Theoretical Computer Science Bridging Course Introduction / General Info

–

Winter Term 2021/22

Fabian Kuhn

About the Course

Topics

- Foundations of theoretical computer science
- Introduction to logic

No lectures

- There are recordings which you are supposed to watch

Exercises

- There will be weekly exercises which you need to do
  - Doing the exercises is not mandatory, but highly recommended

Exam

- A oral exam at the end of the term
  - Details will be published on the course web page a.s.a.p.

About the course

What is the purpose of the course?
Who is it targeted to?

- The course is for incoming M.Sc. students who do not have the necessary theory background required by the M.Sc. program.
  - E.g., students who did not study computer science or students from more applied schools, ...

Website

- All necessary information about the course will be published on

  http://ac.informatik.uni-freiburg.de/teaching/ws21_22/tcs-bridging.php

  – Or go to my group's website: http://ac.informatik.uni-freiburg.de
  – Then follow teaching – winter term 2021/22 – TCS bridging course

- Please check the website for
  – Recordings and slides
  – Exercises and sample solutions
  – Pointers to additional literature
    (e.g., written lecture notes from an older version of this lecture)
  – Information about the exam
  – …

Exercises

There will be weekly exercise sheets:

- Exercise sheets are published at the latest on Monday on the website
- Exercises are due after one week on the Sunday night before the exercise tutorial
  - If you want corrections / comments from your tutor
- Hand in your exercises on paper (in tutorial) or by email

- If you work in a group, the group needs to hand in one solution
  - Make sure that all students participate in solving & writing equally!
- After getting back your exercises, you can meet and discuss the exercises with your tutor
  - On Mondays or if additional help is necessary on request

Exercise Tutorials

Assistant for the course:

- Salwa Faour, salwa.faour@cs.uni-freiburg.de

Weekly Tutorials:

- There is a weekly tutorial on Monday from 10:15 – 12:00
  - The tutorials will be online through Zoom.
- In the tutorial, we discuss the upcoming exercise sheet and your solutions of the last exercise sheet
  - You are required to actively participate in the tutorials and ask questions.
- Also ask your tutor if you have any questions!

Exercises

The exercises are the most important part of the course!

- To pass the exam, it is important that you do the exercises
- If you feel comfortable with all the exercises, you should also be able to pass the exam

- When working in groups, make sure that you all participate in solving the questions and in writing the solutions!
  - You should all be able to explain your solutions to your tutor.

Course Topics

Foundations of Theoretical Computer Science

- Automata theory
- Formal languages, grammars
- Turing machines
- Decidability
- Computational complexity

Introduction to Logic

- Propositional logic
- First order logic

Purpose of the Course

Goal: Understand the **fundamental capabilities** and **limitations** of **computers**

- What does it mean to "compute"?
  - Automata theory
- What can be computed?
  - Theory on computability/decidability
- What can be computed efficiently?
  - Computational complexity

Meaning of “Computing”

Mathematical Models

• Turing machines	1930s
• Finite state automata	1940s
• Formal grammars	1950s

Practical Aspects

• Compute architectures	1970s
• Programming languages	1970s
• Compilers	1970s

Is My Function Computable?

Write an algorithm / computer program to compute it

- Can it compute the right answer for every instance?
- Does it always give an answer (in finite time)?
- Then you are done.

Otherwise, there are two options

- There is an algorithm, but you don't know it
- There is no algorithm → the problem is unsolvable

Formally proving computability is sometimes hard!

- But you will learn how to approach this…

Is My Function Computable?

- Many “known” problems are solvable
  - Sorting, searching, knapsack, TSP, …
- Some problems are not solvable
  - Halting problem
  - Gödel incompleteness theorem
- Don’t try to solve unsolvable problems!

Can I Compute My Function Efficiently?

- Some problems are "easy"
  - Can we formally define what this means?
- **Complexity theory** is about this
  - Complexity classes, tools for checking membership
- It is important to know how hard a problem is!

- **Feasible problems:**
  - E.g., sorting, linear programming, LZW compression, primality testing, …
  - Time to solve is polynomial in the size of the input
- **Problems that are considered infeasible**
  - Some scheduling problems, knapsack, TSP, graph coloring, …
  - Important open question: "Is $\mathrm{P} = \mathrm{NP}$"?
- **Unfeasible problems**
  - Time exponential in input, e.g., quantified Boolean formula

Questions?

Warming up

for TCS Bridging Course

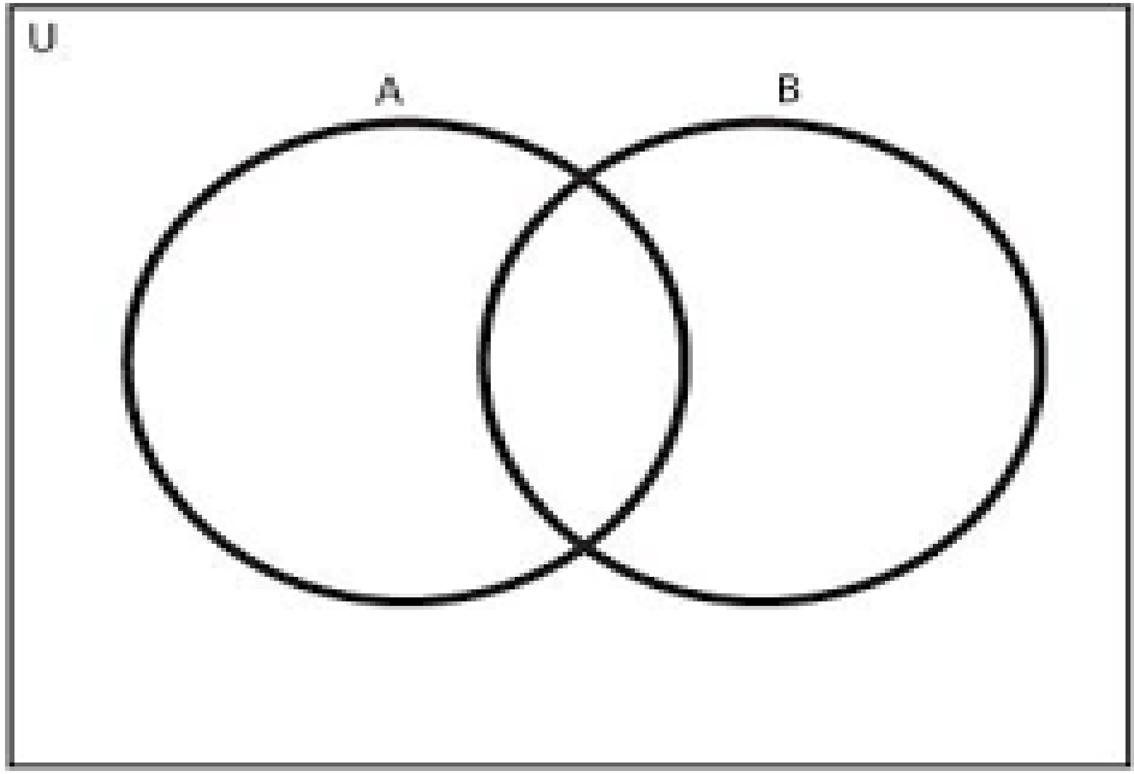
- ***Mathematical objects, tools, notions:***
- Sets
- Sequences
- Functions
- Graphs
- Strings and languages

- ***Types of Proof:***
- By construction
- By contradiction
- By induction
- By counterexample

- The alphabet set $\sum$ = { a, c, n, o, r}
- A = {no, corona}
- B = {no, corona, roar, ac}
- Is $A \subseteq B$?
- Is $B \subseteq A$?
- $A \cup B$?
- $A \cap B$?
- $B \setminus A$?
- $A \setminus B$?

- For any two sets A and B,

$A \,\Delta B = \emptyset \Leftrightarrow A = B$



- For any two sets A and B,

$$A \, \Delta B = \emptyset \Leftrightarrow A = B$$

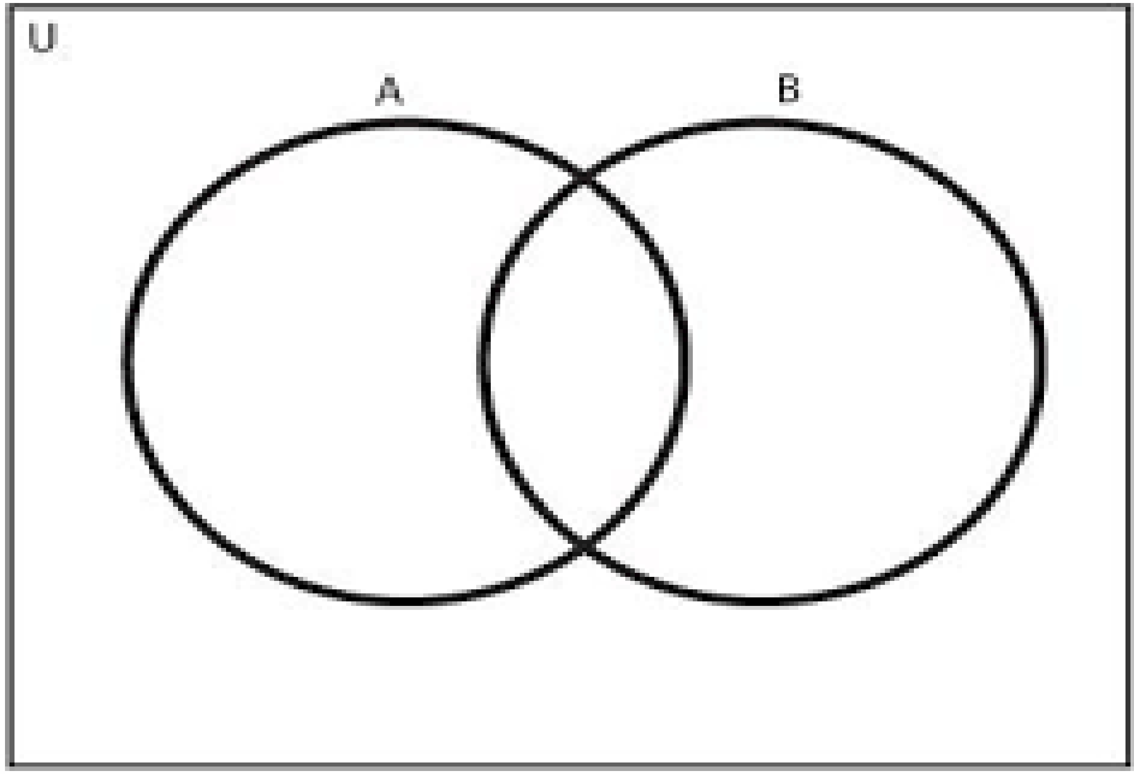
Proof:

$\Rightarrow$): $A \, \Delta B = (A \setminus B) \cup (B \setminus A) = \emptyset$

$(A \setminus B) = \emptyset$ and $(B \setminus A) = \emptyset$

$A \subseteq B$ $\qquad$ $B \subseteq A$

A=B



- ***<u>Mathematical objects, tools, notions:</u>***
- Sets
- Sequences
- Functions
- Graphs
- Strings and languages

- ***<u>Types of Proof:</u>***
- By construction
- By contradiction
- By induction
- By counterexample

A 10 Year Old Discovered This Famous Formula

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

Mind
Your
Decisions

Induction

- **Goal:** for an integer $n \geq 0$, use mathematical induction to prove a statement holds true for all values of *n.*

<u>2 STEPS:</u>

- **Base step** :

prove the statement true for $n = 0$

- **Induction step**:

assume the statement holds for any given case $n = k$, where $k \geq 0$ and use this assumption to prove the statement true for $n = k + 1$.

- Use proof by induction to prove
- *1+2+…+n*= $\frac{n(n+1)}{2}$, for n ≥ 1

- Use proof by induction to prove
- *1+2+…+n=* $\frac{n(n+1)}{2}$, for n $\geq$ 1
- Base step: for n=1, we have $\frac{1(1+1)}{2}$ = 1
- Induction step: assume for any case *n=k* holds, where k is some integer k $\geq$1

i.e. *1+2+…+k* = $\frac{k(k+1)}{2}$, where k is some integer k $\geq$1

Now, let's prove the statement true for *n=k+1*

i.e. 1+2+…+(k+1)= $\frac{(k+1)(k+2)}{2}$ (is it true?)

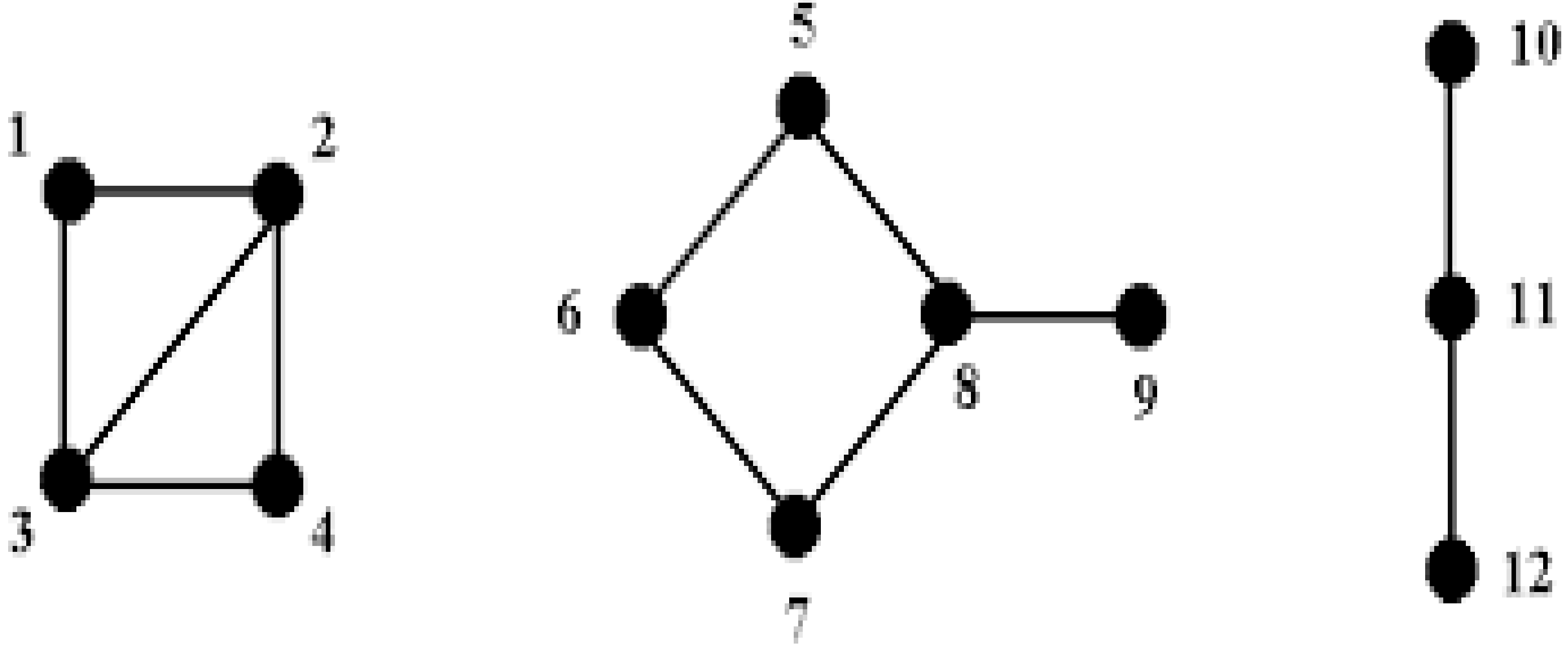
1+2+…k + (k+1)= $\frac{k(k+1)}{2}$ + *(k+1)=* $\frac{k(k+1)+2(k+1)}{2}$ = $\frac{(k+1)(k+2)}{2}$ (Yes!)

- ***Mathematical objects, tools, notions:***
- Sets
- Sequences
- Functions
- Graphs
- Strings and languages

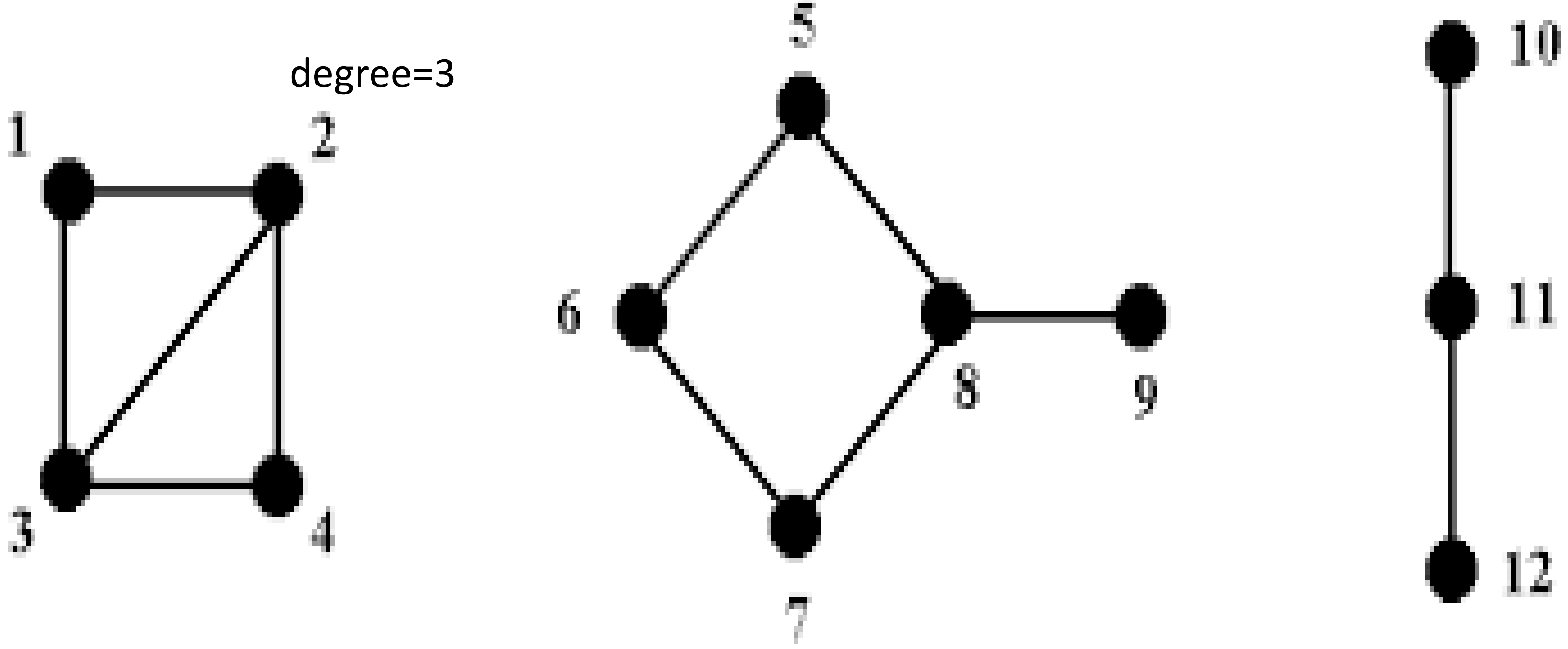
- ***Types of Proof:***
- By construction
- By contradiction
- By induction
- By counterexample

Graphs

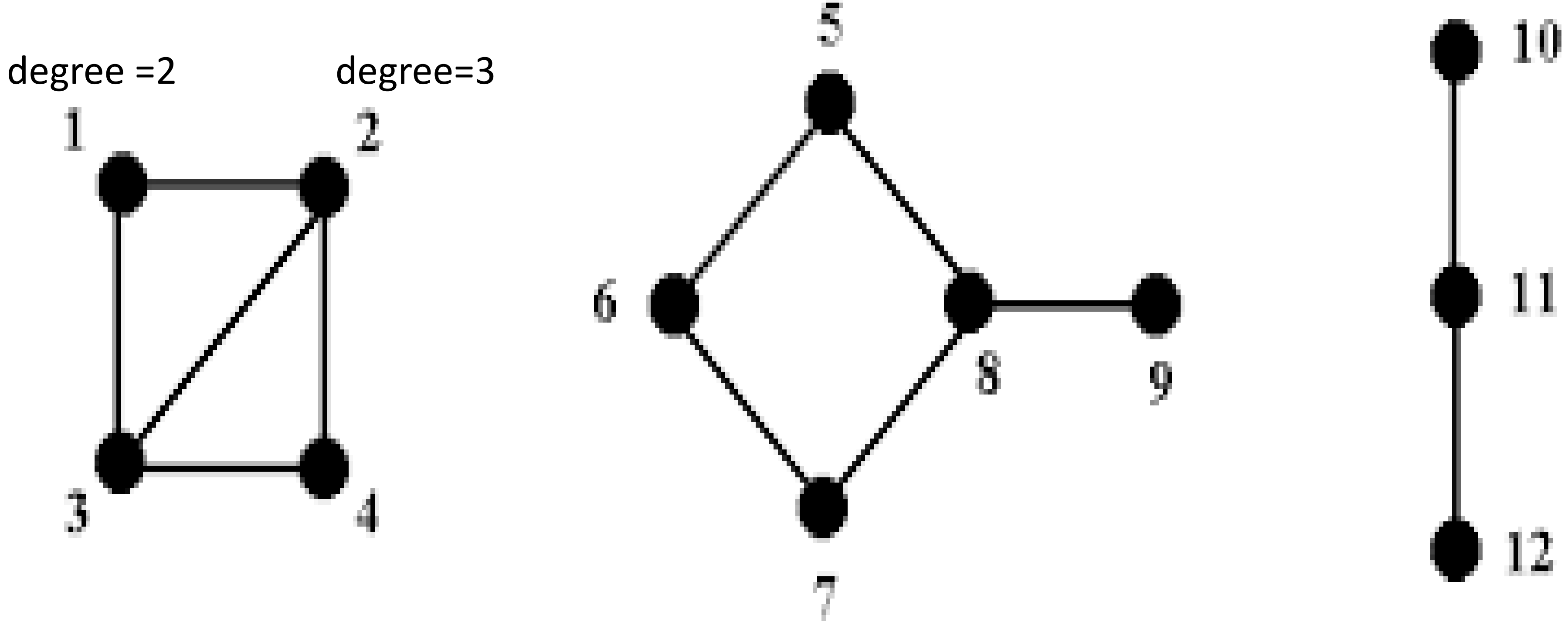
We write $G = (V, E)$.



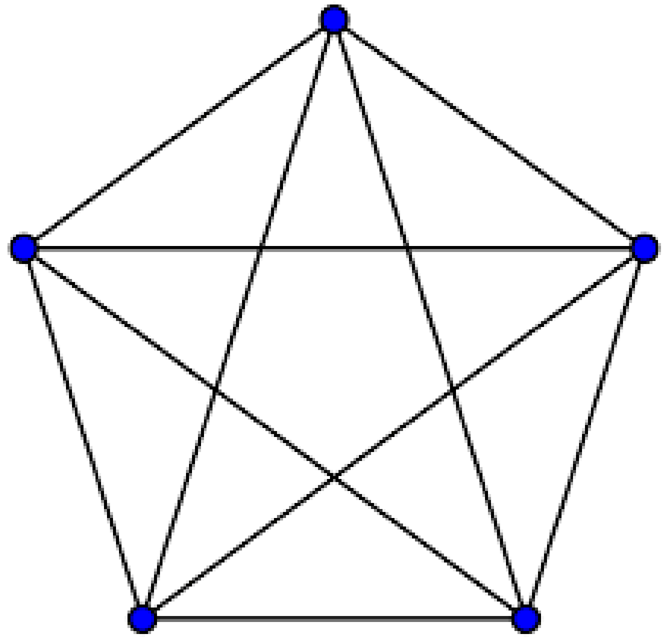
Graphs



Graphs

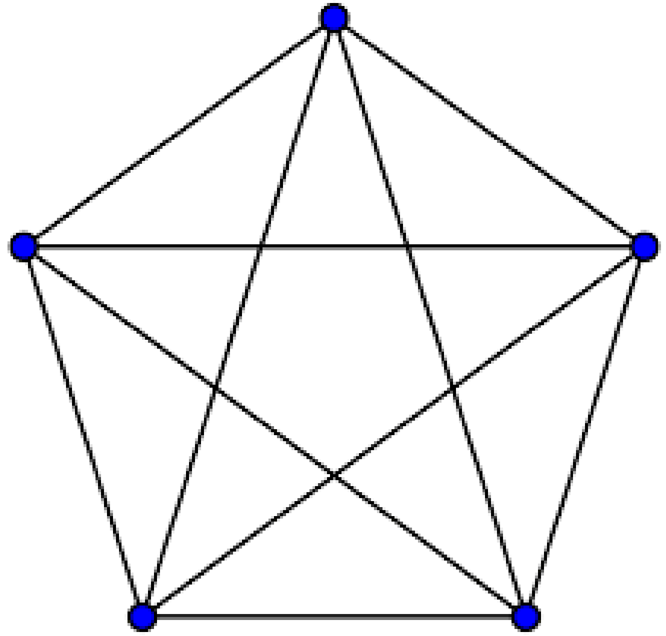


Q. How many edges are there in a *complete graph on **n** vertices*?



Q. How many edges are there in a *complete graph on **n** vertices*?

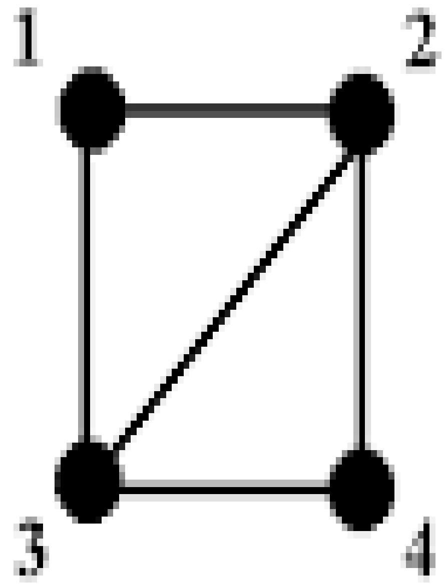
$$\frac{\sum_{v \in V} degree\,(v)}{2} = \frac{n(n-1)}{2} \text{ edges}$$



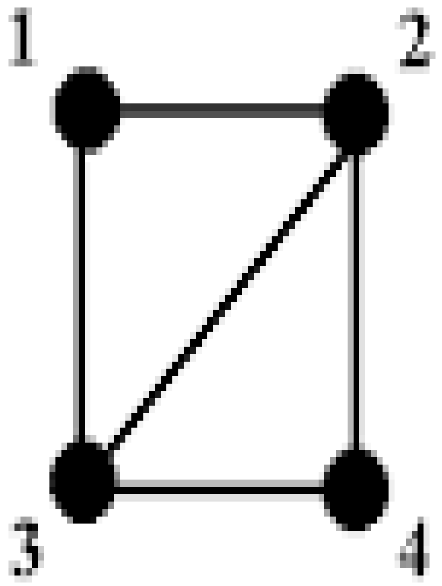
Don't count each edge twice!

- How many edges are there in a simple graph $G = (V, E)$?

- $\sum_{v \in V} degree\ (v) =$



- How many edges are there in a simple graph $G = (V, E)$?
- $\sum_{v \in V} degree\ (v) = 2\,|E|$ (Handshaking Lemma)
- Each edge contributes 2 to the sum on the left.



Can you?

- Draw a graph on 5 nodes such that each node is of degree 3.

Can you?

Draw a graph on 5 nodes such that each node is of degree 3

- Solution: you can't!

- Sum of all degrees= 5 x 3= 15

- See you Next Week !