Algoritmi e Principi dell'Informatica
Appello del 24 Gennaio 2020
Chi deve sostenere l'esame integrato (API) deve svolgere tutti gli esercizi in 2 ore.
Chi deve sostenere solo il modulo di *Informatica teorica* deve svolgere gli Esercizi 1 e 2 in 1 ora.
Chi deve sostenere solo il modulo di *Informatica 3* deve svolgere gli Esercizi 3 e 4 in 1 ora.
NB: i punti attribuiti ai singoli esercizi hanno senso solo con riferimento all'esame integrato e hanno valore puramente indicativo.

Esercizio 1 (8 punti)
Dimostrare che la famiglia di insiemi semidecidibili è chiusa rispetto all'intersezione, cioè, dati due insiemi qualunque S_1 e S_2, entrambi semidecidibili, la loro intersezione $S_{12} = S_1 \cap S_2$ è pure semidecidibile. A questo scopo, si risponda alle seguenti domande:

1. Delineare una procedura $semDec_{S12}$ che "semidecide" S_{12}, cioè tale che per $x \in S_{12}$ $semDec_{S12}(x)$ termini e restituisca "vero", mentre per $x \notin S_{12}$ $semDec_{S12}(x)$ non termini; si può assumere l'esistenza di simili procedure $semDec_{S1}$ e $semDec_{S2}$ che semidecidono S_1 e S_2.
2. Delineare una procedura che enumeri S_{12}.
3. Con l'assunzione che $S_{12} \neq \emptyset$, scrivere una definizione della funzione generatrice g_{S12} di S_{12} (con questo termine si intende la funzione totale e calcolabile la cui immagine sia S_{12}).
   Suggerimento: sfruttare il fatto che le funzioni generatrici g_{S1} di S_1 e g_{S2} di S_2 esistono; assicurarsi che g_{S12} sia totale e computabile e che la sua immagine coincida con S_{12}.

Esercizio 2 (8 punti)
Sia dato un alfabeto Σ. Date due stringhe qualunque $x,y \in \Sigma^*$, diciamo che y è una *sottostringa* di x se e solo se y può essere ottenuta da x rimuovendo alcuni caratteri (anche nessuno o tutti). Dunque, ad esempio, tutte e sole le sottostringhe di $x = abaa$ sono: ε, *a, b, ab, ba, aa, aba, aaa, baa, abaa.*
Dato un qualunque linguaggio L di alfabeto Σ, ossia $L \subseteq \Sigma^*$, definiamo SUBSEQ(L) come il linguaggio formato da tutte le sottostringhe di tutte le stringhe in L, ossia:

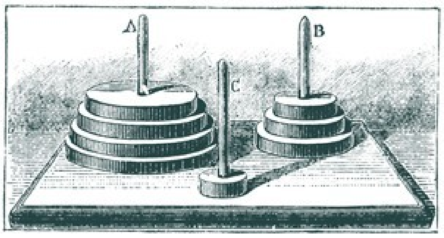
$$\text{SUBSEQ(L)} = \{y \in \Sigma^* \mid y \text{ è una sottostringa di } x, \text{ per qualche } x \in L\}$$

Sia ora Σ un alfabeto unario, ossia $|\Sigma| = 1$. Si dimostri, o si confuti, la seguente affermazione:

Per *ogni* linguaggio L su alfabeto unario, SUBSEQ(L) è un linguaggio regolare.

Esercizio 3 (7 punti)
Si descriva una MT a nastro singolo che accetti il linguaggio $\{a^n \mid n = 3^k, k \geq 0\}$, calcolandone la complessità spaziale e temporale.

Esercizio 4 (8 punti + 2 per il bonus, che sarà valutato solo se la soluzione della prima parte varrà almeno 6 punti)
Nel rompicapo della *Torre di Hanoi* vi sono tre paletti (indicati con A, B, e C) e n di dischi di grandezza decrescente, che possono essere infilati in uno qualsiasi dei paletti. Il gioco inizia con tutti i dischi incolonnati sul paletto A in ordine decrescente. Lo scopo è portare tutti i dischi da A a B, seguendo due semplici regole:



- si può spostare solo un disco alla volta;
- si può mettere un disco solo sopra un altro disco più grande, mai su uno più piccolo.

Il problema di stampare tutte le mosse (cioè le coppie <paletto-di-partenza, paletto-di-arrivo>), dato n, ammette un'immediata soluzione ricorsiva, basata sull'ipotesi di saper risolvere lo stesso problema con n-1 dischi. Si codifichi tale soluzione e se ne valuti la complessità asintotica temporale.

Bonus: si numerino i dischi da 1 (disco più piccolo) a n (disco più grande) e si consideri il problema di stampare il numero del disco spostato alla i-esima mossa dell'algoritmo ricorsivo di cui sopra. Ad esempio, con n=4, la sequenza di dischi mossi è 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1. Si descriva una soluzione opportuna e se ne valutino le complessità asintotiche spaziale e temporale.

Tracce delle soluzioni

Esercizio 1

1. *semDec*$_{S12}$ per ogni input *x* esegue in modo alternato *semDec*$_{S1}$ e *semDec*$_{S2}$ per un numero crescente di volte; si arresta e restituisce "vero" se e solo se sia *semDec*$_{S1}$ sia *semDec*$_{S2}$ si arrestano e restituiscono "vero".
2. Basta eseguire una simulazione "diagonale" (dovetailing) di *semDec*$_{S12}$ che usi valori crescenti dei propri argomenti e del numero di passi da eseguire per ogni argomento.
3. Si consideri la biiezione $d(x, y)=(x+y)(x+y+1)/2+1$ tra coppie di interi e interi.
   Sia $(x, y) = d^{-1}(n)$ e K un qualunque elemento di S_{12} (che per ipotesi non è vuoto), allora
   $g_{S12}(n)$ = se $g_{S1}(x)=g_{S2}(y)$ allora $g_{S1}(x)$ (o equivalentemente $g_{S2}(y)$) altrimenti K

Esercizio 2

L'affermazione è vera. Si consideri infatti un qualunque linguaggio L su alfabeto unario (per fissare le idee: $\Sigma = \{a\}$). Ora si hanno due casi: o L è infinito o è finito.

Se L è *finito*, significa che è definita la lunghezza massima di una stringa in L, cioè $\max_{x\in L}|x| = n$ per qualche $n\in N$. Essendo l'alfabeto unario, $a^n \in L$. Ma allora è facile capire che tutte e sole le sottostringhe di a^n sono l'insieme $S=\{a^k \mid 0 \le k \le n\}$, e per di più non possono esisterne altre in SUBSEQ(L), ossia S = SUBSEQ(L). Essendo anche SUBSEQ(L) di dimensione finita, è per forza regolare.

Se invece L è *infinito*, significa che esistono in L stringhe arbitrariamente grandi, ossia per ogni $n\in N$, esiste un $n'>n$ tale che $a^{n'}\in L$. Perciò, per ogni $n\in N$ esiste un $n'>n$ tale che l'insieme $\{a^k \mid 0 \le k \le n'\}$ è un sottoinsieme di SUBSEQ(L). Pertanto, in pratica, SUBSEQ(L) = $\{a^k \mid k\in N\}$, e dunque SUBSEQ(L) è il linguaggio universo su Σ, che è evidentemente regolare.

In verità si può addirittura dimostrare che per ogni linguaggio L su qualunque alfabeto finito Σ (di qualunque cardinalità), SUBSEQ(L) è regolare. La dimostrazione, piuttosto impegnativa, è nota come Lemma di Higman; si veda A. G. Higman: Ordering by divisibility in abstract algebra. Proceedings of the London Mathematical Society, 3:326-336, 1952.

Esercizio 3
si fanno k passate sull'ingresso:
1) ogni 3 'a' si cancellano il secondo e il terzo: se non ci sono, non si accetta
2) se rimane un solo 'a' si accetta, altrimenti si torna ad 1)
Sia $n = |x| = 3^k$: $S(n) = \Theta(n)$, $T(n) = \Theta(n \log n)$

Esercizio 4

```
Hanoi(n) {
        Hanoi(n, “A”, “B”, “C”)
}
Hanoi(n, da, a, via) {
        if n = 0 then return
        Hanoi(n - 1, da, via, a)
        print(“<” + da + “,” + a + “>”)
        Hanoi(n - 1, via, a, da)
}
```

La complessità asintotica è data dalla ricorrenza $T(n)=2T(n-1)+c$, dove $c=\Theta(1)$.
Dall'"ulteriore risultato" presentato dopo il teorema dell'esperto deriva immediatamente la complessità $T(n)=O(2^n)$. Si può ottenere lo stesso risultato anche con il metodo di sostituzione.
Alternativamente, in questo caso è semplice arrivare alla soluzione sostituendo iterativamente la ricorrenza fino ad inserire il termine $T(0)$, che possiamo assumere essere pari a una costante d:

$T(n) = 2 (2 T(n-2) + c) + c$
$T(n) = 2^2 T(n-2) + 2^1 c + 2^0 c$
$T(n) = 2^k T(n-k) + 2^{k-1} c + 2^{k-2} c + \ldots + 2^0 c$
$T(n) = 2^n d + (2^{n-1} + 2^{n-2} + \ldots + 2^0)c = O(2^n)$

Parte bonus. Il disco più piccolo si sposta in tutte le mosse dispari. Il disco 2 si sposta nelle mosse pari, ma solo quelle non divisibili per 4. Il disco 3 si sposta nelle mosse divisibili per 4 ma non per 8. E così via. In altre parole, dato *i* (numero della mossa) se ne testa la divisibilità per potenze via via crescenti di 2. Sia *p* il più piccolo intero tale per cui *i* non è divisibile per 2^p: il disco da spostare è *p*.
Poiché le mosse sono al massimo 2^n-1, i test di divisibilità per potenze di 2 sono al più $O(\log_2(2^n-1))=O(n)$. In particolare, basta contare il massimo numero *p* di bit meno significativi di *i* posti a 0, quindi con una complessità temporale di $O(n)$.
La complessità spaziale è $O(1)$ (*i* e *n* sono in input e l'unico parametro in più gestito dal programma è *p*).