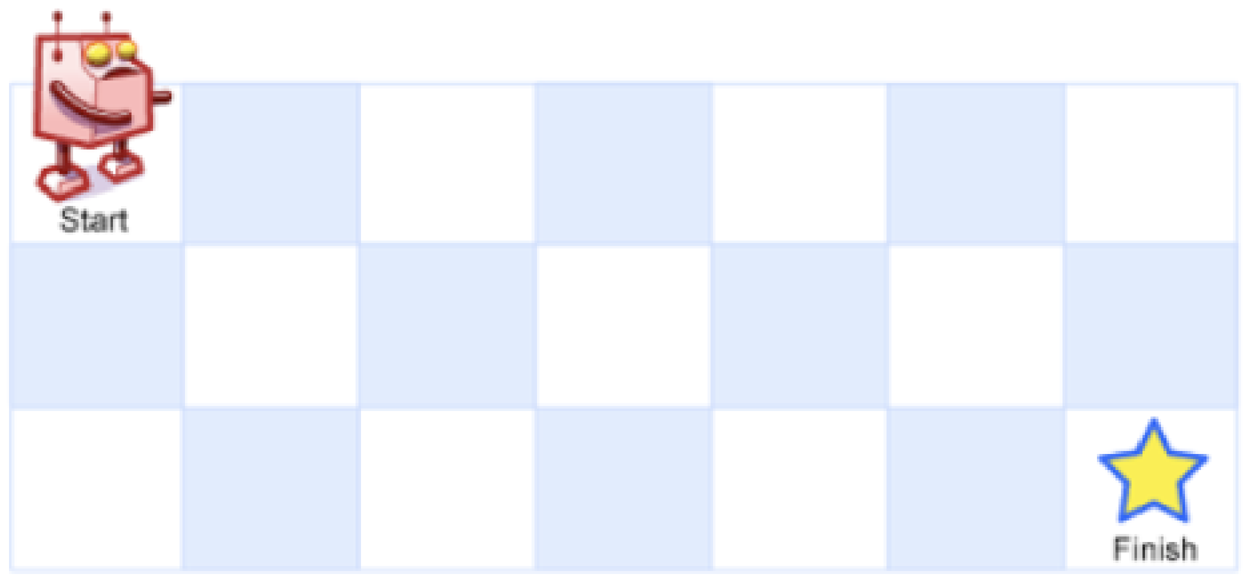
Tutorial #6

02/12/2022

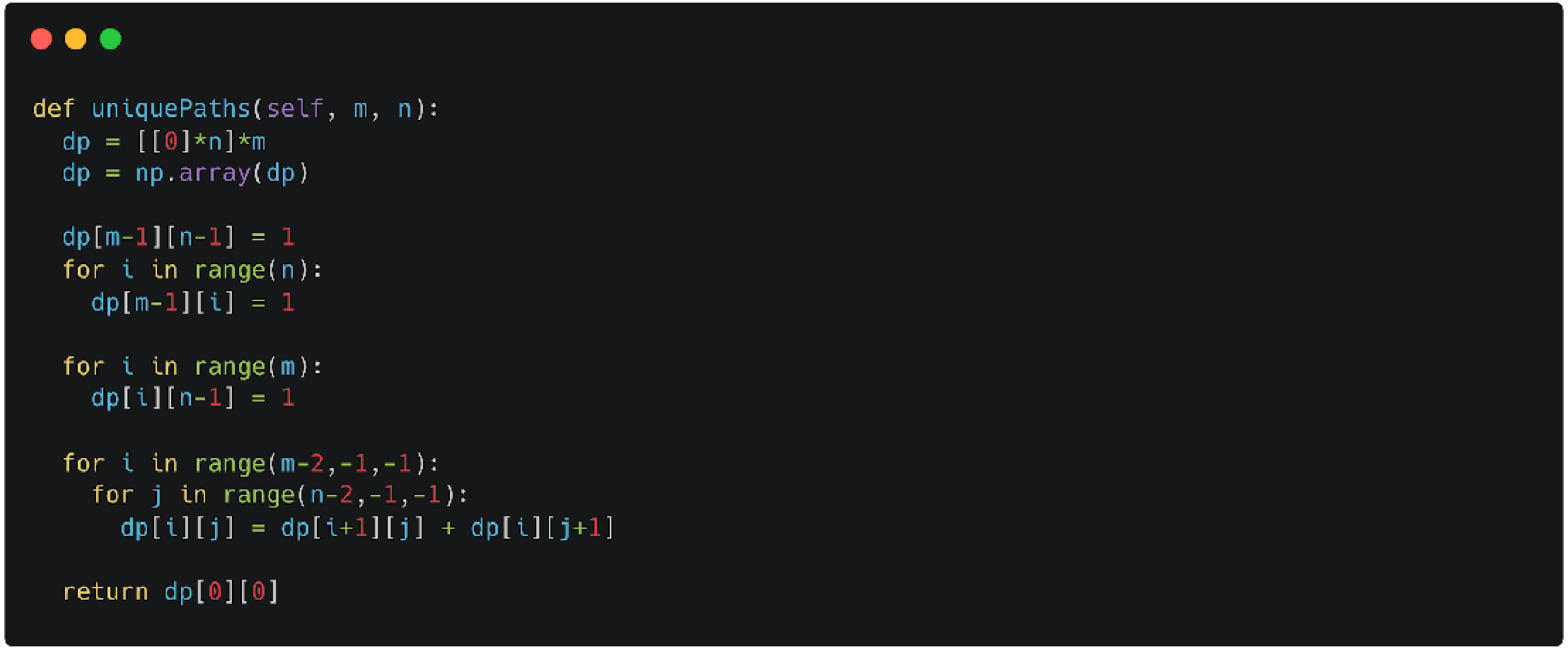
There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.



```
Input: m = 3, n = 7
Output: 28
```

https://leetcode.com/problems/unique-paths/



```python
def uniquePaths(self, m, n):
  dp = [[0]*n]*m
  dp = np.array(dp)

  dp[m-1][n-1] = 1
  for i in range(n):
    dp[m-1][i] = 1

  for i in range(m):
    dp[i][n-1] = 1

  for i in range(m-2,-1,-1):
    for j in range(n-2,-1,-1):
      dp[i][j] = dp[i+1][j] + dp[i][j+1]

  return dp[0][0]
```

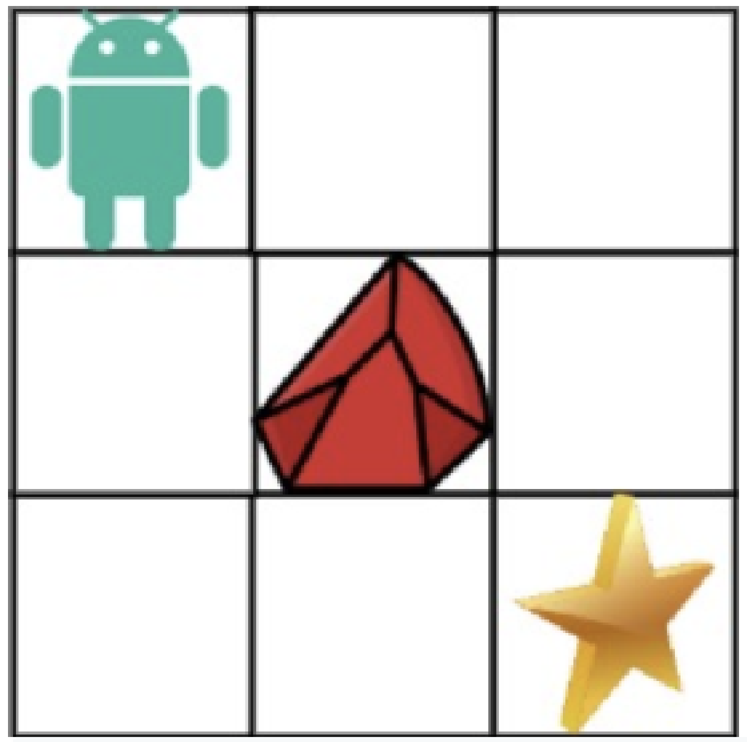
You are given an `m x n` integer array `grid`. There is a robot initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

An obstacle and space are marked as `1` or `0` respectively in `grid`. A path that the robot takes cannot include **any** square that is an obstacle.

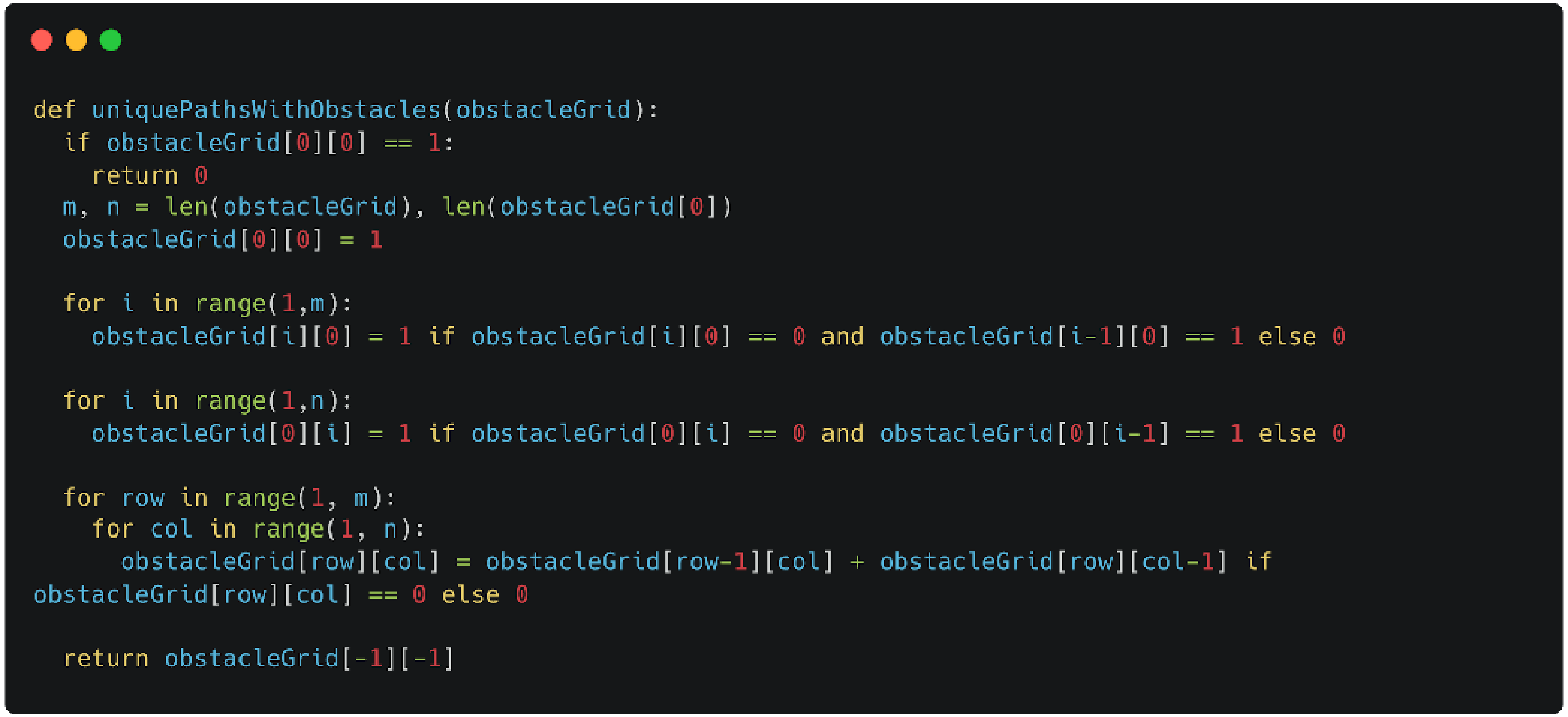
Return *the number of possible unique paths that the robot can take to reach the bottom-right corner.*

The testcases are generated so that the answer will be less than or equal to $2 * 10^9$.

Example 1:



```
Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
Output: 2
Explanation: There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right
```



```python
def uniquePathsWithObstacles(obstacleGrid):
  if obstacleGrid[0][0] == 1:
    return 0
  m, n = len(obstacleGrid), len(obstacleGrid[0])
  obstacleGrid[0][0] = 1

  for i in range(1,m):
    obstacleGrid[i][0] = 1 if obstacleGrid[i][0] == 0 and obstacleGrid[i-1][0] == 1 else 0

  for i in range(1,n):
    obstacleGrid[0][i] = 1 if obstacleGrid[0][i] == 0 and obstacleGrid[0][i-1] == 1 else 0

  for row in range(1, m):
    for col in range(1, n):
      obstacleGrid[row][col] = obstacleGrid[row-1][col] + obstacleGrid[row][col-1] if
obstacleGrid[row][col] == 0 else 0

  return obstacleGrid[-1][-1]
```

Problem 1 (2 points) Show that in a network with unit weight capacities, the value of the minimum cut is integer.

Solution:
We use the Ford-Fulkerson algorithm. The algorithm will find augmenting paths and augment along those paths. Each of the augmentation flow values by the definition of the Ford-Fulkerson algorithm. Hence, the value of the maximum flow is integer. Since this is equal to the value of the minimum cut (by the Theorem of Ford-Fulkerson), the value of the minimum cut is integer.

In mathematics, a **unimodular matrix** *M* is a square integer matrix having determinant +1 or −1. Equivalently, it is an integer matrix that is invertible over the integers: there is an integer matrix *N* that is its inverse (these are equivalent under Cramer's rule). Thus every equation *Mx* = *b*, where *M* and *b* both have integer components and *M* is unimodular, has an integer solution.

Let $A \in \mathbb{Z}^{m \times n}$ with $\operatorname{rank}(A) = m$. Then

(1) A is unimodular

$\Updownarrow$

(2) $\forall b \in \mathbb{Z}^m$, the polyeder $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ has only corners with whole numbers

$\Updownarrow$

(3) ——— " $\overset{\text{same}}{———}$ with a constraint $u \in \mathbb{Z}^n$

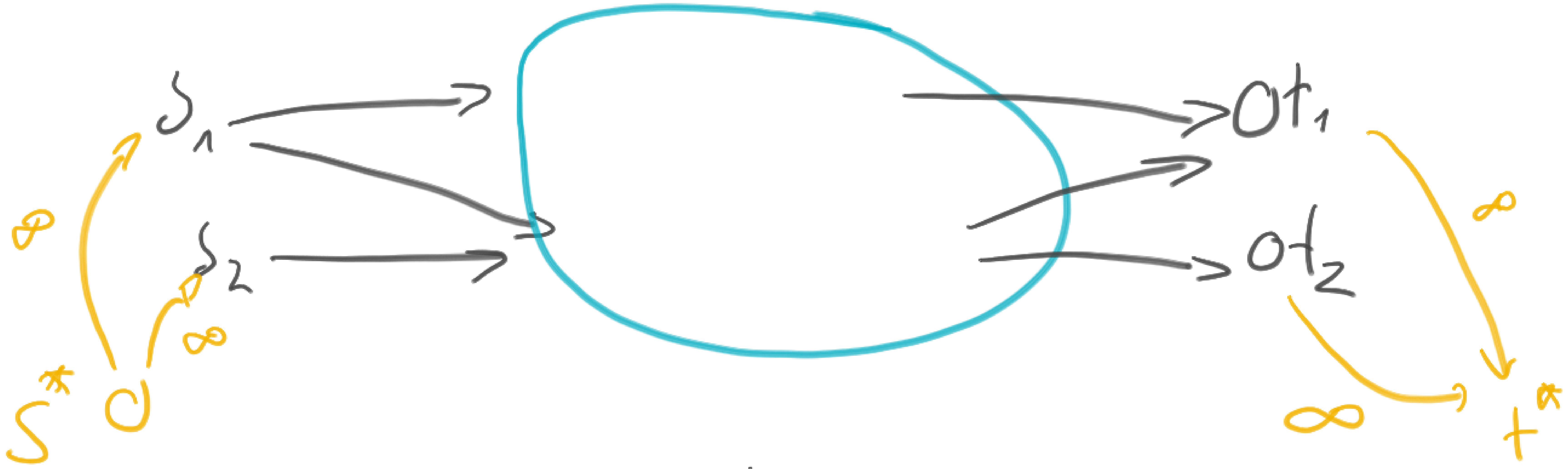
$P_{uF} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0, x \leq u\}$

Schrijver, 2003

Adjacency matrices of simple Digraphs are unimodular.

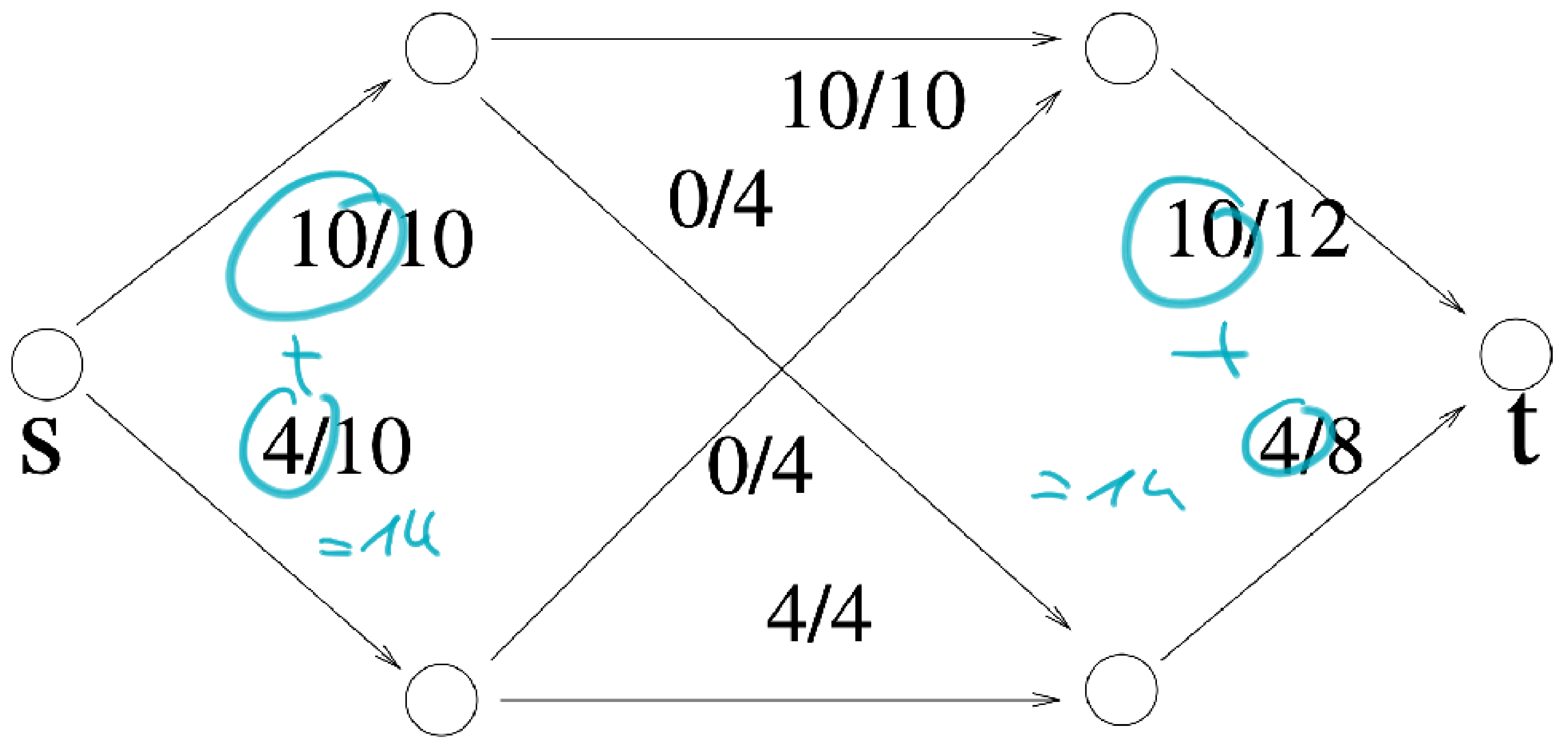
$\Rightarrow$ Hence here the flow is $\in \mathbb{Z}$

Problem 2 (2 points) In the maximum flow problem, a source node is a node that is allowed to have negative excess and a sink node is a node that is allowed to have positive excess. In the standard form of the maximum flow problem, there is only one source and one sink. In a more general form, there are multiple sources and multiple sinks. The value of a flow is the sum of the excesses of the sinks. Show how the multiple-source multiple-sink maximum flow problem can be solved by reducing it to the standard maximum flow problem.



Problem 3 (4 points) Given is the following network with capacities and a st-flow. Each edge is labeled with f/c, where f is the value of the flow on that edge and c is the initial capacity of the edge. If you don't have a printer, transfer the graph as is to your paper first.

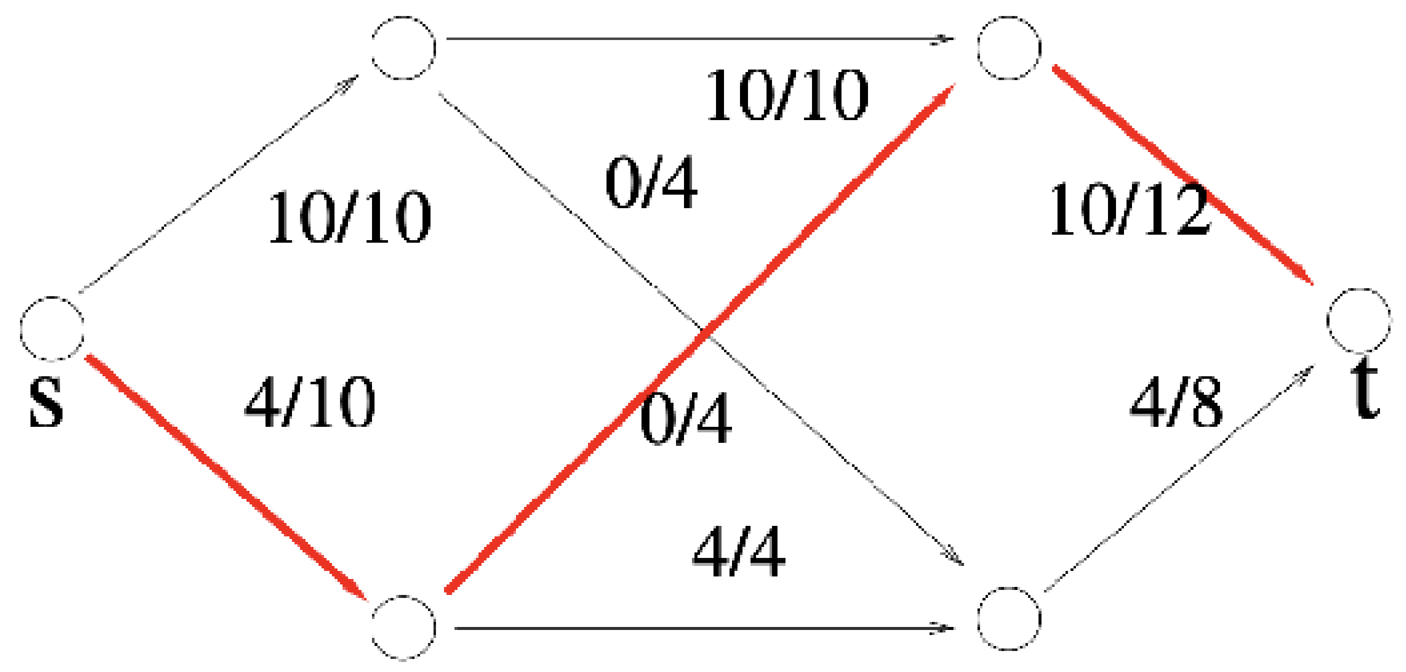
1. What is the value of the given flow?
2. Draw an augmenting path.
3. Augment along this path as in Ford-Fulkerson's algorithm. Draw the resulting graph.
4. What is the value of the flow now? Is the flow maximum? If so why? If not why?
5. What is the value of the minimum cut in the network?
6. Draw a minimum cut in the network.



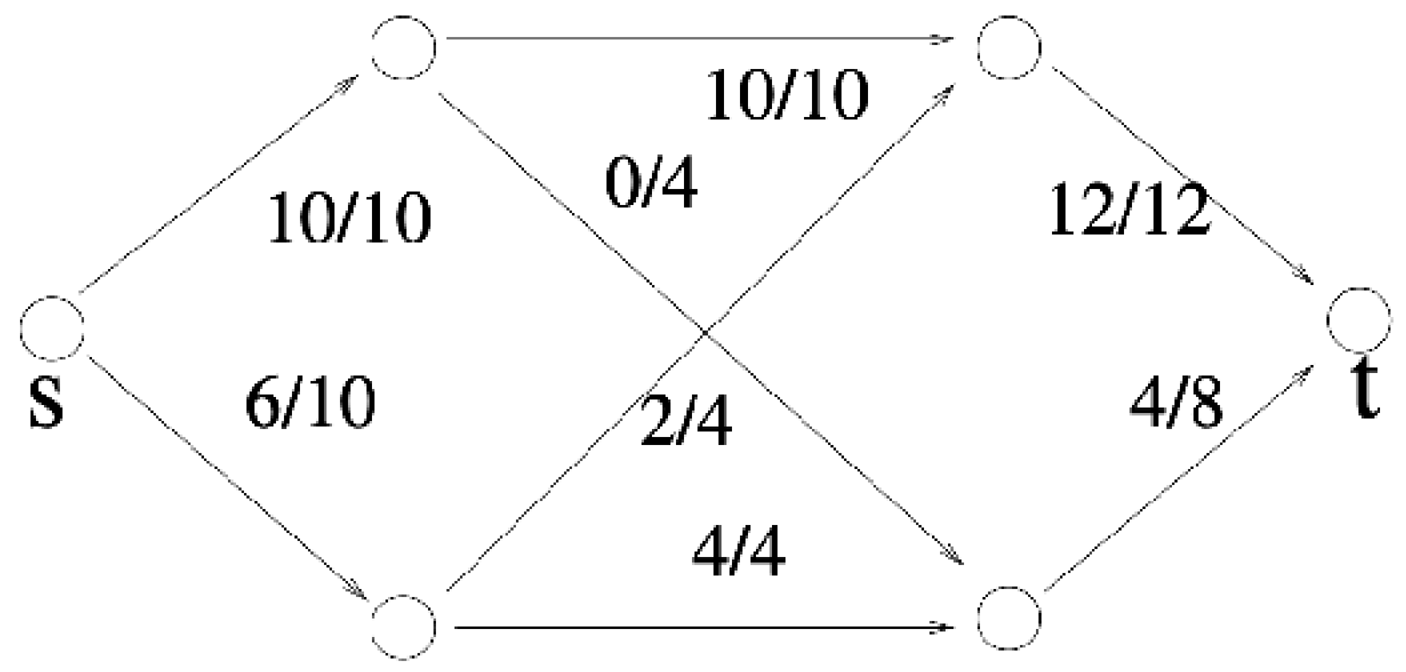
14 ✓

1. The value of the flow is 14.

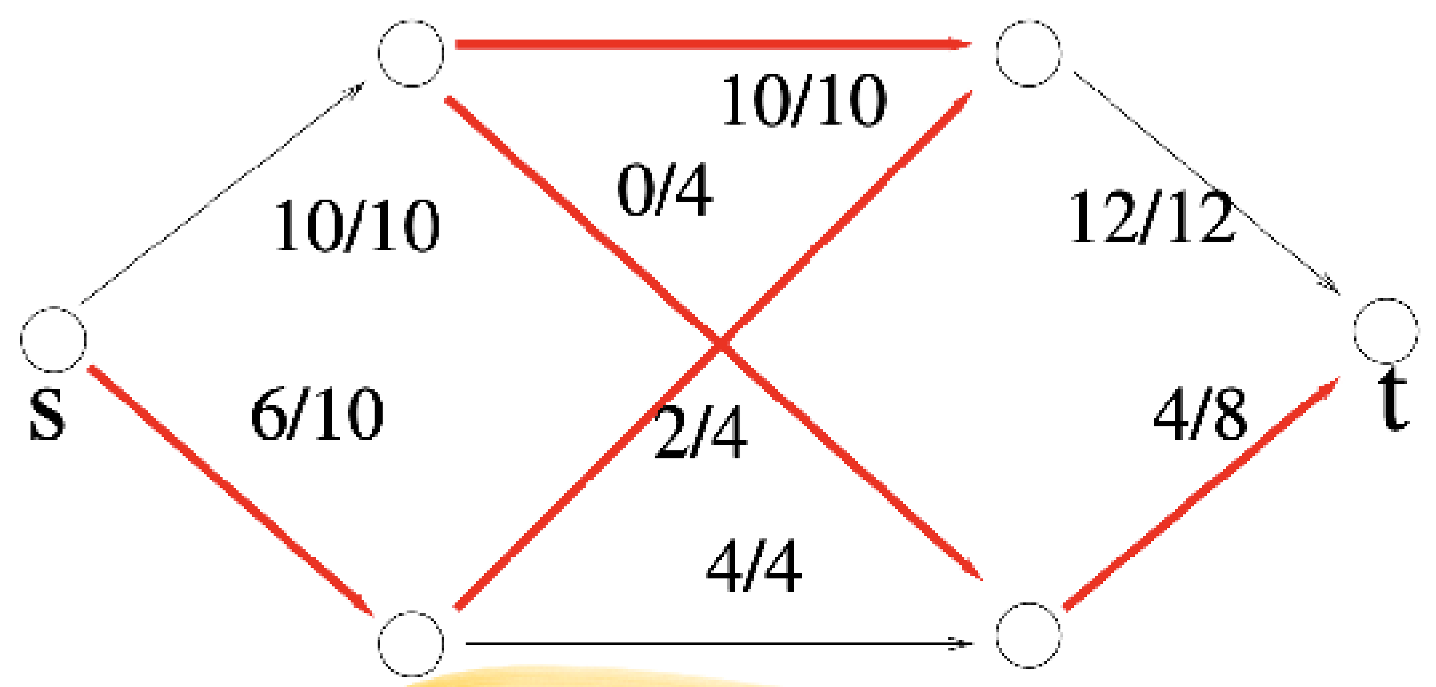
2. One augmenting path is shown in red (there are other solutions as well):



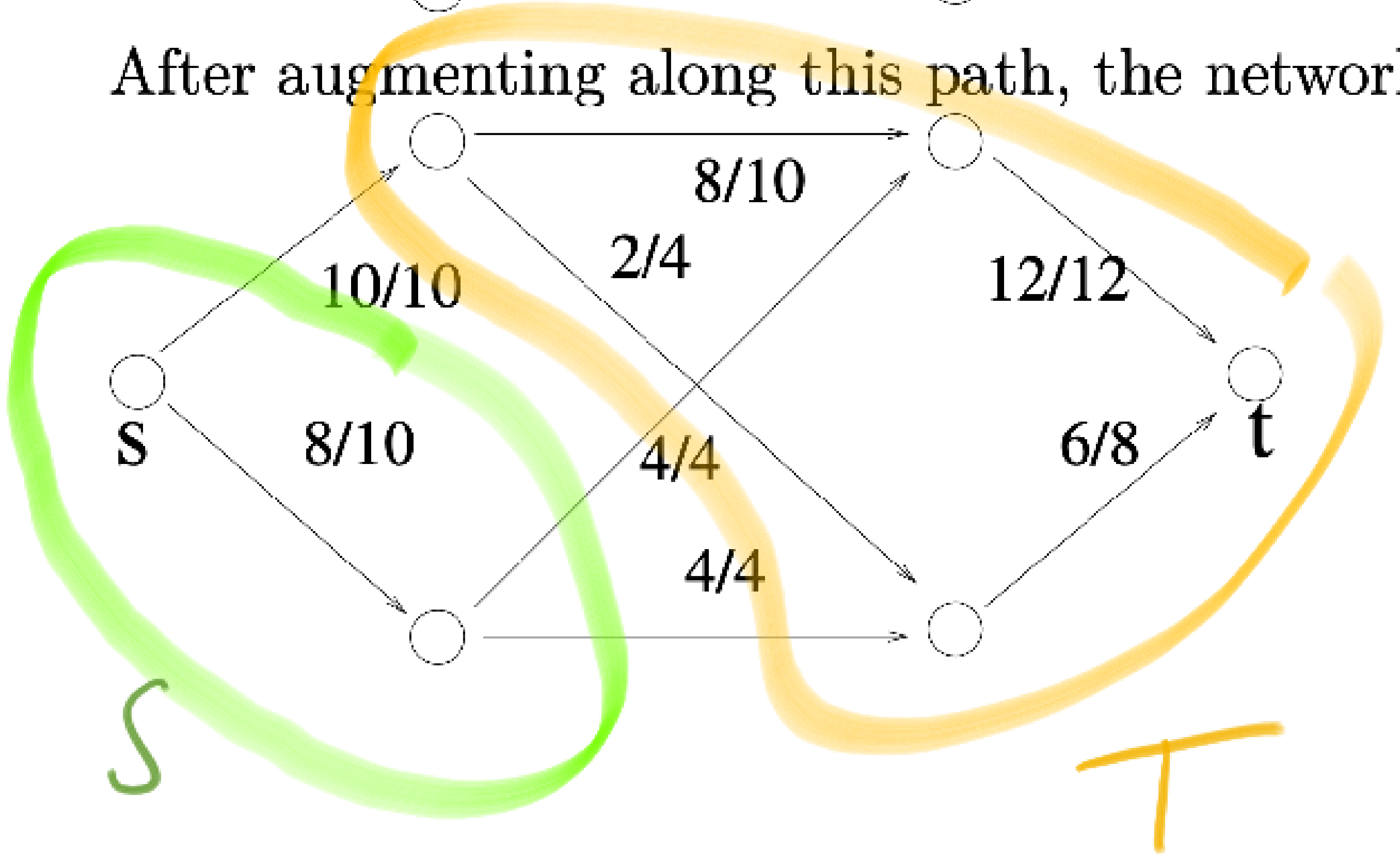
3. The resulting graph is as follows:



4. The resulting flow is not maximum as there is another augmenting path left, e.g.

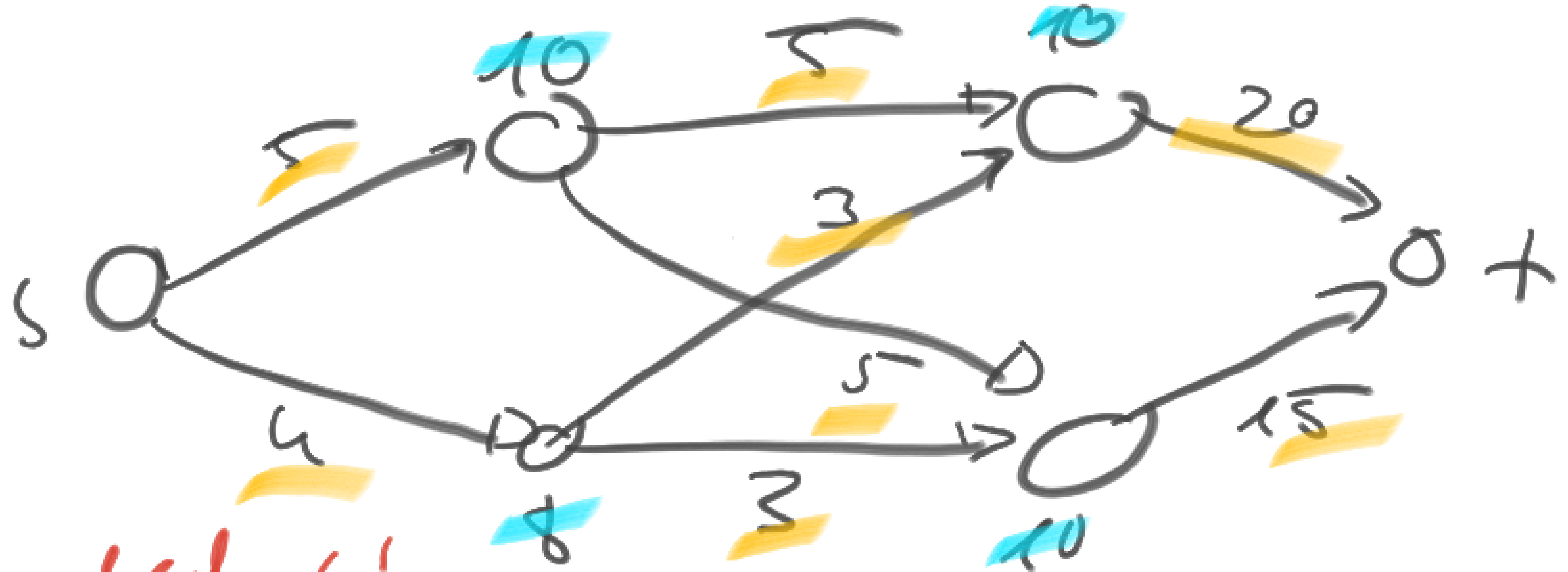


After augmenting along this path, the network looks like this



flow is 18

Problem 4 (4 points) In the following, we are interested in flows in which we have edge capacities **and** we have node capacities. That means, each node v gets a *node capacity* cap(v) and edge capacities are defined as usual. Moreover, the sum of the incomming or outgoing flows can not exceed cap(v). Given an algorithm that computes a maximum flow in a network with node and edge capacities. Argue why your algorithm computes a feasible and optimial flow.



draw expanded G' on blackboard

Let $G' = (V', E')$ and it is constructed as follows: $\forall v \in V(G) \setminus \{s, t\}$

$$V' \cup \{v, v'\} \text{ and}$$

$$\forall e = (u, v, c) \in E(G) : E' \cup \{(u, v, c)\}$$

$$\forall e = (v, w, c) \in E(G) : E' \cup \{(v', w, c)\}$$

$$E' \cup \{(v, v', \text{nodecap}(v))\}$$

Solution:
Let $G = (V, E)$ be the original graph. We define a new graph $G' = (V', E')$. For each node v in V there are two nodes v_{in} and v_{out} in V'. From v_{in} to v_{out} there is an edge with capacity $\text{cap}(v)$. Moreover, each edge $(u, v) \in E$ yields an edge $(u_{\text{aus}}, v_{\text{ein}})$ in E'. (the capacity of this edge is set to the capacity of the original edge)

Now one computes a maximum flow from s_{in} to t_{out} in G' and transforms the flow values for the edges back to G. The flow is feasible in G w.r.t. to the node and edge capacities. since the node flow was restricted through the edge capacity of the corresponding edge. More over it is optimal, since if there would be an augmenting path that respects node capacities, then a slightly modified path would also exist in G'.

Problem 5 (4 points) Suppose you have already computed a maximum (s, t)-flow f in a flow network $G = (V, E)$ with integer capacities (so f is given to you). Let k be an arbitrary positive integer, and let e be an arbitrary edge in G. Now suppose we increase the capacity of e by k units. Show that the maximum flow in the updated graph can be computed in $O((|V| + |E|)k)$ time.

Solution:
If the capacity of e is increased by k, the by the max-flow min-cut theorem the mincut can only have increased by k. This is the case for example, if e was in a minimum cut before. Since we increased the capacity of e, f is still a feasible flow. Hence, we compute the residual network and use augmenting paths to find larger flows. Computing an augmenting path can be done in time $O(|V| + |E|)$. Each augmenting path will increase the flow by one (since we have integer capacities). Hence, the algorithm will stop after at most k iterations.