ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дожность |  |  |  | [Рогачев С.A](https://pro.guap.ru/inside/profile/911) |
| [старший преподаватель](https://guap.ru/rasp/?p=317) |  | подпись, дата |  | инициалы, фамилия |

|  |  |
| --- | --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |  |
| Машина Тьюринга |  |
| по дисциплине: [Теория вычислительных процессов](https://pro.guap.ru/inside/students/subjects/3391377) |  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4236 |  |  |  | Л. Мвале |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2025

1. **Цель работы:** Освоить понятие рекурсивных функций, научиться преобразовывать арифметические функции в рекурсивную форму с использованием примитивной рекурсии. Реализовать функцию умножения итеративным и рекурсивным способом на языке высокого уровня
2. **Основные сведения из теории**

Рекурсия – это метод определения функций, при котором значение функции в текущий момент выражается через её значение на предыдущих шагах (для меньших аргументов). Иными словами, функция вызывает сама себя до тех пор, пока не достигнет базового случая.

Примитивная рекурсия – это особый вид рекурсии, где функция определяется через:

- базис (значение функции для минимального аргумента),

- рекурсивное правило (вычисление для большего аргумента через результат для меньшего).

Примитивная рекурсия всегда заканчивается, потому что каждый рекурсивный шаг уменьшает аргумент и в итоге приводит к базовому случаю.

Примеры рекурсивных функций:

Факториал:

* n! = 1, если n = 0
* n! = n × (n – 1)!, если n > 0

Последовательность Фибоначчи:

* F(0) = 0, F(1) = 1
* F(n) = F(n – 1) + F(n – 2), если n > 1

Умножение через сложение (наш случай):

* x1 \* 0 = 0 (база рекурсии)
* x1 \* x2 = x1 + (x1 \* (x2 – 1)) (рекурсивное определение)

Таким образом, операция умножения может быть сведена к многократному сложению, а её рекурсивное определение соответствует принципу примитивной рекурсии.

1. **Постановка задачи**  
    Необходимо реализовать программу, которая вычисляет произведение двух целых чисел двумя способами: итеративным (с помощью сложения) и рекурсивным (с помощью примитивной рекурсии). Пользователь вводит числа, программа выводит результаты обоими методами и проверяет их корректность.
2. **Преобразование арифметической функции**

Шаг преобразования:

x1 \* 0 = 0 (база рекурсии).

x1 \* x2 = x1 + (x1 \* (x2 - 1)) (рекурсивное определение).

1. **Листинг программы**

class TuringMachine:

    def \_\_init\_\_(self, input\_tape):

        self.tape = list(input\_tape)

        self.head = 0

        self.state = 'q0'

        self.steps = 0

    def \_get\_current\_symbol(self):

        if self.head < 0 or self.head >= len(self.tape):

            return '\_'

        return self.tape[self.head]

    def \_set\_current\_symbol(self, symbol):

        if self.head < 0:

            self.tape = [symbol] + ['\_'] \* (-self.head - 1) + self.tape

            self.head = 0

        elif self.head >= len(self.tape):

            self.tape = self.tape + ['\_'] \* (self.head - len(self.tape)) + [symbol]

        else:

            self.tape[self.head] = symbol

    def \_move(self, direction):

        if direction == 'R':

            self.head += 1

        elif direction == 'L':

            self.head -= 1

        elif direction != 'S':

            raise ValueError(f"Неверное направление: {direction}")

    def \_print\_state(self):

        tape\_str = ''.join(self.tape)

        head\_pos = ' ' \* self.head + '^'

        print(f"Шаг {self.steps:3d}: Состояние={self.state:2s} | Лента: {tape\_str}")

        print(f"           {' ' \* 12} {head\_pos}")

    def run(self, max\_steps=500, verbose=True):

        if verbose:

            print("Начальное состояние:")

            self.\_print\_state()

            print("\n" + "="\*50)

        # ИСПРАВЛЕННЫЕ ПРАВИЛА ПЕРЕХОДОВ

        transitions = {

            # Начальное состояние - найти первую 1 для пометки

            ('q0', '1'): ('X', 'R', 'q1'),

            ('q0', '\*'): ('\*', 'R', 'q9'),

            ('q9', '1'): ('y', 'R', 'q9'),

            ('q9', '='): ('=', 'S', 'halt'),

            # После пометки левой 1, перейти к правой части

            ('q1', '1'): ('1', 'R', 'q1'),

            ('q1', '\*'): ('\*', 'R', 'q2'),

            # Найти первую 1 в правой части для копирования

            ('q2', '1'): ('Y', 'R', 'q3'),

            ('q2', 'Y'): ('Y', 'R', 'q2'),

            ('q2', '='): ('=', 'L', 'q7'),

            # Перейти к концу ленты чтобы добавить новую 1

            ('q3', '1'): ('1', 'R', 'q3'),

            ('q3', '='): ('=', 'R', 'q4'),

            ('q4', '\_'): ('1', 'L', 'q5'),

            ('q4', '1'): ('1', 'R', 'q4'),

            # Вернуться к правой части

            ('q5', '1'): ('1', 'L', 'q5'),

            ('q5', '='): ('=', 'L', 'q6'),

            ('q6', '1'): ('1', 'L', 'q6'),

            ('q6', 'Y'): ('Y', 'L', 'q6'),

            ('q6', '\*'): ('\*', 'R', 'q2'),

            # Все правые 1 обработаны, вернуться к левой части - СБРОСИТЬ Y обратно в 1

            ('q7', 'Y'): ('1', 'L', 'q7'),  # Сбросить Y в 1

            ('q7', '1'): ('1', 'L', 'q7'),

            ('q7', '\*'): ('\*', 'L', 'q8'),

            # Очистка - все левые 1 обработаны

            ('q8', '1'): ('1', 'L', 'q8'),

            ('q8', 'X'): ('X', 'R', 'q0'),  # ДОБАВЛЕНО: Очистить оставшиеся Y

        }

        while self.state != 'halt' and self.steps < max\_steps:

            current\_symbol = self.\_get\_current\_symbol()

            key = (self.state, current\_symbol)

            if key not in transitions:

                if verbose:

                    print(f"\nНе определен переход для (состояние={self.state}, символ={current\_symbol})")

                    print("Текущая лента:", ''.join(self.tape))

                break

            write\_symbol, move\_direction, next\_state = transitions[key]

            self.\_set\_current\_symbol(write\_symbol)

            self.\_move(move\_direction)

            self.state = next\_state

            self.steps += 1

            if verbose and self.steps <= 100:

                self.\_print\_state()

        if verbose:

            print("\n" + "="\*50)

            if self.state == 'halt':

                print("Вычисления завершены успешно!")

            else:

                print(f"Вычисления остановлены после {self.steps} шагов")

            result = ''.join([c for c in self.tape if c == '1'])

            print(f"Финальный результат: {len(result)} единиц -> {result}")

        return self.tape, self.state

def create\_input\_string(a, b):

    """Создать входную строку в формате '111\*111=' для заданных чисел"""

    left\_ones = '1' \* a

    right\_ones = '1' \* b

    return f"{left\_ones}\*{right\_ones}=\_\_\_\_\_\_\_\_\_\_\_\_\_\_"

def test\_multiplication():

    """Протестировать машину Тьюринга для умножения"""

    test\_cases = [

        (1, 1),  # 1 × 1 = 1

        (2, 1),  # 2 × 1 = 2

        (1, 2),  # 1 × 2 = 2

        (2, 2),  # 2 × 2 = 4

        (2, 3),  # 2 × 3 = 6

        (3, 2),  # 3 × 2 = 6

    ]

    print("Тестирование машины Тьюринга для умножения")

    print("=" \* 40)

    for a, b in test\_cases:

        input\_str = create\_input\_string(a, b)

        expected = a \* b

        print(f"\nТестирование {a} × {b} = {expected}")

        print(f"Вход: {input\_str}")

        tm = TuringMachine(input\_str)

        final\_tape, final\_state = tm.run(verbose=False, max\_steps=500)

        result\_ones = ''.join([c for c in final\_tape if c == '1'])

        print(f"Ожидается: {expected} единиц")

        print(f"Получено:  {len(result\_ones)} единиц")

        if len(result\_ones) == expected:

            print("✓ УСПЕХ!")

        else:

            print("✗ НЕУДАЧА!")

            print(f"Финальная лента: {''.join(final\_tape)}")

        print("-" \* 40)

def debug\_case(a, b):

    """Отладить конкретный случай с подробным выводом"""

    print(f"\nОТЛАДКА {a} × {b}:")

    print("=" \* 30)

    input\_str = create\_input\_string(a, b)

    expected = a \* b

    print(f"Вход: {input\_str}")

    print(f"Ожидается: {expected} единиц")

    tm = TuringMachine(input\_str)

    final\_tape, final\_state = tm.run(verbose=True, max\_steps=200)

    result\_ones = ''.join([c for c in final\_tape if c == '1'])

    print(f"\nОжидается: {expected} единиц, Получено: {len(result\_ones)} единиц")

    if len(result\_ones) == expected:

        print("✓ УСПЕХ!")

    else:

        print("✗ НЕУДАЧА!")

        print(f"Финальная лента: {''.join(final\_tape)}")

if \_\_name\_\_ == "\_\_main\_\_":

    # Запустить комплексные тесты

    test\_multiplication()

    # Отладить конкретные случаи которые не работают

    print("\n" + "="\*60)

    print("ОТЛАДКА НЕУДАЧНЫХ СЛУЧАЕВ:")

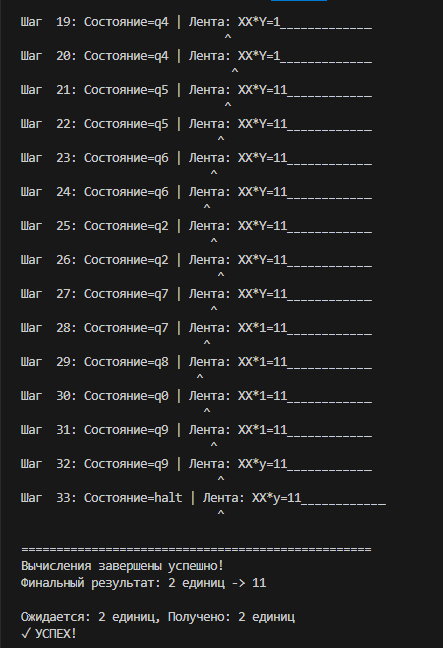
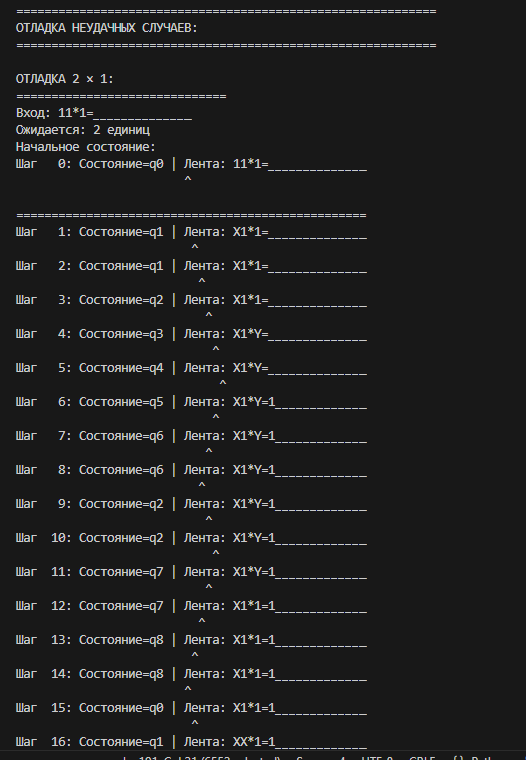
    print("="\*60)

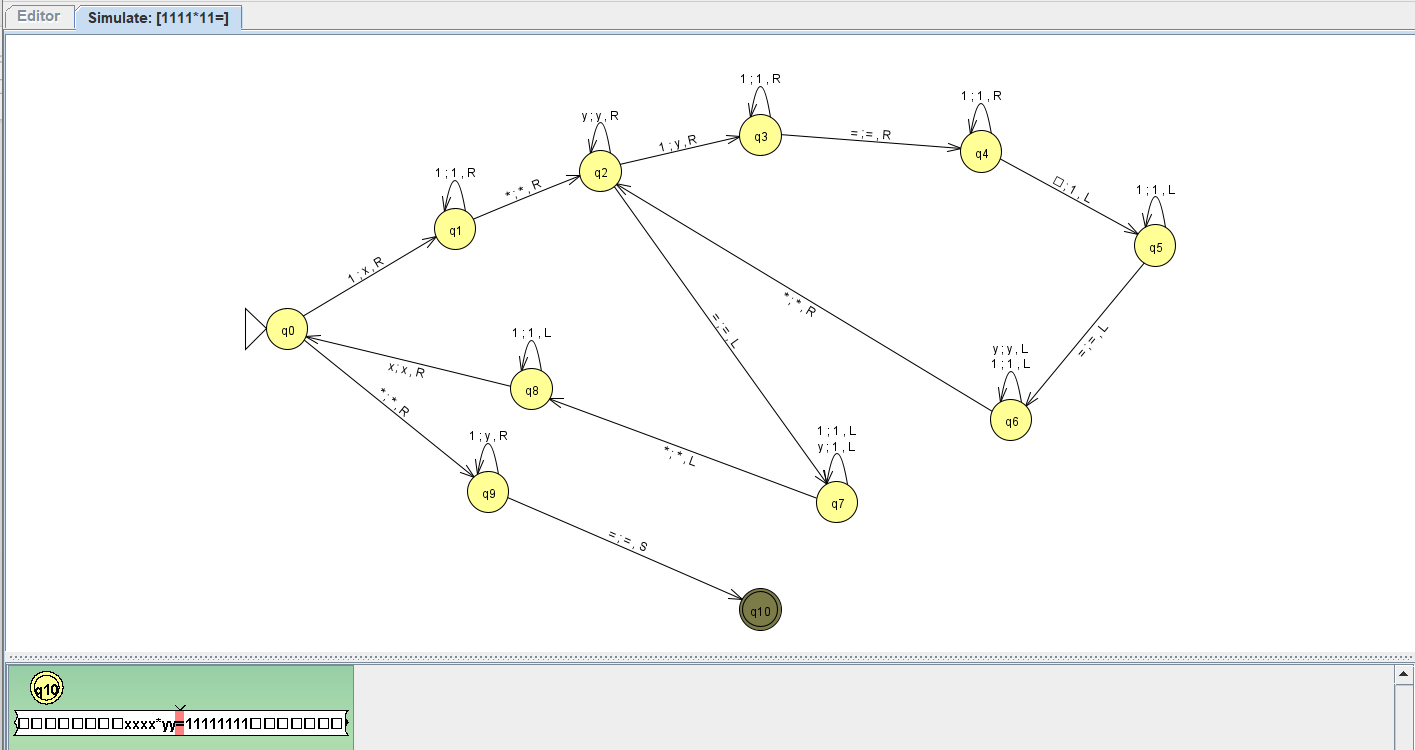
    debug\_case(2, 1)

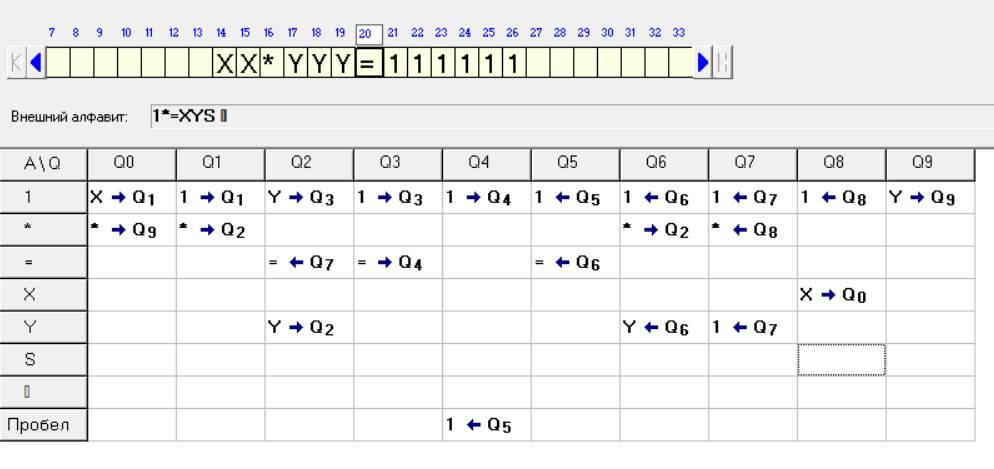
    debug\_case(2, 2)

1. **Пример выполнения**









1. **Выводы**

В ходе работы были изучены рекурсивные функции, реализованы алгоритмы итеративного и рекурсивного умножения. Оба метода дают одинаковые результаты, что подтверждает правильность реализации примитивной рекурсии.