REOP - Session 2 : Shortest paths

I/ Homework

~~Ex 3.2~~ Ex 3.3 Ex 3.10
~~Prop 3.2 ($\Longleftrightarrow$ Ex 3.5)~~ sorry

Ex 3.3

Adjacency matrix $A_{uv} = \begin{cases} 1 & \text{if } \exists \text{ an edge btw } u \text{ \& } v \\ 0 & \text{otherwise} \end{cases}$

$$(A^2)_{uv} = \sum_{w \in V} \underbrace{A_{uw} A_{wv}}_{= 1 \text{ iff both the edges } (u,w) \text{ \& } (w,v) \text{ exist}} = \sum_{w \in V} \mathbb{1}\{\text{the path } (u,w,v) \text{ exists}\}$$

A^2_{uv} = nb of paths of length 2 in G from u to v

The result can be proved by induction

$$A^k_{uv} = (A^{k-1} \cdot A)_{uv} = \sum_w \underbrace{A^{k-1}_{uw}}_{\text{by ind, = nb of } (k-1)\text{-paths } u \rightsquigarrow w} \overbrace{A_{wv}}^{\text{existance of edge } (w,v)}$$

Ex 3.10

S is a stable $\iff$ no edge has both endpoints in S
$\iff$ every edge has at least 1 endpoint in $V \setminus S$
$\iff$ the vertices of $V \setminus S$ cover every edge
$\iff$ $V \setminus S$ is a vertex cover

S is the largest stable $\iff$ $V \setminus S$ is the smallest vertex cover

$$\alpha(G) = |S| = |V| - |V \setminus S| = |V| - \tau(G)$$

II/ Shortest paths

1) Statement

Input : - a graph $G = (V, E)$
- a cost function $c : E \longrightarrow \mathbb{R}$
- two vertices o (origin) & d (destination)

Question : Find an $o \rightsquigarrow d$ path P of minimum cost $c(P) = \sum_{e \in P} c(e)$
(or a proof that none exists)

We denote by $c(v)$ the cost of a shortest $o \rightsquigarrow v$ path

2) Integer Programming Formulation

Reminders

A Linear Program (LP) is an optimization problem with a linear objective & linear constraints

(LP) $\min c^T x \quad \text{s.t.} \quad Ax \leq b$ | $x \in \mathbb{R}^n$ variable, A, b, c are params

A Mixed Integer Linear Program is a Linear Program where some of the variables are constrained to take integer values

$$\min_{x \in \mathbb{R}^{n-p} \times \mathbb{Z}^p} c^T x \quad \text{s.t } Ax \le b$$

Rq: $Ax \le b$ is a vector ineq. It means $\forall i \in [\![1, m]\!],\ (Ax)_i \le b_i$

x has size n
c — n
A — $m \times n$
b — m

n = nb of variables
m = nb of constraints

Writing a MILP doesn't mean writing A, b, c explicitly

$$\min 5x + 2y \quad \text{s.t. } x \in \mathbb{Z},\ y \in \mathbb{R}$$

many constraints: write them separately & symbolically

$$3x + 3y \le 1$$
$$x - y \le 2$$

$c = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$

$A = \begin{pmatrix} 3 & 7 \\ 1 & -1 \end{pmatrix}$ $b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

In theory solving a MILP is (NP-)HARD

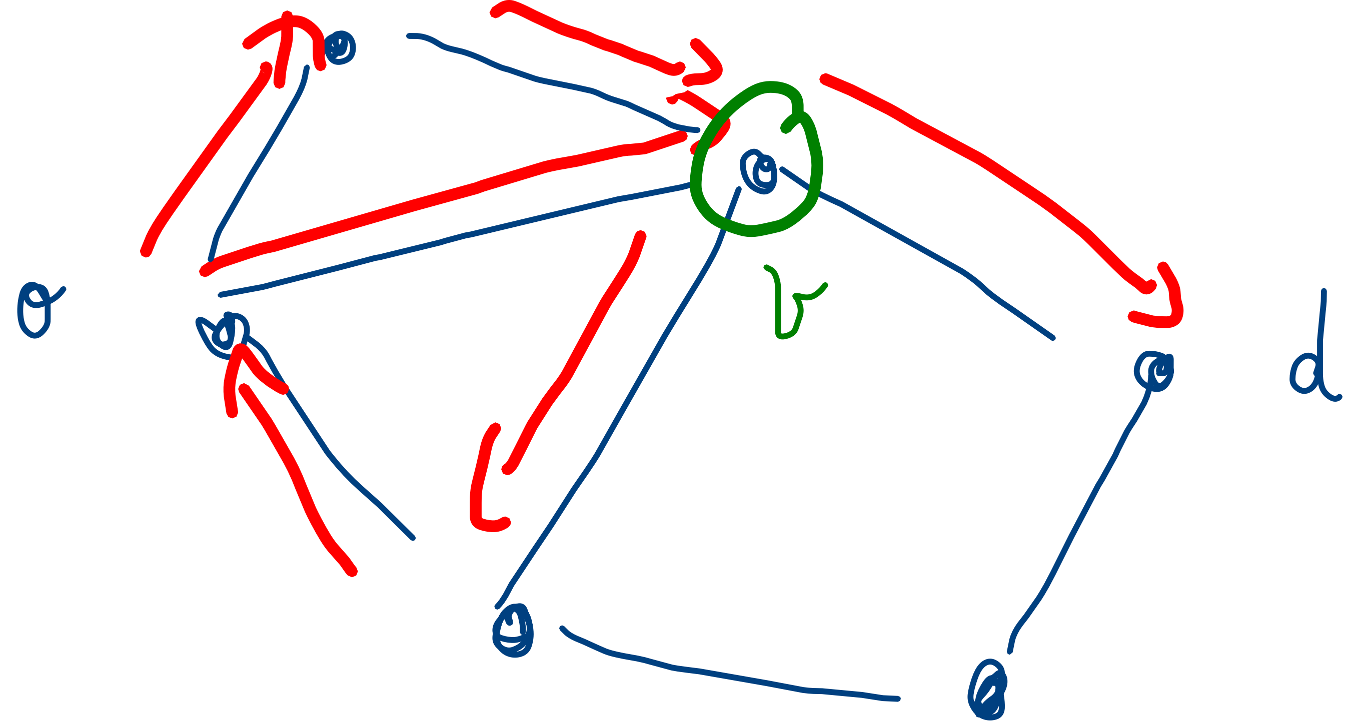
BUT
- they are a really useful modeling tool
- there are very efficient industrial (or open source) solvers

Formulation of "Shortest path" as a MILP

Let x_{uv} be a binary variable $= 1$ iff the edge (u,v) is selected in a path

Objective: $\min \underbrace{\sum_{(u,v)\in E} x_{uv}\, c(u,v)}_{c(P_x)}$

Constraints: $x \in \{0,1\}^E$ "defines an o-d path"



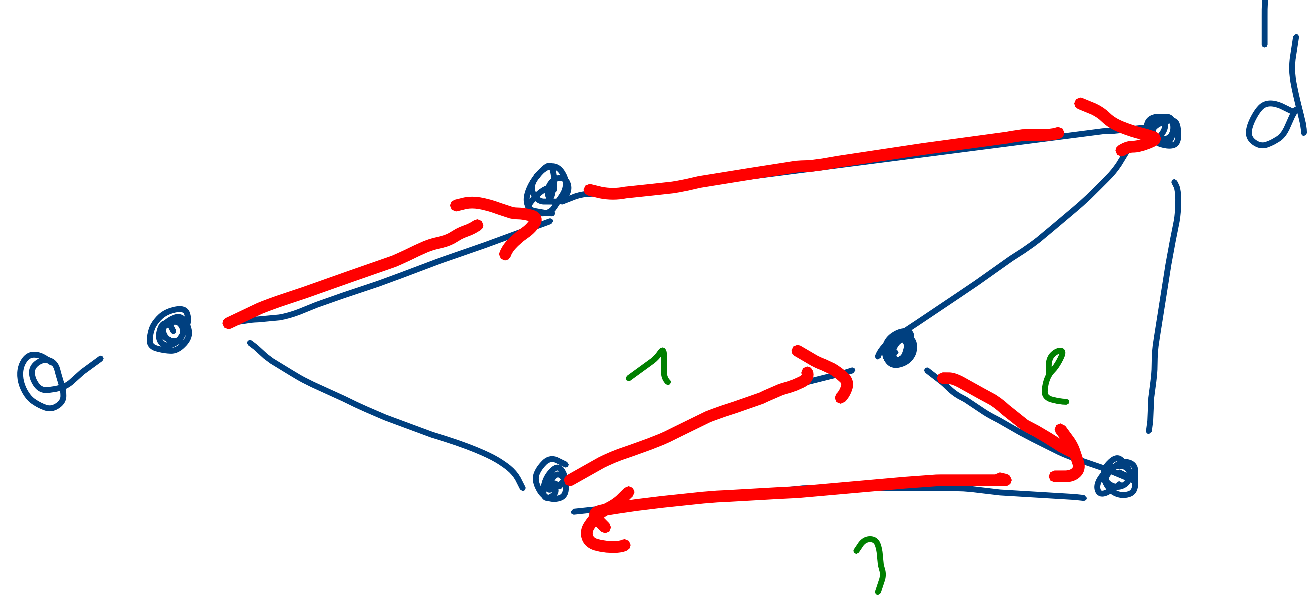
If a vertex $v \notin \{o, d\}$ is crossed, it should have as many edges going in and out

This means $\forall v$, $\sum_{u \in N^-(v)} x_{uv} - \sum_{w \in N^+(v)} x_{vw} = \begin{cases} 0 & \text{if } v \notin \{o,d\} \\ 1 & \text{if } v = d \\ -1 & \text{if } v = o \end{cases}$

($x \in \{0,1\}^{V \times V} \rightarrow$ additional ctr)

Rq: since $x \in \{0,1\}^E$ we are actually modeling a shortest simple path (no repeated edge)

Rq: all the solutions are not paths but (in the nice cases) the optimal solutions are

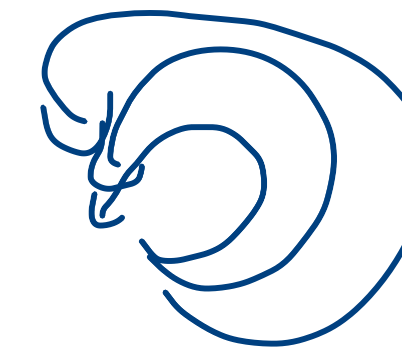


$\varepsilon > 0$ optimal solution has no cycle

3) Complexity

Thm: The shortest path pb is | NP hard in the general case
| polynomial in nice cases

Why? → If there are cycles with < 0 cost in G
then the "shortest" path is not defined

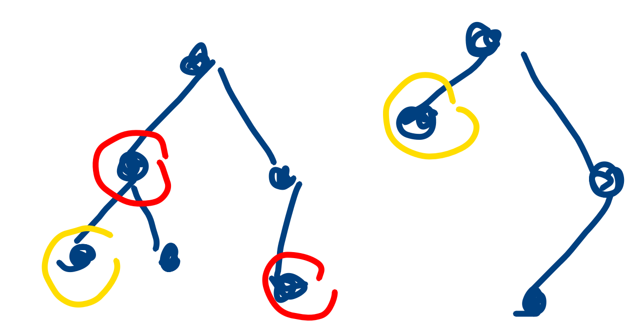


→ Then we look for the shortest simple path
but it is at least as hard as the Hamiltonian path

Nice cases:

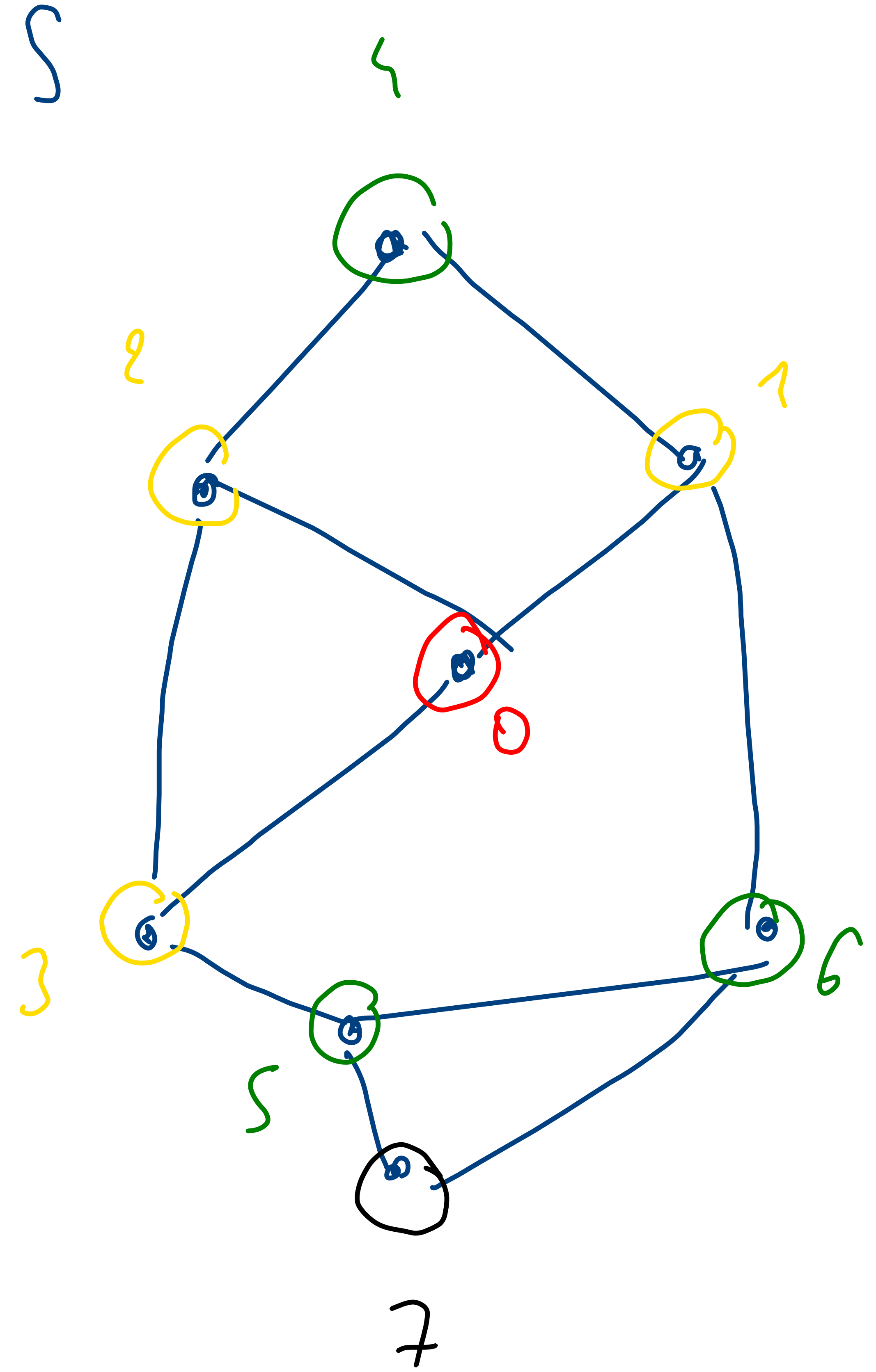
$\neq$ Depth-First

- Unweighted graphs ($c = 1$) → Breadth First Search (BFS)
- Acyclic graphs:
  - Undirected: forest (at most 1 simple path o - d)
  - Directed: topological sort ← DFS

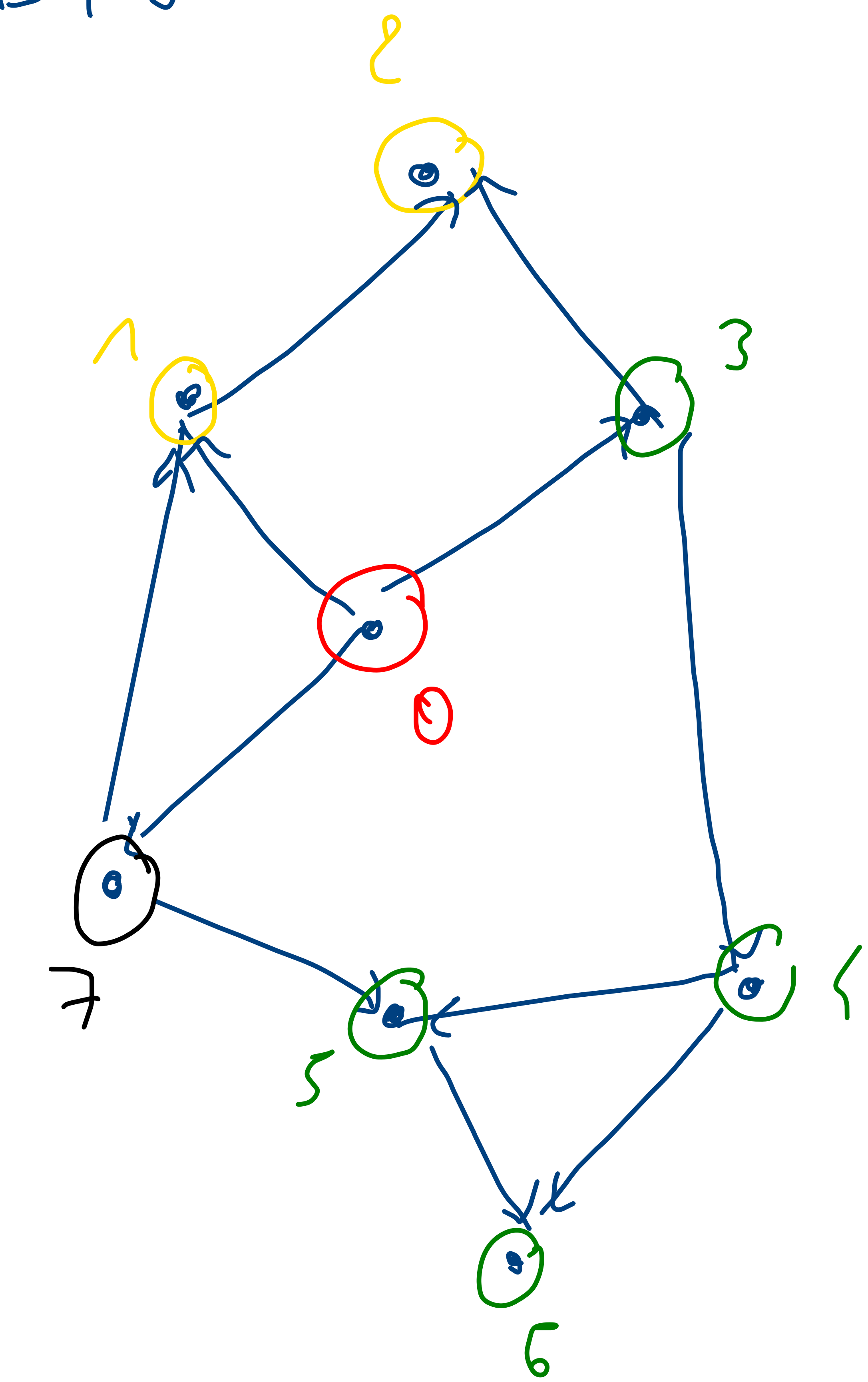


- $c > 0$: Dijkstra's algorithm ← program
- No | negative cycles:
  | absorbing
  - Directed: Bellman-Ford ←
  - Undirected: T-joints (season 2)

BFS



DFS



4) Generalizations

- Single origin, single destination — 1 o, 1 d
- ______________, mult destinations — 1 o, all possible d's
- Multiple origins, ______________ — all pairs (Floyd-Warshall)

Variants:

- Shortest paths with resource constraints (A. Parmentier)
- Multi-criteria
- Transportation networks (timed trips)

III / Algorithms

1) Dynamic Programming

Invented by Bellman. Principle:

A subtrajectory of an optimal trajectory is also optimal

In order to solve a problem, you generalize it. This is done by changing the bounds or varying some constants

2) Bellman - Ford algorithm

Setting: Directed graphs with no cycles of negative cost

We want to compute the cost $c(d)$ of a shortest o-d path

Generalization: we will compute

$c_k(v)$ = the cost of a shortest o-v path with exactly k edges (if it exists)

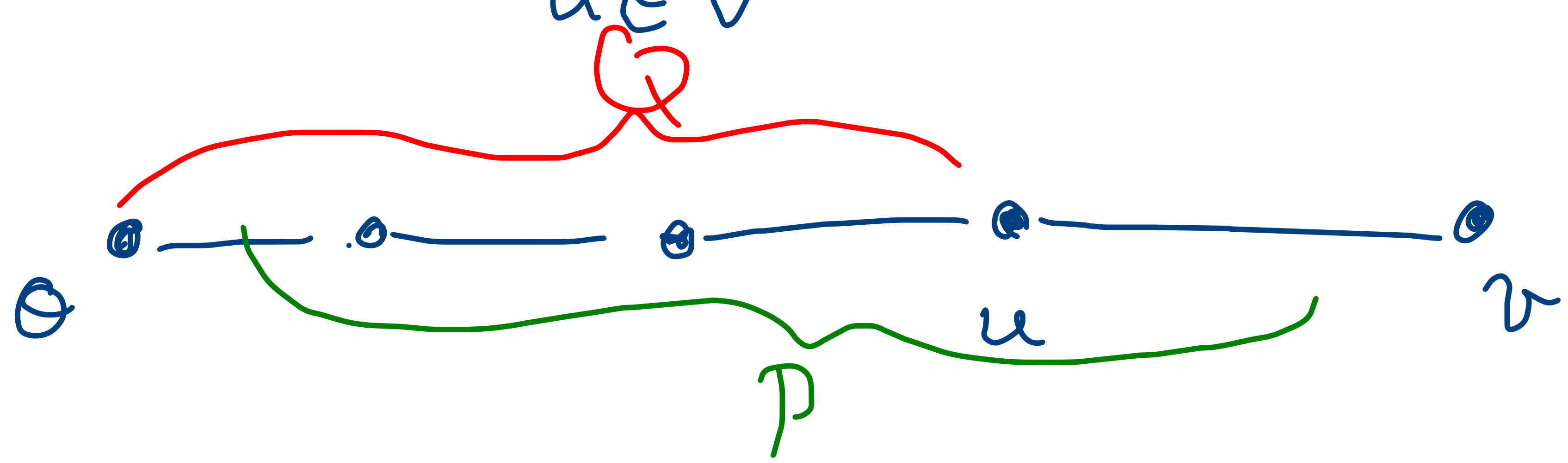
We can retrieve the info we want by taking the minimum of all the $c_k(d)$ values for $k \ldots$ in $[\![1, M]\!]$

Since there are no negative cycles, a shortest path will be acyclic and have at most $n-1$ edges (trees have $\leq n-1$ edges) (there will be at least one such shortest path)

Bellman equation / recursion:

we use Q & extend it by 1

to build P $\quad c_k(v) = \min\limits_{u \in V} \left(c_{k-1}(u) + c(u,v) \right)$ $\quad$ (B)



compute $c_k(v)$

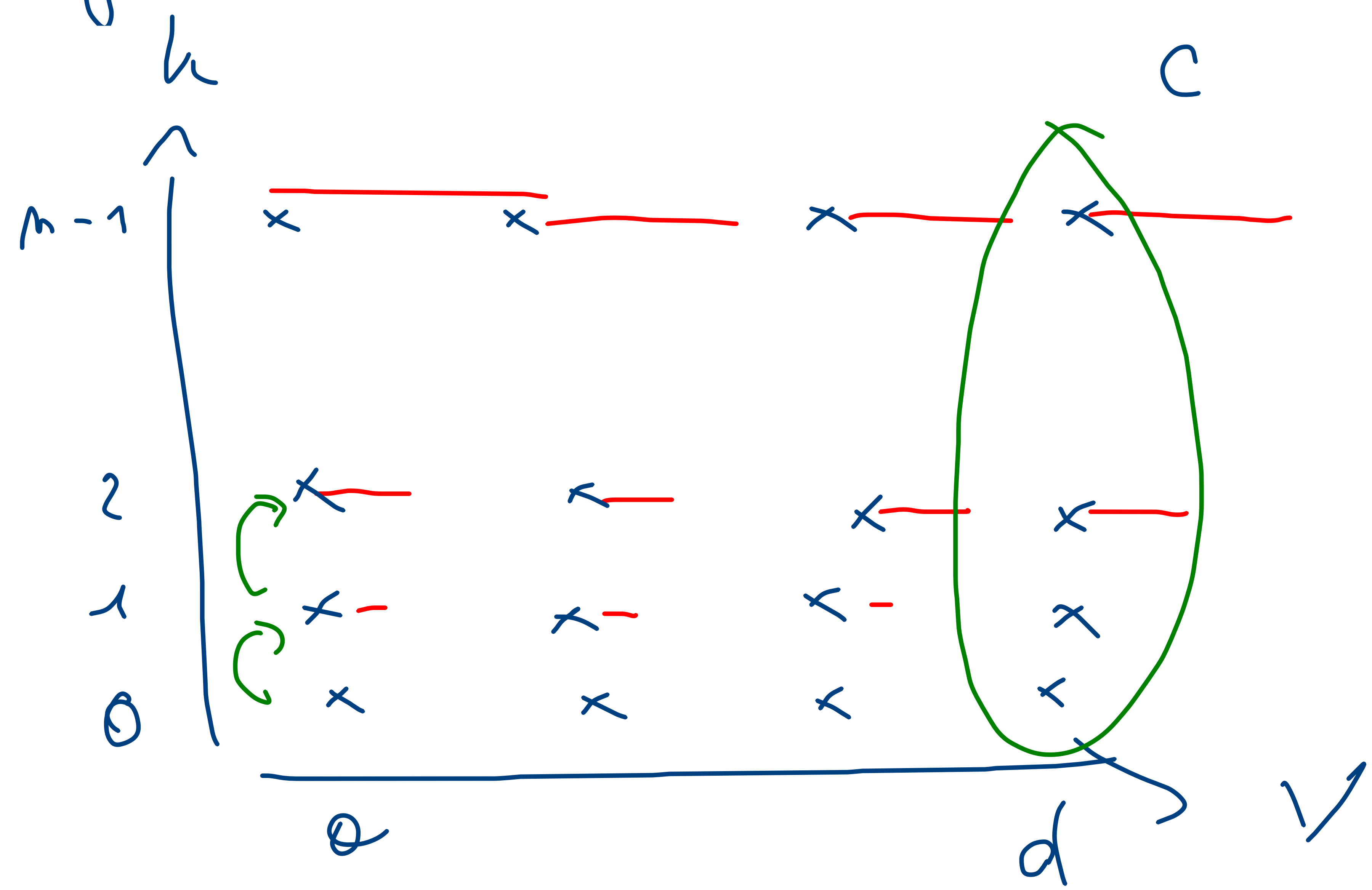
remember $(v,k) \to u$

Why? If P is a shortest k-path from o to v, and if u is the vertex just before v on P, then Q is a shortest $(k-1)$-path from o to u

$$c_0(v) = \begin{cases} 0 & \text{if } v = o \\ \infty & \text{otherwise} \end{cases}$$

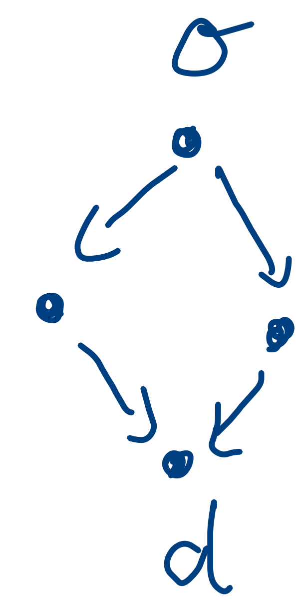
We have the cost, now how do we get the path?
At each iteration of the recursion, store the vertex u that achieves the minimum in (B)
For each (v, k), we know the penultimate vertex on an optimal path u. Then we look it up for $(u, k-1)$ etc etc and we go back to o

Building a matrix



2) Topological sort

Setting: DAGs (Directed Acyclic Graphs)



In a DAG, all paths are elementary (B)

Bellman equation : $c(v) = \min_{\substack{\cancel{u \in V} \\ u \in N^-(v)}} \left(c(u) + c(u,v) \right)$ parents of v

Before computing $c(v)$, we need to have computed $c(u)$ for all its parents

Find an order on the vertices such that children nodes always come after their parents

! This is possible because we are in a DAG using topological sorting (see notes)

Topological sort $\Longleftrightarrow$ Depth-First Search

DFS(v) is a procedure that
- opens v
- scans its children (applies DFS to them)
- closes v

consistent iff no cycles

The order in which nodes are closed is a reverse topological order

3) Dijkstra's algorithm

Setting: $c > 0$ see Algorithm 3

U = set of visited vertices
$d(v)$ = "tentative" distance $\geq c(v)$
↳ updated iteratively until it reaches the true distance

Lemma: For all $u \in U$, $d(u) = c(u)$

Ex 5.8

Evolution of the tentative distance

s	a	b	c	d	e	f	g	h	t	
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	s
0	1	2	3	∞	∞	∞	∞	∞	∞	a
0	1	2	3	∞	∞	3	∞	∞	∞	b
0	1	2	3	∞	7	3	∞	∞	∞	c
0	1	2	3	7	5	3	∞	∞	∞	f
0	1	2	3	7	4	3	6	∞	∞	e
0	1	2	3	6	4	3	6	7	5	t
0	1	2	3	6	4	3	6	7	5	

We only visited vertices v s.t. $c(v) \leq c(d)$

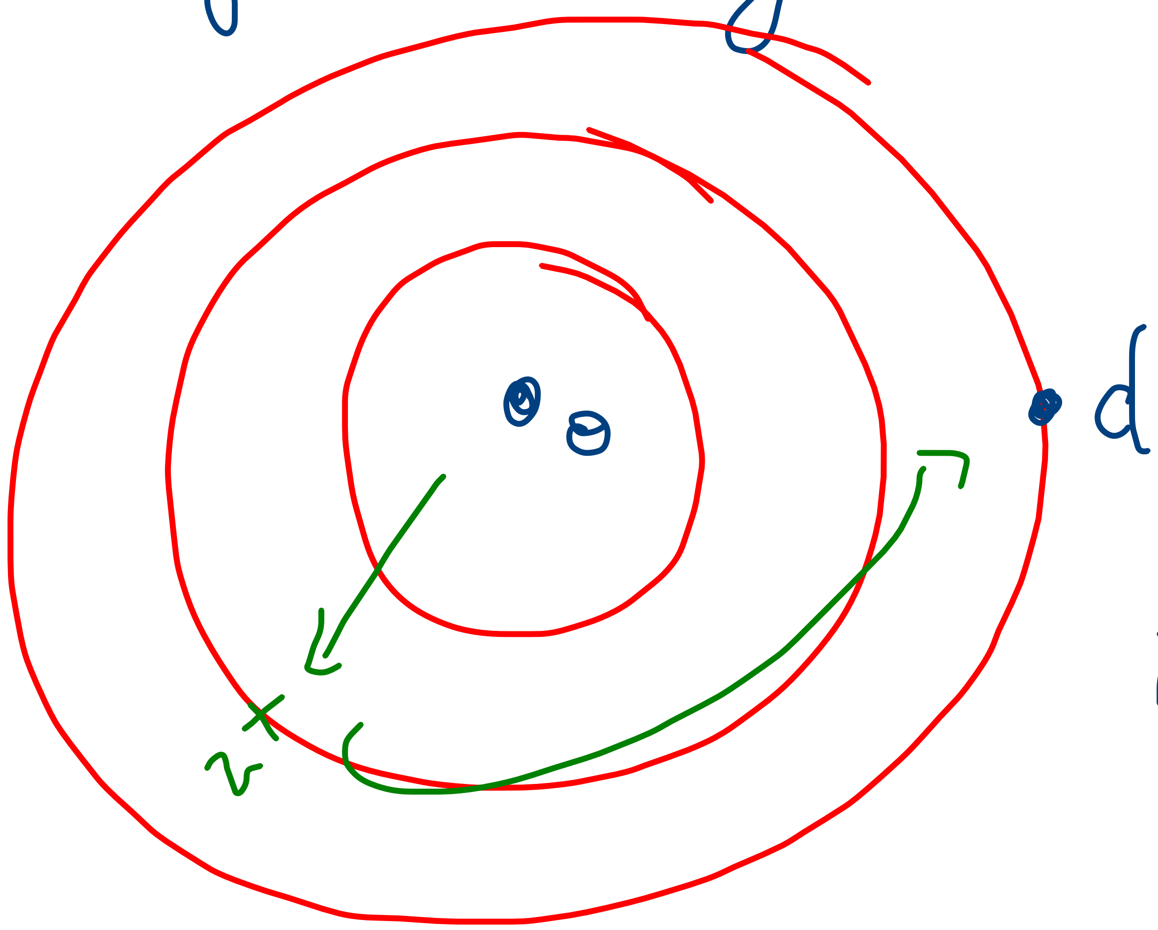
To get the shortest path, same as in Bellman-Ford

line 6 of the code: either update or leave unchanged

when you update, store a pointer $w \longrightarrow v$

"the best path $s \rightsquigarrow v$ we know for now ends with v"

A* algorithm: Dijkstra on steroids



A* alg. guides the search towards d by changing the vertex selection method

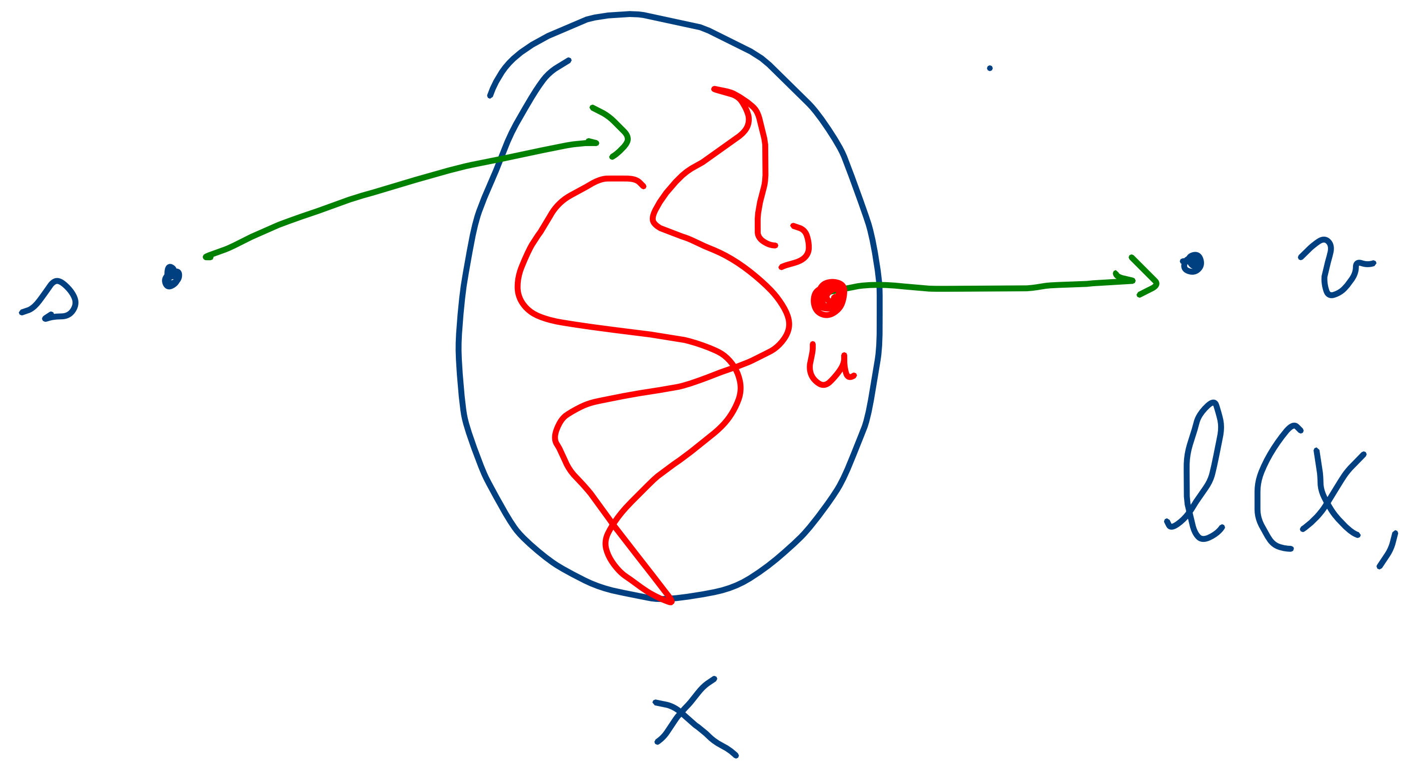
Dijkstra : criterion = $d(v)$

A* : criterion = $d(v) + h(v, d)$

choose the right heuristic

Ex 5.20 : Held-Karp algorithm for TSP

$\ell(X, v)$ = the cost of a shortest path from s to v that visits every $x \in X$ exactly once



Bellman equation

$$\ell(X, v) = \min_{u \in X} \{ \ell(X \setminus \{u\}, u) + c(u, v) \}$$

Compute $\ell(X, v)$ for increasing subset size

IV/ Homework

- Find the complexity in Ex 5.20
- Ex 3.5 (Eulerian graphs) $\longrightarrow$ solutions in the notes on my website
- Ex 5.12 & Ex 5.19