МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ

Р. М. Бабаков

ТЕХНОЛОГІЇ ЗБЕРІГАННЯ, ОБРОБКИ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ

Методичні вказівки до виконання лабораторних робіт
для здобувачів вищої освіти ОС «Магістр» денної форми здобуття освіти
спеціальності F3 Комп’ютерні науки

Вінниця
2025

УДК 004.6:004.9(076.5)
Б 12

Рекомендовано до друку вченою радою
факультету інформаційних і прикладних технологій
Донецького національного університету імені Василя Стуса
(протокол № 4 від 24 жовтня 2025 р.)

Автор:
Бабаков Р. М., д-р техн. наук, доцент, професор кафедри інформаційних технологій Донецького національного університету імені Василя Стуса.

Рецензенти:
Січко Т. В., канд. техн. наук, доцент кафедри інформаційних технологій Донецького національного університету імені Василя Стуса;
Потапова Н. А., канд. екон. наук, доцент кафедри інформаційних технологій Донецького національного університету імені Василя Стуса.

Р. М. Бабаков
Б 12 Технології зберігання, обробки та візуалізації даних: методичні рекомендації до виконання лабораторних робіт для здобувачів вищої освіти ОС «Магістр» денної форми здобуття освіти спеціальності F3 Комп'ютерні науки. Вінниця: ДонНУ імені Василя Стуса, 2025. 41 с.

Методичні рекомендації є навчально-методичним документом, який містить рекомендації для отримання практичних навичок розв'язання прикладних задач під час виконання практикуму з дисципліни «Технології зберігання, обробки та візуалізації даних».

Для здобувачів ступеня освіти «Магістр» спеціальності F3 Комп'ютерні науки факультету інформаційних і прикладних технологій ДонНУ імені Василя Стуса.

УДК 004.6:004.9(076.5)

© Бабаков Р. М., 2025
© ДонНУ імені Василя Стуса, 2025

ЗМІСТ

ВСТУП 4
Лабораторна робота № 1 Обробка тривимірних масивів у мові Python.....5
Лабораторна робота № 2 Векторні операції в NumPy.....16
Лабораторна робота № 3 Робота з бібліотекою Matplotlib 21
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ 40

ВСТУП

Дослідницька діяльність людства нерозривно пов'язана з накопиченням і обробкою різноманітної інформації. У процесі накопичення даних збільшується необхідність застосування комп'ютерних методів і технологій аналізу даних, що дають змогу досліднику отримати додаткові знання з предметної галузі, в якій він працює.

Одним з основних інструментів аналізу даних сьогодні є високорівнева мова програмування Python. Для обробки, зберігання і візуалізації даних у мові Python найбільш широко використовуються такі бібліотеки:

1. NumPy – основний пакет для виконання наукових розрахунків. Надає засоби для ефективної роботи з багатовимірними масивами, операції лінійної алгебри, генерацію псевдовипадкових чисел тощо.

2. Matplotlib – призначена для створення графіків, діаграм та інших способів візуалізації двовимірних та тривимірних даних.

3. Pandas – базується на бібліотеках NumPy та Matplotlib і надає функції для покращення роботи зі структурованими даними.

4. SciPy – збірка пакетів для вирішення стандартних обчислювальних задач, як-то: чисельне інтегрування, розв'язання диференційних рівнянь, робота з розрідженими матрицями та інші.

У межах навчальної дисципліни «Технології зберігання, обробки та візуалізації даних» розглядаються бібліотеки NumPy та Matplotlib. Практична частина курсу містить три лабораторні роботи, в основу яких покладено задачі обробки тривимірних масивів даних. Перша лабораторна робота присвячена розв'язанню задачі обробки тривимірних даних стандартними засобами мови Python без використання можливостей бібліотеки NumPy. У другій лабораторній роботі та сама задача вирішується засобами NumPy, причому здійснюється порівняльна оцінка швидкості розв'язання задачі у першій і другій лабораторних роботах. Третя лабораторна робота присвячена бібліотеці Matplotlib і полягає у візуалізації результатів, отриманих у другій лабораторній роботі. Також у навчальному курсі передбачене виконання індивідуального творчого завдання, в якому здобувачі освіти повторюють виконання третьої лабораторної роботи за допомогою альтернативних бібліотек візуалізації даних (на власний вибір).

Методичні рекомендації містять завдання, пояснення, приклади виконання лабораторних робіт та індивідуального творчого завдання з дисципліни «Технології зберігання, обробки та візуалізації даних», забезпечують отримання програмних результатів навчання, загальних та спеціальних компетентностей для відповідної освітньої компоненти.

Лабораторна робота № 1
Обробка тривимірних масивів у мові Python

Метою лабораторної роботи є дослідження швидкодії мови Python під час розв'язання задач обробки тривимірних масивів даних великого розміру.

Завдання до лабораторної роботи

Інститут геологічних досліджень проводить пошук корисних копалин. Територія пошуку (об'єм ґрунту) має площу $m * n$ сотень метрів, глибина пошуку – k сотень метрів. Попередні геологічні дослідження дали змогу розробити докладну карту вмісту усього обсягу ґрунту з точністю до сотні метрів та представити її у вигляді тривимірного масиву цілих чисел розміром $m * n * k$ елементів. Перші два виміри відповідають площині, третій вимір – глибині (0 – поверхня, k – найбільша глибина).

Кожен елемент масиву може мати таке значення:

0 – корисні копалини відсутні, гірська порода м'яка;

1 – корисні копалини відсутні, гірська порода тверда;

2 – корисні копалини відсутні, гірська порода надтверда;

3 – вода;

4 – нафта;

5 – кам'яне вугілля;

6 – залізна руда;

7 – золото.

Видобуток корисних копалин дає прибуток. Прибуток від розробки одного елементу масиву, що містить нафту або кам'яне вугілля, – однаковий і дорівнює трьом умовним одиницям. Прибуток від розробки залізної руди у два рази більший, ніж від нафти. Прибуток від розробки золота удесятеро більший від нафти. Прибуток від видобутку води та гірських порід відсутній.

Складність видобутку корисних копалин визначається кількістю елементів масиву, які треба пройти вертикально вниз (вздовж третього виміру масиву), щоб дістатись елементу, що містить потрібний ресурс. Заглиблення на один елемент, що містить м'які гірські породи, коштує одну умовну одиницю трудовитрат; будь-які корисні копалини та воду – дві одиниці трудовитрат; тверді гірські породи – три одиниці трудовитрат; надтверді гірські породи – п'ять одиниць трудовитрат.

Необхідно провести аналіз території пошуку корисних копалин згідно з індивідуальним варіантом завдання з таблиці 1. Номер рядка в таблиці відповідає номеру студента в журналі підгрупи. Аналіз території пошуку корисних копалин передбачає виконання таких дій:

1. Розробити програму на мові Python, що вирішує задачу згідно з індивідуальним варіантом завдання. Програма повинна коректно обробляти масиви будь-

якого розміру. Якщо масив має розмір 1*1*1, а алгоритм передбачає мінімально достатній розмір 3*3*3, програма повинна повідомляти про недостатність розміру масиву.

2. Розробити 2–3 тестові приклади для масивів маленького розміру, які демонструють правильність роботи алгоритму. Для цих прикладів вміст масивів необхідно задавати штучно, заповнюючи їх константами за допомогою циклів або вручну за фіксованими значеннями координат.

3. Додати в програму можливість створення масивів будь-якого розміру, заповнених псевдовипадковими значеннями в діапазоні від 0 до 7. Заповнення може відбуватись не просто за допомогою random.randint(0, 7), а бути трохи хитрішим. Наприклад, значення 0 будуть зустрічатися значно частіше, ніж значення 7, або навпаки. Або спочатку масив заповнюється значеннями random.randint(0, 6), а потім в окремому циклі додається декілька значень, рівних 7. Зрозуміло, що за однакових початкових значень генератора псевдовипадкових чисел масив повинен заповнюватись абсолютно однаково (щоб була можливість багаторазово отримати і перевірити той самий результат).

4. У тексті програми перед початком та в кінці роботи алгоритму додати команди для обчислення часу виконання алгоритму. Наприклад, можна використати команди модуля time:

```
import time
...
# Введення розмірів та завдання вмісту масиву
...
time_start = time.time()          # Початковий час
...
# Алгоритм обробки
...
time_finish = time.time()         # Кінцевий час
t = time_finish - time_start      # Різниця в часі
print("Час виконання: ", t)
...
# Виведення результатів роботи алгоритму
...
```

5. Оцінити швидкодію розробленого алгоритму, заповнивши другий стовпчик таблиці 2. У першому стовпчику вказаний час (у секундах), який витрачається на роботу програми. Треба підібрати такі розміри тривимірного масиву, щоб час виконання приблизно дорівнював часу, вказаному в першому стовпчику. Ці розміри вказуються у другому стовпчику табл. 2. Під час заповнення табл. 2 треба заповнювати масив псевдовипадковими значеннями згідно з порядком, розробленим у. п. 3.

Під час заповнення кожного рядка табл. 2 краще робити не один, а декілька запусків програми, щоб переконатись у правильності результатів та унеможливити вплив різних програмно-апаратних факторів конкретного комп’ютера.

Здобувач освіти повинен бути готовий продемонструвати роботу програми викладачу під час захисту лабораторної роботи. Водночас результати, вказані у табл. 2, повинні збігатися з результатами демонстрації на комп'ютері студента. Якщо робота програми демонструється не на власному ноутбуці, а на університетському комп'ютері або на ноутбуці іншого студента, треба спочатку заповнити табл. 2 даними, отриманими на цьому комп'ютері, і тільки після цього переходити до захисту лабораторної роботи.

Таблиця 1 – Варіанти завдань до лабораторної роботи № 1

№	Індивідуальне завдання
1	Знайти середню кількість кожної корисної копалини в одному пласті (шарі ґрунту), в одному стовпці ґрунту та в загальному обсязі ґрунту
2	Знайти середню кількість трудовитрат на видобуток кожної корисної копалини в усьому об'ємі ґрунту та в кожному пласті (шарі ґрунту)
3	Для кожної корисної копалини знайти об'єм розміром 3*3*3 елементи з найбільшим вмістом цієї копалини
4	Визначити неперервну послідовність пластів (шарів ґрунту) мінімальної товщини, яка містить не менш ніж 50 % усього золота. Повторити цю задачу для інших корисних копалин
5	Знайти найбільший об'єм ґрунту, повністю заповнений нафтою, та найкраще місце для встановлення бурової вишки для її видобутку (на поверхні над цим об'ємом нафти)
6	Знайти на поверхні площину 3*3 елементи, яка є найбільш прибутковою за сукупними корисними копалинами без урахування трудовитрат
7	Знайти на поверхні найбільший прямокутник, під яким не міститься жодних корисних копалин та води
8	Знайти на поверхні прямокутник мінімального розміру, під яким міститься усе золото досліджуваного об'єму ґрунту
9	Дізнатися, як змінюється кількість нафти в таких напрямках: з заходу на схід (вздовж першого виміру масиву); з півночі на південь (вздовж другого виміру масиву); зі збільшенням глибини (вздовж третього виміру масиву). Можливі варіанти відповіді: переважно збільшується; переважно зменшується; суттєво не змінюється
10	Знайти на поверхні найбільший вертикальний розріз довжиною від одного краю до іншого (зліва направо чи зверху донизу), який є найбільш прибутковим для видобутку кам'яного вугілля без урахування трудовитрат. Знайти те саме для залізної руди. Знайти те саме для кам'яного вугілля та залізної руди разом
11	Розбити усю поверхню на чотири чверті. Для кожної чверті визначити, за яким видом корисних копалин вона є найбільш прибутковою без урахування трудовитрат
12	Усі пласти (шари ґрунту) розбиті на три третини – верхню, середню та нижню. Для кожної третини визначити, якими є середні трудовитрати на видобування золота (з урахуванням шарів ґрунту, що знаходяться понад цією третиною)
13	Знайти найбільший об'єм ґрунту, повністю заповнений водою, та кількість золота, що знаходиться нижче цього об'єму
14	Дізнатися, як змінюються трудовитрати на видобуток залізної руди в таких напрямках: з заходу на схід (вздовж першого виміру масиву); з півночі на південь (вздовж другого виміру масиву); зі збільшенням глибини (вздовж третього виміру масиву). Можливі варіанти відповіді: переважно збільшується; переважно зменшується; суттєво не змінюється
15	Знайти на поверхні площину *m*n* елементів, яка є найскладнішою за трудовитратами без урахування прибутку від корисних копалин

Таблиця 2 – Результати досліджень швидкодії роботи алгоритму

Час виконання алгоритму	**Розміри масиву**
1 секунда	
10 секунд	
30 секунд	
60 секунд	
120 секунд	

1. Програма повинна мати консольний інтерфейс.

2. Розміри масиву мають вводитися з клавіатури або задаватись константами на початку програми і надалі зберігатись у змінних *m*, *n*, *k*. Не варто використовувати замість імен *m*, *n*, *k* якісь інші імена, наприклад, *rozmir*1, *size_x* тощо. Дотримання цієї вимоги спрощує перевірку лабораторної роботи.

3. Результати роботи програм треба представити у текстовій формі (в режимі консольного виводу). Якщо розміри масиву невеликі і його вміст може «влізти» на екран (наприклад, масив розміром 3*3*3), вміст масиву можна виводити в якості результату. Якщо розміри масиву великі (наприклад, 200*300*100), виведення усього масиву на екран є абсолютно недоцільним, оскільки щось розібрати буде неможливо. Необхідно виводити тільки коротенькі кінцеві результати. Наприклад, у варіанті 1, незалежно від розмірів масиву, треба виводити лише дванадцять чисел – по три числа для кожної з чотирьох копалин.

4. Під час написання програми треба користуватися тільки стандартною бібліотекою Python, а також бібліотеками *random* і *time*. Не можна користуватися спеціальними бібліотеками для обробки даних – *numpy*, *pandas* тощо.

Структура звіту з лабораторної роботи

Звіт з лабораторної роботи має містити такі складники:

1. Титульний аркуш, оформлений за стандартами університету.

2. Індивідуальний варіант завдання (рядок із табл. 1).

3. Алгоритм роботи програми у будь-якій зручній формі (словесний опис, опис функцій, блок-схема тощо). Алгоритм повинен бути достатньо детальним, щоб інший програміст був у змозі відтворити алгоритм на будь-якій іншій мові програмування.

4. Два-три тестові приклади, що підтверджують працездатність програми (пункт 2 завдання).

5. Заповнена табл. 2.

6. Лістинг програми з докладними коментарями.

7. Висновки до лабораторної роботи.

Приклад висновків до лабораторної роботи

У цій лабораторній роботі я отримав практичні навички розв'язання задач, обробки багатомірних масивів з допомогою мови Python та оцінив швидкодію виконання розробленого алгоритму.

Приклад виконання лабораторної роботи № 1

Наведений нижче приклад виконання роботи є частиною звіту одного зі здобувачів освіти попередніх років навчання. Приклад не обов'язково відповідає високій оцінці за лабораторну роботу, може містити помилки і призначений лише ***для демонстрації загального оформлення звіту з лабораторної роботи.*** У цьому прикладі загальний лістинг програми не наводиться (з метою скорочення обсягу прикладу).

Завдання: знайти найбільший об'єм ґрунту, повністю заповнений нафтою, і найкраще місце для встановлення бурової вишки для її видобутку (на поверхні над цим об'ємом нафти).

Алгоритм роботи програми:

1. Даємо користувачу зробити вибір між запуском тестових прикладів і створенням власного масиву.

2. У випадку вибору створення власного масиву даємо користувачу ввести *m*, *n*, *k* із клавіатури і заповнюємо масив випадковими числами.

3. Шукаємо найбільшу ділянку, повністю заповнену нафтою шляхом перебору всіх можливих варіантів об'єму ґрунту.

4. Якщо ділянка повністю заповнена нафтою, то записуємо координати усіх елементів цієї ділянки у окремий масив координат. Визначаємо об'єм і порівнюємо із максимальним значенням об'єму. Також визначаємо довжину, ширину і глибину ділянки.

5. Якщо у найбільшої ділянки об'єм 1, то шукаємо у початковому масиві перший елемент нафти і записуємо його координати у масив координат.

6. Шукаємо найкраще місце для встановлення вишки на базі масиву із координатами найбільшої за розмірами ділянки. На цьому етапі пошук найкращого місця для вишки розраховується за допомогою трудовитрат.

7. Виводимо всі результати на екран.

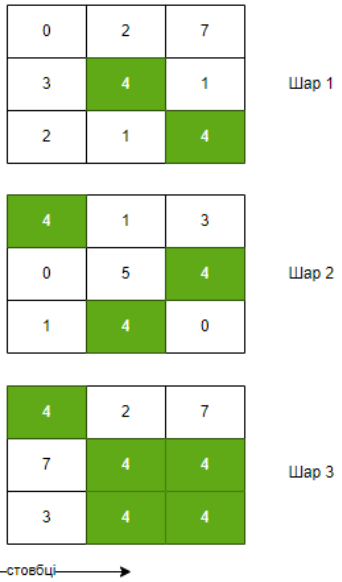
Тестові приклади:

1. Масив 3*3*3

arr = [[[2, 1, 3], [1, 4, 4], [4, 0, 4]],
[[3, 0, 7], [4, 5, 4], [1, 4, 4]],
[[0, 4, 4], [2, 1, 2], [7, 3, 7]]]

Обрахуємо вручну:

Зеленим позначені усі місця, де знаходиться нафта. Найбільшою є ділянка у третьому шарі ($k = 3$) розмірами 2*2*1. Об’єм = 4.



Координати ділянки: [0, 1, 2], [1, 1, 2], [0, 2, 2], [1, 2, 2].

Трудовитрати на встановлення бурової вишки над координатою:

[0, 1, 2]: 3 + 2 = 5;

[1, 1, 2]: 1 + 3 = 4;

[0, 2, 2]: 1 + 3 = 4;

[1, 2, 2]: 3 + 3 = 6.

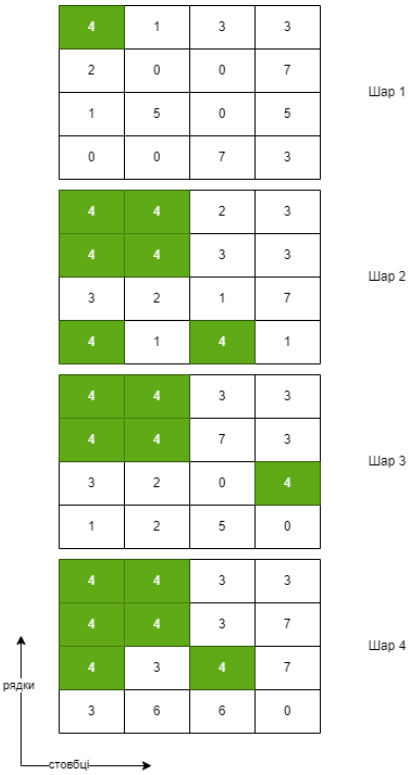
Найоптимальнішим місцем для встановлення вишки на поверхні є координати [1, 1, 0] (над [1, 1, 2]) або [0, 2, 0] (над [0, 2, 2]).

Тепер порівняємо із результатами, які обрахувала програма:

```
+---+
| Array 3 * 3 with depth 3
+---+
| The area of the soil is completely filled with oil:
| Area size   = 2 * 2 * 1
| Area volume = 4
| The best places on surface for setting up a drilling rig:
| [0, 2, 0]
+---+
| Execution time: 0.0010020732879638672
+---+
```

2. Масив 4*4*4

Arr = [[[0, 4, 1, 3], [0, 1, 2, 6], [7, 4, 5, 6], [3, 1, 0, 0]],
[[1, 3, 3, 4], [5, 2, 2, 3], [0, 1, 0, 4], [5, 7, 4, 7]],
[[2, 4, 4, 4], [0, 4, 4, 4], [0, 3, 7, 3], [7, 3, 3, 7]],
[[4, 4, 4, 4], [1, 4, 4, 4], [3, 2, 3, 3], [3, 3, 3, 3]]]



Обрахуємо вручну:

Зеленим позначені усі місця, де знаходиться нафта. Найбільшою є ділянка розмірами 2*2*3. Об’єм = 2.

Координати ділянки, найближчої до поверхні: [2, 0, 1], [3, 0, 1], [2, 1, 1], [3, 1, 1] (шар 2).

Трудовитрати на встановлення бурової вишки над координатою:

[2, 0, 1]: 5

[3, 0, 1]: 3

[2, 1, 1]: 1

[3, 1, 1]: 2

Найоптимальнішим місцем для встановлення вишки на поверхні є координата [2, 1, 0] (над [2, 1, 1]).

Тепер зрівняємо із результатами, які обрахувала програма:

```
+---+
| Array 4 * 4 with depth 4
+---+
| The area of the soil is completely filled with oil:
| Area size   = 2 * 2 * 3
| Area volume = 12
| The best places on surface for setting up a drilling rig:
| [2, 1, 0]
+---+
| Execution time: 0.003000020980834961
+---+
```

Результати досліджень швидкодії роботи алгоритму:

Час виконання алгоритму	Розміри масиву
1 секунда	14*14*14
10 секунд	21*21*20
30 секунд	25*25*24
60 секунд	28*28*28
120 секунд	32*31*31

Лістинг програми

Нижче наведена основна частина програми, де відбувається виклик функцій та замір часу роботи програми. Також тут відбувається перевірка: якщо об’єм ділянки, повністю заповненої нафтою, дорівнює 1, то відбувається пошук першого елементу нафти.

```
if __name__ == '__main__':
    m, n, k, soil = action_panel()
    coords = []

    t1 = time.time()
    volume, m1, n1, k1 = find_oil(m, n, k, soil)

    if volume == 1:
        for i in range(k):
            for j in range(n):
                for l in range(m):
                    if soil[i][j][l] == 4:
                        coords.append([i, j, l])
                        break

    best_place = find_place_for_drilling(m, n, k, soil, coords)
    t2 = time.time()
    t = t2 - t1

    print_array(m, n, k, volume, m1, n1, k1, best_place)
    print(f"| Execution time: {t} \n"
          f"+---+")
```

Нижче наведена функція, яка дає змогу користувачу обрати варіант вхідних даних для програми (запуск двох прикладів або введення власних даних):

```
def action_panel():
    while 1:
        try:
            print('1 - Run program with example 1 (array 3 * 3 * 3) with constant
values \n'
                  '2 - Run program with example 2 (array 4 * 4 * 4) with constant
values \n'
                  '3 - Run program with own values (with entering array dimension
and filling with random values)')
            val = int(input('Enter value: '))
            if val == 1:
                m, n, k = 3, 3, 3
                arr = [[[2, 1, 3], [1, 4, 4], [4, 0, 4]],
                       [[3, 0, 7], [4, 5, 4], [1, 4, 4]],
                       [[0, 4, 4], [2, 1, 2], [7, 3, 7]]]
                break
            elif val == 2:
                m, n, k = 4, 4, 4
                arr = [[[0, 4, 1, 3], [0, 1, 2, 6], [7, 4, 5, 6], [3, 1, 0, 0]],
                       [[1, 3, 3, 4], [5, 2, 2, 3], [0, 1, 0, 4], [5, 7, 4, 7]],
                       [[2, 4, 4, 4], [0, 4, 4, 4], [0, 3, 7, 3], [7, 3, 3, 7]],
                       [[4, 4, 4, 4], [1, 4, 4, 4], [3, 2, 3, 3], [3, 3, 3, 3]]]
                break
            elif val == 3:
                m, n, k, arr = enter_array_dimension()
                break
            else:
                print('Error. Choose the number from actions below')
        except ValueError:
            print('Error. Only numbers allowed')

    return m, n, k, arr
```

Функція завдання розмірності масиву з клавіатури:

```
def enter_array_dimension():
    while 1:
        try:
            m = int(input('Enter m: '))
            n = int(input('Enter n: '))
            k = int(input('Enter k: '))
            if m < 3 or n < 3 or k < 3:
                print('Error. Array dimension must be bigger than 2')
            else:
                array = fill_array(m, n, k)
                break
        except ValueError:
            print('Error. Only numbers allowed')

    return m, n, k, array
```

Нижче наведена функція заповнення масиву псевдовипадковими числами. Заповнення відбувається за допомогою двох циклів. Перший цикл заповнює кож-

ну комірку випадковим числом, а другий – додає у випадково згенеровані комірки нафту (число 4). Кількість випадково згенерованих комірок масиву, які заповняться нафтою, дорівнює розмірності масиву (об'єму ділянки). Тобто для масиву *m*n* = 5*5 заповняться нафтою 25 випадкових комірок.

```
def fill_array(m, n, k):
    arr = [[[0 for i in range(m)] for j in range(n)] for l in range(k)]

    for i in range(k):
        for j in range(n):
            for l in range(m):
                arr[i][j][l] = rd.randint(0, 7)

    for temp in range(m * n):
        i1, j1, l1 = rd.randint(0, k-1), rd.randint(0, n-1), rd.randint(0, m-1)
        arr[i1][j1][l1] = 4
    return arr
```

Нижче наведена функція, що робить пошук найбільшого об'єму, повністю заповненого нафтою, та визначає його розміри. Для зменшення громіздкості звіту основні коментарі наведені безпосередньо у лістингу цієї функції. Випадок із повною відсутністю нафти ми не розглядаємо, оскільки заповнення масиву відбувається так, що там обов'язково міститиметься нафта.

```
def find_oil(m, n, k, soil):
    # лічильник для кількості нафти
    count_oil = 0
    for i in range(k):
        for j in range(n):
            for l in range(m):
                if soil[i][j][l] == 4:
                    count_oil += 1

    max_volume = 0
    max_length, max_width, max_depth = 0, 0, 0

    # якщо кількість клітинок з нафтою дорівнює максимальній розмірності земельної ділянки, тоді всюди нафта
    if count_oil == (m * n * k):
        print("The area is completely filled with oil")
        return m * n * k, m, n, k
    # інакше шукаємо найбільший об'єм ґрунту повністю заповнений нафтою
    else:
        # побудова циклів для перебору усіх можливих варіантів об'єму ґрунту
        for i in range(k):
            for j in range(n):
                for l in range(m):
                    if soil[i][j][l] == 4:
                        x = 0
                        while (i + x < k) and (soil[i + x][j][l] == 4):
                            y = 0
                            while (j + y < n) and (soil[i][j + y][l] == 4):
                                z = 0
                                while (l + z < m) and (soil[i][j][l + z] == 4):
```

```
                                    # перевірка на те, чи даний об'єм ґрунту повністю заповнений нафтою
                                    full_oil = is_full_oil(i, i + x, j, j + y, l, l + z, soil)
                                    if full_oil == 1:
                                        # перевірка на те, чи об'єм нового ґрунту більший за попередній
                                        for index in range(m * n * k):
                                            volume = ((index + x) - index + 1) * ((j + y) - j + 1) * ((l + z) - l + 1)
                                            if (volume > max_volume):
                                                # перевизначення довжин та максимального об'єму
                                                max_volume = volume
                                                max_length = (index + x) - index + 1
                                                max_width = (j + y) - j + 1
                                                max_depth = (l + z) - l + 1
                                    z += 1
                                y += 1
                            x += 1
    return max_volume, max_length, max_width, max_depth
```

Нижче наведена допоміжна функція, що викликається у попередній функції і перевіряє, чи є обраний прямокутник повністю заповнений нафтою. Якщо так, функція додає координати ділянки до масиву із координатами ділянки.

```
def is_full_oil(d1, d2, w1, w2, l1, l2, mas):
    temp_length = 0
    for i in range(d1, d2 + 1):
        for j in range(w1, w2 + 1):
            for l in range(l1, l2 + 1):
                if mas[i][j][l] != 4:
                    return 0
                temp_length += 1
        if [i, j, l] not in coords and temp_length != 1:
            coords.append([i, j, l])
    return 1
```

Розглянемо функцію знаходження найкращого місця для встановлення бурової вишки. Для визначення найкращого місця необхідні такі змінні:

– *new_coords:* масив для координат, які знаходяться найближче до поверхні;

– *depth_index:* індекс глибини, який потрібен для того, щоб проходитись по усіх клітинках, які знаходяться над *new_coords*;

– *best_cost* і *cost:* для знаходження мінімальної трудовитрати кінцевий результат міститиметься у змінній *best_cost*;

– *place:* для шуканої найоптимальнішої координати, під якою знаходяться елементи із найменшою складністю видобутку.

За допомогою першого циклу знаходиться координата глибини, на якій елементи розміщені найближче до поверхні. Другий цикл знаходить і записує у масив координати елементів, які розміщені найближче до поверхні (за допомогою

знайденого у попередньому циклі шару). Третій цикл виконує розрахунок трудовитрат і пошук місця на видобуток над кожною клітинкою нафти і записує координати найоптимальнішого місця у змінну *place* і загальну складність видобутку у *cost*.

```
def find_place_for_drilling(m, n, k, soil, coords_arr):
    new_coords = []
    depth_index = k
    best_cost = 1000
    cost = 0
    place = []

    for coord in coords_arr:
        if coord[2] <= depth_index:
            depth_index = coord[2]

    for coord in coords_arr:
        if coord[2] == depth_index:
            new_coords.append(coord)

    for i in range(k):
        for j in range(n):
            for l in range(m):
                for item in new_coords:
                    if i == item[0] and j == item[1] and l == item[2]:
                        height = 0
                        while height < depth_index:
                            if soil[i][j][height] == 0:
                                cost += 1
                            elif soil[i][j][height] == 1:
                                cost += 2
                            elif soil[i][j][height] == 2:
                                cost += 5
                            else:
                                cost += 3
                            height += 1

                        if cost < best_cost:
                            best_cost = cost
                            place = [i, j, 0]
                        cost = 0
    return place
```

Функція виведення результатів на екран:

```
def print_array(m, n, k, volume, length, width, depth, place):
    print(f'+---+\n'
          f'| Array {m} * {n} with depth {k}')
    print(f'+---+ \n'
          f'| The area of the soil is completely filled with oil: \n'
          f'| Area size   = {length} * {width} * {depth} \n'
          f'| Area volume = {volume} \n'
          f'| The best places on surface for setting up a drilling rig: \n'
          f'| {place} \n'
          f'+---+')
```

Лабораторна робота № 2
Векторні операції в NumPy

Метою лабораторної роботи є практичне засвоєння можливостей бібліотеки NumPy для обробки багатовимірних масивів із використанням векторних операцій над масивами типу *ndarray*.

Завдання до лабораторної роботи

Індивідуальний варіант завдання береться з першої лабораторної роботи. Виконання першої лабораторної роботи до виконання другої є обов'язковим, оскільки деякі результати першої лабораторної роботи є вхідними даними для другої.

Для індивідуального варіанта завдання необхідно виконати таке:

1. Реалізувати алгоритм із лабораторної роботи № 1 на мові Python із використанням векторних операцій бібліотеки NumPy над масивами *ndarray*.

2. Адаптувати алгоритм заповнення тривимірного масиву псевдовипадковими числами (пункт 3 завдання з лабораторної роботи № 1) до масиву типу *ndarray*.

3. Заповнити таблицю 1 (див. нижче) у такий спосіб:

– стовпець t_1 повторює стовпець «Час виконання алгоритму» табл. 2 з лабораторної роботи № 1;

– стовпець «Розміри масиву» повторює стовпець «Розміри масиву» таблиці 2 з лабораторної роботи № 1;

– у стовпець t_2 заносяться експериментально отримані витрати часу (в секундах) на виконання алгоритму із використанням NumPy для розмірів масиву, вказаних у першому стовпці;

– в останній стовпець вносяться значення, які показують, у скільки разів значення t_1 довше за значення t_2, тобто у скільки разів збільшується швидкодія програми за рахунок NumPy.

Таблиця 1 – Витрати часу на обробку масиву

Розміри масиву	t_1	t_2	t_1 / t_2
	1 секунда		
	10 секунд		
	30 секунд		
	60 секунд		
	120 секунд		

Вимоги до розроблюваної програми

Вимоги співпадають з пунктами 1–3 вимог з першої лабораторної роботи.

Структура звіту з лабораторної роботи

1. Титульний аркуш.

2. Індивідуальний варіант завдання (із лабораторної роботи № 1).

3. Алгоритм роботи програми у будь-якій зручній формі (словесний опис, блок-схема тощо). Алгоритм повинен бути достатньо детальним, щоб інший програміст був у змозі відтворити алгоритм на будь-якій іншій мові програмування. В алгоритмі повинні бути чітко виділені місця, де використовуються векторні операції над масивами NumPy.

4. Два-три тестові приклади, що підтверджують працездатність програми (за аналогією з лабораторною роботою № 1). Приклади не повинні братися з лабораторної роботи № 1, а повинні бути розроблені заново за допомогою програми, що використовує NumPy.

5. Заповнена табл. 1.

6. Лістинг програми з докладними коментарями.

7. Висновки до лабораторної роботи.

Приклад висновків за лабораторною роботою

Під час лабораторної роботи я отримав практичні навички роботи з бібліотекою NymPy. Я навчився використовувати векторні операції над масивами типу *ndarray* під час вирішення задач обробки багатовимірних масивів.

Приклад виконання лабораторної роботи № 2

Наведений нижче приклад виконання роботи є частиною звіту одного зі здобувачів освіти попередніх років навчання. Приклад не обов'язково відповідає високій оцінці за лабораторну роботу, може містити помилки і призначений лише ***для демонстрації загального оформлення звіту з лабораторної роботи.*** У цьому прикладі загальний лістинг програми не наводиться (з метою скорочення обсягу прикладу).

Завдання: знайти найбільший об'єм ґрунту, повністю заповнений нафтою, та найкраще місце для встановлення бурової вишки для її видобутку (на поверхні над цим об'ємом нафти).

Алгоритм роботи програми (аналогічний до алгоритму попередньої лабораторної):

1. Даємо користувачу зробити вибір між запуском тестових прикладів і створенням власного масиву.

2. У випадку вибору створення власного масиву даємо користувачу ввести (m, n, k) із клавіатури і заповнюємо масив випадковими числами за допомогою бібліотеки NumPy.

3. Шукаємо найбільшу ділянку, повністю заповнену нафтою шляхом перебору всіх можливих варіантів об'єму ґрунту.

4. Якщо ділянка повністю заповнена нафтою, то записуємо координати усіх елементів цієї ділянки в окремий масив координат. Визначаємо об'єм і порівнює-

мо із максимальним значенням об’єму. Також визначаємо довжину, ширину і глибину ділянки.

5. Якщо у найбільшої ділянки об’єм 1, то шукаємо у початковому масиві перший елемент нафти і записуємо його координати у масив координат за допомогою бібліотеки NumPy.

6. Шукаємо найкраще місце для встановлення вишки на базі масиву із координатами найбільшої за розмірами ділянки. На цьому етапі пошук найкращого місця для вишки розраховується за допомогою трудовитрат. На цьому етапі також відбувається використання бібліотеки NumPy.

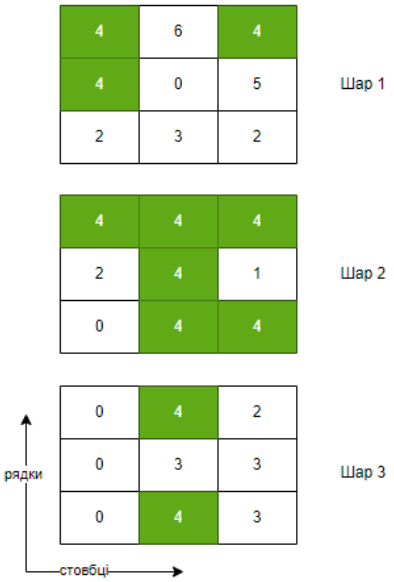
7. Виводимо усі результати на екран.

Тестові приклади

Масив 3*3*3:

arr = [[[2, 0, 0], [3, 4, 4], [2, 4, 3]],
[[4, 2, 0], [0, 4, 3], [5, 1, 3]],
[[4, 4, 0], [6, 4, 4], [4, 4, 2]]]

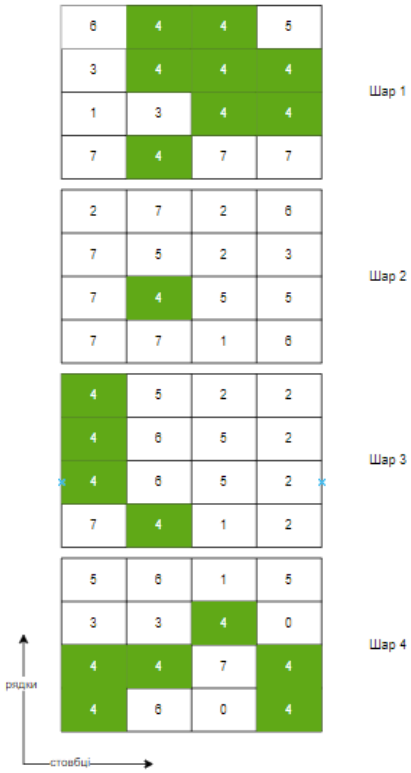
```
+---+
| Array 3 * 3 with depth 3
+---+
| The area of the soil is completely filled with oil:
| Area size   = 3 * 1 * 1
| Area volume = 3
| The best places on surface for setting up a drilling rig:
| [2, 0, 0]
+---+
| Execution time: 0.0009999275207519531
+---+
```



Масив 4*4*4:

arr = [[[7, 7, 7, 4], [4, 7, 4, 6], [7, 1, 1, 0], [7, 6, 2, 4]],
[[1, 7, 4, 4], [3, 4, 6, 4], [4, 5, 5, 7], [4, 5, 2, 4]],
[[3, 7, 4, 3], [4, 5, 6, 3], [4, 2, 5, 4], [4, 3, 2, 0]],
[[6, 2, 4, 5], [4, 7, 5, 6], [4, 2, 2, 1], [5, 6, 2, 5]]]

```
+---+
| Array 4 * 4 with depth 4
+---+
| The area of the soil is completely filled with oil:
| Area size   = 2 * 2 * 1
| Area volume = 4
| The best places on surface for setting up a drilling rig
| [1, 3, 0]
+---+
| Execution time: 0.0010039806365966797
+---+
```



Результати досліджень швидкодії роботи алгоритму:

Розміри масиву	t_1	t_2	t_1 / t_2
14*14*14	1 секунда	0,042	23,81
21*21*20	10 секунд	0,209	47,85
25*25*24	30 секунд	0,419	71,6
28*28*28	60 секунд	0,741	80,97
32*31*31	120 секунд	1,437	83,51

Таблиця містить порівняльні результати оцінки швидкодії роботи алгоритму із цієї та попередньої лабораторних робіт. Можна визначити значну різницю у швидкості, яка іноді перевищує 80 разів. Це підтверджує потужність бібліотеки NumPy під час роботи з багатовимірними масивами даних.

Розглянемо фрагменти коду, які виконуються за допомогою бібліотеки NumPy:

1. Заповнення масиву випадковими числами.

```
# створюємо масив потрібної розмірності із елементів в діапазоні від 0 до 7
arr = np.random.randint(0, 8, size=(k, n, m))
```

2. Розрахунок загальної кількості нафти в усьому об’ємі ґрунту.

```
# лічильник для кількості нафти
count_oil = np.count_nonzero(soil == 4)
```

3. Функція пошуку найкращого місця для встановлення бурової вишки.

```
def find_place_for_drilling(soil, coords_arr):
    cost = 0
    costs_arr = []
    coords_arr = np.asarray(coords_arr)
    transposed_coords = np.transpose(coords_arr)
    # знаходимо найменше k (індекс глибини), шляхом транспонування матриці
    # (шукаємо мінімум у третьому рядку, де знаходяться всі координати глибини k)
    depth_index = transposed_coords[2].min()
    # знаходимо індекси усіх координат, де k дорівнює мінімальному індексу глибини і записуємо ці елементи у масив координат, найближчих до поверхні
    new_coords = coords_arr[np.where(transposed_coords[2] == depth_index)]

    for item in new_coords:
        height = depth_index - 1
        item_copy = item
        while height >= 0:
            item_copy[2] = height
            cost += calculate_cost(soil[item_copy[0], item_copy[1], item_copy[2]])
            height -= 1
        costs_arr.append(cost)  # записуємо значення трудовитрат над кожною коміркою у масив
        cost = 0

    min_index = np.where(costs_arr == np.min(costs_arr))  # шукаємо індекс найменшого значення трудовитрати
```

```
    return new_coords[min_index[0][0]]  # повертаємо першу координату із міні-
мальною трудовитратою (якщо їх декілька)
```

Цикл реалізований у такий спосіб, щоб зайвий раз не відбувався прохід по непотрібних нам комірках, а лише тільки по тим комірках, які знаходяться над найбільшим об’ємом нафти. Це реалізується шляхом зменшення значення k, тобто координати глибини.

Принцип знаходження мінімальної висоти за трудовитратами є таким: у масив трудовитрат записуються загальні трудовитрати усієї висоти над кожною клітинкою верхньої поверхні паралелепіпеда з нафтою, а потім за допомогою бібліотеки NumPy знаходиться мінімальне значення та його індекс у масиві. Після чого повертається координата з нафтою за знайденим індексом. Це і є найкраще місце для встановлення бурової вишки. Якщо мінімальних значень декілька, наприклад, декілька трійок, то повертається індекс першої трійки, яка зустрічається у масиві трудовитрат.

Для зручності обрахунок значень трудовитрат винесений у окрему функцію:

```
def calculate_cost(cell_value):
    if cell_value == 0:
        return 1
    elif cell_value == 1:
        return 2
    elif cell_value == 2:
        return 5
    else:
        return 3
```

4. Знаходження першої клітинки нафти у випадку, якщо об’єм максимальної ділянки з нафтою дорівнює 1:

```
     if volume == 1:
    # шукаємо клітинки із нафтою і записуємо у масив координат перший елемент
    oil_index = np.count_nonzero(soil == 4)
    coords.append([oil_index[0][0], oil_index[1][0], oil_index[2][0]])
```

Лабораторна робота № 3
Робота з бібліотекою Matplotlib

Метою лабораторної роботи є практичне засвоєння можливостей бібліотеки Matplotlib для візуалізації даних, що зберігаються у багатовимірних масивах типу *ndarray*.

Завдання до лабораторної роботи

Індивідуальним варіантом завдання є повністю виконана друга лабораторна робота, результатом якої є програма з обробки тривимірного масиву, написана мовою Python із використанням бібліотеки NumPy.

У цій лабораторній роботі необхідно зробити презентацію, в якій докладно відобразити результати другої лабораторної роботи. Презентація повинна являти собою DOC- або PDF-файл, у якому у графічному і текстовому вигляді продемонстровано, що саме зроблено у другій лабораторній роботі. Необхідно показати таке:

– який алгоритм реалізовано у другій лабораторній роботі;

– як цей алгоритм працює для масивів маленького розміру (приблизно 5*5*5) та масивів більшого розміру;

– як поводить себе алгоритм у якихось особливих ситуаціях (для різних варіантів завдання це можуть бути різні ситуації: масив безкорисних копалин, одношаровий масив, масив, заповнений тільки золотом, тощо);

– яка ефективність алгоритму на базі NumPy, порівняно з алгоритмом із першої лабораторної роботи;

– якісь інші результати (на власний розсуд).

Загалом презентація повинна надавати глядачу усі необхідні відомості про другу лабораторну роботу та її результати. Якихось програмних елементів другої лабораторної роботи у презентації бути не повинно (як-от опис функцій, фрагменти коду, блок-схеми, лістинг тощо). Рисунки, наведені у презентації, повинні демонструвати вміст тривимірного масиву (чи його частин – шарів, стовпців, тривимірних підмасивів) і отримані чисельні результати роботи алгоритму. Можна візуалізовувати результати якихось окремих кроків алгоритму. Доцільно показати у графічному вигляді результати порівняння швидкодії алгоритмів з першої і другої лабораторних робіт.

Усі графічні матеріали, які наводяться у презентації, повинні бути підготовлені за допомогою бібліотеки Matplotlib. Це можуть бути різноманітні графіки, діаграми, контурні графіки, теплові карти тощо. Здобувач освіти сам обирає, у якій графічній формі йому зручніше представити ту чи іншу інформацію. Водночас не дозволяється використовувати якісь інші бібліотеки, окрім Matplotlib.

Вимоги до розроблюваної програми

Під час використання бібліотеки Matplotlib усі графіки будуються програмно. Для цього необхідно взяти програмний код другої лабораторної роботи і додати у потрібних місцях команди виклику функцій бібліотеки Matplotlib.

Наприклад, якщо здобувач бажає візуалізувати по черзі кожен із п’яти шарів тривимірного масиву у вигляді теплової карти, він повинен:

1. Знайти в коді програми місце, в якому вміст шарів масиву буде вже сформований і готовий до візуалізації.

2. Організувати цикл, у якому будуть перебиратись окремі шари масиву.

3. У тілі циклу викликати функцію pyplot.imshow(), передавши в неї поточний шар масиву.

4. Скопіювати вміст вікна Matplotlib, що містить побудований графік, і вставити його у файл презентаці. У Windows для копіювання вмісту вікна програми в буфер обміну можна використовувати комбінацію клавіш ALT + PrtScr.

5. У файлі презентації правильно оформити вставлену картинку (розташувати по центру, підібрати масштаб тощо) та надати кілька рядків текстового опису: «На цьому рисунку можна бачити вміст верхнього шару ґрунту. Кольори теплової карти демонструють відсутність золота в цьому шарі».

6. «Прокрутити» цикл перебору шарів та повторити кроки 3–5 для інших чотирьох шарів масиву.

Подібні програмні вставки слід додавати в тих місцях програми, де є потреба візуалізувати якісь дані. Зрозуміло, що не слід візуалізовувати усе підряд. Треба заздалегідь продумати порядок подання інформації глядачу презентації, після чого візуалізовувати дані у потрібному порядку.

Не треба домагатись того, щоб програма одночасно виводила усі рисунки, які будуть у презентації. Таких рисунків може бути 10, 20 і більше. Варто використовувати програму лише як засіб для побудови рисунків, але остаточне розташування і опис рисунків будуть тільки у презентації. Тобто: додали в одному місці програми фрагмент коду для побудови графіків, побудували графіки, скопіювали їх у презентацію, а потім закоментували цей фрагмент коду. Далі додаєте подібний фрагмент в іншому місці програми, і так далі. У звіті з лабораторної роботи слід навести повний лістинг другої лабораторної роботи з усіма закоментованими фрагментами візуалізації.

Структура звіту з лабораторної роботи

1. Титульний аркуш.

2. Індивідуальний варіант завдання (із лабораторної роботи № 1).

3. Програмний код другої лабораторної роботи із доданими фрагментами для візуалізації тих чи інших даних.

4. Висновки до лабораторної роботи.

5. Окремий файл з презентацією (в форматі DOC або PDF).

Приклад виконання лабораторної роботи № 3

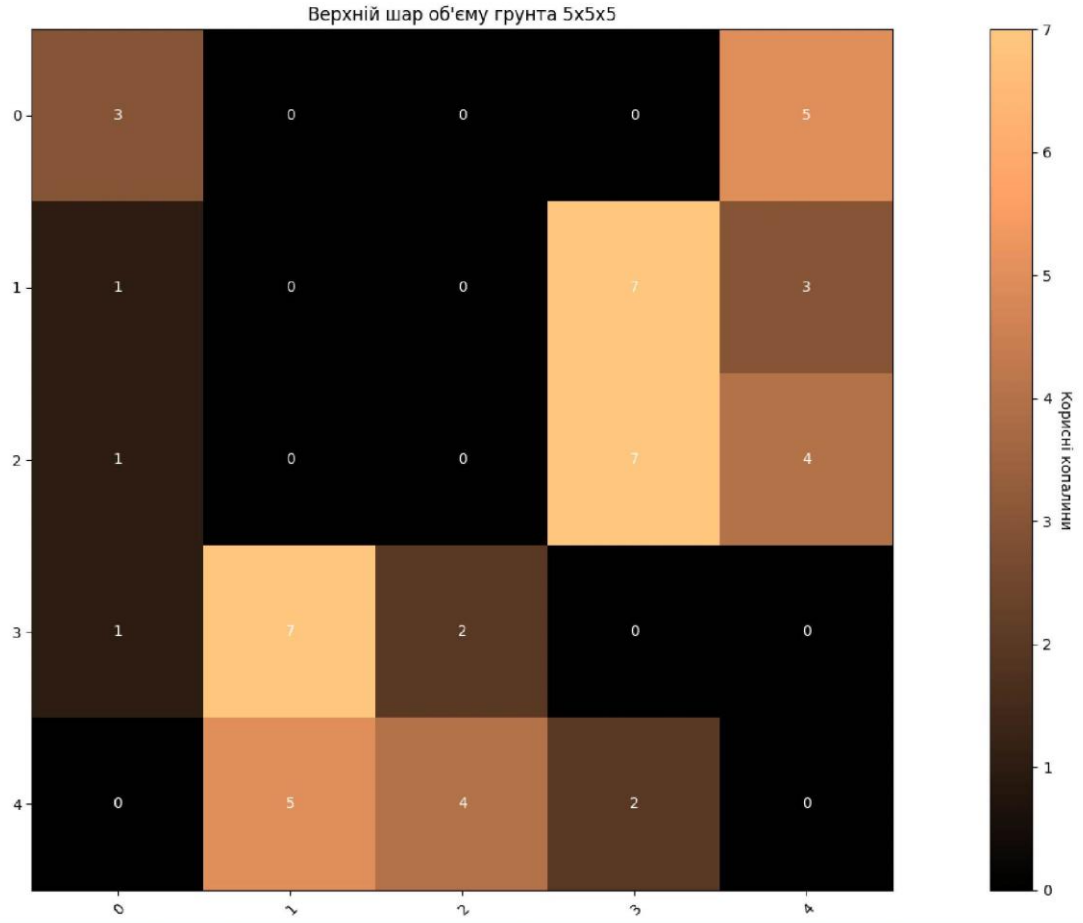
Наведений нижче приклад виконання роботи розроблений одним зі здобувачів освіти попередніх років навчання. У прикладі наведено лише презентацію, окрім якої повинен бути зроблений сам звіт із лістингом програми (див. вище). Презентація зроблена в альбомному (горизонтальному) форматі, що загалом не є обов'язковим.

Цей приклад не пов'язаний із прикладами у двох попередніх лабораторних роботах, може містити помилки і призначений лише ***для демонстрації загального оформлення презентації.***

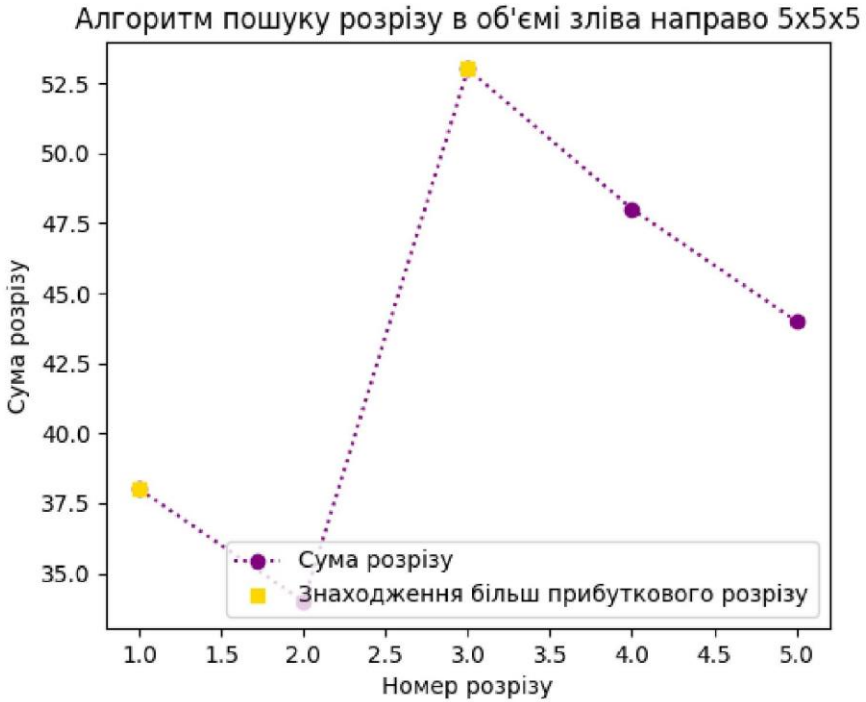
Лабораторна робота №3
з дисципліни «Технології обробки, зберігання та візуалізації даних»
на тему «Робота з бібліотекою Matplotlib»

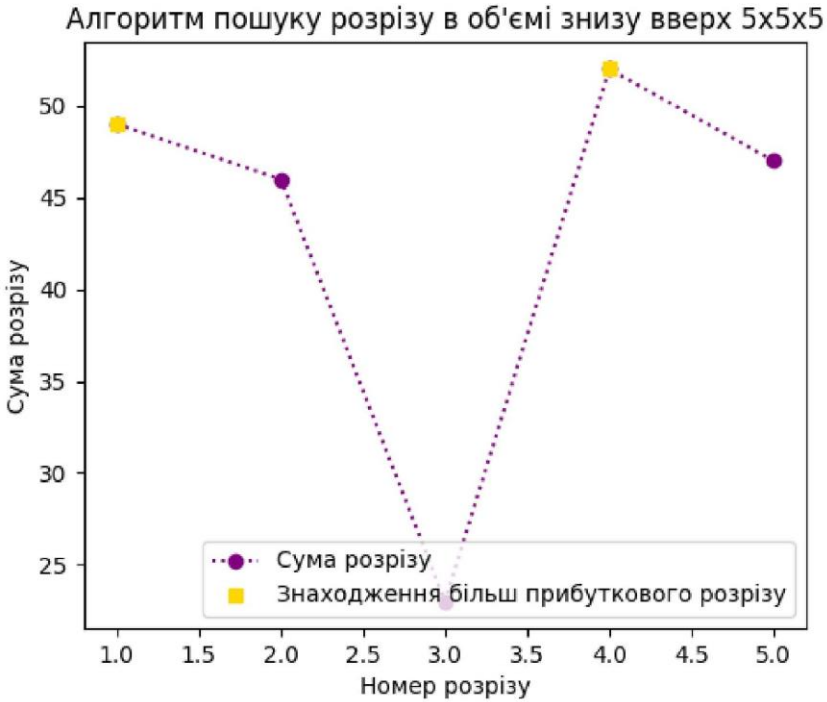
Виконав:
студент 1 курсу СО «Магістр»
спеціальності 122 Комп'ютерні науки
Гаврилов Андрій Юрійович

На рисунку нижче представлена теплова карта для вмісту корисних копалин у верхньому шарі грунту об’ємом 5x5x5

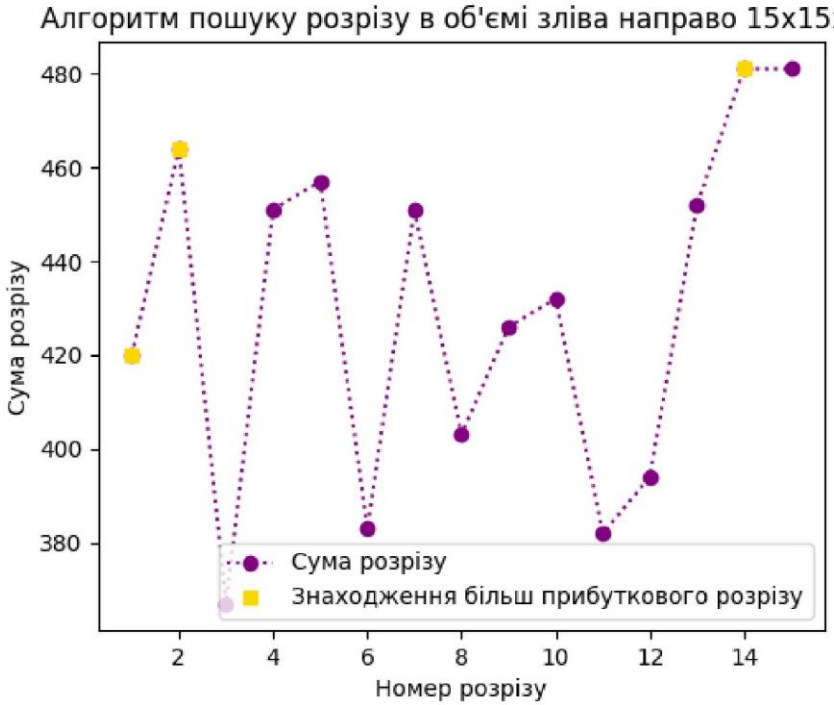


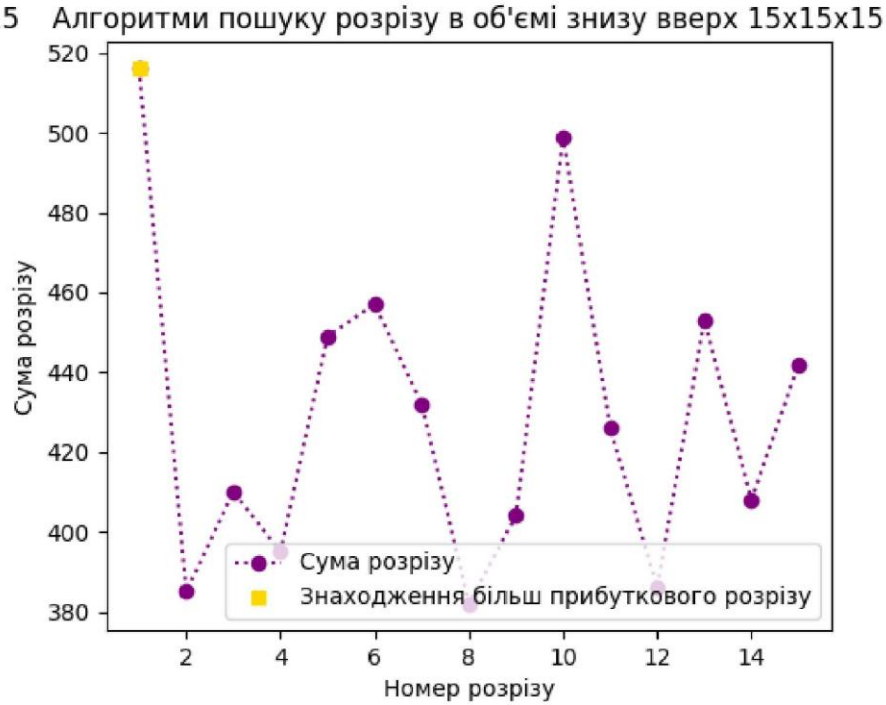
Тут ми можемо побачити роботу алгоритма пошуку найкращого розрізу зліва направо та знизу вверх для об’єму грунта 5x5x5



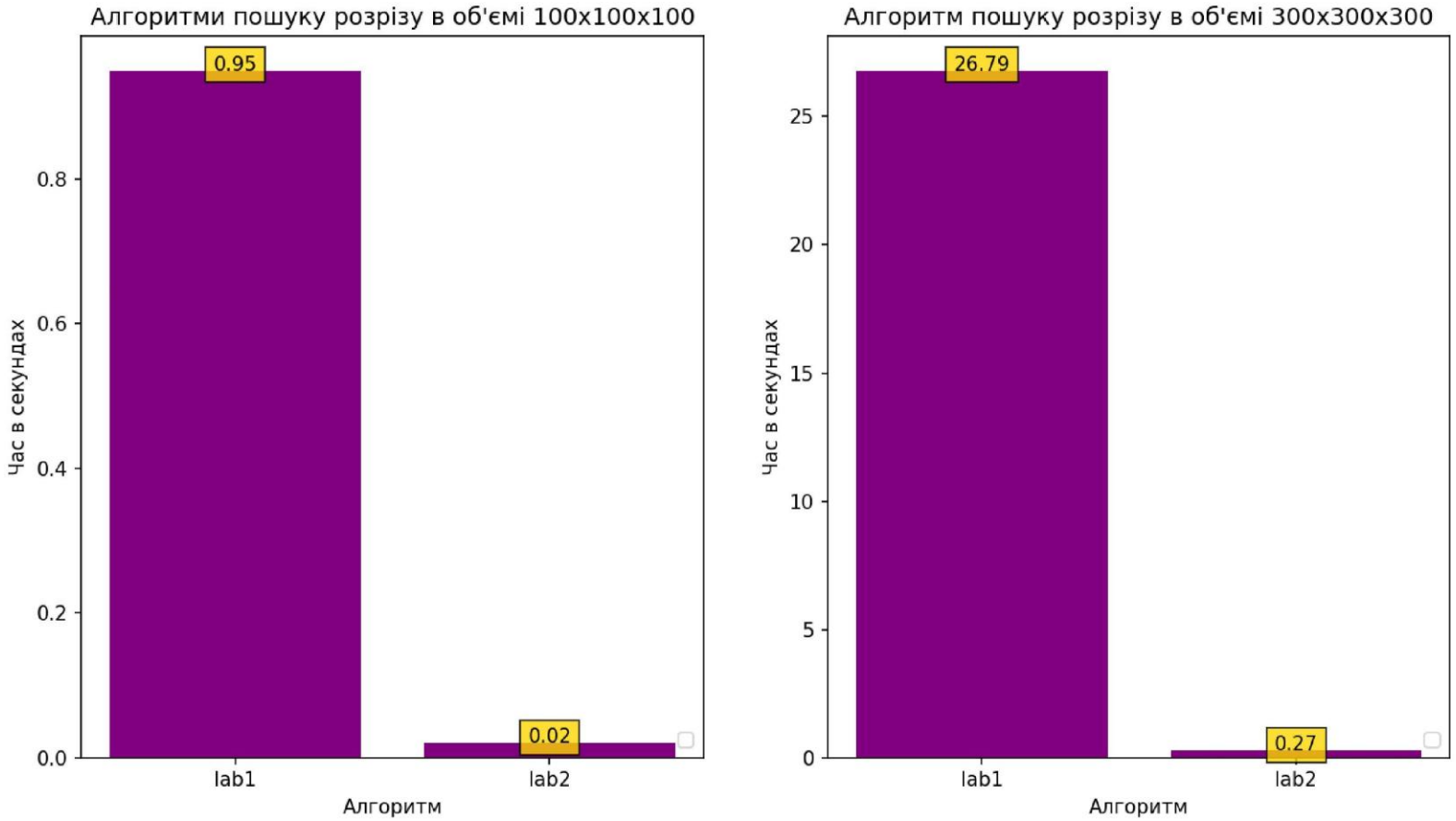


На даному малюнку зображено роботу алгоритма пошуку найкращого розрізу зліва направо та знизу вверх 15x15x15





Нижче надано порівняння роботи алгоритмів з першої та другої лабораторної в задачах пошуку розрізу в об’ємах 100x100x100 та 300x300x300



Індивідуальне творче завдання

Метою індивідуального творчого завдання (ІТЗ) є знайомство студентів із сучасними бібліотеками мови Python, призначеними для візуалізації даних.

В індивідуальному творчому завданні треба ще раз виконати лабораторну роботу № 3, використавши замість бібліотеки Matplotlib альтернативні бібліотеки візуалізації даних. Дозволяється використовувати якусь одну альтернативну бібліотеку або декілька різних. Результатом виконання ІТЗ є презентація, подібна презентації з лабораторної роботи № 3. Дозволяється, щоб ця презентація була більш об'ємною і містила додаткові рисунки, яких не було в презентації третьої лабораторної роботи. Презентація оформлюється одному з форматів: DOC (бажано), PDF або PowerPoint.

Бібліотека Matplotlib, вивченню якої присвячена третя лабораторна робота, вважається класичною бібліотекою для візуалізації даних у мові Python. Сьогодні вона вважається трохи застарілою і використовується не надто часто. Утім знання Matplotlib є обов'язковим принаймні з таких причин:

– із використанням Matplotlib написано доволі багато програмного коду, який треба вміти розуміти;

– на основі Matplotlib розробена низка більш сучасних бібліотек, які використовують у якості способу зберігання даних масиви NumPy і у своєму функціоналі багато в чому базуються на бібліотеці Matplotlib.

Сьогодні для візуалізації різноманітних даних використовуються Python-бібліотеки: Plotly, Seaborn, Bokeh, Altair, Missngno, Bashplotlib, Pygal, Ggplot, Gleam, Leather, NetworkX, Holoviews, Vincent, Toyplot, Yellowbrick, Glumpy, Chaco, PyQTgraph, Geoplotlib, MayaVi та інші. Кожна з бібліотек має свої особливості і не обов'язково підходить для заміни бібліотеки Matplotlib у межах лабораторної роботи № 3. Здобувачам освіти пропонується самостійно проаналізувати можливості бібліотек і обрати ті, які прийшлися до вподоби з тих чи інших причин. Саме у цьому полягає творчий складник індивідуального творчого завдання.

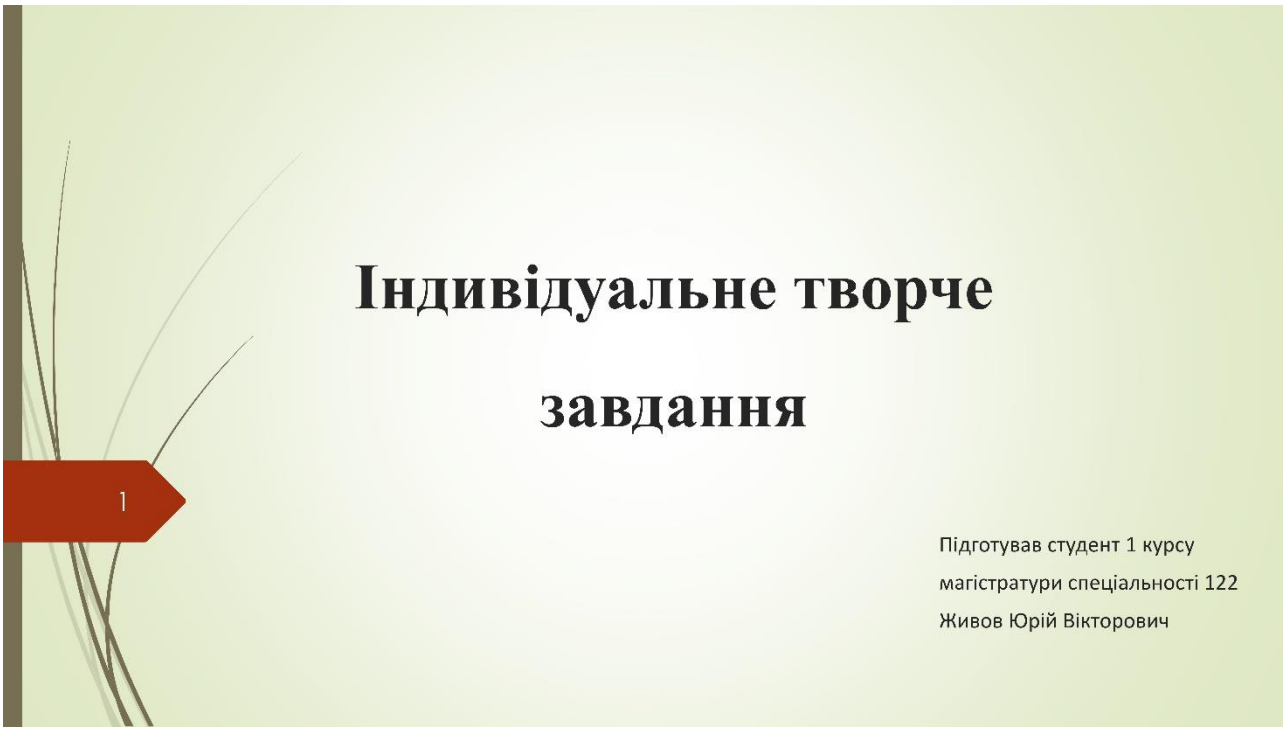
Окрім презентації, необхідно підготувати звіт з ІТЗ, структура якого схожа на звіт з лабораторної роботи № 3, але на титульному листі замість «Звіт з лабораторної роботи № 3» вказується «Звіт з індивідуального творчого завдання». Основну частину звіту займає лістинг програми. Звіт з ІТЗ і презентація – це два окремі файли.

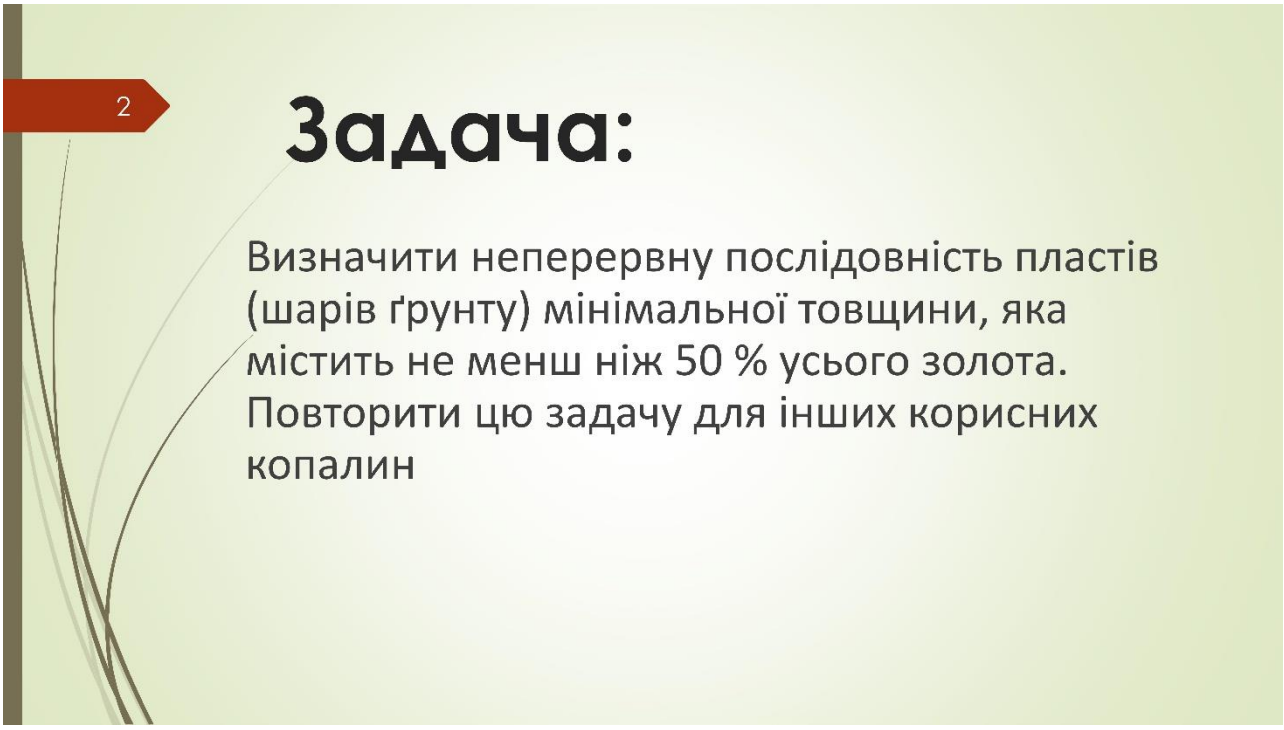
Як і лабораторні роботи, ІТЗ передбачає процедуру захисту. Під час захисту перед викладачем звіту з ІТЗ здобувач освіти повинен бути готовим пояснити усі фрагменти коду програми, а також зміст і доцільність кожного рисунку зробленої презентації.

Приклад виконання індивідуального творчого завдання

Наведений нижче приклад виконання ІТЗ розроблений одним зі здобувачів освіти попередніх років навчання. У прикладі наведено лише презентацію, окрім якої повинен бути зроблений сам звіт із лістингом програми. Презентація зроблена в альбомному (горизонтальному) форматі, що загалом не є обов'язковим.

Цей приклад не пов'язаний із розглянутими раніше прикладами, може містити помилки і призначений лише ***для демонстрації загального оформлення презентації.***





3

Бібліотека для візуалізації даних:



seaborn

4

Алгоритм роботи програми

Пошук золота

5

Для прикладу роботи алгоритму розглянимо наступну послідовність шарів. Розмір землі 3x3x4.

7	0	7
7	0	7
0	0	6

Верхній шар

6	0	6
6	0	4
0	0	6

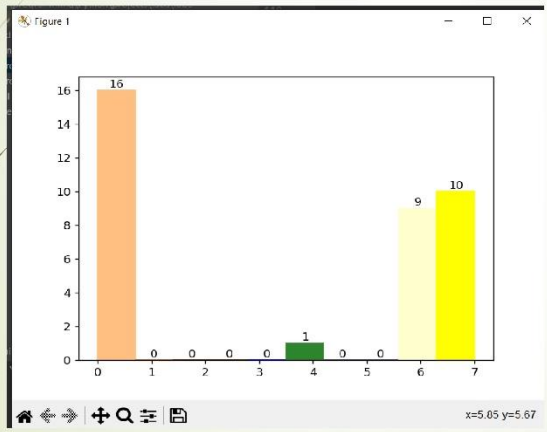
7	0	6
6	0	7
0	0	6

7	0	7
7	0	7
0	0	6

Нижній шар

6

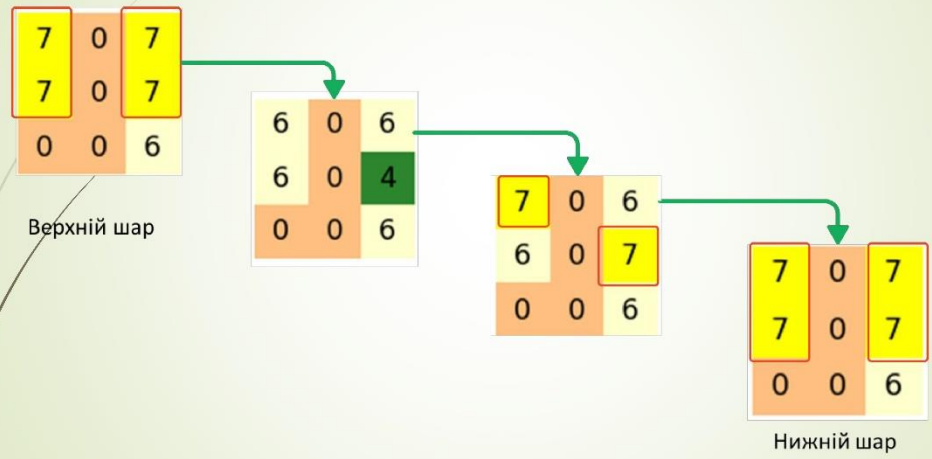
1. Знаходимо загальну кількість копалини в пластах ґрунту.



В даному прикладі загальна кількість золота 10 одиниць (ідентифікатор 7)

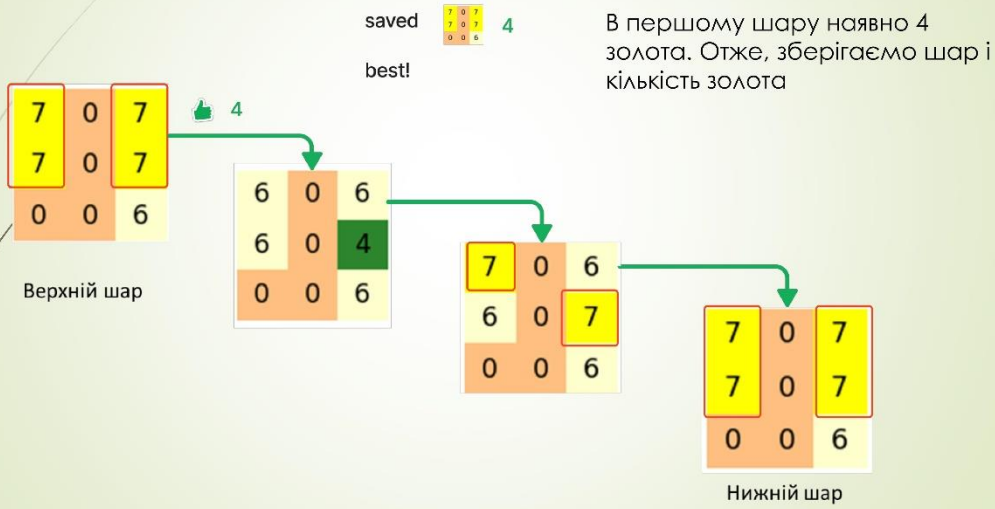
7

2. Поетапно проходимо по шарам. Підраховуємо загальний вміст потрібної копалини в кожному пласті ґрунту.



8

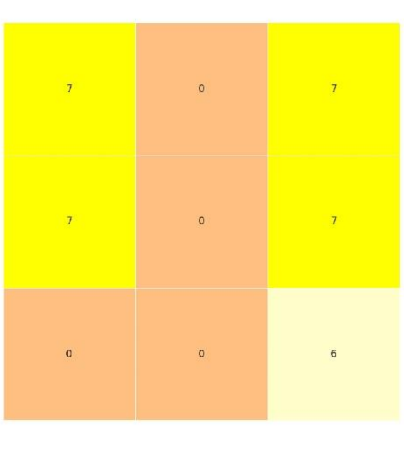
2.1 Якщо в шару наявна копалина – запам'ятовуємо цей пласт і кількість копалини в ньому



В першому шару наявно 4 золота. Отже, зберігаємо шар і кількість золота

9

Перший шар і кількість копалин в ньому

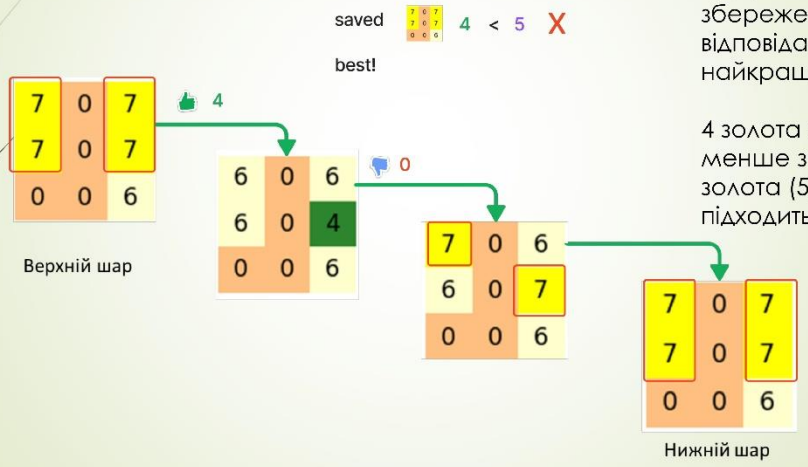


10

2.2 Якщо відсутня - перевіряємо чи було в нас запам'ятовано попередні пласти. Якщо так, то зберігаємо як кращу послідовність, якщо вона відповідає умові пошуку (містить не менш ніж 50 % усього золота – 10/2 = 5) і вона краща за попередню (мінімальної товщини).

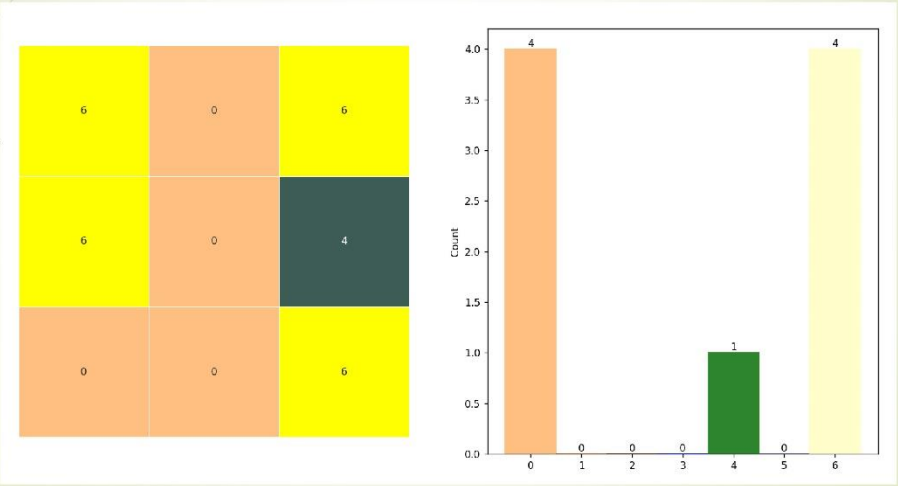
В другому шару золото відсутнє. Перевіряємо чи збережені пласти відповідають умові і є найкращими.

4 золота в першому шарі менше за 50% усього золота (5). Отже, він нам не підходить.



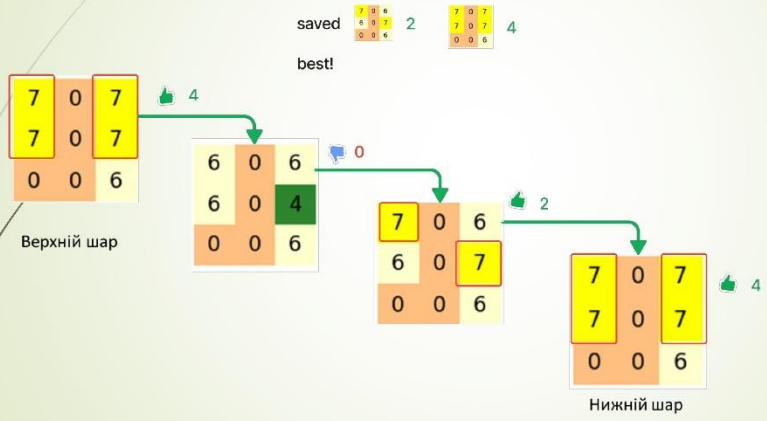
11

Другий шар та кількість копалин в ньому



12

Повторяємо ітерації для всіх інших шарів



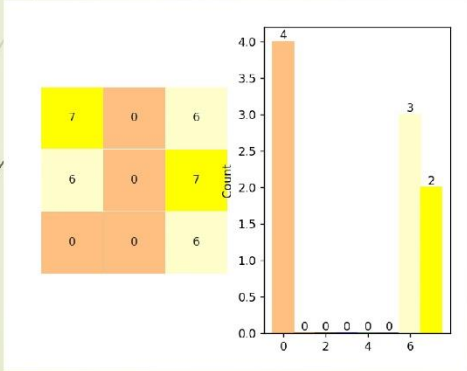
Шар 3 містить 2 золота, отже зберігаємо його. Шар 4 містить 4 золота, теж зберігаємо.

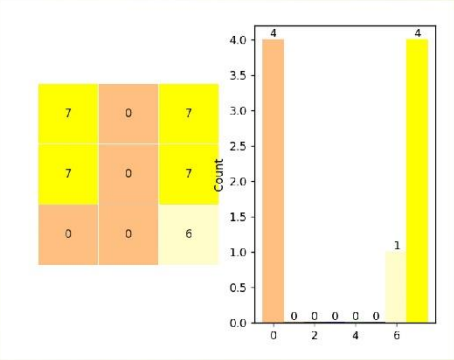
Оскільки закінчилися пласти ґрунту виконуємо ще раз перевірку.

Збережена послідовність містить 2 + 4 золота, що задовольняє умову (>=5), а також є найкращою в даному прикладі (єдина послідовність яка задовольняє умову)

13

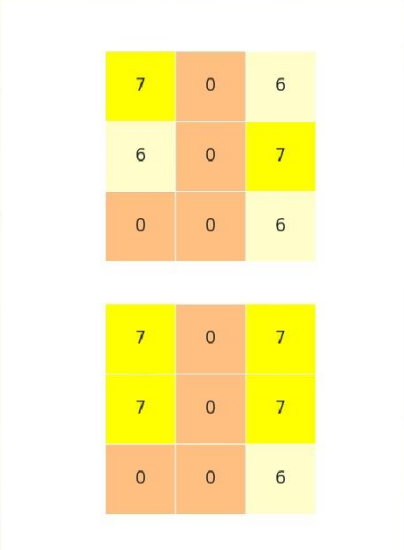
Третій та четвертий шар





14

3. Вивід найкращої послідовності



Останні два пласта є найкращою послідовністю з вмістом золота 6

15

Виконання програми для масива розміру 5x5x5

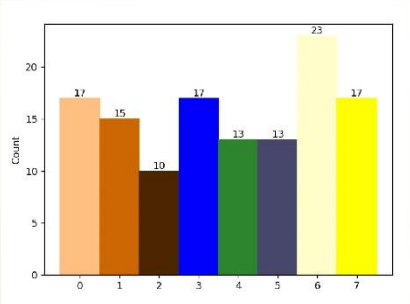
Генерація масиву випадковим чином

```
Введіть ширину землі: 5
Введіть довжину землі: 5
Введіть глибину землі (кількість шарів ґрунту): 5
Введіть ключ випадкових чисел: 5
Random land generated
Землю згенеровано:
Розмір землі = (5, 5, 5)
```

Найкраща послідовність



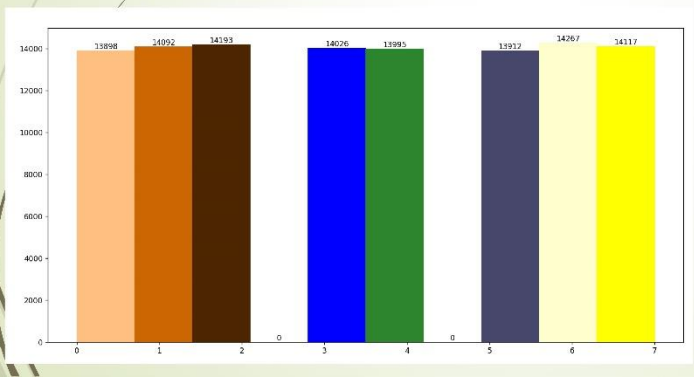
Загальна кількість копалин в згенерованій землі

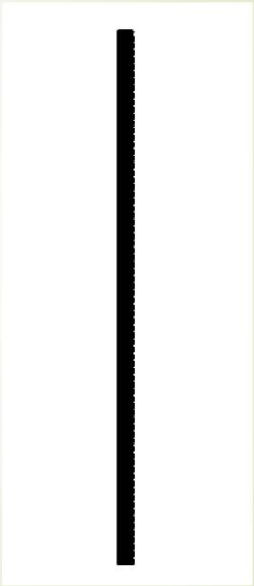


16

Виконання програми для масивів великого розміру

```
D:\programming\Python\projects\labs\lab5\venv\Scripts
Введіть ширину землі: 30
Введіть довжину землі: 50
Введіть глибину землі (кількість шарів ґрунту): 75
Введіть ключ випадкових чисел: 13
Random land generated
Землю згенеровано:
Розмір землі = (75, 50, 30)
```



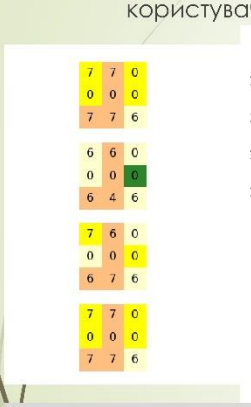


¯_(ツ)_/¯

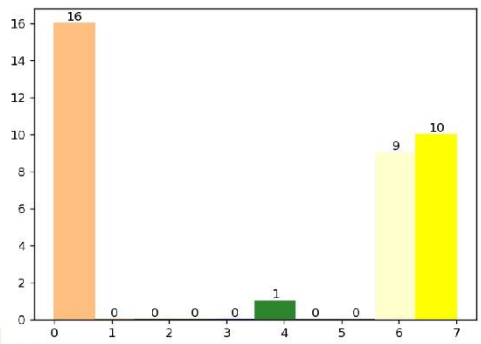
17

Поведінка в особливих ситуаціях:

При відсутності потрібного ресурсу нічого не виводиться на екран користувачу



Згенерована земля



Кількість копалин в землі

Шуканий ресурс – Вода(3).

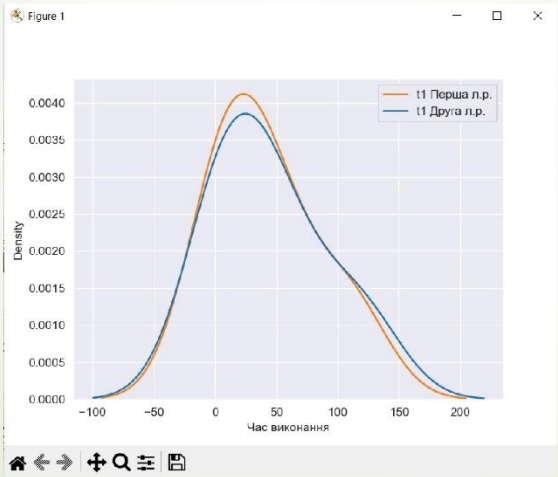
Оскільки даний ресурс відсутній в землі то користувачу нічого не відобразиться.

В консолі можемо побачити виконання тесту для даної ситуації: назва шуканого елемента, найкращу послідовність (пустий список), повідомлення про успішне завершення роботи

```
Best Sequence with WATER
[]
test_no_resource_in_land SUCCESS!
```

18

Ефективність алгоритму на базі Numpy у порівнянні з алгоритмом з першої лабораторної роботи




19
Дякую за увагу!

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна література

1. Черняк О. І. Інтелектуальний аналіз даних: підручник. Київ: Знання, 2014. 599 с.

2. Марченко О. О., Россада Т. В. Актуальні проблеми Data Mining: навчальний посібник для студентів факультету комп'ютерних наук та кібернетики. Київ, 2017. 150 с.

3. Васильєв О. Програмування мовою Python. Видавництво навчальна книга «Богдан», 2019. 504 с.

4. Бахрушин В. Є. Методи аналізу даних: навчальний посібник для студентів. Запоріжжя: КПУ, 2011. 268 с.

5. Олійник А. О., Субботін С. О., Олійник О. О. Інтелектуальний аналіз даних: навчальний посібник, Запоріжжя: ЗНТУ, 2012. 278 с.

Допоміжна література

1. Pal Christopher, Hall Mark, Frank Eibe, Witten Ian. Data Mining: Practical Machine Learning Tools and Techniques, 4rd ed. / M. Kaufmann, 2016.

2. Reese Jennifer, Reese Richard. Java for Data Science / Packt Publishing, 2017.

3. Bell Jason. Machine Learning: Hands-On for Developers and Technical Professionals / John Wiley & Sons, 2014.

4. Маттес Е. Пришвидшений курс Python. Львів: Видавництво Старого Лева, 2021. 600 с.

Інформаційні ресурси в мережі Інтернет

1. Чому вам слід використовувати NumPy. URL: https://devzone.org.ua/post/comu-vam-slid-vikoristovuvati-numpy

2. 10 корисних порад обробки даних у Pandas. URL: https://dou.ua/forums/topic/42873/

3. Stanford University Data Mining Lecture Notes. 2020. URL: http://infolab.stanford.edu/~ullman/mining/2003.html.

4. Matplotlib: Tutorials. URL: https://matplotlib.org/stable/tutorials/index.html

Навчальне видання

Бабаков Роман Маркович

ТЕХНОЛОГІЇ ЗБЕРІГАННЯ, ОБРОБКИ ТА ВІЗУАЛІЗАЦІЇ ДАНИХ

Методичні вказівки до виконання лабораторних робіт
для здобувачів вищої освіти ОС «Магістр» денної форми здобуття освіти
спеціальності F3 Комп'ютерні науки

Редактор О. А. Солдатова
Технічний редактор О. К. Гомон

Підписано до друку 20.11.2025.
Формат 60×84/16. Папір офсетний.
Друк – цифровий. Умовн. друк. арк. 2,38.
Тираж 30. Зам. 161.

Донецький національний університет імені Василя Стуса
21021, м. Вінниця, 600-річчя, 21.
Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру
серія ДК № 5945 від 15.01.2018