Lezione18 – la classe NP

Lezione del 16/05/2023

Problemi e classi di complessità

- Siamo pronti ad affrontare la questione della collocazione di problemi effettivi nelle corrette classi di complessità
  - ossia, ci troviamo di fonte ad un problema - un problema vero, del quale abbiamo bisogno di trovare la soluzione
    - ossia, di trovare l'algoritmo che decide quali delle sue istanze sono istanze sì
  - e vogliamo capire a quale classe di complessità appartiene quel problema
- La dispensa 8 si occupa della classe P: come dimostrare che un problema decisionale appartiene a P
- Non affrontiamo insieme la dispensa 8
  - perché per dimostrare che un problema appartiene a P occorre trovare un algoritmo che lo risolve e dimostrare che richiede tempo polinomiale nella dimensione dell'input
  - e di questo di occupa il corso di Algoritmi
- Tuttavia, in essa viene mostrato come utilizzare "in positivo" la riducibilità polinomiale
  - si prende un problema decisionale su grafi – il problema 2COL
  - lo si riduce ad un problema che sappiamo essere in P
  - e, oplà, il gioco è fatto
- Se a qualcuno interessasse…

La classe NP

- Allo studio della classe NP è dedicata la dispensa 9
- In particolare, la dispensa si occupa di due questioni "strutturali"
  - studiare la struttura dei problemi che popolano la classe NP
  - studiare la struttura della classe NP
- Prima di addentrarci in queste questioni strutturali, però, cerchiamo di capire: perché la classe NP è così importante?
- Tanto importante che qualcuno (piuttosto insigne, peraltro) ha pensato di mettere una taglia da un milione di dollari sulla congettura $P \neq NP$… ?!
- La classe P, è chiaro: è importante perché, se collochiamo un problema in P, quel problema sappiamo risolverlo *per davvero*
- Ma la classe NP?!
- Cosa ce ne importa di sapere che un certo problema
  - per il quale, magari, non riusciamo a progettare un algoritmo polinomiale
- che ce ne importa di sapere che quel problema è deciso da una macchina *non deterministica* in tempo polinomiale?

L'importanza di NP

- Che ce ne importa di sapere che un certo problema è deciso da una macchina *non deterministica* in tempo polinomiale?
- Se l'importanza di NP non va individuata nel modello di calcolo sul quale è basata, allora non può che risiedere nei problemi che la popolano!
- In effetti nella classe NP si trovano tanti (ma tanti) problemi
  - acquistare i biglietti aerei per un giro di tutte le capitali dell'UE, spendendo in totale al massimo 10000 euro
  - suddividere un insieme di oggetti (ciascuno di peso diverso) sui due piatti di una bilancia in modo tale che alla fine la bilancia sia in equilibrio
  - piastrellare un pavimento con mattonelle di forme e dimensioni diverse in modo tale che non rimangano spazi scoperti
  - scegliere al più 10 rappresentanti degli studenti ai quali comunicare una direttiva in modo tale che ogni altro studente conosca almeno uno di quelli che sono stati scelti così da poter essere informato
  - ...e tanti (ma tanti) altri ...
- che hanno grande rilevanza pratica
- che *non si riesce* a risolvere mediante algoritmi (deterministici) polinomiali …
- … ma che *non si riesce* nemmeno a dimostrare che non possono essere risolti in tempo deterministico polinomiale

La struttura dei problemi in NP

- Dunque, in NP si trovano un sacco di problemi (decisionali) importanti.
- E un problema è in NP quando esiste una macchina non deterministica che **accetta** le sue istanze sì in tempo polinomiale
  - anche se poi sappiamo che se esiste una macchina che accetta le istanze sì in tempo polinomiale, allora esiste anche una macchina che decide le sue istanze in tempo polinomiale
- Ma allora perché continuiamo a dire che in NP si trovano i linguaggi **accettati** in tempo non deterministico polinomiale ? Perché continuiamo a non usare la parola "decisi"?
- Per comprenderlo, dobbiamo tornare ad una vecchia conoscenza: il Genio burlone e pasticcione che costituisce uno dei modelli di una computazione non deterministica
- Quando, durante una computazione non deterministica NT(x), la macchina si trova in un certo stato q e legge un certo simbolo s, e nell'insieme delle quintuple di NT esistono tante quintuple che iniziano con la coppia (q,s), quale quintupla esegue NT?!
- Corre dal Genio, e lo chiede a lui!

Multi-quintuple e Genio

- Quando, durante una computazione non deterministica NT(x), la macchina si trova in un certo stato q e legge un certo simbolo s, e nell'insieme delle quintuple di NT esistono tante quintuple che iniziano con la coppia (q,s),
  - ossia NT si trova alle prese con una ***multi-quintupla***
- quale quintupla della multi-quintupla esegue NT?!
- Corre dal genio, e lo chiede a lui!
- Cioè, non avendo criteri per scegliere quale quintupla eseguire, ricorre al Genio sperando che, con l'aiuto delle sue arti magiche, riesca a scegliere la ***quintupla giusta*** da eseguire
  - ossia, la quintupla che, ***nell'ipotesi che x sia una istanza sì del problema***, porti NT ad accettare
- L'intervento del genio possiamo modellarlo inventando una istruzione apposita in PascalMinimo:
- l'istruzione **scegli**
- Vediamola in azione, scrivendo il codice corrispondente ad una Macchina di Turing non deterministica NT ad un nastro

Multi-quintuple, Genio, e l'istruzione **scegli**

- **Input**: parola $x_1x_2 \ldots x_n$ memorizzata nell'array N, con $N[i] = x_i$ per $i = 1,\ldots,n$.
- **Costanti**: l'insieme delle quintuple $P=\{\langle q_{11},s_{11},s_{12},q_{12},m_1\rangle,\ldots,\langle q_{k1},s_{k1},s_{k2},q_{k2},m_k\rangle\}$ che descrivono NT.
- $q \leftarrow q_0$; $t \leftarrow 1$; primaCella $\leftarrow t$; ultimaCella $\leftarrow n$;

while $(q \neq q_A \wedge q \neq q_R)$ **do begin**

$\Psi \leftarrow \{\langle q_{i1}, s_{i1}, s_{i2}, q_{i2}, m_i \rangle \in P: q_{i1} = q \wedge s_{i1} = N[t]\}$; *Ψ è una multi-quintupla*

if $(\Psi \neq \phi)$ **then begin**

scegli $\langle q_{i1}, s_{i1}, s_{i2}, q_{i2}, m_i \rangle \in \Psi$; *il Genio sceglie quale quintupla eseguire*

$N[t] \leftarrow s_{i2}$; $q \leftarrow q_{i2}$; $t \leftarrow t+m_i$; *e la esegue*

end
if (t < primaCella) **then begin**
primaCella $\leftarrow t$; $N[t] \leftarrow P$;
end
if (t > ultimaCella) **then begin**
ultimaCella $\leftarrow t$; $N[t] \leftarrow P$;
end
end
Output: q.

Multi-quintuple, Genio, e l'istruzione **scegli**

- La simulazione di una macchina non deterministica mediante un algoritmo in PascalMinimo già l'avevamo vista…
- Perciò, non dovreste aver problemi a comprendere il codice
  - fino a quando la macchina NT non entra nello stato q_A o nello stato q_R
    - (e lo stato in cui si trova NT è memorizzato nella variabile q)
  - calcola l'insieme Ψ delle quintuple che può eseguire trovandosi nello stato q e leggendo N[t],
    - (dove t indica la posizione della testina sul nastro di NT)
  - se Ψ contiene almeno una quintupla, ne sceglie una da eseguire
  - e la esegue
    - (gestendo le porzioni iniziali e finali del nastro mediante primaCella e ultimaCella)
- In questo caso, invece di simulare tutte le computazioni deterministiche di NT(x)
- l'algoritmo si affida al Genio per scegliere, di volta in volta, le quintuple da eseguire
- Soltanto una osservazione: se, ad un certo istante Ψ non contiene quintuple e q non è q_A e non è q_R, l'algoritmo entra in loop!
  - ma questo accade solo se P non è totale (ricordate?)

Ma il Genio è pasticcione

- Cerchiamo, ora, di capire quali conseguenze comporta ricorrere al Genio
  - *del quale proprio non ci si può fidare!*
- Eseguiamo l'algoritmo al lucido 7 con input x - chiamiamo $\mathcal{A}$ l'algoritmo e $\mathcal{A}$(x) la sua esecuzione sull'input x
- $\mathcal{A}$(x) termina in q_A o in q_R
  - assumendo che P sia totale: e noi lo assumiamo!
- Se $\mathcal{A}$(x) termina in q_A allora possiamo essere certi che il Genio ci ha indicato le risposte corrette
  - perché il Genio ha trovato una sequenza di quintuple da eseguire che termina nello stato q_A
  - e quella sequenza costituisce una computazione accettante di NT(x)
  - e, dunque, ***esiste*** *una computazione accettante di NT(x)!*
- Se $\mathcal{A}$(x) termina in q_A , allora, possiamo essere certi che possiamo accettare

Ma il Genio è pasticcione

- Ma se $\mathcal{A}(x)$ termina in in q_R allora qualche dubbio ci viene…
  - perché il Genio ha trovato una sequenza di quintuple da eseguire che termina nello stato q_R
  - e quella sequenza costituisce una computazione di NT(x) che rigetta
  - e, dunque, ***esiste*** una computazione di NT(x) che rigetta
  - ma questo non dimostra che ***tutte*** le computazioni di NT(x) rigettano!
- Ecco, allora: non fidandoci noi del Genio
  - e noi non ci fidiamo!
- possiamo solo concludere che il Genio non ha trovato la sequenza di quintuple che porta NT nello stato di accettazione
- **Ma non possiamo sapere se non l'ha trovata**
  - **perché una sequenza di quintuple che induce NT ad accettare non esiste**
  - **o perché il Genio non è stato sufficientemente abile (o onesto) da trovarla!**
- Ecco perché continuiamo a parlare di linguaggi accettati, piuttosto che decisi

La struttura dei problemi in NP

- In effetti, questa questione del Genio "a mezzo servizio"
  - il cui utilizzo è fondamentale quando lavora su una istanza sì di un problema
  - ma che non è di alcuna utilità quando lavora su istanze no del problema
- gioca un ruolo fondamentale per comprendere la struttura dei problemi che popolano NP
- E, per comprendere questa struttura, facciamo un po' di esempi
  - esempi di problemi
  - ed esempi di **algoritmi non deterministici** che li risolvono
- E, siccome ci accingiamo a progettare algoritmi che decidono problemi anziché linguaggi
  - e abbiamo ben compreso il ruolo giocato dalle codifiche
- i nostri algoritmi li descriveremo "ad alto livello"
  - utilizzando il PascalMinimo (o sue variazioni a più alto livello) corredato dell'istruzione **scegli**
  - mettendo da parte, per il momento, le macchine di Turing
- Ma prima di farlo dobbiamo ancora chiarire una questioncina

Multi-quintuple, Genio, e l'istruzione **scegli**

- In effetti, è comodo poter disporre di un Genio - seppure "a mezzo servizio"
- Rimane, però, una questione fondamentale: ma quanto è potente questo Genio?
- Cioè, se dispongo di un Genio, perché non gli chiedo direttamente "l'istanza x è un'istanza sì del mio problema?" ?
- Innanzi tutto, perché delle risposte del Genio non mi fido:
  - se gli chiedo di indicarmi quale quintupla eseguire ad un certo punto della computazione, poi posso verificare che mi ha indicato una quintupla che posso eseguire davvero
  - se gli chiedo di dirmi se x è una istanza sì, poi come la verifico la risposta?
- Poi, soprattutto, **abbiamo introdotto il genio per modellare il non determinismo**
  - per questo gli chiediamo di scegliere quale quintupla eseguire a ciascun passo della computazione
  - **e il numero di quintuple fra le quali scegliere è il grado di non determinismo della macchina**
  - che è **COSTANTE**
- Perciò, va bene trasportare il Genio nel mondo degli algoritmi ad alto livello
- a patto, però, di proporgli sempre di operare le sue scelta fra un numero **COSTANTE** di opzioni!

Il problema 3SAT

- Torniamo ad un **Esempio** che abbiamo già incontrato (quasi l'**Esempio 9.1**):
- dati un insieme X di variabili booleane ed un predicato f, definito sulle variabili in X e contenente i soli operatori $\wedge$, $\vee$ e $\neg$, decidere se esiste una assegnazione a di valori in {vero, falso} alle variabili in X tale che f(a(X))=vero
- Consideriamo soltanto predicati f in forma 3-congiuntiva normale (3CNF), ossia,
  - f è la congiunzione di un certo numero di clausole: $\mathbf{f = c_1 \wedge c_2 \ldots \wedge c_m}$
  - e ciascuna c_j è la disgiunzione ($\vee$) di tre letterali (3CNF), ad esempio $x_1 \vee \neg x_2 \vee x_3$
  - (un letterale è una variabile o una variabile negata)
- Questo problema prende il nome di 3SAT, ed è così formalizzato:
  - $\mathfrak{I}_{\mathbf{3SAT}}$ = { ⟨ X, f ⟩ : X è un insieme di variabili booleane ∧ f e un predicato su X in 3CNF}
  - $\mathbf{S}_{\mathbf{3SAT}}$(X, f) = { a: X → {vero, falso} } (S è l'insieme delle assegnazioni di verità alle variabili in X)
  - $\boldsymbol{\pi}_{\mathbf{3SAT}}$ (X, f, $\mathbf{S}_{\mathbf{3SAT}}$(X,f))= ∃ a ∈ $\mathbf{S}_{\mathbf{3SAT}}$(X,f) : f(a(X)) = vero
- Descriviamo, ora, un possibile algoritmo non deterministico che accetta 3SAT

Il problema 3SAT

- **Input**: un insieme di variabili booleane $X = \{x_1, x_2, \dots, x_n\}$ e una funzione booleana f definita sulle variabili in X.

- $i \leftarrow 1$;

  while ($i \leq n$) **do begin**

  scegli $a(x_i)$ nell'insieme {vero,falso};

  $i \leftarrow i+1$;

  end

  $i \leftarrow 1$;

  while ($i \leq n$) **do begin**

  sostituisci in f ogni occorrenza di x_i con $a(x_i)$, e ogni occorrenza di $\neg x_i$ con $\neg a(x_i)$;

  $i \leftarrow i+1$;

  end

  if ($f(a(X))$ =vero) **then** $q \leftarrow q_A$;

  else $q \leftarrow q_R$;

  Output: q.

Il problema 3SAT

- L'algoritmo che abbiamo appena visto è logicamente suddiviso in due parti:
- la prima parte ha carattere prettamente non deterministico
  - serve a scegliere una assegnazione di verità a per le variabili in f
- la seconda parte ha carattere prettamente deterministico
  - serve a verificare deterministicamente che l'assegnazione scelta soddisfi effettivamente f
- Poiché il numero di possibilità fra le quali scegliere ad ogni passo è pari a 2, si tratta, effettivamente, di un algoritmo non deterministico
- Poiché l'algoritmo accetta se e solo se **esiste** una sequenza di scelte che soddisfa f, allora è un algoritmo che accetta 3SAT
- Per quanto riguarda la sua complessità, osserviamo che
  - il primo ciclo **while** (nella parte non deterministica) richiede tempo non deterministico lineare in n = |X|.
  - il secondo ciclo **while** (nella parte deterministica) richiede tempo deterministico in O(|X|·| f |) = O(3 n m) = O(n m).
- In conclusione, l'algoritmo accetta $\langle X,f \rangle \in \mathfrak{I}_{3SAT}$ in tempo O(|X| · |f|), e questo prova che 3SAT ∈ NP.

Il problema CLIQUE

- Siamo all'**Esempio 9.2**:
- Il problema CLIQUE consiste nel decidere, dati un grafo non orientato G=(V,E) ed un intero $k \in \mathbb{N}$, se G contiene un sottografo completo di almeno k nodi.
  - un grafo si dice completo se in esso esiste un arco per ogni coppia di nodi
- Formalmente, il problema è descritto dalla tripla
  - $\mathfrak{I}_{\mathbf{CLIQUE}} = \{ \langle G = (V,E), k \rangle : G \text{ è un grafo non orientato} \wedge k \in \mathbb{N} \}$
  - $\mathbf{S}_{\mathbf{CLIQUE}}(G = (V,E), k) = \{ V' \subseteq V \}$ (S è l'insieme dei sottoinsiemi di V)
  - $\boldsymbol{\pi}_{\mathbf{CLIQUE}}(G, k, \mathbf{S}_{\mathbf{CLIQUE}}(G, k)) = \exists V' \in \mathbf{S}(G, k) : (\forall u, v \in V' [(u,v) \in E]) \wedge |V'| \geq k$
    - ossia, comunque si scelgano due nodi in V', essi sono collegati da un arco
- Descriviamo, ora, un possibile algoritmo non deterministico che accetta CLIQUE

Il problema CLIQUE

- **Input**: un grafo non orientato G = (V,E), con V = {v_1, v_2, ... ,v_n} e un intero k ∈ ℕ
- V' ← ϕ ; i ← 1;

while (i ≤ n) **do begin**

  scegli se inserire v_i in V' oppure no ;

  i ← i+1;

end

trovata ← vero; i ← 1;

while (i ≤ n -1) **do begin**

  j ← i+1;

  while (j ≤ n ∧ $v_i \neq v_j$) **do begin**

    if (v_i ∈ V' ∧ v_j ∈ V' ∧ (v_i, v_j) ∉ E) **then** trovata ← falso;

    j ← j+1;

  end

  i ← i+1;

end

if (trovata =vero ∧ |V'| ≥ k) **then** q ← q_A; **else** q ← q_R;

Output: q.

Il problema CLIQUE

- L'algoritmo che abbiamo appena visto è logicamente suddiviso in due parti:
- la prima parte ha carattere prettamente non deterministico
  - serve a scegliere un sottoinsieme V' di V
- la seconda parte ha carattere prettamente deterministico
  - serve a verificare deterministicamente che il sottoinsieme scelto soddisfi effettivamente $\pi_{\mathbf{CLIQUE}}(G, k, \mathbf{S}_{\mathbf{CLIQUE}}(G, k))$
- Esattamente come per l'algoritmo che accetta 3SAT!
- Poiché il numero di possibilità fra le quali scegliere ad ogni passo è pari a 2, si tratta, effettivamente, di un algoritmo non deterministico
- Poiché l'algoritmo accetta se esiste una sequenza di scelte che soddisfa il predicato di CLIQUE $\pi_{\mathbf{CLIQUE}}(G, k, \mathbf{S}_{\mathbf{CLIQUE}}(G, k))$, allora è un algoritmo che accetta CLIQUE
- Per quanto riguarda la sua complessità, osserviamo che
  - il primo ciclo **while** (nella parte non deterministica) richiede tempo non deterministico lineare in $n = |V|$
  - il secondo ciclo **while** (nella parte deterministica) richiede tempo deterministico in $O(|V|^2(|V| + |E|))$.
- In conclusione, l'algoritmo accetta $\langle G,k \rangle \in \mathfrak{I}_{\mathbf{CLIQUE}}$ in tempo $O(|V|^2 |E|)$, e questo prova che CLIQUE $\in$ NP.

Facciamo attenzione!

- Siamo all'**Esempio 9.3**, che descrive il problema LONG PATH: dati un grafo non orientato G=(V,E), due suoi nodi a e b, ed un intero k ∈ ℕ, vogliamo decidere se G contiene un percorso fra a e b di **esattamente** k archi.
- Poiché un percorso è una sequenza ordinata di nodi, possiamo pensare di rappresentare un percorso fra a e b lungo k con un array p di k-1 elementi tali che (a, p[1]) ∈ E, (p[1], p[2]) ∈ E, … , (p[i], p[1+1]) ∈ E, … , (p[k-2], p[k-1]) ∈ E, (p[k-1], b) ∈ E
- Potremmo pensare ad un algoritmo non deterministico che decide se ⟨ G , a, b, k ⟩ è una istanza sì del problema che segue lo schema degli algoritmi che abbiamo già visto:
  - Fase 1 (non deterministica): sceglie i nodi p[1], p[2], … , p[k-1]
  - Fase 2: verifica se ⟨ a, p[1], p[2], … , p[k-1], b ⟩ è effettivamente un percorso in G
- Potremmo implementare la Fase 1 mediante il seguente ciclo
  - i ← 1;
    while (i ≤ k-1) **do begin**
    scegli un nodo in V come elemento p[i];
    i ← i+1;
    end

Facciamo attenzione!

- Potremmo implementare la Fase non deterministica mediante il seguente ciclo
  - i ← 1;
    while (i ≤ k-1) **do begin**
    scegli un nodo in V come elemento p[i];
    i ← i+1;
    end
- Siamo proprio sicuri?...
- Il numero di possibilità fra le quali scegliere ad ogni passo è pari a | V |
- Non è costante!
  - V è parte dell'input
- ***Non*** si tratta, perciò, di un algoritmo non deterministico!
- In effetti, la fase non deterministica che accetta LONG PATH è sensibilmente più complessa
  - e, se vi interessa, la trovate nella dispensa
- In conclusione, quando progettate un algoritmo non deterministico, fate attenzione al fatto che il numero delle opzioni fra le quali scegliere sia **costante**!

Ma non sarà allora che…

- Uhmm…
- I tre problemi in NP che abbiamo visto in questa lezione
  - 3SAT, CLIQUE, LONG PATH
- hanno qualcosa in comune: la struttura del predicato π
  - in tutti e tre i problemi π ha la forma seguente: esiste un elemento di S (ossia, una soluzione possibile) che soddisfa certe proprietà – che chiamiamo η
  - **$\pi(x, S(x))$ = esiste $y \in S(x)$ tale che $\eta(x,y)$**
- Non solo, ma anche gli algoritmi per la loro decisione che abbiamo analizzato seguivano tutti lo stesso schema: con input x,
  - Fase 1 (non deterministica): sceglie una soluzione possibile $y \in S(x)$
  - Fase 2 (deterministica): verifica se y soddisfa il predicato $\eta(x, y)$
- E non basta:
  - la Fase 1, ossia, sceglie una soluzione possibile y, richiede tempo polinomiale in $|x|$
  - la Fase 2, ossia, la verifica che x e y soddisfino il predicato η, richiede tempo polinomiale **in $|x|$**
- Quante coincidenze!

Ma non sarà allora che…

- Uhmm… Quante coincidenze!
- Beh, certamente, tutti i problemi decisionali tali che
  - $\pi(x, S(x))$ = esiste $y \in S(x)$ tale che $\eta(x,y)$
- possono essere risolti da un algoritmo non deterministico che opera in due fasi:
  - Fase 1 (non deterministica): sceglie una soluzione possibile $y \in S(x)$
  - Fase 2 (deterministica): verifica se y soddisfa il predicato $\eta(x, y)$
- E tale algoritmo richiede tempo (non deterministico) polinomiale se
  - la Fase 1, ossia, la scelta di una soluzione possibile y, richiede tempo polinomiale in $|x|$
  - la Fase 2, ossia, la verifica che y soddisfi il predicato η, richiede tempo polinomiale in $|x|$
- E quindi possiamo dire che ogni problema
  - il cui predicato ha la forma $\pi(x, S(x))$ = esiste $y \in S(x)$ tale che $\eta(x,y)$
  - in cui la scelta di un elemento y di S(x) richiede tempo polinomiale in $|x|$
  - in cui la verifica che y soddisfi il predicato η, richiede tempo polinomiale in $|x|$
- appartiene ad NP

Ma non sarà allora che…

- Uhmm… Quante coincidenze!
- E quindi possiamo dire che ogni problema
  - il cui predicato ha la forma $\pi(x, S(x))$ = esiste y ∈ S(x) tale che $\eta(x,y)$
  - in cui la scelta di un elemento y di S(x) richiede tempo non deterministico polinomiale in |x|
  - in cui la verifica che y soddisfi il predicato η, richiede tempo deterministico polinomiale in |x|
- appartiene ad NP

- Uhmmm… Chissà dove ci porta questa considerazione…