Automi, Complessità e Calcolabilità
Prof. E. Fachini

introduzione al corso:

- **il programma in sintesi**
- **aspetti organizzativi**
- **il libro di testo**
- **i momenti di verifica**

Introduzione al corso: cosa si studierà

1. Linguaggi formali e modelli di calcolo
2. Teoria della calcolabilità
3. Teoria della complessità computazionale

Linguaggi formali e modelli di calcolo

Introdurremo vari modelli matematici di computazione di potere computazionale via via crescente.

Dovremo quindi avere un metodo uniforme di confronto del potere computazionale di diversi modelli: il linguaggio

Linguaggio: definizione

Una parola è una sequenza di simboli presi da un insieme finito, l'alfabeto.

Un linguaggio è un insieme di parole.

Esempio:
sia {0,1} l'alfabeto
allora tutte le sequenze binarie costituiscono l'insieme delle parole che si possono ottenere su {0,1}.
Un qualsiasi suo sottoinsieme, anche l'insieme vuoto o l'insieme di tutte le stringhe binarie, è un linguaggio.
Per esempio $L= \{0^n1^n \mid n\geq 0\}$ è il linguaggio delle parole su {0,1} che sono formate da 0 seguiti da 1 e che sono nello stesso numero.

Riconoscitori di linguaggi

Un riconoscitore di un linguaggio L è una macchina che riceve in input una parola sull'alfabeto di L e dà in output sì o no a seconda se la parola in input è in L o no.

La teoria dei linguaggi formali è ricca di risultati di grande valore applicativo per l'informatica, dalla progettazione di compilatori agli algoritmi per la manipolazione di testi, etc.
Ma vedere i problemi come linguaggi ne semplifica anche lo studio dal punto di vista della complessità computazionale.

Linguaggi formali e modelli di calcolo

Introdurremo gli automi a stati finiti, gli automi a pila e le macchine di Turing e ne studieremo le proprietà fondamentali.

Questi modelli di calcolo sono considerati dei riconoscitori di linguaggi, piuttosto che delle macchine che dato un input producono un output, cioè calcolano una funzione o una relazione.

Più avanti dimostreremo che questa scelta non è limitativa, quando si considera la complessità dei problemi.

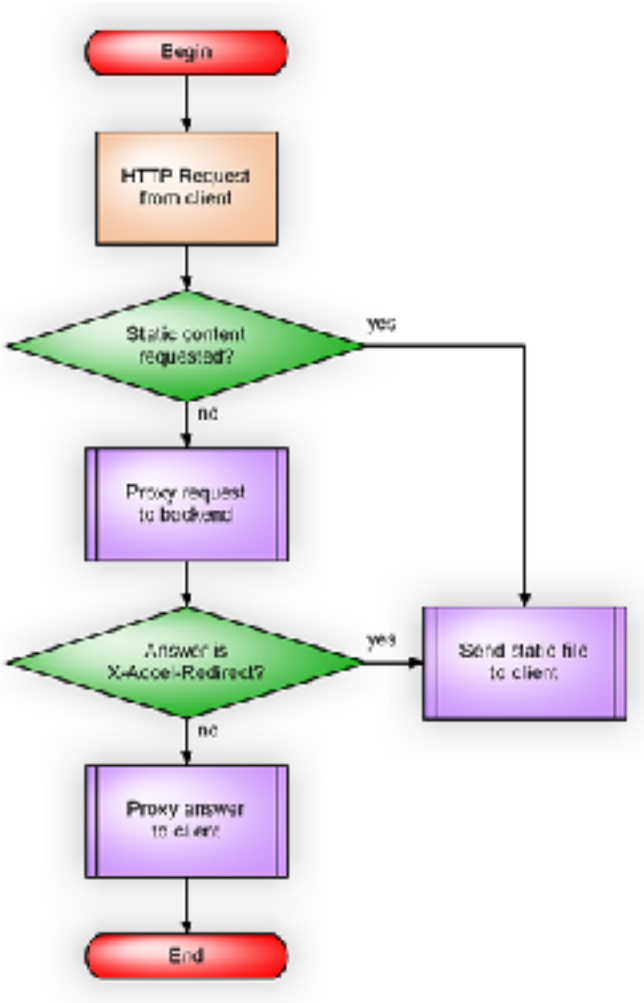
Teoria della calcolabilità

È la teoria che studia quali problemi possono essere risolti con un calcolatore.

Trovare una soluzione algoritmica per un problema può essere arduo, in molti casi è impossibile!

Algoritmo

Quando ci troviamo di fronte ad un algoritmo che risolve un problema siamo in grado di riconoscerlo come tale.



NON esistenza di un algoritmo

Volendo provare che per un particolare problema non esiste un algoritmo è necessario disporre di una definizione formale di algoritmo.

Church e Turing indipendentemente nel 1936 diedero due risposte diverse, subito provate equivalenti.

Per quale problema hanno dimostrato la non esistenza di un algoritmo?

David Hilbert

1862 - 1943

Un forte impulso verso la definizione precisa di cio' che e' calcolabile viene da Hilbert. Hilbert pensava che tutta la matematica potesse essere assiomatizzata.

Una volta ottenuta questa assiomatizzazione si sarebbe potuta concepire una procedura effettiva che avrebbe deciso in un numero finito di passi se una affermazione matematica era deducibile dagli assiomi o no, quindi se era vera o falsa, tenuto conto che l'assiomatizzazione doveva essere completa e consistente.

Entscheidungsproblem (problema della decisione)

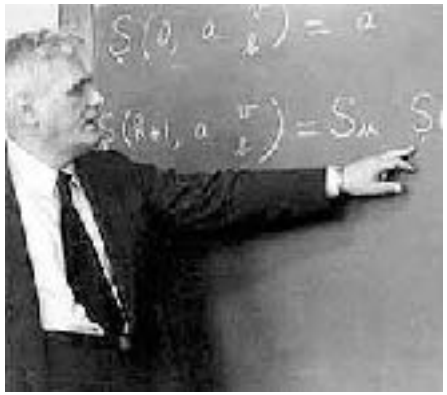
Un algoritmo come quello cercato da Hilbert avrebbe potuto risolvere problemi aperti della matematica come

la

Goldbach's conjecture: ogni intero pari maggiore di 2 è esprimibile come la somma di due interi primi.

La congettura vale fino a 4x1018, ma non è dimostrata in generale.

Il problema della decisione di Hilbert (1862-1943)



Risolto negativamente da Church, introducendo il λ-calcolo e

nel 1936

da Turing, introducendo le macchine di Turing

Turing e Church

- **Si dimostra che una funzione calcolabile con una macchina di Turing è λ-definibile e viceversa**

- **L'insieme delle funzioni calcolabili è indipendente dal modello di calcolo (purchè generale).**

Una funzione f è calcolabile se esiste un algoritmo che

- **dà in output il valore della funzione sull'input x, se f è definita su x**
- **e non si ferma altrimenti.**

Alonso Church

4/6/1903-11/8/1995

La nozione di λ-definibile è dovuta a Church e a Kleene (1933) e quella di ricorsività è dovuta a Gödel, la prova della loro equivalenza è dovuta a Kleene.

Il λ-calcolo ha avuto una grande influenza sulla costruzione dei linguaggi di programmazione, in particolare i funzionali (pure Lisp, Haskell, Miranda and ML).

E' il metalinguaggio della semantica denotazionale

Una funzione di interi positivi è calcolabile se è λ-definibile oppure se è ricorsiva.

Alan Turing

1912- 1954

Da wikipedia:
Nel 1934 si laureò con il massimo dei voti presso l'Università di Cambridge
Nello stesso anno si trasferì alla Princeton University dove studiò per due anni, sotto la direzione di Church, ottenendo infine un Ph.D.

In quegli anni pubblicò l'articolo "*On computable Numbers, with an application to the Entscheidungsproblem*" nel quale descriveva, per la prima volta, quella che sarebbe poi stata chiamata la *macchina di Turing* e nel quale risolve negativamente il problema della decisione di Hilbert.
Qui egli dimostra che il problema della fermata per macchine di Turing non è algoritmicamente risolvibile.

“Mathematical Problems” di Hilbert

Hilbert presenta 10 (su 23 da lui individuati) problemi al Secondo Congresso Internationale di Matematica in Parigi, nel 1900. Tra questi uno è chiaramente un problema algoritmico :

C’e’ un algoritmo che, ricevuto in input un polinomio a coefficienti interi, stabilisce se la relativa equazione ha soluzione intera?

Hilbert: “…to devise a process according to which it can be determined by a finite number of operations whether the (diophantine) equation is solvable in rational integers.”

Nota. Diophantine: da Diophantus, matematico greco del 3° secolo a.c.

10° problema di Hilbert:
c'è un algoritmo che riceve in input un polinomio a coefficienti interi e dà in output "sì" se l'equazione corrispondente ha soluzioni intere e "no" altrimenti?

Risolto negativamente da Matijasevic (1947) nel 1970, grazie ai risultati precedenti di Julia Robinson (1019-1985) e altri.

Teoria della calcolabilità: cosa ne studieremo

E' una vasta area di studio che comprende

- **la definizione, lo studio e il confronto di vari modelli di calcolo,**
- **metodi di prova per dimostrare la non esistenza di un algoritmo per risolvere un problema**

Noi introdurremo un unico modello di calcolo generale, la Macchina di Turing, e alcune sue varianti,

- **un unico metodo di prova, la funzione di riduzione,**
- **e dimostreremo l'indecidibilità di alcuni problemi, tra cui quello della fermata per le macchine di Turing e altri la cui indecidibilità si ottiene con il metodo della riduzione.**

Teoria della complessità computazionale

Ordinare n numeri è facile: il tempo di esecuzione può essere in Θ(n lg n).

Mentre determinare se in un grafo diretto con n vertici c'è un cammino hamiltoniano, da un vertice s a uno t, cioè un cammino in cui tutti i vertici sono attraversati una e una sola volta, è più difficile.

L'algoritmo basato sulla ricerca esaustiva è in O(n!), dove n è il numero dei vertici del grafo in input.

Teoria della complessità computazionale

Ma la complessità esponenziale è proibitiva, per dimensioni di istanze anche ragionevolmente piccole, tanto da rendere i problemi di complessità esponenziale intrattabili, come se l'algoritmo che li risolve non ci fosse!

Per esempio se il tempo di esecuzione nel caso peggiore è in $O(2^n)$, per n = 54, occorreranno 6 mesi per avere la risposta, su un computer che esegua 2^{30} operazioni al secondo.

La teoria fornisce risultati volti a capire le ragioni di questa complessità e a "dominarla", se possibile.

Classi di complessità computazionale

Raggruppiamo in una classe tutti i problemi che possono essere risolti con algoritmi della stessa complessità (di tempo o di spazio) nel caso peggiore.

P = la classe dei problemi che possono essere risolti con un algoritmo di complessità di tempo polinomiale nel caso peggiore.
[cioè in $O(n^k)$, per $k \geq 0$]

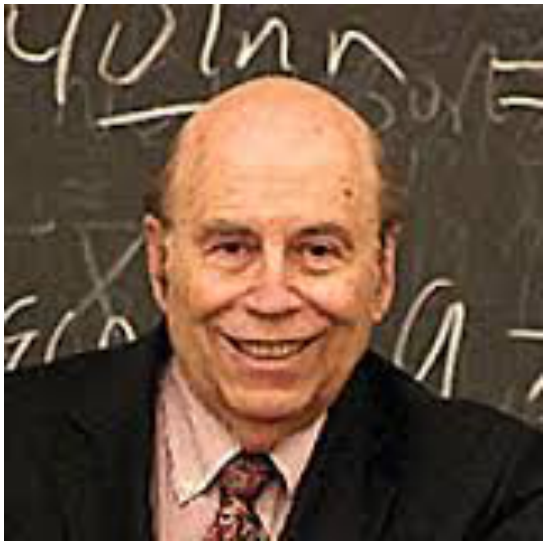
Es. ordinamento è in P

I problemi trattabili

Cobham [64], Edmonds [65] e Rabin [66] hanno contribuito alla identificazione di P come la classe dei problemi trattabili:

Tesi di Cobham o anche Cobham-Rabin

Jack Edmonds



Michael Rabin

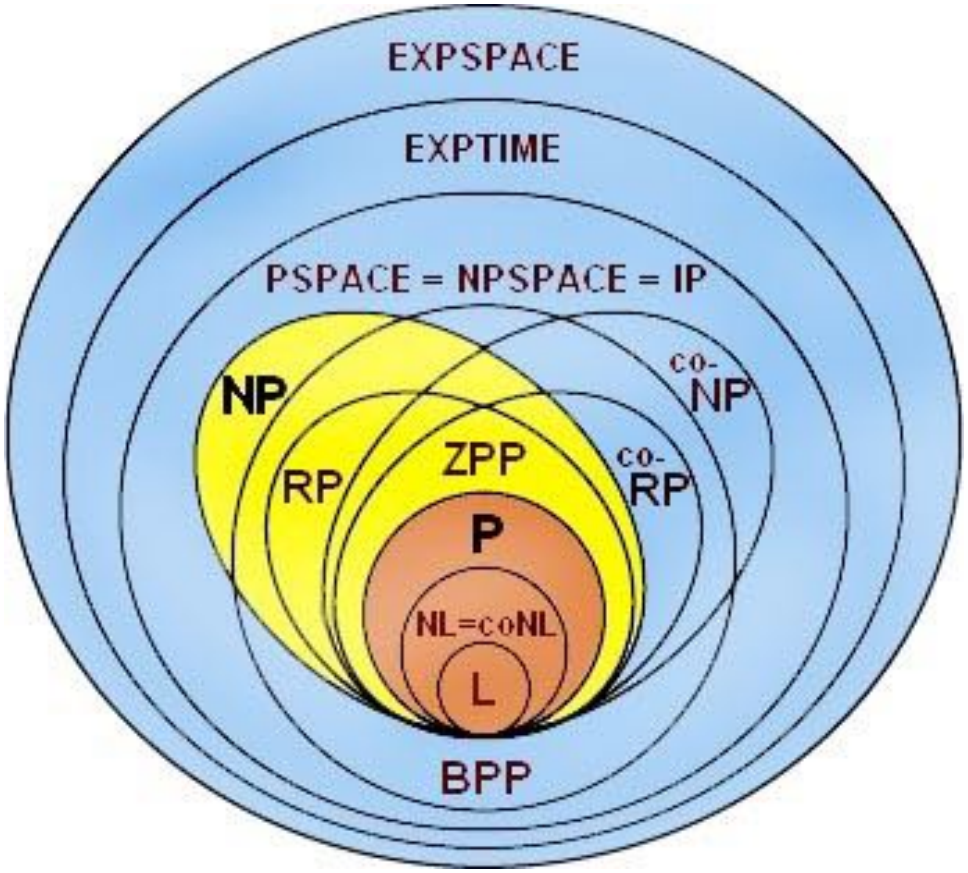
Esempi di classi di complessità

EXP = la classe dei problemi che possono essere risolti con un algoritmo con complessità di tempo esponenziale [cioè in $O(2^{n^k})$, per $k \geq 0$] nel caso peggiore.

Es. il problema del cammino hamiltoniano, che può essere risolto da un algoritmo il cui tempo di esecuzione nel caso peggiore è in $O(n^2 2^n)$.

Teoria della complessità computazionale

Studieremo le relazioni tra queste classi, in qualche caso non sappiamo la risposta!



I problemi intrattabili, forse

Cook [71] e Levin [73] hanno introdotto il concetto di NP-completo. La classe NP è quella dei problemi verificabili in tempo polinomiale

Stephen Cook

Se si trova un algoritmo polinomiale per un problema NP-completo si dimostra che P = NP

Leonid Levin

Complessità

- **Studieremo le classi di complessità più note**
- **Introdurremo il concetto di riduzione polinomiale tra problemi come strumento per dimostrare l'appartenenza o meno a una classe di complessità.**
- **esporremo il più importante problema aperto dell'informatica teorica: P vs NP.**

Aspetti organizzativi

- **Il testo di riferimento è**

  Autore: Michael Sipser

  Titolo: Introduction to the Theory of Computation

Titolo italiano: Introduzione alla teoria della computazione a cura di Clelia de Felice, Luisa Gargano e Paolo D'Arco

Editore: Course Technology (Thomson)

per la traduzione: Apogeo

- **La pagina del corso, su TWIKI, è raggiungibile dalla pagina degli insegnamenti**
- **Le lezioni si svolgeranno con il seguente orario: le prime due ore dalle 10,30 alle 12, la terza ora dalle 12,15 alle 13.**

Momenti di verifica

- **Quest'anno vi propongo il seguente schema:**
- **ogni mercoledì, dalle 13,15 alle 13,50, a turni alterni la metà di voi risolverà in aula un esercizio e risponderà a una domanda di teoria.**
- **chi avrà ottenuto almeno la sufficienza in 2 sui primi 3 momenti valutativi potrà proseguire con questa modalità di esame e affrontare gli ultimi 2.**
- **chi avrà ottenuto la sufficienza anche nei 2 ultimi avrà concluso l'esame.**
- **Chi invece ha abbandonato questa modalità potrà sostenere l'esame scritto consistente in 3 esercizi sul programma.**
- **chi avrà ottenuto la sufficienza solo in 4 momenti valutativi potrà recuperare nelle sessioni di gennaio e febbraio.**

Modelli di calcolo

Durante il corso dovremo confrontare il potere computazionale di diversi modelli di calcolo.

Per poter confrontare tra loro modelli di calcolo diversi bisogna decidere cosa intendiamo per computazione.

Abbiamo già detto che consideriamo le macchine come riconoscitori di linguaggi.

Questa scelta si rivela vincente anche per la teoria della complessità.

Prima considerazione: trattiamo solo problemi decisionali

Un problema decisionale è un problema che ha solo risposta sì o no.

Esempio di problema decisionale: Dato un grafo diretto G=(V,E) e due suoi vertici s e t, esiste un cammino hamiltoniano nel grafo da s a t, cioè un cammino che passa per ogni vertice una sola volta?

Un istanza di un problema è un particolare input per quel problema.

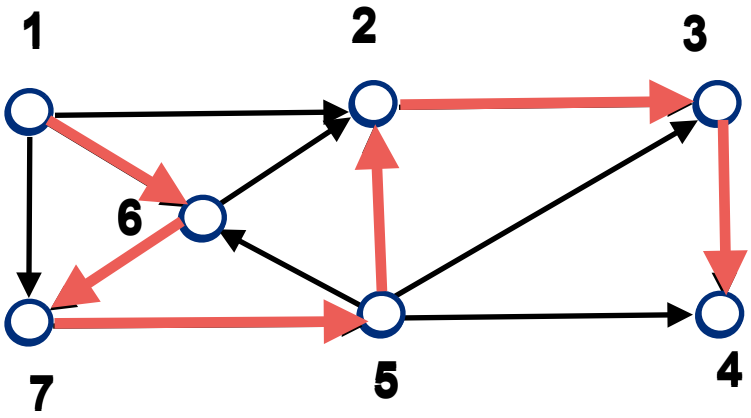
Istanze sì

Esempio di istanza: Un particolare grafo diretto, e due suoi vertici costituiscono un'istanza del problema del cammino hamiltoniano.

Chiamiamo istanze sì quelle che hanno soluzione sì e istanze no quelle che hanno soluzione no.

input G, 1,4

G=

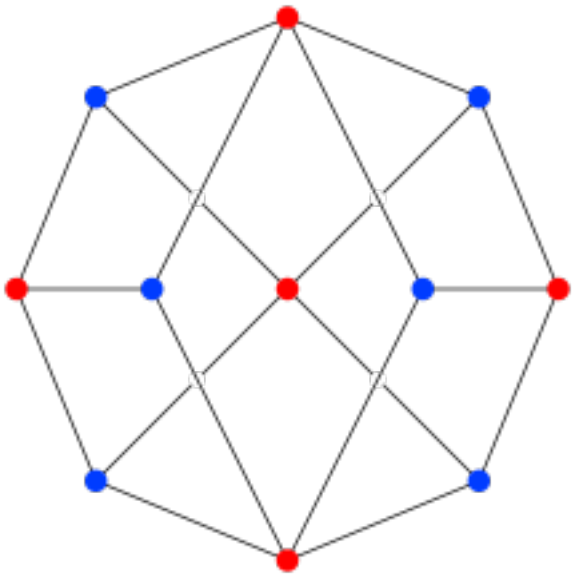


istanza sì

Problemi decisionali e linguaggi e istanze no

Esempio di istanza: Un particolare grafo diretto, e due suoi vertici costituiscono un'istanza del problema del cammino hamiltoniano.

Chiamiamo istanze sì quelle che hanno soluzione sì e istanze no quelle che hanno soluzione no.



istanza no

Linguaggi e problemi

Un problema decisionale può essere facilmente "visto" come un linguaggio:

scelto un alfabeto di codifica per il problema decisionale:

- **una parola corrisponde a un' istanza**
- **il problema = insieme delle sue istanze**
- **l'insieme delle (codifiche delle) sue istanze sì è il linguaggio associato al problema**

Macchine e problemi

- **Modelli di calcolo sono riconoscitori di linguaggi**

- **Una "macchina" del modello di calcolo che riconosce il linguaggio delle (codifiche delle) istanze sì corrisponde a un algoritmo che risolve il problema**

Problemi decisionali e linguaggi

Una computazione è identificata con il riconoscimento di una parola e un algoritmo con il riconoscimento un linguaggio. Di seguito giustifichiamo questa scelta

Il problema: Dato un grafo diretto G=(V,E), e due suoi vertici s e t esiste un cammino hamiltoniano da s a t nel grafo?

Il linguaggio associato al problema è quello delle istanze sì del problema.

L'alfabeto è quello della codifica delle istanze del problema del cammino hamiltoniano, (per esempio l'alfabeto binario) e il linguaggio che ci interessa è quello delle parole (binarie) che codificano grafi diretti e due vertici del grafo s e t che hanno un cammino hamiltoniano da s a t, cioè le istanze sì.

Problemi decisionali e linguaggi

Il problema: Dato un grafo diretto G=(V,E), e due suoi vertici s e t esiste un cammino hamiltoniano da s a t nel grafo?

L'alfabeto è quello della codifica delle istanze del problema del cammino hamiltoniano, (per esempio l'alfabeto binario) e il linguaggio che ci interessa è quello delle parole (binarie) che codificano grafi diretti e due vertici del grafo s e t che hanno un cammino hamiltoniano da s a t, cioè le istanze sì.

Un dispositivo che riconosce, tra le parole che codificano le istanze, quelle che codificano le istanze sì è un algoritmo che risolve il problema decisionale del cammino hamiltoniano in un grafo.

Problemi decisionali e linguaggi

Una computazione è identificata con il riconoscimento di una parola e

un algoritmo con il riconoscimento un linguaggio.

Problemi decisionali e ricerca di una soluzione

Un problema decisionale è un problema la cui risposta è di tipo SI/NO. Considerare problemi di questo tipo consente di concentrarsi sulle difficoltà computazionali.

In generale in realtà siamo interessati a trovare una soluzione piuttosto che a sapere se c'è una soluzione.

Chiamiamo problemi di ricerca, in contrapposizione ai problemi di decisione, quelli per i quali cerchiamo una soluzione.

Problemi decisionali e ricerca di una soluzione: l'esempio

Per esempio il problema della ricerca di un cammino hamiltoniano consiste nel fornire in output un cammino hamiltoniano in un grafo diretto G da un suo vertice s a un suo vertice t e non solo la risposta che un tale cammino c'è.

Ma possiamo usare un algoritmo per il problema decisionale per costruire un algoritmo per il problema della ricerca.

Il caso del cammino hamiltoniano

Esempio: Eseguendo l'algoritmo di decisione verifichiamo se un grafo diretto ha un cammino hamiltoniano tra due vertici dati.

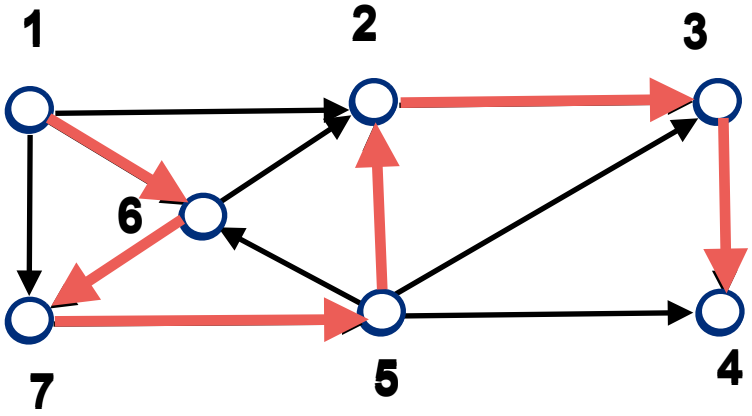
Se la risposta è no il processo termina, altrimenti rimuoviamo un arco se il grafo ottenuto dopo la rimozione ha ancora un cammino hamiltoniano, verificandolo utilizzando l'algoritmo per il problema di decisione.

Al termine del processo resta giusto il cammino hamiltoniano.

Esempio Hamilton

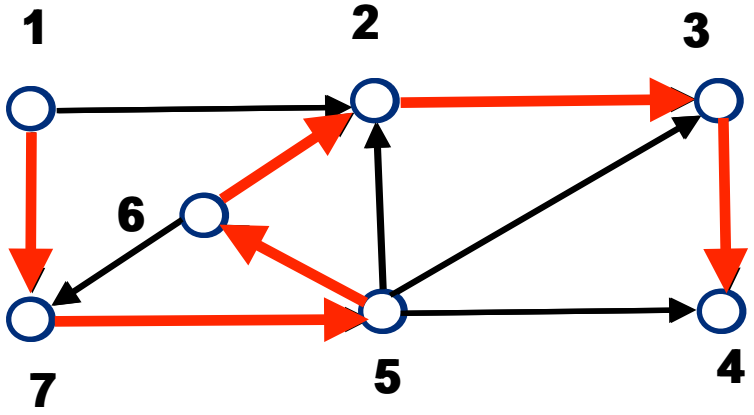
input G, 1,4

G=

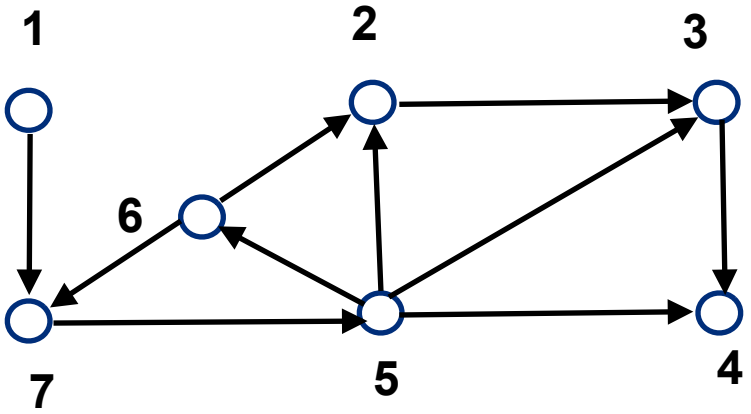


Considera (1,6)

G=



G=

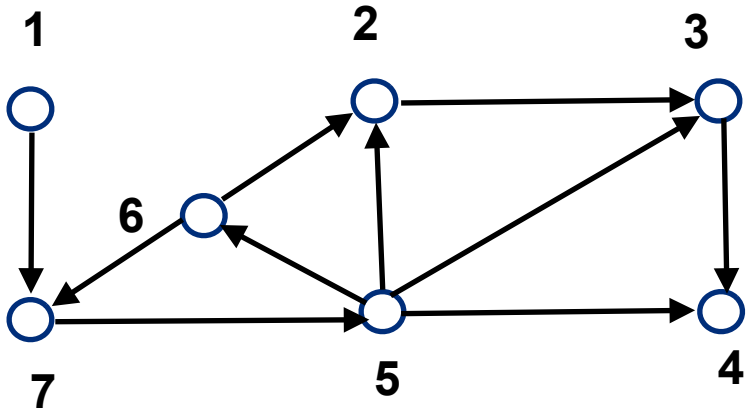


L'eliminazione di (1,2) anche lascia un grafo con un cammino hamiltoniano

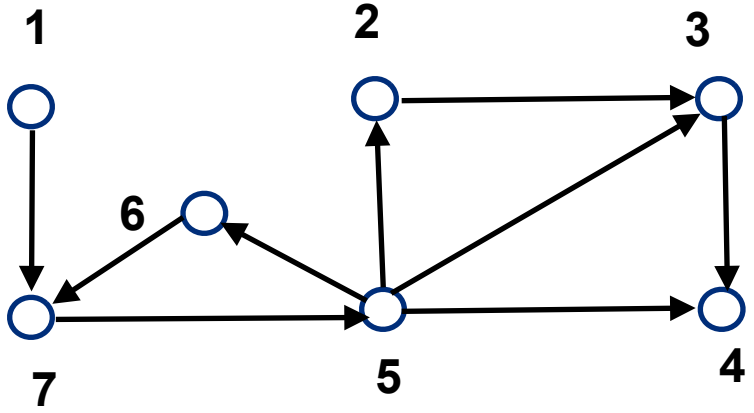
Esempio Hamilton

Considera (6,2)

G=



G=



L'eliminazione di (6,2) produrrebbe un grafo che non ha un hampath 1-4, quindi non viene tolto e si passa a un altro arco.

Correlazione tra le complessità dei problemi decisionali e di ricerca

Esempio: Eseguendo l'algoritmo di decisione verifichiamo se un grafo diretto ha un cammino hamiltoniano tra due vertici dati.

Se la risposta è no il processo termina, altrimenti rimuoviamo un arco se il grafo ottenuto dopo la rimozione ha ancora un cammino hamiltoniano, verificandolo utilizzando l'algoritmo pr il problema di decisione. Al termine del processo resta giusto il cammino hamiltoniano.

Se $t_{DHam}(n)$ è la funzione che esprime il tempo di esecuzione nel caso peggiore per il problema decisionale, possiamo dire che questo algoritmo è applicato al più tante volte quanti sono gli archi del grafo. Quindi se $t_{RHam}(n)$ è il tempo di esecuzione nel caso peggiore dell'algoritmo di ricerca, possiamo dire che $t_{RHam}(n) = O(|E| t_{DHam}(n))$.

SAT

SAT ={ φ | φ è una formula booleana soddisfacibile}.

Esempio di formula booleana :

$$\varphi = (x1 \vee \neg x2) \wedge (x2 \vee x3 \vee \neg x1) \wedge (x4 \vee \neg x3)$$

SAT

Problema SAT

input: φ, dove φ una formula booleana

Problema di decisione: esiste un assegnamento di valori di verità che soddisfa φ?

Risposta: Sì o no

Però la soddisfacibilità di una formula booleana è un tipico problema di ricerca (search): cerchiamo un assegnamento di valori di verità che renda vera la formula, tra tutti quelli possibili.

RSAT

Esempio di formula booleana :

$$\varphi = (x1 \vee \neg x2) \wedge (x2 \vee x3 \vee \neg x1) \wedge (x4 \vee \neg x3)$$

Problema RSAT

input: φ, dove φ una formula booleana

Problema di ricerca: trovare un assegnamento di valori di verità, se esiste, che soddisfa φ.

Risposta: un assegnamento che soddisfa φ o "φ non è soddisfacibile"

Decisione vs ricerca per SAT

Supponiamo di disporre di un algoritmo per SAT, M_{SAT} tale che:

M_{SAT}

input: φ, dove φ una formula booleana

risposta sì se φ è soddisfacibile

altrimenti risposta no.

Possiamo usarlo come "subroutine" in un algoritmo che calcola un assegnamento che soddisfa φ.

IDEA:

Si esegue M_{SAT} su φ se la risposta è no, si termina con risposta "φ non è soddisfacibile", altrimenti

si assegna 0 alla prima variabile e si esegue M_{SAT} sulla formula ottenuta φ_0, se φ_0 non è soddisfacibile si pone la prima variabile a 1,

in entrambi i casi

si prosegue nello stesso modo per la seconda variabile, e così via fino ad esaurimento delle variabili.

Si esegue M_{SAT} su φ se la risposta è no, si termina con risposta "φ non è soddisfacibile", altrimenti

si assegna 0 alla prima variabile e si esegue M_{SAT} sulla formula ottenuta φ_1, se φ_1 non è soddisfacibile si pone la prima variabile a 1,

in entrambi i casi si prosegue nello stesso modo per la seconda variabile, e così via fino ad esaurimento delle variabili.

Esempio:

$\varphi(x1, x2, x3, x4) = (x1 \vee \neg x2) \wedge (x2 \vee x3 \vee \neg x1) \wedge (x4 \vee \neg x3)$

Si esegue M_{SAT} su

$\varphi(x1, x2, x3, x4) = (x1 \vee \neg x2) \wedge (x2 \vee x3 \vee \neg x1) \wedge (x4 \vee \neg x3)$,

poichè è soddisfacibile, si costruisce

$\varphi(0, x2, x3, x4) = \varphi_0(x2, x3, x4) = (0 \vee \neg x2) \wedge (x2 \vee x3 \vee 1) \wedge (x4 \vee \neg x3)$

e si esegue M_{SAT} su φ_0

se la risposta è sì, allora si prosegue assegnando 0 a x2,

se la risposta è no, allora si considera

$\varphi(1, x2, x3, x4) = \varphi_1(x2, x3, x4) = (1 \vee \neg x2) \wedge (x2 \vee x3 \vee 0) \wedge (x4 \vee \neg x3)$ perchè visto che φ è soddisfacibile uno dei due valori è quello giusto!

Decisione vs ricerca

Risolvere un problema di ricerca comporta la soluzione dell'associato problema di decisione.

Potremmo dire che la decisione si riduce alla ricerca .

Che dire del contrario??

Nei casi che abbiamo visto le soluzioni ai problemi di decisione hanno fornito le soluzioni ai problemi di ricerca.

Uno dei motivi per cui si considerano problemi di decisione è che spesso le soluzioni algoritmiche per questi si possono utilizzare per i problemi di ricerca associati.

Algoritmo di ricerca per SAT: tempo di esecuzione

Si esegue M_{SAT} su φ se la risposta è no, si termina con risposta "φ non è soddisfacibile", altrimenti

si assegna 0 alla prima variabile e si esegue M_{SAT} sulla formula ottenuta φ_1, se φ_1 non è soddisfacibile si pone la prima variabile a 1,

in entrambi i casi si prosegue nello stesso modo per la seconda variabile, e così via fino ad esaurimento delle variabili.

Detta n la lunghezza della formula, sia $t_{DSAT}(n)$ la funzione che esprime il tempo di esecuzione nel caso peggiore per il problema decisionale.

Possiamo osservare che questo algoritmo è applicato tante volte quanto è il numero delle variabili nella formula in input.

Quindi il tempo di esecuzione nel caso peggiore dell'algoritmo di ricerca è anche in questo caso polinomialmente correlato con il tempo di esecuzione nel caso peggiore dell'algoritmo di decisione.

Complessità problemi decisionali e di ricerca

L'algoritmo per determinare un cammino hamiltoniano e quello per determinare un assegnamento che soddisfa una formula booleana ha una complessità che dipende polinomialmente da quello per la ricerca.

D'altro canto se un problema decisionale non ammette un algoritmo che lo risolve non ci potrà essere un algoritmo che risolve il problema della ricerca associato.

Problemi decisionali

Lo studio delle classi di complessità dei problemi è semplificato dal ridursi a considerare problemi decisionali e i risultati sono spesso applicabili ai problemi di ricerca associati.

Queste osservazioni giustificano la concentrazione su problemi di decisione.

Inoltre vedendo i problemi decisionali come linguaggi possiamo sfruttare i risultati e metodi della teoria dei linguaggi formali.