МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ
до проведення практичних занять
із навчальної дисципліни

«ПРИКЛАДНІ ЗАДАЧІ ДИСКРЕТНОГО АНАЛІЗУ»

(для здобувачів першого (бакалаврського) рівня вищої освіти всіх форм навчання зі спеціальностей 122, F3 – Комп'ютерні науки, 126, F6 – Інформаційні системи та технології)

Харків
ХНУМГ ім. О. М. Бекетова
2025

Методичні рекомендації до проведення практичних занять із навчальної дисципліни «Прикладні задачі дискретного аналізу» (для здобувачів першого (бакалаврського) рівня вищої освіти всіх форм навчання зі спеціальностей 122, F3 – Комп’ютерні науки, 126, F6 – Інформаційні системи та технології) / Харків. нац.ун-т міськ. госп-ва ім. О. М. Бекетова ; уклад. М. В. Новожилова. – Харків : ХНУМГ ім. О. М. Бекетова, 2025. – 57 с.

Укладач д-р фіз.-мат. наук, проф. М. В. Новожилова

Рецензент

Н. Д. Сізова, доктор фізико-математичних наук, професор, професор кафедри комп’ютерних наук та інформаційних технологій Харківського національного університету міського господарства імені О. М. Бекетова

Рекомендовано кафедрою комп’ютерних наук та інформаційних технологій, протокол № 19 від 30.05.2025.

ЗМІСТ

ВСТУП

Метою викладання дисципліни «Прикладні задачі дискретного аналізу» є засвоєння студентами знань про моделі та методи дискретного аналізу та комбінаторної оптимізації на прикладах реальних ситуацій предметної області з використанням сучасних інформаційних середовищ проєктування, які можуть безпосередньо використовуватися в галузі комп'ютерних наук та інформаційних технологій.

Вивчення дисципліни «Прикладні задачі дискретного аналізу» безпосередньо спирається на дисципліни «Оптимізаційні методи та моделі», «Програмування».

У результаті вивчення дисципліни здобувач першого (бакалаврського) рівня вищої освіти за спеціальностями F3 – Комп'ютерні науки та F6 – Інформаційні системи та технології знатиме технології моделювання дискретного аналізу під час розв'язання задач галузі комп'ютерних наук, отримає навички застосування знання фундаментальних і природничих наук до розв'язання задач дискретного аналізу та програмування щодо складання та реалізації алгоритмів розв'язання задач дискретного аналізу, навчиться оцінювати складність та ефективність алгоритмів та способів передачі інформації, набуде практичних навичок застосування математичного та алгоритмічного апарату дискретного аналізу для моделювання та розв'язання задач інженерної та інформаційної інфраструктури міста.

Ці методичні рекомендації містять опис практичних робіт за двома змістовими модулями та шістьма темами дисципліни, у яких розглядаються такі питання.

Змістовий модуль 1 Математичний апарат дискретного аналізу (Теми 1–3).

Дискретний аналіз – математична дисципліна, у якій досліджуються явища та процеси, що мають дискретний характер, або такі, що можуть бути приведені до дискретного виду для спрощення обчислень. Складовими дискретного аналізу є теорія множин, математична логіка, комбінаторний аналіз, теорія графів, комбінаторика.

Змістовий модуль 2 Задачі дискретної оптимізації: постановка, властивості, обчислювальна складність (Теми 4–6).

Задачі дискретної оптимізації – це задачі знаходження екстремуму функції заданої на дискретній множині точок. Предметна галузь дискретної оптимізації – це дослідження структури і властивостей різних класів NP-

складних задач, розробки та обґрунтування методів їхнього розв'язання, дослідження ефективності запропонованих методів, створення відповідного програмного забезпечення, що дає можливість будувати нові інформаційні технології та розв'язувати нові прикладні задачі. Теорія графів – невід'ємна складова дискретного аналізу як математичного фундаменту дискретної оптимізації.

Зміст практичних робіт включає завдання, теоретичний матеріал та приклади виконання практичних робіт, а також перелік використаних та додаткових джерел.

Методичні рекомендації містять опис програмних застосунків, створених мовою високого рівня Python у хмарному сервісі Google Colaboratory (https://colab.research.google.com/), що дозволяє користувачеві Google писати та запускати вихідний код у своєму браузері. Зокрема, Google Colaboratory підтримує мову програмування Python, яка має множину корисних бібліотек, орієнтованих на завдання машинного навчання, аналіз даних тощо.

Процедура оцінювання практичних робіт здійснюється за трьома критеріями:

- критерій 1. Знання теоретично обґрунтованої методики реалізації завдання практичної роботи. Кількість балів: 1–2;
- критерій 2. Наявність програмної реалізації. Кількість балів: 1–2;
- критерій 3. Наявність звіту з практичної роботи / вміння усно презентувати результати практичної роботи. Кількість балів: 1.

Автор виражає подяку студентам груп ІСТ 2023-1 Дяченку Богдану та Михайленку Миколі за допомогу під час складання прикладів: практичні роботи «Операції над множинами» та «Алгоритм найкоротшого шляху Дейкстри» відповідно.

1 GOOGLE COLABORATORY – ХМАРНЕ СЕРЕДОВИЩЕ ПРОГРАМУВАННЯ МОВОЮ PYTHON

1.1 Введення у Google Colaboratory

Google Colaboratory або Colab (https://colab.research.google.com/) – це безкоштовний хмарний сервіс від Google Research, заснований на Jupyter Notebook (http://www.jupyter.org), який дозволяє будь-якому користувачеві з обліковим записом gmail писати та запускати вихідний код у своєму браузері.

Зокрема, Google Colaboratory підтримує мову програмування Python, яка має множину корисних бібліотек, орієнтованих на завдання машинного навчання, аналіз даних тощо.

Google Colaboratory – це інтегроване середовище розробки (IDE – Integrated Development Environment), що містить редактор початкового коду, а також інструменти для автоматизації складання та тестування програм.

Google Colaboratory не вимагає налаштування. Після виклику Google Colaboratory в будь-якому браузері користувач отримує доступ до так званого блокнота (notebook) Google Colaboratory, який є браузерним інтерфейсом для створення, налагодження та запуску python-скриптів.

Необхідно зазначити, що коли користувач отримує доступ до Colab за допомогою свого облікового запису gmail, то одночасно реалізується віртуальна машина, на якій можна запускати свій код, ізольований від інших користувачів і ресурсів.

Отже, можна відновити віртуальну машину до початкового стану у разі виникнення проблем. Однак після закриття браузера віртуальні машини будуть видалені після певного періоду бездіяльності, щоб звільнити ресурси. Тому потрібно зберігати свої блокноти з початковим кодом у Google Drive, щоб потім завантажити їх локально (формат Jupyter з відкритим кодом .ipynb). Таким чином, база Google Colaboratory може розміщуватися на Google Drive, причому у сховищі зберігаються тільки файли користувача, немає необхідності завантажувати додаткові бібліотеки, що суттєво оптимізує використання наявні ресурси. Крім того, Google відключає файли блокнота після приблизно 30 хвилин бездіяльності, щоб не перевантажувати надані ресурси.

Головна особливість та перевага Google Colaboratory – безкоштовні потужні графічні процесори GPU та TPU, завдяки яким можна займатися не лише базовою аналітикою даних, а й складнішими дослідженнями в галузі

машинного навчання. З тим, що CPU обчислює годинами, GPU або TPU справляються за хвилини або секунди.

Нагадаємо, що CPU – це центральний процесор комп'ютера, який виконує операції з файлами. Він є універсальним і може використовуватися під час виконання майже всіх завдань, від запису файлів на визначені носії або завантаження у хмару, до моделювання фізичних процесів.

GPU – це графічний процесор. Обробляє файли швидше, оскільки завдання виконує паралельно, а не послідовно, як CPU. Він створений винятково для завдань з обробки графічної інформації, тому на ньому зручніше працювати із зображенням та відео, наприклад займатися 3D-моделюванням.

TPU – це тензорний процесор, розробка Google. Він призначений для тренування нейромереж. У цього процесора у рази вища продуктивність при великих обсягах обчислювальних завдань.

Самі процесори дорогі, і не кожен користувач може їх собі дозволити. Платформа Google Colaboratory дає можливість безкоштовно та безперервно користуватися ними протягом 12 годин. Потім Goolge ненадовго може обмежити доступ до GPU, щоб надати можливість використовувати процесори іншим користувачам.

1.2 Функції Google Colaboratory

Google Colaboratory є дружнім, інтуїтивно зрозумілим та простим у використанні середовищем розробки програмного забезпечення.

Він містить довідник із документацією, а також кілька прикладів, щоб почати робити перші кроки, змінити вже створені коди та перейти до тестування.

Найвідомішими функціями та перевагами Google Colaboratory є:

1. Редагування та запуск коду мови Python та можливість працювати з Python-бібліотеками для аналізу даних онлайн.
2. Збереження програмних проєктів на Google Drive (GDrive).
3. Завантаження програмного коду з GitHub.
4. Можливість віддаленої сумісної роботи кількох користувачів у блокнотах з текстом, кодом, результатами та коментарями.
5. Імпорт блокнотів Jupyter або IPython.
6. Завантаження будь-яких блокнотів Colab локально з GDrive.
7. Потужні процесори для хмарних обчислень.

8. Інтуїтивно зрозумілий інтерфейс, який дозволяє не перевантажувати комп'ютер файлами та процесами та робити всі обчислення швидко.

9. Доступ до облікового запису з файлами з будь-яких пристроїв.

1.3 Починаємо працювати з Google Colaboratory

Виклик https://colab.research.google.com/ у браузері ініціює появу екрану із доступними блокнотами (рис. 1.1).

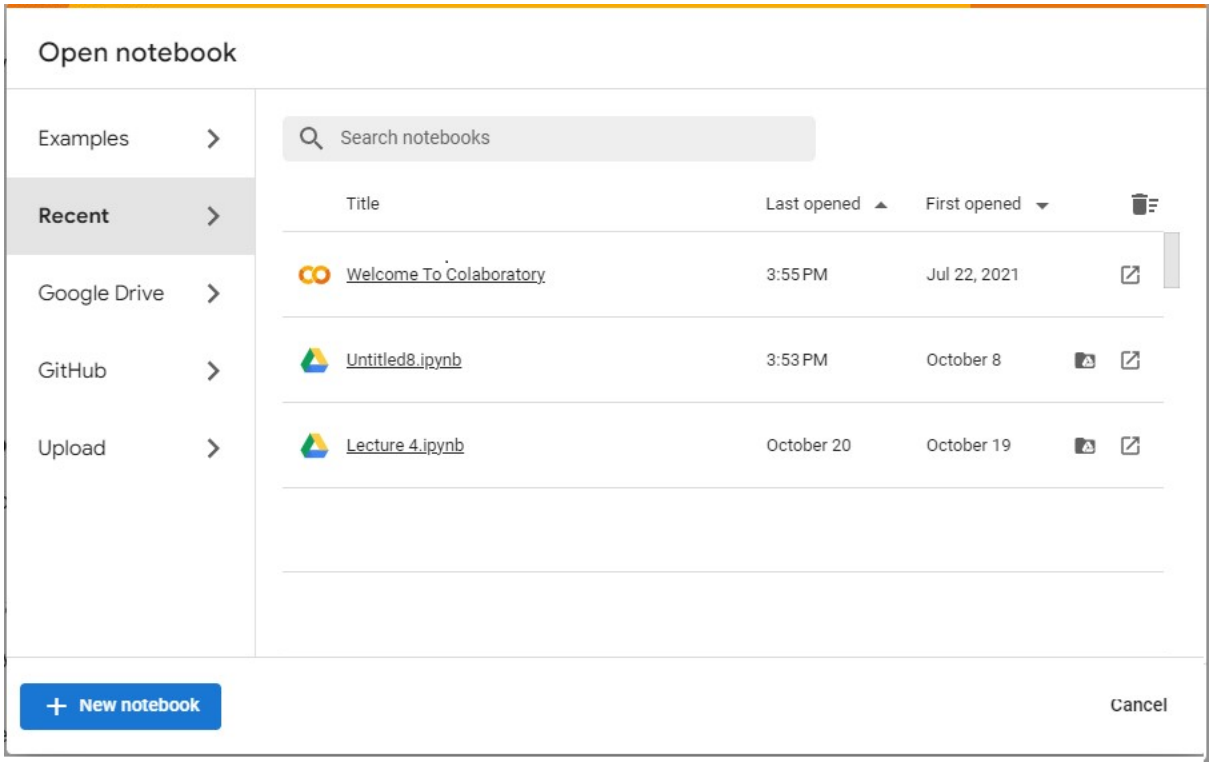


Рисунок 1.1 – Головна сторінка блокнота Google Colab

Можна створювати новий (кнопка у лівому нижньому куті екрана) або завантажувати вже розроблений Python код з Google Диску.

Вид робочого поля блокнота Google Colab показаний на рисунку 1.2.

Перший рядок зверху – назва блокнота. Назву можна легко змінити прямо на робочому полі.

Аркуш ноутбука загалом розділений на чотири частини, на рисунку 1.2 можна побачити три з них:

- верхнє меню;
- меню, що випливає ліворуч;
- робоче поле.

Розглянемо ці складові більш детально.

Верхнє меню містить перелік команд налаштування сеансу у середовищі Google Colab.

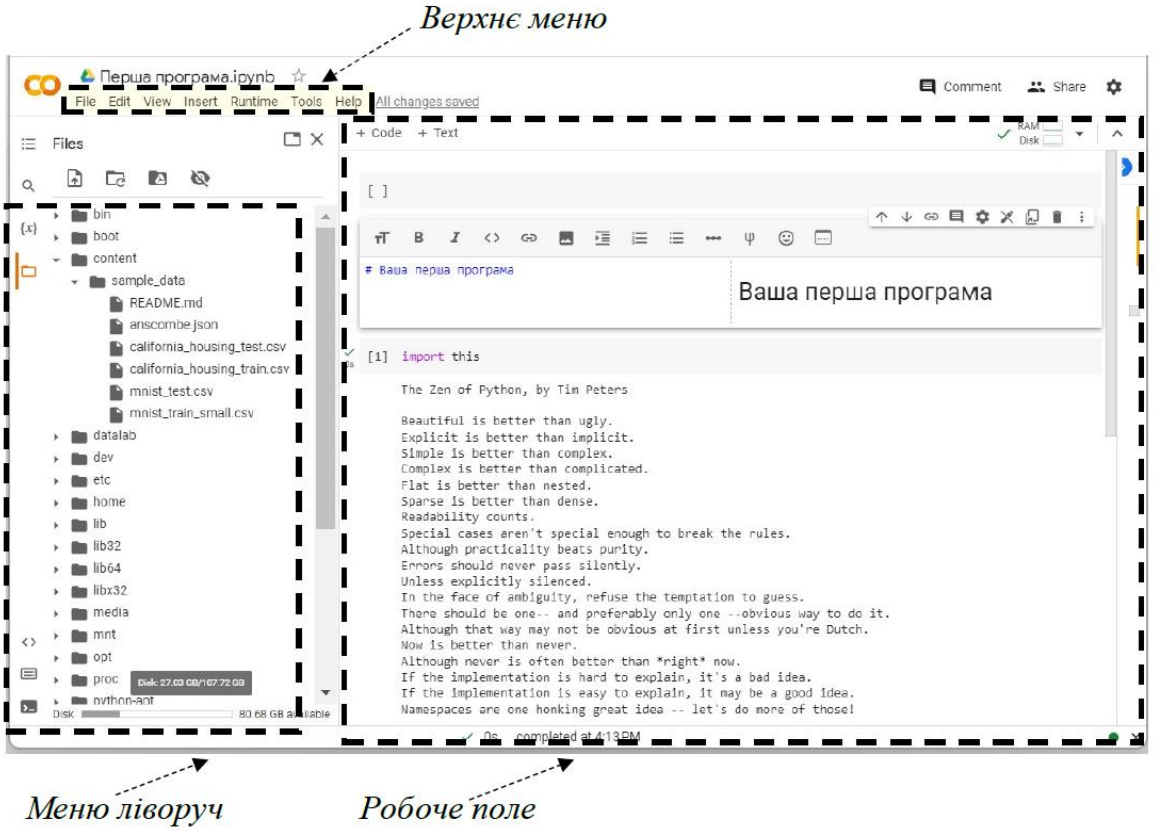


Рисунок 1.2 – Робоче поле блокнота Google Colab

Меню «File» (рис. 1.3), крім інтуїтивно зрозумілих пунктів, містить різноманітні можливості збереження файлів.

Ще одна перевага Colab – це інтеграція з GitHub. Він відкриває доступ до будь-якого сховища, якщо йому надати профіль на сервісі. За це відповідає команда «Save a copy in GitHub».

Щоб завантажити певний блокнот із GitHub, необхідно додати

http://colab.research.google.com/github/

до шляху блокнота у GitHub.

Наприклад (приклад взятий з сайту довідки Colab), щоб завантажити блокнот

https://github.com/tensorflow/docs/blob/master/site/en/tutorials/_index.ipynb

у Colab, необхідно скористатися такою URL-адресою:

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/_index.ipynb

Команда «Download» дозволяє завантажити ноутбук на персональний комп'ютер користувача.

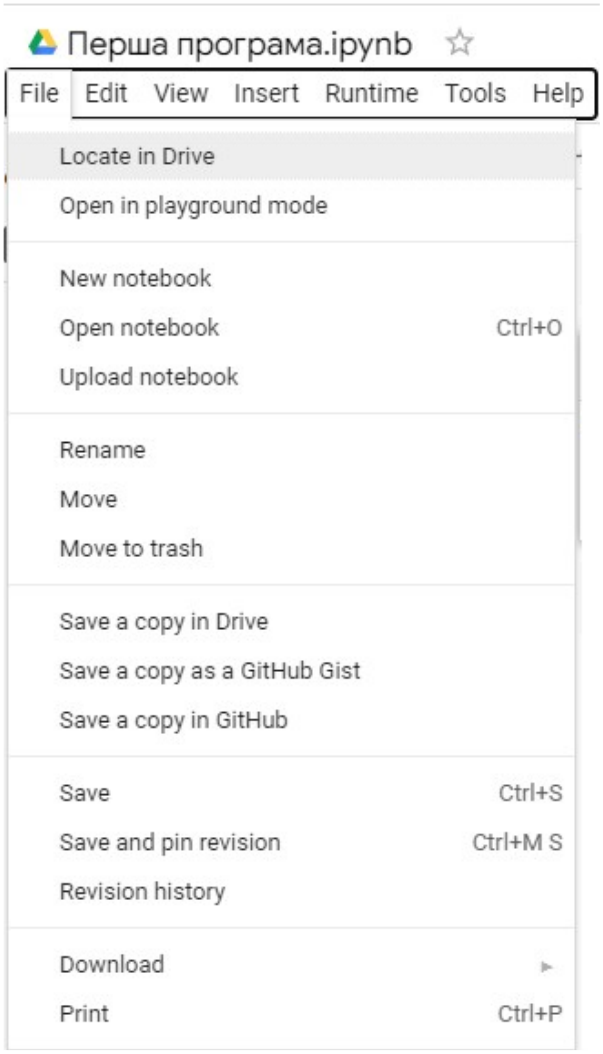


Рисунок 1.3 – Вкладка «File»

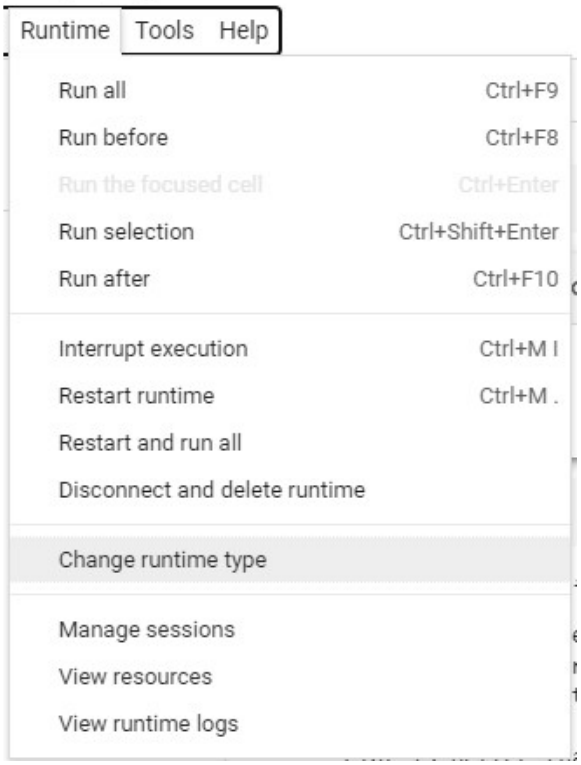


Рисунок 1.4 – Вкладка «Runtime»

Крім того, для певних завдань, наприклад, для роботи з Big Data, у Google Colab можна вибрати відповідний за потужністю процесор.

Для цього необхідно змінити середовище виконання у вкладці «Change runtime type» (рис. 1.4) і у налаштуваннях блокнота зробити вибір між GPU і TPU.

Меню ліворуч містить декілька важливих елементів.

Елемент з позначкою 2.1 на рисунку 1.5 – це вкладка «Files», ініціація якої дозволяє розгорнути перелік та структуру директорій, що формує Google

для організації роботи користувача. На рисунку 1.5 на вкладці ліворуч показана ця структура.

Зазначимо, що файли, створені користувачем, зберігаються у теці /content, що є кореневою папкою Google Colab і має бути доданою до всіх шляхів, що використовуються в блокноті.

Крім того, тека /content містить заздалегідь створену директорію sample_data, яка включає набір файлів різних типів, корисних для початкового знайомства з можливостями мови та бібліотек Python щодо обробки та аналізу даних.

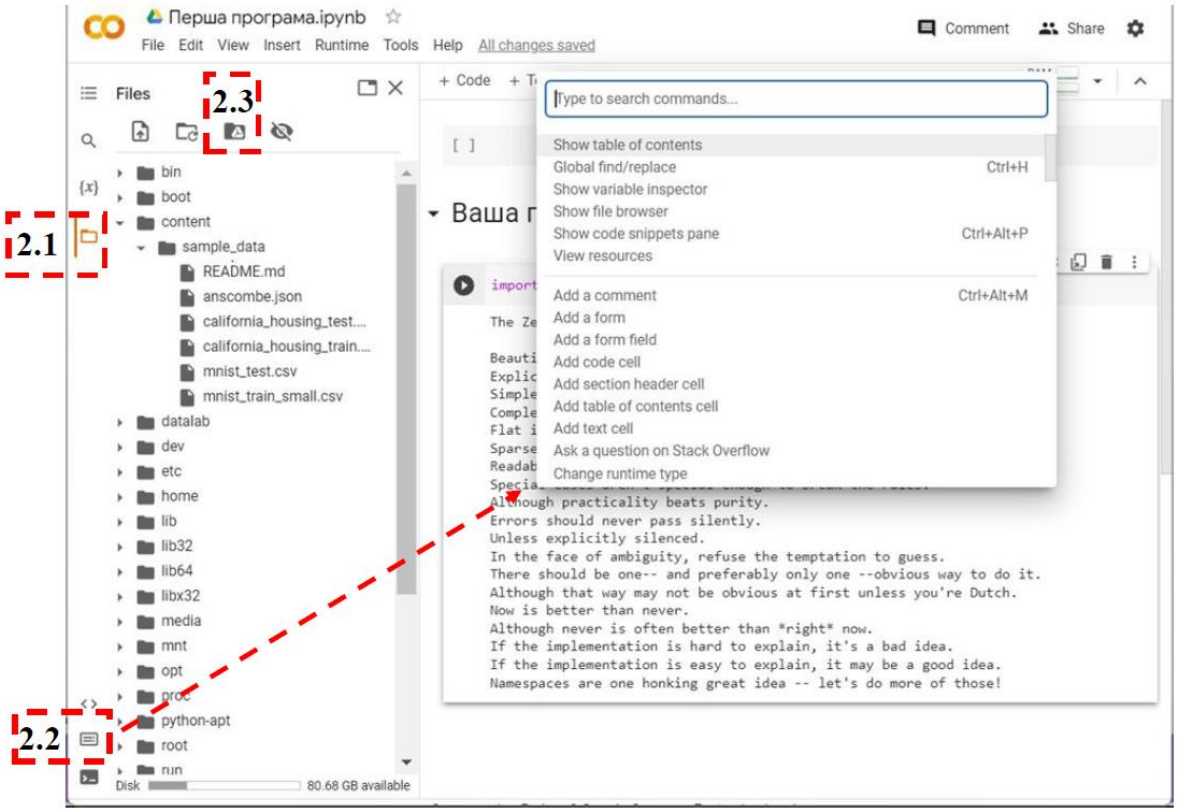


Рисунок 1.5 – Вигляд меню ліворуч середовища Google Colaboratory

Елемент з позначкою 2.2 на рисунку 1.5 – це перелік всіх можливих команд в середовищі Google Colaboratory. Щоб працювати з файлами та кодом Google Colab з особистого диска, потрібно використовувати команду «Mount Drive» – елемент з позначкою 2.3 на рисунку 1.5.

Для перевірки підключення Google-диску користувача можна застосувати команду

!ls "/content/drive/MyDrive",

яку потрібно набрати на робочому полі середовища Google Colaboratory – третя частина на рисунку 1.2. Ця команда покаже вміст Google-диска користувача.

Google Colab максимально спростив усі процеси програмування мовою Python: у ньому є і базові бібліотеки (NumPy, scikit-learn, Pandas), і складніші (Keras, TensorFlow або PyTorch), не потрібно ставити програми та середовища самостійно, можна просто одразу писати код.

Якщо ж базових бібліотек Google Colab недостатньо, завжди можна додати необхідні за допомогою інсталятора PIP, наприклад:

```
%pip install emoji
```

Останнім часом Google випускає багато прикладів коду машинного навчання у формі блокнотів Colab. Таким є середовище Seedbank (https://www.seedbankuk.com/) – репозиторій програм-прикладів інтерактивного машинного навчання, які можна запускати у своєму браузері без налаштувань. Кожен приклад можна сприймати як маленьке зерно, яке можна редагувати, розширювати та розвивати власні проєкти та ідеї, від проблем аналізу даних до мистецьких проєктів. Серед прикладів – демонстрація роботи та навчальні посібники для tensorflow.org, прискорений курс машинного навчання, дослідницькі статті на distill.pub та багато іншого.

У Colab можна ділитися файлами з іншими, залишати коментарі, редакторські нотатки і загалом робити все, що є в тих же Google Документах.

Тому при загальному доступі до блокнота весь його вміст є доступним іншим користувачам (текст, код, коментарі, вихідні дані). Останнє можна вимкнути: потрібно вибрати пункт «Notebook settings» в меню «Edit» (рис. 1.6). У вікні Google Colab, що з’явиться, поставити галочку «Omit code cell output when saving this notebook», і тоді в блокноті збережеться тільки код, але не результати його виконання.

Робоче поле інтерактивного середовища Colab – це місце блокнота Colab, де дозволяється писати та виконувати код. Робоче поле складається з клітинок, які можуть бути клітинками коду або текстовими клітинками.

Наприклад, на рисунку 1.2. напис «Ваша перша програма» – виконаний у текстовій клітинці, а завантаження основних правил програмування мовою Python – так званий дзен Python – виконано у клітинці коду.

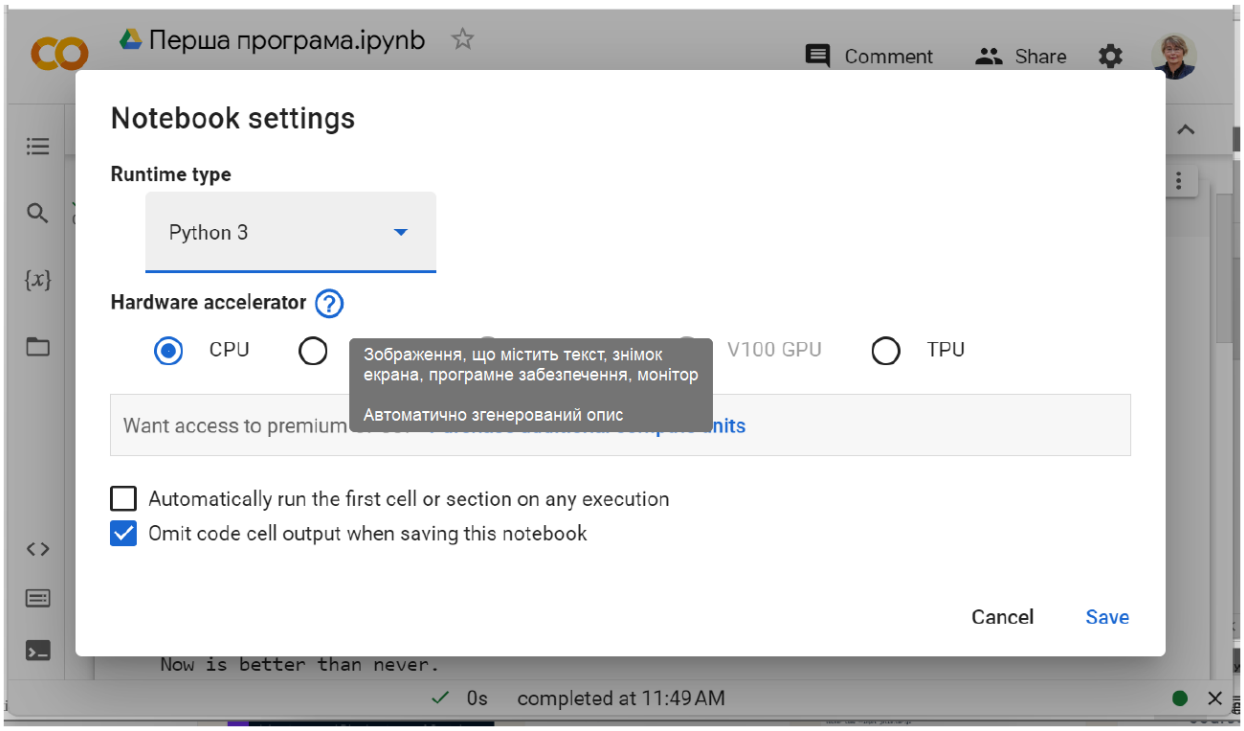


Рисунок 1.6 – Встановлення режиму збереження блокнота Colab

Щоб виконати код у клітинці, виберіть її, а потім натисніть кнопку відтворення ліворуч від коду. Щоб змінити код, достатньо натиснути на комірку.

Змінні, задані в одній клітинці, можна використовувати в інших клітинках нижче. Таким чином, у блокнотах Colab можна використовувати в одному документі код, форматований текст, зображення, розмітку HTML, набір LaTeX і не тільки.

Блокноти Colab зберігатимуться Google Диску користувача. Власник програмного коду може відкрити доступ до нього партнерам або колегам, дозволивши їм переглядати або навіть редагувати документ, а також залишати коментарі. Це можна зробити за допомогою засобу «Share» у верхньому правому кутку інтерфейсу блокнота Colab.

І кілька слів про Дзен Пайтона – The Zen of Python (рис. 1.2). Розробники мови Python дотримуються певної філософії програмування, яка має назву «The Zen of Python». Її текст видається інтерпретатором Python за командою

```
import this
```

яка працює один раз за сесію.

Загалом ця сукупність правил, що складена Тімом Пітерсом, підходить до програмування будь-якою мовою.

2 ПРАКТИЧНА РОБОТА «ОПЕРАЦІЇ НАД МНОЖИНАМИ»

2.1 Завдання на практичну роботу

Практична робота містить чотири завдання.

Кожному студенту необхідно вибрати по одному із кожних перших трьох типів завдань (згідно з номером у журналі групи).

Зауваження 2.1. Третє завдання орієнтовано на роботу в команді виконавців та визначення ролей розробників.

Четверте завдання – графічне подання діаграми Венна з третього завдання за допомогою модулів venn2, venn2_circles, venn3, venn3_circles бібліотеки matplotlib мови програмування Python.

Зауваження 2.2. Друге та четверте завдання супроводжуються прикладом програмного застосунку мовою Python.

За результатами виконаних завдань створити звіт та викласти у відповідний дистанційний курс Moodle.

Завдання 1

Необхідно відповісти на одне із запитань згідно з номером варіанта, а також на ще одне питання за вибором студента.

1. Назвіть відомі Вам способи завдання множин. У якому випадку не можна застосувати той або інший спосіб?
2. Чи можуть два елементи однієї множини бути однаковими?
3. Визначте поняття підмножин та включення множин?
4. Чим відрізняється строге включення від нестрогого. Наведіть приклад.
5. Яка множина називається порожньою?
6. Що таке потужність множини?
7. Яка множина називається універсальною?
8. Дайте визначення операціям на множинах?
9. Під час виконання якої операції використовується універсальна множина?
10. Чи можуть деякі з операцій бути виражені через інші? У який спосіб?
11. Як в алгебрі Кантора виконується операція об’єднання, якщо множини задані двійковим представленням. Навести приклад.

12. Як в алгебрі Кантора виконується операція перетину, якщо множини задані двійковим представленням. Навести приклад.

13. Як в алгебрі Кантора виконується операція симетричної різниці, якщо множини задані двійковим представленням. Навести приклад.

14. Як в алгебрі Кантора виконується операція доповнення, якщо множини задані двійковим представленням. Навести приклад.

15. Надайте поняття зліченості та незліченості множин.

16. Класифікація числових множин.

17. Дискретні та неперервні множини. Приклади.

18. Визначення та приклад операції симетричної різниці.

19. Декартовий добуток дискретних множин. Приклади.

20. Декартовий добуток неперервних множин. Приклади.

21. Надайте приклад операції об'єднання числових множин.

22. Що таке породжувальна процедура?

23. Як задається різниця множин? Приклади.

24. Що таке круги Ейлера. Приклади.

25. Що таке діаграма Венна. Приклади.

26. Декартовий добуток трьох числових дискретних множин. Приклад.

27. Чи можуть числові множини містити від'ємні числа?

28. Знайдіть перетин множини дійсних чисел та множини комплексних чисел та поясніть результат.

29. Які характеристики елементів множин враховуються при визначенні теоретико-множинних операцій?

30. Що таке булеан?

Завдання 2

Нехай є заданими дві множини: множина A та множина B, а також тип теоретико-множинної операції:

$$A \cup B,\ A \cap B\ ,\ A / B\ ,\ A \Delta B,\ B / A\ ,\ A \times B.$$

Необхідно написати програму мовою Python в хмарному середовищі Google Colab для реалізації заданої теоретико-множинної операції.

Під час виконання цього завдання вітається робота в команді з двох студентів з визначеними інтелектуальними ролями: алгоритміст, розробник, тестувальник тощо.

Тип операції, вихідні множини A та B задані варіантами в таблиці 2.1.

Таблиця 2.1 – Варіанти завдання 2

Ч. ч.	Операція	A ={x}	B
1	A ∪ B	{x∈A| x∈N, ділиться на 4 і x ≤ 40}	{x∈B | x∈N, ділиться на 5 і x ≤ 60}
2	B / A	{x∈A| x∈N, x – парні, x ≤ 20}	{ x∈B | x∈Z, ділиться на 10 і |x| ≤ 40
3	A × B	{“троянда”, “айстра”, “лілія”}	{“біла”, “червона”, “синя”}
4	A ∩ B	[2, 5]	[3,7]
5	A ∪ B	{a, b, c, e, d}	{a, e, l, k, n}
6	A ∩ B	[0, 2, 7, 10, 11]	[-3, 2, 9, 10, 13]
7	B / A	[3, 5]	[2, 4]
8	A × B	{1, 5, 3, 0, 7}	{2, 4, 0, 6}
9	A / B	{1, 2, 3, 4}	{2, 4, 6, 8}
10	A ∩ B	{x∈A| x ∈ R, – 2 ≤ x ≤ 2}	{ x∈B| x ∈ R, – 1 ≤ x ≤ 1}
11	A ∩ B	{“рік”, “місяць”, “тиждень”, “день”}	{“хвилина”, “година”, “день”, “тиждень”}
12	A Δ B	{a, c, b}	{c, d, k}
13	A ∪ B	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}	{x∈N, x – парне і x ≤ 11}
14	A ∩ B	{x∈A| x – просте число і x ≤ 15}	{x ∈ B| x∈N і x ≤ 50}
15	A / B	{2, 3, 7, 8, 9, 11, 19, 20}	{x∈B| x∈N і ділиться на 3 і x ≤ 100}
16	A ∪ B	[3, 5]	[2, 4]
17	B / A	{x∈A|x ∈ N, ділиться на 7 і x ≤ 30}	{x∈B| x∈N і поділяється на 3 або 5 і x ≤ 50}
18	A ∩ B	{x∈A|x ∈ N, x – непарне і x ≤ 15}	{x∈B| x∈N і x ≤ 50}
19	A Δ B	[3, 5]	[2, 4]
20	A ∩ B	{a, b, c, e, d}	{a, e, l, k, n}

Приклад програми визначення декартового добутку двох множин (автор – Дяченко Богдан, група ІСтаТ 2023-1) подано на рисунку 2.1.

Програму написано мовою Python в середовищі Google Colaboratory.

```python
#Дяченко Богдан, група ІСтаТ-2023-1
#Програма обчислення декартового добутку двох множин

from itertools import product

# Функція обчислення декартового добутку двох множин
def cartesian_product(A, B):
    return list(product(A, B))

def main():
    # Зчитуємо множини A і B
    A = set(map(str, input("Введіть елементи множини A (через пробіл): ").split()))
    B = set(map(str, input("Введіть елементи множини B (через пробіл): ").split()))

    # Обчислюємо декартовий добуток множин A і B - множина result
    result = cartesian_product(A, B)

    # Виводимо результат
    print("Декартовий добуток множин A і B:")
    for pair in result:
        print('element',pair)
    print('set',set(result))

if __name__ == "__main__":
    main()
```

Рисунок 2.1 – Вихідний код програми визначення декартового добутку двох множин

Результат дії програми визначення декартового добутку двох множин показаний на рисунку 2.2.

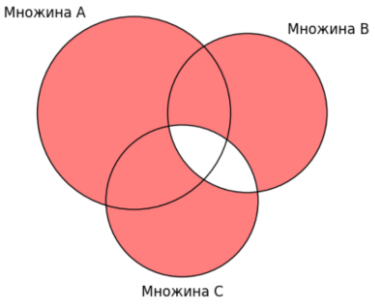
```
Введіть елементи множини A (через пробіл): 2 3 4
Введіть елементи множини B (через пробіл): f d c
Декартовий добуток множин A і B:
element ('2', 'd')
element ('2', 'c')
element ('2', 'f')
element ('4', 'd')
element ('4', 'c')
element ('4', 'f')
element ('3', 'd')
element ('3', 'c')
element ('3', 'f')
set {('2', 'f'), ('2', 'd'), ('4', 'd'), ('3', 'c'), ('2', 'c'), ('4', 'c'), ('3', 'f'), ('3', 'd'), ('4', 'f')}
```

Рисунок 2.2 – Результат дії програми визначення декартового добутку двох множин

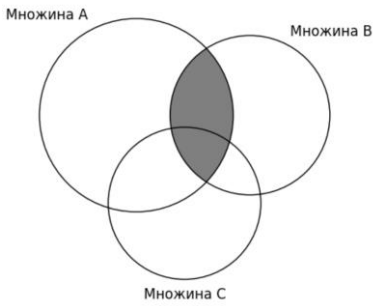
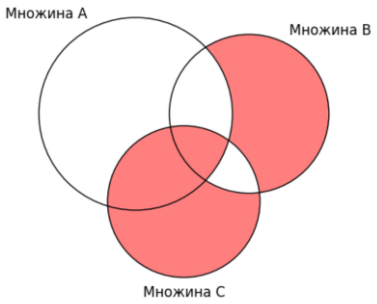
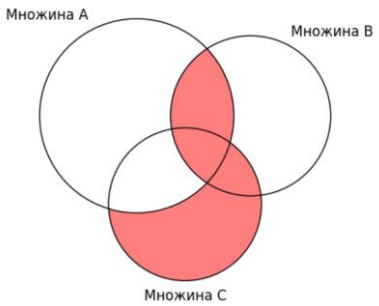
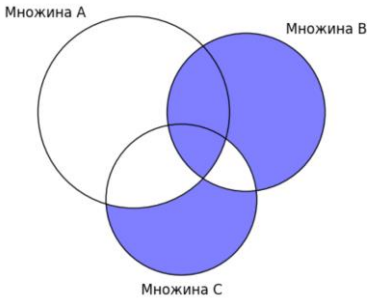
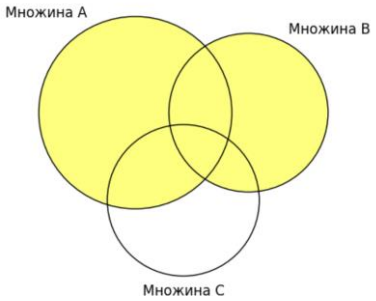
Завдання 3

Розглянути надані діаграми Венна та, застосовуючи теоретико-множинні операції { ∪ , ∩ , / , Δ }, надати аналітичний опис множини, яка відповідає кольоровій частині діаграми (табл. 2.2).

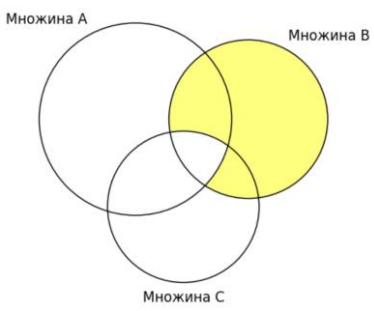
Таблиця 2.2 – Варіанти завдання 3

Ч. ч.	Завдання	Ч. ч.	Завдання
1	2	3	4
1		2	
3		4	
5		6	
7		8	

Продовження таблиці 2.2

1	2	3	4
9		10	
11		12	
13		14	
15		16	
17		18	

Закінчення таблиці 2.2

1	2	3	4
19		20	

Завдання 4

Створити програму мовою програмування Python з використанням модулів venn2, venn2_circles, venn3, venn3_circles бібліотеки matplotlib – щодо графічного подання діаграми Венна з третього завдання за обраним варіантом.

2.2 Побудова діаграм Венна за допомогою бібліотек Python

Розглянемо теоретичні нотатки та приклад виконання завдання 4 – побудови діаграм Венна за допомогою бібліотек Python. Нижче наводиться діаграма Венна (рис. 2.3), що побудована за допомогою програми, текст якої наведено на рисунку 2.4.

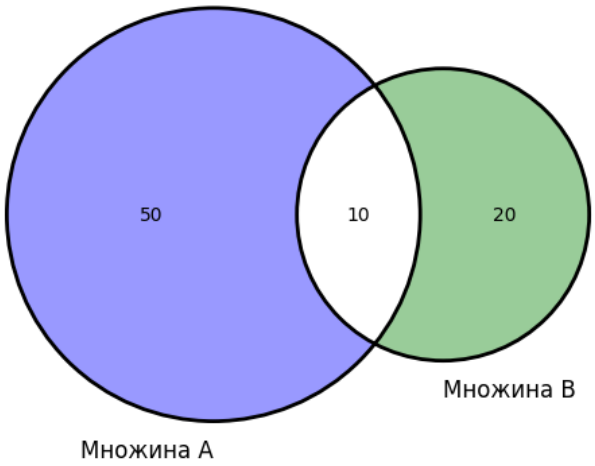


Рисунок 2.3 – Діаграма Венна з двома множинами

```
# Побудова діаграми Венна з 2-ма множинами
# Підключення бібліотек (import modules) з модулю matplotlib та
matplotlib_venn

from matplotlib_venn import venn2, venn2_circles
from matplotlib import pyplot as plt

# Застосовуються наступні домовленості з визначення елементів
# графічного подання діаграми
#    Ab = Належить множині A, але не множині B (50):  10
#    aB = Належить множині B, але не множині A (20): 01
#    AB = Належить обом множинам A та B (10): 11

v=venn2(subsets = (50, 20, 10), set_labels = ('Множина A',
'Множина B'))
v.get_patch_by_id('11').set_color('white')
v.get_patch_by_id('01').set_color('green')
v.get_patch_by_id('10').set_color('blue')

# Стиль околів
venn2_circles(subsets=(50,20,10))
plt.show()
```

Рисунок 2.4 – Python-програма побудови діаграми Венна з двома множинами

Бібіліотека `matplotlib_venn` також дозволяє скласти програму побудови діаграми Венна з трьома множинами (рис. 2.5).

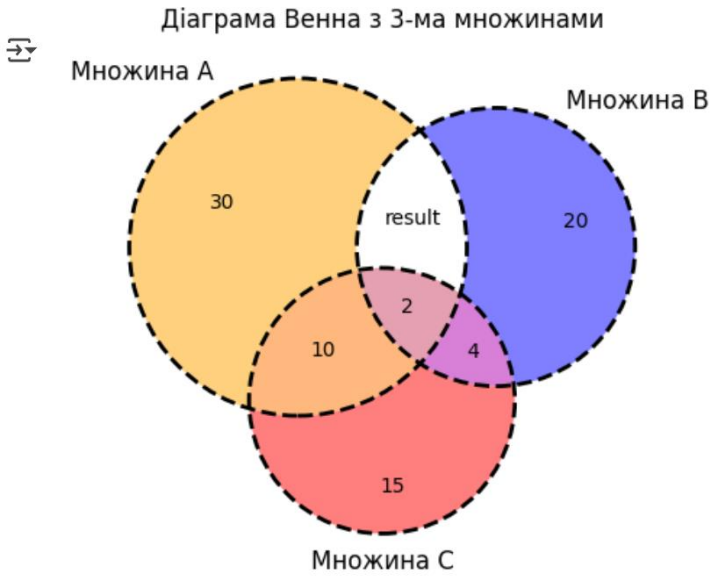


Рисунок 2.5 – Діаграма Венна з трьома множинами: A, B, C

Текст відповідної програми наведено на рисунку 2.6.

```
# Побудова діаграми Венна з 3-ма множинами
# Підключення бібліотек (import modules) з модулю matplotlib та
matplotlib_venn

from matplotlib_venn import venn3, venn3_circles
from matplotlib import pyplot as plt

# Побудова діаграми Венна з 3-ма множинами

v=venn3(subsets=(30, 20, 8, 15, 10, 4, 2),
set_labels=('Множина A', 'Множина B', 'Множина C'),
set_colors=('orange', 'blue', 'red'), alpha=0.5)

# Визначення стилю та товщини околів

venn3_circles(subsets=(30, 20, 8, 15, 10, 4,2),
linestyle= 'dashed', linewidth=2)

# Застосовуються наступні домовленості з визначення елементів
# графічного подання діаграми

# Abc = Належить множині A, але не множинам B та C (30) 100
# aBc = Належить множині B, але не множинам A та C (20) 010
# abC = Належить множині C, але не множинам A та B (20) 001
# ABc = Належить обом множинам A та B, але не множині C (8) 110
# AbC = Належить обом множинам A та C, але не множині B (10) 101
# aBC = Належить обом множинам B та C, але не множині A (4) 101
# ABC = Належить трьом множинам A, B та C (2) 111

#Визначення користувацього стилю підписів та кольору елементів
діаграми

v.get_patch_by_id('110').set_color('white')
v.get_label_by_id('110').set_text('result')

# Назва діаграми Венна

plt.title('Діаграма Венна з 3-ма множинами')
plt.show()
```

Рисунок 2.6 – Python-програма побудови діаграми Венна з трьома множинами

Наведені програми є адаптованою версію вихідного коду, розміщеного за адресою

https://www.google.com/url?q=https%3A%2F%2Fwww.geeksforgeeks.org%2Fhow-to-create-and-customize-venn-diagrams-in-python%2F

Повну інформацію щодо побудови діаграм Венна можна знайти в офіційній документації за посиланням https://pypi.org/project/matplotlib-venn/https://pypi.org/project/matplotlib-venn/.

3 ПРАКТИЧНА РОБОТА «АЛГЕБРА ВИСЛОВЛЮВАНЬ. ПОБУДОВА ФУНКЦІЙ ІСТИННОСТІ»

3.1 Завдання на практичну роботу

Практична робота містить три завдання.

Метою завдання 1 є імплементація теоретичних відомостей щодо понять відношення та відображення множин.

Завдання 2 містить набір практичних питань щодо застосування формул алгебри висловлювань, що супроводжується прикладами виконання.

Завдання 3 містить методичні рекомендації щодо розробки програмного забезпечення побудови функцій істинності формул алгебри висловлювань.

За результатами виконаних завдань необхідно створити звіт та викласти у відповідний дистанційний курс Moodle.

Завдання 1

1. Навести визначення скрізь визначеного відображення. Навести приклади.
2. Навести визначення сюр'єктивного відображення. Навести приклади.
3. Навести визначення ін'єктивного відображення. Навести приклади.
4. Навести визначення бієктивного або взаємно однозначного відображення. Навести приклади.

Приклад 3.1. Нехай множина A = {1,3,5,7}, множина B = {1,2,3,4,5,6,7,8}.

Розглянемо відношення G = { (1,2), (3,4), (5,6), (7,8) }. Відношення G є функціональним, бо образ кожного елемента з A є одноелементною множиною в B, отже, це є відображення, що є скрізь визначеним, не сюр'єктивним і не ін'єктивним.

Приклад 3.2. Прикладом відношення може слугувати словник. Причому, таке відношення частково визначене, не сюр'єктивне і не ін'єктивне. Дійсно, жоден словник не охоплює всієї множини слів мови, при цьому одне слово (наприклад, англійське) може мати кілька значень при перекладі.

Завдання 2

Опрацювати операції над висловленнями.

2.1. Записати по два речення, які є:

– істинними висловленнями;
– хибними висловленнями;
– такі, що не є висловленнями.

2.2. Нехай задано елементарні (прості) висловлення:
– a: 7 – ціле число;
– b: 7 – від'ємне число;
– c: 7 – просте число;
– d: число 7 кратне 3.

Сформулювати словами складене висловлення, визначити його істинність чи хибність для таких формул:
– $a \wedge b$;
– $a \vee b$;
– $a \rightarrow \neg b$;
– $(b \rightarrow \neg a) \vee c$;
– $(\neg a \wedge \neg c) \rightarrow d$.

Приклад виконання. Першій із цих формул: $a \wedge b$ відповідає складене висловлення

7 – ціле число і 7 – від'ємне число

(звичайно таке висловлення формулюється у вигляді 7 – ціле і від'ємне число).

Отримане висловлення є хибним, оскільки перша його елементарна складова a є істинним висловленням, а друга b – хибним. За означенням операції кон'юнкції $\wedge$ значенням цього складеного висловлення є хибно, або 0.

2.3. Визначити істинність чи хибність складеного висловлення, використавши його логічну структуру й виходячи з відомих значень істинності простих висловлень, із яких воно складається.

а) Число 777 кратне 7, але не кратне 11.

б) Число 36 кратне 6 або 7.

в) Якщо $4 < 3$, то $42 < 32$

г) $-2 > -3$ та $(-1/2) > (-1/3)$.

д) $-2 > -3$, але $(-2)^2 < (-3)^2$.

е) Якщо $2 < 3$, то $2^2 < 3^2$.

ж) Принаймні одне із чисел 21, 24 чи 27 парне.

и) Якщо число 24 кратне 6, то число 24 кратне 3.

к) Якщо число 27 кратне 6, то число 27 кратне 3.

л) Якщо число 27 кратне 3, то число 27 кратне 6.

м) Якщо $2 < 3$ та $3 < 1$, то $2 < 1$.

н) Неправильно, що принаймні одне з чисел 35, 57, 77 просте.

Приклад виконання. Вираз а) має логічну структуру $a \wedge b$, де пропозиційній змінній a відповідає елементарне висловлювання «Число 777 кратне 7», а змінній b – висловлювання «Число 777 не кратне 11». Обидва висловлювання істинні, тому й задане складене висловлювання істинне.

2.4. Визначити значення істинності висловлень a, b, c, d, які фігурують у складених висловленнях, якщо складені висловлення
а) та b) – істинні, а c) та d) – хибні.

a) Якщо 7 – просте число, то a.

b) Якщо b, то 7 – складене число.

c) Якщо 7 – просте число, то c.

d) Якщо d, то 7 – складене число.

Приклад виконання. Оскільки перше висловлення істинне та його елементарна складова 7 – просте число є також істинним висловленням, то із правила виконання операції імплікації випливає, що його друга елементарна складова a має бути істинним висловленням.

2.5. Записати словами у вигляді твердження задане висловлювання різними способами, використовуючи вирази: необхідна умова; достатня умова; тоді, коли; тільки тоді, коли.

а) (24 кратне 6) $\rightarrow$ (24 кратне 3).

б) Число 45 кратне 15 тільки тоді, коли 45 кратне 3 і кратне 5.

Приклад виконання. Для пункту а) такими твердженнями будуть:

– висловлення 24 кратне 3 є необхідною умовою для висловлення 24 кратне 6;

– висловлення 24 кратне 6 є достатньою умовою для висловлення 24 кратне 3;

– 24 кратне 3 тоді, коли 24 кратне 6;

– 24 кратне 6 тільки тоді, коли 24 кратне 3.

Зауваження 3.1. У математиці імплікацію p → q трактують так: твердження p є достатньою умовою для q, а твердження q – необхідною умовою для твердження p.

Вираз p → q інтерпретують ще як: q тоді, коли p або p тільки тоді, коли q.

В імплікації p → q операнд p називають **антецедентом**, або засновком, а q – **консеквентом**, або висновком.

2.6. За допомогою відповідних таблиць істинності переконатись у тому, що формули нижче є тавтологіями.

- (p ∨ (¬p)) (закон виключення третього);
- (¬(p ∧ (¬p))) (закон виключення суперечності);
- (p → p) (закон тотожності).

Визначення 2.1. Формулу алгебри висловлень $A(p_1, p_2, \ldots, p_n)$ називають **тавтологією**, коли їй відповідає функція істинності, що тотожно дорівнює 1.

Те, що формула A є тавтологією, позначають так: |=A.

Тавтології ще називають тотожно істинними формулами, або законами алгебри висловлень.

Завдання 3

Розробити Python-програму побудови таблиць функцій істинності формул алгебри висловлювань. Варіанти завдань (згідно з номером у журналі групи) подані в таблиці 3.1.

Таблиця 3.1 – Варіанти завдання 3 (згідно з номером у журналі групи)

Номер варіанта	Формула алгебри висловлювання
1	2
1	((x ∧ (¬ y)) → (y ≡ ((¬ z) → x))) ∨ (¬ x)
2	(¬ x ∧ y) → ¬(z ∨ x)
3	x →(y ∨ ((z →¬ x) ≡¬y))
4	(((x → y) ∧ (y ≡ z)) → (¬x ∨ z))

Продовження таблиці 3.1

1	2
5	¬ ((¬ x ∨ y) ∧ (x ∨ ¬y) → (¬ x ∨ ¬ z))
6	((x → y) → ((x→ (y ∨ z)) →(x ∨ z)))
7	((¬x ∨ y) ∧ ¬ z) ≡ z
8	((¬y ∧ ¬z) ∨ ¬x) ∧ (¬ (z ∨ y) ≡ x)
9	((x→ (y ∨ z)) → ((x ≡ y) →(x → z)))
10	¬ (¬z ∨ (¬y ∧ ¬x) ∧ (¬z ∨ y)
11	(x → z) → (¬ x ∧ y) → (x ∨ y)
12	(x ∧ y) ≡ (y → z)
13	((x →y)→ (z →¬ x)) ∨ ¬ y
14	x →(y ∨ ((z →¬ x) ≡ ¬y)))
15	((x ∨ y) → (x ∧ y))→ z
16	¬ ((¬ x ∨ y) ∧ (x ≡ ¬y) ∧∧ (¬ x ∨ ¬ z))
17	((x → y) ∧ z) ~ ((¬ x) → (y ∨ z));
18	((¬ x) ∧ y) → (x ≡ ((¬ y) ∨ x))
19	(x → z) →((y → z)→ ((x ∨ y) → z))
20	(¬x ∨ ¬ y) ∧ (x ∨ y)

3.2 Теоретичні нотатки до виконання завдання 3

Операції над висловлюваннями визначаються за допомогою функцій істинності, які задаються таблицями істинності.

Функція істинності є функцією, яка повертає істинне значення (1, true) або хибне значення (0, false) на основі вхідних значень змінних (аргументів), які також можуть бути істинними або хибними.

Таблиця істинності – це прямокутна таблиця, що виражає відповідність між усіма наборами величин змінних (аргументів функції істинності) і величин функції істинності.

Умовні позначення логічних операцій в Python задані в таблиці 3.2.

Таблиця 3.2 – Умовні позначення логічних операцій в Python

Ч. ч.	Назва операції	Позначення	Приклад	Оператор Python	Приклад
1	Кон'юнкція	$\wedge$	(A $\wedge$ B)	and	A and B
2	Диз'юнкція	$\vee$	(A $\vee$ B)	or	A or B
3	Заперечення	$\neg$	($\neg$ A)	not	not A
4	Імплікація	$\rightarrow$	(A $\rightarrow$ B)	<=	A <= B
5	Тотожність	$\equiv$	(A $\equiv$ B)	==	A == B

Приклад виконання. Покладемо, що логічна функція F задається формулою

$$((x \vee \neg y) \wedge (z \rightarrow x)) \equiv (y \vee z)$$

Програмна реалізація побудови таблиці істинності функції F подана на рисунку 3.1.

```
def main():
# Побудова таблиці функції істинності формули алгебри висловлювань
 print('x y z F')
 range=[0,1]
 for x in range:
  for y in range:
   for z in range:
      if ((x or not(y)) and (z <= x)) == (y or z):
        print(x, y, z, "1")
      else: print(x, y, z, "0")

if __name__ == "__main__":
  main()
```

```
x y z F
0 0 0 0
0 0 1 0
0 1 0 0
0 1 1 0
1 0 0 0
1 0 1 1
1 1 0 1
1 1 1 1
```

Рисунок 3.1 – Побудова таблиці істинності функції F прикладу виконання

4 ПРАКТИЧНА РОБОТА «РОЗВ'ЯЗАННЯ ТИПОВИХ ЗАДАЧ КОМБІНАТОРИКИ. ПЕРЕСТАНОВКИ, КОМБІНАЦІЇ, РОЗМІЩЕННЯ»

4.1 Завдання на практичну роботу

1. За лекційним матеріалом дистанційного курсу «Прикладні задачі дискретного аналізу» (https://dl.kname.edu.ua/mod/resource/view.php?id=76326):
   – розібрати методику застосування принципу включення та виключення;
   – розібрати приклади застосування основних формул комбінаторики.
2. Виконати завдання для самостійної роботи (п. 3.3). При цьому з множини кожного з номерів самостійних завдань необхідно обрати тільки одне.
3. Створити Python програму реалізації одного з завдань комбінаторики п. 3.3. за вибором. Взяти до уваги теоретичні нотатки п. 4.4.
4. За результатами виконаних завдань створити звіт та викласти у відповідний дистанційний курс Moodle.

4.2 Теоретичні нотатки

Розглянемо підхід до розв'язання комбінаторних задач на прикладах.

Приклад 4.1. Знайти кількість натуральних чисел, що не перевищують 1 000 і не діляться на жодне з чисел 2, 5 та 7.

Застосуємо принцип включення та виключення. Позначимо через A_k множину натуральних чисел, що не перевищують 1 000 і діляться на k. Тоді множиною натуральних чисел, що не перевищують 1 000 і діляться або на 2, або на 5, або на 7, є множина

$$A_2 \cup A_5 \cup A_7,$$

а шуканою множиною є

$$E \setminus (A_2 \cup A_5 \cup A_7),$$

де множина-універсум E містить всі натуральні числа від 1 до 1 000.

Тоді

$$| E \setminus (A_2 \cup A_5 \cup A_7) | =$$

$$| E | - | E \cap (A_2 \cup A_5 \cup A_7) | = | E | - | A_3 \cup A_5 \cup A_7 |$$

$| E | = 1\,000$, а величину $| A_3 \cup A_5 \cup A_7 |$ обчислюють за формулою

$$| A \cup B \cup C | = | A | + | B | + | C | - (| A \cap B | + | A \cap C | + | B \cap C |) + | A \cap B \cap C |.$$

Крім того,

$$| A_k | = [1\,000/k] \text{ і } A_k \cap A_p = A_m,$$

де m – найменше спільне кратне чисел k і p (через [x/y] позначено цілу частину від ділення x на y).

Приклад 4.2. З'ясуємо, скількома способами можна розподілити k різних предметів серед n осіб.

Нехай A – множина осіб, серед яких розподіляють предмети.

Кожному варіанту розподілу поставимо у відповідність кортеж

$$(a_{i1}, a_{i2}, \ldots, a_{ik}),$$

де a_{ij} – і-та особа, яка одержала j-й предмет.

Установлена відповідність взаємно однозначна, отже, множина варіантів розподілу містить таку саму кількість елементів, що й множина всіх кортежів довжиною k, утворених з елементів множини A.

Тому шукане число становить $|A^k| = |A|^k = n^k$.

Приклад 4.3. Скількома способами можна розмістити на шахівниці 8 тур так, щоб вони не били одна одну?

Методичні пояснення. Є вісім способів розташувати першу туру на першій вертикалі шахівниці, сім способів розташувати другу туру на другій вертикалі і так далі, один спосіб розташувати восьму туру на останній вертикалі.

Приклад 4.4. Скільки існує n-значних десяткових чисел, усі цифри яких парні?

Методичні пояснення. Першу цифру такого числа можна обрати чотирма способами (2, 4, 6 або 8), а кожну наступну – п'ятьма.

Приклад 4.5. Скільки існує n-значних десяткових чисел, у запису яких є принаймні одна парна цифра?

Методичні пояснення. Від кількості всіх n-значних десяткових чисел потрібно відняти кількість n-значних чисел, усі цифри яких непарні.

Приклад 4.6. Визначити, скільки різних натуральних дільників має число:

а) $2^{1\,001} \cdot 3^{2\,015} \cdot 7^{2\,002}$.

Методичні пояснення. Довільний дільник цього числа має вигляд $2^n \cdot 3^m \cdot 7^k$, де n може набувати значення в діапазоні від 0 до 1 001 (тобто є 1 002 варіанти обрати n), m – від 0 до 2 015, а k – від 0 до 2 002.

Отже, шукана кількість дільників дорівнює $1\,002 \times 2\,016 \times 2\,003$;

б) $4\,004^{2\,124}$.

Методичні пояснення. Потрібно розкласти число 4 004 на прості множники й застосувати міркування, наведені в попередньому пункті.

Приклад 4.7. Скільки є перестановок цифр 0, 1, 2, …, 9, у яких цифра 3 займає третє місце, а цифра 5 – п'яте?

Методичні пояснення. Зафіксуємо в кожній перестановці цифру 3 на третьому місці, а цифру 5 – на п'ятому. На всіх інших восьми позиціях розташуємо всіма можливими способами решту вісім цифр.

Відповідь: 8!

Приклад 4.8. Скількома способами можна розташувати n нулів і k одиниць так, щоб жодні дві одиниці не стояли поряд?

Методичні пояснення. Запишемо спочатку всі n нулів.

Для запису одиниць існує n + 1 місце:

- одне – перед першим нулем,
- (n – 1) – між нулями та
- одне – після останнього нуля,

на які можна записувати по одній одиниці, щоб вони не стояли поряд. Отже, для кожного заданого розташування варто обрати k місць з n + 1 можливих.

Відповідь: C(n + 1, k).

4.3 Варіанти самостійних завдань

1. Нехай M – скінченна множина і A, B ⊆ M. Розташуйте у порядку неспадання такі величини:

а) $|B|, |A \cup B|, |\emptyset|, |A \cap B|, |M|$;

б) $|A \setminus B|, |A| + |B|, |A \Delta B|, |\emptyset|, |A \cup B|$.

2. Довести нерівності:

а) $|A \cup B| \le |A| + |B|$;

б) $|A \setminus B| \le |A|$;

в) $|A \Delta B| \le |A| + |B|$;

г) $|A \setminus B| \geq |A| - |B|$;

д) $|A \Delta B| \leq |A \cup B|$;

е) $|A| < |\beta(A)|$;

ж) $|A \cap B| \leq |A|$;

и) $|A \cap B| \leq |A \cup B|$.

3. Нехай A – скінченна множина, $a \in A$ – елемент множини A, $B \subseteq A$ – підмножина множини A. Яких підмножин множини A більше:

а) тих, що містять елемент a, чи тих, що не містять елемента a;

б) тих, що включають множину B, чи тих, що не перетинаються із множиною B;

в) тих, що містять множину B, чи тих, що не містять множину B?

4. Обстеження читацьких смаків студентів показало, що:

– 60 % студентів читають журнал А,
– 50 % – журнал Б,
– 50 % – журнал В,
– 30 % – журнали А та Б,
– 20 % – журнали Б та В,
– 40 % – журнали А та В,
– 10 % – журнали А, Б та В.

Скільки відсотків студентів:

а) читають принаймні один журнал;

б) не читають жодного з журналів;

в) читають точно два журнали;

г) читають не менше двох журналів?

5. Знайти кількість і суму чотиризначних натуральних чисел, що не діляться на жодне з таких чисел:

а) 2, 5, 11;

б) 6, 10, 18.

6. Скільки існує n-значних десяткових чисел, у запису яких:

а) немає цифри 7;

б) обов'язково є цифра 5;

в) які починаються із двох однакових цифр;

г) у яких сусідні цифри різні;

д) усі цифри яких непарні;

е) у запису яких є принаймні одна непарна цифра?

7. Номер автомашини складається із трьох літер українського алфавіту (що містить 33 літери) і чотирьох цифр.

Скільки можна скласти різних номерів автомашин?

8. Скількома способами можна розмістити на шахівниці розміром m × n дві різнокольорові тури так, щоб вони не били одна одну?

9. Скількома способами в множині A з n елементів можна вибрати дві підмножини, що не перетинаються?

10. Визначити, скільки різних натуральних дільників має число:

а) $3^{2\,015} \cdot 5^{2\,016} \cdot 11^{2\,017}$;

б) $2\,448^{2\,124}$;

в) $3\,124^{4\,004}$;

г) $9\,216^{2\,331}$.

11. На одній прямій дано n точок, а на іншій, паралельній першій, – m точок. Скільки існує трикутників, вершинами яких є ці точки?

12. Скількома способами з 6 інженерів і 14 робітників можна створити бригаду, яка складалася б із 2 інженерів і 5 робітників?

13. Визначити, скількома способами можна вибрати з натуральних чисел від 1 до 100 три натуральні числа так, щоб їх сума:

(а) була непарною;

(б) ділилася на три.

14. Скількома способами можна вибрати з n осіб групу людей для роботи, якщо група має складатися не менше ніж із p осіб?

15. Скількома способами можна посадити за круглий стіл n чоловіків і m жінок так, щоб жодні дві жінки не сиділи поруч?

16. Скількома способами можна роздати 28 кісток доміно чотирьом гравцям так, щоб кожний одержав 7 кісток?

Нехай дано n символів a, m символів b. Визначити кількість різних слів:

а) які можна скласти зі всіх цих символів;

б) які складаються зі всіх цих символів і у яких жодні два символи b не стоять поряд.

17. Скільки слів довжиною 5 можна утворити з літер a, b, c, якщо:

а) кожна із літер зустрічається у слові без обмежень;

б) літеру a можна використати в слові лише один раз;

в) літеру a можна використати в слові не більше двох разів?

4.4 Використання модулю itertools Python для комбінаторних задач

Модуль itertools – інструмент стандартної бібліотеки Python, що містить найпоширеніші шаблони ітераторів. Ітератор – це об’єкт, який дозволяє

індексувати (ітерувати) елементи деякої послідовності. Ітератор не зберігає всі значення відразу, а дозволяє послідовно пройти їх у циклі.

Для збереження всіх значень ітератора його можна перетворити на список за допомогою функції «list()».

Модуль itertools Python дозволяє вирішувати програмні завдання, побудовані на комбінаторних структурах. Розглянемо деякі корисні функції.

Переставлення (permutations)

Повертає послідовні перестановки k елементів в об'єкті, що ітерується. Елементи розглядаються як унікальні залежно від їхньої позиції, а не від їхнього значення. Таким чином, якщо вхідні елементи унікальні, то в кожній перестановці не буде значень, що повторюються.

Приклад 4.9. Знайти всі перестановки квітів вихідної множини A= «flowers»

A = {«троянда», «айстра», «лілія»}

та вивести кількість перестановок.

Програмна реалізація завдання прикладу мовою Python подана на рисунку 4.1.

```
import itertools # Завантаження бібліотеки itertools
flowers = [' троянда ',' айстра ',' лілія '] # Визначення вихідної множини A
print(' Перестановки:','\n')
for b in itertools.permutations (flowers): # Генерація перестановок
 print(*b)

print('\n','Всього перестановок:', len(list(itertools.permutations (flowers))))
```

```
 Перестановки:

троянда   айстра   лілія
троянда   лілія   айстра
айстра   троянда   лілія
айстра   лілія   троянда
лілія   троянда   айстра
лілія   айстра   троянда

Всього перестановок: 6
```

Рисунок 4.1 – Програмна реалізація генерації перестановок мовою Python

Сполуки (combinations)

Сполуки (комбінації) по k елементів – це вибрані з множини iterable об’єктів комбінації k об’єктів, що відрізняються хоча б одним об’єктом. Сполуки розрізняються лише елементами, порядок їх не є важливими: ab і ba – це одна й та ж сполука.

Приклад 4.10. Знайти всі сполуки періодів року – множина A = «seasons» – по два:

A = {«осінь», «зима», «весна», «літо»}

та вивести кількість сполук.

Програмна реалізація мовою Python подана на рисунку 4.2.

```
import itertools
seasons = [' осінь',' зима',' весна', 'літо'] # Визначення вихідної множини A
print(' Сполуки','\n')
for comb in itertools.combinations (seasons, 2):# Генерація сполук
 print(*comb)
print('\n','Всього сполук:', len (list (itertools.combinations (seasons, 2))))
```

```
Сполуки

осінь  зима
осінь  весна
осінь літо
зима  весна
зима літо
весна літо

Всього сполук: 6
```

Рисунок 4.2 – Програмна реалізація генерації сполук мовою Python

Розміщення permutations (iterable, k)

Розміщення з n елементів по k (k≤n) – будь-яка множина, що складається з k елементів, взятих у певному порядку з даних iterable елементів. Це ті ж сполуки, котрим важливий порядок появи елементів.

Приклад 4.10. Знайти всі сполуки періодів року – множина A = «seasons» – по два:

A = {«осінь», «зима», «весна», «літо»}

та вивести кількість розміщень.

Програмна реалізація мовою Python подана на рисунку 4.3.

```
import itertools
seasons = [' осінь',' зима',' весна', ' літо'] # Визначення вихідної множини A
print(' Розміщення','\n')
for perm in itertools.permutations (seasons, 2): # Генерація розміщень
    print (*perm)
print('\n','Всього розміщень:', len(list(itertools.permutations (seasons, 2))))
```

```
 Розміщення

 осінь  зима
 осінь  весна
 осінь літо
 зима  осінь
 зима  весна
 зима літо
 весна  осінь
 весна  зима
 весна літо
літо  осінь
літо  зима
літо  весна

 Всього розміщень: 12
```

Рисунок 4.3 – Програмна реалізація генерації розміщень мовою Python

Модуль itertools стандартизує основний набір швидких, ефективних за використанням пам’яті інструментів, які корисні як самі по собі, так і в комбінації. Повну документацію за модулем itertools можна знайти за адресою https://docs.python.org/3/library/itertools.html.

5 ПРАКТИЧНА РОБОТА «РОЗВ'ЯЗАННЯ ЗАДАЧ ЦІЛОЧИСЕЛЬНОЇ ОПТИМІЗАЦІЇ. ЗАДАЧА ПРО ПРИЗНАЧЕННЯ»

5.1 Завдання на практичну роботу

1. Побудувати математичну модель задачі про призначення у припущенні, що кількість виконавців та робіт дорівнює 4.

Матрицю вартості виконання робіт студент має скласти самостійно.

2. Розв'язати задачу про призначення (пункт 1) із застосуванням засобу **Solver** програмного середовища **MS Excel.**

3. Побудувати математичну модель задачі про призначення у припущенні, що кількість виконавців перевищує кількість робіт: 6 виконавців та 4 роботи:

а) розглянути необхідність призначення тільки одного виконавця на одну роботу;

б) розглянути можливість виконання однієї роботи двома виконавцями.

При цьому, як і раніше, кожна робота має бути виконана.

Матрицю вартості виконання робіт студент має скласти самостійно.

4. Розв'язати задачі про призначення (пункт 2) із застосуванням засобу **Solver** програмного середовища **MS Excel.**

5. Скласти звіт з практичної роботи.

5.2 Теоретичні нотатки

Розглянемо основні етапи побудови математичної моделі та програмної реалізації задачі про призначення (задачі з бінарними змінними) на такому прикладі.

Приклад розв'язання.

Маємо чотири виконавця $\{r_1, r_2, r_3, r_4\}$ і чотири види робіт $\{t_1, t_2, t_3, t_4\}$.

Вартість c_{ij} виконання і-м виконавцем j-ї роботи наведена в таблиці 5.1, де під рядком розуміємо виконавця, а під стовпчиком – роботу.

Необхідно так скласти план виконання робіт, щоб усі роботи були виконаними, при цьому кожен виконавець виконував лише одну роботу, а сумарна вартість виконання всіх робіт була мінімальною.

Для розв'язання цієї задачі побудуємо її математичну модель.

Таблиця 5.1 – Вартість виконання робіт

		Види робіт			
		t_1	t_2	t_3	t_4
Робітники	r_1	1	4	6	3
	r_2	9	10	7	9
	r_3	4	5	11	7
	r_4	8	7	8	5

Позначимо символом x_{ij} змінну, яка має лише два допустимих значення: 0 або 1. Такі змінні є бінарними.

При цьому будемо вважати, що:

- $x_{ij} = 1$, якщо і-м виконавцем виконується j-та робота;
- $x_{ij} = 0$, якщо і-м виконавцем не виконується j-та робота.

Тоді ***математичну модель задачі про призначення*** можна сформулювати так:

- мінімізувати

$$z=\sum_{i=1}^{4}\sum_{j=1}^{4} c_{ij}x_{ij},$$

- при обмеженнях

$$\sum_{i=1}^{4} x_{ij} = 1,$$

$$\sum_{j=1}^{4} x_{ij} =1,$$

$$x_{ij} \in \{0,1\},\ i=\overline{1,4},\ j=\overline{1,4}.$$

Для розв'язання цієї задачі за допомогою засобу **Solver** програмного середовища **MS Excel** необхідно виконати такі попередні дії (рис. 5.1):

1. У комірки діапазону A2:D5 ввести вартість робіт (на рис. 5.1 вміст відповідних комірок показаний зеленим кольором).

2. В комірках діапазону F2:I5 занотувати незалежні змінні.

3. Ввести у комірку G7 функцію цілі, що обчислює вартість робіт:

=SUMPRODUCT(A2:D5; F2:I5)

1. У комірки діапазону F10:F13 и F14:F17 ввести формули, які задають ліві частини обмежень (рис. 5.1. бузковий – за стовпчиками матриці вартостей – та блакитний – за рядками матриці вартостей – кольори відповідно).

Комірка	Формула	Комірка	Формула
F10	=SUM(F2:F5)	F14	=SUM(F2:I2)
F11	=SUM(G2:G5)	F15	=SUM(F3:I3)
F12	=SUM(H2:H5)	F16	=SUM(F4:I4)
F13	=SUM(I2:15)	F17	=SUM(F5:I5)

На рисунку 5.1 подано оптимальний розв’язок задачі про призначення.

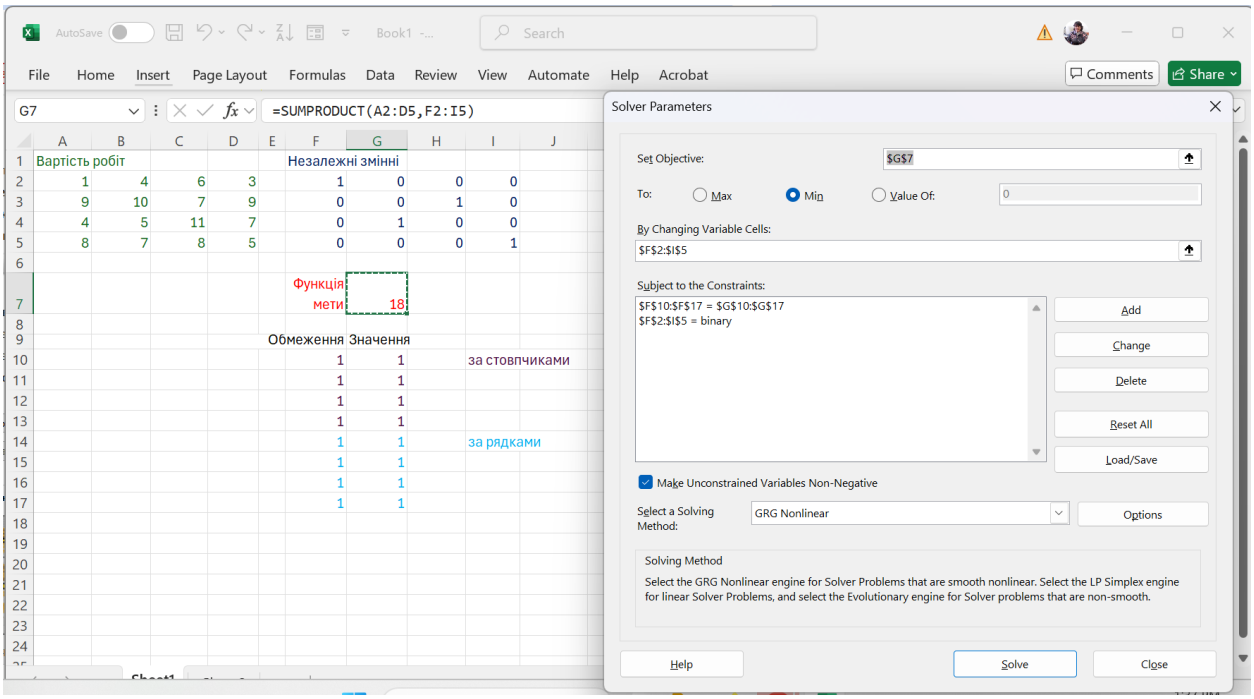


Рисунок 5.1 – Вихідні дані задачі про призначення і заповнене діалогове вікно засобу **Solver**

6 ПРАКТИЧНА РОБОТА «РОЗВ'ЯЗАННЯ ЗАДАЧ ЦІЛОЧИСЕЛЬНОЇ ОПТИМІЗАЦІЇ. ЗАДАЧА ПРО РЮКЗАК»

6.1 Завдання на практичну роботу

1. Побудувати математичну модель задачі про рюкзак, використовуючи дані таблиці 6.1 у припущенні, що предмет кожного типу q_j розміщений в рюкзаку тільки один раз.

Таблиця 6.1 – Предмети для задачі про рюкзак

Номер предмета	Предмет	Об'єм	Вартість	Номер предмета	Предмет	Об'єм	Вартість
1	Риба			8	Кофе		
2	Молоко			9	М'ясо		
3	Хліб			10	Шоколад		
4	Вода			11	Масло		
5	Папір			12	Телефон		
6	Яблука			13	Сік		
7	Сир			14	Огірки		

Вартість та об'єм предметів (табл. 6.2) студент має скласти самостійно.

2. Побудувати блок-схему жадібного алгоритму розв'язання задачі про рюкзак.

3. Виконати програмну реалізацію жадібного алгоритму розв'язання задачі про рюкзак із застосуванням мови програмування високого рівня.

4. Скласти звіт із практичної роботи.

6.2 Теоретичні нотатки щодо жадібного алгоритму

Постановка задачі про рюкзак є такою. Мандрівник, збираючись в похід, хоче укласти в свій рюкзак певну скінчену множину предметів $Q = \{q_1, q_2, \ldots, q_n\}$, для кожного $q_j \in Q$ відома вартість c_j і визначений об'єм v_j. Також є рюкзак об'ємом V. Традиційно вважають, що c_j, v_j, V – цілі невід'ємні числа.

Необхідно запакувати рюкзак так, щоб загальна вартість запакованих предметів була якнайбільшою, а їхній загальний об'єм не перевищував V.

Побудова математичної моделі.

Позначимо через $x_1, x_2, \ldots, x_n$ – кількість взятих предметів.

Тоді математична модель задачі є такою:

$$z = \sum_{j=1}^{n} c_j x_j$$

$$z \to \max,$$

$$\sum_{j=1}^{n} v_j x_j \le V,$$

$$x_j \ge 0,\ x_j \text{ – ціле},\ j=1,2,\ldots, n.$$

Зауваження 6.1. Найпоширеніші окремі випадки цієї задачі, коли $x_j = 0$ або 1.

Задача про рюкзак має складність NP-повної задачі.

На сьогодні невідомий точний метод розв'язання NP-повної задачі про рюкзак. Разом із тим існують поліноміальні алгоритми, що генерують не оптимальний розв'язок задачі, а близький до нього, тобто наближений розв'язок. Знайти наближений розв'язок може бути цілком достатньо для практичного застосування.

До наближених методів для задачі про рюкзак відносять:

• жадібні алгоритми;

• алгоритми мурашиної колонії;

• генетичні алгоритми та ін.

Жадібний алгоритм для задачі про рюкзак такий:

• спочатку множина предметів Q упорядковується за зменшенням «питомої цінності» (або ціни одиниці об'єму) предметів, тобто так, щоб

$$\frac{c_1}{v_1} \ge \frac{c_2}{v_2} \ge \ldots \ge \frac{c_n}{v_n};$$

• потім, починаючи з порожньої множини, до наближеного розв'язку Q′ послідовно додаються предмети з упорядкованої множини Q;

• при кожному черговому додаванні предмета в рюкзак виконується перевірка, чи не перевищує / перевищує його об'єм величини того об'єму рюкзака, що залишився;

• закінчення перегляду всіх предметів завершує процес побудови наближеного розв'язку задачі про рюкзак.

7 ПРАКТИЧНА РОБОТА «ОПИС ГРАФОВИХ СТРУКТУР»

7.1 Завдання на практичну роботу

1. Розглянути граф G=< V, E > відповідно до варіанта завдання.

Варіанти завдань подані в таблиці 7.1. Графи побудовані за допомогою програмного застосунку Graph drawer (https://g.ivank.net/).

Представити граф явним способом, тобто записати множини V, E, де V – множина вершин, і E – множина ребер, причому кожне ребро задати парою інцидентних цьому ребру вершин.

У варіантах завдань вершини графа позначені цифрами, тому множина V є числовою, а елементами множини E є пари чисел – відповідних номерів вершин, інцидентних цьому ребру.

Наприклад, у завданні варіанта 1:

– множина V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

– множина E = {(1, 3), (1,4), (1,6), (2,4), (2,5), (2,7), (3,5), (3,8), (4,9), (5,10), (6,7), (6,10), (7,8), (8,9), (9,10)}.

2. Побудувати алгоритм визначення матриці інцидентності (непарні варіанти завдань) та матриці суміжності (парні варіанти завдань) графа, використовуючи явне завдання графа, отримане в п. 1 завдання.

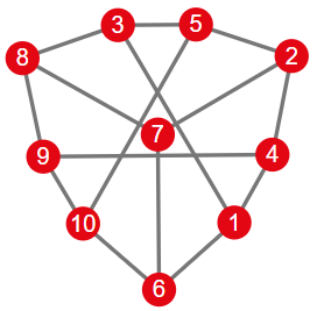
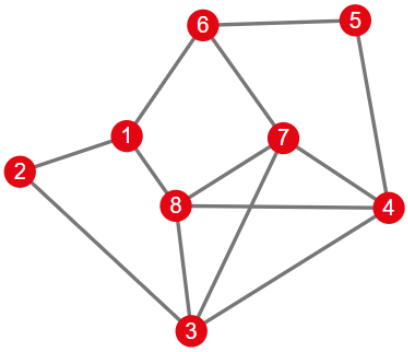
3. Створити програмну реалізацію алгоритмів пункту 2 завдання.

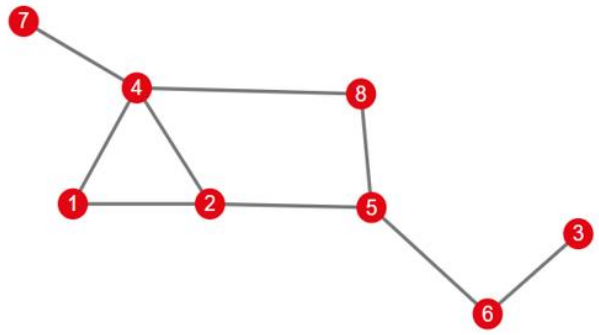
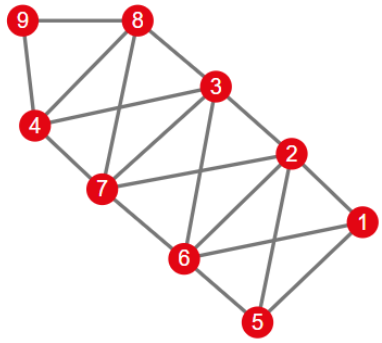
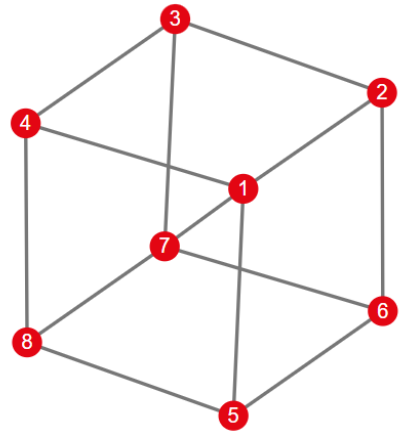
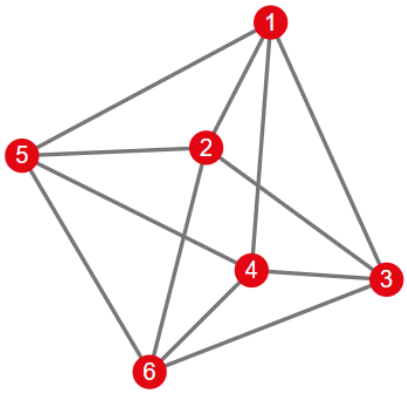
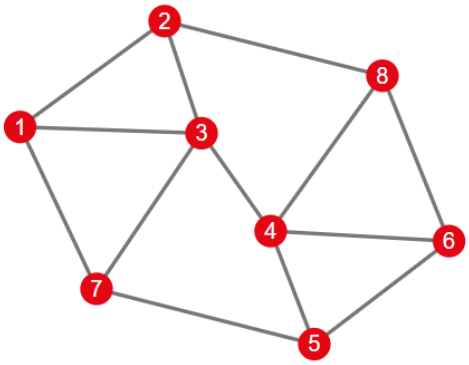
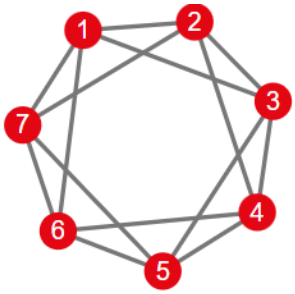
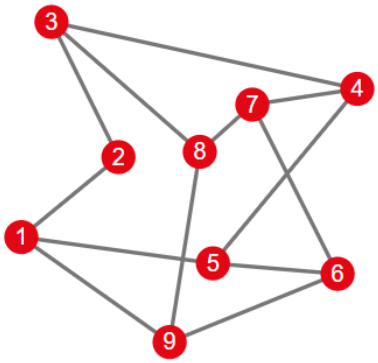
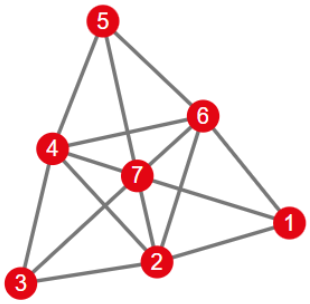
4. Протестувати програму на завданні відповідного варіанта.

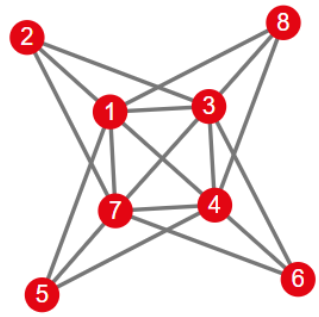
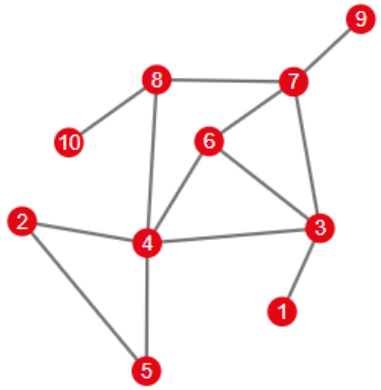
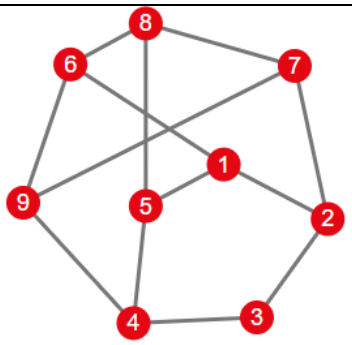
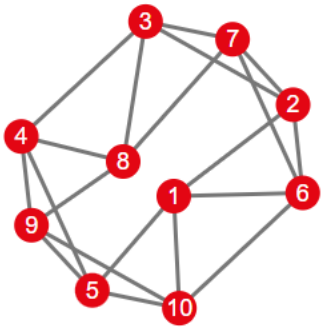
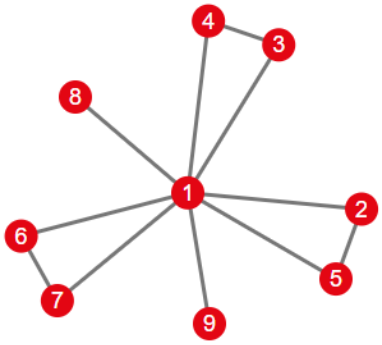
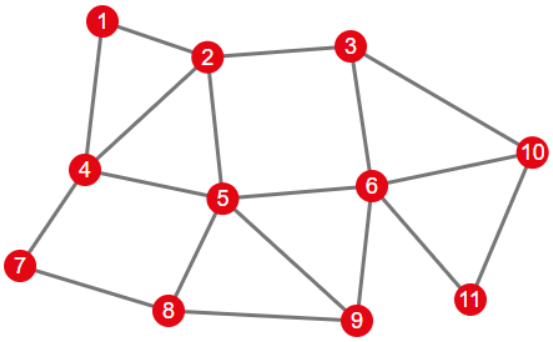
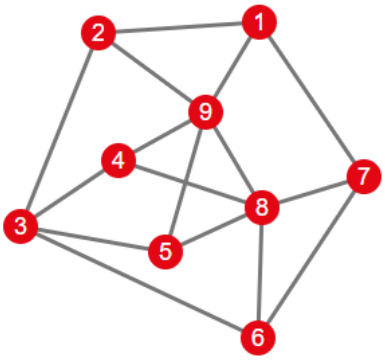
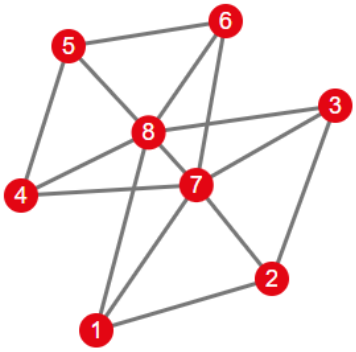
5. Відтворити граф варіанта завдання за допомогою програмного застосунку Graph drawer (https://g.ivank.net/).

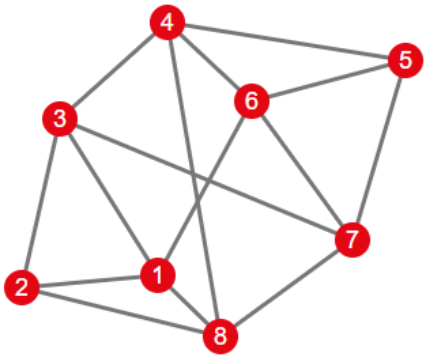
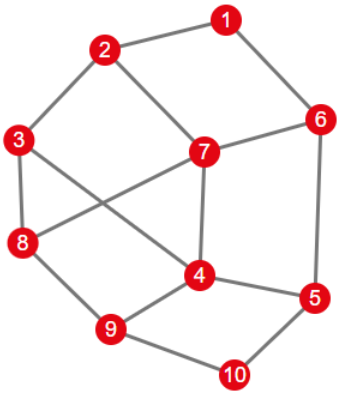
6. Скласти звіт з практичної роботи.

Таблиця 7.1 – Варіанти завдань

Варіант 1	Варіант 2
	

Варіант 3	Варіант 4
	
Варіант 5	Варіант 6
	
Варіант 7	Варіант 8
	
Варіант 9	Варіант 10
	

Варіант 11	Варіант 12
	
Варіант 13	Варіант 14
	
Варіант 15	Варіант 16
	
Варіант 17	Варіант 18
	

Варіант 19	Варіант 20
	

8 ПРАКТИЧНА РОБОТА «АЛГОРИТМ НАЙКОРОТШОГО ШЛЯХУ ДЕЙКСТРИ»

8.1 Завдання на практичну роботу

1. Розглянути граф G=< V, E > відповідно до варіанта завдання (табл. 7.1).

2. Задати ваги ребер графу G=< V, E >.

Ваги ребер графу студент має задати самостійно.

3. Розглянути алгоритм найкоротшого шляху Дейкстри (див. теоретичні нотатки нижче).

4. Створити програмну реалізацію визначення найкоротшого шляху Дейкстри, використовуючи мову високого рівня, наприклад Python.

5. Протестувати програму на завданні відповідного варіанта.

6. Скласти звіт із практичної роботи.

8.2 Теоретичні нотатки щодо алгоритму Дейкстри

Розглянемо таку задачу.

Нехай є зважений неорієнтований граф G=< V, E >, де V – множина вершин графу G, $|V|= N_V$, $E=\{(v_u,v_w)\}$ – множина ребер графу,і певна початкова вершина (джерело) графа G.

Необхідно знайти найкоротші шляхи з ребер графа G від джерела до всіх вершин графа.

Таким чином, необхідно побудувати дерево найкоротшого шляху (**shortest path tree – SPT**) із заданим джерелом як коренем дерева.

Сутність алгоритму Дейкстри є такою. Генеруються два набори вершин графу G, один набір − V_1 − містить вершини дерева найкоротшого шляху, інший набір − V_2 − включає вершини, ще не додані до дерева найкоротшого шляху. На початку роботи алгоритму $V_2 = V$.

На кожному кроці алгоритму визначається вершина, яка знаходиться в наборі V_2 , є суміжною до певної вершини набору V_1 і має мінімальну відстань від джерела.

Нижче наведено докладні кроки, використані в алгоритмі Дейкстри для пошуку найкоротшого шляху від джерела до всіх інших вершин цього графу G=< V, E >.

Алгоритм Дейкстри.

1) задати граф G=< V, E >, V – множина вершин графу G, E – множина ребер графу G.

Ввести:

N_V − кількість вершин, $|V|= N_V$, N_E − кількість ребер, $|E|=N_E$, $E=\{(v_u,v_w)\}$, $\{u,w\} \in \{1,2,\ldots, N_V\}$, $u \neq w$.

$D_E = \{d_{uw}\}$ – множина ваг ребер графу G, $\{u,w\}$, $u \neq w$, $|D_E|=N_E$;

2) створити набір V_1, який відстежує вершини, включені до дерева найкоротшого шляху. Спочатку цей набір порожній. На першому кроці алгоритму всі вершини графа належать набору V_2;

3) обрати джерело – вихідну вершину $v_{start} \in V_2$. Не втрачаючи загальності, можна покласти, що це вершина v_1, тобто $v_{start} = v_1$.

Призначити значення відстані 0 для вихідної вершини v_1 − джерела, як першої вершини. Помістити вершину v_1 до набору V_1.

Ініціалізувати значення відстаней S_i, $i = 2,3,\ldots, N_V$ всім вершинам вхідного графу від джерела – початкової вершини як дуже великі (у термінах початкової інформації про граф);

4) призначити всім сусіднім до вершини v_1 вершинам $v_w \in V_2$ вагу S_w таку, що $S_w = d_{1w}$, $d_{1w} \in D_E$;

5) вибрати вершину v_w, якої немає в V_1, яка є сусідньою до вершин набору V_1 (на першому кроці алгоритму – до вершини v_1) та має мінімальне значення відстані від вершини v_1. На першому кроці алгоритму ця вершина v_w обирається як суміжна з v_1 і така, що ребро (v_1, v_w) має найменшу вагу;

6) включити вершину v_w до набору V_1;

7) оновити значення відстані всіх вершин $v \in V_2$, сусідніх з вершинами набору V_1.

Для цього для кожної вершини $v_w \in V_2$, сусідньої з вершинами v_u набору V_1, визначити суму S_w значення ваги S_u і ваги ребра (v_u, v_w):

$$\forall\, v_w \in V_2, v_u \in V_1, (v_u, v_w) \in E, \quad S_w = S_u + d_{uw},$$

Якщо поточне значення ваги вершини більше за оцінку S_w, то оновити значення ваги;

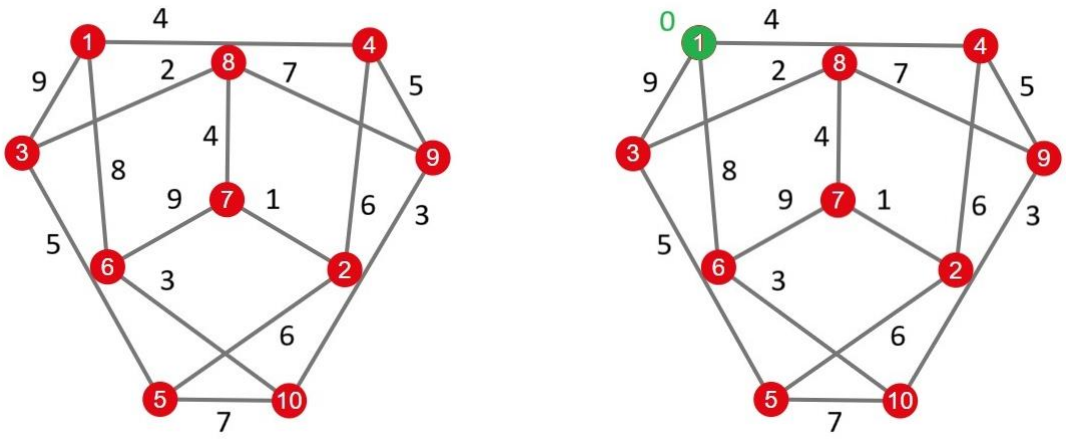
8) якщо кількість вершин набору V_1 менша за N_V, то повернутися до кроку 4.

Приклад розв'язання. Нехай є заданим граф G=< V, E >, |V|= 10, |E|=15 (рис. 8.1, а). Червоним кольором на рисунку 8.1 (і далі на рисунках прикладу) помічені вершини набору V_2, зеленим кольором – вершини набору V_1.

Набір V_1 спочатку містить тільки вершину v_1, а відстані, призначені вершинам, складають S ={0, SUP, SUP, SUP, SUP, SUP, SUP, SUP, SUP, SUP}, де SUP позначає точну верхню границю, яку можна визначити як

$$SUP=\sum_{j=1}^{N_E} d_j. \qquad (8.1)$$

Після включення 0 до S оновлюємо значення відстані сусідніх вершин. Сусідні вершини 0 – це вершини 3, 4 та 6. На рисунку 8.2 (і далі на рисунках прикладу) суміжні вершини позначені синім кольором.

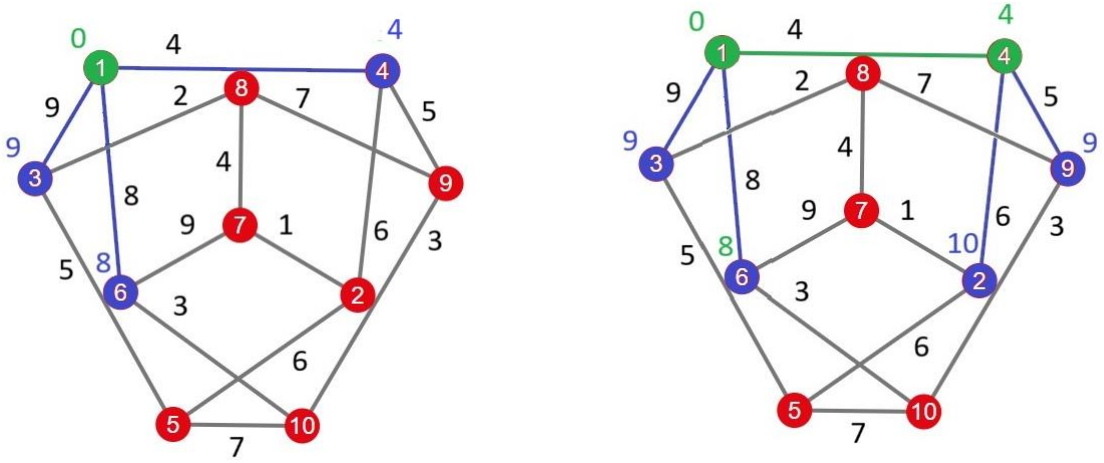


а) б)

Рисунок 8.1 – Вихідний граф та перший крок алгоритму Дейкстри:
а) вихідний граф G=< V, E >, |V|= 10, |E|=15;
б) вершина $v_1 \in V_1$

Значення відстаней до вершин {3, 4, 6} оновлюються як {9, 4, 8} відповідно (рис. 8.2, а):

S ={0, SUP, 9, 4, SUP, 8, SUP, SUP, SUP, SUP}.



а) б)

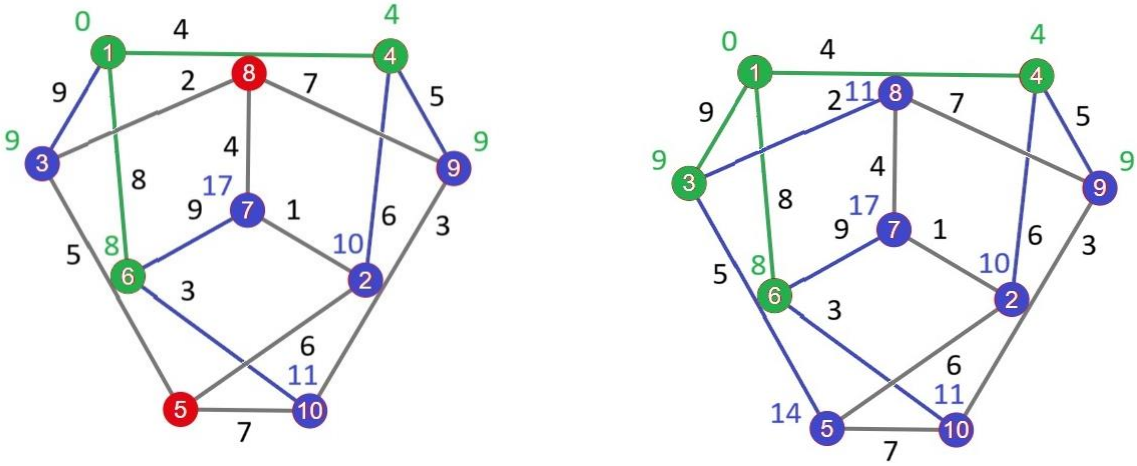
Рисунок 8.2 – Другий крок алгоритму Дейкстри:
а) вершини набору V_2, суміжні з вершинами набору V_1, позначені синім;
б) на другому кроці алгоритму Дейкстри обрано вершину v_4: $v_4 \in V_1$.

Вершина v_4 з мінімальним значенням ваги обирається для включення в V_1: V_1={1, 4}.

Значення відстаней до вершини $\{v_2, v_3, v_6, v_9\} \in V_2$, що є суміжними до вершин $\{v_1, v_4\} \in V_1$, оновлюються відповідно (рис. 8.2, б):

S ={0, 10, 9, 4, SUP, 8, SUP, SUP, 9, SUP}.

На наступній ітерації алгоритму саме ці вершини $\{v_2, v_3, v_6, v_9\}$ розглядаються та обирається вершина з мінімальним значенням відстані, яка ще не включена в V_1. Це вершина v_6, що додається до V_1: V_1={1, 4, 6} (рис. 8.3, а).



а) б)

Рисунок 8.3 – Третій та четвертий кроки алгоритму Дейкстри:
а) на третьому кроці алгоритму Дейкстри обрано вершину v_6: $v_6 \in V_1$;
б) на четвертому кроці алгоритму Дейкстри обрано вершину v_3: $v_3 \in V_1$.

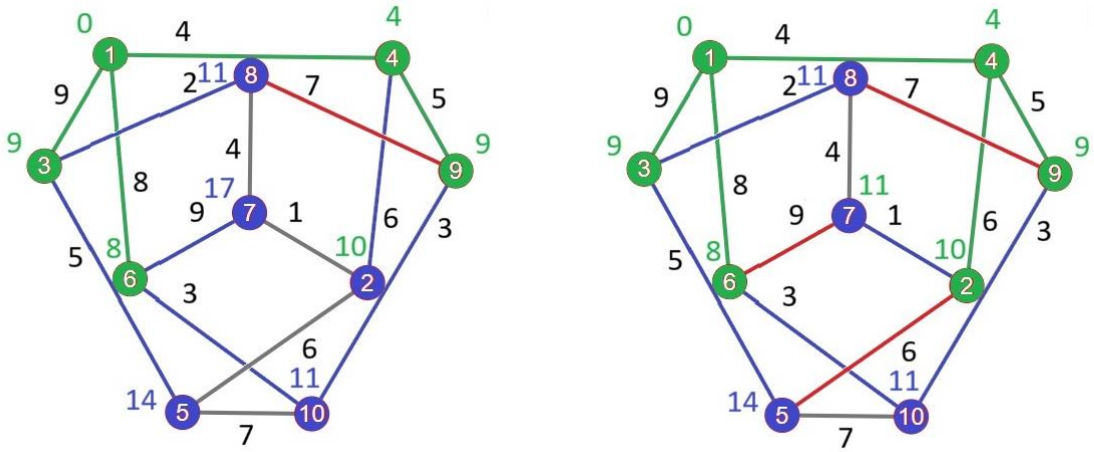
Після включення вершини v_6 до набору V_1 оновлюємо множину сусідніх вершин (синій колір на рис. 8.3, а) та значення відстаней:

S ={0, 10, 9, 4, SUP, 8, 17, SUP, 9, 11}.

Дві вершини: {v_3, v_9} мають однакову оцінку відстані від початкової вершини v_1, обираємо вершину з меншим номером, тобто v_3 (рис. 8.3, б).
Множина S відстаней набуває вигляду

S ={0, 10, 9, 4, 14, 8, 17, 11, 9, 11}.

Отже, наступним кроком додаємо вершину v_9 до набору V_1 (рис. 8.4, а). При цьому виключається з розгляду ребро (9,8) (на рис. 8.4 це ребро показано червоним), тому що шлях до вершини v_8 через вершину v_3 коротший (оцінка – 11 одиниць) ніж через вершину v_9 (оцінка – 16 одиниць).



а) б)

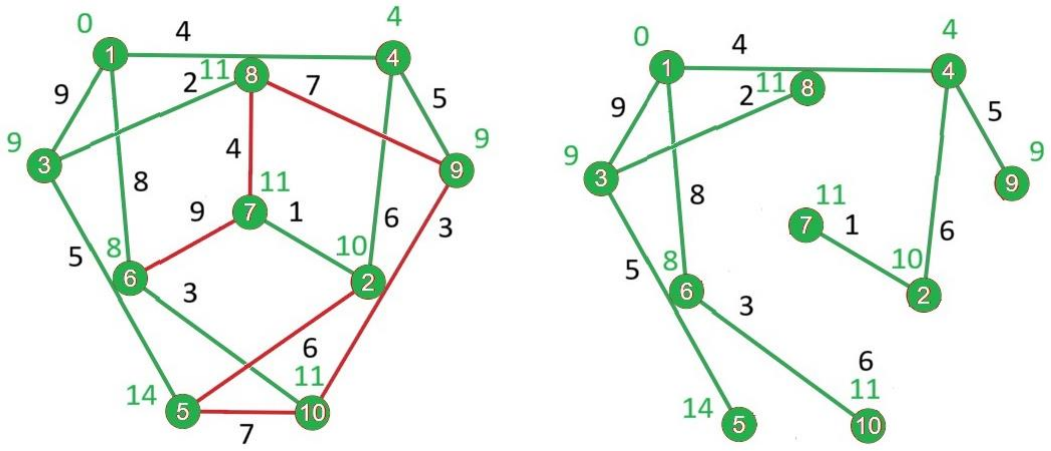
Рисунок 8.4 – П’ятий та шостий кроки алгоритму Дейкстри:
а) на п’ятому кроці алгоритму Дейкстри обрано вершину v_9: $v_9 \in V_1$;
б) на шостому кроці алгоритму Дейкстри обрано вершину v_2: $v_2 \in V_1$.

За множиною відстаней S обирається вершина v_2 з мінімальною оцінкою 10 (рис. 8.4, б). Множина S відстаней оновлюється:

S ={0, 10, 9, 4, 14, 8, 11, 11, 9, 11}.

та ребра (6,7) та (2,5) не розглядаються, тому що до відповідних вершин v_5, v_7 на графі від початкової вершини v_1 існує коротший шлях.

Діючи аналогічно і повторюючи кроки алгоритму Дейкстри, як результат отримуємо дерево найкоротших шляхів (рис. 8.5).



а) б)

Рисунок 8.5 – Третій та четвертий кроки алгоритму Дейкстри:
а) на третьому кроці алгоритму Дейкстри обрано вершину v_6: $v_6 \in V_1$;
б) на четвертому кроці алгоритму Дейкстри обрано вершину v_3: $v_3 \in V_1$.

Одночасно отримуємо множину **S = {0, 10, 9, 4, 14, 8, 11, 11, 9, 11}** найкоротших відстаней на заданому графі.

8.3 Приклад програмної реалізації алгоритму Дейкстри мовою Python

Програмна реалізація алгоритму Дейкстри, виконана Миколою Михайленком, студентом гр. ІСтаТ 2023-1, подана на рисунках 8.6–8.10.

Програма добре задокументована, містить частину введення інформації про граф (рис. 8.6), три функції: `build_graph`, `dijkstra`, `reconstruct_path` (рис.8.7–8.9 відповідно), а також основну частину (рис. 8.10) тому наведена практично без додаткових коментарів.

Результати роботи програми на вихідних даних прикладу розв’язання (рис. 8.1, а) наведені на рисунку 8.11.

```python
import heapq
from collections import defaultdict

# Список ребер графу: (звідки, куди, вага)

edges = [
    (1, 4, 4), (1, 3, 9), (1, 6, 8), (4, 2, 6),
    (2, 5, 6), (2, 7, 1), (3, 5, 5), (3, 8, 2), (4, 9, 5),
    (5, 10, 7), (6, 7, 9), (6, 10, 3),
    (7, 8, 4), (8, 9, 7), (9, 10, 3)
]
```

Рисунок 8.6 – Введення інформації про ребра графу

```python
# Побудова графа у вигляді словника суміжності
def build_graph(edges):
    graph = defaultdict(list)  # Кожній вершині відповідає список її сусідів
                               #(вершина + вага ребра)
    for src, dest, weight in edges:  # Проходимо по всіх ребрах
        graph[src].append((dest, weight))  # Додаємо напрямок з src у dest
                                           # з відповідною вагою
        graph[dest].append((src, weight))  # Додаємо зворотній звязок
    return graph  # Повертаємо побудований граф
```

Рисунок 8.7 – Функція «`build_graph`» повертає граф у вигляді словника

```python
# Реалізація алгоритму Дейкстри
def dijkstra(graph, start, vertices):
    distances = {v: float('inf') for v in vertices}  # Ініціалізуємо відстані
                                                     # як нескінченність
    predecessors = {v: None for v in vertices}  # Таблиця попередників для
                                                # відновлення шляху
    distances[start] = 0  # Відстань до стартової вершини = 0
    queue = [(0, start)]  # Черга з пріоритетом: (відстань, вершина)

    while queue:  # Поки черга не порожня
      current_dist, current_vertex = heapq.heappop(queue) # Визначаємо вершину
                                                          # з мінімальною відстанню
      if current_dist > distances[current_vertex]:  # Якщо ця відстань вже застаріла
                                                    # – пропускаємо
            continue
      for neighbor, weight in graph[current_vertex]:  # Перебираємо всіх сусідів
                                                      # поточної вершини
         new_dist = current_dist + weight  # Обчислюємо нову відстань до сусіда
         if new_dist < distances[neighbor]:  # Якщо нова відстань краща
               distances[neighbor] = new_dist  # Оновлюємо найкоротшу відстань
               predecessors[neighbor] = current_vertex  # Зберігаємо попередника
               heapq.heappush(queue, (new_dist, neighbor))  # Додаємо сусіда до черги
    return distances, predecessors  # Повертаємо словники відстаней та попередників
```

Рисунок 8.8 – Функція «`dijkstra`» повертає найкоротші шляхи від всіх інших вершин графа до вершини з номером start

```python
# Відновлення шляху від старту до заданої вершини
def reconstruct_path(predecessors, vertex):
    path = []  # Список для побудови шляху
    while vertex is not None:  # Рухаємось до початку шляху (поки не дійдемо до None)
        path.append(vertex)  # Додаємо вершину до шляху
        vertex = predecessors[vertex]  # Переходимо до попередника
    return path[::-1]  # Повертаємо шлях у правильному порядку (від старту до цілі)
```

Рисунок 8.9 – Функція «reconstruct_path» повертає найкоротші шляхи від вершини з номером start до всіх інших вершин графа

```python
# Основна частина програми

graph = build_graph(edges)  # Створюємо граф з ребер

# Визначаємо всі унікальні вершини на основі списку ребер
vertices = set()
for u, v, _ in edges:
    vertices.update([u, v])  # Додаємо обидві вершини кожного ребра

# Виконуємо алгоритм Дейкстри від вершини 1
distances, predecessors = dijkstra(graph, 1, vertices)

# Виводимо результати
print("Найкоротші шляхи з вершини 1:")
for v in sorted(vertices):  # Перебираємо всі вершини у відсортованому порядку
    path = reconstruct_path(predecessors, v)  # Відновлюємо шлях до вершини v
    weight = distances[v]  # Отримуємо вагу найкоротшого шляху
    if weight == float('inf'):  # Якщо шлях недосяжний
        print(f"{v}: недосяжна")  # Виводимо повідомлення
    else:  # Інакше
        # Виводимо вагу і шлях
        print(f"{v}: вага = {weight}, шлях: {' → '.join(map(str, path))}")
```

Рисунок 8.10 – Основна частина програми, що містить виклики функцій та упорядкування виведення результатів

```
Найкоротші шляхи з вершини 1:
1: вага = 0, шлях: 1
2: вага = 10, шлях: 1 → 4 → 2
3: вага = 9, шлях: 1 → 3
4: вага = 4, шлях: 1 → 4
5: вага = 14, шлях: 1 → 3 → 5
6: вага = 8, шлях: 1 → 6
7: вага = 11, шлях: 1 → 4 → 2 → 7
8: вага = 11, шлях: 1 → 3 → 8
9: вага = 9, шлях: 1 → 4 → 9
10: вага = 11, шлях: 1 → 6 → 10
```

Рисунок 8.11 – Результат побудови дерева найкоротших шляхів

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Дистанційний курс «Прикладні задачі дискретного аналізу» [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу: https://dl.kname.edu.ua/course/view.php?id=1752, вільний (дата звернення 05.05.2025). – Назва з екрана.

2. Федак І. В. Курс лекцій з функціонального аналізу та теорії міри. Ч.1. Вимірні множини та вимірні функції [Електрон. ресурс]: навч. посіб. / І. В. Федак. – Електрон. текст. дані. – Івано-Франківськ : ПНУ імені Василя Стефаника, 2020. – 52 с. – Режим доступу: https://kmfa.pnu.edu.ua/wp-content/uploads/sites/64/2020/02-Федак-І.В.-Курс-лекцій-з-функціонального-аналізу-та-теорії-міри-Ч1.pdf, вільний (дата звернення 05.05.2025). – Назва з екрана.

3. Rothvoss T. Discrete Optimization [Electronic resource]: Course Notes / T. Rothvoss. – Electronic text data. – Washingtong: University of Washingtong, 2020. – 86 p. – Regime of access: https://sites.math.washington.edu/~rothvoss/lecturenotes/DisOpt409-Spring2020.pdf, free (application date: 05.05.2025). – Header from the screen.

4. Темнікова О. Л. Дискретна математика : Конспект лекцій (Частина 1) [Електрон. ресурс] : навч. посіб / О. Л. Темнікова. – Електрон. текст. дані. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 154 с. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/42839/1/LectureDM1Temnikova.pdf, вільний (дата звернення 05.05.2025). – Назва з екрана.

5. Кузьменко І.М. Теорія графів [Електрон. ресурс] : навч. посіб. / І. М. Кузьменко. – Електрон. текст. дані. – Київ : КПІ ім. Ігоря Сікорського. – 2020. – 71 с. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/35854/1/Teoriia_hrafiv.pdf, вільний (дата звернення 05.05.2025). – Назва з екрана.

6. W. J. Martin III. Discrete Optimization [Electronic resource] : Course Notes / Martin III W. J. – Electronic text data. – Worcester: Worcester Polytechnic Institute. – 2022. – 232 p. – Regime of access: https://users.wpi.edu/~martin/TEACHING/3233/MA3233LectureNotes.pdf, free (application date: 05.05.2025). – Header from the screen.

7. Дослідження операцій: Вступ до дискретного програмування: Практикум: [Електрон. ресурс] : навч. посіб. / О. Г. Жданова, В. Д. Попенко, М. О. Сперкач. – Електрон. текст. дані. – Київ : КПІ ім. Ігоря Сікорського, 2019. – 47 с. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/32225/1/Praktykum2_Dosl.operats.-DyskrProhr.pdf, вільний (дата звернення 05.05.2025). – Назва з екрана.

8. Course of Discrete Mathematics [Electronic resource]. – Electronic text data. – Regime of access: https://www.coursera.org/learn/discrete-mathematics, free (application date: 05.05.2025). – Header from the screen.

9. Course of Data Structures and Algorithms Specialization. Algorithms on Graphs [Electronic resource]. – Electronic text data. – Regime of access: https://www.coursera.org/learn/algorithms-on-graphs?specialization=data-structuresalgorithms, free (application date: 05.05.2025). – Header from the screen10.

10. Discrete Mathematics. Master Discrete Math with 400+ Practice Questions and Quizzes: Foundational for Computer Science and Math Students [Electronic resource]. – Electronic text data. – Regime of access: https://www.udemy.com/course/discrete-math/?srsltid=AfmBOoq6m7ASVGKewZKeoxFrxiuYhRgLkLYXZCVfBP4FXm9daW2wcbSI, free (application date: 05.05.2025). – Header from the screen.

Електронне навчальне видання

Методичні рекомендації
до проведення практичних занять
із навчальної дисципліни

«ПРИКЛАДНІ ЗАДАЧІ ДИСКРЕТНОГО АНАЛІЗУ»

(*для здобувачів першого (бакалаврського) рівня вищої освіти всіх форм навчання зі спеціальностей 122, F3 – Комп'ютерні науки, 126, F6 – Інформаційні системи та технології*)

Укладач **НОВОЖИЛОВА** Марина Володимирівна

Відповідальний за випуск *М. Ю. Карпенко*
Редактор *О. В. Михаленко*
Комп'ютерне верстання *М. Ю. Карпенко*

План 2022, поз. 245М

Підп. до друку 06.06.2025. Формат 60 × 84/16.
Ум. друк. арк. 3,3.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Чорноглазівська (Маршала Бажанова), 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК № 5328 від 11.04.2017.