Tutorial #7

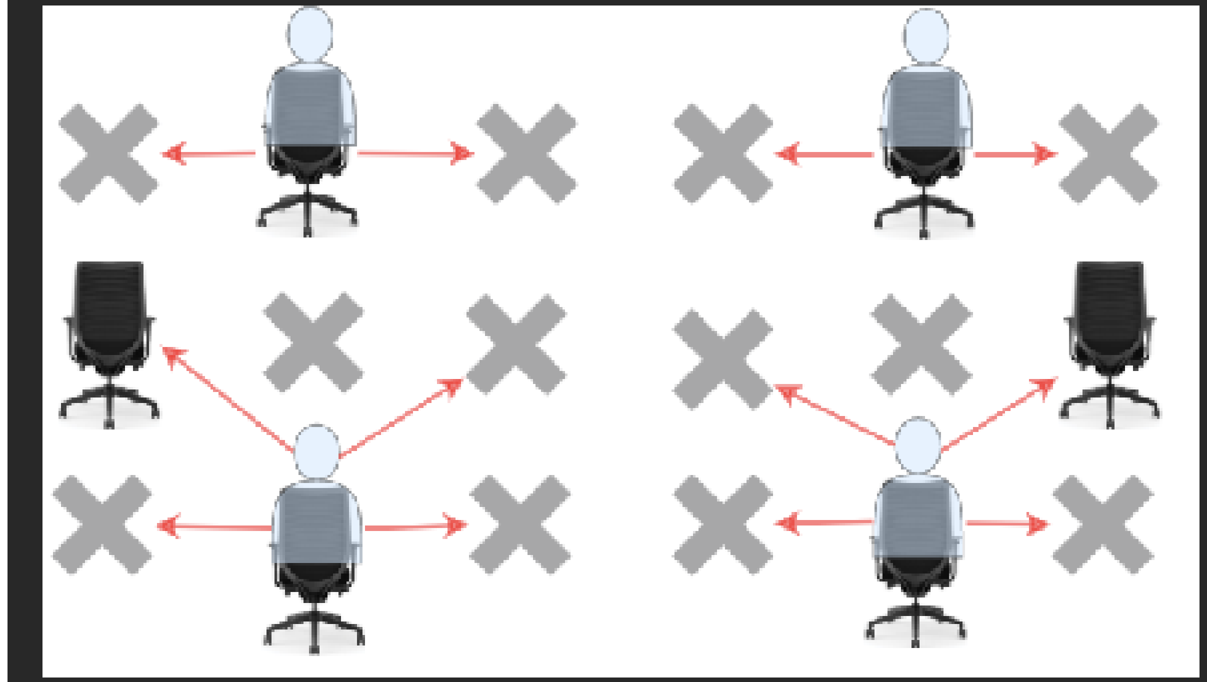
09/12/2022

Given a `m * n` matrix `seats` that represent seats distributions in a classroom. If a seat is broken, it is denoted by `'#'` character otherwise it is denoted by a `'.'` character.

Students can see the answers of those sitting next to the left, right, upper left and upper right, but he cannot see the answers of the student sitting directly in front or behind him. Return the **maximum** number of students that can take the exam together without any cheating being possible..

Students must be placed in seats in good condition.

Example 1:



```
Input: seats = [["#",".","#","#",".","#"],
                [".","#","#","#","#","."],
                ["#",".","#","#",".","#"]]
Output: 4
Explanation: Teacher can place 4 students in available seats so they don't
cheat on the exam.
```

https://leetcode.com/problems/maximum-students-taking-exam/description/

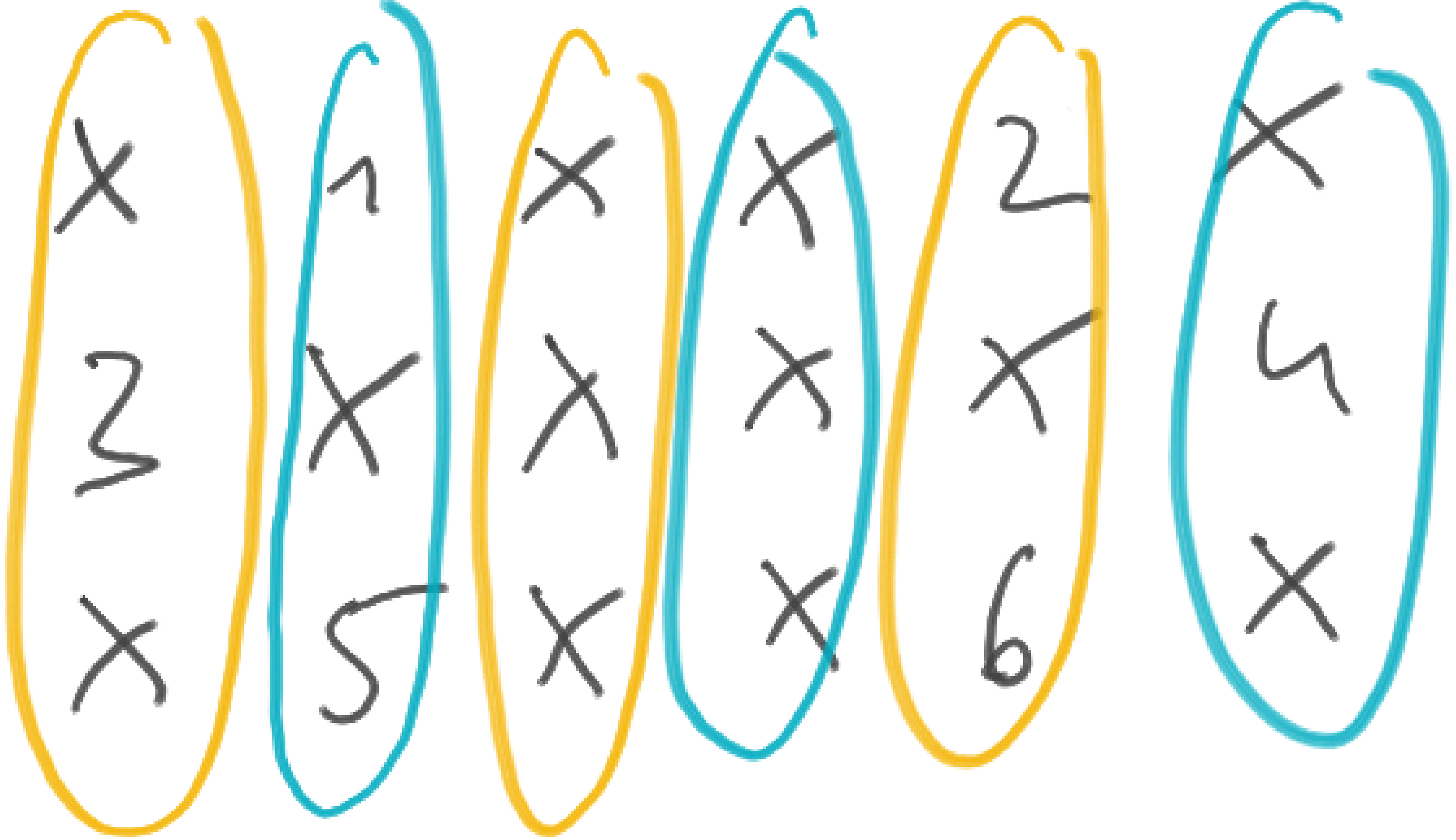
The number of students taking the exam without cheating is a maximum independet set (when modelling neighbours via edges)

Definition 6.1 (Independent Set). *Given an undirected Graph $G = (V, E)$ an* independent set *is a subset of nodes $U \subseteq V$, such that no two nodes in U are adjacent. An independent set is* maximal *if no node can be added without violating independence. An independent set of* maximum *cardinality is called maximum.*

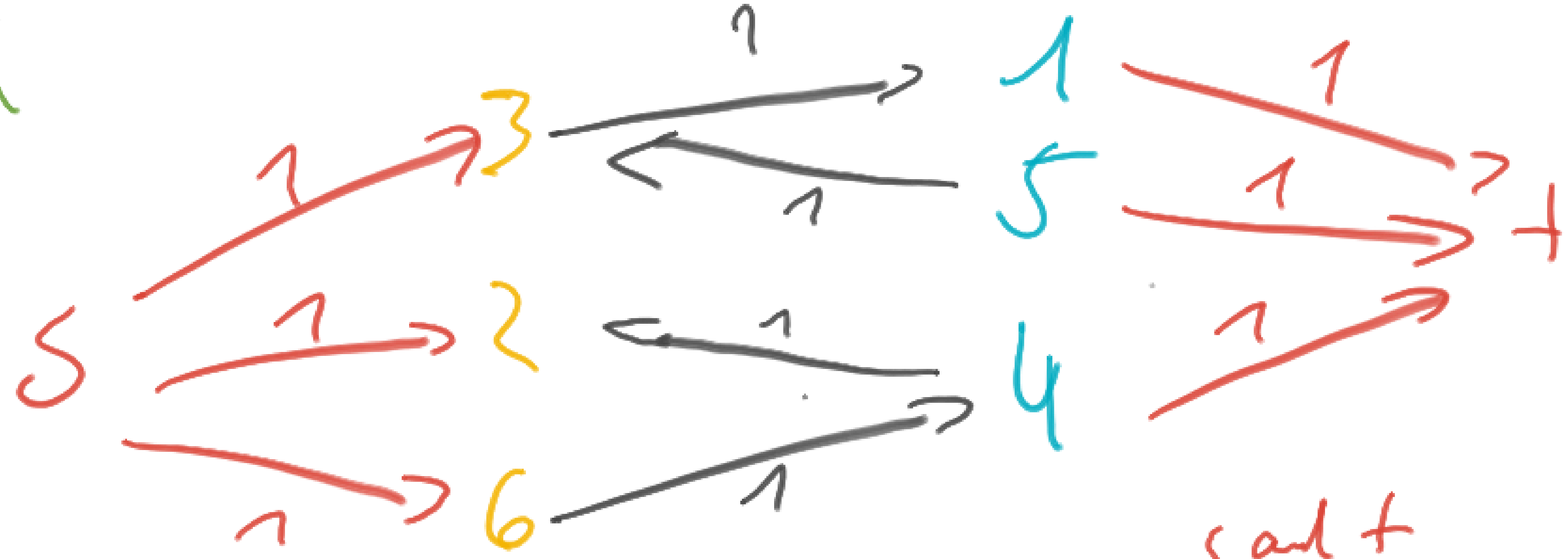
complement of independet set <=> vertex cover

In graph theory, a **vertex cover** (sometimes **node cover**) of a graph is a set of vertices that includes at least one endpoint of every edge of the graph.

Königs Theorem: min vertex cover = max matching in a bipartite graph



bip. Graph G $\Rightarrow$



s and t

max flow: 2 $\Longrightarrow$ $|G| - 2 - \text{maxflow} = 4$

Problem 1 (6 points)

1. A feasible distance function $d(\cdot)$ for *Dinitz algorithm* is given by:

   - $d(t) = 0$
   - $d(u) \leq d(v) + 1 \quad \forall\ (u, v) \in G_f$

   Show: If $d(s) \geq n$, then there is no *augmenting path.*

2. We have shown in the lecture that the running time of *Dinitz algorithmus* for networks with unit capacities (let that capacity be 1) (*unit edge weights*) is in $O((n + m)\sqrt{m})$. Compare that to the running time of the *Ford Fulkerson algorithm.* For which graphs with *unit edge weights* is which of the two algorithms faster?

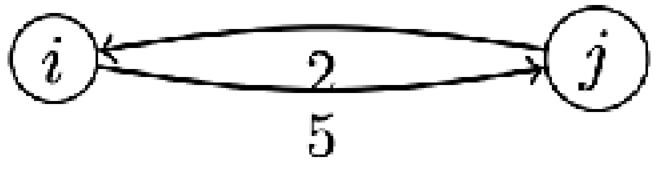
3. Let $G = (V, E)$ be a directed network, in which a maximal flow has to be computed. Let $e = (i, j) \in E$ and $e' = (j, i) \in E$, i.e. G contains a pair of edges that run in opposite directions. Moreover, let $c(e) \geq c(e')$. Disprove the following claim by giving a counter example:

   If one removes e' from E and reduces $c(e) := c(e) - c(e')$, then the maximum flow does not change.

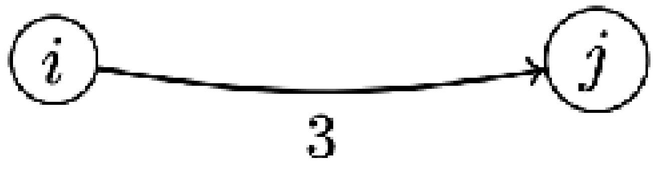
1. A path can have at most n disjoint nodes. Look at an arbitrary augmenting path in G_f. The distance to s increases by at most one for each edge on this path. Hence, in an augmenting path you can have at most $d(s) \leq n - 1$.

2. In a network with unit capacities is the running time of *Ford Fulkerson* in $O(nm)$ (since $U = 1!$). Hence, *Dinics algorithm* is faster than *Ford Fulkerson Algorithmus* iff $O((n+m)\sqrt{m}) < O(nm)$. We get $n > O(\frac{m}{\sqrt{m}-1}) = O(\sqrt{m})$.

3. If we do the described technique in the following example:



we get



For $s := j$ and $t := i$ no more flow is possible. In the initial network a flow of 2 was possible. This is a counter example for the claim.

Problem 2 (8 points)

a. Let (S, T) and (S', T') be two minimum (s, t) cuts in a flow network G. Show or disprove that $(S \cup S', T \cap T')$ and $(S \cap S', T \cup T')$ are also minimum cuts (s, t).

1. Let λ be the value of the minimum s-t-cut of G and c_{XY} be the value of the cut between vertex sets X and Y. We can divide the graph into the following parts:

    - $A = S \cap S'$
    - $B = S \cap T'$
    - $C = T \cap S'$
    - $D = T \cap T'$

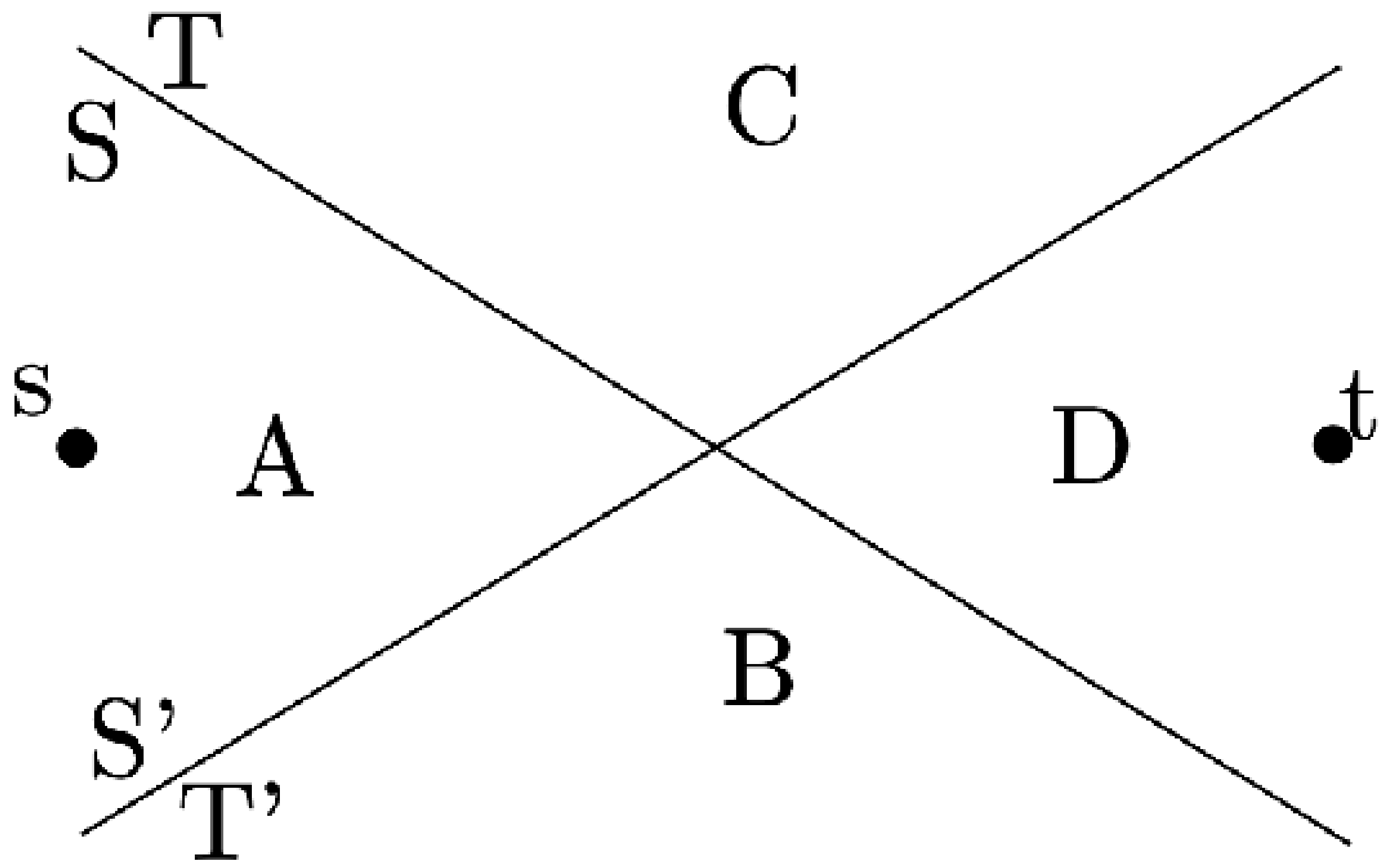


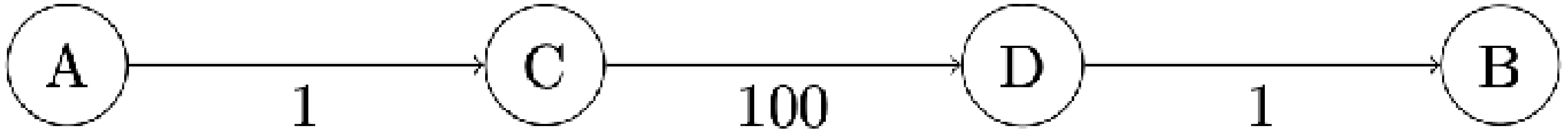
Figure 1: Graph partition according to cuts (S,T) and (S',T')

The value of the cut (S,T) is defined as $c_{AC} + c_{AD} + c_{BD} + c_{BC}$ and (S',T') is defined as $c_{AB} + c_{AD} + c_{CD} + c_{CB}$. As those are minimum cuts, their value is λ. Their sum has a value of 2λ and is $c_{AB} + c_{AC} + c_{BD} + c_{CD} + 2c_{AD}$.

The cut $(S \cup S', T \cap T')$ is defined as $c_{AD} + c_{BD} + c_{CD}$ and $(S \cap S', T \cup T')$ is defined as $c_{AB} + c_{AC} + c_{AD}$. Their sum is $c_{AB} + c_{AC} + c_{BD} + c_{CD} + 2c_{AD} = 2\lambda - c_{BC} - c_{CB}$. These 2 cuts have a combined value of less than 2λ. As the minimum cut is λ, none of those cuts can be smaller than λ. The two cuts therefore both have to be minimum cuts, as the sum of their values is 2λ and they both have a value of exactly λ.

b. Let (S, T) be a minimum (s, t) cut in a flow network G. Show or disprove, (S, T) is a minimum (x, y) cut for all $(x, y) \in S \times T$.

2. The assumption is not true. See example:



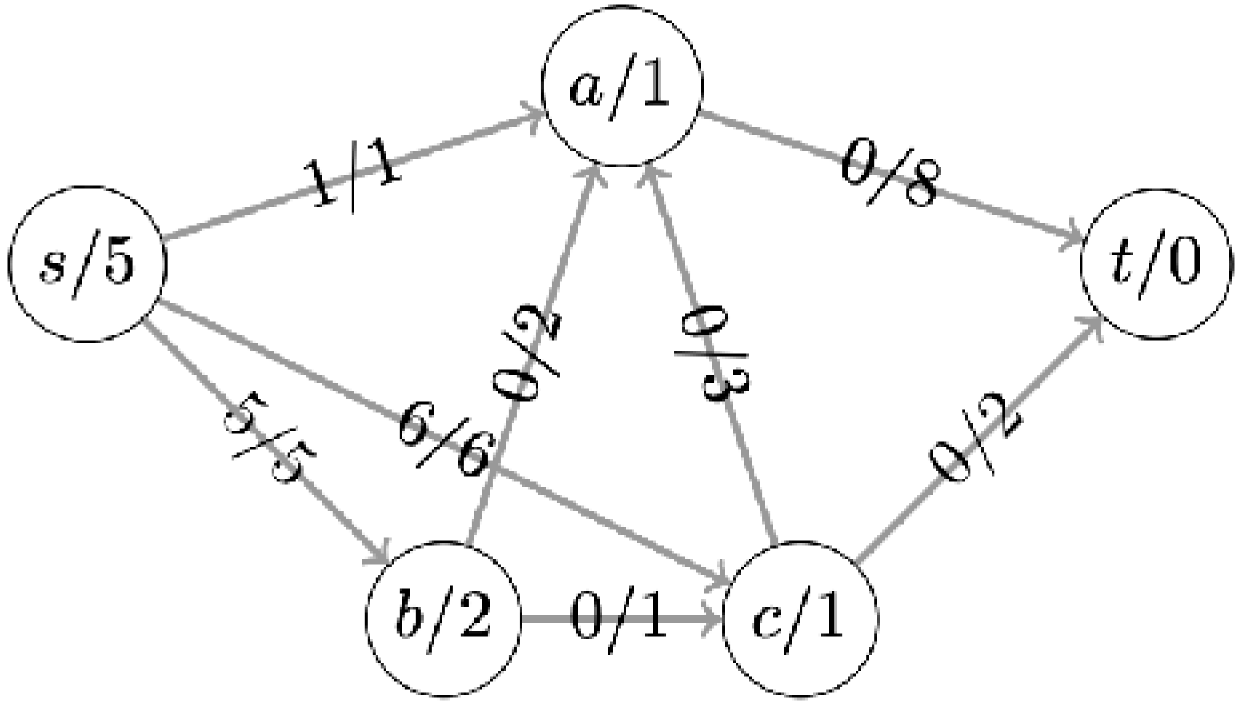
$(\{A, C\}, \{B, D\})$ is a minimum C-D-cut with value 100, but not a minimum A-B-cut, as the cut has a value of 100. $(\{A\}, \{B, C, D\})$ has a total value of 1, which is smaller than 100.

c. Given are two states of the preflow-push algorithm, a start- and end-state.

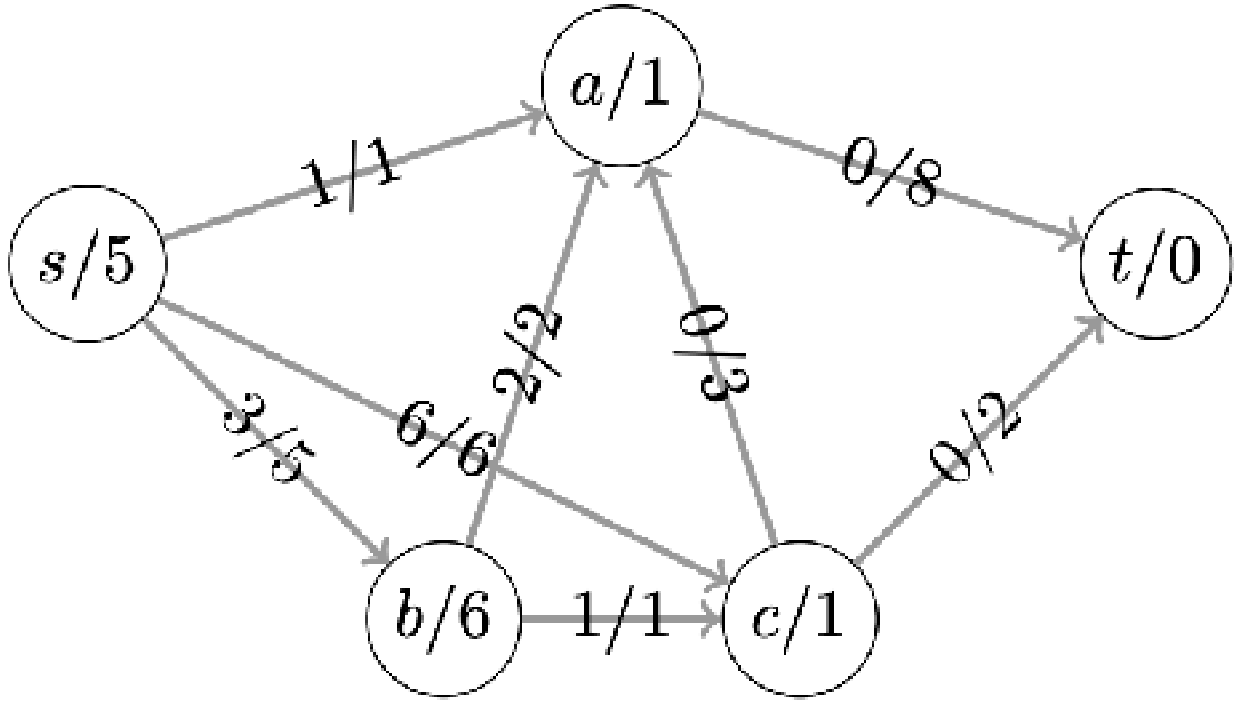
Give a potential sequence of four moves (each *push* or *relabel* with all parameters) of the algorithm, that transfers the start-state into the end-state. Multiple successive *relabel*-operations should be done in one move. Also give the intermediate flow networks.

Node description "‘node name"’/"‘level (d)"’, edge description "‘current flow"’/"‘capacity"’. The edges of the residual graph as well as the excess are omitted here.

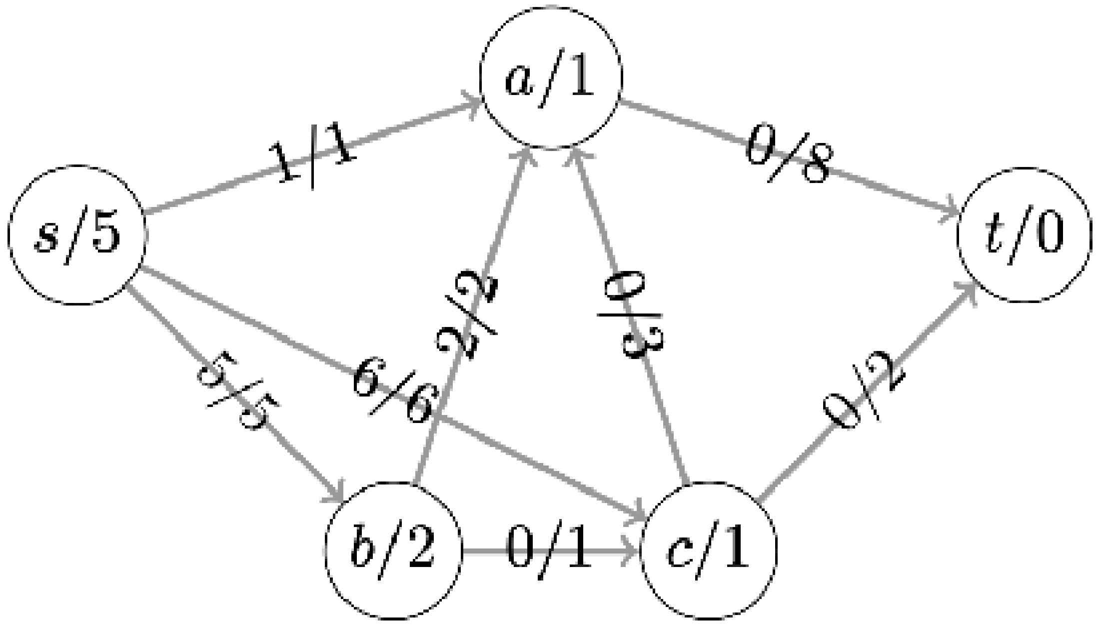
start-state



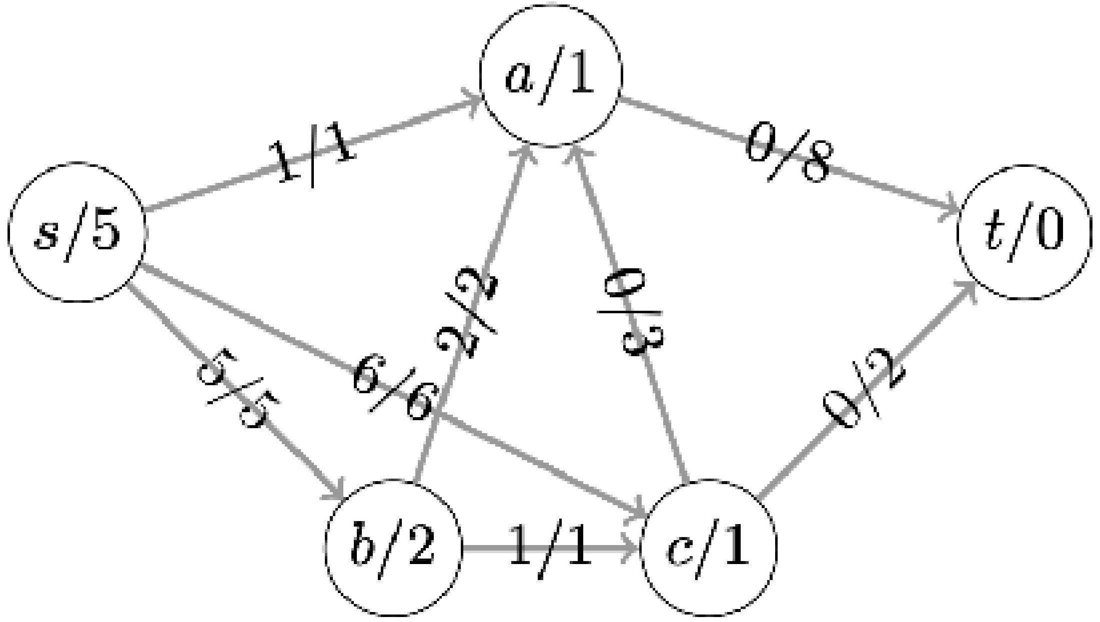
end-state



1. Push 2 flow from b to a.



2. Push 1 flow from b to c.



3. Relabel b to 6, as the lowest target in the residual graph is s with label 5.

Procedure genericPreflowPush(G=(V,E), f)

  forall $e = (s, v) \in E$ **do** push$(e, c(e))$ // saturate

  $d(s) := n$

  $d(v) := 0$ for all other nodes

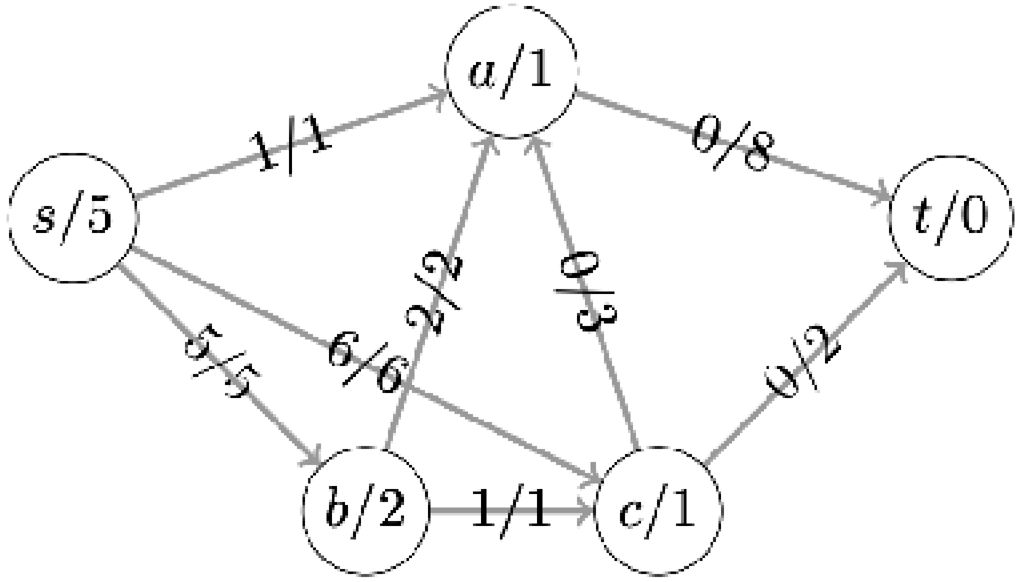
  while $\exists v \in V \setminus \{s, t\} : \mathsf{excess}(v) > 0$ **do** // active node

    if $\exists e = (v, w) \in E_f : d(w) < d(v)$ **then** // eligible edge

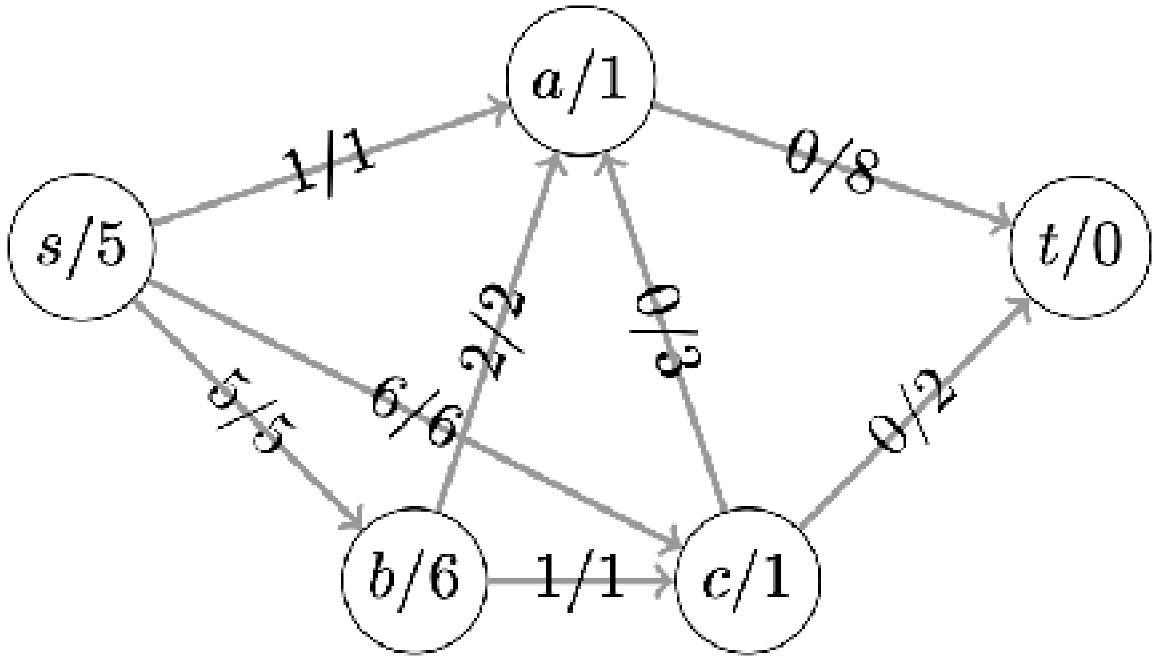
      choose some $\delta \leq \min \left\{ \mathsf{excess}(v), c_e^f \right\}$

      push(e, δ) // no new steep edges

    else $d(v)$++ // relabel. No new steep edges



3. Relabel b to 6, as the lowest target in the residual graph is s with label 5.



4. Push 2 from b to s. This results in the desired end-state.

Procedure genericPreflowPush(G=(V,E), f)
    forall $e = (s, v) \in E$ **do** push$(e, c(e))$ // saturate
    $d(s) := n$
    $d(v) := 0$ for all other nodes
    while $\exists v \in V \setminus \{s, t\} : \mathsf{excess}(v) > 0$ **do** // active node
        if $\exists e = (v, w) \in E_f : d(w) < d(v)$ **then** // eligible edge
            choose some $\delta \leq \min\left\{\mathsf{excess}(v), c_e^f\right\}$
            push(e, δ) // no new steep edges
        else $d(v)$++ // relabel. No new steep edges