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.

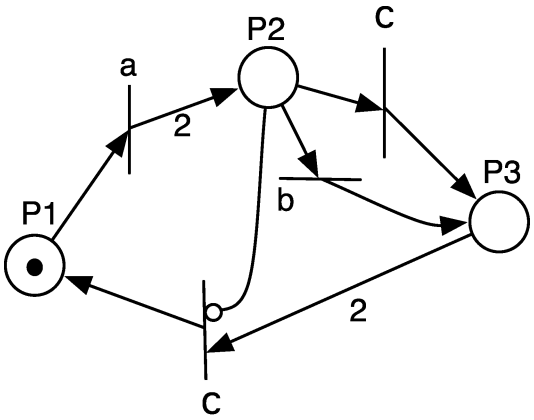
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define N 3
4
5  int main(int argc, char* argv[]){
6
7     int M[N][N]= {{7,8,15},
8                   {5,6,12},
9                   {1,4,9}};
10
11    int riga=0,col=0;
12    int to_find;
13    to_find = atoi(argv[1]);
14    while ((riga < N ) && (col < 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 }
```

Algoritmi e Principi dell'Informatica
Parte I – Modelli e computabilità
Appello d'esame - 25 Febbraio 2011

Tempo a disposizione: 1h30

NB: Motivare sempre adeguatamente le proprie risposte

Esercizio 1 (11 punti)
Sia data la rete di Petri in figura, con F = {M_F}, $M_F(P_1) = 1$, $M_F(P_2) = 0$, $M_F(P_3) = 0$.



1. Scrivere una grammatica che generi lo stesso linguaggio L della rete di Petri. La grammatica deve avere il minor numero di simboli **nonterminali** possibile.
2. Il tipo della grammatica scritta è a potenza generativa minima tra quelle in grado di generare il linguaggio L in oggetto?
3. La rete di Petri è a potenza minima tra quelle in grado di generare L? Ossia, vi sono tipi di e reti di Petri meno potenti del genere di quella scritta in grado di generare L?

Esercizio 2 (12 punti + Bonus)
Si indichi, come al solito, con f_y la funzione calcolata dalla y-esima Macchina di Turing.

1. E' decidibile il problema di stabilire se f_{10} è definita per ogni $x \leq 10$
   (cioè se $\forall x(x \leq 10 \rightarrow f_{10}(x) \neq \perp)$)?
2. E' decidibile il problema di stabilire se, per y generico, f_y è definita per ogni $x \leq 10$?
3. E' semidecidibile il problema del punto 2?
4. E' decidibile il problema di stabilire se, per y generico, f_y è definita per ogni $x > 10$?
5. **Bonus:** E' semidecidibile il problema del punto 4?

Esercizio 3 (7 punti + Bonus)
Si consideri la grammatica G_1 con le seguenti produzioni:

$$T \rightarrow aTbT \mid bTaT \mid \varepsilon$$

e simbolo iniziale T . G_1 genera tutte e sole le stringhe in cui le *a* e le *b* sono in ugual numero.
Si consideri ora la grammatica G_2 con le seguenti produzioni:

$$S \rightarrow TaT$$
$$T \rightarrow aTbT \mid bTaT \mid \varepsilon$$

e simbolo iniziale S.
1. Qual è il linguaggio generato da G_2 ? Motivare la risposta.
2. **Bonus:** fornire una dimostrazione della risposta data al punto 1.

Algoritmi e Principi dell'Informatica

Parte II – Complessità, algoritmi, strutture dati

Appello d'esame - 25 Febbraio 2011

Tempo a disposizione: 1h30

Esercizio 1 (12 punti)

Si consideri il problema di riconoscere il linguaggio $L = \{ ww \mid w \in \{a,b\}^* \}$.

1. Si descriva in modo informale, ma sufficientemente preciso, una macchina di Turing deterministica a k nastri di memoria che riconosce il linguaggio L desiderato minimizzando la complessità *spaziale*. Si diano le complessità temporale e spaziale della macchina scritta.
2. Si descriva in modo informale, ma sufficientemente preciso, una macchina di Turing **non**deterministica a k nastri di memoria che riconosce il linguaggio L desiderato minimizzando la complessità *temporale*. Si diano le complessità temporale e spaziale della macchina scritta.

Esercizio 2 (14 punti)

1. Si scriva un algoritmo che, dato in ingresso un array A di numeri naturali (tutti diversi tra di loro) ed un valore x, restituisce *true* se nell'array esistono tre indici i, j, k diversi tra di loro tali che $A[i] < A[j] < A[k]$ e che la somma di $A[j] - A[i]$ con $A[k] - A[j]$ dia x, *false* altrimenti

Per esempio, data in ingresso la sequenza A = [8, 20, 27, 6, 4, 0, 17, 95, 1] ed il valore x = 10, l'algoritmo deve restituire *true*, in quanto presi come indici $i = 7$, $j = 2$, e $k = 3$ (quindi $A[7] = 17$, $A[2] = 20$, $A[3] = 27$) si ha che la somma di $A[2]$ - $A[7]$ (cioè 3) e di $A[3]$ - $A[2]$ (cioè 7) dà 10 come desiderato.

2. Si dia la complessità temporale dell'algoritmo scritto.

3. Supponendo di realizzare l'algoritmo descritto al punto 1 con una macchina RAM (della quale non è necessario scrivere il codice), si diano le complessità temporale e spaziale dell'algoritmo scritto calcolate a criterio di costo logaritmico. Si supponga che la sequenza di numeri non sia già in memoria, ma debba essere letta da standard input.

NB1: E' preferibile che l'algoritmo venga scritto in pseudocodice; tuttavia, verranno accettate anche soluzioni in cui l'algoritmo è descritto in modo informale, purché la descrizione sia sufficientemente precisa per poterne valutare la complessità.
NB2: Il punteggio assegnato sarà tanto più alto quanto migliore sarà la complessità dell'algoritmo proposto.

Esercizio 3 (8 punti)

Si vuole realizzare una applicazione in cui c'è da gestire un insieme di elementi per il quale si sa (o si suppone) che il numero di esecuzioni di operazioni di ricerca sia molto più alto delle esecuzioni delle operazioni di inserimento/cancellazione.
Per questo si vuole realizzare una tabella hash in cui la risoluzione dei conflitti viene fatta memorizzando le chiavi con stesso valore hash non più in una lista, ma in un albero binario di ricerca (BST).

1. Dire come cambiano, se cambiano, le complessità temporali nel caso **medio** delle operazioni di INSERT, DELETE, e SEARCH effettuate sulla tabella basata su BST rispetto alla soluzione basata su lista, sotto l'ipotesi di hashing uniforme semplice.

2. Date le caratteristiche della applicazione, possiamo dire che la scelta del progettista di modificare la struttura dati sia stata azzeccata oppure no? Motivare la risposta.

3. Cambierebbero le complessità del caso medio se, invece di usare dei BST, il progettista avesse usato degli alberi red-black?

Soluzioni – Parte I

Esercizio 1

1. Il linguaggio riconosciuto dalla rete di Petri è: L = (a(b|c)(b|c)c)*. Una grammatica G con il numero minimo di nonterminali (1) che generi L è la seguente:
   S → ε | abbcS | abccS | acbcS | acccS
2. La grammatica G è di tipo non contestuale. Il linguaggio L è regolare, pertanto G non è a potenza minima in quanto basterebbe una grammatica regolare per generare L.
3. La rete di Petri del testo fa uso di archi inibitori, pertanto non è a potenza minima, in quanto per riconoscere L, che è un linguaggio regolare, sarebbe senz'altro sufficiente una rete di Petri senza archi inibitori (addirittura ne basterebbe una 1-limitata).

Esercizio 2

1. Si tratta di una domanda con risposta chiusa sì/no che non dipende da alcun input, pertanto il problema è decidibile.
2. Non è decidibile per il teorema di Rice, in quanto l'insieme di funzioni descritto non è l'insieme vuoto, né è l'insieme universo.
3. E' semidecidibile, in quanto basta provare le computazioni da 0 a 10 di f_y per un numero crescente di passi, secondo l'usuale tecnica diagonale di simulazione. Se tutte e 11 le esecuzioni terminano, prima o poi lo scopriamo, da cui la semidecidibilità.
4. No, per il teorema di Rice, con ragionamento analogo al punto 2.
5. Non è nemmeno semidecidibile. Se fosse semidecidibile anche questo problema, unitamente al fatto che è semidecidibile stabilire se f_y sia definita per $x \leq 10$ (vedi punto 3), allora sarebbe semidecidibile anche il problema di stabilire se una generica funzione sia definita per ogni x, ovvero sia totale, il che è notoriamente falso.

Esercizio 3

1. Il linguaggio generato da G_2 è il linguaggio di tutte e sole le stringhe in cui il numero di *a* è pari al numero di *b* più uno, ossia $L(G_2) = \{x \mid x \in \{a,b\}^* \wedge \#a(x) = \#b(x) + 1\}$. Come affermato nel testo, le stringhe generate a partire da T sono quelle del linguaggio $L(G_1)$ di tutte e sole le stringhe in cui le *a* e le *b* sono in ugual numero. Le stringhe di $L(G_2)$, tramite la produzione S → TaT, sono generate dalla giustapposizione di una *a* tra due stringhe di L(G1). Poiché una qualunque stringa *x* tale che #a(x) = #b(x) + 1 è tale che esiste una *a* senza una corrispondente *b*, la grammatica G_2 genera *tutte* le stringhe con numero di *a* uguale al numero di *b* più 1.

2. Da quanto scritto sopra, le stringhe generate da G_2 hanno la forma $x \cdot a \cdot y$, dove $x,y \in L(G_1)$. E' immediato concludere che G_2 genera soltanto stringhe in cui vi sia una a in più rispetto alle *b*, poiché sia *x* sia *y* sono già bilanciate, appartenendo a $L(G_1)$. Per mostrare che G_2 genera tutte le stringhe in cui vi sia una *a* in più rispetto alle *b*, si consideri una generica stringa *w* tale per cui #a(w) = #b(w) + 1. Poiché *w* contiene più *a* che *b*, scandendo *w* da sinistra a destra, esiste necessariamente un carattere di *w* che è la prima *a* in soprannumero rispetto alle *b* incontrate (incluso il caso in cui non si siano incontrate *b* e cioè che la *a* sia il primo carattere di *w*). Sia x la sottostringa composta da tutti i caratteri a sinistra di quella *a*, e *y* quella composta da tutti i caratteri a destra della *a*. Per quanto scritto, la *a* che separa *x* da *y* è la prima *a* in soprannumero, quindi in *x* le *a* e le *b* sono in ugual numero. A questo punto segue necessariamente che anche nella *y* le *a* e le *b* sono in ugual numero, altrimenti la stringa $w = x \cdot a \cdot y$ non soddisferebbe più il vincolo #a(w) = #b(w) + 1.

Soluzioni – Parte II

Esercizio 1

1. Per riconoscere il linguaggio L con una MT deterministica minimizzando la complessità spaziale è sufficiente usare una MT a 2 nastri di memoria, che funzioni nel seguente modo:

1. prima legge tutti i simboli in input, mano a mano incrementando un contatore binario memorizzato nel nastro T1;
2. divide il valore del contatore eliminando lo 0 finale (si bocca in stato non finale se l'ultima cifra binaria non è uno 0);
3. copia il contenuto di T1 nel nastro T2, e quindi si porta sul primo carattere in input
4. memorizza il carattere corrente nello stato, e si porta avanti n/2 celle (con n = |ww| il valore memorizzato in T1), quindi controlla che la cella di arrivo contenga un carattere uguale a quello della cella di partenza;
5. copia il contatore da T1 a T2 e torna indietro n/2-1 celle (sfruttando di nuovo il contatore in T2), quindi di nuovo copia il contatore da T1 a T2 e ripete dal passo 4;
6. la MT accetta se arriva in fondo al nastro di input senza avere incontrato coppie di caratteri diverse.

La complessità spaziale di una tale MT è $\Theta(\log(n))$ (lo spazio occupato dai contatori), mentre quella temporale è $\Theta(n^2\log(n))$ (l'analisi di ognuno degli n/2 caratteri di w richiede di spostarsi n/2 celle in avanti, ed ogni spostamento richiede il decremento di un contatore di lunghezza log(n)).

2. Una MT nondeterministica a 2 nastri di memoria può semplicemente scandire il nastro in ingresso, copiando l'input sul nastro T1, fino a che, nondeterministicamente, decide di cominciare a copiare i caratteri in input sul nastro T2, fino alla fine dell'input. A quel punto deve scandire all'indietro i 2 nastri, e controllare che contengano gli stessi caratteri.

Le complessità temporale e spaziale di una simile macchina di Turing sono entrambe $\Theta(n)$

Esercizio 2

1. Un algoritmo che risolve il problema dato è il seguente:

```
EXACT-DIFF(A, x)
1  MERGE-SORT(A, 1, A.length)
2  i := 1
3  k := 3
4  while k ≤ A.length and i < A.length-1
5    if k = i+1
6       k := k+1
7    elseif A[k] - A[i] = x
8      return true
9    elseif A[k] - A[i] < x
10     k := k+1
11   else i := i+1
12 return false
```

2. La complessità temporale dell'algoritmo definite al punto 1 è determinata dall'ordinamento della riga 1, che ha complessità $\Theta(n \log(n))$. Infatti il ciclo 4-11 viene eseguito al più *n* volte in quanto nessuno degli indici *i*, *k* viene mai decrementato.

3. L'algoritmo realizzato con macchina RAM avrebbe complessità temporale, valutata a criterio di costo logaritmico, $\Theta(n \log^2(n))$ in quanto il MERGE-SORT richiede di accedere alle celle di un array di lunghezza n, ed il costo di tali accessi è $\Theta(\log(n))$ (consideriamo per semplicità i valori contenuti nell'array come delle costanti, altrimenti andrebbe aggiunto un fattore log(v), con v il valore massimo presente nell'array).
La complessità spaziale invece sarebbe $\Theta(n)$ in quanto occorrono *n* celle per memorizzare i valori dell'array A.

Esercizio 3

1. Nel caso medio, il numero di elementi in conflitto è pari a quella che sarebbe la lunghezza della lista, cioè α, con α fattore di carico; di conseguenza, in media il numero di elementi in ogni albero è anch'esso α. Supponendo che gli elementi arrivino in modo casuale, ogni albero ha in media altezza $\log(\alpha)$, quindi le operazioni di INSERT, DELETE e SEARCH hanno tutte complessità $O(\log(\alpha))$ nel caso medio.

2. La scelta è stata opportuna, in quanto l'operazione più usata, la ricerca, risulta essere più efficiente nel caso di alberi (la complessità è $O(\log(\alpha))$ invece che $O(1+ \alpha)$ come per le liste).

3. Nel caso di alberi red-black la complessità media non cambia, in quanto ancora il numero medio di elementi negli alberi sarebbe α, e la loro l'altezza sarebbe $\log(\alpha)$.

Algoritmi e Principi dell'Informatica

Appello del 21 Febbraio 2013

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 3 ore.

Chi deve sostenere solo il modulo di Informatica teorica deve svolgere gli Esercizi 1, 2, 3 in 1 ora e 30 minuti.

Chi deve sostenere solo il modulo di Informatica 3 deve svolgere gli Esercizi 4, 5, 6 e in 1 ora e 30 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Modulo I

Esercizio 1 (punti 5)

Si considerino i seguenti linguaggi

$L_1 = \left\{ \left(a^{i_j} b^{i_j}\right)^k d \mid k > 0, 1 \leq j \leq k, i_j > 0 \right\}$ = {abd, aabbabd, aabbaabbd, abaabbd, … abaaabbbabd, … aabbabaaabbbd, …}

$L_2 = \left\{ a^h \left(b^{i_j} c a^{i_j}\right)^k d \mid h > 0,\ k > 0, 1 \leq j \leq k, i_j > 0 \right\}$ = {abcad, … aaabbcaad, aaabbbcaaad, aabbbcaaabcad, …}

1. Definire una grammatica non contestuale che generi L_1 e un automa a pila deterministico che lo accetti.
2. Definire una grammatica non contestuale che generi L_2 e un automa a pila, preferibilmente deterministico, che lo accetti.
3. Definire una grammatica non contestuale che generi $L_1 \cup L_2$. Se possibile, definire un automa a pila deterministico che lo accetti; altrimenti argomentare in modo informale ma convincente che il linguaggio $L_1 \cup L_2$ non può essere accettato da un automa a pila deterministico, e definire un automa a pila nondeterministico che accetti $L_1 \cup L_2$.

Esercizio 2 (punti 5)

Si specifichi in logica del prim'ordine una funzione f : A* × N → A, dove A è un alfabeto di simboli e N è l'insieme dei numeri naturali, che, data una stringa costruita sull'alfabeto A e un numero i, restituisce il carattere in posizione (i+1)-esima nella stringa (cioè il carattere con indice i nella stringa, dove 0 è l'indice del primo carattere, 1 del secondo, e così via). Nella specifica, è consentito l'uso delle seguenti funzioni:

$s \cdot t$ (concatenazione delle stringhe s e t)

$|s|$ (lunghezza di una stringa s)

e del predicato = (uguaglianza). Ovviamente *non* è consentito l'uso della notazione $s[i]$, che indica l'i-esimo carattere della stringa s.

Esercizio 3 (punti 5)

Si dica, giustificando brevemente la risposta, se la seguente funzione è calcolabile o no:

$f : N \times N \times N \rightarrow \{0, 1\}$

$f(0, 0, 0) = 0;$

$f(x, y, z) =$

se z è pari allora

0 se x è la radice quadrata intera esatta di y,

$\perp$ altrimenti;

se z è dispari allora

1 se ($\forall$x dispari $\exists$y pari e k, h $\in$ N tali che $k^x = h^y$,

$\wedge$

$\forall$x pari $\exists$y dispari e k, h $\in$ N tali che $k^x = h^y$)

$\perp$ altrimenti.

Modulo II

Esercizio 4 (punti 6)

Si risolva la seguente equazione alle ricorrenze (a meno della relazione O):

$T(n) = 2T(n-4) + O(\log(n))$.

Esercizio 5 (punti 6)

Scrivere un algoritmo che, dato un array A, determina se esiste un sottoarray (per sottoarray si intende una sequenza di elementi *consecutivi* dell'array base) di A che sia a sua volta scomponibile in 2 sottoarray tali che la somma degli elementi nel primo sia uguale alla somma degli elementi del secondo. L'algoritmo deve restituire il sottoarray più lungo tra quelli che soddisfano la condizione di cui sopra, se esiste.

Indicare la complessità temporale asintotica dell'algoritmo scritto.

Esercizio 6 (punti 5)

Si definisca (senza necessariamente codificarne tutti i dettagli) un macchina di Turing a nastro singolo che riconosca il linguaggio delle stringhe del tipo x#y (x, y $\in$ {a, b}*) con

(1) $|x| \neq |y|$ e

(2) |x| pari e |y| dispari o viceversa,

minimizzando la complessità temporale della macchina; se ne valuti la complessità spaziale e temporale a meno della relazione Θ.

Tali complessità cambierebbero se si rimuovesse il vincolo (2)?

Cosa cambierebbe se si usasse una macchina a k nastri invece di una macchina a nastro singolo?

Tracce di soluzione

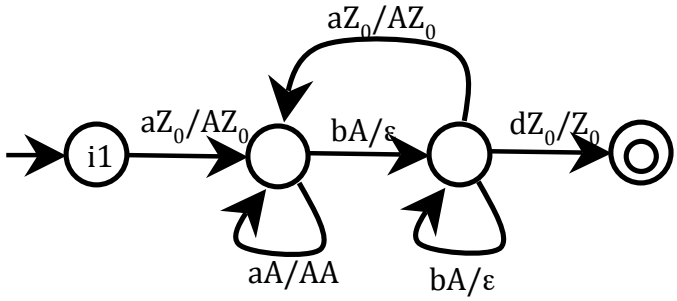
Esercizio 1

1. La grammatica seguente, con assioma S_1, genera L_1

$S_1 \rightarrow A\ S_1 \mid A\ d$

$A \rightarrow a\ A\ b \mid a\ b$

L'automa deterministico seguente accetta L_1.



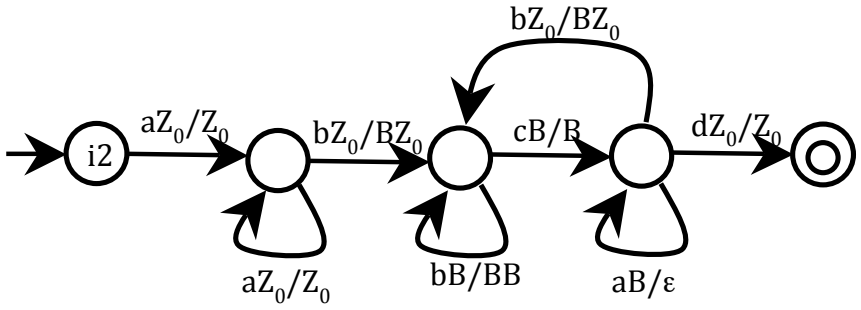
2. La grammatica seguente, con assioma S_2, genera L_2

$S_2 \rightarrow a\ S_2 \mid a\ X\ d$

$X \rightarrow C\ X \mid C$

$C \rightarrow b\ C\ a \mid b\ c\ a$

L'automa deterministico seguente accetta L_2.

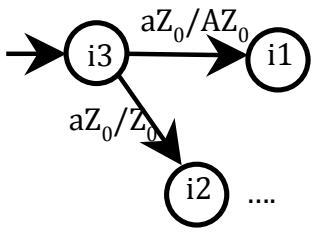


3. La grammatica, con assioma S, genera $L_1 \cup L_2$

$S \rightarrow S_1 \mid S_2$ … più le regole per i linguaggi L_1 ed L_2 …

L'automa a pila che accetta $L_1 \cup L_2$ non può essere deterministico: mentre legge il primo gruppo di simboli *a*, l'automa deve memorizzare nella pila un conteggio unario del numero di *a* e poi, mentre legge il primo gruppo di *b*, deve usare la pila per verificare (nel caso che la stringa $\in L_1$) che il numero delle *b*sia uguale al numero delle *a*. Ma facendo ciò il contenuto della pila va perso, quindi la pila non può essere usata per verificare (nel caso la stringa $\in L_2$) che il numero delle *a*che seguono la successiva*c*eguagli il numero delle *b*che la precedono.

Un automa nondeterministico che accetti $L_1 \cup L_2$ può essere ottenuto come semplice composizione dei due automi che accettano L_1ed L_2.



Esercizio 2

$\forall x\ \forall y\ \forall i\ (f(x, i) = y \leftrightarrow \exists w\ \exists z\ (x = w \cdot y \cdot z \wedge |y| = 1 \wedge |w| = i))$

Esercizio 3

f è calcolabile perché :

- è decidibile se z sia pari o dispari
- nel primo caso è decidibile stabilire se x sia la radice quadrata intera esatta di y o no e produrre 1 o, rispettivamente, $\perp$ nei due casi;
- nel secondo caso la domanda è chiusa (non importa stabilire se la condizione indicata sia vera o falsa: la funzione di fatto non dipende da x e y in quanto quantificate) e quindi una delle due macchine che per z dispari producono in output 1 oppure non terminano è la macchina che computa questa parte della funzione.

Esercizio 4

Mediante il metodo dell'albero di ricorsione si ottiene il guess $O(2^{n/4} \cdot \log(n))$: il costo di ogni livello è $O(2^k \log(n - 4k))$; quindi la profondità dell'albero è $k = n/4$.
Poi si verifica che, con opportune costanti, $T(n) \leq c \cdot 2^{n/4} \cdot \log(n)$.
In effetti, per arrivare a concludere che $T(n) \leq c \cdot 2^{n/4} \cdot \log(n)$, è utile mostrare un vincolo leggermente più stretto, e cioè che $T(n) \leq c \cdot 2^{n/4} \cdot \log(n) - bn^2$. Infatti si ha:

$$\begin{aligned} T(n) &= 2 \cdot T(n-4) + d \cdot \log(n) \leq 2c2^{((n-4)/4)} \log(n-4) - 2b(n-4)^2 + d \cdot \log(n) = \\ &= c2^{n/4} \log(n-4) - bn^2 - bn^2 + 16bn - 32 + d \cdot \log(n) \leq \\ &\leq (c2^{n/4} \log(n) - bn^2) - bn^2 + 16bn - 32 + d \cdot \log(n) \leq \\ &\leq c2^{n/4} \log(n) - bn^2 \end{aligned}$$

per b ed n sufficientemente grandi.

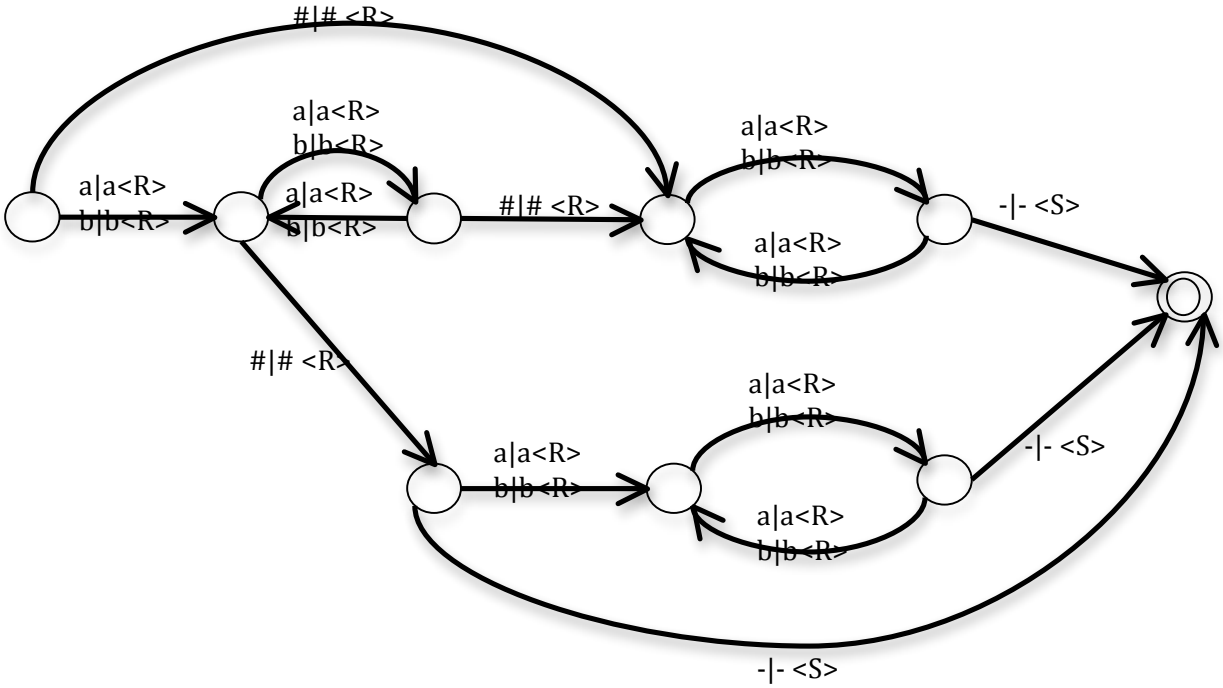
Esercizio 5

Un modo naturale per risolvere il problema consiste nell'elencare tutti gli n^2 sottoarray di A (lungo n), e controllare se questi hanno la proprietàdesiderata;tale verifica si può evidentemente fare in tempo O(n) per ogni sottoarray.

Di conseguenza, la complessità temporale complessiva è $\Theta(n^3)$.

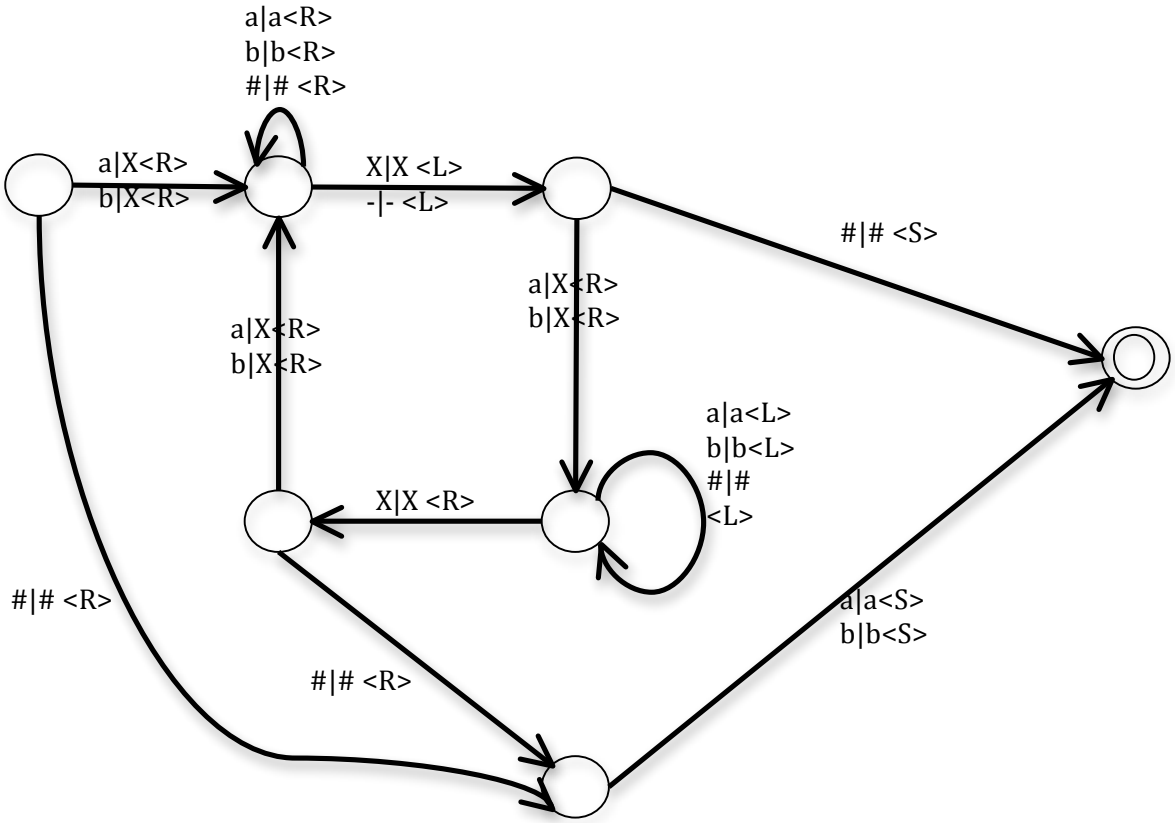
Esercizio 6

Una MT di Turing a nastro singolo che riconosce il linguaggio descritto opera con complessità temporale e spaziale Θ(n). Il linguaggio descritto è, infatti, un linguaggio regolare e il controllo sulla condizione (2) (che *implica la condizione* (1)) può essere fatto usando solo gli stati come nella macchina seguente.



La complessità spaziale è data dal fatto che l'unico nastro contiene la stringa di ingresso.

Se si rimuovesse il vincolo (2) il linguaggio non sarebbe più regolare e una macchina che minimizza la complessità temporale potrebbe operare nel seguente modo:



La complessità temporale diventerebbe quindi $\Theta(n^2)$, mentre quella spaziale rimarrebbe pari a $\Theta(n)$.

Se invece che una macchina a nastro singolo si usasse una macchina a k nastri le complessità cambierebbero nel seguente modo.

Nel primo caso la macchina a k nastri opererebbe esattamente come la macchina a nastro singolo, mantenendo quindi una complessità temporale $\Theta(n)$, mentre la complessità spaziale diventerebbe $\Theta(1)$, poiché i nastri di memoria e ingresso sono distinti e quindi la memoria in questo caso, non viene usata.

Rimossa la condizione (2), una macchina con un nastro di memoria può riconoscere il linguaggio scorrendo la prima parte della stringa e copiandola nel nastro di memoria. Una volta raggiunto il simbolo # legge la seconda parte della stringa, avvolgendo indietro il nastro in cui è stata copiato la prima parte della stringa. Se il nastro raggiunge Z_0 prima che la stringa sia finita o la stringa finisce prima di arrivare a Z_0 sul nastro, la parola viene accettata.

La complessità temporale diventa quindi $\Theta(n)$, così come quella spaziale.

Algoritmi e Principi dell'Informatica

Appello dell'11 Settembre 2013

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 3 ore.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1, 2 e 3, in 1 ora e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 4, 5, 6 in 1 ora e 30 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (punti 4)
Sia data la seguente grammatica G:
S → B S a | C
BC → C b b
C → c

1. Si dica quale è il linguaggio generato da G.
2. Si scriva un automa che riconosce il linguaggio generato da G. L'automa deve appartenere alla classe di potenza riconoscitiva minima tra quelle che riconoscono il linguaggio generato da G.

Esercizio 2 (punti 5)

Si consideri il seguente programma (in pseudocodice), il cui scopo è ordinare un generico array A di lunghezza variabile il cui valore è memorizzato nel campo *length* (si supponga che il campo *key* dei vari elementi sia un intero in valore assoluto ≤ 100):

```
INSERTION-SORT(A)
1 for j := 2 to A.length
2   key := A[j]
3   //Inserisce A[j] nella sequenza ordinata A[1..j−1]
4   i := j − 1
5   while i > 0 and A[i] > key
6     A[i + 1] := A[i]
7     i := i − 1
8   A[i + 1] := key
```

1. E' decidibile il problema di stabilire se il programma suddetto ordina correttamente un generico array di lunghezza ≤ 100?
2. E' decidibile il problema di stabilire se il programma suddetto ordina correttamente un generico array di lunghezza qualsiasi?

Esercizio 3

Prima parte (obbligatoria:punti 6)

Si consideri un sistema costituito da un processo che gestisce una risorsa R utilizzata da due processi utenti U1 e U2.

- Ognuno dei due utenti richiede di tanto in tanto l'accesso alla risorsa R, in *mutua esclusione*.
- Né U1 né U2 possono chiedere nuovamente l'uso di R se la loro precedente richiesta non è stata prima soddisfatta.
- Il processo gestore assegna R ai richiedenti, in *mutua esclusione e in alternanza*.
- Al termine dell'uso di R il processo utente rilascia la risorsa.

Si formalizzi questo comportamento del sistema mediante una rete di Petri.

Seconda parte (facoltativa: ulteriori punti 4. NB: la parte facoltativa verrà valutata solo se sarà stata svolta correttamente la parte obbligatoria.)

La versione precedente ha il difetto di imporre una rigida alternanza tra il servizio di U1 e quello di U2, con l'effetto di bloccare entrambi i processi se uno dei due, pur potendo effettuare una richiesta, non la esegue quando è il proprio turno. Ad esempio non sarebbe possibile la seguente sequenza, assumendo che inizialmente né U1 né U2 abbiano richiesto la risorsa:

U1 richiede la risorsa e la ottiene; successivamente la risorsa viene rilasciata e U1 la chiede nuovamente; U2 però nel frattempo non l'ha ancora richiesta e quindi la risorsa viene nuovamente assegnata a U1 (mentre se U2 nel frattempo l'avesse richiesta, essa dovrebbe essere *obbligatoriamente* assegnata ad U2).

Si modifichi la precedente rete di Petri in modo da permettere che la risorsa venga assegnata anche più volte di seguito allo stesso processo ma solo se l'altro non l'ha richiesta.

Esercizio 4 (punti 5)

Si consideri il problema di cercare un elemento x compreso tra 0 e 9 in una sequenza ordinata S di lunghezza n che contiene solo i numeri fra 0 e 9 (ciascun elemento può apparire più volte in S).

Si descriva a grandi linee il comportamento di una macchina di Turing a k nastri che, ricevendo in ingresso la stringa $x\$S$, sia in grado, facendo uso di un algoritmo di ricerca binaria, di stabilire se x appartiene a S e se ne calcolino la complessità temporale e spaziale. Sono preferibili soluzioni che minimizzano la complessità *temporale*.

Esercizio 5 (punti 6)

Si descriva un algoritmo, mediante opportuno pseudocodice, che calcoli il determinante del prodotto tra due matrici A e B $n \times n$, sotto l'ipotesi che A e B siano entrambe triangolari superiori (ossia abbiano tutti 0 al di sotto della diagonale principale) o, simmetricamente, triangolari inferiori. Si valuti la complessità asintotica dell'algoritmo in funzione della dimensione n delle matrici.

Suggerimento (per chi avesse qualche difficoltà a richiamare alla memoria gli elementi basilari di algebra lineare)

Ricavare proprietà e algoritmi generali estrapolandoli da esempi relativi a matrici "piccole": e.g., 2×2, 3×3, ...

Esercizio 6 (punti 5)

Si risponda alle seguenti domande, motivando opportunamente le risposte.

1. Si considerino alberi red-black che ammettono l'esistenza di chiavi duplicate:

   a. è possibile che ci siano due elementi con la stessa chiave che *non sono* uno il padre dell'altro?

   b. è possibile che ci siano due elementi con la stessa chiave che *non sono* uno antenanto dell'altro?

2. Modificare gli algoritmi di `RB-INSERT-RB` e `RB-DELETE` visti a lezione in modo da impedire che ci possano essere in un albero chiavi duplicate.
   Dire come cambia (se cambia) la complessità degli algoritmi modificati rispetto alle versioni originali.

Tracce di soluzioni

Esercizio 1

Il linguaggio generato dalla grammatica è L(G) = {c b^{2n} a^{n} | n >= 0}, che è riconosciuto da un semplice automa a pila deterministico.

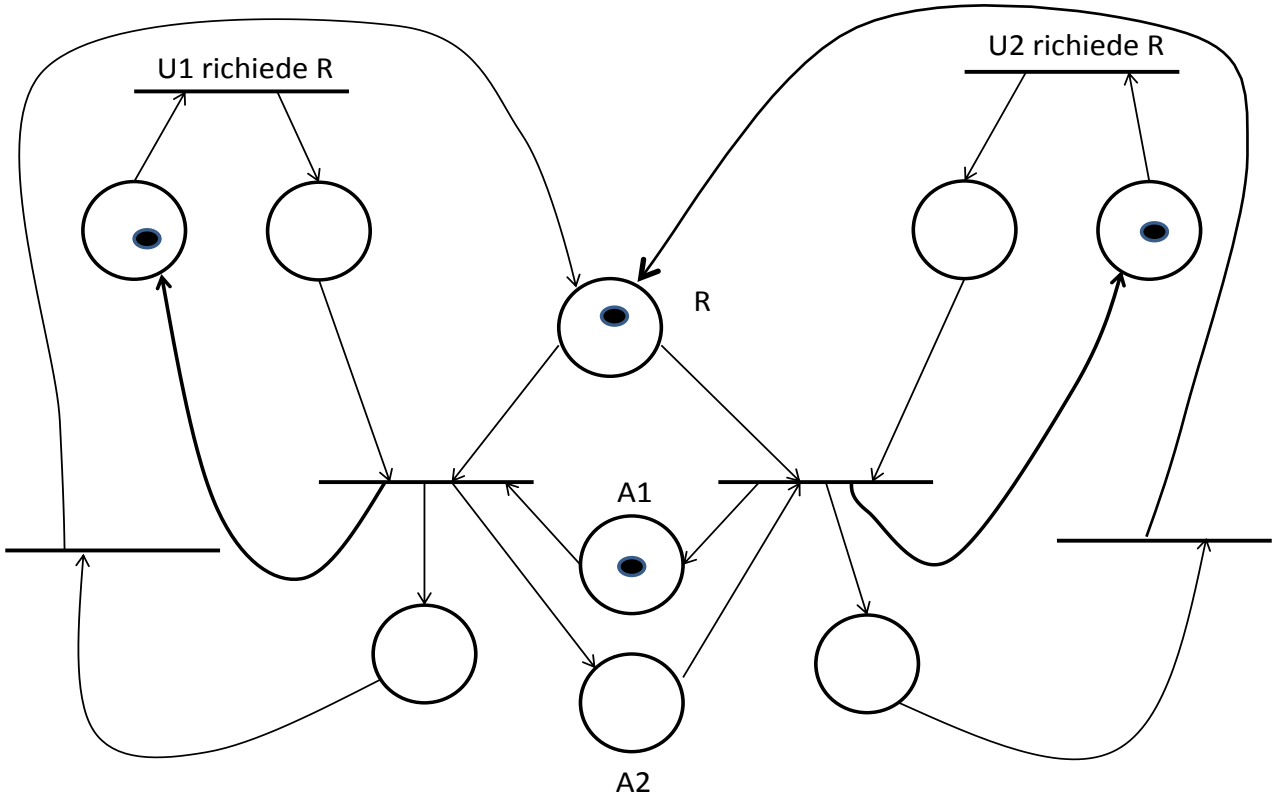
Esercizio 2

La risposta è positiva in entrambi i casi. Infatti il problema può essere *deciso, dimostrando* in effetti la correttezza dell'algoritmo codificato dal programma: in letteratura si trovano diverse dimostrazioni matematiche della sua correttezza, ma anche una semplice analisi informale del codice può portare alla *decisione* che esso effettivamente ordina correttamente un qualsiasi array. Ovviamente, una volta deciso il problema nel caso generale ne consegue la stessa decisione nel caso particolare.

Tuttavia nel primo caso è possibile constatare immediatemente la decidibilità del problema posto, anche senza deciderlo: basta infatti constatare che in tal caso il dominio del problema è finito (l'insieme degli array di lunghezza ≤ 100 e i cui elementi siano a loro volta interi compresi tra -100 e + 100): di conseguenza il problema può essere risolto alla peggio mediante testing esaustivo –posto che a sua volta il problema di stabilire se un array è ordinato è decidibile-.

Esercizio 3

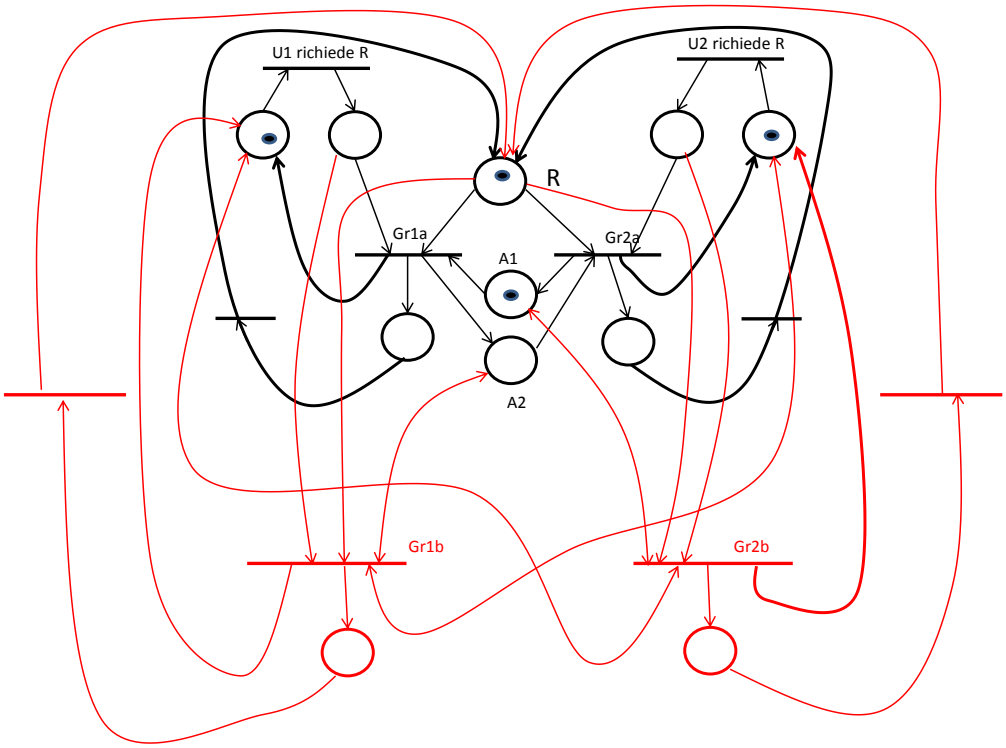
Prima parte (obbligatoria)



A1 e A2 impongono l'alternanza tra il servizio di U1 e quello di U2

Seconda parte (opzionale)

Aggiungendo alla rete precedente le parti in rosso della rete seguente si ottiene l'effetto che, ad esempio, Gr1a (ossia grant di R a U1 avvenga solo se "è il turno di U1"; ma se fosse il turno di U2 e U2 non avesse richiesto la risorsa – e solo in tal caso!-, essa sarebbe ugualmente assegnata ad U1 tramite la transizione Gr1b. In sostanza il comportamento della rete viene reso deterministico in ambo i casi, ma nel primo caso imponendo un'alternanza obbligatoria, nel secondo determinando la scelta tra la concessione di tipo a e quella di tipo b a seconda che sia il proprio turno oppure no ma in assenza di richiesta da parte dell'altro processo.



Esercizo 4

Una soluzione banale consiste nel codificare l'algoritmo tradizionale di ricerca binaria che effettua log(n) accessi alla sequenza S. Tale algoritmo può essere codificato dalla MT con complessità temporale O($n$log$n$); infatti poiché nelle macchine di Turing l'accesso è sequenziale ciascun accesso richiede fino a n passi.

È però possibile ottenere una complessità lineare nel seguente modo, con una macchina di Turing con 3 nastri.

1) Copiare x sul nastro 1 e spostare la testina all'inizio di S.
2) Salvare la lunghezza di S sul nastro 2 (alla fine del procedimento la testina sul nastro di ingresso sarà alla fine di S così come la testina sul nastro 2).
3) Scandire il nastro 2 da sinistra verso destra. Ogni due passi sul nastro 2, muovere di una posizione a sinistra la testina sul nastro di ingresso e scrivere un simbolo sul nastro 3. (alla fine la testina sul nastro in ingresso sarà sul simbolo centrale di S, la testina sul nastro 2 sarà all'inizio del nastro e quella sul nastro 3 sarà alla fine del nastro).

4) Confrontare il simbolo centrale con *x* (salvato nel nastro 1):
    a. Se coincidono, l'esecuzione termina.
    b. Se *x* è minore del simbolo centrale, ripetere i passi 3 e 4 usando il nastro 3 al posto del 2 e il 2 al posto del 3 e muovendosi a destra invece che a sinistra sul nastro di ingresso a partire dalla posizione corrente (si considera come *S* la seconda metà della sequenza corrente).
    c. Se *x* è maggiore del simbolo centrale, ripetere i passi 3 e 4 usando il nastro 3 al posto del 2 e il 2 al posto del 3 (si considera come *S* la prima metà della sequenza corrente).

Con questo procedimento la complessità diventa lineare in quanto la macchina effettua al più log(n) operazioni che costano rispettivamente n, $n/2$, $n/4$, … , cioè $\sum_{i=1}^{logn} 2^i \cong n$.

In entrambe le soluzioni la complessità spaziale è lineare.

Esercizo 5

Si rammenti che il prodotto tra una matrice $n \times m$ e una matrice $m \times p$ è una matrice $n \times p$ il cui generico elemento in osizione *i, j* è datto dalla $\sum_{k=1}^{m} a[i,k] \cdot b[kj]$.

Si constata allora facilmente che il prodotto tra due matrici triangolari (ad esempio, superiori) è a sua volta una matrice triangolare superiore i cui elementi sulla diagonale principale sono il prodotto *a[i, i]*. *b[i,i]*. Notoriamente il determinante di una matrice triangolare è il prodotto degli elementi sulla diagonale principale; esso è quindi dato dalla formula

$$\prod_{i=1}^{n} a[i,i] \cdot b[i,i]$$

che evidentemente può essere calcolata in O(n) a criterio di costo costante.

Esercizo 6

1.a E' possibile, per esempio inserendo le seguenti chiavi: 3, 2, 5, 3

1.b E' possibile, per esempio inserendo 3 volte la stessa chiave (2 sono foglie)

2. C'è da modificare solo l'algoritmo di `RB-NSERT`; la modifica può essere fatta in diverse maniere, per esempio ricercando, prima di inserire l'elemento, se c'è già nell'albero la chiave da inserire. La complessità dell'algoritmo modificato non cambia in termini di comportamento asintotico.

Si può modificare l'algoritmo in modo anche da controllare se esiste già la chiave da inserire mentre si effettua l'inserimento.

Algoritmi e Principi dell'Informatica

Appello del 7 Luglio 2014

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1a, 1b e 2 in 1 ora e 15 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora e 15 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1a (punti 5)

Si consideri il linguaggio L_k, dove k è un parametro, costituito da stringhe sull'alfebto {a, b} tali che ogni volta che si trovino k a consecutive ne segua immediatamente almeno una b.

Si costruisca una rete di Petri che accetti L_3. NB: è preferibile che la rete non contenga transizioni che siano sia in ingresso che in uscita al medesimo posto.

Esercizio 1b (punti 5)

Si scriva una formula del prim'ordine che specifichi un generico linguaggio L_k per un dato valore del parametro k (la formula deve quindi dipendere da k, variabile libera) dell'esercizio 1.a usando variabili intere che indichino la posizione di un carattere nella stringa e i predicati a(i) e b(i) per indicare che il carattere in posizione i-esima è a o b, rispettivamente.

Ad esempio la formula $a(1) \wedge b(2)$ indica che la stringa deve inziare con ab.

NB: qualora fosse utile si può fare uso del predicato $\perp$ (indefinito) per indicare l'assenza di carattere in posizione i: per convenzione si può assumere che valga $\perp$ (0) e che per una stringa lunga n valga $(a(n) \vee b(n)) \wedge \perp (n+1)$.

Esercizio 2 (punti 7)

Una macchina di Turing M_i (che calcola la funzione f_i) è detta *riproducibile* se esiste un'altra macchina di Turing M_j (che calcola la funzione f_j) tale per cui $f_i=f_j$.

Si consideri l'insieme delle MT definite sull'alfabeto di due caratteri {0, 1}. Al suo interno siano:

- F l'insieme delle funzioni calcolate da macchine di Turing riproducibili.
- G l'insieme delle funzioni calcolate da macchine di Turing riproducibili e con meno di 10 stati.
- H l'insieme delle funzioni calcolate da macchine di Turing riproducibili e con più di 10 stati.

Dire se i seguenti problemi sono decidibili:

1) Stabilire se una generica macchina di Turing calcoli una funzione in F.

2) Stabilire se una generica macchina di Turing calcoli una funzione in G.

3) Stabilire se una generica macchina di Turing calcoli una funzione in H.

Esercizio 3 (punti 7)

Si consideri la traduzione $\tau(x) = a^k b^h$, dove $x \in \{a,b\}^+$, k è il numero di b in x e h il numero di a in x.

Si descriva il funzionamento di un MT a nastro singolo che implementi τ usando esclusivamente le celle di memoria contenenti x. Se ne valutino la complessità temporale e spaziale.

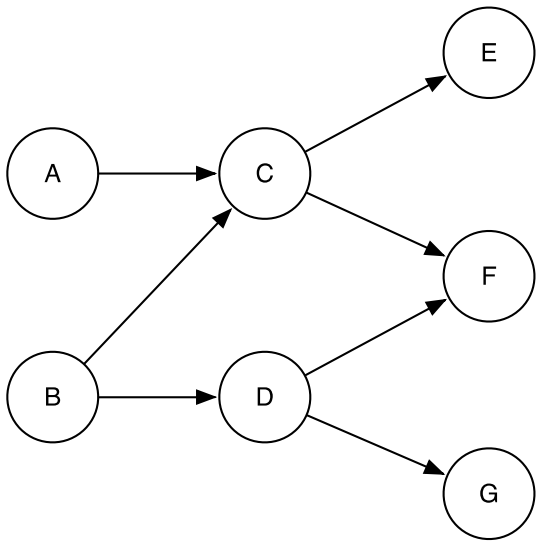
Esercizio 5 (punti 9)

Dati: un elenco di attività e di vincoli di precedenza fra esse

scrivere un algoritmo che rappresenta queste attività in un grafo di precedenze o fallisce se i vincoli di precedenza sono incoerenti (as esempio, se A precede B e B precede A, sia direttamente che indirettamente, i vincoli sono incoerenti).

Indicare la complessità dell'algoritmo

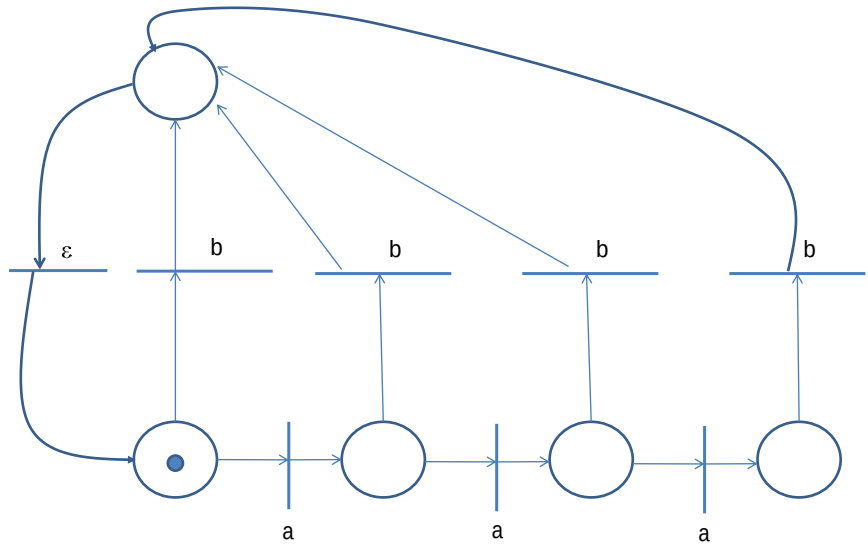
Ad esempio:
Date le 7 attività
A, B, C, D, E, F, G
con i seguenti vincoli di precedenza
A –> C, B –> C, C –> E, D –> F, B –> D, C –> F, D –> G
il grafo di precedenze risultante è



Tracce delle Soluzioni

Esercizio 1°

La rete di Petri seguente, dove è indicata la marcatura iniziale e ogni possibile marcatura seguente viene riconosciuta come finale (un solo token può marcare un solo posto), riconosce il linguaggio L_3.



Esercizio 1.b

$\forall x \, \forall i \, (((x \leq i < x+k) \rightarrow a(i)) \rightarrow b(x+k))$

Esercizio 2

Il problema 1 è decidibile ed è anche deciso. Infatti, ogni macchina di Turing è riproducibile: come è noto, data una funzione calcolabile, esistono infinite macchine di Turing che la calcolano. Quindi F coincide con l'insieme universo di tutte le funzioni calcolabili.

Il problema 2 è indecidibile per il teorema di Rice. Infatti, l'insieme delle macchine di Turing con meno di 10 stati e con alfabeto di due caratteri è finito (proprietà che abbiamo anche usato per enumerare le macchine di Turing), pertanto sarà finito (e non vuoto) anche l'insieme di funzioni calcolabili calcolate da dette macchine di Turing. Allora l'insieme G non è né l'insieme vuoto, né l'insieme di tutte le funzioni computabili, da cui l'indecidibilità per il teorema i Rice.

Il problema posto nel quesito 3 è decidibile per il teorema di Rice. Infatti, ogni funzione calcolabile è calcolabile (anche) mediante una macchina con più di 10 stati. Come scritto in precedenza, per ogni funzione computabile esistono infinite macchine di Turing che la calcolano, mentre le macchine di Turing con un numero di stati minore o uguale a 10 sono in

numero finito, quindi ogni funzione calcolabile dev'essere calcolata anche da macchine con più di 10 stati.

Pertanto H è l'insieme di tutte le funzioni computabili e il problema è allora decidibile per il teorema di Rice.

Esercizio 3

1) la macchina fa una passata scambiando le a con le b e viceversa: complessità ($\Theta(n)$)

2) la macchina fa una passata cercando coppie ba e scambiando la b con la a (scambio: costo costante; costo della passata: $\Theta(n)$)

3) se alla fine della passata non trova coppie ba, si ferma, altrimenti ripete il passo 2)

Al max si fanno n passate, quindi la complessità totale sarà $\Theta(n^2)$.

Esercizio 4

Come suggerito nel testo dell'esercizio, si costruisca un grafo che rappresenti le attività (nodi) e i relativi vincoli di precedenza (archi): i vincoli sono incoerenti se e solo se il grafo contiene cicli. Ciò può essere verificato mediante una opportuna modifica dell'algoritmo di visita Depth-first (costituito da diverse visite ricorsive) che distingua tra i nodi scoperti in una precedente visita e quelli scoperti nella visita corrente; se finisce su uno del primo tipo, la visita corrente termina, e se ne lancia un'altra; nel secondo caso la visita termina restituendo false.

Se l'algoritmo ha successo, cioè non trova cicli, il grafo ottenuto è un DAG e ad esso può essre applicato un topological sort.

FIND-LOOP(G)

/* ritorna flalse se I vincoli sono incoerenti, e quindi se il grafo contiene un ciclo*/

```
1 for each u ∈ G.V
2   u.color := 0
3 c := 0
4 for each u ∈ G.V
5   if u.color = 0
6     c := c+1
7     if FIND-LOOP-VISIT(u, c) = false
8       return false
9 return true
```

FIND-LOOP-VISIT(u, c)

```
1 u.color := c ∈
2 for each v    u.Adj
3   if v.color = c
4     return false
5   if v.color = 0
6     return FIND-LOOP-VISIT(v, c)
7 return true
```

L’algoritmo proposto è quindi O(|V| + |E|) e potrebbe essere incluso come parte integrante in un algoritmo di topological sort senza alterarne la complessità.

Algoritmi e Principi dell'Informatica

Appello del 2 Settembre 2015

Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore e 30 minuti.

Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1 e 2 in 1 ora e 15 minuti.

Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora e 15 minuti.

NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (Punti 8)

Si considerino i linguaggi seguenti:

$L_1 = \{ z \in \{a,b,c\}^* \mid$

$\forall x (\exists y (z = x.y) \rightarrow (\#a(x) \geq \#b(x) \geq \#c(x)) \wedge (\#a(x) - \#b(x) \leq 2) \}$

$L_2 = \{ z \in \{a,b,c\}^* \mid$

$\forall x (\exists y (z = x.y) \rightarrow (\#a(x) \geq \#b(x) \geq \#c(x)) \wedge (\#b(x) - \#c(x) \leq 2) \}$

$L_3 = \{ z \in \{a,b,c\}^* \mid$

$\forall x (\exists y (z = x.y) \rightarrow (\#a(x) \geq \#b(x) \geq \#c(x)) \wedge (\#a(x) - \#c(x) \leq 2) \}$

dove, per un carattere α e una stringa *x*, l'espressione $\#\alpha(x)$ denota il numero di volte in cui il carattere α si ripete nella stringa *x*. Ad es. #a(ababaccc) = 3.

Si costruisca una macchina astratta che riconosca $L = L_1 \cap L_2 \cap L_3$. Tra le diverse macchine astratte che riconoscono L sono preferite macchine "a potenza minima" ossia appartenenti alla categoria di automi a minor potenza riconoscitiva possibile.

Esercizio 2 (9 punti)

Sia $d : N \times N \rightarrow N$ la biiezione tra N e $N \times N$ definita come segue:

$$d(x,y) = (x+y)(x+y+1)/2 + x$$

Si considerino i seguenti insiemi:

- $S_1 = \{ d(x,y) \mid f_x(y) \neq \perp \}$
- $S_2 = \{ x \mid f_x(0) \neq \perp \}$
- $S_3(k) = \{ x \mid f_k(x) \neq \perp \}$, dove k è un parametro.

Per ciascuno dei predetti insiemi, si dica se esso è ricorsivo, motivando brevemente la risposta.

Esercizio 3 (8 punti)

Si definisca un algoritmo per determinare se due alberi binari di ricerca T1 e T2 sono uguali sia per il valore delle chiavi sia per la struttura. Valutare la complessità dell'algoritmo definito.

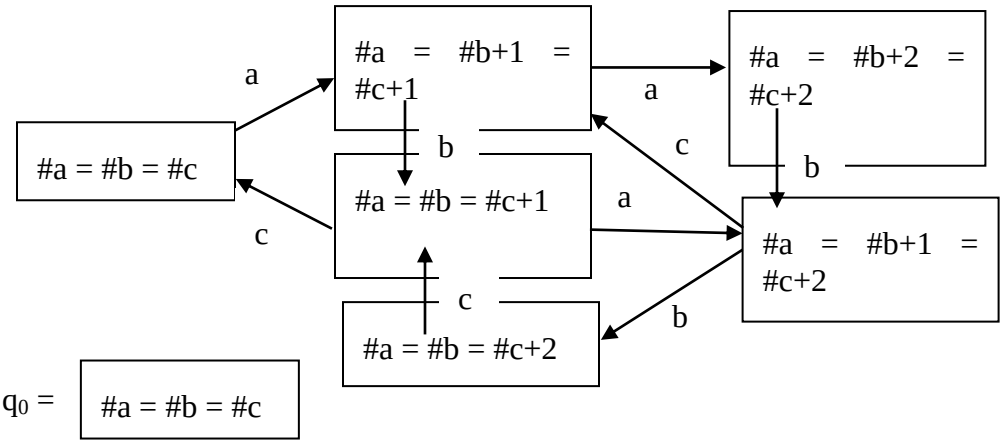
Esercizio 4 (punti 8)

Si consideri il seguente insieme di numeri interi:
{5, 60, 18, 23, 10, 15}.
Si individui una sequenza di inserimenti (senza cancellazioni) di questi valori come chiavi di un albero r-b (rosso-nero) inizialmente vuoto in modo che l'albero risultante abbia altezza complessiva doppia dell'altezza nera. Si ricorda che per convenzione la radice di un sottoalbero non viene computata nell'altezza (né nera né complessiva) mentre le foglie (i nodi T-NIL, neri) vengono computate.
Si mostri il risultato finale ottenuto dopo i vari inserimenti e almeno un risultato intermedio, ad esempio dopo l'inserimento dalla prima metà dei dati.

Tracce delle soluzioni

Esercizio 1

Poiché la differenza tra il numero di a, b e c in ogni prefisso delle stringhe del linguaggio è limitata, un automa a stati finiti è sufficiente per riconoscere il linguaggio:



F = Q

Esercizio 2

S_1 non è ricorsivo. Infatti la funzione d stabilisce banalmente una biiezione tra N e NxN, e se si potesse stabilire l'insieme delle coppie di numeri (x,y) tali per cui $f_x(y) \neq \perp$ allora il problema dell'arresto risulterebbe decidibile, il che è assurdo.

S_2 non è ricorsivo. Sia F l'insieme di funzioni calcolabili definite nel punto 0; tale insieme non è né l'insieme di tutte le funzioni computabili, né l'insieme vuoto. L'insieme S_2 di indici delle macchine di Turing che calcolino funzioni in F non è pertanto ricorsivo per il teorema di Rice.

La ricorsività di $S_3(k)$ dipende da k. Per alcuni valori di k si sa esattamente per quali valori di ingresso x la funzione $f_k(x)$ è definita. Ad esempio, se $f_k(x)$ fosse la funzione x^2, si saprebbe che è definita sempre e quindi il problema sarebbe decidibile (e deciso). Per altri valori di k, invece, il problema è indecidibile. Se, ad esempio, $f_k(x)$ fosse la funzione $f_x(0)$, tale funzione sarebbe certamente calcolabile, ma, come ad esempio osservato al punto b), non se ne potrebbe stabilire l'insieme di definizione.

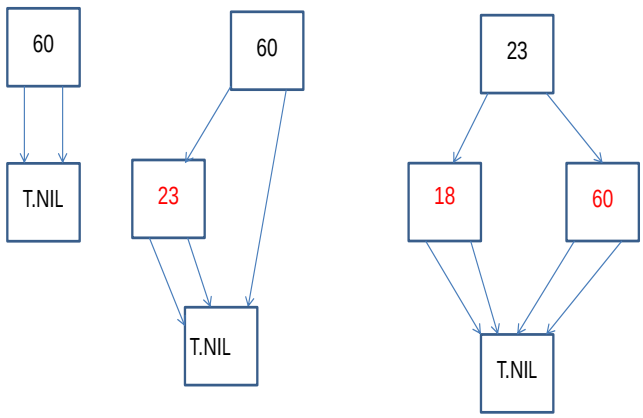
Esercizio 3

È possibile definire un algoritmo ricorsivo che, dopo aver verificato che le radici degli alberi non sono nulle e che le chiavi sono uguali, visita ricorsivamente il sottoalbero sinistro e quello destro e restituisce valore false se una delle condizioni verificate è falsa. La complessità dell'algoritmo è proporzionale al numero di nodi.

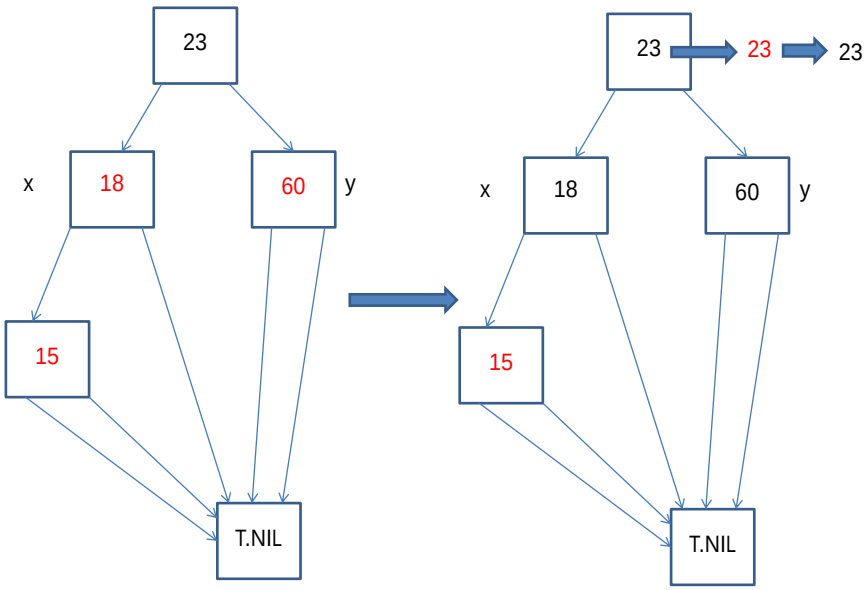
```
boolean compare(Tree T1, Tree T2){
if (T1 == null && T2 == null)
    return true;
if (T1 != null && T2 != null)
    return((T1.key == T2.key) && compare(T1.left, T2.left) &&
           compare(T1.right, T2.right));
  else return(false);
}
```

Esercizio 4

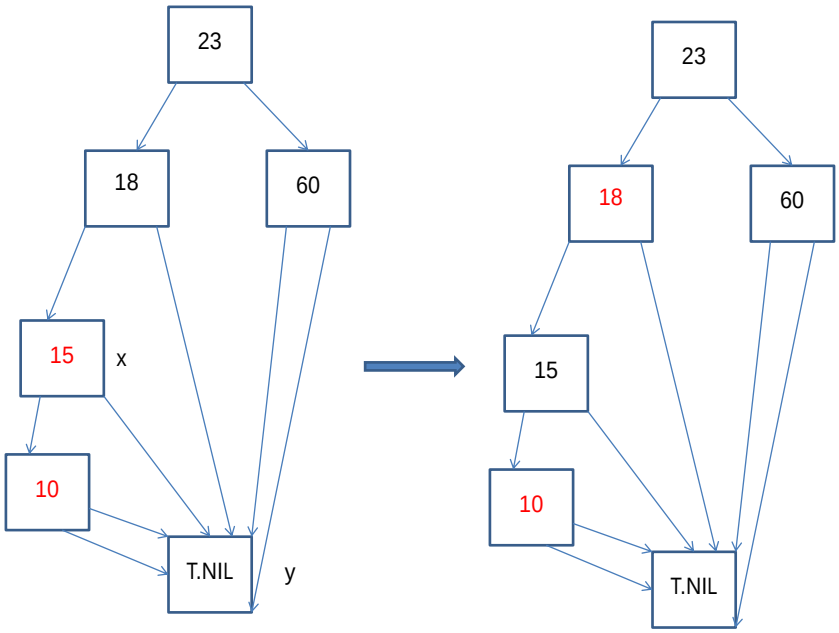
Se si inseriscono i dati in ordine crescente o decrescente (e.g. {60, 23, 18, 15,10, 5} si ottiene un albero in cui ogni sottoalbero ha la proprietà che il cammino sinistro alterna nodi rossi e nodi neri mentre il cammino destro ha solo nodi neri ed è quindi lungo la metà del sinistro. La figura seguente mostra lo stato dell'albero dopo l'inserimento di, rispettivamente:

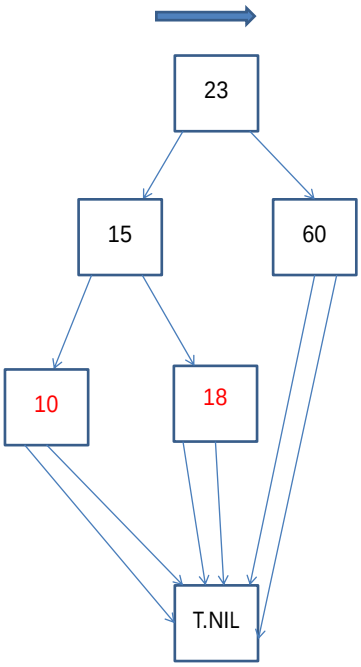


Inserimento di 60, 23, 18

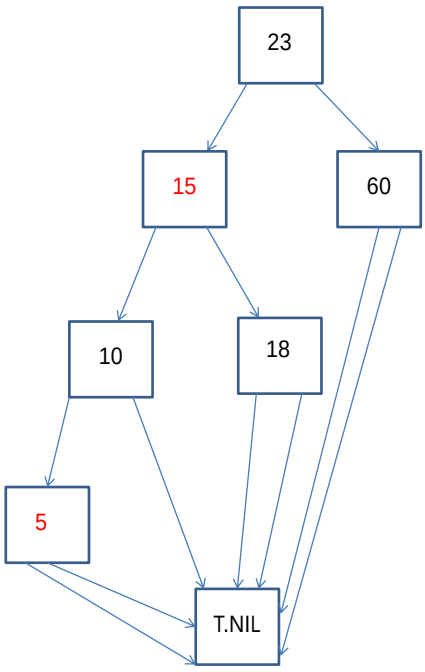


Inserimento di 15





Inserimento di 10



Risultato finale.