Analisi della complessita'

Le classi di complessita' suddividono i problemi in base alla **risolubilita'** con un certo quantitativo di risorse (tempo / spazio, si tratta il tempo).

3 categorie principali: 1. Problemi che ammettono algoritmi di soluzione efficienti (**polinomiali**) 2. Problemi intrattabili (irrisolvibili tramite algoritmi efficienti) 3. Problemi per i quali una soluzione efficiente **non e' ancora stata trovata**, ma non e' stato dimostrato che non esista.

A questi si aggiungono i problemi che non sono risolvibili algoritmicamente. La maggior parte dei problemi trattati si identifica nella classe *3*, presentando caratteristiche molto simili gli uni agli altri. Per questo si possono definire metodi / tecniche comuni ai vari problemi.

Problemi di Decisione

Un problema di decisione **ammette solo due possibili soluzioni** (si/no). Puo' essere rappresentato da una coppia `<I,q>` dove *I* e' l'insieme delle istanze e *q* e' un predicato su *I*, ovvero una funzione `q: I -> {0,1}`

Algoritmi decisionali non deterministici

Gli algoritmi decisionali mirano a risolvere problemi di decisione.

Un algoritmo non deterministico di fronte a una scelta: * Sceglie, dove possibile, **la strada giusta** generando **tutte le alternative contemporaneamente** utilizzando **piu' copie di se stesso**.

Se **esiste almeno una scelta che porta al successo**, allora l'algoritmo risponde *si'*.

Schema generale di un A.ND

Gli algoritmi non deterministici operano su di una **sequenza di scelte**: ognuna avviene su un sottoinsieme che e' **funzione delle scelte precedenti**.

Schema generale:

```
non_det(...)
    ...
    for i <- 1 to n
        S[i] <- choice(I[i])
        if (S[1]..S[i] non puo' essere soluzione) failure
        if (S[1]..S[i] e' soluzione) success
    failure / success
```

Certificare la soluzione di un A.ND

La soluzione scelta dev'essere verificata (*certificata*) in maniera efficiente (polinom.) Solitamente, la

risposta ***success*** viene data solo **dopo la verifica**.

Questo significa che **prendendo la decisione giusta in ogni punto di decisione si potrebbe determinare velocemente se esiste una soluzione.** Questo processo e' chiamato ***Oracolo***.

Gli algoritmi non deterministici danno una risposta tramite una serie di *mosse corrette* che poi verificano in maniera efficiente. Quelli deterministici prendono decisioni *provando ad una ad una tutte le possibili scelte*.

Si nota che per quanto un algoritmo sia non deterministico, esistono algoritmi di verifica della soluzione prodotta che sono *deterministici* e con complessita' polinomiale.

Simulare un programma ND con uno D

Per simulare un programma non deterministico con uno deterministico, e' necessario **generare ed esplorare deterministicamente lo spazio delle soluzioni**.

Questo significa visitare l'albero delle scelte corrispondente alla computazione ND, il che ha complessita' **esponenziale** per un numero di foglie **superpolinomiale**.

A.ND: Commesso viaggiatore

Il problema del commesso viaggiatore puo' essere riformulato come un problema di decisione:

```
Dato un grafo G non orientato, completo e pesato, esiste un “ciclo semplice”
(che tocca tutti i vertici) di costo non superiore a un costo prefissato ?
```

(algoritmo su slides)

Si nota come l'insieme delle scelte **varia** durante l'esecuzione dell'algoritmo, in quanto le citta' visitate vengono eliminate dalle successive scelte possibili.

A.ND: Cricca

```
Dato un grafo G non orientato, esiste in G una cricca di
dimensione prefissata k, ossia  esistono in G k vertici
ogni coppia dei quali risulta adiacente ?
```

La soluzione associa ad ogni vertice un attributo *S[i]* booleano che ricorda se il vertice e' gia' stato scelto per la cricca attuale. L'insieme delle scelte non varia.

(algoritmo su slides)

Classi di complessita'

Si effettuano alcune ipotesi per poter trattare in modo **omogeneo** problemi eterogenei:

1. I problemi di decisione non possono essere piu' difficili di uno di ricerca o ottimizzazione. L'intrattabilita' di un problema decisionale **implica l'intrattabilita' delle altre formulazioni del problema**.
2. Si considera il modello di calcolo **RAM**, che coincide con le funzioni calcolabili tramite i formalismi della macchina di Turing, del lambda-calcolo, etc. Questa classe di funzioni e' detta **ricorsive-parziali** ed e' estremamente robusta.
3. Codifica dati **concisa**: la rappresentazione dei dati in una **qualunque base** non cambia la risolubilita' polinomiale del problema, ad eccezione della rappresentazione in base **unaria**.
4. **Costi logaritmici**: vedi **criterio di costo uniforme / logaritmico**.

Criterio di costo uniforme / logaritmico

Basandosi sull'**analisi asintotica della complessita' nel tempo**, in particolare sull'ordine di grandezza nel tendere ad infinito, si puo' definire la rapidita' di incremento nel tempo di calcolo al crescere delle **dimensioni del problema**.

Questo permette di stabilire per che dimensioni dell'input un algoritmo e' utilizzabile e per confrontarne le prestazioni con altri algoritmi.

Per determinare le quantita' di tempo e spazio impiegati da un programma RAM, si usano **due criteri**:

- **Criterio di costo uniforme**: l'esecuzione di un'*istruzione* del programma richiede **1 unita' di tempo indipendentemente dal costo degli operandi**. Utile per casi in cui la dimesione degli interi calcolati non aumenta in maniera eccessiva.
- **Criterio di costo logaritmico**: il tempo necessario a calcolare ogni istruzione **dipende dal numero di bit necessari a rappresentare gli operandi**.

Per questo si puo' definire: **Il tempo di calcolo richiesto al programma P su input x secondo il criterio logaritmico e' la somma dei costi delle istruzioni eseguite nella computazione P su x**.

Questo significa: costo in tempo == somma dei costi delle istruzioni.

Nota: il criterio uniforme e il criterio logaritmico portano alle volte a valutazioni drasticamente differenti, in cui il criterio uniforme risulta in un tempo **molto minore** *del criterio logaritmico.* -> esempio `y = 3^(2n)`

Classe P

La classe P rappresenta tutti i problemi di decisione per la cui soluzine esistono algoritmi deterministici di complessita' polinomiale *nella dimensione dei dati in input.*

Classe NP

La classe NP rappresenta tutti i problemi di decisione per la cui soluzione esistono algoritmi non deterministici di complessita' polinomiale *nella dimensione dei dati in input.*

I problemi appartenenti alla classe NP sono quelli per cui **un algoritmo deterministico di**

complessita' polinomiale non e' ancora stato trovato, ma **esistono algoritmi di certificazione deterministici con complessita' polinomiale**.

Pertanto: NP e' anche definita come la classe di problemi per cui esistono certificati polinomiali.

Problemi interessanti in NP

1. Soddisfattibilita' di circuiti logici (AND,OR,NOT)
2. Soddisfattibilita' di formule (3-SAT)
3. Copertura di vertici
4. Insieme indipendente
5. Copertura esatta di insiemi
6. Circuito Hamiltoniano
7. Programmazione lineare 0/1

P e NP

Si sa che P appartiene a NP (e' un subset), ma non si sa' se la contenenza e' **propria** (non include l'uguale) oppure impropria (include l'uguale). In pratica, **non si e' riusciti a dimostrare se P = NP** oppure se **P != NP**.

Riducibilita' polinomiale

Due problemi, Q e R tale che R: Q e' riducibile a R se:

1. ***esiste una funzione f che trasformi le istanze di input per Q in istanze di input per R.***
2. Per ogni istanza di input di Q, Q risponde *si* su quell'istanza ***se e solo se R risponde si su f(x)***.

```
la 2 matematicamente: Q(x) <=> R(f(x))
```

Riducibilita' polinomiale e NP-completezza

Teoremi

1. Q e' riducibile a R e R appartiene a NP => Q appartiene a NP
2. Q e' riducibile a R e R appartiene a P => Q appartiene a P

Teorema di Cook-Levin

Per ogni Q appartenente a NP => **Q e' riducibile al problema Domino-limitato**.

La dimostrazione del problema, complessa, e' basata sull'utilizzo di una *macchina di turing non-deterministica* che risolva il problema in tempo polinomiale. *(vedi slides)*

Corollario

P = NP <=> Domino-limitato appartiene a P

NP-hard (NP-arduo)

Un problema Q e' detto NP-arduo se: *Per ogni Q' appartente a NP, Q' e' riconducibie a Q*

Questo significa che ***per dimostrare che un problema e' NP-arduo basta dimostrare che esiste un problema NP-arduo polinomialmente riducibile ad esso.*** esempio -> domino-limitato riconducibile a cricca.

NP-completezza

Un problema Q e' NP-completo se: 1. Q e' NP-hard 2. Q e' NP

Riducibilita' - esempi

(slides)

Problemi NP-completi

Importante risultato della riducibilita' polinomiale e' la possibilita' di **esprimere i problemi NP-completi** in forma di altri problemi NP-completi. *(vedi slides per albero)*

Relazioni tra classi di complessita'

NP e NP-completi

Teorema 1

Se **P != NP**, esiste un problema Q che appartiene a **NP-P** non **NP-completo**. Questi problemi sono detti **NP-intermedi**.

Il corollario che ne deriva e': Se **P != NP**, **NP-completi** appartengono (in maniera propria) a **NP-P**

Teorema 2

Se **P != NP**, esistono due problemi Q e R, **NP-intermedi**, tali che: * Q non e' riducibile a R * R non e' riducibile a Q Sono quindi detti *non confrontabili in* `riducibilita'`

Esempi di NP-intermedi

- Isomorfismo tra grafi
- Fattorizzazione

NP-completezza forte

Si considerino i problemi con cardinalita' *n* dell'insieme di dati in input. I problemi per cui **ogni istanza di lunghezza ***n*** e' ristretta a interi di dimensione al piu' ***p(n), funzione polinomiale in n***, sono detti *****fortemente NP-completi***.

Esempi: Cricca, ciclo Hamiltoniano Esempi di problemi non fortemente NP-completi: Somma di sottoinsieme, zaino. Questi infatti dipendono sia dalla cardinalita' dell'insieme in input **sia dai valori degli elementi (piccoli, grandi)**.

(slides per prova)

co-P e co-NP

Il **problema complemento** e' il problema definito sullo stesso dominio del problema originale, ma che da risposta opposta in ogni caso (Si/No -> No/Si)

Le classi complemento co-P e co-NP identificano quei problemi complemento dei problemi in P, NP.

E' facilmente dimostrabile che **co-P coincide con P**, ma dato un problema in **NP**, non si puo' dire **nulla** sul suo complemento.

Si puo' affermare pero' che il complemento di un problema in NP **non e' mai piu' facile del problema originale, semmai piu' difficile**. Questo risultato deriva dal fatto che **per un problema in NP esiste una funzione polinomiale che verifica la soluzione trovata dimostrando che e' vera, mentre per un problema complemento e' necessario dimostrare che nessuna soluzione sia vera, il che e' molto piu' difficile, se non impossibile**. (differenza *Soddisfacibilita'/Falsita'*)

Teorema 3

se **co-NP != NP**, allora **P != NP**

Questo si dimostra per assurdo, supponendo che **P = NP** si ottiene: **NP = P = co-P = co-NP** che e' una contraddizione.

Teorema 4

Se esiste un problema **NP-completo** il cui problema complemento e' **NP**, allora: **co-NP = NP**

Gerarchia di classi di complessita'

(slides per rappresentazione grafica)

Classe E

La classe **E** coincide con tutti i problemi di decisione risolubili con **algoritmi deterministici** in tempo **esponenziale k-plo**, ossia con *k* esponenziali concatenati.

Questa classe coincide con la classe delle funzioni **elementari** cioe' quelle **definibili mediante un numero finito di applicazioni delle quattro operazioni elementari dell' aritmetica/esponenziazione/logaritmiche/di potenza/di radice/trigonometriche**.

Alcuni problemi non sono elementari.

Problemi indecidibili

Problemi che **non possono essere risolti da un computer** anche con **tempo illimitato**. Un esempio e' il **Domino non limitato** (tiling problem), ma anche la corrispondenza di Post (PCP) e l'**halting problem**.

Un problema P che non ammette nessun algoritmo e' **incalcolabile**. Nel caso di un problema

decisionale, viene detto **problema indecidibile**.