Automi e Linguaggi Formali
Università degli Studi di Padova
Maggio 2024

Homework 9 - Problemi trattabili e problemi intrattabili

Gabriel Rovesti

1. Sia $MODEXP = \{\langle a, b, c, p\rangle \mid a, b, c, p$ sono numeri interi binari positivi tali che $a^b = c(modp)\}$.
   Si dimostri che il problema $MODEXP \in P$.

   Soluzione

   Per dimostrare che $MODEXP \in P$, presentiamo un algoritmo polinomiale che risolve il problema.

   Algoritmo:

```
function modexp(a, b, c, p):
    result = 1
    a = a mod p
    while b > 0:
        if b mod 2 == 1:
            result = (result * a) mod p
        a = (a * a) mod p
        b = b // 2
    return result == c mod p
```

   L'algoritmo utilizza il metodo dell'esponenziazione rapida (fast exponentiation) per calcolare $a^b \pmod p$ in tempo logaritmico rispetto a b. L'idea principale è quella di scomporre b in forma binaria e di calcolare a^b attraverso una serie di quadrati e moltiplicazioni.

   Ad ogni iterazione del ciclo while, se il bit meno significativo di b è 1, moltiplichiamo *result* per a modulo p. Poi, in ogni caso, eleviamo al quadrato a modulo p e dividiamo b per 2 (scartando il resto). Questo processo continua fino a quando b diventa 0.

   Alla fine, l'algoritmo restituisce $True$ se *result* è congruente a c modulo p, altrimenti restituisce $False$. Un *certificato* per $MODEXP$ è semplicemente un intero c tale che $a^b \equiv c\ modp$. Il verificatore prende

in input la quadrupla $\langle a, b, c, p\rangle$ e verifica se $a^b \equiv c \pmod{p}$ in tempo polinomiale.

Il *verificatore* utilizza lo stesso algoritmo di esponenziazione rapida presentato nella dimostrazione che $MODEXP \in P$. Calcola $a^b \pmod{p}$ e controlla se il risultato è congruente a c modulo p. Se la congruenza è verificata, il verificatore accetta la quadrupla, altrimenti la rifiuta.

Poiché abbiamo presentato un algoritmo polinomiale che risolve $MODEXP$, abbiamo dimostrato che $MODEXP \in P$.

2. Sia G un grafo indiretto. Si consideri $SPATH = \{\langle G, a, b, k\rangle \mid G$ contiene un percorso (path) semplice tra a e b di lunghezza al più $k\}$. Si dimostri che $SPATH \in P$.

   Soluzione

   L'algoritmo di marcatura per il riconoscimento del $PATH$ può essere modificato per tenere traccia della la lunghezza dei percorsi più brevi scoperti. Di seguito la descrizione.
   "Sull'input $\langle G, a, b, k\rangle$ dove un grafo a m nodi G ha i nodi a, b:

   (a) Posiziona un mark 0 sul nodo a

   (b) Per ogni i da 0 ad m: Se un arco (s, t) viene trovato connettendo s nodi marcati "i" ad un nodo non marcato t, marca il nodo t con "$i + 1$"

   (c) Se b viene marcato con un valore al più k, *accetta*, altrimenti *rifiuta*.

3. In entrambe le parti, fornisci un'analisi della complessità temporale del tuo algoritmo.

   a. Dimostra che EQDFA ∈ P - ricordiamo che EQDFA è il problema di decidere se due automi a stati finiti deterministici (DFA) sono equivalenti, ovvero se accettano lo stesso linguaggio.

   b. Si dice che un linguaggio A è star-closed se A = A*. Fornisci un algoritmo in tempo polinomiale per verificare se un DFA riconosce un linguaggio star-closed. (Nota che EQNFA non è noto essere in P.)

   Soluzione

   Parte a: Per dimostrare che EQDFA ∈ P, forniremo un algoritmo in tempo polinomiale per determinare se due DFA sono equivalenti.

Algoritmo:

(a) Sia $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ i due DFA da confrontare.

(b) Costruisci il DFA $M = (Q, \Sigma, \delta, q_0, F)$ come segue:
- $Q = Q_1 \times Q_2$
- $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ per ogni $(p, q) \in Q$ e $a \in \Sigma$
- $q_0 = (q_1, q_2)$
- $F = \{(p, q) \in Q \mid p \in F_1 \Leftrightarrow q \in F_2\}$

(c) Esegui una ricerca in ampiezza (BFS) a partire da q_0 in M.

(d) Se durante la BFS viene raggiunto uno stato in $Q \setminus F$, restituisci "non equivalente". Altrimenti, restituisci "equivalente".

Analisi della complessità temporale:

- La costruzione di M richiede $O(|Q_1|\times|Q_2|\times|\Sigma|)$ operazioni, poiché per ogni coppia di stati (p, q) e per ogni simbolo $a \in \Sigma$, dobbiamo calcolare la funzione di transizione δ.
- La BFS richiede $O(|Q_1| \times |Q_2| + |Q_1| \times |Q_2| \times |\Sigma|)$ operazioni, poiché visita ogni stato al più una volta e segue ogni transizione al più una volta.

Pertanto, la complessità temporale complessiva dell'algoritmo è $O(|Q_1|\times |Q_2| \times |\Sigma|)$, che è polinomiale rispetto alle dimensioni degli input M_1 e M_2. Quindi, EQDFA $\in$ P.

Parte b: Algoritmo per verificare se un DFA riconosce un linguaggio star-closed:

(a) Sia $M = (Q, \Sigma, \delta, q_0, F)$ il DFA da verificare.

(b) Verifica se $\varepsilon \in L(M)$, cioè se $q_0 \in F$. Se non lo è, restituisci "non star-closed".

(c) Per ogni stato $q \in Q$, esegui una ricerca in profondità (DFS) a partire da q.

(d) Durante la DFS, se viene raggiunto uno stato accettante $f \in F$ e f è raggiungibile da se stesso attraverso un percorso non vuoto, restituisci "star-closed". Altrimenti, continua la DFS.

(e) Se tutte le DFS sono state completate senza restituire "star-closed", restituisci "non star-closed".

Analisi della complessità temporale:

- Il controllo se $\varepsilon \in L(M)$ richiede un tempo costante $O(1)$.
- Per ogni stato $q \in Q$, la DFS richiede $O(|Q|+|Q|\times|\Sigma|)$ operazioni, poiché visita ogni stato al più una volta e segue ogni transizione al più una volta.
- Poiché eseguiamo una DFS per ogni stato $q \in Q$, la complessità temporale complessiva è $O(|Q|\times(|Q|+|Q|\times|\Sigma|))$, che si semplifica in $O(|Q|^2 \times |\Sigma|)$.

Pertanto, l'algoritmo ha una complessità temporale polinomiale rispetto alla dimensione dell'input M, dimostrando che il problema di determinare se un DFA riconosce un linguaggio star-closed può essere risolto in tempo polinomiale.

4. Sia $CONNECTED = \{\langle G\rangle \mid$ Il grafo indiretto G è connesso $\}$. Si dimostri che $CONNECTED \in P$

   Soluzione

   (a) Dato un grafo indiretto $G = (V, E)$, dove V è l'insieme dei nodi e E è l'insieme degli archi:

   (b) Scegli un nodo di partenza $s \in V$

   (c) Marca s come visitato

   (d) Esplora ricorsivamente tutti i nodi raggiungibili da s, marcandoli come visitati

   (e) Se rimangono nodi non visitati, il grafo non è connesso

   (f) Altrimenti, il grafo è connesso

   La complessità temporale di questo algoritmo è $O(|V| + |E|)$, che è polinomiale rispetto alla dimensione dell'input (il numero di nodi e archi del grafo). Infatti, nel caso peggiore, l'algoritmo visita ogni nodo e ogni arco una sola volta.

   Poiché esiste un algoritmo che risolve il problema CONNECTED in tempo polinomiale, possiamo concludere che CONNECTED $\in P$.