Discrete Math for Computer Science

Peter Schaefer

Freshman Fall

Contents

1 Computation

1.1 An introduction to algorithms

An algorithm is a step by step method for solving a problem. It usually includes:

- name
- brief description
- description of input
- description of output
- sequence of steps to follow

Algorithms are often described in **pseudocode**

Assignment operator

```
x := y
```

Return statement

```
Return(value)
```

If-else statement

```
If(x = 5), y := 7
```

```
If(condition)
  Step 1
  Step 2
  ...
  Step n
End-if
```

```
If(condition)
  Step(s)
Else
  Step(s)
End-if
```

For-loop

```
For i = s to t    <- first value is s, then s+1, until t is reached
  Step(s)
End-for
```

While-loop

```
While(condition)
  Step(s)
End-while
```

Nested Loops

```
Input: sequence a_1, ..., a_n; n

count := 0
For i = 1 to n-1
  For j = i+1 to n
    If(a_i = a_j) count := count+1
  End-for
End-for

Return(count)
```

1.2 Asymptotic growth of functions

Consider $f : \mathbb{Z}^+ \to \mathbb{R}^{\geq}$, where $\mathbb{R}^{\geq}$ denotes the set of non-negative real numbers. **Asymptotic growth** of a function f is a measure of how fast the object $f(n)$ grows as the input n grows. Classification of functions $\mathcal{O}, \Omega$, and Θ provide a way to concisely characterize the growth of a function.

$$f = \mathcal{O}(g) \quad "f \text{ is Oh of } g"$$

Constant factors

$$7n^3 \to 7 \text{ is constant factor}$$
$$5n^2 \to 5 \text{ is constant factor}$$
$$3 \to 3 \text{ is constant factor}$$

$\mathcal{O}$ notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$. Then $f = \mathcal{O}(g)$ if there is are positive real numbers c and n_0 such that for any $n \in \mathbb{Z}^+$ such that $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

Constants c and n_0 are said to be a *witness* to the fact $f = \mathcal{O}(g)$

Ω notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$. Then $f = \Omega(g)$ if there is are positive real numbers c and n_0 such that for any $n \in \mathbb{Z}^+$ such that $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

$f = \Omega(g)$ is read "f is Omega of g"

Theorem: Relationship of $\mathcal{O}$-notation and Ω-notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$. Then $f = \Omega(g) \iff g = \mathcal{O}(f)$

Θ notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$.

$$f = \Theta(g) \text{ if} : f = \mathcal{O}(g) \wedge f = \Omega(g)$$

- $f = \Theta(g)$ is read "f is Theta of g"
- if $f = \Theta(g)$, then f is said to be the *order of* g.

Theorem: Asymptotic Growth of Polynomials

Let $p(n)$ be a degree-k polynomial of the form

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 \text{ where } a_k > 0$$

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

Asymptotic Growth of Logarithm Functions with Different Bases

Let a and b be two real numbers greater than 1. Then

$$\log_a n = \Theta(\log_b n)$$

This is because of the fact that

$$\log_a n = \log_a b \cdot \log_b n, \text{ for } a, b > 1$$

when a function is said to be the $\mathcal{O}$ or Ω of a logarithm function, the base is often omitted because it is understood that as long as the base is greater than 1, the value of the base does not matter.

Growth rate of common functions

Constant Functions

- function that does not depend on n at all
- any constant function is $\Theta(1)$

Linear

- $\Theta(n)$

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
$\Theta(n^m)$ for $m \in \mathbb{Z}^+$	Power
$\Theta(c^n)$ for $c > 1$	Exponential
$\Theta(n!)$	Factorial

Rules about Asymptotic Growth

Let f, g, and h be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$.

- if $f = \mathcal{O}(h)$ and $g = \mathcal{O}(h)$, then $f + g = \mathcal{O}(h)$
- if $f = \Omega(h)$ and $g = \Omega(h)$, then $f + g = \Omega(h)$
- if $f = \mathcal{O}(g)$, $c \cdot f = \mathcal{O}(g)$, $c \in \mathbb{R}^{\geq}$
- if $f = \Omega$, $c \cdot f = \Omega(g)$, $c \in \mathbb{R}^{\geq}$
- if $f = \mathcal{O}(g)$ and $g = \mathcal{O}(h)$, then $f = \mathcal{O}(h)$
- if $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$

1.3 Analysis of algorithms

Resources an algorithm requires to run

- time, called *time complexity*
- space, called *space complexity*
- Together called **computational complexity**

```
ComputeSum
Input: a_1, a_2, ..., a_n  (n is length of sequence)
Output: the sum of the numbers in the sequence
sum := 0                        | 1 assignment operation
For i = 1 to n                  | loop iterated n times
  sum := sum + a_i              |   for loop test and increments (2 operations)
End-for                         |   1 addition and 1 assignment (2 operations)
Return(sum)                     | 1 op for return statement
```

$$\begin{aligned} f(n) &= 1 + n[2+2] + 1 \\ &= 1 + 4n + 1 \\ &= 4n + 2 \\ &= \mathcal{O}(n) \end{aligned}$$

Growth rates for different input sizes

$f(n)$	$n=10$	$n=50$	$n=100$	$n=1000$	$\cdots$
$\log_2 n$	$3.3\mu s$	$5.6\mu s$	$6.6\mu s$	$10\mu s$	$\cdots$
n	$10\mu s$	$50\mu s$	$100\mu s$	$1000\mu s$	$\cdots$
$n \log_2 n$	$.03ms$	$.28ms$	$.66ms$	$10ms$	$\cdots$
n^2	$.1ms$	$2.5ms$	$10ms$	$1s$	$\cdots$
n^3	$1ms$	$.125s$	$1s$	$16.7min$	$\cdots$
2^n	$1ms$	$35.7yrs$	$4 \times 10^{16} yrs$	$3.4 \times 10^{287} yrs$	$\cdots$

Worst-case analysis

Worst-case analysis evaluates the time complexity on the input which takes the longest time.

- upper bound: use $\mathcal{O}$-notation

  upper bound must apply for every input of size n
- lower bound: use Ω-notation

  lower bound need only apply for one possible input of n

Average-case analysis takes an average running time of algorithm on random inputs.

```
For(----)
  operations        -> linear (n)
End-for

For(----)
  For(----)
    operations      -> quadratic (n^2)
  End-for
End-for
```

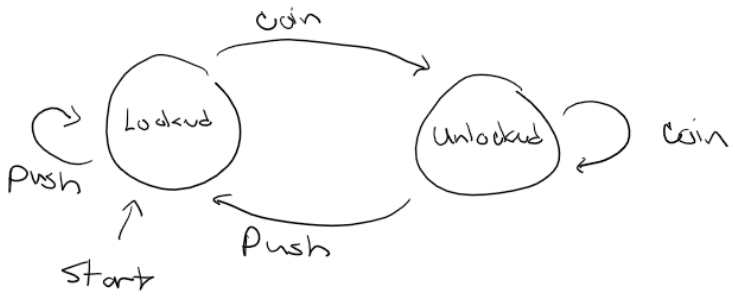
```
For(----)
  For(----)
    For(----)
      operations  -> cubic (n^3)
    End-for
  End-for
End-for
```

and so on. An algorithm runs in polynomial time if its time complexity is $\mathcal{O}^k$ for some fixed constant k. An algorithm is considered "efficient" if it runs in polynomial time. For example,

$$\mathcal{O}(n^5) \text{ is "efficient"}$$
$$\mathcal{O}(n^{\log n}) \text{ is not "efficient"}$$

1.4 Finite state machines

A **finite state machine** consists of a finite set of states, with transitions between states triggered by different input actions. A finite state machine is sometimes called *finite state automation*.



states: Q ={locked, unlocked}

The reaction of a finite state machine to the input received is denoted by a **transitive function**, often denoted by the symbol 'δ'

$$\delta([\text{state}], [\text{action}]) = [\text{state}]$$

In the case of the coin machine,

$$\delta(\text{Locked}, \text{Coin}) = \text{Unlocked}$$

State transition table:

- rows represent current state
- columns represent possible inputs
- each entry for a particular row and column indicate the new state resulting from that state/input combination

For example, the state transition table for the coin machine is

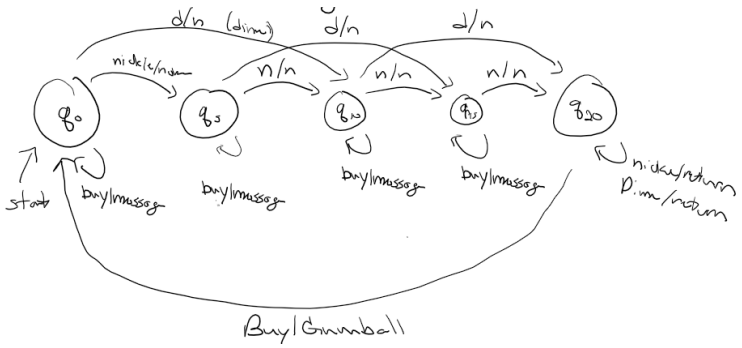
	Coin	Push
Locked	unlocked	locked
Unlocked	unlocked	locked

Components of a Finite State Machine

Notation	Description
Q	finite set of states
$q_0 \in Q$	q_0 is the start state
I	finite set of actions
$\delta : Q \times I \to Q$	transition function

FSM with Output

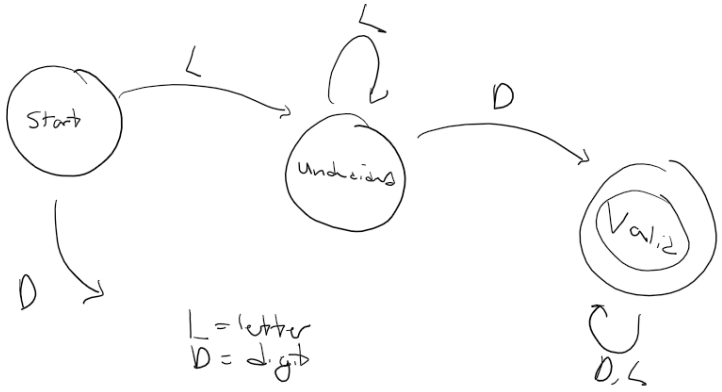
$$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}\}$$
$$I = \{\text{NICKLE}, \text{DIME}, \text{BUY}\}$$
$$O = \{\text{Gumball}, \text{Return}, \text{Message}, \text{None}\}$$



An accepted state is a state that is okay to end in.

$$A \subseteq Q, \text{ Accepted states are a subset of the total states}$$

Example, recognizing valid password



A valid password must begin with a letter and contain at least one digit.

1.5 Turing machines

FSMs are unable to solve even simple computational tasks such as determining whether a binary string has more 0's than 1's.

Church-Turing conjecture

Any problem that can be solved efficiently on any computing device can be solved efficiently by a Turing Machine.

Definition of a Turing Machine

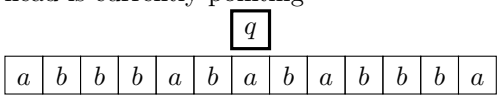
- memory is a 1-dimensional tape.

a	b	b	b	a	b	a	b	a	b	b	b	a

  example tape for $\{a, b, *\}$

- blank symbol (represented by a $*$ symbol)

- a configuration consists of the contents of the tape, the current state, and the tape cell to which the head is currently pointing



- action is determined by a transition function δ

Input to Turing Machine is the Input Alphabet, denoted by Σ, which much be a subset of the tape alphabet Γ

$$\Sigma \subset \Gamma$$

Components of a Turing Machine

Notation	Description
Q	finite set of states
Γ	finite set of tape symbols
$\Sigma \subset \Gamma$	A subset of the tape symbols are input symbols
$q_0 \in Q$	q_0 is the start state
$q_{acc} \in Q$	q_{acc} is the accept state
$q_{rej} \in Q$	q_{rej} is the reject state
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$	Transition Function

Additional Rules

- if Turing machine reaches the accept state from a particular input x, the **Turing machine accepts** x
- if Turing machine reaches the reject state from a particular input x, the **Turing machine accepts** x
- if Turing machine *accepts* or *rejects* x, then the Turing machine **Halts** on x

1.6 Decision problems and languages