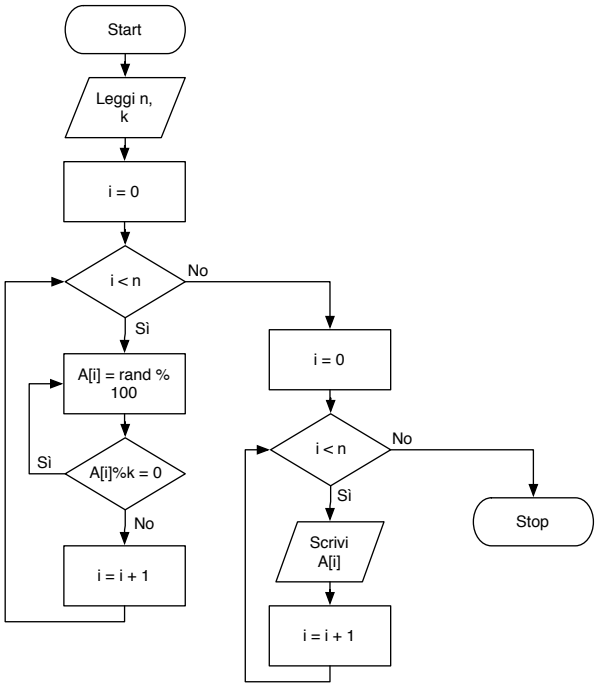
Algoritmi e Strutture Dati (IN110)
Esercitazione n. 4

Marco Liverani *

Esercizio n. 1

Letti in input due numeri interi $n > 0$ e $k > 1$ costruire un array A di n numeri interi casuali minori di 100 che non siano multipli di k. Stampare l'array A.

Diagramma di flusso



*Università degli Studi Roma Tre, Corso di Laurea in Matematica, Corso di Algoritmi e Strutture Dati (IN110) – sito web del corso http://www.mat.uniroma3.it/users/liverani/IN110/

Pseudo-codifica dell'algoritmo

1: leggi n e k
2: **per** $i = 0, 1, 2, \ldots, n-1$ **ripeti**
3: **ripeti**
4: sia A_i un numero casuale compreso tra 0 e 100
5: **fino a quando** A_i % $k \neq 0$
6: **fine-ciclo**
7: **per** $i = 0, 1, 2, \ldots, n-1$ **ripeti**
8: scrivi A_i
9: **fine-ciclo**
10: stop

Codifica in linguaggio C

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define MAX 50

int costruisci_array(int X[]) {
  int i, n, k;
  printf("Inserisci n e k: ");
  scanf("%d %d", &n, &k);
  srand((unsigned)time(NULL));
  for (i=0; i<n; i++) {
    do {
      X[i] = rand()%100;
    } while (X[i] % k == 0);
  }
  return(n);
}

void stampa_array(int X[], int n) {
  int i;
  for (i=0; i<n; i++)
    printf("%d ", X[i]);
  printf("\n");
  return;
}

int main(void) {
  int A[MAX], n;
  n = costruisci_array(A);
  stampa_array(A, n);
  return(0);
}
```

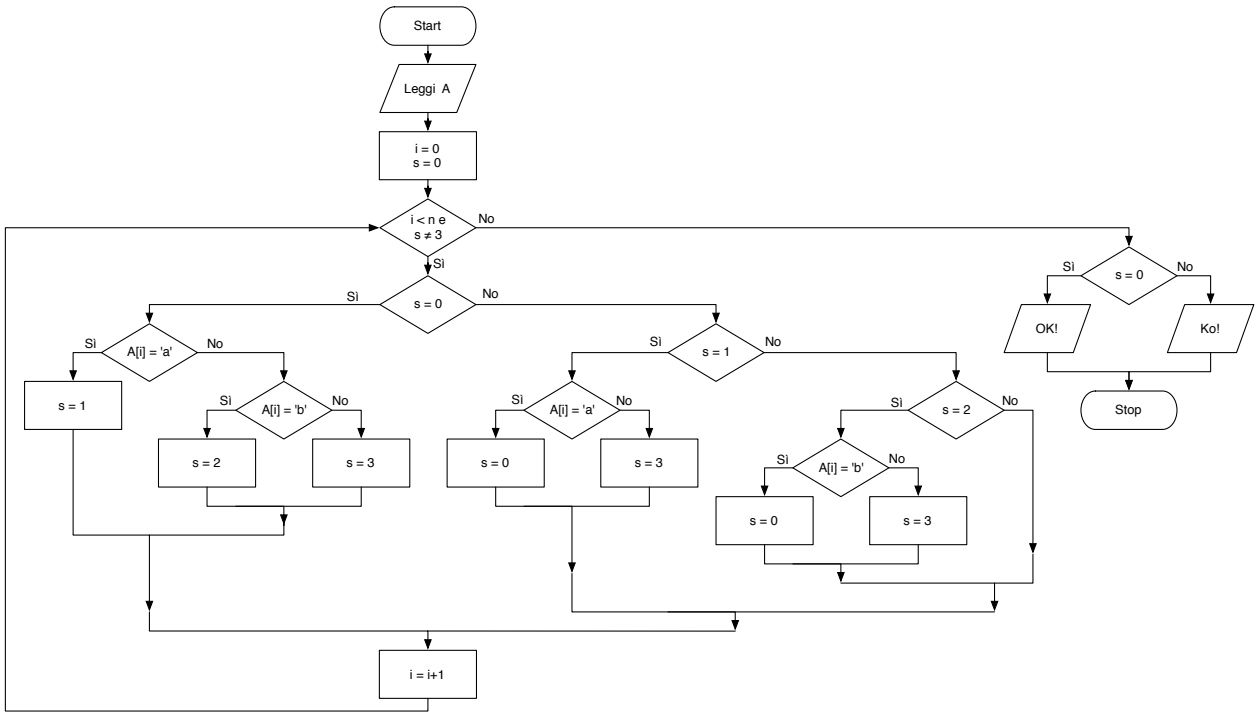
Esercizio n. 2

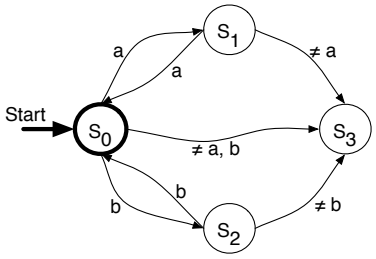
Letta una stringa di caratteri verificare se à costituita da sequenze alternate di lunghezza pari di “a” e di “b”. Ad esempio la stringa “aaaabbaaaaaabbbbaa” rispetta la regola descritta nel testo dell’esercizio; viceversa le stringhe “aabaabbb” e “aaaabbccaabbbb” non la rispettano.

Prima soluzione

La prima soluzione realizza un automa a stati finiti (quattro stati) che accetta o rifiuta le stringhe del linguaggio definito nel testo dell’esercizio. Lo stato s varia durante l’analisi della stringa: $s = 0$ indica che le sequenze alternate di “a” e “b” sono di lunghezza pari, $s = 1$ che le “a’ sono dispari, $s = 2$ che le “b” sono dispari, $s = 3$ che ci sono caratteri diversi da “a” e “b”; inizialmente $s = 0$ e naturalmente se al termine dell’analisi di tutta la stringa risulta $s = 0$ allora vuol dire che la stringa può essere accettata. La tecnica utilizzata può essere facilmente generalizzata per riconoscere molti altri “pattern”.

Diagramma di flusso





Grafo delle transizioni di stato

Pseudo-codifica dell'algoritmo

1: leggi la stringa a di n caratteri
2: sia s la variabile che indica lo stato del processo di analisi: $s = 0$ se le sequenze alternate di "a" e "b" sono di lunghezza pari, $s = 1$ se le "a' sono dispari, $s = 2$ se le "b" sono dispari, $s = 3$ se ci sono caratteri diversi da "a" e "b"
3: sia inizialmente $s = 0$
4: **per ogni** carattere a_i della stringa ($i = 0, 1, \ldots, n-1$) se $s \neq 3$ **ripeti**
5: **se** $s = 0$ **allora**
6: **se** a_i = "a" **allora**
7: poni $s = 1$, perché le "a" sono dispari
8: **altrimenti se** a_i = "b" **allora**
9: poni $s = 2$, perché le "b" sono dispari
10: **altrimenti**
11: poni $s = 3$ perché a_i non è né una "a" né una "b"
12: **fine-condizione**
13: **altrimenti se** $s = 1$ **allora**
14: **se** a_i = "a" **allora**
15: poni $s = 0$, perché le "a" sono tornate pari
16: **altrimenti**
17: poni $s = 3$ perché a_i non è una "a" e quindi si è interrotta una sequenza di caratteri "a" di lunghezza dispari
18: **fine-condizione**
19: **altrimenti se** $s = 2$ **allora**
20: **se** a_i = "b" **allora**
21: poni $s = 0$, perché le "b" sono tornate pari
22: **altrimenti**
23: poni $s = 3$ perché a_i non è una "b" e quindi si è interrotta una sequenza di caratteri "b" di lunghezza dispari
24: **fine-condizione**
25: **fine-condizione**
26: **fine-ciclo**
27: **se** $s = 0$ **allora**
28: la stringa rispetta la regola
29: **altrimenti**
30: la stringa NON rispetta la regola
31: **fine-condizione**
32: stop

Codifica in linguaggio C

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define MAX 50

int main(voi) {
  char s[MAX];
  int stato, i;
  printf("Inserisci una stringa di caratteri: ");
  scanf("%s", s);
  stato = 0;
  for (i=0; i<strlen(s) && stato != 3; i++) {
    if (stato == 0) {
```

```
14      if (s[i] == 'a')
15        stato = 1;
16      else if (s[i] == 'b')
17        stato = 2;
18      else
19        stato = 3;
20    } else if (stato == 1) {
21      if (s[i] == 'a')
22        stato = 0;
23      else
24        stato = 3;
25    } else if (stato == 2) {
26      if (s[i] == 'b')
27        stato = 0;
28      else
29        stato = 3;
30    }
31  }
32  if (stato == 0)
33    printf("La stringa '%s' rispetta la regola.\n", s);
34  else
35    printf("La stringa '%s' NON rispetta la regola.\n", s);
36  return(1);
37 }
```

Seconda soluzione

La seconda soluzione, più elementare, consiste nello scorrimento della stringa e nella verifica della consistenza della condizione specifica con cui è stato definito questo "linguaggio"; è più complicato adattare la stessa strategia al riconoscimento di *pattern* costruiti sulla base di altre regole.

Pseudo-codifica dell'algoritmo

1: leggi la stringa a composta da n caratteri
2: $i = 0$, $ok = 1$
3: **fintanto che** $i < n$ e $ok = 1$ **ripeti**
4: $ca = 0$, $cb = 0$
5: **se** $a_i \neq$ "a" e $a_i \neq$ "b" **allora**
6: $ok = 0$
7: **altrimenti**
8: **fintanto che** $i < n$ e $a_i =$ "a" **ripeti**
9: $ca = ca + 1$, $i = i + 1$
10: **fine-ciclo**
11: **se** ca è dispari **allora**
12: $ok = 0$
13: **altrimenti**
14: **fintanto che** $i < n$ e $a_i =$ "b" **ripeti**
15: $cb = cb + 1$, $i = i + 1$
16: **fine-ciclo**
17: **se** cb è dispari **allora**
18: $ok = 0$
19: **fine-condizione**

20: **fine-condizione**
21: **fine-condizione**
22: **fine-ciclo**
23: **se** $ok = 1$ **allora**
24: la stringa rispetta la regola
25: **altrimenti**
26: la stringa NON rispetta la regola
27: **fine-condizione**
28: stop

Codifica in linguaggio C

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void) {
  char a[100];
  int i, ok, ca, cb;
  printf("Inserisci una stringa: ");
  scanf("%s", a);
  i = 0;
  ok = 1;
  while (i < strlen(a) && ok == 1) {
    ca = 0;
    cb = 0;
    if (a[i] != 'a' && a[i] != 'b') {
      ok = 0;
    } else {
      while (i<strlen(a) && a[i] == 'a') {
        ca++;
        i++;
      }
      if (ca % 2 == 1) {
        ok = 0;
      } else {
        while (i < strlen(a) && a[i] == 'b') {
          cb++;
          i++;
        }
        if (cb % 2 == 1) {
          ok = 0;
        }
      }
    }
  }
  if (ok == 1)
    printf("La stringa rispetta la regola.\n");
  else
    printf("La stringa NON rispetta la regola.\n");
  return(1);
}
```

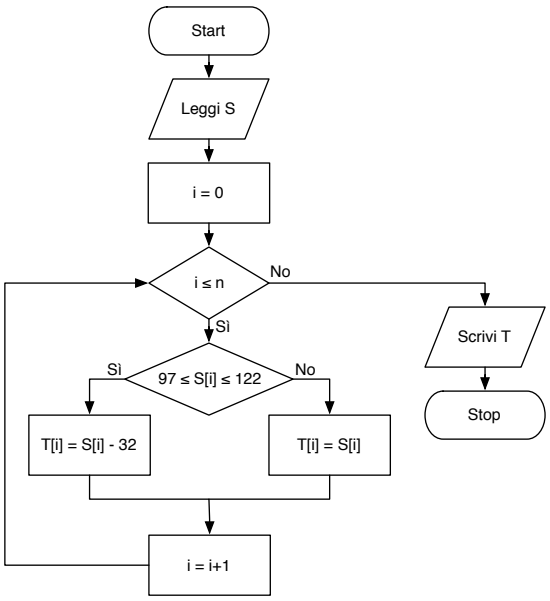
Esercizio n. 3

Letta una stringa s, costruire la stringa t con la stessa paola di s, ma con caratteri maiuscoli.

Pseudo-codifica dell'algoritmo

1: leggi la stringa s con n caratteri
2: **per** $i = 0, 1, 2, \ldots, n-1$ **ripeti**
3: **se** s_i è una lettera dell'alfabeto minuscola **allora**
4: $t_i = s_i - 32$
5: **altrimenti**
6: $t_i = s_i$
7: **fine-condizione**
8: **fine-ciclo**
9: stop

Diagramma di flusso



Codifica in linguaggio C

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void) {
  char s[100], t[100];
  int i;

  printf("Inserisci una stringa: ");
  scanf("%s", s);
  for (i=0; i<=strlen(s); i++) {
    if (s[i] >= 97 && s[i] <= 122)
      t[i] = s[i] - 32;
    else
      t[i] = s[i];
  }
  printf("Stringa maiuscola: %s\n", t);
  return(1);
}
```