Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №5

Выполнили:

Петрова Н. Г

Садовая А. Р

Оншин Д. Н

Проверил:

Мусаев А.А.

Санкт-Петербург,

2023

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ

Данная работа представляет собой отчет о выполненных заданиях:

1. Задача 1 (результат игры в крестики-нолики 3x3).
2. Задача 2 (поиск элемента в матрице)
3. Задча 3 (шахматы)
4. Задача 4 (лестница)
5. Задача 5 (массив и стеки)
6. Задача 6 (экспоненциальный фильтр)
7. Задача 7 (поиск наименьшего пропущенного числа в массиве)

1 Задача 1

Данный код представляет собой решение задачи определения результата игры в крестики-нолики на поле 3x3. Рассмотрим подробнее, как он работает:

1. В начале определяются три вспомогательные функции: row data, col data и diag data, которые проверяют соответственно строки, столбцы и диагонали на наличие одинаковых символов (крестиков или ноликов). Если в какой-либо строке, столбце или диагонали все символы одинаковы, то функция возвращает строку с сообщением о победителе. В противном случае, функция возвращает 0.
2. Затем создается пустой двумерный список data, который будет представлять собой поле крестики-нолики размером 3x3.
3. Далее, пользователю предлагается ввести данные игры. Он должен ввести 9 символов: '0' (нолик) или '1' (крестик). В цикле происходит разбиение введенных символов на строки и заполнение списка data соответствующими значениями. После каждого ввода строки игрового поля выводится на экран.
4. Затем происходит проверка победителя путем вызова вспомогательных функций row data, col data и diag data. Если хотя бы одна из функций возвращает ненулевое значение, то на экран выводится сообщение о победителе, и переменная flag устанавливается в 1.
5. Если после проверки строк, столбцов и диагоналей flag остается равной 0, то проверяется наличие ничьей. Если функция diag data возвращает 0, то на экран выводится сообщение "Draw".
6. Если flag остается равной 0, и функция diag data не возвращает 0, то на экран выводится сообщение о победителе в диагонали.

```
def task_1():
    def row_data(k):
        if data[k][0] == data[k][1] == data[k][2]:
            s = str(data[k][0]) + ' win'
            return s
        return 0

    def col_data(k):
        if data[0][k] == data[1][k] == data[2][k]:
            s = str(data[0][k]) + ' win'
            return s
        return 0

    def diag_data():
        if data[0][0] == data[1][1] == data[2][2] or data[0][2] == data[1][1] == data[2][0]:
            s = str(data[1][1]) + ' win'
            return s
        return 0

    data = [[], [], []]
    flag = 0
    s_data = input('Input data: ')
    for i in range(len(s_data)):
        data[i // 3].append(int(s_data[i]))
        if i == 2 or i == 5 or i == 8:
            print(data[i // 3])
```

Рисунок 1.1 — Код функции для задачи 1

```
for i in range(3):
    if row_data(i) != 0:
        print(row_data(i))
        flag = 1
        break
    elif col_data(i) != 0:
        print(col_data(i))
        flag = 1
        break
if flag == 0:
    if diag_data() == 0:
        print('Draw')
    else:
        print(diag_data())
```

Рисунок 1.2 — Продолжение кода функции для задачи 1

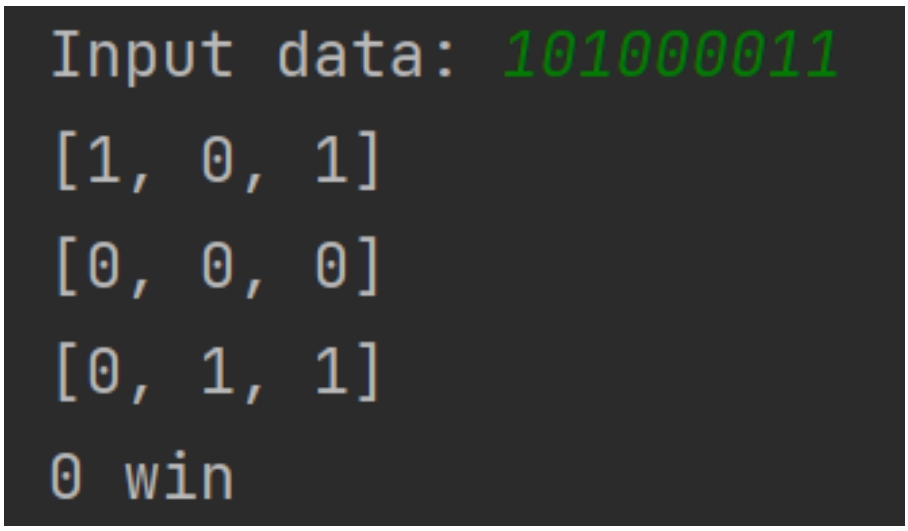


Рисунок 1.3 — Пример вывода функции для задачи 1

2 Задача 2

Алгоритм ищет указанное значение в матрице и выводит его индексы (позицию) в случае совпадения
Алгоритм по шагам:

1. Создается матрица matrix, представляющая собой двумерный массив чисел.
2. Матрица выводится на экран с помощью функции print(matrix).
3. Пользователю предлагается ввести целевое значение (target) с помощью функции input().
4. Определяются количество строк и столбцов матрицы с помощью функций len(matrix) и len(matrix[0]).
5. Инициализируются переменные row и col, которые будут использоваться для обхода матрицы.
6. Устанавливается флаг flag в 0.
7. Запускается цикл while, который выполняется, пока row меньше количества строк и col больше или равно нулю.
8. Внутри цикла проверяется текущий элемент matrix[row][col] на равенство с целевым значением target.
9. Если элемент равен целевому значению, выводятся индексы этого элемента (позиция в матрице) с помощью функции print(row, col).
10. Устанавливается флаг flag в 1 и цикл прерывается с помощью оператора break.
11. Если элемент меньше целевого значения, индекс строки row увеличивается на 1.
12. Если элемент больше целевого значения, индекс столбца col уменьшается на 1.
13. После цикла проверяется значение флага flag. Если флаг равен 0, выводится сообщение "Wrong target!".

```python
def task_2():
    matrix = [[1, 3, 10, 12],
              [10, 11, 16, 20],
              [23, 30, 34, 50]]
    print(matrix)
    target = int(input('Input target: '))

    rows = len(matrix)
    cols = len(matrix[0])
    row = 0
    col = cols - 1
    flag = 0
    while row < rows and col >= 0:
        if matrix[row][col] == target:
            print(row, col)
            flag = 1
            break
        elif matrix[row][col] < target:
            row += 1
        else:
            col -= 1
    if flag == 0:
        print('Wrong target! ')
```

Рисунок 2.1 — Код функции для задачи 2

```
[[1, 3, 10, 12], [10, 11, 16, 20], [23, 30, 34, 50]]
Input target: 13
Wrong target!
```

Рисунок 2.2 — Пример вывода функции для задачи 2

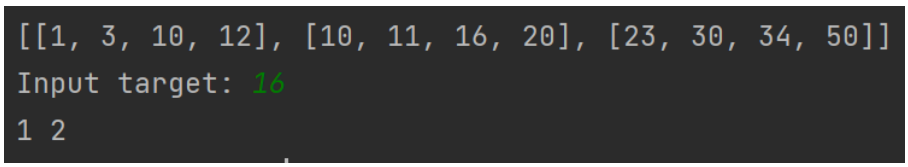


Рисунок 2.3 — Пример вывода функции для задачи 2

3 Задача 3

Данный код решает задачу о расстановке 8 ферзей на шахматной доске размером 8x8 таким образом, чтобы никакие две фигуры не находились на одной диагонали.
Алгоритм по шагам:

1. Инициализируется переменная n со значением 8, которая представляет размерность шахматной доски.
2. Создается пустая доска board, представляющая собой двумерный массив размером 8x8, заполненный нулями. Каждая клетка доски будет содержать 0 или 1, где 0 обозначает пустую клетку, а 1 обозначает размещенного на клетке ферзя.
3. Инициализируется пустой список solutions, в который будут добавляться найденные решения (варианты расстановки ферзей).
4. Определяются две вспомогательные функции: is safe(row, col): Проверяет, является ли текущая позиция (row, col) безопасной для размещения ферзя на доске. Функция проверяет следующие условия: Нет ферзей на одной вертикали (проверка board[i][col] == 1 для всех i от 0 до row-1). Нет ферзей на одной левой диагонали (проверка board[i][col - (row - i)] == 1 для всех i от 0 до row-1). Нет ферзей на одной правой диагонали (проверка board[i][col + (row - i)] == 1 для всех i от 0 до row-1). place queen(row): Рекурсивная функция для размещения ферзей на доске. Если row равна n, то все 8 ферзей уже размещены на доске, и текущая расстановка добавляется в список решений solutions. В противном случае, для каждого столбца col в текущей строке row, проверяется, является ли размещение ферзя в данной позиции безопасным (is safe(row, col)). Если безопасно, то ферзь размещается в клетку (row, col), вызывается рекурсивно функция place queen(row + 1) для следующей строки, а затем ферзь убирается из клетки (row, col) для проверки других возможных позиций.
5. Запускается функция place queen(0), которая начинает процесс размещения ферзей на доске, начиная с первой строки (нулевой индекс).

6. После выполнения функции place queen, список solutions будет содержать все возможные варианты расстановки 8 ферзей на шахматной доске, удовлетворяющие условию безопасности.
7. Выводится количество найденных решений с помощью функции print(len(solutions)).

```python
def task_3():
    n = 8
    board = [[0] * n for _ in range(n)]
    solutions = []
    def is_safe(row, col):
        for i in range(row):
            if board[i][col] == 1:
                return False
            if col - (row - i) >= 0 and board[i][col - (row - i)] == 1:
                return False
            if col + (row - i) < n and board[i][col + (row - i)] == 1:
                return False
        return True
    def place_queen(row):
        if row == n:
            solution = []
            for i in range(n):
                for j in range(n):
                    if board[i][j] == 1:
                        solution.append('Q')
                    else:
                        solution.append('.')
            solutions.append(solution)
```

Рисунок 3.1 — Код функции для задачи 3

```python
        else:
            for col in range(n):
                if is_safe(row, col):
                    board[row][col] = 1
                    place_queen(row + 1)
                    board[row][col] = 0
    place_queen(0)
    print(len(solutions))
```

Рисунок 3.2 — Продолжение кода функции для задачи 3

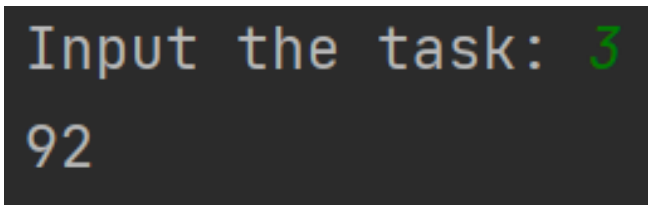


Рисунок 3.3 — Пример вывода функции для задачи 3

4 Задача 4

В этой программе функция принимает количество ступенек n и использует подход динамического программирования для вычисления количества возможных вариантов перемещения ребенка по лестнице. Базовые случаи, когда на лестнице 0, 1 и 2 ступеньки, обрабатываются отдельно. Затем используется цикл для вычисления количества способов для каждой ступеньки, начиная с третьей. Результат выводится на экран. Алгоритм по шагам:

1. Принимается аргумент n, представляющий количество ступенек на лестнице (ввод осуществляется при вызове функции)
2. В условном операторе проверяются базовые случаи:
3. Если на лестнице 0 или 1 ступенька, то есть только один способ перемещения (ребенок остается на месте или перемещается на единственную ступеньку). В этом случае функция возвращает 1.
4. Если на лестнице 2 ступеньки, то есть два способа перемещения: ребенок может сразу переместиться на вторую ступеньку или сделать два отдельных шага по одной ступеньке. Функция возвращает 2.
5. Если n больше 2, создается список ways длиной n + 1, инициализируется нулями. Этот список будет хранить количество способов перемещения для каждой ступеньки.
6. Устанавливаются базовые случаи: ways[0] = 1, ways[1] = 1, ways[2] = 2. Таким образом, у нас уже есть значения для первых трех ступенек.
7. С помощью цикла for перебираются ступеньки от 3 до n (включительно). Для каждой ступеньки вычисляется количество способов перемещения, используя формулу ways[i] = ways[i - 1] + ways[i - 2] + ways[i - 3]. Это означает, что количество способов перемещения на текущей ступеньке равно сумме количества способов перемещения на предыдущих трех ступенках. Это связано с тем, что ребенок может переместиться на текущую ступеньку либо с предыдущей, либо с предпоследней, либо с антипредпоследней ступеньки.

8. По завершении цикла возвращается значение ways[n], которое представляет количество возможных вариантов перемещения ребенка по лестнице из n ступенек.

```python
def task_4(n):
    if n == 0 or n == 1:
        return 1
    elif n == 2:
        return 2
    else:
        ways = [0] * (n + 1)
        ways[0] = 1
        ways[1] = 1
        ways[2] = 2

        for i in range(3, n + 1):
            ways[i] = ways[i - 1] + ways[i - 2] + ways[i - 3]

        return ways[n]
```

Рисунок 4.1 — Код функции для задачи 4

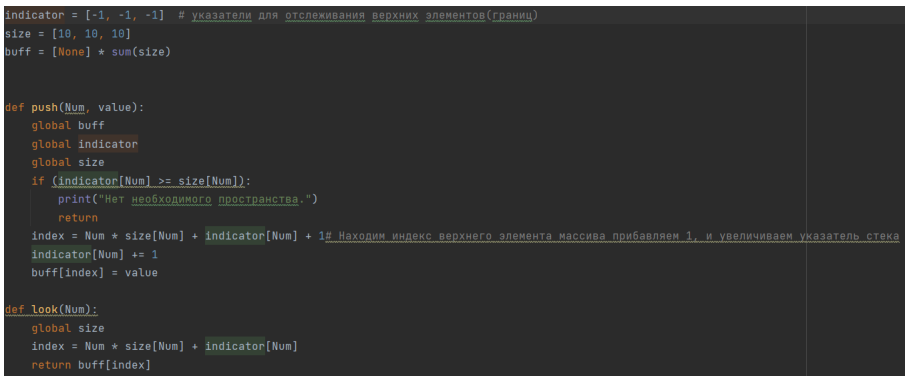
```
Введите длину лестницы: 3
Количество возможных вариантов перемещения по лестнице из 3 ступенек: 4
Input the task: 4
Введите длину лестницы: 10
Количество возможных вариантов перемещения по лестнице из 10 ступенек: 274
```

Рисунок 4.2 — Пример вывода функции для задачи 4

5 Задача 5

Данный код представляет реализацию структуры данных множественный стек, где несколько стеков хранятся в одном массиве. Подробное описание каждой части кода:

1. ‘indicator = [-1, -1, -1]‘: Создается список indicator, который содержит указатели на верхние элементы (границы) каждого из стеков. Изначально все указатели установлены в -1, что означает, что стеки пусты.
2. ‘size = [10, 10, 10]‘: Создается список size, который содержит размеры каждого из стеков.
3. ‘buff = [None] * sum(size)‘: Создается массив buff с общим размером, равным сумме размеров стеков. Каждый элемент массива инициализируется значением None.
4. ‘push(Num, value)‘: Функция push используется для добавления элемента value в стек с номером Num. Сначала происходит проверка на наличие свободного места в стеке. Если свободного места нет, выводится информация об этом. Затем вычисляется индекс для добавления элемента в общий массив buff, увеличивается указатель стека, и значение value сохраняется в соответствующем индексе.
5. ‘look(Num)‘: Функция look возвращает значение верхнего элемента стека с номером Num, но не удаляет его. Вычисляется индекс верхнего элемента в общем массиве buff, и значение этого элемента возвращается.
6. ‘emp(Num)‘: Функция emp проверяет, является ли стек с номером Num пустым. Если указатель стека равен 0, то стек считается пустым, и функция возвращает True. В противном случае, функция возвращает False.
7. ‘delet(Num)‘: Функция delet используется для удаления и возврата верхнего элемента из стека с номером Num. Сначала происходит проверка на пустоту стека. Если стек пуст, выводится информация об этом. Затем вычисляется индекс верхнего элемента в общем массиве buff, уменьшается указатель стека, значение верхнего элемента сохраняется, а соответствующий элемент в buff становится равным None. Затем возвращается сохраненное значение верхнего элемента.



```
indicator = [-1, -1, -1]  # указатели для отслеживания верхних элементов(границ)
size = [10, 10, 10]
buff = [None] * sum(size)


def push(Num, value):
    global buff
    global indicator
    global size
    if (indicator[Num] >= size[Num]):
        print("Нет необходимого пространства.")
        return
    index = Num * size[Num] + indicator[Num] + 1# Находим индекс верхнего элемента массива прибавляем 1, и увеличиваем указатель стека
    indicator[Num] += 1
    buff[index] = value

def look(Num):
    global size
    index = Num * size[Num] + indicator[Num]
    return buff[index]
```

Рисунок 5.1 — Код функций для задачи 5

```
def look(Num):
    global size
    index = Num * size[Num] + indicator[Num]
    return buff[index]


def emp(Num):
    global indicator
    return indicator[Num] == 0


def delet(Num):
    global indicator
    global size
    if (indicator[Num] == -1):
        print("Использование пустого стека")
        return
    index = Num * size[Num] + indicator[Num]
    indicator[Num] -= 1
    value = buff[index]
    buff[index] = None
    return value
```

Рисунок 5.2 — Продолжение кода функций для задачи 5

6 Задача 6

Код реализует экспоненциальный фильтр. Экспоненциальный фильтр применяется для сглаживания временных рядов или шумовых сигналов путем усреднения предыдущего значения с текущим значением с использованием весового коэффициента alpha.

Подробное описание кода:

1. Определение функции exponential filter(input value, output value, alpha), которая принимает входное значение input value, текущее значение output value и коэффициент сглаживания alpha.
2. Внутри функции вычисляется новое значение output value с помощью формулы экспоненциального фильтра: output value = alpha * input value + (1 - alpha) * output value.
3. Функция возвращает новое значение output value.
4. В основной части кода: Ввод значений input value, output value и alpha с помощью функции input(). Каждый ввод должен быть числом с плавающей точкой.
5. Проверка условия $0 < \text{alpha} < 1$, чтобы убедиться, что коэффициент сглаживания alpha находится в допустимом диапазоне (открытый интервал от 0 до 1).
6. Если условие выполняется, вызывается функция exponential filter() с передачей введенных значений input value, output value и alpha в качестве аргументов. Результат выводится на экран.
7. Если условие не выполняется, выводится сообщение "Wrong alpha!".

```
def task_6():
    def exponential_filter(input_value, output_value, alpha):
        output_value = alpha * input_value + (1 - alpha) * output_value
        return output_value

    input_value = float(input('Input_value: '))
    output_value = float(input('Output_value: '))
    alpha = float(input('Alpha: '))
    if 0 < alpha < 1:
        print(exponential_filter(input_value, output_value, alpha))
    else:
        print('Wrong alpha!')
```

Рисунок 6.1 — Код функции для задачи 6

В данном примере мы задаем входное значение input value равным 10, текущее значение output value равным 5 и коэффициент сглаживания alpha равным 0.5. После выполнения алгоритма, получим следующий результат:

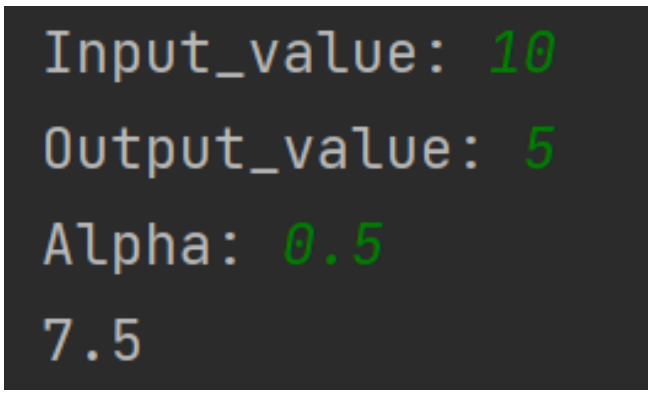


Рисунок 6.2 — Пример вывода функции для задачи 6

7 Задача 7

Алгоритм выполняет поиск наименьшего пропущенного числа в списке data. Он проверяет, какие числа от 1 до len(data) включительно присутствуют в списке, и находит первое пропущенное число. Если все числа уже присутствуют, то выводится наименьшее пропущенное число, равное len(data)+1

Подробное описание кода:

1. Создается пустой список data.
2. Генерируется случайное количество случайных чисел от 0 до 100, и каждое число добавляется в список data.
3. Выводится список data.
4. Создается список marked длиной len(data) + 1, заполненный логическими значениями False. Этот список будет использоваться для отметки чисел, которые уже встречались в data.
5. Проходит цикл по каждому числу num из списка data.
6. Если число num находится в диапазоне от 0 до len(data) включительно, устанавливается соответствующий элемент marked[num] в значение True. Это отмечает, что число num было обнаружено в data.
7. Проходит цикл по числам от 1 до len(data) + 1.
8. Если элемент marked[i] равен False, выводится сообщение о наименьшем пропущенном числе i. Затем устанавливается флаг flag в значение 1 и происходит выход из цикла.
9. Если флаг flag равен 0, значит все числа от 1 до len(data) включительно присутствуют в data. В этом случае выводится сообщение о наименьшем пропущенном числе, равном len(data) + 1.

```python
def task_7():
    data = []
    flag = 0
    for i in range(random.randint(0, 100)):
        data.append(random.randint(0, 50))
    print(data)
    marked = [False] * (len(data) + 1)
    for num in data:
        if 0 <= num <= len(data):
            marked[num] = True
    for i in range(1, len(data) + 1):
        if not marked[i]:
            print('Наименьшее пропущенное число:', i)
            flag = 1
            break
    if flag == 0:
        print('Наименьшее пропущенное число:', len(data) + 1)
```

Рисунок 7.1 — Код функции для задачи 7

```
[64, 64, 7, 19, 21, 55, 57, 16, 48, 41, 76, 92, 36, 13, 41,
Наименьшее пропущенное число: 3
```

Рисунок 7.2 — Пример вывода функции для задачи 7

```
92, 2, 1, 82, 97, 25, 94, 41, 88, 38, 42, 10, 7,
```

Рисунок 7.3 — Пример вывода функции для задачи 7

ЗАКЛЮЧЕНИЕ

Таким образом, были разработаны программы на языке Python с применением методов динамического программирования для решения различных задач. Изучены новые подходы и методы. В результате, для каждой задачи были представлены программы и их вывод. Все программы можно найти на репозитории в GitHub [1].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GitHub [Электронный ресурс]: `https://github.com/NatalyaPetrova/Algoritms` (дата обращения 28.05.2023).