

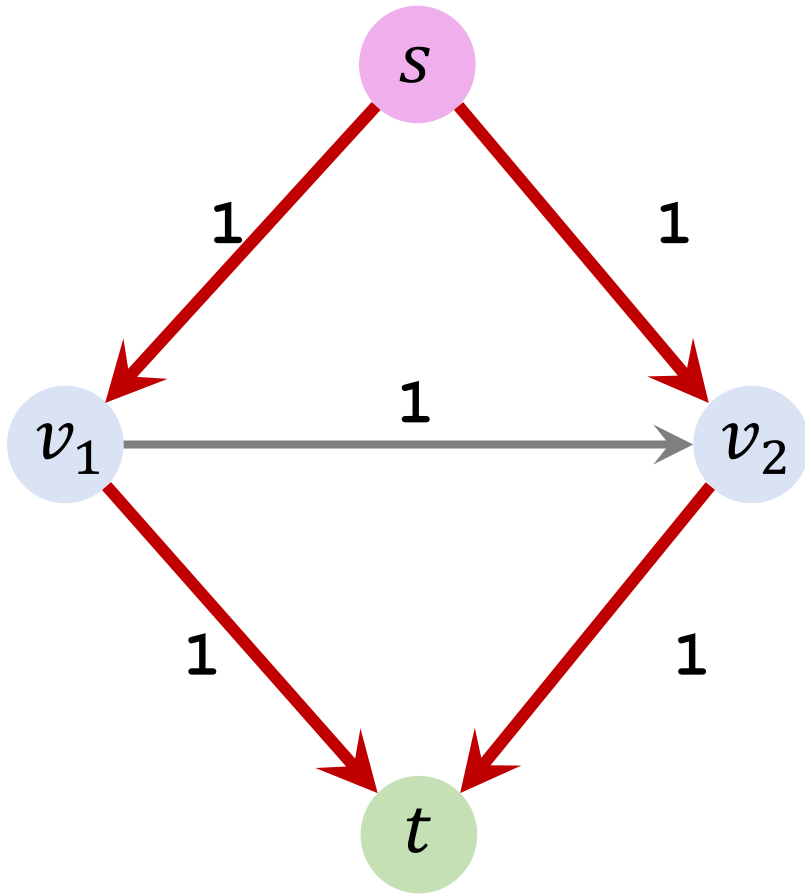
Ford-Fulkerson Algorithm

Shusen Wang

<http://wangshusen.github.io/>

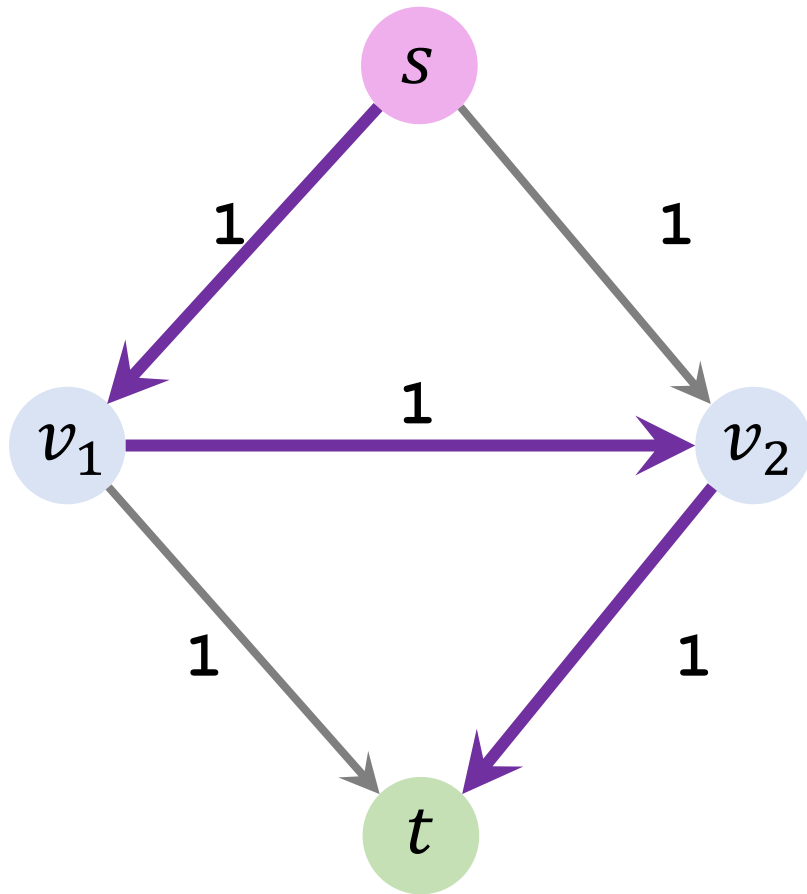
Problem with the naïve algorithm

- The amount of **max flow** is 2.



Maximum Flow

Problem with the naïve algorithm



Not Maximum Flow

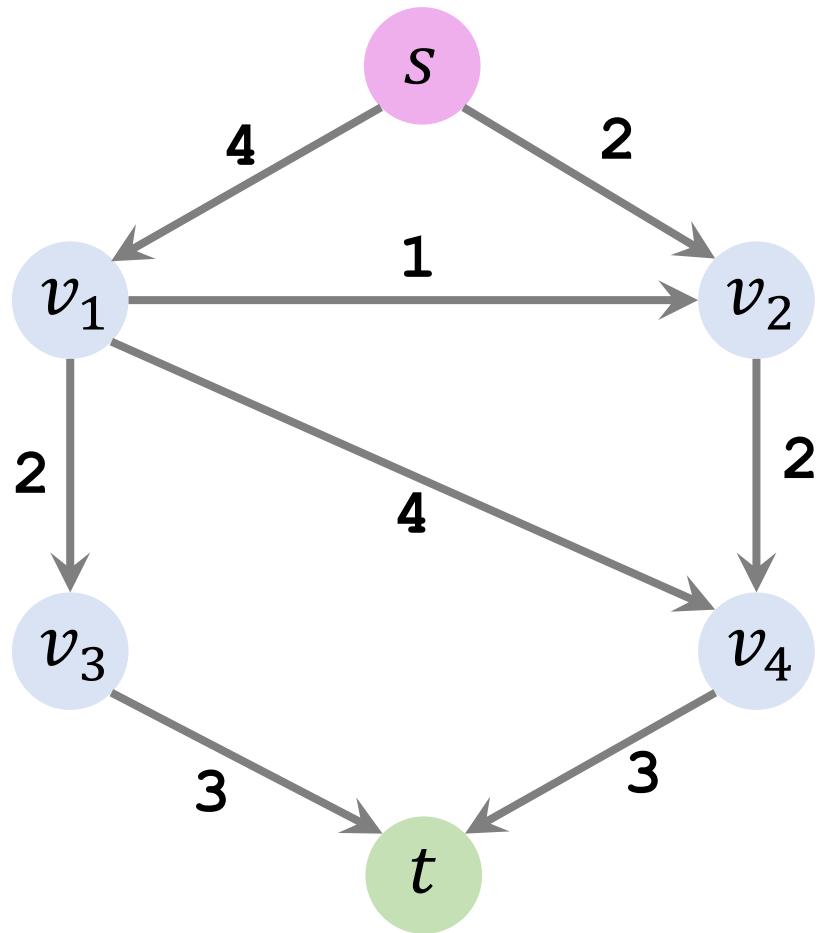
- The amount of **max flow** is 2.
- The amount of **blocking flow** is 1.
- Once a bad path is selected, the naïve algorithm cannot make corrections.

Ford-Fulkerson Algorithm

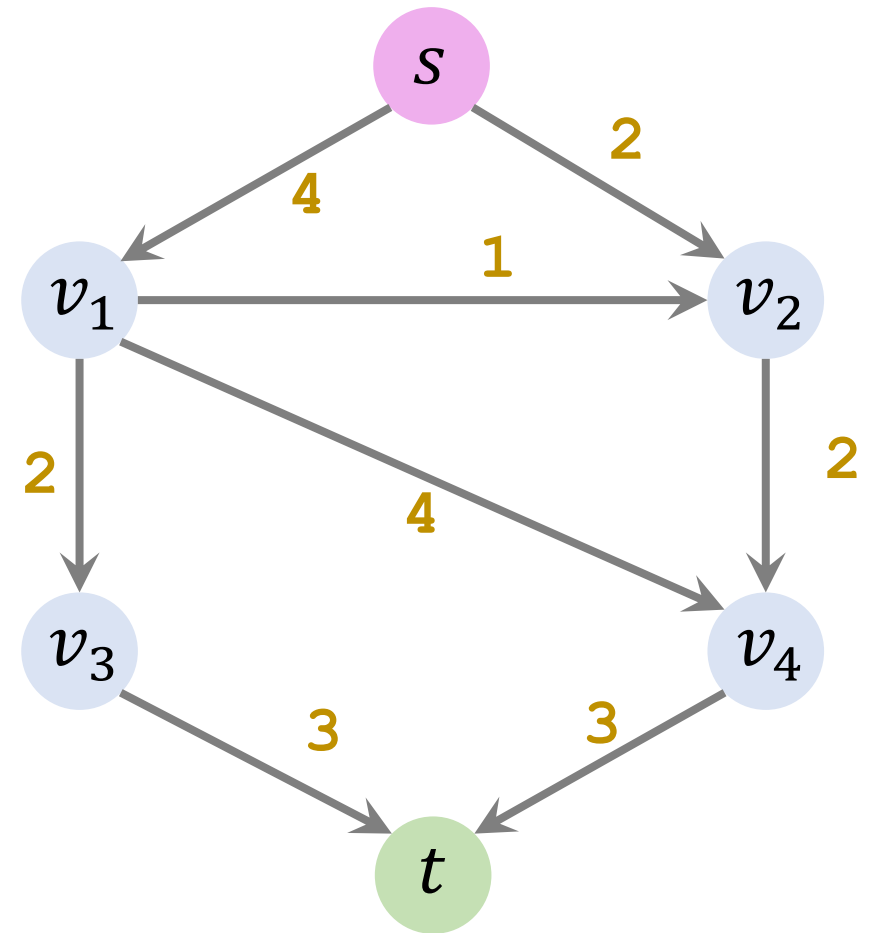
Reference

- L. R. Ford and D. R. Fulkerson. [Maximal flow through a network](#). *Canadian Journal of Mathematics*, 8: 399–404, 1956.

Initialization

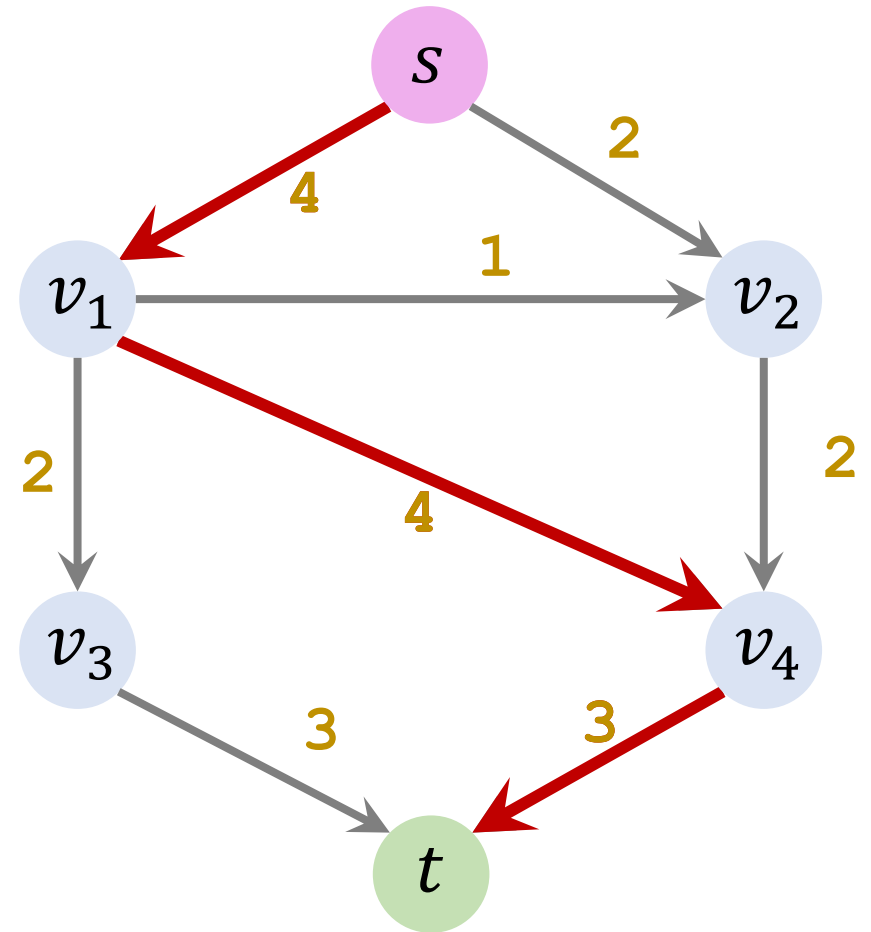


Original Graph



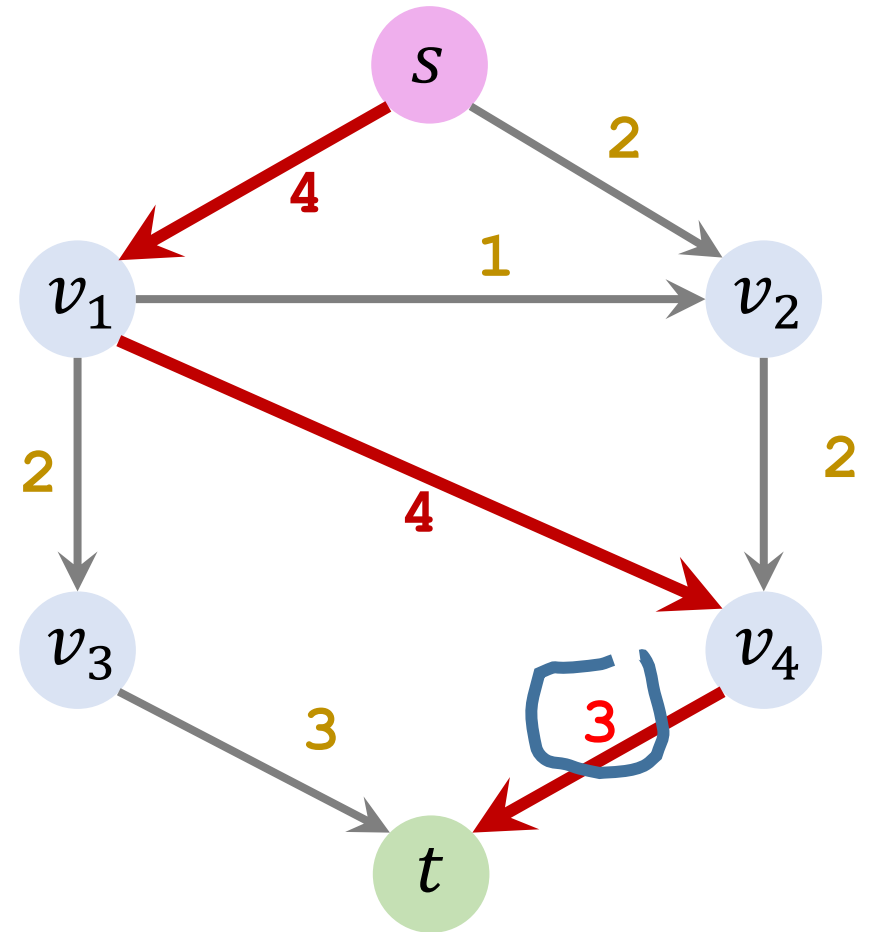
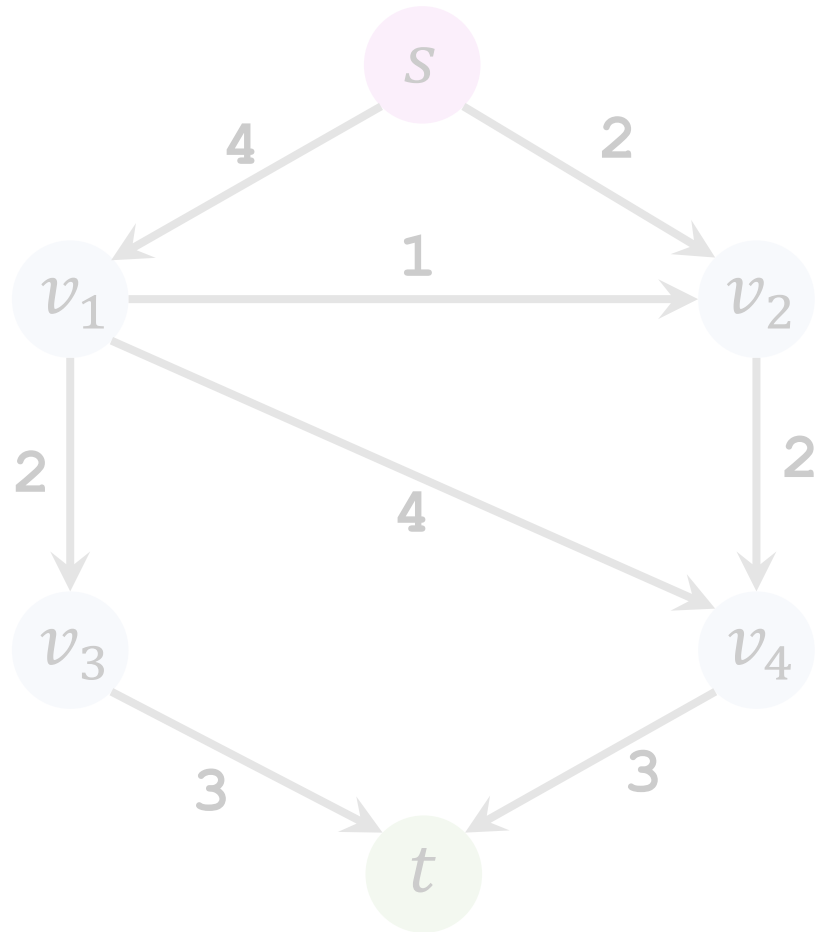
Residual Graph

Iteration 1: Find an augmenting path



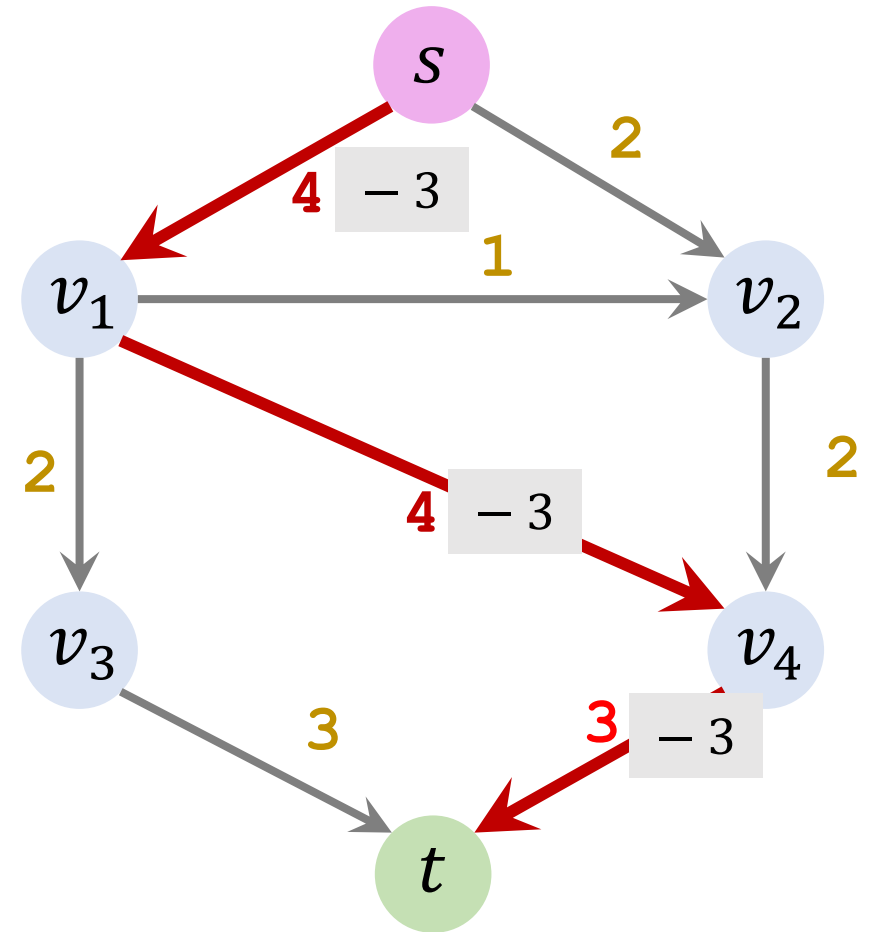
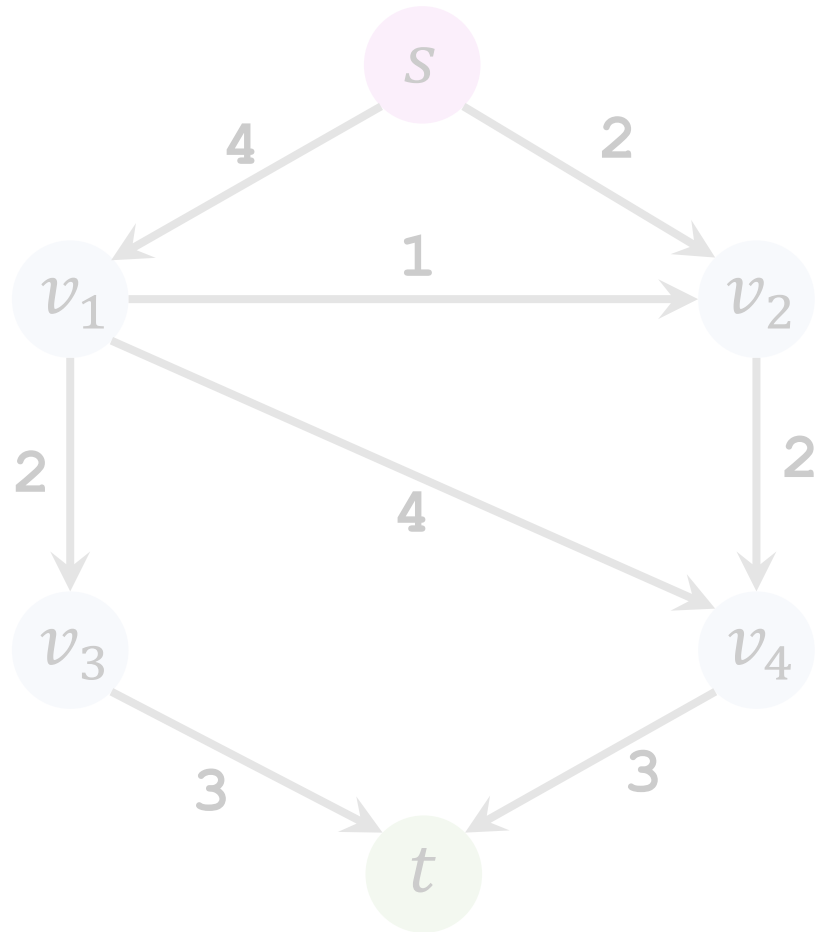
Found path $s \rightarrow v_1 \rightarrow v_4 \rightarrow t$.

Iteration 1: Find an augmenting path

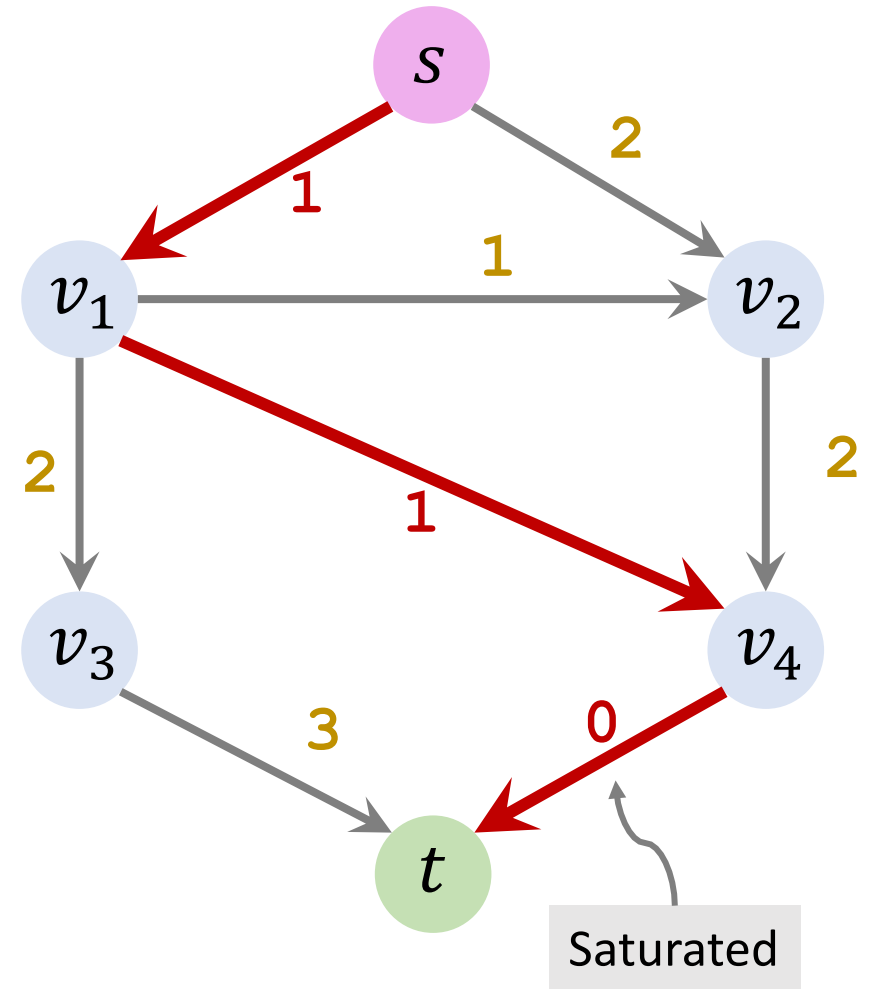
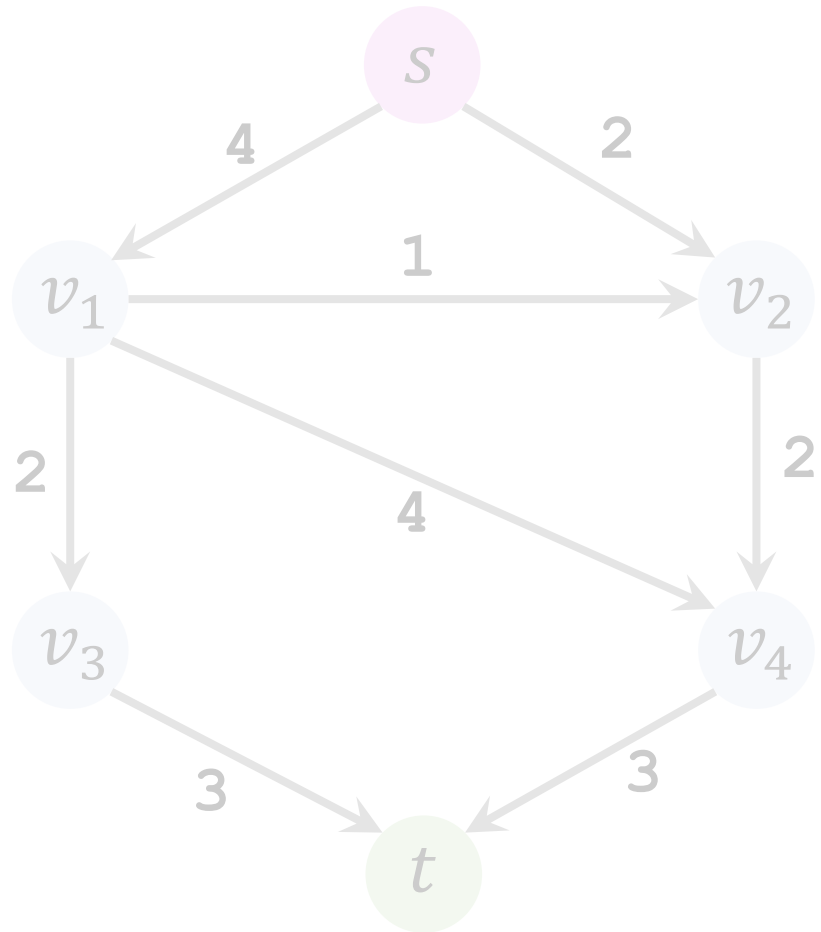


Found path $s \rightarrow v_1 \rightarrow v_4 \rightarrow t$. (Bottleneck capacity = 3.)

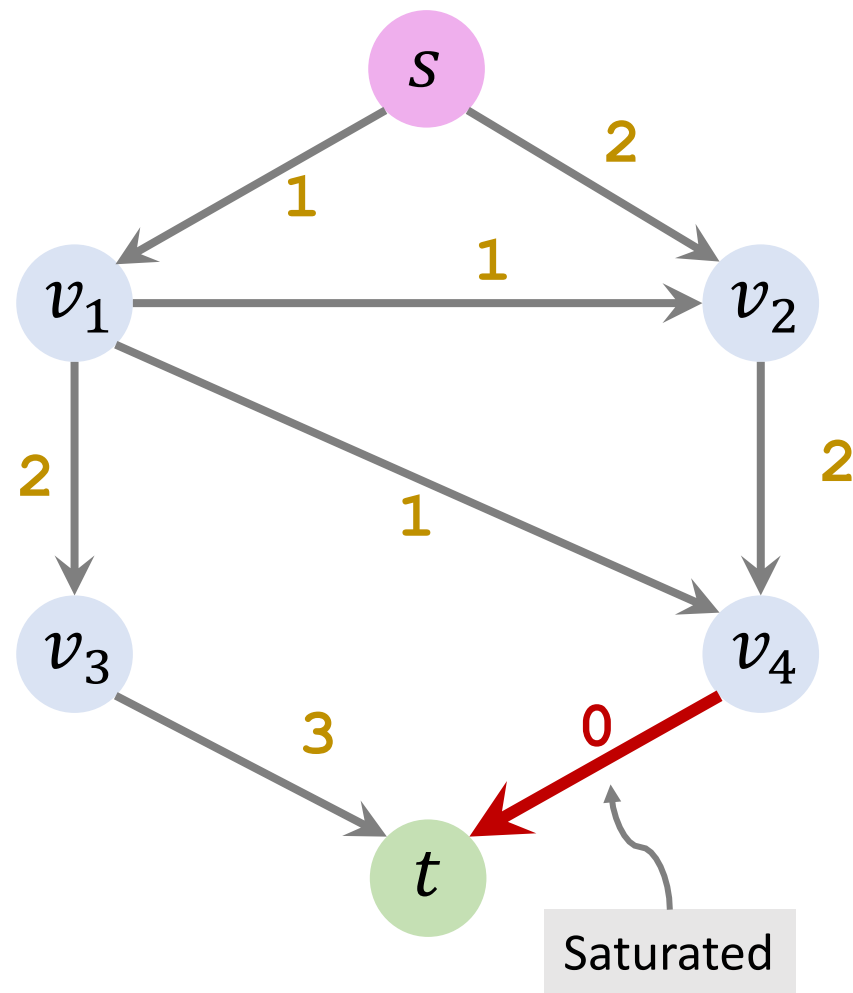
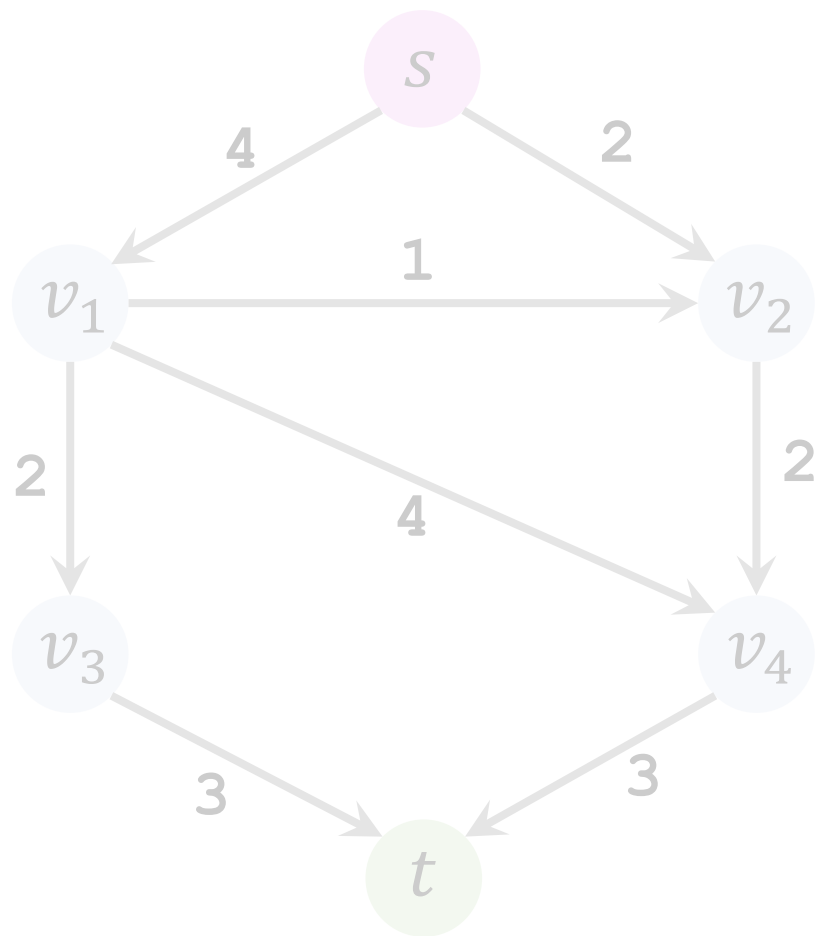
Iteration 1: Update residuals



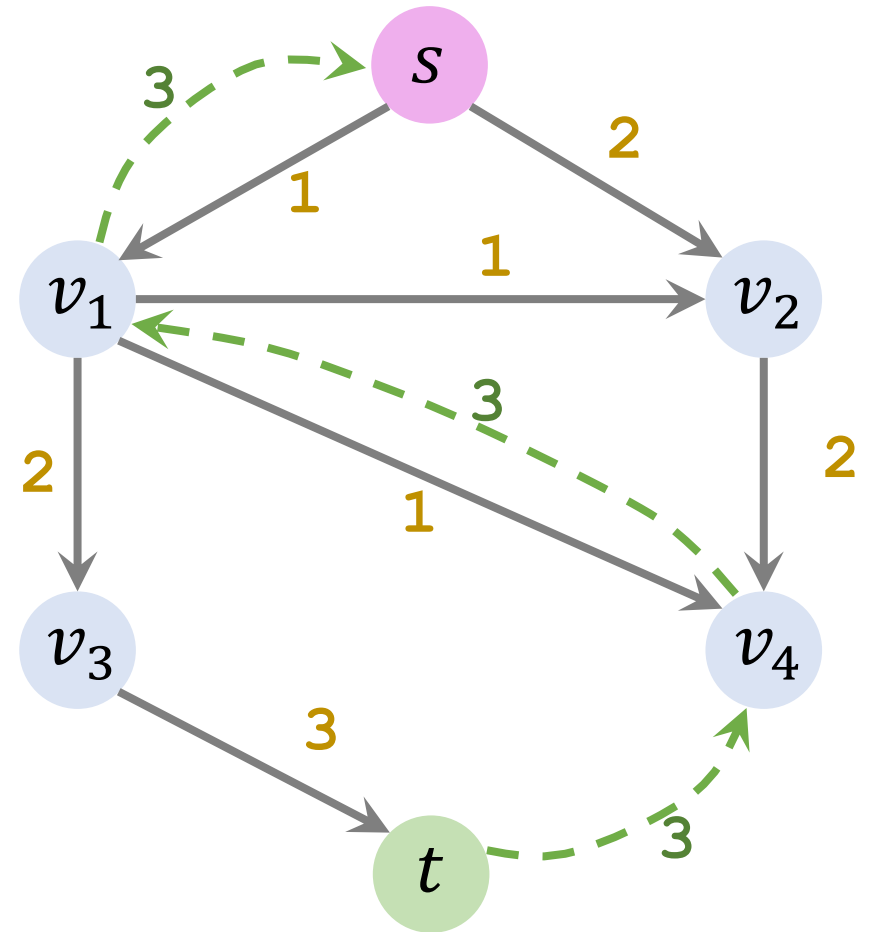
Iteration 1: Update residuals



Iteration 1: Remove saturated edges

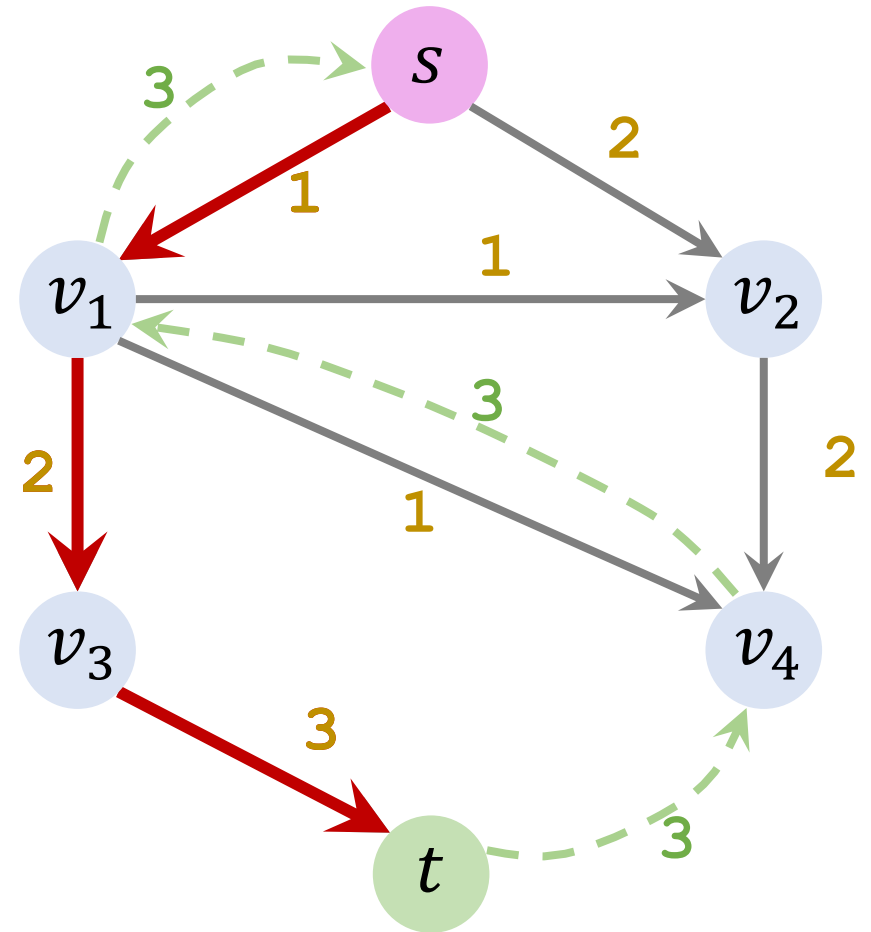


Iteration 1: Add a backward path



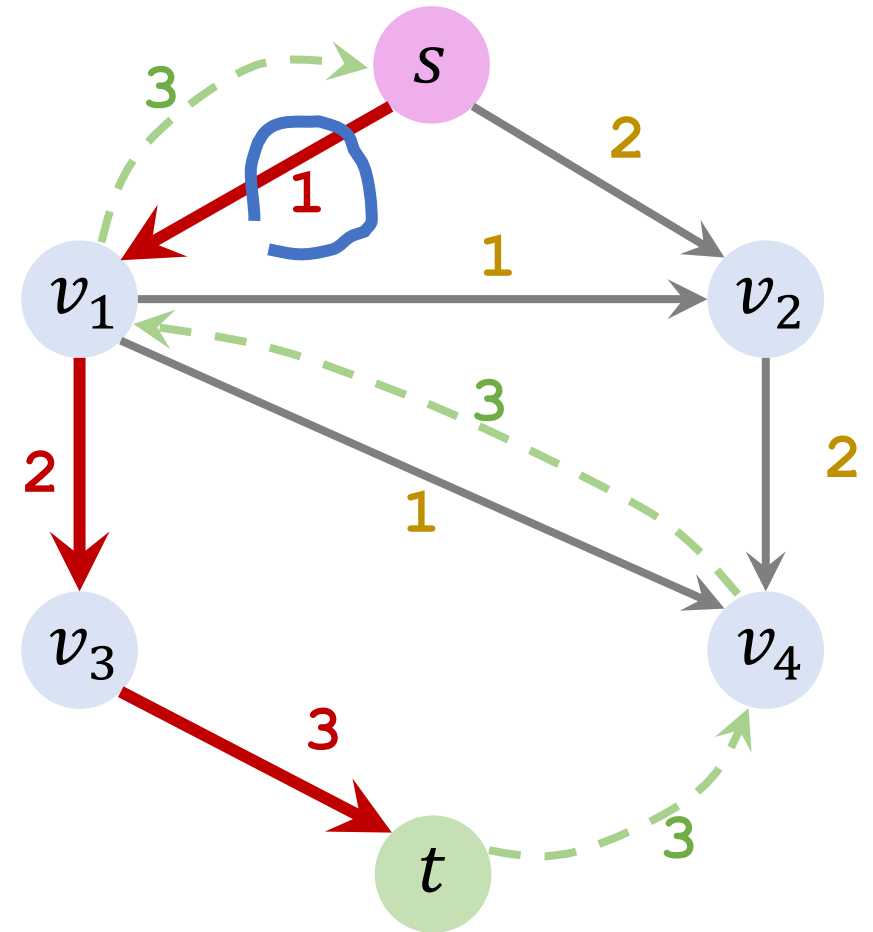
Add path $t \rightarrow v_4 \rightarrow v_1 \rightarrow s$ with capacity = 3. (Allow “undoing”.)

Iteration 2: Find an augmenting path



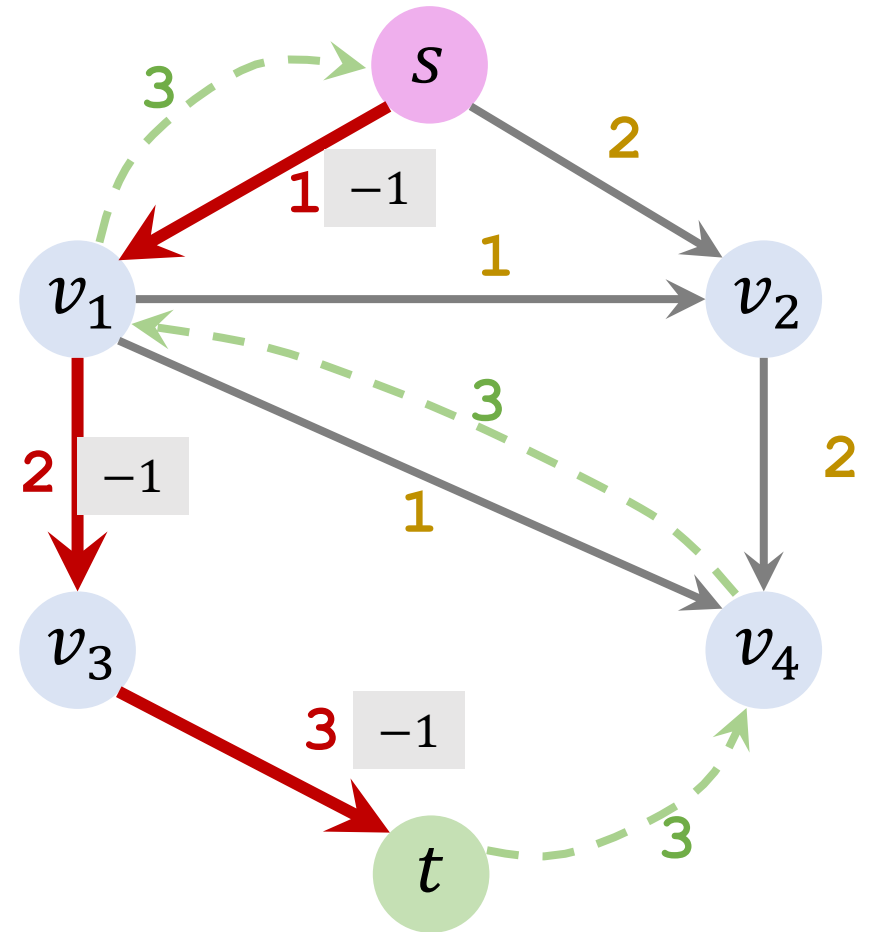
Found path $s \rightarrow v_1 \rightarrow v_3 \rightarrow t$.

Iteration 2: Find an augmenting path

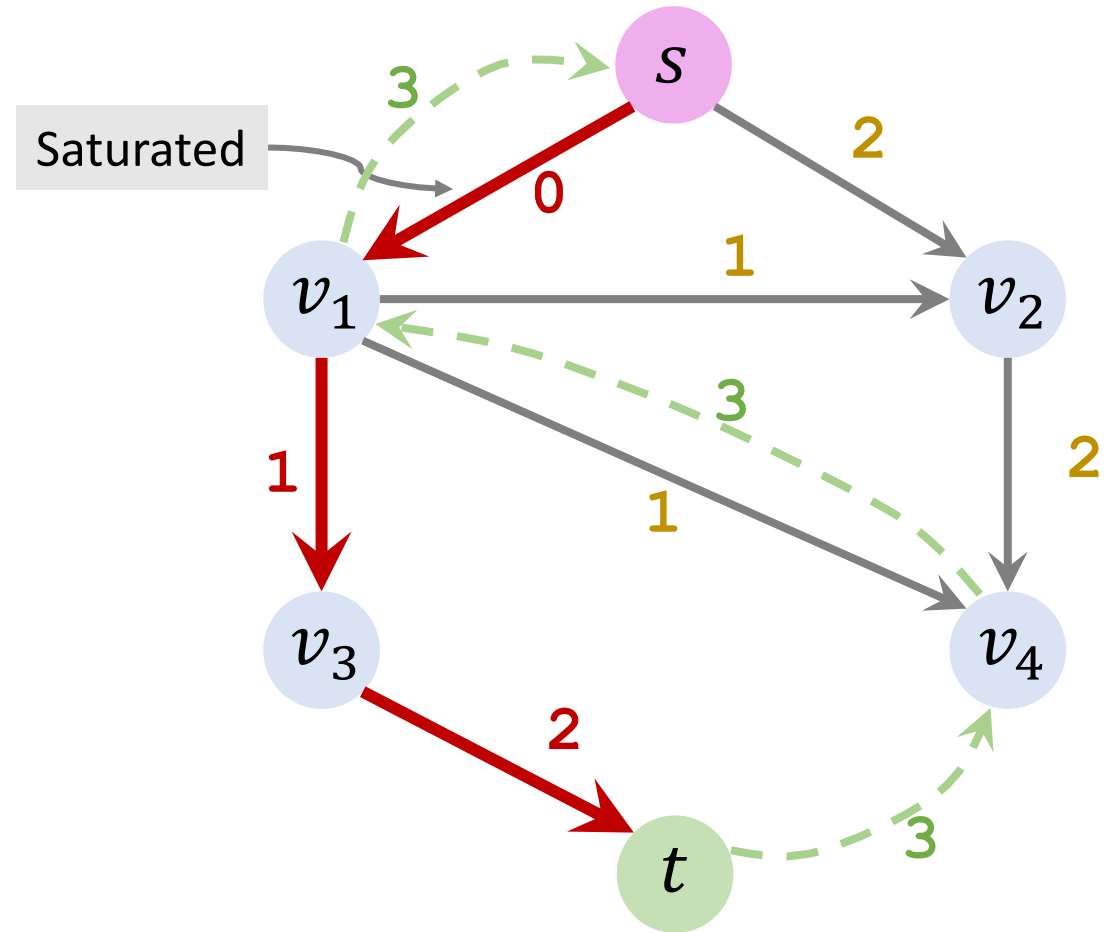
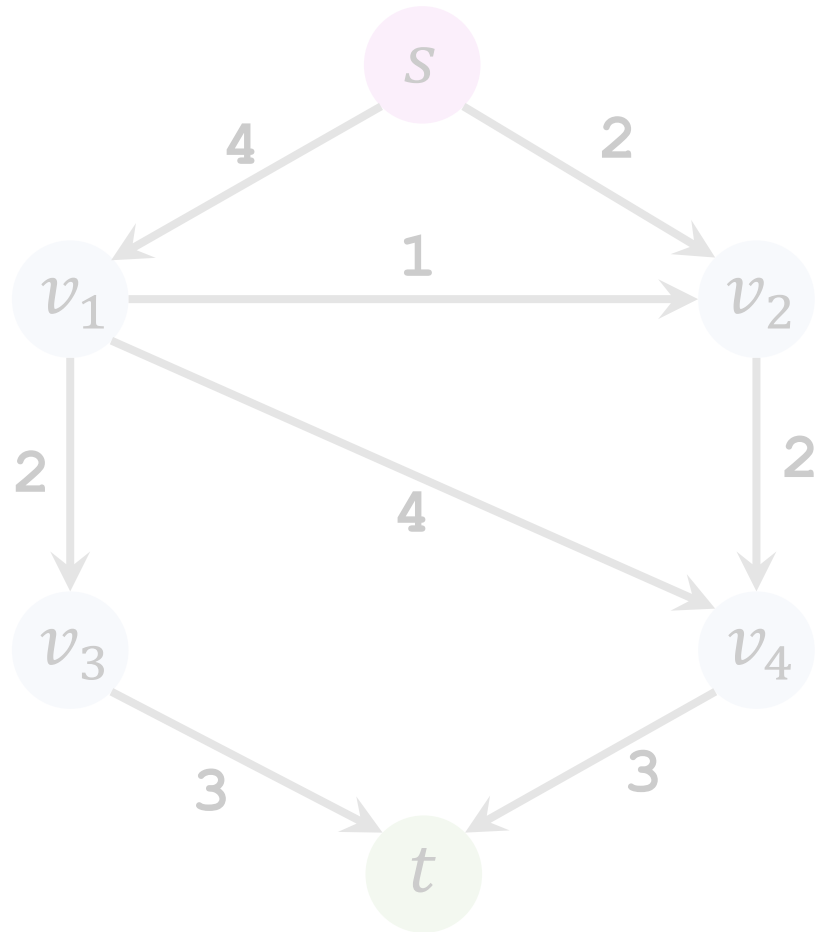


Found path $s \rightarrow v_1 \rightarrow v_3 \rightarrow t$. (Bottleneck capacity = 1.)

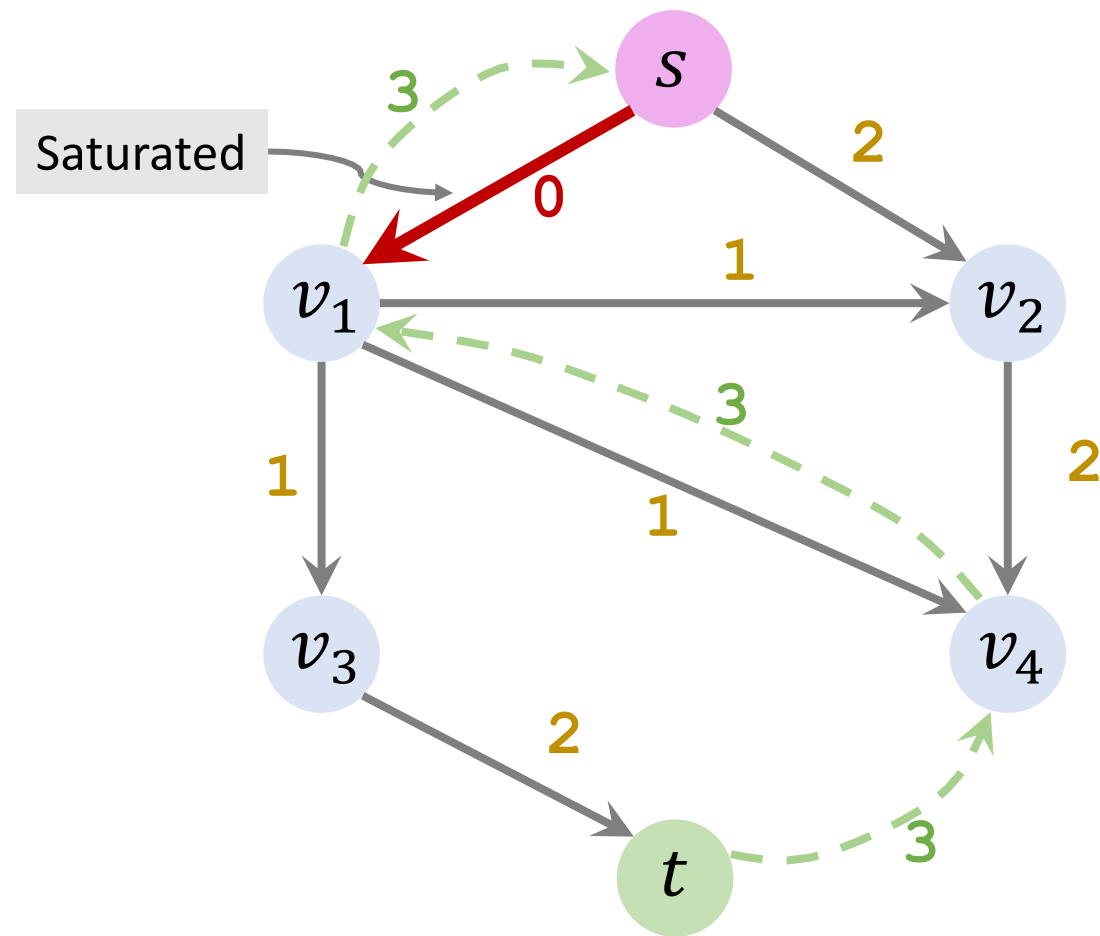
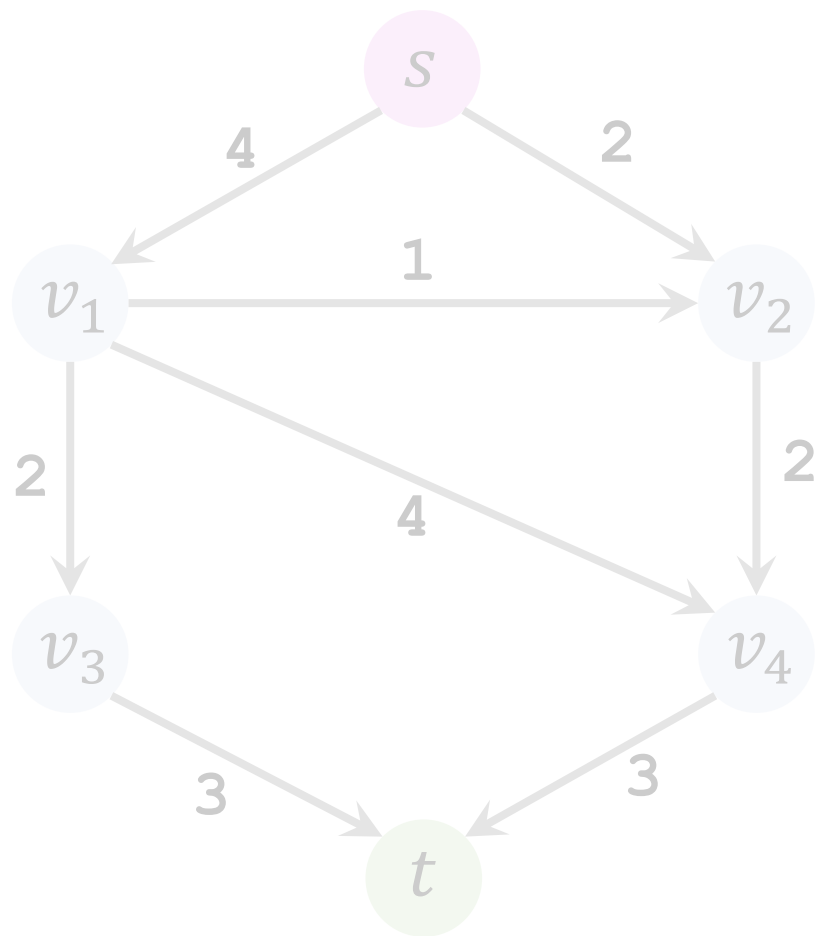
Iteration 2: Update residuals



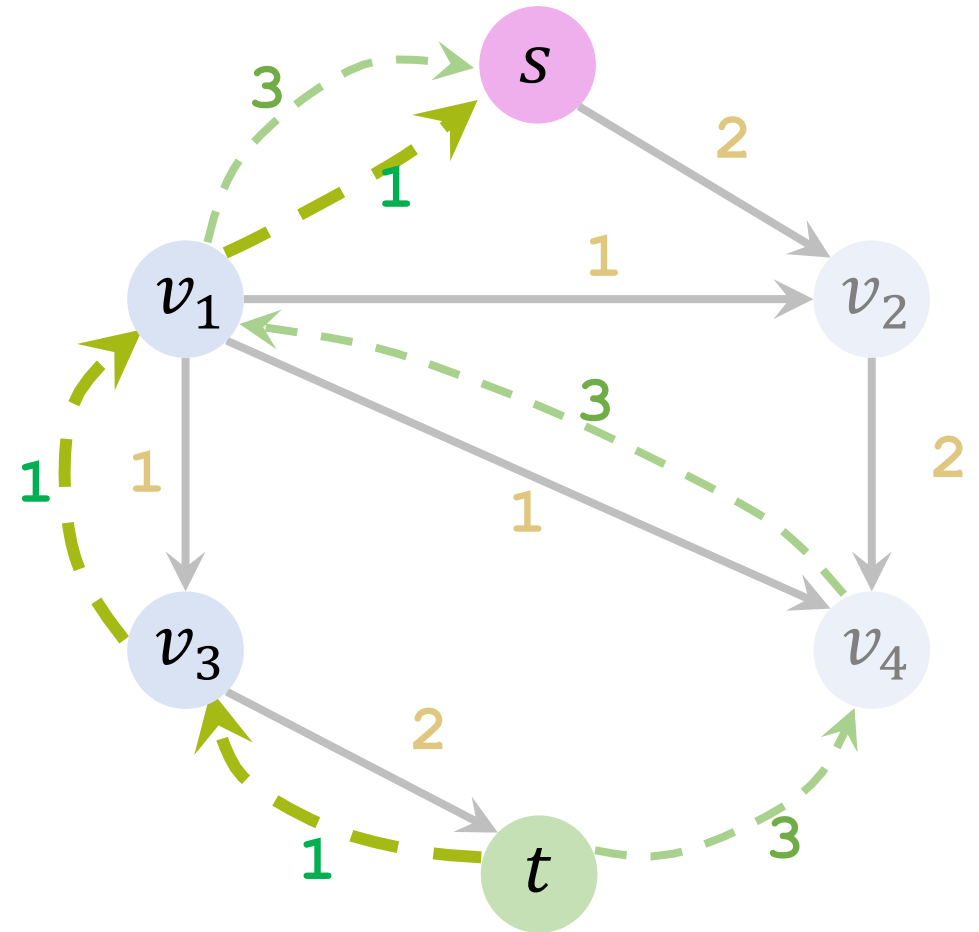
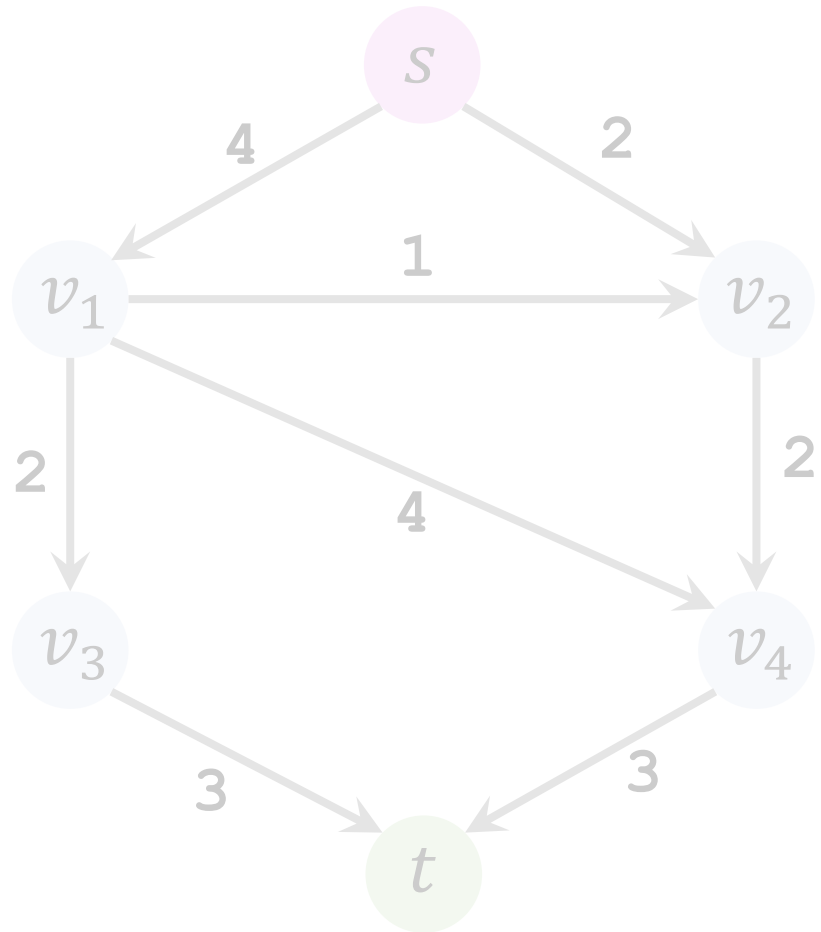
Iteration 2: Update residuals



Iteration 2: Remove saturated edges

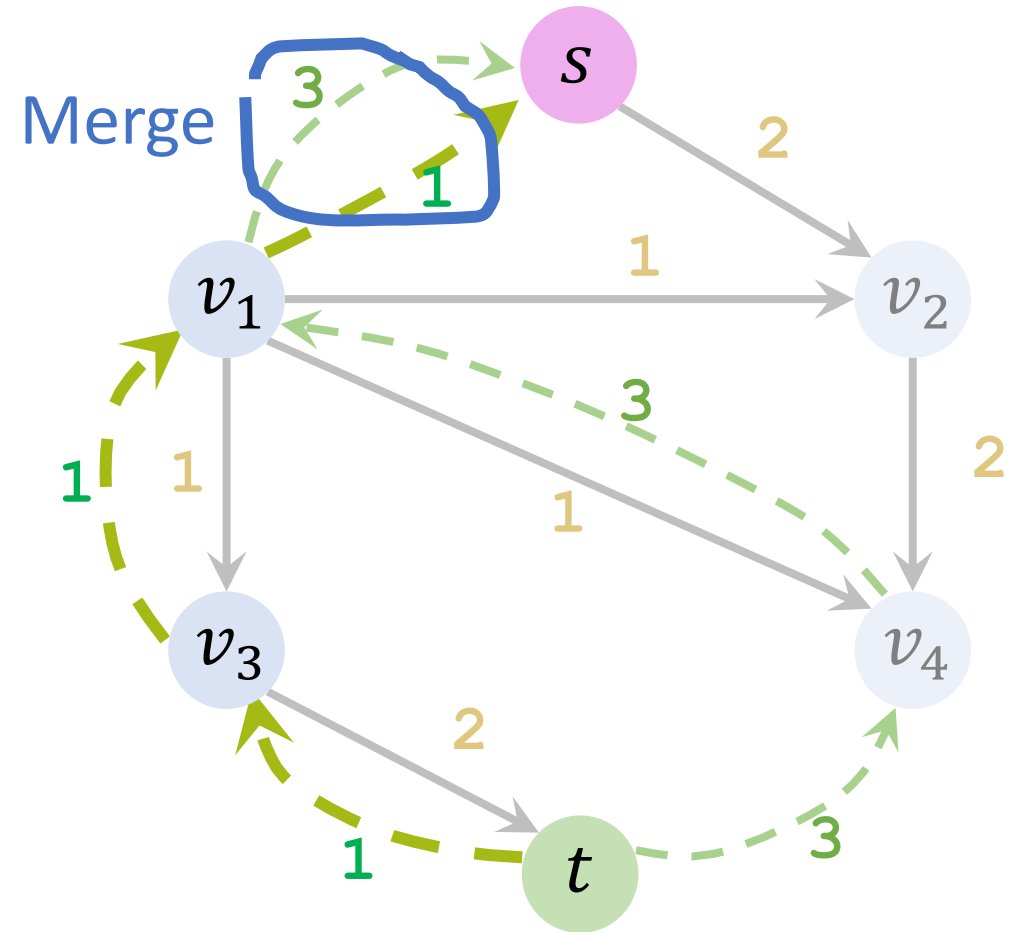
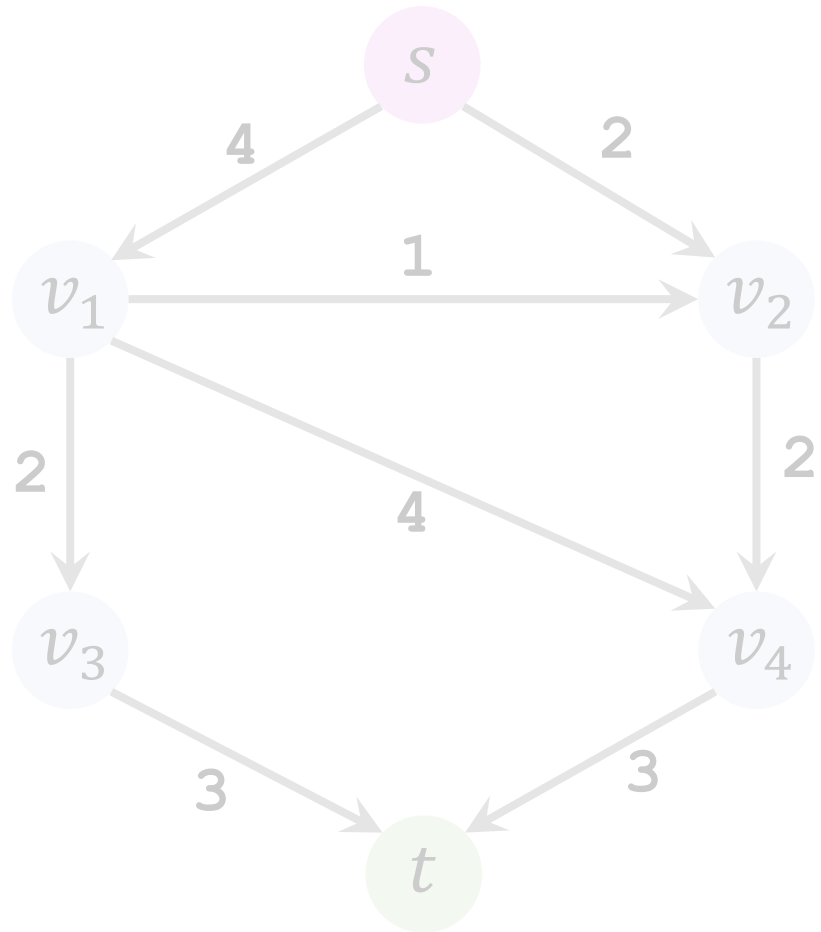


Iteration 2: Add a backward path



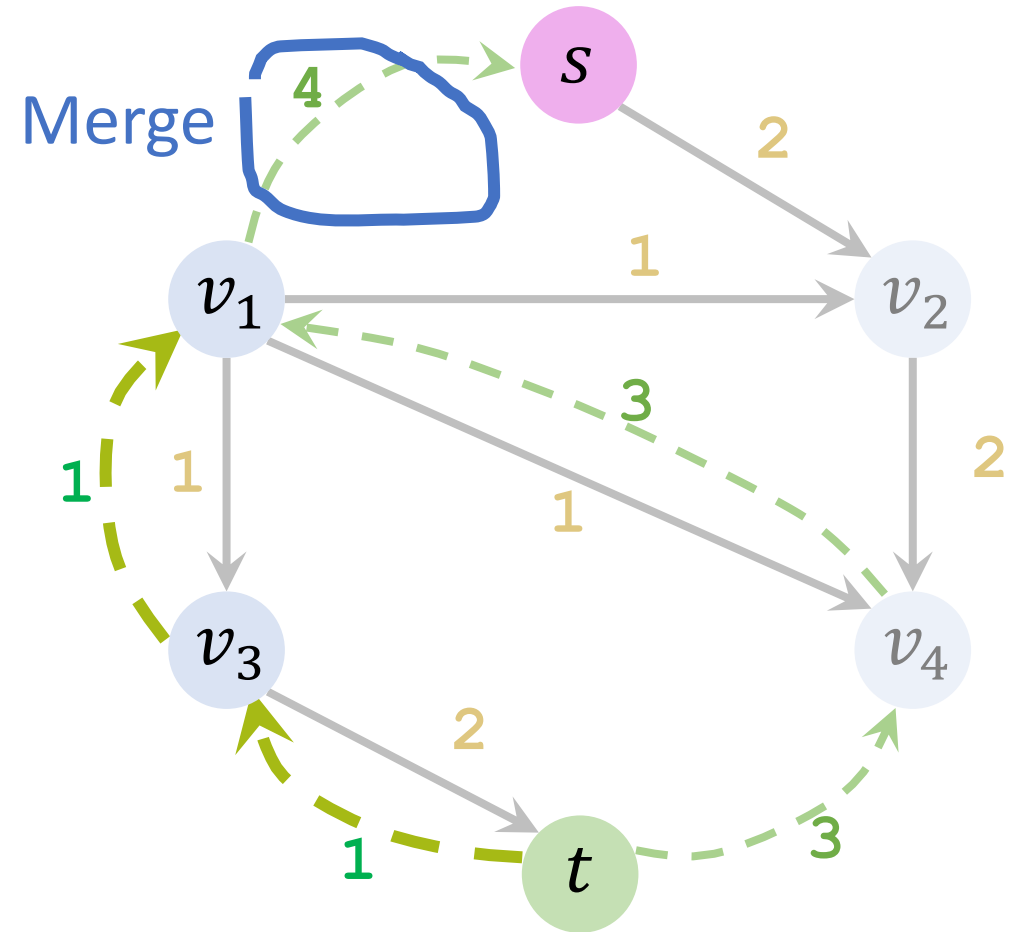
Add path $t \rightarrow v_3 \rightarrow v_1 \rightarrow s$ with capacity = 1.

Iteration 2: Add a backward path



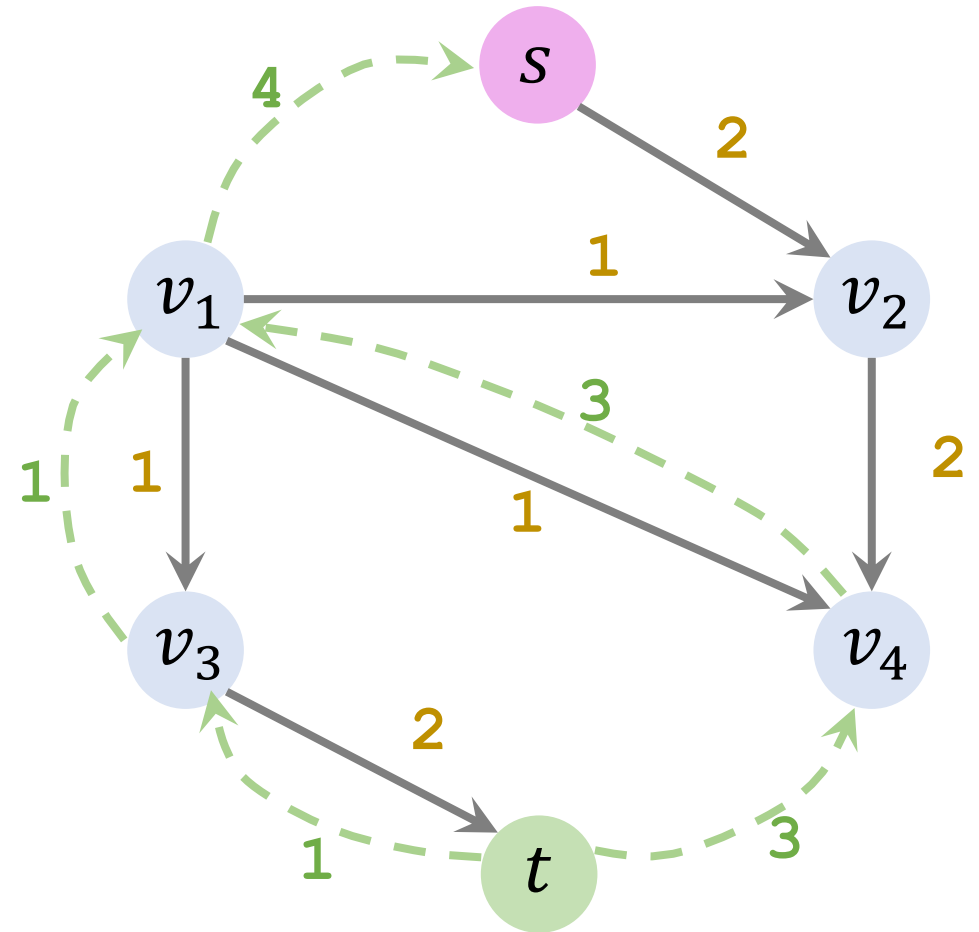
Add path $t \rightarrow v_3 \rightarrow v_1 \rightarrow s$ with capacity = 1.

Iteration 2: Add a backward path

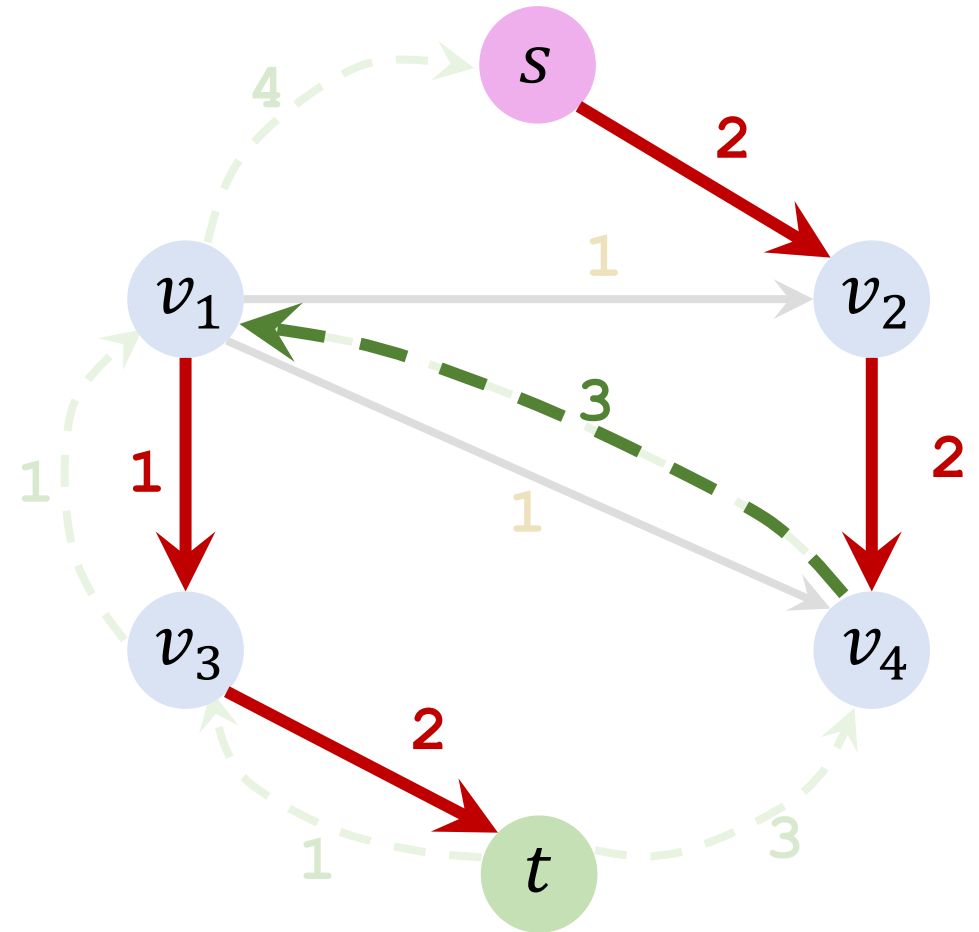


Add path $t \rightarrow v_3 \rightarrow v_1 \rightarrow s$ with capacity = 1.

Iteration 3: Find an augmenting path

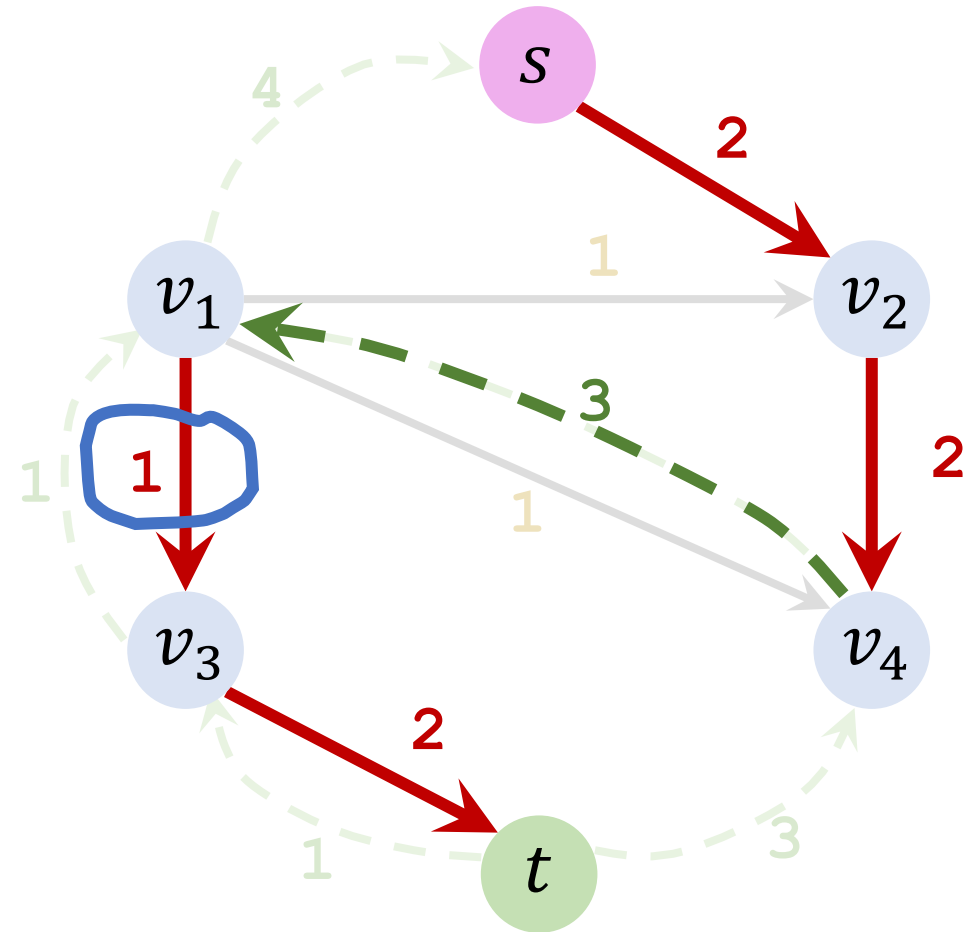


Iteration 3: Find an augmenting path



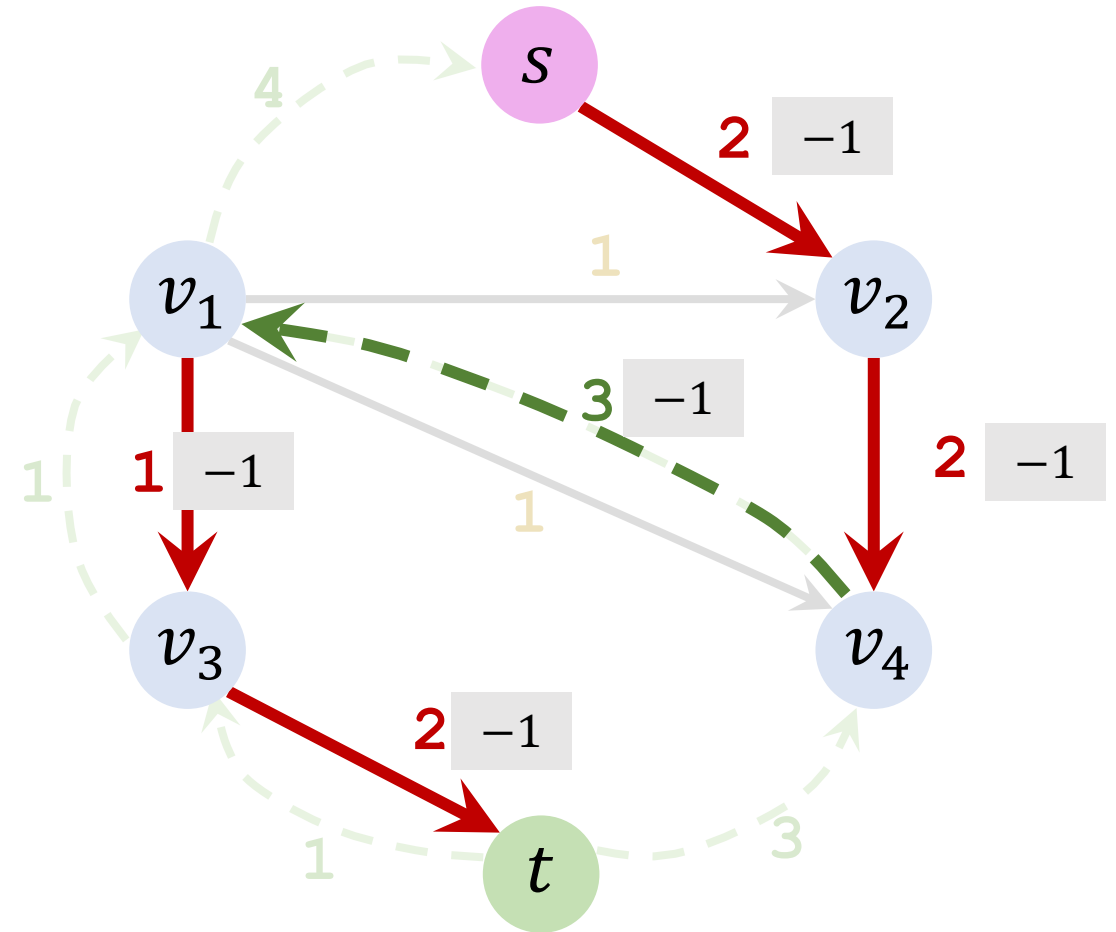
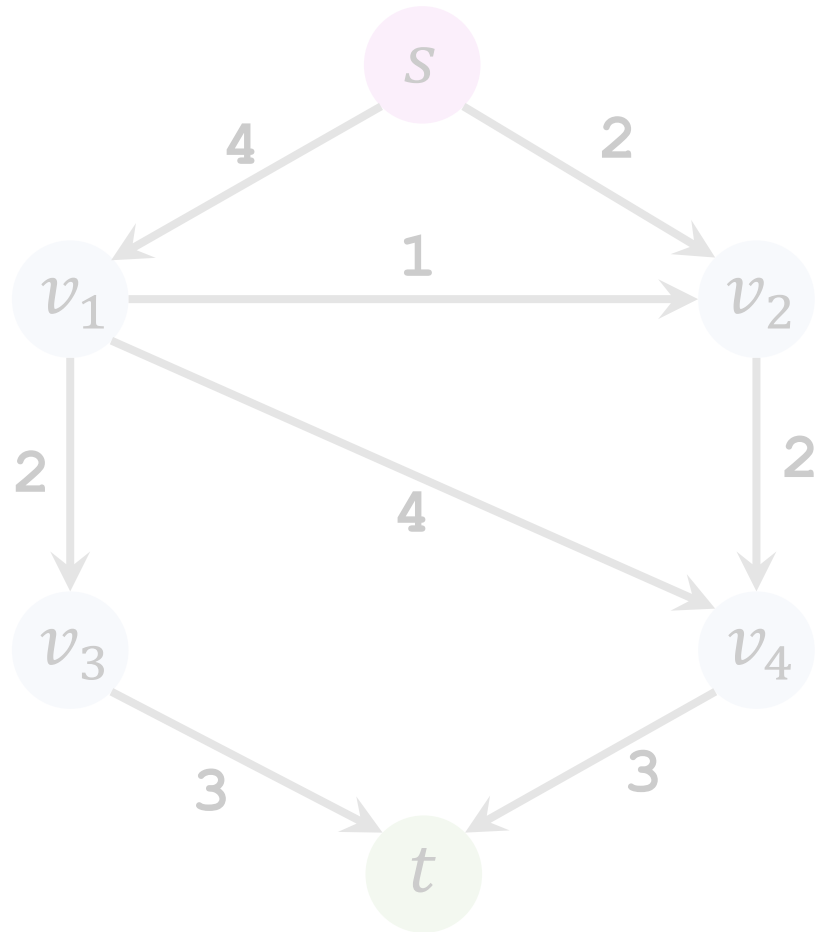
Found path $s \rightarrow v_2 \rightarrow v_4 \rightarrow v_1 \rightarrow v_3 \rightarrow t$.

Iteration 3: Find an augmenting path

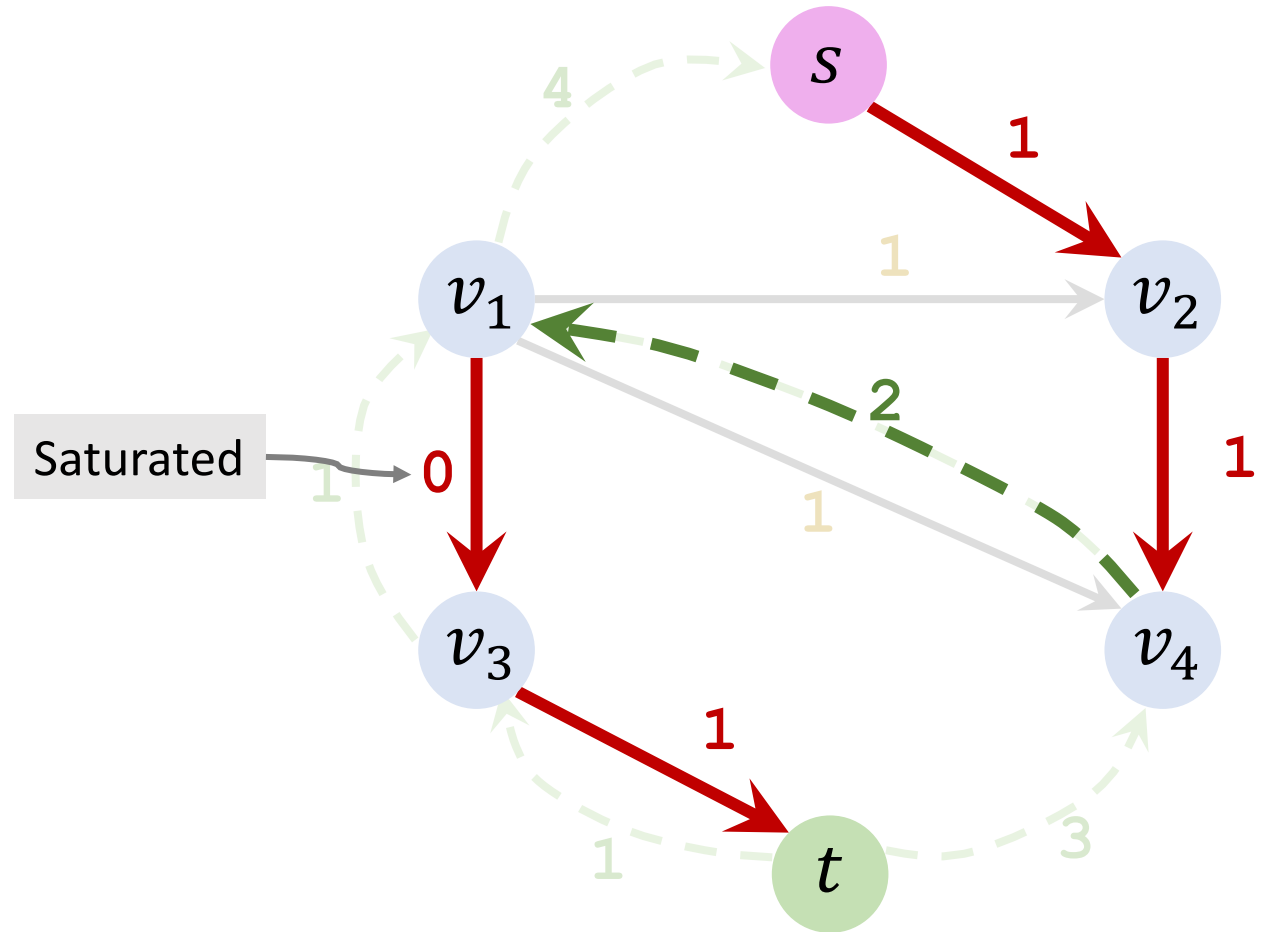
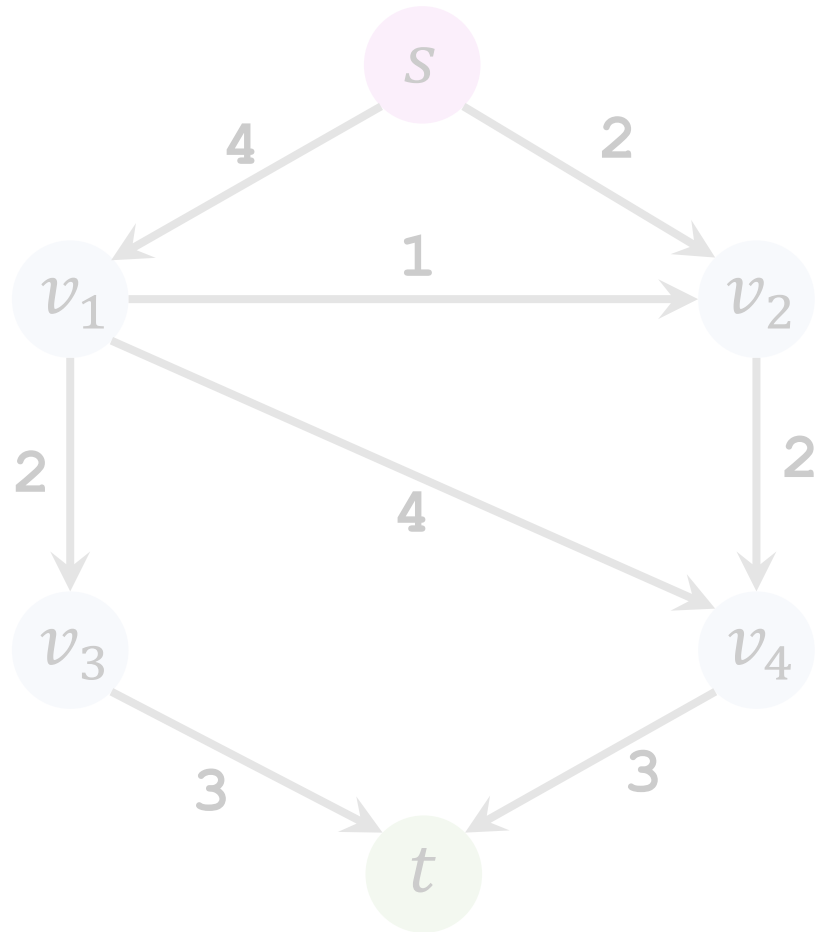


Found path $s \rightarrow v_2 \rightarrow v_4 \rightarrow v_1 \rightarrow v_3 \rightarrow t$. (Bottleneck capacity = 1.)

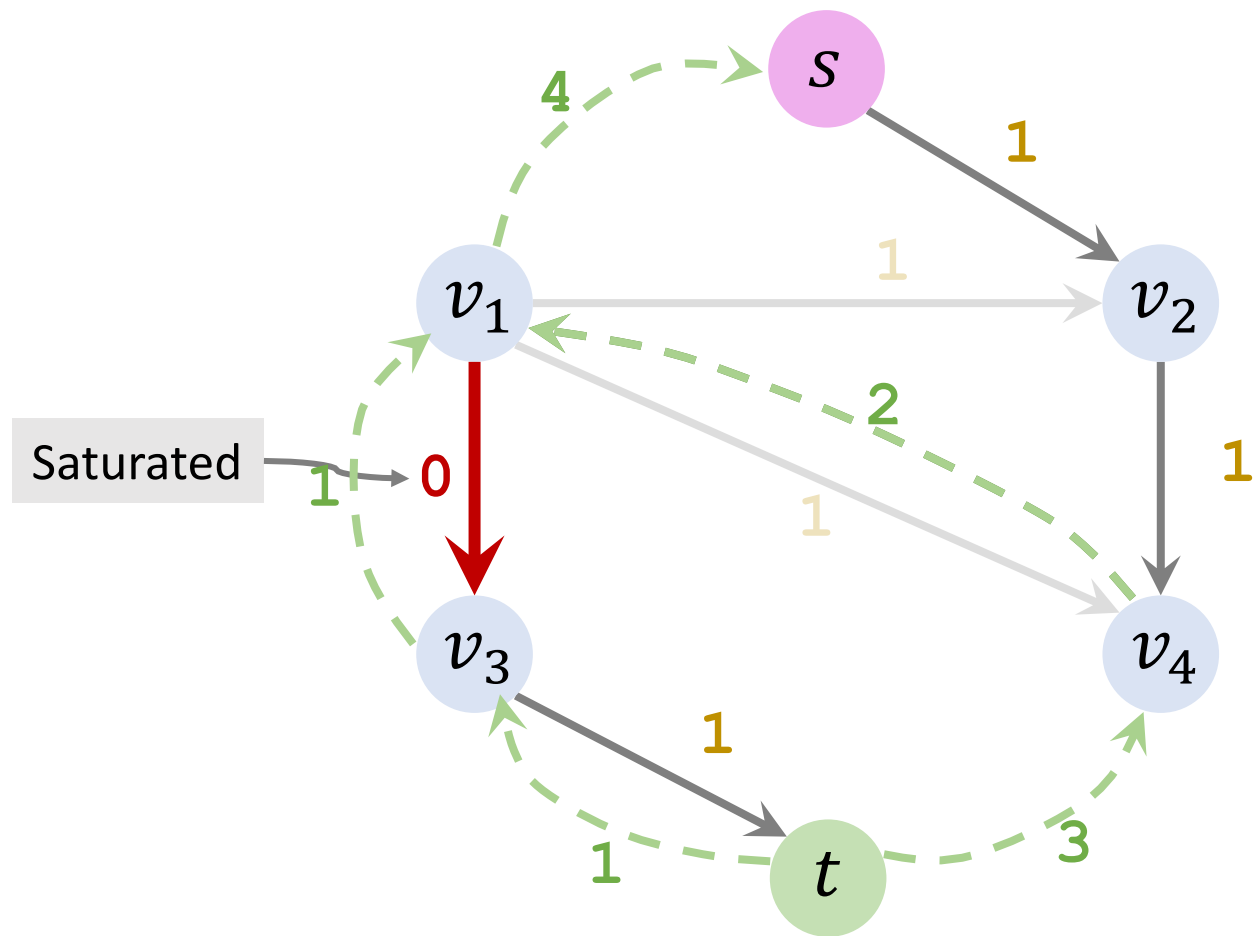
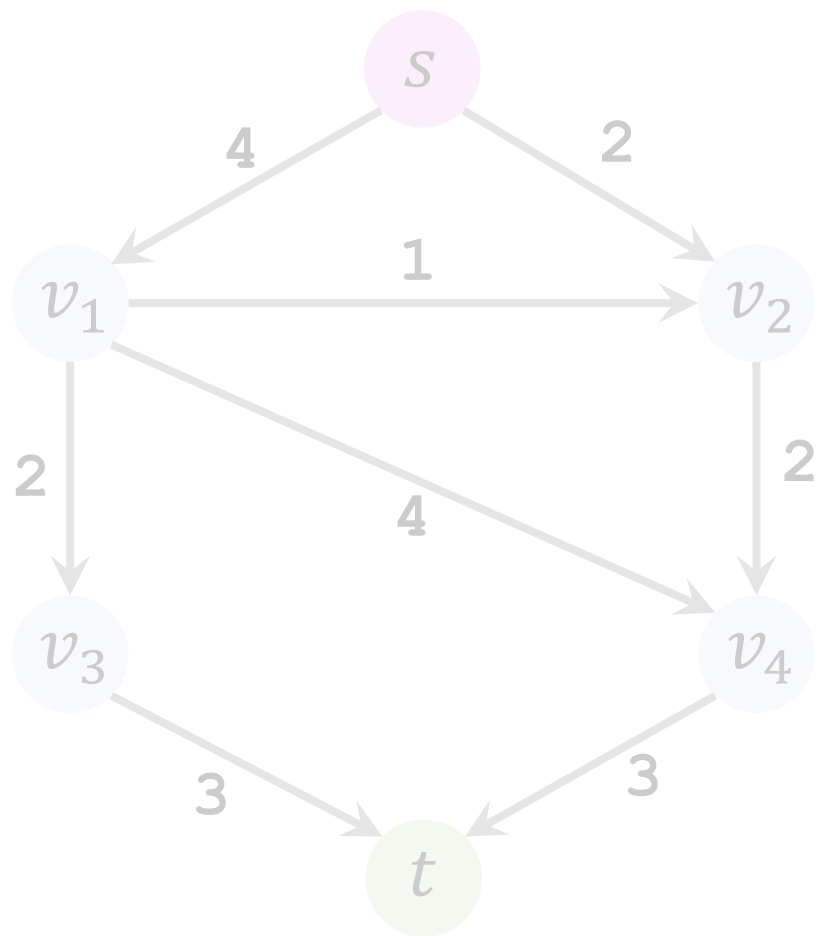
Iteration 3: Update residuals



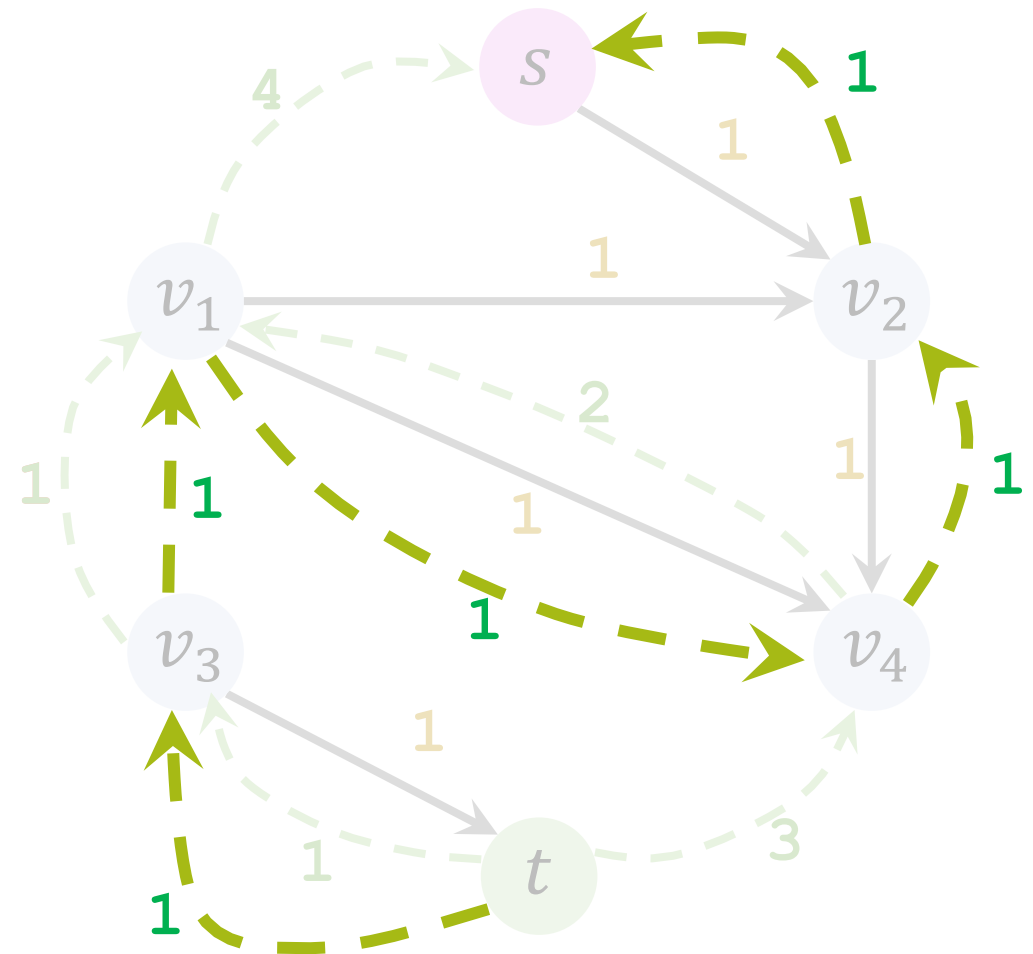
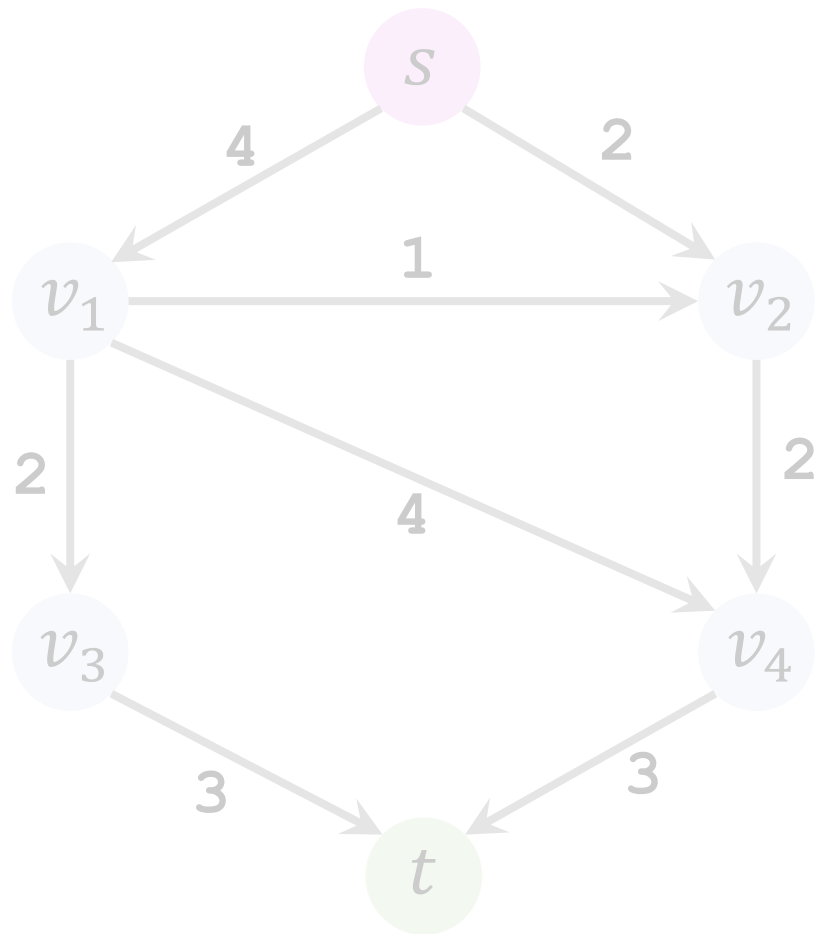
Iteration 3: Update residuals



Iteration 3: Remove saturated edges

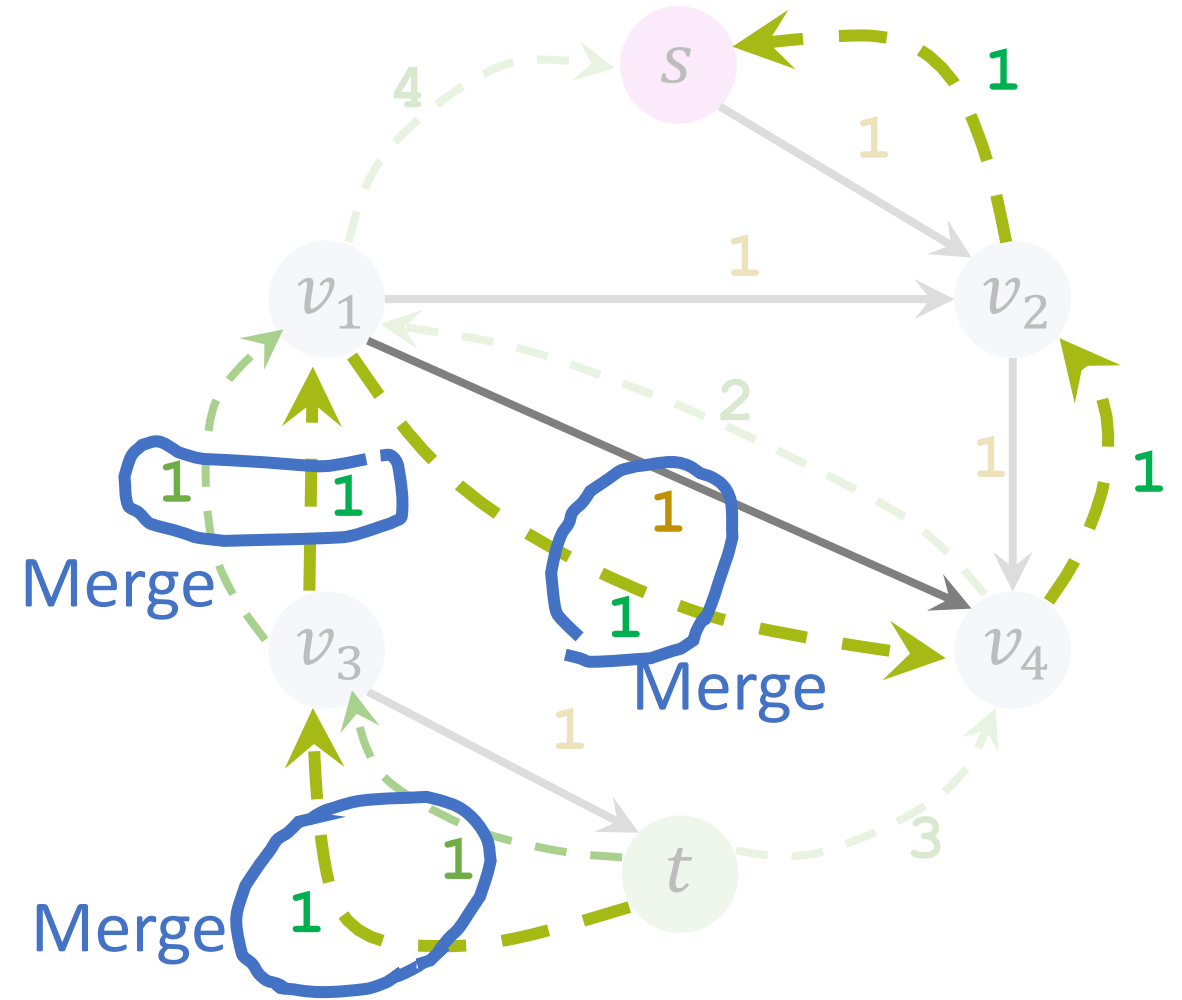


Iteration 3: Add a backward path

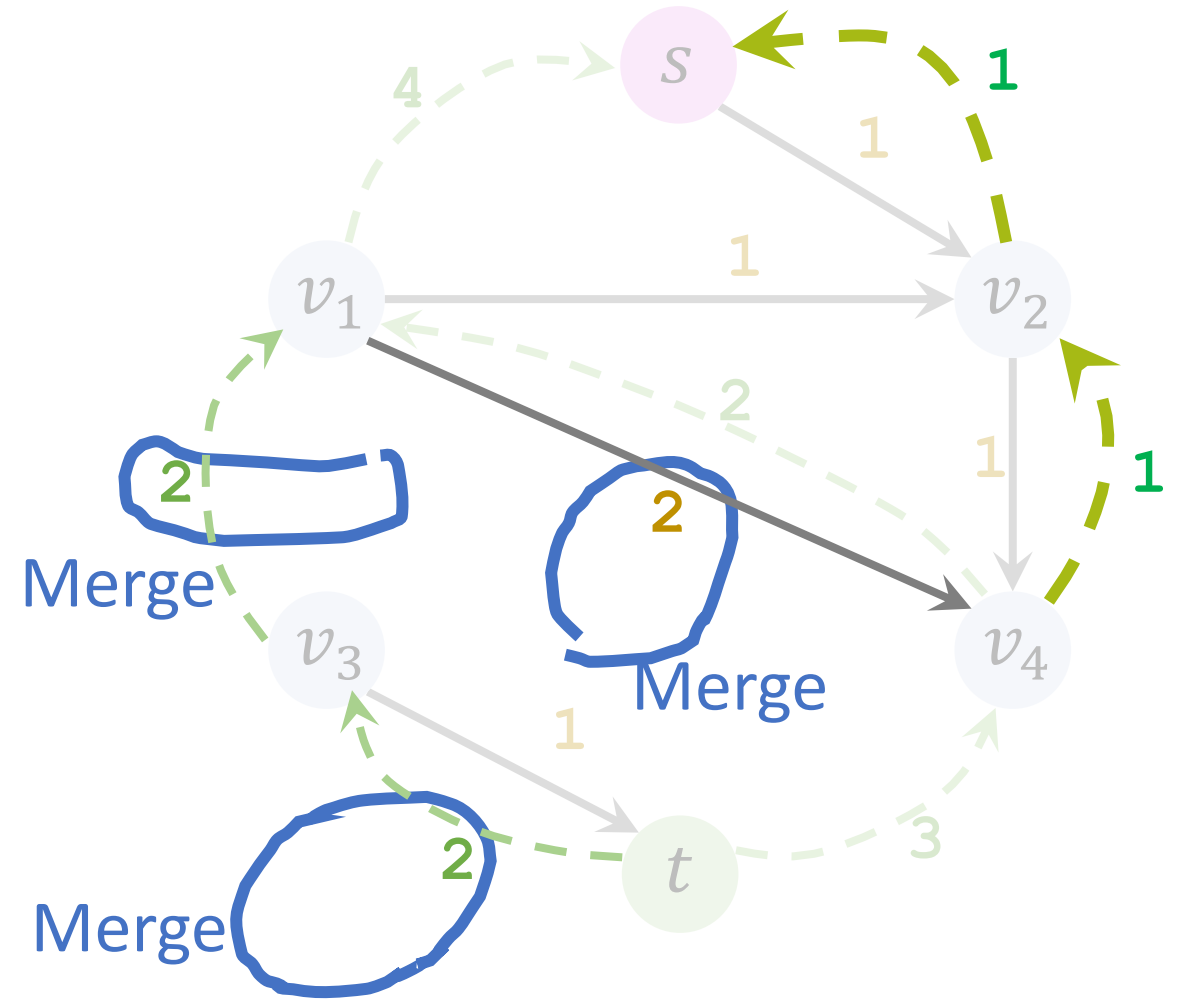


Add backward path $t \rightarrow v_3 \rightarrow v_1 \rightarrow v_4 \rightarrow v_2 \rightarrow s$ with capacity = 1.

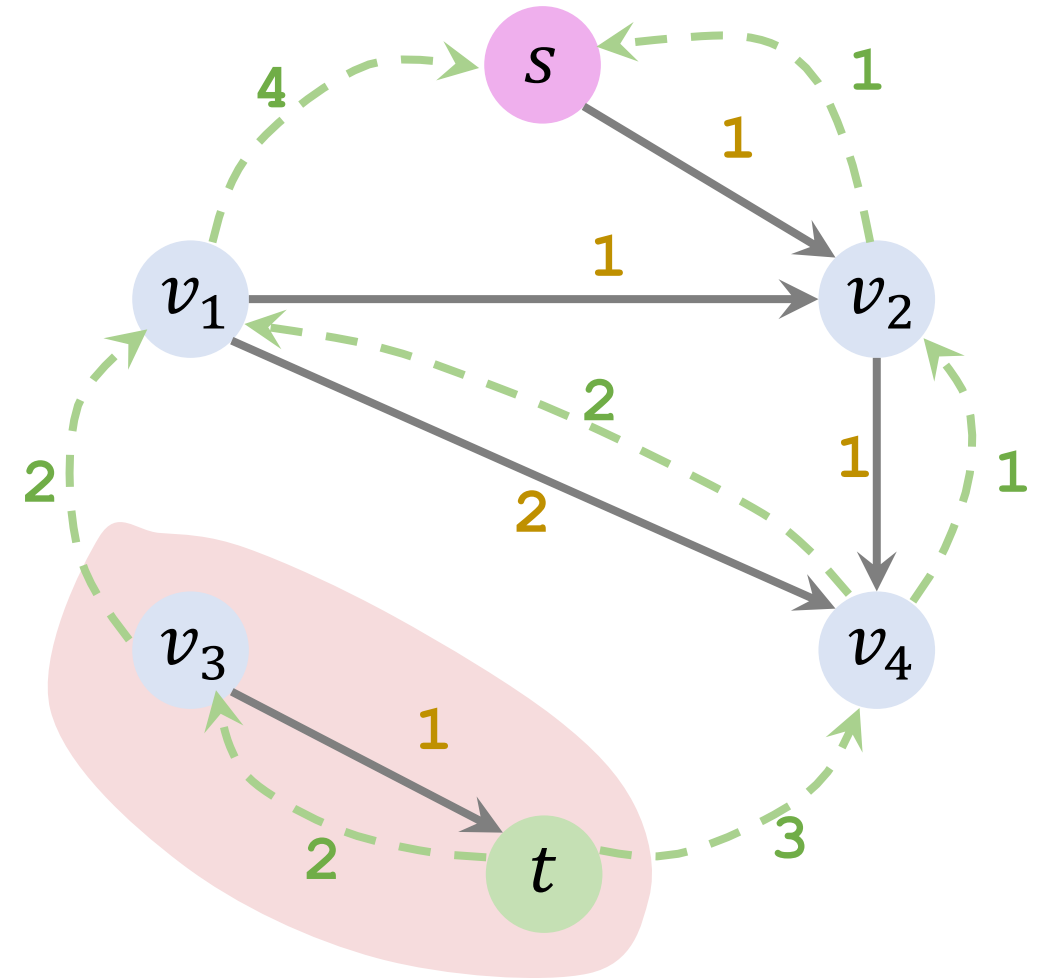
Iteration 3: Add a backward path



Iteration 3: Add a backward path

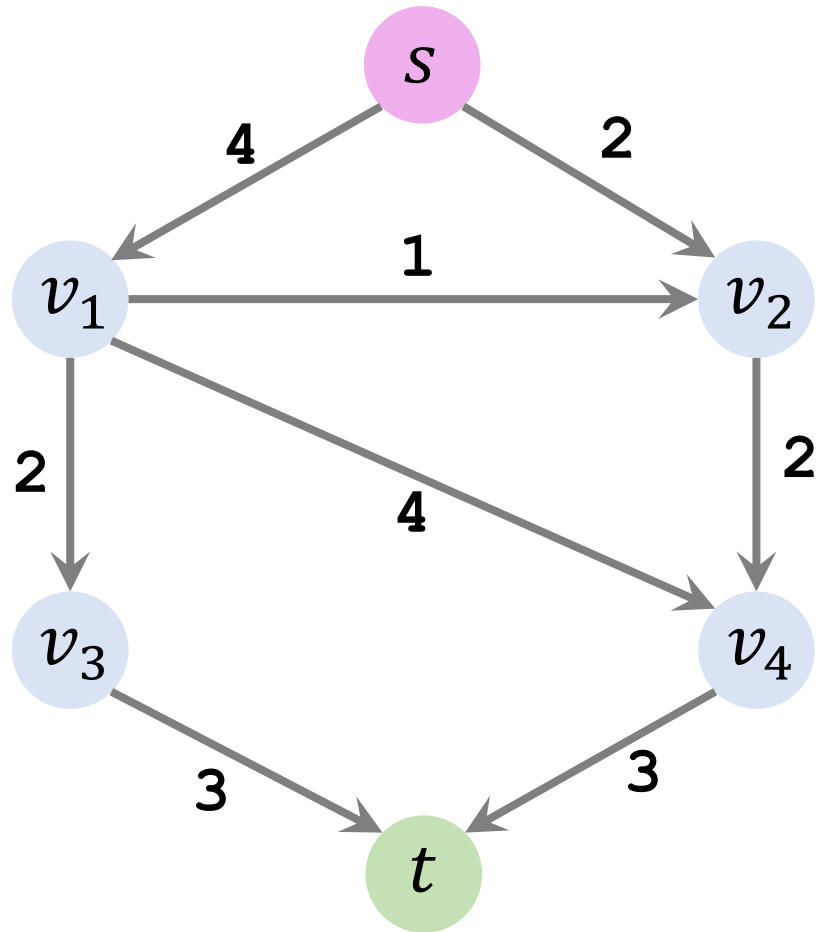


Iteration 4: Find an augmenting path

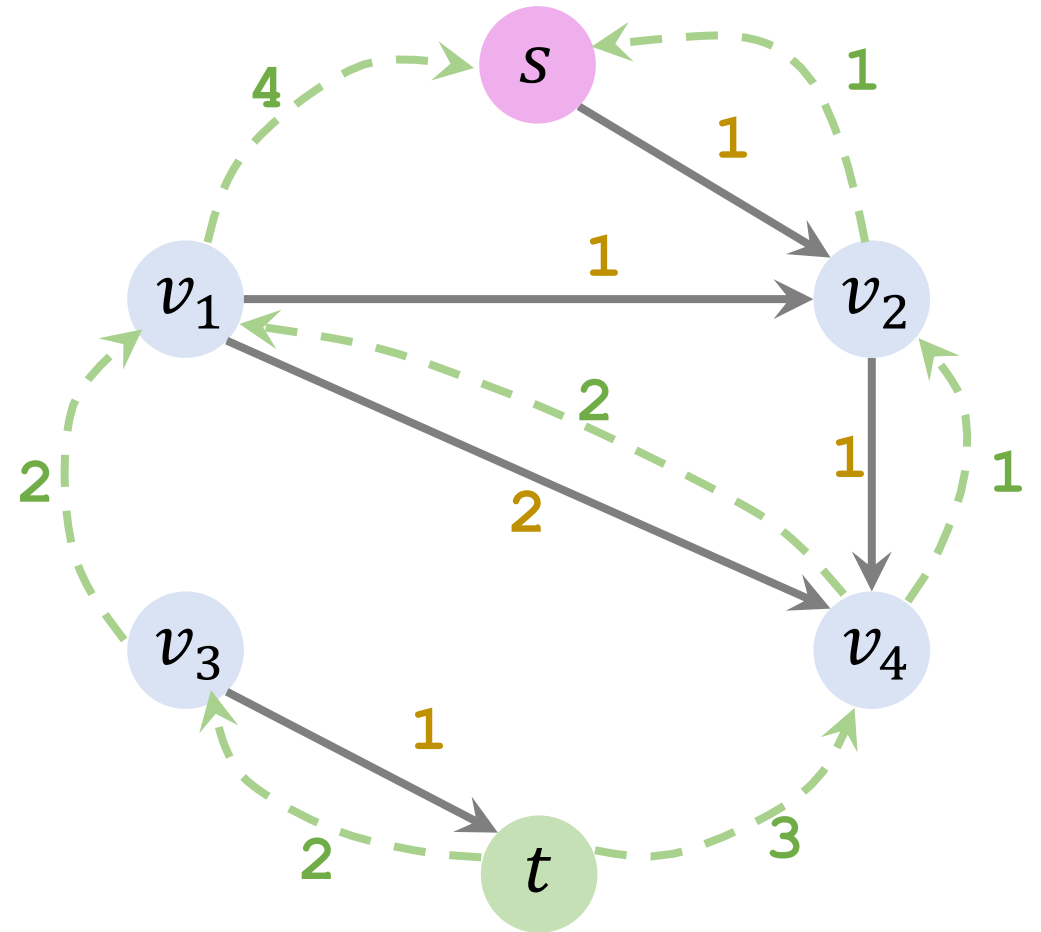


Cannot find any path from source to sink.

End of Procedure

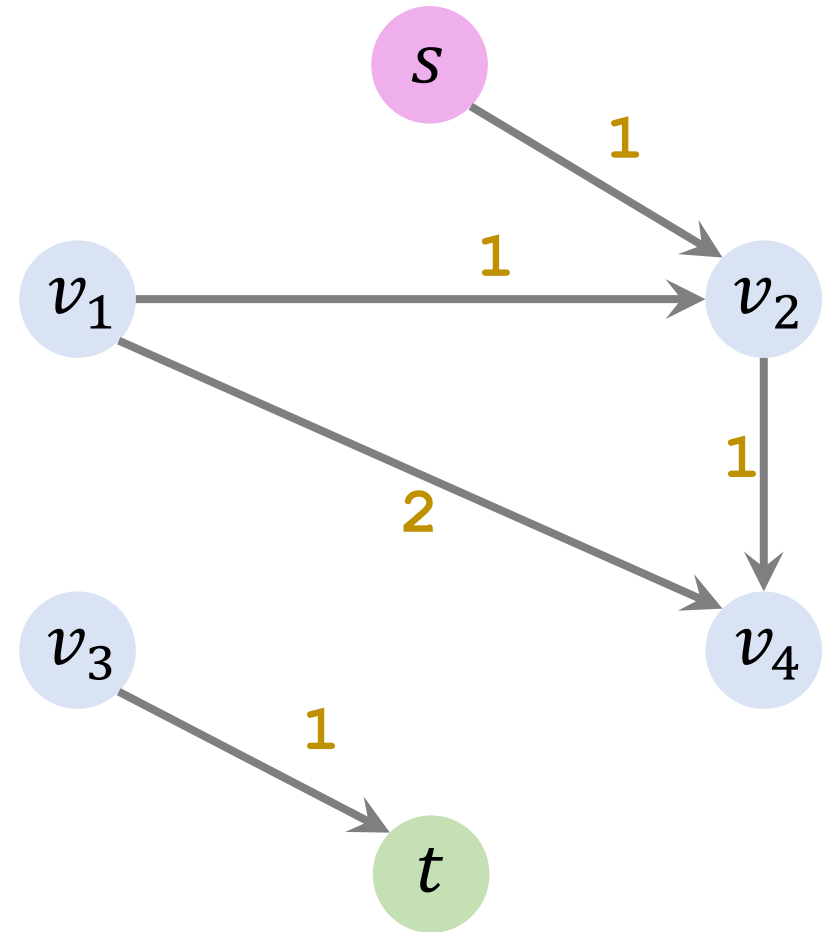
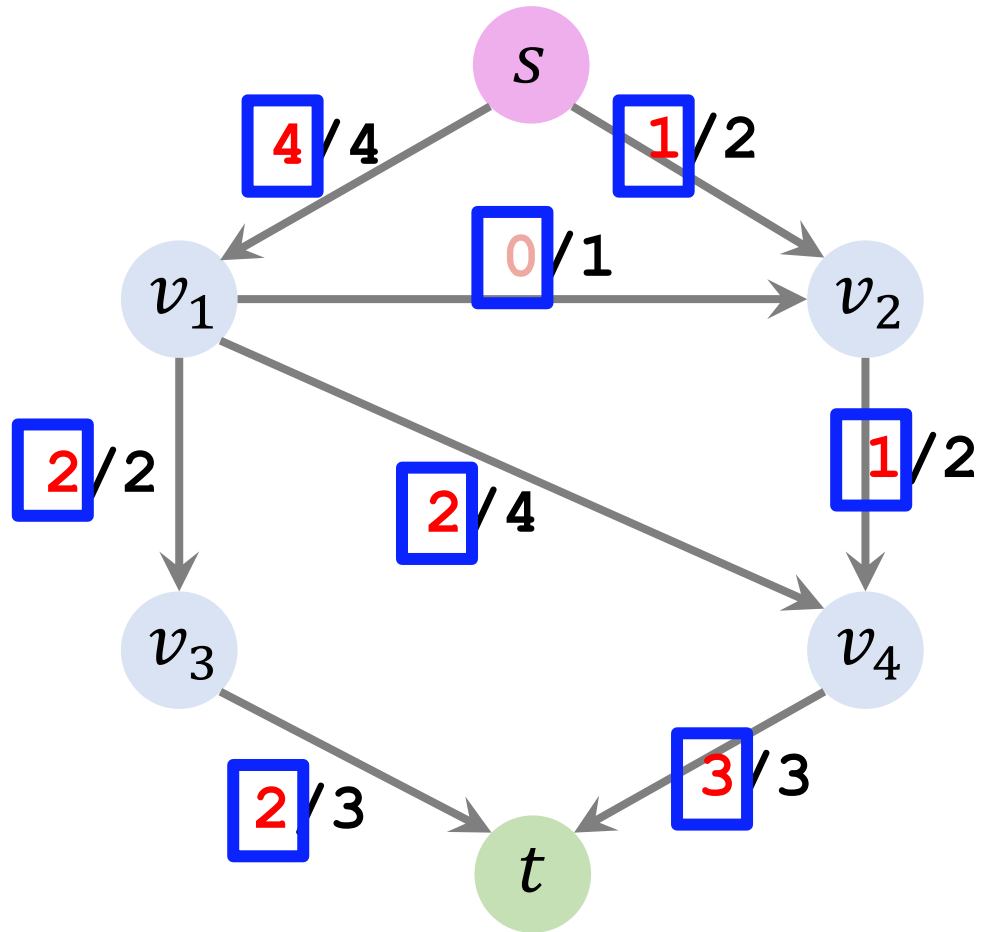


Original Graph



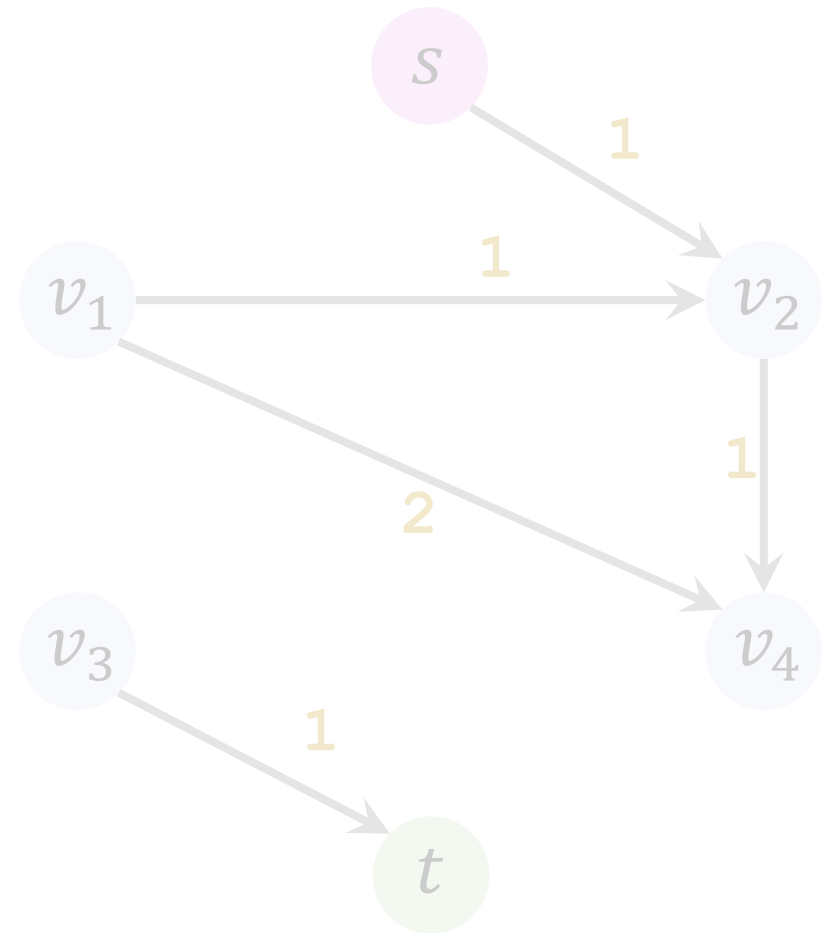
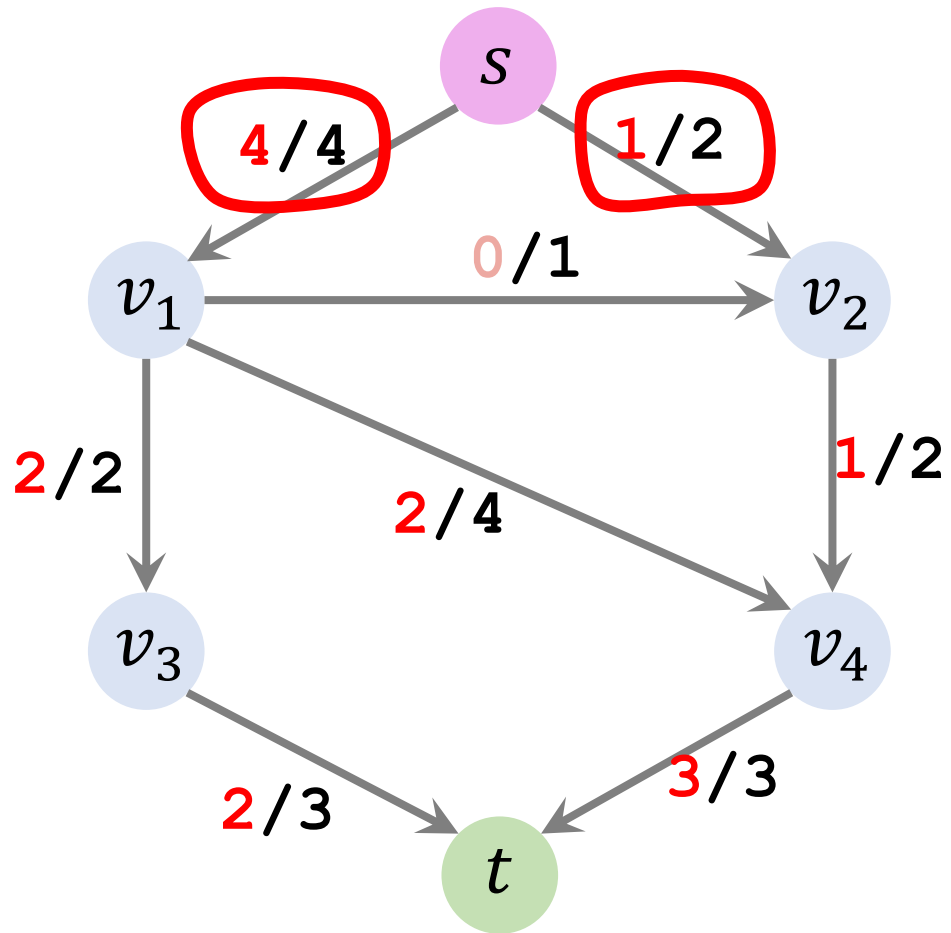
Residual Graph

End of Procedure



Flow = Capacity - Residual.

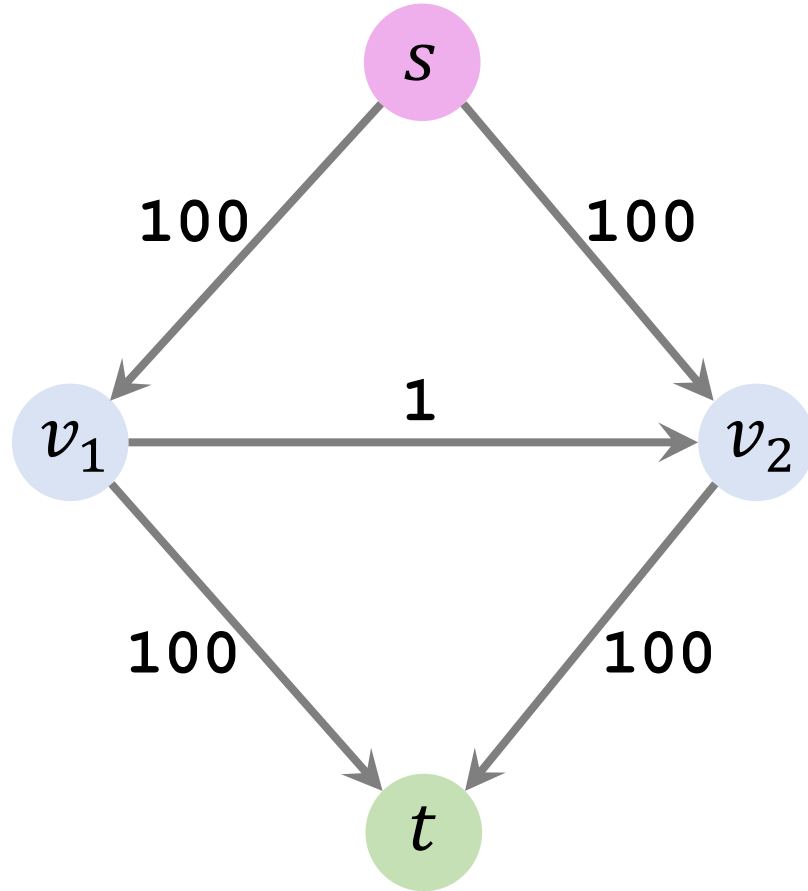
End of Procedure



Amount of Max Flow = 5

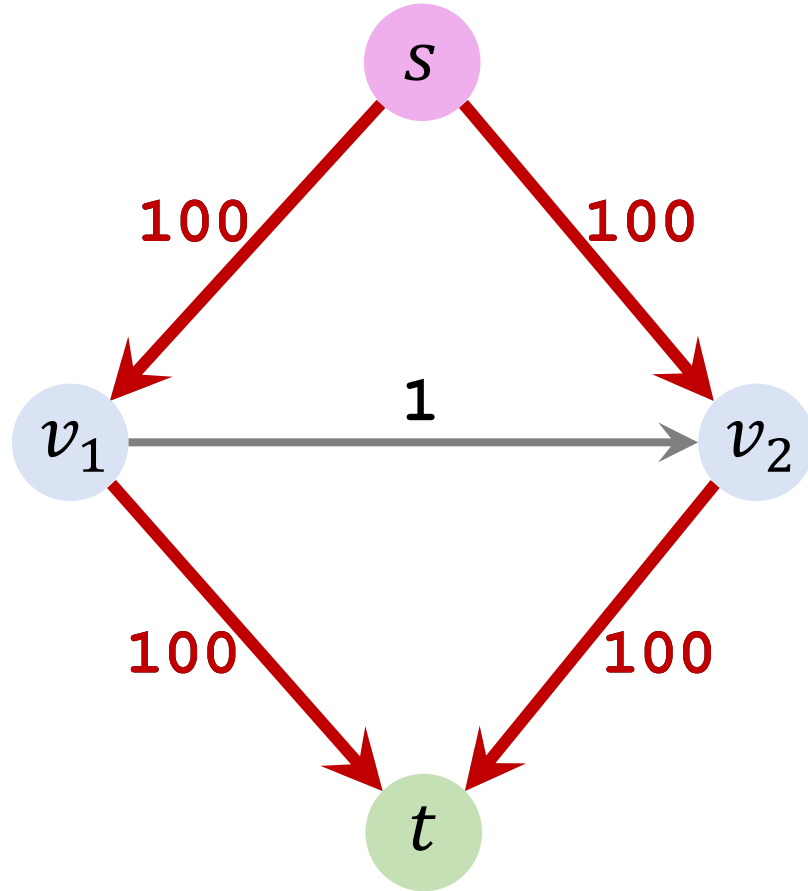
Worst-Case Time Complexity

A bad case for Ford-Fulkerson algorithm



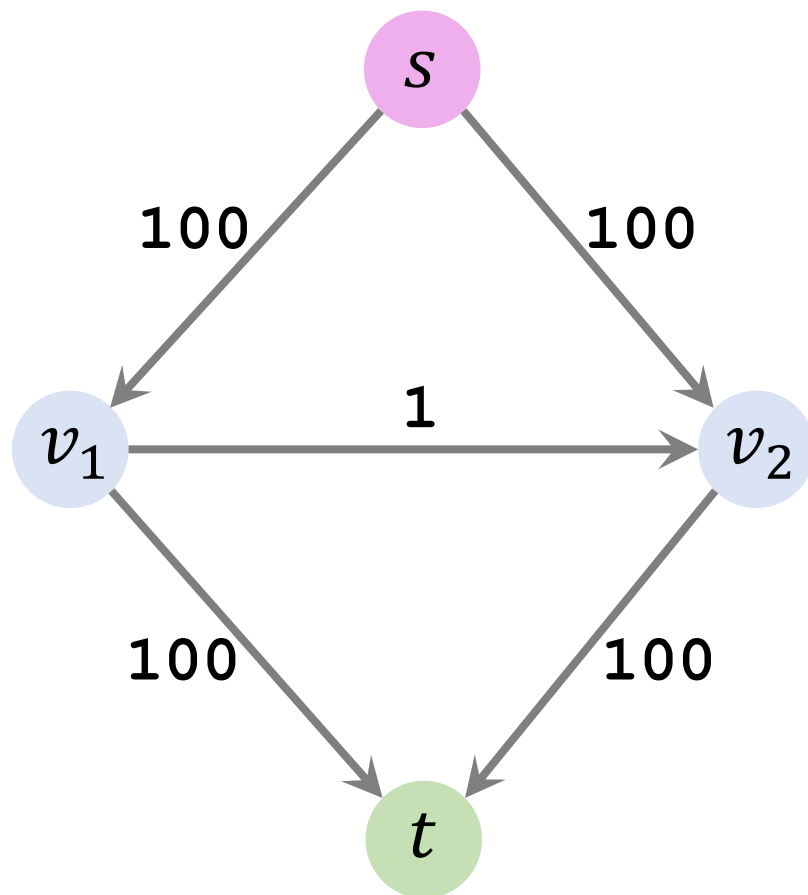
- The amount of the max flow is 200.

A bad case for Ford-Fulkerson algorithm

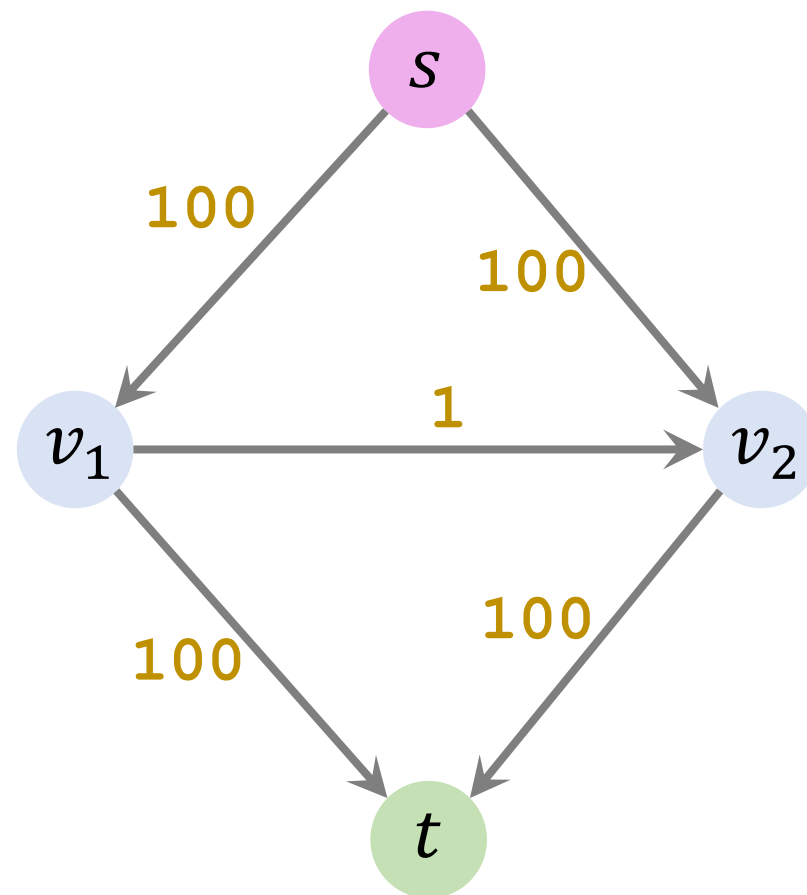


- The amount of the max flow is 200.
- Ford-Fulkerson algorithm may take 200 iterations to find the max flow.

Initialization

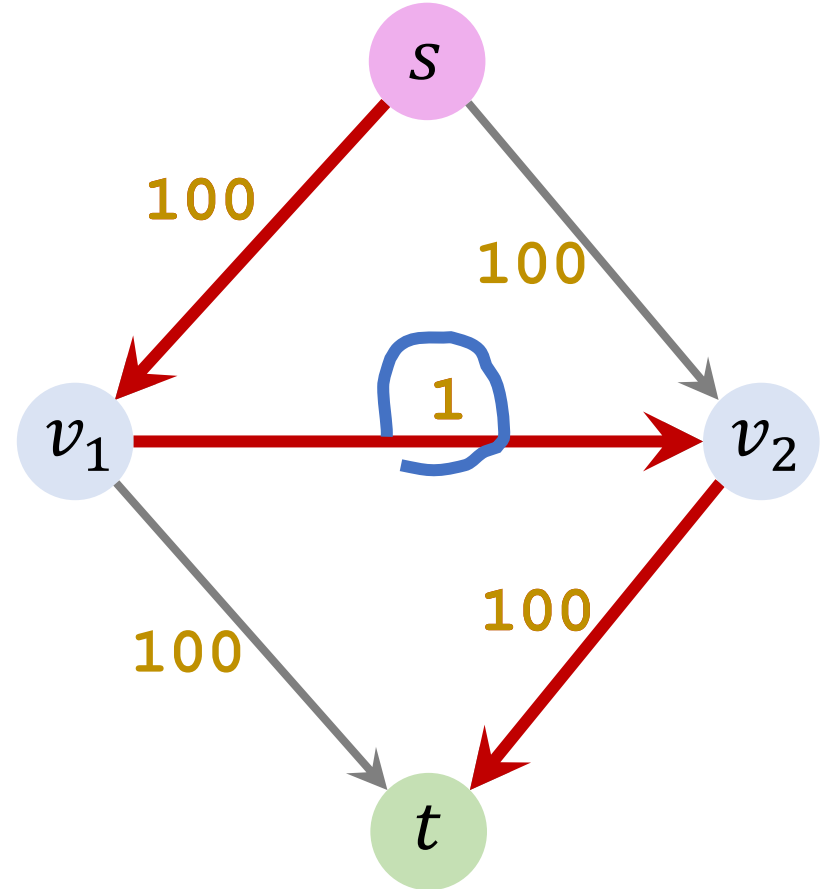
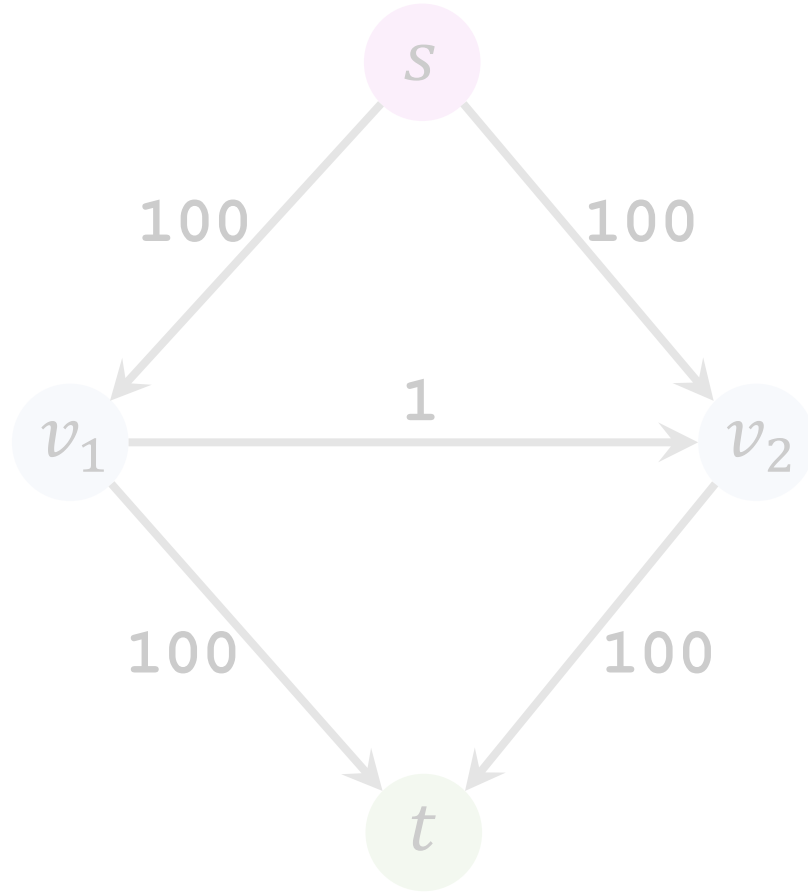


Original Graph



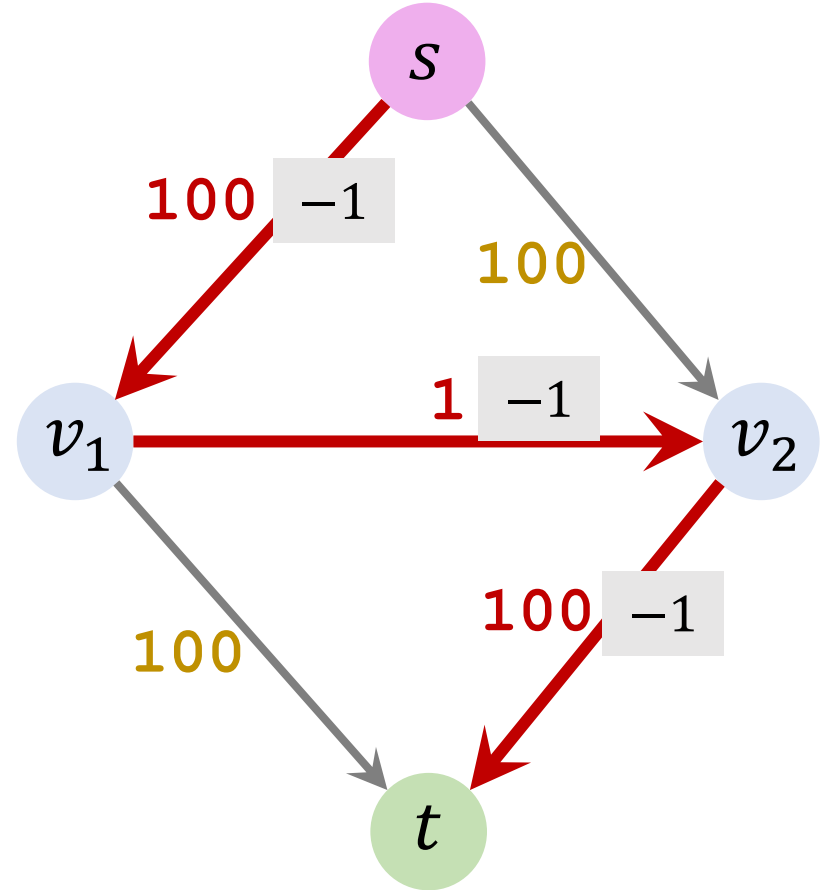
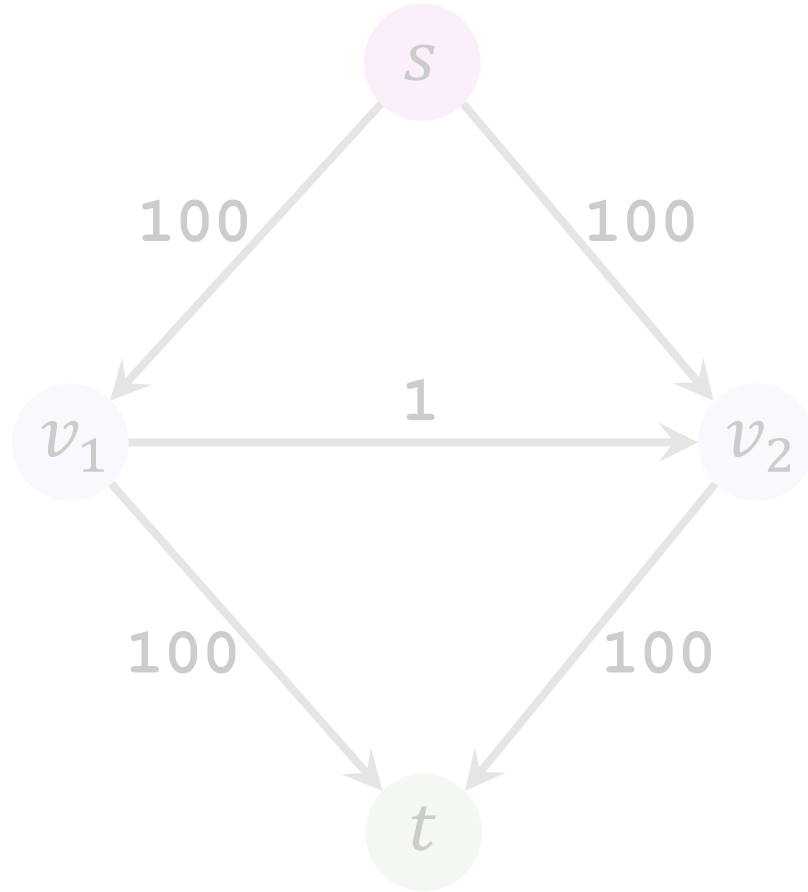
Residual Graph

Iteration 1: Find an augmenting path

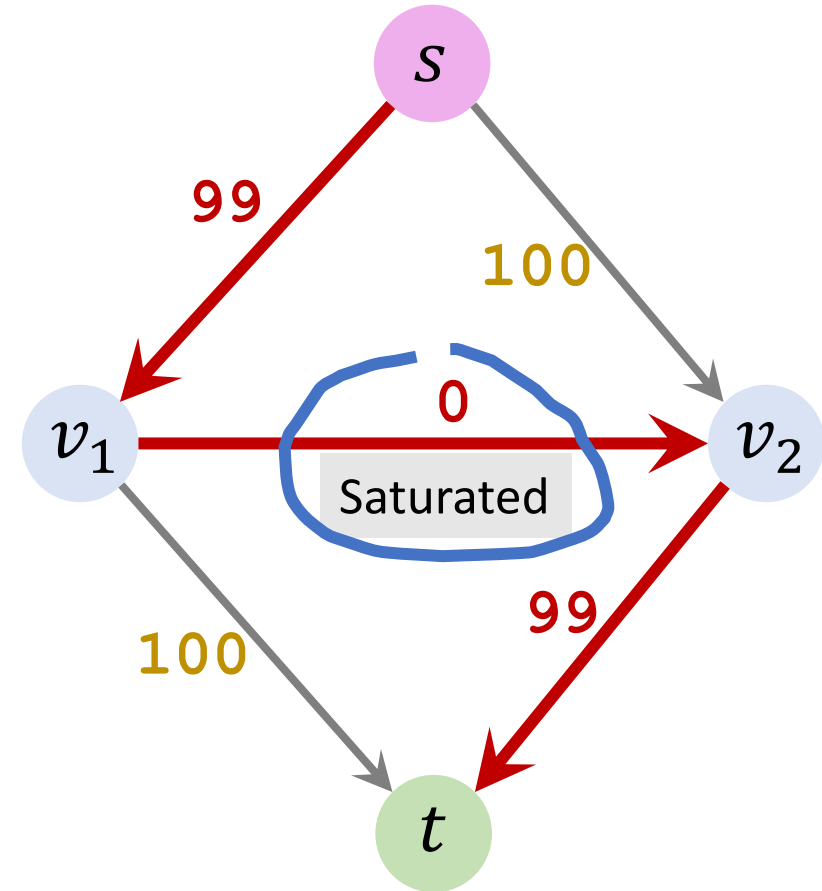
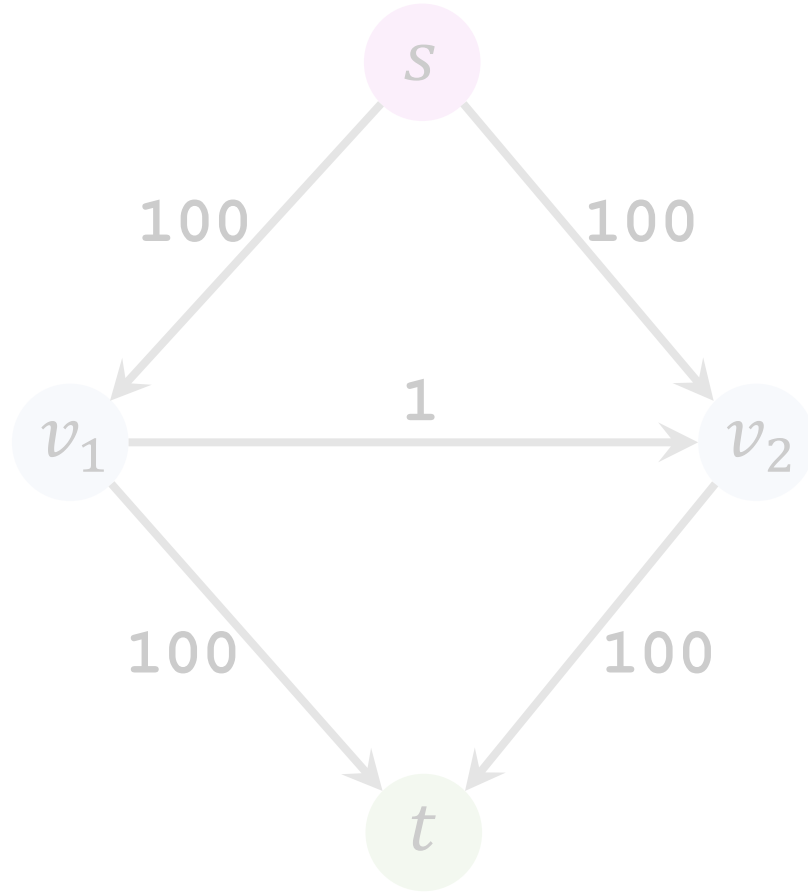


Found path $s \rightarrow v_1 \rightarrow v_2 \rightarrow t$. (Bottleneck capacity = 1.)

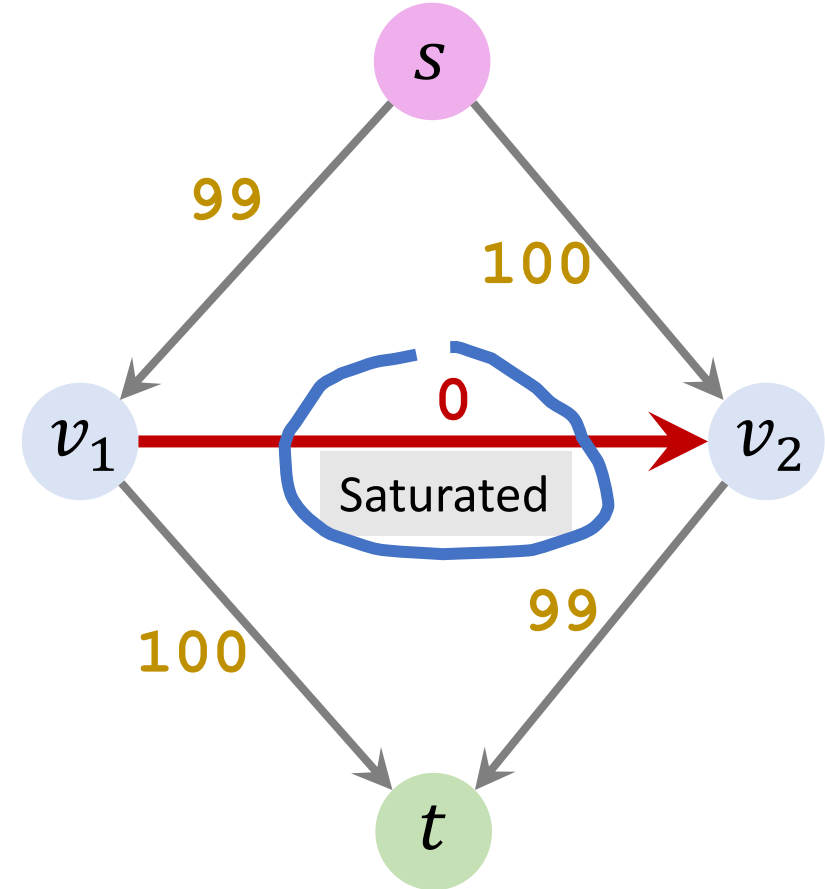
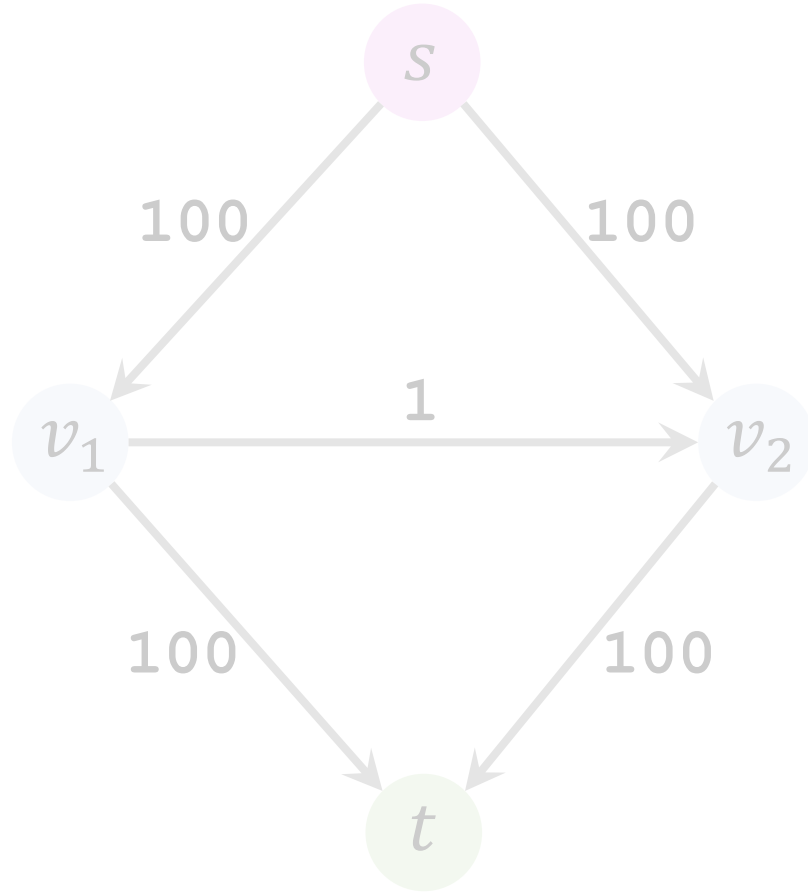
Iteration 1: Update residuals



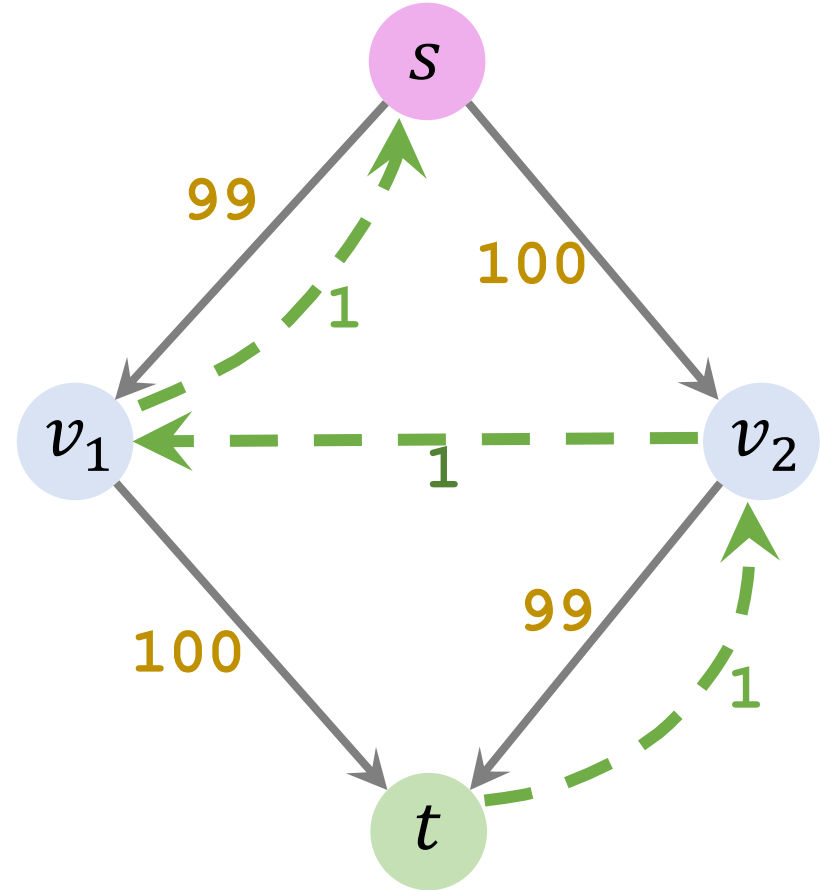
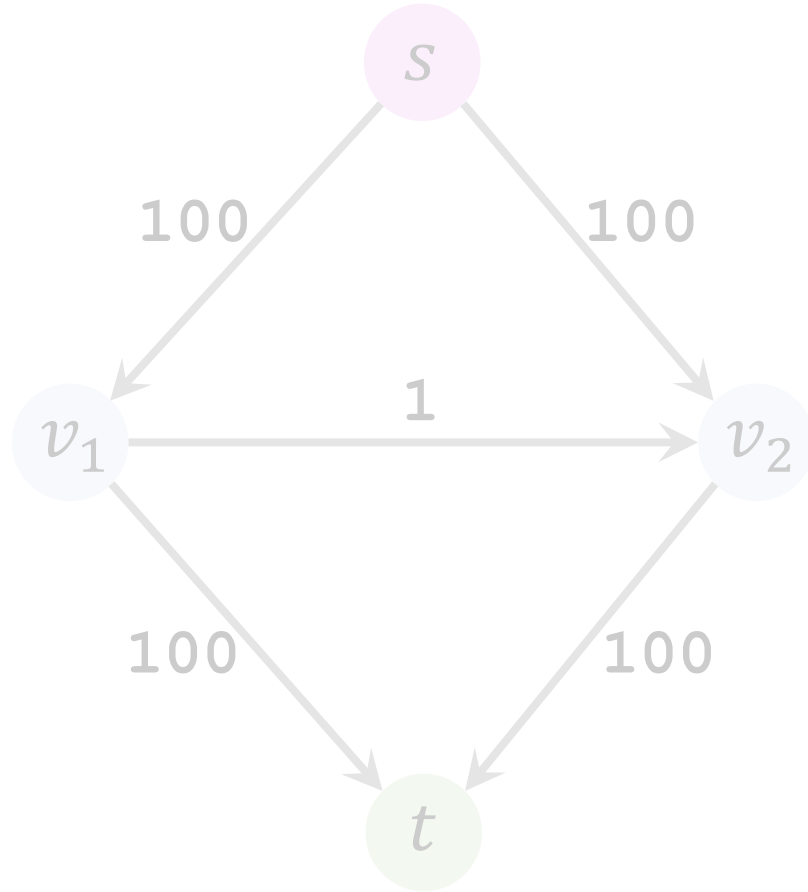
Iteration 1: Update residuals



Iteration 1: Remove saturated edges

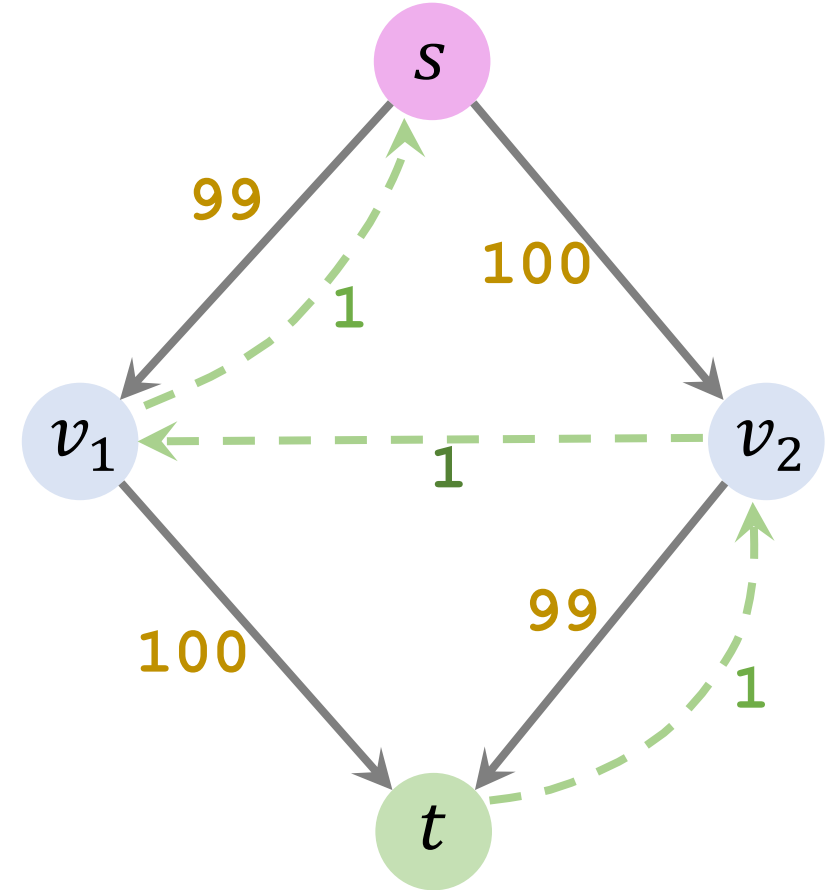
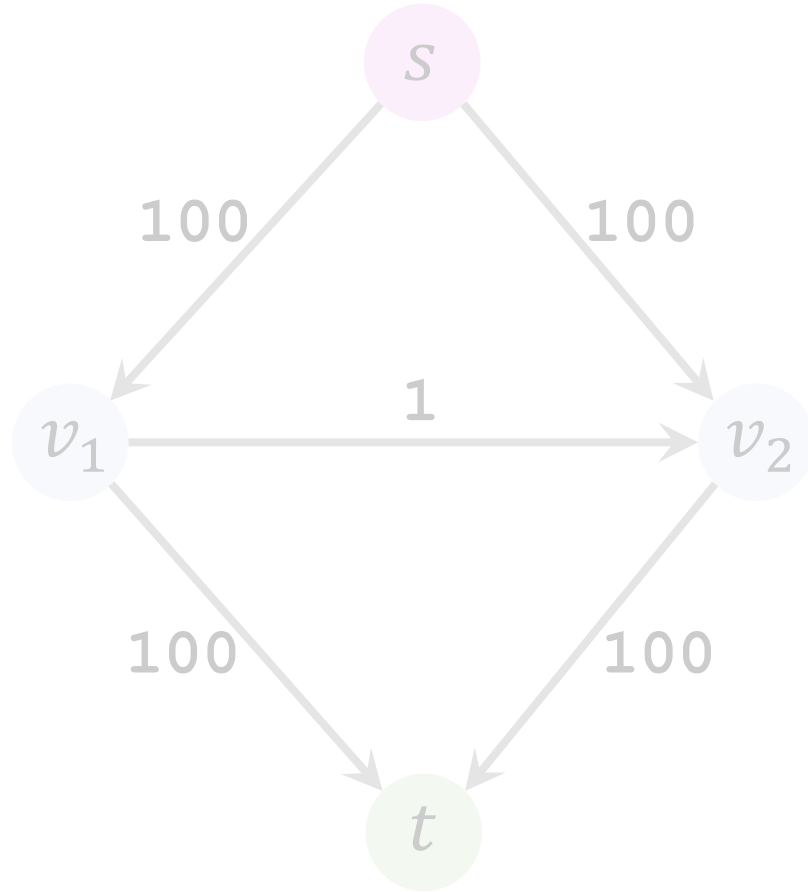


Iteration 1: Add backward path

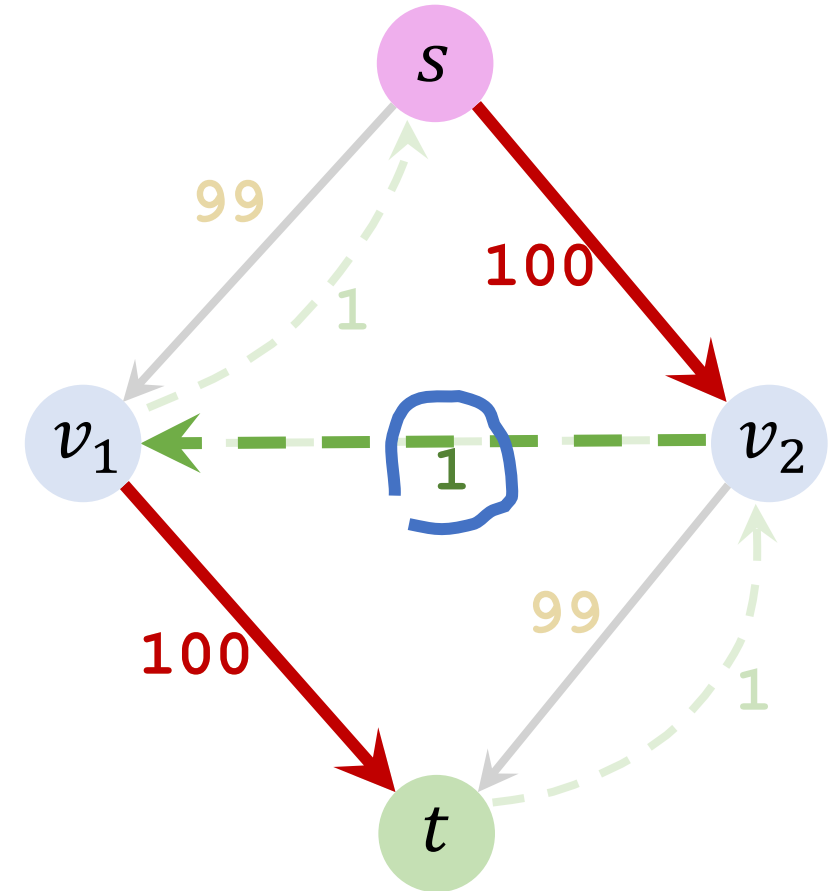
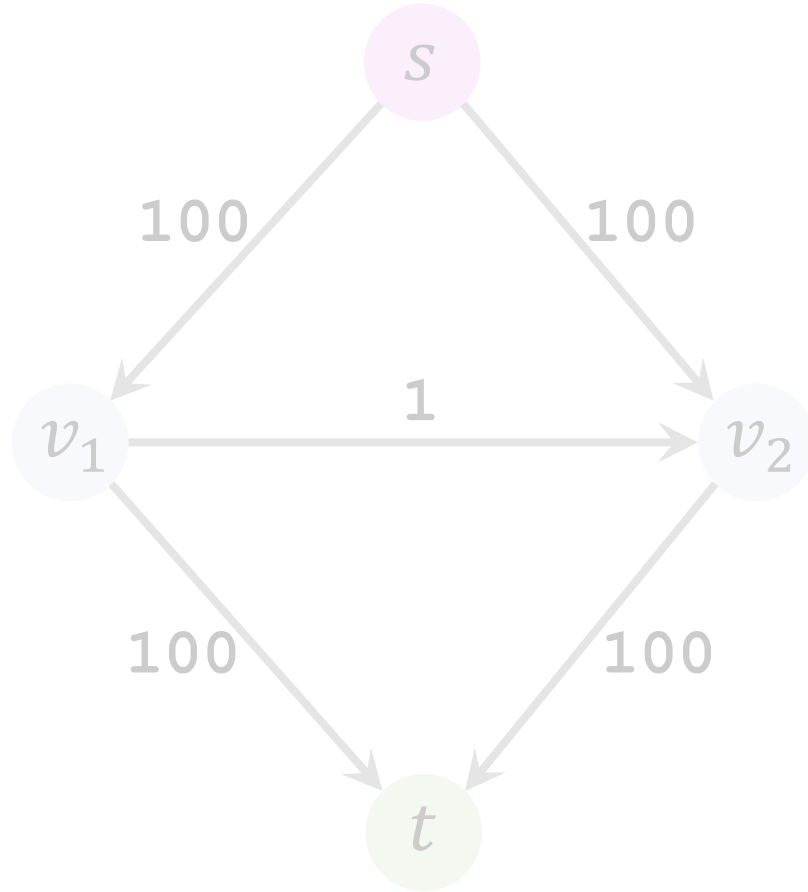


Add backward path $t \rightarrow v_2 \rightarrow v_1 \rightarrow s$ with capacity = 1.

Now, the amount of flow is 1

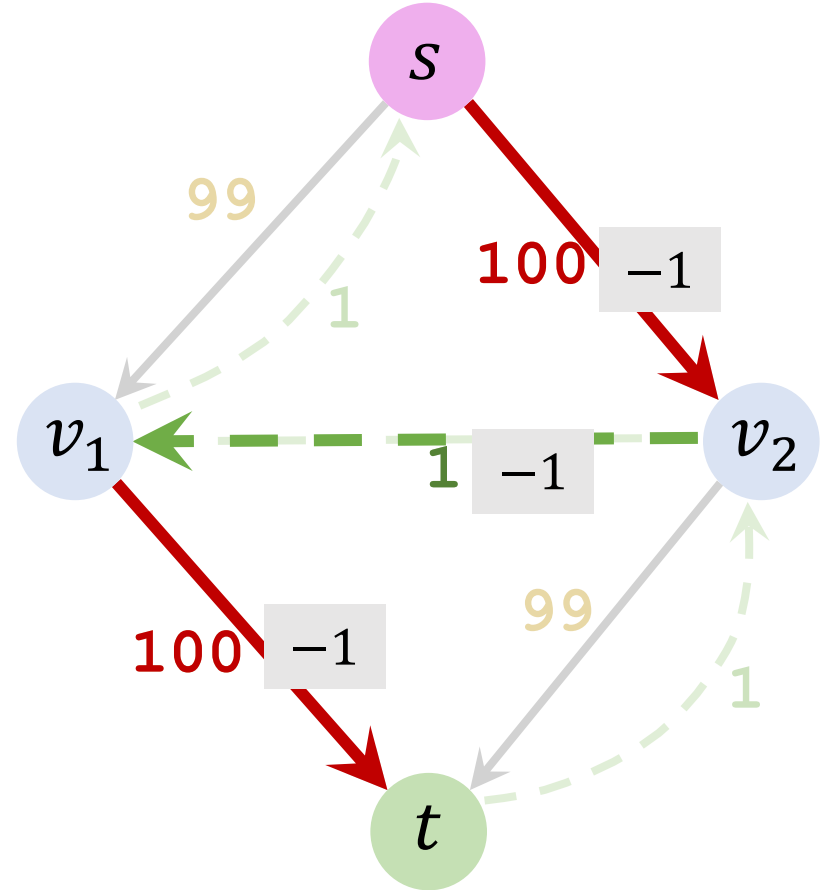
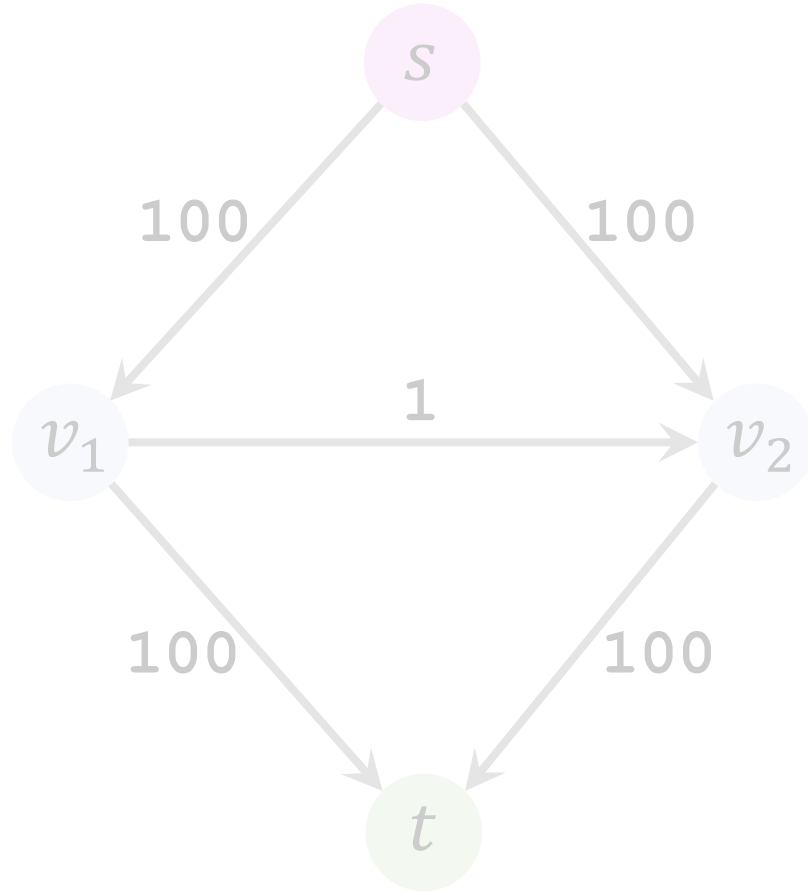


Iteration 2: Find an augmenting path



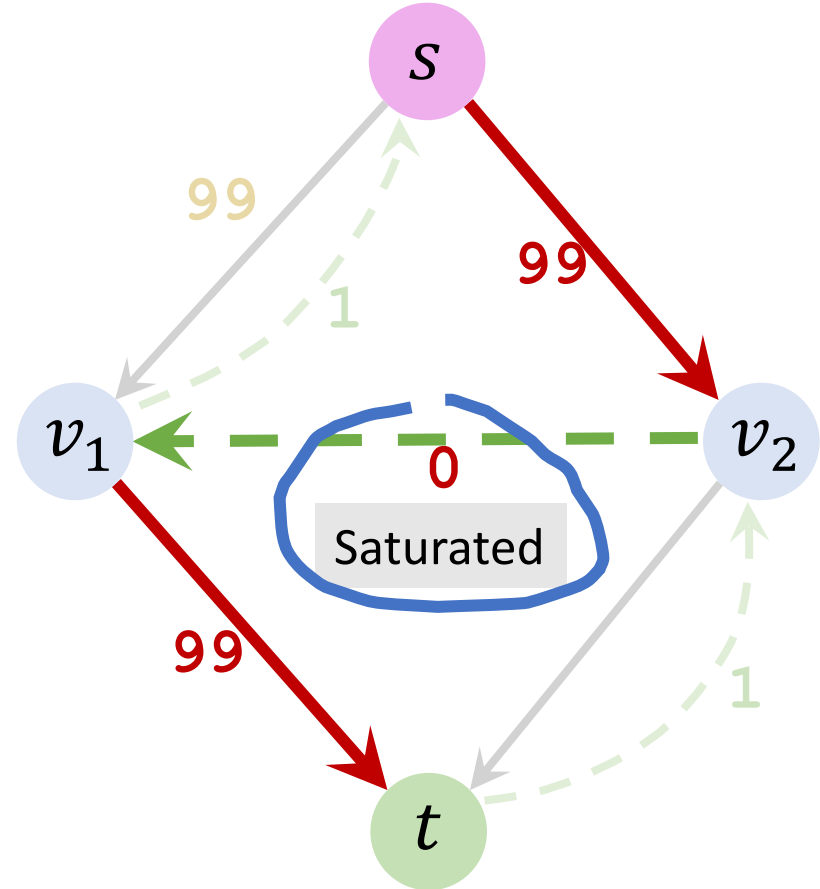
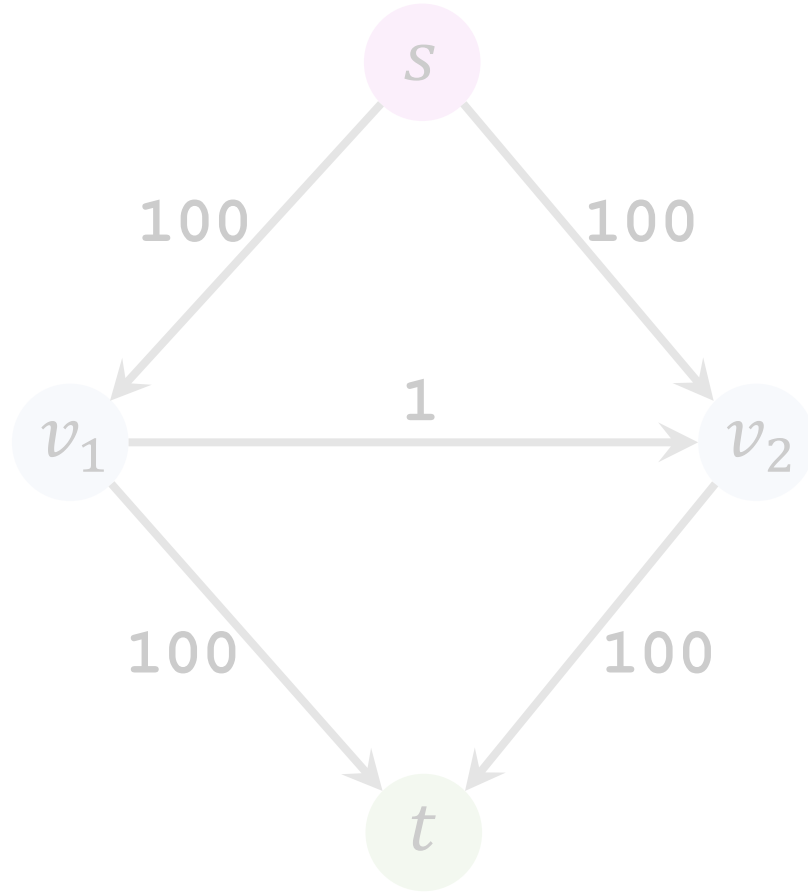
Found path $s \rightarrow v_2 \rightarrow v_1 \rightarrow t$. (Bottleneck capacity = 1.)

Iteration 2: Update residuals



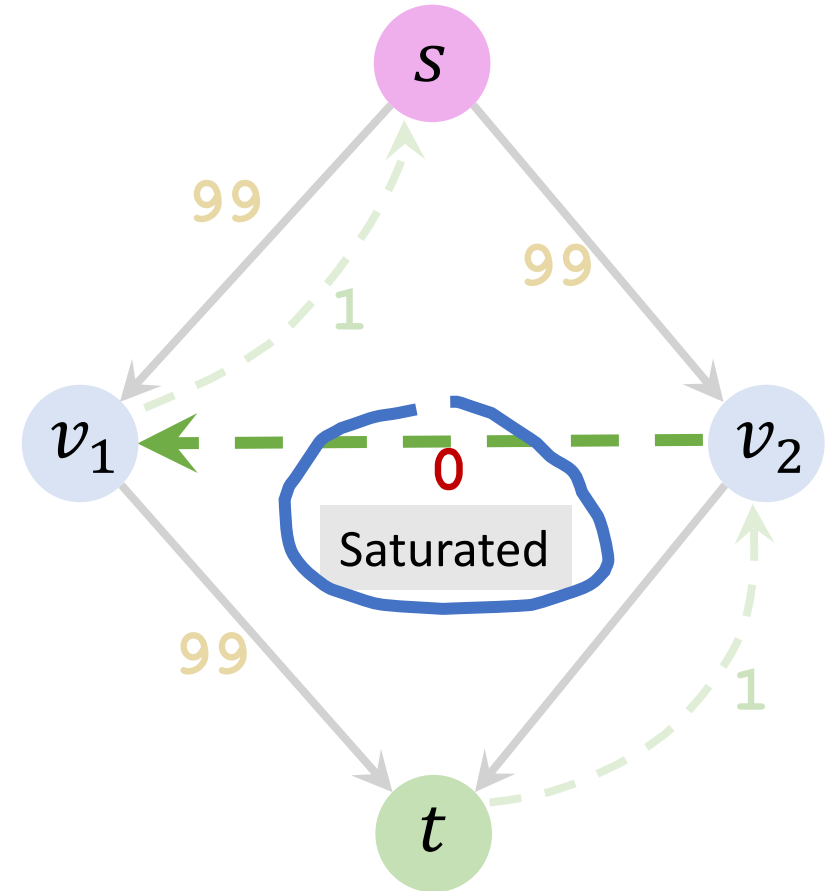
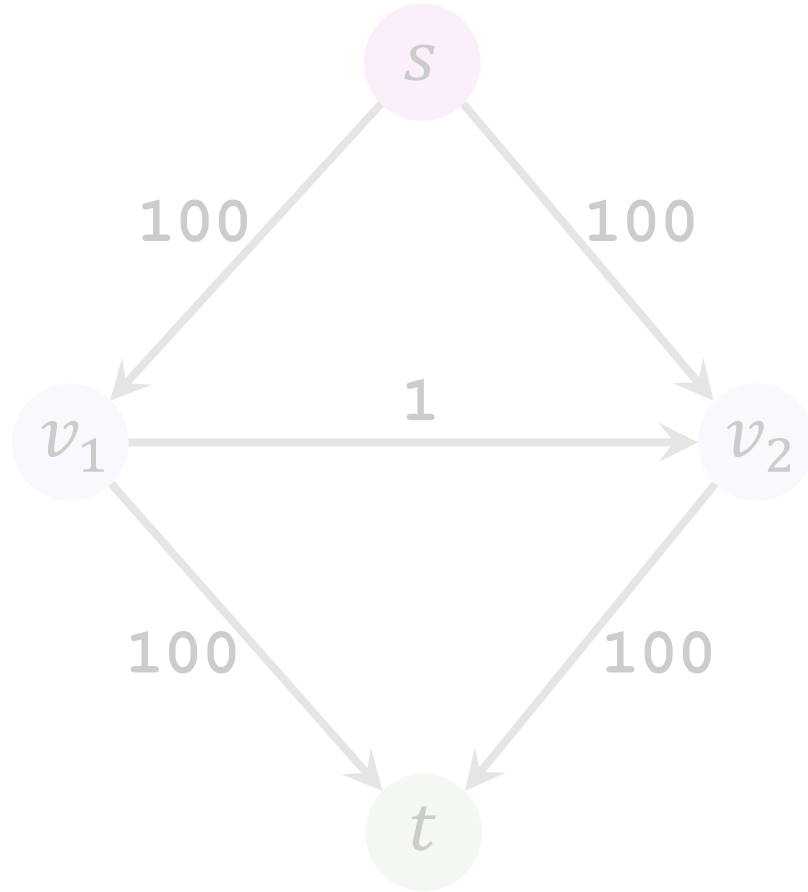
Found path $s \rightarrow v_2 \rightarrow v_1 \rightarrow t$. (Bottleneck capacity = 1.)

Iteration 2: Update residuals

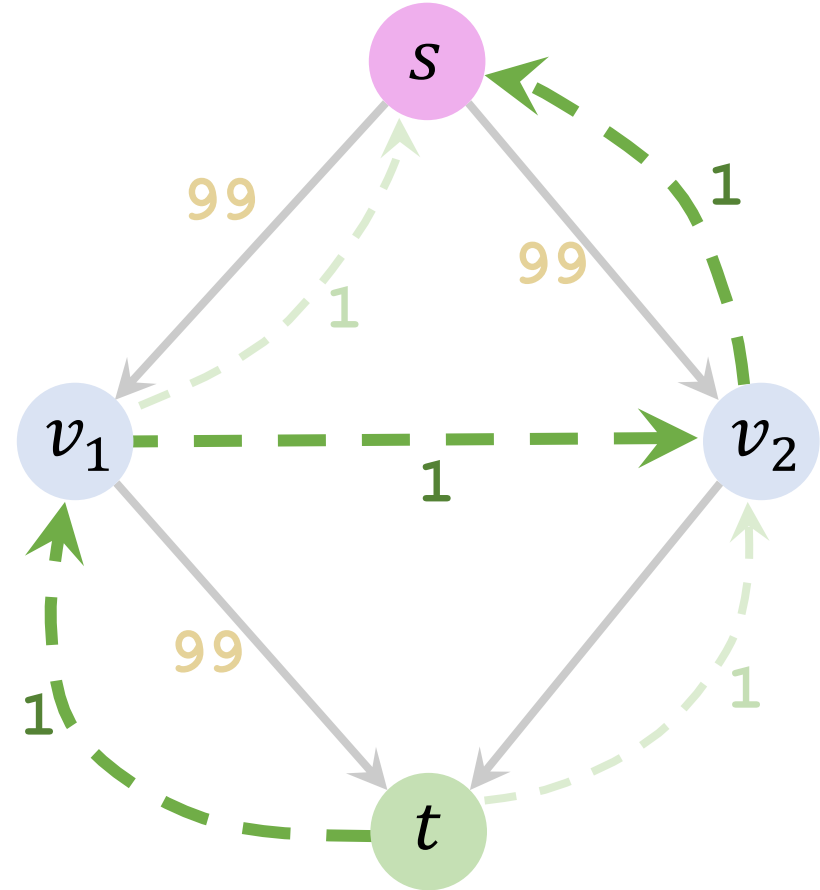
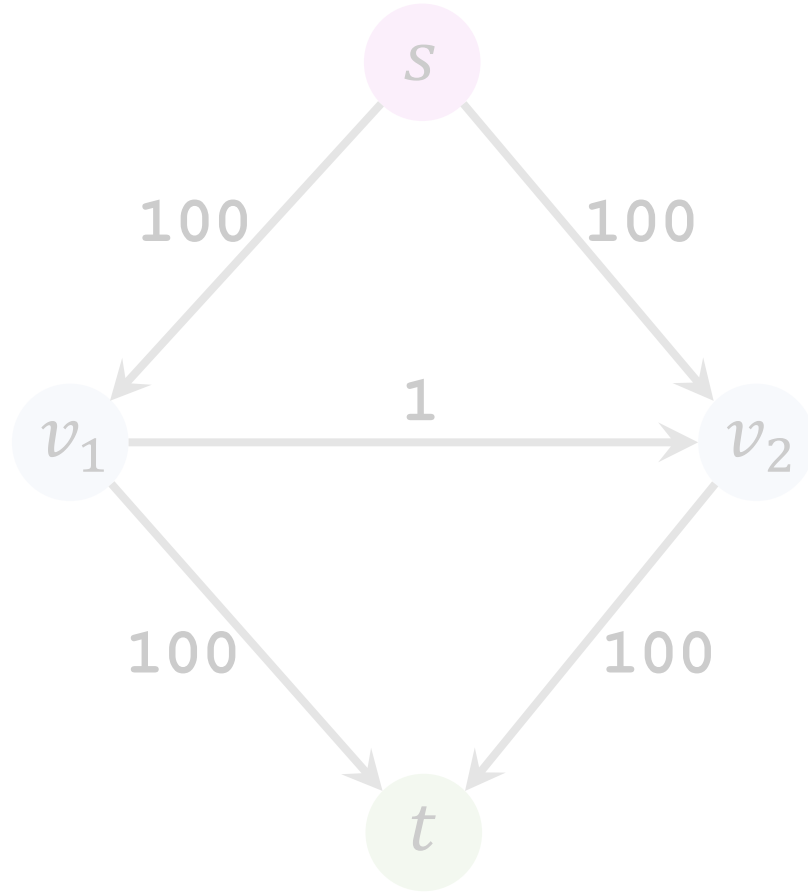


Found path $s \rightarrow v_2 \rightarrow v_1 \rightarrow t$. (Bottleneck capacity = 1.)

Iteration 2: Remove saturated edges

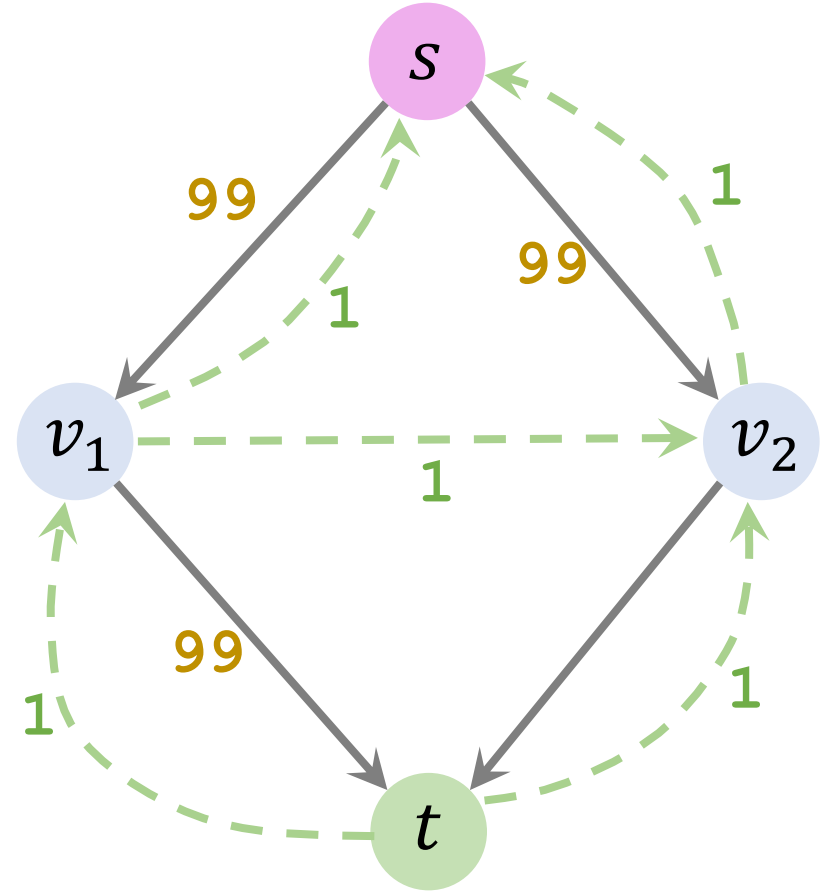
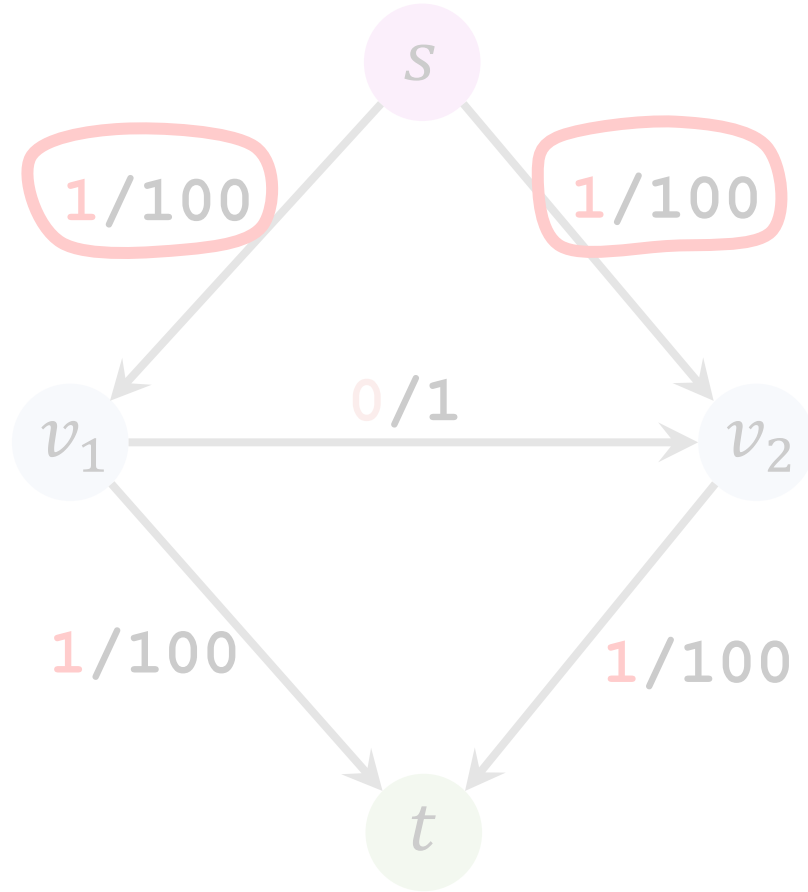


Iteration 2: Add backward path

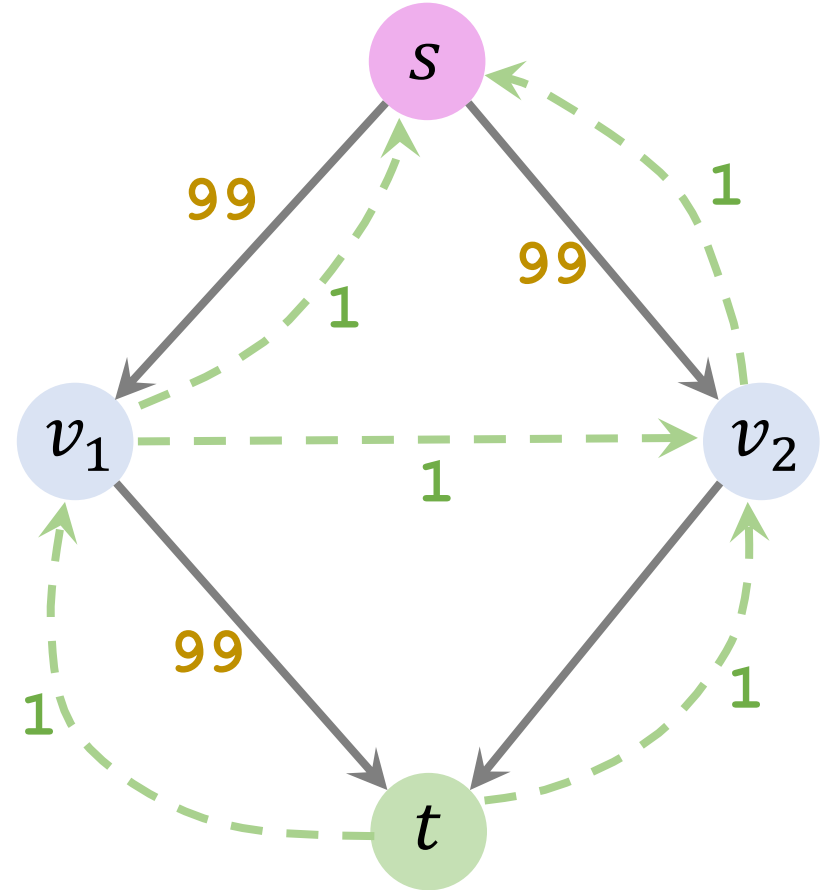
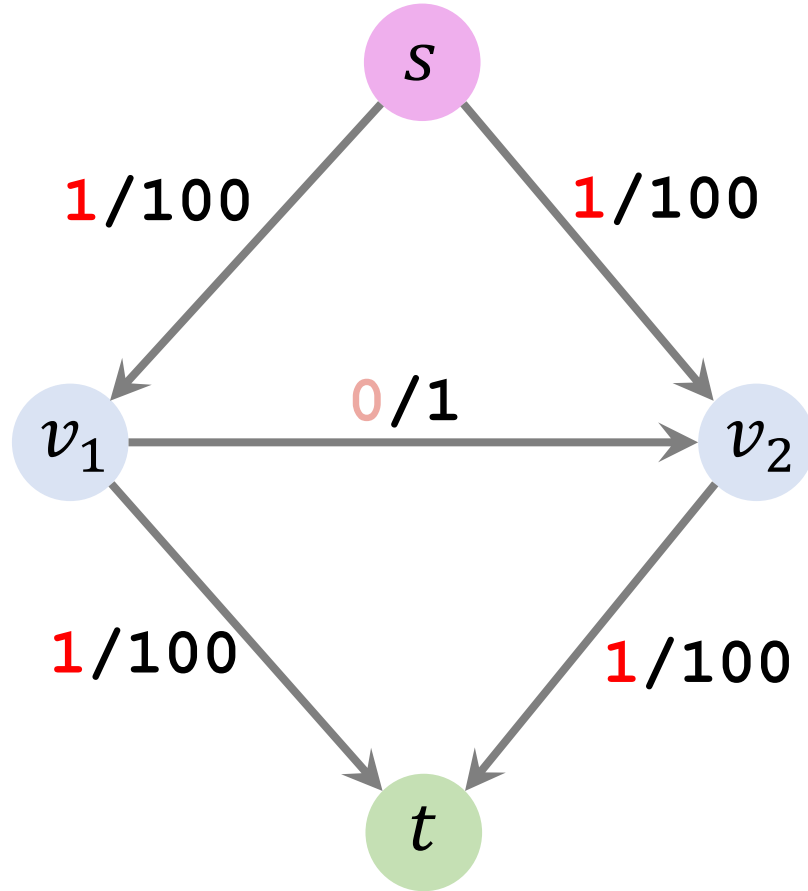


Add backward path $t \rightarrow v_1 \rightarrow v_2 \rightarrow s$ with capacity = 1.

Now, the amount of flow is 2



Slow improvement...



In every iteration, the amount of flow increases by **1**.

Worst-Case Iteration Complexity

- Each iteration increases the amount of flow by at least 1.
 - Thus, $\text{\#Iterations} \leq \text{MaxFlow}$.
-
- In our example, each iteration increases the flow by only 1.
 - Thus, $\text{\#Iterations} = \text{MaxFlow}$.
-
- In sum, the worst-case \#Iterations is equal to MaxFlow .

Worst-Case Time Complexity

- Let m be the number of edges.
- It takes $O(m)$ time to find a path in unweighted graph.
(Ignore the weights in the residual graph.)
- Thus, the per-iteration time complexity is $O(m)$.

Worst-Case Time Complexity

- Let m be the number of edges.
 - It takes $O(m)$ time to find a path in unweighted graph.
(Ignore the weights in the residual graph.)
 - Thus, the per-iteration time complexity is $O(m)$.
-
- Let the maximum flow be f .
 - The worst-case time complexity is $O(f \cdot m)$.
 - (In practice, the time complexity is not so bad.)

Summary

Ford-Fulkerson Algorithm

1. Build a **residual graph**; initialize the residuals to the capacities.

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to the capacities.
2. While augmenting path can be found:
 - a. Find an augmenting path (on the residual graph.)
 - b. Find the bottleneck capacity x on the augmenting path.
 - c. Update the residuals. ($\text{residual} \leftarrow \text{residual} - x$.)
 - d. **Add a backward path.** (Along the path, all edges have weights of x .)

Ford-Fulkerson Algorithm

1. Build a residual graph; initialize the residuals to the capacities.
2. While augmenting path can be found:
 - a. Find an augmenting path (on the residual graph.)
 - b. Find the bottleneck capacity x on the augmenting path.
 - c. Update the residuals. ($\text{residual} \leftarrow \text{residual} - x$.)
 - d. Add a backward path. (Along the path, all edges have weights of x .)

Time complexity: $O(f \cdot m)$. (f is the max flow; m is #edges.)

Thank You!

<http://wangshusen.github.io/>