Ілля Данилюк

ББК Ш111с51
УДК 81'322

ЛІНГВІСТИЧНЕ ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ: ПРИКЛАДИ ВИКОРИСТАННЯ РЕГУЛЯРНИХ ВИРАЗІВ

У статті розглянуто теоретичні основи використання стандартизованих регулярних виразів. Наведено галузі застосування РВ у прикладній лінгвістиці, визначено типологію спеціальних символів у РВ, принципи їх побудови, подано пропозиції щодо розширення переліку метасимволів, запропоновано низку шаблонів для виконання поширених завдань автоматичного опрацювання тексту.

Ключові слова: регулярний вираз, символ, рядок, метасимвол, квантифікатор, модифікатор.

Серед різних методів автоматизації опрацювання текстової інформації, прискорення пошуку, заміни окремих символів чи їх сукупностей регулярні вирази посідають одне з найважливіших місць. Регулярні вирази – це формальна мова пошуку й здійснення маніпуляцій з підрядками в тексті, що ґрунтується на використанні метасимволів [Фридл 2001: 19]. За іншим визначенням, це форма стандартного запису текстових даних, що використовується в автоматах зі скінченною кількістю станів [Jurafsky 2009: 17].

Існує чимала кількість визначень (і серед них важко виділити єдино правильне) поняття мови. Зокрема, Ноам Хомський називає мову «множиною (скінченною або нескінченною) речень, кожне з яких має скінченну довжину і побудоване зі скінченної множини елементів» [Хомский 1962: 416]. Очевидно, що РВ відповідають цьому визначенню й можуть бути детерміновані як метамова або спосіб описання будь-якої мови-об'єкта.

На сьогодні сформувалося кілька версій РВ. Попри ідентичність основних правил та метасимволів, вони відрізняються підтримуваними платформами (наприклад UNIX, Perl), інтерпретацією окремих шаблонів та синтаксисом. Зокрема, розрізняють т.зв. *базові регулярні вирази*, або *стандартні регулярні вирази для UNIX* – використовують умовно застарілий синтаксис, залишаючись при цьому – з огляду на те, що вони з'явилися першими, – сумісними з різними сучасними платформами. У базових РВ використовуються метасимволи ., [ ], [^], ^, $, *, перед фігурними та круглими дужками як операторами ставиться обернена скісна \, тобто нормативним є запис \{ \} та \(\), а також наявна можливість покликатися на попередні частини РВ за допомогою оператора \n. На заміну цьому стандарту було розроблено *стандарт POSIX,* або *розширені регулярні вирази*, у якому дещо змінено синтаксис, а саме скасовано використання оберненої скісної перед дужками, натомість її наявність перед метасимволом, навпаки, скасовує його спеціальне значення (наприклад, \? позначає звичайний символ ?), крім того, додано метасимволи +, ? та |, завдяки яким значно зросла функціональність розширених регулярних виразів. Нарешті, багатший синтаксис запропоновано у *регулярних виразах, сумісних з Perl* – інтерпретованою динамічною мовою програмування високого рівня загального призначення.

Регулярні вирази можуть допомогти вирішувати такі завдання аналізу тексту:

– автоматичний морфологічний аналіз тексту (див. [Марчук 2000: 60-70; Карпіловська 2006: 126-137; Дарчук 2008: 13-95]);

– лематизація (див. [Марчук 2000: 87-89]);

– лінгвістичне дешифрування (див. [Марчук 2000: 49-55]);

– аналітико-синтетичне опрацювання документів (див. [Партико 2008: 103-122]);

– інформаційний пошук (див. [Волошин 2004: 43-64]);

– завдання обчислювальної лексикографії (див. [Волошин 2004: 331-342; Марчук 2000: 70-87; Перебийніс 2009: 188-206]);

– лінгвістичні квантитативні дослідження загалом (див. [Баранов 2003: 38-51]);

– завдання синтаксичного аналізу (див. [Марчук 2000: 111-125; Карпіловська 2006: 139-143; Дарчук 2008: 95-145]), що включають побудову й використання систем видобування інформації, які ґрунтуються на атрибутах (див. [Рассел 2006: 1121]).

Конкретнішими завданнями можна назвати, наприклад:

– сортування слів, знайдених у тексті;

– обчислення частотності використання слів;

– одержання доступу до довільних фрагментів тексту;

– статистичне вимірювання лексичної різноманітності мови;

– автоматична побудова графіка на основі видобутої з тексту інформації;

– знаходження загальних контекстів для різних слів;

– розпізнавання граматики мови програмування;

– організація підсвічування синтаксису мови програмування;

– ведення діалогу ЕОМ з користувачем тощо.

У пропонованій статті ми ставимо за мету подати типологію використовуваних у стандартизованих системах метасимволів та їх комбінацій, а також запропонувати зразки вирішення типових завдань з обробки певних текстових даних. Реалізація мети передбачає виконання кількох завдань:

1. Описати основні засоби та способи використання РВ;
2. Згрупувати метасимволи РВ за функціональним критерієм;
3. Продемонструвати приклади ефективного використання РВ.

Регулярні вирази за [Форта 2005: 15-16] складаються з констант і операторів, які визначають множини рядків і множини операцій щодо них відповідно. У певному скінченному алфавіті Σ можна визначити такі константи: (порожня множина) ∅; (порожній рядок) ε – позначає рядок, що не містить жодного символу й має еквівалент «»; (символьний літерал) «a», де a – символ алфавіту Σ; і такі операції: (зчеплення, конкатенація) RS позначає множину {αβ | α ∈ R & β ∈ S}. Наприклад, {«boy», «girl»} {«friend», «cott»} = {«boyfriend», «girlfriend», «boycott», «girlcott»}; (диз'юнкція, чергування) R|S позначає об'єднання R і S, наприклад, {«ab», «c»}|{«ab», «d», «ef»} = {«ab», «c», «d», «ef»}; (замикання Кліні, зірка Кліні) R* – позначає мінімальну надмножину множини R, яка містить ε і є замкнутою щодо конкатенації, тобто це множина всіх рядків, отриманих конкатенацією нуля або більше рядків з R, наприклад, {«Новий», «Рік»}* = {ε, «Новий», «Рік», «НовийНовий», «НовийРік», «РікНовий», «РікРік», «НовийНовийНовий», «НовийНовийРік», «НовийРікНовий», ...}.

У регулярних виразах використовуються звичайні (літерали) та спеціальні символи (метасимволи). Більшість символів у регулярному виразі представляють самі себе, за винятком спеціальних символів [ ] \ ^ $. | ? * + () {}, перед якими може стояти символ \ (обернений слеш) для представлення їх самих як символів тексту – так зване «екранування». Можна «екранувати» цілу послідовність символів, помістивши її між метасимволом \Q і \E. Для вказування меж РВ традиційно використовується символ / (слеш), при цьому сам він не входить до РВ, а для позначення меж текстових рядків використовуються лапки «». Таким чином, РВ /a\.?/ відповідає рядку тексту «a.» або «a»; /a\\\\b/ – «a\\b»; /a\[F\]/ – «a[F]»; /\Q+-*/\E/ – «+-*/».

Одним з найбільш вживаних є метасимвол . (крапка), який позначає один будь-який символ, включно із символом нового рядка у деяких реалізаціях РВ. Наприклад, РВ /гр.б/ відповідатиме послідовностям символів «гриб», «гріб», «граб», «гррб», «гр4б», «прогрябати» тощо.

У деяких випадках слід віддавати перевагу представленню символів за їхнім кодом (Табл. 1).

Таблиця 1. Представлення у РВ символів за їхнім кодом

Представлення	**Пояснення**	**Кодування**
\0*n*	*n* – вісімкове число від 0 до 377	8-бітове
\x*dd*	*d* – шістнадцяткова цифра	
\u*dddd*		16-бітне (Юнікод)

У РВ, пов'язаних з автоматизацією процесів обробки текстів, часто виникає необхідність використання спеціальних керівних символів (Табл. 2).

Таблиця 2. Керівні символи в РВ

Представлення	**Символ**	**Позначення**
\t	Табуляція	HT
\v	Вертикальна табуляція	VT
\r	Повернення каретки	CR
\n	Переведення рядка	LF
\f	Кінець сторінки	FF
\e	Escape-символ	ESC
\b	Backspace-символ, який повинен знаходитися всередині квадратних дужок (інакше інтерпретується як межа слова)	BS
\cA … \cZ	Ctrl+A … Ctrl+Z	

Спеціальні символи дозволяють позиціонувати регулярний вираз щодо елементів тексту: початку і кінця рядка, меж слова (див. Таблиця 3.).

Набір символів у квадратних дужках [ ] іменується символьним класом і дозволяє вказати інтерпретатору регулярних виразів, що на цьому місці в рядку може стояти один з перерахованих символів. Наприклад, [абв] задає можливість появи в тексті одного з трьох зазначених символів, а [1234567890] відповідає одній із цифр. Існує можливість вказати діапазон символів: наприклад, [А-Яа-я] відповідає всім

буквам українського алфавіту, за винятком літер «І» та «і»[1] . Якщо потрібно вказати символи, які не входять до вказаного набору, то використовують символ ^ всередині квадратних дужок, наприклад**,** [^0-9] означає будь-який символ, окрім цифр.

Таблиця 3. Метасимволи PB для позиціонування шуканого виразу

Представлення	**Позиція**	**Приклад**	**Відповідність**
^	Початок рядка	/^a/	«aaa aaa»
$	Кінець рядка	/a$/	«aaa aaa»
\b	Межа слова	/a\b/	«aaa aaa»
		/\ba/	«aaa aaa»
\B	Не межа слова	/\Ba\B/	«aaa aaa»
\G	Попередній успішний пошук	/\Ga/	«aaa aaa» (пошук зупинився на 4-ій позиції – там, де не знайшлося *a*)

Для часто використовуваних символьних класів існують короткі позначення (Табл. 4).

Таблиця 4. Стандартні спеціальні символьні класи

Представлення	**Еквівалент**	**Значення**
\d	[0-9]	Цифра
\D	[^\d]	Будь-який символ, крім цифри
\w	[A-Za-Za-Яа-Я0-9_]	Символи, що утворюють «слово» (літери, цифри і символ підкреслення). Слід звернути увагу, що вказаний еквівалент неповний, оскільки до діапазону входять всі букви всіх мов.
\W	[^\w]	Символи, що не утворюють «слово»
\s	[\t\v\r\n\f]	Пробільний символ
\S	[^\s]	Непробільний символ

Окремі діапазони символів залежать від обраних налаштувань локалізації. У POSIX стандартизовано оголошення деяких класів і категорій символів (Табл. 5).

Таблиця 5. Символьні класи і категорії POSIX

POSIX-клас	**Еквівалент**	**Значення**
[:upper:]	[A-Z]	Символи верхнього регістру
[:lower:]	[a-z]	Символи нижнього регістру
[:alpha:]	[[:upper:][:lower:]]	Букви
[:digit:]	[0-9], тобто \d	Цифри
[:xdigit:]	[[:digit:]A-Fa-f]	Шістнадцяткові цифри
[:alnum:]	[[:alpha:][:digit:]]	Літери і цифри
[:word:]	[[:alnum:]_], тобто \w	Символи, що утворюють «слово»
[:punct:]	[-!»#$%&'()*+,./:;<=>?@[\\\]_`{|}~]	Знаки пунктуації
[:blank:]	[\t]	Пробіл і табуляція
[:space:]	[[:blank:]\v\r\n\f], тобто \s	Пробільні символи
[:cntrl:]	[\x00-\x1F\x7F]	Символи керування
[:graph:]	[\x21-\x7E]	Друковані символи
[:print:]	[\x20-\x7E], тобто [[:graph:]]	Друковані символи з пробілом

[1] Для використання послідовностей літер необхідно встановити правильну кодову сторінку, в якій ці послідовності йтимуть у порядку від і до вказаних символів. Для української мови це Windows-1251, ISO 8859-5 і Юнікод, оскільки в DOS-855, DOS-866 і KOI8-R та KOI8-U українські літери не йдуть однією цілою групою або не впорядковані за алфавітом. Окрему увагу слід приділяти буквам з діакритичними знаками, на зразок українських Ї/ї, що, як правило, розташовані поза основними діапазонами символів.

Крім того, у РВ широко використовується квантифікація, тобто пошук послідовностей символів. При цьому застосовуються квантифікатори, які розташовані після символу, символьного класу або групи і визначають, скільки разів попередній вираз може зустрічатися. Слід враховувати, що квантифікатор може стосуватися більш ніж одного символу у РВ, тільки якщо це символьний клас або група (Табл. 6).

Таблиця 6. Квантифікатори

Представлення	**Кількість повторень**	**Приклад**	**Відповідність**
{*n*}	Рівно *n* разів	colou{3}r	colouuur
{*m*,*n*}	Від *m* до *n* включно	colou{2,4}r	colouur, colouuur, colouuuur
{*m*,}	Не менше *m*	colou{2,}r	colouur, colouuur, colouuuur і т. д.
{,*n*}	Не більше *n*	colou{,3}r	color, colour, colouur, colouuur

Крім цифрових квантифікаторів, використовуються також символьні (Табл. 7).

Таблиця 7. Символьні квантифікатори

Представлення	**Кількість повторень**	**Еквівалент**	**Приклад**	**Відповідність**
*	Нуль або більше	{0,}	colou*r	color, colour, colouur і т. д.
+	Одне або більше	{1,}	colou+r	colour, colouur і т. д. (але не color)
?	Нуль або одне	{0,1}	colou?r	color, colour

Часто використовується послідовність /.*/ для позначення будь-якої кількості будь-яких символів між двома частинами регулярного виразу. Символьні класи в поєднанні з квантифікатори дозволяють встановлювати відповідності з реальними текстами: стовпцями цифр, телефонами, поштовими адресами, елементами HTML-розмітки тощо. Якщо символи { } не утворюють квантифікатор, їхнє спеціальне значення ігнорується.

У деяких реалізаціях квантифікаторам у регулярних виразах відповідає максимально довгий рядок із можливих (квантифікатори є *жадібними,* англ. *greedy).* В окремих випадках це може виявитися проблемою. Наприклад, часто очікують, що вираз (<.*>) знайде в тексті теги HTML. Однак, якщо в тексті є більше одного HTML-тега, то цьому виразу відповідає цілий рядок, що містить безліч тегів:

```
<p>Сторінка <b>Курси</b> містить навчальний матеріал з різних тем.</p>
```

Є два шляхи вирішення цієї проблеми:

1. Враховувати символи, що не відповідають бажаному зразку (<[^>]*> для описаного вище випадку).
2. Визначити квантифікатор як *нежадібний (ледачий,* англ. *lazy)* – більшість реалізацій РВ дозволяють це зробити, додавши після нього знак питання *(Див. Табл. 8*).

Використання ледачих квантифікаторів може спричинити зворотну проблему, коли виразу відповідає занадто короткий, зокрема, порожній рядок.

Таблиця 8. Жадібні і ледачі квантифікатори

Жадібний	**Ледачий**
*	*?
+	+?
{*n*,}	{*n*,}?

Також загальною проблемою як жадібних, так і ледачих виразів є точки повернення для перебору варіантів виразу. Точки ставляться після кожного повтору, або ітерації квантифікатора. Якщо інтерпретатор не знайшов відповідності після квантифікатора, то він починає повертатися за всіма встановленими точкам, перераховуючи звідти вираз по-іншому.

Нарешті, існує так звана ревнива, або *наджадібна* квантифікація, яка, на відміну від звичайної (жадібної), не тільки намагається знайти максимально довгий варіант, але ще й не дозволяє алгоритму повертатися до попередніх кроків пошуку для того, щоб знайти можливі відповідності для решти регулярного виразу.

Використання ревнивих квантифікаторів збільшує швидкість пошуку, особливо в тих випадках, коли рядок не відповідає регулярному виразу. Крім того, ревниві квантифікатори можуть бути використані для виключення небажаних збігів.

Таблиця 9. Жадібні і ревниві квантифікатори

Жадібний	**Ревнивий**
*	*+
?	?+
+	++
{*n*,}	{*n*,}+

Наприклад, у рядку «abxaa bxaa» РВ /ab (xa)*+a/ знайде «abxaabxaa», але не «abxa bxaa», оскільки виділене сірим а зайняте попередньою ітерацією.

У всіх версіях РВ широко використовується прийом групування, що має подвійне призначення – для визначення меж дії і пріоритету операцій. Шаблон всередині групи обробляється як єдине ціле й може бути квантифікований. Наприклад, вираз /(тр[ау]м-?)*/ знайде послідовність вигляду «трам-трам-трумтрам-трум-трамтрум». Одне із застосувань групування – повторне використання раніше знайдених груп символів (підрядків, блоків, виділених підвиразів). При обробці виразу підрядки, знайдені за шаблоном всередині групи, зберігаються в окремій області пам'яті і отримують номер, починаючи з одиниці. Кожному підрядку відповідає пара дужок у регулярному виразі. Квантифікація групи не впливає на збережений результат, тобто зберігається лише перше входження. Зазвичай підтримується до 9 нумерованих підрядків з номерами від 1 до 9, але деякі інтерпретатори дозволяють працювати з більшою кількістю. Згодом у межах одного регулярного виразу можна використовувати позначення від \1 до \9 для перевірки на збіг з раніше знайденим підрядком.

Наприклад, регулярний вираз /(та|ту)-\1/ знайде рядок «та-та» або «ту-ту», але пропустить рядок «та-ту».

Також раніше знайдені підрядки можна використовувати при заміні за регулярним виразом. У такому випадку в заміну тексту вставляються ті ж позначення, що і в межах самого виразу. Наприклад, якщо для пошуку задати РВ /\bДмитр([аоеіїу]+[мв]?[і]?)\b/, а для заміни /Петро\1/, то всі відмінкові форми імені «Дмитро» будуть замінені на «Петро».

У разі якщо частина регулярного виразу, взята в дужки (група), не повинна отримувати особливий номер у списку груп і займати місце в пам'яті, в неї після першої дужки додають оператор /?:/, наприклад (виділено): якщо для пошуку задати РВ /\b[А-яІіЇїЄєҐг]+(?:ий|ій|ого|ому|їм|ім)\b\s\b Дмитр([аоеіїу]+[мв]?[і]?)\b/, а для заміни /Петро\1/, то всі словосполучення, в яких перше слово буде закінчуватися на одне з можливих закінчень прикметника (ий|ій|ого|ому|їм|ім), а другим буде одна з відмінкових форм імені «Дмитро» (наприклад, «веселий Дмитро», «цікавому Дмітрові»), будуть замінені на «Петро», оскільки знайдена група можливих закінчень прикметника не отримає номер 1, на відміну від групи закінчень іменника «Дмитро», покликання на яку й буде використане при заміні.

Атомарне групування вигляду (?>*Шаблон*), так само як групування без зворотного зв'язку, не створює зворотних зв'язків. На відміну від нього, таке групування забороняє повертатися назад у рядку, якщо частина шаблону вже знайдена. Наприклад, у рядку «abccaxcc» регулярний вираз /a(?>bc|b|x)cc/ знайде тільки «abccaxcc», але не «abccaxcc», оскільки варіант з *x* знайдений, інші будуть проігноровані. Атомарне групування виконується ще швидше, ніж групування без зворотного зв'язку, і зберігає процесорний час при виконанні решти виразу, оскільки забороняє перевірку будь-яких інших варіантів всередині групи, коли один варіант вже знайдений. Це дуже корисно при оптимізації груп з безліччю різних варіантів.

У всіх версіях РВ використовуються модифікатори, які діють з моменту входження і до кінця регулярного виразу або протилежного модифікатора (Табл. 10). Деякі інтерпретатори можуть застосувати модифікатор до всього виразу, а не з моменту його входження.

Групи-модифікатори можна об'єднувати в одну групу: (?i-sm). Така група вмикає режим *i* і вимикає режим *s, m.* Якщо використання модифікаторів потрібне тільки в межах групи, то необхідний шаблон вказується всередині групи після модифікаторів і двокрапки. Наприклад, РВ /(?-i)(?i:TV)set/ знайде «TVset», але не «TVSET».

Для додавання коментарів у регулярний вираз можна використовувати групи-коментарі вигляду(? *#коментар).* Така група інтерпретатором повністю ігнорується й не перевіряється на входження в текст. Наприклад, вираз /А(?#Тут коментар)Б/ відповідає рядку «АБ».

Таблиця 10. Модифікатори у РВ

Синтаксис	**Опис**	
(?i)	Вмикає	нечутливість виразу до регістру символів
(?-i)	Вимикає	
(?s)	Вмикає	режим відповідності крапки (.) символам переносу рядка й повернення каретки
(?-s)	Вимикає	
(?m)	Символи ^ і $ викликають відповідність тільки	після і до символів нового рядка
(?-m)		з початком і кінцем рядка
(?x)	Вмикає	режим без врахування пробілів між частинами регулярного виразу і дозволяє використовувати # для коментарів
(?-x)	Вимикає	

Вертикальна риса розділяє допустимі варіанти. Наприклад, /gray|grey/ відповідає «gray» або «grey». Слід пам'ятати, що перебір варіантів виконується зліва направо, як вони вказані. Якщо потрібно вказати перелік варіантів всередині складнішого регулярного виразу, то його потрібно помістити в групу. Наприклад, /gray|grey/або /gr(a|e)y/ описують рядок «gray» або «grey». У випадку з односимвольними альтернативами найкращим є варіант /gr[ae]y/, оскільки порівняння з символьним класом виконується простіше, ніж обробка групи з перевіркою на всі її можливі модифікатори й генерацією зворотного зв'язку.

У більшості реалізацій регулярних виразів є спосіб проводити пошук фрагмента тексту, «переглядаючи» (але не включаючи до знайденого) сусідній текст, який розташований до або після шуканого фрагмента тексту (Табл. 11). Наприклад, у такий спосіб легко знайти ім'я тега HTML, не включаючи до результатів пошуку кутові дужки поруч або інші знаки, але й не випускаючи їх «з уваги» при пошуку потрібного контексту. Перегляд з запереченням використовується рідше і «стежить» за тим, щоб вказані відповідності, навпаки, не зустрічалися до або після шуканого текстового фрагмента.

Таблиця 11. Метасимволи для перегляду сусіднього тексту

Представлення	**Вид перегляду**	**Приклад**	**Відповідність**
(? =*шаблон)*	Позитивний перегляд вперед	/Людовік (?=XVI)/	ЛюдовікXV, ЛюдовікXVI, ЛюдовікXVIII, ЛюдовікLXVII, ЛюдовікXXL
(?!*шаблон*)	Негативний перегляд вперед (із запереченням)	/Людовік(?!XVI)/	ЛюдовікXV, ЛюдовікXVI, ЛюдовікXVIII, ЛюдовикLXVII, ЛюдовикXXL
(?<=*шаблон*)	Позитивний перегляд назад	/(?<=Сергій) Іванов /	Сергій Іванов, Ігор Іванов
(?<!*шаблон*)	Негативний перегляд назад (із запереченням)	/(?<! Сергій) Іванов /	Сергій Іванов, Ігор Іванов

У багатьох реалізаціях регулярних виразів існує можливість вибирати, яким шляхом піде перевірка в тому чи іншому місці регулярного виразу на підставі вже знайдених значень.

Представлення	**Пояснення**	**Приклад**	**Відповідність**
(?(?=*якщо) то*|*інакше)*	Якщо операція перегляду успішна, то далі виконується частина *то,* інакше виконується частина *інакше.* У виразі може використовуватися будь-яка з чотирьох операцій перегляду. Слід враховувати, що операція перегляду нульової ширини, тому частини *то* в разі позитивного чи *інакше* у разі негативного перегляду повинні включати в себе опис шаблону з операції перегляду.	(?(?<=а)м|п)	мам,пап
(?(*п*)*то*|інакше)	Якщо *п-а* група повернула значення, то пошук за умовою виконується за шаблоном *то,* інакше за шаблоном *інакше.*	(а)?(?(1)м|п)	мам,пап

Розгляньмо приклади вирішення типових завдань з опрацювання тексту з використанням РВ, зокрема, пошук рядків, у яких є або, навпаки, немає певних слів.

Якщо потрібно знайти рядок, у якому зустрічається одне з кількох слів, то достатньо використовувати РВ з альтернативами, а саме: вираз /^.*\b(один|два|три)\b.*$/ знайде усі рядки, що будуть містити будь-яке зі слів «один», «два» або «три». Першим зворотним покликанням буде знайдене в рядку слово. Якщо рядок міститиме два або усі три слова, то тільки останнє з них (крайнє справа) потрапить у комірку пам'яті як перше зворотне покликання. Це тому, що зірка * вмикає «жадібний» пошук. Якщо зробити першу зірку «ледачою», як у РВ /^.*?\b(один|два|три)\b.*$/, то зворотне покликання міститиме перше слово зліва.

Якщо рядок повинен містити всі три потрібні слова, то слід використати інструмент перегляду вперед. РВ /^(?=.*?\bодин\b)(?=.*?\bдва\b)(?=.*?\bтри\b).*$/ знайде тільки такі рядки з тексту, що містять усі три слова – «один», «два» і «три». Всі три вказані у цьому РВ перегляди вперед повинні бути успішними, щоб рядок загалом відповідав умові РВ. Зверніть увагу, що замість власне слова, наприклад, /\bодин\b/, в умові перегляду вперед можна використати будь-який складніший РВ.

Якщо потрібно, щоб рядок не містив певного тексту, слід використовувати негативний перегляд вперед. Так, РВ /^((?!бодин\b).)*$/ знайде цілий рядок, що не містить слова «один». Зверніть увагу, що на відміну від попереднього прикладу з використанням позитивного перегляду вперед, повторено разом негативний перегляд вперед і крапку, оскільки для позитивного перегляду вперед потрібно лише знайти місце в тексті, де РВ спрацює, а для негативного перегляду вперед треба перевірити всі без винятку позиції символу в рядку. Іншими словами, слід переконатися, що РВ не дає позитивного результату пошуку всюди, а не тільки в якомусь одному місці.

Нарешті, можна об'єднати кілька позитивних і негативних переглядів уперед, наприклад: /^(?=.*?\bобов'язковий\b)(?=.*?\bнеобхідний\b)((?!непотрібний|факультативний).)*$/. Цей РВ знаходитиме рядки, у яких присутні слова «обов'язковий» і «необхідний», але немає слів «непотрібний» чи «факультативний». При перевірці кількох позитивних переглядів уперед, /.*/ у РВ гарантує, що перед шуканими словами може бути інший текст.

Іншим завданням автоматичного опрацювання тексту може бути пошук двох близько розташованих слів.

Деякі пошукові інструменти, окрім використання логічних операторів на зразок і/або також мають спеціальний оператор з умовною назвою «поряд», «біля». Пошукова умова «слово1 біля слово2» знаходить всі входження слово1 і слово2, які перебувають на певні відстані у тексті одне від одного. Відстанню є кількість слів. Ця кількість залежить від інструменту пошуку і часто може бути налаштована користувачем.

Подібне завдання може бути реалізоване за допомогою РВ. РВ у цьому випадку є досить нескладним: перше слово, певна кількість невідомих слів, і друге слово. Невідомі слова можна описати символьним класом \w+. Пробіли та інші символи між словами можна описати символьним класом \W+. Загальний вигляд РВ: /\bслово1\W+(?:\w+\W+){1,6}?слово2\b/. Квантифікатор {1,6}? у цьому РВ вказує, що відстань між словами може складати від 1 до 6 слів.

Якщо слова можуть зустрічатися також у зворотному порядку, то це слід вказати, а саме: /\b(?:слово1\W+(?:\w+\W+){1,6}слово2|слово2\W+(?:\w+\W+){1,6}слово1)\b/.

Якщо потрібно знайти пару з двох слів з певного списку, то РВ може мати вигляд: /\b(слово1|слово2|слово3)(?:\W+\w+){1,6}?\W+(слово1|слово2|слово3)\b/. Цей регулярний вираз також знайде повтор того самого слова..

Постійно виникає потреба видалення у списку або базі даних рядків або записів, що дублюють один одного.

Наприклад, у файлі, де всі рядки відсортовані в алфавітному порядку, можна легко видалити повторювані рядки. Відкривши файл у текстовому редакторі з підтримкою РВ, слід здійснити пошук шаблону /^(.*)(\n\1)+$/ і замінити його на шаблон /\1/.

У системах автоматичного опрацювання текстів, зокрема, автоматичного морфологічного або синтаксичного аналізу, частиномовних статус окремих слів, зокрема, слів у найближчому контексті, дозволяє розмежовувати різноманітні випадки граматичної омонімії. Наприклад, щоб відрізнити Д.в. одн. іменників від М.В. одн., необхідно у контексті знайденої словоформи знайти (або не знайти) прийменник. Послідовність створення РВ можна представити так (обмежимося для прикладу правилом, що словоформи у Д.в. та М.в. одн. мають закінчення *–і, -ові, -у*):

1.Шукаємо слова, які закінчуються на *і, ові, у* – /\b[А-яІіЇїЄєҐґ]+(і|ові|у)\b/

2.Шукаємо слова, які закінчуються на *і, ові, у,* а безпосередньо перед ними буде один із прийменників *на, по, при*: /(?<=\bна\W|\bпо\W|\bпри\W)\b[А-яІіЇїЄєҐґ]+(і|ові|у)\b/

Вважаємо, що для окремих локалізацій інтерпретаторів РВ було б перспективним розробити специфічні символьні класи, наприклад, клас непохідних прийменників української мови [:ukrprep:] :

Символьний клас	**Еквівалент**	**Значення**
[:ukrprep:]	на|при|про|зі|з|із|під …	Непохідні прийменники

Це дозволило б скоротити запис РВ для попереднього завдання до вигляду:

/(?<=\b[:ukrprep:]\W)\b[А-яІіЇїЄєҐґ]+(і|ові|у)\b/

Отже, використання РВ при опрацюванні текстової інформації уможливлює автоматизацію чималої кількість завдань та процедур. При роботі з РВ потребує пильної уваги та правильного налаштування локалізація операційної системи та програмного забезпечення для створення максимальної сумісності стандартизованих версій РВ з опрацьовуваною мовою. Перспективним видається створення спеціальних версій РВ для окремих мов, що мають свої особливості графіки, у тому числі української, а також опис спеціальних символьних класів, а ширше – класів слів чи словоформ.

Література

Баранов 2003: Баранов, А.Н.Введение в прикладную лингвистику [Текст] / А.Н.Баранов. – М. : Едиториал УРСС, 2003. – 360 с. – ISBN 5-8360-0196-0.

Волошин 2004: Волошин, В.Г. Комп'ютерна лінгвістика : Навчальний посібник [Текст] / В.Г.Волошин. – Суми : ВТД «Університетська книга», 2004. – 382 с. – ISBN 966-680-134-5.

Дарчук 2008: Дарчук, Н.П. Комп'ютерна лінвістика: Автоматичне опрацювання тексту [Текст] / Н.П. Дарчук. – К. : Видавничо-поліграфічний центр «Київський університет», 2008. – 351 с. – ISBN 978-966-439-079-5.

Карпіловська 2006: Карпіловська, Є.А. Вступ до прикладної лінгвістики : комп'ютерна лінгвістика. Підручник [Текст] / Є.А.Карпіловська. – Донецьк : ТОВ «Юго-Восток, Лтд», 2006. – 188 с. – ISBN 966-374-078-7.

Марчук 2000: Марчук, Ю.Н. Основы компьютерной лингвистики [Текст] / Ю. Н. Марчук. – М. : Народный учитель, 2000. – 320 с. – ISBN 5-17-039480-2.

Партико 2008: Партико, З.В. Прикладна і комп'ютерна лінгвістика: Вступ до спеціальності [Текст] / З.В. Партико. – Львів: Афіша, 2008. – 224 с. – ISBN 978-966-325-092-2.

Перебийніс 2009: Перебийніс В.І., Сорокін В.М. Традиційна та комп'ютерна лексикографія : Навчальний посібник [Текст] / В.І.Перебийніс. – К.: Вид. центр КНЛУ, 2009. – 218 с.

Рассел 2006: Рассел, С., Норвиг, П. Искусственный интеллект: современный подход [Текст] / Пер. с англ. – М. : Издательский дом «Вильямс», 2006. – 1408 с. – ISBN: 5-8459-0887-2.

Смит 2006: Смит, Б. Методы и алгоритмы вычислений на строках (regexp) = Computing Patterns in Strings [Текст] / Пер. с англ. – М. : Издательский дом «Вильямс», 2006. – 496 с. — ISBN 0-201-39839-7.

Форта 2005: Форта, Бен. Освой самостоятельно регулярные выражения. 10 минут на урок = Sams Teach Yourself Regular Expressions in 10 Minutes. — М.: «Вильямс», 2005. — 184 с. — ISBN 5-8459-0713-6

Фридл 2001: Фридл, Дж. Регулярные выражения [Текст] / Дж. Фридл. – СПб : Питер, 2001. – 352 с. – ISBN 5-318-00056-8.

Хомский 1962: Хомский Н. Синтаксические структуры [Текст] / Н.Хомский // Новое в лингвистике. – Вып. 2. – М. : Издательство иностранной литературы, 1962. – С. 412-528.

Jurafsky 2009: Jurafsky, D., Martin, J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition [Text]. – Prentice Hall, 2009. – 988 p. – ISBN 0-13-095069-6.

В статье рассмотрены теоретические основы использования стандартизированных регулярных выражений. Приведены области применения РВ в прикладной лингвистике, определены типология специальных символов в РВ, принципы их построения, предложены варианты расширения перечня метасимволов, а также ряд шаблонов для выполнения распространенных задач автоматической обработки текста.

Ключевые слова: регулярное выражение, символ, строка, метасимвол, квантификаторы, модификатор.

The article reviews the theoretical foundations of standard regular expressions (regexp). The use of regexp in applied linguistics is shown, the typology of special symbols in regexp is defined, as well as the principles of their construction, and some proposals to expand the list of metasymbols are made, the series of templates to perform common tasks of automatic processing of text is proposed also.

Keywords: regular expression, symbol, string, metasymbol, quantifier, modifier.

Надійшла до редакції 16 вересня 2010 року.

Инна Заваруева

ББК Ш 111.4
УДК 811.161.1'374

СЛОВАРНАЯ СТАТЬЯ КАК ОСОБОЕ ЛЕКСИКОГРАФИЧЕСКОЕ ПОСТРОЕНИЕ (НА ПРИМЕРЕ ЭЛЕКТРОННЫХ СЛОВАРЕЙ)

Проаналізовано структуру словникової статті в різних видах тлумачних електронних словників у порівнянні зі словниками паперовими та іншими видами електронних словників, помічено те, що структура