Teoria della complessità

- Materiale consigliato:

  testo del corso – capitolo 34
  Introduzione agli algoritmi e strutture dati
  T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
  McGraw Hill, 2005

Definizione di teoria della complessità

- **Teoria della complessità**: classifica i problemi in base alla quantità delle risorse necessarie (in particolare tempo di calcolo) per ottenere la soluzione. Analizza la trattabilità dei problemi quando si pongono dei limiti alle risorse utilizzabili

  In generale, siamo interessati a trovare l'algoritmo più **efficiente** rispetto all'uso delle risorse di calcolo, quali il **tempo** e lo **spazio**

Classificazione dei problemi

- Tre categorie di problemi
  - Problemi che ammettono algoritmi con soluzioni più o meno efficienti
  - Problemi che per loro natura non possono essere risolti con algoritmi efficienti e che sono quindi intrattabili
  - Problemi per cui non sono stati trovati algoritmi efficienti ma per i quali nessuno ha dimostrato che tali algoritmi non esistano

Valutazione dell'efficienza di un algoritmo

- L'efficienza di un algoritmo dal punto di vista del costo nel tempo viene in generale valutata rispetto al comportamento asintotico, ovvero quando la dimensione del problema tende all'infinito
- **Algoritmo efficiente** → tempo di esecuzione limitato da una funzione polinomiale nella dimensione del problema
- **Algoritmo proibitivo** → tempo di esecuzione limitato da una funzione esponenziale nella dimensione del problema

Algoritmo polinomiale

- Un algoritmo si dice di **costo in tempo polinomiale** se il suo tempo di esecuzione è limitato da una funzione polinomiale nella dimensione del problema, ovvero nella lunghezza dei dati di input
- Ogni altro algoritmo che non ricade nella categoria sopra descritta si dice di **costo in tempo esponenziale**
  (si noti che in questa categoria sono compresi gli algoritmi il cui tempo di calcolo è una funzione del tipo $n^{\log n}$, con n dimensione del problema)

Definizione di problema

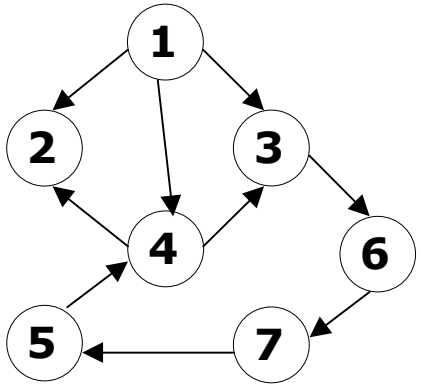
- **Problema**: è una questione di carattere generale alla quale si deve rispondere
- Un problema viene descritto mediante **parametri** i cui valori non sono specificati

- Esempio:
  dato un grafo orientato *G*, determinare un cammino dal vertice `u` al vertice `v`
  Parametri: grafo orientato *G*
  vertici `u` e `v`

Definizione di istanza

- **Istanza**: si dice istanza di un problema la specificazione dei valori assunti dai parametri

- Esempio:
  dato il grafo orientato *G*
  in figura, determinare un
  cammino dal vertice
  `1` al vertice `7`



Definizione di algoritmo

- **Algoritmo**: è una procedura generale per risolvere un problema

- Un algoritmo è **corretto** se per ogni istanza del problema produce la soluzione corrispondente

  Un algoritmo **non corretto** può non terminare in corrispondenza di certe istanze oppure terminare fornendo una soluzione sbagliata

Problemi di decisione

- **Problema di decisione**: è un problema la cui soluzione equivale ad una scelta tra due valori alternativi: SÌ e NO ("1" e "0")

  Esempio:
  dato un grafo orientato *G* e i vertici `u` e `v`,
  esiste un cammino da `u` a `v`?

Problemi di ottimizzazione

- **Problema di ottimizzazione**: è un problema in cui ogni soluzione ammissibile ha un valore associato e si vuole trovare quella con valore migliore (massimo o minimo)

  Esempio: (problema **SHORTEST-PATH**)
  dato un grafo orientato *G* e i vertici `u` e `v`,
  determinare il cammino da `u` a `v` che usa il minor numero di archi, ossia un cammino minimo da `u` a `v`

Problemi di ottimizzazione → problemi di decisione

- Dato un problema di ottimizzazione è possibile trattarlo come un problema di decisione imponendo un **limite** al valore di ottimizzazione

  Esempio: (problema **PATH**)
  dato un grafo orientato *G,* i vertici `u` e `v` e un intero `k` , esiste un cammino da `u` a `v` formato al più da `k` archi?

- Oggetto principale della teoria della complessità sono i problemi di decisione

Problema astratto

- **Problema astratto *Q***: è una relazione che lega l'insieme *I* delle **istanze** del problema e l'insieme *S* di **soluzioni** del problema

  Esempio: problema di decisione **PATH**

  Q: *I* → *S*
  dove
  I: <*G*, u , v , k> istanza del problema
  S: {0, 1} risposta al problema

Problema concreto

- Per poter risolvere un problema astratto con un calcolatore è necessario **codificarlo** mediante un linguaggio da esso interpretabile
- **Codifica**: una funzione *e* che mappa *S* nell'insieme delle stringhe binarie
  $e(Q)$: $I \rightarrow \{0, 1\}^*$
- **Problema concreto**: un problema il cui insieme di istanze è l'insieme delle stringhe binarie

Classe di complessità P

- Un problema concreto è **risolvibile in tempo polinomiale** se esiste un algoritmo che lo risolve nel tempo $O(n^k)$ per qualche costante k

- **classe di complessità P**: è l'insieme dei problemi di decisione concreti che sono risolvibili in tempo polinomiale

Codifica ragionevole

- Una codifica si dice **ragionevole** se
  - utilizza almeno due simboli
  - non introduce dati irrilevanti, né richiede una generazione esponenziale di dati per la rappresentazione di un'istanza del problema

- Proprietà:
  se la codifica è ragionevole, la complessità del problema è indipendente dalla codifica scelta per risolvere il problema

Codifica ragionevole - esempio

Codifica decimale (alfabeto con 10 elementi)			Codifica unaria (alfabeto con un singolo elemento)		
numero simboli dell'alfabeto utilizzati	→	numero simboli rappresentabili	numero simboli dell'alfabeto utilizzati	→	numero simboli rappresentabili
1	→	10	1	→	1
2	→	100	10	→	10
3	→	1000	100	→	100
4	→	10000	1000	→	1000
il numero di simboli cresce di un fattore 10 mentre il numero di cifre cresce come il logaritmo			il numero di simboli cresce linearmente con il numero di cifre		
CODIFICA RAGIONEVOLE			CODIFICA IRRAGIONEVOLE		

Struttura dei linguaggi formali

- **Alfabeto Σ**: è un insieme finito di simboli

- **Linguaggio L su Σ**: è un insieme qualsiasi di stringhe formate da simboli di Σ

  Esempio:
  se Σ = {0, 1} è l'alfabeto allora
  L = {1, 10, 11, 101, 111, 1011, 1101, …} è il linguaggio delle rappresentazioni binarie dei numeri primi

Struttura dei linguaggi formali (2)

- la **stringa vuota** è rappresentata da ε
- il **linguaggio vuoto** è rappresentato da Ø
- il linguaggio di tutte le stringhe su Σ viene indicato con Σ*

  Esempio:
  dato Σ = {0, 1} allora
  Σ* = {ε, 0, 1, 00, 01, 10, 11, 000, …}

Problema di decisione come linguaggio

- Sia Q un problema di decisione e $\Sigma = \{0, 1\}$ l'alfabeto usato per codificare le sue istanze.

  L'insieme Σ^* può essere partizionato in tre sottoinsiemi:
  - l'insieme delle stringhe che NON codificano istanze di Q
  - l'insieme D_Q delle stringhe che codificano istanze di Q
  - l'insieme $Y_Q \subseteq D_Q$ delle stringhe che codificano istanze positive

- Il linguaggio L corrispondente al problema di decisione Q è dato dall'insieme

  $L = \{x \in \Sigma^* : Q(x) = 1\}$

Linguaggio accettato

- Un algoritmo A **accetta** una stringa $x \in \{0, 1\}^*$ se $A(x)=1$
- Un algoritmo A **rifiuta** una stringa $x \in \{0, 1\}^*$ se $A(x)=0$
- Il **linguaggio accettato** da un algoritmo A è l'insieme delle stringhe
  $L = \{x \in \{0, 1\}^*: A(x)=1\}$
  - si noti che anche se L è il linguaggio accettato da A, l'algoritmo non necessariamente rifiuterà una stringa $x \notin L$

Linguaggio deciso

- Un linguaggio è **deciso** da un algoritmo A se ogni stringa binaria in L è accettata da A e ogni stringa che non appartiene ad L è rifiutata da A

- per accettare un linguaggio un algoritmo si occupa solo delle stringhe in L

- per decidere un linguaggio un algoritmo deve accettare o rifiutare correttamente ogni stringa in {0, 1}*

Linguaggi accettati e decisi in tempo polinomiale

- Un linguaggio L è **accettato in tempo polinomiale** da A se è accettato da A e se esiste una costante k tale che, per qualsiasi $x \in L$ di lunghezza n, l'algoritmo accetta x nel tempo $O(n^k)$
- Un linguaggio L è **deciso in tempo polinomiale** da un algoritmo A se esiste una costante k tale che per qualsiasi $x \in \{0, 1\}^*$ di lunghezza n, l'algoritmo decide se $x \in L$ nel tempo $O(n^k)$

Classe di complessità P

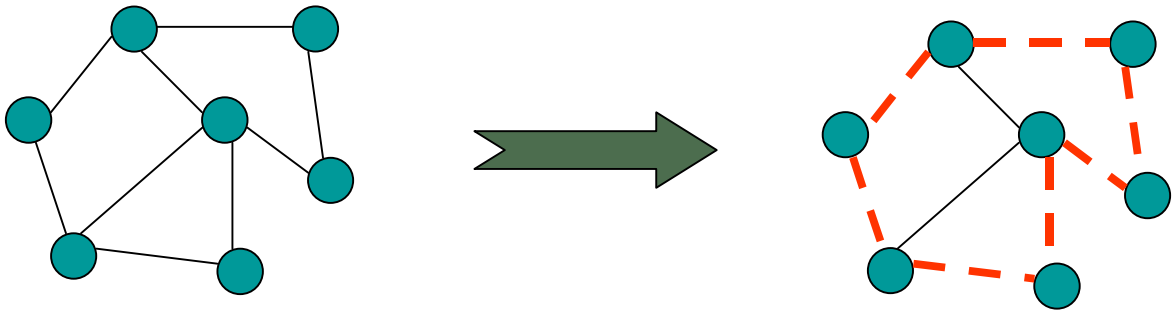
- **Classe di complessità P** è la classe dei linguaggi per i quali esiste un algoritmo che decide L in tempo polinomiale

  $P = \{L \subseteq \{0, 1\}^*$: esiste un algoritmo A che decide L in tempo polinomiale $\}$

  - Si noti come la classe di complessità P sia anche la classe dei linguaggi che possono essere accettati in tempo polinomiale

Problema del ciclo hamiltoniano

- Dato un grafo non orientato G = (V, E) si dice **ciclo hamiltoniano** un ciclo semplice che contiene ogni vertice del grafo una e una sola volta
- Un grafo che contiene un ciclo hamiltoniano si dice **grafo hamiltoniano**



Problema del ciclo hamiltoniano (2)

- **Problema del ciclo hamiltoniano**: un grafo G ha un ciclo hamiltoniano?
  HAM-CYCLE ={G: G è un grafo hamiltoniano}

- Possibile algoritmo di decisione:
  - elenca tutte le permutazioni dei vertici di G
  - verifica se ciascuna permutazione è un ciclo hamiltoniano

- Tempo di calcolo non polinomiale

Verifica in tempo polinomiale

- Esempio: (problema **HAM-CYCLE**)
  dati
  - una particolare istanza *G*
  - una sequenza di vertici $p = <v_1 , v_2 , v_3 , \ldots >$

  è facile verificare in tempo polinomiale se la sequenza di vertici data è un ciclo hamiltoniano.
  Se ciò accade possiamo considerare *p* come **certificato** del fatto che l'istanza data appartenga a PATH

Linguaggio verificato

- **Algoritmo di verifica**: è un algoritmo con due parametri
  1. la stringa di input x
  2. la stringa binaria y detta **certificato**

- Il **linguaggio verificato** da un algoritmo A è

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

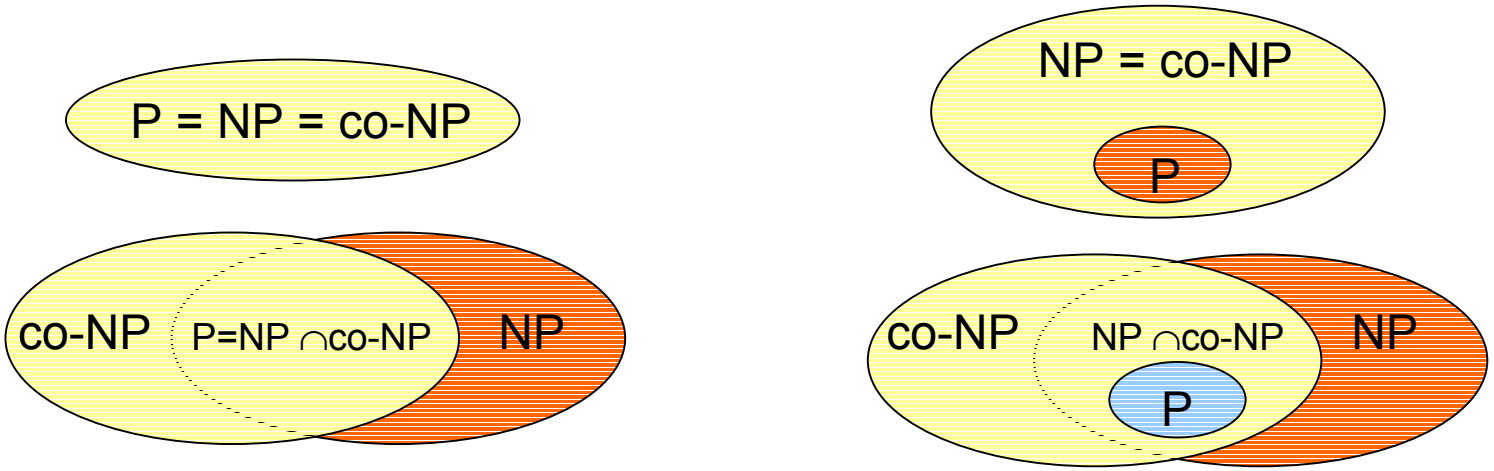
Classe di complessità NP

- **Classe di complessità NP** è la classe dei linguaggi per i quali esiste un algoritmo che verifica L in tempo polinomiale

  NP = {L $\subseteq$ {0, 1}*: esiste un algoritmo A che verifica L in tempo polinomiale }

Relazione classi P e NP

- P = NP ???
  - classe P - sono problemi rapidamente risolvibili
  - classe NP - sono problemi rapidamente verificabili
- NP è chiusa rispetto al complemento ???
  (NP = co-NP)



Classe di problemi NP-completi

- proprietà:
  se risultasse che un *qualsiasi* problema NP-completo può essere risolto in tempo polinomiale, *allora* tutti i problemi in NP avrebbero un algoritmo in tempo polinomiale e quindi sarebbe P=NP

- Nonostante gli anni di studio ancora non si è scoperto un algoritmo con tempo polinomiale per un qualsiasi problema NP-completo

Riducibilità

- Un problema *Q* può essere ridotto a un altro problema *Q′* se un'istanza qualsiasi di *Q* può essere facilmente riformulata come istanza di *Q′*

- esempio:
  il problema di risolvere le equazioni lineari in x
  si riduce al problema di risolvere le equazioni quadratiche
  $ax + b = 0 \quad \rightarrow \quad 0x^2 + ax + b = 0$

- *Q* non è più difficile da risolvere rispetto a *Q′*

Linguaggio riducibile in tempo polinomiale

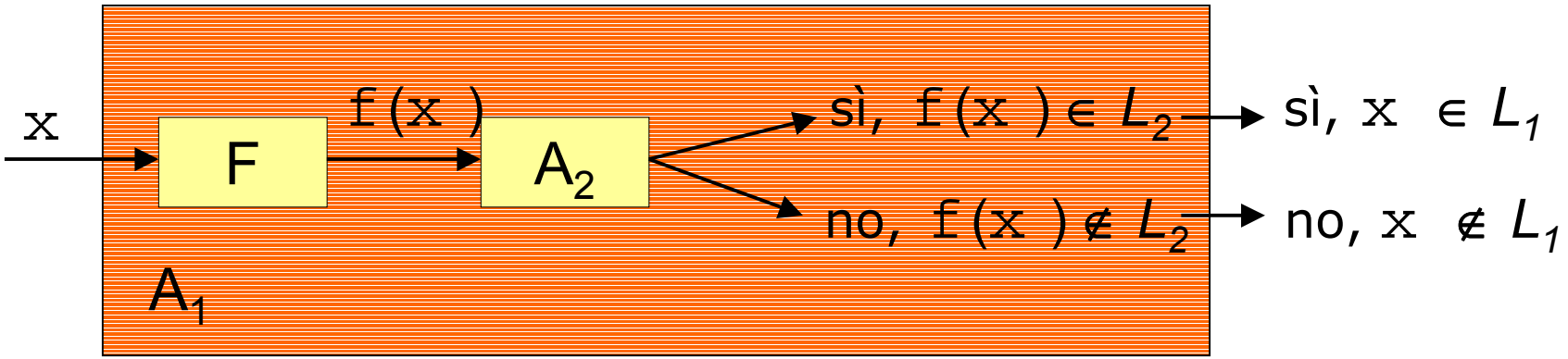
- Un linguaggio L_1 è **riducibile in tempo polinomiale** a un linguaggio L_2 ($L_1 \leq_p L_2$) se esiste una funzione `f`, calcolabile in tempo polinomiale, tale che per ogni $x \in \{0, 1\}^*$ $x \in L_1$ se e soltanto se $f(x) \in L_2$

  dove

  - $f:\{0, 1\}^* \rightarrow \{0, 1\}^*$ è detta **funzione di riduzione**
  - F che calcola `f` è detto **algoritmo di riduzione**

Riducibilità

- graficamente



  - l'algoritmo viene eseguito in tempo polinomiale perché entrambi gli algoritmi F e A vengono eseguiti in tempo polinomiale
  - Le riduzioni in tempo polinomiale sono uno strumento formale che permette di dimostrare che un problema è difficile al più quanto un altro, a meno di un fattore di tempo polinomiale

NP-completezza

- Un linguaggio $L \subseteq \{0, 1\}^*$ è **NP-completo** se
  1- $L \in$ NP
  2- $L' \leq_p L$ per ogni $L' \in$ NP

- L'insieme dei linguaggi NP-completi (NPC) rappresenta i problemi più difficili in NP

- Se un linguaggio L soddisfa la proprietà 2 ma non necessariamente la 1 allora è **NP-difficile**

Teorema

- Se un qualsiasi problema NP-completo è risolvibile in tempo polinomiale, allora P = NP. Analogamente, se un qualsiasi problema in NP non è risolvibile in tempo polinomiale, allora nessun problema NP-completo è risolvibile in tempo polinomiale
  Dimostrazione:
  $L \in$ NPC e $L \in$ P → per qualsiasi $L' \in$ NP si ha $L' \leq_p L$ quindi $L' \in$ P
  La seconda parte della dimostrazione viene dimostrata notando che è la contrapposizione della prima parte