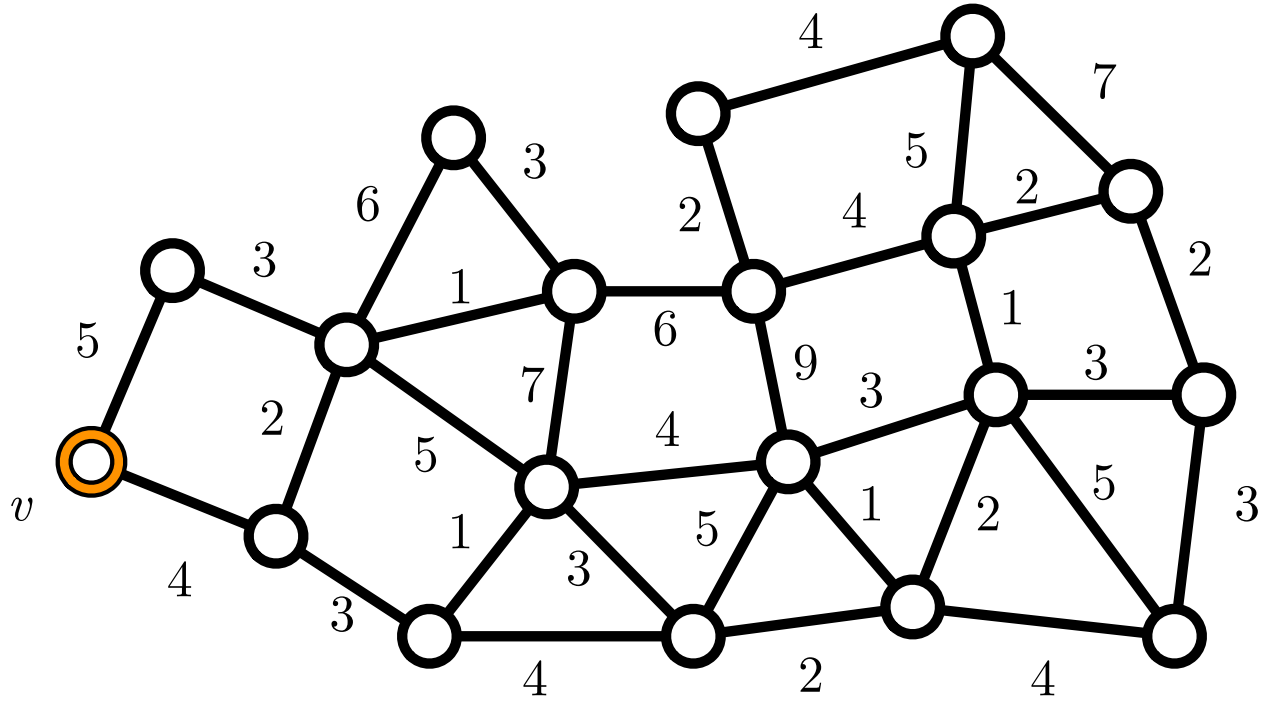
CS-E4530

Computational Complexity Theory

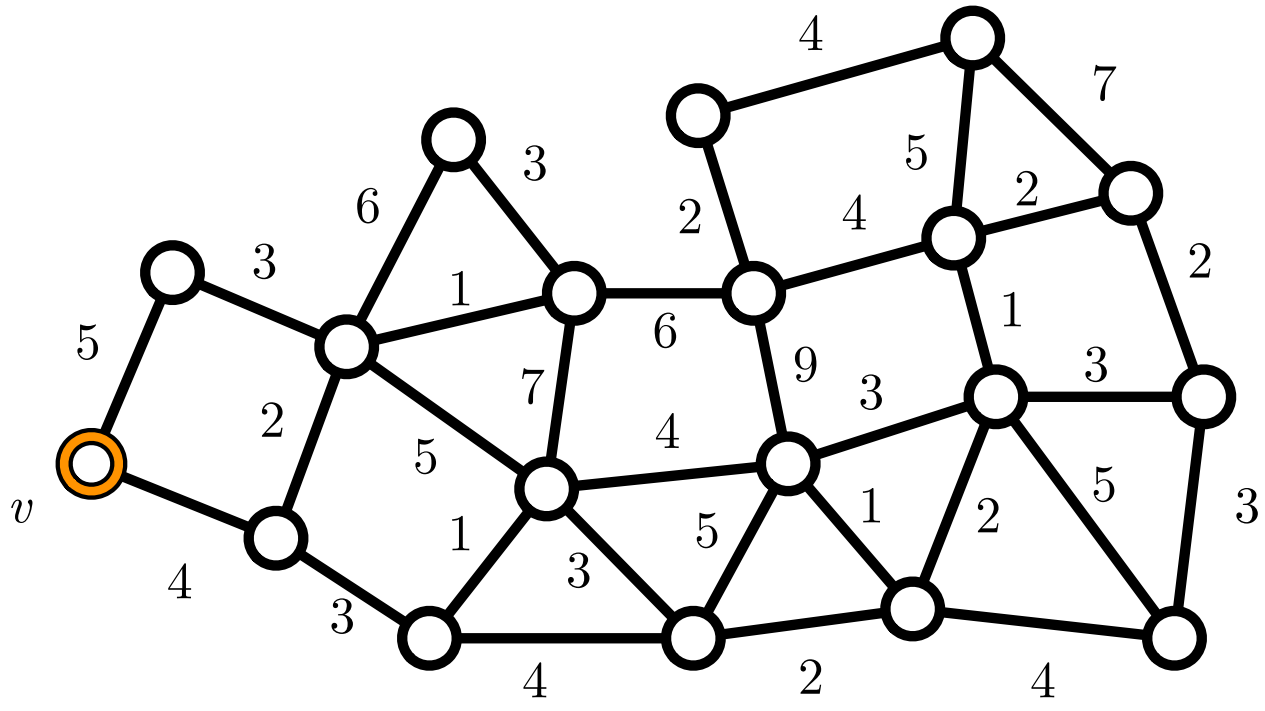
Janne H. Korhonen
Aalto University
Spring 2018

Lecture 1:

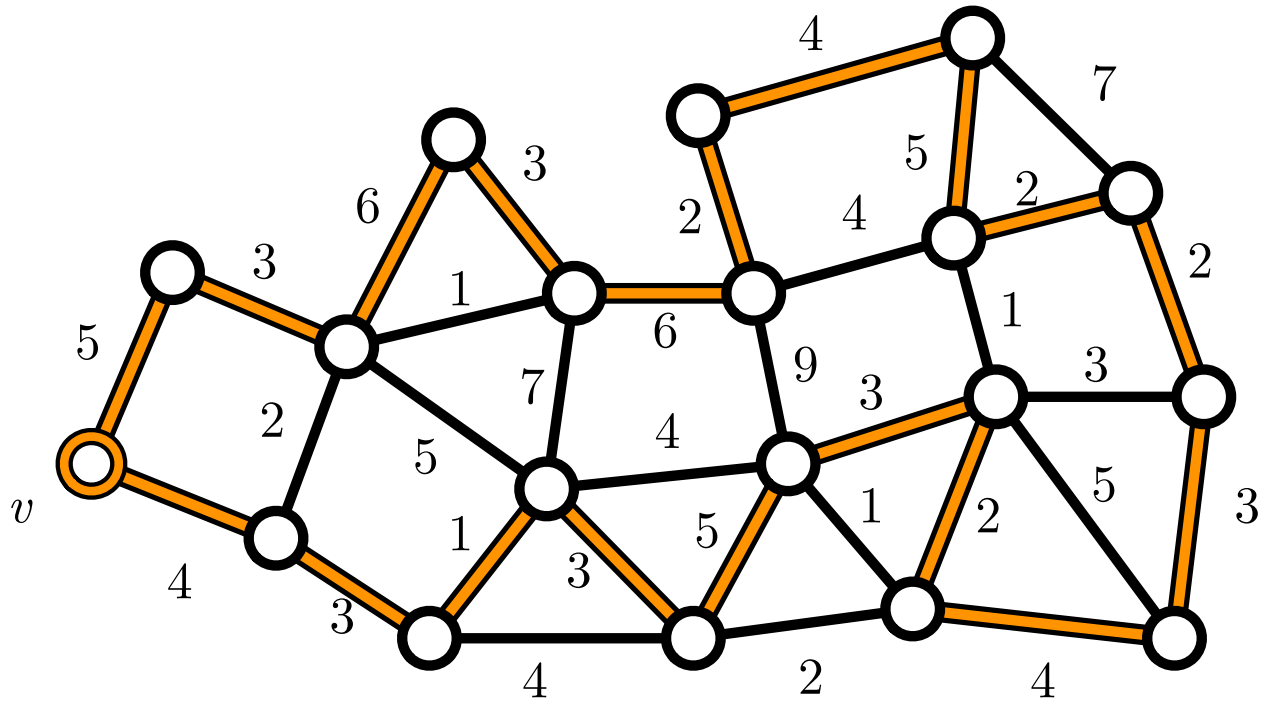
Introduction and Preliminaries



- **Given:** Graph $G = (V, E)$, a vertex v



- **Given:** Graph $G = (V, E)$, a vertex v
- **Find:** minimum-weight tour starting at v that visits all other vertices



- **Given:** Graph $G = (V, E)$, a vertex v
- **Find:** minimum-weight tour starting at v that visits all other vertices

Travelling Salesman Problem

- **Naive algorithm:** try all possible permutations
  - $O(n!)$ tours, not very practical
  - $19! = 121\,645\,100\,408\,832\,000$
- **No polynomial-time solution is known**
  - Conjecture: no such algorithm exists
  - equivalent to the more general **P vs. NP** problem

Computational Complexity

- **Point of this course is to ask questions like:**
  - What is the *best algorithm* for this problem?
  - How much *time* and *memory* do I need to solve this problem?
  - Why is there (probably) no *polynomial-time* algorithm?
  - Can I still solve the problem even though there is no polynomial-time algorithm?

Computational Complexity

- **First, we need to ask other questions:**
  - What is an **algorithm**?
  - What is a **computational problem**?
  - How to measure **computational complexity**?

Computational Complexity

- **This course: mathematical analysis**
- **We are not interested in the effects of:**
  - Programming languages
  - Computer architecture
  - . . .
- **Our perspective: asymptotic analysis**
  - Focus on scaling as input grows
  - Ignores various constant factors

Practicalities

Practicalities

- See **MyCourses**
- Exercises every week, starting **this week**!
- **Reference textbook:**
  - S. Arora and B. Barak. *Computational Complexity: A Modern Approach*
  - http://theory.cs.princeton.edu/complexity/

Schedule

- **Weekly workload**
  - **Lectures:** **Monday**, **Wednesday** 10-12
  - **Exercise sessions:** **Tuesday** 10-12 (B337)
  - **Exercise deadline:** **Friday** at 23:59
  - about **4-5** hours of independent work

- **Bonus exercises**
  - return at any time before second midterm

Grading

- **Two midterms:** pass/fail
- **Weekly exercises:** maximum **86** points (+ bonus)
  - **grade 1/5:** pass exams
  - **grade 5/5:** pass exams + **64** points
- **First exercise set worth 14 points**

Course Discussions

- **Slack workspace for discussions**
  - chat with other students
  - get help with exercises

- https://aalto-cct.slack.com
- **Sign up with your aalto.fi email**

Course Overview and Learning Objectives

Learning Objectives 1

“You can formulate a concrete real-world computational task as a formal computational problem.”

Key Concepts 1:

Models

- **Models of computation**
  - **Turing Machines** (det./nondet./rand.)
  - **Register machines** (word RAM, real RAM)
  - **Circuits** (later in the course)
- **Complexity measures**
  - **Time** and **Space**

Key Concepts 1:

Models

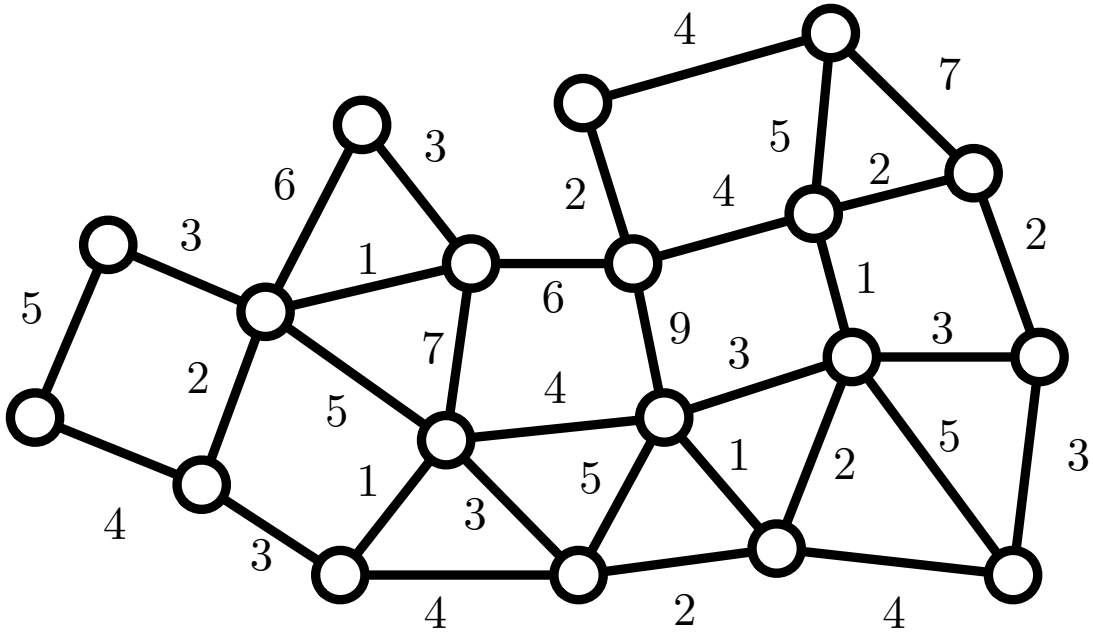
- **Formal definition of a computational problem**
  - *Problem* vs. *Instance* vs. *Algorithm*
- **Computational problem:**
  - Infinite set of **instances**
  - An associated **question** we want to solve

Example:

Computational Problem

Travelling Salesman Problem

- **Instance:** Graph $G = (V, E)$ with positive edge weights
- **Question:** Find a minimum-weight tour that visits all vertices exactly once, or decide that no such tour exists



Key Concepts 1:

Problems vs. Algorithms

- **Complexity of an algorithm**
  - **Upper bounds:** time/space guarantees
  - **Lower bounds:** worst-case behaviour
- **Complexity of a problem**
  - Roughly: **complexity of the best algorithm**
  - **Upper bounds:** algorithms
  - **Lower bounds:** proving something about *all* algorithms

Learning Objectives 2

“You can formally reason about the computational complexity of a problem in comparison with well-known problems and complexity classes.”

Key Concepts 2:

Complexity Classes

- **Complexity class = set of problems**
  - defined in terms of computational complexity

- **Polynomial time:** class **P**
  - Problems with polynomial-time algorithms
  - Problems in P: **tractable**
  - Problems not in P: **intractable**

- **Non-deterministic polynomial time:** class **NP**
- **Other classes:** **LOGSPACE**, **PSPACE**, **EXPTIME**

Key Concepts 2:

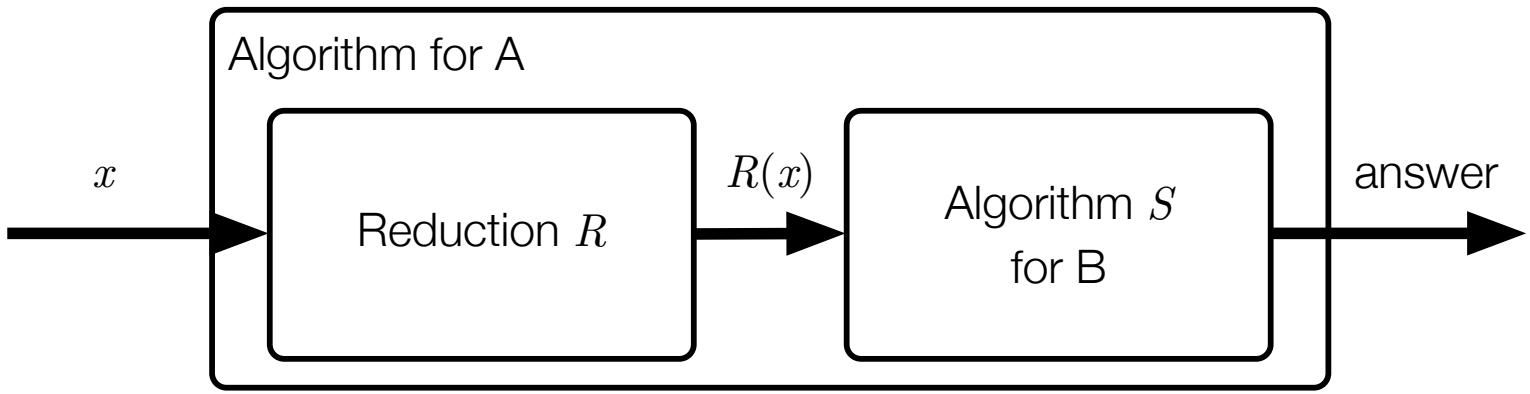
Reductions

- **Reduction R from problem A to problem B:**
  - an algorithm that transform an instance x of problem A to an equivalent instance $R(x)$ of problem B
- **Relates the complexities of problems A and B**
  - Technical requirement: **efficiency**
  - Different notions of reduction

Key Concepts 2:

Reductions

- **Assume we have:**
  - Efficient reduction R from A to B
  - Efficient algorithm S for B
- **Then we can solve A as follows:**
  - Transform instance x of A into an instance $R(x)$ of B
  - Solve $R(x)$ using the algorithm S



Key Concepts 2:

Reductions

- **Efficient reduction from A to B implies B is more difficult than B (““A $\leq$ B”)**
- **Complexity-theoretic tool:**
  - Reduce a difficult problem A to problem B
  - This implies that B is also difficult
- **Algorithm design technique:**
  - Reduce A into an easy problem B
  - This implies that A is also easy

Learning Objectives 3

“You can use reductions and other complexity-theoretic concepts to identify which algorithmic tools – such as randomisation, approximation, parameterisation and parallelism – can and cannot help with the efficient solution of a given problem.”

Key Concepts 3

- **Meta-computational techniques and associated complexity classes**
  - randomisation
  - approximation
  - parallelisation and circuit complexity
  - parameterisation and structural complexity
- **Tools for understanding and dealing with computational complexity**

Preliminaries

Set Notations

- **Naturals:** $\mathbb{N} = \{0, 1, 2, \dots\}$
- **Integers:** $\mathbb{Z} = \{\cdots, -2, -1, 0, 1, 2, \dots\}$
- **Positive reals:** $\mathbb{R}^+ = \{x \in \mathbb{R} : x \geq 0\}$

- **Power set:** $2^X = \{Y : Y \subseteq X\}$ for any set X
- **Product set:** $X \times Y = \{(x, y) : x \in X, y \in Y\}$
- **Iterated product:** $X^k = \underbrace{X \times X \times \cdots \times X}_{k \text{ times}}$

Strings

- **Alphabet:** a set of symbols S
- **String:** a sequence $x = x_1 x_2 \ldots x_k$ of symbols from S
  - Example: $S = \{0, 1\}$, $x = 0101101$
  - ϵ = empty string of length 0

- S^k = all strings of length k over alphabet S
- S^* = all strings over alphabet S
  - Example: $\{0, 1\}^*$ = all binary strings

- $|x|$ = length of the string x
- x^k = x concatenated with itself k times
  - Example: $1^7 = 1111111$, $(01)^3 = 010101$

Asymptotic Notation

- Let $f\colon \mathbb{N} \to \mathbb{R}^+$ and $g\colon \mathbb{N} \to \mathbb{R}^+$ be functions

f grows at most as fast as g

- $f(n) = O(g(n))$, if there exist $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, it holds $f(n) \leq c \cdot g(n)$.

f grows at least as fast as g

- $f(n) = \Omega(g(n))$, if there exist $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0$, it holds $f(n) \geq c \cdot g(n)$.

f grows exactly as fast as g

- $f(n) = \Theta(g(n))$, if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Asymptotic Notation

- Let $f\colon \mathbb{N} \to \mathbb{R}^+$ and $g\colon \mathbb{N} \to \mathbb{R}^+$ be functions

f grows strictly slower than g

- $f(n) = o(g(n))$, if for any $c > 0$ there exists $n_c > 0$ such that for all $n \geq n_c$, it holds $f(n) < c \cdot g(n)$.

Asymptotic Notation

- Let $f : \mathbb{N} \to \mathbb{R}^+$ and $g : \mathbb{N} \to \mathbb{R}^+$ be functions

- $f(n) = O(g(n))$: f **grows at most as fast as** g
- $f(n) = \Omega(g(n))$: f **grows at least as fast as** g
- $f(n) = \Theta(g(n))$: f **grows exactly as fast as** g
- $f(n) = o(g(n))$: f **grows strictly slower than** g

Asymptotic Notation Abuse

- **We usually write things like** $n^2 = O(n^3)$
  - Meaning: For functions $f(n) = n^2$ and $g(n^3)$, we have $f(n) = O(g(n))$

- **Alternative definition:**
  - $O(g) = \{f : N \to \mathbb{R}^+ : f(n) = O(g(n))\}$
  - $O(g)$ is the set of functions that grow at most as fast as g
  - Then it would be correct to write $f \in O(g)$
  - Allows us to write things like

$$n^{O(g(n))} = \left\{h : \mathbb{N} \to \mathbb{R}^+ : h(n) = n^{f(n)} \text{ for some } f \in O(g)\right\}$$

Common Growth Rates

- **Constant:** $O(1)$
- **Linear:** $O(n)$
- **Quadratic:** $O(n^2)$
- **Cubic:** $O(n^3)$
- **Polynomial:** $O(n^d)$ for some constant $d > 0$
- **Exponential:** $O(c^n)$ for some constant $c > 1$

Lecture 1: Summary

- Distinction between algorithm, problem and instance
- The idea of reduction
- Asymptotic notations