

Elementi di Complessità Computazionale

Ultima modifica 23.06.2004

Il problema

- Esiste una misura oggettiva per valutare l'efficienza di un algoritmo?
- In che relazione sono gli input di un algoritmo con il suo tempo di esecuzione?
- Quando un algoritmo può essere considerato “buono”?

Relazione tra gli input e il tempo di esecuzione

- Esiste una relazione tra la dimensione dell'input e il tempo di esecuzione.
- All'aumentare della dimensione dell'input aumenta il tempo di esecuzione.
- Analizzare la bontà dell'algoritmo al variare della dimensione dell'input equivale ad un'analisi di prestazione nel tempo.

Notazione asintotica

- Indica il comportamento asintotico dell'algoritmo, $f(\mathbf{n})$.
- Se “ \mathbf{n} ” è la dimensione dell'input, il comportamento asintotico rappresenta l'andamento del tempo di computazione per \mathbf{n} “molto grande”.
- La notazione utilizzata per rappresentare tale andamento è la notazione O-grande

Notazione O-grande (1)

- Date due funzioni **g** ed **f** a valori positivi:
 - $f(n)$ è $O(g(n))$ se esistono due numeri positivi **c** ed **N** tali che $f(n) \leq c * g(n)$ per qualsiasi $n \geq N$.
- La precedente definizione afferma che **g(n)** è un limite superiore al valore di **f(n)**
- A lungo andare **f** cresce al massimo tanto velocemente quanto **g**

Notazione O-grande (2)

- Quante coppie (c, N) esistono?
 - Il valore di c è correlato a quello di N e viceversa.
- La notazione asintotica dà una rappresentazione dell'andamento della crescita di $f(n)$
- Quante funzioni g esistono che limitano superiormente f ?
 - Si sceglie la funzione minore

Proprietà di O-grande

- (transitività) se $f(n)$ è $O(g(n))$ e $g(n)$ è $O(h(n))$, allora $f(n)$ è $O(h(n))$
- Se $f(n)$ è $O(h(n))$ e $g(n)$ è $O(h(n))$, allora $f(n)+g(n)$ è $O(h(n))$
- La funzione an^k è $O(n^k)$
- La funzione n^k è $O(n^{k+j})$ per ogni j positivo

Ogni funzione polinomiale è O-grande di n elevato alla sua maggiore potenza

Classi di complessità P ed NP

Trattabilità e intrattabilità

- Tutti i problemi risolvibili in tempo polinomiale sono considerati **problemi trattabili**.
- I problemi risolvibili in tempo polinomiale sono chiusi rispetto all'addizione, alla moltiplicazione e alla composizione.

Problemi astratti e Problemi di decisione (1)

- Si definisce “**problema astratto**” Q una relazione binaria sull'insieme I delle istanze del problema e l'insieme S delle soluzioni del problema.
- Un “**problema di decisione**” si può vedere come una funzione che fa corrispondere l'insieme delle istanze I all'insieme delle soluzioni $\{0,1\}$

Problemi astratti e Problemi di decisione (2)

- Molti problemi astratti sono problemi di ottimizzazione (cammino minimo, copertura minima, ciclo hamiltoniano,...)
- Per poter applicare la teoria della NP-completezza ai problemi di ottimizzazione bisogna riformularli come problemi di decisione.

Codifica e Problemi Concreti

- Un problema di decisione astratto prende in input una **codifica dell'istanza del problema**.
- Un problema il cui insieme di istanze è codificato in un insieme di stringhe binarie si chiama **problema concreto**.
- Dato un problema di decisione astratto $Q: I \rightarrow \{0,1\}$, una codifica $e: I \rightarrow \{0,1\}^*$ può essere usata per ottenere un corrispondente problema di decisione concreto $e(Q)$

Risolvibilità di un problema concreto

- Un algoritmo risolve un problema concreto in tempo $O(T(n))$ se produce la soluzione su una istanza \mathbf{i} di lunghezza \mathbf{n} al più in tempo $O(T(n))$.
- Un problema concreto è **risolvibile in tempo polinomiale** se esiste un algoritmo per risolverlo in tempo $O(n^k)$ per qualche costante k

Classe di complessità **P**

- La classe di complessità **P** si può definire come l'insieme dei problemi di decisione concreti che sono risolvibili in tempo polinomiale.

Funzioni correlate polinomialmente

- Per un dato insieme \mathbf{I} di istanze di problema, si dice che due codifiche \mathbf{e}_1 ed \mathbf{e}_2 sono correlate polinomialmente se esistono due funzioni \mathbf{f}_{12} ed \mathbf{f}_{21} calcolabili in tempo polinomiale tali che per ogni $\mathbf{i} \in \mathbf{I}$ si abbia $\mathbf{f}_{12}(\mathbf{e}_1(\mathbf{i})) = \mathbf{e}_2(\mathbf{i})$ ed $\mathbf{f}_{21}(\mathbf{e}_2(\mathbf{i})) = \mathbf{e}_1(\mathbf{i})$
- La codifica $\mathbf{e}_1(\mathbf{i})$ può essere calcolata dalla codifica $\mathbf{e}_2(\mathbf{i})$ attraverso un algoritmo di tempo polinomiale e viceversa

Lemma

- Sia \mathbf{Q} un problema di decisione astratto su un insieme \mathbf{I} di istanze e siano \mathbf{e}_1 ed \mathbf{e}_2 due codifiche di \mathbf{I} correlate polinomialmente. Allora $\mathbf{e}_1(\mathbf{Q}) \in \mathbf{P}$ se e solo se $\mathbf{e}_2(\mathbf{Q}) \in \mathbf{P}$.

Linguaggi formali e problemi di decisione

Linguaggi formali

- Un **alfabeto** Σ è un insieme finito di simboli.
 - ex. $\Sigma = \{0,1\}$
- Un **linguaggio** L su Σ è un qualsiasi insieme di stringhe costituite da simboli di Σ .
 - ex. $L = \{10,11,101,111,\dots\}$ è il linguaggio delle rappresentazioni binarie dei numeri primi.
- Se $\Sigma = \{0,1\}$, allora $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$ è l'insieme di tutte le stringhe binarie. Ogni linguaggio L su Σ è un sottoinsieme di Σ^* .

Relazione tra linguaggi formali e problemi di decisione (1)

- Un algoritmo **A accetta** (o **riconosce**) una stringa $x \in \{0,1\}^*$ se $A(x)=1$.
- Il linguaggio **accettato** da **A** è l'insieme **$L = \{x \in \{0,1\}^* : A(x) = 1\}$**
 - L'insieme delle stringhe accettate da **A**
- Un algoritmo **rifiuta** una stringa x se $A(x)=0$

OSS. Anche se un linguaggio L è riconosciuto da un algoritmo A , non necessariamente rifiuta una stringa $x \notin L$

Relazione tra linguaggi formali e problemi di decisione (2)

- Un linguaggio L è **deciso** da A se ogni $x \in L$ è accettata da A e ogni $x \notin L$ non è accettata da A .
- L è **ricosciuto** in tempo polinomiale da A se, per qualsiasi stringa x di lunghezza n , l'algoritmo accetta x in tempo $O(n^k)$ se e solo se $x \in L$.
- L è **deciso** in tempo polinomiale da A se, dato x , l'algoritmo fornisce una risposta su $x \in L$ o $x \notin L$ in tempo $O(n^k)$.

Linguaggi formali e classi di complessità

- Una **classe di complessità** può essere definita come un **insieme di linguaggi**, per i quali l'appartenenza alla classe è determinata in base ad una misura di complessità dell'algoritmo che **determina se una data stringa x appartiene al linguaggio L** .
- **$P = \{L: L \text{ è riconosciuto da un algoritmo di tempo polinomiale}\}$**

Algoritmo di verifica

- E' un algoritmo A a due argomenti: una stringa di input \mathbf{x} e una stringa detta certificato \mathbf{y} .
- Un algoritmo A a due argomenti **verifica** una stringa \mathbf{x} di input se esiste un certificato \mathbf{y} tale che $\mathbf{A(x,y)=1}$.
- Il **linguaggio verificato** da un algoritmo A di verifica è:

$L = \{x \in \{0,1\}^* : \text{esiste } y \in \{0,1\}^* \text{ tale che } A(x,y) = 1\}$

Classe di complessità **NP**

- La classe di complessità NP è la classe dei linguaggi che **possono** essere **verificati** mediante un algoritmo di tempo polinomiale.

$L = \{x \in \{0,1\}^* : \text{esiste un certificato } y \text{ con } |y| = O(|x|^c), \text{ tale che } A(x,y) = 1\}$

Relazioni tra P ed NP (1)

- Dalla precedente definizione risulta evidente che **$P \subseteq NP$**
 - Ignoro il certificato ed eseguo esattamente i passi passi dell'algorithmo per ottenerlo
- **$P = NP$?**

Linguaggi NP-completi

Riducibilità

- Un problema **Q** può essere ridotto ad un altro problema **Q'** se **una qualsiasi istanza di Q** può essere **reformulata come una istanza di Q'**, e la soluzione di quest'ultima fornisce una soluzione all'istanza di Q.

Riducibilità e Linguaggi

- Un linguaggio L_1 è riducibile in tempo polinomiale ad un linguaggio L_2 ($L_1 \leq_p L_2$), se esiste una funzione calcolabile in tempo polinomiale

$$f: \{0,1\}^* \rightarrow \{0,1\}^*$$

tale che per ogni $x \in \{0,1\}^*$, $x \in L_1$ se e solo se $f(x) \in L_2$

La riducibilità è transitiva.

- **Lemma: Se $L_1, L_2 \subseteq \{0,1\}^*$ sono linguaggi tali che $L_1 \leq_p L_2$, allora $L_2 \in P$ implica $L_1 \in P$.**

NP-completezza

- Un linguaggio $L \subseteq \{0,1\}^*$ è NP-completo se verifica entrambe le seguenti condizioni:
 - **$L \in \text{NP}$**
 - **$L' \leq_p L$ per ogni $L' \in \text{NP}$ (hardness)**

Teorema - Enunciato

- Se un qualsiasi problema NP-completo è risolvibile in tempo polinomiale, allora $P=NP$.
- Equivalentemente, se un qualsiasi problema in NP non è risolvibile in tempo polinomiale, allora tutti i problemi NP-completi non sono risolvibili in tempo polinomiale.

Teorema - Dimostrazione

- (prima asserzione). Si supponga che $L \in P$ e anche $L \in NPC$. Per la proprietà di hardness si ha che $L' \leq_p L$ per ogni $L' \in NP$. Ma per il lemma sulla riducibilità dei linguaggi si ha che $L' \in P$
- (seconda asserzione). Si supponga l'esistenza di un $L \in NP$ tale che $L \notin P$. Sia $L' \in NPC$ un qualsiasi linguaggio NP-completo e, per assurdo, si ipotizzi $L' \in P$. Per il lemma sulla riducibilità dei linguaggi si ha che se $L \leq_p L'$ risulta $L \in P$

A che serve la NP-completezza?

- La ricerca sul quesito $P \neq NP$ si concentra sulla ricerca di problemi NP-completi.

Dimostrazioni di NP-completezza

- Lemma: se L è un linguaggio tale che $L' \leq_p L$ per qualche $L' \in \text{NPC}$, allora L è NP-arduo. Inoltre se $L \in \text{NP}$, allora $L \in \text{NPC}$
- Dimostrazione: poiché L' è NP-completo, per tutti gli $L'' \in \text{NP}$, si ha $L'' \leq_p L'$. Per ipotesi, $L' \leq_p L$ e quindi per transitività si ha che $L'' \leq_p L$, per cui L è NP-hard. Se $L \in \text{NP}$, si ha anche che $L \in \text{NPC}$

Esempi di problemi NP-completi noti

- Ciclo hamiltoniano
- Problema del commesso viaggiatore
- Copertura di vertici
- Copertura di insiemi
- Problema della cricca
- Somma di sotto-insieme

Riferimento

Introduzione agli **ALGORITMI**

Thomas H. Cormen

Charles E. Leiserson

Ronald L. Rivest