Algoritmi e Principi dell'Informatica

Soluzioni al Tema d'esame
26 Aprile 2020

1 Informatica teorica

Esercizio 1 (prima variante)
Si consideri l'alfabeto $A = \{d, s, t\}$ e il linguaggio L di tutte e sole le stringhe su A che soddisfano i requisiti seguenti:

1. ogni stringa inizia con un prefisso che può essere la stringa vuota, la stringa d o la stringa ds;
2. dopo il prefisso, c'è la sequenza tds ripetuta un numero arbitrario di volte (anche nessuna volta);
3. infine, la stringa si chiude con la sequenza ts;
4. la sequenza dtd non appare mai all'interno della stringa;
5. la stringa contiene tutti e tre i simboli alfabetici.

Esempi di stringhe appartenenti al linguaggio sono: dts, dsts, tdsts, dstdsts, tdstdsts. Esempi di stringhe non appartenenti al linguaggio sono: ddts, dtdsts, ts, d.

Scrivere una grammatica a potenza generativa minima che generi L.

Soluzione

Basta una grammatica regolare. Il vincolo 5 impone che, in caso di prefisso vuoto, la sequenza tds sia ripetuta almeno una volta. Il vincolo 4 significa che, in caso di prefisso d, la sequenza tds sia ripetuta zero volte. L'espressione regolare corrispondente a L è la seguente: $dts|((tds|ds)(tds)^*ts)$
Una grammatica regolare è come segue:

$S \to A|B|C$
$A \to tB$
$B \to dD$
$D \to sA|sL$
$C \to dL$
$L \to tM$
$M \to s$

Un modo equivalente per risolvere l'esercizio è di iniziare con una grammatica non regolare che genera il linguaggio:
$S \to dts|tdsR|dsR$
$R \to tdsR|ts$
e di trasformarla in una grammatica regolare evitando le sequenze di terminali:

$S \to tS1|dS6$
$S1 \to dS2$
$S2 \to sS3$
$S3 \to tS4|tS7$
$S4 \to dS5$
$S5 \to sS3$
$S6 \to tS3|tS7$
$S7 \to s$

Esercizio 1 (seconda variante)

Testo e soluzione sono come nella prima variante, dopo aver opportunamente sostituito t con f, s con q e d con r.

Esercizio 2 (prima variante)

Per ciascuno dei seguenti insiemi si stabilisca, fornendo un'opportuna motivazione, se è ricorsivo.

- $S_1 = \{i \mid i$ è un numero primo$\}$
- $S_2 = \{i \mid i$ è un numero primo minore di100$\}$
- $S_3 = \{i \mid i$ è l'indice di una macchina di Turing che stabilisce se il suo ingresso è un numero primo$\}$
- $S_4 = \{i \mid i$ è un indice, minore di 100, di una macchina di Turing che stabilisce se il suo ingresso è un numero primo$\}$
- $S_5 = \{i \mid i$ è l'indice di una macchina di Turing con un numero dispari di stati$\}$
- $S_6 = \{i \mid i$ è un indice, minore di 100, di una macchina di Turing con un numero dispari di stati$\}$

Soluzione
S_2, S_4 e S_6 sono ricorsivi in quanto finiti.

S_1 è ricorsivo perché la sua funzione caratteristica è computabile (deve stabilire se un numero è primo, il che è decidibile).

S_3 non è ricorsivo per il teorema di Rice (all'insieme $F = \{f\}$, dove f è la funzione che stabilisce se il suo argomento è primo o meno, si applicano le condizioni del teorema).

S_5 è ricorsivo. Si noti che avere un numero dispari di stati non è una proprietà della funzione calcolata dalla macchina, quindi non si può applicare il teorema di Rice. Basta ispezionare il diagramma degli stati ricavato in questo modo per stabilire se il numero di stati è pari o dispari.

Esercizio 2 (seconda variante)

Per ciascuno dei seguenti insiemi si stabilisca, fornendo un'opportuna motivazione, se è ricorsivo.

- $S_1 = \{i \mid i$ è un numero pari scrivibile con 10 bit in notazione binaria$\}$
- $S_2 = \{i \mid i$ è un numero pari$\}$
- $S_3 = \{i \mid i$ è un indice, scrivibile con 10 bit in notazione binaria, di una macchina di Turing con

un numero primo di stati}

- $S_4 = \{i \mid i$ è l'indice di una macchina di Turing con un numero primo di stati$\}$
- $S_5 = \{i \mid i$ è un indice, scrivibile con 10 bit in notazione binaria, di una macchina di Turing che stabilisce se il suo ingresso è un numero pari$\}$
- $S_6 = \{i \mid i$ è l'indice di una macchina di Turing che stabilisce se il suo ingresso è un numero pari$\}$

Soluzione
S_1, S_3 e S_5 sono ricorsivi in quanto finiti.

S_2 è ricorsivo perché la sua funzione caratteristica è computabile (deve stabilire se un numero è pari, il che è decidibile).

S_4 è ricorsivo. Si noti che avere un numero primo di stati non è una proprietà della funzione calcolata dalla macchina, quindi non si può applicare il teorema di Rice. Basta usare l'enumerazione algoritmica delle macchine di Turing e ispezionare il diagramma degli stati ricavato in questo modo per stabilire se il numero di stati è primo o meno.

S_6 non è ricorsivo per il teorema di Rice (all'insieme $F = \{f\}$, dove f è la funzione che stabilisce se il suo argomento è pari o meno, si applicano le condizioni del teorema).

2 Algoritmi e strutture dati

Esercizio 1 (prima variante)
Calcolare i limiti di complessità asintotica per le seguenti ricorrenze:

1. $T(n) = 16T(\frac{n}{2}) + \Theta(n^2)$
2. $T(n) = 8T(\frac{n}{3}) + \Theta(n^3)$
3. $T(n) = nT(n-3) + \Theta(1)$

Soluzione

1. Si nota che la ricorrenza rispetta le ipotesi del Master theorem; $\log_b(a) = \log_2(16) = 4$, si ha quindi che $n^{4-\varepsilon} = n^2$ per $\varepsilon = 2$ (Master theorem, caso 1, nessuna ipotesi aggiuntiva). La ricorrenza è quindi $\Theta(n^4)$.
2. Come sopra, la ricorrenza rispetta le ipotesi del Master theorem. $\log_b(a) = \log_3(8) \approx 1,893$, con $f(n) = n^3$, ci troviamo nel terzo caso. La condizione di regolarità è automaticamente rispettata poiché nella ricorrenza il termine $f(n)$ è del tipo $\Theta(n^k)$ (qui $k = 3$). La ricorrenza è quindi $\Theta(n^3)$.
3. Con l'albero di ricorsione si arriva facilmente a osservare $T(n) = \mathcal{O}(n!)$, che poi si può mostrare per induzione. Un limite più stretto si può ottenere espandendo la ricorrenza come

segue:

$$\begin{aligned}
T(n) &= nT(n-3) + \Theta(1) \\
&= n((n-3)T(n-6) + \Theta(1)) + \Theta(1) \\
&= (n^2 - 3n)T(n-6) + \Theta(n) + \Theta(1) \\
&= (n^2 - 3n)((n-6)T(n-9) + \Theta(1)) + \Theta(n) + \Theta(1) \\
&= (n^3 - 9n^2 + 18n)T(n-9) + \Theta(n^2) + \Theta(n) + \Theta(1)
\end{aligned}$$

La ricorsione è profonda $n/3$ passi e a ogni passo si aumenta di 1 il grado del polinomio. La complessità finale è $O(n^{n/3})$.

Esercizio 1 (seconda variante)
Calcolare i limiti di complessità asintotica per le seguenti ricorrenze:

1. $T(n) = 81T(\frac{n}{3}) + \Theta(n^3)$
2. $T(n) = 6T(\frac{n}{2}) + \Theta(n^3)$
3. $T(n) = nT(n-4) + \Theta(1)$

Soluzione

1. Si nota che la ricorrenza rispetta le ipotesi del Master theorem; $\log_b(a) = \log_3(81) = 4$, si ha quindi che $n^{4-\varepsilon} = n^3$ per $\varepsilon = 1$ (Master theorem, caso 1, nessuna ipotesi aggiuntiva). La ricorrenza è quindi $\Theta(n^4)$.

2. Come sopra, la ricorrenza rispetta le ipotesi del Master theorem. $\log_b(a) = \log_2(6) \approx 2,58$, con $f(n) = n^3$, ci troviamo nel terzo caso. La condizione di regolarità è automaticamente rispettata poiché nella ricorrenza il termine $f(n)$ è del tipo $\Theta(n^k)$ (qui $k = 3$). La ricorrenza è quindi $\Theta(n^3)$.

3. Con l'albero di ricorsione si arriva facilmente a osservare $T(n) = \mathcal{O}(n!)$, che poi si può mostrare per induzione. Un limite più stretto si può ottenere espandendo la ricorrenza come segue:

$$\begin{aligned}
T(n) &= nT(n-4) + \Theta(1) \\
&= n((n-4)T(n-8) + \Theta(1)) + \Theta(1) \\
&= (n^2 - 4n)T(n-8) + \Theta(n) + \Theta(1) \\
&= (n^2 - 4n)((n-8)T(n-12) + \Theta(1)) + \Theta(n) + \Theta(1) \\
&= (n^3 - 12n^2 + 32n)T(n-12) + \Theta(n^2) + \Theta(n) + \Theta(1)
\end{aligned}$$

La ricorsione è profonda $n/4$ passi e a ogni passo si aumenta di 1 il grado del polinomio. La complessità finale è $O(n^{n/4})$.

Esercizio 2 (prima variante)
Si consideri una matrice quadrata di numeri interi positivi, avente lato n. La matrice contiene numeri interi ordinati in ordine crescente lungo ogni riga e ordinati in ordine decrescente lungo ogni colonna colonna.

Si descriva un algoritmo di ricerca di un valore intero dato all'interno della matrice stessa, avente minime complessità temporale e spaziale.

Soluzione

E' possibile individuare il valore cercato, o determinare l' assenza, in $\mathcal{O}(n)$ tempo e $\mathcal{O}(1)$ spazio. La soluzione si basa sul fatto che che la matrice, ordinata in questo modo, offra la possibilità di effettuare una ricerca considerando che tutti gli elementi in una riga successiva siano minori del corrente, e tutti quelli in una colonna successiva maggiori di esso.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define N 3
4
5  int main(int argc, char* argv[]){
6
7      int M[N][N]= {{7,8,15},
8                    {5,6,12},
9                    {1,4,9}};
10
11     int riga=0,col=0;
12     int to_find;
13     to_find = atoi(argv[1]);
14     while ((riga < N ) && (col < N)){
15         if(M[riga][col] == to_find) {
16             printf("found␣at␣(%d,%d)\n",riga,col);
17             return 0;
18         }
19         if ( to_find > M[riga][col] ) {
20              col++;
21         } else {
22             riga++;
23         }
24     }
25     printf("Not␣found\n");
26     return 1;
27 }
```

Esercizio 2 (seconda variante)
Si consideri una matrice quadrata di numeri interi positivi, avente lato n. La matrice contiene numeri interi ordinati in ordine decrescente lungo ogni riga e ordinati in ordine crescente lungo ogni colonna colonna.

Si descriva un algoritmo di ricerca di un valore intero dato all'interno della matrice stessa, avente minime complessità temporale e spaziale.

Soluzione

E' possibile individuare il valore cercato, o determinare l' assenza, in $\mathcal{O}(n)$ tempo e $\mathcal{O}(1)$ spazio. La soluzione si basa sul fatto che che la matrice, ordinata in questo modo, offra la possibilità di effettuare una ricerca considerando che tutti gli elementi in una riga successiva siano minori del corrente, e tutti quelli in una colonna successiva maggiori di esso.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define N 3
4
5  int main(int argc, char* argv[]){
6
7     int M[N][N]= {{7,8,15},
8                   {5,6,12},
9                   {1,4,9}};
10
11    int riga=0,col=0;
12    int to_find;
13    to_find = atoi(argv[1]);
14    while ((riga < N ) && (col < N)){
15        if(M[riga][col] == to_find) {
16            printf("found␣at␣(%d,%d)\n",riga,col);
17            return 0;
18        }
19        if ( to_find > M[riga][col] ) {
20             riga++;
21        } else {
22            col++;
23        }
24    }
25    printf("Not␣found\n");
26    return 1;
27 }
```