Aniello Murano

Classe dei problemi NP

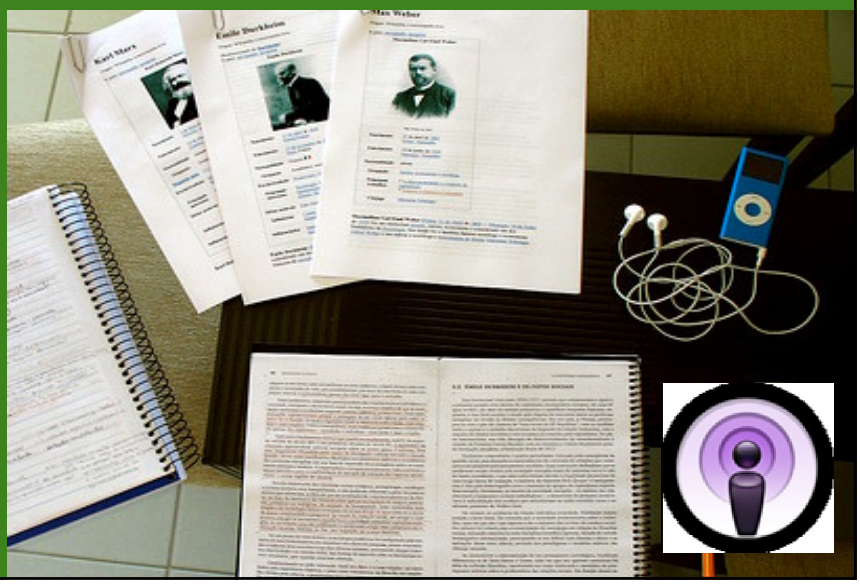
Lezione n.13
Parole chiave:
Classe NP

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Introduzione alla lezione

- Nella lezione precedente abbiamo visto alcuni problemi che ammettono soluzione polinomiale
- Tuttavia, esistono problemi per cui si sa che non è possibile risolverli in tempo polinomiale
- Esistono poi degli altri per i quali, pur conoscendo una soluzione esponenziale, non si sa se esiste o meno una soluzione polinomiale.
- All'interno di quest'ultima classe, esiste una classe di problemi legati dalla seguente proprietà computazionale:

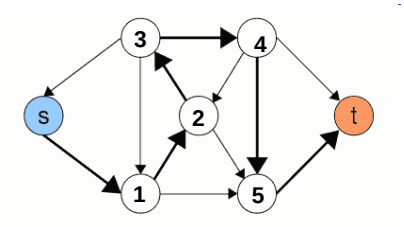
  "la soluzione polinomiale di uno solo di essi
  porta alla soluzione polinomiale di tutti"

- Un problema di questa classe è verificare l'esistenza di un percorso Hamiltoniano in un grafo orientato

Il problema del grafo hamiltoniano (1)

- Dato un grafo diretto G, un percorso in G è detto **hamiltoniano** se esso attraversa tutti i nodi di G esattamente una volta.
- Consideriamo il caso particolare di trovare un percorso hamiltoniano da un nodo s a un nodo t in G, per s e t dati.
- Formalmente, definiamo con **HAMPATH** il linguaggio che vogliamo testare nel modo seguente:

HAMPATH = {(G, s, t) | *G è un grafo diretto con un percorso hamiltoniano da s a t}*



- Si consideri il grafo G in figura con i nodi s e t. (G, s, t) appartiene a *HAMPATH visto che il percorso s,1,2,3,4,5,t è un percorso hamiltoniano*

Il problema del grafo hamiltoniano(2)

- Una soluzione immediata per HAMPATH è data considerando tutti i possibili percorsi in G da s a t e poi verificando che essi includano tutti i nodi di G.
- Questa soluzione è esponenziale in G
- Non è nota l'esistenza di un algoritmo polinomiale per questo problema.
- D'altro canto, questo problema ha la proprietà di essere verificabile in tempo polinomiale
- Quest'ultima proprietà è fondamentale per individuare la classe di complessità a cui appartiene questo problema

Il problema della composizionalità

- Un altro problema verificabile in tempo polinomiale è quello della composizionalità (o anche detto della primalità)
- Un numero naturale è **composto** se non è primo.
- Formalmente, l'insieme dei composti è definito come

$$COMP = \{x \mid x = pq, \text{ per } p, q > 1 \text{ e interi}\}.$$

- La verifica è semplice se si ha a disposizione un divisore del numero.
- Recentemente è stato dimostrato che è possibile decidere la composizionalità in tempo polinomiale.
- Trovare i divisori di un numero non primo è invece più difficile. Al momento, l'algoritmo più efficiente necessita di un tempo esponenziale nel numero di digit che compongono il numero.

Problemi non verificabili in tempo polinomiale

- Esistono tuttavia problemi che non sono neppure verificabili in tempo polinomiale.
- Questo è per esempio il caso per il complemento di *HAMPATH*.
- Anche se possiamo determinare (in qualche modo) che un grafo non ha un cammino hamiltoniano, non conosciamo un modo per verificare la sua non esistenza in tempo polinomiale.
- In particolare, possiamo usare l'algoritmo in tempo esponenziale utilizzato per determinare se esiste un cammino hamiltoniano (visto in precedenza).
- Nella prossima diapositiva, definiamo formalmente il concetto di **verificatore**

Definizione di verificatore

- Un **verificatore** per un linguaggio A è un algoritmo V, dove

  A = {*w* | V accetta *(w, c)* per qualche stringa *c}.*

- c è **chiamato** certificato di A.
- Un verificatore in **tempo polinomiale** gira in tempo polinomiale nella lunghezza dell'istanza w e del certificato c.
- Un linguaggio A è verificabile **polinomialmente** se ha un verificatore che impiega tempo polinomiale.
- **Nota:** Per un verificatore polinomiale il certificato ha lunghezza polinomiale rispetto alla lunghezza dell'istanza w.
- Per il problema HAMPATH, un certificato per una stringa (G,s,t) in HAMPAT è semplicemente il percorso hamiltoniano da s a t.
- Per il problema dei composti (comp), un certificato per il numero composto x è semplicemente uno dei suoi divisori.

Definizione di NP

- **Definizione: NP** è la classe dei linguaggi che ha un verificatore in tempo polinomiale.
  - È una classe importante in quanto molti problemi "fondamentali" appartengono a **NP**
  - **NP** è l'acronimo per "non deterministic Polynomial-time": NP è la classe dei linguaggi decisi in tempo polinomiale da una macchina di Turing non deterministica.

- Esempio di macchina non deterministica per Hampath:

  N1 = "Con input **<G,s,t>**, dove **G** è un grafo direzionato con i nodi **s** e **t**:

  1. Scrivi una lista di m membri $\mathbf{p_1,...,p_m}$, dove m è il numero di nodi di **G**. Ciascun numero nella lista è selezionato **non deterministicamente**.
  2. Verifica se ci sono ripetizioni nella lista, se ce ne sono allora *reject*.
  3. Verifica che $\mathbf{s=p_1}$ e $\mathbf{t=p_m}$. Altrimenti *reject*.
  4. Per ciascun elemento (nodo) i tra **1** e **m-1**, verifica che l'arco $\mathbf{(p_i, p_{i+1})}$ è in **G**. Se non lo è allora *reject*. Altrimenti, tutti i test hanno avuto successo quindi *accept*."

- Si noti come l'unico passo nondeterministico è in 1, mentre tutti i passi 2-4 possono essere eseguiti in tempo polinomiale.

Np in termini di macchina di Turing non deterministica

- **Teorema:** Un linguaggio è in **NP** se e solo è decidibile da una macchina di Turing non deterministica in tempo polinomiale.
- **Dimostrazione:** L'idea è di mostrare come convertire un verificatore in tempo polinomiale in una macchina di Turing non deterministica M, e viceversa. "→" M simula il verificatore indovinando il certificato. "←" Il verificatore simula M utilizzando il ramo accettante come certificato.

→ Assumiamo che **A**∈**NP**. Allora sia V un verificatore per A tale che **V**∈**TIME(**n^k**)** che, per la definizione di NP, deve esistere.
Costruiamo M come segue:
M := "Con input w di lunghezza n:
1. Non deterministicamente seleziona una stringa c di lunghezza al massimo n^k.
2. Esegui il verificatore V su input <w,c>.
3. V, se accetta, accept, altrimenti reject."

← Assumiamo che il linguaggio A sia decidibile da una macchina di Turing non deterministica M in tempo polinomiale.
Costruiamo il verificatore V come segue:
V := "su input <w,c>, dove w e c sono le stringhe:
1. Verifica se c (codifica) è un ramo di accettazione del calcolo di m su input w.
2. Se sì, accept, altrimenti reject. "

NTIME(t(n))

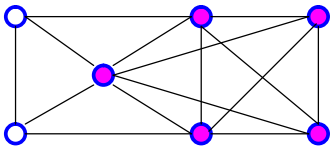
- Analogamente a quanto fatto per **P**, definiamo NTIME e **NP** nel modo seguente
- NTIME(t(n)) = **{L|** L è un linguaggio deciso in tempo O(t(n)) da una macchina di Turing **non**deterministica}.

$$\mathbf{NP} = U_k \, \mathrm{NTIME}(n^k).$$

- Come **P**, anche **NP** è insensibile alla scelta del modello computazionale sottostante, poiché tutti i modelli sono polinomialmente equivalenti.
- Così, nel descrivere e analizzare algoritmi non deterministici polinomiali, possiamo seguire le precedenti convenzioni per gli algoritmi in tempo polinomiale deterministici.

Esempi di problemi NP: CLIQUE (1)

- Una *clique* in un grafo non orientato è un sottografo in cui ogni coppia di nodi è collegata tramite un arco. Una *k-clique* è una clique che contiene k nodi con la proprietà precedentemente descritta.
- Il seguente esempio mostra una 5-Clique. Infatti, il sottografo rappresentato dai 5 nodi pieni formano una CLIQUE.



- Il problema della CLIQUE è definire se esiste in un grafo una k-clique.
- Definizione di CLIQUE:

CLIQUE = {<G,k> | G è un grafo non direzionato con una k-clique}

Esempi di problemi NP: CLIQUE (2)

CLIQUE = {<G,k> | G è un grafo non direzionato con una k-clique}

- **CLIQUE è un problema in NP, dimostriamolo:**
  - Idea: Una Clique è il certificato
  - Dimostrazione: Costruiamo un verificatore in tempo polinomiale V per Clique:

    V:= "su input <<G,k>, c>:
    1. Verifica se c è un insieme di nodi k in G.
    2. Verifica se G contiene tutti gli archi che connettono i nodi in c.
    3. Se entrambi i test hanno successo, accept, altrimenti reject. "

SubSetSum

- Di seguito definiamo il problema del Subset-Sum

SUBSET-SUM = {<S,t> | S è un insieme di interi e, per un sottoinsieme **R⊆S**, la somma di tutti gli elementi di **R** è uguale a **t}**

- Subset-sum prende in input un insieme di interi S e un intero t. Il problema consiste nel trovare un sottoinsieme R di S tale che la somma degli elementi di R sia proprio uguale a t.
- Per esempio, si consideri S= {1,3,5,9}.
  - È facile vedere che <S,4> appartiene a SUBSET-SUM visto che per R={1,3} ⊆ S si ha che 1+3=4.
  - Viceversa, <S, 7> non appartiene a SUBSET-SUM visto che non esiste un sottoinsieme di S tale per cui la somma dei suoi elementi sia uguale a 7.

SubSetSum è in NP

SUBSET-SUM = {<S,t> | S è un insieme di interi e, per un sottoinsieme **R⊆S**, la somma di tutti gli elementi di **R** è uguale a **t}**

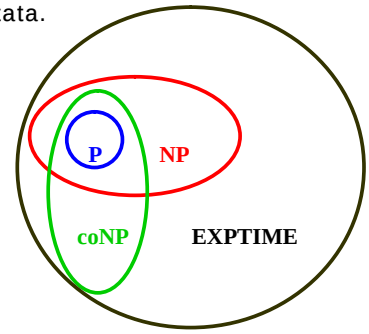
- **Dimostriamo che SubSet-Sum appartiene ad NP:**
  - **Idea:** Il sottoinsieme è un certificato
  - Dim. Costruiamo un verificatore V in tempo polinomiale per SUBSET-SUM:
    V := "su input <<S,t>, c>:
    1. Verifica se c è un insieme di numeri la cui somma è t.
    2. Verifica se S contiene tutti i numeri in c.
    3. Se entrambi i test hanno successo, accept, altrimenti reject. "
- Questo algoritmo prende tempo $O(n2^n)$, dato che esistono 2^n sottoinsiemi e per ognuno di essi dobbiamo fare n somme.

P vs Np vs CoNp vs Exptime

- **Definizione di CoNp**: **{L | L** è il complemento di un linguaggio in **NP}**.
- **Definizione di ExpTime**: **TIME(2^{n^1}) ∪ TIME(2^{n^2}) ∪ TIME(2^{n^3}) ∪ ...**

- **Considerazioni:**
  - **P**: l'appartenenza può essere facilmente decisa.
  - **NP**: l'appartenenza può essere facilmente verificata.
  - **coNP**: l'appartenenza può essere facilmente refutata.
  - **EXPTIME**: nessuna delle precedenti.

- La figura a lato mostra la situazione più verosimile (con alcune congetture)
- Al momento non si conosce se P ≠ NP
- Molti congetturano che lo sia



NP ∪ coNP ≠ EXPTIME