COSC1285/2123: Algorithms & Analysis

Course Overview

Jeffrey Chan

RMIT University
Email : jeffrey.chan@rmit.edu.au

Lecture 1

Outline

1. Preliminaries
2. What is an algorithm and motivation for its study
3. Abstract Data Types
4. Data Structures
5. Summary

Overview

1. Preliminaries
2. What is an algorithm and motivation for its study
3. Abstract Data Types
4. Data Structures
5. Summary

Staff

Lecturer: Jeffrey Chan
Email: jeffrey.chan@rmit.edu.au
Consultation Time: Tuesday 5pm, 14.08.15

Head Tutor: Ayad Turky
Email: ayad.turky@rmit.edu.au
Consultation Time: Wednesday 11am-12pm, 14.08.12

Team of tutors and demonstrators: Ayad, Phuc, Sharlene, Jake, Donald, Robert, Lettesia, Sajal

Course Homepage: Go to Canvas, https://rmit.instructure.com/courses/51759 (COSC1285/2123, 2019 semester 1).

General Course Goals

Key Goal

Study algorithms, data structures and their analysis in order to **find practical solutions to real-world problems**

General Course Goals

Key Goal

Study algorithms, data structures and their analysis in order to **find practical solutions to real-world problems**

Our goals are:

- **Theory :**
  - Learn a variety of algorithms and data structures
  - Understand how to estimate running times of algorithms.

General Course Goals

Key Goal

Study algorithms, data structures and their analysis in order to **find practical solutions to real-world problems**

Our goals are:

- **Theory :**
  - Learn a variety of algorithms and data structures
  - Understand how to estimate running times of algorithms.
- **Practice :**
  - Understand how to measure algorithm complexity and performance empirically.
  - Learn to implement various algorithms and data structures to solve a variety of problem.

Why study Algorithms, Data Structures and their evaluation?

In order to solve problems, including building computer programs to do so, we need to have (algorithmic) tools and evaluate how efficient the tools are.

Why do we need a algorithmic toolset and learn how to evaluate their efficiency?

Why study Algorithms, Data Structures and their evaluation?

In order to solve problems, including building computer programs to do so, we need to have (algorithmic) tools and evaluate how efficient the tools are.

Why do we need a algorithmic toolset and learn how to evaluate their efficiency?

Answer:

- Need to know how the algorithms work and how to measure their efficiency in order to select the right one for the right problem/job.

Why study Algorithms, Data Structures and their evaluation?

In order to solve problems, including building computer programs to do so, we need to have (algorithmic) tools and evaluate how efficient the tools are.

Why do we need a algorithmic toolset and learn how to evaluate their efficiency?

Answer:

- Need to know how the algorithms work and how to measure their efficiency in order to select the right one for the right problem/job.
- Designing new algorithms or modify exiting ones to solve your problems.

Why study Algorithms, Data Structures and their evaluation?

In order to solve problems, including building computer programs to do so, we need to have (algorithmic) tools and evaluate how efficient the tools are.

Why do we need a algorithmic toolset and learn how to evaluate their efficiency?

Answer:

- Need to know how the algorithms work and how to measure their efficiency in order to select the right one for the right problem/job.
- Designing new algorithms or modify exiting ones to solve your problems.
- **It makes you a better problem solver and programmer!**

Prequisites

- This course uses **Java**.
- Prequisites:
  - *COSC2401 Software Architecture: Design and Implementation* or *COSC1295 Advanced Programming* or *COSC1076 Advanced Programming Techniques*, which have either have pre-requisites courses that teaches Java or teach Java and concepts helpful for this course.

Prequisites

- This course uses **Java**.
- Prequisites:
  - *COSC2401 Software Architecture: Design and Implementation* or *COSC1295 Advanced Programming* or *COSC1076 Advanced Programming Techniques*, which have either have pre-requisites courses that teaches Java or teach Java and concepts helpful for this course.
- This course is **not** about programming, but we do need a language for the practical parts:
  - Java: At a minimum, you should be comfortable with *basic data types and structures, defining new classes and data types, inheritance and polymorphism*.

Prequisites

- This course uses **Java**.
- Prequisites:
  - *COSC2401 Software Architecture: Design and Implementation* or *COSC1295 Advanced Programming* or *COSC1076 Advanced Programming Techniques*, which have either have pre-requisites courses that teaches Java or teach Java and concepts helpful for this course.
- This course is **not** about programming, but we do need a language for the practical parts:
  - Java: At a minimum, you should be comfortable with *basic data types and structures, defining new classes and data types, inheritance and polymorphism*.
  - Warning: The assignments and labs are all in Java, so you will struggle if you are not comfortable with Java.

Prequisites

- This course uses **Java**.
- Prequisites:
  - *COSC2401 Software Architecture: Design and Implementation* or *COSC1295 Advanced Programming* or *COSC1076 Advanced Programming Techniques*, which have either have pre-requisites courses that teaches Java or teach Java and concepts helpful for this course.
- This course is **not** about programming, but we do need a language for the practical parts:
  - Java: At a minimum, you should be comfortable with *basic data types and structures, defining new classes and data types, inheritance and polymorphism*.
  - Warning: The assignments and labs are all in Java, so you will struggle if you are not comfortable with Java.

Maths :)

This course has some **maths** in it.

Maths :)

This course has some **maths** in it.

We need to count how many times certain operations occur to analysis algorithms.

- Equations (summations, recursions). $\sum_{i=0}^{n} 2$.
- Arithmetic and partial sum simplifications. $\sum_{i=0}^{n} 2 = 2n + 2$.

I understand some of you haven't see maths for years (or run for nearest train station when it appears), but help is available.

Maths :)

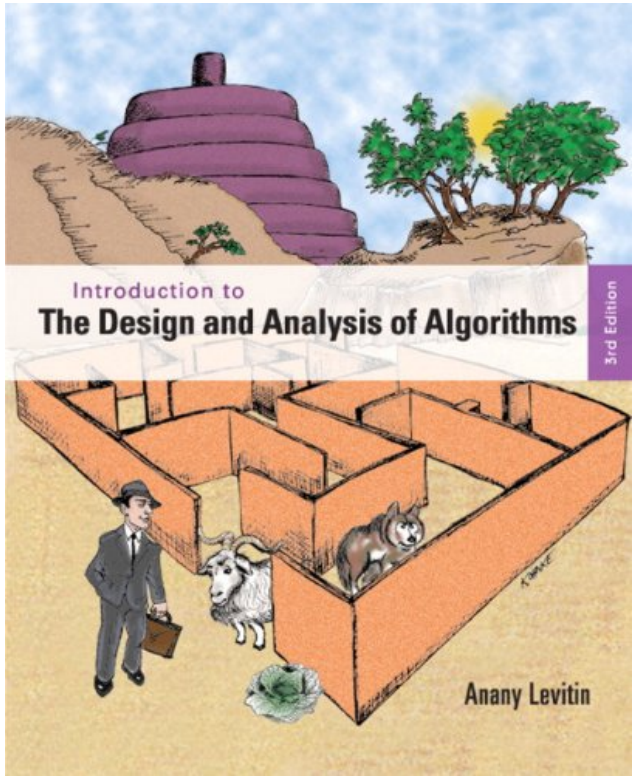
This course has some **maths** in it.

We need to count how many times certain operations occur to analysis algorithms.

- Equations (summations, recursions). $\sum_{i=0}^{n} 2$.
- Arithmetic and partial sum simplifications. $\sum_{i=0}^{n} 2 = 2n + 2$.

I understand some of you haven't see maths for years (or run for nearest train station when it appears), but help is available.

Official Textbook



A. Levitin. *Introduction to the design and analysis of algorithms*. Addison Wesley, Boston, third edition, 2012, ISBN-13: 9780132316811.

Other Reference Books


T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, third edition, 2008.


J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson, Boston, 2006.


M. A. Weiss. *Data structures and algorithm analysis in C*. Addison Wesley, New York, second edition, 1996.


B. Kernighan and D. Ritchie. *C programming language*. Prentice Hall, New York, second edition, 1988.

Course Syllabus

Lecture	Topic	Reading
1	Problem Types & Basics	Levitin – Ch 1
2	Algorithmic Analysis	Levitin – Ch 2
3	Brute Force	Levitin – Ch 3
4	Decrease & Conquer	Levitin – Ch 4
5	Divide & Conquer	Levitin – Ch 5
6	Transform & Conquer	Levitin – Ch 6
7	**In-class Mid-semester Test**	
8	Time & Space Tradeoffs	Levitin – Ch 7
9	Dynamic Programming	Levitin – Ch 8
10	Greedy Techniques	Levitin – Ch 9
11	Incremental Improvement	Levitin – Ch 10
12	Case Study & Course Review	

Assessment

50% of your assessment will come from a closed book two-hour examination.

5% of your assessment will come from weekly online quizzes that you can attempt on Canvas.

15% of your assessment will come from the closed book Week 7 Mid-semester test. It will be held during the lecture time of week 7.

The remaining 30% is divided equally among two assignments.

Weekly Quizzes

- Every week, there will be a 5 question quiz.

Weekly Quizzes

- Every week, there will be a 5 question quiz.
- A total of 12 quizzes, one per week, will be available for you to complete through Canvas.

Weekly Quizzes

- Every week, there will be a 5 question quiz.
- A total of 12 quizzes, one per week, will be available for you to complete through Canvas.
- Each quiz contributes up to 0.5% of your final course mark. If you complete more than 10 quizzes, we take your 10 best results to compute the quiz mark component

Weekly Quizzes

- Every week, there will be a 5 question quiz.
- A total of 12 quizzes, one per week, will be available for you to complete through Canvas.
- Each quiz contributes up to 0.5% of your final course mark. If you complete more than 10 quizzes, we take your 10 best results to compute the quiz mark component .
- For each quiz, if you score at least 4 out of 5, you will get will get the 0.5% mark, if you score 3 out of 5, you will get 0.25% marks, and if you score less than 3, you will get 0 marks for that quiz.

Weekly Quizzes

- Every week, there will be a 5 question quiz.
- A total of 12 quizzes, one per week, will be available for you to complete through Canvas.
- Each quiz contributes up to 0.5% of your final course mark. If you complete more than 10 quizzes, we take your 10 best results to compute the quiz mark component .
- For each quiz, if you score at least 4 out of 5, you will get will get the 0.5% mark, if you score 3 out of 5, you will get 0.25% marks, and if you score less than 3, you will get 0 marks for that quiz.
- Each quiz must be completed before 11:59:59PM on the Sunday that is 1 week after the lecture (e.g, for week 1, this will be the Sunday of week 2).

Science Professional Micro Creds

- 4 modules, EQ, Cultural Awareness, Information Literarcy, Academic Integrity

Science Professional Micro Creds

- 4 modules, EQ, Cultural Awareness, Information Literarcy, Academic Integrity
- This semester, optional, but we strongly encourage you to do them.
- You will get a credential got each of them.
- See Canvas.

Examinable Material

Note

Everything we cover in the lectures, tutorials and labs is examinable unless explicitly stated otherwise.

Examinable Material

Note

Everything we cover in the lectures, tutorials and labs is examinable unless explicitly stated otherwise.

- Tutorials are an integral part of the course, it is important to try the questions and problems before seeing the solutions.
  - It might be easy to follow a solution, but when asked to solve the problem, it is a completely different proposition.
  - PLEASE ATTEND!

Examinable Material

Note

Everything we cover in the lectures, tutorials and labs is examinable unless explicitly stated otherwise.

- Tutorials are an integral part of the course, it is important to try the questions and problems before seeing the solutions.
  - It might be easy to follow a solution, but when asked to solve the problem, it is a completely different proposition.
  - PLEASE ATTEND!
- Laboratories consolidate your understanding, and helps you translate your knowledge of the data structures and algorithms into running code.
  - Google and other high-tech companies typically have on-the-spot coding interviews.
  - Need them to practice before assignments.

Academic Misconduct

- Please read the **University Plagiarism Statement** in the course guide very carefully.
- In short, cheating, whether by fabrication, falsification of data, or representing the work of someone else as your own is an offense subject to University disciplinary procedures.
- Plagiarism may result in charges of academic misconduct which carry a range of penalties including cancellation of results and exclusion from the course.
- Exact penalties are decided in formal plagiarism hearings.
- **All assignment** are to be done in groups of 2 (pairs), and **weekly quizzes, mid-semester and exam** must be done individually.

Sources of help (We are here to help!)

- *See me* if you are having difficulties which may prevent you from completing the course as soon as possible.

Sources of help (We are here to help!)

- *See me* if you are having difficulties which may prevent you from completing the course as soon as possible.
- Ask tutors and lab demonstrators!
- 2 weekly consultations times / week, one with myself and one with Ayad (head tutor), see `https://rmit.instructure.com/courses/51759/pages/teaching-team-2?module_item_id=1557175` for times.

Sources of help (We are here to help!)

- *See me* if you are having difficulties which may prevent you from completing the course as soon as possible.
- Ask tutors and lab demonstrators!
- 2 weekly consultations times / week, one with myself and one with Ayad (head tutor), see `https://rmit.instructure.com/courses/51759/pages/teaching-team-2?module_item_id=1557175` for times.
- Discussion forum and each other!

Sources of help (We are here to help!)

- *See me* if you are having difficulties which may prevent you from completing the course as soon as possible.
- Ask tutors and lab demonstrators!
- 2 weekly consultations times / week, one with myself and one with Ayad (head tutor), see `https://rmit.instructure.com/courses/51759/pages/teaching-team-2?module_item_id=1557175` for times.
- Discussion forum and each other!
- Extensions:
  - Where possible, let us know if you may be delayed in submitting an assignment or other assessable material. Please do **not** wait until the day before it is due.
  - Extensions are only granted in exceptional circumstances and require the *official process of "Special Consideration"* to be followed.

Answering Rumours

"I heard the course is really hard."

Answering Rumours

"I heard the course is really hard."

Jeff's answer: This course can be different to what you done before, but it is not as difficult as what you may have heard.

Answering Rumours

"I heard the course is really hard."

Jeff's answer: This course can be different to what you done before, but it is not as difficult as what you may have heard.

Advice:

- From previous semesters, those that studied throughout the semester, did **well**.
- Those that attended and engaged in tutorials and labs, did **well**.
- Those that crammed for exams, not **so well**.

Answering Rumours

"I heard the course is really hard."

Jeff's answer: This course can be different to what you done before, but it is not as difficult as what you may have heard.

Advice:

- From previous semesters, those that studied throughout the semester, did **well**.
- Those that attended and engaged in tutorials and labs, did **well**.
- Those that crammed for exams, not **so well**.
- Do *something* for the course everyday : read, code, work a problem . . . this is **not** a course you can **cram**.

Answering Rumours

"I heard the course is really hard."

Jeff's answer: This course can be different to what you done before, but it is not as difficult as what you may have heard.

Advice:

- From previous semesters, those that studied throughout the semester, did **well**.
- Those that attended and engaged in tutorials and labs, did **well**.
- Those that crammed for exams, not **so well**.
- Do *something* for the course everyday : read, code, work a problem . . . this is **not** a course you can **cram**.

A journey of a thousand miles begins with a single step

Week 1 Overview

Levitin – The design and analysis of algorithms

This week we will be covering the material from Chapter 1.

Learning outcomes:

- Understand the concept of algorithms and data structures and the motivation behind their analysis.
- Learn about different abstract data types and data structures (expanded upon in subsequent classes).

Overview

1. Preliminaries
2. What is an algorithm and motivation for its study
3. Abstract Data Types
4. Data Structures
5. Summary

What is an algorithm?

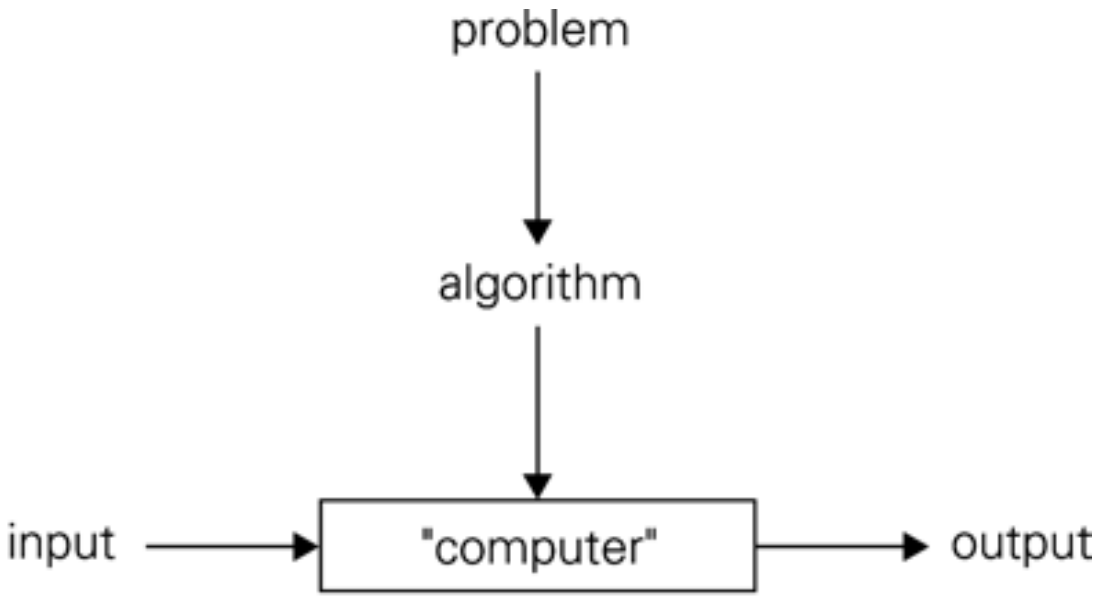
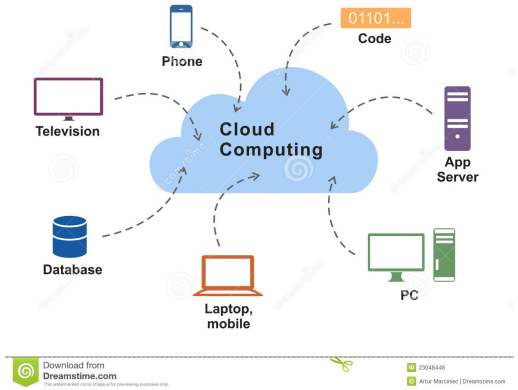


FIGURE 1.1 Notion of algorithm

An algorithm is a sequence of (unambiguous) instructions/steps for solving a problem.

Examples of Algorithms ... ?



Example: Euclid's Algorithm

Problem : Find $\textbf{GCD}(m, n)$, the greatest common divisor of two non-negative integers m and n.

Examples: $\textbf{GCD}(60, 24) = 12$; $\textbf{GCD}(60, 0) = 60$; $\textbf{GCD}(0, 0) = ?$

Example: Euclid's Algorithm

Problem : Find $\mathbf{GCD}(m, n)$, the greatest common divisor of two non-negative integers m and n.

Examples: $\mathbf{GCD}(60, 24) = 12$; $\mathbf{GCD}(60, 0) = 60$; $\mathbf{GCD}(0, 0) = ?$

Solution (*sketch*)

Euclid's algorithm is based on repeated application of the equality $\mathbf{GCD}(m, n) = \mathbf{GCD}(n, m \bmod n)$ until the second number (n) becomes 0. The answer is then the first number (m).

Example : $\mathbf{GCD}(60, 24) = \mathbf{GCD}(24, 12) = \mathbf{GCD}(12, 0) = 12$

Euclid's Algorithm Pseudocode

GCD

ALGORITHM **Euclid** (m, n)
// Computes **GCD**(m, n) by Euclid's algorithm
// INPUT : Two non-negative integers m and n
// OUTPUT : Greatest Common Divisor of m and n

1: **if** $n > 0$ **then**
2: return **GCD**$(n, m \mod n)$
3: **end if**
4: **return** m

Euclid's Algorithm

Alternative approaches to this problem?

Is it more "efficient"?

GCD Alternative (Sketch)

Alternative - Common primes:

Step 1 Find the prime factors of m.
Step 2 Find the prime factors of n.
Step 3 Identify all the common factors in the two prime expansions from Step 1 and Step 2.
Step 4 Compute the product of all common factors and return it as the greatest common divisor.

Example:
$60 = 2 \cdot 2 \cdot 3 \cdot 5$
$24 = 2 \cdot 2 \cdot 2 \cdot 3$
$\textbf{GCD}(60, 24) = 2 \cdot 2 \cdot 3 = 12$

Motivation for studying algorithms

▸ Video

Overview

1. Preliminaries
2. What is an algorithm and motivation for its study
3. Abstract Data Types
4. Data Structures
5. Summary

Abstract Data Types and Data structures

Algorithms process input data, might need to process some immediate data form, then output some results, which could be stored as data.

Abstract Data Types and Data structures

Algorithms process input data, might need to process some immediate data form, then output some results, which could be stored as data.

But we do not want to consider algorithms for each type of input data.

Instead we seek to talk about the data input, immediate storage and output in terms of general characteristics and operations the data should possess.

These data abstractons are called Abstract data types (ADT).

Abstract Data Types and Data structures

Algorithms process input data, might need to process some immediate data form, then output some results, which could be stored as data.

But we do not want to consider algorithms for each type of input data.

Instead we seek to talk about the data input, immediate storage and output in terms of general characteristics and operations the data should possess.

These data abstractons are called Abstract data types (ADT).

- An ADT can be defined by the collection of common operations that are accessed through an *interface* and its characteristics.

Abstract Data Type

Data structure is the implementation of an abstract data type.

Abstract Data Type

Data structure is the implementation of an abstract data type.

Example:

- Integer (-4,-1, 0, 29) is an abstract data type. Its operations include add, subtract, multiply etc.
- Bit vectors is a data structure. It is an implementation of an integer.
- Hexadecimal vectors is a data structure. It is an implementation of an integer.

Abstract Data Type

Data structure is the implementation of an abstract data type.

Example:

- Integer (-4,-1, 0, 29) is an abstract data type. Its operations include add, subtract, multiply etc.
- Bit vectors is a data structure. It is an implementation of an integer.
- Hexadecimal vectors is a data structure. It is an implementation of an integer.

Motivation:

- We can specify the input and data forms of our algorithms in terms of ADT, and not worry about the actual implementation until we need to implement it.
- We can change data structure implementations to cater for the problem at hand.

Fundamental ADTs

- set
- sequences
- dictionary/map
- stack
- queue
- priority queue
- graph
- tree

Sets

Sets

A *set* is a collection of distinguishable objects, often called *members* or *elements*.

- Sets can be finite or infinite, and do not impose any ordering on the members.
- Typical operations: add, remove, search
- Each element may only appear in the set once. If elements may appear multiple times, the resulting collection is referred to as a bag or multiset.

Sets

Example (Sets)

Simple examples of sets include:

(1) binary : $S_1 = \{0, 1\}$,

(2) character : $S_2 = \{c, a, y, s, t\}$,

(3) word : $S_3 = \{$ car, house, man, sat, truck $\}$,

(4) delimiter : $S_4 = \{\{!\}, \{;\}, \{?\}, \{.\}\}$,

(5) variable, prefix-free : $S_5 = \{0, 10, 110, 1110\}$.

Sequences

Sequences

A *sequence* is a collection of elements in which the order of the elements must be maintained, and elements can occur any number of times.

- A sequence is simply a multiset in which the order is important.
- Typical operations: add, remove, search
- In computer science, can also be referred to as a list (ordered collection of elements).

Sequences

Example (Sequences)

Examples of sequences include:

(1) Binary : $\mathcal{T}_1 =$ "101010110001",

(2) English : $\mathcal{T}_2 =$ "The red car belongs to me.",

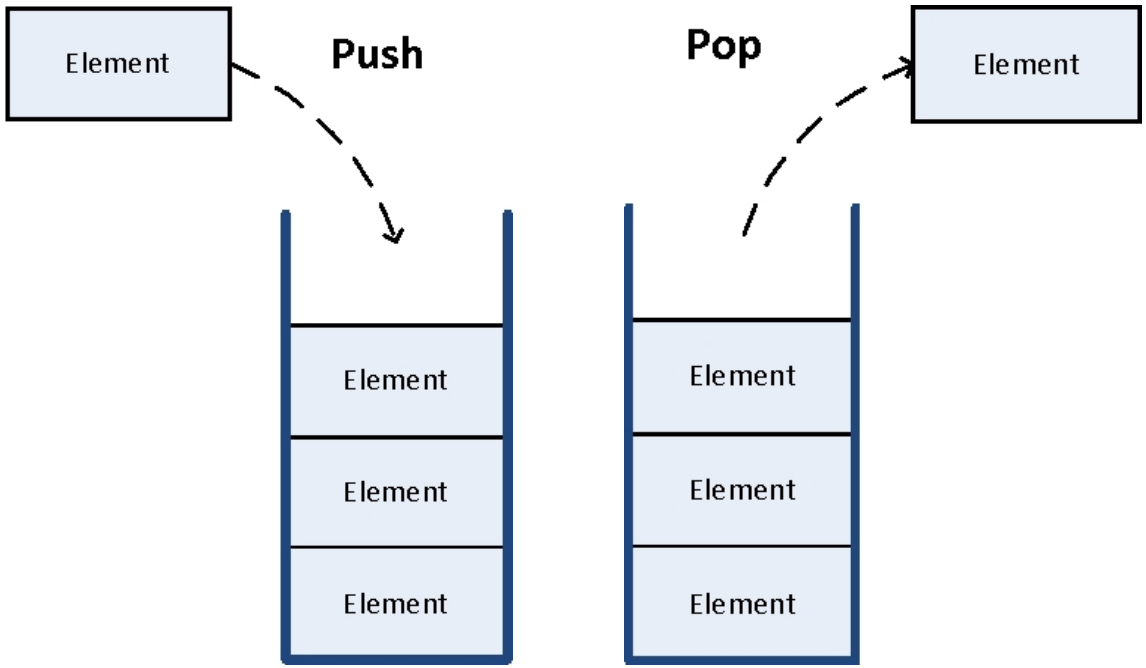
(3) Genomic : $\mathcal{T}_3 =$ "gattcaggaatccgccggtaacgcgcatataattt".

Stack

Stack

A *stack* is a collection with two defining operations, *push* and *pop*. Push adds new element at top of stack, pop removes element at top from stack.

Stack ADT implements LIFO principle (last in, first out).



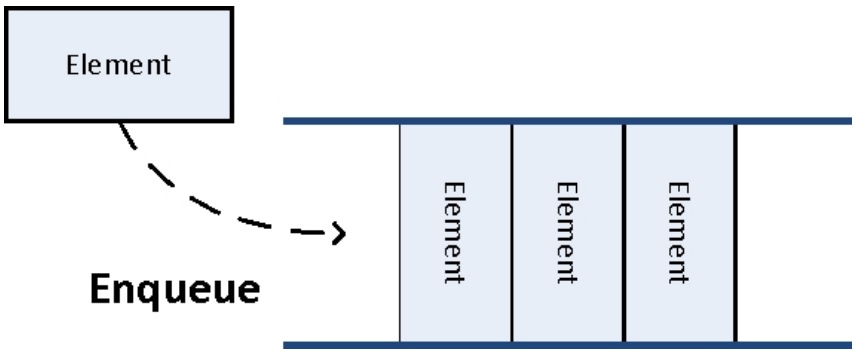
(a) push (b) pop

Queue

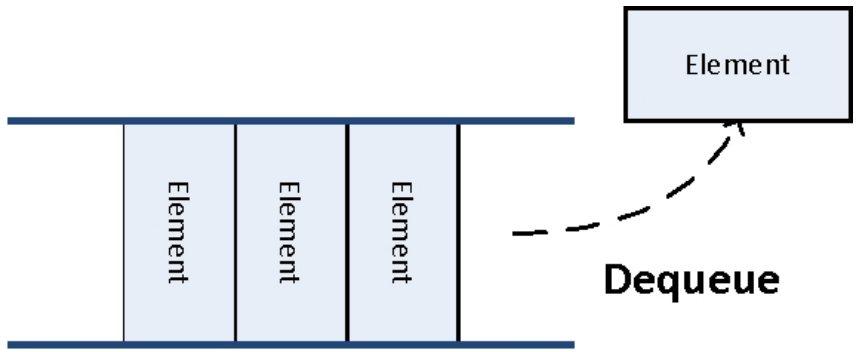
Queue

A *queue* is a collection with two defining operations, *enqueue* and *dequeue*. Enqueue adds new element at back of queue, dequeue removes element at front from queue.

Queue ADT implements FIFO principle (first in, first out).



(c) enqueue



(d) dequeue

Priority Queue

Queue

A priority queue is a queue that has a priority associated with each element it holds. Enqueue is the same as a (non-priority) queue, but for dequeue, the element with the highest priority is removed first.

Priority Queue

Queue

A priority queue is a queue that has a priority associated with each element it holds. Enqueue is the same as a (non-priority) queue, but for dequeue, the element with the highest priority is removed first.

Example: Queue for boarding planes, passengers who are members of the airlines or elderly have priority board plane (dequeue).

Dictionaries/Maps

Dictionary/Maps

A *dictionary/map* is a collection of *(key,value)* pairs, such that each key can only appear once in the dictionary/map.

Dictionaries/Maps

Dictionary/Maps

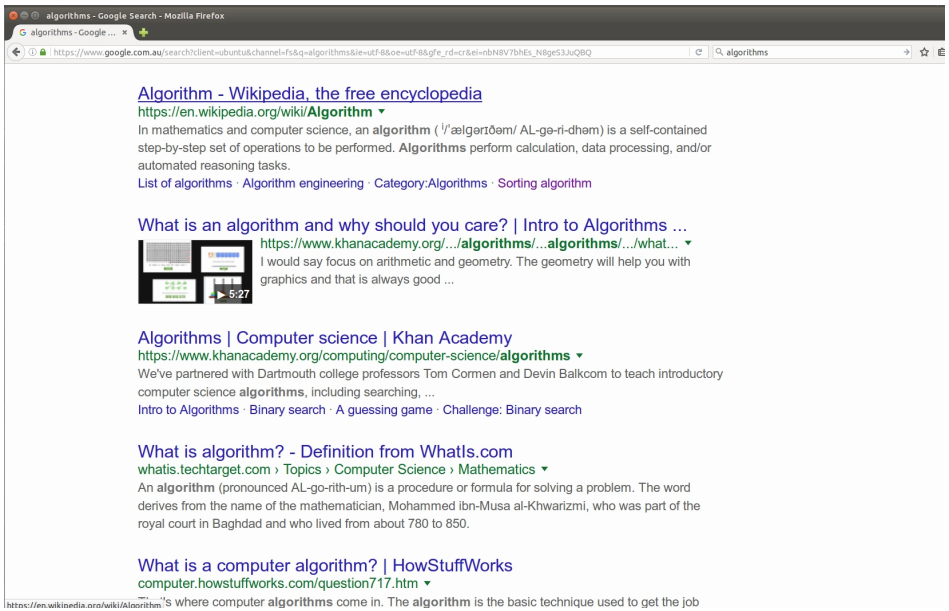
A *dictionary/map* is a collection of *(key,value)* pairs, such that each key can only appear once in the dictionary/map.

Example (Dictionary/map)

Example of a dictionary of former leaders and country they led:

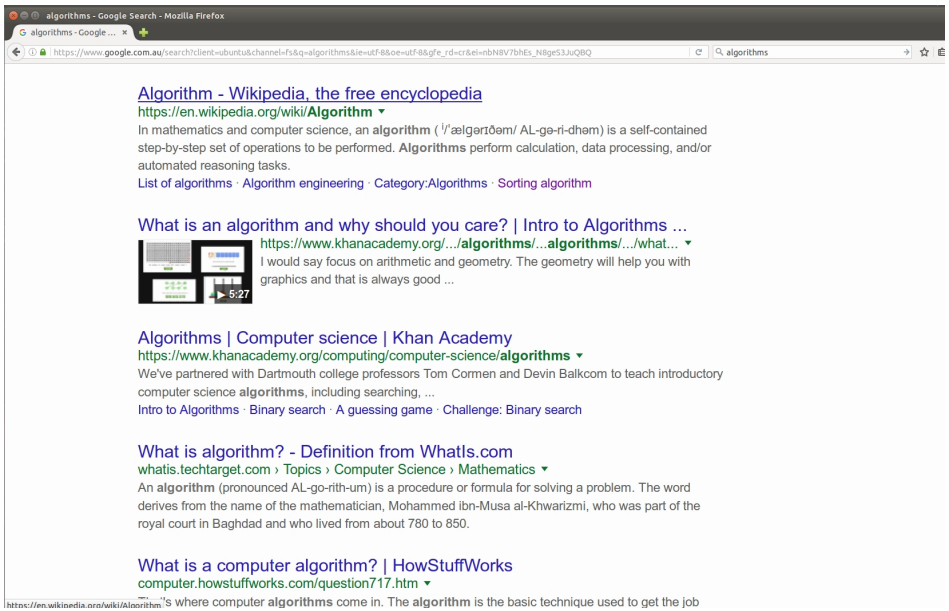
key	**value**
Vladimir Lenin	Soviet Union
Winston Churchill	UK
Mao Ze Dong	China
Tony Abbott	Australia

2 min Break!



Trivia: Who invented Pagerank and which company was founded based on it?

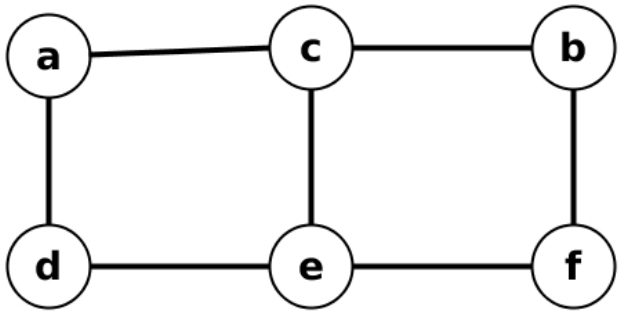
2 min Break!



Trivia: Who invented Pagerank and which company was founded based on it?

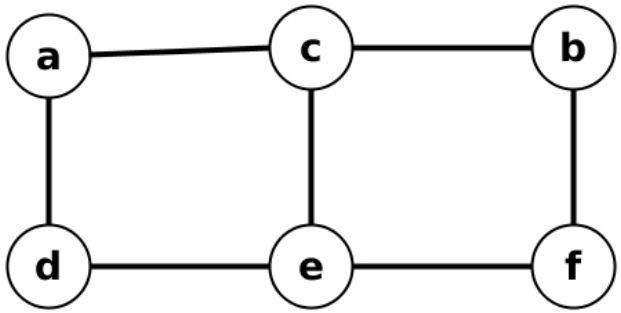
Answer: Larry Page and Sergey Brin, Google.

Graphs



A graph $G = \langle V, E \rangle$ is defined by a pair of two sets: a finite set V of items called vertices and a set E called edges, representing links/relations/connections between pairs of vertices.

Graphs



A graph $G = \langle V, E \rangle$ is defined by a pair of two sets: a finite set V of items called vertices and a set E called edges, representing links/relations/connections between pairs of vertices.

Example:
$V = \{a, b, c, d, e, f\}$
$E = \{(a, d), (a, c), (d, e), (c, e), (c, b), (e, f), (b, f)\}$

Graphs

- A graph G is undirected if the edges do not have a direction (i.e., all of the pairs of vertices in E are unordered).
- A graph G is directed if the edges from a direction (i.e., all of the pairs of vertices in E have an ordering imposed).

Graphs

- A graph G is undirected if the edges do not have a direction (i.e., all of the pairs of vertices in E are unordered).
- A graph G is directed if the edges from a direction (i.e., all of the pairs of vertices in E have an ordering imposed).

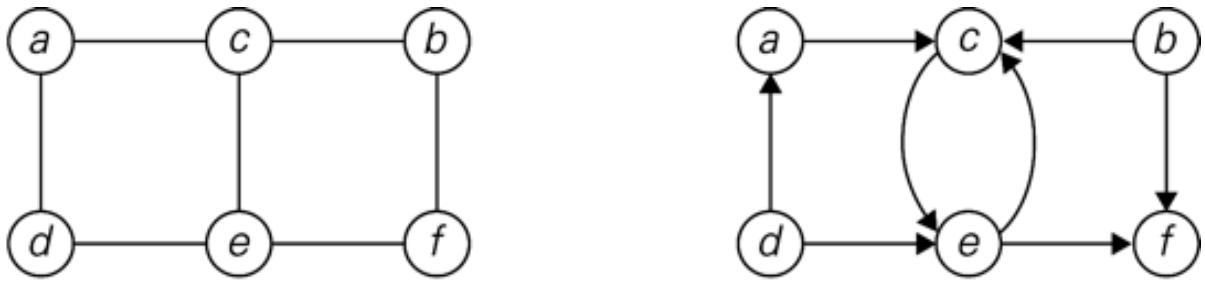
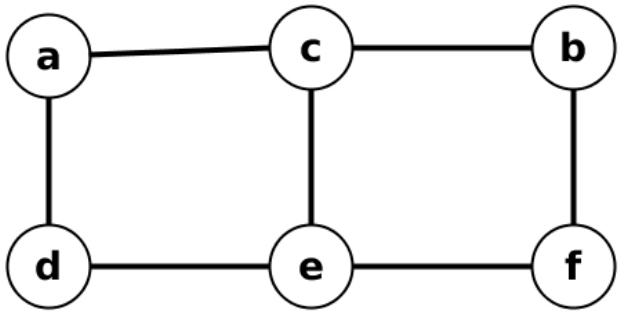


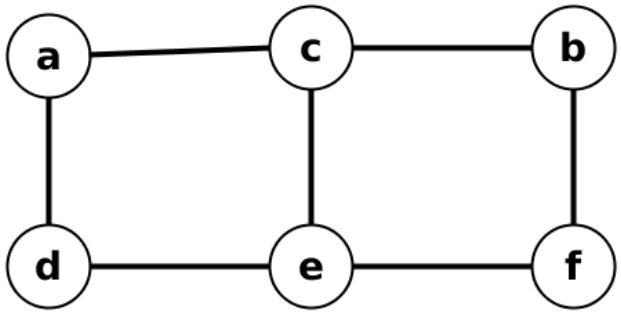
FIGURE 1.6 (a) Undirected graph. (b) Digraph.

Graph Representations



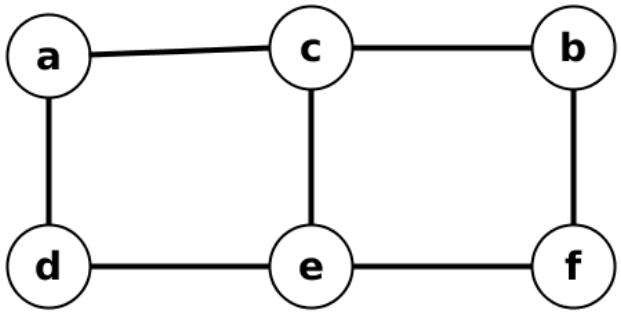
How to represent a graph without drawing it?

Graph Representations



How to represent a graph without drawing it? Vertex and edge lists (previously)? Other possibilities?

Graph Representations



How to represent a graph without drawing it? Vertex and edge lists (previously)? Other possibilities?

	a	b	c	d	e	f
a	0	0	1	1	0	0
b	0	0	1	0	0	1
c	1	1	0	0	1	0
d	1	0	0	0	1	0
e	0	0	1	1	0	1
f	0	1	0	0	1	0

(a)

a	→	c	→	d		
b	→	c	→	f		
c	→	a	→	b	→	e
d	→	a	→	e		
e	→	c	→	d	→	f
f	→	b	→	e		

(b)

FIGURE 1.7 (a) Adjacency matrix and (b) adjacency lists of the graph in Figure 1.6a

Weighted Graphs

What if edges have weights associated with them?

Weighted Graphs

What if edges have weights associated with them?

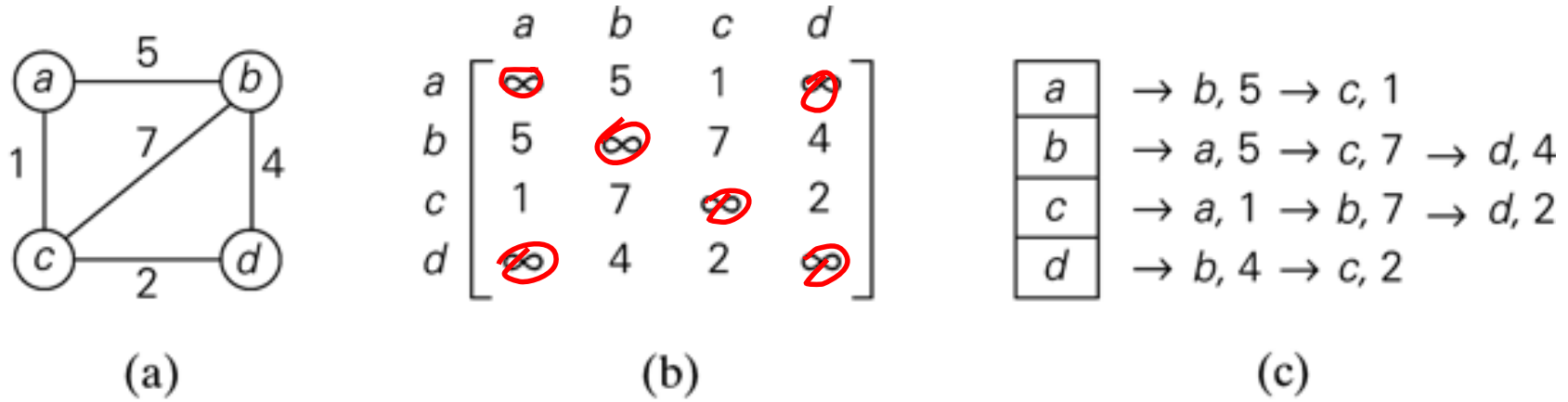
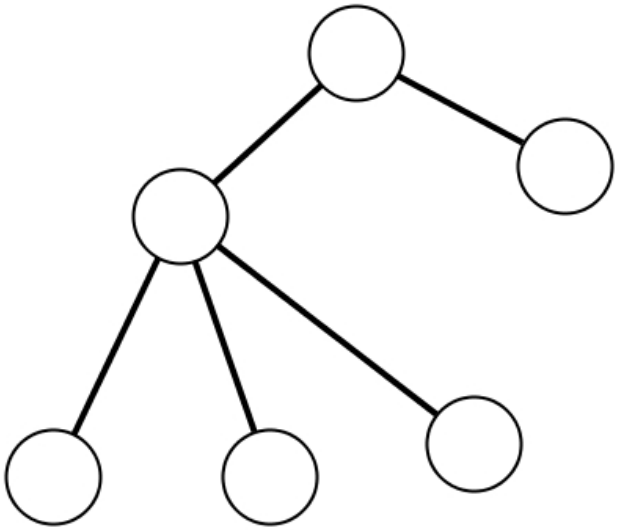


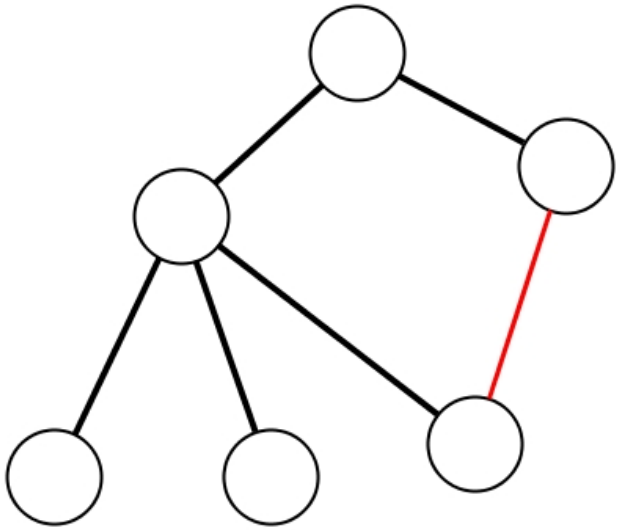
FIGURE 1.8 (a) Weighted graph. (b) Its weight matrix. (c) Its adjacency lists.

Trees

A tree is a connected acyclic graph.



(e) tree



(f) Not a tree (but a graph)

Overview

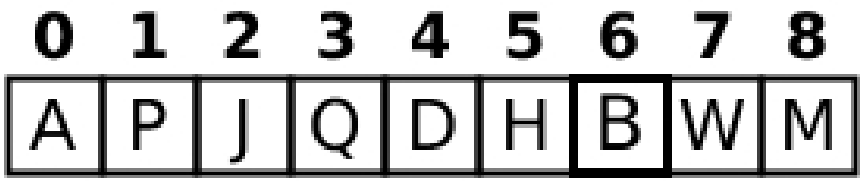
1. Preliminaries
2. What is an algorithm and motivation for its study
3. Abstract Data Types
4. Data Structures
5. Summary

Data Structures

Now we describe several data structures that can implement different abstract data types.
We will learn more data structures as we progress through the course.

Two important data structures are array and linked-list.
Both can implement many ADT, e.g., sequences, set, dictionary.

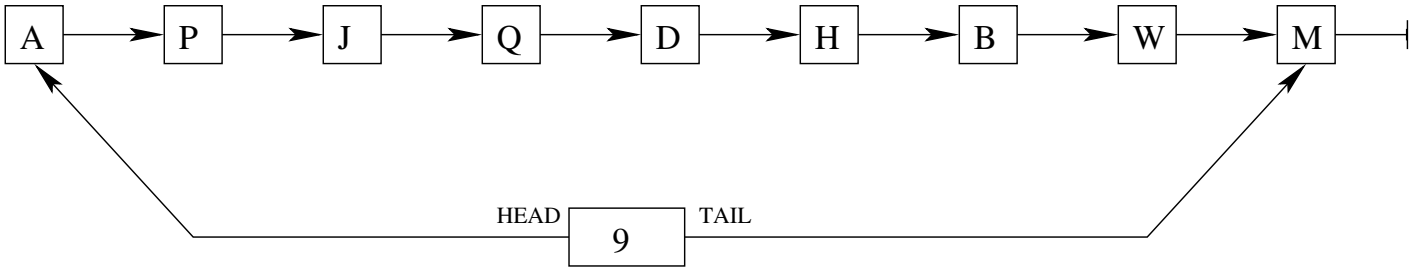
Array



Characteristics:

- Collection of elements usually stored in a continuous manner, accessed by indices.
- Fast random access.
- Resizing can be costly (may require copying), hence often need to estimate size of array beforehand.

Linked Lists



- Each node stores element and a pointer to next node.
- There is a head pointer that points to the start of the list.
- Somtimes there is a tail pointer that points to the end of the list.
- Other variants of linked list (e.g., doubly linked lists, go to labs to see!)

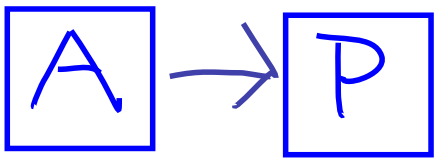
Linked Lists - insertion

Insert the elements of following sequence of letters, one by one [A,P,J,Q,D,H,B,W,M]:

1

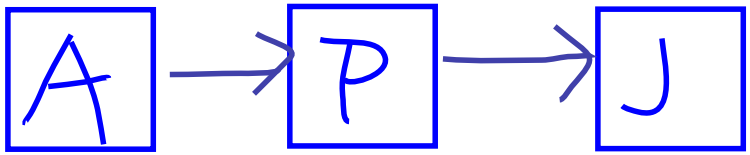
create node for A

2



create node + add link

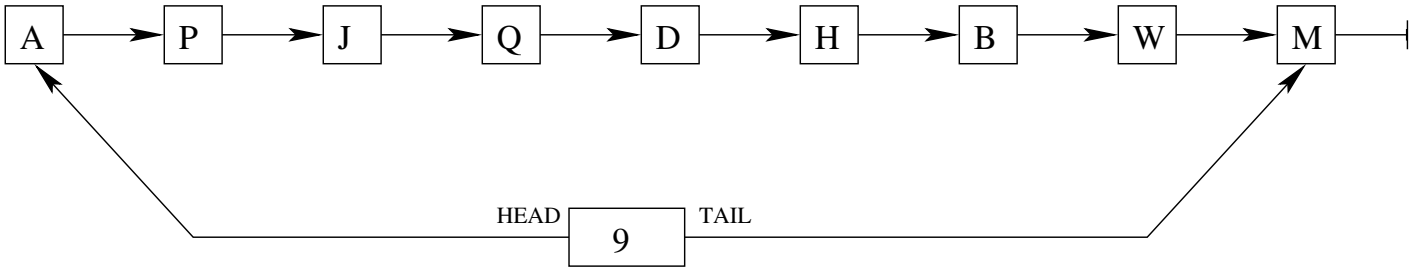
3



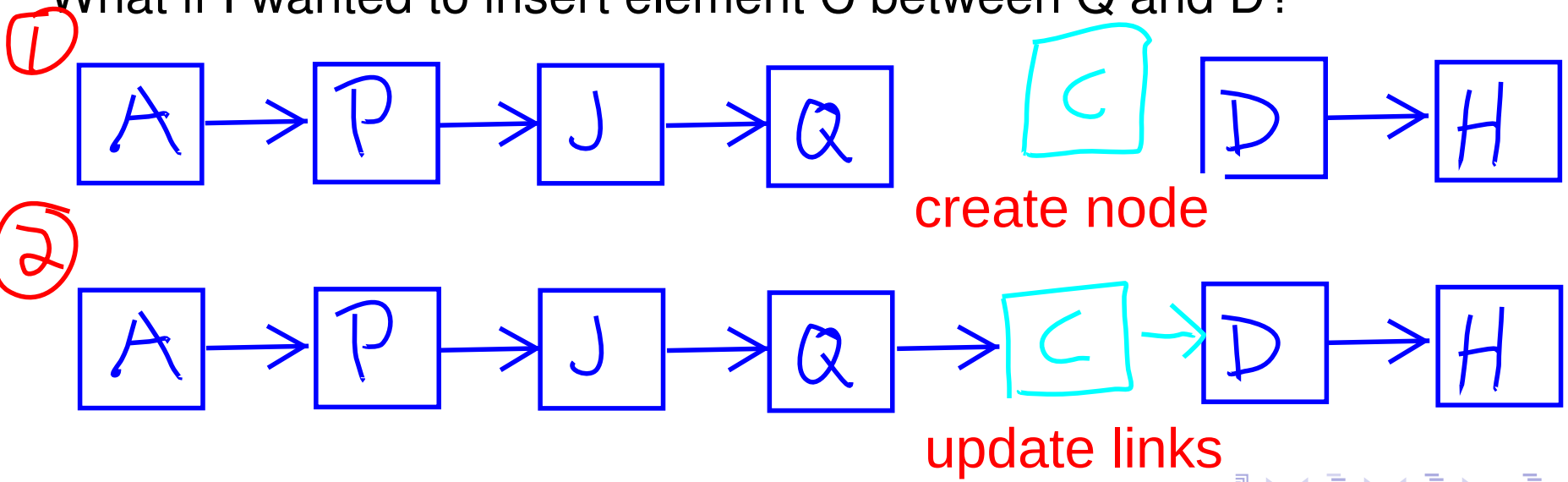
create node + add link

Linked Lists - insertion

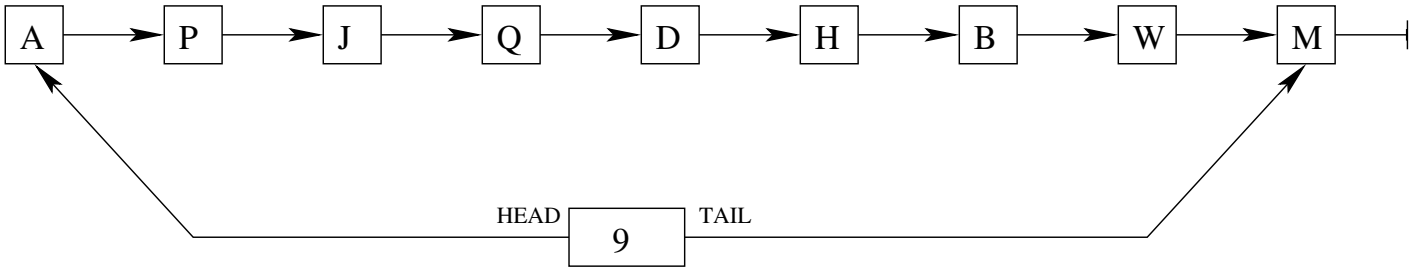
Insert the elements of following sequence of letters, one by one [A,P,J,Q,D,H,B,W,M]:



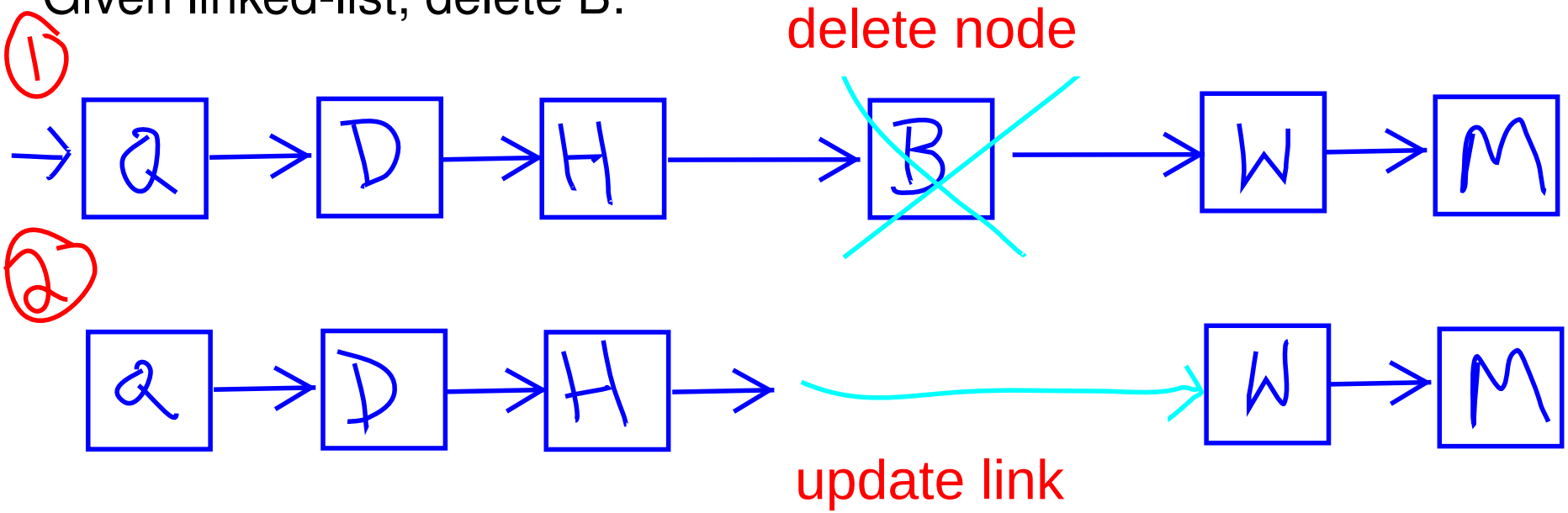
What if I wanted to insert element C between Q and D?



Linked Lists - deletion



Given linked-list, delete B:



Overview

1. Preliminaries
2. What is an algorithm and motivation for its study
3. Abstract Data Types
4. Data Structures
5. Summary

Summary

- What is an algorithm, examples of it, and why the study of algorithms and data structures are important.
- Discussed the GCD problem and how it can be solved.
- Learnt about abstract data structures and two data structures (arrays and linked lists).

Summary

- What is an algorithm, examples of it, and why the study of algorithms and data structures are important.
- Discussed the GCD problem and how it can be solved.
- Learnt about abstract data structures and two data structures (arrays and linked lists).
- Ultimately, remember the study of algorithms and data structures is for us to become better at solving problem.

Todo for next week

What should I be doing?

- Buy the textbook.
- Read Chapter 1 and Chapter 2 carefully.
- Next week (Chapter 2) covers complexity analysis.
- Be ready for next week. Chapter 2 arguably contains some of more difficult material of the course.
- Your Laboratory and Tutorial next week will focus on the fundamental data structures in Section 1.4.