CSE 421 Section 8

Max Flow / Min Cut

Administrivia

Announcements & Reminders

- Midterm Exam
  - If you think something was graded incorrectly, submit a regrade request!
  - If you have concerns about your overall grade in the course, send an email course staff to discuss privately

- HW7
  - Due tomorrow, Friday 2/23

- HW8
  - Due Friday 3/1

Ford-Fulkerson Algorithm

Finding the Max-Flow / Min-Cut

We use the Ford-Fulkerson algorithm to find Max-Flow / Min-Cut.

Key Ideas:

- Keep searching through the residual graph to find a path from s to t that we can send more flow down.
- Keep updating the residual graph to track how much flow we can still push through and how much flow we can potentially reroute.
- When we can no longer reach t in the residual graph, we can't send any more flow, so the algorithm terminates!

Residual Graph

The residual graph indicates how much flow can still go along an edge, and how much flow we could potentially reroute back from an edge.

Key ideas:

- the sum of the residual edges between any two nodes should be equal to the value of the edge between them in the original graph
- The residual edge pointing in the direction of the original edge should have a value equal to the amount of flow that could still pass through that edge
- The residual edge pointing in the opposite direction of the original edge should have a value equal to the amount of flow you have currently sent down that edge

Ford-Fulkerson (formally)

```
**While** (flow is not maximum)
        Run BFS in residual graph starting from $s$
        Record predecessors to find an $s,t$-path
        Iterate through path, finding $c$ minimum residual capacity on path
        Add $c$ to every edge on path in flow
        Update residual graph
```

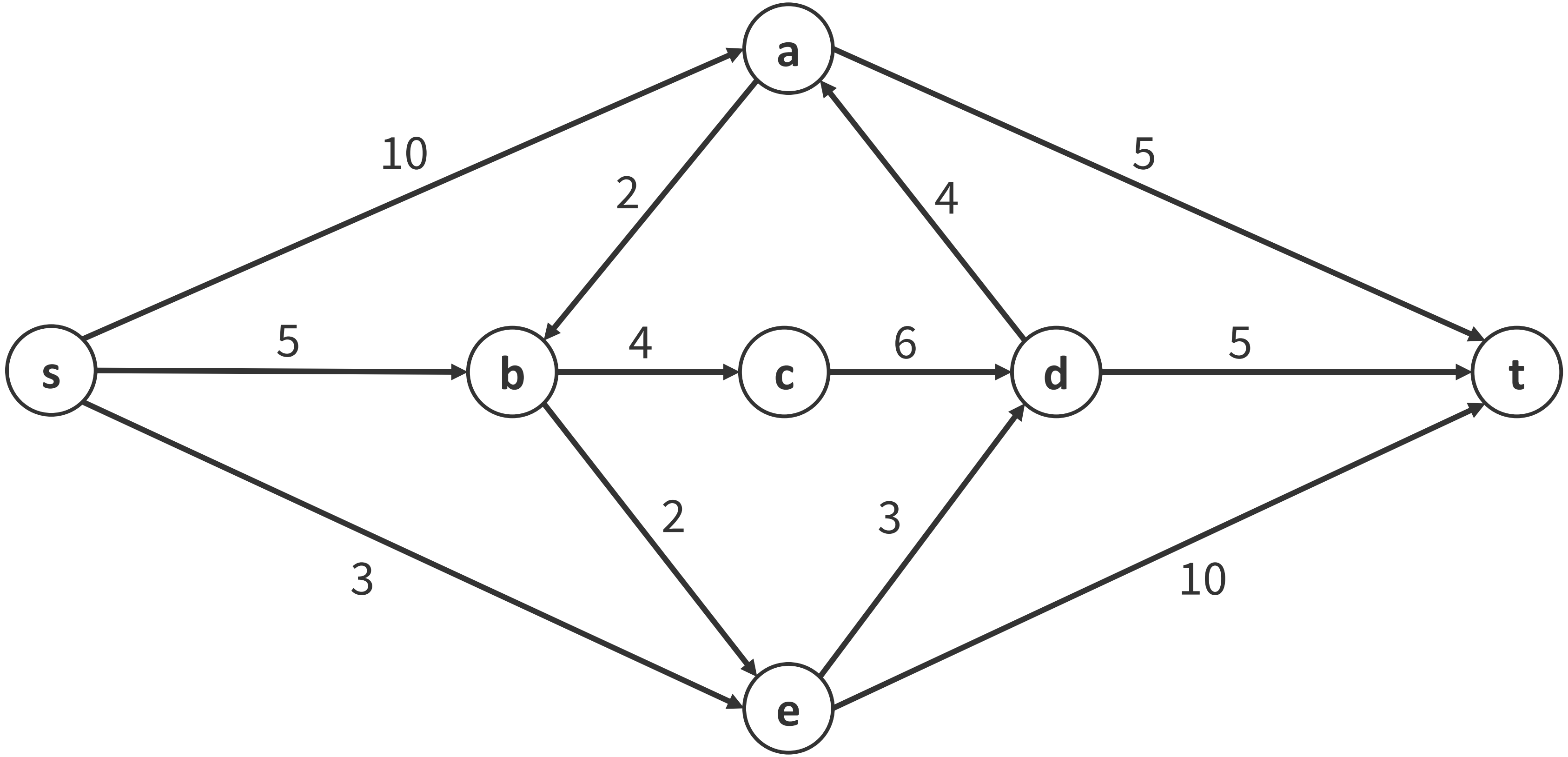
1. Go With the Flow

Problem 1 – Go With the Flow

Using Ford-Fulkerson, find the maximum $s - t$ flow in the graph G below, the corresponding residual graph, and list out the corresponding minimum cut.

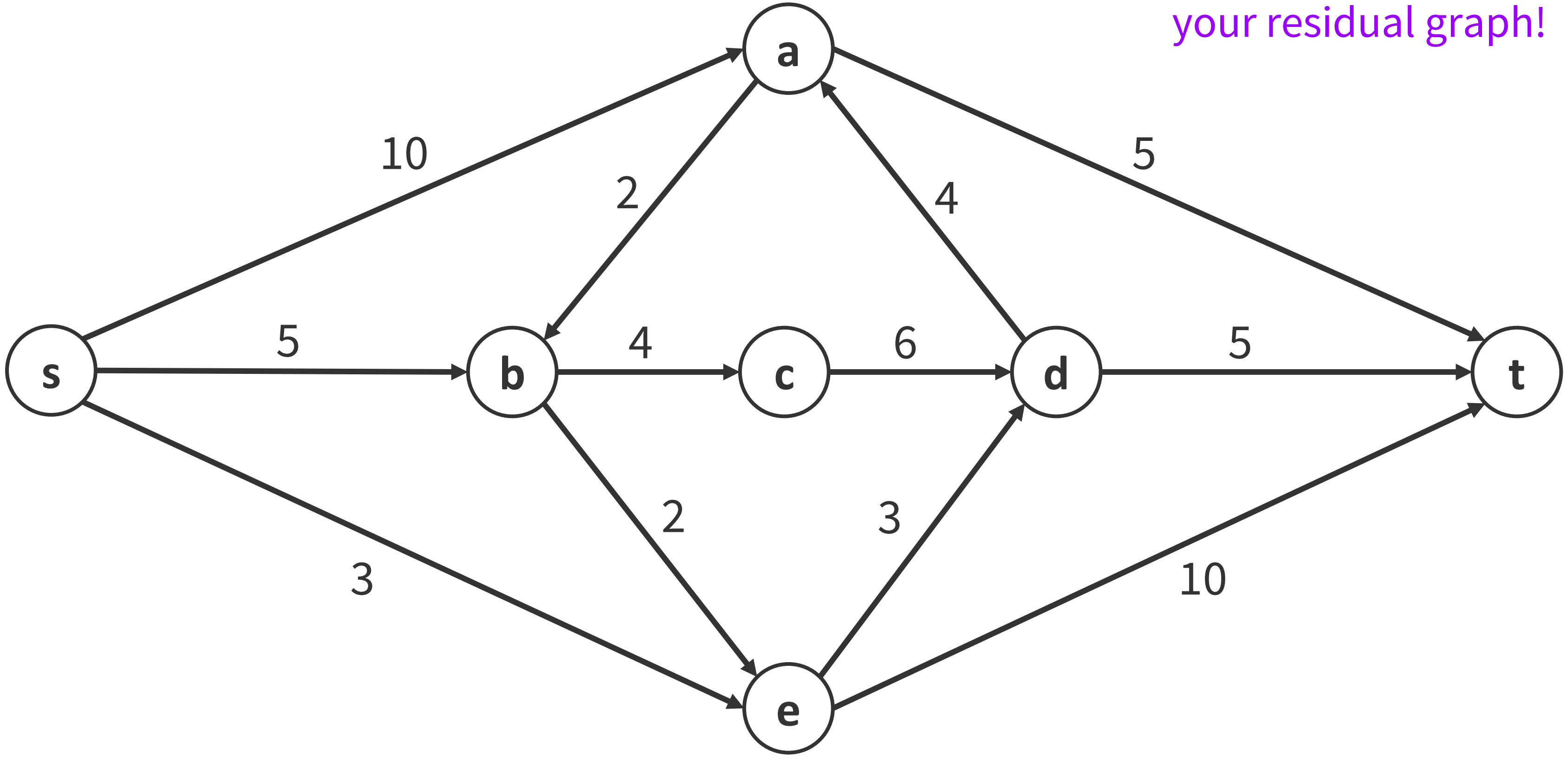
Work through this problem with the people around you, and then we’ll go over it together!

Problem 1 – Go With the Flow



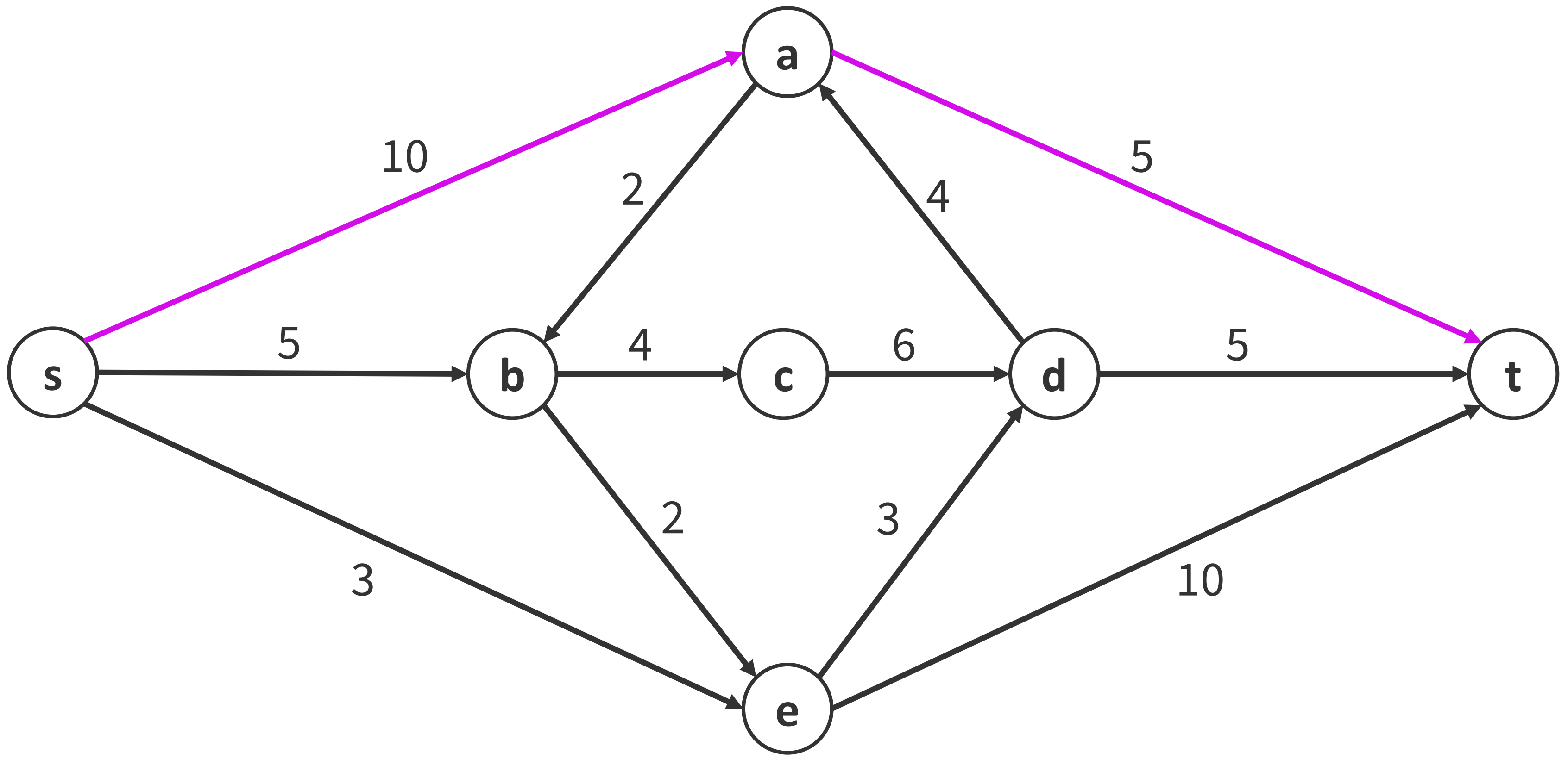
Problem 1 – Go With the Flow

Start by making a copy of the original graph to be your residual graph!



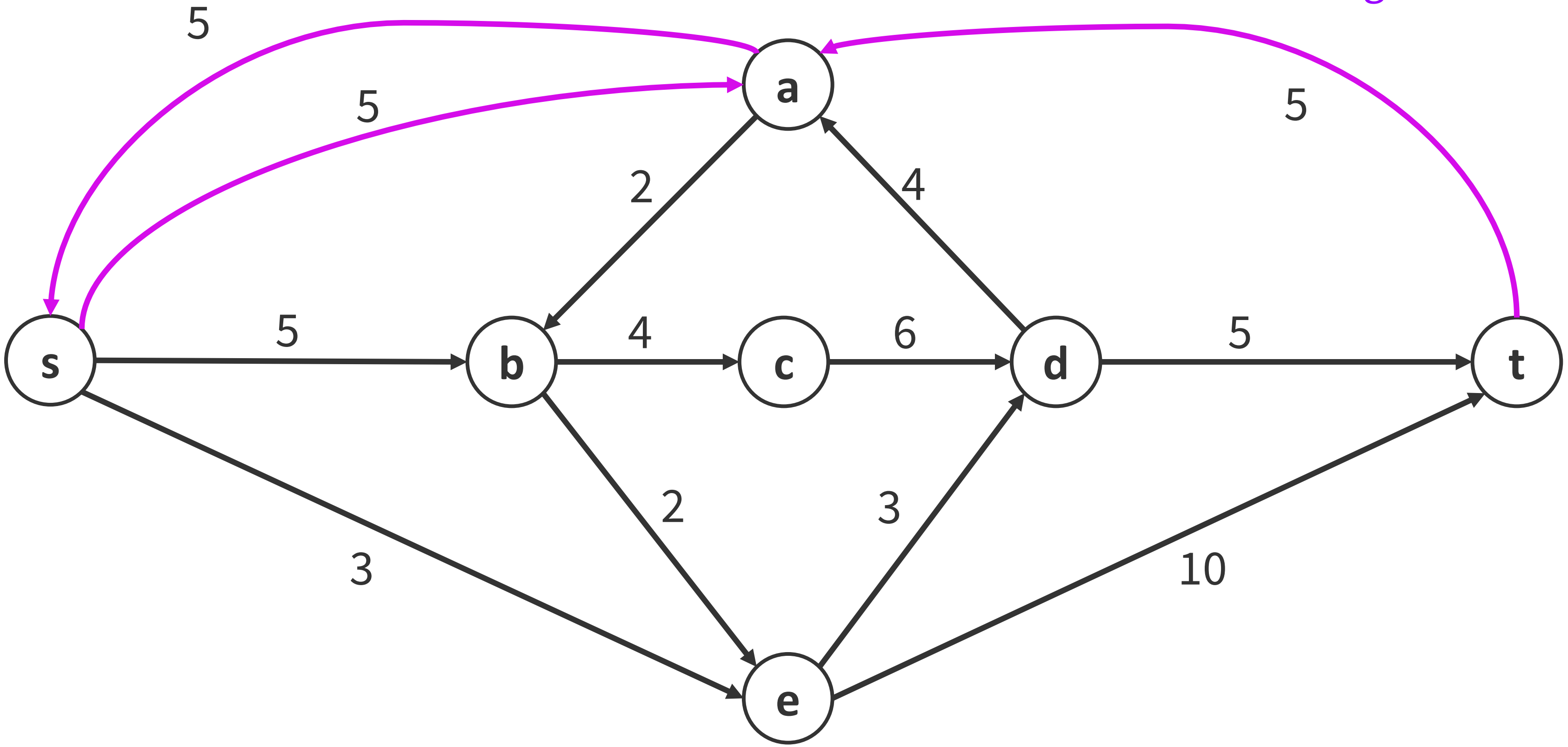
Problem 1 – Go With the Flow

Find a path from s to t

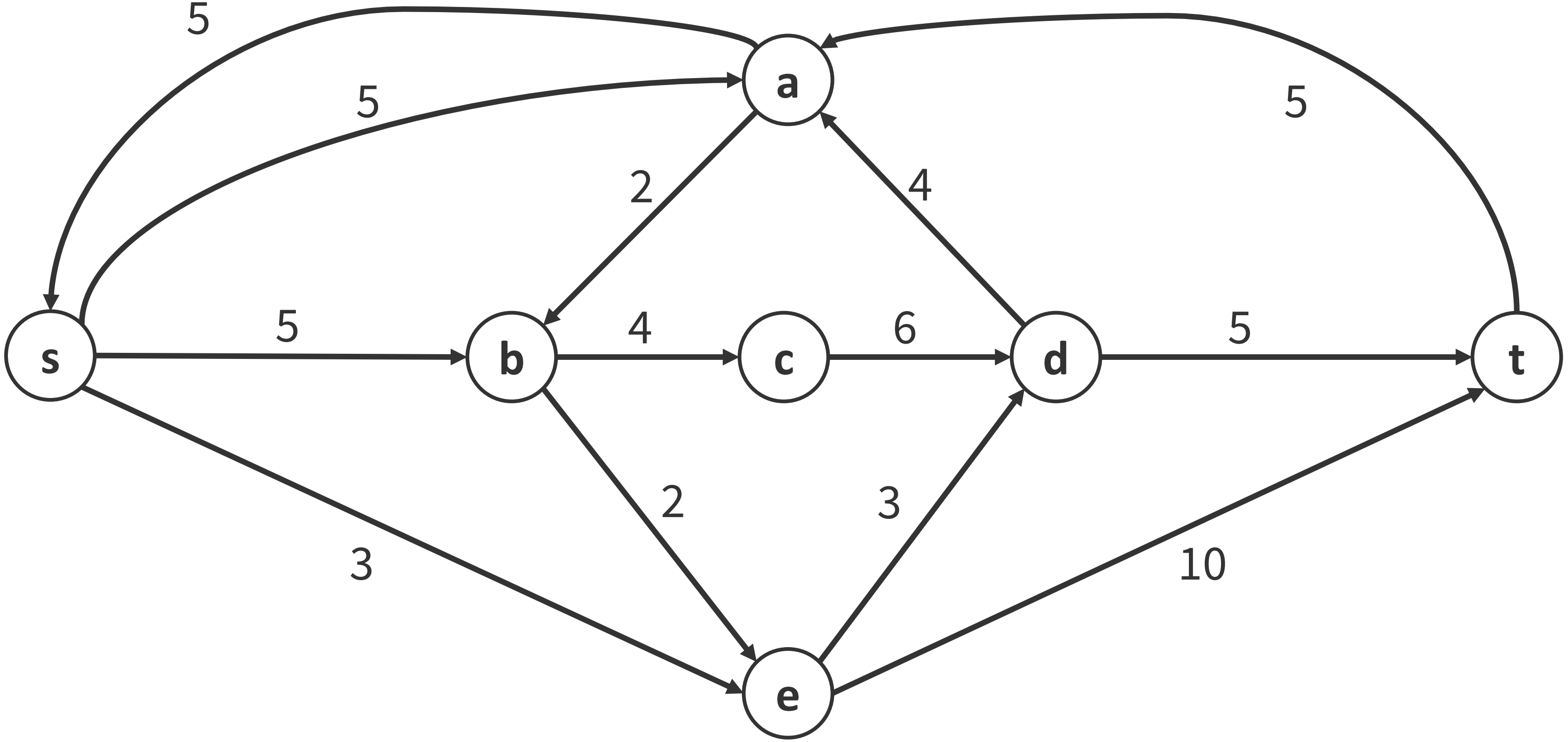


Problem 1 – Go With the Flow

Update the residual edges to push the maximum flow through

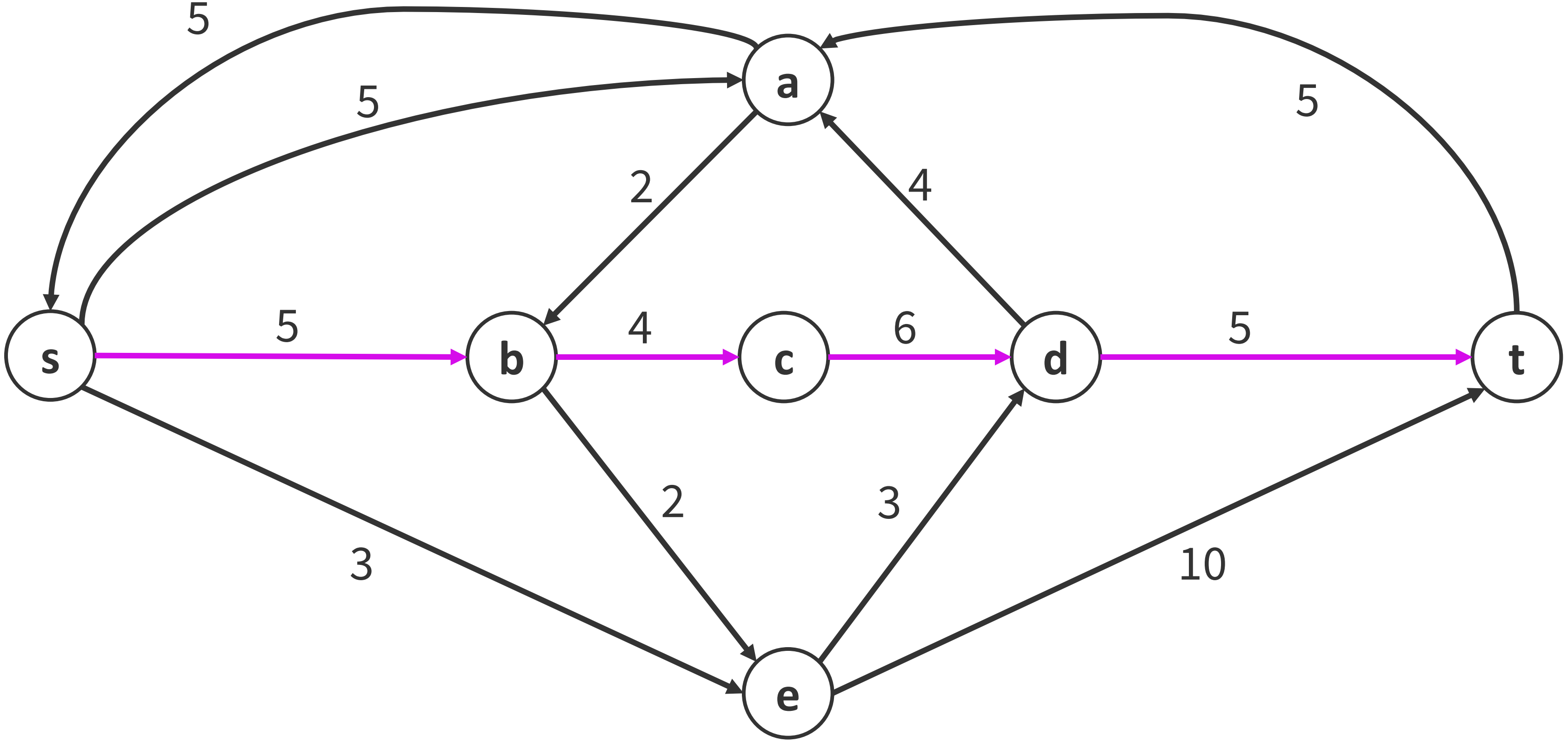


Problem 1 – Go With the Flow



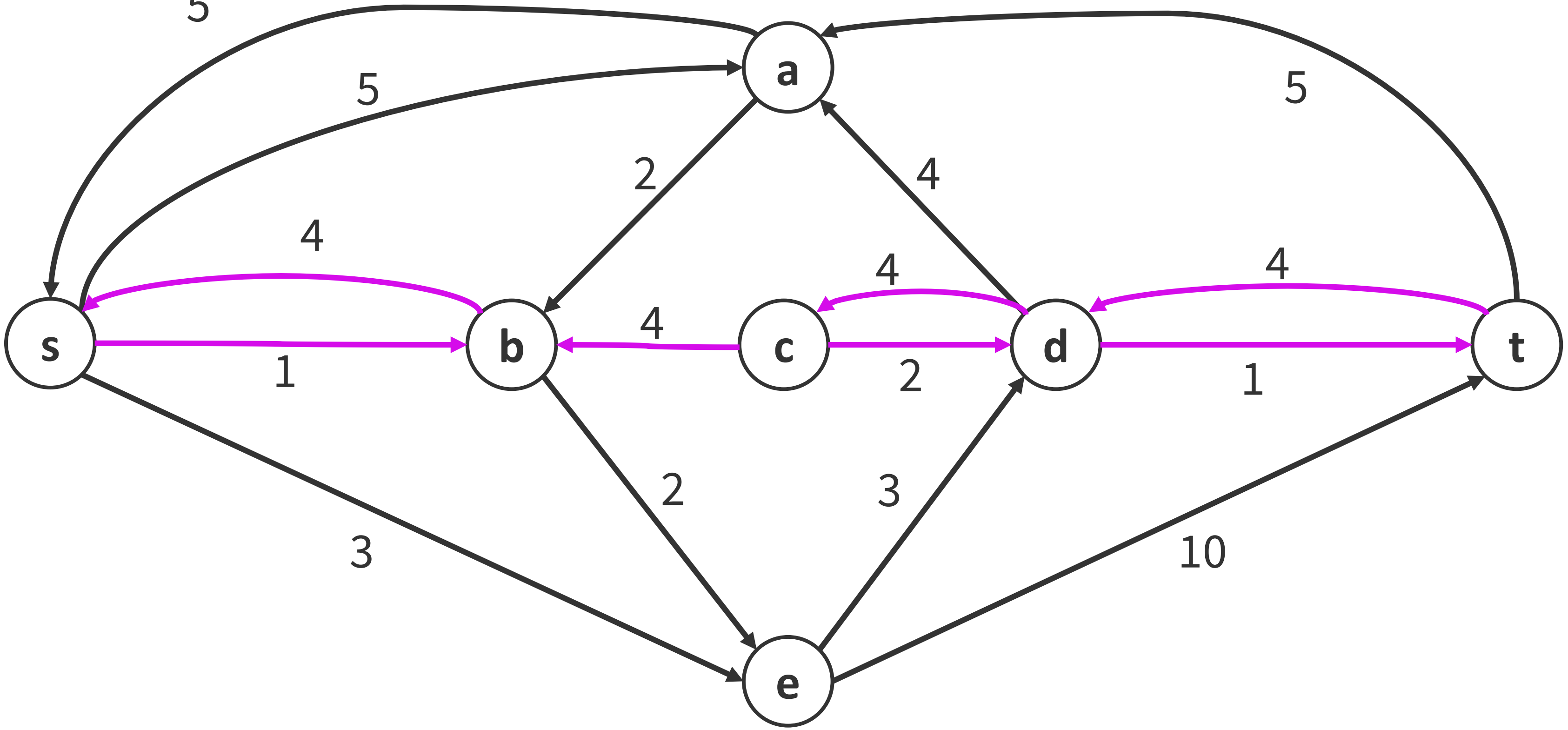
Problem 1 – Go With the Flow

Look to see if there is another path from s to t

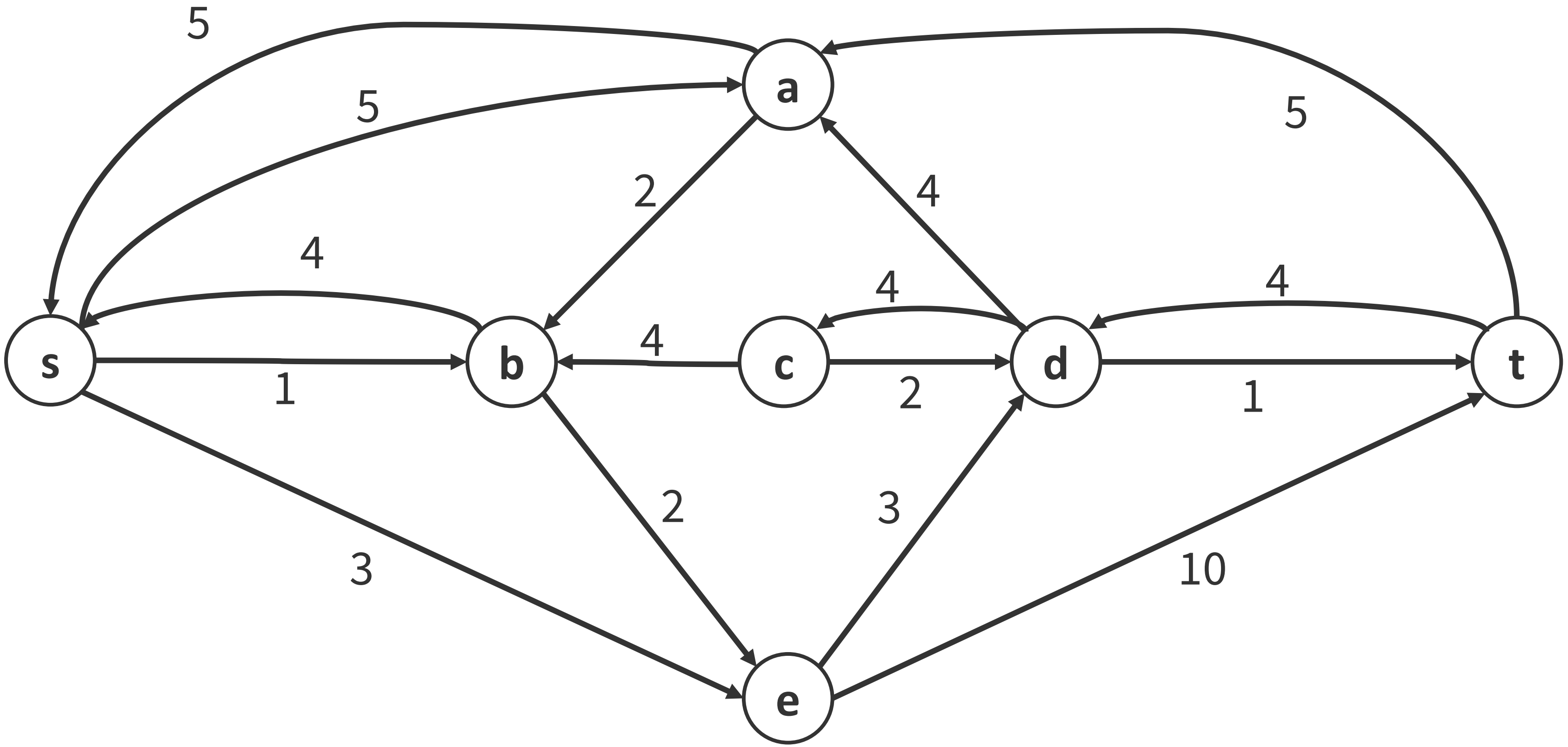


Problem 1 – Go With the Flow

Update the residual edges

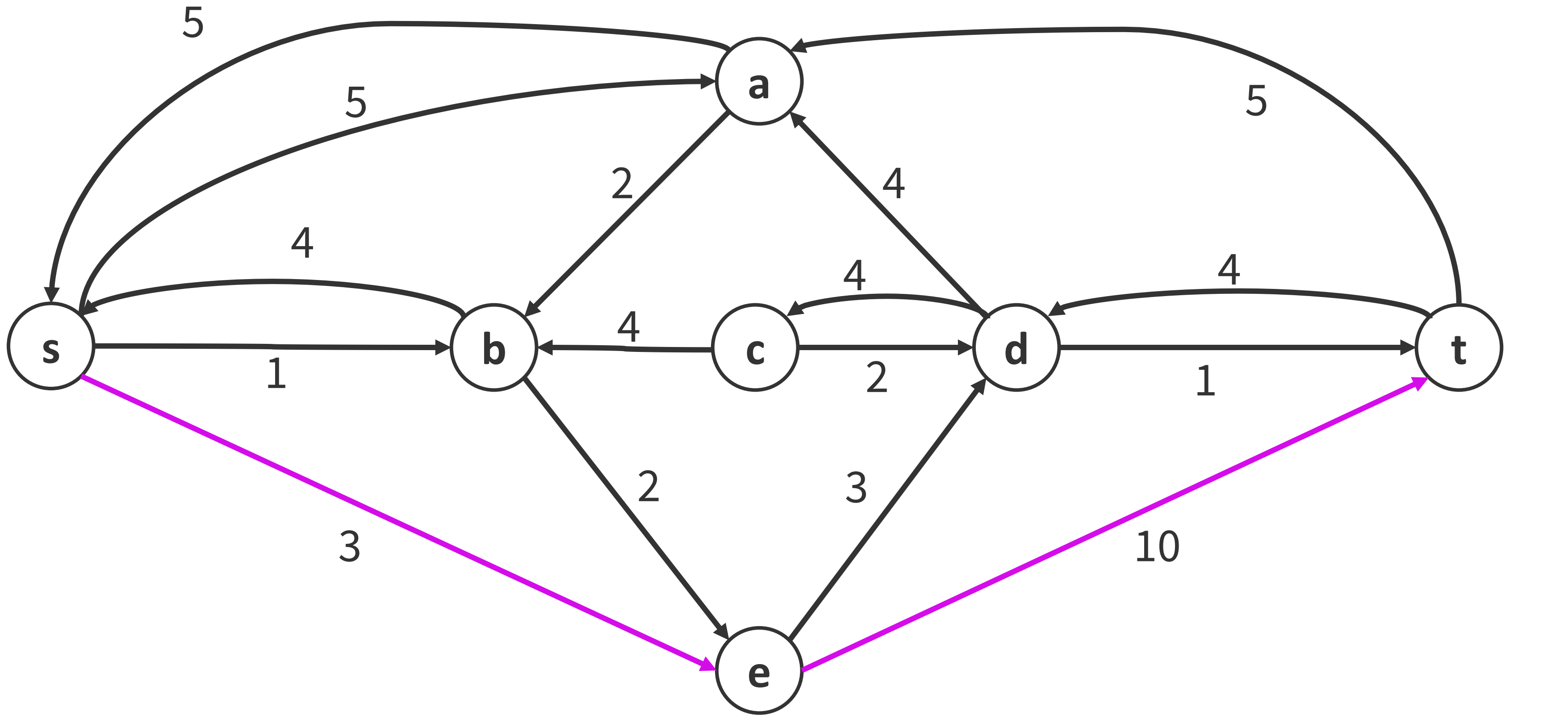


Problem 1 – Go With the Flow



Problem 1 – Go With the Flow

Look to see if there is another path from s to t

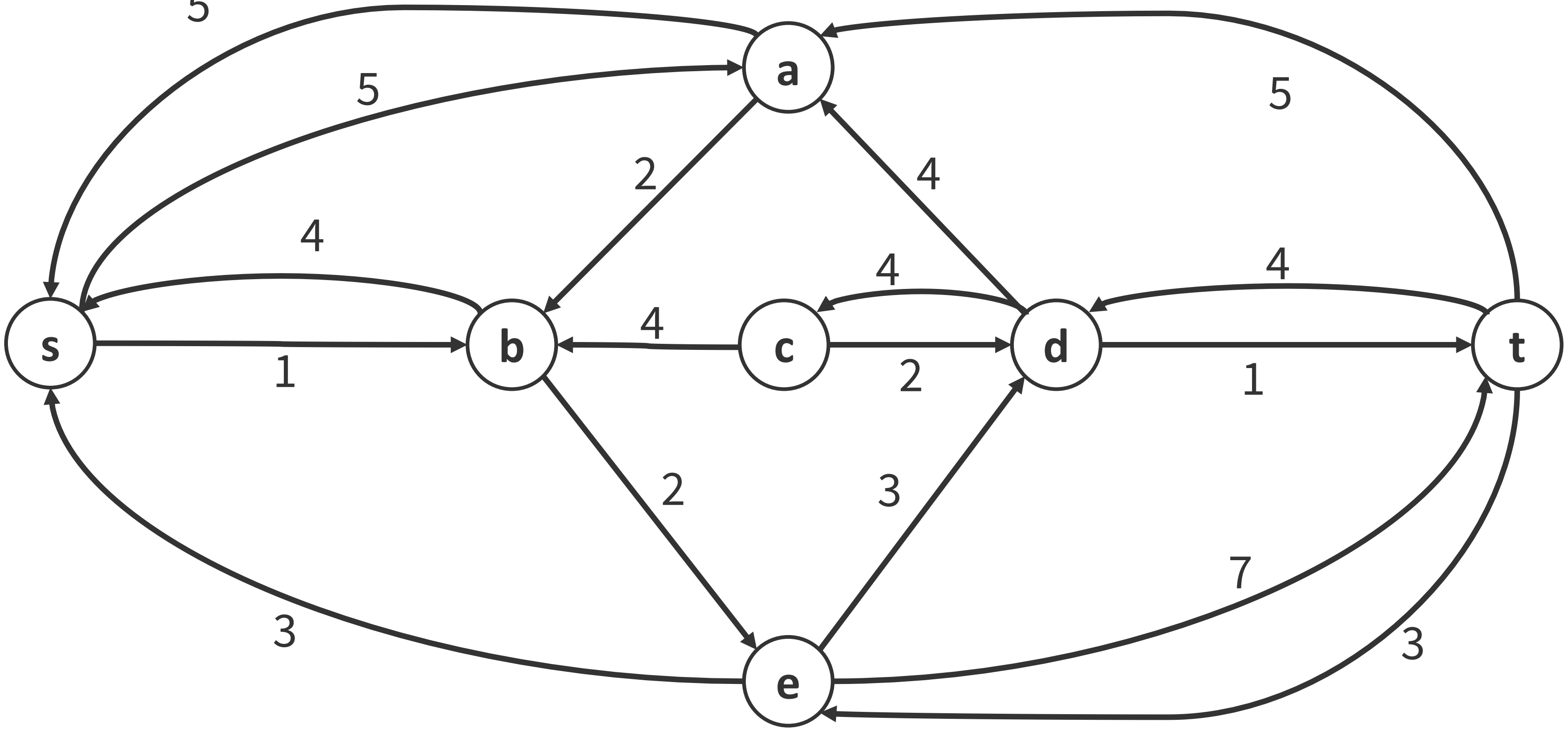


Problem 1 – Go With the Flow

Update the residual edges

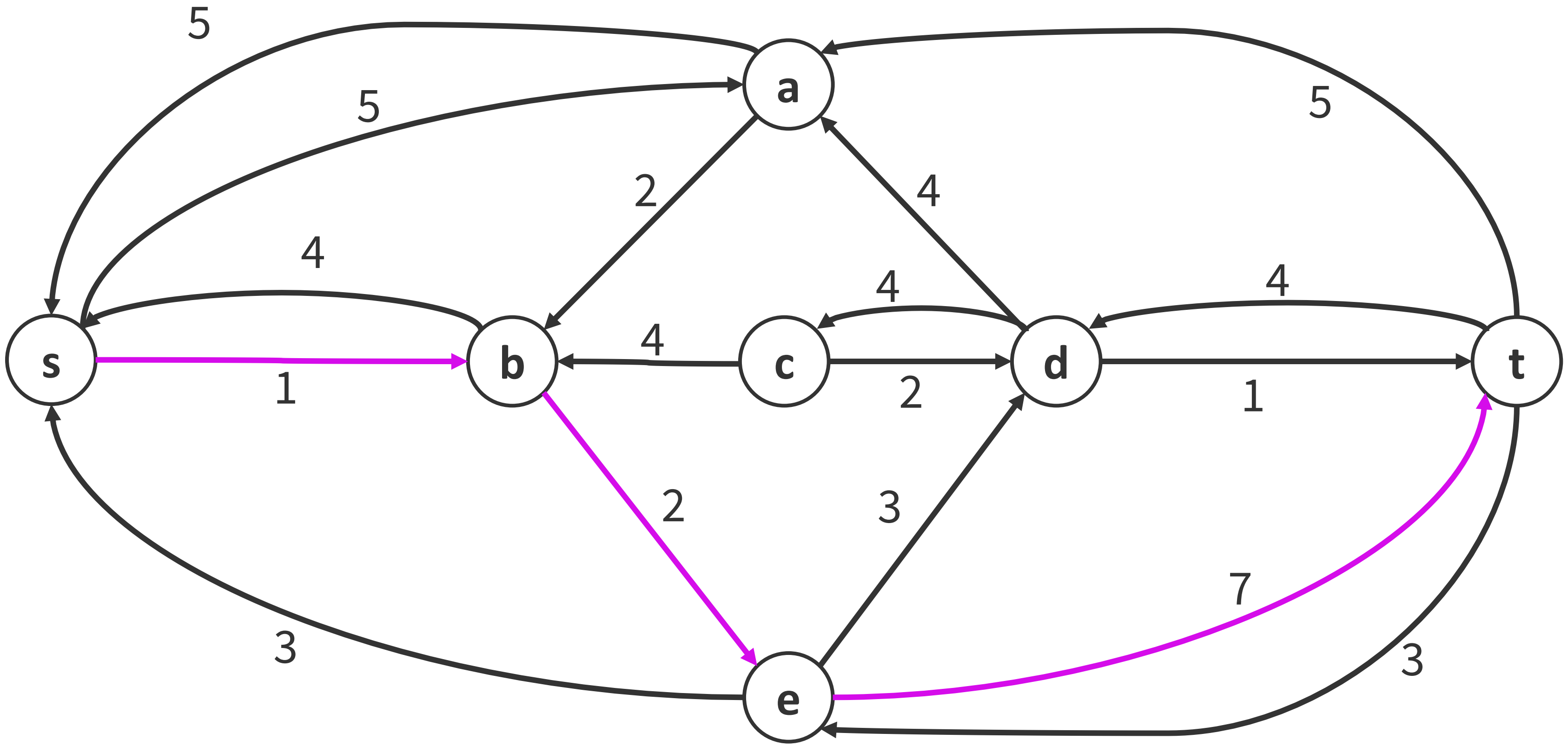
$$\begin{tikzcd}
 & & a \arrow[dll, bend right=40, "5"'] \arrow[dl, "2"'] & & \\
s \arrow[urr, bend left=10, "5"] \arrow[r, "1"'] & b \arrow[l, bend right, "4"'] \arrow[dr, "2"] & c \arrow[l, "4"'] \arrow[r, "2"'] & d \arrow[ul, "4"'] \arrow[l, bend right, "4"'] \arrow[r, "1"'] & t \arrow[ull, bend right=40, "5"'] \arrow[l, bend right, "4"'] \arrow[dll, bend left=40, "3"] \\
 & & e \arrow[ull, bend left=30, "3"] \arrow[ur, "3"'] \arrow[urr, bend right=20, "7"'] & &
\end{tikzcd}$$

Problem 1 – Go With the Flow



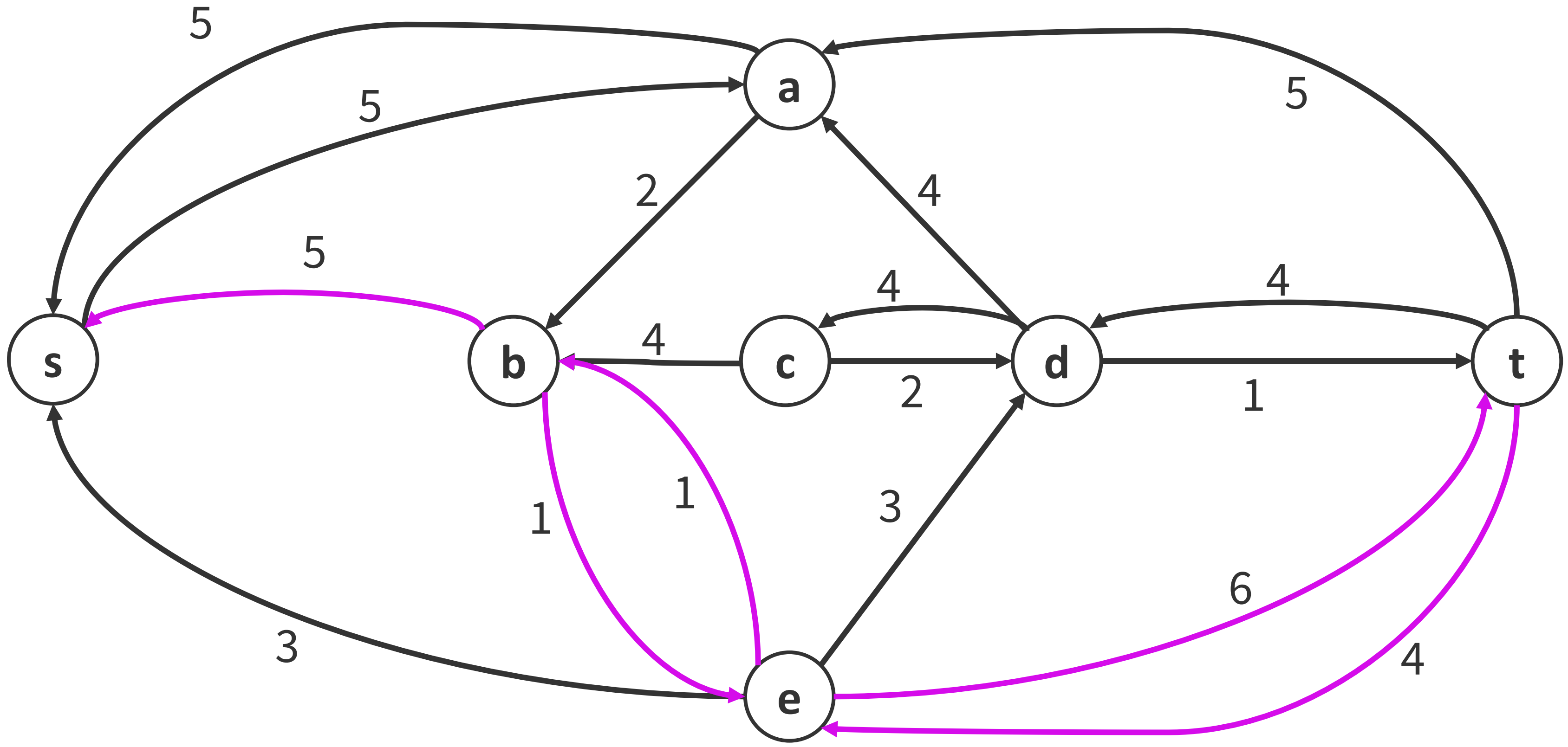
Problem 1 – Go With the Flow

Look to see if there is another path from s to t

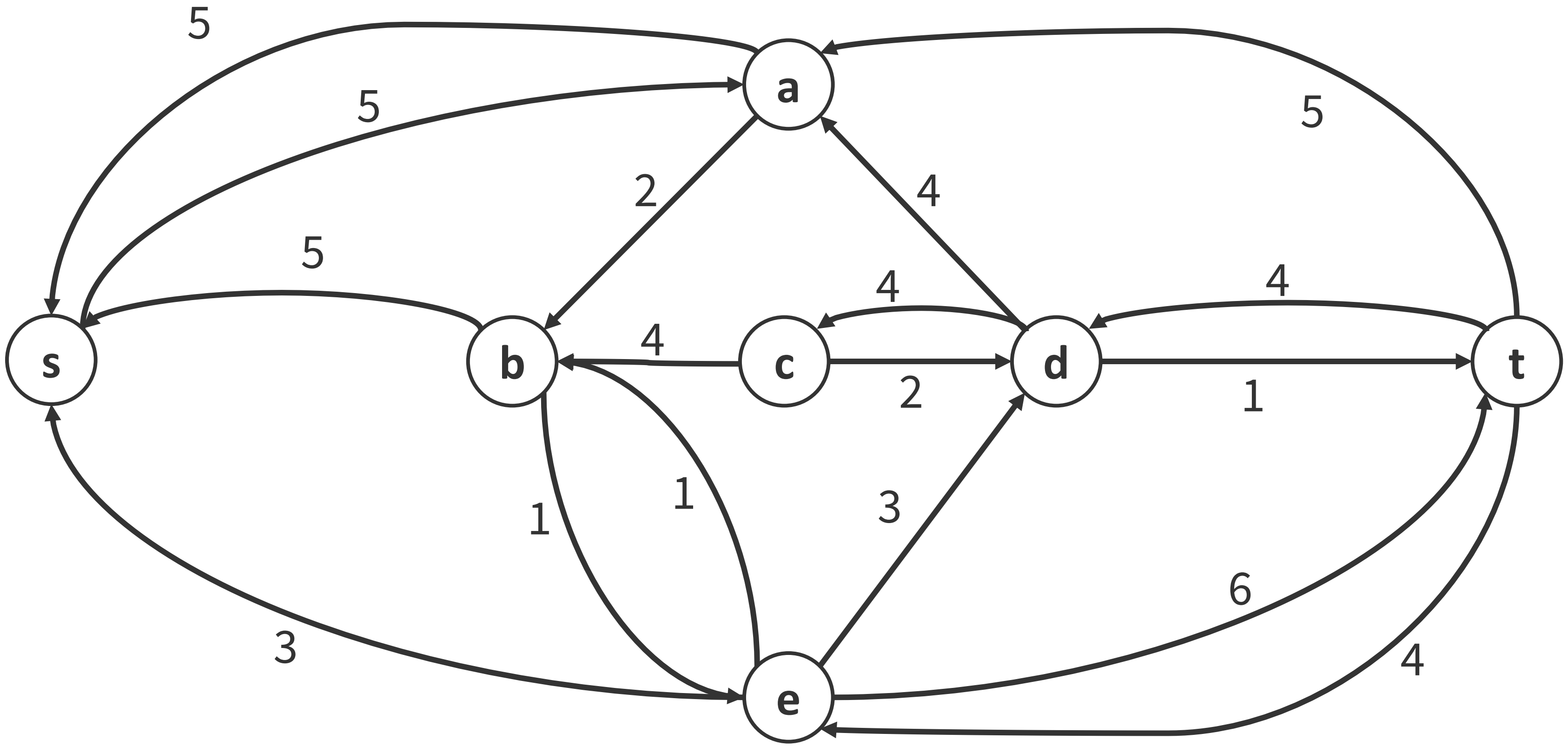


Problem 1 – Go With the Flow

Update the residual edges

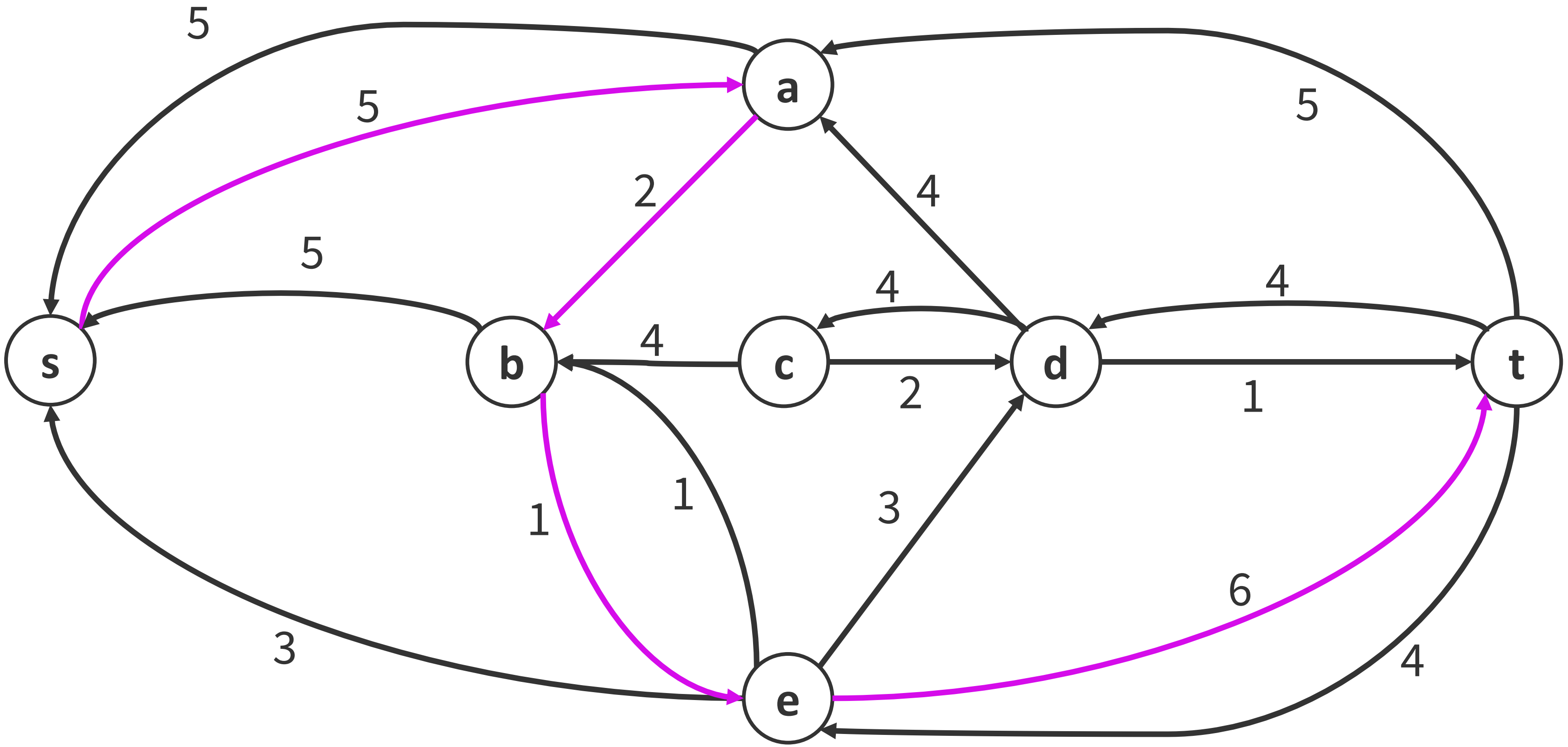


Problem 1 – Go With the Flow



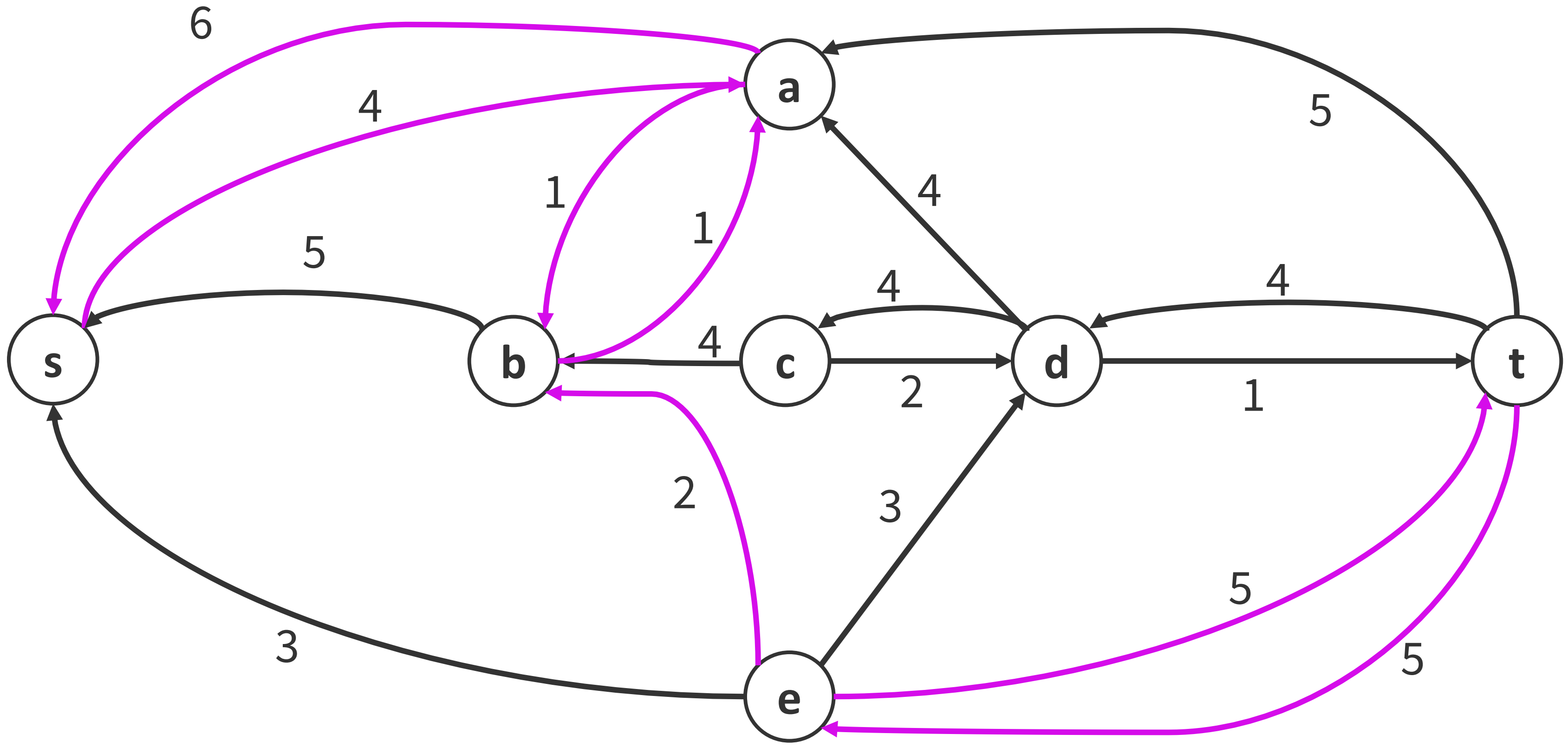
Problem 1 – Go With the Flow

Look to see if there is another path from s to t



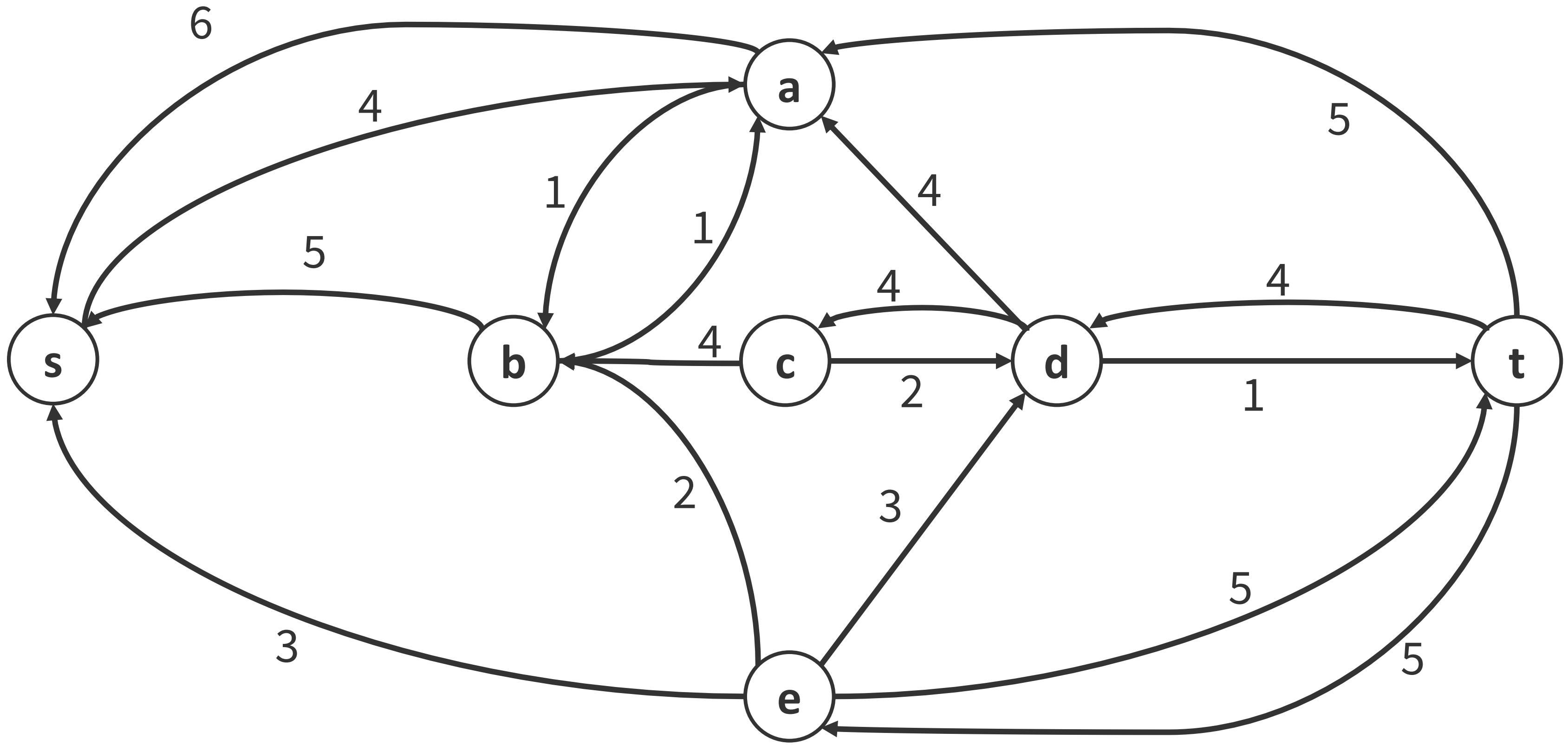
Problem 1 – Go With the Flow

Update the residual edges



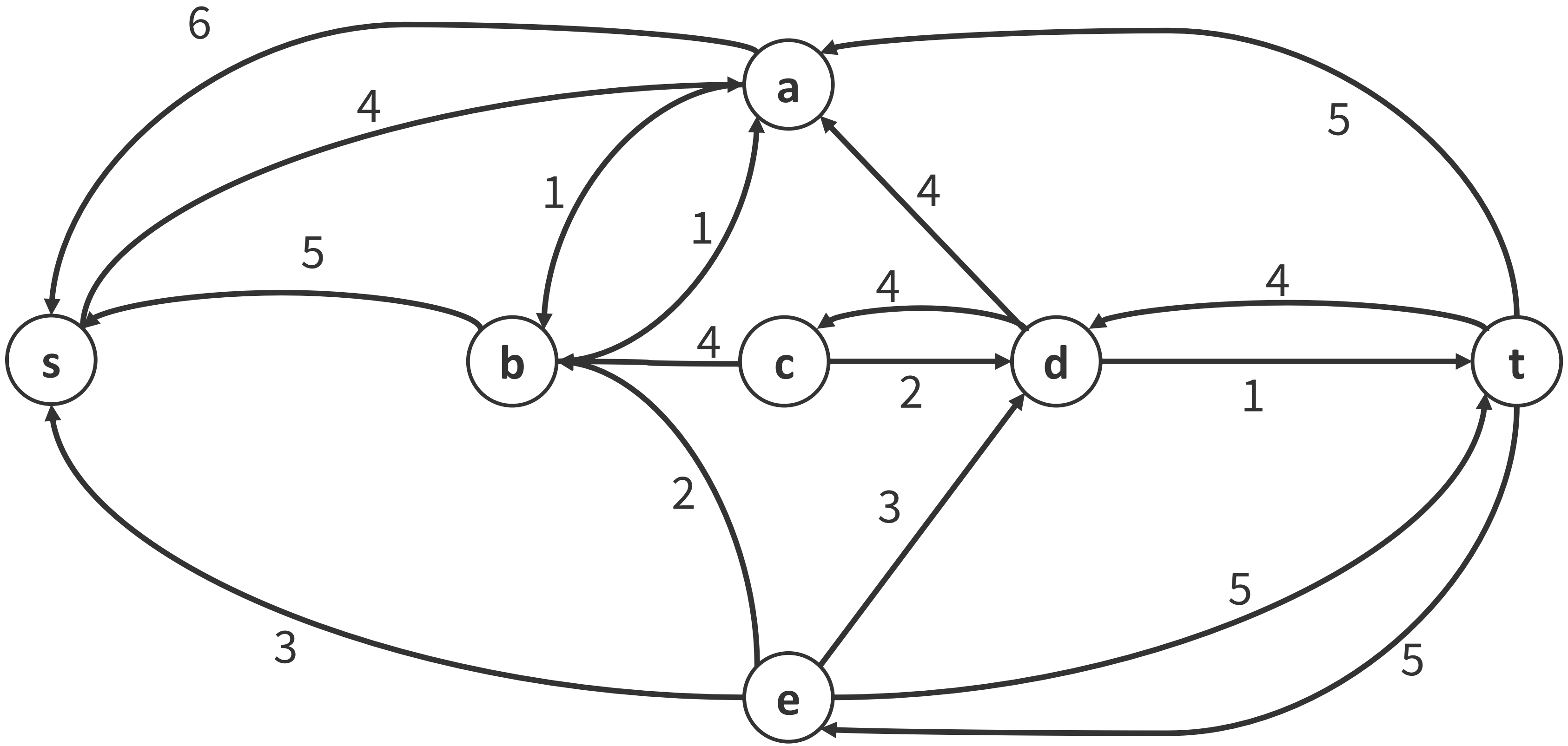
Problem 1 – Go With the Flow

Are there still any paths from s to t?



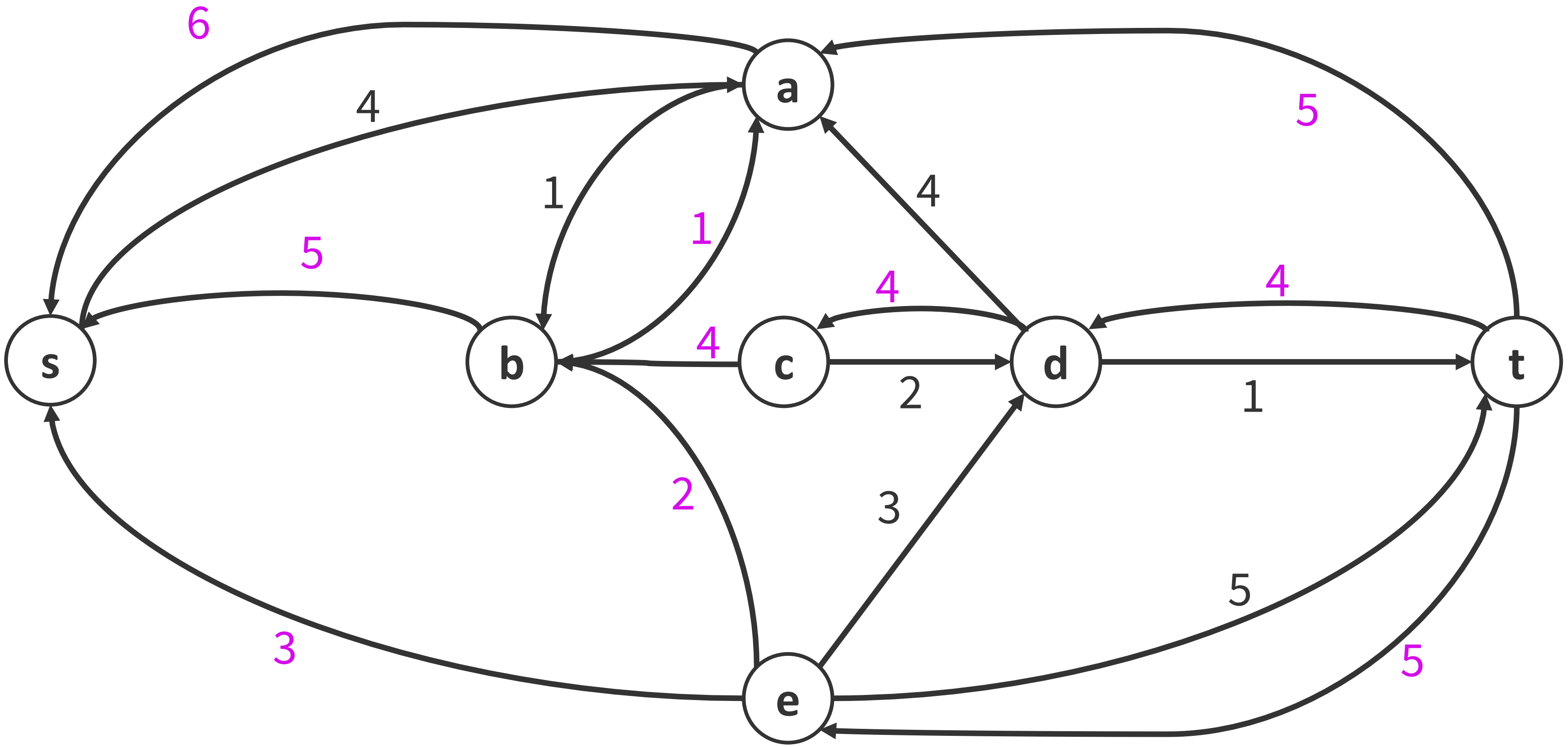
Problem 1 – Go With the Flow

No more paths, so we're done updating!



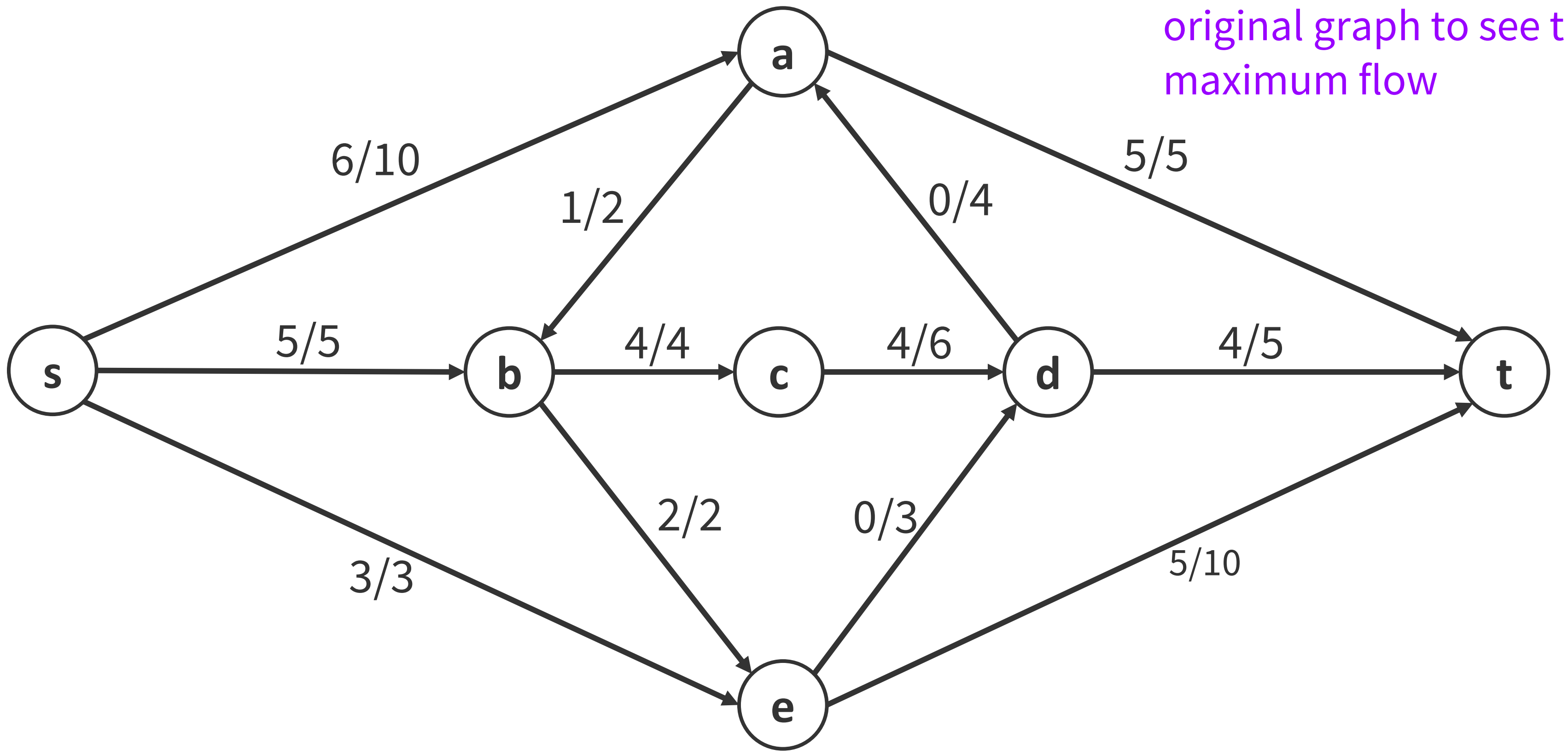
Problem 1 – Go With the Flow

All the residual edges going backwards show the flow we are sending down that path



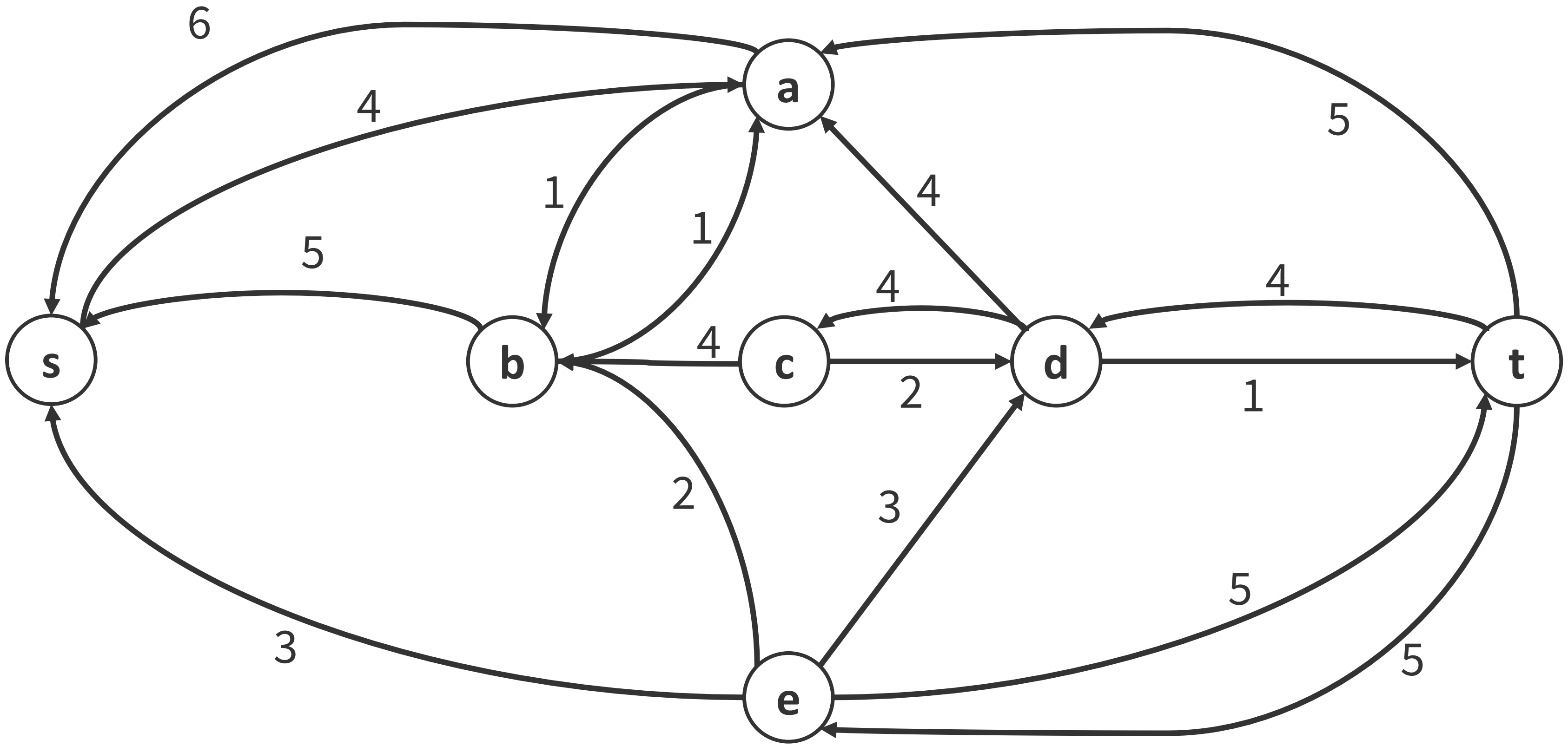
Problem 1 – Go With the Flow

We can put all these final flow values back into our original graph to see the maximum flow



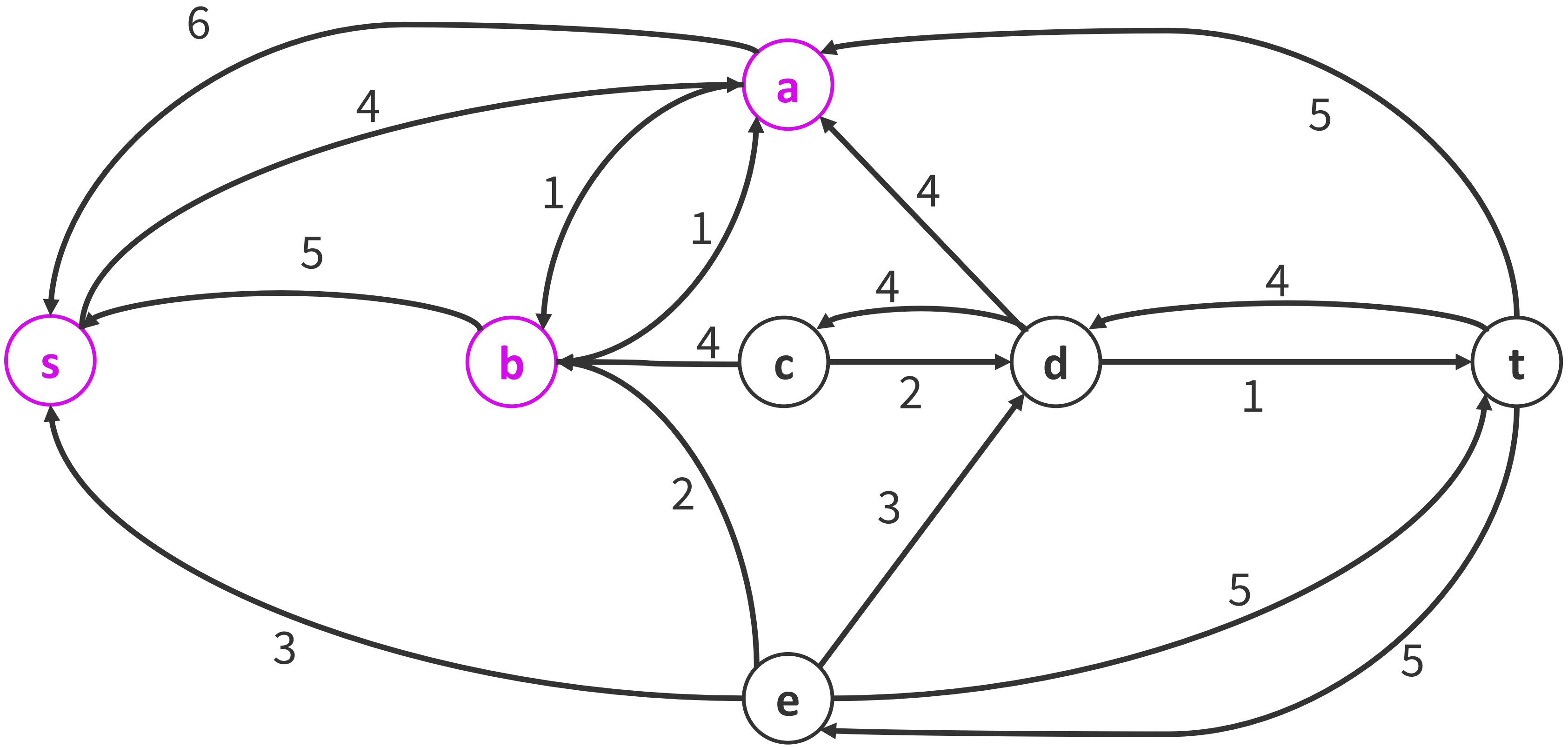
Problem 1 – Go With the Flow

Now let’s find the min cut.



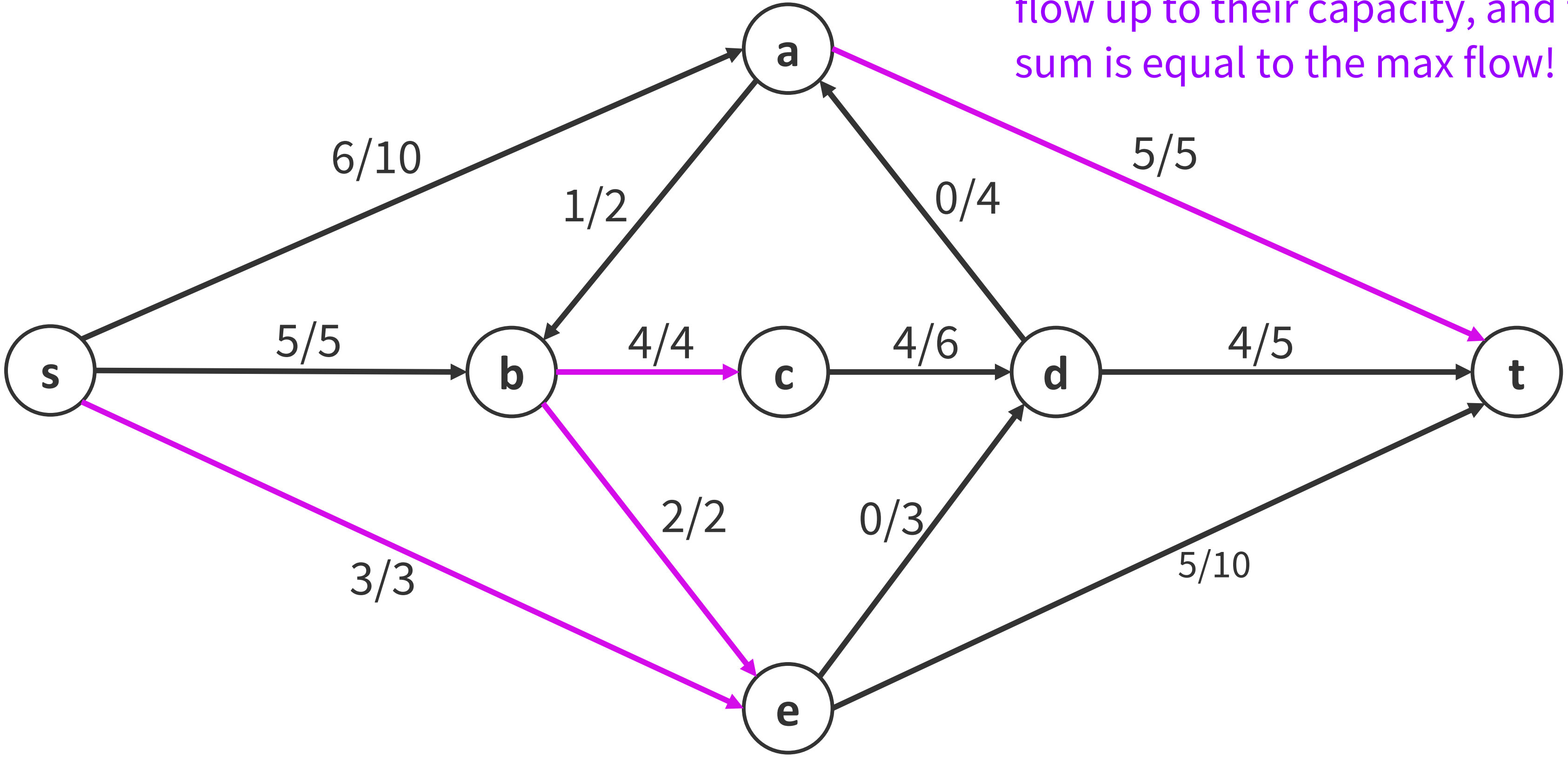
Problem 1 – Go With the Flow

The min cut is s and the vertices you can reach from it in the residual graph on one side, and everything else on the other side.



Problem 1 – Go With the Flow

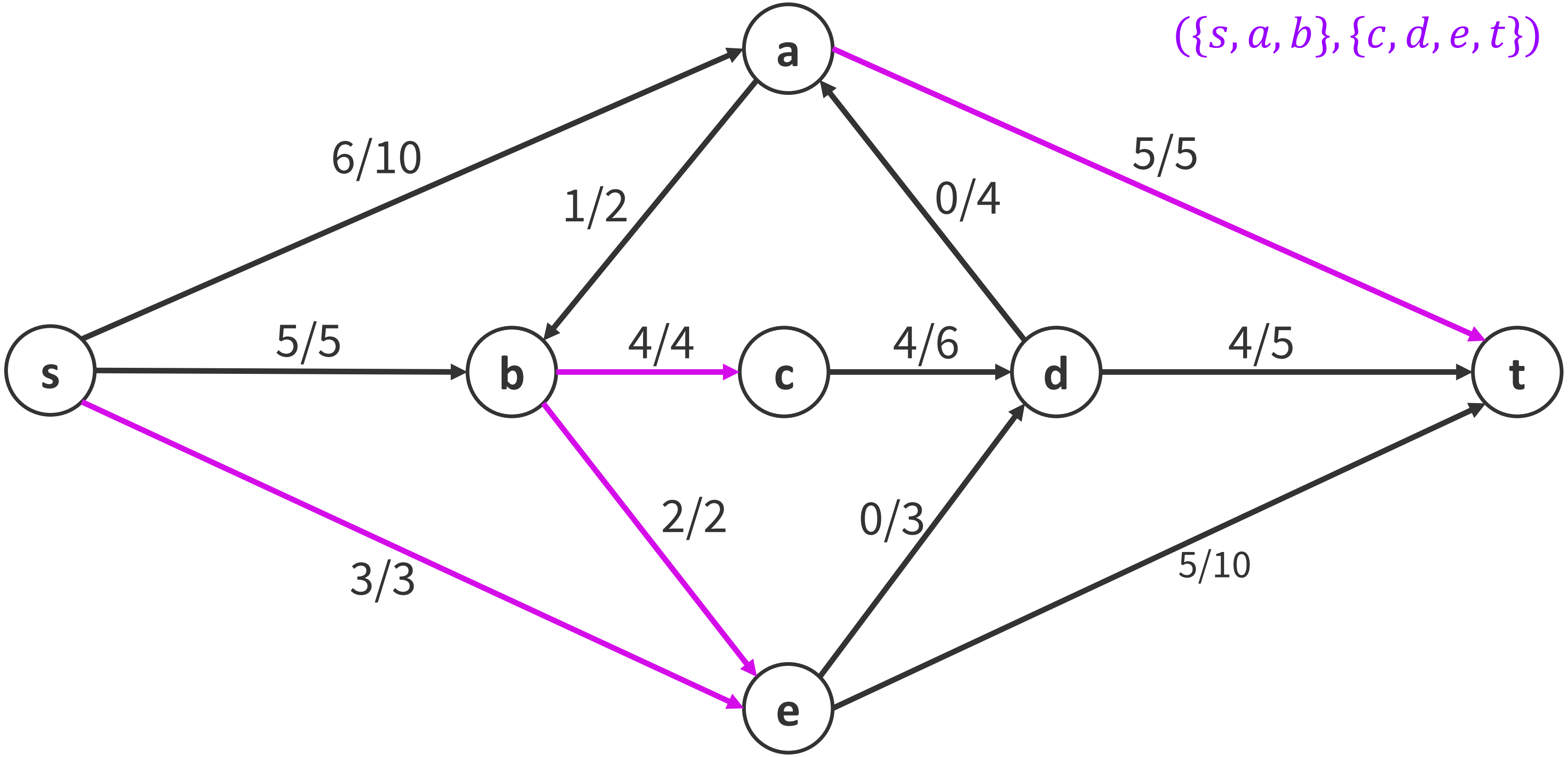
The edges going across the min cut from the s side to the t side all have flow up to their capacity, and the sum is equal to the max flow!



Problem 1 – Go With the Flow

The maximum flow is 14

The $s - t$ cut is $(\{s, a, b\}, \{c, d, e, t\})$



Max-Flow / Min-Cut Tricks

Max-Flow / Min-Cut

We can use the concepts of Max-Flow / Min-Cut and the Ford-Fulkerson algorithm to solve a wide variety of problems. Since we already have an algorithm, we can just call it like a library function.

Most of the difficulty comes in taking a problem and turning it into a good graph so that max-flow / min-cut gives us the solution we are actually looking for. So how can we do it?

The Strategy

1. Read the Problem Carefully
2. Make a Basic Model
3. Brainstorm: How can you fix the graph?
4. Correctness and Running Time

The Tricks

We have three tricks that can be really helpful in converting a problem into a good form for max-flow or min-cut. Sometimes you only need one, but sometimes you can use them in a combination. There are other things you might need to do in a given problem, but these are three very common tricks to try:

- Add “dummy vertices” for source or sink
- Split vertices to add vertex capacity
- Use infinite weight for edges that shouldn’t be considered for max-flow or min-cut

We have m professors and n graduate students. Each professor have a cap c_i on the number of graduate students they can advise, and each graduate student have a specified set of interested professors that they would like to work with.
Design a polynomial time algorithm that returns yes if there exists a valid advising schedule on students’ side (i.e. each student has exactly one advisor and advisors don't exceed the cap) or false if that's not possible.

What if each professor also have a set of students that they would like to advise (i.e. professor may not accept all students)?
What if students can request more than one professor? (Here we say an advising schedule is valid if each student gets exactly the number of professor they request as advisors)

Given an $n \times n$ chess board where some cells are removed. Design a polynomial time algorithm to find the maximum number of knights that can be placed on this board such that no two knights attack each other.

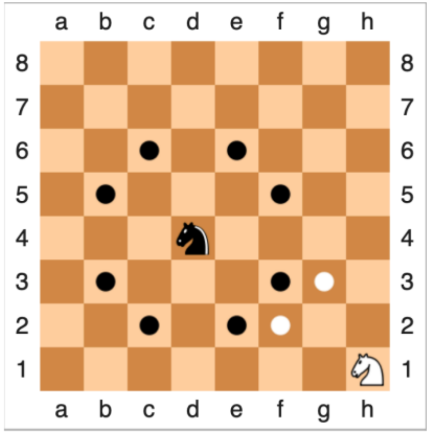
For example in the following 3×3 chessboard the removed cells are marked with X. You can put 4 knights such that no two can attack each other as we did in the right. The location of every knight is marked with a •.

X		
		X
	X	

$\Rightarrow$

X		•
	•	X
•	X	•

Please see the following image for locations that a knight can attack. In general a knight can attack at most 8 cells if they exist. For example, the white knight can only attack two cells in the following picture.



Source: 421 sp22 hw7

- Use Konig Theorem to design algorithm and show correctness: For any bipartite graph, the size of maximum matching equals to the size of minimal vertex cover.

- What if we create a new kind of piece, Z. Define a position that are attacked by Z if it’s diagonally neighbored with Z. So in general, Z can attack 4 positions. Z works like Bishop, just that they can only attack positions on the diagonal one step away but not arbitrary many steps.
- The problem is the same. Remove some cells and design a polynomial time algorithm that returns maximum number of non-attacking Zs.

- Given a network flow instance with all integer capacity and a feasible flow with integer flow value f (but flow value on all edges are not necessarily integers), determine if there exists a flow with value f + 1 in O(m + n) time.

- Do a B/DFS starting from s on residual graph.
- Return true iff t is reachable.

That’s All, Folks!

Thanks for coming to section this week!
Any questions?