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 і жадібні алгоритми»

з дисципліни «Програмування»

Виконав:

студент ІІІ курсу групи 31І

Шарабар Ярослав Анатолійович

Перевірила:

викладач Устименко О.Б.

Оцінка\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_

Київ - 2024

**Зміст**

[Мета роботи 3](#_Toc150170352)

[1 Постановка задачі 4](#_Toc150170353)

[2 Основна частина 5](#_Toc150170354)

[2.1 Опис вхідних та вихідних даних 5](#_Toc150170355)

[2.1 Блок-схема 6](#_Toc150170356)

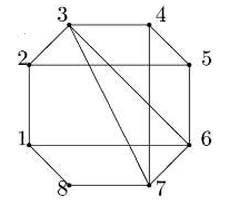
[Висновки 7](#_Toc150170357)

**Мета роботи**

Мета лабораторної роботи ознайомитись з основними принципами та підходами до вирішення задач за допомогою динамічного програмування та жадібних алгоритмів у контексті графів. Розглянути та порівняти різні підходи до вирішення задач на графах з використанням цих алгоритмів, зосереджуючись на їх застосуванні до таких задач, як знаходження найкоротших шляхів, мінімальних остовних дерев.

**Постановка задачі**

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



""

from collections import deque

def bfs(graph, start\_vertex):

visited = set()

queue = deque([start\_vertex])

visited.add(start\_vertex)

while queue:

vertex = queue.popleft()

print(vertex, end=" ")

for neighbor in graph[vertex]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

# Представлення графа списком суміжності

graph = {

1: [2, 6, 8],

2: [1, 3, 5],

3: [2, 4, 6, 7],

4: [3, 5, 7],

5: [2, 4, 6],

6: [1, 3, 5, 7],

7: [3, 4, 6, 8],

8: [1, 7]

}

#Запустіть BFS з вершини яку обирете нижче

bfs(graph, 5)

**2. Дано рядок без пробілів і словник слів. Потрібно визначити, чи можна розбити цей рядок на послідовність слів зі словника.**

def word\_break(s, word\_dict):

word\_set = set(word\_dict)

dp = [False] \* (len(s) + 1)

dp[0] = True

for i in range(1, len(s) + 1):

for j in range(i):

if dp[j] and s[j:i] in word\_set:

dp[i] = True

break

return dp[len(s)]

s = "blackdoor"

word\_dict = ["black", "door"]

print(word\_break(s, word\_dict))

**Основна частина**

**2.1 Опис вхідних та вихідних даних**

**Задача №1**

**Вхідні дані:**

1. graph: словник, який представляє граф у вигляді списку суміжності. Ключі словника - вершини графа, а значення - списки суміжних вершин.

**Вихідні дані:**

1. Функція виводить вершини графа в порядку їх обходу за допомогою BFS.

**Змінні та функції:**

1. visited: множина, що зберігає відвідані вершини графа.
2. queue: черга, яка використовується для обходу вершин графа в порядку BFS.
3. start\_vertex: початкова вершина, з якої розпочинається обхід графа.
4. bfs(graph, start\_vertex): функція, яка виконує обхід графа в порядку BFS, виводить вершини графа в процесі обходу.

**Задача №2**

**Вхідні дані:**

* **s**: рядок, який потрібно розбити на слова.
* **word\_dict**: список слів, який представляє словник слів, за допомогою якого потрібно розбити рядок **s**.

**Вихідні дані:**

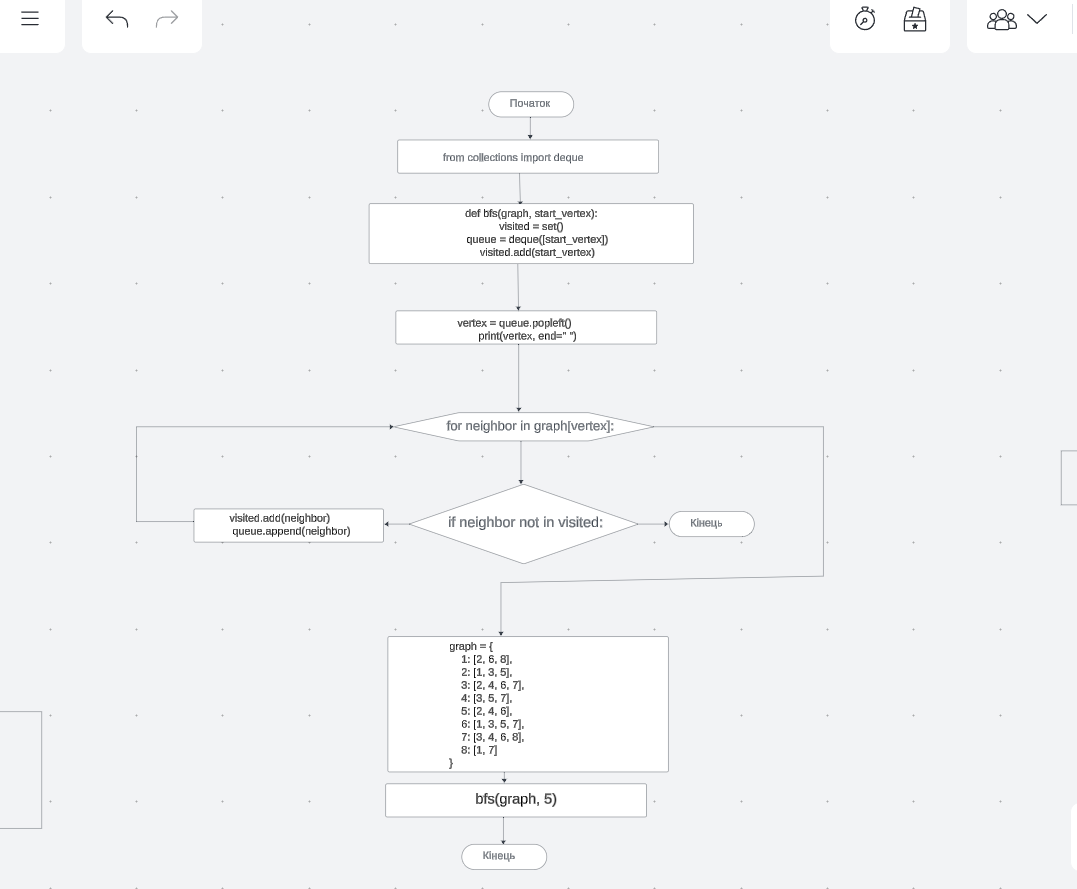
* Логічне значення: **True**, якщо рядок **s** можна розбити на послідовність слів зі словника **word\_dict**, або **False**, якщо це неможливо.

**Змінні та функції:**

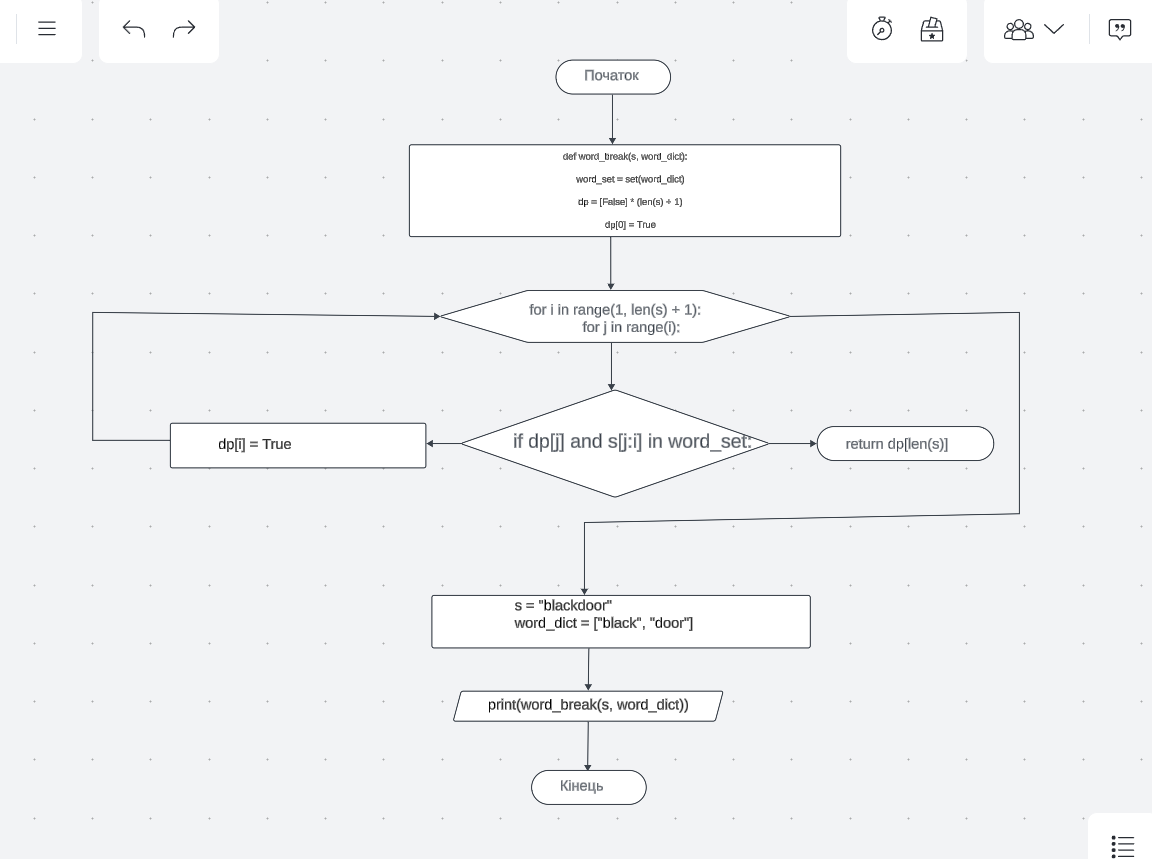
* **word\_set**: множина, яка містить слова зі словника **word\_dict**. Це зроблено для оптимізації, оскільки пошук у множині є більш ефективним за пошук у списку.
* **dp**: список, що представляє динамічний масив для збереження проміжних результатів. **dp[i]** дорівнює **True**, якщо підрядок **s[0:i]** можна розбити на слова зі словника **word\_dict**.
* **word\_break(s, word\_dict)**: функція, яка приймає рядок **s** та словник слів **word\_dict** і повертає логічне значення, що показує, чи можна рядок розбити на послідовність слів зі словника.
* **print(word\_break(s, word\_dict))**: друк результату виклику функції **word\_break()** з вхідними даними **s** і **word\_dict**. Виводить **True**, якщо рядок можна розбити на слова зі словника, або **False** в іншому випадку.

**2.2 Блок-схема**

**Задача № 1**



**Задача №2**



**Висновки**

В цій лабораторній роботі ми досліджували теми жадібних алгоритмів, графів та динамічного програмування. Вони є ключовими концепціями в області алгоритмів та ігнорувати їх неможливо, оскільки вони є основою для розв'язання багатьох складних проблем. Жадібні алгоритми вирішують проблеми, обираючи на кожному кроці найкращий доступний варіант, сподіваючись, що такий локальний вибір приведе до оптимального розв'язку задачі в цілому. Вони часто використовуються для оптимізаційних задач та проблем мінімізації або максимізації. Однак варто пам'ятати, що жадібні алгоритми не завжди дають оптимальний результат, і їх слід використовувати тільки тоді, коли можна довести, що вони приведуть до оптимального розв'язку. Графи є потужним інструментом для моделювання різноманітних ситуацій, де об'єкти можуть бути пов'язані між собою. Вони застосовуються в багатьох областях, таких як транспортні мережі, соціальні мережі, комп'ютерні мережі тощо. Алгоритми на графах дозволяють вирішувати різноманітні задачі, такі як пошук шляхів, знаходження компонентів зв'язності, пошук циклів тощо. Динамічне програмування - це потужний метод розв'язання проблем, які можуть бути розбиті на підзадачі, і для яких можна знайти оптимальний розв'язок шляхом комбінування оптимальних розв'язків підзадач. Цей метод часто використовується для вирішення задач оптимізації, таких як пошук найкоротшого шляху, рюкзакова задача, розбиття послідовності тощо.

**Додатки(лістинги програми)** 