Programming Assignment 4

Mohamad Mahdi Ziaee

109561770

Spring 2016

Contents

[Description 1](#_Toc445802117)

[Application specifications 1](#_Toc445802118)

[com.bio.main.NRDatabaseApp 2](#_Toc445802119)

[com.bio.main.pojo.BlastNRecord 3](#_Toc445802120)

[com.bio.main.pojo.Query 4](#_Toc445802121)

[com.bio.main.pojo.Database 5](#_Toc445802122)

[com.bio.main.util.DatabaseUtil 6](#_Toc445802123)

[com.bio.main.util.FileUtil 8](#_Toc445802124)

# Description

The main purpose of this application is to remove duplicate records from \*.fa files. All the files must be in Assignment4/io folder. Please refer to com.bio.main.NRDatabaseApp class to see the list of constants and file names. Here is the set of steps needed for this assignment:

1. Generate BlastN output for Query: MetaHIT20000 DB: HMP2000. The result is called Output-Q-MetaHIT-DB-HMP and it must be placed in io folder.
2. Un comment the top 3 lines in com.bio.main.NRDatabaseApp and comment the last 3 lines just like below:

Database db = FileUtil.getInstance().readBlastNRecords(Q\_METAHIT\_DB\_HMP\_OUTPUT);

DatabaseUtil.getInstance().findDuplicateRecords(db);

FileUtil.getInstance().copyFileExcludeRedundantQueries(META\_HIT\_NR\_HMP\_FA, METAHIT\_20000\_FA, db.getDuplicateQueries());

// Database db = FileUtil.getInstance().readBlastNRecords(Q\_HMP\_DB\_META\_HIT\_NR\_HMP\_OUTPUT);

// DatabaseUtil.getInstance().findDuplicateRecords(db);

// FileUtil.getInstance().copyFileExcludeRedundantQueries(HMP\_NR\_META\_HIT\_NR\_HMP\_FA, HMP\_2000\_FA, db.getDuplicateQueries());

1. Run the program and the result of the output will be stored in a file called MetaHIT-nr-HMP.fa which is a copy of MetaHIT-20000.fa but without the duplicate records.
2. Generate BlastN output for Query: HMP2000 DB: MetaHIT-nr-HMP. The result is called Output-Q-HMP-DB-MetaHIT-nr-HMP and it must be placed in io folder.
3. Comment out the top 3 lines in com.bio.main.NRDatabaseApp and enable the last 3 lines.
4. Run the program and the result of the output will be stored in a file called HMP-nr-(MetaHIT-nr-HMP).fa which is a copy of HMP-2000.fa but without the duplicate records.

## Application specifications

**Programming language:** JAVA

**Other required installations:** JRE 1.8 to run the application,

JDK 1.8 to compile the application.

**Version Control:** Git (GitHub) - https://github.com/momazia/Bioinformatic/tree/master/Assignment4

**IDE:** Eclipse (Mars 4.5.1)

**Build automation tool:** Maven (Refer to /Assignment4/pom.xml for the list of the libraries used)

# com.bio.main.NRDatabaseApp

package com.bio.main;

import java.io.IOException;

import com.bio.main.pojo.Database;

import com.bio.main.util.FileUtil;

import com.bio.main.util.DatabaseUtil;

/\*\*

\* The main class to be executed for 4th assignment. The main application will read the BlastN records generated and finds the duplicate records.

\* Later it will remove those records from the query file. All the files are written and read from /Assignment4/io folder.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class NRDatabaseApp {

/\*\*

\* List of all the file names

\*/

public static final String HMP\_2000\_FA = "HMP-2000.fa";

public static final String HMP\_NR\_META\_HIT\_NR\_HMP\_FA = "HMP-nr-(MetaHIT-nr-HMP).fa";

public static final String Q\_HMP\_DB\_META\_HIT\_NR\_HMP\_OUTPUT = "Output-Q-HMP-DB-MetaHIT-nr-HMP";

public static final String META\_HIT\_NR\_HMP\_FA = "MetaHIT-nr-HMP.fa";

public static final String Q\_METAHIT\_DB\_HMP\_OUTPUT = "Output-Q-MetaHIT-DB-HMP";

public static final String METAHIT\_20000\_FA = "MetaHIT-20000.fa";

public static void main(String[] args) {

try {

// Depending on which files are being processed, uncomment/comment the 3 lines below.

Database db = FileUtil.getInstance().readBlastNRecords(Q\_METAHIT\_DB\_HMP\_OUTPUT);

DatabaseUtil.getInstance().findDuplicateRecords(db);

FileUtil.getInstance().copyFileExcludeRedundantQueries(META\_HIT\_NR\_HMP\_FA, METAHIT\_20000\_FA, db.getDuplicateQueries());

// Database db = FileUtil.getInstance().readBlastNRecords(Q\_HMP\_DB\_META\_HIT\_NR\_HMP\_OUTPUT);

// DatabaseUtil.getInstance().findDuplicateRecords(db);

// FileUtil.getInstance().copyFileExcludeRedundantQueries(HMP\_NR\_META\_HIT\_NR\_HMP\_FA, HMP\_2000\_FA, db.getDuplicateQueries());

} catch (IOException e) {

e.printStackTrace();

}

}

}

# com.bio.main.pojo.BlastNRecord

package com.bio.main.pojo;

import java.util.List;

/\*\*

\* POJO to hold the values read from the BlastN file output

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class BlastNRecord {

/\*\*

\* Holds the whole raw record string

\*/

private StringBuffer str = new StringBuffer();

/\*\*

\* length of the query

\*/

private Integer length;

/\*\*

\* List of all the alignment lengths given for each query

\*/

private List<Integer> alignmentLengths;

/\*\*

\* Query string

\*/

private String queryString;

public Integer getLength() {

return length;

}

public void setLength(Integer length) {

this.length = length;

}

public StringBuffer getStr() {

return str;

}

public void setStr(StringBuffer str) {

this.str = str;

}

public List<Integer> getAlignmentLengths() {

return alignmentLengths;

}

public void setAlignmentLengths(List<Integer> alignmentLengths) {

this.alignmentLengths = alignmentLengths;

}

public String getQueryString() {

return queryString;

}

public void setQueryString(String queryString) {

this.queryString = queryString;

}

}

# com.bio.main.pojo.Query

package com.bio.main.pojo;

/\*\*

\* POJO to hold information about the queries read from the files.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class Query {

private String name;

private String str;

public Query(String name, String str) {

super();

this.setName(name);

this.setStr(str);

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getStr() {

return str;

}

public void setStr(String str) {

this.str = str;

}

}

# com.bio.main.pojo.Database

package com.bio.main.pojo;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

/\*\*

\* The main DB class which holds BlastN records read from the output and a unique set of the duplicate queries.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class Database {

private List<BlastNRecord> blastNRecords;

private Set<String> duplicateQueries;

public boolean addRedundantQuery(String arg0) {

if (duplicateQueries == null) {

duplicateQueries = new HashSet<>();

}

return duplicateQueries.add(arg0);

}

public Database(List<BlastNRecord> blastNRecords) {

this.setBlastNRecords(blastNRecords);

}

public Set<String> getDuplicateQueries() {

return duplicateQueries;

}

public void setDuplicateQueries(Set<String> duplicateQueries) {

this.duplicateQueries = duplicateQueries;

}

public List<BlastNRecord> getBlastNRecords() {

return blastNRecords;

}

public void setBlastNRecords(List<BlastNRecord> blastNRecords) {

this.blastNRecords = blastNRecords;

}

}

# com.bio.main.util.DatabaseUtil

package com.bio.main.util;

import java.util.ArrayList;

import java.util.List;

import org.apache.commons.lang3.StringUtils;

import com.bio.main.pojo.Database;

import com.bio.main.pojo.BlastNRecord;

/\*\*

\* Database utility class (Singleton) which is responsible for processing the output of BlastN records.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class DatabaseUtil {

/\*\*

\* List of constants

\*/

private static final int LOWER\_BOUND\_PERCENTAGE = 90;

private static final int UPPER\_BOUND\_PERCENTAGE = 110;

private static DatabaseUtil instance = null;

private DatabaseUtil() {

super();

}

public static DatabaseUtil getInstance() {

if (instance == null) {

instance = new DatabaseUtil();

}

return instance;

}

/\*\*

\* Populates the length property of {@link BlastNRecord} by finding it in the record's str.

\*

\* @param record

\*/

public void findQueryLength(BlastNRecord record) {

String lengthStr = StringUtils.substringBetween(record.getStr().toString(), "Length=", System.getProperty("line.separator"));

record.setLength(Integer.valueOf(lengthStr));

}

/\*\*

\* Populates the alignment lengths for the given record in its str.

\*

\* @param record

\*/

public void findAlignmentLengths(BlastNRecord record) {

String[] alignmentLengths = StringUtils.substringsBetween(record.getStr().toString(), " Identities = ", " (");

if (alignmentLengths != null) {

List<Integer> alignments = new ArrayList<>();

for (String alignmentLength : alignmentLengths) {

alignments.add(Integer.valueOf(StringUtils.substringAfter(alignmentLength, "/")));

}

record.setAlignmentLengths(alignments);

}

}

/\*\*

\* Checks to see if the record given is duplicate by running 90% <= alignment\_length/query\_length <= 110% validation against all the alignment

\* records and as long as one of them fits into the logic, it is considered duplicate. If there is no alignment length, it will return false.

\*

\* @param record

\* @return

\*/

public boolean isRedundant(BlastNRecord record) {

if (record.getAlignmentLengths() != null) {

for (Integer alginmentLength : record.getAlignmentLengths()) {

Double percentage = alginmentLength.doubleValue() / record.getLength().doubleValue() \* 100;

if (LOWER\_BOUND\_PERCENTAGE <= percentage && percentage <= UPPER\_BOUND\_PERCENTAGE) {

return true;

}

}

}

return false;

}

/\*\*

\* Runs the main logic in which it finds the duplicate queries for the given BlastN Records.

\*

\* @param db

\*/

public void findDuplicateRecords(Database db) {

for (BlastNRecord record : db.getBlastNRecords()) {

findQueryLength(record);

findAlignmentLengths(record);

if (isRedundant(record)) {

System.out.println("Redundant query: [" + record.getQueryString() + "]");

db.addRedundantQuery(record.getQueryString());

}

}

System.out.println("Number of Duplicate Records found: [" + db.getDuplicateQueries().size() + "]");

}

}

# com.bio.main.util.FileUtil

package com.bio.main.util;

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.List;

import java.util.Set;

import org.apache.commons.io.FileUtils;

import org.apache.commons.lang3.StringUtils;

import org.apache.commons.lang3.mutable.MutableInt;

import com.bio.main.pojo.BlastNRecord;

import com.bio.main.pojo.Database;

import com.bio.main.pojo.Query;

/\*\*

\* This is a utility class in charge of the operations related to reading/saving to files.

\*

\* @author Mohamad Mahdi Ziaee

\*

\*/

public class FileUtil {

/\*\*

\* List of constants

\*/

public static final String SEPARATOR = ">";

private static final String QUERY = "Query= ";

public static final String IO\_PATH = "../Assignment4/io/";

private static FileUtil instance = null;

private FileUtil() {

super();

}

public static FileUtil getInstance() {

if (instance == null) {

instance = new FileUtil();

}

return instance;

}

/\*\*

\* Creates a database by reading the BlastN output file name given.

\*

\* @param fileName

\* @return

\* @throws IOException

\*/

public Database readBlastNRecords(String fileName) throws IOException {

System.out.println("Reading file [" + fileName + "]...");

// Reading the file line by line

List<BlastNRecord> records = new ArrayList<>();

List<String> fileLines = Files.readAllLines(Paths.get(IO\_PATH + fileName));

for (int fileIndex = 0; fileIndex < fileLines.size(); fileIndex++) {

String line = fileLines.get(fileIndex);

// If it contains query string

if (isQueryString(line)) {

BlastNRecord record = new BlastNRecord();

MutableInt mutableIndex = new MutableInt(fileIndex);

findQuery(fileLines, mutableIndex, record);

fileIndex = mutableIndex.getValue();

records.add(record);

}

}

return new Database(records);

}

/\*\*

\* Populates the record object passed. It also changes the fileIndex as it reads through the file lines passed.

\*

\* @param fileLines

\* @param fileIndex

\* @param record

\*/

private void findQuery(List<String> fileLines, MutableInt fileIndex, BlastNRecord record) {

// Adding the first line containing the query itself

String line = fileLines.get(fileIndex.getValue());

addLine(record, line, fileLines, fileIndex, true);

fileIndex.increment();

// Finding the rest of the record data from file.

while (!endOfFile(fileLines, fileIndex.getValue())) {

line = fileLines.get(fileIndex.getValue());

if (!isQueryString(line)) {

addLine(record, line, fileLines, fileIndex, false);

fileIndex.increment();

} else {

fileIndex.decrement();

return;

}

}

}

/\*\*

\* Populates str property of the given BlastN record. If it is a query string line we are looking at, it will also populate Query string property

\* of BlastN record.

\*

\* @param record

\* @param line

\* @param fileLines

\* @param fileIndex

\* @param isQueryString

\*/

private void addLine(BlastNRecord record, String line, List<String> fileLines, MutableInt fileIndex, boolean isQueryString) {

if (isQueryString) {

record.setQueryString(StringUtils.substringAfter(line, QUERY));

}

record.getStr().append(line);

record.getStr().append(System.getProperty("line.separator"));

}

/\*\*

\* Checks to see if we are looking at the last line of the list given.

\*

\* @param fileLines

\* @param fileIndex

\* @return

\*/

private boolean endOfFile(List<String> fileLines, int fileIndex) {

return fileIndex >= fileLines.size();

}

/\*\*

\* Checks to see if constant {@link FileUtil#QUERY} exists in the line passed.

\*

\* @param line

\* @return

\*/

private boolean isQueryString(String line) {

return StringUtils.contains(line, QUERY);

}

/\*\*

\* Reads the query list from the source file path given and for each of them, it writes them into the target file path if they are not registered

\* as a duplicate query.

\*

\* @param targetFilePath

\* @param sourceFilePath

\* @param duplicateQueries

\* @throws IOException

\*/

public void copyFileExcludeRedundantQueries(String targetFilePath, String sourceFilePath, Set<String> duplicateQueries) throws IOException {

System.out.println("Copying the unique queries into [" + targetFilePath + "]..");

List<Query> queries = readQueries(sourceFilePath);

for (Query query : queries) {

if (!duplicateQueries.contains(query.getName())) {

// FileUtils.writeStringToFile(new File(IO\_PATH + targetFilePath), query.getStr(), true);

StringBuffer strBuffer = new StringBuffer();

strBuffer.append(SEPARATOR);

strBuffer.append(query.getName());

strBuffer.append(System.getProperty("line.separator"));

FileUtils.writeStringToFile(new File(IO\_PATH + targetFilePath), strBuffer.toString(), true);

}

}

}

/\*\*

\* Creates a list of queries from the source file path given.

\*

\* @param sourceFilePath

\* @return

\* @throws IOException

\*/

public List<Query> readQueries(String sourceFilePath) throws IOException {

List<Query> result = new ArrayList<>();

String fileContent = new String(Files.readAllBytes(Paths.get(IO\_PATH + sourceFilePath)));

// Splitting the file content using ">"

String[] splitStrs = StringUtils.split(fileContent, SEPARATOR);

for (String str : splitStrs) {

String queryName = StringUtils.substringBefore(str, "\n");

StringBuffer strBuffer = new StringBuffer();

strBuffer.append(SEPARATOR);

strBuffer.append(str);

String queryStr = strBuffer.toString();

result.add(new Query(queryName, queryStr));

}

return result;

}

}