**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Информатика»**

**Тема: Машина Тьюринга**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Колесниченко М.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2023

## Цель работы

Изучение принципов работы машины Тьюринга и написание программы, имитирующей такую машину с помощью языка Python

## Задание

Вариант 1.

На вход программе подается строка неизвестной длины. Каждый элемент является значением в ячейке памяти ленты Машины Тьюринга.

На ленте находится последовательность латинских букв из алфавита {a, b, c}.

Напишите программу, которая удаляет в исходной строке два символа, следующих за первым встретившимся символом ‘b’. Если первый встретившийся символ ‘b’ – последний в строке, то удалить его. Если первый встретившийся символ ‘b’ – предпоследний в строке, то удалить один символ, следующий за ним, т. е. последний в строке. Если в строке символ ‘b’ отсутствует, то удалить самый первый символ строки. После удаления в строке не должно оставаться пробелов и пустых мест!

Указатель на текущее состояние Машины Тьюринга изначально находится слева от строки с символами (но не на первом ее символе). По обе стороны от строки находятся пробелы.

Алфавит:

* a
* b
* c
* " " (пробел)

Соглашения:

1. Направление движения автомата может быть одно из R (направо), L

(налево), N (неподвижно).

2. Гарантируется, что длинна строки не менее 5 символов и не более 13.

3. В середине строки не могут встретиться пробелы.

4. При удалении или вставке символов направление сдвигов подстрок не

принципиально (т. е. результат работы алгоритма может быть сдвинут по ленте в любую ее сторону на любое число символов).

5. Курсор по окончании работы алгоритма может находиться на любом

символе.

Ваша программа должна вывести полученную ленту после завершения работы.

В отчете предоставьте таблицу состояний. Отдельно кратко опишите каждое состояние, например: q1 - начальное состояние, которое необходимо, чтобы найти первый встретившийся символ ‘b’.

## Выполнение работы

Данная программа написана на языке Python. Она получает на вход

строку (является лентой в машине Тьюринга). Было реализовано несколько функций, которые описывают действия машины Тьюринга в каждом из состояний. Описание состояний представлено в табл. 1. С помощью цикла while в функции main() проверяется не находится ли машина в конечном состоянии. Если состояние не равно конечному, выполняется функция, соответствующая текущему состоянию. В конце выводится результат обработки ленты машины.

Таблица 1 – Описание состояний

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Состояние/текущий символ | a | b | c | “ “ |
| q1 | (a, R, q2) | (b, R, q3) | (c, R, q2) | (“ “, R, q1) |
| q2 | (a, R, q2) | (b, R, q3) | (c, R, q2) | (“ “, L, q4) |
| q3 | (a, R, q7) | (b, R, q7) | (c, R, q7) | (“ “, L, q6) |
| q4 | (a, L, q4) | (b, L, q4) | (c, L, q4) | (“ “, R, q5) |
| q5 | (“ “, N, qend) | (“ “, N, qend) | (“ “, N, qend) | (“ “, N, qend) |
| q7 | (a, R, q9) | (b, R, q9) | (c, R, q9) | (“ “, L, q8) |
| q9 | (a, L, q12) | (b, L, q12) | (c, L, q12) | (“ “, L, q10) |
| q10 | (“ “, L, q11) | (“ “, L, q11) | (“ “, L, q11) | (“ “, L, q11) |
| q12 | (“ “, L, q13) | (“ “, L, q13) | (“ “, L, q13) | (“ “, L, q13) |
| q13 | (“ “, L, q14) | (“ “, L, q14) | (“ “, L, q14) | (“ “, L, q14) |
| q14 | (“ “, R, qa) | (“ “, R, qb) | (“ “, R, qc) | (“ “, N, qend) |
| qa | (a, L, qav) | (b, L, qav) | (c, L, qav) | (“ “, R, qa) |
| qb | (a, L, qbv) | (b, L, qbv) | (c, L, qbv) | (“ “, R, qb) |
| qc | (a, L, qcv) | (b, L, qcv) | (c, L, qcv) | (“ “, R, qc) |
| qav | (a, L, qn1) | (a, L, qn1) | (a, L, qn1) | (a, L, qn1) |
| qbv | (b, L, qn1) | (b, L, qn1) | (b, L, qn1) | (b, L, qn1) |
| qcv | (c, L, qn1) | (c, L, qn1) | (c, L, qn1) | (c, L, qn1) |
| qn1 | (a, N, q14) | (b, N, q14) | (c, N, q14) | (“ “, L, qn2) |
| qn2 | (a, N, q14) | (b, N, q14) | (c, N, q14) | (“ “, L, qn3) |
| qn3 | (a, N, q14) | (b, N, q14) | (c, N, q14) | (“ “, N, qend) |

q1- начальное состояние

q2 – при первой найденной букве (не b)

q3 – при первой найденной b

q4 – возврат к началу слова если не было найдено ни одной b

q5 – удаление символа

q7 – первая буква после первой b

q9 – после b есть как минимум 2 буквы

q10 – удаление последней буквы если b – 3 с конца

q12 – удаление второй буквы после b

q13 – удаление первой буквы после b

q14 – перенос левой части к правой, удаление последнего символа левой части

qa – ищем последний пропуск перед правой частью слова

qb – ищем последний пропуск перед правой частью слова

qc – ищем последний пропуск перед правой частью слова

qav – вставляем букву a

qbv – вставляем букву b

qcv – вставляем букву c

qn1 – идём назад и считаем количество пробелов (1)

qn2 – идём назад и считаем количество пробелов (2)

qn3 – 3 пробела – вернулись в начало слова (удаляли только 2 символа)

qend – конечное состояние

Разработанный программный код см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 2.

Таблица 2 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | abcabc | abbc | Ответ корректный |
|  | accccaca | ccccaca | Ответ корректный |
|  | acaabc | acaab | Ответ корректный |

## Выводы

Были изучены принципы работы машины Тьюринга и реализована ее имитация на языке Python.

# Приложение А Исходный код программы

Название файла: main.py

#MAIN

memory = list(input())

def main():

global memory

state = 'q1'

i = 0

while state != 'qend':

memory, i, state = state\_functions.get(state, lambda x, y: (x, y, state))(memory, i)

memory = ''.join(memory)

print(memory)

# STATES

def q1(memory, i):

if memory[i] == 'b':

return memory, i+1, 'q3'

elif memory[i] == ' ':

return memory, i+1, 'q1'

else:

return memory, i+1, 'q2'

def q2(memory, i):

if memory[i] == 'b':

return memory, i+1, 'q3'

elif memory[i] == ' ':

return memory, i-1, 'q4'

else:

return memory, i+1, 'q2'

def q3(memory, i):

if memory[i] == ' ':

return memory, i-1, 'q5'

else:

return memory, i+1, 'q7'

def q4(memory, i):

if memory[i] == ' ':

return memory, i+1, 'q5'

else:

return memory, i-1, 'q4'

def q5(memory, i):

memory[i] = ' '

return memory, i, 'qend'

def q7(memory, i):

if memory[i] == ' ':

return memory, i-1, 'q5'

else:

return memory, i+1, 'q9'

def q9(memory, i):

if memory[i] == ' ':

return memory, i-1, 'q10'

else:

return memory, i-1, 'q12'

def q10(memory, i):

memory[i] = ' '

return memory, i-1, 'q5'

def q12(memory, i):

memory[i] = ' '

return memory, i-1, 'q13'

def q13(memory, i):

memory[i] = ' '

return memory, i-1, 'q14'

def q14(memory, i):

if memory[i] == 'b':

memory[i] = ' '

return memory, i+1, 'qb'

if memory[i] == 'a':

memory[i] = ' '

return memory, i+1, 'qa'

if memory[i] == 'c':

memory[i] = ' '

return memory, i+1, 'qc'

if memory[i] == ' ':

memory[i] = ' '

return memory, i, 'qend'

def qb(memory, i):

if memory[i] == ' ':

return memory, i+1, 'qb'

else:

return memory, i-1, 'qbv'

def qbv(memory, i):

memory[i] = 'b'

return memory, i-1, 'qn1'

def qn1(memory, i):

if memory[i] == ' ':

return memory, i-1, 'qn2'

else:

return memory, i, 'q14'

def qn2(memory, i):

if memory[i] == ' ':

return memory, i-1, 'qn3'

else:

return memory, i, 'q14'

def qn3(memory, i):

if memory[i] == ' ':

return memory, i, 'qend'

else:

return memory, i, 'q14'

def qa(memory, i):

if memory[i] == ' ':

return memory, i+1, 'qa'

else:

return memory, i-1, 'qav'

def qav(memory, i):

memory[i] = 'a'

return memory, i-1, 'qn1'

def qc(memory, i):

if memory[i] == ' ':

return memory, i+1, 'qc'

else:

return memory, i-1, 'qcv'

def qcv(memory, i):

memory[i] = 'c'

return memory, i-1, 'qn1'

#DICT OF STATES NAME:FUNCTIONS

state\_functions = {

'q1': q1, 'q2': q2, 'q3': q3, 'q4': q4, 'q5': q5, 'q7': q7, 'q9': q9, 'q10': q10, 'q12': q12, 'q13': q13,

'q14': q14, 'qa': qa, 'qb': qb, 'qc': qc, 'qav': qav, 'qbv': qbv,

'qcv': qcv, 'qn1': qn1, 'qn2': qn2, 'qn3': qn3

}

#RUNNING

if \_\_name\_\_ == '\_\_main\_\_':

main()