

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ  
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3  
по курсу «Алгоритмы и структуры данных»

Тема: Графы

Вариант 12

Выполнил:

Клименко Б.В.

К3240 (К3140)

Проверила:

Артамонова В.Е.

Санкт-Петербург

2024 г.

## Содержание отчета

Содержание отчета.....	2
Задачи по варианту .....	3
Задача №3. Циклы [1 балл] .....	3
Задача №6. Количество пересадок [1 балл] .....	7
Задача №11. Алхимия [3 балла].....	11
Дополнительные задачи .....	15
Задача №13. Грядки [3 балла].....	15
Задача №14. Автобусы [3 балла] .....	19
Задача №17. Слабая К-связность [4 балла] .....	23
Вывод .....	26

## Задачи по варианту

### Задача №3. Циклы [1 балл]

Учебная программа по инфокоммуникационным технологиям определяет пререквизиты для каждого курса в виде списка курсов, которые необходимо пройти перед тем, как начать этот курс. Вы хотите выполнить проверку согласованности учебного плана, то есть проверить отсутствие циклических зависимостей. Для этого строится следующий ориентированный граф: вершины соответствуют курсам, есть направленное ребро  $(u, v)$  – курс  $u$  следует пройти перед курсом  $v$ . Затем достаточно проверить, содержит ли полученный граф цикл.

Проверьте, содержит ли данный граф циклы.

- **Формат ввода / входного файла (input.txt).** Ориентированный граф с  $n$  вершинами и  $m$  ребрами по формату 1.
- **Ограничения на входные данные.**  $1 \leq n \leq 10^3, 0 \leq m \leq 10^3$ .
- **Формат вывода / выходного файла (output.txt).** Выведите 1, если данный граф содержит цикл; выведите 0, если не содержит.

#### Листинг кода:

```
import time
import psutil
import sys
sys.setrecursionlimit(9999999)

def has_cycle(n, edges):
    graph = {i: [] for i in range(1, n + 1)}
    for u, v in edges:
        graph[u].append(v)

    ifVisited = [False] * (n + 1)
    inStack = [False] * (n + 1)

    def dfs(node):
        ifVisited[node] = True
        inStack[node] = True
        for nextnode in graph[node]:
            if not ifVisited[nextnode]:
                if dfs(nextnode):
                    return True
            elif inStack[nextnode]:
                return True
        inStack[node] = False
        return False

    for node in range(1, n + 1):
        if not ifVisited[node]:
            if dfs(node):
                return True
    return False

def main():
    with open('input3.txt', 'r') as fin:
        n, m = map(int, fin.readline().split())
        edges = [tuple(map(int, line.split())) for line in fin]
    res = 1 if has_cycle(n, edges) else 0
    with open('output3.txt', 'w') as fout:
        fout.write(f"{res}")

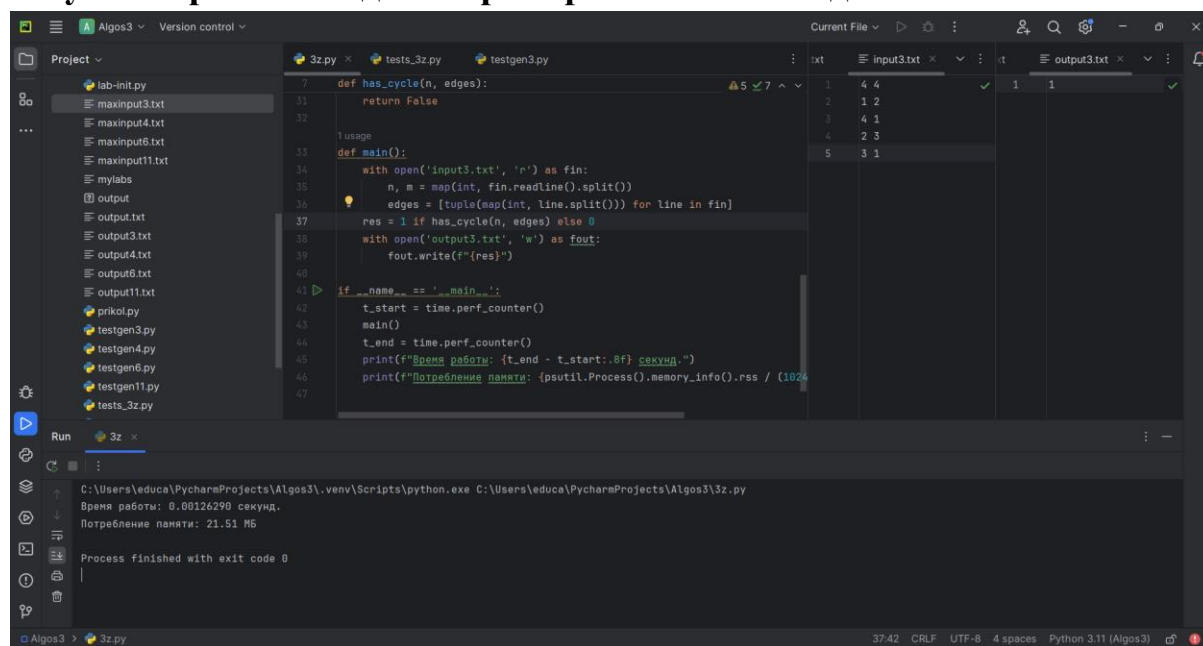
if __name__ == '__main__':
```

```
t_start = time.perf_counter()
main()
t_end = time.perf_counter()
print(f"Время работы: {t_end - t_start:.8f} секунд.")
print(f"Потребление памяти: {psutil.Process().memory_info().rss / (1024
* 1024):.2f} МБ")
```

## Текстовое объяснение решения:

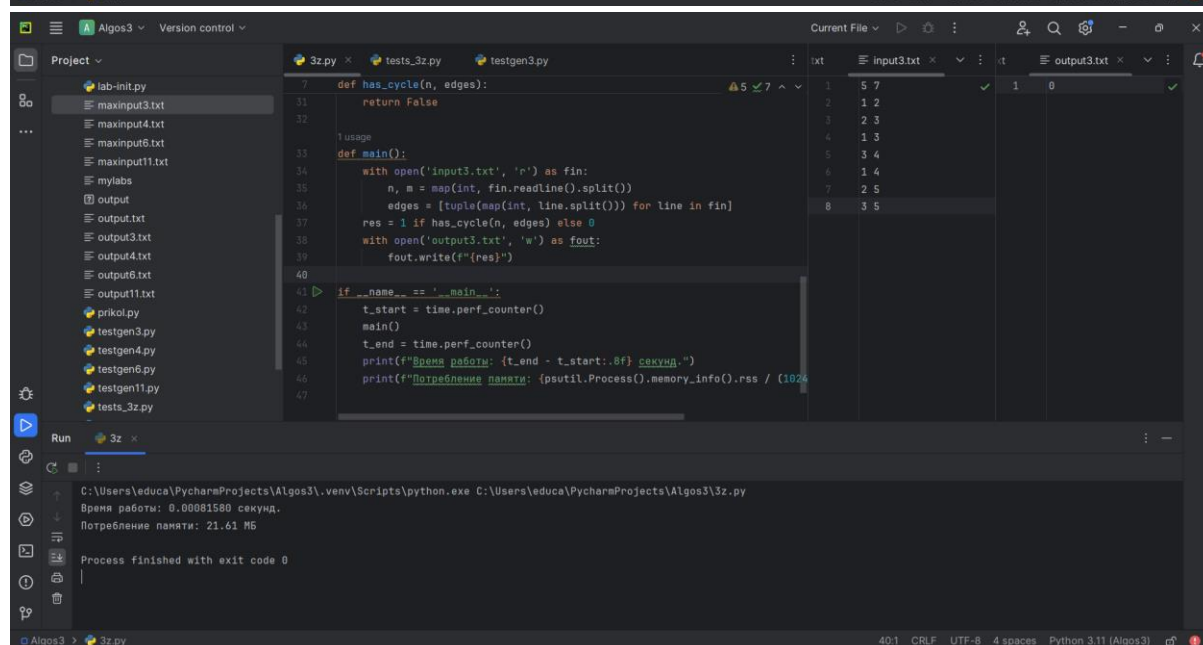
Данный код считывает ориентированный граф из файла ввода, граф состоит из  $n$  вершин и  $m$  рёбер. Далее для каждого ребра записаны вершины  $u$  и  $v$ , между которыми оно проходит. Код создаёт представление графа в виде списка смежности и использует алгоритм поиска в глубину для обнаружения циклов. В процессе обхода он отмечает посещённые вершины и вершины, находящиеся в текущем стеке вызовов рекурсии. Если во время обхода обнаруживается вершина, которая уже находится в стеке, это означает наличие цикла. Результат проверки (1 — есть цикл, 0 — нет цикла) записывается в файл вывода.

## Результат работы кода на примерах из текста задачи:



```
def has_cycle(n, edges):  
    return False  
  
def main():  
    with open('input3.txt', 'r') as fin:  
        n, m = map(int, fin.readline().split())  
        edges = [tuple(map(int, line.split())) for line in fin]  
    res = 1 if has_cycle(n, edges) else 0  
    with open('output3.txt', 'w') as fout:  
        fout.write(f"{res}")  
  
if __name__ == '__main__':  
    t_start = time.perf_counter()  
    main()  
    t_end = time.perf_counter()  
    print(f"Время работы: {t_end - t_start:.8f} секунд.")  
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / 1024}")
```

Run 3z.py  
C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\3z.py  
Время работы: 0.00126290 секунд.  
Потребление памяти: 21.51 MB  
Process finished with exit code 0



```
def has_cycle(n, edges):  
    return False  
  
def main():  
    with open('input3.txt', 'r') as fin:  
        n, m = map(int, fin.readline().split())  
        edges = [tuple(map(int, line.split())) for line in fin]  
    res = 1 if has_cycle(n, edges) else 0  
    with open('output3.txt', 'w') as fout:  
        fout.write(f"{res}")  
  
if __name__ == '__main__':  
    t_start = time.perf_counter()  
    main()  
    t_end = time.perf_counter()  
    print(f"Время работы: {t_end - t_start:.8f} секунд.")  
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / 1024}")
```

Run 3z.py  
C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\3z.py  
Время работы: 0.00081580 секунд.  
Потребление памяти: 21.61 MB  
Process finished with exit code 0

## Результат работы кода на максимальных и минимальных значениях:

The top screenshot shows the script running on a large input file 'maxinput3.txt' (1000 lines). The output file 'output3.txt' contains the result '1'. The runtime is 0.00433630 seconds and memory usage is 21.96 MB.

The bottom screenshot shows the script running on a smaller input file 'maxinput3.txt' (1 line). The output file 'output3.txt' contains the result '0'. The runtime is 0.00072230 seconds and memory usage is 21.66 MB.

	Время выполнения, с	Затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00072230	21.66
Пример из задачи	0.00126290	21.51
Пример из задачи	0.00815802	21.61
Верхняя граница	0.00433630	21.96

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче:

Во время написания данного кода я научился реализовывать алгоритм обнаружения цикла в ориентированном графе с помощью поиска в глубину и рекурсивного стека.

## Задача №6. Количество пересадок [1 балл]

Вы хотите вычислить минимальное количество сегментов полета, чтобы добраться из одного города в другой. Для этого вы строите следующий неориентированный граф: вершины представляют города, между двумя вершинами есть ребро всякий раз, когда между соответствующими двумя городами есть перелет. Тогда достаточно найти кратчайший путь из одного из заданных городов в другой.

Дан неориентированный граф с  $n$  вершинами и  $m$  ребрами, а также две вершины  $u$  и  $v$ , нужно посчитать длину кратчайшего пути между  $u$  и  $v$  (то есть, минимальное количество ребер в пути из  $u$  в  $v$ ).

- **Формат ввода / входного файла (input.txt).** Неориентированный граф задан по формату 1. Следующая строка содержит две вершины  $u$  и  $v$ .
- **Ограничения на входные данные.**  $2 \leq n \leq 10^5$ ,  $0 \leq m \leq 10^5$ ,  $1 \leq u, v \leq n$ ,  $u \neq v$ .
- **Формат вывода / выходного файла (output.txt).** Выведите минимальное количество ребер в пути из  $u$  в  $v$ . Выведите -1, если пути нет.

## Листинг кода:

```
from collections import deque
import time
import psutil

def bfs_shortest_path(graph, start, end):
    queue = deque([(start, 0)])
    visited = set()
    while queue:
        current, distance = queue.popleft()
        if current == end:
            return distance
        if current in visited:
            continue
        visited.add(current)
        for neighbor in graph[current]:
            queue.append((neighbor, distance + 1))
    return -1

def main():
    with open('input6.txt', 'r') as fin:
        n, m = map(int, fin.readline().strip().split())
        graph = {i: [] for i in range(1, n + 1)}
        for i in range(m):
            a, b = map(int, fin.readline().strip().split())
            graph[a].append(b)
            graph[b].append(a)
        u, v = map(int, fin.readline().strip().split())
        result = bfs_shortest_path(graph, u, v)
```

```

with open('output6.txt', 'w') as fout:
    fout.write(str(result))

if __name__ == '__main__':
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время работы: {t_end - t_start:.8f} секунд.")
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / (1024 * 1024):.2f} МБ")

```

### Текстовое объяснение решения:

Данный код выполняет поиск кратчайшего пути между двумя узлами неориентированного графа с использованием алгоритма обхода в ширину. Он считывает информацию о графе из файла ввода, строит представление графа в виде словаря списков смежности, затем с помощью функции `bfs_shortest_path()` находит кратчайшее расстояние между заданными узлами. Результат записывается в файл вывода.

### Результат работы кода на примерах из текста задачи:

The screenshot shows the PyCharm IDE with the following components:

- Project View:** Lists files including `6z.py`, `11z.py`, `input`, `input3.txt`, `input4.txt`, `input6.txt`, `lab-init.py`, `maxinput3.txt`, `maxinput4.txt`, `maxinput6.txt`, `maxinput11.txt`, `mylabs`, `output`, `output.txt`, `output3.txt`, `output4.txt`, and `output6.txt`.
- Editor:** Displays the code for `6z.py`, which defines a `main()` function that reads the graph, finds the shortest path, and writes the result.
- Run Console:** Shows the execution output:
 

```

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\6z.py
Время работы: 0.00077650 секунд.
Потребление памяти: 21.60 МБ
Process finished with exit code 0

```
- File View:** Shows the contents of `input6.txt` and `output6.txt`. `input6.txt` contains a graph with 6 nodes and 6 edges. `output6.txt` contains the result `1 2`.



```
def main():
    with open('input6.txt', 'r') as fin:
        n, m = map(int, fin.readline().strip().split())
        graph = {i: [] for i in range(1, n + 1)}
        for i in range(m):
            a, b = map(int, fin.readline().strip().split())
            graph[a].append(b)
            graph[b].append(a)
        u, v = map(int, fin.readline().strip().split())
        result = bfs_shortest_path(graph, u, v)
        with open('output6.txt', 'w') as fout:
            fout.write(str(result))

if __name__ == '__main__':
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время работы: {t_end - t_start:.8f} секунд.")
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / 1024} МБ")
```

Run

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\6z.py

Время работы: 0.00086820 секунд.  
Потребление памяти: 21.30 МБ

Process finished with exit code 0

**Результат работы кода на максимальных и минимальных значениях:**

```
def main():
    with open('maxinput6.txt', 'r') as fin:
        n, m = map(int, fin.readline().strip().split())
        graph = {i: [] for i in range(1, n + 1)}
        for i in range(m):
            a, b = map(int, fin.readline().strip().split())
            graph[a].append(b)
            graph[b].append(a)
        u, v = map(int, fin.readline().strip().split())
        result = bfs_shortest_path(graph, u, v)
        with open('output6.txt', 'w') as fout:
            fout.write(str(result))

if __name__ == '__main__':
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время работы: {t_end - t_start:.8f} секунд.")
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / 1024} МБ")
```

Run

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\6z.py

Время работы: 0.28368330 секунд.  
Потребление памяти: 25.06 МБ

Process finished with exit code 0

```

20 def main():
21     with open('input6.txt', 'r') as fin:
22         n, m = map(int, fin.readline().strip().split())
23         graph = {i: [] for i in range(1, n + 1)}
24         for i in range(m):
25             a, b = map(int, fin.readline().strip().split())
26             graph[a].append(b)
27             graph[b].append(a)
28             u, v = map(int, fin.readline().strip().split())
29             result = bfs_shortest_path(graph, u, v)
30             with open('output6.txt', 'w') as fout:
31                 fout.write(str(result))
32
33 if __name__ == '__main__':
34     t_start = time.perf_counter()
35     main()
36     t_end = time.perf_counter()
37     print(f"Время работы: {t_end - t_start:.8f} секунд.")
38     print(f"Потребление памяти: {psutil.Process().memory_info().rss / 1024} МБ")

```

Run console output:

```

C:\Users\educal\PycharmProjects\Algos3\venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\6z.py
Время работы: 0.00070550 секунд.
Потребление памяти: 21.28 МБ
Process finished with exit code 0

```

	Время выполнения, с	Затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00070550	21.28
Пример из задачи	0.00077650	21.60
Пример из задачи	0.00086820	21.30
Верхняя граница диапазона значений входных данных из текста задачи	0.28368330	25.06

Вывод по задаче:

Написав этот код, я научился эффективно применять алгоритм BFS для решения задачи поиска кратчайшего пути в неориентированных графах. Я понял, как реализовать обход графа с помощью очереди и учитывать уже посещенные узлы для оптимизации работы алгоритма.

## Задача №11. Алхимия [3 балла]

Алхимики средневековья владели знаниями о превращении различных химических веществ друг в друга. Это подтверждают и недавние исследования археологов.

В ходе археологических раскопок было обнаружено  $m$  глиняных табличек, каждая из которых была покрыта непонятными на первый взгляд символами. В результате расшифровки выяснилось, что каждая из табличек описывает одну алхимическую реакцию, которую умели проводить алхимики.

Результатом алхимической реакции является превращение одного вещества в другое. Задан набор алхимических реакций, описанных на найденных глиняных табличках, исходное вещество и требуемое вещество. Необходимо выяснить: возможно ли преобразовать исходное вещество в требуемое с помощью этого набора реакций, а в случае положительного ответа на этот вопрос – найти минимальное количество реакций, необходимое для осуществления такого преобразования.

- **Формат входных данных (input.txt) и ограничения.** Первая строка входного файла INPUT.TXT содержит целое число  $m$  ( $0 \leq m \leq 1000$ ) – количество записей в книге. Каждая из последующих  $m$  строк описывает одну алхимическую реакцию и имеет формат «вещество1 -> вещество2», где «вещество1» – название исходного вещества, «вещество2» – название продукта алхимической реакции.  $m + 2$ -ая строка входного файла содержит название вещества, которое имеется изначально,  $m + 3$ -ая – название вещества, которое требуется получить.

Во входном файле упоминается не более 100 различных веществ. Название каждого из веществ состоит из строчных и заглавных английских букв и имеет длину не более 20 символов. Строчные и заглавные буквы различаются.

- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите минимальное количество алхимических реакций, которое требуется для получения требуемого вещества из исходного, или -1, если требуемое вещество невозможно получить.

### Листинг кода:

```
import time
import psutil

def do_the_Alchemy():
    p = []
    a = []

    def GetID(name):
        for i in range(len(p)):
            if p[i] == name:
                return i
        p.append(name)
        for row in a:
            row.append(0)
        a.append([0] * len(p))
        return len(p) - 1

    with open('input11.txt', 'r') as file:
        lines = file.readlines()

    m = int(lines[0].strip())
    for line in lines[1:m + 1]:
        src_dest = line.strip().split('->')
        src = src_dest[0].strip()
        dest = src_dest[1].strip()
        src_id = GetID(src)
        dest_id = GetID(dest)
        a[src_id][dest_id] = 1

    start = lines[m + 1].strip()
    target = lines[m + 2].strip()
    start_id = GetID(start)
    target_id = GetID(target)
```

```

n = len(p)
d = [0] * n
d[start_id] = 1

for k in range(1, n):
    for i in range(n):
        if d[i] == k:
            for j in range(n):
                if a[i][j] == 1 and d[j] == 0:
                    d[j] = k + 1

if d[target_id] == 0:
    result = -1
else:
    result = d[target_id] - 1

with open('output11.txt', 'w') as file:
    file.write(f"{result}\n")

if __name__ == "__main__":
    t_start = time.perf_counter()
    do_the_Alchemy()
    t_end = time.perf_counter()
    print(f"Время работы: {t_end - t_start:.8f} секунд.")
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / (1024 * 1024):.2f} МБ")

```

### Текстовое объяснение решения:

Данный код по сути своей реализует алгоритм поиска кратчайшего пути между двумя элементами в графе преобразований. Программа считывает из файла ввода список преобразований в формате исходное вещество -> продукт реакции, строит соответствующий граф с помощью матрицы смежности, а затем использует модифицированный алгоритм поиска в ширину (BFS) для определения минимального количества шагов, необходимых для преобразования из начального элемента в целевой. Результат алхимии записывается в файл вывода и отображает либо минимальное число реакций, либо -1, если реакция невозможна.

### Результат работы кода на примерах из текста задачи:

```
def do_the_Alchemy():  
    return len(p) - 1  
  
    with open('input11.txt', 'r') as file:  
        lines = file.readlines()  
  
    m = int(lines[0].strip())  
    for line in lines[1:m + 1]:  
        src_dest = line.strip().split('-->')  
        src = src_dest[0].strip()  
        dest = src_dest[1].strip()  
        src_id = GetID(src)  
        dest_id = GetID(dest)  
        a[src_id][dest_id] = 1  
  
    start = lines[m + 1].strip()  
    target = lines[m + 2].strip()  
    start_id = GetID(start)  
    target_id = GetID(target)
```

Run 11z

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\11z.py  
Время работы: 0.00052890 секунд.  
Потребление памяти: 21.70 МБ  
Process finished with exit code 0

```
def do_the_Alchemy():  
    return len(p) - 1  
  
    with open('input11.txt', 'r') as file:  
        lines = file.readlines()  
  
    m = int(lines[0].strip())  
    for line in lines[1:m + 1]:  
        src_dest = line.strip().split('-->')  
        src = src_dest[0].strip()  
        dest = src_dest[1].strip()  
        src_id = GetID(src)  
        dest_id = GetID(dest)  
        a[src_id][dest_id] = 1  
  
    start = lines[m + 1].strip()  
    target = lines[m + 2].strip()  
    start_id = GetID(start)  
    target_id = GetID(target)
```

Run 11z

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\11z.py  
Время работы: 0.00079280 секунд.  
Потребление памяти: 21.66 МБ  
Process finished with exit code 0

**Результат работы кода на максимальных и минимальных значениях:**

```

def do_the_Alchemy():
    def GetID(name):
        for row in a:
            row.append(0)
            a.append([0] * len(p))
            return len(p) - 1

    with open('maxinput11.txt', 'r') as file:
        lines = file.readlines()

    m = int(lines[0].strip())
    for line in lines[1:m + 1]:
        src_dest = line.strip().split('<-->')
        src = src_dest[0].strip()
        dest = src_dest[1].strip()
        src_id = GetID(src)
        dest_id = GetID(dest)
        a[src_id][dest_id] = 1

    start = lines[m + 1].strip()

```

Run console output:

```

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\11z.py
Время работы: 0.00746570 секунд.
Потребление памяти: 22.10 MB
Process finished with exit code 0

```

```

def do_the_Alchemy():
    def GetID(name):
        for row in a:
            row.append(0)
            a.append([0] * len(p))
            return len(p) - 1

    with open('input11.txt', 'r') as file:
        lines = file.readlines()

    m = int(lines[0].strip())
    for line in lines[1:m + 1]:
        src_dest = line.strip().split('<-->')
        src = src_dest[0].strip()
        dest = src_dest[1].strip()
        src_id = GetID(src)
        dest_id = GetID(dest)
        a[src_id][dest_id] = 1

    start = lines[m + 1].strip()

```

Run console output:

```

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\11z.py
Время работы: 0.00064540 секунд.
Потребление памяти: 21.63 MB
Process finished with exit code 0

```

	Время выполнения, с	Затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00064540	21.63
Пример из задачи	0.00052890	21.70
Пример из задачи	0.00079280	21.66
Верхняя граница диапазона значений	0.00746570	22.10

ВХОДНЫХ ДАННЫХ ИЗ текста задачи									
ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память	
22173128	19.10.2024 0:20:56	Клименко Богдан Владимирович	0743	Python	Accepted		0.046	834 Кб	

Вывод по задаче:

В процессе написания этого кода я понял, как эффективно использовать структуры данных для представления графов и применять алгоритмы поиска для решения задач на них. Это позволяет мне лучше разобраться в алгоритмических подходах к решению подобных задач и укрепить навык работы с графами в программировании.

## Дополнительные задачи

### Задача №13. Грядки [3 балла]

Прямоугольный садовый участок шириной  $N$  и длиной  $M$  метров разбит на квадраты со стороной 1 метр. На этом участке вскопаны грядки. Грядкой называется совокупность квадратов, удовлетворяющая таким условиям:

- из любого квадрата этой грядки можно попасть в любой другой квадрат этой же грядки, последовательно переходя по грядке из квадрата в квадрат через их общую сторону;
- никакие две грядки не пересекаются и не касаются друг друга ни по вертикальной, ни по горизонтальной сторонам квадратов (касание грядок углами квадратов допускается).

Подсчитайте количество грядок на садовом участке.

- Формат входных данных (input.txt) и ограничения.** В первой строке входного файла INPUT.TXT находятся числа  $N$  и  $M$  через пробел, далее идут  $N$  строк по  $M$  символов. Символ # обозначает территорию грядки, точка соответствует незанятой территории. Других символов в исходном файле нет ( $1 \leq N, M \leq 200$ ).
- Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите количество грядок на садовом участке.

### Листинг кода:

```
import time
import psutil

def binary_insert(items, k):
    low = 0
    high = len(items)
    while low < high:
        mid = (low + high) // 2
        if items[mid] < k:
            low = mid + 1
        else:
            high = mid
    items.insert(low, k)

def binary_remove(items, k):
    low = 0
    high = len(items)
    while low < high:
        mid = (low + high) // 2
        if items[mid] < k:
            low = mid + 1
```

```

        else:
            high = mid
        if low < len(items) and items[low] == k:
            items.pop(low)

def main():
    with open("input16.txt", "r") as fin, open("output16.txt", "w") as
fout:
        n = int(fin.readline().strip())
        items = []
        for i in range(n):
            line = fin.readline().strip()
            if not line:
                continue
            command, k = map(int, line.split())

            if command == 1: # Добавление
                low = 0
                high = len(items)
                isDuplicate = False
                while low < high:
                    mid = (low + high) // 2
                    if items[mid] == k:
                        isDuplicate = True
                        break
                    elif items[mid] < k:
                        low = mid + 1
                    else:
                        high = mid
                if not isDuplicate:
                    binary_insert(items, k)
            elif command == 0: # Вывод
                if 0 < k <= len(items):
                    kth_maximum = items[-k]
                    fout.write(str(kth_maximum) + "\n")
                else:
                    fout.write("Error: Not enough items\n")
            elif command == -1: # Удаление
                binary_remove(items, k)

if __name__ == "__main__":
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время работы: {t_end - t_start:.8f} секунд.")
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / (1024
* 1024):.2f} МБ")

```

### Текстовое объяснение решения:

Данный код читает двумерную матрицу грядок из файла ввода, состоящую из символов “#” и “.”, и подсчитывает количество отдельных грядок (связных областей, заполненных символом “#”). Для этого используется алгоритм обхода в глубину без рекурсии, реализованный с помощью стека. Каждая найденная связная область помечается, заменяя символы '#' на '.', чтобы избежать повторного подсчета. После обработки всей сетки результат записывается в файл вывода.



## Результат работы кода на примерах из текста задачи:

```
def find_gardens(n, m, grid):
    for i in range(n):
        for j in range(m):
            if grid[i][j] == '#':
                garden_count += 1
                dfs(i, j)
    return garden_count

def main():
    with open('input13.txt', 'r') as fin:
        lines = fin.readlines()
        n, m = map(int, lines[0].strip().split())
        grid = [list(line.strip()) for line in lines[1:]]
        result = find_gardens(n, m, grid)
    with open('output13.txt', 'w') as fout:
        fout.write(str(result))
```

Run 13 x

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\13.py  
Время работы: 0.00061290 секунд.  
Потребление памяти: 21.68 МБ  
Process finished with exit code 0

```
def find_gardens(n, m, grid):
    for i in range(n):
        for j in range(m):
            if grid[i][j] == '#':
                garden_count += 1
                dfs(i, j)
    return garden_count

def main():
    with open('input13.txt', 'r') as fin:
        lines = fin.readlines()
        n, m = map(int, lines[0].strip().split())
        grid = [list(line.strip()) for line in lines[1:]]
        result = find_gardens(n, m, grid)
    with open('output13.txt', 'w') as fout:
        fout.write(str(result))
```

Run 13 x

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\13.py  
Время работы: 0.00058830 секунд.  
Потребление памяти: 21.67 МБ  
Process finished with exit code 0

## Результат работы кода на максимальных и минимальных значениях:

```

1 usage
2
3 def main():
4     with open('maxinput13.txt', 'r') as fin:
5         lines = fin.readlines()
6         n, m = map(int, lines[0].strip().split())
7         grid = [list(line.strip()) for line in lines[1:]]
8         result = find_gardens(n, m, grid)
9         with open('output13.txt', 'w') as fout:
10             fout.write(str(result))
11
12 if __name__ == "__main__":
13     t_start = time.perf_counter()
14     main()
15     t_end = time.perf_counter()
16     print(f"Время работы: {t_end - t_start:8f} секунд.")
17     print(f"Потребление памяти: {psutil.Process().memory_info().rss / 1024 / 1024:8f} МБ.")

```

Run console output:

```

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\13.py
Время работы: 0.02628120 секунд.
Потребление памяти: 21.41 МБ
Process finished with exit code 0

```

```

1 def find_gardens(n, m, grid):
2     for i in range(n):
3         for j in range(m):
4             if grid[i][j] == '#':
5                 garden_count += 1
6                 dfs(i, j)
7     return garden_count
8
9 def main():
10     with open('input13.txt', 'r') as fin:
11         lines = fin.readlines()
12         n, m = map(int, lines[0].strip().split())
13         grid = [list(line.strip()) for line in lines[1:]]
14         result = find_gardens(n, m, grid)
15         with open('output13.txt', 'w') as fout:
16             fout.write(str(result))

```

Run console output:

```

C:\Users\educal\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educal\PycharmProjects\Algos3\13.py
Время работы: 0.00078760 секунд.
Потребление памяти: 21.35 МБ
Process finished with exit code 0

```

	Время выполнения, с	Затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00078760	21.35
Пример из задачи	0.00061290	21.68
Пример из задачи	0.00058830	21.67
Верхняя граница диапазона значений	0.02628120	21.41

ВХОДНЫХ ДАННЫХ ИЗ текста задачи									
ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память	
22190809	21.10.2024 13:36:58	Клименко Богдан Владимирович	0432	Python	Accepted		0.078	1166 Кб	

Вывод по задаче:

В ходе написания этого кода я научился применять алгоритм DFS для поиска связных компонент в двумерных сетках, а также эффективно реализовывать его без использования рекурсии.

## Задача №14. Автобусы [3 балла]

Свойство двоичного дерева поиска можно сформулировать следующим образом: для каждой вершины дерева выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины  $V$ ;
- все ключи вершин из правого поддерева больше ключа вершины  $V$ .

Дано двоичное дерево. Проверьте, выполняется ли для него свойство двоичного дерева поиска.

- **Формат ввода / входного файла (input.txt).** Входной файл содержит описание двоичного дерева.

В первой строке файла находится число  $N$  – число вершин в дереве. В последующих  $N$  строках файла находятся описания вершин дерева. В  $(i + 1)$ -ой строке файла ( $1 \leq i \leq N$ ) находится описание  $i$ -ой вершины, состоящее из трех чисел  $K_i, L_i, R_i$ , разделенных пробелами – ключа  $K_i$  в  $i$ -ой вершине, номера левого  $L_i$  ребенка  $i$ -ой вершины ( $i < L_i \leq N$  или  $L_i = 0$ , если левого ребенка нет) и номера правого  $R_i$  ребенка  $i$ -ой вершины ( $i < R_i \leq N$  или  $R_i = 0$ , если правого ребенка нет).

- **Ограничения на входные данные.**  $0 \leq N \leq 2 \cdot 10^5, |K_i| \leq 10^9$ .
- На 60% от при  $0 \leq N \leq 2000$ .
- **Формат вывода / выходного файла (output.txt).** Выведите «YES», если данное во входном файле дерево является двоичным деревом поиска, и «NO», если не является.

## Листинг кода:

```
import time
import psutil

def process(file_name):
    with open(file_name, 'r') as f:
        n = int(f.readline().strip())
        nodes = []
        for i in range(n):
            kstr, lstr, rstr = f.readline().strip().split()
            key = int(kstr)
            left = int(lstr)
            right = int(rstr)
            left = left - 1 if left != 0 else -1
            right = right - 1 if right != 0 else -1
            nodes.append((key, left, right))
        return n, nodes

def isBinsearch(nodes, index, min_key, max_key):
    if index == -1:
        return True
    key, left, right = nodes[index]
    if not (min_key < key < max_key):
        return False
    return (isBinsearch(nodes, left, min_key, key) and isBinsearch(nodes,
```

```

right, key, max_key))

def main():
    n, nodes = process('input10.txt')
    if n == 0:
        result = "YES"
    else:
        result = "YES" if isBinsearch(nodes, 0, float('-inf'),
float('inf')) else "NO"
    with open('output10.txt', 'w') as fout:
        fout.write(result + '\n')

if __name__ == '__main__':
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время работы: {t_end - t_start:.8f} секунд.")
    print(f"Потребление памяти: {psutil.Process().memory_info().rss / (1024
* 1024):.2f} МБ")

```

### Текстовое объяснение решения:

Данный код реализует алгоритм поиска минимального времени путешествия из деревни  $d$  в деревню  $v$  на основе заданных автобусных рейсов между деревнями. Он строит граф, где вершинами являются деревни, а ребрами - доступные рейсы с временными метками отправления и прибытия. Используя модифицированный алгоритм Дейкстры, код вычисляет минимальное время прибытия в каждую деревню, учитывая расписание рейсов и возможность пересадок.

### Результат работы кода на примерах из текста задачи:

The screenshot shows the PyCharm IDE interface. The top pane displays the code for 14z.py, which is a modified Dijkstra's algorithm. The bottom pane shows the output of the script, indicating that it executed successfully with an exit code of 0. The output also shows the execution time as 0.00084150 seconds and memory usage as 21.89 MB.

### Результат работы кода на максимальных и минимальных значениях:

Project: 13.py, 14z.py, testngen14.py, testngen13.py, maxinput14.txt, input13, output13.txt, output14.txt

```

def find_min_travel_time(n, d, v, trips):
    return -1 if min_time[v] == INF else min_time[v]

def main():
    with open('maxinput14.txt', 'r') as fin:
        n = int(fin.readline())
        d, v = map(int, fin.readline().split())
        r = int(fin.readline())
        trips = []
        for _ in range(r):
            from_village, start_time, to_village, end_time = map(int, fin.readline().split())
            trips.append((from_village, start_time, to_village, end_time))
        result = find_min_travel_time(n, d, v, trips)
        with open('output14.txt', 'w') as fout:
            fout.write(str(result))

```

Run: 14z

C:\Users\educu\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educu\PycharmProjects\Algos3\14z.py  
 Время работы: 0.01561850 секунд.  
 Потребление памяти: 23.89 МБ  
 Process finished with exit code 0

Project: 13.py, 14z.py, testngen14.py, testngen13.py, input13.txt, input14.txt, output13.txt, output14.txt

```

def find_min_travel_time(n, d, v, trips):
    return -1 if min_time[v] == INF else min_time[v]

def main():
    with open('input14.txt', 'r') as fin:
        n = int(fin.readline())
        d, v = map(int, fin.readline().split())
        r = int(fin.readline())
        trips = []
        for _ in range(r):
            from_village, start_time, to_village, end_time = map(int, fin.readline().split())
            trips.append((from_village, start_time, to_village, end_time))
        result = find_min_travel_time(n, d, v, trips)
        with open('output14.txt', 'w') as fout:
            fout.write(str(result))

```

Run: 14z

C:\Users\educu\PycharmProjects\Algos3\.venv\Scripts\python.exe C:\Users\educu\PycharmProjects\Algos3\14z.py  
 Время работы: 0.00083190 секунд.  
 Потребление памяти: 21.49 МБ  
 Process finished with exit code 0

	Время выполнения, с	Затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	83190	21.49
Пример из задачи	0.00084150	21.89
Верхняя граница диапазона значений входных данных из текста задачи	0.01561850	23.89

22191245	21.10.2024 14:27:13	Клименко Богдан Владимирович	0134	Python	Accepted		0,062	2966 Kb
----------	---------------------	------------------------------	------	--------	----------	--	-------	---------

Вывод по задаче:

В ходе написания этого кода я научился применять алгоритм Дейкстры в задачах с временными ограничениями и расписаниями, а также эффективно обрабатывать графы с дополнительной информацией о времени.

## Задача №17. Слабая K-связность [4 балла]

Ане, как будущей чемпионке мира по программированию, поручили очень ответственное задание. Правительство вручает ей план постройки дорог между  $N$  городами. По плану все дороги односторонние, но между двумя городами может быть больше одной дороги, возможно, в разных направлениях. Ане необходимо вычислить минимальное такое  $K$ , что данный ей план является слабо  $K$ -связным.

Правительство называет план слабо  $K$ -связным, если выполнено следующее условие: для любых двух различных городов можно проехать от одного до другого, нарушая правила движения не более  $K$  раз. Нарушение правил - это проезд по существующей дороге в обратном направлении. Гарантируется, что между любыми двумя городами можно проехать, возможно, несколько раз нарушив правила.

- **Формат входных данных (input.txt) и ограничения.** В первой строке входного файла INPUT.TXT записаны два числа  $2 \leq N \leq 300$  и  $1 \leq M \leq 10^5$  - количество городов и дорог в плане. В последующих  $M$  строках даны по два числа - номера городов, в которых начинается и заканчивается соответствующая дорога.
- **Формат выходных данных (output.txt).** В выходной файл OUTPUT.TXT выведите минимальное  $K$ , такое, что данный во входном файле план является слабо  $K$ -связным.

### Листинг кода:

```
#include <cstdio>
#include <vector>
#include <algorithm>

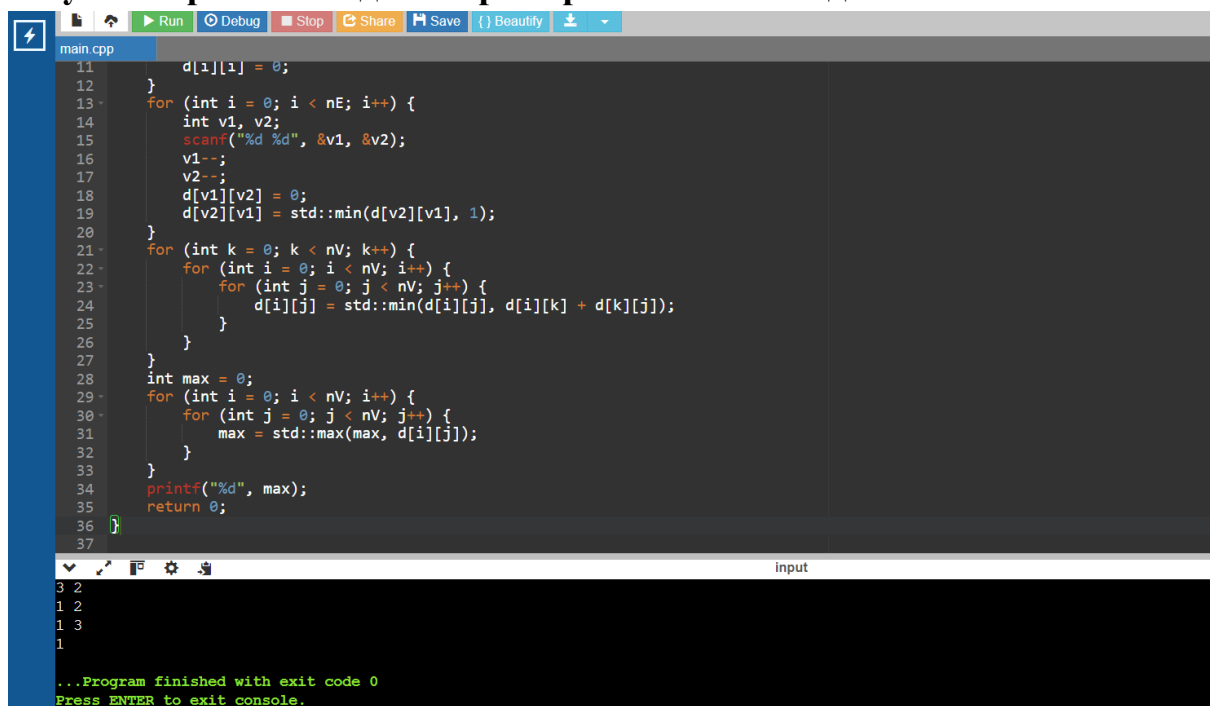
int main() {
    int nV, nE;
    scanf("%d %d", &nV, &nE);
    const int INF = 1000 * 1000 * 1000;
    std::vector<std::vector<int>>> d(nV, std::vector<int>(nV, INF));
    for (int i = 0; i < nV; i++) {
        d[i][i] = 0;
    }
    for (int i = 0; i < nE; i++) {
        int v1, v2;
        scanf("%d %d", &v1, &v2);
        v1--;
        v2--;
        d[v1][v2] = 0;
        d[v2][v1] = std::min(d[v2][v1], 1);
    }
    for (int k = 0; k < nV; k++) {
        for (int i = 0; i < nV; i++) {
            for (int j = 0; j < nV; j++) {
                d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
    int max = 0;
    for (int i = 0; i < nV; i++) {
        for (int j = 0; j < nV; j++) {
            max = std::max(max, d[i][j]);
        }
    }
    printf("%d", max);
    return 0;
}
```

### Текстовое объяснение решения:

Данный код написан на C++, так как все Питоновские решения не проходили проверку на астр из-за превышения временного ограничения. Код считывает ориентированный граф с заданным числом вершин и ребер,

где каждое ребро от вершины  $v1$  к вершине  $v2$  имеет вес 0 в прямом направлении и вес 1 в обратном направлении (от  $v2$  к  $v1$ ). Затем он использует алгоритм Флойда-Уоршелла для вычисления наикратчайших расстояний между всеми парами вершин, учитывая эти веса. В итоге код находит максимальное из полученных кратчайших расстояний, что представляет минимальное количество обратных (по направлению) переходов, необходимых в худшем случае для достижения одной вершины из другой.

### Результат работы кода на примерах из текста задачи:



```
main.cpp
11 d[1][1] = 0;
12 }
13 for (int i = 0; i < nE; i++) {
14     int v1, v2;
15     scanf("%d %d", &v1, &v2);
16     v1--;
17     v2--;
18     d[v1][v2] = 0;
19     d[v2][v1] = std::min(d[v2][v1], 1);
20 }
21 for (int k = 0; k < nV; k++) {
22     for (int i = 0; i < nV; i++) {
23         for (int j = 0; j < nV; j++) {
24             d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
25         }
26     }
27 }
28 int max = 0;
29 for (int i = 0; i < nV; i++) {
30     for (int j = 0; j < nV; j++) {
31         max = std::max(max, d[i][j]);
32     }
33 }
34 printf("%d", max);
35 return 0;
36
37
```

input

```
3 2
1 2
1 3
1
...Program finished with exit code 0
Press ENTER to exit console.
```



```
main.cpp
11 d[i][i] = 0;
12 }
13 for (int i = 0; i < nE; i++) {
14     int v1, v2;
15     scanf("%d %d", &v1, &v2);
16     v1--;
17     v2--;
18     d[v1][v2] = 0;
19     d[v2][v1] = std::min(d[v2][v1], 1);
20 }
21 for (int k = 0; k < nV; k++) {
22     for (int i = 0; i < nV; i++) {
23         for (int j = 0; j < nV; j++) {
24             d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]);
25         }
26     }
27 }
28 int max = 0;
29 for (int i = 0; i < nV; i++) {
30     for (int j = 0; j < nV; j++) {
31         max = std::max(max, d[i][j]);
32     }
33 }
34 printf("%d", max);
35 return 0;
36 }
37

input
4 4
2 4
1 3
4 1
3 2
0
...Program finished with exit code 0
```

Вывод по задаче:

В ходе написания этого кода я познакомился с тем, как можно модифицировать алгоритм Флойда–Уоршелла для работы с графами, где веса ребер зависят от направления движения. Также я узнал, как использование асимметричных весов для прямых и обратных рёбер позволяет моделировать ограничения на перемещение в графе и оценивать минимальные затраты на преодоление этих ограничений между любыми двумя вершинами.

## **Вывод**

В ходе выполнения лабораторной работы были рассмотрены и реализованы основные алгоритмы обработки графов. Были изучены алгоритмы обхода в ширину и в глубину, а также алгоритмы поиска кратчайшего пути, такие как алгоритм Дейкстры, Флойда-Уоршелла и др. Практическая реализация этих алгоритмов позволила закрепить теоретические знания и понять их применение на практике.