INF421, Lecture 9
Shortest paths

Leo Liberti

LIX, École Polytechnique, France

Course

- **Objective**: to teach you some data structures and associated algorithms
- **Evaluation**: TP noté en salle info le 16 septembre, Contrôle à la fin. Note: $\max(CC, \frac{3}{4}CC + \frac{1}{4}TP)$
- **Organization**: fri 26/8, 2/9, 9/9, 16/9, 23/9, 30/9, 7/10, 14/10, 21/10, amphi 1030-12 (Arago), TD 1330-1530, 1545-1745 (SI31,32,33,34)
- **Books**:
  1. Ph. Baptiste & L. Maranget, *Programmation et Algorithmique*, Ecole Polytechnique (Polycopié), 2009
  2. G. Dowek, *Les principes des langages de programmation*, Editions de l'X, 2008
  3. D. Knuth, *The Art of Computer Programming*, Addison-Wesley, 1997
  4. K. Mehlhorn & P. Sanders, *Algorithms and Data Structures*, Springer, 2008
- **Website**: `www.enseignement.polytechnique.fr/informatique/INF421`
- **Contact**: `liberti@lix.polytechnique.fr` (e-mail subject: INF421)

Lecture summary

- Shortest Path Problems (SPP) and variants
- Dijkstra's algorithm
- Floyd-Warshall's algorithm
- Modelling shortest paths: flows
- A dual "algorithm"

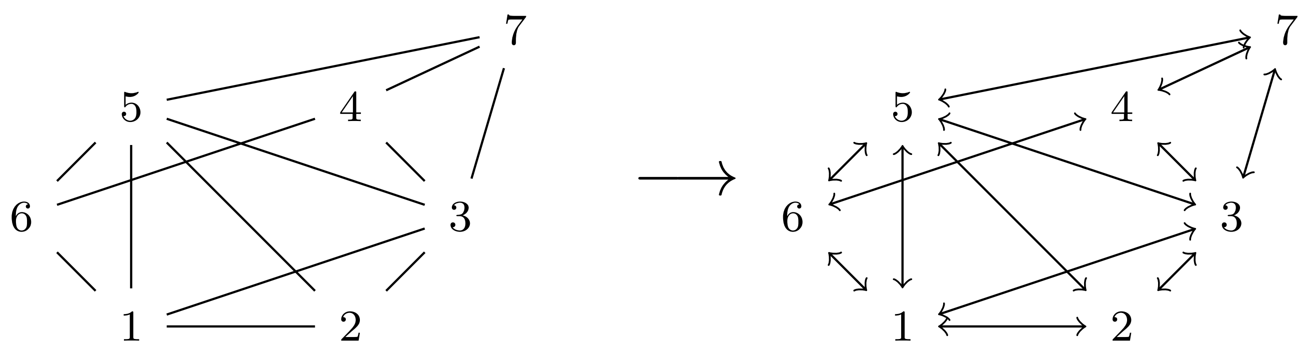
Minimal knowledge

- **Main SPP variants**: POINT-TO-POINT SHORTEST PATH (P2PSP), SHORTEST PATH TREE (SPT), unit / nonnegative arc costs, NEGATIVE CYCLE detection (NC), ALL SHORTEST PATHS (ASP)
- **SPT on unit costs**: use BFS (Lecture 2)
- **Dijkstra's algorithm**: like GRAPH SCANNING (Lecture 6) but with a priority queue; requires nonnegative arc costs
- **Floyd-Warshall's algorithm**: solves ASP and NC
- **Flows**: assignment of values to arcs so that some conservation constraints hold at each node, can be used to model SPPs with Mathematical Programming (MP)
- **Duality**: the dual MP formulation for P2PSP yields a surprising solution method!

Shortest path problems

Graphs or digraphs?

- In most applications, the correct model for SPPs is given by **arcs** and **digraphs** rather than **edges** and **graphs**
- SPPs also occur as sub-problems in complicated algorithms: we may need to solve SPPs on graphs
- Although directed paths are also called **walks** (Lectures 6, 8), we still use the term **path** for historical reasons
- Similarly, we use the term **cycle** to also mean circuits
- An SPP on a graph is equivalent to an SPP on the digraph where each edge is replaced two antiparallel arcs
- Conversely, replacing each arc (or pair of antiparallel arcs) of a digraph with an edge gives rise to the **underlying graph**



Motivation

Several SP problems can be solved in polynomial time

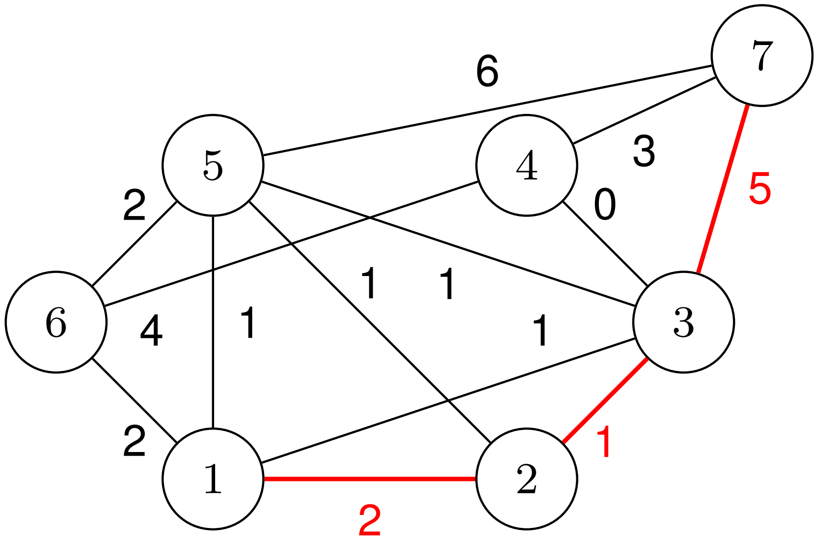
Cost of a path

- We consider a **weighted digraph** $G = (V, A)$ with **arc costs**
- I.e. we are given a function $c : A \to \mathbb{Q}$
- If $P \subseteq G$ is a path $u \to v$ in G then

$$c(P) = \sum_{(u,v)\in P} c_{uv},$$

where $c_{uv} = c((u,v))$

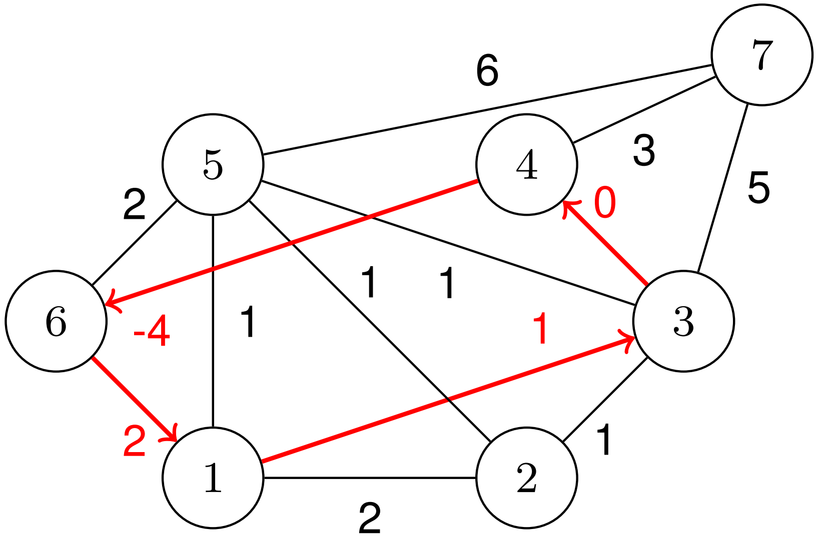
- For example, the path $1 \to 2 \to 3 \to 7$ has cost $2 + 1 + 5 = 8$



Shortest path = path P having minimum cost $c(P)$

Negative cycles

The red cycle has *negative cost* $1 + 0 - 4 + 2 = -1 < 0$



Thm.

If $G = (V, A)$ has a cycle C with $c(C) < 0$, $\not\exists$ no SP in G

Proof

Suppose P is SP $u \to v$ with cost c^*. Let $w \in V(C)$, consider path $Q = Q_1 \cup Q_2 \cup Q_3$ where Q_1 $u \to w$, $Q_2 = Q_1^{-1}$, and Q_3 consists of $k = \lceil \frac{c(Q_1)+c(Q_2)+c^*}{|c(C)|} \rceil + 1$ tours around C. Then $c(Q) = c(Q_1) + c(Q_2) + kc(C) < c^* \Rightarrow Q$ shorter than P (contradiction)

$\Rightarrow$ Need to assume c yields no negative cycles

Negative cycles: comments

- If c yields no negative cycles, call c **conservative**
- In order to construct Q in proof of above thm., we toured several times around negative cycle C
- $\Rightarrow Q$ is not a simple path
- If we look for the *shortest simple path* in graphs then we don't have this unboundedness problem
- The SHORTEST SIMPLE PATH (SSP) problem, however, is **NP**-hard on general non-conservatively weighted graphs
- Solving the LONGEST PATH problem is also **NP**-hard

  (Prove this by polynomially transforming SSP to LONGEST PATH, see Lecture 8 for an example of polynomial transformation)

Assumptions

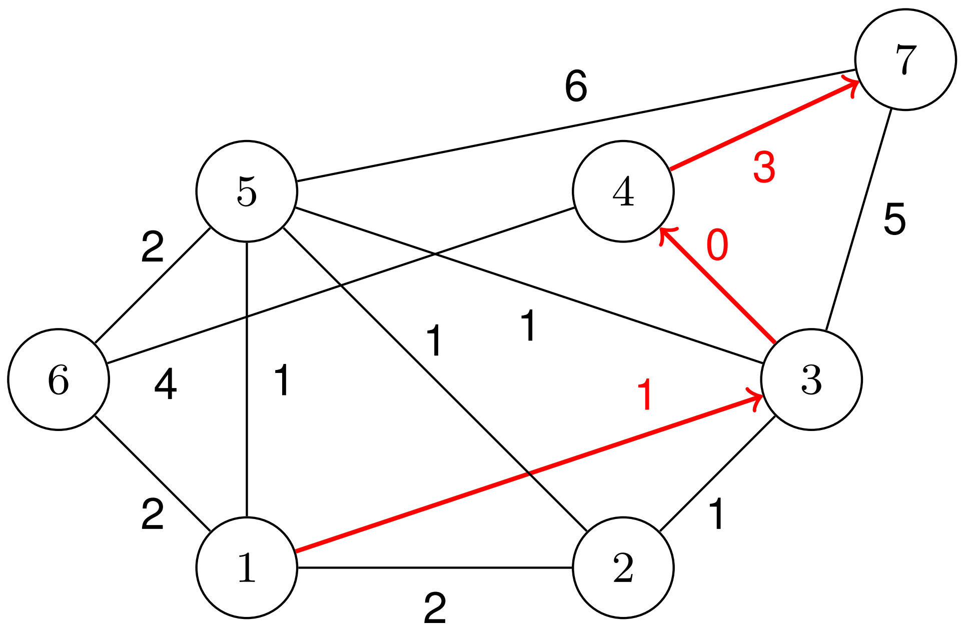
For the rest of these slides, if not otherwise specified, assume:

- G is connected (graph) or strongly connected (digraph)
- The arc costs c are conservative

Point-to-point shortest path

POINT-TO-POINT SHORTEST PATH (P2PSP). Given a digraph $G = (V, A)$, a function $c : A \to \mathbb{Q}$ and two distinct nodes $s, t \in V$, find a SP $s \to t$

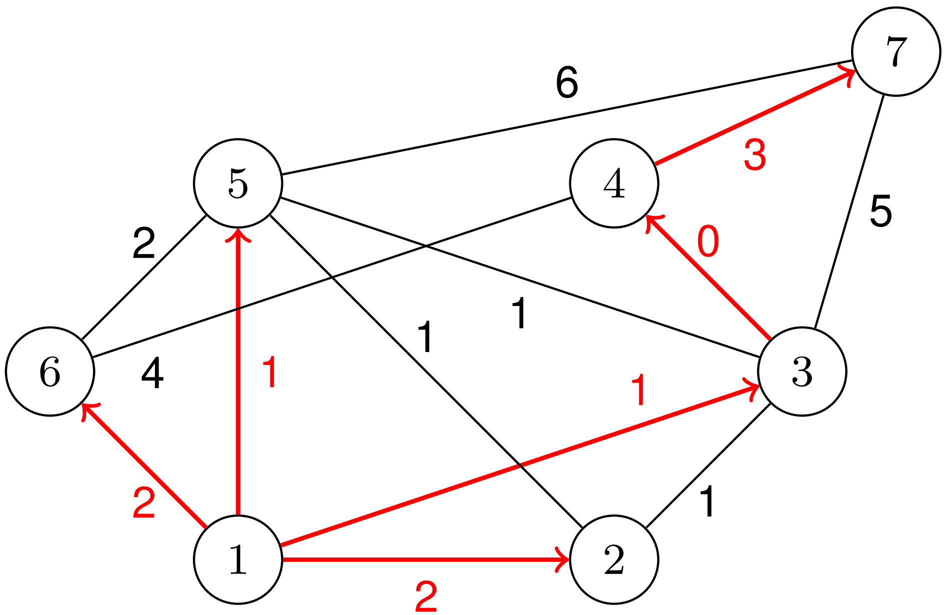
A shortest path $1 \to 7$



Shortest path tree

SHORTEST PATH TREE (SPT). Given a digraph $G = (V, A)$, a function $c : A \to \mathbb{Q}$ and a source node $s \in V$, find SPs $s \to v$ for all $v \in V \setminus \{s\}$

- *Remark*: there may be more than one SP $s \to v$
- **Consistency**: one can always choose SP P_{sv} $u \to v$ so that $T = \bigcup_{v \neq s} P_{sv}$ is a spanning oriented tree ($\Leftrightarrow \forall v \neq s \ (N_T^-(v) = 1)$)
- **Thm. A** If c is conservative, every initial subpath of a SP is a SP (e.g. subpath $1 \to 4$ of SP $1 \to 7$ below is a SP $1 \to 4$)



Let P be a SP $s \to w$ and Q a SP $s \to v$ through w; if the **predecessor of** w **in** P **is** $\mathsf{p}_P(w) = z_1$ *and* $\mathsf{p}_Q(w) = z_2$ *with $z_1 \neq z_2$, then no sp. or. tree T can contain $P \cup Q$. By Thm. A above, the initial subpath P' to w of Q is also a SP $s \to w$, so replace P with P' and obtain $|N^-_{P' \cup Q}(w)| = 1$ as required.*

All shortest paths

ALL SHORTEST PATHS (ASP). Given a digraph $G = (V, A)$ and a function $c : A \to \mathbb{Q}$, find SPs $u \to v$ for all pairs u, v of distinct nodes in V

Variants

- **Unit costs**: for all $(u, v) \in A$ we have $c_{uv} = 1$
- **Non-negative costs**: for all $(u, v) \in A$ we have $c_{uv} \geq 1$
- Several others, too many to list them all
- *A remarkable one*: SPT on undirected graphs with $c : E \to \mathbb{N}$ can be solved in linear time [Thorup 1997]
- SPT on unit costs: use BFS (see Lectures 2, 6), $O(m + n)$

Dijkstra's algorithm

The problem it targets

Dijkstra's algorithm solves the SPT on weighted digraphs $G = (V, A)$ with non-negative costs (with a given source node $s \in V$)

- If $c \geq 0$ then c is conservative (why?)
- Worst-case complexity: $O(n^2)$ on general digraphs, $O(m + n \log n)$ on sparse graphs, where $n = |V|$ and $m = |A|$
- Used as a sub-step in innumerable algorithms
- Main application: routing in networks (usually transportation and communication)

Data structures

- We maintain two functions
  - $d : V \to \mathbb{Q}_+$

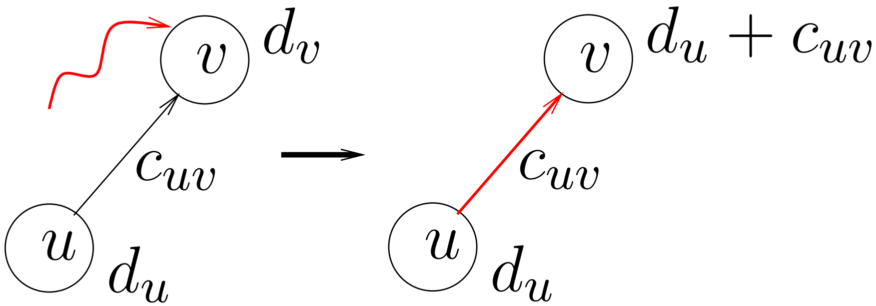
    $d_v = d(v)$ is the cost of a SP $s \to v$ for all $v \in V$
  - $\mathsf{p} : V \to V$

    $\mathsf{p}_v = \mathsf{p}(v)$ is the predecessor of v in a SP $s \to v$ for all $v \in V$
- Initialization
  - $d_s = 0$ and $d_v = \infty$ for all $v \in V \smallsetminus \{s\}$
  - $\mathsf{p}(v) = s$ for all $v \in V$

Settle and Relax

- A node $v \in V$ is **settled** when d_v no longer changes
- **Relaxing** an arc $(u, v) \in A$ consists in:

  if $d_u + c_{uv} < d_v$ **then**
  Let $d_v = d_u + c_{uv}$;
  Let $\mathsf{p}_v = u$;
  end if



- When (u, v) is relaxed and v is not settled yet, d_v might change

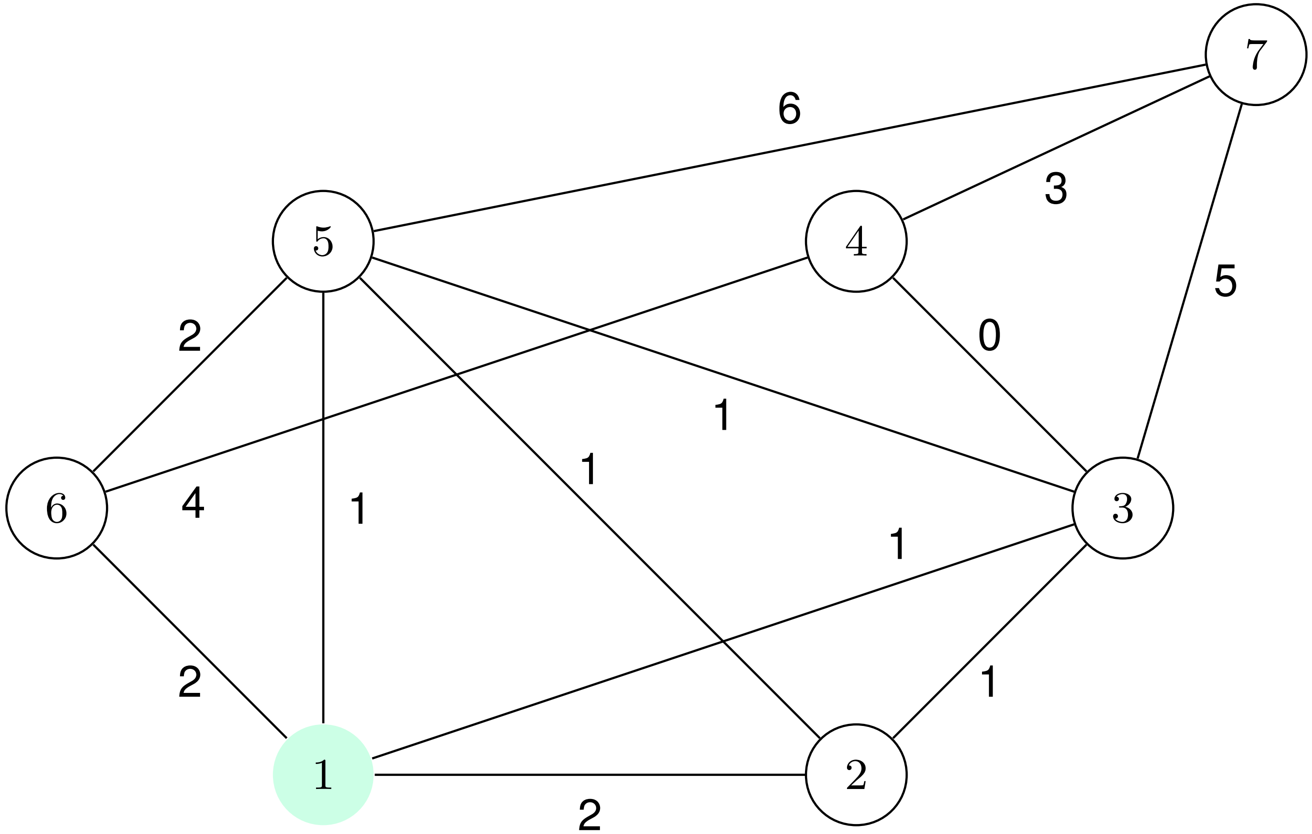
Description

- Dijkstra's algorithm :

```
1: while ∃ unsettled nodes do
2:   Let u be an unsettled node with minimum d_u;
3:   Settle u;
4:   for (u, v) ∈ A do
5:     Relax (u, v);
6:   end for
7: end while
```

- If $d_v = \infty$ at Step 4, relaxing (u, v) will necessarily change d_v (why?)
- Nodes $v \in V$ such that $d_v < \infty$ are **reached**
- A simple implementation is $O(n^2)$

Example with $s = 1$

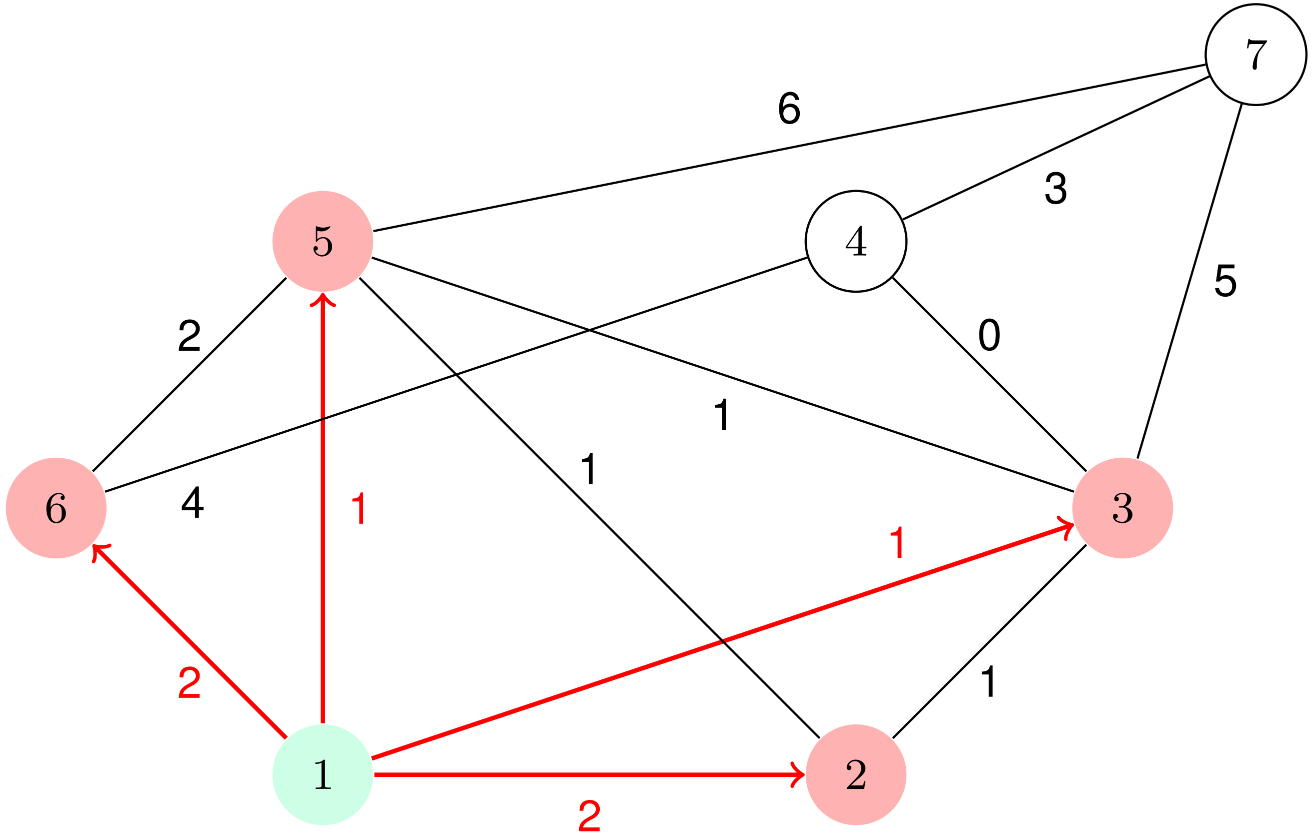


d :	1	2	3	4	5	6	7
	0	∞	∞	∞	∞	∞	∞

p :	1	2	3	4	5	6	7
	1	1	1	1	1	1	1

initialize (settle) $s = 1$

Example with $s = 1$



d :

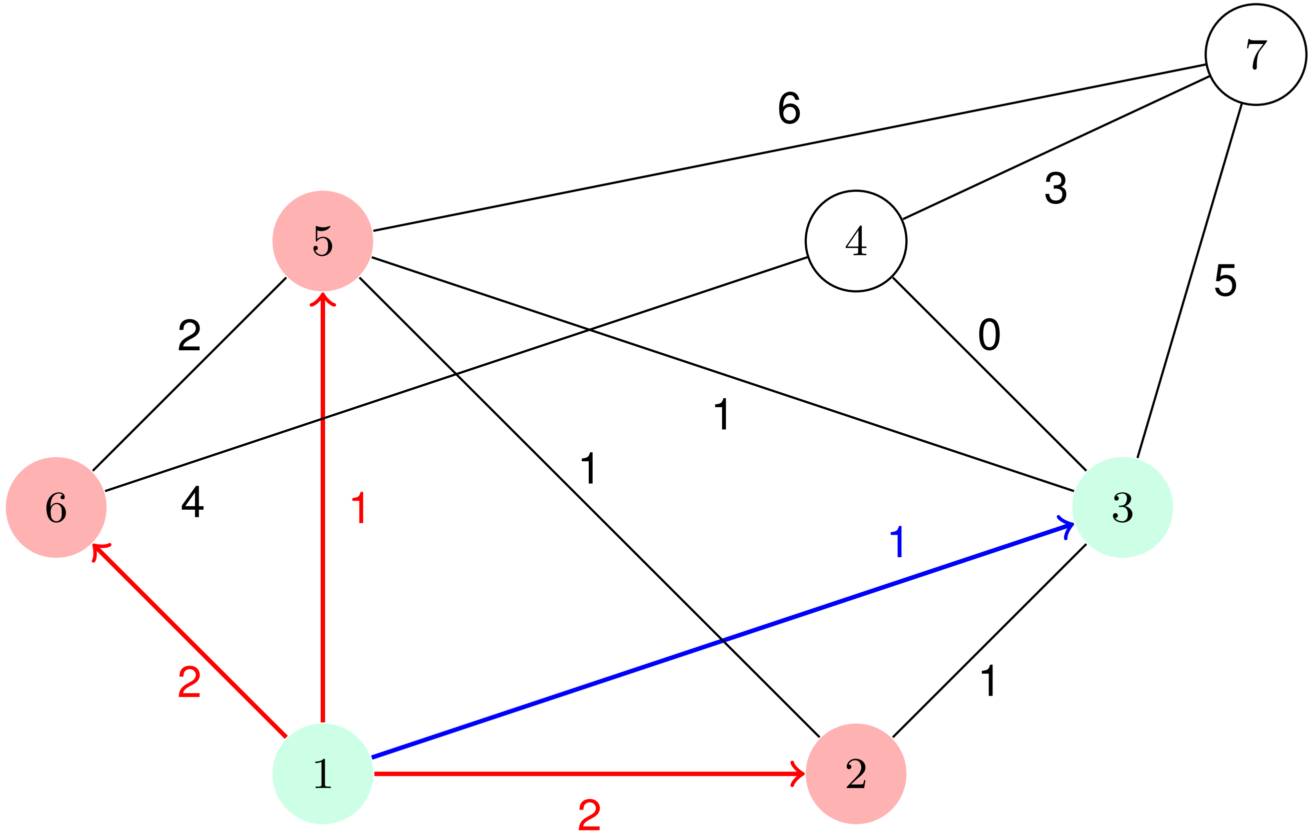
1	2	3	4	5	6	7
0	2	1	∞	1	2	∞

p :

1	2	3	4	5	6	7
1	1	1	1	1	1	1

relax $\delta^+(1)$, update $2, 3, 5, 6$

Example with $s = 1$



d :

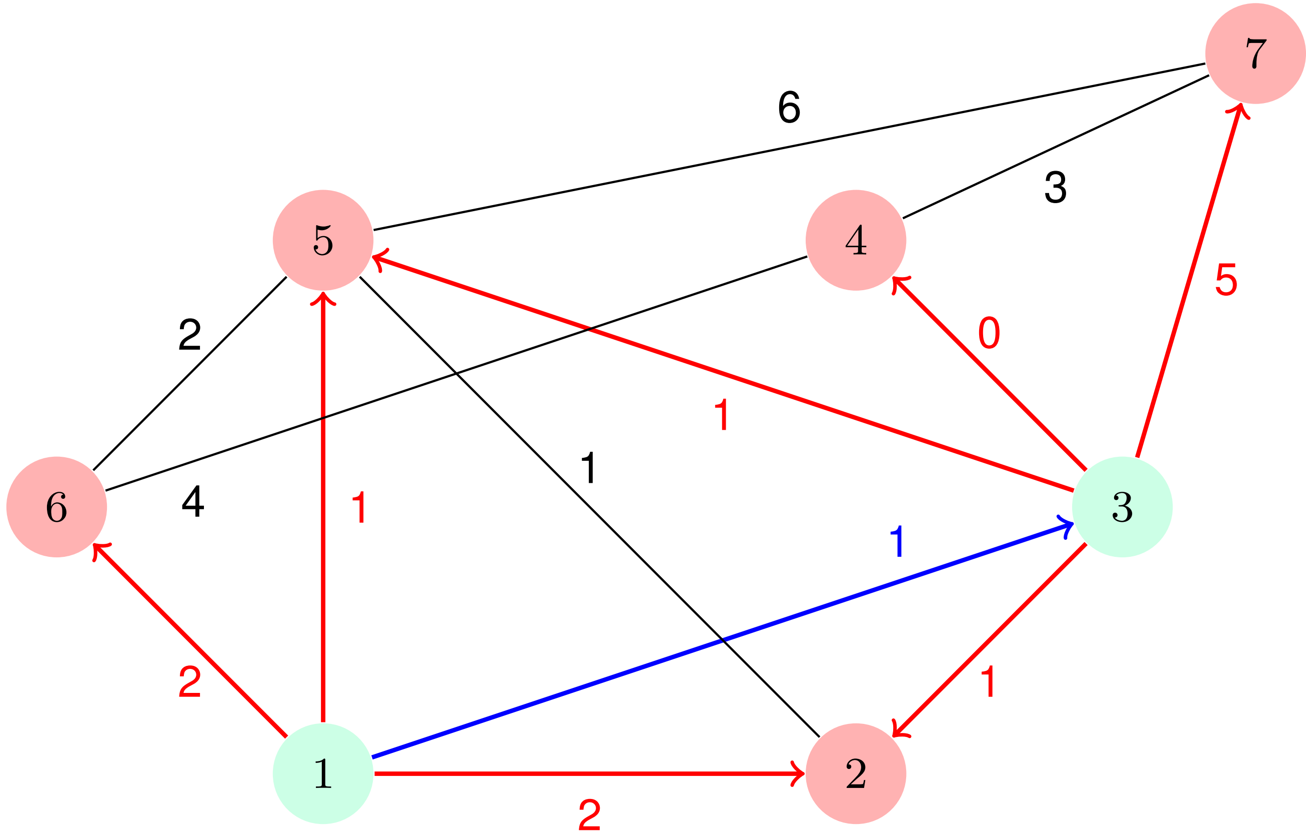
1	2	3	4	5	6	7
0	2	1	∞	1	2	∞

p :

1	2	3	4	5	6	7
1	1	1	1	1	1	1

settle 3 ($d_3 = 1$ is minimum)

Example with $s = 1$



d :

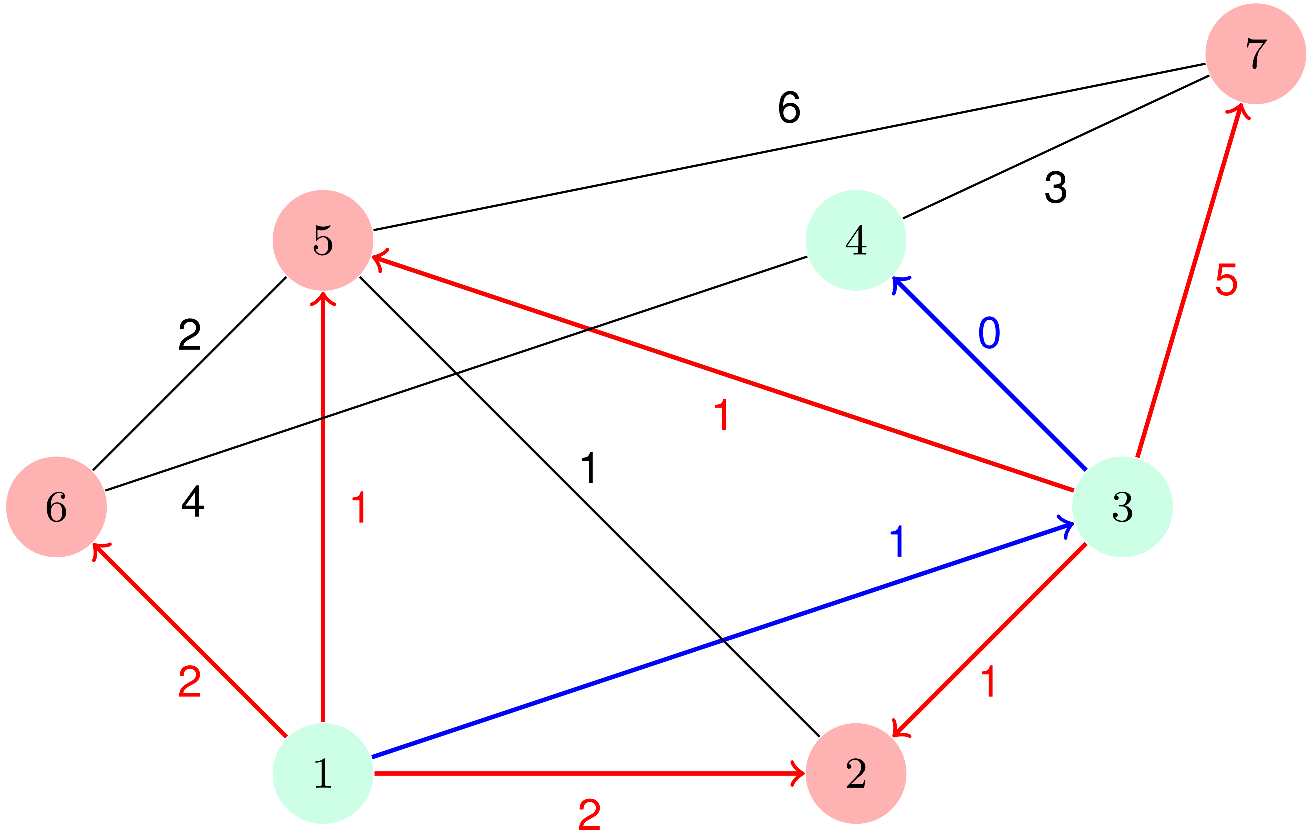
1	2	3	4	5	6	7
0	2	1	1	1	2	6

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	3

relax $\delta^+(3)$, update $4, 7$

Example with $s = 1$



d :

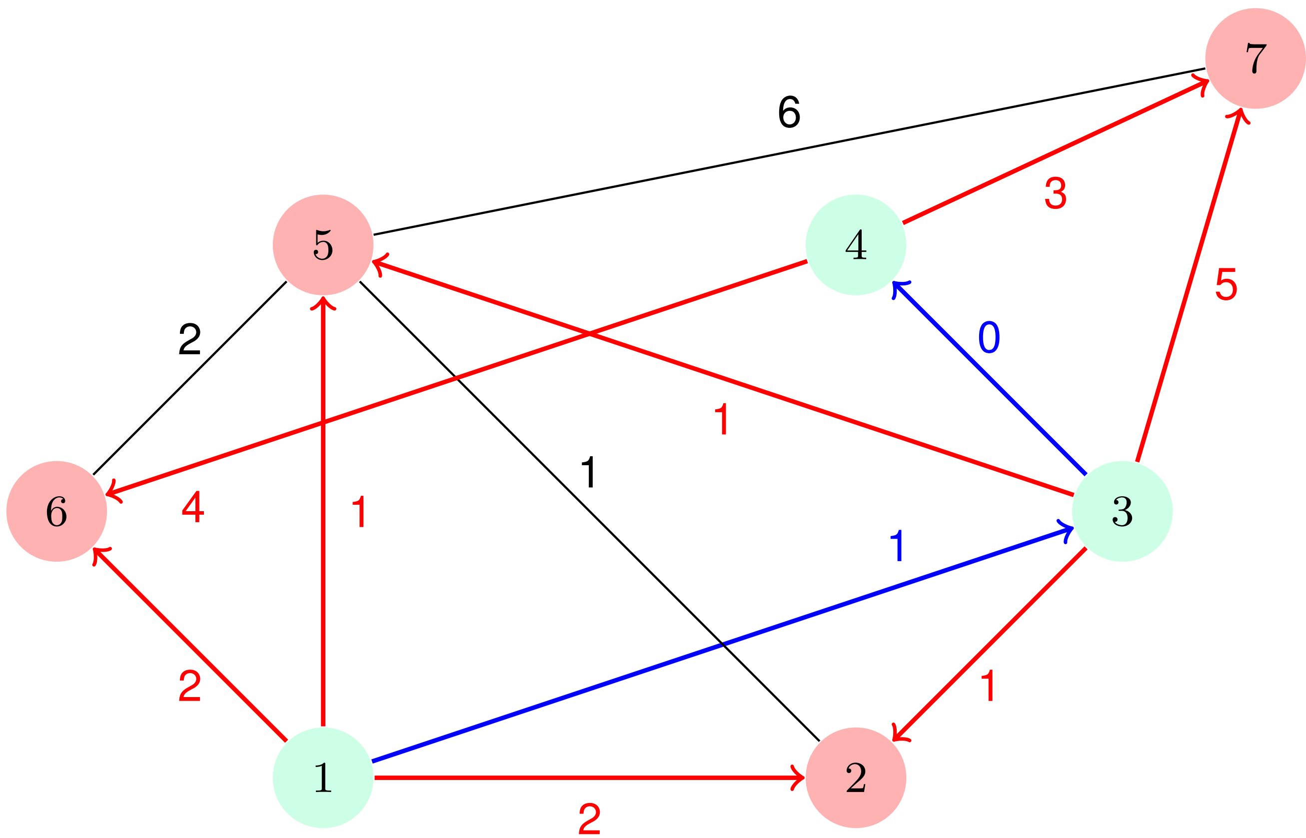
1	2	3	4	5	6	7
0	2	1	1	1	2	6

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	3

settle 4 ($d_4 = 1$ is minimum)

Example with $s = 1$

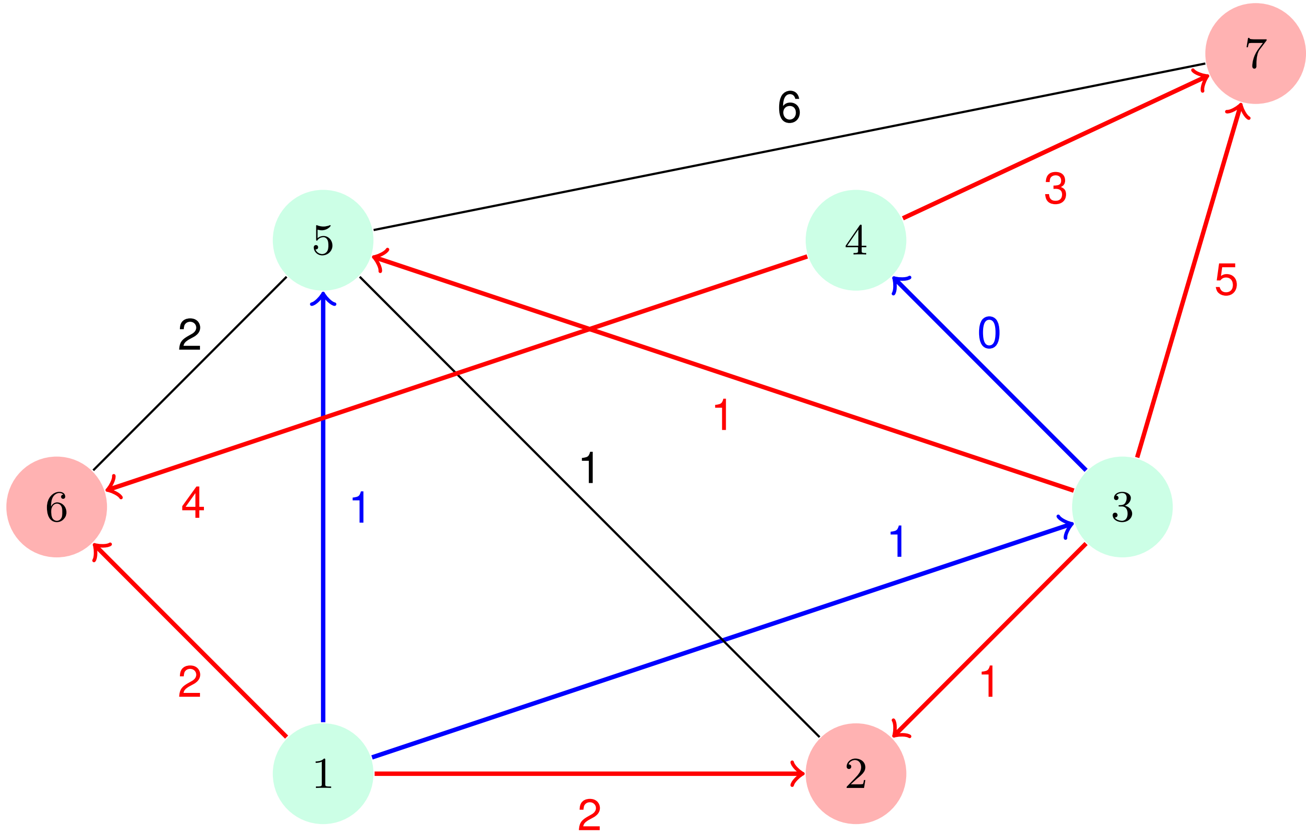


d :	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

relax $\delta^+(4)$, update 7

Example with $s = 1$



d :

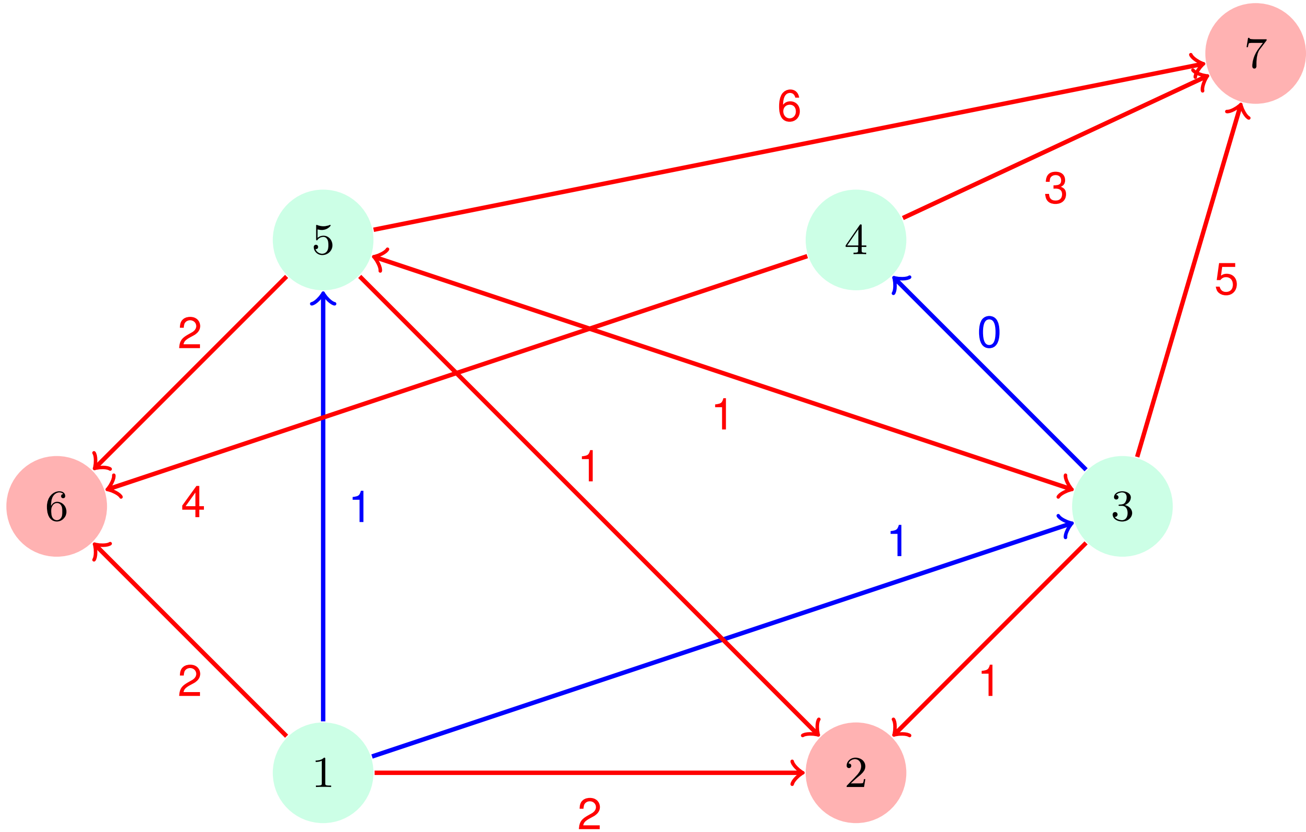
1	2	3	4	5	6	7
0	2	1	1	1	2	4

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	4

settle 5 ($d_5 = 1$ is minimum)

Example with $s = 1$

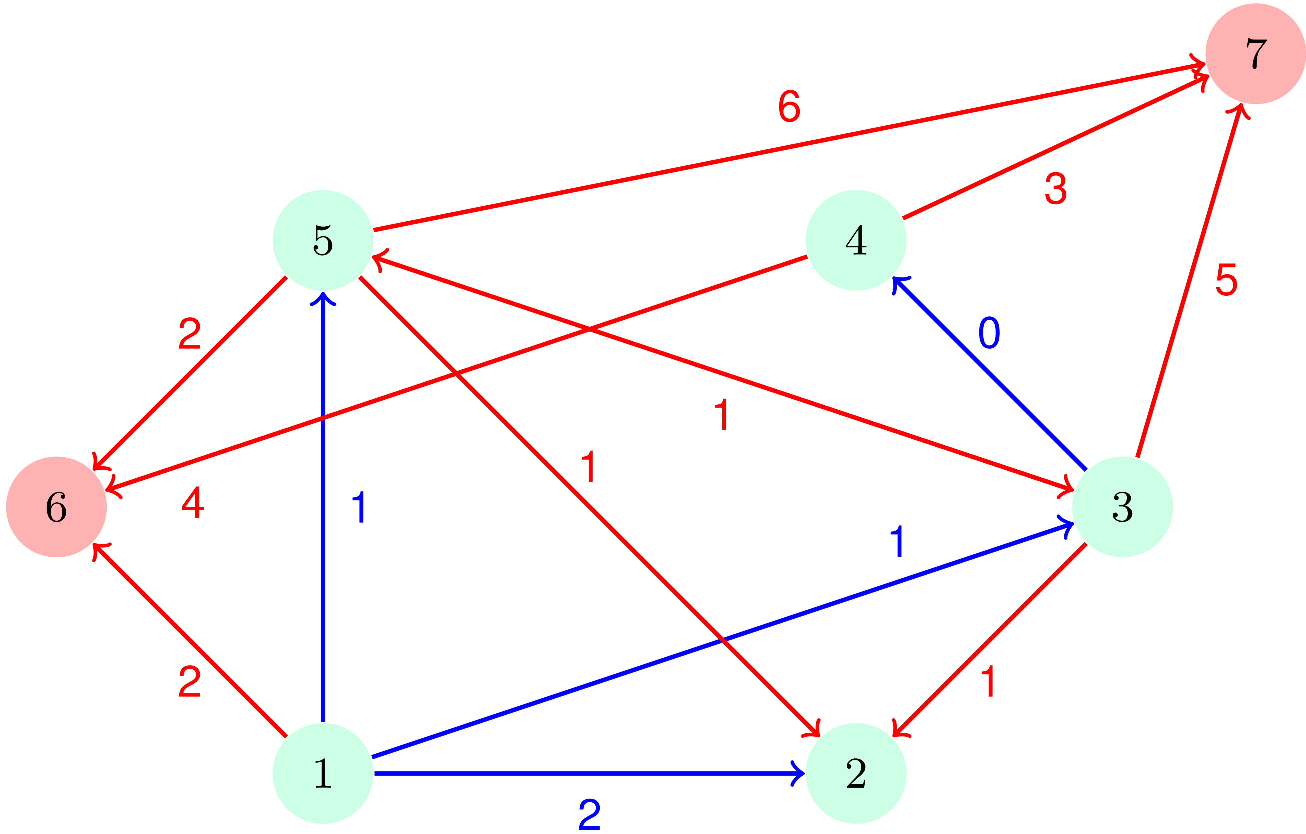


d :	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

relax $\delta^+(5)$

Example with $s = 1$

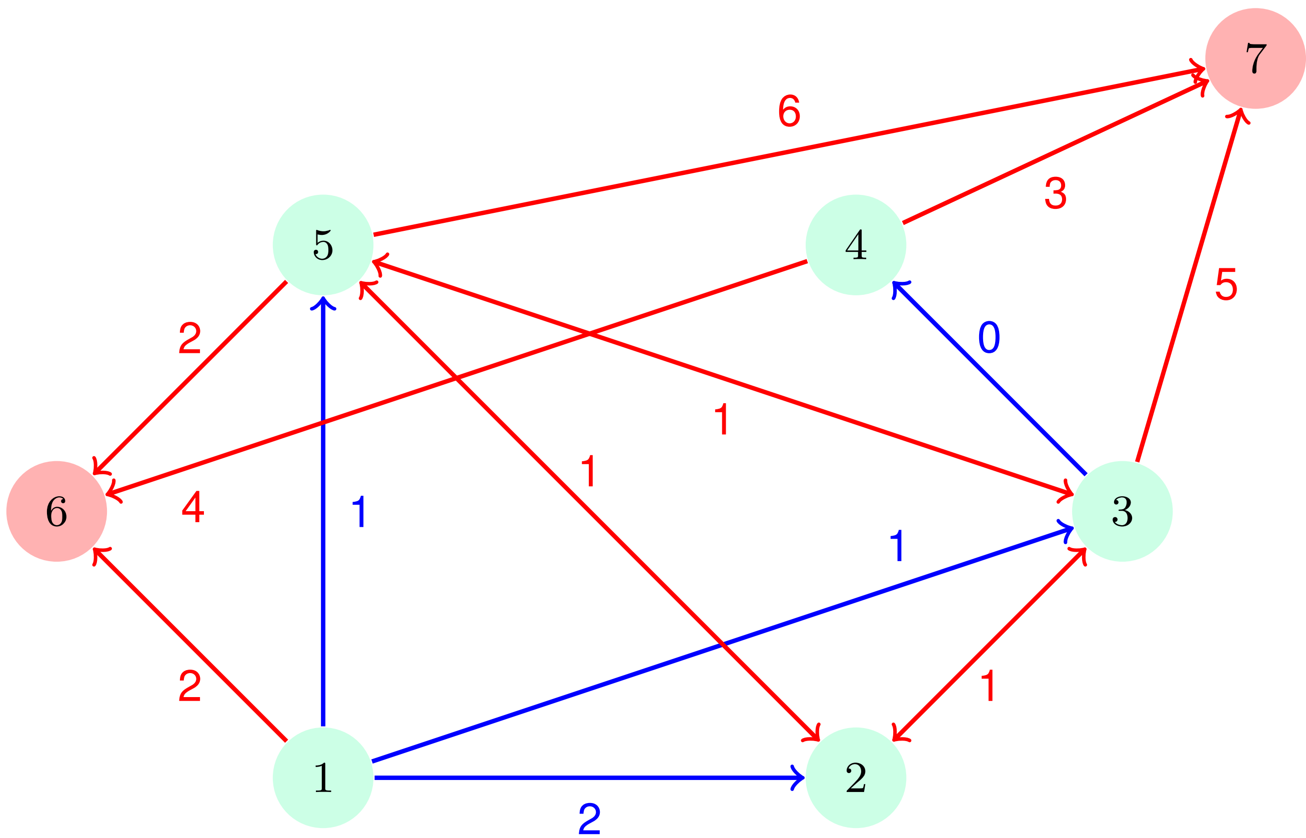


d : 1	2	3	4	5	6	7
0	2	1	1	1	2	4

p : 1	2	3	4	5	6	7
1	1	1	3	1	1	4

settle 2 ($d_2 = 2$ is minimum)

Example with $s = 1$



d :

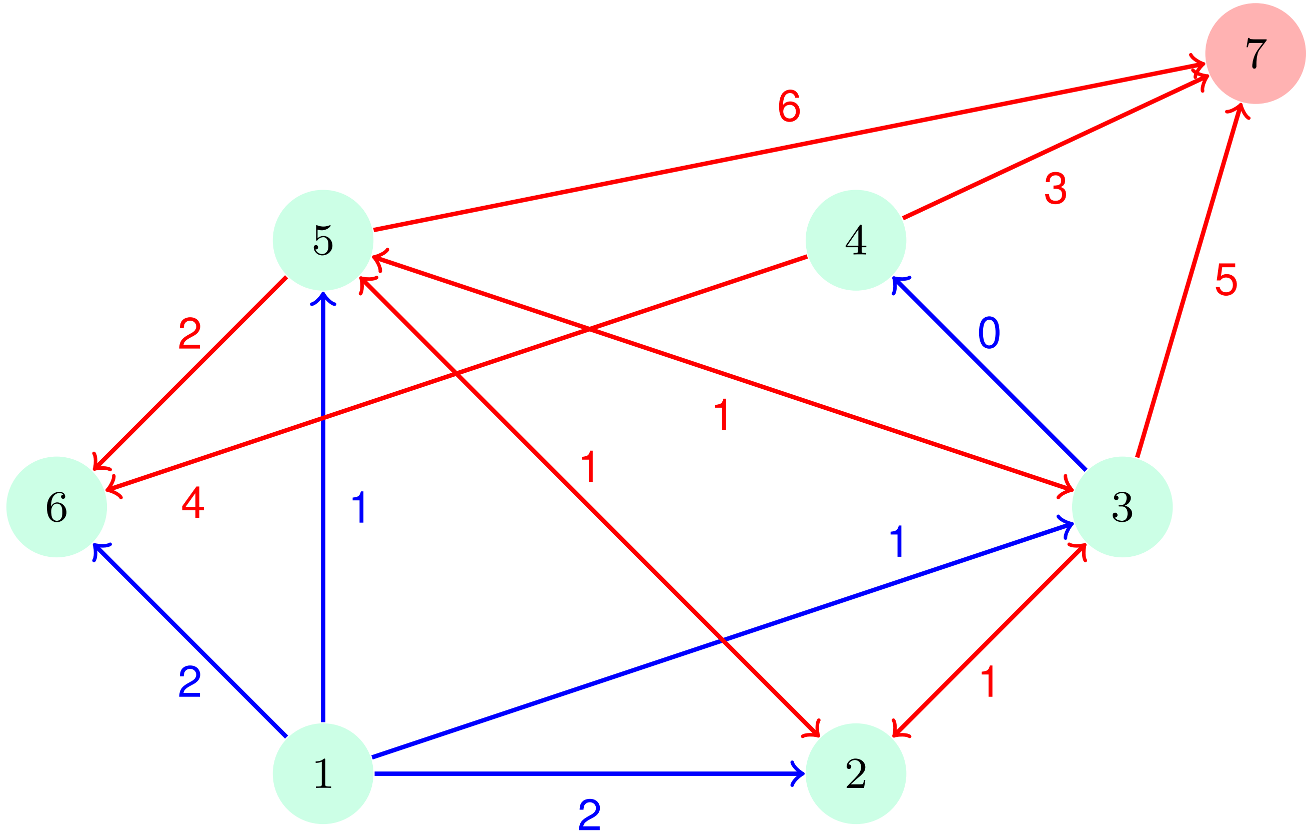
1	2	3	4	5	6	7
0	2	1	1	1	2	4

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	4

relax $\delta^+(2)$

Example with $s = 1$

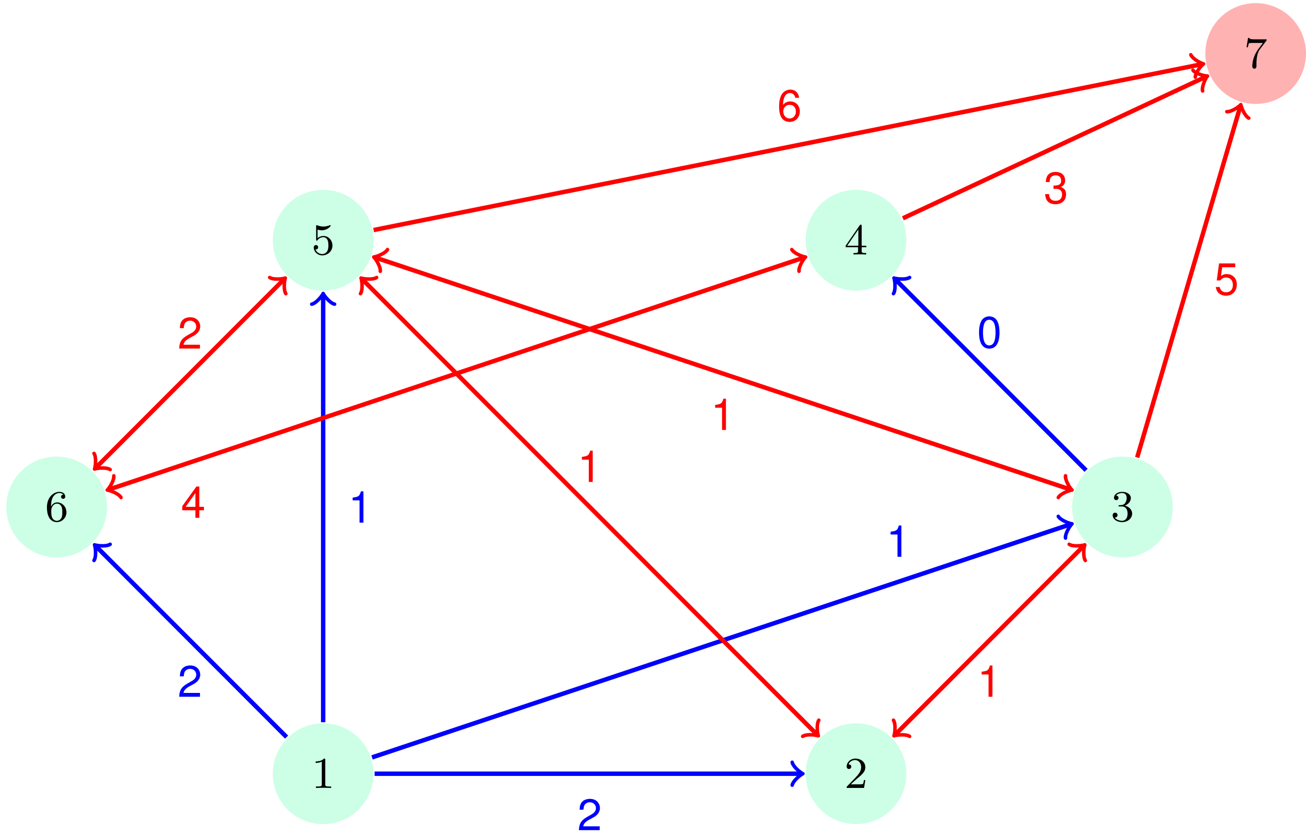


$d:$ 1	2	3	4	5	6	7
0	2	1	1	1	2	4

p : 1	2	3	4	5	6	7
1	1	1	3	1	1	4

settle 6 ($d_6 = 2$ is minimum)

Example with $s = 1$



d :

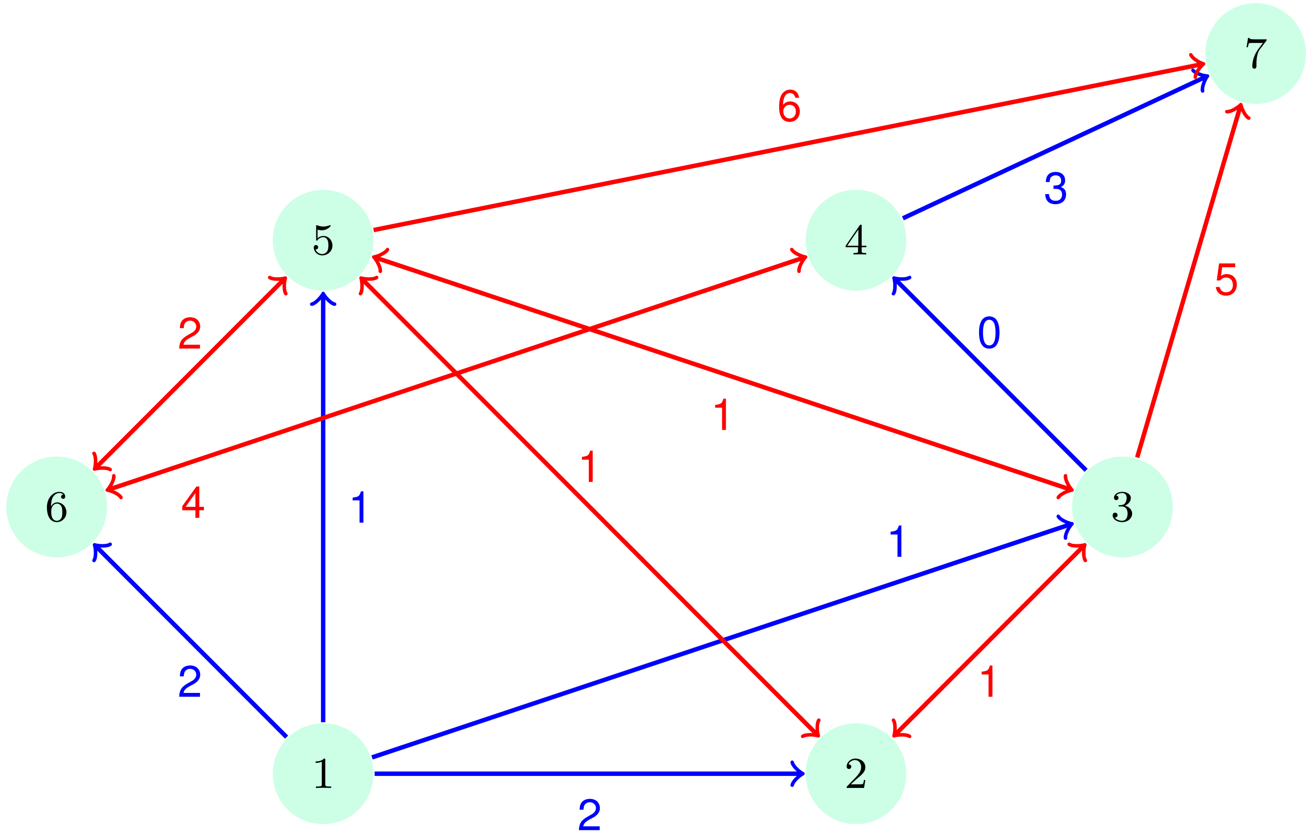
1	2	3	4	5	6	7
0	2	1	1	1	2	4

p :

1	2	3	4	5	6	7
1	1	1	3	1	1	4

relax $\delta^+(6)$

Example with $s = 1$

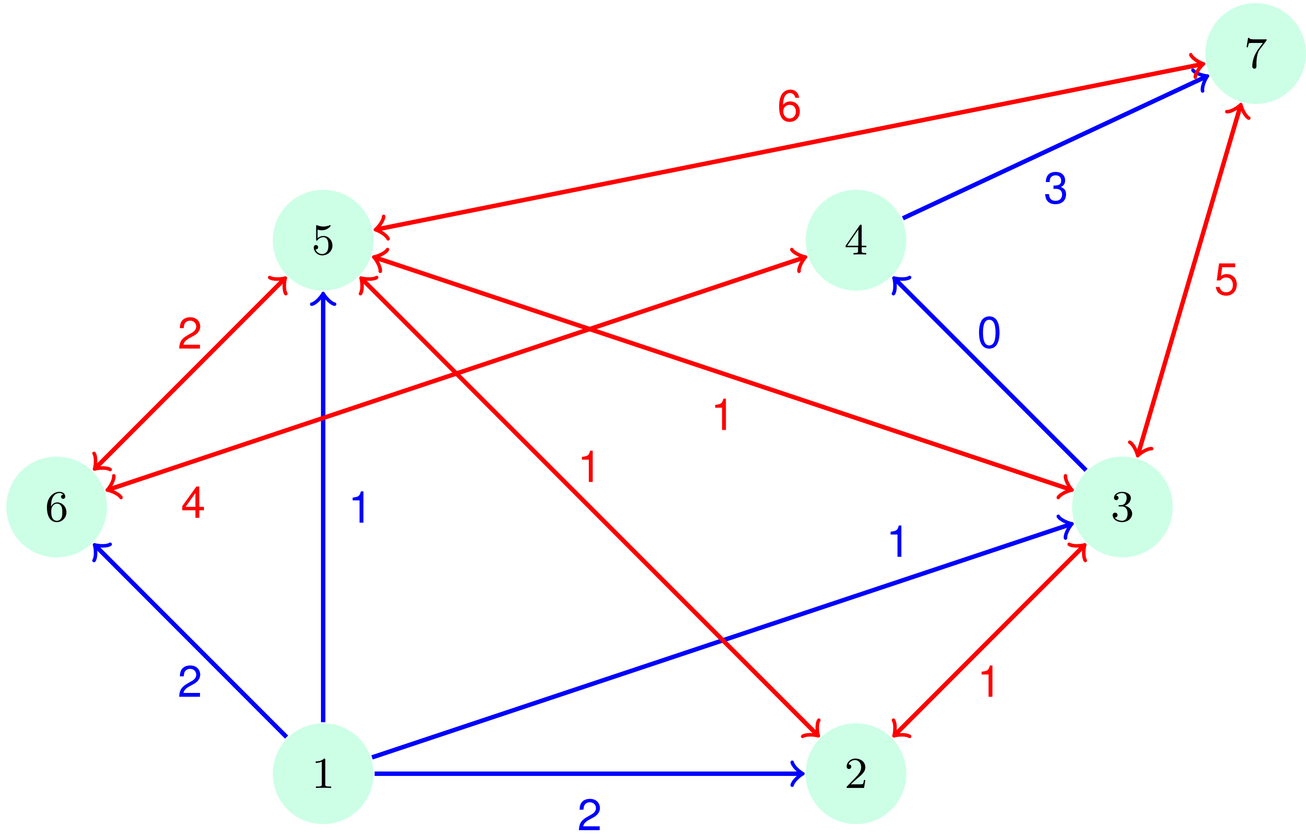


d : 1	2	3	4	5	6	7
0	2	1	1	1	2	4

p : 1	2	3	4	5	6	7
1	1	1	3	1	1	4

settle 7 ($d_7 = 4$ is minimum)

Example with $s = 1$

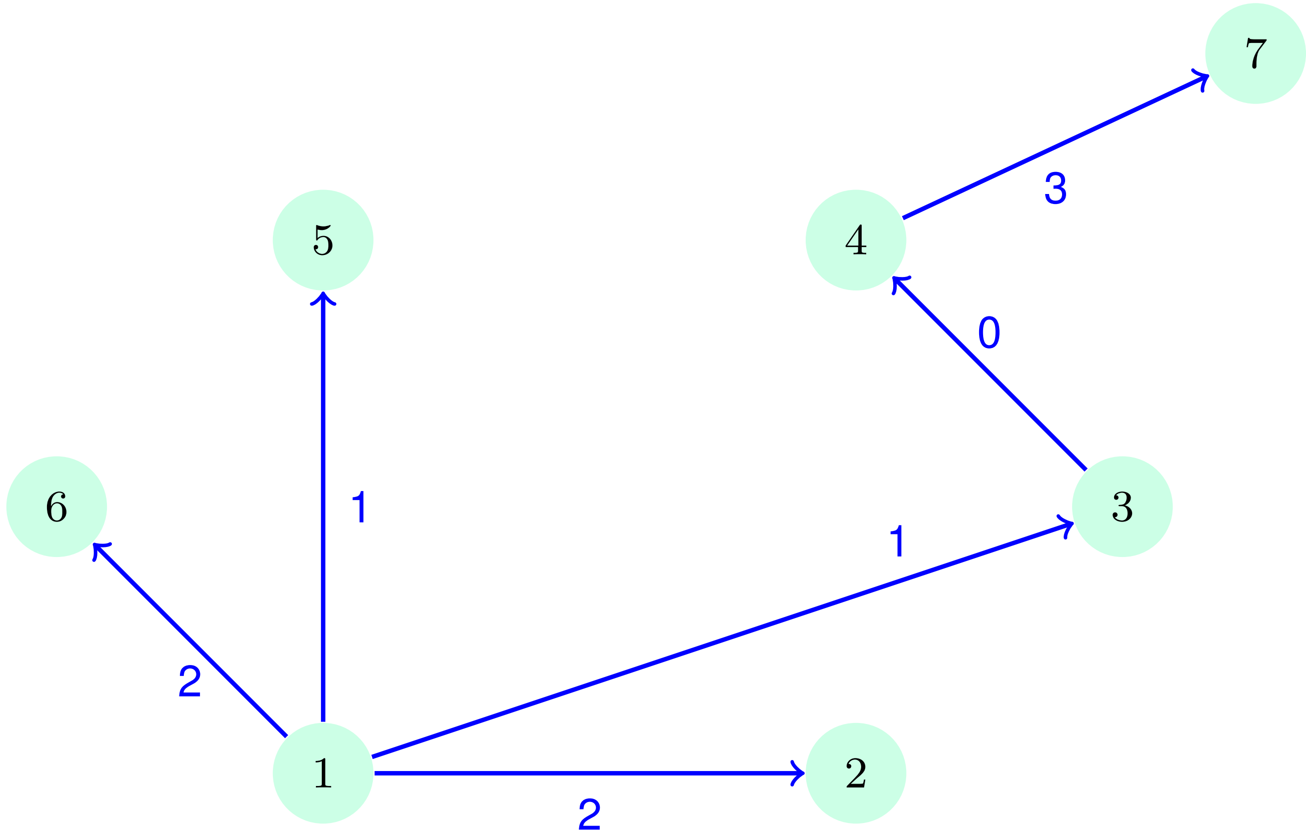


d:	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

p:	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

relax $\delta^+(7)$

Example with $s = 1$



$d:$	1	2	3	4	5	6	7
	0	2	1	1	1	2	4

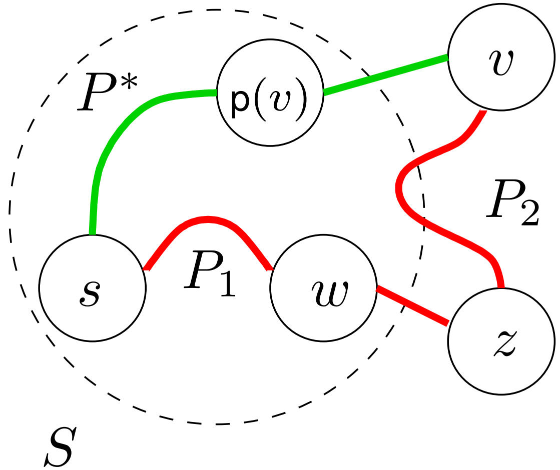
p :	1	2	3	4	5	6	7
	1	1	1	3	1	1	4

An optimal SPT solution

The algorithm is correct 1/2

Thm.

At any iteration and for each $v \in V$, d_v is the cost of a SP $s \to v$ where all predecessors of v are settled



Proof

By induction on itn. index k. Let S be the set of settled nodes at itn. $k-1$, let u be chosen at Step 2 of itn. k, and P^* be the path $s \to v$ determined by the alg. Suppose $\exists$ another path P from s to v with cost $c(P)$. Since $v \not\in S$, there must be $(w,z) \in A$ with $w \in S$ and $z \not\in S$ s.t. $P = P_1 \cup \{(w,z)\} \cup P_2$, where $V(P_1) \subseteq S$. Then $c(P) = c(P_1) + c_{wz} + c(P_2) \geq c(P_1) + c_{wz}$ *(because we subtracted $c(P_2)$)* $= d_w + c_{wz}$ *(by induction)* $= d_z \geq d_v$ *(because otherwise d_v would not be minimum, contradicting the choice of v at Step 2)* $= c(P^*)$, so that P^* is a SP $s \to v$

The algorithm is correct 2/2

- Remains to prove: at the end of the algorithm, every node is settled
- Similar to proof that **Graph Scanning** reaches all vertices in a graph (Lecture 6)
- Left as an exercise

Implementation

- No unreached node v can ever have minimum d_v at Step 2 since $d_v = \infty$ if v unreached
- The minimum choice at Step 2 occurs over unsettled, reached nodes $\Rightarrow$ **maintain a data structure containing unsettled, reached nodes**
- Data structure that provides minimum in constant time: **priority queue**
- When arc (u, v) is relaxed and v is already reached, the priority d_v might be updated
- We update a priority by deleting then re-inserting the element with the new priority (can implement `delete` in $O(\log n)$)

Pseudocode

$\forall v \in V\ d_v = \infty$, $d_s = 0$;
$\forall v \in V$ p$_v = s$;
Q.insert(s, d_s);
while $Q \neq \varnothing$ **do**
  Let $u = Q$.popMin();
  for $(u, v) \in \delta^+(u)$ **do**
    Let $\Delta = d_u + c_{uv}$;
    if $\Delta < d_v$ **then**
      Let $d_v = \Delta$;
      Let p$_v = u$;
      Q.delete(v); // if $v \notin Q$ this does nothing
      Q.insert(v, d_v);
    end if
  end for
end while

Worst-case complexity

- Each node is settled exactly once (why? argue by contradiction) $\Rightarrow$
  1. `popMin`() is called $O(n)$ times $\Rightarrow O(n \log n)$
  2. each arc is relaxed exactly once $\Rightarrow O(m \log n)$
- This yields an $O((n+m)\log n)$ algorithm
- Worse than $O(n^2)$ if graph is dense, however graphs in practice are usually sparse: competitive
- Can improve to $O(m + n \log n)$ with more refined data structures

Point-to-point SPs

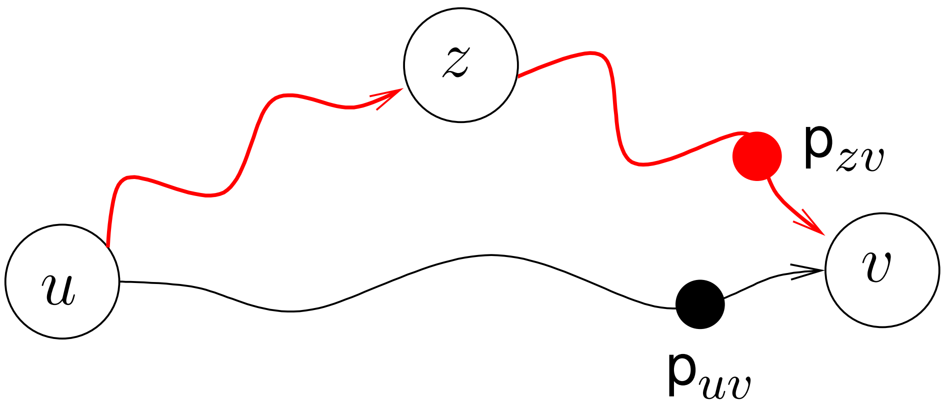
- The P2PSP from s to t on nonnegatively weighted digraphs can be solved by Dijkstra's algorithm
- Simply terminate as soon as v is settled
- Insert the following code between Step 5 and 6:

if $u = t$ **then**
  exit;
end if

Floyd-Warshall's algorithm

Solves ASP

- Solves the ASP with conservative arc costs c
- Data structures: two $n \times n$ matrices d, p
  - d_{uv} =cost of SP $u \to v$
  - p_{uv} =predecessor of v in SP from u
- For each node z and pair u, v of nodes, see if SP $u \to v$ can be improved by passing through z



- If so, update d_{uv} to $d_{uz} + d_{zv}$ and p_{uv} to p_{zv}

The simplest algorithm!

$\forall u, v \in V \; d_{uv} = \begin{cases} c_{uv} & \text{if } (u,v) \in A \\ \infty & \text{otherwise} \end{cases}$

$\forall u, v \in V$ $\mathsf{p}_{uv} = u$

for $z \in V$ **do**

  for $u \in V$ **do**

    for $v \in V$ **do**

      $\Delta = d_{uz} + d_{zv}$;

      if $\Delta < d_{uv}$ **then**

        $d_{uv} = \Delta$;

        $\mathsf{p}_{uv} = \mathsf{p}_{zv}$;

      end if

    end for

  end for

end for

Remarks

- **Worst-case complexity**: clearly $O(n^3)$
- **Algorithm is correct**: every possible triangulation was tested
- Also solves NEGATIVE CYCLE (NC):
  - Assume there is a negative cycle through u
  - When $u = v$, triangulations will eventually yield $d_{uu} < 0$
  - Whenever that happens, terminate: a negative cycle was found
  - After Step 6, insert code:

    if $\Delta < 0$ **then**
      exit;
    end if

Flows

Definitions

Defn.

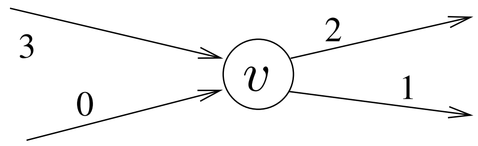
A **flow** is a pair of functions $(x : A \to \mathbb{R}, b : V \to \mathbb{R})$ s.t.:

$$\forall u \in V \qquad \sum_{(u,v)\in A} x_{uv} - \sum_{(v,u)\in A} x_{vu} = b_u$$

- Whenever $b_v = 0$ for some $v \in V$, then the above becomes

$$\forall v \in V \quad b_v = 0 \to \sum_{(u,v)\in A} x_{uv} = \sum_{(v,u)\in A} x_{vu} \qquad (1)$$

- The entering flow in v is equal to the exiting flow



- Eq. (1) are the **flow conservation** equations

Mathematical Programming

- Flow equations help define connected subgraphs:

  G connected $\Rightarrow \forall u \neq v \in V(G)$ a unit of flow entering u will exit u as long as $b_z = 0$ for all $z \neq u, v$. Conversely: $\forall u \neq v \in V(G)\ \exists$ a flow (x, b) where $b_u = 1, b_v = -1, \forall z \neq u, v (b_z = 0) \Rightarrow G$ connected

- Can use flow equations in Mathematical Programs (MP)
- E.g. a SP $s \to t$ is the connected subgraph of minimum cost containing s, t:

$$\left.\begin{array}{rrcl} \min\limits_{x:A\to\mathbb{R}} & \sum\limits_{(u,v)\in A} c_{uv}x_{uv} & & \\ \forall u \in V & \sum\limits_{(u,v)\in A} x_{uv} - \sum\limits_{(v,u)\in A} x_{vu} & = & \left\{\begin{array}{rl} 1 & u = s \\ -1 & u = t \\ 0 & \text{othw.} \end{array}\right. \\ \forall (u,v) \in A & x_{uv} & \in & \{0,1\} \end{array}\right\} \text{[SP]}$$

Test this with AMPL

A dual algorithm

MP in flat form

- Every MP involving linear forms only can be written in the form

$$\left.\begin{array}{rrcl} \min_x & \gamma^{\mathsf{T}} x & & \\ & Ax & \le & \beta \\ & x & \in & X \end{array}\right\} [P]$$

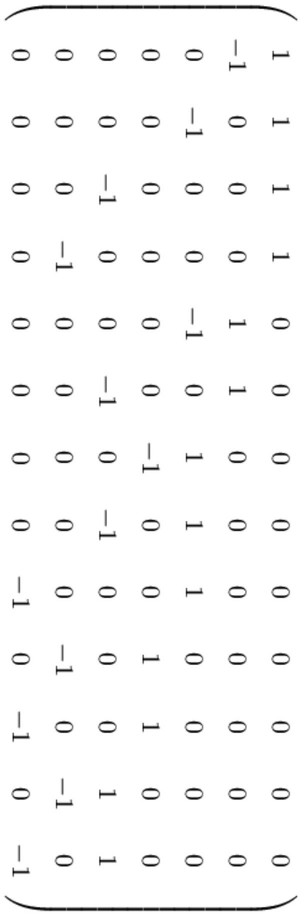
- $\gamma, x \in \mathbb{R}^n$, $\beta \in \mathbb{R}^m$, A is $m \times n$, X is the set where variables range
- For P2PSP on our usual graph with $s = 1$ and $t = 7$ we have:
  - $\gamma = (1, \ldots, 1)$, $\beta = (1, 0, 0, 0, 0, 0, 1)$, $X = \{0, 1\}^{13}$
  - $A =$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 \end{pmatrix}$$

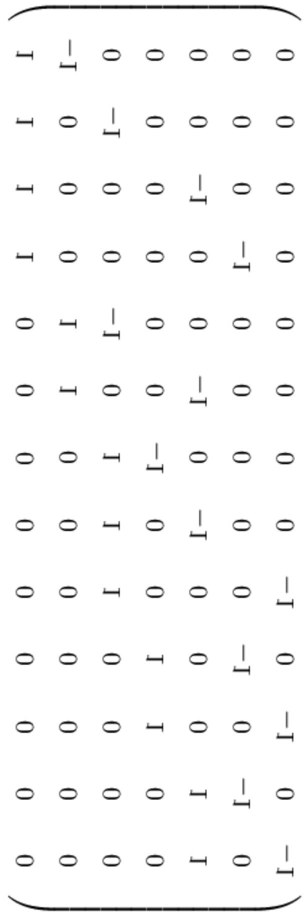
Transpose

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 \end{pmatrix}$$

(turn) $\longrightarrow$



(reflect) $\longrightarrow$



A dual view

- Let $A^{\mathsf{T}} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$
- Turn rows into columns (constraints into variables)
- . . . and columns into rows (variables into constraints)

LP Dual

- For each constraint define a variable y_i ($i \le 7$)
- The **Linear Programming Dual** is

$$\left.\begin{array}{rcl} \max_y & -y\beta & \\ yA & \le & \gamma \end{array}\right\} [D]$$

- In the case of the SP formulation, the dual is:

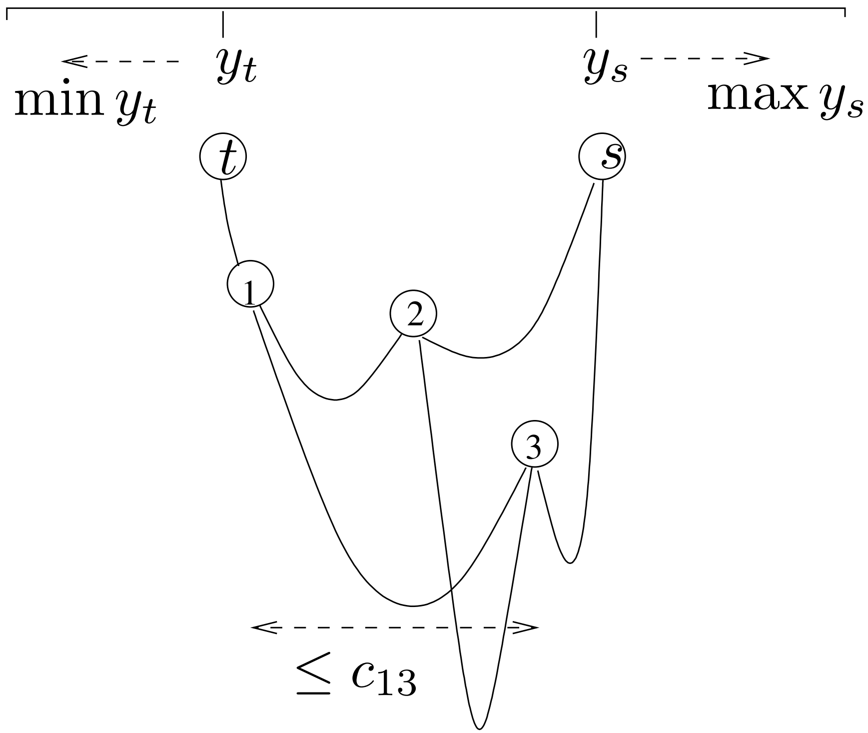
$$\left.\begin{array}{rrcl} \max_y & y_t - y_s & & \\ \forall (u,v) \in A & y_v - y_u & \le & c_{uv} \end{array}\right\} [D_{\text{SP}}]$$

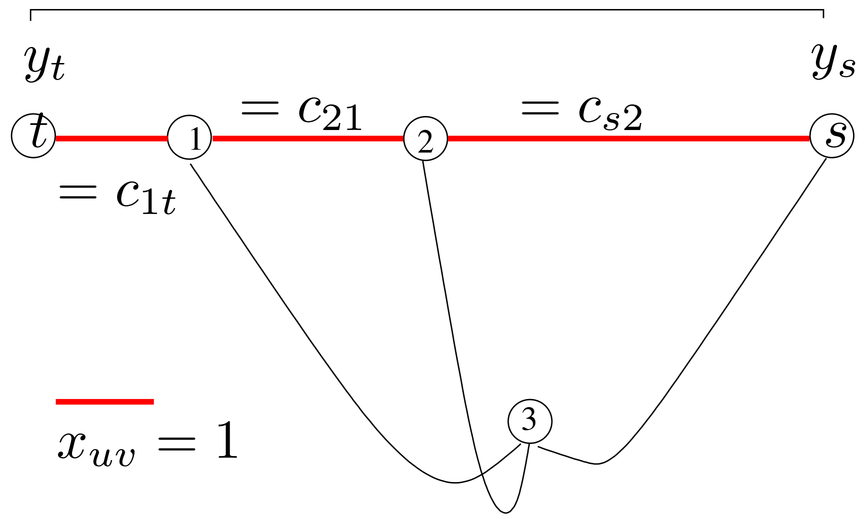
- Dual solution encodes the same solution as the "primal" (test with AMPL)

How the hell is this an SP formulation?

A mechanical algorithm

- Weighted arcs = strings as long as the weights
- Nodes = knots
- Pull nodes s, t as far as you can
- At maximum pull, strings corresponding to arcs (u, v) in SP have horizontal projections whose length is exactly c_{uv}





Open question

What is the worst-case complexity of the mechanical algorithm?

End of Lecture 9

AND END OF COURSE!

Thanks for your attention