PROGRAMMAZIONE I (A,B) - a.a. 2016-17
Prima Verifica – 31 ottobre 2016

Esercizio 1

Si scriva una funzione **C** che, dato un array a di dimensione dim, restituisce il seguente valore di verità

$$\#\{j \mid j \in [0, dim-1) \wedge (\forall i.\ i \in [j+1, dim) \rightarrow a[i] \neq a[j])\} = dim - 1$$

Si ricorda che, dato un insieme finito S, $\#S$ ne denota la cardinalità.

Soluzione Supponendo di avere una funzione

```
int conta (int x, int a[], int init, int end)
```

che conta le occorrenze dell'intero x nell'intervallo $[init, end)$ dell'array a, un semplice frammento di programma risolutivo è il seguente

```
int tmp = 0;
for (int i = 0; i < dim-1; i++)
    if (conta(a[i], a, i+1, dim) == 0) tmp++;
if (tmp != dim-1) return 0;
else return 1;
```

Ma se si nota che la richiesta vuol semplicemente dire che tutti gli elementi dell'array sono diversi, una soluzione più semplice è

```
for (int i = 0; i < dim-1; i++)
    if (conta(a[i], a, i+1, dim) != 0) return 0;
return 1;
```

Esercizio 2

Si scriva una funzione **C** con la seguente firma

```
int contaUnico (int a[], int b[], int dima, int dimb)
```

che, dati due array a e b di dimensioni rispettivamente $dima$ e $dimb$, restituisce il numero di elementi diversi di a che sono contenuti in b. Ad esempio, se applicata sui seguenti array

```
a = [60; 20; 30; 10; 20; 40; 60; 80]
b = [60; 30; 30; 10; 20; 60]
```

la funzione deve restituire il valore 4, corrispondente all'occorrenza in b degli elementi 60, 20, 30 e 10 appartenenti ad a.

Soluzione Il punto è capire in quale momento eseguire la verifica sulla presenza in b per un elemento di a, in modo da non controllare più volte lo stesso elemento. Una scelta possibile è l'ultima occorrenza dell'elemento in a. In formula, tale richiesta è

$$\#\{j \mid j \in [0, dima) \wedge (\forall i.\ i \in [j+1, dima) \rightarrow a[i] \neq a[j]) \wedge (\exists k.\ k \in [0, dimb) \wedge a[j] = b[k])\}$$

dove rispetto alla formula dell'esercizio precedente viene controllato anche il valore $a[dima - 1]$.

Supponendo sempre di avere a disposizione la funzione *conta*, e assumendo che tale funzione restituisca 0 nel caso $init \geq end$, si può scrivere il frammento

```
int tmp = 0;
for (int i = 0; i < dima; i++)
    if ((conta(a[i], a, i+1, dima) == 0) && (conta(a[i], b, 0, dimb) > 0))
       tmp++;
return tmp;
```

Esercizio 3

Si definisca una grammatica libera che genera il seguente linguaggio $\mathcal{L}$ sull'alfabeto $\Lambda = \{a, b, c\}$

$$\mathcal{L} = \{a^p b^n a^m b^q \mid m, n, p, q > 0 \wedge m > n \wedge p > q\}$$

Si dica inoltre se il linguaggio $\mathcal{L}$ è o meno regolare, fornendone la prova.

Soluzione Il linguaggio può essere descritto anche nella maniera sottostante

$$\mathcal{L} = \{a^q a^r b^n a^n a^o b^q \mid n, o, q, r > 0\}$$

e a questo punto diventa semplice trovare la grammatica, che è fornita sotto

```
S -> aSb | aANAb
A -> aA | a
N -> bNa | ba
```

La non regolarità si dimostra con una applicazione standard del pumping lemma, scegliendo ad esempio per un intero k la string $a^k abaab^k$.

Esercizio 4

Si dica qual è il linguaggio generato dalla seguente grammatica libera sull'alfabeto $\Lambda = \{a, b, c\}$

```
S -> aaS | Scc | A
A -> AbA | b
```

e se tale linguaggio è o meno regolare, fornendone la prova.

Soluzione È immediato notare che le a e le c devono occorrere in numero pari, appena meno ovvio che le b devono occorrere in numero dispari, e dunque il linguaggio è

$$\mathcal{L} = \{a^{2m} b^{2n+1} c^{2o} \mid m, n, o \geq 0\}$$

La grammatica però non è regolare, e il modo più semplice per provare che il linguaggio invece lo è è fornire il semplice automa che lo riconosce.