Università di Pisa

Pietro Ducange

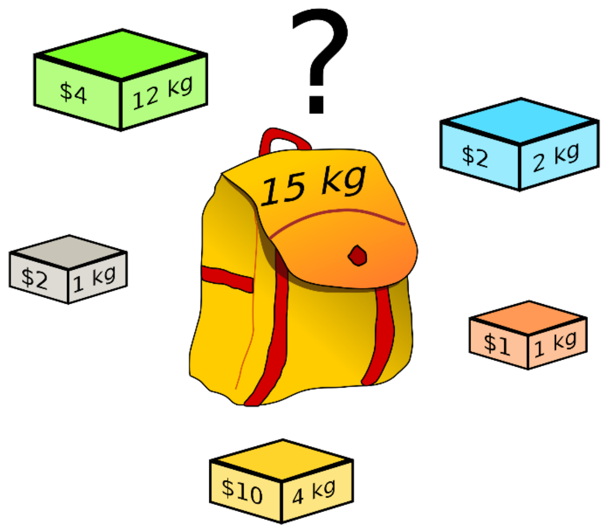
Algoritmi e strutture dati

a.a. 2019/2020

NP-completezza

Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione la maggior parte delle slide utilizzate nella presente lezione

Problemi difficili: cenni alla NP-completezza



Ottimizzare il riempimento dello zaino:

- **ogni oggetto ha un peso e un valore**
- **determinare il numero di oggetti di ogni tipo in modo tale che il peso sia minore o uguale di un dato limite (valore= capacità dello zaino) e il valore totale sia il maggiore possibile.**

Problemi difficili : commesso viaggiatore

trovare il percorso di minore lunghezza che un commesso viaggiatore deve seguire per visitare tutte le città una e una sola volta per poi tornare alla città di partenza

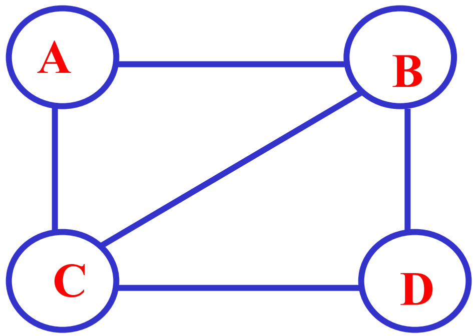
Problemi difficili : ciclo Hamiltoniano

Willian Rowan Hamilton (1859)

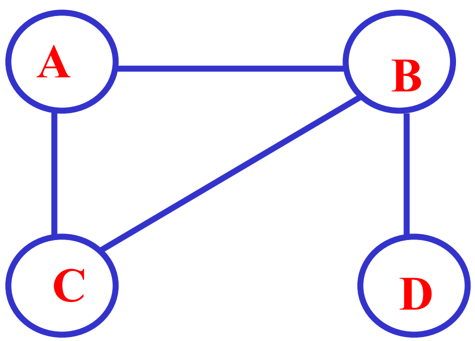
HC: Dato un multi-grafo, trovare, se esiste, un ciclo che tocca tutti i nodi una e una sola volta

Un grafo è Hamiltoniano se possiede un ciclo Hamiltoniano

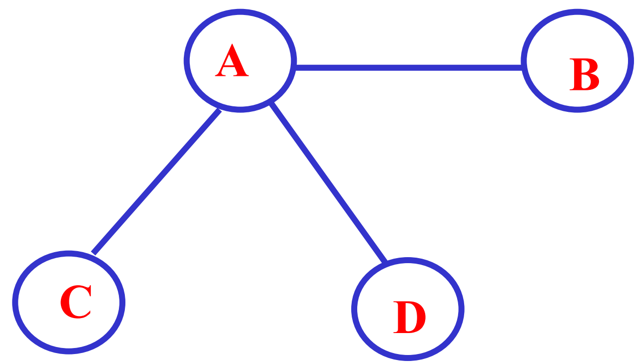
ciclo Hamiltoniano



ABDCA

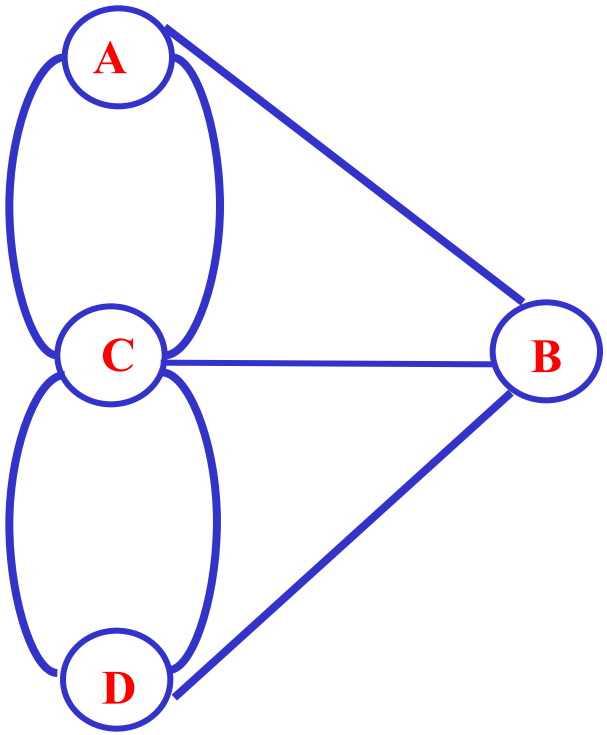


NO



NO

ciclo Hamiltoniano



ACDBA

SAT : soddisfattibilità di una formula nella logica dei predicati

Data una formula F con n variabili, trovare, se esiste, una combinazioni di valori booleani che, assegnati alle variabili di F, la rendono vera

Es.

F= (x and not x) or (y and not y)   n=2   non sodd.

F= (x and not y) or (not x and y)   n=2   x=0, y=1

Problemi difficili

algoritmi conosciuti oggi per zaino, commesso viaggiatore, ciclo Hamiltoniano e SAT:

provare tutte (o quasi) le combinazioni

↓

complessità esponenziale

SAT

Se le variabili che compaiono nella formula sono n, si provano tutte le combinazioni di valori delle variabili, che sono 2^n

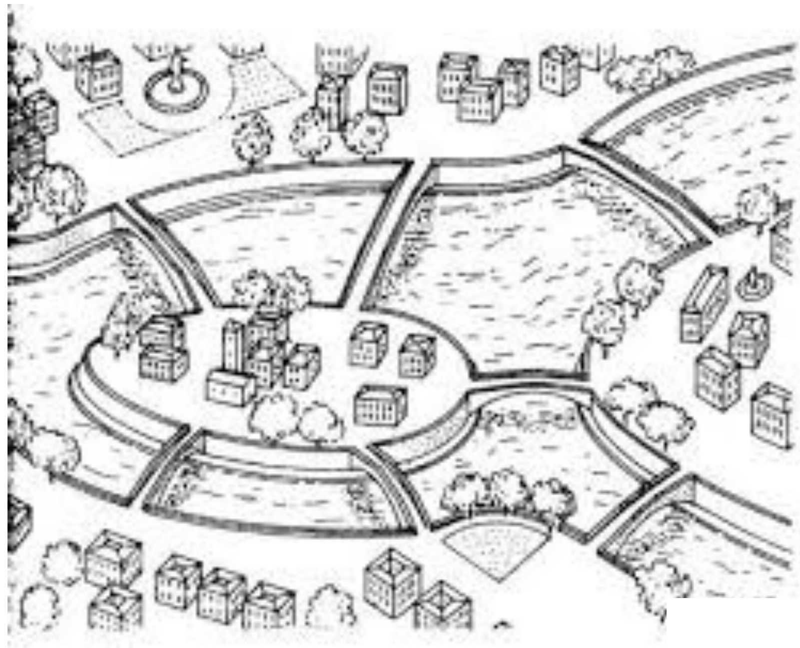
Problema del ciclo Euleriano

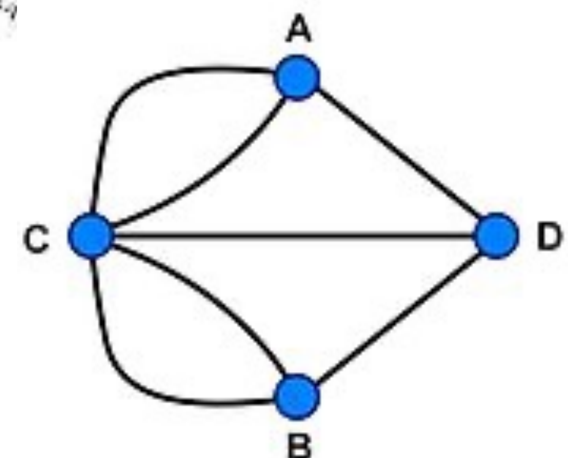
Eulero (1736)

Dato un multi-grafo, trovare, se esiste, un ciclo che percorre tutti gli archi una e una sola volta

Un grafo è Euleriano se possiede un ciclo Euleriano

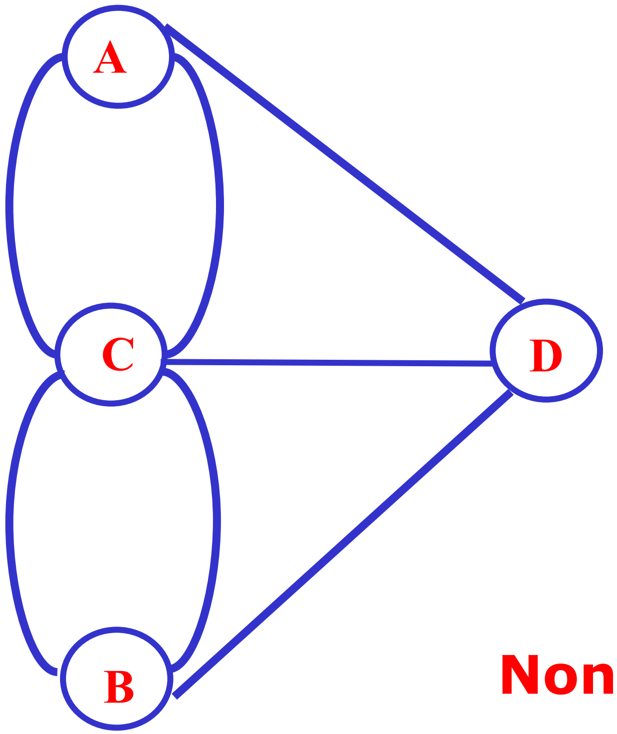
I ponti di Konigsberg





Immagini estratte da Wikipedia

I ponti di Konigsberg: esiste un ciclo Euleriano?



Non esiste un ciclo Euleriano

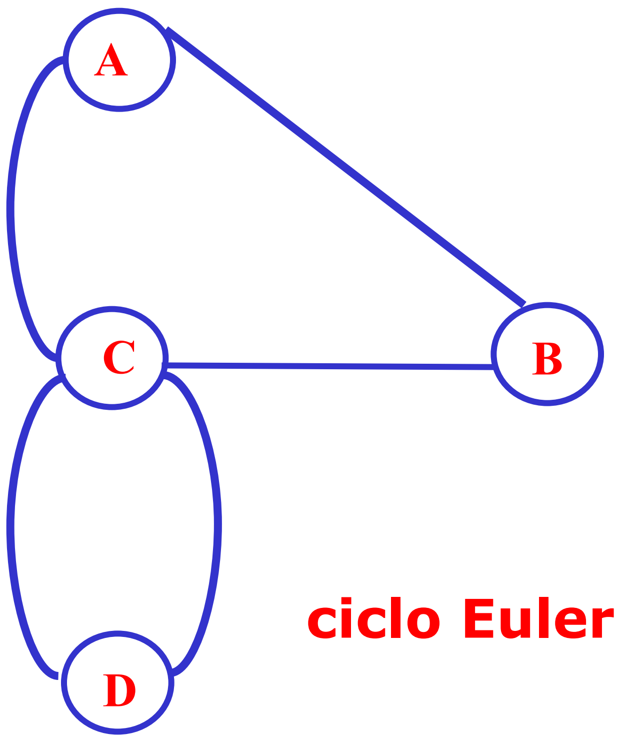
TEOREMA DI EULERO

Un multi-grafo non orientato contiene un ciclo Euleriano se e solo se gli archi che partono da ogni nodo sono in numero pari.

Quindi basta controllare questa proprietà sul grafo (O(m)) per sapere se un ciclo esiste.

Trovarlo ha complessità polinomiale.

ciclo Euleriano



ciclo Euleriano: ABCDCA

Teoria della NP-completezza

Classifica un insieme di problemi difficili ma non dimostrati come esponenziali (come Hanoi, permutazioni, …)

Si applica a problemi decisionali: con risposta SI o NO

Ogni problema può essere riformulato come problema decisionale.

Il problema decisionale ha complessità minore o uguale al problema non decisionale corrispondente.

Quindi se il problema decisionale è difficile, a maggior ragione lo sarà il corrispondente.

Problemi decisionali

Commesso viaggiatore: dato un intero **k**, esiste nel grafo un ciclo senza ripetizione di nodi di lunghezza minore di **k**?

Zaino: dato un valore **v**, esiste un riempimento dello zaino con valore maggiore o uguale a **v**?

Ciclo Hamiltoniano: dato un grafo, esiste un ciclo Hamiltoniano?

Ciclo Euleriano: dato un grafo, esiste un ciclo Euleriano? O(m)

Formula logica: data una formula, esiste un assegnamento alle variabili che rende vera la formula?

Algoritmi nondeterministici

Si aggiunge il comando

choice(I)

dove I è un insieme

choice(I) **sceglie arbitrariamente un elemento dell'insieme I**

Un algoritmo nondeterministico per la soddisfattibilità

```
int nsat(Formula f,int *a,int n) {
   for (int i=0; i < n; i++)
         a[i]=choice({0,1});
   if (value(f,a))
         return 1;
         else
          return 0;
 }
```

O(n)

Restituisce 1 se esiste almeno una scelta con risultato 1

Un algoritmo nondeterministico di ricerca in array

```
int nsearch(int* a, int n, int x) {
   int i=choice({0..n-1});
   if (a[i]==x)
        return 1;
   else
        return 0;
 }
```

O(1)

Un algoritmo nondeterministico di ordinamento

```
int nsort(int* a, int n) {
    int b [n];
    for (int i=0; i<n; i++)
                b[i]=a[i];
    for (int i=0; i<n; i++)
                a[i]=b[choice({0..n-1})];
    if (ordinato(a))
          return 1;
    return 0;
 }
```

O(n)

Relazione fra determinismo e nondeterminismo

Per ogni algoritmo nondeterministico ne esiste uno deterministico che lo simula, esplorando lo spazio delle soluzioni, fino a trovare un successo.

Se le soluzioni sono in numero esponenziale, l'algoritmo deterministico avrà complessità esponenziale.

P e NP

P = insieme di tutti i problemi decisionali risolubili in tempo Polinomiale con un algoritmo deterministico

NP = insieme di tutti i problemi decisionali risolubili in tempo Polinomiale con un algoritmo Nondeterministico

NP : Nondeterministico Polinomiale

P e NP

P = { ricerca, ordinamento, ciclo Euleriano …}

NP = { ricerca, ordinamento, fattorizzazione, soddisfattibilità, zaino, commesso viaggiatore, ciclo Hamiltoniano… }

Caratterizzazione alternativa della classe NP

NP = insieme di tutti i problemi decisionali che ammettono un algoritmo deterministico polinomiale di verifica di una soluzione (certificato)

Per dimostrare che un problema R appartiene a NP si dimostra che la verifica di una soluzione di R è fatta in tempo polinomiale

Esempi di verifica

Ciclo Hamiltoniano: si può verificare con complessità $O(n)$ se un cammino è un ciclo Hamiltoniano.

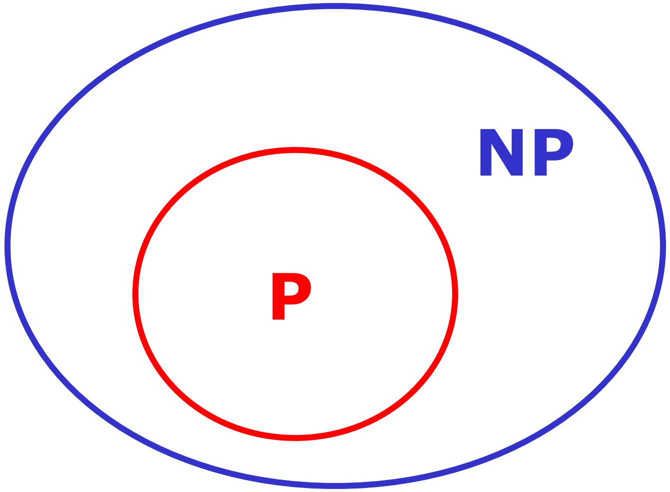
Formula logica: si può controllare in tempo $O(n)$ se dato un assegnamento di valori booleani alle variabili rende vera la formula

P e NP

P = problemi decisionali facili da **risolvere**

NP = problemi decisionali facili da **verificare**

P e NP



$P \subseteq NP$

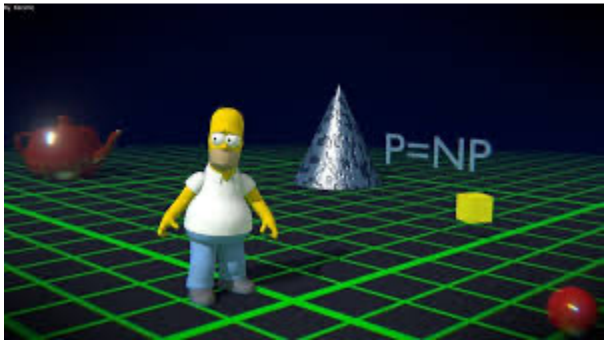
$P = NP$?

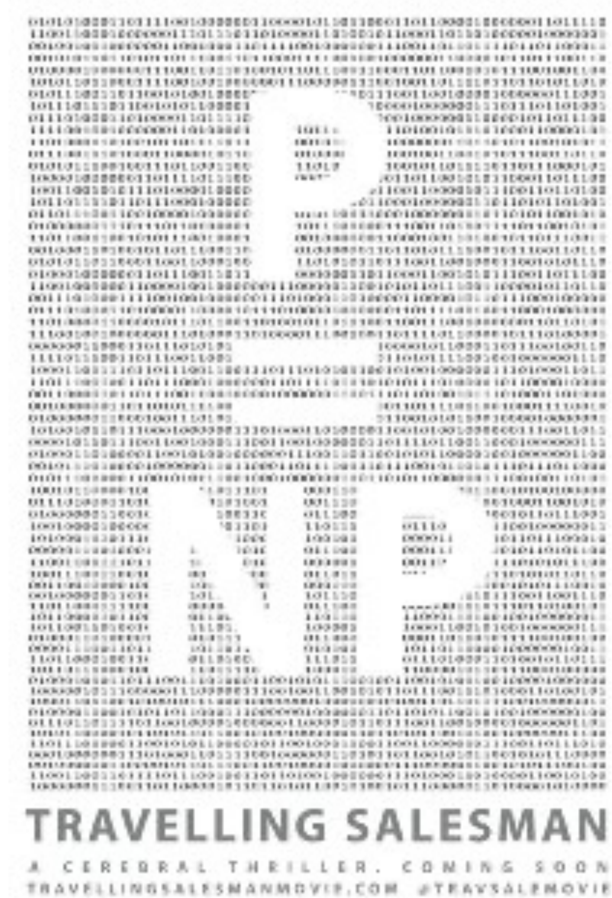
P = NP?

Uno dei 7 problemi del millennio (Clay Mathematical Institute, 2000)

Un milione di dollari per chi lo risolve (in un senso o nell'altro)

P = NP





«Macchine come me»
di Ian McEwan (2019)

Film del 2012

Riducibilità

La riducibilità è un metodo per convertire l'istanza di un problema P1 in un'istanza di un problema P2 e utilizzare la soluzione di quest'ultimo per ottenere la soluzione di P1

Riducibilità

Un problema P1 si riduce in tempo polinomiale a un problema P2 se ogni soluzione di P1 può ottenersi deterministicamente in tempo polinomiale da una soluzione di P2

input di P1 → **input di P2**

compl. polinomiale

P2

soluzione di P1 ← **soluzione di P2**

compl. polinomiale

$$\mathbf{P1 \le P2}$$

Conseguenze della riducibilità

- **P1 $\leq$ P2**
- **P2 è risolubile in tempo polinomiale**

P1 è risolubile in tempo polinomiale

Teorema di Cook

Qualsiasi problema R in NP è riducibile al problema della soddisfattibilità della formula logica :

$$\forall R \in NP : R \leq SAT$$

Quindi SAT è più difficile di tutti i problemi in NP

Un problema R è NP-completo se

- **R $\in$ NP e**
- **SAT $\leq$ R**

NP-completezza

Se un problema è NP-completo, è difficile tanto quanto SAT e può essere usato al posto di SAT nella dimostrazione di NP-completezza di un altro problema.

I problemi NP-completi hanno tutti la stessa difficoltà e sono i più difficili della classe NP

Se si trovasse un algoritmo polinomiale per SAT o per qualsiasi altro problema NP-completo, allora tutti i problemi in NP sarebbero risolubili in tempo polinomiale e quindi P sarebbe uguale ad NP

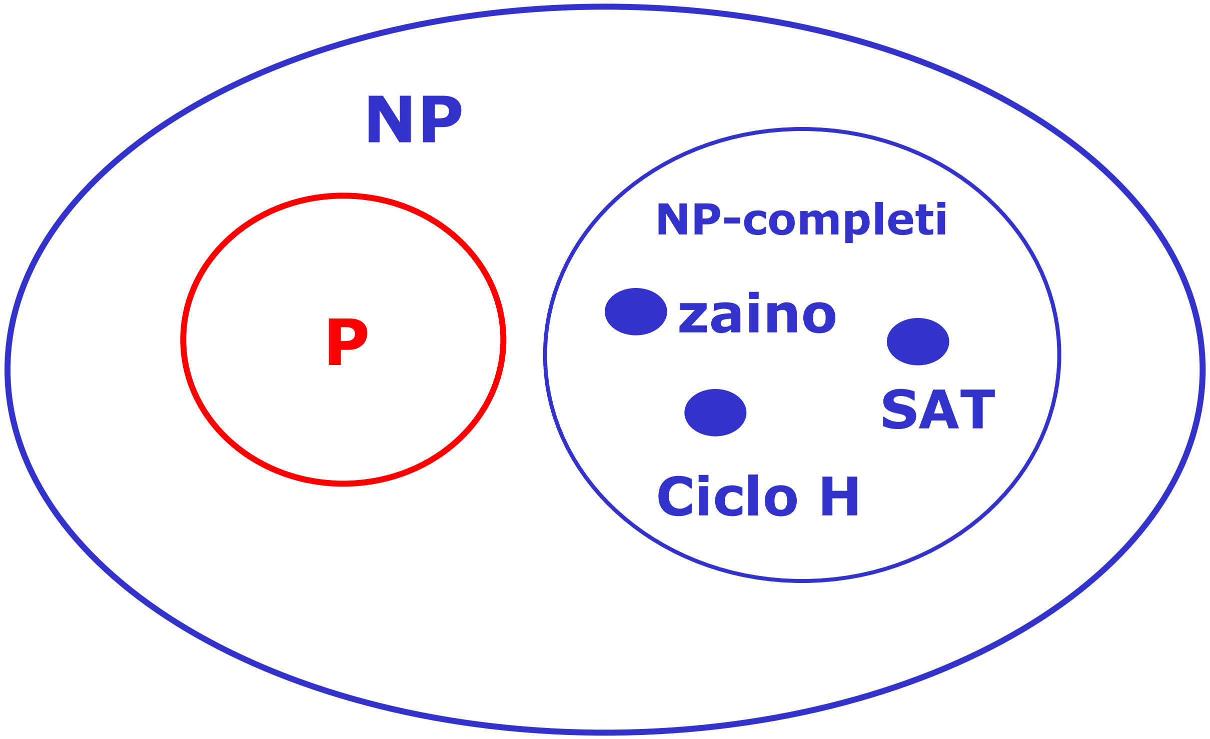
Problemi NP-completi

E' stato dimostrato che i seguenti problemi decisionali sono NP-completi:

- **Commesso viaggiatore**
- **Zaino**
- **Ciclo hamiltoniano**

Moltissimi altri problemi sono stati dimostrati NP-completi

Problemi NP-completi



Problemi NP-completi

Per dimostrare che un problema R è NP-completo:

dimostrare che R appartiene ad NP

individuare un algoritmo polinomiale nondeterministico per risolvere R (oppure dimostrare che la verifica di una soluzione di R può essere fatta in tempo polinomiale)

dimostrare che esiste un problema NP-completo che si riduce a R

se ne sceglie uno fra i problemi NP-completi noti che sia facilmente riducibile a R

Utilizzo

Perché serve dimostrare che un problema è NP-completo?

Perché non riusciamo a risolverlo con un algoritmo polinomiale e vogliamo dimostrare che non ci si riesce a meno che P non sia uguale ad NP, problema tuttora non risolto

Utilizzo

I can't find an efficient algorithm, but neither can all these famous people.

Il problema della Fattorizzazione di un numero

FATT (Fattorizzazione): Scomposizione di un numero in fattori primi

Es: $150 = 2 \times 3 \times 5^2$

Come in tutti problemi di teoria dei numeri, la complessità si calcola in funzione del numero di cifre del numero da fattorizzare (dimensione dell'input)

Il problema della Fattorizzazione

La moltiplicazione è $O(n^{\log_2 3})$ o O(n^2)

1.634.733.645.809.253.848.443.133.883.865.090.859.841.783.670.033.0
92.312.181.110.842.389.333.100.104.508.151.212.118.167.511.579 **x**

1.900.871.281.664.822.113.126.851.573.935.413.975.471.896.789.968.5
15.493.666.638.539.088.027.103.802.104.498.957.191.261.465.571
=

3.107.418.240.490.043.721.350.750.035.888.567.930.037.346.022.842.7
27.545.720.161.948.823.206.440.518.081.504.556.346.829.671.723.286.
782.437.916.272.838.033.415.471.073.108.501.919.548.529.007.337.724
.822.783.525.742.386.454.014.691.736.602.477.652.346.609

Richiede meno di un secondo di tempo di calcolo!

Il problema della Fattorizzazione

L’inverso della moltiplicazione è difficile

Trovare A e **B tali che**
A * B = 3.107.418.240.490.043.721.350.750.035.888.567.930.037.
346.022.842.727.545.720.161.948.823.206.440.518.081.504.556.346.
829.671.723.286.782.437.916.272.838.033.415.471.073.108.501.919.
548.529.007.337.724.822.783.525.742.386.454.014.691.736.602.477.
652.346.609

A=1.634.733.645.809.253.848.443.133.883.865.090.859.841.783.670
.033.092.312.181.110.842.389.333.100.104.508.151.212.118.167.511
.579
B= 1.900.871.281.664.822.113.126.851.573.935.413.975.471.
896.789.968.515.493.666.638.539.088.027.103.802.104.498.957.191.
261.465.571

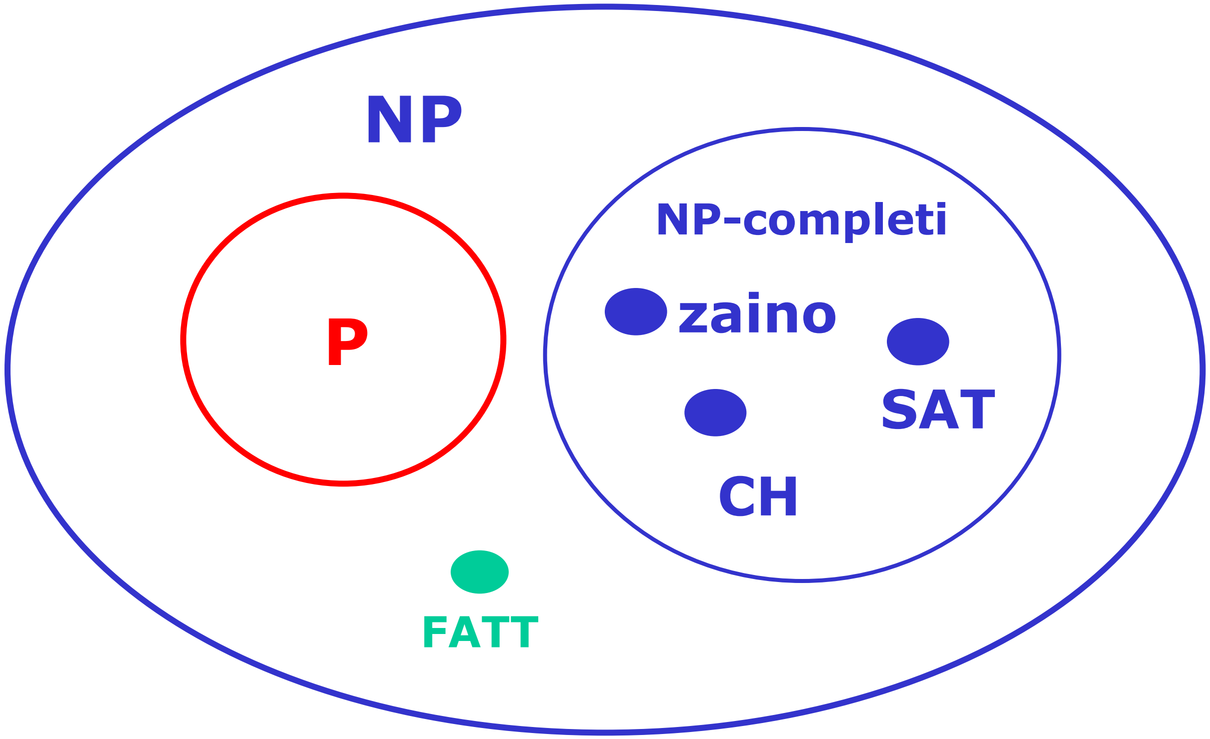
Oggi richiede più di una decina di anni di tempo di calcolo!

Il problema della Fattorizzazione

- **Per ora si conoscono soltanto algoritmi esponenziali per FATT**
- **FATT è in NP: la verifica è polinomiale (moltiplicazione)**
- **E' questione aperta se FATT sia in P, ma quasi sicuramente non è NP-completo**
- **Praticamente impossibile scomporre un numero di 200 o più cifre con gli algoritmi attuali**

Il problema della Fattorizzazione

- **Sulla difficoltà di FATT si basano i meccanismi della crittografia a chiave pubblica (operazioni facili con inversa difficile) che si basano su numeri primi molto grandi**
- **Un algoritmo polinomiale per FATT non dimostrerebbe P=NP, ma metterebbe in forte crisi i meccanismi della crittografia**
- **algoritmo polinomiale di Shor (1994) basato su Quantum Computing**


NP
NP-completi
P
zaino
SAT
CH
FATT

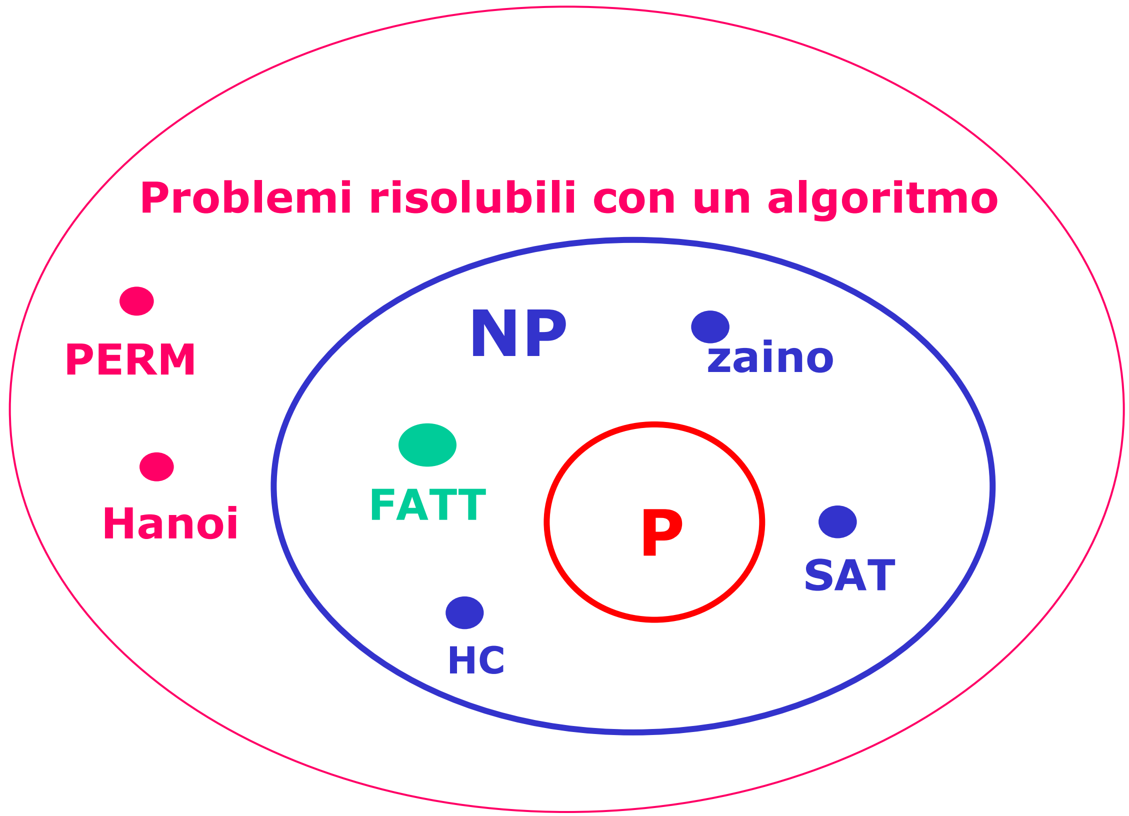
metodologie per affrontare i problemi difficili

- **Algoritmi di approssimazione**
- **Algoritmi probabilistici**
- **Reti neurali**
- **Quantum Computing**

Problemi non in NP

PERM: Trovare tutte le permutazioni di un insieme (n!)

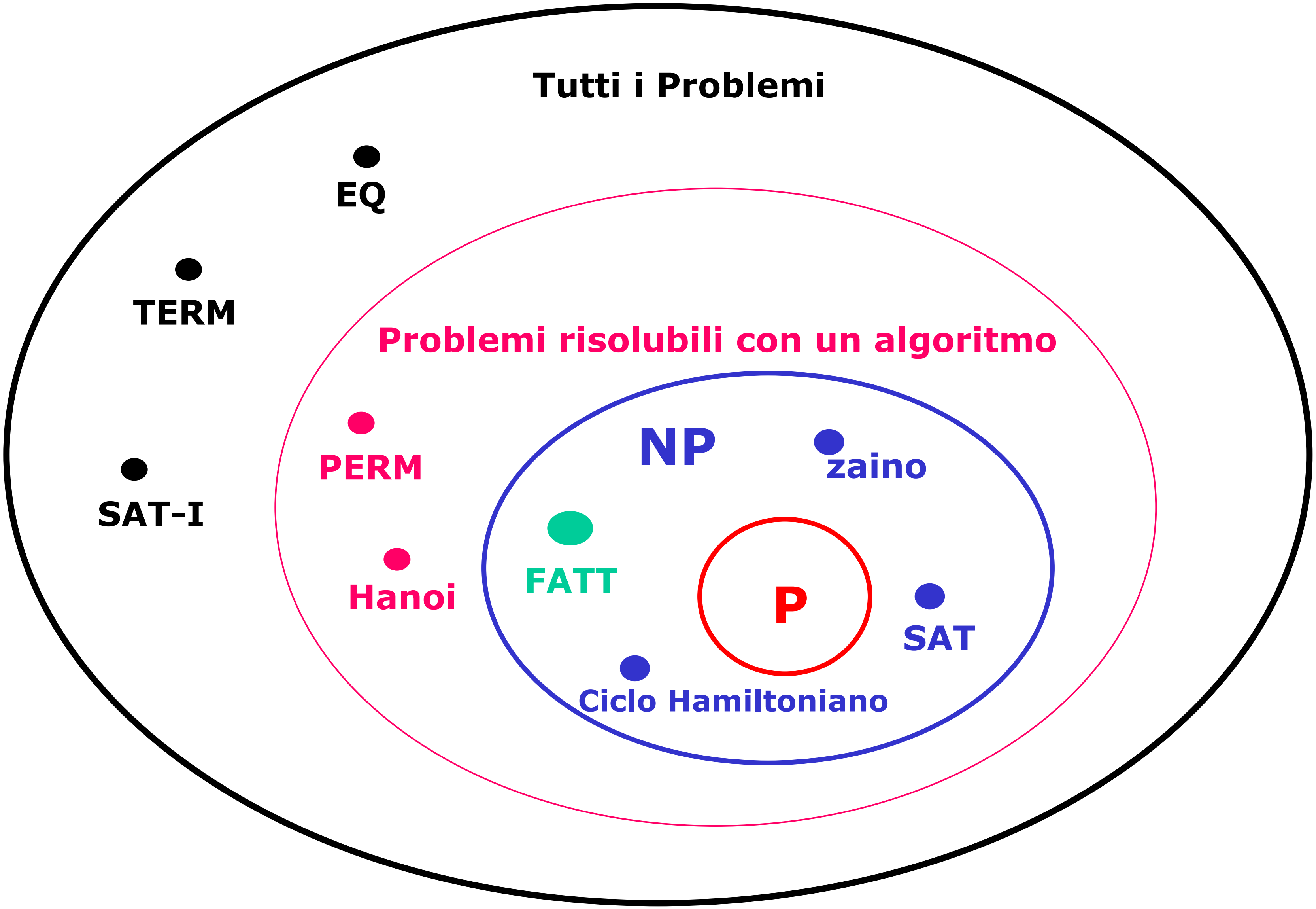
Torre di Hanoi



Problemi non risolubili con un algoritmo

- **TERM: Decidere la terminazione di un programma su un input**
- **SAT-I: soddisfattibiità di una formula nella logica del I ordine**
- **EQ: Decidere l'equivalenza di due programmi**

Problemi


Tutti i Problemi
EQ
TERM
SAT-I
Problemi risolubili con un algoritmo
PERM
Hanoi
NP
zaino
FATT
P
SAT
Ciclo Hamiltoniano

Riferimenti Bibliografici

Demetrescu:

Capitolo 16

Cormen:

Capitolo 34