МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Український державний університет імені Михайла Драгоманова

Факультет математики, інформатики та фізики

Кафедра інформаційних технологій та програмування

**Звіт**

з лабораторної роботи №10

«Графи, динамічне програмування, жадібні алгоритми»

з дисципліни «Програмування»

Виконав:

студент ІІІ курсу групи 31І

Cавченко В. Г.

Перевірила:

викладач Устименко О.Б.

Оцінка\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_

Київ - 2024

**Зміст**

[Мета роботи 3](#_Toc167703227)

[1 Постановка задачі 4](#_Toc167703228)

[2 Основна частина 5](#_Toc167703229)

[2.1 Опис вхідних та вихідних даних 5](#_Toc167703230)

[2.2 Опис вхідних та вихідних даних 6](#_Toc167703231)

[2.2 Блок-схема 7](#_Toc167703232)

[Висновки 8](#_Toc167703233)

[Додатки 9](#_Toc167703234)

# Мета роботи

Мета лабораторної роботи – скласти програми мовою Python для опрацювання графів та вирішення задач за допомогою жадібних алгоритмів чи динамічного програмування.

# 1 Постановка задачі

Варіант №16

1. Написати програму для обходу графа за алгоритмом пошуку в ширину:

Зображення, що містить ряд, годинник, коло, дизайн

Автоматично згенерований опис

1. Припустімо, у вас є рюкзак і набір предметів із певними вагами та вартостями. Вам потрібно заповнити рюкзак таким чином, щоб максимізувати вартість обраних предметів, не перевищуючи місткість рюкзака.

# 2 Основна частина

## 2.1 Опис вхідних та вихідних даних

Вхідні дані: задаються безпосередньо в коді на основі малюнку графа

Основа програми це клас Graph, який містить в собі 3 функції, які описують виконання алгоритму пошуку в ширину

 def \_\_init\_\_(self):

       self.graph = {}

    def add\_edge(self, u, v):

        if u not in self.graph:

            self.graph[u] = []

        self.graph[u].append(v)

    def bfs(self, start):

        visited = set()

        queue = []

        visited.add(start)

        queue.append(start)

        while queue:

            vertex = queue.pop(0)

            print(vertex, end='')

            for neighbor in self.graph.get(vertex, []):

                if neighbor not in visited:

                    visited.add(neighbor)

                    queue.append(neighbor)

Вихідні дані:

print("Результат пошуку в ширину")

g.bfs(7)

## 2.2 Опис вхідних та вихідних даних

Вхідні дані: описуються в середині коду задля демонстрації роботи жадібного алгоритму

items = [

  {"weight": 2, "value": 3},

  {"weight": 3, "value": 4},

  {"weight": 4, "value": 5},

  {"weight": 5, "value": 6},

]

max\_weight = 10

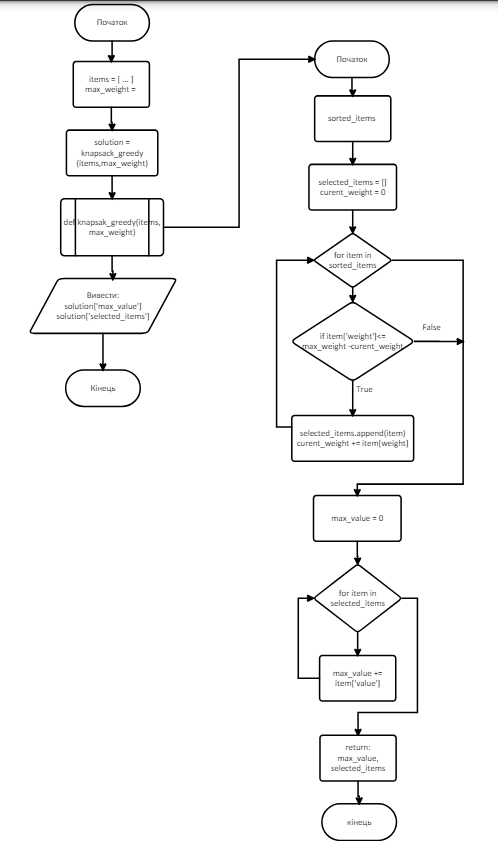
Основна частина коду це функція knapsack\_greedy в середині, якої описано алгоритм сортування та вибору речей відповідно до їх вартості, ваги та максимальної можливої ваги

Вихідні дані:

print(f"Максимальна сумарна вартість: {solution['max\_value']}")

print("Обрані предмети:", solution["selected\_items"])

## 2.2 Блок-схема



# Висновки

В ході лабораторної роботи було проведено ознайомлення з графами, динамічним програмуванням та жадібними алгоритмами за допомогою мови програмування Python.

В результаті виконання лабораторної роботи було розроблено дві програми:

* Алгоритм пошуку на графі в ширину для заданого графа
* Реалізація жадібного алгоритму для задачі про рюкзак

# Додатки

Лістинги програм:

Завдання 1

class Graph:

    def \_\_init\_\_(self):

       self.graph = {}

    def add\_edge(self, u, v):

        if u not in self.graph:

            self.graph[u] = []

        self.graph[u].append(v)

    def bfs(self, start):

        visited = set()

        queue = []

        visited.add(start)

        queue.append(start)

        while queue:

            vertex = queue.pop(0)

            print(vertex, end='')

            for neighbor in self.graph.get(vertex, []):

                if neighbor not in visited:

                    visited.add(neighbor)

                    queue.append(neighbor)

g = Graph()

g.add\_edge(1, 3)

g.add\_edge(2, 4)

g.add\_edge(2, 5)

g.add\_edge(3, 1)

g.add\_edge(3, 7)

g.add\_edge(3, 6)

g.add\_edge(3, 4)

g.add\_edge(4, 3)

g.add\_edge(4, 2)

g.add\_edge(4, 5)

g.add\_edge(4, 8)

g.add\_edge(5, 4)

g.add\_edge(5, 2)

g.add\_edge(6, 3)

g.add\_edge(6, 7)

g.add\_edge(6, 4)

g.add\_edge(7, 3)

g.add\_edge(7, 6)

g.add\_edge(7, 9)

g.add\_edge(7, 8)

g.add\_edge(8, 10)

g.add\_edge(8, 4)

g.add\_edge(8, 7)

g.add\_edge(9, 7)

g.add\_edge(10, 8)

print("Результат пошуку в ширину")

g.bfs(7)

Завдання 2

def knapsack\_greedy(items, max\_weight):

  sorted\_items = sorted(items, key=lambda item: item["value"] / item["weight"], reverse=True)

  selected\_items = []

  current\_weight = 0

  for item in sorted\_items:

    if item["weight"] <= max\_weight - current\_weight:

      selected\_items.append(item)

      current\_weight += item["weight"]

  max\_value = 0

  for item in selected\_items:

    max\_value += item["value"]

  return {

    "max\_value": max\_value,

    "selected\_items": selected\_items

  }

items = [

  {"weight": 2, "value": 3},

  {"weight": 3, "value": 4},

  {"weight": 4, "value": 5},

  {"weight": 5, "value": 6},

]

max\_weight = 10

solution = knapsack\_greedy(items, max\_weight)

print(f"Максимальна сумарна вартість: {solution['max\_value']}")

print("Обрані предмети:", solution["selected\_items"])