УДК 378.147:004.432

Рєзіна О.В.[1], Косюг Р.М.[2]
[1] Центральноукраїнський державний педагогічний університет імені Володимира Винниченка, Кропивницький, Україна
[2] Компанія AVA.codes, Київ, Україна

ТЕХНОЛОГІЇ СТВОРЕННЯ ПРОГРАМИ ПЕРЕВІРКИ ОРФОГРАФІЇ

DOI: 10.14308/ite000698

Програми перевірки орфографії створюються для того, щоб контролювати та виправляти помилки в документі користувача. Робота таких програм базується на порівнянні кожного набраного слова зі списком правильно написаних слів та використанні алгоритмів визначення коректного написання. У статті розглядаються та аналізуються технології створення програми перевірки орфографії, а також методика навчання студентів цих технологій. Досліджено програму контролю правопису, написану Пітером Норвігом, та визначено шляхи модифікації цієї програми для опрацювання текстів українською мовою. Запропоновано підхід до реалізації мовної моделі, тобто створення списку правильно написаних слів, заснований на використанні матеріалів «Браунського корпусу української мови».

Визначено особливості побудови регулярного виразу для виокремлення слів із тексту українською мовою. У ролі матеріалу для тестування програми перевірки орфографії використано тексти, що містять субтитри українською мовою, які створено у межах волонтерського перекладацького проекту «To Be Announced». Описано програму, що опрацьовує цей текстовий масив з метою перевірки правопису, та проаналізовано отримані результати. Зроблено висновок про те, що отримані результати були коректними, що заохочує до подальших досліджень.

Ключові слова: *програма перевірки орфографії; мовна модель; регулярний вираз; мова програмування Python; навчання програмування.*

Постановка проблеми. Програмами перевірки орфографії користуються мільйони людей у всьому світі. На ринку програмного забезпечення пропонується велика кількість платних та безкоштовних сервісів перевірки орфографії. Робота таких програм постійно вдосконалюється для різних мов світу. Тому доцільність та практична значущість навчання студентів спеціальностей 035.10 Філологія (Прикладна лінгвістика), 122 Комп'ютерні науки, 014.09 Середня освіта (Інформатика) створення та використання програм перевірки орфографії не викликає сумніву. Робота над програмою перевірки орфографії передбачає формування та розвиток таких умінь:

- застосування сучасних засобів програмування;
- оновлення, оптимізація, розширення функціональності наявних програм;
- використання різних структур даних;
- опрацювання текстових файлів;
- добір методів та засобів тестування програм;
- аналіз та статистичне опрацювання даних.

Це сприяє:

- удосконаленню техніки програмування;
- розвитку логічного мислення та навичок вирішення проблем;

– кращому розумінню методології розробки програмного забезпечення.

Слід зазначити, що наведені вище вміння входять до переліку професійних вимог работодавців до працівників ІТ-галузі [1; 2].

Аналіз останніх досліджень і публікацій. Дослідження науковців в галузі комп'ютерного опрацювання текстів і побудови програм корекції орфографії сфокусовані на вдосконаленні наявних програм та створенні програм перевірки правопису національних мов. Так, у роботі [3] аналізуються сучасні дослідження в галузі комп'ютерного опрацювання природної мови (Natural Language Processing) і зазначається, що використання нейронних мереж та машинного навчання надало можливості для великих досягнень у цій галузі. Але автори доходять висновку, що більшість досліджень зосереджені на англійській мові, і пропонують підхід до створення програми корекції орфографії шведської мови, що базується на корпусі Göteborgs-Posten та використанні довгої короткотермінової пам'яті рекурентної нейронної мережі (Long Short-term Memory Recurrent Neural Network). Роботи [4; 5] присвячені проблемам створення програм перевірки орфографії арабської мови і мов бенгальської та хінді. У статті [6] учасниками проекту Vocalizer описано шляхи вирішення проблеми виправлення помилок у тестах, отриманих за допомогою програми оптичного розпізнавання текстів. Засобом перевірки орфографії обрано програму PyEnchant.

Різні методичні та технологічні аспекти навчання студентів програмування розглянуто у працях Семерікова С.О. [7], Спіріна О.М. [8], Литвинової С.Г. [9], Кривоноса О.М. [10] та ін. Але питання методики навчання студентів створення, модифікації та тестування програми перевірки орфографії української мови залишається малодослідженим.

Мета статті – дослідити технології створення програми перевірки орфографії текстів українською мовою та висвітлити методичні аспекти навчання студентів спеціальностей «Прикладна лінгвістика», «Комп'ютерні науки», «Середня освіта (Інформатика)» цієї теми.

Виклад основного матеріалу. Створення професійної програми перевірки орфографії є складним процесом, що передбачає використання потужного математичного апарату, крос-лінгвістичного аналізу, засобів нейронних мереж [5; 11]. Розглянемо програму перевірки орфографії, написану американським ученим Пітером Норвігом (Peter Norvig) [12]. Перевагами цієї програми є простота, зрозумілість, короткий код, точність (75%), швидкість роботи (опрацювання 35-40 слів за секунду). Автор виділяє чотири складові програми:

1. Засіб створення – мова програмування (вибрано Python).
2. Модель кандидата – множина всіх можливих виправлень для заданого слова: видалення (видалити одну букву), транспозиція (переставити дві сусідні букви), заміна (замінити одну букву іншою), вставка (вставити букву). Відповідна функція програми – функція редагування – повертає множину рядків (рядок може бути словом або ні), що отримуються за допомогою одного простого виправлення, тобто відстань редагування дорівнює один.
3. Мовна модель – забезпечення можливості оцінки імовірності слова для виправлення за допомогою підрахунку його частотності в заданому текстовому файлі. Бажано, щоб файл містив близько мільйона слів. Для досягнення цієї мети автор пропонує використовувати великі масиви текстових даних, такі як книги проекту Гутенберг, списки слів Вікісловника, Британський національний корпус.
4. Модель помилки – створення непорожньої множини слів-кандидатів для виправлення у порядку пріоритету:
    1) задане слово, якщо воно відоме; інакше
    2) список відомих слів із відстанню редагування один, якщо вони є; інакше
    3) список відомих слів із відстанню редагування два, якщо вони є; інакше
    4) задане слово, навіть якщо воно невідоме.

Із побудованої множини слів-кандидатів обирається елемент із максимальним значенням імовірності.

Програма написана і працює правильно для англійської мови. Поставимо задачу модифікувати її для опрацювання текстів українською мовою.

Для цього необхідно:

1. Перетворити функцію редагування для роботи з символами українського алфавіту.
2. Для покращення статистичного аналізу результатів роботи програми змінити функцію створення множини слів-кандидатів таким чином, щоб вона повертала значення «None» у разі, якщо варіантів виправлення для слова не знайдено.
3. Дібрати лексичний матеріал для забезпечення мовної моделі та визначити засоби його опрацювання.
4. Побудувати регулярний вираз для виокремлення слів із заданого тексту таким чином, щоб українські слова правильно опрацьовувалися, наприклад, слова з апострофом.

Після виконання поставлених завдань код програми перевірки орфографії, адаптованої для опрацювання текстів українською мовою, матиме вигляд (ім'я файлу spellcheck.py):

```
import re
import codecs
from collections import Counter
from nltk.corpus import PlaintextCorpusReader
from bs4 import BeautifulSoup

def words(text): return re.findall("[\w']+", text.lower())

corpus = PlaintextCorpusReader(r'.\nltk_data\corpora', '.*')
raw = ""
for file in corpus.fileids():
    raw += BeautifulSoup(corpus.raw(file), "lxml").get_text()

WORDS = Counter(words(raw))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    "Імовірність слова."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    "Найімовірніше виправлення для слова."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    "Генерування можливих виправлень для слова."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or
[None])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    "Підмножина можливих варіантів написання слова, які зустрічаються у
словнику WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    "Усі рядки, отримані одним редагуванням слова."
    letters    = 'абвгдежзийклмнопрстуфхцчшщьюяєіїґ'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in
letters]
    inserts    = [L + c + R               for L, R in splits for c in
letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
```

```
    "Усі рядки, отримані двома редагуваннями слова."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

Ключові оператори та функції програми

До функції edits1(word) як параметр передається слово для перебору можливих варіантів написання цього слова при видаленні, перестановці, заміні або вставці одного символу. Для рядка "*житя*" генерується 297 варіантів виправлень:

```
>>> edits1('житя')
{'житян', 'жпитя', 'житхя', 'жфитя', 'жтя', 'житж', 'жищя', 'житб', 'жиуя',
'житяю', 'жиґя', 'житея', 'жеитя', 'жихя', 'іжитя', 'житє', 'житй',
'жиутя', 'жгтя', 'іитя', 'житщ', 'сжитя', 'щжитя', 'житґя', 'жияя',
'житпя', 'житця', 'питя', 'живя', 'житщя', 'витя', 'жижя', 'ритя', 'жжитя',
'жиґтя', 'шитя', 'жиея', 'пжитя', 'жифтя', 'жєтя', 'житяу', 'житв', 'жття',
'жнитя', 'живя', 'житч', 'живтя', 'жигя', 'жкитя', 'жютя', 'житмя', 'щитя',
'житяз', 'жчитя', 'жхтя', 'жиия', 'жилтя', 'житчя', 'жибтя', 'жжтя',
'жиєя', 'житяе', 'жиктя', 'житая', 'кжитя', 'жьтя', 'жктя', 'жшитя',
'жиштя', '**дитя**', 'жртя', 'єитя', 'ждтя', 'итя', 'жмтя', 'фжитя', 'жцтя',
'жится', 'житєя', 'жиітя', 'юитя', 'жбтя', 'цжитя', 'битя', 'ижитя',
'жийтя', 'бжитя', 'житяь', 'житз', 'жихтя', 'житяа', 'жлтя', 'житяд',
'жйитя', 'житящ', 'житл', 'житс', 'жятя', 'житья', 'житкя', 'житю',
'яжитя', 'ьжитя', 'жмитя', 'житяи', 'житія', 'жтия', 'ситя', 'житяй',
'ґитя', '**життя**', 'житшя', 'жстя', 'жибя', 'йжитя', 'жьитя', 'жїитя',
'житр', 'жичтя', 'жптя', 'жчтя', 'житвя', 'жити', 'ґжитя', 'жюитя', 'житґ',
'фитя', 'шжитя', 'тжитя', 'житяф', 'китя', 'жнтя', 'жситя', 'жиптя',
'житяц', 'жвтя', 'житяґ', 'жищтя', 'жия', 'ежитя', 'житгя', 'гитя', 'жетя',
'житяж', 'хитя', 'жиртя', 'жиюя', 'нжитя', 'житял', 'жиая', 'житуя',
'жритя', 'джитя', 'житня', 'житм', 'нитя', 'жиїя', 'жития', 'житяш',
'жиетя', 'ржитя', 'жяитя', 'жхитя', 'жиоя', 'читя', 'ужитя', 'живтя',
'житят', 'житяч', 'жижтя', 'иитя', 'житяр', 'ьитя', 'литя', 'вжитя',
'житяг', 'житйя', 'южитя', 'жикя', 'жиатя', 'житяс', 'йитя', 'житц',
'жбитя', 'жидя', 'житд', 'жвитя', 'житжя', 'хжитя', 'жипя', 'жишя', 'житі',
'єжитя', 'гжитя', 'жимтя', 'житдя', 'житя', 'оитя', 'титя', 'жиля',
'ожитя', 'їитя', 'митя', 'житяв', 'жзтя', 'житяє', 'жштя', 'жиютя',
'жтитя', 'житк', 'житія', 'мжитя', 'жиотя', 'жица', 'житяі', 'жиія',
'ижтя', 'житоя', 'житбя', 'жітя', 'жинтя', 'жиятя', 'жатя', 'житяя',
'житп', 'жить', 'житяб', 'житюя', 'житш', 'жуитя', 'жотя', 'жито', 'жфтя',
'жидтя', 'жйтя', 'жигтя', 'яитя', 'жїтя', 'жзитя', 'житяк', 'зжитя',
'житфя', 'житу', 'аитя', 'жизя', 'житря', 'жичя', 'еитя', 'жиря', 'уитя',
'їжитя', 'жцитя', 'жґитя', 'житт', 'житля', 'жите', 'житг', 'зитя', 'жися',
'житї', 'жіитя', 'жєитя', 'жгитя', 'жиєтя', 'житям', 'житях', 'жутя',
'житх', 'жлитя', 'житяо', 'жифя', 'лжитя', 'жийя', 'жиня', 'житяп', 'жита',
'жимя', 'житф', 'житзя', 'жият', 'жизтя', 'жґтя', 'жоитя', 'жистя', 'цитя',
'жицтя', 'жиїтя', 'жщитя', 'жиитя', 'ждитя', 'жщтя', 'житн', 'жаитя',
'жит', 'ажитя', 'чжитя', 'житяї'}

>>> len(edits1('житя'))
297
```

Функція known(words) вибирає із згенерованої множини тільки ті слова, які є у словнику, що вагомо зменшує кількість варіантів написання слова:

```
>>> known(edits1('житя'))
{'дитя', 'життя'}
```

Функція edits2(word) повертає множину рядків, що отримуються за допомогою двох простих виправлень (відстань редагування дорівнює два). Розмір цієї множини значно більший ніж той, що повертає функція edits1(word). Але тільки декілька елементів є відомими словами:

```
>>> len(set(edits2('житя')))
39332
>>> known(edits2('житя'))
{'житло', 'пити', 'жива', 'дитям', 'житті', 'дитя', 'пиття', 'життя'}
```

Якщо функції candidates(word) передати як параметр рядок '*житя*', то результатом буде множина із двох елементів:

```
>>> candidates('житя')
{'життя', 'дитя'}
```

Для рядка 'абабаґаламаґа' результатом буде значення [None], оскільки такого слова в словнику немає:

```
>>> candidates('абабаґаламаґа')
[None]
```

З урахуванням імовірності слова в тексті функція correction(word) з параметром '*житя*' повертає остаточний варіант виправлення – слово '*життя*':

```
>>> correction('житя')
'життя'
```

Забезпечення мовної моделі

Реалізація мовної моделі передбачає створення словника, для якого припускається, що слова, які входять до його складу, мають правильну орфографічну форму. Релевантним ресурсом для створення такого словника є матеріали «Браунського корпусу української мови» (https://github.com/brown-uk/corpus). На цей момент корпус налічує 752 тексти низької, середньої та високої якості. Робота над корпусом наразі триває. Метою роботи авторів ресурсу є створення відкритого, збалансованого за жанрами та в подальшому проанотованого корпусу сучасної української мови (БрУК) обсягом 1 мільйон унікальних слів.

Сформувати словник бажано на основі текстів середньої та високої якості. Таке поєднання дає змогу створити словник із великою кількістю термінів та мінімізувати кількість потенційних помилок. Для запропонованого дослідження було обрано 686 текстів із загальною кількістю слів 457398 одиниць, кількість унікальних лексичних одиниць склала 81880.

Ефективним засобом зчитування текстів із «Браунського корпусу української мови» є програма «Plaintext Corpus Reader» – доступний ресурс платформи NLTK (Natural Language Toolkit) [13]. До програми «Plaintext Corpus Reader» передаються два параметри:

1) шлях до теки, в якій розташовані тексти корпусу – '.\nltk_data\corpora';

2) маска імен файлів для імпорту – '.*', тобто файли з будь-яким розширенням. Файли, зчитані із корпусу, заносяться у змінну corpus. Список файлів можна отримати за допомогою методу корпусу fileids() [14].

Файли Браунського корпусу української мови мають структуру xml-документа, тому містять не лише текст твору, але і метадані про нього. Для відділення тегів від тексту нами використано бібліотеку «BeautifulSoup» та додатковий синтаксичний аналізатор (парсер) «lxml». При цьому текст метаданих додається до словника, оскільки він містить слова української мови, що можна використати для перевірки правопису так само, як і будь-яке інше слово з основної частини тексту.

Особливості побудови регулярного виразу для виокремлення слів із тексту українською мовою

Створення словника WORDS стає можливим завдяки тому, що функція words(text) виокремлює слова із тексту. Для цього будується регулярний вираз, що використовується як параметр у функції re.findall(). Ця функція входить до бібліотеки модуля регулярних виразів re і повертає список рядків, що відповідають заданому шаблону. У програмі Пітера Норвіга шаблон для виокремлення слів побудований так: '\w+', що означає послідовність букв або

цифр. Тестування програми для текстів українською мовою показало, що для слів з апострофом такий шаблон є непридатним. Розглянемо цю проблему та способи її вирішення.

Регулярні вирази не працюють однаково для всіх мов світу. Оскільки вони розроблялися в англомовному світі, то найкраще адаптовані для англійської мови. Як результат, використання регулярних виразів при роботі з іншими мовами викликає необхідність модифікувати сам вираз та програму в цілому. Наприклад, при розв'язанні задачі знаходження першого повнозначного слова в рядку регулярний вираз '\w+' використовується як аргумент функції пошуку re.search('\w+'). Результатами роботи цієї функції для рядків "doctor's" англійською мовою та "обов'язковий" українською мовою є: "doctor" та "обов" відповідно. Для рядка "обов'язковий" послідовність, визначена шаблоном, закінчується на апострофі, оскільки апостроф не є цифрою або буквою. Рядок "doctor" є повнозначним словом, і для нього можна визнати роботу регулярного виразу коректною. Рядок "обов" є лише частиною слова, отже для "обов'язковий" побудований регулярний вираз не працює.

Необхідно зазначити, що в англійській мові слова складаються лише з літер, а будь-які інші символи в середині слова, відмінні від літер, не є характерними. Винятком може бути дефіс, однак він властивий для складених слів і поєднує два простих слова, що розглядаються окремо як повнозначні. Апостроф в англійській мові використовується для позначення присвійного відмінку, що у більшості випадків утворюється додаванням апострофа та літери "s" до повнозначного слова, тому регулярний вираз, який визначає послідовність до апострофа, повертає повнозначне слово, не створюючи при цьому помилок у роботі програми. Для української мови характерні слова з апострофом у середині слова, при цьому частини слова, розділеного апострофом, часто не є повнозначними, а тому не можуть бути використані в подальшому опрацюванні.

Способи вирішення проблеми:

1. Додавання апострофу до наявного регулярного виразу – "[\w']+", що означає послідовність літер і апостроф. Результатами застосування цього виразу для обробки рядків, наведених вище, є: "doctor's" та "обов'язковий". Така модифікація вирішує проблему для української мови, але вносить небажані зміни в результат для англійської мови.
2. Побудова регулярного виразу "[\u0400-\u04FF\']+ ". \u0400-\u04FF, що означає послідовність кодів символів Unicode, яка відповідає кирилиці. Результатами є слово "обов'язковий" для української мови та порожній рядок "" для англійської мови, оскільки жодна з літер англійського алфавіту не відповідає заданому діапазону символів.
3. Об'єднання двох регулярних виразів в один: "[\u0400-\u04FF\']+|\w+", що означає послідовність літер кирилиці і апостроф або послідовність літер латиниці. Результати роботи цього виразу для розглядуваних рядків – "doctor" та "обов'язковий". Обидва результати є коректними, але запропонований регулярний вираз має недоліки: швидкість роботи програми зменшується зі збільшенням кількості мов, що опрацьовуються, а також робота з виразом для кількох мов ускладнюється через нагромадження символів та операторів у регулярному виразі.
4. Використання регулярного виразу відповідно до мови, що опрацьовується, оскільки тексти, в яких наявні кілька мов, є рідкістю.

В аналізованій програмі перевірки орфографії використано регулярний вираз, описаний у пункті 1, а саме "[\w']+".

Наступний крок дослідження – застосування програми до великих текстових масивів та аналіз отриманих результатів.

Застосування програми перевірки орфографії

У ролі матеріалу для перевірки орфографії та аналізу результатів використано тексти, що містять субтитри українською мовою. Ці тексти створено у межах проекту «To Be Announced» – волонтерської програми, учасники якої здійснюють переклад англомовних

серіалів українською мовою та створюють субтитри на основі цих перекладів. Мета втілення проекту – збільшити обсяг україномовного контенту та зробити його більш доступним для усіх верств населення. Тексти, загальним обсягом 289795 слів, включають матеріали різних функціональних стилів, а також велику кількість субстандартної лексики та власних назв.

Для опрацювання текстового масиву було створено програму, що виконує такі функції:

- імпорт функції correction() із програми перевірки орфографії (ім'я файлу spellcheck.py);
- читання з файлу тексту для перевірки;
- виділення окремих слів;
- перевірка правильності написання згідно зі словником ;
- запис результату у файл: якщо слово написано неправильно і програма не знаходить варіант виправлення, у файл записується тільки слово; якщо слово написано неправильно і програма пропонує варіант виправлення, у файл записується слово та варіант виправлення. Для полегшення пошуку та виправлення помилок кожний рядок у підсумковому файлі починається з номера рядка у заданому файлі;
- виведення на екран кількості опрацьованих рядків – для відстеження процесу роботи програми в режимі реального часу;
- виведення на екран загального часу виконання програми – після завершення обробки вихідного файлу та запису в файл результату, виводиться на екран час виконання програми.

Повний текст описаної програми є таким (ім'я файлу fileChecker.py):

```
from spellcheck import correction
from re import finditer
from time import time

startTime = time()

SEGMENTS_FILE = 'tba_reasons.txt'
RESULT_FILE = 'corrections.txt'

with open(SEGMENTS_FILE, encoding="utf-8") as f, open(RESULT_FILE, 'w',
encoding='utf-8') as f2:
 lines = [(number, line) for number, line in enumerate(f)]
 fileLength = len(lines)
 for number, line in lines:
       for m in finditer("[\w']+", line.casefold()):
             corrected = correction(m.group(0))
             if corrected == None:
                   f2.write('%d. %s\n' % (number, m.group(0)))
             elif corrected != m.group(0):
                   f2.write('%d. %s --- %s\n' % (number, m.group(0),
corrected))

       print("%d із %d рядків опрацьовано" % (number + 1, fileLength))

print(time() - startTime)
```

Аналіз отриманих результатів.

Проведемо аналіз результатів роботи програми fileChecker.py на основі тексту субтитрів до серіалу «13 Reasons Why» (табл. 1):

Таблиця № 1.

Результати роботи програми fileChecker.py

Загальний обсяг тексту	23721 слово

Час виконання програми	1167.1577906608582 секунд або 19 хвилин 45 секунд
Загальна кількість виправлень	2649
Кількість унікальних помилок	1403
Кількість повторів помилок	1246

Продовження таблиці № 1.

Кількість справжніх помилок	30
Кількість помилкових виправлень (наприклад, бредлі --- брехні, відповіси --- відповісти, бісові --- лісові)	1373
Кількість слів, для яких запропоновано виправлення	2303
Кількість слів, для яких не запропоновано виправлення	346

Із результатів роботи програми видно, що в заданому тексті обсягом 23721 слово знайдено 2649 помилок (11,16% від загального обсягу тексту), з яких 1403 одиниці є унікальними (52,96% від загальної кількості помилок), а 1246 (47,03%) є дублікатами. Лише 30 слів (0,1265%) є справжніми помилками, а інші слова відсутні у створеному словнику для перевірки орфографії. Це зумовлено трьома ключовими факторами:

1. українська мова – це синтетична мова і форми слів утворюються за допомогою префіксів, суфіксів та закінчень. Тому для правильної роботи програми словник повинен містити всі форми слів. Однак, обсяг такого словника буде занадто великим і робота з ним буде тривалою та непродуктивною. Хоча існують і розробляються словники, які містять слова та дані щодо створення словоформ, а самі форми слова генеруються на програмному рівні. Однак програмне забезпечення для генерування таких словоформ, написане мовою програмування Python, відсутнє;
2. жанр тексту – оскільки у субтитрах відтворюється переважно спілкування молоді між собою, текст містить багато субстандартної лексики, переважно сленгу (чуваче, пресуха), що не включено до словника перевірки орфографії;
3. регіон, в якому відбуваються події (США) – не є регіоном використання мови, що перевіряється (Україна), а тому більшість власних назв, що зустрічаються в тексті (Ханна, Джастін) та їхні словоформи відсутні в словнику для перевірки орфографії;

Помилки, знайдені програмою, – це, переважно, помилки друку, в яких або відсутні літери (важали), присутні зайві літери (виихід) або присутні неправильні літери (щанс).

Приклади запропонованих програмою виправлень для помилок:

- принаймі --- принаймні;
- щанс --- шанс;
- кожно --- кожна;
- важали --- вважали;
- виихід --- вихід;
- назавждиусерцях;
- того_ --- того;
- щась --- щось;
- пішоло --- пішло;
- відбується --- відбувається.

У більшості випадків програма пропонує можливе виправлення, що є правильним у цьому контексті, окрім слова «назавждиусерцях», оскільки це три слова, які злилися в одне.

Висновки. Дослідження показало, що робота студентів над програмою перевірки орфографії не зводиться до простого кодування відомого алгоритму, а передбачає виконання діяльності, що можна розглядати як процес розробки програмного забезпечення, обчислювальний експеримент. Виконання такого виду діяльності сприяє фундаменталізації інформатичної освіти [15].

Навчання студентів створенню програми перевірки орфографії ґрунтується на компетентністному підході, вимагає впровадження дослідницької діяльності, прийняття рішень у нестандартних ситуаціях. У процесі роботи над програмою формується та розвивається дослідницька компетентність, складовими якої є: аналіз стану дослідження, оцінювання інформаційних джерел, розпізнавання суперечливих результатів, формулювання гіпотез, планування дослідницького процесу, вибір і застосування методів та засобів дослідження, розуміння практичної значущості дослідження, навички спілкування, презентація результатів дослідження тощо. Проведення таких досліджень допомагає формуванню уявлень студентів про майбутню професію.

Перспективи подальших досліджень вбачаються в удосконаленні створених програм, а саме: 1) визначенні можливостей опрацювання різних форм українських слів; 2) виявленні шляхів пришвидшення роботи програм. Також необхідним є подальший пошук лексичного матеріалу для їх тестування та аналізу результатів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Doyle, A. (2018). *The balance careers. Software Engineer Skills List.* Retrieved from https://www.thebalancecareers.com/software-engineer-skills-list-2062483.
2. IT career finder (2018). *Computer Programmer*. Retrieved from https://www.itcareerfinder.com/it-careers/computer-programmer.html.
3. Gudmundsson, J. & Menkes, F. (2018). *Swedish Natural Language Processing with Long Short-term Memory Neural Networks: A Machine Learning-powered Grammar and Spell-checker for the Swedish Language*. Retrieved from http://lnu.diva-portal.org/smash/get/diva2:1232482/FULLTEXT01.pdf.
4. Nejja, M. & Yousfi, A. (2018). *The vocabulary and the morphology in spell checker*. Retrieved from https://www.sciencedirect.com/science/article/pii/S187705091830111X#.
5. Choudhury, M., Thomas, M., Mukherjee, A., Basu, A. & Ganguly, N. (2007). *How difficult is it to develop a perfect spell-checker? A Cross-linguistic Analysis through Complex Network Approach*. Retrieved from https://arxiv.org/abs/physics/0703198.
6. Cappelatti, E., De Oliveira Heidrich, R., Oliveira, R., Monticelli, C., Rodrigues, R., Goulart, R. & Velho, E. (2018). *Post-correction of OCR Errors Using PyEnchant Spelling Suggestions Selected Through a Modified Needleman–Wunsch Algorithm.* Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-92270-6_1.
7. Стрюк, А. М., Семеріков, С. О. & Тарасов, І. В. (2015). Компетентність бакалавра інформатики з програмування. *Інформаційні технології і засоби навчання*, 2(46), 91-108. Відновлено з http://lib.iitta.gov.ua/11134/1/1225-4544-1-PB.pdf.
8. Спірін, О. М., & Вакалюк, Т. А. (2017). Критерії добору відкритих Web-орієнтованих технологій навчання основ програмування майбутніх учителів інформатики. *Інформаційні технології і засоби навчання*, 4 (60), 275-287. Відновлено з https://journal.iitta.gov.ua/index.php/itlt/article/view/1815.
9. Проскура, С. Л. & Литвинова, С. Г. (2018). Підготовка фахівців з інформаційних технологій у закладах вищої освіти: стан, проблеми і перспективи. *Інформаційні технології в освіті*, 2(35), 72-88. Відновлено з http://ite.kspu.edu/issue_35/p-72-88.

10. Кривонос, О. М. (2014). Використання задачного підходу в процесі навчання програмування майбутніх учителів інформатики. *Інформаційні технології і засоби навчання*, 2(40), 83-91. Відновлено з https://journal.iitta.gov.ua/index.php/itlt/article/view/1005.
11. Chollampatt, S. & Hwee Tou Ng. (2018). *A Multilayer Convolutional Encoder-Decoder Neural Network for Grammatical Error Correction.* Retrieved from https://arxiv.org/abs/1801.08831.
12. Norvig, P. (2016). *How to Write a Spelling Corrector*. Retrieved from http://norvig.com/spell-correct.html.
13. NLTK 3.4 documentation (2018). *Source code for nltk.corpus.reader.plaintext.* Retrieved from http://www.nltk.org/_modules/nltk/corpus/reader/plaintext.html.
14. NLTK 3.4 documentation (2018). *Source code for nltk.corpus.reader.api.* Retrieved from https://www.nltk.org/_modules/nltk/corpus/reader/api.html#CorpusReader.fileids.
15. Семеріков, С. О. (2009). *Теоретико-методичні основи фундаменталізації навчання інформатичних дисциплін у вищих навчальних закладах* (автореф. дис. ... д-ра пед. наук : 13.00.02). Нац пед. ун-т ім. М. П. Драгоманова, Київ. Відновлено з http://enpuir.npu.edu.ua/bitstream/123456789/226/3/Semerikov.pdf.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Doyle, A. (2018). *The balance careers. Software Engineer Skills List.* Retrieved from https://www.thebalancecareers.com/software-engineer-skills-list-2062483.
2. IT career finder (2018). *Computer Programmer*. Retrieved from https://www.itcareerfinder.com/it-careers/computer-programmer.html.
3. Gudmundsson, J. & Menkes, F. (2018). *Swedish Natural Language Processing with Long Short-term Memory Neural Networks: A Machine Learning-powered Grammar and Spell-checker for the Swedish Language*. Retrieved from http://lnu.diva-portal.org/smash/get/diva2:1232482/FULLTEXT01.pdf.
4. Nejja, M. & Yousfi, A. (2018). *The vocabulary and the morphology in spell checker*. Retrieved from https://www.sciencedirect.com/science/article/pii/S187705091830111X#.
5. Choudhury, M., Thomas, M., Mukherjee, A., Basu, A. & Ganguly, N. (2007). *How difficult is it to develop a perfect spell-checker? A Cross-linguistic Analysis through Complex Network Approach*. Retrieved from https://arxiv.org/abs/physics/0703198.
6. Cappelatti, E., De Oliveira Heidrich, R., Oliveira, R., Monticelli, C., Rodrigues, R., Goulart, R. & Velho, E. (2018). *Post-correction of OCR Errors Using PyEnchant Spelling Suggestions Selected Through a Modified Needleman–Wunsch Algorithm.* Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-92270-6_1.
7. Striuk, A., Semerikov, S. & Tarasov, I. (2015). Bachelor of informatics competence in programming. *Information Technologies and Learning Tools*, 2(46), 91-108. Retrieved from http://lib.iitta.gov.ua/11134/1/1225-4544-1-PB.pdf.
8. Spirin, O., & Vakaliuk, T. (2017). Criteria of open web-operated technologies of teaching the fundamentals of programs of future teachers of informatics. *Information Technologies and Learning Tools*, 4 (60), 275-287. Retrieved from https://journal.iitta.gov.ua/index.php/ itlt/article/view/1815.
9. Proskura, S. & Lytvynova, S. (2018). Information technologies specialists training in higher education institutions of ukraine: general state, problems and perspectives. *Information Technologies in Education*, 2(35), 72-88. Retrieved from http://ite.kspu.edu/issue_35/p-72-88.
10. Kryvonos, O. (2014). Using of task approach method while teaching programming to the future informatics teachers. Information Technologies and Learning Tools, 2(40), 83-91. Retrieved from https://journal.iitta.gov.ua/index.php/itlt/article/view/1005.
11. Chollampatt, S. & Hwee Tou Ng. (2018). *A Multilayer Convolutional Encoder-Decoder Neural Network for Grammatical Error Correction.* Retrieved from https://arxiv.org/abs/1801.08831.

12. Norvig, P. (2016). *How to Write a Spelling Corrector*. Retrieved from http://norvig.com/spell-correct.html.
13. NLTK 3.4 documentation (2018). *Source code for nltk.corpus.reader.plaintext.* Retrieved from http://www.nltk.org/_modules/nltk/corpus/reader/plaintext.html.
14. NLTK 3.4 documentation (2018). *Source code for nltk.corpus.reader.api.* Retrieved from https://www.nltk.org/_modules/nltk/corpus/reader/api.html#CorpusReader.fileids.
15. Semerikov, S. (2009). *Theoretical and methodic foundations of fundamentalization teaching of the Computer Science at the high educational institutions* (abstract of Doctor's of Pedagogical Sciences Thesis). National Dragomanov Pedagogical University, Kyiv. Retrieved from http://enpuir.npu.edu.ua/bitstream/123456789/226/3/Semerikov.pdf.

Стаття надійшла до редакції 25.03.2019.
The article was received 25 March 2019.

Olga Riezina[1], Roman Kosiuh[2]
[1] Volodymyr Vynnychenko Central Ukrainian State Pedagogical University, Kropyvnytskyi, Ukraine
[2] AVA.codes Company, Kyiv, Ukraine

TECHNOLOGIES OF CREATING SPELL CHECKER

Spell checkers are created to control and correct mistakes in a user document. They are based on the comparison of every word against the spelling dictionary and on the use of correct spelling detection algorithms. The article dwells on technologies of creating spell checker, as well as methods of teaching this technology. Spell checker by Peter Norvig has been studied. Modifications for this program necessary to process Ukrainian texts have been defined. Approach to implementation of language model, that is creating spelling dictionary, based on the Ukrainian Brown Corpus has been suggested. Peculiarities of designing a regular expression for distinguishing words in Ukrainian text have been defined. Texts containing Ukrainian subtitles, created within the volunteer translation project «To Be Announced», have been used as a means of test material for the spell checker. The program that processes this text material in order to check spelling has been described and the obtained results have been analysed. The obtained resulted were concluded to be correct, which encourages further research.

Keywords: spell checker; language model; regular expression; programming language Python; teaching programming.