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

Р. М. Бабаков

Методичні рекомендації
до виконання лабораторних робіт з дисципліни

ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

для здобувачів освіти ОС «Бакалавр» денної форми навчання
спеціальності 122 Комп’ютерні науки

Вінниця
2024

УДК 004.75(076.5)
Б 12

Рекомендовано до друку вченою радою
факультету інформаційних і прикладних технологій
Донецького національного університету імені Василя Стуса
(протокол № 14 від 19 червня 2024 р.)

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

Рецензенти:
О. В. Зелінська, канд. техн. наук, завідувач кафедри інформаційних технологій Донецького національного університету імені Василя Стуса;
Т. В. Січко, канд. техн. наук, доцент кафедри інформаційних технологій Донецького національного університету імені Василя Стуса.

Р. М. Бабаков
Б 12 Методичні рекомендації до виконання лабораторних робіт з дисципліни «Технології розподілених та паралельних обчислень» для здобувачів ОС «Бакалавр» денної форми навчання спеціальності 122 Комп'ютерні науки. Вінниця: ДонНУ імені Василя Стуса, 2024. 76 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 розподілених та паралельних обчислень».

Для здобувачів ОС «Бакалавр» спеціальності 122 Комп'ютерні науки факультету інформаційних і прикладних технологій ДонНУ імені Василя Стуса.

УДК 004.75(076.5)

ЗМІСТ

ВСТУП

Крок за кроком технологічний прогрес привів людство до межі можливостей збільшення швидкодії процесорних елементів комп'ютерної техніки. 15–20 років тому частота роботи процесорів досягла значень 3–5 ГГц і «завмерла» на цій позначці. Подальше суттєве зростання частоти і продуктивності процесорів виявилось неможливим із суто технічних причин: швидкість передачі інформації всередині процесора та між процесором і оперативною пам'яттю стала дорівнювати швидкості електричного струму у провіднику. Ця межа, близька до швидкості світла, є фізичним обмеженням нашого всесвіту, яке людство поки що подолати не в змозі.

Тоді людство пішло іншим шляхом. Воно стало збільшувати кількість процесорів в одному комп'ютері. Це дало змогу вирішувати декілька складних задач одночасно, хоча швидкість вирішення кожної окремої задачі все одно залишилась обмеженою швидкодією окремого процесора. Багатопроцесорність, багатоядерність стали нормою як для звичайної комп'ютерної техніки, так і для мобільних пристроїв. Це породило (а точніше, актуалізувало) сферу програмування паралельних та розподілених обчислень.

Майже кожна сучасна мова програмування підтримує засоби розробки паралельних програм. Не є винятком і мова Python, яка широко використовується під час розв'язання прикладних та наукових задач. Навчальний курс «Технології розподілених та паралельних обчислень», що викладається в ДонНУ імені Василя Стуса для спеціальності 122 Комп'ютерні науки, присвячений застосуванню мови Python для розробки програм, основаних на принципі паралельної та розподіленої обробки даних. Методичні рекомендації містять завдання та пояснення до виконання лабораторних робіт за цим навчальним курсом, забезпечують отримання програмних результатів навчання, загальних та спеціальних компетентностей для відповідної освітньої компоненти.

ЛАБОРАТОРНА РОБОТА № 1
ПАРАЛЕЛІЗМ НА ОСНОВІ ПОТОКІВ

Метою роботи є поглиблення у здобувачів освіти практичних навичок використання механізму потоків і модуля *threading* для розпаралелювання програм, написаних мовою Python.

Завдання до лабораторної роботи

Завдання складається з двох незалежних частин.

У першій частині завдання необхідно розпаралелити дії, які не навантажують процесор комп'ютера, однак містять операції введення даних (запити до інтернет-сайтів), які потребують значного часу очікування. Використання механізму потоків для виконання паралельних запитів на сайт повинно демонструвати високу ефективність із погляду економії часу роботи програми.

У другій частині завдання необхідно розпаралелити дії, які потребують значних обчислювальних зусиль з боку процесора. В якості таких дій у лабораторній роботі виступає обробка двовимірних масивів. Треба очікувати, що використання механізму потоків у цьому випадку не буде ефективним і не призведе до виграшу в часі виконання програми.

Завдання 1

Варіант завдання визначається номером здобувача освіти в журналі підгрупи. Відповідно до номера варіанта з таблиці 1 обирається предметна область. Правило вибору таке: здобувачі підгруп А, В, Д обирають рядки таблиці від першого в бік збільшення; здобувачі підгруп Б, Г обирають рядки таблиці від останнього (20-го) в бік зменшення. Наприклад, здобувач підгрупи А під номером 7 обирає з таблиці варіант № 7, здобувач підгрупи Б під номером 5 обирає з таблиці варіант № 16.

Таблиця 1 – Індивідуальні варіанти для завдання 1

№	Предметна область	№	Предметна область
1	Спеціальність «Computer Science» в закордонних вишах	11	НЛО (UFO)
2	Бронювання турів, готелів, купівля квитків	12	Виставки художників, картинні галереї онлайн
3	Театри світового рівня	13	Відомі історичні особи
4	Природа Антарктики	14	Первинна медична допомога в різних ситуаціях
5	Огляди найсучасніших мобільних телефонів	15	Кіберспорт
6	Казки народів світу	16	Україна очима інших країн
7	Приготування кулінарних делікатесів	17	Пандемія COVID-19

№	Предметна область	№	Предметна область
8	Тестування та підготовка до тестування з англійської мови	18	Колективи та виконавці класичної музики
9	Криптовалютні біржі, курси криптовалют тощо	19	Методи паралельного програмування
10	Комп'ютерні ігри жанру «інді»	20	Історія давніх цивілізацій

Для заданої предметної області треба виконати таке.

1.1. Знайти в мережі Інтернет 20 інтернет-сторінок, пов'язаних із заданою предметною областю. Водночас сторінок українських або україномовних сайтів має бути не більше 5. Інші сторінки мають належати закордонним сайтам і бути представлені мовами інших країн (будь-яких). Не дозволяється використовувати російські або російськомовні інтернет-ресурси.

1.2. Переконатися, що всі обрані сторінки коректно відкриваються як у браузері, так і за допомогою бібліотеки *urllib.request* (приклад наведений у лекційному матеріалі). Якщо якась зі сторінок не відкривається, треба замінити цю сторінку на іншу з подібною тематикою.

1.3. Написати програму мовою Python, яка вимірює час звернення до кожної з обраних інтернет-сторінок, а також загальний час послідовного звернення до усіх сторінок. Програма не повинна використовувати ніякі механізми розпаралелювання.

Під зверненням до інтернет-сторінки треба розуміти команди:

```
req = urllib.request.urlopen(s)
pageHtml = req.read()
```

Вимірювання часу здійснювати за допомогою функції *time* модуля *time* (приклад наведений у лекційному матеріалі).

1.4. Запустити програму, розроблену в пункті 1.3, послідовно десять разів підряд. Зафіксувати отримані результати та визначити середній час послідовного звернення до всіх інтернет-сторінок. За бажанням це можна автоматизувати завдяки додаванню в програму відповідних фрагментів коду.

1.5. Модифікувати програму, розроблену в пункті 1.3, так, щоб звернення до інтернет-сторінок здійснювалося в паралельному режимі з використанням механізму потоків (звернення до кожної сторінки – в окремому потоці). У програмі має бути передбачене вимірювання загального часу звернення до усіх сторінок.

1.6. Повторити пункт 1.4 для програми, розробленої в пункті 1.5.

1.7. Порівняти результати, отримані в пунктах 1.4 та 1.6. Зробити висновки щодо отриманих результатів.

Завдання 2

У цьому завданні треба реалізувати алгоритм обробки двовимірного масиву. Оскільки обробка потоків у мові Python реалізується на одному процесорному ядрі, розбиття задачі на окремі потоки не призведе до зменшення часу роботи програми. Навпаки – розбиття задачі на окремі частини, передача даних у потоки та збирання результатів у головному потоці призведуть до збільшення часу роботи, порівняно з однопотоковою версією програми. Однак здобувачам необхідно зробити таке розпаралелювання, щоб на власні очі переконатись у неефективності багатопотокового виконання завдань, які значною мірою залежать від продуктивності процесора.

Заданий двовимірний масив розміром M рядків на N стовпців (розміри масиву задаються на початку програми). Кожним елементом масиву є псевдовипадкове ціле число в діапазоні від –1 000 до 1 000. Необхідно обробити його згідно з індивідуальним варіантом завдання. Варіант завдання обирається з таблиці 2 за тим самим принципом, що й у випадку завдання 1.

Таблиця 2 – Індивідуальні варіанти до завдання 2

№	Варіант завдання
1	У кожному рядку впорядкувати елементи в порядку збільшення кількості цифр у числі. Потім знайти стовпець із найбільшою кількістю непарних чисел (непарність чисел розглядати незалежно від знака числа)
2	Замінити всі значення елементів масиву їх текстовими назвами. Наприклад, значення –728 замінити на текстовий рядок «мінус сімсот двадцять вісім». Потім впорядкувати елементи кожного стовпця в порядку збільшення символьної довжини елементів стовпця
3	Вважаючи елементи масиву значеннями температури в градусах Фаренгейта, перевести їх у градуси Цельсія. Потім знайти рядок масиву, в якому знаходиться найбільша кількість елементів, що відповідають комфортній температурі всередині житла (на власний розсуд)
4	У кожному рядку таблиці впорядкувати елементи за збільшенням їх квадратів (самі елементи не змінювати). Потім у кожному стовпці знайти найбільший та найменший елементи
5	Вважаючи кожен елемент масиву значенням кута, записаним у градусах, замінити його відповідним значенням у радіанах. Наприклад, значення 90 замінюється на 1,570796, тобто на $\pi/2$ радіан. Далі знайти чотири стовпця: – стовпець, у якому найбільша кількість кутів з першої чверті; – стовпець, у якому найбільша кількість кутів з другої чверті; – стовпець, у якому найбільша кількість кутів з третьої чверті; – стовпець, у якому найбільша кількість кутів з четвертої чверті
6	Знайти в масиві найдовшу послідовність нульових елементів, розташованих по горизонталі або по вертикалі
7	Відповідно до таблиці символів ASCII, маленькі латинські літери мають десяткові коди в діапазоні [97; 122]. Вважаючи значення елементів масиву кодами символів, необхідно проглянути рядки і стовпці масиву та знайти усі послідовності символів, які теоретично можуть розглядатись як англомовні слова довжиною не менш ніж K літер, записані маленькими літерами (значення K задається). Враховувати тільки додатні значення у

№	Варіант завдання
	відповідних діапазонах. Наприклад, словом можна вважати послідовність елементів масиву [97, 100, 97, 122] (за умови $K \geq 4$)
8	Знайти в масиві загальну кількість простих чисел. Для тих стовпців, у яких є прості числа, впорядкувати елементи за збільшенням
9	Для кожного елементу масиву розташувати цифри у зворотному порядку (знак числа не змінювати). Наприклад, число –289 буде замінене на число –982. Потім з'ясувати, як внаслідок цих дій змінились сума та добуток усіх елементів масиву
10	Відповідно до таблиці символів ASCII, цифрові символи мають десяткові коди від 48 до 57, знак «мінус» має код 45, знак «плюс» має код 43. Вважаючи значення елементів масиву кодами символів, необхідно проглянути рядки і стовпці масиву та знайти усі послідовності чисел, які можуть вважатись цілими числами зі знаком довжиною не менш ніж K цифр (значення K задається). Число може або мати попереду знак, або не мати його. Наприклад, при $K = 1$ числами можуть вважатись такі послідовності елементів [43, 51, 48, 55], [45, 49, 48], [57, 49, 53, 48, 48, 50], [48, 48, 48, 48, 48], [45, 49], [43, 48], [50]
11	Кожен елемент масиву взяти за модулем. Якщо елемент не є числом Фібоначі, замінити його на найближче до нього число Фібоначі. Потім з'ясувати, як внаслідок цих дій змінилися сума та добуток усіх елементів масиву
12	Знайти рядок масиву, в якому знаходиться найбільша кількість елементів, що є квадратами цілих чисел. Видалити цей рядок з масиву, після чого знайти стовпець, у якому найбільша кількість елементів, що є кубами цілих чисел. Видалити цей стовпець із масиву
13	Елементи кожного рядка впорядкувати так, щоб спочатку були розташовані усі від'ємні елементи рядка, потім нулі, потім додатні елементи (без зміни відносного порядку елементів). Наприклад, рядок з елементами [25, 0, –7, –15, 2, 8, 10, –3, 0, 0, 5] внаслідок впорядкування матиме вигляд [–7, –15, –3, 0, 0, 0, 25, 2, 8, 10, 5]. Потім знайти: – кількість стовпців, які містять тільки від'ємні елементи; – кількість стовпців, які містять тільки нульові елементи; – кількість стовпців, які містять тільки додатні елементи
14	Вважаючи елементи масиву значеннями температури у градусах Кельвіна, перевести їх у градуси Цельсія. Потім знайти рядок масиву, в якому наявна найдовша послідовність елементів, що відповідають комфортній температурі всередині житла (на власний розсуд)
15	Знайти в масиві кількість елементів, у значенні яких усі цифри однакові (знак не враховувати). Потім видалити з масиву рядок із найбільшою кількістю таких елементів. Потім видалити з масиву стовпець із найменшою кількістю таких елементів
16	У кожному рядку масиву впорядкувати елементи так, щоб спочатку розташовувались усі елементи з непарними значеннями, потім нулі, потім усі елементи з парними значеннями (відносний порядок елементів має бути збережений). Наприклад, рядок [–5, 2, 150, 201, 0, 8, –1, 0] буде перетворений на рядок [–5, –201, –1, 0, 0, 2, 150, 8]. Потім у кожному стовпці впорядкувати елементи за збільшенням модулів елементів
17	Вважаючи кожен елемент масиву значенням кута, записаним у градусах, замінити його відповідним значенням у радіанах у діапазоні $[0; 2\pi]$. Потім впорядкувати елементи стовпців масиву в порядку збільшення
18	У кожному елементі масиву записаний грошовий дохід підприємства за один день. Додатне значення означає прибуток, від'ємне – збиток. Розглядаючи рядки масиву незалежно один від іншого, знайти серед усіх рядків найбільш прибутковий тиждень та найбільш збитковий тиждень (послідовність із семи елементів рядка). Потім замінити знаки усіх елементів масиву на протилежні та повторити пошук тижнів. Порівняти результати
19	Кожен елемент масиву перевести в рядковий формат, замінити нульові цифри на випадкові, перевести отриманий рядок у формат цілого числа та записати на своє місце в масиві. Знак числа повинен бути збережений. Перевірити, як внаслідок цього змінилося середнє арифметичне всіх елементів масиву

№	Варіант завдання
20	Кожен елемент масиву перевести у двійковий формат зі знаком, потім розвернути двійкові розряди у зворотному порядку, перевести результат у ціле число (з урахуванням знаку) та записати на своє місце в масиві. Наприклад, число –49 буде виглядати як 1110001_2 (старша одиниця відповідає знаку «мінус»). Після розгортання розрядів матимемо число 1000111_2 (старший розряд виявився також одиницею, тобто число від'ємне). Після переведення у десятковий формат отримаємо число $–000111_2 = –7_{10}$. Потім порахувати, як змінилась загальна кількість додатних та від'ємних елементів масиву

Відповідно до індивідуального варіанта з таблиці 2 необхідно зробити таке:

2.1. Написати програму, в якій задаються значення *M* та *N* (можна задавати константами), створюється список Python розміром *M* * *N* та заповнюється псевдовипадковими числами в діапазоні [–1000; 1000].

2.2. Розв'язати на мові Python поставлену задачу без використання яких-небудь засобів розпаралелювання. Перевірити на невеликих прикладах (на масивах розміром 3 * 3, 4 * 4 тощо) працездатність алгоритму.

2.3. Додати в програму команди для виміру часу обробки масиву. Ці команди повинні вимірювати лише час обробки, без урахування часу на створення та псевдовипадкову генерацію масиву й виведення результатів. Враховуватись повинна тільки безпосередньо обробка масиву.

2.4. Заповнити таблицю 3. У цій таблиці перший стовпець містить різні значення часу виконання програми. Вони задані фіксовано і змінювати їх не можна. Другий стовпець – це розміри масиву, для яких обробка триває стільки часу, скільки вказано в першому стовпці. Здобувач повинен самостійно підібрати розміри масиву, щоб досягти на своєму комп'ютері потрібного часу виконання. Підібрані розміри масиву вказуються у другому стовпці. Бажано, щоб розміри масиву були рівними або близькими один до одного, тобто щоб масив був якомога більш квадратним.

Таблиця 3 – Результати виміру часу роботи програми без розпаралелювання

Час виконання програми (час t_1)	Розміри масиву
1 секунда	
2 секунди	
5 секунд	
10 секунд	
30 секунд	

Наприклад, заповнена таблиця може виглядати так:

Час виконання програми (час t_1)	Розміри масиву
1 секунда	300 * 310
2 секунди	520 * 500
5 секунд	1100 * 1070

Час виконання програми (час t_1)	Розміри масиву
10 секунд	3000 * 3200
30 секунд	5500 * 5300

Значення в другому стовпці таблиці мають бути отримані на особистому ноутбуці здобувача або на комп'ютері в комп'ютерному класі. В будь-якому випадку здобувач під час захисту лабораторної роботи повинен бути готовий продемонструвати роботу програми та отримати результати, близькі до зазначених ним у таблиці.

2.5. Розв'язати на мові Python задачу з табл. 2 з використанням паралелізму на основі потоків. Перевірити на невеликих прикладах (на масивах розміром 3 * 3, 4 * 4 тощо) працездатність алгоритму. Розбиття задачі на обробку окремими потоками здійснюється на розсуд здобувача. Зазвичай у кожній задачі розбиття має виконуватись не один раз (так підібрані завдання). Розроблена програма повинна уникати виникнення перегонів даних та ситуацій типу Deadlock.

2.6. Додати в програму команди для виміру часу обробки масиву. Ці команди повинні вимірювати лише час обробки, без урахування часу на створення та псевдовипадкову генерацію масиву й виведення результатів. Враховуватись повинна тільки безпосередньо обробка масиву.

2.7. Заповнити таблицю 4. У цій таблиці перший стовпець повторює раніше заповнений здобувачем другий стовпець таблиці 3 і відповідає розмірам масиву, для яких відбувається експеримент. У другому стовпці вказується час, витрачений на обробку масиву програмою, розробленою в п. 2.5. Нижче наведений приблизний вигляд таблиці 4 (значення у першому стовпці у кожного здобувача будуть різними).

Таблиця 4 – Результати виміру часу роботи програми з розпаралелюванням на основі потоків

Розміри масиву	Час виконання програми (час t_2), секунд
300 * 310	
520 * 500	
1100 * 1070	
3000 * 3200	
5500 * 5300	

2.8. Порівняти час обробки масиву в табл. 3 (час t_1) з відповідним часом обробки, вказаним у табл. 4 (час t_2). Зробити висновок про ефективність розпаралелювання процесорозалежних програм за допомогою механізму потоків.

Оформлення звіту з лабораторної роботи

Звіт оформлюється на листах формату А4 і повинен містити такі складники:

– титульний лист, оформлений за стандартом Університету;
– індивідуальний варіант для виконання завдань 1 і 2 (береться з таблиць 1 і 2);
– опис і результати виконання завдання 1 (пункти 1.1–1.7);
– опис і результати виконання завдання 2 (пункти 2.1–2.8);
– висновки до лабораторної роботи.

Якщо лабораторна робота здається в режимі «онлайн», до звіту повинні окремо додаватись файли з програмами для 1 і 2 завдань. Замість стандартного розширення «.ру» краще надати цим файлам розширення «.txt».

Якщо лабораторна робота здається в режимі «офлайн», здобувач повинен бути готовий продемонструвати працездатність програм на персональному або університетському комп'ютері.

Приклад виконання лабораторної роботи

Розглянемо приклад виконання лабораторної роботи. Цей приклад не є ідеальним і не обов'язково оцінюється на високу кількість балів.

Завдання 1

Нехай варіантом завдання буде тематика, присвячена кораблям та судно-будуванню.

1.1. Оберемо з мережі Інтернет такі 20 сторінок, присвячених заданій тематиці. Перші п'ять сторінок виберемо з українського інтернет-сегменту, інші – з закордонних сайтів:

1	https://ua.sudohodstvo.org/peredumovy-vidrodzhennya-sudnobuduvannya-ukrayiny/
2	https://esu.com.ua/article-75226
3	https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky/
4	https://tripvenue.com.ua/ukraine/mykolayiv/experience/istoriya-sudnobuduvannya-mista-mykolayiv
5	https://esu.com.ua/article-3451
6	https://www.marineinsight.com/types-of-ships/10-longest-ships-in-the-world-in-2011/
7	https://www.industrytap.com/behemoths-on-the-sea-the-worlds-largest-ships-by-gross-tonnage/51733
8	https://www.ship-technology.com/features/featureoceanic-titans-the-worlds-biggest-ships/?cf-view
9	https://www.marineinsight.com/know-more/biggest-ships-in-the-world/
10	https://nauticalchannel.com/new/biggest-ships
11	https://www.aph.gov.au/Parliamentary_Business/Committees/Senate/Foreign_Affairs_Defence_and_Trade/Completed_inquiries/2004-07/shipping/report/c03
12	https://www.linkedin.com/pulse/jakobstads-history-shipbuilding-seafaring-anders-kurt%C3%A9n
13	https://shipsforcanada.ca/about/history

14	https://depts.washington.edu/chinaciv/miltech/warship.htm
15	http://www.clydewaterfront.com/clyde-heritage/river-clyde/shipbuilding-on-the-clyde
16	https://www.wisconsinhistory.org/Records/Article/CS1822
17	https://www.imm-hamburg.de/museum/ausstellungsrundgang/tour-of-the-exhibitions-english/deck-3-shipbuilding/
18	https://www.cna.org/archive/CNA_Files/pdf/d0006988.a1.pdf
19	http://shipbuildinghistory.com/shipyards/large/newportnews.htm
20	https://www.scafom-rux.com/en/scaffolding-blog/the-beginnings-of-shipbuilding

Помістимо усі наведені адреси в текстовий файл з іменем «pages.txt». Для цього можна використати, наприклад, програму «Блокнот» (Notepad). Вміст файла буде приблизно таким:

```
https://ua.sudohodstvo.org/peredumovy-vidrodzhennya-sudnobuduvannya-ukrayiny/
https://esu.com.ua/article-75226
https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky/
https://tripvenue.com.ua/ukraine/mykolayiv/experience/istoriya-sudnobuduvannya-
mista-mykolayiv
https://esu.com.ua/article-3451
https://www.marineinsight.com/types-of-ships/10-longest-ships-in-the-world-in-2011/
https://www.industrytap.com/behemoths-on-the-sea-the-worlds-largest-ships-by-gross-
tonnage/51733
https://www.ship-technology.com/features/featureoceanic-titans-the-worlds-
biggest-ships/?cf-view
https://www.marineinsight.com/know-more/biggest-ships-in-the-world/
https://nauticalchannel.com/new/biggest-ships
https://www.aph.gov.au/Parliamentary_Business/Committees/Senate/Foreign_Affairs_
Defence_and_Trade/Completed_inquiries/2004-07/shipping/report/c03
https://www.linkedin.com/pulse/jakobstads-history-shipbuilding-seafaring-anders-
kurt%C3%A9n
https://shipsforcanada.ca/about/history
https://depts.washington.edu/chinaciv/miltech/warship.htm
http://www.clydewaterfront.com/clyde-heritage/river-clyde/shipbuilding-on-the-clyde
https://www.wisconsinhistory.org/Records/Article/CS1822
https://www.imm-hamburg.de/museum/ausstellungsrundgang/tour-of-the-exhibitions-
english/deck-3-shipbuilding/
https://www.cna.org/archive/CNA_Files/pdf/d0006988.a1.pdf
http://shipbuildinghistory.com/shipyards/large/newportnews.htm
https://www.scafom-rux.com/en/scaffolding-blog/the-beginnings-of-shipbuilding
```

1.2. Перевіримо можливість програмного доступу до перелічених сторінок. Для цього скористаємось Python-модулем *urllib.request*, застосувавши його функцію *urlopen* (приклад наведений у лекційному матеріалі). Помістимо команди запиту до сайта в окрему функцію *zapit*. Далі прочитаємо вміст файла «pages.txt» у список текстових рядків і будемо в циклі перебирати елементи списку, передаючи кожен елемент у функцію *zapit*. Код програми виглядає так:

```
import urllib.request                    # Підключення модуля

def zapit(s):                            # Функція для інтернет-запитів
    req = urllib.request.urlopen(s)
    pageHtml = req.read()
    print("Запит успішний:", s)          # Якщо запит успішний
```

```
f = open("pages.txt", "rt")
pages_list = f.readlines()              # Читання усього файла в список рядків

for s in pages_list:                    # Цикл за всіма адресами
    s = s.replace("\n", "")             # Видалення з рядка s символа "\n"
    zapit(s)                            # Виконання чергового інтернет-запиту

print("\nDone.")
```

У текстовому файлі кожен рядок містить у кінці керуючий символ «\n» (символ переходу на новий рядок). Хоча цей символ не заважає виконанню інтернет-запиту, він є зайвим і його перед передачею рядка у функцію *zapit* краще прибрати. Для прибирання використано таку команду:

```
    s = s.replace("\n", "")             # Видалення з рядка s символа "\n"
```

Результат роботи програми виглядає приблизно так:

```
Запит успішний: https://ua.sudohodstvo.org/peredumovy-vidrodzhennya-sudnobuduvan
nya-ukrayiny/
Запит успішний: https://esu.com.ua/article-75226
Запит успішний: https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky/
Запит успішний: https://tripvenue.com.ua/ukraine/mykolayiv/experience/istoriya-s
udnobuduvannya-mista-mykolayiv
Запит успішний: https://esu.com.ua/article-3451
Запит успішний: https://www.marineinsight.com/types-of-ships/10-longest-ships-in
-the-world-in-2011/
Запит успішний: https://www.industrytap.com/behemoths-on-the-sea-the-worlds-larg
est-ships-by-gross-tonnage/51733
Запит успішний: https://www.ship-technology.com/features/featureoceanic-titans-t
he-worlds-biggest-ships/?cf-view
Запит успішний: https://www.marineinsight.com/know-more/biggest-ships-in-the-world/
Запит успішний: https://nauticalchannel.com/new/biggest-ships
Запит успішний: https://www.aph.gov.au/Parliamentary_Business/Committees/Senate/
Foreign_Affairs_Defence_and_Trade/Completed_inquiries/2004-07/shipping/report/c03
Запит успішний: https://www.linkedin.com/pulse/jakobstads-history-shipbuilding-s
eafaring-anders-kurt%C3%A9n
Запит успішний: https://shipsforcanada.ca/about/history
Запит успішний: https://depts.washington.edu/chinaciv/miltech/warship.htm
Запит успішний: http://www.clydewaterfront.com/clyde-heritage/river-clyde/shipbu
ilding-on-the-clyde
Запит успішний: https://www.wisconsinhistory.org/Records/Article/CS1822
Запит успішний: https://www.imm-hamburg.de/museum/ausstellungsrundgang/tour-of-t
he-exhibitions-english/deck-3-shipbuilding/
Запит успішний: https://www.cna.org/archive/CNA_Files/pdf/d0006988.a1.pdf
Запит успішний: http://shipbuildinghistory.com/shipyards/large/newportnews.htm
Запит успішний: https://www.scafom-rux.com/en/scaffolding-blog/the-beginnings-of
-shipbuilding

Done.
```

Отже, програмні запити до усіх обраних сайтів виконуються успішно.

1.3. Додамо у програму функції вимірювання часу виконання. Для цього скористаємось функціями модуля *time*. Будемо вимірювати як час виконання кожного окремого запиту (хоча це не обов’язково), так і загальний час виконання усіх

запитів. Лістинг програми з доданими командами, позначеними сірим фоном, наведений нижче.

```
import urllib.request                    # Підключення модулів
import time

def zapit(s):                            # Функція для інтернет-запитів
    t1 = time.time()                     # Початковий час запиту
    req = urllib.request.urlopen(s)
    pageHtml = req.read()
    t2 = time.time()                     # Кінцевий час запиту
    print(t2-t1, "seconds:")             # Виведення часу запиту
    print(s, "\n")

f = open("pages.txt", "rt")
pages_list = f.readlines()               # Читання усього файла в список рядків

t_start = time.time()                    # Початковий час усіх запитів

for s in pages_list:                     # Цикл за всіма адресами
    s = s.replace("\n", "")              # Видалення з рядка s символа "\n"
    zapit(s)                             # Виконання чергового інтернет-запиту

t_end = time.time()                      # Кінцевий час усіх запитів

print("\nDone in", t_end-t_start, "seconds.")
```

Результат роботи програми буде приблизно таким:

```
0.5038979053497314 seconds:
https://ua.sudohodstvo.org/peredumovy-vidrodzhennya-sudnobuduvannya-ukrayiny/

2.7324962615966797 seconds:
https://esu.com.ua/article-75226

3.1244964599609375 seconds:
https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky/

0.6625335216522217 seconds:
https://tripvenue.com.ua/ukraine/mykolayiv/experience/istoriya-sudnobuduvannya-mista-mykolayiv

2.744845390319824 seconds:
https://esu.com.ua/article-3451

1.4349582195281982 seconds:
https://www.marineinsight.com/types-of-ships/10-longest-ships-in-the-world-in-2011/

1.4514775276184082 seconds:
https://www.industrytap.com/behemoths-on-the-sea-the-worlds-largest-ships-by-gross-tonnage/51733

0.39281678199768066 seconds:
https://www.ship-technology.com/features/featureoceanic-titans-the-worlds-biggest-ships/?cf-view

1.3398044109344482 seconds:
https://www.marineinsight.com/know-more/biggest-ships-in-the-world/

2.239494562149048 seconds:
https://nauticalchannel.com/new/biggest-ships
```

```
1.9665281772613525 seconds:
https://www.aph.gov.au/Parliamentary_Business/Committees/Senate/Foreign_Affairs_
Defence_and_Trade/Completed_inquiries/2004-07/shipping/report/c03

1.0551230907440186 seconds:
https://www.linkedin.com/pulse/jakobstads-history-shipbuilding-seafaring-anders-
kurt%C3%A9n

0.9578289985656738 seconds:
https://shipsforcanada.ca/about/history

1.0089399814605713 seconds:
https://depts.washington.edu/chinaciv/miltech/warship.htm

0.37871408462524414 seconds:
http://www.clydewaterfront.com/clyde-heritage/river-clyde/shipbuilding-on-the-clyde

1.021955966949463 seconds:
https://www.wisconsinhistory.org/Records/Article/CS1822
1.0857946872711182 seconds:
https://www.imm-hamburg.de/museum/ausstellungsrundgang/tour-of-the-exhibitions-e
nglish/deck-3-shipbuilding/

0.28276515007019043 seconds:
https://www.cna.org/archive/CNA_Files/pdf/d0006988.a1.pdf

1.515312671661377 seconds:
http://shipbuildinghistory.com/shipyards/large/newportnews.htm

0.6164994239807129 seconds:
https://www.scafom-rux.com/en/scaffolding-blog/the-beginnings-of-shipbuilding

Done in 27.096880197525024 seconds.
```

1.4. Цей пункт можна реалізувати різними способами. Єдина бажана вимога полягає в тому, щоб не виконувати запити до однієї і тієї ж сторінки кілька разів підряд. Найкращим підходом буде виконати запити до усіх 20 сторінок від першої до останньої, після чого повторити цей процес ще 10 разів. Тоді запити до однієї і тієї ж сторінки будуть здійснюватися з певним інтервалом.

Розглянемо модифіковану програму.

Початок програми (до читання рядків з файла включно) виглядає майже так, як і в попередньому прикладі. Єдина різниця – функція *zapit* тепер повертає час виконання запиту:

```
import urllib.request                  # Підключення модулів
import time

def zapit(s):                          # Функція для виконання інтернет-запитів
    t1 = time.time()                   # Початковий час запиту
    req = urllib.request.urlopen(s)
    pageHtml = req.read()
    t2 = time.time()                   # Кінцевий час запиту
    return (t2-t1)

f = open("pages.txt", "rt")            # Відкриття файла для читання
pages_list = f.readlines()             # Читання усього файла в список рядків
```

Далі ми визначаємо кількість інтернет-адрес, прочитаних з файла. Для цього застосовуємо стандартну функцію *len* до списку адрес:

```
n = len(pages_list)                    # Кількість інтернет-адрес у файлі
```

Тепер створимо змінні, в яких будемо накопичувати значення часу звернення у вигляді суми. Це необхідно для подальшого розрахунку середнього арифметичного. Нам потрібен список із *n* змінних для зберігання часу звернення до кожної з *n* інтернет-адрес, а також ще одна окрема змінна для зберігання суми загального часу доступу до усіх *n* сторінок.

```
n = len(pages_list)                    # Кількість інтернет-сторінок у файлі

times_list = [0]*n                     # Список для зберігання часу виконання
                                       # запиту до кожної з інтернет-сторінок

global_time = 0                        # Сумарний час виконання усіх запитів
```

Далі організуємо цикл із $k = 10$ повторів. У кожному повторі циклу будемо по черзі звертатись за всіма адресами, отримувати час звертання та додавати його до відповідної комірки списку *times_list*. У кінці кожного повтору будемо додавати загальний час звертання до змінної *global_time*.

```
k = 10                                 # Кількість повторів експерименту
for i in range(0, k):                  # Головний цикл на k повторів
    t_start = time.time()              # Початковий час усіх запитів

    j = 0
    for s in pages_list:               # Цикл за всіма адресами
        s = s.replace("\n", "")        # Видалення з рядка s символа "\n"
        t = zapit(s)                   # Виконання чергового інтернет-запиту
        times_list[j] += t             # Враховуємо час чергового запиту
        j = j + 1

    t_end = time.time()                # Кінцевий час усіх запитів

    global_time += t_end - t_start

    print("\n", i, ": Done in", t_end-t_start, "seconds.\n\n")
```

Після завершення головного циклу нам залишається розрахувати і вивести на екран середнє арифметичне часу звернення до кожного із сайтів та середнє арифметичне загального часу звернення до усіх сайтів. Для цього ми ділимо суму усіх значень часу на кількість експериментів *k*.

```
for j in range(0, n):
    times_list[j] /= k       # Середнє арифметичне часу для окремих сторінок
    print(times_list[j], "seconds:", pages_list[j])

global_time /= k             # Середнє арифметичне загального часу
print("Середній загальний час усіх запитів:", global_time)
```

Загальний лістинг програми виглядає так:

```
import urllib.request                  # Підключення модулів
import time

def zapit(s):                          # Функція для виконання інтернет-запитів
    t1 = time.time()                   # Початковий час запиту
    req = urllib.request.urlopen(s)
    pageHtml = req.read()
    t2 = time.time()                   # Кінцевий час запиту
    return (t2-t1)

f = open("pages.txt", "rt")            # Відкриття файла для читання
pages_list = f.readlines()             # Читання усього файла в список рядків

n = len(pages_list)                    # Кількість інтернет-адрес у файлі

times_list = [0]*n                     # Список для зберігання часу виконання
                                       # запиту до кожної з інтернет-сторінок

global_time = 0                        # Сумарний час виконання усіх запитів

k = 10                                   # Кількість повторів експерименту
for i in range(0, k):                    # Головний цикл на k повторів
    t_start = time.time()                # Початковий час усіх запитів

    j = 0
    for s in pages_list:                 # Цикл за всіма адресами
        s = s.replace("\n", "")          # Видалення з рядка s символу "\n"
        t = zapit(s)                     # Виконання чергового інтернет-запиту
        times_list[j] += t               # Враховуємо час чергового запиту
        j = j + 1

    t_end = time.time()                  # Кінцевий час усіх запитів

    global_time += t_end - t_start

for j in range(0, n):
    times_list[j] /= k       # Середнє арифметичне часу для окремих сторінок
    print(times_list[j], "seconds:", pages_list[j])

global_time /= k             # Середнє арифметичне загального часу
print("Середній загальний час усіх запитів:", global_time)
```

Результат роботи програми буде приблизно таким:

```
0.49303059577941893 seconds: https://ua.sudohodstvo.org/peredumovy-vidrodzhennya
-sudnobuduvannya-ukrayiny/

2.721459984779358 seconds: https://esu.com.ua/article-75226

4.825270104408264 seconds: https://uprom.info/news/ships/ukrayinske-sudnobuduvan
nya-pidsumky/

0.5797169208526611 seconds: https://tripvenue.com.ua/ukraine/mykolayiv/experienc
e/istoriya-sudnobuduvannya-mista-mykolayiv

2.846228313446045 seconds: https://esu.com.ua/article-3451

1.3038681507110597 seconds: https://www.marineinsight.com/types-of-ships/10-long
est-ships-in-the-world-in-2011/
```

```
1.5428482294082642 seconds: https://www.industrytap.com/behemoths-on-the-sea-the
-worlds-largest-ships-by-gross-tonnage/51733

0.3773672580718994 seconds: https://www.ship-technology.com/features/featureocea
nic-titans-the-worlds-biggest-ships/?cf-view

1.3601145505905152 seconds: https://www.marineinsight.com/know-more/biggest-ship
s-in-the-world/

2.378354287147522 seconds: https://nauticalchannel.com/new/biggest-ships

1.9599764585494994 seconds: https://www.aph.gov.au/Parliamentary_Business/Commit
tees/Senate/Foreign_Affairs_Defence_and_Trade/Completed_inquiries/2004-07/shippi
ng/report/c03

1.0279681205749511 seconds: https://www.linkedin.com/pulse/jakobstads-history-sh
ipbuilding-seafaring-anders-kurt%C3%A9n

0.9518182754516602 seconds: https://shipsforcanada.ca/about/history

0.8950380563735962 seconds: https://depts.washington.edu/chinaciv/miltech/warship.htm

0.389139461517334 seconds: http://www.clydewaterfront.com/clyde-heritage/river-c
lyde/shipbuilding-on-the-clyde

0.9581798791885376 seconds: https://www.wisconsinhistory.org/Records/Article/CS1822

0.7338152170181275 seconds: https://www.imm-hamburg.de/museum/ausstellungsrundga
ng/tour-of-the-exhibitions-english/deck-3-shipbuilding/

0.27255997657775877 seconds: https://www.cna.org/archive/CNA_Files/pdf/d0006988.
a1.pdf

1.0854928016662597 seconds: http://shipbuildinghistory.com/shipyards/large/newpo
rtnews.htm

0.5607535123825074 seconds: https://www.scafom-rux.com/en/scaffolding-blog/the-b
eginnings-of-shipbuilding

Середній загальний час усіх запитів: 27.26555953025818
```

1.5. Модифікуємо наведену вище програму так, щоб задіяти механізм потоків. Оскільки інтернет-запитів у нас відносно небагато, домовимось виконувати кожний запит в окремому потоці.

Внесемо в програму, розглянуту в пункті 1.4, такі зміни.

Під'єднаємо модуль *threading,* що відповідає за використання потоків:

```
import urllib.request
import time
import threading
```

У якості функції, яка буде запускатись у кожному потоці, використаємо функцію *zapit*. Оскільки ми вже бачили, скільки сягає середній час запиту до кожного сайта, будемо вимірювати тільки середній час звертання до усіх 20 інтернет-сторінок. Це дасть змогу прибрати з функції *zapit* усі команди, що стосуються виміру часу:

```
def zapit(s):                          # Функція для виконання інтернет-запитів
    req = urllib.request.urlopen(s)
    pageHtml = req.read()
```

Відповідно нам не знадобиться список *times_list*. Однак змінна *global_time* буде потрібна, оскільки вона відповідає за вимір загального часу виконання усіх 20 запитів:

```
f = open("pages.txt", "rt")            # Відкриття файла для читання
pages_list = f.readlines()             # Читання усього файла в список рядків

n = len(pages_list)                    # Кількість інтернет-адрес у файлі

global_time = 0                        # Сумарний час виконання усіх запитів
```

Далі нам необхідно організувати цикл із $k = 10$ повторів, в тілі якого будуть створюватись і працювати 20 потоків. Для зберігання потоків зручно використовувати масив потоків, оскільки це дає змогу застосовувати цикли та скорочувати програмний код.

У наведеному нижче фрагменті коду зовнішній цикл повторюється *k* разів. У тілі циклу виконуються такі дії:

– запам’ятовування початкового часу роботи;

– створення списку потоків на основі функції *zapit,* у кожен з яких передається своя інтернет-адреса;

– послідовний запуск кожного потоку;

– очікування завершення кожного потоку;

– запам’ятовування кінцевого часу роботи;

– додавання визначеного часу роботи до змінної *global_time*;

– виведення на екран інформації про час виконання чергового проходу циклу.

```
k = 10                                 # Кількість повторів експерименту
for i in range(0, k):                  # Головний цикл на k повторів

    t_start = time.time()              # Початковий час усіх запитів

    t_list = []                        # Підготовка масиву потоків

    for s in pages_list:               # Цикл за всіма інтернет-адресами
        s = s.replace("\n", "")        # Видалення \n
        t = threading.Thread(target=zapit, args=(s, )) # Створення потоку
        t_list.append(t)                               # Додавання в список

    for t in t_list:                   # Цикл за запуском потоків
        t.start()

    for t in t_list:                   # Цикл за очікуванням потоків
        t.join()

    t_end = time.time()                # Кінцевий час усіх запитів

    global_time += t_end - t_start
    print("Прохід", i, ": час виконання =", t_end-t_start, "секунд.")
```

Тепер нам залишилось тільки розрахувати і вивести на екран середнє значення виконання усіх 20 запитів:

```
global_time /= k              # Середнє арифметичне загального часу
print("Середній загальний час усіх запитів:", global_time)
```

Загальний лістинг програми виглядає так:

```
import urllib.request
import time
import threading

def zapit(s):                           # Функція для виконання інтернет-запитів
    req = urllib.request.urlopen(s)
    pageHtml = req.read()

f = open("pages.txt", "rt")             # Відкриття файла для читання
pages_list = f.readlines()              # Читання усього файла в список рядків

n = len(pages_list)                     # Кількість інтернет-адрес у файлі

global_time = 0                         # Сумарний час виконання усіх запитів


k = 10                                    # Кількість повторів експерименту
for i in range(0, k):                     # Головний цикл на k повторів

    t_start = time.time()                 # Початковий час усіх запитів

    t_list = []                           # Підготовка масиву потоків

    for s in pages_list:                  # Цикл за всіма інтернет-адресами
        s = s.replace("\n", "")           # Видалення \n
        t = threading.Thread(target=zapit, args=(s, )) # Створення потоку
        t_list.append(t)                                # Додавання в список

    for t in t_list:                      # Цикл за запуском потоків
        t.start()

    for t in t_list:                      # Цикл за очікуванням потоків
        t.join()

    t_end = time.time()                   # Кінцевий час усіх запитів

    global_time += t_end - t_start
    print("Прохід", i, ": час виконання =", t_end-t_start, "секунд.")

global_time /= k              # Середнє арифметичне загального часу
print("Середній загальний час усіх запитів:", global_time)
```

1.6. Запустивши програму, отримаємо приблизно такий результат:

```
Прохід 0 : час виконання = 2.9140753746032715 секунд.
Прохід 1 : час виконання = 2.8192250728607178 секунд.
Прохід 2 : час виконання = 2.9296538829803467 секунд.
Прохід 3 : час виконання = 2.876713275909424 секунд.
Прохід 4 : час виконання = 3.0511977672576904 секунд.
Прохід 5 : час виконання = 16.150056838989258 секунд.
```

```
Прохід 6 : час виконання = 9.999681234359741 секунд.
Прохід 7 : час виконання = 14.279959440231323 секунд.
Прохід 8 : час виконання = 13.739457845687866 секунд.
Прохід 9 : час виконання = 9.957761764526367 секунд.
Середній загальний час усіх запитів: 7.871778249740601
```

Як можна бачити, в одних випадках загальний час паралельних запитів до усіх 20 сайтів сягає 2,8 секунди, а в деяких випадках може сягати 16 секунд. У нашому випадку це значення приблизно дорівнює часу доступу до найбільш «повільної» інтернет-сторінки (конкретно – сторінки https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky).

Декілька послідовних запусків програми показують, що час доступу до сайтів може суттєво відрізнятись від запуску до запуску. Наприклад:

```
Прохід 0 : час виконання = 2.908026695251465 секунд.
Прохід 1 : час виконання = 2.883260488510132 секунд.
Прохід 2 : час виконання = 2.8843600749969482 секунд.
Прохід 3 : час виконання = 3.0934462547302246 секунд.
Прохід 4 : час виконання = 3.1569645404815674 секунд.
Прохід 5 : час виконання = 2.8481059074401855 секунд.
Прохід 6 : час виконання = 2.9307777881622314 секунд.
Прохід 7 : час виконання = 2.912468910217285 секунд.
Прохід 8 : час виконання = 2.7909111976623535 секунд.
Прохід 9 : час виконання = 2.8315846920013428 секунд.
Середній загальний час усіх запитів: 2.9239906549453734
```

```
Прохід 0 : час виконання = 3.381065607070923 секунд.
Прохід 1 : час виконання = 3.304666042327881 секунд.
Прохід 2 : час виконання = 31.449093341827393 секунд.
Прохід 3 : час виконання = 4.007504224777222 секунд.
Прохід 4 : час виконання = 5.2810139656066895 секунд.
Прохід 5 : час виконання = 32.35204005241394 секунд.
Прохід 6 : час виконання = 4.025395631790161 секунд.
Прохід 7 : час виконання = 4.277862787246704 секунд.
Прохід 8 : час виконання = 28.344917058944702 секунд.
Прохід 9 : час виконання = 3.3843703269958496 секунд.
Середній загальний час усіх запитів: 11.980792903900147
```

```
Прохід 0 : час виконання = 4.31280779838562 секунд.
Прохід 1 : час виконання = 2.817047595977783 секунд.
Прохід 2 : час виконання = 2.9561069011688232 секунд.
Прохід 3 : час виконання = 2.8775148391723633 секунд.
Прохід 4 : час виконання = 3.03558349609375 секунд.
Прохід 5 : час виконання = 3.662721872329712 секунд.
Прохід 6 : час виконання = 16.27401852607727 секунд.
Прохід 7 : час виконання = 6.2562150955200195 секунд.
Прохід 8 : час виконання = 2.8437039852142334 секунд.
Прохід 9 : час виконання = 2.886451482772827 секунд.
Середній загальний час усіх запитів: 4.792217159271241
```

1.7. Результати експериментальних досліджень, отримані в пунктах 1.4 та 1.6, показують, що середній час паралельних інтернет-запитів до усіх 20 сторінок у разі використання багатопотокового підходу є у кілька разів меншим, ніж у

випадку послідовного виконання інтернет-запитів. Це свідчить про те, що в цьому випадку розпаралелювання на основі потоків дає змогу отримати економію часу виконання програми.

Завдання 2

Заданий двовимірний масив розміром *M* рядків на *N* стовпців (розміри масиву задаються на початку програми). Кожним елементом масиву є псевдовипадкове ціле число в діапазоні від –1 000 до 1 000. Знайти кількість елементів масиву, сума цифр яких, незалежно від знаку числа, дорівнює 10 (наприклад, 91, –253, 730, –28, 55). Замінити на це число усі елементи масиву, які є квадратами цілих чисел.

2.1. Задамо розміри масиву у вигляді констант. Створимо двовимірний масив розміром *M* * *N*, заповнений псевдовипадковими цілими числами в діапазоні [–1000, 1000].

```
import random

M, N = 3, 4

mas = []                                  # Підготовка порожнього масиву рядків
for i in range(0, M):                     # Цикл за рядками
    mas_i = []                            # Підготовка порожнього рядка
    for j in range(0, N):                 # Цикл за стовпцями
        n = random.randint(-1000, 1000)   # Генерація рандомного числа
        mas_i.append(n)                   # Додавання числа в рядок
    mas.append(mas_i)                     # Додавання рядка у двовимірний масив
```

Додавання в кінці цього коду команди виведення масиву на екран показує такий результат:

```
[[-1000, 348, -145, 22], [-428, -919, -902, -67], [-151, -732, -663, 997]]
```

Отже, генерація масиву працює коректно.

2.2. Розв’яжемо поставлену задачу без використання розпаралелювання. Для цього спочатку знайдемо кількість елементів масиву, сума цифр яких, незалежно від знака числа, дорівнює 10.

Для зберігання кількості знайдених елементів будемо використовувати змінну *n*10:

```
n10 = 0
```

Далі організуємо вкладений цикл, всередині якого будемо брати з масиву черговий елемент та рахувати суму його цифр. Програмний код може виглядати приблизно так:

```
for i in range(0, M):                     # Цикл за рядками
    for j in range(0, N):                 # Цикл за стовпцями
```

```
        n = mas[i][j]                   # Беремо елемент масиву
        s = str(n)                      # Переводимо в символьний формат
        s = s.replace("-", "")          # Видаляємо знак "мінус", якщо він є

        m = 0                           # Початкове значення суми цифр
        for c in s:                     # Перебирання символів рядка
            m += int(c)                 # Додаємо цифру до суми

        if m == 10:                     # Якщо сума цифр дорівнює 10
            n10 = n10 + 1               # Збільшуємо кількість
            print(n)

print("Знайдено чисел:", n10)
```

У цьому фрагменті команда:

```
            print(n)
```

додана для перевірки працездатності програми і виводить на екран усі числа, сума цифр яких дорівнює 10. Ця команда потрібна лише на етапі тестування правильності роботи програми. Далі, особливо під час роботи з масивами великого розміру, команду треба прибрати, оскільки виведення інформації на екран може займати значний відсоток часу роботи програми.

Результат роботи програми для масиву розміром 10 * 10 виглядає приблизно так:

```
640
-118
226
-82
-910
-28
-172
73
Знайдено чисел: 8
```

Отже, можна бачити, що програма працює правильно.

Тепер розв’яжемо другу частину задачі: замінимо усі елементи масиву, що є квадратами цілих чисел, на знайдене число *n*10. Для цього будемо проглядати в циклі усі елементи масиву та перевіряти, чи є черговий елемент додатним і чи є корінь із нього цілим числом. Для цього використаємо такий фрагмент програмного коду:

```
for i in range(0, M):                   # Цикл за рядками
    for j in range(0, N):               # Цикл за стовпцями
        n = mas[i][j]                   # Беремо елемент масиву

        if n > 0:                           # Якщо число додатне
            if n ** 0.5 == int(n ** 0.5):   # Якщо корінь є цілим числом
                mas[i][j] = n10             # Робимо заміну елемента на n10
                print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")
```

Як і в попередньому фрагменті, остання команда *print* потрібна лише на етапі налагодження програми. Якщо програма працює правильно, команду *print* бажано прибрати, щоб вона не вносила погрішність у вимірювання часу виконання програми (це буде зроблено в наступному пункті).

Загальний лістинг програми виглядає так, як показано нижче. В кінці цього лістингу можна було б додати команди виведення усього обробленого масиву на екран, однак виведення на екран масиву великих розмірів було б неінформативним та займало б багато часу.

```
import random

M, N = 10, 10

mas = []                                # Підготовка порожнього масиву рядків
for i in range(0, M):                   # Цикл за рядками
    mas_i = []                          # Підготовка порожнього рядка
    for j in range(0, N):               # Цикл за стовпцями
        n = random.randint(-1000, 1000) # Генерація рандомного числа
        mas_i.append(n)                 # Додавання числа в рядок
    mas.append(mas_i)                   # Додавання рядка у двовимірний масив

n10 = 0                                 # Кількість знайдених елементів

for i in range(0, M):                   # Цикл за рядками
    for j in range(0, N):               # Цикл за стовпцями
        n = mas[i][j]                   # Беремо елемент масиву
        s = str(n)                      # Переводимо в символьний формат
        s = s.replace("-", "")          # Видаляємо знак "мінус", якщо він є

        m = 0                           # Початкове значення суми цифр
        for c in s:                     # Перебирання символів рядка
            m += int(c)                 # Додаємо цифру до суми

        if m == 10:                     # Якщо сума цифр дорівнює 10
            n10 = n10 + 1               # Збільшуємо кількість
            print(n)

print("Знайдено чисел:", n10)

for i in range(0, M):                   # Цикл за рядками
    for j in range(0, N):               # Цикл за стовпцями
        n = mas[i][j]                   # Беремо елемент масиву

        if n > 0:                           # Якщо число додатне
            if n ** 0.5 == int(n ** 0.5):   # Якщо корінь є цілим числом
                mas[i][j] = n10             # Робимо заміну елемента на n10
                print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")
```

Задавши розміри масиву рівними $M = 10$, $N = 10$ і запустивши програму, ми отримаємо приблизно такий результат:

```
244
370
28
-442
-703
```

```
Знайдено чисел: 5
[3][6] = 256 -> 5
[4][5] = 784 -> 5
[7][3] = 9 -> 5
```

Як можна бачити, в цьому випадку було знайдено 5 чисел, сума цифр яких дорівнює 10. Потім було знайдено три числа, що є квадратами цілих чисел: 256, 784, 9. Ці числа були замінені на число 5. Хоча наша програма не виводить на екран кінцевий вміст масиву, наступна команда:

```
                mas[i][j] = n10              # Робимо заміну елементу на n10
```

не викликає сумнівів, що усі необхідні зміни в масиві дійсно зроблені.

Треба також зазначити, що масив розміром 10 * 10 не є достатньо великим, щоб у ньому зустрічалось багато псевдовипадкових чисел, що є квадратами цілих чисел. У наведеному вище прикладі таких чисел лише три. Якщо запустити програму ще декілька разів, можна побачити такі результати:

```
73
325
-424
28
Знайдено чисел: 4
[2][2] = 169 -> 4
```

```
-514
-172
-442
451
-217
19
82
307
Знайдено чисел: 8
[7][8] = 25 -> 8
[8][2] = 169 -> 8
```

```
-235
415
19
55
181
-316
Знайдено чисел: 6
```

В останньому прикладі квадрати цілих чисел взагалі не були знайдені, і це цілком можлива ситуація для масиву невеликого розміру. Якщо задати розміри масиву 20 * 20, знайдених квадратів буде більше:

```
-208
-442
-703
```

```
271
514
550
712
190
640
-244
154
-280
-622
-613
-442
820
424
541
-343
-523
46
-550
91
-73
Знайдено чисел: 24
[5][10] = 36 -> 24
[10][11] = 576 -> 24
[11][13] = 256 -> 24
[11][19] = 1 -> 24
[14][13] = 729 -> 24
[17][9] = 121 -> 24
[19][12] = 529 -> 24
```

2.3. Додамо в програму необхідні команди для виміру часу її роботи. Для цього зробимо таке:

– приєднаємо модуль *time*;

– після команд, що створюють і заповнюють масив, додамо команду запам’ятовування поточного часу у змінну *t*1;

– в самому кінці програми додамо команди запам’ятовування поточного часу в змінну *t*2 і виведення на екран різниці (*t*2 – *t*1);

– команди *print*, що розташовані всередині циклів, приберемо (закоментуємо).

Лістинг програми наведений нижче. Додані та закоментовані команди виділені сірим фоном.

```
import random
import time

M, N = 10, 10

mas = []                                # Підготовка порожнього масиву рядків
for i in range(0, M):                   # Цикл за рядками
    mas_i = []                          # Підготовка порожнього рядка
    for j in range(0, N):               # Цикл за стовпцями
        n = random.randint(-1000, 1000) # Генерація рандомного числа
        mas_i.append(n)                 # Додавання числа в рядок
    mas.append(mas_i)                   # Додавання рядка у двовимірний масив

t1 = time.time()                        # Початковий час

n10 = 0                                 # Кількість знайдених елементів
```

```
for i in range(0, M):                    # Цикл за рядками
    for j in range(0, N):                # Цикл за стовпцями
        n = mas[i][j]                    # Беремо елемент масиву
        s = str(n)                       # Переводимо в символьний формат
        s = s.replace("-", "")           # Видаляємо знак "мінус", якщо він є

        m = 0                            # Початкове значення суми цифр
        for c in s:                      # Перебирання символів рядка
            m += int(c)                  # Додаємо цифру до суми

        if m == 10:                      # Якщо сума цифр дорівнює 10
            n10 = n10 + 1                # Збільшуємо кількість
            #print(n)

print("Знайдено чисел:", n10)

for i in range(0, M):                    # Цикл за рядками
    for j in range(0, N):                # Цикл за стовпцями
        n = mas[i][j]                    # Беремо елемент масиву

        if n > 0:                           # Якщо число додатне
            if n ** 0.5 == int(n ** 0.5):   # Якщо корінь є цілим числом
                mas[i][j] = n10             # Робимо заміну елементу на n10
                #print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")

t2 = time.time()                         # Кінцевий час
print("Час роботи програми:", t2-t1, "секунд.")
```

2.4. Заповнена таблиця 3 наведена нижче. Дослідження виконувались на комп’ютері з процесором i5-13500 під управлінням Windows 11, версія Python 3.12.0. За розмірів масиву, вказаних у другому стовпці, час виконання обробки масиву був найбільш близьким до значення, вказаного в першому стовпці таблиці (за результатами декількох запусків програми).

Таблиця 3 – Результати виміру часу роботи програми без розпаралелювання

Час виконання програми (час t_1)	**Розміри масиву**
1 секунда	1230 * 1230
2 секунди	1770 * 1770
5 секунд	2800 * 2800
10 секунд	3950 * 3950
30 секунд	6850 * 6850

2.5. Цей пункт лабораторної роботи є найбільш складним і потребує детального обговорення.

Під час переробки звичайної непаралельної програми в паралельну завжди виникають два основні питання:

1. Які етапи розв’язання задачі можуть виконуватись паралельно?

2. Як краще розподілити роботу, що може виконуватись паралельно, між кількома дочірніми потоками або процесами?

Розглянемо перше питання.

Розв'язувана задача складається з двох частин. У першій частині ми шукаємо кількість елементів масиву, сума цифр яких дорівнює 10. У другій частині ми шукаємо і замінюємо елементи, що є квадратами цілих чисел.

У кожній частині задачі ми працюємо з усіма елементами масиву, причому порядок перегляду елементів масиву не має значення. Також неважливо, які розміри в оброблюваного масиву. Масив може бути квадратним, прямокутним і навіть одновимірним (одновимірний масив є окремим випадком двовимірного). Форма масиву у нашому випадку не має значення.

Це дає нам можливість задіяти для пошуку елементів декілька дочірніх потоків. Кожному потоку можна надати якусь частину масиву, і потік буде розв'язувати задачу тільки для цієї частини. Коли потік завершить роботу, він передасть головному потоку результати своєї роботи і завершиться. Головний потік може швидко обробити результати роботи дочірніх потоків.

Розглянемо друге питання.

У першій частині задачі ми шукаємо кількість елементів масиву, сума цифр яких дорівнює 10. Ми можемо розділити цю задачу на підзадачі, доручивши кожну підзадачу окремому потоку. Підзадачі можуть бути різними за складністю і обсягом роботи. Наприклад:

1. Кожен дочірній потік виконує обробку тільки одного рядка двовимірного масиву. Результатом роботи є кількість потрібних нам елементів, які містяться в цьому рядку масиву. Наприкінці роботи дочірнього потоку знайдена кількість додається до глобальної змінної, в якій ведеться накопичення результатів роботи кожного потоку. У цьому випадку нам потрібна кількість потоків, рівна кількості рядків таблиці.

2. Кожен дочірній потік виконує обробку якогось діапазону рядків таблиці. Результатом роботи є кількість потрібних нам елементів, що містяться в цих рядках таблиці. Наприкінці роботи потоку знайдена кількість додається до глобальної змінної. У цьому випадку нам потрібна менша кількість дочірніх потоків, ніж у першому випадку.

3. Кожен дочірній потік обробляє лише один елемент масиву. Результатом обробки буде значення 1, якщо сума цифр елементу дорівнює десяти, або 0, якщо не дорівнює. Це значення додається до глобальної змінної, в якій ведеться накопичення результатів роботи кожного потоку. У цьому випадку нам потрібна кількість дочірніх потоків, рівна кількості елементів масиву.

Звісно, в нашій задачі можна обробляти масив не за окремими рядками, а за окремими стовпцями або якось інакше. Усі ці варіанти є різновидами варіанта № 2.

Проаналізуємо три розглянуті варіанти розподілу роботи між дочірніми потоками.

Перший варіант реалізується найбільш просто, однак у разі великої кількості рядків масиву виникне необхідність у створенні великої кількості потоків. Така

кількість може уповільнювати роботу програми та вимагати значних обсягів пам'яті для зберігання об'єктів потоків.

Другий варіант дає змогу користуватись меншою кількістю потоків, однак потребує використання окремого алгоритму розподілу діапазонів рядків між потоками. До того ж кожному потоку виділяється більший обсяг роботи, що збільшує час роботи кожного потоку. Це може бути особливо чутливим не у випадку потоків (у мові Python усі потоки виконуються на одному процесорному ядрі), а у випадку процесів. Робота з процесами розглядається у наступній лабораторній роботі.

Третій варіант є найменш привабливим, оскільки потребує дуже багато потоків, тоді як робота, що надається кожному потоку, є дуже простою. У разі кількості потоків, рівній кількості елементів масиву, 99.9 % часу буде витрачатись на створення потоків, перемикання між ними та очікування завершення їх роботи.

У випадку нашого завдання оберемо перший варіант. Щоб знайти кількість елементів масиву, які мають суму цифр, рівну 10, зробимо таке.

Залишимо команди створення і заповнення масиву такими, як у попередніх прикладах. На початку програми під'єднаємо модуль *threading*.

```
import random
import threading

M, N = 10, 10                              # Розміри масиву

mas = []                                   # Підготовка порожнього масиву рядків
for i in range(0, M):                      # Цикл за рядками
    mas_i = []                             # Підготовка порожнього рядка
    for j in range(0, N):                  # Цикл за стовпцями
        n = random.randint(-1000, 1000)    # Генерація рандомного числа
        mas_i.append(n)                    # Додавання числа в рядок
    mas.append(mas_i)                      # Додавання рядка у двовимірний масив
```

Додамо в програму глобальний замок типу *threading.Lock*. Він буде використовуватись для монопольного використання потоками команд *print* і доступу до глобальних змінних з метою упередження станів перегонів.

```
L = threading.Lock()                       # Замок
```

Створимо змінну *n*10, у якій буде накопичуватись кількість знайдених елементів масиву.

```
n10 = 0                                    # Кількість знайдених елементів
```

Напишемо функцію *f*1, яка приймає номер рядка двовимірного масиву, виконує в межах цього рядка пошук кількості елементів, сума цифр яких дорівнює 10, та додає цю кількість до глобальної змінної *f*1.

```
def f1(i):                                  # Пошук у рядку з номером i
    global n10                              # Будемо змінювати глобальну змінну

    k = 0                                   # Кількість знайдених ел-тів у рядку

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву
        s = str(n)                          # Переводимо в символьний формат
        s = s.replace("-", "")              # Видаляємо знак "мінус", якщо він є

        m = 0                               # Початкове значення суми цифр
        for c in s:                         # Перебирання символів рядка
            m += int(c)                     # Додаємо цифру до суми

        if m == 10:                         # Якщо сума цифр дорівнює 10
            k = k + 1                       # Збільшуємо кількість
            L.acquire()
            print(n)
            L.release()

    L.acquire()
    n10 += k                                # Додаємо результат до глоб. змінної
    L.release()
```

У цій програмі ми використовуємо замок *L* під час виконання команди *print* і під час додавання значення *k* до глобальної змінної *n*10. Можливо, в нашому конкретному випадку можна було б обійтись без замка, але використання замка є правильним підходом, якого треба дотримуватись.

Створимо порожній список потоків, після чого в циклі створимо *M* дочірніх потоків (*M* – кількість рядків масиву). Кожен потік будемо створювати на базі функції *f*1 і передавати до неї номер рядка масиву.

```
t_list = []                                 # Підготовка списку потоків

for i in range(0, M):                       # Цикл за створенням потоків
    t = threading.Thread(target=f1, args=(i, ))
    t_list.append(t)
```

Далі виконаємо два стандартні цикли: цикл запуску потоків та цикл очікування потоків:

```
for t in t_list:                            # Цикл із запуску потоків
    t.start()

for t in t_list:                            # Цикл за очікуванням потоків
    t.join()
```

Після того, як робота усіх потоків буде завершена, у змінній *n*10 буде знаходитись кількість елементів масиву, сума цифр яких дорівнює десяти. Виведемо значення змінної *n*10 на екран:

```
print("Знайдено чисел:", n10)
```

Отже, ми завершили написання першої частини нашої «паралельної» програми. Згідно з «правилами хорошого тону» переставимо опис функції *f*1 на початок програми після підключення модулів. Загальний лістинг першої частини програми наведений нижче.

```
import random
import threading

def f1(i):                                  # Пошук у рядку з номером i
    global n10                              # Будемо змінювати глобальну змінну

    k = 0                                   # Кількість знайдених ел-тів у рядку

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву
        s = str(n)                          # Переводимо в символьний формат
        s = s.replace("-", "")              # Видаляємо знак "мінус", якщо він є

        m = 0                               # Початкове значення суми цифр
        for c in s:                         # Перебирання символів рядка
            m += int(c)                     # Додаємо цифру до суми

        if m == 10:                         # Якщо сума цифр дорівнює 10
            k = k + 1                       # Збільшуємо кількість
            L.acquire()                     # Захоплення замка
            print(n)
            L.release()                     # Вивільнення замка

    L.acquire()
    n10 += k                                # Додаємо результат до глоб. змінної
    L.release()


M, N = 10, 10                               # Розміри масиву

L = threading.Lock()                        # Замок

mas = []                                    # Підготовка порожнього масиву рядків
for i in range(0, M):                       # Цикл за рядками
    mas_i = []                              # Підготовка порожнього рядка
    for j in range(0, N):                   # Цикл за стовпцями
        n = random.randint(-1000, 1000)     # Генерація рандомного числа
        mas_i.append(n)                     # Додавання числа в рядок
    mas.append(mas_i)                       # Додавання рядка у двовимірний масив

n10 = 0                                     # Кількість знайдених елементів

t_list = []                                 # Підготовка списку потоків

for i in range(0, M):                       # Цикл за створенням потоків
    t = threading.Thread(target=f1, args=(i, ))
    t_list.append(t)

for t in t_list:                            # Цикл за запуском потоків
    t.start()

for t in t_list:                            # Цикл за очікуванням потоків
    t.join()

print("Знайдено чисел:", n10)
```

Нижче наведені результати роботи програми для декількох запусків. Аналіз результатів свідчить про правильність написаного коду.

```
280
64
-271
721
-433
172
28
460
Знайдено чисел: 8
```

```
136
91
-703
Знайдено чисел: 3
```

```
-145
208
181
-415
-604
-145
307
Знайдено чисел: 7
```

```
820
-46
361
442
-721
28
-136
-631
118
-55
-631
181
-451
307
Знайдено чисел: 14
```

Тепер «розпаралелимо» другу частину завдання, а саме процес заміни елементів масиву, що є квадратами цілих чисел, на значення кількості елементів, сума цифр яких дорівнює десяти (тобто на значення змінної *n*10).

Як і в першій частині завдання, будемо розпаралелювати обробку масиву на рівні окремих рядків. Тобто створимо кількість потоків, рівну кількості рядків масиву, і доручимо кожному потоку обробити один рядок масиву згідно з поставленим завданням.

Для цього зробимо таке.

Напишемо функцію *f*2, яка приймає в якості параметру номер рядка двовимірного масиву, проглядає цей рядок та замінює елементи, що є квадратами цілих чисел, на значення глобальної змінної *n*10.

```
def f2(i):                                  # Заміна у рядку з номером i
    global mas                              # Будемо змінювати глобальний масив

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву

        if n > 0:                              # Якщо число додатне
            if n ** 0.5 == int(n ** 0.5):      # Якщо корінь є цілим числом
                L.acquire()                    # Захоплення замка
                mas[i][j] = n10                # Робимо заміну елементу на n10
                print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")
                L.release()                    # Вивільнення замка
```

Тут операції запису в глобальний масив та виведення на екран виконуються у «монопольному» режимі після захоплення замка.

Об’єднавши першу і другу частину програми, отримаємо такий лістинг:

```
import random
import threading

def f1(i):                                  # Пошук у рядку з номером i
    global n10                              # Будемо змінювати глобальну змінну

    k = 0                                   # Кількість знайдених ел-тів у рядку

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву
        s = str(n)                          # Переводимо в символьний формат
        s = s.replace("-", "")              # Видаляємо знак "мінус", якщо він є

        m = 0                               # Початкове значення суми цифр
        for c in s:                         # Перебирання символів рядка
            m += int(c)                     # Додаємо цифру до суми
        if m == 10:                         # Якщо сума цифр дорівнює 10
            k = k + 1                       # Збільшуємо кількість
            L.acquire()                     # Захоплення замка
            print(n)
            L.release()                     # Вивільнення замка

    L.acquire()
    n10 += k                                # Додаємо результат до глоб. змінної
    L.release()

def f2(i):                                  # Заміна у рядку з номером i
    global mas                              # Будемо змінювати глобальний масив

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву

        if n > 0:                              # Якщо число додатне
            if n ** 0.5 == int(n ** 0.5):      # Якщо корінь є цілим числом
                L.acquire()                    # Захоплення замка
                mas[i][j] = n10                # Робимо заміну елементу на n10
                print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")
                L.release()                    # Вивільнення замка


M, N = 10, 10                               # Розміри масиву

L = threading.Lock()                        # Замок
```

```
mas = []                                  # Підготовка порожнього масиву рядків
for i in range(0, M):                     # Цикл за рядками
    mas_i = []                            # Підготовка порожнього рядка
    for j in range(0, N):                 # Цикл за стовпцями
        n = random.randint(-1000, 1000) # Генерація рандомного числа
        mas_i.append(n)                   # Додавання числа в рядок
    mas.append(mas_i)                     # Додавання рядка у двовимірний масив

# Перша частина задачі

n10 = 0                                   # Кількість знайдених елементів

t_list = []                               # Підготовка списку потоків

for i in range(0, M):                     # Цикл за створенням потоків
    t = threading.Thread(target=f1, args=(i, ))  # Функція f1
    t_list.append(t)

for t in t_list:                          # Цикл за запуском потоків
    t.start()

for t in t_list:                          # Цикл за очікуванням потоків
    t.join()

print("Знайдено чисел:", n10)

# Друга частина задачі

t_list = []                               # Підготовка списку потоків

for i in range(0, M):                     # Цикл за створенням потоків
    t = threading.Thread(target=f2, args=(i, ))  # Функція f2
    t_list.append(t)

for t in t_list:                          # Цикл за запуском потоків
    t.start()

for t in t_list:                          # Цикл за очікуванням потоків
    t.join()
```

Великий розмір лістингу пояснюється тим, що ми двічі виконували роботу зі створення, запуску та очікування потоків. Можливо, можна було об’єднати ці ділянки коду всередині ще однієї функції, однак це не є принциповим і може ускладнити розуміння програми.

Нижче наведені результати кількох окремих запусків програми для масиву розміром 10 * 10.

```
 451
-811
-208
172
-253
-721
226
244
Знайдено чисел: 8
[0][2] = 441 -> 8
[2][8] = 676 -> 8
[4][6] = 961 -> 8
```

```
262
118
712
28
-721
Знайдено чисел: 5
[6][4] = 400 -> 5
```

```
-181
-19
370
-154
334
802
721
73
514
Знайдено чисел: 9
[0][8] = 625 -> 9
[5][3] = 225 -> 9
[5][5] = 625 -> 9
[5][6] = 289 -> 9
[7][5] = 121 -> 9
```

```
-235
-181
-235
Знайдено чисел: 3
```

Аналіз наведених результатів дає змогу зробити висновок, що програма працює правильно.

2.6. Нижче наведений лістинг програми з доданими командами виміру часу. Додані команди позначені сірим фоном.

```
import random
import threading
import time

def f1(i):                                  # Пошук у рядку з номером i
    global n10                              # Будемо змінювати глобальну змінну

    k = 0                                   # Кількість знайдених ел-тів у рядку

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву
        s = str(n)                          # Переводимо в символьний формат
        s = s.replace("-", "")              # Видаляємо знак "мінус", якщо він є

        m = 0                               # Початкове значення суми цифр
        for c in s:                         # Перебирання символів рядка
            m += int(c)                     # Додаємо цифру до суми

        if m == 10:                         # Якщо сума цифр дорівнює 10
            k = k + 1                       # Збільшуємо кількість
            #L.acquire()                     # Захоплення замка
            #print(n)
            #L.release()                     # Вивільнення замка

    L.acquire()
```

```
    n10 += k                                # Додаємо результат до глоб. змінної
    L.release()

def f2(i):                                  # Заміна у рядку з номером i
    global mas                              # Будемо змінювати глобальний масив

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas[i][j]                       # Беремо елемент масиву

        if n > 0:                              # Якщо число додатне
            if n ** 0.5 == int(n ** 0.5):      # Якщо корінь є цілим числом
                L.acquire()                    # Захоплення замка
                mas[i][j] = n10                # Робимо заміну елементу на n10
                #print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")
                L.release()                    # Вивільнення замка

M, N = 10, 10                               # Розміри масиву

L = threading.Lock()                        # Замок

mas = []                                    # Підготовка порожнього масиву рядків
for i in range(0, M):                       # Цикл за рядками
    mas_i = []                              # Підготовка порожнього рядка
    for j in range(0, N):                   # Цикл за стовпцями
        n = random.randint(-1000, 1000)     # Генерація рандомного числа
        mas_i.append(n)                     # Додавання числа в рядок
    mas.append(mas_i)                       # Додавання рядка у двовимірний масив

# Перша частина задачі

n10 = 0                                     # Кількість знайдених елементів

t1 = time.time()                            # Початковий час

t_list = []                                 # Підготовка списку потоків

for i in range(0, M):                       # Цикл за створенням потоків
    t = threading.Thread(target=f1, args=(i, ))  # Функція f1
    t_list.append(t)

for t in t_list:                            # Цикл за запуском потоків
    t.start()

for t in t_list:                            # Цикл за очікуванням потоків
    t.join()

print("Знайдено чисел:", n10)

# Друга частина задачі

t_list = []                                 # Підготовка списку потоків

for i in range(0, M):                       # Цикл за створенням потоків
    t = threading.Thread(target=f2, args=(i, ))  # Функція f2
    t_list.append(t)

for t in t_list:                            # Цикл за запуском потоків
    t.start()

for t in t_list:                            # Цикл за очікуванням потоків
    t.join()


t2 = time.time()                            # Кінцевий час
print("Час роботи програми:", t2-t1, "секунд.")
```

2.7. Нижче наведено заповнену таблицю 4. Експерименти проводились за тих самих умов, що й під час заповнення табл. 3.

Таблиця 4 – Результати виміру часу роботи програми з розпаралельвання

Розміри масиву	**Час виконання програми (час t_2), секунд**
1230 * 1230	0,80
1770 * 1770	0,96
2800 * 2800	3,52
3950 * 3950	6,84
6850 * 6850	20,5

2.8. Порівняємо час обробки масиву без використання потоків (табл. 3, час t_1) з відповідним часом обробки з використанням потоків (табл. 4, час t_2). Для цього об'єднаємо таблиці 3 і 4 в таблицю 5:

Таблиця 5 – Порівняння результатів виміру часу роботи програм

Розміри масиву	**Час t_1, секунд**	**Час t_2, секунд**	**Співвідношення (t_2 / t_1) * 100 %**
1230 * 1230	1	0,80	80
1770 * 1770	2	0,96	48
2800 * 2800	5	3,52	70
3950 * 3950	10	6,84	68
6850 * 6850	30	20,5	68

Останній стовпець таблиці показує, що час роботи багатопотокової програми становить приблизно 70 % від часу роботи однопотокової програми. Тобто ми отримали виграш у часі, рівний приблизно 30 %. Така ситуація є тим більш дивною, що ми очікували отримати не виграш, а програш у часі виконання, оскільки всі створені дочірні потоки все одно працюють на одному ядрі процесора, а створення, запуск і очікування завершення потоків – це додаткові витрати часу і ресурсів комп'ютера. Однак ми отримали виграш, якого бути не повинно. Цей виграш невеликий, він не є кількаразовим, але він отриманий без залучення додаткових ядер процесора, без якихось додаткових витрат ресурсів комп'ютера. І причину такого виграшу назвати важко.

В якості додаткових експериментів був виміряний час, що витрачається на першу частину завдання (пошук кількості елементів, сума цифр яких дорівнює 10) і другу частину завдання (заміна елементів, що є квадратами цілих чисел). Дослідження, проведені для масиву розміром 6850 * 6850, показали таке:

– для програми без розпаралелювання перша частина виконується в середньому за 23 секунди, друга частина – за 7 секунд;

– для багатопотокової програми перша частина виконується приблизно за 14 секунд, друга частина – за 6 секунд.

Як відомо з технічної документації, процесор i5-13500 має два типи процесорних ядер: продуктивні ядра (P-ядра), що працюють на частоті до 4.8 ГГц, та енергоефективні ядра (E-ядра), що працюють на частоті до 3.5 ГГц. Було висунуто гіпотезу, що у випадку однопотокової програми операційна система розподіляла виконувану програму на одне з E-ядер, тоді як у випадку багатопотокової програми використовувалось одне з P-ядер, завдяки чому відбувалося зменшення часу виконання програми.

Для перевірки цієї гіпотези в центральному процесорі були примусово відключені (через BIOS) усі ядра, окрім одного ядра P-типу. Це ядро містить у собі два логічні процесори. Тож після відключення операційна система Windows мала доступ лише до двох логічних процесорів замість двадцяти у випадку використання усіх ядер.

Експерименти показали, що у випадку використання одного ядра P-типу час виконання кожної з програм суттєво не змінився, а точніше, збільшився приблизно на одну секунду. Таке збільшення обумовлене тим, що операційна система була змушена виконувати усі службові та інші програми на тому самому ядрі, що й виконувані нами Python-програми. Це, безумовно, трохи збільшило загальний час роботи досліджуваних програм.

Отже, використання продуктивних ядер процесора замість енергоефективних не пояснює приріст швидкодії у процесорозалежній задачі, тому це питання залишається відкритим.

Висновки до лабораторної роботи

Під час виконання лабораторної роботи було досліджено ефективність розпаралелювання Python-програм на основі механізму потоків із використанням модуля *threading*.

Результати роботи показали, що під час розв’язання задач, пов’язаних із вводом-виводом даних, багатопотоковість дає значний приріст у швидкодії програми. В розглянутому прикладі приріст був приблизно чотирикратним і залежав від часу доступу до окремих інтернет-сторінок.

Натомість у випадку процесорозалежних задач приріст швидкодії був незначним і сягав приблизно 30 %. Причина такого приросту не може бути пояснена на рівні розробленого програмного коду і потребує додаткових досліджень.

ЛАБОРАТОРНА РОБОТА № 2
ПАРАЛЕЛІЗМ НА ОСНОВІ ПРОЦЕСІВ

Метою роботи є поглиблення практичних навичок використання механізму потоків і модуля *multiprocessing* для розпаралелювання програм, написаних мовою Python.

Завдання до лабораторної роботи

Приступати до виконання лабораторної роботи № 2 треба після виконання лабораторної роботи № 1, оскільки ця робота спирається на результати, отримані в роботі № 1.

Завдання до лабораторної роботи № 2 складається з двох незалежних частин.

У першій частині завдання необхідно розпаралелити дії, які не навантажують процесор комп'ютера, однак містять операції введення даних (запити до інтернет-сайтів), які потребують значного часу очікування. Використання механізму процесів для виконання паралельних запитів на сайт повинно демонструвати приблизно таку саму ефективність з погляду економії часу роботи програми, як і використання механізму потоків у першій лабораторній роботі.

У другій частині завдання необхідно розпаралелити дії, які потребують значних обчислювальних зусиль з боку процесора. В якості таких дій у лабораторній роботі виступає обробка двовимірних масивів. Варто очікувати, що використання механізму процесів у цьому випадку дасть тим більший виграш у часі виконання програми, чим більше буде використано процесорних ядер.

Завдання 1

Варіант завдання співпадає з варіантом першого завдання з лабораторної роботи № 1. Під час виконання роботи треба зробити таке.

1.1. Переробити програму, розроблену в пункті 1.5 лабораторної роботи № 1, у такий спосіб, щоб замість механізму потоків використовувати механізм процесів (модуль *multiprocessing*). Кількість використовуваних процесів повинна строго дорівнювати загальній кількості запитів, тобто 20 (незалежно від кількості фізичних ядер використовуваного процесора). Приклад програми розглянутий нижче.

1.2. Запустити перероблену програму кілька разів та визначити середній час послідовного доступу до усіх 20 сайтів (не до кожного окремо).

1.3. Порівняти між собою:

– середній час доступу до усіх сайтів без використання будь-яких засобів розпаралелювання (дані треба взяти з пункту 1.4 лабораторної роботи № 1);

– середній час доступу до усіх сайтів із використанням розпаралелювання на основі потоків (дані треба взяти з пункту 1.6 лабораторної роботи № 1);

– середній час доступу до усіх сайтів із використанням розпаралелювання на основі процесів, отриманий у п. 1.2 цієї лабораторної роботи.

Зробити висновки про ефективність використання кожного з механізмів розпаралелювання в програмах, що використовують «повільні» операції вводу-виводу даних.

Завдання 2

Варіант завдання співпадає з варіантом другого завдання з лабораторної роботи № 1. Під час виконання роботи треба зробити таке.

2.1. Переробити програму, розроблену в пункті 2.5 лабораторної роботи № 1, у такий спосіб, щоб замість механізму потоків використовувати механізм процесів. Обов'язковою вимогою до програми є можливість обирати кількість використовуваних процесів. Це можна робити введенням з клавіатури або задаванням константи. Приклад програми розглянутий нижче.

2.2. Переконатися, що програма працює коректно, на декількох прикладах двовимірних масивів. Приблизний розмір масивів – 3 * 3 або 4 * 4.

2.3. Додати у програму команди для виміру часу обробки масиву. Дані команди повинні вимірювати лише час самої обробки, без урахування часу на створення і псевдовипадкову генерацію масиву та на виведення результатів. Враховуватись повинна передача даних у дочірні процеси, обробка масиву дочірніми процесами, отримання результатів від дочірніх процесів головним процесом та підбиття підсумків головним процесом (за потреби).

2.4. Визначити за допомогою функції *cpu_count* кількість P доступних логічних процесорів. Переконатися, що отримане значення співпадає з технічними характеристиками використовуваного процесора, взятими в інтернеті. Модель процесора треба подивитись у розділі «Про систему» операційної системи Windows.

2.5. Взяти заповнену таблицю 3 із лабораторної роботи № 1. Вона виглядає приблизно так:

Час виконання програми (час t_1)	**Розміри масиву**
1 секунда	300 * 310
2 секунди	520 * 500
5 секунд	1100 * 1070
10 секунд	3000 * 3200
30 секунд	5500 * 5300

На основі таблиці 3 заповнити таблицю 6. У цій таблиці перший стовпець повторює другий стовпець таблиці 3 і відповідає розмірам масиву, для яких було проведено експеримент. В інших стовпцях вказуються значення часу (в секундах), отримані для різної кількості використовуваних процесів. Нижче наведена струк-

тура таблиці 6. Значення *P* означає кількість доступних логічних процесорів (згідно з п. 2.4).

Таблиця 6 – Час виконання програми за різної кількості процесів, секунд

Розміри масиву	**Кількість задіяних процесів**									
	1	**2**	**3**	**4**	**8**	**16**	***P* – 1**	***P***	***P* + 1**	**2*P***
300 * 310										
520 * 500										
1100 * 1070										
3000 * 3200										
5500 * 5300										

Якщо, наприклад, на конкретному комп'ютері значення *P* = 4, то стовпець «P» буде повторювати вміст стовпця «4» цієї ж таблиці, а стовпець «*P* – 1» – повторювати значення стовпця «3». Видаляти повторювані стовпці не треба, хай залишаються.

2.6. Із табл. 6 взяти максимальні досліджені розміри масиву (у прикладі вище це розміри 5500 * 5300). Для цих розмірів масиву знайти таку кількість задіяних логічних процесорів (ядер), за якої подальше її збільшення не дає суттєвої економії часу виконання програми.

Наприклад, були отримані такі результати:

Кількість задіяних логічних процесорів	1	2	3	4	5	6	7	8	9	10
Час виконання програми, с	85	48	37	29	21	18	17	16	16	20

Кількість задіяних логічних процесорів	11	12	13	14	15	16	17	18	19	20
Час виконання програми, с	22	21	24	23	25	24	26	30	28	29

За наведеними результатами можна стверджувати, що для розв'язуваної задачі достатньо 8 логічних процесорів. Як менша, так і більша кількість не приводять до зменшення часу виконання програми.

2.7. Порівняти результати, отримані для завдання 2 в першій і другій лабораторних роботах. А саме:

– час виконання програми в однопотоковому режимі (таблиця 3 з лабораторної роботи № 1);

– час виконання програми в багатопотоковому режимі (таблиця 4 з лабораторної роботи № 1);

– час виконання програми в багатопроцесовому режимі (таблиця 6 цієї лабораторної роботи).

Зробити висновки про ефективність розпаралелювання на основі механізму процесів під час розв'язання процесорозалежних задач.

Оформлення звіту з лабораторної роботи

Звіт оформлюється на листах формату А4 і повинен містити такі складники:

– титульний лист, оформлений за стандартом Університету;

– індивідуальний варіант для виконання завдань 1 і 2 (береться з лабораторної роботи № 1);

– опис і результати виконання завдання 1 (пункти 1.1–1.3);

– опис і результати виконання завдання 2 (пункти 2.1–2.7);

– висновки до лабораторної роботи.

Якщо лабораторна робота здається в режимі «онлайн», до звіту повинні окремо додаватись файли з програмами для 1 і 2 завдань. Замість стандартного розширення «.ру» краще надати цим файлам розширення «.txt».

Приклад виконання лабораторної роботи

Розглянемо приклад виконання лабораторної роботи. Приклад не треба розглядати як демонстрацію ідеальних рішень. Мета прикладу – показати особливості різних підходів до розпаралелювання на основі процесів.

Завдання 1

Як і в лабораторній роботі № 1, розташуємо відвідувані сайти в окремому текстовому файлі, вміст якого у нашому прикладі буде таким:

```
https://ua.sudohodstvo.org/peredumovy-vidrodzhennya-sudnobuduvannya-ukrayiny/
https://esu.com.ua/article-75226
https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky/
https://tripvenue.com.ua/ukraine/mykolayiv/experience/istoriya-sudnobuduvannya-
mista-mykolayiv
https://esu.com.ua/article-3451
https://www.marineinsight.com/types-of-ships/10-longest-ships-in-the-world-in-
2011/
https://www.industrytap.com/behemoths-on-the-sea-the-worlds-largest-ships-by-
gross-tonnage/51733
https://www.ship-technology.com/features/featureoceanic-titans-the-worlds-
biggest-ships/?cf-view
https://www.marineinsight.com/know-more/biggest-ships-in-the-world/
https://nauticalchannel.com/new/biggest-ships
https://www.aph.gov.au/Parliamentary_Business/Committees/Senate/Foreign_Affairs_
Defence_and_Trade/Completed_inquiries/2004-07/shipping/report/c03
https://www.linkedin.com/pulse/jakobstads-history-shipbuilding-seafaring-anders-
kurt%C3%A9n
https://shipsforcanada.ca/about/history
https://depts.washington.edu/chinaciv/miltech/warship.htm
http://www.clydewaterfront.com/clyde-heritage/river-clyde/shipbuilding-on-the-
clyde
https://www.wisconsinhistory.org/Records/Article/CS1822
```

```
https://www.imm-hamburg.de/museum/ausstellungsrundgang/tour-of-the-exhibitions-
english/deck-3-shipbuilding/
https://www.cna.org/archive/CNA_Files/pdf/d0006988.a1.pdf
http://shipbuildinghistory.com/shipyards/large/newportnews.htm
https://www.scafom-rux.com/en/scaffolding-blog/the-beginnings-of-shipbuilding
```

1.1. Переробимо програму, розроблену в пункті 1.5 лабораторної роботи № 1, так, щоб замість механізму потоків використовувався механізм процесів. Кількість задіяних процесів буде строго дорівнювати загальній кількості запитів, тобто 20 (незалежно від кількості фізичних ядер використовуваного процесора).

Імпортуємо необхідні модулі:

```
import urllib.request
import time
import multiprocessing as mp
```

Запит до сайтів будемо, як і раніше, робити через функцію *zapit*, яка матиме такий вигляд:

```
def zapit(s):           # Функція для виконання інтернет-запитів
    req = urllib.request.urlopen(s)
    pageHtml = req.read()
```

Додамо команди, обов'язково необхідні для мультипроцесової програми. Всередині блоку *if* розташовані команди, виконувані тільки головним процесом програми. Зауважимо, що виклик функції *freeze_support* в нашому випадку і у більшості інших випадків є зайвим, тому її можна прибрати.

```
if __name__ == "__main__":
    mp.freeze_support()
```

Прочитаємо з текстового файла список рядків:

```
    f = open("pages.txt", "rt")         # Відкриття файла для читання
    pages_list = f.readlines()          # Читання усього файла у список рядків
```

Запам'ятаємо початковий час виконання програми:

```
    global_time = 0                     # Сумарний час виконання усіх запитів
```

Побудуємо глобальний цикл на *k* повторів (тіло циклу буде розглянуте нижче):

```
    k = 10                                  # Кількість повторів експерименту
    for i in range(0, k):                   # Головний цикл на k повторів

        ...

        Тіло циклу
```

```
        ...

    global_time /= k              # Середнє арифметичне загального часу
    print("Середній загальний час усіх запитів:", global_time)
```

Тіло циклу буде таким:

```
        t_start = time.time()               # Початковий час усіх запитів

        p_list = []                         # Підготовка масиву процесів

        for s in pages_list:                # Цикл за всіма інтернет-адресами
            s = s.replace("\n", "")         # Видалення \n
            p = mp.Process(target=zapit, args=(s, )) # Створення процесу
            p_list.append(p)                          # Додавання в список

        for p in p_list:                    # Цикл запуску процесів
            p.start()

        for p in p_list:                    # Цикл очікування процесів
            p.join()

        t_end = time.time()                 # Кінцевий час усіх запитів

        global_time += t_end - t_start
        print("Прохід", i, ": час виконання =", t_end-t_start, "секунд.")
```

У тілі циклу виконуються такі дії:

– запам’ятовується початковий час виконання чергового «пакету» запитів до інтернет-сайтів;

– створюється порожній список процесів;

– відповідно до кількості інтернет-адрес створюються процеси, прив’язані до функції *zapit*;

– усі процеси послідовно запускаються;

– головний процес послідовно очікує завершення усіх дочірніх процесів;

– розраховується та виводиться на екран час виконання «пакету» запитів.

Повний лістинг програми виглядає так:

```
import urllib.request
import time
import multiprocessing as mp

def zapit(s):                          # Функція для виконання інтернет-запитів
    req = urllib.request.urlopen(s)
    pageHtml = req.read()

if __name__ == "__main__":
    mp.freeze_support()

    f = open("pages.txt", "rt")          # Відкриття файла для читання
    pages_list = f.readlines()           # Читання усього файла у список рядків

    global_time = 0                      # Сумарний час виконання усіх запитів
```

```
    k = 10                                  # Кількість повторів експерименту
    for i in range(0, k):                   # Головний цикл на k повторів

        t_start = time.time()               # Початковий час усіх запитів

        p_list = []                         # Підготовка масиву процесів

        for s in pages_list:                # Цикл за всіма інтернет-адресами
            s = s.replace("\n", "")         # Видалення \n
            p = mp.Process(target=zapit, args=(s, )) # Створення процесу
            p_list.append(p)                          # Додавання в список

        for p in p_list:                    # Цикл запуску процесів
            p.start()

        for p in p_list:                    # Цикл очікування процесів
            p.join()

        t_end = time.time()                 # Кінцевий час усіх запитів

        global_time += t_end - t_start
        print("Прохід", i, ": час виконання =", t_end-t_start, "секунд.")

    global_time /= k              # Середнє арифметичне загального часу
    print("Середній загальний час усіх запитів:", global_time)
```

1.2. Запустимо програму кілька разів та визначимо середній час послідовного доступу до усіх 20 сайтів.

У процесі виконання цього пункту несподівано було з’ясовано, що доступ до сторінки:

```
https://uprom.info/news/ships/ukrayinske-sudnobuduvannya-pidsumky/
```

відсутній з невідомих причин. Тому було прийнято рішення замінити в текстовому файлі дану адресу на наступну, також присвячену суднобудуванню в Україні:

```
https://www.pollawlife.com.ua/2014/11/blog-post_24.html
```

Експериментально отримані результати роботи програми для значення $k = 10$ (для десяти повторів) виглядають так:

```
Прохід 0 : час виконання = 4.427789211273193 секунд.
Прохід 1 : час виконання = 2.9980099201202393 секунд.
Прохід 2 : час виконання = 3.0012433528900146 секунд.
Прохід 3 : час виконання = 3.3401908874511172 секунд.
Прохід 4 : час виконання = 3.1328299045562744 секунд.
Прохід 5 : час виконання = 2.9898080825805664 секунд.
Прохід 6 : час виконання = 3.821261405944824 секунд.
Прохід 7 : час виконання = 3.6943349838256836 секунд.
Прохід 8 : час виконання = 3.7533011436462402 секунд.
Прохід 9 : час виконання = 3.5264413356781006 секунд.
Середній загальний час усіх запитів: 3.4685210227966308
```

Результати ще одного запуску:

```
Прохід 0 : час виконання = 3.3721718788146973 секунд.
Прохід 1 : час виконання = 3.9884238243103027 секунд.
Прохід 2 : час виконання = 5.039010047912598 секунд.
Прохід 3 : час виконання = 6.126359939575195 секунд.
Прохід 4 : час виконання = 5.160638093948364 секунд.
Прохід 5 : час виконання = 3.175748109817505 секунд.
Прохід 6 : час виконання = 3.8132002353668213 секунд.
Прохід 7 : час виконання = 4.209027051925659 секунд.
Прохід 8 : час виконання = 3.7431859970092773 секунд.
Прохід 9 : час виконання = 3.1992790699005127 секунд.
Середній загальний час усіх запитів: 4.182704424858093
```

1.3. Задача організації запитів до інтернет-сторінок не є задачею, яка потребує значних потужностей центрального процесора, і належить до класу задач, пов'язаних із вводом-виводом інформації. Аналіз отриманих результатів, а також результатів, отриманих під час виконання завдання 1 лабораторної роботи № 1, дають змогу зробити такі висновки.

1. Без використання механізмів розпаралелювання можливе лише послідовне виконання інтернет-запитів. Для обраних 20 сайтів час послідовного виконання запитів становить приблизно 27 секунд (за результатами виконання п. 1.4 лабораторної роботи № 1).

2. Використання механізму розпаралелювання на основі потоків (згідно з результатами п. 1.6 лабораторної роботи № 1) дає змогу виконати всі запити приблизно за 3–4 секунди (за умов достатньої швидкості інтернет-каналу).

3. Використання механізму розпаралелювання на основі процесів, за результатами п. 1.2 цієї лабораторної роботи, дає змогу виконати всі запити також за 3–4 секунди.

Отже, використання як механізму потоків, так і механізму процесів дає змогу у кілька разів зменшити час виконання множини інтернет-запитів. У загальному випадку треба очікувати, що час паралельного виконання множини інтернет-запитів буде наближатись до часу запиту до найбільш «повільного» сайта.

Це свідчить про високу ефективність використання обох досліджених механізмів розпаралелювання в програмах, що не залежать значною мірою від швидкодії центрального процесора комп'ютера.

Завдання 2

Друге завдання присвячене паралельній обробці двовимірних масивів і формулюється так:

Заданий двовимірний масив розміром M рядків на N стовпців (розміри масиву задаються на початку програми). Кожним елементом масиву є псевдовипадкове ціле число в діапазоні від –1 000 до 1 000. Знайти кількість елементів масиву, сума

цифр яких, незалежно від знака числа, дорівнює 10 (наприклад, 91, –253, 730, –28, 55). Замінити на це число усі елементи масиву, які є квадратами цілих чисел.

Задача чітко розділяється на дві частини. Ці частини повинні виконуватись послідовно, оскільки друга частина залежить від результатів першої частини. Для кращого розуміння будемо розглядати кожну частину окремо, після чого об'єднаємо обидві частини в одну програму.

Завдання 2, частина 1

Знайти кількість елементів масиву, сума цифр яких, незалежно від знака числа, дорівнює 10 (наприклад, 91, –253, 730, –28, 55).

Основною проблемою під час використання механізму процесів є те, що дочірні процеси працюють як самостійні програми, не пов'язані з головним процесом. Кожен дочірній процес виконується на окремому процесорному ядрі і має власну ділянку пам'яті, не доступну іншим процесам. Відповідно дочірні процеси, на відміну від потоків, не можуть безпосередньо використовувати глобальні змінні і масиви даних, створені в головному процесі.

Для того, щоб дочірній процес зміг обробляти дані, ці дані треба йому передати. На передачу даних витрачається певний час. Якщо обсяг даних великий, а алгоритм їх обробки відносно простий, час на передачу даних може бути більшим за час обробки. До того ж дочірній процес у більшості випадків повинен повернути результат своєї роботи головному процесу, що також займає певний час.

Якщо час, що витрачається на передачу даних між процесами, є надто великим, розпаралелювання програми може бути невигідним. Тоді треба обирати реалізацію заданого алгоритму у вигляді одного процесу. В загальному випадку ми не можемо заздалегідь оцінити ефективність від розпаралелювання програми, тому змушені проводити відповідні дослідження та експерименти.

Під час використання модуля *multiprocessing* існують різні способи передачі даних у дочірні процеси. Найбільш поширеними є такі:

– передача даних через аргументи цільової функції;
– використання черг (queue);
– використання менеджера процесів.

Розглянемо кожен із цих способів та дослідимо, який з них забезпечує кращу швидкодію під час розв'язання нашої задачі. Під час розгляду кожного зі способів домовимось, що кожен дочірній процес буде обробляти лише один рядок двовимірного масиву. Отже, кількість створюваних процесів буде дорівнювати кількості рядків масиву. Після того, як ми визначимо найбільш ефективний спосіб передачі даних у процеси, ми модифікуємо програму так, щоб кількість використовуваних процесів не залежала від кількості рядків масиву.

Спосіб 1. Передача даних через аргументи цільової функції

Насамперед зробимо таке:

– імпортуємо необхідні модулі;

– задамо розміри масиву;

– створимо двовимірний масив, заповнений псевдовипадковими числами в діапазоні від –1 000 до 1 000.

```
import random
import multiprocessing as mp
import time

if __name__ == "__main__":                  # Для роботи головного процесу
    mp.freeze_support()                     # Хай буде

    M, N = 3, 3                             # Розміри масиву

    L = mp.Lock()                           # Замок

    mas = []                                # Підготовка масиву рядків
    for i in range(0, M):                   # Цикл за рядками
        mas_i = []                          # Підготовка порожнього рядка
        for j in range(0, N):               # Цикл за стовпцями
            n = random.randint(-1000, 1000) # Генерація рандомного числа
            mas_i.append(n)                 # Додавання числа в рядок
        mas.append(mas_i)                   # Додавання рядка у масив
```

Як відомо, результатом розв'язання першої частини задачі є кількість елементів двовимірного масиву, сума цифр яких дорівнює 10. Якщо кожному дочірньому процесу передати один рядок масиву, дочірній процес зможе знайти кількість відповідних елементів в отриманому рядку. Цю кількість треба якось передати в головний процес та додати до загальної кількості знайдених елементів.

Для цього можна скористатись класом *Value,* який дає змогу створювати спільні міжпроцесові змінні. Створимо змінну *n*10, яка буде мати цілий тип та містити загальну кількість знайдених елементів масиву.

```
    n10 = mp.Value("i", 0)                  # Кількість елементів
```

Оскільки змінна *n*10 буде доступна для запису усім дочірнім процесам, можливе виникнення стану перегонів. Для його усунення скористаємось звичайним замком, який створимо в головному процесі:

```
    L = mp.Lock()                           # Замок
```

Створимо функцію *f*1, яка буде виступати в якості цільової функції кожного дочірнього процесу. Функція виконує такі дії:

– отримує в якості аргументів усі необхідні для роботи дані: рядок масиву, глобальну змінну *n*10, глобальний замок *L*;

– знаходить в отриманому рядку кількість елементів, сума цифр у яких дорівнює 10;

– додає знайдену кількість до глобальної змінної *n*10.

```
def f1(mas_line, n10, L):                  # Функція пошуку в рядку
    N = len(mas_line)                      # Довжина рядка
    k = 0                                  # Кількість знайдених елементів

    for j in range(0, N):                  # Цикл за стовпцями
        n = mas_line[j]                    # Беремо елемент масиву
        n = abs(n)                         # Прибираємо знак "-", якщо є
        s = str(n)                         # Переводимо в символьний формат

        m = 0                              # Початкове значення суми цифр
        for c in s:                        # Перебирання символів рядка
            m += int(c)                    # Додаємо цифру до суми
        if m == 10:                        # Якщо сума цифр дорівнює 10
            k = k + 1                      # Збільшуємо кількість

    L.acquire()
    n10.value = n10.value + k              # Додаємо результат до глоб. змінної
    L.release()
```

Далі в циклі створимо *M* процесів, у кожен з яких будемо передавати один рядок масиву *mas*.

```
    t1 = time.time()                       # Початковий час

    p_list = []                            # Підготовка списку потоків

    for i in range(0, M):                  # Цикл за створенням потоків
        p = mp.Process(target=f1, args=(mas[i], n10, L))
        p_list.append(p)

    for p in p_list:                       # Цикл за запуском потоків
        p.start()

    for p in p_list:                       # Цикл за очікуванням потоків
        p.join()

    for i in mas:                          # Виведення масиву за рядками
        print(i)

    print("Знайдено чисел:", n10.value)

    t2 = time.time()                       # Кінцевий час
    print("Час виконання:", t2-t1, "секунд.")
```

Зазначимо, що цикл виведення масиву за рядками доречно застосовувати тільки у випадку масивів невеликого розміру (під час тестування програми). Для масивів великого розміру цей цикл треба прибрати (закоментувати).

Нижче наведений повний лістинг програми.

```
import random
import multiprocessing as mp
```

```
import time

def f1(mas_line, n10, L):                  # Функція пошуку в рядку
    N = len(mas_line)                      # Довжина рядка
    k = 0                                  # Кількість знайдених елементів
    for j in range(0, N):                  # Цикл за стовпцями
        n = mas_line[j]                    # Беремо елемент масиву
        n = abs(n)                         # Прибираємо знак "-", якщо є
        s = str(n)                         # Переводимо в символьний формат

        m = 0                              # Початкове значення суми цифр
        for c in s:                        # Перебирання символів рядка
            m += int(c)                    # Додаємо цифру до суми
        if m == 10:                        # Якщо сума цифр дорівнює 10
            k = k + 1                      # Збільшуємо кількість

    L.acquire()
    n10.value = n10.value + k              # Додаємо результат до глоб. змінної
    L.release()

if __name__ == "__main__":                     # Для роботи головного процесу
    mp.freeze_support()                        # Хай буде

    M, N = 3, 3                                # Розміри масиву

    L = mp.Lock()                              # Замок

    mas = []                                   # Підготовка масиву рядків
    for i in range(0, M):                      # Цикл за рядками
        mas_i = []                             # Підготовка порожнього рядка
        for j in range(0, N):                  # Цикл за стовпцями
            n = random.randint(-1000, 1000)    # Генерація рандомного числа
            mas_i.append(n)                    # Додавання числа в рядок
        mas.append(mas_i)                      # Додавання рядка у масив

    n10 = mp.Value("i", 0)                     # Кількість елементів

    t1 = time.time()                           # Початковий час

    p_list = []                                # Підготовка списку потоків

    for i in range(0, M):                      # Цикл за створенням потоків
        p = mp.Process(target=f1, args=(mas[i], n10, L))
        p_list.append(p)

    for p in p_list:                           # Цикл за запуском потоків
        p.start()

    for p in p_list:                           # Цикл за очікуванням потоків
        p.join()

    for i in mas:                              # Виведення масиву за рядками
        print(i)
    print("Знайдено чисел:", n10.value)

    t2 = time.time()                           # Кінцевий час
    print("Час виконання:", t2-t1, "секунд.")
```

Запустивши програму кілька разів для значень $N = 3$, $M = 3$, отримаємо приблизно такі результати:

```
[938, -516, 154]
[95, 805, 543]
[-369, -904, -446]
Знайдено чисел: 1
Час виконання: 0.11059951782226562 секунд.
```

```
[475, -751, 431]
[-191, -991, -400]
[714, -251, -50]
Знайдено чисел: 0
Час виконання: 0.14136362075805664 секунд.
```

```
[-325, -445, 634]
[537, 450, 216]
[-280, 329, 856]
Знайдено чисел: 2
Час виконання: 0.12595725059509277 секунд.
```

Отже, будемо вважати, що програма працює коректно.

У лабораторній роботі № 1 ми проводили дослідження часу обробки масивів з такими розмірами: 1230 * 1230, 1770 * 1770, 2800 * 2800, 3950 * 3950, 6850 * 6850. Запустивши наведену вище програму для зазначених розмірів масиву (тобто задавши відповідні значення змінних *M* і *N*), отримаємо значення часу виконання програми, наведені в табл. 7.

Таблиця 7 – Час виконання програми (спосіб 1)

Розміри масиву	Час виконання, секунд
1230 * 1230	40
1770 * 1770	58
2800 * 2800	92
3950 * 3950	158
6850 * 6850	273

Спосіб 2. Передача даних через об’єкт класу «Черга»

Модифікуємо програму в такий спосіб:

– перед передачею в дочірній процес кожен рядок масиву кладеться в окремий об’єкт типу *mp.Queue* (в об’єкт класу «черга»);

– цей об’єкт передається в цільову функцію *f*1 як аргумент;

– на початку функції міст переданої черги прочитується в локальний список *mas_line*, після чого робота ведеться з цим списком.

Лістинг програми наведений нижче. Рядки, які були змінені, порівняно з попереднім лістингом, позначені сірим фоном.

```
import random
import multiprocessing as mp
import time
```

```
def f1(q, n10, L):                          # Функція пошуку в рядку

    mas_line = q.get()                      # Читання з черги рядка масиву

    N = len(mas_line)                       # Довжина рядка
    k = 0                                   # Кількість знайдених елементів

    for j in range(0, N):                   # Цикл за стовпцями
        n = mas_line[j]                     # Беремо елемент масиву
        n = abs(n)                          # Прибираємо знак "-", якщо є
        s = str(n)                          # Переводимо в символьний формат

        m = 0                               # Початкове значення суми цифр
        for c in s:                         # Перебирання символів рядка
            m += int(c)                     # Додаємо цифру до суми
        if m == 10:                         # Якщо сума цифр дорівнює 10
            k = k + 1                       # Збільшуємо кількість

    L.acquire()
    n10.value = n10.value + k               # Додаємо результат до глоб. змінної
    L.release()


if __name__ == "__main__":                  # Для роботи головного процесу
    mp.freeze_support()                     # Хай буде

    M, N = 3, 3                             # Розміри масиву

    L = mp.Lock()                           # Замок

    mas = []                                # Підготовка масиву рядків
    for i in range(0, M):                   # Цикл за рядками
        mas_i = []                          # Підготовка порожнього рядка
        for j in range(0, N):               # Цикл за стовпцями
            n = random.randint(-1000, 1000) # Генерація рандомного числа
            mas_i.append(n)                 # Додавання числа в рядок
        mas.append(mas_i)                   # Додавання рядка у масив

    n10 = mp.Value("i", 0)                  # Кількість елементів

    t1 = time.time()                        # Початковий час

    p_list = []                             # Підготовка списку потоків

    for i in range(0, M):                   # Цикл за створенням потоків
        q = mp.Queue()                      # Створення об'єкта «черга»
        q.put(mas[i])                       # Взяття в чергу рядка масиву
        p = mp.Process(target=f1, args=(q, n10, L))
        p_list.append(p)

    for p in p_list:                        # Цикл за запуском потоків
        p.start()

    for p in p_list:                        # Цикл за очікуванням потоків
        p.join()

    print("Знайдено чисел:", n10.value)

    t2 = time.time()                        # Кінцевий час
    print("Час виконання:", t2-t1, "секунд.")
```

Результати дослідження часу виконання для масивів різного розміру наведені в табл. 8.

Таблиця 8 – Час виконання програми (спосіб 2)

Розміри масиву	Час виконання, секунд
1230 * 1230	9,25
1770 * 1770	13,3
2800 * 2800	21,5
3950 * 3950	32,7
6850 * 6850	60,5

Спосіб 3. Передача даних через менеджер процесів

Модифікуємо програму в такий спосіб.

– у головному процесі створимо об’єкт класу *mp.Manager*;

– у створеному об’єкті-менеджері створимо список *arr*, що є копією початкового масиву *mas*;

– передамо в цільову функцію список *arr* та номер рядка, який треба обробити.

Лістинг програми виглядатиме так:

```
import random
import multiprocessing as mp
import time

def f1(arr, i, n10, L):                 # Функція пошуку в рядку

    mas_line = arr[i]                   # Читання рядка масиву

    N = len(mas_line)                   # Довжина рядка
    k = 0                               # Кількість знайдених елементів

    for j in range(0, N):               # Цикл за стовпцями
        n = mas_line[j]                 # Беремо елемент масиву
        n = abs(n)                      # Прибираємо знак "-", якщо є
        s = str(n)                      # Переводимо в символьний формат

        m = 0                           # Початкове значення суми цифр
        for c in s:                     # Перебирання символів рядка
            m += int(c)                 # Додаємо цифру до суми
        if m == 10:                     # Якщо сума цифр дорівнює 10
            k = k + 1                   # Збільшуємо кількість

    L.acquire()
    n10.value = n10.value + k           # Додаємо результат до глоб. змінної
    L.release()

if __name__ == "__main__":                  # Для роботи головного процесу
    mp.freeze_support()                     # Хай буде

    M, N = 1230, 1230                       # Розміри масиву

    L = mp.Lock()                           # Замок

    mas = []                                # Підготовка масиву рядків
    for i in range(0, M):                   # Цикл за рядками
```

```
        mas_i = []                              # Підготовка порожнього рядка
        for j in range(0, N):                   # Цикл за стовпцями
            n = random.randint(-1000, 1000) # Генерація рандомного числа
            mas_i.append(n)                     # Додавання числа в рядок
        mas.append(mas_i)                       # Додавання рядка у масив

    n10 = mp.Value("i", 0)                      # Кількість елементів

    t1 = time.time()                            # Початковий час

    man = mp.Manager()                          # Менеджер процесів

    arr = man.list(mas)                         # Список у межах менеджера

    p_list = []                                 # Підготовка списку потоків

    for i in range(0, M):                       # Цикл зі створення потоків
        q = mp.Queue()
        q.put(mas[i])
        p = mp.Process(target=f1, args=(arr, i, n10, L))
        p_list.append(p)

    for p in p_list:                            # Цикл за запуском потоків
        p.start()

    for p in p_list:                            # Цикл за очікуванням потоків
        p.join()

    print("Знайдено чисел:", n10.value)

    t2 = time.time()                            # Кінцевий час
    print("Час виконання:", t2-t1, "секунд.")
```

Результати дослідження часу виконання наведені в табл. 9.

Таблиця 9 – Час виконання програми (спосіб 3)

Розміри масиву	Час виконання, секунд
1230 * 1230	10,5
1770 * 1770	14,7
2800 * 2800	25,6
3950 * 3950	38,2
6850 * 6850	70,1

Аналіз таблиць 6–8 показує, що перший спосіб є найбільш повільним, тоді як другий і третій способи дають значно менший час виконання. Хоча за результатами експерименту другий спосіб (використання черг *Queue*) дав трохи кращий результат, менеджер процесів має більше можливостей. Тому умовимось надалі використовувати передачу даних у дочірні процеси з використанням менеджера процесів (тобто спосіб 3).

Зазначимо таке. Будь-який процесор комп’ютера має обмежену кількість ядер. Кожне ядро містить один або два логічні процесори (в документації логічні процесори часто називають потоками). Поняття ядра та логічного процесора часто

ототожнюють. Наявну кількість логічних процесорів можна визначити за допомогою функції *cpu_count* модуля *multiprocessing*.

Процесор комп'ютера здатен одночасно виконувати кількість задач, яка дорівнює кількості наявних логічних процесорів. Це означає, що немає практичного сенсу створювати кількість процесів, більшу за кількість логічних процесорів (ядер) комп'ютера. Якщо процесор містить 10 ядер, немає сенсу створювати 20, 30, 100 процесів – все одно вони будуть виконані псевдопаралельно, по 10 процесів одночасно.

Зробимо такі дії:

1. Визначимо наявну кількість логічних процесорів (ядер). Будемо позначати цю кількість як *p_num*.

2. Розділимо задачу на *p_num* частин так, щоб усі частини мали приблизно однаковий обсяг роботи. В нашому випадку кожна частина – це певний діапазон рядків двовимірного масиву. Якщо в попередніх прикладах ми передавали в кожен процес тільки один рядок масиву, то тепер будемо передавати діапазон рядків.

3. Виконаємо *p_num* процесів, після чого зберемо та обробимо отримані результати.

Наш підхід буде дуже простим. Нехай у процесора є чотири ядра. Якщо двовимірний масив містить 100 рядків (з 0 по 99), то на перше ядро виділимо рядки з 0 до 24, на друге ядро – рядки з 25 до 49, на 3 ядро – рядки з 50 до 74, на 4 ядро – рядки з 75 до 99. Отже, на кожне процесорне ядро буде припадати однакова кількість роботи. Водночас не треба створювати величезну кількість процесів та керувати ними. Все одно одночасно зможуть виконуватись тільки чотири процеси.

В загальному випадку кількість процесорних ядер буде дорівнювати *p_num*, а кількість рядків масиву дорівнюватиме *M*. Отже, постає задача розділити *M* рядків на *p_num* приблизно однакових частин. Це можна зробити по-різному. У цьому прикладі розглянемо найбільш простий спосіб.

Спочатку, як завжди, під'єднаємо необхідні модулі:

```
import random
import multiprocessing as mp
import time
```

Функцію *f*1 переробимо так, щоб вона приймала не просто рядок масиву, а номери першого і останнього рядків масиву *start_line* та *end_line*. Сам масив передається як змінна *arr*, яка є посиланням на об'єкт типу *mp.Manager.list*.

На початку роботи функція *f*1 виділяє потрібний їй підмасив у локальну змінну *sub_mas*, самостійно визначає його розміри і шукає в ньому кількість елементів, сума цифр яких дорівнює 10. Наприкінці роботи знайдена кількість елементів додається до глобальної змінної *n*10.

```
def f1(start_line, end_line, arr, n10, L):   # Пошук у підмасиві

    sub_mas = arr[start_line:end_line]       # Отримання підмасиву

    M = len(sub_mas)                       # Кількість рядків підмасиву
    N = len(sub_mas[0])                    # Кількість стовпців підмасиву

    k = 0                                  # Кількість знайдених елементів
    for i in range(0, M):
        for j in range(0, N):                  # Цикл за стовпцями
            n = sub_mas[i][j]                  # Беремо елемент масиву
            s = str(n)                         # Переводимо в символьний формат
            s = s.replace("-", "")             # Видаляємо знак "мінус", якщо є

            m = 0                              # Початкове значення суми цифр
            for c in s:                        # Перебирання символів рядка
                m += int(c)                    # Додаємо цифру до суми

            if m == 10:                        # Якщо сума цифр дорівнює 10
                k = k + 1                      # Збільшуємо кількість
    L.acquire()
    n10.value = n10.value + k              # Додаємо результат до глоб. змінної
    L.release()
```

Звернемо увагу, що оскільки функція працює не з одним рядком, а з двовимірним підмасивом, перегляд цього підмасиву буде відбуватись за допомогою двох вкладених циклів – за рядками (цикл «*for i*») і за стовпцями (цикл «*for j*»).

Для функції абсолютно не важливо, яким буде розмір отриманого підмасиву. «Розрізанням» масиву на підмасиви буде займатись головний процес програми. У крайньому випадку, якщо процесор містить лише одне ядро, функція *f*1 отримає в якості підмасиву увесь масив цілком. Це буде тотожнім обробці масиву в одному процесі без розпаралелювання.

Тепер розглянемо головний процес програми. Його початок буде схожим з раніше розглянутими прикладами:

```
if __name__ == "__main__":                 # Початок головного процесу
    mp.freeze_support()                    # Хай буде

    M, N = 100, 100                        # Розміри масиву

    L = mp.Lock()                          # Замок

    mas = []                               # Підготовка масиву рядків
    for i in range(0, M):                  # Цикл по рядкам
        mas_i = []                         # Підготовка порожнього рядка
        for j in range(0, N):              # Цикл по стовпцям
            n = random.randint(-1000, 1000) # Генерація рандомного числа
            mas_i.append(n)                # Додавання числа в рядок
        mas.append(mas_i)                  # Додавання рядка у масив
```

Далі створимо об’єкт класу *mp.Manager*, на основі якого створимо спільний список типу *mp.Manager.list*. Також створимо окрему спільну змінну *n*10, яка

буде доступна в дочірніх процесах, та визначимо кількість процесорних ядер за допомогою функції *cpu_count*.

```
man = mp.Manager()                          # Менеджер процесів
arr = man.list(mas)                         # Список у межах менеджера

n10 = mp.Value("i", 0)                      # Кількість елементів

p_num = mp.cpu_count()                      # Кількість процесів
```

Розрахуємо кількість рядків масиву *arr*, які будуть передаватись одним дочірнім процесом.

Спочатку розглянемо пояснюючий приклад. Нехай масив *arr* містить $M = 75$ рядків, тоді як процесор має $p\_num = 8$ процесорних ядер. Розділивши 77 на 8, отримаємо значення 9.375, тобто в один дочірній процес треба передати 9.375 рядка масиву. Але ми можемо передавати тільки цілу кількість рядків. Оскільки частка M / p_num не є цілим числом, ми округлимо його до найближчого більшого цілого, тобто до 10.

Отже, в один дочірній процес будемо передавати 10 рядків масиву. Неважко порахувати, що першим 7 процесам будуть передані перші 70 рядків (по 10 в один процес). На останній, восьмий, процес залишиться лише 5 рядків, оскільки загальна кількість рядків дорівнює 75. Така ситуація є цілком нормальною: останній процес отримає ту частину роботи, яка залишилась. Якщо б рядків у масиві було 80, кожен із 8 дочірніх процесів отримав би по 10 рядків масиву.

Розглянутий алгоритм виражається такими рядками коду:

```
block_size_float = M / p_num            # Дробова кількість рядків
block_size = int(block_size_float)      # Ціла кількість рядків
if block_size != block_size_float:      # Якщо вони не рівні
    block_size = int(block_size) + 1    # Округлення до більшого цілого
```

Тепер напишемо цикл, який буде шукати черговий шматок рядків розміром *block_size*, визначаючи для нього номери початкового і кінцевого рядків (змінні *start_line* та *end_line*). Ці номери будуть передаватись у створюваний дочірній процес, після чого цикл шукатиме наступний діапазон рядків. Ось як це виглядає.

```
p_list = []                                 # Підготовка списку процесів

start_line = 0                              # Початковий рядок блоку
end_line = 0                                # Кінцевий рядок блоку

while end_line < M :   # Поки не усі рядки розподілені

    end_line = start_line + block_size  # Номер кінцевого рядка + 1

    if end_line > M:       # Якщо перестрибнули останній рядок,
        end_line = M       # обмежуємось останнім рядком
```

```
        # Створення процесу і додавання в список процесів
        p = mp.Process(target=f1, args=(start_line, end_line, arr, n10, L))
        p_list.append(p)

        start_line = end_line;
```

Далі йдуть стандартні цикли запуску та приєднання процесів, а також виведення кінцевого результату.

```
    for p in p_list:                          # Цикл за запуском процесів
        p.start()

    for p in p_list:                          # Цикл за очікуванням процесів
        p.join()

    print("Знайдено чисел:", n10.value)
```

Увесь лістинг програми, разом із доданими командами для виміру часу виконання, має такий вигляд.

```
import random
import multiprocessing as mp
import time

def f1(start_line, end_line, arr, n10, L):    # Пошук у підмасиві

    sub_mas = arr[start_line:end_line]        # Взяття підмасиву

    M = len(sub_mas)                      # Кількість рядків підмасиву
    N = len(sub_mas[0])                   # Кількість стовпців підмасиву

    k = 0                                 # Кількість знайдених елементів
    for i in range(0, M):
        for j in range(0, N):                 # Цикл за стовпцями
            n = sub_mas[i][j]                 # Беремо елемент масиву
            s = str(n)                        # Переводимо в символьний формат
            s = s.replace("-", "")            # Видаляємо знак "мінус", якщо є

            m = 0                             # Початкове значення суми цифр
            for c in s:                       # Перебирання символів рядка
                m += int(c)                   # Додаємо цифру до суми

            if m == 10:                       # Якщо сума цифр дорівнює 10
                k = k + 1                     # Збільшуємо кількість

    L.acquire()
    n10.value = n10.value + k             # Додаємо результат до глоб. змінної
    L.release()

if __name__ == "__main__":                    # Початок головного процесу
    mp.freeze_support()                       # Хай буде

    M, N = 100, 100                           # Розміри масиву

    L = mp.Lock()                             # Замок

    mas = []                                  # Підготовка масиву рядків
    for i in range(0, M):                     # Цикл за рядками
        mas_i = []                            # Підготовка порожнього рядка
```

```
        for j in range(0, N):                # Цикл за стовпцями
            n = random.randint(-1000, 1000) # Генерація рандомного числа
            mas_i.append(n)                 # Додавання числа в рядок
        mas.append(mas_i)                   # Додавання рядка у масив

    # Перша частина задачі

    t1 = time.time()                        # Початковий час

    man = mp.Manager()                      # Менеджер процесів
    arr = man.list(mas)                     # Список у межах менеджера

    n10 = mp.Value("i", 0)                  # Кількість елементів

    p_num = mp.cpu_count()                  # Кількість процесів

    # Розраховуємо розмір блоку одного завдання
    # (кількість рядків матриці, що надсилаються одному процесу)
    block_size_float = M / p_num           # Дробова кількість рядків
    block_size = int(block_size_float)     # Ціла кількість рядків
    if block_size != block_size_float:     # Якщо вони не рівні
        block_size = int(block_size) + 1   # Округлення до більшого цілого

    p_list = []                             # Підготовка списку процесів

    start_line = 0                          # Початковий рядок блоку
    end_line = 0                            # Кінцевий рядок блоку
    while end_line < M :   # Поки не усі рядки розподілені

        end_line = start_line + block_size  # Номер кінцевого рядка + 1

        if end_line > M:       # Якщо перестрибнули останній рядок,
            end_line = M       # обмежуємось останнім рядком

        # Створення процесу і додавання у список процесів
        p = mp.Process(target=f1, args=(start_line, end_line, arr, n10, L))
        p_list.append(p)

        start_line =  end_line;             # Кінцевий рядок стає початковим

    for p in p_list:                        # Цикл за запуском процесів
        p.start()

    for p in p_list:                        # Цикл за очікуванням процесів
        p.join()

    print("Знайдено чисел:", n10.value)

    t2 = time.time()                        # Кінцевий час
    print("Перша частина:", t2-t1, "секунд.")
```

Перевіримо швидкодію програми. Для значень $M = 6850$, $N = 6850$ отримано приблизно такі результати (у разі використання 20 логічних процесорів):

```
Знайдено чисел: 2955834
Перша частина: 4.79215669631958 секунд.
```

```
Знайдено чисел: 2955166
Перша частина: 4.90946102142334 секунд.
```

```
Знайдено чисел: 2957216
Перша частина: 4.824645280838013 секунд.
```

Треба зазначити, що під час виконання програми витрачалось близька 4 Гб оперативної пам'яті, яка після завершення програми була вивільнена «збирачем сміття» інтерпретатора Python.

Завдання 2, частина 2

Замінити на знайдене в частині 1 число всі елементи масиву, які є квадратами цілих чисел.

Результатом роботи першої частини програми є значення змінної *n*10, яке дорівнює кількості елементів масиву, сума цифр яких дорівнює 10. Напишемо функцію *f*2, яка буде замінювати на значення *n*10 усі елементи глобального масиву *arr*, які є квадратами цілих чисел.

Оскільки на комп'ютері будуть одночасно виконуватись декілька екземплярів функції *f*2 (кожен екземпляр – в окремому дочірньому процесі), функція *f*2 повинна вміти обробляти будь-який заданий діапазон рядків масиву *arr* (так само, як і функція *f*1). Це дає змогу зробити перелік аргументів функції *f*2 таким самим, як і у функції *f*1:

```
def f2(start_line, end_line, arr, n10, L):  # Заміна елементів підмасиву
```

Далі, як і у функції *f*1, прочитаємо зі спільного масиву *arr* підмасив рядків згідно до значень *start_line* та *end_line*. Також визначимо розміри *M* і *N* прочитаного підмасиву.

```
    sub_mas = arr[start_line:end_line]      # Отримання підмасиву

    M = len(sub_mas)                        # Кількість рядків підмасиву
    N = len(sub_mas[0])                     # Кількість стовпців підмасиву
```

Організуємо два вкладені цикли, які будуть перебирати по черзі всі елементи підмасиву *sub_mas*. Зовнішній цикл перебиратиме рядки підмасиву, внутрішній – стовпці. Всередині циклу будемо прочитувати черговий елемент підмасиву та, якщо він є квадратом цілого числа, замінювати його на значення змінної *n*10:

```
    for i in range(0, M):
        for j in range(0, N):               # Цикл за стовпцями
            n = sub_mas[i][j]               # Беремо елемент масиву

            if n > 0:                           # Якщо число додатне
                if n ** 0.5 == int(n ** 0.5):  # Якщо корінь є цілим числом
                    sub_mas[i][j] = n10.value  # Робимо заміну на n10
```

Після завершення обробки підмасиву необхідно оновити відповідні рядки у спільному масиві *arr*. Це можна зробити, наприклад, у такий спосіб:

```
    L.acquire()
    arr[start_line:end_line] = sub_mas   # Оновлення всього підмасиву відразу
    L.release()
```

Можна було б оновлювати рядки не після завершення обробки підмасиву, а після кожного обробленого рядка. Це виглядало б так:

```
    for i in range(0, M):
        for j in range(0, N):                # Цикл за стовпцями
            n = sub_mas[i][j]                # Беремо елемент масиву

            if n > 0:                           # Якщо число додатне
                if n ** 0.5 == int(n ** 0.5):  # Якщо корінь є цілим числом
                    sub_mas[i][j] = n10.value  # Робимо заміну на n10
        L.acquire()
        arr[start_line+i] = sub_mas[i]          # Оновлюємо черговий рядок
        L.release()
```

У будь-якому випадку команду оновлення масиву *arr* краще брати в «замок» для запобігання утворенню перегонів даних.

Тепер розглянемо другу частину головного процесу програми. Ця частина розташована після першої частини і майже точно її повторює.

```
    print("\nДруга частина розпочала роботу.")

    t1 = time.time()                         # Початковий час

    p_list = []                              # Підготовка списку процесів

    start_line = 0                           # Початковий рядок блоку
    end_line = 0                             # Кінцевий рядок блоку
    while end_line < M :   # Поки не всі рядки розподілені

        end_line = start_line + block_size  # Номер кінцевого рядка + 1
        if end_line > M:       # Щоб не перестрибнути останній рядок
            end_line = M

        p = mp.Process(target=f2, args=(start_line, end_line, arr, n10, L))
        p_list.append(p)

        start_line =  end_line;              # Кінцевий рядок стає початковим

    for p in p_list:                         # Цикл за запуском процесів
        p.start()

    for p in p_list:                         # Цикл за очікуванням процесів
        p.join()

    t2 = time.time()                         # Кінцевий час
    print("Друга частина:", t2-t1, "секунд.")
```

У цьому фрагменті коду кінцевий результат на екран не виводиться. Точніше, виводиться тільки час виконання другої частини. Однак програма робить свою справу і замінює необхідні елементи масиву.

Для того, щоб переконатись, що програма працює правильно, можна додати в кінець програми команди виведення початкового і обробленого масивів на екран:

```
    print("\nКінцевий результат:\n")

    print()
    for i in mas:
        print(i)

    print()
    for i in arr:
        print(i)
```

Але ці команди рекомендується використовувати з масивами невеликих розмірів (приблизно 10 * 10) і тільки з метою перевірки правильності роботи програми. Перед обробкою масивів великих розмірів ці команди краще закоментувати або прибрати.

Нижче наведений повний лістинг обох частин програми. Функція *f*1 позначена синім фоном, функція *f*2 – зеленим фоном, перша частина коду головного процесу – жовтим фоном, друга частина коду головного процесу – рожевим фоном.

```
import random
import multiprocessing as mp
import time

def f1(start_line, end_line, arr, n10, L):    # Пошук у підмасиві

    sub_mas = arr[start_line:end_line]        # Отримання підмасиву

    M = len(sub_mas)                        # Кількість рядків підмасиву
    N = len(sub_mas[0])                     # Кількість стовпців підмасиву

    k = 0                                   # Кількість знайдених елементів

    for i in range(0, M):
        for j in range(0, N):                   # Цикл за стовпцями
            n = sub_mas[i][j]                   # Беремо елемент масиву
            n = abs(n)
            s = str(n)                          # Переводимо в символьний формат

            m = 0                               # Початкове значення суми цифр
            for c in s:                         # Перебирання символів рядка
                m += int(c)                     # Додаємо цифру до суми

            if m == 10:                         # Якщо сума цифр дорівнює 10
                k = k + 1                       # Збільшуємо кількість

    L.acquire()
    n10.value = n10.value + k           # Додаємо результат до глоб. змінної
    L.release()
```

```
def f2(start_line, end_line, arr, n10, L):  # Заміна елементів підмасиву

    sub_mas = arr[start_line:end_line]      # Отримання підмасиву

    M = len(sub_mas)                        # Кількість рядків підмасиву
    N = len(sub_mas[0])                     # Кількість стовпців підмасиву

    for i in range(0, M):
        for j in range(0, N):               # Цикл за стовпцями
            n = sub_mas[i][j]               # Беремо елемент масиву

            if n > 0:                          # Якщо число додатне
                if n ** 0.5 == int(n ** 0.5):  # Якщо корінь є цілим числом
                    sub_mas[i][j] = n10.value  # Робимо заміну на n10

        L.acquire()
        arr[i+start_line] = sub_mas[i]      # Оновлюємо черговий рядок
        L.release()


if __name__ == "__main__":                  # Початок головного процесу
    mp.freeze_support()                     # Хай буде

    M, N = 10, 10                           # Розміри масиву

    L = mp.Lock()                           # Замок

    mas = []                                # Підготовка масиву рядків
    for i in range(0, M):                   # Цикл за рядками
        mas_i = []                          # Підготовка порожнього рядка
        for j in range(0, N):               # Цикл за стовпцями
            n = random.randint(-1000, 1000) # Генерація рандомного числа
            mas_i.append(n)                 # Додавання числа в рядок
        mas.append(mas_i)                   # Додавання рядка у масив


    # Перша частина задачі
    print("Перша частина розпочала роботу.")

    t1 = time.time()                        # Початковий час

    man = mp.Manager()                      # Менеджер процесів
    arr = man.list(mas)                     # Список у межах менеджера

    n10 = mp.Value("i", 0)                  # Кількість елементів

    p_num = mp.cpu_count()                  # Кількість процесів

    # Розраховуємо розмір блоку одного завдання
    # (кількість рядків матриці, що надсилаються одному процесу)
    block_size_float = M / p_num            # Дробова кількість рядків
    block_size = int(block_size_float)      # Ціла кількість рядків
    if block_size != block_size_float:      # Якщо вони не рівні
        block_size = int(block_size) + 1    # Округлення до більшого цілого

    p_list = []                             # Підготовка списку процесів

    start_line = 0                          # Початковий рядок блоку
    end_line = 0                            # Кінцевий рядок блоку
    while end_line < M :   # Поки не усі рядки розподілені

        end_line = start_line + block_size  # Номер кінцевого рядка + 1

        if end_line > M:      # Щоб не перестрибнути останній рядок
```

```
        end_line = M

    # Створення процесу і додавання в список процесів
    p = mp.Process(target=f1, args=(start_line, end_line, arr, n10, L))
    p_list.append(p)

    start_line =  end_line;             # Кінцевий рядок стає початковим


for p in p_list:                        # Цикл за запуском процесів
    p.start()

for p in p_list:                        # Цикл за очікуванням процесів
    p.join()

print("Знайдено чисел:", n10.value)

t2 = time.time()                        # Кінцевий час
print("Перша частина:", t2-t1, "секунд.")


# Друга частина задачі
print("\nДруга частина розпочала роботу.")

t1 = time.time()                        # Початковий час

p_list = []                             # Підготовка списку процесів

start_line = 0                          # Початковий рядок блоку
end_line = 0                            # Кінцевий рядок блоку
while end_line < M :   # Поки не усі рядки розподілені

    end_line = start_line + block_size  # Номер кінцевого рядка + 1

    if end_line > M:      # Щоб не перестрибнути останній рядок
        end_line = M

    # Створення процесу і додавання в список процесів
    p = mp.Process(target=f2, args=(start_line, end_line, arr, n10, L))
    p_list.append(p)

    start_line =  end_line;             # Кінцевий рядок стає початковим

for p in p_list:                        # Цикл за запуском процесів
    p.start()

for p in p_list:                        # Цикл за очікуванням процесів
    p.join()

t2 = time.time()                        # Кінцевий час
print("Друга частина:", t2-t1, "секунд.")

print("\n\nКінцевий результат:\n")

print()
for i in mas:
    print(i)

print()
for i in arr:
    print(i)
```

Для масиву розміром 10 * 10 результат роботи програми буде приблизно таким:

```
Перша частина розпочала роботу.
Знайдено чисел: 8
Перша частина: 0.1914963722229004 секунд.

Друга частина розпочала роботу.
Друга частина: 0.10509109497070312 секунд.


Кінцевий результат:

[-735, 841, -895, -187, 165, -726, 518, -10, -268, -842]
[132, 276, 328, -547, 884, -522, 714, -179, 79, 784]
[728, 428, 181, -544, -137, -832, -118, -799, -169, 640]
[188, 414, 950, 430, 311, -200, 900, -63, 503, 185]
[-127, -359, -508, 570, -166, -623, 695, -331, -637, -126]
[13, 749, 282, -289, 458, 848, 682, -303, -991, -847]
[94, 898, -672, -737, 591, 453, -398, 551, -419, -787]
[140, 323, 289, 847, -313, -603, 433, -137, -165, 949]
[964, 871, -166, 112, -503, -946, 316, -117, 253, -442]
[200, -302, -942, -758, -657, -815, -603, 950, -98, 138]

[-735, 8, -895, -187, 165, -726, 518, -10, -268, -842]
[132, 276, 328, -547, 884, -522, 714, -179, 79, 8]
[728, 428, 181, -544, -137, -832, -118, -799, -169, 640]
[188, 414, 950, 430, 311, -200, 8, -63, 503, 185]
[-127, -359, -508, 570, -166, -623, 695, -331, -637, -126]
[13, 749, 282, -289, 458, 848, 682, -303, -991, -847]
[94, 898, -672, -737, 591, 453, -398, 551, -419, -787]
[140, 323, 8, 847, -313, -603, 433, -137, -165, 949]
[964, 871, -166, 112, -503, -946, 316, -117, 253, -442]
[200, -302, -942, -758, -657, -815, -603, 950, -98, 138]
```

У першому (початковому) масиві блакитним фоном позначені елементи, сума цифр яких дорівнює 10. Таких елементів було знайдено 8, що показано вище як результат роботи першої частини програми.

Друга частина програми замінила усі елементи, що є квадратами цілих чисел, на знайдене число 8. Такі елементи позначені жовтим фоном, і в наведеному прикладі їх було знайдено чотири.

Якщо закоментувати останні сім рядків коду та запустити програму для значень $M = 6850$, $N = 6850$, результат буде приблизно таким:

```
Перша частина розпочала роботу.
Знайдено чисел: 2955892
Перша частина: 5.2338128089904785 секунд.

Друга частина розпочала роботу.
Друга частина: 3.57606840133667 секунд.
```

Хоча програма виводить значення часу роботи першої та другої частин окремо, це зроблено лише з метою профілювання. Загальний час роботи програми розраховується як сума цих значень і для наведеного прикладу сягає приблизно 8,8 секунд (у разі використання 20 логічних процесорів).

Наскільки час роботи програми залежить від кількості процесорів, ми дослідимо нижче, а зараз зазначимо, що без використання розпаралелювання еквівалентна програма виконувалась приблизно 30 секунд, у разі розпаралелювання за допомогою механізму потоків – приблизно 20 секунд. Отже, маємо хоч і не надто значну, але очевидну економію часу виконання програми.

Розглянута програма та аналіз коректності її роботи відповідають пунктам 2.1–2.4 завдання до лабораторної роботи. Тепер виконаємо пункти 2.5–2.7, використавши для цього комп'ютер на базі процесора Intel 13500 з 64 Гб оперативної пам'яті.

2.5. Згідно з результатами виконання лабораторної роботи № 1, таблиця 3 для нашого випадку наведена нижче.

Таблиця 3 – Результати виміру часу роботи програми без розпаралелювання

Час виконання програми (час t_1)	**Розміри масиву**
1 секунда	1230 * 1230
2 секунди	1770 * 1770
5 секунд	2800 * 2800
10 секунд	3950 * 3950
30 секунд	6850 * 6850

На основі таблиці 3 сформуємо таблицю 6, у якій перший стовпець повторює другий стовпець таблиці 3 і відповідає розмірам масиву, для яких було проведено експеримент. В інших стовпцях вказуються значення часу (в секундах), отримані для різної кількості використовуваних процесів. Значення P будемо вважати рівним 20. Даний час будемо позначати через t_3.

Таблиця 6 – Час t_3 виконання програми при різній кількості процесів, секунд

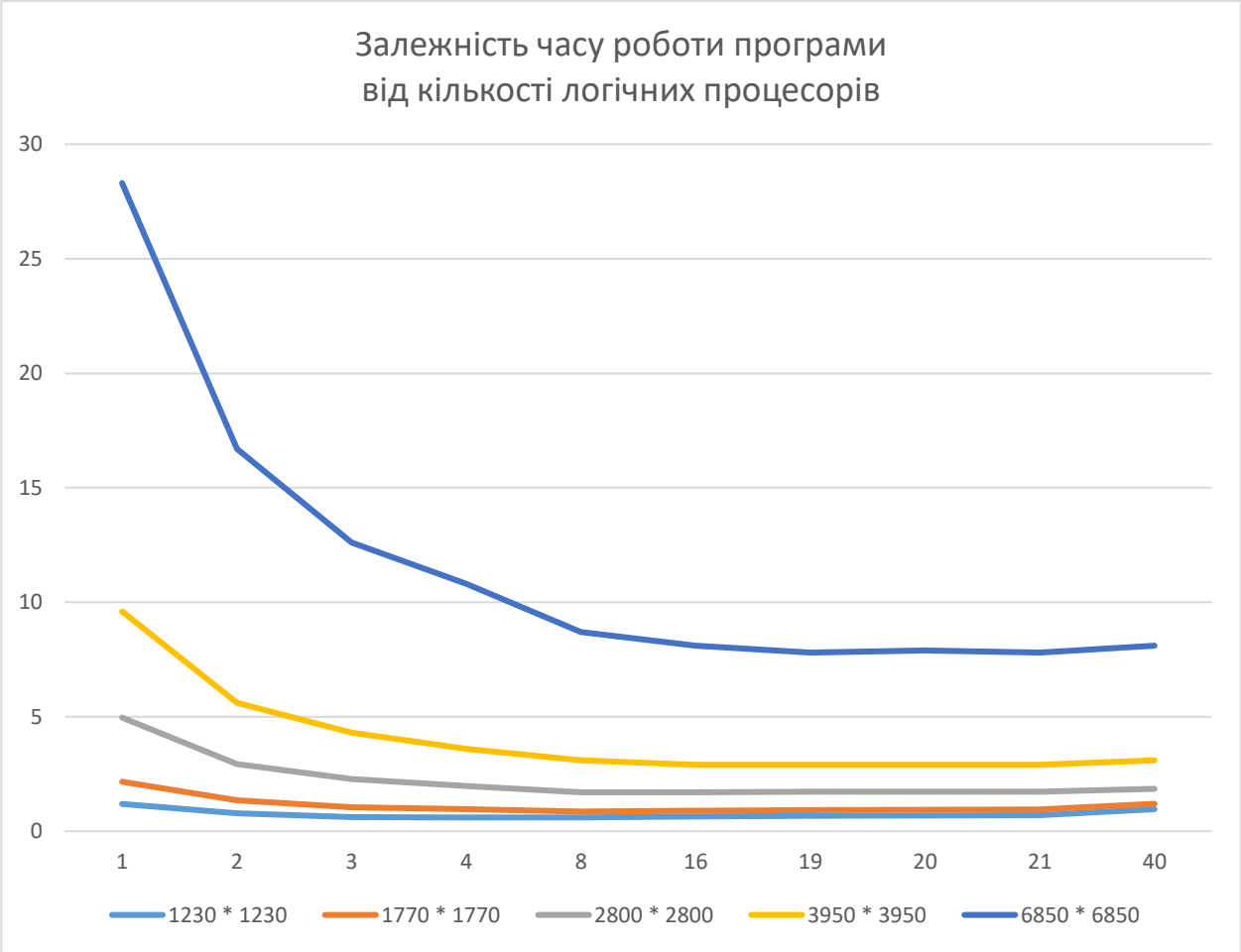
Розміри масиву	**Кількість задіяних процесів**									
	1	**2**	**3**	**4**	**8**	**16**	**19**	**20**	**21**	**40**
1230 * 1230	1,19	0,78	0,62	0,60	0,61	0,65	0,67	0,69	0,70	0,96
1770 * 1770	2,16	1,35	1,05	0,96	0,86	0,89	0,92	0,93	0,95	1,20
2800 * 2800	4,96	2,93	2,28	1,97	1,70	1,70	1,73	1,73	1,73	1,85
3950 * 3950	9,6	5,6	4,3	3,6	3,1	2,9	2,9	2,9	2,9	3,1
6850 * 6850	28,3	16,7	12,6	10,8	8,7	8,1	7,8	7,9	7,8	8,1

Для підвищення точності експериментів було використано команду:

```
random.seed(25)
```

Ця команда забезпечила однаковий вміст масиву для кожного рядка таблиці 5, що зробило час виконання незалежним від рандомного вмісту масиву.

Нижче наведений графічний вигляд вмісту таблиці 5. Ці графіки отримані за допомогою Microsoft Excel. Таблиця 5 та її графічне представлення дають змогу зробити такі висновки.



1. Збільшення кількості використовуваних процесорних ядер сприяє зменшенню часу виконання програми.

2. Для розглянутої задачі використання більш ніж 8 ядер не дає значного приросту швидкодії. Це можна пояснити тим, що розв'язувана задача передбачає активний обмін даними між дочірніми і головним процесами за допомогою менеджера процесів. Чим більша кількість ядер, тим більше звертань до менеджера процесів, що нівелює виграш у часі від збільшення кількості ядер.

3. Коли кількість дочірніх процесів стає більшою за кількість доступних ядер процесора, швидкодія програми починає повільно знижуватись. Це є наслідком того, що велика кількість процесів все одно обробляється лише наявними фізичними ядрами, тоді як збільшення кількості процесів передбачає додаткові витрати часу на їх створення, запуск та очікування.

2.6. Для масиву розміром 6850 * 6850 дослідимо час виконання програми для різної кількості задіяних процесорних ядер (логічних процесорів), яка не перевищує фізичну кількість ядер у використовуваному процесорі (у нашому випадку – 20).

Наприклад, були отримані такі результати:

Кількість задіяних логічних процесорів	1	2	3	4	5	6	7	8	9	10
Час виконання програми, с	28,9	16,7	12,9	10,9	9,9	9,5	9,1	8,8	8,7	8,5

Кількість задіяних логічних процесорів	11	12	13	14	15	16	17	18	19	20
Час виконання програми, с	8,5	8,4	8,3	8,2	8,2	8,1	8,1	8,2	8,2	8,2

Отримані результати дають змогу визначити, що для розглянутої задачі та використовуваного процесора оптимальною може вважатись кількість ядер, рівна 16. Це допомагає, по-перше, отримати найменший час виконання програми, а по-друге, залишити 4 ядра для потреб операційної системи та інших програм.

2.7. Зберемо в одній таблиці значення часу розв'язання задачі для таких випадків:

– розпаралелювання не використовується взагалі (час t_1, табл. 3 з лабораторної роботи № 1);

– використовується розпаралелювання на основі потоків (час t_2, табл. 4 з лабораторної роботи № 1);

– використовується розпаралелювання на основі процесів (час t_3, значення зі стовпця «16» таблиці 6 цієї лабораторної роботи).

В результаті отримаємо таблицю 9.

Таблиця 9 – Порівняння ефективності різних способів розпаралелювання

Розміри масиву	**Час t_1, секунд**	**Час t_2, секунд**	**Час t_3, секунд**
1230 * 1230	1	0,80	0,65
1770 * 1770	2	0,96	0,89
2800 * 2800	5	3,52	1,70
3950 * 3950	10	6,84	2,9
6850 * 6850	30	20,5	8,1

Аналіз вмісту табл. 9 дає змогу зробити такі висновки.

1. Спосіб розпаралелювання на основі процесів є більш ефективним, ніж реалізація алгоритму з розпаралелюванням на основі потоків та без розпаралелювання.

2. Ефективність застосування розпаралелювання зростає зі збільшенням розмірів масиву.

3. Збільшення кількості використовуваних ядер з 1 до 16 не приводить до зменшення часу виконання програми у 16 разів. Для розглянутої задачі та масиву розміром 6850 * 6850 час виконання програми на 16 процесорних ядрах є у 3.75 рази меншим, ніж під час виконання програми на одному ядрі.

Висновки до лабораторної роботи

Розпаралелювання на основі процесів є ефективним засобом зменшення часу виконання програм, які активно використовують центральний процесор комп'ютера. Однак ефект від розпаралелювання є неоднозначним і повною мірою залежить від поставленої задачі та алгоритму її реалізації. Отримані під час експериментів числові значення, наведені вище, є справедливими виключно для задачі, що розв'язувалась, та використовуваного обладнання (процесора та оперативної пам'яті комп'ютера). У нашому випадку найкращий результат дало використання розпаралелювання на основі процесів, виконане на 16 процесорних ядрах, яке дало зменшення часу виконання програми у 3.75 рази для масиву максимально дослідженого розміру порівнянно з лінійним розв'язанням задачі (без розпаралелювання).

ІНДИВІДУАЛЬНЕ ТВОРЧЕ ЗАВДАННЯ

Навчальна дисципліна «Технології розподілених та паралельних обчислень» викладається на четвертому курсі, після завершення якого передбачені виконання і захист кваліфікаційної бакалаврської роботи. Невід'ємним складником кваліфікаційної роботи є презентація, яка допомагає здобувачеві зробити якісну доповідь та сприяє кращому розумінню отриманих результатів.

Технічно презентація робиться за допомогою програми PowerPoint або подібної (Google Slides, Prezi, Canva) і являє собою певний набір слайдів. У ДонНУ імені Василя Стуса та на факультеті інформаційних і прикладних технологій розроблені методичні рекомендації щодо оформлення кваліфікаційних робіт, які, зокрема, регламентують вимоги до оформлення презентації. Однак якісна презентація – це не лише дотримання вимог, але й підхід до подання матеріалу, який, зокрема, передбачає таке.

1. Правильне визначення цільової аудиторії.
2. Чітке висвітлення основної ідеї та ключових моментів доповіді.
3. Вдалий вибір візуального стилю, кольорової гами, шрифтів, анімацій тощо.
4. Оптимальне співвідношення кількості слайдів та їх інформаційної завантаженості.

Чим більше презентацій підготує здобувач освіти до захисту кваліфікаційної роботи, тим більше досвіду він отримає і тим кращою буде його презентація під час захисту роботи. Чим більше доповідей він зробить, тим якіснішою буде його доповідь на захисті і тим кращим буде підсумковий результат.

Отже, метою індивідуального творчого завдання з дисципліни «Технології розподілених та паралельних обчислень» є підготовка презентації за результатами виконаних лабораторних робіт.

Завдання

За допомогою програми PowerPoint або їй подібної розробити презентацію виконаних лабораторних робіт відповідно до індивідуального варіанту завдання.

Нижче наведені основні вимоги та рекомендації до підготовки презентації, яких треба дотримуватись. Все, що не обмежене вимогами, належить до творчого складника завдання і виконується здобувачем освіти на власний розсуд. Докладні критерії оцінювання індивідуального творчого завдання наведені нижче.

Вимоги та рекомендації до презентації

1. Презентація повинна бути оформлена українською мовою на основі шаблону ДонНУ імені Василя Стуса, в якому кожен слайд містить спеціальні колонтитули та фоновий герб університету. Файл шаблону можна отримати у викладача.

2. Перша частина слайдів має бути присвячена лабораторній роботі № 1, друга частина – лабораторній роботі № 2. Обсяг слайдів для представлення кожної лабораторної роботи – приблизно 10. Загальний обсяг презентації – приблизно 20 слайдів.

3. Перший слайд має бути титульною сторінкою презентації. На ньому треба розмістити інформацію про назву дисципліни та лабораторні роботи, а також ПІБ та академічну групу автора презентації. Внизу по центру треба вказати слово «Вінниця» і поточний рік.

4. Останній слайд повинен містити фразу «Дякую за увагу!» або «Доповідь завершено. Дякую за увагу!».

5. Кожен проміжний слайд повинен мати назву. Назва слайда повинна розташовуватися зверху по центру і містити 3–5 слів. Розмір шрифту назви слайда повинен бути трохи більшим за розмір шрифту інших текстових матеріалів на слайді.

6. Усі слайди повинні нумеруватись. Найкраще розташування номера слайда – в правому нижньому куті. Титульний слайд вважається слайдом № 1, однак на ньому номер не ставиться. Наступний після титульного слайд буде містити номер 2. Значок номера «№» не ставиться, вказується просто число. Розмір шрифту треба обирати таким, щоб номер слайда було добре видно.

7. Передостанній слайд повинен містити узагальнені висновки за обома лабораторними роботами. Достатньо 2–3 висновків, по 2–3 рядки кожен. Розмір шрифту і розташування мають бути такими, щоб висновки займали більшу частину слайда. Вгорі слайда має бути назва «Висновки». Не треба додавати на слайд з висновками якісь графічні картинки або зображення. Також не треба робити у презентації слайди з висновками за кожною лабораторною роботою окремо. Тільки цей підсумковий слайд.

8. Не треба розташовувати на слайдах багато матеріалу у текстовій формі. Варто пам’ятати, що під час демонстрації презентації доповідач усно надає всю необхідну «текстову» інформацію. Слайди мають містити лише ті матеріали, які у текстовій формі виразити складно – рисунки, графіки, таблиці, знімки екрана, фрагменти програмного коду. Не треба змушувати аудиторію читати на слайдах багато тексту – це відволікає увагу від слухання, втомлює і врешті-решт роздратовує. Коротенькі текстові пояснення використовувати можна, але лише поруч із ілюстративними матеріалами. Вкрай небажаними є слайди, що містять багато рядків тексту і жодних ілюстративних матеріалів (хоча іноді без таких слайдів не обійтись).

9. Враховуючи матеріал лабораторних робіт, у презентації рекомендується висвітлити такі елементи:

– індивідуальні варіанти завдань (тематика та перелік сайтів, формулювання другого завдання);

– один-два приклади виконання другого завдання «вручну» для масивів невеликого розміру (3 * 3 або 4 * 4) з поясненням результатів обробки;

– пояснення алгоритмів програм (у вигляді блок-схем, UML-діаграм або текстового опису функцій та структур даних);

– декілька найбільш важливих фрагментів програмного коду з необхідними коментарями (повний лістинг у жодному разі не наводити);

– декілька скриншотів із результатами роботи програм (бажано, щоб був темний текст на білому фоні);

– таблиці з результатами виміру часу виконання програм, а також дані про використовуване апаратно-програмне середовище (процесор, обсяг оперативної пам'яті, операційна система, версія мови Python);

– представлення вмісту таблиць у вигляді графіків або діаграм (за бажанням).

Послідовність розташування перелічених елементів повинна бути такою, щоб спочатку була зроблена повна доповідь про першу лабораторну роботу, потім – про другу лабораторну роботу, а потім підбиті порівняльні підсумки обох робіт.

10. У кожен слайд має бути впроваджений аудіофрагмент із частиною доповіді, що стосується цього слайда. Тобто презентація повинна містити в собі дикторський супровід. В реальному житті такий супровід зазвичай не буде використовуватись, оскільки автор презентації буде доповідати самостійно. Однак за певних умов наявність аудіодоповіді може бути корисною. Доповідь повинна бути саме доповіддю, а не озвучуванням вмісту слайда.

Варто знати, що останнім часом почастішали випадки шахрайства, коли зловмисники отримують якимось шляхом голос людини (наприклад, через телефонний дзвінок), після чого навчають нейронну мережу говорити схожим голосом. Далі зловмисники використовують навчену нейронну мережу для «спілкування» зі знайомим цієї людини з метою заволодіння грошима, паролями та іншими конфіденційними речами, цінностями тощо.

У зв'язку з цим дозволяється (навіть рекомендується) використовувати для озвучення презентації не власний голос, а голос, синтезований комп'ютером за текстовим описом. Для цього можна використовувати різноманітні сервіси, які доступні в мережі Інтернет. Аудіодоповідь повинна бути зроблена українською мовою. Тривалість доповіді – в середньому від пів хвилини до хвилини на слайд.

11. Текст слайдів повинен мати переважно чорний колір. Ілюстративні матеріали повинні бути кольоровими, але такими, щоб виглядали розбірливо та естетично. Для підготовки ілюстративних матеріалів можна використовувати будь-які доступні та знайомі програмні продукти (Word, Excel, Paint, Photoshop тощо).

12. Не рекомендується використовувати візуальні ефекти під час перегортання слайдів. Також не треба налаштовувати автоматичне перегортання слайдів за таймером. Слайди повинні перегортатись лише вручну.

Структура звіту з індивідуального творчого завдання

Головним результатом виконання ІТЗ є файл презентації. Для захисту ІТЗ і для виставлення оцінки необхідно підготувати звіт. Він оформлюється на листах формату А4 (з однієї сторони листа) та містить такі складники:

1. Титульний лист, оформлений відповідно до стандартів Університету.

2. Індивідуальний варіант завдання (тематика сайтів та задача з обробки двовимірного масиву).

3. Усі слайди презентації, розгорнуті в альбомному форматі (один слайд на сторінці). Для слайдів достатньо звичайного чорно-білого друку, кольоровий друк не потрібен (хоча не забороняється).

4. Висновки до індивідуального творчого завдання (2–3 речення на окремому листі).

Критерії оцінювання індивідуального творчого завдання

Під час оцінювання ІТЗ окремо розглядаються оформлення презентації та її зміст. Кожен із цих складників оцінюється незалежно від іншого в діапазоні від 0 до 20 балів. Загальна оцінка ІТЗ – від 0 до 40 балів.

Нижче наведені критерії оцінювання кожного зі складників. Кожен критерій оцінюється від 0 до 5 балів на розсуд викладача.

Оформлення		
Критерій	**Кількість балів**	**Коментар**
1. Наповненість слайдів	0–5	Слайди не перевантажені інформацією. Кожен слайд має заголовок та номер. Наявні всі структурні складові презентації – титульний лист, висновки тощо. На слайдах відсутні значні обсяги вільного місця
2. Візуальна чіткість та кольорова гама	0–5	Текст та ілюстративні матеріали є чіткими, розбірливими, не надто дрібними. Використана кольорова гама дає змогу добре розрізняти матеріал на білому фоні листа, допомагає виділяти найбільш важливі елементи слайда, повною мірою гармонує з кольорами брендбуку ДонНУ імені Василя Стуса. Однотипні елементи слайдів (заголовки, нумерація тощо) мають єдиний стиль оформлення
3. Текстовий складник	0–5	Текст та ілюстративні матеріали не містять орфографічних, пунктуаційних та стилістичних помилок. Речення та назви починаються з великої букви. Усі англомовні та сленгові терміни замінені на україномовні аналоги. Не використовуються скорочення (окрім загальновідомих) та перенос слів
4. Аудіодоповідь	0–5	Аудіодоповідь є послідовною. Мова є чіткою, культурною, в середньому темпі. В доповіді зазвичай не повторюються текстові написи, розміщені на слайдах (окрім назв слайдів)

Зміст		
Критерій	**Кількість балів**	**Коментар**
5. Послідовність викладення матеріалу	0–5	Слайди розташовані в такому порядку, щоб утворювати єдину послідовну доповідь про виконання лабораторних робіт та отримані результати. Порядок подання матеріалу загалом співпадає з порядком виконання лабораторних робіт
6. Корисність матеріалу	0–5	Кожен слайд містить матеріал, необхідний для розуміння наступних слайдів. Фрагменти програмного коду демонструють тільки найбільш важливі деталі. Для графіків обрано зручний масштаб. Приклади виконання алгоритмів та результати роботи програм зрозумілі та добре прокоментовані
7. Формальна коректність	0–5	Для всіх формул наведене пояснення їх компонент та змінних. Усі графіки мають підписані координатні осі з позначенням одиниць виміру. Усі діаграми, таблиці, рисунки мають назви (можливо, і номери) та містять усі необхідні підписи, легенди тощо
8. Потенційні запитання	0–5	Вміст слайдів разом із доповіддю сприяють виникненню якомога меншої кількості змістовних питань з боку аудиторії. Усі зроблені висновки є однозначними, чіткими, підкріпленими числовими показниками. В процесі перегляду презентації не виникає потреби у поверненні до попередніх слайдів та їх повторному перегляді

Презентація, яка висвітлює тільки одну з двох лабораторних робіт, оцінюється за цими ж критеріями, але у половинному діапазоні балів (від 0 до 20). Презентація, яка не повністю висвітлює навіть одну лабораторну роботу або зроблена відповідно до чужого варіанта завдання, до захисту не приймається і оцінюється в нуль балів.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна література

1. Коцовський В. М. Теорія паралельних обчислень: навч. посіб. Ужгород: ПП «АУТДОР-Шарк», 2021. 188 с.

2. Минайленко Р. М. Паралельні та розподілені обчислення: навч. посіб. Кропивницький: Видавець Лисенко В. Ф., 2021. 153 с.

3. Ваврук Є. Організація паралельних обчислень: навч. посіб. / укл. Є. Ваврук, О. Лашко. Львів: Національний університет «Львівська політехніка», 2007. 70 с.

4. Програмування числових методів мовою Python: підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий; за ред. А. В. Анісімова. Київ: Видавничо-поліграфічний центр «Київський університет», 2014. 640 с.

5. Лісовенко І. Д., Яковлєва І. Д. «Паралельні та розподілені обчислення»: навч. посіб. Чернівці: ЧНУ, 2022. 120 с. (електронне видання).

6. Качко О. Г. Паралельне програмування. Харків. нац. ун-т радіоелектроніки. Харків: ХНУРЕ, 2016. 403 с.

7. Лазарович І. М. Паралельні обчислювальні середовища. Лабораторний практикум. *Видавництво Прикарпатського національного університету імені Василя Стефаника*. Івано-Франківськ, 2014. 65 с.

Допоміжна література

1. Lutz M. Learning Python, 5th Edition. O'Reilly Media Inc., 2013. 1648 p.

2. Мельник А. О., Яковлєва І. Д. Структурний аналіз і синтез паралельних алгоритмів: монографія. Чернівці: Чернівецький нац. ун-т, 2018. 184 с.

3. Семеренко В. П. Технології паралельних обчислень: навч. посіб. Вінниця: ВНТУ, 2018. 104 с.

4. Burns B. Designing Distributed Systems. Sebastopol: O'Reilly Media, 2018. 149 p.

Інформаційні ресурси в мережі Інтернет

1. Python Software Foundation. The Python Tutorial. URL: https://docs.python.org/3/tutorial/index.html

2. Українська команда розподілених обчислень. URL: https://distributed.org.ua/

3. A Guide to Python Multiprocessing and Parallel Programming. URL: https://www.sitepoint.com/python-multiprocessing-parallel-programming/

4. Parallel Processing in Python. URL: https://www.geeksforgeeks.org/parallel-processing-in-python/

5. Multiprocessing – Process-based parallelism. URL: https://docs.python.org/3/library/multiprocessing.html

Навчальне видання

Бабаков Роман Маркович

Методичні рекомендації
до виконання лабораторних робіт з дисципліни

ТЕХНОЛОГІЇ РОЗПОДІЛЕНИХ ТА ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

для здобувачів освіти ОС «Бакалавр» денної форми навчання
спеціальності 122 Комп’ютерні науки

Редактор О. А. Солдатова
Технічний редактор Т. О. Важеніна-Гопрак

Підписано до друку 16.12.2024
Формат 60 × 84/16. Папір офсетний.
Друк – цифровий. Умовн. друк. арк. 4,41.
Тираж 30. Зам. 60.

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