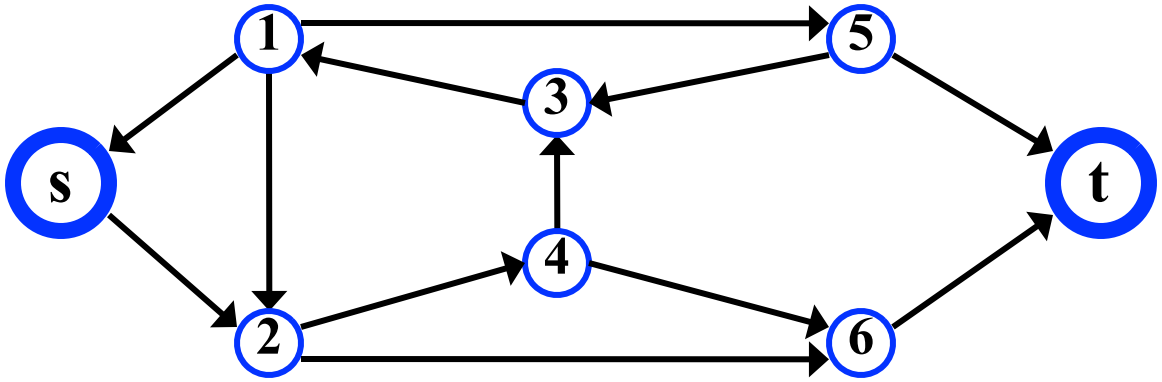
Sommario

- **Caratterizzazione alternativa di NP: il verificatore polinomiale**
- **la classe coNP e le relazioni tra P, NP,coNP e EXPTIME**

HAMPATH - 1

HAMPATH = {<G,s,t> | G è un grafo diretto con un cammino hamiltoniano da s a t (cioè un cammino che attraversa tutti i nodi di G una e una sola volta)}

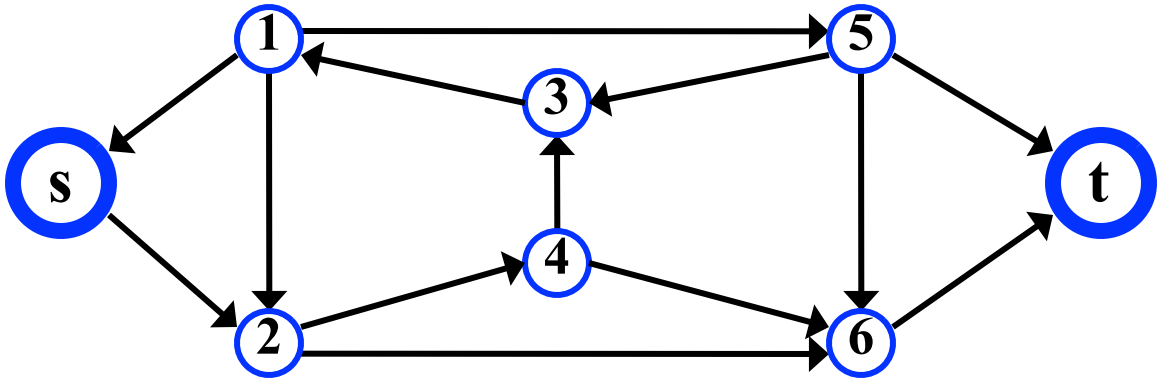
Istanza 1



$\in$ HAMPATH?

NO

Istanza 2



$\in$ HAMPATH?

SI

HAMPATH - 2

HAMPATH = {<G,s,t> | G è un grafo diretto con un cammino hamiltoniano da s a t (cioè un cammino che attraversa tutti i nodi di G una e una sola volta)}

Osserviamo che se p= $v_1,...,v_n$ è una sequenza di vertici in un grafo diretto G si può controllare che si tratta di un cammino hamiltoniano da s a t verificando che:

1. **v_1= s, v_n = t, che siano tutti diversi e in numero pari al numero di vertici di G**
2. **(v_i,v_{i+1}) è un arco in G, per i = 1,...,n-1**

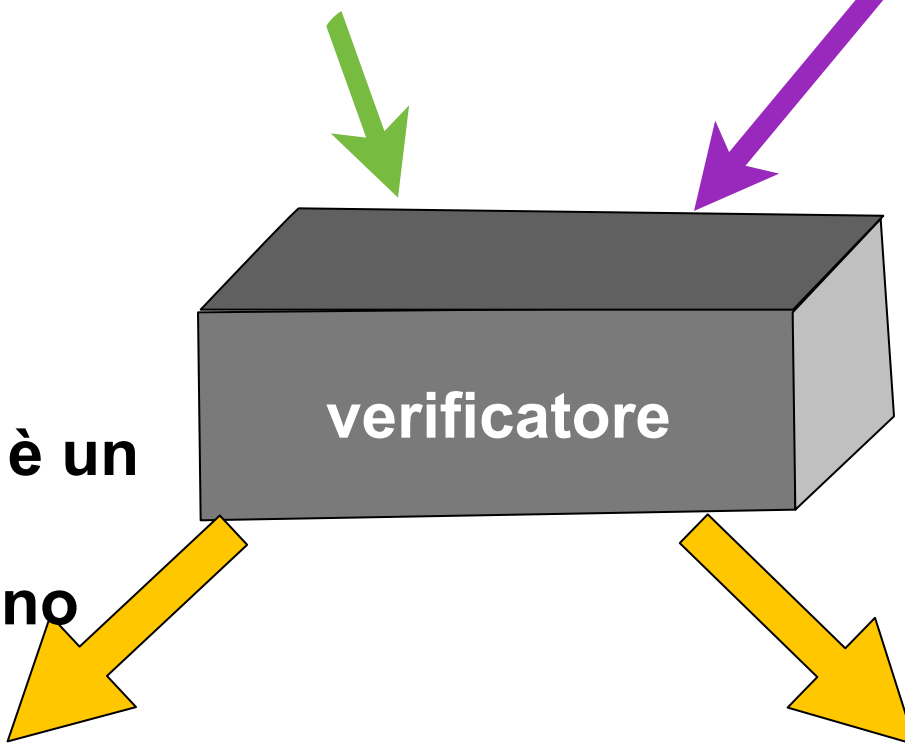
Questi controlli sono semplici computazionalmente, cioè eseguibili in tempo polinomiale

L’esempio del ciclo hamiltoniano

HAMPATH = {<G,s,t> | G è un grafo diretto con un cammino hamiltoniano da s a t (cioè un cammino che attraversa tutti i nodi di G una e una sola volta)}

input del problema: <G,s,t>

certificato: p= $v_1,\ldots,v_n$



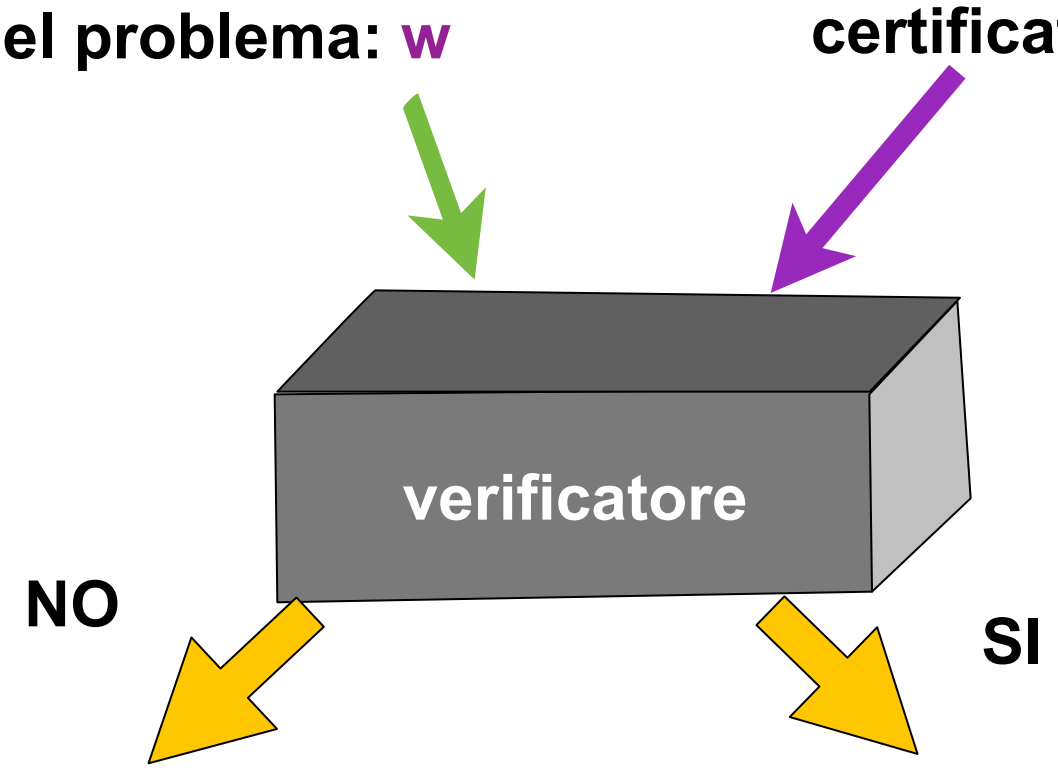
NO: p non è un cammino hamiltoniano

SI: p è un cammino hamiltoniano

Il verificatore

Un verificatore per A ⊆ Σ* è un algoritmo V (una TM) tale che A = { w | w ∈ Σ* e V accetta <w,c> per qualche stringa c, il certificato}

Un verificatore V è detto polinomiale se accetta una stringa w in tempo polinomiale nella lunghezza di w.



Il verificatore

Un linguaggio A è polinomialmente verificabile se ha un verificatore polinomiale.

Nota: il certificato c deve essere di lunghezza polinomiale nella lunghezza di w perchè il verificatore sia polinomiale.

Un verificatore per HAMPATH

Un verificatore per

HAMPATH = {<G,s,t> | G è un grafo diretto con un cammino hamiltoniano da s a t (cioè un cammino che attraversa tutti i nodi di G una e una sola volta)}

è un algoritmo V_H tale che

HAMPATH = {<G,s,t> | V_H accetta <<G,s,t>,<v_1,...,v_n>>, dove G è grafo, s, t due suoi vertici e v_1,…,v_n è una permutazione dei vertici di G}

Quindi V_H è un algoritmo che effettua i controlli che abbiamo visto per decidere se v_1,...,v_n è un cammino hamiltoniano o no. Il verificatore V_H è polinomiale.

Non sappiamo se c'è un algoritmo polinomiale che decide HAMPATH

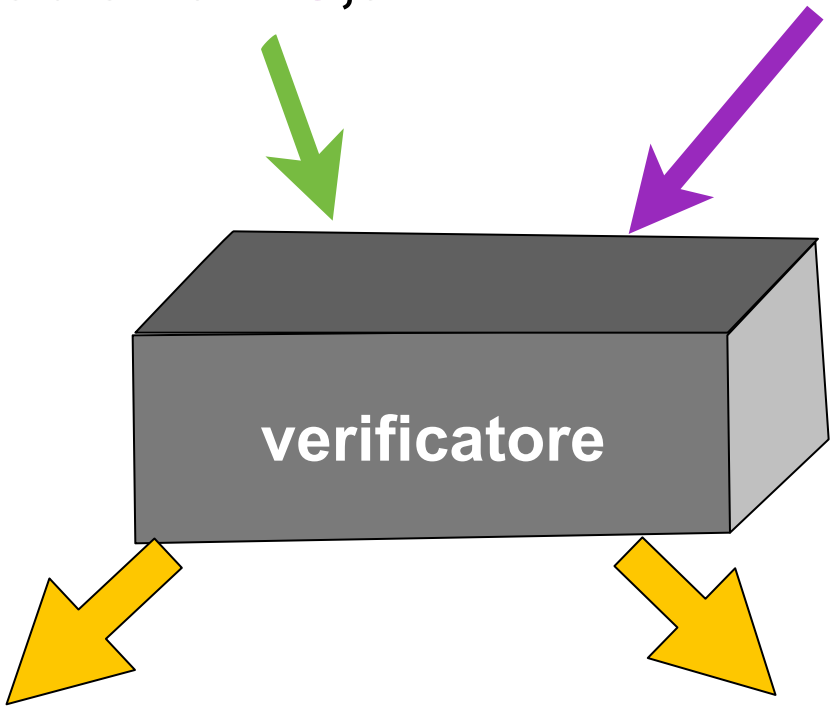
Un verificatore per SUBSET-SUM

SUBSET-SUM = { <S,t> | S è un insieme di interi dotato di un sottoinsieme a somma t}

Es.: S={10,-3,5,-1,-8,4}, t=0 è in SUBSET-SUM perchè Y ={-3,-1,4} è un sottoinsieme a somma t.

input del problema: <S,t>

certificato: <Y>



Il verificatore deve controllare che Y sia un sottoinsieme di S la cui somma è t.

Un verificatore per SUBSET-SUM

Un verificatore per SUBSET-SUM è un algoritmo V tale che

SUBSET-SUM = { <S,t> | V accetta <<S,t>,Y >, dove S è un insieme di interi, t un intero positivo e Y un sottoinsieme di S}

V deve controllare che Y sia un sottoinsieme di S la cui somma è t.

Questo controllo si può fare in tempo polinomiale, mentre non sappiamo se c'è un algoritmo polinomiale per decidere SUBSET-SUM.

NP e verificatori polinomiali

Teorema. Un linguaggio L è deciso da una NMT in tempo polinomiale $\Leftrightarrow$ L ha un verificatore polinomiale.

Così potremo dire che NP è l'insieme dei problemi verificabili in tempo polinomiale, cioè dei problemi per i quali si può decidere in tempo polinomiale se un dato elemento è una soluzione del problema.

Scompare dalla definizione ogni riferimento al nondeterminismo!

NP e verificatori polinomiali

Prova:

1. Un linguaggio L è deciso da una NMT in tempo polinomiale $\Rightarrow$ L ha un verificatore polinomiale.

Sia L un linguaggio per il quale esiste una NTM T che lo accetta in tempo polinomiale, n^k. Sia r il massimo grado di non determinismo in T e sia c una stringa su {1,...,r} di lunghezza n^k.

Considera il seguente verificatore:

V_T= input <w,c>

- **esegui T su w seguendo il cammino computazionale descritto da c**
- **se il cammino è di accettazione, accetta altrimenti rifiuta**

Un linguaggio L è deciso da una NMT in tempo polinomiale $\Rightarrow$ L ha un verificatore polinomiale.

La costruzione del verificatore è corretta perchè

$$L(T) = \{w \mid V_T \text{ accetta } \langle w, c \rangle \text{ per qualche stringa } c\}$$

infatti per ogni parola w accettata dalla NTM T esiste un cammino di accettazione per w, dunque esiste un certificato c per cui V_T accetta. Mentre se una parola è rifiutata da T ogni certificato porta al rifiuto in V_T. Inoltre il certificato è polinomiale nella dimensione dell'input perchè la NTM ha complessità di tempo polinomiale.

NP e verificatori polinomiali

2. Un linguaggio L ha un verificatore polinomiale $\Rightarrow$

L è deciso da una NMT in tempo polinomiale

Se V è un verificatore polinomiale per $L \subseteq \Sigma^*$, e quindi L = { w | w è in Σ^* e V accetta <w,c> per qualche stringa c} possiamo considerare V come una TM di complessità n^k, dove |w|=n.

NTM T_V per L :

T_V = input w

- **scegli non deterministicamente una stringa c di lunghezza n^k**
- **esegui V sull'input <w,c>**
- **se V accetta, accetta e se V rifiuta, rifiuta**

Un linguaggio L ha un verificatore polinomiale $\Rightarrow$ L è deciso da una NMT in tempo polinomiale

Si tratta di fa vedere che

$$L = \{ w \mid V \text{ accetta } \langle w,c \rangle \text{ per qualche stringa } c\}$$

è accettato dalla NTM T_V.

Per ogni parola w tale che <w,c> è accettata da V esiste un cammino di accettazione di T_V per w, visto che li "tenta" tutti. D'altro canto se una parola non ha un certificato di accettazione in V, tutti i cammini di T_V su w saranno di rifiuto e quindi la parola non è accettata.

NP

In conclusione NP si può definire anche come la classe dei problemi che ammettono un verificatore polinomiale

P vs NP

Uno dei problemi fondamentali
dell'Informatica teorica è capire se

$$P \subset NP \text{ o } P = NP$$

Se $P = NP$, allora

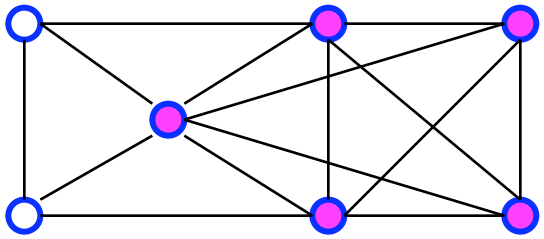
- **ogni problema verificabile in tempo polinomiale sarebbe anche risolvibile in tempo polinomiale**
- **avremmo algoritmi efficienti per TSP, FACTOR.**
- **schemi crittografici come RSA sarebbero inutili, con conseguenze drammatiche per il sistema bancario**

Problemi in NP - 1

Dato un grafo non diretto un clique del grafo è un sottografo indotto completo.

Un k-clique è un clique di k nodi

CLIQUE = {<G,k> | G è un grafo non diretto con un k-clique} è un linguaggio in NP



Prova 1

CLIQUE è in NP.

Dato un grafo G=(V,E) un certificato per il k-clique è semplicemente un sottoinsieme c di k vertici di G.

Verificatore per CLIQUE:

Input <<G,k>,c>

1. **verifica che c sia un sottoinsieme di k elementi di V**
2. **verifica se tutti i vertici in c sono connessi tra loro.**
3. **se entrambi i test hanno avuto successo, accetta altrimenti rifiuta.**

Prova 2

CLIQUE è in NP.

Dato un grafo G=(V,E) un certificato per il k-clique è semplicemente un sottoinsieme c di k vertici di G.

NTM per CLIQUE:

Input <G,k>

1. **nondeterministicamente genera un sottoinsieme c di k elementi di V**
2. **verifica se tutti i vertici in c sono connessi tra loro.**
3. **se sì, accetta altrimenti rifiuta.**

Problemi in NP - 2

SAT={ϕ | ϕ è una formula booleana soddisfacibile}

Es. (A $\vee$ ¬B) $\wedge$ (¬C $\vee$ ¬B) $\wedge$ (C $\vee$ A$\vee$ B)

Due tipi di prova

Verificatore per SAT:

Input <<φ>,c>

1. **verifica che c sia un assegnamento di valori di verità per tutte le variabili di φ**
2. **verifica se φ risulta vera e in tal caso accetta, altrimenti rifiuta.**

Per SAT abbiamo una NTM T_{SAT} di complessità di tempo polinomiale così definita:

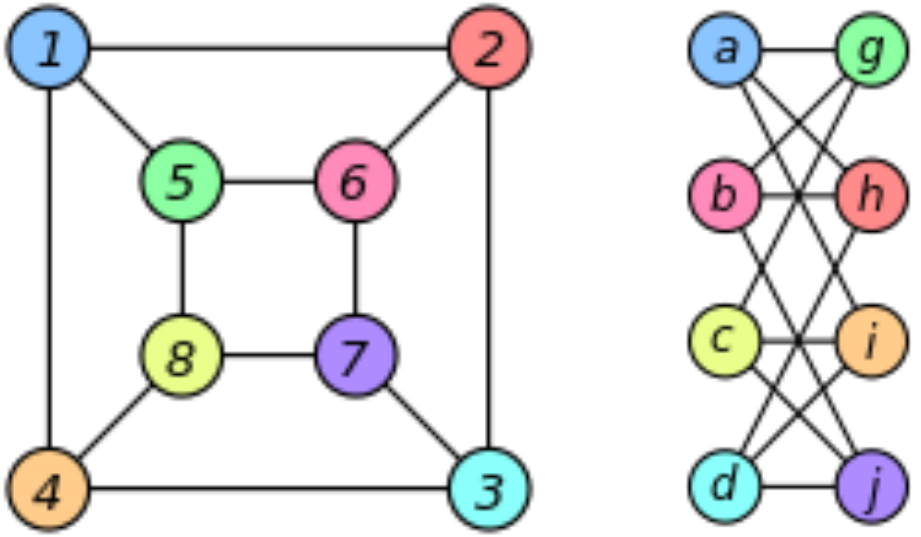
T_{SAT} = INPUT φ

1. **non deterministicamente genera un possibile assegnamento di valori 0 e 1 per le variabili di φ**
2. **verifica se φ vale 1, se sì accetta altrimenti rifiuta.**

Isomorfismo tra grafi

GI è il problema di determinare se due grafi G = (V,E) e G'= (V',E') sono isomorfi, cioè se esiste una funzione biettiva f : V →V' tale che se {u,v} è un arco di G, allora {f(u),f(v)} è un arco di G'. Si tratta di determinare se esiste una biezione tra i vertici che conservi l'adiacenza.

Informalmente è il problema di stabilire se due grafi disegnati diversamente sono lo stesso grafo.



Esempio preso da Wikipedia

I due grafi sono isomorfi, basta associare vertici dell'uno a quelli dell'altro con lo stesso colore.

Isomorfismo tra grafi

GI = {(G,G') | G e G' sono grafi non diretti e isomorfi}
è in NP

Infatti non è difficile immaginare un verificatore A che prende in input due grafi G=(V,E) e G'=(V',E') e un sottoinsieme U di n coppie in VxV', dove n è il numero dei vertici dei due grafi. Si tratta di verificare se i due grafi hanno lo stesso numero di archi, se le coppie in U sono tali da comprendere sulla prima componente tutti i vertici di V e sulla seconda tutti quelli di V' e se per ogni arco {u,v} in G c'è l'arco {u',v'} in G' se (u,u') e (v,v') sono in U.
Se è così il verificatore dà risposta positiva.
Questo lavoro di verifica è fatto in tempo polinomiale.

Isomorfismo tra grafi: algoritmi

L'algoritmo migliore, risultato recentissimo e non ancora pubblicato di Làszlò Babai (premio Knuth 2015) è di complessità quasi polinomiale, cioè in $2^{O(\lg^c n)}$

L'algoritmo migliore precedente, sempre di Babai, con Luks, è stato presentato al FOCS (IEEE Symposium on Foundations of Computer Science) nel 1983 e ha complessità in $2^{O(\sqrt{n} \lg n)}$

Per le applicazioni c'è un'euristica che lavora molto bene i cui autori sono Brendan McKay e Adolfo Piperno, che lo descrivono in un articolo apparso sul Journal of Symbolic Computation, nel 2014.
Altre informazioni: http://pallini.di.uniroma1.it

coNP

La classe coNP contiene i complementi dei linguaggi in NP.

Esempi di problemi in coNP:

UnSAT = {ϕ | ϕ è falsa per ogni assegnamento alle variabili }

NoCLIQUE = {<G,k> | G è un grafo non diretto senza un k-clique}

coNP: altro esempio

Val = {ϕ | ϕ è vera per ogni assegnamento alle variabili }

Val è in coNP?

sì, perché unVAL è in NP (prova analoga a quella per SAT!)

Verificatore per unVAL:

Input <<φ>,c>

1. **verifica che c sia un assegnamento di valori di verità per tutte le variabili di φ**
2. **verifica se φ risulta falsa e in tal caso accetta, altrimenti rifiuta.**

PRIMES and COMPOSITES

E' facile provare che COMPOSITES è in NP.
COMPOSITES={x | x=p*q, per p,q interi e p,q≠1}
Basta dare in input a un verificatore gli interi x e p come istanza del problema e certificato.

not COMPOSITES =
PRIMES = {x | x è un numero primo}.
Quindi PRIMES è in coNP

La prova che PRIMES è in NP è meno semplice (teorema di PRATT, 1975)
Quindi PRIMES NP ∩ coNP .

Primes in P

PRIMES: Dato un intero positivo n, n è primo?

Agrawal-Kayal-Saxena hanno dimostrato nel 2002 che PRIMES si risolve con un algoritmo polinomiale. Più precisamente in $\tilde{O}(\log^{12} n)$, nel 2005 abbassato a $\tilde{O}(\log^{6} n)$ da Carl Pomerance and H. W. Lenstra, Jr..

$f(n) = \tilde{O}(g(n))$ è un'abbreviazione per $f(n) = O(g(n) \log^{k} g(n))$ per qualche k

Factor in NP co-NP

FACTORIZE: Dato un intero x trova la sua fattorizzazione

FACTOR: Dati due interi x e U, x ha un fattore non banale minore di U?

FACTOR è in NP∩co-NP.

Un certificato per FACTOR e notFACTOR è dato da una sequenza di fattori primi.
Il verificatore controlla se si tratta di primi, in tempo polinomiale, e poi se si tratta della fattorizzazione di x.
Ora il verificatore per FACTOR risponde sì se uno dei fattori è minore di U, mentre quello per notFactor risponde sì se sono tutti maggiori di U.

P, NP, coNP e EXPTIME

$$P \subseteq NP \subseteq EXPTIME$$

dove

$$EXPTIME = \cup TIME(2^{n^k})$$

e

$$P \subseteq coNP \subseteq EXPTIME$$

Prova $P \subseteq coNP$. Infatti P=coP (l'insieme dei complementi dei linguaggi in P) e $P \subseteq NP \Rightarrow coP \subseteq coNP$, infatti se $P \subseteq NP$ allora $L \in P \Rightarrow L \in NP \Rightarrow \neg L \in coNP$, quindi $P = coP \subseteq coNP$!

P, NP e coNP

$P \subseteq NP \cap coNP$ ma $P = NP \cap coNP$?

Non noto (si suppone di no)

Per i problemi in $NP \cap coNP$ disponiamo di un verificatore polinomiale sia per le istanze sì che per le istanze no.
Quindi sono problemi per i quali può essere ragionevole aspettarsi l'appartenenza a P.
Tra questi c'è anche Factor.

P, NP e coNP : i possibili scenari