Relazioni tra classi di complessità

A.A. 2017-2018

Relazione tra le classi dei problemi NP ed NP-completi

Teorema 1

P $\neq$ **NP** $\Rightarrow$ esiste un problema Q $\in$ **NP** – **P** non **NP**-completo

I problemi in **NP** – **P** che non sono **NP**-completi vengono chiamati **NP**-intermedi.

Corollario

P $\neq$ **NP** $\Rightarrow$ **NP**-completi $\subset$ **NP** – **P**

Teorema 2

P $\neq$ **NP** $\Rightarrow$ esistono due problemi Q e R, **NP**-intermedi, tali che Q $\not\propto$ R e R $\not\propto$ Q, cioè Q ed R non sono confrontabili nella relazione $\propto$.

Relazione tra le classi dei problemi NP ed NP-completi

Tra i candidati per essere problemi NP-intermedi troviamo:

- Isomorfimo tra grafi. Dati due grafi G_1 e G_2 esiste una biiezione *h* tra i vertici di G_1 e G_2 che conserva tutti gli archi (cioè (i,j) è un arco in G_1 se e solo se (h(i), h(j)) è un arco in G_2 ?

- Fattorizzazione. Dato un intero N e un intero M<N esiste un intero d $\leq$M tale che d sia un divisore di N?

NP-completezza forte

SOMMA DI SOTTONSIEME

ISTANZA: Un insieme $A = \{a_1,\ldots,a_n\}$ di interi positivi e un intero k.
DOMANDA: Esiste un sottoinsieme $S \subseteq A$ tale che:

$$\sum_{h \in S} a_h = k \quad ?$$

SOMMA DI SOTTOINSIEME è un altro dei (tanti) problemi che si dimostrano NP-completi

NP-completezza forte

Abbiamo dimostrato che "**somma di sottinsieme**" è **NP**-completo.

L'algoritmo di programmazione dinamica basato sulla seguente definizione, dove

$$P[i, j] = \text{true} \quad \Leftrightarrow \quad \exists S' \subseteq \{a_1,\ldots,a_i\}: \sum_{h \in S'} a_h = j$$

$P[i, 0] = \textit{true}$ $\qquad 0 <= i <= n$

$P[0, j] = \textit{false}$ $\qquad 1 <= j <= k$

Per $i \neq 0$ e $j \neq 0$

$$P[i, j] = \begin{cases} P[i-1, j] & \text{se } j - a_i < 0 \\ P[i-1, j] \text{ or } P[i-1, j-a_i] & \text{se } j - a_i >= 0 \end{cases}$$

ha complessità in tempo $O(n \cdot k)$.

NP-completezza forte: somma di sottinsieme

```
Subset-Sum (n, W)
        M: array [0..n, 0..W]
        for w ← 0 to W M[0, w] = 0
        for i ← 1 to n
              for w ← 0 to W
              M[i, w] ← max (M(i-1, w), w_i + M(i-1, w-w_i))
        return M[n, W]
```

```
Subset-Sum (n, k)
        P: array [0..n, 0..k]
        for j ← 0 to k P[0, j] = 0
        for i ← 1 to n
              for j ← 0 to k
              P[i, j] ← P(i-1, j) or P(i-1, j - a_i))
        return P[n, k]
```

NP-completezza forte

Assumendo $a_i <= k$ $(1<= i <= n)$, la dimensione dei dati in ingresso è $\mathbf{O}(n \cdot \log(k))$, pertanto la complessita` $\mathbf{O}(n \cdot k)$ non e` polinomiale nella dimensione del problema. Se k è limitato da un polinomio in n , cioè k è $\mathbf{O}(n^t)$, con t costante, allora *Sommasott* ha complessità polinomiale $\mathbf{O}(n^{t+1})$. Se invece k è superpolinomiale in n, anche la complessità dell'algoritmo risulta superpolinomiale.
(se k è $\mathbf{O}(2^n)$, $n \cdot k = \mathbf{O}(n \cdot 2^n)$).

La complessità di "**Somma di sottinsieme**" non dipende soltanto dalla cardinalità n dell'insieme in input, ma anche dai valori degli elementi. Tale complessità risulta polinomiale o meno a seconda che i valori degli elementi in ingresso siano "piccoli" o "grandi" rispetto al loro numero.

NP-completezza forte

Anche per "**zaino**" sappiamo che è **NP**-completo e abbiamo costruito un algoritmo di programmazione dinamica di complessità $\mathbf{O}(n \cdot C)$, con n numero di oggetti e C capacità dello zaino.

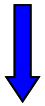
$n \cdot C$ ***non è una funzione polinomiale della dimensione dell'input***, in quanto la lunghezza dell'input è $\mathbf{O}(n \cdot \log(C))$.

Per "**Zaino**" possiamo fare considerazioni analoghe a quelle fatte per "**Somma di sottinsieme**". Se C è limitato da un polinomio in n, cioè C è $\mathbf{O}(n^t)$, con t costante, allora "Zaino" ha complessità polinomiale $\mathbf{O}(n^{t+1})$. Se invece C è superpolinomiale in n, anche la complessità dell'algoritmo risulta superpolinomiale.

NP-completezza forte

È importante distinguere tra **Somma di sottinsieme** e **Zaino**, da una parte, e problemi come **Cricca** e **Ciclo Hamiltoniano**, dall'altra.

In questi ultimi i numeri sono usati solo come nomi di nodi e indici di variabili; questi problemi restano **NP**-completi anche se ogni istanza di lunghezza n è ristretta a interi di dimensione al più $p(n)$ (funzione polinomiale in n).



problemi *fortemente NP-completi*

Somma di sottinsieme e **Zaino** non sono invece fortemente **NP**-completi.

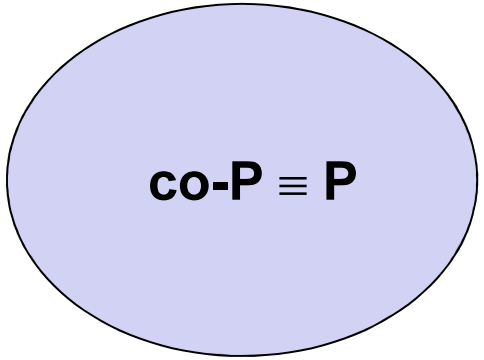
Classi co-P e co-NP

Sia Q un problema di decisione. Il **problema complemento** di Q: Q^C è il problema decisionale che, definito sullo stesso dominio, dà risposta SI quando Q dà risposta NO e risposta NO quando Q dà risposta SI.

co-**P** e co-**NP** indicano le classi dei problemi complemento dei problemi in **P** e in **NP** rispettivamente.

È facile dimostrare:

P ≡ co-**P**



Classi co-P e co-NP

Problema: **dato un intero n, n è pari?**

```
Pari (n)
if (n mod 2 = 0) return  true
else  return  false
```

Il problema ha un algoritmo di soluzione deterministico di complessità polinomiale, cioè è in **P**.

Problema complemento: **dato un intero n, n è non pari?**

E' risolto dallo stesso algoritmo in cui true e false vengono scambiati

```
Dispari (n)
if (n mod 2 = 0) return  false
else  return  true
```

Classi co-P e co-NP

Se Q $\in$ **NP** in generale non si può dire *niente* sul suo complemento.

Si può comunque dire che Q^C *non* è più facile di Q, anzi, assumendo **P** $\neq$ **NP**, Q^C è presumibilmente più difficile di Q.

Per i problemi in **NP** esiste una funzione booleana calcolabile in tempo polinomiale da un algoritmo deterministico (il certificato), che, dato un input x e una soluzione y, verifica che y è proprio una soluzione (testimone) nella formula:

$$\exists y\ P(x, y) = true$$

Per i problemi complemento si deve invece verificare che non esista nessuna soluzione y, cioè:

$$\forall y\ P(x, y) = false$$

Classi co-P e co-NP

Il passaggio da un quantificatore esistenziale ad uno universale complica il problema (se $P \neq NP$).

Ad esempio, mentre si sa che **Soddisfacibilità** è in **NP**, non si sa se il suo complemento, **Falsità**, è in **NP** oppure no.

Soddisfacibilità: data una formula booleana in forma normale congiuntiva esiste una assegnazione di valori di verità alle variabili booleane che rende la formula vera?

Falsità: data una formula booleana in forma normale congiuntiva tale formula è falsa per tutte le possibili assegnazioni di valori di verità alle variabili booleane?

Relazione tra le classi NP e co-NP

Teorema 3

co-NP $\neq$ **NP** $\Rightarrow$ **P** $\neq$ **NP**

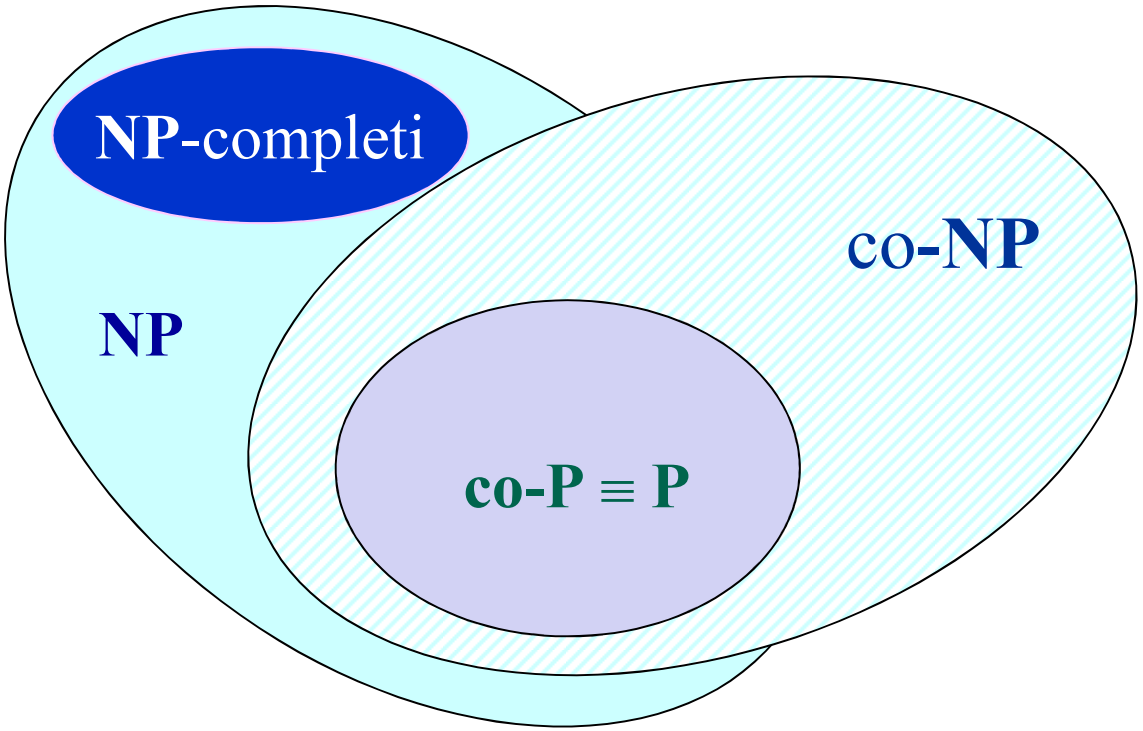
La dimostrazione per assurdo è facile: supponendo **P = NP** avremmo **NP = P = co-P = co-NP**. Una contraddizione!

Teorema 4

Se esiste un problema Q, **NP**-completo, il cui problema complemento $Q^C \in$ **NP**, allora co-**NP** = **NP**.

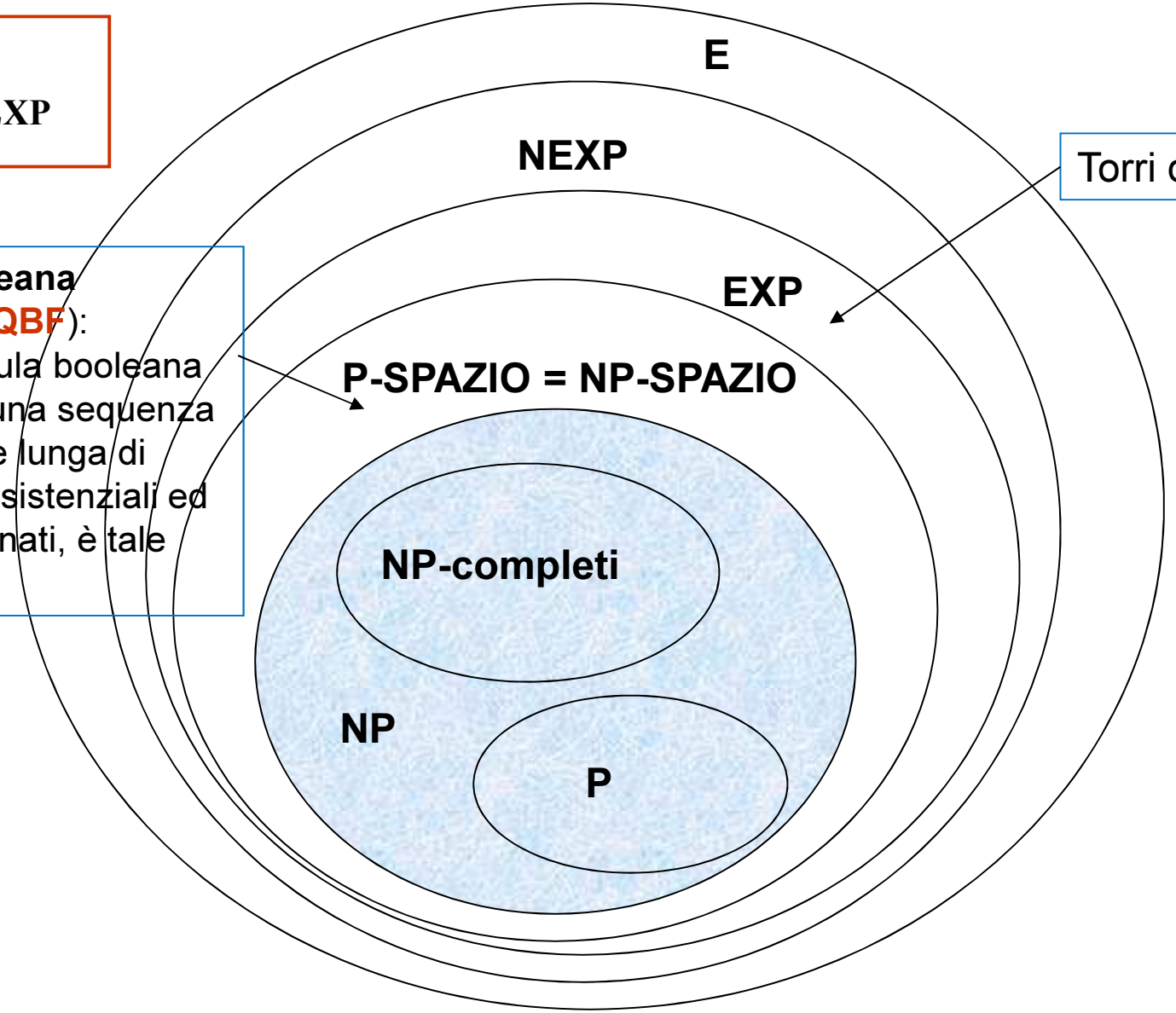
Classi di complessità

L'ipotesi più accreditata è che si abbia la seguente relazione tra le classi viste.



Classi di complessita`: gerarchia

$P \subset EXP$
$NP \subset NEXP$



Formula booleana quantificata (QBF):
Data una formula booleana preceduta da una sequenza arbitrariamente lunga di quantificatori esistenziali ed universali alternati, è tale formula vera ?

La classe E

E: classe dei problemi di decisione risolubili con algoritmi ***deterministici*** in tempo ***esponenziale k-plo***, per k = 0, 1, 2, ...
cioè in tempo:

$$\bigcup_{k\geq 0} O\left[2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}\right] \quad (k \text{ volte } 2)$$

La classe **E** coincide con la classe delle funzioni "**elementari**", cioè quelle *definibili mediante un numero finito di applicazioni delle quattro operazioni elementari dell'aritmetica, dell'esponenziazione, dei logaritmi, dell'elevamento a potenza e dell'estrazione di radice (comprese le funzioni trigonometriche).*

N.B. vi sono problemi ancora più "difficili", che non sono elementari.

Problemi indecidibili

Alcuni problemi non possono essere risolti da un computer anche con tempo illimitato

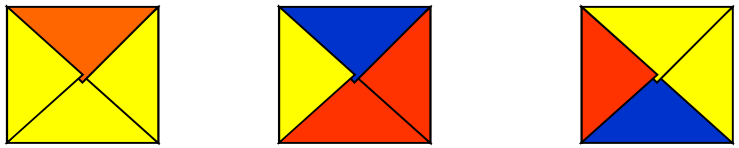
Esempio: Domino non limitato (The Tiling Problem)

• Una *tessera* e` un quadrato 1 × 1, diviso in 4 triangoli dalle sue diagonali, ogni triangolo ha un dato colore e ogni tessera ha un orientamento fisso

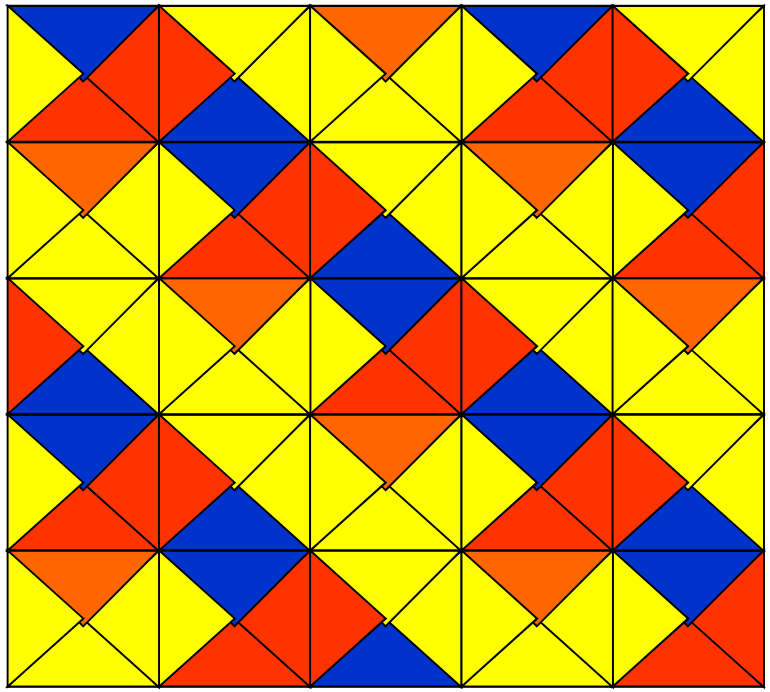
• *Istanza*: un insieme finito S of descrizioni di tessere
• *domanda*: usando le tessere di S in modo che i lati adiacenti abbiano lo stesso colore, e` possibile coprire un'area finita di dimensione arbitraria?

Problemi indecidibili: domino non limitato

Esempio di descrizioni di tessere:

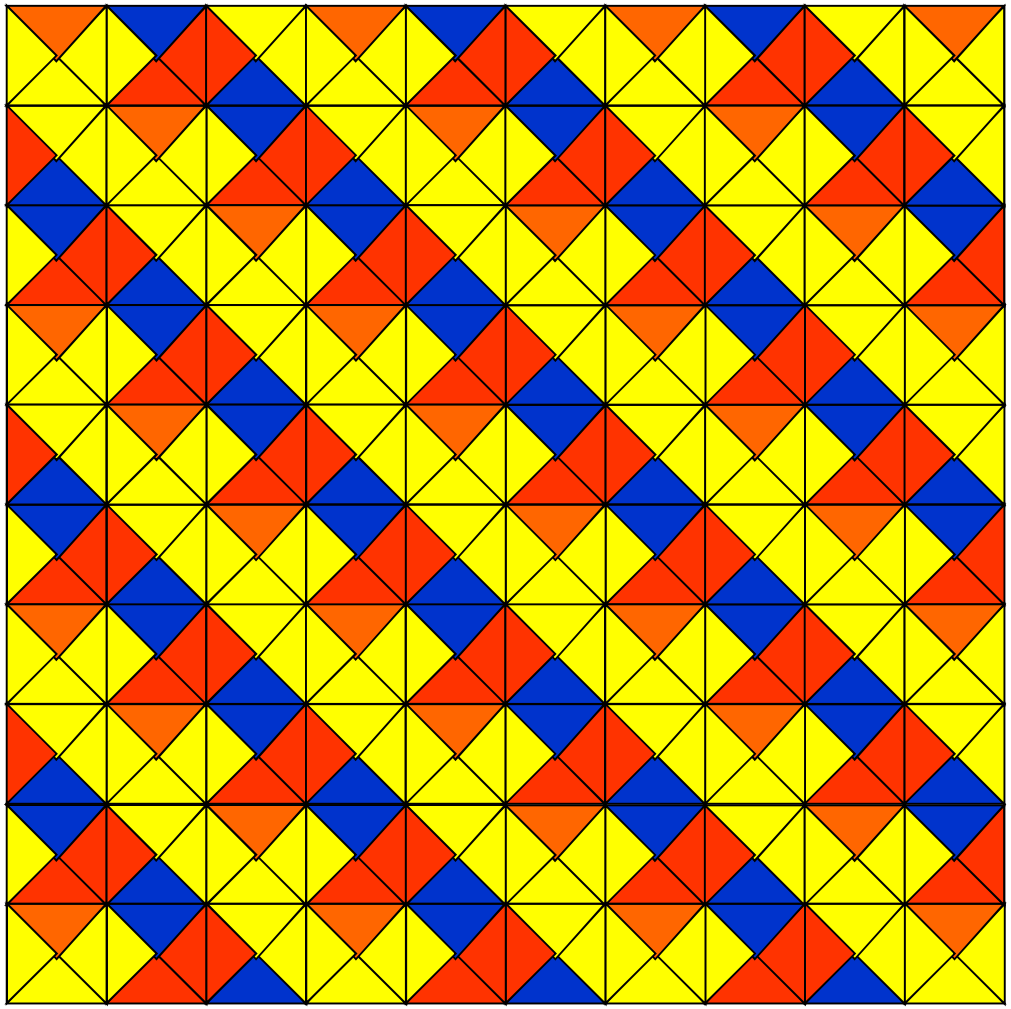


Esempio di copertura di un quadrato 5 x 5



Problemi indecidibili: domino non limitato

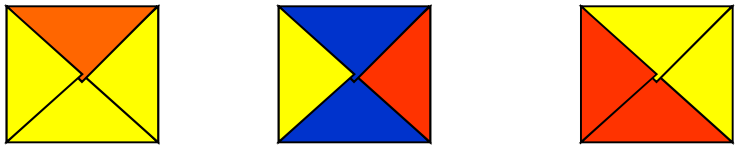
Esempio di copertura di un quadrato 10 x 10



Ogni area finita puo` essere riempita con le tessere date

Problemi indecidibili: domino non limitato

Modifichiamo le tessere:



Impossibile con esse riempire un quadrato 3 × 3

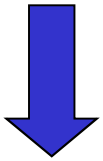
Non esiste nessun algoritmo per il Tiling Problem, cioe` per ogni algoritmo A che possiamo formulare vi e` un insieme di tessere S per cui:

(1) A non termina *oppure*

(2) A da' la risposta sbagliata

Problemi indecidibili

- Un problema P che non ammette nessun algoritmo e` detto *non-calcolabile*
- Se P e` un problema di decisione e non ammette nessun algoritmo, P e` *indecidibile*



Il “Tiling Problem” e` indecidibile

Problemi indecidibili: corrispondenza di Post (PCP)

- Una *parola* e` una stringa finita su un dato alfabeto
- *Istanza*: due sequenze finite di parole X(1), . . ., X(n) e Y(1), . . ., Y(n) sullo stesso alfabeto
- *Domanda*: esiste una sequenza di interi $i_1, i_2, \ldots, i_r$ in {1, . . ., n} tali che $X(i_1)X(i_2) \ldots X(i_r) = Y(i_1)Y(i_2) \ldots Y(i_r)$?
  Cioe`, concatenando le X e le Y nelle posizioni $i_1, i_2, \ldots, i_r$ si ottiene la stessa parola?

Esempio:

	1	2	3	4	5
X	abb	a	bab	baba	aba
Y	bbab	aa	ab	aa	a

La soluzione e` data dalla sequenza 2, 1, 1, 4, 1, 5

La stringa e`: aabbabbbabaabbaba da X e
aabbabbbabaabbaba da Y

Problemi indecidibili: corrispondenza di Post (PCP)

Modifichiamo le parole in input:

Togliamo la prima lettera sia da X(1) sia da Y(1)

	1	2	3	4	5
X	bb	a	bab	baba	aba
Y	bab	aa	ab	aa	a

non vi e` nessuna corrispondenza

Il problema della “corrispondenza di Post” e` indecidibile

Problemi indecidibili: problema dell'Halt (HP)

Un progetto impossibile:
scrivere un programma Q che ha in input:

- un programma legale X (ad esempio in C)
- una stringa S in input per il programma X

e ritorna come output

- *si* se X si ferma sull'input S
- *no* se X sull'input S entra in un loop infinito

HP e` indecidibile