МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
КАФЕДРА СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

Розрахункова робота
з дисципліни
«Технології розподілених систем та паралельних обчислень»
на тему
«Паралелізація веб-скрапінгу»

Виконали:
студенти групи КН-308
Гецянин Д. Р.
Келемен С. Й.
Викладач:
Морчурад Л. І.

Львів – 2020 р.

ЗМІСТ

ВСТУП

У сучасному світі проблема збору, фільтрування і обробки даних як ніколи актуальна. Величезні обсяги цінної інформації додаються, оновлюються та зберігаються у Всесвітньому павутинні [1]. Пошукові роботи-павуки повзають (англ. crawl) по ньому, індексуючи та роблячи можливою зручну навігацію [26]. Завдяки цьому і з'явилося поняття веб-скрапінгу (англ. web scraping), яке досліджуватиметься у нашій роботі.

Даний процес полягає у збереженні і структуризації інформації з веб-сторінок, які призначені для використання людиною через веб-браузер. Спочатку ресурс завантажується (так само, як при перегляді сторінки), а потім розпочинається процес видобування даних. Він може здійснюватись вручну, із використанням різноманітних мов програмування (їх бібліотек) чи за допомогою готових програмних засобів. Отримана інформація зберігається в базах даних, електронних таблицях чи текстових файлах для подальшого аналізу та використання в інших цілях, наприклад, для порівняння цін, накопичення контактних даних людей, проведення наукових досліджень, загального моніторингу змін на веб-сайтах та безлічі інших задач [3].

Веб-скрапінг відноситься до технологій, що інтенсивно розвиваються. Подальше удосконалення роботи алгоритмів для вирішення цієї задачі потребує активних досліджень у сфері обробки текстових даних, штучного інтелекту, семантичного аналізу та людино-машинної взаємодії загалом. Крім того, для збільшення ефективності процесу добування інформації часто застосовують паралельні обчислення [2].

Існує велике різноманіття засобів для здійснення веб-скрапінгу: від ручного збирання інформації до автоматизованих систем, що можуть завантажити і обробити цілий веб-сайт. Кожен ресурс потребує в певній мірі індивідуального підходу, адже в кожному окремому випадку дані структуруються по-різному. Через це використовуються методи з різними підходами та функціональними можливостями [14].

Метою даної розрахункової роботи є проведення веб-скрапінгу сторінок з використанням технологій паралельних обчислень. Буде експериментально перевірено ряд гіпотез, виведених у відповідному порядку з попередніх чотирьох абзаців. Отже, вони передбачають:

1. Можливість ефективного застосування методу веб-скрапінгу для добування корисної інформації в Інтернеті.
2. Використання мов програмування та спеціальних бібліотек для автоматизованого завантаження, обробки та структуризації даних.
3. Збільшення продуктивності даного процесу за допомогою розпаралелювання.
4. Важливість вибору підходящого методу для конкретного випадку застосування.

1 АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

На даний момент розроблено тисячі рішень для веб-скрапінгу з різними рівнем складності реалізації та сценарієм використання. Тому, аналізуючи наявні підходи, краще розглядати не окремі технології, а узагальнену класифікацію за основним принципам дії. Отже, умовно можна виділити такі методи:

- Копіювання і вставка в ручному режимі

  Найпростіший спосіб веб-скрапінгу, який полягає у копіюванні даних з веб-сторінки у текстовий файл або електронну таблицю. Його незмінна актуальність зумовлена тим, що навіть найефективніший алгоритм іноді не може замінити людську здатність до аналізу та тонких ручних маніпуляцій. Крім того, поширеним випадком використання є ситуація, коли веб-сайт навмисно запобігає машинній автоматизації тестом CAPTCHA [8]. До переваг даного методу можна віднести високу релевантність, швидкість пошуку та якість отримуваного контенту. До недоліків – необхідність спеціальних знань і навичок пошуку в Інтернеті для використання, обмежену можливість отримання великої кількості якісних результатів (до декількох сотень в день), а також непередбачуваний людський фактор загалом. [24]

- Шаблони

  Досить простий, і водночас ефективний метод отримання інформації з веб-сторінок. У ньому застосовуються функції для знаходження тексту за допомогою регулярних виразів (шаблонів), які доступні у більшості мов програмування. Основним плюсом є те, що даний метод дозволяє позбутися невеликих помилок в результаті, не змінюючи основний контент (наприклад, очистити залишки зайвого HTML-коду). До мінусів можна віднести відносну складність застосування та потребу змінювати регулярні вирази при зміні HTML-розмітки. [27]

- HTTP-програмування

  При застосуванні даного способу динамічні та статичні дані отримуються за допомогою відправлення HTTP-запиту на веб-сервер із

використанням програмування сокетів [19]. Перевагами є отримання коду сторінки у виді HTTP-запитів та відповідно накопичення великої кількості контенту, яка обмежена лише ресурсами веб-серверу та швидкістю Інтернету. Основним недоліком – те, що більшість веб-серверів мають захист від повторюваних запитів і здатні закрити доступ до сайту. [16]

- HTML-парсинг

  Багато веб-ресурсів зберігають великий набір сторінок, які динамічно генеруються із бази даних. Дані однакової категорії зазвичай структуровані схожим чином за допомогою використання загального скрипту або шаблону. Програма, яка добуває інформацію на основі виявлення шаблонів, називається обгорткою (англ. wrapper). Вона проводить аналіз, завантаження та зберігання потрібних даних з веб-сайту. По суті, такий підхід є узагальненим алгоритмом роботи найбільш використовуваних сучасних інструментів і, порівняно з іншими, оптимально швидкий, зручний та низько затратний [20]. Технології саме на основі HTML-парсингу ми і будемо використовувати у даній роботі.

- DOM-аналіз

  Алгоритм вбудовується у веб-браузер для отримання динамічних даних, які є результатом роботи скриптів на клієнтській (браузерній) стороні. Це дає змогу отримати частини веб-сторінки, оскільки сам веб-браузер аналізує DOM-дерево [11] для його коректного відображення користувачу. Плюсами є можливість автоматизації процесу і отримання якісного контенту у великих кількостях, а мінусами – висока складність реалізації і, відповідно, значні затрати на розробку. [15]

- Програмне забезпечення для веб-скрапінгу

  Існує багато готових програмних рішень для вирішення даної завдання. Вони можуть автоматично розпізнавати розмітку веб-сторінки, забезпечувати добування та обробку контенту, а потім зберігати отриману інформацію в базу даних – і це все без потреби написання

програмного коду вручну. Основним плюсом такого підходу є простота реалізації і зручність використання навіть непрофесійним користувачем, а очевидним мінусом – висока вартість програмного забезпечення. [7]

Наприклад, є декілька платформ, спеціально розроблених для вертикальної агрегації. Вони автоматично генерують і контролюють велику кількість «ботів» для кожного завдання без використання людських ресурсів та прив'язки до конкретного веб-сайту. Ефективність такої системи виражається у якості отриманої інформації (як правило, кількість полів) і масштабованості (наскільки швидко платформа адаптується для роботи з сотнями або тисячами веб-ресурсів) [25].

- Комп'ютерне бачення веб-сторінок

Застосування машинного навчання та комп'ютерного бачення для ідентифікації і добування інформації з веб-сторінок, інтерпретуючи їх візуально так, як це б робила людина. Перевагами є висока якість отримуваного результату і здатність працювати зі сторінками, де відсутній доступ до внутрішньої структури, а основними недоліками – складна та дороговартісна реалізація. [12]

2 МАТЕРІАЛИ ТА МЕТОДИ

Основним підходом до веб-скрапінгу у нашій роботі є HTML-парсинг за допомогою мови Python [21] з бібліотекою Beautiful Soup [6] та фреймворком Scrapy [22].

Спочатку розглянемо Beautiful Soup, оскільки це простіший метод для невеликих задач, які полягають у завантаженні, очистці та збереженні інформації з веб-сторінок. По суті, це просто набір функцій, класів, структур, змінних тощо, які використовуються для зручного веб-скрапінгу. Він має досить невисоку продуктивність, якщо не використовувати розпаралелювання [23].

Scrapy, у свою чергу, є складним універсальним інструментом із комплексною будовою. На рисунку 3.1 схематично зображено його програмну архітектуру [21].

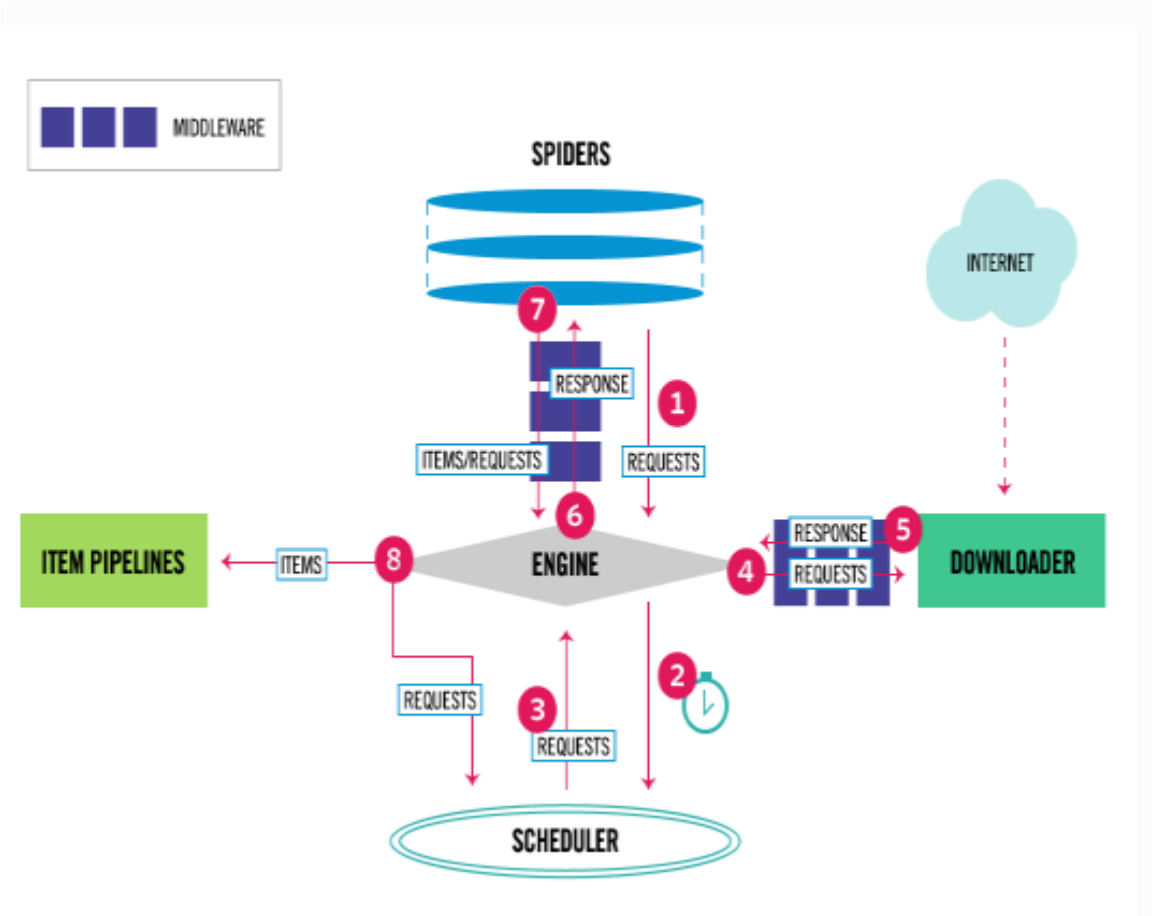

Рис 2.1. Архітектура фреймворку Scrapy

Опишемо основні абстрактні компоненти, які беруть участь у процесі веб-скрапінгу за допомогою Scrapy:

- Двигун Scrapy (англ. Scrapy Engine)

  Двигун відповідає за контроль потоків даних між усіма компонентами системи та спрацювання сценаріїв роботи при відповідних діях (командах) користувача.

- Планувальник (англ. Scheduler)

  Планувальник отримує запити від двигуна і ставить їх в чергу для майбутньої передачі тому ж таки двигуну, коли він запросить їх знову.

- Завантажувач (англ. Downloader)

  Цей компонент відповідає за завантаження веб-сторінок і передачу їх двигуну, який, у свою чергу, передає їх павукам.

- Павуки (англ. Spiders)

  Павуки є кастомними (від англ. custom – виготовлений на замовлення), тобто власноруч створеними, класами, написаними користувачами Scrapy, для парсингу відповідей веб-ресурсу і добування елементів сторінок з них.

- Конвеєр елементів (англ. Item Pipeline)

  Цей компонент відповідає за обробку елементів після того як вони були добуті павуками. Його типовими завданнями є очистка, перевірка та збереження інформації у базу даних.

Scrapy – це ефективний інструмент, який по замовчуванню використовує асинхронні запити та оптимізований алгоритм. У даній розрахунковій роботі ми використовуватимемо його для порівняння з власним розпаралеленим рішенням на основі Beautiful Soup.

Веб-сайтом, вибраним для веб-скрапінгу, є IMDb [17] – найбільший у світі веб-ресурс про кінематограф. Матеріал для роботи складає загалом 3000 елементів, які включають в себе назву, рік, жанр і рейтинг фільму. Для збереження

добутої інформації використовується файл формату csv [9], оформлений як дата-сет.

Робота здійснюється на комп'ютері Asus FX553VD з такою конфігурацією:

- Процесор: Intel(R) Core i5-7300HQ @ 2.50Hz, 4 ядра, 8 потоків
- Пам'ять: 8GB DDR4 2400 MHz
- ОС: Microsoft Windows 10 64 bit

Для розпаралелювання застосовується бібліотека multiprocessing [18] для Python з класом Pool. Загальна схема роботи алгоритму зображена на рисунку 3.2. Розпаралелення за допомогою даного методу полягає у розподілі завдань між доступними процесорами, використовуючи чергу FIFO (англ. first in, first out) [13], тобто принцип «перший зайшов, перший вийшов». Вхідні дані передаються різним процесорами, а вихідні збираються з усіх процесорів. Виконувані процеси зберігаються в оперативній пам'яті, а невиконувані – поза нею.

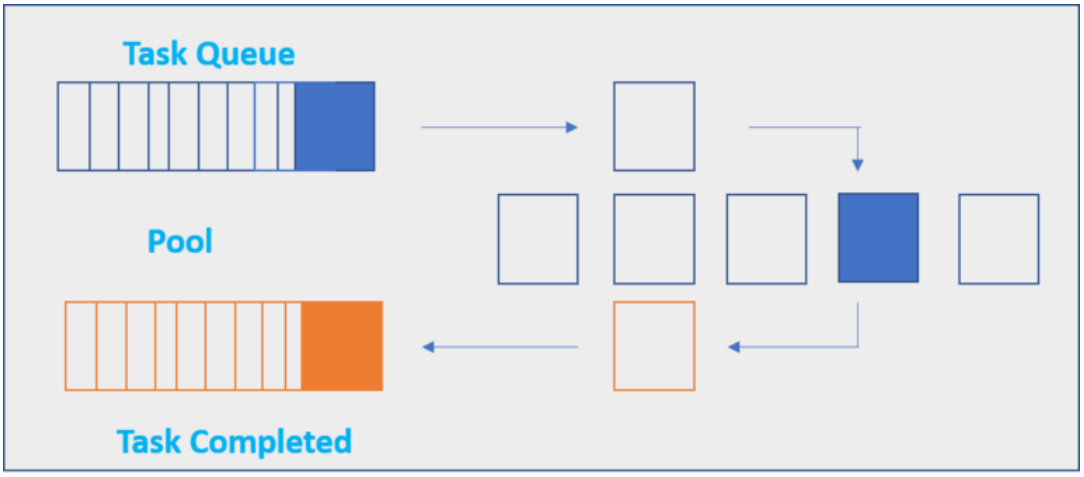

Рис. 2.2 Схема роботи класу Pool в бібліотеці multiprocessing

Клас Pool дозволяє виконувати багато робіт кожним процесом, що робить простішим завдання паралелізації програми, якщо існує дуже багато завдань, які необхідно запустити паралельно. Створюється Pool з таким числом процесів, яке відповідає кількості ядер CPU [10], а потім список завдань передається функції pool.map. Вона розподіляє ці завдання між працюючими процесорами, збирає значення, які вони повернули, у список і передає його батьківському процесу.

По результатам дослідження будуть обчислені статистичні показники. Прискорення (оцінює швидкість виконання):

$$S = \frac{T_1}{T_p}, \text{де} \quad (1)$$

T_1 – час виконання послідовного алгоритму,

T_p – час виконання паралельного алгоритму,

P – кількість процесів (потоків).

Ефективність (оцінює ефективність роботи кожного процесу або потоку):

$$E = \frac{S}{p} \quad (2)$$

3 РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ

Спочатку був реалізований послідовний алгоритм веб-скрапінгу сторінок із загальним вмістом 2700 елементів, використовуючи Python з бібліотекою Beautiful Soup (див. додаток А). Час виконання склав 2284 секунд.

Після цього, було проведено розпаралелення реалізованого алгоритму за допомогою бібліотеки multiprocessing (див. додаток Б). Час виконання в залежності від кількості потоків зображений у порівняльній таблиці 3.1.

Потоки	**1**	**2**	**4**	**8**
Час виконання, с	2284	1526	710	516

Таблиця 3.1 Результати виконання алгоритму (Beautiful Soup)

Крім того, алгоритм було застосовано при різних розмірах даних для опрацювання (кількість записів – k). Результати наведені у таблиці 3.2:

Розмір даних, k	**Час розрахунку паралельного алгоритму, с**			
	1 потік	**2 потоки**	**4 потоки**	**8 потоків**
150	108	47	26	18
200	124	74	38	26
250	166	101	52	33
300	250	122	71	57
600	567	386	170	106
1000	732	401	209	135

Таблиця 3.2 Результати виконання при різних розмірах (Beautiful Soup)

Порівняльним методом реалізації є Scrapy (див. додаток В). Він автоматично розпаралелює процес веб-скрапінгу. Час виконання алгоритму, на тих самих 2700 елементах склав 247 секунд. Результат при різних розмірах даних у таблиці 3.3:

Розмір даних, k	**Час виконання, с**
150	14
200	20
250	21
300	22
600	25
1000	36

Таблиця 3.3 Результати виконання при різних розмірах (Scrapy)

4 ОБГОВОРЕННЯ

Обчислимо прискорення та ефективність паралельного алгоритму (з використанням Beautiful Soup) порівняно з послідовним за формулою (1) та (2) відповідно, які наведені у розділі 2, та відобразимо результати у таблиці 4.1:

Потоки	Час виконання,с	Прискорення	Ефективність
2	1526	1,497	0,748
4	710	3,217	0,804
8	516	4,426	0,553

Таблиця 4.1 Прискорення та ефективність при різній кількості потоків (бібліотека Beautiful Soup)

Можна чітко побачити, що зі збільшенням кількості потоків зменшується час виконання і зростає прискорення. Візуалізація цього на Рис. 4.1 та 4.2 відповідно:

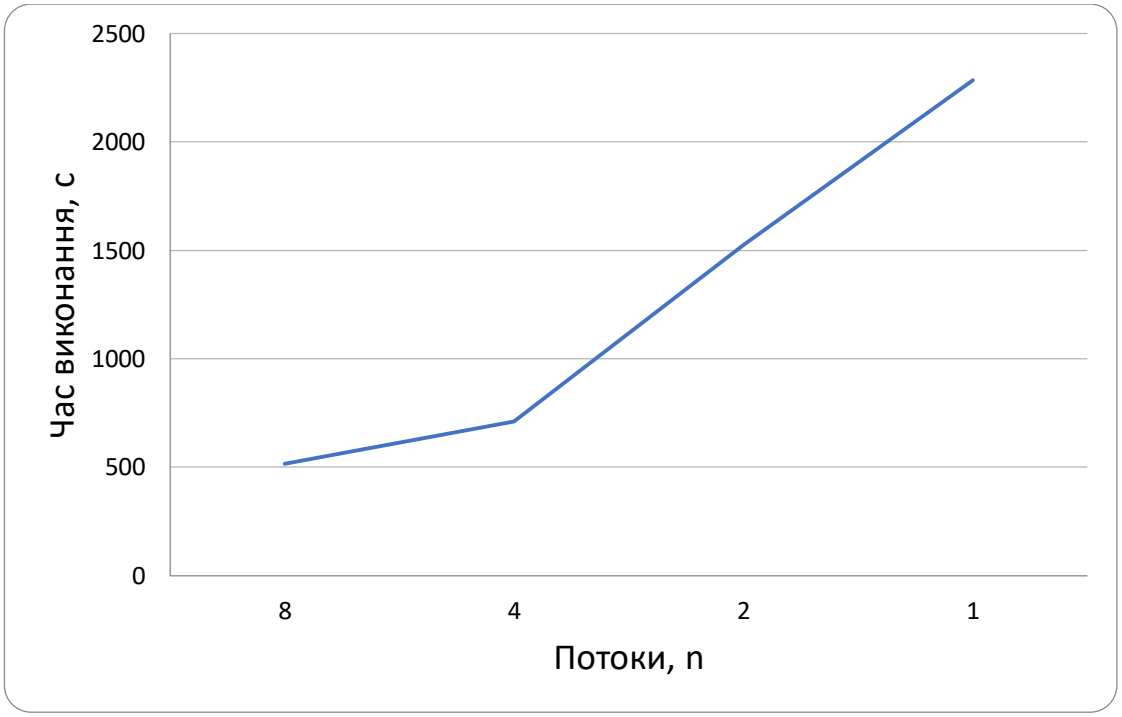

Рис. 4.1 Графік залежності часу виконання від кількості потоків (Beautiful Soup)

Найбільша ж ефективність спостерігається при 4-ох потоках (див. Рис 4.3).

Також порівняємо час виконання при різній кількості потоків в залежності від розміру даних на Рис 4.4. Видно, що, незалежно від кількості потоків, є тенденція зростання часу виконання при збільшенні розміру даних, однак, чим більше потоків, тим менш помітне це зростання.

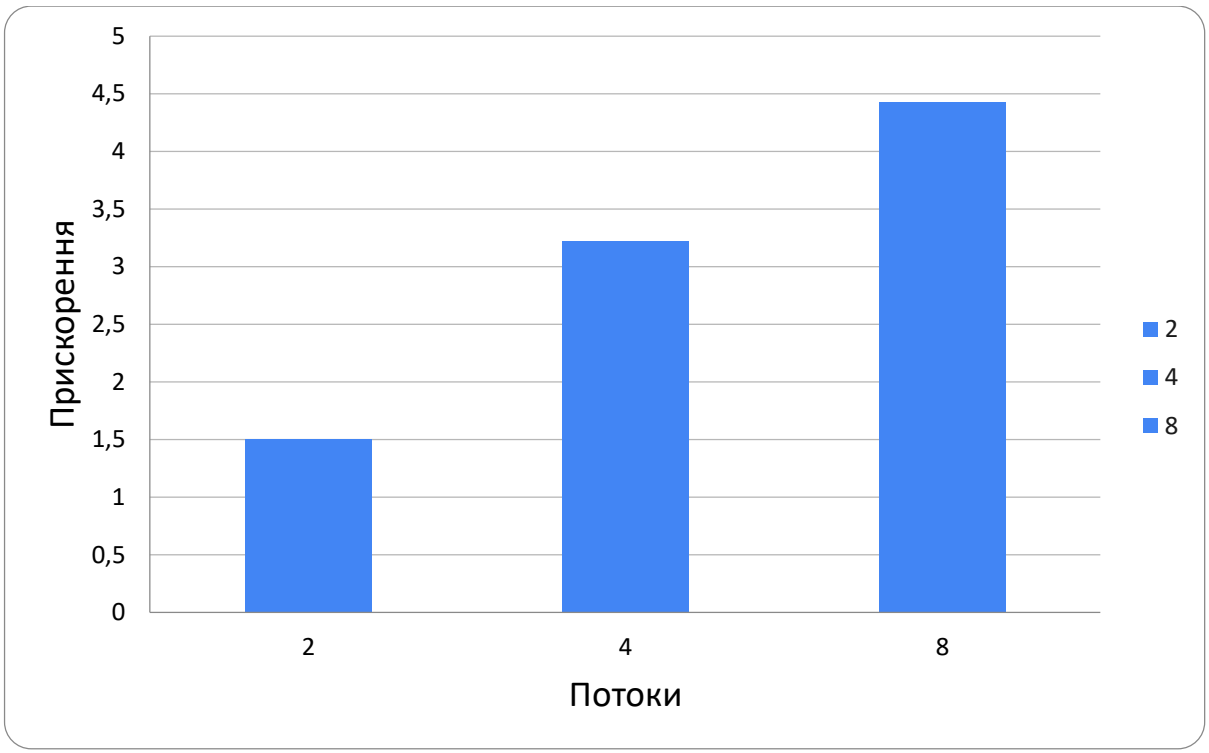

Рис. 4.2 Діаграма залежності прискорення від кількості потоків (Beautiful Soup)

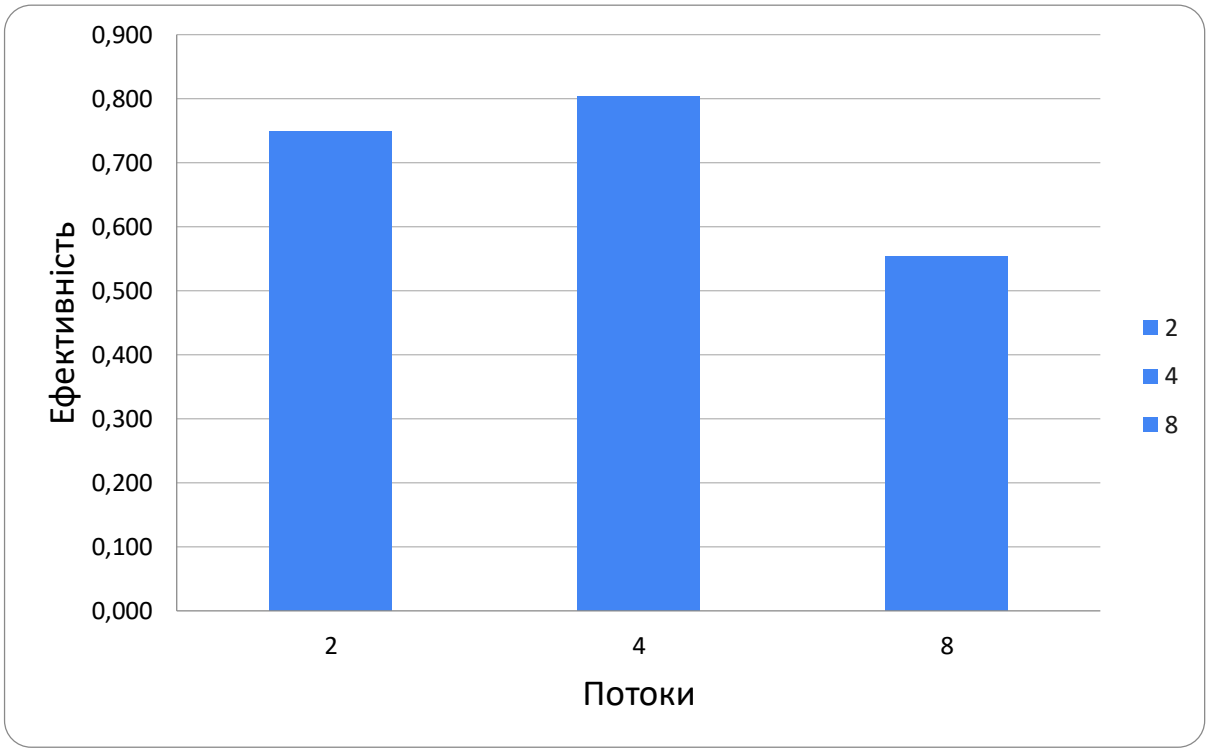

Рис. 4.1 Стовпчикова діаграма ефективності при різній кількості потоків (Beautiful Soup)

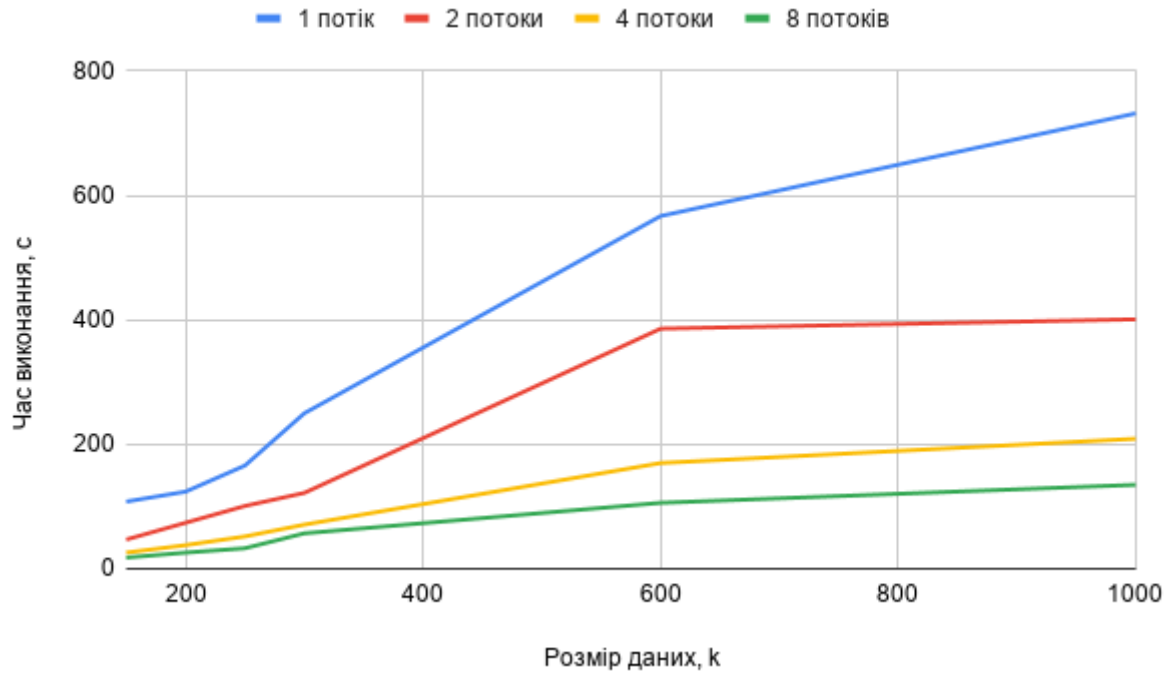

Рис. 4.4 Графік залежності часу виконання від розміру даних при різній кількості потоків (Beautiful Soup)

Далі проаналізуємо результати роботи алгоритму з використанням Scrapy та порівняємо його ефективність із нашою розпаралеленою реалізацією. На Рис. 4.5 візуалізація таблиці 3.3.

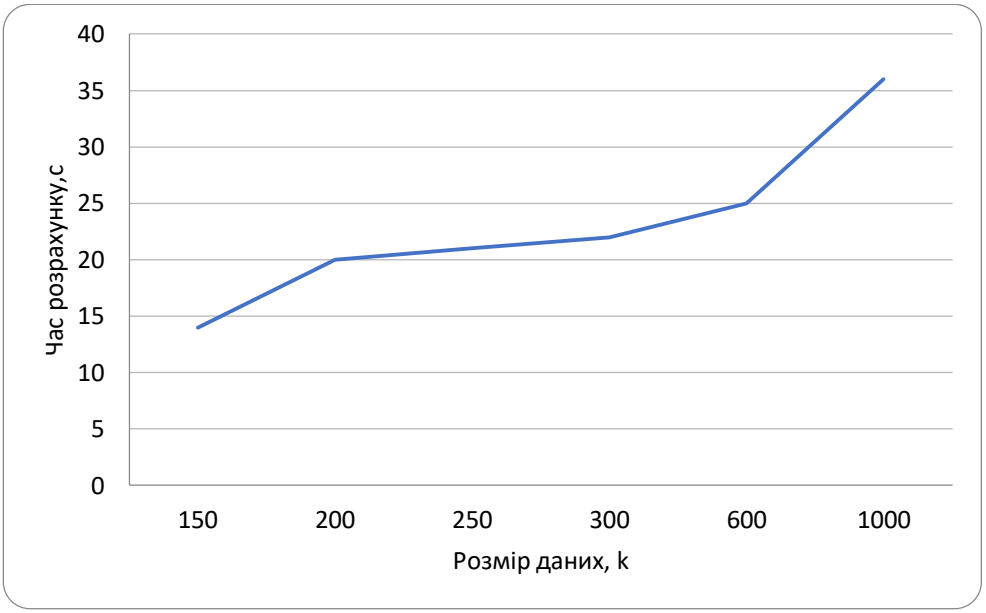

Рис. 4.5 Графік залежності часу виконання від розміру даних (Scrapy)

Порівняємо швидкість виконання алгоритму з використанням Scrapy із розпаралеленим алгоритмом на основі Beautiful Soup при різних вхідних даних у таблиці 4.2:

Розмір даних, k	Час розрахунку паралельного алгоритму, с				
	1 потік	2 потоки	4 потоки	8 потоків	Scrapy
150	108	47	26	18	14
200	124	74	38	26	20
250	166	101	52	33	21
300	250	122	71	57	22
600	567	386	170	106	25
1000	732	401	209	135	36

Таблиця 4.2 Порівняння час виконання при використанні Beautiful Soup з різною кількістю потоків та Scrapy

Добре видно, що наш алгоритм розпаралелення поступається у ефективності Scrapy. Візуалізуємо для наочності таблицю 4.2 на Рис. 4.6, а також обчислимо прискорення на 8 потоках з використанням Beautiful Soup та при застосуванні Scrapy (див. таблиця 4.3) і зобразимо порівняльний графік на Рис. 4.7.

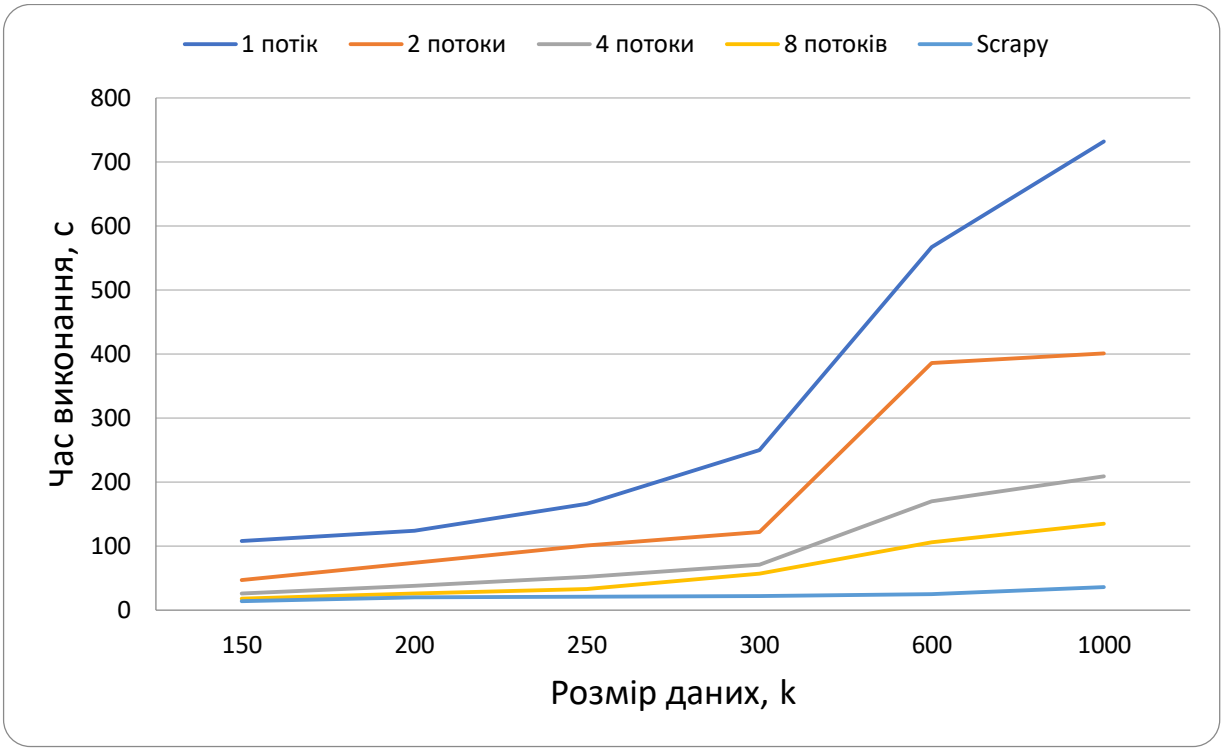

Рис. 4.6 Графік залежності часу виконання від розміру даних при використанні Beautiful Soup з різною кількістю потоків та Scrapy

Розмір даних, k	Прискорення	
	8 потоків	Scrapy
150	6,000	7,714
200	4,769	6,200
250	5,030	7,905
300	4,386	11,364
600	5,349	22,680
1000	5,422	20,333
2700	4,426	8,397

Таблиця 5.3 Прискорення при різному розмірі даних при виконанні паралельного алгоритму на 8 потоках (Beautiful Soup) та Scrapy

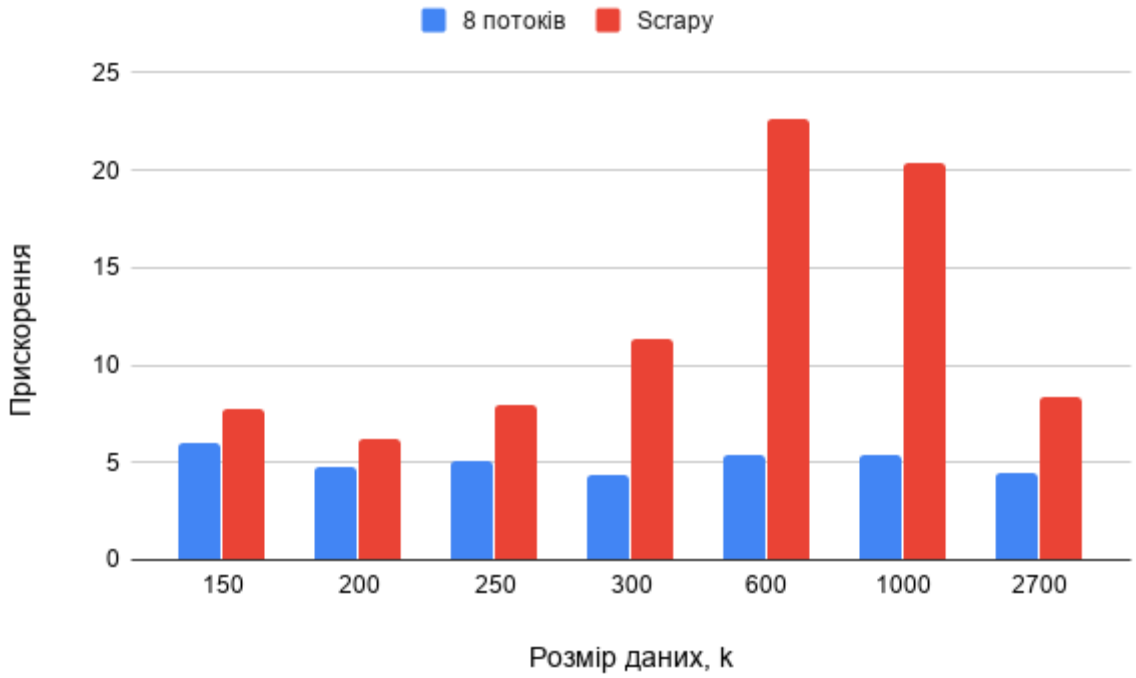

Рис. 5.3 Графік залежності прискорення від розміру даних при виконанні паралельного алгоритму на 8 потоках (Beautiful Soup) в порівнянні з Scrapy

Перевага Scrapy очевидна. Причина такої ефективності криється в нетривіальних алгоритмах асинхронних запитів та складній, продуманій, оптимізованій програмній архітектурі. Звичайно, нашому алгоритму не зрівнятися з цим. Однак все не так однозначно, як здається на перший погляд. При невеликих розмірах даних ефективність обох алгоритмів практично співпадає, що дозволяє використовувати Beautiful Soup для невеликих задач веб-скрапінгу. Адже

навчитися користуватися простою бібліотекою легше і швидше, ніж таким комплексним рішенням як фреймворк Scrapy. Саме з цих причин Beautiful Soup можна рекомендувати для початківців та недосвідчених користувачів. Наше дослідження доводить можливість його нескладного та відносно ефективного розпаралелення й застосування, враховуючи те, що послідовний алгоритм майже в 4,5 рази повільніше за паралельний у випадку використання 8 потоків.

ВИСНОВКИ

Отже, всі 4 гіпотези, виражені у вступі, були перевірені та підтверджені. HTML-парсинг за допомогою мови Python з використанням бібліотеки Beautiful Soup (для невеликих задач) або фреймворку Scrapy (для масштабних проектів) – це гнучкі та надійні способи добування даних з інтернет-ресурсів.

Реалізований алгоритм вдалося ефективно розпаралелити та на основі аналізу результатів роботи зрозуміти, що кожен з використовуваних інструментів по-своєму корисний та необхідний, хоч і виявилося, що власноруч проведена паралелізація не дає такі хороші результати, як готове асинхронне рішення.

Можна порекомендувати використовувати дане дослідження при виборі методів для практичного застосування у веб-проектах та у навчальних цілях, адже у розділі обговорення у достатній мірі повноти було наведено показники ефективності з детальною візуалізацією.

У висновку до даної роботи вважаємо, що варто звернути увагу і на юридичні аспекти веб-скрапінгу. У різних країнах до такого способу отримання даних з веб-сайтів відносять по-різному, але загальне бачення, в принципі, збігається: не варто зловживати методом веб-скрапінгу у комерційних цілях у випадку, якщо інформація веб-ресурсу обмежена угодою користування чи іншими правовими документами (зокрема, ліцензіями, авторськими правами тощо), а також здійснювати надмірне навантаження на веб-сервери великою кількістю запитів, оскільки це може потягнути за собою перебої у його роботі та блокування користувача, який причетний до цих подій.

Подальший розвиток даної теми досліджень вважаємо перспективним, адже, як сказав англо-німецький фінансист Натан Ротшильд, «хто володіє інформацією, той володіє світом», а з цим важко посперечатися у сучасному світі глобальних пошукових систем та соціальних мереж. У майбутніх дослідженнях можна сконцентруватися на порівнянні програмних рішень на зразок Scrapy із методами використання комп’ютерного зору та машинного навчання для веб-скрапінгу. Тим більше, враховуючи те, що такі затратні по обчислювальним ресурсам сфери так само потребують застосування розпаралелення та розподілених обчислень.

БІБЛІОГРАФІЯ

1. Alessandro Fiori. Trends and Applications of Text Summarization Techniques : монографія. Герші, США : IGI Globlal, 1982. 334 с.
2. Salim Khali, Mohamed Fakir. RCrawler: An R package for parallel web crawling and scraping. Бені-Мелаль, Мароко : Department of Informatics, Faculty of Sciences and Technics Beni Mellal, 2017. 9 с.
3. Веб-скрапінг // Вікіпедія : веб-сайт. URL: https://uk.wikipedia.org/wiki/Web_scraping (дата звернення: 05.04.20).
4. Всесвітнє павутиння // Вікіпедія : веб-сайт. URL : https://tinyurl.com/ya7elbej (дата звернення 05.04.20).
5. Architecture overview // Scrapy : веб-сайт. URL: https://docs.scrapy.org/en/latest/topics/architecture.html (дата звернення: 09.05.20).
6. Beautiful Soup (HTML parser) // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Beautiful_Soup_(HTML_parser)/ (дата звернення: 09.05.20).
7. Best Free and Paid Web Scraping Tools and Software // ScrapeHero : веб-сайт. URL: https://www.scrapehero.com/top-free-and-paid-web-scraping-tools-and-software/ (дата звернення: 08.05.20).
8. CAPTCHA // Вікіпедія : веб-сайт. URL: https://uk.wikipedia.org/wiki/CAPTCHA (дата звернення 27.04.20).
9. Comma-separated values // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Comma-separated_values (дата звернення: 05.05.20).
10. CPU // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Central_processing_unit (дата звернення: 07.05.20).
11. Document Object Model // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Document_Object_Model/ (дата звернення: 07.05.20).
12. Experimental web-scraping using the Google Cloud Platform Vision offering // Medium : веб-сайт. URL: https://medium.com/vendasta/experimental-web-scraping-using-the-google-cloud-platform-vision-offering-d0e87657e112 (дата звернення: 09.05.20).

13. FIFO // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics) (дата звернення: 05.05.20).
14. General techniques used for web scraping // IGN : веб-сайт. URL: https://www.ign.com/wikis/general-techniques-used-for-web-scraping/ (дата звернення 06.04.20).
15. How to build a Web scraper with DOM parsing in 10 minutes // Markus Östberg : веб-сайт. URL: https://www.ostberg.dev/projects/2015/08/25/how-to-build-a-web-scraper.html (дата звернення: 07.05.20).
16. HTML Scraping // The Hitchhiker`s Guide to Python : веб-сайт. URL: https://docs.python-guide.org/scenarios/scrape/ (дата звернення: 07.05.20).
17. IMDb : веб-сайт. URL: https://www.imdb.com/ (дата звернення: 05.05.20).
18. multiprocessing — Process-based parallelism // Python Documentation : веб-сайт. URL: https://docs.python.org/3.4/library/multiprocessing.html?highlight=process (дата звернення: 05.05.20).
19. Network socket // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Network_socket (дата звернення: 09.05.20).
20. Parsing HTML: A Guide to Select the Right Library // Federico Tomassetti Software Architect : веб-сайт. URL: https://tomassetti.me/parsing-html/ (дата звернення: 07.05.20).
21. Python : веб-сайт. URL: https://www.python.org/about/ (дата звернення: 09.05.20).
22. Scrapy / Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Scrapy (дата звернення: 09.05.20).
23. Scrapy Vs Selenium Vs Beautiful Soup for Web Scraping // Medium : веб-сайт. URL: https://medium.com/analytics-vidhya/scrapy-vs-selenium-vs-beautiful-soup-for-web-scraping-24008b6c87b8 (дата звернення: 09.05.20).
24. The Most Effective Web Scraping Methods // JetRuby : веб-сайт. URL: https://expertise.jetruby.com/the-most-effective-web-scraping-methods-62e7e34ada69 (дата звернення 28.04.20).

25. Vertical aggregation & Pattern matching crawlers // webhose.io : веб-сайт. URL: https://webhose.io/blog/api/vertical-aggregation-pattern-matching-crawlers/ (дата звернення: 08.05.20).
26. Web crawler // Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Web_crawler (дата звернення 05.04.20).
27. Web Scraping, Regular Expressions, and Data Visualization: Doing it all in Python // towards data science : веб-сайт. URL: https://towardsdatascience.com/web-scraping-regular-expressions-and-data-visualization-doing-it-all-in-python-37a1aade7924 (дата звернення: 07.05.20).

ДОДАТКИ

ДОДАТОК А

Послідовний алгоритм з використанням бібліотеки Beautiful Soup

```
import requests
from bs4 import BeautifulSoup as bs
import csv
from datetime import datetime
import sys

def get_html(url):
    r = requests.get(url)  # response

    return r.text  # returns html code


def get_all_links(html):
    soup = bs(html, 'lxml')
    item_headers = soup.find('div', class_='article').find_all('h3',
class_='lister-item-header')
    links = []

    for h3 in item_headers:
        href = h3.find('a').get('href')
        link = 'https://www.imdb.com' + href
        links.append(link)

    return links


def get_nextpage_link(html):
    soup = bs(html, 'lxml')
    next_page = soup.find('div', class_='article').find('div', class_='desc')
    try:
        href = next_page.find('a', class_='lister-page-next next-
page').get('href')
        next_link = 'https://www.imdb.com' + href
    except:
        end = datetime.now()
        total = end - start
        sys.exit(total)

    return next_link

def get_page_data(html):
    soup = bs(html, 'lxml')

    try:
        title = soup.find('div', class_='title_wrapper').find('h1').next.strip()
    except:
        title = ''

    try:
        year = soup.find('span', id='titleYear').find('a').text.strip()
    except:
        year = ''
```

```
    try:
        genre = soup.find('div', class_='subtext').find('a').text.strip()
    except:
        genre = ''

    try:
        rating = soup.find('span', itemprop='ratingValue').text.strip()
    except:
        rating = ''

    data = {'title': title,
            'year': year,
            'genre': genre,
            'rating': rating}

    return data


def write_csv(data):
    with open('imdb.csv', 'a') as f:
        writer = csv.writer(f)
        writer.writerow((data['title'],
                         data['year'],
                         data['genre'],
                         data['rating']))
        print(data['title'], data['year'], data['genre'], data['rating'],
'parsed')


def parse(url):
    html = get_html(url)
    next_page_link = get_nextpage_link(html)
    print(next_page_link)
    all_links = get_all_links(html)
    for url in all_links:
        html = get_html(url)
        data = get_page_data(html)
        write_csv(data)
    parse(next_page_link)



def main():
    global start
    start = datetime.now()
    print(start)

    url = 'https://www.imdb.com/search/title/?country_of_origin=ua&ref_=tt_dt_dt'

    parse(url)

    end = datetime.now()
    total = end - start
    print(str(total))

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

ДОДАТОК Б

Розпаралелення послідовного алгоритму з використанням бібліотеки Beautiful Soup

```
import requests
from bs4 import BeautifulSoup as bs
import csv
from datetime import datetime
from multiprocessing import Pool


def get_html(url):
    r = requests.get(url)  # response

    return r.text  # returns html code


def get_all_links(html):
    soup = bs(html, 'lxml')
    item_headers = soup.find('div', class_='article').find_all('h3',
class_='lister-item-header')
    links = []

    for h3 in item_headers:
        href = h3.find('a').get('href')
        link = 'https://www.imdb.com' + href
        links.append(link)

    return links


def get_nextpage_link(html):
    soup = bs(html, 'lxml')
    next_page = soup.find('div', class_='article').find('div', class_='desc')
    try:
        href = next_page.find('a', class_='lister-page-next next-
page').get('href')
        next_link = 'https://www.imdb.com' + href
    except:
        next_link = '---exception---'

    return next_link


def check_exception(next_link):
    if next_link == '---exception---':
        parse(url.pop(0))


def get_page_data(html):
    soup = bs(html, 'lxml')

    try:
        title = soup.find('div', class_='title_wrapper').find('h1').next.strip()
```

```
    except:
        title = ''

    try:
        year = soup.find('span', id='titleYear').find('a').text.strip()
    except:
        year = ''

    try:
        genre = soup.find('div', class_='subtext').find('a').text.strip()
    except:
        genre = ''

    try:
        rating = soup.find('span', itemprop='ratingValue').text.strip()
    except:
        rating = ''

    data = {'title': title,
            'year': year,
            'genre': genre,
            'rating': rating}

    return data


def write_csv(data):
    with open('imdb_multi.csv', 'a') as f:
        writer = csv.writer(f)
        writer.writerow((data['title'],
                         data['year'],
                         data['genre'],
                         data['rating']))
    print(data['title'], data['year'], data['genre'], data['rating'], 'parsed')


def make_all(url):
    html = get_html(url)
    data = get_page_data(html)
    write_csv(data)


def parse(url):
    html = get_html(url)
    next_page_link = get_nextpage_link(html)
    print(next_page_link)
    all_links = get_all_links(html)

    with Pool(2) as p:
        p.map(make_all, all_links)

    try:
        parse(next_page_link)
    except:
        check_exception(next_page_link)


def main():
```

```
    global start, url
    start = datetime.now()
    print(start)

    url = ['https://www.imdb.com/search/title/?countries=ao',
           'https://www.imdb.com/search/title/?countries=ly',
           'https://www.imdb.com/search/title/?countries=ad',
           'https://www.imdb.com/search/title/?countries=bi',
           'https://www.imdb.com/search/title/?countries=ao',
           'https://www.imdb.com/search/title/?countries=ao',
           'https://www.imdb.com/search/title/?countries=ao',
           'https://www.imdb.com/search/title/?countries=ao',
           'https://www.imdb.com/search/title/?countries=ao',
           'https://www.imdb.com/search/title/?countries=ly',
           ]

    parse(url.pop(0))

    end = datetime.now()
    total = end - start
    print(str(total))


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

ДОДАТОК В

Алгоритм виконання веб-скрапінгу за допомогою Scrapy

```
from ddd.items import DddItem

import scrapy
from urllib.parse import urljoin
from datetime import datetime


class IMDbSpyder(scrapy.Spider):

    def __init__(self, name=None, **kwargs):
        self.starting_time = datetime.now()

    name = "pauk"
    start_urls = ['https://www.imdb.com/search/title/?countries=ao',
                  'https://www.imdb.com/search/title/?countries=ly',
                  'https://www.imdb.com/search/title/?countries=ad',
                  'https://www.imdb.com/search/title/?countries=bi',
                  'https://www.imdb.com/search/title/?countries=ao',
                  'https://www.imdb.com/search/title/?countries=ao',
                  'https://www.imdb.com/search/title/?countries=ao',
                  # 'https://www.imdb.com/search/title/?countries=ao',
                  # 'https://www.imdb.com/search/title/?countries=ao',
                  # 'https://www.imdb.com/search/title/?countries=ly',
                  ]
    # custom_settings = {
    #     'CONCURRENT_REQUESTS': 8
    # }
    visited_urls = []

    def parse(self, response):
        if response.url not in self.visited_urls:
            self.visited_urls.append(response.url)
            item_link = response.xpath(
                '//div[@class="lister-item-content"]/h3/a/@href').extract()
            for post_link in item_link:
                url = urljoin(response.url, post_link)
                print(url)
                yield response.follow(url, callback=self.parse_post)

            next_pages = response.xpath(
                '//div[@class="nav"]/div[@class="desc"]/a/@href').extract()
            next_page = next_pages[-1]

            next_page_url = urljoin(response.url + '/', next_page)
            yield response.follow(next_page_url, callback=self.parse)

    def parse_post(self, response):
        item = DddItem()
        title =
response.xpath('.//div[@class="title_wrapper"]/h1/text()').extract_first()
        item['title'] = title
        year =
response.xpath('.//div[@class="title_wrapper"]/h1/span/a/text()').extract_first()
        item['year'] = year
```

```
        genre =
response.xpath('.//div[@class="subtext"]/a/text()').extract_first()
        item['genre'] = genre
        rating =
response.xpath('.//span[@itemprop="ratingValue"]/text()').extract_first()
        item['rating'] = rating
        yield item

    def closed(self, response):
        self.ending_time = datetime.now()
        duration = self.ending_time - self.starting_time
        print("\nTime of execution: " + str(duration) + "\n")
```