МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра теории вероятностей и компьютерного моделирования

Лабораторная работа № 2
По спецкурсу «Теория сложности алгоритмов»

Контекстно-свободные языки

Выполнил: Покхарел П.К.
Группа: М8О-101М-22, Вариант 4
Преподаватель: Рассказова В.А.

Москва, 2023

Задание. Для заданного КС-языка B над алфавитом $\Sigma = \{0, 1\}$

1. построить диаграмму состояний МП-автомата, распознающего A;
2. реализовать данный МП-автомат в виде программы, которая для произвольной входной строки w должна выводить историю вычислений МП-автомата на ней в виде последовательности состояний и содержимого стэка памяти.

$$B = \{0^n 1^i 0^j : n = i \text{ или } n = j\};$$

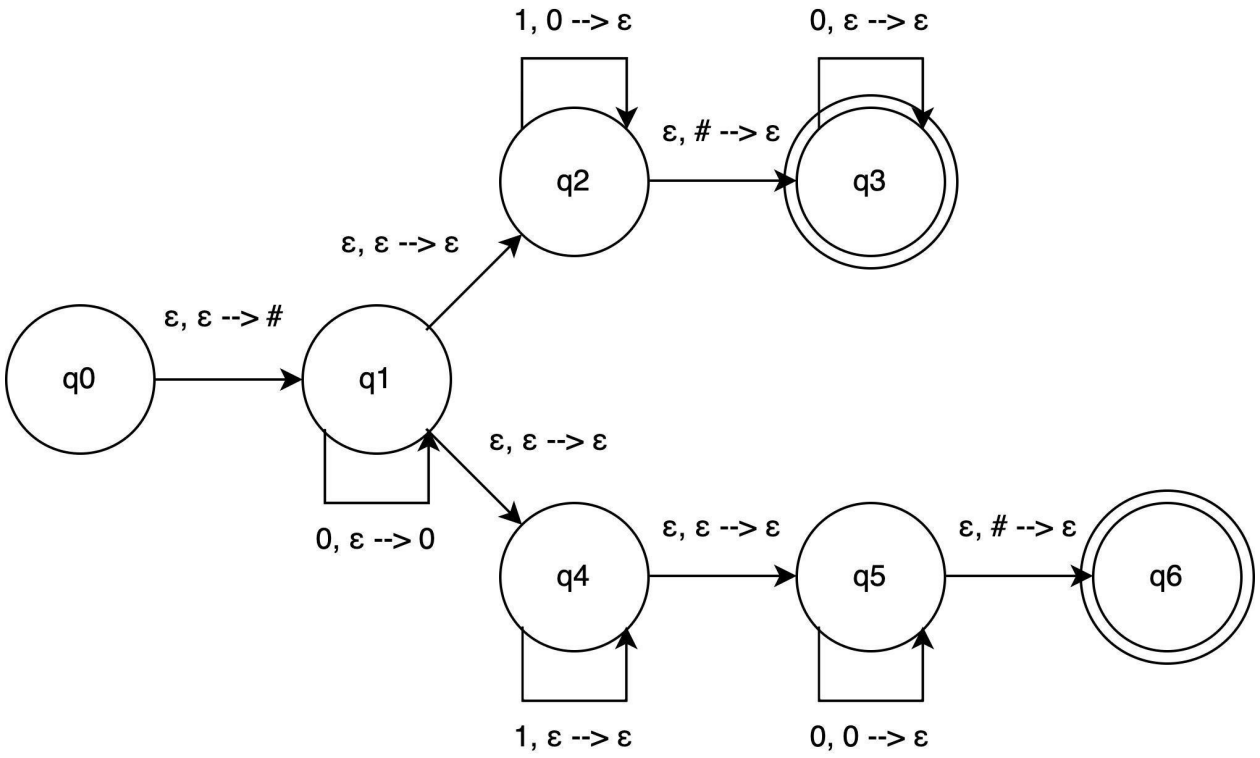

Рис 1 Диаграмма состояний МП-автомата

	0			1			ε		
	0	1	#	0	1	#	0	1	#
q0	Ø	Ø	Ø	Ø	Ø	Ø	{q1, #}	Ø	{q1, #}
q1	{q1, 0}	Ø	{q1, 0}	Ø	Ø	Ø	{q2, ε}; {q4, ε}	Ø	{q2, ε}; {q4, ε}
q2	Ø	Ø	Ø	{q2,ε}	Ø	{q2,ε}	Ø	Ø	{q3,ε}
q3	{q3,ε}	Ø	{q3,ε}	Ø	Ø	Ø	Ø	Ø	Ø
q4	Ø	Ø	Ø	{q4,ε}	Ø	{q4,ε}	{q5,ε}	Ø	{q5,ε}
q5	{q5,0}	Ø	{q5,0}	Ø	Ø	Ø	Ø	Ø	{q6,ε}

Рис 2 Таблица состояний МП-автомата

Программная реализация

```
def is_valid_string(s):
    stack = ['', '#']
    states = ['q0', 'q1']
    transitions = {
        ('q1', '0', '#'): [('q1', '0', 'i')],
        ('q1', '0', '0'): [('q1', '0', 'i')],
        ('q1', '1', '0'): [('q2', '0', 'd')],
#       ('q1', '1', '0'): [('q4', '', '')],
        ('q2', '1', '0'): [('q2', '0', 'd')],
        ('q2', '', '#'): [('q3', '#', 'd')],
        ('q2', '0', '#'): [('q3', '#', 'd')],
        ('q3', '0', ''): [('q3', '', 'd')],

    }
    new_states = ['q0']
```

```
    new_state = states[-1]


    s = list(s)


    for char in s:
#       print(char)
        state = new_state
#       print(state)
        check = 0
        for transition in transitions:
            if state == transition[0] and char == transition[1] and stack[-1] == transition[2]:
#               print("OKAY")
                for new_state, push_char, st_char in transitions[transition]:
                    new_stack = stack.copy()
                    if push_char != '' and st_char == 'i':
                        new_stack.append(push_char)
                    if push_char != '' and new_stack[-1] == push_char and st_char == 'd':
#                       print('delete')
                        new_stack.pop()
#                   if new_state not in new_states:
                    new_states.append(new_state)
#                   print(f"({state}, {char}, {stack[-1]}) -> ({new_state}, {push_char}), stack={new_stack}")
                check += 1
        if not check:
            new_state = 'fail'
            break
```

```
            stack = new_stack
            states = new_states.copy()
        return 'q3' == new_state, '->'.join(new_states)


def is_valid_string_2(s):
    stack = ['', '#']
    states = ['q0', 'q1']
    transitions = {
        ('q1', '0', '#'): [('q1', '0', 'i')],
        ('q1', '0', '0'): [('q1', '0', 'i')],
        ('q1', '1', '#'): [('q4', '', 'd')],
        ('q1', '1', '0'): [('q4', '', 'd')],
        ('q4', '1', '0'): [('q4', '', 'd')],
        ('q4', '1', '#'): [('q4', '', 'd')],
        ('q4', '', '#'): [('q5', '', 'd')],
        ('q4', '0', '#'): [('q5', '0', 'd')],
        ('q4', '0', '0'): [('q5', '0', 'd')],
        ('q5', '0', '0'): [('q5', '0', 'd')],
        ('q5', '', '#'): [('q6', '#', 'd')],
    }
    new_states = ['q0']
    new_state = states[-1]

    s = list(s) + ['', '']

    for char in s:
#       print(char)
```

```
        state = new_state
 #      print(state)
        check = 0
        for transition in transitions:
            if state == transition[0] and char == transition[1] and stack[-1] == transition[2]:
#               print("OKAY")
                print
                for new_state, push_char, st_char in transitions[transition]:
                    new_stack = stack.copy()
#                   print(push_char == new_stack[-1])
                    if push_char != '' and st_char == 'i':
                        new_stack.append(push_char)
                    if push_char != '' and new_stack[-1] == push_char and st_char == 'd':
 #                      print('delete')
                        new_stack.pop()
 #                  if new_state not in new_states:
                    new_states.append(new_state)
 #                  print(f"({state}, {char}, {stack[-1]}) -> ({new_state}, {push_char}), stack={new_stack}")
                check += 1
        if not check:
#           new_state = 'fail'
            break

        stack = new_stack
        states = new_states.copy()
    return 'q6' == new_state, '->'.join(new_states)
```

```
def is_valid(s):
    line1 = is_valid_string(s)
    line2 = is_valid_string_2(s)
    need_state = line1[0] or line2[0]
    if need_state:
        return line1 if line1[0] else line2
    return line1[0], line1[1], 'or', line2[1]


def qf_lang_check(s):
    print(*is_valid(s), sep='\n')


word = input()
qf_lang_check(word)
```