УДК 681.3: 658.56.

А.М. Глибовець, М.В. Кравченко

Програмна система побудови індексу для реалізації ранжування наукових документів

Описана программная система построения индекса для реализации ранжирования научных документов на украинском языке. На архитектурном уровне представлены разработанные модули системы для автоматического поиска документов, их обработки, восстановления структуры, получения необходимых для создания индекса цитирования данных и построение такого индекса.

The application system of index construction for the implementation of the scientific documents ranking in the Ukrainian language is presented. The detailed description of the implemented modules of the system for the automatic document searching, processing, structure recovery and extracting data from the text, needed for building citation index of such a collection of documents is proposed.

Описано програмну систему побудови індексу для реалізації ранжування наукових документів українською мовою. На архітектурному рівні представлено розроблені модулі системи для автоматичного пошуку документів, їх обробки, відновлення структури, отримання необхідних для створення індексу цитувань даних та побудова такого індексу.

Вступ. Пошукові системи та технології є одним з найактуальніших напрямів дослідження в сфері інформаційних технологій [1]. Багато відомих компаній асигнують величезні ресурси для покращення якості інформаційного пошуку для своїх користувачів. Одним з головних методів є ранжування документів у пошуковій видачі. Наразі до найпоширеніших пошукових запитів в мережі Інтернет належить пошук наукових статей за назвою або тематикою. Для отримання якісніших результатів залучається ранжування за допомогою індексу цитувань авторів.

Створено декілька успішних систем, які сприяють вирішенню цієї задачі, але вони орієнтовані на західні формати оформлення наукових документів і не можуть працювати з більшістю українських ресурсів.

Тому автори, проаналізувавши алгоритми пошуку, обробки документів для побудови індексу цитувань документів та автоматизацію процесів, пов'язаних з обробкою документів та побудови індексу цитувань, вирішили розробити *програмну* систему автоматизованої побудови індексу цитувань на вхідній колекції документів, яка буде корисною для реалізації алгоритму ранжування документів. Опис розробки системи становить основу цієї статті.

В процесі імплементації системи використовувався підхід модульності для полегшення розробки та тестування окремих компонент. При створенні модулів застосовано принципи розробки, що керується тестами (*Test-Driven Development*). Для реалізації модуля пошуку наукових документів використано модель акторів як метод забезпечення ефективного паралельного виконання задач. Розроблений та реалізований алгоритм дослідження структури *pdf*-документів, оформлених згідно з форматами українських видань, та пошуку елементів, що входять до списку посилань таких документів, став основою створеної системи автоматичної побудови індексу цитувань вхідної колекції наукових документів з україномовних джерел. Більшу частину програмного коду реалізовано на мові програмування *Scala* з використанням окремих бібліотек для мови *Java*.

Індекс цитувань

Це – реферативна база даних наукових публікацій, яка індексує посилання між публікаціями та дозволяє легко визначити, в яких більш нових документах цитують більш старі. Цей індекс є ключовим показником, що широко використовується у всьому світі для оцінки впливу вченого на світову науку.

Першою серйозною спробою створити базу індексу цитувань були цитування Шепарда (*Shepard's Citations*), датовані 1873 р. Він індексував базу юридичних творів. Пізніше, уже в 1960 р., заснований Юджином Гарфілдом, Ін-

ститут Наукової Інформації (*ISI*) оприлюднив перший індекс цитувань для статей, надрукованих в наукових журналах. Спочатку цей індекс був сформований для технічних наук (*Science Citation Index – SCI*), після нього був створений індекс цитувань для соціальних наук (*Social Sciences Citation Index – SSCI*) і останнім був створений індекс для художніх та гуманітарних наук (*Arts and Humanities Citation Index – AHCI*).

Першою автоматизованою системою для побудови індексу посилань стала система *CiteSeer*, випущена в 1997 р. Одним із найвідоміших проектів можна назвати проект *Google Scholar*, випущений в 2004 р. компанією *Google*.

Одним із застосувань таких індексів цитувань було визначення коефіцієнту впливовості журналів або окремих авторів. Найпростішим способом обчислення такого коефіцієнту є підрахунок сумарної кількості цитувань автора. Але у такого простого методу виявилося багато недоліків, тому з часом були запропоновані складніші методи обчислення коефіцієнту впливовості авторів: *h*-, *g*- та *i*10-індекси.

Модуль завантаження наукових документів

Для роботи з колекцією наукових документів необхідно знайти та завантажити їх з певних джерел. Тому найперше опишемо модуль системи, відповідальний за пошук за заданим ресурсом в мережі Інтернет, автоматичний пошук документів, які відповідають пошуковому запиту, та завантаження цих документів для подальшої обробки в інших модулях.

Оскільки алгоритм пошуку містить елементи, які можна легко розпаралелити, було вирішено використовувати одну з успішних сьогодні моделей паралельного програмування – модель акторів [2].

Модуль завантаження документів за вказаною адресою на вхід отримує кореневий ресурс, з якого необхідно починати пошук, в результаті роботи знаходить та завантажує знайдені ресурси, які відповідають сконфігурованому критерію. Такий пошуковий рушій мусить не тільки обробити кореневий ресурс, але й виокремити потенційні ресурси, що можуть містити ресурси, відповідні пошуковому запиту, та опрацювати їх. Цей процес має продовжуватися, поки не буде досягнуто максимальної глибини пошуку, яку теж можна конфігурувати. Всі конфігураційні параметри зберігатимуться в окремому конфігураційному файлі, який можна буде легко змінювати. Синтаксис визначення критеріїв відповідатиме синтаксису регулярних виразів.

Модуль реалізовано за допомогою мови *Scala* з використанням бібліотеки акторів *Akka*.

Основними елементами архітектури є такі актори: *Реєстратор*, *Керівник*, *Пошукач*. Ці актори взаємодіють за допомогою відправлення асинхронних повідомлень.

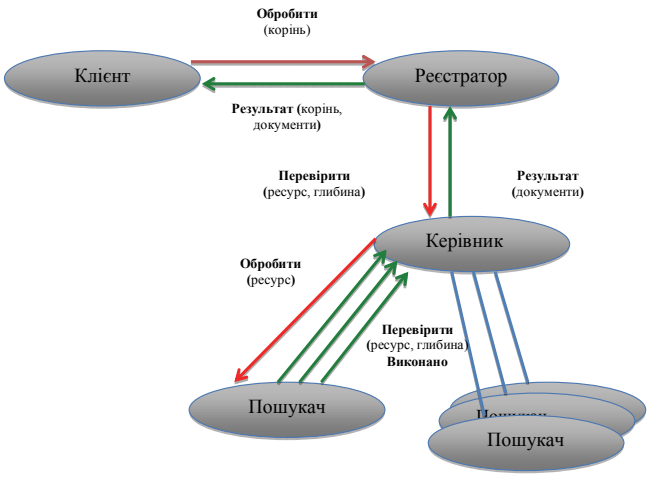
Наведемо опис взаємодії акторів в системі:

- *Клієнт* створює актора *Реєстратор* та надсилає йому повідомлення *Обробити*(*кореневий_ресурс*).
- *Реєстратор* ставить цей запит у чергу виконання; коли надходить черга запиту, то *Реєстратор* створює актора *Керівник* та надсилає йому повідомлення *Перевірити*(*кореневий_ресурс*, *максимальна_глибина*).
- Актор *Керівник* запам'ятовує ресурс *кореневий_ресурс* та створює актора *Пошукач* і передає йому повідомлення *Обробити*(*ресурс*), делегуючи йому обробку цього ресурсу.
- Актор *Пошукач* обробляє ресурс, знаходить «цікаві» ресурси та для кожного з них надсилає актору *Керівник* повідомлення *Перевірити*(*ресурс*, *глибина*), коли ресурс повністю оброблений, він надсилає актору *Керівник* повідомлення *Виконано*;
- кожне повідомлення *Перевірити*(*ресурс*, *глибина*) актор *Керівник* обробляє подібно до кореневого ресурсу; якщо глибина дорівнює нулю, то *Керівник* не генерує задачі *Пошукач* для обробки цих ресурсів; ця перевірка необхідна, щоб не зайти в нескінечнний цикл обробки; якщо ресурс задовольняє критерію завантаження, він зберігається в акумуляторі результатів;
- коли актор *Керівник* отримує повідомлення *Виконано*, він відзначає, що ресурс повністю оброблено;
- коли всі актори *Пошукачі* відправили повідомлення *Виконано* і всі ресурси відзначені

як оброблені, то *Керівник* вважає пошук завершеним та повертає збережені в акумуляторі результати актора *Реєстратор* за допомогою повідомлення *Результат*(*документи*);

• актор *Реєстратор* в свою чергу надсилає клієнтові повідомлення *Результат*(*кореневий ресурс*, *документи*) та переходить до обробки наступного запиту, беручи його з черги).

Діаграму взаємодії наведено на рисунку.



В системі знаходиться тільки один актор класу *Реєстратор* та один актор типу *Контролер*. В той же час акторів класу *Пошукач Імен* може бути багато, адже обробка одного ресурсу незалежна від обробки іншого і може виконуватися паралельно.

Тут відслідковується одна з важливих концепцій програмування за допомогою акторів: в задачі потрібно виокремити такі підзадачі, які можна було б виконувати незалежно одну від одної. Тоді на кожну підзадачу можна виділити власного актора, завданням якого буде розв'язати її та якось повідомити про результат роботи. Для обробки таких результатів рекомендовано виділяти в архітектурі окремих акторів.

Модуль обробки *pdf*-документів

Модуль використовує отримані в результаті завантаження документи, виконуючи наступні дії: зчитування та перетворення документа у зручну для обробки структуру даних; виокремлення структурних елементів документа (наприклад, колонок); пошук необхідних для алгоритму побудови індексу цитувань даних. Розглядаються документи формату *pdf*, адже вони є стандартним форматом для збереження наукових документів в електронній формі.

Реалізація зчитування документа у форматі pdf. Видобування текстової інформації та структури з документів у форматі *pdf* залишається складною задачею, в основному через дизайн стандарту *PDF*, орієнтованого виключно на відображення документа. Таке представлення ускладнює задачу доступу до структурованої та цілісної інформації в *pdf*-документі. Хоча за допомогою деяких бібліотек можна отримати текст документа, але найчастіше в процесі видобування тексту втрачається структура документа.

При використанні низькорівневих операторів можна отримати набір елементів документа з різними властивостями: форматування, розміщення (у вигляді координат на сторінці), тип шрифту тощо. Але така обробка не дає інформації про вищі абстракції текстових документів, таких, наприклад, як слова, рядки або колонки. Якщо розглянути структуру *pdf*-документа, то роздільники між словами, реченнями та колонками подані у вигляді пробілів (*whitespace*), певне слово, представлене у вигляді текстових елементів необов'язково однорідної структури (тобто дві перші літери можуть бути одним елементом, всі інші – наступним). Цієї інформації досить для правильного відображення парсером на екрані, але зовсім мало для автоматичного видобування тексту.

В науковій літературі та періодиці часто зустрічаються документи, текст яких розділений на дві колонки. Така структура також ускладнює роботу можливих аналізаторів через те, що не існує чітких правил, які б визначали, в якому порядку дані згруповані не фізично, а логічно.

Для полегшення роботи над *pdf*-документами була використана бібліотека з відкритим кодом *The Apache PDFBox*^{TM}, написана на мові *Java*, що дозволяє створювати нові документи, маніпулювати існуючими та діставати текстову і графічну інформацію з документів.

Для отримання тексту з *pdf*-документа використовується клас *PDFTextStripper* [3], який приймає на вхід документ та повертає на виході

текст, який знаходиться в документі. Цей клас всередині під час обробки документа може виділяти слова та неперервні лінії тексту в документі за місцем їх розташування на сторінці та отримувати інформацію про шрифт тексту, графічні властивості сторінки тощо. Ця інформація використовується в алгоритмі виокремлення структури документів, який буде описано далі.

Виокремлення структурних елементів документа. Для знаходження структурних елементів в документі, наприклад, роздільник між колонками або початок нового блоку тексту, необхідно знайти місця між текстом, які за розміром відрізняються від стандартних пробілів між символами, словами або рядками. Така задача відома і називається в геометричному аналізі структури документів *задачею пошуку максимального порожнього прямокутника*.

Задача формулюється так.

Нехай є набір прямокутників $C = \{r_0, \ldots r_n\}$ на площині, кожен з яких перебуває в межах певного обмежувального прямокутника r_b. В структурному аналізі документів, зазвичай r_i відповідає прямокутникам, які «оточують» певні зв'язні компоненти, наприклад, слова або неперервні лінії тексту, а загальним обмежувальним прямокутником r_b є вся сторінка.

Визначається оціночна функція для прямокутників $Q: \boldsymbol{R}^4 \to \boldsymbol{R}$, що для двох прямокутників r та r' задовольняє умову:

$$r \subseteq r' \Rightarrow Q(r) \le Q(r') \qquad (1)$$

У загальному випадку як функцію Q можна взяти площу прямокутника, яка задовольняє умову (1).

Задачею пошуку максимального порожнього прямокутника є пошук такого прямокутника $\hat{r} \subseteq r_b$, який максимізував би $Q(T)$ серед усіх можливих прямокутників $r \subseteq r_b$, де r не перетинається з жодним з прямокутників множини C. Також це можна виразити математичним рівнянням:

$$\hat{r} = \hat{r}(C, r_b, Q) = \arg\max_{r \in U} Q(r),$$

де $U = \left\{ r \subseteq r_b \middle| \forall c \in C : r \cap c = \varnothing \right\}$.

Існує кілька алгоритмів для розв'язання задачі пошуку максимального порожнього прямокутника в обчислювальній геометрії [4], але вони надто складні у реалізації і не знайшли широкого вжитку. Тому автори скористалися алгоритмом, представленим у [5].

Пошук необхідних для алгоритму побудови індексу цитувань даних. На цьому етапі роботи системи з *pdf*-документа були відокремлені неперервні лінії тексту та структура документа (колонки, межі блоків тощо). За цими даними система може знайти текст у кожному структурному елементі. Отже, система має всі можливості для пошуку необхідних для побудови індексу цитувань даних.

Майже у кожній науковій статті подано релевантні та використані роботи. Такий розділ оформлюється у відповідності до певних вимог. Автоматичний пошук списку посилань у документі має не тільки розпізнати розділ з посиланнями, але й так інтерпретувати результат, щоб не втратити важливої інформації.

Існують рішення для пошуку посилань у документі [6, 7], але вони мають багато недоліків, пов'язаних з орієнтуванням на західні формати *ADS*, *BIB*, *EndNote*, *WordBib* оформлення електронних наукових документів.

Для пошуку списку посилань [8, 9] використовується відокремленість від іншого тексту за допомогою певних зв'язувальних слів, наприклад, *список посилань* (*References*), *Бібліографія* (*Bibliography*), *список цитованих джерел* (*Literature Cited*) тощо. Після розгляду більшості запропонованих рішень стає очевидним, що їх розробники проектували системи з урахуванням потрапляння на вхід документів англійською мовою зі списком посилань також англійською мовою. Багато таких рішень мають закритий код, що не дозволяє їх розширювати або відлагодити для власних цілей.

Оскільки стає неможливим використання жодного з перелічених можливих готових сервісів, було вирішено для розв'язання задачі виокремлення структурних елементів посилання використати бібліотеку *OpenNLP* [10] з відкритою ліцензією.

Бібліотека *OpenNLP* являє собою набір інструментів для обробки текстів, написаних природною мовою. Основними можливостями бібліотеки є токенізація, пошук речень в документах, пошук в реченнях частин мови та іменованих сутностей в документах. Для обробки тексту всередині бібліотеки використовуються механізми машинного навчання – навчання на базі максимальної ентропії та за допомогою персептрона.

За допомогою бібліотеки *OpenNLP* розв'язано задачу виокремлення іменованих сутностей зі списку посилань – автор та назва статті тощо. Як приклад роботи з бібліотекою розглянемо *шукач імен* (*Name Finder*), який виконує функції розпізнавання іменованих сутностей (*Name Entity Recognition*) [11].

Для його роботи необхідно завантажити спеціальну базу правил, яка називається *моделлю*. Вона залежить від мови та типу іменованих сутностей, для яких ця модель була натренована (навчена).

Проект *OpenNLP* пропонує набір готових моделей. Для випадків, коли готова модель недоступна для бажаної мови, не здатна знаходити бажані сутності або має низьку швидкодію, бібліотека *OpenNLP* дозволяє провести тренування для розпізнавача на власній навчальній базі.

Для створення власної моделі для виокремлення необхідних даних з посилань створено навчальну базу з наступною структурою записів:

1. ***<START:person>*** *Непеивода Н. Н.* ***<END>*** ***<START:title>*** *Прикладная логика* ***<END>***/ ***<START:person>*** *Н.Н. Непеивода* ***<END>***. – *Новосибирск* : *НГУ*, 2000. – 521 *с.*

2. ***<START:title>*** *Модели и алгоритмы оптимизации надежности сложных систем* ***<END>***/***<START:person>****В. Л. Волкович* ***<END>***, ***<START:person>*** *А. Ф. Волошин* ***<END>***, ***<START:person>*** *В. А. Заславский* ***<END>***, ***<START:person>*** *И. А. Ушаков* ***<END>*** ; – *К.* : *Наукова думка*, 1992. – 312 *с.*

За допомогою такої навчальної бази проводилося тренування Пошукача Імен для наступних типів: *person* (автор використаного ресурсу) та *title* (назва самого використаного джерела). Слід також зазначити, що розглянуто кілька форматів посилань: 1. – спочатку визначає автора, потім назву статті, далі знов автора, в той час як в 2. – перед назвою статті немає визначення автора.

Розробка модуля побудови та збереження індексу цитувань

Як видно з опису минулих модулів, процес пошуку посилань у документах є трудомісткою задачею. Якщо кожного разу при запиті користувача або програмного інтерфейсу на отримання певної частини такого індексу перебудовувати весь індекс, затримка у пошуковій видачі буде дуже велика. Тому постає необхідність у збереженні знайдених даних у певному сховищі.

При аналізі структури даних можна впевнитись, що вони утворюють собою орієнтований граф певної структури: вершинами є автори, а дугами між ними виступають цитування, представлені у певній узагальненій структурі. Зазначимо, що в такому графі можливі цикли, оскільки автори можуть цитувати власні роботи.

Нехай автор A в своєму творі $S1$ посилається на статтю $S2$ автора B. Тоді збереження такого зв'язку у графі може бути подано так: A та B – є вершинами у графі; з вершини A у вершину B виходить дуга E, значенням якої є структура S, що складається з двох елементів: з роботи, у якій використовується посилання ($S1$), і роботи, на яку йде посилання ($S2$).

Слід також зазначити, що у випадку, коли автор цитує роботу, написану кількома авторами, для кожного автора цитованої роботи створюється дуга з автором, який її цитує. Це полегшує створення індивідуального індексу цитувань для кожного автора, у порівнянні з можливим варіантом створення окремого вузла, який представлятиме групу авторів.

Отже, для збереження такої структури даних у сховищі (базі даних), треба визначитися з доцільним типом бази даних.

Найпопулярнішими рішеннями сьогодні є реляційні бази даних, нереляційні (*NoSQL*) та графові бази даних. Через необхідність збереження даних, які характеризуються зв'язками та мають схожу на граф структуру, все частіше популярними стають графові бази даних.

Зв'язки в графі у природний спосіб формують дуги. Запит до таких баз даних являє собою обхід графу по певних дугах. Через те, що модель даних подається у вигляді, орієнтованому на графове представлення, внутрішні рушії бази даних можуть використовувати ці властивості для оптимізації збереження, обробки та запитів до таких даних.

Серед реалізацій графових баз даних, доступних для мови *Java*, виділяють *AllegroGraph*, *HyperGraphDB*, *InfiniteGraph*, *OrientDB*, *Titan* та *Neo4j*. Серед них обрано базу даних *Neo4j*, яка наразі є найбільш популярною та активно доопрацьовується (останню версію випущено в квітні 2014 р.).

Для роботи з базою даних *Neo4j* використано компонент бібліотеки *Spring* під назвою *Spring Data Neo4j* – проект, який є модулем *Spring Data*, що надає зрозумілий та легкий програмний інтерфейс для доступу до графової бази даних *Neo4j* [12].

Висновки. Багато зусиль та коштів вкладається у створення автоматичних або принаймні напівавтоматичних систем побудови індексів цитувань вхідної колекції документів. Існує багато зарубіжних сервісів для індексування наукових документів. Основним недоліком таких сервісів є низька підтримка україномовних видань, що є наслідком їх орієнтування на зовсім інакші правила оформлення наукових документів.

У даній статті представлена архітектура та методи для розв'язання задачі побудови індексу для реалізації ранжування наукових документів українською мовою. Серед основних виконаних задач можна виділити такі: досліджено можливі варіанти для побудови індексу цитувань; представлено детальну архітектуру компонента, що відповідає за пошук документів за певним ресурсом та завантаження їх для подальшої обробки; досліджено методи роботи над *pdf*-документами для розв'язання задачі відновлення структури та даних з них; розглянуто можливості пошуку та обробки необхідних даних у документі, наприклад, списку використаних джерел; запропоновано підхід для збереження побудованого індексу цитувань у базі даних з урахуванням ефективності та можливого масштабування; наведено аргументацію обраних рішень для забезпечення виконання кожної задачі.

Використано велику кількість прогресивних методів архітектури програмних систем: від моделі акторів до обробки природних текстів за допомогою механізмів машинного навчання. Результат досліджень підтверджує можливість реалізації автоматизованої системи побудови індексу цитувань для україномовних видань.

1. *Структурированные* данные и семантическая паутина: технологии *Wiki* / А.Н. Глибовец, Н.Н. Глибовец, Д.Е. Покопцев и др. // Проблемы программирования. – 2013. –№ 1. – С. 45–67.
2. *William D. Clinger.* Foundations of Actor Semantics. – http://dspace.mit.edu/bitstream/handle/1721.1/6935/AITR-633.pdf
3. *Apache* PDFBox 1.8.5 API / Apache PDFBox docs. – http://pdfbox. apache.org/docs/1.8.5/javadocs/
4. *Orlowski M.* A new algorithm for the largest empty rectangle problem. – http://link.springer.com/article/10.1007%2FBF01840377
5. *Thomas M. Breuel.* Two Geometric Algorithms for Layout Analysis. – http://citeseerx.ist.psu.edu/viewdoc/summary? doi=10.1.1.17.5758
6. *References* Extractor / High-Energy Physics Literature Database – https://inspirehep.net/textmining/
7. *ParsCit*: An open-source CRF Reference String and Logical Document Structure Parsing Package. – http:// aye.comp. nus.edu.sg/parsCit/
8. *Donna Bergmark.* Automatic Extraction of Reference Linking Information from Online Documents. – http:// www.cs.cornell.edu/cdlrg/reference%20linking/extraction.pdf
9. *Roman Kern.* Extraction of References Using Layout and Formatting Information from Scientific Articles. – http://www.dlib.org/dlib/september13/kern/09kern.html
10. *OpenNLP* documentation / Apache OpenNLP. – https:// opennlp.apache.org/documentation.html
11. *Name* Finder / Apache OpenNLP documentation. – http:// opennlp.apache.org/documentation/manual/opennlp.html #tools.namefind
12. *Jonas Partner*, *Aleksa Vukotic*, *Nick Watt* Neo4j in Action. – Manning, 2012 – P. 6–11.

Поступила 18.07.2014
Тел. для справок: +38 067 409-4355, 093 447-6960
E-mail: *andriy@glybovets.com.ua*;
mike.kravchenko.ua@gmail.com.

А.Н. Глибовец, М.В. Кравченко

Программная система построения индекса для реализации ранжирования научных документов

Введение. Поисковые системы и технологии – одно из самых актуальных направлений исследований в области информационных технологий [1]. Многие известные компании ассигнуют огромные ресурсы для повышения качества информационного поиска для своих пользователей. Одним из главных методов такого улучшения можно считать ранжирование документов в поисковой выдаче. К самым распространенным поисковым запросам в сети Интернет в настоящее время относится поиск научных статей на определенную тематику или с определенным названием. Чтобы получить более качественные результаты, применяется ранжирование с помощью индекса цитирования авторов.

Создано несколько успешных систем, способствующих решению этой задачи, но они ориентированы на западные форматы оформления научных документов и не работают с большинством украинских ресурсов.

Поэтому авторы, проанализировав алгоритмы поиска, обработки документов для построения индекса цитирования документов и автоматизации процессов, связанных с обработкой документов и построением индекса цитирования, решили разработать *программную* систему автоматизированного построения индекса цитирований коллекции документов, которую можно будет применить к реализации алгоритма ранжирования документов. Описание разработки системы – основа данной статьи.

В процессе имплементации системы использовался подход модульности ради упрощения разработки и тестирования отдельных компонент. Для создания модулей применялись принципы разработки через тестирование (*Test-Driven Development*). Для реализации модуля поиска научных документов использована модель актеров как метод гарантирования эффективного параллельного выполнения задач. Разработанный и реализованный алгоритм исследования структуры *pdf*-документов, оформленных в соответствии с требованиями украинских издательств, и поиска элементов, входящих в список ссылок на источники, стал основой созданной системы автоматического построения индекса цитирования входной коллекции документов из украиноязычных источников. Большая часть программного кода реализована на языке программирования *Scala* с использованием некоторых библиотек для языка *Java*.

Индекс цитирования

Это – реферативная база данных научных публикаций, которая индексирует ссылки между публикациями и позволяет легко определить, в каких более новых документах цитируются более старые. Это – ключевой показатель, используемый во всем мире для оценки влияния ученого на мировую науку.

Первой серьезной попыткой создать базу индекса цитирования стали цитирования Шепарда (*Shepard's Citations*), датируемые 1873 г. Он осуществил индексацию базы юридических документов. Позже, в 1960 г., основанный Юджином Гарфилдом Институт Научной Информации (*ISI*) представил общественности первый индекс цитирования статей, напечатанных в научных журналах. Сначала он был сформирован для технических наук (*Science Citation Index – SCI*), затем расширен индексом цитирования для социальных наук (*Social Sciences Citation Index – SSCI*) и для художественных и гуманитарных наук (*Arts and Humanities Citation Index – AHCI*).

Первой автоматизированной системой для построения индекса цитирования считается выпущенная в 1997 г. система *CiteSeer*. Одним из самых известных проектов стал *Google Scholar*, выпущенный в 2004 г. компанией *Google*.

Одним из применений таких индексов цитирования было определение коэффициента влияния журналов или отдельных авторов. Самый простой способ получения такого коэффициента – вычисление общего количества цитирования автора. Но у такого метода оказалось много недостатков, поэтому со временем были предложены более сложные методы определения коэффициента влиятельности авторов, такие, например, как *h*-, *g*- и *i*10-индексы.

Модуль загрузки научных документов

Для работы с коллекцией научных документов необходимо сначала найти и загрузить их с некоторых источников. Поэтому первым будет описан модуль системы, ответственный за поиск на заданном ресурсе в сети Интернет, автоматический поиск документов, соответствующих поисковому запросу, и загрузку этих документов для последующей обработки в других модулях.

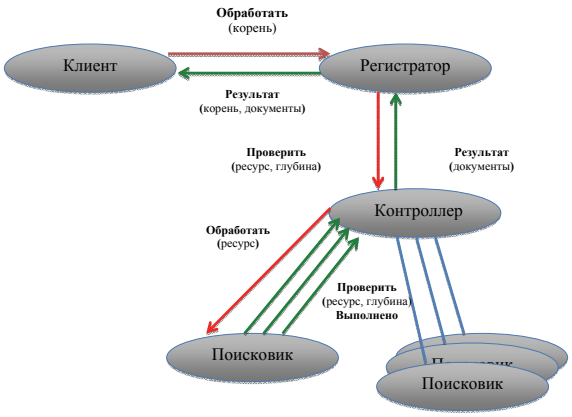
Поскольку алгоритм содержит элементы, которые можно легко распараллелить, авторы решили использовать одну из самых успешных моделей параллельного программирования – модель актеров [2].

Модуль загрузки документов по указанному адресу получает на вход корневой ресурс, с которого необходимо начать поиск, и который в ходе работы находит и загружает найденные ресурсы, соответствующие некоторому заданному критерию. Такой поисковый движок должен не только обработать корневой ресурс, но и сначала выделить, а затем обработать потенциальные ресурсы, которые могут содержать соответствующие поисковому запросу ресурсы. Такой процесс должен продолжаться до тех пор, пока не будет достигнута максимальная глубина поиска, которая также должна быть задана для предотвращения бесконечной работы программы. Все конфигурационные параметры считываются с отдельного файла, который легко можно изменять. Для определения параметров в таком файле используется синтаксис регулярных выражений.

Модуль был реализован на языке программирования *Scala* с использованием библиотеки актеров *Akka*. Основными элементами архитектуры являются следующие актеры: *Регистратор, Контроллер и Поисковик.* Эти актеры взаимодействуют путем обмена асинхронных сообщений. Приведем протокол взаимодействия актеров в системе:

- *Клиент* создает актера *Регистратор* и отправляет ему сообщение *Обработать*(*корневой_ресурс*).
- *Регистратор* ставит этот запрос в очередь выполнения; когда очередь подходит к запросу, *Регистратор* создает актера *Контроллер* и отправляет ему сообщение *Проверить*(*корневой_ресурс*, *максимальная_глубина*).
- Актер *Контроллер* запоминает ресурс *корневой_ресурс*, создает актера *Поисковик* и отправляет ему сообщение *Обработать*(*ресурс*), тем самым делегируя ему реальную обработку этого ресурса.
- Актер *Поисковик* обрабатывает ресурс, находит «интересные» ресурсы и для каждого из них отправляет актеру *Контроллер* сообщение *Проверить* (*ресурс*, *глубина*); когда ресурс будет полностью обработан, *Поисковик* отправляет актеру *Контроллер* сообщение *Выполнено*;
- каждое сообщение *Проверить* (*ресурс*, *глубина*) актер *Контроллер* обрабатывает похожим способом, как и корневой ресурс; если глубина равна нулю, то *Контроллер* больше не генерирует задачи *Поисковику*; эта проверка нужна, чтобы не зайти в бесконечный цикл обработки; если ресурс удовлетворяет критерию загрузки, он сохраняется в аккумуляторе результатов;
- когда актер *Контроллер* получает сообщение *Выполнено*, он считает ресурс полностью обработанным;
- когда все актеры *Поисковики* отправили сообщение *Выполнено* и все ресурсы отмечены как обработанные, *Контроллер* считает поиск завершенным и возвращает накопленные в аккумуляторе результаты актеру *Регистратор* с сообщением *Результат* (*документы*);
- актер *Регистратор* в свою очередь отправляет клиенту сообщение *Результат*(*корневой_ресурс*, *документы*) и переходит к обработке следующего запроса, получая его из очереди.

Диаграмма взаимодействия приведена на рисунке.



В системе в одно и то же время находится только один актер класса Регистратор и один актер класса Контроллер. В то же время актеров класса Поисковик может быть много, поскольку обработка одного ресурса не зависит от обработки другого и может происходить параллельно.

Здесь прослеживается одна из самых важных концепций программирования с помощью актеров: в задаче необходимо выделить такие подзадачи, которые можно было бы решать независимо одну от другой. Тогда на каждую подзадачу можно выделить собственного актера, который будет решать ее и в конце как-то сообщить результат работы. Для обработки таких результатов рекомендуется выделять в архитектуре отдельных актеров.

Модуль обработки *pdf*-документов

Модуль использует документы, полученные в результате загрузки, выполняя следующие действия: считывание документов и преобразование в удобную для обработки структуру данных; выделения структурных элементов документа (например, колонок); поиск необходимых для алгоритма построения индекса цитирования данных. Рассматриваются документы в формате *pdf*, поскольку они есть стандартным форматом для сохранения научных документов в электронной форме.

Реализация считывания документа в формате pdf. Добывание текстовой информации и структуры из документов в формате *pdf* остается и теперь сложной задачей, так как дизайн стандарта *PDF* ориентирован исключительно на визуальное отображение документа. Такое представление ощутимо усложняет задачу доступа к структурированной и целостной информации в *pdf*-документе. Хотя с помощью некоторых библиотек и можно получить текст документа, но чаще всего во время получения текста нарушается структура документа.

При использовании низкоуровневых операторов можно получить набор элементов документа с разными свойствами: форматирование, размещение (координаты на странице), тип шрифта и др. Но такая обработка не дает информации о более высоких абстракциях текстовых документов, таких как, например, слова, строки или колонки. При рассмотрении структуры *pdf*-документа можно обнаружить, что разделители между словами, предложениями и колонками обозначены пробелами (*whitespace*), слова – текстовыми элементами необязательно однородной структуры (две первых буквы могут быть одним элементом, а все другие – следующим). Такой информации достаточно для правильного отображения парсером на экране, но совсем недостаточно для автоматического получения текста.

В научной литературе и периодике часто встречаются документы, текст которых разделен на две колонки. Такая структура также усложняет работу возможных анализаторов, так как нет четких правил, которые определяли бы, в каком порядке данные сгруппированы логически, а не физически.

Для упрощения работы с *pdf*-документами была использована написанная на языке *Java* библиотека с откры-

тым исходным кодом *The Apache PDFBox* ^{TM}. Она позволяет создавать новые документы, манипулировать существующими и добывать текстовую и графическую информацию из них.

Для получения текста из *pdf*-документа использован класс *PDFTextStripper* [3], который принимает на вход документ и возвращает на выходе текст документа. Этот класс внутри во время обработки может выделять слова и непрерывные линии текста в документе по местам их размещения на странице и получать информацию о шрифте текста, графических характеристиках страницы и пр. Эта информация используется в алгоритме выделения структуры документа, который будет описан далее.

Выделение структурных элементов документа. Чтобы найти такие элементы в документе, например, разделитель между колонками или начало нового блока текста, необходимо обнаружить такие места, которые по размеру отличаются от стандартных пробелов между символами, словами или строками. Эта задача известна и называется в области геометрического анализа структуры документов *задачей поиска максимального пустого прямоугольника*.

Задача формулируется так.

Пусть на площади дан набор прямоугольников $C = \{r_0,\dots r_n\}$, каждый из которых находится в области некоторого ограничивающего прямоугольника r_b. В структурном анализе документов обычно r_i – прямоугольник, который «окружают» некоторые связные компоненты, например, слова или непрерывные линии текста, а общим ограничивающим прямоугольником r_b есть вся страница.

Определяется оценочная функция Q для прямоугольников $Q : \boldsymbol{R}^4 \to \boldsymbol{R}$, которая для двух прямоугольников r и r' удовлетворяет условию:

$$r \subseteq r' \Rightarrow Q(r) \le Q(r') . \qquad (1)$$

В общем случае в качестве функции Q можно использовать функцию вычисления площади прямоугольника, которая соответственно будет удовлетворять условию (1).

Задачей поиска максимального пустого прямоугольника есть поиск такого прямоугольника $\hat{r} \subseteq r_b$, который максимизировал бы $Q(T)$ для всех возможных прямоугольников $r \subseteq r_b$, где r не пересекается ни с одним из прямоугольников множества C. Это можно выразить математическим равенством

$$\hat{r} = \hat{r}(C, r_b, Q) = \arg\max_{r \in U} Q(r) ,$$

где $U = \left\{ r \subseteq r_b \,\middle|\, \forall c \in C : r \cap c = \varnothing \right\}$.

В вычислительной геометрии существует несколько алгоритмов для решения задачи нахождения максимального пустого прямоугольника [4], но они слишком сложны в реализации и не нашли широкого использования. Поэтому авторы воспользовались алгоритмом, предложенным в [5].

Поиск необходимых для алгоритма построения индекса данных. На этом этапе работы системы из *pdf*-документа были выделены непрерывные линии текста и структура документа (колонки, границы между текстовыми блоками и др.). С этими данными система получает возможность найти текст, который находится в каждом структурном элементе. Соответственно система может начать поиск необходимых данных для построения индекса цитирования.

Почти в каждой научной статье есть раздел, где перечисляются все релевантные и использованные работы. Такой раздел оформляется в соответствии с некоторыми требованиями. Автоматический поиск списка ссылок в документе должен не только распознать раздел со ссылками, но и интерпретировать результат так, чтобы не утратить важной информации.

Существуют решения для поиска ссылок в документе [6, 7], но в них есть недостатки, связанные с ориентированием на западные форматы *ADS*, *BIB*, *EndNote*, *WordBib*.

Для поиска списка ссылок [8, 9] используется обособленность от другого текста посредством определенных связующих слов, например, *список ссылок* (*References*), *Библиография* (*Bibliography*), *список цитированных источников* (*Literature Cited*) и пр. После рассмотрения большинства предлагаемых решений становится очевидным, что их разработчики проектировали системы с учетом попадания на вход документов на английском языке со списком ссылок также на английском языке. Множество таких решений имеют закрытый код, что не позволяет их расширить или отладить для собственных целей.

Поскольку становится невозможно использовать ни одного из предполагаемых сервисов, было решено воспользоваться для решения задачи выделения структурных элементов ссылки библиотекой с открытой лицензией *Apache OpenNLP* [10].

Библиотека *OpenNLP* представляет собой набор инструментов для обработки текстов, написанных природным языком. Основные возможности библиотеки – токенизация, поиск предложений в документах, нахождение в предложениях частей языка, а так же именованных сущностей в документах. Для обработки текста внутри библиотеки используются механизмы машинного обучения, такие как обучение на основе максимальной энтропии и с помощью персептрона.

Посредством библиотеки *OpenNLP* решена задача выделения именованных сущностей из списка ссылок – *автор статьи* и *название*. Как пример работы с библиотекой будет рассмотрен *поисковик имен* (*Name Finder*), распознающий именованные сущности (*Name Entity Recognition*) [11].

Для его работы необходимо загрузить специальную базу правил – модель. Последняя зависит от языка и типа именованных сущностей, для которых эта модель была обучена.

Проект *OpenNLP* предлагает набор готовых моделей. В случае, когда готовая модель недоступна для нужного языка, не способна находить нужные сущности или имеет низкое быстродействие, библиотека *OpenNLP*

позволяет провести обучение для распознавателя на собственной учебной базе.

Для создания собственной модели выделения необходимых данных со ссылки создана учебная база со следующей структурой записей:

1. ***<START:person>*** *Непеивода Н. Н.* ***<END>*** ***<START:title>*** *Прикладная логика* ***<END>*** / ***<START:person>*** *Н. Н. Непеивода* ***<END>*** . *– Новосибирск* : *НГУ*, 2000. – 521 *с.*

2. ***<START:title>*** *Модели и алгоритмы оптимизации надежности сложных систем* ***<END>*** / ***<START: person>*** *В. Л. Волкович* ***<END>*** , ***<START:person>*** *А. Ф. Волошин* ***<END>*** , ***<START:person>*** *В. А. Заславский* ***<END>*** , ***<START:person>*** *И. А. Ушаков* ***<END>*** ;– *К.* : *Наукова думка*, 1992. – 312 *с.*

С помощью такой учебной базы проводилось обучение Поисковика Имен для таких типов сущностей: *person* (автор использованного источника) и *title* (название использованного источника). Следует отметить, что рассматривалось несколько форматов цитирования: 1. – сначала идет автор, потом – название статьи, затем – снова автор, в то время, как в 2. – перед названием статьи нет определения автора.

Разработка модуля построения и сохранения индекса цитирований

Как видно из описания предыдущих модулей, процесс поиска ссылок в документах представляет собой трудоемкую задачу. Если каждый раз при запросе пользователя или программного интерфейса на получение некоторой части такого индекса перестраивать весь индекс, то задержка в поисковой выдаче будет велика. Поэтому возникает необходимость хранения найденных данных в некотором хранилище.

При анализе возможной структуры данных можно убедиться, что цитирования создают ориентированный граф определенной структуры: узлы – это авторы, а ребра между ними – цитирования, представленные в некоторой обобщенной форме. Следует отметить, что в таком графе возможны циклы, поскольку авторы могут цитировать собственные работы.

Пусть автор A в своей статье $S1$ цитирует статью $S2$ автора B. Тогда сохранение такой связи в графе может быть представлено следующим образом: A и B – узлы графа; узлы A и B связаны направленным ребром E, значение которого – структура S, состоящая из двух элементов: первый – работа, в которой используется ссылка ($S1$), второй – работа, на которую ссылаются ($S2$).

Отметим также, что в случае, когда цитируют работу, написанную несколькими авторами, то для каждого автора цитируемой работы создается связь с автором, цитирующим ее. Это облегчает создание индивидуального индекса цитирований для каждого автора, в сравнении с возможным вариантом создания отдельного узла, который представлял бы группу авторов.

Поэтому для хранения такой структуры данных в определенном хранилище (базе данных) необходимо определится с типом базы данных, которую уместно было бы использовать.

Самые популярные решения сегодня – это реляционные базы данных, не-реляционные базы данных (*NoSQL*) и графовые базы данных. В силу необходимости хранения данных, которые характеризируются связями и имеют похожую на граф структуру, все более популярными становятся графовые базы данных.

Связи в графе естественно представлены ребрами. Запросы к таким базам данных – это обход графа по определенным путям. Поскольку модель данных представлена в ориентированном на графовое представление виде, внутренние механизмы баз данных могут использовать эти свойства для оптимизации хранения, обработки и запросов к таким данным.

Среди реализаций графовых баз данных, доступных для языка *Java*, выделяют *AllegroGraph*, *HyperGraphDB*, *InfiniteGraph*, *OrientDB*, *Titan* и *Neo*4*j*. Из них авторы выбрали *Neo*4*j*, которая пользуется самой большой популярностью и активно разрабатывается (последняя версия выпущена в апреле 2014 г.).

Для работы с базой данных *Neo*4*j* использован компонент библиотеки *Spring* – *Spring Data Neo*4*j*, предоставляющий понятный программный интерфейс для доступа к графовой базе данных *Neo*4*j* [12].

Заключение. Много средств и усилий вкладывается в создание автоматических или, по крайне мере, полуавтоматических систем создания индексов цитирования коллекции документов. Существует множество зарубежных сервисов для индексирования научных документов. Основной недостаток таких сервисов – низкая поддержка украиноязычных изданий.

В данной статье представлена архитектура и методы решения задачи построения индекса для реализации ранжирования научных документов на украинском языке. Среди основных решенных задач можно выделить следующие: исследование возможных вариантов для построения индекса цитирования; подача подробной архитектуры компонента, ответственной за поиск документов на некотором ресурсе и их загрузку для последующей обработки; исследование методов работы с *pdf*-документами для решения задачи восстановления структуры данных; рассмотрение возможности поиска и обработки необходимых данных в документе, например, списка использованных источников; предложение подхода к сохранению построенного индекса цитирований в базе данных с учетом эффективности и возможного масштабирования; аргументация принятых решений для обеспечения выполнения каждой задачи.

Использовано большое количество прогрессивных методов архитектуры программных систем: от модели актеров до обработки текстов на естественном языке с помощью механизмов машинного обучения. Результат исследований подтверждает возможность реализации автоматизированной системы построения индекса цитирований для украиноязычных изданий.