella storia

- Euclide (IV secolo a.C.)
- formalizzazione di Church e Turing (XX secolo, anni 30)
- sviluppi recenti



Perché gli algoritmi?

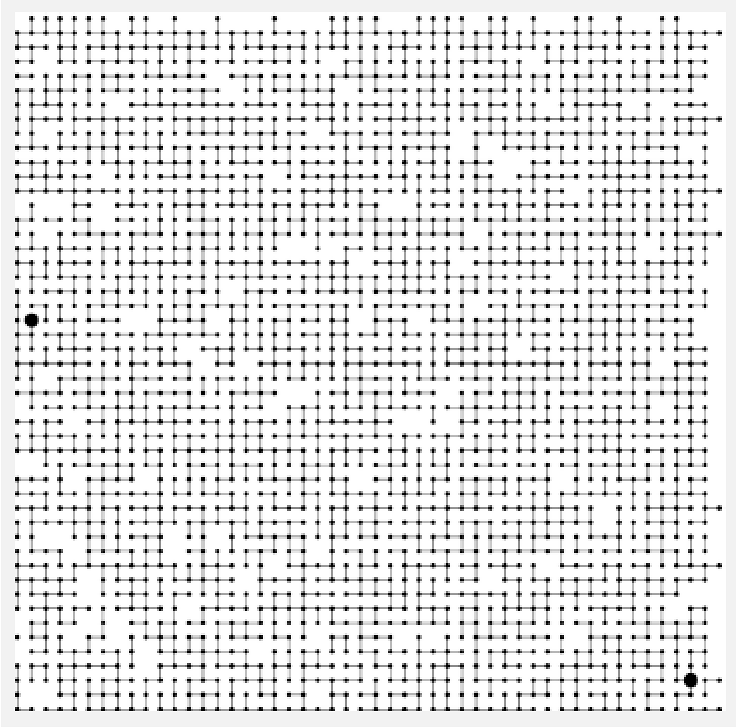
- per risolvere problemi in moltissimi campi
- per fare qualcosa di altrimenti impossibile
- per curiosità intellettuale
- per programmare bene
- per creare modelli
- per divertimento o per denaro.

Per risolvere problemi in moltissimi campi:

- Internet: Web search, packet routing, distributed file sharing
- Biologia: genoma umano
- Computer: strumenti CAD, file systems, compilatori
- Grafica: realtà virtuale, videografica
- Multimedia: MP3, JPG, DivX, HDTV
- Social Networks: recommendations, news feed, pubblicità
- Sicurezza: e-commerce, cellulari
- Fisica: simulazione di collisione di particelle …

Per fare qualcosa di altrimenti impossibile:

- in questa rete, i 2 punti evidenziati sono connessi tra di loro (network connectivity)?



Per programmare bene:

" I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. "

— Linus Torvalds (creator of Linux)

Per creare modelli:

- in molte scienze i modelli computazionali stanno sostituendo quelli matematici

$$E = mc^2$$

$$F = ma$$

$$F = \frac{G m_1 m_2}{r^2}$$

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V(r)\right]\Psi(r) = E\,\Psi(r)$$

```
for (double t = 0.0; true; t = t + dt)
   for (int i = 0; i < N; i++)
   {
      bodies[i].resetForce();
      for (int j = 0; j < N; j++)
         if (i != j)
            bodies[i].addForce(bodies[j]);
   }
```

Formule matematiche

Modello computazionale

Il modello computazionale

- Chi o cosa esegue un algoritmo?
  - uomo
  - macchina
- Ci sono limiti sulla potenza delle macchine che possiamo costruire?
- Esiste un modello universale di computazione?

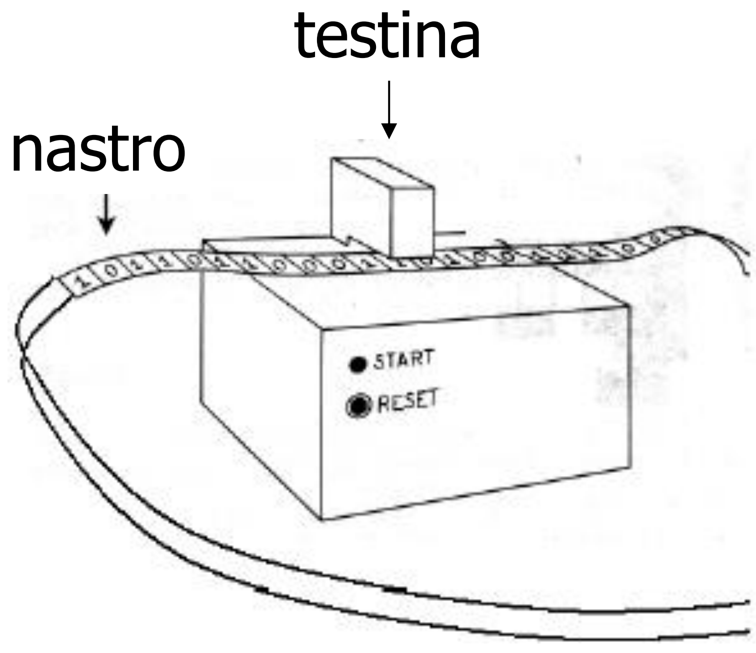
La macchina di Turing

La macchina di Turing

La Macchina di Turing è un modello che, data una funzione *f* e un *input*, al termine della computazione genera il corrispondente *output*.

E' definita da:

- un nastro
- una testina
- uno stato interno
- un programma
- uno stato iniziale

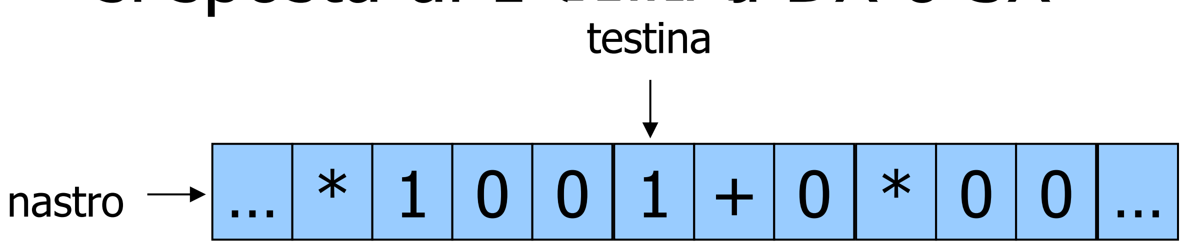


Il nastro:

- memorizza input, output e risultati intermedi
- ha lunghezza infinita ed è diviso in celle
- ogni cella contiene un simbolo $\in$ alfabeto

La testina:

- punta a una cella del nastro
- legge o scrive la cella corrente
- si sposta di 1 cella a DX o SX



Lo stato interno è la configurazione della macchina in funzione dell'input corrente e della «storia» passata.

La macchina, in funzione dello stato e dell'input correnti, scrive un valore sul nastro e sposta la testina a DX o SX (programma).

La tesi di Church-Turing (1936)

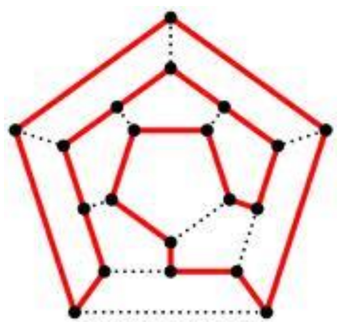
«La Macchina di Turing può calcolare qualsiasi funzione che possa essere calcolata da una macchina fisicamente realizzabile.»

Tesi e non teorema perché è un'affermazione sul mondo fisico non soggetta a prova, però in 80 anni non si sono trovati controesempi.

Tutti i modelli computazionali finora trovati sono equivalenti alla Macchina di Turing.

Tipologie di problemi

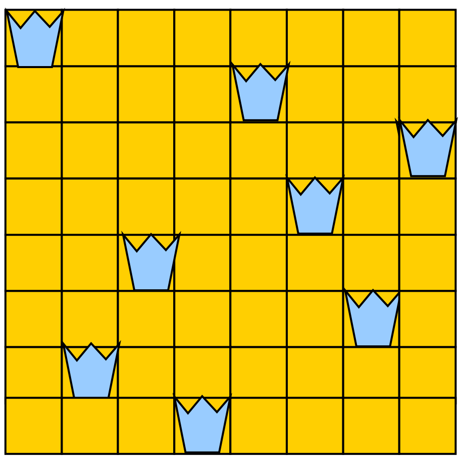
- Problemi di decisione: problemi che ammettono una risposta sì/no
  - dati 2 interi x e y, x divide esattamente y?
  - dato un intero positivo x, x è primo?
  - dato un intero positivo n, esistono 2 interi positivi e >1 p e q tali che n = pq?



- Problemi di ricerca: esiste e quale è una soluzione valida? La soluzione si trova in uno spazio delle soluzioni eventualmente infinito:
  - ciclo Hamiltoniano: dato un grafo non orientato, esiste e quale è un ciclo semplice che contiene tutti i vertici?
  - quale è il k-esimo numero primo?
  - dato un vettore di interi, riordinarlo in ordine ascendente.

- Problemi di verifica: data una soluzione (certificato), appurare che è davvero tale:

8 regine



Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

- Problemi di ottimizzazione: se esiste, quale è la soluzione migliore?
  - cammini minimi: dato un grafo orientato e pesato, quale è il cammino semplice di lunghezza minima, se esiste, tra i nodi i e j?

I problemi di ottimizzazione possono avere una versione di decisione introducendo un limite sulle soluzioni valide:

- cammini minimi: dato un grafo orientato e pesato, dati i nodi i e j e una distanza massima d, esiste un cammino semplice tra i e j di lunghezza $\leq$ d?

Ogni problema di ottimizzazione è almeno altrettanto difficile della sua versione di decisione.

I problemi di decisione

- Possono essere:
  - decidibili (esiste un algoritmo che li risolve)
    - determinare se un numero è primo

02prime.c

```
int Prime(int n) {
  int fact;
    if (n == 1) return 0;
   fact = 2;
   while (n % fact != 0)
     fact  = fact +1;
  return (fact == n);
}
```

- indecidibili (non esiste alcun algoritmo che li risolve)
  - dato un algoritmo A e un dato D, entrambi arbitrari, stabilire se la computazione A(D) termina in un numero finito di passi (Turing halting problem, 1937)
  - Congettura di Goldbach (XVII secolo): ogni numero intero pari maggiore di 2 è la somma di due numeri primi p e q

    $\forall\ n \in \aleph, (n > 2) \wedge (n \text{ pari}) \Rightarrow (\exists\ p, q \in \wp \in, n = p + q)$

```c
#define  upper 20
```

03Goldbach.c

```c
void Goldbach(void) {
  int n = 2, counterexample, p, q;
  do {
     n = n + 2;
     printf("I try for n = %d\n", n);
     counterexample = 1;
     for (p = 2; p <= n-2; p++){
       q = n - p;
       if (Prime(p) == 1 && Prime(q) == 1){
         counterexample = 0;
         printf("%d %d\n", p, q);
       }
     }
  } while (counterexample == 0 && n < upper);
    if (counterexample == 1)
      printf(“Counterexample is: %d \n", n);
    else
      printf("Until n= %d none found\n", upper);
  return;
}
```

I problemi di decisione decidibili possono essere:

- trattabili, cioè risolvibili in tempi “ragionevoli”:
  - ordinare un vettore di n interi
- intrattabili, cioè non risolvibili in tempi “ragionevoli”:
  - le Torri di Hanoi

La Classe P

Problemi di decisione decidibili e trattabili

$\Leftrightarrow$

$\exists$ un algoritmo polinomiale che li risolve
(Tesi di Edmonds-Cook-Karp, anni '70)

Polinoniale: algoritmo che, operando su n dati, data una costante c>0, termina in un numero di passi limitato superiormente da n^c

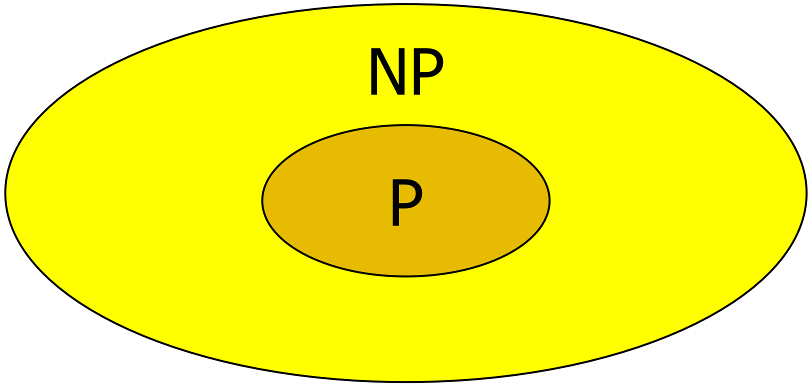
- In pratica c non deve eccedere 2.

La Classe NP

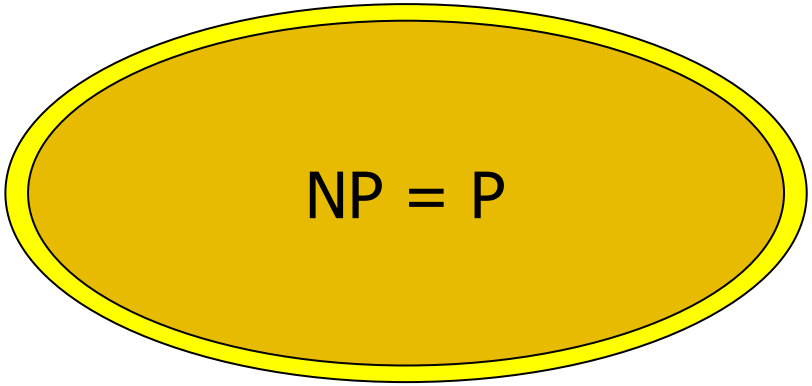
- Esistono problemi di decisione decidibili per cui conosciamo algoritmi di soluzione esponenziali, ma non conosciamo algoritmi polinomiali. Non possiamo però escludere che esistano
- Conosciamo algoritmi polinomiali per **verificare** che una soluzione (*certificato*) ad un'istanza è davvero tale
  - Sudoku, soddisfacibilità di una funzione booleana

PS: NP sta per Non-deterministico Polinomiale e fa riferimento alla Macchina di Turing non deterministica.

- $P \subseteq NP$, ma non è noto se P è un sottoinsieme proprio di NP o se al limite coincide con esso. E' probabile che sia un sottoinsieme proprio.



probabile

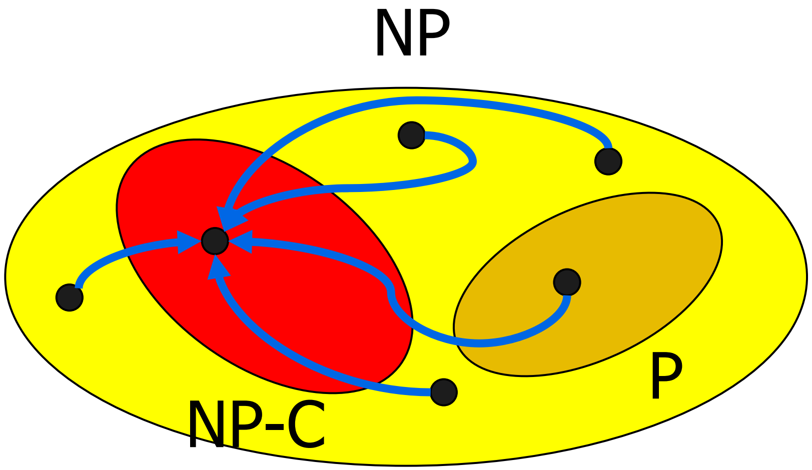


improbabile

La Classe NP-C

Un problema è NP-**completo** se:

- è NP
- ogni altro problema in NP è riducibile ad esso attraverso una trasformazione polinomiale



Se un problema NP-completo fosse risolvibile con un algoritmo polinomiale, allora con trasformazioni polinomiali si troverebbero algoritmi polinomiali per tutti i problemi NP.

MOLTO IMPROBABILE!

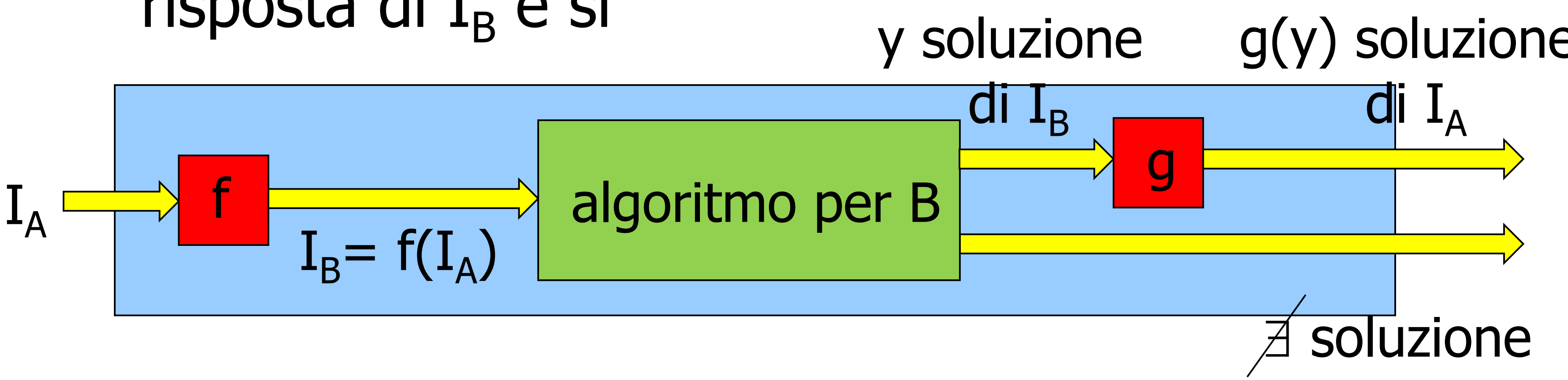
L'esistenza di NP-C rende probabile che $P \subset NP$

Esempio di problema NP-C: soddisfacibilità

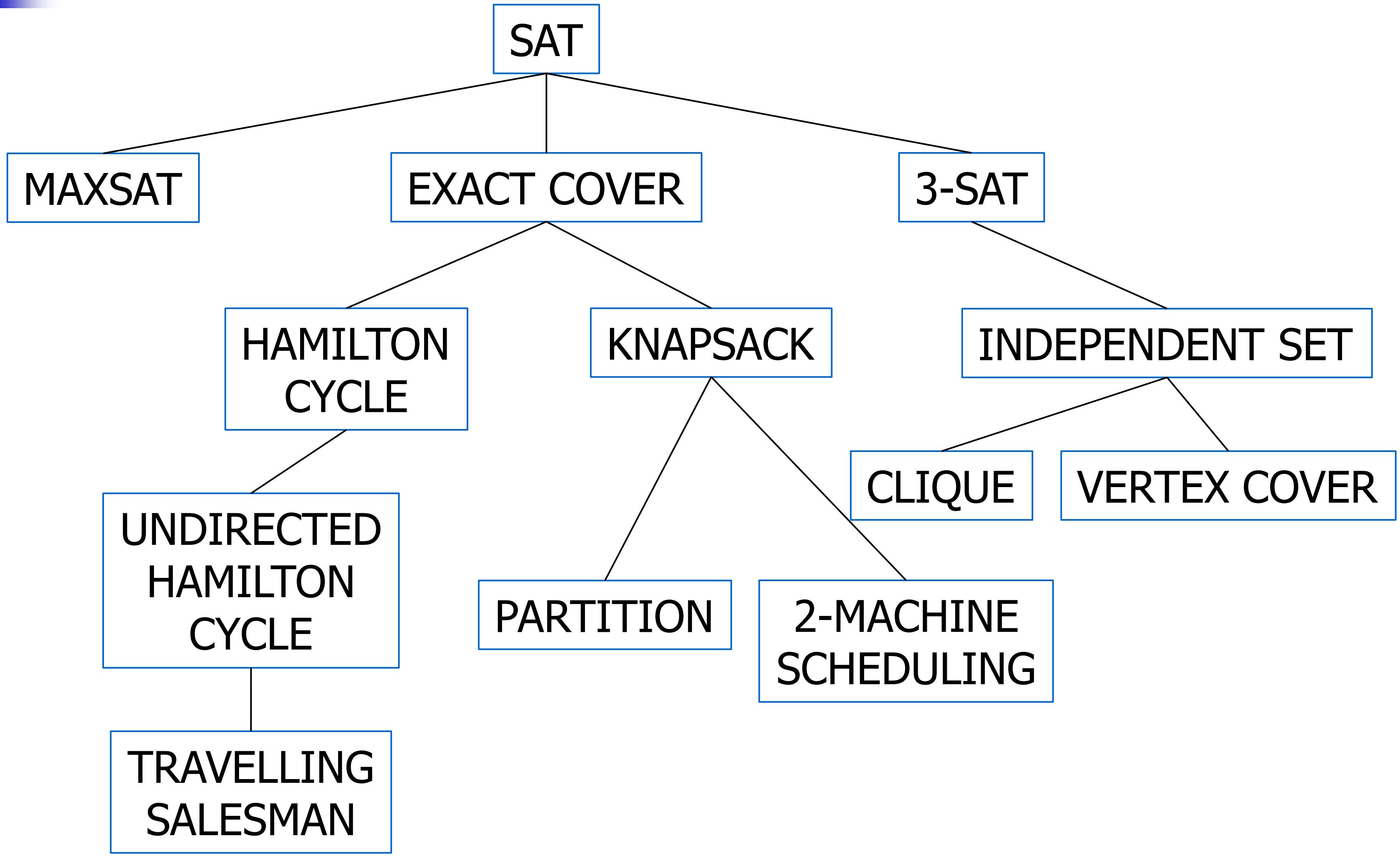
data una funzione booleana, determinare se esiste una qualche combinazione di valori delle variabili di ingresso per cui la funzione risulti vera.

Dati 2 problemi di decisione A e B, una riduzione polinomiale da A a B ($A \leq_P B$) è una procedura che trasforma ogni istanza I_A di A in una istanza I_B di B:

- con costo polinomiale
- tale che la risposta per I_A è sì se e solo se la risposta di I_B è sì



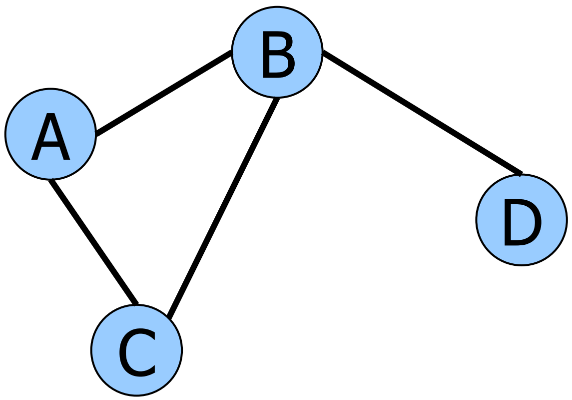
Esempi di riduzioni



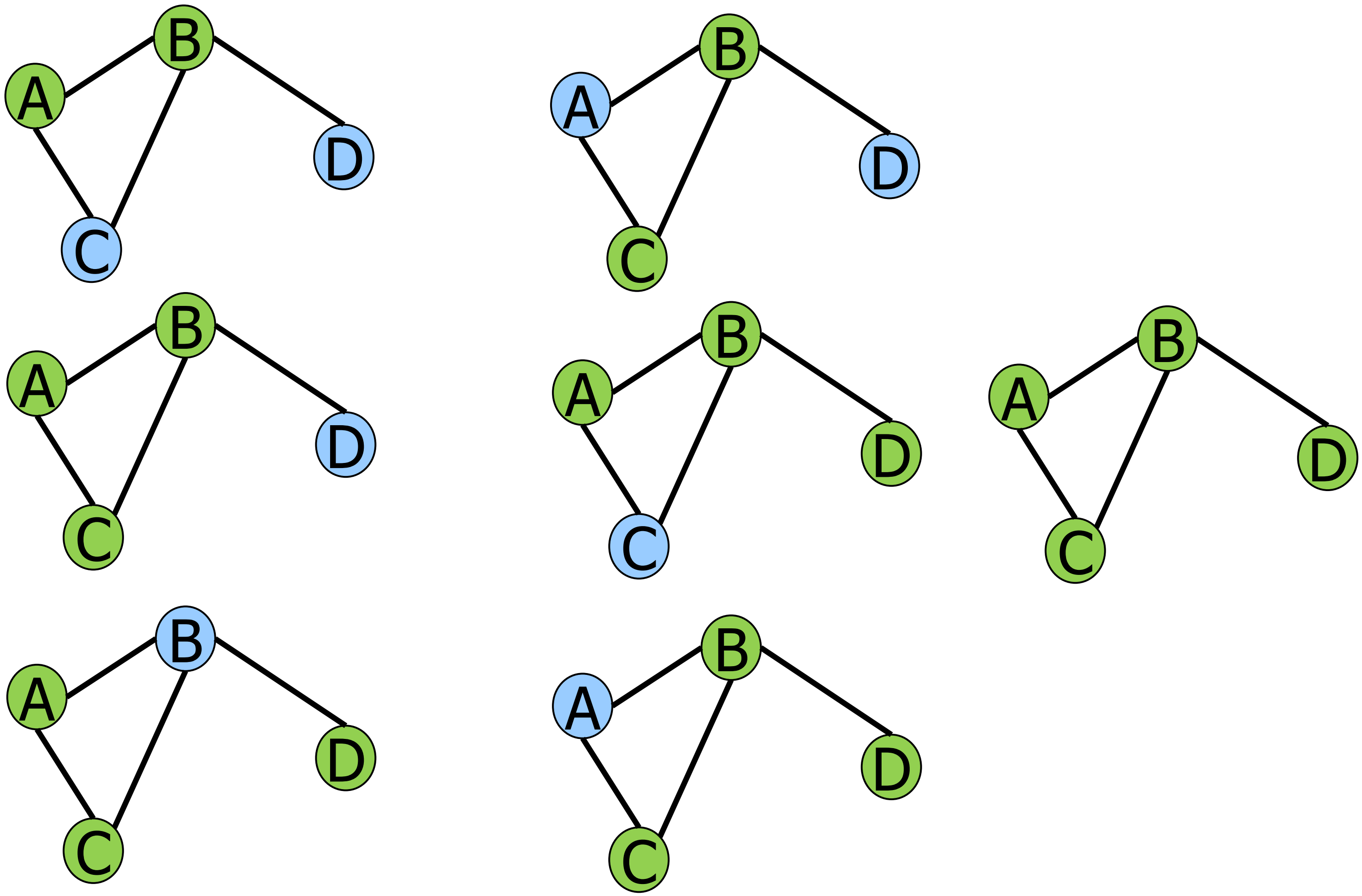
Vertex-cover

Dato un grafo non orientato, un vertex-cover è un sottoinsieme W dei vertici tali che per tutti gli archi (u,v) o $u \in W$ o $v \in W$

Esempio

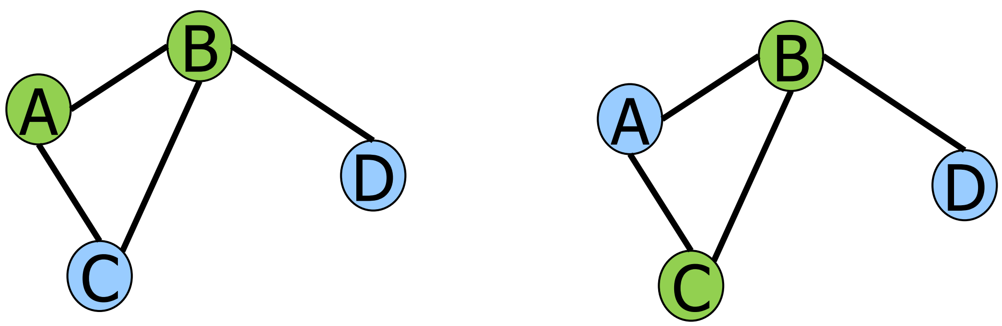


7 soluzioni



- Problema di ricerca: trovare un vertex-cover
- Problema di ottimizzazione: trovare un vertex-cover di cardinalità minima
- Problema di decisione: esiste un vertex-cover di cardinalità $\leq$ k?

Problema di decisione con k = 2
2 soluzioni

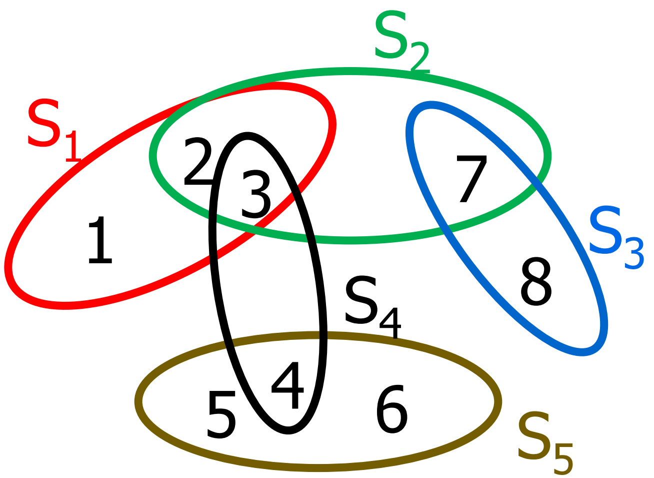


Set-cover

Problema di decisione:

dato un insieme U di elementi, una collezione $S_1, S_2, \ldots, S_n$ di suoi sottoinsiemi e un intero k, esiste una collezione di al massimo k sottoinsiemi la cui unione sia U?

Esempio



$U = \{1,2,3,4,5,6,7,8\}$

$S_1 = \{1,2,3\}$

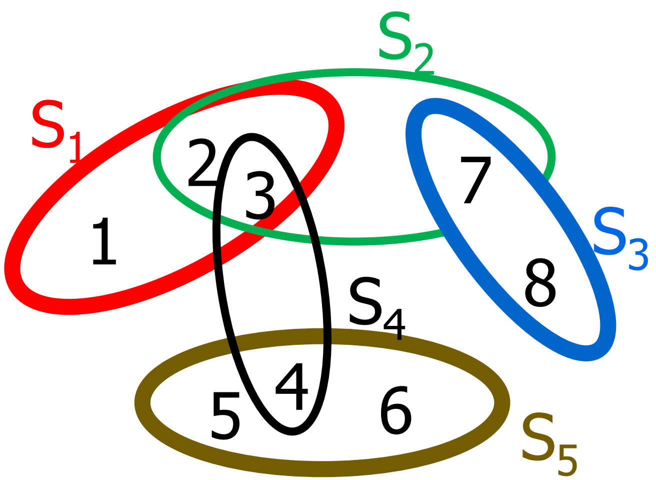
$S_2 = \{2,3,7\}$

$S_3 = \{7,8\}$

$S_4 = \{3,4\}$

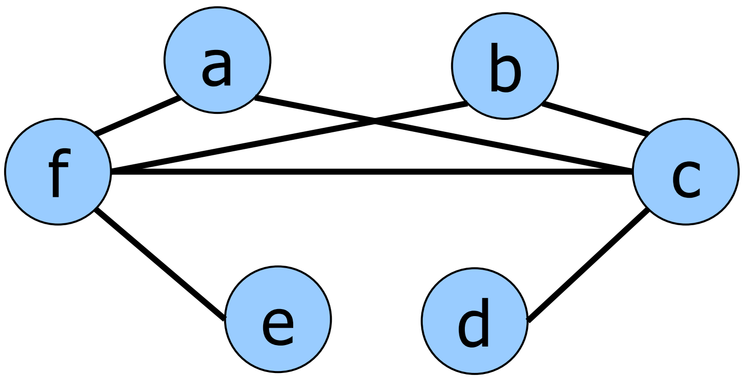
$S_5 = \{4,5,6\}$

Soluzione per k = 3



Esempio

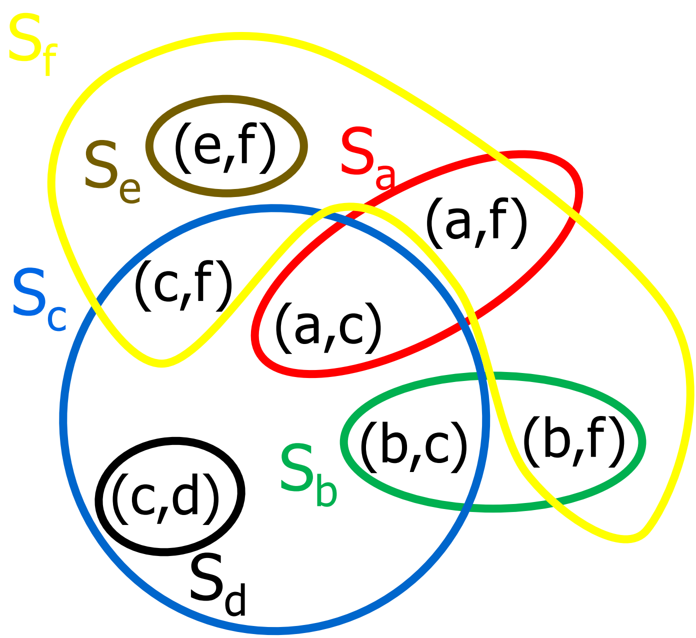
Problema di decisione: dato il grafo non orientato, esiste un vertex-cover di cardinalità ≤ 2?



$G = (V, E)$
$V = \{a,b,c,d,e,f\}$
$E = \{(a,c),(a,f),(b,c)$
$(b,f),(c,d),(c,f)(e,f)\}$

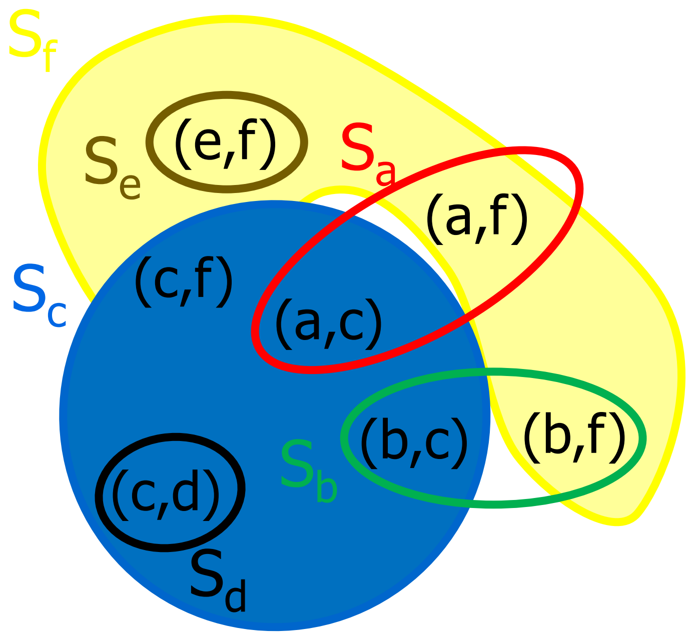
vertex-cover $\leq_P$ set-cover

Creare un problema di decisione set-cover con $U = E$ e $S_i = \{$archi che insistono sul vertice i$\}$



$U = \{(a,c),(a,f),(b,c)$
$(b,f),(c,d),(c,f)(e,f)\}$
$S_a = \{(a,c),(a,f)\}$
$S_b = \{(b,c),(b,f)\ \}$
$S_c = \{(a,c),(b,c),(c,d),(c,f)\}$
$S_d = \{(c,d)\}$
$S_e = \{(e,f)\}$
$S_f = \{(a,f),(b,f),(c,f)(e,f)\}$

Risolvere il problema di set-cover



$U = \{(a,c),(a,f),(b,c)$
$(b,f),(c,d),(c,f)(e,f)\}$

$S_a = \{(a,c),(a,f)\}$

$S_b = \{(b,c),(b,f)\ \}$

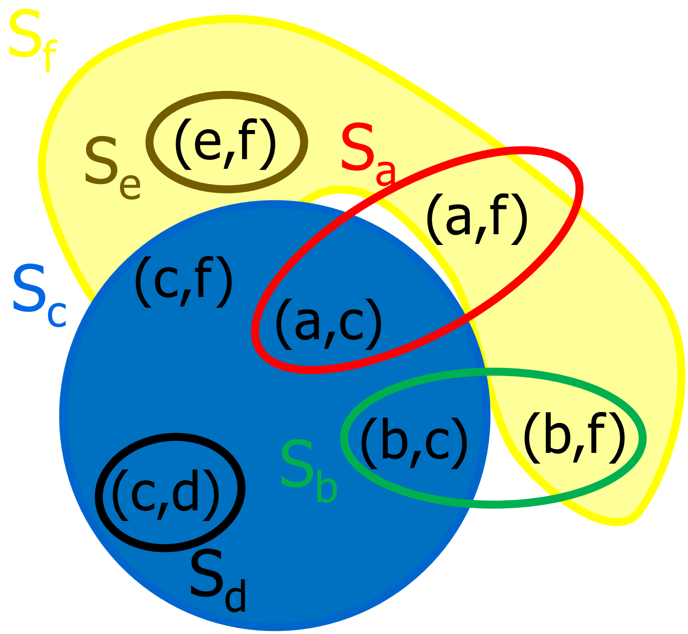
$S_c = \{(a,c),(b,c),(c,d),(c,f)\}$

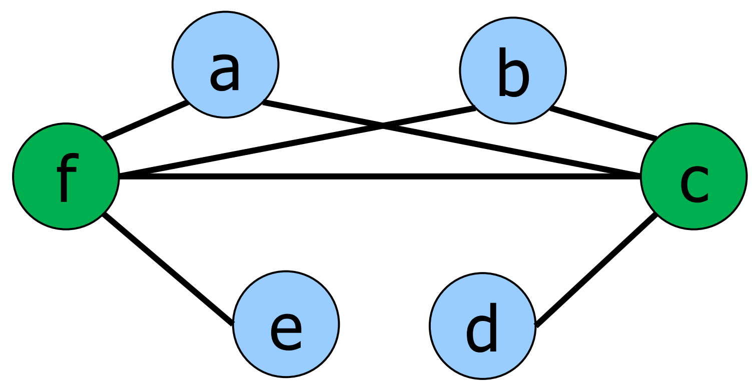
$S_d = \{(c,d)\}$

$S_e = \{(e,f)\}$

$S_f = \{(a,f),(b,f),(c,f)(e,f)\}$

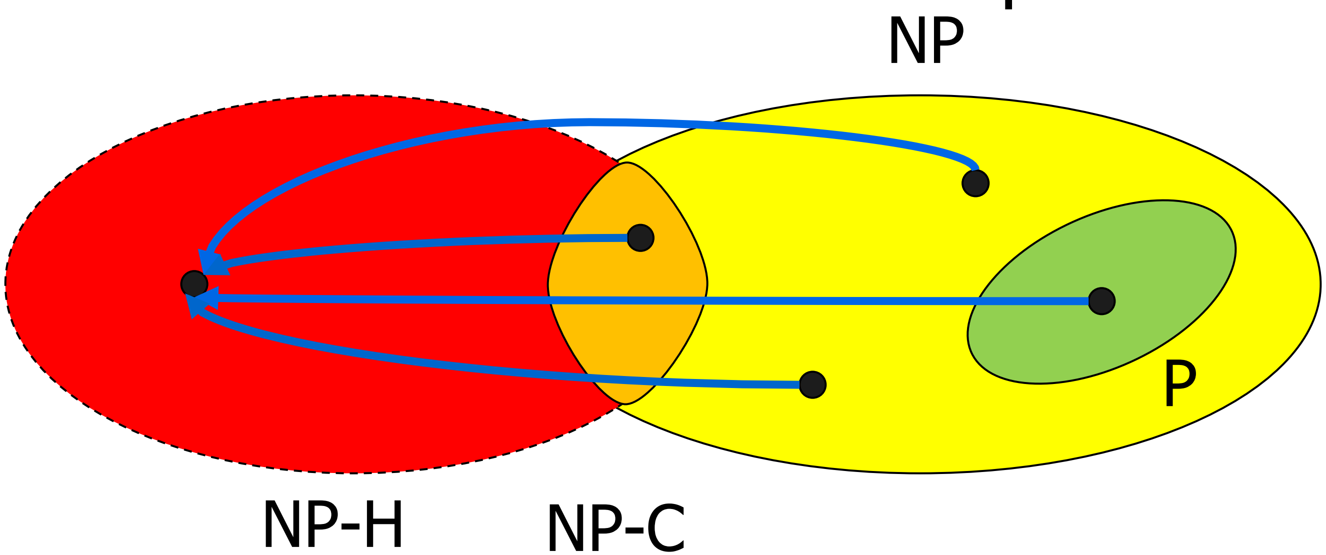
Risolvere il problema di vertex-cover



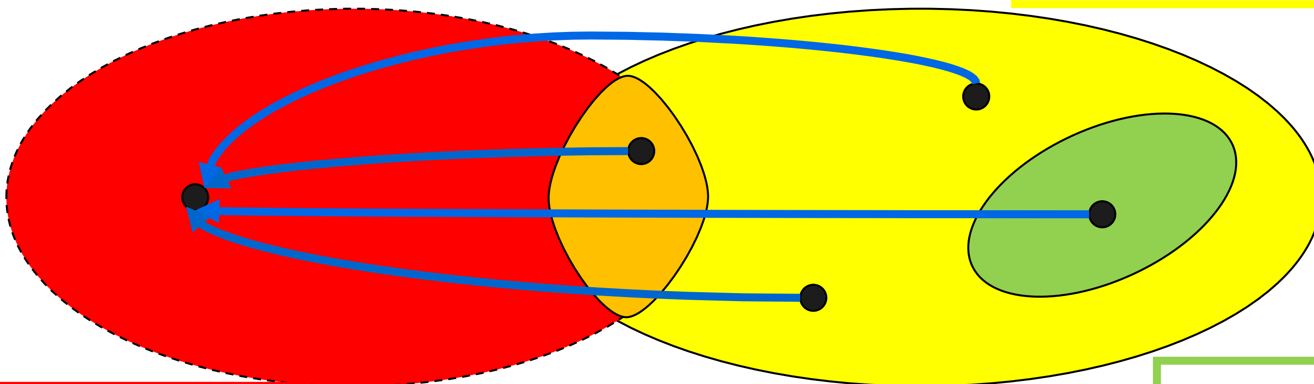


La Classe NP-H

- Un problema è NP-**hard** se ogni problema in NP è riducibile ad esso in tempo polinomiale (anche se non appartiene ad NP)
- ogni altro problema in NP è riducibile ad esso attraverso una trasformazione polinomiale



NP:
- Fattorizzazione
- Isomorfismo grafi



NP-H:
- Permanente di una matrice

NP-C:
- Soddisfacibilità
- Ciclo di Hamilton
- Clique

P:
- Connettività grafi
- Primalità
- Determinante

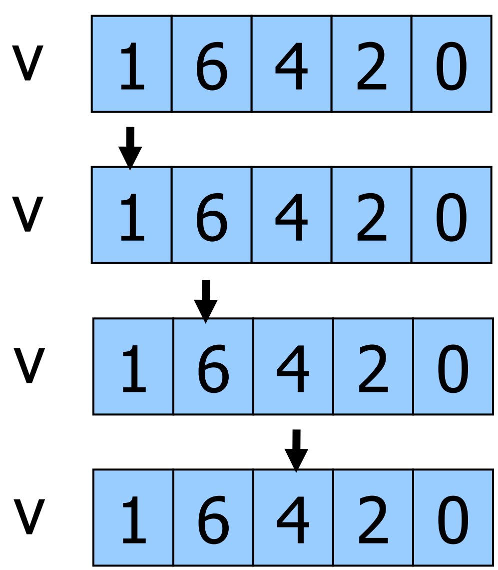
Algoritmi di ricerca su vettori

Ricerca: la chiave k è presente all'interno di un vettore v[N]? Sì/No

- Input: v[N], k
- Output: sì/no, se sì in quale posizione (indice del vettore)

Algoritmo 1: Ricerca sequenziale

Ricerca sequenziale: scansione del vettore da primo potenzialmente fino all’ultimo elemento, confronto con la chiave k



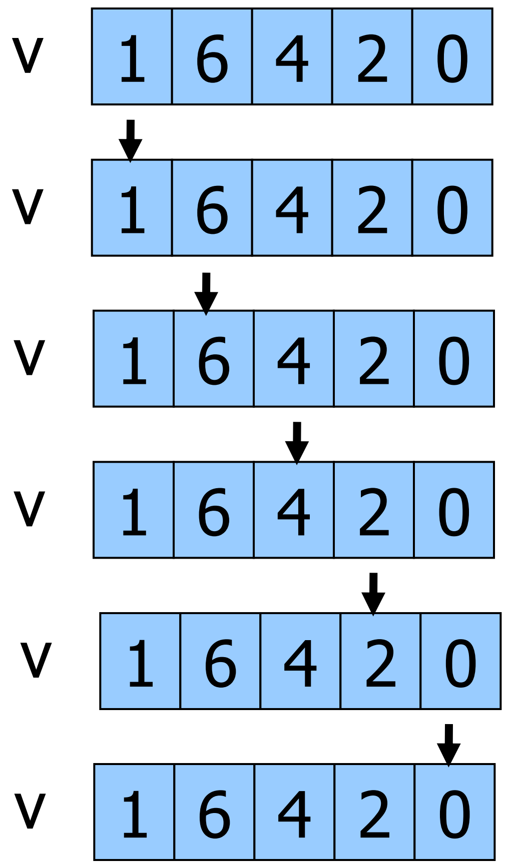
k 4

Ricerca con successo

v[0] ≠ k

v[1] ≠ k

v[2] = k, return indice =2



k 8

$v[0] \neq k$

$v[1] \neq k$

$v[2] \neq k$

$v[3] \neq k$

$v[4] \neq k$, return indice = -1

Ricerca con fallimento

04linear-search.c

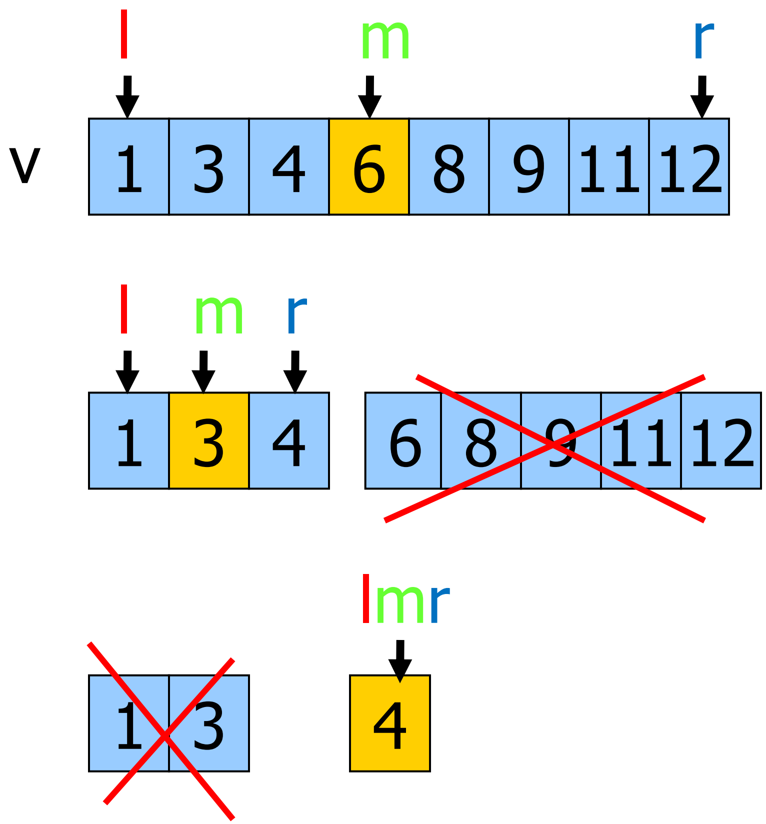
```c
int LinearSearch(int v[], int l, int r, int k) {
  int i = l;
  int found = 0;

  while (i <= r && found == 0)
    if (k == v[i])
      found = 1;
    else
      i++;

  if (found == 0)
    return -1;
  else
    return i;
}
```

Algoritmo 2: Ricerca binaria o dicotomica

Ricerca binaria o dicotomica: la chiave k è presente all'interno di un vettore ordinato v[N]? Sì/No

Approccio

- ad ogni passo: confronto k con elemento centrale del vettore:
  - =: terminazione con successo
  - <: la ricerca prosegue nel sottovettore di SX
  - >: la ricerca prosegue nel sottovettore di DX



k 4

y = elemento di mezzo

l = indice estremo di SX

r = indice estremo di DX

m = indice elemento di mezzo

Ricerca con successo

v[2] = k, return indice =2

05iterative-binary-search.c

```c
int BinSearch(int v[], int l, int r, int k) {
  int m, found;

  while(l <= r){
    m = l + (r - l)/2;
    if(v[m] == k)
      return(m);
    if(v[m] < k)
      l = m+1;
    else r = m-1;
  }
  return(-1);
}
```

Algoritmo 3: Insertion Sort

Ordinamento:

- Input: simboli $<a_1, a_2, \ldots, a_n>$ di un insieme con relazione d'ordine totale $\leq$
- Output: permutazione $<a'_1, a'_2, \ldots, a'_n>$ dell'input per cui vale la relazione d'ordine $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Relazione d'ordine $\leq$:

relazione binaria tra elementi di un insieme A che soddisfa le seguenti proprietà:

- riflessiva $\forall\ x \in A\ x \leq x$
- antisimmetrica $\forall\ x, y \in A\ \ x \leq y \wedge y \leq x \Rightarrow x = y$
- transitiva $\forall\ x, y, z \in A\ \ x \leq y \wedge y \leq z \Rightarrow x \leq z$

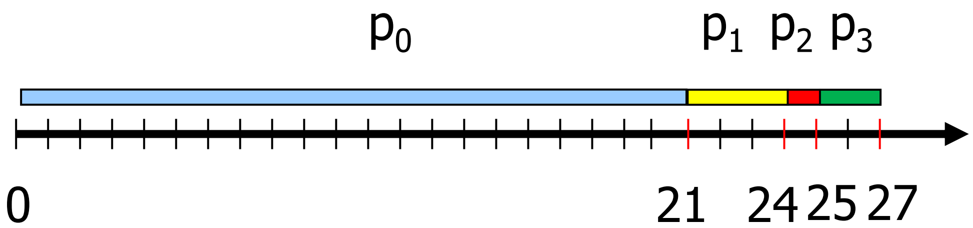
A è un insieme parzialmente ordinato (poset). Se la relazione $\leq$ vale $\forall\ x, y \in A$, A si dice totalmente ordinato

Esempi di relazione d'ordine $\leq$:

- relazione (totale) $\leq$ su numeri naturali, relativi, razionali e reali N, Z, Q, R, ordine alfabetico di stringhe, ordine cronologico di date
- relazione (parziale) di divisibilità su naturali escluso 0

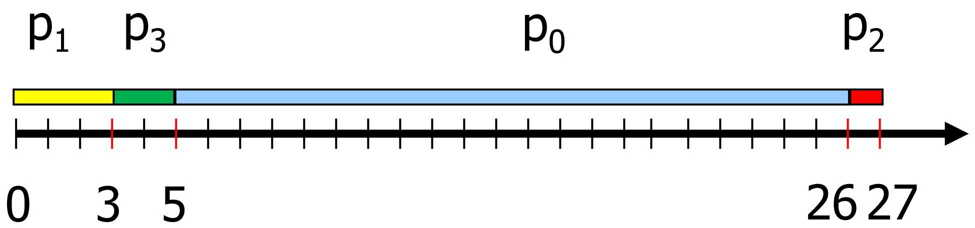
L'importanza dell'ordinamento

- Il 30% dei tempi di CPU è per ordinare dati.
- Esempio: CPU scheduling: processi p_i con durata, impatto dell'ordinamento sul tempo medio di attesa: p_0 21, p_1 3, p_2 1, p_3 2
  - scheduling p_0-p_1-p_2-p_3



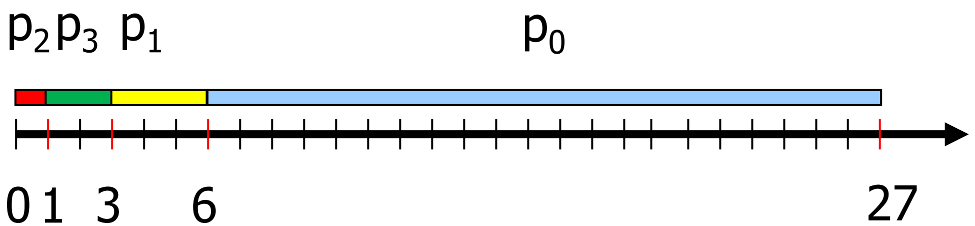
tempo medio di attesa (0+21+24 +25)/4 =17,5

- scheduling p_1-p_3-p_0-p_2



tempo medio di attesa (0+3+5 +26)/4 =8,5

- scheduling (ordinamento) p_2-p_3-p_1-p_0



tempo medio di attesa (0+1+3 +6)/4 =2,5

Applicazioni dell'ordinamento

- ordinamento di una lista di nomi
- organizzazione di un libreria MP3
- visualizzazione dei risultati di Google PageRank
- ...

applicazioni ovvie

- trovare la mediana
- ricerca binaria in un database
- trovare i duplicati in una mailing list
- ...

problemi semplici se i dati sono ordinati

- compressione dei dati
- computer graphics (es. inviluppo complesso)
- biologia computazionale
- ...

applicazioni non banali

Approccio

- Dati: interi in un vettore A con indici compresi tra l e r
- Vettore diviso in 2 sotto-vettori:
  - di sinistra: ordinato
  - di destra: disordinato
- Un vettore di un solo elemento è ordinato
- Approccio incrementale: ad ogni passo si espande il sotto-vettore ordinato inserendovi un elemento (invarianza della proprietà di ordinamento)
- Terminazione: tutti gli elementi inseriti ordinatamente.

Passo i-esimo: inserimento ordinato

Passo i-esimo: collocazione nella posizione corretta di $x = A_i$

- scansione del sotto-vettore ordinato (da A_{i-1} a A_0) fino a trovare un $A_j > A_i$
- shift a destra di una posizione degli elementi da A_j ad A_{i-1}
- inserimento di A_i nella posizione corretta.

Esempio

Già ordinati

Non ancora considerati

6	3	5	1	7	21	8	4	12	0	9	8	4

i=1

Elemento da inserire x **3**

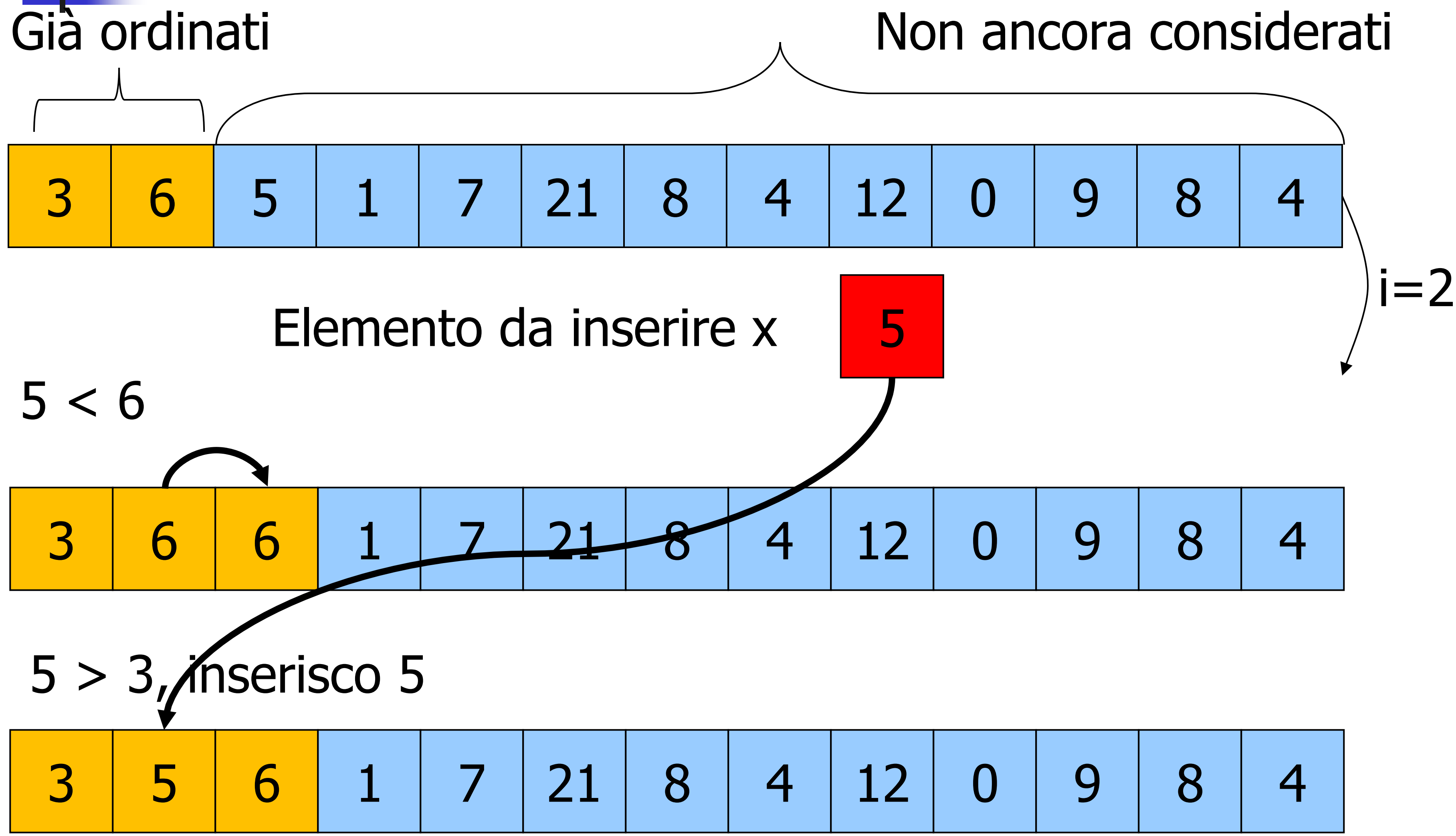
3 < 6

6	6	5	1	7	21	8	4	12	0	9	8	4

raggiunto l’estremo sinistro, inserisco 3

3	6	5	1	7	21	8	4	12	0	9	8	4

Esempio


Già ordinati
Non ancora considerati
3 6 5 1 7 21 8 4 12 0 9 8 4
i=2
Elemento da inserire x
5
5 < 6
3 6 6 1 7 21 8 4 12 0 9 8 4
5 > 3, inserisco 5
3 5 6 1 7 21 8 4 12 0 9 8 4

Esempio

Già ordinati

Non ancora considerati

3	5	6	1	7	21	8	4	12	0	9	8	4

i=3

Elemento da inserire x

1

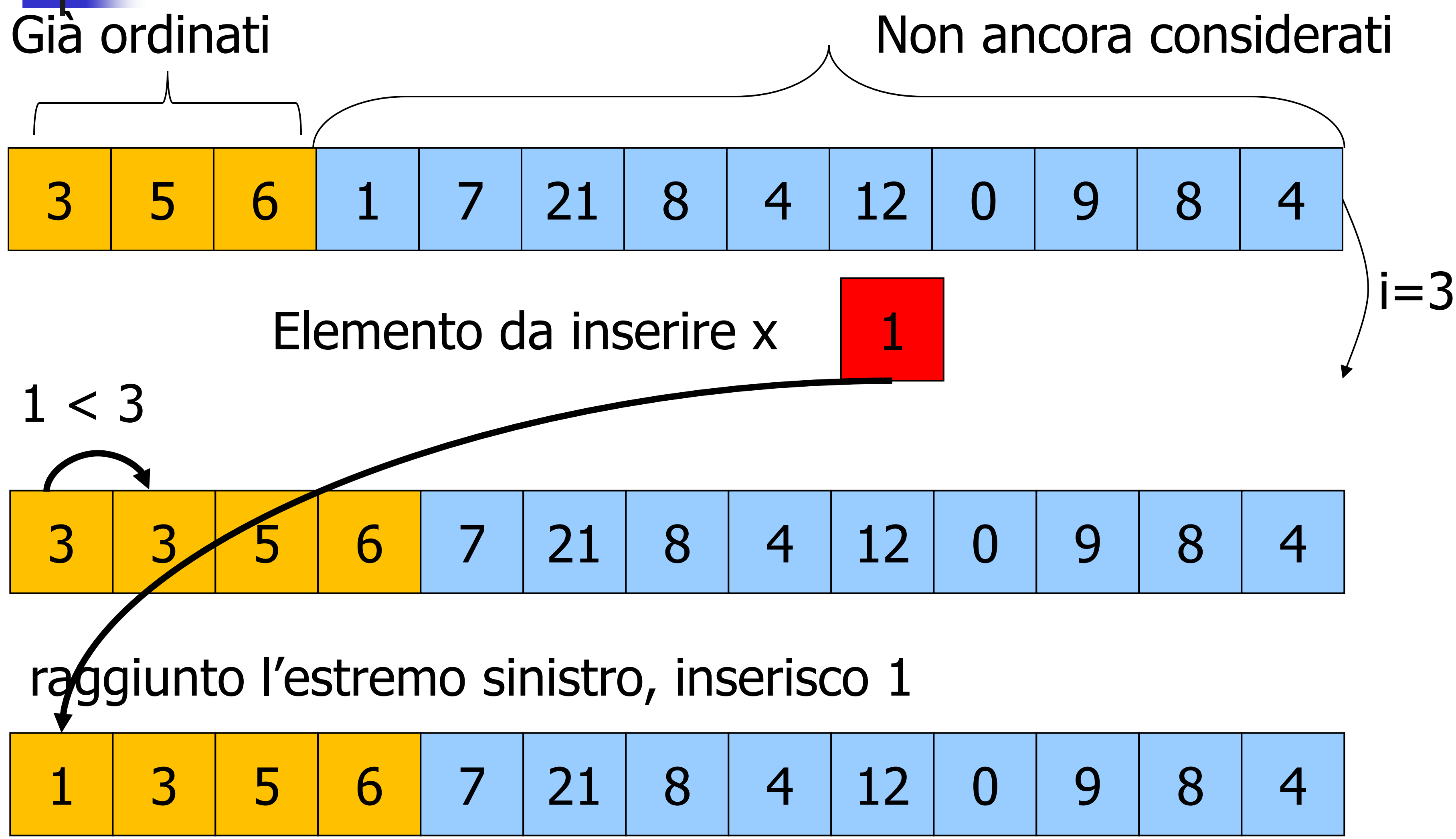
$1 < 6$

3	5	6	6	7	21	8	4	12	0	9	8	4

$1 < 5$

3	5	5	6	7	21	8	4	12	0	9	8	4

Esempio


Già ordinati
Non ancora considerati
3 5 6 1 7 21 8 4 12 0 9 8 4
i=3
Elemento da inserire x
1
1 < 3
3 3 5 6 7 21 8 4 12 0 9 8 4
raggiunto l’estremo sinistro, inserisco 1
1 3 5 6 7 21 8 4 12 0 9 8 4

06insertion-sort.c

```c
void InsertionSort(int A[], int l, int r) {
  int i, j, x;
   for(i = l+1; i <= r; i++) {
     x = A[i];
     j = i - 1;
     while (j >= l && x < A[j]){
       A[j+1] = A[j];
       j--;
     }
     A[j+1] = x;
   }
}
```

Riferimenti

- Algoritmi:
  - C. Toffalori «Algoritmi», Collana «Raccontare la Matematica», Il Mulino, 2015
- Congettura di Collatz:
  - G. Gopalakrishnan «Computation Engineering», Springer 2006, 2.9
- Congettura di Goldbach:
  - Crescenzi 1.2

- CPU scheduling:
  - Crescenzi 2.2
- Insertion sort:
  - Cormen 1.1
  - Sedgewick 6.1