

Lez 1

Calcolabilità: capire quali problemi possono essere risolti automaticamente.

Complessità: capire quali dei problemi che possono essere risolti, possono essere risolti per davvero.

"Quanto fa $5+3$ " potrebbe sembrare un problema ma in realtà è un **istanza** di un problema. Il problema a cui corrisponde l'istanza è:

PROBLEMA SOMMA: dati due numeri naturali n e K , calcolare il valore della somma di n con K (ossia, $n+K$).

Un **problema** è la descrizione di un insieme di parametri, chiamati **dati**, collegati da un certo insieme di relazioni, associate alla richiesta di derivare da essi un altro insieme di parametri, che costituiscono la soluzione.

Un' **istanza** di un problema è un particolare insieme di valori associati ai dati.

Per trovare la soluzione di talune istanze di taluni problemi posso sfruttare le caratteristiche di quelle istanze ma a volte non è così semplice altre volte impossibile.

Quando l'istanza di un problema non ha soluzione diciamo che essa è una **istanza negativa**.

Risolvere un problema significa individuare un metodo che sappia trovare la soluzione di qualunque istanza positiva del problema e che sappia riconoscere se un'istanza è negativa. Ossia trovare un procedimento che, data una qualunque istanza del problema, indichi la sequenza di azioni che devono essere eseguite per trovare la soluzione di quell'istanza o per concludere che una soluzione non c'è.

Un **procedimento** è la descrizione di un insieme di azioni unite alla specifica dell'ordine con il quale le azioni devono essere eseguite.

Le azioni indicate in un procedimento devono essere semplici, cioè, che possono essere eseguite con facilità.

A ciascuna azione viene dato il nome di **istruzione**, queste istruzioni devono essere **elementari**, cioè, azioni che possono essere eseguite con facilità.

Turing osservò che qualunque istruzione per poter essere definita elementare deve avere le seguenti caratteristiche:

- deve essere scelta un insieme di "poche" istruzioni disponibili.
- deve scegliere l'azione da eseguire all'interno di un insieme di "poche" azioni possibili.
- deve poter essere eseguita ricordando una quantità limitata di dati, ossia, utilizzando una quantità limitata di memoria.

Sono quindi operazioni che possono essere eseguite a mente.

Considerando il PROBLEMA SOMMA con $n = 37895$ e $k = 441238$ nessuno di noi lo fa a mente perché la nostra memoria è limitata. Abbiamo memorizzato la tabella che ci permette di calcolare a mente somma di qualunque coppia di numeri di una cifra ciascuno.

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Se avessimo una tabella sufficientemente grande:

+	0	1	2	...	37895	...	1000000
0	0	1	2	...	37895	...	1000000
1	1	2	3	...	37896	...	1000001
2	2	3	4	...	37897	...	1000002
...
441238	441238	441239	441240	...	479133	...	1441238
...
1000000	1000000	1000001	1000002	...	1037895	...	2000000

NON funziona in questo modo!

Occorre infatti indicare un procedimento che sappia addizionare qualunque coppia di numeri, per quanto grandi essi siano, perciò non possiamo avere una tabella infinita.

Bisogna utilizzare un procedimento che:

- utilizza un numero limitato di operazioni elementari. (coppie da una cifra)
- in cui ogni operazione elementare utilizza una quantità limitata di dati, (due cifre e riporto)

La somma di due numeri viene quindi fatta in colonna.

Procedimento:

- 1) posizionati sulla coppia di cifre più 2 destra e ponì $r=0$.
- 2) fino a quando leggi una coppia di cifre, esegui la somma della coppia, aggiungi r a tale valore e scrivi una cifra nel risultato calcolando il nuovo r , poi spostati a sinistra.
 - se $r=0$ le due cifre sono 0 e 0, allora scrivi 0 ponì $r=0$, e spostati una posizione a sinistra
 - :
 - :
 - :
- 3) fino a quando leggi una sola cifra aggiungi r ad essa e scrivi una cifra del risultato calcolando il nuovo valore di r , poi spostati a sinistra.
 - se $r=0$ e cifra è 0, scrivi 0, ponì $r=0$, spostati di una posizione a sinistra
 - :
 - :
 - :
- 4) se entrambe le cifre sono terminate, calcola eventuale ultima cifra e termina.
 - se $r=0$ e le cifre di entrambi i numeri sono terminate, allora termina
 - se $r=1$ e le cifre di entrambi i numeri sono terminate, allora scrivi 1 e termina.

E' quindi una sequenza di: "se sono vere certe **condizioni** allora esegui **azioni**"
ad ogni coppia (certe condizioni, queste azioni) corrisponde un istruzione.
dove "certe condizioni" è ciò che viene letto e il valore del riporto,
e "queste azioni" è ciò che viene scritto, la modifica del valore del riporto, e lo

spostamento, o in alcuni casi è l'indicazione che la somma è stata completa.

Il numero di istruzioni, azioni e la quantità di memoria necessaria sono costanti: non dipendono da quello che chiameremo input.

Le istruzioni ti dicono, per ogni condizione possibile, esattamente quali azioni devi eseguire in quelle condizioni. Questo significa che l'insieme di istruzioni è non ambiguo: non può contenere due (o più) istruzioni che, a partire dalle stesse condizioni, ti indica diverse azioni da eseguire.

L'ordine in cui eseguire le istruzioni è indicato implicitamente nel meccanismo stesso del "se ... allora ...".

Per ottenere il risultato devi eseguire le istruzioni, sono una sorta di ordini.

Questa idea di istruzione, nata dall'analisi di Turing, è alla base di molti linguaggi di programmazione detti imperativi.

Informalmente:

Risolvere automaticamente un problema significa progettare un procedimento che risolve tutte le istanze di quel problema e che può essere eseguito da un automa, ossia, da un esecutore che può non avere alcuna idea del problema né del significato delle istruzioni contenute nel procedimento.

Le istruzioni verranno scritte in questo modo:

$\langle q_0, (\square, 5), 5, q_0, sx \rangle$

q_0 indica lo stato interiore, \square indica che non viene letto nulla.

Se mi trovo nello stato q_0 e leggo \square sul primo nastro, 5 sul secondo, allora scrivo 5 sul nastro di output, vado nello stato q_0 e sposta le testine a sinistra.

Non appena viene scritto qualcosa sui nastri, dipendentemente dallo stato interiore dell'automa e da quello che viene letto, l'automa inizia a computare ossia a eseguire le quintuple del procedimento.

In una macchina di Turing occorre descrivere cosa viene scritto su ogni nastro.

$$\langle q_0, (\square, 5), 5, q_0, \text{sx} \rangle \implies \langle q_0, (\square, 5, \square), (\square, 5, 5), q_0, \text{sx} \rangle$$

Queste sono quintuple e i stati intermedi si chiamano **stati interni**, l'esecuzione delle quintuple su un insieme fissato di dati si chiama **computazione**.

Questa è una descrizione informale della macchina di Turing che è la descrizione di un procedimento di risoluzione di un problema espresso nel ^(m)linguaggio definito da Alan Turing.

Linguaggio che costituisce un modello di calcolo: ^(M)**Macchina di Turing**.

Definizioni

Un **problema** è definito da un insieme di istanze, per ciascuna delle quali è necessario trovare una soluzione che rispetti i vincoli del problema. Una **istanza** del problema è un insieme di dati. Una soluzione di una istanza del problema è un insieme, correlati ai dati, che rispettano l'insieme dei vincoli del problema.

Un **algoritmo** è la descrizione di una sequenza di passi elementari che permettono ad un qualche esecutore di calcolare una soluzione di un problema a partire da una qualsiasi sua istanza.

Sia Σ un alfabeto finito e Q un insieme finito di stati nel quale di distinguiamo uno stato iniziale q_0 ed un insieme di stati finali Q_f . Una **Macchina di Turing** T sull'alfabeto Σ e sull'insieme di stati Q è un dispositivo di calcolo dotato di

- una unità di controllo che, ad ogni istante della computazione, può trovarsi in uno qualsiasi degli stati in Q ,
- d: un nastro di lettura/scrittura, di lunghezza infinita, suddiviso in celle indicizzate contenenti, ciascuna, un simbolo in Σ oppure il carattere \square (cella vuota) e alle quali si può accedere sequenzialmente,

- di una testina di lettura/scrittura che, ad ogni istante della computazione, è posizionata su una cella del nastro,
- un programma, ossia, un insieme P di quintuple del tipo $\langle q_1, \alpha_1, \beta_1, q_2, m \rangle$ in cui $q_1 \in Q - Q_f$, $q_2 \in Q$, $\alpha_1, \beta_1 \in \Sigma$ e $m \in \{sx, dx, f\}$ e che ha il seguente significato: se la testina legge nella cella quale si trova il simbolo α_1 e l'unità di controllo si trova nello stato q_1 , allora la testina scrive il simbolo β_1 nella cella quale si trova, poi lo stato interno dell'unità di controllo diventa q_2 , infine la testina si sposta di una cella nella direzione specificata da m .
Data una parola x (ossia, una sequenza finita di elementi di Σ), informalmente, una **computazione** $T(x)$ della macchina T sull'input x è l'applicazione delle quintuple in P ad x partendo dal primo simbolo di x con l'unità di controllo che si trova nello stato iniziale q_0 di T .

L'esecuzione di una singola quintupla di una macchina di Turing è un **passo elementare** di calcolo.

Lec 2

Una macchina di Turing ad un nastro è:

una unità di controllo che, ad ogni istante, può trovarsi in uno stato interno appartenente ad un certo insieme Q che contiene, fra gli altri, lo stato particolare q_0 che fa partire la computazione e un sottoinsieme Q_F di stati che fanno terminare la computazione;

un nastro suddiviso in un numero infinito di celle, ciascuna delle quali, ad ogni istante può essere vuota o contenere un simbolo appartenente ad un alfabeto Σ , e sul quale nastro si muove una testina di lettura/scrittura;

ad ogni istante, dipendentemente dallo stato interno e da ciò che è letto dalla testina, viene eseguita una quintupla scelta in un insieme P di quintuple.

All'inizio l'unità di controllo si trova nello stato q_0 , la testina legge il simbolo contenuto nella cella che sta scandendo e la macchina cerca una quintupla i cui primi due elementi sono q_0 e simbolo letto (può essere anche blank \square) e, se trova una tale quintupla, la esegue, se non la trova non compie alcuna azione e la computazione termina.

Eseguire una quintupla significa eseguire le tre azioni in essa indicate:
sovrascrivere il simbolo nella cella scandita dalla testina con il simbolo indicato nella quintupla;
cambiare (eventualmente) stato interno;
muovere (eventualmente) la testina.

Eseguita una quintupla se ne cerca un'altra da eseguire fino a quando nessuna quintupla può essere eseguita.

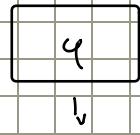
es: macchina di Turing ad un nastro T parità definita su $\Sigma = \{0, 1, p, d\}$ e

$Q = \{q_0, q_p, q_d, q_f\}$ e $P = \{< q_0, 0, \square, q_p, \text{destra} >, < q_0, 1, \square, q_d, \text{destra} >,$

$< q_p, 0, \square, q_p, \text{destra} >, < q_d, 1, \square, q_d, \text{destra} >,$

$< q_p, 1, \square, q_d, \text{destra} >, < q_d, 1, \square, q_p, \text{destra} >,$

$< q_p, \square, p, q_f, \text{destra} >, < q_d, \square, d, q_f, \text{destra} >\}$



$\boxed{0} \ 1 \ \boxed{0} \ 1 \ 1 \ 1 \ \boxed{0}$

La macchina Tpari scandisce la sequenza di caratteri sul nastro cancellandoli via via che vengono scanditi, e verificando se tale sequenza contiene un numero pari o dispari di "1": al termine della scansione scrive "p" o "d".

Ma sul nastro possiamo scrivere ciò che vogliamo come "polo" ma P non ha nessuna quintupla che inizia con la coppia (q_0, p) perciò nessuna quintupla può essere eseguita.

Quindi:

una macchina di Turing (ad un nastro) è una quintupla $\langle \Sigma, Q, q_0, Q_f, P \rangle$.

Σ = un insieme finito di caratteri chiamato **alfabeto**.

Q = un insieme finito di **stati interni**.

q_0 = uno stato interno particolare chiamato **stato iniziale**.

Q_f = un sottoinsieme di Q di **stati finali**.

P = un insieme $P \subseteq Q \times \Sigma \times \Sigma \times \{sx, dx, f\}$ di **quintuple** non ambiguo, in effetti: P è una funzione $P: Q \times \Sigma \rightarrow \Sigma \times Q \times \{sx, dx, f\}$.

Se la macchina ha K nastri le quintuple nell'insieme P hanno la forma:

$\langle q_1, (z_1, \dots, z_K), (b_1, \dots, b_K), q_2, (m_1, \dots, m_K) \rangle$

dove "z" sono i caratteri letti, b quelli che devono essere scritti, m sono i movimenti che le testine devono eseguire, "1, ..., K" indicano quale nastro.

Oss: Per capire quale sia il numero di nastri di una macchina di Turing è sufficiente osservare le quintuple contenute in P .

In generale, quindi, una macchina di Turing è una quintupla $\langle \Sigma, Q, q_0, Q_f, P \rangle$, è la descrizione di un procedimento di calcolo ossia un **algoritmo descritto utilizzando le regole** introdotte da Alan Turing.

Le regole introdotte da Alan Turing per descrivere procedimenti di calcolo costituiscono un **modello di calcolo** che prende il nome di Macchina di Turing.

Il modello di calcolo Macchina di Turing richiede che in ogni macchina l'insieme

degli stati e l'alfabeto abbiano cardinalità finita (lo stesso vale per il numero di nastri) (di conseguenza anche P è finito). Se fossero infinite non potremmo costruire la macchina. Non possiamo scrivere quindi $\forall n \in \mathbb{N}$.

E' necessario che il numero di stati, numero di simboli dell'alfabeto, numero di quintuple e numero di nastri siano costanti.

Dato un alfabeto finito Σ , con il simbolo Σ^* indichiamo l'insieme delle parole, ossia, l'insieme delle sequenze costituite da un numero finito (eventualmente nullo) di elementi di Σ :

$$\Sigma^* = \{ \langle x_1, x_2, \dots, x_n \rangle : n \in \mathbb{N} \wedge \forall i \leq n [x_i \in \Sigma] \}$$

Uno stato globale SG di una macchina di Turing T è una "fotografia" della macchina ad un certo istante: SG contiene una descrizione della porzione non blank del nastro di T , della posizione della testina e dello stato interno.

Esiste una transizione da uno stato globale SG_1 ad uno SG_2 quando esiste $\langle q_1, \sigma_1, o_1, q_2, m \rangle \in P$ tale che q_1 e σ_1 sono, rispettivamente, lo stato interno e il simbolo letto dalla testina nello stato SG_1 , q_2 e σ_2 sono rispettivamente, lo stato interno e il simbolo che sostituisce σ_1 nello stato SG_2 e la testina si è spostata in accordo ad m nel passaggio da SG_1 a SG_2 .

Una computazione (deterministica) $T(x)$ della macchina T su input

$x = \langle x_1, x_2, \dots, x_n \rangle \in \Sigma^*$ è una successione $SG_0(x), SG_1(x), \dots, SG_K(x), \dots$ t.c.:

- SG_0 è lo stato globale iniziale di $T(x)$, in cui la testina è posizionata sulla cella del nastro sulla quale è scritto x , e lo stato interno di SG_0 è lo stato iniziale di T ;

- se per qualche $K \geq 0$ lo stato interno di $SG_K(x)$ è uno stato finale, allora, $\forall i > K, SG_i$ non è definito; in questo caso, SG_K è uno stato globale finale di $T(x)$
- $\forall i \geq 0$ tale che $SG_i(x)$ non è uno stato globale finale, esiste una transizione da $SG_i(x)$ a $SG_{i+1}(x)$.

Se una computazione $T(x)$ contiene uno stato globale finale, allora la computazione termina, ossia, una volta raggiunto lo stato finale globale, T non esegue più alcuna istruzione.

Macchine **trasduttore** hanno come compito di **calcolare** il valore di una funzione, dispongono di un nastro di output sul quale scrivono il valore della funzione calcolata, ha un solo stato finale con il quale termina la computazione: q_f

Macchine **riconoscitore** hanno come compito **decidere** se l'input appartiene all'insieme, non dispongono di un nastro di output, il valore calcolato (0 o 1) viene memorizzato nello stato interno con il quale la macchina termina la computazione: q_A se il valore appartiene all'insieme, cioè stato di accettazione, q_R altrimenti, cioè stato di rigetto

Si indica con $O_T(x)$ l'**esito della computazione** $T(x)$ della macchina T su input x .

Sia T una macchina di Turing di tipo trasduttore, la funzione $O_T: \Sigma^* \rightarrow \Sigma^*$ è definita per: soli $x \in \Sigma^*$ tali che $T(x)$ termina e, per tali x , il valore $O_T(x)$ è la parola calcolata da tale computazione (scritto sul nastro di output di T).

Sia T una macchina di Turing di tipo riconoscitore, la funzione $O_T: \Sigma^* \rightarrow \{q_A, q_R\}$ è definita per: soli $x \in \Sigma^*$ tali che $T(x)$ termina e, per tali x , il valore $O_T(x)$ è lo stato finale di terminazione della computazione.

Lec 3

In una macchina di Turing a testine **solidali**, in ogni istruzione, le celle dei nastri scandite dalle testine di lettura/scrittura hanno il medesimo indirizzo: se dopo un certo numero di passi le testine sono posizionate sulle celle di indirizzo h , al passo successivo tutte le testine saranno posizionate sulle celle $h, h+1, h-1$.

Una quintupla ha la forma:

$$\langle q_i, \bar{s}_1, \bar{s}_2, q_j, \text{mov} \rangle \quad \bar{s}_1 = (s_{1,1}, \dots, s_{1,k}), \bar{s}_2 = (s_{2,1}, \dots, s_{2,k}) \text{ e } \text{move} \{s_x, d_x, f\}$$

In una macchina di Turing a testine **indipendenti**, in seguito all'esecuzione di una quintupla le testine si muovono indipendentemente le une dalle altre. Dopo un certo numero di passi la posizione di una testina non ha alcuna relazione con la posizione delle altre.

Una quintupla ha la forma:

$$\langle q_i, \bar{s}_1, \bar{s}_2, q_j, \overline{\text{mov}} \rangle \quad \bar{s}_1 = (s_{1,1}, \dots, s_{1,k}), \bar{s}_2 = (s_{2,1}, \dots, s_{2,k}) \text{ e } \overline{\text{mov}} = (\text{mov}_1, \dots, \text{mov}_k)$$

Una macchina a testine **indipendenti** può essere **simulata** da una macchina a testine **solidali**.

Sia T_2 la macchina a testine indipendenti a due nastri e T_3 la macchina a testine solidali con 3 nastri: i primi due sono una copia di T_2 , mentre il terzo nastro contiene un solo carattere non appartenente a Σ , " $*$ ", nella cella di indirizzo 0, e viene utilizzato per segnalare alle testine su quali celle posizionarsi.

Assumiamo che all'inizio della computazione le testine di T_2 e T_3 scandiscano le celle di indirizzo 0 dei rispettivi nastri.

Assumiamo che T_2 non scriva mai $\#$.

La macchina T_3 simula T_2 mediante una sequenza di shift dei contenuti dei primi due nastri in modo tale che, ad ogni passo, la testina di T_3 sia sempre posizionata sulle celle il cui indirizzo coincide con quello del terzo nastro in cui è scritto il carattere " $*$ ".

Sia $\langle q_1, (s_{1,1}, s_{1,2}), (s_{2,1}, s_{2,2}), q_2, (\text{mov}_1, \text{mov}_2) \rangle$ una quintupla di T_2 .

Se $\text{mov}_1 = \text{mov}_2$ allora la coppia di quintuple di T_3 :

$\langle q_1, (s_1, s_{1z}, *), (s_2, s_{2z}, \square), q_2^+, \text{mov} \rangle$

$\langle q_2^+, (x, y, \square), (x, y, *), q_2^-, f \rangle, \forall x, y \in \Sigma \cup \{\square\}$ q_2^+ non appartiene a T_2

Se $\text{mov}_1 \neq \text{mov}_2$, allora dopo aver scritto i caratteri " s_1 " e " s_{2z} ", eseguiamo uno shift dei primi due nastri in modo tale che i caratteri che devono essere letti al passo successivo si trovino nelle celle aventi lo stesso indirizzo della cella del terzo nastro "*".

Supponiamo $\text{mov}_1 = dx, \text{mov}_2 = sx$ allora le quintuple di T_3 sono:

$\langle q_1, (s_1, s_{1z}, *), (s_2, s_{2z}, *), q_{1,0}^{ds}(q_2), dx \rangle$ scrive.

si sposta sull'ultimo carattere non \square del nastro.

$\langle q_{1,0}^{ds}(q_2), (x, y, z), (x, y, z), q_{1,0}^{ds}(q_2), dx \rangle \forall x \in \Sigma, \forall y \in \Sigma \cup \{\square\} \forall z \in \{*, \square\}$

si prepara a spostare verso sinistra

$\langle q_{1,0}^{ds}(q_2), (\square, y, z), (\square, y, z), q_{1,s}^{ds}(q_2), sx \rangle \forall y \in \Sigma \cup \{\square\} \forall z \in \{*, \square\}$

sposta ciascun carattere a sinistra

$\langle q_{1,0}^{ds}(q_2, a), (x, y, z), (a, y, z), q_{1,s}^{ds}(q_2, x), sx \rangle \forall a \in \Sigma \cup \{\square\} \forall z \in \{*, \square\} \forall x, y \in \Sigma \cup \{\square\}$

quando si raggiunge la prima posizione si prepara lo spostamento sul $x \neq \square \vee y \neq \square$

$\langle q_{1,0}^{ds}(q_2, a), (\square, \square, z), (a, \square, z), q_{2,0}^{ds}(q_2, \square), dx \rangle \forall a \in \Sigma \cup \{\square\} \forall z \in \{*, \square\}$

sposta tutto verso destra

$\langle q_{2,0}^{ds}(q_2, b), (x, y, z), (x, b, z), q_{2,0}^{ds}(q_2, y), dx \rangle \forall b \in \Sigma \cup \{\square\} \forall x \in \Sigma \cup \{\square\} \forall z \in \{*, \square\}$

finito lo spostamento si prepara a tornare su

$\langle q_{2,0}^{ds}(q_2, b), (x, \square, z), (x, b, z), q_{2,s}^{ds}(q_2), f \rangle \forall b \in \Sigma \cup \{\square\} \forall x \in \Sigma \cup \{\square\} \forall z \in \{*, \square\}$

torna su

$\langle q_{2,s}^{ds}(q_2), (x, y, \square), (x, y, \square), q_{2,s}^{ds}(q_2), sx \rangle \forall x, y \in \Sigma \cup \{\square\}$

entra nello stato q_2 la quintupla presa in considerazione

$\langle q_{2,s}^{ds}(q_2), (x, y, *), (x, y, *), q_2^-, f \rangle \forall x, y \in \Sigma \cup \{\square\}$ è terminata

Una macchina a K nastri può essere simulata da una macchina ad un nastro.

Sia T_K una macchina a K nastri a testine solidali e T_1 la macchina ad un nastro che utilizza lo stesso alfabeto Σ utilizzato da T_K ed il cui insieme degli stati è $Q \times \Sigma^*$.

Inizialmente, l'input $x = (x_1, x_2, \dots, x_{1K})(x_2, \dots, x_{2K}) \dots (x_n, \dots, x_{nK})$ di T_K è scritto sull'unico nastro di T_1 , a partire dalla cella 1, come concatenazione di tutti i simboli

di x , nella forma seguente $x_1, x_2, \dots, x_{i_k}, x_{i_k+1}, x_{i_k+2}, \dots, x_{i_k+m}, \dots, x_n$.

Sia $\langle q_1, (s_1, s_2, \dots, s_{i_k}), (s_{i_k+1}, s_{i_k+2}, \dots, s_{i_k+m}), q_2, m \rangle$ una qualsiasi quintupla di T_k .

Per T_k è sufficiente una singola operazione di lettura per poter garantire la corretta esecuzione della quintupla, T_i deve eseguire K operazioni di lettura consecutive in quanto la quintupla può essere eseguita solo se viene letto s_1 , ed è seguito da s_2, \dots, s_{i_k} . Verranno eseguite queste quintuple:

$\langle q_1, s_1, s_2, q(q_1, s_1), dx \rangle$

$\langle q(q_1, s_1), s_2, s_3, q(q_1, s_1, s_2), dx \rangle$

...

$\langle q(q_1, s_1, s_2, \dots, s_{i_k-1}), s_{i_k}, s_{i_k+1}, q(q_1, s_1, s_2, \dots, s_{i_k}), sx \rangle$

Ora T_i ha verificato che la quintupla può essere eseguita, deve riportare la testina a sx di K celle:

$\langle q(q_1, s_1, \dots, s_{i_k}), s_{i_k+1}, s_{i_k+2}, q(q_1, s_1, \dots, s_{i_k}, K-2), sx \rangle$

$\langle q(q_1, s_1, \dots, s_{i_k}, i), s_i, s_{i+1}, q(q_1, s_1, \dots, s_{i_k}, i+1), sx \rangle \quad \forall i=2, \dots, K-2$

La testina ora è posizionata sul carattere corrispondente al carattere scritto sul 1° nastro di T_k (s_1) e può procedere all'esecuzione

$\langle q(q_1, s_1, \dots, s_{i_k}, 1), s_1, s_2, q^{wr}(q_1, s_1, \dots, s_{i_k}, 2), dx \rangle$

$\langle q^{wr}(q_1, s_1, \dots, s_{i_k}, i), s_i, s_{i+1}, q^{wr}(q_1, s_1, \dots, s_{i_k}, i+1), dx \rangle \quad \forall i=2, \dots, K$

$\langle q^{wr}(q_1, s_1, \dots, s_{i_k}, K), s_{i_k}, s_{i_k+1}, q', m' \rangle$

T_i ha eseguito la prima parte della quintupla, resta il cambio di stato interno e il movimento della testina:

Se $m = dx$ allora $q' = q_2$ e $m' = dx$

Se $m = f$ allora $q' = q^s(q_2, K-1)$ e $m' = sx$ per poi eseguire le quintuple:

$$\langle q^s(q_2, i), x, x, q^s(q_2, i-1), s_x \rangle \quad \forall x \in \Sigma \cup \{\square\} \quad \forall i=2, \dots, K-1$$

$$\langle q^s(q_2, 1), x, x, q_2, f \rangle \quad \forall x \in \Sigma \cup \{\square\}$$

Se $m = s_x$ allora $q' = q^s(q_2, 2K-1)$ e $m' = s_x$ per poi eseguire le quintuple:

$$\langle q^s(q_2, i), x, x, q^s(q_2, i-1), s_x \rangle \quad \forall x \in \Sigma \cup \{\square\} \quad \forall i=2, \dots, 2K-1$$

$$\langle q^s(q_{2,1}), x, x, q_2, f \rangle \quad \forall x \in \Sigma \cup \{\square\}$$

Q di T_i ha cardinalità maggiore rispetto a Q di T_K e Q_F coincidono.

Una macchina definita su un alfabeto Σ t.c. $|\Sigma| > 2$ può essere simulata da una macchina definita sull'alfabeto $\{0, 1\}$.

Sia T una macchina che utilizza un alfabeto la cui cardinalità è > 2 e sia $T_{0,1}$ la macchina che utilizza l'alfabeto $\{0, 1\}$.

Sia $\Sigma = \{\sigma_1, \dots, \sigma_n\}$ rappresenteremo ciascun simbolo di Σ mediante codifica binaria b che utilizza $K = \lceil \log_2 n \rceil$ cifre.

$$\forall \sigma \in \Sigma \quad b(\sigma) = (b_1(\sigma), b_2(\sigma), \dots, b_K(\sigma)). \quad b(\square): \text{una sequenza di } K \text{ } \square.$$

Ogni volta che T scrive (legge) \square , $T_{0,1}$ dovrà scrivere (leggere) $K \square$. $T_{0,1}$ utilizza un insieme di stati $Q_{0,1}$ molto più ampio di Q (si evincerà dall'esame delle quintuple di $T_{0,1}$).

Per costruire $P_{0,1}$: $\forall p = \langle q, \sigma, \sigma', q', m \rangle \in P \quad P_{0,1}(p) = P_1(p) \cup P_2(p)$ dove $P_1(p)$ verificano l'eseguibilità di p e sovrascrivono i bit e le quintuple $P_2(p)$ simulano il movimento della testina corrispondente a m e portano la macchina nello stato q' .

$$P_1(p): \langle q, b_1(\sigma), b_1(\sigma'), q(b_1(\sigma)), d \rangle$$

...

$$\langle q(b_1(\sigma), \dots, b_{K-1}(\sigma)), b_K(\sigma), b_K(\sigma'), q(\sigma, K-1), s \rangle \quad \text{legge le } K \text{ cifre}$$

$$\langle q(\sigma, K-1), b_{K-1}(\sigma), b_{K-1}(\sigma'), q(\sigma, K-2), s \rangle \quad \text{inizializza a scrivere, a ritirare}$$

$$\langle q(\sigma, K-i), b_{K-i}(\sigma), b_{K-i}(\sigma'), q(\sigma, K-i-1), s \rangle$$

$$\langle q(\sigma, 1), b_1(\sigma), b_1(\sigma'), q^{mr}(\sigma, K), f \rangle$$

$P_2(p)$ dipendono dal valore di m :

Se f : $\langle q^{mv}(\sigma, K), b, (\sigma'), b, (\sigma'), q', f \rangle$

Se do s: $\langle q^{mv}(\sigma, K), b, (\sigma'), b, (\sigma'), q^{mv}(\sigma, K-1), s \rangle$

$\langle q^{mv}(\sigma, K-1), a, a, q^{mv}(\sigma, K-2), s \rangle \quad \forall a \in \{0, 1, \square\}$

...

$\langle q^{mv}(\sigma, 1), a, a, q', s \rangle \quad \forall a \in \{0, 1, \square\}$

$\forall x \in \Sigma^*$, l'esito della computazione $T(x)$ coincide con l'esito della computazione $T_a(b(x))$

Lec 4

Una macchina di Turing T , definita sull'alfabeto Σ e sull'insieme Q di stati, un elemento di P è una quintupla $\langle q_1, s_1, s_2, q_2, m \rangle$ il cui scopo è indicare alla macchina quale azione intraprendere quando si trova nello stato q_1 e legge s_1 . Una quintupla dunque non è altro che una istruzione. È sufficiente avere P per ricavare Σ e Q , dobbiamo però conoscere anche quale sia lo stato iniziale e quali siano gli stati finali.

Possiamo dire che P è una corrispondenza che associa ad elementi dell'insieme $Q \times \Sigma$ elementi dell'insieme $\Sigma \times Q \times \{s, d, f\}$.

Totalità: se

$$\forall (q, s) \in (Q - Q_f) \times \Sigma \quad \exists (s_2, q_2, m) \in \Sigma \times Q \times \{s, d, f\} : \langle q, s, s_2, q_2, m \rangle \in P$$

La totalità di P è connessa alle verifiche delle precondizioni.

La corrispondenza P può non essere totale: considerando P come un insieme di quintuple, esso può non contenere le quintupla che iniziano con coppie di simboli (stato, carattere) che si riferiscono a configurazioni del nastro che non rispettano le precondizioni previste.

Sia T una macchina di tipo riconoscitore ad un nastro, e supponiamo si trovi nello stato q , legga s e che nell'insieme P non esista nessuna quintupla che inizi con la coppia (q, s) : la macchina T non riuscirà a raggiungere lo stato di accettazione, possiamo quindi aggiungere la quintupla $\langle q, s, s, q_f, f \rangle$

Sia T una macchina di tipo trasduttore ad un nastro, che, ad un certo passo della computazione iniziata con input $x \in \Sigma^*$, si trovi nello stato q e legga s , nel caso in cui nell'insieme P delle sue quintuple non esista alcuna quintupla che inizia con la coppia (q, s) : in tal caso T non è in grado di completare il suo compito, non è in grado di produrre l'output corrispondente all'input x . Possiamo affermare che la computazione $T(x)$ non produce alcun output. Pertanto, possiamo considerare una nuova macchina T' dove $P' = P \cup \{ \langle q, (s, x), (s, x), q, f \rangle : x \in \Sigma^* \}$.

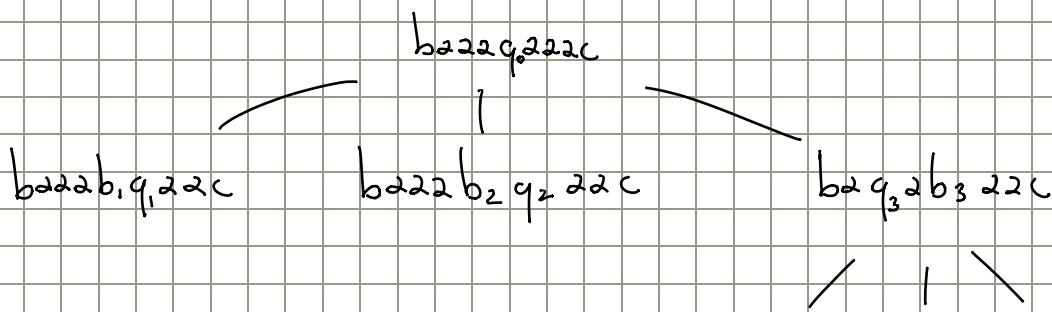
il cui comportamento coincide con il comportamento di T:

$T'(x) = T(x) \quad \forall x \in \Sigma^*$: $T(x)$ calcola un output e $T'(x)$ non termina
 $\forall x \in \Sigma^*$: $T(x)$ non calcola un output.

P non contiene coppie di quintuple che hanno gli stessi due elementi iniziali perciò può essere vista anche come una **funzione**. Il comportamento di queste macchine è totalmente determinato. Le macchine di Turing di questo genere vengono dette **deterministiche**.

E' stato definito anche un modello di macchina di Turing **non deterministica**, il cui insieme P può contenere un numero arbitrario di quintuple che hanno gli stessi due elementi iniziali. Il **grado** di non-determinismo è il massimo numero di quintuple della macchina che inizia con la stessa coppia stato-carattere. Il grado è al più $3 \cdot |Q| \cdot |\Sigma|$.

Possiamo visualizzare una computazione non deterministica come un albero sorgente i cui nodi sono gli stati globali utilizzati dalla computazione. La radice è lo stato globale iniziale e i figli di ciascun nodo su sono gli stati globali che corrispondono all'esecuzione di una istruzione a partire da su. Se K è il grado di non-determinismo, un nodo ha al più K-figli. Chiamiamo **computazione deterministica** di $NT(x)$ ciascun percorso nell'albero uscente dalla radice e che termina quando incontra una foglia.



$$O_{NT}(x) = \begin{cases} q_1 & \text{se almeno una delle computazioni deterministiche di } NT(x) \text{ termina in } q_1. \\ q_2 & \text{se tutte le computazioni deterministiche di } NT(x) \text{ terminano in } q_2. \\ \text{non definito} & \text{altrimenti.} \end{cases}$$

Teo 2.1: $\forall \text{NT } \exists T: \forall \text{ possibile input } x \text{ di NT } [O_{NT}(x) = O_T(x)]$.

dim. (Tecnica della simulazione)

Costruiamo una macchina deterministica T che simula il comportamento della macchina non deterministica NT con grado di non determinismo K . La macchina T su input x esegue una visita in ampiezza, basata sulla tecnica della coda di rondine con ripetizioni, dell'albero corrispondente alla computazione $NT(x)$.

Partendo dallo stato globale iniziale $Sc(T, x, o)$ si simulano tutte le computazioni (lunghe un passo al più K); se non possiamo concludere nulla allora torniamo a $Sc(T, x, o)$ e simuliamo tutte le computazioni lunghe due passi (al più K^4) e così via. \square

Lez 5

La macchina di Turing Universale è una macchina di Turing particolare U che riceve in input la descrizione di un'altra macchina di Turing T e un possibile input x di T ed esegue la computazione $U(T, x)$ il cui esito coincide con quello della computazione $T(x)$. La macchina di Turing Universale è capace di simulare la computazione che qualunque macchina di Turing potrebbe eseguire su qualunque parola una volta che le descrizioni della macchina e della parola vengono scritte sul nastro di input. E' quindi il progetto logico di un calcolatore.

La macchina di Turing universale che andiamo a definire è una macchina di Turing che utilizza 4 nastri a testine indipendenti (possibile trasformarla ad un nastro):

N_1 , il nastro in cui è memorizzata la descrizione di T ;

N_2 , il nastro di lavoro di U su cui è memorizzato l'input x della macchina T ;

N_3 , il nastro su cui sarà memorizzato lo stato attuale della macchina T ;

N_4 , il nastro su cui verrà scritto lo stato di accettazione di T .

Indichiamo $Q_T = \{w_0, \dots, w_m\}$ dove w_0, w_1, \dots, w_m sono rispettivamente lo stato iniziale, di accettazione e di rigetto di T . Indichiamo con $P = \{p_1, \dots, p_n\}$ l'insieme delle quintuple di T e con $p_i = \langle w_{i1}, b_{i1}, b_{i2}, w_{i2}, m_i \rangle$. Assumiamo che T sia descritta dalla parola $p_T \in [Q_T \cup \{0, 1, \oplus, \otimes, -\}]^*$

$$p_T = w_0 - w_1 \otimes w_2 - b_{11} - b_{12} - w_{12} - m_1 \oplus \dots \oplus w_n - b_{n1} - b_{n2} - w_{n2} - m_n \oplus$$

Quando la computazione $U(p_T, x)$ ha inizio, le testine di N_1 e N_2 sono posizionate sui simboli più a sinistra $\neq \square$. La macchina U esegue l'argomento di seguito descritto:

i) Nello stato q_0 vengono copiati w_0 su N_3 e w_1 su N_4 , la testina di N_1 si sposta sul simbolo a destra di \otimes e la macchina entra nello stato q_1 :

$$\langle q_0, (x, a, \square, \square), (x, a, x, \square), q_0, (d, t, f, t) \rangle \quad \forall x \in Q_T \wedge \forall a \in \{0, 1, \square\}$$

$$\langle q_0, (-, a, x, \square), (-, a, x, \square), q_0, (d, f, f, f) \rangle \quad \forall a \in \{0, 1, \square\} \wedge \forall x \in Q_T$$

$$\langle q_0, (y, a, x, \square), (y, a, x, y), q_0, (d, f, f, f) \rangle \quad \forall a \in \{0, 1, \square\} \wedge \forall x, y \in Q_T$$

$$\langle q_0, (\otimes, a, x, y), (\otimes, a, x, y), q_0, (d, f, f, f) \rangle \quad \forall a \in \{0, 1, \square\} \wedge \forall x, y \in Q_T$$

2) Nello stato q_1 ha iniziato la ricerca di una quintupla su N_1 che abbia come primo simbolo lo stesso simbolo letto dalla testina su N_3 e come secondo quello letto dalla testina di N_2 :

(a) nello stato q_1 legge lo stesso simbolo su N_1 ed N_3 , sposta la testina di N_1 a destra di due posizioni ed entra in $q_{\text{stato corretto}}$:

$$\langle q_1, (x, a, x, y), (x, a, x, y), q_1, (d, f, f, f) \rangle \quad \forall x, y \in Q_T \wedge \forall a \in \{0, 1, \square\}$$

$$\langle q_1, (-, a, x, y), (-, a, x, y), q_{\text{stato corretto}}, (d, f, f, f) \rangle \quad \forall a \in \{0, 1, \square\} \wedge \forall x, y \in Q_T$$

Ora la testina di N_1 è posizionata sul secondo elemento che si sta esaminando:

i. Se legge lo stesso simbolo su N_1 ed N_2 allora sposta la testina di N_1 a destra di due posizioni ed entra nello stato q_{scrivi} .

ii. Se legge simboli differenti su N_1 ed N_2 entra nello stato q_2 e sposta la testina di N_1 a destra fino a posizionarla sul primo simbolo successivo a \oplus che incontra, se non è \square , entra in q_1 , altrimenti nello stato di rigetto.

(b) nello stato q_1 legge simboli differenti su N_1 ed N_3 , entra nello stato q_3 e sposta la testina a destra fino a posizionarla sul primo simbolo successivo a \oplus che incontra, se non è \square , entra in q_1 , altrimenti confronta quello che sta leggendo su N_3 con lo stato di accettazione w_2 di T scritto su N_4 , e se uguali entra in q_{scrivi} altrimenti in q_2

3) Nello stato q_{scrivi} inizia l'esecuzione della quintupla scrivendo su N_2 ed entra nello stato $q_{\text{cambiastato}}$ muovendo a destra di due posizioni la testina di N_1 .

a) Nello stato $q_{\text{cambiastato}}$ prosegue l'esecuzione della quintupla modificando il contenuto su N_3 ed entra in q_{nuovi} muovendo a destra di due posizioni la testina di N_1 .

5) Nello stato q_{inizio} termina l'esecuzione della quintupla muovendo la testina su N_2 ed entra in q_{inizio} muovendo la testina a sinistra la testina di N_1 .

6) Nello stato q_{inizio} muovendo a sinistra la testina del nastro N_1 , fino a quando non legge un \otimes su N_1 , e poi entra nello stato q_1 muovendo a destra la testina di N_1 .

La computazione $U(p_T, x)$ rigetta ogni volta che U non trova la quintupla da eseguire e lo stato attuale di T non è lo stato di accettazione di T , dunque U rigetta il suo input (p_T, x) senza verificare che la computazione $T(x)$ abbia rigettato.

La descrizione descritta è ad alto livello perché utilizza l'insieme degli stati Q della macchina T da simulare come alfabeto di lavoro. Poiché U deve simulare qualsiasi macchina non è un'assunzione ragionevole. Per ovviare a tale problema assumiamo che l'alfabeto di lavoro di U sia $\Sigma = \{0, 1, \oplus, \otimes, -, +, s, d\}$

Sia $b^Q: Q \rightarrow \lceil \log(Q) \rceil$ una funzione che codifica in binario T utilizzando per ciascuno di essi $m = \lceil \log(Q) \rceil$ cifre, e, per ogni $w \in Q$ indichiamo con

$b^Q(w) = b_1^Q(w)b_2^Q(w)\dots b_n^Q(w)$ la codifica di w , la descrizione di T è la parola $\beta_T \in \Sigma^*$: $\beta_T = b^Q(w_0) - b^Q(w_1) \oplus b^Q(w_2) - b^Q(w_3) - m_1 \oplus \dots \oplus b^Q(w_n) - b^Q(w_1) - b^Q(w_2) - b^Q(w_3) - m_n \otimes$

(con conseguenti modifiche alle quintuple $p \in P$).

Let 6

Una macchina di Turing di tipo riconoscitore calcola una funzione booleana: dato un input $x \in \Sigma^*$, verifica se x soddisfa una certa proprietà, o predicato π .

$$O_T(x) = q_A \Leftrightarrow \pi(x)$$

$$\{x \in \Sigma^* : O_T(x) = q_A\} = \{x \in \Sigma^* : \pi(x)\}$$

- Un linguaggio L è un sottoinsieme di Σ^* : $L \subseteq \Sigma^*$

Il linguaggio complemento L^c di un linguaggio $L \subseteq \Sigma^*$ è l'insieme delle parole non contenute in L : $L^c = \Sigma^* - L$

Un linguaggio $L \subseteq \Sigma^*$ è accettabile se esiste una macchina di Turing T tale che

$$\forall x \in \Sigma^* [O_T(x) = q_A \Leftrightarrow x \in L]$$

La macchina T è detta accettare L . Nel caso in cui $x \notin L$ $O_T(x) = q_R$ oppure $T(x)$ non raggiunge mai uno stato finale, l'accettabilità di L non dà alcuna indicazione circa l'accettabilità di L^c .

- Un linguaggio $L \subseteq \Sigma^*$ è decidibile se esiste una macchina di Turing T che termina

$\forall x \in \Sigma^*$ e tale che:

$$\forall x \in \Sigma^* [O_T(x) = q_A \Leftrightarrow x \in L].$$

La macchina T è detta decidere L . Se una macchina T decide $L \subseteq \Sigma^*$ allora

$$O_T(x) = \begin{cases} q_A & \text{se } x \in L \\ q_R & \text{se } x \notin L \end{cases}$$

Teo 3.1: Un linguaggio $L \subseteq \Sigma^*$ è decidibile sse L e L^c sono accettabili.

dim:

$$\Rightarrow) \text{Se } L \text{ è decidibile} \Rightarrow \exists T: \forall x \in \Sigma^* O_T(x) = \begin{cases} q_A & \text{se } x \in L \\ q_R & \text{se } x \notin L \end{cases}$$

dunque $O_T(x) = q_A$ sse $x \in L$, ossia T accetta L ,

deriviamo una nuova macchina T' con stati finali q_A' e q_R' e stesse quintuple di T

$$\text{più: } \langle q_A, v, v, q_A', f \rangle, \langle q_R, v, v, q_R', f \rangle \quad \forall v \in \Sigma \cup \{\square\}$$

$\forall x \in \Sigma^*$, la computazione di $T'(x)$ coincide con $T(x)$ tranne per l'ultima istruzione

dove se $T(x)$ termina in q_A , $T(x)$ esegue un'altra istruzione e rigetta in q_B , simile per q_R e q_S . T accetta x sse $x \in L'$, ossia T accetta L' .

\Leftarrow Se L e L' sono accettabili $\Rightarrow \exists T_1, T_2 : \forall x \in \Sigma^* T_i(x)$ accetta sse $x \in L$ e $T_2(x)$ accetta sse $x \in L'$. Componendo opportunamente T_1 e T_2 definiamo una nuova macchina T che, simulando le computazioni eseguite da T_1 e da T_2 su input $x \in \Sigma^*$ decide L . Se T simulasse prima l'intera computazione di T_1 e poi l'intera computazione di T_2 non si avrebbe garanzia di terminazione pertanto: T ha due nastri, su entrambi è inizialmente scritto l'input x : la computazione $T(x)$ avviene alternando singole istruzioni di T_1 e T_2 :

- 1) esegui una singola istruzione di T_1 sul nastro 1: se T_1 entra in q_A allora T accetta
altrimenti esegui il passo 2)
- 2) esegui una singola istruzione di T_2 sul nastro 2: se T_2 entra in q_A allora T rigetta
altrimenti esegui il passo 1)

Quindi T decide L . \square

Una funzione parziale da un dato insieme A ad un insieme B è una qualunque funzione $f: A \rightarrow B$. Se il dominio di f coincide con A , f si dice totale.

Dato un qualsiasi alfabeto finito Σ , indichiamo con Σ^* l'insieme delle parole su Σ , ossia l'insieme delle stringhe di lunghezza finita costituite da caratteri in Σ .

Siano Σ e Σ' due alfabeti finiti; una funzione (parziale) $f: \Sigma^* \rightarrow \Sigma'$ è una funz. calcolabile se esiste una macchina di Turing T di tipo trasduttore che, dato in input $x \in \Sigma^*$, termina con la stringa $f(x)$ scritta sul nastro di output sse $f(x)$ è definita.

Sia Σ un alfabeto finito ed $L \subseteq \Sigma^*$ un linguaggio. La funzione caratteristica $\chi_L: \Sigma^* \rightarrow \{0, 1\}$ di L è una funzione totale tale che:

$$\forall x \in \Sigma^* \quad \chi_L(x) = \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases}$$

Teo 3.2: Un linguaggio L è decidibile sse la funzione χ_L è calcolabile.

dim.

\Rightarrow Supponiamo che $L \subseteq \Sigma^*$ sia decidibile, allora esiste una macchina T tipo riconoscitore

con stato di accettazione q_A e di rigetto q_R tale che: $O_T(x) = \begin{cases} q_A & \text{se } x \in L \\ q_R & \text{se } x \notin L \end{cases}$

A partire da T , definiamo una macchina di tipo trasduttore T' con due nastri, che, con input $x \in \Sigma^*$ opera nella seguente maniera:

- 1) sul primo nastro, è scritto l'input x , esegue la computazione $T(x)$;
- 2) se $T(x)$ termina nello stato q_A , allora sul nastro di output sarà scritto 1, altrimenti 0.

Successivamente termina;

\Leftarrow) Supponiamo che X_L sia calcolabile, allora esiste una macchina T tipo trasduttore, che, $\forall x \in \Sigma^*$, calcola $X_L(x)$.

A partire da T , definiamo una macchina di tipo riconoscitore T' con due nastri, che, con input $x \in \Sigma^*$ opera nella seguente maniera:

- 1) sul primo nastro è scritto l'input x , esegue la computazione $T(x)$, scrivendo il risultato sul secondo nastro.
- 2) se sul secondo nastro è scritto 1, allora la computazione $T(x)$ termina nello stato di accettazione, altrimenti di rigetto. \square

Data una qualsiasi funzione f , possiamo associare ad f un linguaggio che sia decidibile sse f è calcolabile? Iniziamo con associare ad ogni funzione $f: \Sigma^* \rightarrow \Sigma^*$, il linguaggio $L_f = \{(x, y) : x \in \Sigma^* \wedge y \in \Sigma^* \wedge y = f(x)\}$

Teo 3.3: Se la funzione $f: \Sigma^* \rightarrow \Sigma^*$ è totale e calcolabile allora il linguaggio

$$L_f \subseteq \Sigma \times \Sigma^*$$

è decidibile.

Poiché f è calcolabile e totale, allora esiste una macchina di Turing T di tipo trasduttore, che, $\forall x \in \Sigma^*$, calcola $f(x)$.

A partire da T , definiamo una macchina di tipo riconoscitore T' con due nastri, che, con input (x, y) , con $x \in \Sigma^*$ e $y \in \Sigma^*$. Opera nella seguente maniera:

- 1) sul primo nastro è scritto l'input (x, y)
- 2) sul secondo nastro esegue la computazione $T(x)$, scrivendo il risultato z
- 3) esegue un confronto tra z e y : se $z = y$ allora la computazione $T'(x)$ termina nello stato di accettazione, altrimenti nello stato di rigetto. \square

Teo 3.4: Sia $f: \Sigma^* \rightarrow \Sigma^*$ una funzione. Se il linguaggio $L_f \subseteq \Sigma^* \times \Sigma^*$, è decidibile allora f è calcolabile. \square

Poiché $L_f \subseteq \Sigma^* \times \Sigma^*$ è decidibile, esiste una macchina di Turing di tipo riconoscitore T , tale che $\forall x \in \Sigma^*$ e $\forall y \in \Sigma^*$

$$Q_f(x) = \begin{cases} q_A & \text{se } y = f(x) \\ q_B & \text{altrimenti} \end{cases}$$

A partire da T , definiamo una macchina di tipo trasduttore T' , che, con input $x \in \Sigma^*$ opera nella seguente maniera:

- 1) scrive il valore $i=0$ su N_1 ;
- 2) enumera tutte le stringhe $y \in \Sigma^*$, la cui lunghezza è pari al valore scritto sul primo nastro, simulando per ciascuna di esse la computazione $T(x, y)$, cioè opera come segue:
 - a) sia y la prima stringa di lunghezza i non ancora enumerata, allora scrive y su N_2 .
 - b) esegue la computazione $T(x, y)$ su N_3 .
 - c) se $T(x, y)$ termina nello stato q_A allora scrive sul nastro di output la stringa y e termina, altrimenti, eventualmente incrementando il valore i scritto su N_1 se y era l'ultima stringa di lunghezza i , torna al passo 2).

Poiché L_f è decidibile il passo (b) termina $\forall x, y$. Se x appartiene al dominio di f allora $\exists \bar{y} \in \Sigma^* : \bar{y} = f(x)$ e quindi $\langle x, \bar{y} \rangle \in L_f$. Prima o poi la stringa \bar{y} verrà scritta su N_2 e $T'(x, \bar{y})$ terminerà in q_A . Questo dimostra che f è calcolabile. \square

Cor: Un linguaggio L è decidibile sse L^c è decidibile.

Teo 5.7: Se L_1 e L_2 sono due linguaggi accettabili allora $L_1 \cap L_2$ è un linguaggio accettabile. Se L_1 e L_2 sono due linguaggi decidibili allora $L_1 \cap L_2$ è un linguaggio decidibile.

Tuo s.8: Se L_1 e L_2 sono due linguaggi accettabili allora $L_1 \cup L_2$ è un linguaggio accettabile. Se L_1 e L_2 sono due linguaggi decidibili allora $L_1 \cup L_2$ è un linguaggio decidibile.

Let 7 e 8

Tutti i modelli di calcolo definiti fino ad ora sono Turing-equivalenti, qualunque funzione calcolabile mediante uno di tali modelli è calcolabile anche mediante una macchina di Turing.

Tesi di Church-Turing: se la soluzione di un problema può essere descritta da una serie finita di passi elementari, allora esiste una macchina di Turing in grado di calcolarlo.

E' calcolabile tutto (e solo) ciò che può essere calcolato da una macchina di Turing.

Questa è universalmente accettata, ma tuttora non esiste per essa una dimostrazione formale. Dimostriamo però che il potere computazionale di un linguaggio di programmazione non differisce da quello delle macchine di Turing. Il linguaggio preso in considerazione è il

Pascal Minimo che ha le seguenti istruzioni:

istruzione di assegnazione: $a \leftarrow b$

istruzione condizionale: if ... then ... else

istruzioni di loop: while(...) do e for(...)

istruzioni per input e output

dispone di variabili semplici (interi, caratteri,...), collezione di oggetti e array

Teo: Per ogni programma scritto in accordo con il linguaggio di programmazione Pascal Minimo esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.

dim.

Sia P un programma in Pascal Minimo, prima di costruire una macchina di Turing T che "si comporta" come P, scriviamo P in una forma opportuna ossia:

- 1.) utilizzando variabili ausiliarie, eliminiamo gli array eventualmente presenti nelle condizioni
- 2.) scriviamo una sola istruzione in ciascuna riga del programma
- 3.) numeriamo le righe del programma

$p \leftarrow 1; i \leftarrow 1;$

while ($i \leq n$) do

if ($A[i] > p$) then $p \leftarrow A[i];$

if ($p > K$) then $ris \leftarrow K$ else $ris \leftarrow p$
output ris

diventa:

```
1  p ← 1;
2  i ← 1;
3  while (i ≤ n) do
4    ausA ← A[i];
5    if (ausA > p) then
6      p ← A[i];
7    if ( $p > K$ ) then  $ris \leftarrow K$ 
8    else  $ris \leftarrow p$ 
9  output  $ris$ 
```

Sia P un programma in Pascal Minimo costruiamo T , multinastro a testine indipendenti
 T utilizza:

un nastro per ciascuna variabile,

un nastro aggiuntivo per ogni variabile di tipo array sul quale memorizzare in unario
l'indice dell'array al quale si vuole accedere,

un nastro di lavoro per la valutazione delle espressioni e delle condizioni contenute
nel programma.

I contenuti dei nastri sono codificati in binario (o unario) con una sola eccezione:
nei nastri per array viene utilizzato il carattere $\$$ come separatore tra gli elementi
dell'array.

La macchina corrispondente al programma sopra descritto utilizza 8 nastri: 5 per
variabili semplici ($N_p, N_i, N_{aus}, N_k, N_{ris}$), un nastro per l'array A e uno per l'indice (N_a, N_{ind})
infine un nastro di lavoro.

Per costruire le quintuple di T si utilizzerà una descrizione ad alto livello. Ogni volta
che un istruzione finisce le testine vanno riportate sul primo simbolo dei nastri. (sx)

Assegnazione (variabile)

i) $a \leftarrow b$

$\langle q_i, (\text{copia } N_b \text{ su } N_a \text{ sx}), q_{i+1}, f \rangle$

i+1) ...

Assegnazione (array)

Assumiamo che ogni elemento di A possa essere contenuto in un'unica cella di N_A .

i) $A[i] \leftarrow b$

$\langle q_i, (\text{copia } N_i \text{ su } N_{\text{nodo}_A} \text{ in unario sx}), q_i, f \rangle$

i+1) ...

$\langle q_i, (\text{muovi } N_A \text{ e } N_{\text{nodo}_A} \text{ a destra fino a } \square), q_i, f \rangle$

$\langle q_i, (\text{copia } N_b \text{ su } N_A \text{ sx}), q_{i+1}, f \rangle$

Selezione

i) $\text{if } (a > s) \text{ then begin}$

i+1) $c \leftarrow a$

i+2) ... end

i+3) ...

calcolato tramite
nastro di lavoro

else

$\langle q_i, (\text{se } a > s), q_{i+1}, f \rangle$

$\langle q_i, (\text{se } a \leq s), q_{i+3}, f \rangle$

$\langle q_{i+1}, \dots, q_{i+2}, f \rangle$

$\langle q_{i+2}, \dots, q_{i+3}, f \rangle$

Loop

i) $\text{while } (a > s) \text{ do begin}$

i+1) $c \leftarrow a$

i+2) ... end

i+3) ...

calcolato tramite
nastro di lavoro

else

$\langle q_i, (\text{se } a > s), q_{i+1}, f \rangle$

$\langle q_i, (\text{se } a \leq s), q_{i+3}, f \rangle$

$\langle q_{i+1}, \dots, q_{i+2}, f \rangle$

$\langle q_{i+2}, \dots, q_i, f \rangle$

Teo: Per ogni macchina di Turing T di tipo riconoscitore ad un nastro esiste un programma P scritto in accordo alle regole di Pascal Minimo tale che per ogni stringa x, se $T(x)$ termina nello stato finale $q_F \in \{q_A, q_R\}$ allora P con input x restituisce q_F in output.

dim.

Dimostriamo questo teorema progettando un programma U che si comporta come la macchina Universale $T = \langle \Sigma, Q, P, q_0, \{q_A, q_R\} \rangle$.

Per memorizzare le quintuple della macchina T che si vuole simulare utilizziamo 5 array:

Q_1, S_1, S_2, Q_2, M e usiamo i valori -1, 0, +1 rispettivamente per s, f, d.

Per l'i-esima quintupla avremo:

$\langle q, s_1, s_2, q', d \rangle$ $Q_1[i] = q, S_1[i] = s_1, S_2[i] = s_2, Q_2[i] = q', M[i] = d$.

Rappresentiamo il nastro di T mediante l'array N, può avere anche indici negativi.

Programma che simula U: In Input viene fornita la descrizione di T, nella variabile q memorizziamo lo stato interno di T e in t la posizione della testina, il programma consiste in un loop while che termina una volta raggiunto uno stato finale.

Input: stringa $x_1 \dots x_n$ memorizzata nell'array N, con $N[i] = x_i$ per $i=1, \dots, n$, array

Q_1, S_1, S_2, Q_2, M descritti nel testo e q_0, q_A, q_R .

1. $q \leftarrow q_0; t \leftarrow 1;$
2. $pC \leftarrow 1; uC \leftarrow n;$
3. while ($q \neq q_A \wedge q \neq q_R$) do
4. $j \leftarrow 1; trovata \leftarrow \text{falso};$
5. while ($j \leq K \wedge \text{trovata} = \text{falso}$) do
6. if ($q = Q_1[j] \wedge N[t] = S_1[j]$) then $\text{trovata} \leftarrow \text{vero};$
7. else $j \leftarrow j + 1;$
8. if ($\text{trovata} = \text{vero}$) then
9. $N[t] \leftarrow S_2[j]; q \leftarrow Q_2[j]; t \leftarrow t + M[j];$
10. if ($t < pC$) then
11. $pC \leftarrow t; N[t] = \square;$
12. if ($t > uC$) then
13. $uC \leftarrow t; N[t] = \square;$
14. else $q \leftarrow q_R;$
15. Output: $q;$

In 3-7, vengono esaminate ordinatamente le quintuple di T fino a quando: ne viene trovata una che può essere eseguita e in questo caso si pone $\text{trovata} = \text{true}$, oppure non ne viene trovata alcuna che può essere eseguita (si aumenta j).

In 2, Con pC e uC memorizziamo gli indici dell'array N che individuano la cella più a sinistra e più a destra della porzione di nastro non blank di T.

In 3, Se la quintupla da eseguire è stata trovata la si esegue aggiornando nastro, stato interno e muovendo la testina.

In 10-13, Se eseguendo la quintupla la testina di T viene spostata su una cella a sinistra o destra della porzione di nastro sinora utilizzata allora aggiorna pC e uC.

In 14, Se non viene trovata alcuna quintupla da eseguire si porta la macchina in qr..

Lez 3

Il lavoro di Cantor dimostra che esistono insiemi infiniti "piccoli" e insiemi infiniti "grandi" dove "piccolo" e "grande" sono basati sul concetto di corrispondenza biunivoca e un numero transfinito è la cardinalità di un insieme infinito.

Cantor ha dimostrato che non esiste una corrispondenza biunivoca fra l'insieme dei numeri naturali e l'insieme dei numeri reali e questo prova che l'insieme dei numeri reali è strettamente "più grande" dell'insieme dei naturali.

Cantor ha dimostrato che non esiste una corrispondenza biunivoca fra l'insieme dei naturali e l'intervallo reale $[0, 1]$.

Esiste un problema che non può essere risolto? Per dimostrarlo, dimostriamo che le macchine di Turing sono tante quanto i naturali e che esiste almeno un linguaggio che non è deciso da alcuna macchina di Turing. Ossia esiste almeno un problema che non può essere risolto con una macchina di Turing

Teo 5.1: Sia Σ un insieme finito. Allora l'insieme Σ^* costituito dalle parole (di lunghezza finita) di caratteri di Σ è numerabile. □

Teo 5.2: L'insieme T delle macchine di Turing definite sull'alfabeto $\{0, 1\}$ e dotate di un singolo nastro (più eventualmente quello di output) è numerabile. □
dim. X leggera modifica

Occorre dimostrare l'esistenza di una biezione fra tale insieme e l'insieme \mathbb{N} .

Sia T una macchina di Turing ad un nastro (più eventualmente quello di output) definita sull'alfabeto $\{0, 1\}$ e su un insieme finito di stati Q , ove q_0 è lo stato iniziale e q_f è lo stato di accettazione se T è riconoscitore o finale se T è trasduttore.

Sia $b^Q: Q \rightarrow \{0, 1\}^m$ una codifica binaria degli stati di T che utilizza $m = \lceil \log(Q) \rceil$ cifre e $\forall q \in Q$ indichiamo con $b^Q(q) = b_1^Q(q) | b_2^Q(q) | \dots | b_n^Q(q)$ la codifica di q . Allora rappresentiamo T mediante la parola $B_T \in \Sigma^*$ con $\Sigma = \{0, 1, \oplus, \otimes, -, +, f, s, d\}$

$$B_T = b^Q(q_0) - b^Q(q_f) \otimes b^Q(q_{f+}) - b_{11} - b_{12} - b^Q(q_{f+2}) - m_1 \oplus \dots \oplus b^Q(q_{f+n}) - b_{n1} - b_{n2} - b^Q(q_{f+n+1}) - m_n \oplus$$

Fissati P , stato iniziale e finale, è fissato anche il comportamento della macchina di Turing su un qualunque $x \in \{0,1\}^*$. T'è univocamente rappresentata dalla parola $\beta_T \in \Sigma^*$:

$$\forall T, T' \in \Upsilon : T \neq T' \Leftrightarrow \beta_T \neq \beta_{T'}$$

Questo prova che abbiamo una biezione tra Υ e un sottoinsieme di Σ^* e, dunque, poiché Σ è un insieme finito, esso è numerabile.

Possiamo trasformare la codifica β_T di una macchina di Turing T in un numero naturale $v(T)$ in modo tale che per ogni coppia di macchine di Turing distinte T e T' $v(T) \neq v(T')$.

Sia dunque $T \in \Upsilon$ una macchina di Turing ad un nastro (più, eventualmente, il nastro di output) e sia $\beta_T \in \Sigma^*$ la sua codifica. Trasformiamo $\beta_T \in \Sigma^*$ in una parola in $\{0,1,\dots,7\}^*$ nel seguente modo:

- sostituendo ogni carattere "s", "f" e "d" in β_T rispettivamente con i caratteri "5", "6" e "7";
- sostituendo ogni carattere "-" in β_T con il carattere "4";
- sostituendo ogni carattere " \oplus " e " \otimes " in β_T rispettivamente con i caratteri "3" e "2";
- premettendo il carattere "2" alla stringa ottenuta.

La parola in $\{0,1,\dots,7\}^*$ ottenuta può essere considerata come un numero espresso in notazione decimale: ecco il numero $v(T) \in \mathbb{N}$ associato a T . □

$$b^0(q_0) - b^0(q_1) \otimes b^0(q_{11}) - b_{11} - b_{12} - b^0(q_{12}) - m, \oplus \dots$$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
 2 0 1 0 4 0 1 1 2 1 0 0 4 1 4 0 4 0 0 1 4 7 3 ...
 $\{5,6,7\}$

A questo punto possiamo ordinare le macchine: $T_h < T_k$ se $h < k$

Anche l'input di una TM è rappresentato da un numero intero, allora abbiamo un "primo" input 1 - rappresenta la parola "□"
 "secondo" input 10 - rappresenta la parola "0"
 "terzo" input 11 - rappresenta la parola "1"

"quarto" input 100 - rappresenta la parola "00" e così via

Ossia una generica parola binaria b è rappresentata dal numero n la cui rappresentazione binaria è ottenuta premettendo un 1 alla parola b , e dunque possiamo pensare che l'input di una TM sia un numero intero.

Possiamo quindi costruire una matrice M binaria con infinite righe e colonne tale che

$$M[i, k] =$$

		parole					
		1	2	3	4	5	...
MT	T_{h_1}	0	0	1	0	1	$\dots = L_{T_{h_1}} = \{3, 5, \dots\}$
	T_{h_2}	1	0	1	0	0	\dots
	\dots	\dots	\dots	\dots	\dots	\dots	\dots

M rappresenta tutti i linguaggi accettati dalle MT.

Consideriamo la diagonale della matrice e costruiamo un linguaggio L che rappresenta la diagonale "complementata".

Formalmente $L = \{K : M[K, K] = 0\}$ o anche $L = \{K : (K) \text{ non accetta}\}$

Il linguaggio L non è accettato da nessuna delle macchine che compaiono come righe della matrice M . Ma tutte le TM sono rappresentate in M . Questo significa che non esiste alcuna TM che accetti L , ossia, L è un linguaggio **non accettabile**.

Questa corollario è puramente esistenziale ma non costruttiva, non fornisce alcuna metodologia per costruire un linguaggio non accettabile. Inoltre il corollario non chiarisce il ruolo dei linguaggi decidibili, non chiarisce quindi se i linguaggi non decidibili sono tutti e soli i linguaggi non accettabili oppure se esistono linguaggi accettabili ma non decidibili.

Consideriamo la funzione $v : \mathbb{T} \rightarrow \mathbb{N}$ e definiamo il seguente linguaggio:

$$L_H = \{(i, x) : i \text{ è la codifica di una macchina di Turing } \stackrel{\wedge}{\overbrace{T_i(x) \text{ termina}}} \} \subseteq \mathbb{N} \times \mathbb{N}$$

Se un numero naturale i non è la codifica di alcuna macchina di Turing in \mathbb{T} , allora comunque si scelga $x \in \{0, 1\}^*$, $(i, x) \notin L_H$ (Halting problem)

Teo 5.3: L_H è un linguaggio accettabile.

dim. ~~leggera modifica~~

Bisogna dimostrare che esiste una macchina di Turing T tale che:

$$\forall (i, x) \in \mathbb{N} \times \mathbb{N}, O_T(i, x) = q_A \Leftrightarrow (i, x) \in L_H$$

La macchina T cercata è, una modifica della macchina di Turing universale U . La macchina T in questione presenta un paio di modifiche: T inizia la sua computazione verificando che i non contenga le cifre "8" e "9" e che non inizi con "2": se non è così termina nello stato di rigetto altrimenti cancella la cifra "2" iniziale e traduce quello che rimane nell'alfabeto di lavoro Σ di U . Poi T simula la computazione di U e, se U (q_S o q_P) termina allora T termina nello stato di accettazione.

Si osserva che, se i non è la codifica di alcuna macchina di Turing, poiché l'insieme delle quintuple di una qualsiasi macchina di Turing è totale e le computazioni con input che non rispettano le specifiche non terminano, allora $U(i, x)$ non termina.

Sia $(i, x) \in L_H$: allora, la computazione $T_i(x)$ termina e quindi, in virtù di quanto appena descritto circa la macchina T , la computazione $T(i, x)$ accetta.

Viceversa, sia $(i, x) \in \mathbb{N} \times \mathbb{N}$ tale che $T(i, x)$ accetta; poiché la computazione $T(i, x)$ simula la computazione $U(i, x)$, allora anche $U(i, x)$ termina e, dunque, i è la codifica di una macchina di Turing e $T_i(x)$ termina, quindi $(i, x) \in L_H$.

Teo 5.4: Il linguaggio L_H non è decidibile

dim.

Supponiamo per assurdo che L_H sia decidibile. Allora esiste una macchina di Turing T che:

$$T(i, x) = \begin{cases} q_A & \text{se } (i, x) \in L_H \\ q_R & \text{se } (i, x) \notin L_H \end{cases}$$

Da T possiamo, complementando gli stati di accettazione e di rigetto di T , derivare una nuova macchina T' che, terminando su ogni input, accetta tutte e solo le coppie $(i, x) \in \mathbb{N} \times \mathbb{N} - L_H$ ossia:

$$T'(i, x) = \begin{cases} q_A & \text{se } (i, x) \notin L_H \\ q_R & \text{se } (i, x) \in L_H \end{cases}$$

A partire da T deriviamo una terza macchina T' che, su una coppia di interi, opera su un singolo input $i \in \mathbb{N}$. Inoltre $T^*(i)$ accetta se $T'(i, i)$ accetta, mentre non termina se $T'(i, i)$ rigetta. E' possibile farlo apportando a T' le seguenti modifiche:

- sostituendo lo stato q_F con un nuovo stato non finale q'_F in tutte le quintuple di T' che terminano nello stato q_F
- aggiungiamo alle quintuple di T' la quintupla $\langle q'_F, y, y, q_F, f \rangle$ $\forall y \in \{0, 1\}$

$$T^*(i) = \begin{cases} \text{non termina} & \text{se } T'(i, i) \text{ rigetta} \\ q_A & \text{se } T'(i, i) \text{ accetta} \end{cases}$$

Poiche T è un insieme numerabile e $T^* \in T$, allora deve esistere $K \in \mathbb{N}$ tale che $T^* = T_K$. Qual'è l'esito della computazione $T_K(K)$?

Se $T_K(K) = T^*(K)$ accettasse, allora $T'(K, K)$ dovrebbe accettare anch'essa, allora allora $(K, K) \in L_H$ ossia $T_K(K)$ non termina, ma per definizione di L_H , K è la codifica di una macchina di Turing $\wedge T_K(K)$ termina

(ma è assurdo dato che abbiamo supposto che $T_K(K)$ accetta e quindi termina)

Se $T^*(K)$ non termina, allora $T'(K, K)$ rigetta, quindi $(K, K) \notin L_H$ dunque $T_K(K)$ termina.

Entrambe le ipotesi portano ad una contraddizione. Allora la macchina T^* non può esistere. Poiche T^* è ottenuta tramite semplici modifiche da T' allora T non può esistere, quindi T non può esistere perciò L_H non è decidibile. □

Di conseguenza dato che: "un linguaggio L è decidibile sse L è accettabile e L^c è accettabile", poiche L_H è accettabile L_H^c non lo è.

Lez 10

Il concetto di **riducibilità funzionale** permette di correlare tra loro linguaggi in modo che:

- la decidibilità/accettabilità di un linguaggio implica la decidibilità/accettabilità dei linguaggi ad esso riducibili;
- la non decidibilità/accettabilità di un linguaggio implica la non decidibilità/accettabilità dei linguaggi cui esso si riduce.

Siano $L_1 \subseteq \Sigma^*$ e $L_2 \subseteq \Sigma^*$ due linguaggi; diciamo che L_1 è (many to one) **riducibile** ad L_2 , se esiste una funzione totale e calcolabile $f: \Sigma^* \rightarrow \Sigma^*$ tale che $\forall x \in \Sigma^* [x \in L_1 \Leftrightarrow f(x) \in L_2]$.

Se L_1 è riducibile ad L_2 scriviamo $L_1 \leq L_2$.

$(\exists f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ totale e calcolabile: $\forall (i, x) \in \mathbb{N} \times \mathbb{N} [i \cdot f(i, x) \in L_2 \Leftrightarrow (i, x) \in L_1]$)

P: Sia L_3 un linguaggio decidibile e sia L_4 un secondo linguaggio tale che $L_4 \leq L_3$: allora L_4 è decidibile.

dim.

Indichiamo con $f_{4,3}$ la funzione che riduce L_4 ad L_3 . L_3 è decidibile allora

$$\exists T_3: \forall x \in \Sigma_3^*, T_3(x) = \begin{cases} q_A & \text{se } x \in L_3 \\ q_R & \text{se } x \notin L_3 \end{cases}$$

$L_4 \leq L_3$, quindi $\exists f_{4,3}$ totale e calcolabile: $f(x) \in L_3 \Leftrightarrow x \in L_4$

$$\Rightarrow \exists T_f: \forall x \in \Sigma_4^* [x \in L_4 \Leftrightarrow T_f(x) \in L_3]$$

Costruisco la macchina T : input $x \in \Sigma_4$

1) simulo $T_f(x)$, sia y l'output di $T_f(x)$

2) simulo $T_3(y)$: se accetta $\Rightarrow q_A$

se rigetta $\Rightarrow q_R$ \square

P: Sia L_1 un linguaggio non decidibile e sia L_2 un secondo linguaggio tale che $L_1 \leq L_2$: allora L_2 è non decidibile

dim.

Supponiamo per assurdo che L_2 è decidibile \Rightarrow per il Teorema sopra dimostrato anche L_1 è decidibile, assurdo. \square

P : (zueftabili e non zueftabili) \times

Esempio: $L_{H_0} = \{i \in \mathbb{N} : T_i(o) \text{ termina}\}$, mostriamo che $L_H \leq L_{H_0}$.

Dobbiamo quindi definire la funzione totale e calcolabile che, ad ogni coppia $(i, x) \in \mathbb{N} \times \mathbb{N}$ associa un intero $K \in \mathbb{N}$ tale che $(i, x) \in L_H$ sse $K \in L_{H_0}$.

Fissiamo la coppia $(i, x) \in \mathbb{N} \times \mathbb{N}$ con $x = x_1 \dots x_n$, deriviamo l'intero $K = f(i, x)$:

1) se i non è la codifica della macchina di Turing, allora costruiamo una macchina di Turing $M_{(i,x)}$ che entra in loop qualunque sia il suo input.

2) se i è la codifica di una macchina di Turing, allora costruiamo una macchina di Turing $M_{(i,x)}$ che, con input o , simula $T_i(x)$:

$\langle q_1, o, x_1, q_2, d \rangle$

$\langle q_2, \square, x_2, q_3, d \rangle$ (riscrive $x_1 \dots x_n$ al posto di o)

...

$\langle q_n, \square, x_n, q_0, s \rangle$

riposiziona la testina sul primo carattere dell'input ed entra in q_0 di T_i

a questo punto $M_{(i,x)}$ inizia a simulare $T_i(x)$

Il numero degli stati $M_{(i,x)}$ dipende da x e potrebbe sembrare non costante, non è così:

x non è input per $M_{(i,x)}$ (input è solo o), poi abbiamo considerato una coppia (i, x) e solo dopo averla fissata abbiamo costruito $M_{(i,x)}$. $\forall x$ abbiamo una $M_{(i,x)}$, ossia x è costante per $M_{(i,x)}$

3) $K_{(i,x)}$ è la codifica di $M_{(i,x)}$, ossia $f(i, x) = K_{(i,x)}$

$\forall (i, x) \in \mathbb{N} \times \mathbb{N}$, poiché $K_{(i,x)}$ è la codifica di $M_{(i,x)}$, ossia $T_{K_{(i,x)}} = M_{(i,x)}$ allora $K_{(i,x)} \in L_{H_0}$ sse $T_{K_{(i,x)}}(o) = M_{(i,x)}(o)$ termina.

Se $T_{K_{(i,x)}}(o) = M_{(i,x)}(o)$ termina, allora i è la codifica di una macchina di Turing e $T_i(x)$ termina. Ossia se $K_{(i,x)} \in L_{H_0}$ allora $(i, x) \in L_H$.

Se $K_{(i,x)} \in L_{\#0}$ allora $T_{K(i,x)}(0) = M_{(i,x)}(0)$ non termina, allora $(i,x) \notin L_H$.

Ossia se $K_{(i,x)} \in L_{\#0}$ allora $(i,x) \notin L_H$.

Let 11

Una misura di complessità è una funzione c che associa ad ogni suo problema possibile input un valore numerico che corrisponde al "costo" della computazione della macchina sull'input considerato.

Affinché una funzione f possa essere considerata una misura di compatibilità, essa deve soddisfare le due seguenti proprietà, note come assiomi di Blum:

- 1) la funzione f è definita solo per computazioni che terminano
se una computazione $T(x)$ non termina, non ha senso considerare che tale computazione abbia come costo un valore finito;
- 2) f deve essere una funzione calcolabile

Ossia, deve esistere una macchina di Turing M che, ricevendo in input una macchina di Turing T ed un suo input x , calcola $f(T, x)$ ogniqualvolta $f(T, x)$ è definita (cioè, ogniqualvolta $T(x)$ termina) - questo significa che, il costo di una computazione $T(x)$, dobbiamo poterlo calcolarlo effettivamente.

Iniziamo con le misure di complessità che si riferiscono a computazioni deterministiche.

Per macchina di Turing deterministica T (riconoscitore o trasduttore) definita su un alfabeto Σ , e $\forall x \in \Sigma^*$ tali che $T(x)$ termina, definiamo le due funzioni seguenti:

$dtime(T, x)$ = numero di istruzioni eseguite da $T(x)$.

$dspace(T, x)$ = numero di celle utilizzate da $T(x)$.

Dimostriamo ora che le funzioni $dtime$ e $dspace$ soddisfano i due assiomi di Blum.

- 1) Per definizione, \forall macchina di Turing deterministica T e $\forall x \in \Sigma^*$, $dtime(T, x)$ e $dspace(T, x)$ sono definite sse $T(x)$ termina.
- 2) Consideriamo una modifica U della macchina di Turing universale U : aggiungiamo a U il nastro N_5 che fungerà da contatore del numero di istruzioni della computazione $T(x)$.

$Udtime(T, x)$ si comporta come $U(T, x)$ con l'unica differenza che, dopo aver eseguito un'istruzione della macchina T su input x ed essersi preparata ad eseguire l'istruzione successiva, scrive 1 sul nastro N_5 e muove a destra la testina sul tale nastro.

Al termine della computazione $U\text{dtime}(T, x)$ (se esso termina) il nastro N_5 conterrà, codificato in unario, il numero di passi eseguiti dalla computazione $T(x)$: dunque, dtime è una funzione calcolabile.

3) La dimostrazione che dspace è un' funzione calcolabile è simile.

Possiamo definire le misure di complessità che si riferiscono a computazioni non deterministiche.

Per una macchina di Turing non deterministica NT (riconoscitore) definita su un alfabeto Σ , e $\forall x \in \Sigma^*$ tali che $NT(x)$ accetta definiamo le due funzioni seguenti:

$n\text{time}(NT, x)$ = minimo numero di istruzioni eseguite da una computazione deterministica accettante di $NT(x)$.

$n\text{space}(NT, x)$ = minimo numero di celle utilizzate da una computazione deterministica accettante di $NT(x)$.

$n\text{time}$ e $n\text{space}$ sono funzioni parziali avendole definite solo per computazioni accettanti. Per poter estendere la definizione a computazioni che rigettano è necessario tenere in considerazione la assimmetria delle definizioni di accettazione e rigetto di una macchina non deterministica:

$NT(x)$ accetta se esiste una computazione deterministica che accetta.

$NT(x)$ rigetta se tutte le sue computazioni deterministiche rigettano.

Per una macchina di Turing non deterministica NT (riconoscitore) definita su un alfabeto Σ , e $\forall x \in \Sigma^*$ tali che $NT(x)$ rigetta definiamo le due funzioni seguenti:

$n\text{time}(NT, x)$ = massimo numero di istruzioni eseguite da una computazione deterministica di $NT(x)$ se $NT(x)$ rigetta.

$n\text{space}(NT, x)$ = massimo numero di celle utilizzate da una computazione deterministica di $NT(x)$ se $NT(x)$ rigetta.

Anche con questa estensione, le funzioni $n\text{time}$ e $n\text{space}$ restano funzioni parziali.

Teo 6.1: Sia T una macchina di Turing deterministica, definita su un alfabeto Σ (non contenente il simbolo \square) e un insieme di stati Q e sia $x \in \Sigma^*$ tale che $T(x)$ termina.

Allora:

$$\text{dspace}(T, x) \leq \text{dtime}(T, x) \leq \text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)}$$

Analogamente: Sia NT una macchina di Turing non deterministica, definita su un alfabeto Σ (non contenente il simbolo \square) e un insieme di stati Q e sia $x \in \Sigma^*$ tale che $T(x)$ termina. Allora:

$$\text{nspase}(T, x) \leq \text{nime}(T, x) \leq \text{nspase}(T, x) |Q| (|\Sigma| + 1)^{\text{nspase}(T, x)}$$

dim.

$$1) \text{dspace}(T, x) \leq \text{dtime}(T, x)$$

Se $T(x)$ utilizza $\text{dspace}(T, x)$ celle di memoria, quelle celle deve almeno leggerle, per leggere ciascuna cella impiega un'istruzione. Esistono anche casi "anomali" dove non dobbiamo leggere tutte le celle.

$$2) \text{dtime}(T, x) \leq \text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)}$$

$\text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)}$ è il numero di stati globali possibili di T nel caso in cui non vengono utilizzate più di $\text{dspace}(T, x)$ celle dalla computazione $T(x)$: poiché ogni cella del nastro può contenere un simbolo di Σ oppure il blank, il numero di possibili configurazioni di $\text{dspace}(T, x)$ celle è: $(|\Sigma| + 1)^{\text{dspace}(T, x)}$; per ognuna di queste configurazioni la testina può trovarsi su una qualsiasi delle $\text{dspace}(T, x)$ celle e la macchina può essere in uno qualsiasi dei $|Q|$ stati interni.

$$\text{(Chiamiamo } K(T, x) = \text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)})$$

Una computazione (deterministica) è una successione di stati globali tali che si passa da uno stato globale al successivo eseguendo una quintupla.

Se $T(x)$ durasse più di $K(T, x)$ passi (senza uscire dalle $\text{dspace}(T, x)$ celle) allora sarebbe una successione di stati globali contenente almeno due volte uno stesso stato globale:

$$s_{G_1} \rightarrow s_{G_2} \rightarrow \dots \rightarrow s_{G_{K(T,x)}} \rightarrow \dots \rightarrow s_{G_{K(T,x)}}$$

Ma T è deterministica: a partire da s_n , è possibile eseguire un'unica quintupla ed essa viene eseguita tutte le volte in cui $T(x)$ si trova in s_n . $T(x)$ sarebbe in loop (contro l'ipotesi che terminal).

Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione totale e calcolabile, sia Σ un alfabeto finito e sia $x \in \Sigma^*$ indichiamo con $|x|$ il numero di caratteri di x (non cardinalità non è un insieme bensì lunghezza)

Un linguaggio $L \subseteq \Sigma^*$ è deciso in tempo (spazio) deterministico $f(n)$ se:

$\exists T$ che decide $L: \forall x \in \Sigma^* [dtime(T, x) \leq f(|x|)] \quad (dspace(T, x) \leq f(|x|))$.

Un linguaggio $L \subseteq \Sigma^*$ è accettato in tempo (spazio) non deterministico $f(n)$ se:

$\exists NT$ che accetta $L: \forall x \in L [ntime(NT, x) \leq f(|x|)] \quad (nspace(NT, x) \leq f(|x|))$.

Un linguaggio $L \subseteq \Sigma^*$ è deciso in tempo (spazio) non deterministico $f(n)$ se:

$\exists NT$ che decide $L: \forall x \in \Sigma^* [ntime(NT, x) \leq f(|x|)] \quad (nspace(NT, x) \leq f(|x|))$.

Ogni qualvolta una funzione totale e calcolabile limita la quantità di risorse disponibili al fine di accettare le parole di un linguaggio, i concetti di accettabilità e di decidibilità coincidono.

Teo 6.2: Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione totale e calcolabile. Se $L \subseteq \Sigma^*$ è accettata da una macchina di Turing non deterministica NT tale che, $\forall x \in L ntime(NT, x) \leq f(|x|)$ allora L è decidibile.

dim.

f è totale e calcolabile $\Rightarrow \exists T_f: \forall n \in \mathbb{N} [\cup_T(n) = f(n)]$, con $f(n)$ scritto in unario sul nastro di output.

Costruiamo una nuova macchina non deterministica NT' , a tre nastri che decide L :

$\forall x \in \Sigma^* :$

1) $NT'(x)$ scrive $|x|$ in unario sul N_2 : scorre il primo nastro e finché legge un carattere diverso da \square , viene scritto 1 su N_2 , facendo avanzare le testine di entrambi. Quando su N_1 viene letto \square , le testine dei primi due nastri vengono riposizionate sul carattere più a sx.

2) simula $T_f(|x|)$ utilizzando N_2 come nastro di input e N_3 come nastro di output; al termine della computazione sul terzo nastro si troverà scritto $f(|x|)$ in unario e posiziona la testina di N_3 sul carattere più a sinistra.

3) simula $NT(x)$ utilizzando N_1 come nastro di input e N_3 come contatore del numero di istruzioni eseguite: fino a quando viene letto 1 e non è stato raggiunto uno stato finale, viene eseguita un'istruzione di $NT(x)$ e la testina di N_3 viene spostata di una posizione a destra. Se la macchina NT raggiunge lo stato di accettazione o rigetto, la computazione $NT'(x)$ termina nel medesimo stato. Se viene letto 0 su N_3 , la computazione $NT'(x)$ termina nello stato di rigetto.

Le computazioni di NT' terminano sempre, inoltre:

- se $x \in L$ allora $NT(x)$ accetta in al più $f(|x|)$ passi e quindi $NT'(x)$ accetta
- se $x \notin L$ allora o $NT(x)$ rigetta in al più $f(|x|)$ passi e quindi $NT'(x)$ rigetta oppure $NT(x)$ non termina entro $f(|x|)$ passi e quindi $NT'(x)$ rigetta.

NT' decide L e quindi L è decidibile. \square

Non possiamo concludere che L è decidibile in tempo (spazio) non deterministico $f(n)$, perché sappiamo solo che $T_f(|x|)$ termina, ma non in quanto tempo.

Dimostrazione analoga per ntime.

Tutti i modelli di calcolo deterministici sono fra loro polynomialmente correlati:

\forall macchina di Turing T esiste una macchina di Turing T' ed un polinomio p tali che T' risolve lo stesso problema risolto da T e, per ogni x , $dtime(T', x) \leq p(dtime(T, x))$ e $dspace(T', x) \leq p(dspace(T, x))$.

Un problema è trattabile se il tempo necessario a risolverlo è polinomiale (nella dimensione dell'input).

Lec 12

Teo 6.6: Sia $L \subseteq \Sigma^*$ un linguaggio deciso da una macchina di Turing deterministica ad un nastro T : $\forall x \in \Sigma^*$ $d\text{space}(T', x) = s(|x|)$ e sia $K > 0$ una costante. Allora

\exists una macchina di Turing ad un nastro T' che decide L , e $\forall x \in \Sigma^*$,
 $d\text{space}(T', x) \leq \lceil \frac{s(|x|)}{K} \rceil + O(|x|)$. (**compressione lineare**)

L'addendo $O(|x|)$ deriva dal fatto che l'input di T' è lo stesso di T . T' deve inanzitutto codificare in forma compressa il proprio input e poi lavorare sull'alfabeto compresso: l'alfabeto compresso Σ^K (un carattere compresso è una parola di K caratteri di Σ) e che l'alfabeto di T' è $\Sigma^* \cup \Sigma$.

Teo 6.7: Sia $L \subseteq \Sigma^*$ un linguaggio deciso da una macchina di Turing deterministica ad un nastro T : $\forall x \in \Sigma^*$ $d\text{time}(T', x) = t(|x|)$ e sia $K > 0$ una costante. Allora:

\exists una macchina di Turing ad un nastro T' che decide L , e $\forall x \in \Sigma^*$,
 $d\text{time}(T', x) \leq \lceil \frac{t(|x|)}{K} \rceil + O(|x|^2)$.

\exists una macchina di Turing a due nastri T'' che decide L , e $\forall x \in \Sigma^*$,
 $d\text{time}(T'', x) \leq \lceil \frac{t(|x|)}{K} \rceil + O(|x|)$. (**accelerazione lineare**).

(gli addendi $O(|x|)$ e $O(|x|^2)$ derivano dal fatto che, le macchine T' e T'' per poter essere più veloci devono innanzi tutto codificare in forma compressa il proprio input: se la codifica compressa viene scritta su un nastro apposito sono sufficienti $O(|x|)$ passi, se si dispone di un solo nastro occorrono $O(|x|^2)$ passi).

Per i due teoremi, se un linguaggio L è deciso in tempo (spazio) deterministico $f(n)$, allora L è anche deciso in tempo (spazio) deterministico $\frac{f(n)}{K}$ $\forall K > 0$.

Una classe di complessità è definita mediante una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$, chiamata **funzione limite della classe**, e rappresenta una classe di linguaggi decidibili o accettabili da una macchina di Turing che utilizza una quantità di risorse limitata da f ; tuttavia non ha molto senso parlare di "quantità di risorse limitate da f ", perché la quantità di risorse è individuabile solo a meno di costanti moltiplicative. Perciò nelle definizioni si ricorre all'utilizzo della notazione O .

$\text{DTIME}[f(n)] = \{L \subseteq \{0,1\}^*: \exists T \text{ che decide } L \text{ e, } \forall x \in \{0,1\}^* [\text{dtime}(T,x) \in O(f(|x|))]\}$

$\text{NTIME}[f(n)] = \{L \subseteq \{0,1\}^*: \exists NT \text{ che accetta } L \text{ e, } \forall x \in L [\text{ntime}(T,x) \in O(f(|x|))]\}$

$\text{DSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: \exists T \text{ che decide } L \text{ e, } \forall x \in \{0,1\}^* [\text{dspace}(T,x) \in O(f(|x|))]\}$

$\text{NSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: \exists NT \text{ che accetta } L \text{ e, } \forall x \in L [\text{nspace}(T,x) \in O(f(|x|))]\}$

$\text{coDTIME}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{DTIME}[f(n)]\}$

$\text{coNTIME}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{NTIME}[f(n)]\}$

$\text{coDSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{DSPACE}[f(n)]\}$

$\text{coNSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{NSPACE}[f(n)]\}$

L'2 ragione per cui abbiamo linguaggi decisi e accettabili è da ricercarsi nella
asimmetria delle definizioni di accettazione e di rigetto nelle macchine non deterministiche.

(le dim non vanno studiate)

Teo 6.8: $\forall f \text{ totale e calcolabile } f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DTIME}[f(n)] \subseteq \text{NTIME}[f(n)] \subseteq \text{DSPACE}[f(n)] \subseteq \text{NSPACE}[f(n)]$$

Una macchina di Turing deterministica è una particolare macchina di Turing non deterministica avendo grado di non determinismo pari a 1 e, ogni parola decisa in K passi è anche accettata in K passi. □

Teo 6.9: $\forall f \text{ totale e calcolabile } f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DTIME}[f(n)] \subseteq \text{DSPACE}[f(n)] \subseteq \text{NTIME}[f(n)] \subseteq \text{NSPACE}[f(n)]$$

Sia $L \subseteq \{0,1\}^*: L \in \text{DTIME}[f(n)]$ allora $\exists T \text{ che decide } L$:

$$\forall x \in \{0,1\}^* \text{ dtime}(T,x) \in O(f(|x|)).$$

Poiché $\text{dspace}(T,x) \leq \text{dtime}(T,x)$ allora $\text{dspace}(T,x) \in O(f(|x|))$ e che dunque $L \in \text{DSPACE}[f(n)]$. □

Teo 6.10: $\forall f \text{ totale e calcolabile } f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DSPACE}[f(n)] \subseteq \text{DTIME}[2^{O(1)f(n)}] \subseteq \text{NSPACE}[f(n)] \subseteq \text{NTIME}[2^{O(1)f(n)}]$$

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DSPACE}[f(n)]$ allora $\exists T$ che decide L :

$$\forall x \in \{0,1\}^* \text{ dspace}(T,x) \in O(f(|x|)).$$

$$\text{Poiché } \text{dtime}(T,x) \leq \text{dspace}(T,x)|Q|(|\Sigma|_+)^{\text{dspace}(T,x)} = \text{dspace}(T,x)|Q|_3^{\text{dspace}(T,x)}$$

$$= 2^{\log \text{dspace}(T,x)} |Q| [2^{\log 3}]^{\text{dspace}(T,x)} = |Q| 2^{\log \text{dspace}(T,x) + \text{dspace}(T,x) \log 3} \leq |Q| 2^{[\dots + \log 3] \text{dspace}(T,x)}$$

allora $\text{dtime}(T,x) \in O(2^{O(f(|x|))})$ e dunque $L \in \text{DTIME}[2^{O(f(n))}]$. \square

Teo 6.11: $\forall f$ totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DTIME}[f(n)] = \text{coDTIME}[f(n)] \quad \text{e} \quad \text{DSPACE}[f(n)] = \text{coDSPACE}[f(n)]$$

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DTIME}[f(n)]$ allora $\exists T$ che decide L :

$$\forall x \in \{0,1\}^* \text{ dtime}(T,x) \in O(f(|x|)).$$

Poiché T decide L allora $T(x) = q_0$ se $x \in L$, e $T(x) = q_R$ se $x \in \{0,1\}^* - L = L^c$.

Costruiamo una macchina T' identica a T ma con stati di accettazione e rifiuto invertiti. Dunque T' decide L^c e, $\forall x \in \{0,1\}^*$, $\text{dtime}(T',x) \in O(f(|x|))$.

Quindi $L^c \in \text{DTIME}[f(n)]$. Poiché L è un qualunque linguaggio in $\text{DTIME}[f(n)]$ e, quindi, L^c è un qualunque linguaggio in $\text{coDTIME}[f(n)]$, questo significa che:

$$\forall L^c \in \text{coDTIME}[f(n)], L^c \in \text{DTIME}[f(n)] - \text{ossia } \text{coDTIME}[f(n)] \subseteq \text{DTIME}[f(n)],$$

$$\forall L \in \text{DTIME}[f(n)], \text{ poiché } L^c \in \text{DTIME}[f(n)], \text{ allora } L \in \text{coDTIME}[f(n)],$$

$$\text{ossia } \text{DTIME}[f(n)] \subseteq \text{coDTIME}[f(n)]. \quad \square$$

Date $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ due funzioni, $f(n) \in O(g(n))$ se esistono $n_0 \in \mathbb{N}$ e $c \in \mathbb{N}$:

$$\forall n \geq n_0, f(n) \leq cg(n), \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{N}: \forall n \geq n_0 [f(n) \leq cg(n)] \text{ ossia } O(f(n)) \subseteq O(g(n))$$

Teo 6.12: \forall coppie di funzioni $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ tali che $\exists n_0 \in \mathbb{N}: \forall n \geq n_0 [f(n) \leq g(n)]$

$$\text{ossia } f(n) \leq g(n). \quad \text{DTIME}[f(n)] \subseteq \text{DTIME}[g(n)], \quad \text{NTIME}[f(n)] \subseteq \text{NTIME}[g(n)],$$

$$\text{DSPACE}[f(n)] \subseteq \text{DSPACE}[g(n)], \quad \text{NSPACE}[f(n)] \subseteq \text{NSPACE}[g(n)].$$

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DTIME}[f(n)]$ allora $\exists T$ che decide L :

$$\text{dtime}(T,x) \in O(f(|x|)) \leq O(g(|x|)). \quad \text{Questo significa che } L \in \text{DTIME}[g(n)]. \quad \square$$

Di contro, nella definizione di una teoria di complessità in grado di classificare significativamente i linguaggi in classi di complessità crescente, sarebbe auspicabile che $\text{DTIME}[f(n)]$ non fosse

contenuto in $\text{DTIME}[g(n)]$ quando $f(n)$ è molto più grande di $g(n)$ - in particolare quando $f(n) = 2^{g(n)}$. Questo è contraddetto dal prossimo teorema.

Teo 6.13: Esiste una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$:

$$\text{DTIME}\left[2^{f(n)}\right] \subsetneq \text{DTIME}[f(n)]$$

Al fine di non incorre in comportamenti "anomali" del tipo evidenziato nel Gap Theorem, è necessario considerare opportuni sottoinsiemi delle funzioni totali e calcolabili nei quali venga tenuto conto della quantità di risorse che occorrono per calcolarle.

Una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$ è **time-constructible** se esiste una macchina di Turing T di tipo trasduttore che, preso in input un intero n espresso in notazione unaria, scrive sul nastro output il valore $f(n)$ in unario e $\text{dtime}(T, n) \in O(f(n))$.

Una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$ è **space-constructible** se esiste una macchina di Turing T di tipo trasduttore che, preso in input un intero n espresso in notazione unaria, scrive sul nastro output il valore $f(n)$ in unario e $\text{dspace}(T, n) \in O(f(n))$.

Una funzione time- o space-constructible è molto più che una funzione totale e calcolabile, è una funzione che può essere calcolata in tempo proporzionale al suo valore.

Sono time- e space-constructible tutte le funzioni "regolari" quali:

- tutti i polinomi - ossia $f(n) = n^k$ con k costante
- funzioni esponenziali - ossia $f(n) = 2^n$ o $f(n) = n^n$
- tantissime altre

La funzione definita nel gap theorem non è time-constructible.

Teo 6.14: Siano $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ due funzioni tale che f è space-constructible e

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0 \quad \text{allora, } \text{DSPACE}[g(n)] \subset \text{DSPACE}[f(n)], \text{ ossia } \exists L: L \in \text{DSPACE}[f(n)] \text{ e } L \notin \text{DSPACE}[g(n)].$$

Teo 6.15 : Siano $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ due funzioni tale che f è time-constructible e

$$\lim_{n \rightarrow \infty} \frac{g(n) \log g(n)}{f(n)} = 0 \text{ allora, } \text{DTIME}[g(n)] \subset \text{DTIME}[f(n)], \text{ ossia } \exists L: L \in \text{DTIME}[f(n)] \text{ e } L \notin \text{DTIME}[g(n)]$$

Quando f è space- e time-constructible $\text{DTIME}[f(n)]$ non è contenuto in $\text{DTIME}[g(n)]$ quando $f(n)$ è molto più grande di $g(n)$ - ad esempio quando $f(n) = 2^{g(n)}$.

Lez 13

Teo 6.16: Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione time-constructible. Allora $\forall L \in \text{NTIME}[f(n)]$, si ha che L è decidibile in tempo non deterministico in $O(f(n))$.

dim.

Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione time-constructible, allora esiste una macchina di Turing di tipo trasduttore T_f che, avendo scritto sul suo nastro di input $n \in \mathbb{N}$ codificato in unario, in $O(f(n))$ passi scrive sul nastro di output il valore $f(n)$ codificato in unario.

Sia $L \subseteq \Sigma^*$ tale che $L \in \text{NTIME}[f(n)]$, allora esiste una macchina di Turing non deterministica NT che accetta L : $\forall x \in L, \text{ntime}(NT, x) \in O(f(|x|))$. Costruiamo, ora, a partire da T_f e NT, la macchina NT' descritta nella dimostrazione del Teo 6.2 che decide L :

- 1) Scrive su N_2 in unario, termina in $O(|x|)$ passi.
- 2) Simula $T_f(x)$ e scrive risultato in unario su N_3 termina in $O(f(|x|))$ passi.
- 3) finché legge 1 su N_3 esegue un'istruzione di NT(x), termina in $O(f(|x|))$ passi.

Se termina in q_a o q_r NT' termina nel medesimo stato, se viene letto 0 su N_3

Dunque NT' decide L , e $\forall x \in \Sigma^*, \text{ntime}(NT', x) \in O(f(|x|))$. \square

Analogo per f space-constructible.

Teo 6.17: $\forall f$ time-constructible $f: \mathbb{N} \rightarrow \mathbb{N}$,

$$\text{NTIME}[f(n)] \subseteq \text{DTIME}[2^{O(f(n))}]$$

dim.

Sia $L \subseteq \Sigma^*$ tale che $L \in \text{NTIME}[f(n)]$; allora esistono una macchina di Turing non deterministica NT che accetta L e una costante h tali che, $\forall x \in L, \text{ntime}(NT, x) \leq h f(|x|)$. Poiché f è time-constructible esiste una macchina di Turing T_f che, con input la rappresentazione in unario di un intero n , calcola la rappresentazione in unario di $f(n)$ in tempo $O(f(n))$.

Indichiamo con K il grado di non determinismo di NT e utilizziamo la tecnica della simulazione per definire una macchina di Turing deterministica T che simuli il comportamento di NT: su input x , T simula tutte le computazioni deterministiche di NT(x) di lunghezza $hf(|x|)$:

- 1) Simula la computazione $T_f(|x|)$: $\forall x$, scrive sul nastro N_2 un carattere "1" e in seguito, calcola $f(|x|)$ scrivendolo su N_3 . Infine concatena h volte il contenuto

di N_3 ottenendo il valore $hf(|x|)$.

2) Simula, una alla volta, tutte le computazioni deterministiche di $NT(x)$ di lunghezza $hf(|x|)$ utilizzando, per ciascuna computazione, la posizione della testina sul nastro N_3 come contatore.

Se $x \in L$, $ntime(NT, x) \leq hf(|x|)$ allora o in $hf(|x|)$ passi $NT(x)$ termina nello stato di accettazione oppure $x \notin L$. Quindi, se dopo aver simulato tutte le computazioni deterministiche di $NT(x)$ di lunghezza $hf(|x|)$, T non ha raggiunto lo stato di accettazione, allora può entrare nello stato di rifiuto. Questo prova che T decide L .

Detto K il grado di non determinismo di NT , il numero di computazioni deterministiche di $NT(x)$ di lunghezza $hf(|x|)$ è $K^{hf(|x|)}$ e ciascuna di esse viene simulata in $O(f(|x|))$ passi. Poiché il passo i) descritto sopra richiede $O(f(|x|))$ passi, allora:

$$dtime(T, x) \in O(f(|x|) K^{hf(|x|)}) \subseteq O(2^{O(f(|x|)))})$$

$$\exists T_i : \forall x \quad O_{T_i}(x) = O_T(x) \text{ e } dtime(T_i, x) \leq dtime(T, x) \in O(2^{O(f(|x|)))})$$

Quindi $L \in \text{DTIME}[2^{O(f(|x|))}]$. \square

Altre classi di complessità:

$P = \bigcup_{K \in \mathbb{N}} \text{DTIME}[n^K]$: è la classe dei linguaggi decidibili in tempo deterministico polinomiale;

$NP = \bigcup_{K \in \mathbb{N}} \text{NTIME}[n^K]$: è la classe dei linguaggi accettabili in tempo non deterministico polinomiale;

$PSPACE = \bigcup_{K \in \mathbb{N}} \text{DSPACE}[n^K]$: è la classe dei linguaggi decidibili in spazio deterministico polinomiale;

$NPSPACE = \bigcup_{K \in \mathbb{N}} \text{NSPACE}[n^K]$: è la classe dei linguaggi accettabili in spazio non deterministico polinomiale;

$EXPTIME = \bigcup_{K \in \mathbb{N}} \text{DTIME}[2^{n^K}]$: è la classe dei linguaggi decidibili in tempo deterministico esponenziale ove l'esponente che descrive la funzione limite è un polinomio;

$NEXPTIME = \bigcup_{K \in \mathbb{N}} \text{NTIME}[2^{n^K}]$: è la classe dei linguaggi decidibili in tempo non deterministico esponenziale ove l'esponente che descrive la funzione limite è un polinomio;

Corollario 6.1: NP, NPSPACE e NEXPTIME sono le classi dei linguaggi decidibili, rispettivamente, in tempo non deterministico polinomiale, in spazio non deterministico polinomiale e in tempo non deterministico esponenziale.

Corollario 6.2: Valgono le relazioni di inclusione:

$$P \subseteq NP, PSPACE \subseteq NPSPACE \quad (\text{conseguenza del Teo 6.8})$$

$$P \subseteq PSPACE, NP \subseteq NPSPACE \quad (\text{conseguenza del Teo 6.9})$$

$$PSPACE \subseteq EXPTIME, NPSPACE \subseteq NEXPTIME \quad (\text{conseguenza del Teo 6.10})$$

$$NP \subseteq EXPTIME \quad (\text{conseguenza del Teo 6.17})$$

Possiamo indicare anche i loro complementi:

$$coP = \{L \subseteq \Sigma^*: \Sigma \text{ è un alfabeto finito e } L^c \in P\}$$

$$coNP = \{L \subseteq \Sigma^*: \Sigma \text{ è un alfabeto finito e } L^c \in NP\}$$

Corollario 6.3: $coP = P$.

Abbiamo dimostrato, fino ad ora, inclusioni non proprie fra classi di complessità; per ciascuna di esse non siamo in grado di dimostrare né l'inclusione propria né la coincidenza delle due classi che la costituiscono.

Teo 6.18: $P \subseteq EXPTIME$.

(Come conseguenza del Teo 6.15 (Teorema di gerarchia temporale))

Teo 6.19: $PSPACE = NPSPACE$.

Lez 14

La maggior parte delle relazioni fra classi di complessità che abbiamo visto, fino ad ora, sono inclusioni improprie. (a parte $P \subseteq EXPTIME$ e $PSPACE = NPSPACE$)

Ossia, non siamo in grado di dimostrare né l'inclusione propria né la coincidenza delle due classi che le costituiscono.

Una possibile tecnica di dimostrazione che due classi di complessità C_1 e C_2 tali che $C_1 \subseteq C_2$ sono distinte consiste nell'individuare un linguaggio **separatore**, ossia, un linguaggio $L \in C_2 - C_1$. La dimostrazione che $L \in C_1$ richiede di mostrare che nessuna macchina di Turing (det. o non det. in base al tipo di classe C_1) è in grado di decidere o accettare il linguaggio L utilizzando la quantità di risorse indicata nella definizione di C_1 .

È possibile individuare i linguaggi **candidati** ad essere separatori fra due classi. Utilizzando una nozione collegata al concetto di riducibilità funzionale, la nozione di linguaggio **completo** per una data classe. Per definire tale nozione, osserviamo che il concetto di riducibilità funzionale può essere specializzato richiedendo che la funzione che trasforma parole di un linguaggio in parole di un altro linguaggio soddisfi qualche predicato π : ad esempio, che la parola a cui applichiamo la funzione di riduzione e la parola in cui viene trasformata abbiano la stessa lunghezza. Chiamiamo π -riduzione una riduzione funzionale che soddisfa il predicato π , e scriviamo $L_1 \leq_\pi L_2$.

Un altro strumento che permette di individuare i linguaggi separatori fra due classi di complessità è basato sui seguenti due concetti che si riferiscono alle π -riduzioni:

chiusura di una classe rispetto a π -riduzione

completezza di un linguaggio per una classe rispetto a una π -riduzione

Una classe di complessità C è **chiusa** rispetto ad una generica π -riduzione se, per ogni coppia di linguaggi L_1 e L_2 tali che $L_1 \leq_\pi L_2$ e $L_2 \in C$, si ha che $L_1 \in C$.

Sia C una classe di complessità di linguaggi e sia \leq_{π} una generica π -riduzione.

Un linguaggio $L \subseteq \Sigma^*$ è C -completo rispetto alla π -riducibilità se:

a) $L \in C$ e

b) $\forall L' \in C$, vale che $L' \leq_{\pi} L$

Abbiamo due classi di complessità C_1 e C_2 tali che $C_1 \subseteq C_2$ e sappiamo che C_1 è chiusa rispetto ad una qualche π -riduzione: allora $\forall L_1 \in C_1$: $L_1 \leq_{\pi} L_2$ e $L_2 \in C_1$ si ha che $L_1 \in C_1$.

Se per caso troviamo un linguaggio L C_2 -completo rispetto \leq_{π} ossia, $L \in C_2$ e

\forall altro $L_0 \in C_2$ [$L_0 \leq_{\pi} L$] e se dimostriamo che $L \in C_1$ abbiamo che:

\forall altro $L_0 \in C_2$ [$L_0 \leq_{\pi} L$] e $L \in C_1$ allora per la chiusura di C_1 rispetto alla π -riduzione,

\forall altro $L_0 \in C_2$ [$L_0 \in C_1$] e dunque $C_1 = C_2$.

Ne consegue che se $C_1 \subseteq C_2$ e L è C_2 -completo e se qualcuno riuscisse a dimostrare che $C_1 \neq C_2$ allora sapremmo che $L \in C_1$.

Ne consegue anche che se $C_1 \subseteq C_2$ e C_1 è chiusa rispetto ad una qualche π -riduzione

se per caso trovassimo un linguaggio L C_2 -completo rispetto a \leq_{π} e se qualcuno riuscisse a dimostrare che $C_1 \neq C_2$ allora sapremmo che $L \in C_2$.

Teo 6.20: Siano C e C' due classi di complessità tali che $C \subseteq C'$. Se C è chiusa rispetto ad una π -riduzione allora, per ogni linguaggio L che sia C -complemento rispetto a tale π -riduzione $L \in C$ se e solo se $C = C'$

dim.

Se $C = C'$ allora $L \in C'$.

Viceversa, supponiamo che $L \in C'$, poiché L è C completo rispetto alla π -riducibilità, allora, $\forall L' \in C$, $L' \leq_{\pi} L$. Poiché C è chiusa rispetto alla π -riduzione, questo implica che, $\forall L' \in C$, $L' \in C'$: quindi $C = C'$. □

Siano $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$ due linguaggi: diciamo che L_1 è **polynomialmente riducibile** ad L_2 , e scriviamo $L_1 \leq_p L_2$, se esiste una funzione totale e calcolabile $f: \Sigma_1^* \rightarrow \Sigma_2^*$: $f \in FP$ e $\forall x \in \Sigma_1^* [x \in L_1 \Leftrightarrow f(x) \in L_2]$.

Teo 6.21: La classe P è chiusa rispetto alla riducibilità polinomiale.

dim.

Siano $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$ due linguaggi: $L_1 \leq L_2$ e $L_2 \in P$. Indichiamo con $f: \Sigma_1 \rightarrow \Sigma_2$ la funzione in FP che riduce L_1 a L_2 , con T_f la macchina di Turing che calcola f in tempo polinomiale, e con T_2 la macchina di Turing deterministica che decide L_2 in tempo polinomiale.

Poiché T_f e T_2 operano in tempo polinomiale, esistono due costanti $h, k \in \mathbb{N}$ tali che $\forall x \in \Sigma_1^*$ e $\forall y \in \Sigma_2^*$, $\text{dtime}(T_f, x) \leq |x|^h$ e $\text{dtime}(T_2, y) \leq |y|^k$.

Combiniamo ora le due macchine per ottenere una macchina di Turing deterministica T_1 che decide L_1 . Opera nella seguente maniera:

- 1) simula $T_f(x)$ e scrive $f(x)$ sul secondo nastro.
- 2) simula $T_2(f(x))$ sul secondo nastro.

Se $T_2(f(x))$ accetta $T_1(x)$ accetta, se $T_2(f(x))$ rigetta allora $T_1(x)$ rigetta ($T_2(f(x))$ non può non terminare perché T_2 decide L_2): $T_1(x)$ accetta sse $T_2(f(x))$ accetta, ossia, $f(x) \in L_2$.

Ricordiamo che f è una riduzione da L_1 a L_2 e, quindi, $f(x) \in L_2$ sse $x \in L_1$. T_1 termina sse input x è $T_1(x)$ accetta sse $x \in L_1$, ossia T_1 decide L_1 .

La simulazione $T_f(x)$ richiede $\text{dtime}(T_f, x) \leq |x|^h$ passi e la simulazione di $T_2(f(x))$ richiede $\text{dtime}(T_2, f(x)) \leq |f(x)|^k$ passi: dunque, $\text{dtime}(T_1, x) \leq |x|^h + |f(x)|^k$. Poiché $\text{dtime}(T_f, x) \leq |x|^h$ e T_f deve almeno scrivere il suo output $f(x)$, allora $|f(x)| \leq |x|^h$

quindi $\text{dtime}(T_1, x) \leq |x|^h + |f(x)|^k \leq |x|^h + (|x|^h)^k = |x|^h + |x|^{hk}$

poiché h e k sono costanti $L_1 \in P$.

Teo 6.22: Le classi NP, PSPACE, EXPTIME, NEXPTIME sono chiuse rispetto alla riducibilità polinomiale.

Corollario 6.4: Se $P \neq NP$ allora, per ogni linguaggio L NP-completo, $L \in P$.

dim.

Supponiamo L sia un linguaggio NP-completo e che $L \in P$. Poiché L è NP-completo, per ogni linguaggio $L' \in NP$, $L' \leq L$;

Se $L \in P$, poiché P è chiusa rispetto a \leq , questo implica che, per ogni $L' \in NP$,
 $L' \in P$. Ossia, $P = NP$, contraddicendo l'ipotesi.

Lec 15

Alle classi introdotte consideriamo i corrispondenti complementi:

$$\text{coP} = \{L \subseteq \{0,1\}^*: L^c \in P\}$$

$$\text{coNP} = \{L \subseteq \{0,1\}^*: L^c \in NP\}$$

allo stesso modo le classi coEXPTIME , coSPACE e in generale coDTIME , coDSPACE , coNTIME , coNSPACE .

Nella definizione delle classi di complessità complemento non viene specificato come vengono decisi (o accettati) i linguaggi che vi appartengono, invece viene specificato come vengono decisi (o accettati) i complementi dei linguaggi che vi appartengono.

Potrebbe sembrare che, analogamente alla classe P che coincide con il suo complemento, le due classi NP e $coNP$ coincidono. Iniziamo con il ricordare la assimmetria delle definizioni di accettazione e rigetto di una macchina di Turing non deterministica: una parola viene accettata se esiste una computazione deterministica che termina nello stato di accettazione, una parola viene rifiutata se ogni computazione deterministica termina nello stato di rigetto.

Mostriamo che per decidere L^c non è sufficiente, come nel caso deterministico (b.ii), invertire gli stati di accettazione e di rigetto di NT . Senza perdita di generalità, possiamo supporre che, $\forall y \in \Sigma^*$ una delle computazioni deterministiche di $NT(y)$ termini nello stato di rigetto, se così non fosse, potremmo costruire una macchina NT' identica a NT aggiungendo all'insieme delle quintuple le seguenti quintuple: $\langle q_0, a, a, q_f, f \rangle \quad \forall a \in \Sigma$.

Qualunque sia $y \in \Sigma^*$, una computazione di $NT'(y)$ termina in q_f e poiché le altre quintuple sono identiche alle quintuple di NT , $\forall y \in \Sigma^*$:

- se esiste una computazione deterministica di $NT(y)$ che termina nello stato di accettazione, allora esiste una computazione deterministica di $NT'(y)$ che termina nello stato di accettazione
- se nessuna computazione deterministica di $NT(y)$ termina nello stato di accettazione, allora nessuna computazione deterministica di $NT'(y)$ termina nello stato di accettazione

Questo prova che $\forall y \in \Sigma^*$, $NT'(y)$ accetta L se $NT(y)$ accetta.

Costruiamo quindi \overline{NT} identico a NT con gli stati di accettazione e rigetto invertiti.
Supponendo che ogni $nt(y)$ ha un ap-

Affinché \overline{NT} accetti una parola $x \in \Sigma^*$ una delle sue computazioni deterministiche deve

terminare nello stato di accettazione.

Sia $x \in L$ poiché $x \in L$ una computazione deterministica di $NT(x)$ termina nello stato di accettazione e in virtù dell'assunzione fatta sopra, una computazione deterministica di $NT(x)$ termina nello stato di rigetto, quindi, la corrispondente computazione $\bar{NT}(x)$ terminerà nello stato di accettazione inducendo così \bar{NT} ad accettare $x \in L^c$. Questo prova che il linguaggio accettato da \bar{NT} non è L^c . (perché NT accetta qualunque sia x)

Non possiamo quindi affermare che $\text{coNP} = \text{NP}$.

II congettura della teoria della complessità computazionale: $\text{coNP} \neq \text{NP}$

Le due congetture non sono totalmente indipendenti:

Teo 6.23: Se $\text{coNP} \neq \text{NP}$, allora $P \neq NP$

dim.

Supponiamo che $P = NP$ allora $\text{coP} = \text{coNP}$ ma $\text{coP} = P$ allora $\text{coNP} = \text{coP} = P = NP$ □

Non è stata dimostrata l'implicazione opposta perciò $\text{coNP} \neq \text{NP}$ è una congettura.

Teo 6.24: La classe coNP è chiusa rispetto alla riducibilità polinomiale.

dim.

poiché NP è chiusa rispetto alla riducibilità polinomiale, $\forall L_2 \in NP \Rightarrow \forall L_1 : L_1 \leq L_2$
[$L_1 \in NP$] $\Rightarrow \forall L_2 \in \text{coNP} \Rightarrow \forall L_1 : L_1 \leq L_2$ [$L_1 \in \text{coNP}$] □

Teo 6.25: Un linguaggio L è NP-completo sse L^c è coNP-completo

dim.

\Rightarrow Se L è NP-completo \Rightarrow

1) $L \in NP \Rightarrow L^c \in \text{coNP}$

2) $\forall L' \in NP [L \leq L']$

$\forall L' \in NP$ esiste una funzione $f_{L'} : \sum^* \rightarrow \sum^*$ tale che $f_{L'} \in FP$ e

$\forall x \in \sum^*, [x \in L' \text{ sse } f_{L'}(x) \in L]$

$\Rightarrow \forall x \in \sum^*, [x \notin L' \text{ sse } f_{L'}(x) \notin L]$

$\Rightarrow \forall x \in \sum^*, [x \in L^c \text{ sse } f_{L'}(x) \in L]$

quindi L^c è completo per coNP

\Leftrightarrow) Se L^c è coNP-completo \Rightarrow

1) $L^c \in \text{coNP} \Rightarrow L \in \text{NP}$

2) $\forall L' \in \text{coNP} [L' \leq L]$

$\forall L' \in \text{coNP}$ esiste una funzione $f_{L'}: \Sigma^* \rightarrow \Sigma^*$ tale che $f_{L'} \in \text{FP}$ e

$$\forall x \in \Sigma^*, [x \in L' \iff f_{L'}(x) \in L^c]$$

$$\Rightarrow \forall x \in \Sigma^*, [x \notin L' \iff f_{L'}(x) \notin L^c]$$

$$\Rightarrow \forall x \in \Sigma^*, [x \in L' \iff f_{L'}(x) \in L]$$

quindi L è completo per NP \square

Teo 6.26: $\forall L \in \text{NP}$ completo [se $L \in \text{coNP}$ allora $\text{NP} = \text{coNP}$]

dim.

Se $L \in \text{NP} \cap \text{coNP}$ allora anche $L^c \in \text{NP} \cap \text{coNP}$, poiché L è NP-completo \Rightarrow

L^c è coNP-completo quindi $\forall L' \in \text{coNP} [L' \leq L^c]$. Ma NP è chiusa rispetto \leq e $L^c \in \text{NP}$

$\Rightarrow \forall L' \in \text{coNP} [L' \in \text{NP}]$ quindi $\text{coNP} \subseteq \text{NP}$.

Poiché L è NP-completo $\Rightarrow \forall L'' \in \text{NP} [L'' \leq L]$ ma $L \in \text{NP} \cap \text{coNP}$, quindi $L \in \text{coNP}$. Ma coNP è

chiusa rispetto \leq $\Rightarrow \forall L'' \in \text{NP} [L'' \in \text{coNP}]$ quindi $\text{NP} \subseteq \text{coNP}$

Dunque $\text{NP} = \text{coNP}$ \square

Lez 16

La struttura di un problema è la seguente:

Dati: un insieme di oggetti conosciuti - l'insieme dei dati che costituisce un'istanza del problema - all'interno di un secondo insieme di oggetti - l'insieme delle soluzioni possibili - cercare gli oggetti che soddisfino certi vincoli e, sulla base degli oggetti trovati, fornire un qualche tipo di risposta.

Un problema è una quintupla $\langle I, R, S, \eta, p \rangle$ in cui I è l'insieme delle istanze, R è l'insieme delle risposte, S è la funzione che specifica, per una data istanza, l'insieme delle soluzioni possibili per quell'istanza, $\eta: S(I) \rightarrow 2^{S(I)}$ è la funzione che ricava dall'insieme delle soluzioni possibili per un'istanza del problema un insieme di soluzioni effettive di quella istanza e $p: I \times \eta(S(I)) \rightarrow R$ è la richiesta del problema.

La risposta ad una istanza $x \in I$ di un problema è $p(x, \eta(S(x)))$.

es.

dato un numero $n \in \mathbb{N}$... [domanda sui divisori del numero]

$$I = \mathbb{N}$$

$$S(n) = \{x \in \mathbb{N} : x \leq n\}$$

$$\eta(n, S(n)) = \{x \in S(n) : x \text{ è divisore di } n\}$$

1) elencare tutti i divisori di n , $p: \eta(n, S(n)) \rightarrow R$

$$R = \eta(n, S(n)) \text{ e } p(\eta(n, S(n))) = \eta(n, S(n))$$

2) decidere se n è un numero primo

$$R = \{0, 1\} \quad p(\eta(n, S(n))) = [\eta(n, S(n)) = \{1, n\}]$$

3) calcolare un divisore non banale di n

$$R = \eta(n, S(n)) - \{1, n\} \quad p(\eta(n, S(n))) = x \in R$$

4) calcolare il più grande divisore non banale di n

$$R = \max \{\eta(n, S(n)) - \{1, n\}\} \quad p(\eta(n, S(n))) = R$$

Possiamo definire diversi tipi di funzione p rispetto allo stesso insieme di istanze e rispetto alla stessa definizione di soluzioni effettive.

1) è un problema di **enumerazione**.

2) è un problema di **decisione**. (decisionali)

3) è un problema di **ricerca**.

4) è un problema di **ottimizzazione**.

Notiamo come solo per i problemi di decisione viene utilizzata una macchina di Turing di tipo riconoscitore, mentre per gli altri un trasduttore.

Nel caso di problemi decisionali, seppiamo che $R = \{\text{vero, falso}\}$, questo significa che P è un predicato, ossia, una funzione booleana. Allora possiamo assumere π, P, R in unico predicato $\pi: \pi(x, S(x)) = \text{vero} \Leftrightarrow$ l'insieme delle soluzioni possibili per x soddisfa i vincoli del problema. Quindi un problema decisionale è descritto da $\langle I, S, \pi \rangle$.

es: Dati un grafo non orientato G , una coppia di nodi $s \neq t$, e un intero K , decidere se esiste in G un percorso da s a t di lunghezza $= K$

$I = \{\langle G, s, t, K \rangle\}$: G grafo non orientato $\wedge s, t$ sono due nodi di $G \wedge K \in \mathbb{N}$

$S(G, s, t, K) = \{u_0, \dots, u_K\}$: per $i=0, \dots, K$, u_i è un nodo del grafo

$\pi(G, s, t, K, S(G, s, t, K)) = \exists \langle u_0, \dots, u_K \rangle \in S(G, s, t, K) : s = u_0 \wedge t = u_K \wedge$

$\forall i=0, \dots, K-1, [(u_i, u_{i+1}) \text{ è un arco del grafo}]$

3SAT

Dati un insieme X di variabili booleane ed un predicato f , definito sulle variabili X in forma 3CNF decidere se f è soddisfacibile.

$X = \{x_1, \dots, x_n\}$

$f = c_1 \wedge \dots \wedge c_m$ clausola $c_j = v$ di 3 letterali (x_n, \bar{x}_m)

$I_{3SAT} = \{\langle f, X \rangle : f \text{ è in 3CNF}\}$

$S_{3SAT}(f, X) = \{c_i : X \rightarrow \{0, 1\}\}$

$\pi_{3SAT}(f, S_{3SAT}(f, X)) = \exists c_i \in S_{3SAT}(f, X) : f(c_1(x_1), \dots, c_l(x_n)) = \text{vero}$

Per poter utilizzare una macchina di Turing per risolvere un problema decisionale abbiamo bisogno di trasformare le istanze di quel problema in parole. Occorre codificare le istanze di un problema decisionale.

Per il problema 3 sì abbiamo due possibilità:

- 1) codifichiamo la struttura di f
- 2) codifichiamo il "significato" di f

Codifica X_1 : Sia $\Sigma = \{0, 1, 2, 3, 4\}$, codifichiamo X ; con n caratteri il cui unico "1" è quello in posizione i , rappresentiamo un letterale in una clausola mediante la rappresentazione della variabile corrispondente al letterale preceduta da "0" se il letterale è la variabile non negata, da "1" se il letterale è la variabile negata, gli v sono rappresentati da "3" mentre gli $\neg v$ da "4", premettiamo alla codifica tanti "1" quanti gli elementi di X seguito dal carattere separatore "2".

$$f(x_1, x_2, x_3) = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$$X_1(f, \{x_1, x_2, x_3\}) = 111211003001030001401003101030001$$

Codifica X_2 : codifichiamo le istanze di 3SAT, sia $\Sigma = \{0, 1, 2\}$, ogni f è identificata univocamente dai valori che assume su ogni possibile assegnazione di verità delle sue variabili.

$$f(x_1, x_2, x_3) = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

x_1	x_2	x_3	f
0	0	0	1
0	0	1	1
0	1	0	0
...

Scriviamo le righe della tavola separate da un "2"

$$X_2(f, \{x_1, x_2, x_3\}) = 000120011201002\dots$$

Consideriamo il seguente algoritmo che verifica, data $\langle f, X \rangle \in I_{3SAT}$ se f è soddisfacibile:

- 1) calcola $n = |X|$
- 2) \forall assegnazione di verità a all'insieme delle n variabili in X : verifica se $f(a(X)) = \text{vero}$ e, in tal caso termina nello stato di accettazione q_A ;
- 3) termina nello stato di rifiuto q_R .

Vediamo come si comporta l'algoritmo con entrambe le codifiche.

Se $\langle f, X \rangle$ è codificata mediante X_1 : il numero n viene individuato leggendo i primi n caratteri della parola $X_1(f)$; poi, per ogni assegnazione di verità alle n variabili, in un numero di passi proporzionale alla lunghezza di $X_1(f)$ si verifica se f è soddisfatta da tale assegnazione. Poiché il numero di assegnazioni di verità ad n variabili booleane è pari a 2^n , in un numero di passi proporzionale a $n + |X_1(f)|_2$ l'algoritmo verifica se f è soddisfacibile. Se n è il numero di variabili su cui f è definita, il numero di clausole a 3 variabili è limitato da $(2n)^3$, qualunque sia f :

$$|X_1(f, X)| \leq n + 1 + 3m(2n)^3 \leq 26n^4$$

L'algoritmo richiede tempo proporzionale a:

$$n + |X_1(f, X)|_2 \geq \frac{\sqrt[4]{|X_1(f, X)|}}{26} + |X_1(f, X)|_2 \frac{\sqrt[4]{|X_1(f, X)|}}{26}$$

ossia tempo esponenziale nella lunghezza dell'input.

Se $\langle f, X \rangle$ è codificata mediante X_2 : il numero n viene individuato leggendo la parola $X_2(f, X)$ fino a quando non viene incontrato il carattere "2", poiché f è codificata rappresentando tutte le possibili assegnazioni di verità, viene deciso se la funzione è soddisfacibile semplicemente scandendo $X_2(f, X)$: detto m il numero di clausole di cui è costituita f , f è soddisfacibile se viene trovato un "1" in posizione $1(n+1) \dots m(n+1)$. L'algoritmo verifica se $\langle f, X \rangle$ è soddisfacibile in un numero di passi proporzionale a:

$$|X_2(f, X)| + n \leq 2|X_2(f, X)|$$

un numero di passi polinomiale nella lunghezza dell'input.

La ragione per cui l'algoritmo opera in tempo polinomiale in $|X_2(f)|$ è il fatto che $X_2(f)$ rappresenta esplicitamente tutte le assegnazioni di verità per f , contrariamente per X_1 , che le rappresenta in forma implicita. $|X_1(f)| \leq 26n^4$, $|X_2(f)| = n + 1 + 2^{(n+1)}$.

Possiamo affermare che X_2 è irragionerdamente lunga.

Sia $T = \langle I_r, S_r, \Pi_r \rangle$ un problema decisionale, e sia X una codifica per I_r . La

codifica X è ragionevole per T se, per ogni altra codifica X' per T_P esiste un intero $K \in \mathbb{N}$ tale che, $\forall x \in I_P$:

$$|X(x)| \in O(|X'(x)|^k).$$

Possiamo anche dire che X non è una codifica ragionevole per T se esistono un'altra codifica X' per T_P e una funzione "più che polinomiale" F tale che:

$$|X(x)| \in \Omega(F(|X'(x)|^k)).$$

Se X_1 e X_2 sono due codifiche ragionevoli per un problema T , allora esse sono polinomialmente correlate, ossia, esistono due interi $h, k \in \mathbb{N}$ tali che, $\forall x \in I_P$ $|X_1(x)| \in O(|X_2(x)|^h)$ e $|X_2(x)| \in O(|X_1(x)|^k)$.

Lec 17

Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \Pi_\Gamma \rangle$ un problema decisionale, e sia $X: I_\Gamma \rightarrow \Sigma^*$ una codifica (ragionevole) per Γ . La codifica X partiziona Σ^* in tre sottoinsiemi di parole:

- l'insieme Y_Γ delle parole che codificano istanze sì di Γ .
- l'insieme N_Γ delle parole che codificano istanze no di Γ .
- l'insieme delle parole che non codificano istanze di Γ .

Il linguaggio associato a Γ mediante la codifica X è il sottoinsieme $L_\Gamma(X)$ di Σ^* contenente le parole che codificano l'insieme Y_Γ , ossia:

$$L_\Gamma(X) = \{x \in \Sigma^*: \exists y \in I_\Gamma [x = X(y) \wedge \Pi_\Gamma(y, S_\Gamma(y))]\}$$

Il linguaggio delle istanze di Γ :

$$X(I_\Gamma) = \{x \in \Sigma^*: \exists y \in I_\Gamma [x = X(y)]\}$$

X è una codifica di I_Γ , quindi, $X(y) \neq X(z)$ se $y, z \in I_\Gamma$ sono due istanze differenti, possiamo definire il linguaggio $L_\Gamma(X)$ anche nella maniera seguente:

$$L_\Gamma(X) = \{x \in \Sigma^*: x \in X(I_\Gamma) \wedge \Pi_\Gamma(X^{-1}(x), S_\Gamma(X^{-1}(x)))\}$$

Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \Pi_\Gamma \rangle$ un problema decisionale e sia \mathcal{C} una classe di complessità (spaziale o temporale, deterministica o non). Diciamo che $\Gamma \in \mathcal{C}$ se esiste una codifica ragionevole $X: I_\Gamma \rightarrow \Sigma^*$ per Γ tale che $L_\Gamma(X) \in \mathcal{C}$.

es:

Consideriamo il seguente problema decisionale Γ_{PNC} :

Dato un grafo non orientato $G = (V, E)$ che contiene un ciclo che passe una ed una sola volta per ciascuno dei suoi nodi, siano dati due suoi nodi $s, t \in V$; decidere se esiste in G un percorso $s \rightarrow t$.

$$\Gamma_{\text{PNC}} = \langle I_{\Gamma_{\text{PNC}}}, S_{\Gamma_{\text{PNC}}}, \Pi_{\Gamma_{\text{PNC}}} \rangle$$

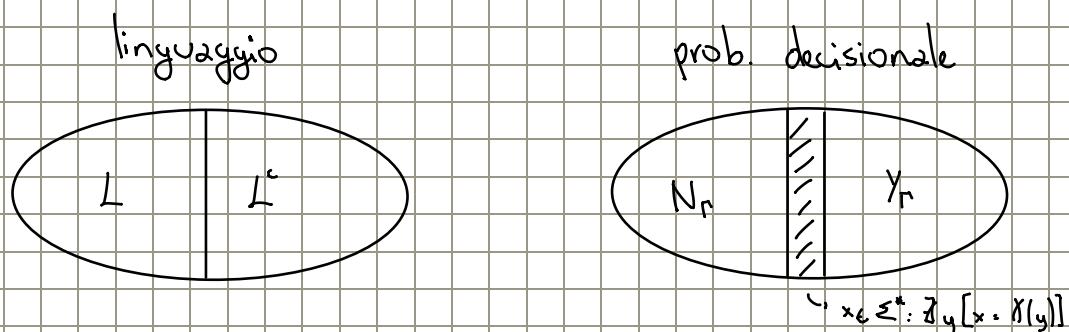
$$I_{\Gamma_{\text{PNC}}} = \{\langle G = (V, E), s, t \rangle : G \text{ contiene un ciclo hamiltoniano} \wedge s, t \in V\}$$

$$S_{\Gamma_{\text{PNC}}}(G, s, t) = \{p : p \text{ è un percorso in } G\}$$

$$\Pi_{\Gamma_{\text{PNC}}}(G, s, t, S_{\Gamma_{\text{PNC}}}(G, s, t)) = \exists p \in S_{\Gamma_{\text{PNC}}}(G, s, t) \text{ che connette } s \rightarrow t.$$

Se sappiamo che G contiene un ciclo hamiltoniano allora $S_{\text{PAC}}(G, s, t)$ contiene anche la coppia di percorsi contenuti nel ciclo che congiungono s e t . Questo significa che ogni istanza del problema è una istanza si. Quindi indipendentemente dalla codifica utilizzata, decidere se una qualunque istanza soddisfa il predicato del problema richiede costo costante. Data una qualunque codifica ragionevole χ per Γ_{PAC} per decidere se una parola $x \in \{0,1\}^n$ è contenuta in $L_{\text{PAC}}(\chi)$, dobbiamo verificare sia se x è la codifica di un grafo che contiene un ciclo hamiltoniano e di una coppia di suoi nodi, sia se questo grafo contiene un percorso che connette i due nodi. La prima di queste verifiche è un noto problema NP-completo. Perciò Γ_{PAC} è un problema NP-completo.

E' presente un evidente assimmetria fra la partizione generata in Σ^* da un linguaggio e la partizione generata in Σ^* da un problema decisionale



Dato un problema decisionale $\Gamma = \langle I_r, S_r, \Pi_r \rangle$ il suo problema complemento è:

$$\Gamma^c = \langle I_r, S_r, \neg \Pi_r \rangle$$

Dunque, fissata una codifica ragionevole $\chi: I_r \rightarrow \Sigma^*$ per il problema decisionale Γ , il linguaggio associato al problema Γ^c è:

$$L_{\Gamma^c}(\chi) = \{x \in \Sigma^*: x \in \chi(I_r) \wedge \neg \Pi_r(\chi^{-1}(x), S_r(\chi^{-1}(x)))\}.$$

$L_{\Gamma^c}(\chi)$ non coincide con $L_r(\chi)$ di $L_r(\chi)$.

$$L_r(\chi) = L_r \cup \{x \in \Sigma^*: x \notin \chi(I_r)\}.$$

Riprendendo il problema Γ_{PAC} ogni istanza del problema è un istanza si. Equivalentemente, ogni istanza del problema è un'istanza no del problema Γ_{PAC}^c , quindi indipendentemente dalla

codifica, decidere se un'istanza soddisfa il predicato del problema T_{PAU}^C richiede costo costante. Data una qualunque codifica ragionevole χ per T_{PAU}^C per decidere se una parola $x \in \{0,1\}^n$ è contenuta in $L_{T_{\text{PAU}}}(\chi)$, dobbiamo verificare se x è la codifica di un grafo che contiene un ciclo hamiltoniano e di una coppia di suoi nodi. Dunque anche il linguaggio $L_{T_{\text{PAU}}}^c(\chi)$ è NP-compl. In virtù del Teo 6.21 $L_{T_{\text{PAU}}}^c$ è coNP-completo.

Teo 7.1: Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \pi_\Gamma \rangle$ un problema decisionale e sia $\chi: I_\Gamma \rightarrow \Sigma^*$ una codifica ragionevole. Se $\chi(I_\Gamma) \in P$ e $L_\Gamma(\chi) \in NP \Rightarrow L_{\Gamma^c}(\chi) \in \text{coNP}$

dim.

Poiché $\chi(I_\Gamma) \in P$, allora esistono una macchina di Turing deterministica T e un intero h tali che: $\forall x \in \Sigma^*, T$ decide se $x \in \chi(I_\Gamma)$ e $\text{ntime}(T, x) \in O(|x|^h)$.

Se $L_\Gamma(\chi) \in NP$, allora esistono una macchina di Turing non deterministica NT_Γ e un intero K t.c.: $\forall x \in L_\Gamma(\chi), NT_\Gamma$ accetta x e $\text{ntime}(NT_\Gamma, x) \in O(|x|^K)$. Combinando T e NT_Γ deriviamo una macchina NT' che con input x opera in due fasi:

1) simula $T(x)$: se $T(x)$ termina nello stato di rigetto allora NT' termina nello stato di accettazione, altrimenti va in fase 2.

2) simula $NT_\Gamma(x)$: se $NT_\Gamma(x)$ accetta allora NT' accetta.

Quindi NT' accetta sse $x \in \chi(I_\Gamma)$ oppure $x \in \chi(I_{\Gamma^c})$, ossia, sse $x \in (L_{\Gamma^c}(\chi))^c$. Inoltre $\text{ntime}(NT', x) \in O(|x|^{\max\{h, K\}})$. Quindi $(L_{\Gamma^c}(\chi))^c$ è in NP e $L_{\Gamma^c}(\chi) \in \text{coNP}$ □

Analogo EXPTIME, NEXPTIME. Poiché $\text{coNPSPACE} = \text{PSPACE} = \text{NPSPACE} = \text{coNPSPACE}$ $L_\Gamma(\chi) \in \text{PSPACE}$ sse $L_{\Gamma^c}(\chi) \in \text{PSPACE}$.

Let 18

Per dimostrare che un problema appartiene a P occorre trovare un algoritmo che lo risolve e dimostrare che richiede tempo polinomiale nella dimensione dell'input.

NP è la classe dei linguaggi accettati in tempo polinomiale da una macchina di Turing non deterministica NT e un intero $K \in \mathbb{N}$ tali che $\forall x \in L, NT(x)$ accetta in $|x|^K$ passi.

Se $x \in \Sigma^* - L, NT(x)$ non accetta. Poiché sappiamo che, su input $x \in \Sigma^*$, se NT accetta lo fa in $|x|^K$ passi, se al passo $|x|^K$ la computazione $NT(x)$ non ha ancora raggiunto lo stato di accettazione, possiamo concludere che non lo raggiungerà mai: $x \notin L$. Possiamo quindi derivare una macchina NT che decide L. Quindi NP è la classe dei linguaggi decisi in tempo polinomiale da una macchina di Turing non deterministica.

Ciascuna macchina NT ha la possibilità di valutare in parallelo, ad ogni passo, un numero costante p_{NT} di possibilità (p_{NT} = grado di non determinismo). In una computazione di K passi, vengono valutate p^K_{NT} possibilità.

La computazione di una macchina non deterministica non esamina in parallelo tutte le scelte possibili: consiste in una sequenze di scelte, in un percorso nell'albero della computazione che connette la radice ad una foglia. Lo scopo di una computazione non deterministica è verificare se esiste una scelta che permette di affermare che l'input appartiene al linguaggio, se è un istanza sì del problema.

La computazione non deterministica va considerata come l'assegnazione alla macchina non deterministica del compito di indovinare la scelta giusta da fare, quella che porterà ad uno stato globale di accettazione, se tale scelta non esiste, allora non ha alcuna importanza quale quintupla la macchina deciderà di eseguire.

Per questo la classe NP è definita come la classe dei linguaggi accettati (e non decisi) in tempo polinomiale da una macchina di Turing non deterministica: se l'input non appartiene al linguaggio la sequenza delle scelte della macchina di Turing non deterministica è priva di

significato.

Codice corrispondente ad una macchina di Turing non deterministica NT ad un nastro:

Input: $x = x_1 \dots x_n$ memorizzato in N ($N[i] = x_i$)

Costanti: l'insieme delle quintuple $P = \{ \langle q_{i1}, s_{i1}, s_{i2}, q_{i2}, m \rangle \dots \langle q_{k1}, s_{k1}, s_{k2}, q_{k2}, m \rangle \}$

$q \leftarrow q_0$

$t \leftarrow 1$

$pc \leftarrow t$ $uc \leftarrow n$

while ($q \neq q_A$ \wedge $q \neq q_R$) do begin

$\Psi \leftarrow \{ \langle q_{i1}, s_{i1}, s_{i2}, q_{i2}, m_i \rangle \in P : q_{i1} = q \wedge s_{i1} = N[t] \}$

if ($\Psi \neq \emptyset$) then begin

scegli $\langle q_{i1}, s_{i1}, s_{i2}, q_{i2}, m_i \rangle \in \Psi$

$N[t] \leftarrow s_{i2}$

$q \leftarrow q_{i2}$

$t = t + m$

end

if ($t < pc$) then begin

$pc \leftarrow t$

$N[t] \leftarrow \square$

end

if ($t > uc$) then begin

$uc \leftarrow t$

$N[t] \leftarrow \square$

end

end

Output q

Se termina in q_A esiste una computazione accettante di $NT(x)$.

Se termina in q_R esiste una computazione di $NT(x)$ che rigetta ma non dimostra che tutte le computazioni di $NT(x)$ rigettano.

La cardinalità di $\Psi(t, x)$ è l'insieme delle scelte possibili ad un passo t su input x ed è al più il grado di non determinismo di NT, quindi è costante.

Problema 3SAT

Data una funzione booleana f , decidere se esiste una assegnazione di verità a tutte le sue variabili che soddisfa f .

$$I_{3SAT} = \{ \langle X, f \rangle : f = c_1 \wedge \dots \wedge c_n, \forall i [c_i = l_i \vee \neg l_i \vee l_{ij}, l_i, l_{ij} \in X] \}$$

$$S_{3SAT}(X, f) = \{ a : X \rightarrow \{0, 1\} \}$$

$$\Pi_{3SAT}(X, f, S_{3SAT}(X, f)) = \exists a \in S_{3SAT}(X, f) : f(a(X)) = \text{vero}$$

Input: X, f

$i \leftarrow 1$

for ($i \leftarrow 1$; $i \leq |X|$; $i \leftarrow i+1$) do

$O(|X|)$

scegli $a(x_i) \in \{\text{vero, falso}\}$

for ($i \leftarrow 1$; $i \leq |X|$; $i \leftarrow i+1$) do

$O(|X|f)$

sostituisci $a(x_i)$ con x_i in f e $\neg a(x_i)$ con $\neg x_i$

if ($f(a(X)) = \text{vero}$) then $q \leftarrow q_a$

else $q \leftarrow q_b$

Output q

Problema CLIQUE

Dati un grafo non orientato $G = (V, E)$ ed un intero $K \in \mathbb{N}$, decidere se G contiene una clique di almeno K nodi.

$$I_{\text{clique}} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato e } K \in \mathbb{N} \}$$

$$S_{\text{clique}}(G, K) = \{ V' \subseteq V \}$$

$$\Pi_{\text{clique}}(G, K, S_{\text{clique}}(G, K)) = \exists V' \in S_{\text{clique}}(G, K) : \forall u, v \in V' [(u, v) \in E] \wedge |V'| \geq K$$

Input: $G = (V, E), K \in \mathbb{N}$

$V' \leftarrow \emptyset$

for ($v \in V$) do

$O(|V|)$

scegli se $v \in V'$ o $v \notin V'$

```

clique ← vero
if ( $|V'| < K$ ) then
    clique ← falso
for ( $v \in V'$ ) do
    for ( $u \in V'$ ) do
        if ( $u \neq v \wedge (u, v) \in E$ ) then clique ← falso
return clique

```

$$n = |V|$$

$$\mathcal{O}(|V|) \quad n$$

$$\mathcal{O}(|V|^2 |E|) \quad n^2$$

Problema Long Path

...

Quando si progetta un algoritmo non deterministico il numero delle opzioni fra le quali scegliere deve essere costante.

In tutti questi problemi π ha la forma seguente: esiste un elemento di S che soddisfa certe proprietà η

$$\pi(x, S(x)) = \exists y \in S(x) : \eta(x, y)$$

Inoltre tutti gli algoritmi seguono lo stesso schema: con input x ,
 fase 1 (non deterministica): sceglie una soluzione possibile $y \in S(x)$
 fase 2 (deterministica): verifica se y soddisfa il predicato $\eta(x, y)$

Entrambe le fasi richiedono tempo polinomiale in $|x|$.

Ogni problema con questa struttura è NP.

Lez 19

Tutti i problemi decisionali tali che

- 1) il predicato ha la forma $\pi(x, S(x)) = \exists y \in S(x) : M(x, y)$
 - 2) la scelta di un elemento y di $S(x)$ richiede tempo non deterministico polinomiale in $|x|$
 - 3) la verifica che y soddisfi il predicato M , richiede tempo polinomiale in $|x|$
- appartengono ad NP.

Teo 9.1: Un linguaggio $L \subseteq \Sigma^*$ è NP se esistono una macchina di Turing deterministica T e una costante $K \in \mathbb{N}$ tali che $\forall x \in \Sigma^* [x \in L \Leftrightarrow \exists y \in \{0,1\}^* : |y_x| \leq |x|^K \wedge T(x, y_x) \text{ accetta} \wedge \text{dtime}(T, x, y_x) \in O(|x|^K)]$.

dim.

\Rightarrow) Sia $L \subseteq \Sigma^*$ un linguaggio in NP allora esistono una macchina di Turing non deterministica NT e un intero $h \in \mathbb{N}$ t.c. NT accetta L e, $\forall x \in L$ $\text{dtime}(NT, x) \leq |x|^h$.
 $\forall x \in L$ esiste una computazione deterministica di NT(x) che termina nello stato di accettazione.

Poniamo, allora,

$$y(x) = q_1, s_1, s_2, q_2, m_1, \dots, q_{(h)}, s_{(h)}, s_{(h)}, q_{(h)}, m_{(h)}$$

ossia $y(x)$ è la parola ottenuta concatenando le parole che corrispondono alla sequenza accettante di quintuple.

Definiamo ora una macchina di Turing deterministica \bar{T} che corrisponde alla macchina NT: possiede codificata nelle sue quintuple, la descrizione dell'insieme P delle quintuple di NT. La computazione $\bar{T}(x, y)$ con input $x \in \Sigma^*$ scritto sul primo nastro e y scritto sul secondo nastro procede come segue:

- 1) \bar{T} verifica che y sia nella forma giusta altrimenti rigetta.
- 2) \bar{T} verifica che $\forall i: 1 \leq i \leq n^h, \langle q_i, s_i, s_{(i)}, q_{(i)}, m_i \rangle \in P$ altrimenti rigetta.
- 3) \bar{T} verifica che $q_1 = q_0$ e $q_{(n^h)} = q_h$ altrimenti rigetta.
- 4) \bar{T} verifica che $\forall i: 2 \leq i \leq n^h, q_i = q_{(i-1)}$ altrimenti rigetta.
- 5) \bar{T} simula la computazione di NT(x) descritta da y .
- 6) \bar{T} accetta.

Sia $x \in L$ allora $\exists y(x) : \bar{T}(x, y(x))$ accetta, se invece $x \notin L$ allora qualunque sia $y(x)$ $\bar{T}(x, y(x))$ non può accettare: o $y(x)$ non codifica una possibile computazione accettante

di NT. (rigetta al passo 1 o 2) o 3 o 4) o $y(x)$ codifica una computazione accettante di NT che però non può essere eseguita sull'input x e rigetta al passo 5).

Dunque $x \in L$ sse $\exists y(x) : \bar{T}(x, y(x))$ accetta. \bar{T} opera in tempo polinomiale in $|x|$ e $|y(x)|$ e $y(x)$ è costituita di al più $|x|^k$ passi, quindi se $x \in L$, $|y(x)| \in O(|x|^k)$ e \bar{T} opera in tempo polinomiale in $|x|$. Infine basta trasformare \bar{T} in una macchina il cui alfabeto è $\{0, 1\}$.

\Leftarrow) Sia $L \subseteq \Sigma^*$ un linguaggio per il quale esistono una macchina di Turing T che opera in tempo polinomiale e una costante $K \in \mathbb{N}$ tali che $\forall x \in \Sigma^* \quad x \in L \Leftrightarrow \exists y \in \{0, 1\}^* \text{ t.c. } |y| \leq |x|^K \wedge T(x, y) \text{ accetta.}$

Assumiamo T disponga di un solo nastro, inizialmente sono scritte x e y separate da " \square ".

Definiamo la macchina di Turing non deterministica NT che opera in due fasi:

- fase 1: genera una parola $y \in \{0, 1\}^*$ di lunghezza al più $|x|^K$ scrivendolo alla destra dell'input x e separato da " \square ".
- fase 2: simula $T(x, y)$.

La prima fase richiede al più $O(|x|^K)$ passi e la seconda richiede tempo proporzionale a ottime $|T(x, y)|$, polinomiale in $|x|$.

Se $x \in L$, per ipotesi, esiste una parola y_x tale che $T(x, y_x)$ accetta, allora durante la fase 1 esiste una sequenza di scritte che genera y_x che nella fase 2 porta NT ad accettare. Di contro, se NT accetta x allora esiste una parola y_x generata durante la prima fase che induce la seconda fase ad accettare.

$x \in L$ sse $NT(x)$ accetta e dato che NT opera in tempo polinomiale: $L \in NP$. \square

La parola y_x può essere considerata prova che x sia contenuto in L , una prova che deve essere successivamente verificata. Ad essa ci riferisce come ad un certificato in quanto certifica l'appartenenza di una parola ad un linguaggio. Ogni problema T in NP può essere formalizzato nel seguente modo:

- un insieme di istanze I_r .
- $\forall x \in I_r$, un insieme di soluzioni possibili $S_r(x)$ generabili non deterministicamente in tempo polinomiale.
- $\forall x \in I_r$, un predicato $\pi_r(x, S(x))$ nella forma $\pi_r(x, S(x)) = \exists y \in S_r(x) : \gamma_r(x, y)$ tale che

$M_n(x, y)$ è decidibile in tempo polinomiale di $|x|$.

Un problema è in NP se le parole che esso contiene ammettono certificati verificabili in tempo polinomiale nella loro dimensione.

Lez 20

La classe NP contiene la sottoclasse NPC dei problemi NP-completi. Un problema Γ è NP-completo se:

- 1) $\Gamma \in NP$ e
- 2) $\forall \Gamma' \in NP \quad \Gamma' \leq \Gamma$.

Quindi se un linguaggio Γ è NP-completo, la soluzione di ogni altro linguaggio in NP è riducibile alla soluzione di Γ . Se un problema NP-completo è contenuto in P allora $P = NP$. Dunque, i problemi NP-completi sono i candidati ad essere problemi separatori fra P e NP.

Teo di Cook-Levin: SAT è NP-completo

dim.

Bisogna dimostrare che è possibile ridurre a SAT ogni problema in NP. Se x è un'istanza sì di Γ allora l'istanza nella quale x viene trasformata è un'istanza sì di SAT. Se x è un'istanza no di Γ allora l'istanza nella quale x viene trasformata è un'istanza no di SAT.

Consideriamo un generico problema $\Gamma \in NP$ e sia $L_p = \{0, 1\}^*$ il linguaggio contenente la codifica ragionevole delle istanze sì di Γ , scriviamo sottoforma di espressione booleana il predicato " $x \in L_p$ ".

Sia NT_p una macchina di Turing non deterministica ad un nastro che decide L_p in tempo polinomiale: ossia, esiste un polinomio p t.c. $\forall x \in \{0, 1\}^* [\text{time}(NT_p, x) \leq p(|x|)]$,

$$NT_p(x) = q_a \text{ se } x \in L_p \quad NT_p(x) \neq q_a \text{ se } x \notin L_p.$$

L'affermazione " $x \in L_p$ " è equivalente all'affermazione $y(x) = "x \text{ è scritto sul nastro di } NT_p, \text{ e la testina di } NT_p \text{ è posizionata sul primo carattere di } x, \text{ e } NT_p \text{ è nel suo stato iniziale, e esiste una sequenza di al più } p(|x|) \text{ quintuple di } NT_p \text{ che possono essere eseguite una di seguito all'altra e portano la macchina nello stato } q_a"$.

$$x \in L_p \text{ sse } y(x) \text{ è vera.}$$

Bisogna descrivere $y(x)$ sotto forma di espressione booleana $E(x)$ che sia soddisfacibile sse $y(x)$ è vera. $E(x)$ deve descrivere una computazione di NT_p che ha inizio con x scritto sul nastro, ogni computazione è una sequenza di stati globali.

Per costruire $E(x)$, bisogna introdurre le variabili booleane che descrivono, per ogni passo t della computazione ($0 \leq t \leq p(|x|)$) lo stato globale in cui si troverebbe NT_p al passo t della computazione $NT_p(x)$:

- un insieme N di variabili booleane che permettono di rappresentare il contenuto in ciascuna cella del nastro di lavoro di NT_p ad ogni passo della computazione $NT_p(x)$;
- un insieme M di variabili booleane che permettono di rappresentare lo stato interno di NT_p ad ogni passo della computazione $NT_p(x)$;
- un insieme R di variabili booleane che permettono di rappresentare la cella del nastro di lavoro sulla quale è posizionata la testina di NT_p ad ogni passo della computazione $NT_p(x)$;

Variabili per lo stato interno M

Sia $Q = \{q_0, \dots, q_k\}$ l'insieme degli stati di NT_p , q_0 è lo stato iniziale, $q_1 = q_a$ e $q_2 = q_r$. L'insieme M di variabili, insieme ad una porzione E_m dell'espressione $E(x)$ che stiamo costruendo, servono a descrivere in quale interno si trova NT_p ad ogni passo della computazione $NT_p(x)$: vogliamo che

Ogni volta che i valori assegnati alle variabili in M fanno assumere E_m il valore vero, osservando i valori assegnati alle variabili contenute in M dobbiamo essere in grado di rispondere alla domanda "è q_i lo stato interno di NT_p al passo x della computazione $NT_p(x)$?"

\forall passo t ($0 \leq t \leq p(|x|)$) $\forall i \in \{0, \dots, k\}$, M contiene una variabile booleana M_i^t :

$$M = \{M_i^t : 0 \leq t \leq p(|x|) \wedge i \in \{0, \dots, k\}\}$$

assegnando a M_i^t il valore vero rappresentiamo il fatto che, al passo t della computazione $NT_p(x)$, la macchina NT_p si trova nello stato q_i .

Introduciamo $p(|x|)+1$ espressioni: $E_m^0, \dots, E_m^{p(|x|)}$ dove E_m^t è l'espressione che corrisponde all'affermazione: al passo t di $NT_p(x)$ la macchina NT_p si trova in uno solo dei suoi stati interni.

$$E_m^t = [M_0^t \wedge M_1^t \wedge \dots \wedge M_k^t] \vee [M_1^t \wedge M_0^t \wedge \dots \wedge M_k^t] \vee \dots$$

$[M_k^t \wedge M_0^t \wedge \dots \wedge M_{k-1}^t]$: NT_p al passo t di $NT_p(x)$ si trova nello stato $q_0 \circ q_1 \circ \dots \circ q_k$. Il valore è vero se una delle variabili M_i^t è assegnato il valore vero.

Variabili per la posizione della testina R

Poiché N_{Tr} ha a disposizione $p(|x|)$ passi per eseguire x , allora non utilizza al più di $p(|x|)$ celle. Assumiamo quindi che $N_{Tr}(x)$ utilizzi le celle $1 \dots p(|x|)$.

L'insieme R di variabili, insieme ad una porzione E_R dell'espressione $E(x)$ che stiamo costruendo, servono a descrivere su quale cella è posizionata la testina di N_{Tr} ad ogni passo della computazione $N_{Tr}(x)$: vogliamo che

ogni volta che i valori assegnati alle variabili in R fanno assumere E_R il valore vero, osservando i valori assegnati alle variabili contenute in R dobbiamo essere in grado di rispondere alla domanda "la testina di N_{Tr} è posizionata sulla cella i al passo x della computazione $N_{Tr}(x)$?"

Al passo t ($0 \leq t \leq p(|x|)$) $\forall i \in \{1, \dots, p(|x|)\}$, R contiene una variabile booleana R_i^+ :

$$R = \{R_i^+ : 0 \leq t \leq p(|x|) \wedge i \in \{1, \dots, p(|x|)\}\}^{+ \text{ celle}}$$

assegnando a R_i^+ il valore vero rappresentiamo il fatto che, al passo t della computazione $N_{Tr}(x)$, la testina di N_{Tr} si trova sulla cella i.

Introduciamo $p(|x|)+1$ espressioni: $E_R^0, \dots, E_R^{p(|x|)}$ dove E_R^t è l'espressione che corrisponde all'affermazione: al passo t di $N_{Tr}(x)$ la testina di N_{Tr} è posizionata su una sola delle sue celle

$$E_R^t = [R_0^+ \wedge R_1^+ \wedge \dots \wedge R_{p(|x|)}^+] \vee [R_1^+ \wedge R_0^+ \wedge \dots \wedge R_{p(|x|)}^+] \vee \dots$$

$[R_{p(|x|)}^+ \wedge R_0^+ \wedge \dots \wedge R_{p(|x|)-1}^+]$: N_{Tr} al passo t di $N_{Tr}(x)$ la testina si trova sulla cella $i_0 i_1 i_2 \dots p(|x|)$. Il valore è vero se e solo se una delle variabili R_i^+ è assegnato il valore vero.

Variabili per il contenuto delle celle N

N_{Tr} utilizza al più $p(|x|)$ celle e $L_r \subseteq \{0, 1\}^*$, ogni cella può contenere 0 o 1.

L'insieme N di variabili, insieme ad una porzione E_N dell'espressione $E(x)$ che stiamo costruendo, servono a descrivere quale simbolo è contenuto su ogni cella del nastro di N_{Tr} ad ogni passo della computazione $N_{Tr}(x)$: vogliamo che

ogni volta che i valori assegnati alle variabili in N fanno assumere E_N il valore vero, osservando i valori assegnati alle variabili contenute in N dobbiamo essere in grado di rispondere alla domanda "è i il simbolo contenuto nella cella i al passo x della

computazione $NT_p(x)$?

\forall passo t ($0 \leq t \leq p(|x|)$), $i \in \{0, \dots, p(|x|)\}$, $j \in \{0, 1, \square\}$, N contiene una variabile booleana N_{ij}^+ :

$$N = \{N_{ij}^+ : 0 \leq i \leq p(|x|) \wedge i \in \{0, \dots, p(|x|)\} \wedge j \in \{0, 1, \square\}\}$$

assegnando a N_{ij}^+ il valore vero rappresentiamo il fatto che, al passo t della computazione $NT_p(x)$, la cella i contiene il simbolo j .

Introduciamo $p(|x|)+1$ espressioni: $E_N^0, \dots, E_N^{p(|x|)}$ dove E_N^t è l'espressione che corrisponde all'affermazione: al passo t di $NT_p(x)$ la cella i di NT_p contiene un solo elemento dell'insieme $\{0, 1, \square\}$

$$E_N^+ = [N_{i0}^+ \wedge N_{i1}^+ \wedge \neg N_{i0}^+] \vee [N_{i1}^+ \wedge \neg N_{i0}^+ \wedge \neg N_{i1}^+] \vee [N_{i0}^+ \wedge \neg N_{i0}^+ \wedge \neg N_{i1}^+]:$$

Nella cella i di NT_p , al passo t di $NT_p(x)$ si trova o il simbolo 0 o 1 o \square . Il valore è vero se una delle variabili N_{ij}^+ è assegnato il valore vero.

E_N^{ti} è l'espressione nelle variabili in R che corrisponde all'affermazione al passo t di $NT_p(x)$ la cella i contiene un elemento dell'insieme $\{0, 1, \square\}$

$$\forall t: 0 \leq t \leq p(|x|) \quad E_N^t = E_N^{t1} \wedge E_N^{t2} \wedge \dots \wedge E_N^{tp(|x|)}$$

Per descrivere un generico **stato globale** $S^t = E_M^t \wedge E_R^t \wedge E_N^t$. Un assegnazione di verità che soddisfa S^t rappresenta l'unico stato interno in cui NT si trova al tempo t , l'unica cella su cui è posizionata la testina al tempo t e l'unico simbolo contenuto in ciascuna cella del nastro al tempo t .

Allo stesso modo dobbiamo rappresentare le computazioni della macchina NT_p che accettano in al più $p(|x|)$ passi, ossia:

Esiste una sequenza di al più $p(|x|)$ quintuple di NT_p che possono essere eseguite una di seguito all'altra e portano la macchina nello stato q_f .

Quindi al passo 0, NT_p esegue una quintupla e

al passo t , NT_p è nello stato q_t o esegue una quintupla e

L'affermazione "la quintupla $\langle q_{i1}, s_1, s_2, q_{i2}, m \rangle$ è eseguita al passo t mentre la testina è posizionata nella cella i " equivale a:

$$G_i^+ \left(u, \langle q_{i1}, s_1, s_2, q_{i2}, m \rangle \right) = M_{iu}^+ \wedge R_u^+ \wedge N_{us1}^+ \wedge N_{us2}^{t_{ui}} \wedge M_{iz}^+ \wedge R_{uzm}^{t_{zi}}$$

Ma al passo t la testina potrebbe essere posizionata su qualunque cella, allora per esprimere che "su qualunque cella sia posizionata la testina, la quintupla è eseguita al passo t " equivale a:

$$G_i^+ \left(\langle q_{i1}, s_1, s_2, q_{i2}, m \rangle \right) = G_i^+ \left(1, \langle q_{i1}, s_1, s_2, q_{i2}, m \rangle \right) \vee G_i^+ \left(2, \langle q_{i1}, s_1, s_2, q_{i2}, m \rangle \right) \vee \dots \\ G_i^+ \left(p(|x|), \langle q_{i1}, s_1, s_2, q_{i2}, m \rangle \right)$$

Cioè per una singola quintupla, ma NT_p ha h quintupla perciò per esprimere l'affermazione "al passo t viene eseguita una quintupla di NT_p " equivale a:

$$G_i^+ = G_i^+ \left(\langle q_{i1}, s_{i1}, s_{i2}, q_{i2}, m_i \rangle \right) \vee \dots \vee G_i^+ \left(\langle q_{ih}, s_{h1}, s_{h2}, q_{h2}, m_h \rangle \right)$$

Per esprimere quindi l'affermazione "al passo t , NT_p è nello stato q_n oppure esegue una quintupla" equivale a: (ricordiamo che $q_n = q_1$)

$$M_i^+ \vee G_i^+$$

Descriviamo ora la prima parte di $E(x)$: " NT_p è nel suo stato iniziale" la testina di NT_p è posizionata sul primo carattere " x è scritto sul nastro di NT_p " equivale a:

$$H = M_0^0 \wedge R_1^0 \wedge N_{1x_1}^0 \wedge N_{2x_2}^0 \wedge \dots \wedge N_{nx_n}^0 \wedge N_{n+1}^0 \wedge N_{n+2}^0 \wedge \dots \wedge N_{p(|x|)}^0$$

Se $x = 1001$

$$H = M_0^0 \wedge R_1^0 \wedge N_{11}^0 \wedge N_{20}^0 \wedge N_{30}^0 \wedge N_{41}^0 \wedge N_{50}^0 \wedge \dots \wedge N_{p(4)}^0$$

L'espressione $E(x)$ equivale quindi a:

$$E(x) = H \wedge S^0 \wedge (M_i^+ \vee G_i^+) \wedge S^1 \wedge (M_i^+ \vee G_i^+) \wedge \dots \wedge S^{p(|x|-1)} \wedge (M_i^+ \vee G_i^+) \wedge S^{p(|x|)} \wedge M_i^{p(|x|)}$$

Stiamo mostrando che L_p è riducibile polinomialmente a SAT e che quindi abbiamo mostrato come trasformare x in $E(x)$. La macchina NT_p gioca il ruolo di costante: non c'è l'istanza. Dobbiamo a questo punto mostrare che $x \in L_p$ sse \exists un'assegnazione di verità in M, R, N , che soddisfa $E(x)$.

Se $x \in L_p$, allora esiste una computazione di $NT_p(x)$ che termina in q_p in al più $p(|x|)$ passi. bisogna quindi costruire un'assegnazione di verità che soddisfa $E(x)$.

1: usiamo SC_0 .

Poniamo $a(M^0) = a(R^0) = \text{vero}$, per $j=1..|x|$ se bit j di x è 0 poniamo $a(N_{j0}^0) = \text{vero}$ altrimenti $a(N_{j1}^0) = \text{vero}$, per $j=|x|..p(|x|)$, poniamo $a(N_{j0}^0) = \text{vero}$, a assegna falso in tutte le altre variabili in M^0, R^0, N^0 , pertanto a soddisfa $H^0 S^0$.

2: usiamo SC_1, \dots, SC_u .

Definiamo $a(M_i^+), a(R_i^+), a(N_{ji}^+)$ come fatto per il punto 1. $\forall t=1, \dots, u$ a soddisfa S^t .

3: usiamo le quintuple

$\forall t=0, \dots, u-1$ può essere eseguita la quintupla t della sequenza quindi $\forall t$ a soddisfa C_t^+ .

4: lo stato interno di SC_u è q_p

allora $a(M_u^u) = \text{vero}$, a soddisfa $(M^u, \vee C^u)$

5: $t > u$

$\forall i=0, \dots, K$ poniamo $a(M_i^+) = a(M_i^u)$, $\forall j=1, \dots, h$ poniamo $a(R_j^+) = a(R_{j0}^u), a(N_{ji}^+) = a(N_{ji0}^u), a(N_{ji1}^+) = a(N_{ji10}^u)$, a soddisfa S^t e M^t .

Questo dimostra che a soddisfa $E(x)$.

Supponiamo ora che esista un'assegnazione di verità a alle variabili M, N, R che soddisfa $E(x)$, ossia a soddisfa $H, S^{p(|x|)}$ e $M^{p(|x|)}$ e $\forall t=0, \dots, p(|x|)-1$ a soddisfa S^t e $(M^t, \vee C^t)$.

Poiché ogni assegnazione di verità che soddisfa S^t descrive uno stato globale di NT_p allora a descrive una sequenza $SC_1, \dots, SC^{p(|x|)-1}$ di stati globali di NT_p .

Poiché, a soddisfa $H, a(S^0)$ descrive SC_0 , inoltre $\forall t=0, \dots, p(|x|)-1$, a soddisfa $(M^t, \vee C^t)$

quindi o viene eseguita una quintupla ($a(C^t) = \text{vero}$) che fa passare da SC^t a SC^{t+1} oppure

NT_p è in q_p ($a(M^t) = \text{vero}$). Entrambe non possono essere vere. a corrisponde ad una sequenza di stati globali $SC_1, \dots, SC^{p(|x|)-1}$ di NT_p dove $\forall t=0, \dots, p(|x|)-1$ $a(S^t)$ corrisponde allo stato

globale SC^t . Poiché a soddisfa $E(x)$ allora $a(M^{p(|x|)-1}) = \text{vero}$, quindi esiste un indice h

tale che $a(M_h^h) = \text{vero}$ e $\forall t=h$ $a(M_t^h) = \text{vero}$. Sia $v \in \{0, \dots, p(|x|)\}$ il primo intero per cui $a(M_v^v) = \text{vero}$ allora $\langle SC_0, \dots, SC^v \rangle$ è una computazione accettante di $NT_p(x)$ e quindi $x \in L_p$.

Calcolare S^t e C^t richiede $O(p(|x|))$ passi, calcolarli tutti richiede $O(p(|x|)^2)$ passi,

H richiede $O(p(|x|))$ passi. $E(x)$ è quindi calcolabile in $O(p(|x|)^2)$ passi, $x \in L_p$ sse $E(x)$ è soddisfacibile, trasformare $E(x)$ in CNF richiede tempo $O(p(|x|))$ passi e poiché 3SAT ∈ NP e SAT ∈ NP allora SAT ∈ NP-completo.

Lez 21

Abbiamo dimostrato che SAT è un problema NP-completo.

Se $P = NP$ e si trovasse un algoritmo (deterministico) polinomiale che decide SAT, il teorema di Cook-Levin ci permette di costruire un algoritmo deterministico polinomiale per decidere qualunque problema in NP.

Se $P \neq NP$ allora SAT $\in P$ e ogni volta che si dimostra che un problema è NP-completo supponiamo che quel problema non è in P, ossia i problemi NPC sono i problemi separatori tra P e NP .

Dati due problemi decisionali Γ_1 e Γ_2 allora $\Gamma_1 \leq \Gamma_2$ quando $L_{\Gamma_1} \leq L_{\Gamma_2}$ dove L sono i linguaggi associati alle codifiche ragionevoli delle istanze dei problemi.

$\Gamma_1 \leq \Gamma_2$ se $\exists f: I_{\Gamma_1} \rightarrow I_{\Gamma_2}$ t.c.:

- $f \in FP$.

- x è una istanza sì di Γ_1 sse $f(x)$ è un'istanza sì di Γ_2 .

non bisogna quindi utilizzare ogni volta Cook-Levin

Tes 9.3: Sia Γ un problema in NP, se esiste un problema NP-completo riducibile a Γ allora Γ è NP-completo.

dim.

Sia Γ_0 un problema NP-completo tale che $\Gamma_1 \leq \Gamma_0 \Rightarrow \exists f_{10}: I_{\Gamma_1} \rightarrow I_{\Gamma_0}$ t.c. $f_{10} \in FP$ e

$\forall x \in I_{\Gamma_1} [x \in \Gamma_1 \Leftrightarrow f_{10}(x) \in \Gamma_0]$. Poiché Γ_1 è NPC \forall problema $\Gamma_2 \in NP$ si ha che $\Gamma_2 \leq \Gamma_1$ e dunque $\exists f_{21}: I_{\Gamma_2} \rightarrow I_{\Gamma_1}$ t.c. $f_{21} \in FP$ e $\forall x \in I_{\Gamma_2} [x \in \Gamma_2 \Leftrightarrow f_{21}(x) \in \Gamma_1]$.

Poiché $f_{21} \in FP \Rightarrow \exists T_{21}, K \in \mathbb{N}: \forall x \in I_{\Gamma_2} [\text{dtime}(T_{21}, x) \leq |x|^k]$, analogamente

Poiché $f_{10} \in FP \Rightarrow \exists T_{10}, h \in \mathbb{N}: \forall x \in I_{\Gamma_1} [\text{dtime}(T_{10}, x) \leq |x|^h]$

$x \in \Gamma_2 \Leftrightarrow f_{21}(x) \in \Gamma_1 \Leftrightarrow f_{10}(f_{21}(x)) \in \Gamma_0$. Indichiamo con f_{20} la composizione delle funzioni.

Possiamo quindi definire T_{20} che calcola f_{20} , la computazione ha inizio con l'input $x \in \Gamma_2$ in seguito:

1) T_{20} simula $T_{21}(x)$ scrivendo $f_{21}(x)$ sul nastro di lavoro.

2) T_{20} simula $T_{10}(f_{21}(x))$ scrivendo $f_{10}(f_{21}(x))$ sul nastro output.

$\forall x \in \Gamma_2 \text{ dtime}(T_{20}, x) \leq |x|^k + |f_{21}(x)|^h \leq |x|^k + |x|^{kh} \leq 2|x|^{hk+1}$, essendo h, k costanti

$f_{21}(x) \leq |x|^h$ perché fa parte dell'esecuzione di T_{21}

$f_{20} \in FP$. Perciò $\Gamma_2 \leq \Gamma_0$ e ogni problema in NP è $\leq \Gamma_0$. Dato che $\Gamma_0 \in NP$ segue che $\Gamma_0 \in NPC$ \square

Per dimostrare che un problema Γ è NPC è sufficiente:

- mostrare che $\Gamma \in NP$
- scegliere un problema NPC noto Γ_2 e dimostrare che $\Gamma_2 \leq \Gamma$

3SAT

$$I_{\text{SAT}} = \{ \langle f, X \rangle : f \text{ è in CNF} \}$$

$$S_{\text{SAT}}(f, X) = \{ a : X \rightarrow \{0,1\} \}$$

$$\Pi_{\text{SAT}}(f, S_{\text{SAT}}(f, X)) = \exists a \in S_{\text{SAT}}(f, X) : f(a(x_1), \dots, a(x_n)) = \text{Vero}$$

$$I_{\text{3SAT}} = \{ \langle f, X \rangle : f \text{ è in 3CNF} \}$$

$$S_{\text{3SAT}}(f, X) = \{ a : X \rightarrow \{0,1\} \}$$

$$\Pi_{\text{3SAT}}(f, S_{\text{3SAT}}(f, X)) = \exists a \in S_{\text{3SAT}}(f, X) : f(a(x_1), \dots, a(x_n)) = \text{Vero}$$

$$I_{\text{3SAT}} \subseteq I_{\text{SAT}}$$

Sappiamo già che 3SAT è NP dimostriamo che sia NPC mediante una riduzione da SAT

$$\text{SAT} \leq \text{3SAT}$$

dobbiamo trasformare $\langle X, f \rangle$ in un'istanza di 3SAT $\langle X', f' \rangle$ t.c. f è soddisfacibile sse f' lo è.

caso 1: c_j contiene un letterale, $c_j = l$

$$d_j = (l \vee z_{j1} \vee z_{j2}) \wedge (\bar{l} \vee \bar{z}_{j1} \vee z_{j2}) \wedge (\bar{l} \vee z_{j1} \vee \bar{z}_{j2}) \wedge (\bar{l} \vee \bar{z}_{j1} \vee \bar{z}_{j2})$$

qualsiasi assegnazione
di z rende falsa una
clausola

caso 2: $c_j = l_1 \vee l_2$

$$d_j = (l_1 \vee l_2 \vee z_j) \wedge (\bar{l}_1 \vee \bar{l}_2 \vee z_j)$$

caso 3: $c_j = l_1 \vee l_2 \vee l_3$

rimane com'è

caso 4: $c_j = l_1 \vee l_2 \vee l_3 \vee l_4$

$$d_j = (l_1 \vee l_2 \vee z_j) \wedge (\bar{l}_1 \vee \bar{l}_2 \vee z_j) \wedge (\bar{l}_1 \vee l_3 \vee z_j) \wedge (\bar{l}_1 \vee l_3 \vee \bar{z}_j)$$

se tutti sono falsi
allora d_j è falso

caso 5: $c_j = l_1 \vee \dots \vee l_s$

$$d_j = (l_1 \vee l_2 \vee z_{j1}) \wedge (\bar{l}_1 \vee \bar{l}_2 \vee z_{j1}) \wedge (\bar{l}_1 \vee l_3 \vee z_{j2}) \wedge (\bar{l}_1 \vee l_3 \vee \bar{z}_{j2}) \wedge \dots \wedge (\bar{l}_1 \vee l_s \vee z_{js}) \wedge (\bar{l}_1 \vee l_s \vee \bar{z}_{js})$$

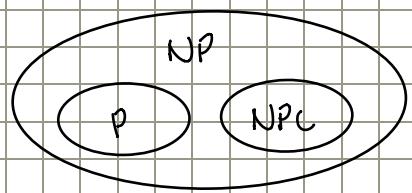
caso 6: $c_j = l_1 \vee \dots \vee l_h$

$$d_j = (l_1 \vee l_2 \vee z_{j1}) \wedge (\bar{l}_1 \vee \bar{l}_2 \vee z_{j1}) \wedge (\bar{l}_1 \vee l_3 \vee z_{j2}) \wedge (\bar{l}_1 \vee l_3 \vee \bar{z}_{j2}) \wedge \dots \wedge (\bar{l}_1 \vee l_h \vee z_{jh}) \wedge (\bar{l}_1 \vee l_h \vee \bar{z}_{jh})$$

Per costruire ciascuna d_j occorrono $O(|X|)$ passi e devono essere fatte $O(|f|)$ congiunzioni

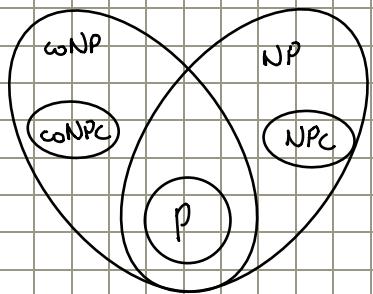
$\exists SAT \in NPC$.

Teo di Ladner: Se $P \neq NP$ allora esiste un problema Γ^* in $NP-P$ che non è NPC



Questi problemi si dicono NP -intermedi, non si può dimostrare che un problema è NP -int.

Per adesso possiamo ridurre la struttura delle classi come:



Lec 22

VERTEX COVER VC

Un vertex cover è un insieme di nodi che copre tutti gli archi del grafo.

$$I_{vc} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{vc}(G, K) = \{ V' \subseteq V \mid$$

$$\Pi_{vc}(G, K, S_{vc}(G, K)) = \exists V' \in S_{vc}(G, K) : |V'| \leq K \wedge \forall (u, v) \in E [u \in V' \vee v \in V']$$

1: dimostriamo che vc è NP mostrando un certificato che sia verificabile in tempo polinomiale

Un certificato è un sottoinsieme V' di V , verificare che V' soddisfi Π_{vc} , dobbiamo quindi esaminare ciascun arco (u, v) di G e verificare che $u \in V'$ o $v \in V'$. Si verifica in tempo $O(|E||V|)$

2: dimostriamo che vc è NPC $3SAT \leq vc$

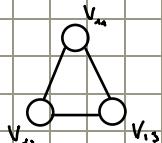
Le istanze dei due problemi sono diverse, dobbiamo far corrispondere ad ogni struttura dell'istanza di A una struttura dell'istanza di B. Tali strutture si chiamano **gadget**.

In questo caso introduciamo il gadget-variabile che trasformi ciascuna variabile in X in un arco.

$$X = \{x_1, x_2, \dots, x_n\} \quad \overset{v_i}{\bullet} \quad \overset{\neg v_i}{\bullet}$$

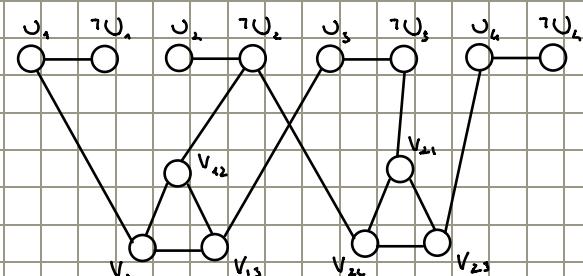
Scegliendo un nodo in ogni gadget-variabile otteniamo un vc con cardinalità almeno $|X|$

Usiamo anche il gadget-clausola che trasforma ogni clausola in un ciclo di 3 nodi.



Scegliendo due nodi in ogni gadget-clausola otteniamo un vc con cardinalità almeno $2|f|$

Ora bisogna collegare i gadget, per farlo utilizziamo degli archi obliqui per collegare nodo e variabile



$$f = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$

Costruire questo grafo corrispondente a $\langle X, f \rangle$ richiede tempo polinomiale e

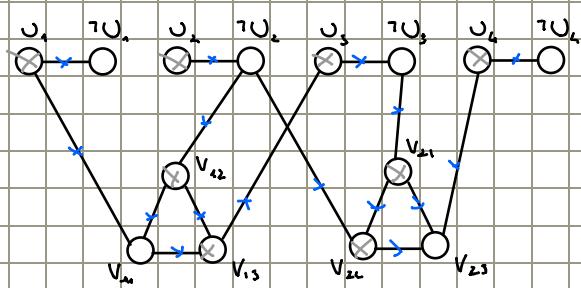
$K = |X| + 2|f| = n + 2m$. Bisogna dimostrare che f è soddisfacibile sse G ha un vc di al-

più $K = n+2m$ nodi.

Se g è soddisfacibile, costruiamo l'insieme V' nel seguente modo:

1) inseriamo in V' n nodi dei gadget-variabile, se $a(x_i) = \text{vero} \Rightarrow v_i \in V'$, se $a(x_i) = \text{falso} \Rightarrow \neg v_i \in V'$

2) $\forall m=1, \dots, n$ scegliamo un letterale l_{gh} nella clausola c_g al quale è stato assegnato il valore vero da a e inseriamo in V' i due nodi del gadget-clausola associati a c_g che non sono v_{gh}



$x_i = \text{vero}$

$$f = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$

$$|V'| = n+2m$$

Se f è soddisfacibile, allora G contiene un vc di $K(n+2m)$ nodi. Se G contiene un vc di K nodi allora V' contiene esattamente un nodo per ogni gadget-variabile e due per ogni gadget-clausola, $a(x_i) = \text{vero}$ se $v_i \in V'$, $a(x_i) = \text{falso}$ se $\neg v_i \in V'$, poiché V' contiene due nodi per ogni clausola un arco obliquo è sicuramente non coperto, perciò un arco obliquo deve essere coperto da un nodo di una variabile, questo significa che ogni clausola contiene un letterale con valore vero e quindi f è soddisfacibile.

INDEPENDENT SET IS

Un independent set è un insieme di nodi tali che nessuna coppia di nodi condivide un arco

$$I_{IS} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato } \wedge K \in \mathbb{N} \}$$

$$S_{IS}(G, K) = \{ I \subseteq V \}$$

$$\Pi_{IS}(G, K, S_{IS}(G, K)) = \exists I \in S_{IS}(G, K). |I| \geq K \wedge \forall u, v \in I [(u, v) \notin E]$$

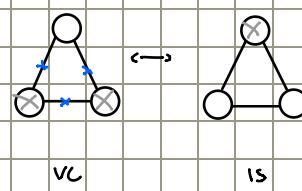
∴ dimostriamo che $IS \in NP$ mostrando un certificato che sia verificabile in tempo polinomiale

Un certificato è un sottoinsieme I di V , verificare che I soddisfi Π_{IS} , dobbiamo quindi esaminare ciascuna coppia di nodi u, v in I e verificare che $(u, v) \notin E$. Si verifica in tempo $O(|E||V|^2)$

2: dimostriamo che $IS \in NPC \quad VC \leq IS$

Basta osservare che: $I \subseteq V$ è un is per $G \Leftrightarrow V = V - I$ è un vc per G

$$\langle G = (V, E), K \rangle \in VC \rightarrow \langle G \cdot (V, E), |V| - K \rangle \in IS$$



CLIQUE CL

Una clique è un insieme di nodi tali che ogni coppia di nodi condivide un arco

$$I_{cl} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{cl}(G, K) = \{ C \subseteq V \mid$$

$$\pi_{cl}(G, K, S_{cl}(G, K)) = \exists C \in S_{cl}(G, K) : |C| \geq K \wedge \forall u, v \in C [(u, v) \in E]$$

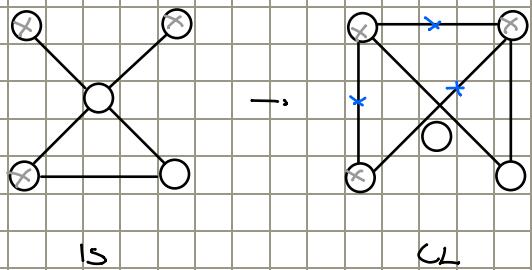
1: dimostriamo che $CL \in NP$ mostrando un certificato che sia verificabile in tempo polinomiale

Un certificato è un sottoinsieme C di V , verificare che C soddisfi π_{cl} , dobbiamo quindi esaminare ciascuna coppia di nodi u, v in C e verificare che $(u, v) \in E$. Si verifica in tempo $O(|E||V|^2)$

2: dimostriamo che $CL \in NPC \quad IS \leq CL$

Basta osservare che: G^c è il grafo complementare di G : (u, v) è un arco di G^c sse (u, v) non è un arco di G . $I \subseteq V$ è un IS per $G \Leftrightarrow I$ è una clique per G^c .

$$\langle G = (V, E), K \rangle \in IS \rightarrow \langle G^c = (V, E^c), K \rangle \in CL$$



DOMINATING SET DS

Una dominating set è un insieme D di nodi tali che ogni nodo che non è in D ha almeno un vicino in D .

$$I_{ds} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{ds}(G, K) = \{ D \subseteq V \mid$$

$$\pi_{ds}(G, K, S_{ds}(G, K)) = \exists D \in S_{ds}(G, K) : |D| \leq K \wedge \forall v \in V - D [\exists v \in D : (v, v) \in E]$$

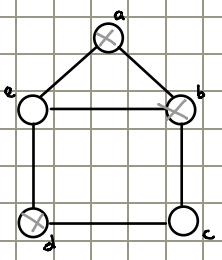
1: dimostriamo che $DS \in NP$ mostrando un certificato che sia verificabile in tempo polinomiale

Un certificato è un sottoinsieme D di V , verificare che D soddisfi π_{ds} , dobbiamo quindi esaminare ciascun nodo v in $V - D$ e verificare che $\exists v \in D : (v, v) \in E$. Si verifica in tempo $O(|E||V|^2)$

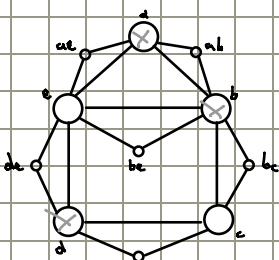
2: dimostriamo che $DS \in NPC \quad VC \leq DS$

Trasformiamo un istanza di VC nell'istanza di DS dove $V_{ds} = V \cup W$ con $W = \{uv : (u, v) \in E\}$

$$E_{ds} = E \cup F \text{ con } F = \{(u, uv), (v, uv) : (u, v) \in E\}$$



VC



DS

$$\langle G = (V, E), K \rangle \in VC \rightarrow \langle G_{\text{os}} = (V_{\text{os}}, E_{\text{os}}), K \rangle \in DS$$

Se G ha un VC V' , allora V' è un DS. $V' \subseteq V \subseteq V_{\text{os}}$, comunque scegliamo un nodo u in V_{os} se $u \notin V - V'$: poiché G è connesso esiste $(u, v) \in E$ e poiché V' è un VC per G $v \in V'$

se $u = xy \in W$ poiché V' è un VC per G allora $x \in V'$ o $y \in V'$

Se G_{os} ha un DS D allora trasformiamo D in un D' f.c. $D' \subseteq V$ e $|D'| = |D|$, se D contiene $uv \in W$ sostituiamolo con u o v .

D' è un VC per G : $\forall (u, v) \in E$ poiché D' è un DS per G_{os} allora $u \in D'$ o $v \in D'$

Lec 23

HAMILTONIAN CYCLE HC

Un ciclo hamiltoniano è un ciclo in G che passa una ed una sola volta attraverso ogni nodo di G

$$I_{HC} = \{ \langle G = (V, E) \rangle : G \text{ è un grafo non orientato} \}$$

$$S_{HC}(G) = \{ \langle v_1, \dots, v_n \rangle : \text{per } i=1, \dots, n \quad v_i \in V \wedge n = |V| \}$$

$$\Pi_{HC}(G, S_{HC}(G)) = \{ \langle v_1, \dots, v_n \rangle \in S_{HC}(G) : (v_n, v_1) \in E \wedge \forall i=1, \dots, n-1 [(v_i, v_{i+1}) \in E] \wedge \forall i, j=1, \dots, n \text{ con } i \neq j [v_i \neq v_j] \}$$

HC ∈ NPC

HAMILTONIAN PATH HP

Un percorso hamiltoniano da s a t è un percorso che passa una e una sola volta attraverso ciascun nodo di G

$$I_{HP} = \{ \langle G = (V, E), s, t \rangle : G \text{ è un grafo non orientato} \wedge s, t \in V \}$$

$$S_{HP}(G, s, t) = \{ \langle v_1, \dots, v_n \rangle : \text{per } i=1, \dots, n \quad v_i \in V \wedge n = |V| \}$$

$$\Pi_{HP}(G, s, t, S_{HP}(G, s, t)) = \{ \langle v_1, \dots, v_n \rangle \in S_{HP}(G, s, t) : s = v_1 \wedge t = v_n \wedge \forall i=1, \dots, n-1 [(v_i, v_{i+1}) \in E] \wedge \forall i, j=1, \dots, n \text{ con } i \neq j [v_i \neq v_j] \}$$

1: dimostriamo che $HP \in NP$ mostrando un certificato che sia verificabile in tempo polinomiale

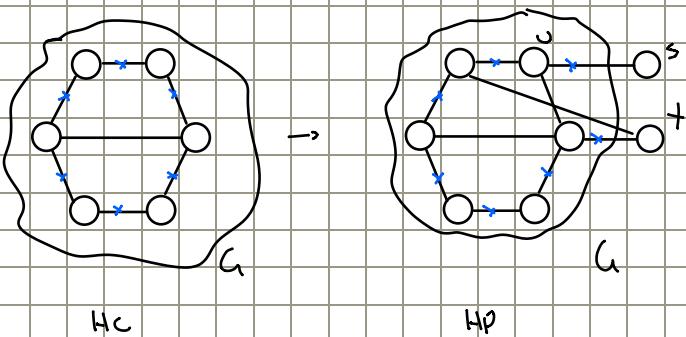
Un certificato è una sequenza di nodi $S = \langle v_1, \dots, v_n \rangle$, verifichiamo che S soddisfi Π_{HP} in tempo $O(|E||V| + |V|^2)$

2: dimostrare che $HP \in NPC$ $HC \leq HP$

Un grafo che contiene un ciclo hamiltoniano non contiene necessariamente un percorso hamiltoniano fra due nodi s, t . Trasformiamo quindi l'istanza.

$$\langle G = (V, E) \rangle \in HC \rightarrow \langle G' = (V', E'), s, t \rangle \in HP$$

s, t sono due nuovi nodi $s, t \notin V$, $V' = V \cup \{s, t\}$, per E' scegliamo un nodo $u \in V$, collegando $s \rightarrow u$ e collegando $t \rightarrow$ tutti i nodi che in G sono adiacenti a u : $E' = E \cup \{(s, u), (t, x) : (u, x) \in E\}$



tempo polinomiale per
costruirlo

Se G contiene un ciclo hamiltoniano $\langle v_1, \dots, v_n \rangle$ scegliamo $v_1 = v$, poiché $(v_i, v_{i+1}) \in E$

$\forall i=1, \dots, n$ e $v_i \neq v_j$ per $i \neq j \Rightarrow \langle s, v_1, \dots, v_n, t \rangle$ è un percorso hamiltoniano in G .

Se G' contiene un percorso hamiltoniano $\langle s, v_1, \dots, v_n, t \rangle$, poiché $(v_i, v_{i+1}) \in E \quad \forall i=1, \dots, n-1$ e $v_i \neq v_j$ per $i \neq j$ e poiché $(v_1, v_n) \in E$ (per costruzione di G' t è collegato a tutti i nodi adiacenti a v_n in G) $\Rightarrow \langle v_1, \dots, v_n \rangle$ è un ciclo hamiltoniano in G .

LONG PATH LP

Un long path è un percorso in G di almeno K archi.

$I_{LP} = \{ \langle G = (V, E), s, t, K \rangle : G \text{ è un grafo non orientato} \wedge s, t \in V \wedge K \in \mathbb{N} \}$

$S_{LP}(G, s, t, K) = \{ \langle v_1, \dots, v_h \rangle : \text{per } i=1, \dots, h \quad v_i \in V \}$

$\Pi_{LP}(G, s, t, K, S_{LP}(G, s, t, K)) = \exists \langle v_1, \dots, v_h \rangle \in S_{LP}(G, s, t, K) : s = v_1 \wedge t = v_h \wedge \forall i=1, \dots, h-1 \quad [(v_i, v_{i+1}) \in E] \wedge \forall i, j=1, \dots, h \text{ con } i \neq j \quad [v_i \neq v_j] \wedge h \geq K$

1: dimostriamo che $LP \in NP$ mostrando un certificato che sia verificabile in tempo polinomiale

Un certificato è una sequenza di nodi $S = \langle v_1, \dots, v_h \rangle$, verifichiamo che S soddisfi Π_{LP} in tempo $O(|E||V| + |V|^2)$

2: dimostrare che $LP \in NPC$ $HP \leq LP$

Questo problema è molto simile a HP , infatti HP è un'istanza di LP in cui $K = |V|$. Trasformiamo l'istanza.

$$\langle G, s, t \rangle \in HP \rightarrow \langle G', s', t', K \rangle \in LP \quad G' = G, s' = s, t' = t \text{ e } K = |V|$$

TRAVELLING SALESMAN PROBLEM TSP

Dato un grafo G non orientato, completo e pesato vedere se esiste un ciclo hamiltoniano tale che la somma dei pesi degli archi è $\leq K$.

$I_{TSP} = \{ \langle G = (V, E, w), K \rangle : G \text{ è un grafo non orientato} \wedge w: E \rightarrow \mathbb{N} \wedge (v, v) \in E \quad \forall \text{ coppia } v, v \in V \wedge K \in \mathbb{N} \}$

$S_{TSP}(G, K) = \{ \langle v_1, \dots, v_n \rangle : n = |V| \}$

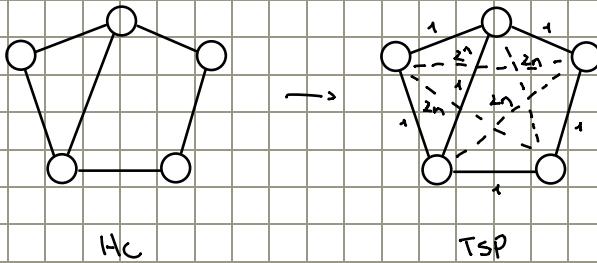
$\Pi_{TSP}(G, K, S_{TSP}(G, K)) = \exists \langle v_1, \dots, v_n \rangle \in S_{TSP}(G, K) : \forall i, j=1, \dots, n \text{ con } i \neq j \quad [v_i \neq v_j] \wedge \sum_{(v_i, v_j)} w(v_i, v_j) \leq K$

2: dimostrare che $TSP \in NPC$ $HC \leq TSP$

Questo problema è molto simile a HC ma abbiamo un grafo completo e pesato. Trasformiamo l'istanza.

$$\langle G = (V, E) \rangle \in HC \rightarrow \langle G' = (V, E', w), K \rangle \in TSP$$

E' è ottenuto aggiungendo ad E gli archi mancanti. $E' = E \cup \{(u, v) : u, v \in V \wedge (u, v) \notin E\}$, la funzione peso w è definita come: $\forall (u, v) \in E \quad w(u, v) = 1$, $\forall (u, v) \notin E \quad w(u, v) = 2|V| = 2n$



Se G contiene un ciclo hamiltoniano, tale ciclo è anche contenuto in G' : è costituito da $|V|$ archi contenuti in E dove la somma dei pesi in G' è $|V| \Rightarrow G'$ contiene un ciclo hamiltoniano di costo $\leq K$.

Se G contiene un ciclo hamiltoniano C tale che la somma degli archi è $\leq |V|$. C non può contenere archi appartenenti a $E - E$ in quanto gli archi hanno peso $2|V|$ perciò prendiamo solo archi contenuti in E ossia C è un ciclo hamiltoniano in G .

COLORABILITÀ COL

Dato un grafo G colorare ciascun nodo in G in modo che nodi adiacenti devono essere colorati con colori diversi.

$$I_{col} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato } \wedge K \in \mathbb{N} \}$$

$$S_{col}(G, K) = \{ c : V \rightarrow \{1, 2, \dots, K\} \}$$

$$\Pi_{col}(G, K, S_{col}(G, K)) = \exists c \in S_{col}(G, K) : \forall (u, v) \in E [c(u) \neq c(v)]$$

1: dimostriamo che $COL \in NP$ mostrando un certificato che sia verificabile in tempo polinomiale

Un certificato è una colorazione $c : V \rightarrow \{1, \dots, K\}$, verifichiamo che c è una colorazione per G dove $\forall (u, v) \in E \quad c(u) \neq c(v)$ in tempo $O(|E|)$

K-COLORABILITÀ K-COL

Dato un grafo G esiste un assegnazione di K colori in modo che nodi adiacenti devono essere colorati con colori diversi. K è costante.

$$I_{kcol} = \{ \langle G = (V, E) \rangle : G \text{ è un grafo non orientato} \}$$

$$S_{kcol}(G) = \{ c : V \rightarrow \{1, \dots, K\} \}$$

$$\Pi_{kcol}(G, S_{kcol}(G)) = \exists c \in S_{kcol}(G) : \forall (u, v) \in E [c(u) \neq c(v)]$$

$\text{1-COL} \leq \text{2-COL} \in P$, $\text{3-COL} \in \text{NPC}$, dimostriamo che $\text{4-COL} \in \text{NPC}$ con una $\leq 3\text{-COL}$

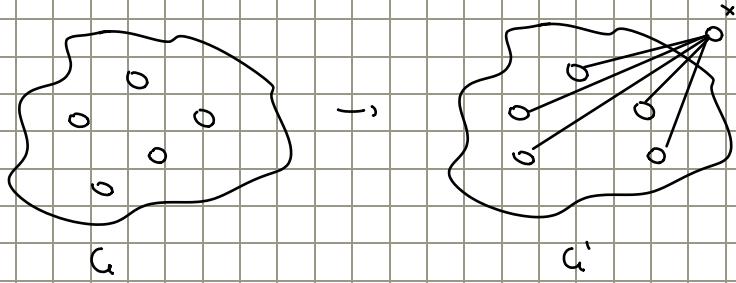
$3\text{-COL} \leq 4\text{-COL}$

$$\langle G = (V, E) \rangle \in 3\text{-COL} \rightarrow \langle G' = (V', E') \rangle \in 4\text{-COL}$$

V' è ottenuto aggiungendo a V un nuovo nodo x : $V' = V \cup \{x\}$

E' è ottenuto aggiungendo ad E gli archi che collegano x a tutti i nodi di V :

$$E' = E \cup \{(x, v) : v \in V\}$$



Se i nodi di G possono essere colorati con 3 colori allora chiamiamo 1,2,3 i colori e coloriamo con gli stessi colori i nodi $V' - \{x\}$, coloriamo x con il colore 4. G' è 4-colorabile.

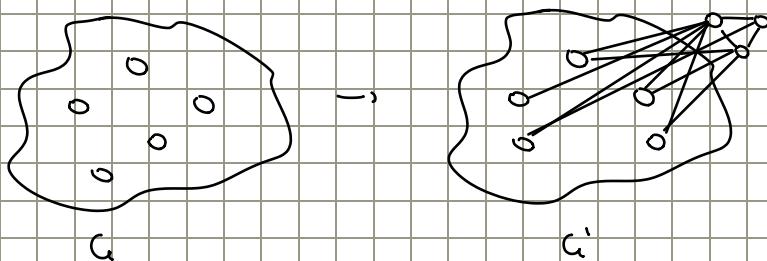
Se i nodi di G possono essere 4-colorati chiamiamo 4 il colore di x , poiché x è adiacente a tutti i nodi in V , 4 non può più essere utilizzato e $\forall u, v \in V : (u, v) \in E \quad c(u) \neq c(v)$. c colora i nodi di G con 3 colori quindi G è 3-COL.

$3\text{-COL} \leq K\text{-COL}$

Sia $K \in \mathbb{N}$ un intero fissato perciò costante si può dimostrare che $3\text{-COL} \leq K\text{-COL}$

$$\langle G = (V, E) \rangle \in 3\text{-COL} \rightarrow \langle G' = (V', E') \rangle \in K\text{-COL}$$

G' è ottenuto aggiungendo a G una clique C_K di $K-3$ nuovi nodi x_1, \dots, x_{K-3} e poi aggiungendo gli archi che collegano x_i a tutti i nodi in V .



$3\text{-COL} \leq \text{COL}$

$$\langle G = (V, E) \rangle \in 3\text{-COL} \rightarrow \langle G' = (V', E'), K \rangle \in \text{COL} \quad G' = G \text{ e } K = 3$$

Lez 24

Quando fissiamo un K costante, per la maggior parte dei valori K i problemi come K col e $3SAT$ rimangono NPc. Se per il problema $3SAT$ stabiliamo che il numero di variabili booleane è una costante $K \in \mathbb{N}$, $X = \{x_1, \dots, x_K\}$, X è un insieme costante, il numero di assegnazioni di verità a 2^K , costante. Verificare una alla volta tutte le assegnazioni di verità alla ricerca di una che soddisfi f ha costo $O(2^K |f|)$, con $|f|$ costante perché $\leq (2^K)^3$.

Possiamo quindi decidere un'istanza di $3SAT[K]$ in tempo costante, ossia $3SAT[K] \in P$.

Se h è una costante e $|X|=h$ allora il numero di clausole in f è al più $\binom{2^h}{3} \leq (2h)^3$.

Se $L \in NP \Rightarrow \exists T_v, c \in \mathbb{N} : \forall x \in \{0,1\}^{|x|} [x \in L \Leftrightarrow \exists y_x : |y_x| \leq |x|^c \wedge T_v(x, y_x) = q_a \text{ e } \text{dtime}(T_v, x, y_x) \in O(|x|^c)]$

Per dimostrarlo abbiamo costruito una macchina NT che opera come segue: input x

1: genera non deterministicamente una parola binaria y di lunghezza $\leq |x|^c$.

in tempo non deterministico $O(|x|^c)$.

2: invoca $T_v(x, y)$ e termina nello stesso stato.

in tempo deterministico $O(|x|^c)$.

$\forall x \in L \text{ dtime}(NT, x) \in O(|x|^c)$.

Possiamo trasformare NT in una macchina deterministica T che opera come segue: input x

1: genera deterministicamente l'insieme U di tutte le parole binarie y di lung. $\leq |x|^c$.

in tempo deterministico $O(2^{|x|^c})$.

2: $\forall y \in U$: invoca $T_v(x, y)$ e termina in q_a se $T_v(x, y) = q_a$, altrimenti se $\exists y \in U : T_v(x, y) = q_a$

termina in q_R .

in tempo deterministico $O(|x|^c 2^{|x|^c})$.

Te' un algoritmo di ricerca esauritiva per il linguaggio L che impiega tempo

esponenziale $\text{dtime}(T, x) \in O(|x|^c 2^{|x|^c})$.

Se, però, il numero di possibili parole y fosse polinomiale in x , cioè $|U| = |x|^K$ per K costante allora l'algoritmo di ricerca esauritiva impiegherebbe tempo polinomiale.

Cercare un vc di dimensione h in un grafo, con $h \in \mathbb{N}$ costante: un certificato di h - vc ha dimensione costante e quindi possibile generare tutti i possibili certificati in tempo

polinomiale. Per costruire \cup bisogna elencare tutti i sottoinsiemi di V contenenti h nodi cioè $\binom{|V|}{h} \leq |V|^h$ ossia $|\cup| \leq |V|^h$ e si può dimostrare che \cup si può costruire in tempo $O(|V||\cup|) = O(|V|^{h+1})$ ossia tempo polinomiale, quindi $h\text{-vc} \in P$.

Lo stesso ragionamento può essere ripetuto per molti problemi: sia $h \in \mathbb{N}$ una costante $h \geq 1$, $h \geq p$, $h \geq cL$.

Chiamiamo $h\text{-MIN3SAT}$ la versione di 3SAT dove bisogna decidere se esiste un'assegnazione di verità per X che soddisfi f e assegni il valore vero ad al più h variabili.

$\cup = \{X' \subseteq X : |X'| \leq h\}$, $|\cup| = \binom{|X|}{2} + \binom{|X|}{3} + \dots + \binom{|X|}{h} \leq h|X|^h$ ma h è costante perciò $h\text{-MIN3SAT} \in P$.

MIN2SAT

Se h non è una costante l'algoritmo di ricerca esaustiva ha complessità non polinomiale.

Dimostriamo che nonostante 2SAT $\in P$ MIN2SAT è NPC.

$$I_{\text{MIN2SAT}} = \{<X, f, h> : f \text{ è in 2CNF} \wedge h \in \mathbb{N}\}$$

$$S_{\text{MIN2SAT}}(X, f, h) = \{a : X \rightarrow \{0, 1\}\}$$

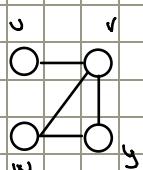
$$\Pi_{\text{MIN2SAT}}(X, f, h, S_{\text{MIN2SAT}}(X, f, h)) = \exists a \in S_{\text{MIN2SAT}}(X, f, h) : f(a(x_1), \dots, a(x_n)) = \text{vero} \wedge |\{x \in X : a(x) = \text{vero}\}| \leq K$$

$$vc \leq \text{MIN2SAT}$$

$$<(G = (V, E), K)> \in vc \rightarrow <X, f, h> \in \text{MIN2SAT}$$

Ad ogni nodo $v \in V$ associamo una variabile booleana x_v , a ogni arco $(u, v) \in E$ associamo

la clausola $c_{uv} = (x_u \vee x_v)$, poniamo $h = K$



$$\rightarrow f = (x_u \vee x_v) \wedge (x_w \vee x_y) \wedge (x_v \vee x_y) \wedge (x_w \vee x_y)$$

Se esiste in G un vc V' : $|V'| \leq K$ allora $\forall x_v \in X$ poniamo: $a(x_v) = \text{vero}$ se $v \in V'$, $a(x_v) = \text{falso}$ se $v \notin V'$, poiché $|V'| \leq K$ allora $|X| \leq h$, poiché $\forall (u, v) \in E [v \in V' \Rightarrow v \in V']$ allora $\forall c_{uv} [a(x_u) = \text{vero} \Rightarrow a(x_v) = \text{vero}]$: ossia a soddisfa f .

Se esiste $a(X)$ che soddisfa f : $\{x_v \in X : a(x_v) = \text{vero}\} \leq K$ allora $\forall v \in V$ poniamo $V' = \{v \in V : a(x_v) = \text{vero}\}$, poiché $|X| \leq h$ allora $|V'| \leq K$, poiché a soddisfa f , allora $\forall c_{uv} [a(x_u) = \text{vero} \Rightarrow a(x_v) = \text{vero}]$ quindi $\forall (u, v) \in E [v \in V' \Rightarrow v \in V']$ ossia V' è un vc .

Consideriamo una serie di problemi che combinano VC e 3COL:

1) VC v 3COL

2) VC \uparrow 3COL

3) VC v \uparrow 3COL

4) VC \wedge \uparrow 3COL

1) $I_{VC \vee 3COL} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$

$$S_{VC \vee 3COL}(G, K) = \{ (V, c) : V \subseteq V \wedge c: V \rightarrow \{1, 2, 3\} \}$$

$$\Pi_{VC \vee 3COL}(G, K, S_{VC \vee 3COL}(G, K)) = \exists (V, c) \in S_{VC \vee 3COL}(G, K) : [|V| \leq K \wedge \forall (u, v) \in E [c(u) \neq c(v)]] \vee [\forall (u, v) \in E [c(u) \neq c(v)]]$$

1: VC \vee 3COL \in NP

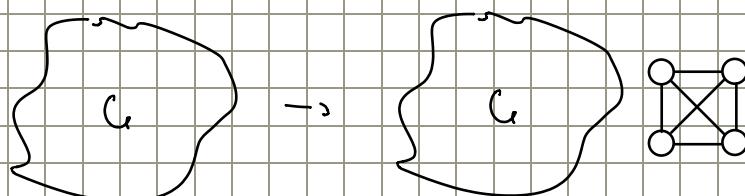
Un certificato è una coppia (V, c) e verificare se soddisfa il predicato richiede tempo polinomiale nell'istanza.

2: dimostrare VC \vee 3COL \in NPC

VC \leq VC \vee 3COL

$$\langle G = (V, E), K \rangle \in VC \rightarrow \langle G' = (V', E'), K' \rangle \in VC \vee 3COL$$

G' è ottenuto aggiungendo a G una clique di 4 nodi e ponendo $K' = K + 3$



Se G ha un vc di K nodi $\Rightarrow G'$ ha un vc di $K+3$ nodi.

Se G' è $VC \vee 3COL \Rightarrow$ poiché G' non è 3COL sicuramente avrà un istanza si di vc.

2) $I_{VC \wedge 3COL} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$

$$S_{VC \wedge 3COL}(G, K) = \{ (V, c) : V \subseteq V \wedge c: V \rightarrow \{1, 2, 3\} \}$$

$$\Pi_{VC \wedge 3COL}(G, K, S_{VC \wedge 3COL}(G, K)) = \exists (V, c) \in S_{VC \wedge 3COL}(G, K) : [|V| \leq K \wedge \forall (u, v) \in E [c(u) \neq c(v)]] \wedge [\forall (u, v) \in E [c(u) \neq c(v)]]$$

1: VC \wedge 3COL \in NP

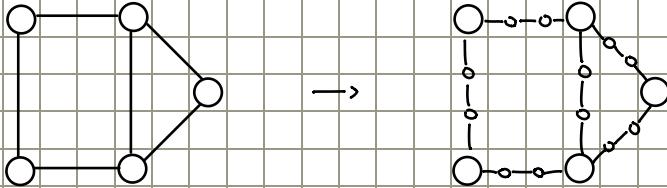
Un certificato è una coppia (V, c) e verificare se soddisfa il predicato richiede tempo polinomiale nell'istanza.

2: dimostrare $VC \cap 3COL \in NP$

$$VC \leq VC^* 3COL$$

$$\langle G = (V, E), K \rangle \in VC \rightarrow \langle G = (V, E), K \rangle \in VC^* 3COL$$

G' è ottenuto sostituendo ad ogni arco in G una catena di 4 nodi e ponendo $K' = K + |E|$



G' è 3COL perciò vi dimostrato che G' è istanza si di $VC^* 3COL$ sse ha un vc di $K + |E|$ nodi. In qualunque VC per G' i $|E|$ nodi devono essere quelli aggiunti per coprire gli archi nelle catene costruite.

Se G ha un vc di K nodi $\Rightarrow G'$ ha un vc di $K + |E|$ nodi.

Se G non ha un vc di K nodi $\Rightarrow G'$ non ha un vc di $K + |E|$ nodi.

$$4) I_{VC \cap 3COL} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{VC \cap 3COL}(G, K) = \{ (V', c) : V' \subseteq V \wedge c: V' \rightarrow \{1, 2, 3\} \}$$

$$\Pi_{VC \cap 3COL}(G, K, S_{VC \cap 3COL}(G, K)) = \exists (V', c) \in S_{VC \cap 3COL}(G, K) : \left[|V'| \leq K \wedge \forall (u, v) \in E[V \cup V'] \wedge \forall (V', c) \in S_{VC \cap 3COL}(G, K) : \left[\exists (u, v) \in E \left[c(u) = c(v) \right] \right] \right]$$

1: $VC \cap 3COL \in NP$

Un certificato è l'insieme di tutte le coppie (V', c) e verificare se soddisfa il predicato richiede tempo più che polinomiale nell'istanza in quanto bisogna verificare che in tutte le coppie V' non ci sia un vc di dimensione al più K e che, fra tutte queste coppie ce ne sia almeno una che è una 3-colorazione per G . Perciò $VC \cap 3COL \notin NP$

a) Si trova nella stessa situazione di 3).