7. TEORIA DELLA COMPLESSITÀ

Con lo studio precedente, ovvero quello della ***calcolabilità***, si è visto che nell'insieme di tutti i problemi vi sono alcuni che possiamo risolvere col calcolatore ed alcuni non risolvibili in questo modo, in più questo studio non considera la ***difficoltà*** dei problemi, distingue solo ciò che è risolubile da ciò che non lo è.

Anche quando un problema è decidibile, e quindi computazionalmente risolvibile, può non essere risolvibile praticamente, se la soluzione richiede una quantità eccessiva di tempo o di memoria.

Lo studio della ***complessità*** considera ***solo problemi solubili***, allo scopo di fornire una caratterizzazione dal punto di vista della ***quantità di risorse di calcolo necessario a risolverli*** (chiamata ***difficoltà di risoluzione***), qualunque algoritmo ha bisogno di una quantità minima di risorse.

Le ***risorse*** di cui si tiene principalmente conto quando si scrivono o si utilizzano programmi sono relative al ***tempo*** e allo ***spazio*** (ma non solo le uniche risorse usate durante il calcolo).

Ci limiteremo a considerare ***solo il tempo*** utilizzato per la soluzione di un problema e considereremo ***problemi di decisione***, quelli che hanno come soluzione una risposta sì o no, e quelli che sono ***decidibili***.

Ricorda:

Dato un problema decidibile lo possiamo esprimere come un linguaggio decidibile, formalmente $P \Leftrightarrow L_P$, e quello che dobbiamo considerare per andare a discutere di tempo di esecuzione è la dimensione dell'input, ad esempio se dobbiamo ordinare una lista di 10 elementi avremo bisogno di un determinato tempo, ma se gli elementi fossero milioni avremmo bisogno di un tempo diverso, quindi il tempo necessario non è un tempo assoluto ma dipende dalla dimensione dell'input.

Quando andiamo a considerare un linguaggio, la dimensione in questo caso è la lunghezza della stringa che rappresenta l'input, ovvero $|\langle x \rangle|$.

Esempio:

Se prendiamo un problema che prende in input un grafo e chiede se G è un grafo connesso sappiamo che il linguaggio associato a questo problema è l'insieme di tutte le stringhe che rappresentano un grafo G dove questo è connesso. La dimensione del problema per un grafo normalmente è il numero dei nodi, e quindi nella rappresentazione di G, la stringa che rappresenta il grafo avrà una lunghezza proporzionale al numero di nodi e quindi quello che ci interessa è la lunghezza della stringa che rappresenta il grafo in input.

G è un grafo connesso $\Leftrightarrow \{\langle G \rangle | G \text{ è un grafo connesso}\}$

Dimensione di G (numero nodi) $\Leftrightarrow |\langle G \rangle|$

Complessità temporale:

Andremo a considerare la quantità di tempo necessaria all'esecuzione di un programma per risolvere un certo problema, tecnicamente è chiamato ***complessità temporale***.

Il numero di passi che utilizza un algoritmo su un particolare input può dipendere da diversi parametri, per semplicità, si calcola il tempo di esecuzione di un algoritmo semplicemente in funzione della lunghezza della stringa che rappresenta l'input e non si considerano eventuali altri parametri.

Nell'analisi del ***caso peggiore***, che noi considereremo ovvero ***O-grande***, valuta il tempo di esecuzione massimo tra tutti gli input di una determinata lunghezza. Per utilizzare questa analisi asintotica, lo facciamo prendendo in considerazione solo il termine di ordine maggiore dell'espressione del tempo di esecuzione dell'algoritmo, trascurando sia il coefficiente di tale termine, che tutti i termini di ordine inferiore, perché il termine di ordine più alto domina gli altri termini quando l'input è grande.

Possiamo formalizzare tutto ciò nella seguente definizione:

Sia R^+ l'insieme dei numeri reali non negativi e siano f e g funzioni $f,g: N \to R^+$.
Si dice che ***$f(n)=O(g(n))$*** se esistono interi positivi c e n_0 tali che per ogni $n \geq n_0$, risulti ***$f(n) \leq cg(n)$***.

Quando f(n)=O(g(n)), diciamo che g(n) è un ***limite superiore*** per f(n), o più precisamente che g(n) è un ***limite superiore asintotico*** per f(n), per sottolineare che stiamo ignorando le costanti.

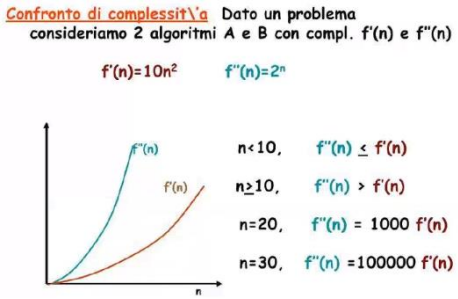
Esempio:

La funzione $f(n)= 6n^3+2n^2+20n+45$ ha quattro termini e quello di ordine maggiore è $6n^3$, trascuriamo il coefficiente 6 e diciamo che f è asintoticamente al più n^3. In generale, se si ha un polinomio vale $a_c n^c + \dots + a_1 n + a_0 = O(n^c)$.

Quindi la notazione asintotica (O-grande) per descrivere questo rapporto è $f(n)=O(n^3)$.

Altro esempio può essere: $3n \log_2 n+5n \log_2 \log_2 n+2=O(n \log n)$.

La nostra divisione fondamentale sarà tra ***polinomiale*** (indipendentemente dall'esponente) ed ***esponenziale*** (indipendentemente dalla base). È importante fare queste due distinzioni perché la variazione di n può essere molto significativa nei due casi, ad esempio:



Espressione O	Nome informale
$O(1)$	costante
$O(\log n)$	logaritmico (base?)
$O(n)$	lineare
$O(n^2)$	quadratico
$O(n^3)$	cubico
$O(n^k), k \geq 1$	polinomiale
$O(d^n), d \geq 2$	esponenziale

Espressione O	Nome informale
$O(n^k), k \geq 1$	polinomiale
$O(d^n), d \geq 2$	esponenziale

7.1 COMPLESSITÀ TEMPORALE DI UNA MACCHINA DI TURING

Sia $M = (Q, \sum, \Gamma, f, q_0, q_{accept}, q_{reject})$ una MdT deterministica, che si arresta su ogni input. La complessità temporale di M è la funzione $f: N \to N$ dove f (n) è il massimo numero di passi eseguiti da M su un qualsiasi input di lunghezza n, per ogni $n \in N$.
Cioè f (n) = massimo numero di passi in $q_0 w \to^* uqv$, $q \in \{q_{accept}, q_{reject}\}$, al variare di w in $\sum^n$.

Se M ha complessità temporale f(n), diremo che ***M decide L(M) in tempo f(n).***

La complessità temporale dipende dalla codifica utilizzata, codificare un numero intero *n* in base 2 richiede $\lceil \log n + 1 \rceil$ (= più piccolo intero $\geq n + 1$) cifre binarie, mentre codificarlo in base unaria richiede *n* cifre unarie. Essendo l'input più lungo, la macchina ha più tempo a disposizione.

Avere una complessità temporale $O(n)$ rispetto alla codifica unaria, può voler dire che la complessità temporale rispetto alla codifica binaria sia $O(2^n)$.

Esempio:

PRIMO: Dato un numero intero *x*, *x* è primo?

Algoritmo semplice: dividi *x* per tutti gli interi $i < x$. Se tutti i resti di tali divisioni sono diversi da zero, *x* è primo.

Richiede: ponendo **n = |⟨x⟩|**

- $O(n)$ passi se *x* è rappresentato in unario,
- $O(2^n)$ passi se *x* è rappresentato in base 2.

Quindi se ho un intero 1000, non vale mille ma lo rappresento in binario nella macchina e quindi ho bisogno non di mille bit ma della lunghezza della stringa che me lo rappresenta è lunga dieci, e quindi quando vado a considerare il numero di passi, se faccio 10 passi ho una complessità lineare, se invece faccio mille passi ho una complessità 2^{10} e quindi è esponenziale. Quindi per gli interi consideriamo una codifica in base 2 o una qualsiasi altra pur che si maggiore o uguale a 2. Per i grafi, che dobbiamo rappresentare nodi e archi, in generale possiamo rappresentarli mediante liste delle adiacenze o matrici di adiacenze. La rappresentazione per insiemi, relazioni e funzioni mediante enumerazione delle codifiche dei relativi elementi. Codifiche "ragionevoli" dei dati sono ***polinomialmente correlate***: è possibile passare da una di esse a una qualunque altra codifica "ragionevole" delle istanze dello stesso problema in un tempo polinomiale rispetto alla rappresentazione originale.

Le varianti di macchine di Turing deterministiche possono simularsi tra di loro con un sovraccarico computazionale ***polinomiale***.

Anche gli altri modelli di calcolo possono simularsi con un sovraccarico computazionale polinomiale, ad eccezione di quelle ***non deterministiche***.

Esaminiamo come la scelta del modello di calcolo può influire sulla complessità di tempo dei linguaggi.

Teorema:

Sia t(n) una funzione tale che ***t(n) ≥ n***. Per ogni MdT multinastro M con complessità temporale t(n) esiste una MdT a nastro singolo M' con complessità temporale ***$O(t^2(n))$***.

Analizziamo i passi delle due macchine per determinare la quantità di tempo supplementare richiesta.

Dimostrazione:

Sia M una TM a k-nastri avente tempo di esecuzione t(n), costruiamo una TM S a singolo nastro che ha tempo di esecuzione $O(t^2(n))$. La macchina S opera simulando M. Nel rivedere tale simulazione, ricordiamo che S utilizza il suo unico nastro per rappresentare il contenuto di tutti i k nastri di M. Inizialmente, S mette il nastro nei formato che rappresenta tutti i nastri di M e poi simula i passi di M. Per simulare un passo, S scorre tutte le informazioni memorizzate sul suo nastro per determinare i simboli presenti sotto le testine di M. Poi S esegue un'altra scansione del suo nastro per aggiornare il contenuto dei nastri e le posizioni delle testine. Se una testina di M si sposta a destra su una parte non letta del nastro, S deve aumentare la quantità di spazio allocato per questo nastro. Lo fa spostando una parte del suo nastro di una cella a destra.

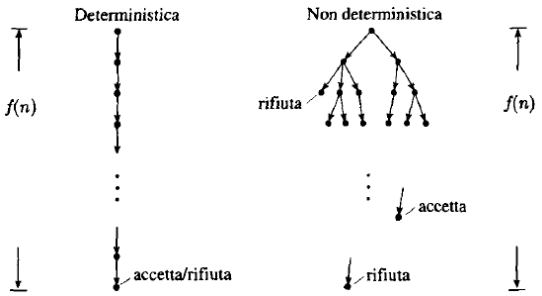
Da quanto detto, per ogni passo di M, la macchina S fa due passi sulla parte attiva del suo nastro. Il primo ottiene le informazioni necessarie per determinare la prossima mossa e il secondo la esegue. La lunghezza della parte attiva del nastro di S determina il tempo che S impiega per eseguire la scansione, quindi dobbiamo determinare un limite superiore per questa lunghezza. Per farlo prendiamo la somma delle lunghezze delle parti attive dei k nastri di M. Ciascuna di queste parti attive ha lunghezza al più t(n) poiché M utilizza t(n) Celle del nastro in t(n) passi, se la testina si sposta verso destra ad ogni passo e anche meno se vi sono spostamenti di qualche testina a sinistra. Quindi una scansione della parte di nastro attiva di S impiega O(t(n)) passi. Per simulare ciascuna delle fasi di M, S esegue due scansioni ed eventualmente fino a k spostamenti a destra. Ognuno impiega un tempo O(t(n)), per cui il tempo totale per S per simulare un passo di M è O(t(n)). Ora possiamo limitare il tempo totale impiegato dalla simulazione. La fase iniziale, in cui S mette il nastro nel formato corretto, usa O(n) passi. In seguito, S simula ciascuno dei t(n) passi di M, utilizzando O(t(n)) passi, per cui questa parte della simulazione utilizza $t(n) \times O(t(n)) = O(t^2(n))$ passi. Quindi l'intera simulazione di M utilizza $O(n) + O(t^2(n))$ passi.

Abbiamo assunto che $t(n) \geq n$ (un'ipotesi ragionevole perché M non potrebbe nemmeno leggere l'intero input in meno tempo). Pertanto, il tempo di esecuzione di S è $O(t^2(n))$, quindi ***MdT multinastro e a singolo nastro non fa variare il tipo di complessità***.

Teorema:

Sia t(n) una funzione tale che ***t(n) ≥ n***. Per ogni MdT a nastro singolo, non deterministica N e con complessità temporale t(n) esiste una MdT a nastro singolo, deterministica e con complessità temporale ***$2^{O(t(n))}$***.

Dimostrazione:



Sia N una TM non deterministica avente tempo di esecuzione t(n). Costruiamo una TM deterministica D che simula N effettuando una ricerca sull'albero delle computazioni di N. Su un input di lunghezza n, ogni ramificazione dell'albero delle computazioni di N ha lunghezza al più t(n). Ogni nodo dell'albero pub avere al più b figli, dove b il massimo numero di scelte possibili in accordo alla funzione di transizione di N. Così il numero totale di foglie nell'albero è al massimo $b^{t(n)}$. La simulazione procede esplorando l'albero prima in ampiezza. In altre parole, si visitano tutti i nodi a profondità d prima di passare ad uno qualsiasi dei nodi a profondità d + 1. Inizia inefficientemente dalla radice e si sposta in basso verso un nodo ogni volta che visita il nodo stesso. Tuttavia, l'eliminazione di tale inefficienza non altera l'enunciato del Teorema, quindi la lasciamo in questa forma. Il numero totale di nodi dell'albero inferiore al doppio del numero di foglie, quindi è limitato da $O(b^{t(n)})$. Il tempo per partire dalla radice e raggiungere un nodo è O(t(n)). Pertanto, il tempo di esecuzione di D è $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.

Come descritto dal teorema che afferma che per una MdT non deterministica esiste una MdT deterministica equivalente, la MdT D ha tre nastri. Convertirla in una a singolo nastro al più fa si che si elevi al quadrato il tempo di esecuzione, per il teorema precedente. Quindi il tempo di esecuzione di quello a singolo nastro è $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$, quindi ***non manteniamo la complessità***.

7.2 CLASSE P

Vogliamo definire classi chiuse rispetto al cambio del modello di calcolo utilizzato e al cambio di rappresentazione dei dati.

La ***classe P*** è l'insieme dei linguaggi L per i quali esiste una MdT M con un solo nastro che decide L e per cui $t_M(n) = O(n^k)$ per qualche $k \geq 1$.

Ricordando la tesi di Church-Turing il quale afferma che tutto quello che risulta computabile può essere computato da una MdT deterministica.

La versione forte di questa tesi afferma la ***correlazione polinomiale*** nel tempo tra algoritmi e computazione di una MdT deterministica.

Quindi, possiamo definire *P* come l'insieme dei problemi computazionali che ammettono un ***algoritmo polinomiale***, i così detti problemi "***trattabili***".

I problemi che sono risolvibili in teoria, ma non possono essere risolti nella pratica, sono chiamati "***intrattabili***", ad esempio al suo interno troviamo diversi problemi che richiedono almeno $n^{1000000}$ operazioni, ma contiene anche problemi naturali come determinare se un numero è primo.