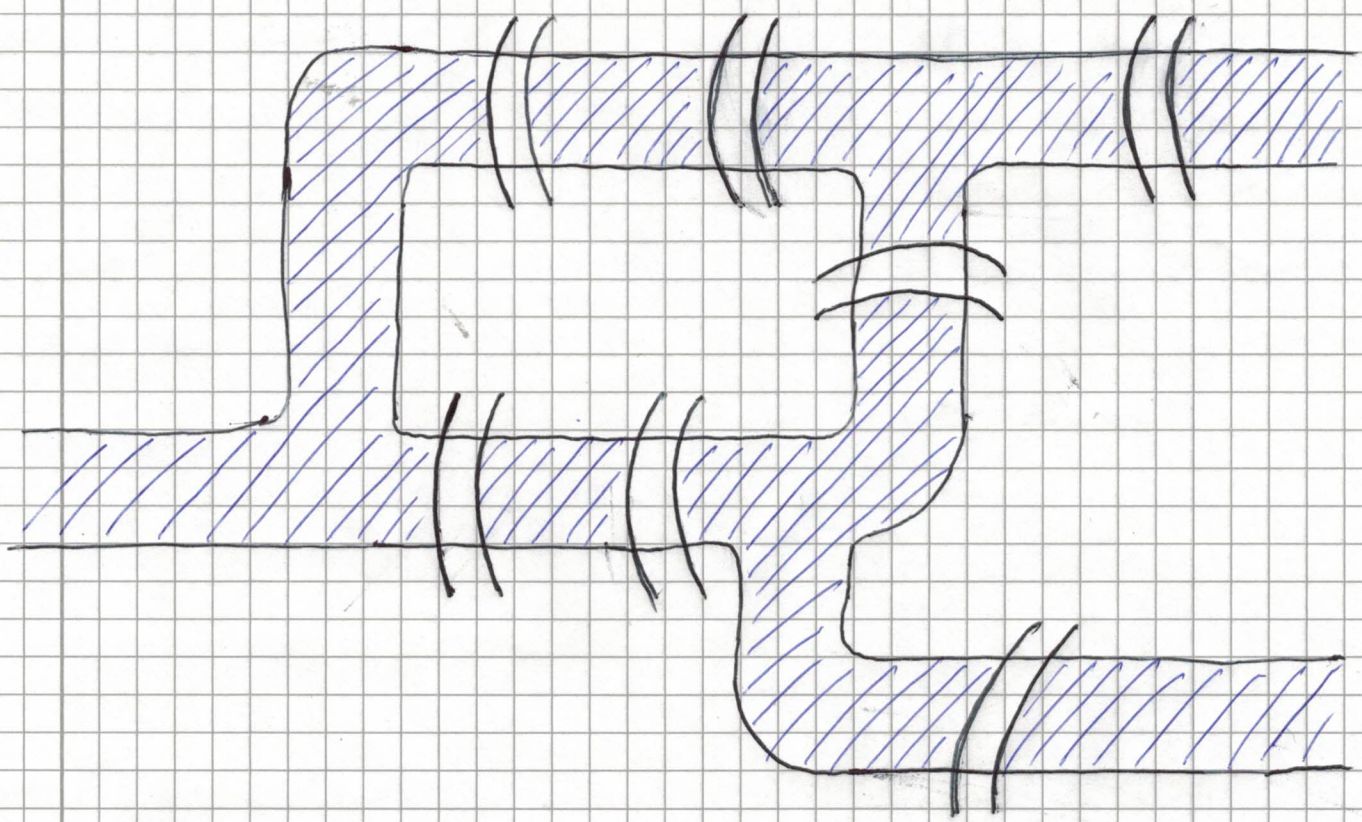
7 bridges of Königsberg (now Kaliningrad)

Problem solved by Euler on 1736

Laid the foundations for graph theory

(central topic in algorithms)



PROBLEM:

Walk across all bridges exactly once

and return to starting position

ALGORITHM DESIGN & ANALYSIS

- Model problem
- Design algorithms
- Prove correctness
- Analyze complexity ⇐

COMPLEXITY MEASURES

- running time ⇐
- memory usage
- parallelism
- communication
- et cetera

COMPUTATIONAL MODEL

Give full math details later

For now:

- standard computer
- your favourite programming language (Java, Python, C++)

Math model (very) robust to such details

Let's model problem and design algorithms to analyze ☺

Only BRIDGES in Königsberg matter

Graphs $G = (V, E)$

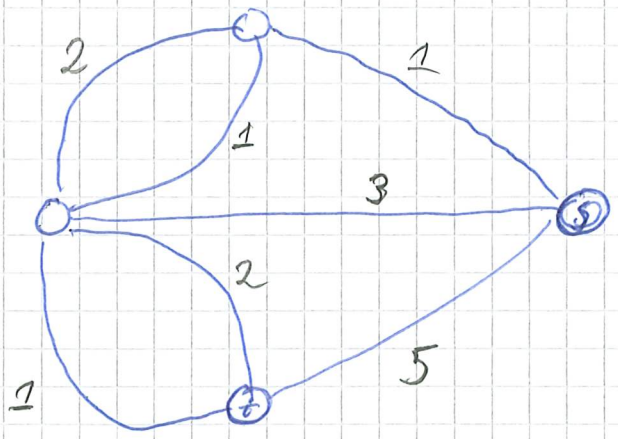
Vertices v

Edges (between pairs of vertices) (u, v)

Extensions:

Edges can have weights

(how long does it take to travel over the bridge, say)



CYCLE: Walk along edges returning to start vertex

Computational problems given $G = (V, E)$ as input

EULERIAN CYCLE: Is there cycle using every edge once?

ECP

HAMILTONIAN CYCLE: Is there cycle visiting every vertex ~~once~~?

HCP

TRAVELING SALESMAN PROBLEM: Cycle visiting every vertex one of length $\leq L$ (say $L = 7$ here)

TSP

SHORTEST PATH PROBLEM: Path from s to t of length at most L' (say, $L' = 3$ here)

SPP

EULERIAN CYCLE ALGORITHM

Observation: For every vertex, edges (bridges) are used in pairs incoming – outgoing.

⇒ After finished cycle, must have even # edges incident to each vertex

(Can prove this is also sufficient)

Algorithm 1 Given graph $G(V, E)$

For every vertex
check if # incident even
If so answer "yes", else answer "no"

(Informal pseudo-code — can be translated to formal program easily)

The other problems ~~are about~~ focus on vertices

Generic algorithm 2

Given graph $G = (V, E)$

For every possible ordering $V = v_1, v_2, \ldots, v_n$

HCP: check if edges (v_i, v_{i+1}) and (v_n, v_1) exist
TSP: Calculate also total length of cycle
SPP: Calculate length of subpath $s \rightsquigarrow t$

Answer "yes" if conditions ever met,
otherwise "no"

COMPLEXITY ANALYSIS

Measure # "basic operations"

What does "basic operation" mean? More details soon

Here for instance:

- add two numbers
- check if edge (u,v) exists

What is a good (i.e., fast) algorithm?

Both our algorithms will be fast on our small example graph. Will take much longer if we run them on, say, the Facebook friend graph — but then the input is also so much larger!

KEY INSIGHT Measure how running time / # operations scales with input size

Suppose input size doubles — then reasonable that running time at least doubles, since we need to read the input

If running time increases by at most constant factor K, then algorithm **EFFICIENT**

Equivalently: Algorithm is efficient if for input size n running time scales like polynomial n^k

$K = 2$	linear in n	
$K = 4$	$\sim n^2$	quadratic
$K = 8$	$\sim n^3$	cubic

COMPLEXITY CLASS P

All problems that can be solved in polynomial time $\approx$ "EFFICIENTLY SOLVABLE" TRACTABLE

scaling like n^k for some constant k

ECP $\in$ P

COMPLEXITY CLASS EXP

All problems solvable in time scaling like 2^{n^k} for some constant k

Not efficient — Problems not solvable in poly time INTRACTABLE

Also many other complexity classes

EULERIAN CYCLE ALGORITHM 1

Runs in time $\sim |V| + |E|$ = linear

GENERIC ALGORITHM 2

For $|V| = n$, runs in time something like

$\sim n \cdot n! \approx 2^{n \log n}$

Even for Facebook graph restricted to Sweden or Denmark, even if every atom in known universe is modern super computer running since beginning of time 13.7 Gillion years ago, will be nowhere close to finish before the sun dies

COMPUTATIONAL COMPLEXITY THEORY

Analyze computational problems
Pin down how hard or easy they are
= how fast can the best algorithm be?

Looking back at our problems

Eulerian cycle $\in P$

Shortest path $\in P$ because there is a better algorithm

Dijkstra's algorithm:
Set known distance for $s = 0$
other vertices $= \infty$
Pick vertex v with smallest distance $d(v)$
For all neighbours u, update $d(u)$ to $d(v)$ + length of edge (v,u)

SKIP DETAILS FOR NOW

Runs in linear time Why linear time? And why is the algorithm correct?

Hamiltonian cycle problem
Travelling salesman problem

Not known! ① Many believe exponential time necessary

LEARN MORE about this at AADS & CoCo courses

② Proving $\notin P$ is one of the MILLENNIUM PRIZE PROBLEMS with a 1 MUSD award.

③ Know answer will be the same for HCP, TSP, and many other problems

INTRODUCTION TO DISCRETE MATHEMATICS AND ALGORITHMS (IDMA)

A course about

DISCRETE MATH

- numbers, logic, graphs, combinations
- NOT continuous functions, derivatives, integrals et cetera

Mathematics

- Start from obvious facts (axioms)
- Derive true facts from these axioms
- We won't give fully axiomatic treatment (far from it), but will stress rigorous reasoning
- In mathematics, we derive absolute truth – very different from other fields of science, where there are hypotheses and experiments

ALGORITHMS

- Our computational problems have concrete, correct answers (not about ML predictions)
- Algorithms are precise descriptions of automated methods for solving such problems (focusing on discrete objects)
- Can be coded up in different programming languages (but we won't do programming in this course)

PRO TIP

- Read course material before lectures
- Maybe even watch online videos for challenging topics

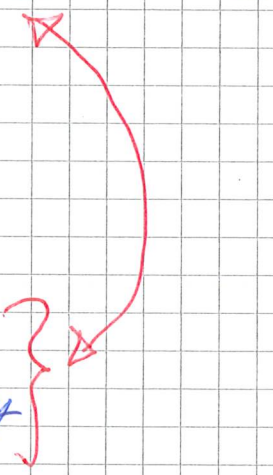
COURSE CONTENTS

MATHEMATICS

- definitions of important concepts
- theorems
- proofs
- develop your abilities for rigorous (but creative!) reasoning

ALGORITHMS

- (pseudo) code
- proofs of correctness
- analysis of efficiency / complexity



LECTURER

JAKOB NORDSTRÖM, PROFESSOR
ALGORITHMS & COMPLEXITY SECTION
DEPARTMENT OF COMPUTER SCIENCE (DIKU)
Also at Lund University in Sweden

Research on:
- Proving that really hard problems are really hard
- Solving them as fast as possible anyway
- Producing auditable certificates that solutions are correct

LANGUAGE OF TUITION

Officially Danish
But Swedish qualifies as Danish for official purposes
I suggest we continue in English...

WHAT YOU NEED TO DO

- Be on top of courseplanning
- Follow lectures (or digest material on your own)
- Read textbooks before and in between lectures!
  (Lectures do NOT cover everything!
- Work on exercises (use TAs at exercise classes)
- Solve and hand in problem sets — REQUIRED FOR EXAM
- Interact with coursemates — help each other

Do yourself a favour: WORK HARD on this course! Will help a lot in later courses!

PROBLEM SETS

- Help you study ~~throughout the course~~ (instead of panicking the week before the exam)
- Problems look like (and are) exam problems — graded in the same way

Rules (check also Absalon):

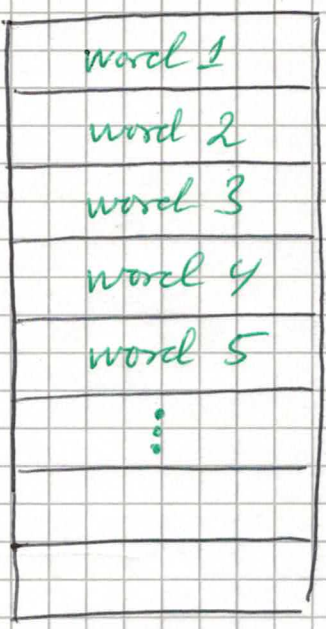
- Collaborate in groups of 2-3
- But write up everything individually!
- NO COPYING OR SHARING of text, ~~code, formulas~~
  - from each other
  - from the Internet

PLAGIARISM WILL BE REPORTED

- Submit on time on Absalon as PDF file
- Write up in LaTeX (or other math-aware system)
- Graded after one week
- Corrections needed? One week for resubmission

QUESTIONS? Ask (a) now or (b) TAs or (c) on Absalon

RAM MODEL



How we think of computer

Big chunk of MEMORY partitioned into WORDS

In one time unit, we can
- read word from memory
- store word in memory
- perform operation

WORD

A number (w) of bits (0s & 1s)

We assume "basic objects" fit in a word
- characters
- integers (not ridiculously large)
- pointers (to other location in memory)

What does NOT fit in a word:
- arrays / tables
- lists
- graphs

OPERATIONS

Assume following operations take constant time
- arithmetic +, −, *, /
- Boolean operations and, or, not
- Bitwise operations & (bitwise and)
  | (bitwise or)
  ~ (bitwise not)

This is a MODEL, not a description of reality

But good enough for our purposes

PSEUDO-CODE

Not actual code
But "close enough" that you can easily code it up in your favourite programming language

- Focus on essential details
- Ignore irrelevant syntax details
- Assume that "obvious"/"reasonable" methods exist in some (pseudo-) standard library

What is important?
- NOT the exact syntax
- But that we get a crisp, clear, unambiguous description of the algorithm

COMPLEXITY ANALYSIS

Given algorithm A solving some task
Given input of size n
How many steps does algorithm need for input of size n?

Count:
- Reading, assigning, storing
- arithmetic operations
- comparisons
- program flow constructs — if-then-else, for loops
- method/function calls

What time complexity ?

- Best case
- Worst case
- Average-case
- "Real-case"

Focus on worst case (in this course)

Often most relevant for the basic algorithms we will study

But certainly not the only way to measure!

Simplifying assumptions

All operations take the same time

Only care about highest-order term

Example

Algorithm A running time $n^2 - 4n + 3$

Algorithm B — " — $1000n + 1000000$

A scales like $\sim n^2$

B scales like $\sim n$

So B will become (much) better as input size grows

In reality factor 1000 is of course important

But this is an introductory course — focus on the fundamentals. And linear-time algorithm will quickly beat quadratic-time algorithm in practice