- This assignment is **due in Week 4 on Sunday 28 August, 11:59pm**. You will make four submissions:
  - Submit your answers to problems 1 and 4 to Gradescope (text entry boxes)
  - Submit your answers to problems 2 and 5 to Gradescope (as plaintext files)
  - Submit your answer to problem 3 to Gradescope (as a pdf file)
  - Submit your answer to problem 6 to Gradescope (source code)
- All work must be **done individually** without consulting anyone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.
- You will be evaluated not just on the correctness of your answers, but on your ability to present your ideas clearly and logically. **You should always explain how you arrived at your answer unless explicitly asked not to do so.** Your goal should be to convince the person reading your work that your answers are correct and your methods are sound.
- For clarifications and more details on all aspects of this assignment (e.g., level of justification expected, late penalties, repeated submissions, what to do if you are stuck, etc.) you are expected to regularly monitor the Ed Forum post "Assignment FAQ".

Problem 1. (10 marks, 5 marks each)

Let $\Sigma = \{a, b\}$. For each of the following regular expressions, provide a clear and concise English description for the language of that regular expression. No justification is necessary.

1. $(a|b)^*((aaa)|(bbb))(a|b)$
2. $b^*((ab)b^*)^*(a|\epsilon)$

Problem 2. (15 marks, 5 marks each)

Let $\Sigma = \{a, b\}$. For each of the following languages, provide a regular expression for that language. No justification is necessary.

1. The set of strings that contain at most two b.
2. The set of strings that have even length or contain no a, but not both.
3. The set of strings that contain exactly two out of the four length 2 strings (aa, ab, ba, bb) as substrings.

The expressions should be submitted in separate files `problem_2_1.txt`, `problem_2_2.txt` and `problem_2_3.txt`, for parts 1, 2 and 3 respectively. The only thing in each file should be a regular expression. You can write `epsilon` to denote ϵ and use the normal ASCII characters `|` and `*` to denote union and star closure. Please see this post on Ed for more detail: "Regex submission format"

Problem 3. (15 marks) Give a recursive procedure that decides, given R, if $L(R)$ is infinite. Explain why each case is correct.

Hint: You may find it helpful to design a procedure that obtains and uses some additional information about the subexpressions beyond whether their languages are infinite.

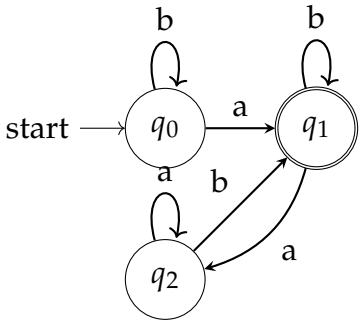
Note: Your solution should be a recursive procedure operating directly on R. For example, using the RE to NFA conversion (from week 3) is not allowed.

At least 5 marks will be awarded for the base cases and union case.

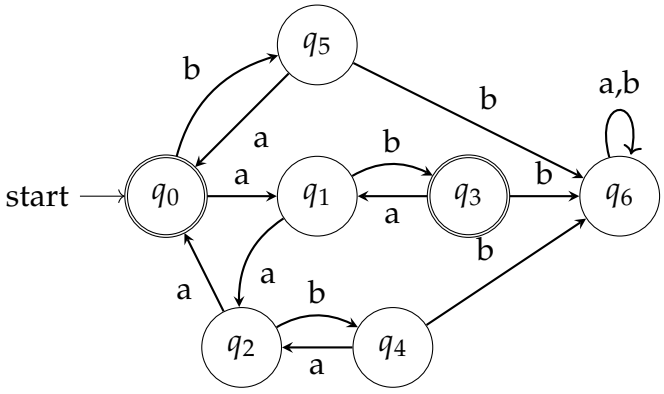
Problem 4. (10 marks, 5 marks each)

Let $\Sigma = \{a, b\}$. For each of the following deterministic finite automata, provide a clear and concise English description for the language of that automaton. No justification is needed.

1.



2.



Problem 5. (10 marks, 5 marks each)

Let $\Sigma = \{a, b\}$. For each of the following languages, provide a deterministic finite automaton for that language. No justification is necessary.

1. The set of strings where the first letter is different from the last letter (in particular, the length is at least 2).
2. The set of strings where the third last letter is the same as the last letter (in particular, the length is at least 3).

The DFA should be submitted in separate files `problem_5_1.txt` and `problem_5_2.txt`, for parts 1 and 2 respectively. The format is described in this post on Ed: "DFA/NFA submission formats"

For example:

```
Sigma = a b
Q = q0 q1 q2 q3
start = q0
F = q1 q2
q0 a q1
q0 b q0
q1 a q1
q1 b q2
q2 a q2
q2 b q3
q3 a q3
q3 b q3
```

Problem 6. (20 marks)

Let $rotate : \Sigma^* \to \Sigma^*$ be the function that moves the first letter of a string to the end. For example, $rotate(bbcaabc) = bcaabcb$, and $rotate(c) = c$. We define $rotate(\epsilon) = \epsilon$.

We define the rotation of a language L, denoted $R(L)$, as the language obtained from applying *rotate* to each string in the original language.

More formally,

$$R(L) = \{rotate(y) : y \in \Sigma^*\}$$

For example, $R(\{a, ba, abc\}) = \{a, ab, bca\}$, and $R(L(a^*b)) = L((a^*ba)|b)$.

Your task is to write code that takes an NFA N from stdin, and outputs an NFA for $R(L(N))$ to stdout. Some skeleton code will be provided (in Python), that demonstrates how to handle the input/output.

Input will be given in the same format as the DFA in problem 5, except there may also be epsilon transitions, where the word `epsilon` is written instead of an alphabet symbol, and there may be more or less than 1 transition per state and symbol pair. The format is described in this post on Ed: "DFA submission format, NFA input format"

You may assume that the input is a valid description of an NFA. Your program will not be tested on invalid input.

Your output is expected in the same format.

For example, suppose the input were:

```
Sigma = 0 1
Q = q0 q1 q2 q3
start = q0
F = q1 q3
q0 0 q1
```

```
q1 1 q2
q2 1 q3
```

This recognises the (finite) language $L = \{0, 011\}$. Since $R(L) = \{0, 110\}$, a correct output would be:

```
Sigma = 0 1
Q = q0 q1 q2 q3 q4
start = q0
F = q1 q4
q0 0 q1
q0 1 q2
q2 1 q3
q3 0 q4
```

Two specific input cases are provided. At least 10 marks will be awarded for these cases. For these cases alone, hardcoding is permitted. You are permitted to solve these problems by hand and write a program that outputs your manual solutions, and a correct submission that does this will be awarded at least 10 marks. On the other hand, if you are able to solve the general problem, you may use your general solution for these cases, in which case you will not need to solve these test cases manually.

Input case 1:

```
Sigma = 0 1
Q = q0 q1
start = q0
F = q1
q0 0 q0
q0 1 q0
q0 1 q1
```

Input case 2:

```
Sigma = 0 1
Q = q0 q1 q2
start = q0
F = q2
q0 0 q0
q0 0 q2
q0 1 q0
q0 1 q1
q1 0 q2
q1 1 q2
q2 0 q1
q2 1 q1
```

For full marks, your program should work for any input and be reasonably efficient. Guidelines on this efficiency will be posted on Ed.