COMP2230/COMP6230 Algorithms

Lecture 1

Professor Ljiljana Brankovic

Lecture Overview

- Course Overview

- Math Review:
  - Chapter 2, Section 2.1
    - Exponents, Logarithms, Series
    - Theorems and Proofs (not in the text)
    - Sequences and Strings, Logic, Binomial Coefficients
    - Inequalities, Upper Bounds, Lower Bounds …
  - Chapter 2, Section 2.2 Mathematical Induction
  - Chapter 2, Section 2.5 Graphs
  - Chapter 2, Section 2.6 Trees

Course Overview

- Lecturer: Prof Ljiljana Brankovic
  - Room: ES237
  - Email: Ljiljana.Brankovic@newcastle.edu.au
  - Office Hours: Wednesday 13:00 – 14:00

  Lectures:
  - Wednesdays 10.00 – 12.00 (GP201)

- Tutor: Mr Aaron Freemantle
  - Email: Aaron.Freemantle@uon.edu.au

  Tutorials:
  - Thursdays 14:00 – 16:00 (EFG20)
  - Fridays 10:00 – 12:00 (ES305)
  - Fridays 12.00-14.00 (EFG20)
  - Fridays 14.00-16.00 (AVG14)

Recommended reading – we will use the first book the most:

- "Algorithms" by Richard Johnsonbaugh and Marcus Schaefer, Pearson Prentice Hall, 2004.
- "Introduction to Design and Analysis of Algorithms" by A. Levitin, Addison Wesley, 2012.
- "Fundamentals of Algorithms" by G. Brassard and P. Bratley, Prentice Hall, 1996.

Web Page: Blackboard

All course materials, such as lecture slides, tutorial notes and assignment specification, will be made available online through Blackboard. A course discussion board will be available, which all students should read on a regular basis for useful information. The lecturer and tutor will monitor the discussion board regularly.

Course Description:

This course introduces students to the notion of algorithm efficiency and computational complexity. The basic data structures encountered in first year, such as lists, trees and graphs, are reviewed in light of their efficiency and correctness. Asymptotic measures of complexity are covered, and recurrence relations are introduced as an analytical tool. Problem-solving techniques such as the greedy strategy, divide-and-conquer, dynamic programming, and graph searching are covered. These techniques are illustrated upon optimization problems chosen for their practical relevance.

Assumed knowledge:

Students undertaking this course are assumed to have some experience in writing programs and data structures, to the level of SENG1120. They are also expected to have basic math knowledge to the level of MATH 1510.

Timeline

Week 1	19 – 23 July		
Week 2	26 – 30 July	Tutorials start	
Week 3	2 – 6 August	Quiz 1	3%
Week 4	9 – 13 August		
Week 5	16 – 20 August		
Week 6	23 – 27 August	Quiz 2	3%
Week 7	30 August - 3 September		
Week 8	6 - 10 September	Mid-term 1	15%
Week 9	13 – 17 September		
Midterm Break			
Week 10	12 - 16 October	Ass/Report due	16%
Week 11	19 - 23 October		
Week 12	26 - 30 October	Mid-term 2 Quiz 3	20% 3%
	Sometimes in the exam period	Final Exam	40%

- Quizzes will be done via Blackboard. You will be allowed to attempt each quiz as may time as you like within a specified time window, until you are happy with your knowledge (and your mark ;-). The highest score will count towards your mark.

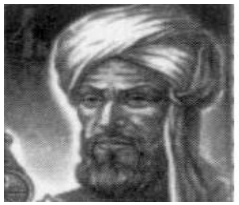
- I expect each one of you to score 100% in each of the 3 quizzes and will be rather disappointed if you settle for less. Thus, quizzes will be used both for learning and assessment.

- In order to pass the course, a student has to score at least 40% in the final exam.
  - Student achieving ≥ 25% but less than 40% will be offered an alternate assessment if, and only if, all other assessment items have been submitted, and the overall mark is ≥ 50%.
  - Students obtaining < 25% will not be offered an alternate assessment, and will fail the course, unless students have submitted Adverse Circumstances in accordance with the Adverse Circumstances Policy.

Introduction to Algorithms

An algorithm is a precise, unambiguous, step-by-step procedure for carrying out some calculation or, more generally, for solving some problem.

The word "algorithm" is believed to have been derived from the name of a ninth century Persian mathematician Musa Al-Khwarizmi.



Born around 780 in Baghdad (?), died around 850, see more at
http://www-history.mcs.st-and.ac.uk/Biographies/Al-Khwarizmi.html

Algorithmics is the study of algorithms (design and analysis). Algorithms are the heart of computer science – without algorithms, computer programs could not exist.

Studying algorithms will also help you to develop analytical and problem-solving skills.

Example 1. Is this an algorithm?

How to live a happy life

1. Find something to do, something to love and something to hope for.
2. Don't try to solve problems by the same level of thinking that created them.
3. Make others happy wherever you go, not whenever you go.
4. Keep in mind that twenty years from now you will be more disappointed by the things that you didn't do than by the ones you did do.
5. If you are going through hell, keep going.
6. Always remember that life is what happens to you while you're busy making other plans.

Example 2. Is this an algorithm?

Grandma's Sugar Cookies

Ingredients

1 cup shortening (part butter)
2 cup sugar
2 large eggs
5 cups flour
1 cup milk
2 ts baking soda
2 ts baking powder
Pinch of salt
Ground nutmeg

Instructions

1. Beat shortening, sugar and eggs together until light and lemon-colored. Add the rest of the ingredients.
2. Drop cookies onto ungreased cookie sheet with a teaspoon.
3. Bake until just light brown.
4. Cool on racks and pipe your favourite frosting.

Introduction to Algorithms

Algorithms are typically (but not always) expected to have the following:

- Input
- Output
- Precision
- Determinism
- Finiteness
- Correctness
- Generality

Not all algorithms have all the properties that we listed.

An operating system can be taught of as an online algorithm that never terminates, rather than a finite algorithm with input and output.

Randomized algorithms do not require that results of each step be uniquely defined.

Sometimes we use heuristics, which are not general;
or approximation algorithms which are not "correct" in the sense that they provide a solution that is approximate rather than exact – we'll explain this better later.

Example 3.

Find the maximum of the given 3 numbers (a, b and c).

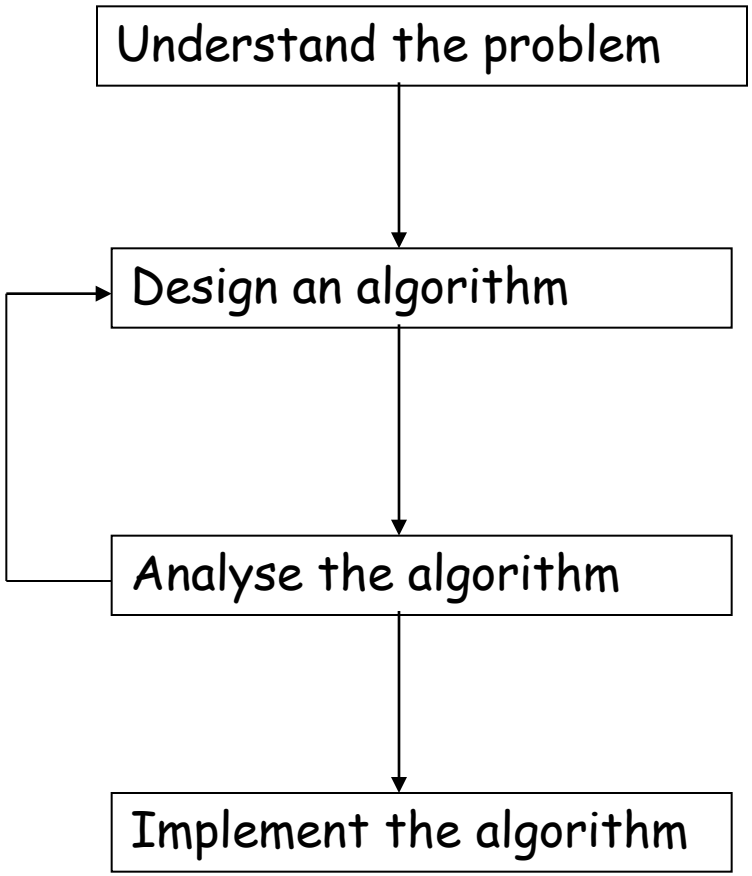
```
Input Parameters: a, b, c
Output Parameter: x
max(a,b,c,x) {
   x = a
   if (b > x) // if b is larger than x, update x
        x = b
   if (c > x) // if c is larger than x, update x
        x = c
}
```

Note the use of pseudocode!

Example 4. Algorithm 1.2.2
Finding the Maximum Value in an Array Using a While Loop

This algorithm finds the largest number in the array $s[1], s[2], \ldots, s[n]$.

```
Input Parameter: s
Output Parameters: None
array_max_ver1(s) {
    large = s[1]
    i = 2
    while (i ≤ s.last) {
        if (s[i] > large) // larger value found
            large = s[i]
        i = i + 1
    }
    return large
}
```


Understand the problem
Design an algorithm
Analyse the algorithm
Implement the algorithm

Step 1: Understanding the problem

It is very important to fully understand the problem. It is usually helpful to play with a few small examples.

If in doubt, seek clarification.

Step 2: Designing an algorithm

Algorithm design is still more art than science.

You first need to decide what kind of algorithm you want:

- Sequential vs parallel
- Exact vs approximation (or heuristic).

Choose appropriate data structures.

Choose basic techniques: brute force, divide-and-conquer, greedy algorithms, etc.

Step 3: Analysing the algorithm

- Correctness
- Termination
- Simplicity
- Generality
- Time analysis
- Space analysis

Example 5: Cryptography

It is easy to construct a brute force algorithm to break a cryptosystem; such an algorithm is correct and finite; however, the time taken by the algorithm would typically be extremely long which makes the algorithm impractical.

Future challenges:

- Quantum Computing
  - Uses qubits instead of bits
  - While a bit can be either 0 or 1, qubit can be 0, 1 or any quantum superposition of 0 and 1
- DNA Computing
  - Uses DNA sequence (a sequence over the following alphabet: A, C, T and G, where A-T and C-G are complements)

Some important problem types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

Mathematics Review

Exponentiation:

1. $x^a \cdot x^b = x^{a+b}$
2. $\frac{x^a}{x^b} = x^{a-b}$
3. $(x^a)^b = x^{ab}$
4. $x^n + x^n = 2x^n$
5. $2 \cdot 2^n = 2^{n+1}$

Mathematics Review

Logarithms ($b > 0, b \neq 1$):

1. For $b > 0$ and $b \neq 1$, $b^e = x$ if and only if $\log_b x = e$
2. $\log xy = \log x + \log y;\ x, y > 0$
3. $\log \frac{x}{y} = \log x - \log y;\ x, y > 0$
4. $\log_b x^y = y \log_b x$
5. if $a > 0, a \neq 1, \log_a x = \frac{\log_b x}{\log_b a}$

   (If it's easier to think in exponents than logarithms,
   think of $a = b^d$, thus $d = \log_b a$
   let $x = a^e$, thus $e = \log_a x$
   also, $x = (b^d)^e = b^{de}$, thus $de = \log_b x$,
   and so $e = \frac{\log_b x}{d} = \frac{\log_b x}{\log_b a}$.
   Therefore, $\log_a x = \frac{\log_b x}{\log_b a}$

6. if $b > 1$ and $x > y > 0$, $\log_b x > \log_b y$

Mathematics Review

Series:

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Theorems & Proofs

- A ***theorem*** is a mathematical statement which has been proved true.
- A ***lemma*** is a 'small' theorem, usually used in a proof of a more important mathematical statement.
- A ***corollary*** is a mathematical statement which easily follows from a theorem.
- A ***proof*** is a logical argument that a mathematical statement is true.

Proof by construction:

A mathematical statement about the existence of an object can be proved by constructing the object.

Proof by contradiction:

Assume that a mathematical statement is false and show that the assumption leads to a contradiction.

Proof by induction:

Consider a statement about a sequence of integers.

- ***Base case***: prove the statement true for the smallest considered value a_1.
- ***Induction step***: assume that the statement holds for the first k values $a_1, \dots, a_k$ (***induction hypothesis***); prove the statement true for value a_{k+1}.

Example 6:

Theorem. Let $P(X)$ be the power set of a finite set X. Then $|P(X)| = 2^{|X|}$.

The power set $P(X)$ of the set X is the collection of all subsets of X.

Example 7: Let $X_1 = \{1,2,3\}$. Then

. $P(X_1) = \{\emptyset,\ \{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,2,3\}\}$

Example 8: Let $X_2 = \{1,2,3,4\}$. Then

$$P(X_2) = \{\emptyset,\ \{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,2,3\},$$
$$\{4\},\ \{1,4\},\ \{2,4\},\ \{3,4\},\ \{1,2,4\},\ \{1,3,4\},\ \{2,3,4\},\ \{1,2,3,4\}\}$$

Example 6 cont.

Proof of the theorem:

Base case:

$|X| = 0$.

Then $X = \emptyset$, $P(X) = \{\emptyset\}$ and so $|P(X)| = 1 = 2^0 = 2^{|X|}$.

Induction step:

Assume $|P(X)| = 2^{|X|}$ for all sets of cardinality $0, 1, 2, \ldots, n$.

Let $|X| = n + 1$, $x \in X$ and $Y = X - \{x\}$. Then $|Y| = n$. It is easy to show that $|P(X)| = 2|P(Y)|$. (Show it!)

$$|P(X)| = 2|P(Y)|$$
$$= 2\,(2^n) \qquad \text{by induction hypothesis}$$
$$= 2^{n+1}$$
$$= 2^{|X|}$$

Example 9

Show that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Base case:

$$\sum_{i=1}^{1} i = \frac{1(1+1)}{2}$$

$$1 = 1$$

Induction step: Assume $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

We need to show that $\sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2}$

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1) = \frac{n(n+1)}{2} + (n+1)$$

$$= \frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

Polynomials

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$$

n – degree of the polynomial $p(x)$

$c_n, c_{n-1}, \ldots, c_1, c_0$ – coefficients of the polynomial $p(x)$

Example 10. $p(x) = 10x^2 + 3$

$n = 2, c_2 = 10, c_1 = 0, c_0 = 3$

Example 11. $p(x) = -3x^5 + x$

$n = 5, c_5 = -3, c_4 = c_3 = c_2 = c_0 = 0, c_1 = 1$

Example 12. $p(x) = 3$

$n = 0, c_0 = 3$

Polynomials – contd.

If $p(x)$ and $q(x)$ are polynomials, so are $p(x) + q(x)$, $p(x) - q(x)$ and $p(x) \times q(x)$.

Example 13:

If the degree of $p(x)$ is 8 and the degree of $q(x)$ is 7, what is the degree of $p(x) - q(x)$?

When both degrees are 7?

Intervals

Intervals can be defined over various sets of numbers, e.g., N (natural numbers, i.e., positive integers), R (real numbers), etc.

- Closed interval: $[a, b] = \{ x \mid a \leq x \leq b \}$
- Open interval: $(a, b) = \{ x \mid a < x < b \}$
- Half-open interval:

$$[a, b) = \{ x \mid a \leq x < b \}$$

or

$$(a, b] = \{ x \mid a < x \leq b \}$$

Sequences and Strings

A finite sequence a is a function from the set $\{0,1,\ldots,n\}$ to a set X.

The subscript i (as in a_i or $a[i]$) is called the ***index*** of the sequence.

A finite sequence can be written as $a_0, a_1, \ldots, an$ or

$a[0], a[1], \ldots, a[n]$ or

$a[0..n]$

An infinite sequence a is a function from the set $\{0,1,\ldots\}$ to a set X.

An infinite sequence is denoted as $a_0, a_1, \ldots$ or

$a[0], a[1], \ldots$ or

$\{a_i\}_{i=0}^{\infty}$

Example 14 – finite sequence of weightings of assessment items in COMP2230/6230: $3,3,15,16,20,3,40$

Example 15 – an infinite sequence: $a_i = 2i + 3,\ i \geq 0$

$3, 5, 7, 9, 11, \ldots$

A sequence a of real numbers is:

- increasing if $a_i < a_{i+1}$
- decreasing if $a_i > a_{i+1}$
- nonincreasing *if* $a_i \geq a_{i+1}$
- nondecreasing *if* $a_i \leq a_{i+1}$

for all i for which i and $i + 1$ are indexes of the sequence

Example 16: Sequence $1, 2, 3, 4, 5$ is increasing and nondecreasing.

Example 17: Sequence $1,2,2,3,3,4$ is nondecreasing but not increasing.

Example 18: Sequence 16 is increasing, decreasing, nonincreasing and nondecreasing.

A subsequence of a given sequence a consists of only certain terms of a which appear in the same order as in a.

Example 19:

12 25 46 *is a subsequence of*

0 1 8 12 15 17 25 33 37 46

but 25 12 46 *is not.*

A ***string*** (***word***) over alphabet X is a finite sequence $t[1], t[2], \ldots, t[n]$, where n is the length of the string.

The empty string (word) ε is a string of length zero.

$t[i..j]$ is a substring of $t[1], t[2], \ldots, t[n]$

- If $i < j$ then $t[i..j]$ is the substring $t[i], \ldots, t[j]$
- If $i = j$ then $t[i..j]$ is the substring $t[i]$
- If $i > j$ then $t[i..j]$ is the empty string

Logic

A Boolean variable p can be either ***true*** or ***false***. Basic Boolean operators:

$\vee$ (or)

$\wedge$ (and)

$\bar{}$ (not)

We can use other operators such as $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence).

Precedence of Boolean operators: not, and, or.

A literal is a variable (for example p) or the negation of the variable ($\bar{p}$).

A ***Boolean expression*** is an expression containing Boolean variables, operators and parentheses.

Example 20: $p_1 \vee p_2 \wedge p_3$

Normal forms:

1. *Conjunctive normal form* (CNF): $c_1 \wedge c_2 \wedge \ldots \wedge c_n$ where each c_i is a ***clause*** of the form $p_1 \vee p_2 \vee \ldots \vee p_n$ and each p_i is a literal.

2. *Disjunctive normal form* (DNF): $c_1 \vee c_2 \vee \ldots \vee c_n$ where each c_i is a ***clause*** of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ and each p_i is a literal.

p	q	$\bar{p}$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Binomial Coefficients

The number of k-subsets in a set of n elements:

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!} = \frac{n(n-1)(n-2)\ldots(n-k+1)}{k!}$$

Example 21: $S = \{1,2,3,4,5\}$

The k-subsets of the set S:

0	1	2	3	4	5
∅	1	12	123	1234	12345
	2	13	124	1235	
	3	23	134	1245	
	4	14	234	1345	
	5	24	125	2345	
		34	135		
		15	235		
		25	145		
		35	245		
		45	345		

Upper and Lower Bound

X – a nonempty set of real numbers.

- An ***upper bound***: a number u, such that $x \leq u$ for all $x \in X$.
- *A **lower bound***: a number l, such that $x \geq l$ for all $x \in X$.
- The least upper bound (***supremum***): the number s such that
  1. s is an upper bound, and
  2. for any upper bound a, $a \geq s$.
- The greatest lower bound (***infimum***): the number *i* such that
  1. i is a lower bound, and
  2. for any lower bound a, $a \leq i$.

Example 21: $X = \{\sum_{i=0}^{n} \frac{1}{2^i} \mid n \geq 0\}$

We can show that $\sum_{i=1}^{n} \frac{1}{2^i} < 1$

It follows that $\sum_{i=0}^{n} \frac{1}{2^i} < 2$

So 2 is an upper bound for X.

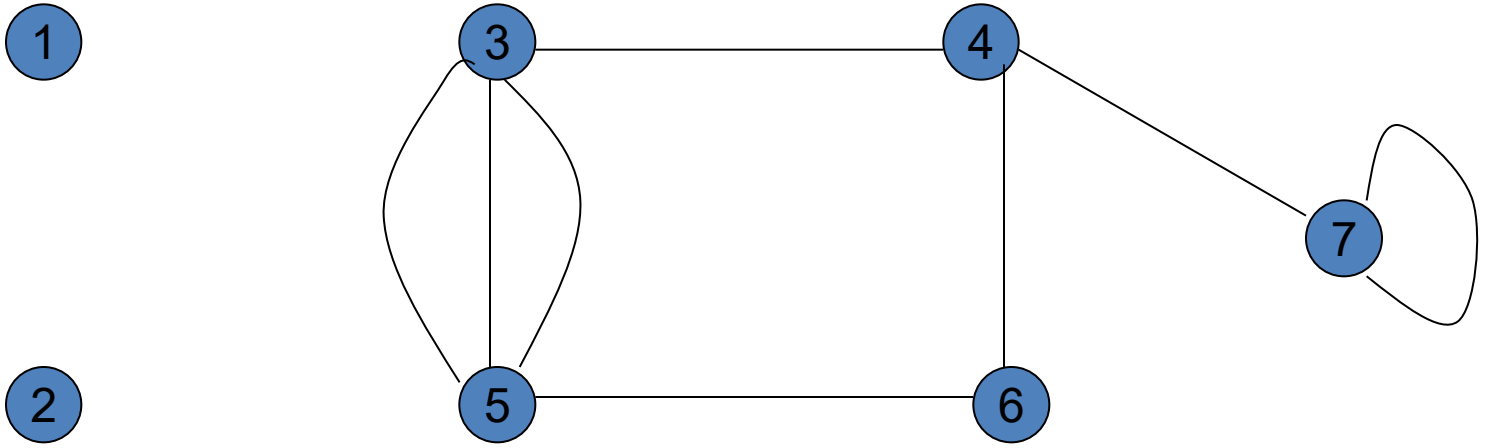
Can you show that 2 is also the supremum for X?

Graphs

An undirected graph $G = (V, E)$ consists of a set V of vertices and a set E of edges. An edge $e \in E$ is an unordered pair of vertices.

Example 22: $V = \{1,2,3,4,5,6,7\}$

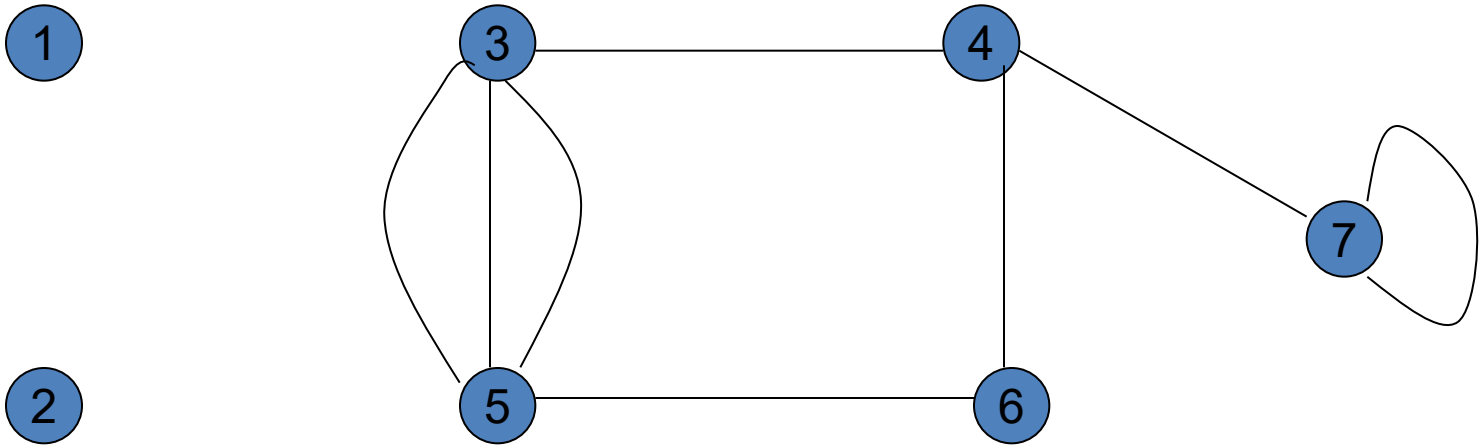
$$E = \{(3,5), (3,5), (5,3), (3,4), (4,6), (5,6), (4,7), (7,7)\}$$



Vertices 1 and 2 are called ***isolated*** vertices (they have no edges incident on them); there are ***multiple*** edges connecting vertices 3 and 5. There is a ***loop*** at vertex 7 (an edge incident on a single vertex).

A graph without loops and multiple edges is called a ***simple*** graph.

The ***degree*** of a vertex v is the number of edges incident on v.

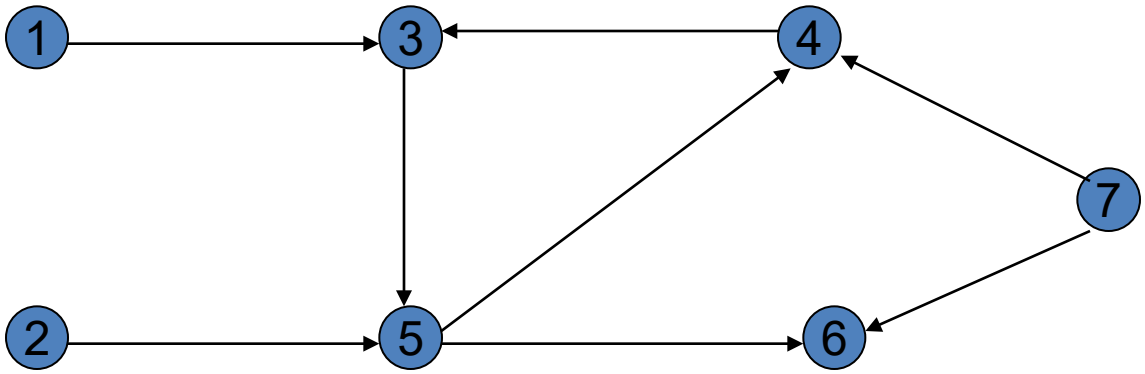


vertex	1	2	3	4	5	6	7
degree	0	0	4	3	4	2	3

A ***directed graph*** (***digraph***) G consists of a set V of vertices and a set E of directed edges where each $e \in E$ is an ***ordered pair*** of vertices.

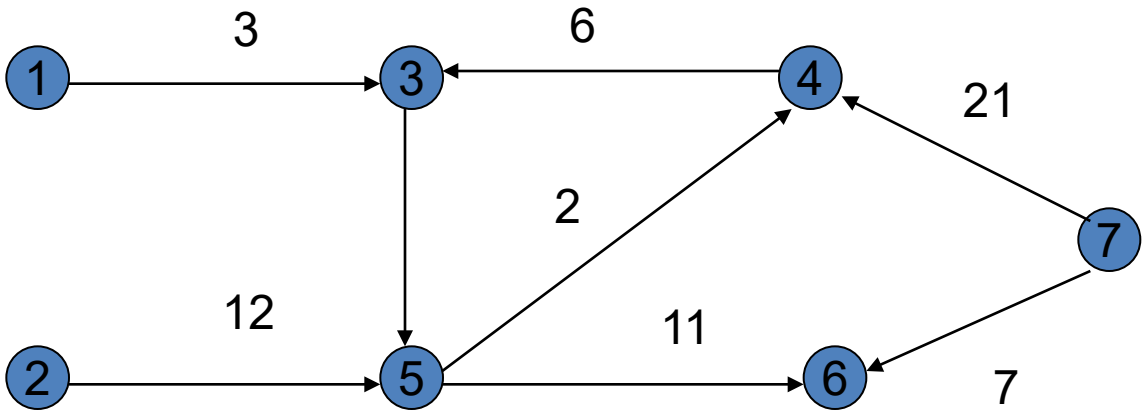
Example 23: $V = \{1,2,3,4,5,6,7\}$

$$E = \{(1,3), (2,5), (3,5), (4,3), (5,4), (5,6), (7,4), (7,6)\}$$

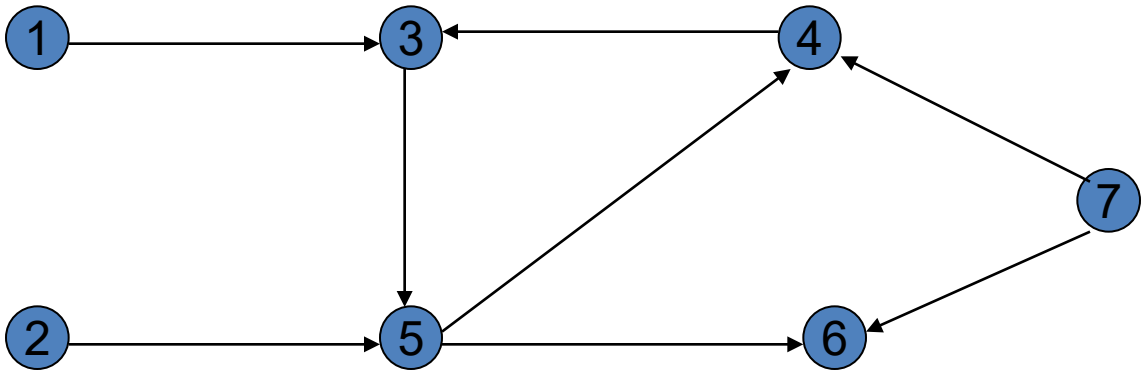


vertex	1	2	3	4	5	6	7
in-degree	0	0	2	2	2	2	0
out-degree	1	1	1	1	2	0	2

In a ***weighted graph*** every edge has a real number associated with it – the ***weight*** or ***cost***.



Graphs can be represented by an ***adjacency matrix***.

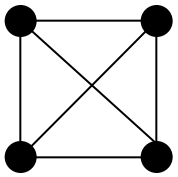


	1	2	3	4	5	6	7
1	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0
5	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0
7	0	0	0	1	0	1	0

Some special graphs

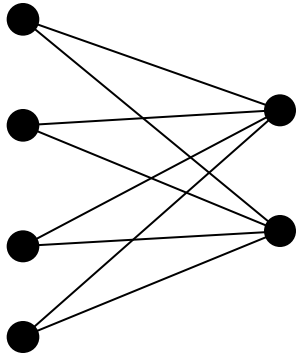
K_n – complete graph on n vertices

Example 24: K_4



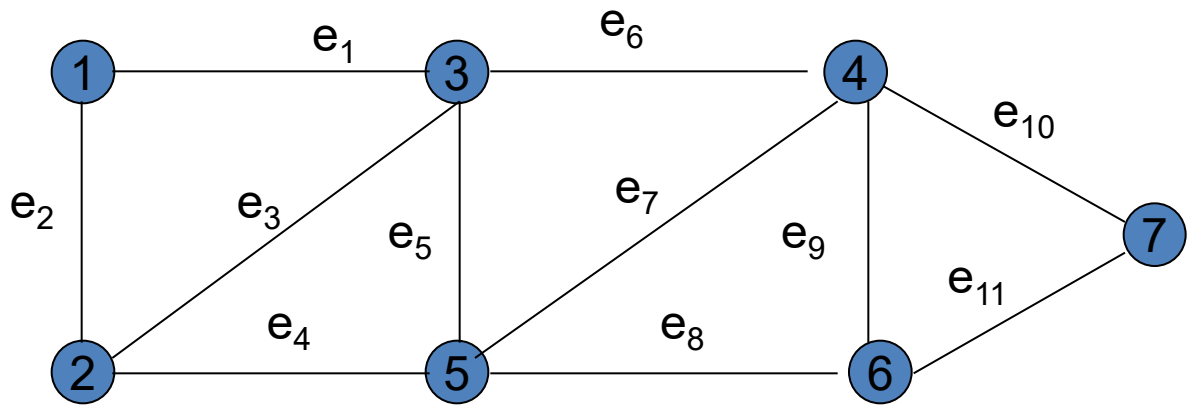
$K_{m,n}$ – complete bipartite graph

Example 25: $K_{4,2}$



A path of length n from vertex v_0 to vertex v_n in a graph G is an alternating sequence of $n+1$ vertices and n edges, starting with vertex v_0 and ending with vertex v_n.

Example 26:



A path of length 7 from 2 to 7: $(2, e_2, 1, e_1, 3, e_3, 2, e_4, 5, e_8, 6, e_9, 4, e_{10}, 7)$

In a weighted graph, the length of a path is the sum of the weights on the edges in the path.

- A graph G is ***connected*** if there exists a path between any two vertices of G.
- A disconnected graph is a union of its connected ***components***.
- A distance $dist(v, w)$ from vertex v to vertex w is:
  - the length of a shortest path from v to w
  - ∞, if there is no path from v to w
- The ***diameter*** of graph G is $\max\{dist(v, w) \mid v \text{ and } w \text{ are vertices in } G\}$
- A ***simple*** path from v to w is a path from v to w with no repeated vertices.
- A cycle is a path of length greater than 0 from v to v with no repeated edges.
- A simple cycle is a cycle without repeated vertices (except for the beginning and ending vertex).
- Acyclic graph is a graph with no cycles.

A ***Hamiltonian*** cycle in a graph G is a cycle that contains each vertex in G exactly once.

Finding a Hamiltonian cycle in a graph is a difficult problem and no efficient algorithm is known.

Euler cycle in a graph G is a path from v to v with no repeated edges that contains all of the edges and all of the vertices of G.

A graph G has an Euler cycle if and only if G is connected and the degree of every vertex is even.

The complement of a simple graph G is denoted $\bar{G}$; it contains the same vertices as G, and it contains all the edges that are not in G. Formally, an edge exists in $\bar{G}$ if and only if it does not exist in G.

Two graph, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there exists a bijection (a one-to-one and onto function) f from V_1 to V_2 such that vertices v and w are adjacent in G_1 if and only if vertices $f(v)$ and $f(w)$ are adjacent in V_2.

Graph coloring is coloring of the vertices of a graph such that adjacent vertices receive different colors.

Trees

A tree T is a simple graph such that for any two vertices v and w in T there is a unique simple path from v to w.

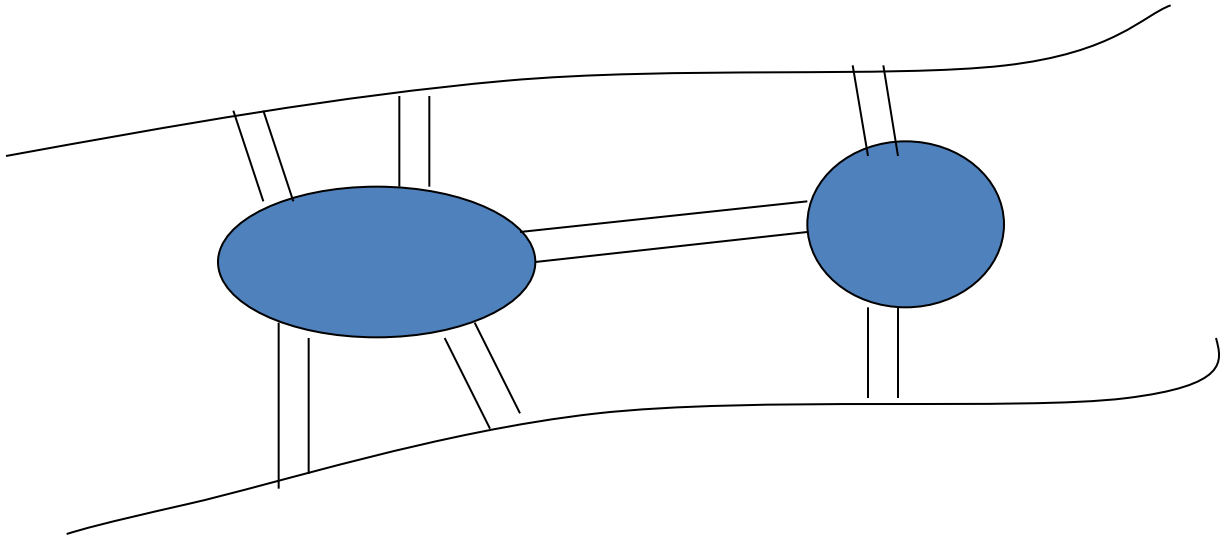
The following statements about a graph G with n vertices are equivalent:

- T is a tree.
- T is connected and acyclic.
- T is connected and has $n-1$ edges.
- T is acyclic and has $n-1$ edges.

- In a rooted tree a particular vertex is designated as a root.
- The level of a vertex v is the length of a simple path from the root to v.
- The height of a rooted tree is the maximum level that exists in the tree.
- Parent, child, leaf vertex, inner vertex, subtree at vertex v.
- A binary tree is a tree in which every vertex has at most two children.

Königsberg Bridges

(solved by the Swiss mathematician Leonard Euler (1707-1783) in 1736)



Can you cross all the bridges exactly once and return to the starting point?

Icosian Game (A Voyage Around the World)

(invented by the Irish mathematician Sir William Hamilton, 1805-1865)

Find Hamiltonian cycle.