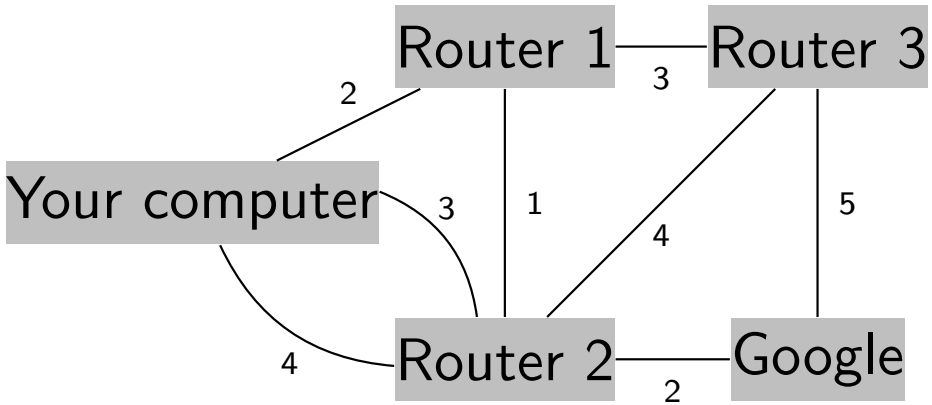
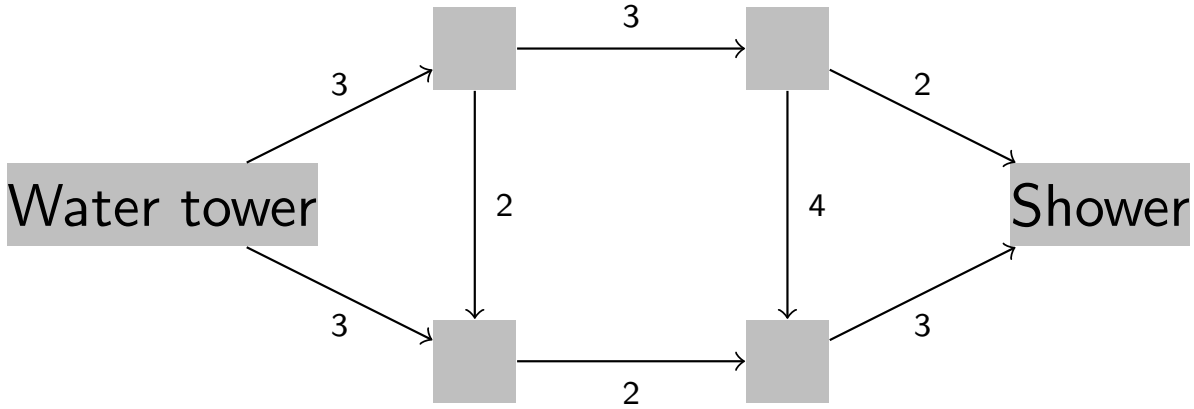
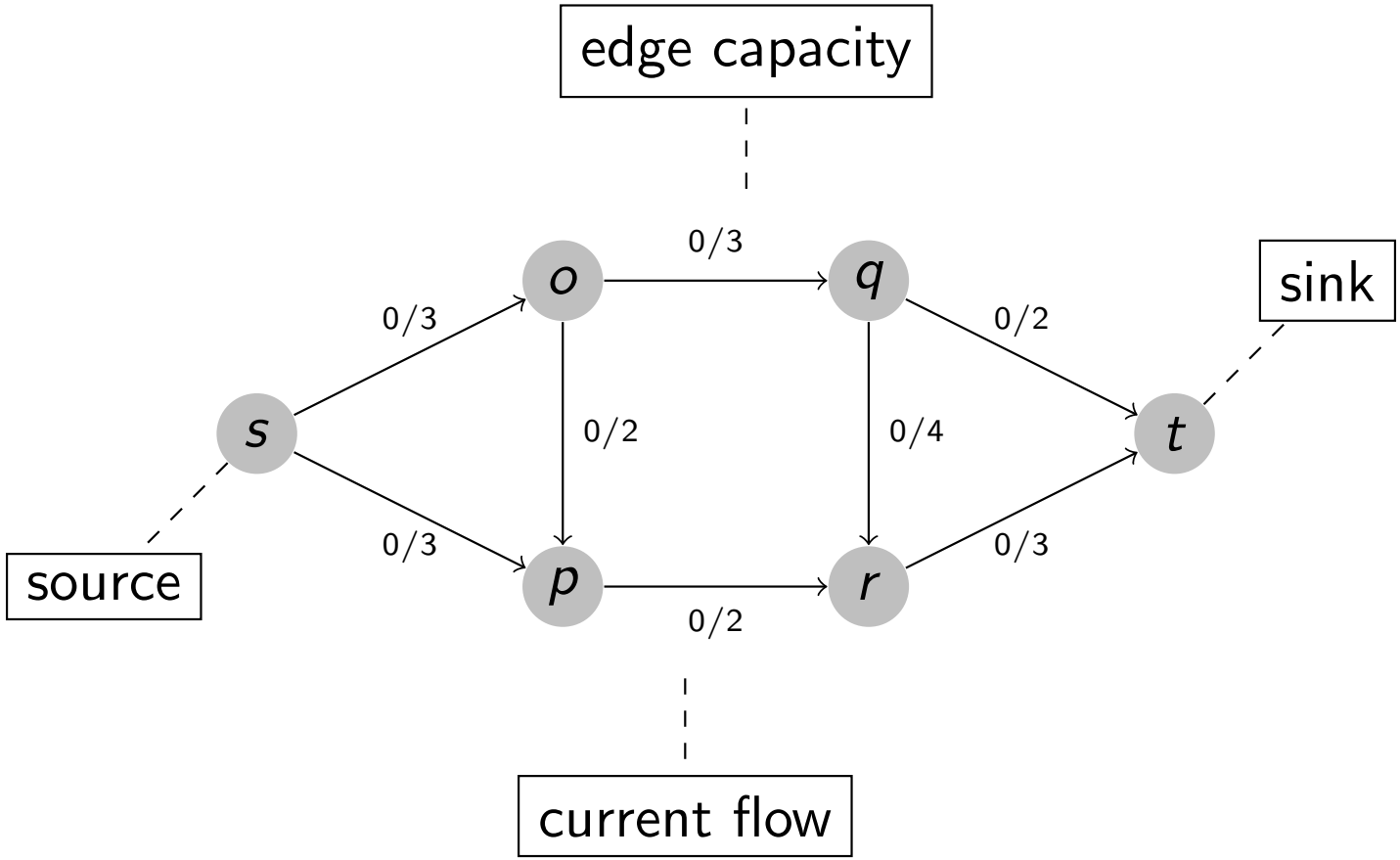
Flows

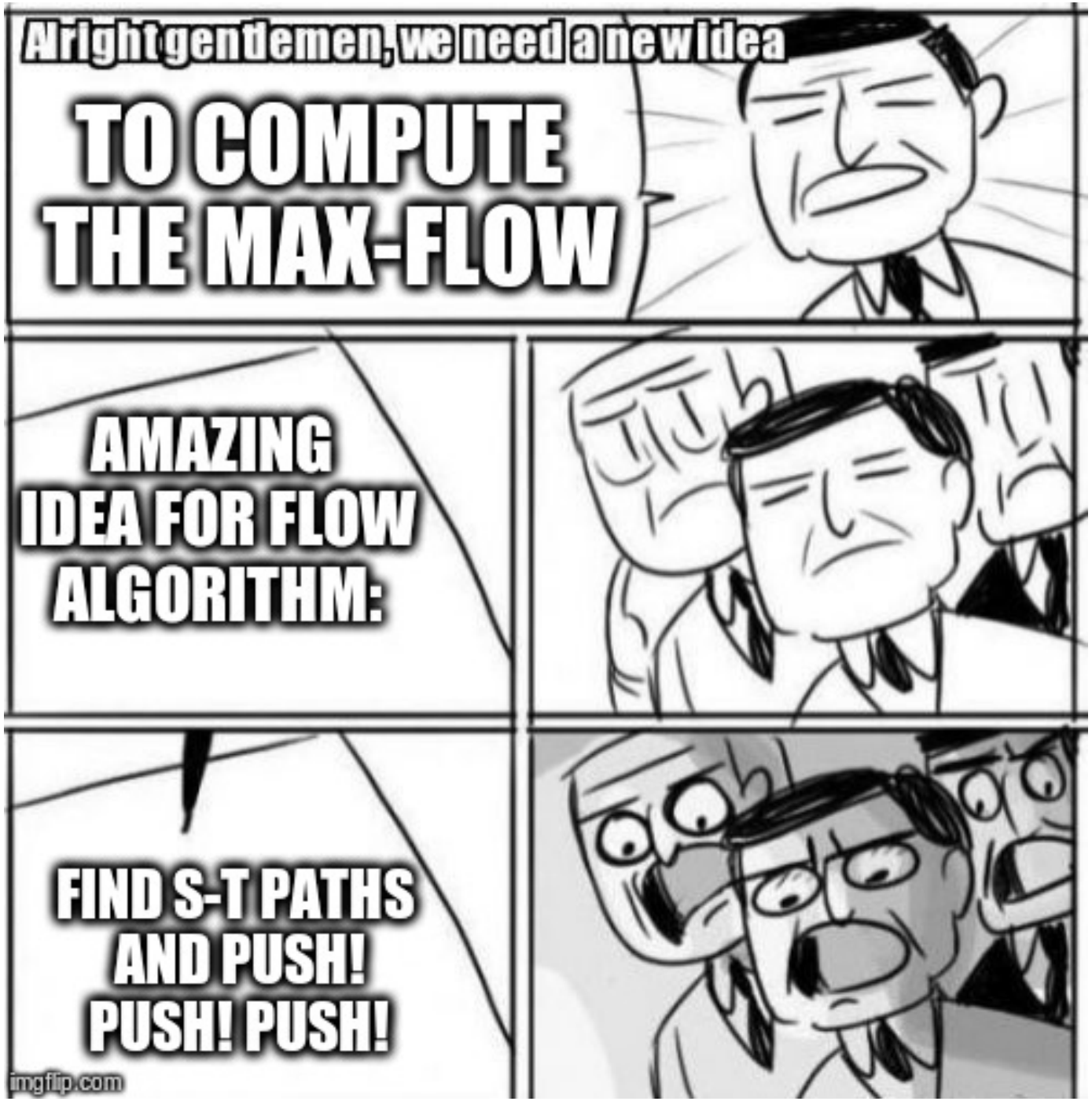

Router 1
Router 3
2
3
Your computer
3
1
4
5
4
Router 2
Google
2

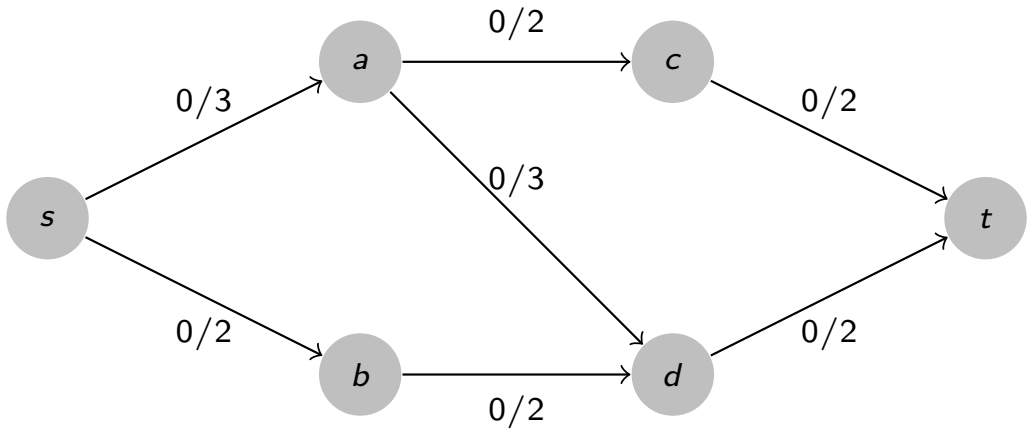

Water tower
3
3
3
2
2
4
3
2
Shower

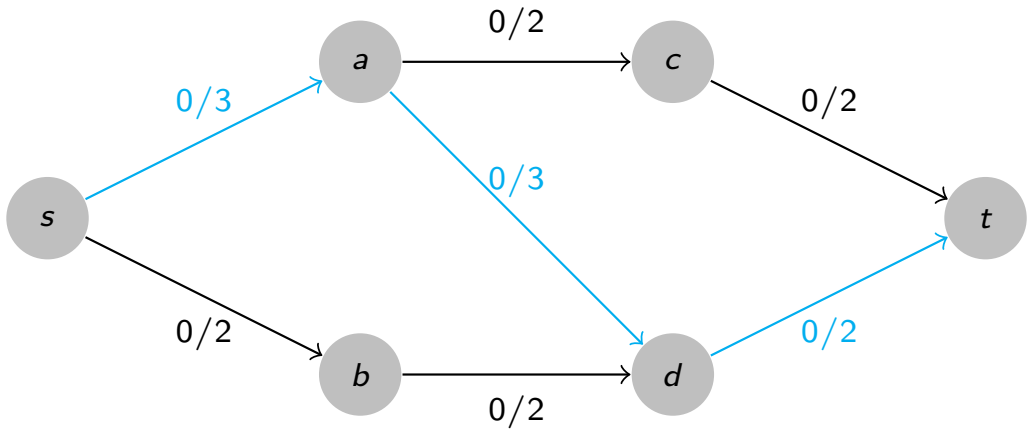
The **Maximum-flow** problem consists of finding the maximum amout of information that can be transferred from a source node into a sink node in a graph.

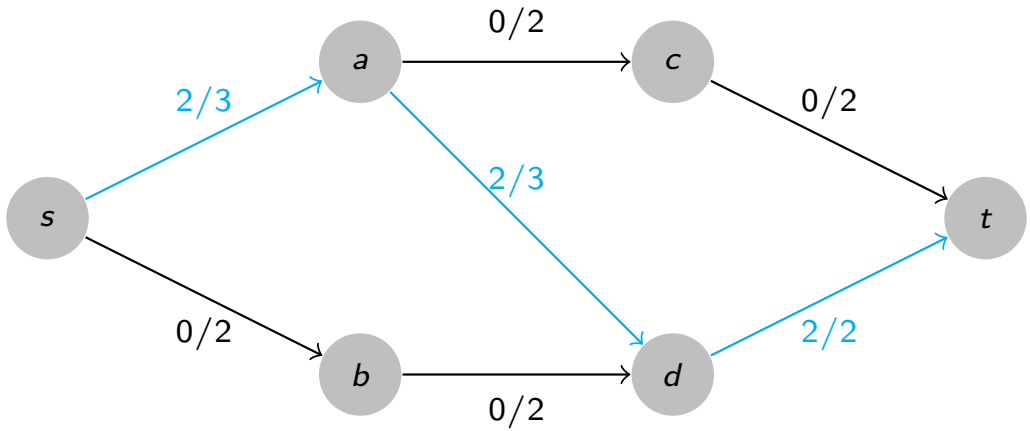
Each edge has a **capacity**, this defines the maximum amount of information that can go through that edge.

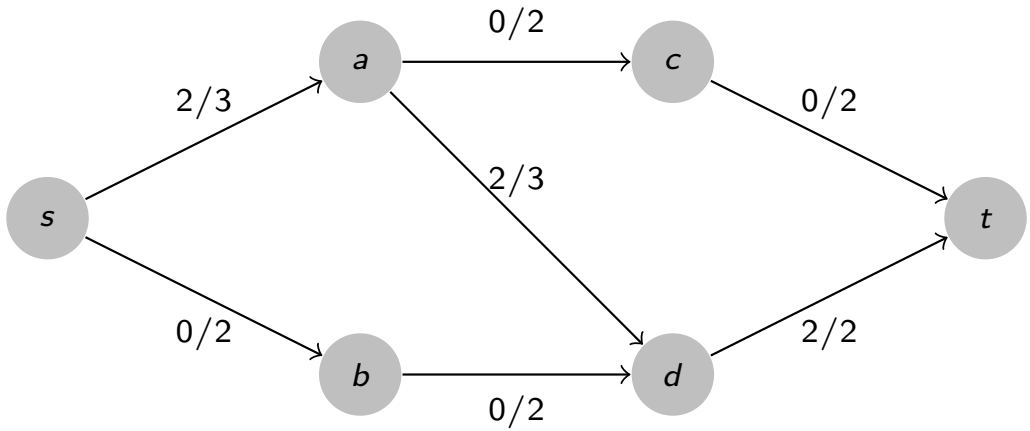

edge capacity
0/3
o
q
sink
0/3
0/2
s
0/2
0/4
t
source
0/3
p
r
0/3
0/2
current flow

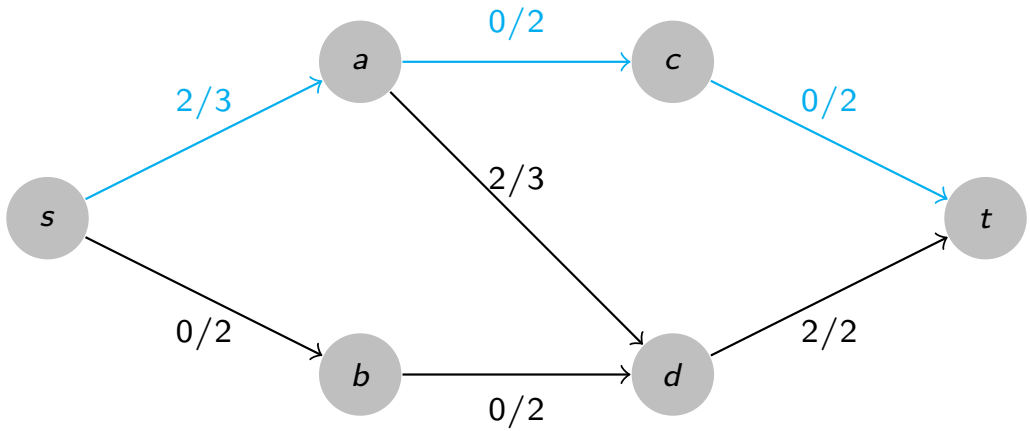

Alright gentlemen, we need a new idea
TO COMPUTE THE MAX-FLOW
AMAZING IDEA FOR FLOW ALGORITHM:
FIND S-T PATHS AND PUSH! PUSH! PUSH!
imgflip.com

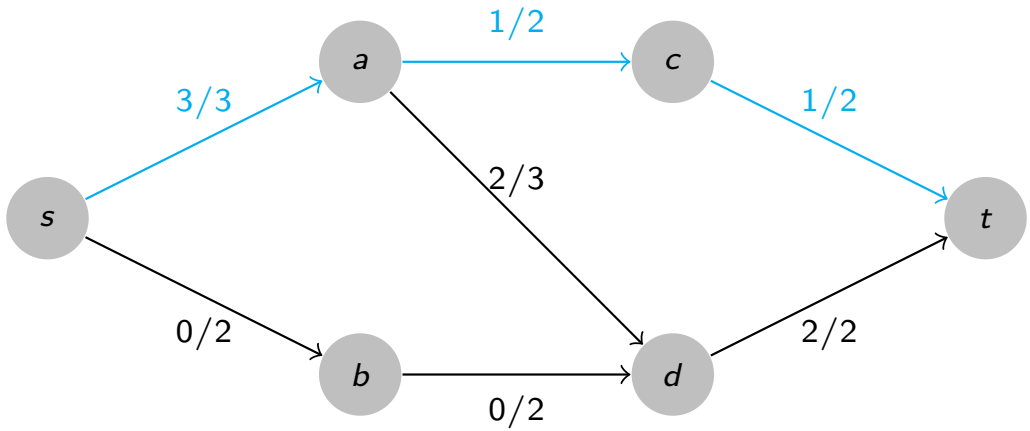

a
c
s
t
b
d
0/2
0/3
0/2
0/3
0/2
0/2
0/2

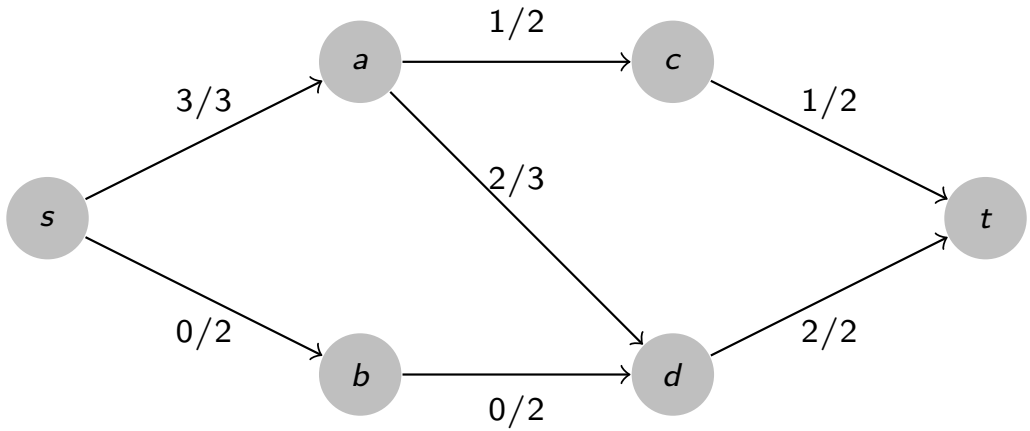

a
c
s
t
b
d
0/2
0/3
0/2
0/3
0/2
0/2
0/2

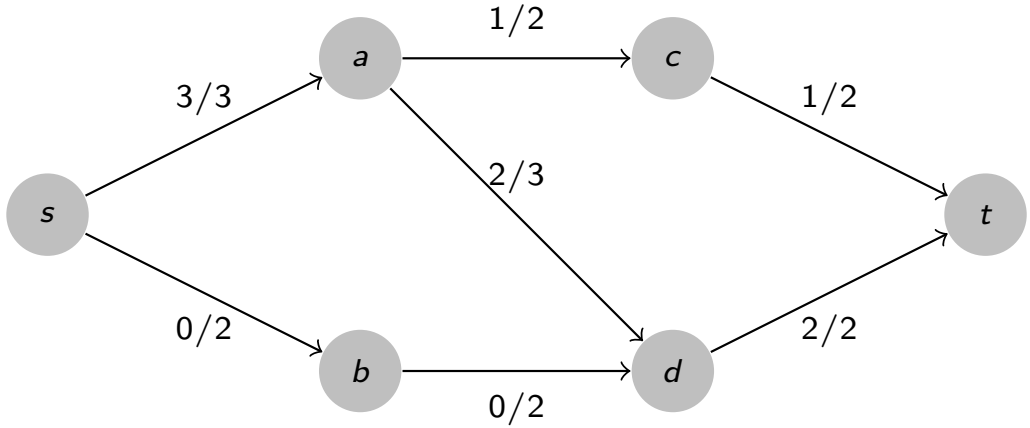

a
c
s
t
b
d
0/2
2/3
0/2
2/3
0/2
2/2
0/2


a
c
s
t
b
d
0/2
2/3
0/2
2/3
0/2
2/2
0/2

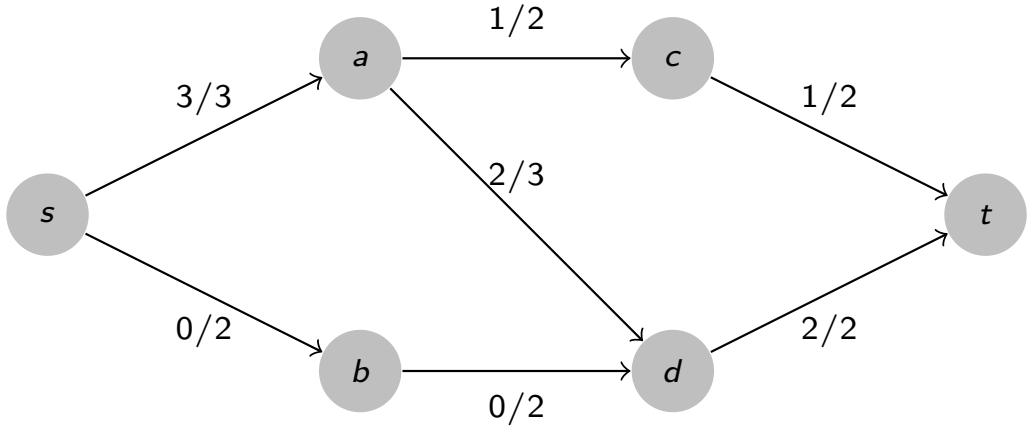

a
c
s
t
b
d
0/2
2/3
0/2
2/3
0/2
2/2
0/2


a
c
s
t
b
d
1/2
3/3
1/2
2/3
0/2
2/2
0/2


s
a
b
c
d
t
3/3
0/2
1/2
2/3
0/2
1/2
2/2

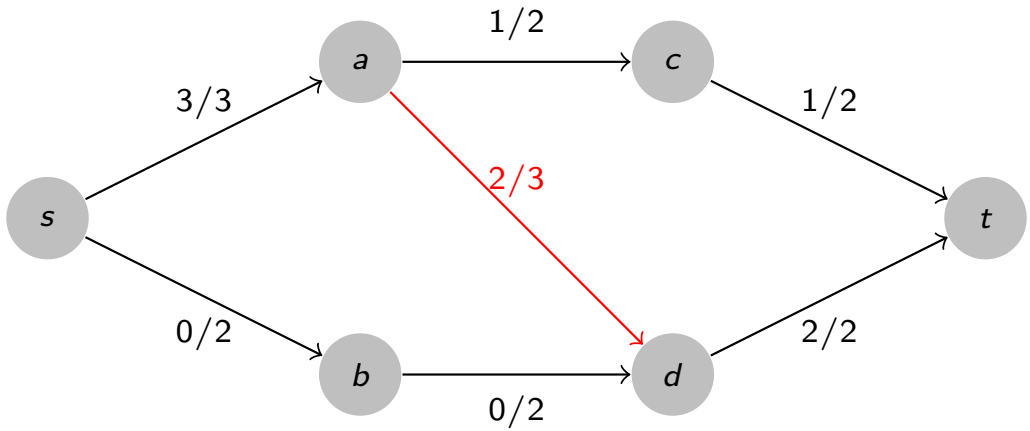


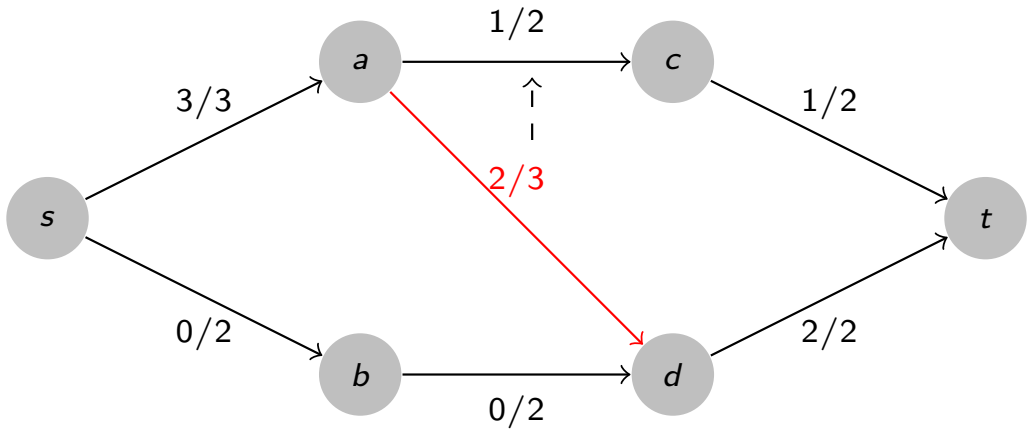
No more paths. Found flow of value 3. **Not** optimal!

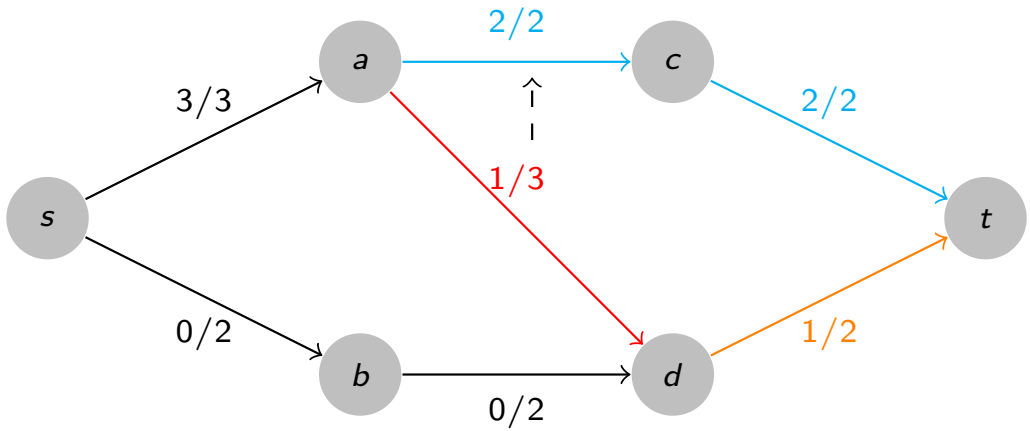


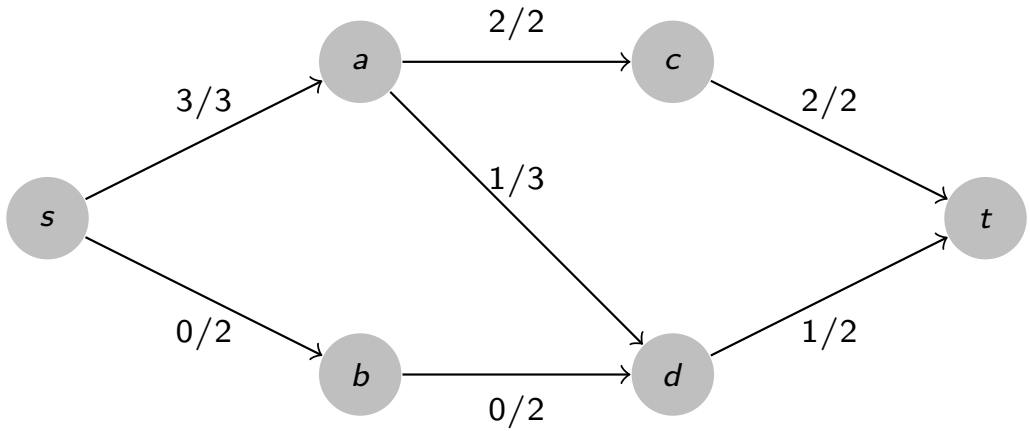
No more paths. Found flow of value 3. **Not** optimal!

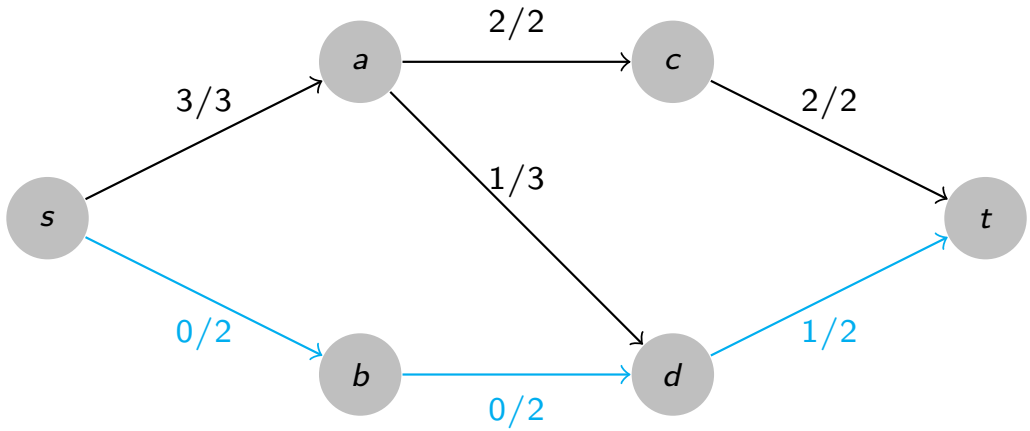
We would like to be able to reconsider decisions.

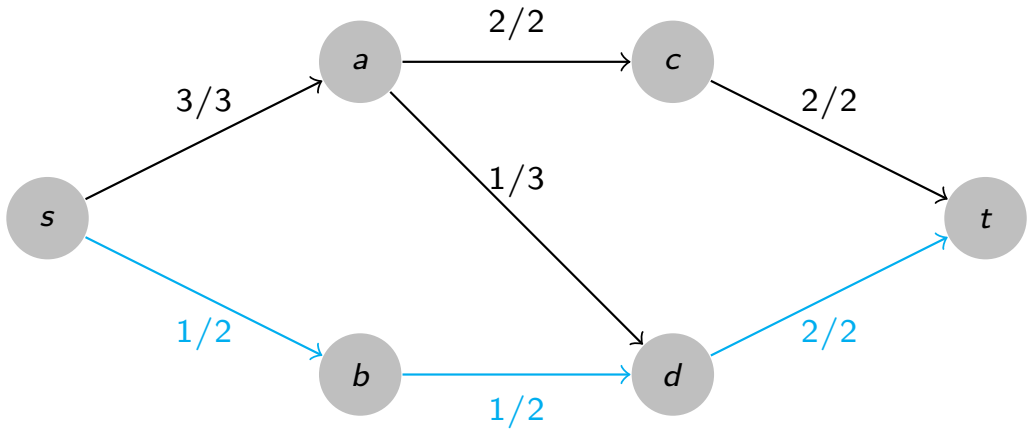

a
c
s
t
b
d
1/2
3/3
1/2
2/3
0/2
2/2
0/2

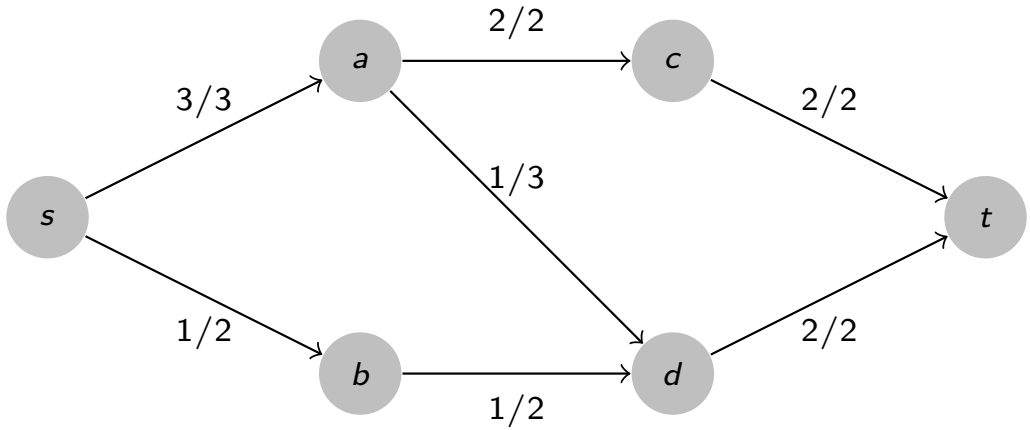

a
c
s
t
b
d
1/2
3/3
1/2
2/3
0/2
2/2
0/2


2/2
a
c
3/3
2/2
1/3
s
t
0/2
1/2
b
d
0/2


s
a
b
c
d
t
3/3
0/2
2/2
1/3
0/2
2/2
1/2

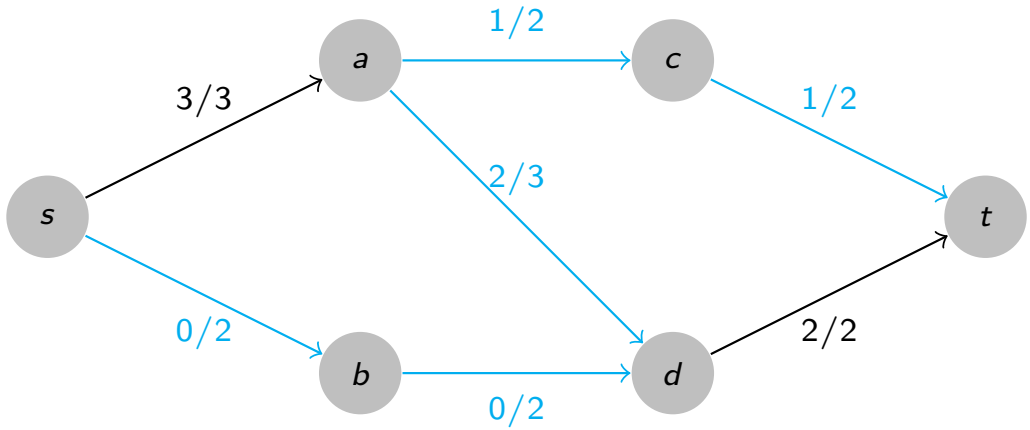

a
c
s
t
b
d
2/2
3/3
2/2
1/3
0/2
1/2
0/2


a
c
s
t
b
d
2/2
3/3
2/2
1/3
1/2
2/2
1/2

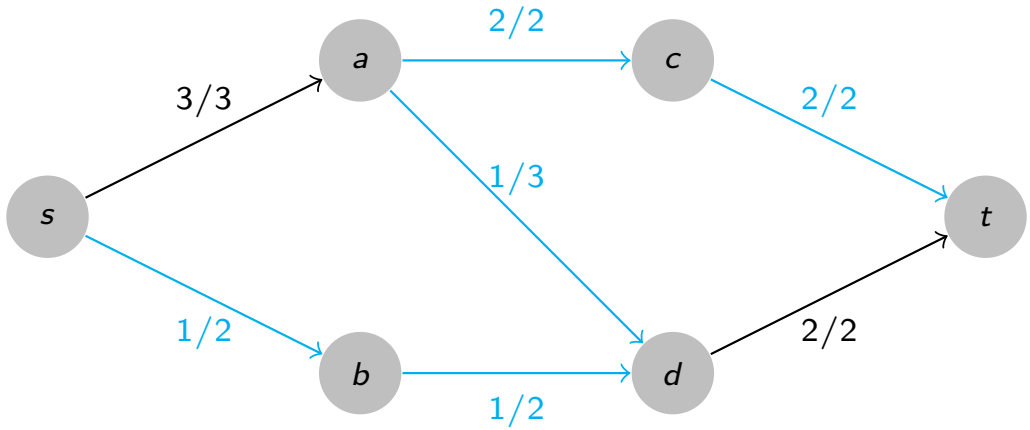


- No more paths. Found flow of value 4. Optimal!

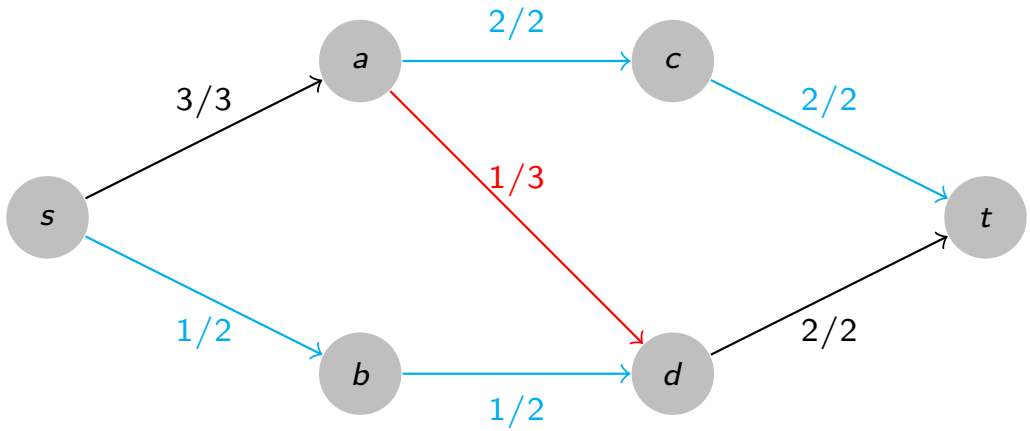
What we did corresponds to pushing flow on the following path:



What we did corresponds to pushing flow on the following path:

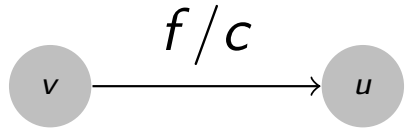


What we did corresponds to pushing flow on the following path:

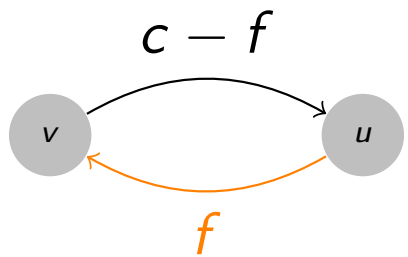


This leads to the definition of **residual graph**.

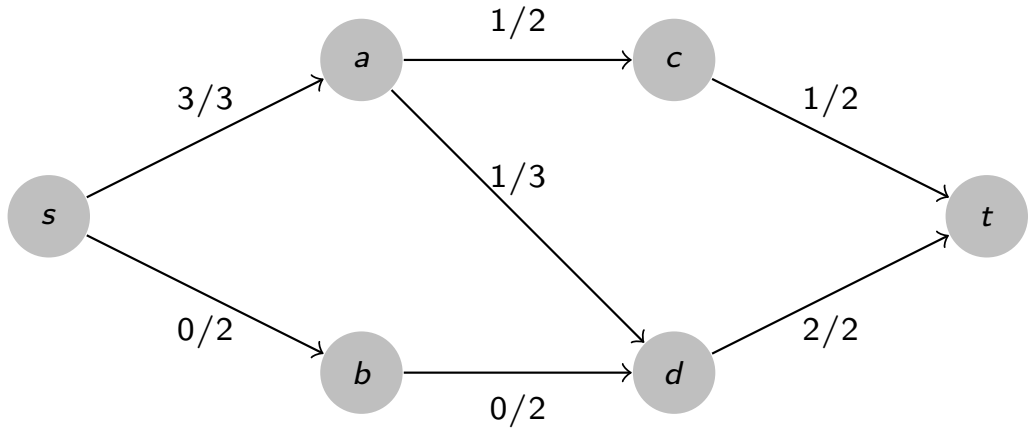
Original



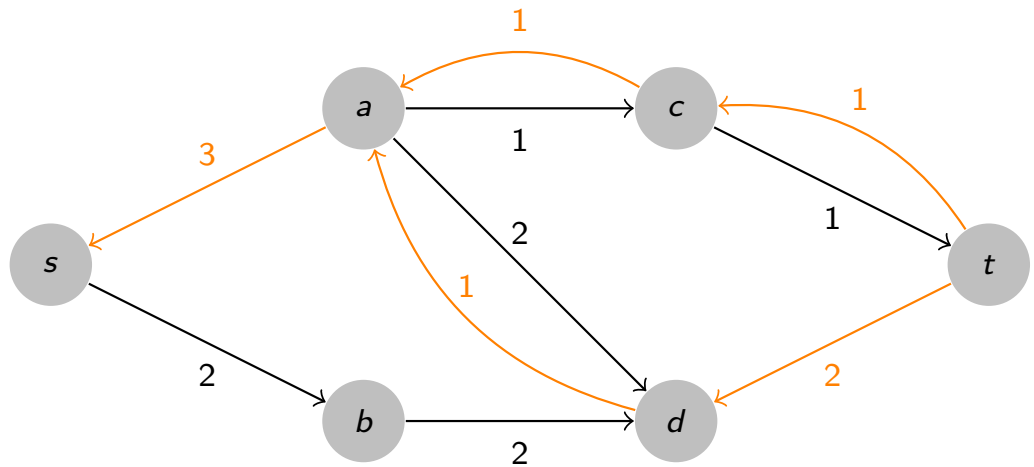
Residual



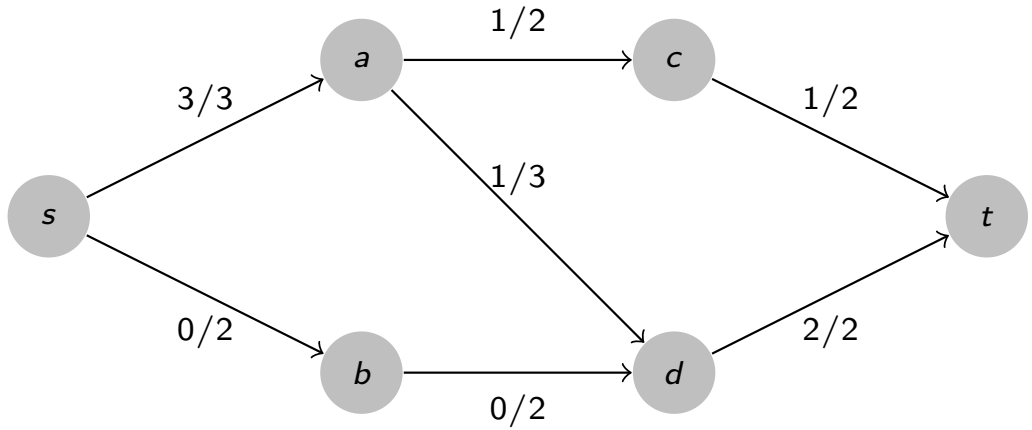
original



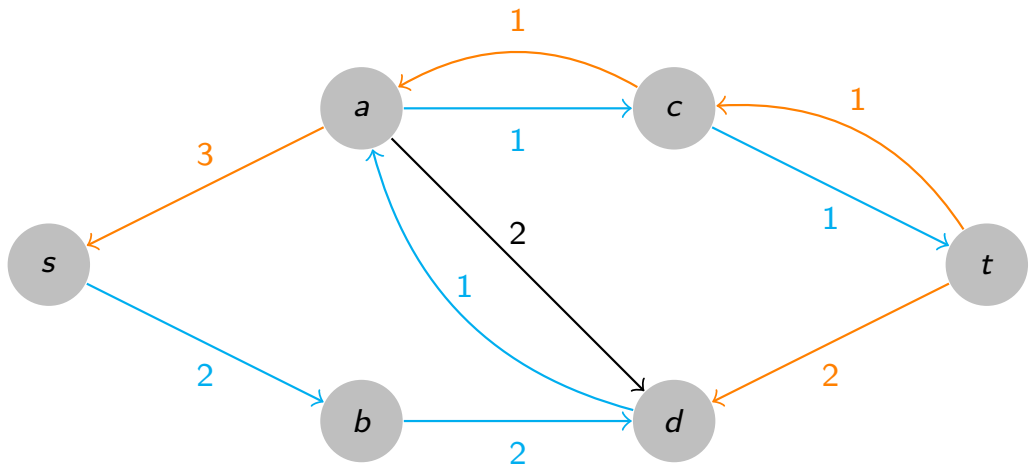
residual



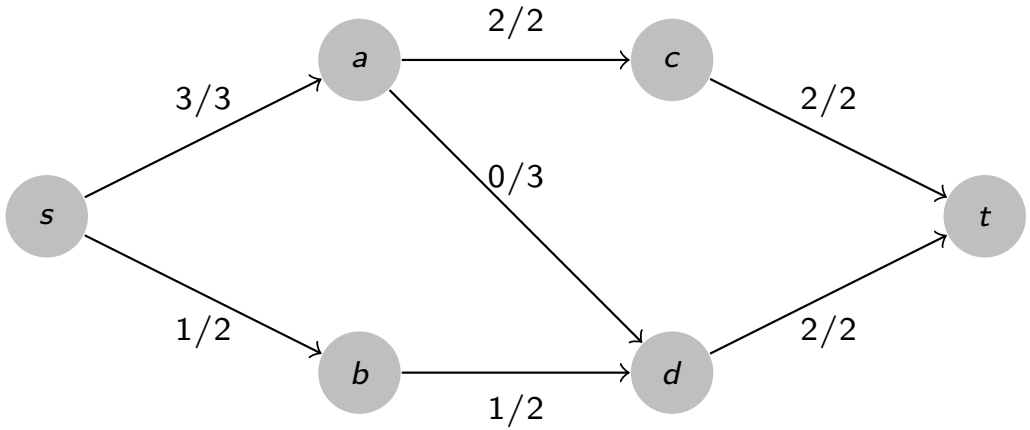
original



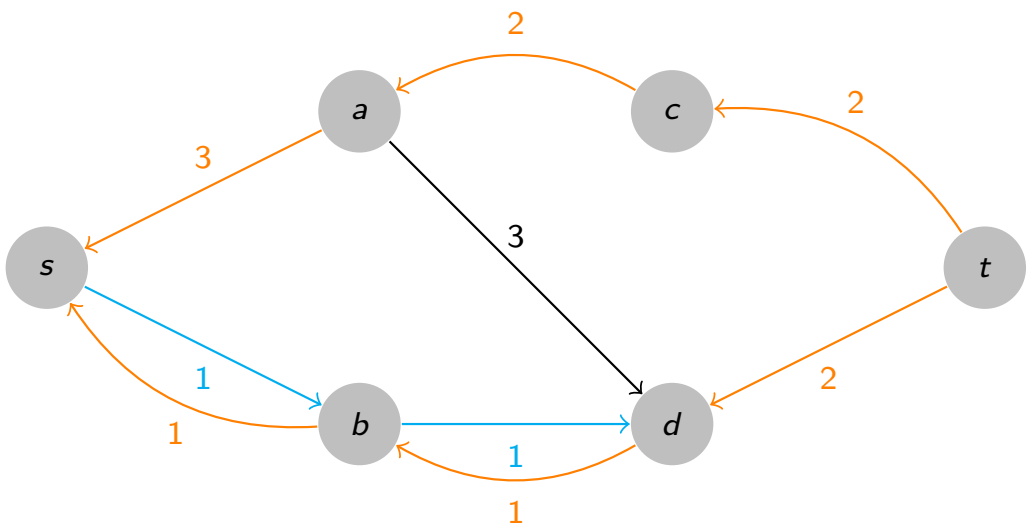
residual



original



residual



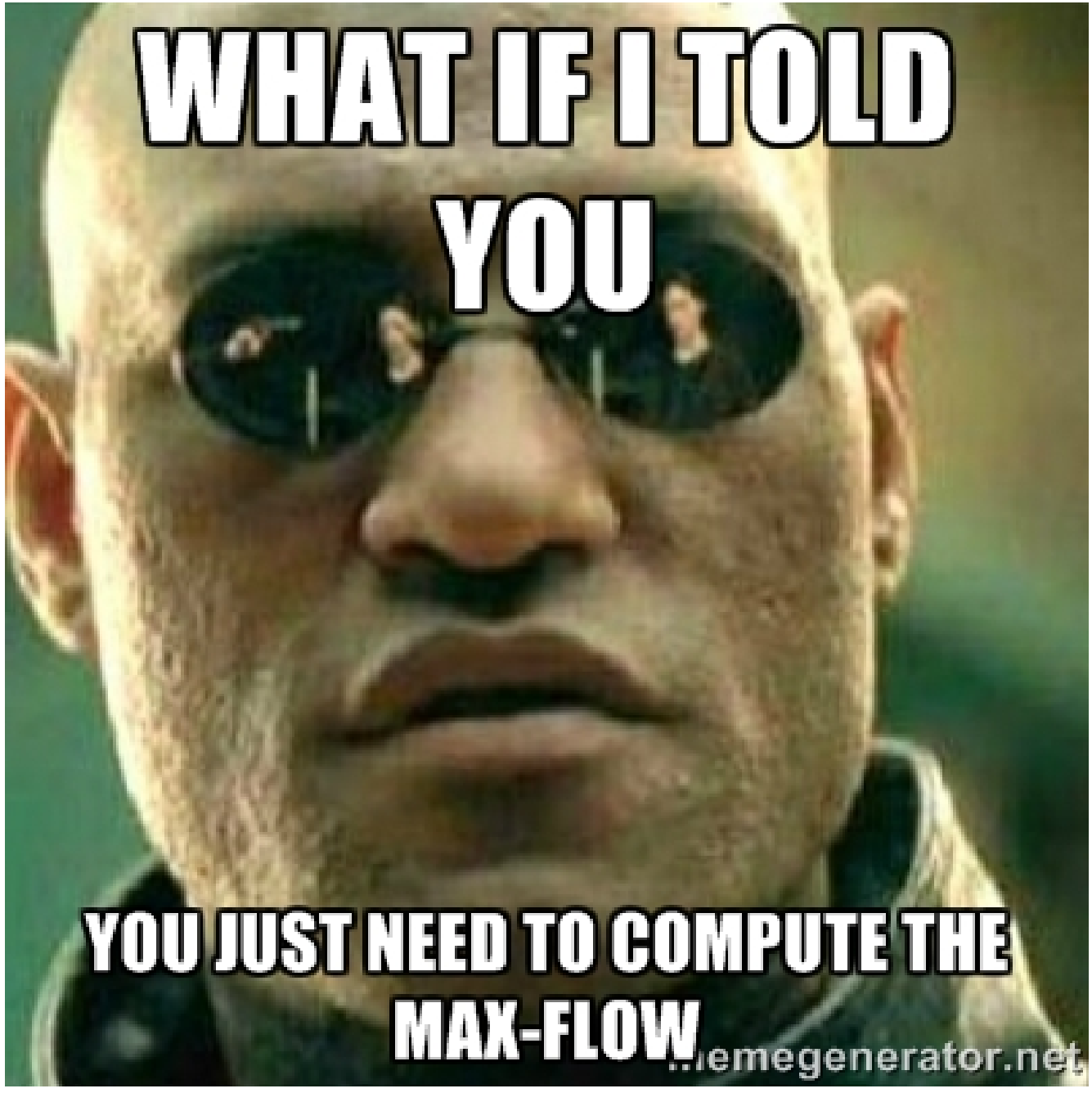
- The algorithm works directly on the residual graph G_r.
  1. While there is an s-t path p in G_r
  2. Push as much flow as possible through p
  3. Update the residual graph G_r.
- One can show that this works.
- If we find paths with BFS we get an $O(|V||E|^2)$ algorithm (aka. Edmonds-Karp algorithm).
- If we use DFS we get time complexity $O(|E|f^*)$ (aka. Ford-Fulkerson algorithm).

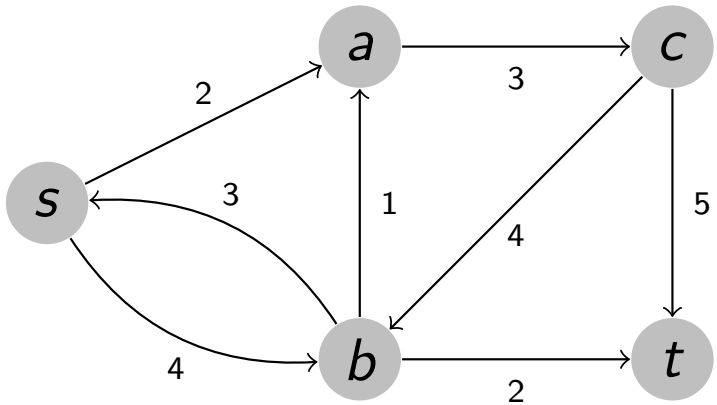
Minimum cut

The **Minimum-cut** problem consits of finding a set of edges whose removal disconnects two nodes in the network and has minimal cost. The cost of removing an edge is given by its weight.

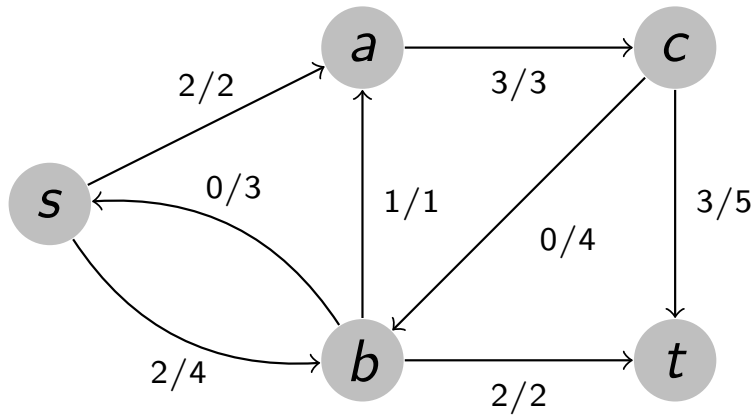
The **Minimum-cut** problem consits of finding a set of edges whose removal disconnects two nodes in the network and has minimal cost. The cost of removing an edge is given by its weight.

Ideas?...

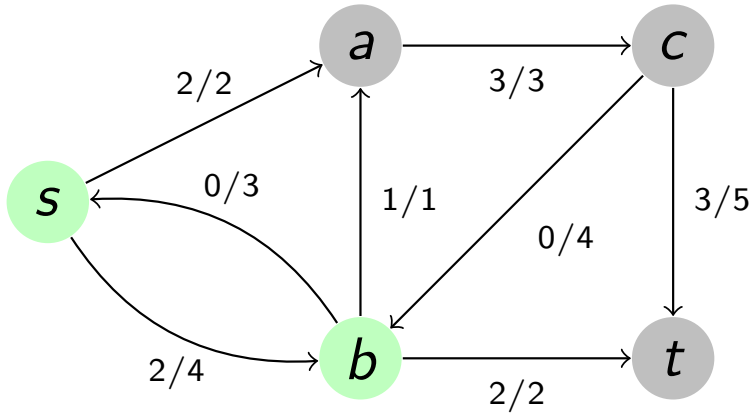

WHAT IF I TOLD YOU
YOU JUST NEED TO COMPUTE THE MAX-FLOW
memegenerator.net


a
c
s
b
t
2
3
3
1
4
5
4
2

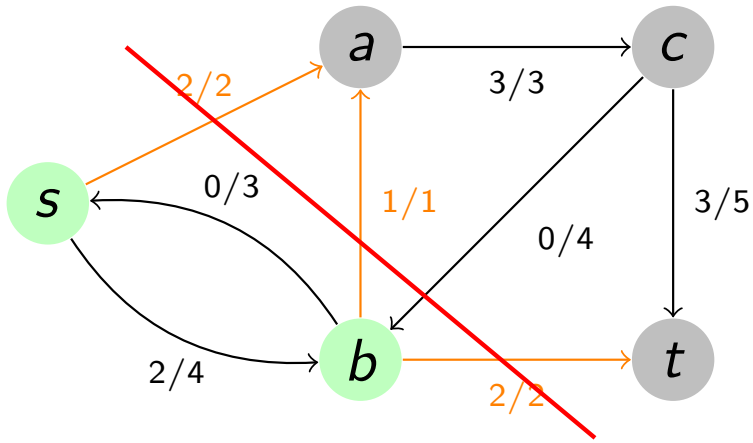
Max-flow:



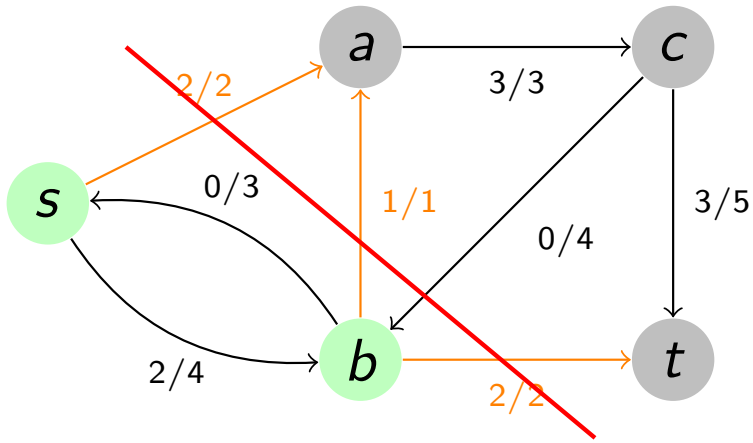
Look at the nodes reachable from s in the residual graph.



The minimum cut consists of the edges between the two sets.

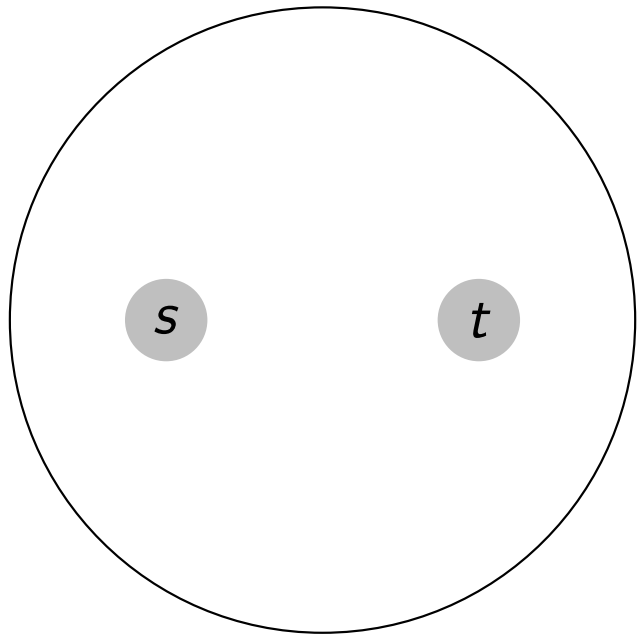


The minimum cut consists of the edges between the two sets.

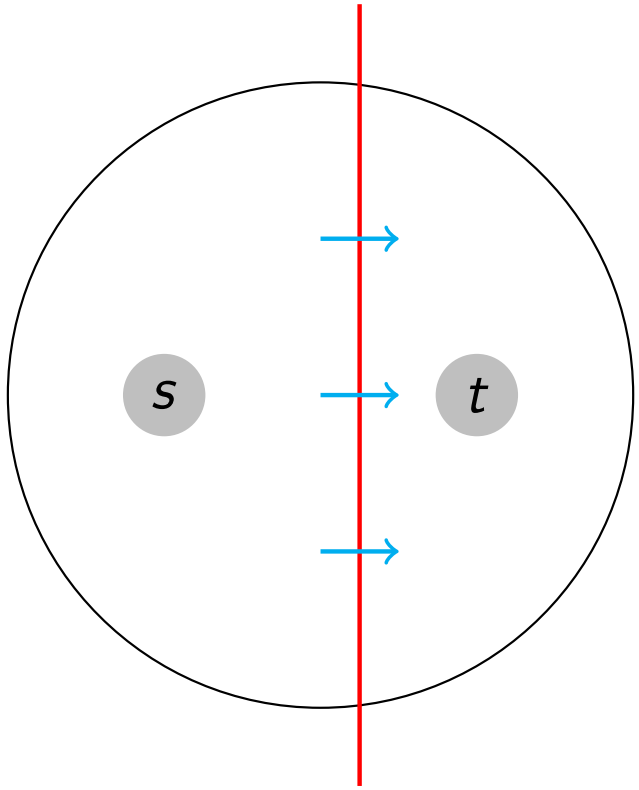


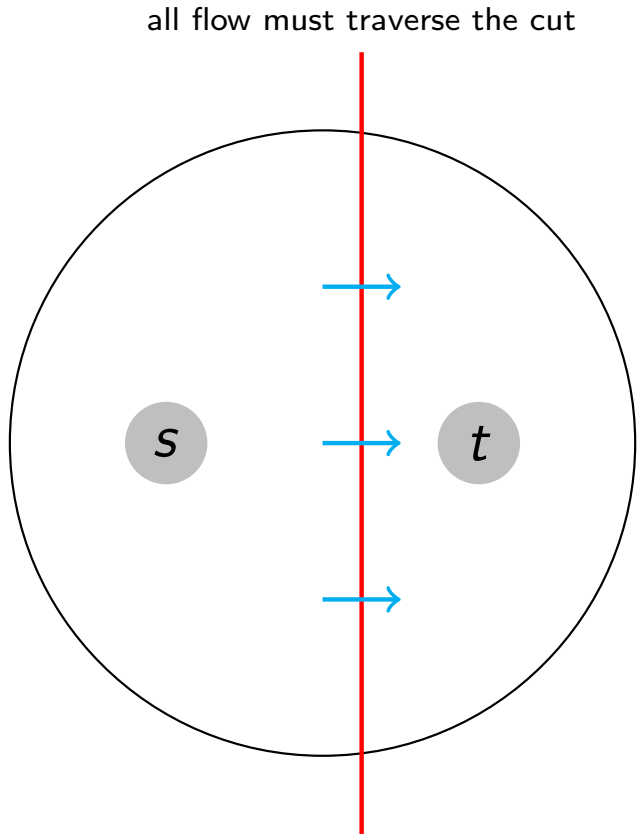
Also known as the **Min-Cut Max-Flow theorem**:

$$\text{Max-Flow}(s, t, G) = \text{Min-Cut}(s, t, G)$$


s
t

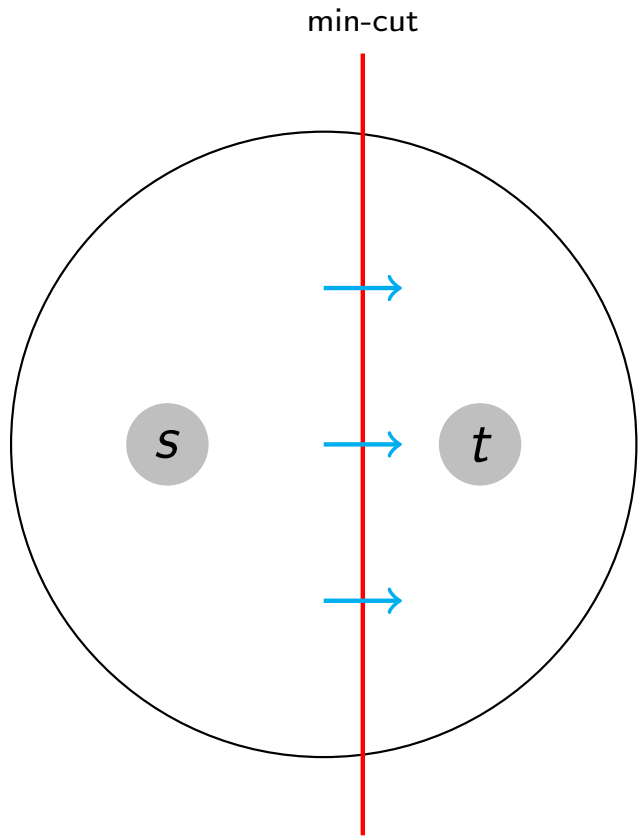
all flow must traverse the cut



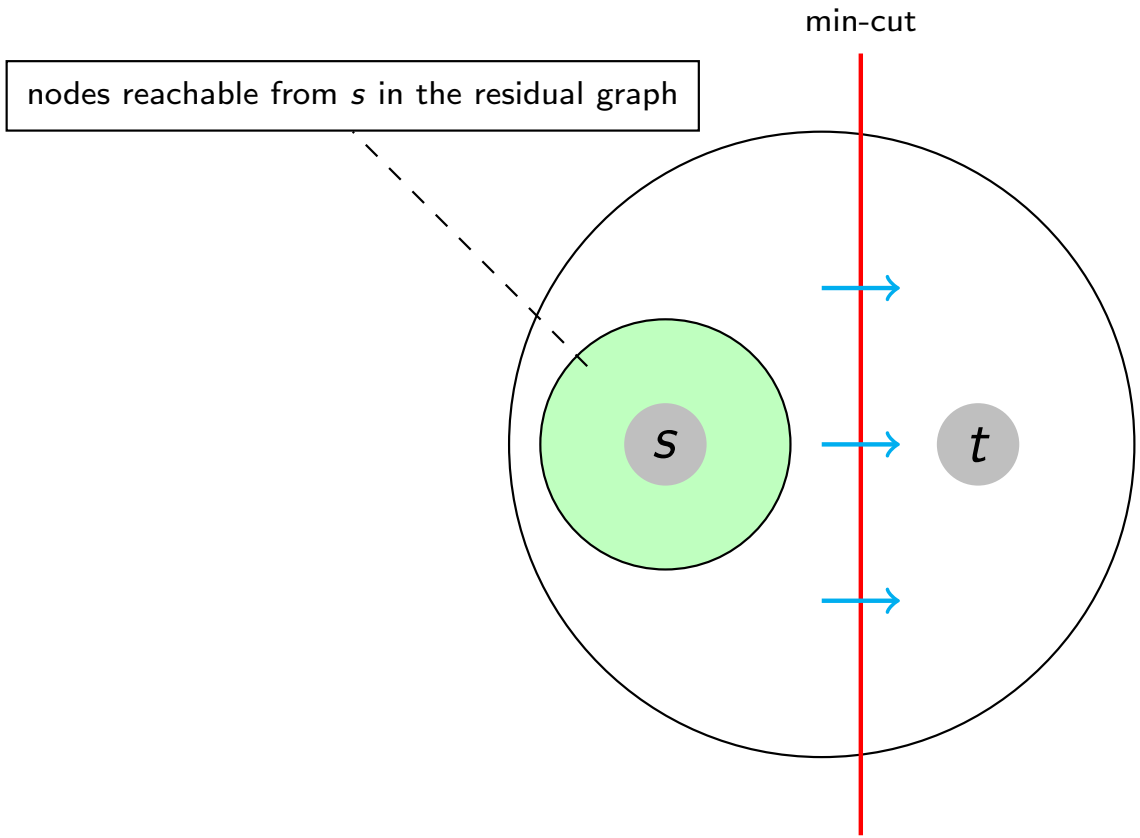


$$\text{Max-Flow}(s, t, G) \leq \text{Min-Cut}(s, t, G)$$

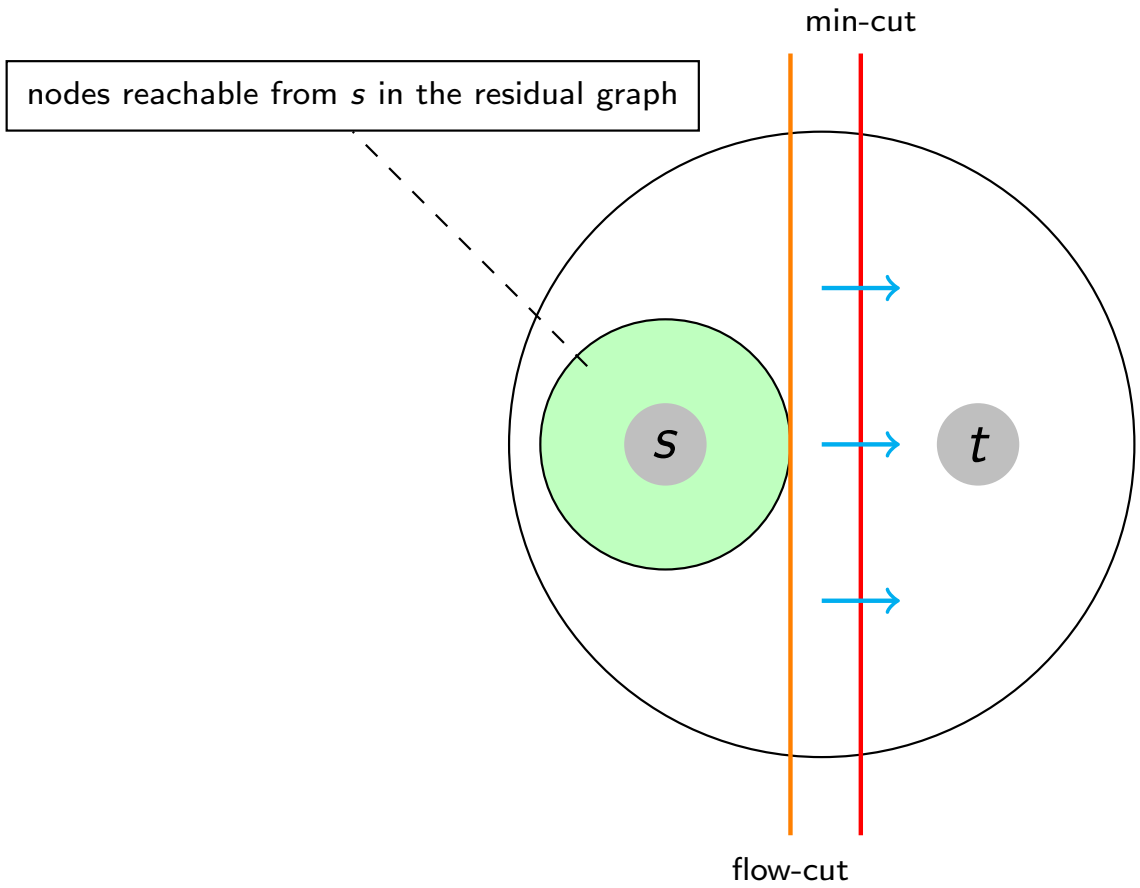
Suppose that Max-Flow$(s, t, G) <$ Min-Cut(s, t, G)



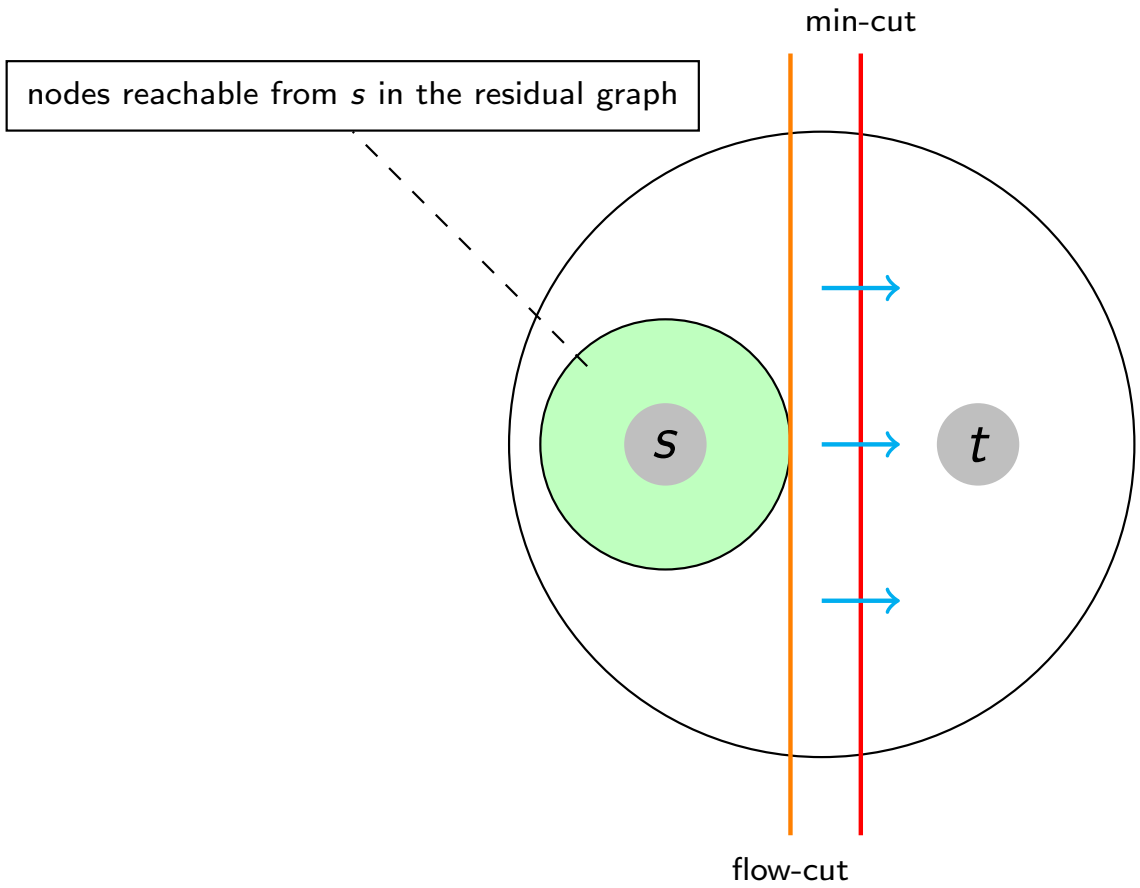
Suppose that Max-Flow$(s, t, G) <$ Min-Cut(s, t, G)



Suppose that Max-Flow$(s, t, G) <$ Min-Cut(s, t, G)

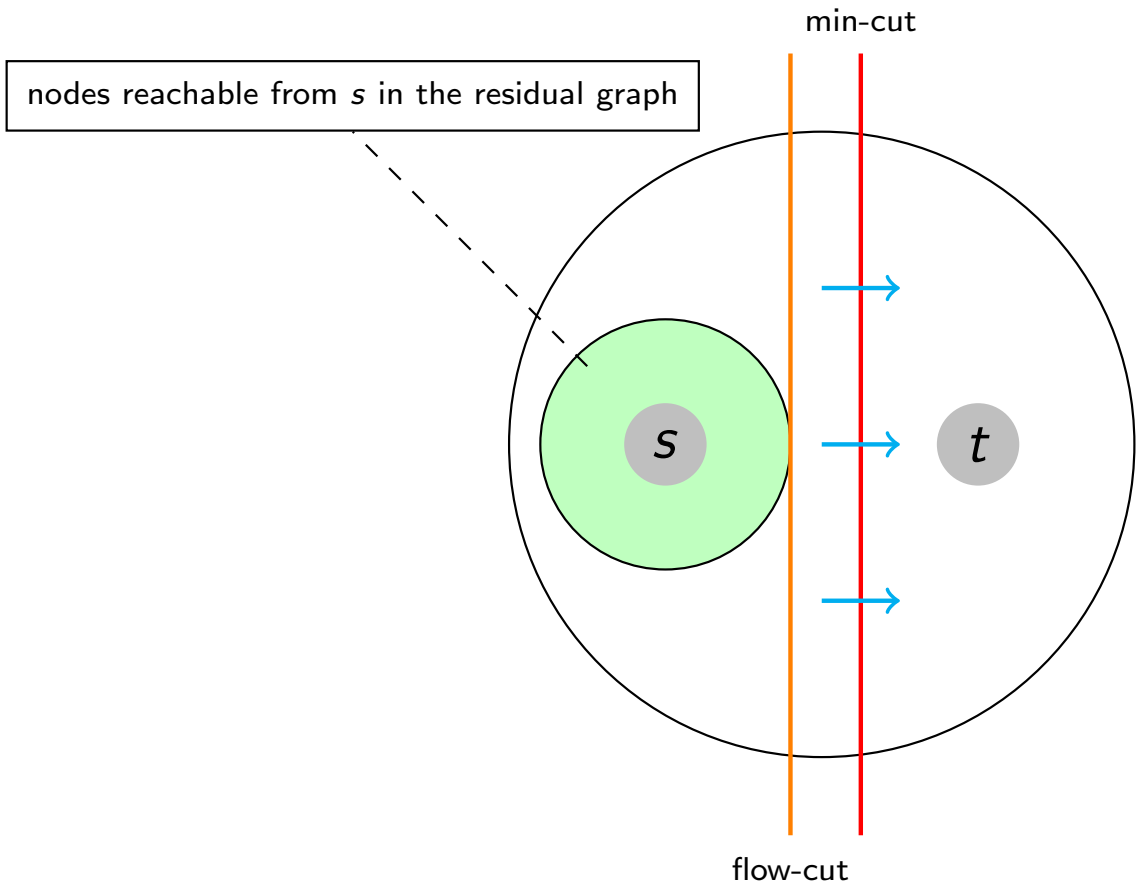


Suppose that Max-Flow$(s, t, G) <$ Min-Cut(s, t, G)



$$\text{flow-cut} = \text{flow} < \text{Min-Cut(s, t, G)}$$

Suppose that Max-Flow$(s, t, G) <$ Min-Cut(s, t, G)

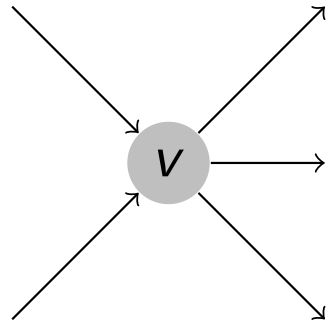


flow-cut $=$ flow $<$ Min-Cut(s, t, G)

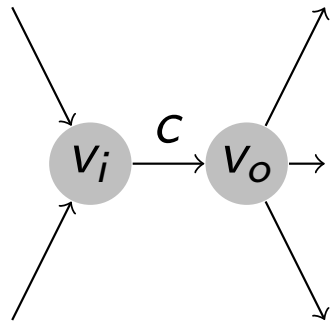
CONTRADICTION

Related problems and modeling

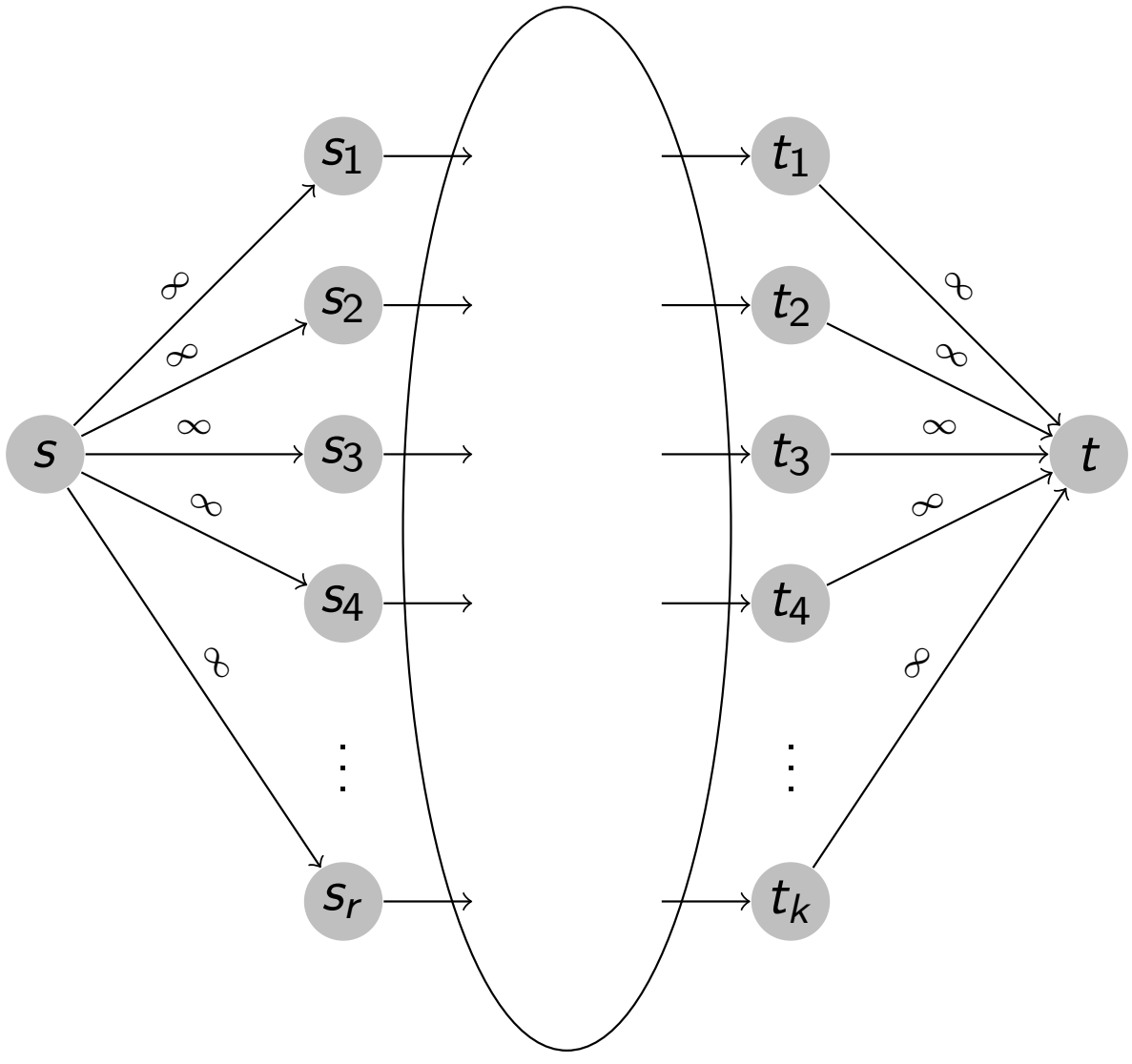
Node capacities:



Split nodes:

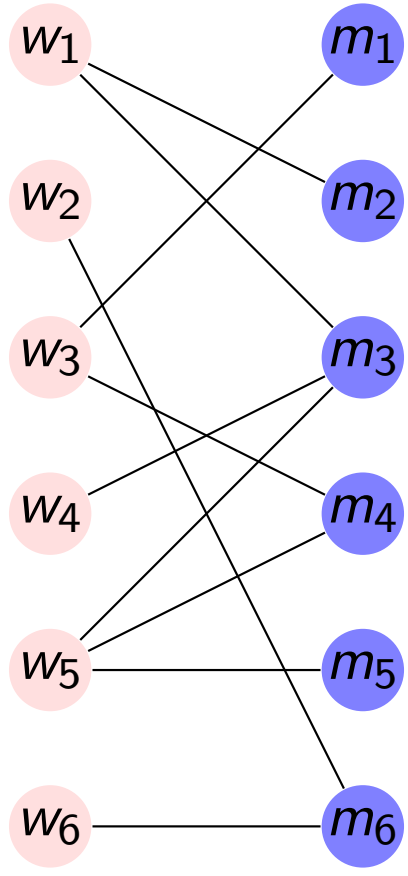


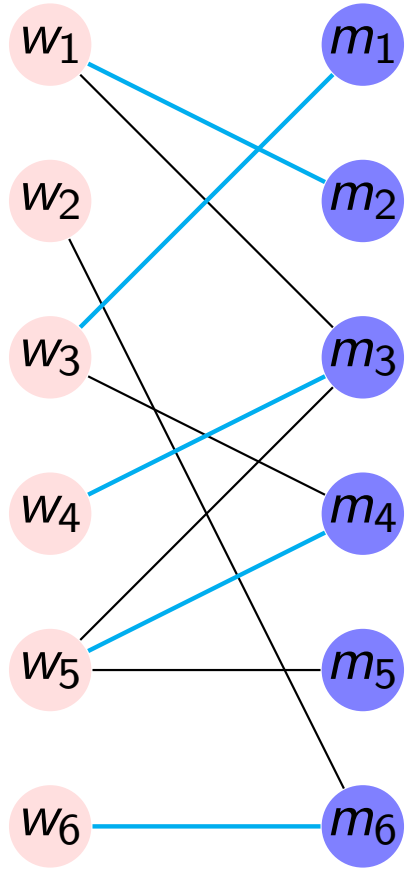
Multiple sources / sinks:

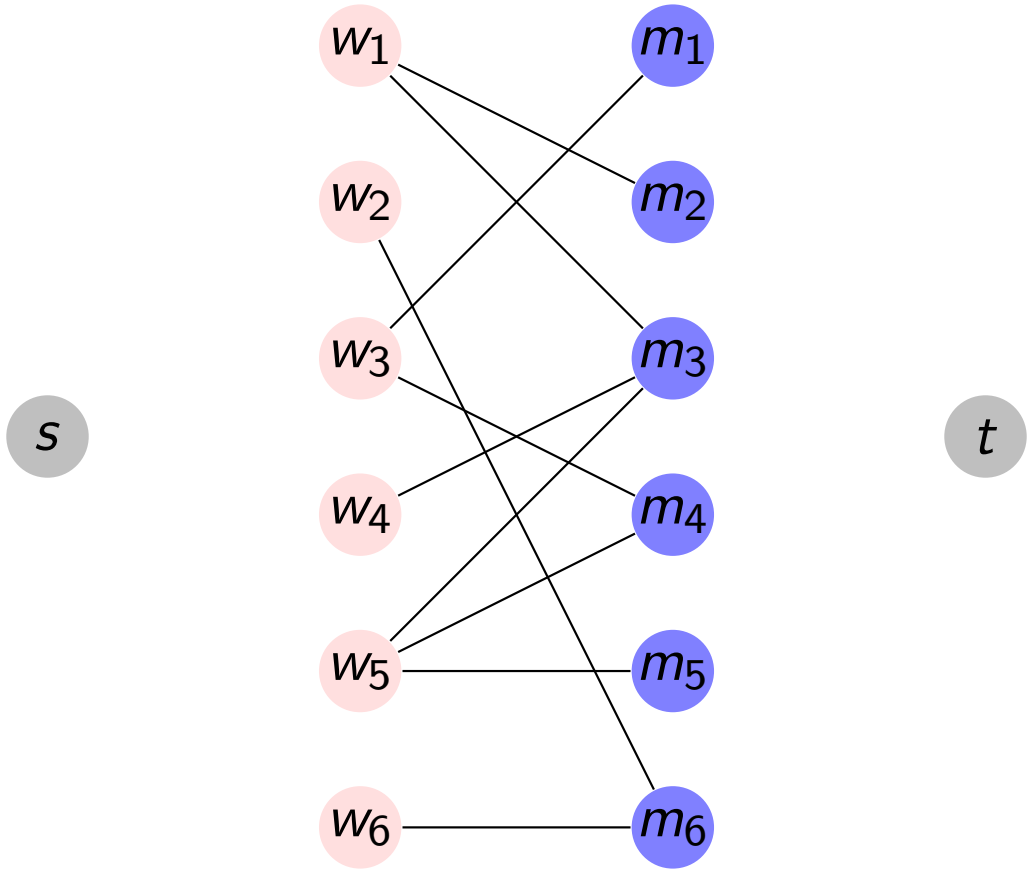


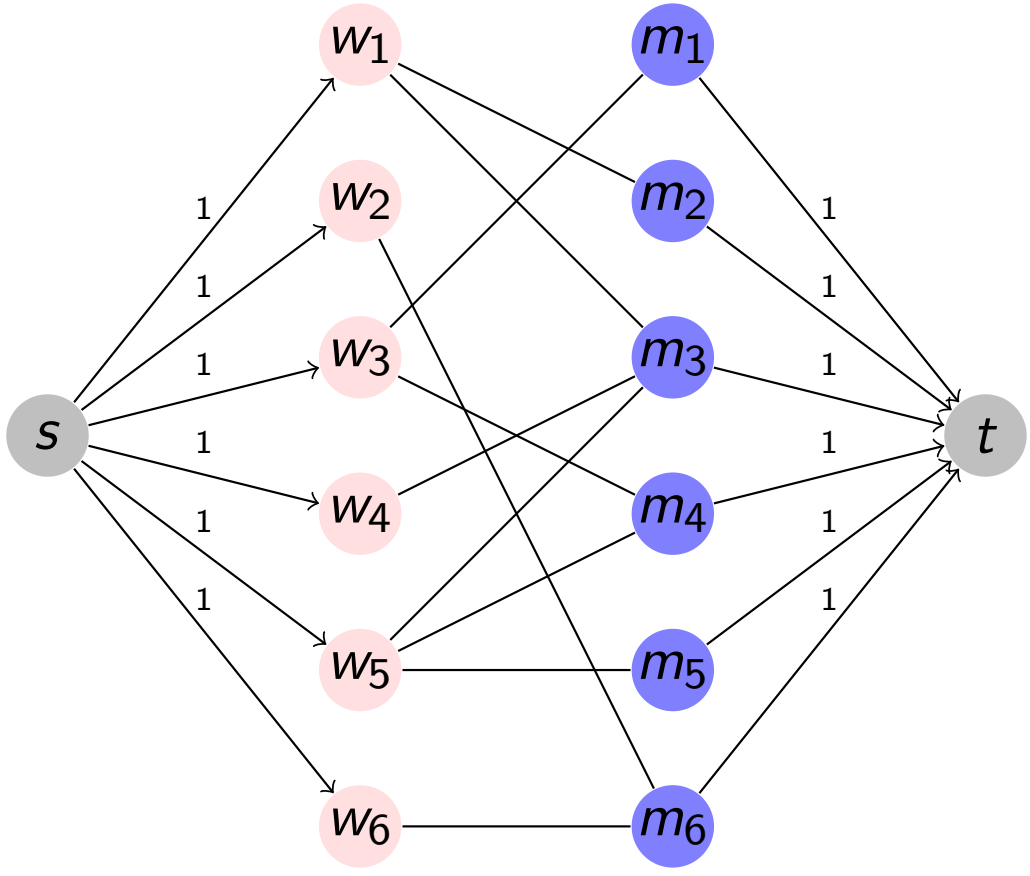
Maximum bipartite matching:

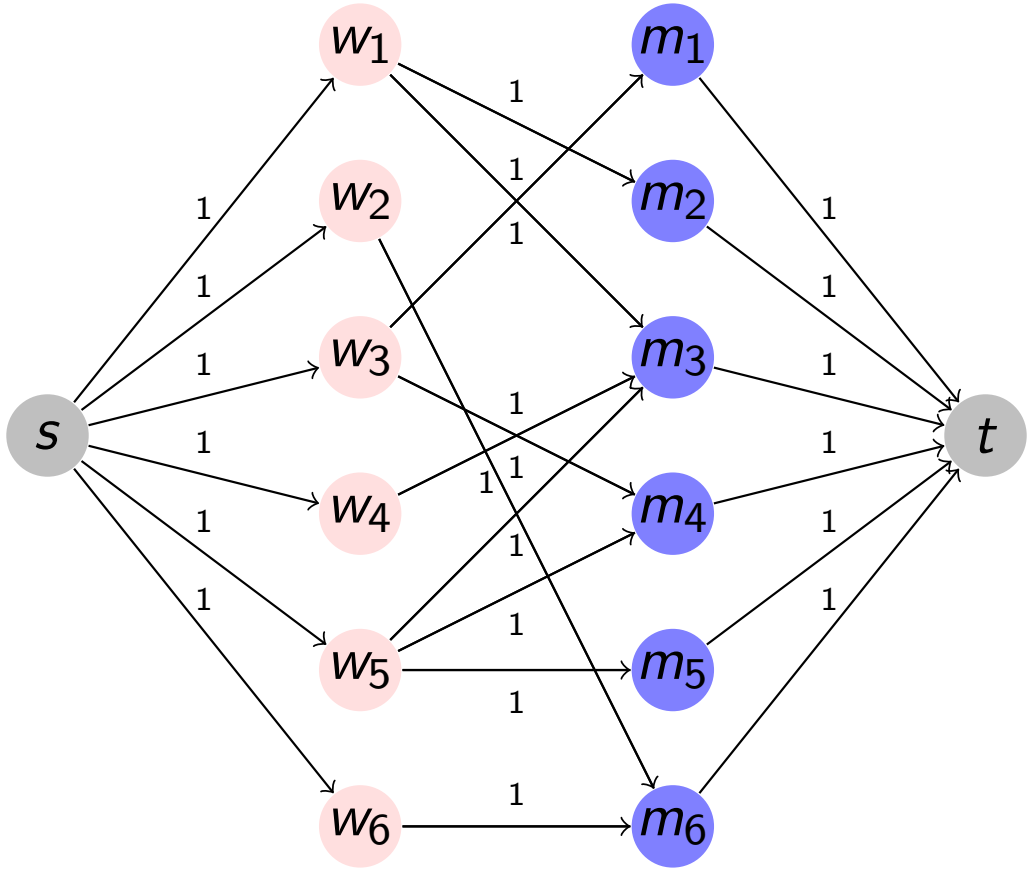
There are N women and M men. Each woman is interested in some of the men. The goal is to find maximum number of couples that can be formed.

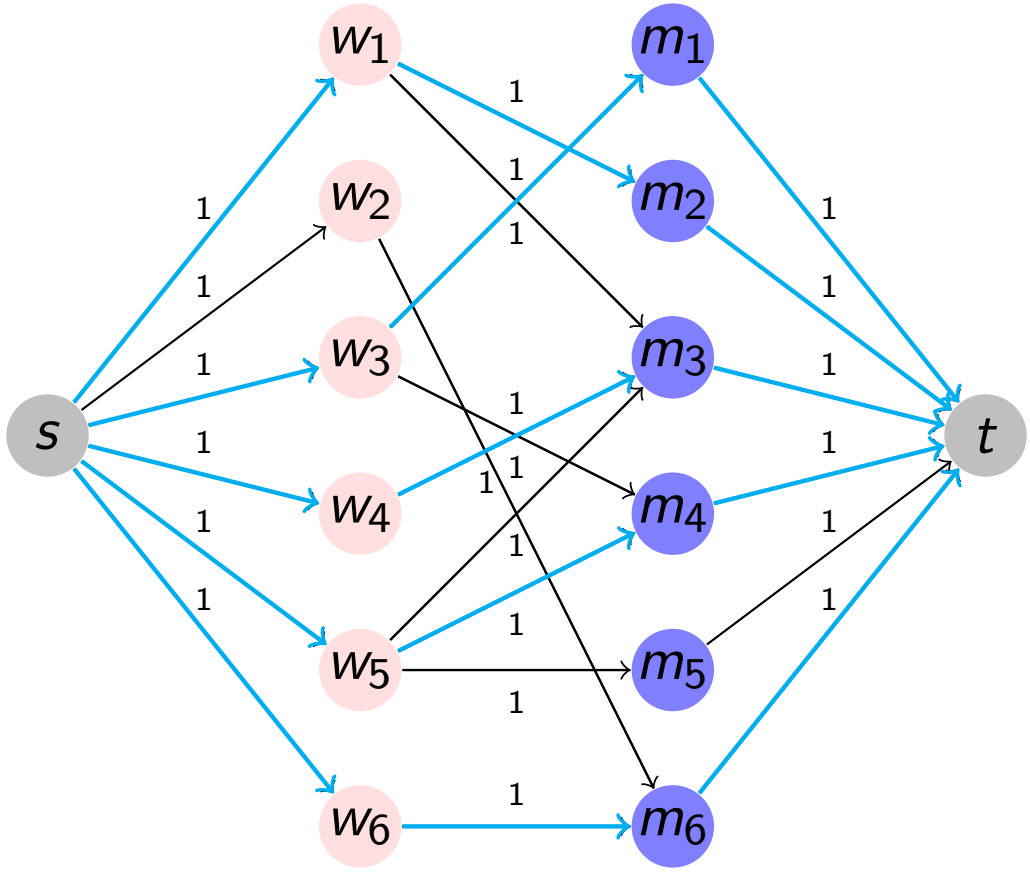

w_1
w_2
w_3
w_4
w_5
w_6
m_1
m_2
m_3
m_4
m_5
m_6

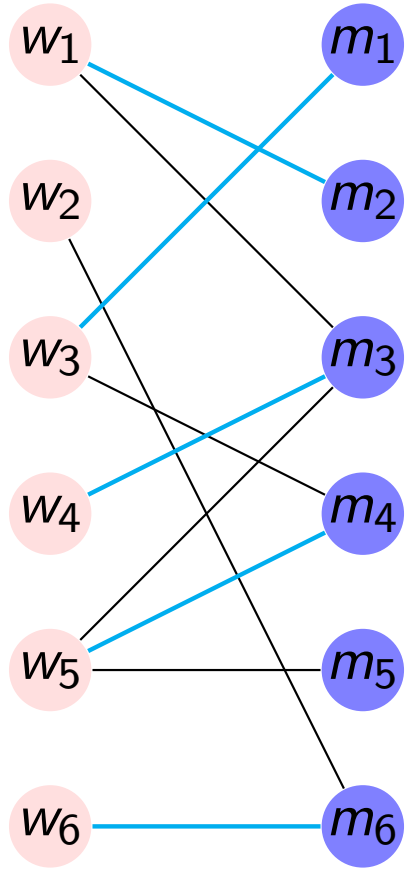

w_1
w_2
w_3
w_4
w_5
w_6
m_1
m_2
m_3
m_4
m_5
m_6


s
w1
w2
w3
w4
w5
w6
m1
m2
m3
m4
m5
m6
t


w_1
w_2
w_3
w_4
w_5
w_6
m_1
m_2
m_3
m_4
m_5
m_6
s
t
1
1
1
1
1
1
1
1
1
1
1
1

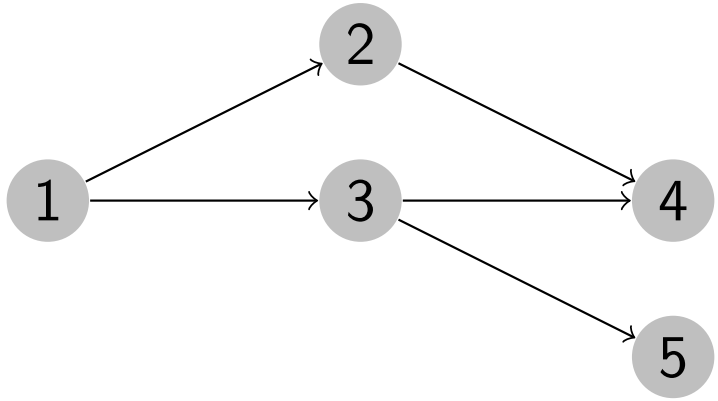

w_1
w_2
w_3
w_4
w_5
w_6
m_1
m_2
m_3
m_4
m_5
m_6
s
t
1

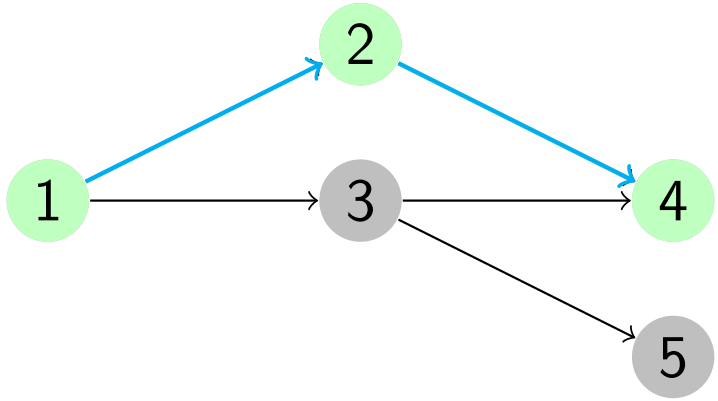

w_1
w_2
w_3
w_4
w_5
w_6
m_1
m_2
m_3
m_4
m_5
m_6
s
t
1

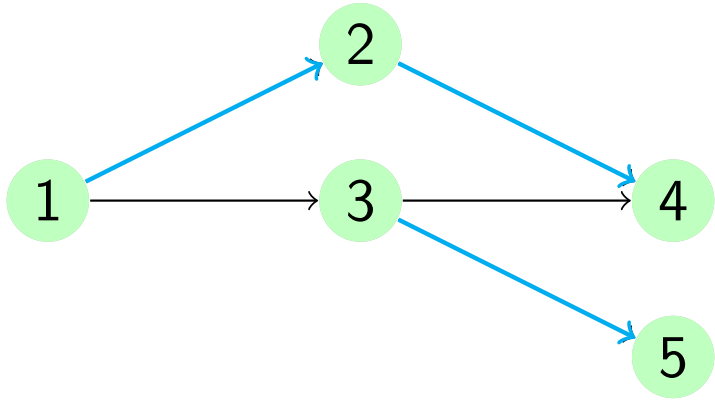

w_1
w_2
w_3
w_4
w_5
w_6
m_1
m_2
m_3
m_4
m_5
m_6

Min Path Cover on DAG:

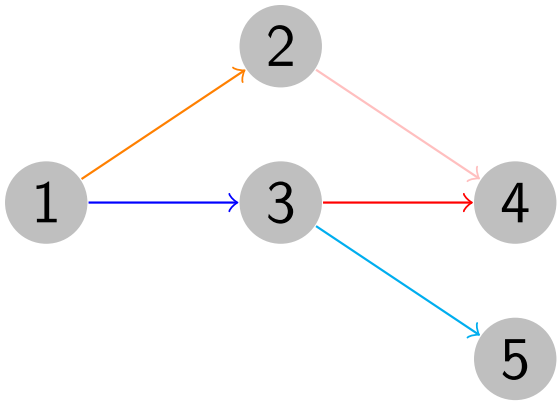
Given a DAG compute the minimum number of **node disjoint** paths required to cover all the nodes.

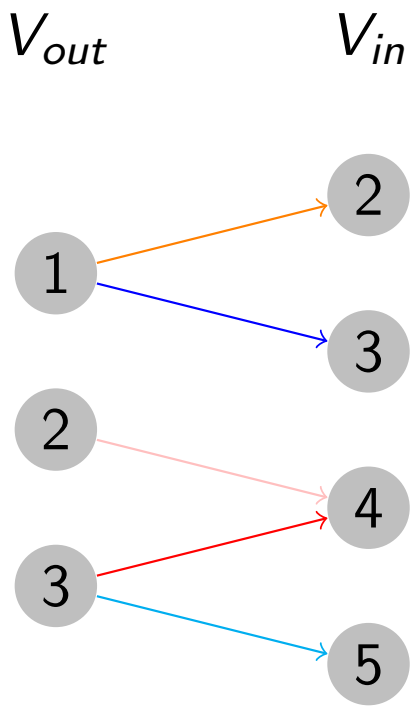

2
1
3
4
5


2
1
3
4
5

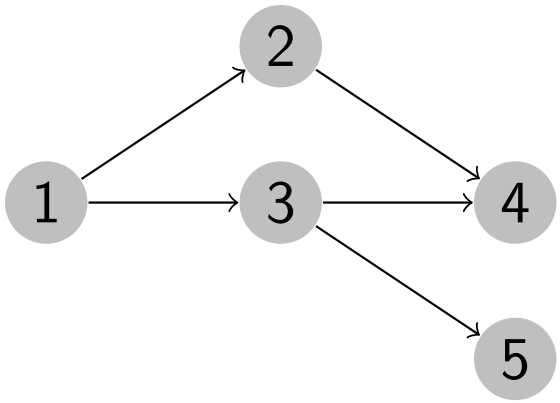

2
1
3
4
5

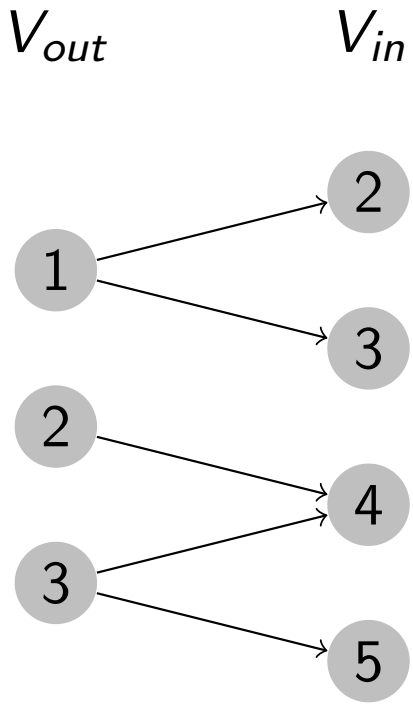
Define a bipartite graph containing the edges of input graph:



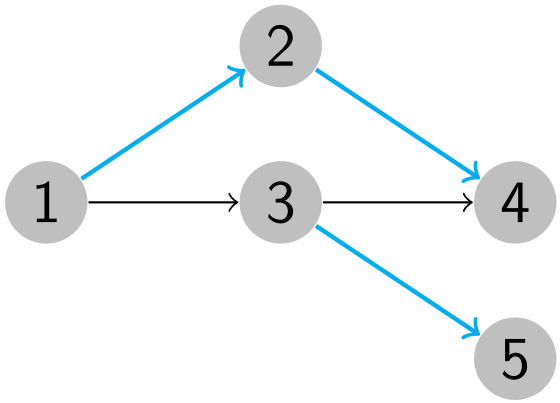


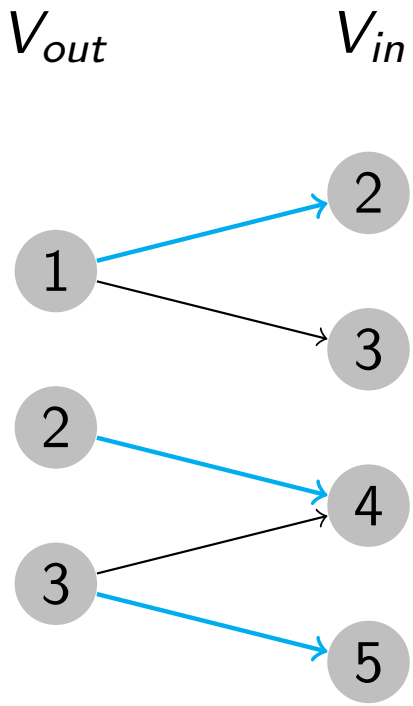
Compute maximum bipartite matching.



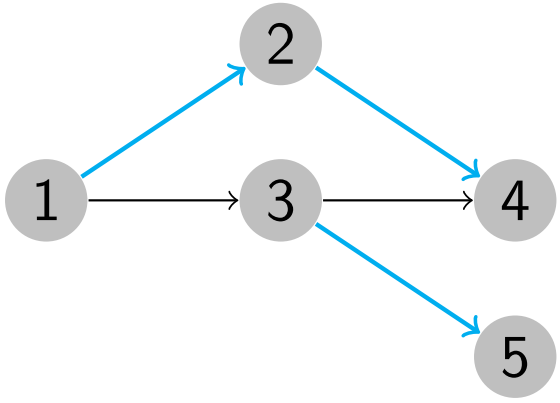


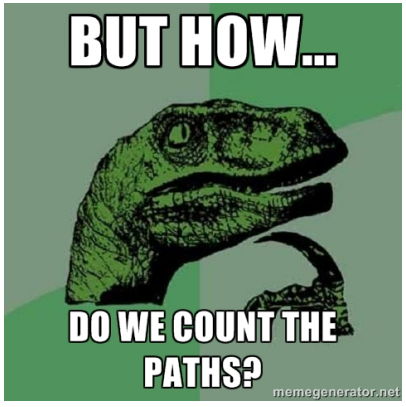
Compute maximum bipartite matching.

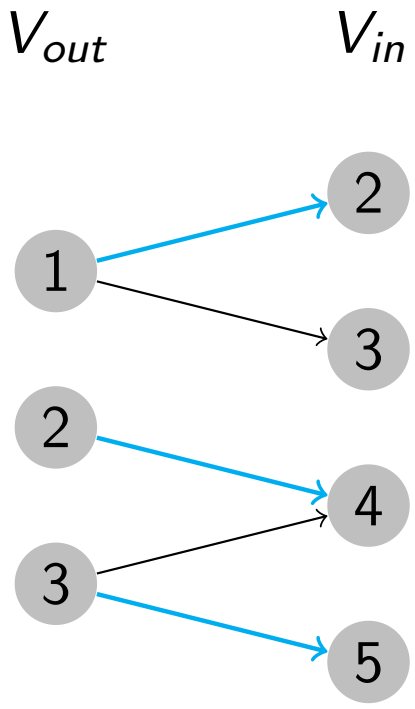




Compute maximum bipartite matching.







- The n nodes can be covered with n paths of length 0.
- Matching nodes a and b says that we can use one less path.
- If the MCBM has size m, then we just need $n - m$ paths.

- The n nodes can be covered with n paths of length 0.
- Matching nodes a and b says that we can use one less path.
- If the MCBM has size m, then we just need $n - m$ paths.

Variants:

1. What if we do not require node disjoint paths?

- The n nodes can be covered with n paths of length 0.
- Matching nodes a and b says that we can use one less path.
- If the MCBM has size m, then we just need $n - m$ paths.

Variants:

1. What if we do not require node disjoint paths?

   Same algorithm but on the **transitive closure**.

- The n nodes can be covered with n paths of length 0.
- Matching nodes a and b says that we can use one less path.
- If the MCBM has size m, then we just need $n - m$ paths.

Variants:

1. What if we do not require node disjoint paths?

   Same algorithm but on the **transitive closure**.

2. Edge disjoint paths.

- The n nodes can be covered with n paths of length 0.
- Matching nodes a and b says that we can use one less path.
- If the MCBM has size m, then we just need $n - m$ paths.

Variants:

1. What if we do not require node disjoint paths?

   Same algorithm but on the **transitive closure**.

2. Edge disjoint paths.

$$\sum_{v \in V} \max\left(\delta^+(v) - \delta^-(v), 0\right)$$

Maximum edje-disjoint paths:

Given a graph and two nodes s and t compute the maximum number edge disjoint paths between them.

Maximum edje-disjoint paths:

Given a graph and two nodes s and t compute the maximum number edge disjoint paths between them.

Solution:
Set a unit capacity on every edge and compute the maximum flow.

What about node-disjoint paths?

Maximum edje-disjoint paths:

Given a graph and two nodes s and t compute the maximum number edge disjoint paths between them.

Solution:
Set a unit capacity on every edge and compute the maximum flow.

What about node-disjoint paths?
Set also a unit capacity on every node (by slitting).

Minimum cost maximum flow

Minimum cost flow:

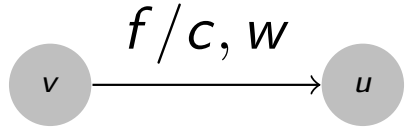
Same problem as max-flow but each edge has a weight on the edges. Find the cheapest possible way of sending some information through the network. The weight represents the cost **per unit of flow**.

Minimum cost flow:

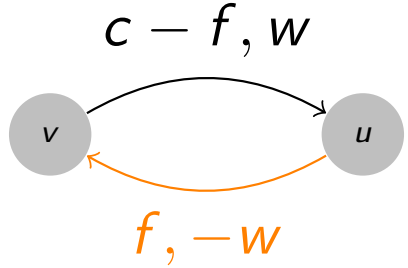
Same problem as max-flow but each edge has a weight on the edges. Find the cheapest possible way of sending some information through the network. The weight represents the cost **per unit of flow**.

Residual graph for minimum cost flow:

Original



Residual



Same intuition as before: residual edges reverse choices.

Therefore we must undo what we payed $\Rightarrow$ weight $= -w$.

How to we find paths? BFS? DFS? Dijkstra? Other?

Same intuition as before: residual edges reverse choices.

Therefore we must undo what we payed $\Rightarrow$ weight $= -w$.

How to we find paths? BFS? DFS? Dijkstra? Other?

Bellman-Ford! (negative edge costs)

When do we stop?

Same intuition as before: residual edges reverse choices.

Therefore we must undo what we payed $\Rightarrow$ weight $= -w$.

How to we find paths? BFS? DFS? Dijkstra? Other?

Bellman-Ford! (negative edge costs)

When do we stop?

No shortest path $\Leftrightarrow$ A negative cycle exists

Complexity: $O(mCU)$ with C is max cap and U is max cost.

There are many other (and faster) algorithms for the min-cost flow problem.

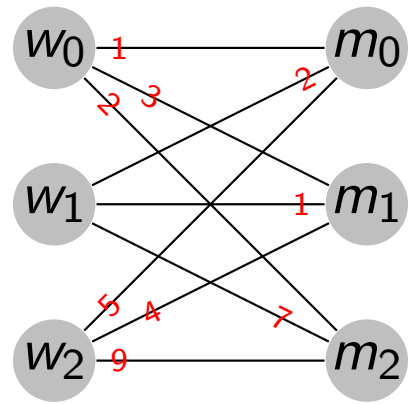
This problem is still rare in CP so we will only see this one.

It might be too slow for CP one day...

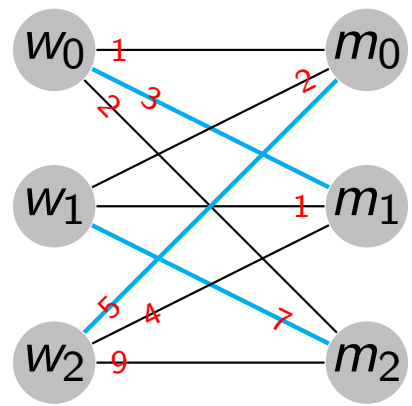
Maximum assignment problem:

There are N women and N men. We have a compatibility matrix P such that $P[w][m]$ gives a score to the couple (w, m). Make N couples while maximizing the sum of the compatibilities.

	m_0	m_1	m_2
w_0	1	3	2
w_1	2	1	7
w_2	5	4	9



	m_0	m_1	m_2
w_0	1	**3**	2
w_1	2	1	**7**
w_2	**5**	4	9



Can be solved with min-cost flow (same modelization as max-bipartie matching).

Specialized algorithm in the cheat sheets: $O(n^3)$ (much faster)

To minimize, run algorithm on $-P$.

If non-square input matrix make it square by adding $\pm\infty$.

Identifying flow problems

Look for:

- Problems with relativelly small graphs.
- Some information has to be moved on the graph.
- You are asked to make pairs of vertices.
- You are asked the robustness of a graph / network.
- Distribute resources / goods.

12125 March of the Penguins

Somewhere near the south pole, a number of penguins are standing on a number of ice floes. Being social animals, the penguins would like to get together, all on the same floe. The penguins do not want to get wet, so they have use their limited jump distance to get together by jumping from piece to piece. However, temperatures have been high lately, and the floes are showing cracks, and they get damaged further by the force needed to jump to another floe. Fortunately the penguins are real experts on cracking ice floes, and know exactly how many times a penguin can jump off each floe before it disintegrates and disappears. Landing on an ice floe does not damage it. You have to help the penguins find all floes where they can meet.

10092 The Problem with the Problem Setter

The number of students interested to participate in this year's *Intra-BUET Programming Contest* is huge. Since it is very difficult to accommodate such a large number of students in our labs, we have decided to arrange a *Screening Test*. The test will be paper-based and may include as many as 100 analytical problems from as many as 20 categories. I have been assigned the job of setting problems for this test.

At first, the job seemed to be very easy since I was told that I would be provided with a pool of about 1000 analytical problems already divided into appropriate categories. But after getting the problems I discovered that for many problems the original authors were not sure about the appropriate categories and so they wrote down multiple category-names in the category fields. Since in the *Screening Test* a problem cannot be placed under more than one category and the number of problems to be set under each category is fixed, setting problems for this test is not actually easy.

I know that a program can be written that can do the job automatically. But since I don't like writing programs, I seek your help.

563 Crimewave

Nieuw Knollendam is a very modern town. This becomes clear already when looking at the layout of its map, which is just a rectangular grid of streets and avenues. Being an important trade centre, Nieuw Knollendam also has a lot of banks. Almost on every crossing a bank is found (although there are never two banks at the same crossing). Unfortunately this has attracted a lot of criminals. Bank hold-ups are quite common, and often on one day several banks are robbed. This has grown into a problem, not only to the banks, but to the criminals as well. After robbing a bank the robber tries to leave the town as soon as possible, most of the times chased at high speed by the police. Sometimes two running criminals pass the same crossing, causing several risks: collisions, crowds of police at one place and a larger risk to be caught.

To prevent these unpleasant situations the robbers agreed to consult together. Every Saturday night they meet and make a schedule for the week to come: who is going to rob which bank on which day? For every day they try to plan the get-away routes, such that no two routes use the same crossing. Sometimes they do not succeed in planning the routes according to this condition, although they believe that such a planning should exist.

Given a grid of $(s \times a)$ and the crossings where the banks to be robbed are located, find out whether or not it is possible to plan a get-away route from every robbed bank to the city-bounds, without using a crossing more than once.