علیرضا نعمتی ۹۹۳۱۰۸۱

طبق صورت سوال باید ماشین تورینگی برای تابع مقابل طراحی کنیم:

$$f(x) = \begin{cases} \frac{x}{2} \ .if\ x\ is\ even \\ \frac{x+1}{2}\ if\ x\ is\ odd \end{cases}$$

در این گزارش به توضیح کد میپردازیم سپس به سراغ توضیحاتی درباره نحوه طراحی ماشین تورینگ می رویم.

نحوه ورودی دادن:

تنها نیاز است که عدد مورد نظر خود را به شکل باینری وارد کنید.

به عنوان مثال: ۱۰۱۰۰

توضیحات بیشتر:

در ابتدا و انتهای رشته نیاز است که blank را قرار دهیم که در این کد با "-" نشان دادیم که این کار را هنگام ورودی گرفتن در کد انجام میشود.

```
System.out.println("Please enter a binary number: ");

//get our tape input.
Scanner sc = new Scanner(System.in);
String inputTape = sc.nextLine();

String[] strSplit = inputTape.split("");
ArrayList<String> tape = new ArrayList<>();

for (int i = 0; i < strSplit.length + 2; i++) {

    //insert blank character at the beginning.
    if (i == 0) {
        tape.add("-");
        continue;
    }

    //insert blank character at the end.
    if (i == strSplit.length + 1) {
        tape.add("-");
        break;
    }

    //add input to tape.
    tape.add(strSplit[i - 1]);
}
```

توضیحات کد

ابتدا متغیر هایی را که برای طراحی تورینگ ماشین مان داریم را بررسی میکنیم که توضیحاتش بدین شکل است:

1 - State: برای نگه داشتن استیت فعلی تورینگ ماشین.
2 - Tape: که نوار ورودی ما است که ابتدا به شکل رشته از کاربر دریافت کرده ایم و داخل لیستی آن را ذخیره کرده ایم.
3 - Tapehead: head نوار در ماشین تورینگ مثل پوینتری است که روی لیست نوار ما حرکت میکند. Tapeheadدر واقع ایندکسی از لیست نوار است که در حال حاضر head روی آن قرار دارد.

بعد از تعریف متغیر ها آن را با مقادیر ابتدایی مقدار دهی میکنیم.

```java
/**
 * The type Turing machine class.
 */
public class TuringMachine {
    /**
     * The current State.
     */
    String state;

    /**
     * The Tape.
     */
    ArrayList<String> tape;

    /**
     * The Tape head(pointer).
     */
    int tapeHead;

    /**
     * Instantiates a new Turing machine.
     *
     * @param tape the tape
     */
    public TuringMachine(ArrayList<String> tape) {
        this.tape = tape;
        this.state = "q0";
        this.tapeHead = 0;
    }
```

اکنون به توضیح دو تابع پر اهمیت updateMachine و moveHead میپردازیم.

تابع updateMachine: سه پارامتر میگیرد. پارامتر کاراکتر input را میگیرد و با کاراکتری که tape head به آن اشاره میکند مقایسه میکند که یکی باشد که اگر یکی بود باید با پارامتر تابع newInput که کاراکتری که تابع تورینگ میگوید جایگزین شود و در نهایت اخرین پارامتر ما استیت یا وضعیت جدیدی است که باید به آن برویم.

تابع moveHead: جهت تابع را میگیرد (با یک بولین (که پارامتر تابع است مشخص میکند) که آیا باید هد نوار به راست برود یا نه) سپس با توجه به جهت مورد نظر ایندکسی که هد نوار به آن اشاره میکند کم یا زیاد میشود.

```java
/**
 * Updates machine with new input and new state.
 *
 * @param input    the input
 * @param newInput the new input
 * @param newState the new state
 * @return the boolean
 */
public boolean updateMachine(String input, String newInput, String newState) {
    if (Objects.equals(tape.get(getTapeHead()), input)) {
        tape.set(this.tapeHead, newInput);
        setState(newState);
        return true;
    }
    return false;
}

/**
 * Move head to left or right direction.
 *
 * @param goRight the go right
 */
public void moveHead(boolean goRight) {
    if (goRight) {
        this.tapeHead++;
    } else {
        this.tapeHead--;
    }
}
```

پس از نوشتن این تابع تنها نیاز است که در یک حلقه چک کنیم در چه استیتی هستیم و با توجه به استیتی که هستیم و تابع انتقالی که در شکل استیت ماشینمان تعریف کردیم، استیت فعلی و بعدی و همچینین مقداری که از نوار بخوانیم و با آن جایگزین کنیم را مشخص کنیم و به تابع updateMachine این اطلاعات را بدیم و جهت هم به تابع moveHead برای جا به جایی درست هد نوار منتقل کنیم.

یک قسمت از کد طبق طراحی استیت ماشین:

```
case "q2":
   if (updateMachine("۰", "۰", "q3")) {
      moveHead(true);
   }
   if (updateMachine("۱", "۰", "q6")) {
      moveHead(false);
   }
   break;
```

کد بالا معادل تابع انتقال زیر است:

$\delta(q2, ۰) = (q3, ۰, R)$
$\delta(q2, ۱) = (q6, ۰, L)$

و به همین روال با توجه به تابع انتقال دیتا را وارد کد میکنیم.
باید توجه داشته باشیم که این چک کردن کیس ها داخل کد تا زمانی ادامه پیدا میکند به یکی از حالت های acceptance برسیم که در این حالت از حلقه بیرون می آییم و در نهایت برنامه به پایان میرسد و جواب مورد نظر ما چاپ میشود پس از اعمال تغییرات لازم.

```
//check machine acceptance requirement
if (state.equals("q5") || state.equals("q9")) {
    System.out.println("accepted with status: ");
    printStatus(tape, state, tapeHead);

    return;
}
```

چند تست کیس برای ارزیابی کد

ورودی۱۰۱۰۰ است که وقتی بر دو تقسیم شود برابر ۱۰۱۰ میشود.

```
10100
tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q0
tape head is on index: 0


tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q1
tape head is on index: 1


tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q1
tape head is on index: 3


tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q1
tape head is on index: 5


tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q2
tape head is on index: 5


tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q3
tape head is on index: 6


tape: [-, 1, 0, 1, 0, 0, -]
we are at state: q4
tape head is on index: 5
```

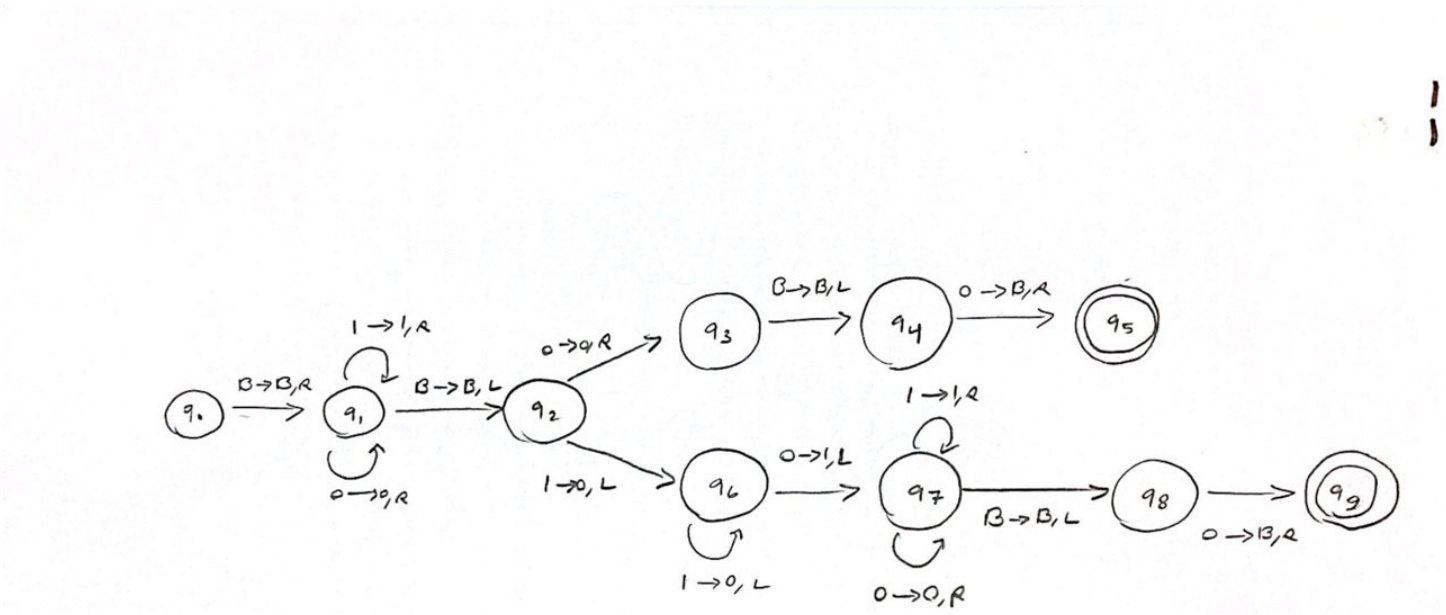
```
accepted with status:
tape: [-, 1, 0, 1, 0, -, -]
we are at state: q5
tape head is on index: 6
```

ورودی ۱۰۰۱ است که وقتی با یک جمع کنیم و بر دو تقسیم کنیم باید ۱۰۱ شود.

خروجی هم ۱۰۱ نشان میدهد که صحت کد را نشان میدهد.

```
Please enter a binary number:
1001
tape: [-, 1, 0, 0, 1, -]
we are at state: q0
tape head is on index: 0


tape: [-, 1, 0, 0, 1, -]
we are at state: q1
tape head is on index: 1


tape: [-, 1, 0, 0, 1, -]
we are at state: q1
tape head is on index: 3


tape: [-, 1, 0, 0, 1, -]
we are at state: q1
tape head is on index: 4


tape: [-, 1, 0, 0, 1, -]
we are at state: q2
tape head is on index: 4


tape: [-, 1, 0, 0, 0, -]
we are at state: q6
tape head is on index: 3


tape: [-, 1, 0, 1, 0, -]
we are at state: q7
tape head is on index: 2
tape: [-, 1, 0, 1, 0, -]
we are at state: q7
tape head is on index: 3


tape: [-, 1, 0, 1, 0, -]
we are at state: q8
tape head is on index: 4


accepted with status:
tape: [-, 1, 0, 1, -, -]
we are at state: q9
tape head is on index: 5
```

طراحی استیت ماشین



توضیحات راجع به نحوه طراحی:

از استیت اولیه یا q0 که شروع میکنیم، وقتی که به استیت q2 رسیدیم در راست ترین رقم هستیم و به این شکل با تشخیص ۰ یا ۱ بودن این عدد به زوج یا فرد بودن عدد پی میبریم.

در صورت زوج بودن نیاز است که به اولین رقم سمت راست برویم و آن رقم را دور بیندازیم (روش تقسیم بر دو باینری)

در صورت فرد بودن باید ابتدا عدد را با ۱ جمع کنیم، که این کار با به این شکل انجام میدهیم که ۱ ها را ۰ میکنیم از سمت راست تا جایی که به صفر برسیم و آن را برابر یک بگذاریم. سپس مثل حالت زوج عملیات تقسیم بر دو را انجام میدهیم.

مقدار دهی کد با توجه به استیت ماشین

```
while (true) {
    printStatus(tape, state, tapeHead);

    //apply switch case for different machine states and input and move head.
    switch (state) {
        case "q0":
            if (updateMachine("-", "-", "q1"))
                moveHead(true);
            break;

        case "q1":
            if (updateMachine("1", "1", "q1")) {
                moveHead(true);
            }
            if (updateMachine("0", "0", "q1")) {
                moveHead(true);
            }
            if (updateMachine("-", "-", "q2")) {
                moveHead(false);
            }
            break;

        case "q2":
            if (updateMachine("0", "0", "q3")) {
                moveHead(true);
            }
            if (updateMachine("1", "0", "q6")) {
                moveHead(false);
            }
            break;

        case "q3":
            if (updateMachine("-", "-", "q4")) {
                moveHead(false);
            }
            break;

        case "q4":
            if (updateMachine("0", "-", "q5")) {
                moveHead(true);
            }
            break;

        case "q6":
            if (updateMachine("1", "0", "q6")) {
                moveHead(false);
            }
            if (updateMachine("0", "1", "q7")) {
                moveHead(false);
            }
            break;

        case "q7":
            if (updateMachine("1", "1", "q7")) {
                moveHead(true);
            }
```

```
            if (updateMachine("0", "0", "q7")) {
                moveHead(true);
            }
            if (updateMachine("-", "-", "q8")) {
                moveHead(false);
            }
            break;

        case "q8":
            if (updateMachine("0", "-", "q9")) {
                moveHead(true);
            }
            break;
    }

    //check machine acceptance requirement
    if (state.equals("q5") || state.equals("q9")) {
        System.out.println("accepted with status: ");
        printStatus(tape, state, tapeHead);

        return;
    }
}
```