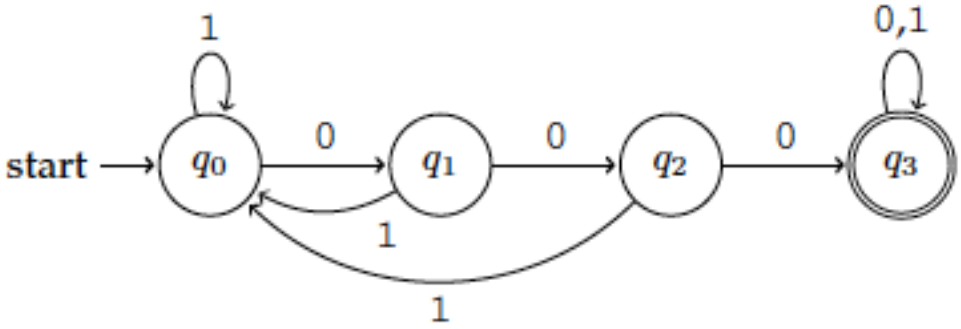
1. Implementazione di un DFA in Java



Esercizio 1.1

- Lo scopo di questo esercizio e dei successivi è l'implementazione di un metodo Java che sia in grado di discriminare le stringhe del linguaggio riconosciuto da un automa a stati finiti deterministico (DFA) dato.
- Il primo automa che prendiamo in considerazione, mostrato in Figura 1, è definito sull'alfabeto {0, 1} e riconosce le stringhe in cui compaiono almeno 3 zeri consecutivi

Esercizio 1.1 – Progettazione Automa

Metodo:
1) Progettazione dell'automa
2) Implementazione in Java
3) Test su esempi di input

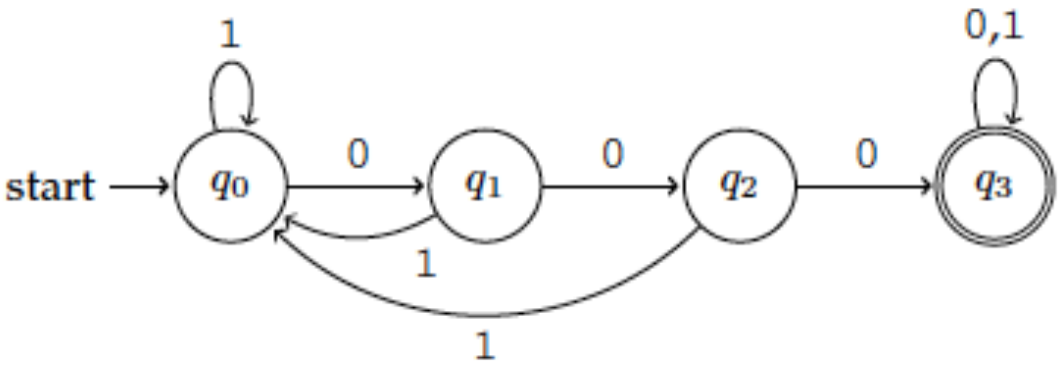


Figura 1: DFA che riconosce stringhe con 3 zeri consecutivi.

Esercizio 1.1 - Implementazione

Metodo:
1) Progettazione dell'automa
2) Implementazione in Java
3) Test su esempi di input

- L'automa è implementato mediante il metodo scan:
  - Prende in input una stringa s
  - Restituisce un valore booleano che indica se la stringa appartiene o meno al linguaggio riconosciuto dall'automa.
- Lo stato dell'automa è rappresentato per mezzo di una variabile intera state
- La variabile i contiene l'indice del prossimo carattere della stringa s da analizzare.
- Inizializzate le variabili

```
public class TreZeri
{
    public static boolean scan(String s)
    {
        int state = 0;
        int i = 0;
```

Esercizio 1.1 - Implementazione

```java
public class TreZeri
{
    public static boolean scan(String s)
    {
        int state = 0;
        int i = 0;

        while (state >= 0 && i < s.length()) {
            final char ch = s.charAt(i++);

            switch (state) {
            case 0:
                if (ch == '0')
                    state = 1;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 1:
                if (ch == '0')
                    state = 2;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 2:
                if (ch == '0')
                    state = 3;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 3:
                if (ch == '0' || ch == '1')
                    state = 3;
                else
                    state = -1;
                break;
            }
        }
        return state == 3;
    }

    public static void main(String[] args)
    {
        System.out.println(scan(args[0]) ? "OK" : "NOPE");
    }
}
```

Figura 2: Implementazione Java del DFA di Figura 1.

Il corpo principale del metodo è un ciclo che, analizzando il contenuto della stringa s un carattere alla volta, effettua un cambiamento dello stato dell'automa secondo la sua funzione di transizione.

Esercizio 1.1 - Implementazione

```java
public class TreZeri
{
    public static boolean scan(String s)
    {
        int state = 0;
        int i = 0;

        while (state >= 0 && i < s.length()) {
            final char ch = s.charAt(i++);

            switch (state) {
            case 0:
                if (ch == '0')
                    state = 1;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 1:
                if (ch == '0')
                    state = 2;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 2:
                if (ch == '0')
                    state = 3;
                else if (ch == '1')
                    state = 0;
                else
                    state = -1;
                break;

            case 3:
                if (ch == '0' || ch == '1')
                    state = 3;
                else
                    state = -1;
                break;
            }
        }
        return state == 3;
    }

    public static void main(String[] args)
    {
        System.out.println(scan(args[0]) ? "OK" : "NOPE");
    }
}
```

Figura 2: Implementazione Java del DFA di Figura 1.

Osserva: viene assegnato il valore **-1** alla variabile state se viene incontrato un simbolo diverso da 0 e 1.

Tale valore non è uno stato valido, ma rappresenta una condizione di errore irrecuperabile.

Esercizio 1.1
Implementazione e Test

Metodo:
1) Progettazione dell'automa
2) Implementazione in Java
3) Test su esempi di input

- Scaricare il codice TreZeri.java dalla pagina del corso (fig. 2 del documento)
- Compilarlo e testarlo su un insieme significativo di stringhe:
  - "010101 ", "1100011001 ", "10214 ", etc.

Esercizio 1.1 bis

Metodo:
1) Progettazione dell'automa
2) Implementazione in Java
3) Test su esempi di input

- Come deve essere modificato il DFA in Figura 1 per riconoscere il linguaggio complementare, ovvero il linguaggio delle stringhe di 0 e 1 che non contengono 3 zeri consecutivi?
- Progettare e implementare il DFA modificato, e testare il suo funzionamento

Esercizio 1.2

- Progettare e implementare un DFA che riconosca il linguaggio degli identificatori in un linguaggio in stile Java
- un identificatore è una sequenza non vuota di lettere, numeri, ed il simbolo di "underscore" _ che:
  - non comincia con un numero
  - non può essere composto solo dal simbolo _
- Compilare e testare il suo funzionamento su un insieme significativo di esempi.

Esercizio 1.2

Metodo:

1) Progettazione dell'automa
2) Implementazione in Java
3) Test su esempi di input

Esercizio 1.3

- Progettare e implementare un DFA che riconosca il linguaggio di stringhe che contengono un numero di matricola seguito (subito) da un cognome, dove la combinazione di matricola e cognome corrisponde a studenti del turno 2 o del turno 3 del laboratorio di LFT.
- Regole per suddivisione di studenti in turni:
  - Turno 1: cognomi la cui iniziale è compresa tra A e K, e il numero di matricola è dispari;
  - Turno 2: cognomi la cui iniziale è compresa tra A e K, e il numero di matricola è pari;
  - Turno 3: cognomi la cui iniziale è compresa tra L e Z, e il numero di matricola è dispari;
  - Turno 4: cognomi la cui iniziale è compresa tra L e Z, e il numero di matricola è pari.
- Esempio:
  - "123456Bianchi " e "654321Rossi " sono stringhe del linguaggio,
  - "654321Bianchi " e "123456Rossi " non sono stringhe del linguaggio.
- Nel contesto di questo esercizio, un numero di matricola non ha un numero prestabilito di cifre ma deve essere composto di almeno una cifra.
- Un cognome deve essere composto di almeno una lettera. Quindi l'automa deve accettare le stringhe "2Bianchi " e "122B " ma non "654321 " e "Rossi ".
- Assicurarsi che il DFA sia minimo.

Esercizio 1.3

Metodo:

1) Progettazione dell'automa
2) Implementazione in Java
3) Test su esempi di input

Esercizio 1.3 bis

- Progettare e implementare un DFA che riconosca il linguaggio di stringhe che contengono un numero di matricola seguito (subito) da un cognome, dove la combinazione di matricola e cognome corrisponde a studenti del turno 1 o del turno 4 del laboratorio di LFT.
- Regole per suddivisione di studenti in turni:
  - Turno 1: cognomi la cui iniziale è compresa tra A e K, e il numero di matricola è dispari;
  - Turno 2: cognomi la cui iniziale è compresa tra A e K, e il numero di matricola è pari;
  - Turno 3: cognomi la cui iniziale è compresa tra L e Z, e il numero di matricola è dispari;
  - Turno 4: cognomi la cui iniziale è compresa tra L e Z, e il numero di matricola è pari.
- Esempio:
  - “123456Bianchi ” e “654321Rossi ” non sono stringhe del linguaggio,
  - “654321Bianchi ” e “123456Rossi ” sono stringhe del linguaggio.
- Nel contesto di questo esercizio, un numero di matricola non ha un numero prestabilito di cifre ma deve essere composto di almeno una cifra.
- Un cognome deve essere composto di almeno una lettera. Quindi l’automa deve accettare le stringhe “2Bianchi ” e “122B ” ma non “654321 ” e “Rossi ”.
- Assicurarsi che il DFA sia minimo.

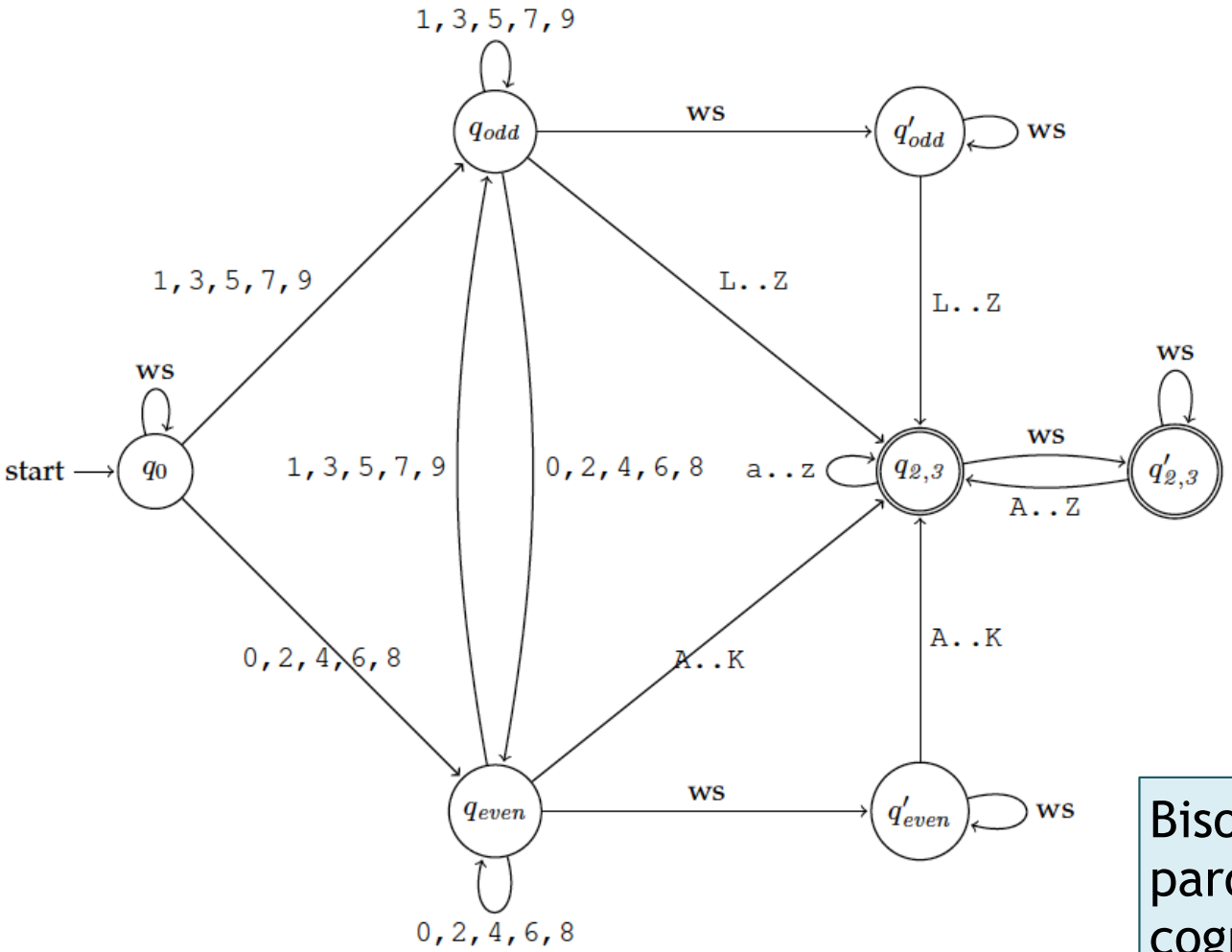
Esercizio 1.4

- Modificare l'automa dell'esercizio precedente in modo che riconosca le combinazioni di matricola e cognome di studenti del turno 2 o del turno 3 del laboratorio, dove il numero di matricola e il cognome:
- possono essere separati da una sequenza di spazi
- possono essere precedute e/o seguite da sequenze eventualmente vuote di spazi.
- Per esempio, l'automa
  - deve accettare la stringa "654321 Rossi " e " 123456 Bianchi " (dove, nel secondo esempio, ci sono spazi prima del primo carattere e dopo l'ultimo carattere)
  - ma non "1234 56Bianchi " e "123456Bia nchi ".

1) Provate prima a pensare a una soluzione senza considerare cognomi composti
2) Poi progettare una variante dell'automa in cui i cognomi composti (con un numero arbitrario di parti) possono essere accettati:
   - Per esempio, la stringa "123456De Gasperi " deve essere accettata.
   - Modificare l'implementazione Java dell'automa di conseguenza.
3) Modificate l'automa dell'esercizio in modo che riconosca le combinazioni di matricola e cognome di studenti del turno 1 o del turno 4 del laboratorio (per il resto, per quanto riguarda il white space stessi vincoli sull'accettazione delle stringhe)

Esercizio 1.4 T2/T3

variante con cognomi composti ma senza considerare apostrofi e trattini



Bisogna che ogni parola del cognome composto cominci con una maiuscola

Esercizio 1.5

- Esercizio 1.5. Progettare e implementare un DFA che, come in Esercizio 1.3, riconosca il linguaggio di stringhe che contengono matricola e cognome di studenti del turno 2 o del turno 3 del laboratorio, ma in cui il cognome precede il numero di matricola
  - in altre parole, le posizioni del cognome e matricola sono scambiate rispetto all'Esercizio 1.3).
- Assicurarsi che il DFA sia minimo.

Esercizio 1.6

- Esercizio 1.6. Progettare e implementare un DFA che riconosca il linguaggio dei numeri binari (stringhe di 0 e 1) il cui valore è multiplo di 3.
  - Per esempio, “110” e “1001” sono stringhe del linguaggio (rappresentano rispettivamente i numeri 6 e 9), mentre “10” e “111” no (rappresentano rispettivamente i numeri 2 e 7).
- Suggerimento: usare tre stati per rappresentare il resto della divisione per 3 del numero.