МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Информатика»
Тема: Машина Тьюринга

Студент гр. 2381 _____ Ильясов М.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2022

Цель работы

Изучить принцип действия машины Тьюринга, реализовать механизм работы на языке программирования Python 3 и написать программу по тексту задания для закрепления полученных знаний.

Задание

Вариант №4. 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}, которая начинается с символа 'a'.

Напишите программу, которая оборачивает исходную строку. Результат работы алгоритма - исходная последовательность символов в обратном порядке.

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 (можно расширять при необходимости):

- 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 отчет включите таблицу состояний. Отдельно кратко опишите каждое состояние, например:

q1 - начальное состояние, которое необходимо, чтобы обнаружить конец строки.

Выполнение работы

1. Инициализация вспомогательных переменных и объявление дополнительных функций. Строка *alf* содержит алфавит *„ #abc“* (был добавлен один дополнительный символ *„#“*), словарь *tape*, где ключ - позиция на ленте, а значение — соответствующий ему символ задаётся входными данными через *dict(enumerate(input()))*. Функция *current* возвращает символ из словаря по индексу, с заполнением пробелом, в случае отсутствия ключа в ассоциативном массиве. Функция *upd* производит *dict.update()* с возвратом копии (стандартная функция изменяет сам словарь и возвращает *None*). Переменные *L*, *N*, *R* задаются значениями *-1*, *0*, *1* и обозначают сдвиг индекса автомата при перемещении. Лямбда-функции *move_rigth* и *move_left* созданы для сокращения записи при составлении таблицы, возвращают на основе состояния *state* строку таблицы, реализующую циклический сдвиг автомата вправо и влево соответственно. Функция *show()* выводит итоговое значение ленты, через прохождение циклом *for* по ключам *tape*.

2. Написание таблицы и реализация работы машины Тьюринга

Таблица состояний:

состояние	' '	'#'	'a'	'b'	'c'
'start'	<' ', R, 'start'>	<'#', R, 'start'>	<'a', R, 'get'>	<'b', R, 'start'>	<'c', R, 'start'>
'get'	<' ', L, 'clean'>	<'#', R, 'get'>	<'#', L,'put_a'>	<'#', L,'put_b'>	<'#', L,'put_c'>

'put_a'	<'a', R,'find_#'>	<'#', L, 'put_a'>	<'a', L, 'put_a'>	<'b', L, 'put_a'>	<'c', L, 'put_a'>
'put_b'	<'b', R, 'find_#'>	<'#', L, 'put_b'>	<'a', L,'put_b'>	<'b', L, 'put_b'>	<'c', L,'put_b'>
'put_c'	<'c', R, 'find_#'>	<'#', L, 'put_c'>	<'a', L, 'put_c'>	<'b', L, 'put_c'>	<'c', L,'put_c'>
'find_#'	<' ', R,'find_#'>	<'#', R, 'get'>	<'a', R,'find_#'>	<'b',R,'find_#'>	<'c',R,'find_#'>
'clean'		<' ', L, 'clean'>	<'a', N, 'end'>		
'end'					

start – стартовое состояние для поиска открывающего символа *„a“*. При получении автомат переходит в состояние *„get“*.

get – состояние поиска среди символов первого, который не является служебным символом *«#»*. В случае, если таких символов нет, автомат переходит в состояние clean, иначе символ заменяется «#» и идёт переход в *put_a*, *put_b* или *put_c*.

put_a, *put_b*, *put_c* – состояния, для переноса соответствующих символов влево (посредством чего реализуется инверсия строки). Идёт движение (переход влево с заменой видимого символа на него самого) до первого пробела, на место которого ставится *„a“*, *„b“* или *„c“*.

clean — состояние, для замены *„#“* на пробелы. Переходит в *end* при пробеле.

end – конец работы программы.

Далее в программе реализован цикл *while*, вызывающий *current()* для получения видимого символа, с последующим получением нужной строки состояний в *table*. Обновляется состояние *state*, записывается *delta* и сдвигается *index* на соответствующее значение. Обработка продолжается до состояния *„end“*, после чего лента выводится на экран.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	aaabbbccc	cccbbbaaa	верная обработка

2.	abcabcabcabca	acbacbacbacba	крайний случай (макс. 13 символов)
3.	abbca	acbba	крайний случай (мин. 5 символов)

Выводы

Были изучены основы машины Тьюринга, принцип её действия. Была сделана реализация автомата на Python. Составлена программа для машины Тьюринга, осуществляющая разворот строки.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: menu.py

```
alf = ' #abc'
tape = dict(enumerate(input()))

def current():
      simbol = tape.get(index,' ')
      tape.update({index: simbol})
      return simbol

def upd(d1, d2):
      d = d1.copy()
      d.update(d2)
      return d

L, N, R = [-1,0,1]

move_rigth = lambda state: {a:[a, R, state] for a in alf}
move_left = lambda state: {a:[a, L, state] for a in alf}

table = {
      'start': upd(move_rigth('start'), {'a': ['a', R,'get']}),
      'get': upd({'#':['#',R,'get'], ' ':[' ', L, 'clean']}, {x:
['#', L, f'put_{x}'] for x in alf[2:]}),

      **{f'put_{x}': upd(move_left(f'put_{x}'), {' ': [x, R,
'find_#']}) for x in alf[2:]},

      'find_#': upd(move_rigth('find_#'), {'#': ['#', R, 'get']}),
      'clean': {'#':[' ', L, 'clean'], 'a':['a', N, 'end']},
      'end': {},
}

"""
#Таблица в явном виде:

def show_table():
      print('table = {')
      for k in list(table.keys()):
            print('\t'+f"'{k}': {table[k]}")
      print('}')

table = {
      'start': {' ': [' ', 1, 'start'], '#': ['#', 1, 'start'],
'a': ['a', 1, 'get'], 'b': ['b', 1, 'start'], 'c': ['c', 1, 'start']}
      'get': {'#': ['#', 1, 'get'], ' ': [' ', -1, 'clean'], 'a':
['#', -1, 'put_a'], 'b': ['#', -1, 'put_b'], 'c': ['#', -1, 'put_c']}
      'put_a': {' ': ['a', 1, 'find_#'], '#': ['#', -1, 'put_a'],
'a': ['a', -1, 'put_a'], 'b': ['b', -1, 'put_a'], 'c': ['c', -1,
'put_a']}
```

```
          'put_b': {' ': ['b', 1, 'find_#'], '#': ['#', -1, 'put_b'],
'a': ['a', -1, 'put_b'], 'b': ['b', -1, 'put_b'], 'c': ['c', -1,
'put_b']}
          'put_c': {' ': ['c', 1, 'find_#'], '#': ['#', -1, 'put_c'],
'a': ['a', -1, 'put_c'], 'b': ['b', -1, 'put_c'], 'c': ['c', -1,
'put_c']}
          'find_#': {' ': [' ', 1, 'find_#'], '#': ['#', 1, 'get'],
'a': ['a', 1, 'find_#'], 'b': ['b', 1, 'find_#'], 'c': ['c', 1,
'find_#']}
          'clean': {'#': [' ', -1, 'clean'], 'a': ['a', 0, 'end']}
          'end': {}
     }

     show_table()
     """

     def show():
          for k in sorted(list(tape.keys())):
               print(tape[k], end='')
          print('',end='\n')


     index = 0
     state = 'start'
     while state != 'end':
          tape[index], delta, state = table[state][current()]
          index += delta
     show()
```