

CORSO DI ALGORITMI E STRUTTURE DATI

INTRODUZIONE AI PROBLEMI NP-COMPLETI

NUOVA VERSIONE DEL CAPITOLO 13 DELLE DISPENSE
(BASATA SUI MODELLI NON DETERMINISTICI)

ANNO ACCADEMICO 2007/2008

Corso di laurea in Informatica
Università degli Studi di Milano

Massimiliano Goldwurm

Indice

1	Problemi intrattabili	1
2	La classe P	3
3	Macchine non deterministiche	3
4	La classe NP	7
5	Il problema della soddisfacibilità	9
6	Riducibilità polinomiale	10
6.1	Riduzione polinomiale da SODD-FNC a CLIQUE	11
6.2	Riduzione polinomiale da SODD-FNC a 3-SODD-FNC	12
7	Il teorema di Cook	12
7.1	Macchine di Turing	13
7.2	Dimostrazione	17

Gli argomenti principali trattati nel nostro corso riguardano la sintesi e l'analisi degli algoritmi e in particolare le tecniche di progettazione di un algoritmo e i metodi che permettono di valutare le risorse di calcolo utilizzate (tempo e spazio). Questi temi sono stati illustrati nelle dispense del corso cui faremo riferimento in questa nota.

Vogliamo ora spostare l'attenzione sui problemi, studiando la possibilità di classificare questi ultimi in base alla quantità di risorse necessarie per ottenere la soluzione. Si è riscontrato infatti, che per certi gruppi di problemi, le difficoltà incontrate per trovare un algoritmo efficiente sono sostanzialmente le stesse. Tenendo conto dei risultati ottenuti nella letteratura, possiamo grossolanamente raggruppare i problemi in tre categorie:

1. I problemi che ammettono algoritmi di soluzione efficienti;
2. I problemi che per loro natura non possono essere risolti mediante algoritmi efficienti e che quindi sono intrattabili;
3. I problemi per i quali algoritmi efficienti non sono stati trovati ma per i quali nessuno ha finora provato che tali algoritmi non esistano.

Molti problemi di notevole interesse appartengono al terzo gruppo e presentano tra loro caratteristiche così simili dal punto di vista algoritmico che risulta naturale introdurre metodi e tecniche che consentano di studiarne le proprietà complessivamente. Questo ha portato a definire e studiare le cosiddette "classi di complessità", cioè classi di problemi risolubili utilizzando una certa quantità di risorse (per esempio di tempo oppure di spazio). Questi strumenti consentono anche di confrontare la difficoltà intrinseca dei vari problemi, verificando per esempio se un problema dato è più o meno facile di un altro, o se è possibile trasformare un algoritmo per il primo in uno per il secondo che richieda all'incirca la stessa quantità di risorse.

Tra le classi di complessità definite in base al tempo di calcolo quelle più note sono le classi P e NP. Queste contengono gran parte dei problemi considerati in letteratura e le problematiche legate allo studio delle loro proprietà sono in realtà comuni all'analisi di molte altre classi di complessità. L'interesse nello studio di questi argomenti è inoltre legato alla presenza di molti problemi aperti alcuni dei quali sono fra i più importanti dell'informatica teorica.

1 Problemi intrattabili

Descriviamo ora con maggior precisione le tre classi di problemi sopra menzionate ¹.

La principale classe di problemi considerata in letteratura è quella dei problemi risolubili in tempo polinomiale. Vengono così chiamati quei problemi che ammettono un algoritmo di soluzione il cui tempo di calcolo, nel caso peggiore, è limitato da un polinomio nelle dimensioni

¹È bene ricordare che l'analisi che svolgiamo si riferisce comunque a problemi che ammettono un algoritmo di soluzione. Infatti, è noto che non tutti i problemi possono essere risolti mediante un algoritmo. I problemi che *non* ammettono un algoritmo di soluzione sono detti *non risolubili* o *indecidibili* e sono stati ampiamente studiati a partire dagli anni '30. Tra questi quello più noto è il *problema dell'arresto*; nel nostro contesto, tale problema può essere formulato nel modo seguente:

Istanza : un programma RAM P e un input I per P .

Domanda : il programma P si arresta sull'input I ?

Questo è solo l'esempio più famoso di una vasta gamma di problemi indecidibili che riguardano il comportamento dei programmi oppure le proprietà delle funzioni da loro calcolate.

dell'input. Questa classe può essere definita mediante diversi modelli di calcolo ed è sostanzialmente robusta, cioè non dipende dal particolare modello considerato. Intuitivamente essa rappresenta la classe dei problemi “trattabili” cioè quelli che ammettono un algoritmo di soluzione efficiente. Quasi tutti i problemi considerati nei capitoli precedenti appartengono a questa classe. Chiaramente è poco realistico considerare trattabile un problema risolubile solo da algoritmi che richiedono un tempo dell'ordine di n^k con k elevato. Tuttavia, quasi tutti i problemi di interesse, risolubili in tempo polinomiale, ammettono un algoritmo di soluzione che richiede $O(n^3)$ passi su un input di dimensione n . Inoltre la maggior parte di questi problemi hanno algoritmi che ammettono tempi poco più che lineari e questo spesso significa, per esempio, poter risolvere il problema in pochi secondi anche su istanze di dimensione relativamente elevate ($n = 10^6$). Ricordiamo che la classe dei problemi di decisione risolubili in tempo polinomiale è nota in letteratura come classe P .

La naturale controparte della classe appena descritta è quella dei problemi che *non* possono essere risolti in un tempo polinomiale. Per questi problemi si può provare che ogni algoritmo risolutivo richiede, nel caso peggiore, un tempo di calcolo esponenziale o comunque asintoticamente superiore ad ogni polinomio. Questi problemi sono chiamati “intrattabili” perché, pur essendo risolubili per via automatica, richiedono un tempo di calcolo molto elevato, tale da rendere ogni algoritmo di fatto inutilizzabile anche per dimensioni piccole dell'input. Nella tabella presentata nella sezione 1.3 abbiamo visto per esempio che con tempi di calcolo dell'ordine di 2^n , anche eseguendo 10^6 operazioni al secondo, sono necessari parecchi anni per risolvere istanze di dimensione $n = 50$.

Sono certamente intrattabili tutti quei problemi che ammettono soluzione di dimensione esponenziale rispetto all'input. Un esempio è costituito dal problema di determinare i circuiti hamiltoniani di un grafo, ovvero i cicli che passano una e una volta sola per ogni nodo. È chiaro infatti che se il grafo possiede n nodi, può esistere un numero esponenziale di circuiti hamiltoniani (se il grafo è completo ve ne sono $n - 1!$).

Vi sono poi altri problemi (in verità piuttosto rari) che pur ammettendo una uscita di dimensione limitata non possono essere risolti in tempo polinomiale. In generale dimostrare una tale proprietà è abbastanza difficile e sono pochi i risultati di questo genere in letteratura ².

Le due classi precedenti sono ben definite ed essendo l'una il complementare dell'altra, dovrebbero contenere tutti i problemi risolubili. In realtà molti problemi di notevole interesse non sono stati collocati in nessuna delle due classi. Non sono stati trovati, per questi problemi, algoritmi di soluzione che funzionano in tempo polinomiale e neppure è stato dimostrato che tali algoritmi non esistono. Tra questi ricordiamo, come esempio, alcuni problemi di decisione già incontrati nei capitoli precedenti e altri, dall'evidente significato intuitivo, che definiremo formalmente nelle sezioni successive: Clique, Knapsack 0-1, Circuito hamiltoniano, Commesso viaggiatore. Si tratta di problemi classici ampiamente studiati in letteratura che trovano applicazioni in vari settori e per i quali sono noti algoritmi che funzionano in tempo esponenziale o subesponenziale, oppure algoritmi di approssimazione polinomiali.

La classe più rappresentativa di questo folto gruppo è quella dei problemi NP -completi. Questa è stata definita all'inizio degli anni '70 e raccoglie una grande varietà di problemi che sorgono naturalmente in vari settori dell'informatica, della ricerca operativa e della matematica

²Si veda per esempio J.E.Hopcroft, J.D.Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979.

discreta. Su di essi è stata sviluppata una notevole letteratura ³ e il loro numero è ormai di parecchie migliaia, raggruppati e studiati per settori specifici.

I problemi *NP*-completi sono problemi di decisione definiti mediante un paradigma basato sulla nozione di macchina non deterministica (ma anche qui si possono dare diverse definizioni equivalenti). Tali problemi sono computazionalmente equivalenti fra loro, nel senso che un eventuale algoritmo polinomiale per uno solo di essi implicherebbe l'esistenza di un analogo algoritmo per tutti gli altri. Proprio l'assenza di una tale procedura, nonostante gli sforzi compiuti, ha portato alla congettura che i problemi *NP*-completi non siano risolvibili in tempo polinomiale e quindi non siano contenuti nella classe *P* (anche se finora nessuno ha dimostrato un tale risultato). Questa congettura è ormai così affermata in ambito informatico che stabilire la *NP*-completezza di un problema equivale di fatto a qualificare il problema come intrattabile o comunque talmente difficile che nessun ricercatore sarebbe oggi in grado di definire un algoritmo efficiente per risolverlo.

2 La classe P

Ricordiamo innanzitutto che un *problema di decisione* è un problema che ammette solo due possibili soluzioni (intuitivamente “sì” o “no”). Esso può essere rappresentato da una coppia $\langle I, q \rangle$, dove I è l'insieme delle istanze del problema, mentre q è un predicato su I , ovvero una funzione $q : I \rightarrow \{0, 1\}$. Nel seguito rappresenteremo spesso il predicato q mediante una opportuna domanda relativa a una istanza qualsiasi $x \in I$.

Definiamo allora *P* come la classe dei problemi di decisione risolvibili da una RAM in tempo polinomiale secondo il criterio di costo logaritmico. Quindi, usando i simboli introdotti nel capitolo 3, possiamo affermare che un problema di decisione π appartiene a *P* se e solo se esiste una RAM Ψ che risolve π e un polinomio $p(n)$ tali che, per ogni $n \in \mathbf{N}$ e ogni input x di dimensione n ,

$$T_{\Psi}^l(x) \leq p(n).$$

Chiaramente, per dimensione di una istanza x intendiamo il numero di bit necessari per rappresentarla.

È bene osservare che nella definizione precedente il criterio logaritmico non può essere sostituito con quello uniforme. Infatti i due criteri possono fornire valutazioni molto diverse fra loro, anche se per molti programmi RAM i corrispondenti tempi di calcolo sono polinomialmente legati tra loro. In particolare, è abbastanza facile descrivere programmi RAM che, su un input x , richiedono un tempo lineare in $|x|$ secondo il criterio uniforme e uno esponenziale secondo quello logaritmico. Abbiamo già incontrato un programma di questo genere nell'esempio 3.3 .

3 Macchine non deterministiche

Introduciamo in questa sezione un modello di calcolo non deterministico. Si tratta di un modello per alcuni aspetti controintuitivo ma tuttavia importante se vogliamo cogliere a pieno il significato della classe *NP* che sarà introdotta nella sezione successiva.

³Si veda in proposito M.R.Garey, D.S.Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman, 1979.

In parole povere possiamo dire che una macchina non deterministica è un modello di calcolo che può compiere una o più scelte durante ogni computazione. Di conseguenza il suo funzionamento su un dato input non è più determinato da un'unica computazione ma da un insieme di computazioni distinte, una per ogni possibile sequenza di scelte compiute. La macchina in questo caso è un semplice accettore, ovvero ogni computazione di arresto termina restituendo in uscita 1 oppure 0. Diremo allora che una macchina non deterministica risolve un problema di decisione se, per ogni istanza che ammette risposta positiva, esiste una computazione della macchina su tale istanza che restituisce il valore 1 e, per ogni istanza che ammette risposta negativa, tutte le computazioni relative forniscono in uscita il valore 0.

Prima di dare la definizione formale del modello che consideriamo ricordiamo il funzionamento di una macchina RAM tradizionale seguendo la descrizione data nel capitolo 3. Data una macchina RAM M e un input x , la computazione di M su x consiste in una sequenza (finita o infinita) di configurazioni

$$C_0, C_1, \dots, C_i, \dots,$$

ciascuna delle quali descrive l'immagine della macchina in un qualunque istante del calcolo prima dell'esecuzione di ciascuna istruzione. Ogni configurazione definisce quindi il contenuto dei registri $R_0, R_1, \dots, R_n, \dots$, il valore del contatore che indica l'istruzione corrente da eseguire, il contenuto dei due nastri di input e output e la posizione della testina di ingresso. La configurazione C_0 è quella iniziale, nella quale tutti i registri R_i sono azzerati, il nastro di uscita contiene solo blank e quello di ingresso l'input x , le due testine sono posizionate sulle prime celle dei rispettivi nastri e il contatore indica la prima istruzione del programma. Ogni C_i , con $i \geq 1$, viene ottenuta dalla configurazione precedente eseguendo l'istruzione indicata dal contatore in C_{i-1} . Diciamo che la macchina M passa dalla configurazione C_{i-1} alla configurazione C_i in un passo e rappresentiamo questa transizione mediante l'espressione $C_{i-1} \vdash_M C_i$. Inoltre, se l'istruzione corrente in C_i non è ben definita (cioè il valore del contatore non indica correttamente una istruzione del programma), allora C_i è di arresto e non esiste una configurazione C' tale che $C_{i-1} \vdash_M C'$. In questo caso la computazione è una sequenza finita e C_i è l'ultima configurazione raggiunta dalla macchina. Se questo non si verifica allora la sequenza C_0, C_1, \dots è infinita e diciamo che la macchina M non si *arresta* sull'input considerato.

In questo modo \vdash_M definisce una relazione binaria sull'insieme delle configurazioni della macchina M , chiamata anche *relazione di transizione*. Essa è una relazione univoca per ogni macchina RAM M , nel senso che per ogni configurazione C esiste al più una configurazione C' tale che $C \vdash_M C'$.

Una macchina RAM *non deterministica* è una macchina RAM nel cui programma oltre alle istruzioni note possono comparire istruzioni della forma

$$\text{CHOICE}(0, 1)$$

L'esecuzione di questa istruzione avviene semplicemente scegliendo di inserire nell'accumulatore (cioè il registro R_0) il valore 0 oppure il valore 1. La macchina sceglie quindi di eseguire l'istruzione $\text{LOAD} = 0$ oppure l'istruzione $\text{LOAD} = 1$. L'esecuzione delle altre istruzioni rimane invariata rispetto al modello tradizionale (che nel seguito chiameremo *deterministico*) così come restano invariate le altre caratteristiche della macchina.

Anche per una RAM non deterministica M possiamo definire la nozione di configurazione e quella di relazione di transizione \vdash_M . Ovviamente in questo caso la relazione \vdash_M non è più

univoca: per ogni configurazione C su M , se l'istruzione corrente indicata in C è un'istruzione di scelta CHOICE(0, 1), esisteranno due configurazioni C_1 e C_2 che si ottengono da C eseguendo rispettivamente LOAD = 0 e LOAD = 1; di conseguenza avremo $C \vdash_M C_1$ e $C \vdash_M C_2$.

Tutto questo implica che, per ogni input x di M , vi saranno in generale più computazioni distinte di M su x , tutte della forma

$$C_0, C_1, \dots, C_i, \dots,$$

tali che $C_{i-1} \vdash_M C_i$ per ogni $i \geq 1$ e dove C_0 è la configurazione iniziale di M su input x . Tali computazioni si ottengono semplicemente eseguendo le istruzioni di scelta via via incontrate in tutti i modi possibili.

L'insieme delle computazioni di M su un dato input sono rappresentabili da un albero con radice, dotato eventualmente di un numero infinito di nodi, che gode delle seguenti proprietà:

- (i) ogni nodo è etichettato da una configurazione di M ;
- (ii) la radice è etichettata dalla configurazione iniziale sull'input assegnato;
- (iii) se un nodo v è etichettato da una configurazione C ed esiste una sola configurazione C' tale che $C \vdash_M C'$, allora v possiede un solo figlio etichettato da C' ;
- (iv) se un nodo v è etichettato da una configurazione C ed esistono due configurazioni C_1 e C_2 tali che $C \vdash_M C_1$ e $C \vdash_M C_2$, allora v possiede due figli etichettati rispettivamente da C_1 e C_2 ;
- (iv) un nodo v è una foglia se e solo se v è etichettato da una configurazione di arresto.

Dalla definizione precedente è evidente che una computazione di M su un dato input è determinata da un ramo dell'albero di computazione corrispondente, ovvero da un cammino che va dalla radice a una foglia.

Poiché in una macchina non deterministica non esiste una sola computazione su un dato input, dobbiamo definire un paradigma particolare per descrivere il problema risolto da un modello di questo genere. Diremo che un problema di decisione $\pi = \langle I, q \rangle$ è risolto da una RAM non deterministica M se si verificano le condizioni seguenti:

1. per ogni $x \in I$ tale che $q(x) = 1$ esiste una computazione di M su input x che termina restituendo il valore 1 in uscita (si dice anche che la computazione *accetta* l'input);
2. per ogni $x \in I$ tale che $q(x) = 0$ tutte le computazioni di M su input x terminano restituendo il valore 0 in uscita (*rifiutano* l'input).

Nota che in questo modo la stessa macchina M non può essere usata per risolvere il problema complementare di π , definito da $\pi^c = \langle I, p \rangle$ con $p(x) = 1 - q(x)$.

Esempio 1

Considera il seguente problema di decisione:

CLIQUE

Istanza: un grafo non orientato $G = \langle V, E \rangle$ e un intero $k \leq \#V$.

Domanda: esiste una clique di dimensione k in G , ovvero un insieme $C \subseteq V$ di cardinalità k tale che $\{v, w\} \in E$ per ogni coppia di nodi distinti $v, w \in C$?

Il problema può essere risolto da una RAM non deterministica che esegue la seguente procedura:

```
begin
  A := ∅
  for v ∈ V do
```

```

begin
  i := Choice(0,1)
  if i = 1 then A := A ∪ {v}
end
if #A = k ∧ A forma una clique
  then return 1
  else return 0
end

```

Come si verifica facilmente, ogni computazione della macchina è divisa in due fasi distinte: nella prima si costruisce un insieme A di nodi usando l'istruzione di scelta, si dice anche che la macchina *genera in modo non deterministico* l'insieme A ; nella seconda fase si verifica se A è effettivamente una clique di dimensione k in G . Nota che questa seconda parte del calcolo è puramente deterministica perché l'istruzione di scelta non viene usata. Inoltre sul grafo $G = \langle V, E \rangle$ la macchina ammette esattamente una computazione per ogni sottoinsieme di V . Quindi se n è il numero dei nodi di G , abbiamo 2^n possibili computazioni distinte sull'input $G = \langle V, E \rangle$.

Chiaramente la procedura (non deterministica) risolve il problema CLIQUE secondo il paradigma formulato: se $G = \langle V, E \rangle$ ammette una clique C di dimensione k la computazione che genera il sottoinsieme C accetta l'input; se invece $G = \langle V, E \rangle$ non ammette una clique di dimensione k tutte le computazioni rifiutano.

Notiamo infine che il numero di computazioni che accettano l'input coincide con il numero di clique di dimensione k in G . ■

Esempio 2

Un altro esempio significativo è relativo al problema del circuito hamiltoniano:

CIRCUITO HAMILTONIANO

Istanza: un grafo $G = \langle V, E \rangle$ non orientato.

Domanda: esiste in G un circuito hamiltoniano, ovvero una permutazione (v_1, v_2, \dots, v_n) dei nodi del grafo tale che $\{v_i, v_{i+1}\} \in E$ per ogni $i = 1, \dots, n-1$, e inoltre $\{v_n, v_1\} \in E$?

Questo problema può essere risolto da una RAM non deterministica M che, in una prima fase, genera in modo non deterministico una sequenza $(v_{i_1}, v_{i_2}, \dots, v_{i_n})$ di vertici di G ($n = \#V$); in seguito M controlla che tale sequenza sia una permutazione dei nodi del grafo, ovvero che tutti i suoi elementi siano distinti, e quindi verifica se, per ogni $j = 1, 2, \dots, n-1$, $\{v_{i_j}, v_{i_{j+1}}\}$ appartiene a E e se $\{v_{i_n}, v_{i_1}\} \in E$. Se tutti questi controlli danno esito positivo la macchina accetta, altrimenti rifiuta.

Il suo funzionamento è descritto dalla procedura seguente:

```

begin
  siano  $v_1, v_2, \dots, v_n$  i vertici di  $G$ 
  A :=  $\Lambda$  (lista vuota)
  for j = 1, 2, ..., n do
    begin
      k := 1
      for i = 1, 2, ..., n-1 do {  $\ell := \text{Choice}(0,1)$ 
                               {  $k := k + \ell$ 
      A := INSERISCI_IN_TESTA(A, v_k)
    end
  if esistono due nodi uguali in A
    then return 0
  else if la sequenza di nodi in A forma un ciclo
    then return 1
    else return 0
end

```

4 La classe NP

Il tempo di calcolo di una macchina non deterministica è definito dalla più lunga computazione della macchina sull'input considerato. Nel nostro caso per valutare il tempo di calcolo di una RAM non deterministica assumiamo il criterio logaritmico. Nel seguito tutti i tempi di calcolo saranno valutati assumendo questo criterio.

Quindi, data una RAM non deterministica M e un input x per M , il tempo richiesto da M su input x è il massimo tempo di calcolo richiesto da una computazione di M su input x secondo il criterio logaritmico. Denoteremo tale quantità con $T_M(x)$. Se esiste una computazione che non termina poniamo chiaramente $T_M(x) = +\infty$. Se invece tutte le computazioni hanno termine possiamo considerare l'albero di computazione di M su x , associando ad ogni lato il costo logaritmico dell'istruzione corrispondente; in questo modo ogni cammino dalla radice a una foglia (computazione) possiede un costo dato dalla somma dei costi dei suoi lati (questi non è altro che il tempo logaritmico della corrispondente computazione). $T_M(x)$ coincide allora con il valore massimo tra i costi di questi cammini.

Inoltre, per ogni $n \in \mathbf{N}$, denotiamo con $T_M(n)$ il massimo tempo di calcolo richiesto da M su un input di dimensione n (come al solito la dimensione di un input in questo caso è data dal numero di bit necessari per rappresentarlo). Quindi

$$T_M(n) = \max\{T_M(x) \mid |x| = n\}$$

Nel seguito diremo anche che M funziona in tempo $f(n)$ se $T_M(n) \leq f(n)$ per ogni $n \in \mathbf{N}$ (con $f : \mathbf{N} \rightarrow \mathbf{N}$ funzione qualsiasi).

La classe NP è allora la classe dei problemi di decisione risolvibili da una macchina RAM non deterministica che funziona in tempo polinomiale. Quindi un problema di decisione π appartiene a NP se esiste un polinomio $p(x)$ e una macchina RAM non deterministica M che risolve π , tali che $T_M(n) \leq p(n)$ per ogni $n \in \mathbf{N}$.

È facile verificare che le procedure descritte negli esempi 1 e 2 corrispondono a RAM non deterministiche che funzionano in tempo polinomiale. Di conseguenza i problemi CLIQUE e CIRCUITO HAMILTONIANO appartengono a NP.

Un problema che sorge ora in modo naturale è quello di confrontare i tempi di calcolo delle macchine deterministiche e di quelle non deterministiche. Se per esempio un problema di decisione è risolvibile in tempo $f(n)$ su una RAM non deterministica, ci chiediamo in quanto tempo possiamo risolvere lo stesso problema usando una RAM tradizionale (il problema inverso non si pone perché ogni RAM deterministica è una particolare RAM non deterministica). Una macchina non deterministica può essere simulata da una deterministica che semplicemente esplora l'albero di computazione della precedente. Il procedimento è illustrato dalla seguente proposizione che mostra anche come i due modelli di calcolo risolvano la stessa classe di problemi di decisione anche se impiegando tempi di calcolo differenti.

Proposizione 1 *Per ogni funzione $f : \mathbf{N} \rightarrow \mathbf{N}$ e ogni problema di decisione π risolvibile da una RAM non deterministica in tempo $f(n)$, esiste un valore $c > 1$ e una RAM deterministica che risolve π in tempo $O(c^{f(n)})$.*

Dimostrazione. Sia M una RAM non deterministica che risolve π in tempo $f(n)$. L'idea è quella di simulare M mediante una RAM deterministica M' che, su input x , esplora l'albero

di computazione di M su x mediante una ricerca in ampiezza. Durante il calcolo Q rappresenta la coda che mantiene le configurazioni di M raggiungibili dalla configurazione iniziale. Il funzionamento della macchina M' sull'input x è definito dalla seguente procedura.

```

begin
  Sia  $C_0(x)$  la configurazione iniziale di  $M$  su input  $x$ 
   $Q := \text{ENQUEUE}(\Lambda, C_0(x))$ 
   $A := 0$ 
  while  $Q \neq \Lambda \wedge A = 0$  do
    begin
       $C := \text{FRONT}(Q)$ 
       $Q := \text{DEQUEUE}(Q)$ 
      if  $C$  accettante then  $A = 1$ 
      if esiste una sola configurazione  $C'$  tale che  $C \vdash_M C'$ 
        then  $Q := \text{ENQUEUE}(Q, C')$ 
      if esistono due configurazioni  $C_1, C_2$  tali che  $C \vdash_M C_1$  e  $C \vdash_M C_2$ 
        then  $\begin{cases} Q := \text{ENQUEUE}(Q, C_1) \\ Q := \text{ENQUEUE}(Q, C_2) \end{cases}$ 
    end
  return  $A$ 
end

```

Non è difficile verificare che ogni configurazione di M su un input x , dove $|x| = n$, può essere rappresentata da una sequenza di interi di dimensione complessiva $O(f(n))$. Data la rappresentazione di una configurazione C di M , la macchina M' deve essere in grado di determinare la rappresentazione delle configurazioni successive, cioè delle configurazioni C' tali che $C \vdash_M C'$. Questo calcolo può essere certamente eseguito in un tempo $O(f(n))^2$, tenendo anche conto del costo di indirizzo degli interi coinvolti.

Ritornando al comportamento di M su input x , osserviamo che nel corrispondente albero di computazione vi sono al più $2^{f(n)+1}$ nodi. Quindi il ciclo while della precedente procedura può essere eseguito al più $O(2^{f(n)})$ volte. Ogni ciclo esegue un numero costante di operazioni su configurazioni di M e, per l'osservazione precedente, ogni singola esecuzione richiede al più $O(f(n))^2$ passi. Il tempo complessivo risulta così $O(f(n)^2 2^{f(n)}) = O(3^{f(n)})$. ■

Una immediata conseguenza della proposizione precedente è data dal seguente

Corollario 2 *Ogni problema in NP è risolvibile su una RAM deterministica in tempo $O(c^{p(n)})$ per qualche $c > 1$ e qualche polinomio $p(x)$.*

Questo significa che tutti i problemi in NP possono essere risolti in tempo esponenziale da una macchina deterministica.

Esercizio

Dimostrare che il seguente problema appartiene a NP:

COMMESSO VIAGGIATORE

Istanza: un intero $k > 0$, un grafo diretto $G = \langle V, E \rangle$ e una funzione peso $d : E \rightarrow \mathbf{N}$.

Domanda: esiste un circuito che congiunge tutti i nodi del grafo di peso minore o uguale a k , ovvero una permutazione (v_1, v_2, \dots, v_n) di V tale che $\sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_n, v_1) \leq k$?

5 Il problema della soddisfacibilità

In questa sezione illustriamo il problema della soddisfacibilità per formule booleane in forma normale congiunta. Questo problema ha una particolare importanza in questo contesto perché è stato il primo problema NP-completo introdotto in letteratura.

Innanzitutto ricordiamo che, per definizione, una formula booleana è un letterale, cioè una variabile x o una variabile negata \bar{x} , oppure una delle seguenti espressioni:

- (i) $(\neg\phi)$,
- (ii) $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_k$,
- (iii) $(\phi_1 \vee \phi_2 \vee \cdots \vee \phi_k)$,

dove $k \geq 2$, mentre $\phi, \phi_1, \phi_2, \dots, \phi_k$ sono formule booleane. Nel seguito rappresenteremo con i simboli di somma e prodotto le tradizionali operazioni \vee e \wedge . Ogni formula booleana nella quale compaiono k variabili distinte definisce in modo ovvio una funzione $f : \{0, 1\}^k \rightarrow \{0, 1\}$.

Diciamo che una formula booleana ϕ è in forma normale congiunta (FNC per brevità) se ϕ è un prodotto di clausole di letterali, ovvero

$$\phi = E_1 \cdot E_2 \cdot \dots \cdot E_k \quad (1)$$

dove $E_i = (\ell_{i1} + \ell_{i2} + \cdots + \ell_{it_i})$ per ogni $i = 1, 2, \dots, k$, e ciascun ℓ_{ij} è un letterale.

Un esempio di formula booleana in FNC è dato dall'espressione

$$U(y_1, y_2, \dots, y_k) = \left(\sum_{i=1}^k y_i \right) \cdot \prod_{i \neq j} (\bar{y}_i + \bar{y}_j). \quad (2)$$

È evidente che $U(y_1, y_2, \dots, y_k)$ vale 1 se e solo se esattamente una delle sue variabili y_i assume valore 1.

Si può dimostrare che ogni funzione booleana può essere rappresentata da una formula in forma normale congiunta. Per verificare questa semplice proprietà, introduciamo la seguente definizione: per ogni variabile x e ogni $c \in \{0, 1\}$, poniamo

$$\bar{x}^c = \begin{cases} \bar{x} & \text{se } c = 1 \\ x & \text{se } c = 0 \end{cases}$$

Nota che \bar{x}^c assume il valore 1 se e solo se i valori di x e c sono diversi.

Lemma 3 *Sia $f : \{0, 1\}^n \rightarrow \{0, 1\}$ una funzione booleana e sia $S_0 = \{\underline{c} \in \{0, 1\}^n / f(\underline{c}) = 0\} \neq \emptyset$. Allora per ogni $(x_1, x_2, \dots, x_n) \in \{0, 1\}^n$, denotando con \underline{c} il vettore (c_1, c_2, \dots, c_n) , abbiamo*

$$f(x_1, x_2, \dots, x_n) = \prod_{\underline{c} \in S_0} (\bar{x}_1^{c_1} + \bar{x}_2^{c_2} + \cdots + \bar{x}_n^{c_n})$$

Dimostrazione. Osserviamo che, per la definizione precedente, la clausola $(\bar{x}_1^{c_1} + \bar{x}_2^{c_2} + \cdots + \bar{x}_n^{c_n})$ ha valore 1 se e solo se, per qualche i , x_i assume un valore diverso da c_i . Ne segue che la produttoria precedente è uguale a 1 se e solo se la n -pla (x_1, x_2, \dots, x_n) assume un valore in $\{0, 1\}^n$ che non appartiene a S_0 . \square

Il problema della soddisfacibilità per formule booleane in FNC è definito nel modo seguente:

SODD-FNC

Istanza: una formula booleana ϕ in forma normale congiunta.

Domanda: esiste un assegnamento di valori 0 e 1 alle variabili che rende vera ϕ ?

È facile provare che questo problema appartiene a NP. Infatti, su input ϕ , una RAM può generare in modo non deterministico un assegnamento A di valori alle variabili di ϕ . Quindi la macchina verifica in modo deterministico se l'assegnamento A rende vera ϕ . In caso affermativo la macchina accetta, altrimenti rifiuta. È evidente che ogni computazione della macchina termina dopo un numero polinomiale di passi.

Proposizione 4 *SODD-FNC appartiene alla classe NP.*

Esercizi

1) Dare la definizione di formula booleana in forma normale disgiunta e dimostrare che ogni funzione booleana, a un solo valore, può essere rappresentata da una formula di questo tipo.

2) Il problema della soddisfacibilità per formule booleane in forma normale *disgiunta* appartiene a P?

6 Riducibilità polinomiale

Consideriamo due problemi di decisione $\pi_1 = \langle I_1, q_1 \rangle$ e $\pi_2 = \langle I_2, q_2 \rangle$, dove I_1 e I_2 sono gli insiemi delle istanze, mentre q_1 e q_2 i rispettivi predicati. Diciamo che π_1 è *polinomialmente riducibile* a π_2 se esiste una funzione $f : I_1 \rightarrow I_2$, calcolabile in tempo polinomiale da una RAM deterministica (secondo il criterio logaritmico), tale che per ogni $x \in I_1$, $q_1(x) = 1$ se e solo se $q_2(f(x)) = 1$. Diremo anche che f definisce una riduzione polinomiale da π_1 a π_2 o anche che π_1 è riducibile a π_2 mediante la funzione f .

È facile verificare che la riduzione polinomiale gode della proprietà transitiva. Infatti, dati tre problemi di decisione $\pi_1 = \langle I_1, q_1 \rangle$, $\pi_2 = \langle I_2, q_2 \rangle$ e $\pi_3 = \langle I_3, q_3 \rangle$, se π_1 è polinomialmente riducibile a π_2 mediante una funzione f , e π_2 è polinomialmente riducibile a π_3 mediante una funzione g , allora la funzione composta $h(x) = g(f(x))$ definisce una riduzione polinomiale da π_1 a π_3 : per ogni $x \in I_1$, $q_1(x) = 1$ se e solo se $q_3(h(x)) = 1$; inoltre la funzione h è calcolabile da una RAM che, su input $x \in I_1$, simula prima la macchina che calcola f , ottenendo $f(x)$, e quindi la macchina che calcola g , ricavando $g(f(x))$. Il tempo richiesto da quest'ultimo calcolo è comunque polinomiale perché maggiorato dalla composizione di due polinomi.

Le classi P e NP definite nelle sezioni precedenti sono chiaramente chiuse rispetto alla riduzione polinomiale.

Proposizione 5 *Dati due problemi di decisione $\pi_1 = \langle I_1, q_1 \rangle$ e $\pi_2 = \langle I_2, q_2 \rangle$, supponiamo che π_1 sia polinomialmente riducibile a π_2 . Allora se π_2 appartiene a P anche π_1 appartiene a P. Analogamente, se π_2 appartiene a NP, anche π_1 appartiene a NP.*

Dimostrazione. Sia f la funzione che definisce una riduzione polinomiale da π_1 a π_2 e sia M una RAM che calcola f in tempo $p(n)$, per un opportuno polinomio p . Se $\pi_2 \in P$ allora esiste una RAM deterministica M' che risolve π_2 in tempo $q(n)$, dove q è un'altro polinomio. Allora possiamo definire una RAM deterministica M'' che, su input $x \in I_1$, calcola la stringa $y = f(x) \in I_2$ simulando la macchina M ; quindi M'' simula la macchina M' su input $f(x)$ e

mantiene la risposta di quest'ultima. Poiché la lunghezza di $f(x)$ è al più data da $p(|x|)$, la complessità in tempo di M'' è maggiorata da un polinomio:

$$T_{M''}(|x|) \leq p(|x|) + q(p(|x|))$$

In modo analogo si prova che, se π_2 appartiene a NP, anche π_1 appartiene a NP. ■

Una immediata conseguenza della proprietà precedente è data dal seguente corollario.

Corollario 6 *Dati due problemi di decisione π_1 e π_2 , supponiamo che π_1 sia polinomialmente riducibile a π_2 . Allora se π_1 non appartiene a P anche π_2 non appartiene a P. Analogamente, se π_1 non appartiene a NP, anche π_2 non appartiene a NP.*

6.1 Riduzione polinomiale da SODD-FNC a CLIQUE

Presentiamo ora un esempio di riduzione polinomiale tra problemi di decisione.

Proposizione 7 *SODD-FNC è polinomialmente riducibile a CLIQUE.*

Dimostrazione. Descriviamo una funzione f tra le istanze dei due problemi che definisce la riduzione. Sia ϕ una formula in FNC definita dalla uguaglianza

$$\phi = E_1 \cdot E_2 \cdot \dots \cdot E_k$$

dove $E_i = (\ell_{i1} + \ell_{i2} + \dots + \ell_{it_i})$ per ogni $i = 1, 2, \dots, k$, e ciascun ℓ_{ij} è un letterale. Allora $f(\phi)$ è l'istanza del problema CLIQUE data dall'intero k , pari al numero di clausole di ϕ , e dal grafo $G = \langle V, E \rangle$ definito nel modo seguente:

- $V = \{[i, j] \mid i \in \{1, 2, \dots, k\}, j \in \{1, 2, \dots, t_i\}\}$;
- $E = \{\{[i, j], [u, v]\} \mid i \neq u, \ell_{ij} \neq \overline{\ell_{uv}}\}$.

Quindi i nodi di G sono esattamente le occorrenze di letterali in ϕ , mentre i suoi lati sono le coppie di letterali che compaiono in clausole distinte e che non sono l'uno il negato dell'altro.

È facile verificare che la funzione f è calcolabile in tempo polinomiale.

Proviamo ora che ϕ ammette un assegnamento che la rende vera se e solo se G ha una clique di dimensione k . Supponiamo che esista un tale assegnamento A . Chiaramente A assegna valore 1 ad almeno un letterale ℓ_{ij_i} per ogni clausola E_i di ϕ . Sia $\{j_1, j_2, \dots, j_k\}$ l'insieme degli indici di questi letterali. Allora l'insieme $C = \{[i, j_i] \mid i = 1, 2, \dots, k\}$ è una clique del grafo G perché, se per qualche $i, u \in \{1, 2, \dots, k\}$, $\ell_{ij_i} = \overline{\ell_{uj_u}}$, l'assegnamento A non potrebbe rendere veri entrambi i letterali.

Viceversa, sia $C \subseteq V$ una clique di G di dimensione k . Allora, per ogni coppia $[i, j], [u, v]$ in C , abbiamo $i \neq u$ e $\ell_{ij} \neq \overline{\ell_{uv}}$. Sia ora S_1 l'insieme delle variabili x tali che $x = \ell_{ij}$ per qualche $[i, j] \in C$. Analogamente, denotiamo con S_0 l'insieme delle variabili y tali che $\overline{y} = \ell_{ij}$ per qualche $[i, j] \in C$. Definiamo ora l'assegnamento A che attribuisce valore 1 alle variabili in S_1 e valore 0 alle variabili in S_0 . A è ben definito perché, essendo C una clique, l'intersezione $S_1 \cap S_0$ è vuota. Inoltre A rende vera la formula ϕ , perché per ogni clausola E_i vi è un letterale ℓ_{ij_i} che assume valore 1. ■

6.2 Riduzione polinomiale da SODD-FNC a 3-SODD-FNC

Il problema SODD-FNC può essere ridotto polinomialmente anche a una sua variante che si ottiene restringendo le istanze alle formule booleane in FNC che possiedono solo clausole con al più 3 letterali.

3-SODD-FNC

Istanza: un formula booleana $\psi = F_1 \cdot F_2 \cdot \dots \cdot F_k$, dove $F_i = (\ell_{i1} + \ell_{i2} + \ell_{i3})$ per ogni $i = 1, 2, \dots, k$ e ogni ℓ_{ij} è una variabile o una variabile negata.

Domanda: esiste un assegnamento di valori 0 e 1 alle variabili che rende vera ψ ?

Proposizione 8 *SODD-FNC è polinomialmente riducibile a 3-SODD-FNC.*

Dimostrazione. Sia ϕ una formula booleana in FNC e sia $E = (\ell_1 + \ell_2 + \dots + \ell_t)$ una clausola di ϕ dotata di $t \geq 4$ letterali. Sostituiamo E in ϕ con un prodotto di clausole $f(E)$ definito da

$$f(E) = (\ell_1 + \ell_2 + y_1) \cdot (\ell_3 + \overline{y_1} + y_2) \cdot (\ell_4 + \overline{y_2} + y_3) \cdot \dots \cdot (\ell_{t-2} + \overline{y_{t-4}} + y_{t-3}) \cdot (\ell_{t-1} + \ell_t + \overline{y_{t-3}})$$

dove y_1, y_2, \dots, y_{t-3} sono nuove variabili.

Proviamo ora che esiste un assegnamento che rende vera la clausola E se e solo se ne esiste uno che rende vera $f(E)$. Infatti, se per qualche i $\ell_i = 1$, basta porre $y_j = 1$ per ogni $j < i - 1$ e $y_j = 0$ per ogni $j \geq i - 1$; in questo modo otteniamo un assegnamento che rende vera $f(E)$. Viceversa, se esiste un assegnamento che rende vera $f(E)$ allora deve esistere qualche letterale ℓ_i che assume valore 1. Altrimenti è facile verificare che il valore di $f(E)$ sarebbe 0. Ne segue che lo stesso assegnamento rende vera la E .

Operando questa sostituzione per tutte le clausole di ϕ dotate di più di 3 addendi, otteniamo una formula 3-FNC che soddisfa le condizioni richieste. È inoltre evidente che il tempo di calcolo necessario per realizzare la sostituzione è polinomiale. ■

7 Il teorema di Cook

Diciamo che un problema di decisione π è *NP-completo* se π appartiene a NP e inoltre ogni altro problema in NP è polinomialmente riducibile a π .

Intuitivamente i problemi NP-completi sono i problemi più difficili nella classe NP. L'esistenza di un algoritmo che risolve in tempo polinomiale un problema NP-completo implicherebbe l'equivalenza delle classi P e NP. Tuttavia l'ipotesi $P=NP$ è considerata molto improbabile, dato l'elevato numero di problemi in NP per i quali non sono stati trovati algoritmi polinomiali, anche se finora nessuno ha formalmente provato che le due classi sono diverse⁴. Quindi dimostrare che un problema π è NP-completo significa sostanzialmente provare che il problema è intrattabile ovvero che, quasi certamente, π non ammette algoritmi di soluzione che lavorano in tempo polinomiale.

Il primo problema NP-completo scoperto in letteratura è proprio il problema SODD-FNC definito nella sezione 5. Questo risultato, provato da Cook nel 1971, è molto importante perché ha consentito di determinare l'NP-completezza di molti altri problemi per i quali non erano noti algoritmi di risoluzione polinomiali. Questo da una parte ha permesso di spiegare l'inerte difficoltà di questi problemi, dall'altra ha dato avvio allo studio delle loro proprietà comuni.

⁴Il problema di dimostrare che P è diverso da NP è considerato oggi uno dei più importanti problemi aperti della teoria degli algoritmi.

Teorema 9 *Il problema SODD-FNC è NP-completo.*

Osserviamo subito che, poiché la riducibilità polinomiale gode della proprietà transitiva, se SODD-FNC è polinomialmente riducibile a un problema $\pi \in NP$, anche quest'ultimo risulta NP-completo. Applicando quindi i risultati presentati nella sezione precedente, possiamo enunciare la seguente proposizione.

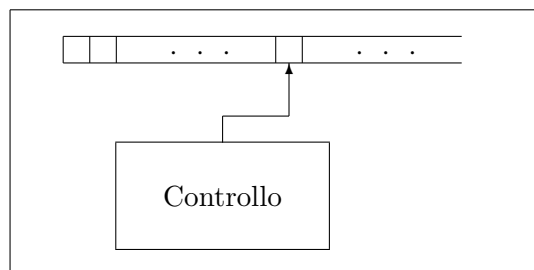
Corollario 10 *I problemi CLIQUE e 3-SODD-FNC sono NP-completi.*

Questo corollario mostra come, usando la riduzione polinomiale sia possibile dimostrare l'NP-completezza di un problema. Quasi tutti i problemi NP-completi noti in letteratura sono stati ottenuti mediante riduzione polinomiale da un problema dello stesso tipo.

La dimostrazione tradizionale del teorema di Cook richiede l'introduzione di un modello di calcolo elementare, più semplice rispetto alle macchine RAM. Si tratta della nota macchina di Turing introdotta già negli anni '30 per studiare le proprietà dei problemi indecidibili. Questo modello di calcolo è computazionalmente equivalente alle RAM (assumendo il costo logaritmico) ma è troppo elementare per descrivere in maniera efficace il funzionamento di algoritmi e procedure come sono oggi comunemente intese. La sua semplicità tuttavia consente di provare in maniera abbastanza diretta alcune proprietà generali di calcolo tra le quali anche l'esistenza delle riduzioni polinomiali menzionate proprio nel teorema di Cook.

7.1 Macchine di Turing

Intuitivamente una macchina di Turing (MdT nel seguito) è un modello di calcolo costituito da un insieme di stati, un nastro suddiviso in celle e una testina di lettura posizionata su una di queste. L'insieme degli stati contiene uno stato particolare, chiamato stato iniziale e un sottoinsieme di stati detti finali. Ogni cella contiene un solo simbolo estratto da un alfabeto fissato. Il funzionamento della macchina è determinato da un controllo centrale che consente di eseguire una sequenza di mosse a partire da una configurazione iniziale. In questa configurazione lo stato corrente è quello iniziale, una stringa di input è collocata nelle prime celle del nastro e la testina legge il primo simbolo; tutte le altre celle contengono un simbolo speciale che chiameremo blank. Quindi ogni mossa è univocamente determinata dallo stato nel quale la macchina si trova e dal simbolo letto dalla testina. Eseguendo una mossa la macchina può entrare in un nuovo stato, stampare un nuovo simbolo nella cella su cui è posizionata la testina di lettura e, infine, spostare quest'ultima di una posizione a destra oppure a sinistra. Se la sequenza di mosse eseguite è finita diciamo che la macchina si arresta sull'input considerato e diciamo che tale input è accettato se lo stato raggiunto nell'ultima configurazione è finale. In questo modo si può definire precisamente il problema di decisione risolto da una MdT: diciamo che la macchina M risolve un problema di decisione $\pi = \langle I, q \rangle$ se I è l'insieme delle possibili stringhe di input della macchina, M si arresta su ogni input $x \in I$ e accetta x se e solo se $q(x) = 1$.



Formalmente possiamo quindi definire una macchina di Turing (deterministica) come una vettore

$$M = \langle Q, \Sigma, \Gamma, q_0, B, \delta, F \rangle$$

dove Q è un insieme finito di stati, Γ un alfabeto di lavoro, $\Sigma \subset \Gamma$ un alfabeto di ingresso, $B \in \Gamma \setminus \Sigma$ un simbolo particolare che denota il blank, $q_0 \in Q$ lo stato iniziale, $F \subseteq Q$ l'insieme degli stati finali e δ una funzione transizione, cioè una funzione parziale

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{-1, +1\}$$

Per ogni $q \in Q$ e ogni $a \in \Gamma$, il valore $\delta(q, a)$ definisce la mossa di M quando la macchina si trova nello stato q e legge il simbolo a : se $\delta(q, a) = (p, b, \ell)$ allora p rappresenta il nuovo stato, b il simbolo scritto nella cella di lettura, ℓ lo spostamento della testina. La testina si sposta rispettivamente di una cella a sinistra o a destra a seconda se $\ell = -1$ oppure $\ell = +1$.

Una configurazione di M è l'immagine complessiva della macchina prima o dopo l'esecuzione di una mossa. Questa è determinata dallo stato corrente, dal contenuto del nastro e dalla posizione della testina di lettura. Definiamo quindi una configurazione di M come una stringa ⁵

$$\alpha q \beta$$

dove $q \in Q$, $\alpha \in \Gamma^*$, $\beta \in \Gamma^+$. α rappresenta la stringa collocata a sinistra della testina di lettura mentre β , seguita da infiniti blank, è la sequenza di simboli che si trova alla sua destra. In particolare la testina di lettura è posizionata sul primo simbolo di β . Inoltre supporremo che B non sia un suffisso di β a meno che $\beta = B$. In questo modo $\alpha\beta$ definisce la parte significativa del nastro: nel seguito diremo che essa rappresenta la porzione non blank del nastro. Denoteremo inoltre con \mathcal{C}_M l'insieme delle configurazioni di M .

La configurazione iniziale di M su input $w \in \Sigma^*$ è $q_0 B$, se w è la parola vuota, mentre è la stringa $q_0 w$, se $w \in \Sigma^+$. Nel seguito denoteremo con $C_0(w)$ tale configurazione. Diremo inoltre che una configurazione $\alpha q \beta$ è accettante se $q \in F$.

⁵Ricordiamo che una stringa (o parola) su un dato alfabeto A è una concatenazione finita di simboli estratti da A . L'insieme di tutte le stringhe su A , compresa la parola vuota ϵ , è denotato da A^* , mentre A^+ rappresenta l'insieme delle parole su A diverse da ϵ . La lunghezza di una parola x è il numero di simboli che compaiono in x ed è denotata da $|x|$. Chiaramente $|\epsilon| = 0$.

Possiamo ora definire la relazione di transizione in un passo. Questa è una relazione binaria \vdash_M sull'insieme \mathcal{C}_M delle configurazioni di M . Intuitivamente, per ogni $C, C' \in \mathcal{C}_M$, vale $C \vdash_M C'$ se la macchina M nella configurazione C raggiunge mediante una mossa la configurazione C' . Più precisamente, data la configurazione $\alpha q \beta \in \mathcal{C}_M$, supponiamo che $\beta = b\beta'$ dove $b \in \Gamma$, $\beta' \in \Gamma^*$ e che $\delta(q, b) = (p, c, \ell)$. Distinguiamo quindi i seguenti casi:

1) se $\ell = +1$ allora

$$\alpha q b \beta' \vdash_M \begin{cases} \alpha c p \beta' & \text{se } \beta' \neq \epsilon \\ \alpha c p B & \text{se } \beta' = \epsilon \end{cases}$$

2) se $\ell = -1$ e $|\alpha| \geq 1$ allora, ponendo $\alpha = \alpha'a$ con $\alpha' \in \Gamma^*$ e $a \in \Gamma$, abbiamo

$$\alpha q b \beta' \vdash_M \begin{cases} \alpha' p a & \text{se } c = B \text{ e } \beta' = \epsilon \\ \alpha' p a c \beta' & \text{altrimenti} \end{cases}$$

Nel seguito denotiamo con \vdash_M^* la chiusura riflessiva e transitiva di \vdash_M .

Osserviamo che se $\delta(q, b)$ non è definito, oppure $\ell = -1$ e $\alpha = \epsilon$, allora non esiste alcuna configurazione $C' \in \mathcal{C}_M$ tale che $\alpha q \beta \vdash_M C'$. In questo caso diciamo che $\alpha q \beta$ è una configurazione di arresto per M . Senza perdita di generalità possiamo supporre che ogni configurazione accettante sia una configurazione di arresto.

Una sequenza finita $\{C_i\}_{0 \leq i \leq m}$ di configurazioni di M è una computazione di M su input $w \in \Sigma^*$ se $C_0 = C_0(w)$, $C_{i-1} \vdash_M C_i$ per ogni $i = 1, 2, \dots, m$ e C_m è una configurazione di arresto per M . Se inoltre C_m è accettante diciamo che M accetta l'input w , altrimenti diciamo che M rifiuta.

Se invece la macchina M su input w non si ferma la sua computazione è definita da una sequenza infinita di configurazioni $\{C_i\}_{0 \leq i < +\infty}$, tali che $C_0 = C_0(w)$ e $C_{i-1} \vdash_M C_i$ per ogni $i \geq 1$.

Supponi ora che M si arresti su ogni input $x \in \Sigma^*$. Allora diciamo che M risolve il problema di decisione $\langle \Sigma^*, q \rangle$ dove, per ogni $x \in \Sigma^*$,

$$q(x) = \begin{cases} 1 & \text{se } M \text{ accetta } x \\ 0 & \text{altrimenti} \end{cases}$$

Esempio 3

Consideriamo ora il linguaggio $L = \{x \in \{a, b\}^* \mid x = x^R\}$, dove x^R è l'inversa di x . Si può facilmente descrivere una MdT che verifica se una stringa appartiene a L . Tale macchina, su un input $y \in \{a, b\}^*$ di lunghezza n , confronta i simboli di posizione i e $n - i + 1$ e accetta se questi sono uguali tra loro per ogni $i = 1, 2, \dots, n$. Il calcolo avviene percoccando n volte la stringa di input alternando la direzione di spostamento della testina e marcando opportunamente i simboli letti. ■

Definiamo ora la nozione di tempo di calcolo di una MdT su un dato input. Data una MdT $M = \langle Q, \Sigma, \Gamma, q_0, B, \delta, F \rangle$, per ogni $w \in \Sigma^*$, denotiamo con $T_M(w)$ il numero di mosse compiute da M su input w . Se M non si arresta poniamo $T_M(w) = +\infty$. Inoltre, per ogni $n \in \mathbf{N}$, denotiamo con $T_M(n)$ il massimo numero di mosse compiute da M su un input di lunghezza n :

$$T_M(n) = \max\{T_M(w) \mid w \in \Sigma^*, |x| = n\}$$

In questo modo T_M è una funzione $T_M : \mathbf{N} \longrightarrow \mathbf{N} \cup \{+\infty\}$ che chiameremo complessità in tempo di M . Data una funzione $f : \mathbf{N} \longrightarrow \mathbf{R}^+$, diremo che M lavora in tempo $f(n)$ se $T_M(n) \leq f(n)$ per ogni $n \in \mathbf{N}$. Se inoltre $f(n)$ è un polinomio in n diremo che M funziona in tempo polinomiale.

Esempio 4

È facile verificare che la MdT descritta nell'esempio 3 lavora in tempo $O(n^2)$. ■

È evidente che per ogni MdT possiamo definire una macchina RAM che esegue la stessa computazione (a patto di codificare opportunamente i simboli dell'alfabeto di lavoro). Vale inoltre una proprietà inversa che consente di determinare, per ogni macchina RAM, una MdT equivalente. Inoltre i tempi di calcolo delle due macchine sono polinomialmente legati tra loro.

Proposizione 11 *Se un problema di decisione π è risolubile in tempo $T(n)$ da una RAM M secondo il criterio logaritmico, allora esiste una MdT M' che risolve π in tempo $p(T(n))$ per un opportuno polinomio p .*

Questo significa che la classe P coincide con la classe dei problemi di decisione risolubili da una MdT in tempo polinomiale.

Come per le RAM anche per le macchine di Turing possiamo definire un modello non deterministico.

Formalmente una MdT non deterministica è un vettore $M = \langle Q, \Sigma, \Gamma, q_0, B, \delta, F \rangle$ dove Q , Σ , Γ , q_0 , B e F sono definite come nel caso precedente, mentre δ è una funzione

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{-1, +1\}}$$

La macchina M , trovandosi nello stato $q \in Q$ e leggendo il simbolo $a \in \Gamma$, può scegliere la mossa da compiere nell'insieme $\delta(q, a)$. Come nella sezione precedente, possiamo definire le configurazioni di M , le relazioni di transizione \vdash_M e \vdash_M^* , e le computazioni di M . È evidente che, in questo caso, per ogni configurazione $C \in \mathcal{C}_M$ possono esistere più configurazioni raggiungibili da C in un passo. Per questo motivo, su un dato input, la macchina M può eseguire computazioni diverse, una per ogni possibile sequenza di scelte compiute da M a ogni passo.

Diremo che un input $w \in \Sigma^*$ è accettato da M se esiste una computazione di M su input w che conduce la macchina in una configurazione accettante, ovvero se esiste $C \in \mathcal{C}_M$ tale che $C = \alpha q \beta$, $q \in F$ e $C_0(w) \vdash_M^* C$.

Viceversa, se tutte le computazioni di M su input x terminano in una configurazione non accettante, diciamo che M rifiuta x .

Se tutte le computazioni di M hanno termine su ogni possibile input diciamo che M risolve il problema di decisione $\pi = \langle \Sigma^*, q \rangle$ dove, per ogni $x \in \Sigma^*$,

$$q(x) = \begin{cases} 1 & \text{se } M \text{ accetta } x \\ 0 & \text{altrimenti} \end{cases}$$

Data una MdT non deterministica $M = \langle Q, \Sigma, \Gamma, q_0, B, \delta, F \rangle$ e una stringa $w \in \Sigma^*$, denotiamo con $T_M(w)$ il massimo numero di mosse che la macchina può compiere in una computazione su input w . In altre parole $T_M(w)$ rappresenta l'altezza dell'albero di computazione di M su input w . Chiaramente, $T_M(w) = +\infty$ se e solo se esiste una computazione di M su tale input che non si arresta.

Inoltre, per ogni $n \in \mathbf{N}$, denotiamo con $T_M(n)$ il massimo valore di $T_M(w)$ al variare delle parole $w \in \Sigma^*$ di lunghezza n :

$$T_M(n) = \max\{T_M(w) \mid w \in \Sigma^*, |w| = n\}$$

Di nuovo, data una funzione $f : \mathbf{N} \longrightarrow \mathbf{R}^+$, diremo che una MdT non deterministica M lavora in tempo $f(n)$ se $T_M(n) \leq f(n)$ per ogni $n \in \mathbf{N}$.

Anche per le MdT non deterministiche valgono le proprietà di simulazione del corrispondente modello RAM. Questo consente di enunciare la seguente

Proposizione 12 *Un problema di decisione π appartiene a NP se e solo se esiste un polinomio p e una MdT M non deterministica che risolve π in un tempo $f(n)$ tale che $f(n) \leq p(n)$ per ogni $n \in \mathbf{N}$.*

7.2 Dimostrazione

In questa sezione presentiamo la dimostrazione del teorema 9. Nella sezione 5 abbiamo già dimostrato che SODD-FNC appartiene a NP. Dobbiamo quindi provare che ogni problema $\pi \in \text{NP}$ è polinomialmente riducibile a SODD-FNC. In altre parole vogliamo dimostrare che per ogni MdT non deterministica M , che lavora in tempo polinomiale, esiste una funzione f , calcolabile in tempo polinomiale, che associa a ogni stringa di input w di M una formula booleana ϕ , in forma normale congiunta, tale che w è accettata dalla macchina se e solo se esiste un assegnamento di valori alle variabili che rende vera ϕ .

Senza perdita di generalità, possiamo supporre che, per un opportuno polinomio p e per ogni input w di M , tutte le computazione della macchina su w abbiano la stessa lunghezza $p(|w|)$. Infatti, se M non soddisfa quest'ultima condizione, possiamo sempre costruire una nuova macchina che, su input w , prima calcola $p(|w|)$, poi simula M su w tenendo un contatore del numero di mosse eseguite e prolungando ogni computazione fino a $p(|w|)$ passi.

Supponiamo inoltre che la macchina M sia definita da $M = \langle Q, \Sigma, \Gamma, q_1, B, \delta, F \rangle$, dove $Q = \{q_1, q_2, \dots, q_s\}$ e $\Gamma = \{a_1, a_2, \dots, a_r\}$. Data ora una stringa $w \in \Sigma^*$ di lunghezza n , sappiamo che ogni computazione di M su w è una sequenza di $p(n) + 1$ configurazioni, ciascuna delle quali è rappresentabile da una stringa $\alpha q \beta$ tale che $|\alpha \beta| \leq p(n) + 1$. La corrispondente formula $\phi = f(w)$ sarà definita su tre tipi di variabili booleane: $S(u, t)$, $C(i, j, t)$ e $L(i, t)$, dove gli indici i, j, t, u variano in modo opportuno. Il significato intuitivo di queste variabili è il seguente:

- la variabile $S(u, t)$ assumerà il valore 1 se al tempo t la macchina si trova nello stato q_u ;
- la variabile $C(i, j, t)$ assumerà valore 1 se al tempo t nella cella i -esima si trova il simbolo a_j ;
- la variabile $L(i, t)$ assumerà il valore 1 se al tempo t la testina di lettura è posizionata sulla cella i -esima.

È chiaro che $u \in \{1, 2, \dots, s\}$, $t \in \{0, 1, \dots, p(n)\}$, $i \in \{1, 2, \dots, p(n) + 1\}$ e $j \in \{1, 2, \dots, r\}$. Un assegnamento di valori 0 e 1 a queste variabili rappresenta una computazione accettata di M su input w se le seguenti condizioni sono soddisfatte:

1. per ogni t esiste un solo u tale che $S(u, t) = 1$ ovvero, in ogni istante la macchina si può trovare in un solo stato;
2. per ogni t esiste un solo i tale che $L(i, t) = 1$ ovvero, in ogni istante la macchina legge esattamente una cella;

3. per ogni t e ogni i esiste un solo j tale che $C(i, j, t) = 1$ ovvero, in ogni istante ciascuna cella contiene esattamente un simbolo;
4. i valori delle variabili che hanno indice $t = 0$ rappresentano la configurazione iniziale su input w ;
5. esiste una variabile $S(u, p(n))$, che assume valore 1, tale che $q_u \in F$, ovvero M raggiunge uno stato finale al termine della computazione;
6. per ogni t e ogni i , se $L(i, t) = 0$, allora le variabili $C(i, j, t)$ e $C(i, j, t + 1)$ assumono lo stesso valore. In altre parole il contenuto delle celle che non sono lette dalla macchina in un dato istante, resta invariato all'istante successivo;
7. se invece $L(i, t) = 1$, allora il valore delle variabili $C(i, j, t + 1)$, $S(u, t + 1)$ e $L(i, t + 1)$ rispettano le mosse della macchina all'istante t .

Associamo ora a ciascuna condizione una formula booleana, definita sulle variabili date, in modo tale che ogni assegnamento soddisfi la condizione se e solo se rende vera la formula associata. A tale scopo utilizziamo l'espressione $U(y_1, y_2, \dots, y_k)$ definita in (2) che assume valore 1 se e solo se una sola delle sue variabili ha valore 1. La sua lunghezza è limitata da un polinomio nel numero delle variabili (in particolare $|U(y_1, y_2, \dots, y_k)| = O(k^2)$).

1. La prima condizione richiede che per ogni t vi sia un solo u tale che $S(u, t) = 1$. Possiamo allora scrivere la seguente formula

$$A = \prod_{t=0}^{p(n)} U(S(1, t), S(2, t), \dots, S(s, t))$$

la cui lunghezza è chiaramente $O(np(n))$, perché s è una costante che dipende solo da M e non dalla dimensione dell'input. È chiaro che un assegnamento rende vera la formula A se e solo se soddisfa la condizione 1.

Le altre formule si ottengono in modo simile e hanno tutte lunghezza polinomiale in n .

2. $B = \prod_{t=0}^{p(n)} U(L(1, t), L(2, t), \dots, L(p(n) + 1, t))$

3. $C = \prod_{t,i} U(C(i, 1, t), C(i, 2, t), \dots, C(i, r, t))$

4. Supponendo che $w = x_1 x_2 \dots x_n$ e rappresentando impropriamente l'indice di ogni x_i ,

$$D = S(1, 0) \cdot L(1, 0) \cdot \prod_{i=1}^n C(i, x_i, 0) \cdot \prod_{i=n+1}^{p(n)+1} C(i, B, 0)$$

5. $E = \sum_{q_u \in F} S(u, p(n))$

6. Per rappresentare la sesta condizione denotiamo con $x \equiv y$ l'espressione $(x + \bar{y}) \cdot (\bar{x} + y)$, che vale 1 se e solo se le variabili x e y assumono lo stesso valore. Per ogni i e ogni t ($t \neq p(n)$), poniamo

$$F_{it} = L(i, t) + \prod_{j=1}^r (C(i, j, t) \equiv C(i, j, t + 1)).$$

Quindi definiamo

$$F = \prod_{t=0}^{p(n)-1} \prod_{i=1}^{p(n)+1} F_{it}$$

7. Per ogni u, t, i, j definiamo

$$G_{utij} = \overline{S(u, t)} + \overline{L(i, t)} + \overline{C(i, j, t)} + \\ + \sum_{(q_{u'}, a_{j'}, v) \in \delta(q_u, a_j)} (S(u', t + 1) \cdot C(i, j', t + 1) \cdot L(i + v, t + 1)).$$

Osserviamo che questa formula non è in forma normale congiunta. Tuttavia, per quanto dimostrato nella sezione 5, sappiamo che esiste una formula equivalente in FNC che denoteremo con \tilde{G}_{utij} . Si può verificare che la lunghezza di \tilde{G}_{utij} è polinomiale. Ne segue che la formula associata alla settima condizione è

$$G = \prod_{u, t, i, j} \tilde{G}_{utij}$$

e anche la sua lunghezza risulta polinomiale in n .

Poiché l'espressione U è in forma normale congiunta, tutte le formule precedenti sono in FNC. Di conseguenza, anche la formula ϕ , ottenuta dal prodotto booleano delle formule precedenti, è in FNC:

$$\phi = A \cdot B \cdot C \cdot D \cdot E \cdot F \cdot G.$$

Tale formula seleziona esattamente gli assegnamenti che rappresentano computazioni accettanti di M su input w . Possiamo allora affermare che esiste un assegnamento che rende vera la formula ϕ se e solo la macchina M accetta l'input w .

È inoltre facile verificare, per una M fissata, ϕ può essere costruita in tempo polinomiale a partire dall'input w .