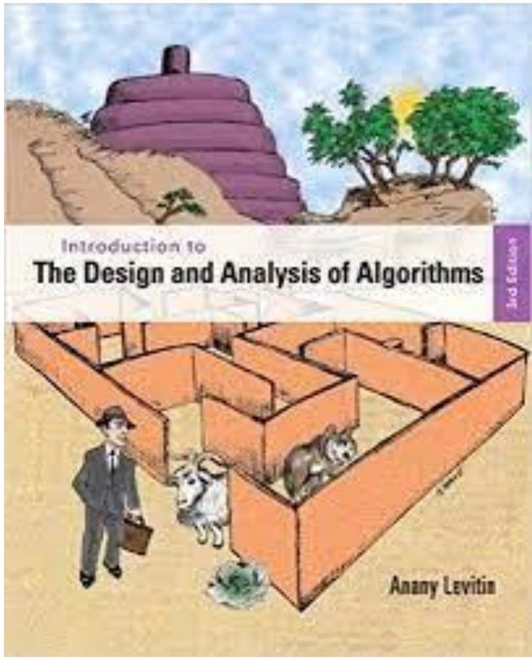
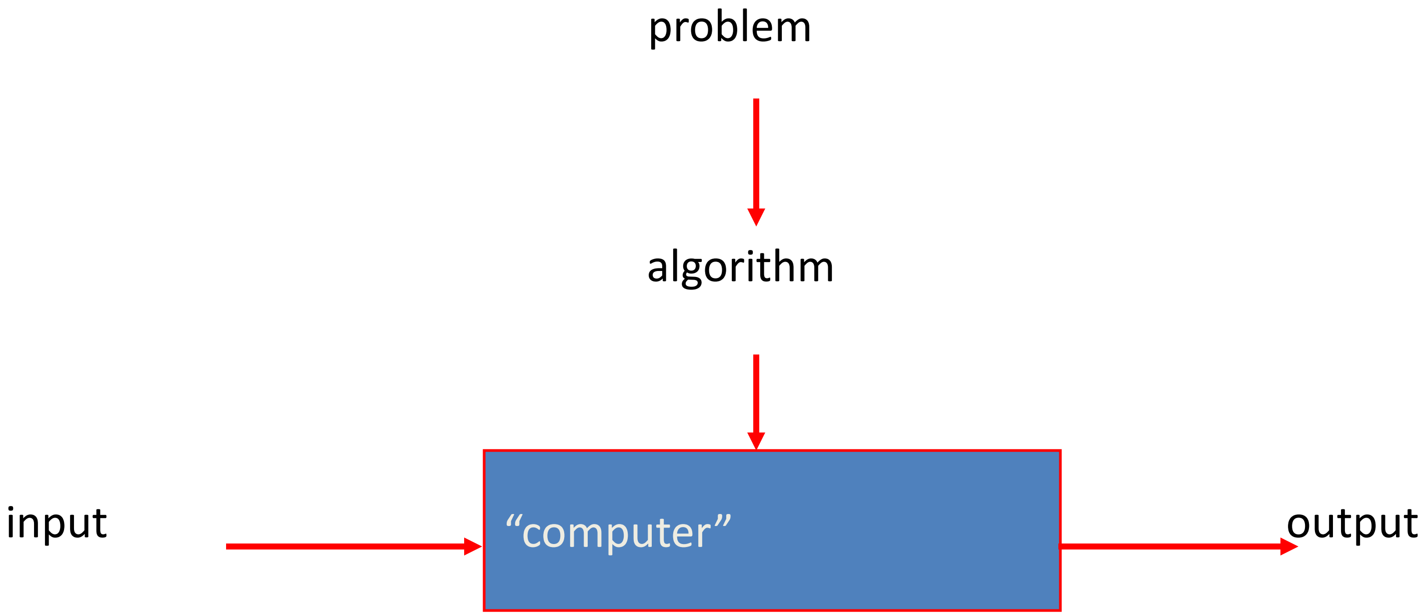
1-Introduction

A. Levitin "Introduction to the Design & Analysis of Algorithms," 3rd ed., Ch. 1



What is an algorithm?

An ***algorithm*** is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Problem: The greatest common divisor of two integers

Find gcd(m,n), the greatest common divisor of two nonnegative, not both zero integers m and n.

Examples:

gcd(60,24) = 12

gcd(60,0) = 60

gcd(25,12) = 1

Problem: The greatest common divisor of two integers

Three methods:

1. Euclid’s algorithm
2. Consecutive integer checking algorithm
3. Middle-school procedure

Problem: The greatest common divisor of two integers

- The nonambiguity requirement for each step of an algorithm cannot be compromised.
- The range of inputs for which an algorithm works has to be specified carefully.
- The same algorithm can be represented in several different ways.
- There may exist several algorithms for solving the same problem.
- Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

Euclid's algorithm

Euclid's algorithm is based on repeated application of equality

$$\gcd(m,n) = \gcd(n, m \bmod n)$$

until the second number becomes 0, which makes the problem trivial.

Example:

gcd(60,24) = gcd(24,12) = gcd(12,0) = 12

Euclid’s algorithm

Step 1 If $n = 0$, return m and stop; otherwise go to Step 2

Step 2 Divide m by n and assign the value fo the remainder to r

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

```
while n ≠ 0 do
        r ← m mod n
   m ← n
   n ← r
return m
```

How do we know that Euclid’s algorithm eventually comes to a stop?

Consecutive integer checking algorithm

Step 1 Assign the value of min{m,n} to t

Step 2 Divide m by t. If the remainder is 0, go to Step 3; otherwise, go to Step 4

Step 3 Divide n by t. If the remainder is 0, return t and stop; otherwise, go to Step 4

Step 4 Decrease t by 1 and go to Step 2

does not work correctly when one of its input numbers is zero.

Middle-school procedure

Step 1 Find the prime factorization of *m*

Step 2 Find the prime factorization of *n*

Step 3 Find all the common prime factors

Step 4 Compute the product of all the common prime factors and return it as gcd*(m,n)*

Is this an algorithm?

Sieve of Eratosthenes

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ to n do $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do

  if $A[p] \neq 0$ //p hasn't been previously eliminated from the list

   $j \leftarrow p * p$

   while $j \leq n$ do

    $A[j] \leftarrow 0$ //mark element as eliminated

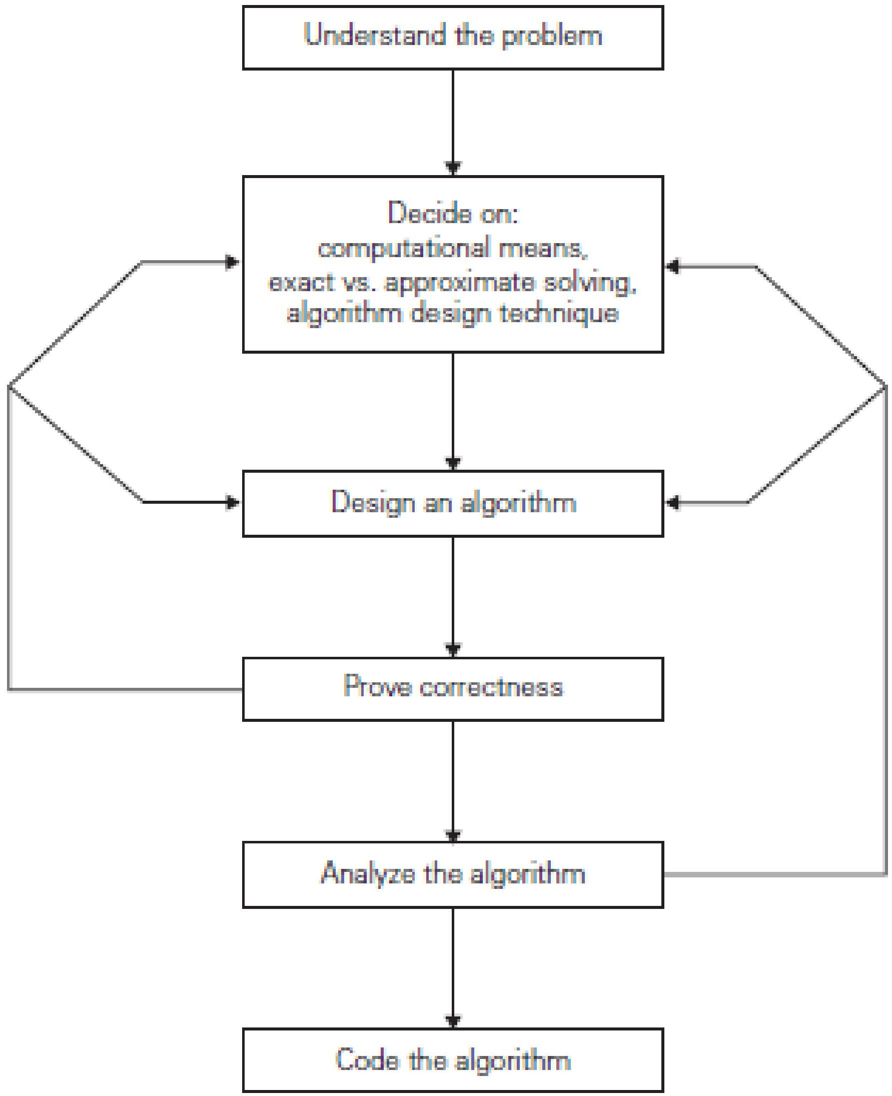
    $j \leftarrow j + p$

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Why study algorithms?

- Theoretical importance
  - the core of computer science
- Practical importance
  - A practitioner’s toolkit of known algorithms
  - Framework for designing and analyzing algorithms for new problems

Algorithm design and analysis process



Two main issues related to algorithms

- How to design algorithms

- How to analyze algorithm efficiency

Algorithm design techniques/strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound

Analysis of algorithms

- How good is the algorithm?
  - time efficiency
  - space efficiency

- Does there exist a better algorithm?
  - lower bounds
  - optimality

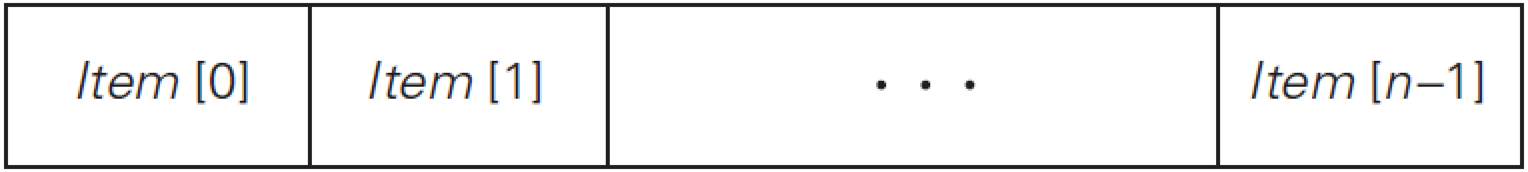
Important problem types

- sorting
- searching
- string processing
- graph problems
- combinatorial problems
- geometric problems
- numerical problems

Fundamental data structures

- List: array, linked list, string
- stack
- queue
- priority queue
- graph
- tree
- set and dictionary

Linear Data Structures



Array of n elements.

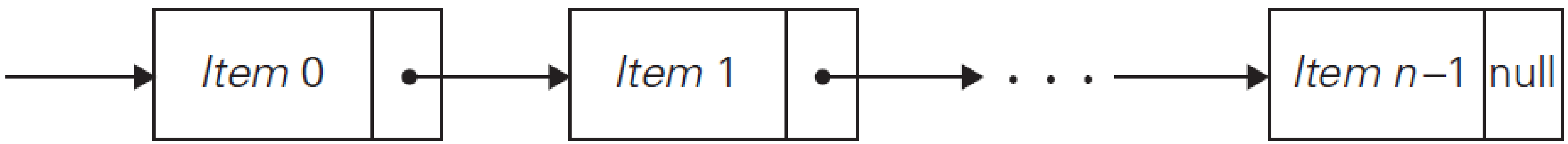


FIGURE 1.4 Singly linked list of n elements.

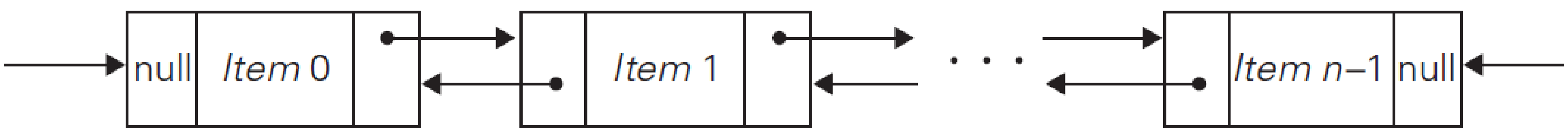
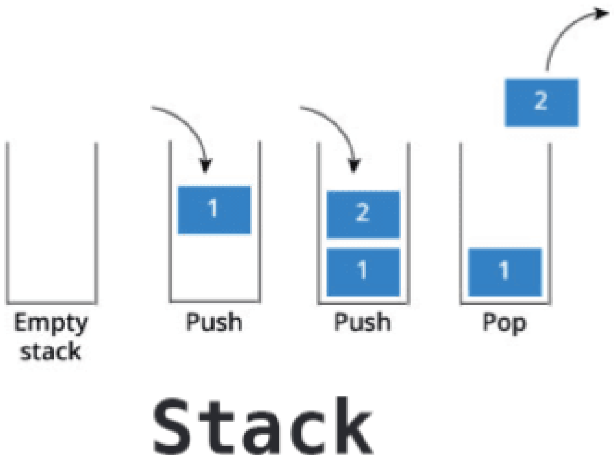
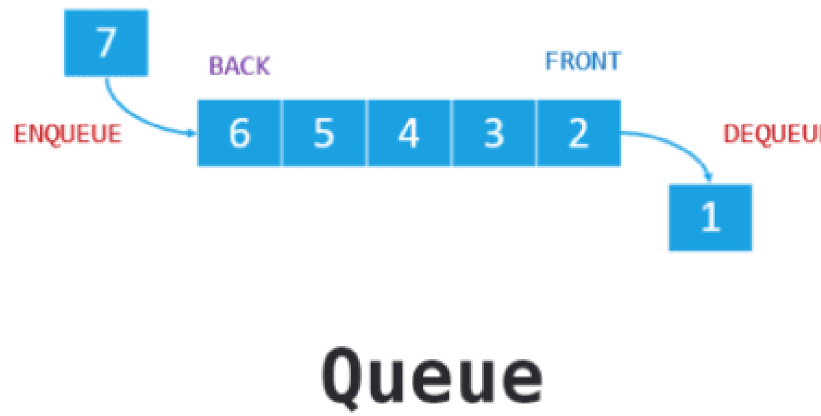


FIGURE 1.5 Doubly linked list of n elements.

Stack and Queue



LIFO

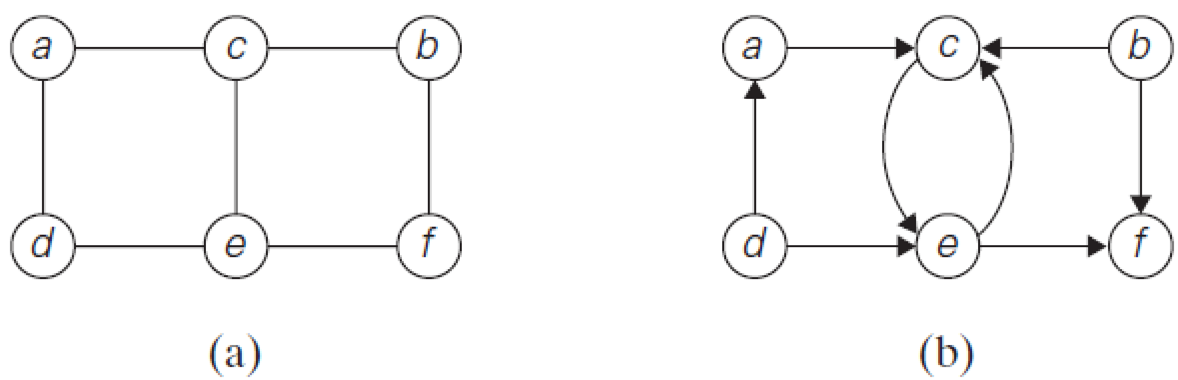


FIFO

https://medium.com/@Adi_Wang1476/stack-and-queue-1823effb6cc

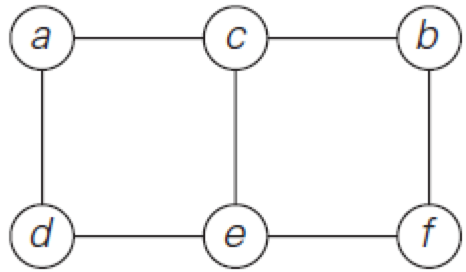
Graph

A ***graph*** $G = (V,E)$ is defined by a pair of two sets: a finite nonempty set V of items called ***vertices*** and a set E of pairs of these items called ***edges***



(a) Undirected graph. (b) Digraph.

Graph Representation



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array}\begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

(a)

$a \rightarrow c \rightarrow d$

$b \rightarrow c \rightarrow f$

$c \rightarrow a \rightarrow b \rightarrow e$

$d \rightarrow a \rightarrow e$

$e \rightarrow c \rightarrow d \rightarrow f$

$f \rightarrow b \rightarrow e$

(b)

(a) Adjacency matrix and (b) adjacency lists of the graph

Graph Representation

Adjacency matrix vs Adjacency lists

- Adjacency matrix: More appropriate for dense graphs
- Adjacency lists: More appropriate for sparse graphs, because less space is used (despite the extra storage consumed by pointers of the linked lists)

Tree

A ***tree*** is a connected acyclic graph