CSE 421 Section 7

Network Flows

Announcements & Reminders

- **HW5**
    - Due yesterday!
- **HW6**
    - Due next Wednesday as usual

Linear Programming Review

Linear Programming

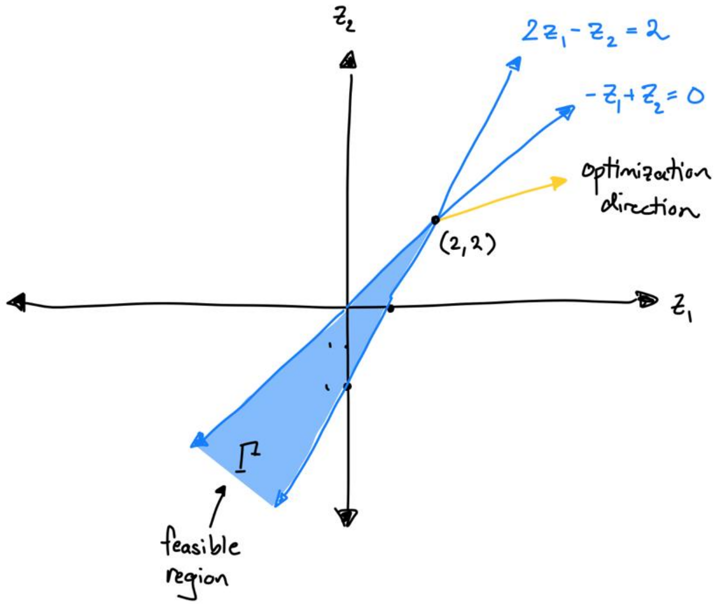
- **Goal:** Maximize an objective function given some constraints
  - Constraints and objective function are linear equations
  - Output is values for variables in the objective function

(example from lecture)

Find values for z_1 and z_2

Objective function to maximize:
$10z_1 + z_2$

Constraints:
$2z_1 - z_2 \leq 2$
$-z_1 + z_2 \leq 0$



Standard Form

A standard way to write the objective function and constraints.

$$\max \quad c^T x$$

$$\text{s.t.} \quad \begin{cases} Ax \leq b \\ x \geq 0 \end{cases}$$

Our objective function is linear (a line), so it can be written as $c_1x_1 + c_2x_2 + \ldots$

Our constraints are also linear, so we stack them up to get a matrix

We always have the constraint that our variable values must be positive (we will see ways to get around this…)

Standard Form

What if we want an equality constraint instead of an inequality?

Split into two inequalities!

$$x_1 + x_2 = 2 \quad \Rightarrow \quad \begin{aligned} x_1 + x_2 &\leq 2 \\ x_1 + x_2 &\geq 2 \end{aligned}$$

Standard Form

What if we wanted to minimize the function instead of maximize?

Maximize the negative of the function instead!

$$
\begin{aligned}
min \quad & y_1 - 2y_2 \\
s.t., \quad & y_1 + 2y_2 = 3 \\
& y_1 - y_2 \geq 1 \\
& y_1, y_2 \geq 0
\end{aligned}
$$

$$
\begin{aligned}
max \quad & -y_1 + 2y_2 \\
s.t., \quad & y_1 + 2y_2 \leq 3 \\
& -(y_1 + 2y_2) \leq -3 \\
& -(y_1 - y_2) \leq -1 \\
& y_1, y_2 \geq 0
\end{aligned}
$$

Standard Form

What if we want to allow some variables to be negative?

Introduce dummy variables!

$$
\begin{aligned}
max \quad & y_1 \\
s.t., \quad & y_1 + y_2 \leq 3 \\
& y_2 \geq 0
\end{aligned}
$$

Replace y_1 with $z_1 - z_1'$ →

$$
\begin{aligned}
max \quad & z_1 - z_1' \\
s.t., \quad & (z_1 - z_1') + y_2 \leq 3 \\
& z_1, z_1', y_2 \geq 0
\end{aligned}
$$

Standard Form

Conclusion: Any LP can be written in standard form.

Example LP

Suppose you want to schedule a diet for yourself. There are four categories of food: veggies, meat, fruits, and dairy. Each category has its own (p)rice, (c)alorie and (h)appiness per pound:

	veggies	meat	fruits	dairy
price	p_v	p_m	p_f	p_d
calorie	c_v	c_m	c_f	c_d
happiness	h_v	h_m	h_f	h_d

For example, we eat 0.5lb of meat and 0.2lb of of fruits, our happiness is $0.5h_m + 0.2h_f$

Constraints: you can eat a maximum of 1500 calories and can spend at most $20.
Goal: Maximize happiness.

Example LP

Let x_v, x_m, x_f, and x_d represent how much veggies, meat, fruits and dairy you eat.

What is the objective function to maximize?

$$\max \quad x_v h_v + x_m h_m + x_f h_f + x_d h_d$$

What are the constraints?

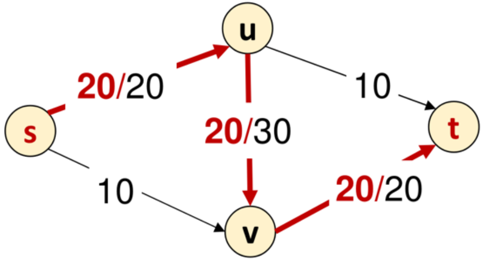
$$x_v p_v + x_m p_m + x_f p_f + x_d p_d \leq 20$$
$$x_v c_v + x_m c_m + x_f c_f + x_d c_d \leq 1500$$
$$x_v, x_m, x_f, x_d \geq 0$$

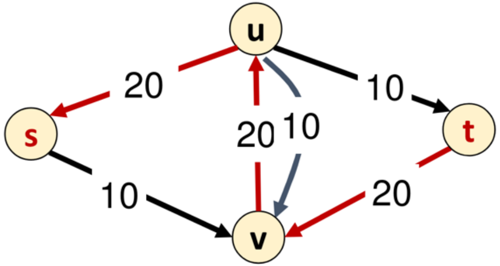
Algorithms for network flows

Algorithms for network flows

Ford–Fulkerson is a class of algorithms to compute maximum flow.

1. Let the residual graph G_f be initialized to G.
2. While there exists an s-t path P in G_f,
   a. Let c be the minimum capacity along this path.
   b. Update f to push c flow along P.
   c. Update edges in G_f along P.





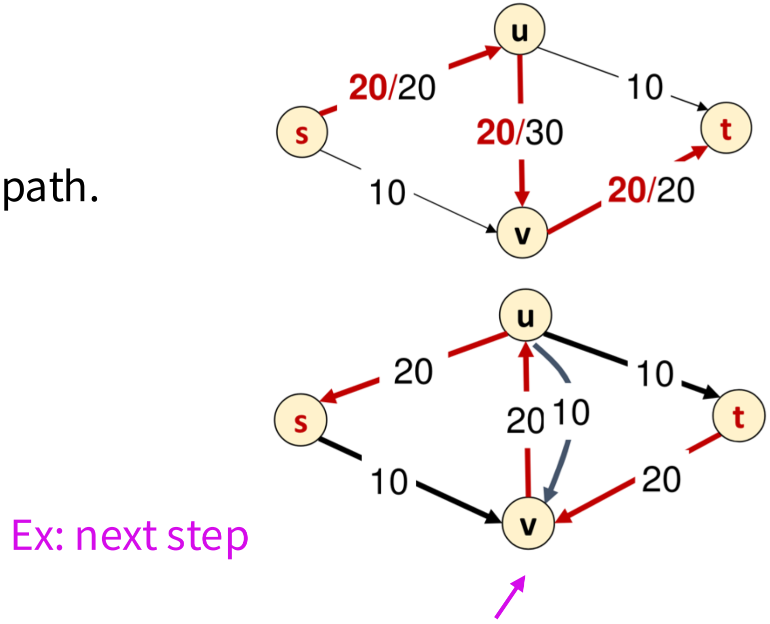
Warmup: How does the update work?

Algorithms for network flows

Ford–Fulkerson is a class of algorithms to compute maximum flow.

1. Let the residual graph G_f be initialized to G.
2. While there exists an s-t path P in G_f,
   a. Let c be the minimum capacity along this path.
   b. Update f to push c flow along P.
   c. Update edges in G_f along P.

If $e \in P$ is a forward edge, increase $f(e)$ by c.
If $e \in P$ is a backward edge, decrease $f(e)$ by c.



Algorithms for network flows

Ford–Fulkerson is a class of algorithms to compute maximum flow.

- Edmonds–Karp implementation: BFS (unweighted shortest path) to select s-t path

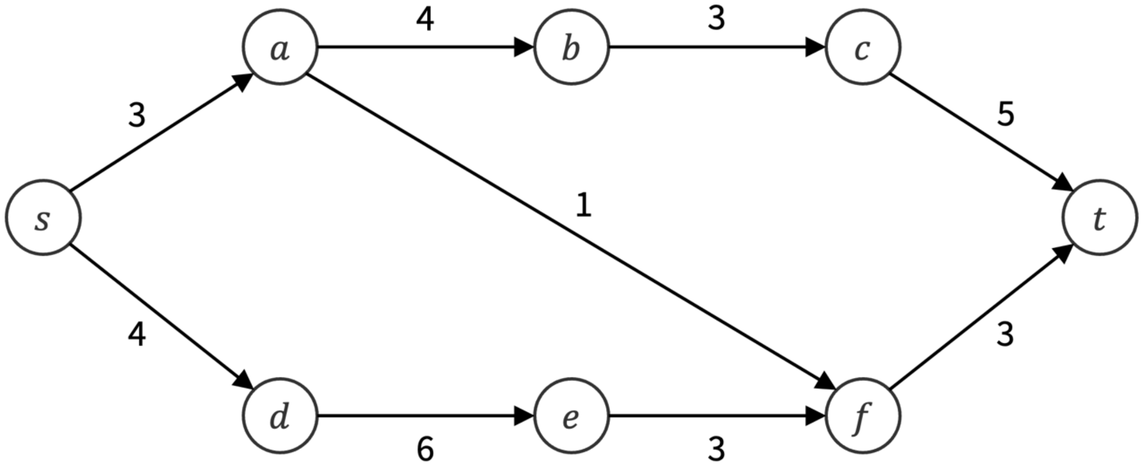
Capacity scaling algorithm: Process capacities one bit at a time

Ford–Fulkerson with BFS		**Capacity scaling**
Ford–Fulkerson bound	**Edmonds–Karp bound**	
$O(mnC)$	$O(m^2n)$	$O(m^2 \log C)$
good when all capacities are small	good with many large capacities	good when there are a few large capacities

Flow algorithms practice

Problem 1 – Flow algorithms practice

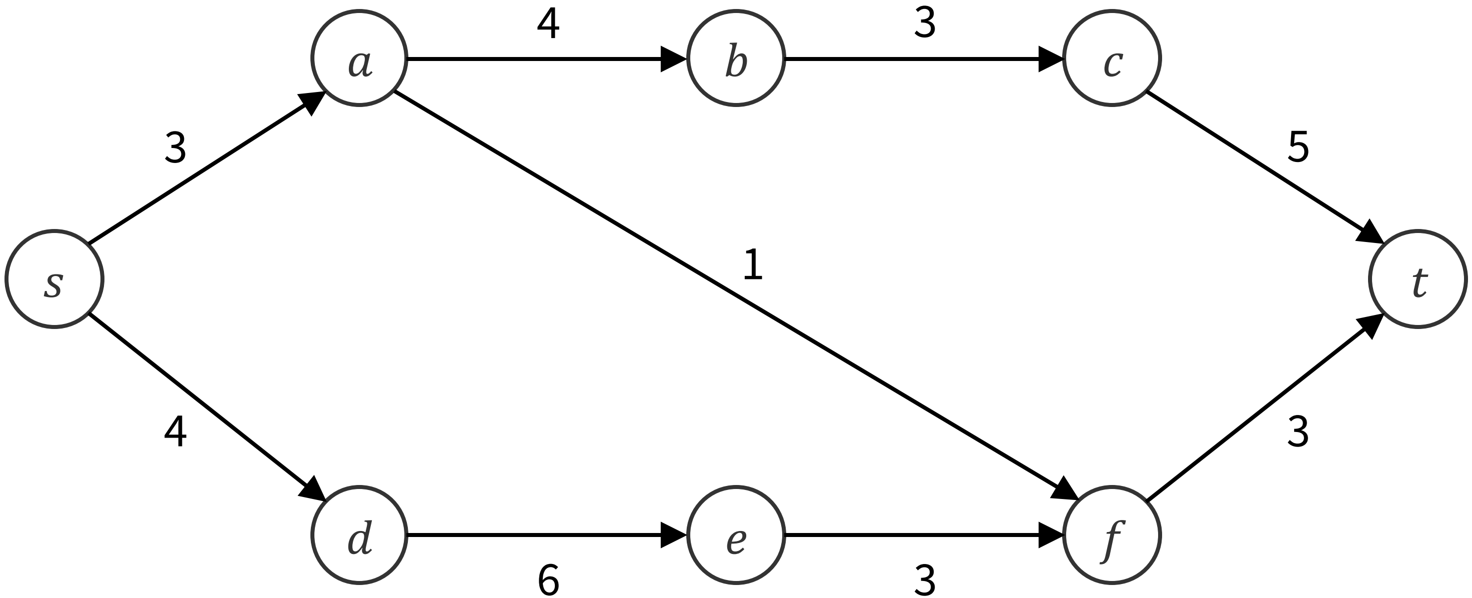
Using Ford–Fulkerson with BFS, find the maximum s-t flow in the graph G below, the corresponding residual graph, and minimum cut.



Work on this with the people around you, then we'll check!

Solution

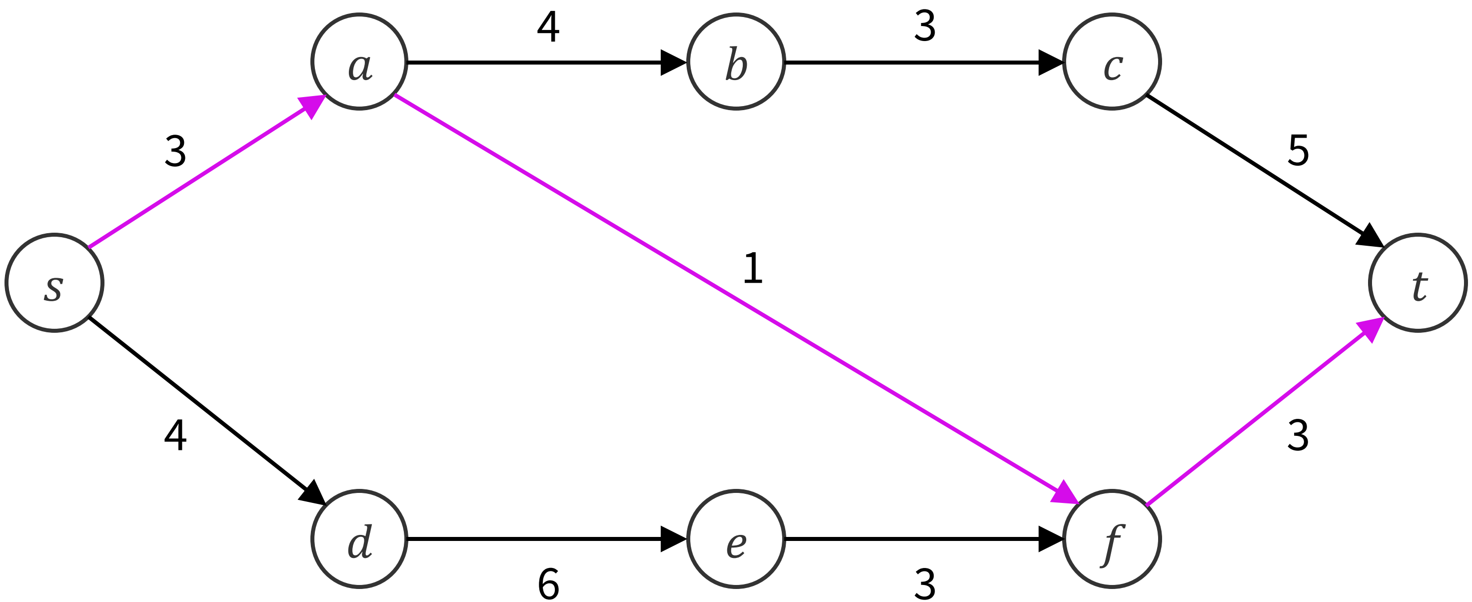
Problem 1 – Flow algorithms practice



Here is our starting residual graph.

Solution

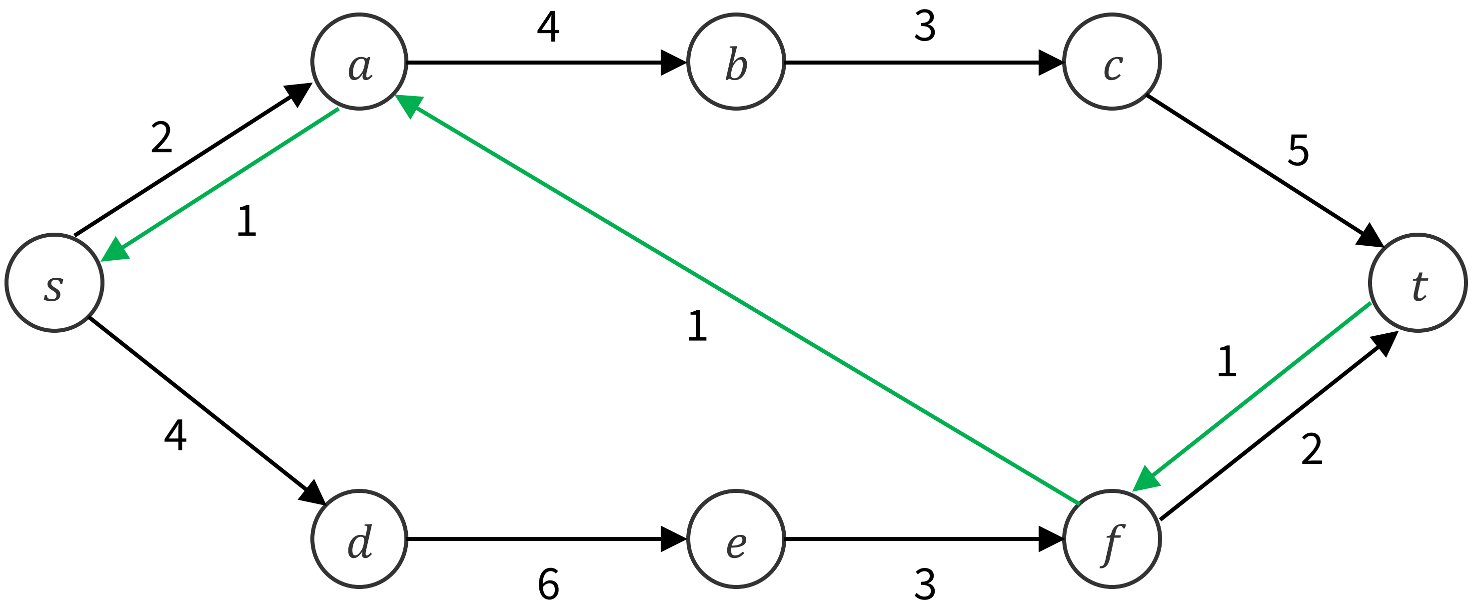
Problem 1 – Flow algorithms practice



Find the shortest s-t path.

Solution

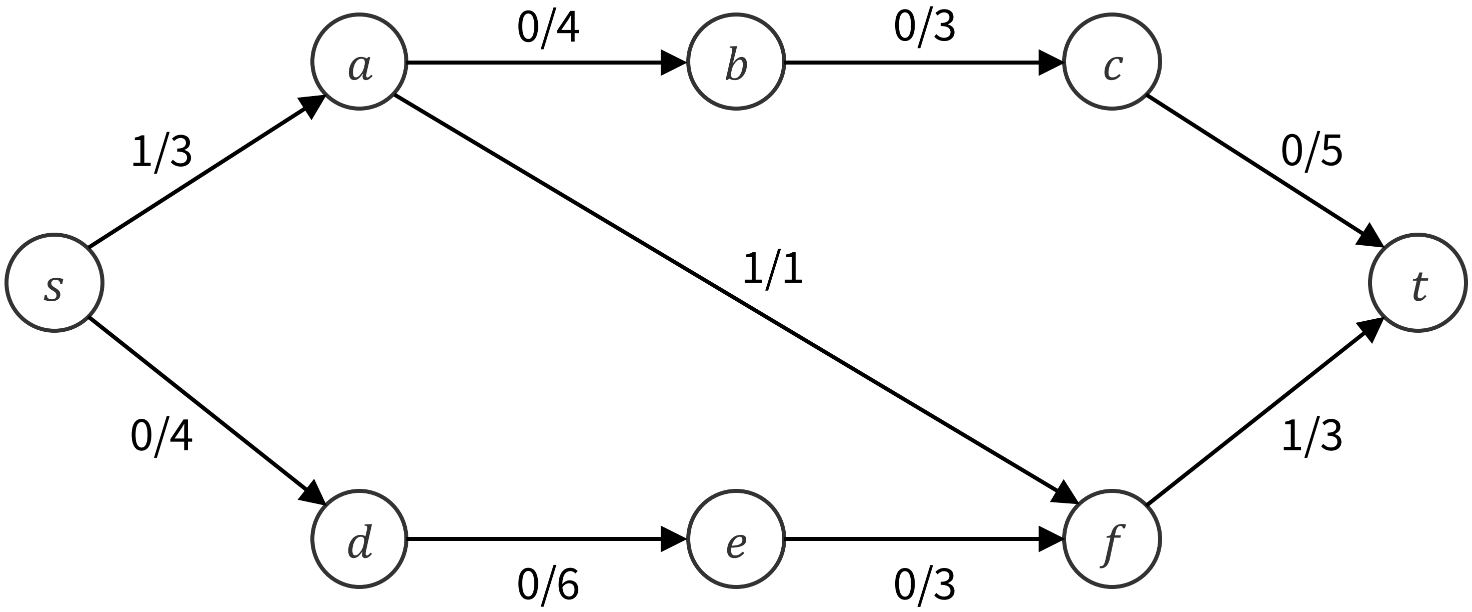
Problem 1 – Flow algorithms practice



Update the residual graph.

Solution

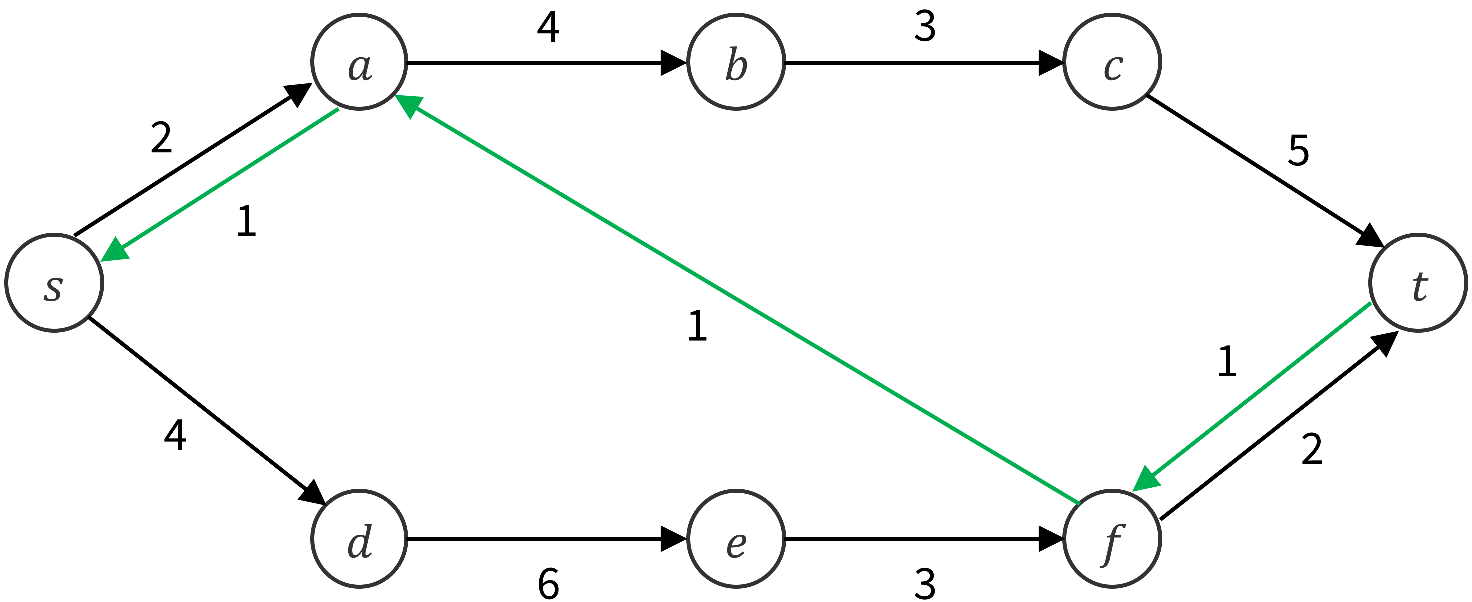
Problem 1 – Flow algorithms practice



Update the flow.

Solution

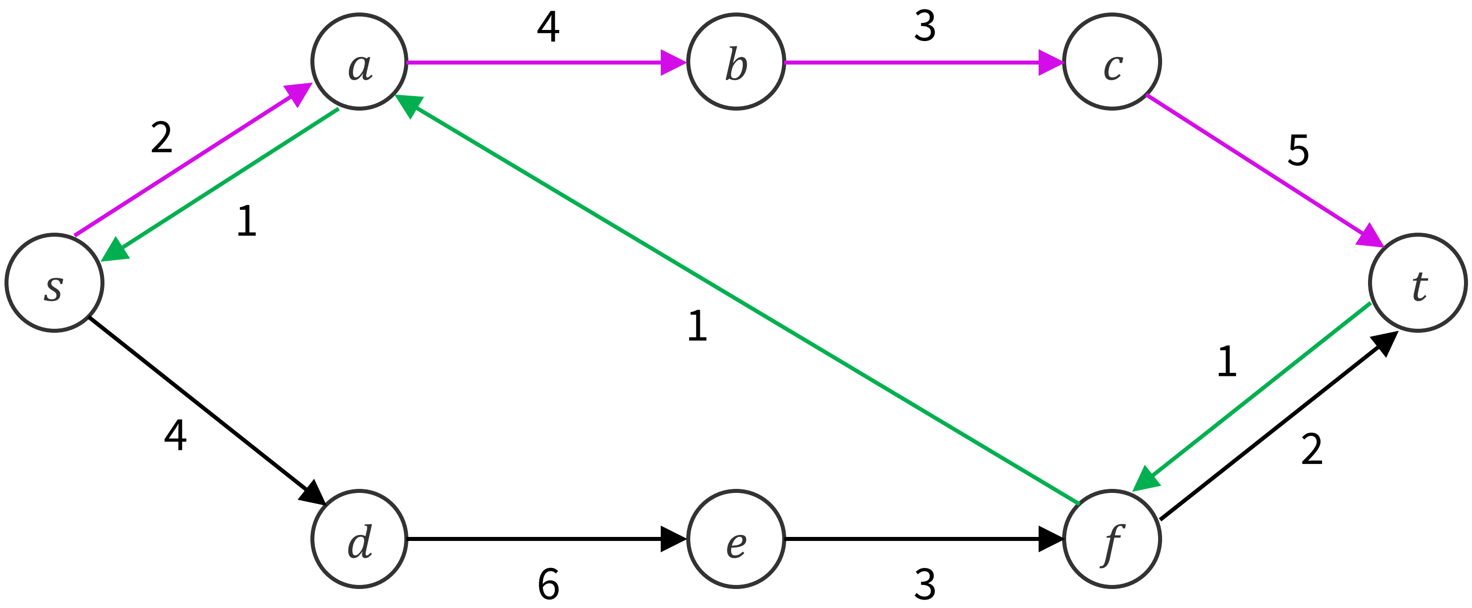
Problem 1 – Flow algorithms practice



Here is our current residual graph.

Solution

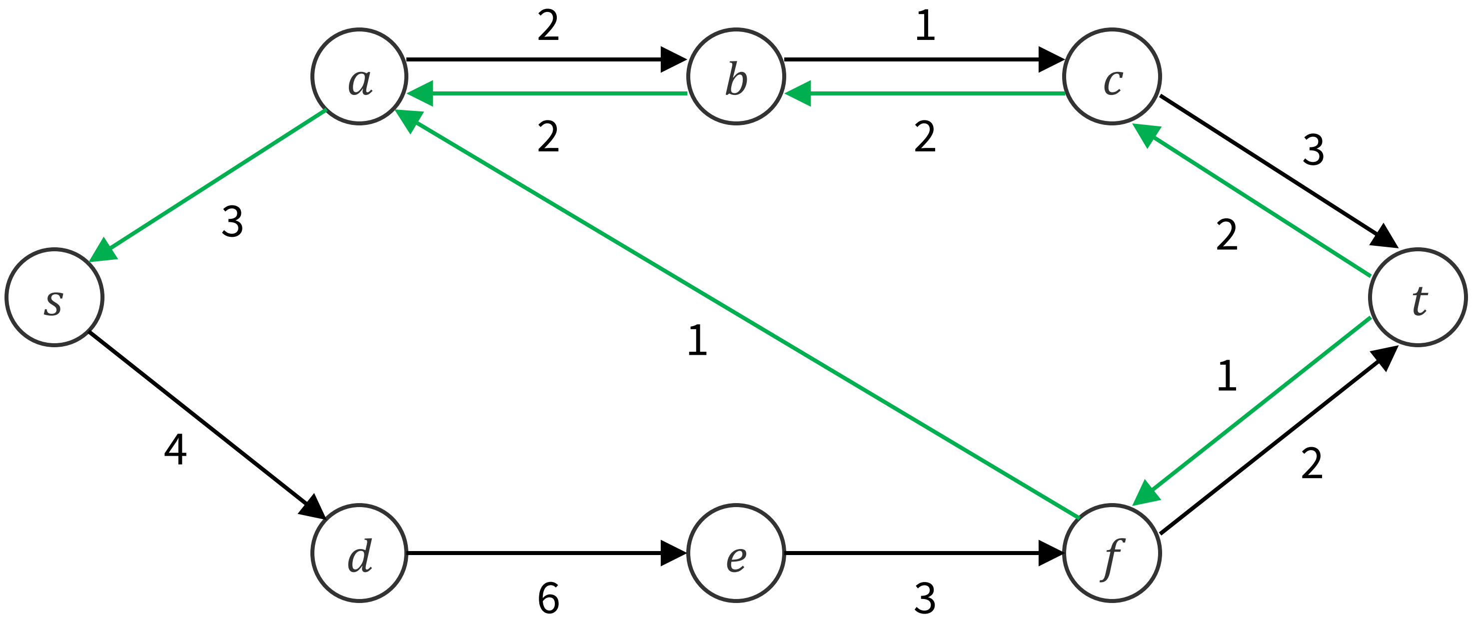
Problem 1 – Flow algorithms practice



Find the shortest s-t path.

Solution

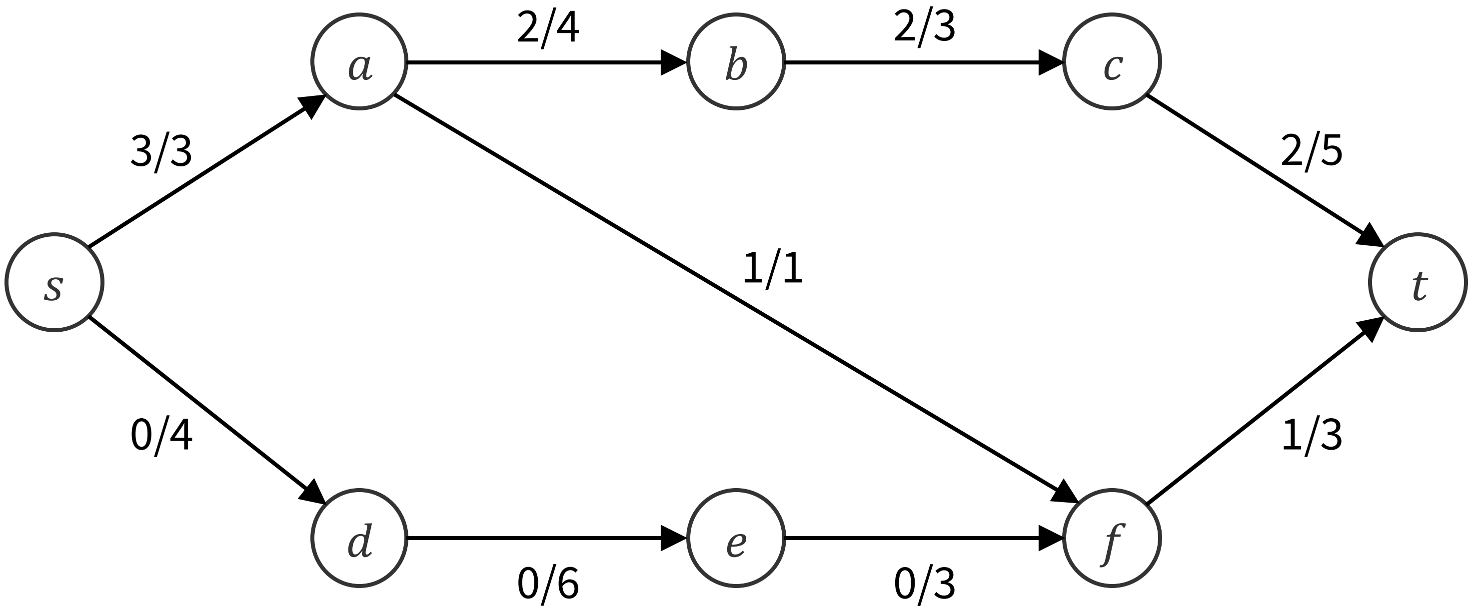
Problem 1 – Flow algorithms practice



Update the residual graph.

Solution

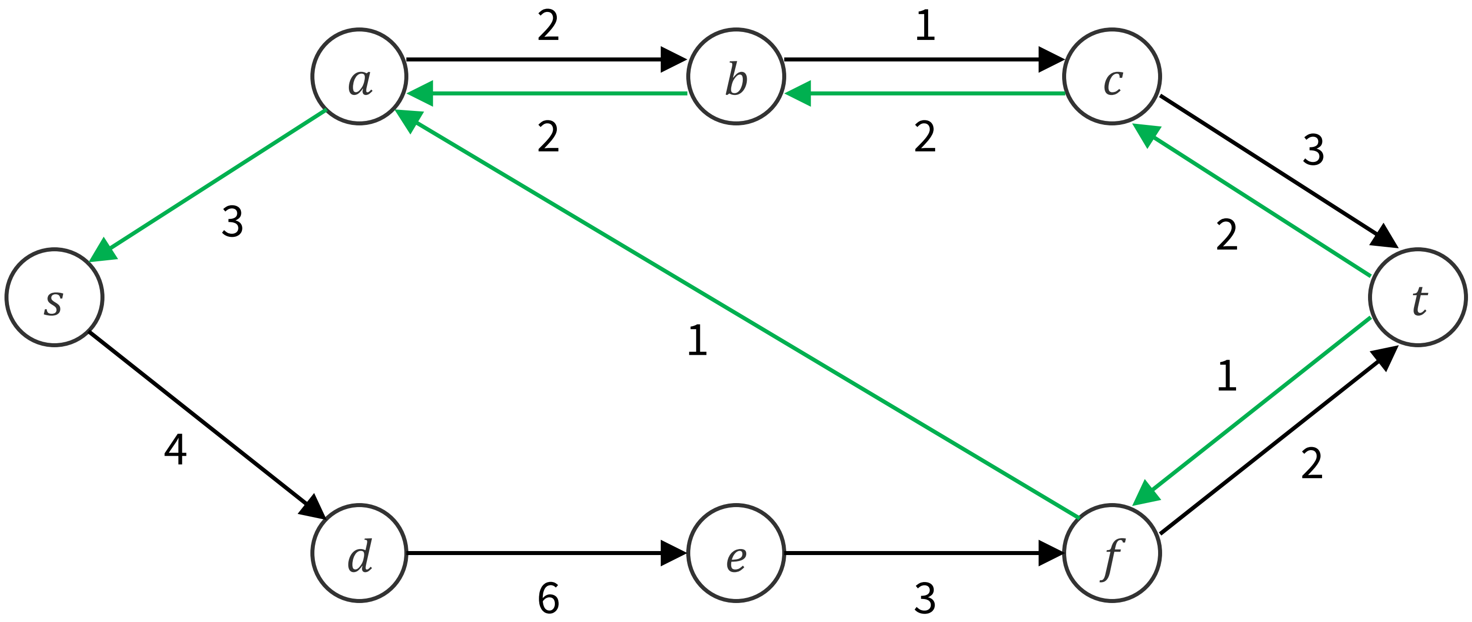
Problem 1 – Flow algorithms practice



Update the flow.

Solution

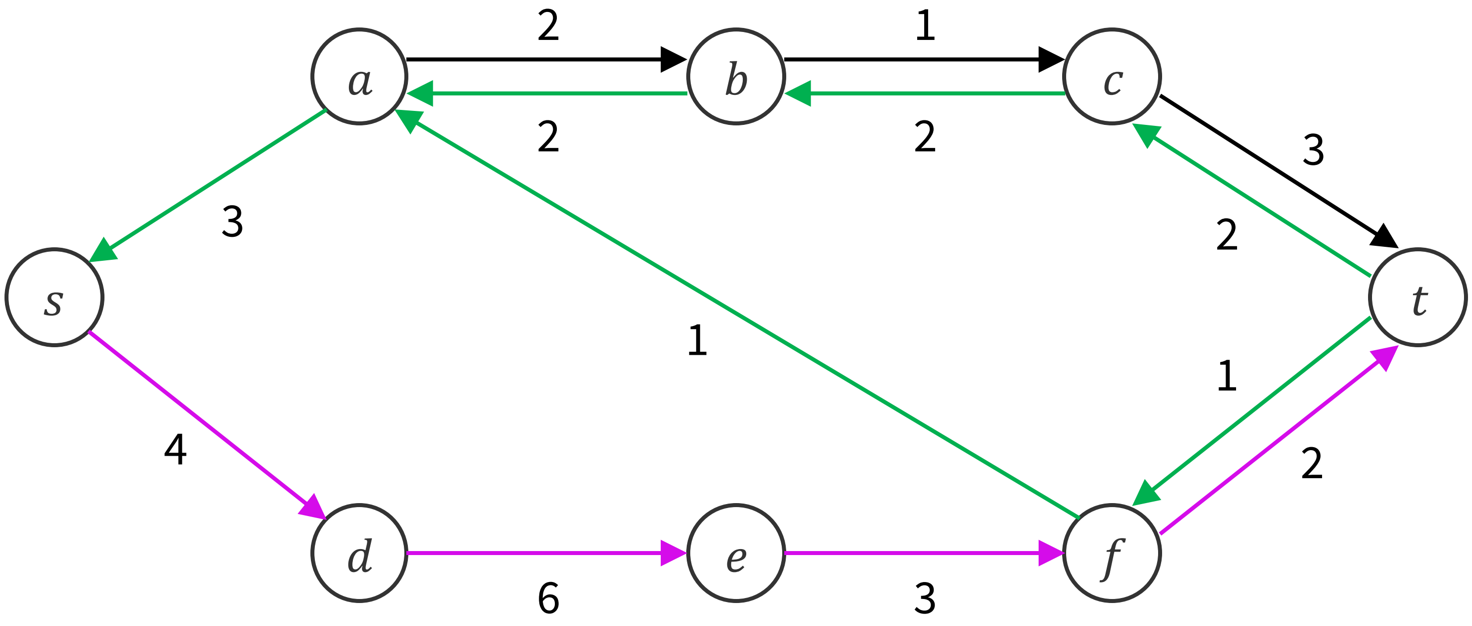
Problem 1 – Flow algorithms practice



Here is our current residual graph.

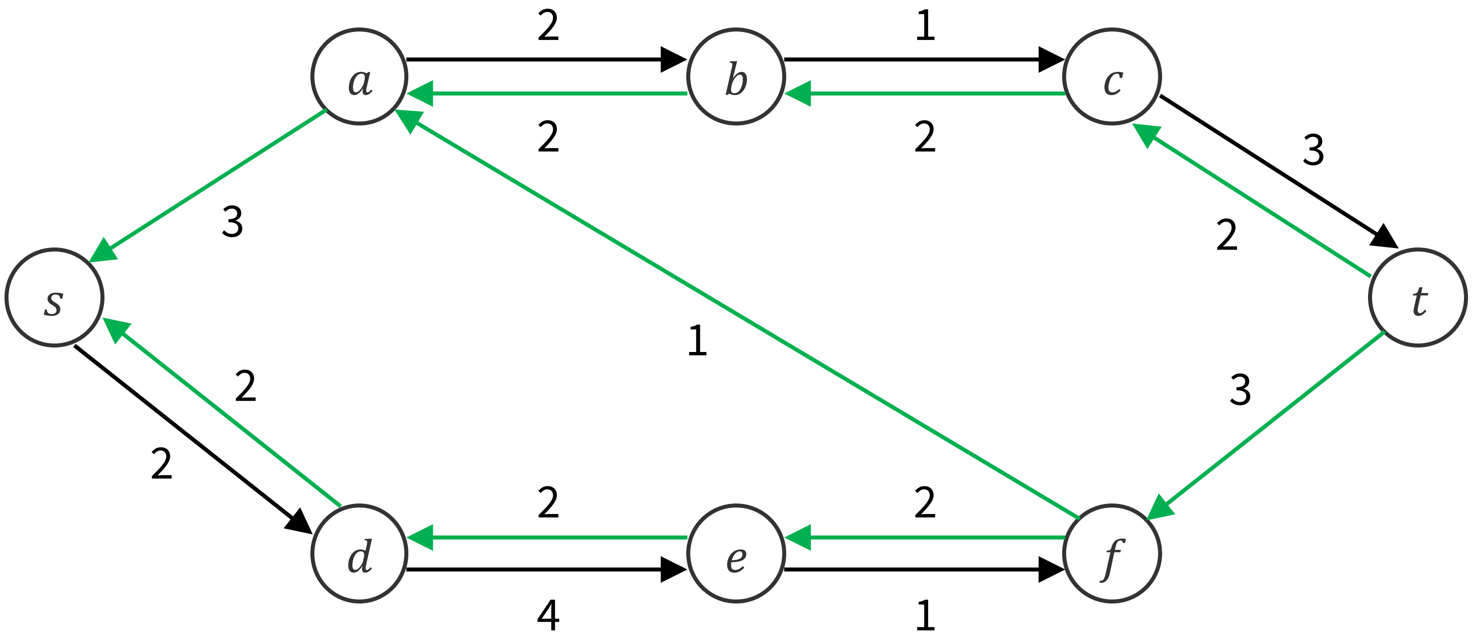
Solution

Problem 1 – Flow algorithms practice



Find the shortest s-t path.

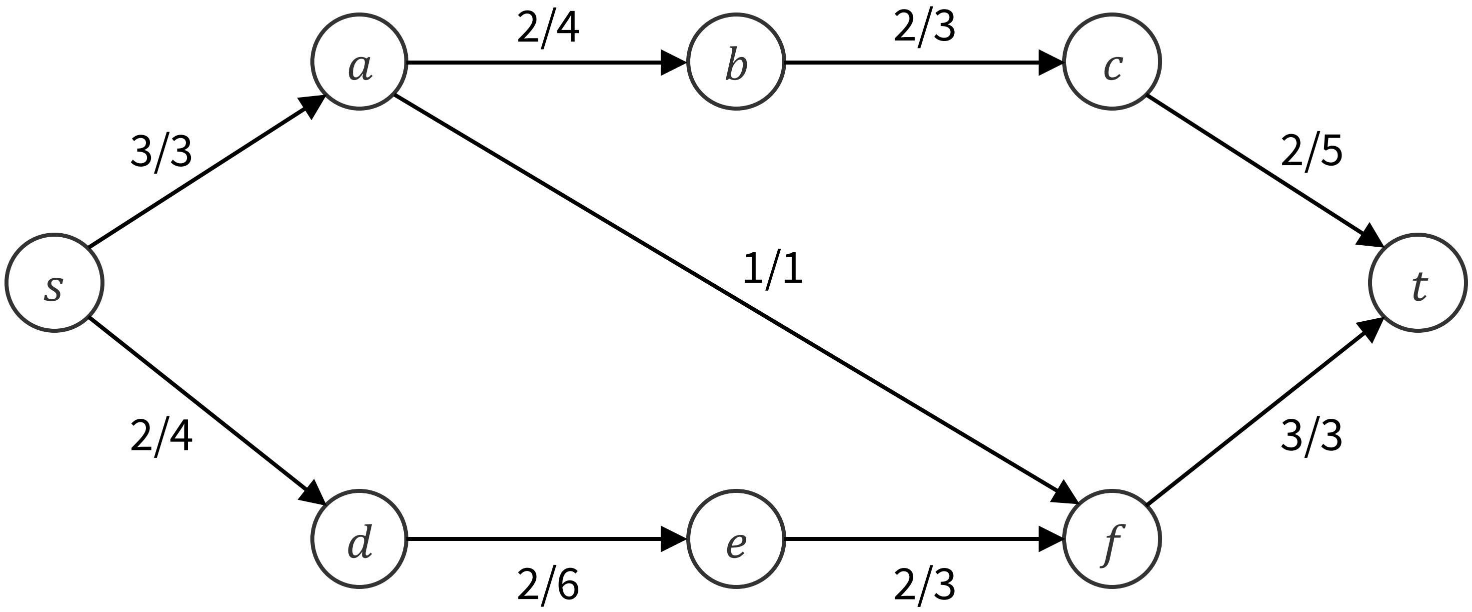
Problem 1 – Flow algorithms practice



Update the residual graph.

Solution

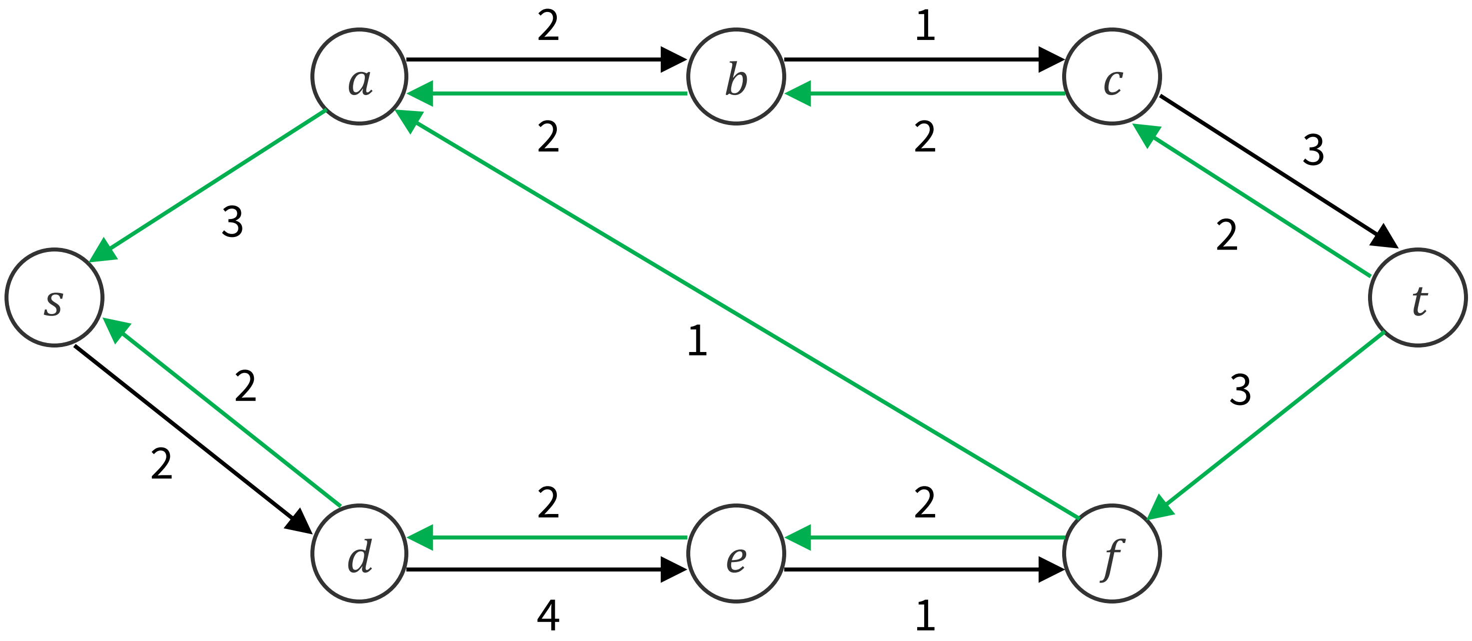
Problem 1 – Flow algorithms practice



Update the flow.

Solution

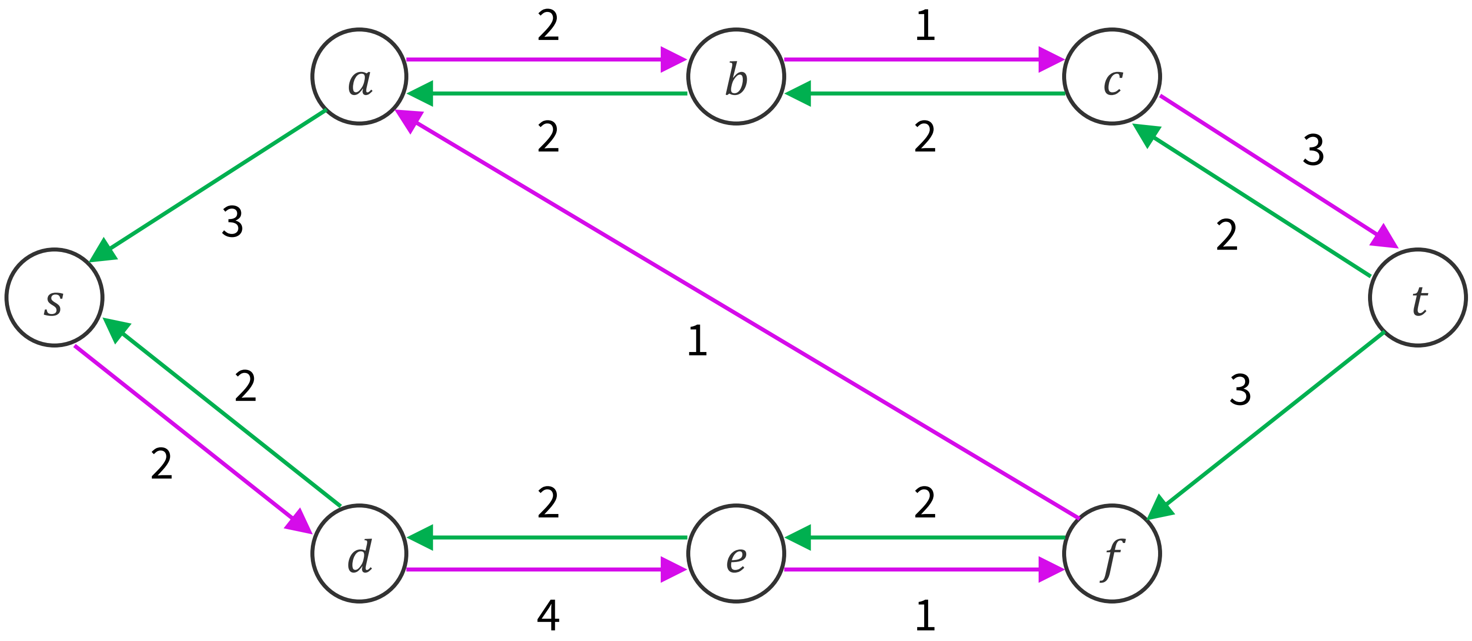
Problem 1 – Flow algorithms practice



Here is our current residual graph.

Solution

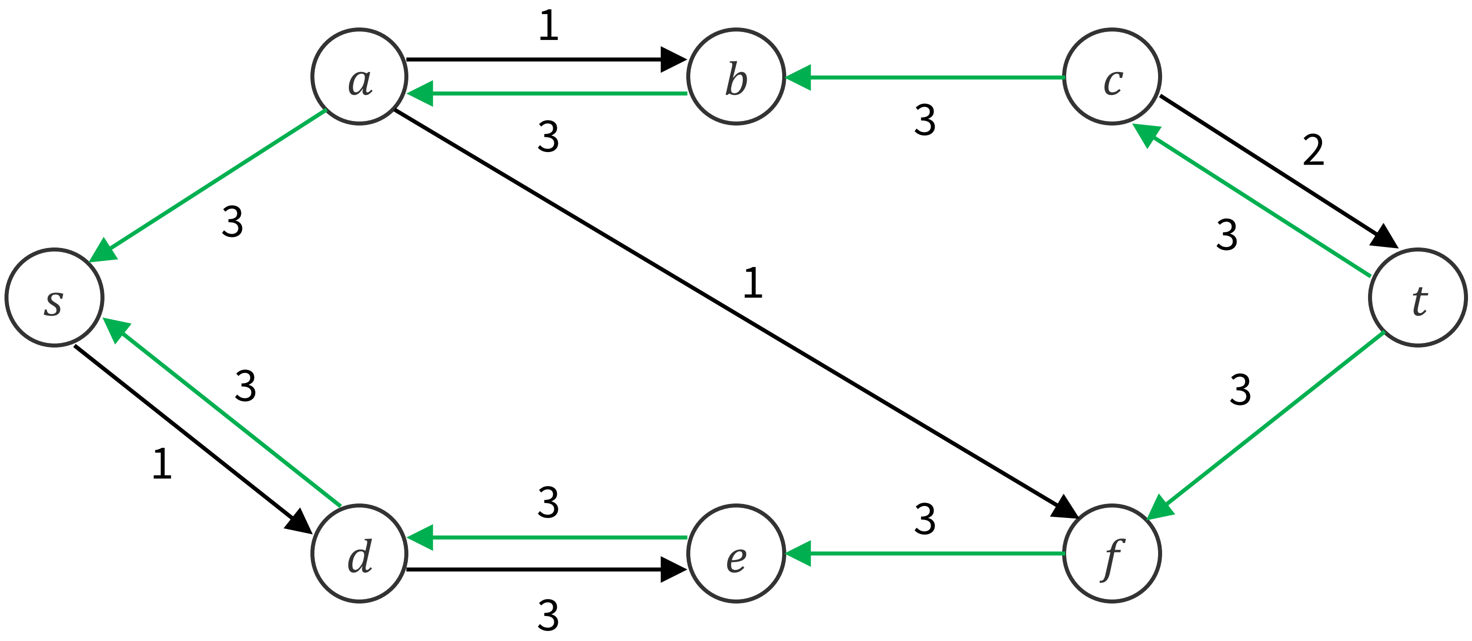
Problem 1 – Flow algorithms practice



Find the shortest s-t path.

Solution

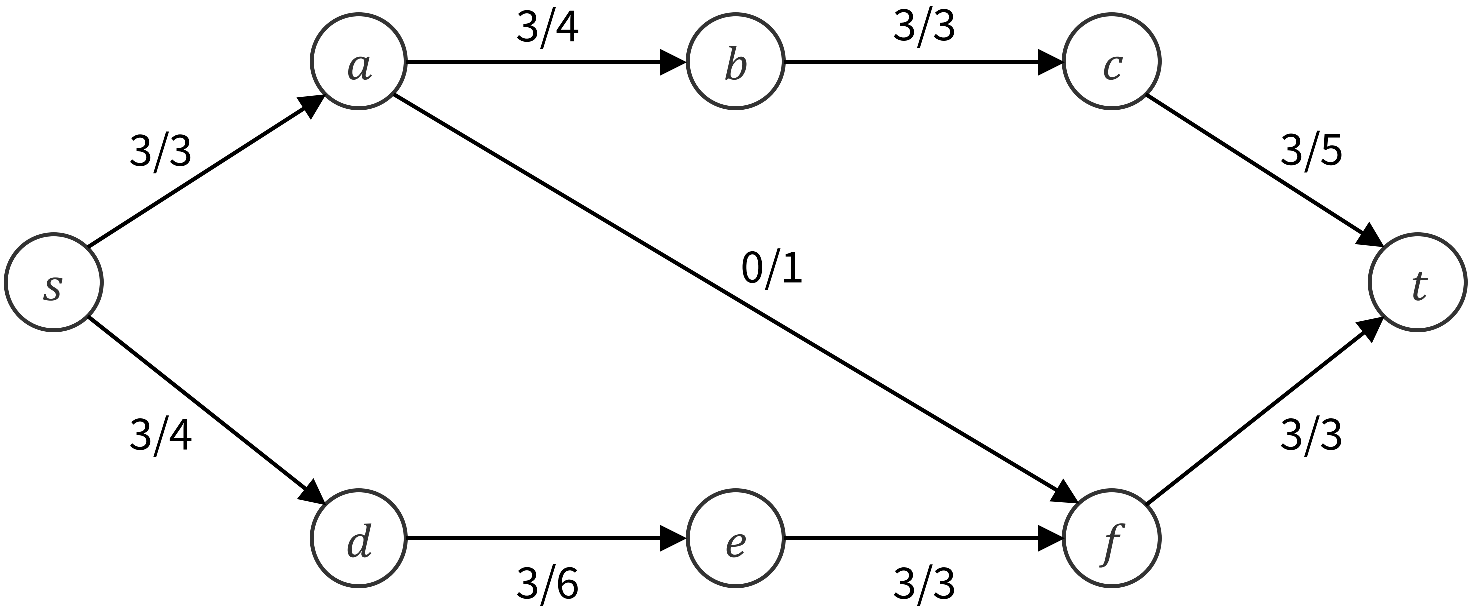
Problem 1 – Flow algorithms practice



Update the residual graph.

Solution

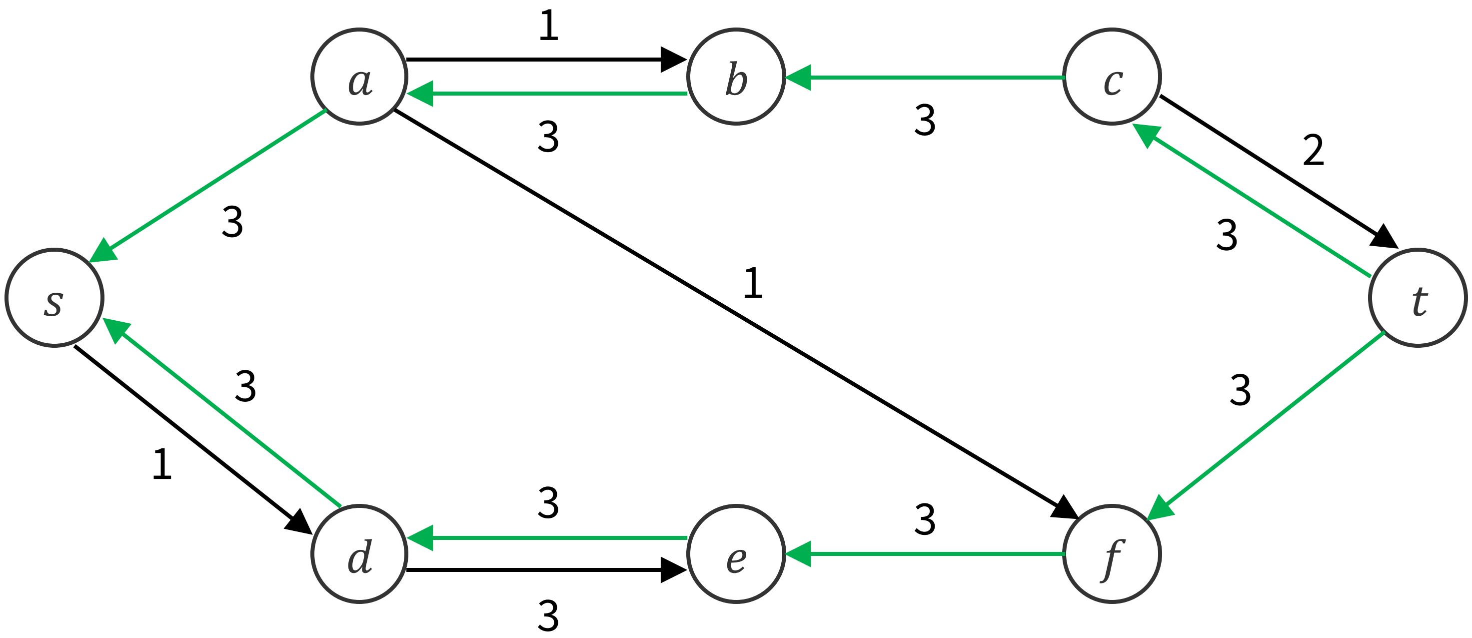
Problem 1 – Flow algorithms practice



Update the flow.

Solution

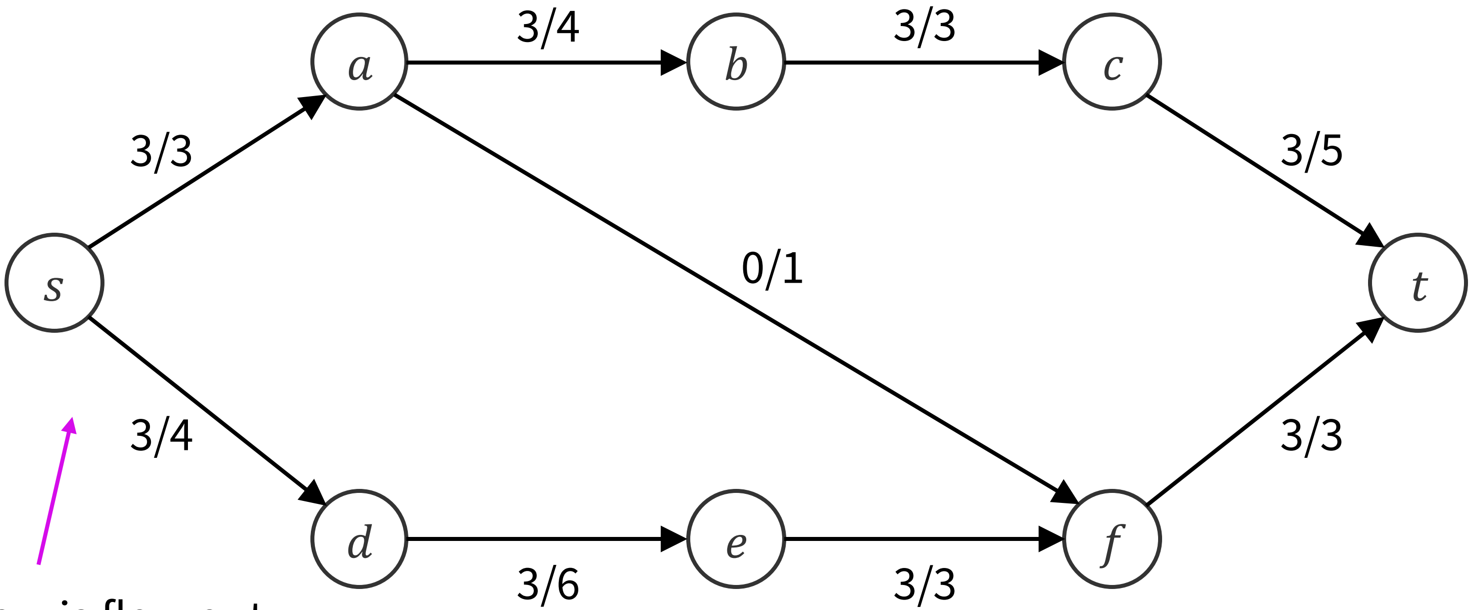
Problem 1 – Flow algorithms practice



There are no more s-t paths.

Solution

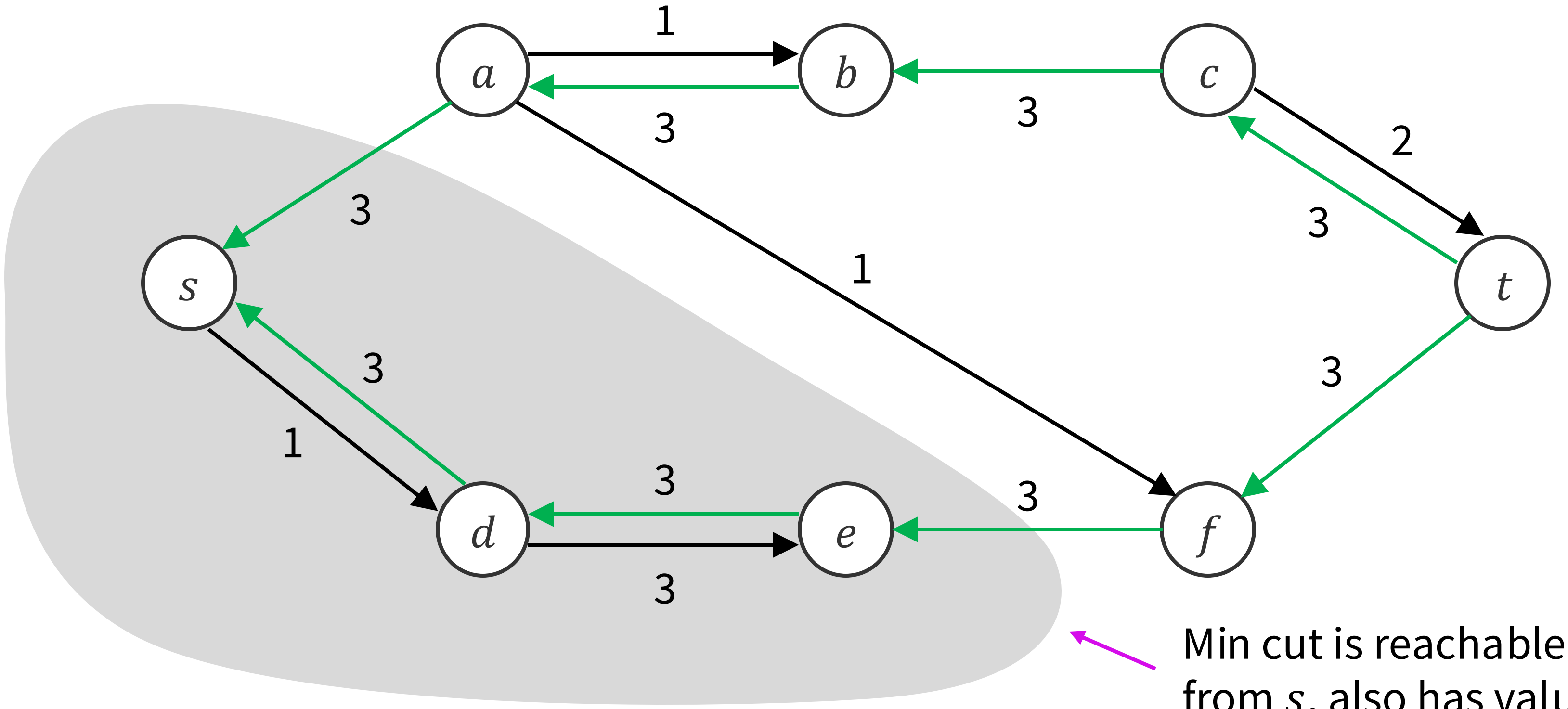
Problem 1 – Flow algorithms practice



Max flow is flow out of s, which is 6.

Solution

Problem 1 – Flow algorithms practice



Min cut is reachable set from s, also has value 6

Problem solving with flows

Problem solving strategy overview

Read and **summarize** the problem

↓

Decide to use **known algorithm** or **techniques from scratch**

↙ ↘

Preprocess the input into the form required by the technique

not covered this section

↓

Postprocess the technique's output into what you want

↘

Write pseudocode, proof, and running time analysis

Three common preprocessing tricks

To preprocess for network flows:

- If you want “multiple sources/sinks,” add **dummy vertices**.
- If you want “vertex capacities,” **split vertices** into two.
- If you want “unconstrained capacity,” just **set capacity to infinity**.

We’ll see examples of each.

Problem 2 – Reservoir balancing

You have a set of overfilled reservoirs $O = \{o_1, \ldots, o_m\}$ and a set of underfilled reservoirs $U = \{u_1, \ldots, u_n\}$, and want to move 10,000 gallons of water from reservoirs in O to reservoirs in U. You only care about the total amount of water moved, not each individual reservoir. You have a directed graph $G = (V, E)$ describing the one-way pipes connecting the reservoirs, where $O \subseteq V$ and $U \subseteq V$. This graph may include intermediate reservoirs, whose water levels should not change through your solution. Each pipe $e \in E$ has an integer maximum rate of flow $c(e)$ in gallons per minute. Find a method to move the water in the shortest amount of time.

a) Write a summary of the problem.

Work on this with the people around you, then we'll check!

Solution

Problem 2 – Reservoir balancing

a) Write a summary of the problem.

Input: Directed graph $G = (V, E)$ with maximum flow rates $c(e)$, sets $O, U \subseteq V$

Output: Time/method to push 10,000 gallons from O to U while respecting flow rates

Problem 2 – Reservoir balancing

a) Write a summary of the problem.

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

- If you want “multiple sources/sinks,” add dummy vertices.
- If you want “vertex capacities,” split vertices into two.
- If you want “unconstrained capacity,” just set capacity to infinity.

Previous Solution

Problem 2 – Reservoir balancing

a) Write a summary of the problem.

Input: Directed graph $G = (V, E)$ with maximum flow rates $c(e)$, sets $O, U \subseteq V$
Output: Time/method to push 10,000 gallons from O to U while respecting flow rates

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

- If you want "multiple sources/sinks," add dummy vertices.
- If you want "vertex capacities," split vertices into two.
- If you want "unconstrained capacity," just set capacity to infinity.

Work on this with the people around you, then we'll check!

Solution

Problem 2 – Reservoir balancing

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

Create dummy vertices s and t, as well as edges (s, o_i) and (u_i, t) for all $o_i \in O$ and $u_i \in U$. Give these new edges infinite capacity, and leave the rest of the graph alone.

Problem 2 – Reservoir balancing

c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

Work on this with the people around you, then we'll check!

Solution

Problem 2 – Reservoir balancing

c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

We get the maximum flow r from s to t as well as the flow $f: E' \to \mathbb{R}$ that achieves it, where E' is the edge set of our new graph.

Denote $f|_E$ the restriction of f to the original edge set E, and our solution will be to push water at rates according to $f|_E$ for $\frac{10{,}000}{r}$ minutes.

Network flows proofs

In a network flow problem, the main claim in your proof will probably be:

"The maximum flow in the graph that I constructed
is equal to the maximum solution in the original problem."

It should be intuitive, but to write thing formally, the strategy is to prove two things:

1. We can turn any **solution of our network** into a **solution of the original problem** of same quality.
2. We can turn any **solution of the original problem** into a **solution of our network** of same quality.

Then, your output works because (1), and no other solution of the original problem is better, by applying (2) and the fact that we found the max flow of our network.

Problem 2 – Reservoir balancing

d) Prove your solution is correct.

Work on this with the people around you, then we'll check!

Solution

Problem 2 – Reservoir balancing

d) Prove your solution is correct.

Turn **solution of our network** into a **solution of original problem** of same quality:

- Simply restrict to the original edge set (note that this is valid for original problem).
- By flow conservation, the flow out of s is equal to the sum of all flow leaving O, so the quality is the same.

Turn **solution of original problem** into a **solution of our network** of same quality:

- Set $f(s, o_i)$ to be the sum of all flow out of o_i (valid because infinite capacity).
- Then, the flow out of s is the sum of all flow leaving O, thus same quality.

Thus, the best solution of our network is equivalent to the best solution of the original.

Solution

Problem 2 – Reservoir balancing

d) Prove your solution is correct.

Lastly, the previous slide only showed that we compute the maximum water pumping rate from O to U, not yet the "best method to move water".

To finish, we just note that the best strategy for filling the reservoirs must have the form "run a fixed strategy with flow rate r for $\frac{10{,}000}{r}$ minutes" (no changes over time).

- Suppose a strategy changed over time. Then, we can improve it by taking the strategy at the point in time where flow rate is fastest, and use that the whole time.

Problem 2 – Reservoir balancing

e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Work on this with the people around you, then we'll check!

Solution

Problem 2 – Reservoir balancing

e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Because we have no control over how large the capacities are, **Ford–Fulkerson with BFS and the Edmonds–Karp bound** is best for this problem.

The graph we constructed has:

- $|V'| = n + 2$ vertices, and
- $|E'| = m + |O| + |U| < m + n \leq 2m$ edges.

Thus, the running time is $O(|V'||E'|^2) = O(nm^2)$.

Problem 3 – Traffic modeling

In most cities, traffic congestion happens only at intersections – segments without intersections are free-flowing. An extremely rough model is that the capacity of an intersection (the total number of vehicles per hour that flow through the intersection in any direction) is proportional to the number of traffic lanes at the intersection. You are given the road network of a city as a graph $G = (V, E)$ (consisting of directed edges, i.e. one-way streets), as well as the number of lanes $c(v)$ at each intersection. Suppose each lane adds a capacity of 300 vehicles per hour, and there are no intersections with more than 12 lanes. Given an origin s and destination t (which also do have limited capacity), compute how many vehicles per hour can move from s to t.

a) Write a summary of the problem.

Work on this with the people around you, then we'll check!

Solution

Problem 3 – Traffic modeling

a) Write a summary of the problem.

Input: Directed graph $G = (V, E)$ with vertex capacities $c(v) \leq 12$, and s and t

Output: Maximum flow rate from s to t

Problem 3 – Traffic modeling

a) Write a summary of the problem.

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

- If you want "multiple sources/sinks," add dummy vertices.
- If you want "vertex capacities," split vertices into two.
- If you want "unconstrained capacity," just set capacity to infinity.

Previous Solution

Problem 3 – Traffic modeling

a) Write a summary of the problem.

Input: Directed graph $G = (V, E)$ with vertex capacities $c(v) \leq 12$, and s and t
Output: Maximum flow rate from s to t

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

- If you want "multiple sources/sinks," add dummy vertices.
- If you want "vertex capacities," split vertices into two.
- If you want "unconstrained capacity," just set capacity to infinity.

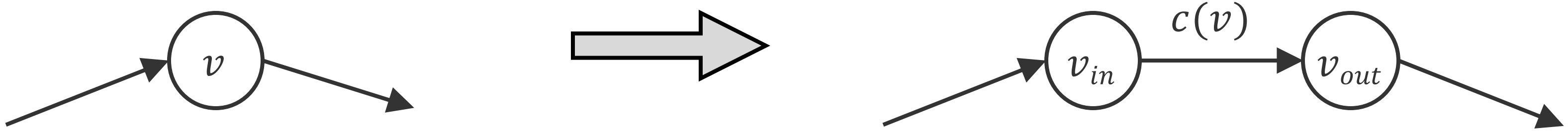
Work on this with the people around you, then we'll check!

Solution

Problem 3 – Traffic modeling

b) Think about the three tricks. Use them to preprocess our input into something suitable as input for a network flows algorithm.

First, put infinite capacity on all edges in the original graph.



Then, split every vertex v into v_{in} and v_{out}, and change all (u, v) into (u_{out}, v_{in}), (as well as (v, w) into (v_{out}, w_{in})). Put capacity $c(v)$ on the new edge (v_{in}, v_{out}).

Lastly, we will compute the flow between $s' = s_{in}$ and $t' = t_{out}$.

Problem 3 – Traffic modeling

c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

This should be quick – what can we do?

Solution

Problem 3 – Traffic modeling

c) After running a max flow algorithm, what do you get? What postprocessing is needed to get the solution?

We get the maximum total flow rate r, and we should return $300r$.

Problem 3 – Traffic modeling

d) Prove your solution is correct.

Work on this with the people around you, then we'll check!

Solution

Problem 3 – Traffic modeling

d) Prove your solution is correct.

Turn **solution of our network** into a **solution of original problem** of same quality:

- For every edge (u_{out}, v_{in}), give flow $f(u_{out}, v_{in})$ to original edge (u, v).
- By flow conservation at v_{in} and $c(v_{in}, v_{out}) = c(v)$, the original vertex v has at most $c(v)$ flow through it, so it is a valid solution.
- By flow conservation, flow out of s_{in} is the same as flow out of s_{out}, which by construction is the flow out of s, so the quality is the same.

Solution

Problem 3 – Traffic modeling

d) Prove your solution is correct.

Turn **solution of original problem** into a **solution of our network** of same quality:

- Set $f(u_{out}, v_{in})$ to be the traffic flow from u to v, and set $f(v_{in}, v_{out})$ as necessary to maintain flow conservation.
- Edges of type (u_{out}, v_{in}) have infinite capacity, so they are fine, and edges of type (v_{in}, v_{out}) have capacity $c(v)$, so they are fine by vertex capacities.
- The quality is the same, since by construction we set the the flow out of s_{in} to be equal to the flow out of s_{out}, which was the flow out of s.

Problem 3 – Traffic modeling

e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Work on this with the people around you, then we'll check!

Solution

Problem 3 – Traffic modeling

e) Which flow algorithm is best for this problem? Then analyze your running time. (Assume $|V| = n$, $|E| = m$, and $n \leq m$.)

Because all (non-infinite) capacities are at most 12, **Ford–Fulkerson with BFS and the original Ford–Fulkerson analysis** is best for this problem.

The graph we constructed has:

- $|V'| = 2n$ vertices, and
- $|E'| = m + n \leq 2m$ edges.

Since $|V'||E'|C = 48nm$, the running time is $O(|V'||E'|C) = O(nm)$.

Summary

- **Three tricks:** dummy s/t, split vertices for vertex capacity, and infinite capacity
- When using Ford–Fulkerson with BFS, pick **original FF bound if small capacities**, and pick **Edmonds–Karp bound if large capacities**.
- **Proof by converting solutions both ways**: between solutions to your constructed flow network and solutions to the original problem.

Thanks for coming to section this week!