УДК 004.738.52

ЦІНОВА ОПТИМІЗАЦІЯ ЗАМОВЛЕНЬ У ОНЛАЙН-КНИГАРНЯХ НА ОСНОВІ ВЕБСКРАПІНГУ

Р. Селіверстов, І. Семчук

Львівський національний університет імені Івана Франка,
вул. Університеська, 1, Львів, 79000,
e-mail: roman.seliverstov@lnu.edu.ua, irynaa.semchuk@gmail.com

Розглянуто вебскрапінг як інструмент збору інформації про послуги онлайнових книгарень і проведено подальший аналіз цієї інформації для формування оптимального за ціною замовлення. Виявлено недоліки вітчизняних онлайнових сервісів для купівлі друкованої художньої літератури. Зокрема, немає змоги отримати найкращу за ціною пропозицію на запит щодо купівлі набору книг не в межах однієї книгарні. Детально описано розроблений на мові Python з використанням бібліотеки BeautifulSoup прототип вебскрапера для отримання цін на книги й умови доставки з сайтів онлайн-книгарень. Запропоновано два підходи до формування на підставі цих даних замовлення списку книг за мінімально можливою ціною. Перший ґрунтується на переборі всіх можливих комбінацій замовлень, другий використовує пошук шляху на орієнтованому графі. Програмна реалізація цих підходів додана до функціональності вебскрапера, що дало змогу усунути згаданий вище недолік і забезпечити якісно новий сервіс. Проаналізовано часові характеристики продуктивності запропонованих підходів за різних гіпотетичних обставин. На підставі цього аналізу вироблено рекомендації стосовно вибору ефективнішого підходу залежно від отриманих даних. Вхідними даними для скрапера є перелік книг із зазначенням кількості примірників, яку бажає придбати користувач. На виході користувач отримує сукупність URL-адрес для придбання книг. У цьому випадку гарантується мінімально можлива загальна вартість замовлення (враховуючи вартість доставок) на певній множині онлайн-книгарень. Наведено результат роботи розробленого вебскрапера на прикладі тестового замовлення. Дані використовували з чотирьох книгарень, технічні служби яких надали дозвіл на автоматизоване опрацювання їхньої інформації. Для зменшення часу виконання програми використано багатопотоковість. Зазначено можливі напрями розширення функціональності програмного продукту та його застосування.

Ключові слова: пошук інформації, вебскрапінг, оптимізація, Python.

1. Вступ

Для підвищення ефективності своєї діяльності підприємства, установи й інші організації активно використовують інформацію, зокрема, отриману з онлайн-ресурсів. Щоб автоматизувати збирання такої інформації, зазвичай застосовуються існуючі пошукові системи або власноруч розроблені чи придбані пошукові машини, функціонування яких ґрунтується на технології вебскрапінгу. Вебскрапінг – це технологічні інструменти для виймання та структуризації даних з Інтернету для подальшого аналізу [1]. Підходи до збирання й обробки даних з використанням вебскрапінгу детально подано у працях [2–4]. Особливості та приклади використання вебскрапінгу для вирішення конкретних задач з опрацювання інформації неодноразово ставали предметом наукових досліджень, наприклад, [5–14]. Правові й етичні аспекти вебскрапінгу описані у [15,16]. Доступність і простота використання готових інструментів і бібліотек для вебскрапінгу (BeautifulSoup, Cheerio, Jsoup, Scraping-Bot, Scrapy, Selenium тощо) дає змогу не тільки науковцям і фахівцям

з інформаційних технологій, а й пересічним користувачам, знайомим з основами програмування, розробляти свої невеликі програмні продукти для вирішення власних потреб: відображення тематичних новин, перевірки наявності товару та порівняння цін в онлайн-магазинах, пошуку дешевих авіаквитків, квитків на театральні дійства тощо [17–19].

Передумовою цього дослідження стала обмеженість функціональності існуючих вітчизняних онлайнових сервісів для купівлі книг. Результати Google-пошуку за прізвищем автора, назвою книги та словом "купити" – це зазвичай посилання на веб-сторінки онлайнових книгарень. Для того, щоб відшукати кращу пропозицію, потрібно почергово переходити на ці сторінки. Якщо врахувати, що до ціни на книгу потрібно додати вартість доставки, то рутини лише більшає. Запит за фразою "порівняти ціни на книги" видає два релевантні результати – пропозицію порівняти ціни від маркетплейсу України prom.ua та знайти потрібну книгу в інтернет-крамницях України на сайті bookfinder.com.ua. Проте ці пропозиції далекі від досконалості. Зокрема, вони не враховують вартість доставки та дають змогу порівняти ціни тільки на одну книгу.

Зрозуміло, що для мінімальної за ціною купівлі кількох (багатьох) книг недостатньо знайти найкращу пропозицію по кожній позиції. Адже, якщо такі пропозиції "розкидані" по різних книгарнях, то доведеться заплатити за доставку з кожної. Іноді трохи дорожча ціна на деякі книги може компенсуватися меншою кількістю доставок. Більше того, деякі книгарні забезпечують безкоштовну доставку за умови замовлення на певну суму, що також потрібно враховувати. Такі особливості можна продовжувати: знижки для зареєстрованих користувачів, постійних клієнтів тощо. Зі збільшенням обсягу замовлення та кількості онлайн-книгарень кількість всіляких можливих варіантів зростає і без спеціалізованого програмного забезпечення вже не обійтися.

У цій статті описано прототип розробленого веб-скрапера для отримання з онлайн-книгарень цін на книги й умови доставки, запропоновано та програмно реалізовано два підходи до формування на основі цих даних замовлення списку книг за мінімально можливою ціною.

2. Формулювання задачі та загальний опис алгоритму

Вхідними даними задачі є перелік книг (із зазначенням кількості примірників), які бажає придбати користувач. Основним результатом – URL для кожної книги, перейшовши за яким, користувач може придбати її. Результуюча сукупність URL-адрес має гарантувати мінімальну загальну ціну замовлення на певній множині онлайн-книгарень (вважається, що частини замовлення можуть виконуватись у різних книгарнях), враховуючи вартість доставок. Додатковими (супровідними) результатами є: ціна, за якою буде придбано книгу; вартості замовлень у кожній книгарні з урахуванням кількості примірників, цін та умов доставки; загальна вартість замовлення.

Загальний алгоритм розв'язку цієї задачі подамо у вигляді дворівневого списку.

1. Зчитування вхідних даних (які книги та у якій кількості потрібно придбати).
2. Отримання інформації про наявність та ціну для кожної книги у кожній книгарні:
   - формування URL пошукового запиту за автором і назвою книги на сайті книгарні;

- отримання HTML-коду сторінки з результатами запиту;
- формування списку посилань на книги, відображені в результатах запиту;
- почерговий перехід за посиланнями зі сформованого списку та верифікація автора та назви до того моменту, доки не отримаємо HTML-код потрібної книги або не зробимо висновок про її відсутність у цій книгарні;
- збереження ціни книги та URL-адреси для її замовлення за умови встановлення її наявності.

3. Отримання інформації про умови доставки з кожної книгарні:
   - отримання HTML-коду сторінки з умовами доставки;
   - вийняття та збереження інформації про вартість доставки;
   - вийняття та збереження інформації про суму замовлення, починаючи з якої доставка безкоштовна, якщо така послуга надається.
4. Формування множини всіляких можливих варіантів замовлень.
5. Обчислення загальної вартості замовлення для кожного варіанту.
6. Знаходження та відображення варіанту замовлення, якому відповідає мінімальна вартість.

Кроки та особливості цього алгоритму детальніше описані у наступних двох розділах і супроводжуються фрагментами програмної реалізації на мові Python 3.

3. Отримання даних з онлайн-книгарень

Виконання програми розпочинається зі зчитування вхідних даних з текстового файлу order.txt, кожен рядок якого через певний розділювач (у нашому випадку це крапка з комою) містить прізвище автора, назву книги та кількість примірників, яку бажає придбати користувач. Вважатимемо, що файл order.txt заповнений коректно. На основі інформації з нього за допомогою функції *get_order_list()* формується список замовлення *order_list* як список словників з ключами *"id"* (порядковий номер книги у замовленні), *"author"* (автор), *"title"* (назва) і *"amount"* (кількість):

```
def get_order_list(filename, sep=";"):
    id_, list_ = 0, []
    with open(filename, encoding="utf-8") as order:
        for line in order:
            info = line.split(sep)
            list_.append({"id": id_, "author": info[0].strip(),
                          "title": info[1].strip(),
                          "amount": int(info[2].strip())})
            id_ += 1
    return list_

order_list = get_order_list("order.txt")
```

Зазначимо, що тут і надалі у наведених фрагментах коду для економії простору свідомо порушено деякі рекомендації PEP 8, три крапки позначають пропущений код, а також вважається, що увесь код міститься в одному файлі.

Отримання інформації про ціну книги та умови доставки реалізовано у вигляді функцій *get_price()* і *get_delivery()*, яким передається доменне ім'я сайту, автор і назва книги, та, за потреби, інші необов'язкові аргументи: кодування символів, назву парсера, дані для авторизації тощо. Для ефективного використання цих

функцій створено конфігураційний словник *hosts*, ключами якого є доменні імена сайтів книгарень, а значеннями – інші словники, які містять необхідні для виконання пошуку значення аргументів. Типовий фрагмент словника *hosts* має такий вигляд:

```
hosts = {"bookclub.ua":
          {"encoding": "windows-1251",
           "parser": "html.parser",
           ... },
          ... }
```

Кожному ключу словника *hosts* відповідає окремий блок *if* у тілі функцій *get_price()* і *get_delivery()*. Це пов'язано з тим, що написання універсального скрапера, який би виймав потрібну інформацію про книгу й умови доставки з сайту будь-якої книгарні, вкрай складна задача. Але усі ці блоки працюють за однаковим загальним алгоритмом, який розглянемо детальніше на прикладі отримання ціни книги.

Перший крок цього алгоритму полягає у формуванні URL пошукового запиту як конкатенації доменного імені сайту книгарні (*host*), відносного шляху до пошукової сторінки (*search_path*) і власне пошукового запиту (*query*), який за замовчуванням збігається з назвою книги:

```
url = f"https://{host}{search_path}{query}"
```

Для правильного передавання в URL символів кирилиці і пробілів, з яких складається текст запиту, використовується одна з функцій *quote()* або *quote_plus()* модуля parse бібліотеки urllib (вибір функції залежить від прийнятого на сайті способу кодування):

```
from urllib.parse import quote, quote_plus
query = quote_plus(title, encoding=hosts[host]["encoding"])
# або query = quote(title)
```

Далі, використовуючи модуль request бібліотеки urllib, отримується HTML-код сторінки з результатами пошукового запиту, на підставі якого створюється об'єкт BeautifulSoup [20]:

```
from urllib.request import urlopen, Request
from bs4 import BeautifulSoup
html = urlopen(Request(url, headers={"User-Agent": "Mozilla/5.0"}))
obj = BeautifulSoup(html.read(), features=hosts[host]["parser"])
```

Застосовуючи до цього об'єкта метод *find_all()*, матимемо список тегів, які містять результати пошукового запиту:

```
tags = obj.find_all(tagname, classname, limit=3)
```

Тут *tagname* – тип тегу (зазвичай *"div"* або *"span"*), *classname* – значення атрибута class цього тегу, *limit* – обмеження на максимальну кількість результатів пошуку (потрібна книга майже завжди є серед перших).

Обходячи поелементно список *tags*, виконуємо такі дії для кожного елемента (*item*):

1. Застосовуючи один чи кілька разів метод *find()*, переходимо до тегу, який містить посилання на вебсторінку книги, яке отримуємо як значення атрибута href тегу <a>:

   ```
   url = item.find(...).find("a").attrs["href"]
   ```

2. Переходимо за отриманим на попередньому кроці посиланням.
3. Шукаємо теги, які містять інформацію про автора та назву книги, та перевіряємо, чи відповідає вона зазначеній у замовленні (значення аргументів *author* і *title* передаються функції *get_price()* під час її виклику). Якщо виявилося, що ми потрапили на вебсторінку не тієї книги, то переходимо до наступного елемента списку *tags* і заново виконуємо дії, починаючи з пункту 1. Інакше – переходимо до виконання наступного пункту.
4. Знаходимо тег, який містить інформацію про ціну книги, отримуємо цю інформацію методом *get_text()*. Використовуючи зрізи або регулярні вирази, виймаємо значення ціни та конвертуємо його у числовий тип.
5. Повертаємо отримані у пунктах 1 і 4 значення *url* та ціни (*price*). Якщо список *tags* матиме нульову довжину або ніколи не досягнеться пункт 4, то функція *get_price()* поверне за замовчуванням значення None, яке означатиме, що на сайті книгарні немає інформації про затребувану книгу.

Щоб збільшити швидкодію, скрапінг кожного сайту реалізовано в окремому потоці:

```
from threading import Thread

def thread(order_list, host):
    for item in order_list:
        args = (item["author"], item["title"], host)
        price_info[item["id"]][host] = get_price(*args)
    delivery_info[host] = get_delivery(host)

price_info = {key: {} for key in range(len(order_list))}
delivery_info = {key: None for key in hosts.keys()}

threads = [Thread(target=thread, args=(order_list, host))
           for host in hosts]
for item in threads:
    item.start()
for item in threads:
    item.join()
```

У підсумку матимемо заповненими два словники – *price_info* і *delivery_info*. Ключами першого з них є ідентифікатори книг (їхні порядкові номери у списку замовлень), а значеннями – інші словники, ключами яких є доменні імена сайтів книгарень, а значеннями – кортежі, які містять ціну книги в книгарні та посилання на сторінку для замовлення цієї книги (або значення None у разі відсутності книги в книгарні). Ключами другого словника є доменні імена сайтів книгарень, а значеннями – кортежі, які містять вартість доставки та суму замовлення, починаючи з якої доставка безкоштовна (якщо послуги безкоштовної доставки немає, то цей елемент кортежа дорівнює значенню inf з математичного модуля math, тобто нескінченності).

4. Аналіз даних: пошук замовлення мінімальної вартості

Розглядатимемо замовлення як кортеж *tpl* елементів, індекси яких збігаються з ідентифікаторами книг, а самі елементи дорівнюють ідентифікаторам книгарень, які ототожнюються з індексами списку *markets* ключів конфігураційного словника *hosts*:

```
markets = list(hosts.keys())
```

Тоді множина всіляких можливих варіантів замовлень дорівнює множині розміщень з повтореннями ідентифікаторів книгарень по k елементів, де k – кількість книг (не примірників) у замовленні. Щоб заощадити пам'ять, замість формування цієї множини створено ітератор *gen_orders*, який генеруватиме її елементи "на льоту", не записуючи в жодну структуру:

```
from itertools import product
gen_orders = product(range(len(markets)), repeat=len(order_list))
```

Тепер пошук мінімального за ціною замовлення зводиться до класичного алгоритму пошуку мінімуму, який можна реалізувати, наприклад, так:

```
min_price, order = inf, None
for tpl in gen_orders:
    p = price(price_info, delivery_info, tpl)
    if p and p[0] < min_price:
        min_price, order = p
```

У наведеному вище фрагменті коду використано функцію *price()*, яка повертає вартість переданого варіанту замовлення і запис цього замовлення, згрупованого за книгарнями:

```
def price(price_info, delivery_info,  order):
    shop_numbers, book_numbers = len(markets), len(order_list)
    orders = [[] for _ in range(shop_numbers)]
    # групування замовлень за книгарнями
    for i in range(len(order)):
        orders[order[i]].append(i)
    # обчислення вартості замовлення за цінами на книги:
    prices = [0 for _ in range(shop_numbers)]
    for shop in range(shop_numbers):
        for book in orders[shop]:
            try:
                amount = order_list[book]["amount"]
                prices[shop] += price_info[book][markets[shop]][0]*amount
            except:
                return False    # такий варіант замовлення неможливий
        # уточнення вартості замовлення з урахуванням доставки:
        if orders[shop]:
            if prices[shop] < delivery_info[markets[shop]][1]:
                prices[shop] += delivery_info[markets[shop]][0]
    return sum(prices), orders
```

У підсумку у змінній *order* матимемо запис мінімального замовлення у вигляді списку, індексами якого є ідентифікатори книгарень, а елементами – списки ідентифікаторів книг, які потрібно замовити у цих книгарнях. Загальна (мінімальна) вартість замовлення записана у змінній *min_price*.

Генерування всіляких можливих розміщень з повтореннями має недолік – зі збільшенням кількості книгарень і обсягу замовлення значно зростає час виконання програми. Якщо припустити, що кожна окрема книга наявна не у кожній книгарні, то множину всіляких можливих варіантів замовлень *orders* можна зменшити до реальної кількості (лише ті варіанти, які можливо виконати), подавши її як множину шляхів у деякому орієнтованому графі.

Вузли згаданого графа розміщені на $k + 2$ (k – кількість унікальних книг у замовленні) рівнях, які нумеруються, починаючи з 0. На нульовому рівні є єдиний стартовий вузол v_0. На наступних k рівнях розміщені основні вузли графа v_{ij}. Індекс i, який змінюється від 1 до k, позначає номер рівня і є фактично збільшеним на одиницю ідентифікатором книги у замовленні. Індекс j, який змінюється від 1 до кількості книгарень, є збільшеним на одиницю ідентифікатором книгарні. На останньому рівні графа є єдиний заключний вузол. Ребра графа будуються за таким правилом: якщо у книгарні з ідентифікатором $j - 1$ є в наявності книга з ідентифікатором $i - 1$, то у вузол v_{ij} входять ребра з усіх вузлів попереднього рівня. У цьому випадку у заключний вузол входять ребра з усіх попередніх вузлів, які відповідають книгарням, у яких є в наявності остання книга зі списку замовлень. Нижче наведений код для побудови матриці суміжності такого графа (з наскрізною нумерацією вузлів) за даними, отриманими на підставі скрапінгу:

```
def build_adj_list(data, shop_list):
    books, shops = len(data), len(shop_list)
    adj = {node: [] for  node in range(books*shops+2)}
    for i in range(shops):
        if data[0][shop_list[i]]:
            adj[0].append(i+1)
    for j in range(books-1):
        for i in range(shops):
            for k in range(shops):
                if data[j+1][shop_list[k]]:
                    adj[j*shops+i+1].append((j+1)*shops+k+1)
    for i in range(shops):
        if data[books-1][shop_list[i]]:
            adj[(books-1)*shops+i+1].append(books*shops+1)
    return adj

adj = build_adj_list(price_info, markets)
```

Тепер для пошуку мінімального за ціною замовлення використано рекурсивну функцію на основі загальновідомого алгоритму пошуку в глибину. Усі можливі шляхи не зберігаються – зберігається лише один шлях, який відповідає поточній мінімальній вартості замовлення. Знайшовши інший шлях (тобто, варіант замовлення), за допомогою функції *price()* обчислюється його вартість і порівнюється з поточною мінімальною вартістю. Це дає змогу заощадити пам'ять комп'ютера:

```
def find_min(graph, start, end):
```

```
    global min_price, order
    if start == end:
        p = price(price_info, delivery_info, route_to_order(route))
        if p and p[0] < min_price:
            min_price, order = p
    else:
        for node in graph[start]:
            if not visited.get(node, False):
                visited[node] = True
                route.append(node)
                find_min(graph, node, end)
                visited[node] = False
                route.remove(node)

visited, route = {}, []
min_price, order = inf, None
find_min(adj, 0, max(adj.keys()))
```

У тілі функції *find_min()* використано функцію *route_to_order()*, яка перетворює шлях по графу в таке подання замовлення, яке сприймає функція *price()*:

```
def route_to_order(route):
    order=[]
    for node in route[:-1]:
        if node>len(markets):
            order.append((node-1)%len(markets))
        else:
            order.append(node-1)
    return order
```

Такий підхід за умови не наявності окремих книг у деяких книгарнях міститиме меншу кількість елементів порівнянно з попереднім. Особливо це стосується вимання даних з великої кількості книгарень або доволі великих за обсягом замовлень. Щоправда, у другому випадку може виникнути інша проблема, пов'язана з перерахунком вартості доставки через збільшення ваги відправлення.

Зазначимо, що не завжди менша кількість варіантів означає менший час виконання програми. Як з'ясуємо у наступному підрозділі, досягти меншого часу виконання програми можна лише за певного рівня розрідженості матриці суміжності, який можна наближено оцінити, наприклад, як відношення сумарної кількості відмінних від None значень у словниках-значеннях словника *price_info* до добутку кількості книгарень на кількість унікальних книг у замовленні.

Для відображення інформації про склад шуканого оптимального за ціною замовлення у зручному для сприйняття вигляді використано функцію *display()*, код якої не наводимо з міркувань його низької інформативності.

5. Приклад застосування

Поточна версія розробленого вебскрапера виймає дані з чотирьох сайтів, технічні служби яких надали дозвіл на кількаразову автоматизовану обробку їхньої інформації через форму зворотного зв'язку: онлайн-книгарні "Видавництва Старого Лева"

(https://starylev.com.ua), книжкових інтернет-магазинів "Клуб сімейного дозвілля" (https://bookclub.ua) і Yakaboo (https://www.yakaboo.ua) та книжкової інтернет-платформи BamBook (https://bambook.com). Розглянемо його застосування на прикладі таких вхідних даних (вміст файлу order.txt):

```
Кідрук; Доки світло не згасне назавжди; 3
Лис; Обітниця; 1
Шкляр; Характерник; 2
Рутківський; Джури і підводний човен; 1
Кундера; Нестерпна легкість буття; 2
Крижанівська; Тіні; 2
```

У підсумку отримано такий варіант найменшого за ціною замовлення (скрапінг здійснювався 17 червня 2020 року):

```
bookclub.ua:
    "Доки світло не згасне назавжди" (Кідрук): 3 * 125.0 = 375.0 грн.
    URL: https://bookclub.ua//catalog/books/pop/product.html?id=51399
    "Обітниця" (Лис): 1 * 99.0 = 99.0 грн.
    URL: https://bookclub.ua//catalog/books/pop/product.html?id=51853
    "Характерник" (Шкляр): 2 * 125.0 = 250.0 грн.
    URL: https://bookclub.ua//catalog/books/pop/product.html?id=51855
    Доставка: 0.0 грн.

yakaboo.ua:
    "Джури і підводний човен" (Рутківський): 1 * 130.0 = 130.0 грн.
    URL: https://www.yakaboo.ua/ua/dzhuri-i-pidvodnij-choven-trilogija-
         dzhuri-kniga-3.html
    "Нестерпна легкість буття" (Кундера): 2 * 150.0 = 300.0 грн.
    URL: https://yakaboo.ua/ua/nesterpna-legkist-buttja.html
    "Тіні" (Крижанівська): 2 * 120.0 = 240.0 грн.
    URL: https://yakaboo.ua/ua/tini.html
Доставка: 40.0 грн.

Загальна вартість замовлення: 1434.0 грн.
```

Зауважимо, що під час скрапінгу виймалася вартість доставки для служби доставки "Нова пошта", зазначена на сайтах книгарень. Пропозиції інших служб доставки, а також можлива вага відправлення до уваги не бралися.

Левову частку часу (трохи більше 20 секунд) зайняло виймання даних з сайтів. Такого досить сприйнятливого для користувача часу вдалося досягти завдяки використанню багатопотоковості. У разі виконання пошуку по усіх книгарнях в одному потоці затрачалося близько однієї хвилини. Пошук мінімального за ціною замовлення тривав 0.03 секунди за використання першого підходу (було згенеровано 4096 розміщень з повтореннями, які репрезентують всілякі можливі замовлення) та 0.005 секунди за використання другого (знайдено 324 шляхи на графі, які відповідають тим замовленням, які можливо виконати).

Для того, щоб порівняти зазначені вище підходи, на більшій кількості книгарень були випадково згенеровані тестові дані та порівнювався час генерування розміщень і шляхів, відповідно. Для 10 книгарень і 9 унікальних книг в рамках першого

підходу 10^9 всіляких можливих варіантів замовлень генерувалися в межах 1,5 хвилин. За умови гіпотетичної не наявності кожної книги в трьох з цих книгарень застосування другого підходу дало змогу зменшити час виконання (затрачений на генерування шляхів на графі) майже вдвічі, а у чотирьох книгарнях – ще принаймі втричі. Проте при невеликій розрідженості матриці суміжності графа ситуація протилежна – за умови гіпотетичної не наявності кожної книги лише у двох книгарнях час виконання зростав у 2,5 раза.

6. Висновки

На підставі аналізу наявних вітчизняних онлайнових книжкових сервісів було з'ясовано, що жоден з них не дає змоги сформувати найдешевше замовлення набору книг, враховуючи інформацію не в межах однієї, а з багатьох книгарень, а також комбінуючи різноманітні способи доставки. Розроблений у межах цього дослідження прототип вебскрапера усуває цей недолік і є зручним інструментом для формування оптимального за ціною замовлення щодо невеликих обсягів друкованої літератури в онлайн-книгарнях. Результати його виконання самодостатні, а для запуску потрібен єдиний вхідний аргумент – текстовий файл або список замовлення з трьох полів: автор книги, назва книги та кількість примірників.

Функціональність вебскрапера може бути розширена шляхом автоматизації відкриття результуючих URL у браузері (для цього достатньо можливостей модуля Webbrowser) та натискання кнопок типу "Додати до кошика" для формування замовлення (тут для взаємодії з браузером найкращим вбачається використання пакету Selenium). Стосовно збільшення кількості книгарень для вебскрапінгу, то для додавання книгарні достатньо написати для неї окрему незалежну функцію/блок. Наявний код, окрім запису у конфігураційному словнику, змінювати не потрібно.

Щоб уникнути можливих проблем, які стосуються порушення прав інтелектуальної власності, перед використанням вебскрапера рекомендується подбати про відповідні дозволи, оскільки майже усі онлайнові книгарні у файлі robots.txt (Стандарт винятків для роботів, Robots Exclusion Standard [21]) кореневого каталогу своїх сайтів декларують заборону на використання користувацьких автоматизованих роботів. Особливо це стосується випадків подальшого публічного чи комерційного використання отриманих даних.

Описані підходи та алгоритми мають навчальне значення, оскільки дають змогу обізнаним з програмуванням та зацікавленим у вебскрапінгу користувачам розробляти свої програмні продукти.

Науково-практичне значення вебскрапера полягає у тому, що він забезпечує якісно новий, порівняно з існуючими, сервіс і може слугувати основою для розробки потужнішого продукту, який зможуть використовувати на постійній основі організації, дотичні до друкованої продукції – книжкові магазини, бібліотеки, літературні клуби тощо.

Список використаної літератури

1. *Krotov V.* Scraping Financial Data from the Web Using R Language / V. Krotov, M. Tennyson // Journal of Emerging Technologies in Accounting. – 2018. – Vol. 15, No. 1. – P. 169–181.
2. *Mitchell R.* Web Scraping with Python / R. Mitchell. – O'Reilly Media, Inc., 2018. – 2 ed. – 306 p.
3. The A-Z of Web Scraping in 2020 [A How-To Guide] [Електронний ресурс] / Dmitry Nar-

izhnykh // Hackernoon, 2020. – Режим доступу: https://hackernoon.com/the-a-z-of-web-scraping-in-2020-a-how-to-guide-sg263y8d – Назва з екрану.

4. *Государев И.* Web scraping как программный процес извлечения и обработки данных в вебе / И. Государев, Н. Жуков, П. Бабарицкий // Современнон образование: традиции и инновации. – 2020. – № 1. – С. 146–152.
5. *Boeing G.* New Insights into Rental Housing Markets across the United States: Web Scraping and Analyzing Craigslist Rental Listings / G. Boeing, P. Waddell // Journal of Planning Education and Research. – 2016. – Vol. 37, No 4. – P. 457–476. – doi:10.1177 / 0739456X16664789.
6. *Glez-Pena D.* Web scraping technologies in an API world / D. Glez-Pena, A. Lourenco, H. Lopez-Fernandez, M. Reboiro-Jato, F. Fdez-Riverola // Briefings in Bioinformatics. – 2014. – Vol. 15, No 5. – P. 788–797. – doi:10.1093/bib/bbt026.
7. *Haddaway N.* The use of web-scraping software in searching for grey literature / N. Haddaway // Grey J. – 2015. – Vol. 11, No 3. – P. 186–190.
8. *Hassanien H.* Web Scraping Scientific Repositories for Augmented Relevant Literature Search Using CRISP-DM / H. Hassanien // Appl. Syst. Innov. – 2019. – Vol. 2, No 37. – doi:10.3390/asi2040037.
9. *Kumar P.* Web Information Retrieval using JShop and Python / P. Kumar, P. Sharma. V. Singh // International Journal for Research in Applied Science & Engineering Technology. – 2020. – Vol. 8, No 6. – P. 1966–1969. – doi:10.22214/ijraset.2020.6322.
10. *Meschenmoser P.* Scraping Scientific Web Repositories: Challenges and Solutions for Automated Content Extraction / P. Meschenmoser, N. Meuschke, M. Hotz, B. Gipp // D-Lib Magazine. – 2016. – Vol. 22, No 9-10. – doi:10.1045/september2016-meschenmoser.
11. *Vargiu E.* Exploiting web scraping in a collaborative filtering-based approach to web advertising / E. Vargiu, M. Urru // Artificial Intelligence Research. – 2013. – Vol. 2, No 1. – P. 44–54. – doi:10.5430/air.v2n1p44.
12. *Басалаева А.* Web-scraping и классификация текстов методом наивного Байеса / А. Басалаева, Г. Гареева, Д. Григорьева // Инновационная наука. – 2018. – Т. 2, № 5ю – С. 11–14.
13. *Москаленко А.* Разработка приложения веб-скрапинга с возможностями обхода блокировок / А. Москаленко, О. Лапонина, В. Сухомлин // Современные информационные технологии и ИТ-образование. – 2019. – Т. 15, № 2. – С. 413–420. – doi:10.25559 / SITITO. 15.201902.413-420.
14. *Вакуленко Ю.* Застосування методу парсингу для ефективного пошуку інформації у дослідницькій діяльності / Ю. Вакуленко, О. Щербіна // Вісник студентського наукового товариства ДонНУ імені Василя Стуса. – 2019. – Т. 2., № 11. – С. 153–156.
15. *Dryer A.* Internet "Data Scraping": A Primer for Counseling Clients / A. Dryer, J. Stockton // New-York Law Journal. – 2013. – Vol. 15. – P. 1–3.
16. *Krotov V.* Legality and Ethics of Web Scraping [Електронний ресурс] / V. Krotov, L. Silva // Twenty-fourth Americas Conference on Information Systems. – New Orleans, 2018. – Режим доступу: https://researchgate.net/publication/324907302 _Legality _and _Ethics _of_Web_Scraping
17. The Easy Way to Web Scrape Articles Online [Електронний ресурс] / Andrew Berry // Medium. – 2020. – Режим доступу: https://towardsdatascience.com/the-easy-way-to-web-scrape-articles-online-d28947fc5979 – Назва з екрану.
18. Web Scraping Tutorial with Python: Tips and Tricks [Електронний ресурс] / Jekaterina Kokatjuhha // Hackernoon. – 2017. – Режим доступу: https://hackernoon.com/web-scraping-tutorial-with-python-tips-and-tricks-db070e70e071 – Назва з екрану.
19. От парсера афиши театра на Python до Telegram-бота. Часть 1 [Електронний ресурс] / YuliyaCl // Хабр. – 2019. – Режим доступу: https://habr.com/ru/post/444460 – Назва з екрану.

20. Beautiful Soup Documentation [Електронний ресурс]. – Режим доступу: https://crummy.com/software/BeautifulSoup/bs4/doc/
21. A Standard for Robot Exclusion [Електронний ресурс] / Martijn Koster. – Режим доступу: http://www.robotstxt.org/orig.html – Назва з екрану.

Стаття: надійшла до редколегії 06.07.2020
доопрацьована 14.09.2020
прийнята до друку 23.09.2020

PRICE OPTIMIZATION OF ORDERS IN ONLINE BOOKSTORES ON THE BASIS OF WEB SCRAPING

R. Seliverstov, I. Semchuk

Ivan Franko National University of Lviv,
Universytetska str, 1, Lviv, 79000,
e-mail: roman.seliverstov@lnu.edu.ua, irynaa.semchuk@gmail.com

The article considers web-scraping as a tool for collecting information about online bookstore services and further analysis of this information to form the best-priced order. The shortcomings of domestic online services for the purchase of printed fiction are identified. In particular, it is not possible to obtain the best price offer on request to purchase a set of books at more than one bookstore. A web scraper prototype to obtain book prices and delivery terms from online bookstores is developed in Python using the Beautiful-Soup library. The two approaches to formation on the basis of these data of the order of books at the minimum possible price are suggested. The first approach is based on a generation of all possible combinations of orders, the second one uses path searching on a directed graph. Software implementation of both of them is added to the the web scraper functionality, which made it possible to eliminate the above-mentioned shortcoming and provide a qualitatively new service. Time characteristics of the productivity of the proposed approaches are analyzed under various hypothetical circumstances. Based on this analysis, recommendations are made for choosing a more effective approach depending on the data obtained. The input for the scraper is a list of books indicating the number of copies the user wants to purchase. As a result the user receives a set of URLs to purchase books. This URLs guarantees the minimum possible total cost of the order (taking into account the cost of delivery) at a fixed online bookstores. The output of the developed web scraper is given on the example of a test order. The data were taken from four bookstores whose technical services gave permission for automated processing of their information. Multithreading is used to reduce program execution time. Possible directions of expanding the functionality of the application and its using are also indicated.

Key words: information retrieving, web scraping, optimization, Python.