ALGORITHM DESIGN AND ANALYSIS

- Model problem mathematically
- Design algorithm (method) to solve the problem
- Prove correctness
- Analyze complexity – is the algorithm efficient?

COMPLEXITY MEASURES

- Running time ⟸ MAIN FOCUS IN THIS COURSE
- Memory usage
- Parallelism
- Energy consumption
- Et cetera

COMPUTATIONAL MODEL

To do rigorous mathematical analysis of algorithms, need to give a precise definition of what we mean by "computer"

In this introductory course, think of
- standard laptop
- your favourite programming language (Python, Java, C, C++)

The actual mathematical model (which we won't talk about in this course) is very robust to variations of details

Let's model the problem and design an algorithm we can analyze!

Observation: Only BRIDGES in Königsberg matter

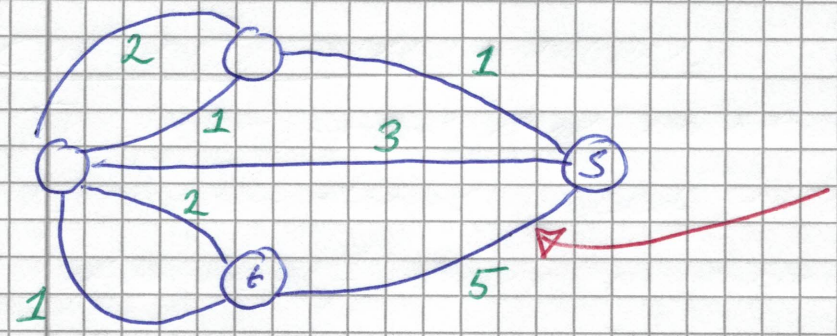
Model as GRAPH $G = (V, E)$

V: vertices

E: edges / arcs between pairs of vertices

Extension: Edges can have weights (what is the length of the bridge, say)

CYCLE: Walk along edges returning to start vertex



COMPUTATIONAL PROBLEMS given $G = (V, E)$ as input

EULERIAN CYCLE — Is there a cycle using every (exactly) edge once?

HAMILTONIAN CYCLE — Is there a cycle visiting every vertex exactly once?

TRAVELLING SALESMAN PROBLEM (TSP) — What is the shortest cycle visiting every vertex exactly once?

SHORTEST PATH PROBLEM — What is the shortest route from s to t?

EULERIAN CYCLE ALGORITHM

OBSERVATION For every vertex, edges (bridges) are used in pairs incoming – outgoing

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

Can prove this is also sufficient

Algorithm 1:

```
EULER (G)
for every vertex v ∈ V
    if # edges incident to v odd
        return "no"
return "yes"
```

This is pseudocode – not real program, but can be translated easily to your favourite programming language

HAMILTONIAN CYCLE ALGORITHM

Simple idea: Try all possible orderings (permutations) of vertices in V and check if cycle

Algorithm 2:

```
HAMILTON (G)
for every ordering O = {v_1, v_2, ..., v_n} of V
    is_cycle := TRUE
    for i := 1 upto n-1
        if (v_i, v_{i+1}) not edge
            is_cycle := FALSE
    if (v_n, v_1) not edge
        is_cycle := FALSE
    if is_cycle return "yes"
return "no"
```

TRAVELING SALESMAN ALGORITHM

Can use similar idea for TSP

Algorithm 3

```
TSP(G)
best_tour := ∞
for every ordering O = {v_1, ..., v_n} of V
    curr_tour := 0
    is_cycle := TRUE
    for i := 1 upto n-1
        if (v_i, v_{i+1}) not edge
            is_cycle := FALSE
        else
            curr_tour := curr_tour + weight(v_i, v_{i+1})
    if (v_n, v_1) not edge
        is_cycle := FALSE
    else
        curr_tour := curr_tour + weight(v_n, v_1)
    if (is_cycle and curr_tour < best_tour)
        best_tour := curr_tour
return best_tour
```

SHORTEST PATH ALGORITHM

Could use similar idea as for TSP, but only start counting from vertex s and stop at vertex t.

COMPLEXITY ANALYSIS

Measure running time by counting # basic operations

For instance:

- adding two numbers
- checking if edge (u, v) exists in graph G

What is an efficient (i.e., fast) algorithm?

Depends on the size of the input

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

If input size doubles, then reasonable that running time doubles — we need to read twice as much input

If running time increases by at most constant factor K when input size doubles, then algorithm EFFICIENT

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

$K = 2$	polynomial n	— linear in n
$K = 4$	polynomial n^2	— quadratic
$K = 8$	polynomial n^3	— cubic

Are our algorithms above efficient?

Eulerian cycle

Just scan input — linear time

Often best we can hope for (in this course) ✓

Hamiltonian cycle and TSP

For graph with $|V| = n$ vertices, have to consider all

$$n! = n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$$

possible permutations

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

Even worse than exponential time!

For just a thousand vertices, won't get results before the sun dies... ✗

Is it possible to find efficient algorithm for Hamiltonian cycle or TSP?

NOT KNOWN 🤷

Many believe that there are no algorithms that will always be efficient and give correct answers

But proving this is one of the MILLENNIUM PRIZE PROBLEMS with a 1 MUSD award

What about SHORTEST PATH PROBLEM?

There is an efficient algorithm!

DIJKSTRA'S SHORTEST PATH ALGORITHM

Computes distances from start vertex s to all other vertices

Algorithm 4

```
DIJKSTRA (G, s)
for all v ∈ V
    d(v) := ∞
d(s) := 0
while exists unprocessed vertices
    v := unprocessed vertex with smallest d(v)
    for all neighbours u of v
        update d(u) to d(v) + weight(v,u)
                    if smaller
    mark v as processed
```

CLAIM 1

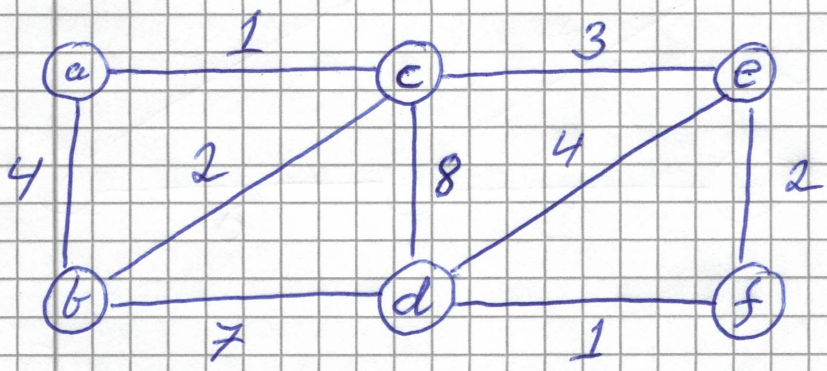
At the end of the algorithm, for all v $d(v)$ is length of shortest path from s to v

CLAIM 2

Dijkstra's algorithm can be implemented to run in nearly linear time

We will prove all this during the last week of the course...

For now, let's see an example dry-run of the algorithm for graph with starting vertex a



DISTANCE ESTIMATE UPDATES

Vertex processed		a	c	b	e	f	d	SHORTEST PATHS FROM a
d(a)	0	0	~	~	~	~	~	0
d(b)	∞	4	3	3	~	~	~	3
d(c)	∞	1	1	~	~	~	~	1
d(d)	∞	∞	9	9	8	7	7	7
d(e)	∞	∞	4	4	4	~	~	4
d(f)	∞	∞	∞	∞	6	6	~	6

Neither correctness nor efficiency is obvious, but we will get back to this