УДК: 005:37

Андрій Олександрович Білощицький
Доктор технічних наук, завідувач кафедри інформаційних технологій
Олександр Васильович Діхтяренко
Аспірант кафедри основ інформатики

Київський національний університет будівництва і архітектури, Київ

НЕЧІТКИЙ ПОШУК В ДОКУМЕНТІ З ВРАХУВАННЯМ МОРФОЛОГІЧНИХ ОСОБЛИВОСТЕЙ

Означено і розглянуто проблему нечіткого пошуку в текстах. Проблему поділено на основні складові елементи і запропоновано рішення по кожному. Також описано загальні техніки та засоби розв'язання таких задач. На основі рішення конкретних завдань в межах проблеми нечіткого пошуку запропоновано алгоритм реалізації, який базується на використанні стемінгу, перестановок та регулярних виразів.

Ключові слова: нечіткий пошук, морфологія, стемінг, регулярні вирази

Обозначена и рассмотрена проблема нечеткого поиска в текстах. Проблема разделена на части и предложено решение по каждой из них. Также описаны общие способы и средства решения данных задач. На основании решения конкретных заданий в рамках проблемы нечеткого поиска предложен алгоритм реализации с помощью стемминга, перестановок и регулярных выражений.

Ключевые слова: нечеткий поиск, морфология, стемминг, регулярне выражения

Posed and reviewed a problem of fuzzy search in the texts. This problem already has a very good, but time-consuming solution with use of morphological analysis and dictionaries. In particular the optimum decision is difficult to implement for this purpose because it is necessary to realize also ways of recognition of the parts of a speech and to create the morphological dictionaries which aren't present in open access for Ukrainian language. The general problem of search was divided into components and the solution on each of them was proposed. Decisions make a start from the ways of data processing known and existing in open access. Use of the chosen methods within the solution of objectives was described in the article, and also their shortcomings are specified. On the basis of the solutions of components of a problem of indistinct search, offered an algorithm of fuzzy search based on permutations, stemming and regular expressions.

Keywords: fuzzy search, morphological analysis, stemming, regular expressions

Вступ

Проблема сучасного світу – інтенсивне зростання кількості інформації, яка до того ж зберігається в неструктурованих базах знань [3]. Тому пошук необхідної інформації може перевищувати час її опрацювання. Звісно, що все можна знайти, якщо знати що саме шукати. Але програмні засоби, які реалізують можливість пошуку за вмістом документів мають один вагомий недолік, а саме – вони шукають по точній фразі і показують лише точні збіги. А це значить, що якщо ми хочемо знайти певну фразу, але не знаємо як вона точно пишеться – ми будемо змушені перебирати різні варіанти для пошуку, або передивлятися весь документ у пошуках цільового фрагменту. Тому система з використанням морфології має значну перевагу над системою з точним пошуком, а саме можливість знаходити нечіткі збіги та знаходити слова в різних комбінаціях, а не тільки як вони введені. Це може в рази збільшити результативність пошуку. У цій статті запропоновано методику реалізації нечіткого пошуку з використанням морфології.

Основною морфологічною проблемою у процесі пошуку фрази є те, що вихідні слова можуть мати різні роди, числа і відмінки, при цьому змінюється написання слова, а суть – лише незначним чином. Тому при пошуку фрази нас цікавлять усі можливі написання слів, що входять у цю фразу. В українській мові слова мають лише одну незмінну характеристику, яка не залежить ні від числа, ні від відмінку – корінь.

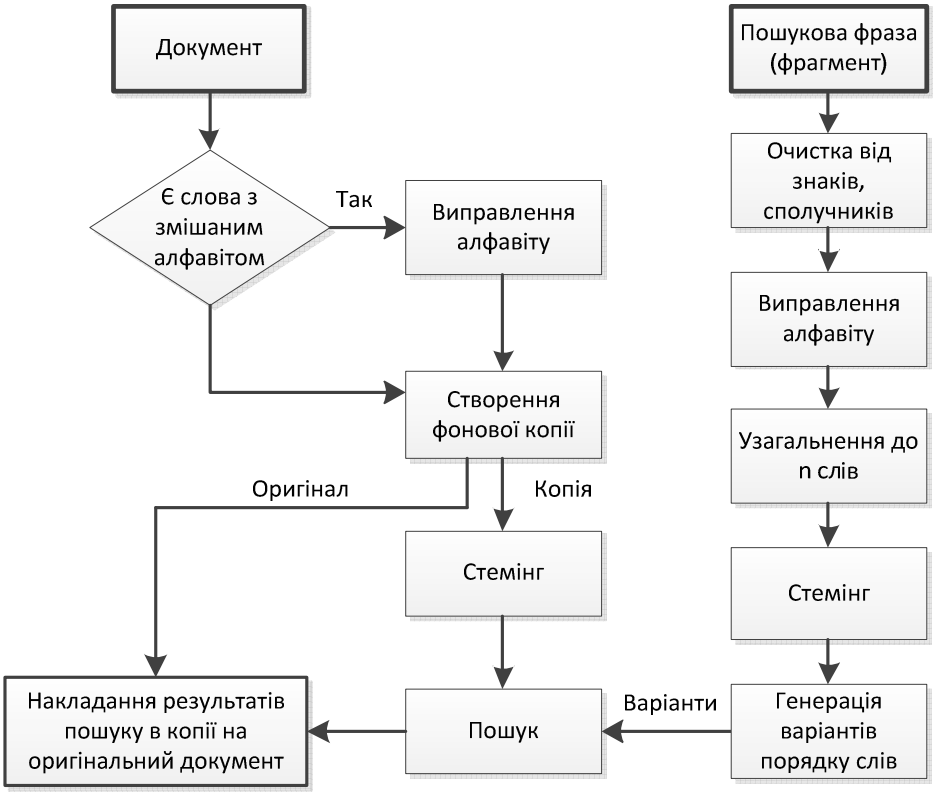

Рис. 1. Схема алгоритму роботи нечіткого пошуку

Ми можемо визначати корені слів і шукати не за цілим словом, а за його коренем. У такому випадку до вибірки в одне слово потрапляють всі спільнокореневі слова, зміст яких може значним чином відрізнятися від початкового. Однак слід враховувати, що метою пошуку переважно буде не одне слово, а ціла фраза, тобто у кожного слова заданої фрази буде певний контекст, складений із значень слів-сусідів. Саме цей контекст і повинен відсіяти більший відсоток можливих помилок, хоча і не усуне їх повністю. Тому при використанні цього методу є певна ймовірність отримання значення, що не релевантне пошуковій фразі, але цей відсоток має бути тим меншим, чим більше інформації для пошуку (довжина пошукової фрази). У статті відсотком помилок знехтувано, оскільки результати пошуку в будь-якому випадку видають людині, яка потім їх і перевіряє.

Знаходження незмінюваної частини слова

У програмуванні є відомий метод отримання частини слова, подібної до кореня, цей метод називається стемінг. Оскільки метод повертає насправді не корінь слова, а частину, що подібна до нього, тому часто стемінг використовують разом з лематизацією – розпізнаванням частин мови для слів. Основна задача стемінгу – відкидати закінчення слів. Але алгоритм не завжди може чітко визначити закінчення, тому він може відкинути не все, або навпаки – відкинути разом із закінченням іншу значущу частину слова, суфікс або й частину кореня. Це дві основні проблеми методу. Лематизація допомагає нівелювати ці помилки за рахунок введення різних правил стемінгу для різних частин мови. Результат неідеальний, але кращий за звичайний метод. Єдиний спосіб цілком правильно визначити корінь слова – це знайти слово у морфологічному словнику. Виняток становитимуть слова, які матимуть відразу кілька варіантів. Крім того, у порівнянні зі стемінгом, цей метод має свої мінуси.

По-перше, тоді доведеться замість незначних за обсягом правил стемінгу тримати у програмі цілий морфологічний словник.

По-друге, слова, які написані з помилками або яким просто не знайшлося відповідності у словнику, все одно доведеться обробляти якимось іншим способом. Ще одним мінусом є порівняно менша швидкодія. У поставленій задачі спосіб вилучення незмінюваної частини слова – це лише перший з етапів, тому краще не розпилювати ресурси на операції, які можна виконати менш затратно. Тому краще використовувати простий алгоритм. Але і реалізації навіть цього простого алгоритму для української мови й досі немає у вільному доступі. Він реалізований у комерційних програмних продуктах, у вільному доступі є лише деякі спроби реалізації стемінгу української мови. Тому необхідно робити свою реалізацію.

Рис. 2. Приблизна схема шаблону пошуку великого фрагменту тексту

Найпростіша реалізація стемера – це алгоритм який проходить по базі закінчень і шукає збіги закінчення слова, у випадку, якщо збіг знайдено – отримане закінчення відкидається. Таким чином ми можемо вирішити морфологічну складову нашого пошуку.

Знаходження усіх можливих комбінацій

Розглянемо проблему порядку слів. Можливо, що пошукова фраза містить слова не в тому порядку, в якому вони зустрічаються в документі. Для рішення цієї проблеми було вирішено поділити фразу на окремі слова і генерувати усі можливі варіанти їх сполучення. Сполучники та розділові знаки при цьому відкидаються, щоб зменшити кількість необхідних розрахунків. Таким чином з пошукової фрази: «радіус планети Марс» отримуємо такі комбінації для пошуку: «радіус планети Марс», «планети радіус Марс», «радіус Марс планети», «радіус Марс планети», «Марс радіус планети» та «Марс планети радіус», усього 6 комбінацій. Кількість можливих перестановок n елементів з множини m елементів обчислюється за формулою: $A_m^n = \frac{m!}{(m-n)!} = \binom{m}{n} n!$, але в нашому випадку вибираємо всі елементи множини, $n = m$ і формула спрощується до вигляду $A_m^m = P_m = n!$. Це значить, що складність пошуку різко зростатиме з кожним новим словом у фразі, оскільки для шести слів ми можемо отримати 6! = 720 їх різноманітних комбінацій, а для семи – вже 5040. Таким чином процес підбору варіантів значно ускладнюється з кожним додатковим словом для пошуку. Цю проблему можна подолати кількома варіантами. Найпростіший – при кількості слів у запиті більше певної кількості n (наприклад шести), ми пропускаємо етап підбору варіантів і обробляємо лише той порядок слів, який заданий. Інший, більш гнучкий варіант – робити «примусову деградацію» запиту, вибираючи з нього лише n слів випадковим чином. Таким чином у нас завжди буде не більше ніж $n!$ варіантів для пошуку. Число n при реалізації алгоритму в програмному забезпеченні можна визначати експериментально, враховуючи час, який необхідний для генерації варіацій з множини, що поступово збільшується і встановити n максимальним числом, при якому час на генерацію не перевищує, наприклад, двох секунд.

Розглянемо питання про забезпечення повноти «деградованого» запиту.

Відкидання непотрібної інформації

Наступна проблема пошуку – можлива наявність розділових знаків та сполучників у пошуковому запиті [4]. Наприклад, запит «Форд і Крайслер» при точному пошуку не знайде послідовність «Форд, Крайслер». Для того, щоб розв'язати і цю задачу, будемо видаляти всі розділові знаки, небуквені символи та сполучники з рядка запиту, та реалізуємо можливість знаходження слів, навіть якщо вони розташовані не поряд, а через кілька слів чи символів. Тобто шаблон пошуку одного слова матиме вигляд, як на рис. 2. Тут проміжок між словами умовно взято s = 20 символів. Зі статті [1] відомо, що середня довжина слова в українській мові становить 5,2 фонеми. За результатами [2] також стає відомо, що середня довжина слова у наукових роботах більше шести. Тому, прийнявши за відстань між словами 20 символів, можна знаходити збіги, якщо між пошуковими словами знаходиться інших 2-3 слова. Ця величина буде сталою лише для запитів довжиною не більше n. Оскільки для пошуку фрагментів довжиною більше n все одно буде використовуватися набір з n випадкових слів, то необхідно збільшувати допустимий проміжок між словами так, щоб вибірка могла покрити весь фрагмент. Ця величина «s» буде залежати від довжини слів, які не ввійшли у вибірку плюс запас на «зайві» слова, які можуть бути у фрагменті, що зустрічається в тексті. Експериментально можна використовувати таку формулу:

$$s = \frac{W - (w + n - 1)}{n} + 20k,$$

де W – кількість символів вихідного фрагменту; w – сумарна кількість символів у словах, що відібрані в пошукову фразу; n – кількість слів у пошуковій фразі; k – загальна кількість слів.

Тут вираховуємо кількість символів у словах, якими знехтували при формуванні пошукової фрази, віднімаємо довжину відібраних слів (враховуючи пробіли між словами), та додаємо по 20 «запасних» символів на кожне слово. Формула виведена теоретично і потребує перевірки на практиці.

Інші проблеми пошуку

Розглянемо дрібні проблеми, що можуть виникнути у процесі пошуку. До них належать спроби приховати скопійований текст шляхом заміни літер кирилиці на аналогічні за виглядом літери латиниці [5], ненавмисні помилки (слово з апострофом або ні, написання сполучень через дефіс

або через пробіл), різний регістр слів у пошуковій фразі та відповідному фрагменті в тексті. Заміну літер можна виявити просто. Для цього беремо до уваги, що літери латиниці і кирилиці не можуть бути в одному слові, а отже між ними як мінімум пробіл.

```
1: Regex rgx = new Regex(“[a-zA-Z][а-яА-Я]|[а-яА-Я][a-zA-Z]”);//регулярний вираз, що знаходить два символи з різних наборів поруч
2: foreach(String word in words)//берем по одному слову word з масива words
3: {
4:    Match mt = rgx.Match(word);
5:    if (mt.Success) //якщо щось знайдено
6:       ConvertToOne(word);
7: }
```

Лістинг 1. Код функції пошуку змішаних символів кирилиці та латиниці в одному слові

Тобто просто треба перевірити чи є в тексті такі літери, які нічим не розділені. Доведеться провести попередню обробку тексту, замінивши всі розділові знаки на пробіли, а потім створити з тексту масив слів. За допомогою регулярних виразів перевіряється кожне слово (приклад мовою C# наведений у лістингу 1).

У лістингу 1 функція ConvertToOne змінює всі літери латиницею на аналогічні кирилицею і навпаки (залежно від основної мови тексту) та виконує заміну у вихідному тексті на «нормальне» слово. Помилки з розділовими знаками, дефісами чи апострофами усуваємо повною заміною їх на пробіли (крім апострофа, він просто видаляється). Весь текст переводиться у нижній регістр, пошукова фраза також.

Як результат можна навести отриману схему пошуку фрази в заданому тексті (рис. 2). Для забезпечення певного функціоналу доведеться робити деяку попередню обробку тексту та створювати фонову копію документа. Сам процес пошуку реалізується за допомогою регулярних виразів.

Таким чином можна створити систему нечіткого пошуку по тексту з врахуванням всіх словоформ та різного порядку слів у тексті та пошуковій фразі.

Список літератури

1. *Макухіна Т.В. Особливості фонемної структури українських та англійських текстів інтерв’ю / Ліпатов В.М. // Матеріали наукової конференції «Наука и технологии: шаг в будущее – 2007».*
2. *Білощицький А.О., Діхтяренко О.В. «Ефективність методів пошуку збігів у текстах» // Управління розвитком складних систем (14). – С. 144-147.*
3. *Высоцкий, В.Ю. Поисковые алгоритмы для автоматизированного обучения [Текст] / В.Ю. Высоцкий, В.Д. Гогунский // Інформаційні технології в освіті, науці та виробництві. – № 3(4), 2013. – С. 105-113.*
4. *Зеленков Ю.Г. Сравнительный анализ методов определения нечетких дубликатов для Web-документов [Електронний ресурс] / Ю.Г. Зеленков, И.В. Сегалович. – Режим доступу: http://download.yandex.ru/company/download /paper_65_v1.pdf.*
5. *Толчеев В.О. Анализ проблемы и разработка процедуры выявления нечетких дубликатов научных статей по библиографическим описаниям [Текст] / В.О. Толчеев. – изд. "Новые технологии", "Информационные технологии", 2011. № 2 (174). – С.17-21.*

References

1. *Makuhina, T. V., Lipatov, V. N. (2007). Features of the phonemic structure of Ukrainian and English texts interview materials science. conf. "Science and technology: step in the future – 2007", Dnipropetrovsk, Ukraine.*
2. *Biloshchytskyi, A., Dikhtyarenko, O. (2013) Effectiveness of methods to search for matches in the texts. Management of complex systems. Kyiv, Ukraine: KNUCA, 14, 144-147.*
3. *Vysotsky, V. Y. Gogunsky, V. D. (2013). Search algorithms for computer-aided instruction. Information technology in education, science and industry, 3 (4), 105-113.*
4. *Zelenkov, J. G., Segalovich I. V. Comparative analysis of duplicate detection methods for Web-documents [E resource]. – Mode of access: http://download.yandex.ru/company/download /paper_65_v1.pdf.*
5. *Tolcheev, V. O. (2011). Analysis PROBLEMS pazpabotka and near-duplicate detection window procedure of scientific articles on bibliogpaficheskim descriptions. "New technologies", "Infopmatsionnye technology", 2 (174), 17-21.*

Стаття надійшла до редколегії 27.01.2014

Рецензент: д-р техн. наук, проф. С.Д. Бушуєв, Київський національний університет будівництва і архітектури.