REPUBLIKA E SHQIPËRISË
UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I TEKNOLOGJISË SË INFORMACIONIT

LABORATOR 5

TEMA : *WordNet*

LËNDA : *Algoritmikë dhe Programim i Avancuar*

DEGA : *Inxhinieri Informatike*

GRUPI : *III B*

PUNOI :
Eni Marsela

PRANOI :
Msc.Paola Shasivari

Në këtë punë laboratori na kërkohet të realizojmë përpunimin e një sistemi fjalësh të organizuar sipas strukturës WordNet.

WordNet është një bazë të dhënash leksikore ku fjalët organizohen në grupe sinonimesh të quajtura **synsets** dhe lidhen mes tyre përmes marrëdhënieve semantike të tipit **hypernym**.

Laboratori kërkon që nga tre skedarë hyrës : **synsets.txt , hypernyms.txt ,diagraph25.txt** të ndërtohet një graf i drejtuar që përfaqëson marrëdhëniet mes fjalëve.

Mbi këtë graf duhet të realizohen disa funksionalitete kryesore:

- kontrolli nëse një fjalë ekziston në WordNet
- llogaritja e distancës semantike midis dy fjalëve
- gjetja e paraardhësit më të afërt të përbashkët
- identifikimi i fjalës që nuk përputhet me një grup (Outcast)

Për zgjidhjen e kësaj detyre u përdor biblioteka algs4, e cila ofron struktura për ndërtimin e grafeve dhe algoritme kërkimi mbi to.

Problemi u nda në disa klasa kryesore, ku secila ka një rol të veçantë dhe bashkëpunon me të tjerat për të realizuar funksionalitetin e plotë të WordNet.

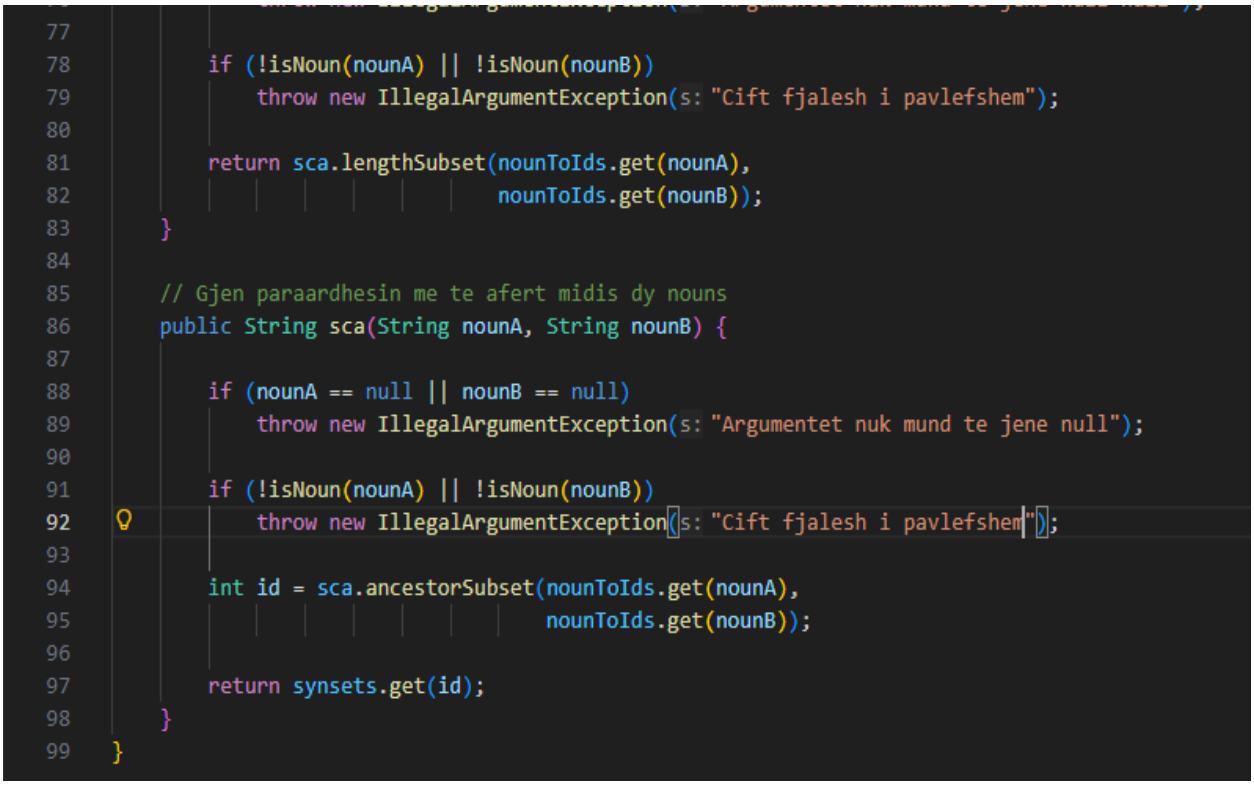
Klasa WordNet

Welcome | ShortestCommonAncestor.java | WordNet.java | OutCast.java | TestWordNet.java

WordNet.java > WordNet > sca(String, String)

```
1   import edu.princeton.cs.algs4.Digraph;
2   import edu.princeton.cs.algs4.In;
3
4   import java.util.*;
5
6   public class WordNet {
7
8       private final Map<String, Set<Integer>> nounToIds = new HashMap<>();
9       private final List<String> synsets = new ArrayList<>();
10      private final ShortestCommonAncestor sca;
11
12      public WordNet(String synsetsFile, String hypernymsFile) {
13
14          if (synsetsFile == null || hypernymsFile == null)
15              throw new IllegalArgumentException(s: "Emrat e file-ve nuk mund te jene null");
16
17          // Leximi i synsets.txt
18          In in = new In(synsetsFile);
19
20          while (in.hasNextLine()) {
21              String line = in.readLine();
22              if (line.isEmpty()) continue;
```

```
23
24              String[] parts = line.split(regex: ",");
25
26              int id = Integer.parseInt(parts[0]);
27              String synsetWords = parts[1];
28
29              synsets.add(synsetWords);
30
31              for (String noun : synsetWords.split(regex: " ")) {
32                  nounToIds.computeIfAbsent(noun, k -> new HashSet<>()).add(id);
33              }
34          }
35
36          // Krijimi i grafit nga hypernyms.txt
37          Digraph G = new Digraph(synsets.size());
38
39          In hyp = new In(hypernymsFile);
40
41          while (hyp.hasNextLine()) {
42              String line = hyp.readLine();
43              if (line.isEmpty()) continue;
44
45              String[] parts = line.split(regex: ",");
46
47              int v = Integer.parseInt(parts[0]);
48
49              for (int i = 1; i < parts.length; i++) {
50                  int w = Integer.parseInt(parts[i]);
51                  G.addEdge(v, w);
52              }
53          }
54
55          // Inicializimi i ShortestCommonAncestor
56          sca = new ShortestCommonAncestor(G);
57      }
58
59      // Kthen te gjitha nouns ne WordNet
60      public Iterable<String> nouns() {
61          return nounToIds.keySet();
62      }
63
64      // Kontrollon nese nje fjale eshte noun ne WordNet
65      public boolean isNoun(String word) {
66          if (word == null)
67              throw new IllegalArgumentException(s: "Argumenti nuk mund te jete null");
68
69          return nounToIds.containsKey(word);
70      }
71
72      // Llogarit distancen midis dy nouns
73      public int distance(String nounA, String nounB) {
74
75          if (nounA == null || nounB == null)
76              throw new IllegalArgumentException(s: "Argumentet nuk mund te jene null null");
```



```
77
78          if (!isNoun(nounA) || !isNoun(nounB))
79              throw new IllegalArgumentException(s: "Cift fjalesh i pavlefshem");
80
81          return sca.lengthSubset(nounToIds.get(nounA),
82                                  nounToIds.get(nounB));
83      }
84
85      // Gjen paraardhesin me te afert midis dy nouns
86      public String sca(String nounA, String nounB) {
87
88          if (nounA == null || nounB == null)
89              throw new IllegalArgumentException(s: "Argumentet nuk mund te jene null");
90
91          if (!isNoun(nounA) || !isNoun(nounB))
92              throw new IllegalArgumentException(s: "Cift fjalesh i pavlefshem");
93
94          int id = sca.ancestorSubset(nounToIds.get(nounA),
95                                      nounToIds.get(nounB));
96
97          return synsets.get(id);
98      }
99  }
```

Klasa WordNet është klasa kryesore e projektit.

Detyrat kryesore të saj janë:

- Leximi i skedarëve synsets.txt dhe hypernyms.txt
- Ndërtimi i strukturave të të dhënave për ruajtjen e fjalëve
- Ndërtimi i grafit të drejtuar
- Ofrimi i metodave kryesore për përdoruesin

Metodat kryesore të kësaj klase janë:

- isNoun(String fjale) – kontrollon nëse një fjalë është pjesë e WordNet
- distance(String a, String b) – llogarit distancën semantike midis dy fjalëve
- sca(String a, String b) – gjen paraardhësin më të afërt të përbashkët

Kjo klasë përdor një strukturë Map për të lidhur çdo fjalë me id-të e synseteve ku ajo shfaqet dhe një listë për të ruajtur përmbajtjen e synseteve.

Klasa ShortestCommonAncestor

Welcome | ShortestCommonAncestor.java | WordNet.java | OutCast.java | TestWordNet.java

ShortestCommonAncestor.java > ShortestCommonAncestor

```java
import edu.princeton.cs.algs4.Digraph;
import edu.princeton.cs.algs4.BreadthFirstDirectedPaths;

public class ShortestCommonAncestor {

    private final Digraph G;

    public ShortestCommonAncestor(Digraph G) {
        if (G == null) throw new IllegalArgumentException();
        this.G = new Digraph(G);
        if (!isRootedDAG()) throw new IllegalArgumentException();
    }

    private boolean isRootedDAG() {
        int roots = 0;
        for (int v = 0; v < G.V(); v++) {
            if (!G.adj(v).iterator().hasNext()) roots++;
        }
        return roots == 1;
    }

    private void validateVertex(int v) {
        if (v < 0 || v >= G.V()) throw new IllegalArgumentException();
    }

    private void validateIterable(Iterable<Integer> it) {
        if (it == null) throw new IllegalArgumentException();
        boolean empty = true;
        for (Integer v : it) {
            if (v == null) throw new IllegalArgumentException();
            validateVertex(v);
            empty = false;
        }
        if (empty) throw new IllegalArgumentException();
    }

    public int length(int v, int w) {
        validateVertex(v);
        validateVertex(w);
        return lengthSubset(java.util.List.of(v), java.util.List.of(w));
    }

    public int ancestor(int v, int w) {
        validateVertex(v);
        validateVertex(w);
        return ancestorSubset(java.util.List.of(v), java.util.List.of(w));
    }

    public int lengthSubset(Iterable<Integer> A, Iterable<Integer> B) {
        validateIterable(A);
        validateIterable(B);

        BreadthFirstDirectedPaths bfsA = new BreadthFirstDirectedPaths(G, A);
        BreadthFirstDirectedPaths bfsB = new BreadthFirstDirectedPaths(G, B);

        int min = Integer.MAX_VALUE;

        for (int v = 0; v < G.V(); v++) {
            if (bfsA.hasPathTo(v) && bfsB.hasPathTo(v)) {
                min = Math.min(min, bfsA.distTo(v) + bfsB.distTo(v));
            }
```

```
        }
        return min == Integer.MAX_VALUE ? -1 : min;
    }

    public int ancestorSubset(Iterable<Integer> A, Iterable<Integer> B) {
        validateIterable(A);
        validateIterable(B);

        BreadthFirstDirectedPaths bfsA = new BreadthFirstDirectedPaths(G, A);
        BreadthFirstDirectedPaths bfsB = new BreadthFirstDirectedPaths(G, B);

        int min = Integer.MAX_VALUE;
        int ancestor = -1;

        for (int v = 0; v < G.V(); v++) {
            if (bfsA.hasPathTo(v) && bfsB.hasPathTo(v)) {
                int dist = bfsA.distTo(v) + bfsB.distTo(v);
                if (dist < min) {
                    min = dist;
                    ancestor = v;
                }
            }
        }
        return ancestor;
    }
}
```

Klasa **ShortestCommonAncestor** merret me përpunimin e grafit të krijuar nga WordNet.

Ajo realizon:

- gjetjen e paraardhësit më të afërt mes dy nyjeve në graf
- llogaritjen e gjatësisë së rrugës më të shkurtër

Për këto llogaritje përdoret algoritmi Breadth First Search (BFS) mbi graf. Klasa pranon si hyrje një Digraph dhe ofron metoda që punojnë mbi indekse nyjesh ose mbi grupe nyjesh.

Klasa OutCast

Klasa Outcast përdor funksionalitetet e WordNet për të analizuar një grup fjalësh dhe për të gjetur atë fjalë që ka lidhjen më të dobët semantike me të tjerat.

Ajo funksionon duke:

- llogaritur distancën e secilës fjalë me të gjitha të tjerat
- duke gjetur fjalën me distancë totale më të madhe

Kjo klasë shërben për të demonstruar në mënyrë praktike përdorimin e metodave distance() dhe sca() të klasës WordNet.

```java
public class OutCast {

    private final WordNet wordnet;

    public OutCast(WordNet wordnet) {
        this.wordnet = wordnet;
    }

    public String outcast(String[] nouns) {
        int max = -1;
        String result = null;

        for (String a : nouns) {
            int sum = 0;
            for (String b : nouns) {
                sum += wordnet.distance(a, b);
            }
            if (sum > max) {
                max = sum;
                result = a;
            }
        }
        return result;
    }
}
```

Klasa TestWordNet

```java
import edu.princeton.cs.algs4.Digraph;

public class TestWordNet {

    Run | Debug
    public static void main(String[] args) {

        System.out.println(x: "\nTESTIMI I WORDNET\n");

        // Krijojme objektin WordNet
        WordNet wn = new WordNet(synsetsFile: "synsets.txt", hypernymsFile: "hypernyms.txt");

        // Test i metodes isNoun
        System.out.println(x: " Testimi i metodes isNoun() ");
        System.out.println("A eshte 'dog' noun?  -> " + wn.isNoun(word: "dog"));
        System.out.println("A eshte 'cat' noun?  -> " + wn.isNoun(word: "cat"));
        System.out.println("A eshte 'asdf' noun? -> " + wn.isNoun(word: "asdf"));

        // Test i distance dhe sca
        System.out.println(x: "\nTestimi i metodave distance() dhe sca()");

        String word1 = "dog";
        String word2 = "cat";

        System.out.println("Fjalet: " + word1 + " dhe " + word2);
        System.out.println("Distanca: " + wn.distance(word1, word2));
        System.out.println("Paraardhesi me i afert: " + wn.sca(word1, word2));

        // Test i ShortestCommonAncestor me graf te vogel
        System.out.println(x: "\nTestimi i klases ShortestCommonAncestor");
```

```
        Digraph G = new Digraph(V: 6);
        G.addEdge(v: 0, w: 1);
        G.addEdge(v: 0, w: 2);
        G.addEdge(v: 1, w: 3);
        G.addEdge(v: 2, w: 3);
        G.addEdge(v: 3, w: 4);
        G.addEdge(v: 4, w: 5);

        ShortestCommonAncestor sca = new ShortestCommonAncestor(G);

        System.out.println("ancestor(1,2) = " + sca.ancestor(v: 1, w: 2));
        System.out.println("length(1,2) = " + sca.length(v: 1, w: 2));

        // Test i klases Outcast
        System.out.println(x: "\nTestimi i klases Outcast");

        OutCast OutCast = new OutCast(wn);

        String[] lista = {"horse", "zebra", "cat", "bear", "table"};

        System.out.println("Fjala qe nuk pershtatet: " + OutCast.outcast(lista));

        // TEST NEGATIV - me fjale qe nuk ekziston
        System.out.println(x: "\nTest me fjale qe nuk ekziston ne WordNet");

        System.out.println("A eshte 'qwerty' noun? -> " + wn.isNoun(word: "qwerty"));

        try {
            wn.distance(nounA: "dog", nounB: "qwerty");
        } catch (Exception e) {
            System.out.println("Gabim i kapur: " + e.getMessage());
        }

        System.out.println(x: "\nTESTIMI PERFUNDOI\n");
    }
}
```

Për verifikimin e saktësisë së implementimit u krijua një klasë testimi TestWordNet.java.

Kjo klasë kryen disa teste kryesore:

- verifikon nëse metoda isNoun funksionon siç duhet
- teston llogaritjen e distancës midis dy fjalëve
- teston gjetjen e paraardhësit më të afërt
- teston funksionimin e klasës Outcast
- provon trajtimin e gabimeve kur jepet një fjalë që nuk ekziston

Outputi

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

\User\Desktop\LABORATOR 5>  & 'C:\Program Files\Java\jdk-25\bin\java.exe' '@C:\Users\Use
t'

 Testimi i metodes isNoun()
A eshte 'dog' noun?  -> true
A eshte 'cat' noun?  -> true
A eshte 'asdf' noun? -> false

Testimi i metodave distance() dhe sca()
Fjalet: dog dhe cat
Distanca: 4
Paraardhesi me i afert: carnivore

Testimi i klases ShortestCommonAncestor
ancestor(1,2) = 3
length(1,2) = 2

Testimi i klases Outcast
Fjala qe nuk pershtatet: table

Test me fjale qe nuk ekziston ne WordNet
A eshte 'qwerty' noun? -> false
Gabim i kapur: Cift fjalesh i pavlefshem

TESTIMI PERFUNDOI

PS C:\Users\User\Desktop\LABORATOR 5>
```

Output-i i testit tregon në mënyrë të qartë:

- cilat fjalë njihen nga WordNet
- sa është distanca midis dy fjalëve
- cili është paraardhësi i tyre i përbashkët
- cila fjalë konsiderohet “outcast” në një listë

Konkluzione

- Struktura WordNet mundëson organizimin e fjalëve në grupe sinonimesh dhe ruajtjen e marrëdhënieve semantike ndërmjet tyre në formën e një grafi të drejtuar.

- Strukturat e të dhënave Map dhe List përdoren për të lidhur fjalët me identifikuesit e synseteve dhe për të ruajtur përmbajtjen e tyre në mënyrë efikase.

- Grafi i drejtuar (Digraph) përfaqëson marrëdhëniet hierarkike hypernym midis synseteve dhe shërben si bazë për të gjitha llogaritjet semantike.

- Klasa ShortestCommonAncestor realizon gjetjen e paraardhësit më të afërt të përbashkët dhe llogarit distancën më të shkurtër midis nyjeve duke përdorur algoritmin BFS.

- Metoda isNoun verifikon nëse një fjalë ekziston në WordNet dhe parandalon përdorimin e fjalëve të pavlefshme.

- Metoda distance llogarit distancën semantike midis dy fjalëve bazuar në rrugën më të shkurtër në graf.

- Metoda sca kthen synsetin që përfaqëson paraardhësin më të afërt të përbashkët të dy fjalëve të dhëna.

- Klasa Outcast analizon një grup fjalësh dhe identifikon fjalën që ka lidhjen më të dobët semantike me pjesën tjetër të grupit.

- Testet e realizuara konfirmojnë që të gjitha funksionalitetet punojnë saktë dhe japin rezultatet e pritshme.

Në përfundim, struktura e ndërtuar përmbush të gjitha kërkesat e laboratorit dhe demonstron përdorimin korrekt të grafeve dhe algoritmeve të kërkimit në përpunimin semantik të fjalëve.