Le classi $\mathbb{P}$ e $\mathbb{NP}$

Salvatore Filippone
salvatore.filippone@uniroma2.it

Gerarchie di problemi

È naturale classificare i problemi che si possono presentare secondo alcune caratteristiche:

- Dato un problema, esiste un algoritmo che lo risolve?
- Dato un problema tale che esista almeno un algoritmo che lo risolve, che complessità ha un tale algoritmo? E qual è l'algoritmo migliore possibile per quel problema?

Gerarchie di problemi

È naturale classificare i problemi che si possono presentare secondo alcune caratteristiche:

- Dato un problema, esiste un algoritmo che lo risolve?
- Dato un problema tale che esista almeno un algoritmo che lo risolve, che complessità ha un tale algoritmo? E qual è l'algoritmo migliore possibile per quel problema?

Per discutere delle questioni di computabilità si dovrebbe fissare un modello di macchina astratta.
È interessante notare che tutti i modelli proposti fino ad oggi si sono dimostrati essere sostanzialmente equivalenti (tesi di *Church-Turing*)

Gerarchie di problemi

È noto che alcuni problemi sono particolarmente insidiosi, ad esempio

Problema dell'arresto

Dato un programma (espresso in un qualche linguaggio) ed i suoi dati di ingresso, la esecuzione del programma terminerà in un tempo finito?

Gerarchie di problemi

È noto che alcuni problemi sono particolarmente insidiosi, ad esempio

Problema dell'arresto

Dato un programma (espresso in un qualche linguaggio) ed i suoi dati di ingresso, la esecuzione del programma terminerà in un tempo finito?

Teorema dell'arresto (Turing)

Non può esistere nessun algoritmo che risolva il problema dell'arresto

Quindi, non tutti i problemi ammettono un algoritmo che li risolva.

N.B.: nel costruire approssimazioni dei problemi della matematica del continuo abbiamo spesso a che fare con degli *schemi computazionali* che non possono arrivare alla soluzione in un tempo finito (p.es. è impossibile calcolare le radici di un polinomio di grado 11 con un numero finito di operazioni), ma possono *convergere* in un tempo finito ad un intorno opportunamente piccolo della soluzione stessa.

Classi di Complessità

Supponiamo ora di avere problemi che ammettono degli algoritmi risolutivi. Molti di questi algoritmi appartengono alla classe

Complessità polinomiale (nel tempo): $\mathbb{P}$

La classe cui appartiene qualunque algoritmo che sia $O(p(n))$ dove p è un polinomio in n.

Quando esiste un algoritmo polinomiale per un certo problema, quasi sempre il migliore algoritmo per lo stesso problema ha una complessità caratterizzata da un polinomio di grado basso (p.es. 2, 3 o 4).

Classi di Complessità

Supponiamo ora di avere problemi che ammettono degli algoritmi risolutivi. Molti di questi algoritmi appartengono alla classe

Complessità polinomiale (nel tempo): $\mathbb{P}$

La classe cui appartiene qualunque algoritmo che sia $O(p(n))$ dove p è un polinomio in n.

Quando esiste un algoritmo polinomiale per un certo problema, quasi sempre il migliore algoritmo per lo stesso problema ha una complessità caratterizzata da un polinomio di grado basso (p.es. 2, 3 o 4).

Per molti problemi non si conosce alcun algoritmo polinomiale

Classi di Complessità

Supponiamo ora di avere problemi che ammettono degli algoritmi risolutivi. Molti di questi algoritmi appartengono alla classe

Complessità polinomiale (nel tempo): $\mathbb{P}$

La classe cui appartiene qualunque algoritmo che sia $O(p(n))$ dove p è un polinomio in n.

Quando esiste un algoritmo polinomiale per un certo problema, quasi sempre il migliore algoritmo per lo stesso problema ha una complessità caratterizzata da un polinomio di grado basso (p.es. 2, 3 o 4).

Per molti problemi non si conosce alcun algoritmo polinomiale

Ma nessuno ha dimostrato che un tale algoritmo non possa esistere.

Problemi in $\mathbb{P}$

Connettività: Stabilire se un grafo G è connesso;

Cammini in un grafo: Dato un grafo orientato $G = (V, E)$ e due sottoinsiemi dei vertici $S, T \subseteq V$, esiste un cammino da un vertice di S ad un vertice di T?

Matching: Dato un grafo G ed un intero k, esiste in G un matching di dimensione $\geq k$?

Spanning tree: Dato un grafo $G = (V, E)$, una funzione di costo $d(E)$ ed un numero L, esiste un albero che ricopra G con un costo $\leq L$?

Programmazione lineare Trovare la soluzione di

$$\begin{array}{rcl} \min & & cx \\ Ax & = & b \\ x & \geq & 0 \end{array}$$

La classe $\mathbb{NP}$

Cosa succede se non si trova un algoritmo polinomiale?

La classe $\mathbb{NP}$

È l'insieme dei problemi che sono risolubili in tempo *polinomiale* da una Macchina di Turing *nondeterministica*

La Macchina di Turing non deterministica esplora tutte le alternative che vuole contemporaneamente, e quindi riesce ad esaminare un insieme di possibilità esponenziale anche in un tempo polinomiale.

La classe $\mathbb{NP}$ ammette una definizione equivalente come *l'insieme dei problemi di cui si può verificare in un tempo polinomiale una proposta di soluzione*.

Ovviamente

$$\mathbb{P} \subseteq \mathbb{NP},$$

ma non si sa se l'inclusione sia stretta.

Maximum Clique: Dato un grafo $G = (V, E)$ ed un intero k, stabilire se il grafo contiene una *cricca* di dimensione k;

Commesso viaggiatore: Dato un grafo (pesato) $G = (V, E)$, trovare, se esiste, un percorso che tocchi *tutti* i vertici, al costo più basso possibile;

Soddisfacibilità: Data una espressione booleana, trovare se esiste una assegnazione delle sue variabili $x_1, \ldots, x_n$ tale che il valore della espressione sia VERO;

Programmazione lineare a valori interi Trovare la soluzione di

$$\begin{array}{rcl} \min & & cx \\ Ax & = & b \\ x & \geq & 0 \end{array}$$

dove A, x e b sono interi.

Consideriamo il caso della *trasformazione* di un problema in un altro.

Definizione

Riducibilità

Dati due problemi P_1 e P_2, diciamo (informalmente) che P_1 si riduce in tempo polinomiale a P_2 se esiste un algoritmo polinomiale A_1 che usa un altro algoritmo A_2 per il problema P_2 nel corso della soluzione di P_1.

In altre parole, per ogni istanza del problema P_1 si può costruire in tempo polinomiale una istanza del problema P_2, la cui soluzione può essere trasformata (in tempo polinomiale) in una soluzione del problema P_1.

Consideriamo il caso della *trasformazione* di un problema in un altro.

Definizione

Riducibilità

Dati due problemi P_1 e P_2, diciamo (informalmente) che P_1 si riduce in tempo polinomiale a P_2 se esiste un algoritmo polinomiale A_1 che usa un altro algoritmo A_2 per il problema P_2 nel corso della soluzione di P_1.

In altre parole, per ogni istanza del problema P_1 si può costruire in tempo polinomiale una istanza del problema P_2, la cui soluzione può essere trasformata (in tempo polinomiale) in una soluzione del problema P_1.

Se un problema è riducibile ad un altro in tempo polinomiale, allora l'esistenza di un algoritmo polinomiale per il secondo problema implica l'esistenza di un algoritmo polinomiale per il primo

Si noti che la riducibilità è transitiva.

La classe dei problemi $\mathbb{NP}$-completi

Definizione

$\mathbb{NP}$-completezza

Un problema (decisionale) si dice $\mathbb{NP}$-completo se

1. *Appartiene a $\mathbb{NP}$;*
2. *Tutti i problemi in $\mathbb{NP}$ sono riducibili ad esso in tempo polinomiale.*

Definizione

La classe $\mathbb{NP}$-hard

Un problema si dice $\mathbb{NP}$-hard (o arduo) se tutti i problemi in $\mathbb{NP}$ sono riducibili ad esso in tempo polinomiale.

Teorema

Teorema di Cook

Il problema della soddisfattibilità è $\mathbb{NP}$-completo

Altri problemi $\mathbb{NP}$-completi:

- Commesso viaggiatore (Traveling Salesman Problem);
- Programmazione Lineare a valori Interi;
- Partizione di un grafo
- Scheduling multiprocessore

$$\mathbb{P} = \mathbb{NP}?$$

È uno dei sette *millenium problems*: in palio un premio da un milione di dollari (oltre alla fama imperitura!)

Ancora su $\mathbb{NP}$-completezza

Alcune considerazioni ulteriori:

1. Teorema di Ladner: se $\mathbb{P} \neq \mathbb{NP}$, allora esistono problemi che non appartengono a $\mathbb{P}$ e però non sono $\mathbb{NP}$-completi;
2. Alcuni problemi sono "debolmente" $\mathbb{NP}$-completi, altri "fortemente". Ad esempio, il problema del commesso viaggiatore è $\mathbb{NP}$-completo anche quando ci sia un limite polinomiale ai numeri contenuti nell'input; viceversa il problema dello zaino è pseudopolinomiale, in quanto si semplifica a polinomiale se le quantità coinvolte sono limitate polinomialmente;
3. Alcuni problemi poi sono intrinsecamente esponenziali, in quanto non è possibile nemmeno *verificare* una soluzione in un tempo polinomiale (es: torre di Hanoi);

La $\mathbb{NP}$-completezza in pratica

Cosa si fa quando vi capita un problema che sembra esponenziale?

Si cerca di dimostrare se sia $\mathbb{NP}$-completo

E poi ?

La $\mathbb{NP}$-completezza in pratica

Cosa si fa quando vi capita un problema che sembra esponenziale?

Si cerca di dimostrare se sia $\mathbb{NP}$-completo

E poi ? La $\mathbb{NP}$-completezza entra in gioco quando si voglia trovare *la soluzione esatta (ottima)* del problema.

La $\mathbb{NP}$-completezza in pratica

Cosa si fa quando vi capita un problema che sembra esponenziale?

Si cerca di dimostrare se sia $\mathbb{NP}$-completo

E poi ? La $\mathbb{NP}$-completezza entra in gioco quando si voglia trovare *la soluzione esatta (ottima)* del problema.

Bisogna quindi "accontentarsi" (rilassare la richiesta):

La $\mathbb{NP}$-completezza in pratica

Cosa si fa quando vi capita un problema che sembra esponenziale?

Si cerca di dimostrare se sia $\mathbb{NP}$-completo

E poi ? La $\mathbb{NP}$-completezza entra in gioco quando si voglia trovare *la soluzione esatta (ottima)* del problema.

Bisogna quindi "accontentarsi" (rilassare la richiesta):

- Si può risolvere un caso particolare del problema;
- Si può accettare un metodo che trovi la soluzione ottima con una probabilità del YY% (ma occasionalmente fallisca);
- Si può accettare un metodo che trovi una soluzione che sia ragionevolmente vicina ma non eguale a quella ottima;
- Si può accettare un metodo che con una probabilità di ZZ% trovi una soluzione *che sia ragionevolmente vicina* a quella ottima.

Spesso ci affidiamo a delle euristiche, ossia delle tecniche di calcolo che funzionano (sufficientemente bene) in pratica ma per le quali non ci sono dimostrazioni rigorose di correttezza ed efficienza.

Copertura dei vertici

Dato un grafo (non orientato) $(\mathcal{V}, \mathcal{E})$, trovare il più piccolo sottoinsieme $\mathcal{V}' \subseteq \mathcal{V}$ tale che se $(u, v) \in \mathcal{E}$, allora $u \in \mathcal{V}'$ oppure $v \in \mathcal{V}'$

Il problema della copertura dei vertici è $\mathbb{NP}$-completo. Un algoritmo possibile (2-approssimato):

Algorithm 1: Approx-Vertex-Cover

Input: $G = (\mathcal{V}, \mathcal{E})$
$C \leftarrow \emptyset$;
$\mathcal{E}' \leftarrow \mathcal{E}$;
while $\mathcal{E}' \neq \emptyset$ **do**
    Trovare un arco $(u, v) \in \mathcal{E}'$;
    $C \leftarrow C \cup \{u, v\}$;
    Rimuovere da $\mathcal{E}'$ tutti gli archi incidenti su u o v;

Il commesso viaggiatore

Dato un grafo (non orientato) $(\mathcal{V}, \mathcal{E})$ un *ciclo hamiltoniano* è un cammino semplice che tocca tutti i nodi. Data una funzione di costo non negativa sugli archi $c(u, v) \geq 0$, trovare il ciclo hamiltoniano di costo minimo.

Il problema del commesso viaggiatore è uno dei più importanti in $\mathbb{NP}$, e trova innumerevoli applicazioni.

Disuguaglianza triangolare

Un grafo non orientato pesato (completo) rispetta la disuguaglianza triangolare se per ogni scelta di tre vertici $u, v, w \in \mathcal{V}$ vale

$$c(u, w) \leq c(u, v) + c(v, w)$$

Algoritmo per commesso viaggiatore

Dato un grafo completo G che rispetta la disuguaglianza triangolare, allora abbiamo un algoritmo per il problema del commesso viaggiatore

Algorithm 2: Approx-TSP-Tour

Input: G

Selezionare un vertice $r \in \mathcal{V}$ come radice;

Trovare un albero di copertura minimo;

Identificare i nodi in una visita anticipata dell'albero;

La lista dei nodi H è il circuito hamiltoniano cercato

Questo algoritmo è 2-approssimato.