Contents

computational complexity

def: problema in computer science

un problema π é una relazione

$$\pi \subseteq I_\pi \times S_\pi$$

dove:

- $I_\pi =$ insieme delle istanze di input del problema
- $S_\pi =$ insieme delle soluzioni del problema

tipologie di problema

- **decisione:**
  - si verifica se una data proprietá é valida per un determinato input
  - $S_\pi = \{true, false\}$ o semplicemente $S_\pi = \{0,1\}$ e la relazione $\pi \subseteq I_\pi \times S_\pi$ corrisponde ad una funzione

    $$f : I_\pi \to \{0,1\}$$

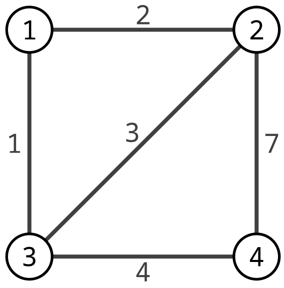
  - esempi: soddisfacibilitá, test di connettivitá di un grafo, etc....
- **ricerca:**
  - data un'istanza $x \in I_\pi$, si chiede di determinare una soluzione $y \in S_\pi$ tale che la coppia $(x,y) \in \pi$ appartengono alla relazione che definisce il problema
  - esempi: soddisfacibilitá, clique, vertex cover, nei quali chiediamo in output un assegnamento di veritá soddisfacente, rispettivamente una clique o un vertex cover, invece di semplicemente "si" o "no"
- **ottimizzazione**
  - data un'istanza $x \in I_\pi$, si chiede di determinare una soluzione $y \in S_\pi$ ottimizzando una data misura della funzione costo
  - esempi: min spanning tree, max SAT, max clique, min vertex cover, min TSP, etc....

complessitá degli algoritmi e dei problemi

- espressa in funzione della taglia dell'input (denotata come $|x|, \forall x \in I_\pi$)
- taglia dell'istanza x
  - quantitá di memoria necessaria a memorizzare x in un computer
  - lunghezza $|x|_c$ della stringa che codifica x in un particolare codice naturale $c : I_\pi \to \Sigma$, dove Σ é l'alfabeto del codice c
- codice naturale
  - conciso: le stringhe che codificano le istanze non devono essere ridondanti o allungate inutilmente
  - numeri espressi in base ≥ 2

esempio: codice

- istanza: grafo G



- codice per G
  - $\Sigma = \{\{, \}, , , 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (simboli)
  - $c(G) = \{1, 2, 3, 4, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, 2, 1, 3, 7, 4\}$
    * $\{1, 2, 3, 4\}$ (nodi)
    * $\{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$ (archi)
    * $\{2, 1, 3, 7, 4\}$ (pesi)
  - $|G|_c = 49$

def: tempo di esecuzione dell'algoritmo A

sia $t_A(x)$ il tempo di esecuzione dell'algoritmo A per l'input x, allora il tempo di esecuzione nel caso peggiore di A é:

$$T_A(n) = \max\{t_A(x) \mid |x| \leq n\}, \qquad \forall n > 0$$

def: complessitá temporale dell'algoritmo A

l'algoritmo A ha complessitá temporale

- $O(g(n))$ se $T_A(n) = O(g(n))$, ovvero

$$\lim_{n\to\infty} \frac{T_A(n)}{g(n)} \leq c, \text{ per una costante } c > 0$$

- $\Omega(g(n))$ se $T_A(n) = \Omega(g(n))$, ovvero

$$\lim_{n\to\infty} \frac{T_A(n)}{g(n)} \geq c, \text{ per una costante } c > 0$$

- $\Theta(g(n))$ se $T_A(n) = \Theta(g(n))$, ovvero

$$T_A(n) = \Omega(g(n)) \text{ e } T_A(n) = O(g(n))$$

def: complessitá di un problema

un problema ha complessitá

- $O(g(n))$ se esiste un algoritmo che lo risolve avente complessitá $O(g(n))$
- $\Omega(g(n))$ se ogni algoritmo A che lo risolve ha complessitá $\Omega(g(n))$
- $\Theta(g(n))$ se ha complessitá $O(g(n))$ e $\Omega(g(n))$

problemi di decisione e classi di complessitá

i problemi di decisione sono solitamente descritti da un’istanza di input (o semplicemente INPUT) e da una DOMANDA sull’input

esempi:

- soddisfacibilitá
  - INPUT: CNF (Conjunctive Normal Form) formula definita su un insieme di variabili
  - DOMANDA: esiste un assegnamento di veritá $\tau : V \to \{0,1\}$?
- clique
  - INPUT: un grafo non orientato $G = (V, E)$ di n nodi e un intero $k > 0$
  - DOMANDA: esiste in G una clique di almeno k nodi ($\geq k$), ovvero un sottoinsieme $U \subseteq V$ tale che $|U| \geq k$ e $\{u, v\} \in E, \forall u, v \in U$?
- vertex cover
  - INPUT: un grafo non orientato $G = (V, E)$ di n nodi e un intero $k > 0$
  - DOMANDA: esiste in G un vertex cover di al massimo k nodi ($\leq k$), ovvero un sottoinsieme $U \subseteq V$ tale che $|U| \leq k$ e $u \in U$ o $v \in U$, $\forall \{u, v\} \in E$?

nei problemi di decisione $I_\pi = Y_\pi \cup N_\pi$

- $Y_\pi =$ insieme di istanze positive, ovvero con soluzione 1
- $N_\pi =$ insieme di istanze negative, ovvero con soluzione 0

def: un algoritmo A risolve π

un algoritmo A risolve $\pi \iff \forall$ input $x \in I_\pi$, A risponde $1 \iff x \in Y_\pi$

def: classe dei problemi $TIME(g(n))$

$TIME(g(n)) =$ classe dei problemi di decisione con complessitá $O(g(n))$

algoritmi non-deterministici per i problemi di decisione

essi si compongono di 2 fasi

- fase 1
  - generano in modo non-deterministico un "certificato" y
- fase 2
  - partendo dall’input x e dal certificato y, verificano se x é un’istanza positiva

def: un algoritmo non-deterministico A risolve π

un algoritmo non-deterministico A risolve π se si ferma per ogni possibile certificato y ed esiste un certificato y per cui A risponde 1 ($true$) $\iff x \in Y_\pi$

- complessitá
  - costo della fase 2
  - espressa in funzione di $|x|$

def: classe dei problemi $NTIME(g(n))$

$NTIME(g(n)) =$ classe di problemi di decisione con complessitá non-deterministica $O(g(n))$

esempio: algoritmo non-deterministico per il problema della clique

- fase 1
  - dato in input il grafo $G = (V, E)$, genera non-deterministicamente un sottoinsieme $U \subseteq V$ di k nodi
- fase 2
  - verifica se U é una clique, ovvero se $\{u, v\} \in E, \forall u, v \in U$, e in tal caso risponde 1, altrimenti risponde 0
- chiaramente l'algoritmo risolve il problema della clique, in quanto si ferma per ogni possibile sottoinsieme U ed esiste un sottoinsieme U per il quale risponde 1 se e solo se esiste una clique di k nodi in G, ovvero $\iff (G, K) \in Y_{clique}$
- complessitá: $O(n^2)$, poiché $|U| \leq |V| = n$

osservazioni (algoritmi deterministici e non-deterministici)

- un algoritmo deterministico é meno potente di uno non-deterministico poiché non puó eseguire la fase 1
- se esiste un algoritmo deterministico A che risolve π, allora esiste anche un algoritmo non-deterministico A' che risolve π con la stessa complessitá come segue:
  - esso esegue al fase 1 e coincide con A nella fase 2, ignorando il certificato y

corollario: $TIME(g(n)) \subseteq NTIME(g(n))$

$$TIME(g(n)) \subseteq NTIME(g(n))$$

- dove:
  - $TIME(g(n)) =$ classe dei problemi deterministicamente risolvibili in tempo $O(g(n))$
  - $NTIME(g(n)) =$ classe dei problemi non-deterministicamente risolvibili in tempo $O(g(n))$

efficienza e trattabilitá

- un problema é trattabile se puó essere risolto efficientemente (deterministicamente)
- sono considerati trattabili o efficientemente risolvibili tutti i problemi aventi complessitá limitata da un polinomio della dimensione dell'input

$$\text{TRATTABILITÁ} \equiv \text{EFFICIENZA} \equiv \text{POLINOMIALITÁ}$$

efficienza e trattabilitá: ragione 1

la crescita delle funzioni polinomiali rispetto a quelle esponenziali (sia per ció che riguarda il tempo di esecuzione sia per ció che riguarda la dimensione delle istanze risolvibili entro un certo tempo di esecuzione)

efficienza e trattabilitá: ragione 2

- la composizione di polinomi é un polinomio e dunque la risolvibilitá in tempo polinomiale di un problema é indipendente da
  - il codice naturale utilizzato, poiché tutti i codici naturali sono correlati in maniera polinomiale
  - il modello computazionale adottato, se ragionevole (cioé costruibile nella pratica o meglio in grado di eseguire un lavoro limitato costante per step), in quanto tali modelli sono polinomialmente correlati, ovvero possono simularsi l'un l'altro in tempo polinomiale

osservazione: macchina di turing non-deterministica

la macchina di turing non-deterministica non é un modello di calcolo ragionevole, poiché la quantitá di lavoro svolto in ogni fase (ciascun livello dell'albero delle computazioni) cresce in modo esponenziale

def: codici polinomialmente correlati

- 2 codici c_1 e c_2 per un problema π sono correlati polinomialmente se esistono 2 polinomi p_1 e p_2 tali che, $\forall x \in I_\pi$:
  - $|x|_{c_1} \leq p_1(|x|_{c_2})$
  - $|x|_{c_2} \leq p_2(|x|_{c_1})$
- se la complessitá rispetto a c_1 é $O(q_1(|x|_{c_1}))$ per un dato polinomio q_1, allora rispetto a c_2 é $O(q_1(p_1(|x|_{c_2}))) = O(q_2(|x|_{c_2}))$ dove q_2 é il polinomio tale che $\forall\lambda\ q_2(\lambda) = q_1(p_1(\lambda))$
- tutti i codici naturali sono correlati polinomialmente, ovvero la risolvibilitá polinomiale non dipende dal particolare codice utilizzato

dimensione dell'input (def: codici correlati polinomialmente)

qualsiasi quantitá polinomialmente correlata ad un codice naturale é dunque correlata ad un qualsiasi codice naturale possibile, dato che tutti i codici naturali sono correlati polinomialmente e che la composizione di polinomi é un polinomio

esempio: codici correlati polinomialmente

- assumiamo che per ogni grafo G di n nodi
  - $|G|_{c_1} = 10n^2$
  - $|G|_{c_2} = n^3$
- se $p_1(\lambda) = 10\lambda$ e $p_2(\lambda) = \lambda^2$ abbiamo che:
  - $|G|_{c_1} = 10n^2 \leq 10n^3 = p_1(|G|_{c_2})$
  - $|G|_{c_2} = n^3 \leq 100n^4 = p_2(|G|_{c_1})$
- dunque i 2 codici sono correlati polinomialmente

- regola pratica:
  - 2 quantitá sono polinomialmente correlate se sono polinomi sulle stesse variabili

esempio: codifica non naturale

- test di primalitá
  - INPUT: un numero intero $n > 0$
  - DOMANDA: n é un numero primo?
  - ALGORITMO (banale):
    * scansiona tutti i numeri da 2 a $n-1$ e risponde 1 ($true$) se nessuno di essi lo divide
  - COMPLESSITÁ: $O(n)$, polinomiale?
  - CODICE c_1 (naturale): n espresso in base 2, ovvero $|n|_{c_1} = \log_2 n$
  - CODICE c_2 (non naturale): n espresso in base 1, ovvero $|n|_{c_2} = n$
- dunque la complessitá dell'algoritmo é:
  - $O(2^{|n|_{c_1}})$ rispetto a c_1, che é esponenziale
  - $O(|n|_{c_2})$ rispetto a c_2, che é polinomiale!
- dimensione dell'input
  - correlata polinomialmente ai codici naturali $|n|_{c_1} = \log_2 n$

def: modelli computazionali simulabili in modo polinomiale

- 2 modelli computazionali M_1 e M_2 sono mutualmente simulabili in modo polinomiale se esistono 2 polinomi p_1 a p_2 tali che:
  1. ogni algoritmo A per M_1 con complessitá $T_A(n)$ puó essere simulato su M_2 in tempo $p_1(T_A(n))$
  2. ogni algoritmo A per M_2 con complessitá $T_A(n)$ puó essere simulato su M_1 in tempo $p_2(T_A(n))$
- dunque se A é polinomiale in M_1 allora é polinomiale anche in M_2 e viceversa
- tutti i modelli computazionali ragionevoli sono mutualmente simulabili in modo polinomiale, ovvero la risolvibilitá polinomiale non dipende dal particolare modello utilizzato

classi P e NP

- $P =$ classe di tutti i problemi risolvibili deterministicamente in tempo polinomiale, ovvero
$$P = \cup_{k=0}^{\infty} TIME(n^k)$$
- $NP =$ classe di tutti i problemi risolvibili non-deterministicamente in tempo polinomiale, ovvero
$$NP = \cup_{k=0}^{\infty} NTIME(n^k)$$
- $P = NP$? nessuno lo a dimostrato

problemi NP-completi

- i problemi piú difficili di NP e tali che se $P \neq NP$ non appartengono a P, viceversa, se 1 di essi appartiene a P, allora $P = NP$
- finora nessuno é riuscito a trovare un algoritmo polinomiale deterministico per nessun problema NP-completo
- congettura: $P \neq NP$

optimization problems

def: problema di ottimizzazione

un problema di ottimizzazione π é una quadrupla $(I_\pi, S_\pi, m_\pi, goal_\pi)$ con:

- $I_\pi =$ insieme delle istanze di input di π
- $S_\pi(x) =$ insieme delle soluzioni ammissibili dell'istanza $x \in I_\pi$
- $m_\pi(x, y) =$ misura della soluzione ammissibile $y \in S_\pi(x)$ per l'input $x \in I_\pi$ (intera)
- $goal_\pi \in \{\min, \max\} =$ specifica se abbiamo un problema di minimizzazione o di massimizzazione

osservazioni (problemi di ottimizzazione)

- assumiamo che $m_\pi(x, y)$ é sempre un numero intero
  - i nostri modelli computazionali possono trattare solo l'approssimazione razionale dei reali
  - scalando tali reali possiamo ottenere numeri interi equivalenti
  - i valori interi rivelano giá le difficoltá intrinseche dei problemi
- quando sono chiari dal contesto (in seguito):
  - π sará omesso
  - $m_(x, y) =$ sará denotato semplicemente come m

esempio: descrizione formale di un problema di ottimizzazione (max clique)

- $I =$ grafo $G = (V, E)$
- $S = \{U \subseteq V \mid \{u, v\} \in E,\ \forall u, v \in U\}$
- $m(G, U) = |U|$
- $goal = \max$

possiamo descrivere i problemi di ottimizzazione nella seguente forma, piú semplice e informale

- MAX CLIQUE
  - INPUT: grafo $G = (V, E)$
  - SOLUZIONE: $U \subseteq V \mid \{u, v\} \in E,\ \forall u, v \in U$
  - MISURA: $|U|$

def: soluzione ottima

- data un'istanza $x \in I_\pi$, una soluzione $y^* \in S_\pi$ é ottima per x se $m(x, y^*) = goal\{m(x, y) \mid y \in S(x)\}$
- la misura di una soluzione ottima (o in modo analogo di tutte le soluzioni ottime) di x é denotata come $m^*(x)$ o semplicemente m^*

problema decisionale sottostante

ogni problema di ottimizzazione ha un problema decisionale sottostante che puó essere ottenuto introducendo un intero k nell'istanza di input e chiedendo se esiste una soluzione ammissibile di misura $\leq k$ (per $\min$) e $\geq k$ (per $\max$)

- problema di ottimizzazione:
  - dato un input x, trova $y \in S(x) \mid m(x,y)$ sia $\min$ o $\max$ (secondo il $goal$)
- problema decisionale sottostante:
  - dato un input x e un intero $k \geq 0$, esiste $y \in S(x) \mid m(x,y) \leq k$ $(\min)$ o $\geq k$ $(\max)$

esempio: descrizione formale di un problema decisionale sottostante (max clique)

- MAX CLIQUE
  - INPUT: grafo $G = (V, E)$
  - SOLUZIONE: $U \subseteq V \mid \{u, v\} \in E, \forall u, v \in U$
  - MISURA: $|U|$
- problema decisionale sottostante:
  - INPUT: grafo $G = (V, E)$ e un intero $k > 0$
  - DOMANDA: esiste una clique U in G tale che $|U| \geq k$

osservazioni (problema decisionale sottostante)

- se esiste un algoritmo polinomiale A per il problema di ottimizzazione, allora esiste un algoritmo polinomiale anche per il problema decisionale sottostante che funziona come segue:
  1. esegue A per determinare la soluzione ottime y^* per l'input x
  2. risponde 1 $(true)$ se $m(x, y^*) \leq k$ $(\min)$ o $\geq k$ $(\max)$
- il problema di ottimizzazione é difficile almeno quanto il problema decisionale sottostante

classi di complessitá dei problemi di ottimizzazione: PO

- un problema di ottimizzazione π appartiene alla classe PO se:
  - per ogni input x, $x \in I$ puó essere verificato in tempo polinomale
  - esiste un polinomio $p \mid \forall x \in I$ e $y \in S(x)$ vale $|y| \leq p(|x|)$
  - $\forall x \in I$ e $y \in S(x)$, $m(x,y)$ puó essere calcolata in tempo polinomale (rispetto a $|x|$)
  - $\forall x \in I$, una soluzione ottima y^* puó essere calcolata in tempo polinomiale
- esempi: shortest path fra 2 nodi, min spanning tree, ecc...

classi di complessitá dei problemi di ottimizzazione: PO

un problema di ottimizzazione π appartiene alla classe NPO se:

- per ogni input x, $x \in I$ puó essere verificato in tempo polinomale
- esiste un polinomio $p \mid \forall x \in I$ e $y \in S(x)$ vale $|y| \leq p(|x|)$
- $\forall x \in I$ e $y \in S(x)$, $m(x,y)$ puó essere calcolata in tempo polinomale (rispetto a $|x|$)

esempi: max clique, min vertex cover, min TSP, ecc...

PO e NPO: nella pratica

- PO: classe dei problemi di ottimizzazione il cui problema decisonale sottostante appartiene a P
- NPO: classe dei problemi di ottimizzazione il cui problema decisonale sottostante appartiene a NP
- chiaramente $PO \subseteq NPO$

def: relazione NPO - NP-$HARD$

un problema di ottimizzazione in NPO é NP-$HARD$ se il problema decisonale sottostante é NP-$Completo$

teorema: relazione tra $P \neq NP$ e risolvibilitá polinomiale dei problemi NP-$HARD$

se $P \neq NP$, un problema di ottimizzazione NP-$HARD$ non puó essere risolto in tempo polinomiale (poiché é difficile almeno quanto il problema decisionale sottostante)

teorema: relazione tra $P = NP$ e $PO = NPO$

se $P = NP$ allora $PO = NPO$

- quasi tutti i problemi che verranno presentati in seguito sono NP-$HARD$, ovvero non efficientemente risolvibili
- verranno progettati algoritmi per tali problemi che restituiscono soluzioni "vicine" a quelle ottime

approximation

greedy

local search

rounding

primal dual

dynamic programming

approximation schemes

alternative approaches

social networks and bibliography

centrality measures

spectral analysis and prestige index

link analysis

web structure

search and advertising

matching markets

auctions

vcg mechanism

gsp mechanism