ДОДАТОК А

Текст програми

ГЮИК. 501310.002– 01 12 01

(позначення документу)

ЗАТВЕРДЖЕНО
ГЮИК.501310.002 – 01 12 01

Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту

Текст програми

ГЮИК. 501310.002 – 01 12 01

АРКУШІВ 14

2020 р.

Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник атестаційної роботи
_____ Колесник Л.В.

Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
ГЮИК. 501310.002 – 01 12 01–ЛУ

РОЗРОБИЛА:
ст. гр. ІТПм-19-1
_____ Волобуєва В.В.

2020 р

Реалізація алгоритму стеммера Портера:

```
import java.util.regex.Matcher;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Porter {

  private static final Pattern PERFECTIVEGROUND1 = Pattern.compile("((ив|ыв
|ившись|ивши|ывши|ывшись)|((?&lt;=[ая])(в|вши|вшись)))$");

  private static final Pattern REFLEXIVE2 = Pattern.compile("(с[яь])$");

  private static final Pattern ADJECTIVE_V =
Pattern.compile("(ее|ие|ые|ей|ими|ыми|ое|ий|ый|ой|ем|им|ым|ом|его|ого|ему|ому|ых|и
х|ую|юю|яя|ая|ою|ею)$");

  private static final Pattern PARTICIPLE_V =
Pattern.compile("((ывш|ивш|ующ)|((?<=[ая])(ем|вш|нн|ющ|щ)))$");

  private static final Pattern VERB1 =
Pattern.compile("((ыла|ила|ена|ейте|уйте|ите|или|ыли|ей|уй|ил|ыл|им|ым|ен|ило|ыло|е
но|ят|уют|ует|ит|ыт|ены|ыть|ить|ишь|ую|ю)|((?<=[ая])(на|ла|ете|йте|ли|й|л|ем|н|ло|но|
ет|ют|ны|ть|ешь|нно)))$");

  private static final Pattern NOUN_V =
Pattern.compile("(а|ев|ов|ье|ие|е|иями|ями|ами|еи|ии|и|ией|ей|ой|ий|й|иям|ям|ием|ем|а
м|ом|о|у|ах|иях|ях|ы|ь|ию|ю|ью|ия|ья|я)$");
```

```
  private static final Pattern RVRE3 = Pattern.compile("^(.*?[аеиоуыэюя])(.*)$");

  private static final Pattern DERIVATIONAL4 =
Pattern.compile(".*[^аеиуэоюяы]+[аеоуиыюяэ].*ость?$");

  private static final Pattern DER_V = Pattern.compile("ость?$");

  private static final Pattern SUPERLATIVE5 = Pattern.compile("(ейш|ейше)$");

  private static final Pattern I1 = Pattern.compile("и$");
  private static final Pattern NN = Pattern.compile("нн$");
  private static final Pattern PP = Pattern.compile("ь$");

  public String stem(String word) {
    word = word.toLowerCase();
    word = word.replace('ё', 'е');
    Matcher m = RVRE3.matcher(word)
;
    if (m.matches()) {
      String pre = m.group(1);
      String rv1 = m.group(2);
      String temp = PERFECTIVEGROUND1.matcher(rv).replaceFirst("");
      if (temp.equals(rv)) {
        rv = REFLEXIVE2.matcher(rv).replaceFirst("");
        temp = ADJECTIVE_V.matcher(rv).replaceFirst("");
        if (!temp.equals(rv)) {
          rv = temp;
          rv = PARTICIPLE_V.matcher(rv).replaceFirst("");
        } else {
          temp = VERB1.matcher(rv).replaceFirst("");
```

```
        if (temp.equals(rv)) {
          rv = NOUN_V.matcher(rv).replaceFirst("");
        } else {
          rv = temp;
        }
       }

     } else {
       rv = temp;
     }

     rv = I.matcher(rv).replaceFirst("");

     if (DERIVATIONAL4.matcher(rv).matches()) {
       rv = DER_V.matcher(rv).replaceFirst("");
     }

     temp = P.matcher(rv).replaceFirst("");
     if (temp.equals(rv)) {
       rv = SUPERLATIVE5.matcher(rv).replaceFirst("");
       rv = NN.matcher(rv).replaceFirst("н");
     } else {
       rv = temp;
     }
     word = pre + rv;
   }
   return word;
  }
}
package com.company;
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {

  private static final Pattern PERFECTIVEGROUND1 =
Pattern.compile("((ивши|ив|ившись|ыв|ывши|ывшись)|((?<=[ая])(в|вши|вшись)))$");

  private static final Pattern REFLEXIVE2 = Pattern.compile("(с[яь])$");

  private static final Pattern ADJECTIVE_V =
Pattern.compile("(ее|ие|ые|ое|ыми|ими|ий|ей|ый|ой|ем|им|ым|ом|его|ого|ему|ому|их|ы
х|ую|юю|яя|ая|ою|ею)$");

  private static final Pattern PARTICIPLE_V =
Pattern.compile("((ивш|ывш|ующ)|((?<=[ая])(ем|нн|вш|ющ|щ)))$");

  private static final Pattern VERB1 =
Pattern.compile("((ила|ыла|ена|уйте|ейте|ите|или|ыли|ей|уй|ил|ыл|ым|им|ен|ыло|ило|е
но|ят|ует|уют|ит|ыт|ены|ить|ыть|ишь|ую|ю)|((?<=[ая])(ла|на|ете|йте|ли|й|л|ем|н|ло|но|
ет|ют|ны|ть|ешь|нно)))$");

  private static final Pattern NOUN_V =
Pattern.compile("(а|ев|ов|ие|ье|е|иями|ями|ами|еи|ии|и|ией|ой|ей|ий|й|иям|ям|ием|ам|е
м|о|ом|у|ах|иях|ях|ы|ь|ию|ью|ю|ья|ия|я)$");

  private static final Pattern RVRE3 = Pattern.compile("^(.*?[аиеоуыюэя])(.*)$");
```

```
  private static final Pattern DERIVATIONAL4 =
Pattern.compile(".*[^аеиуоыэяю]+[аиоеуэюыя].*ость?$");

  private static final Pattern DER_V = Pattern.compile("ость?$");

  private static final Pattern SUPERLATIVE5 = Pattern.compile("(ейше|ейш)$");

  private static final Pattern I1 = Pattern.compile("и$");
  private static final Pattern PP = Pattern.compile("ь$");
  private static final Pattern NN = Pattern.compile("нн$");

  public String stem(String word) {
    word = word.toLowerCase();
    word = word.replace('ё', 'е');
    Matcher m = RVRE.matcher(word);
    if (m.matches()) {
      String pre = m.group(1);
      String rv1 = m.group(2);
      String temp = PERFECTIVEGROUND1.matcher(rv).replaceFirst("");
      if (temp.equals(rv1)) {
        rv = REFLEXIVE2.matcher(rv).replaceFirst("");
        temp = ADJECTIVE_V.matcher(rv).replaceFirst("");
        if (!temp.equals(rv)) {
          rv = temp;
          rv = PARTICIPLE_V.matcher(rv).replaceFirst("");
        } else {
          temp = VERB1.matcher(rv).replaceFirst("");
          if (temp.equals(rv)) {
            rv = NOUN_V.matcher(rv).replaceFirst("");
          } else {
```

```
            rv = temp;
          }
        }
      } else {
        rv = temp;
      }
      rv = I.matcher(rv).replaceFirst("")
      if (DERIVATIONAL4.matcher(rv).matches()) {
        rv = DER_V.matcher(rv).replaceFirst("");
      }
      temp = P.matcher(rv).replaceFirst("");
      if (temp.equals(rv)) {
        rv = SUPERLATIVE5.matcher(rv).replaceFirst("");
        rv = NN.matcher(rv).replaceFirst("н");
      }else{
        rv = temp;
      }
      word = pre + rv;
    }
    return word;
  }
  public static void main(String[] args) {
    System.out.println(new Main().stem("бумажечкой"));
  }
}
```

Реалізація алгоритму TrieMap:

```
 public class TrieMapv {
 private Object[] mChars = new Object[256];
 private Object mPrefixVal; // Use only for values of prefixes keys.

// Simple container for string-value pairs.
private static class TrieMapv {
  public String mStr;
  public Object mVal;
  public Leafv(String str, Object val) {
    mStr = str;
    mVal = val;
  }
}

public TrieMapv() {
}

public boolean isEmpty() {
  if(mPrefixVal != null) {
    return false;
  }
  for(Object o : mChars) {
    if(o != null) {
      return false;
    }
  }
  return true;
}
```

```
  /**
   * Inserts a key/value pair.
   *
   * @param keys may be empty or contain low-order char 0..255 must not be null.
   * @param val Your data. Any data class except another TrieMap. Null values erase entries.
   */
  public void put(String key, Object val) {
    assert key != null;
    assert !(val instanceof TrieMapv); // Only we get to store TrieMap nodes. TODO: Allow it.
    if(key.length() == 0) {
      // All of the original key's chars have been nibbled away
      // which means this node will store this key as a prefix of other keys.
      mPrefixVal = val; // Note: possibly removes or updates an item.
      return;
    }
    char c = key.charAt(0);
    Object cObj = mChars[c];
    if(cObj == null) { // Unused slot means no collision so just store and return;
      if(val == null) {
        return; // Don't create a leaf to store a null value.
      }
      mChars[c] = new Leafv(key, val);
      return;
    }
    if(cObj instanceof TrieMapv) {
      // Collided with an existing sub-branch so nibble a char and recurse.
      TrieMapv childTrie = (TrieMapv)cObj;
```

```
    childTrie.put(key.substring(1), val);
    if(val == null && childTrie.isEmpty()) {
      mChars[c] = null; // put() must have erased final entry so prune branch.
    }
    return;
  }
  // Collided with a leaf
  if(val == null) {
    mChars[c] = null; // Null value means to remove any previously stored value.
    return;
  }
  assert cObj instanceof Leafv;
  // Sprout a new branch to hold the colliding items.
  Leafv cLeaf = (Leafv)cObj;
  TrieMapv branch = new TrieMapv();
  branch.put(key.substring(1), val); // Store new value in new subtree.
  branch.put(cLeaf.mStr.substring(1), cLeaf.mVal); // Plus the one we collided with.
  mChars[c] = branch;
}


/**
* Retrieved a value for a given key or null if not founded.
*/
public Object get(String key) {
  assert key != null;
  if(key.length() == 0) {
    // All of the original key's chars have been nibbled away
    // which means this key is a prefix of another.
    return mPrefixVal;
```

```
  }
  char c = key.charAt(0);
  Object cVal = mChars[c];
  if(cVal == null) {
    return null; // Not found.
  }
  assert cVal instanceof Leafv || cVal instanceof TrieMapv;
  if(cVal instanceof TrieMapv) { // Hash collision. Nibble first char, and recurse.
    return ((TrieMapv)cVal).get(key.substring(1));
  }
  if(cVal instanceof Leafv) {
    // cVal contains a user datum, but does the key match its substring?
    Leafv cPair = (Leafv)cVal;
    if(key.equals(cPair.mStr)) {
      return cPair.mVal; // Return user's data value.
    }
  }
  return null; // Not founded.
}

/**
* Simple example test programm.
*/
public static void main(String[] args) {
  // Inserted the bunch of key or value pairs.
  TrieMapv trieMapv = new TrieMapv();
  trieMapv.put("123", "456");
  trieMapv.put("Mystem", "Test");
  trieMapv.put("noun_v1", "too");
  trieMapv.put("noo", "cow"); // Will collided with " noun_v1".
```

```
    trieMapv.put("noo ", "walk"); // Collide with " noun_v1" and turn "noo" into a prefix.
    trieMapv.put("", "Root"); // You can store 1 value at the empty key if you like.

    // Test for inserted, nonexistent, and deleted keys.
    System.out.println("123 = " + trieMapv.get("123"));
    System.out.println("Mystem = " + trieMapv.get("Mystem "));
    System.out.println("noun_v1= " + trieMapv.get("noun_v1"));
    System.out.println("noo = " + trieMapv.get("noo "));
    System.out.println("Noon = " + trieMapv.get("Noon"));
    System.out.println("No = " + trieMapv.get("No")); // Should return null.
    System.out.println("Empty key = " + trieMapv.get("")); // Should return "Root".
    System.out.println("Noose = " + trieMapv.get("Noose")); // Never add so should
return null.
    System.out.println("Nothing = " + trieMap.get("Nothing")); // Ditto.
    trieMapv.put("123", null); // Removes this leaf entry.
    System.out.println("After removal, 123 = " + trieMapv.get("123")); // Should now
return null.
    trieMapv.put("noo ", null); // Removes this prefix entry. (Special case to test internal
logic).
    System.out.println("After removal, noo = " + trieMapv.get("noo ")); // Should now
return null.
    trieMapv.put("Noon", null); // Internal test of branch pruning.
  }
}
```

Реалізація алгоритму MyStem:

```
import ru.stachek66.nlp.mystem.holding.Factory;
import ru.stachek66.nlp.mystem.holding.MyStem;
import ru.stachek66.nlp.mystem.holding.MyStemApplicationException;
import ru.stachek66.nlp.mystem.holding.Request;
import ru.stachek66.nlp.mystem.model.Info;
import scala.Option;
import scala.collection.JavaConversions;

import java.io.File;

public class MyStemJavaExample {
  private final static MyStem mystemAnalyzer =
      new Factory("-igd --eng-gr --format json --weight")
          .newMyStem("3.0", Option.<File>empty()).get();
  public static void main(final String[] args) throws MyStemApplicationException {

    final Iterable<Info> result =
        JavaConversions.asJavaIterable(
            mystemAnalyzer
                .analyze(Request.apply("И вырвал грешный мой язык"))
                .info()
                .toIterable());

    for (final Info info : result) {
      System.out.println(info.initial() + " -> " + info.lex() + " | " + info.rawResponse());
    }
  }
}
```

ДОДАТОК Б

Графічний матеріал атестаційної роботи

ГЮИК. 501310.002

(позначення документу)

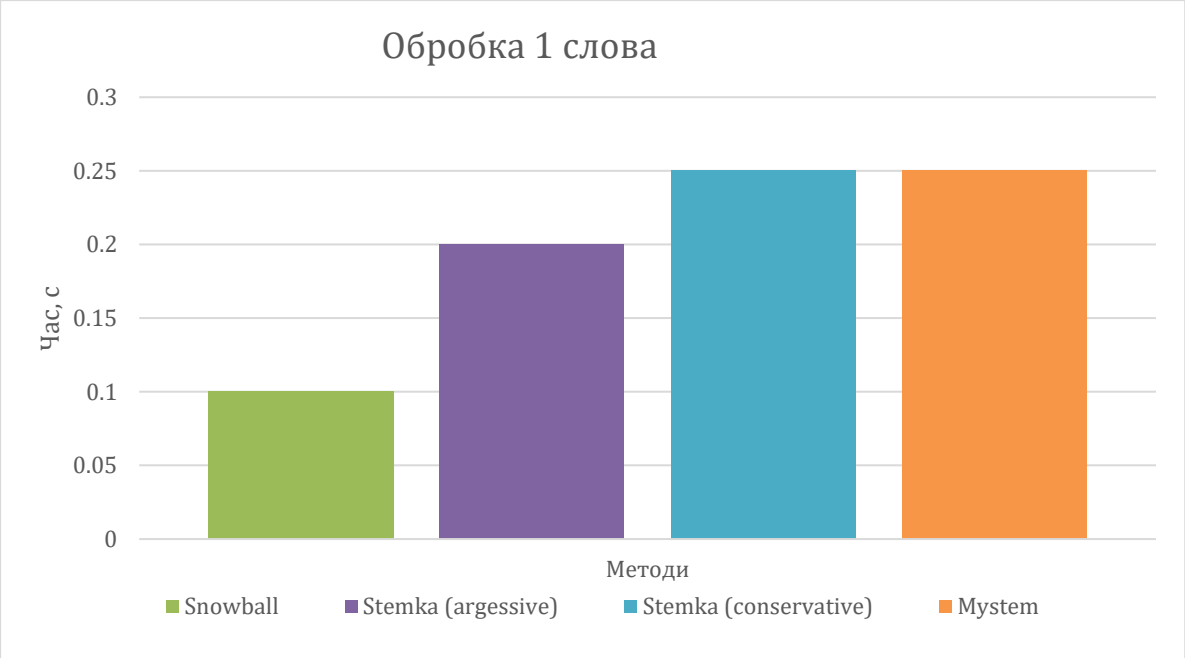
ГЮИК.501310.002 С10

					ГЮИК.501310.002 С10	
Змін.	Аркуш	№ документа	Підпис	Дата		
Розробила		Волобуєва В.В.			Алгоритм роботи стеммінга Портера	Літ. / Маса / Масштаб
Перевірив		Колесник Л.В.				
Т. Контр.						Аркуш 1 / Аркушів 1
Реценз.						
Н. Контр.		Колесник Л.В.				ХНУРЕ Кафедра СТ
Затвердив		Гребеннік І.В.				

ГЮИК.501310.002 С10

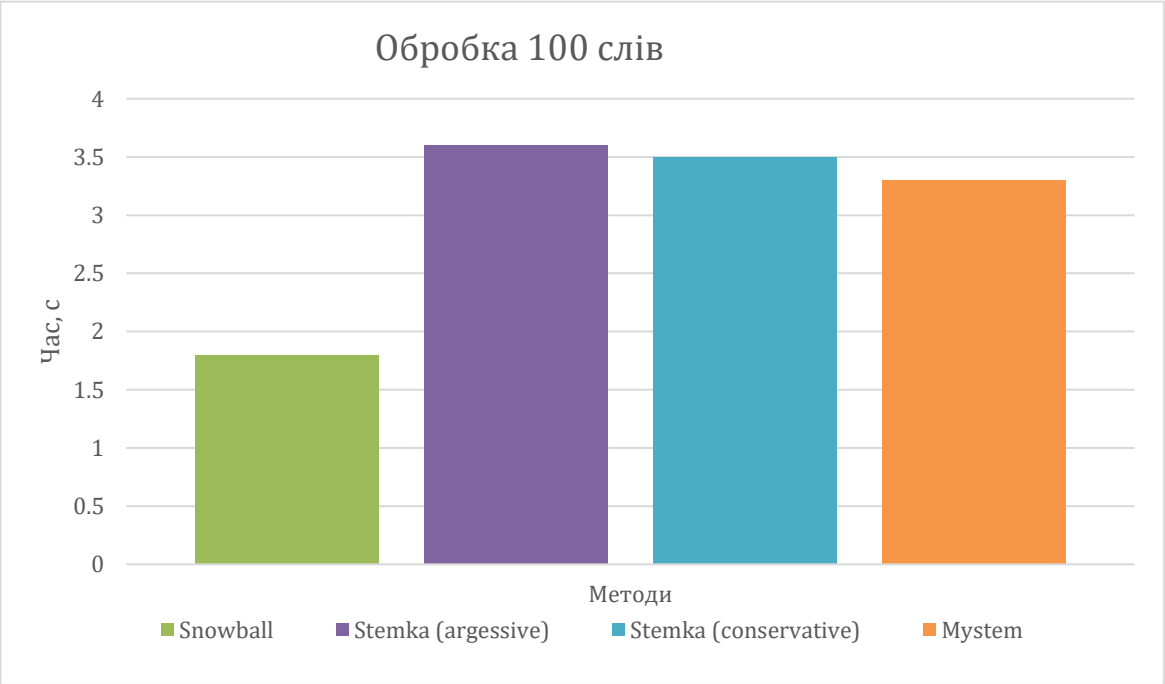
					ГЮИК.501310.002 С10		
					Алгоритм роботи методу Mystem	Літ.	Маса / Масштаб
Змін.	Аркуш	№ документа	Підпис	Дата			
Розробила		Волобуєва В.В.					
Перевірив		Колесник Л.В.				Аркуш 1	Аркушів 1
Т. Контр.							
Реценз.						ХНУРЕ Кафедра СТ	
Н. Контр.		Колесник Л.В.					
Затвердив		Гребеннік І.В.					

РЕЗУЛЬТАТИ ВИМІРЮВАННЯ ШВИДКОСТІ ЗНАХОДЖЕННЯ СТЕММИ ДЛЯ 1 СЛОВА



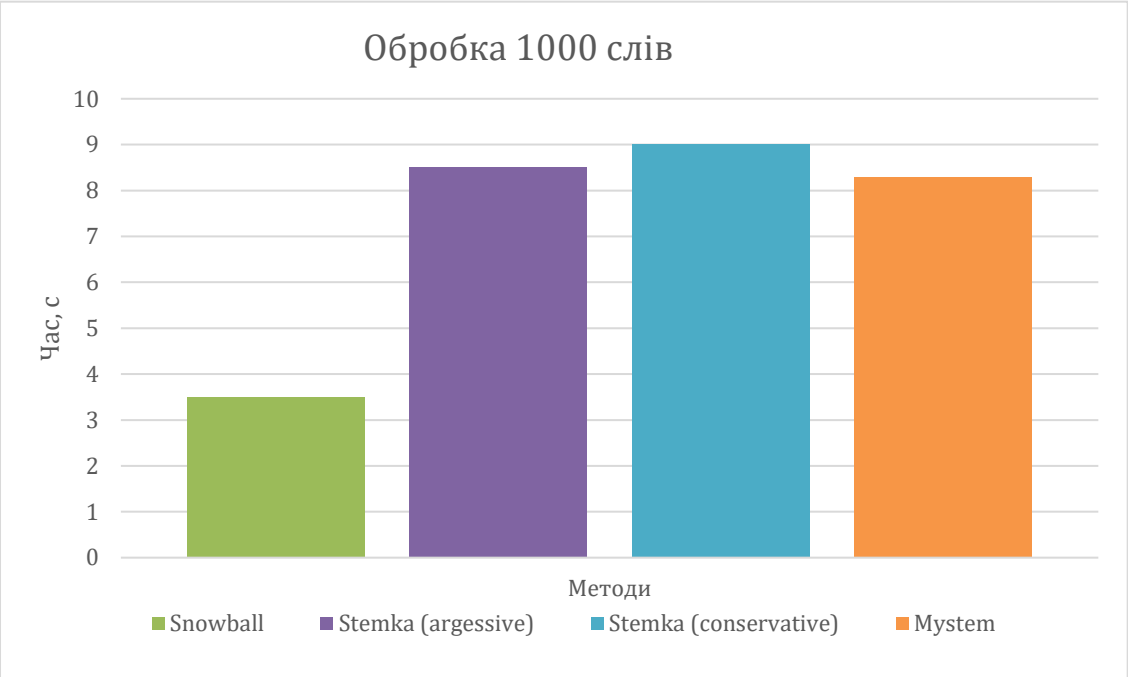
Розробила	Волобуєва В.В.			Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту	
Перевірила	Колесник Л.В.				
Н. Контр.	Колесник Л.В.				
				ІТПм-19-1	Аркуш 1
Затвердив	Гребеннік І.В.			СТ	Аркушів 1

РЕЗУЛЬТАТИ ВИМІРЮВАННЯ ШВИДКОСТІ ЗНАХОДЖЕННЯ СТЕММИ ДЛЯ 100 СЛІВ



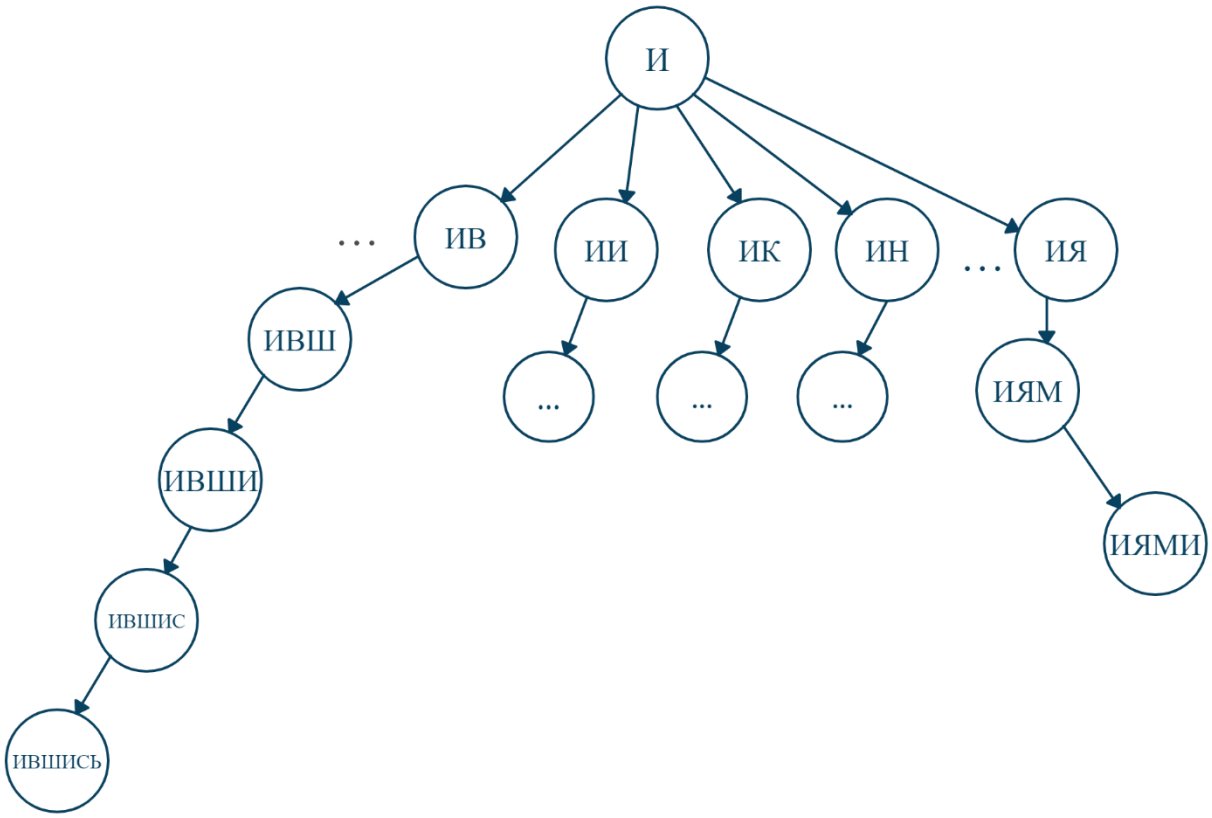
Розробила	Волобуєва В.В.			Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту	
Перевірила	Колесник Л.В.				
Н. Контр.	Колесник Л.В.				
				ІТПм-19-1	Аркуш 1
Затвердив	Гребеннік І.В.			СТ	Аркушів 1

РЕЗУЛЬТАТИ ВИМІРЮВАННЯ ШВИДКОСТІ ЗНАХОДЖЕННЯ СТЕММИ ДЛЯ 1000 СЛІВ



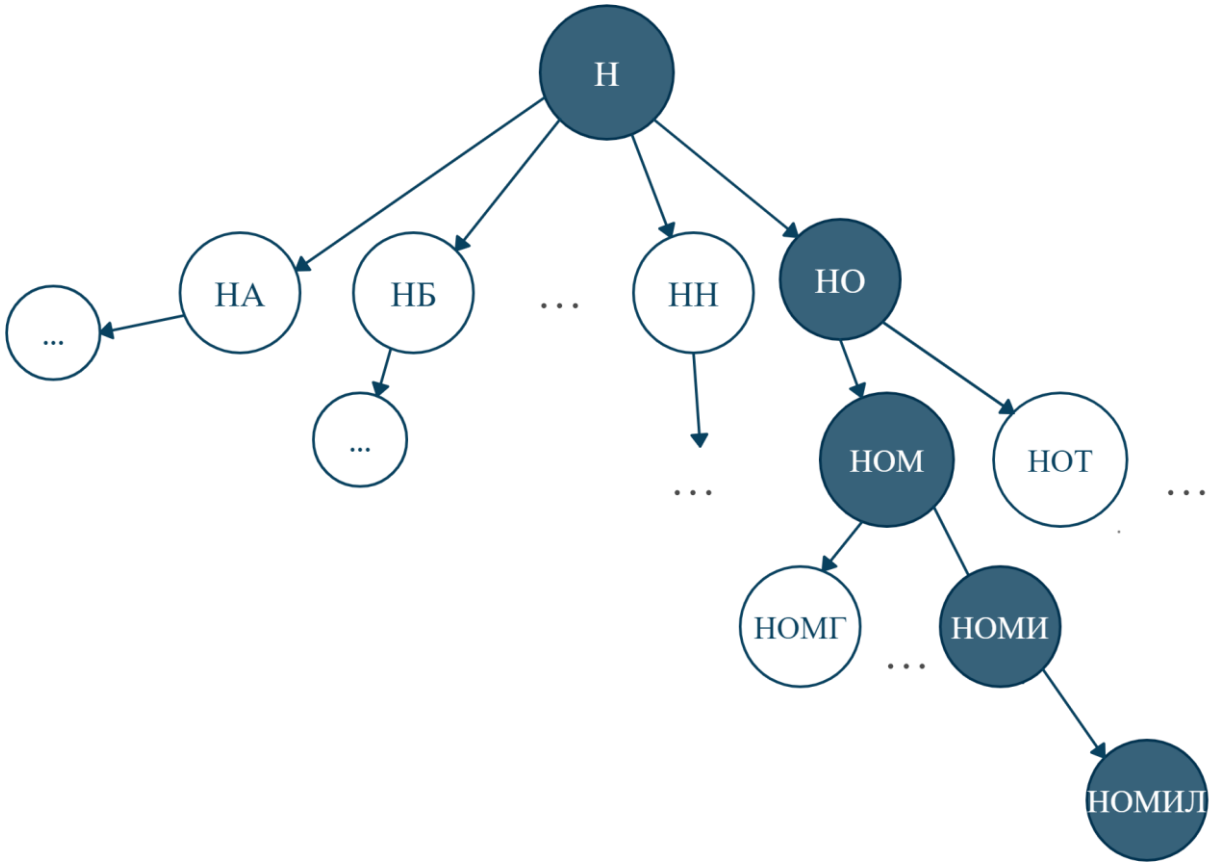
Розробила	Волобуєва В.В.			Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту	
Перевірила	Колесник Л.В.				
Н. Контр.	Колесник Л.В.				
				ІТПм-19-1	Аркуш 1
Затвердив	Гребеннік І.В.			СТ	Аркушів 1

ПРИКЛАД ПОБУДОВИ ДЕРЕВА СУФІКСІВ



Розробила	Волобуєва В.В.			Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту	
Перевірила	Колесник Л.В.				
Н. Контр.	Колесник Л.В.				
				ІТПм-19-1	Аркуш 1
Затвердив	Гребеннік І.В.			СТ	Аркушів 1

ПРИКЛАД ПОБУДОВИ ПРЕФІКСНОГО ДЕРЕВА ІНВЕРТОВАНИХ ОСНОВ



Розробила	Волобуєва В.В.			Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту	
Перевірила	Колесник Л.В.				
Н. Контр.	Колесник Л.В.				
				ІТПм-19-1	Аркуш 1
Затвердив	Гребеннік І.В.			СТ	Аркушів 1

ДОДАТОК В

Сертифікат учасника конференції

СЕРТИФІКАТ УЧАСТІ В II МІЖНАРОДНІЙ НАУКОВО-ПРАКТИЧНІЙ КОНФЕРЕНЦІЇ «PRIORITY DIRECTIONS OF SCIENCE AND TECHNOLOGY DEVELOPMENT»

ДОДАТОК Г

Відомість атестаційної роботи

ГЮИК. 501310.002 ДЗ

(позначення документу)

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1.	ГЮИК.501310.002 ПЗ	Пояснювальна записка	95 с.
2.	ГЮИК.501310.020 - 01 12 01	Додаток А. Текст програми	14 с.
3.		Додаток В. Сертифікат учасника конференції	1 аркуш
		Графічні документи	
4.	ГЮИК.501310.002 С10	Алгоритм роботи стеммінга Портера	1 аркуш
5.	ГЮИК.501310.002 С10	Алгоритм роботи методу Mystem	1 аркуш
6.		Результати вимірювання швидкості знаходження стемми для 1 слова	1 аркуш
7.		Результати вимірювання швидкості знаходження стемми для 100 слів	1 аркуш
8.		Результати вимірювання швидкості знаходження стемми для 1000 слів	1 аркуш
9.		Приклад побудови дерева суфіксів	1 аркуш
10.		Приклад побудови префіксного дерева інвертованих основ	1 аркуш

					ГЮИК. 501310.002 Д3		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Розробила		*Волобуєва В.В.*			*Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту*	Аркуш	Аркушів
Перевірив.		*Колесник Л.В.*				*1*	*1*
Н. контроль		*Колесник Л.В.*				*ХНУРЕ Кафедра СТ*	
Затвердив		*Гребеннік І.В.*					