پروژه درس نظریه زبانها و ماشین ها

علی فدائی منش – ۹۸۳۱۱۳۰

هدف پروژه: طراحی یک ماشین تورینگ برای محاسبه تابع x^5

این ماشین می تواند به این صورت که عرض می کنم پیاده سازی شود:

نحوه پیاده سازی:

به این صورت که ما از چپ ۵ را به صورت unary می نویسیم. سپس یک Blank قرار می دهیم. سپس x که عدد ورودی است را به صورت unary می نویسیم و سپس Blank قرار می دهیم تا بین ۵ و x فاصله بیافتد. سپس عدد ۱ را می نویسیم و سپس Blank. در حال حاضر سه عدد به تفکیک وجود دارند که با Blank از هم جدا شده اند.

عدد اول از سمت راست در ابتدا ۱ است و در هر مرحله در عدد وسطی که همان X است ضرب می شود و حاصل اش در سمت راستش چاپ می شود، سپس عدد سمت راست بجایش L قرار داده می شود تا در دفعات بعدی از آن عبور کنیم.

باید به تعداد سمت چپ(۵ بار)، عدد وسطی را در راستی ضرب کنیم. در نهایت x^5 را به صورت unary در سمت راست خواهیم داشت.

```
BX11111B11C1B
BX1111BB11CJ11B
BX111BBB11CJJ1111B
BX11BBBB11CJJJJ11111111B
BX1BBBBB11CJJJJJJJJJJJJ1111111111111111B
BXBBBBBB11CJJJJJJJJJJJJJJJJJJJJJJJJJJJJ11111111111111111111111111111111B
```

برای پیاده سازی توان از ماشین ضرب کننده x * y استفاده می کنیم.

بررسی کد:

- **پیاده سازی ماشین ها:**
  یک کلاس TuringMachine تعریف کردم که تعدادی متود و فیلد دارد:
  - toRight(): اشاره گر به سمت راست می رود
  - toLeft(): اشاره گر به سمت چپ می رود
  - run(): ماشین شروع به کار می کند و نتیجه خروجی را چاپ می کند (نوار را)
  - next(): ماشین حالت بندی یک مرحله به جلو می رود (خواندن، تصمیم گیری، نوشتن و حرکت)

- write(): یک کاراکتر روی نوار می نویسد
- setState(): تغییر حالت میدهد
- read(): کاراکتر روی نوار را می خواند (اشاره گر به آن اشاره می کند)
- Index: اشاره گر روی نوار (int)
- Tape: یک لیست از کاراکتر های روی نوار (list)
- State: وضعیت فعلی ماشین (string)
- Finish: کار ماشین تمام شده است (Bool)

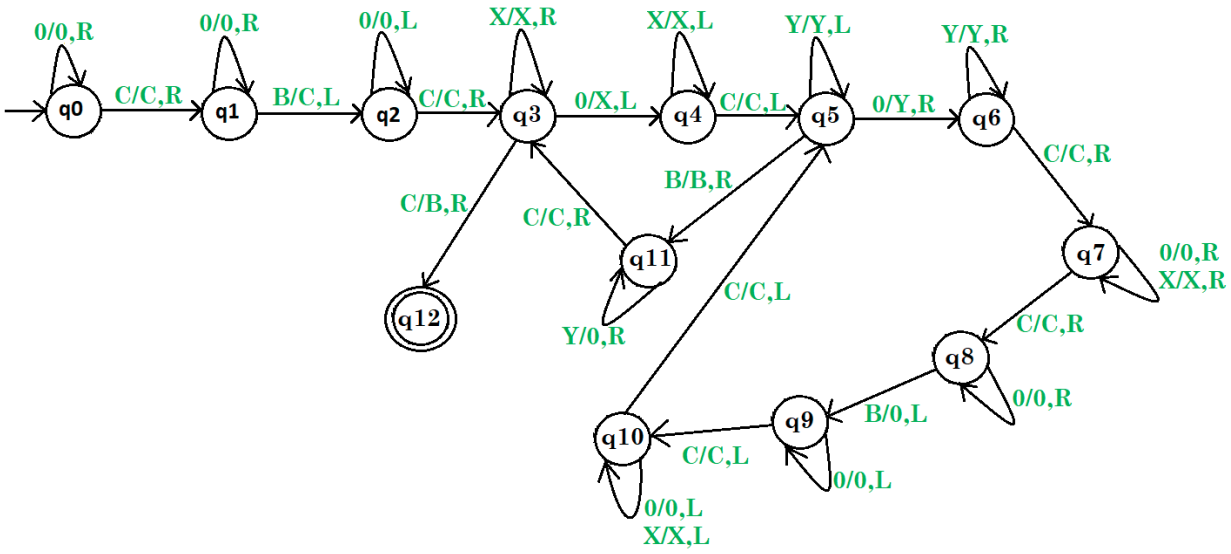
دو ماشین دیگر بر همین اساس تعریف کردم یکی ماشین **MultiplingMachine** که عمل ضرب انجام می دهد و دیگری **X5Machine** که ماشین تابع x^5 است. هر دوی این کلاس ها از **TuringMachine** ارث می برند. و تابع next آنها override می شود. که متغییر های فعلی را می گیرد و یک مرحله به جلو می رود. در واقع هر ماشین تورینگ، توابع انتقالش در همین متود **next** تعریف می شود.

ماشین ضرب کننده در یکی از حالات ماشین تابع x^5 استفاده می شود به این صورت که هنگام آماده بودن operand ها، این ماشین روی نوار صدا زده می شود تا حاصل را محاسبه کند.

پیاده سازی ماشین ضرب کننده (MultiplingMachine):

این ماشین دو عدد را می گیرد به صورت Unary و حاصل را بعد از آنها روی نوار چاپ می کند:

```
Input: (B<operand1>C<operand2>B)
B11C1111BB
Output: <B<operand1>C<J><result>B>
B11CJJJJ11111111BB
```



پیاده سازی ماشین تابع X^5 (X5Machine):

این ماشین ابتدا ۱ ها را رد می کند و اولین B را پیدا می کند. سپس به چپ رفته و اولین ۱ را پیدا کرده و x می کند سپس به راست بر می گردد و ماشین ضرب کننده را صدا می زند. تا عملیات ضرب انجام شود و حاصل در سمت راست ترین چاپ شود. سپس دوباره به چپ برمیگردد و اولین ۱ را پیدا می کند و x میگذارد سپس دوباره به راست B بر میگردد تا ضرب کند و همین طور ۵ بار این کار را انجام میدهد تا x^5 را حساب کند.

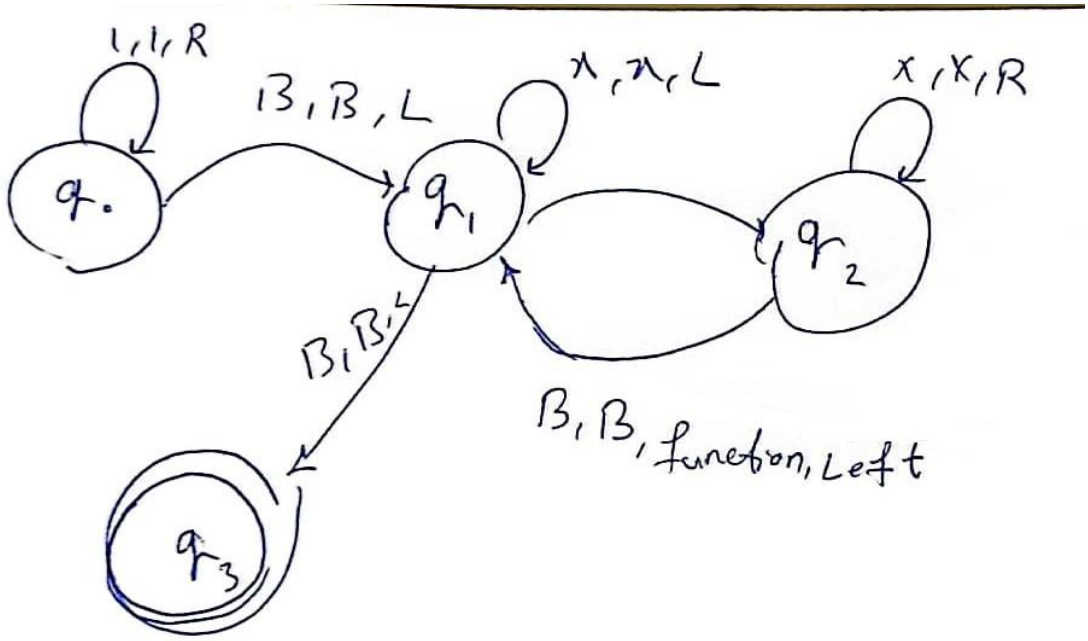
نمونه ورودی خروجی:

```
Input: (B<11111>B<input>C<1>BB)

B11111B11C1BB

Output: <B<operand1>C<J><result>BB>

BXXXXXB11CJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ1111111111111111111111111111111111
1BB
```



نحوه نمایش خروجی ماشین:

ماشین های تورینگ من یک متود print دارند که نوار را به صورت یک رشته چاپ می کند. و به طور خاص کاراکتر فعلی ای که اشاره گر به آن اشاره می کند را قرمز می کند. تا دیدن آن ساده شود. بعلاوه وضعیت state فعلی را به همراه مقدار کاراکتر فعلی را نیز در هر مرحله چاپ میکند.

```
value:  1 , state:  q0
B11111B11C1BB
```

بین هر بار اجرای متود next یک delay تعریف کردم که دیدن حالات انتقال به سادگی و زیبایی انجام شود.

توابع انتقال:

توابع انتقال را در متود next به صورت if else if تعریف کرده ام تا حالت متناسب را پیدا و عمل کند. در هر حالت مقداری روی نوار چاپ می شود، یک خانه به جلو یا عقب حرکتی می کنیم و state را تغییر می دهیم. درست مانند یک ماشین تورینگ

مثال:

```
if value == '1' and state == 'q0':
      self.write('1')
      self.toRight()
      self.setState('q0')
elif value == 'C' and state == 'q0':
      self.write('C')
      self.toRight()
```

کل کد پروژه:

```
from termcolor import colored
import os
import time

STRING = "B11111B11C1BB"
DELAY = .5
def clear(): return os.system('cls')


class TuringMachine:
    def __init__(self, s, initialIndex) -> None:
        self.tape = s
        self.index = initialIndex
        self.state = 'q0'
```

```
        self.finish = False

    def toRight(self):
        self.index += 1

    def toLeft(self):
        self.index += -1
        if self.index < 0:
            raise Exception("saa")

    def next(self):
        pass

    def read(self):
        value = self.s[self.index]
        return value

    def write(self, x):
        self.tape[self.index] = x

    def setState(self, x):
        self.state = x

    def run(self):
        while(not self.finish):
            self.next()
            time.sleep(DELAY)

    def print(self):
        clear()
        s = self.tape
        value = self.tape[self.index]
        state = self.state
        s = ''.join([str(elem) for elem in s])
        print('value: ', value, ', state: ', state)
        print(s[:self.index] + colored(s[self.index], 'red') +
s[self.index + 1:])


class MultiplingMachine(TuringMachine):
    def __init__(self, s, ins) -> None:
        super().__init__(s, ins)
```

```
    def next(self):
        value = self.tape[self.index]
        state = self.state
        if DELAY:
            self.print()

        if value == '1' and state == 'q0':
            self.write('1')
            self.toRight()
            self.setState('q0')
        elif value == 'C' and state == 'q0':
            self.write('C')
            self.toRight()
            self.setState('q1')
        elif value == 'J' and state == 'q1':
            self.toRight()
        elif (value == '1') and state == 'q1':
            self.write('1')
            self.toRight()
            self.setState('q1')
        elif value == 'B' and state == 'q1':
            self.write('C')
            self.toLeft()
            self.setState('q2')

        elif value == '1' and state == 'q2':
            self.write('1')
            self.toLeft()
            self.setState('q2')
        elif value == 'J' and state == 'q2':
            self.toLeft()

        elif value == 'C' and state == 'q2':
            self.write('C')
            self.toRight()
            self.setState('q3')
        elif value == 'X' and state == 'q3':
            self.write('X')
            self.toRight()
            self.setState('q3')
        elif value == 'J' and state == 'q3':
            self.toRight()
```

```
        elif value == '1' and state == 'q3':
            self.write('X')
            self.toLeft()
            self.setState('q4')
        elif value == 'J' and state == 'q3':
            self.toLeft()
        elif value == 'X' and state == 'q4':
            self.write('X')
            self.toLeft()
            self.setState('q4')
        elif value == 'J' and state == 'q4':
            self.toLeft()
        elif value == 'C' and state == 'q4':
            self.write('C')
            self.toLeft()
            self.setState('q5')
        elif value == 'J' and state == 'q4':
            self.toLeft()

        elif value == 'Y' and state == 'q5':
            self.write('Y')
            self.toLeft()
            self.setState('q5')
        elif value == 'J' and state == 'q5':
            self.toLeft()
        elif value == '1' and state == 'q5':
            self.write('Y')
            self.toRight()
            self.setState('q6')
        elif value == 'J' and state == 'q5':
            self.toRight()
        elif value == 'Y' and state == 'q6':
            self.write('Y')
            self.toRight()
            self.setState('q6')
        elif value == 'J' and state == 'q6':
            self.toRight()
        elif value == 'C' and state == 'q6':
            self.write('C')
            self.toRight()
            self.setState('q7')
        elif value == 'J' and state == 'q6':
            self.toRight()
```

```
        elif value == '1' and state == 'q7':
            self.write('1')
            self.toRight()
            self.setState('q7')
        elif value == 'J' and state == 'q7':
            self.toRight()

        elif value == 'X' and state == 'q7':
            self.write('X')
            self.toRight()
            self.setState('q7')

        elif value == 'C' and state == 'q7':
            self.write('C')
            self.toRight()
            self.setState('q8')
        elif value == '1' and state == 'q8':
            self.write('1')
            self.toRight()
            self.setState('q8')
        elif value == 'J' and state == 'q8':
            self.toRight()

        elif value == 'B' and state == 'q8':
            self.write('1')
            self.tape.append('B')
            self.toLeft()
            self.setState('q9')
        elif value == '1' and state == 'q9':
            self.write('1')
            self.toLeft()
            self.setState('q9')
        elif value == 'J' and state == 'q9':
            self.toLeft()

        elif value == 'C' and state == 'q9':
            self.write('C')
            self.toLeft()
            self.setState('q10')
        elif value == '1' and state == 'q10':
            self.write('1')
            self.toLeft()
```

```
            self.setState('q10')
        elif value == 'J' and state == 'q10':
            self.toLeft()

        elif value == 'X' and state == 'q10':
            self.write('X')
            self.toLeft()
            self.setState('q10')
        elif value == 'C' and state == 'q10':
            self.write('C')
            self.toLeft()
            self.setState('q5')
        elif value == 'B' and state == 'q5':
            self.write('B')
            self.toRight()
            self.setState('q11')
        elif value == 'Y' and state == 'q11':
            self.write('1')
            self.toRight()
            self.setState('q11')
        elif value == 'J' and state == 'q11':
            self.toRight()

        elif value == 'C' and state == 'q11':
            self.write('C')
            self.toRight()
            self.setState('q3')
        elif value == 'C' and state == 'q3':
            self.write('J')
            self.toLeft()
            self.setState('q12')
        elif value == 'X' and state == 'q12':
            self.write('J')
            self.toLeft()
            self.setState('q12')
        elif value == 'J' and state == 'q12':
            self.toLeft()

        elif value == 'C' and state == 'q12':
            self.setState('q13')
            self.finish = True
            self.tape.append('B')
            self.print()
```

```
class X5Machine(TuringMachine):
    def __init__(self, s, ins) -> None:
        super().__init__(s, ins)

    def next(self):
        state = self.state
        value = self.tape[self.index]

        if DELAY:
            self.print()
        if value == '1' and state == 'q0':
            self.write('1')
            self.toRight()
            self.setState('q0')
        elif value == 'J' and state == 'q0':
            self.toRight()

        elif value == 'B' and state == 'q0':
            self.write('B')
            self.toLeft()
            self.setState('q1')
        elif value == 'X' and state == 'q1':
            self.write('X')
            self.toLeft()
            self.setState('q1')
        elif value == 'J' and state == 'q1':
            self.toLeft()

        elif value == '1' and state == 'q1':
            self.write('X')
            self.toRight()
            self.setState('q2')
        elif value == 'X' and state == 'q2':
            self.write('X')
            self.toRight()
            self.setState('q2')
        elif value == 'J' and state == 'q2':
            self.toRight()

        elif value == 'B' and state == 'q2':
            self.write('B')
```

```
            MultiplingMachine(self.tape, self.index + 1).run()
            self.toLeft()
            self.setState('q1')
        elif value == 'B' and state == 'q1':
            self.write('B')
            self.toRight()
            self.setState('q4')
            self.finish = True
            self.print()
            print('finish')


11111111111111111111111111111

s = list(STRING)
X5Machine(s, 1).run()
```