Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

28 giugno 2022

Informatica teorica - Tempo a disposizione: 1 ora

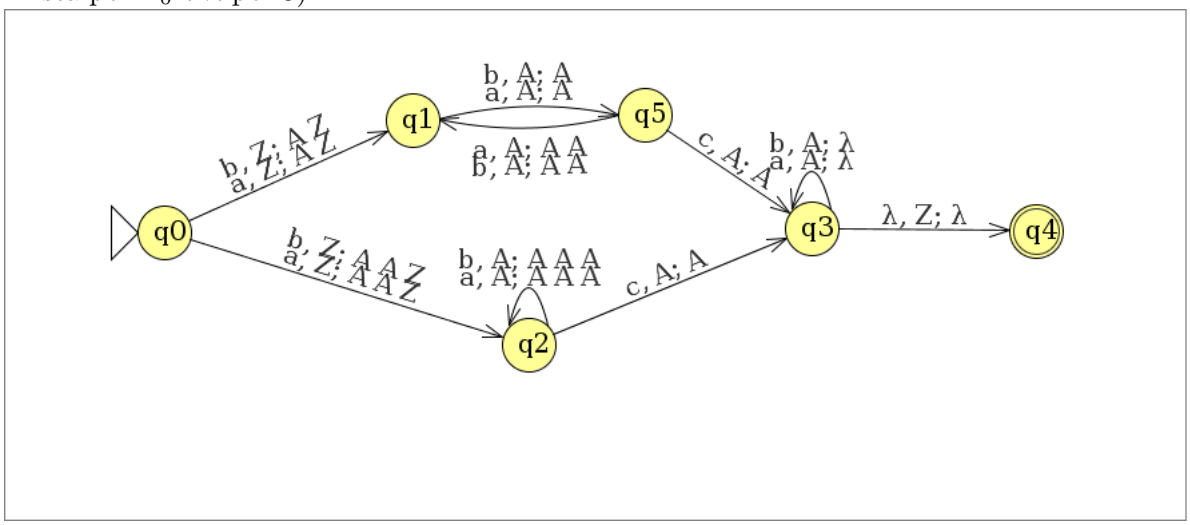
Esercizio 1 (8 punti)

Si definisca un automa a potenza minima per il linguaggio

$$\{xcy \mid x, y \in \{a, b\}^+, |x| = 2|y| \text{ oppure } 2|x| = |y|\}.$$

Soluzione

Si tratta di un automa a pila non-deterministico – se ne allega la versione JFLAP (si ricorda che Z sta per Z_0 e λ per ε).



Esercizio 2 (8 punti). Chi ha diritto alla riduzione del 30% della prova svolga unicamente il punto 2.1.

1. Siano dati due automi a pila deterministici con linguaggi L_1 e L_2: è ricorsivo il linguaggio $L_1 \cap L_2$?
2. Siano dati due automi a pila deterministici con linguaggi L_1 e L_2 e una macchina di Turing M: è decidibile se M calcola $L_1 \cap L_2$?

Soluzione

1. Sì, basta simulare i due APD con una MT e vedere se entrambi accettano la stringa in ingresso.
2. No, per il teorema di Rice: si tratta del consueto problema di determinare la correttezza di una generica macchina M.

Algoritmi e Principi dell'Informatica
Soluzioni al Tema d'esame

28 giugno 2022

Algoritmi e strutture dati - Tempo a disposizione: 1 ora

Esercizio 3 (8 punti)
Si consideri il problema di stampare, senza duplicati, gli elementi di un array `a` non ordinato di n interi strettamente positivi. Per ciascuno dei casi seguenti, si fornisca una soluzione che garantisca la migliore complessità asintotica: a) caso ottimo; b) caso medio; c) caso pessimo. Per ciascuna delle soluzioni presentate, si specifichino le complessità nei tre casi.

Soluzione

Nel caso ottimo, basta una semplice scansione con due cicli nidificati:

```
ottimo(a)
  b := [true, ..., true] // array di n booleani
  for i := 1 to a.length
    if b[i]
      print a[i]
      if i+1 <= a.length
        for j := i+1 to a.length
          if a[i] = a[j]
            b[j] := false
```

Nel caso pessimo e nel caso medio, la complessità è quadratica a causa dei due cicli nidificati. Nel caso ottimo, se c'è n volte lo stesso numero, la complessità è lineare.

Anche l'algoritmo seguente ha la stessa complessità nel caso ottimo, ma migliora il costo nel caso medio mediante una hashtable.

```
medio(a)
  T := [NIL, ..., NIL] // tabella di hash vuota
  for i := 1 to a.length
    if HASH-SEARCH(T,a[i]) = NIL
      print(a[i])
      HASH-INSERT(T,a[i])
```

Nel caso medio, la complessità si abbassa a $O(n)$, poiché la ricerca e l'inserimento hanno complessità $O(1)$. Idem nel caso ottimo. Nel caso pessimo, la complessità è quadratica, poiché ricerca e inserimento sono in $O(n)$.

Si può migliorare la complessità nel caso pessimo preordinando l'array.

```
pessimo(a)
  sort(a)  // mediante heapsort
  k := 0
  for i := 1 to a.length
    if a[i] ≠ k
```

```
        k := a[i]
        print(k)
```

In tutti i casi, prevale il costo dell'ordinamento, con una complessità asintotica di $O(n \log(n))$.

Esercizio 4 (8 punti + bonus). Chi ha diritto alla riduzione del 30% della prova svolga unicamente i punti 4.1 e 4.3.

Si consideri il linguaggio $L = \{ww' \mid w, w' \in \{0,1\}^+, h(w') = w\}$, dove h è l'omomorfismo $h(0) = 1$, $h(1) = 0$.

Si descrivano esaurientemente delle macchine dei tipi seguenti che accettino L, valutandone le complessità spaziali e temporali con tutti i criteri di costo disponibili:

1. macchina di Turing a nastro singolo;
2. macchina RAM.

Bonus: È possibile ottenere una complessità spaziale costante con la macchina RAM con il criterio di costo costante?

Soluzione

1. La macchina di Turing a nastro singolo fa avanti e indietro, marcando i simboli incontrati dalle estremità fino al centro (rifiutando l'input se la lunghezza della stringa non è pari). A questo punto riavvolge il nastro, togliendo la marcatura dalla prima metà dei simboli. Infine controlla le corrispondenze tra i simboli nelle due metà, facendo avanti e indietro, marcando nuovamente i simboli nella prima metà e contrassegnando con un *blank* quelli nella seconda metà. La complessità temporale è $\Theta(n^2)$ e quella spaziale $\Theta(n)$, con n lunghezza della stringa in ingresso.
2. La macchina RAM copia la stringa in memoria, sfruttando un contatore, calcolandone quindi la lunghezza n. Poi verifica la corrispondenza tra le due metà con due contatori, uno che inizia da 0 e uno da $n/2$ mediante accessi diretti. Con il criterio di costo costante, la complessità spaziale è $O(n)$ (2 celle per i contatori e $O(n)$ celle per la stringa) e quella temporale è $O(n)$. Con il criterio di costo logaritmico, la complessità spaziale è sempre $O(n)$ (anche se i contatori occupano $O(\log(n))$ celle), mentre quella temporale è $O(n \log(n))$ a causa del contatore.

Bonus. Si può sfruttare l'inadeguatezza del criterio di costo costante per codificare in una singola cella di memoria tutta la stringa in ingresso, come sequenza di bit. Le operazioni necessarie per leggere e scrivere la sequenza di bit richiedono essenzialmente la gestione di un contatore come nella precedente soluzione; con il criterio di costo costante, questo prevede un costo spaziale costante.