ECEN 227 - Introduction to Finite Automata and Discrete Mathematics

Dr. Mahmoud Nabil
mnmahmoud@ncat.edu

North Carolina A & T State University

October 7, 2019

Talk Overview

1. Algorithms
2. Analysis of Algorithms
3. Asymptotic growth of functions
4. More on the Analysis of Algorithms
5. Finite state machine

Outline

Algorithms

Algorithm

An algorithm is a step-by-step method for solving a problem.

Pseudocode

Algorithms are often described in pseudocode, which is a language in between written English and a computer language.

Ex.

- A recipe is an example of an algorithm in which :
  - Ingredients are the input.
  - Final dish is the output.
  - A sequence of steps to follow recipe.

Example

Algorithm 1 Sum of Three Numbers Algortihm Name

This algorithm finds the sum of three numbers Algortihm Description

Input: real numbers a,b, c Algorithm inputs

Output: Sum of a,b c Algorithm output

1: sum := a+b+c Assignment operation (variable is given a value)

2: **return** sum The output of an algorithm is specified by return statement

Control flow statements

- The statements inside your algorithm (recipe) are generally executed from top to bottom, in the order that they appear.
- Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of statements.

Ex.

- If statement.
- If Else statement
- For loop statement.
- While loop.

If Statement

If Statement

An if-statement tests a condition, and executes one or more instructions if the condition evaluates to true.

Algorithm 2 If statement

```
if Condition1 then
    ...          Executed only if Condition1 is met
    ...          Executed only if Condition1 is met
end if
...              Executed normally
```

If Else Statement

If Else Statement

An if-else-statement tests a condition, executes one or more instructions if the condition evaluates to true, and executes a different set of instructions if the condition evaluates to false.

Algorithm 3 If else statement

```
if Condition1 then
    ...          Executed only if Condition1 is met
else if Condition2 then
    ...          Executed only if Condition2 is met and Condition1 does not met
end if
...              Executed normally
```

If Else Statement

If Else Statement

An if-else-statement tests a condition, executes one or more instructions if the condition evaluates to true, and executes a different set of instructions if the condition evaluates to false.

Algorithm 4 If else statement

```
if Condition1 then
    ...        Executed only if Condition1 is met
else if Condition2 then
    ...        Executed only if Condition2 is met and Condition1 does not met
else if Condition3 then
    ...        Executed only if Condition2 is met and both Condition1 and Condition2 does not met
end if
...            Executed normally
```

Example if

Algorithm 5 Smallest of three

This algorithm finds the minumum of three numbers   Algortihm Description

Input: Real numbers a,b, c   Algorithm inputs

Output: Minumum of a,b c   Algorithm output

```
min:=a
if b < min then
    min:=b
end if
if c < min then
    min:=c
end if
return min
```

For Statement

For Statement

In a for-loop, a block of instructions is executed a fixed number of times as specified in the first line of the for-loop, which defines an index, a starting value for the index, and a final value for the index.

Algorithm 6 For statement

for $j = 1$ to N **do**
    ... Executed N times
end for
... Executed normally

Example for

Algorithm 7 Find smallest in sequence

Input:

1- Sequence of numbers $a_1, a_2, \ldots, a_n$

2- n number of inputs

Output: Minumum of $a_1, a_2, \ldots, a_n$

```
min := a_1
for i = 2 to n do
    if a_i < min then
        min := a_i
    end if
end for
return min
```

While Statement

While Statement

A while-loop iterates an unknown number of times, ending when a certain condition becomes false.

Algorithm 8 While statement

```
1: while Condition1 do
2:     ...            Executed as long as Condition1 is met
3: end while
4: ...                Executed normally
```

Example While

Algorithm 9 Search for a number in a sequence

Input:

1- Sequence of numbers $a_1, a_2, \ldots, a_n$

2- n number of inputs

3- x a number to search for

Output: Index of first occurence of x in the sequence or -1 if x does not occur in the sequence

i := 1
while $a_i \neq x$ and $i < n$ **do**
    i:= i + 1
end while
if a_i = x **then**
    return i
end if
return -1

Example Nested Loop

Algorithm 10 Count duplicates

Input:

1- Sequence of numbers $a_1, a_2, \ldots, a_n$

2- n number of inputs

Output: count: the number of duplicate pairs

```
count := 0
for i := 1 to n-1 do
    for j := i + 1 to n do
        if a_i == a_j then
            count := count + 1
        end if
    end for
end for
return count
```

Outline

Time Complexity

- Given an input of size n to the algorithm, what is the lower bound and the upper bound of the number of operations to be executed?

Time Complexity

- Given an input of size n to the algorithm, what is the lower bound and the upper bound of the number of operations to be executed?
- What are the operations we care for?
  - Assignment operation
  - Arithmetic operations.
  - Comparison operation.
  - Return statements.

Example Time Complexity

Algorithm 11 Compute Sum

Input:

1- Sequence of numbers $a_1, a_2, \ldots, a_n$

2- n number of inputs

Output: Sum of the sequence

```
sum := 0
for i := 1 to n do
    sum := sum + a_i
end for
return sum
```

Example Time Complexity

Algorithm 12 Compute Sum

Input:

1- Sequence of numbers $a_1, a_2, \ldots, a_n$

2- n number of inputs

Output: Sum of the sequence

```
sum := 0                    1 assignment op
for i := 1 to n             For loop compare i and assign i (2 ops)  do
    sum := sum + a_i        1 addition and 1 for assignment (2 ops)
end for
return sum                  1 return op
```

- Time Complexity = f(n) = # of operations on a sequence of length n
- $f(n) = 1 + 2n + 2n + 1 = 4n + 2$

Time Complexity

- In evaluating algorithms, the focus is on how the function f grows with n, ignoring small input sizes and constant factors that depend on the specifics of the implementation and have less impact on the execution time.

Time Complexity

- In evaluating algorithms, the focus is on how the function f grows with n, ignoring small input sizes and constant factors that depend on the specifics of the implementation and have less impact on the execution time.
- Thus, we introduce the notion of the asymptotic time complexity.

Time Complexity

- In evaluating algorithms, the focus is on how the function f grows with n, ignoring small input sizes and constant factors that depend on the specifics of the implementation and have less impact on the execution time.
- Thus, we introduce the notion of the asymptotic time complexity.

Asymptotic time complexity

Asymptotic time complexity of an algorithm is the rate of asymptotic growth of the algorithm's time complexity with the input size.

Outline

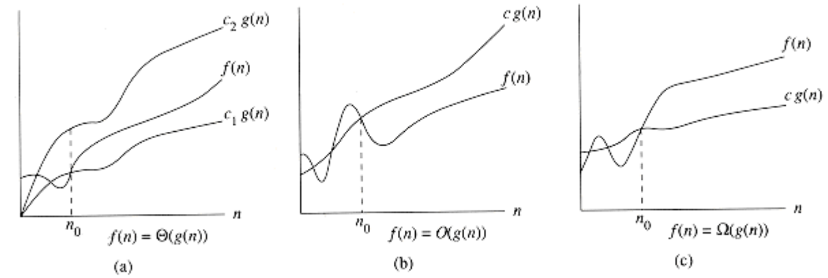
The asymptotic growth

The asymptotic growth

The asymptotic growth of the function f is a measure of how fast the output f(n) grows as the input n grows.

- Three classification of functions using O, Ω,, and Θ notation (called asymptotic notation).
- Asymptotic notation is a useful tool for evaluating the efficiency of algorithms.

The asymptotic growth



Big O notation

Big O

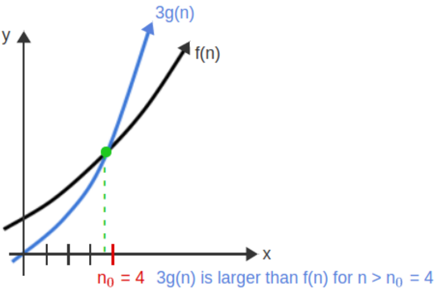
Let f and g be two functions from Z^+ to Z^+. Then f $=$ O(g) if there are positive constants c and n_0 such that for any $n \geq n_0$, f(n) $\leq$ c . g(n).

The constants c and n_0 in the definition of Oh-notation are said to be a witness to the fact that f $=$ O(g).

$f(n) = 2n^3 + 3n^2 + 7$

$g(n) = n^3$

To prove f is O(g) Pick c = 3 $n_0 = 4$



Big O notation

f(n) $= 3n^3 + 5n^2 - 7$
g(n) $= n^3$
Claim: f = O(g).
Proof.

Select c = 8 and $n_0 = 1$.

Big O notation

f(n) $= 3n^3 + 5n^2 - 7$
g(n) $= n^3$
Claim: f = O(g).
Proof.

Select c = 8 and $n_0 = 1$.

- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^2$

Big O notation

f(n) $= 3n^3 + 5n^2 - 7$
g(n) $= n^3$
Claim: f = O(g).
Proof.

Select c = 8 and $n_0 = 1$.

- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^2$
- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^3$

Big O notation

f(n) $= 3n^3 + 5n^2 - 7$
g(n) $= n^3$
Claim: f = O(g).
Proof.

Select c = 8 and $n_0 = 1$.

- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^2$
- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^3$
- $3n^3 + 5n^2 - 7 \leq 8n^3$

Big O notation

f(n) $= 3n^3 + 5n^2 - 7$
g(n) $= n^3$
Claim: f = O(g).
Proof.

Select c = 8 and $n_0 = 1$.

- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^2$
- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^3$
- $3n^3 + 5n^2 - 7 \leq 8n^3$
- f(n) $\leq$ 8 g(n)

Big O notation

f(n) $= 3n^3 + 5n^2 - 7$
g(n) $= n^3$
Claim: f = O(g).
Proof.

Select c = 8 and $n_0 = 1$.

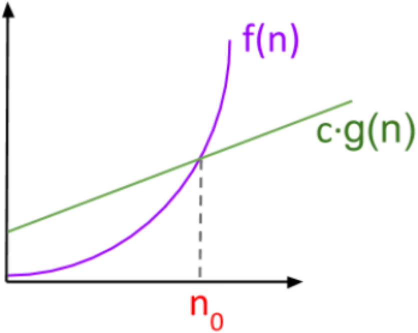
- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^2$
- $3n^3 + 5n^2 - 7 \leq 3n^3 + 5n^3$
- $3n^3 + 5n^2 - 7 \leq 8n^3$
- f(n) $\leq$ 8 g(n)
- f(n) = O(g(n))

Big Omega notation

Big Omega Ω

Let f and g be two functions from Z^+ to Z^+. Then f $= \Omega$(g) if there are positive constants c and n_0 such that for any $n \geq n_0$, f(n) $\geq$ c . g(n).

The constants c and n_0 in the definition of Oh-notation are said to be a witness to the fact that f $=$ O(g).



Big Omega notation

f(n) $= \frac{1}{2}n^2 + 7n + 3$
g(n) $= n^2$
Claim: f $= \Omega$(g).

Proof.

Select c $= \frac{1}{2}$ and $n_0 = 1$.

- n $\geq$ 0

Big Omega notation

f(n) $= \frac{1}{2}n^2 + 7n + 3$
g(n) $= n^2$
Claim: f $= \Omega$(g).
Proof.

Select c $= \frac{1}{2}$ and $n_0 = 1$.

- n $\geq$ 0
- $\frac{1}{2}n^2 + 7n + 3 \geq \frac{1}{2}n^2$

Big Omega notation

f(n) $= \frac{1}{2}n^2 + 7n + 3$
g(n) $= n^2$
Claim: f $= \Omega$(g).
Proof.

Select c $= \frac{1}{2}$ and $n_0 = 1$.

- n ≥ 0
- $\frac{1}{2}n^2 + 7n + 3 \geq \frac{1}{2}n^2$
- f(n) $\geq \frac{1}{2}$ g(n)

Big Omega notation

f(n) $= \frac{1}{2}n^2 + 7n + 3$
g(n) $= n^2$
Claim: f $= \Omega$(g).
Proof.

Select c $= \frac{1}{2}$ and $n_0 = 1$.

- n ≥ 0
- $\frac{1}{2}n^2 + 7n + 3 \geq \frac{1}{2}n^2$
- f(n) $\geq \frac{1}{2}$ g(n)
- f(n) $= \Omega$(g(n))

Relationship of Oh-notation and Ω-notation.

Theorem

Let f and g be two functions from Z^+ to Z^+. Then $f = \Omega(g)$ if and only if $g = O(f)$.

Θ Notation

Θ Notation

Let f and g be two functions Z^+ to Z^+. f $=$ Θ(g) if f $=$ O(g) and f $=$ Ω(g).

Ex.

- f(n) $= 4n^3 + 7n + 16$ 7n and 16 are called the lower order terms
- f(n) $= \Theta(n^3)$

Theorem

Let p(n) be a degree-k polynomial of the form in which $a_k > 0$.

$$(a_k)n^k + (a_{k-1})n^{k-1} + \ldots + (a_1)n + a_0$$

Then p(n) is $\Theta(n^k)$.

Algorithmic Complexity Function

Function	Name
$\Theta(1)$	Constant
$\Theta(\log \log n)$	Log log
$\Theta(\log n)$	Logarithmic
$\Theta(n)$	Linear
$\Theta(n \log n)$	n log n
$\Theta(n^2)$	Quadratic
$\Theta(n^3)$	Cubic
$\Theta(c^n)$, $c > 1$	Exponential
$\Theta(n!)$	Factorial

Algorithmic Complexity Function

f(n)	n=10	n=50	n=100	n=1000	n=10000	n=100000
$\log_2 n$	3.3 μs	5.6 μs	6.6 μs	10.0 μs	13.3 μs	16.6 μs
n	10 μs	50 μs	100 μs	1000 μs	10 ms	.1 s
$n \log_2 n$	.03ms	.28 ms	.66 ms	10.0 ms	.133 s	1.67 s
n^2	.1 ms	2.5 ms	10 ms	1 s	100 s	2.8 hours
n^3	1 ms	.125 s	1 s	16.7 min	11.6 days	31.7 years
2^n	1.0 ms	35.7 years	4.0×10^{16} years	3.4×10^{287} years	6.3×10^{2996} years	*

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$
    - $\Theta(n^8)$
- f(n) $= 2 * 3^n$

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$
  - $\Theta(n^8)$
- f(n) $= 2 * 3^n$
  - $\Theta(3^2)$
- f(n) $= 2^n + 3^n$

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$
  - $\Theta(n^8)$
- f(n) $= 2 * 3^n$
  - $\Theta(3^2)$
- f(n) $= 2^n + 3^n$
  - $\Theta(3^n)$
- f(n) $= 9(n\ log\ n) + 5(log\ log\ n) + 5$

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$
  - $\Theta(n^8)$
- f(n) $= 2 * 3^n$
  - $\Theta(3^2)$
- f(n) $= 2^n + 3^n$
  - $\Theta(3^n)$
- f(n) $= 9(n\ log\ n) + 5(log\ log\ n) + 5$
  - $\Theta(n\ log\ n)$
- f(n) $= n\ log_{37}\ n$

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$
  - $\Theta(n^8)$
- f(n) $= 2 * 3^n$
  - $\Theta(3^2)$
- f(n) $= 2^n + 3^n$
  - $\Theta(3^n)$
- f(n) $= 9(n\ log\ n) + 5(log\ log\ n) + 5$
  - $\Theta(n\ log\ n)$
- f(n) $= n\ log_{37}\ n$
  - $\Theta(n\ log\ n)$
- f(n) $= n^{21} + (1.1)^n$

Excercise

Characterize the rate of growth of each function f below by giving a function g such that $f = \Theta(g)$.

- f(n) $= n^8 + 3n - 4$
  - $\Theta(n^8)$
- f(n) $= 2 * 3^n$
  - $\Theta(3^2)$
- f(n) $= 2^n + 3^n$
  - $\Theta(3^n)$
- f(n) $= 9(n\ log\ n) + 5(log\ log\ n) + 5$
  - $\Theta(n\ log\ n)$
- f(n) $= n\ log_{37}\ n$
  - $\Theta(n\ log\ n)$
- f(n) $= n^{21} + (1.1)^n$
  - $\Theta(1.1^n)$

Outline

Example 1

Algorithm 13 Find smallest in sequence

Input:

1- Sequence of numbers $a_1, a_2, \ldots, a_n$

2- n number of inputs

Output: Minumum of $a_1, a_2, \ldots, a_n$

min $:= a_1$ 1 assignment op
for $i = 2$ to n For loop compare i and assign i (2 ops) **do**
  if $a_i < min$ 1 op for comparison + 1 op (in worst-case) for assignment **then**
    min $:= a_i$
  end if
end for
return min 1 return op

- f(n) = 4(n-1) + 2 = c (n-1) + d = $\theta(n)$

Example 2

Algorithm 14 Search for a number x in a sequence

Input: $a_1, a_2, \ldots, a_n$, n, x

Output: Index if found or -1 if not found

```
i := 1                          1 aasign op
while a_i ≠ x and i < n         3 op = 2 compare and 1 logic and   do
    i:= i + 1                   2 op = 1 add + 1 assign (worst case)
end while
if a_i = x                      1 compare op   then
    return i                    1 return op
end if
return -1                       1 return op
```

- f(n) = # of ops on sequence of length n
- $f(n) \leq 1 + \underbrace{(3)(n)}_{\text{while loop condition}} + \underbrace{(1)(n-1)}_{\text{while loop body}} + 2 \leq 4n = O(n)$

Worst-case complexity

- The number of operations performed by the previous algorithm may depend on the actual data in the input sequence not just the sequence size.
  - What if x is the first element in the sequence? Best Case
  - What if x is the last element in the sequence or does not exist? Worst Case

Worst Case Complexity

Worst-case time complexity

It is defined to be the maximum number of atomic operations the algorithm requires, where the maximum is taken over all inputs of size n.

- Usually we care about the worst case in our analysis.

Worst Case Complexity

Worst-case time complexity

It is defined to be the maximum number of atomic operations the algorithm requires, where the maximum is taken over all inputs of size n.

- Usually we care about the worst case in our analysis.
- We define lower and upper bound for the worst case.

Worst Case Complexity

Worst-case time complexity

It is defined to be the maximum number of atomic operations the algorithm requires, where the maximum is taken over all inputs of size n.

- Usually we care about the worst case in our analysis.
- We define lower and upper bound for the worst case.

Analysis of Nested Loop

Algorithm 15 Count duplicates

Input: $a_1, a_2, \ldots, a_n$, n

Output: count: the number of duplicate pairs

```
count := 0
for i := 1 to n-1 do
    for j := i + 1 to n do
        if a_i == a_j then
            count := count + 1
        end if
    end for
end for
return count
```

Analysis of Nested Loop

Algorithm 16 Count duplicates

Input: $a_1, a_2, \ldots, a_n,\ n$

Output: count: the number of duplicate pairs

```
count := 0
for i := 1 to n-1 do
    for j := i + 1 to n do
        if a_i == a_j then
            count := count + 1
        end if
    end for
end for
return count
```

- f(n) =

$$\underbrace{c}_{\text{inner loop ops}} \ [(n\text{-}1) + (n\text{-}2) + \ldots + 1] + \underbrace{[b + b + \cdots + b]}_{\text{n-1 outer loop}} + \underbrace{d}_{\text{before or after loops}}$$

Analysis of Nested Loop

$$\text{f(n)} = \underbrace{c}_{\text{inner loop ops}} \quad [(\text{n-1}) + (\text{n-2}) + \ldots + 1] + \underbrace{[b + b + \cdots + b]}_{\text{n-1 outer loop}} + \underbrace{d}_{\text{before or after loops}}$$

Note.

- $[(n-1) + (n-2) + \ldots + 1] = \frac{n(n-1)}{2}$
- $\text{f(n)} = c_1 n^2 + c_2 n + c_3 = \Theta(n)$

Outline

Introduction

- FSMs are the simplest model of computation.
- Finite State Machines (FSMs) are (essentially) computers with very small memory
- FSMs are widely used in practice for simple mechanisms: automatic doors, lifts, microwave or washing machine controllers, and many other electromechanical devices.

Example

- For example, in the case of a parking ticket machine, it will not print a ticket when you press the button unless you have already inserted some money.
- The response to the print button depends on the previous history of the use of the system: its memory.

Inputs

What **stimulus (input)** does a ticket machine take account of ?

Inputs

What **stimulus (input)** does a ticket machine take account of ?

- insert some money m,
- press the print ticket button t,
- press the cancel button for refunds r

The alphabet of inputs is a set: $I = \{m, t, r\}$

The machines memory is represented by a set of states.

For example 1=awaiting coins, 2=ready to print 3=finished printing

The set of possible states for a given machine is written $Q = \{1, 2, 3\}$

States are drawn as circles in FSM diagrams.

There are only a finite number of possible states allowed for a Finite State Machine.

Transation

How does computation occur?

The machine has transitions from one state to another depending on the **stimulus (input)** provided.

The **transition function** is of type:

$$T : Q \times I \rightarrow Q$$

Transitions are drawn as edges between the states in FSM diagrams.

Edges are labelled with the input symbol for the transition. For every state, symbol pair there must be a transition to some other state

To simplify FSM diagrams, we sometimes do not show transitions for illegal inputs.

Starting and Stopping

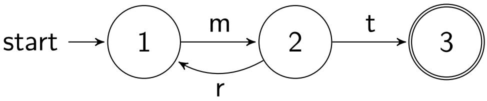
One state from Q is identified as the **starting state**. Think of this as the initial state of the machine before any inputs are received.

The start state is identified by an arrow pointing to it, but not coming from any other state.

A machine can stop in any state: input may cease, or there may be no matching transition to take.

One or more states from Q may be identified as **accepting states**. These are good places to stop. In diagrams, accepting states are denoted by a double circle

Ticket FSM



FSM with output

A finite state machine with output $o \in O$, produces a response that depends on the current state as well as the most recently received input.

For example a=Please insert coins, b=Ready to print c=Finished printing

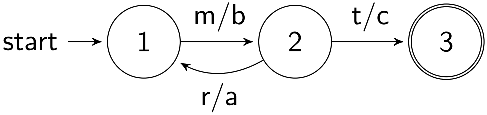
The set of possible outputs for a given machine is written $O = \{a, b, c\}$

Outputs could be written on the transition edges.

The **transition function** is of FSM with output:

$$T : Q \times I \to Q \times O$$

Ticket FSM with output



Formal defination of FSMs

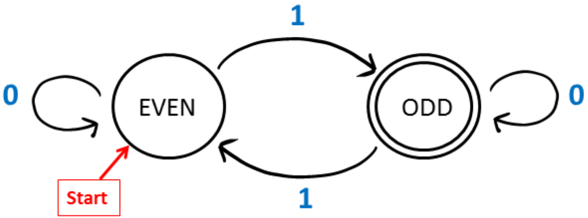
Definition: A finite state machine (FSM) is defined to be a 6-tuple $(Q, q_0, I, O, A, T,)$ where

- Q is a finite set of states;
- $q_0 \in Q$ is the start state;
- I is a finite alphabet of input symbols;
- O is a finite alphabet of output symbols;
- $A \subseteq Q$ is a set of accepting states (A may be the empty set);
- $T : Q \times I \to Q$ is the state transition function

An input string is accepted if the FSM ends up in an accepting state after each character in the string is processed.

Parity FSM

Below fsm that accepts a binary string if and only if the number of 1's in the string is odd. The property of whether a number is odd or even is called the parity of a number.



Excercise 1

Design an FSM with input alphabet $\{0, 1\}$ that accepts a string x if and only if the string has numbers of 1's is a multiple of 3. (Zero is a multiple of 3).

Excercise 2

Design an FSM with input alphabet $\{0, 1\}$ that accepts a string x if and only if the string has at least one 0 and at least one 1.

Excercise 3

Design an FSM with input alphabet $\{0, 1\}$ that accepts a string x if and only if the string has no occurrences of "00" or "11" in the string. (The empty string has no occurrences of "00" or "11".)

Questions