GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

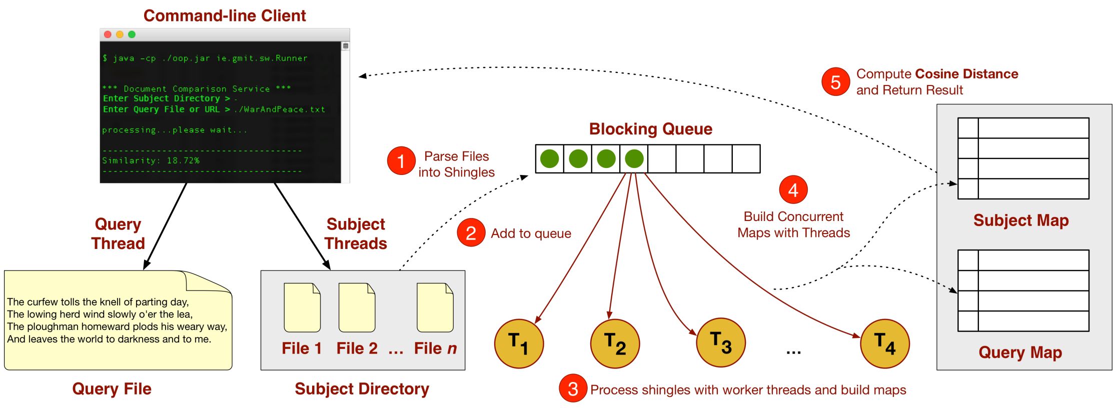
B.Sc. in Software Development

Object Oriented Programming Main Assignment (50%)

A Multithreaded Cosine Distance Computer

1. Overview

You are required to develop a Java API that can rapidly compare a query file against a collection of subject files by computing the ***cosine distance*** between them. The API should uphold the principles of loose-coupling and high cohesion throughout its design by correctly applying abstraction, encapsulation, composition and inheritance.


Command-line Client
Query Thread
Subject Threads
1 Parse Files into Shingles
2 Add to queue
Blocking Queue
3 Process shingles with worker threads and build maps
4 Build Concurrent Maps with Threads
5 Compute Cosine Distance and Return Result
Subject Map
Query Map
File 1
File 2
…
File n
Query File
Subject Directory
T1
T2
T3
…
T4

```
$ java -cp ./oop.jar ie.gmit.sw.Runner

*** Document Comparison Service ***
Enter Subject Directory > .
Enter Query File or URL > ./WarAndPeace.txt

processing...please wait...

----------------------------------------
Similarity: 18.72%
----------------------------------------
```

The curfew tolls the knell of parting day,
The lowing herd wind slowly o'er the lea,
The ploughman homeward plods his weary way,
And leaves the world to darkness and to me.

2. Minimum Requirements

- Use the package name ***ie.gmit.sw***. The application must be deployed and runnable using the specification in Section 3.
- Create a console-based ***menu-driven UI*** to input the path of a directory containing a suite of files to index and a query file to compare. Any other input parameters should be input through the menu-driven UI.
- Each file should be parsed, processed and added to a rapidly searchable data structure.
- The application should be fully threaded and use *Runnable*, *Callable* and *Future* types where appropriate. You should give careful consideration to using the suite of data structures in the ***java.util.concurrent*** package.
- The output of the system should be the percentage similarity of the input query file against the each of the subject files.
- Provide a **UML** diagram of your design and **JavaDoc** your code.

WARNING: *Do not use language features and APIs included after the Java 9 release.*

3. Deployment and Delivery

The project must be submitted by midnight on Sunday 6th January 2019. Before submitting the assignment, you should review and test it from a command prompt on ***a different computer*** to the one that you used to program the project.

- The project must be submitted as a Zip archive ***(not a 7z, rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*A Multithreaded Cosine Distance Computer (50%) Assignment Upload*" heading of Moodle. *Do not add comments to the Moodle assignment upload form.*
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Marks	Category
oop.jar	A Java **archive** containing your API and runner class with a main() method. You can create the JAR file using Ant or with the following command from inside the "bin" folder of the Eclipse project: **jar –cf oop.jar *** The application should be executable from a command line as follows: **java –cp ./oop.jar ie.gmit.sw.Runner**
src	A directory that contains the packaged **source code** for your application.
README.txt	A text file detailing the main **features** of your application. Marks will only be given for features that are described in the README.
design.png	A UML **class diagram** of your API design. The UML diagram should only show the relationships between the key classes in your design. Do not show private methods or attributes in your class diagram. You can create high quality UML diagrams online at **www.draw.io**.
docs	A directory containing the JavaDocs for your application. You can generate JavaDocs using Ant or with the following command from inside the "src" folder of the Eclipse project: **javadoc -d [*path to javadoc destination directory*] ie.gmit.sw** Make sure that you read the JavaDoc tutorial provided on Moodle and comment your source code correctly using the JavaDoc standard.

4. Marking Scheme

Marks for the project will be applied using the following criteria:

Element	Marks	Description
Structure	8	The packaging and deployment correct. All JAR, module, package and runner-class names are correct.
README	8	All features and their design rationale are fully documented. The README should clearly document where and why any design patterns have been used.
UML	8	Class diagram correctly shows all the important structures and relationships between types.
JavaDocs	8	All classes are fully commented using the **JavaDoc** standard and generated docs available in the *docs* directory.
Robustness	38	The fully threaded application executes perfectly, without any manual intervention, using the specified execution requirements.
Cohesion	10	There is very high cohesion between packages, types and methods.
Coupling	10	The API design promotes loose coupling at every level.
Extras	10	Only relevant extras that have been fully documented in the README.

You should treat this assignment as a project specification. Each of the elements above will be scored using the following criteria:

0–30%	**Fail:** Not delivering on basic expectations
40-59%	**Mediocre:** Meets basic expectations
60–79%	**Good:** Exceeds expectations.
80-90%	**Excellent:** Demonstrates independent learning.
90-100%	**Exemplary:** Good enough to be used as a teaching aid

5. Cosine Distance

A commonly employed technique for measuring the degree of similarity between two documents is to represent the documents as sets of letters, words or sub sentences. If we decompose a document into its ***set of constituent words***, we can measure the similarity between them using the cosine distance. Cosine distance is the cosine of the angle between two *n*-dimensional vectors in an *n*-dimensional space. This can be calculated by computing the dot product of the two vectors and then dividing this number by product of the magnitudes (root sum of the squares) of the two vectors:

$$distance(\vec{s}, \vec{t}) = \cos\theta = \frac{\vec{s} \cdot \vec{t}}{\| \vec{s} \| \| \vec{t} \|} = \frac{\sum_{i=1}^{n} s_i t_i}{\sqrt{\sum_{i=1}^{n}(s_i)^2} \times \sqrt{\sum_{i=1}^{n}(t_i)^2}}$$

The similarity or distance is a value between 0 and 1, where

- **0 = Cos 90°** ⇒ Orthogonal and maximally dissimilar.
- **1 = Cos 0°** ⇒ Straight and maximally similar.

Consider the following two string arrays:

```
String[] s = {"I", "think", "I", "may", "have", "signed", "my", "political", "death-warrant", "tonight"};
String[] t = {"I", "have", "signed", "my", "actual", "death-warrant"};
```

Their cosine distance can be measured by counting the frequency of occurrence of each word in each array and then computing their dot product and magnitudes:

Word	**S**	**T**	**Dot Product**
I	2	1	2
think	1	0	0
may	1	0	0
have	1	1	1
signed	1	1	1
my	1	1	1
political	1	0	0
actual	0	1	0
death-warrant	1	1	1
tonight	1	0	0
Magnitude	**√12 = 3.46**	**√6 = 2.44**	**6**

Cosine Distance = 6/(3.46 * 2.44) = 0.707

By calling the ***hashCode()*** method of String, both sets of words can be represented more efficiently as follows, as each hash code is a fixed length 32 bit integer that can be manipulated in a number of different ways.

```
int[] t = {73, 110331122, 73, 107877, 3195240, -902467812, 3500, -210452739, 295511236, -1141101955};
int[] t = {73, 3195240, -902467812, 3500, -1422939762, 295511236};
```

Despite its simplicity, using individual words as the basic unit for a document set element has the disadvantage of eliminating the semantics of a sentence when computing a similarity metric. Consider the following two sequences **A** and **B**:

A = {object, oriented, programming, is, good, fun, for, me}
B = {me, object, for, good, programming, is, fun, oriented}

Although both document sets have different meanings, their cosine distance is **cos(A, B)** = 8 / (2.828 × 2.828) = 0.999999, i.e. they are identical… In practice the following two alternatives to individual words are used:

- **Shingles: the basic unit (element) is a fixed-size group of words**. This has the advantage of retaining some of the semantics of the document. Because each individual word may have a variable length, the resultant shingles may also have variable length. We can represent the sets **A** and **B** with 3-word shingles as follows:

  A = {object oriented programming, is good fun, for me}
  B = {me object for, good programming is, fun oriented}

  The cosine distance is now **cos(A, B)** = 0 / (1.732 × 1.732) = 0, i.e. the semantics of each document set has been preserved. This approach is often used when measuring the similarity of documents intended to be read by humans.

- ***k*-Shingles / *k*-mers: the basic unit (element) is a fixed-size block of characters of length *k***. A *k*-Shingle is more often referred to as a *k*-mer or *l*-mer in computer science (the suffix *-mer* means 'part' in Greek). The sets **A** and **B** can be represented as **5**-mers by creating a tiling of characters of size *k* (spaces are shown below with an underscore for clarity).

  A = {objec, t_ori, ented, _prog, rammi, ng_is, _good, _fun_, for_m, e____}
  B = {me_ob, ject_, for_g, ood_p, rogra, mming, _is_f, un_or, iente, d____}

  The cosine distance is **cos(A, B)** = 0 / (3.1622 × 3.1622 = 0. Note that a single character insertion of deletion (a small change) will have large impact on the resultant set. This approach is heavily utilised in bioinformatics for comparing DNA and protein sequences.

6. Useful Links

- **Web UML Editor:** https://www.draw.io
- **Fuzzy String Matching Using Cosine Similarity:** https://blog.nishtahir.com/2015/09/19/fuzzy-string-matching-using-cosine-similarity/