CSE 202: Design and Analysis of Algorithms

Lecture 8

Instructor: Kamalika Chaudhuri

Last Class: Max Flow Problem

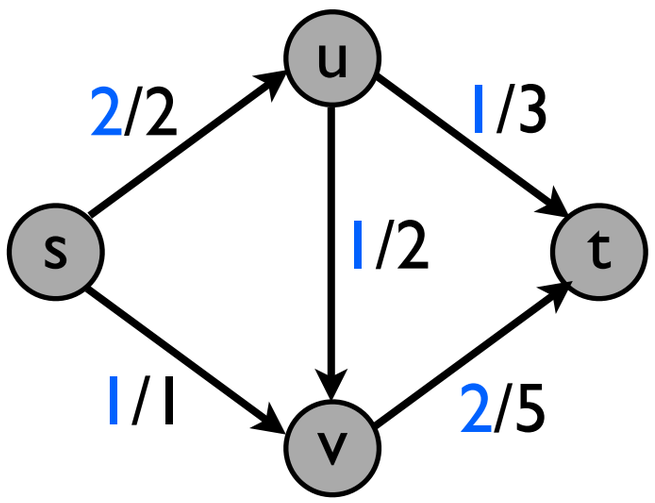
The Max Flow Problem: Given directed graph G=(V,E), source s, sink t, edge capacities c(e), find an s-t flow of maximum size

An s-t flow is a function f: E $\longrightarrow$ R such that:

- 0 <= f(e) <= c(e), for all edges e
- flow into node v = flow out of node v, for all nodes v except s and t,

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

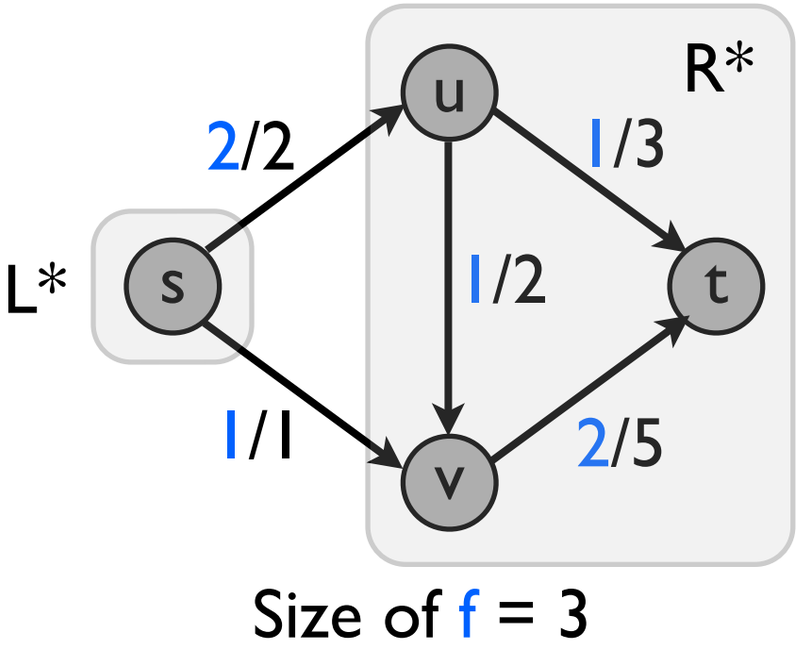
Size of flow f = Total flow out of s = total flow into t



Size of f = 3

Last Class: Flows and Cuts

The Max Flow Problem: Given directed graph G=(V,E), source s, sink t, edge capacities c(e), find an s-t flow of maximum size



Size of f = 3

An **s-t Cut** partitions nodes into groups = (L, R) s.t. s in L, t in R

Capacity of a cut (L, R) = $\sum_{(u,v)\in E, u\in L, v\in R} c(u,v)$

Flow across (L,R) = $\sum_{(u,v)\in E, u\in L, v\in R} f(u,v) - \sum_{(v,u)\in E, u\in L, v\in R} f(v,u)$

Property: For any flow f, any s-t cut (L, R), size(f) <= capacity(L, R)

Max-Flow <= Min-Cut

Thus, a Min Cut is a **certificate of optimality for a flow**

Cuts

Flows

Last Class: Ford-Fulkerson Algorithm

FF Algorithm: Start with zero flow

Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

In any iteration, we have some flow f and we are trying to improve it. How to do this?

1: Construct a residual graph G_f ("what's left to take?")

$G_f = (V, E_f)$ where $E_f \subseteq E \cup E^R$

For any (u,v) in E or E^R,

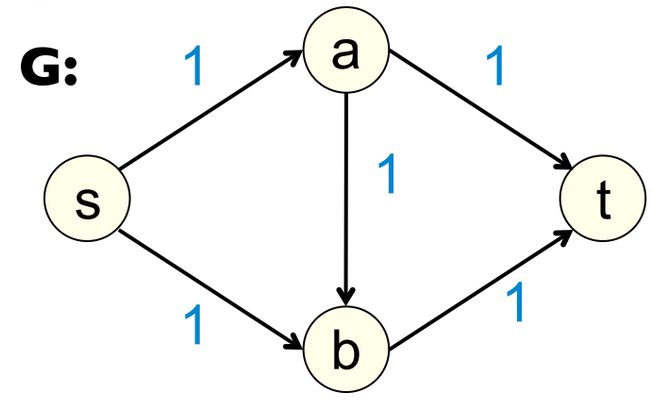
$c_f(u,v) = c(u,v) - f(u,v) + f(v,u)$

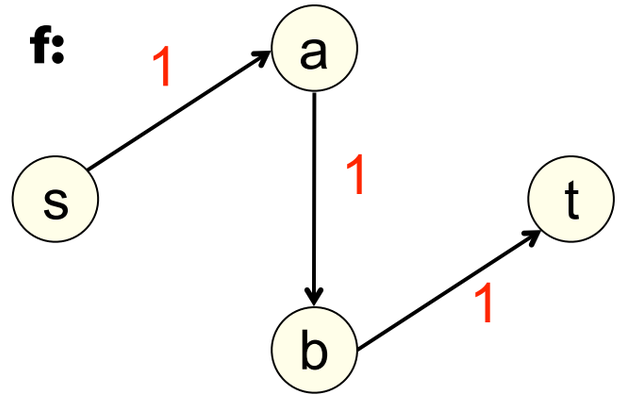
[ignore edges with zero c_f: don't put them in E_f]

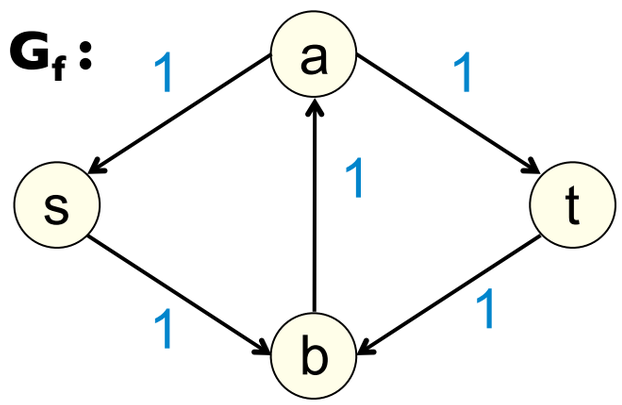
2: Find a path from s to t in G_f

3: Increase flow along this path, as much as possible

Example



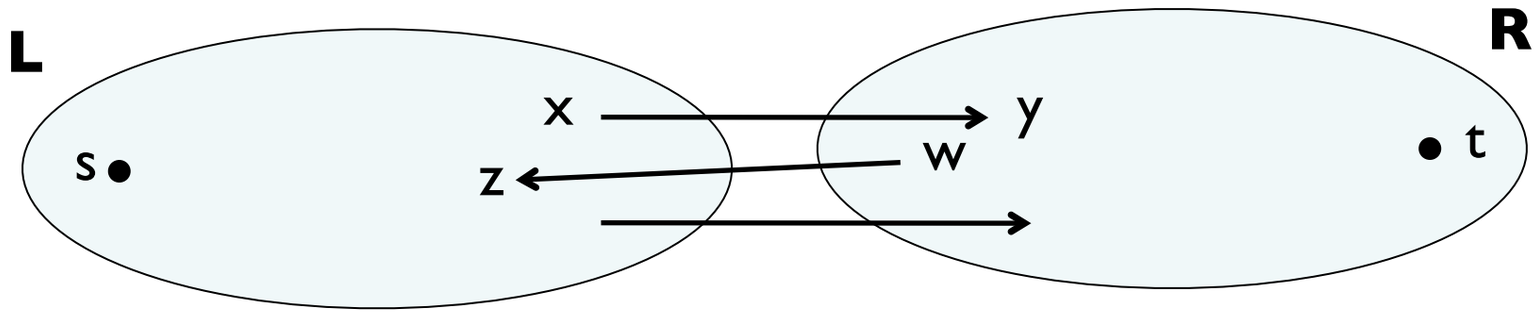




Analysis: Correctness

FF algorithm gives us a valid flow. But is it the **maximum** possible flow?

Consider final residual graph $G_f = (V, E_f)$
Let L = nodes reachable from s in G_f and let R = rest of nodes = V – L
So $s \in L$ and $t \in R$



Edges from L to R must be at full capacity
Edges from R to L must be empty
Therefore, flow across cut (L,R) is

$$\sum_{(u,v)\in E, u\in L, v\in R} c(u,v)$$

Thus, size(f) = capacity(L,R)

Recall: for any flow and any cut,
size(flow) <= capacity(cut)

Therefore f is the **max flow** and (L,R) is the **min cut**!

Thus, **Max Flow** = **Min Cut**

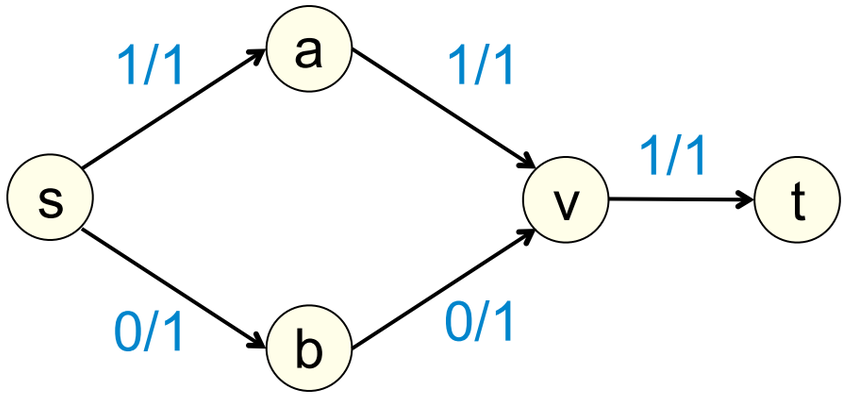
Cuts

Flows

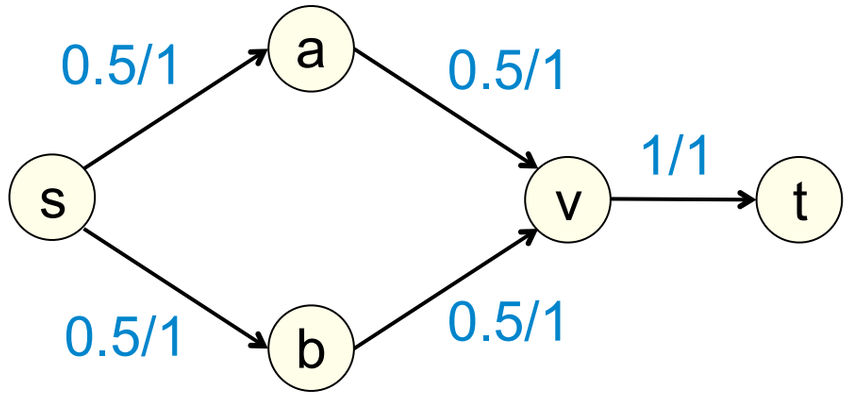
An Observation: Integrality

Integral Flows: A flow f is integral if f(e) is an integer for all e

Example:



An Integral Flow

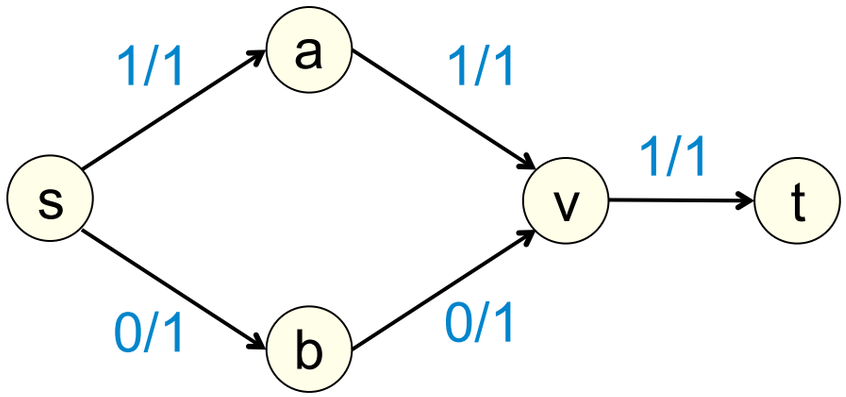


A Fractional Flow

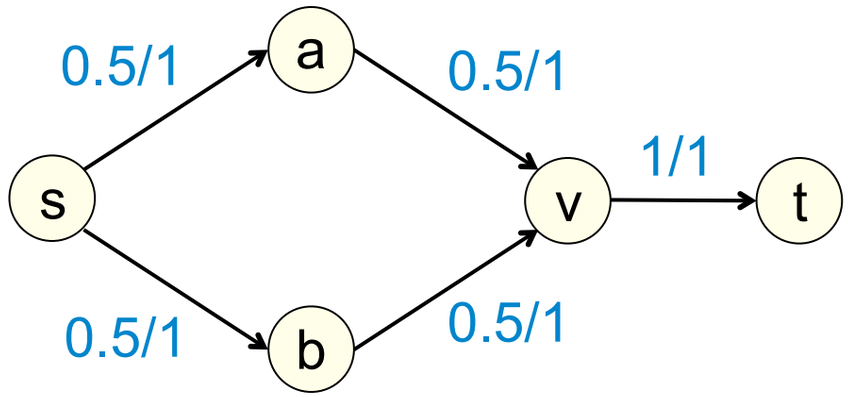
An Observation: Integrality

Integral Flows: A flow f is integral if f(e) is an integer for all e

Example:



An Integral Flow



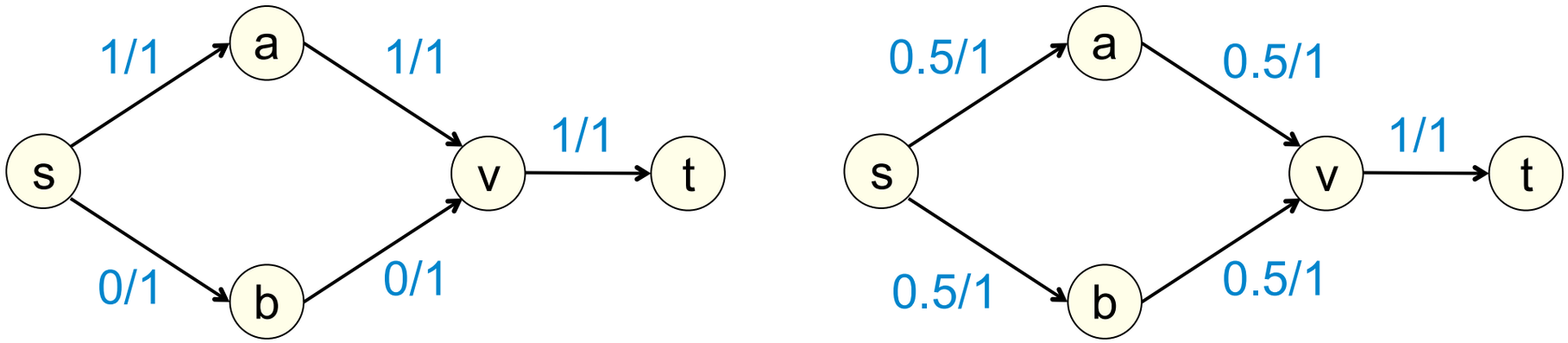
A Fractional Flow

Property: If all edge capacities are integers, then, there is a max flow f which is integral.

An Observation: Integrality

Integral Flows: A flow f is integral if f(e) is an integer for all e

Example:



An Integral Flow **A Fractional Flow**

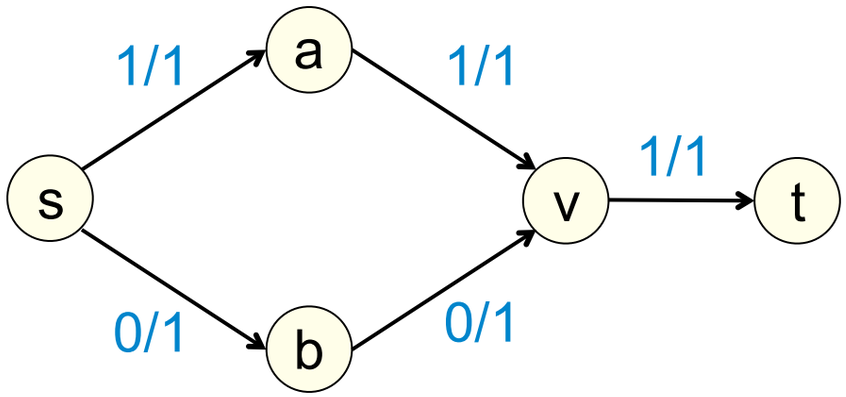
Property: If all edge capacities are integers, then, there is a max flow f which is integral.

Proof: If the edge capacities are integers, then, the FF algorithm always finds an integral flow The FF algorithm also always finds a max flow.

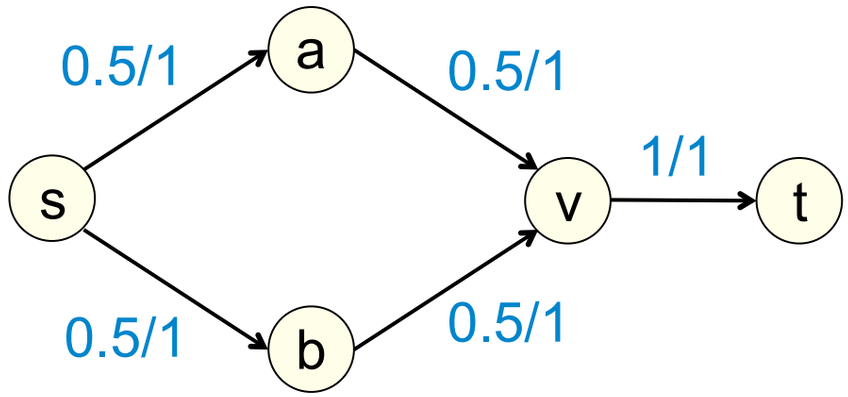
An Observation: Integrality

Integral Flows: A flow f is integral if f(e) is an integer for all e

Example:



An Integral Flow



A Fractional Flow

Property: If all edge capacities are integers, then, there is a max flow f which is integral.

Proof: If the edge capacities are integers, then, the FF algorithm always finds an integral flow The FF algorithm also always finds a max flow.

Note: **All max flows** are not necessarily integral flows!

Analysis: efficiency

FF Algorithm: Start with zero flow

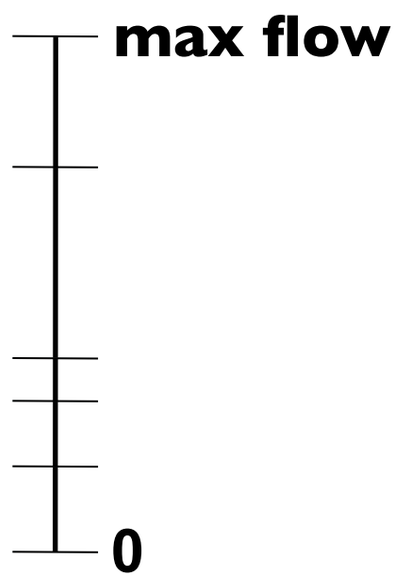
Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

A **hillclimbing** procedure

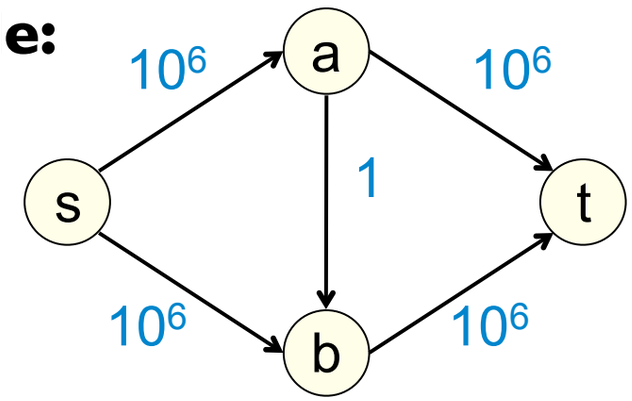
Flow size:

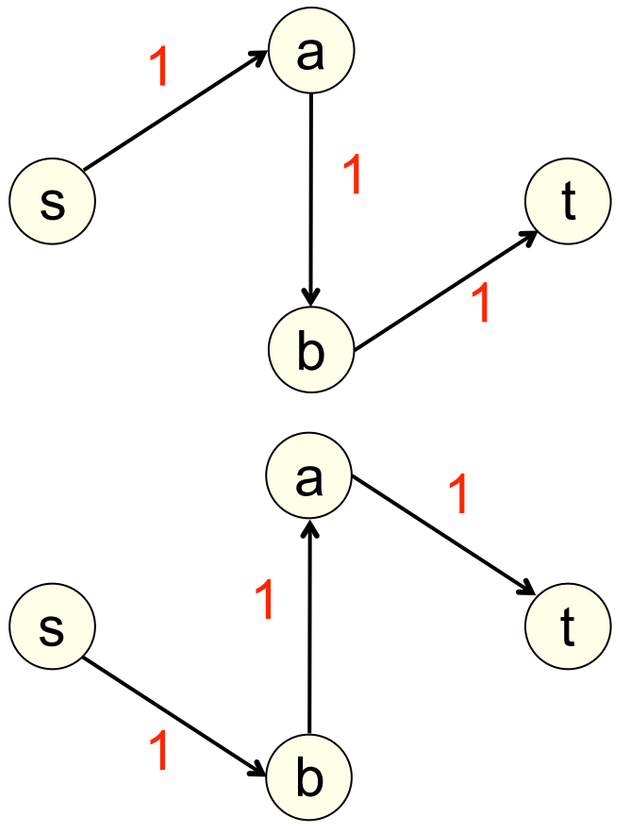


Each iteration is fast (O(|E|) time).

How many iterations are needed to reach the maximum flow?

Example:





#iterations can be Max Capacity

Analysis: efficiency

FF Algorithm: Start with zero flow

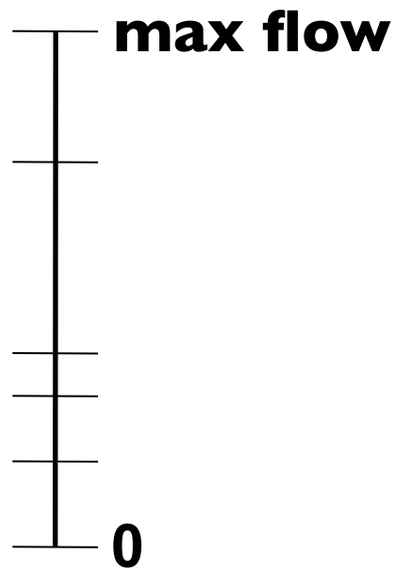
Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

A **hillclimbing** procedure

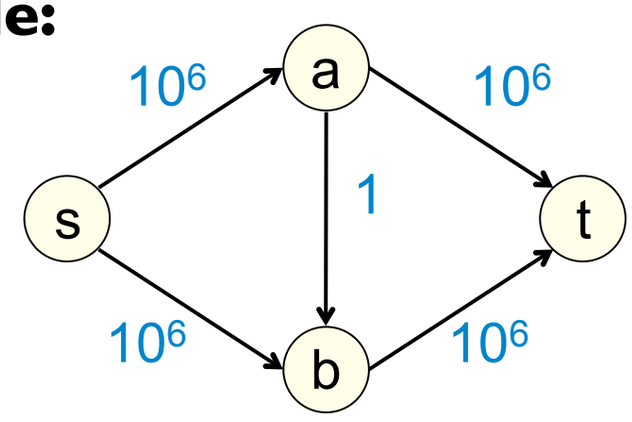
Flow size:



Each iteration is fast (O(|E|) time).

How many iterations are needed to reach the maximum flow?

Example:



#iterations can be Max Capacity (with integer capacities)

How to improve the efficiency?

- Ford-Fulkerson Style Algorithms:
  - Edmonds Karp
  - Capacity Scaling
- Preflow-Push

Edmonds Karp

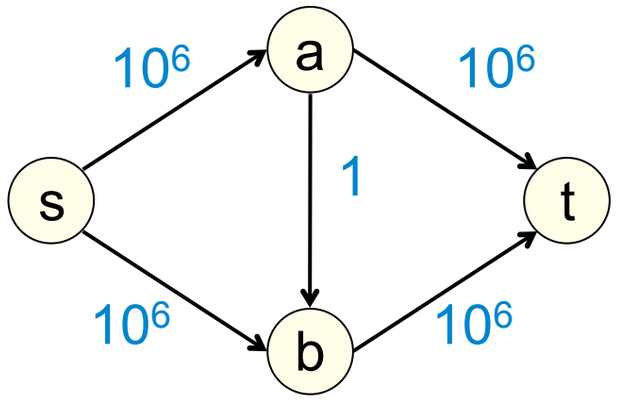
FF Algorithm: Start with zero flow

Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

Bad Example:



Bad Path Sequence:

(s, a, b, t), (s, b, a, t), (s, a, b, t),...

Edmonds Karp

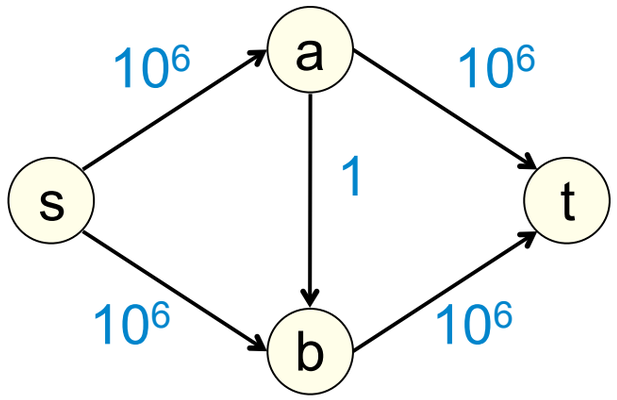
FF Algorithm: Start with zero flow

Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

Bad Example:



Bad Path Sequence:

(s, a, b, t), (s, b, a, t), (s, a, b, t),...

EK Path Selection: Find the **shortest path** along which flow can be increased (shortest path = shortest in terms of #edges)

Edmonds Karp

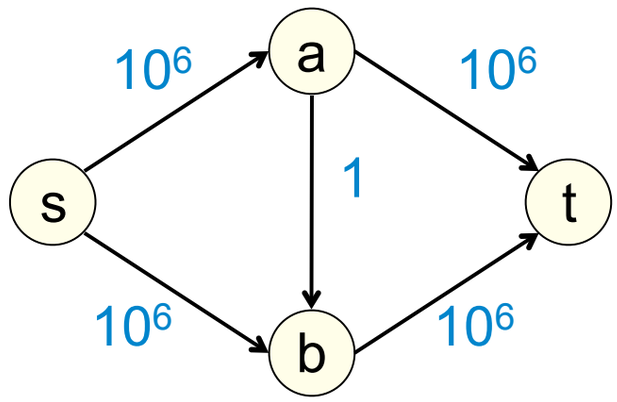
EK Algorithm: Start with zero flow

Repeat:

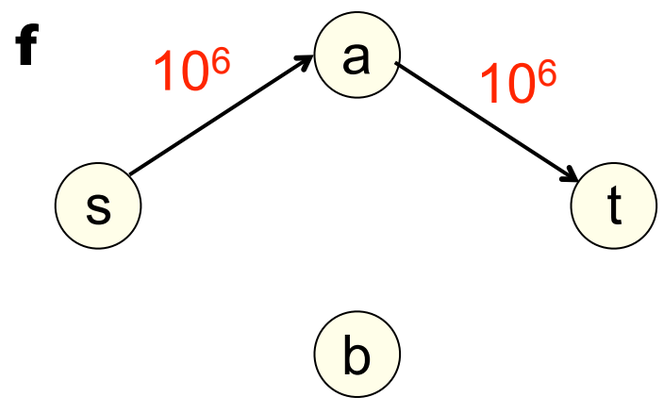
Find the **shortest path** from s to t along which flow can be increased

Increase the flow along that path

Bad Example for FF:



Iteration 1



Edmonds Karp

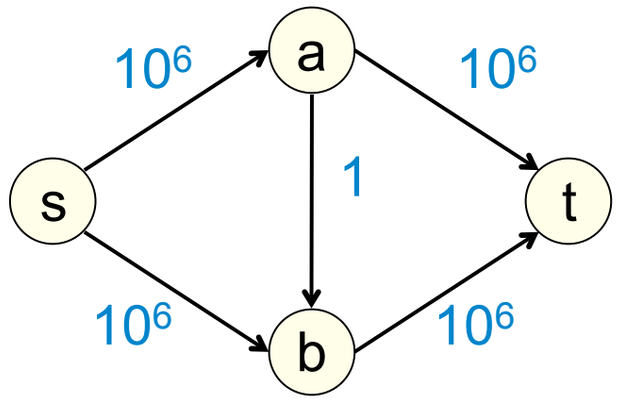
EK Algorithm: Start with zero flow

Repeat:

Find the **shortest path** from s to t along which flow can be increased

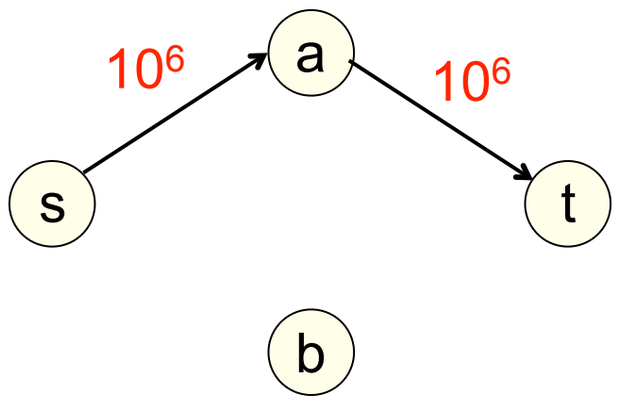
Increase the flow along that path

Bad Example for FF:

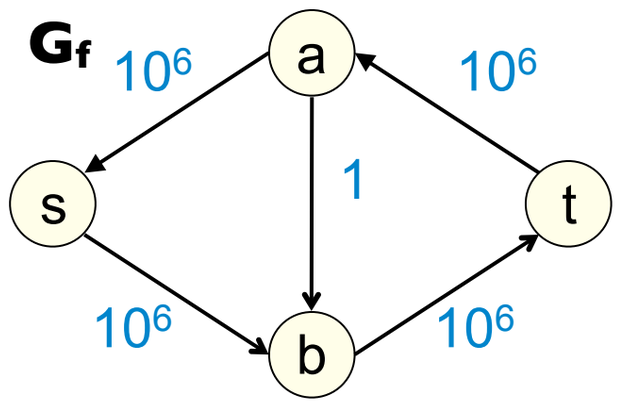


Iteration 1

f



G_f



Edmonds Karp

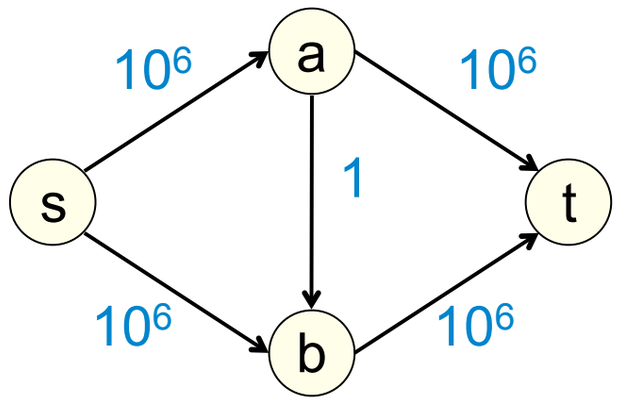
EK Algorithm: Start with zero flow

Repeat:

Find the **shortest path** from s to t along which flow can be increased

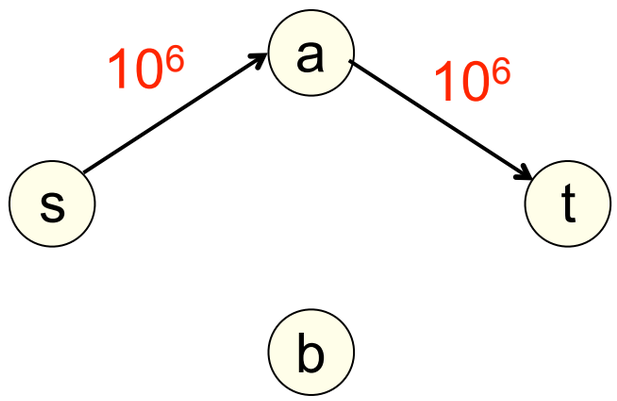
Increase the flow along that path

Bad Example for FF:

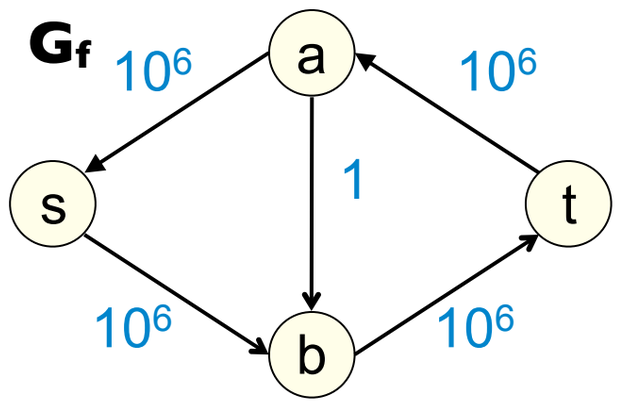


Iteration 1

f

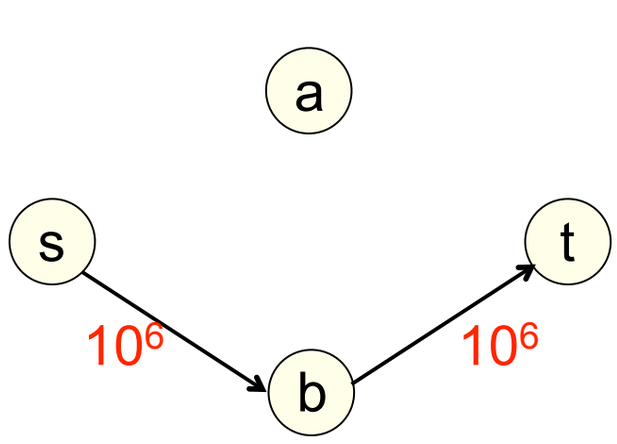


G_f



Iteration 2

f



Edmonds Karp

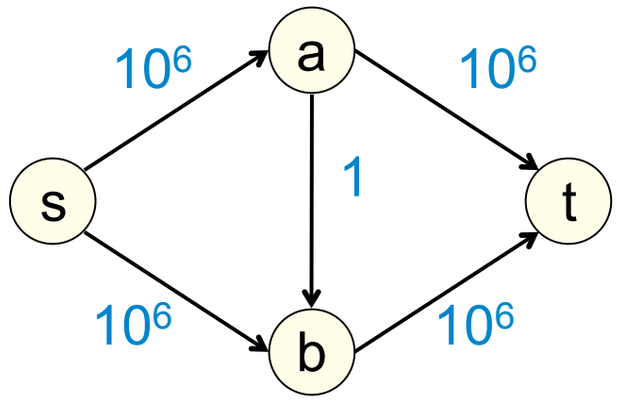
EK Algorithm: Start with zero flow

Repeat:

Find the **shortest path** from s to t along which flow can be increased

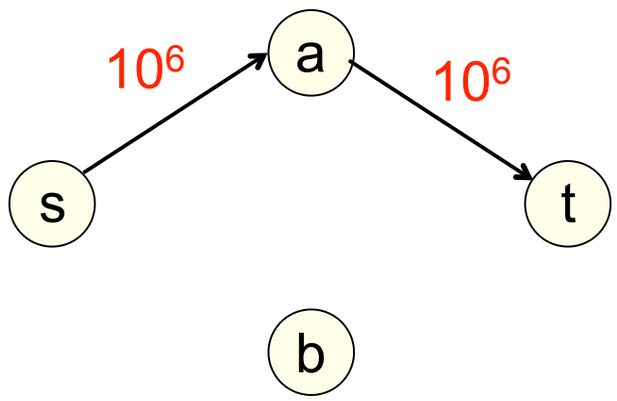
Increase the flow along that path

Bad Example for FF:

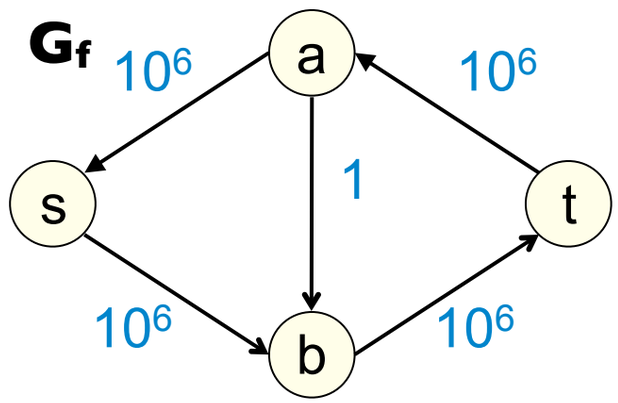


Iteration 1

f

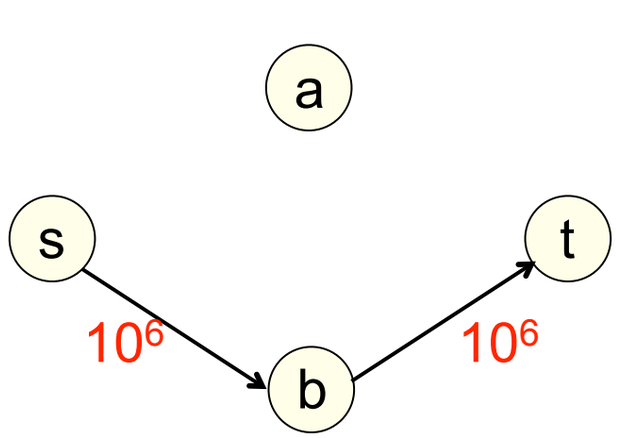


G_f

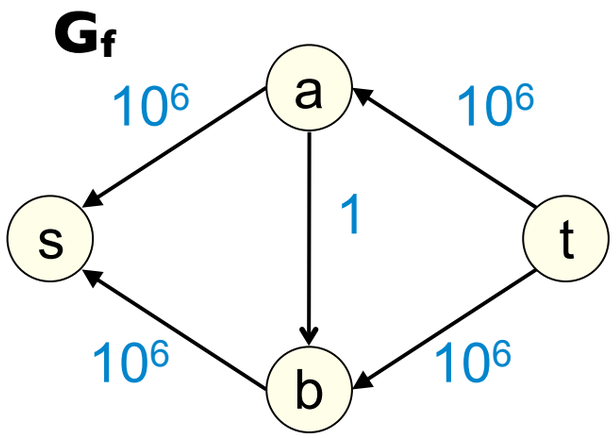


Iteration 2

f



G_f



Edmonds Karp

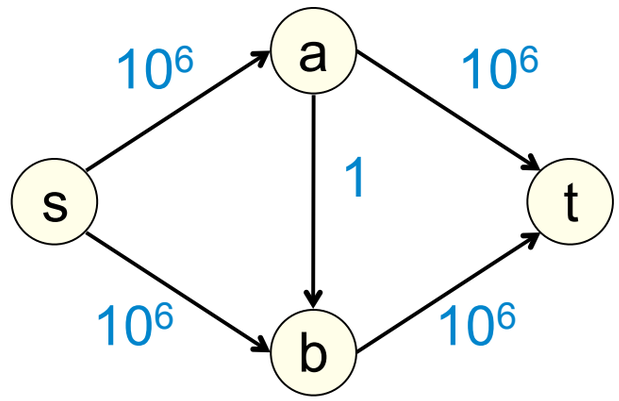
FF Algorithm: Start with zero flow

Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

Bad Example for FF:



Bad Path Sequence:

(s, a, b, t), (s, b, a, t), (s, a, b, t),...

EK Path Selection: Find the **shortest path** along which flow can be increased (shortest path = shortest in terms of #edges)

It can be shown that this requires only O(|V||E|) iterations (Proof not in this class)

Edmonds Karp

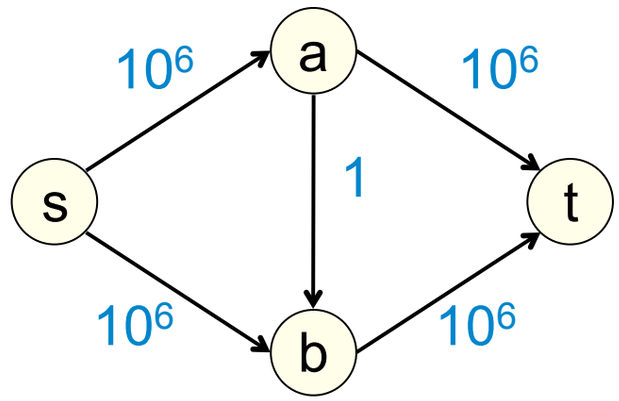
FF Algorithm: Start with zero flow

Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

Bad Example for FF:



Bad Path Sequence:

(s, a, b, t), (s, b, a, t), (s, a, b, t),...

EK Path Selection: Find the **shortest path** along which flow can be increased (shortest path = shortest in terms of #edges)

It can be shown that this requires only O(|V||E|) iterations (Proof not in this class)

Running Time: $O(|V| |E|^2)$

How to improve the efficiency?

- Ford-Fulkerson Style Algorithms:
  - Edmonds Karp
  - Capacity Scaling
- Preflow-Push

Capacity Scaling

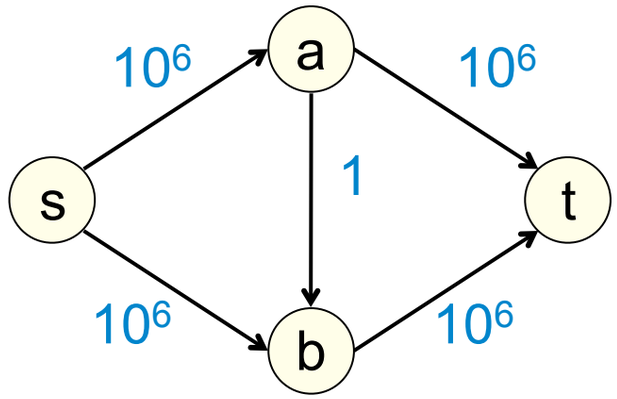
FF Algorithm: Start with zero flow

Repeat:

Find a path from s to t along which flow can be increased

Increase the flow along that path

Bad Example:



Bad Path Sequence:

(s, a, b, t), (s, b, a, t), (s, a, b, t),...

Capacity Scaling: Find **paths of high capacity first** between s and t

Capacity Scaling

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

While D >= 1, repeat:

    $G_f(D)$ = D-residual graph

    While there is a path from s to t in $G_f(D)$ along which flow can be increased

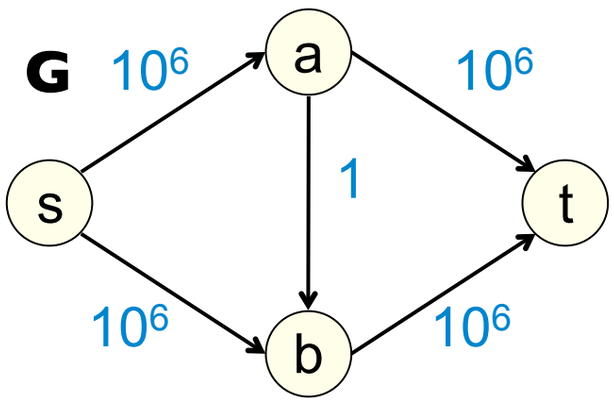
        Increase flow along that path

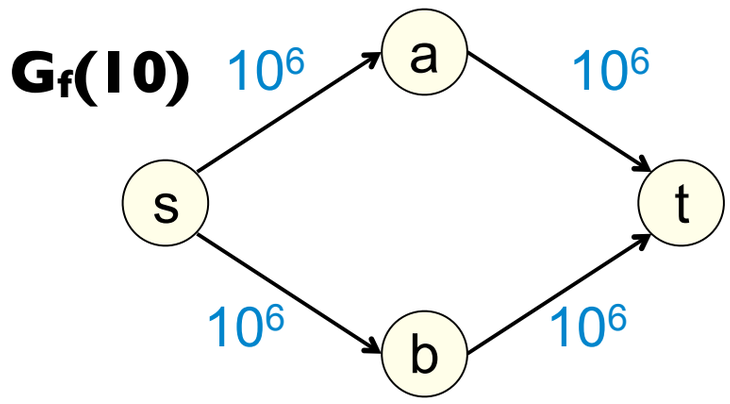
        Update $G_f(D)$

    D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Example: For f = 0





Capacity Scaling: Correctness

C_{max} = max capacity edge. Start with D = C_{max}
Start with zero flow
While D >= 1, repeat:
    $G_f(D)$ = D-residual graph
    While there is a path from s to t in $G_f(D)$ along which flow can be increased
        Increase flow along that path
        Update $G_f(D)$
    D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property: If all edge capacities are integers, algorithm outputs a max flow

Proof: At D=1, $G_f(D) = G_f$. So on termination, $G_f(D)$ has no more paths from s to t

Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

  D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

D scaling phase

2 $G_f(D)$ = D-residual graph

While there is a path from s to t in $G_f(D)$ along which flow can be increased

Increase flow along that path

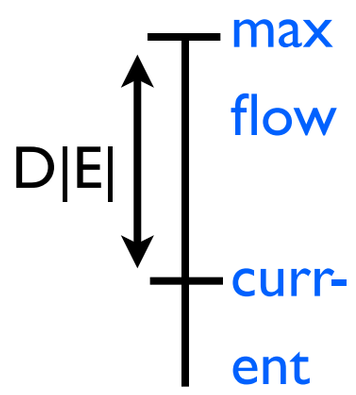
Update $G_f(D)$

D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|



Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

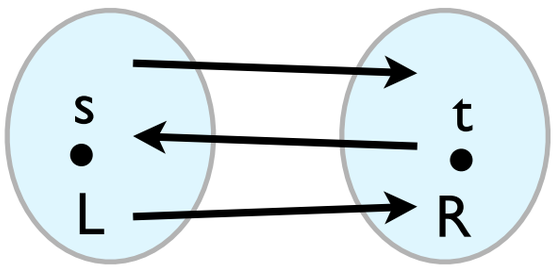
  D = D/2

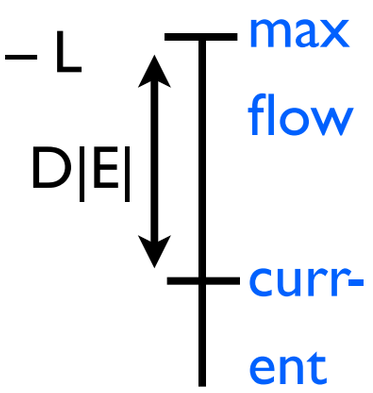
D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Proof: Let L = nodes reachable from s in $G_f(D)$ and let R = rest of nodes = V – L





Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

D scaling phase

2 $G_f(D)$ = D-residual graph

While there is a path from s to t in $G_f(D)$ along which flow can be increased

Increase flow along that path

Update $G_f(D)$

D = D/2

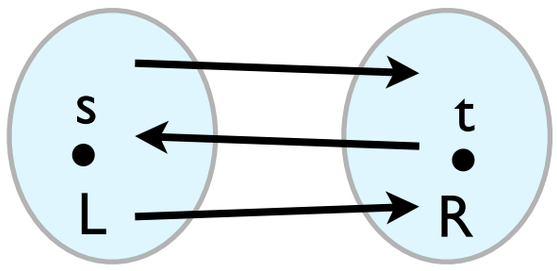
D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

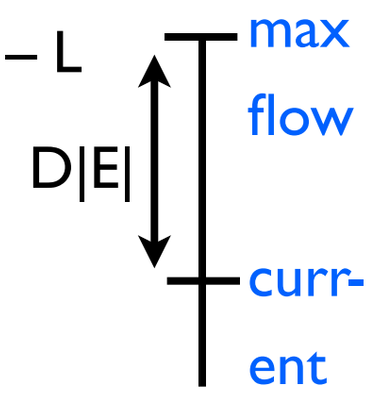
Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Proof: Let L = nodes reachable from s in $G_f(D)$ and let R = rest of nodes = V – L

#edges in $G_f(D)$ in the (L, R) cut = 0





Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

D scaling phase

2 $G_f(D)$ = D-residual graph

While there is a path from s to t in $G_f(D)$ along which flow can be increased

Increase flow along that path

Update $G_f(D)$

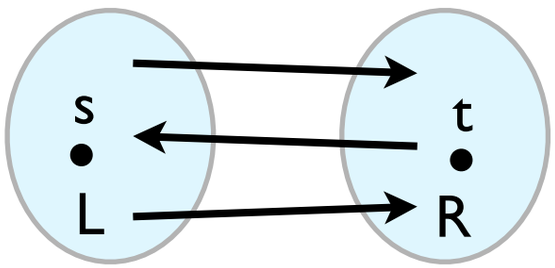
D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

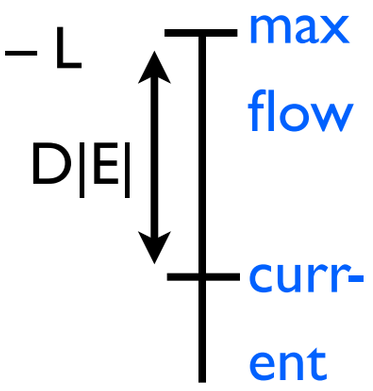
Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Proof: Let L = nodes reachable from s in $G_f(D)$ and let R = rest of nodes = V – L



#edges in $G_f(D)$ in the (L, R) cut = 0

#edges in G_f in the (L,R) cut <= |E|



Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

D scaling phase

$G_f(D)$ = D-residual graph

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

Increase flow along that path

Update $G_f(D)$

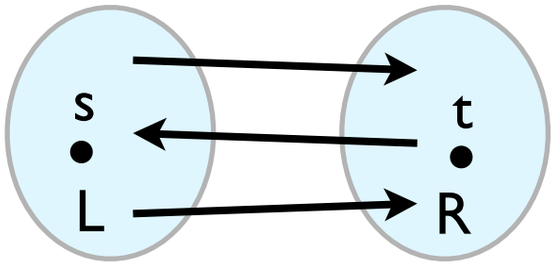
D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed $1 + \log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

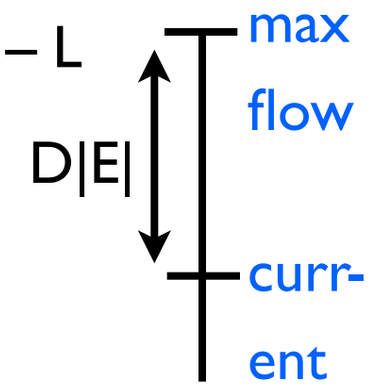
Proof: Let L = nodes reachable from s in $G_f(D)$ and let R = rest of nodes = V – L



#edges in $G_f(D)$ in the (L, R) cut = 0

#edges in G_f in the (L,R) cut <= |E|

Capacity of each such edge < D



Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

D scaling phase

2 $G_f(D)$ = D-residual graph

While there is a path from s to t in $G_f(D)$ along which flow can be increased

Increase flow along that path

Update $G_f(D)$

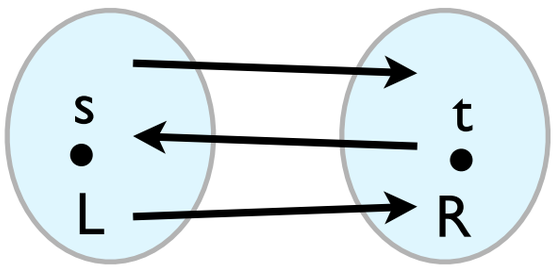
D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Proof: Let L = nodes reachable from s in $G_f(D)$ and let R = rest of nodes = V – L

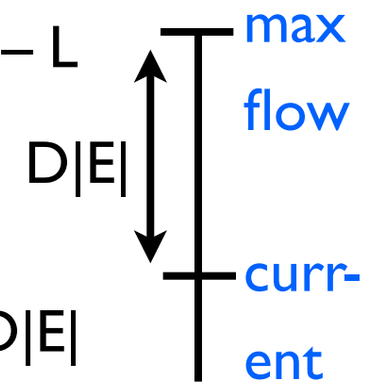


#edges in $G_f(D)$ in the (L, R) cut = 0

#edges in G_f in the (L,R) cut <= |E|

Capacity of each such edge < D

Thus, size(max flow) <= capacity(L,R) <= size(f) + D|E|



Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

  D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Property 3: For any D, #iterations of loop 2 in the D-scaling phase <= 2|E|

Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

  D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Property 3: For any D, #iterations of loop 2 in the D-scaling phase <= 2|E|

Proof: After **previous** (2D-scaling) phase, size(max flow) <= size(current flow) + 2D|E|

Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

  D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Property 3: For any D, #iterations of loop 2 in the D-scaling phase <= 2|E|

Proof: After **previous** (2D-scaling) phase, size(max flow) <= size(current flow) + 2D|E|

Each iteration of loop 2 increases flow size by at least D

Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

  D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Property 3: For any D, #iterations of loop 2 in the D-scaling phase <= 2|E|

Proof: After **previous** (2D-scaling) phase, size(max flow) <= size(current flow) + 2D|E|

Each iteration of loop 2 increases flow size by at least D

Therefore, at most 2|E| iterations in the D-scaling phase

Capacity Scaling: Running Time

C_{max} = max capacity edge. Start with D = C_{max}

Start with zero flow

1 While D >= 1, repeat:

  $G_f(D)$ = D-residual graph **D scaling phase**

2 While there is a path from s to t in $G_f(D)$ along which flow can be increased

    Increase flow along that path

    Update $G_f(D)$

  D = D/2

D-Residual Graph: Subgraph of residual graph with only edges with capacity >= D

Property 1: While loop 1 is executed 1 + $\log_2 C_{max}$ times

Property 2: At the end of a D-scaling phase, size(max flow) <= size(current flow) + D|E|

Property 3: For any D, #iterations of loop 2 in the D-scaling phase <= 2|E|

Total Running Time: $O(|E|^2(1 + \log_2 C_{max}))$

(Recall: Time to find a flow path in a residual graph = O(|E|))

How to improve the efficiency?

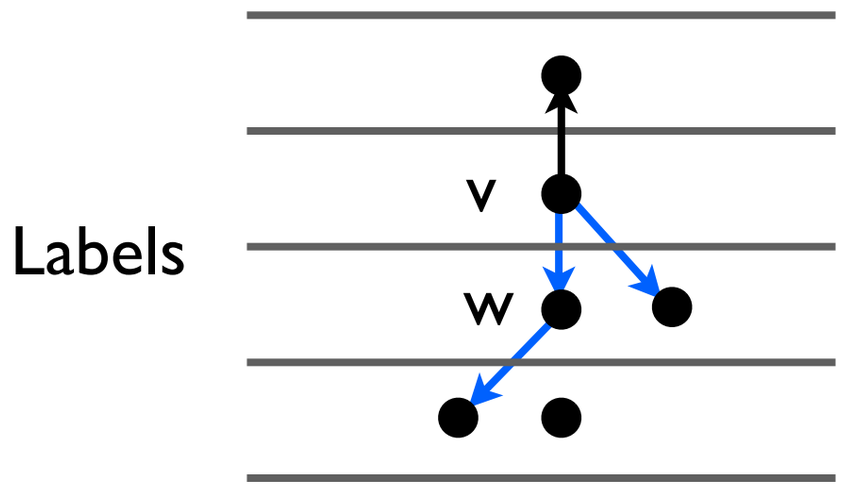
- Ford-Fulkerson Style Algorithms:
  - Edmonds Karp
  - Capacity Scaling

- Preflow-Push

Preflow-Push

Main Idea:

- Each node has a label, which is a potential
- Route flow from high to low potential



Idea: Route flow along blue edges

Preflows

Preflow: A function f: E $\rightarrow$ R is a preflow if:

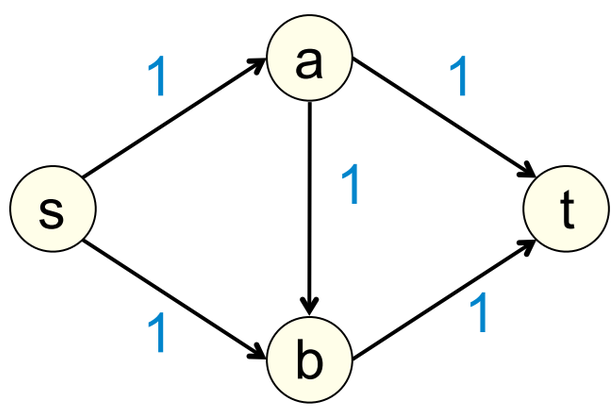
1. **Capacity Constraints:** 0 <= f(e) <= c(e)
2. Instead of conservation constraints:

$$\sum_{e \ into \ v} f(e) - \sum_{e \ out \ of \ v} f(e) \geq 0$$

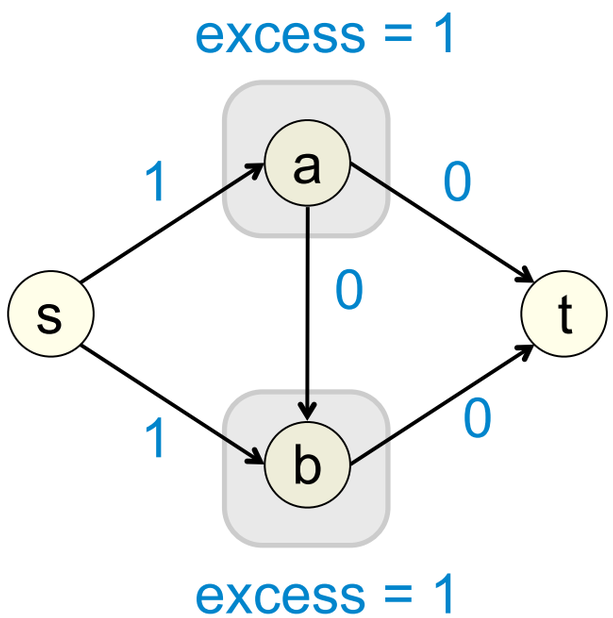
Excess(v) = $\sum_{e \ into \ v} f(e) - \sum_{e \ out \ of \ v} f(e)$

Example

G



f



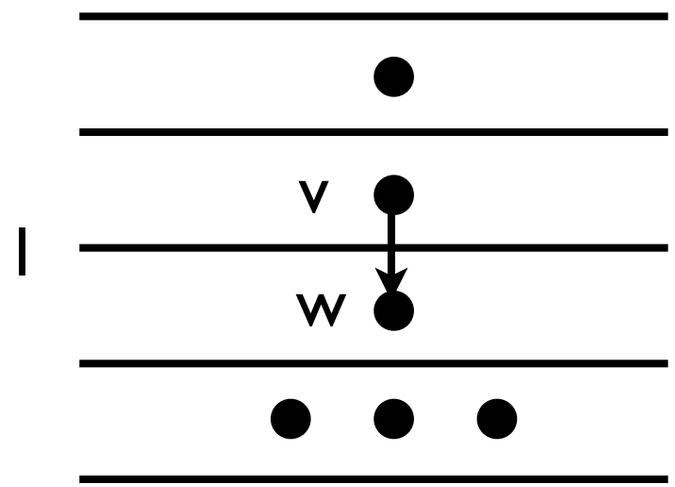
Preflow-Push: Two Operations

Preflow: A function f: E $\rightarrow$ R is a preflow if:

1. **Capacity Constraints:** 0 <= f(e) <= c(e)
2. Instead of conservation constraints:

$$\sum_{e\ into\ v} f(e) - \sum_{e\ out\ of\ v} f(e) \geq 0$$

Excess(v) = $\sum_{e\ into\ v} f(e) - \sum_{e\ out\ of\ v} f(e)$



Labeling h assigns a non-negative integer label h(v) to all v in V

Push(v, w): Applies if excess(v) > 0, h(w) < h(v), (v, w) in E_f

q = min(excess(v), c_f(v, w))

Add q to f(v, w)

Relabel(v): Applies if excess(v) > 0, for all w s.t (v, w) in E_f, h(w) >= h(v)

Increase l(v) by 1

Pre-Flow Push: The Algorithm

Start with labeling: $h(s) = n, h(t) = 0, h(v) = 0$, for all other v
Start with preflow f: $f(e) = c(e)$ for $e = (s, v)$, $f(e) = 0$, for all other edges e

While there is a node (other than t) with positive excess
  Pick a node v with excess(v) > 0
  If there is an edge (v, w) in E_f such that push(v, w) can be applied
    Push(v, w)
  Else
    Relabel(v)

Push(v, w): Applies if excess(v) > 0, $h(w) < h(v)$, (v, w) in E_f
  $q = \min(\text{excess}(v), c_f(v, w))$
  Add q to f(v, w)

Relabel(v): Applies if excess(v) > 0, for all w s.t (v, w) in E_f, $h(w) >= h(v)$
  Increase h(v) by 1