Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ, информационные технологии»**

Лабораторная работа №5

«Модели вычислительных алгоритмов»

ДИСЦИПЛИНА: «Моделирование»

Выполнил: студент гр. ИУК4-72Б ______________ (Сафронов Н.С.)
(подпись) (Ф.И.О.)

Проверил: ______________ (Никитенко У.В.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2023

Цель работы: изучение технологии математического моделирования вычислительных алгоритмов, моделирование вычислительного алгоритма для оценки его трудоемкости, реализация математической модели на ЭВМ.

Постановка задачи

1. Построить по таблице 1, в соответствии с вариантом задания, граф алгоритма.
2. Построить математическую модель вычислительного процесса для оценки трудоемкости алгоритма по методу теории марковских цепей.
3. Построить математическую модель вычислительного процесса для оценки трудоемкости алгоритма сетевым методом.
4. Подготовить программу для расчета модельных характеристик трудоемкости на одном из языков высокого уровня.
5. Подготовить в объектно-ориентированной среде разработки интерактивную форму для управления работой программы и визуализации полученных результатов.

Вариант 3

Таблица 1

Вариант графа алгоритма

Вариант 3	P_{12}	P_{13}	P_{14}	P_{23}	P_{24}	P_{25}	P_{34}	P_{35}	P_{36}	P_{37}	P_{45}
	1.0			0.1	0.3	0.6			1.0		
	P_{46}	P_{47}	P_{56}	P_{57}	P_{61}	P_{67}	P_{68}	P_{71}	P_{72}	P_{78}	P_{81}
	1.0		1.0		0.9	0.1					

Таблица 2

Тип и трудоёмкость операторов

	1	2	3	4	5	6	7	8
Вариант 3	120	120	150	200	300	600	300	-

где затемнённые ячейки – ячейки, которые являются операторами ввода/вывода.

Ход выполнения работы

Оценка трудоёмкости алгоритма по методу теории марковских цепей

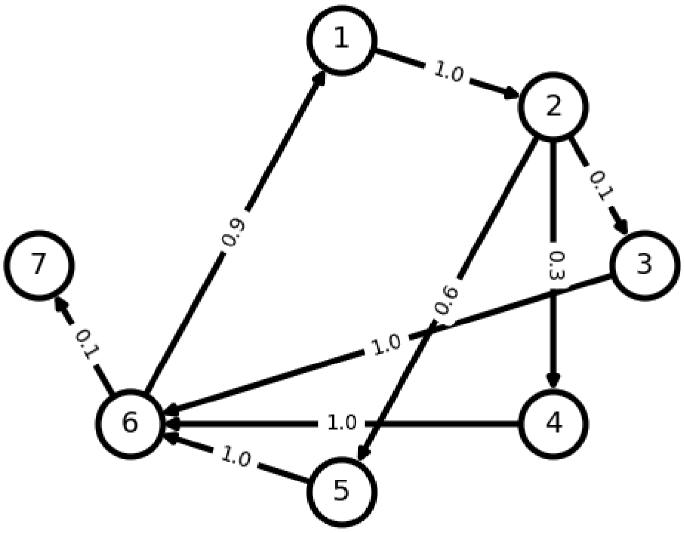

Рисунок 1 - Граф задания

Матрица смежности:

n1	n2	n3	n4	n5	n6	n7	n8
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.1	0.3	0.6	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Рисунок 2 - Матрица смежности графа

Изменённая матрица для построения СЛАУ:

n1	n2	n3	n4	n5	n6	n7	n8
-1.0	0.0	0.0	0.0	0.0	0.9	0.0	0.0
1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.1	-1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.3	0.0	-1.0	0.0	0.0	0.0	0.0
0.0	0.6	0.0	0.0	-1.0	0.0	0.0	0.0
0.0	0.0	1.0	1.0	1.0	-1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.1	-1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	-1.0

Рисунок 3 - Преобразованная матрица смежности для построения СЛАУ

```
Система линейных алгебраических уравнений:
-n1+0.9*n6 = -1
n1-n2 = 0
0.1*n2-n3 = 0
0.3*n2-n4 = 0
0.6*n2-n5 = 0
n3+n4+n5-n6 = 0
0.1*n6-n7 = 0
-n8 = 0
```

Рисунок 4 - Полученная СЛАУ

```
Решение СЛАУ:
n1 = 9.999999999999943
n2 = 9.999999999999941
n3 = 0.9999999999999947
n4 = 2.9999999999999813
n5 = 5.999999999999958
n6 = 9.99999999999994
n7 = 0.9999999999999911
n8 = 0.0
```

Рисунок 5 - Решение полученной СЛАУ

```
Трудоёмкость по основным операторам составляет: 9900 операций

Трудоёмкость по операторам ввода/вывода составляет: 1350 байт
```

Рисунок 6 - Результат вычисления трудоёмкости

Оценка трудоёмкости алгоритма сетевым методом

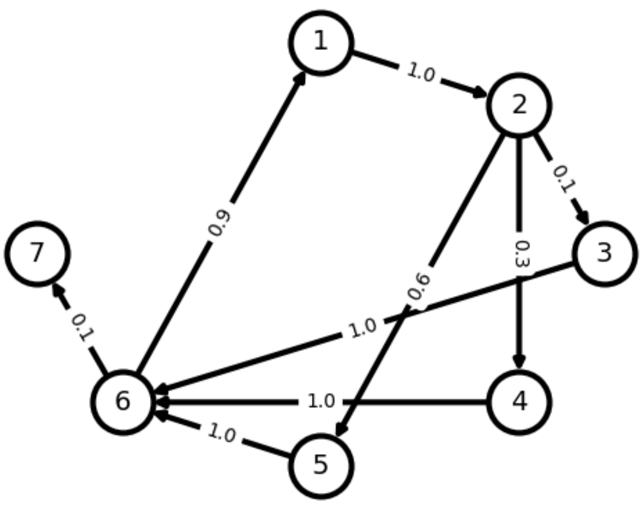

Рисунок 7 - Граф задания

Матрица смежности:

n1	n2	n3	n4	n5	n6	n7	n8
0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.1	0.3	0.6	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Рисунок 8 - Матрица смежности графа

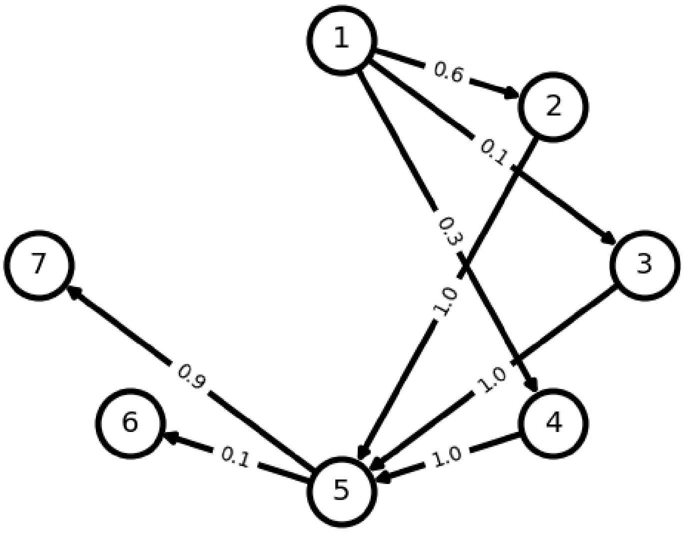

Рисунок 9 - Преобразованный граф

Матрица смежности:

n1	n2	n3	n4	n5	n6	n7	n8
0.0	0.6	0.1	0.3	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.1	0.9	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Рисунок 10 - Матрица смежности преобразованного графа

```
Трудоёмкость по основным операторам составляет: 9900 операций

Трудоёмкость по операторам ввода/вывода составляет: 1350 байт
```

Рисунок 11 - Результат вычисления трудоёмкости

Полученные результаты вычислений при помощи метода теории марковских цепей, а также сетевым методом тождественны. Из этого можно сделать вывод, что результаты исчислений трудоёмкостей верны.

Вывод: в ходе выполнения лабораторной работы были сформированы практические навыки технологии математического моделирования вычислительных алгоритмов, моделирования вычислительного алгоритма для оценки его трудоемкости.

ПРИЛОЖЕНИЯ

Листинг программы

Задание 1

```
from prettytable import PrettyTable
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt


class Matrix:
    __NO_WAY = 0.0
    __SIZE = 8

    def __init__(self,
                 p12=__NO_WAY, p13=__NO_WAY, p14=__NO_WAY,
                 p23=__NO_WAY, p24=__NO_WAY, p25=__NO_WAY,
                 p34=__NO_WAY,       p35=__NO_WAY,       p36=__NO_WAY,
p37=__NO_WAY,
                 p45=__NO_WAY, p46=__NO_WAY, p47=__NO_WAY,
                 p56=__NO_WAY, p57=__NO_WAY,
                 p61=__NO_WAY, p67=__NO_WAY, p68=__NO_WAY,
                 p71=__NO_WAY, p72=__NO_WAY, p78=__NO_WAY,
                 p81=__NO_WAY):
        self.__matrix = np.zeros((self.__SIZE, self.__SIZE))
        self.__matrix[0, 1] = p12
        self.__matrix[0, 2] = p13
        self.__matrix[0, 3] = p14
        self.__matrix[1, 2] = p23
        self.__matrix[1, 3] = p24
        self.__matrix[1, 4] = p25
        self.__matrix[2, 3] = p34
        self.__matrix[2, 4] = p35
        self.__matrix[2, 5] = p36
        self.__matrix[2, 6] = p37
        self.__matrix[3, 4] = p45
        self.__matrix[3, 5] = p46
        self.__matrix[3, 6] = p47
        self.__matrix[4, 5] = p56
        self.__matrix[4, 6] = p57
        self.__matrix[5, 0] = p61
        self.__matrix[5, 6] = p67
        self.__matrix[5, 7] = p68
        self.__matrix[6, 0] = p71
        self.__matrix[6, 1] = p72
        self.__matrix[6, 7] = p78
        self.__matrix[7, 0] = p81

    def print_matrix_base(self):
        names = []
        table = PrettyTable()
        for i in range(self.__SIZE):
            table.add_row(self.__matrix[i])
            names.append("n" + str(i + 1))
```

```
        table.field_names = names
        print("Матрица смежности:")
        print(table)
        print()

    def print_matrix_modav(self):
        names = []
        table = PrettyTable()
        matrix = self.__matrix.transpose().copy()
        for i in range(self.__SIZE):
            matrix[i, i] = -1.0
        for i in range(self.__SIZE):
            table.add_row(matrix[i])
            names.append("n" + str(i + 1))
        table.field_names = names
        print("Изменённая матрица для построения СЛАУ:")
        print(table)
        print()

    def __generate_slau_matrix(self):
        matrix_transpose = self.__matrix.transpose()
        for i in range(self.__SIZE):
            matrix_transpose[i, i] = -1.0
        matrix = [[str()] * self.__SIZE for i in range(self.__SIZE)]
        for i in range(self.__SIZE):
            for j in range(self.__SIZE):
                if matrix_transpose[i, j] != self.__NO_WAY:
                    matrix[i][j] = str(matrix_transpose[i, j]) + "*n"
+ str(j + 1)
                else:
                    matrix[i][j] = ""
        return matrix

    def __generate_slau(self):
        matrix = self.__generate_slau_matrix()
        slau = []
        for i in range(self.__SIZE):
            slau.append("+".join(matrix[i]).replace("++++++++",
"+++++++").replace("+++++++", "++++++") \
                        .replace("++++++",    "+++++").replace("+++++",
"++++").replace("++++", "+++") \
                        .replace("+++",            "++").replace("++",
"+").replace("+-", "-").replace("1.0", "") \
                        .replace("-*",               "-").replace("+*",
"+").removeprefix("+").removesuffix("+").removeprefix("*"))
        slau[0] += " = -1"
        for i in range(self.__SIZE - 1):
            slau[i + 1] += " = 0"
        return slau

    def print_slau(self):
        slau = self.__generate_slau()
        print("Система линейных алгебраических уравнений:")
        for i in range(len(slau)):
            print(slau[i])
        print()
```

```
    def __solve_slau(self):
        matrix = self.__matrix.transpose()
        for i in range(self.__SIZE):
            matrix[i, i] = -1.0
        vector = [-1.0]
        for i in range(self.__SIZE - 1):
            vector.append(0.0)
        vector = np.array(vector)
        answer = np.linalg.lstsq(matrix, vector, rcond=None)
        return answer[0]

    def print_sovle(self):
        sovle = self.__solve_slau()
        print("Решение СЛАУ:")
        for i in range(len(sovle)):
            print("n" + str(i + 1) + " = " + str(sovle[i]))
        print()

    def  __get_laboriousness(self,  k1=0.0,  k2=0.0,  k3=0.0,  k4=0.0,
k5=0.0, k6=0.0, k7=0.0, k8=0.0):
        sovle = self.__solve_slau()
        k = np.array((k1, k2, k3, k4, k5, k6, k7, k8))
        sum = 0.0
        for i in range(len(k)):
            sum += k[i] * sovle[i]
        return sum

    def print_laboriousness_base(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0,
k5=0.0, k6=0.0, k7=0.0, k8=0.0):
        sum = self.__get_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)
        print("Трудоёмкость  по  основным  операторам  составляет:  "  +
str(round(sum)) + " операций")
        print()

    def  print_laboriousness_io(self,  k1=0.0,  k2=0.0,  k3=0.0,  k4=0.0,
k5=0.0, k6=0.0, k7=0.0, k8=0.0):
        sum = self.__get_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)
        print("Трудоёмкость по операторам ввода/вывода составляет: " +
str(round(sum)) + " байт")
        print()

    def print_graph(self):
        G = nx.DiGraph()
        for i in range(self.__SIZE):
            for j in range(self.__SIZE):
                if self.__matrix[i, j] != self.__NO_WAY:
                    G.add_edge(i  +  1,  j  +  1,  weight=self.__matrix[i,
j])
        options = {
            "font_size": 14,
            "node_size": 1000,
            "node_color": "white",
            "edgecolors": "black",
            "linewidths": 3,
            "width": 3,
        }
```

```
        pos = {1: (0.0, 1.0), 2: (0.7, 0.7), 3: (1.0, 0.0), 4: (0.7, -
0.7),
               5: (0.0, -1.0), 6: (-0.7, -0.7), 7: (-1.0, 0.0), 8: (-
0.7, 0.7)}
        nx.draw_networkx(G, pos, **options)
        edges = list(G.edges.data("weight"))
        edge_labels = dict()
        for i in range(len(edges)):
            edge_labels.update({(edges[i][0],           edges[i][1]):
edges[i][2]})
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
        plt.axis("off")
        plt.show()


if __name__ == '__main__':
    m = Matrix(p12=1.0, p23=0.1, p24=0.3, p25=0.6, p36=1.0, p46=1.0,
p56=1.0, p61=0.9, p67=0.1)
    m.print_matrix_base()
    m.print_matrix_modav()
    m.print_graph()
    m.print_slau()
    m.print_sovle()
    m.print_laboriousness_base(k1=120, k4=200, k5=300, k6=600, k7=300)
    m.print_laboriousness_io(k2=120, k3=150)
```

Задание 2

```
from prettytable import PrettyTable
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np


class Matrix:
    __NO_WAY = 0.0
    __SIZE = 8

    def __init__(self,
                 p12=__NO_WAY, p13=__NO_WAY, p14=__NO_WAY,
                 p23=__NO_WAY, p24=__NO_WAY, p25=__NO_WAY,
                 p34=__NO_WAY,       p35=__NO_WAY,       p36=__NO_WAY,
p37=__NO_WAY,
                 p45=__NO_WAY, p46=__NO_WAY, p47=__NO_WAY,
                 p56=__NO_WAY, p57=__NO_WAY,
                 p61=__NO_WAY, p67=__NO_WAY, p68=__NO_WAY,
                 p71=__NO_WAY, p72=__NO_WAY, p78=__NO_WAY,
                 p81=__NO_WAY):
        self.__matrix = np.zeros((self.__SIZE, self.__SIZE))
        self.__matrix[0, 1] = p12
        self.__matrix[0, 2] = p13
        self.__matrix[0, 3] = p14
        self.__matrix[1, 2] = p23
        self.__matrix[1, 3] = p24
        self.__matrix[1, 4] = p25
        self.__matrix[2, 3] = p34
        self.__matrix[2, 4] = p35
```

```
        self.__matrix[2, 5] = p36
        self.__matrix[2, 6] = p37
        self.__matrix[3, 4] = p45
        self.__matrix[3, 5] = p46
        self.__matrix[3, 6] = p47
        self.__matrix[4, 5] = p56
        self.__matrix[4, 6] = p57
        self.__matrix[5, 0] = p61
        self.__matrix[5, 6] = p67
        self.__matrix[5, 7] = p68
        self.__matrix[6, 0] = p71
        self.__matrix[6, 1] = p72
        self.__matrix[6, 7] = p78
        self.__matrix[7, 0] = p81
        self.__G = nx.DiGraph()
        self.__removed_edge_weight = 0.0
        self.__map = {}
        for i in range(self.__SIZE):
            for j in range(self.__SIZE):
                if self.__matrix[i, j] != self.__NO_WAY:
                    self.__G.add_edge(i + 1, j + 1, weight=self.__matrix[i, j])

    def print_matrix(self):
        names = []
        table = PrettyTable()
        for i in range(self.__SIZE):
            table.add_row(self.__matrix[i])
            names.append("n" + str(i + 1))
        table.field_names = names
        print("Матрица смежности:")
        print(table)
        print()

    def print_graph(self):
        options = {
            "font_size": 14,
            "node_size": 1000,
            "node_color": "white",
            "edgecolors": "black",
            "linewidths": 3,
            "width": 3,
        }
        pos = {1: (0.0, 1.0), 2: (0.7, 0.7), 3: (1.0, 0.0), 4: (0.7, -0.7),
               5: (0.0, -1.0), 6: (-0.7, -0.7), 7: (-1.0, 0.0), 8: (-0.7, 0.7)}
        nx.draw_networkx(self.__G, pos, **options)
        edges = list(self.__G.edges.data("weight"))
        edge_labels = dict()
        for i in range(len(edges)):
            edge_labels.update({(edges[i][0], edges[i][1]): edges[i][2]})
        nx.draw_networkx_edge_labels(self.__G, pos, edge_labels=edge_labels)
        plt.axis("off")
        plt.show()
```

```
    def refactory_graph(self, map):
        self.__map = map
        self.__G = nx.relabel_nodes(self.__G, self.__map)
        for edge in nx.edges(self.__G).data("weight"):
            G = self.__G.copy()
            G.remove_edge(edge[0], edge[1])
            if len(list(nx.simple_cycles(G))) == 0:
                self.__G = G
                self.__removed_edge_weight = edge[2]
                break
        edges = list(self.__G.edges.data("weight"))
        for i in range(self.__SIZE):
            for j in range(self.__SIZE):
                self.__matrix[i, j] = 0.0
        for edge in edges:
            self.__matrix[edge[0] - 1, edge[1] - 1] = edge[2]

    def __find_n(self):
        n = np.zeros(self.__SIZE)
        n[0] = 1.0
        for i in range(1, self.__SIZE):
            sum = 0.0
            for j in range(self.__SIZE):
                sum += self.__matrix[j, i] * n[j]
            n[i] = sum
        return n

    def __get_laboriousness(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0,
k5=0.0, k6=0.0, k7=0.0, k8=0.0):
        k_base = np.array((k1, k2, k3, k4, k5, k6, k7, k8))
        k_mapped = k_base.copy()
        for key in self.__map:
            k_mapped[self.__map[key] - 1] = k_base[key - 1]
        k_mapped = np.array(k_mapped)
        sum = 0.0
        n = self.__find_n()
        p = 1.0
        for i in range(self.__SIZE - 1, 0, -1):
            if n[i] != 0.0:
                p = n[i]
                n[i] = 1.0
                break
        for i in range(len(k_mapped)):
            sum += k_mapped[i] * n[i]
        sum /= (1.0 - (p * self.__removed_edge_weight))
        return sum

    def print_laboriousness_base(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0,
k5=0.0, k6=0.0, k7=0.0, k8=0.0):
        sum = self.__get_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)
        print("Трудоёмкость по основным операторам составляет: " +
str(round(sum)) + " операций")
        print()

    def print_laboriousness_io(self, k1=0.0, k2=0.0, k3=0.0, k4=0.0,
k5=0.0, k6=0.0, k7=0.0, k8=0.0):
```

```
        sum = self.__get_laboriousness(k1, k2, k3, k4, k5, k6, k7, k8)
        print("Трудоёмкость по операторам ввода/вывода составляет: " +
str(round(sum)) + " байт")
        print()


if __name__ == '__main__':
    m = Matrix(p12=1.0, p23=0.1, p24=0.3, p25=0.6, p36=1.0, p46=1.0,
p56=1.0, p61=0.9, p67=0.1)
    m.print_graph()
    m.print_matrix()
    map = {2: 1, 5: 2, 6: 5, 7: 6, 1: 7}
    m.refactory_graph(map)
    m.print_graph()
    m.print_matrix()
    m.print_laboriousness_base(k1=120, k4=200, k5=300, k6=600, k7=300)
    m.print_laboriousness_io(k2=120, k3=150)
```