**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизованих систем обробки інформації**

**і управління**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

**«Прикладні задачі теорії графів ч.1»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01 Князєв Ілля Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc65946294)

[2 ЗаВдання 4](#_Toc65946295)

[3 Виконання 5](#_Toc65946296)

[3.1 Псевдокод алгоритму 5](#_Toc65946297)

[3.2 Програмна реалізація алгоритму 5](#_Toc65946298)

[3.2.1 Вихідний код 5](#_Toc65946299)

[3.2.2 Приклад роботи 5](#_Toc65946300)

[3.3 Розв’язання задачі вручну 6](#_Toc65946301)

[Висновок 7](#_Toc65946302)

# Мета лабораторної роботи

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

# 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | Побудова Гамільтонового циклу | Пошук із поверненнями | Неорієнтований | Матриця суміжності |

# Виконання

## Псевдокод алгоритму

input\_matrix()

**input** size

matrix = []

**for** count = 0 **to** size **do**

**input** row

matrix.append(row)

**endfor**

**return** matrix

hamilton(graph, current, seen = None, path = None)

**if** path == None **then**

seen = set{}

path = [current,]

**output** current

**else**

path.append(current)

**output** path

**endif**

**if** len(path) == len(graph) **then**

**if** graph[path[0]][path[-1]] == 1 **then**

**output** path

**return** True

**else**

path.pop()

**output** path

**return** False

**endif**

**endif**

seen.add(current)

**for** next\_node **in** range(len(graph)) **then**

**if** graph[current][next\_node] == 1 and not (next\_node in seen) **then**

**if** hamilton(graph, next\_node, seen, path) **then**

**return** True

**endif**

**endif**

**endfor**

seen.remove(current)

path.pop()

**output** path

**return** False

## Програмна реалізація алгоритму

### Вихідний код

from termcolor import colored

def input\_matrix():

size = int(input('Matrix size: '))

matrix = []

print("input matrix:")

for \_ in range(size):

row = [int(i) for i in input().split()]

matrix.append(row)

print()

return matrix

def hamilton(graph, current, seen = None, path = None):

if path == None:

seen = set()

path = [current,]

print(current + 1)

else:

path.append(current)

print("-".join([str(i + 1) for i in path]))

if len(path) == len(graph):

if graph[path[0]][path[-1]] == 1:

print(colored("->".join([str(i + 1) for i in path]) + f"->{path[0] + 1}", "green"))

return True

else:

path.pop()

print("-".join([str(i + 1) for i in path]))

return False

seen.add(current)

for next\_node in range(len(graph)):

if graph[current][next\_node] == 1 and not (next\_node in seen):

if hamilton(graph, next\_node, seen, path):

return True

seen.remove(current)

path.pop()

print("-".join([str(i + 1) for i in path]))

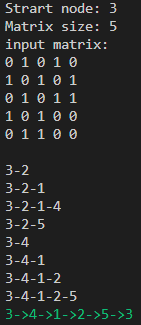
return False

start = int(input('Strart node: ')) - 1

graph = input\_matrix()

hamilton(graph, start)

### Приклад роботи



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

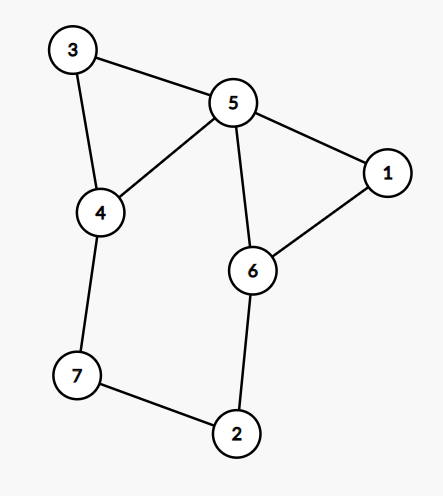
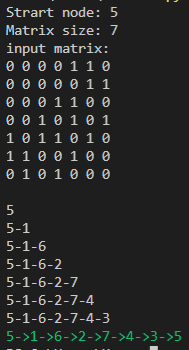
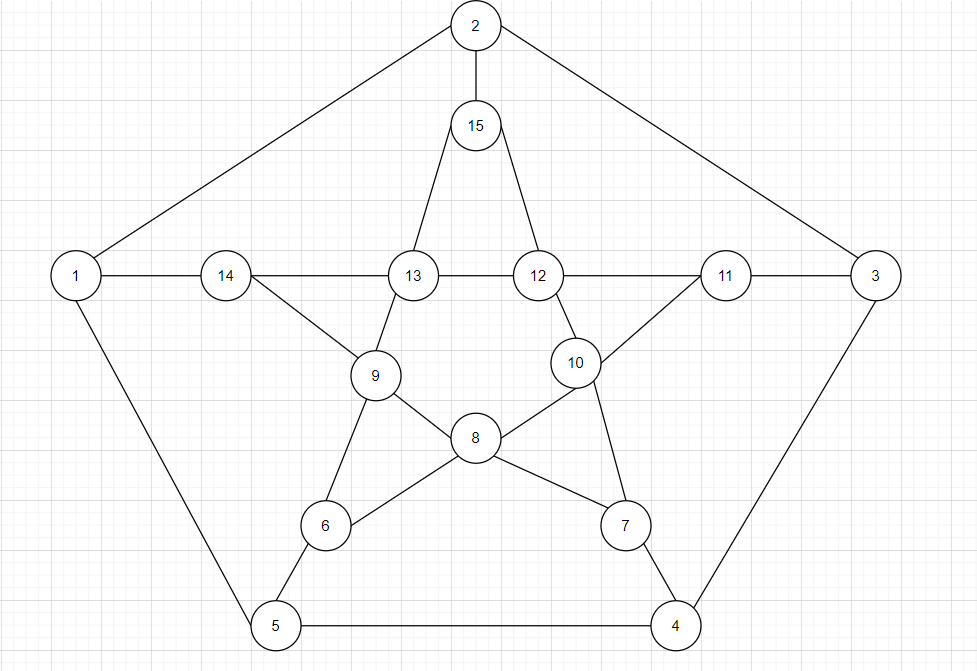
 

Рисунок 3.1



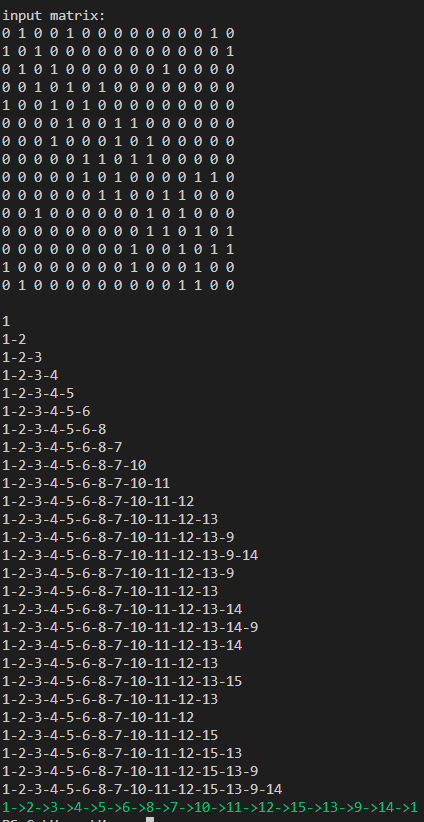


Рисунок 3.2

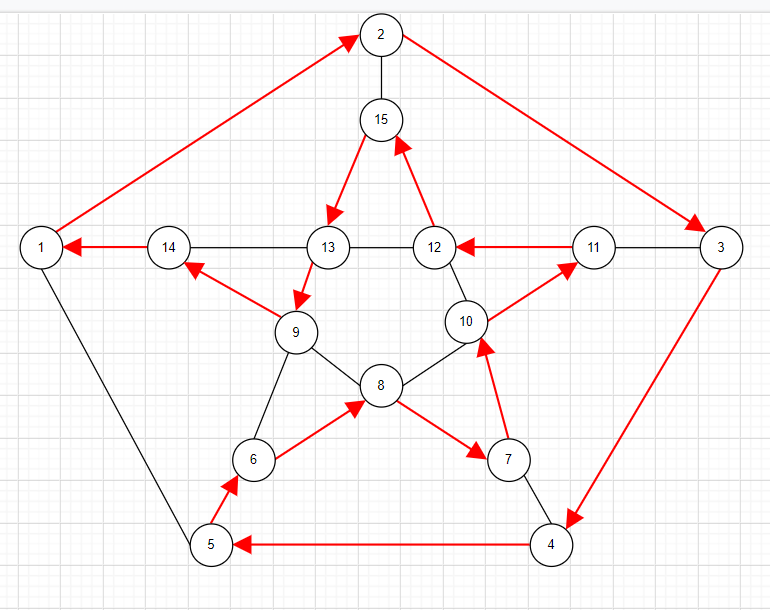
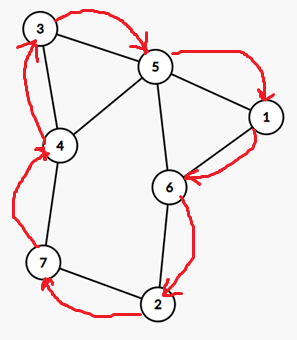


Рисунок 3.2

## Розв’язання задачі вручну

На рисунку 3.3 наведено розв’язання задачі 3.1 вручну.



5

5-1

5-1-6

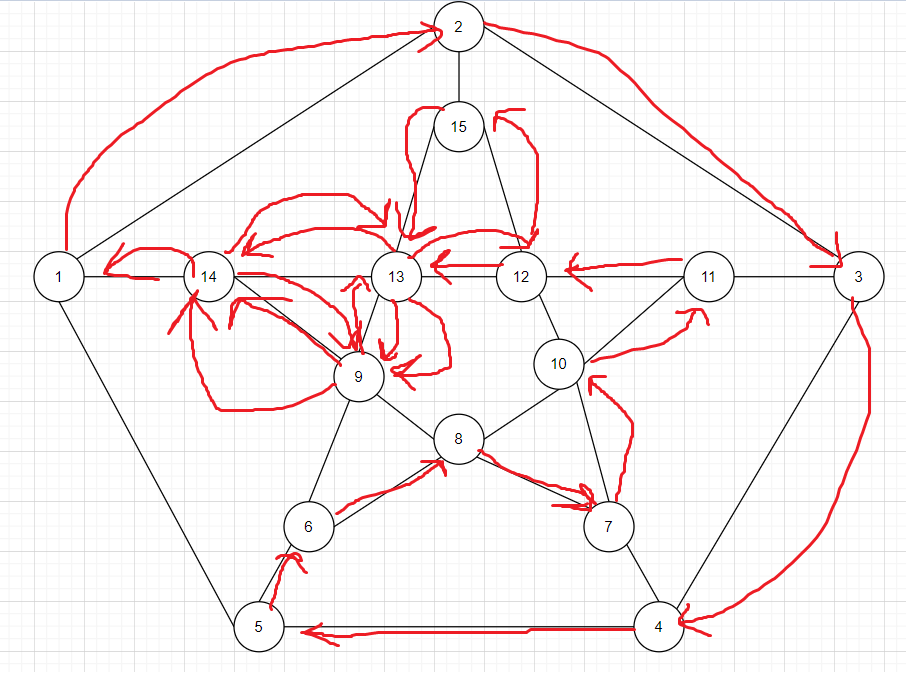
5-1-6-2

5-1-6-2-7

5-1-6-2-7-4

5-1-6-2-7-4-3

На рисунку 3.4 наведено розв’язання задачі 3.2 вручну.



1

1-2

1-2-3

1-2-3-4

1-2-3-4-5

1-2-3-4-5-6

1-2-3-4-5-6-8

1-2-3-4-5-6-8-7

1-2-3-4-5-6-8-7-10

1-2-3-4-5-6-8-7-10-11

1-2-3-4-5-6-8-7-10-11-12

1-2-3-4-5-6-8-7-10-11-12-13

1-2-3-4-5-6-8-7-10-11-12-13-9

1-2-3-4-5-6-8-7-10-11-12-13-9-14

1-2-3-4-5-6-8-7-10-11-12-13-9

1-2-3-4-5-6-8-7-10-11-12-13

1-2-3-4-5-6-8-7-10-11-12-13-14

1-2-3-4-5-6-8-7-10-11-12-13-14-9

1-2-3-4-5-6-8-7-10-11-12-13-14

1-2-3-4-5-6-8-7-10-11-12-13

1-2-3-4-5-6-8-7-10-11-12-13-15

1-2-3-4-5-6-8-7-10-11-12-13

1-2-3-4-5-6-8-7-10-11-12

1-2-3-4-5-6-8-7-10-11-12-15

1-2-3-4-5-6-8-7-10-11-12-15-13

1-2-3-4-5-6-8-7-10-11-12-15-13-9

1-2-3-4-5-6-8-7-10-11-12-15-13-9-14

Висновок

При виконанні даної лабораторної роботи було досліджено Гамільтонів граф. Було створено та протестовано алгоритм знаходження Гамільтонового циклу. Алгоритм було реалізовано мовою Python. Тести проводилися на графах з 7 та 15 вершинами. Ці ж тести було проведено вручну.