CPSC 3400 Languages and Computation
Winter 2024

Homework 6
Due: Wednesday, February 28 at 10:00pm

To start, copy all of the files from the hw6 directory to a local directory:

```
cp /home/fac/bdiazacosta/cpsc3400/hw6/* .
```

Part 1 – Regular Expressions

For each item, create a ***single*** regular expression that matches that item. Note that in ALL cases, the entire string must match without additional characters.

a. A regular expression that matches string that only contain (uppercase or lowercase) letters *and* contains somewhere in the string ***both*** of these substrings "qu" and "zz". Additional notes:

- The string is rejected if it contains anything other than a letter.
- The substrings "qu " and "zz" must be lowercase letters only.

b. A regular expression that matches a phone number in either of the following formats

- `(555)123-4567`
- `(555) 123-4567`
- `555-123-4567`

Additional notes:

- Each digit in the format can be any digit from 0 to 9.
- There is exactly one space after the `)` in the second format.
- The string is rejected if it does not match any of these formats even if it resembles a phone number.

c. A regular expression that matches a string representing an F# list of positive integers (greater than zero) such as: `[1; 4; 6; 12; 3; 70]` Additional notes:

- There must be a semicolon after each integer except for the last integer. There cannot be a semicolon after the last integer.
- Spacing is optional but allowed between individual elements. Individual elements consist of braces, integers, and semicolons. The space between individual elements is not limited to one space and can be arbitrarily large.
- The string must start with a `[` and end with a `]`. Space before and/or after the list is not permitted.
- Leading zeros on the integers are not permitted.
- The list must have at least one integer.

d. Matches expressions that match the C++ **?:** operator. If the expression matches, use a substitution to convert the expression into a Python if-else expression. For example, the expression `a < b ? x : 3 + y` is converted into "`x if a < b else 3 + y`

Additional notes:

- Zero or more spaces may separate the **?** and the **:** from the other parts of the expression. When performing the substitution, use one space before and after if and else.
- The other three parts of the expression may contain any characters except another **?** or **:**. It may include spaces or other punctuation marks (like the example above) but each part must contain at least one character that is not a space.
- The expression is not considered to match if it does not contain the ?: operator or contains multiple **?** or multiple **:**. If the expression does not match, no substitution or conversion takes place.
- Do not worry if the other parts of the expression consist of valid C++ and/or Python expressions. In addition, you may assume the **?:** is not contained in parentheses (or other form of braces) like this: `(isEven ? z : 0) + 6`

To start, use the file `hw6.py` (one of the files you copied above). In the file, you will notice four `re.compile` statements for variables a-d. These variables refer to parts a-d respectively. All you need to do is fill in the patterns for each of these statements. For part d, you will also need to initialize a string that is used in the substitutions. Patterns that are too long to fit on one line can be split into multiple lines like this:

```
x = re.compile(r"dog|"
               r"cat|"
               r"cow|"
               r"duck")
```

The file also contains a section for tests and includes a test that matches and a test that does not match for parts a-c. It also contains a test for part d. You should add additional tests before submitting. It is not necessary to remove the tests before submission.

Part 2 – DFAs

Create DFAs for the following language specifications. The DFA must be expressed as files that can be used as inputs for the DFA simulator described below.

1. (Filename: `dfa1.txt`) All strings on Σ = {A, B} that consist of an odd number of As followed by an odd number of Bs such as ABBB or AAAAAB or AAABBB. All A characters must be before all B characters in the string.

2. (Filename: `dfa2.txt`) All strings on Σ = {A, B, C, D} that end with an A or B and meet at least one of the following conditions:
   - The string contains three consecutive Ds
   - The string contains the substring ABA.
   - The string contains a substring consisting of two Cs separated by only zero or more As

The following strings are accepted:

ABCDDDACA	contains three consecutive Ds and ends with A
ABA	contains ABA and ends with A
CABADB	contains ABA and ends with B
ABCAAACDCB	contains two Cs separated by only zero or more As (in this case – three) and ends with B
CCA	contains two Cs separated by only zero or more As (in this case – zero) and ends with A

3. (Filename: `dfa3.txt`) All strings on Σ = {A, B, C} that adhere to this Python regular expression: `^((B|C)+(AC)*(C?|B+|AB))$`

DFA Simulator

To run the simulator, the file `dfa.py` (one of the files you copied above) needs to be in the current directory:

`python3 dfa.py` *<DFA file>* *<strings file>*

where *<DFA file>* is a file that describes the DFA (and the file you submit for each problem) and *<strings file>* is a set of test strings.

The *DFA file* describes the DFA using the following format:

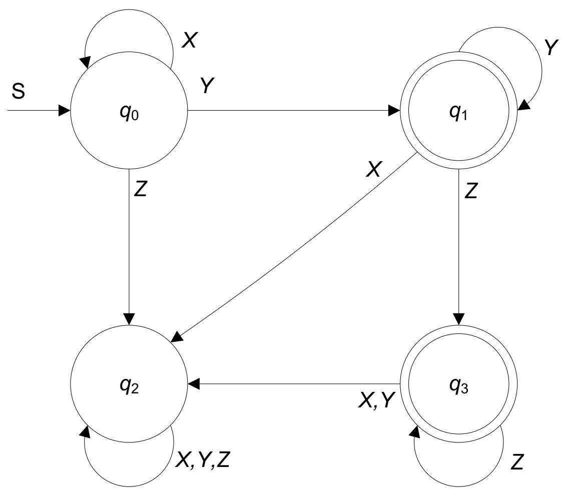
- The first line will contain one or more characters that define the input alphabet.
  - Each character in the input alphabet is separated by a single space.
  - Each character in the input alphabet is a unique *single* letter or digit.
- All subsequent lines will refer to a different state in DFA. The second line in the file (the first line after the input alphabet definition line) will be state 0. The third line in the file will be state 1, the fourth line in the file will be state 2, and so forth.
  - The first character in the line will either be a '+' (accepting state) or '–' (rejecting state).

  - Then there will be *n* nonnegative integers where *n* is the number of characters in the input alphabet. Each integer refers to the destination state for a transition for the corresponding input character.
  - The corresponding input character is based on the order of the states and the order of the input characters on the first line of the file. If the input alphabet line is "D O G". The first integer the destination state for input D, the second is for input O, the third is for input G.
  - The integers are separated by a single space. There is also a single space between the first character (+ or –) and the first integer.
- State 0 is the starting state.
- The simulator does not have any explicit error checking for incorrectly formatted files. Many errors (but not necessarily all of them) will cause an exception to be thrown, aborting the program.

The *strings file* contains one or more strings that will be executed in the DFA.
- Each line will be interpreted as a test string.
- Each string will be processed by the DFA and the simulator will print out the final state along with whether the string is accepted or rejected.
- The lines in the string file must only contain characters from the DFA alphabet. If a letter outside the alphabet is found, a `keyError` exception will abort the script.
- You will need to create your own test scripts but there is no requirement to hand them in.

Files `dfa1.txt`, `dfa2.txt`, and `dfa3.txt` are provided. They contain a single line with the input alphabet for that problem.

Example DFA:



Input file for DFA (available as `sample-dfa.txt`):

```
X Y Z
- 0 1 2
+ 2 1 3
- 2 2 2
+ 2 2 3
```

Sample strings file for above DFA (available as `sample-tests.txt`):

```
X
Y
Zc
XY
XYZ
XXXYYYZZZ
XYZYX
YZZZZ
XXX
YYY
ZZZ
```

Output from simulator:

```
String: X Final state: 0 REJECTED
String: Y Final state: 1 ACCEPTED
String: Z Final state: 2 REJECTED
String: XY Final state: 1 ACCEPTED
String: XYZ Final state: 3 ACCEPTED
String: XXXYYYZZZ Final state: 3 ACCEPTED
String: XYZYX Final state: 2 REJECTED
String: YZZZZ Final state: 3 ACCEPTED
String: XXX Final state: 0 REJECTED
String: YYY Final state: 1 ACCEPTED
String: ZZZ Final state: 2 REJECTED
```

Submitting your Assignment

On `cs1`, run the following script in the directory with your program:

```
/home/fac/bdiazacosta/submit/cpsc3400/hw6_submit
```

This will copy the files `hw6.py`, `dfa1.txt`, `dfa2.txt`, and `dfa3.txt` to a directory that can be accessed by the instructor. Please be sure to keep the same file names or the submission program will not work. Only the last assignment submitted before the due date and time will be graded. ***Late submissions are not accepted and result in a zero.***

Grading

Grading is based on the following rubric. A test that does not finish within a reasonable time is considered to be a failing test.

Regular Expressions a-c *18 points – 6 points each*

• Passes all accepting tests:	(no penalty)
• Fails one or two accepting tests:	-2
• Fails three or more accepting tests:	-3
• Passes all rejecting tests:	(no penalty)
• Fails one or two rejecting tests:	-2
• Fails three or more rejecting tests:	-3

Exceptions:

- If a regular expression accepts all tests or rejects all tests, a 5 point deduction will be assessed instead (6 points if the regular expression is mostly incomplete).
- Regular expressions that don't compile will receive a zero.

Regular Expression d (substitution) *8 points*

• Passes all tests:	(no penalty)
• Fails one or two tests:	-2
• Fails three tests:	-5
• Fails four or more tests:	-8

DFAs *24 points – 8 points each*

• Passes all accepting tests:	(no penalty)
• Fails one or two accepting tests:	-2
• Fails three or more accepting tests:	-4
• Passes all rejecting tests:	(no penalty)
• Fails one or two rejecting tests:	-2
• Fails three or more rejecting tests:	-4

Exceptions:

- If a DFA accepts all tests or rejects all tests, a 7 point deduction will be assessed instead (8 points if the DFA is mostly incomplete).
- If a DFA crashes, possibly due to a file formatting error, it will be considered a failing test. Obviously, if all tests crash, then you will receive a zero for that DFA.