Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

**Лабораторна робота *№4***

із дисципліни ***«Теорія алгоритмів»***

Тема: ***«Теорія графів»***

**Виконали:**

Студенти групи ІА-34

Янович Марія,

Ковальчук Станіслав,

Ястремський Богдан,

Сухоручкін Гліб

**Перевірив:**

Степанов Андрій Сергійович

Київ — 2024

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | ЗМІСТ |  |
| [**1**](#page4) | [**МЕТА ЛАБОРАТОРНОЇ РОБОТИ**](#page4) | | [**..................................................... 3**](#page4) |
| [**2**](#page8) | [**ЗАВДАННЯ...............................................................................................**](#page8) | | [**4**](#page8) |
| [**3**](#page8) | [**ВИКОНАННЯ ..........................................................................................**](#page8) | | [**8**](#page8) |
|  | [3.1](#page8) | [ПСЕВДОКОД АЛГОРИТМУ ........................................................................](#page8) | [8](#page8) |
|  | [3.2](#page8) | [ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ](#page8) | [..................................................... 8](#page8) |
|  | [*3.2.1 Вихідний код ....................................................................................*](#page8) | | [*8*](#page8) |
| [**ВИСНОВОК...................................................................................................**](#page11) | | | [**10**](#page11) |
| **КРИТЕРІЇ ОЦІНЮВАННЯ........................................................................** | | | **11** |

2

* МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні прикладні алгоритми на графах та способи їх імплементації.

3

* ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

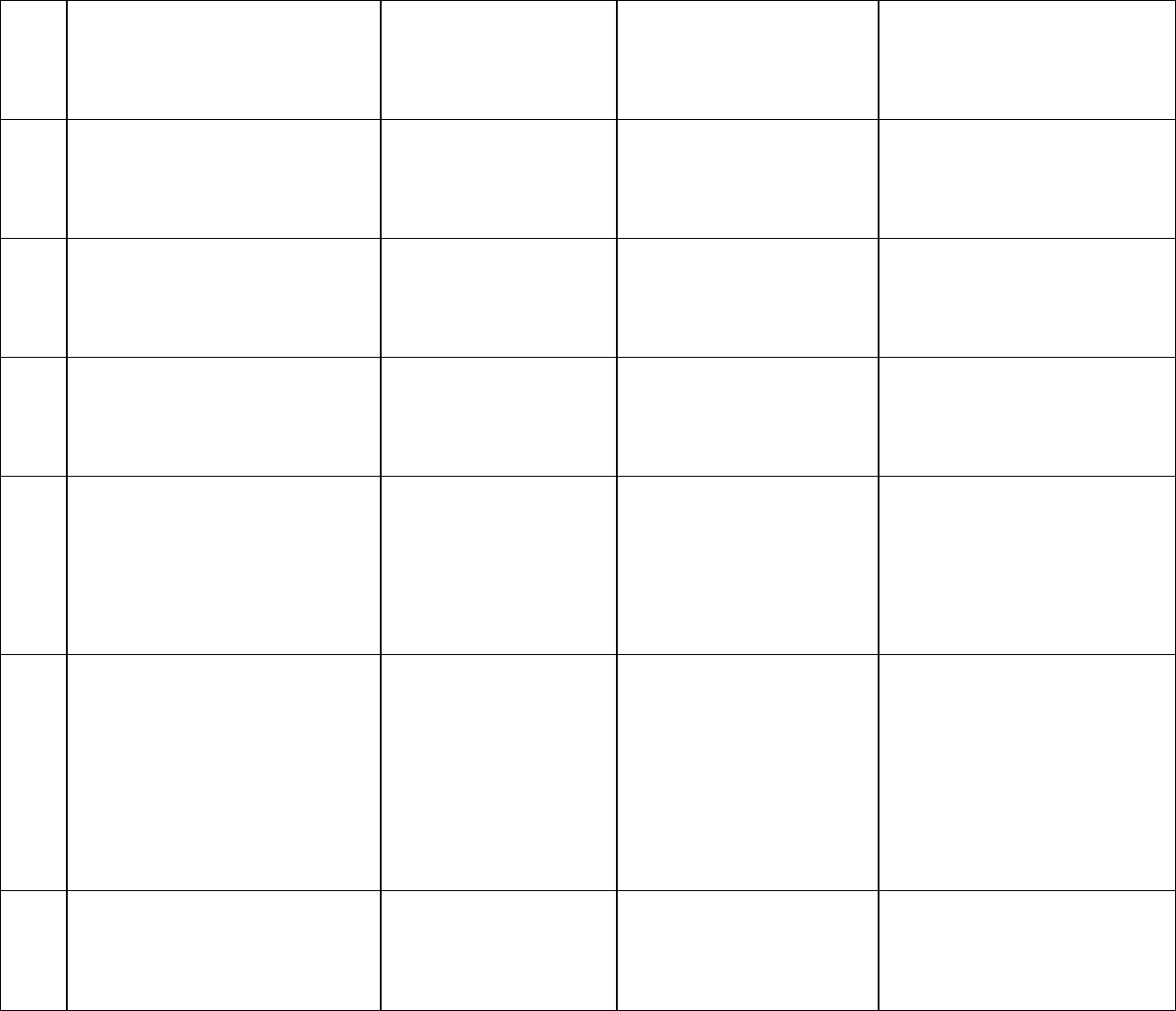
Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 9 вершин) розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.

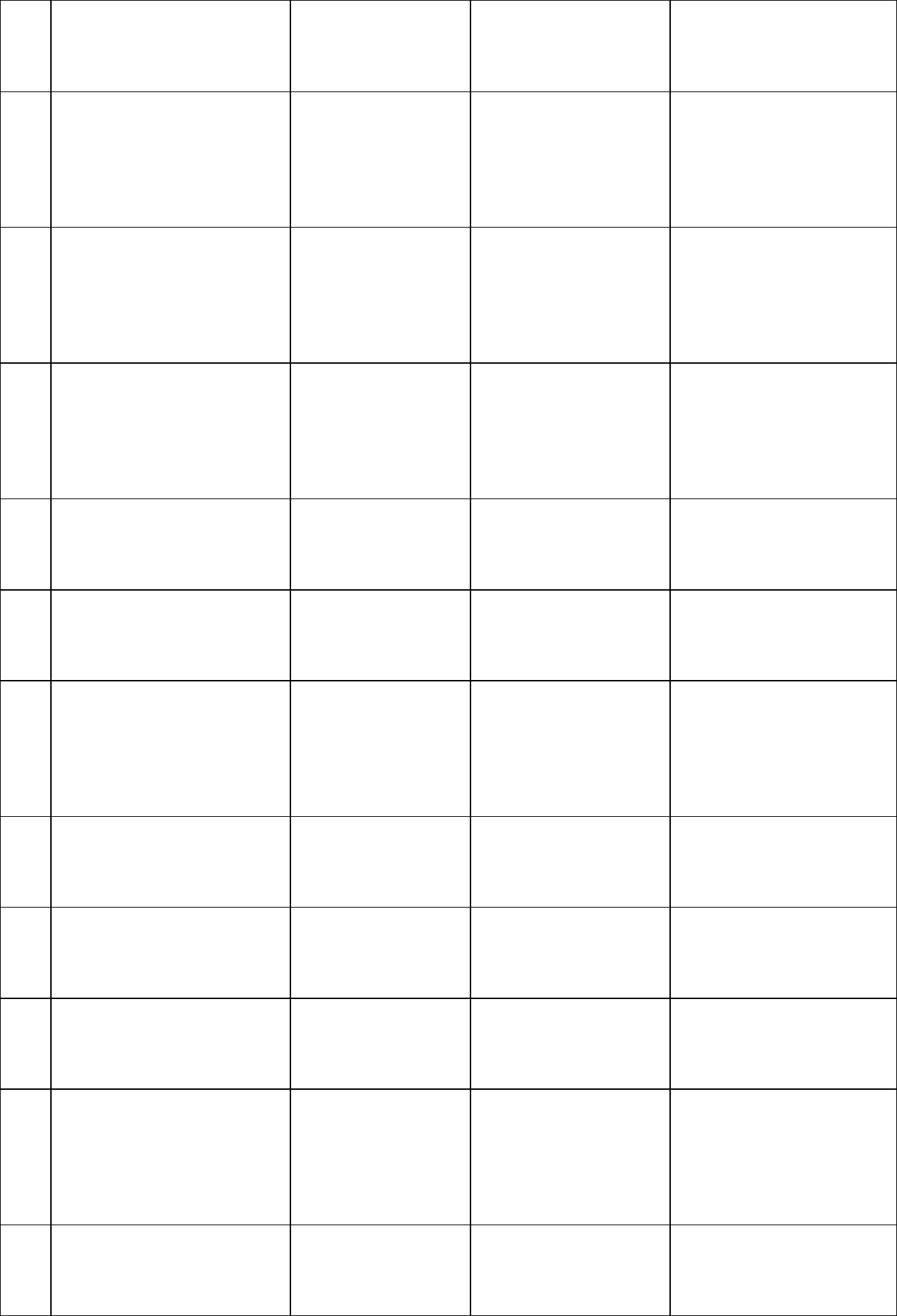
Таблиця 2.1 – Варіанти алгоритмів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Задача** | **Алгоритм** | **Тип графу** | **Спосіб задання** |
|  |  |  |  | **графу** |
| 1 | Обхід графу | DFS | Неорієнтований | Матриця |
|  |  |  |  | суміжності |
| 2 | Обхід графу | BFS | Неорієнтований | Матриця |
|  |  |  |  | суміжності |
| 3 | Пошук маршруту у | Террі | Неорієнтований | Матриця |
|  | графі |  |  | суміжності |
| 4 | Пошук відстані | Хвильовий | Неорієнтований | Матриця |
|  | між вершинами |  |  | суміжності |
|  | графа |  |  |  |
| **5** | **Пошук** | **Дейкстри** | **Орієнтований** | **Матриця вагів** |
|  | **найкоротшого** |  |  |  |
|  | **шляху між парою** |  |  |  |
|  | **вершин** |  |  |  |
| 6 | Пошук | Беллмана- | Орієнтований | Матриця вагів |
|  | найкоротшого | Форда |  |  |



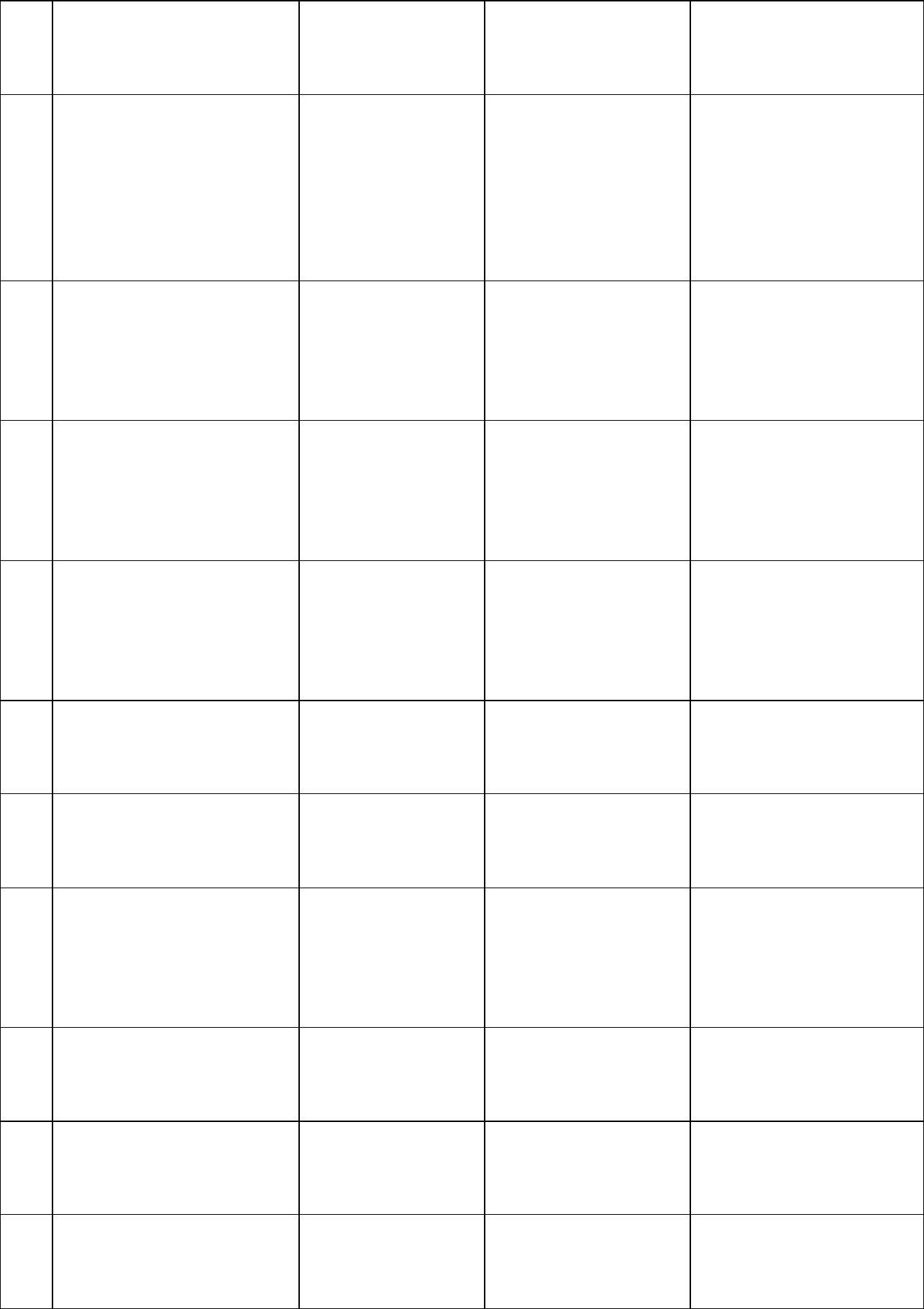
4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | шляху між парою |  |  |  |
|  | вершин |  |  |  |
| 7 | Побудова | Прима | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 8 | Побудова | Крускала | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 9 | Побудова | Борувки | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 10 | Побудова | За | Неорієнтований | Матриця |
|  | Ейлерового циклу | означенням |  | суміжності |
| 11 | Побудова | Флері | Неорієнтований | Матриця |
|  | Ейлерового циклу |  |  | суміжності |
| 12 | Побудова | Пошук із | Неорієнтований | Матриця |
|  | Гамільтонового | поверненнями |  | суміжності |
|  | циклу |  |  |  |
| 13 | Обхід графу | DFS | Неорієнтований | Матриця |
|  |  |  |  | інцидентності |
| 14 | Обхід графу | BFS | Неорієнтований | Матриця |
|  |  |  |  | інцидентності |
| 15 | Пошук маршруту у | Террі | Неорієнтований | Матриця |
|  | графі |  |  | інцидентності |
| 16 | Пошук відстані | Хвильовий | Неорієнтований | Матриця |
|  | між вершинами |  |  | інцидентності |
|  | графа |  |  |  |
| 17 | Пошук | Дейкстри | Орієнтований | Матриця вагів |
|  | найкоротшого |  |  |  |



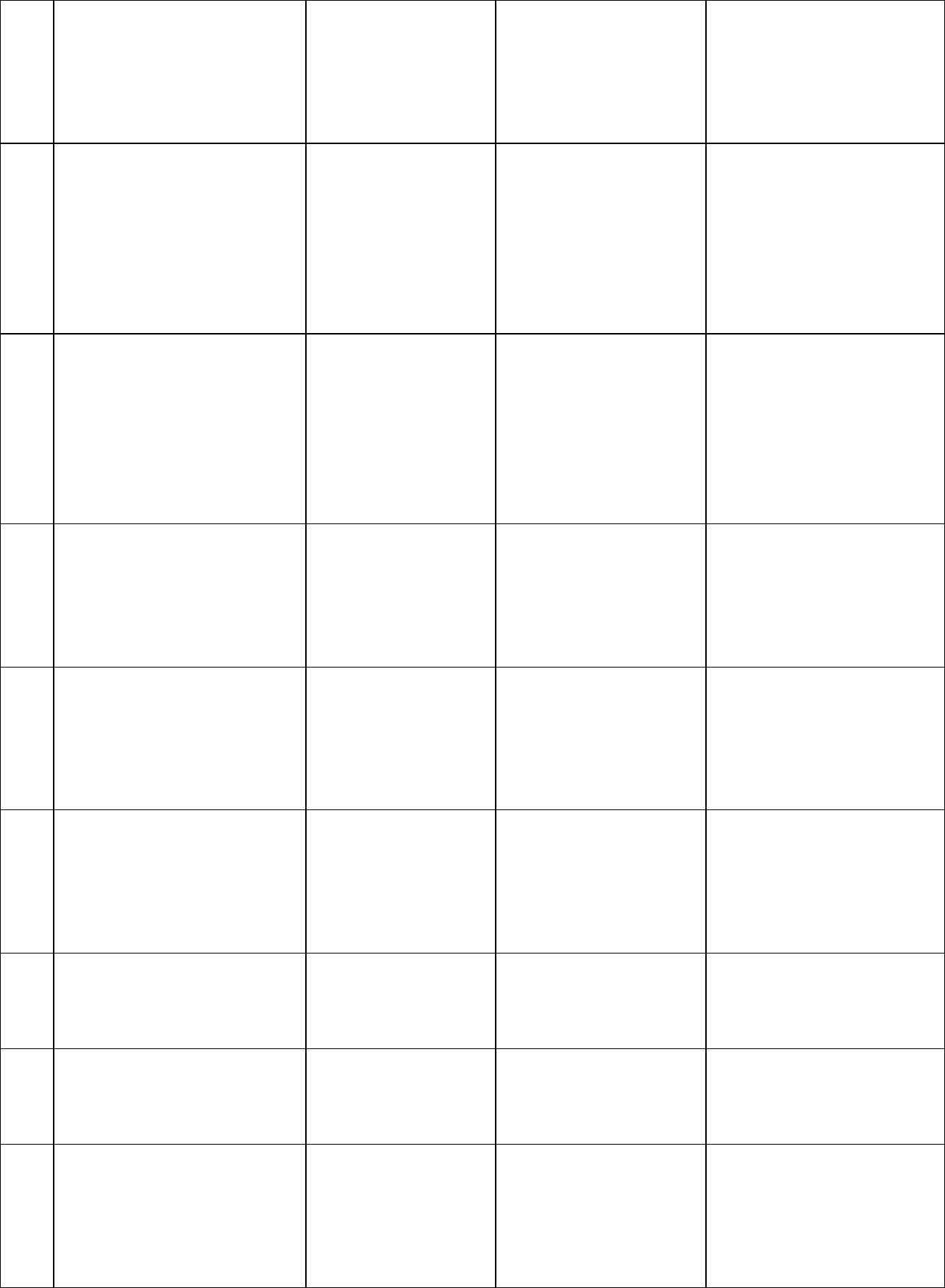
5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | шляху між парою |  |  |  |
|  | вершин |  |  |  |
| 18 | Пошук | Беллмана- | Орієнтований | Матриця вагів |
|  | найкоротшого | Форда |  |  |
|  | шляху між парою |  |  |  |
|  | вершин |  |  |  |
| 19 | Побудова | Прима | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 20 | Побудова | Крускала | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 21 | Побудова | Борувки | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 22 | Побудова | За | Неорієнтований | Матриця |
|  | Ейлерового циклу | означенням |  | інцидентності |
| 23 | Побудова | Флері | Неорієнтований | Матриця |
|  | Ейлерового циклу |  |  | інцидентності |
| 24 | Побудова | Пошук із | Неорієнтований | Матриця |
|  | Гамільтонового | поверненнями |  | інцидентності |
|  | циклу |  |  |  |
| 25 | Обхід графу | DFS | Неорієнтований | Матриця |
|  |  |  |  | суміжності |
| 26 | Обхід графу | BFS | Неорієнтований | Матриця |
|  |  |  |  | суміжності |
| 27 | Пошук маршруту у | Террі | Неорієнтований | Матриця |
|  | графі |  |  | суміжності |



6

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 28 | Пошук відстані | Хвильовий | Неорієнтований | Матриця |
|  | між вершинами |  |  | суміжності |
|  | графа |  |  |  |
| 29 | Пошук | Дейкстри | Орієнтований | Матриця вагів |
|  | найкоротшого |  |  |  |
|  | шляху між парою |  |  |  |
|  | вершин |  |  |  |
| 30 | Пошук | Беллмана- | Орієнтований | Матриця вагів |
|  | найкоротшого | Форда |  |  |
|  | шляху між парою |  |  |  |
|  | вершин |  |  |  |
| 31 | Побудова | Прима | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 32 | Побудова | Крускала | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 33 | Побудова | Борувки | Неорієнтований | Матриця вагів |
|  | мінімальних |  |  |  |
|  | покриваючих дерев |  |  |  |
| 34 | Побудова | За | Неорієнтований | Матриця |
|  | Ейлерового циклу | означенням |  | суміжності |
| 35 | Побудова | Флері | Неорієнтований | Матриця |
|  | Ейлерового циклу |  |  | суміжності |
| 36 | Побудова | Пошук із | Неорієнтований | Матриця |
|  | Гамільтонового | поверненнями |  | суміжності |
|  | циклу |  |  |  |



7

* ВИКОНАННЯ

3.1 Псевдокод алгоритму

class Graph:

function \_\_init\_\_:

edges <- empty dictionary

weights <- empty dictionary

function add\_edge(from\_node, to\_node, weight):

append to\_node to edges[from\_node]

set weights[(from\_node, to\_node)] to weight

function dijkstra(graph, initial, end):

shortest\_paths <- dictionary with initial: (None, 0)

current\_node <- initial

visited <- empty set

while current\_node != end:

add current\_node to visited

destinations <- graph.edges[current\_node]

weight\_to\_current\_node <- shortest\_paths[current\_node][1]

for each next\_node in destinations:

weight <- graph.weights[(current\_node, next\_node)] + weight\_to\_current\_node

if next\_node not in shortest\_paths or shortest\_paths[next\_node][1] > weight:

shortest\_paths[next\_node] <- (current\_node, weight)

next\_destinations <- {node: shortest\_paths[node] for node in shortest\_paths if node not in visited}

if next\_destinations is empty:

return "Route Not Possible"

current\_node <- node in next\_destinations with minimum tentative weight

path <- empty list

while current\_node is not None:

append current\_node to path

next\_node <- shortest\_paths[current\_node][0]

current\_node <- next\_node

path <- reverse(path)

path\_cost <- shortest\_paths[end][1]

return (path, path\_cost)

function matrix\_to\_edges(inp\_matrix):

edges <- empty list

for i in range of length(inp\_matrix):

for j in range of length(inp\_matrix[i]):

if inp\_matrix[i][j] > 0:

append (str(i), str(j), inp\_matrix[i][j]) to edges

return edges

# Example input data

input\_matrix <- [

[0, 4, 0, 0, 0, 0, 0, 8, 0],

[4, 0, 8, 0, 0, 0, 0, 11, 0],

[0, 8, 0, 7, 0, 4, 0, 0, 2],

[0, 0, 7, 0, 9, 14, 0, 0, 0],

[0, 0, 0, 9, 0, 10, 0, 0, 0],

[0, 0, 4, 14, 10, 0, 2, 0, 0],

[0, 0, 0, 0, 0, 2, 0, 1, 6],

[8, 11, 0, 0, 0, 0, 1, 0, 7],

[0, 0, 2, 0, 0, 0, 6, 7, 0]

]

# Create graph from matrix

graph <- Graph()

for each edge in matrix\_to\_edges(input\_matrix):

graph.add\_edge(edge[0], edge[1], edge[2])

# Find shortest path

find\_from <- "3"

find\_to <- "7"

print(dijkstra(graph, find\_from, find\_to))



3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

from collections import defaultdict

class Graph():

def \_\_init\_\_(self):

self.edges = defaultdict(list)

self.weights = {}

def add\_edge(self, from\_node, to\_node, weight):

self.edges[from\_node].append(to\_node)

self.weights[(from\_node, to\_node)] = weight

def dijsktra(graph, initial, end):

shortest\_paths = {initial: (None, 0)}

current\_node = initial

visited = set()

while current\_node != end:

visited.add(current\_node)

destinations = graph.edges[current\_node]

weight\_to\_current\_node = shortest\_paths[current\_node][1]

for next\_node in destinations:

weight = graph.weights[(current\_node, next\_node)] + weight\_to\_current\_node

if next\_node not in shortest\_paths:

shortest\_paths[next\_node] = (current\_node, weight)

else:

current\_shortest\_weight = shortest\_paths[next\_node][1]

if current\_shortest\_weight > weight:

shortest\_paths[next\_node] = (current\_node, weight)

next\_destinations = {node: shortest\_paths[node] for node in shortest\_paths if node not in visited}

if not next\_destinations:

return "Route Not Possible"

current\_node = min(next\_destinations, key=lambda k: next\_destinations[k][1])

path = []

while current\_node is not None:

path.append(current\_node)

next\_node = shortest\_paths[current\_node][0]

current\_node = next\_node

path = path[::-1]

path\_cost = shortest\_paths[end][1]

return path, path\_cost

# Example input data:

input\_matrix = [

[0, 4 , 0, 0 , 0 , 0 , 0, 8 , 0],

[4, 0 , 8, 0 , 0 , 0 , 0, 11, 0],

[0, 8 , 0, 7 , 0 , 4 , 0, 0 , 2],

[0, 0 , 7, 0 , 9 , 14, 0, 0 , 0],

[0, 0 , 0, 9 , 0 , 10, 0, 0 , 0],

[0, 0 , 4, 14, 10, 0 , 2, 0 , 0],

[0, 0 , 0, 0 , 0 , 2 , 0, 1 , 6],

[8, 11, 0, 0 , 0 , 0 , 1, 0 , 7],

[0, 0 , 2, 0 , 0 , 0 , 6, 7 , 0]

]

def matrix\_to\_edges(inp\_matrix):

edges = []

l = len(inp\_matrix)

for i in range(len(inp\_matrix)):

ii = inp\_matrix[i]

if len(ii) != l:

raise Exception("Incorrect input was provided.")

for j in range(len(ii)):

jj = ii[j]

if jj <= 0:

continue

edges.append((str(i), str(j), jj))

return edges

graph = Graph()

for edge in matrix\_to\_edges(input\_matrix):

graph.add\_edge(\*edge)

# Example input data:

find\_from = "3"

find\_to = "7"

print(dijsktra(graph, find\_from, find\_to))

3.2.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.

Рисунок 3.1



(від 1 до 4)

Рисунок 3.2



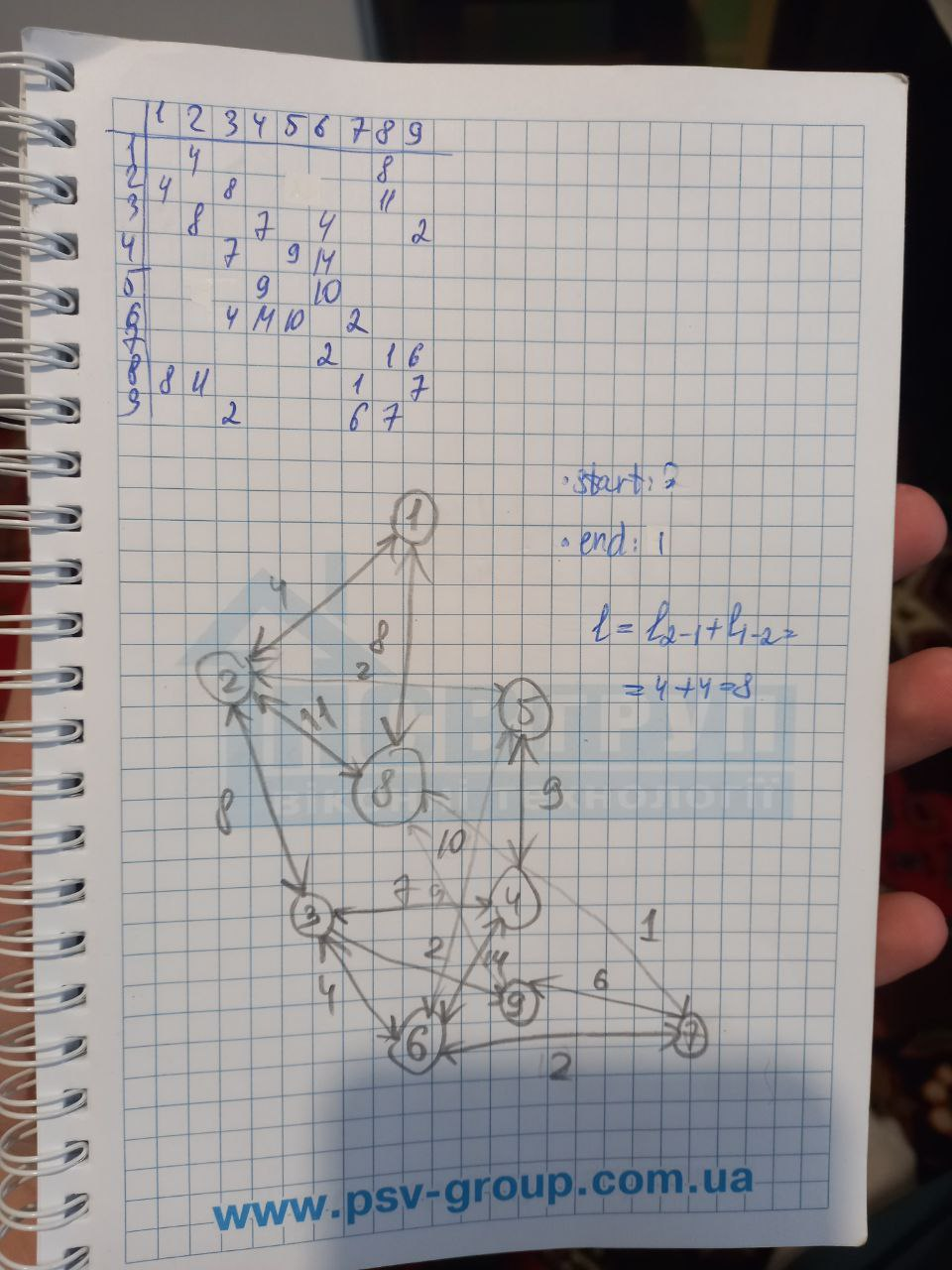
Від 1 до 5

8

3.3 Розв’язання задачі вручну

На рисунку 3.3 наведено розв’язання задачі 5 вручну.

Рисунок 3.3 – Розв’язання задачі 5 вручну



9

ВИСНОВОК

При виконанні даної лабораторної роботи ми познайомилися з графами, а також реалізували алгоритм Дейкстри.

10

КРИТЕРІЇ ОЦІНЮВАННЯ ДИВИСЬ СИЛАБУС



11