Copyright (June 12, 2025) Xunhua Wang

All Rights Reserved

These projects are produced by Dr. Xunhua Wang. If you are taking his course, you can make a *single* machine-readable copy and print a *single* copy of each page for your own reference, so long as the pages contain this copyright statement. Permission for any other use must be obtained from the author (wangxx@jmu.edu) in writing.

The Digital Millennium Copyright Act (DMCA) strictly forbids any unauthorized actions on this document, including content modification, reproduction, and removal of security mechanisms.

Project 5

Regular Expression with Java or another programming language

In the field of computer science, many things in *systems*, *programming languages*, and *software engineering* played important roles for a time but eventually they faded away. According to one anecdote by Richard Hamming (Google *Richard Hamming* if you do not know him), half of the knowledge in these areas becomes obsolete in every 17 years. (It might be shorter these days, as technologies advance at an even faster pace now.) So, when you reach your 40s, half of your systems, programming and software engineering knowledge will become obsolete and you will have to learn, by yourself, to replace this half of your knowledge; otherwise you will get technically obsolete quickly. And most likely your ability to learn new things will degrade over time, trust me. You will struggle with systems and programming languages in the upcoming years. (Eventually, everybody gets obsolete and that is almost inevitable.)

In contrast, a few things in computer science stand the test of time very well. These things may change form over time but their core remains the same. For this part of CS, once you understand their essence, you will be able to recognize them in new applications immediately and will learn the new applications quickly; you will never fall behind with these things.

Computability, *complexity*, *algorithms*, and *regular expression* are in this category. (For algorithms, I really mean those great algorithms that work all the time, on all relevant problem instances. Most existing AI algorithms that work sometime, but fail other times do not count; these fake algorithms will either destroy our physical world or get replaced quickly.) Regular expression, the topic of this project, is in the latter camp.

In computer science, regular expression is actually one of a very few of a kind. On one hand, it is a building block for *formal language* and it establishes the foundation for all modern programming languages (think about Python, Java, C/C++, C#, Ruby, PHP, name a few), making it theoretically significant. On the other hand, regular expression also sees immediate real-world applications for textual searching & replacement in programming languages such as Java, Python, .NET, Ruby, PHP, & C/C++, and tools like grep/egrep, emacs, & vi. (Believe it or not, string matching and replacement is still an important application of

computers; think about those textual web pages, emails, and documents. String matching is also important for information security in detecting malware.)

This theory/practice duality and the no-watering-down from theory to practice of regular expression makes it a really powerful tool. Unfortunately, regular expression is also a subject hard to learn and apply. In this class, you have studied both the theory and practical aspects of regular expression.

5.1 Learning goals of this project

This project aims to

1. Deepen your understanding on regular expression.
2. Provide you firsthand experience of real-world application of regular expression in Java, a programming language that you already know well enough.

5.2 Data file

When Mrs. Hillary Clinton was the US Secretary of State between 2009 and 2013, instead of using her official state department email account, she chose to use a private server for all her email communications. This became a big issue in the 2016 presidential election and might be a reason that Mrs. Clinton lost. In the 2016 presidential election, to clear her name, Mr. Clinton asked the state department to release her 7,000 work-related emails. However, these 7000 emails are just a subset of all her about 33,000 emails in this period; the Clinton people removed, with a security tool kit called *BleachBit*, the remaining emails from their server. According to US representative Trey Gowdy, the deleted emails were deleted "*where even God can't read them.*" Then-candidate Trump called for Russian hackers to recover these deleted emails: *"Russia, if you're listening, I hope you're able to find the 30,000 emails that are missing. I think you will probably be rewarded mightily by our press."*

And WikiLeaks responded and published Clinton's 33,727 emails. The Wikileaks links for Clinton's emails are between `https://wikileaks.org/clinton-emails/emailid/1` and `https://wikileaks.org/clinton-emails/emailid/33727`. (It remains unclear who are the hackers that provided the emails to WikiLeaks and how these hackers got the emails, probably by hacking into Clinton's private email server.)

These emails have been downloaded to my web site at
`https://crypto.cs.jmu.edu/clinton/clinton-33727-emails-by-wiki-03.txt` **In this project, you must use this local file** and develop a regular expression-based program in Java (or your favorite programming language that is not Java) to analyze it. This given file is textual but may occasionally contain unprintable unicode characters. In this file, each email starts with a line like "<<<<<EMAIL sequence-number >>>>>," where *sequence-number* is a monotonically increasing number (between 1 and 33,727) used by WikiLeaks. Most emails in the given file appear in the natural order but a few emails at the end of the file are out of order.

5.3 Email address format

For this project, you will use regular expression to search and find email addresses in a given data file. What constitute a valid email address? For this project, a valid email address must be **case-insensitive** and consist of three parts: *local-part*, @, and *domain name*.

The local-part of an email address may contain the following characters:

1. Lowercase Latin letters a to z

   (Note that in this project, email addresses are case-insensitive and thus uppercase letters are converted to lowercase first.)

2. Digits 0 to 9

3. Special characters

   `!#$%&*+-/=?^_‘{|}~`

   Note that single quote is *not* included above and the character that you might have a question about the above set is apostrophe. On a regular keyboard, you can find apostrophe on the top left corner.

4. Dot (.), with the following restrictions:

   (a) Dot cannot be the first or last character unless quoted

   (b) Dot does not appear consecutively unless quoted. For example, *John..Doe@jmu.edu* is not allowed while *“John..Doe”@jmu.edu* is allowed

5. Space and `"(),:;<>@[\]` characters are allowed with the following restrictions:

   (a) They are allowed only inside a quoted string

   (b) Backslash or double-quote must be preceded by a backslash

   In this project, when a local part is quoted, the whole local part must be in the quotation marks. For example, “wangxx=nerd”@jmu.edu is valid while “wangxx=”nerd@jmu.edu is not.

6. Comments are allowed with parentheses at either end of the local-part. For example, *john.doe(comment)@jmu.edu* and *(comment)john.doe@jmu.edu* are both equivalent to *john.doe@jmu.edu*

A *domain name* consists of a sequence of *labels* connected with dot (.), if dot appears at all.

1. A *label* is case-insensitive and may contain only

   (a) the ASCII letters a through z

   (b) the digits 0 through 9

   (c) the hyphen (-). However, labels cannot start with a hyphen or end with a hyphen. For example, domain name *gmail-.com* is not allowed.

(d) No other symbols, punctuation characters, or blank spaces are permitted

2. Email addresses like wangxx@[134.126.20.79] are valid, where an IP address, not a domain name, in a pair of square bracket, is used.

Email addresses also have length restrictions:

1. The local-part of an email address must not exceed 64 characters
2. The domain part of an email address must not exceed 255 characters
3. The total length of an email address must not exceed 254 characters

(Yes, I know, the last two may look contradictory but they are copied from the actual standards.)

5.4 Example programs with regex

5.4.1 Regex programming in Java

```
1  import java.util.regex.Matcher;
2  import java.util.regex.Pattern;
3
4  public class Re3 {
5     public static void main( String args[] ) {
6        // String to be scanned to find the pattern.
7          String line = "2Cab 4 Level smarter than 3 dogs";
8          String myregex = "([^D-Z]*)([0-9][0-9]*)";
9
10         Pattern r = Pattern.compile(myregex);
11
12         // Now create matcher object.
13         Matcher m = r.matcher(line);
14
15         while (m.find( )) {
16                System.out.println("Found value (whole): " + m.group(0) );
17                System.out.println("Found value (first part): " + m.group(1) );
18                System.out.println("Found value (second part): " + m.group(2) + "\n
                      ");
19         }
20    }
21 }
```

For the above program, you can compile it with *javac Re3.java* and then run it with java Re3

Next, study the output of this Java program and compare it with what you expect.

5.4.2 Regex programming in Python

```
1  #!/usr/bin/python
2  import re
3
4  def regex_example ():
5      line = "2Cab 4 Level smarter than 3 dogs";
6
7      myregex = r'([^D-Z]*)([0-9][0-9]*)'
8
9      #
10     # There are two different ways to do regex search in Python; Just pick one,
           but not both
11     #
12     # 1. The following code demonstrates how to get the matching results in just
           one call
13     #
14     x = re.findall (myregex, line)
15     print ('result1 = ', x, '\n')
16
17     #
18     # 2. The following code demonstrates how to get the matching results
           iteratively
19     #
20     allSearchObjs = re.finditer(myregex, line)
21     # allSearchObjs = re.finditer( myregex, line, re.M|re.I|re.X)
22     #   re.A (ASCII-only matching),
23     #   re.I (ignore case),
24     #   re.L (locale dependent),
25     #   re.M (multi-line),
26     #   re.S (dot matches all),
27     #   re.U (Unicode matching),
28     #   and re.X (verbose),
29     #
30
31     print ('result2 = ')
32     for searchObj in allSearchObjs:
33         # print ("searchObj.group(): ", '|', searchObj.group(), '|')
34         print ("\tsearchObj.group(0): ", searchObj.group(0))
35         print ("\tsearchObj.group(1): ", searchObj.group(1))
36         print ("\tsearchObj.group(2): ", searchObj.group(2))
37         s = searchObj.start()
38         e = searchObj.end()
39         print ('\tString match "%s" at %d:%d' % (line[s:e], s, e))
40         print ('')
41
42 def main():
43     regex_example()
44
```

```
45 if __name__ == "__main__":
46     main()
```

For the above program, you can run it with python re3.py

Next, study the output of this Python program and compare it with what you expect.

5.5 Tasks (30 points)

Use your knowledge about regular expression and develop a Java program (**based on regular expression**) to answer the following questions.

1. (10 points) What is the regular expression that can be used to find all email addresses in the given file?

   Your regular expression must meet the following requirements:

   (a) **(5 points)** For the local-part of an email address, it must at least the first four requirements in Section 5.3

   (b) **(5 points)** For the domain name part, it must meet all the requirements in Section 5.3

   (c) The length requirement in Section 5.3 can be met with additional checking (beyond regular expression)

   (d) **(5 bonus points)** For the local-part of an email address, if your regular expression is correct and also meets the last two requirements in Section 5.3 (i.e., requirements 5 and 6), then bonus points will be awarded.

   Note that your regular expression must be very correct for any bonus points.

   To get any point for this task, your (hard copy, when applicable, and electronic) report must

   (a) have a correct regular expression

   (b) contain a clear, *detailed* explanation how your regular expression works to find all email addresses.

   Your explanation must be specific to your own regular expression. If your explanation is too general, you will get zero points. **No clear detailed, specific explanation, no points for this task**.

   Further notes on this subtask:

   (a) It takes multiple rounds to come up with a decent regular expression. Be patient and try multiple iterations for this subtask.

   (b) This subtask is the basis of the whole project, in the sense that the quality of your regular expression will directly determine your answering numbers for the rest of the project.

As a result, for this subtask, you have the option to get some specific help from the instructor. However, if you choose to do so, you will lose all the points for this subtask.

2. (5 points) How many email addresses are there in those emails in the given file? All email addresses in the given file, including those in either the email headers or the email bodies, must be counted.

   It should also be noted that an email address may appear multiple times and in this task, each appearance is counted once. For example, if *wangxx@jmu.edu* appears twice in the given file, it should be counted twice.

   Your *hard copy* report, if applicable, must include the exact total number of email addresses (reported by your program).

3. (5 points) How many **unique** email addresses are there in those emails in the given file? An email address may appear multiple times in the given project file and *uniqueness* in this task means that it should be counted only once. For example, if the same *wangxx@jmu.edu* appears twenty times in the given project file, it must be counted only once.

   All email addresses in the given file, including those in either the email headers or the email bodies, must be counted.

   (a) Your *hard copy* report, if applicable, must include the exact total number of **unique** email addresses (reported by your program).

   (b) In addition to the total number, your *electronic* submission must include a *separate* text file, with file name *YOURJMUEID_all_emailaddresses.txt*, containing all *unique* email addresses extracted from the file, with one email address per line. Here, YOURJMUEID is your JMU EID; mine is *wangxx* and thus my file name should be *wangxx_all_emailaddresses.txt*.

4. (10 points) For each unique email address, how many times has it appeared in the given file? That is, you need to find out their frequencies. Next, sort the email addresses in terms of their frequencies (in the descending order).

   For this task,

   (a) Your *hard copy* report, if applicable, must include the top 10 email addresses and their frequencies **in the descending order**.

   (b) Your *electronic* submission must include file called *YOURJMUEID_sorted_emailaddresses.txt*, containing **all/complete** sorted email addresses and their frequencies, **in the descending order**, with one email address per line. Here, YOURJMUEID is your JMU EID; mine is *wangxx* and thus my file name for this task should be *wangxx_sorted_emailaddresses.txt*.

   Hint: build a map/dictionary and sort the map/dictionary in terms of value.

5. (5 **bonus** points) **This is a bonus task and you do *not* have to do it.** For each email address found in earlier tasks, you can count the number of emails that it receives from and send to another email account. Based on this data a *directed graph* can be built, in which each node is an email account and on each *directed* edge is the total number of emails sent by the initial vertex to the end vertex. Such a directed graph will show the closeness among email accounts and their owners. (It should be noted that the above graph does not capture all information. For example, the time in seconds between an email and its reply is missing and it might be very important information.)

   In this task, build such a directed graph and draw it. To earn any bonus points, your graph must be crisply clear and informative.

Your program must be developed in a way that when run, it prints out the required answers specified above one by one. (Yes, it needs to print out your regular expression (for Question 1) and the corresponding explanation too.)

5.6 Submission requirements

Your submission for this project must include

1. An electronic submission to Canvas, which must include
   (a) your code.
      i. Your program must be named as *YOURJMUEIDRegEx.py* (in Python) or *YOURJMUEIDRegEx.java* (in Java), where YOURJMUEID is your JMU EID; for example, my program Python should be named as *wangxxRegEx.py* while my program in Java should be named as *wangxxRegEx.java*.
      ii. Your program must take a filename argument from the command line and process the emails in the file.

      When your Java code is tested, it will be run as
      java YOURJMUEIDRegEx clinton-33727-emails-by-wiki-03.txt When your python code is tested, it will be run as
      python YOURJMUEIDRegEx clinton-33727-emails-by-wiki-03.txt
      iii. Your code must run (and print the correct results). **No working code, no points.**
   (b) a screen shot in which your code is run and prints out the answers for your hard copy submission. (Your program also generates the required files but this may be invisible.)

      You must not crop your screen shot.
   (c) File *YOURJMUEID_all_emailaddresses.txt*, containing unique email addresses
   (d) File *YOURJMUEID_sorted_emailaddresses.txt*

2. If applicable, a hard copy submission of your screen shot, with all the aforementioned requirements for a hard copy. **When applicable, no hard copy, no points.**

   Do not try to crop your screen shot. A full screen shot is needed.

5.7 Unique grading policy for this project

1. Your computer program must be based on regular expression. If your program does all the tasks correctly but without regular expression, you will get zero points. I repeat, zero points for any non-regular expression approach.
2. If your electronic submission does not have file *YOURJMUEID_all_emailaddresses.txt*, you will get zero points for that task.
3. If your electronic submission does not have file *YOURJMUEID_sorted_emailaddresses.txt*, you will get zero points for that task.

5.8 Hints

1. How to check the correctness of your regular expression and computer program?

   (a) From the given large data file, you can copy-and-paste the first 20 emails (marked by <<<<<EMAIL number >>>>>) to another file, say test.txt.

   (b) Next, manually count the number of email addresses in test.txt.

   (c) Run your program against test.txt and compare your manual count against the computer count.

   In this way, you can test the correctness of your program.

2. When unexplainable errors happen, how to debug your computer program?

   If you are using Eclipse for Java (or Python), you can set a breakpoint by *right* clicking on a line of code. Next, press F11 to debug-run the program and the execution will stop at the first breakpoint. You can then use F6 to step over each line of code and inspect the variables in your program to check what has happened.

   Another way to check your Java code is to add System.out.println() to your code to print out the values of important variables.

3. If your computer program hangs after processing some lines of a given data file and the hanging happens at difference places of the data file in different run, very much likely there are bugs in your regular expression. You may also notice that your program may work perfectly fine on a smaller file. This is because on some data, the bugs in your regular expression are not triggered.

   One way to confirm this bug is to change your regular expression to a very simple one and then run the program against the same data file. If the program does not hang, then you know that the problem is with your regular expression.

4. Regular expressions can significantly simplify tasks but alone they do *not* always meet all the related requirements of a given application. In this project, you will notice that even with a great regular expression, you may still get some email addresses that pass the regular expression but do not look real. One such example is *//-dyn.politico.com/unsubscribe.cfin?email=nora.toiv@gmail.com* This is normal. In real life, additional cleaning (beyond regular expression) is needed. This additional cleaning is beyond the scope of this project and is *not* required.

5. The data file given in this project may contain typos and errors that cannot be corrected by regular expressions. One such example is *information@publicpolicypolling. com*

   Note the space after the dot in this example. In this project, do **not** try to correct any typos/errors of this type.

6. Clinton's 7000 emails released by US Department of State can be found at

   https://archive.org/stream/hillary-clinton-emails-august-31-release/hillary-clinton-emails-august-31-release_djvu.txt

   A local copy of these 7000 emails can be found at

   https://users.cs.jmu.edu/wangxx/web/tools/code/hillary-clinton-emails-august-31-release_djvu_copy.txt

   When you develop your computer program, you can test it against the above file, which is much smaller than the project file with all 33, 727 emails. This should make your unit testing much easier.

   After the unit test, you must run your program against the project file with 33, 723 emails.

5.9 Most commonly seen mistakes (i.e., you should not be doing these)

Please note that the following behaviors are wrong.

1. Do not upload your file *YOURJMUEID_all_emailaddresses.txt* to Canvas.
2. Do not upload your file *YOURJMUEID_sorted_emailaddresses.txt* to Canvas.
3. Only run your program on the 7000-email file.