Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
МАГІСТРА

на тему *Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики*

Галузь знань *12 – Інформаційні технології*
Шифр і назва галузі знань

Спеціальність *122 – Комп'ютерні науки*
Шифр і назва спеціальності

Виконав: студент *2* курсу, група *КНм-20-1* — Підпис — *С.В. Данчук* (Ініціали, прізвище)

Керівник: *к.т.н., доцент кафедри КНІТ* — Підпис — *Р.О. Багрій* (Ініціали, прізвище)

Нормоконтроль: *к.т.н., доцент кафедри КНІТ* — Підпис — *Р.О. Багрій* (Ініціали, прізвище)

До захисту допускаю:
Зав. кафедри КНІТ, д.т.н., професор — Підпис — *О.В. Бармак* (Ініціали, прізвище)
7 *грудня* 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет *інформаційних технологій*

Кафедра *комп'ютерних наук*

Освітній ступінь *магістр*

Галузь знань *12 – Інформаційні технології*

Спеціальність *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Завідувач кафедри *комп'ютерних наук*

_____

(підпис)

д.т.н., професор О.В. Бармак

«1» вересня *2020* року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

1. Тема кваліфікаційної роботи магістра: *«Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики»*

2. Завдання видано студенту *Данчук Сергій Вікторович*

(прізвище, ім'я, по батькові)

3. Керівник роботи *к.т.н., доцент Багрій Руслан Олександрович*

(прізвище, ім'я, по батькові)

4. Затверджені наказом університету від «25» серпня *202* р. № 102

5. Зміст пояснювальної записки (перелік задач) та вихідні дані:

Мета роботи – розробка автоматизованої системи для збору даних з веб ресурсів для подальшої бізнес-аналітики. Для досягнення мети необхідно визначити типи даних, що необхідні для бізнес-аналітики, дослідити існуючі методи отримання даних з веб-ресурсів, розробити алгоритм синтаксичного розбору веб-сторінок для подальшого імпорту та експорту даних, спроектувати автоматизовану систему збору даних, відповідну програмну реалізацію та провести її тестування.

Реферат

Кваліфікаційна робота магістра присвячена розробці інформаційної системи автоматизованого отримання даних з веб-ресурсів для подальшої бізнес-аналітики.

Актуальність теми.

Останнім часом Інтернет став все більше поповнюватися новими інтернет-магазинами. Тому що на сьогоднішній день наявність власного інтернет - магазину є зручним і перспективним напрямком в бізнесі. Це рішення потребує аналізу для розвитку і конкуренції з іншими веб-ресурсами схожої тематики.

Для цього потрібно постійно збирати дані з веб-сайтів. Наприклад назву товару для структурованого відображення, ціну товару, відгуки, оцінки користувачів, описи для подальшого аналізу товарів. Знайти неіснуючі сторінки, дублі, неповний опис, відсутність певних характеристик або невідповідність даних по складських залишків того, що відображається на сайті. Для цього використовують технологію web-scraping.

Web-scraping сайтів є ефективним рішенням для автоматизації збору і зміни інформації.

У порівнянні з людиною, комп'ютерна програма:

- швидко обійде тисячі веб-сторінок;
- акуратно відокремить технічну інформацію від «людської»;
- безпомилково відбере потрібне і відкине зайве;
- ефективно упакує кінцеві дані в необхідному вигляді.

В результаті роботи технології web-scraping, зазвичай дані зберігаються у базі даних чи електронних таблицях, що потребує подальшої обробки (аналітики).

Застосування web-scraping може бути актуальним в багатьох випадках:

1. Якщо необхідно автоматично оновлювати сторінки вашого сайту, тобто автоматично додавати статті, новини або інший контент.

2. Для підтримки актуальності використовуваної на сайті інформації.

3. Якщо необхідно об'єднати інформацію, так як вона часто перебуває на різних сайтах.

4. Великі обсяги. В епоху бурхливого зростання мережі і жорстокої конкуренції ясно, що успішний веб-проект немислимий без розміщення великої кількості інформації на сайті. Сучасні темпи життя призводять до того, що контенту має бути не просто багато, а дуже багато, в кількостях, що набагато перевищують межі, можливі при ручному заповненні.

5. Часте оновлення. Обслуговування величезного потоку динамічно мінливої інформації не в силах забезпечити одна людина або навіть злагоджена команда операторів. Часом інформація змінюється щохвилини і в ручному режимі оновлювати її навряд чи доцільно.

В процесі web-scraping застосовуються скриптові мови програмування: PHP, Perl, Ruby, Python, JavaScript і багато інших. Він здійснюється шляхом написання так званого парсеру. Парсер – це програма, за допомогою якої відбувається автоматична обробка сторінок сайту для отримання необхідних даних. Парсер сайтів створюється для збору великої бази контенту на сайт для аналізу і для здійснення пошуку необхідної для користувачів інформації.

Мета і задачі роботи. Метою магістерської роботи є створення технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики. Це дозволить ефективно аналізувати інтернет простір, швидко реагувати на зміни трендів та втілювати задумані бізнес ідеї в життя.

Для досягнення поставленої мети визначені наступні задачі:

– визначити типи даних, що необхідно отримувати з веб-сторінок для бізнес-аналітики;

– проаналізувати існуючі методи отримання даних з веб-ресурсів – web-scraping;

– розробити алгоритми синтаксичного розбору веб-сторінок та експорту даних;

– спроектувати інформаційну системи автоматизованого збору даних з веб-ресурсів на основі проведених досліджень;

– створити програмну реалізацію системи та провести її тестування;

Об’єкт дослідження – процес отримання даних з веб-ресурсів.

Предмет дослідження – моделі, методи, підходи та засоби для отримання даних з веб-ресурсів.

Наукова новизна одержаних результатів. В результаті проведеної роботи були отримані такі результати:

– вдосконалено алгоритм синтаксичного розбору веб-сторінки, що дає можливість отримувати та зберігати необхідні для бізнес-аналітики дані.

– вдосконалено метод для доступу до веб-ресурсів, що дозволив обійти встановлені перешкоди для автоматизованої обробки даних;

– розроблено інформаційну системи автоматизованого отримання, обробки та збереження даних з веб ресурсів для подальшої бізнес-аналітики. Це дозволить скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі.

Практичне значення одержаних результатів. На основі розробленої інформаційної системи автоматизованого отримання даних з веб-ресурсів було створено програму реалізацію, що дає можливість обійти встановлені перешкоди для автоматизованої обробки даних та здійснити їх отримання та збереження для подальшої бізнес-аналітики.

Апробація результатів дипломної роботи магістра та публікації.:

Доповідь на тему «Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики» на XIII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук АПКН-2021» (Хмельницький, 9-10 листопада 2020 року, Хмельницький національний університет).

Структура та обсяг роботи. Кваліфікаційна робота магістра складається із реферату, завдання, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань із 8 найменувань та 2 додатка. Загальний обсяг кваліфікаційної роботи магістра становить 98 сторінок, з них 70 сторінок

основного тексту та 28 сторінки додатків. У роботі наведений 22 рисуноки та 1 таблиця.

Ключові слова: Парсер, веб-скрейпінг, видобуток даних, синтаксичний аналіз веб-сторінки, інформаційна система.

Зміст

Перелік скорочень

Скорочення, термін, позначення	Пояснення
DOM	Document Object Model — «об'єктна модель документу»
Парсер	Програма аналізатор

Вступ

Кваліфікаційна робота магістра присвячена розробці інформаційної системи автоматизованого отримання даних з веб-ресурсів для подальшої бізнес-аналітики.

Актуальність теми.

Останнім часом Інтернет став все більше поповнюватися новими інтернет-магазинами. Тому що на сьогоднішній день наявність власного інтернет - магазину є зручним і перспективним напрямком в бізнесі. Це рішення потребує аналізу для розвитку і конкуренції з іншими веб-ресурсами схожої тематики.

Для цього потрібно постійно збирати дані з веб-сайтів. Наприклад назву товару для структурованого відображення, ціну товару, відгуки, оцінки користувачів, описи для подальшого аналізу товарів. Знайти неіснуючі сторінки, дублі, неповний опис, відсутність певних характеристик або невідповідність даних по складських залишків того, що відображається на сайті. Для цього використовують технологію web-scraping.

Web-scraping сайтів є ефективним рішенням для автоматизації збору і зміни інформації.

У порівнянні з людиною, комп'ютерна програма:

– швидко обійде тисячі веб-сторінок;

– акуратно відокремить технічну інформацію від «людської»;

– безпомилково відбере потрібне і відкине зайве;

– ефективно упакує кінцеві дані в необхідному вигляді.

В результаті роботи технології web-scraping, зазвичай дані зберігаються у базі даних чи електронних таблицях, що потребує подальшої обробки (аналітики).

Застосування web-scraping може бути актуальним в багатьох випадках:

6. Якщо необхідно автоматично оновлювати сторінки вашого сайту, тобто автоматично додавати статті, новини або інший контент.

7. Для підтримки актуальності використовуваної на сайті інформації.

8. Якщо необхідно об'єднати інформацію, так як вона часто перебуває на різних сайтах.

9. Великі обсяги. В епоху бурхливого зростання мережі і жорстокої конкуренції ясно, що успішний веб-проект немислимий без розміщення великої кількості інформації на сайті. Сучасні темпи життя призводять до того, що контенту має бути не просто багато, а дуже багато, в кількостях, що набагато перевищують межі, можливі при ручному заповненні.

10. Часте оновлення. Обслуговування величезного потоку динамічно мінливої інформації не в силах забезпечити одна людина або навіть злагоджена команда операторів. Часом інформація змінюється щохвилини і в ручному режимі оновлювати її навряд чи доцільно.

В процесі web-scraping застосовуються скриптові мови програмування: PHP, Perl, Ruby, Python, JavaScript і багато інших. Він здійснюється шляхом написання так званого парсеру. Парсер – це програма, за допомогою якої відбувається автоматична обробка сторінок сайту для отримання необхідних даних. Парсер сайтів створюється для збору великої бази контенту на сайт для аналізу і для здійснення пошуку необхідної для користувачів інформації.

Мета і задачі роботи. Метою магістерської роботи є створення технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики. Це дозволить ефективно аналізувати інтернет простір, швидко реагувати на зміни трендів та втілювати задумані бізнес ідеї в життя.

Для досягнення поставленої мети визначені наступні задачі:

– визначити типи даних, що необхідно отримувати з веб-сторінок для бізнес-аналітики;

– проаналізувати існуючі методи отримання даних з веб-ресурсів – web-scraping;

– розробити алгоритми синтаксичного розбору веб-сторінок та експорту даних;

– спроектувати інформаційну системи автоматизованого збору даних з веб-ресурсів на основі проведених досліджень;

– створити програмну реалізацію системи та провести її тестування;

Об’єкт дослідження – процес отримання даних з веб-ресурсів.

Предмет дослідження – моделі, методи, підходи та засоби для отримання даних з веб-ресурсів.

Наукова новизна одержаних результатів. В результаті проведеної роботи були отримані такі результати:

– вдосконалено алгоритм синтаксичного розбору веб-сторінки, що дає можливість отримувати та зберігати необхідні для бізнес-аналітики дані.

– вдосконалено метод для доступу до веб-ресурсів, що дозволив обійти встановлені перешкоди для автоматизованої обробки даних;

– розроблено інформаційну системи автоматизованого отримання, обробки та збереження даних з веб ресурсів для подальшої бізнес-аналітики. Це дозволить скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі.

Практичне значення одержаних результатів. На основі розробленої інформаційної системи автоматизованого отримання даних з веб-ресурсів було створено програму реалізацію, що дає можливість обійти встановлені перешкоди для автоматизованої обробки даних та здійснити їх отримання та збереження для подальшої бізнес-аналітики.

Апробація результатів дипломної роботи магістра та публікації.:

Доповідь на тему «Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики» на XIII Всеукраїнській науково-практичній конференції «Актуальні проблеми комп’ютерних наук АПКН-2021» (Хмельницький, 9-10 листопада 2020 року, Хмельницький національний університет).

Структура та обсяг роботи. Кваліфікаційна робота магістра складається із реферату, завдання, змісту, переліку скорочень, вступу, 4 розділів, висновків, переліку посилань із 8 найменувань та 2 додатка. Загальний обсяг кваліфікаційної роботи магістра становить 98 сторінок, з них 70 сторінок

основного тексту та 28 сторінки додатків. У роботі наведений 22 рисунки та 1 таблиця.

Ключові слова: Парсер, веб-скрейпінг, видобуток даних, синтаксичний аналіз веб-сторінки, інформаційна система.

Розділ 1

Аналіз сучасного автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики

1.1 Аналіз предметної області

Всього лише 15 років тому ролі «бізнес-аналітик» в IT-проектах не існувало. Вимоги визначали і погоджували розробники, або проектні менеджери, а іноді й сам замовник. З часом кількість проектів збільшувалась, а їх складність настільки зросла, що стало зрозумілим: бізнес-аналіз – окремий великий пласт роботи.

Бізнес-аналіз – це набір методів, які допомагають зрозуміти структуру, особливості компанії клієнта, визначити її потреби і запропонувати варіанти рішення задачі. В процесі бізнес-аналізу вибирають оптимальне рішення, готують до нього вимоги, оцінюють, яка функціональність найбільш важлива для замовника, укладають та погоджують документацію для розробників. В IT бізнес-аналітики працюють з інформаційними системами – сайтами та додатками.

Задачі бізнес-аналізу для сайтів:

- допомагає розвивати сайт та оптимізувати його просування;
- оцінює ефективність рекламних кампаній;
- покращує користування веб-ресурсом на основі тенденцій у поведінці відвідувачів;
- виявляє проблемні місця у структурі порталу та його контенті;
- оцінює трафік;
- допомагає досягти поставленої мети.

Для цього потрібно постійно збирати з сайту дані. Наприклад назву товару для структурованого відображення. Ціну товару, відгуки, оцінки користувачів, описи для подальшого аналізу товарів. Знайти неіснуючі сторінки, дублі, неповне опис, відсутність певних характеристик або невідповідність

даних по складських залишків того, що відображається на сайті. Для цього використовують технологію web-scraping.

Web-scraping – це процес збирання і систематизування інформації, розміщеної на певних сайтах, за допомогою спеціальних програм, що автоматизують процес.

Це програмні продукти, основною функцією яких є отримання необхідних даних, які відповідають заданим параметрам.

Після з'ясування що таке web-scraping, може здатися, що це щось, що не відповідає нормам чинного законодавства. Насправді це не так. Законом не переслідується web-scraping. Під заборону підпадають такі пункти:

– злом сайту (тобто отримання даних особистих кабінетів користувачів і т. п.);

– DDOS-атаки (якщо на сайт в результаті web-scraping даних лягає занадто високе навантаження);

– запозичення авторського контенту (фотографії з копірайтами, унікальні тексти, справжність яких засвідчена у нотаріуса і т. п. краще залишити на їх законному місці).

Web-scraping законний, якщо він стосується збору інформації, що знаходиться у відкритому доступі. Тобто все, що можна і так зібрати власноруч.

Web-scraping просто дозволяють прискорити процес і уникнути помилок через людський фактор. Тому «незаконність» в процес вони не додають.

Інша справа, як власник свіжозібраної бази розпорядиться подібною інформацією. Відповідальність може наступити саме за наступні дії.

Web-scraping використовується для аналізу цінової політики. Щоб зрозуміти середню вартість тих чи інших товарів на ринку, зручно використовувати дані по конкурентах. Однак якщо це сотні і тисячі позицій, зібрати їх вручну оперативно неможливо.

Відстеження змін. Web-scraping можна здійснювати на регулярній основі, наприклад, щотижня, виявляючи на що підвищилися ціни в середньому по ринку і які новинки з'явилися у конкурентів.

Основна проблема сучасного Інтернету - надлишок інформації, яку людина не в змозі систематизувати вручну. У порівнянні з людиною, комп'ютерна програма:

1. швидко обійде тисячі веб-сторінок;

2. акуратно відокремить технічну інформацію від «людської»;

3. безпомилково відбере потрібне і відкине зайве;

4. ефективно упакує кінцеві дані в необхідному вигляді.

Результат (будь то база даних чи електронна таблиця), звичайно ж, потребує подальшої обробки.

1.2 Аналіз видів веб скрейпінгу

Сучасні рішення для скрейпінгу варіюються від спеціальних, що вимагають людських зусиль до повністю автоматизованих систем, які здатні перетворювати цілі веб-сайти в структуровану інформацію у певному форматі. Ідеально, коли сайт, дані якого ви хочете отримати, надає їх через API з дозволеним крос-доменним доступом. Якщо справи не йдуть таким чином, можна звернутися до інших методів скрейпінгу.

Розглянемо більш детально кожен з методів скрейпінгу та визначмо оптимальний варіант для збору інформації для аналітики.

1. Копіювання даних вручну

Іноді навіть найкраща технологія веб-скрейпінгу не може замінити ручну роботу людини, коли користувач копіює та вставляє текст. У деяких випадках це єдине можливе рішення, наприклад, коли веб-сайти встановлюють блокування від веб-скрейпінгу та копіювання тексту.

2. Звернення до проксі-сервісу

Якщо сайт є html- або xml-документом і до нього дозволені крос-доменні запити, то можна отримати вміст документа за допомогою запиту до одного з наявних в Інтернеті проксі-сервісу .

3. Зіставлення текстових шаблонів

Простий, але потужний спосіб отримання інформації з веб-сторінок. Може бути заснований на команді UNIX grep (виконує пошук в одному або декількох файлах за шаблоном) або зіставлення регулярних виразів мов програмування (наприклад, Perl або Python).

4. Синтаксичний аналіз HTML

Багато веб-сайтів складаються з великої кількості сторінок, що динамічно генеруються з основного структурованого джерела – бази даних. Дані однієї категорії зазвичай кодуються в схожі сторінки за допомогою загального скрипта або шаблону. В інтелектуальному аналізі даних програма, яка виявляє такі шаблони у певному джерелі інформації, витягує його вміст та переводить його у форму, називається оболонкою. Передбачається, що аналізовані сторінки системи відповідають загальному шаблону і їх можна легко ідентифікувати в термінах загальної схеми URL. Крім того, деякі напівструктуровані мови запитів до даних, такі як XQuery та HTQL, можуть використовуватися для аналізу HTML-сторінок та отримання та перетворення вмісту сторінок.

5. Document Object Model (DOM)

DOM - програма з API для HTML-і XML-документів. Вбудовуючи повноцінний веб-браузер, наприклад Internet Explorer або елемент керування браузера Mozilla, програми можуть отримувати динамічний вміст, створюваний клієнтськими сценаріями. Скрейпінг DOM-дерева дозволяє отримати доступ до інформації в окремих її частях.

6. Вертикальна агрегація даних

Є кілька компаній, які розробили спеціальні онлайн-платформи, які створюють та контролюють безліч ботів. Боти працюють без прямої участі людини і при цьому їхня взаємодія з користувачами відбувається без зв'язку з цільовим сайтом. Підготовка включає створення бази знань, завдяки якій можлива робота роботів. Боти здійснюють агрегацію даних за окремими властивостями кожного ресурсу відповідно до заданих умов для подальшого зіставлення та аналізу отриманих значень властивостей. Надійність платформи вимірюється якістю отриманої інформації (зазвичай кількістю полів) та її

масштабованістю (до сотень чи тисяч сайтів). Ця масштабованість переважно використовується для перетворення даних, розташованих наприкінці довгого коду сайтів, які звичайні агрегатори вважають складними або надто трудомісткими для збору контенту.

7. Розпізнавання семантичних анотацій

Деякі сторінки можуть містити метадані або семантичну розмітку та анотації, за допомогою методу розпізнавання семантичних анотацій їх можна витягувати з таких сторінок.

8. Аналізатори сторінок

Ведуться розробки у сфері штучного інтелекту, коли машинний зір ідентифікує дані, інтерпретує їх, як це робить людина, та вилучає.

Технологія веб-скрейпінг дозволяє здійснювати отримання даних з веб-сторінок для подальшого їх аналізу. Враховуючи, що вміст сторінки може динамічно змінюватись, найбільш вдалою технологією для збору даних є Document Object Model (DOM).

1.3 Основні терміни та компоненти парсера

1.3.1 Звичайні вирази

Регулярний вирази часто радять не використовувати для аналізу. Це не зовсім правильно, тому що можна використовувати регулярні вирази для аналізу простого введення. Проблема виникає коли для використання доступні лише регулярні вирази. Використовування їх, щоб спробувати розібрати те, що не підходить призводить до створення крихкої системи яка може не впоратись з своєю задачею.

Можна використовувати регулярні вирази для аналізу деяких простих мов, але це виключає більшість мов програмування. Навіть ті, що виглядають досить просто, як HTML. Фактично мови, які можна аналізувати тільки за допомогою регулярних виразів, називаються регулярними мовами.

Одним із важливих наслідків теорії про регулярні мови є те, що вони можуть бути проаналізовані або виражені також за допомогою кінцевого автомата. Тобто регулярні вирази та кінцеві автомати однаково потужні. Причина в тому, що вони використовуються для реалізації лексерів.

Звичайна мова може бути визначена серією регулярних виразів, тоді як складніші мови вимагають чогось більшого. Просте правило: якщо граматика мови має рекурсивні чи вкладені елементи, це звичайна мова. Наприклад, HTML може містити довільну кількість тегів усередині будь-якого тега, тому не є звичайною мовою і не може бути проаналізований з використанням виключно регулярних виразів, незалежно від того, наскільки вона досконала.

Знайомство з регулярними виразами дозволяє їх часто використовувати в визначенні граматики мови. Точніше, їх синтаксис використовується в визначенні правил лексера чи парсера. Наприклад, зірка Кліні (*) використовується в правилі, щоб вказати, що конкретний елемент може бути нуль або нескінченна кількість разів.

Визначенні правила не слід плутати з реалізацією фактичного лексера чи парсера. Можете реалізувати лексер, використовуючи механізм регулярних виразів, що надається мовою програмування. Однак зазвичай регулярні вирази, визначені в граматиці, перетворюють і перетворюються на кінцевий автомат для підвищення продуктивності.

1.3.2 Структура парсера

Виходячи з ролі регулярних виразів, можна подивитись загальну структуру синтаксичного аналізатора. Повний аналізатор зазвичай складається з двох частин: лексера, також відомого як сканер або токенізатор, і правильного синтаксичного аналізатора. Для синтаксичного аналізатора потрібен лексер, оскільки він працює над текстом, і з висновком, створеним лексером. Не всі парсери приймають цю двоетапну схему: деякі парсери не залежать від окремого лексера та поєднують два кроки. Їх називають сканерами без сканера.

Лексер і аналізатор працюють послідовно: лексер сканує вхідні дані і видає токени, аналізатор потім сканує токени і видає результат аналізу.

Приклад розбору стрічки “437 + 734”. Лексер сканує текст і знаходить 4 , 3 , 7 а потім пробіл ( ). Завдання лексера - визнати, що символи 437 складають один токен типу NUM. Потім лексер знаходить символ +, який відповідає другому токену типу PLUS, і, нарешті, він знаходить інший токен типу NUM(рисунок 2.1).

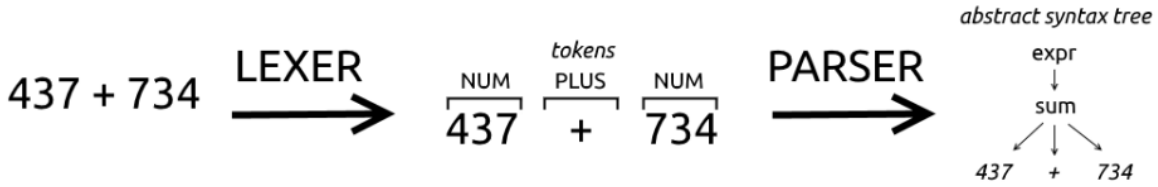


Рисунок 1.1 – Приклад розбору

Парсер зазвичай поєднує токени, створені лексером, і групує їх. Визначення, що використовуються лексерами та парсерами, називаються правилами або продукцією. У прикладі правило лексера буде вказувати, що послідовність цифр відповідає токену типу NUM, а правило синтаксичного аналізатора буде вказувати, що послідовність токенів типу NUM, PLUS, NUM відповідає виразу суми .

В наш час типово знайти набори, які можуть генерувати як лексер, і парсер. У минулому замість цього було більш поширеним поєднувати два різні інструменти: один для створення лексера та один для створення парсера. Наприклад, це було у випадку з парою lex&yacc: за допомогою lex можна було згенерувати лексер, а за допомогою yacc можна було згенерувати парсер.

1.3.3 Сканер без сканера

Парсери без сканера працюють по-різному, тому що вони обробляють безпосередньо вихідний текст замість обробки списку токенів, створених лексером. Тобто парсер без сканера працює як лексер та парсер разом.

Хоча для цілей налагодження, безумовно, важливо знати, чи є конкретний інструмент аналізу синтаксичним аналізатором без сканера чи ні, у багатьох випадках немає сенсу визначати граматику. Це тому, що зазвичай визначається шаблон, який групує послідовність символів для створення (віртуальних) токенів, які потім поєднуються для отримання остаточного результату. Іншими словами, граматика синтаксичного аналізатора без сканера дуже схожа на граматику для інструмента з окремими кроками.

1.3.4 Лексер

Лексери також відомі як сканери або токенізатори. Лексери відіграють роль у синтаксичному аналізі, тому що вони перетворять початкове введення у форму, яка більш керована відповідним аналізатором, який працює на пізнішому етапі. Зазвичай лексери легше писати, ніж парсери. Хоча є особливі випадки, коли обидва досить складні.

Дуже важливою частиною роботи лексера є робота з пробілами. Найчастіше потрібно, щоб лексер відкидав прогалини. Це тому, що інакше парсер повинен був би перевіряти наявність прогалин між кожним токеном, що швидко ставало проблемою.

Є випадки, коли не можна це зробити, тому що пропуск має відношення до мови, як у випадку з Python, де використовується для ідентифікації блоку коду. Однак навіть у цих випадках зазвичай саме лексер вирішує проблему відхилення релевантної прогалини від несуттєвої. Це означає, що потрібно, щоб лексер розумів, які прогалини мають відношення до аналізу. Наприклад, при розборі Python ви потрібно, щоб лексер перевірив, чи визначають пробіли відступи (релевантні) чи пробіл між ключовим словом if та наступним виразом (неактуальним).

Враховуючи, що лексери майже виключно використовуються разом із синтаксичними аналізаторами(парсерами), розділова лінія між ними може бути розмита час від часу. Це тому, що синтаксичний аналіз повинен давати

результат, корисний для конкретної потреби програми. Таким чином, існує не тільки один правильний спосіб синтаксичного аналізу чогось, але ви дбаєте лише про те, щоб задовольнити ваші потреби.

Наприклад, створюється програма, яка має аналізувати журнали сервера, щоб зберегти їх у базі даних. З цією метою лексер визначить послідовність чисел і точок і перетворить в токен IPv4(рисунок 2.2).

```
IPv4: [0-9]+ "." [0-9]+ "." [0-9]+ "." [0-9]+
```

Рисунок 1.2 – Токен IPv4

Потім аналізатор проаналізує послідовність токенів, щоб визначити, чи є це повідомленням, попередженням.

Що б сталося натомість, якби розробляли програмне забезпечення, яке мало використовувати IP-адресу для визначення країни відвідувача? Можливо, потрібно, щоб лексер ідентифікував байти адреси для подальшого використання і зробив IPv4 елементом синтаксичного аналізатор(рисунок 2.3).

```
DOT   : "."
OCTET : [0-9]+

ipv4  : OCTET DOT OCTET DOT OCTET DOT OCTET
```

Рисунок 1.3 – Токен IP-адреси

Це один із прикладів того, як та сама інформація може бути проаналізована по-різному через різні цілі.

1.3.5 Синтаксичний аналізатор

У контексті аналізу синтаксичний аналізатор може відноситися як до програмного забезпечення, яке виконує весь процес, так і просто до відповідного аналізатора, який аналізує токени, що генеруються лексером. Це просто наслідок того факту, що синтаксичний аналізатор піклується про найважливішу і складну частину всього процесу синтаксичного аналізу. Під найважливішим ми маємо на

увазі, що користувач дбає про більшість і буде насправді бачить. Насправді лексер працює як помічник для полегшення роботи парсера.

У певному сенсі мається на увазі синтаксичний аналізатор, його висновок - це організована структура коду, зазвичай дерево. Дерево може бути деревом розбору або абстрактним синтаксичним деревом. Обидва є деревами, але відрізняються тим, наскільки близько вони представляють реальний написаний код і проміжні елементи, визначені синтаксичним аналізатором.

Форма дерева обрана тому, що це простий та природний спосіб роботи з кодом на різних рівнях деталізації. Наприклад, клас у C# має одне тіло, це тіло складається з одного оператора, оператора блоку, який є списком операторів, укладених у пару фігурних дужок, і так далі…

1.3.6 Синтаксична та семантична коректність

Парсер – це фундаментальна частина компілятора чи інтерпретатора, але, звісно, може бути частиною іншого програмного забезпечення. Синтаксичний аналізатор може перевіряти лише синтаксичну коректність фрагмента коду, але компілятор може використовувати свій висновок також у процесі перевірки семантичної достовірності того ж фрагмента коду.

Давайте подивимося приклад коду, який синтаксично правильний, але семантично неправильний.

```
int x = 10
int sum = x + y
```

Проблема в тому, що одна змінна (y) ніколи не визначається і, отже, у разі виконання програми завершиться помилкою. Однак синтаксичний аналізатор не може цього знати, тому що він не відслідковує змінні, просто дивиться на структуру коду.

Натомість компілятор зазвичай спочатку переглядає дерево розбору і зберігає список усіх певних змінних. Потім він повторно переглядає дерево розбору і перевіряє, чи всі змінні визначені правильно. У цьому прикладі їх

немає і він видасть помилку. Так що це один із способів, яким дерево розбору може використовуватися для перевірки семантики компілятором.

1.3.7 Парсер без сканера

Парсер без сканера , чи рідше парсер без лексера, – це парсер, який виконує токенізацію (тобто перетворення послідовності символів у токенах) і правильний синтаксичний аналіз за крок. Теоретично наявність окремого лексера та аналізатора є кращою, оскільки воно дозволяє більш чітко розділити цілі та створити більш модульний аналізатор.

Сканер без сканера — найкращий дизайн для мови, де чітка різниця між лексером та парсером утруднена чи не потрібна. Прикладом є синтаксичний аналізатор для мови розмітки, де море тексту вставляються спеціальні маркери. Це також може полегшити роботу з мовами, у яких традиційний лексинг утруднений, наприклад, C. Це відбувається тому, що аналізатор без сканера може легше справлятися зі складними токенізаціями.

1.3.8 Проблеми з аналізом реальних мов програмування

Теоретично сучасний синтаксичний аналіз призначений до роботи з реальними мовами програмування, практично виникають проблеми з деякими реальними мовами програмування. Принаймні їх було б складніше розібрати, використовуючи звичайні інструменти генератора розбору.

1. Контекстно-залежні частини

Інструменти синтаксичного аналізу зазвичай призначені до роботи з контекстно-вільними мовами, але іноді мови чутливі до контексту. Це може бути у випадку спрощення життя програмістів або просто через поганий дизайн.

Типовим прикладом контекстно-залежних елементів є так звані м'які ключові слова, тобто рядки, які у певних місцях можуть розглядатися як ключові слова, але в іншому випадку можуть використовуватись як ідентифікатори.

2. Пробіли

Пробіли відіграють важливу роль у деяких мовах. Найбільш відомим прикладом є Python, де відступ оператора вказує, чи є частиною певного блоку коду.

У більшості місць пробіл не має значення навіть у Python: пробіли між словами або ключовими словами не мають значення. Ця проблема – це відступ, який використовується для ідентифікації блоку коду. Найпростіший спосіб впоратися з цим - перевірити відступ на початку рядка і перетворити на відповідний токен, тобто створити токен, коли відступ змінюється від попереднього рядка.

На практиці функції користувача в лексері виробляють токени INDENT і DEDENT, коли відступ збільшується або зменшується. Ці токени відіграють роль, яку в С-подібних мовах відіграють фігурні дужки: вони вказують початок та кінець блоків коду.

Цей підхід робить лексинг контекстно-залежним, а чи не контекстно-вільним. Це ускладнює синтаксичний аналіз, і ви зазвичай не хочете цього робити, але ви змушені це робити в цьому випадку.

3. Кілька синтаксисів

Інша поширена проблема пов'язана з тим фактом, що мова може включати кілька різних синтаксисів. Іншими словами, один і той же вихідний файл може містити розділи коду, які випливають з іншого синтаксису. У контексті ефективного аналізу той самий вихідний файл містить різні мови. Найбільш відомим прикладом є, ймовірно, препроцесор C або C++, який насправді є досить складною мовою і може чарівним чином з'являтись усередині будь-якого випадкового коду C.

Найпростішим випадком є інструкції, які є у багатьох сучасних мовах програмування. Серед іншого вони можуть бути використані для обробки коду до його надходження до компілятора. Вони можуть дати команду процесору анотацій будь-яким чином перетворити код, наприклад, для виконання певної

функції перед виконанням анотованої. Їм легше керувати, тому що вони можуть з'являтися лише у певних місцях.

4. Висячий решта

Також else є поширеною проблемою при розборі, пов'язаної з оператором if-then-else. Оскільки пропозиція else є необов'язковою, послідовність операторів if може бути неоднозначною. Наприклад:

```
if one
   then if two
       then two
else ???
```

Неясно, чи належить else першому if або другому.

Це здебільшого проблема стилю написання коду. Більшість рішень насправді не сильно ускладнює синтаксичний аналіз, наприклад, вимагає використання endif або використання блоків для розмежування оператора if коли він включає else.

Однак існують також мови, які не пропонують рішення, тобто які розроблені неоднозначно, наприклад C. Звичайний підхід полягає в тому, щоб зв'язати else з найближчим оператором if, що робить синтаксичний аналіз контекстно-залежним.

1.4 Перешкоди автоматизованого отримання даних

В інших підрозділах розглянули, що таке web-scraping для чого він потрібен, було вивчено предметну область, провели аналіз наявних рішень для оптимізації і покращення збору даних для сайту. Є кілька варіантів обмежень, які можуть ускладнити автоматизоване отримання даних:

1) User-agent. Це запит, в якому програма повідомляє сайту про себе. Web-scraping блокують багато веб-ресурсів. Однак в налаштуваннях дані можна змінити на YandexBot або Googlebot і відсилати правильні запити.

2) Robots.txt, в якому прописана заборона для індексації пошуковими роботами Яндекса або Google (ними ми представилися сайту вище) певних

сторінок. Проблема вирішується правилом яке задає ігнорування файла robots.txt.

3) Блокування за IP-адресою, якщо з неї протягом довгого часу надходять на сайт однотипні запити. Рішення - використовувати VPN для відсилання запитів.

4) CAPTCHA. Якщо дії схожі на автоматичні, виводиться CAPTCHA. Навчити систему автоматизованого отримання даних розпізнавати конкретні види досить складно і дорого.

1.5 Постановка задачі

Метою магістерської роботи є створення технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики. Це дозволить ефективно аналізувати інтернет простір, швидко реагувати на зміни трендів і та втілювати задумані бізнес ідеї в життя.

Для досягнення поставленої мети визначені наступні задачі:

– визначити типи даних, що необхідно отримувати з веб-сторінок для бізнес-аналітики;

– проаналізувати існуючі методи отримання даних з веб-ресурсів – web-scraping;

– розробити алгоритми синтаксичного розбору веб-сторінок та експорту даних;

– спроектувати інформаційну системи автоматизованого збору даних з веб-ресурсів на основі проведених досліджень;

– створити програмну реалізацію системи та провести її тестування.

Висновки до розділу 1

В першому розділі розглянута проблема збору даних з динамічних сайтів. Автоматизований збір даних дозволить ефективно аналізувати інтернет простір,

швидко реагувати на зміни трендів і втілювати задумане в життя. Ручний аналіз даних в цьому випадку набагато дорожчий, чим розробка системи і менш дієвий. Об’єктом дослідження є способи збору даних в інтернеті і покращення ефективності бізнесу за допомогою аналізу даних або швидкого заповнення сайту.

Проведено аналіз видів web-scraping та розглянуто проблеми які виникають при аналізи веб сторінок і їх рішення.

Отже, вирішено спроектувати та розробити автоматизовану систему збору даних з веб-ресурсів для бізнес-аналітики.

Розділ 2

Загальний опис алгоритму синтаксичного розбору веб-сторінки

2.1 Загальний опис інформаційної системи для веб скрапінгу

Проаналізувавши методи веб скрапнігу можна зазначити що він має досить багато перешкод для збору даних для бізнес аналітики. Так збір даних можу проводитись не тільки з нашого сайту а і з інших в передбачає декілька вимог такі як: коректне завантаження сторінки, збір даних зі сторінки, зберігання даних у зручному форматі. Запропонована інформаційна система дозволить реалізувати в собі виконання вимог для збереження даних для бізнес аналітики.

Інформаційна система складається з наступних кроків (рис 2.1)

1. Вхідні данні: сторінки які потрібно проскакувати і теги сторінки
2. Збір та обробка даних з сторінки.
3. Перехід на іншу сторінку при потребі.
4. Збереження даних у структурований файл.
5. Вихідні дінні: CSV файл з інформацією а також json.

Рисунок 2.1 Схема інформаційної системи для отримання даних

Вхідною інформацією є два об’єкта, перший з яких це посилання на сторінку, а другим є html теги для того щоб програма розуміла які дані і звідки їх потрібно взяти. Ці дані потрібні для коректного скрапінгу сторінки і і збереження інформації.

Механізм збору та обробка даних з сторінки дозволить обробити усю необхідну інформацію для подальшого формування файлу.

Механізм переходу на інші сторінки. Також будуть збиратись інгші посилання на внутрішні сторінки для подальшого переходу між ними без участі користувача.

Під час виконання попередніх пунктів дані зберігають в двох форматах для зручного транспортування і аналітики. Це дозволить при потребі внести їх в базу даних або іншу програму для аналітики і перевірити те що цікавить

На виході буде отримано файл з повність зібраною інформацією і готовим для подальших маніпуляцій над ним.

2.2 Алгоритми синтаксичного аналізу

Є дві стратегії для аналізу: аналіз зверху вниз і аналіз знизу вгору. Обидва терміни визначені стосовно дерева аналізу, згенерованого синтаксичним аналізатором:

– аналізатор, що працює зверху вниз, спочатку намагається ідентифікувати корінь дерева розбору, потім він переміщається вниз по під деревам, доки знайде листя дерева.

– аналіз знизу вгору починається з нижньої частини дерева, листя і піднімається вгору, доки визначиться корінь дерева.

Подивимося на приклад, починаючи з дерева розбору (рис. 2.2).

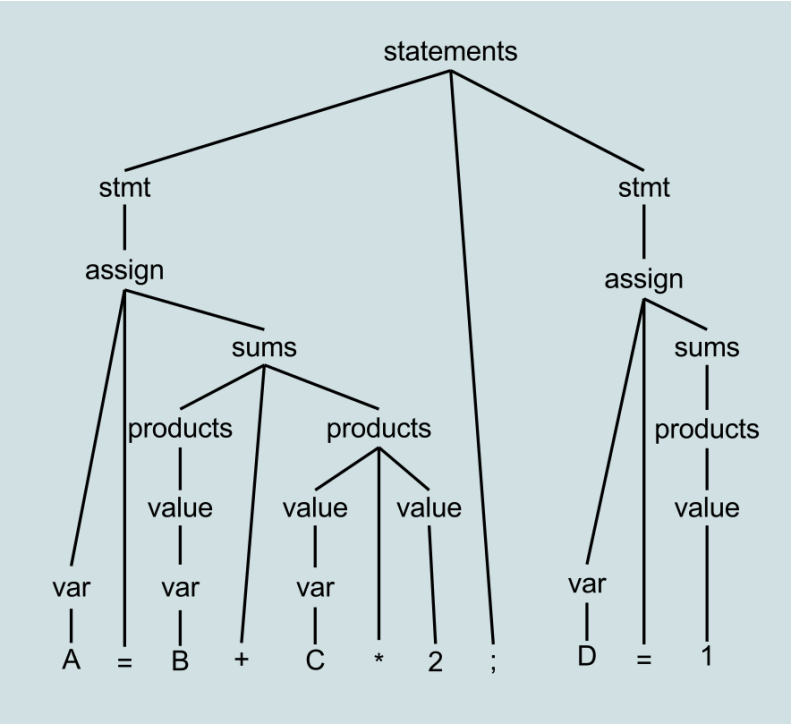


Рисунок 2.2 – Дерево розбору

Те саме дерево буде згенеровано в іншому порядку за допомогою синтаксичного аналізатора зверху вниз (рис. 2.3). і знизу вгору (рис. 2.4).

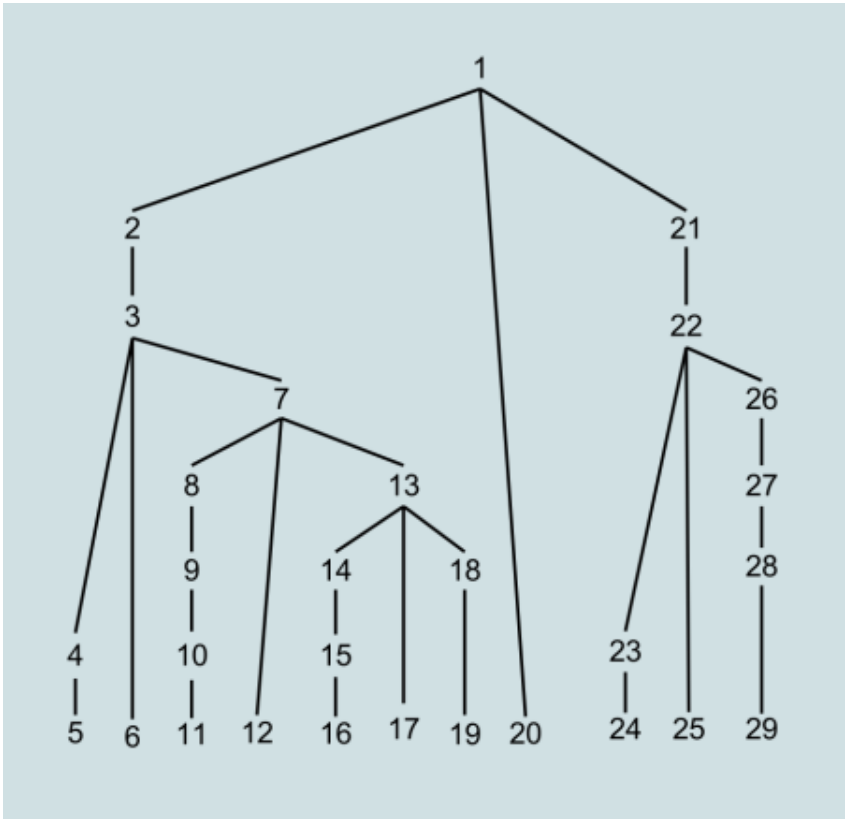


Рисунок 2.4 – Низхідний порядок генерації дерева

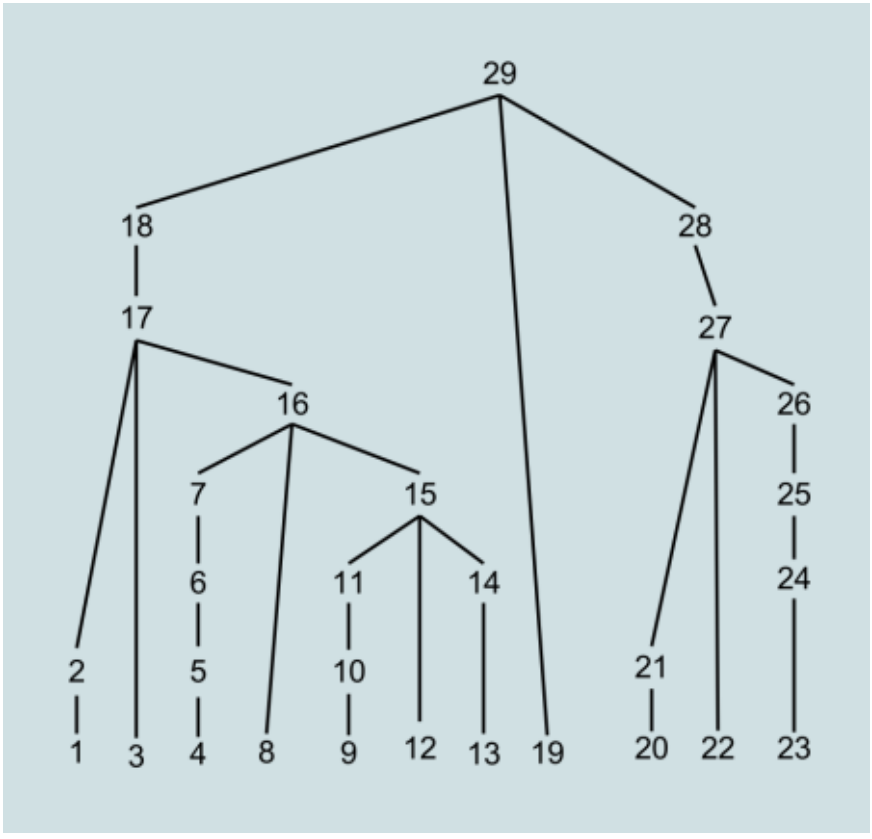


Рисунок 2.6 – Порядок генерації дерева знизу догори

Традиційно синтаксичні аналізатори створювалися простіше, але аналізатори знизу нагору були потужнішими. Тепер ситуація більш збалансована головним чином через просування стратегій синтаксичного аналізу зверху вниз.

У таблиці наведено короткий опис основних особливостей різних алгоритмів синтаксичного аналізу та того, для чого вони зазвичай використовуються.

LL Parser

LL (L EFT-направо читання вхідного, L eftmost виведення) парсери таблиця синтаксичних аналізаторів на основі без повернень, але з випереджальним переглядом. На основі таблиці означає, що вони покладаються на таблицю аналізу, щоб вирішити, яке правило застосувати. Таблицю синтаксичного аналізу використовують як рядки і стовпці нетерміналів і терміналів відповідно.

Щоб знайти правильне правило для застосування: по-перше, парсер переглядає поточний токен та відповідну кількість лексем-ключів потім він намагається застосувати інші правила, доки знайде правильне відповідність.

Концепція парсера LL належить немає конкретного алгоритму, а скоріш до класу парсерів. Вони визначаються стосовно граматики. Тобто синтаксичний аналізатор LL це той, який може аналізувати граматику LL. У свою чергу граматики LL визначаються залежно від кількості лексем, які необхідні для їх аналізу. Це число вказується у дужках поруч із LL, тому у формі LL (k).

Синтаксичний аналізатор LL (k) використовує k токенів попереднього перегляду і, таким чином, може аналізувати не більше граматики, для якої необхідно проаналізувати k токенів попереднього перегляду. Ефективно поняття LL (k) граматики використовується ширше, ніж відповідний парсер. Це означає, що граматики LL (k) використовуються як вимірювач при порівнянні різних алгоритмів. Наприклад, ви б прочитали, що PEG-парсери можуть обробляти граматики LL (*).

Значення граматик LL

Таке використання граматик LL пов'язане як із тим, що синтаксичний аналізатор LL широко використовується, так і з тим, що він трохи обмежений. Фактично граматика LL не підтримує ліворекурсивні правила. Ви можете перетворити будь-яку ліворекурсивну граматику на еквівалентну нерекурсивну форму, але це обмеження має значення з кількох причин: продуктивність та потужність.

Втрата продуктивності залежить від того, що ви повинні написати граматику особливим чином, що потребує часу. Потужність обмежена, тому що граматика, яка може вимагати 1 токен попереджувального запиту, при написанні з ліворекурсивним правилом, може вимагати 2-3 токена попереджувального перегляду, якщо вона написана не рекурсивним способом. Таким чином, це обмеження не просто дратує, але воно обмежує потужність алгоритму, тобто граматики, на яку він може використовуватися.

Втрата продуктивності може бути зменшена за допомогою алгоритму, який автоматично перетворює ліворекурсивну граматику на нерекурсивну. ANTLR це інструмент, який може зробити це, але, звичайно, якщо ви створюєте свій власний парсер, ви повинні зробити це самостійно.

Існує два спеціальні види граматик LL (k): LL (1) і LL (*). У минулому перші види були єдиними, які вважалися практичними, бо їм легко створювати ефективні парсери. До того моменту, що багато комп'ютерних мов було спеціально розроблено, щоб описуватися граматикою LL (1). LL (*), також відомий як LL-регулярний парсер, може працювати з мовами, використовуючи нескінченну кількість лексем.

Парсер Earley

Парсер Earley – це парсер діаграм, названий на честь свого винахідника Джея Ерлі. Алгоритм зазвичай порівнюють з CYK, іншим синтаксичним аналізатором діаграм, який простіше, але також зазвичай гірше за продуктивністю та пам'яті. Відмінною особливістю алгоритму Ерлі є те, що, крім

збереження часткових результатів, він реалізує крок прогнозування, щоб вирішити, яке правило намагатиметься відповідати наступному.

Привабливість синтаксичного аналізатора Earley полягає в тому, що він гарантовано зможе аналізувати всі мови без контексту, тоді як інші відомі алгоритми (наприклад, LL, LR) можуть аналізувати лише їхнє підмножина. Наприклад, у нього немає проблем із ліворекурсивними граматиками. У більш загальному сенсі парсер Earley може мати справу з недетермінованими і неоднозначними граматиками.

Це може зробити це з ризиком погіршення продуктивності (O (n 3)) у гіршому випадку. Однак він має лінійну продуктивність часу для звичайних граматик. Суть у цьому, що набір мов, аналізованих традиційнішими алгоритмами, — це, який нас зазвичай цікавить.

Існує також побічний ефект відсутності обмежень: змушуючи розробника писати граматику певним чином, синтаксичний аналіз може бути ефективнішим. Тобто побудова граматики LL (1) може бути складнішою для розробника, але аналізатор може застосовувати його дуже ефективно. З Earley ви виконуєте менше роботи, тож аналізатор робить більше.

Коротше кажучи, Earley дозволяє використовувати граматики, які легко писати, але які можуть бути неоптимальними з точки зору продуктивності.

Випадки використання Earley

Таким чином, парсери Earley прості у використанні, але перевага, з погляду продуктивності, в середньому може бути відсутнім. Це робить алгоритм відмінним для освітнього середовища або в тих випадках, коли продуктивність важливіша за швидкість.

У першому випадку це корисно, наприклад, тому що більшість часу граматики, які пишуть ваші користувачі, працюють просто відмінно. Проблема в тому, що парсер видаватиме на них незрозумілі і, здавалося б, випадкові помилки. Звичайно, помилки не випадкові, але вони пов'язані з обмеженнями алгоритму, які ваші користувачі не знають або не розуміють. Таким чином, ви

змушуєте користувача розуміти внутрішню роботу парсера, щоб використовувати його, що має бути непотрібним.

Прикладом, коли продуктивність важливіша за швидкість, може бути генератор синтаксичного аналізатора для реалізації підсвічування синтаксису для редактора, який повинен підтримувати безліч мов. В аналогічній ситуації можливість швидкої підтримки нових мов може бути бажанішою, ніж якнайшвидше виконати завдання.

Пакрат (ПЕГ)

Packrat часто асоціюється з формальною граматикою PEG, оскільки вони були винайдені однією і тією ж людиною: Брайаном Фордом. Пакрат був описаний першим у своїй дисертації: «Аналіз Пакрата: практичний алгоритм лінійного часу зі зворотним відстеженням». Назва говорить майже про все, що нас хвилює: вона має лінійний час виконання, у тому числі й тому, що не використовує повернення.

Інша причина його ефективності – це запам'ятовування: збереження часткових результатів у процесі аналізу. Недоліком і причиною, що метод не використовувався донедавна, є кількість пам'яті, необхідне зберігання всіх проміжних результатів. Якщо потрібна пам'ять перевищує доступну, алгоритм втрачає лінійний час виконання.

Packrat також не підтримує ліворекурсивні правила, що є наслідком того, що PEG вимагає завжди вибирати перший варіант. Насправді, деякі варіанти можуть підтримувати прямі ліворекурсивні правила, але за рахунок втрати лінійної складності.

Парсер Packrat може працювати з безліччю попереджень, якщо це необхідно. Це впливає на час виконання, що в гіршому випадку може бути експонентним.

Парсер рекурсивного спуску

Парсер рекурсивного спуску - це парсер, який працює з набором (взаємно) рекурсивних процедур, як правило, по одній для шкірного правила

граматики. Отже, структура синтаксичного аналізатора відбиває структуру граматики.

Термін предиктивний синтаксичний аналізатор використається декількома різними способами: деякі люди розуміють його як синонім синтаксичного аналізатора згори донизу, деякі як синтаксичний аналізатор із рекурсивним спуском, який ніколи не повертається назад.

Протилежністю до цього іншого значення є парсер рекурсивного спуску, який виконує повернення. Тобто той, який знаходить правило, яке збігається з введенням, виконуючи кожне з правил у послідовності, і потім воно повертається щоразу, коли зазнає невдачі.

Зазвичай у синтаксичного аналізатора рекурсивного спуску виникають проблеми з розбором ліворекурсивних правил, тому що алгоритм в кінцевому підсумку буде викликати ту саму функцію знову і знову. Можливе вирішення цієї проблеми використання хвостової рекурсії. Парсері, які використовують цей метод, називаються хвостовими рекурсивними парсерами.

Хвостова рекурсія як така — це просто рекурсія, що відбувається наприкінці функції. Проте хвостова рекурсія використовують у поєднанні з перетвореннями правил граматики. Комбінація перетворення правил граматики та приміщення рекурсії у кінець процесу дозволяє працювати з ліворекурсивними правилами.

Пратт Парсер

Парсер Pratt є широко не використовуваним, але високо цінується (мало хто знає) алгоритмом синтаксичного аналізу, визначеним Воганом Праттом у статті "Пріоритет оператора зверху вниз". Сама стаття починається з полеміки про граматики БНФ, які автор помилково стверджує, що вони є винятковим завданням парсингу досліджень. Це одна із причин відсутності успіху. Фактично алгоритм не спирається на граматику, а працює безпосередньо з токенами, що робить його незвичайним для експертів з аналізу.

Друга причина полягає в тому, що традиційні низхідні парсери чудово працюють, якщо у вас є значний префікс, який допомагає розрізняти різні

правила. Наприклад, якщо ви отримуєте токен FOR, ви переглядаєте оператора for. Оскільки це в основному стосується всіх мов програмування та їхніх операторів, легко зрозуміти, чому аналізатор Pratt не змінив світ синтаксичного аналізу.

Де алгоритм Pratt сяє із виразами. Фактично, концепція пріоритету унеможливлює розуміння структури вхідних даних, просто глянувши на порядок, у якому представлені токени.

По суті, алгоритм вимагає, щоб ви надавали значення пріоритету кожному токену оператора і пару функцій, які визначають, що робити, відповідно до того, що знаходиться ліворуч і праворуч від токена. Потім він використовує ці значення та функції для зв'язування операцій під час проходження введення.

Хоча алгоритм Пратта не був явно успішним, він використовується для аналізу виразів. Він також прийнятий Дугласом Крокфордом (з відомості JSON) для JSLint.

Parser Combinator

Комбінатор синтаксичного аналізатора - це функція вищого порядку, яка приймає функції синтаксичного аналізатора як вхідні дані і повертає нову функцію синтаксичного аналізатора як вихідні дані. Функція парсера зазвичай означає функцію, яка приймає рядок та виводить дерево розбору.

Комбінатор синтаксичного аналізатора є модульним і простим у збірці, але він також повільніше (складність O (n 4) у гіршому випадку) і менш складний. Вони зазвичай використовуються для полегшення аналізу чи створення прототипів. У певному сенсі, користувач комбінатора синтаксичного аналізатора будує парсер частково вручну, але покладаючись на жорстке слово, зроблене тим, хто створив комбінатор синтаксичного аналізатора.

Зазвичай вони не підтримують ліві рекурсивні правила, але є просунутіші реалізації, які роблять саме це.

Багато сучасних реалізації називаються монадичним синтаксичним аналізатором, оскільки вони засновані на структурі функціонального

програмування, званої монадою. Монади — досить складне поняття, яке ми можемо сподіватися пояснити тут. Проте в основному монада здатна комбінувати функції та дії, покладаючись на тип даних. Важливою особливістю є те, що тип даних визначає, як різні значення можуть бути об'єднані.

Найпростіший приклад - монада Maybe. Це обгортка навколо нормального типу, такого як ціле число, яке повертає саме значення, коли значення є дійсним (наприклад, 567), але спеціальне значення Nothing, коли це не так (наприклад, undefined або поділ на нуль). Таким чином, ви можете уникнути використання нульового значення та марного збою програми. Натомість значення Nothing управляється звичайним чином, як і будь-яке інше значення.

CYK Parser

Cocke-Younger-Kasami (CYK) було сформульовано незалежно трьома авторами. Його популярність обумовлена великою продуктивністю у гіршому випадку (O (n 3)), хоча в більшості поширених сценаріїв йому заважає порівняно погана продуктивність.

Однак реальним недоліком алгоритму є те, що він вимагає, щоб граматика була виражена у нормальній формі Хомського.

Це тому, що алгоритм спирається на властивості цієї конкретної форми, щоб мати можливість розділити вхідні дані навпіл, щоб спробувати порівняти всі можливості. Тепер, теоретично, будь-яка граматика, що не залежить від контексту, може бути перетворена у відповідний CNF, але це рідко буває практично робити вручну. Уявіть, що вас дратує той факт, що ви не можете використовувати ліворекурсивні правила, і вас просять вивчити особливий вид форм.

Алгоритм CYK використовується переважно для конкретних завдань. Наприклад, проблема членства: визначити, чи сумісний рядок із певною граматикою. Він також може бути використаний для обробки природної мови для пошуку найбільш ймовірного синтаксичного аналізу між багатьма опціями.

Для всіх практичних цілей, якщо вам потрібно проаналізувати всю неконтекстну граматику з великою продуктивністю у найгіршому випадку, ви хочете використовувати аналізатор Earley.

LR Parser

LR (L EFT-направо читання вхідних, R ightmost виводу) є аналізатори знизу вгору парсери , які можуть обробляти детерміновані контекстно-вільні мови в лінійний час, з попереджувальною вибіркою та без повернень. Винахід парсерів LR приписується відомому Дональд Кнут.

Традиційно їх порівнювали та порівнювали з LL-парсерами. Таким чином, існує аналогічний аналіз, пов'язаний з кількістю очікуваних токенів, необхідних для аналізу мови. Синтаксичний аналізатор LR (k) може аналізувати граматики, для аналізу яких потрібно k токенів попереднього перегляду. Однак граматики LR менш суворі і, отже, потужніші, ніж відповідні граматики LL. Наприклад, немає потреби виключати ліворекурсивні правила.

Технічно граматики LR - це розширений набір граматик LL. Одним із наслідків цього є те, що вам потрібні лише граматики LR (1), тому зазвичай (k) опускається.

Вони також ґрунтуються на таблицях, як і LL-парсери, але їм потрібні дві складні таблиці. У дуже простих термінах:

– одна таблиця повідомляє парсеру, що робити залежно від поточного токена, стану та токенів, які можуть слідувати за поточним (l ookahead sets)

– інший повідомляє парсеру, в який стан перейти

Таблиці, яких вони потребують, важко побудувати вручну і можуть зрости дуже великими для звичайних комп'ютерних мов, тому зазвичай вони переважно використовуються через генератори парсерів. Якщо потрібно зібрати парсер вручну перевага парсеру зверху вниз.

Таблиця 2.1 – Короткий порівняння алгоритмів

АЛГОРИТМ	ОСНОВНІ РИСИ	ВИКОРИСТАННЯ
CYK	Чудова продуктивність у найгіршому випадку (0(n3)). Граматика вимагає спеціальної форми CNF.	Конкретні проблеми пошуку елементів.
Earley	чудова продуктивність у найгіршому випадку (0(n3)), зазвичай лінійна продуктивність. Може обробляти всі види граматик і мови.	Генератори парсерів, які повинні вміти та володіти всіма граматиками та мовами.
LL	Легко реалізувати. Менш потужний, ніж більшість інших алгоритмів (наприклад, не може обробляти ліворекурсивні правила). Історично був найпопулярнішим вибором.	Створені вручну парсери; генератори аналізаторів, їх легше будувати
LR	Важко реалізувати (деякі варіанти є легше за інших). Потужний (може обробляти більшість граматик, деякі варіанти можуть обробляти всі). чудова продуктивність	Генератори парсерів з найкращою продуктивністю

	(зазвичай лінійна, більше потужні варіанти можуть мати найгірший випадок продуктивність 0(n3)).	
Packerat(PEG)	Лінійний час виконання використовує спеціальний формат призначений для розбору комп'ютерних мов.	Прості, але потужні парсери або парсери генератори комп'ютерних мов.
Parser Combinator	Модульний і простий у збірці. Порівняно погана продуктивність (0(n4) в найгірший випадок).	Генератори парсерів, які прості у використанні і розумінні.
Patt	Своєрідний і менш відомий алгоритм, граматика не потрібна.	Розбір виразів або інших випадків у який пріоритет не диктується порядок появи.

2.3 Розробка алгоритму синтаксичного розбору веб-сторінок

Для розробки веб скрапера був обраний метод рекурсивного спуску - алгоритм низхідного синтаксичного аналізу, що реалізується шляхом взаємного виклику процедур, де кожна процедура відповідає одному з правил контекстно-вільної граматики. Застосування правил послідовно, ліворуч поглинають токени, отримані від лексичного аналізатора. Це один із найпростіших алгоритмів синтаксичного аналізу, що підходить для повністю ручної реалізації.

Такий парсер може вимагати експоненційного часу роботи, і не завжди гарантує завершення залежно від граматики. Але він чудово підходить для роботи з дом деревом сайту та збору інформацію тому що воно схоже на дерево

по якому ми рухаємось до кореня в пошуках потрібних елементів. Алгоритму синтаксичного розбору веб-сторінок буде складатись з наступних етапів(рис. 2.7):

1. Вхідні дані.
2. Добування коду сторінки.
3. Добування даних з коду сторінки, синтаксичний аналіз.
4. Збереження результату.
5. Перехід на іншу сторінку.
6. Кінцеве збереження даних для аналітики.

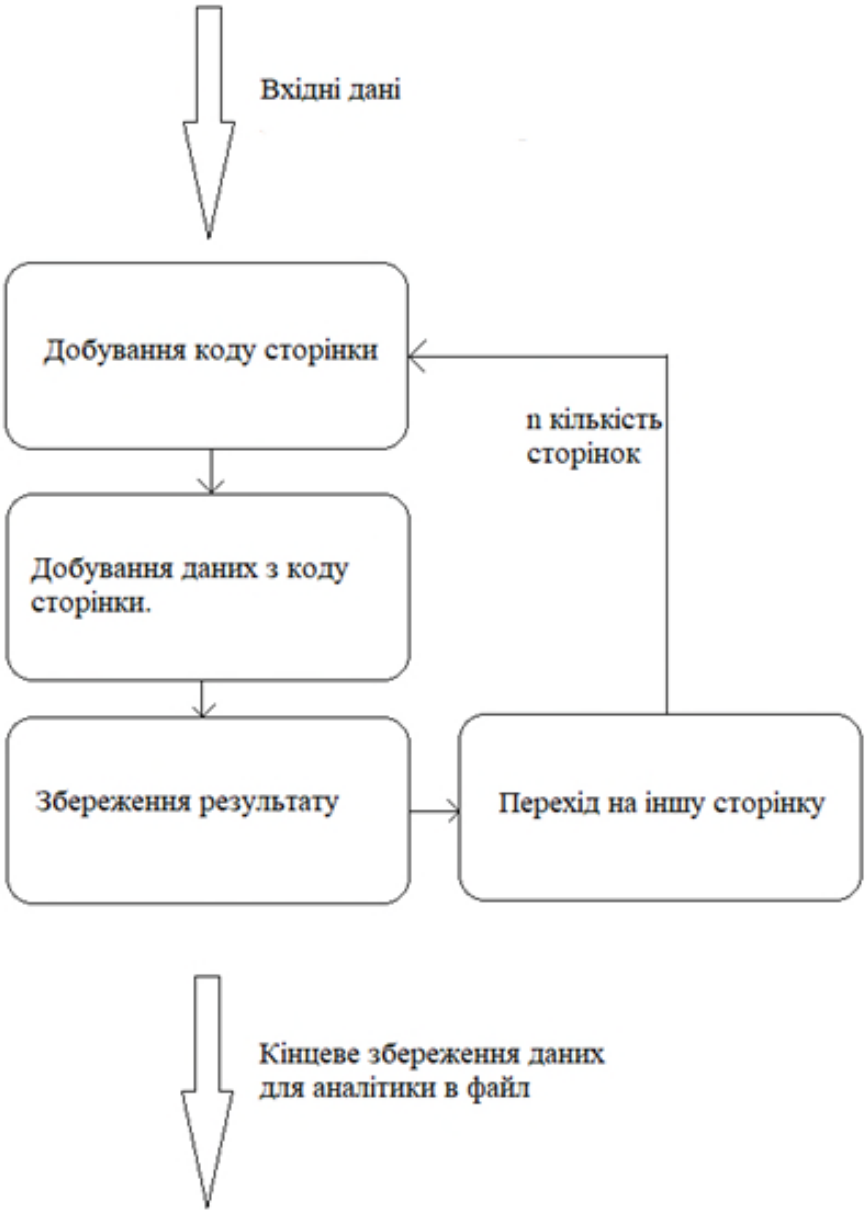


Рисунок 2.5 – Алгоритму синтаксичного розбору веб-сторінок

Також було вирішено використати асинхронний підхід для виконання цього алгоритму. Асинхронність у програмуванні — виконання процесу в режимі не блокування системного виклику, що дозволяє потоку програми продовжити обробку. Це дозволить покращити стандартний алгоритм і збільшити його швидкість.

Процес отримання (парсингу) даних зі веб ресурсу буде складається з таких етапів (рис. 2.5.):

1. Коректне(без помилок) завантаження веб-сторінки.
2. Отримання з DOM документа потрібних даних.
3. Експорт даних у файл CSV.
4. Збереження даних у файл CSV формату для подальшого аналізу, та їх перевірка.

Після вдалої обробки даних , потрібно їх зберегти у вигляді необхідному для аналізу. Інформацію яку отримали з веб ресурсу потрібно записати CSV-файл. Для аналізу веб ресурсу, потрібно про сканувати багато сторінок сайту, тобто всі які потрібні для аналізу. У цьому випадку після проходження всіх етап у алгоритм повинен бути закладений перехід на наступну сторінку сайту, далі вилучати потрібні дані.

2.5 Вдосконалення методу для доступу до веб-ресурсів

1. Блокування по IP адресі. Найпростішим і поширеним способом визначення спроб веб скрайпінгу сайту є аналіз частоти і періодичності запитів до сервера. Якщо з якогось IP адресу запити йдуть занадто часто або їх занадто багато, то ця адреса блокується і щоб його розблокувати часто пропонується ввести каптчу.

Найголовніше в цьому способі захисту - знайти межу між природною частотою і кількістю запитів і спробами скрейпінга щоб не заблокувати ні в чому не винних користувачів. Зазвичай це визначається за допомогою аналізу поведінки нормальних користувачів сайту. Прикладом використання цього

методу може служити Google, який контролюється кількість запитів з певної адреси і видає відповідне попередження з блокуванням IP адреси і пропозицією ввести каптчу.

Є сервіси (на приклад distilnetworks.com), які дозволяють автоматизувати процес відстеження підозрілої активності на вашому сайті і навіть самі включають перевірку користувача за допомогою каптчі.

Обхід цього захисту здійснюється за допомогою використання декількох проксі-серверів, які приховують реальну IP-адресу парсера. Наприклад сервіси типу BestProxyAndVPN надають недорогі проксі, а сервіс SwitchProxy хоч і дорожче, але спеціально призначений для автоматичних парсерів і дозволяє витримати великі навантаження.

2. Використання CAPTCHA. Це теж поширений метод захисту даних від парсингу. Тут користувачеві для доступу до даних сайту пропонується ввести капчу (CAPTCHA). Істотним недоліком цього способу можна вважати незручність користувача в необхідності введення капчі. Тому цей метод найкраще застосовувати в системах, де доступ до даних здійснюється окремими запитами і не дуже часто.

Прикладом використання каптчі для захисту від автоматичного створення запитів можуть служити сервіси перевірки позиції сайту в пошуковій видачі

Обходиться каптча за допомогою програм і сервісів по її розпізнаванню. Вони діляться на дві основні категорії: автоматичне розпізнавання без участі людини (OCR, наприклад програма GSA Captcha Breaker) і розпізнавання за допомогою людини (коли десь в Індії сидять люди і в режимі онлайн обробляють запити на розпізнавання картинок, наприклад може служити сервіс Bypass CAPTCHA). Людське розпізнання зазвичай більш ефективно, але оплата в даному випадку відбувається за кожну каптчу, а не один раз, як при покупці програми.

3. Використання складної JavaScript логіки. Тут в запиті до сервера браузер відсилає спеціальний код (або декілька кодів), які сформовані складною логікою написаною на JavaScript. При цьому, часто код цієї логіки

обфусцированний і розміщений в одному або декількох підвантажуваних JavaScript-файлах.

Типовим прикладом використання даного методу захисту від парсинга є Facebook. Обходиться це за допомогою використання для парсинга реальних браузерів (наприклад, за допомогою бібліотек Selenium або Mechanize). Але це дає цим методом додаткових переваг: виконуючи JavaScript, парсер буде проявляти себе в аналітиці відвідуваності сайту (наприклад Google Analytics), що дозволить вебмайстру відразу помітити недобре.

4. Динамічна зміна структури сторінки. Один з ефективних способів захисту від автоматичного парсинга - це часта зміна структури сторінки. Це може стосуватися не тільки зміна назв ідентифікаторів і класів, але навіть і ієрархії елементів. Це сильно ускладнює написання парсеру, але з іншого боку ускладнює і код самої системи. З іншого боку, ці зміни можуть робитися в ручному режимі десь раз на місяць (або кілька місяців). Це теж істотно зіпсує життя парсерам.

Щоб обійти такий захист потрібне створення більш гнучкого і «розумного» парсеру або ж (якщо зміни робляться не часто) просто ручне виправлення парсеру, коли ці зміни відбулися.

2.5 Експорт даних

JSON-текст є (в закодованому вигляді) однією з двох структур:

Набір пар ключа: значення. У різних мовах це реалізовано як запис, структура, словник, хеш-таблиця, список із ключем або асоціативний масив. Ключом може бути лише рядок (реєстрозалежність не регулюється стандартом, це залишається на розсуд програмного забезпечення. Як правило, регістр враховується програмами - імена з літерами в різних регістрах вважаються різними, наприклад , значенням - будь-яка форма. Імена ключів, що повторюються, допустимі, але не рекомендуються стандартом; обробка таких ситуацій відбувається на розсуд програмного забезпечення, можливі варіанти —

враховувати лише перший такий ключ, враховувати останній такий ключ, генерувати помилку.

Впорядкований набір значень. Багато мовах це реалізовано як масив, вектор, список чи послідовність.

Структури даних, що використовуються JSON, підтримуються будь-якою сучасною мовою програмування, що дозволяє застосовувати JSON для обміну даними між різними мовами програмування і програмними системами.

Як значення в JSON можуть бути використані:

1. запис — це невпорядковане безліч пар ключ: значення, укладене у фігурні дужки «{ }». Ключ описується рядком, між ним та значенням стоїть символ «:». Пари ключ-значення відокремлюються один від одного комами.

2. масив (одномірний) - це впорядковане безліч значень. Масив полягає у квадратних дужках «[ ]». Значення поділяються комами. Масив може бути порожнім, тобто не містити жодного значення. Значення не більше одного масиву можуть мати різний тип.

3. число (ціле чи речове).

4. літерали true (логічне значення «істина»), false (логічне значення «брехня») та null.

5. рядок - це впорядковане безліч з нуля або більше символів юнікод, укладене в подвійні лапки.

Перевагою цього формату є те що це структуровані данні які можна використовувати наприклад для наповнення бази даних. Приклад файлу:

```
{
      "item_title": "Создание web-приложений в Silverlight",
      "item_author": "Буньон Лоран",
      "item_publishing": "ДМК-Пресс",
      "item_new_price": 1018,
      "item_old_price": "Немає старого прайса",
      "item_sale": "Немає знижки",
      "item_status": "На складе"
  },
  {
```

```
        "item_title": "C++ для начинающих",
        "item_author": "Шилдт Герберт",
        "item_publishing": "Эком:Шаг за шагом",
        "item_new_price": 1228,
        "item_old_price": "Немає старого прайса",
        "item_sale": "Немає знижки",
        "item_status": "На складе"
    },
```


Розширення CSV – простий текстовий формат, у якому дані розділені комами і призначено представлення табличних даних. Кожен рядок у файлі CSV відповідає рядку у таблиці. На одній лінії поля розділяються комами, кожне поле належить одному стовпцю таблиці. Розділювачем може бути інший символ, наприклад, точка з комою тощо. Значення, що містять зарезервовані символи (подвійна лапка, кома, крапка з комою, новий рядок) обрамляються подвійними лапками («), іноді подвійними лапками обрамляють і текстові значення. Рядки поділяються парою символів CR LF (0x0D 0x0A), але може бути просто LF (0x0A). роздільники стовпців і рядків можуть бути різними, так само може бути різним кодування текстового файлу та обрамлення подвійними лапками, то все це ускладнює перенесення даних з одних програм до інших, незважаючи на всю простоту реалізації підтримки CSV.

CSV – це формат файлу, який підтримується багатьма програмами .CSV файли часто використовуються для імпорту\експорту табличних даних між двома різними комп'ютерними програмами, наприклад, між базою даних та електронною таблицею.

Висновки до розділу 2

В другому розділі розглянуто основні алгоритми та особливості веб-скрапінгу веб-сайтів, проблеми які виникають. Визначено роль програми-скрапера. Створено алгоритм синтаксичного розбору на основі методу

рекурсивного спуску - алгоритм низхідного синтаксичного аналізу дерева веб-сторінок та запропоновано метод його прискорення.

Описано всі кроки алгоритму програми та розглянуто проблеми які виникають при аналізи веб сторінок і їх рішення.

Отже, створено алгоритм для створення програми веб-скрапера та подальшого збереження даних. Вирішені проблеми з швидкодією і обхід проблем при завантажені сторінки які можуть виникнути.

Розділ 3

Проектування інформаційної систему автоматизованого збору даних з веб-ресурсів на основі проведених досліджень

3.1 Основні вимоги для проектування інформаційної технології

Проектування всіх системи починається з визначення мети проекту та у формуванні основних функціональних вимог.

Метаю роботи є створення технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики. Це дозволить ефективно аналізувати інтернет простір, швидко реагувати на зміни трендів і втілювати задумане в життя:

- функціональну модель, що відповідає узагальненим вимогам;
- модель асинхронного використання алгоритму;
- компоненти для обходу перешкод для завантаження сторінки.

3.2 Модулів інформаційної системи та схема їх взаємодії

Один і з основних модулів є модулів для зчитування інформації зі сторінки.

lxml – це модуль , яка дозволяє обробляти XML та HTML файли. Існує безліч готових парсерів XML/HTML, але для отримання кращих результатів або за певних завдань розробники змушені писати свої власні парсери. Це саме та ситуація, коли виникає потреба в модулі lxml. Ключові переваги цього модулю полягають у тому, що він простий у використанні, надзвичайно швидкий при аналізі великих документів забезпечує просте перетворення вихідних даних у типи даних Python, що спрощує маніпулювання файлами.

1. Робота з атрибутами

Тепер подивимося, як пов'язати атрибути з існуючими елементами, а також як вибирати значення певного атрибута для даного елемента.

```
root.set('newAttribute', 'attributeValue')
print(et.tostring(root, pretty_print=True).decode("utf-8"))
```

Результат:

```
<html version="5.0" newAttribute="attributeValue">
  <head/>
  <title bgcolor="red" fontsize="22"/>
  <body fontsize="15"/>
</html>
```

Тут бачимо, що newAttribute = «attributeValue» справді було додано до кореневому елементу.

Отримати значення атрибутів, які встановили в наведеному вище коді. Можемо отримати доступ до дочірнього елемента, використовуючи індексування масиву кореневого елемента, а потім використовуємо метод get() для отримання атрибута:

```
print(root.get('newAttribute'))
print(root[1].get('alpha')) # root[1] accesses the `title` element
print(root[1].get('bgcolor'))
```

Результат:

```
attributeValue
None
Red
```

2. Вилучення тексту з елементів

```
root = et.Element('html', version="5.0")
et.SubElement(root, 'head')
et.SubElement(root, 'title', bgcolor="red", fontsize="22")
et.SubElement(root, 'body', fontsize="15")
# Додати текст до Elements та SubElements
root.text = "This is an HTML file"
root[0].text = "Тим є head of that file"
root[1].text = "This is the title of that file"
root .
print(et.tostring(root, pretty_print=True).decode("utf-8"))
```

Результат:

<html version="5.0">This is an HTML file<head>This is the head of that file</head><title bgcolor="red" fontsize="22">This is the title of that file</title ><body fontsize="15">Тим є те, що файл і буде розміщено параграфи etc</body></html>

3. Перевірка, чи є елемент дочірні елементи

Далі, є дві дуже важливі речі, які повинні бути в змозі перевірити. По-перше, потрібно перевірити, чи є елемент дочірні елементи, а по-друге, чи є поточний вузол елементом.

```
if len(root) > 0:
    print("True")
else:
    print("False")
```

Наведений вище код виведе True, оскільки кореневий вузл має дочірні вузли. Однак, якщо ми перевіримо те саме для дочірніх вузла кореня, як у коді нижче, результат буде False.

```
for i in range(len(root)):
    if (len(root[i]) > 0):
        print("True")
    else:
        print("False")
```

Результат:

```
False
False
False
```

Тепер зробимо те ж саме, щоб побачити, чи є кожен із вузлів Element чи ні:

```
for i in range(len(root)):
    print(et.iselement(root[i]))
```

Результат:

```
True
True
True
```

Метод iselement корисний для визначення, чи є дійсний об'єкт Element, і, таким чином, чи ви можете продовжити його обхід, використовуючи методи.

4. Перевірка, чи має елемент батько

Щоб пройти ієрархією вгору, тобто перевірити та отримати батьківський елемент для дочірнього вузла.

```
print(root.getparent())
print(root[0].getparent())
print(root[1].getparent())
```

Перший рядок не повинен нічого повертати (тобто повернути None), оскільки сам кореневий вузол немає батька. Два інших рядки повинні повернути кореневий елемент, тобто HTML-тег. Давайте перевіримо висновок, щоб переконатися, що це те, що ми очікуємо:

Результат:

```
None
<Element html at 0x1103c9688>
<Element html at 0x1103c9688>
```

5. Вилучення елементів сусідніх елементів (sibling)

Використовується переміщення вбік ієрархією, яка витягує споріднені елементи елемента в дереві.

Обхід дерева в бік дуже подібний до навігації по вертикалі. Раніше використовували getparent, тепер ми будемо використовувати функції getnext і getprevious.

```
# root[1] має бути тег `title`
print(root[1].getnext()) # Тег після `title`
print(root[1].getprevious()) # Тег до `title`
```

Результат:

```
<Element body at 0x10b5a75c8>
<Element head at 0x10b5a76c8>
```

Тут можна бачити, що root[1].getnext() витягує тег "body", оскільки це був

наступний елемент, а root[1].getprevious() витягує тег "head".

Так само, якби використовували функцію getprevious в root, вона мала б повернула None, і якби ми використовували функцію getnext в root[2]], вона також мала б повернути None.

6. Веб скрапінг XML з рядка

HTML потрібно проаналізувати на необроблений рядок, щоб отримати необхідну інформацію або маніпулювати нею.

```
root = et.XML('<html version="5.0">This is an HTML file<head>This is the head of that file</head><title bgcolor="red" fontsize="22">This is the title of that file</title><body fontsize="15">This is the body of that file and would contain paragraphs etc</body></html>')
root[1].text = "The title text has changed!"
print(et.tostring(root, xml_declaration=True).decode('utf-8'))
```

Результат:

```
<?xml version='1.0' encoding='ASCII'?>
<html version="5.0">Тим є файл HTML<head>Цим є head of that file</head><title bgcolor="red" fontsize="22">The title text has changed!</title> <body fontsize="15">Тим є те, що файл і буде розміщено параграфи etc</body></html>
```

7. Пошук елементів

Пошуку елементів та отримання його текстового вмісту.

```
print(root.find('a')) # Не існує тегів <a> тому буде `None`
print(root.find('head').tag)
print(root.findtext('title')) # Отримання тексту з тега title
```

Результат:

```
None
head
This is the title of that file
```

Модуль для асинхронного виконання скрипта Async. Проблема, яку намагається вирішити асинхронністю - це блокування введення-виводу.

За умовчанням, коли а програма звертається до даних із джерела введення-виводу, вона очікує завершення цієї операції, перш ніж продовжити виконання програми. У багатьох випадках затримка, спричинена блокуванням,

незначна. Однак блокування введення/виводу дуже погано масштабується. Якщо вам потрібно дочекатися читання файлу 1010 або транзакцій мережі, продуктивність помітно знизиться. В нашому випадку це дозволить покращити алгоритм.

Примірники класу Future, які повертаються методом Executor.submit(), концептуально дуже близькі до співпрограм, що використовуються в асинхронному програмуванні. Ось чому ми можемо використовувати виконавців, щоб зробити гібрид між спільною багатозадачністю та багатопроцесорністю чи багатопоточністю.

Ядром цього обхідного шляху є метод BaseEventLoop.run_in_executor (executor, func, *args) класу циклу подій. Це дозволяє планувати виконання функції func у процесі або пулі потоків, представленому аргументом executor. Найважливішим у цьому методі є те, що він повертає новий очікуваний об'єкт (об'єкт, на який можна очікувати за допомогою оператора await). Таким чином, завдяки цьому ви можете виконати блокуючу функцію, яка не є співпрограмою точно як співпрограма, і вона не буде блокувати, незалежно від того, скільки часу потрібно, щоб закінчити. Він зупинить лише функцію, яка очікує на результати від такого виклику, але весь цикл подій буде продовжуватися. Результат роботи модуля(рис 3.1):

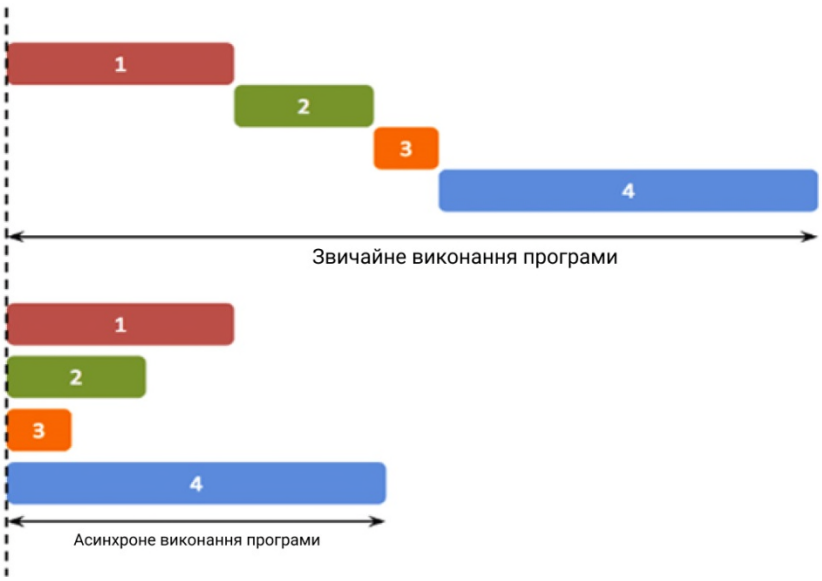


Рисунок 3.1 Результат роботи модуля

Модуль CSV реалізує класи для читання та запису табульованих даних у

форматі CSV. Він дозволяє програмістам говорити, «запиши дані у форматі, що віддається перевагу Excel», або «прочитай дані з файлу, який був створений Excel», не знаючи точних деталей формату CSV, що використовується в Excel.

Монжа описати формати CSV, зрозумілі іншим програмам або визначити власні спеціальні формати CSV. Объекты reader и writer модуля csv читают и пишут последовательности

csv.reader

Повертає об'єкт reader, який ітеруватиметься по рядках у наданому csvfile. csvfile може бути будь-яким об'єктом, який підтримує протокол ітератор і повертає рядок щоразу, коли викликається метод __next__() файлового об'єкта чи об'єкта списку. Якщо csvfile є файловим об'єктом, його потрібно відкрити з параметром newline. Додатковий параметр dialect використовується для визначення ряду параметрів, характерних для специфічного CSV діалекту. Це може бути сутність підкласу класу Dialect або одного з рядків, що повертається функцією list_dialects(). Також можуть передаватися інші додаткові ключові аргументи fmtparams для визначення окремих параметрів форматування в поточному діалекті. Докладніші відомості про параметри діалекту та форматування див. у розділі Діалекти та параметри форматування.

Кожен рядок, зчитаний із файлу csv, повертається у вигляді списку рядків. Автоматичне перетворення типів даних не виконується, якщо не вказано параметр формату QUOTE_NONNUMERIC (у цьому випадку поля без лапок перетворюються на числа з плаваючою точкою).

Короткий приклад використання:

```
with open('eggs.csv', newline='') as csvfile:
... spamreader = csv.reader(csvfile, delimiter='', quotechar='|')
... for row in spamreader:
... print(', '.join(row))
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
csv.writer(csvfile, dialect='excel', **fmtparams)
```

Повертає об'єкт writer, який відповідає за перетворення даних

користувача з окремими рядками на наданий файлоподібний об'єкт. csvfile може бути будь-яким об'єктом з методом write(). Якщо csvfile є файловим об'єктом, його слід відкрити за допомогою newline. Додатковий параметр dialect використовується для визначення низки параметрів, притаманних специфічного CSV діалекту. Це може бути сутність підкласу класу Dialect або одного з рядків, що повертається функцією list_dialects(). Також можуть передаватися інші додаткові ключові аргументи fmtparams для визначення окремих параметрів форматування в поточному діалекті. Докладніші відомості про параметри діалекту та форматування див. у розділі Діалекти та параметри форматування. Для максимально простої взаємодії з модулями, що реалізують DB API, значення None записується як порожній рядок. У той час як це незворотне перетворення, але це допомагає зробити простіше SQL дамп з NULL даними в CSV файли без попередньої обробки даних, що повертаються викликом cursor.fetch*. Всі інші нерядкові дані перед записом стрингифікуються функцією str().

Короткий приклад використання:

```
with open('eggs.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter='',
                            quotechar='|', quoting=csv.QUOTE_MINIMAL)
    spamwriter.writerow(['Spam'] * 5 + ['Baked Beans'])
    spamwriter.writerow(['Spam', 'Lovely Spam', 'Wonderful Spam'])
```

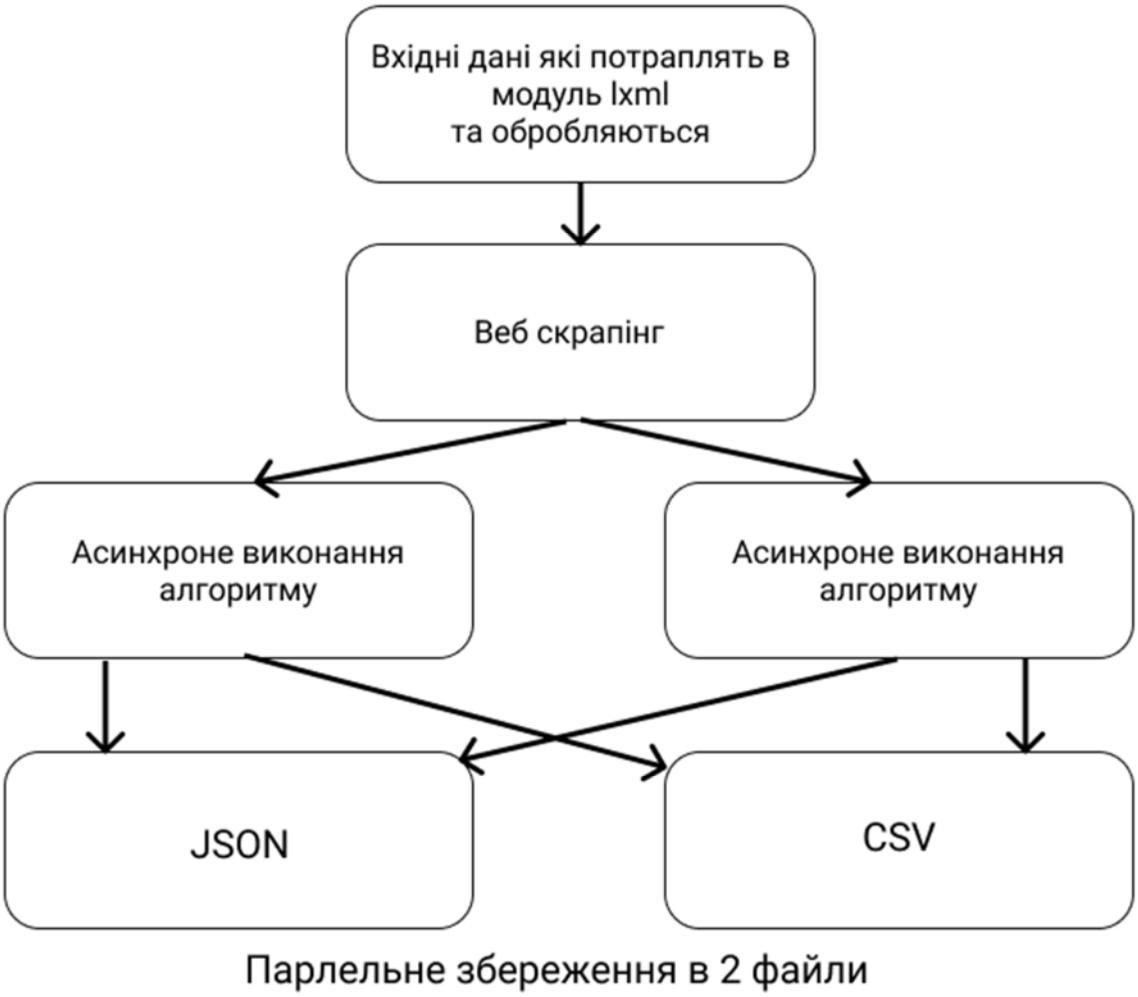


Рисунок 3.2 – Схема взаємодії модулів

3.3 Функціональна модель, що відповідає узагальненим вимогам

Діаграма станів – це діаграма з теорії автоматів зі стандартизованими умовними позначеннями, що може визначати безліч систем від комп'ютерних програм до бізнес-процесів. Використовуються умовні позначення. Створимо діаграму станів веб скрапера (рис. 3.3 – Діаграма станів).

Рисунок 3.3 – Діаграма станів

Також створимо діаграму яка продемонструє функціональної модель (рис. 3.4), що покаже структуру і функції системи, а також потоки даних.

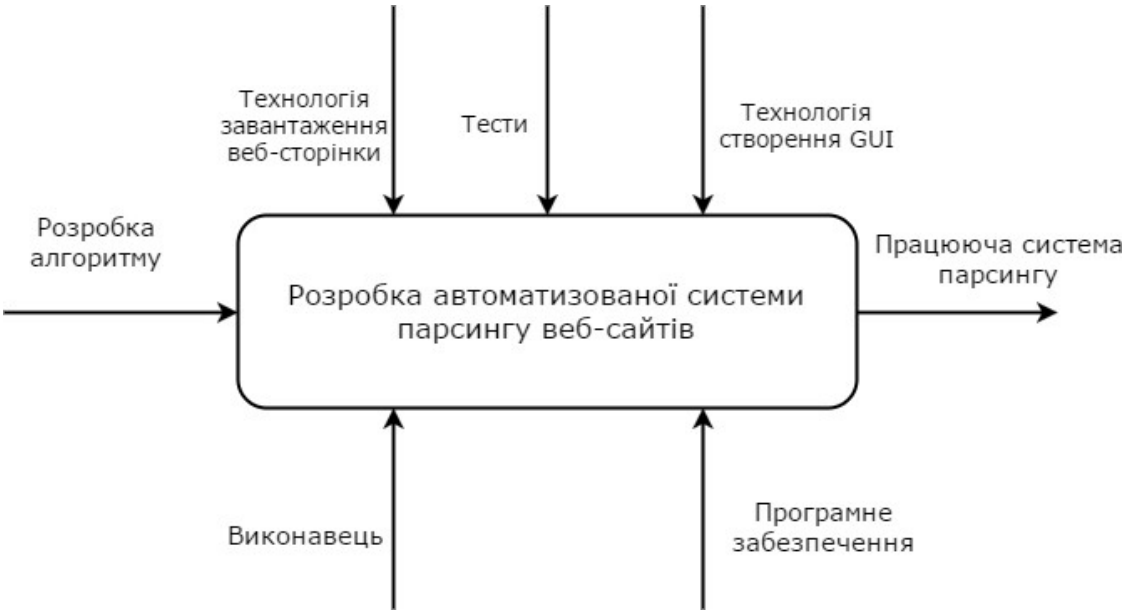


Рисунок 3.4 – Функціональної модель

3.4 Потоки даних

Діаграма потоків — один з основних інструментів структурного аналізу та проектування інформаційних систем, що існували до поширення UML. Незважаючи на наявне в сучасних умовах зміщення акцентів від структурного до об'єктно-орієнтованого підходу до аналізу та проектування систем, «старовинні» структурні нотації, як і раніше, широко і ефективно використовуються як в бізнес-аналізі, так і в аналізі інформаційних систем. Спроектуємо DFD діаграму для веб скрапера (рис. 3.5).

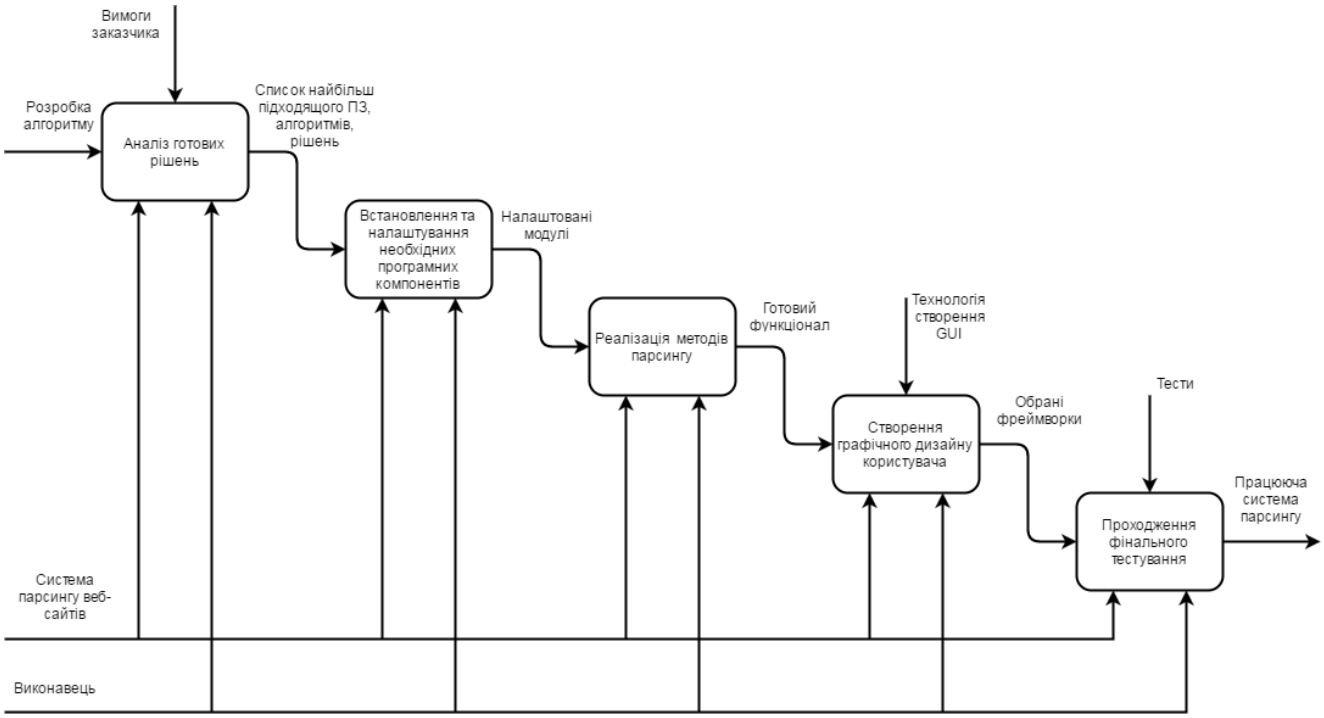


Рисунок 3.5 – DFD діаграма

Асинхронність у програмуванні — виконання процесу в режимі не блокування системного виклику, що дозволяє потоку програми продовжити обробку. Це дозволить покращити стандартний алгоритм і збільшити його швидкість. (рис. 3.6).

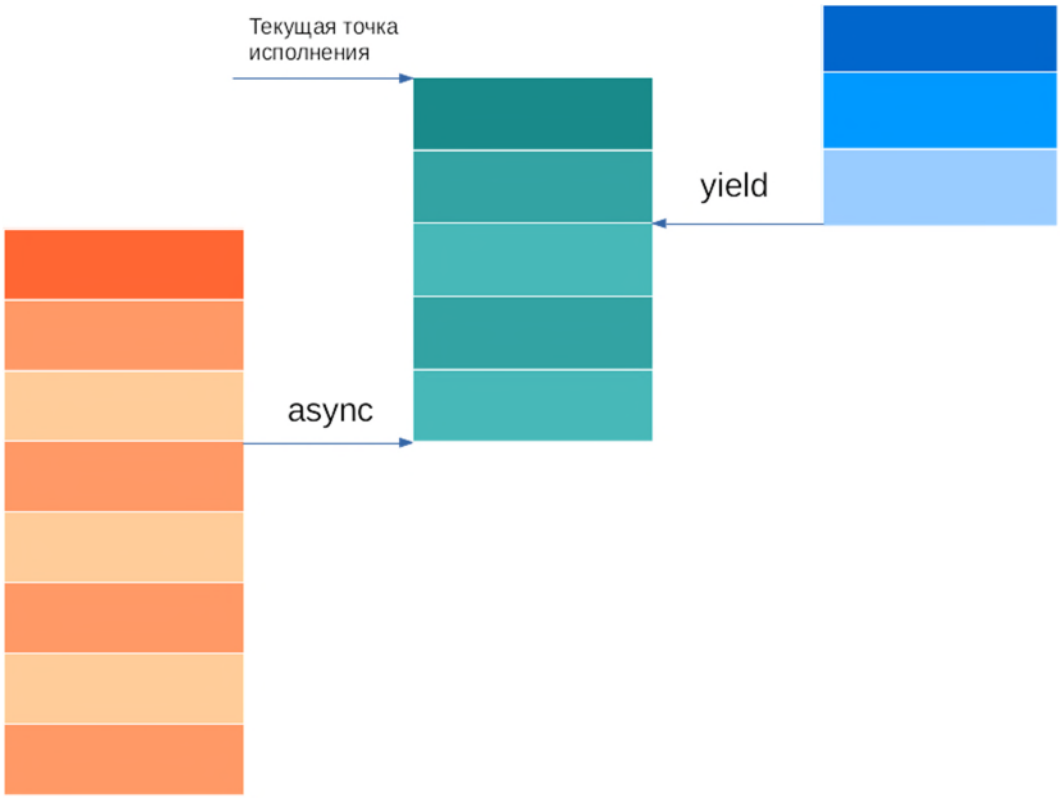


Рисунок 3.6 – Асинхронний підхід виконання алгоритму

3.5 Компоненти для обходу перешкод для завантаження сторінки

Проксі — сервер (комп'ютерна система або програма) в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі (через посередництво проксі-сервера) запити до мережевих сервісів. Спочатку клієнт з'єднується з проксі-сервером і запитує який-небудь ресурс (наприклад, e-mail), розташований на іншому сервері. Потім проксі-сервер або підключається до вказаного сервера та отримує ресурс у нього, або повертає ресурс з власного кешу (у випадках, якщо проксі має свій кеш). У деяких випадках запит клієнта або відповідь сервера може бути змінена проксі-сервером з певною метою. Також проксі-сервер дозволяє захищати клієнтський комп'ютер від деяких мережевих атак і допомагає зберігати анонімність клієнта (рис. 3.7).

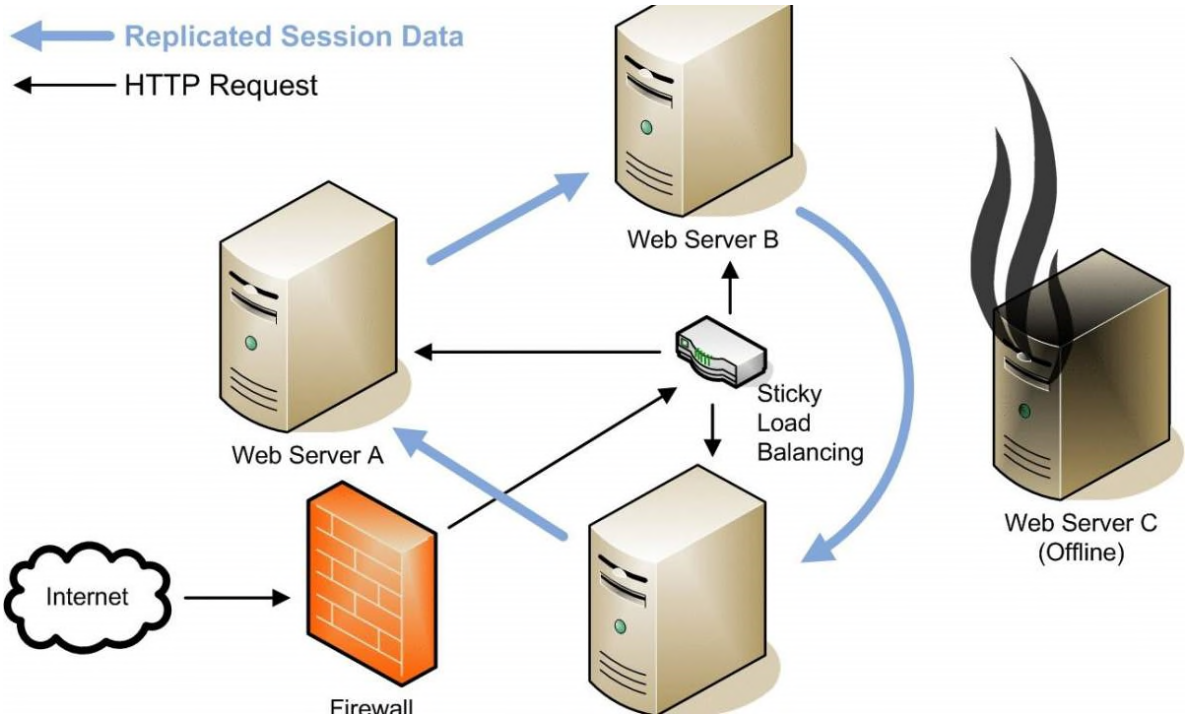


Рисунок 3.7 – Використання проксі сервера

Переваги використання проксі-серверів. Цілком позитивні якості технології розкриваються, якщо говорити про професійні рішення, якими активно користуються в корпоративному секторі. До них можна віднести такі плюси:

– proxy-підключення підтримується усіма поширеними браузерами;

– легко реалізується можливість обліку, контролю та фільтрації трафіку;

– дозволяє на безапаратному рівні локалізувати корпоративну мережу, убезпечивши її від мережевих загроз.

Сьогодні на ринку з'являються спеціальні мережеві маршрутизатори, які за своїми функціями багато в чому копіюють можливості проксі-серверів. Але посередницька технологія передачі та прийому даних продовжує користуватися популярністю. Це пов'язано з тим, що вона набагато безпечніша і простіша у впровадженні.

Види проксі-підключення.

– Прозора схема – вид підключення, що передбачає перенаправлення всього або частини трафіку на проксі. Користувач має можливість використовувати весь обсяг ресурсів сервера, але при цьому він обмежується у виборі способу підключення.

– Зворотна схема – це складніша технологія, яка використовується для рівномірного розподілу навантаження між серверами. При необхідності такий вид підключення може використовуватися як безпечний міжмережевий екран.

Проксі-сервера можна розділити не тільки за способами підключення, але й за технологією шифрування трафіку:

– HTTP – найпопулярніший протокол, оскільки він сумісний з усім стандартним обладнанням та програмним забезпеченням, що робить його універсальним;

– CGI – технологія, розроблена для реалізації анонімного підключення до Інтернету. Протокол підтримується лише браузерами;

– Socks – спеціальний протокол професійного використання. Зазвичай не працює зі стандартними програмами, тому потребує особливого софту;

– FTP – застосовується в корпоративних цілях та веб-розробці. Відрізняється високим ступенем захисту від зовнішніх мережевих атак.

Тонкощі використання безкоштовних проксі.

Головна перевага відкритих серверів є очевидною: немає необхідності витрачати гроші для їх використання. Але варто задуматися про наступні недоліки безкоштовних proxy:

– практично вони працюють надто повільно з великою частотою збоїв;
– послуги часто закриваються чи переходять на платний формат;
– анонімність зазвичай не забезпечується або гарантується;
– деякі сервери використовуються для атак на пристрої користувачів.

Selenium WebDriver — інструмент для автоматизації дій веб-браузера. У більшості випадків використовується для тестування Web-додатків, але цим не обмежується. Зокрема, він може бути використаний для вирішення рутинних завдань адміністрування сайту або регулярного отримання даних із різних джерел (сайтів) (рис. 3.8).

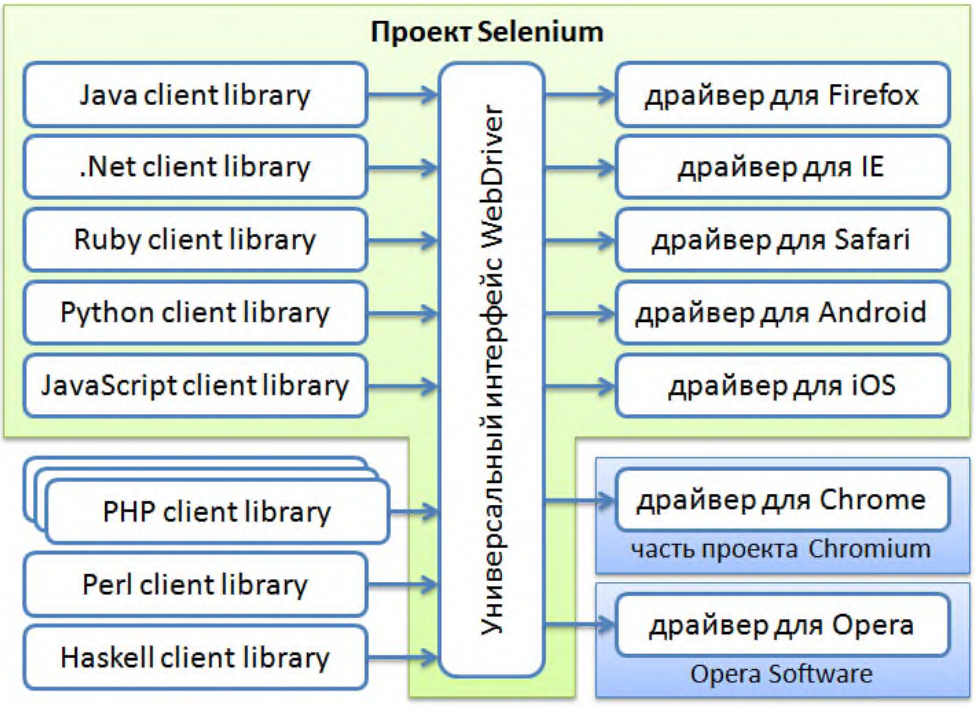


Рисунок 3.8 – Selenium

За своєю сутністю Selenium WebDriver є:

– специфікацію програмного інтерфейсу для керування браузером,
– референсні реалізації цього інтерфейсу для кількох браузерів,
– набір клієнтських бібліотек для цього інтерфейсу кількома мовами програмування.

Драйвер це програма, точніше, програмна бібліотека, яка дозволяє іншим програмам взаємодіяти з деяким пристроєм. Драйвер принтера дозволяє друкувати щось на принтері. Драйвер диска дозволяє читати та писати дані. Драйвер мережної карти дозволяє обмінюватись даними з іншими комп'ютерами через мережу.

Із драйвером користувачі не працюють безпосередньо. Вони працюють з прикладними програмами, які за допомогою драйверів взаємодіють з тими чи іншими пристроями. Драйвер не має інтерфейсу користувача. Стривайте, але іноді буває інтерфейс для налаштування драйвера? Буває. Але це інтерфейс програми для настроювання драйвера, а не самого драйвера. Драйвер має тільки програмний інтерфейс, його призначення полягає в тому, щоб дати можливість прикладним програмам користувача взаємодіяти з пристроєм.

Так ось, Selenium WebDriver, або просто WebDriver - це драйвер браузера, тобто програмна бібліотека, що не має інтерфейсу користувача, яка дозволяє різним іншим програмам взаємодіяти з браузером, керувати його поведінкою, отримувати від браузера якісь дані і змушувати браузер виконувати якісь команди.

Виходячи з цього визначення, ясно, що WebDriver не має прямого відношення до тестування. Він лише надає автотестам доступ до браузера. У цьому його функції закінчуються. І за допомогою такої поведінки цього інструменту можна скрапити сторінку з активним джава скріптовим кодом.

Втім, у рамках проекту Selenium розробляється не тільки драйвер, але ще кілька супутніх продуктів – Selenium Server дозволяє організувати віддалений запуск браузера, за допомогою Selenium Grid можна побудувати кластер із Selenium-серверів. Вони встають в один ряд з перерахованими вище інструментами і фреймворками, тому що також беруть участь у побудові системи запуску тестів. Крім того, є «рекордер», який називається Selenium IDE, він вміє записувати дії користувача та генерувати код, у якому використовується інтерфейс WebDriver для виконання записаних дій.

Але головним у проекті Selenium є WebDriver(рис. 3.9), це ключовий елемент екосистеми Selenium.

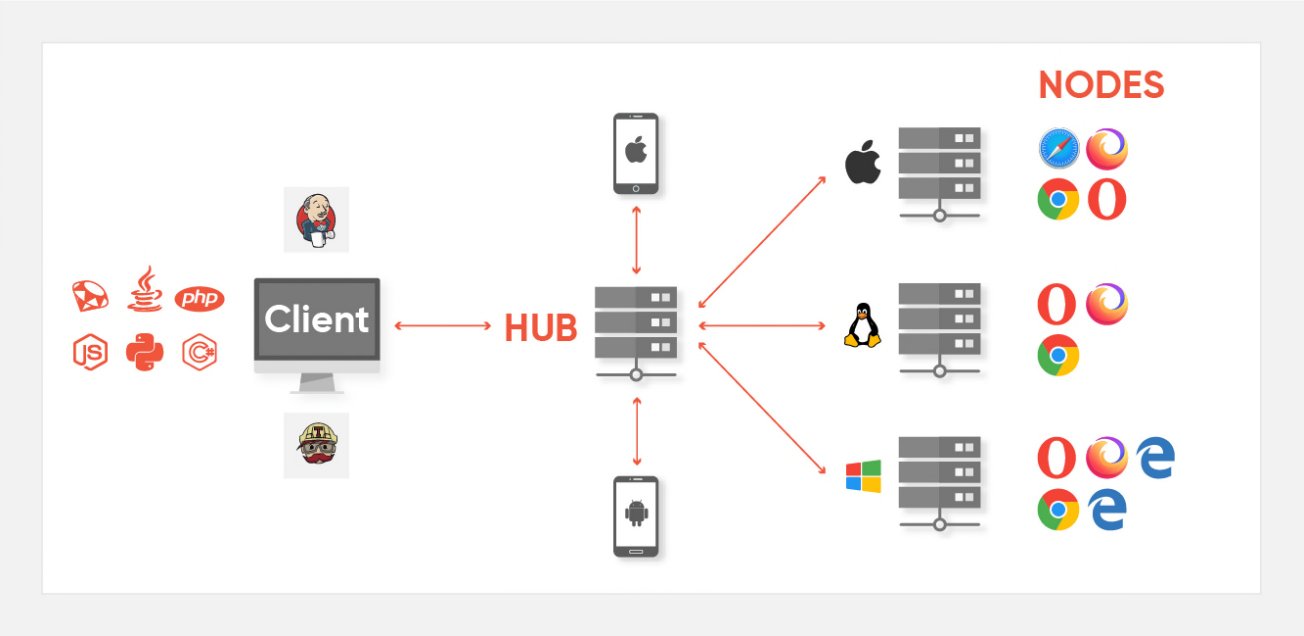


Рисунок 3.9 – WebDriver

Висновки до розділу 3

В результаті виконання розділу розглянуто основні модулі які допоможуть в виконані поставлених цілей та спроектовано інформаційну систему отримання інформації з веб сайтів для бізнес аналітики.

Сформульовано вимоги, які необхідно врахувати при проектуванні інформаційної системи. Запропоновано функціональну модель для забезпечення цих вимог в рамках інформаційної системи.

Розділ 4

Переваги технології над іншими та її програмна реалізація

4.1 Обґрунтування вибору мови програмування для розробки інформаційної системи

Python – високорівнева мова програмування загального призначення з динамічною строгою типізацією та автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду та його якості, а також на забезпечення переносимості написаних на ньому програм. Мова є повністю об'єктно-орієнтованою - все є об'єктами . Незвичайною особливістю мови є виділення блоків коду пробільними відступами . Синтаксис ядра мови мінімалістичний, рахунок чого практично рідко виникає необхідність звертатися до документации. Сама ж мова відома як інтерпретована і використовується в тому числі для написання скриптів .

Python підтримує кілька парадигм програмування, включаючи структурне, об'єктно-орієнтоване, функціональне. Основні архітектурні риси – динамічна типізація, автоматичне управління пам'яттю, повна інтроспекція, механізм обробки винятків, підтримка багатопоточних обчислень та зручні високорівневі структури даних. Код Python організується в функції та класи, які можуть об'єднуватися в модулі (вони можуть бути об'єднані в пакети).

Python, як і будь-яка інша мова програмування, має свої особливості. Отже, можна назвати такі:

1. Кросплатформність.

Python – це мова, що інтерпретується, її інтерпретатори існують для багатьох платформ. Тому із запуском його на будь-якій ОС не повинно виникнути проблем.

З Python є величезна кількість сервісів, середовищ розробки, і фреймворків. Легко можна знайти потрібний продукт для роботи.

Можливість підключити бібліотеки, написані на C. Це дозволяє підвищити ефективність, покращити швидкодію.

Наявність найрізноманітніших джерел інформації про Python. Не важко знайти відповідь на будь-яке питання, так існує багато безкоштовної літератури, навчальних відео-посібників, готових вихідників і шаблонів для роботи у відкритому доступі.

2. Переваги Python щодо інших мов

Python легко конкурує з іншими мовами програмування, оскільки має безліч переваг. По-перше, це зрозуміла і проста програма. Особливо добре для новачків. Можна створити цікаві програми і при цьому не доведеться сидіти тижнями, вивчаючи складний синтаксис.

Динамічна типізація – це одна з головних переваг мови Python. Для новачків це можливість спростити написання коду та уникнути безлічі фатальних помилок та багів у роботі. Також у Python немає операторних дужок, з розставлянням яких часто виникають складнощі.

Еталонною реалізацією Python є інтерпретатор CPython, який підтримує більшість платформ, що активно використовуються. Він розповсюджується під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких додатках.

Python ідеально підходить для веб скрапінгу так я маєбагато бібліотек які прискорюють розробку.

4.2 Метод завантаження сторінки

Для сканування сторінок потрібно їх коректно завантажити. Є досить прости спосіб це зробити за допомогою Python використовуючи urllib2 модуль

Після отримання URL, то функція буде завантажувати веб-сторінки і повертати HTML. Проблема виникає при завантаженні веб-сторінки, ми можемо зіткнутися з помилками, які знаходяться поза нашим контролем. Наприклад, сторінка повертає 404 помилку. У цих випадках urllib2 згенерує заперечення і завершить виконання скрипта. Щоб цього уникнути покращим модуль(рис.4.1):

```
for bi in items_items:
    item_data = bi.find_all("td")

    try:
        item_title = item_data[0].find("a").text.strip()
    except:
        item_title = "Немає назви книги"
```

Рисунок 4.1 – Приклад відлову помилок на прикладі книжкового магазину

Тепер, коли зустрінеться помилка завантаження, виняток перехоплюється і функція повертає в файл що наприклад такої назви не існує.

Часто помилки, які можуть виникнути під час веб скрапінгу можуть носити тимчасовий характер такі як перевантаження серверу. В цьому випадку потрібно повторити завантаження з сервера так як ця проблема не постійна. Винятком слугує помилка 404 і їй подібні так як це означає що сторінки просто не існує і не потрібно її перезавантажувати.

```
def download(url, num_retries=2):
print 'Downloading:', url
try:
html = urllib2.urlopen(url).read()
except urllib2.URLError as e:
print 'Download error:', e.reason
html = None
if num_retries > 0:
if hasattr(e, 'code') and 500 <= e.code < 600:
# recursively retry 5xx HTTP errors
return download(url, num_retries-1)
return html
```

Після зустрічі помилки з кодом 5xx, завантаження повторяється рекурсивно. Це дозволить повторно завантажувати сторінки.

4.3 Налаштування проксі сервера

Для вирішення проблеми з блокуванням був налаштований просі сервер. Реалізувати підтримку проксі стало можливо завдяки urllib2 що дозволило легко використовувати проксі в програмі.

```
def download(url, user_agent='wswp', proxy=None, num_retries=2):
print 'Downloading:', url
headers = {'User-agent': user_agent}
request = urllib2.Request(url, headers=headers)
opener = urllib2.build_opener()
if proxy:
proxy_params = {urlparse.urlparse(url).scheme: proxy}
opener.add_handler(urllib2.ProxyHandler(proxy_params))
try:
html = opener.open(request).read()
if num_retries > 0:
if hasattr(e, 'code') and 500 <= e.code < 600:
html = download(url, user_agent, proxy, num_retries-1)
return html
```

4.4 Веб скрапінг сторінки з динамічним контентом за допомогою селеніума

Selenium WebDriver або просто WebDriver - це драйвер браузера, тобто програмна бібліотека, що не має користувальницького інтерфейсу, яка дозволяє різним іншим програмам взаємодіяти з браузером, керувати його поведінкою, отримувати від браузера якісь дані і змушувати браузер виконувати якісь команди.

Робота Selenium, полягає в створенні підключення до браузеру:

```
from selenium import webdriver from selenium.webdriver.common.by
import By from selenium.webdriver.support.select
import Select from selenium.webdriver.support.ui
import WebDriverWait from selenium.webdriver.support
      import expected_conditions as EC driver = webdriver.Firefox()
driver.get("your url here")
```

```
        # waiting for the page to load wait = WebDriverWait(driver, 10)
wait.until(EC.visibility_of_element_located((By.ID, "content"))) data =
driver.page_source driver.close() soup = BeautifulSoup(data, "my.parser)
```

Після встановлення зв'язку з браузером потрібно встановити який саме елемент потрібно обрати. Selenium також підтримує вибір елементів за допомогою селекторів CSS або XPath. Коли визначено потрібне текстове поле, потрібно ввести вміст у send_keys() метод, який імітує введення тексту:

```
driver.find_element_by_id('search_term').send_keys('.')
```

За допомогою Selenium можна з легкістю фільтрувати дані активувати пошту і завантаження сторінки.

4.5 Обхід капчі

Нажаль не існує унікального методу чи алгоритму обходу капчі. Але робочий варіант реалізації за допомого спеціальних сервісів де CAPTCHA зображення, розпізнаєтеся реальними людьми . Для використання потрібно отримати ключ API що дозволить використовувати сервіс в програмі. В цій роботі це метод не використовується так як він є платним .Приклад сервісу(рис. 4.2):

Submit captcha

URL: `https://www.9kw.eu/index.cgi` (POST)

apikey: your API key

action: must be set to "usercaptchaupload"

file-upload-01: the image to solve (either a file, url or string)

base64: set to "1" if the input is Base64 encoded

maxtimeout: the maximum time to wait for a solution
(must be between 60 - 3999 seconds)

selfsolve: set to "1" to solve this CAPTCHA ourself

Return value: ID of this CAPTCHA

Request result of submitted captcha

URL: `https://www.9kw.eu/index.cgi` (GET)

apikey: your API key

action: must be set to "usercaptchacorrectdata"

id: ID of CAPTCHA to check

info: set to "1" to return "NO DATA" when there is not yet a solution
(by default returns nothing)

Return value: Text of the solved CAPTCHA or an error code

Рисунок 4.2 – Приклад сервісу для розпізнавання CAPTCHA

4.5 Реалізація асинхронного виконання алгоритму парсинга та його тестування

Для покращення нашого алгоритму будемо використовувати бібліотеку asyncio. Вона дозволить використовувати асинхронний спосіб виконання алгоритму. Наведемо базові визначення основних понять asyncio:

Coroutine (супрограма) - генератор, який отримує дані, але не генерує їх. У Python 2.5 було введено новий синтаксис, що дозволяє надсилати значення генератору.

Tasks – планувальники для співпрограм. Є цикл, в якому запускається _step, а він у свою чергу вже викликає наступний крок співпрограми.

Реалізація в програмі виглядає ось так:

```
async def get_page_data(session, page):
    headers = {
        "accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
```

```
                "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.106 Safari/537.36"
            }

            url =
f"https://www.labirint.ru/genres/2308/?available=1&paperbooks=1&display=table&page
={page}"

            async with session.get(url=url, headers=headers) as response:
                response_text = await response.text()

                soup = BeautifulSoup(response_text, "lxml")

                items_items = soup.find("tbody", class_="products-
table__body").find_all("tr")

                for bi in items_items:
                    item_data = bi.find_all("td")

                    try:
                        item_title = item_data[0].find("a").text.strip()
                    except:
                        item_title = "Немає назви книги"
        async def gather_data():
            headers = {
                "accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
                "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.106 Safari/537.36"
            }

            url =
"https://www.labirint.ru/genres/2308/?available=1&paperitems=1&display=table"

            async with aiohttp.ClientSession() as session:
                response = await session.get(url=url, headers=headers)
                soup = BeautifulSoup(await response.text(), "lxml")
                pages_count = int(soup.find("div", class_="pagination-
numbers").find_all("a")[-1].text)

                tasks = []
```

```
    for page in range(1, pages_count + 1):
        task = asyncio.create_task(get_page_data(session, page))
        tasks.append(task)

    await asyncio.gather(*tasks)
```

Результат виконання алгоритма парсингу в звичайному режимі(рис.4.3):

```
PS F:\parser> & C:/Users/Serhii/AppData/Local/Programs/Python/Python310/python.exe
Оброблено 1/17
Оброблено 2/17
Оброблено 3/17
Оброблено 4/17
Оброблено 5/17
Оброблено 6/17
Оброблено 7/17
Оброблено 8/17
Оброблено 9/17
Оброблено 10/17
Оброблено 11/17
Оброблено 12/17
Оброблено 13/17
Оброблено 14/17
Оброблено 15/17
Оброблено 16/17
Оброблено 17/17
Затрачено на роботу веб скрапера: 43.057941198349
```

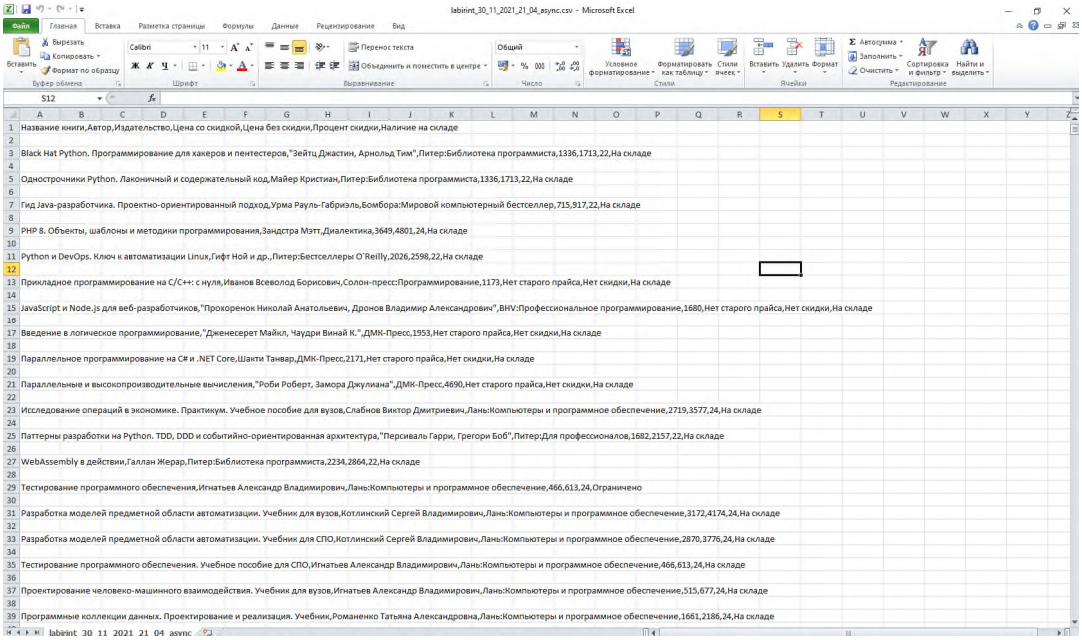
Рисунок 4.3 – Час обробки звичайним алгоритмом

Тепер розглянемо асинхрону його версію (рис.4.4):

```
PS F:\parser> & C:/Users/Serhii/AppData/Local/Programs/Python/Python310/python.exe
[INFO] Обробив сторінку 1
[INFO] Обробив сторінку 15
[INFO] Обробив сторінку 17
[INFO] Обробив сторінку 13
[INFO] Обробив сторінку 16
[INFO] Обробив сторінку 3
[INFO] Обробив сторінку 14
[INFO] Обробив сторінку 2
[INFO] Обробив сторінку 5
[INFO] Обробив сторінку 11
[INFO] Обробив сторінку 4
[INFO] Обробив сторінку 9
[INFO] Обробив сторінку 12
[INFO] Обробив сторінку 7
[INFO] Обробив сторінку 8
[INFO] Обробив сторінку 6
[INFO] Обробив сторінку 10
Затрачено на роботу веб скрапера: 6.227150917053223
```

Рисунок 4.4 – Час обробки звичайним алгоритмом

Як видно з наведених важче результатів вдалась прискорити виконання алгоритму в 7 разів, це дало дуже великий приріст в швидкості, але можна зауважити що навантаження на сервер сайту зросло. Якщо як в нашому випадку сторінок не багато це не є критично. Але якщо їх буде багато в практиці часто запускають парсери на ніч так як навантаження на сервер в нічний час найменше, це дозволить ефективно отримувати дані без шкоди для відвідувачів сайту. Приклад сформованого CSV(рис.4.5) файлу та Json(рис.4.6):



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	Название книги,Автор,Издательство,Цена со скидкой,Цена без скидки,Процент скидки,Наличие на складе																									
2																										
3	Black Hat Python. Программирование для хакеров и пентестеров,"Зейтц Джастин, Арнольд Тим",Питер:Библиотека программиста,1336,1713,22,На складе																									
4																										
5	Однострочники Python. Лаконичный и содержательный код,Майер Кристиан,Питер:Библиотека программиста,1336,1713,22,На складе																									
6																										
7	Гид Java-разработчика. Проектно-ориентированный подход,Урма Рауль-Габриэль,Бомбора:Мировой компьютерный бестселлер,715,917,22,На складе																									
8																										
9	PHP 8. Объекты, шаблоны и методики программирования,Зандстра Мэтт,Диалектика,3649,4801,24,На складе																									
10																										
11	Python и DevOps. Ключ к автоматизации Linux,Гифт Ной и др.,Питер:Бестселлеры O'Reilly,2026,2598,22,На складе																									
12																										
13	Прикладное программирование на C/C++: с нуля,Иванов Всеволод Борисович,Солон-пресс:Программирование,1173,Нет старого прайса,Нет скидки,На складе																									
14																										
15	JavaScript и Node.js для веб-разработчиков,"Прохоренок Николай Анатольевич, Дронов Владимир Александрович",BHV:Профессиональное программирование,1680,Нет старого прайса,Нет скидки,На складе																									
16																										
17	Введение в логическое программирование,"Дженесерет Майкл, Чаудри Винай К.",ДМК-Пресс,1953,Нет старого прайса,Нет скидки,На складе																									
18																										
19	Параллельное программирование на C# и .NET Core,Шакти Танвар,ДМК-Пресс,2171,Нет старого прайса,Нет скидки,На складе																									
20																										
21	Параллельные и высокопроизводительные вычисления,"Роби Роберт, Замора Джулиана",ДМК-Пресс,4690,Нет старого прайса,Нет скидки,На складе																									
22																										
23	Исследование операций в экономике. Практикум. Учебное пособие для вузов,Слабнов Виктор Дмитриевич,Лань:Компьютеры и программное обеспечение,2719,3577,24,На складе																									
24																										
25	Паттерны разработки на Python. TDD, DDD и событийно-ориентированная архитектура,"Персиваль Гарри, Грегори Боб",Питер:Для профессионалов,1682,2157,22,На складе																									
26																										
27	WebAssembly в действии,Галлан Жерар,Питер:Библиотека программиста,2234,2864,22,На складе																									
28																										
29	Тестирование программного обеспечения,Игнатьев Александр Владимирович,Лань:Компьютеры и программное обеспечение,466,613,24,Ограничено																									
30																										
31	Разработка моделей предметной области автоматизации. Учебник для вузов,Котлинский Сергей Владимирович,Лань:Компьютеры и программное обеспечение,3172,4174,24,На складе																									
32																										
33	Разработка моделей предметной области автоматизации. Учебник для СПО,Котлинский Сергей Владимирович,Лань:Компьютеры и программное обеспечение,2870,3776,24,На складе																									
34																										
35	Тестирование программного обеспечения. Учебное пособие для СПО,Игнатьев Александр Владимирович,Лань:Компьютеры и программное обеспечение,466,613,24,На складе																									
36																										
37	Проектирование человеко-машинного взаимодействия. Учебник для вузов,Игнатьев Александр Владимирович,Лань:Компьютеры и программное обеспечение,515,677,24,На складе																									
38																										
39	Программные коллекции данных. Проектирование и реализация. Учебник,Романенко Татьяна Александровна,Лань:Компьютеры и программное обеспечение,1661,2186,24,На складе																									

Рисунок 4.5 – CSV файл для подальшої аналітики


```
[
    {
        "item_title": "Программирование на алгоритмическом языке КуМир",
        "item_author": "Аникеева Людмила Александровна, Гусева Ольга Борисовна",
        "item_publishing": "Солон-пресс:Элективный курс. Профильное обучение",
        "item_new_price": 340,
        "item_old_price": "Немає старого прайса",
        "item_sale": "Немає знижки",
        "item_status": "На складе"
    },
    {
        "item_title": "Создание web-приложений в Silverlight",
        "item_author": "Буньон Лоран",
        "item_publishing": "ДМК-Пресс",
        "item_new_price": 1018,
        "item_old_price": "Немає старого прайса",
        "item_sale": "Немає знижки",
        "item_status": "На складе"
    },
    {
        "item_title": "C++ для начинающих",        "item_author": "Шилдт Герберт",
        "item_publishing": "Эком:Шаг за шагом",
        "item_new_price": 1228,
        "item_old_price": "Немає старого прайса",
        "item_sale": "Немає знижки",
        "item_status": "На складе"
    },
    {
        "item_title": "Цифровая обработка сигналов. Моделирование в Simulink",
        "item_author": "Солонина Алла Ивановна",
        "item_publishing": "BHV:Учебное пособие",
        "item_new_price": 610,
        "item_old_price": "Немає старого прайса",
        "item_sale": "Немає знижки",
        "item_status": "На складе"
    },
    {
        "item_title": "XSLT. Сборник рецептов",
        "item_author": "Мангано Сэл",
        "item_publishing": "BHV",
        "item_new_price": 1110,
        "item_old_price": "Немає старого прайса",
        "item_sale": "Немає знижки",
```


Рисунок 4.6 – Json файл для подальшої аналітики

Висновки до розділу 4

Проаналізовано та обґрунтовано вибір мови програмування для реалізації інформаційної системи. Обґрунтовано вибір програмної технології для розробки інформаційної системи.

Розроблений та реалізований модулі для обходу перешкод завантаження сторінки. Також прискорено стандартний алгоритм методом асинхронного виконання що дала приріст майже у десять разів.

В результаті виконання розділу було зроблено програмну реалізацію інформаційної системи для отримання даних в CSV файл для подальшого аналізу.

Загальні висновки

Запропонована алгоритм та система призначена для прискорення збору інформації з веб ресурсів для бізнес аналітики. Проаналізовано існуючі рішення.

Досліджено існуючі веб скрапери для добутку інформації. Розглянуто, що таке аналізатори і парсери. Детально розглянуто алгоритм добутку інформації з веб сторінки і запропоновано покращення що до його роботи. Також розглянуті проблеми з якими скрапер може стикнутись в ході роботи. Запропоновані рішення та створенні модулі на основі досліджень для виключення помилок в роботі скрапера.

У результаті виконання роботи були поставлені та вирішені наступні завдання:

1. Визначено типи даних, що необхідно отримувати з веб-сторінок для бізнес-аналітики.

2. Проаналізовано існуючі методи отримання даних з веб-ресурсів – web-scraping.

3. Розроблено алгоритм синтаксичного розбору веб-сторінок та експорту даних.

4. Спроектувати інформаційну системи автоматизованого збору даних з веб-ресурсів на основі проведених досліджень.

В результаті роботи були отримані такі інновації та положення наукової новизни:

1. Вдосконалено алгоритм синтаксичного розбору веб-сторінки, що дає можливість отримувати та зберігати необхідні для бізнес-аналітики дані.

2. вдосконалено метод для доступу до веб-ресурсів, що дозволив обійти встановлені перешкоди для автоматизованої обробки даних.

3. Розроблено інформаційну системи автоматизованого отримання, обробки та збереження даних з веб ресурсів для подальшої бізнес-аналітики. Це дозволить скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі.

Запропоновано та описано інформаційну систему веб скрапінгу даних та подальше їх збереження в файл для бізнес аналітики. Спроектовано структуру інформаційної системи, а також її модулі. Використано асинхронну версія алгоритму, яка дала можливість прискорити його майже в десять разів без втрати інформації і великого навантаження.

На основі розробленої інформаційної системи автоматизованого отримання даних з веб-ресурсів було створено програму реалізацію, що дає можливість обійти встановлені перешкоди для автоматизованої обробки даних та здійснити їх отримання та збереження для подальшої бізнес-аналітики.

Перелік посилань

1. Что такое парсинг и как правильно парсить. [Електронний ресурс]. – Режим доступу: https://blog.calltouch.ru/chto-takoe-parsing/
2. Парсинг. Что это и где используется [Електронний ресурс]. – Режим доступу: https://ipipe.ru/info/parsing
3. Парсинг та труднощі з ним. Про що всі мовчать. [Електронний ресурс]. – Режим доступу: https://www.pricecontrol.com.ua/ua/parsing-ta-trudnoshhi-z-nim-pro-shho-vsi-movchat/
4. Веб-скрейпинг. [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BA%D1%80%D0%B5%D0%B9%D0%BF%D0%B8%D0%BD%D0%B3
5. Datacol – парсер для збору інформації з сайтів. [Електронний ресурс]. – Режим доступу: https://vlada-rykova.com/ua/parser-sajtov/
6. 30 бесплатных программ для парсинга сайтов в 2020 году. [Електронний ресурс]. – Режим доступу: https://vc.ru/services/115584-30-besplatnyh-programm-dlya-parsinga-saytov-v-2020-godu
7. Робин Никсон. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5
8. Дэвид Скляр. Изучаем PHP 7. Руководство по созданию интерактивных веб-сайтов

ДОДАТКИ

Додаток А

Програмні коди

```
import json

import time

from bs4 import BeautifulSoup

import datetime

import csv

import asyncio

import aiohttp




items_data = []

start_time = time.time()




async def get_page_data(session, page):

    headers = {

        "accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",

        "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.106 Safari/537.36"

    }



    url =
f"https://www.labirint.ru/genres/2308/?available=1&paperbooks=1&display=table&page={p
age}"
```

```
    async with session.get(url=url, headers=headers) as response:

        response_text = await response.text()


        soup = BeautifulSoup(response_text, "lxml")


        items_items = soup.find("tbody", class_="products-
table__body").find_all("tr")


        for bi in items_items:

            item_data = bi.find_all("td")


            try:

                item_title = item_data[0].find("a").text.strip()

            except:

                item_title = "Немає назви книги"


            try:

                item_author = item_data[1].text.strip()

            except:

                item_author = "Немає автора"


            try:

                item_publishing = item_data[2].find_all("a")

                item_publishing = ":".join([bp.text for bp in item_publishing])

            except:

                item_publishing = "Немає видання"
```

```
            try:
                item_new_price = int(item_data[3].find("div",
class_="price").find("span").find("span").text.strip().replace(" ", ""))
            except:
                item_new_price = "Немає нового прайса"

            try:
                item_old_price = int(item_data[3].find("span", class_="price-
gray").text.strip().replace(" ", ""))
            except:
                item_old_price = "Немає старого прайса"

            try:
                item_sale = round(((item_old_price - item_new_price) /
item_old_price) * 100)
            except:
                item_sale = "Немає знижки"

            try:
                item_status = item_data[-1].text.strip()
            except:
                item_status = "Нема статусу"

            items_data.append(
                {
                    "item_title": item_title,
                    "item_author": item_author,
```

```
                    "item_publishing": item_publishing,
                    "item_new_price": item_new_price,
                    "item_old_price": item_old_price,
                    "item_sale": item_sale,
                    "item_status": item_status
                }
            )

        print(f"[INFO] Обробив сторінку {page}")


async def gather_data():
    headers = {
        "accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
        "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.106 Safari/537.36"
    }

    url =
"https://www.labirint.ru/genres/2308/?available=1&paperitems=1&display=table"

    async with aiohttp.ClientSession() as session:
        response = await session.get(url=url, headers=headers)
        soup = BeautifulSoup(await response.text(), "lxml")
        pages_count = int(soup.find("div", class_="pagination-numbers").find_all("a")[-1].text)
```

```
        tasks = []

        for page in range(1, pages_count + 1):
            task = asyncio.create_task(get_page_data(session, page))
            tasks.append(task)

        await asyncio.gather(*tasks)


def main():
    asyncio.run(gather_data())
    cur_time = datetime.datetime.now().strftime("%d_%m_%Y_%H_%M")

    with open(f"labirint_{cur_time}_async.json", "w") as file:
        json.dump(items_data, file, indent=4, ensure_ascii=False)

    with open(f"labirint_{cur_time}_async.csv", "w") as file:
        writer = csv.writer(file)

        writer.writerow(
            (
                "Назва книги",
                "Автор",
                "Видання",
                "Ціна з знижкою",
```

```
            "Ціна без знижкою",
            "Процент знижки",
            "Наявність на складі"
        )
    )

for item in items_data:

    with open(f"labirint_{cur_time}_async.csv", "a") as file:
        writer = csv.writer(file)

        writer.writerow(
            (
                item["item_title"],
                item["item_author"],
                item["item_publishing"],
                item["item_new_price"],
                item["item_old_price"],
                item["item_sale"],
                item["item_status"]
            )
        )

finish_time = time.time() - start_time
print(f"Затрачено на роботу веб скрапера: {finish_time}")
```

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


import json
import time
import requests
from bs4 import BeautifulSoup
import datetime
import csv


start_time = time.time()


def get_data():
    cur_time = datetime.datetime.now().strftime("%d_%m_%Y_%H_%M")

    with open(f"labirint_{cur_time}.csv", "w") as file:
        writer = csv.writer(file)

        writer.writerow(
            (
                "Назва книги",
                "Автор",
                "Видання",
                "Ціна з знижкою",
                "Ціна без знижкою",
                "Процент знижки",
                "Наявність на складі"
            )
        )

    headers = {
        "accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
```

```
        "user-agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/91.0.4472.106 Safari/537.36"
    }

    url =
"https://www.labirint.ru/genres/2308/?available=1&paperbooks=1&display=table"

    response = requests.get(url=url, headers=headers)
    soup = BeautifulSoup(response.text, "lxml")

    pages_count = int(soup.find("div", class_="pagination-
numbers").find_all("a")[-1].text)

    items_data = []
    for page in range(1, pages_count + 1):
    # for page in range(1, 2):
        url =
f"https://www.labirint.ru/genres/2308/?available=1&paperbooks=1&display=table&pag
e={page}"

        response = requests.get(url=url, headers=headers)
        soup = BeautifulSoup(response.text, "lxml")

        items_items = soup.find("tbody", class_="products-
table__body").find_all("tr")

        for bi in items_items:
            item_data = bi.find_all("td")

            try:
                item_title = item_data[0].find("a").text.strip()
            except:
                item_title = "Немає назви книги"

            try:
                item_author = item_data[1].text.strip()
            except:
                item_author = "Немає автора"
```

```
            try:
                item_publishing = item_data[2].find_all("a")
                item_publishing = ":".join([bp.text for bp in item_publishing])
            except:
                item_publishing = "Немає видання"

            try:
                item_new_price = int(item_data[3].find("div",
class_="price").find("span").find("span").text.strip().replace(" ", ""))
            except:
                item_new_price = "Немає нового прайса"

            try:
                item_old_price = int(item_data[3].find("span", class_="price-
gray").text.strip().replace(" ", ""))
            except:
                item_old_price = "Немає старого прайса"

            try:
                item_sale = round(((item_old_price - item_new_price) /
item_old_price) * 100)
            except:
                item_sale = "Немає знижки"

            try:
                item_status = item_data[-1].text.strip()
            except:
                item_status = "Нема статусу"

            # print(item_title)
            # print(item_author)
            # print(item_publishing)
            # print(item_new_price)
            # print(item_old_price)
            # print(item_sale)
            # print(item_status)
            # print("#" * 10)

            items_data.append(
```

```
                {
                    "item_title": item_title,
                    "item_author": item_author,
                    "item_publishing": item_publishing,
                    "item_new_price": item_new_price,
                    "item_old_price": item_old_price,
                    "item_sale": item_sale,
                    "item_status": item_status
                }
            )

            with open(f"labirint_{cur_time}.csv", "a") as file:
                writer = csv.writer(file)

                writer.writerow(
                    (
                        item_title,
                        item_author,
                        item_publishing,
                        item_new_price,
                        item_old_price,
                        item_sale,
                        item_status
                    )
                )

        print(f"Оброблено {page}/{pages_count}")
        time.sleep(1)

    with open(f"labirint_{cur_time}.json", "w") as file:
        json.dump(items_data, file, indent=4, ensure_ascii=False)


def main():
    get_data()
    finish_time = time.time() - start_time
    print(f"Затрачено на роботу веб скрапера: {finish_time}")


if __name__ == '__main__':
```

```
main()
```

Додаток Б

Ксерокопії наукових публікацій, виконаних при роботі над дипломною роботою магістра

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ
за матеріалами XIII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2021»

15-16 жовтня 2021

Хмельницький 2021

УДК 004:37:001:62

Збірник наукових праць за матеріалами XIII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». Хмельницький – 2021. – 413с.

У збірнику наукових праць подані перспективні практичні розробки аспірантів, студентів та здобувачів в області сучасних інформаційних технологій. Розглянуто актуальні проблеми комп'ютерних наук, комп'ютерної інженерії, прикладної математики й інженерії програмного забезпечення, приведено ряд робіт по впровадженню інформаційних технологій у виробництво та управління. Висвітлено перспективні розробки сучасних систем пошуку, обробки й захисту інформації, медійних та комунікаційних системи.

УДК 004:37:001:62

Матеріали конференції відтворені з авторських оригіналів. При макетуванні можливі незначні зміни компоновки контенту авторських оригіналів.

Участь у конференції та складові всіх її етапів (розгляд праць, макетування, публікація збірника наукових праць та видача сертифікатів) є безкоштовними для всіх учасників. Оргкомітет конференції висловлює подяку учасникам конференції та сподівається на подальшу співпрацю.

З питань проведення конференції та подальшого обміну інформацією звертатись на e-mail конференції: *apkt.khnu@gmail.com*

АКТУАЛЬНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК - 2021

XIII Всеукраїнська науково-практична конференція

Метою конференції є висвітлення актуальних проблем комп'ютерних наук, інформатики та інформаційних технологій.

СЕКЦІЇ КОНФЕРЕНЦІЇ:

1. Комп'ютерні науки та прикладні інформаційні технології.
2. Комп'ютерна інженерія та системи захисту інформації.
3. Математичне моделювання та інженерія програмного забезпечення
4. Телерадіокомунікації, медійні та комунікаційні системи.
5. Проблеми впровадження інформаційних технологій у виробництво та управління.

Робочі мови конференції: українська, англійська

ОРГКОМІТЕТ:

СИНЮК О. М. голова оргкомітету, проректор Хмельницького національного університету з наукової роботи, доктор технічних наук, професор

САВЕНКО О. С. заступник голови оргкомітету, декан факультету Інформаційних технологій ХНУ, доктор технічних наук, професор

БАРМАК О. В. заступник голови оргкомітету, завідувач кафедри Комп'ютерних наук ХНУ, доктор технічних наук, професор

ГОВОРУЩЕНКО Т. О. завідувач кафедри Комп'ютерної інженерії та інформаційних систем ХНУ, доктор технічних наук, професор

ВИСОЦЬКА О. В. доктор технічних наук, завідувач кафедри Радіоелектронних та біомедичних комп'ютеризованих засобів і технологій Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут», професор

ЛАВРОВ Є. А. доктор технічних наук, професор (Сумський державний університет)

ТІМОФЄЄВА Л. В. відповідальна за студентську науково-дослідну роботу ХНУ

МАЗУРЕЦЬ О. В. секретар конференції, к.т.н., доцент кафедри Комп'ютерних наук ХНУ

МОЛЧАНОВА М. О. секретар конференції, викладач кафедри Комп'ютерних наук ХНУ

КОНТАКТНА ІНФОРМАЦІЯ:

e-mail для листування: *apkt.khnu@gmail.com*

УДК 004.4

Данчук С. В., Багрій Р. О.

Хмельницький національний університет

ТЕХНОЛОГІЯ АВТОМАТИЗОВАНОГО ОТРИМАННЯ ДАНИХ З ВЕБ-РЕСУРСІВ ДЛЯ БІЗНЕС-АНАЛІТИКИ

Проаналізовано способи автоматизованого отримання даних з веб-ресурсів та проблеми, які виникають при цьому. Проаналізовано варіанти синтаксичного розбору веб-сторінок для подальшого імпорту та експорту даних з метою аналітики. Запропоновано технологію та описано основні етапи для видобутку даних з веб сторінок. Досліджено інструментарій за допомогою якого можна реалізувати інформаційну систему.

The methods of automated retrieval of data from web resources and the problems that arise are analyzed. The variants of web page parsing are analyzed for further import and export of data for analytical purposes. The technology is proposed and the main stages for data extraction from web pages are described. The tools with which it is possible to implement the information system are investigated.

Сьогодні оновлення інформації відбувається дуже швидко. Обробляти вручну складно і це займає багато часу. Під час пандемії COVID-19 ця проблема набула не аби якої актуальності. Більшість бізнесу перейшло в онлайн. Проблема збору даних для аналітики загострилась. Тому створені спеціальні програми, які в автоматичному режимі аналізують і збирають дані для бізнес аналітики. Вони справляються з величезними обсягами значень, що безперервно оновлюються.

Метою роботи є розробка автоматизованої системи для збору та структурування даних з веб-ресурсів для подальшої аналітики.

Для швидкої обробки інформації застосовується веб-скрейпінг. Це область з активними розробками, що розділяють амбітну ініціативу розвитку взаємодії людини та комп'ютера, яка потребує проривів у обробці та розумінні тексту онлайн-сторінок штучним інтелектом. Сучасні рішення для скрейпінгу варіюються від спеціальних, що вимагають людських зусиль до повністю автоматизованих систем, які здатні перетворювати цілі веб-сайти в структуровану інформацію у певному форматі. Ідеально, коли сайт, дані якого ви хочете отримати, надає їх через API з дозволеним крос-доменним доступом. Якщо справи не йдуть таким чином, можна звернутися до інших методів скрейпінгу.

Розглянемо більш детально кожен з методів скрейпінгу та визначмо оптимальний варіант для збору інформації для аналітики.

1. Копіпаст вручну

Іноді навіть найкраща технологія веб-скрейпінгу не може замінити ручну роботу людини, коли користувач копіює та вставляє текст. У деяких випадках це єдине можливе рішення, наприклад, коли веб-сайти встановлюють блокування від веб-скрейпінгу та копіювання тексту.

2. Звернення до проксі-сервісу

Якщо сайт є html- або xml-документом і до нього дозволені крос-доменні запити, то можна отримати вміст документа за допомогою запиту до одного з наявних в Інтернеті проксі-сервісу .

3. Зіставлення текстових шаблонів

Простий, але потужний спосіб отримання інформації з веб-сторінок. Може бути заснований на команді UNIX grep (виконує пошук в одному або декількох файлах за шаблоном) або зіставлення регулярних виразів мов програмування (наприклад, Perl або Python).

4. Синтаксичний аналіз HTML

Багато веб-сайтів складаються з великої кількості сторінок, що динамічно генеруються з основного структурованого джерела – бази даних. Дані однієї категорії зазвичай кодуються в схожі сторінки за допомогою загального скрипта або шаблону. В інтелектуальному аналізі даних програма, яка виявляє такі шаблони у певному джерелі інформації, витягує його вміст та переводить його у форму, називається оболонкою. Передбачається, що аналізовані сторінки системи відповідають загальному шаблону і їх можна легко ідентифікувати в термінах загальної схеми URL. Крім того, деякі напівструктуровані мови запитів до даних, такі як XQuery та HTQL, можуть використовуватися для аналізу HTML-сторінок та отримання та перетворення вмісту сторінок.

5. Document Object Model (DOM)

DOM - програма з API для HTML-і XML-документів. Вбудовуючи повноцінний веб-браузер, наприклад Internet Explorer або елемент керування браузера Mozilla, програми можуть отримувати динамічний вміст, створюваний клієнтськими сценаріями. Скрейпінг DOM-дерева дозволяє отримати доступ до інформації в окремих її частях.

6. Вертикальна агрегація даних

Є кілька компаній, які розробили спеціальні онлайн-платформи, які створюють та контролюють безліч ботів. Боти працюють без прямої участі людини і при цьому їхня взаємодія з користувачами відбувається без зв'язку з цільовим сайтом. Підготовка включає створення бази знань, завдяки якій можлива робота роботів. Боти здійснюють агрегацію даних за окремими властивостями кожного ресурсу відповідно до заданих умов для подальшого зіставлення та аналізу отриманих значень властивостей. Надійність платформи вимірюється якістю

отриманої інформації (зазвичай кількістю полів) та її масштабованістю (до сотень чи тисяч сайтів). Ця масштабованість переважно використовується для перетворення даних, розташованих наприкінці довгого коду сайтів, які звичайні агрегатори вважають складними або надто трудомісткими для збору контенту.

7. Розпізнавання семантичних анотацій

Деякі сторінки можуть містити метадані або семантичну розмітку та анотації, за допомогою методу розпізнавання семантичних анотацій їх можна витягувати з таких сторінок.

8. Аналізатори сторінок

Ведуться розробки у сфері штучного інтелекту, коли машинний зір ідентифікує дані, інтерпретує їх, як це робить людина, та вилучає.

Технологія веб-скрейпінг дозволяє здійснювати отримання даних з веб-сторінок для подальшого їх аналізу. Враховуючи, що вміст сторінки може динамічно змінюватись, найбільш вдалою технологією для збору даних є *Document Object Model (DOM)*.

Процес отримання даних зі сторінки складається з наступних етапів (рисунок 1):

1. Коректне завантаження веб-сторінки.
2. Отримання з html документа потрібних даних.
3. Експорт даних у систему керування базами даних.
4. Перевірка достовірності результатів шляхом збереження даних у файл CSV формату для подальшого аналізу.
5. Розробка графічного інтерфейсу для користувача програми.

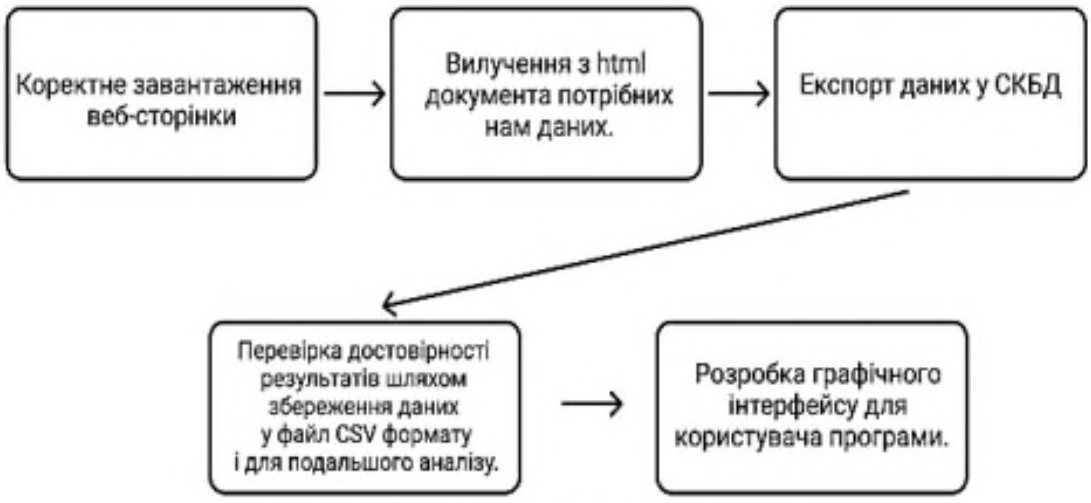


Рисунок 1 – Етапи отримання даних з веб-сторінки

Для реалізації інформаційної системи обрано мову програмування Python, так як вона має досить багато бібліотек для парсингу, таких як Scrapy, Beautiful-Soup, Selenium.

Отже, автоматизованої системи для збору та структурування даних з веб-ресурсів дозволить отримувати файл з структурованими даними для аналітики. Ця система може бути для різних інтернет магазинів, блогів і тому подібне. Основними перевагами системи є:

- автоматизація збору даних;
- економія людських і фінансових ресурсів,
- швидкість.

Перелік посилань

1. Робин Никсон. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5
2. Дэвид Скляр. Изучаем PHP 7. Руководство по созданию интерактивных веб-сайтов
3. Mitchell R. Web Scraping with Python / Ryan Mitchell. – Boston: O\'Reilly Media, 2015. – 256с.
4. Grune D. Parsing Techniques - A Practical Guide / D. Grune, C. Jacobs. – Chichester: Originally published by Ellis Horwood, 1990. – 320 с.
5. Модульне середовище для навчання [Електронний ресурс]. – Режим доступу: https://msn.khnu.km.ua/

Кваліфікаційна робота магістра

Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики

Виконав:

Студент 2 курсу, група КНм-20-1
Данчук Сергій Вікторович

Керівник роботи:

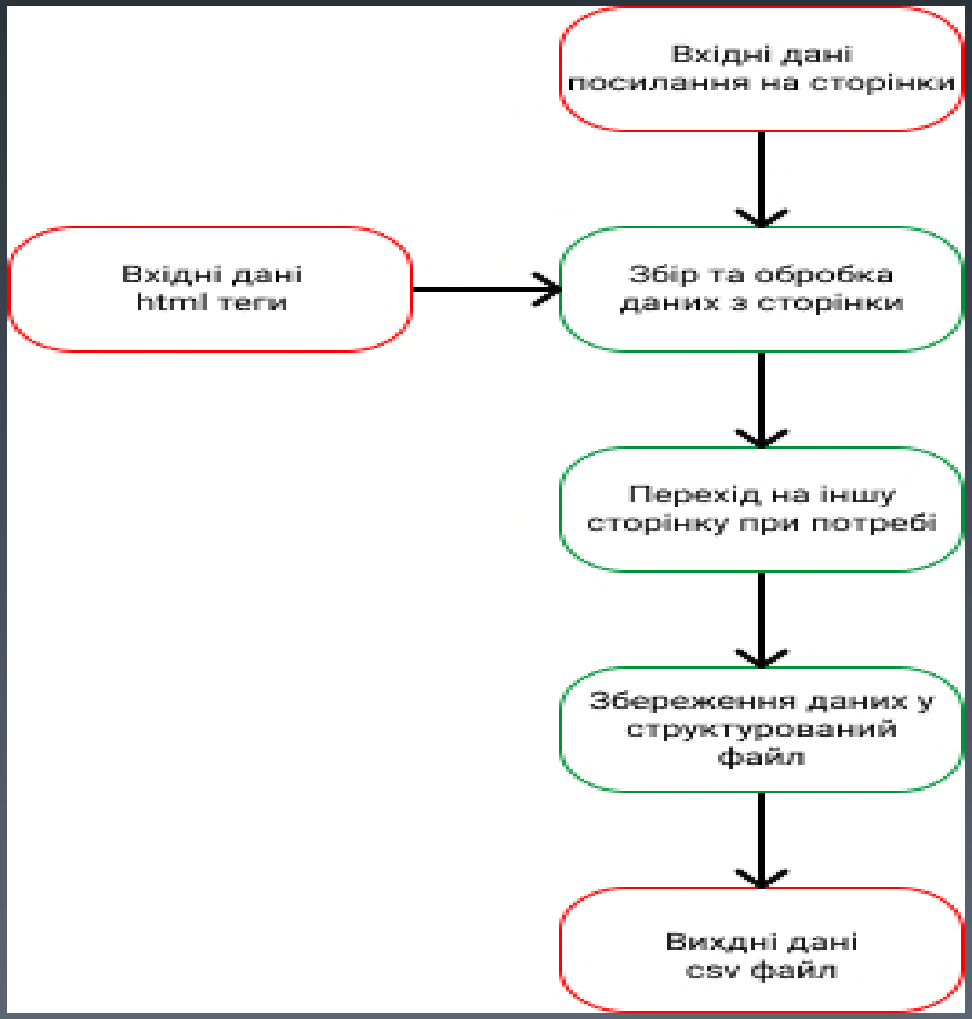
к.т.н., доцент кафедри КНІТ
Багрій Руслан Олександрович

Мета і задачі роботи. Метою магістерської роботи є створення технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики. Це дозволить ефективно аналізувати інтернет простір, швидко реагувати на зміни трендів та втілювати задумані бізнес ідеї в життя.

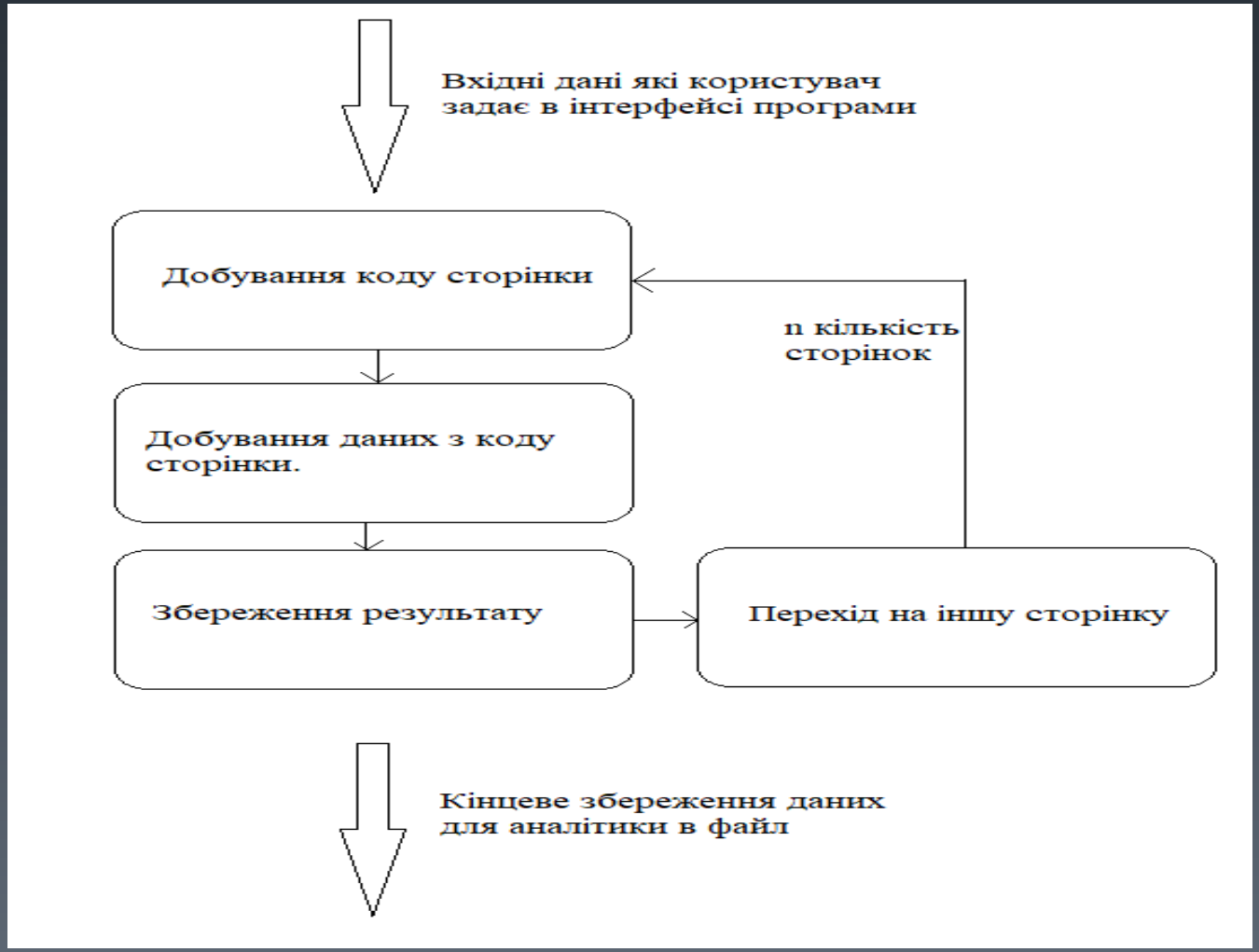
Для досягнення поставленої мети визначені наступні задачі:

- ✓ визначити типи даних, що необхідно отримувати з веб-сторінок для бізнес-аналітики;
- ✓ проаналізувати існуючі методи отримання даних з веб-ресурсів – web-scraping;
- ✓ розробити алгоритми синтаксичного розбору веб-сторінок та експорту даних;
- ✓ спроектувати інформаційну системи автоматизованого збору даних з веб-ресурсів на основі проведених досліджень;
- ✓ створити програмну реалізацію системи та провести її тестування;

Загальний опис системи автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики



Алгоритму синтаксичного розбору веб-сторінок.



Структуру і функції системи, а також потоки інформації і матеріальних об'єктів, що зв'язують ці функції.

Схема Актора

- Для розробки веб скрапера був обраний метод рекурсивного спуску - алгоритм низхідного синтаксичного аналізу, що реалізується шляхом взаємного виклику процедур

Обхід перешкод

- Проксі — сервер (комп'ютерна система або програма) в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі (через посередництво проксі-сервера) запити до мережевих сервісів
- Коректне завантаження сторінки
- Selenium WebDriver для доступу до інформації яка є інтерактивною
- Обхід капчі

Результат роботи покращення алгоритму

```
PS F:\parser> & C:/Users/Serhii/AppData/Local/Programs/Python/
Оброблено 1/17
Оброблено 2/17
Оброблено 3/17
Оброблено 4/17
Оброблено 5/17
Оброблено 6/17
Оброблено 7/17
Оброблено 8/17
Оброблено 9/17
Оброблено 10/17
Оброблено 11/17
Оброблено 12/17
Оброблено 13/17
Оброблено 14/17
Оброблено 15/17
Оброблено 16/17
Оброблено 17/17
Затрачено на роботу веб скрапера: 43.057941198349
```

```
PS F:\parser> & C:/Users/Serhii/AppData/Local/Programs/Python/Python310/python.exe
[INFO] Обробив сторінку 1
[INFO] Обробив сторінку 15
[INFO] Обробив сторінку 17
[INFO] Обробив сторінку 13
[INFO] Обробив сторінку 16
[INFO] Обробив сторінку 3
[INFO] Обробив сторінку 14
[INFO] Обробив сторінку 2
[INFO] Обробив сторінку 5
[INFO] Обробив сторінку 11
[INFO] Обробив сторінку 4
[INFO] Обробив сторінку 9
[INFO] Обробив сторінку 12
[INFO] Обробив сторінку 7
[INFO] Обробив сторінку 8
[INFO] Обробив сторінку 6
[INFO] Обробив сторінку 10
Затрачено на роботу веб скрапера: 6.227150917053223
```

В результаті роботи були отримані такі інновації та положення наукової новизни:

1. Вдосконалено алгоритм синтаксичного розбору веб-сторінки, що дає можливість отримувати та зберігати необхідні для бізнес-аналітики дані.
2. вдосконалено метод для доступу до веб-ресурсів, що дозволив обійти встановлені перешкоди для автоматизованої обробки даних.
3. Розроблено інформаційну системи автоматизованого отримання, обробки та збереження даних з веб ресурсів для подальшої бізнес-аналітики. Це дозволить скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі.

Висновки:

Запропонована алгоритм та система призначена для прискорення збору інформації з веб ресурсів для бізнес аналітики. Проаналізовано існуючі рішення.
Досліджено існуючі веб скрапери для добутку інформації. Розглянуто, що таке аналізатори і парсери. Детально розглянуто алгоритм добутку інформації з веб сторінки і запропоновано покращення що до його роботи. Також розглянуті проблеми з якими скрапер може стикнутись в ході роботи. Запропоновані рішення та створенні модулі на основі досліджень для виключення помилок в роботі скрапера.

Tue Dec 07 11:30:13 EET 2021, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 7.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 11%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 98328 Название: Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики Добавлено в БД: 2021-12-07 Авторы: С.В. Данчук Руководители: Р.О. Багрій Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	88872	782	12303 (14%)	117 (15%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

UNICHECK

Ім'я користувача:
Кафедра КН

ID перевірки:
1009571970

Дата перевірки:
07.12.2021 12:23:17 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.12.2021 12:25:27 EET

ID користувача:
100005671

Назва документа: **Кваліфікаційної_роботи_магістра_Данчук_С_В_Lite**

Кількість сторінок: **80** Кількість слів: **13573** Кількість символів: **106422** Розмір файлу: **943.26 KB** ID файлу: **1009578469**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

12.5%
Схожість

Найбільша схожість: **2.35%** з Інтернет-джерелом (https://stackabuse.com/introduction-to-the-python-lxml-library)

9.32% Джерела з Інтернету	174	Сторінка 82
3.99% Джерела з Бібліотеки	109	Сторінка 84

0.51% Цитат

Цитати	3	Сторінка 85

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	6
Підозріле форматування	13 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНИХ НАУК

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ МАГІСТРА ДО ЗАХИСТУ ЗА РЕЗУЛЬТАТАМИ АНАЛІЗУ ЗВІТУ ПОДІБНОСТІ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: *Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики*

Автор: *Данчук Сергій Вікторович*

Спеціальність: 122 – Компютерні науки

Освітня програма: освітньо-професійна

Науковий керівник: к.т.н., доц. Багрій Р.О.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) за програмою Anti-Plagiarism виявлені 7% є фрагментарними – містять поширені конструкції, загальновідомі терміни, скорочення та визначення.
2) За програмою UNICHECK виявлені 12.5% що є запозиченнями, які розміщені в розділах аналізу існуючих технологій та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 7% і 12.5% відповідно, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Руслан Багрій

Гарант ОП _____ Руслан Багрій

Завідувач кафедри КН _____ Олександр Бармак

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ
кафедра комп'ютерних наук

ВІДГУК ОПОНЕНТА

на кваліфікаційну роботу магістра

гр. КНм-20-1 Данчук Сергій Вікторович за темою: *Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики*

1. **Актуальність обраної теми**

Автоматизоване отримання даних з веб-ресурсів для бізнес-аналітики є актуальною темою, так як частка бізнеса в інтернеті росте з кожним роком, і збір даних з сайтів та їх подальший аналіз дає можливість швидко реагувати на зміну трендів.

2. **Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт**

Робота відповідає вимогам предметної області спеціальності 122 Комп'ютерні науки, оскільки в роботі виконуються теоретичні та експериментальні дослідження в галузі комп'ютерних наук, застосовуються алгоритми синтаксичного розбору текстових даних, проектування та розробка інформаційної системи.

3. **Повнота розкриття мети та завдань дослідження**

Метою магістерської роботи є створення технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики. Це дозволить ефективно аналізувати інтернет простір, швидко реагувати на зміни трендів та втілювати задумані бізнес ідеї в життя. Робота повністю розкриває поставлену мету та усі завдання, що були поставлені.

4. **Наявність наукової новизни**

В результаті проведеної роботи були отримані такі результати: вдосконалено алгоритм синтаксичного розбору веб-сторінки, що дає можливість отримувати та зберігати необхідні для бізнес-аналітики дані; вдосконалено метод для доступу до веб-ресурсів, що дозволив обійти встановлені перешкоди для автоматизованої обробки даних; розроблено інформаційну системи автоматизованого отримання, обробки та збереження даних з веб ресурсів для подальшої бізнес-аналітики. Це дозволить скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі.

5. **Зміст кожного розділу роботи**

В першому розділі подано визначення веб скрапінгу, розглянуто основні його терміни і типи. Проведено аналіз методів скрапінгу сторінки. Визначено недоліки алгоритму і системи в загальному та поставлено задачу розробити покращений алгоритм з врахуванням цих недоліків.

В другому розділі вдосконалено розглянуто алгоритми синтаксичного аналізу та запропоновано алгоритм скрапінгу з застосуванням асинхронного виконання коду, що пришвидшує отримання даних. Також запропоновано методи для доступу до веб-ресурсів, що дають можливість обходу перешкод при скрапінгу. Запропоновані варіанти збереження структурованої інформації для подальшої аналітики.

У третьому розділі визначено основні вимоги та запропоновано функціональну модель для забезпечення цих вимог в рамках інформаційної системи. Описано модулі та їх взаємодію.

У четвертому розділі проаналізовано обґрунтовано вибір програмної технології для розробки інформаційної системи. Розроблена система скрапінгу та продемонстровані результати роботи, а саме швидкість обробки веб-сторінок.

6. Ступінь розкриття теми роботи

В роботі недостатньо уваги приділено алгоритмічному опису запропонованого підходу синтаксичного розбору веб-сторінки. Основний акцент роботи був спрямований на практичне використання запропонованого алгоритму та проблеми, що виникають при цьому.

7. Якість оформлення кваліфікаційної роботи

Загалом оформлення роботи відповідає поставленим вимогам, але є певні недоліки що стосуються мови та граматики.

8. Недоліки кваліфікаційної роботи

В роботі недостатньо уваги приділено алгоритмічному опису запропонованого підходу синтаксичного розбору веб-сторінки, також є несуттєві недоліки в оформлені роботи.

9. Загальний висновок (допускається чи не допускається до захисту), якої оцінки заслуговує кваліфікаційна робота.

Роботі виконана студентом в достатньому обсязі, наявна наукова новизна та виконані усі поставлені завдання. Кваліфікаційна робота допускається до захисту, але враховуючи допущені помилки, заслуговує оцінку “задовільно”.

Опонент _____ к.т.н., доцент Бобровнікова К.Ю.

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МОН УКРАЇНИ
кафедра комп'ютерних наук

ВІДГУК НАУКОВОГО КЕРІВНИКА

на кваліфікаційну роботу магістра

гр. КНм-20-1 Данчука Сергія Вікторовича за темою: Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики

1. Актуальність теми

Актуальність теми достатньо обґрунтована, оскільки розробка інформаційної технології автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики дозволяє скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі.

2. Відповідність роботи предметній області спеціальності 122 Комп'ютерні науки та загальним вимогам до наукових робіт

Теми кваліфікаційної роботи "Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики" відповідає предметній області спеціальності 122 Комп'ютерні науки та вимогам до кваліфікаційної роботи магістра, оскільки об'єктом дослідження є процес отримання даних з веб-ресурсів, предметом дослідження – моделі, методи, підходи та засоби для отримання даних з веб-ресурсів.

3. Професійні та особистісні якості магістранта

Данчук С. В. під час роботи над кваліфікаційною роботою магістра продемонструвала посередній рівень знань та умінь за спеціальністю "Комп'ютерні науки".

4. Ступінь самостійності під час виконання кваліфікаційної роботи

Робота виконана самостійно, академічного плагіату не виявлено, стосовно всіх запозичень наведено відповідні посилання на джерела. Спільно з науковим керівником сформульовано мету та завдання дослідження, проведено обговорення отриманих результатів.

5. Наукова новизна та оригінальність запропонованих підходів

Отримані такі результати: вдосконалено алгоритм синтаксичного розбору веб-сторінки, що дає можливість отримувати та зберігати необхідні для бізнес-аналітики дані; вдосконалено метод для доступу до веб-ресурсів, що дозволив обійти встановлені перешкоди для автоматизованої обробки даних; розроблено інформаційну системи автоматизованого отримання, обробки та збереження даних з веб ресурсів для подальшої бізнес-аналітики, що дозволить скоротити час і витрати на аналіз даних, що є важливим в сучасному інтернет-бізнесі. Отримані результати оприлюднені на XIII всеукраїнської

науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021» 15 листопада 2021 р., м. Хмельницький, Україна, доповідь на тему «Технологія автоматизованого отримання даних з веб-ресурсів для бізнес-аналітики».

6. Ступінь оволодіння методами дослідження

Студент Данчук С. В. має достатній ступінь володіння методами дослідження, що були використанні у роботі.

7. Повнота та якість розкриття теми роботи

Мета роботи повністю розкрита, отримані результати підтверджують достовірність наукових положень.

8. Логічність, послідовність, аргументованість, літературна грамотність викладу матеріалу

Викладення матеріалу логічне, послідовне та аргументоване. Мова і стиль викладення кваліфікаційної роботи магістра відповідають стандартам, що забезпечує доступність сприймання матеріалу і відповідає вимогам до сучасних наукових робіт.

9. Можливість практичного застосування кваліфікаційної роботи, окремих її частин

Може мати практичне значення при отримання та збереження даних з веб-ресурсів для бізнес-аналітики, при цьому дає можливість обійти встановлені перешкоди для автоматизованої обробки даних.

10. Висновок про можливість допуску кваліфікаційної роботи до захисту, на яку оцінку заслуговує робота

Вважаю, що кваліфікаційна робота студента Данчука Сергія Вікторовича може бути рекомендована до захисту та заслуговує на оцінку "задовільно".

Науковий керівник _____ к.т.н., доц. Руслан Багрій