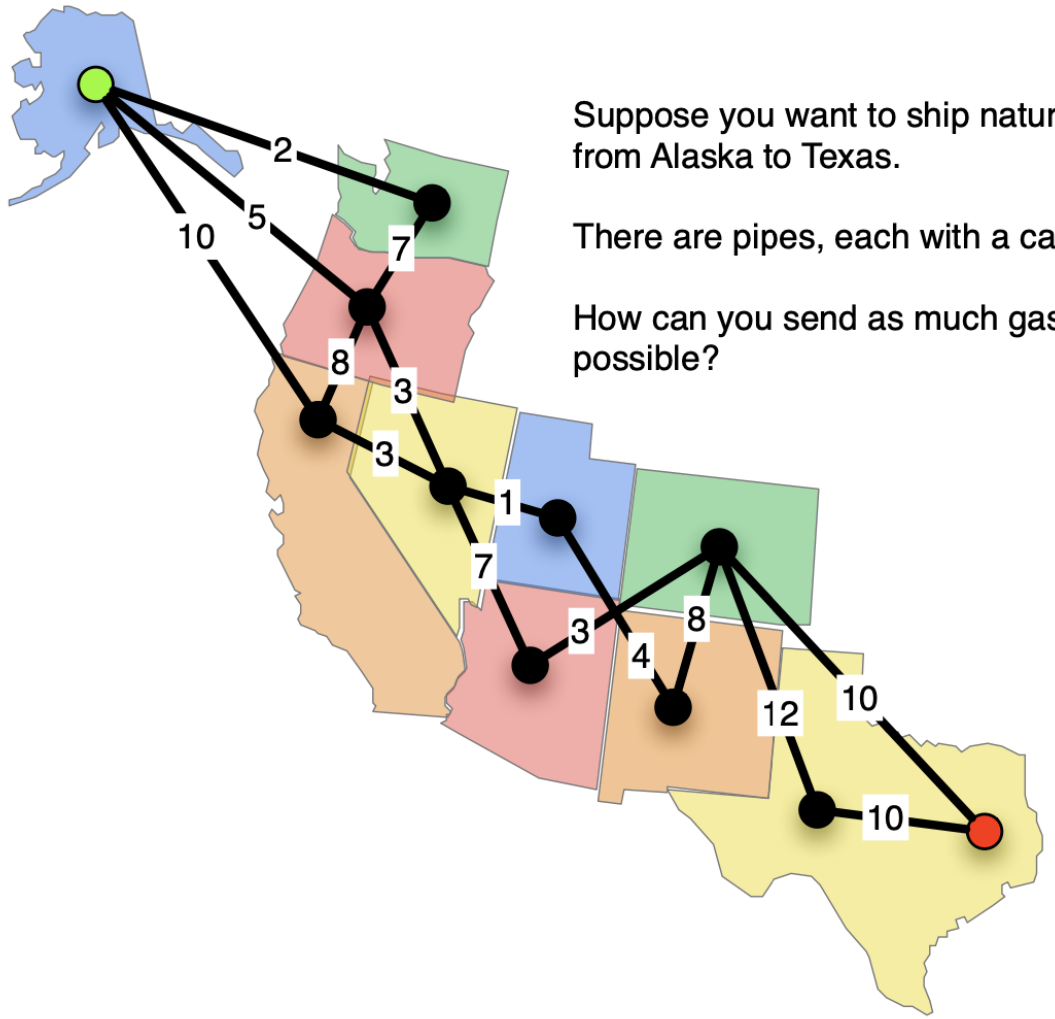
香港科技大学(广州)
THE HONG KONG
UNIVERSITY OF SCIENCE AND
TECHNOLOGY (GUANGZHOU)

Design and Analysis of Algorithms

Jing Tang | DSAA 2043 Fall 2024

Network Flow

Network Flow Problem e.g.



Suppose you want to ship natural gas from Alaska to Texas.

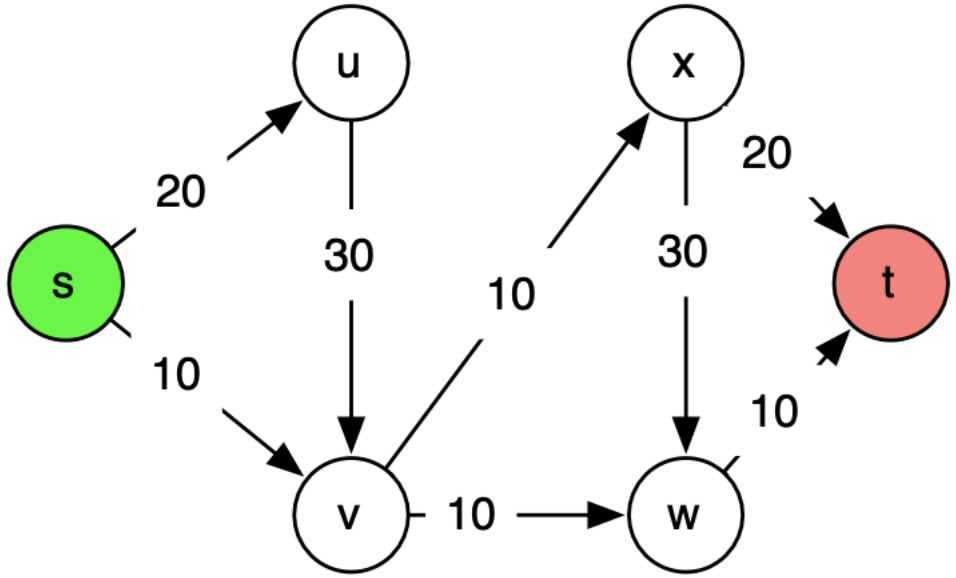
There are pipes, each with a capacity.

How can you send as much gas as possible?

Flow Network

A flow network is a connected, directed graph G = (V,E).

- Each edge e has a non-negative, integer capacity c_e.
- A single source s ∈ V.
- A single sink t ∈ V.
- No edge enters the source and no edge leaves the sink.

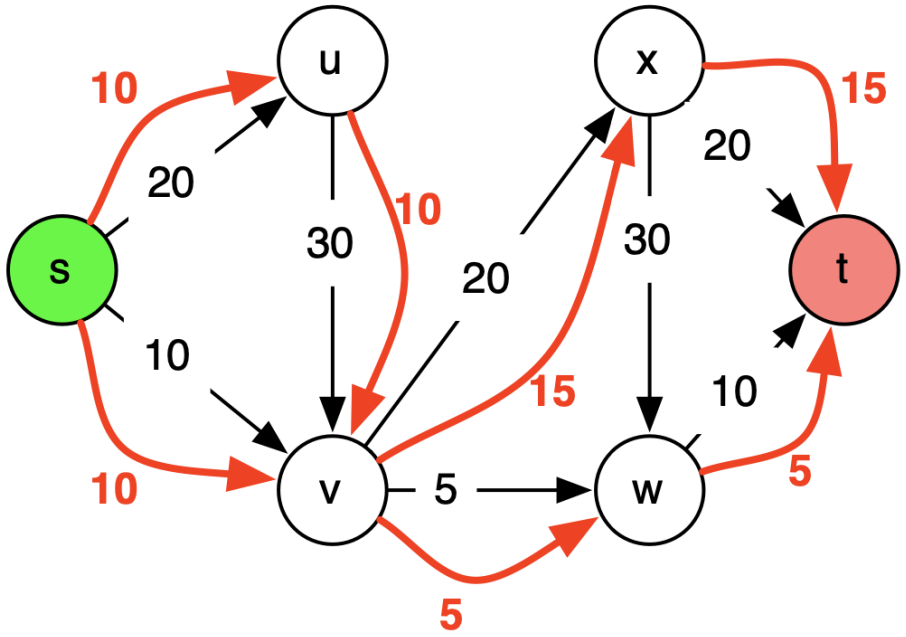


Assumptions

- To repeat, we make these assumptions about the network:
  - Capacities are integers.
  - Every node has one edge adjacent to it.
  - No edge enters the source and no edge leaves the sink.
- These assumptions can all be removed.

Flow

- Def. An s-t flow is a function f: E → $R_{\geq 0}$ that assigns a real number to each edge.
- Intuitively, f(e) is the amount of material carried on the edge e.



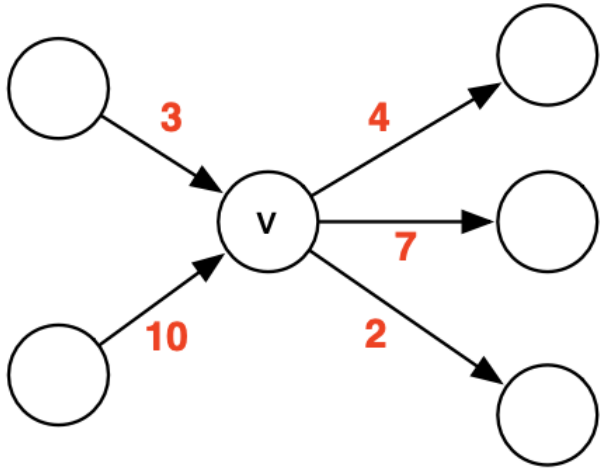
Flow Constraints

Constraints on f:

- $0 \le f(e) \le c_e$ for each edge e. (capacity constraints)
- For each node v except s and t, we have:

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ leaving } v} f(e)$$

(balance constraints: whatever flows in, must flow out).



Notation

- The value of flow f is:

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

This is the amount of material that s is able to send out.

- Notation:
  - $f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$
  - $f^{\text{out}}(v) = \sum_{e \text{ leaving } v} f(e)$
- Balance constraints becomes: $f^{in}(v) = f^{out}(v)$ for all $v \notin \{s,t\}$

Maximum Flow Problem

- Definition (Value)

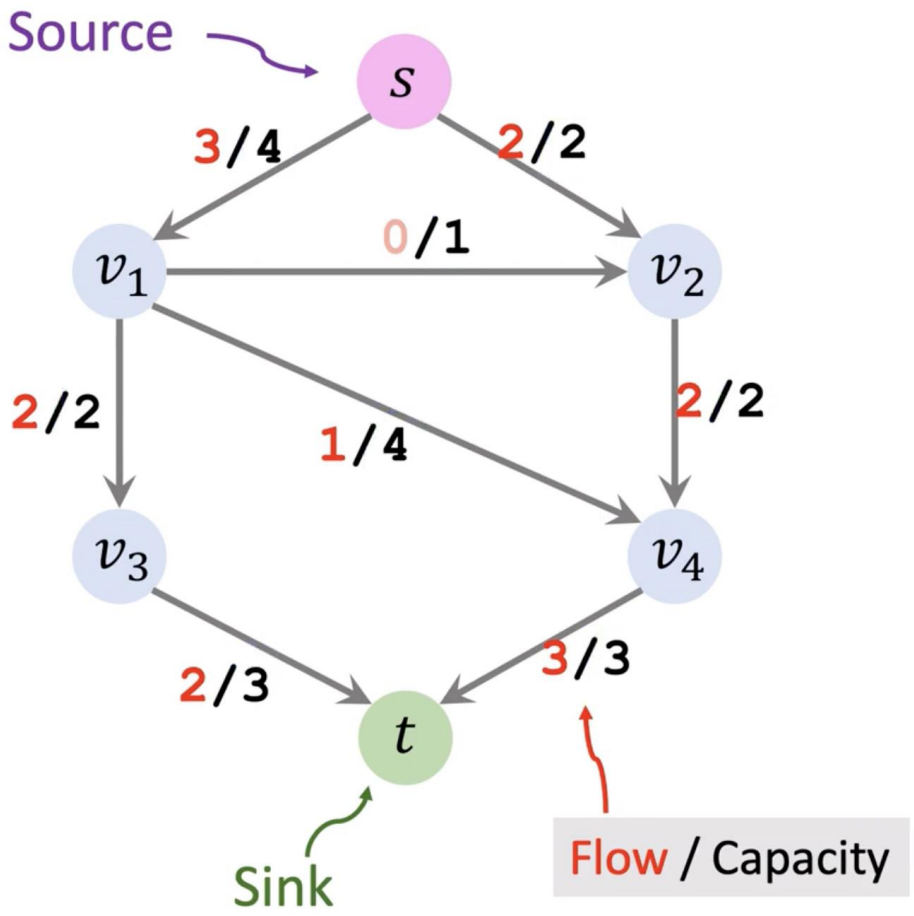
  The value v(f) of a flow f is $f^{\text{out}}(s)$.

  – That is: it is the amount of material that leaves s.

- **Maximum Flow Problem**

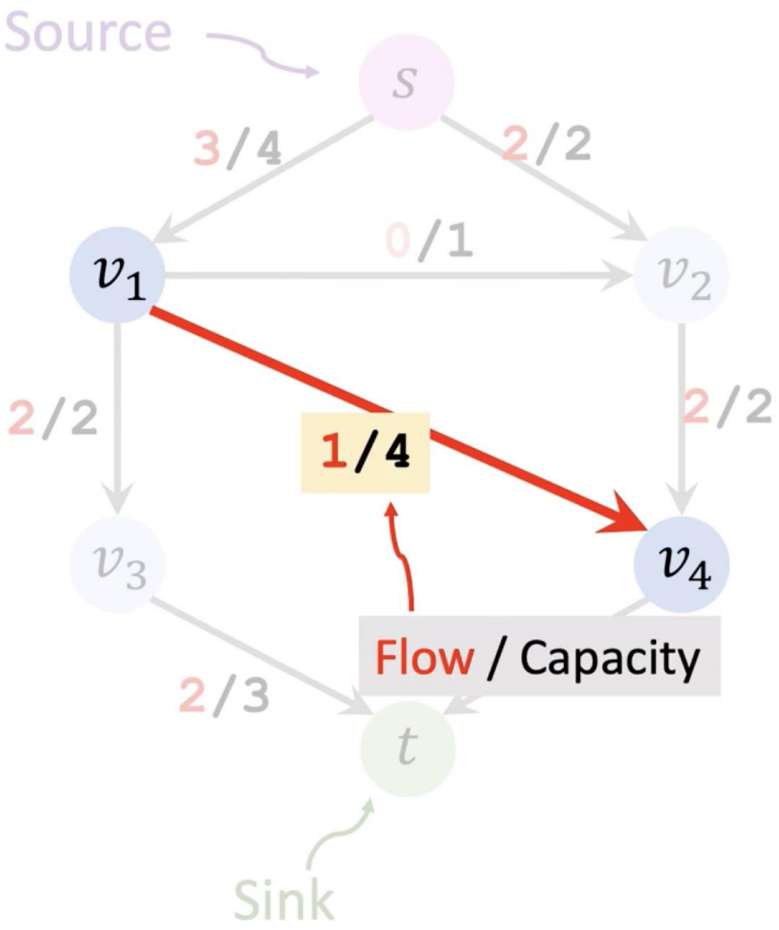
  Given a flow network G, find a flow f of maximum possible value.

Example



- Send water from the source s to the sink t.
- The edges are pipes which have certain capacities, e.g., $4\ m^3/s$.
- How much water can flow from source s to the sink t at most?

Example



- Capacity of the red pipe is $4\ m^3/s$.
- A flow of $1\ m^3/s$ goes through the red pipe.
- It has a residual of $3\ m^3/s$.

A Naïve Start

- Suppose we let f(e) = 0 for all edges (no flow anywhere).

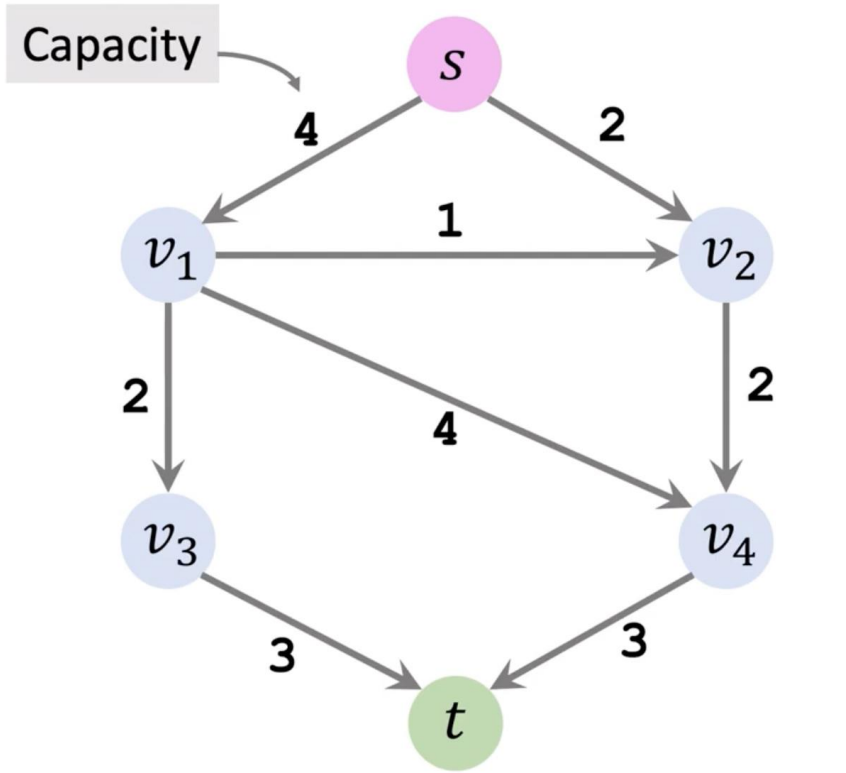
- Choose some s-t path and “push” flow along it up to the capacities.

  Repeat.

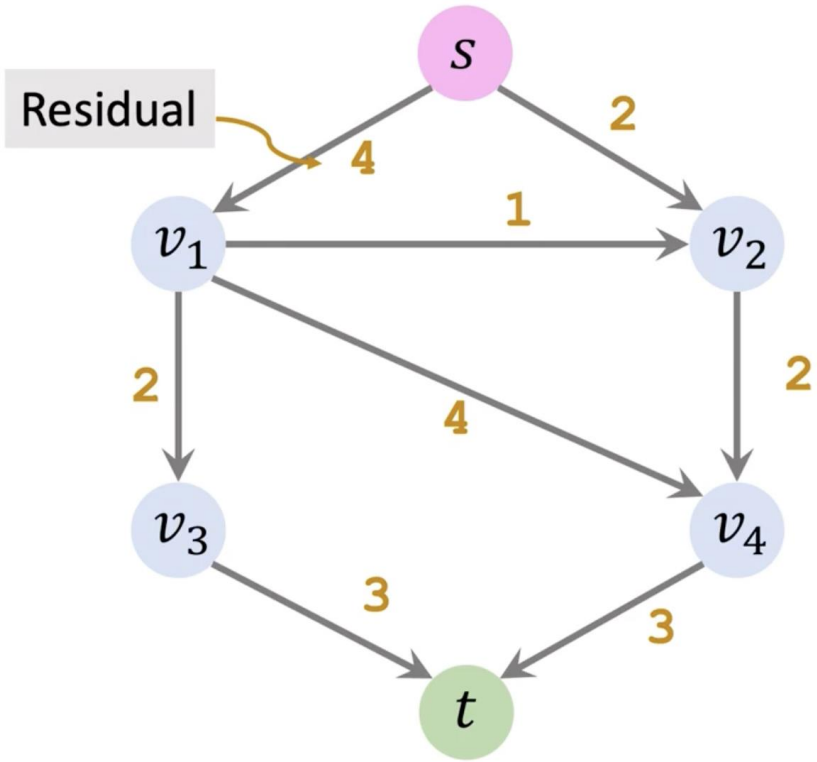
Augmenting Paths

- Let P be an s−t path in the residual graph G_f.
- Let bottleneck(P, f) be the smallest capacity in G_f on any edge of P.
- If bottleneck (P, f) > 0 then we can increase the flow by sending bottleneck(P, f) along the path P.

Naïve Example



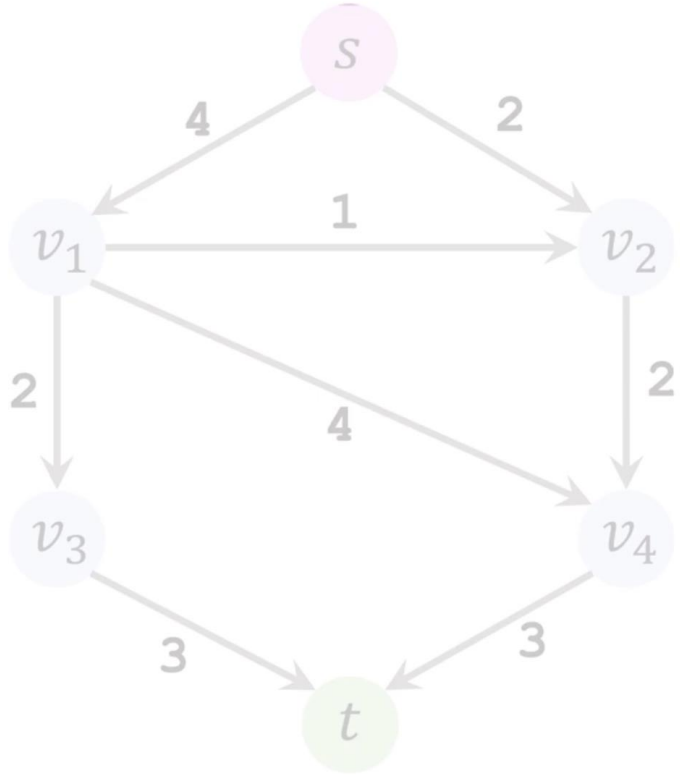
Original Graph

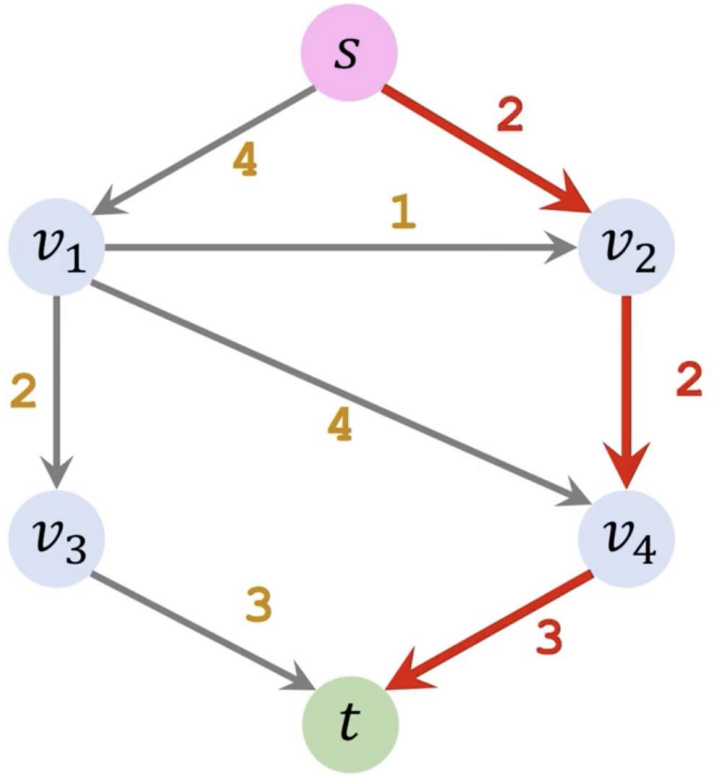


Residual Graph

Naïve Example

Iteration 1: Find an augmenting path

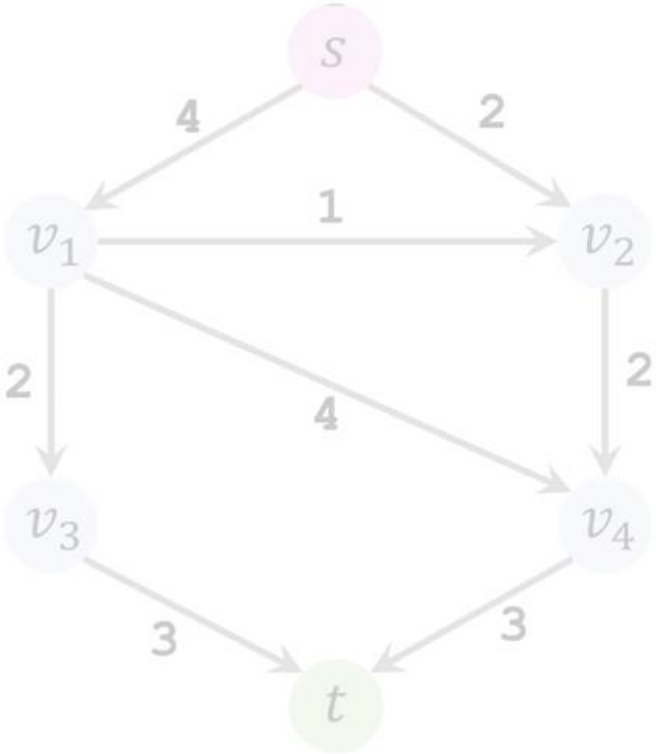


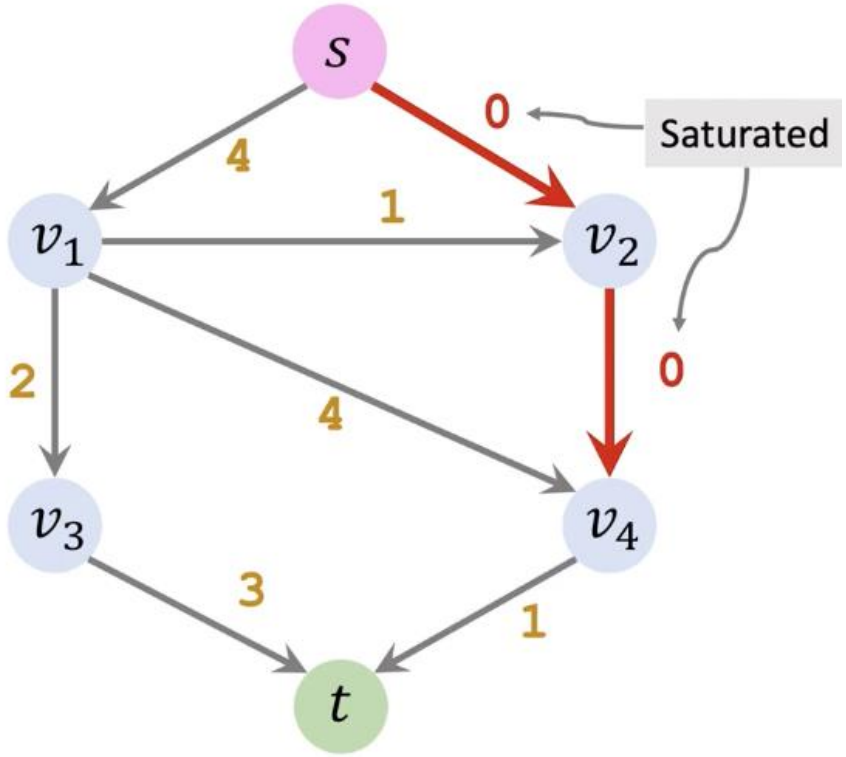


Found path $s \to v_2 \to v_4 \to t$. (Bottleneck capacity $= 2$.)

Naïve Example

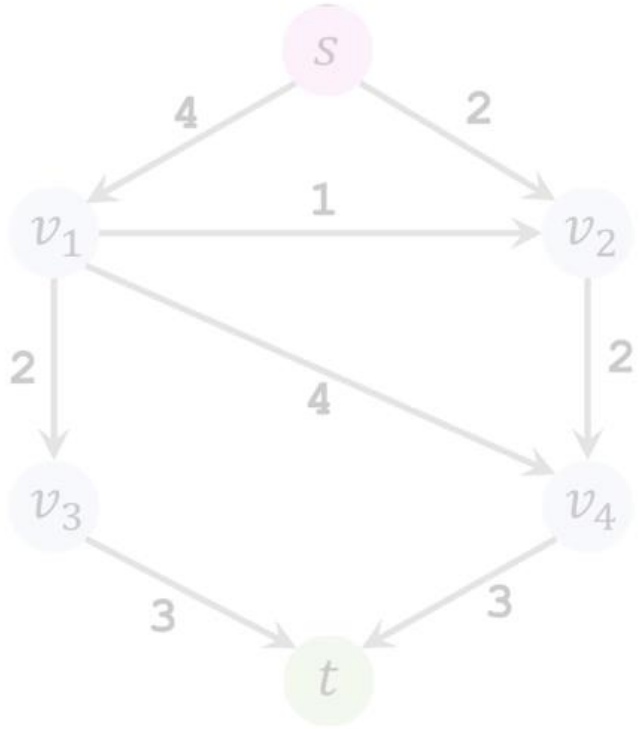
Iteration 1: Remove saturated edges

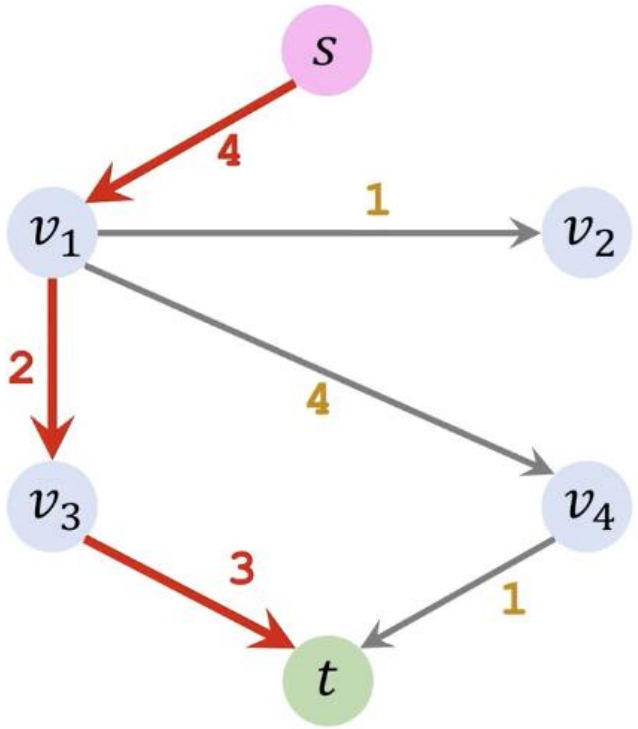




Naïve Example

Iteration 2: Find an augmenting path

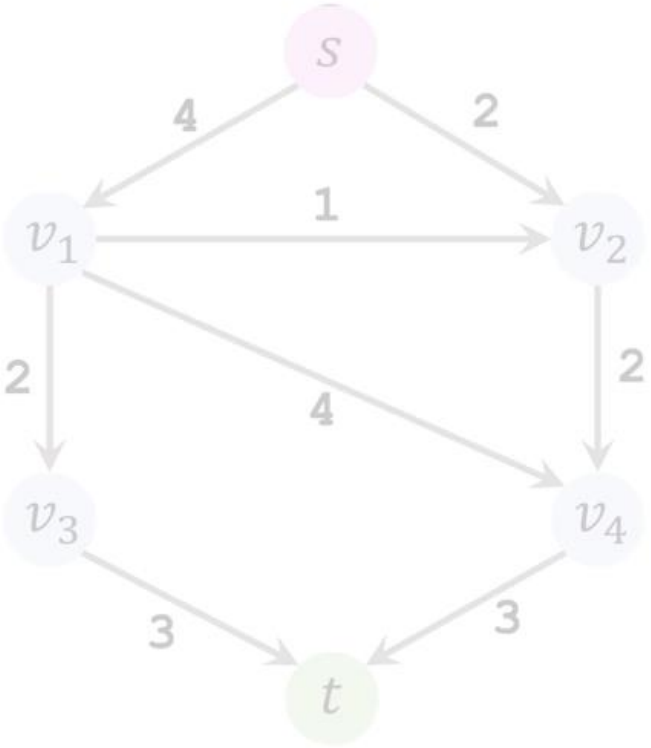


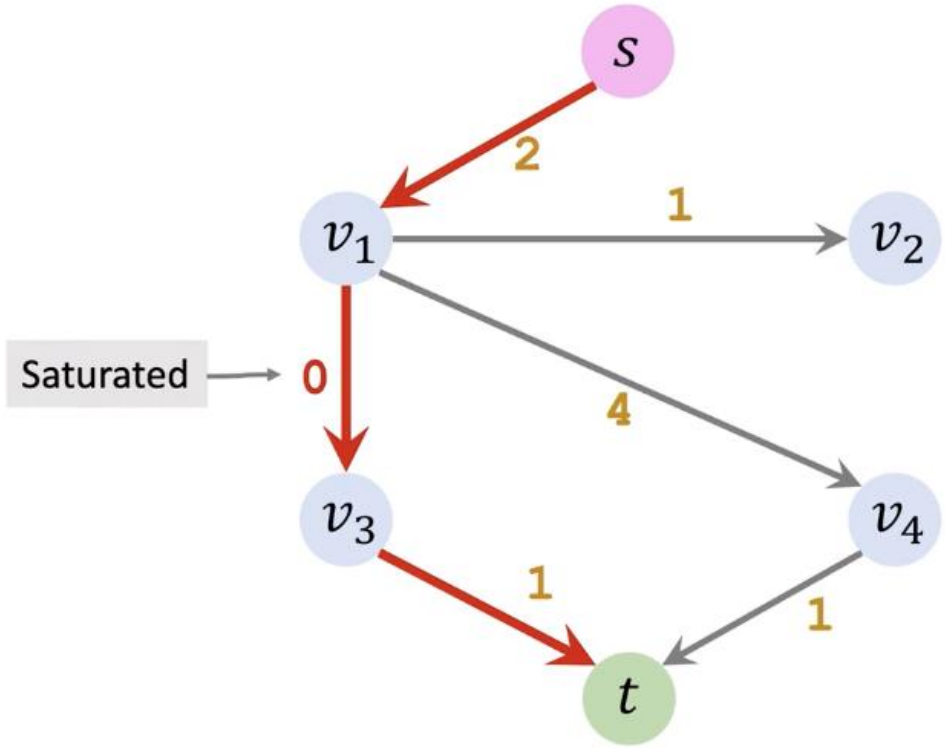


Found path $s \to v_1 \to v_3 \to t$. (Bottleneck capacity $= 2$.)

Naïve Example

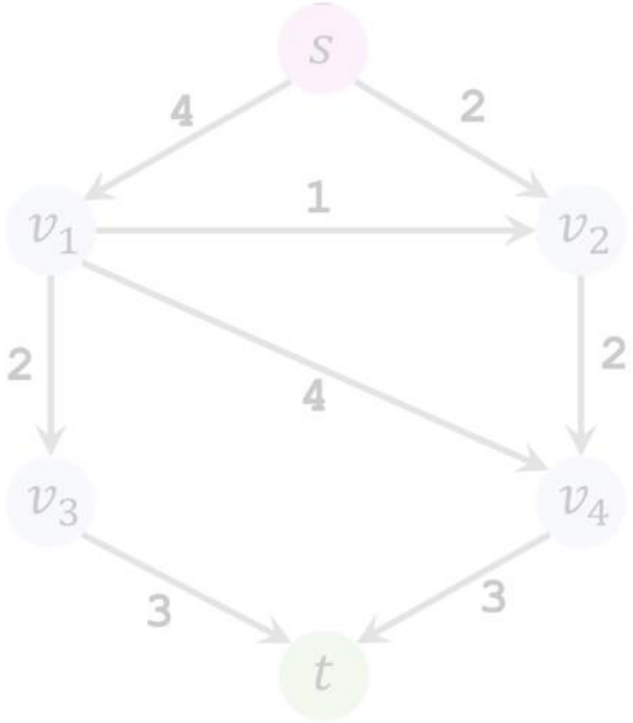
Iteration 2: Remove saturated edges

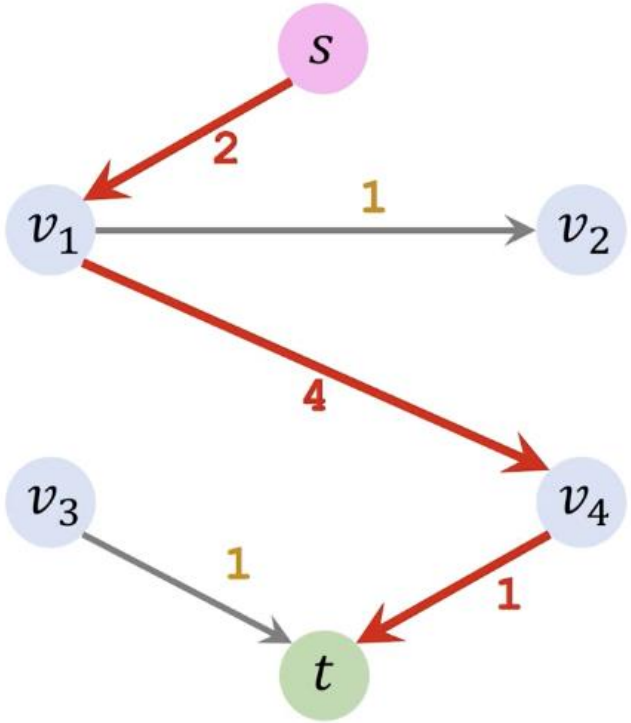




Naïve Example

Iteration 3: Find an augmenting path

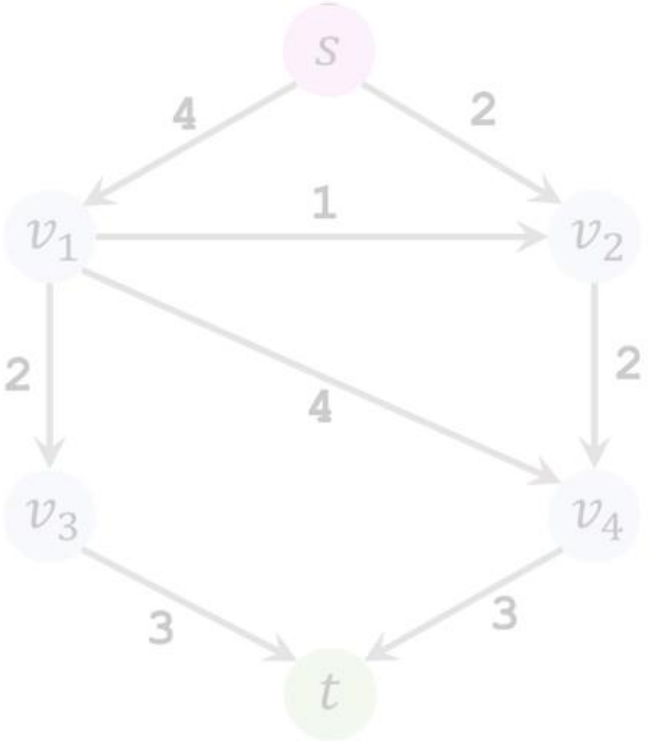


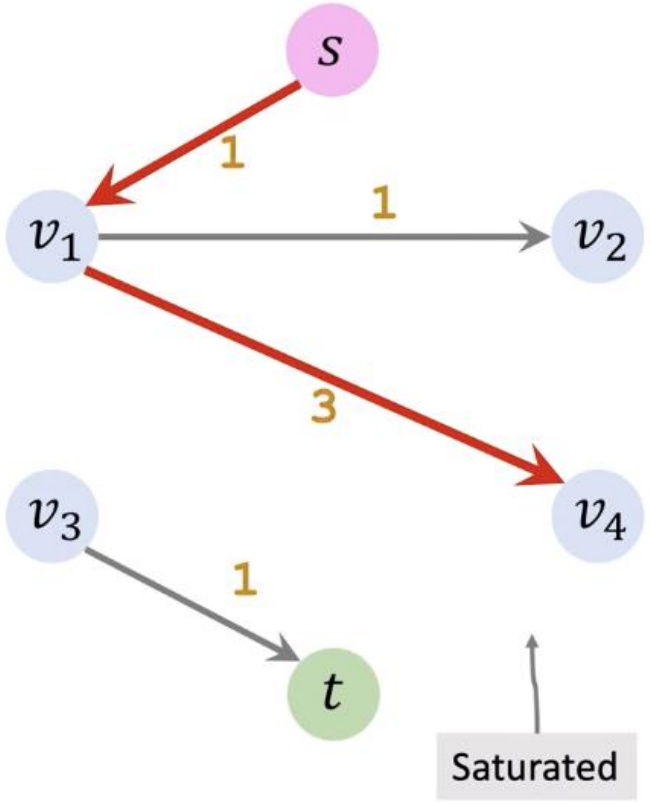


Found path $s \to v_1 \to v_4 \to t$. (Bottleneck capacity $= 1$.)

Naïve Example

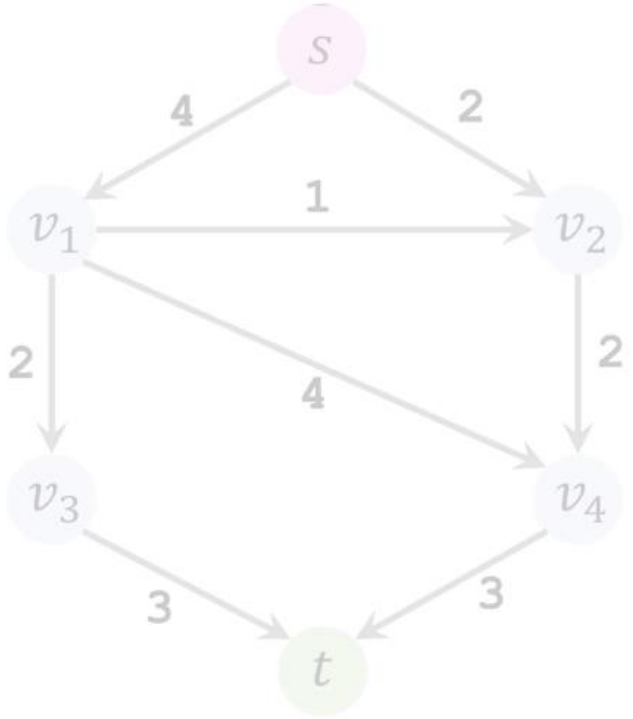
Iteration 3: Remove saturated edges

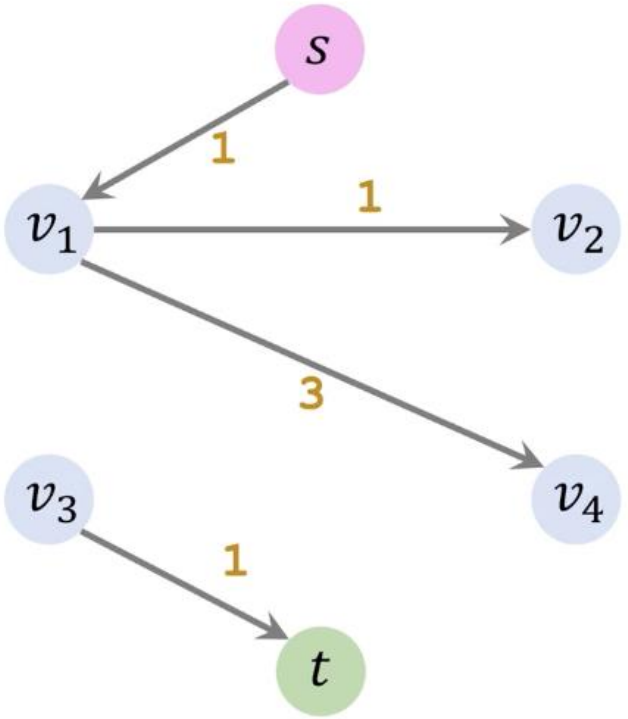




Naïve Example

Iteration 4: Find an augmenting path

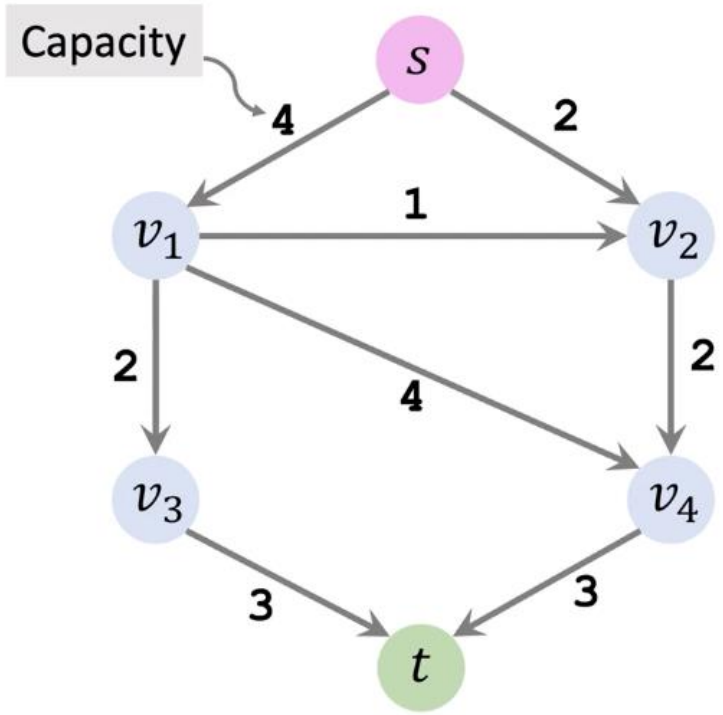


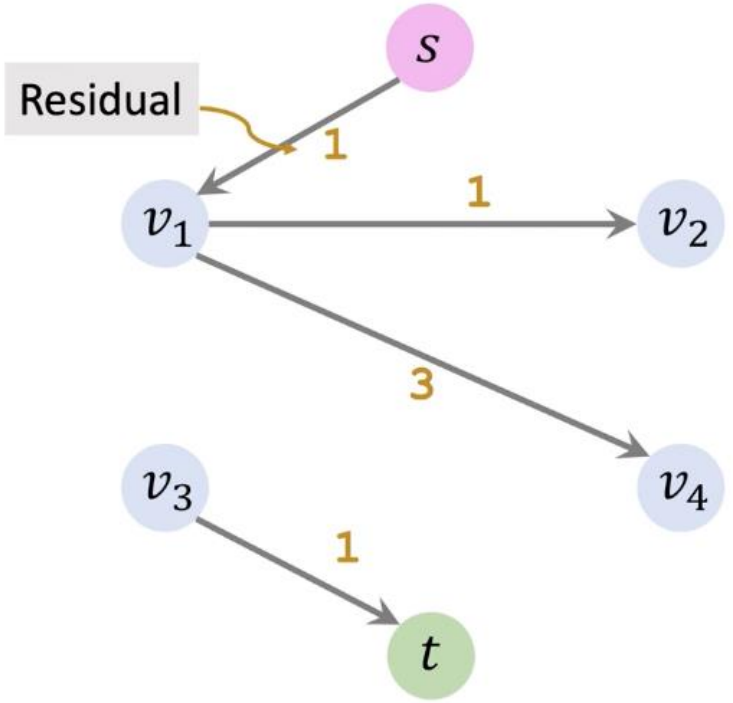


Cannot find any path from source to sink.

Naïve Example

End of Procedure





Flow = Capacity − Residual.

Naïve Example

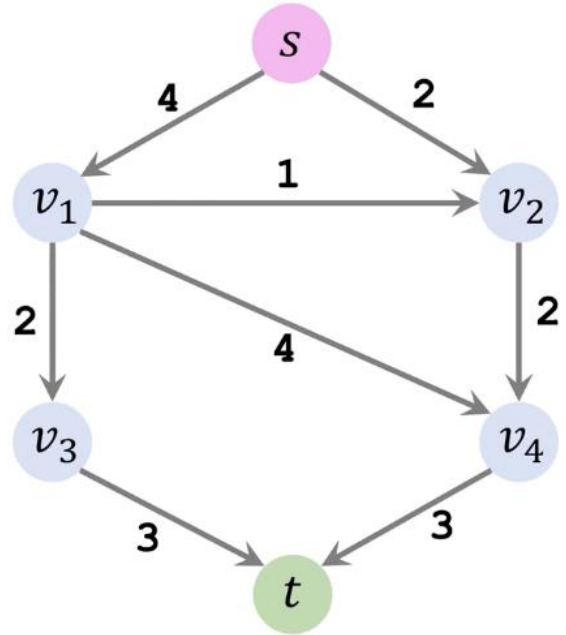
Is the naïve approach always correct?

Problem: Once a bad approach is selected, the naïve approach cannot make any corrections!

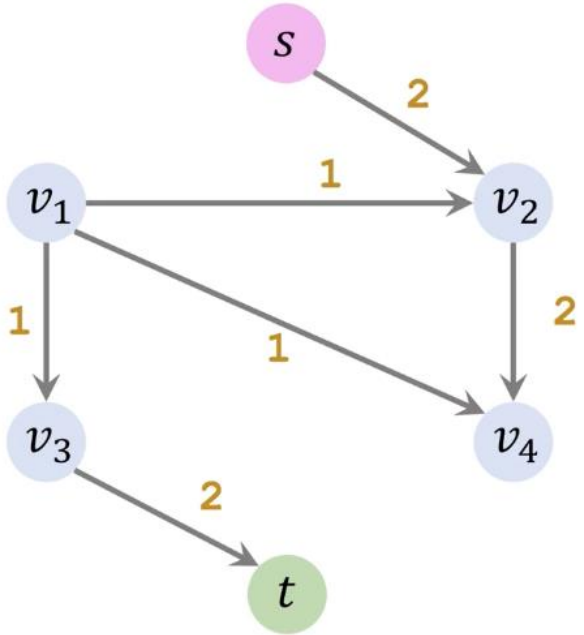
Naïve Example

Consider choosing paths in the order of $\{s, v_1, v_4, t\}$, $\{s, v_1, v_3, t\}$, then we can get the final state as,

End of Procedure



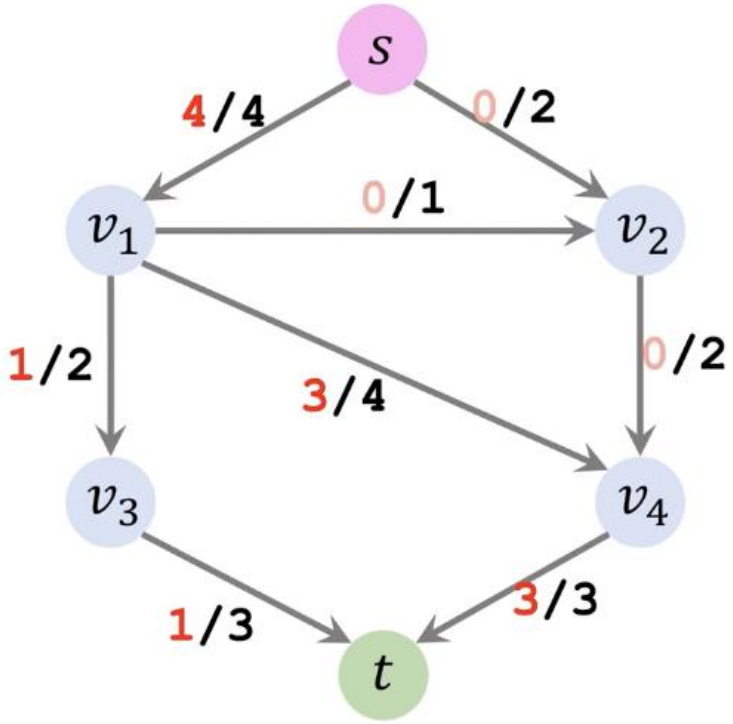
Original Graph



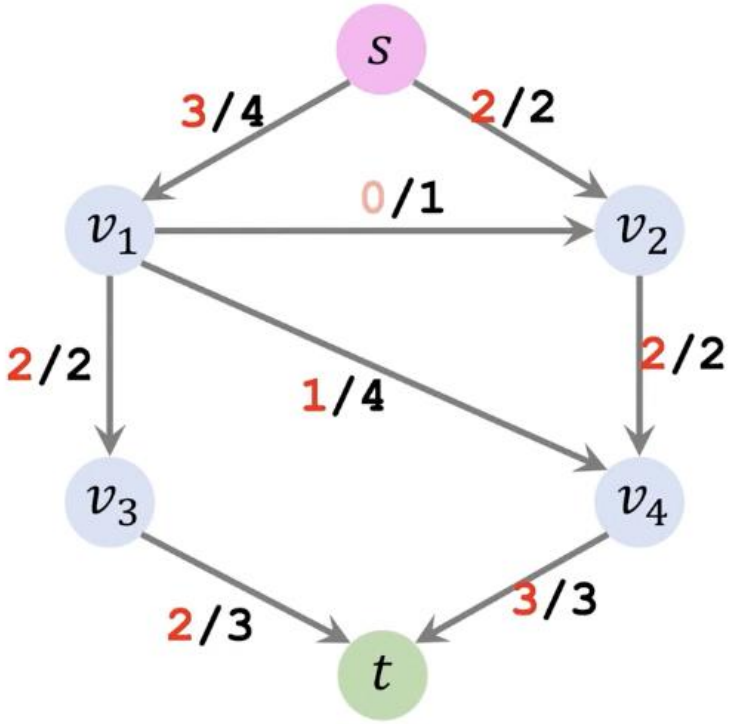
Residual Graph

Naïve Example

The result is not maximum flow!



Flow = 4. (Not maximum!)



Flow = 5. (Maximum!)

Ford-Fulkerson Algorithm

A Greedy Approach

- Suppose we let f(e) = 0 for all edges (no flow anywhere).
- Choose some s-t path and “push” flow along it up to the capacities.

  Repeat.
- When we get stuck, we can erase some flow along certain edges.

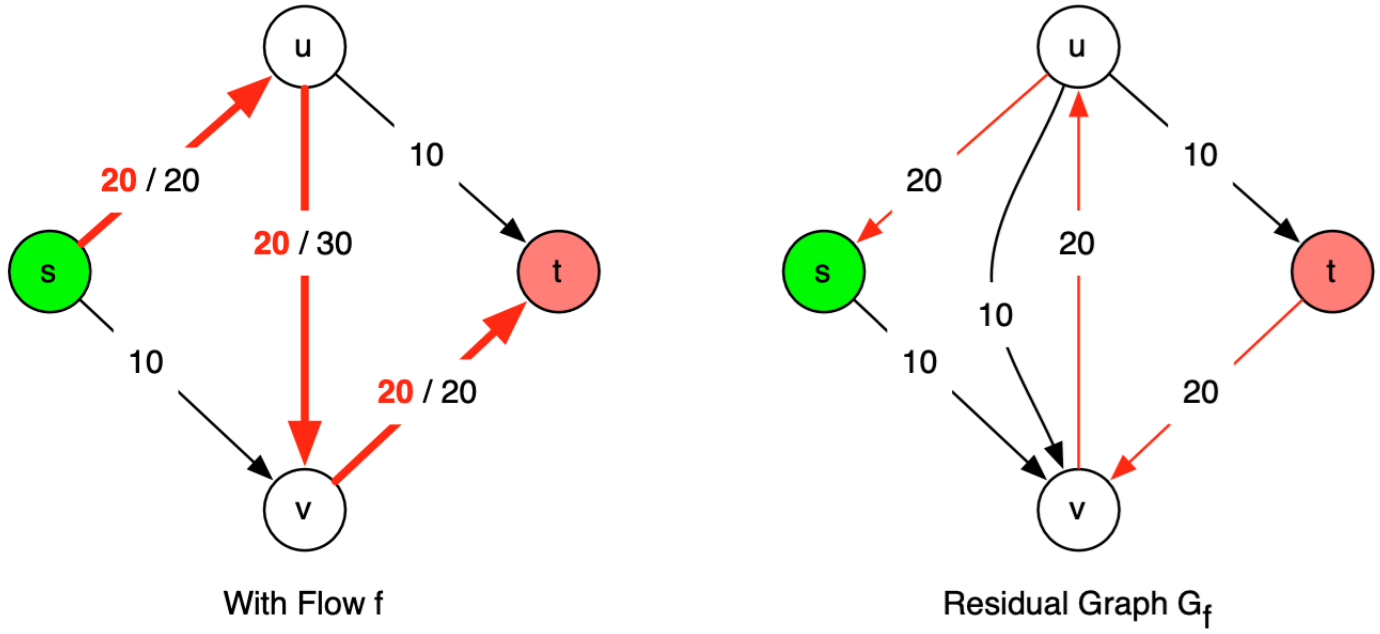
How do we make this more precise?

Residual Graph

- We define a residual graph G_f. G_f depends on some flow f:
  - G_f contains the same nodes as G.
  - Forward edges: For each edge e = (u,v) of G for which $f(e)<c_e$, include an e' = (u,v) in G_f with capacity $c_e - f(e)$.
  - Backward edges: For each edge e = (u,v) in G with $f(e)>0$, we include an e' = (v,u) in G_f with capacity f(e).

Residual Graph: Example

- If f(e)<c_e, add edge to G_f with capacity $c_e - f(e)$
  - Remaining capacity left
- If f(u,v)>0,add reverse edge (v,u) with capacity f(e)
  - Can erase up to f(e) capacity



Augmenting Paths

- Let P be an s−t path in the residual graph G_f.
- Let bottleneck(P, f) be the smallest capacity in G_f on any edge of P.
- If bottleneck(P, f) > 0 then we can increase the flow by sending bottleneck(P, f) along the path P.

Augmenting Paths

```
augment(f, P):
    b = bottleneck(P,f)
    For each edge (u,v) ∈ P:
        If e = (u,v) is a forward edge:
            Increase f(e) in G by b    //add some flow
        Else:
            e' = (v,u)
            Decrease f(e') in G by b  //erase some flow
        EndIf
    EndFor
    Return f
```

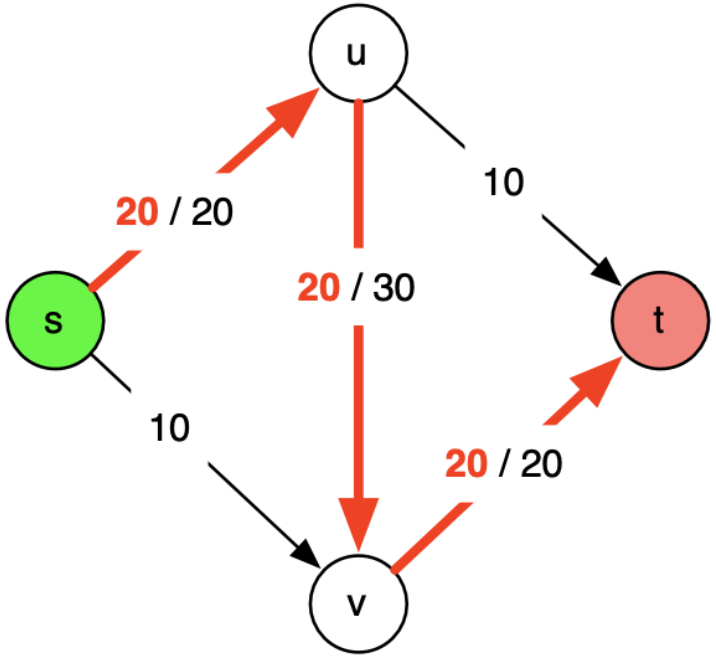
Ford-Fulkerson Algorithm

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

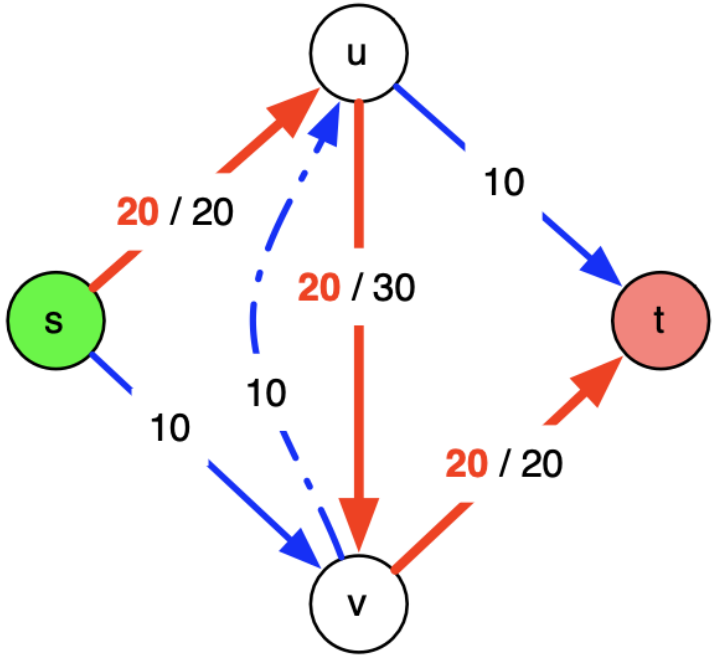
```
MaxFlow(G):
  // initialize:
  Set f[e] = 0 for all e in G

  // while there is an s-t path in Gf:
  While P = FindPath(s,t, Residual(G,f)) != None:
     f = augment(f, P)
     UpdateResidual(G, f)
  EndWhile
  Return f
```

A Glimpse



After 1 path, we've allocated 20 units



Want to send the blue path

After augment, we still have a flow

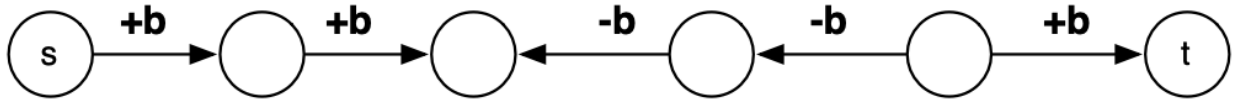
After f′ = augment(P, f), we still have a flow:

Capacity constraints: Let e be an edge on P:

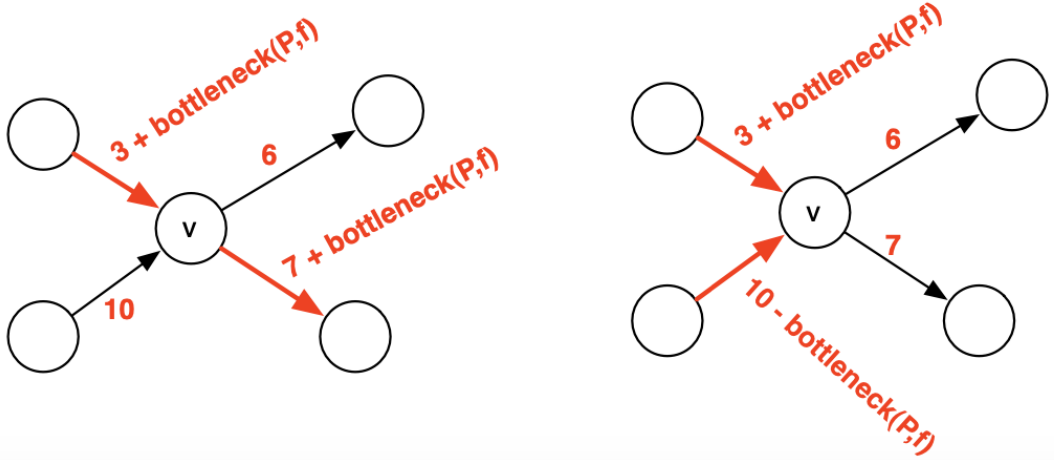
- if e is forward edge, it has capacity $c_e - f(e)$. Therefore,
  - $f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + c_e - f(e) \leq c_e$
- if e is a backward edge, it has capacity f(e). Therefore,
  - $f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$

Still have a flow

- Balance constraints: An s-t path in G_f corresponds to some set of edges in G:



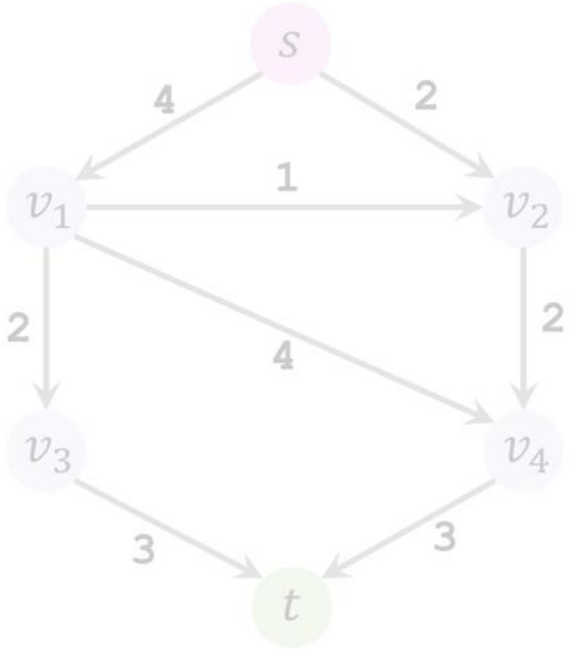
- In other pictures,

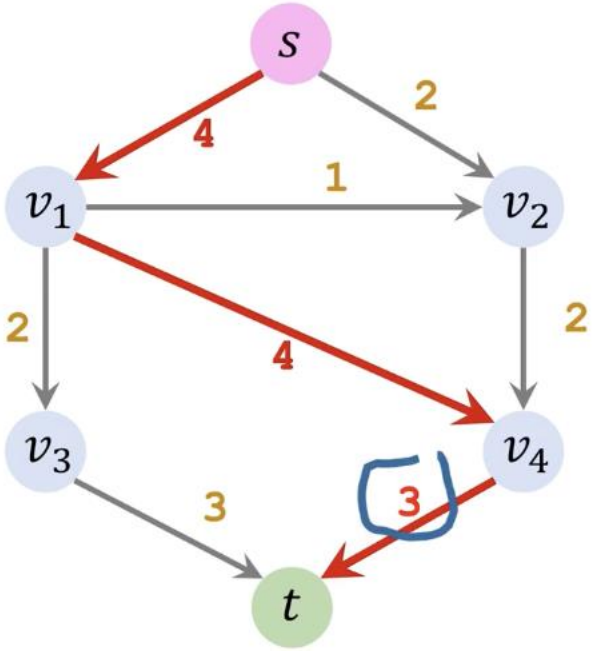


Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

Iteration 1: find an augmenting path

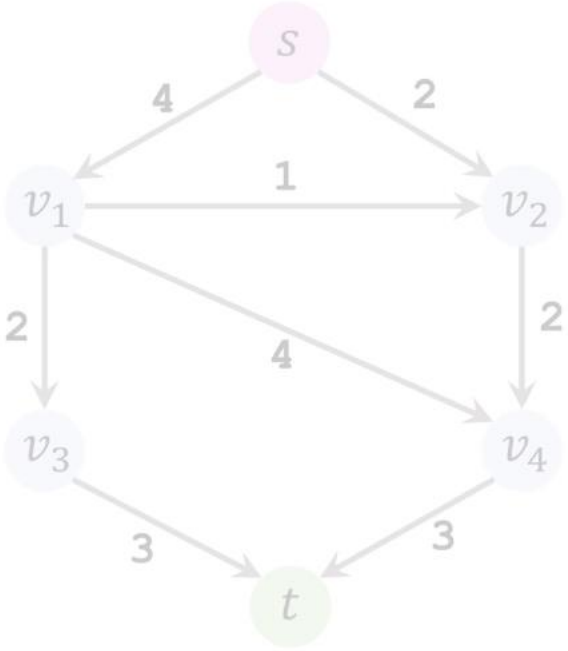


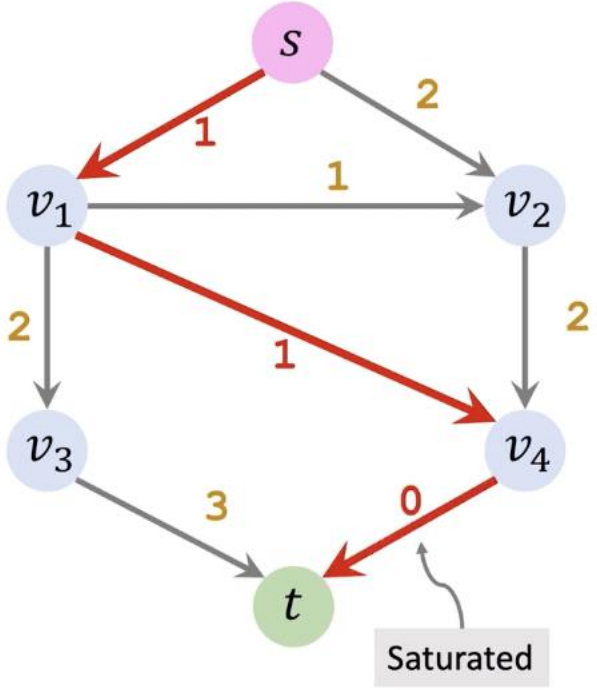


Found path $s \to v_1 \to v_4 \to t$. (Bottleneck capacity $= 3$.)

Ford-Fulkerson Algorithm: Example

Iteration 1: update residual graph (remove saturated and add forward edges)

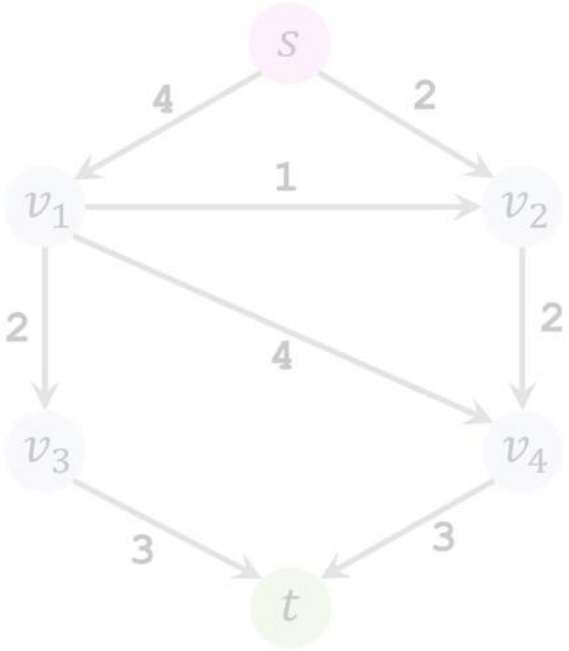


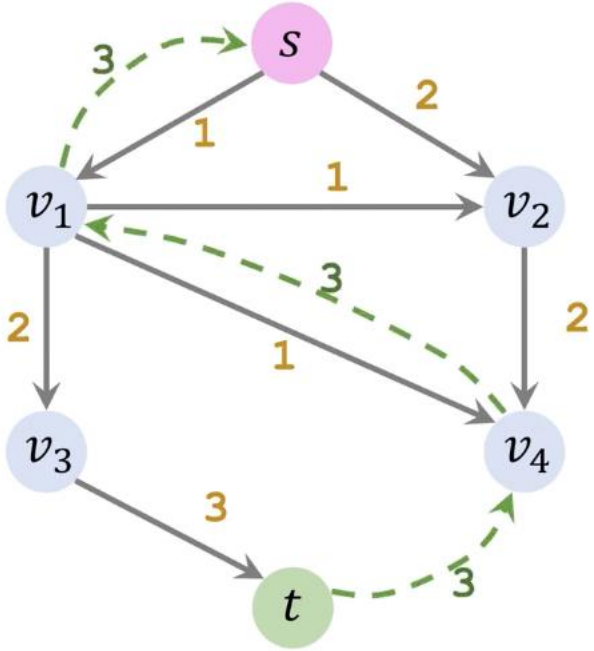


Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

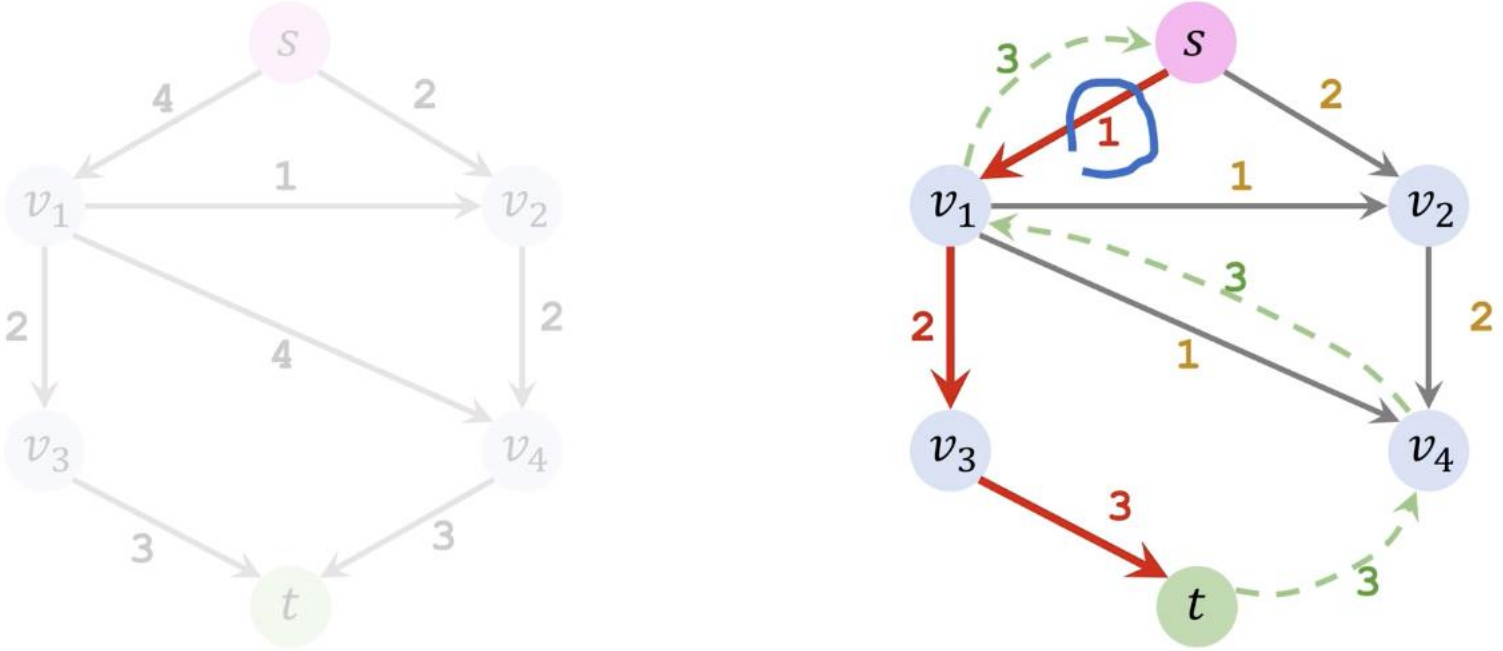
Iteration 1: update residual graph (add backward edges)





Ford-Fulkerson Algorithm: Example

Iteration 2: find an augmenting path

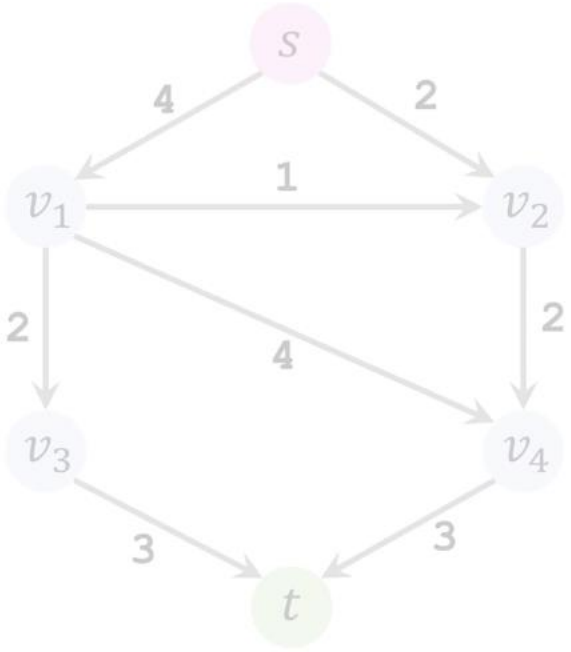


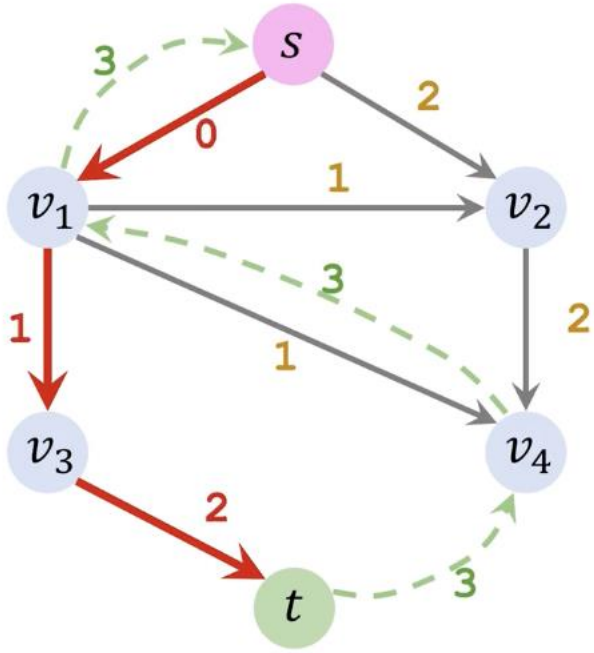
Found path $s \to v_1 \to v_3 \to t$. (Bottleneck capacity $= 1$.)

Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

Iteration 2: update residual graph (remove saturated and add forward edges)

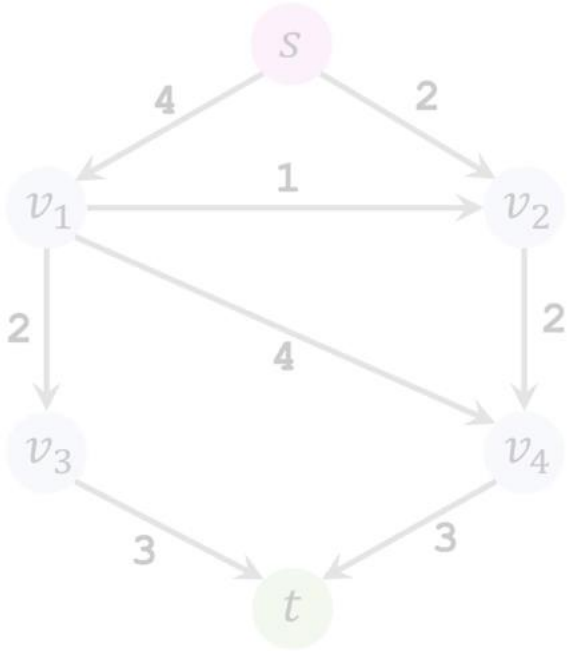


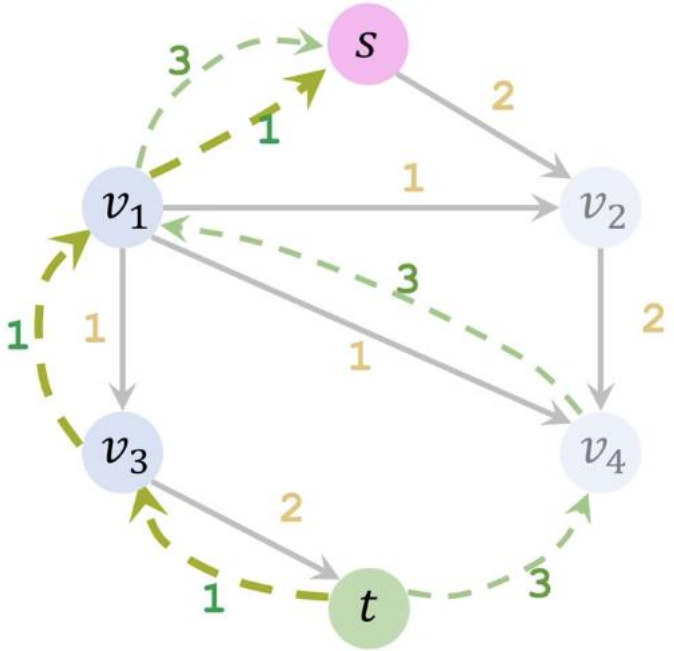


Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

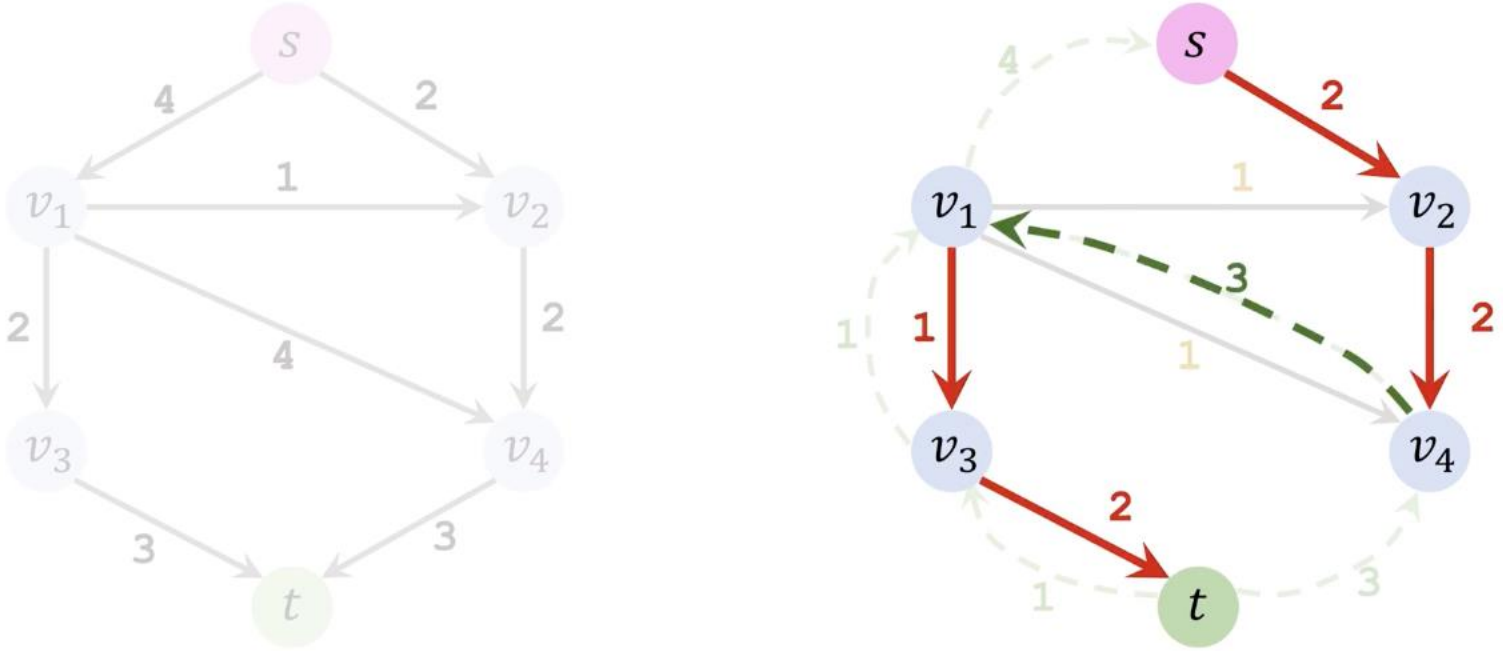
Iteration 2: update residual graph (add backward edges)





Ford-Fulkerson Algorithm: Example

Iteration 3: find an augmenting path

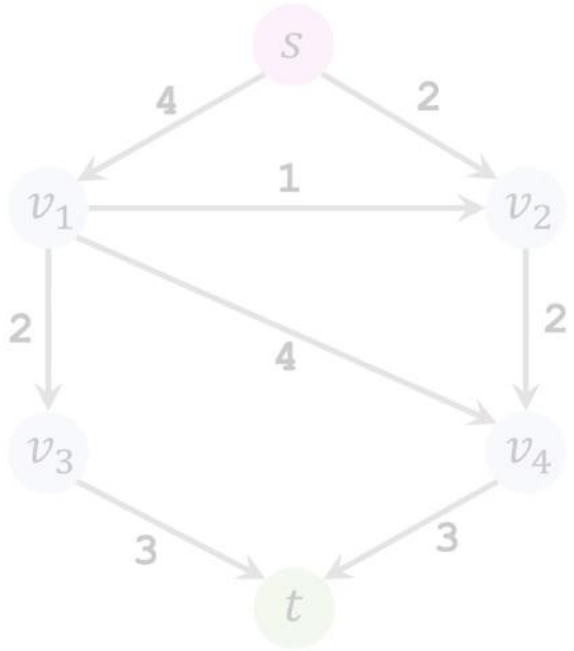


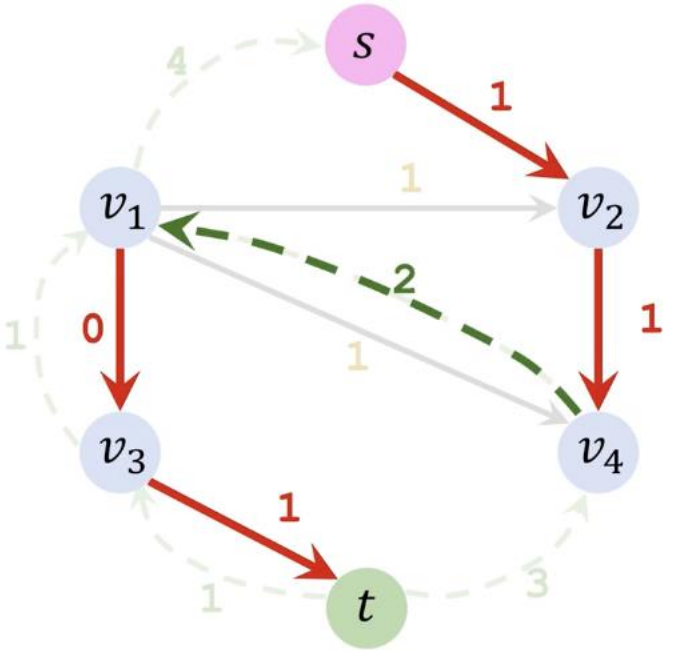
Found path $s \to v_2 \to v_4 \to v_1 \to v_3 \to t$.

Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

Iteration 3: update residual graph

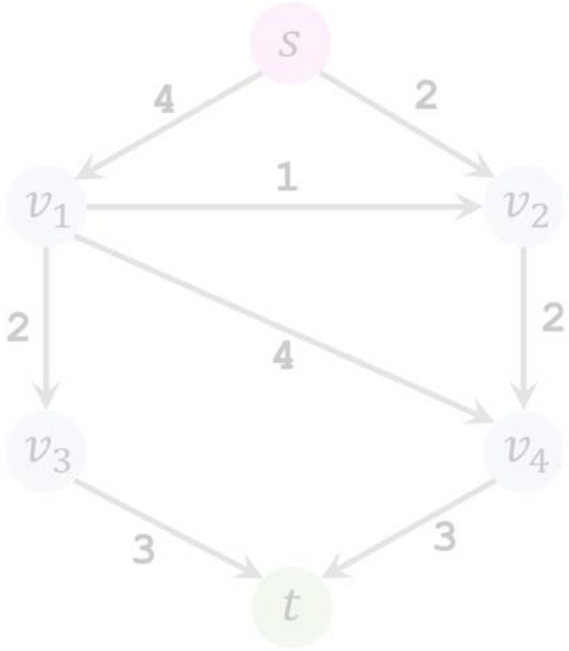


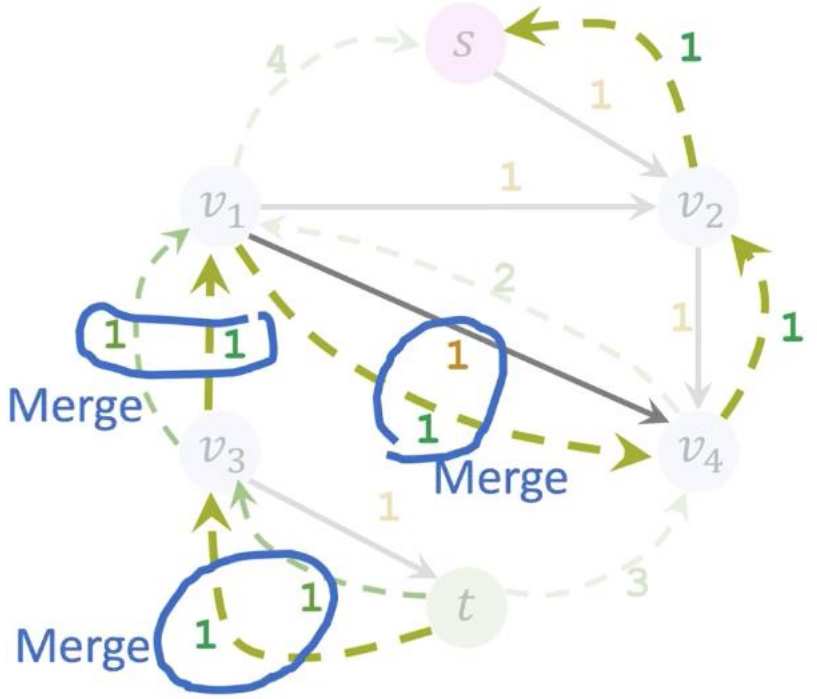


Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

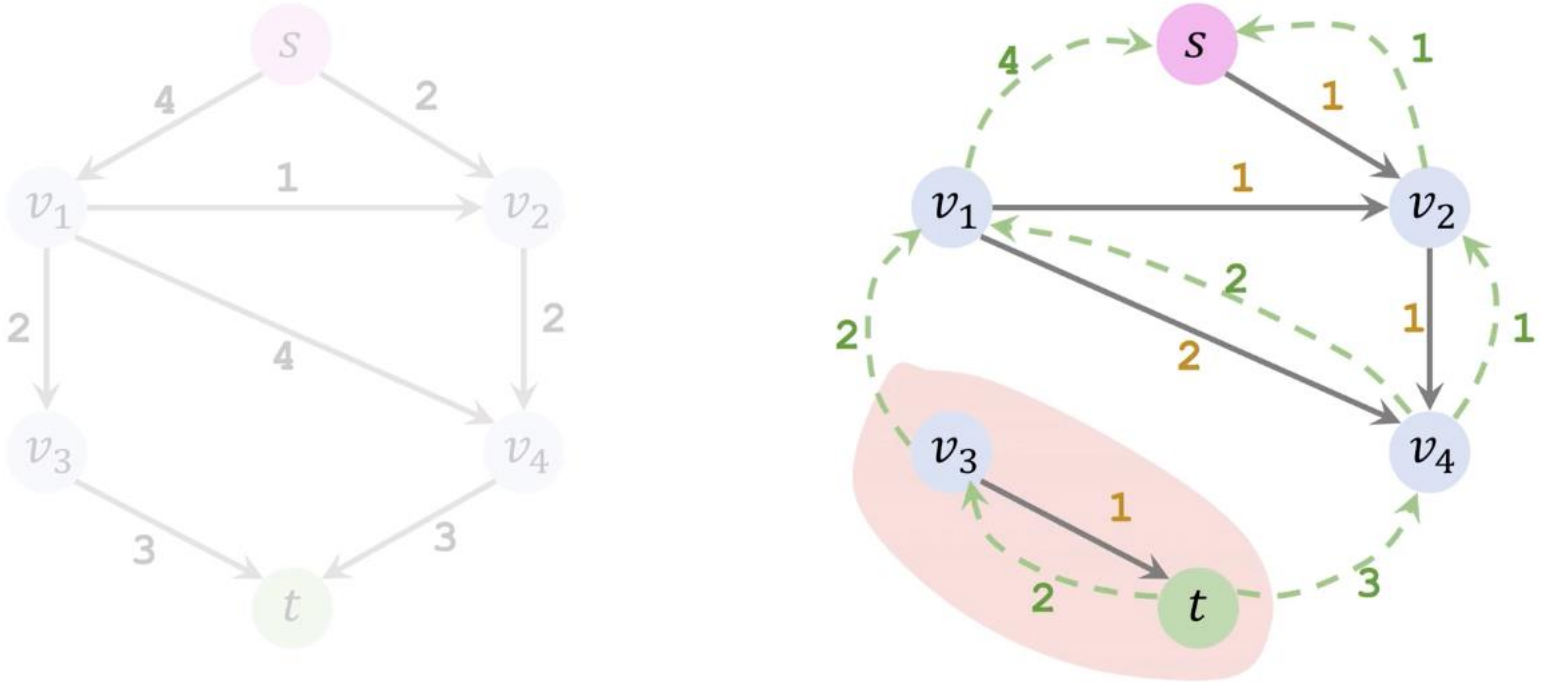
Iteration 3: update residual graph





Ford-Fulkerson Algorithm: Example

Iteration 4: find an augmenting path



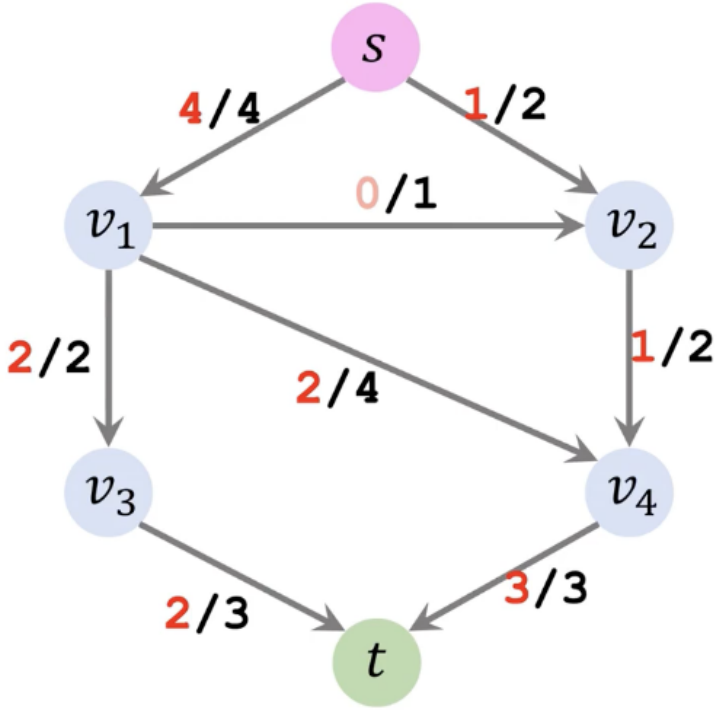
Cannot find any path from source to sink.

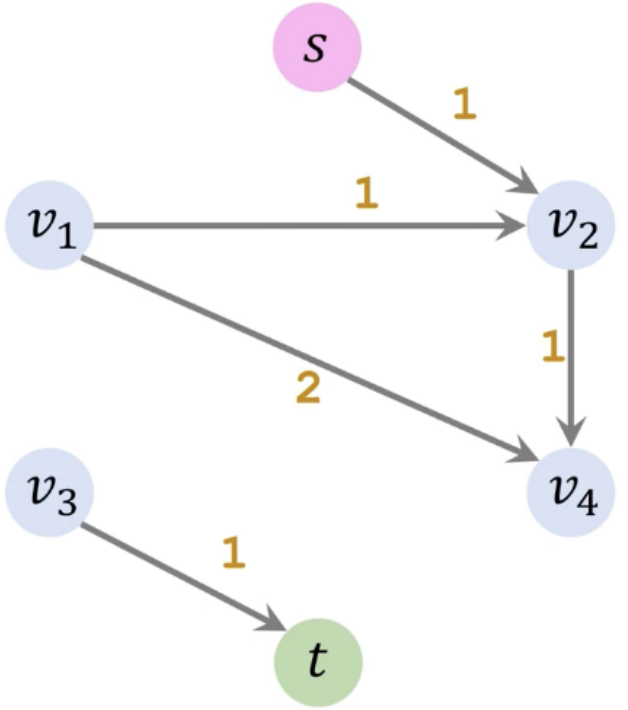
Ford-Fulkerson Algorithm: Example

香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

End of Procedure

(backward edges are removed here)





Running Time

- At every step, the flow values f(e) are integers. Start with integers and always add or subtract integers
- At every step we increase the amount of flow v(f) sent by at least 1 unit.
- We can never send more than C := $\sum_{e \text{ leaving } s} c_e$.

- Theorem: The Ford-Fulkerson algorithm terminates in C iterations of the *While* loop.

Time in the While loop

- If G has m edges, G_f has ≤ 2m edges.
- Can find an s−t path in G_f in time O(m+n) time with DFS or BFS.
- Since m ≥ n/2 (every node is adjacent to some edge), O(m + n) = O(m).

Theorem: The Ford-Fulkerson algorithm runs in O(mC) time.

Caveats, etc.

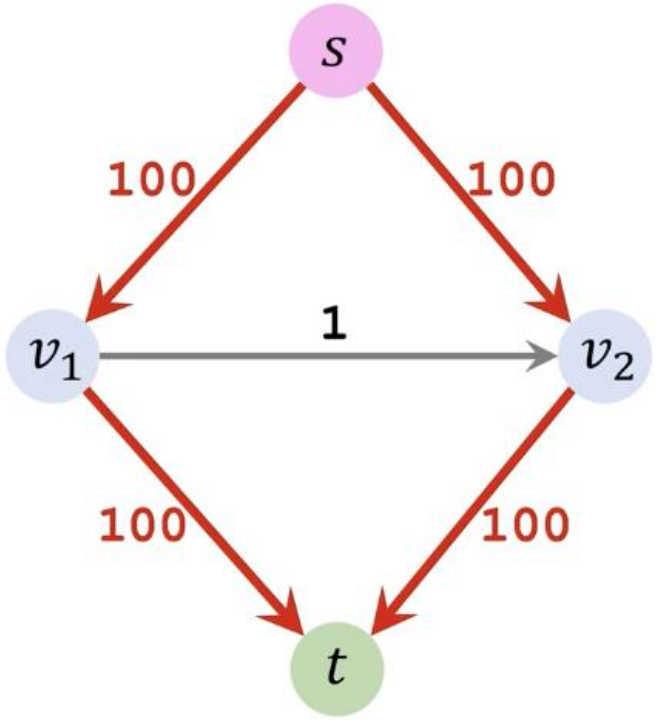
Note this is pseudo-polynomial because it depends on the size of the integers in the input.

You can remove this with slightly different algorithms. E.g.:

- $O(nm^2)$: Edmonds-Karp algorithm (use BFS to find the augmenting path)
- $O(m^2 \log C)$ or
- $O(n^2m)$ or $O(n^3)$

Ford-Fulkerson Algorithm: Example

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.

Edmonds-Karp Algorithm

Overview

- Edmonds-Karp algorithm [2] is almost the same as as Ford-Fulkerson algorithm [1].
- Edmonds-Karp algorithm uses the shortest path from source to sink. (Apply weight 1 to all the edges of the residual graph.)
- Edmonds-Karp algorithm is a special case of Ford-Fulkerson algorithm.

Reference

1. L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8: 399–404, 1956.
2. J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM.* 19 (2): 248–264, 1972.

Ford-Fulkerson Algorithm

```
MaxFlow(G):
  // initialize:
  Set f[e] = 0 for all e in G

  // while there is an s-t path in Gf:
  While P = FindPath(s,t, Residual(G,f)) != None:
     f = augment(f, P)
     UpdateResidual(G, f)
  EndWhile
  Return f
```

Time complexity: $O(Cm)$. (C is the max flow; m is #edges.)

Edmonds-Karp Algorithm

- Follow the same procedure as Ford-Fulkerson, except that we find the shortest augmenting path each time (on the residual graph).
- When finding paths, regard the residual graph as unweighted.
- Time complexity: $O(nm^2)$. (m is #edges; n is #vertices.)

Time Complexity Analysis

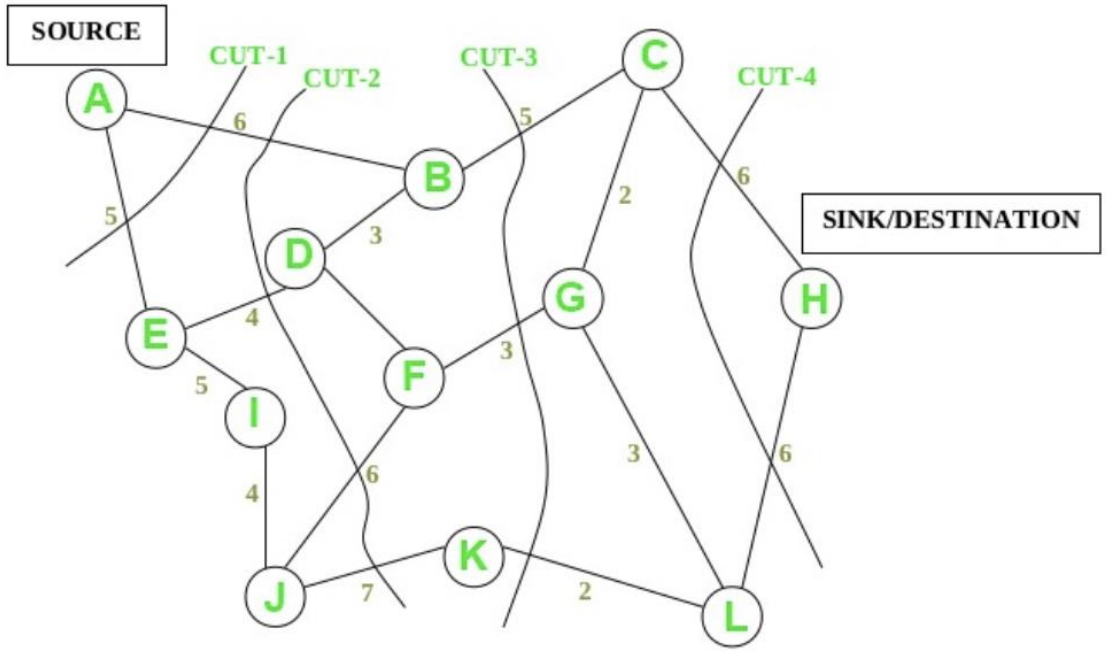
- m: number of edges.
- n: number of vertices.
- Each iteration has $O(m)$ time complexity.
- The number of iterations is at most $m \cdot n$.
- The worst-case time complexity is $O(nm^2)$.

Prove by yourself with ref2 if you are interested

Proof of Correctness

How do we know the flow is maximum?

Cuts in graphs



CUT-1 : AB, AE

CUT-2 : AB, ED,JF,JK

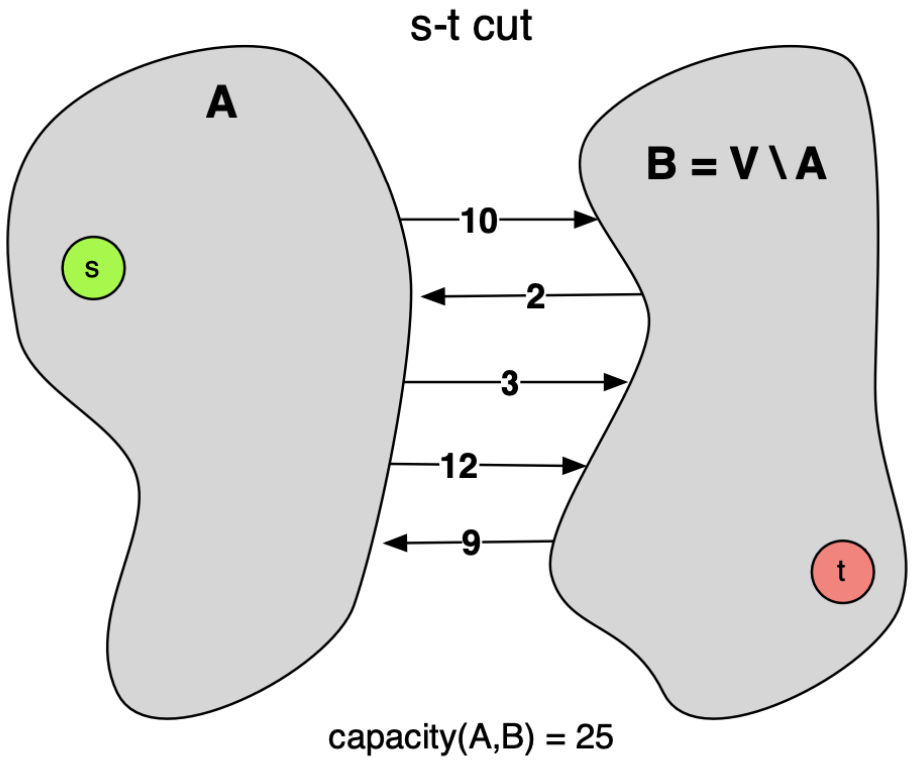
CUT-3 : BC, FG,KL

CUT-4 : CH, HL

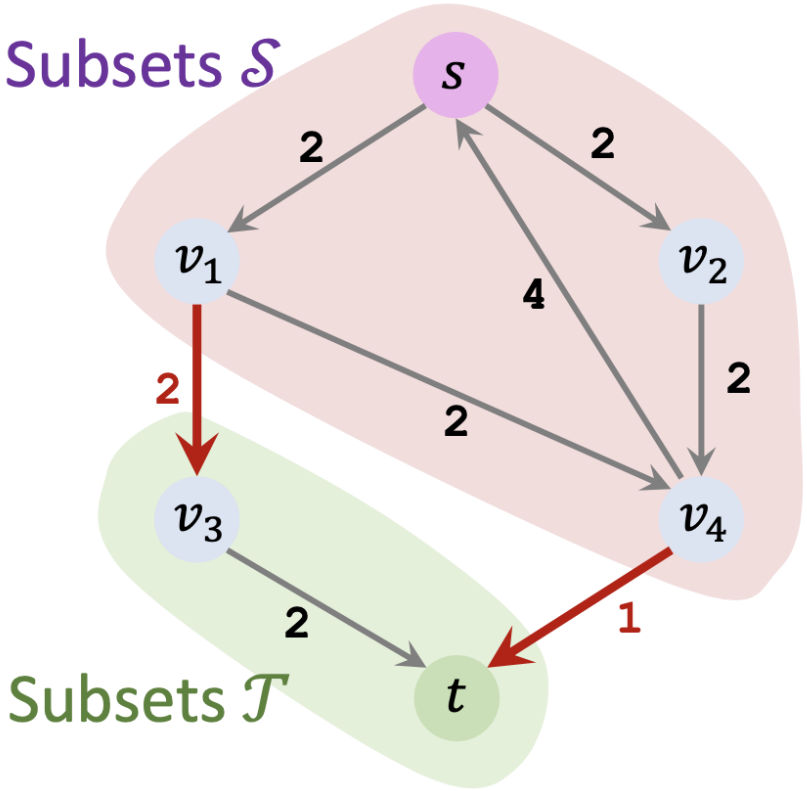
- What is the capacity of a cut? What about if edges are directed?

Cuts and Cut Capacity

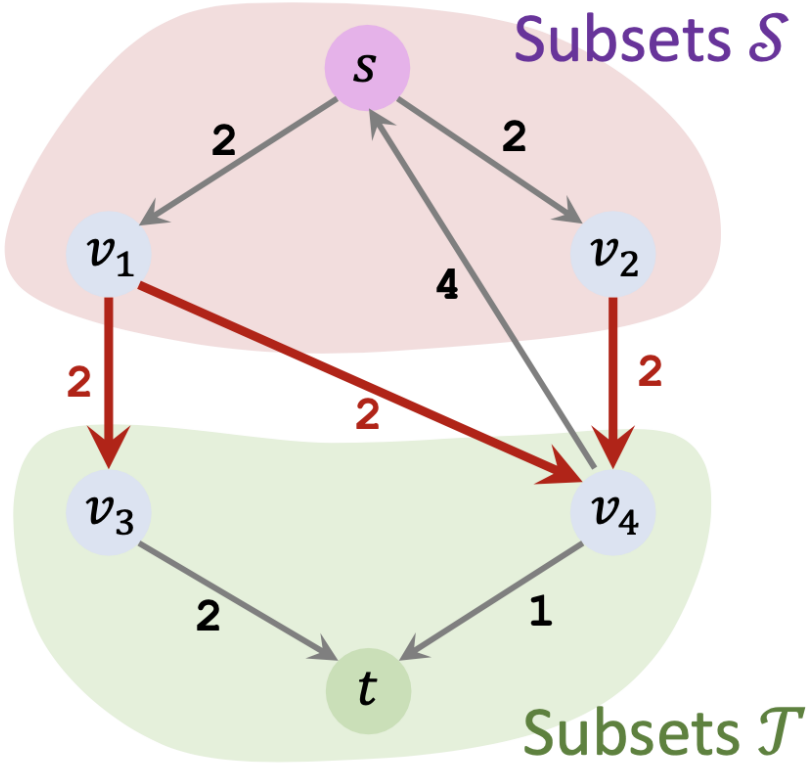
- Definition: The capacity (A,B) of an s-t cut (A,B) is the sum of the capacities of edges leaving A.
- Definition: Min-Cut is the s-t cut that minimizes capacity (A,B).



Example



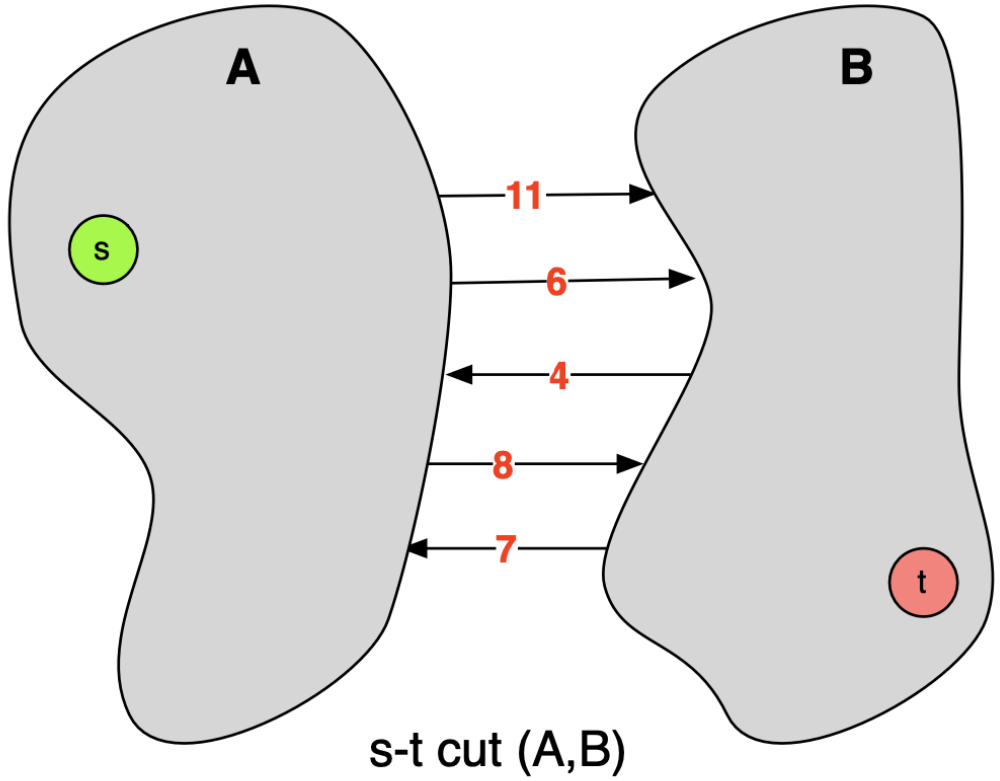
Min-cut



Not min-cut

A Cut Theorem

Theorem: Let f be an s-t flow and (A,B) be an s-t cut. Then $v(f) = f^{out}(A) - f^{in}(A)$.



Another cut theorem

- Theorem: Let f be any s-t flow and (A,B) be any s-t cut. Then v(f) ≤ capacity(A, B).

Proof.

$$
\begin{aligned}
v(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) && \text{prev. thm} \\
&\leq f^{\text{out}}(A) && f^{\text{in}}(A) \text{ is } \geq 0 \\
&= \sum_{e \text{ leaving } A} f(e) && \text{by definition} \\
&\leq \sum_{e \text{ leaving } A} c_e && \text{by capacity constraints} \\
&= \textbf{capacity}(A, B) && \text{by definition}
\end{aligned}
$$

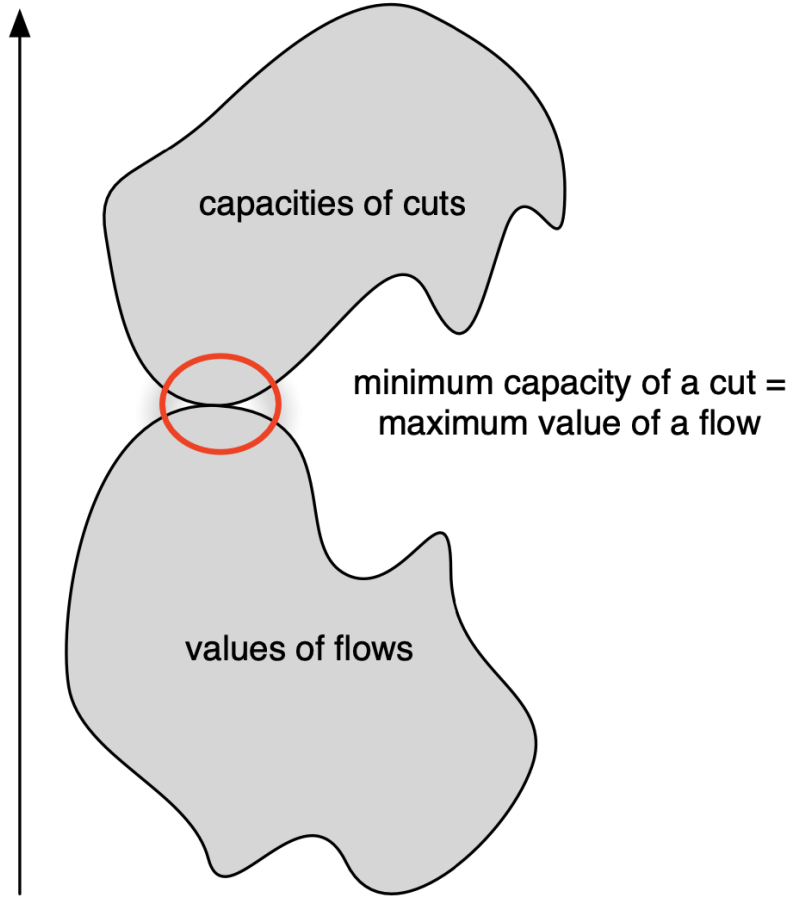
□

Cuts Constrain Flows

Theorem: Let f be any s-t flow and (A,B) be any s-t cut. Then v(f) ≤ capacity(A,B).

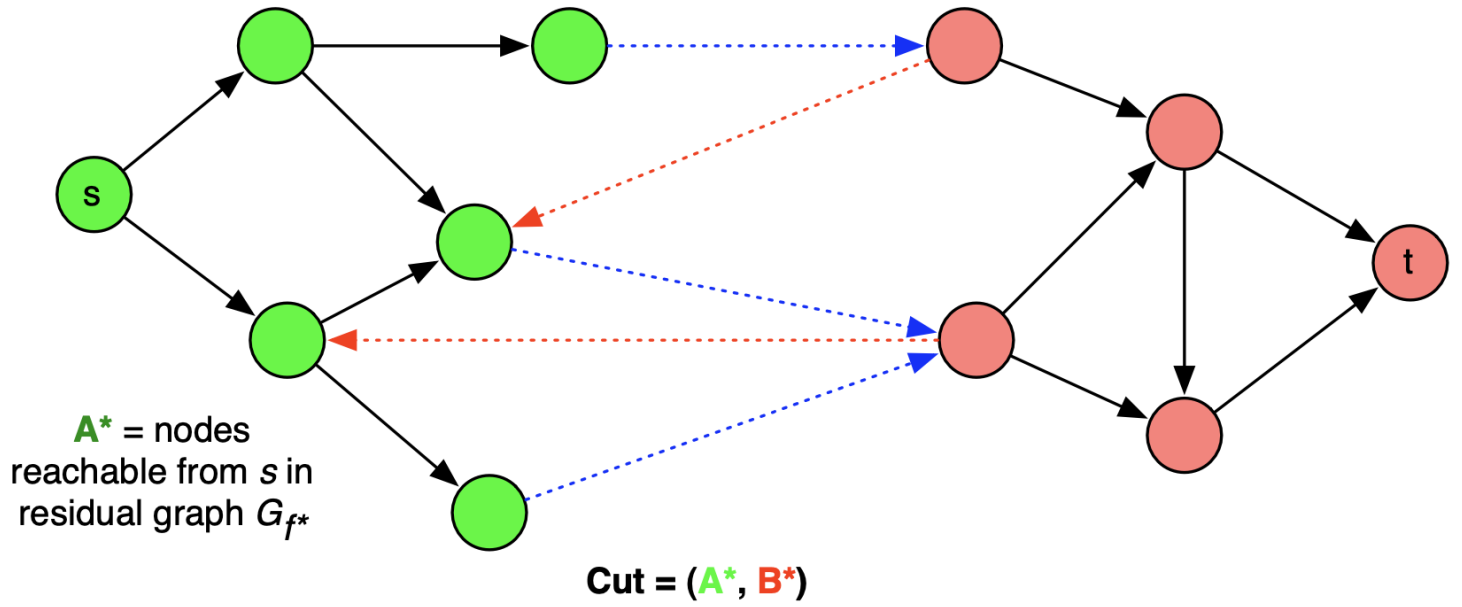
- Says that any cut is bigger than any flow.
- Therefore, cuts constrain flows. The minimum capacity cut constrains the maximum flow the most.
- In fact, the capacity of the minimum cut always equals the maximum flow value.

Max-Flow = Min-Cut



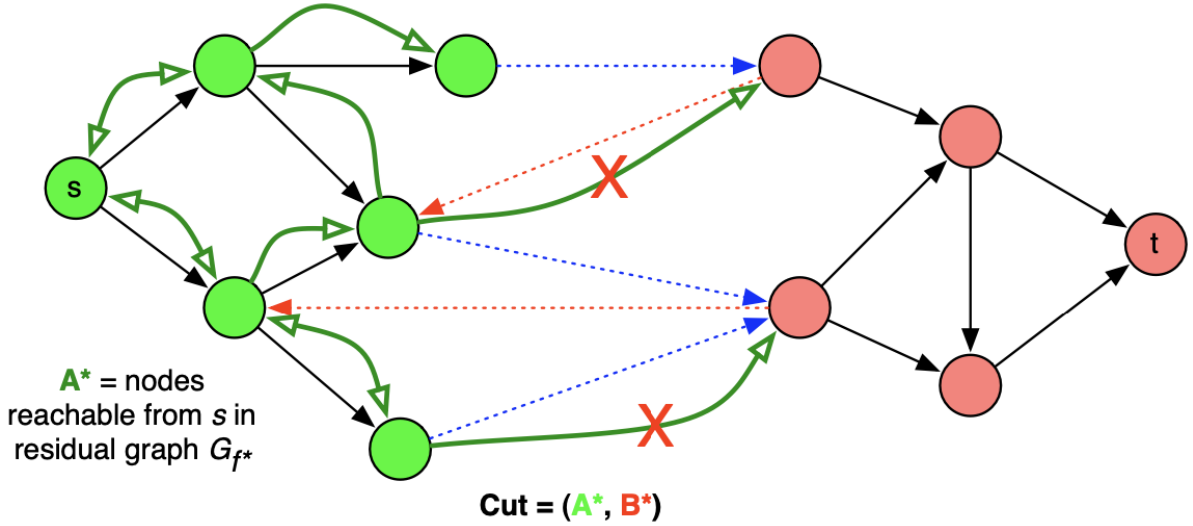
Max-Flow = Min-Cut

- Let f* be the flow returned by our algorithm. Look at G_{f^*} but define a cut in G:



Blue edges must be saturated.
Red edges must have 0 flow $\implies v(f^*) = \text{capacity}(A^*, B^*)$.

Max-Flow = Min-Cut



- (A*,B*) is an s-t cut because there is no path from s to t in the residual graph G_{f*}.
- Edges (u,v) from A* to B* must be saturated — otherwise there would be a forward edge (u,v) in G_{f*} and v would be part of A*.
- Edges (v,u) from B* to A* must be empty — otherwise there would be a back edge (u,v) in G_{f*} and v would be part of A*.

Max-Flow = Min-Cut

Therefore,

- v(f*)=capacity(A*,B*).
- No flow can have value bigger than capacity(A*,B*).
- So, f* must be an maximum flow.
- And (A*,B*) has to be a minimum-capacity cut.

Theorem (Max-flow = Min-cut): The value of the maximum flow in any flow graph is equal to the capacity of the minimum cut.

Reference

- L. R. Ford and D. R. Fulkerson. Flows in Networks. Princeton University Press, 1962.

Finding the Min-Capacity Cut

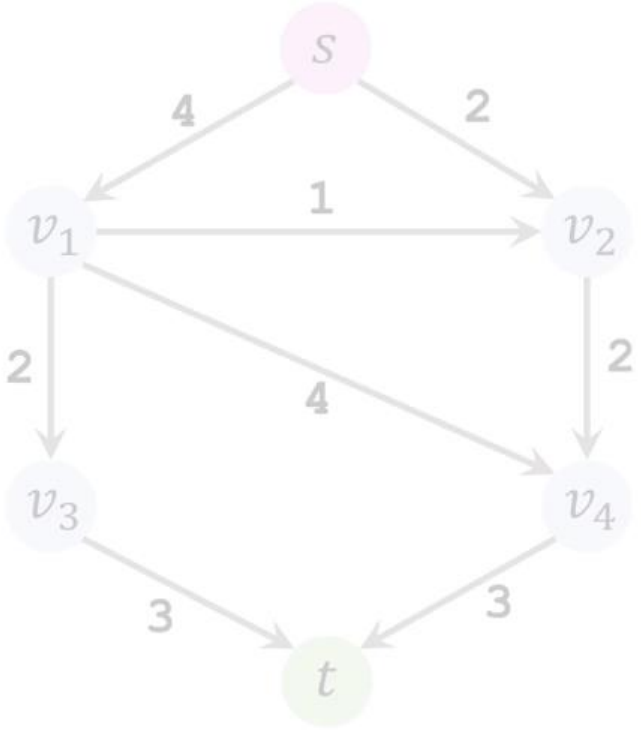
Our proof that maximum flow = minimum cut can be used to actually find the minimum capacity cut:

- Find the maximum flow f*.
- Construct the residual graph G_{f^*} for f*.
- Do a BFS to find the nodes reachable from s in G_{f^*}. Let the set of these nodes be called A*.
- Let B* be all other nodes.
- Return (A*, B*) as the minimum capacity cut.

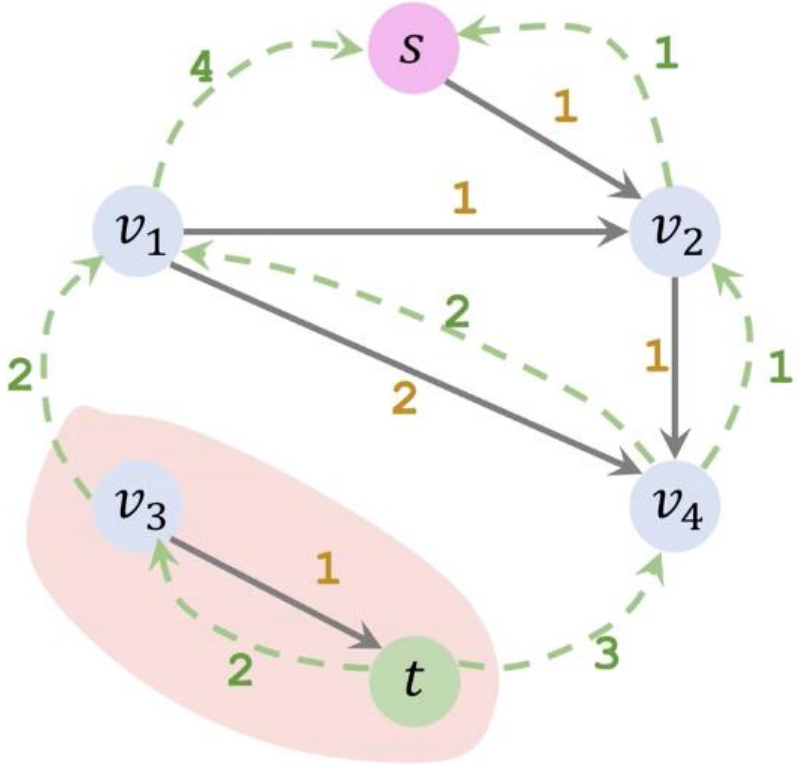
With backward edges

Example

Origianl Graph



Final Residual Graph

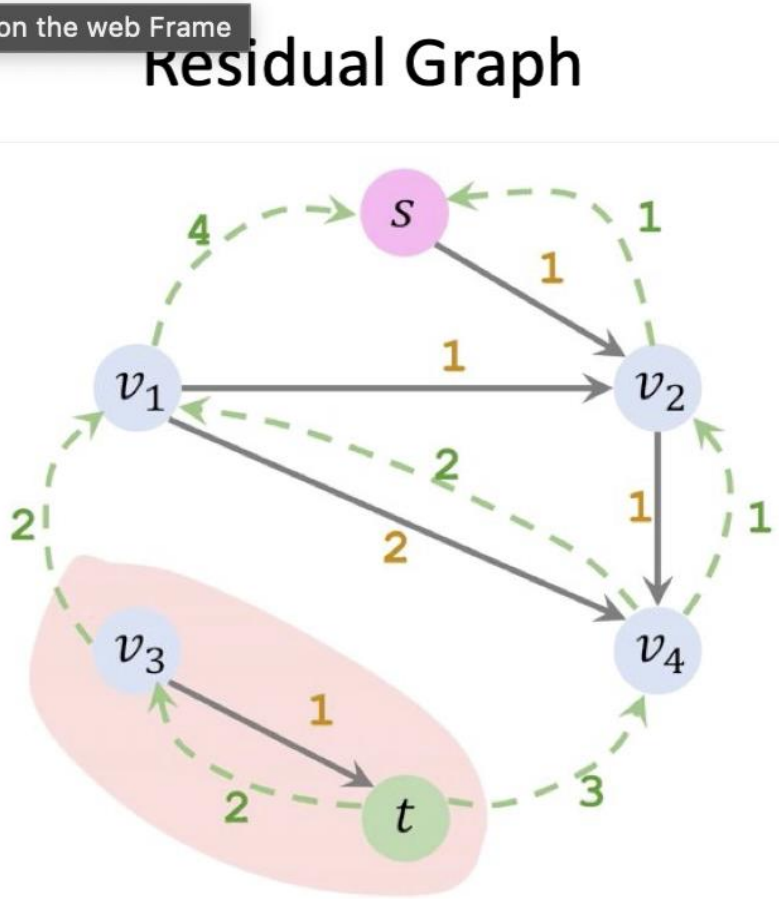


Example

- Find vertices reachable from S
- Subset S = {s, v_1, v_2, v_4}.

- The remaining vertices are t and v3
- Subset T = {t, v_3}

- Q: What is the min-cut then?



Applications

- The Maximum Flow Minimum Cut Theorem is a pivotal concept in network flow theory. This theorem has a wide range of applications across various fields. Here are some specific examples of its applications:
  - Water Resources Management
  - Resource Allocation
  - Transportation and Logistics
  - Communications Networks
  - Economic Decision-Making
  - Image Processing
  - Medical Field
  - Social Network Analysis

Transportation and Logistics

- Water Distribution Networks: Ensuring the maximum water flow from a source to various users.

- Energy Distribution: Optimizing the maximum electricity transmission from power stations to consumers in power networks.

Communications Networks

- Bandwidth Allocation: Ensuring the maximum data transmission rate from source to destination on the internet.
- Routing Algorithms: Choosing paths to maximize network throughput in data packet transmission.
- Production Process Optimization: Optimizing the maximum flow from raw materials to finished goods while minimizing production bottlenecks.

Summary

- Ford-Fulkerson algorithm can find max flow in O(mC) time.
  - Algorithm idea: Send flow along some path with capacity left, possibly "erasing" some flow we've already sent. Use residual graph to keep track of remaining capacities and flow we've already sent.
- We can eliminate C to get a true polynomial algorithm by using BFS to find our augmenting paths.
- All cuts have capacity ≥ the value of all flows.
- Know the flow is maximum because its value equals the capacity of some cut.

The End