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 $G = (V, E)$

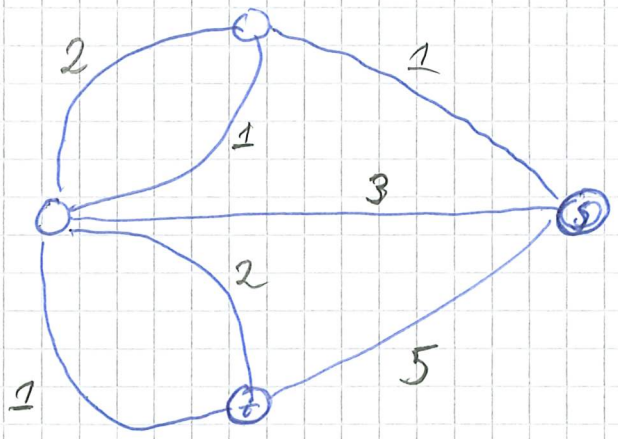
Vertices v

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