Лабораторна робота №4 (застосування алгоритму пошуку в ширину)

Для всіх задач слід написати тести з використанням бібліотеки `unittest`. Ваш проект має бути розділено на окремі папки для коду додатку та тестів (`src` та `test` відповідно).

При написанні коду дотримуйтесь стандарту PEP 8, який визначає правила форматування Python-коду, такі як відступи, довжина рядків, іменування змінних тощо. Для полегшення читабельності коду слід відформатувати ваш код з допомогою `Black`

Рівень 2

Варіант 1 - Лабіринт

Дано лабіринт у формі двійкової прямокутної матриці, знайдіть найкоротший шлях у лабіринті від заданої точки до вказаного пункту призначення.

Шлях можна побудувати лише з використанням комірок, які містіть 1. В будь-який момент ми можемо рухатися лише на один крок в одному з чотирьох напрямків:

```
Go Top: (x, y) ──> (x - 1, y)
Go Left: (x, y) ──> (x, y - 1)
Go Down: (x, y) ──> (x + 1, y)
Go Right: (x, y) ──> (x, y + 1)
```

Наприклад, розглянемо наступну двійкову матрицю. Якщо початкова точка має координати (0, 0), а пункт призначення = (7, 5), тоді найкоротший шлях від джерела до пункту призначення має довжину 12

```
[ 1  1  1  1  1  0  0  1  1  1 ]
[ 0  1  1  1  1  1  0  1  0  1 ]
[ 0  0  1  0  1  1  1  0  0  1 ]
[ 1  0  1  1  1  0  1  1  0  1 ]
[ 0  0  0  1  0  0  0  1  0  1 ]
[ 1  0  1  1  1  0  0  1  1  0 ]
[ 0  0  0  0  1  0  0  1  0  1 ]
[ 0  1  1  1  1  1  1  1  0  0 ]
[ 1  1  1  1  1  0  0  1  1  1 ]
[ 0  0  1  0  0  1  1  0  0  1 ]
```

Вхідні дані містяться у файлі input.txt у форматі:
0, 0 #початкова точка
7, 5 #точка призначення

10,10 # кількість рядків та стовпчиків у матриці

```
[ 1  1  1  1  1  0  0  1  1  1 ]
[ 0  1  1  1  1  1  0  1  0  1 ]
[ 0  0  1  0  1  1  1  0  0  1 ]
[ 1  0  1  1  1  0  1  1  0  1 ]
[ 0  0  0  1  0  0  0  1  0  1 ]
[ 1  0  1  1  1  0  0  1  1  0 ]
[ 0  0  0  0  1  0  0  1  0  1 ]
[ 0  1  1  1  1  1  1  1  0  0 ]
[ 1  1  1  1  1  0  0  1  1  1 ]
[ 0  0  1  0  0  1  1  0  0  1 ]
```

Результуючий файл має містити значення найкоротшого шляху від початкової точки до точки призначення

Варіант 2. Знайдіть кореневу вершину графа

Коренева вершина орієнтованого графа — це вершина u з орієнтованим шляхом від u до v для кожної пари вершин (u, v) у графі. Іншими словами, всі інші вершини в графі можна досягнути з кореневої вершини (Рис. 2)

Граф може мати кілька кореневих вершин. Наприклад, кожна вершина графу може бути мати зв’язки з іншими (Рис.1). У таких випадках ваш код має повернути будь-яку із них. Якщо граф не має кореневих вершин, код має повернути -1.

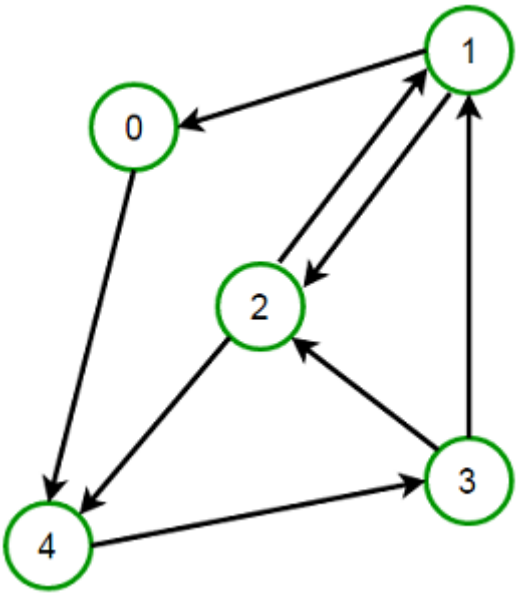


Рис.1 Граф який містить компоненту сильної звязаності

Коренева вершина дорівнює 4, оскільки вона має шлях до кожної іншої вершини в наступному графі:

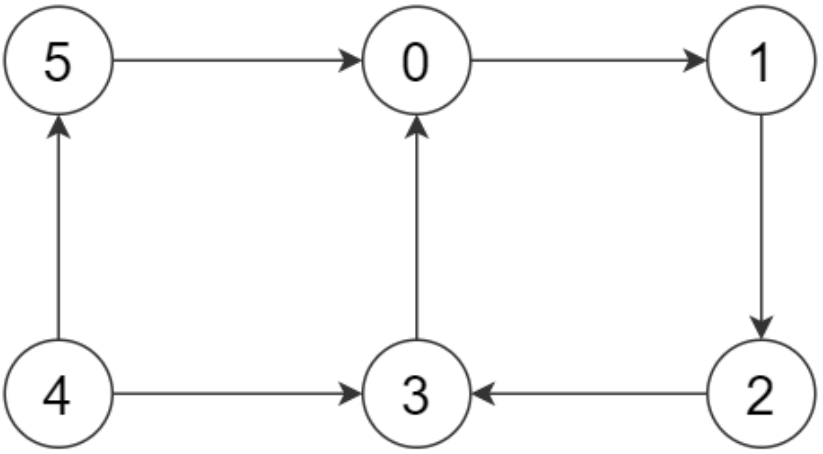


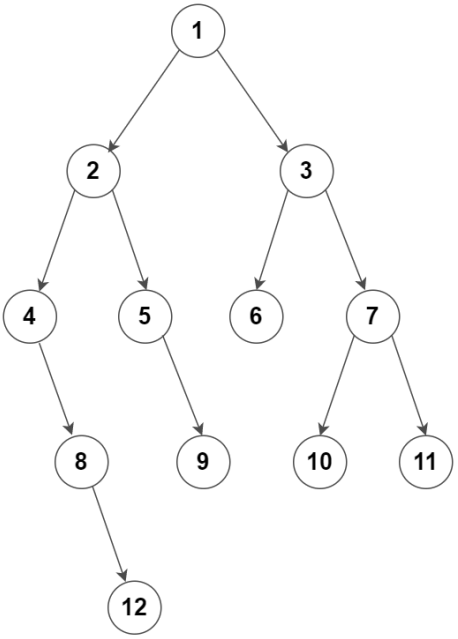
Рис. 2. Граф з кореневою вершиною 4

Для представлення графу слід використати список суміжності, який зчитується з файлу input.txt
Результат слід вивести у файл output.txt

Варіант 3 Пошук мінімальної глибини бінарного дерева

Дано двійкове дерево, знайти його мінімальну глибину. Мінімальна глибина — це загальна кількість вузлів уздовж найкоротшого шляху від кореневого вузла до найближчого кінцевого вузла.

Наприклад, мінімальна глибина наступного бінарного дерева дорівнює 3. Найкоротший шлях — 1 —> 3 —> 6.



Для представлення графу слід використати список суміжності,дані зчитуються з файлу input.txt

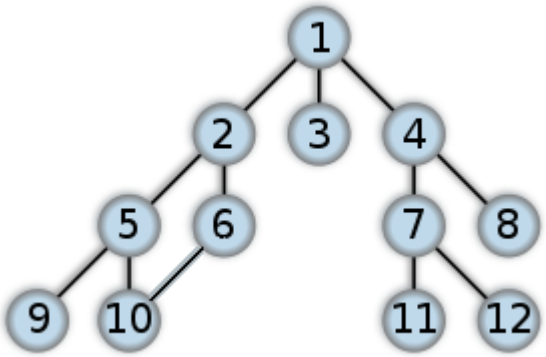
input.txt містить:
1 #корінь дереа
1,2 # список ребер, де порядок вершин визначає напрямок. В даному випадку ребро направлене від 1 до 2
1,3
2,4
…

Результат (мінімальна глибина дерева) слід вивести у файл output.txt

Варіант 4 Перевірте, чи містить неорієнтований граф цикл

Дано зв’язний неорієнтований граф, перевірте, чи містить він цикл чи ні.

Наприклад, наступний графік містить цикл 2–5–10–6–2:



Для представлення графу слід використати список суміжності, який зчитується з файлу input.txt
Результат (True - якщо цикл присутній, False - якщо відсутній) слід вивести у файл output.txt

Рівень 3

Варіант 1. заливка поля

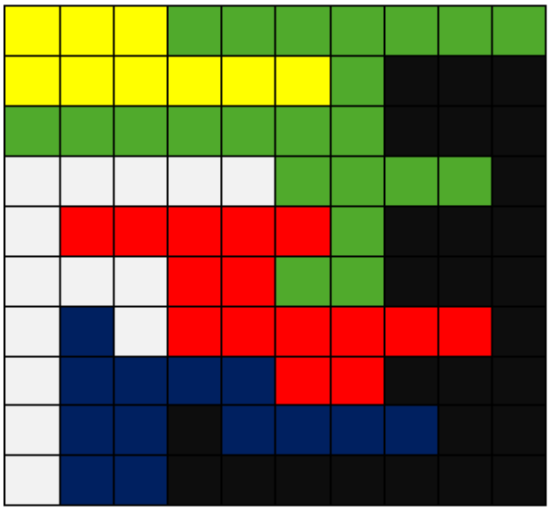
Flood fill (також відомий як початкове заповнення) — це алгоритм, який визначає область, в яку входить певний вузол в багатовимірному масиві.

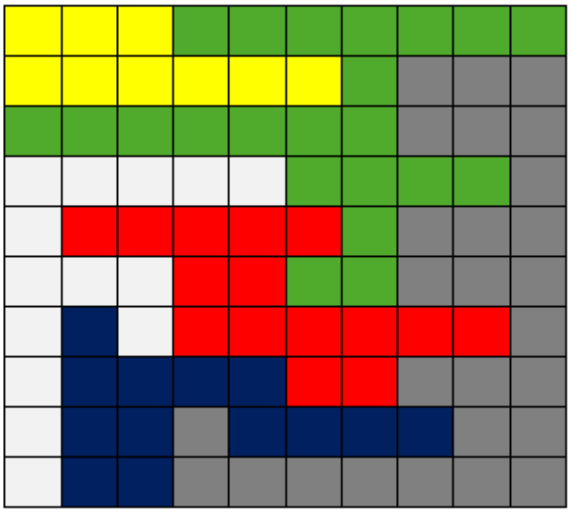
Він використовується в інструменті заповнення «ковша» програми малювання, щоб заповнити з'єднані, однаково забарвлені області іншим кольором, а також у таких іграх, як Go та Minesweeper.
Застосування для заповнення кольором певної обмеженої області також називається заливкою межі.

Алгоритм заливки приймає три параметри: початковий вузол, цільовий колір і колір заміни.

Розглянемо наступну матрицю (див. рисунок)

Нехай ліворуч оригінальне значення кольоріів. Якщо початковий вузол (3, 9), цільовий колір — «ЧОРНИЙ», а колір заміни — «СІРИЙ», тоді алгоритм шукає всі вузли в матриці, які з'єднані з початковим вузлом таким самим кольором та змінює їх на колір заміни. Результат роботи алгоритму наведено праворуч





Формат даних у вхідному файлі input.txt
перший рядок: висота та ширина поля, яке містить кольорові елементи (розділені комою)
другий рядок: координати початкової точки, з якої слід починати зміну кольору
третій рядок: колір заміни

Приклад даних для рисунку зліва:

10,10
3,9
'G'

```
['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G'],
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'X', 'X', 'X'],
['G', 'G', 'G', 'G', 'G', 'G', 'G', 'X', 'X', 'X'],
['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'X'],
```

```
['W', 'R', 'R', 'R', 'R', 'R', 'G', 'X', 'X', 'X'],
['W', 'W', 'W', 'R', 'R', 'G', 'G', 'X', 'X', 'X'],
['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'X'],
['W', 'B', 'B', 'B', 'B', 'R', 'R', 'X', 'X', 'X'],
['W', 'B', 'B', 'X', 'B', 'B', 'B', 'B', 'X', 'X'],
['W', 'B', 'B', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
```

Результат:
файл output.txt містить значення кольорів комірок поля після виконання заміни (для рисунку справа)

```
['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'C', 'C', 'C']
['G', 'G', 'G', 'G', 'G', 'G', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'C']
['W', 'R', 'R', 'R', 'R', 'R', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'R', 'R', 'G', 'G', 'C', 'C', 'C']
['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'C']
['W', 'B', 'B', 'B', 'B', 'R', 'R', 'C', 'C', 'C']
['W', 'B', 'B', 'C', 'B', 'B', 'B', 'B', 'C', 'C']
['W', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'C', 'C']
```

Варіант 2. Задача про шахового коня

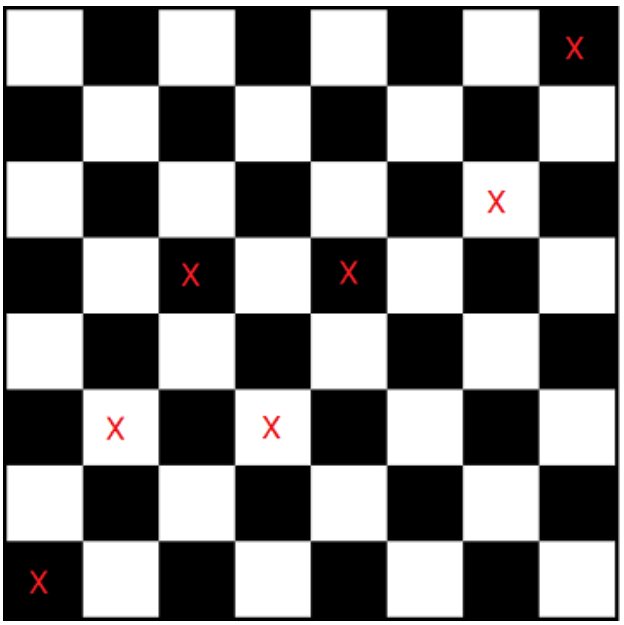
Дано шахівницю, знайдіть найкоротшу відстань (мінімальна кількість ходів), яку пройшов кінь, щоб досягти заданого пункту призначення з заданого джерела.

Наприклад,

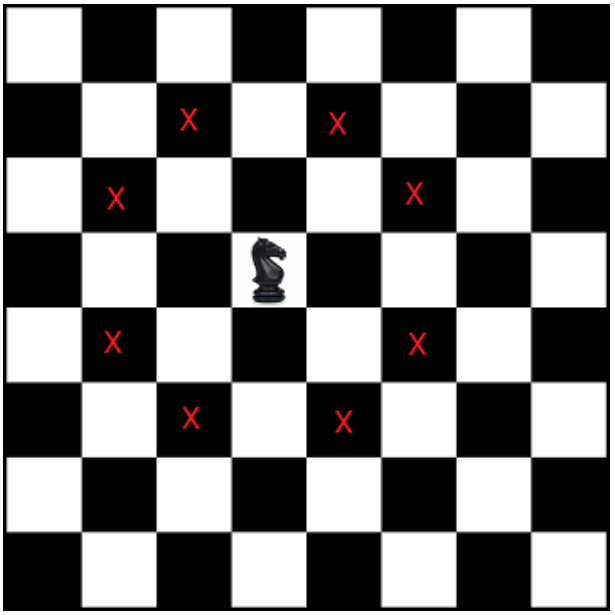
Вхідні дані:

N = 8 (дошка 8 × 8)
Джерело = (7, 0)
Пункт призначення = (0, 7)

Результат: мінімальна необхідна кількість кроків – 6 (див рисунок, червоними хрестиками позначено



Кінь може ходити у восьми можливих напрямках від даної клітини, як показано на наступному малюнку:



Ми можемо знайти всі можливі місця, куди лицар може переміститися з даного місця, використовуючи масив, який зберігає відносну позицію руху лицаря з будь-якого місця. Наприклад, якщо поточне розташування `(x, y)`, ми можемо перейти до `(x + row[k], y + col[k])` для `0 <= k <= 7` за допомогою такого масиву:

```
row[] = [ 2, 2, -2, -2, 1, 1, -1, -1 ]
col[] = [ -1, 1, 1, -1, 2, -2, 2, -2 ]
```

З позиції (x,y) кінь може рухатись на клітинки з координатами:

```
(x + 2, y - 1)
(x + 2, y + 1)
(x - 2, y + 1)
(x - 2, y - 1)
(x + 1, y + 2)
(x + 1, y - 2)
(x - 1, y + 2)
(x - 1, y - 2)
```

Вхідні дані містяться у файлі input.txt:

```
8       # розмір поля
7, 0    # стартова точка
0, 7    # точка призначення
```

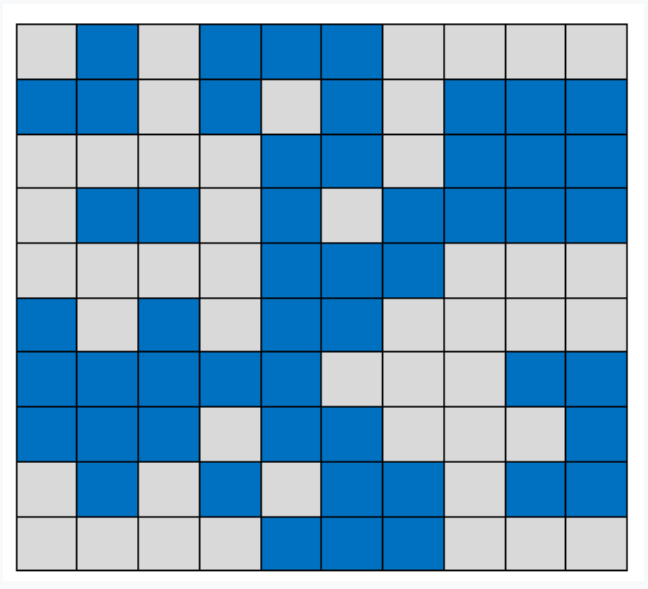
Результуючий файл має містити кількість ходів

Результуючий файл має містити значення найкоротшого шляху від початкової точки до точки призначення

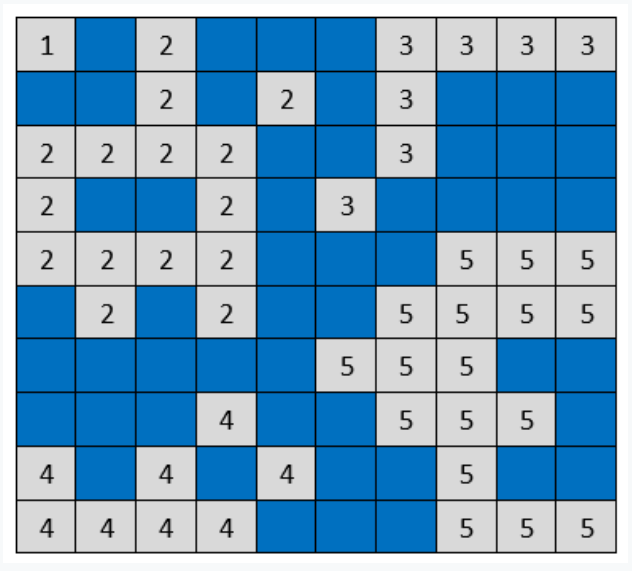
Варіант 3 Обчислення кількості островів

Нехай дано двійкову матрицю, де 0 означає воду, а 1 — сушу, а з’єднані числа 1 утворюють острів, підрахуйте загальну кількість островів.

Наприклад, розглянемо таке зображення, де синім позначено воду (0), а сірим - сушу (1):



Загалом у наведеній вище матриці присутні п’ять островів. На зображенні нижче вони позначені цифрами 1–5.



Варіант 4. Знайдіть найкоротший безпечний маршрут у полі з датчиками

Нехай у вас задане прямокутне поле, на якому встановлені датчики в певних місцях. Перетніть його найкоротшим безпечним шляхом, не активуючи датчики.

Прямокутне поле має форму матриці M × N, і нам потрібно знайти найкоротший шлях від будь-якої клітинки в першому стовпці до будь-якої клітинки в останньому стовпці матриці. Датчики позначаються в матриці значенням 0, і всі її вісім суміжних осередків також можуть активувати датчики. Шлях можна побудувати лише з комірок зі значенням 1, і в будь-який момент ми можемо рухатися лише на один крок в одному з чотирьох напрямків. Допустимі ходи:

Вгору: (x, y) -> (x – 1, y)
Ліворуч: (x, y) -> (x, y – 1)
Перейти вниз: (x, y) -> (x + 1, y)
Праворуч: (x, y) -> (x, y + 1)
Наприклад, розглянемо таку матрицю:

0	1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Найкоротший безпечний шлях має довжину 11. Безпечний маршрут позначений зеленим кольором:

0	0	1	0	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	0
1	1	1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	1	1	1
0	0	0	1	0	0	0	1	1	1
0	0	0	1	1	1	1	1	0	0
0	0	0	1	0	0	0	1	0	0
1	1	1	1	0	0	0	1	0	0
1	1	1	1	0	0	0	1	1	1

Для представлення графу слід використати матрицю, який зчитується з файлу input.txt
Алгоритм має вивести довжину найкоротшого шляху, або -1 якщо такого не існує
Результат слід вивести у файл output.txt