MONASH
INFORMATION
TECHNOLOGY

FIT2004
Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Companez, Aamir Cheema, Arun Konagurthu and Lloyd Allison

Faculty of Information Technology, Monash University

Ready?

Agenda

- Network Flow

Agenda

- Network Flow

- The maximum flow problem
- The residual network
- Path augmentation multiple BFS to augment path

Agenda

- Network Flow

- The maximum flow problem
- The residual network
- Path augmentation

Ford-Fulkerson Method

Agenda

- Network Flow

- The maximum flow problem
- The residual network
- Path augmentation

$\left.\right\}$ **Ford-Fulkerson Method**

- Min-cut Max-flow Theorem

Agenda

- Network Flow

- The maximum flow problem
- The residual network
- Path augmentation

Ford-Fulkerson Method

- Min-cut Max-flow Theorem

- Then we have Bipartite Graph

Agenda

- Network Flow

- The maximum flow problem
- The residual network
- Path augmentation

Ford-Fulkerson Method with extra optimization from FIT3155

- Min-cut Max-flow Theorem

- Then we have Bipartite Graph
  - Matching optimally =)

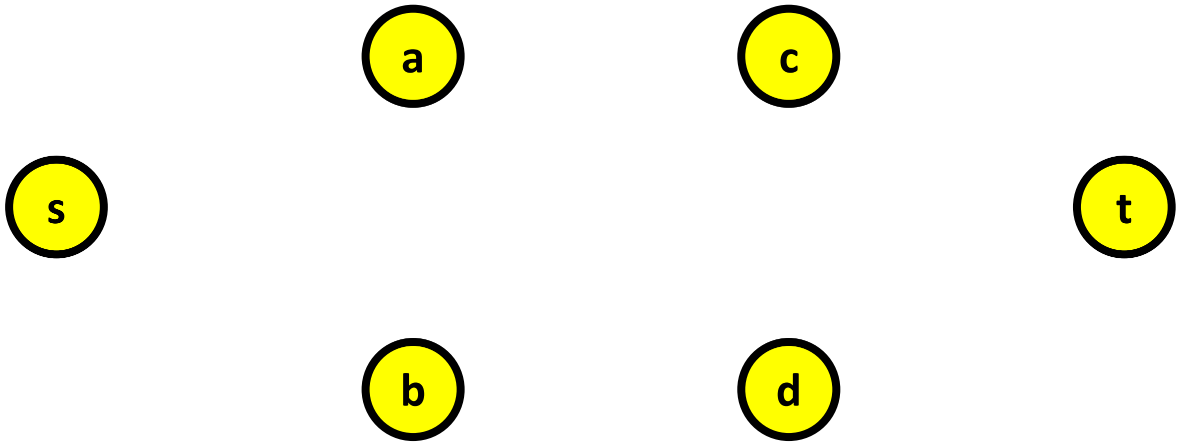
Let us begin…

Flow Network

Transfer of content

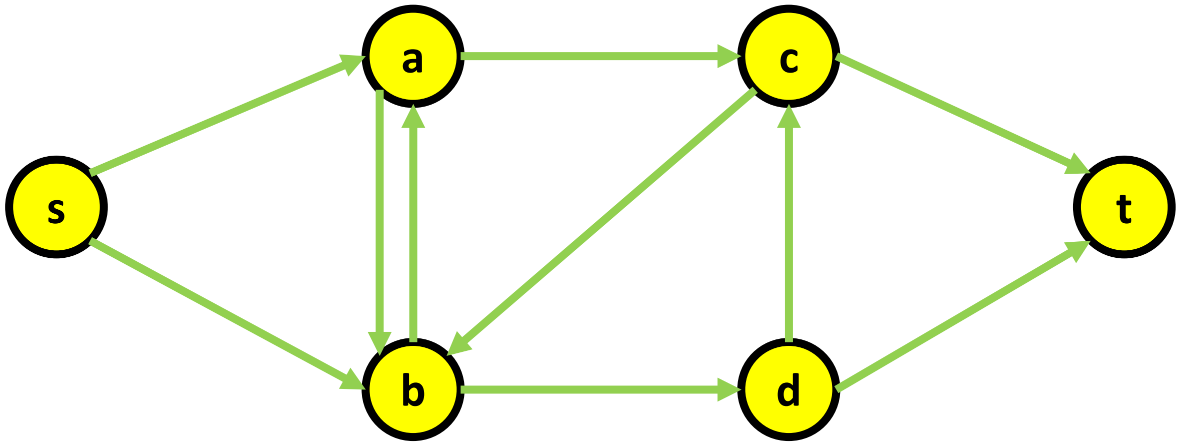
- What is it?

- What is it?
  - It is a graph
    - With vertices



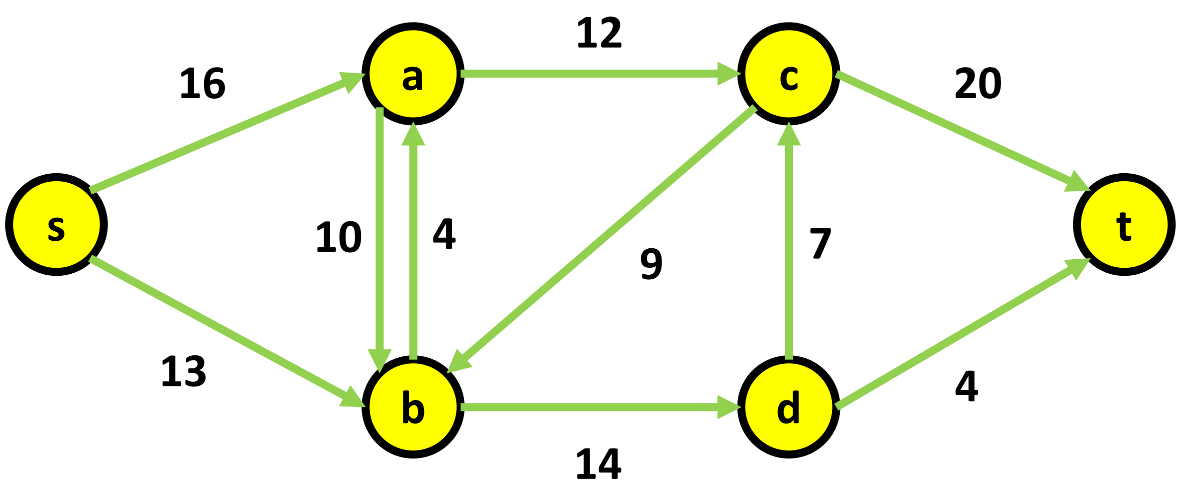
Flow Network
Transfer of content

- What is it?
  - It is a graph
    - With vertices
    - With edges (directed)



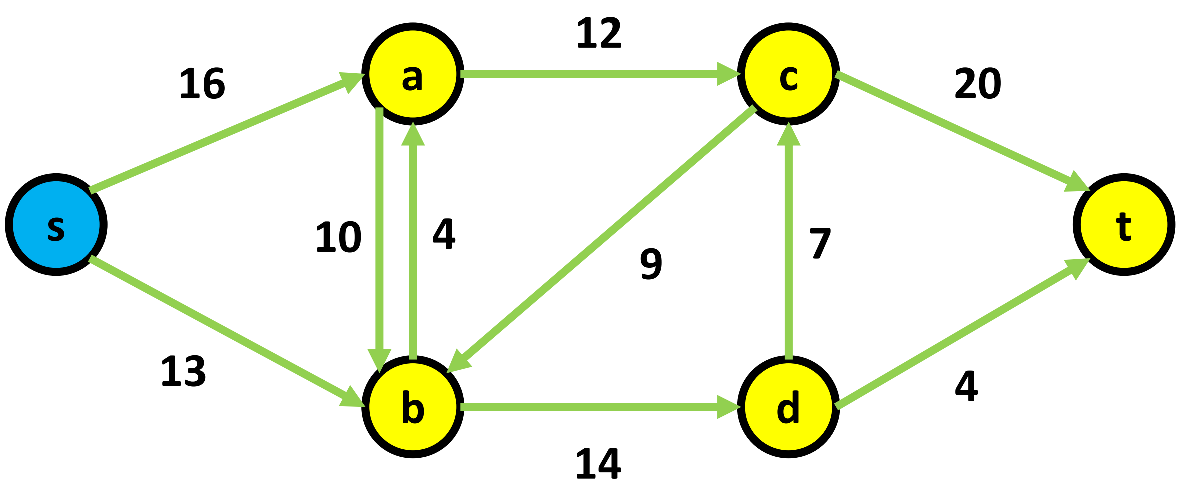
Flow Network
Transfer of content

- What is it?
  - It is a graph
    - With vertices
    - With edges (directed and weighted)

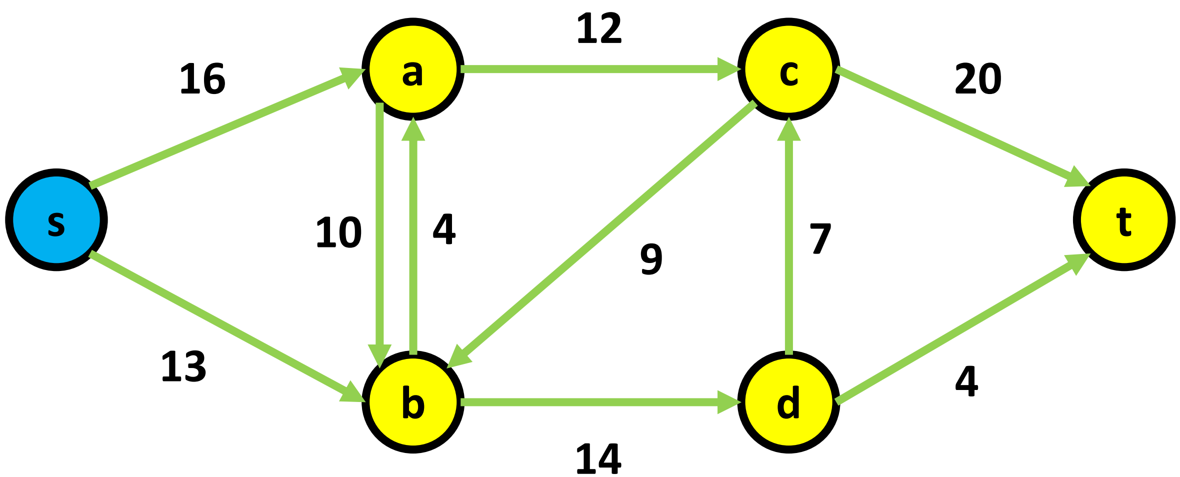


MONASH University

- What is it?
  - It is a graph
    - With vertices
    - With edges (directed and weighted)
    - A vertex without incoming edges



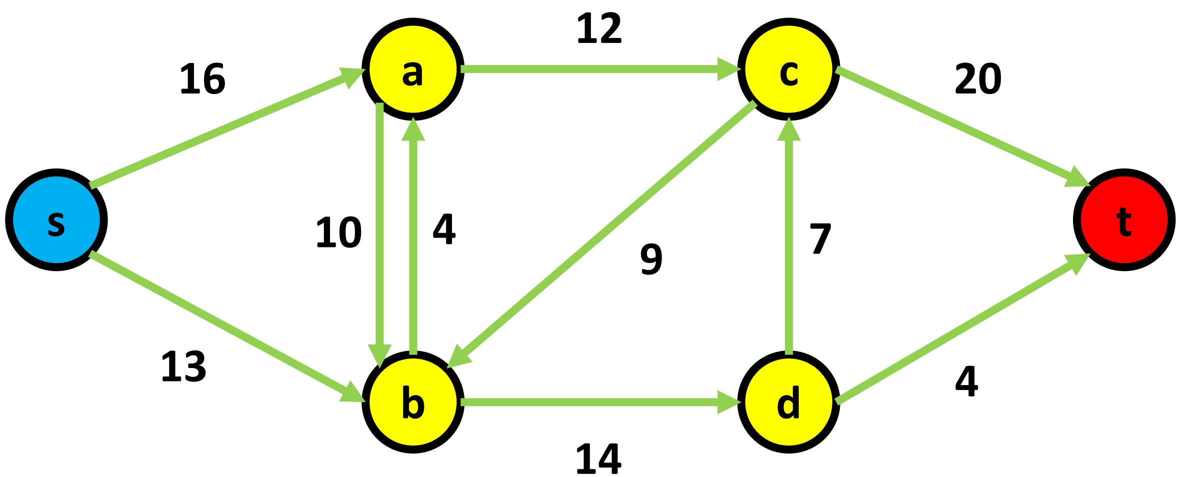
- What is it?
  - It is a graph
    - With vertices
    - With edges (directed and weighted)
    - A vertex without incoming edges know as the source



Flow Network

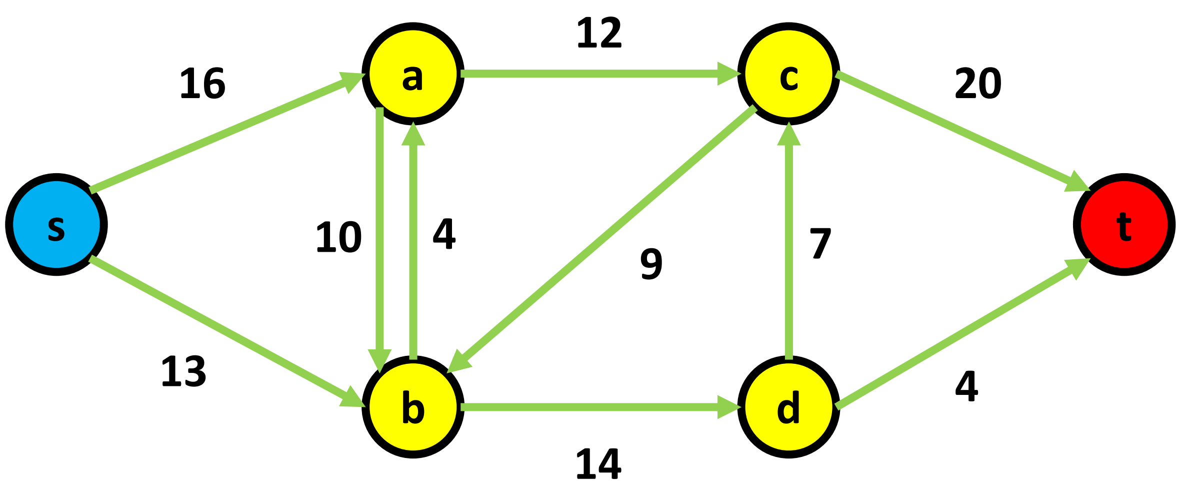
Transfer of content

- What is it?
  - It is a graph
    - With vertices
    - With edges (directed and weighted)
    - A vertex without incoming edges know as the source
    - A vertex without outgoing edges



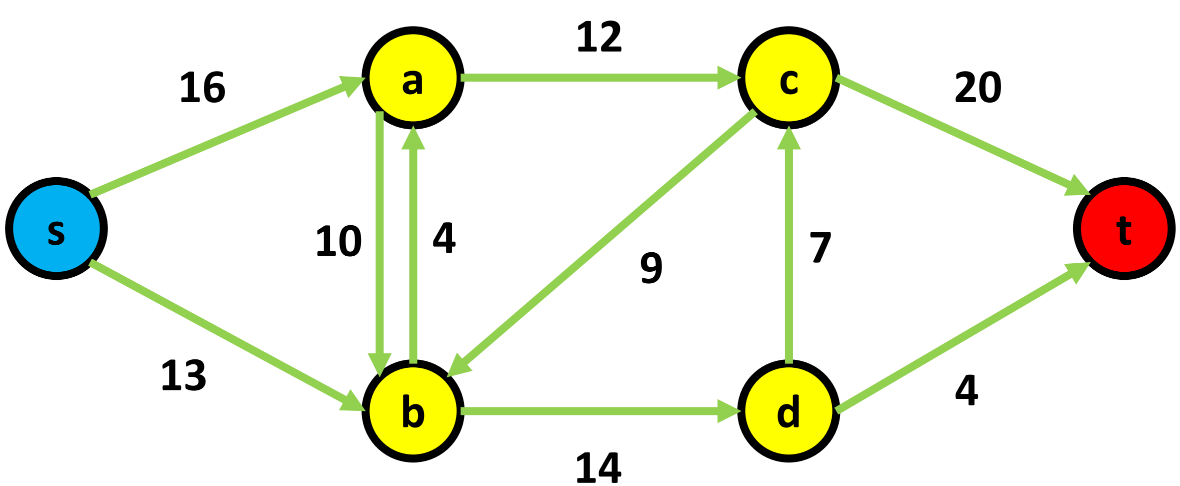
- What is it?
  - It is a graph
    - With vertices
    - With edges (directed and weighted)
    - A vertex without incoming edges know as the source
    - A vertex without outgoing edges known as the target/ destination

sink



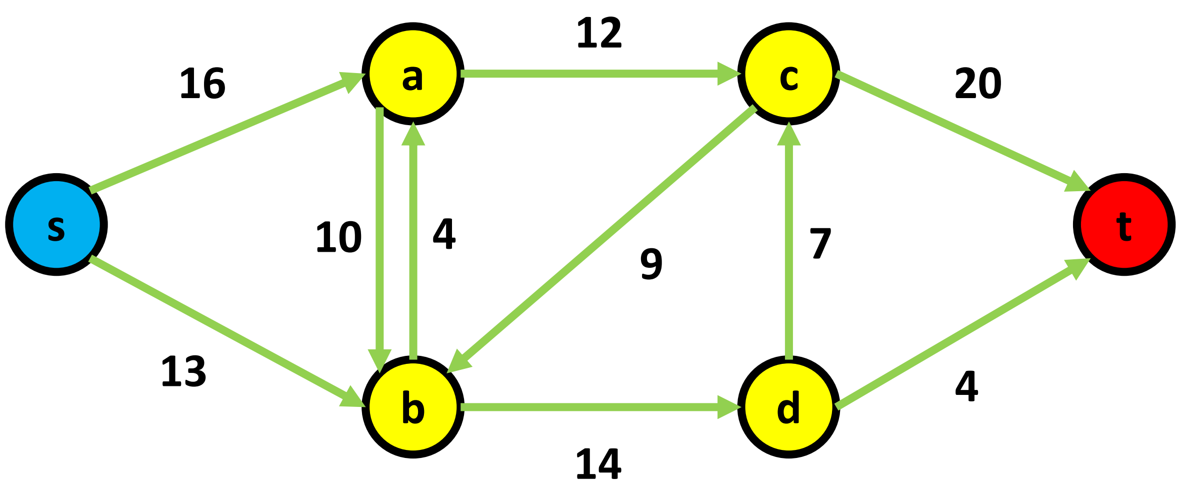
Flow Network
Transfer of content

- What is it?
  - It is a graph
    - With vertices
    - With edges (directed and weighted non-negative known as capacity)
    - A vertex without incoming edges know as the source
    - A vertex without outgoing edges known as the target/ destination

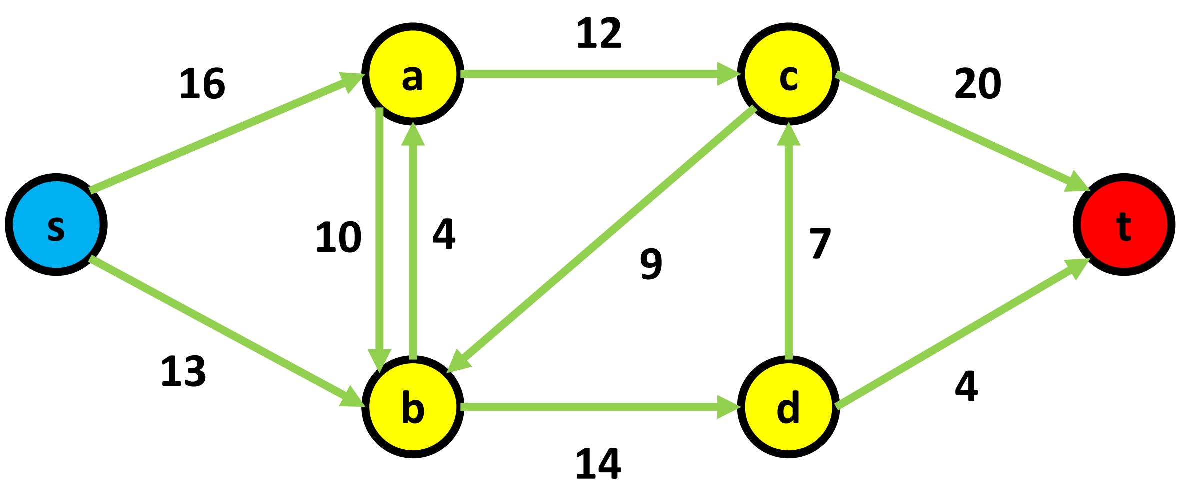


MONASH University

- What is it?
  - Explore the real world problem of transfer
    - From source
    - To destination



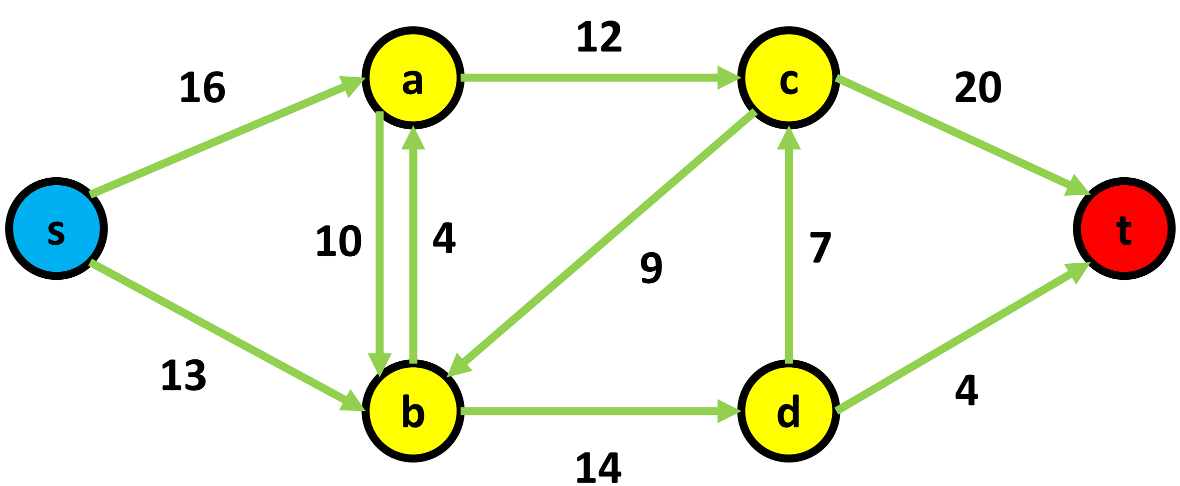
- What is it?
  - Explore the real world problem of transfer
    - From source
    - To destination
    - Within the capacity (which can be bottlenecks)



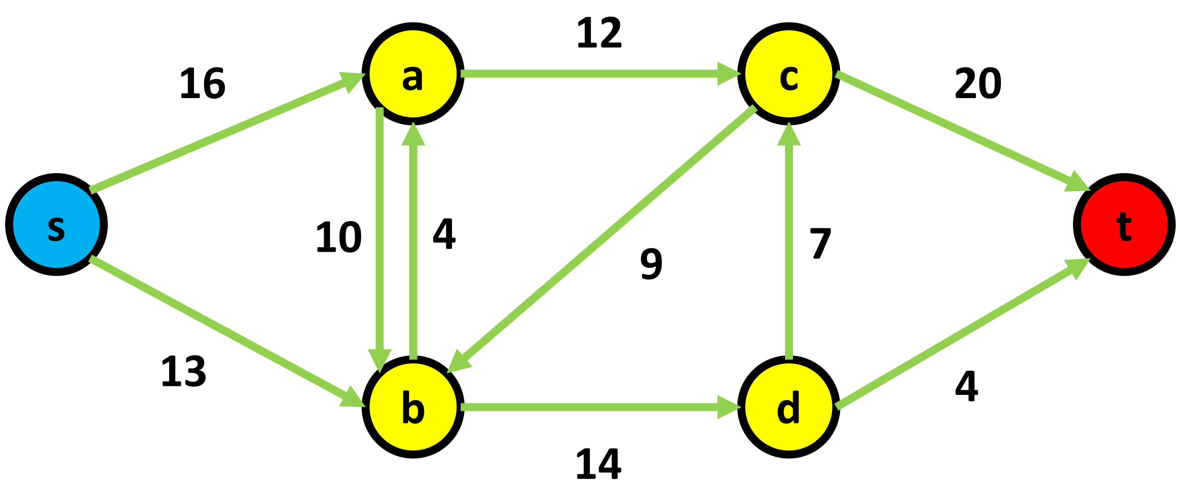
Flow Network

Transfer of content

- What is it?
  – Explore the real world problem of transfer
    - From source
    - To destination
    - Within the capacity (which can be bottlenecks)
    - What is the maximum possible transfer of content?

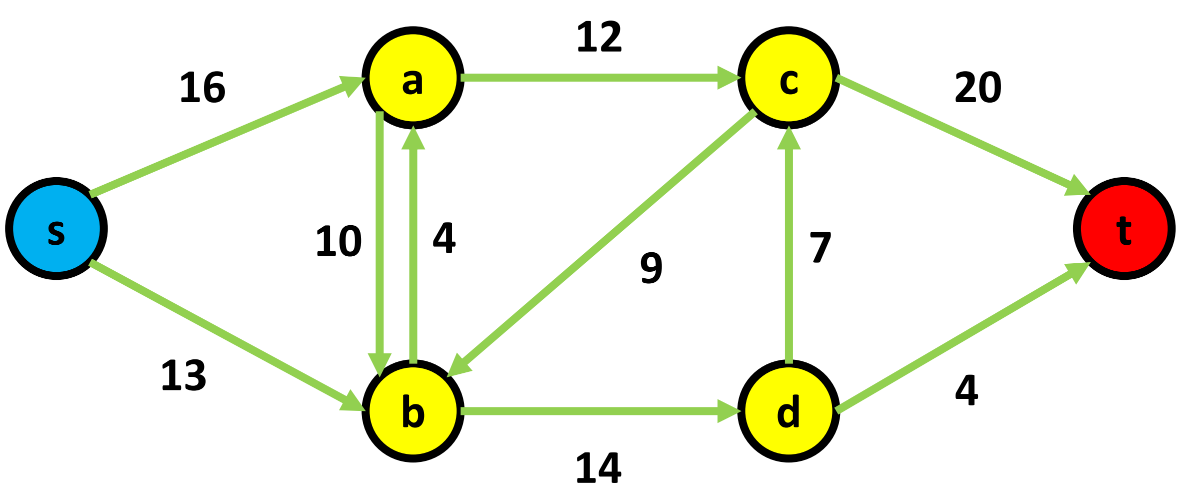


- What is it?
  - Explore the real world problem of transfer
    - From source
    - To destination
    - Within the capacity (which can be bottlenecks)
    - What is the maximum possible transfer of content? Goal here

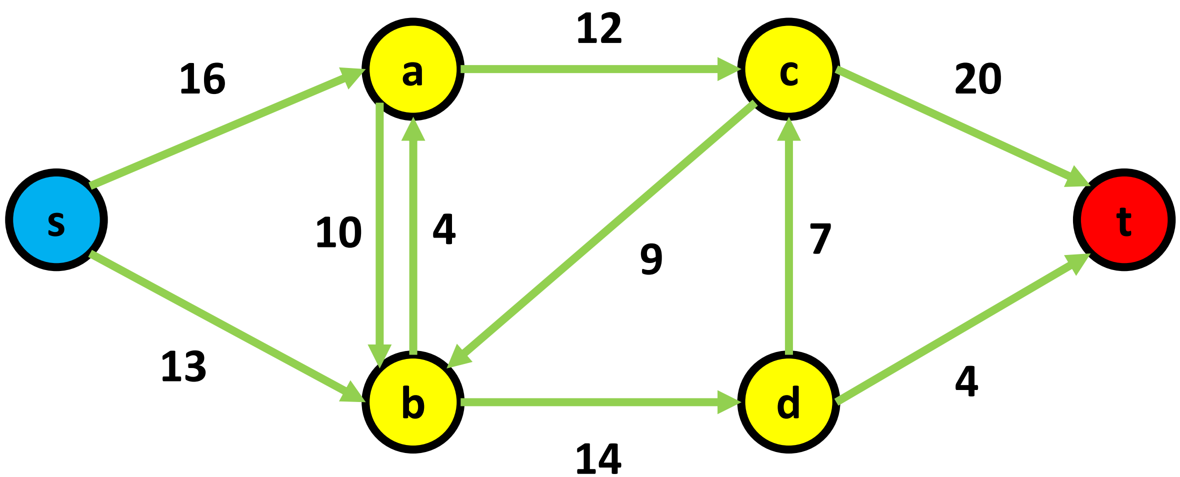


MONASH University

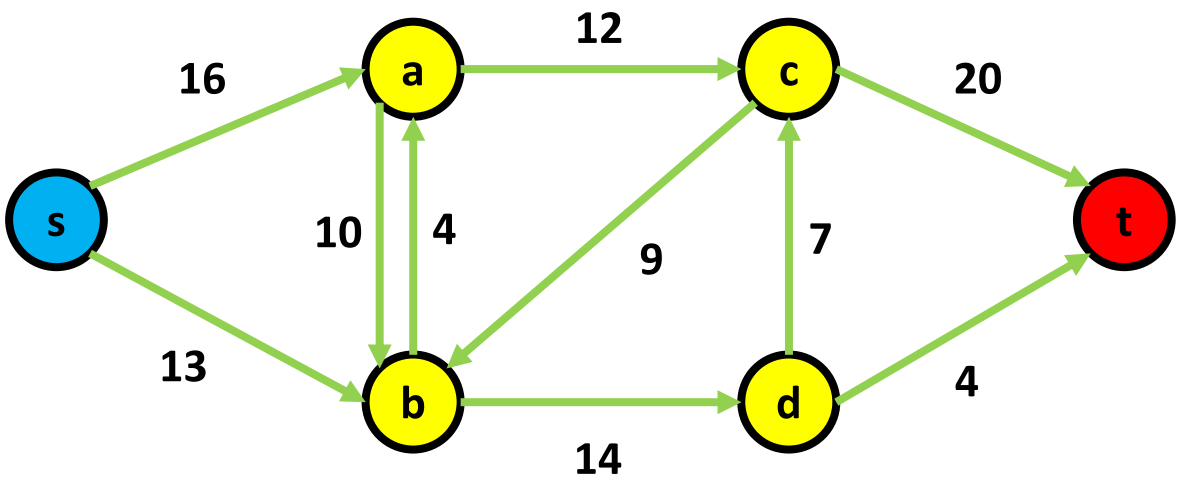
- We can get details from this graph
  - E_in(b) = edges incoming to b
  - E_out(b) = edges outgoing from b



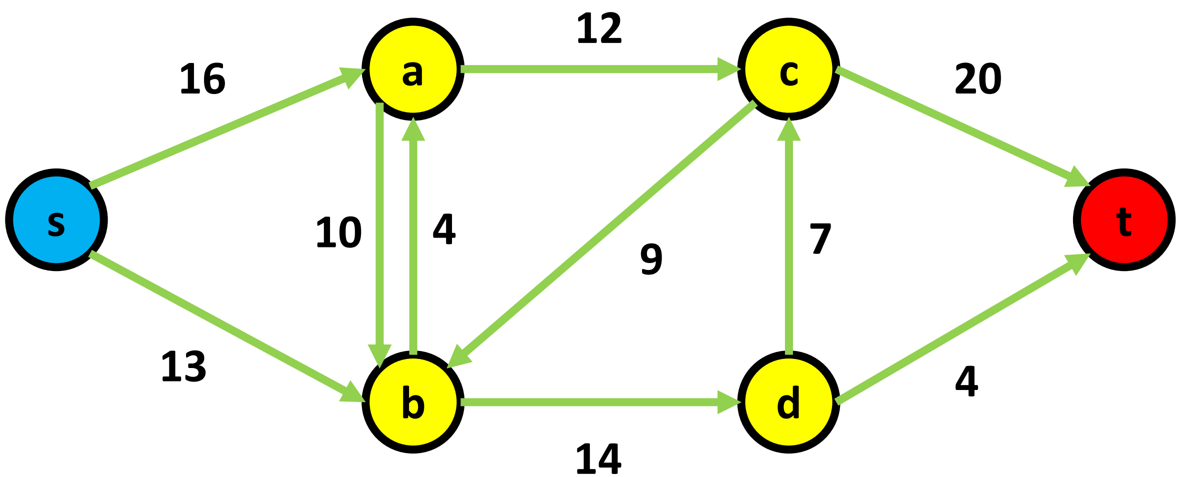
- We can get details from this graph
  - E_in(b) = edges incoming to b = <s,b,13> <a,b,10> <c,b,9>
  - E_out(b) = edges outgoing from b = <b,a,4> <b,d,14>



- Flow network model in real world



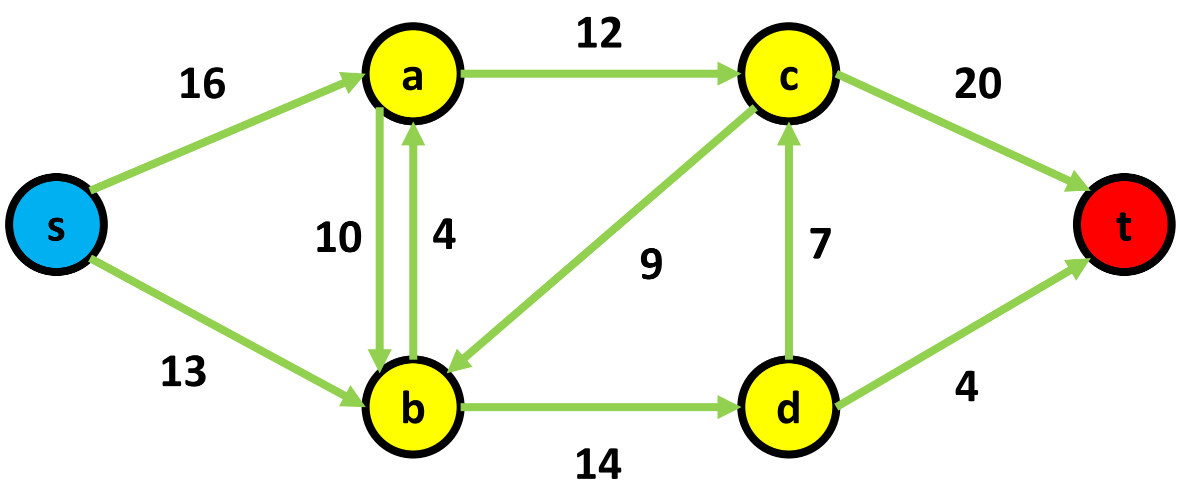
- Flow network model in real world
  - Water flow through pipes
  - Electric through electrical circuits
  - Information flow through communication network



Flow Network

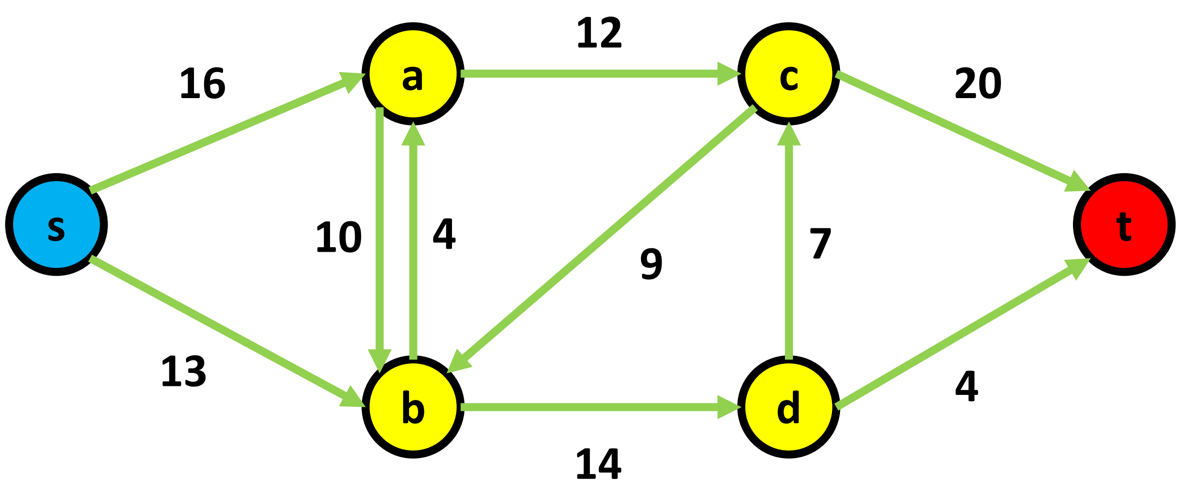
Transfer of content

- Flow network model in real world
  - Water flow through pipes
  - Electric through electrical circuits
  - Information flow through communication network
  - And many more! We can design good networks #engineered



MONASH University

- Flow network model in real world
  - Water flow through pipes
  - Electric through electrical circuits
  - Information flow through communication network
  - And many more! We can design good networks #engineered
  - Was mainly used in WW2 to disrupt enemy supply lines

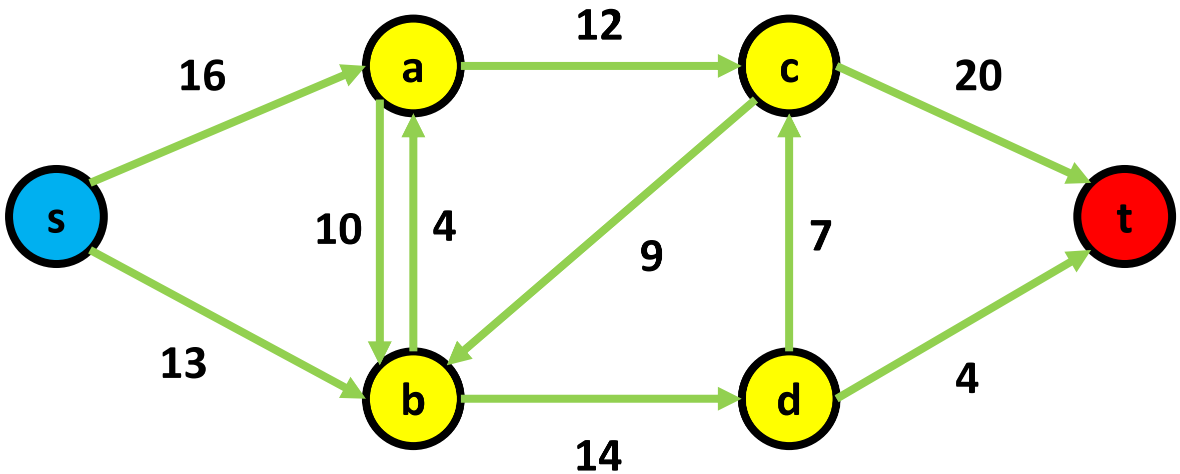


Monash University

Questions?

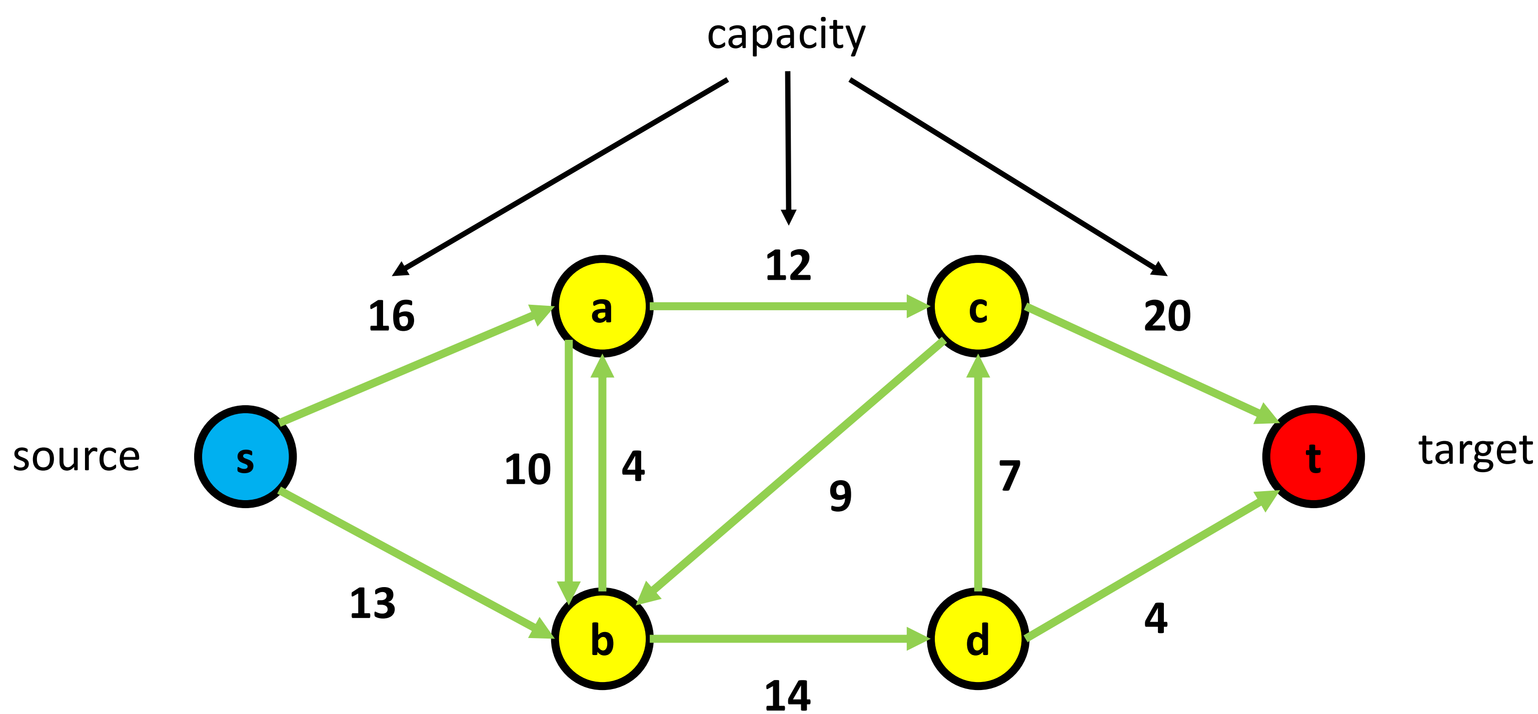
Recap

Of flow network

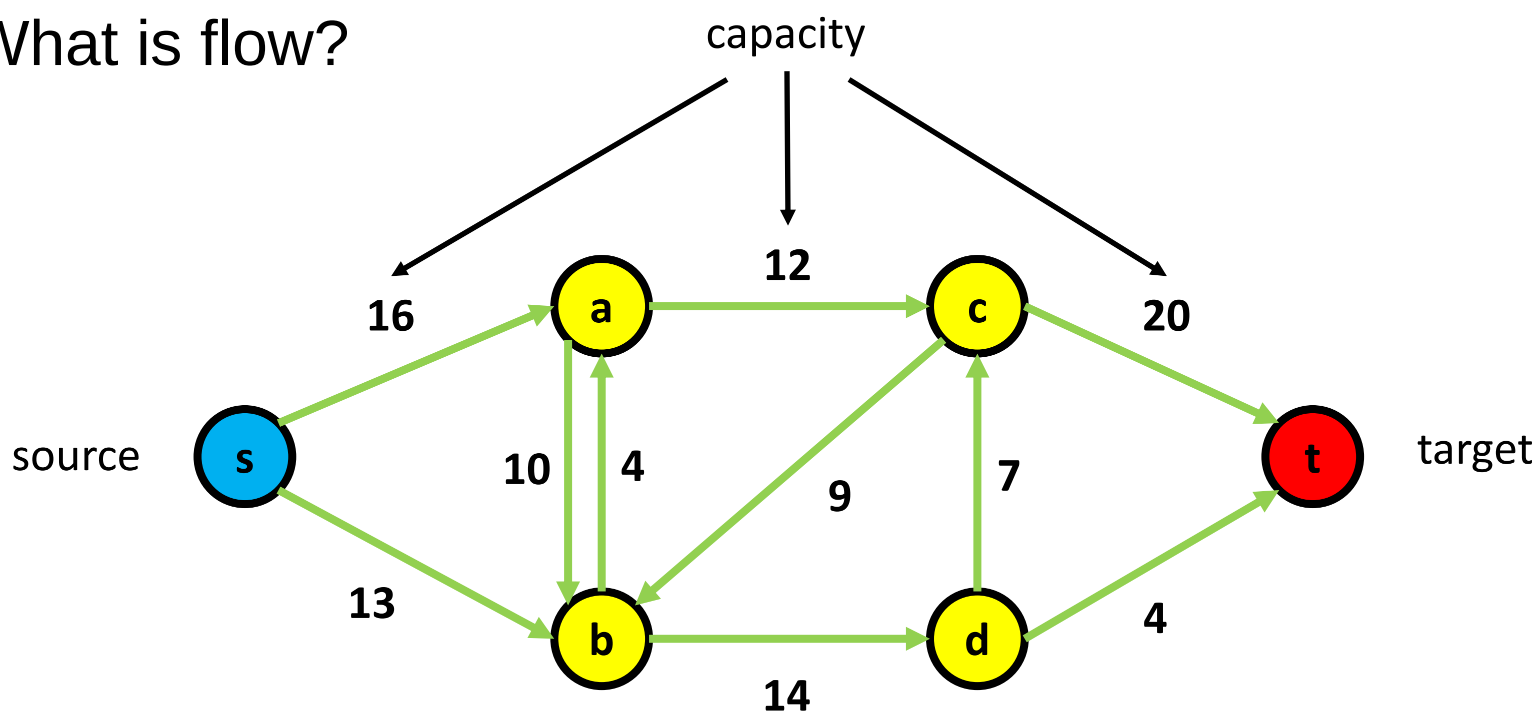


Recap

Of flow network



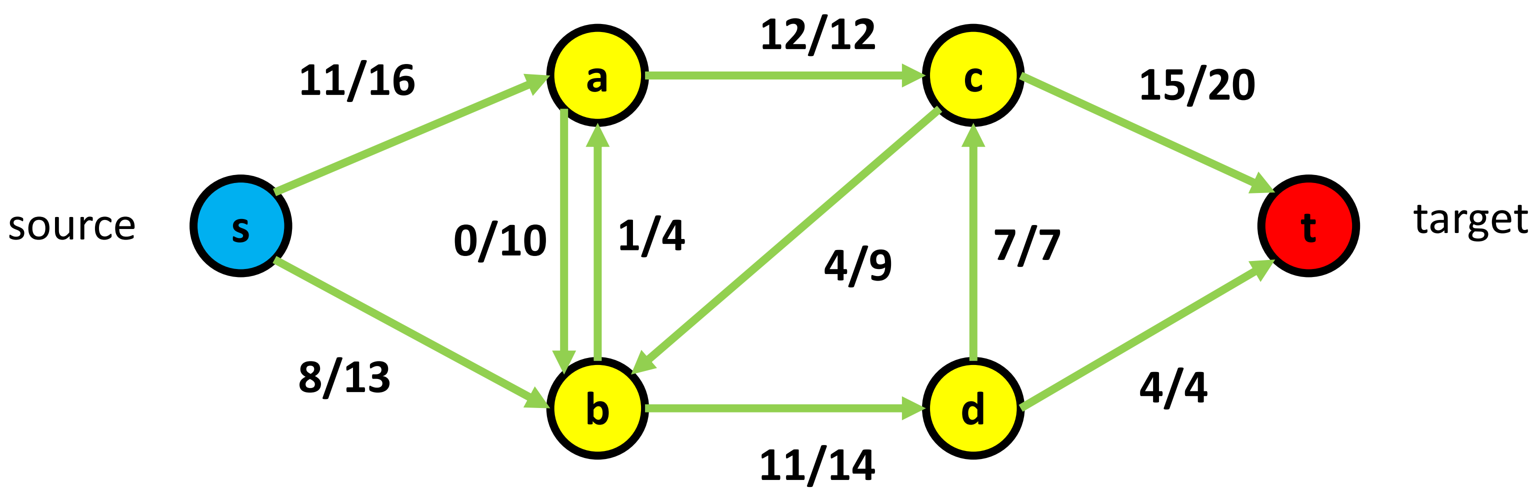
- What is flow?



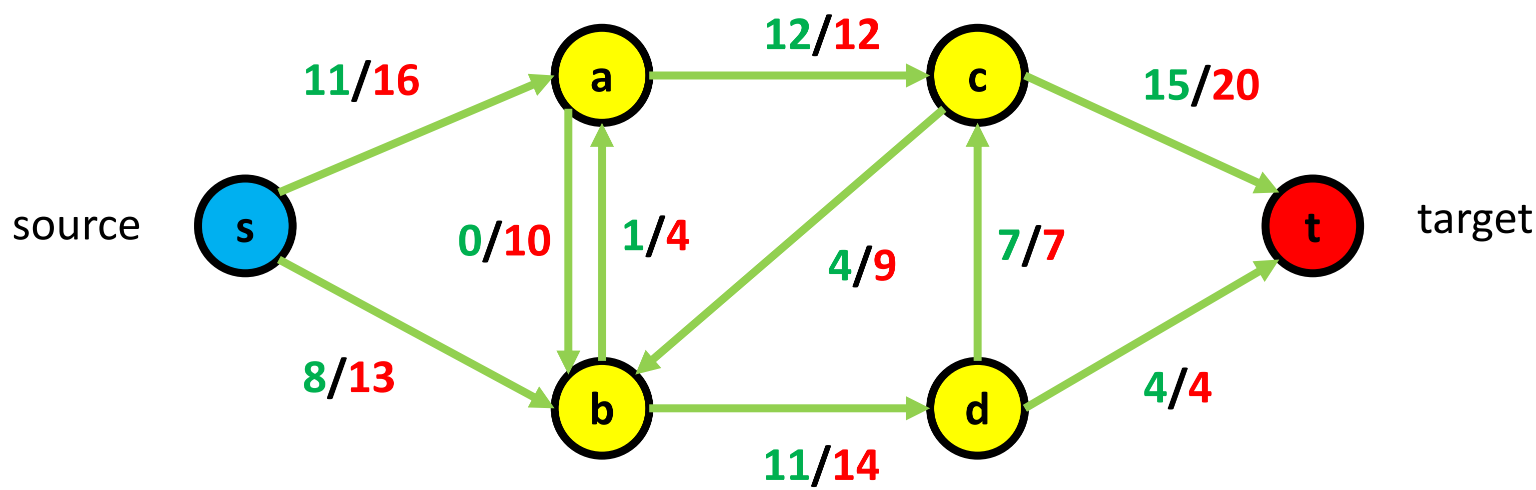
Flow Network

Transfer of content

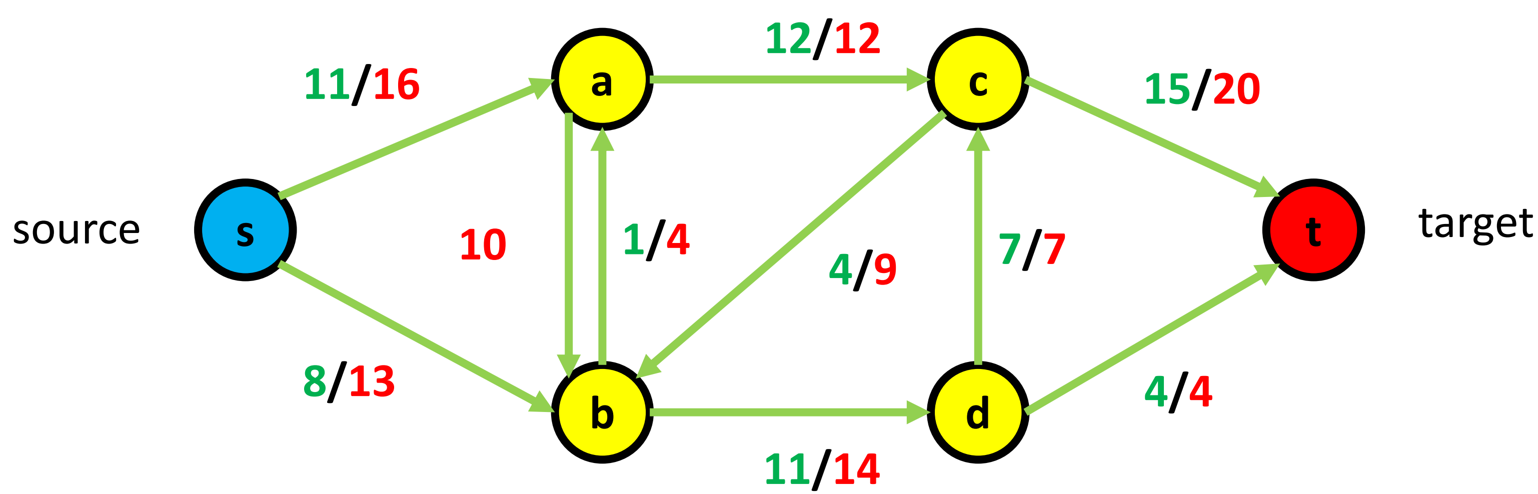
- What is flow?
- What is capacity?



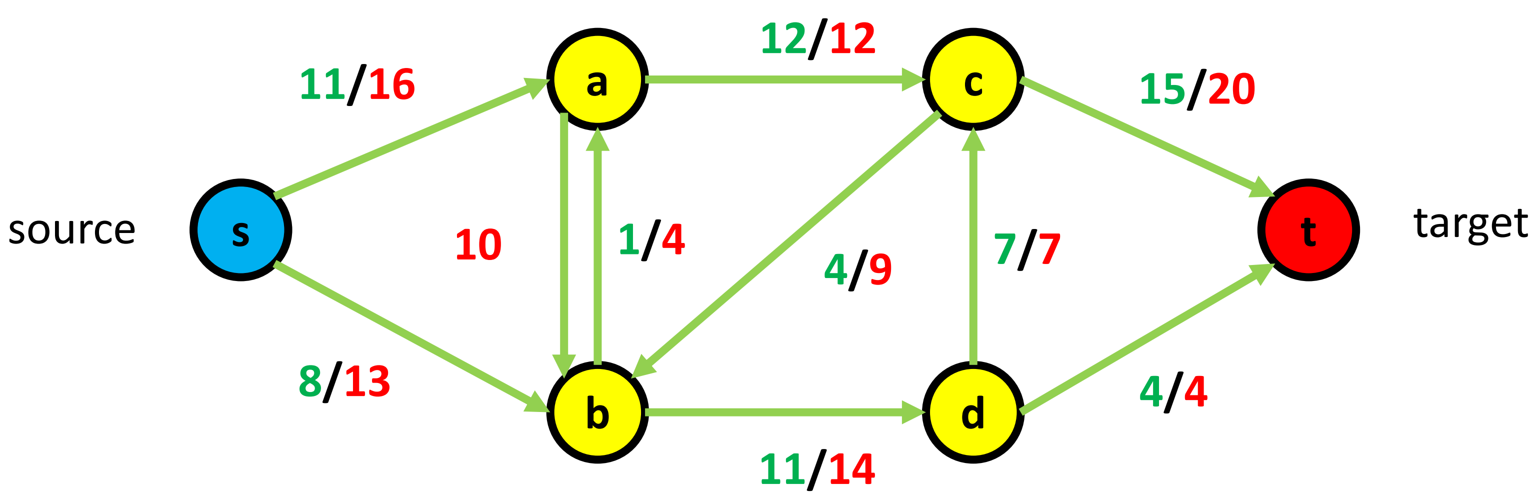
- What is flow?
- What is capacity?



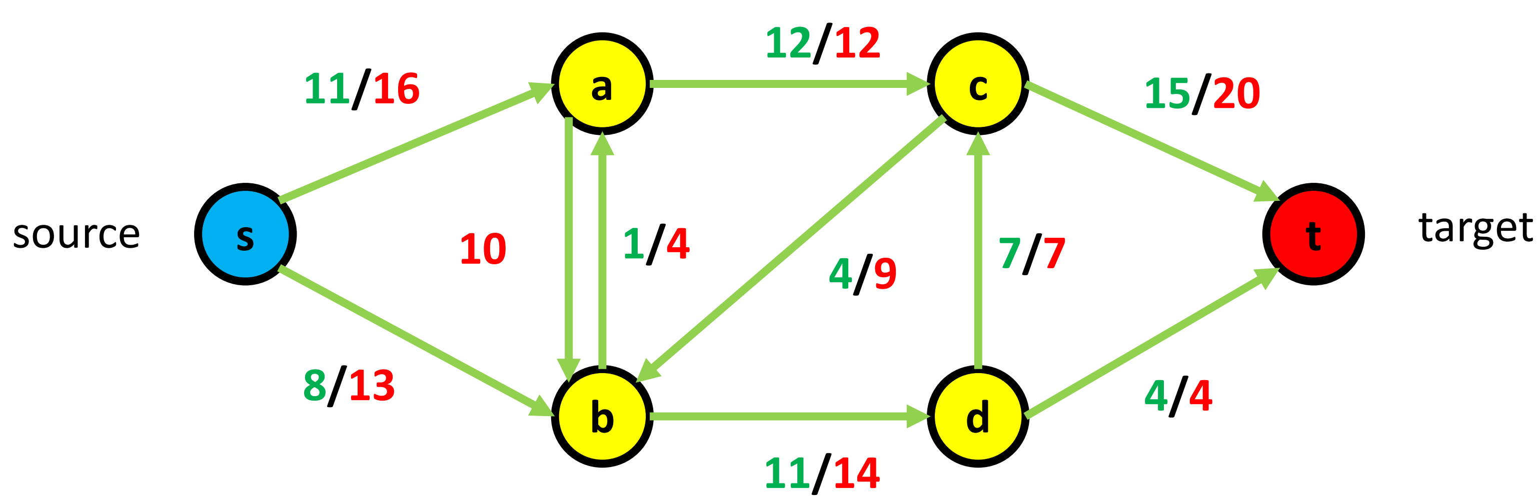
- What is flow?
  - If there is no flow, you can exclude it
  - It is how much material flowing through each edge
- What is capacity?



- Flow constraint property

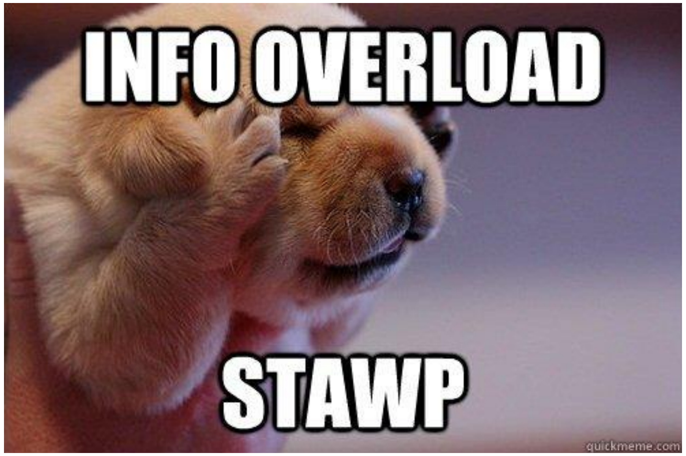


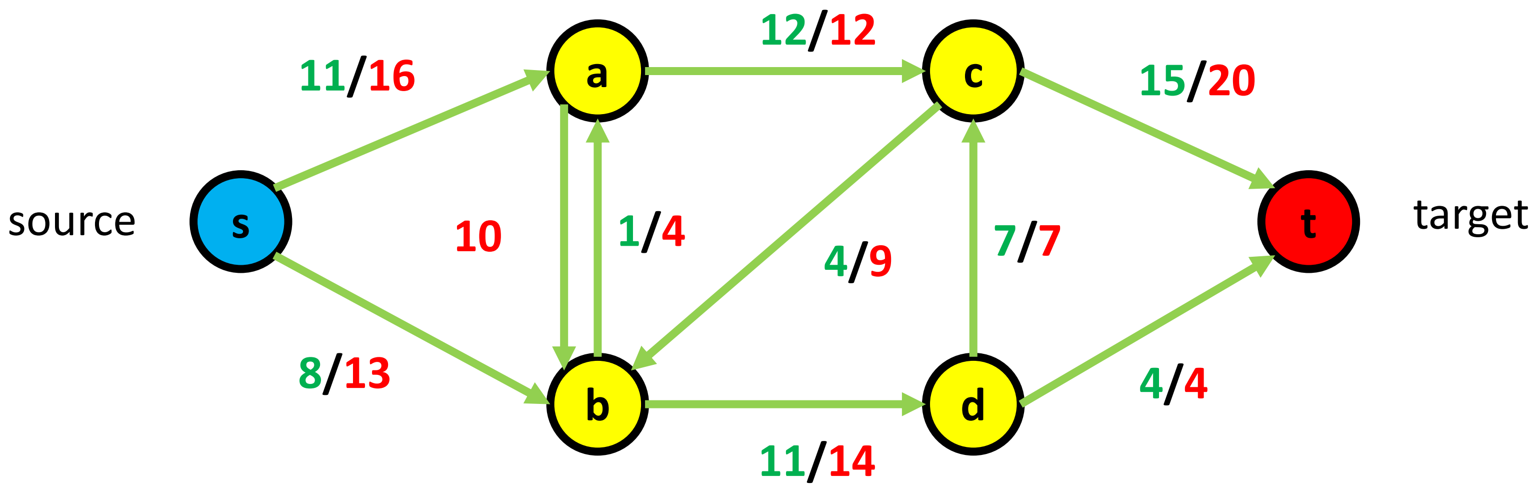
- Flow constraint property
  - For each edge, the flow can't be more than the capacity of the edge



MONASH University

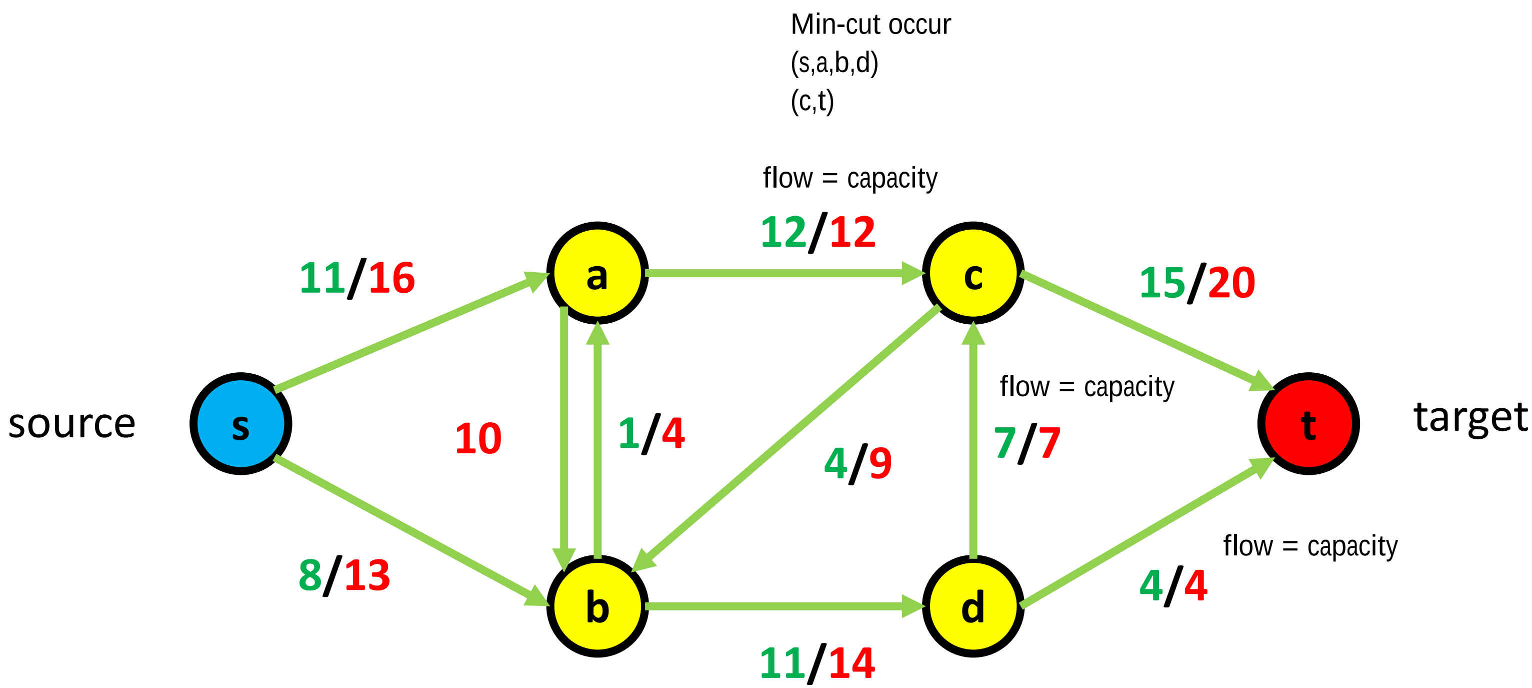
- Flow constraint property
  - For each edge, the flow can't be more than the capacity of the edge
  - In other words, you can't overload



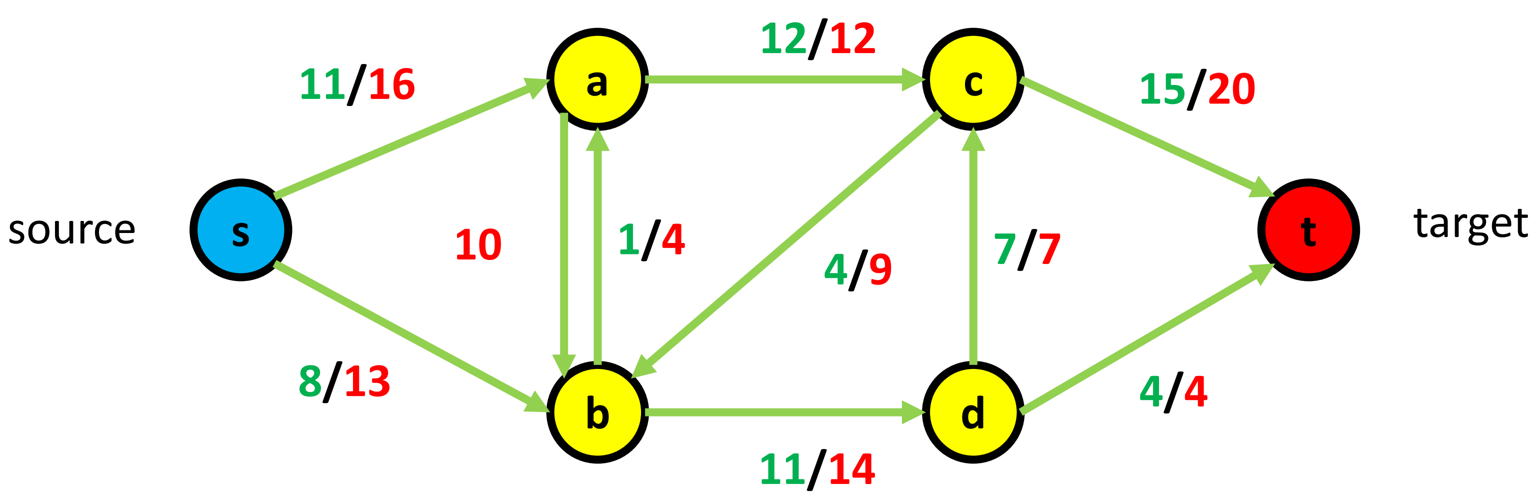


Flow Network
Transfer of content

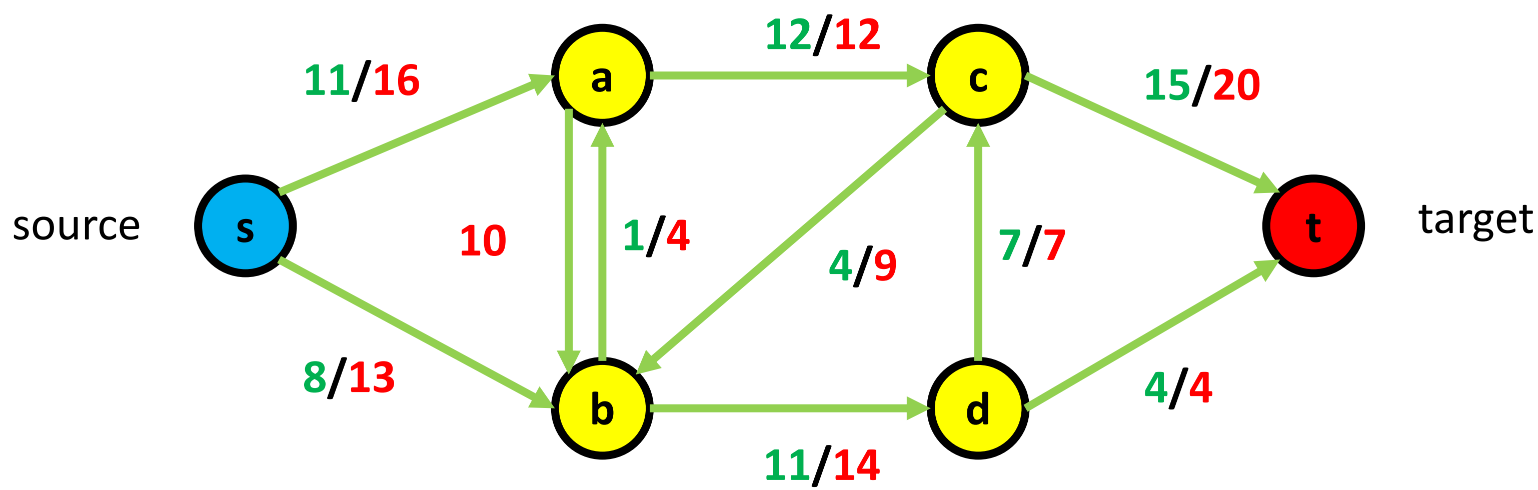
- Flow conservation property



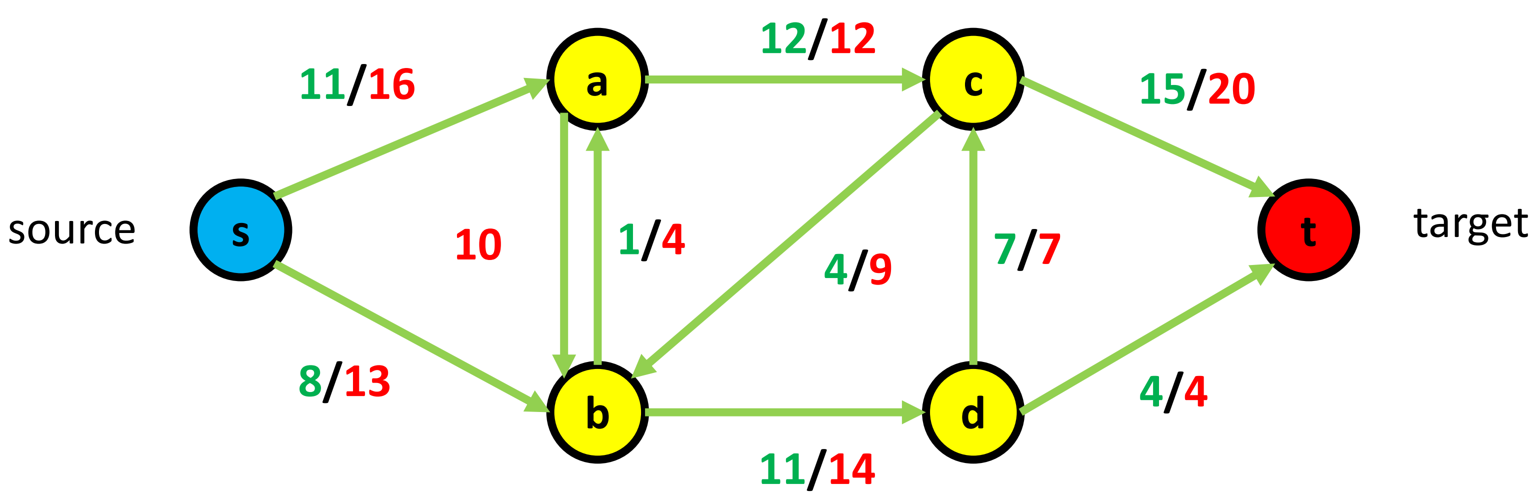
- Flow conservation property
  - For every vertex in the graph (except source and target)
  - incoming flow == outgoing flow



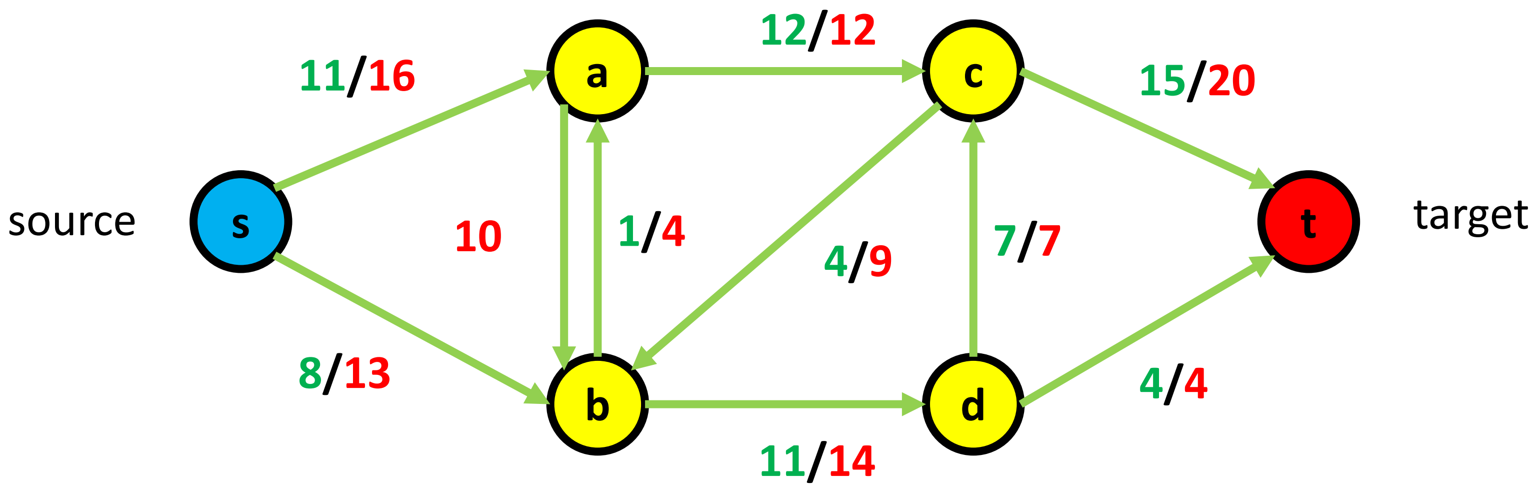
- Flow conservation property
  - For every vertex in the graph (except source and target)
  - total incoming flow == total outgoing flow



- Flow conservation property
  - For every vertex in the graph (except source and target)
  - total incoming flow == total outgoing flow
  - What is the total incoming flow to vertex b?

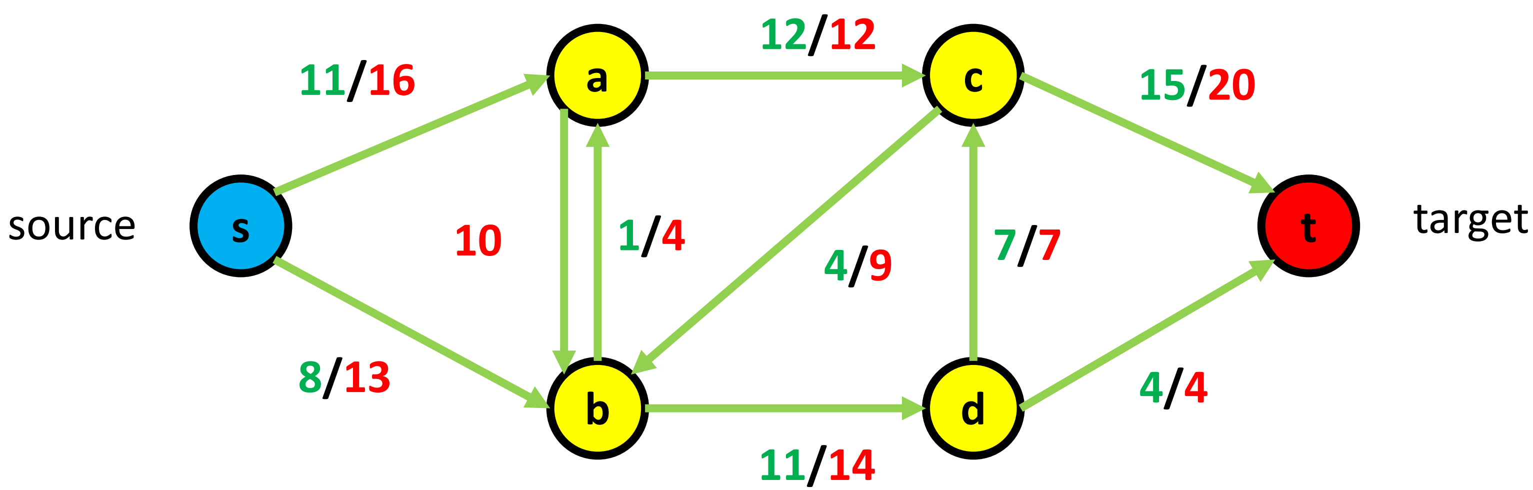


- Flow conservation property
  - For every vertex in the graph (except source and target)
  - total incoming flow == total outgoing flow
  - What is the total incoming flow to vertex b? 12

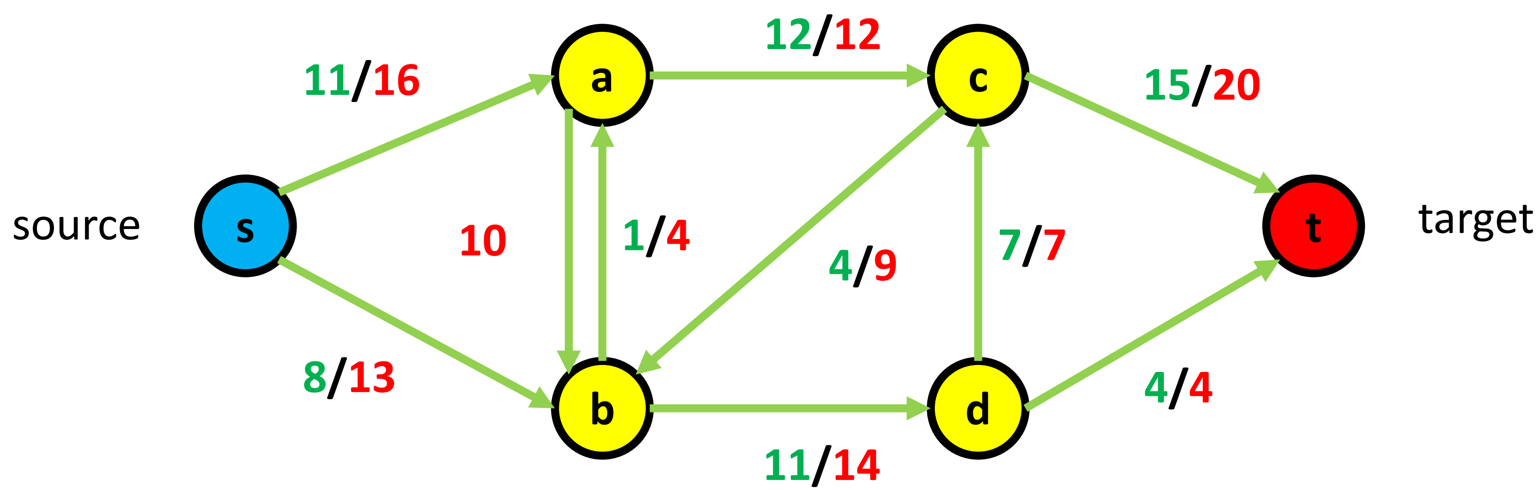


MONASH University

- Flow conservation property
  - For every vertex in the graph (except source and target)
  - total incoming flow == total outgoing flow
  - What is the total incoming flow to vertex b? 12
  - What is the total outgoing flow to vertex b?

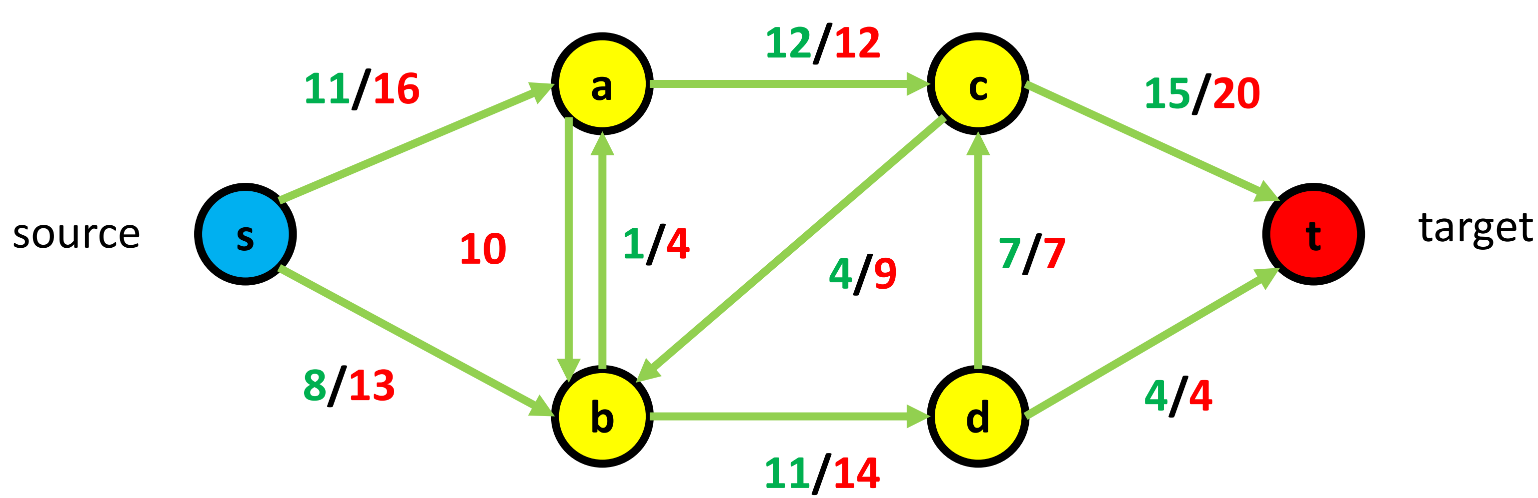


- Flow conservation property
  - For every vertex in the graph (except source and target)
  - total incoming flow == total outgoing flow
  - What is the total incoming flow to vertex b? 12
  - What is the total outgoing flow to vertex b? 12



MONASH University

- Capacity constraint
- Flow conservation property

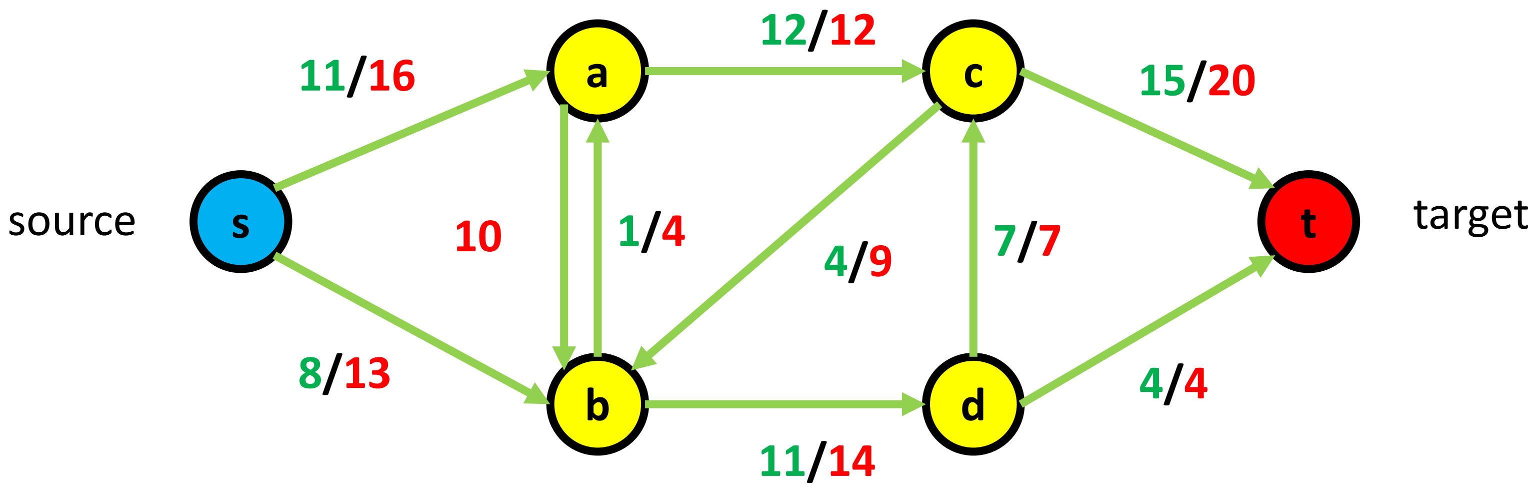


Questions?

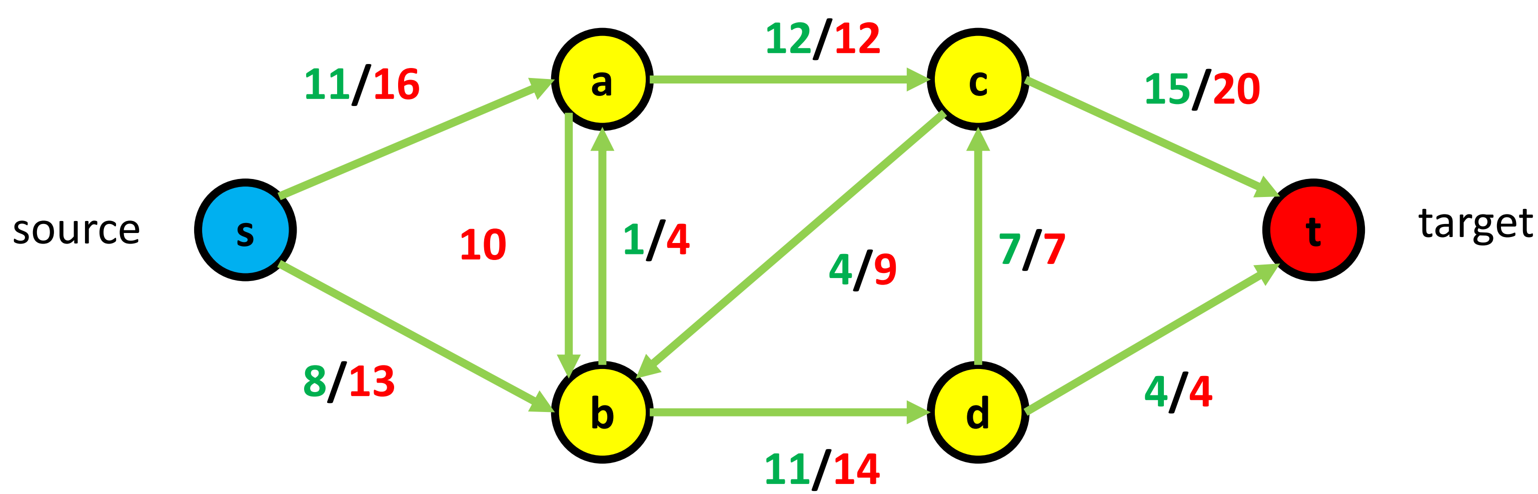
Maximum-Flow Problem

Best network?

- What is the flow of the network?

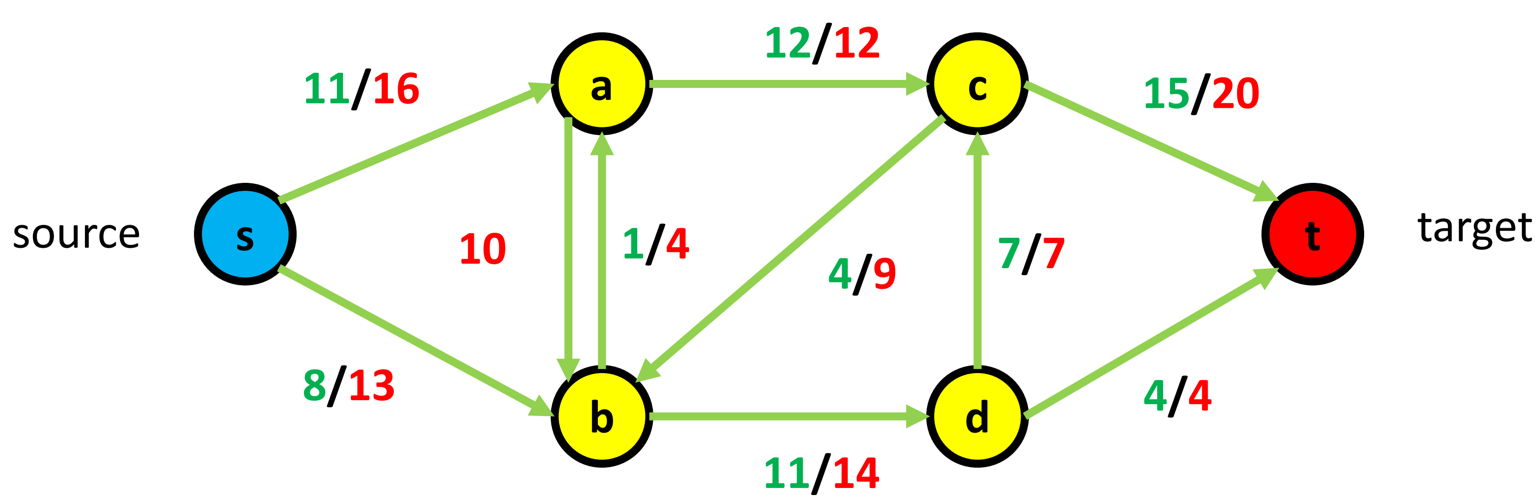


- What is the flow of the network? 19
  - Total flow out of source vertex
  - Total flow into target vertex



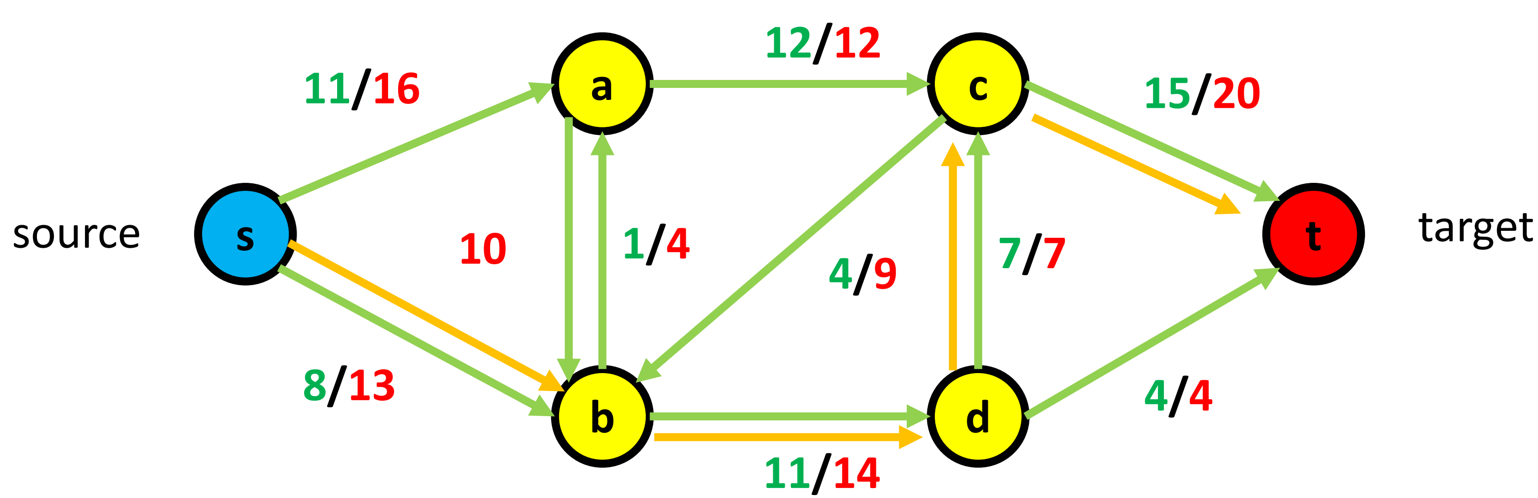
MONASH University

- What is the flow of the network? 19
  - Total flow out of source vertex
  - Total flow into target vertex
- Is this the maximum possible flow for this network?



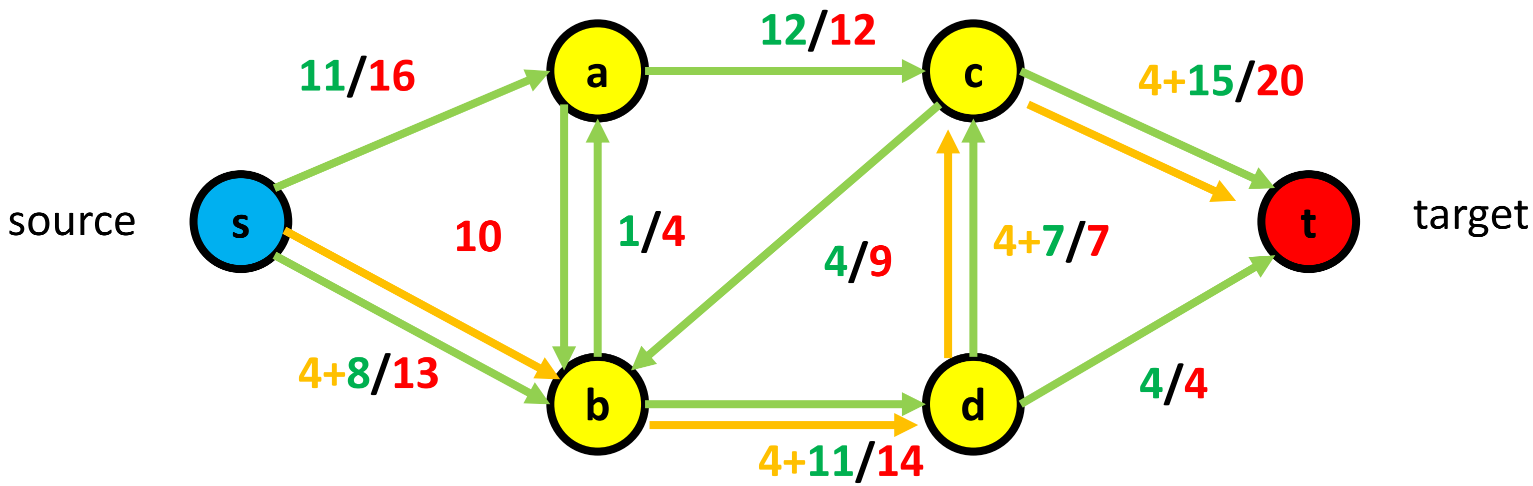
Maximum-Flow Problem
Best network?

- Is this the maximum possible flow for this network?
  – We can push in 4 more through the following route…



MONASH University

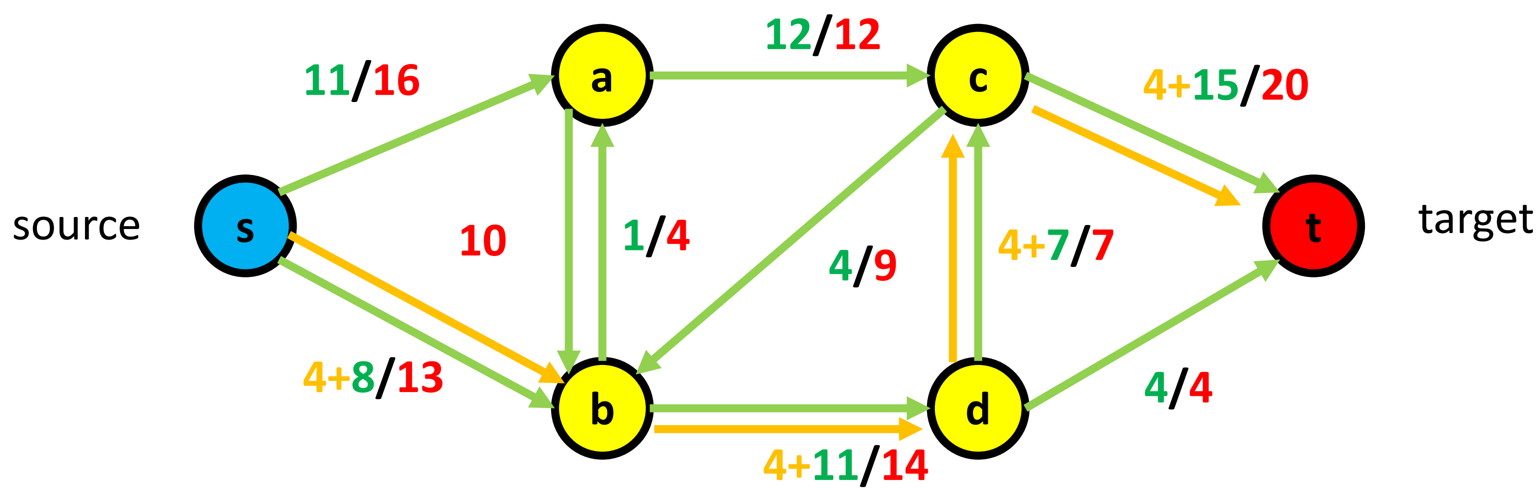
- Is this the maximum possible flow for this network?
  - We can push in 4 more through the following route…



Maximum-Flow Problem
Best network?

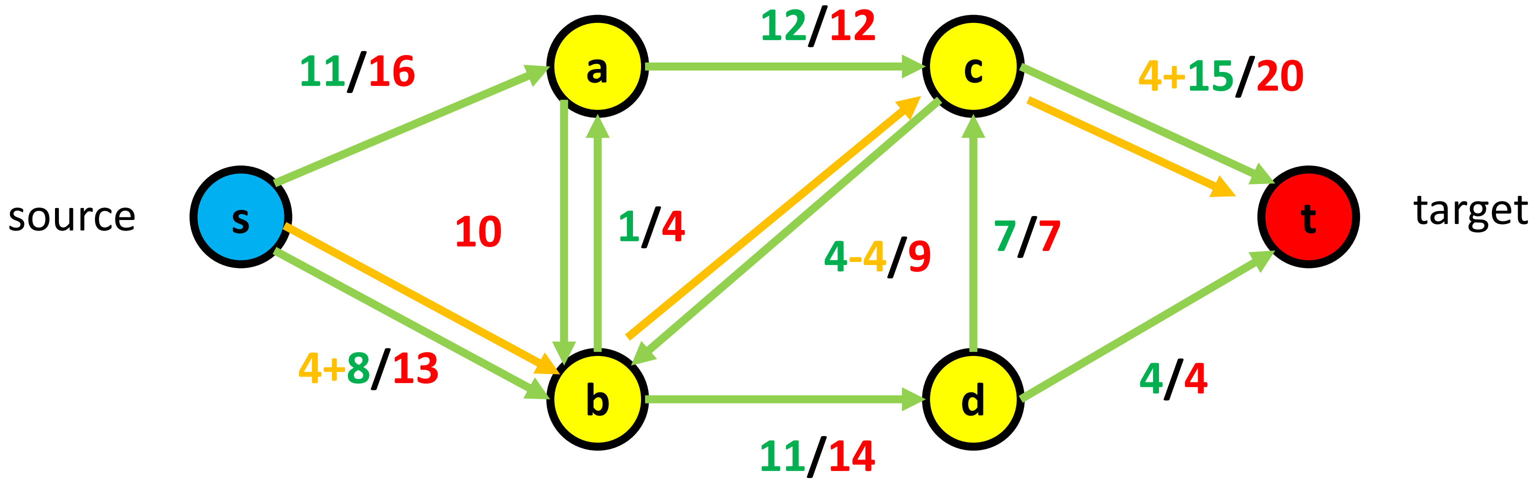
MONASH University

- Is this the maximum possible flow for this network?
  - We can push in 4 more through the following route… we cant! Cause over capacity…

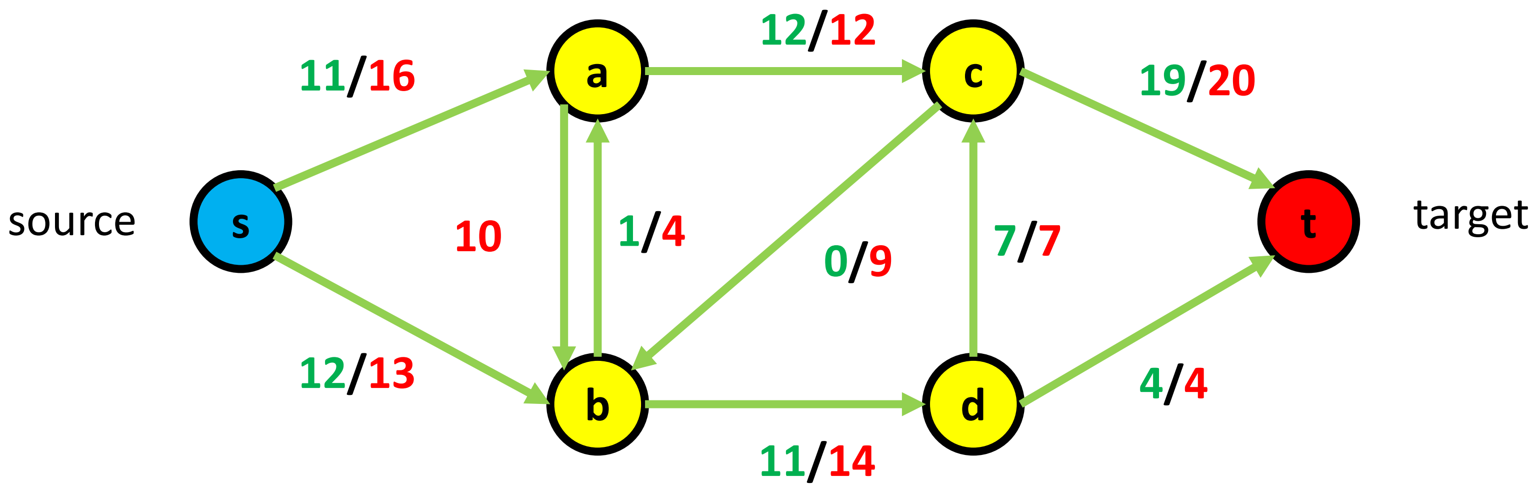


15 > 14 , capacity constraint principle

- Is this the maximum possible flow for this network?
  - We can push in 4 more through the following route… but we can do this, not accepting the opposite…



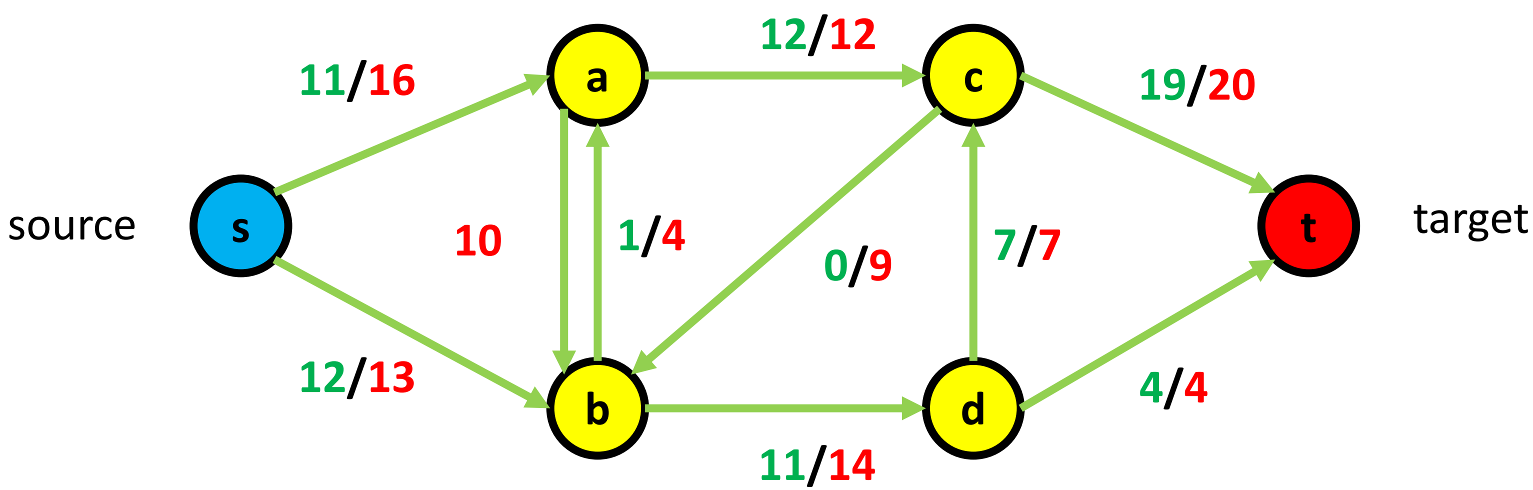
- Is this the maximum possible flow for this network? <u>23!</u>
  - We can push in 4 more through the following route… but we can do this, not accepting the opposite…



Maximum-Flow Problem

Best network?

- Is this the maximum possible flow for this network? 23!
  - We can push in 4 more through the following route… but we can do this, not accepting the opposite…
- Is this easy to do?
  - No of course, but we in CS to make it easy!



Questions?

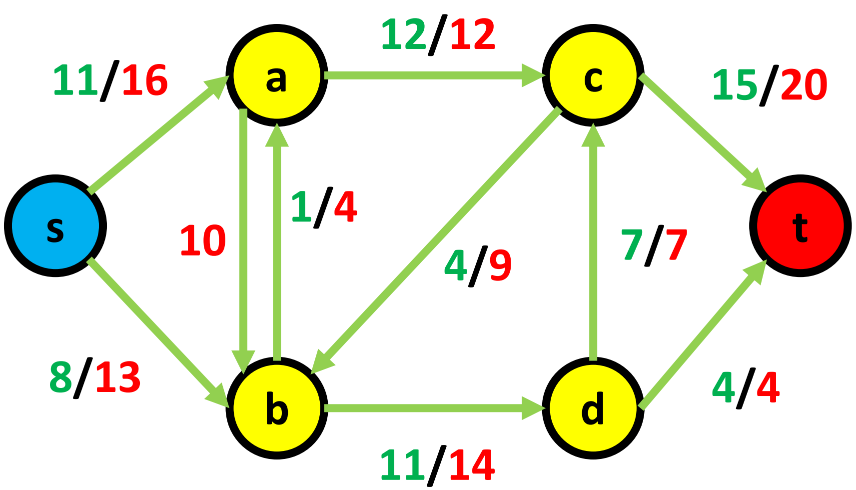
- What we use to find the maximum flow

- What we use to find the maximum flow

- Residue network first…

Residual network
Another freaking network

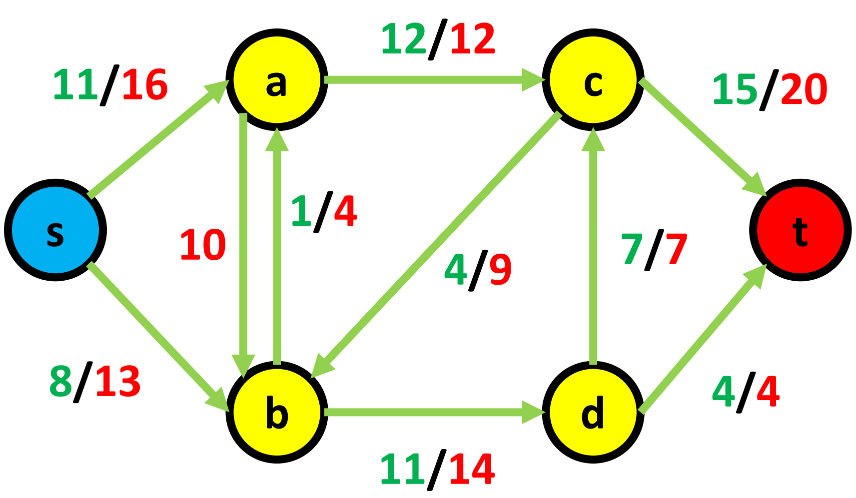
- Consider the following graph (same as earlier)



Residual network

Another freaking network

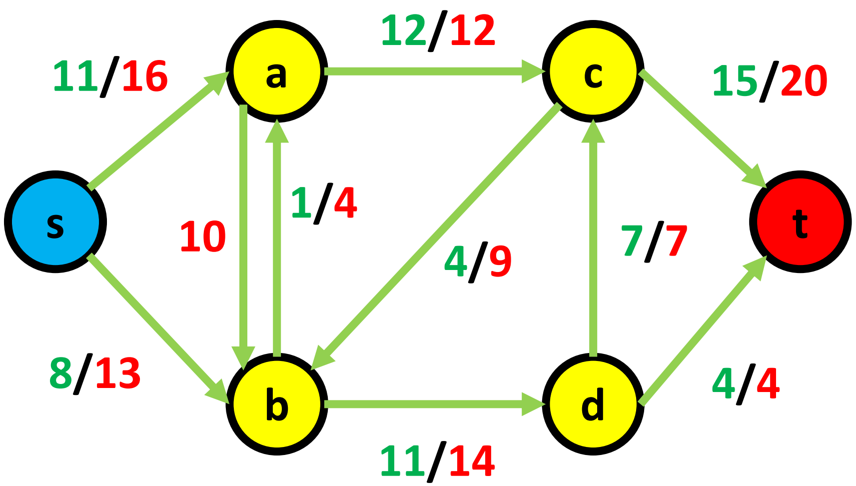
- Consider the following graph (same as earlier)
  - Can you make a residual network?



Residual network

Another freaking network

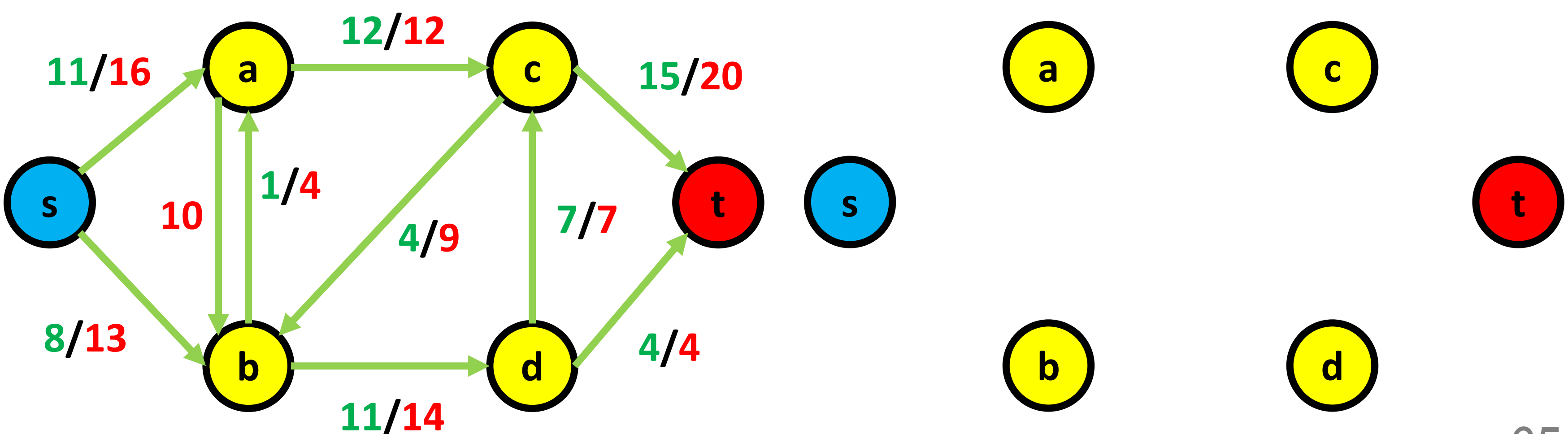
- What is a residual network?



Residual network

Another freaking network

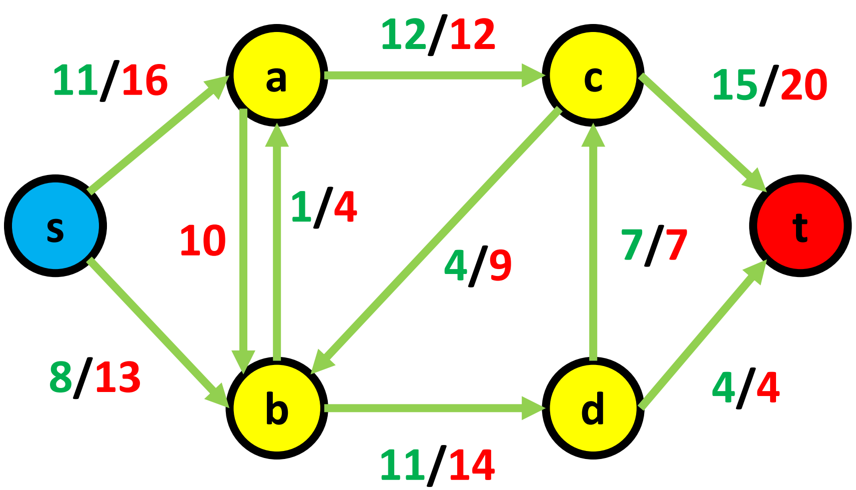
- What is a residual network?
  - Same vertices

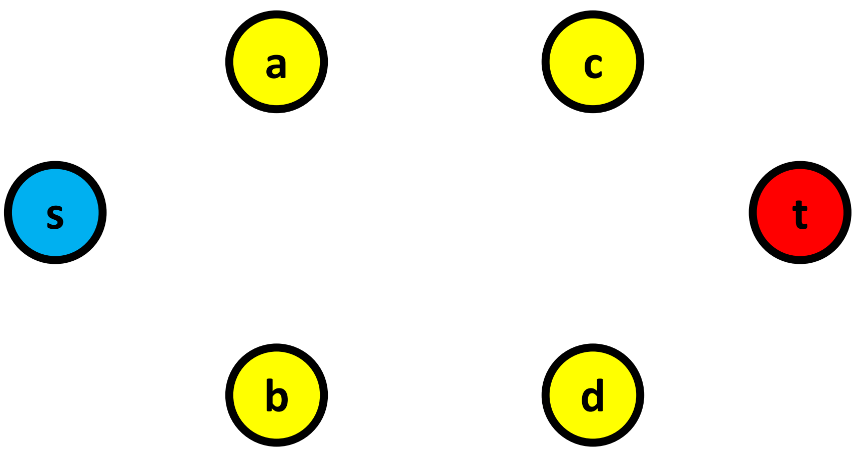


Residual network
Another freaking network

MONASH University

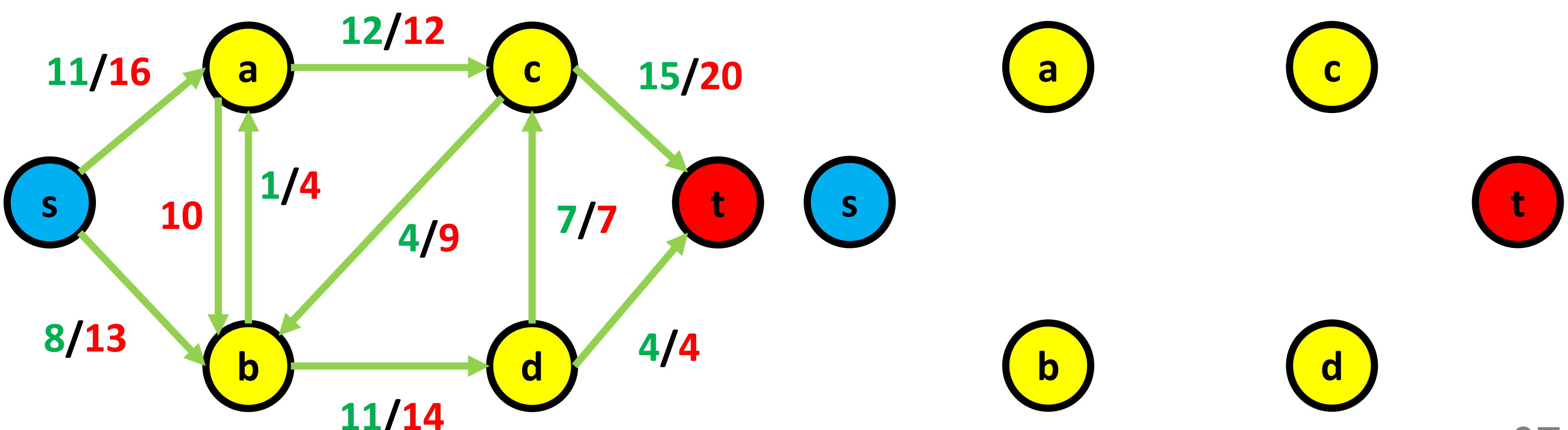
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge





MONASH University

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity

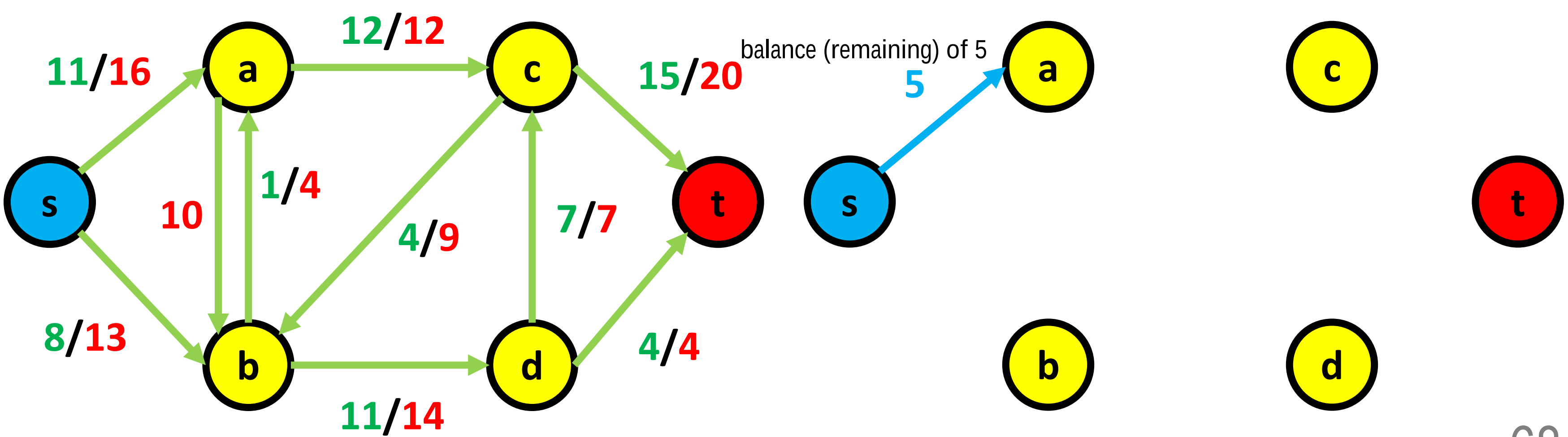


Residual network

Another freaking network

MONASH University

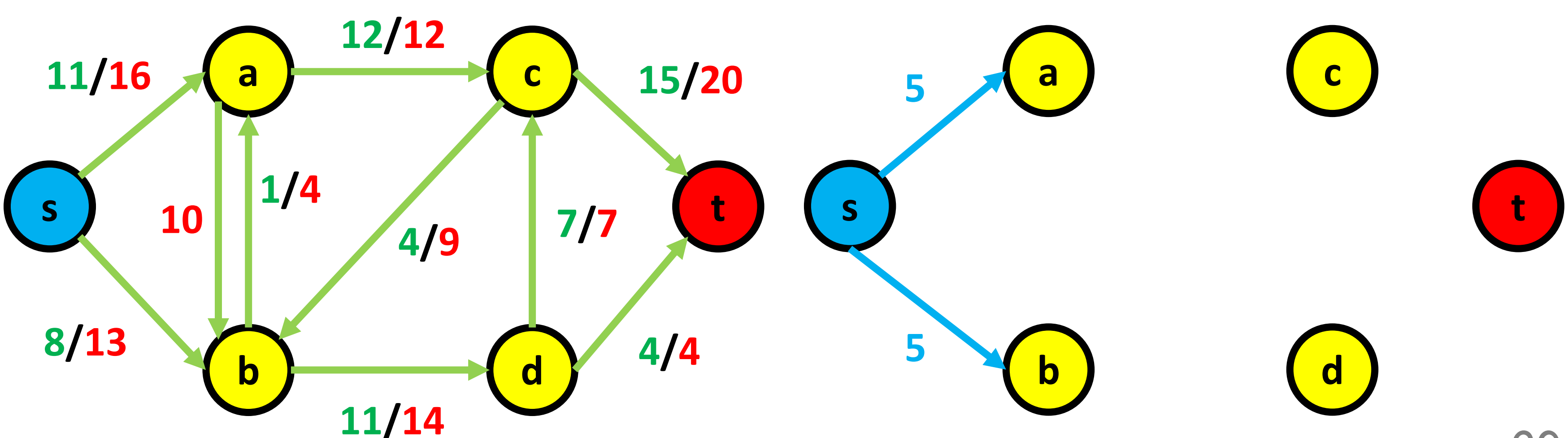
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity



Residual network

Another freaking network

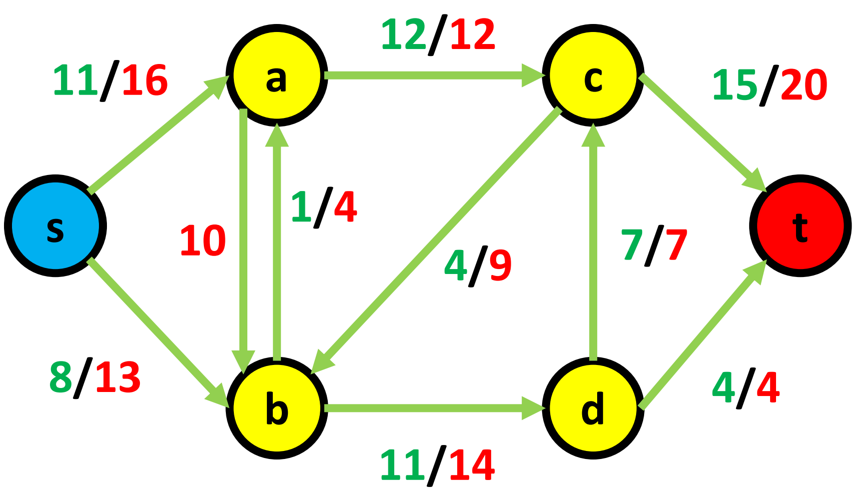
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity

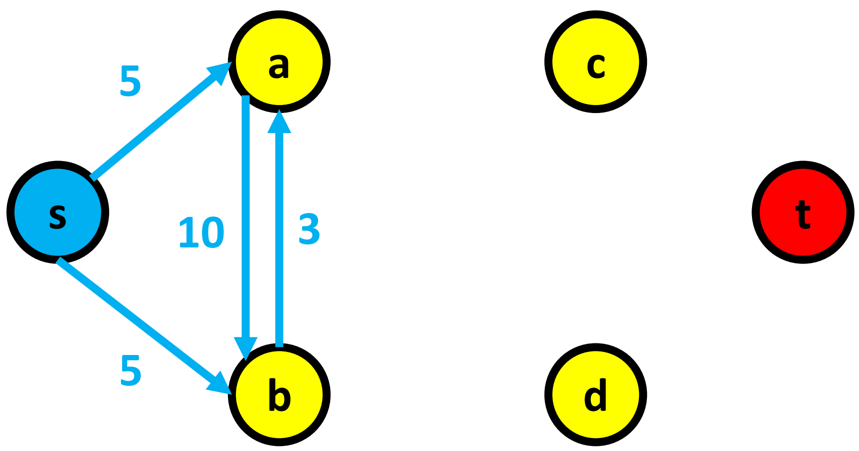


Residual network

Another freaking network

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
    - What about the one between a and b?

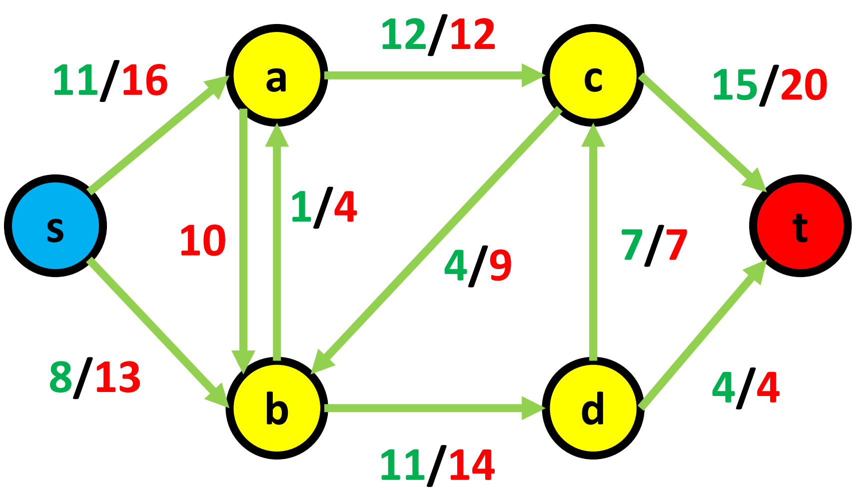


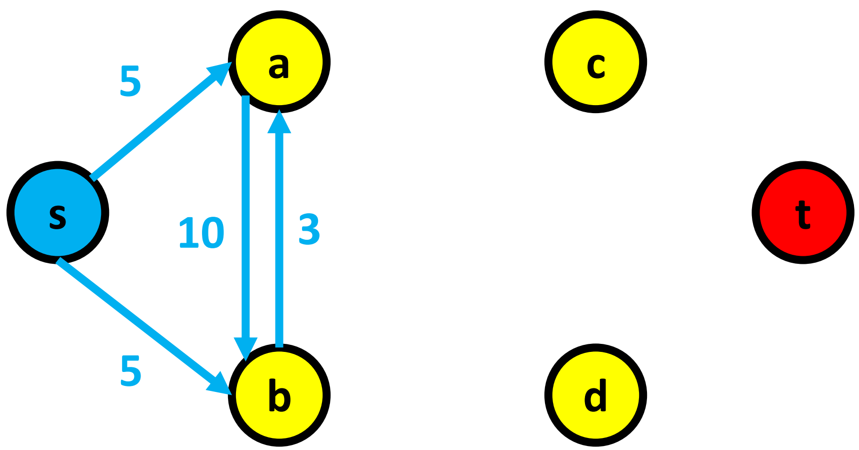


Residual network

Another freaking network

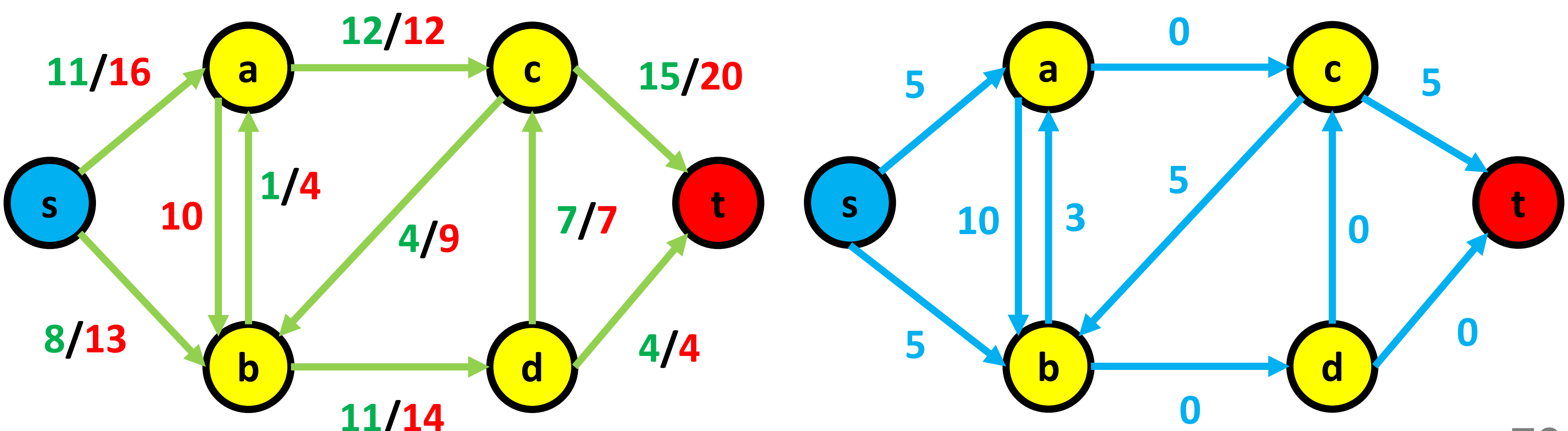
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
    - What about the one between a and b? We will come back to this later…





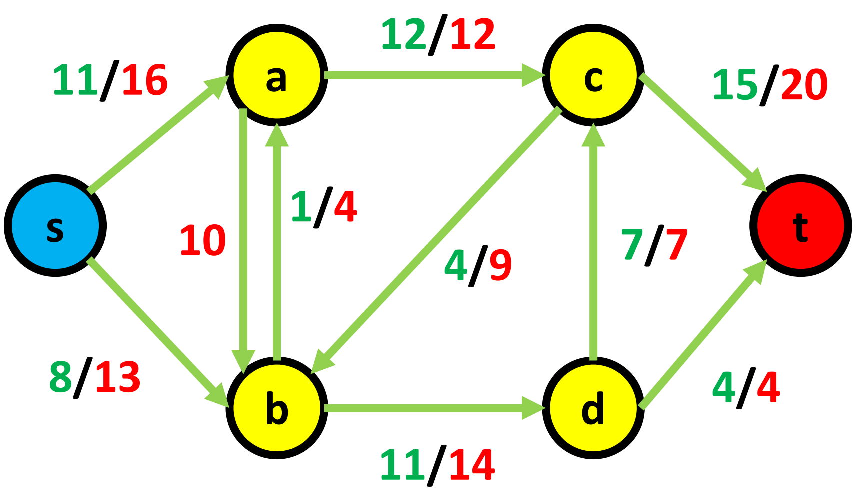
Residual network
Another freaking network

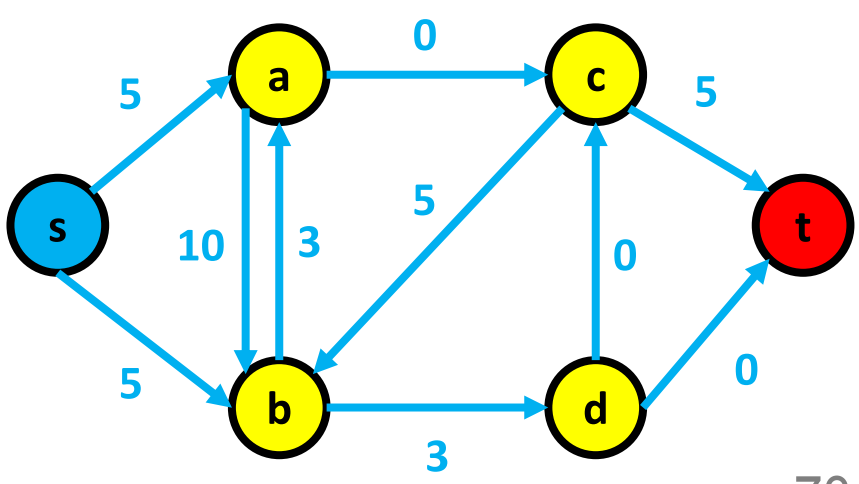
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity



Residual network
Another freaking network

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
    - We can delete the ones with 0

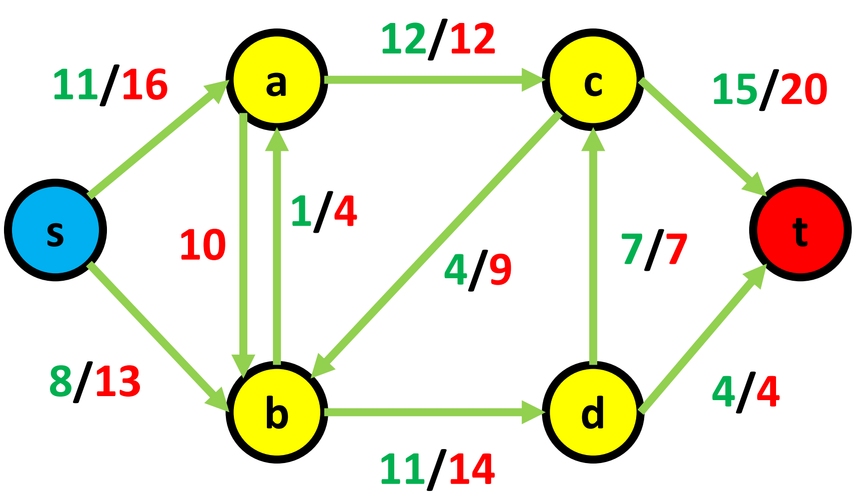


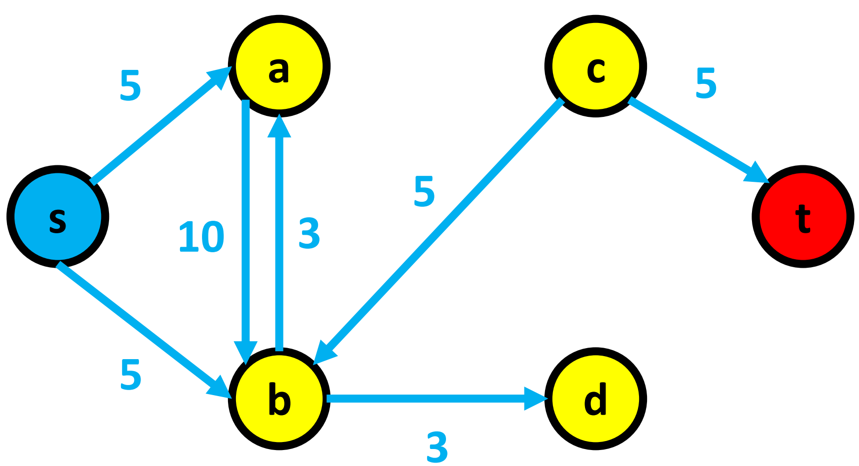


Residual network

Another freaking network

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
    - We can delete the ones with 0

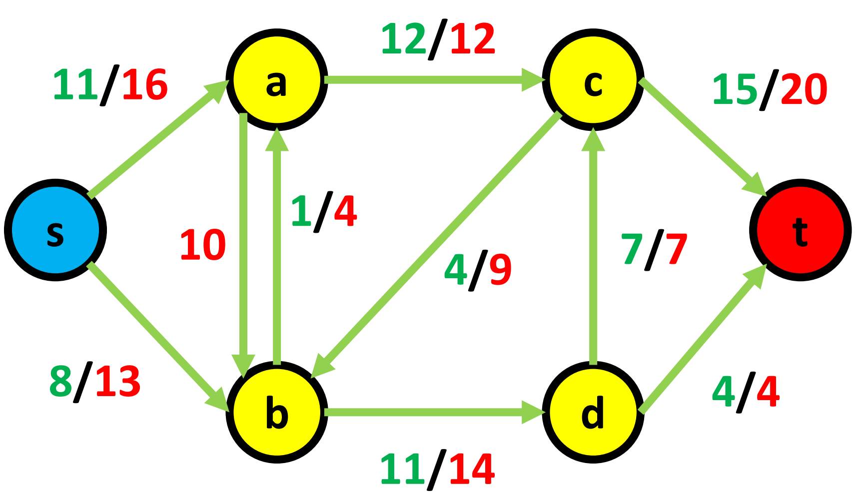


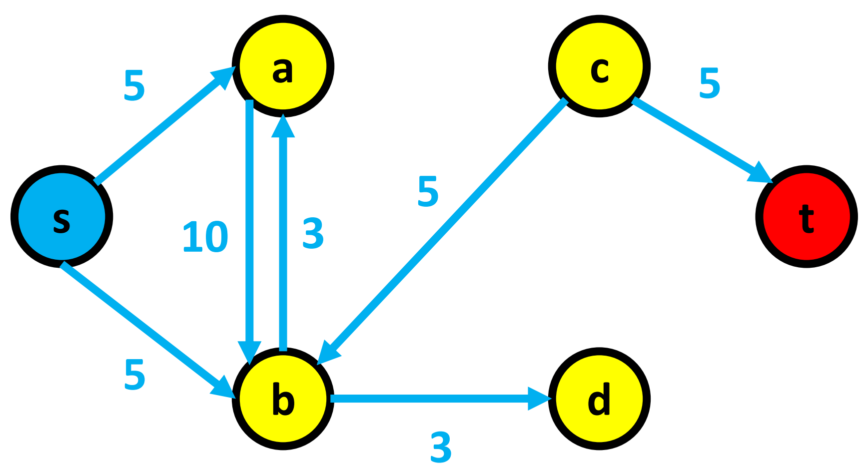


Residual network

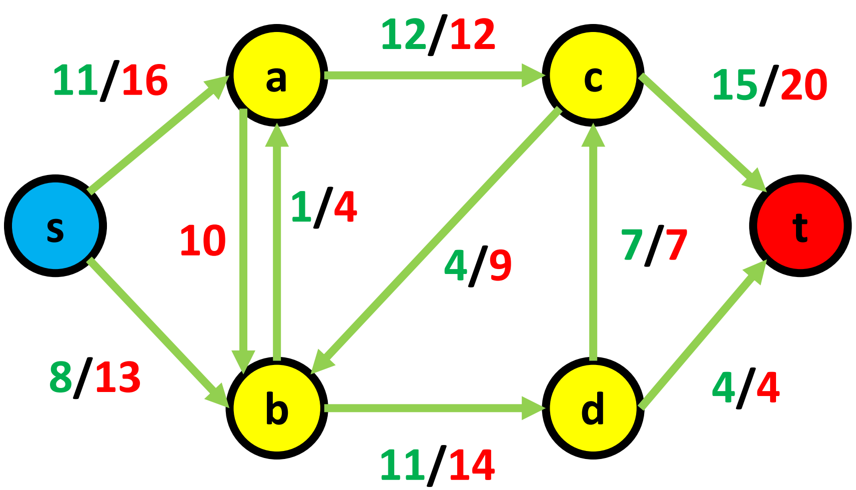
Another freaking network

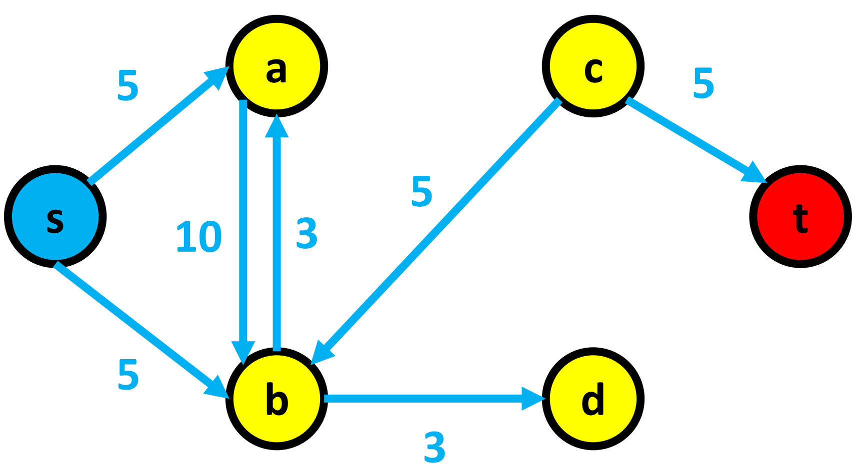
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge





- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - Provided they have been allocated

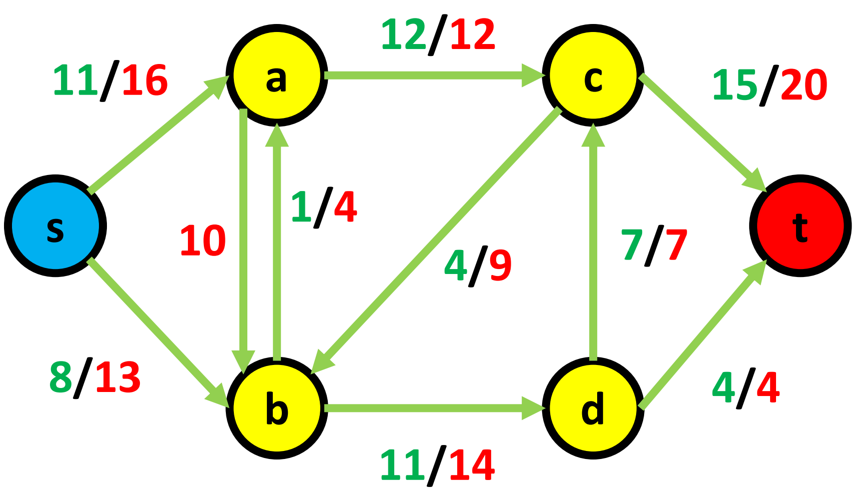


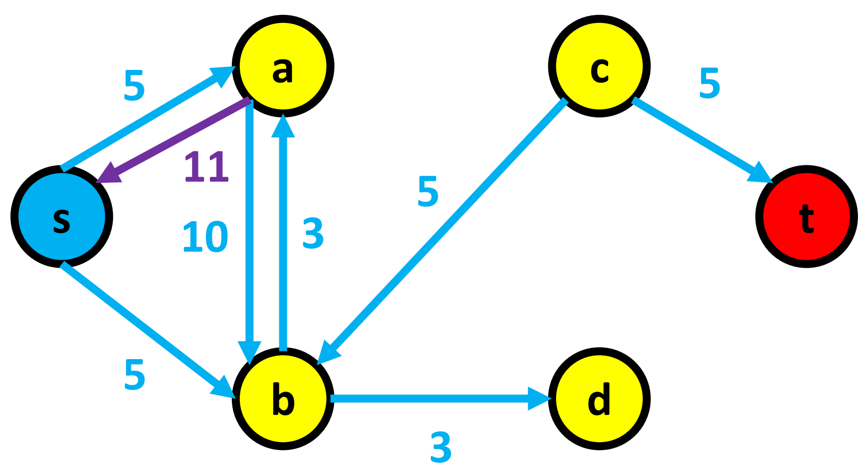


Residual network

Another freaking network

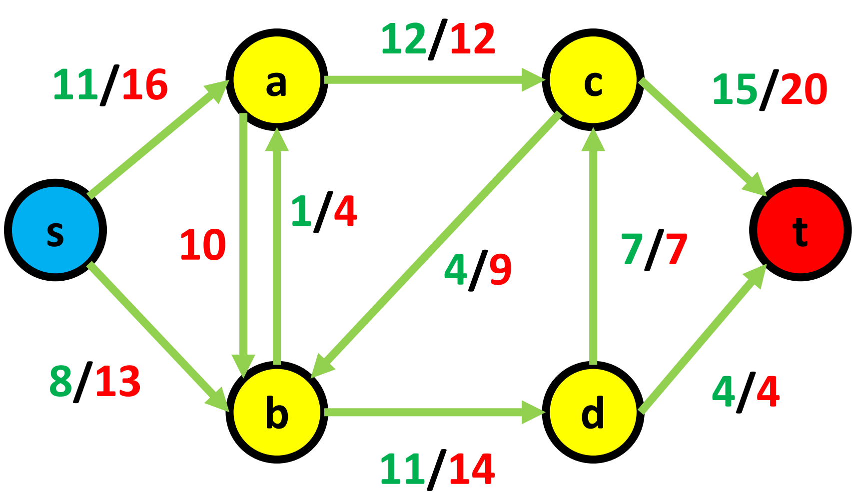
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - Provided they have been allocated

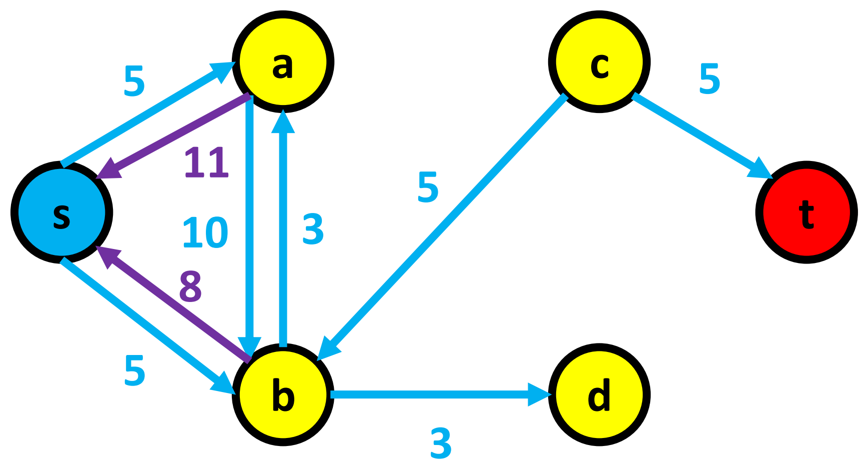




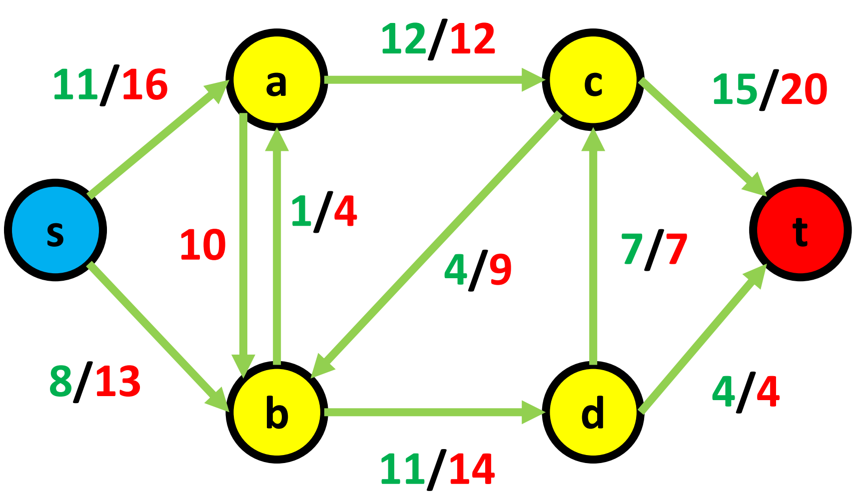
Residual network
Another freaking network

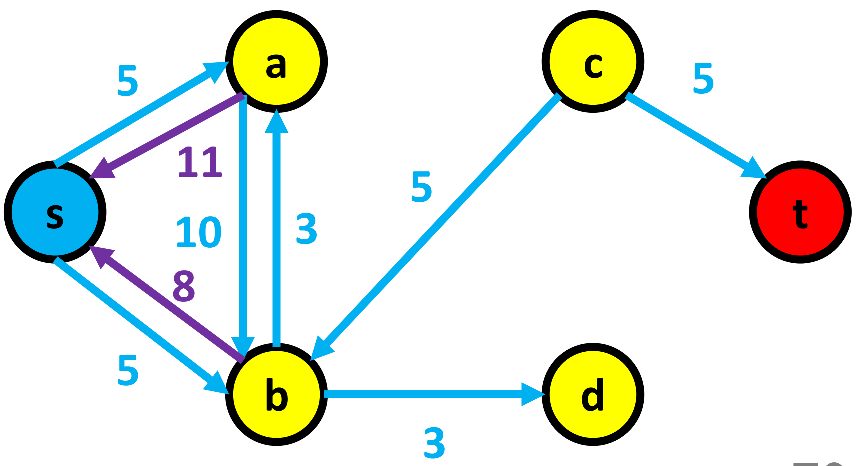
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - Provided they have been allocated





- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - What about the one between a and b?

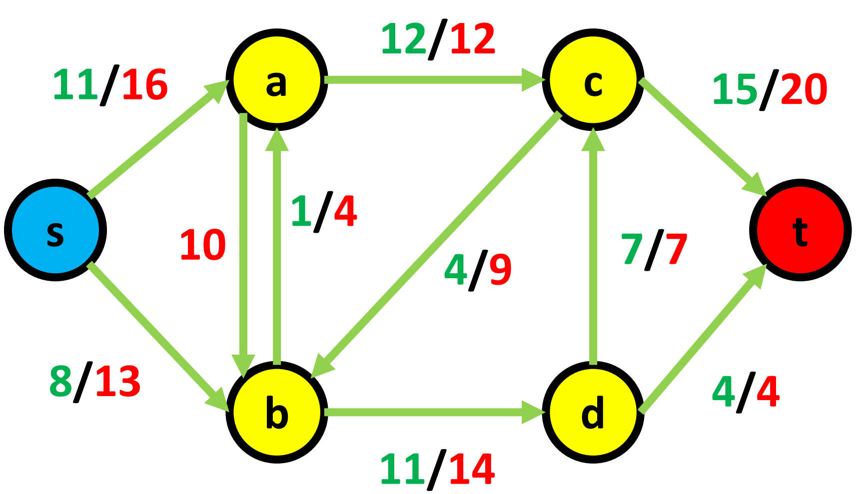


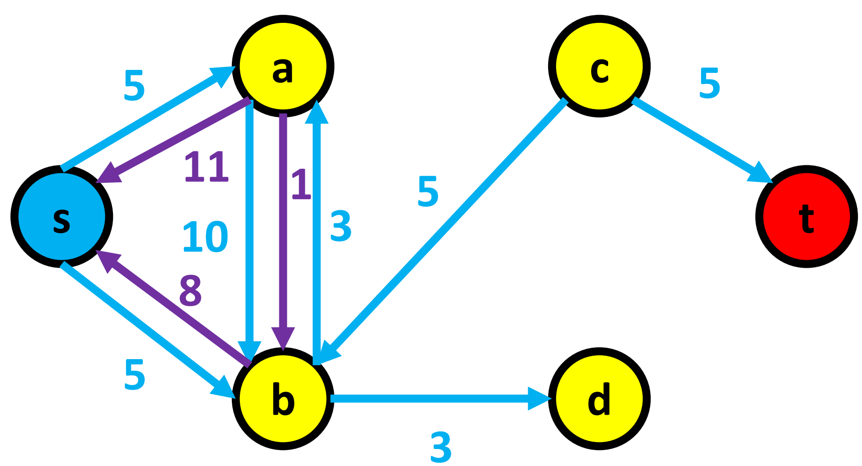


Residual network

Another freaking network

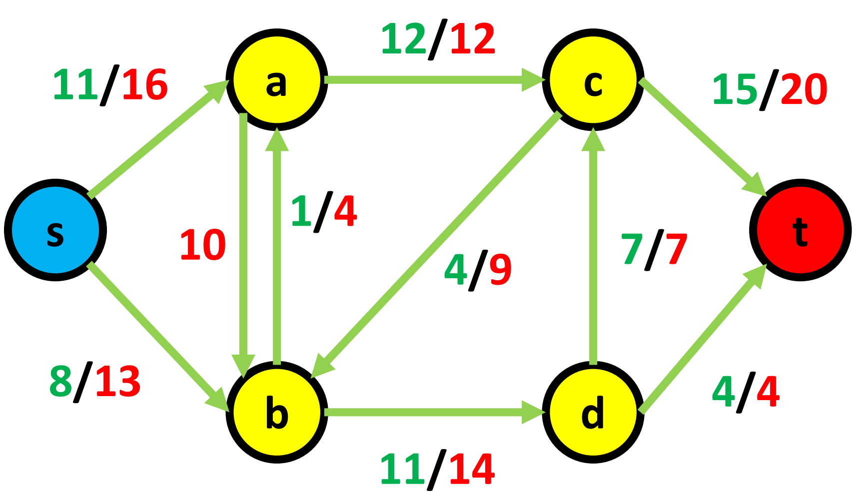
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - What about the one between a and b?

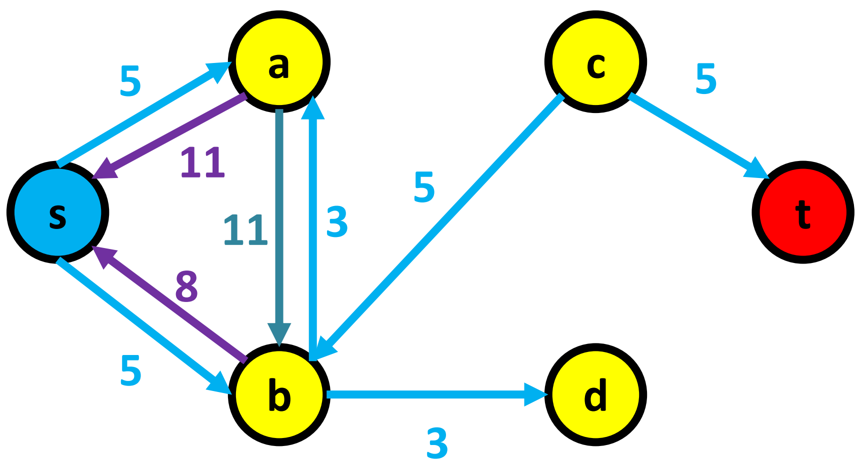




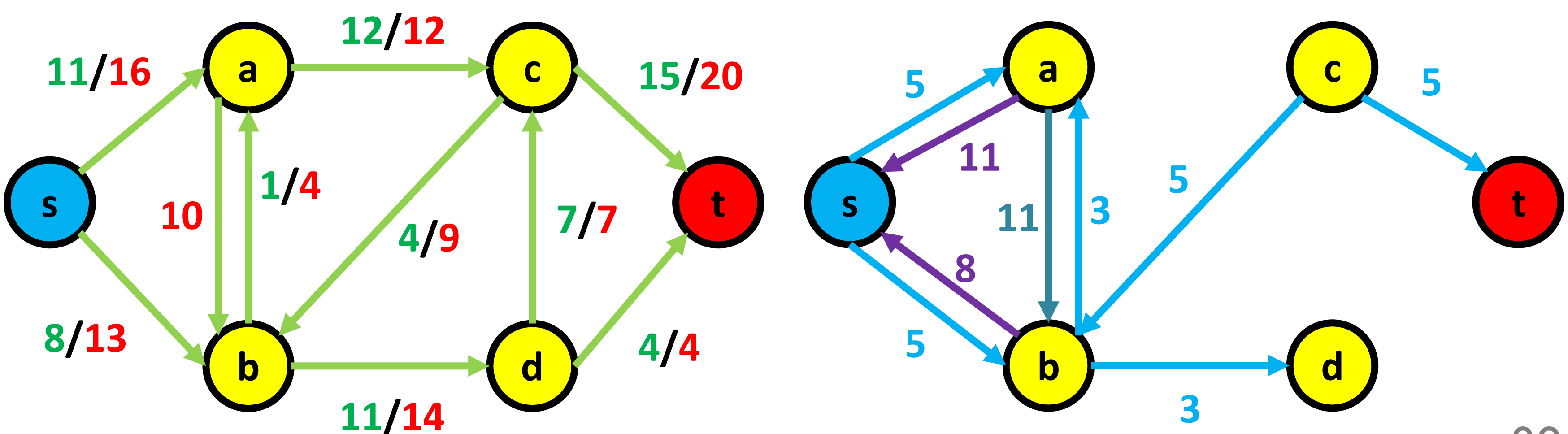
Residual network
Another freaking network

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - What about the one between a and b? We have 2 in the same direction, so we combine both





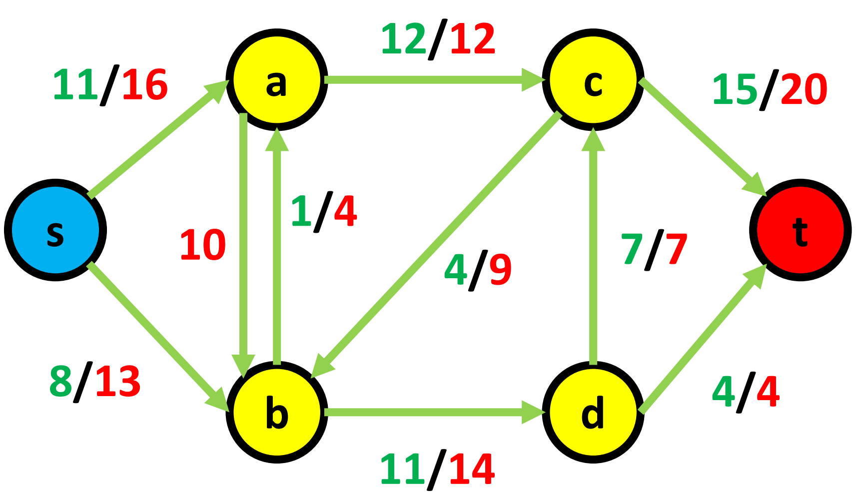
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - What about the one between a and b? We have 2 in the same direction, so we combine both. The other side is 0, so nothing to combine

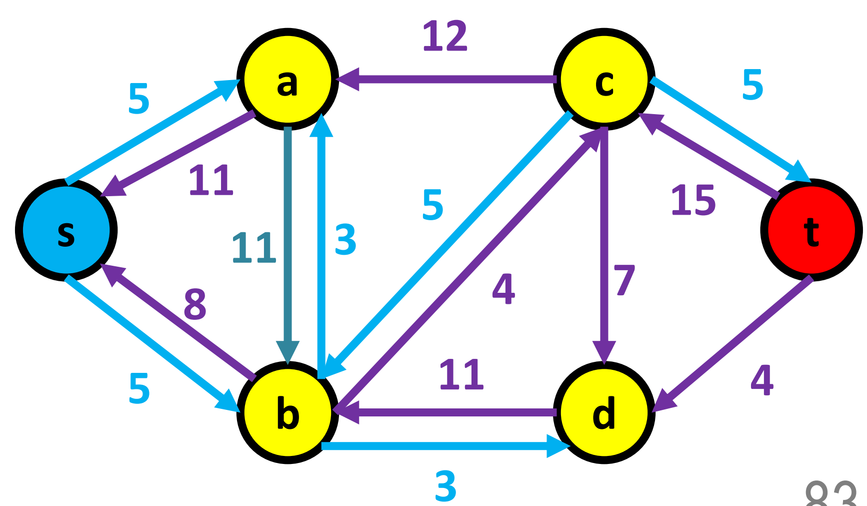


Residual network

Another freaking network

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
    - And we add for all

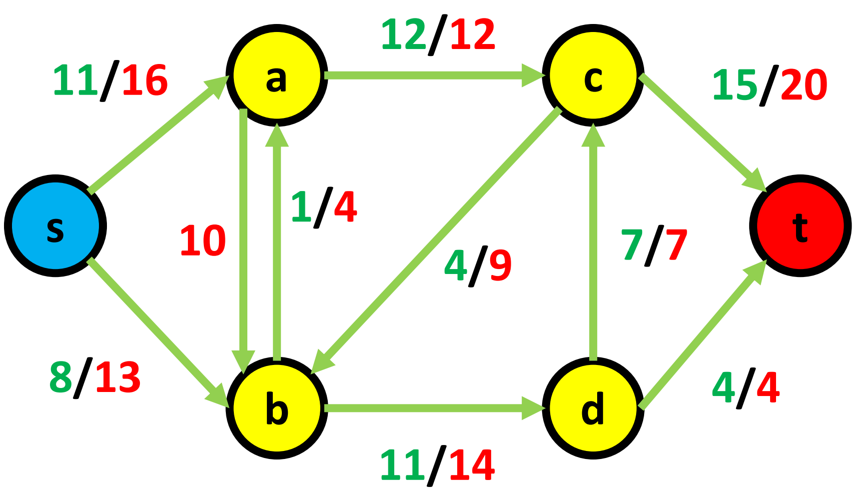


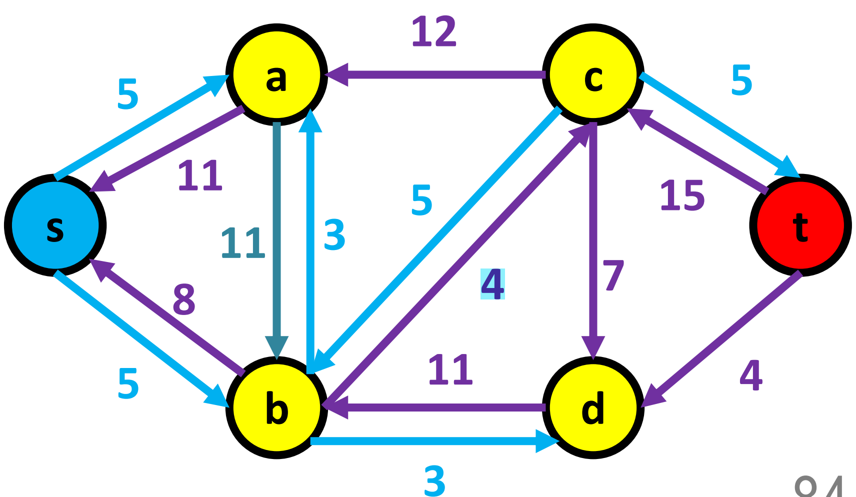


Residual network

Another freaking network

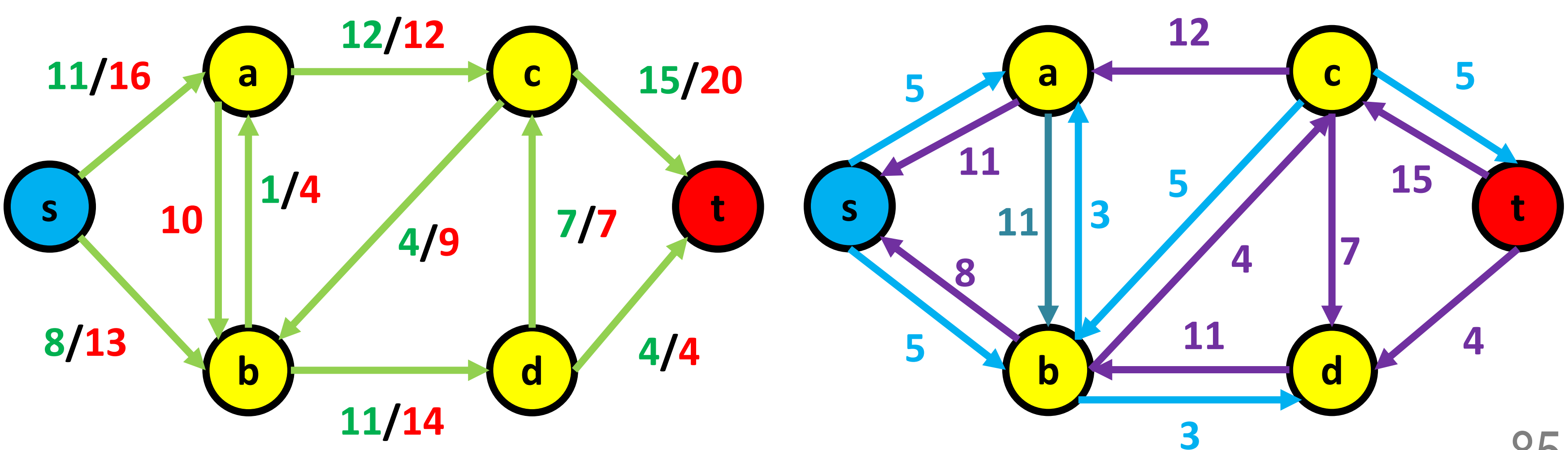
- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled



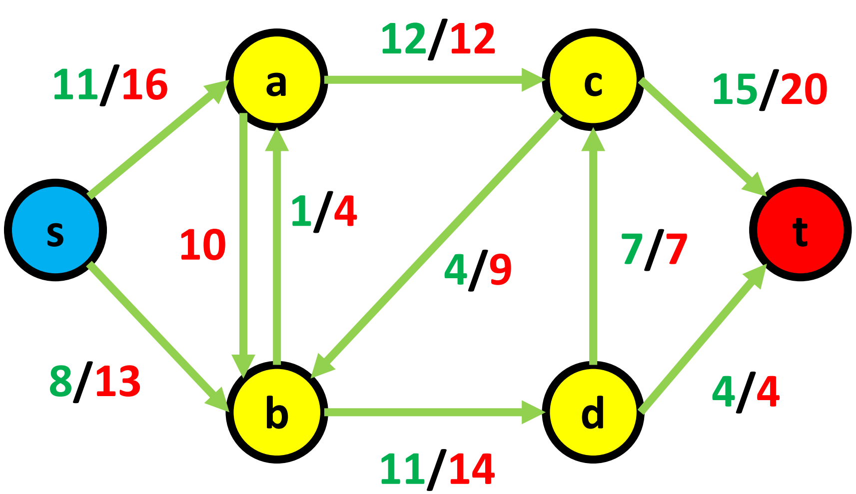


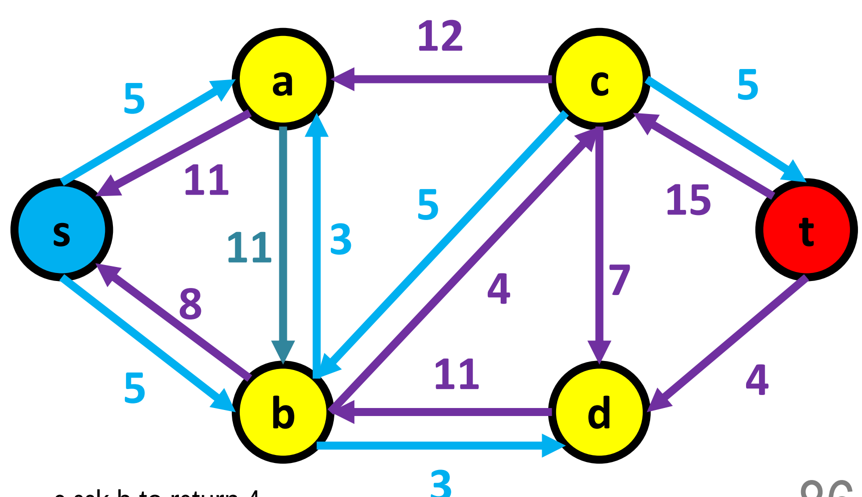
Residual network
Another freaking network

- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
  - Simple graph, so multi edges are merged together



- What is a residual network?
  - Same vertices
  - Forward edge/ residual edge for remaining capacity
  - Backward edge/ reversible edge for flow that can be cancelled
  - Simple graph, so multi edges are merged together
  - Also note, sum of the edge between 2 vertices same as the edge capacity



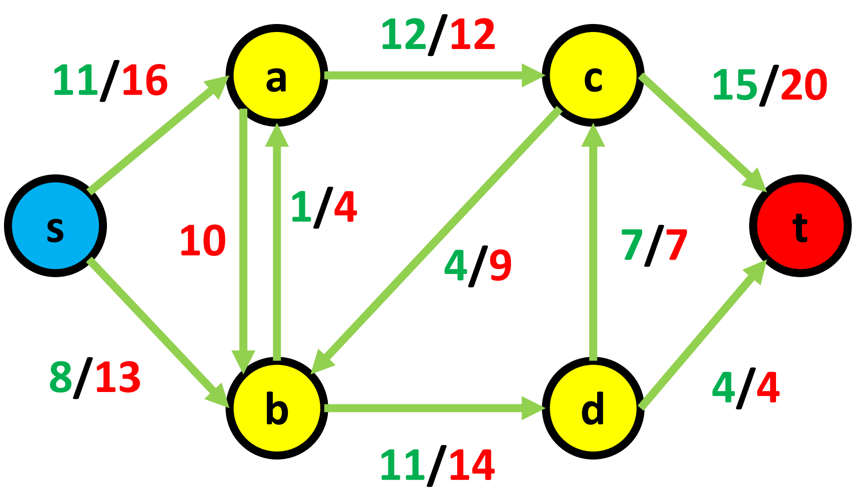


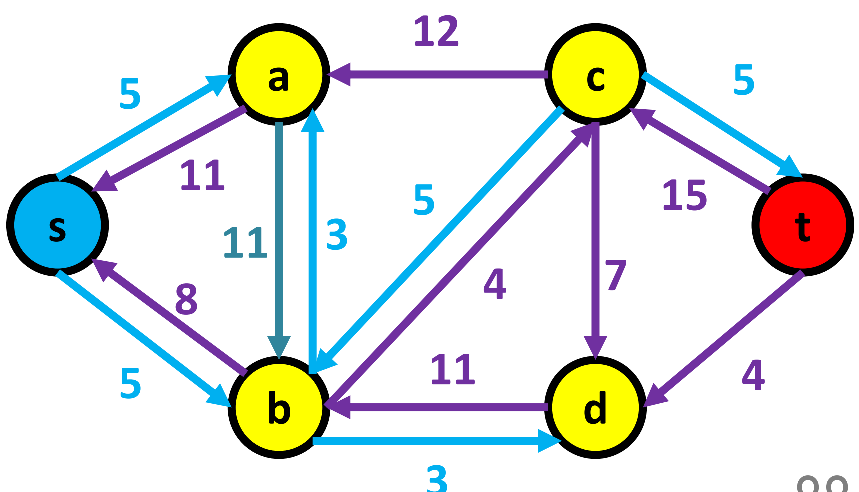
c ask b to return 4
so it can pass 4 flow then flow from c to t

Questions?

Residual network

So what is the purpose?

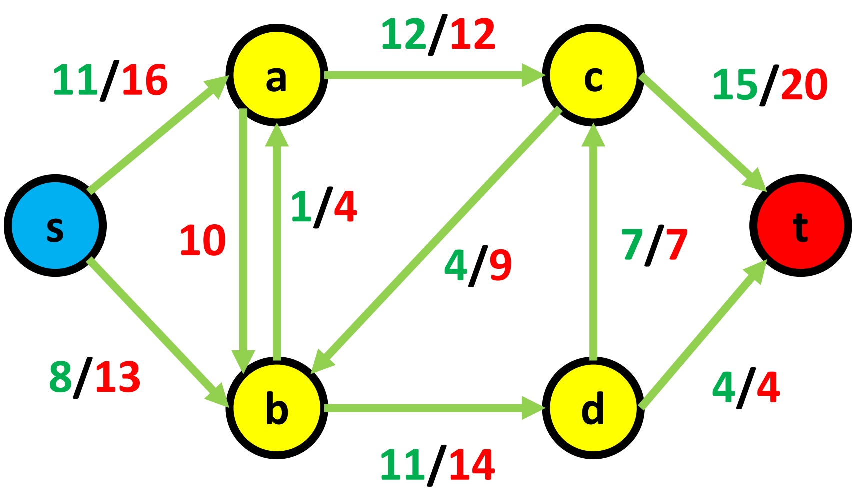


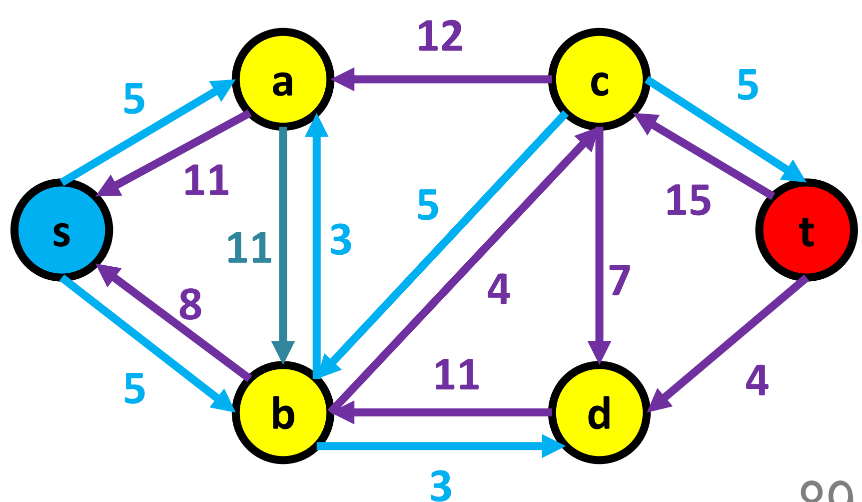


Residual network

Measuring the potential of a network

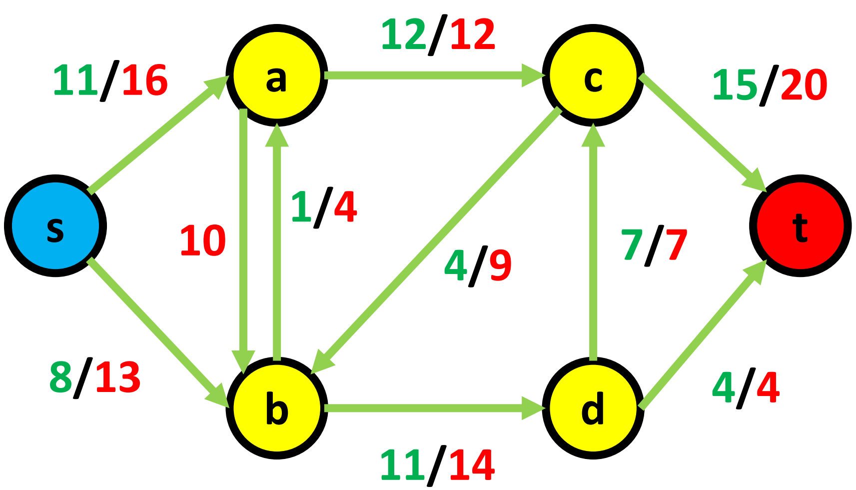
- Stores the network potential

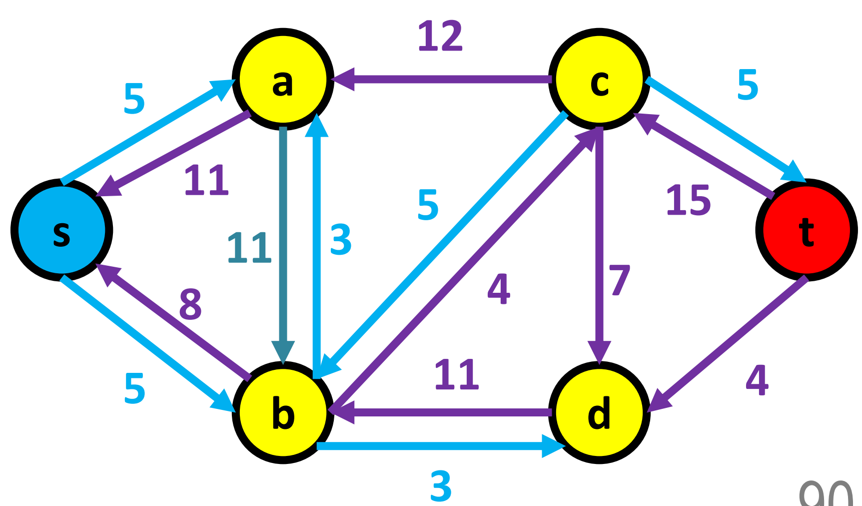




Residual network
Measuring the potential of a network

- Stores the network potential
  - Which we will unleash…

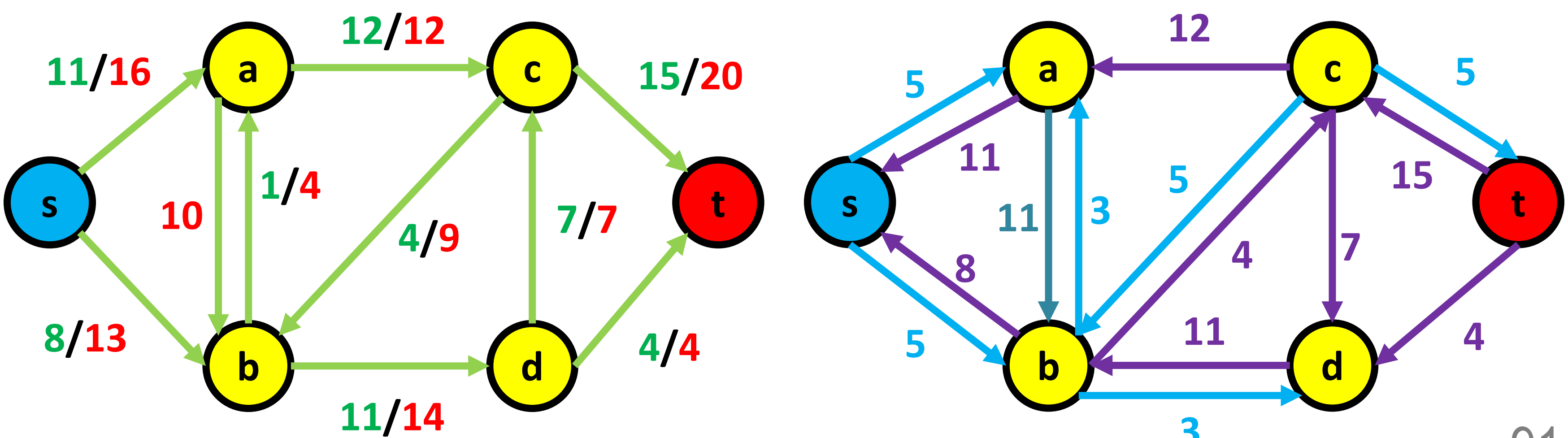




Residual network

Measuring the potential of a network

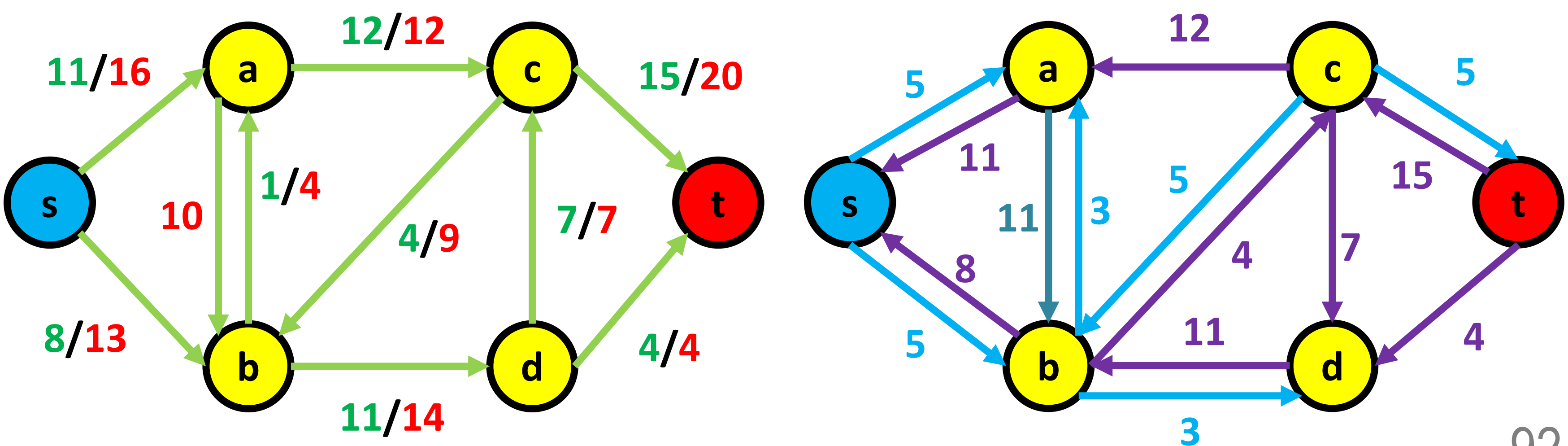
- Stores the network potential
  - Which we will unleash… via path augmentation!



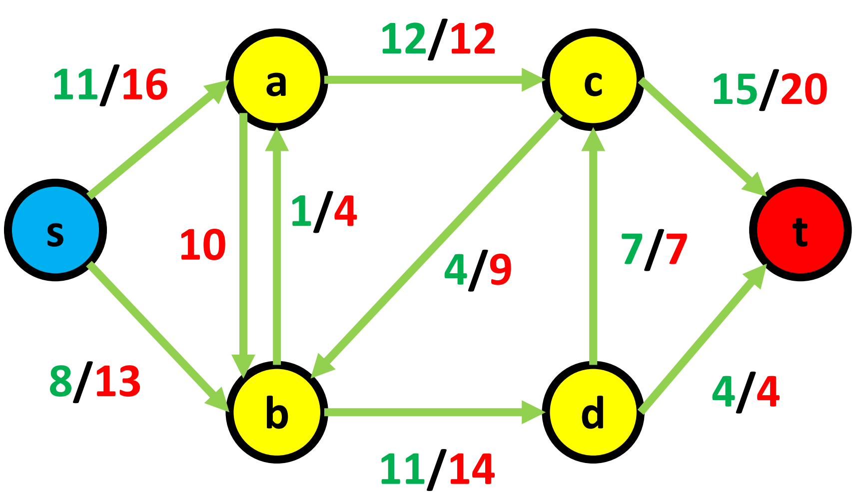
Residual network
Measuring the potential of a network

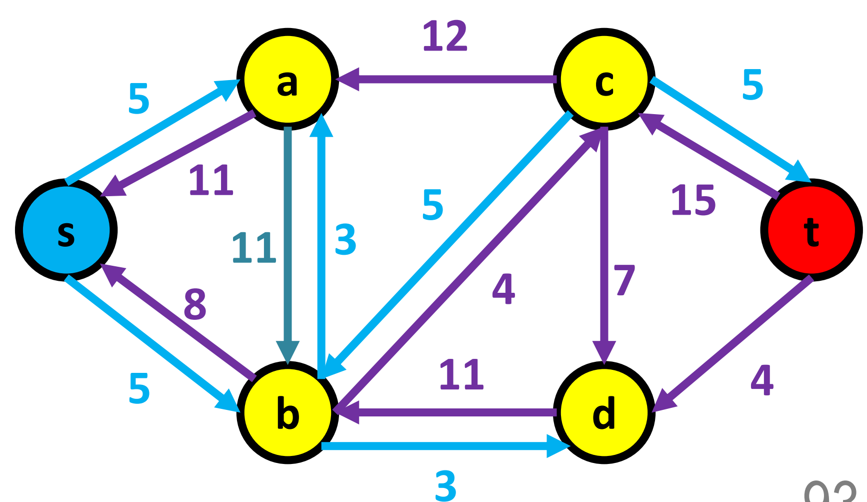
increase the path flow

- So what is path augmentation?

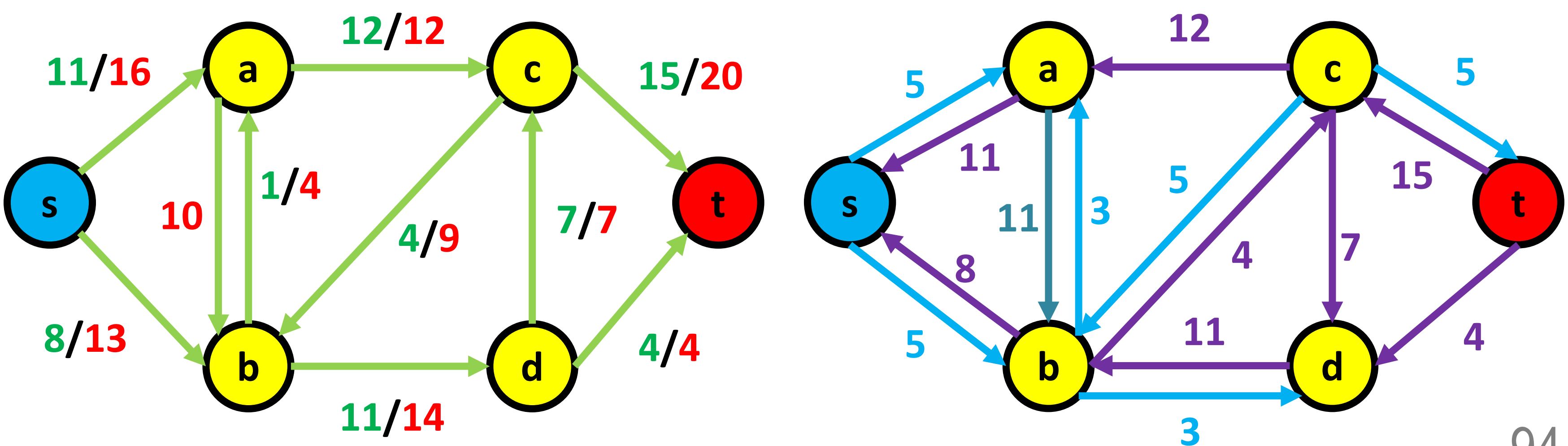


- So what is path augmentation?
  - A traversal in the residual network

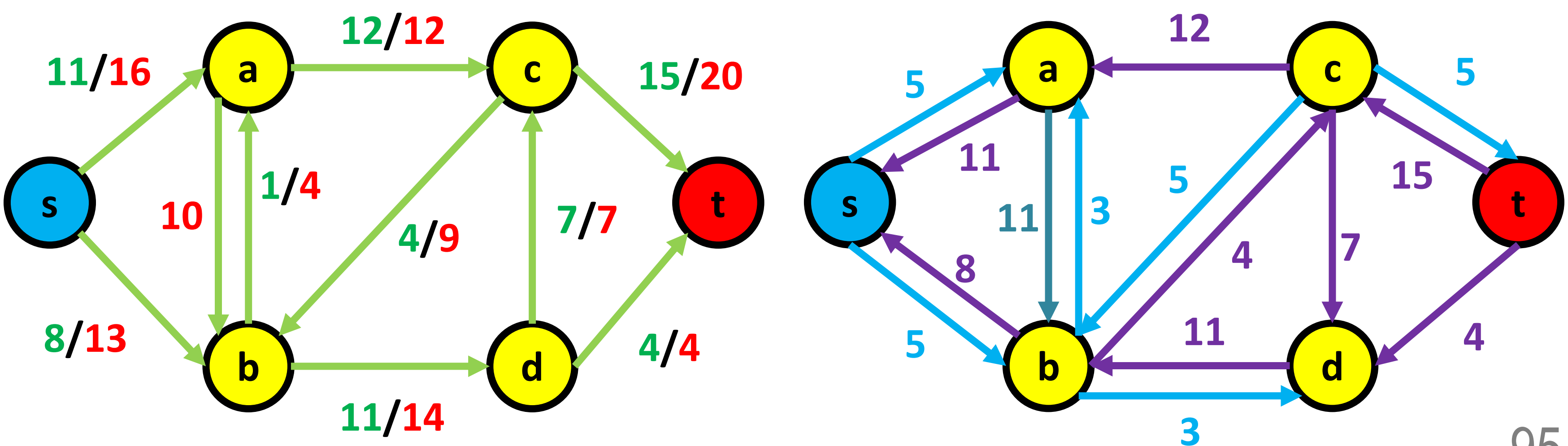




- So what is path augmentation?
  - A traversal in the residual network
  - From source, to target

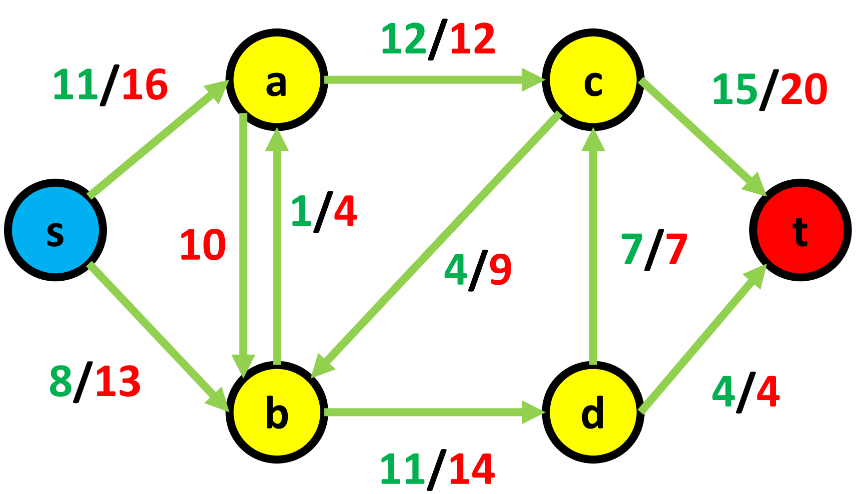


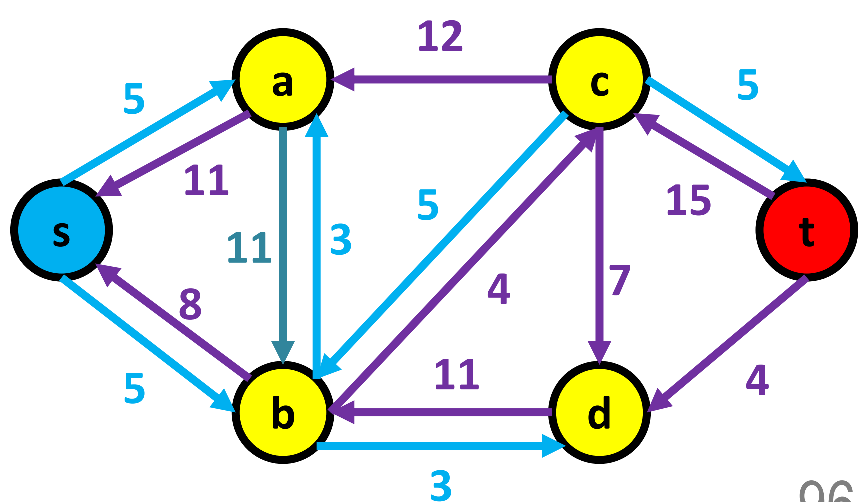
- So what is path augmentation?
  - A traversal in the residual network
  - From source, to target
  - Following the edges in the residual network



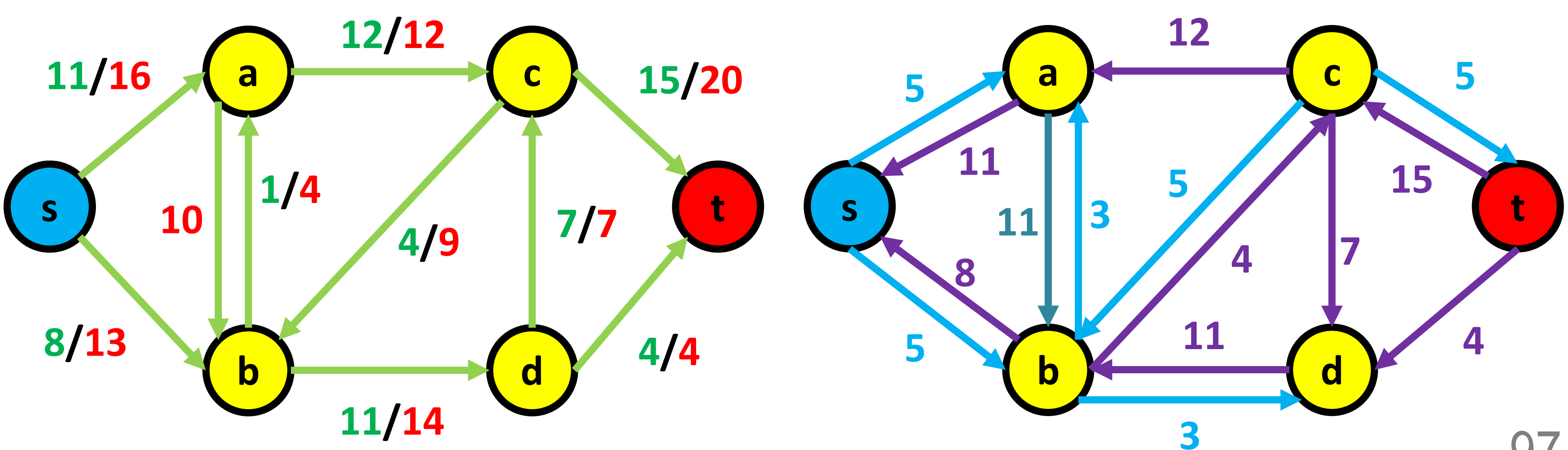
Path Augmentation
Traversal in residual network

- So what is path augmentation?
  - A traversal in the residual network
    - BFS! Or DFS!
  - From source, to target
  - Following the edges in the residual network





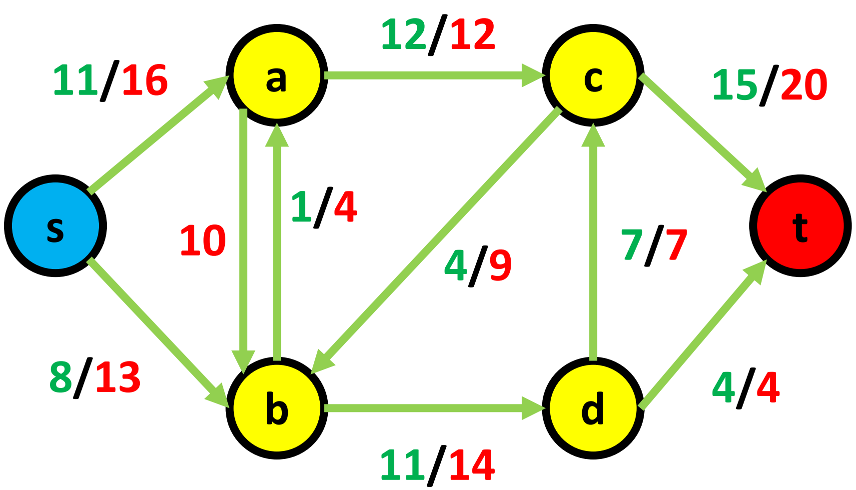
- Is there a path here?

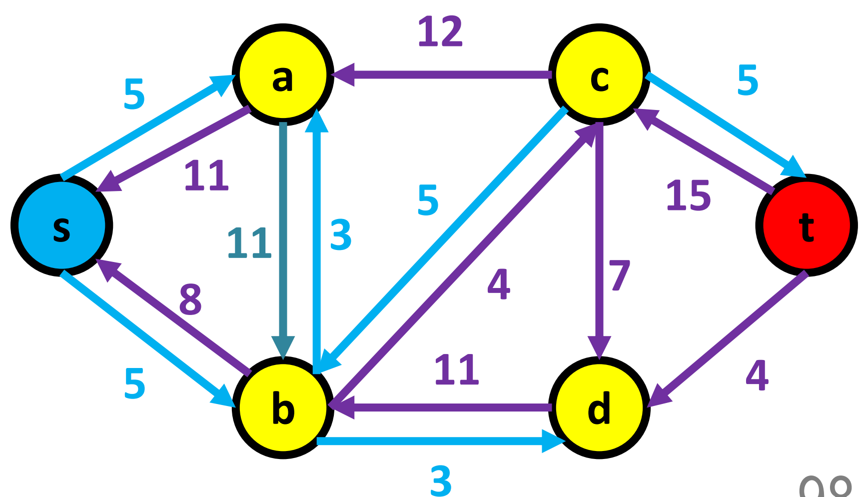


Path Augmentation

Traversal in residual network

- Is there a path here?
  - s -> b -> c -> t



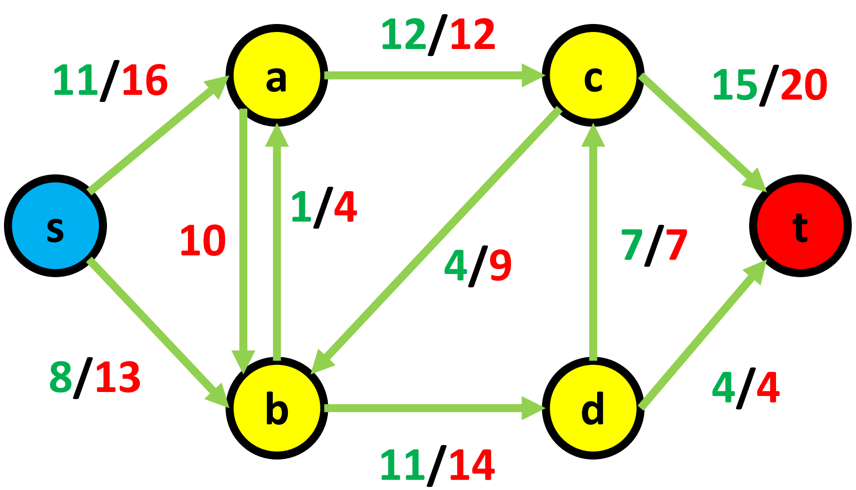


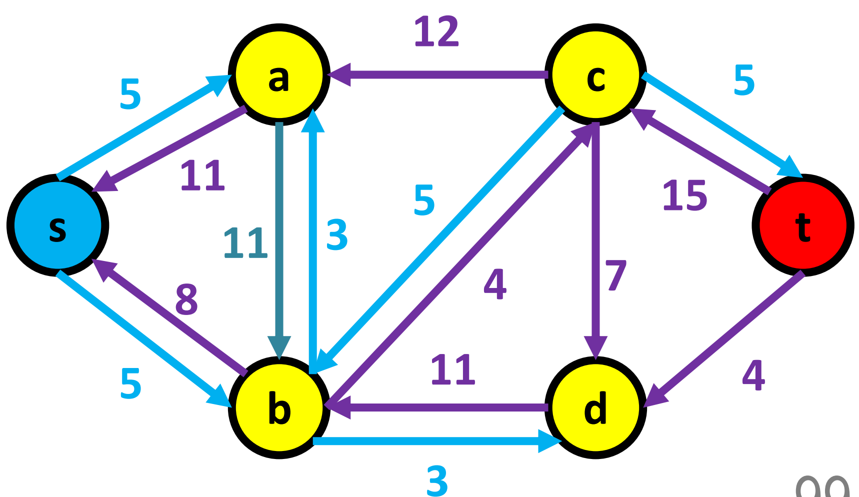
Path Augmentation
Traversal in residual network

- Is there a path here?
  - s -> b -> c -> t
  - s -> a -> b -> c -> t

either BFS or DFS

find a path from source to destination

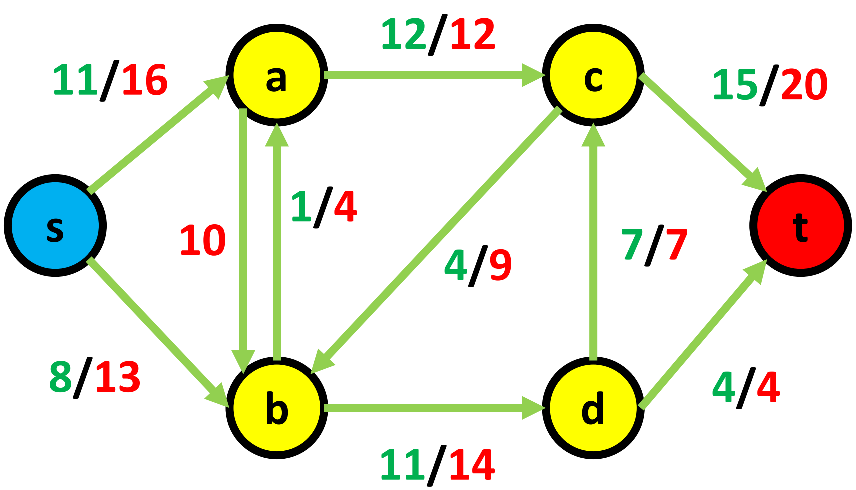


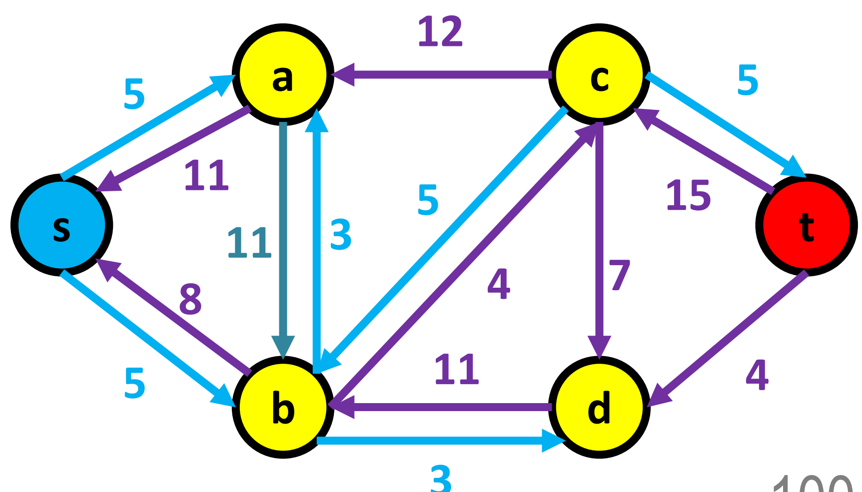


Path Augmentation

Traversal in residual network

- Is there a path here?
  - s -> b -> c -> t… let us look at this one first
  - s -> a -> b -> c -> t

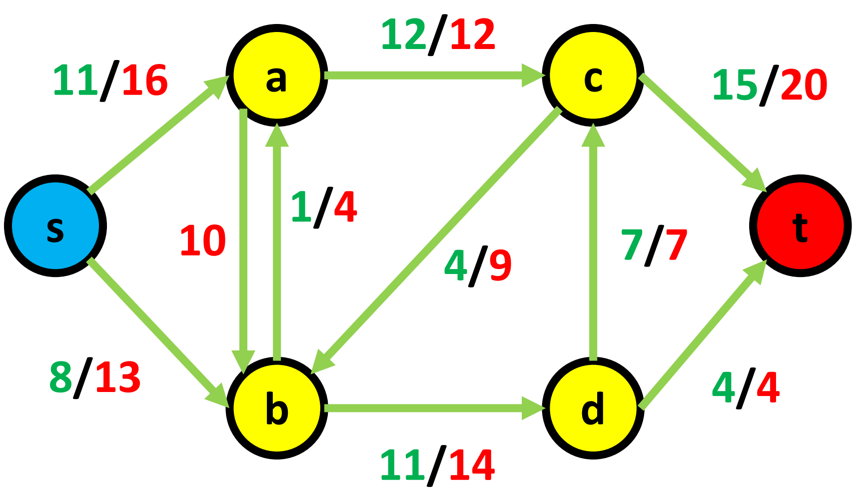


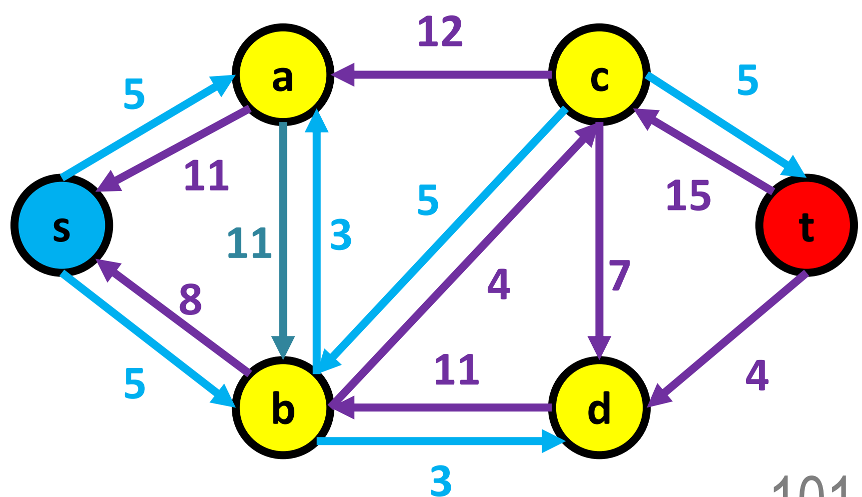


Path Augmentation

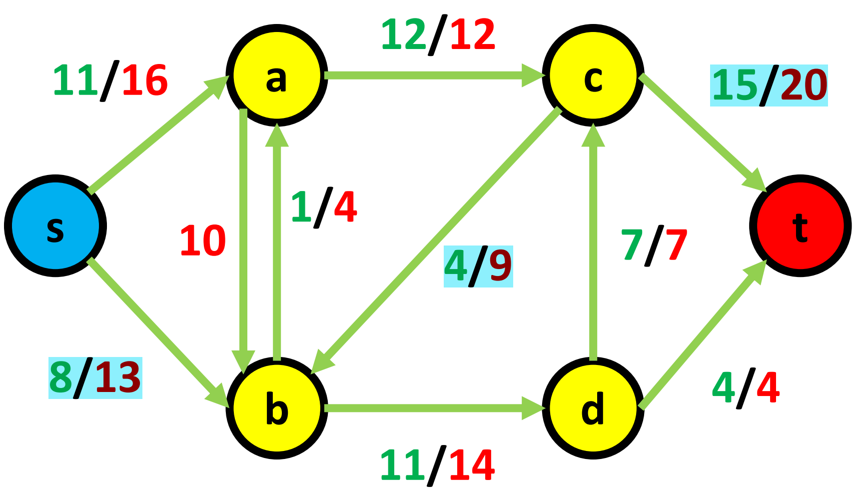
Traversal in residual network

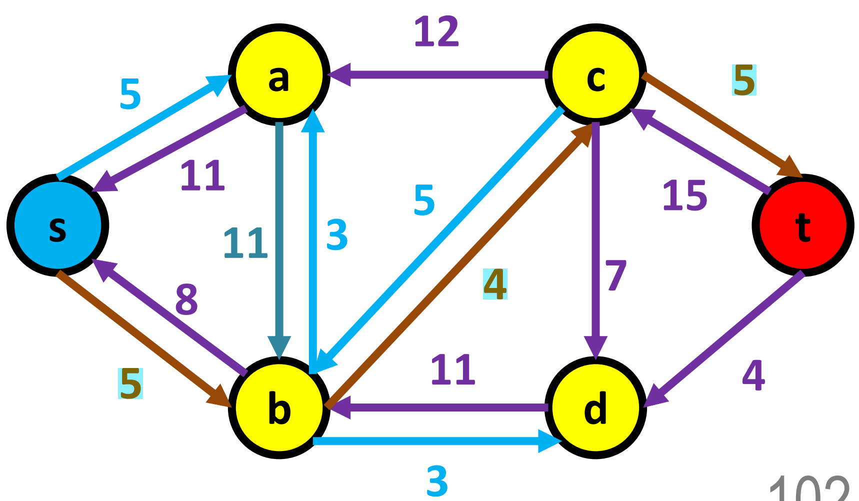
- Is there a path here?
  - s -> b -> c -> t… let us look at this one first



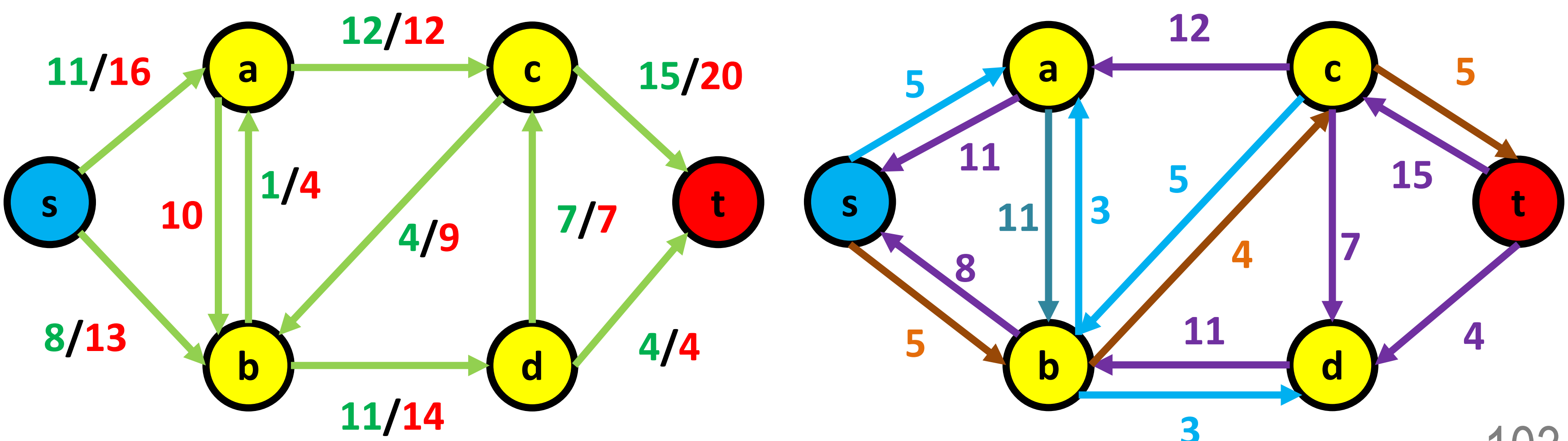


- Is there a path here?
  - s -> b -> c -> t… let us look at this one first



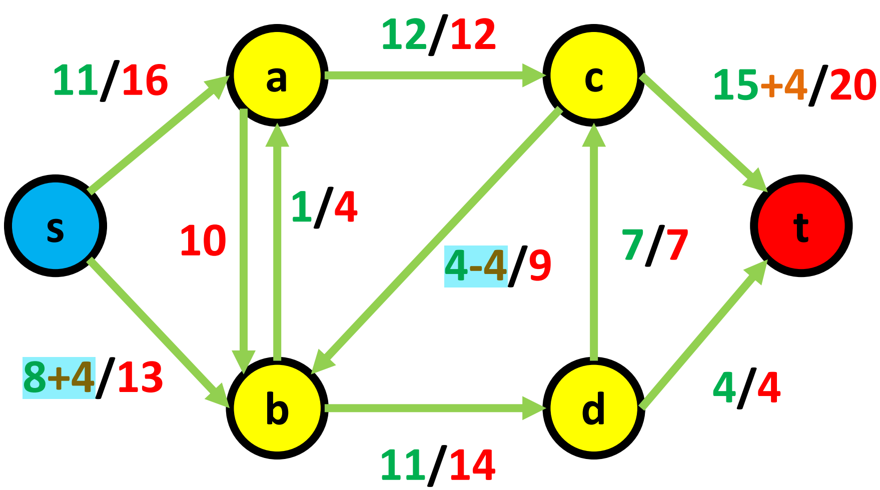


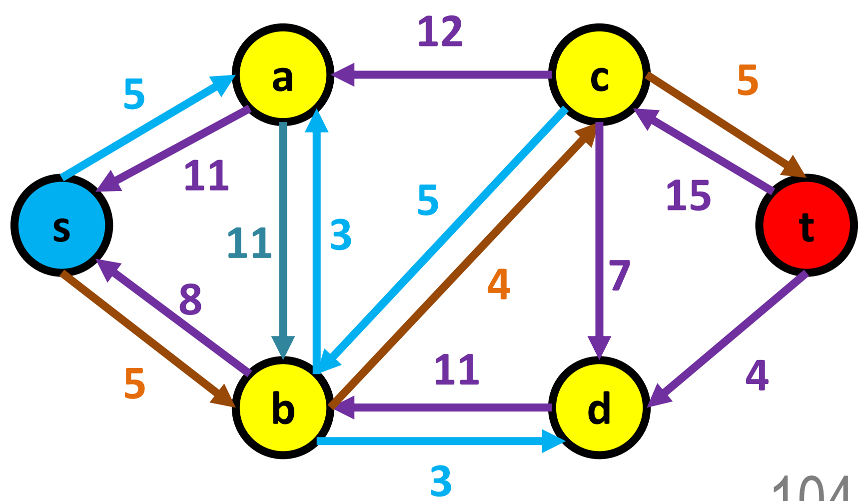
- Is there a path here?
  - s -> b -> c -> t... let us look at this one first
  - That the smallest value which is 4
    - This value can flow from source to target



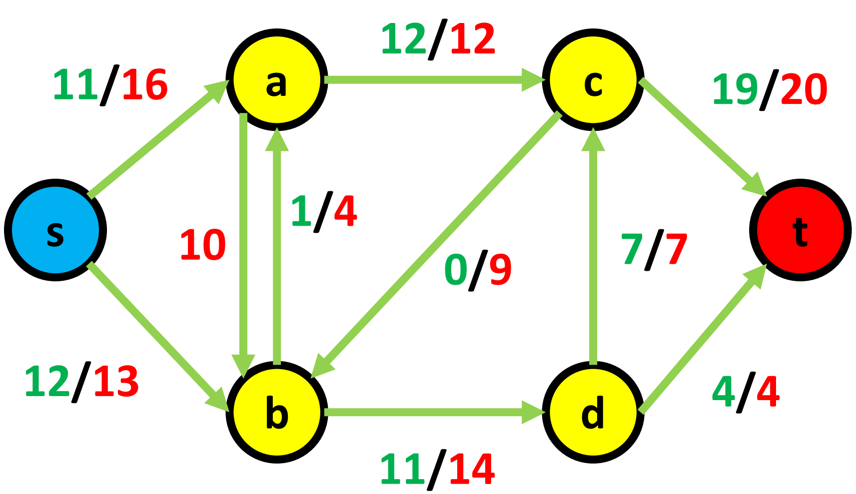
Path Augmentation
Traversal in residual network

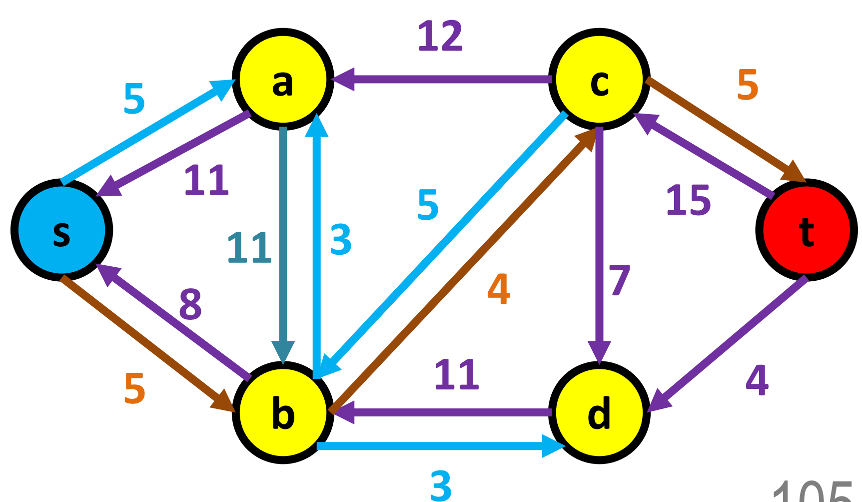
- Is there a path here?
  - s -> b -> c -> t… let us look at this one first
  - That the smallest value which is 4
    - This value can flow from source to target





- Is there a path here?
  - s -> b -> c -> t… let us look at this one first
  - That the smallest value which is 4
    - This value can flow from source to target

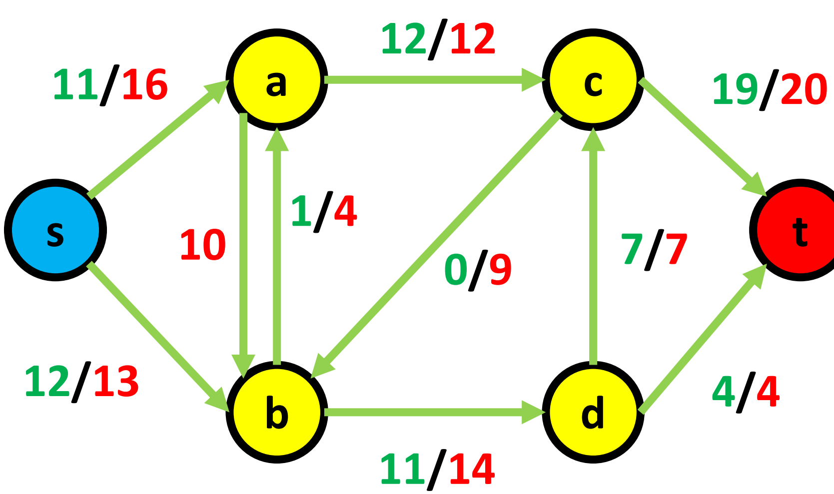


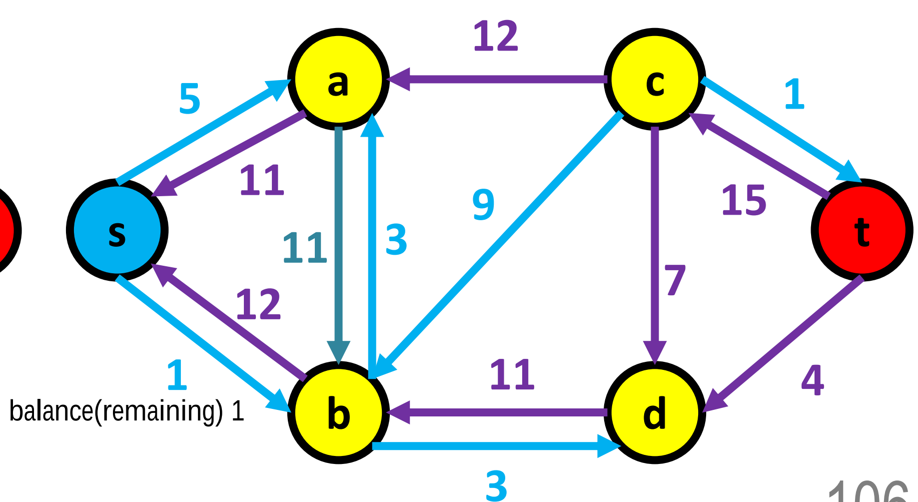


Path Augmentation

Traversal in residual network

- Is there a path here?
  - s -> b -> c -> t… let us look at this one first
  - That the smallest value which is 4
    - This value can flow from source to target

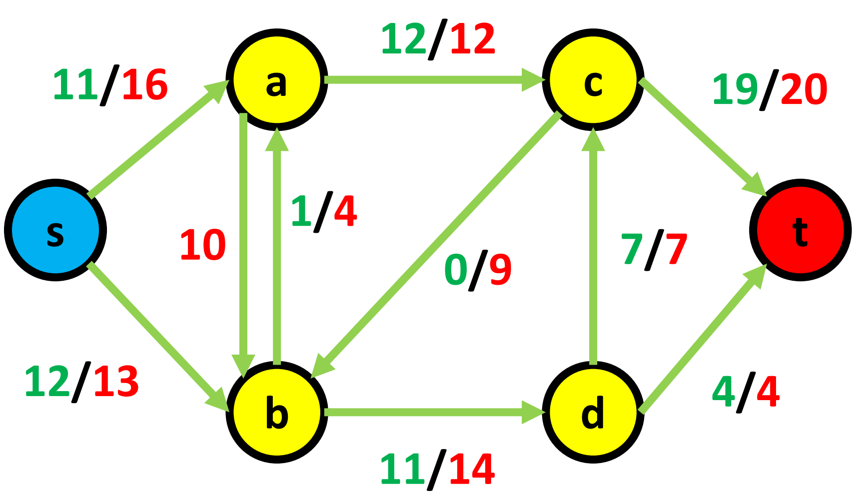


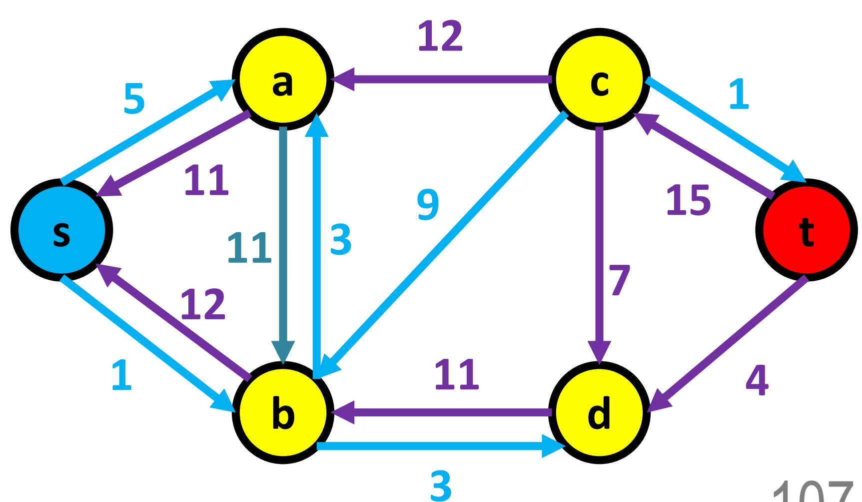


Path Augmentation
Traversal in residual network

- Is there a path here?
  - s -> b -> c -> t… let us look at this one first
  - That the smallest value which is 4
    - This value can flow from source to target
- With this, the flow in the network goes from 19 to 23!

BFS on residual network,
to find the find the path from s to t
backtrack path to find the previous edge
remember minimum flow value
update on the original network
then due to to update on original network
then update residual network

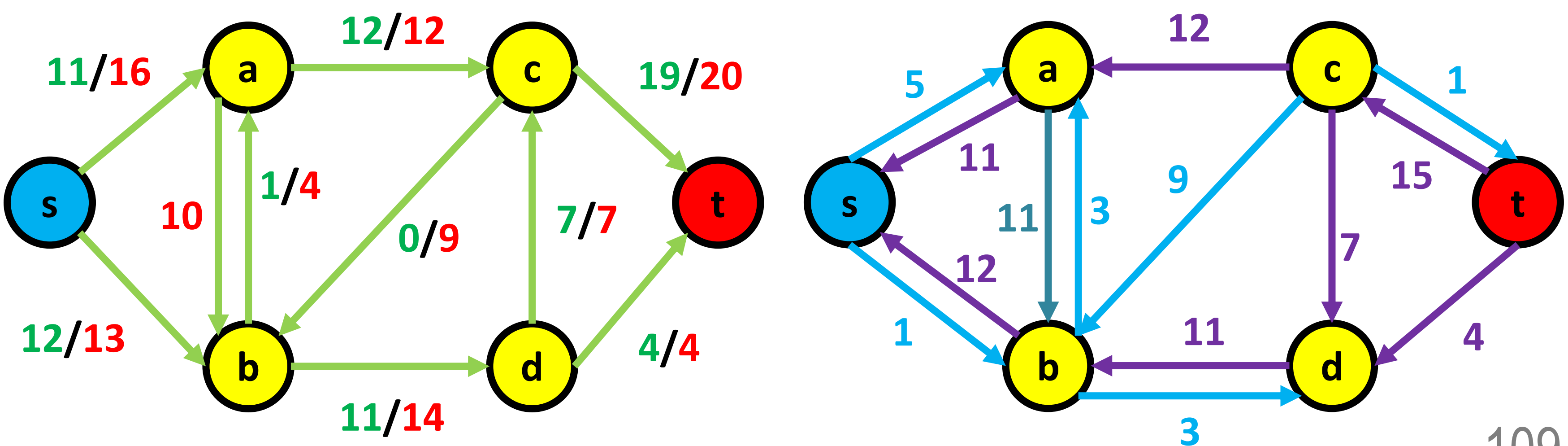




Questions?

- Is there another path here?

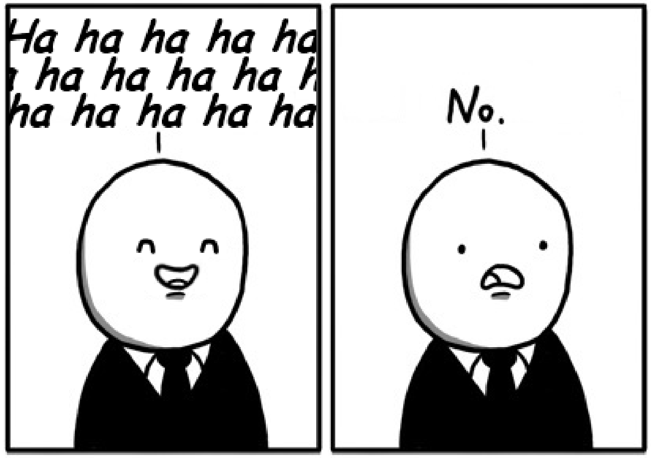
finish when residual network has not path from s to t

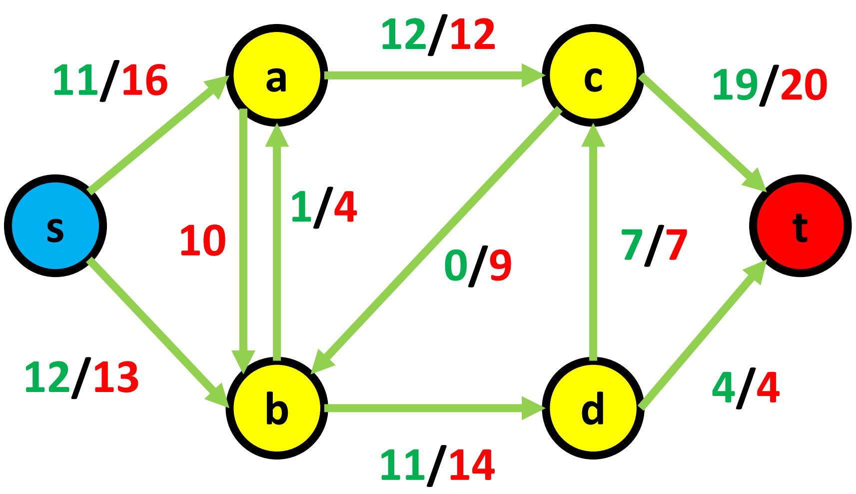


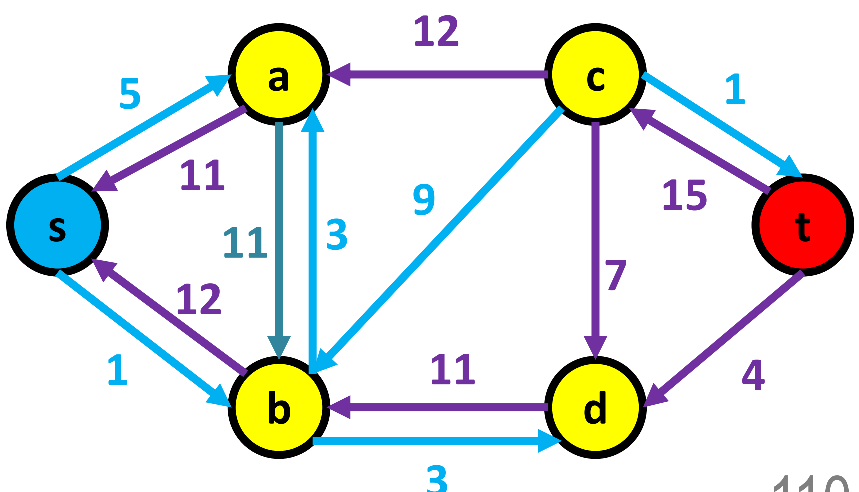
Path Augmentation
Traversal in residual network

- Is there another path here?
  - NO!

reach the point that has no another path







Questions?

Ford-Fulkerson Method

Finding the maximum flow

- So we have learnt all the components for the Ford-Fulkerson Method…

- So we have learnt all the components for the Ford-Fulkerson Method…
  - Residual network
  - Path augmentation

- So we have learnt all the components for the Ford-Fulkerson Method…
  - Residual network
  - Path augmentation

- Now let us look at the algorithm…

Break?

Ford-Fulkerson Method
Finding the maximum flow

- Let break it down slowly…

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

build residual network from graoh
see 62

repeat until there is no path
from s to t
if there is a path,
meaning: flow can be increased
corresponding path can be
augmented

see slide 88 to 107
update flow with capacity

Ford-Fulkerson Method

Finding the maximum flow

- Let break it down slowly…

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

See slide
62 to 86

Ford-Fulkerson Method

Finding the maximum flow

MONASH University

- Let break it down slowly…

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

See slide 88 to 107

Ford-Fulkerson Method

Finding the maximum flow

- Let break it down slowly…

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ends when it is like slide 109 to 110

- And that's all…

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Questions?

- Want a trial run?

Ford-Fulkerson Method

Finding the maximum flow

- Want a trial run?

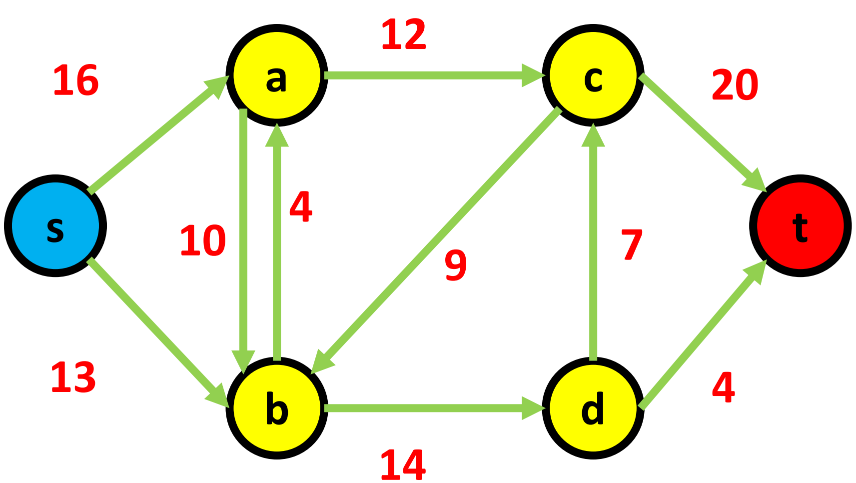
```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

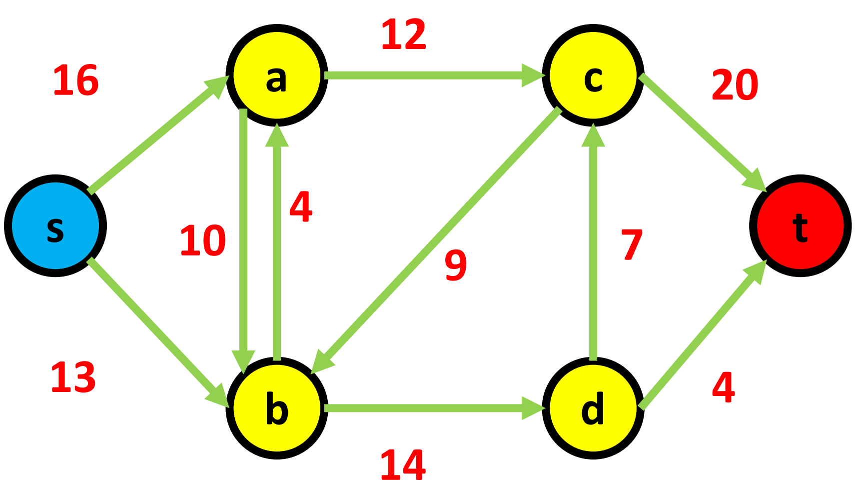
- Want a trial run?



Ford-Fulkerson Method
Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0

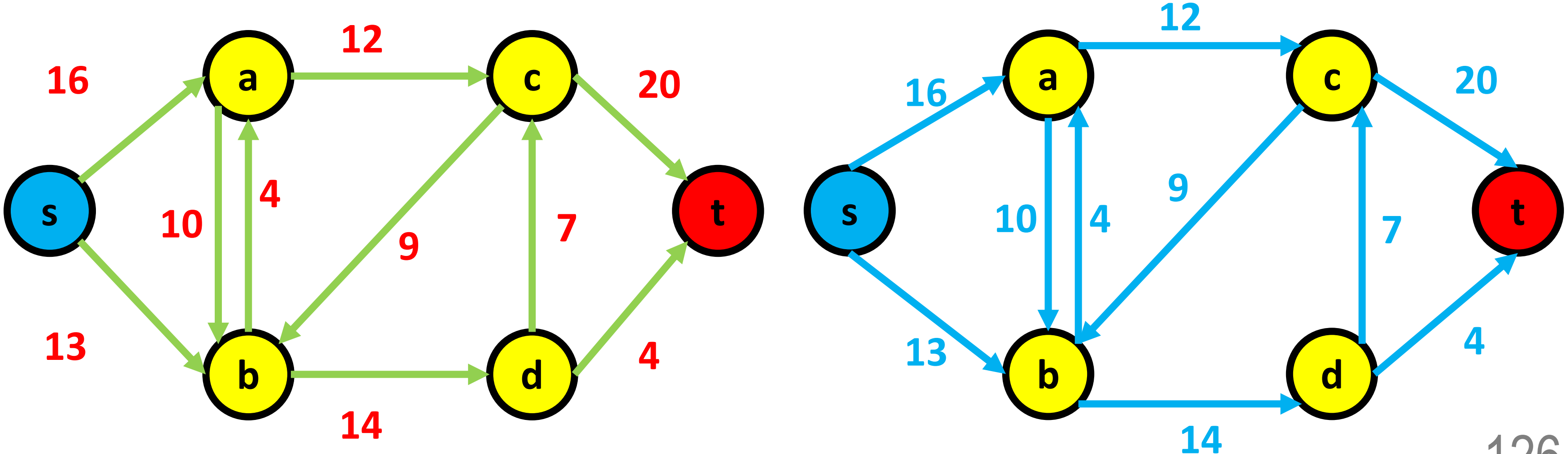


Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network

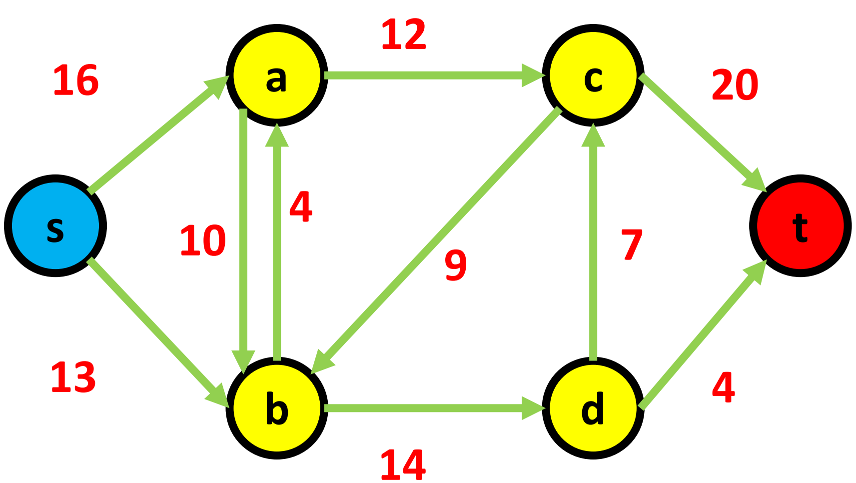


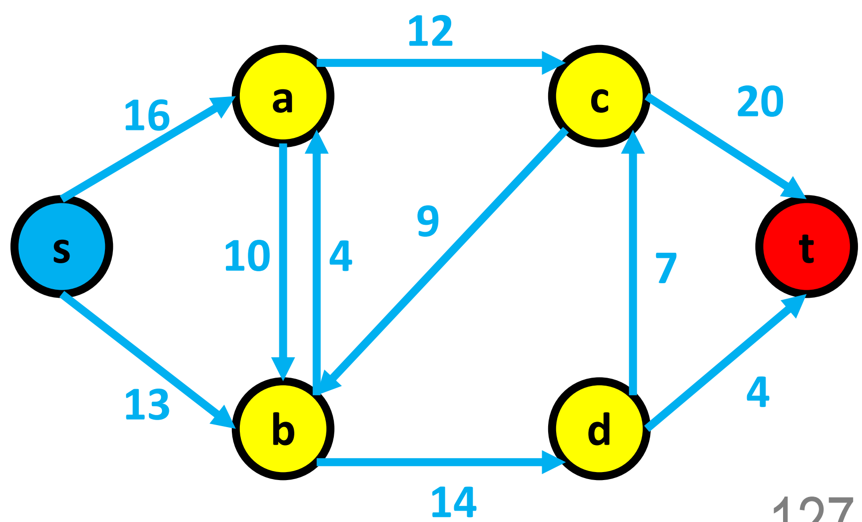
Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?



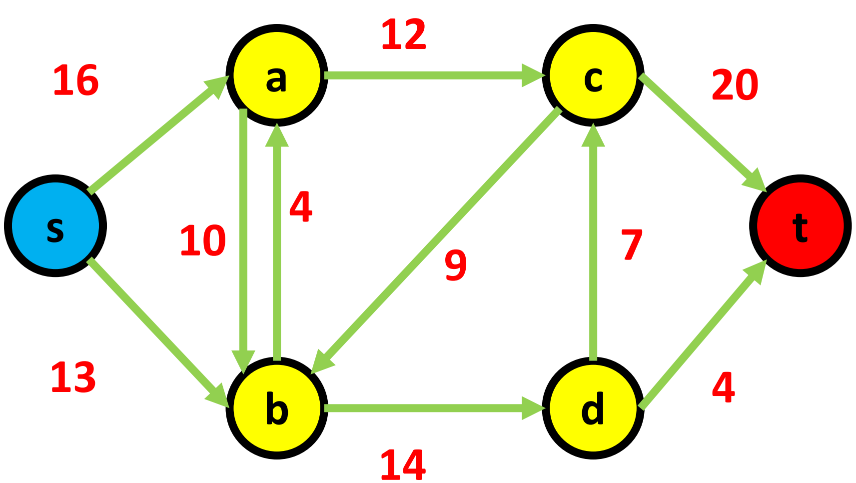


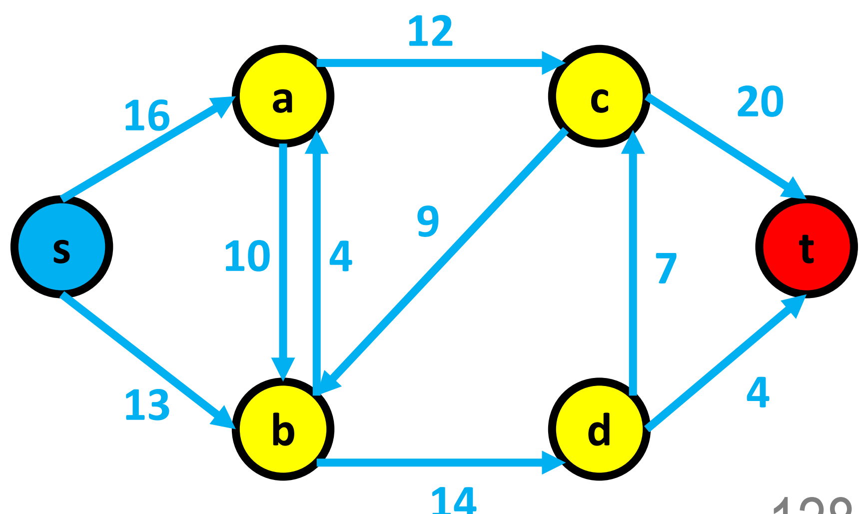
Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path?



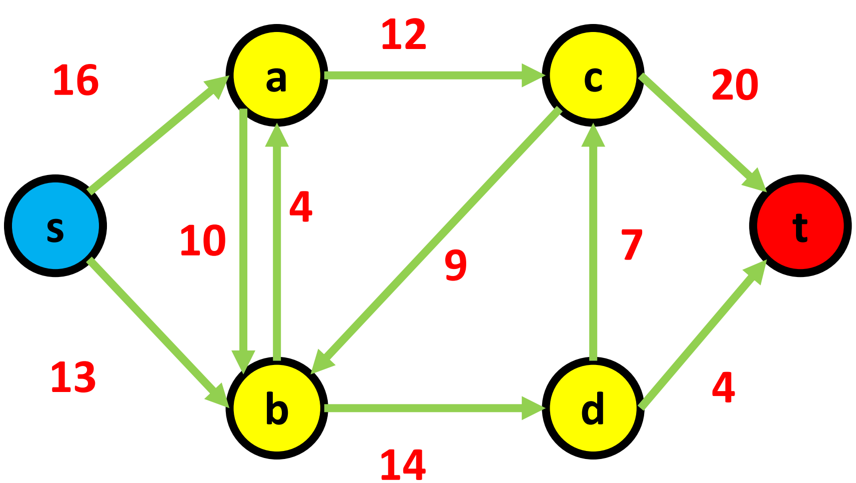


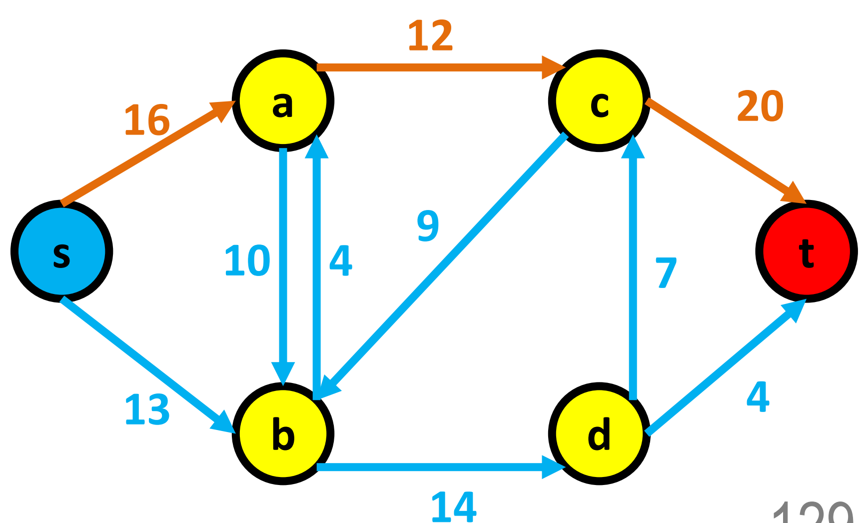
Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path? s -> a -> c -> t



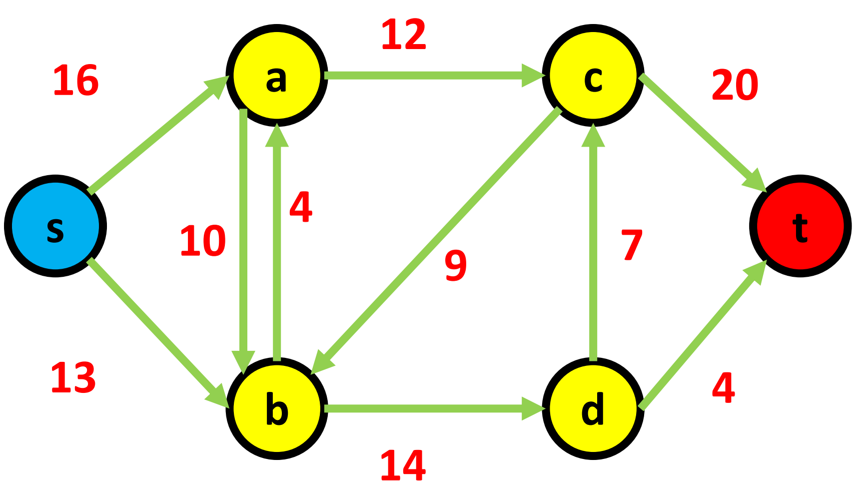


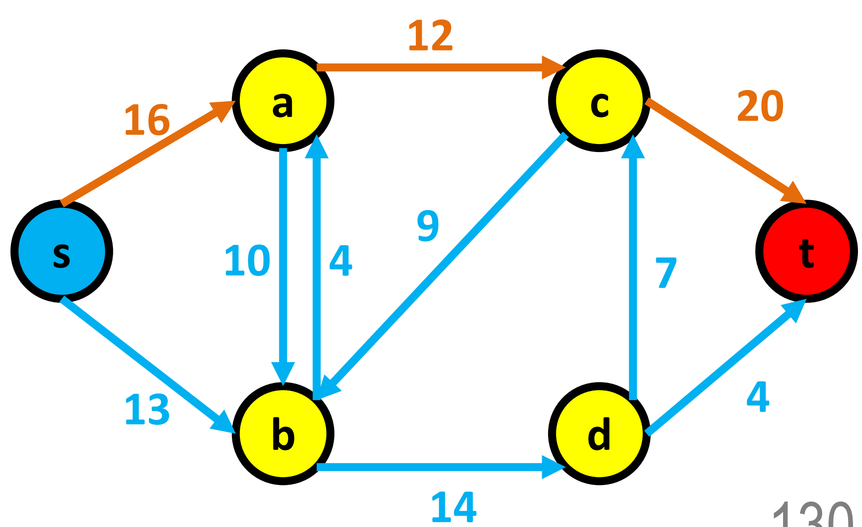
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path? s -> a -> c -> t
    - Take the smallest value, 12



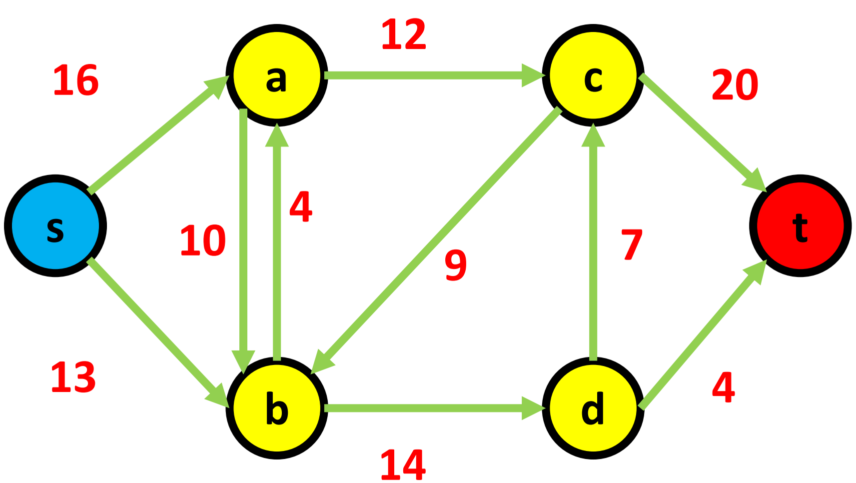


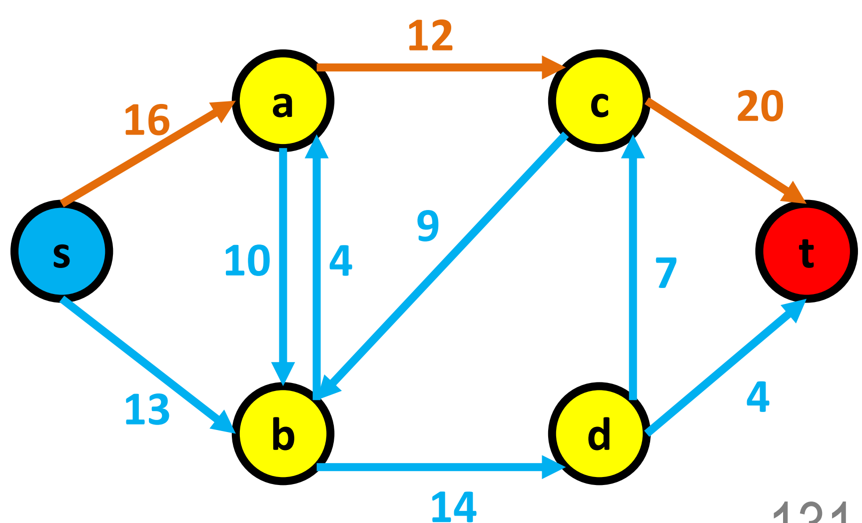
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path? s -> a -> c -> t
    - Take the smallest value, 12 and update the network



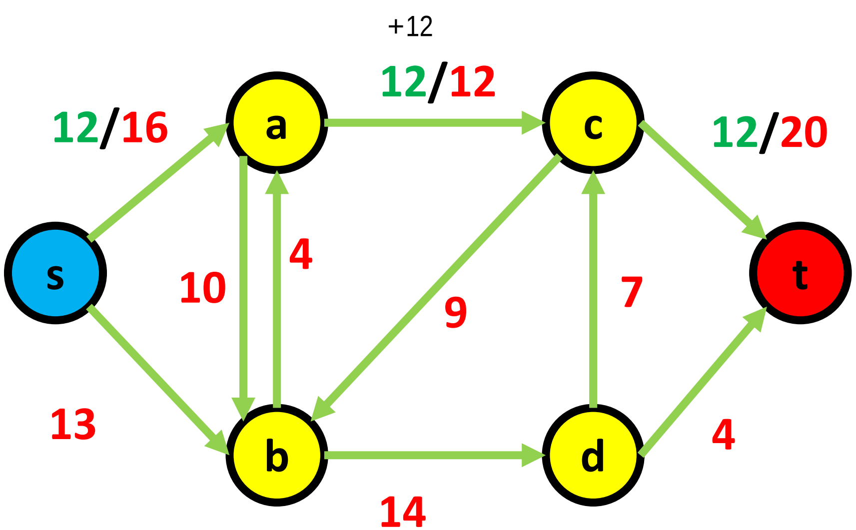


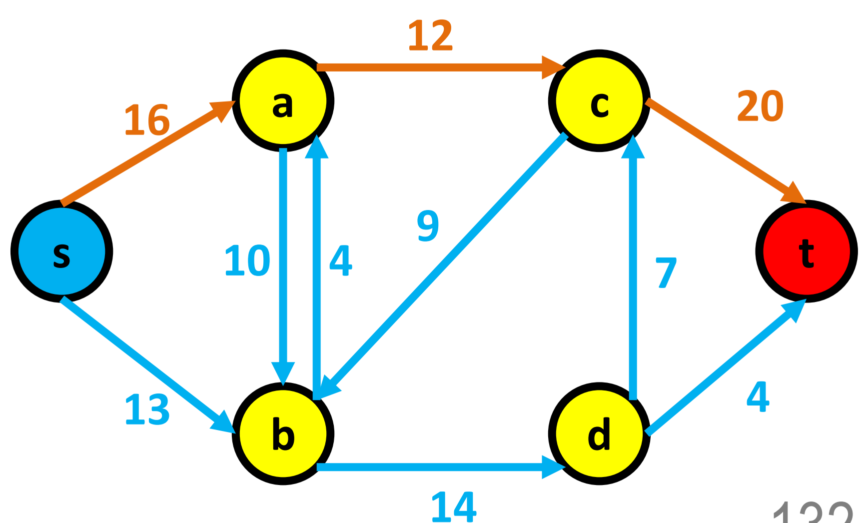
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path? s -> a -> c -> t
    - Take the smallest value, 12 and update the network



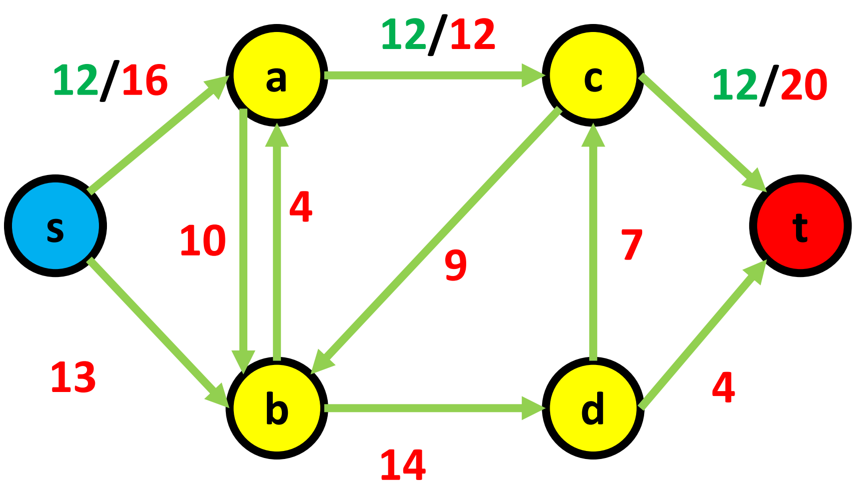


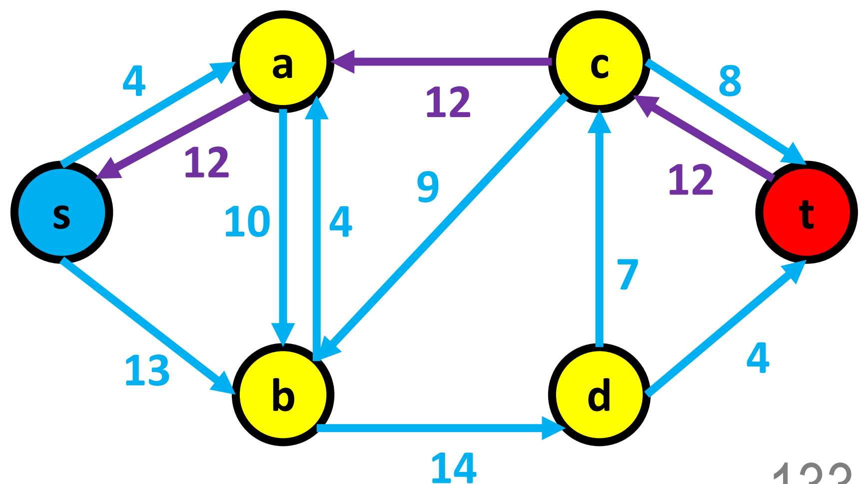
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 0
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path? s -> a -> c -> t
    - Take the smallest value, 12 and update the network
    - Update the residual network as well



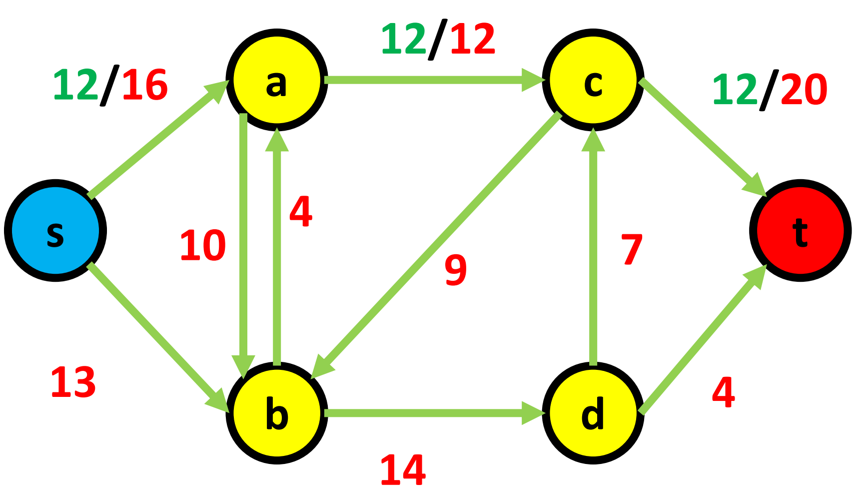


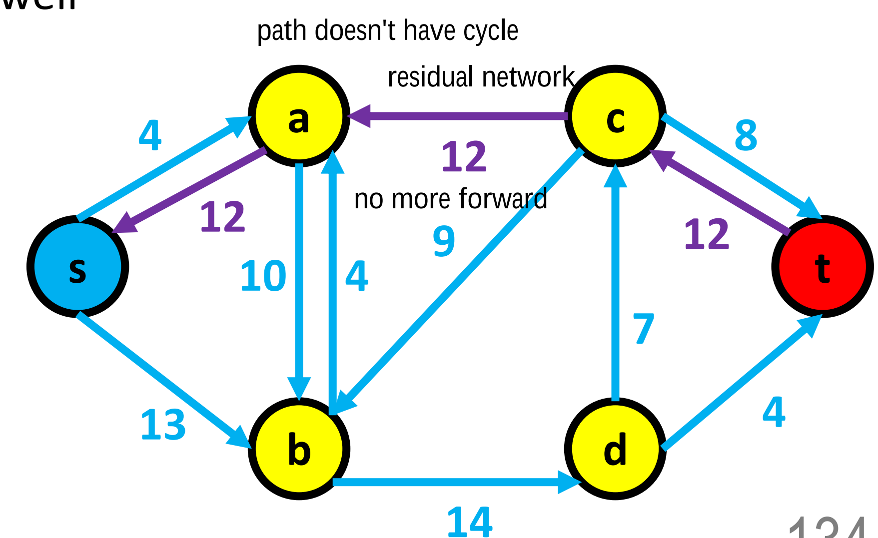
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = **<u>0+12 = 12</u>**
  - Make residual network
  - Do we have an augmenting path?
    - Yes! What is the path? s -> a -> c -> t
    - Take the smallest value, 12 and update the network
    - Update the residual network as well



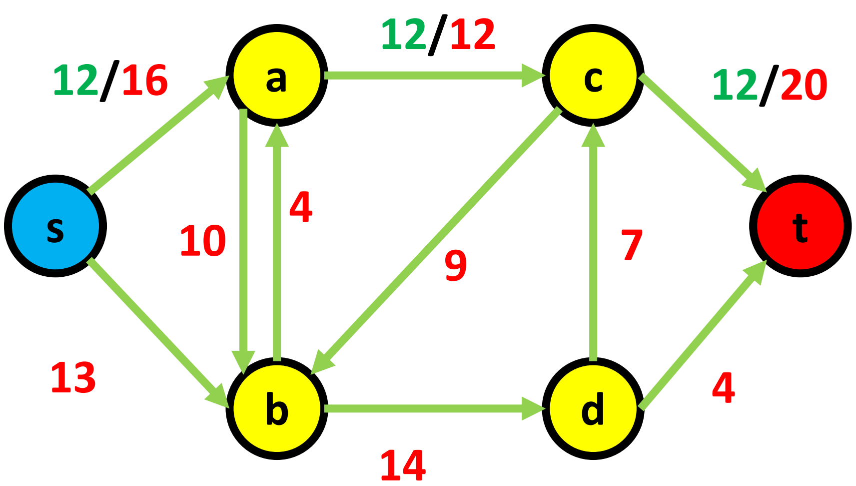


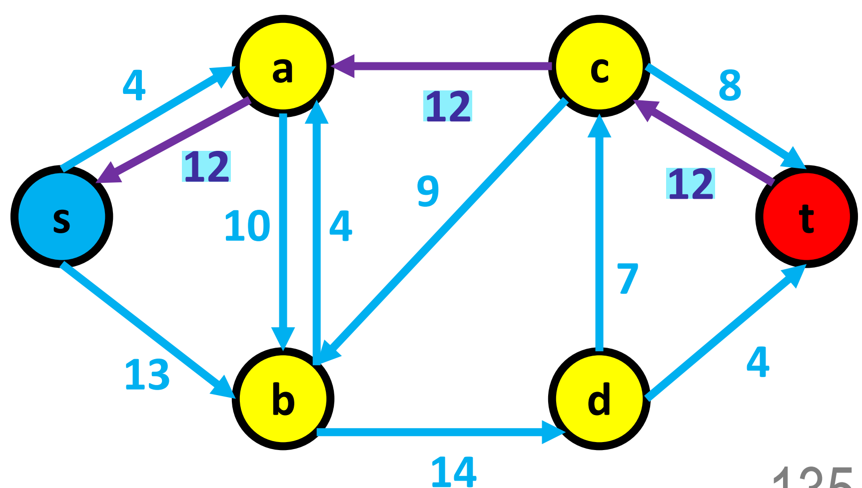
Ford-Fulkerson Method
Finding the maximum flow

- Want a trial run?
  - Flow = 12
  - So we just repeat it

path does not have duplicate

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```



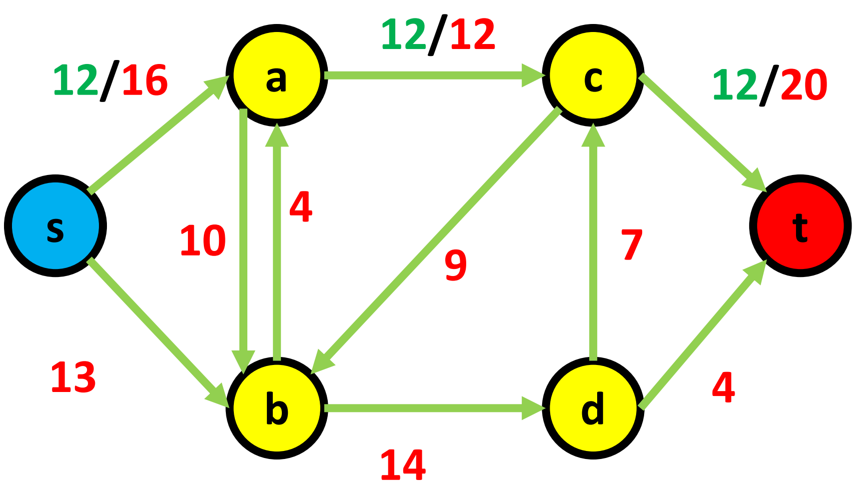


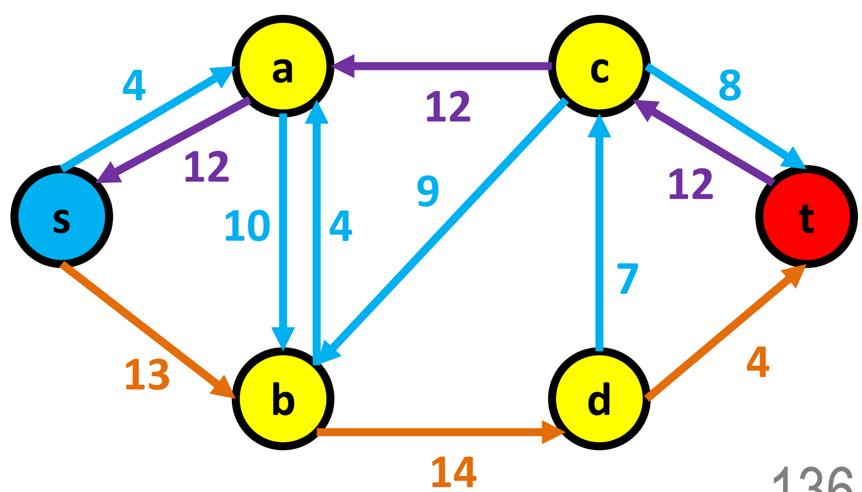
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 12
  - So we just repeat it with path SBDT





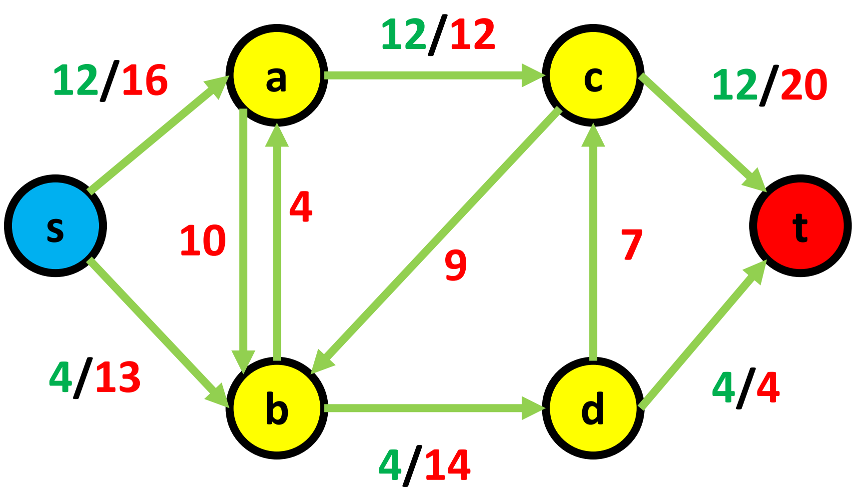
Ford-Fulkerson Method

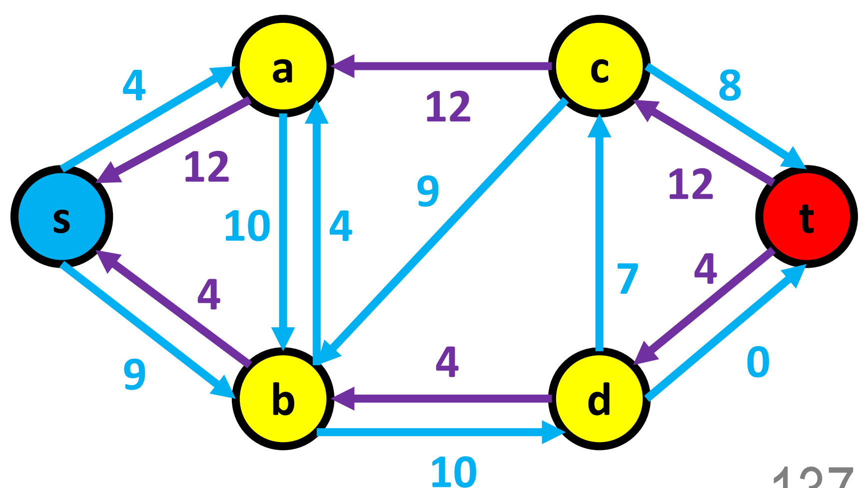
Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 12+4 = 16
  - So we just repeat it with path SBDT

another path



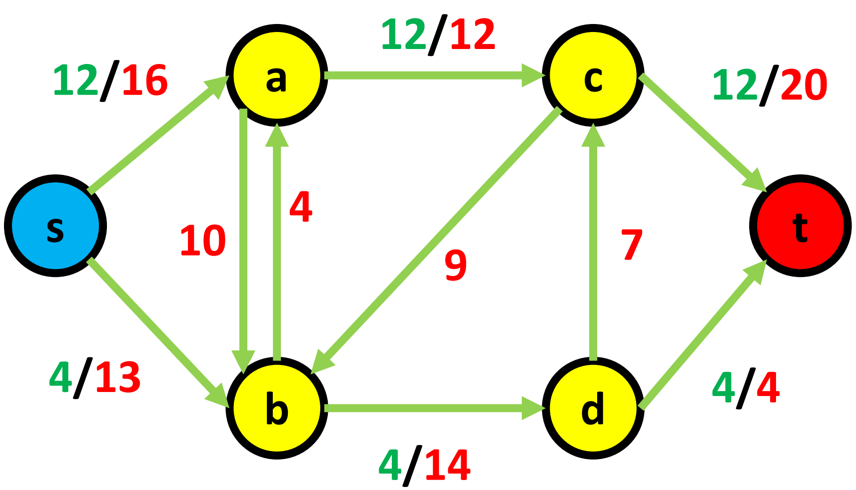


Ford-Fulkerson Method

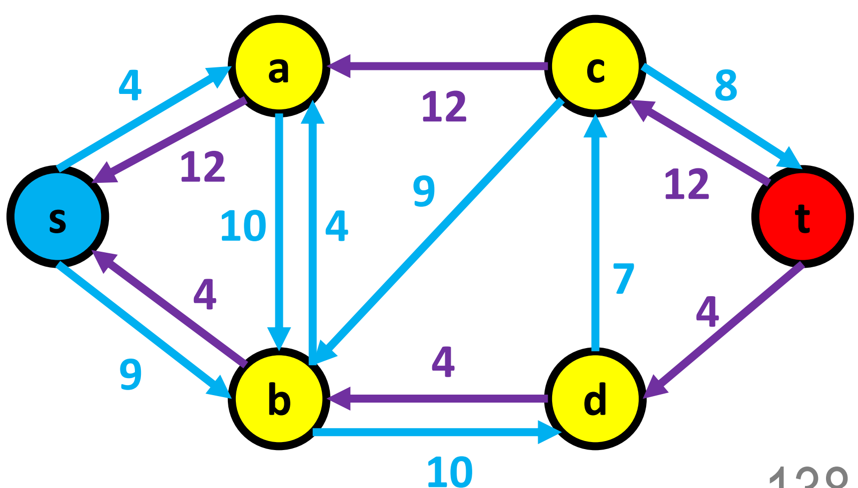
Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 16



residual network
the capacity that is still available

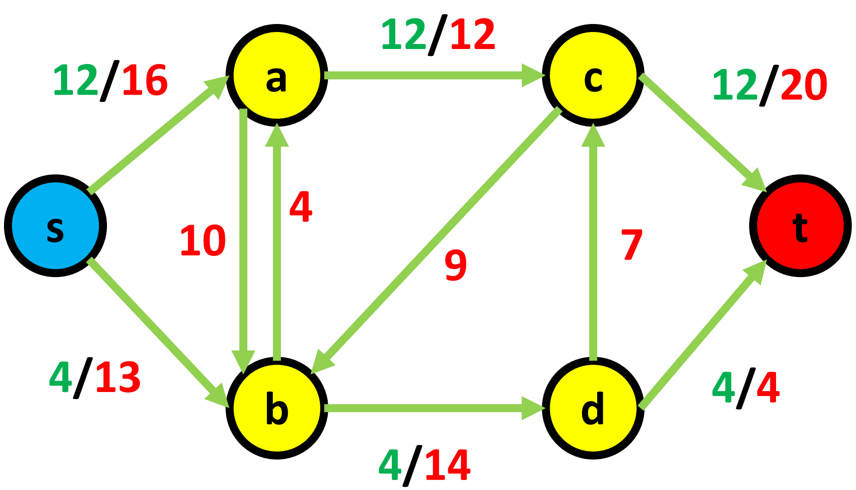


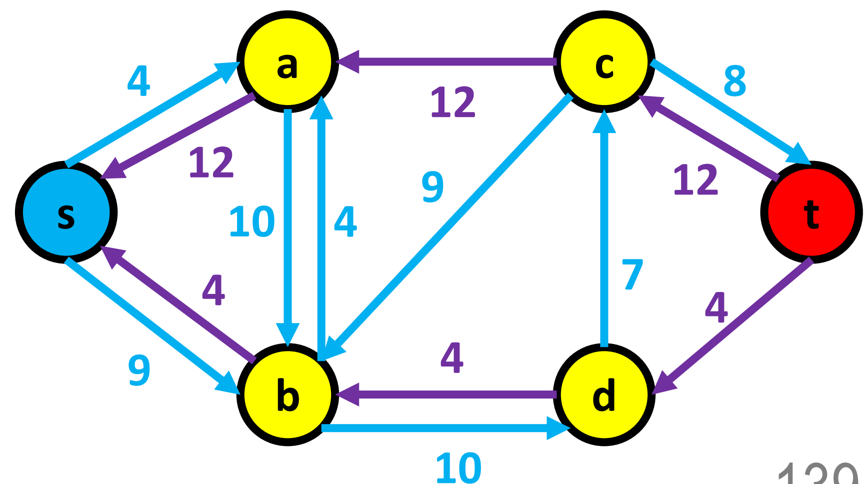
Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 16
  - So now we repeat again



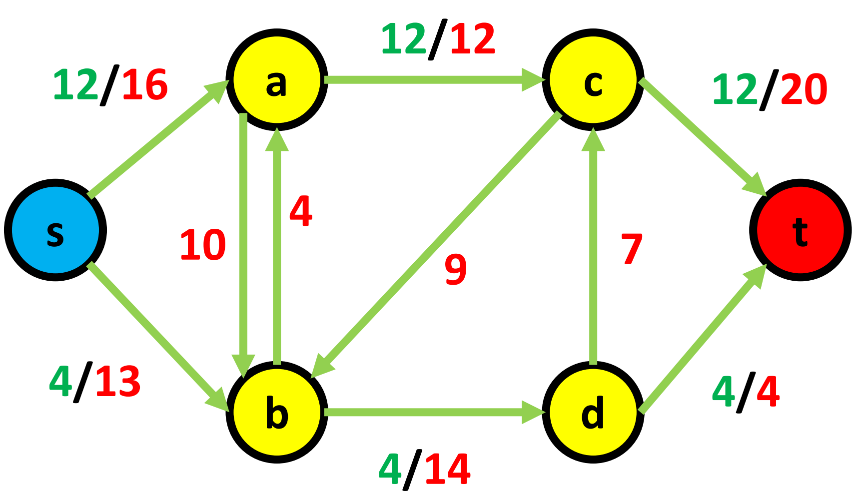


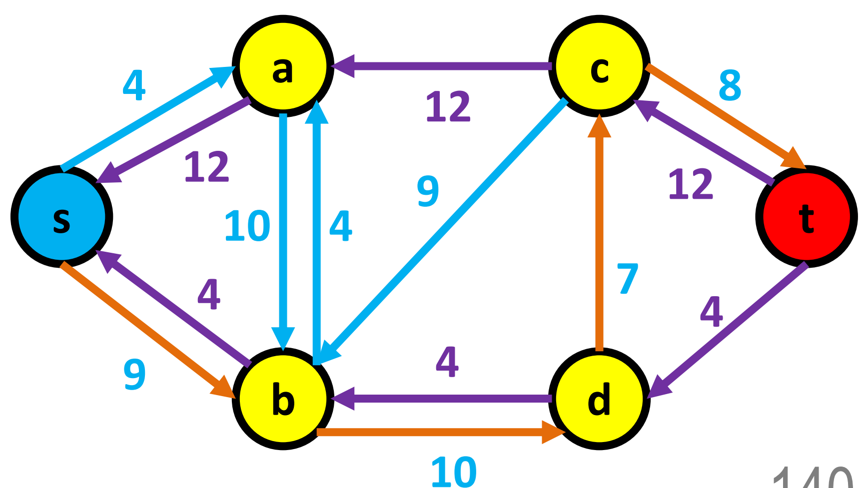
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 16
  - So now we repeat again with path SBDCT

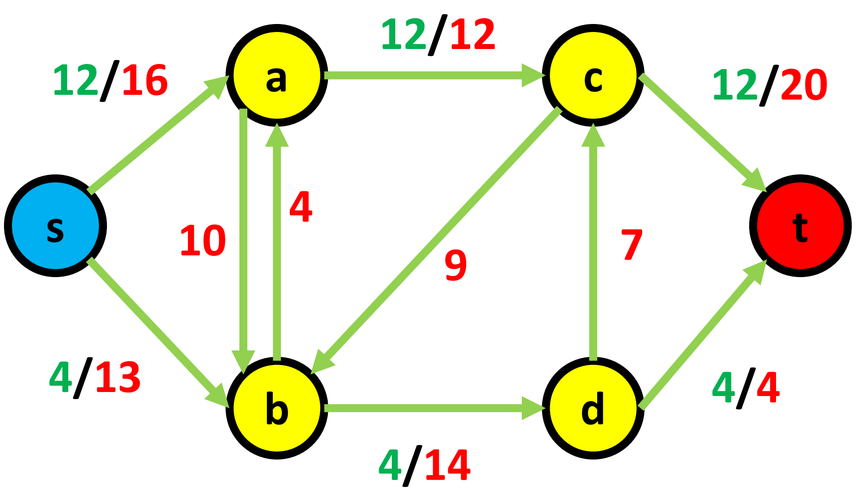


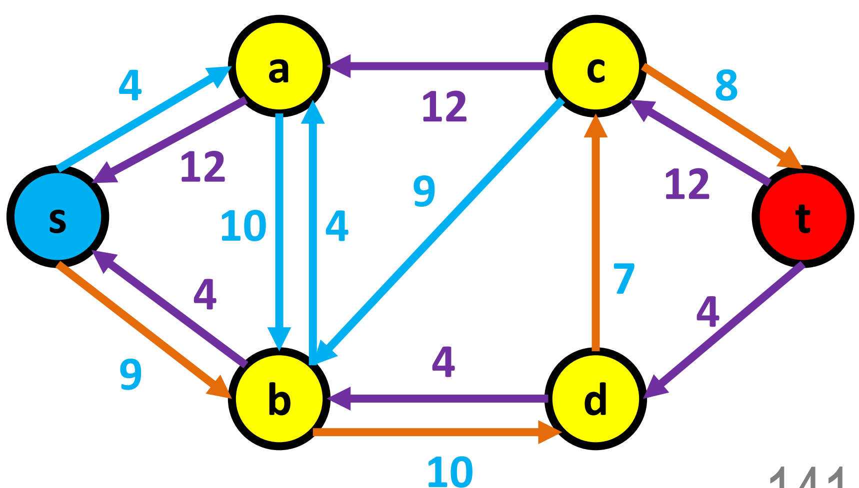


Ford-Fulkerson Method
Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 16+7
  - So now we repeat again with path SBDCT



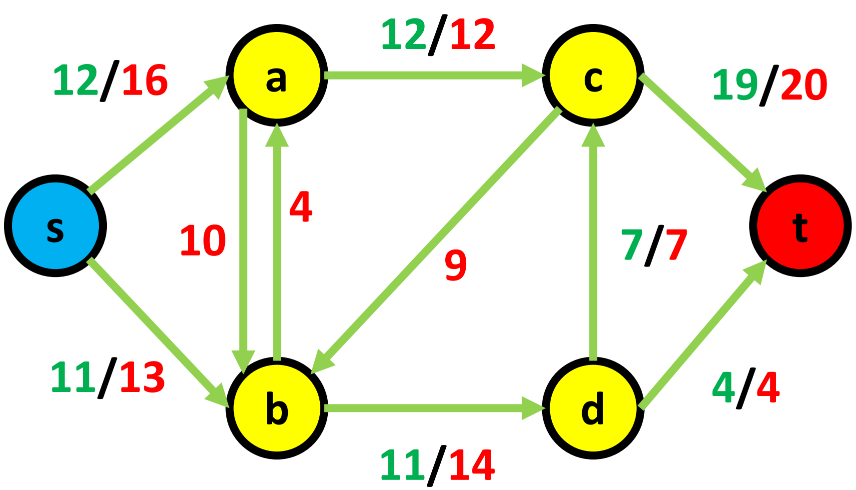


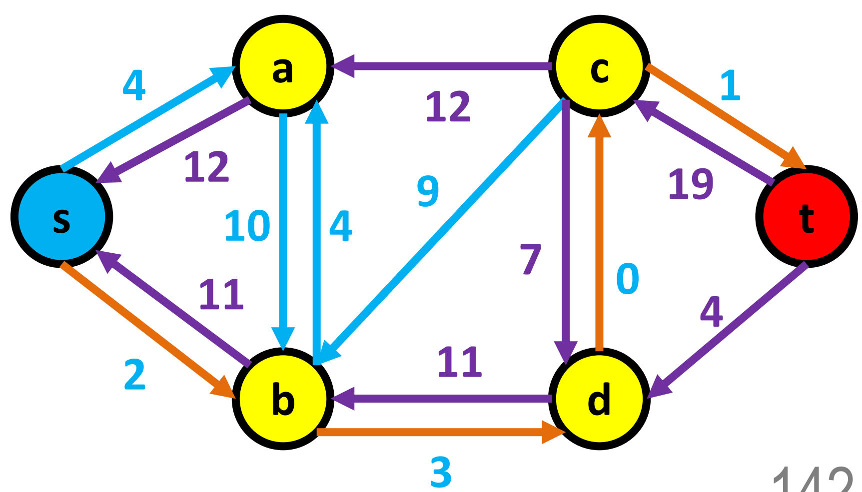
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 16+7
  - So now we repeat again with path SBDCT



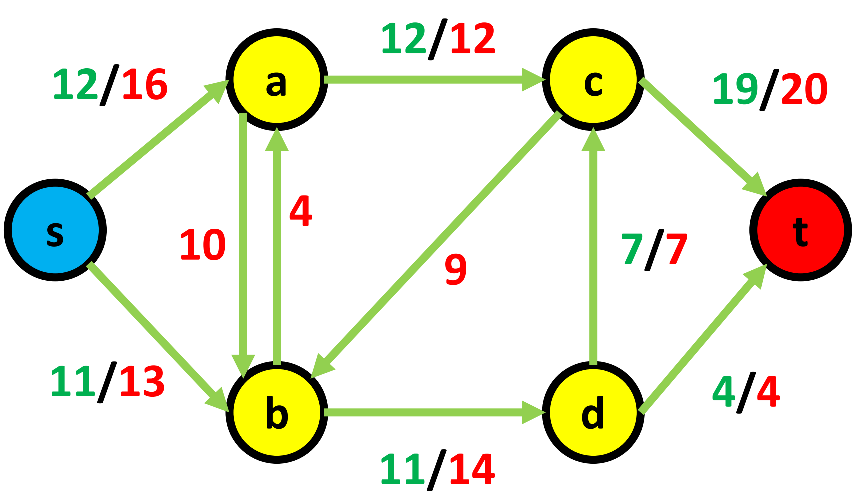


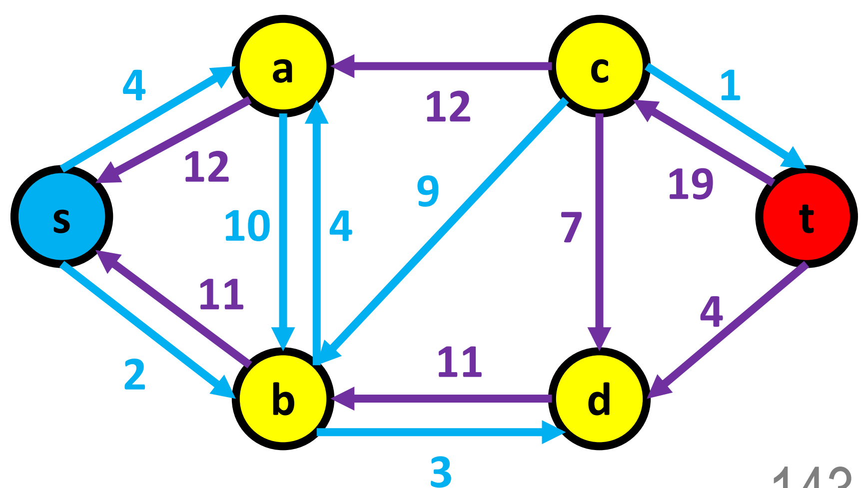
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 23



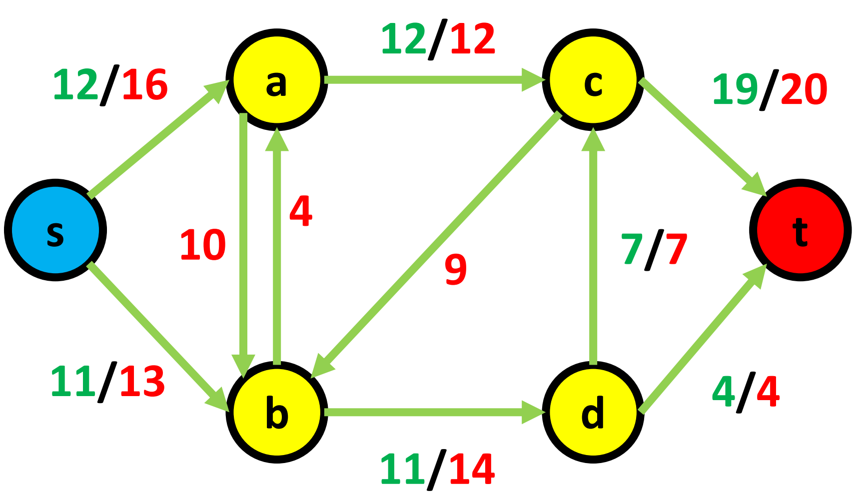


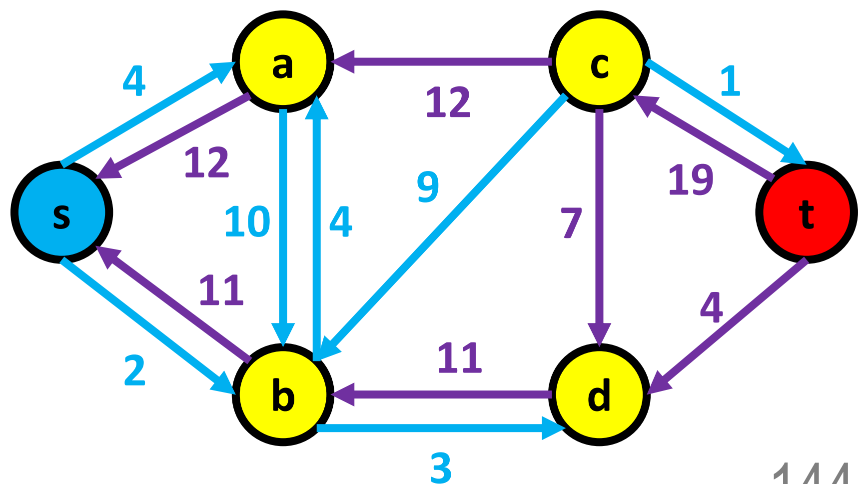
Ford-Fulkerson Method

Finding the maximum flow

- Want a trial run?
  - Flow = 23
  - Do we still have a path?

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```



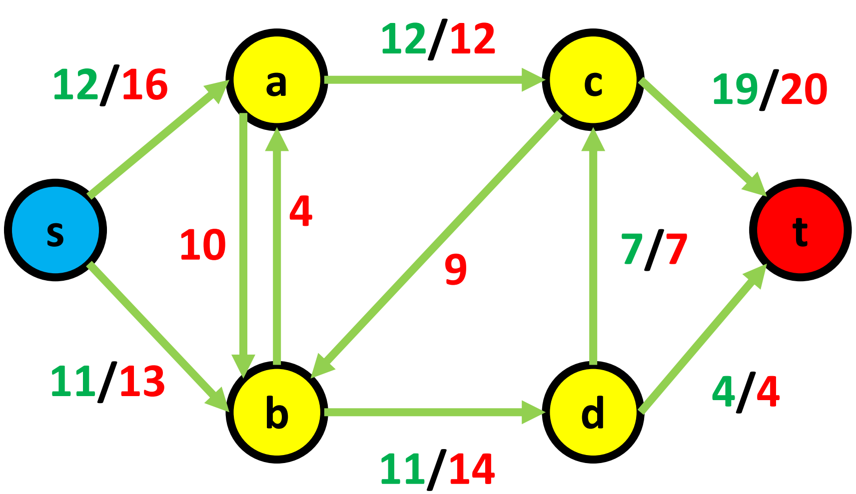


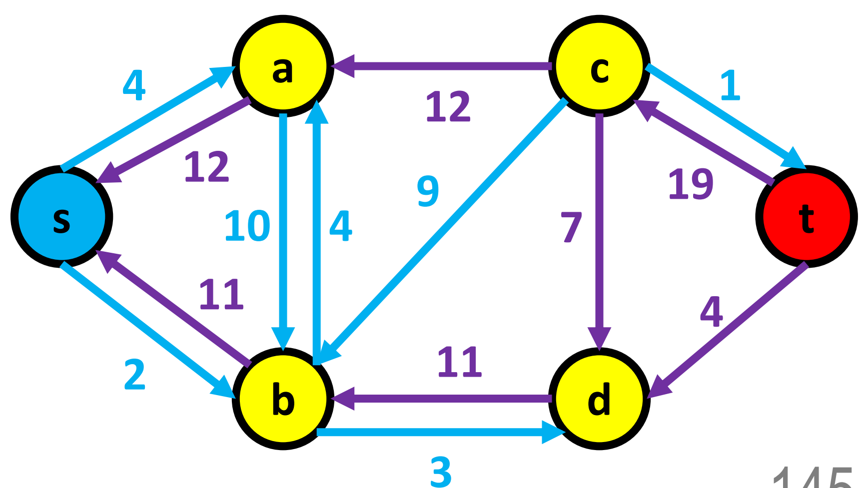
Ford-Fulkerson Method

Finding the maximum flow

- Want a trial run?
  - Flow = 23 max, run out of possible routes
  - Do we still have a path?
    No more, so we are done!

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```



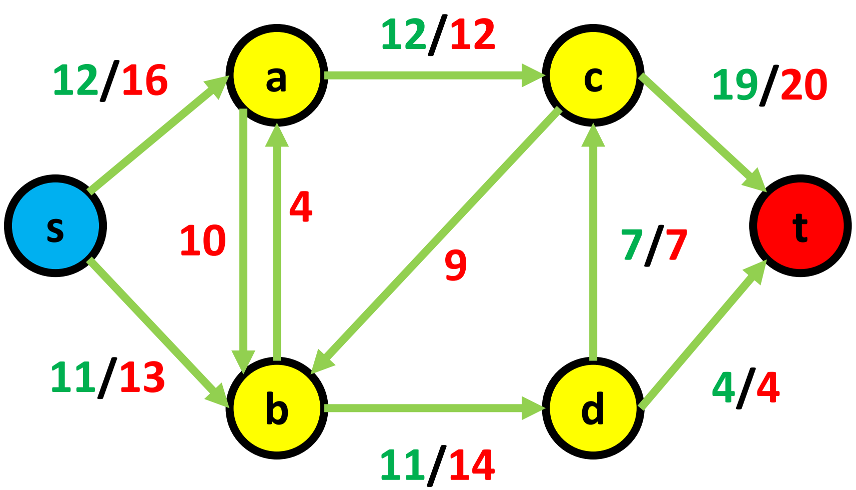


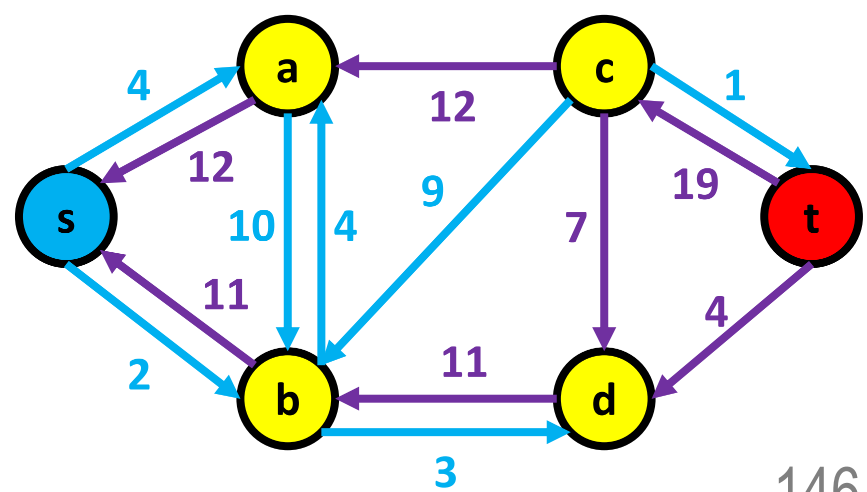
Ford-Fulkerson Method

Finding the maximum flow

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 23
  - Do we still have a path?
    No more, so we are done!
- Note: The answer in MUA's slide is <u>same value</u>, but different network flow



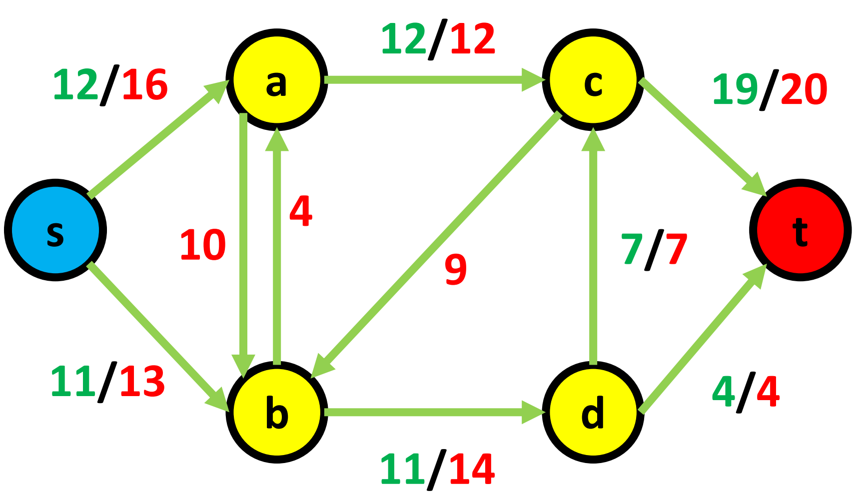


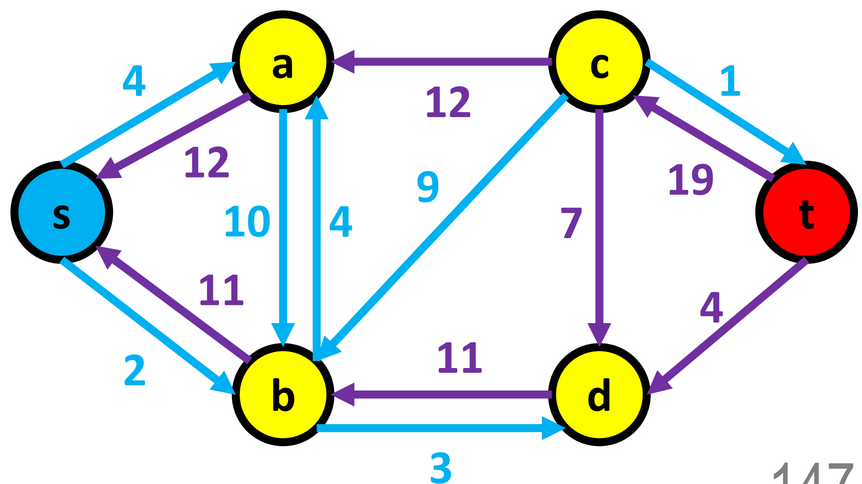
Ford-Fulkerson Method

Finding the maximum flow

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Want a trial run?
  - Flow = 23 unique maximum flow
  - Do we still have a path?
    No more, so we are done!
- Note: The answer in MUA's slide is <u>same value</u>, but different network flow. Thus, answer not unique!



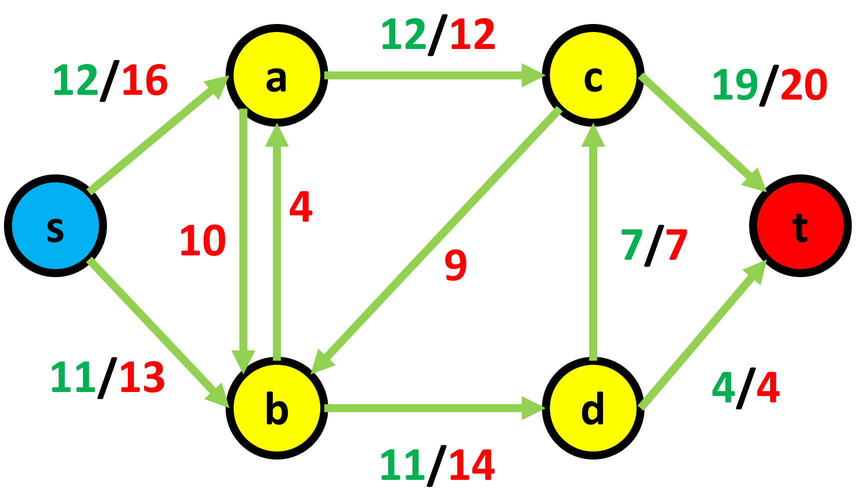


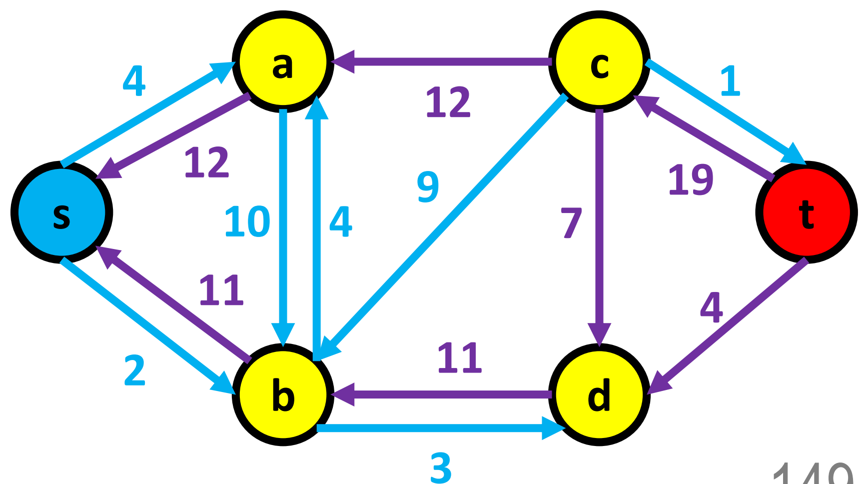
Questions?

Ford-Fulkerson Method

Finding the maximum flow

- Also remember, we maintain all of the properties!

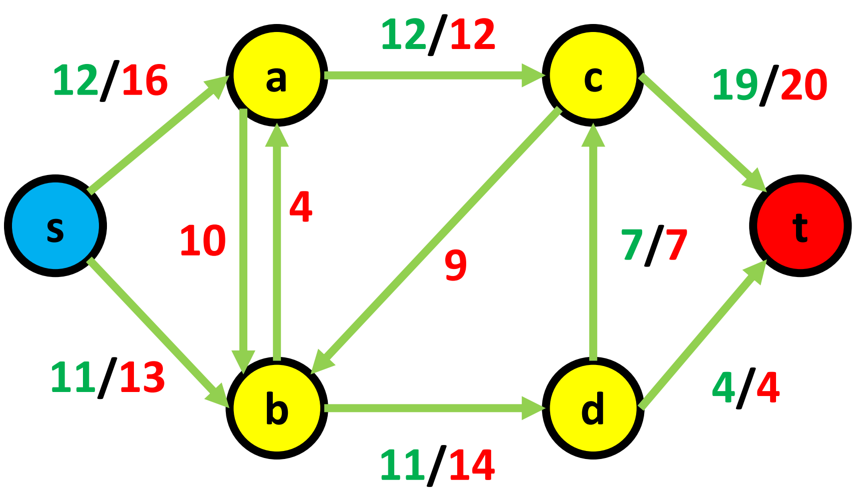


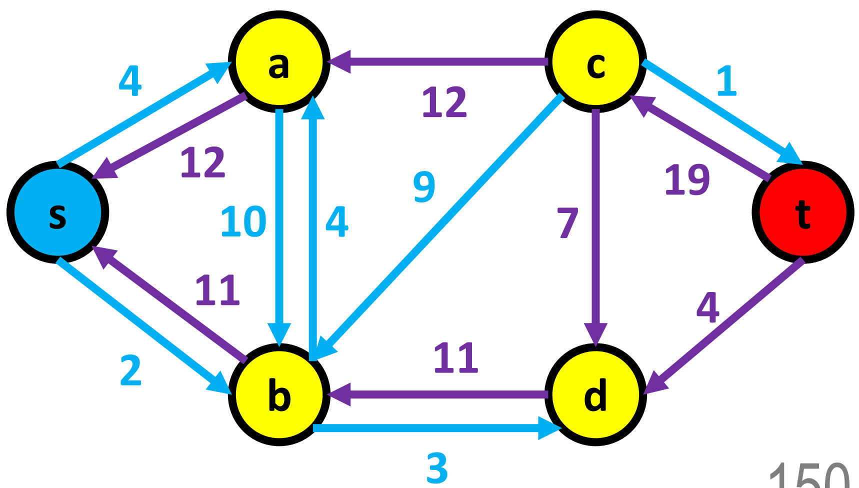


Ford-Fulkerson Method

Finding the maximum flow

- Also remember, we maintain all of the properties!
  - The flow doesn't exceed capacity
  - Incoming to a vertex is the same as outgoing to a vertex





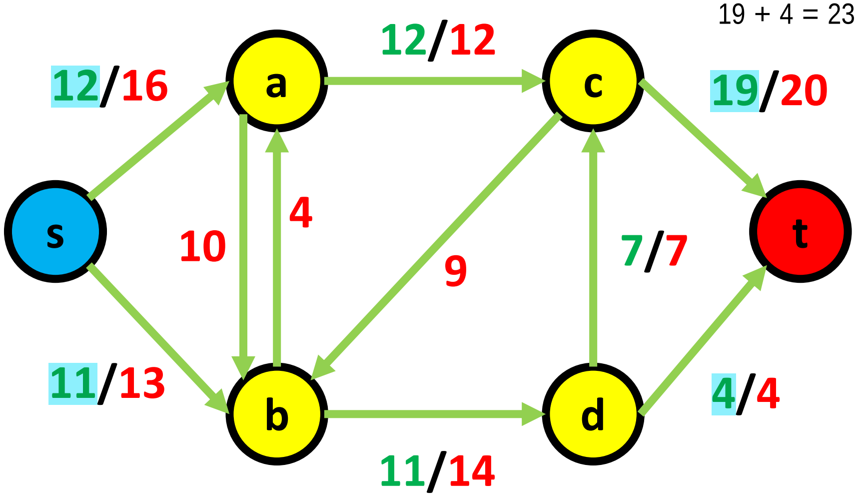
- Also remember, we maintain all of the properties!
  - Capacity constraint: The flow doesn't exceed capacity
  - Flow conservation: Incoming to a vertex is the same as outgoing to a vertex

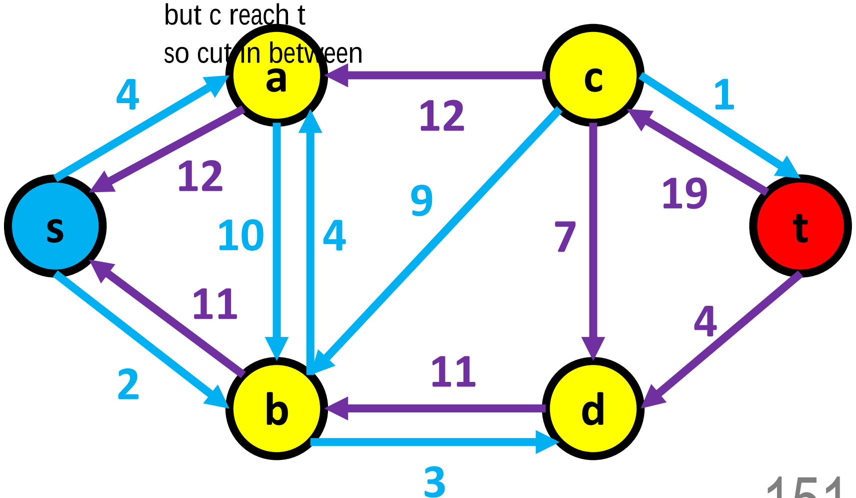
outgoing from source = incoming to destination

see residual network
s can go to a, b, d but not c
sine no more forward remaining capacity
so no more forward edge to reach c and t
to be reachable to c and t
but c reach t
so cut in between

12 + 11 = 23

19 + 4 = 23





Questions?

Ford-Fulkerson Method

Complexity analysis

- So what is the complexity?

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- So what is the complexity?
  - Residual network have a total of 2E edges, thus O(E)

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

O(E)

- So what is the complexity?
  - Everything with augmenting path is O(V+E). Why?

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ford-Fulkerson Method

Complexity analysis

- So what is the complexity?
  - Everything with augmenting path is O(V+E). Why? Cause everything is just BFS! Or DFS if you wish to be hipster…

    recommend BFS

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

O(V+E)

Ford-Fulkerson Method

Complexity analysis

MONASH University

- So what is the complexity?
  - But how many times do the loop repeat?

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ford-Fulkerson Method

Complexity analysis

- So what is the complexity?
  - But how many times do the loop repeat? Total of O(F) where F is the flow itself.

    number of augmentation path

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

O(F)

Ford-Fulkerson Method

Complexity analysis

MONASH University

- So what is the complexity?
  - But how many times do the loop repeat? Total of O(F) where F is the flow itself. Why?

```
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

O(F)

Ford-Fulkerson Method

Complexity analysis

- So what is the complexity?
  - But how many times do the loop repeat? Total of O(F) where F is the flow itself. Why? Because we increase the flow in each iteration till we can't anymore. Minimum increment is F = F + 1

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

O(F)

Ford-Fulkerson Method

Complexity analysis

- Total?

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ford-Fulkerson Method

Complexity analysis

- Total? O(FV + FE) = O(FE)

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ford-Fulkerson Method

Complexity analysis

- Total? O(FV + FE) = O(FE)

  since F is not part of the input

  – This is pseudo-polynomial since F can be very very large…

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

- Total? O(FV + FE) = O(FE)
  - This is pseudo-polynomial since F can be very very large…
  - But can be proven to be O(VE^2) via Edmonds-Karp (in FIT3155)

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Ford-Fulkerson Method

Complexity analysis

- Total? O(FV + FE) = O(FE)
  - This is pseudo-polynomial since F can be very very large…
  - But can be proven to be O(VE^2) via Edmonds-Karp (in FIT3155)

```python
def ford_fulkerson(my_graph):
    # initialzie flow
    flow = 0
    # initialize the residual network
    residual_network = ResidualNetwork(my_graph)
    # as long s there is a augmenting path
    while residual_network.has_AugmentingPath():
        # take the path
        path = residual_network.get_AugmentingPath()
        # augment the flow equal to the residual capacity
        flow += path.residual_capacity
        # updating the residual network
        residual_network.augmentFlow(path)
    return flow
```

Questions?

Break?

- But does it work???

- Assume every capacity is integer

- Assume every capacity is integer
- Flow always increase by 1 at each iteration and we know flow is finite

Ford-Fulkerson Method

Proof of correctness

- Assume every capacity is integer
- Flow always increase by 1 at each iteration and we know flow is finite

  until max Flow

- Therefore the algorithm do terminate!

  finitely reach compacity of the some necessary edge in the path

Ford-Fulkerson Method

Proof of correctness

- Assume every capacity is integer
- Flow always increase by 1 at each iteration and we know flow is finite
- Therefore the algorithm do terminate!

- But does it terminate with the max flow?

Ford-Fulkerson Method

Proof of correctness

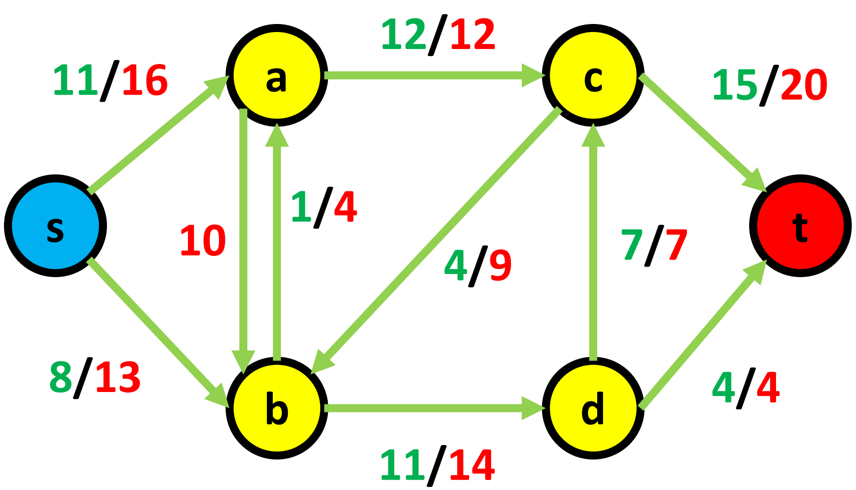
- Assume every capacity is integer
- Flow always increase by 1 at each iteration and we know flow is finite
- Therefore the algorithm do terminate!

- But does it terminate with the max flow?
  - That is why we need the min-cut max-flow theorem to finish our proof

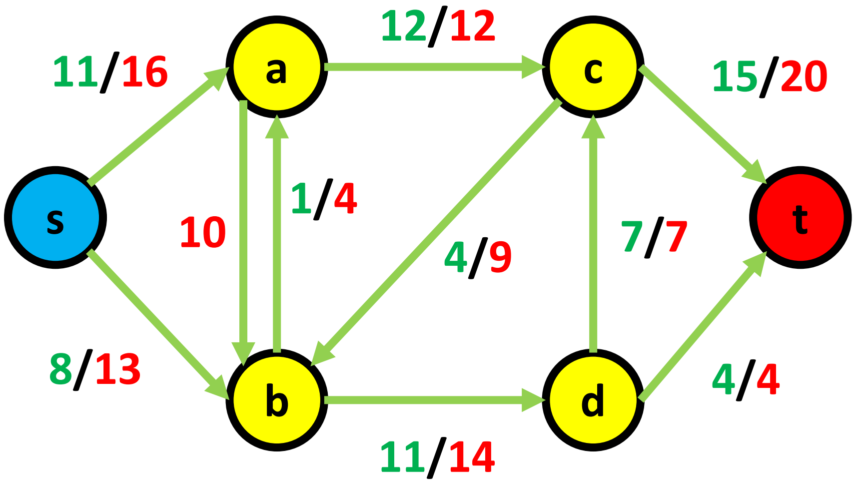
Questions?

- What is a cut?

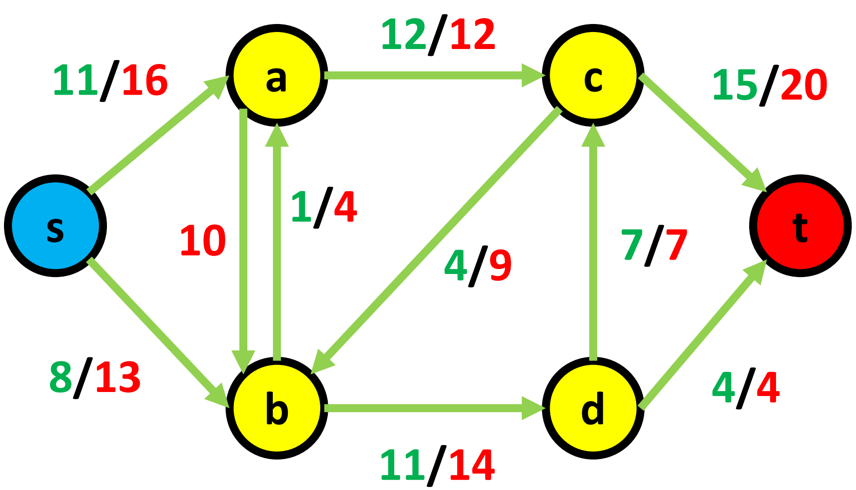
- What is a cut?



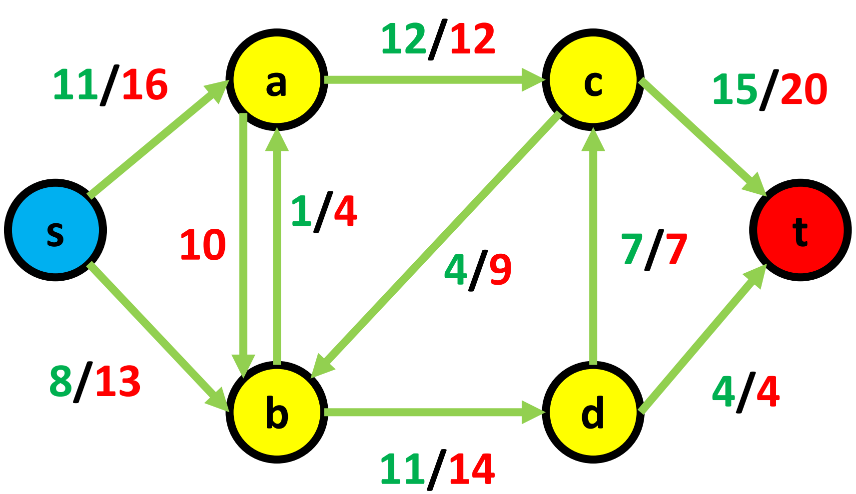
- What is a cut?
  - A cut (S,T)



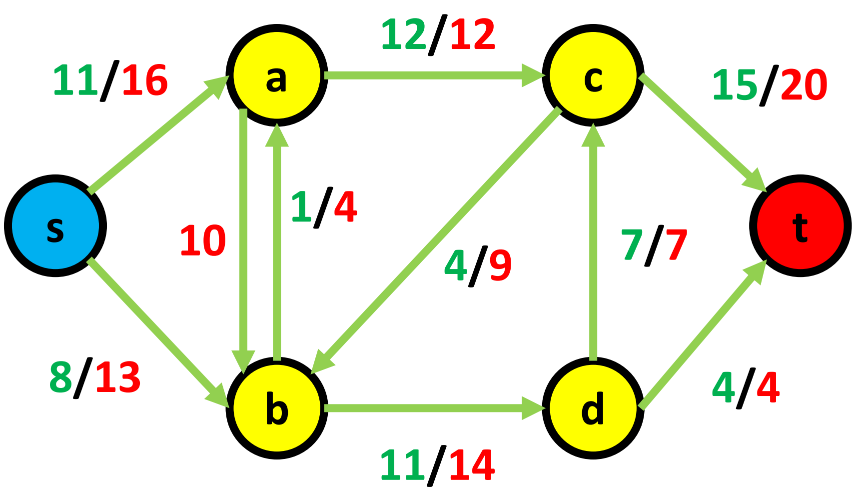
- What is a cut?
  - A cut (S,T)
    - S must contain vertex s
    - T must contain vertex t



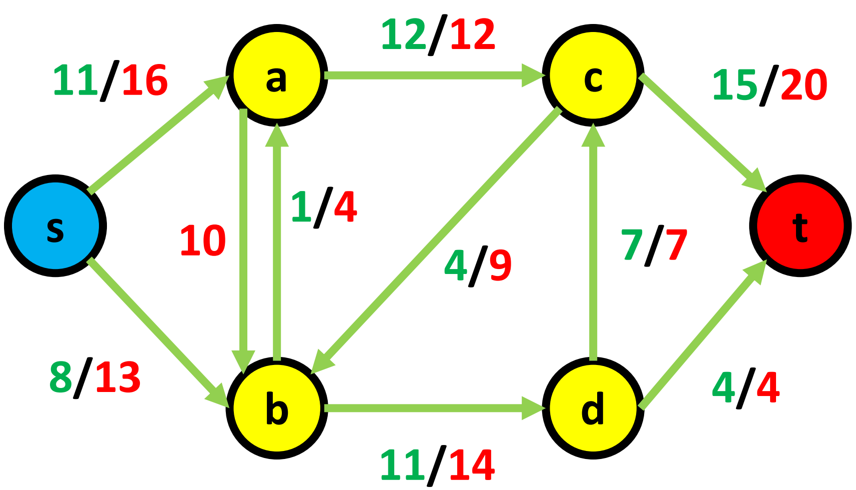
- What is a cut?
  - A cut (S,T)
    - S must contain vertex s
    - T must contain vertex t



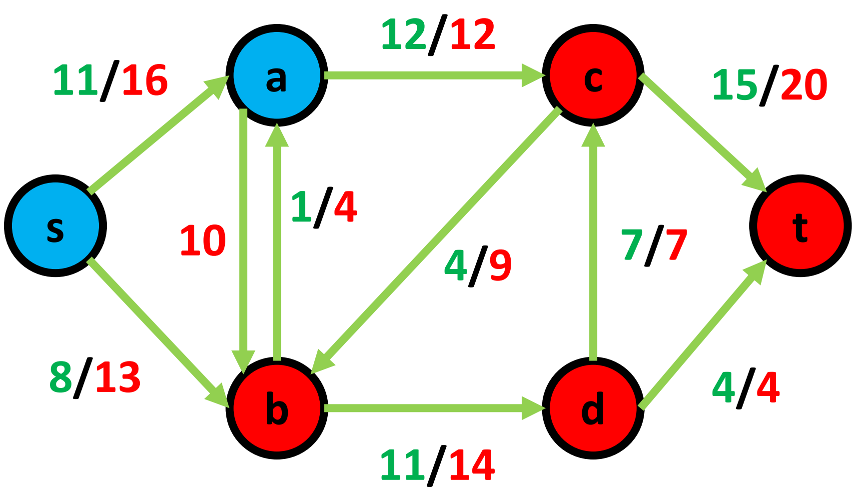
- What is a cut?
  - A cut (S,T)
    - S must contain vertex s
    - T must contain vertex t
  - Thus we can do the following…



- What is a cut?
  - A cut (S,T) cut into two parts: source in one part, destination in another
    - S must contain vertex s. S = (s,a)
    - T must contain vertex t. T = (b,c,d,t)
  - Thus we can do the following…

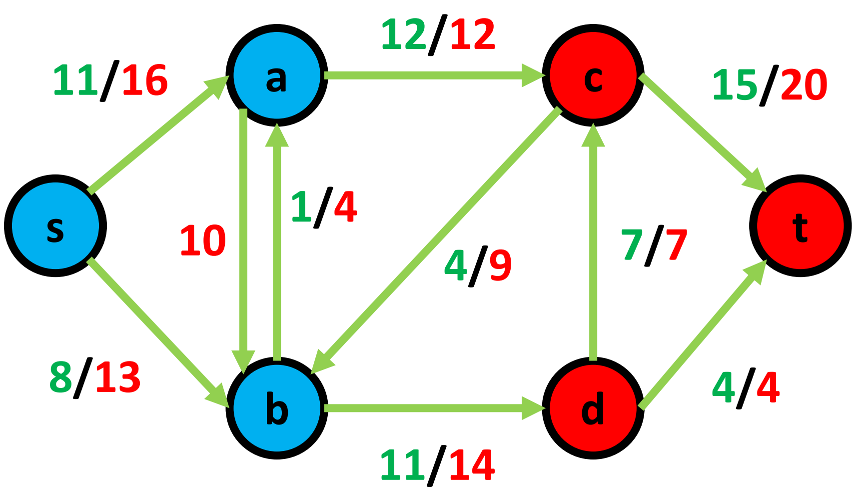


- What is a cut?
  - A cut (S,T)
    - S must contain vertex s. S = (s,a)
    - T must contain vertex t. T = (b,c,d,t)
  - Thus we can do the following…

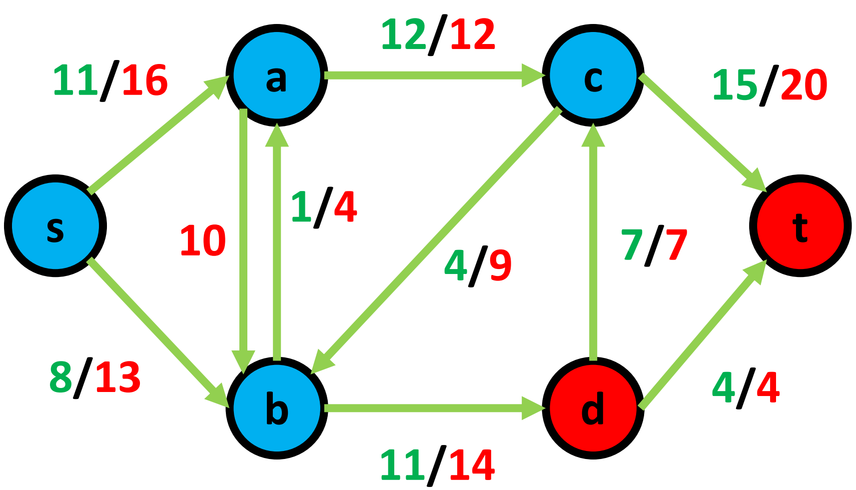


Cut
Flow and Capacity

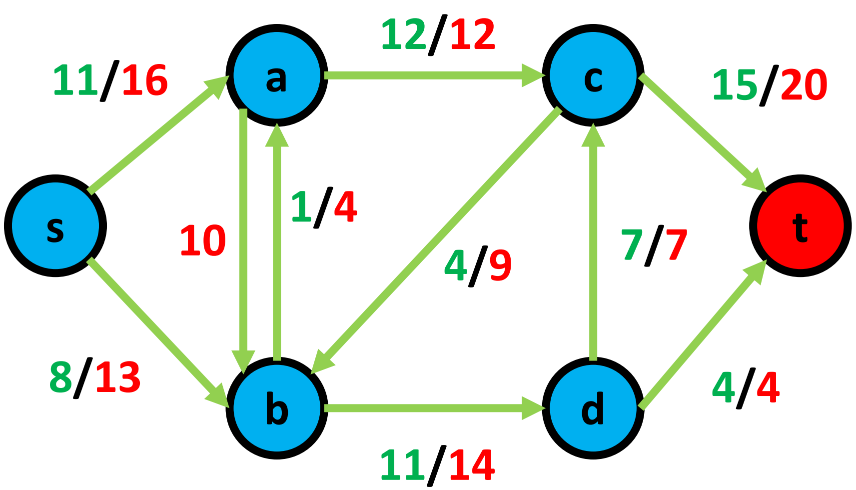
- What is a cut?
  - A cut (S,T) Min Cut = Max Flow
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - Thus we can do the following…



- What is a cut?
  - A cut (S,T)
    - a, b, c must connect to s (start)
    - S must contain vertex s. S = (s,a,b,c)
    - T must contain vertex t. T = (d,t) t must connect to t (sink)
  - Thus we can do the following…

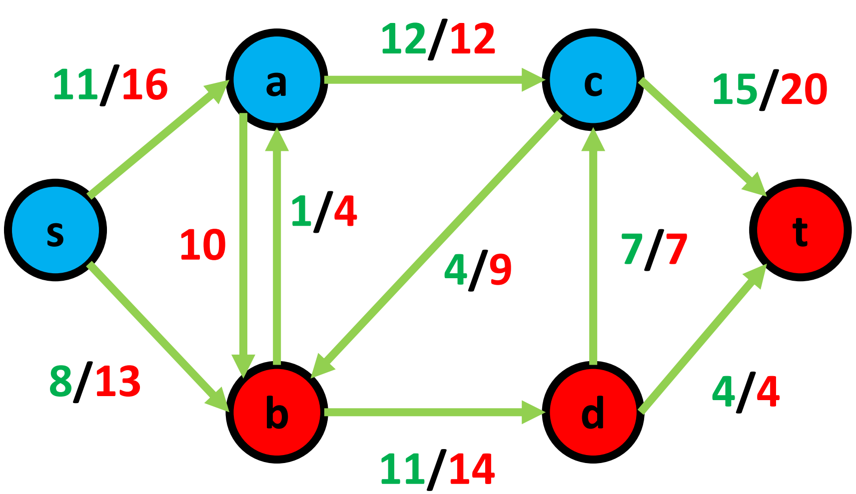


- What is a cut?
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b,c,d)
    - T must contain vertex t. T = (t)
  - Thus we can do the following…

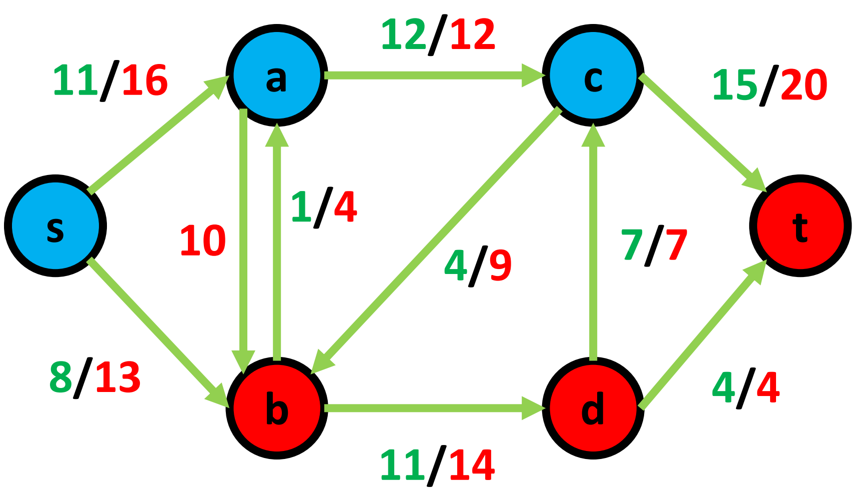


MONASH University

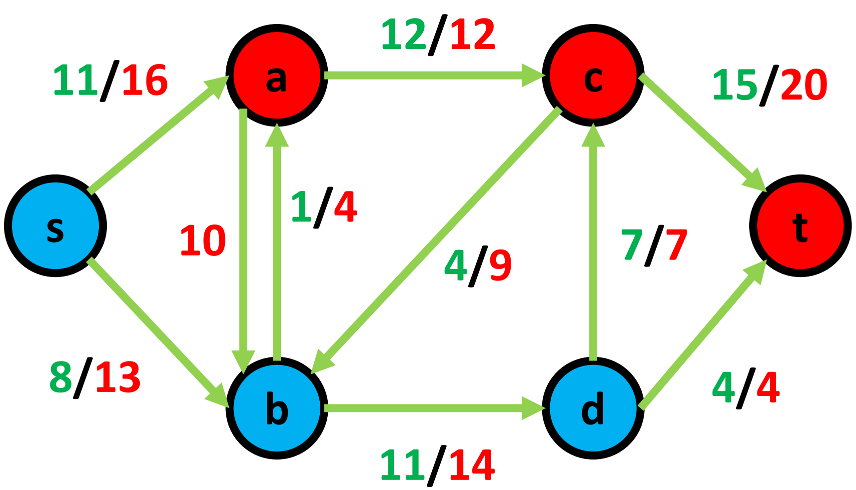
- What is a cut?
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,c)
    - T must contain vertex t. T = (b,d,t)
  - Thus we can do the following…
  - And more!!!



- What is a cut?
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,c) reachable from s to a, c
    - T must contain vertex t. T = (b,d,t) vertex reachable to t
  - Thus we can do the following…
  - And more!!! Because the S and T are still connected with path from s and path to t

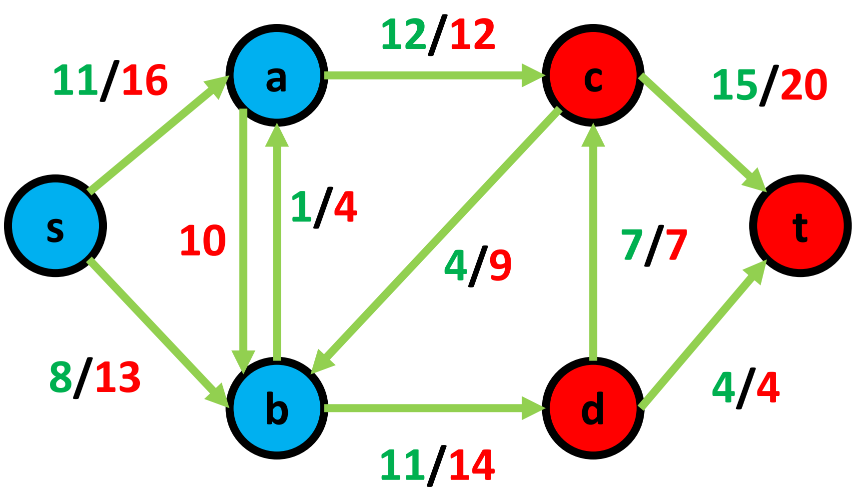


- What is a cut?
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,c)
    - T must contain vertex t. T = (b,d,t)
  - Thus we can do the following…
  - And more!!! Because the S and T are still connected with path from s and path to t

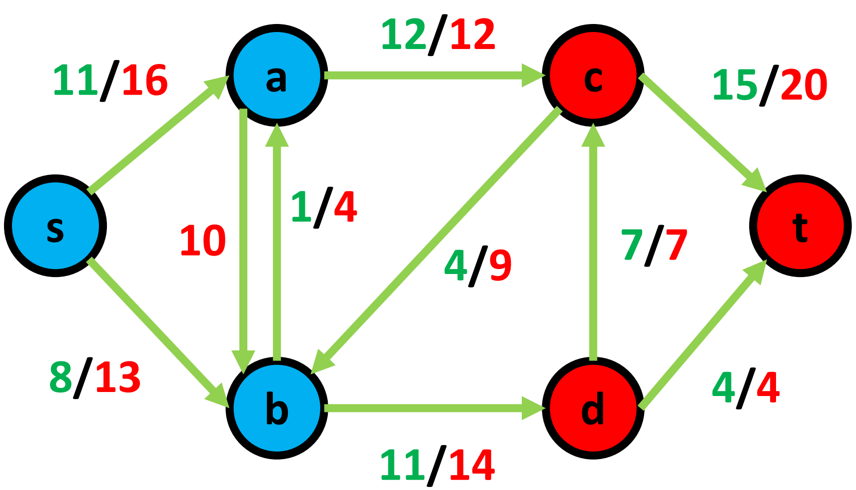


Questions?

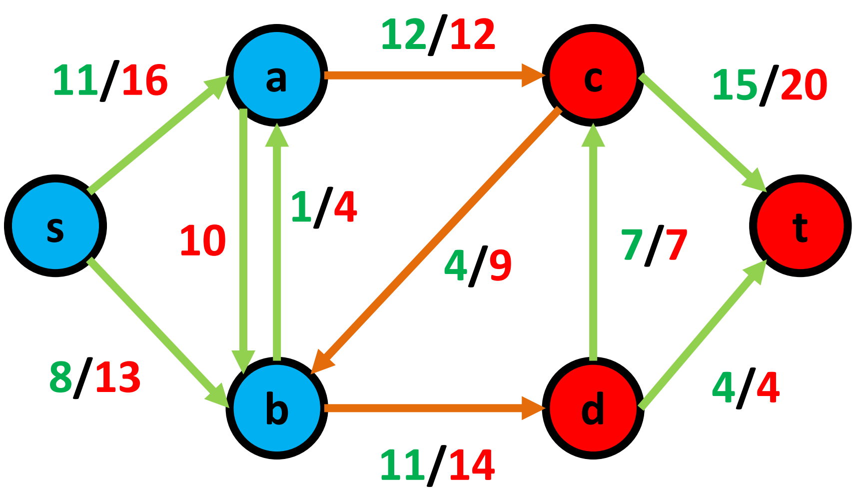
- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)



- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - We have edges crossing the cut

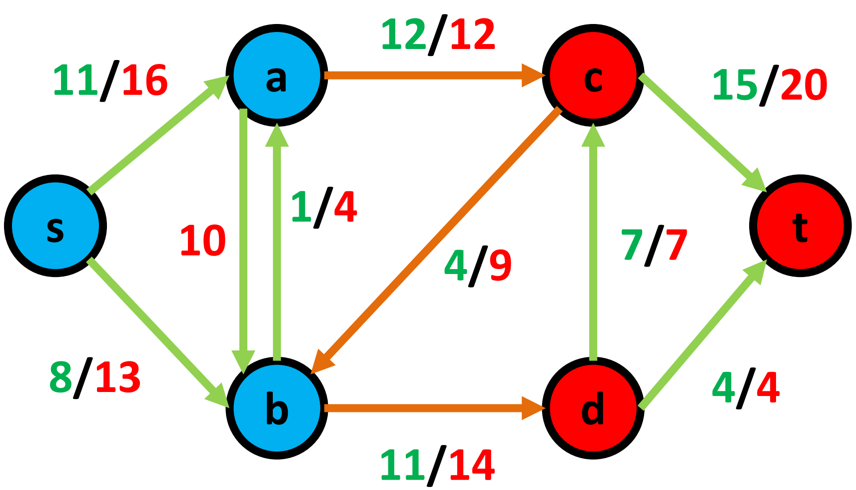


- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - We have edges crossing the cut

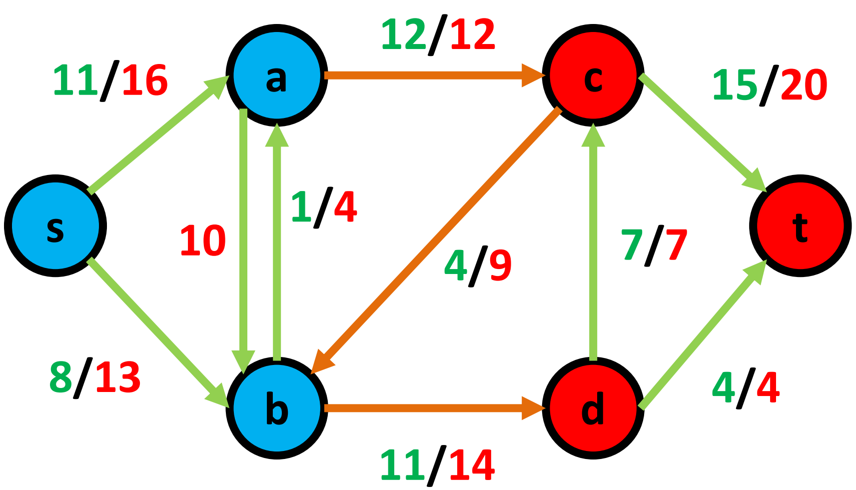


MONASH University

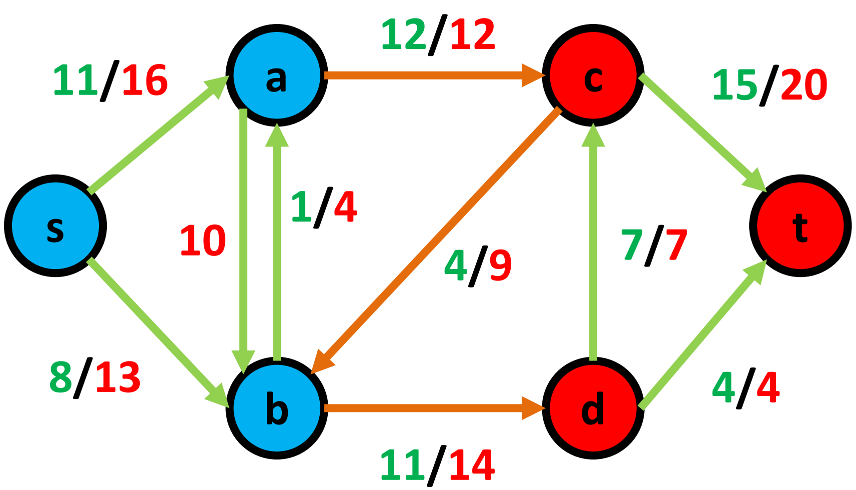
- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) =
  - Flow of a cut (S,T) =



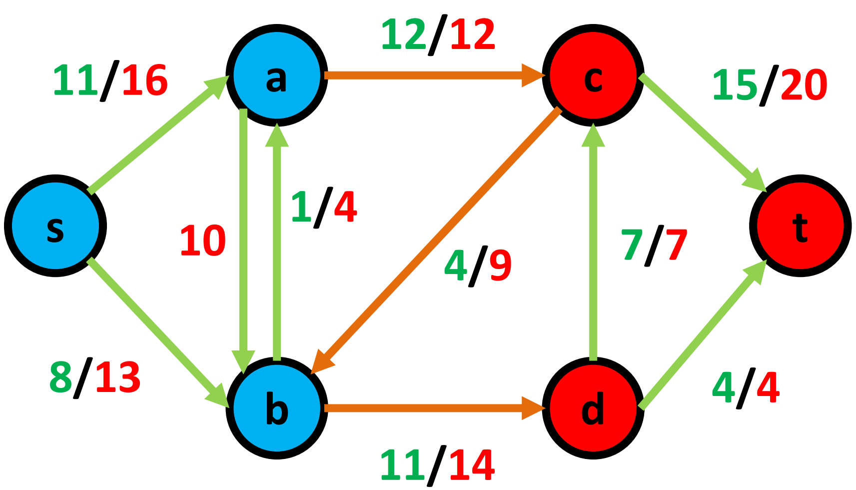
- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) = capacity of outgoing edges
  - Flow of a cut (S,T) =



- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) = capacity of outgoing edges = 12+14 = 26
  - Flow of a cut (S,T) =

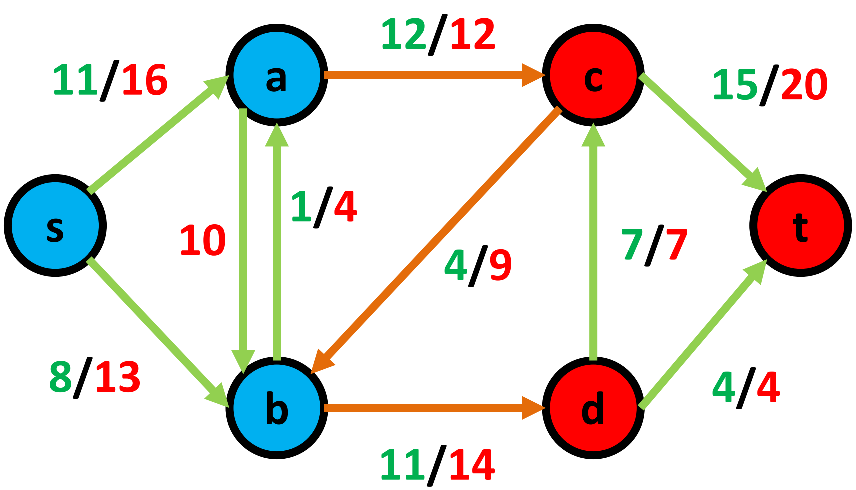


- Let us use this example
  - A cut (S,T)
    - S must contain vertex s. S = (s,a,b)
    - T must contain vertex t. T = (c,d,t)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) = capacity of outgoing edges = 12+14 = 26
  - Flow of a cut (S,T) = flow of outgoing edges – flow of incoming edges



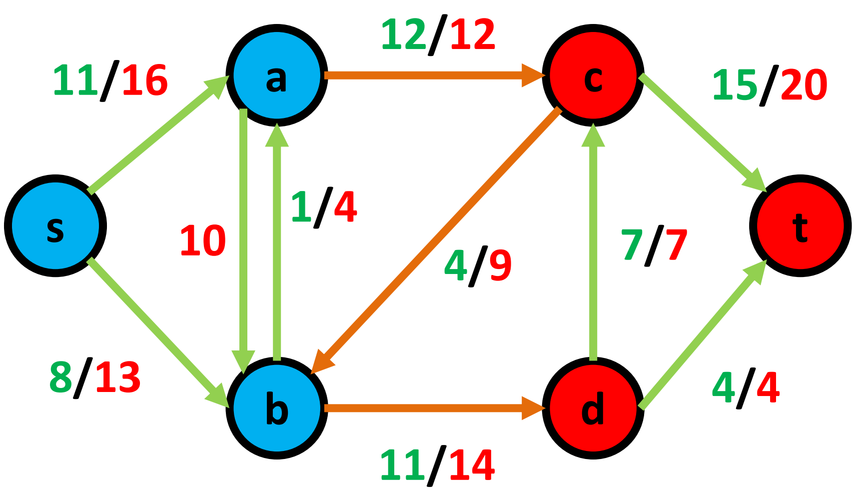
Cut
Flow and Capacity

- Let us use this example
  - A cut (S,T)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) = capacity of outgoing edges = 12+14 = 26
  - Flow of a cut (S,T) = flow of outgoing edges – flow of incoming edges

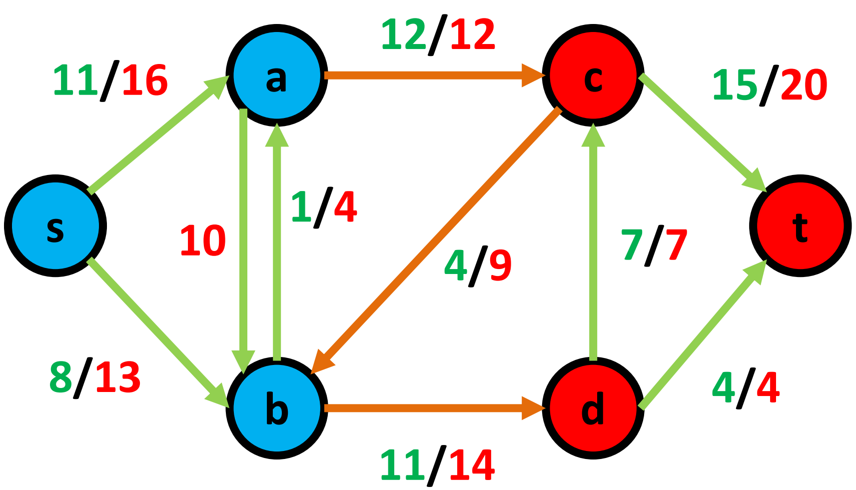


- Let us use this example
  - A cut (S,T)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) = capacity of outgoing edges = 12+14 = 26
  - Flow of a cut (S,T) = flow of outgoing edges – flow of incoming edges = 12 + 11 – 4

capacity = 26, flow = 19 = 19/24



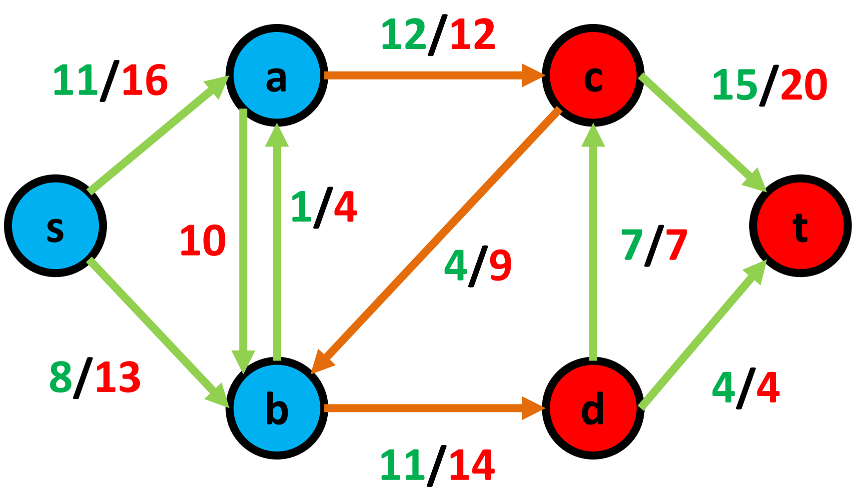
- Let us use this example
  - A cut (S,T)
  - We have edges crossing the cut
  - Capacity of a cut (S,T) = capacity of outgoing edges = 12+14 = 26
  - Flow of a cut (S,T) = flow of outgoing edges – flow of incoming edges
    = 12 + 11 – 4 = 19



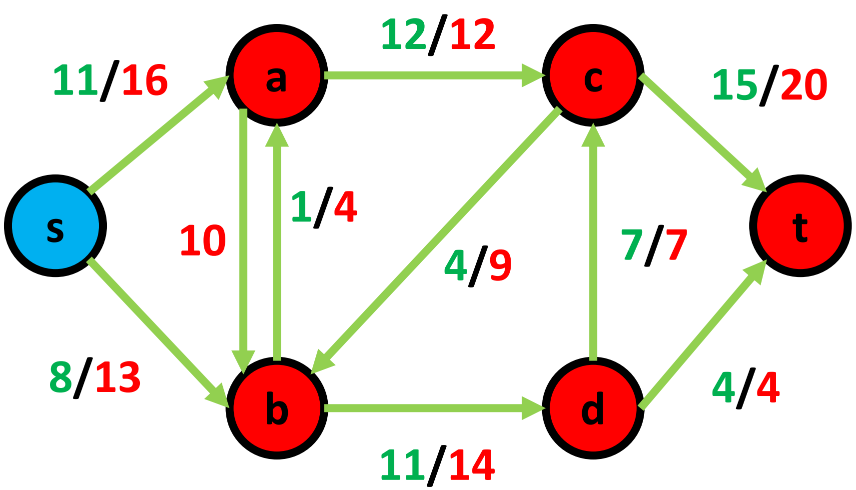
Questions?

- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19

potential 7 outgoing flow -> another route



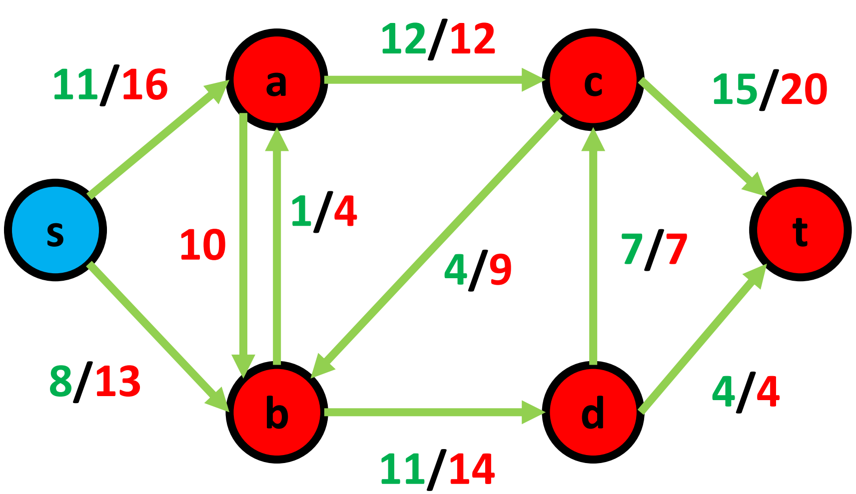
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?



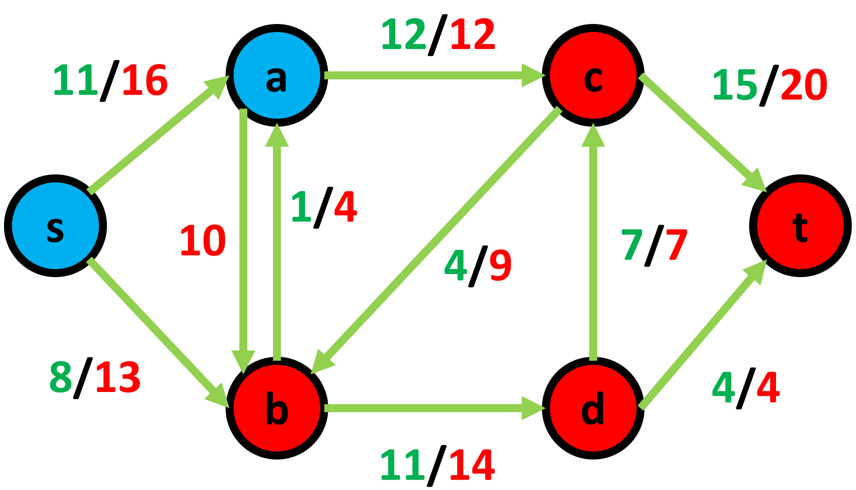
Cut

Flow and Capacity

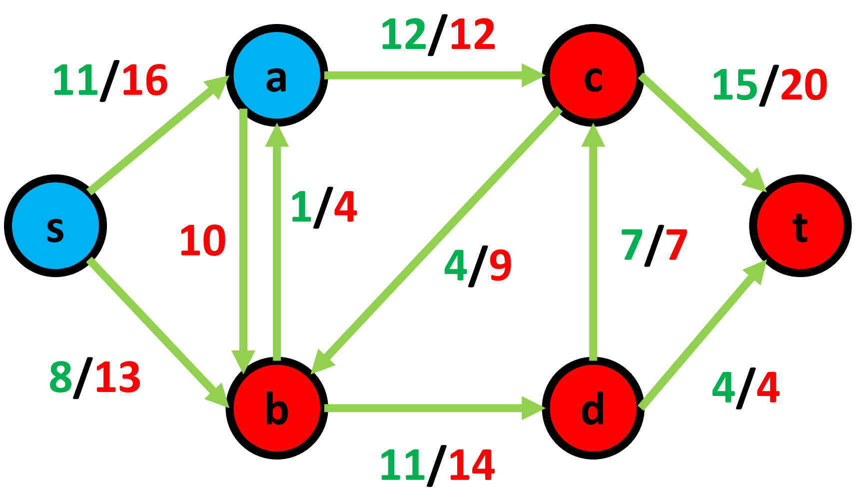
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 29
    - Flow = 19



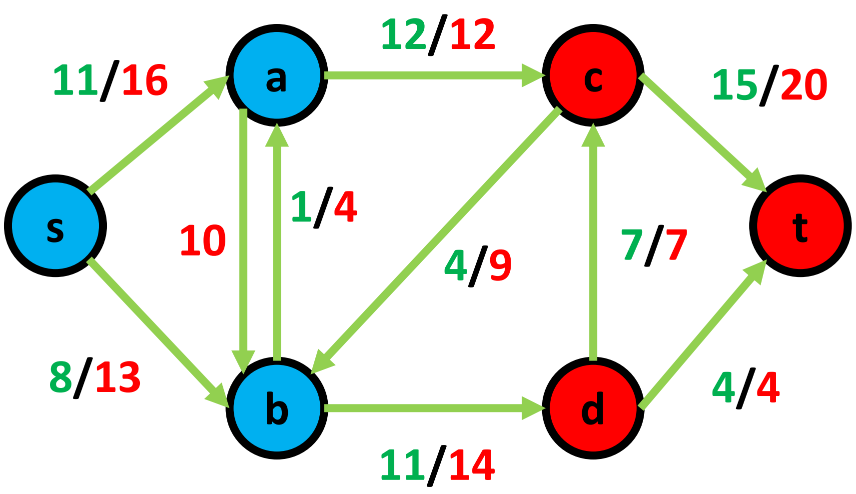
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = ?
    - Flow = ?



- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 35
    - Flow = 19



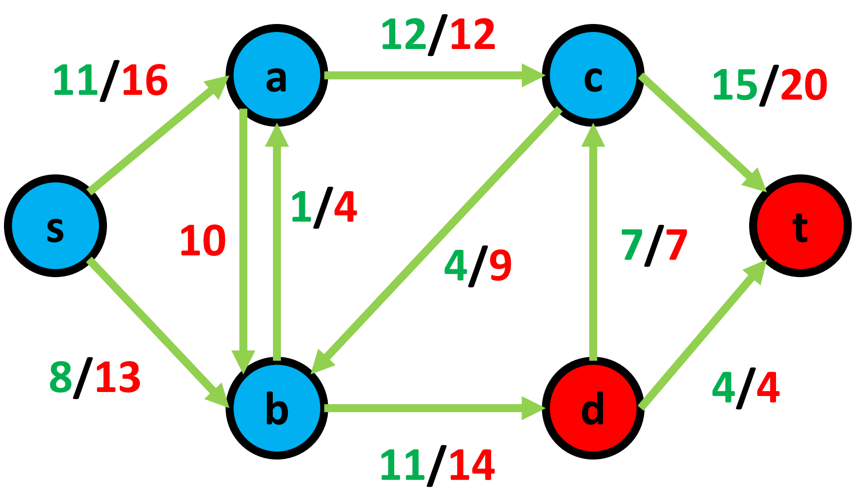
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 26
    - Flow = 19



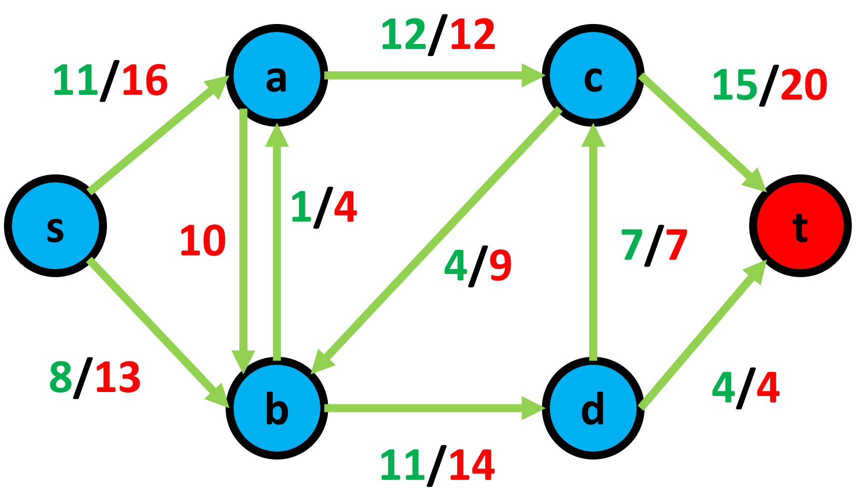
Cut

Flow and Capacity

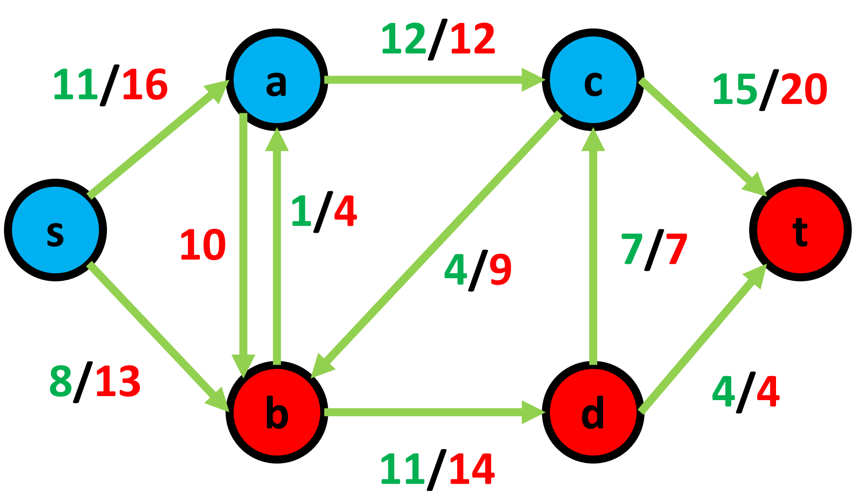
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 34
    - Flow = 19



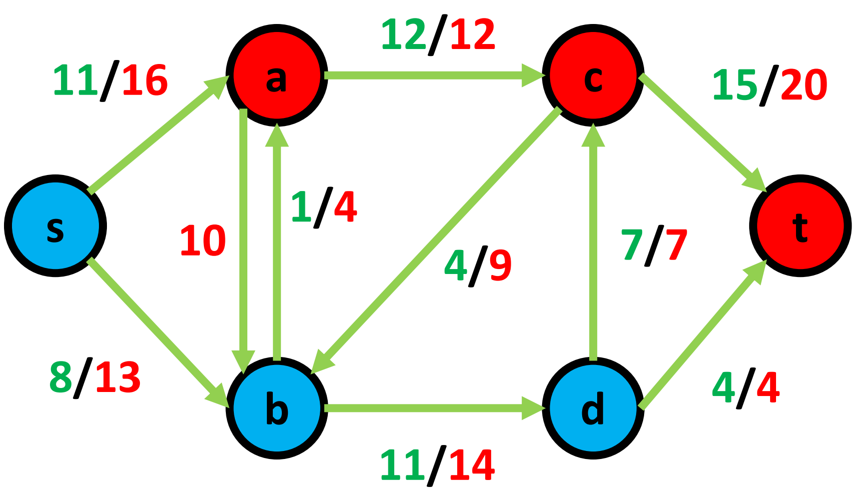
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 24
    - Flow = 19



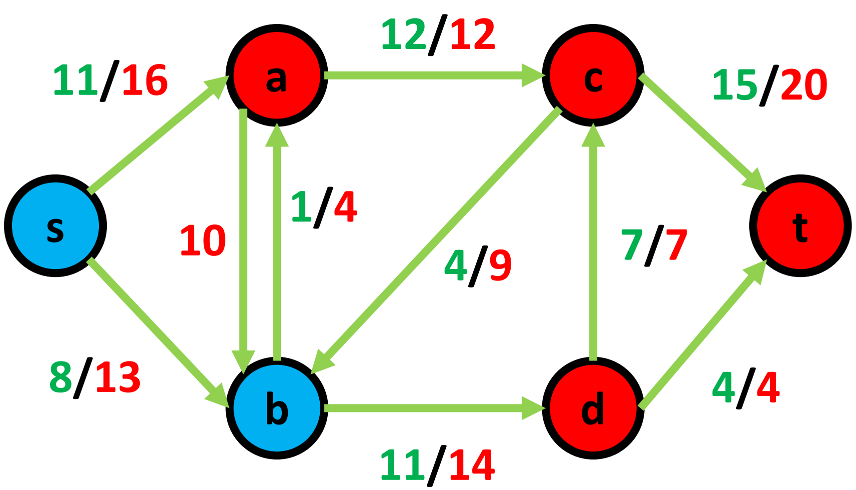
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 52
    - Flow = 19



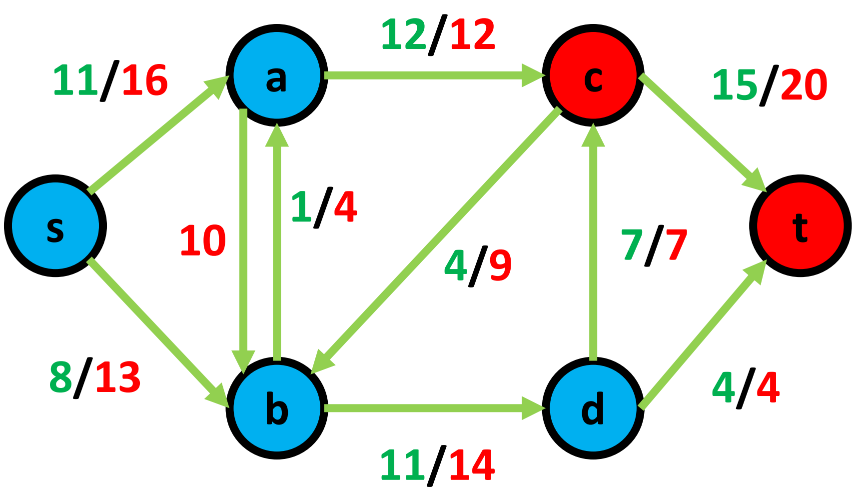
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 31
    - Flow = 19



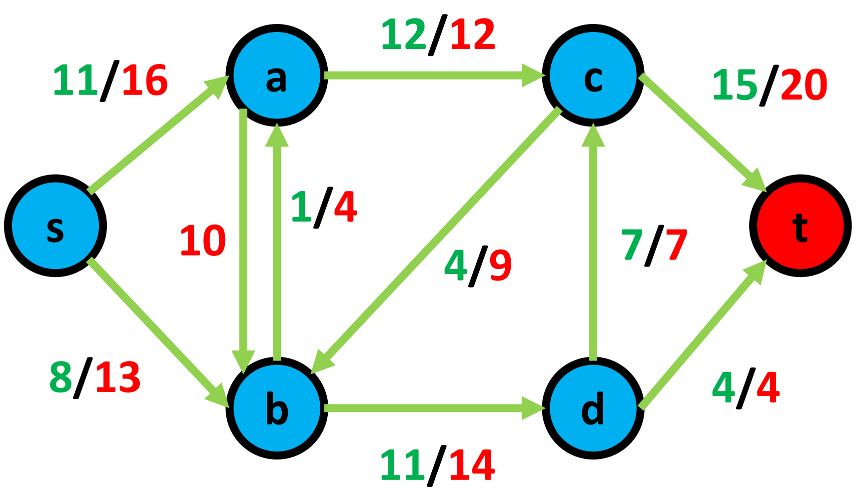
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 34
    - Flow = 19



- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity = 23
    - Flow = 19

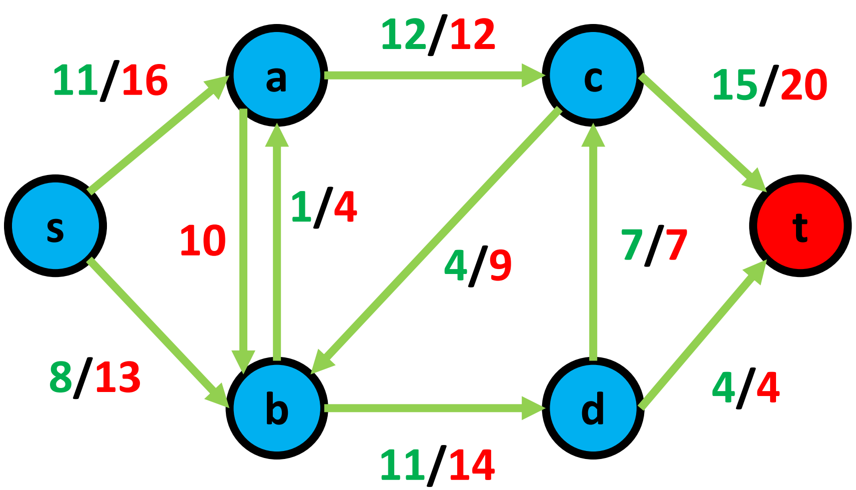


- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity differs
    - Flow the same

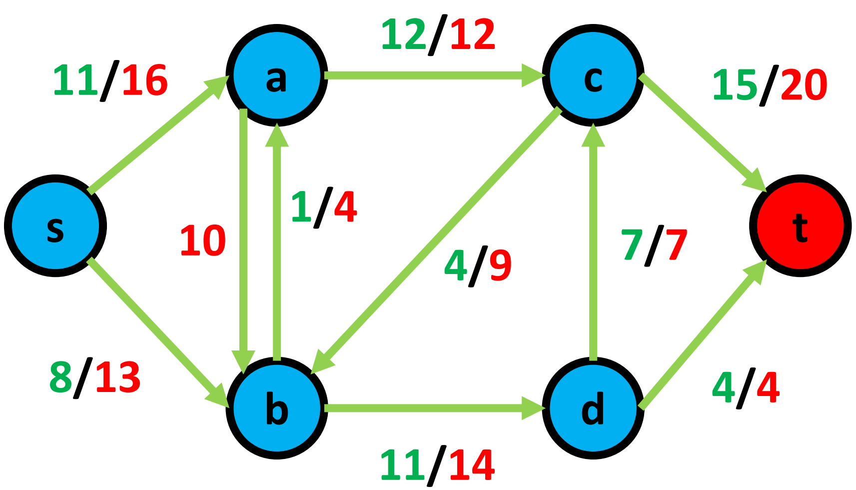


Cut
Flow and Capacity

- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity differs
    - Flow the same
    - But we know flow <= capacity



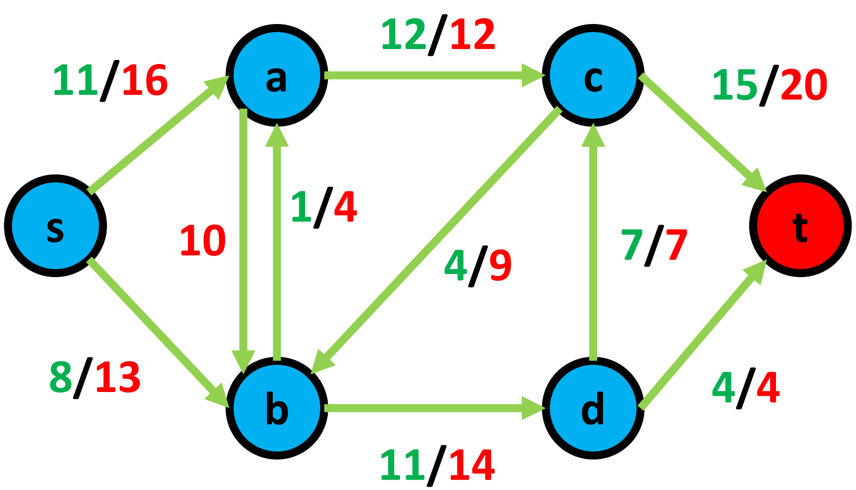
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting?
    - Capacity differs
    - Flow the same
    - But we know flow <= capacity
    - Thus smallest capacity is the upper bound of flow?



Cut

Flow and Capacity

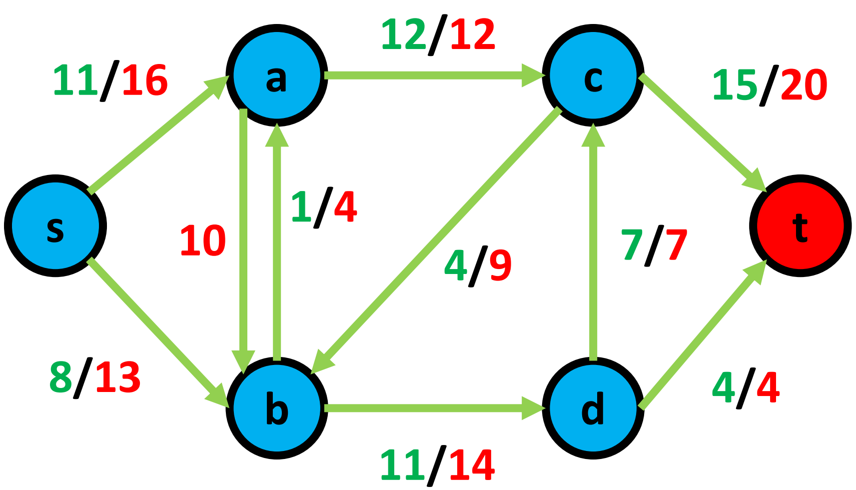
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting? Flow of every cut = flow of network
    - Capacity differs
    - Flow the same
    - But we know flow <= capacity
    - Thus smallest capacity is the upper bound of flow?



Cut
Flow and Capacity

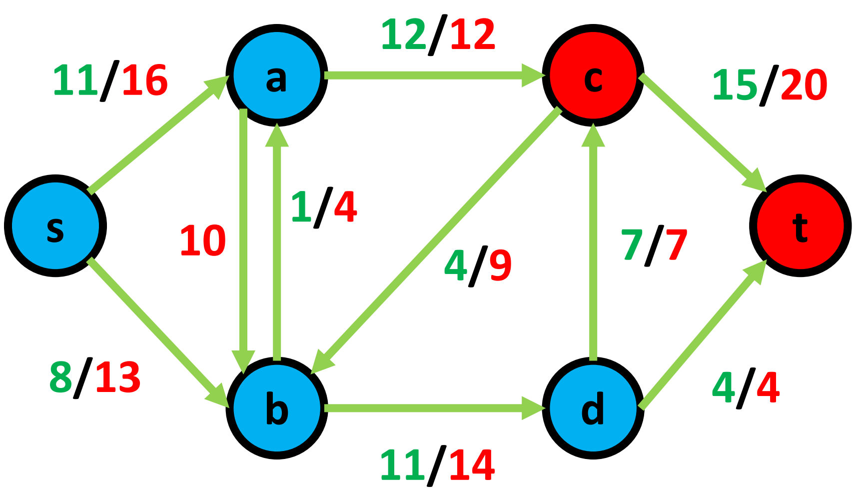
MONASH University

- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting? Flow of every cut = flow of network
    - Capacity differs, smallest is 23!
    - Flow the same
    - But we know flow <= capacity
    - Thus smallest capacity is the upper bound of flow?



Cut
Flow and Capacity

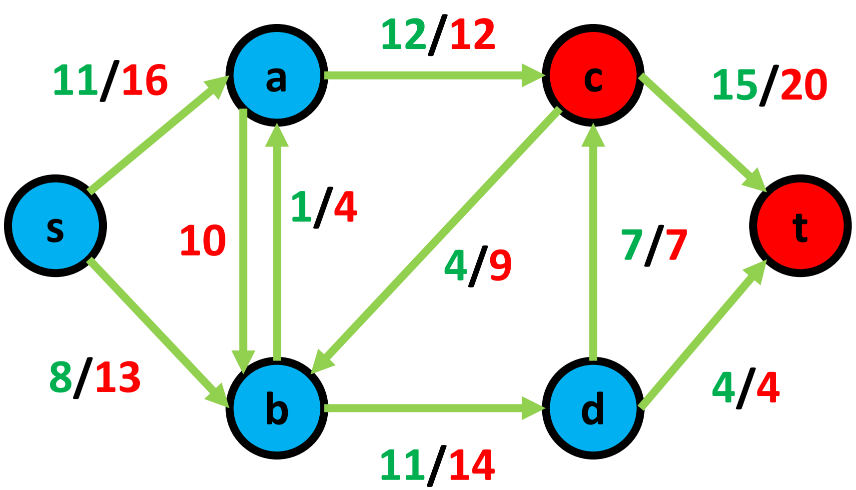
- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting? Flow of every cut = flow of network
    - Capacity differs, smallest is 23!
    - Flow the same
    - But we know flow <= capacity
    - Thus smallest capacity is the upper bound of flow?



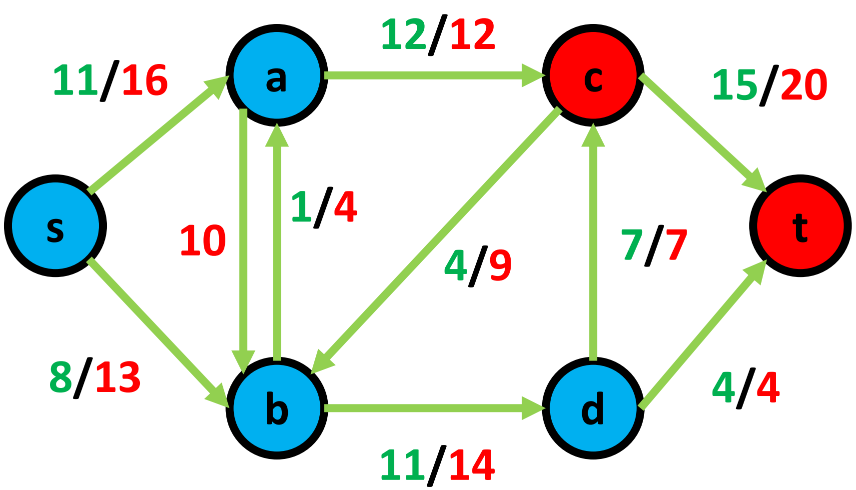
Cut

Flow and Capacity

- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting? Flow of every cut = flow of network
    - Capacity differs, smallest is 23!
    - Flow the same, so max flow is 23!
    - But we know flow <= capacity
    - Thus smallest capacity is the upper bound of flow?



- Now we realize something….
  - Remember our earlier cut have capacity of 26 and flow of 19
  - What about other way of cutting? Flow of every cut = flow of network
    - Capacity differs, smallest is 23!
    - Flow the same, so max flow is 23! And we saw earlier when we run Ford-Fulkerson that the maximum flow is 23!
    - But we know flow <= capacity

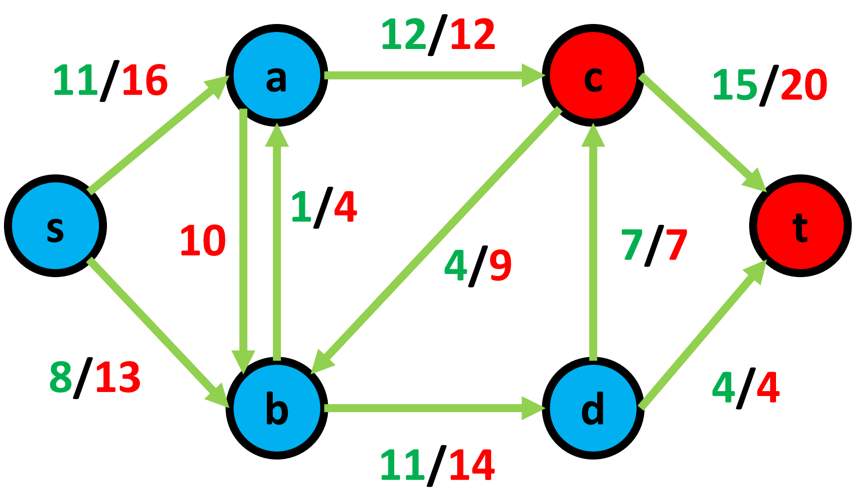


Questions?

Min-Cut Max-Flow Theorem

The key to this…

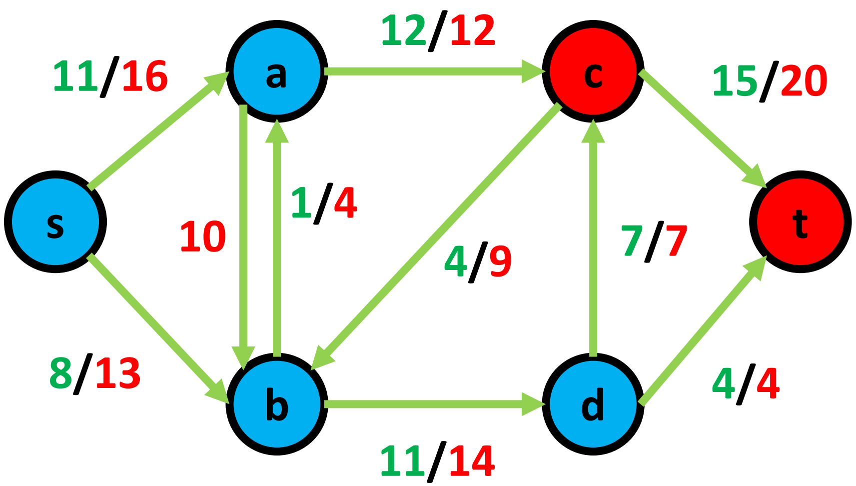
- Minimum capacity of all cut is 23
- Flow of cut <= capacity of cut
- Flow of a cut == flow of a network



Min-Cut Max-Flow Theorem

The key to this…

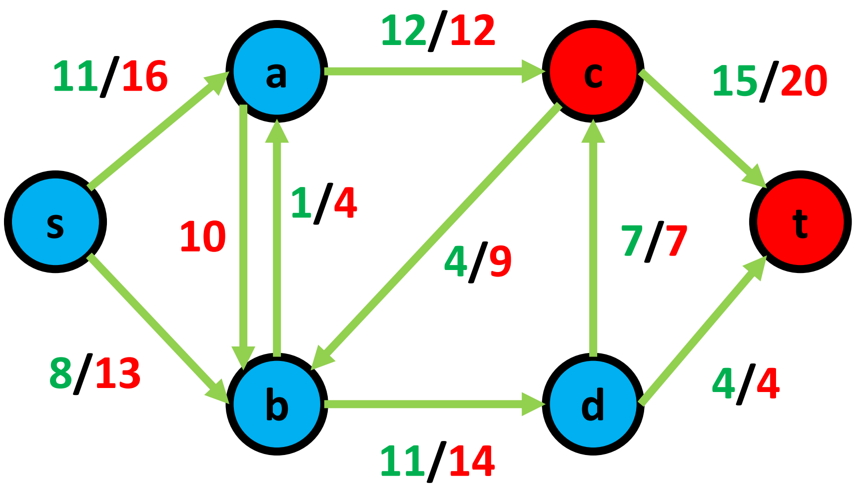
- Minimum capacity of all cut is 23
- Flow of cut <= capacity of cut
- Flow of a cut == flow of a network
- Therefore, capacity of a min-cut = max-flow of a network



Min-Cut Max-Flow Theorem

The key to this…

- Minimum capacity of all cut is 23
- Flow of cut <= capacity of cut
- Flow of a cut == flow of a network
- Therefore, capacity of a min-cut = max-flow of a network



Min-Cut Max-Flow Theorem

The key to this…

- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:

Min-Cut Max-Flow Theorem

The key to this…

- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge

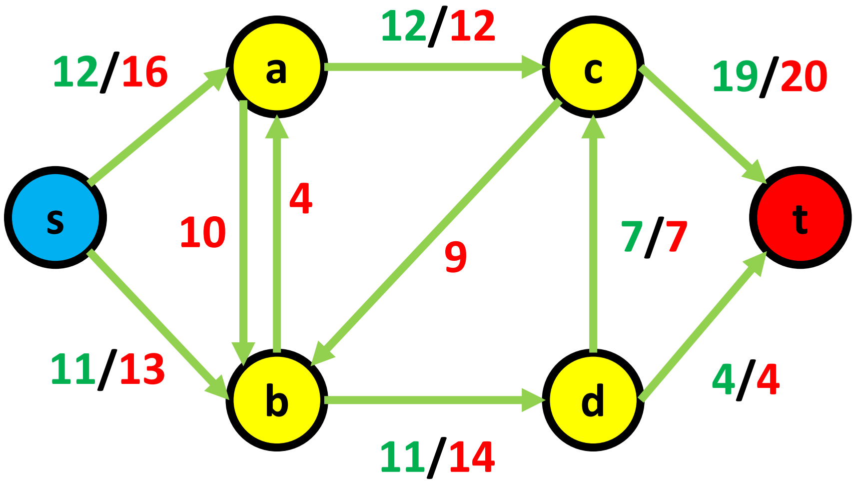
Min-Cut Max-Flow Theorem
The key to this…

- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero

Min-Cut Max-Flow Theorem

The key to this…

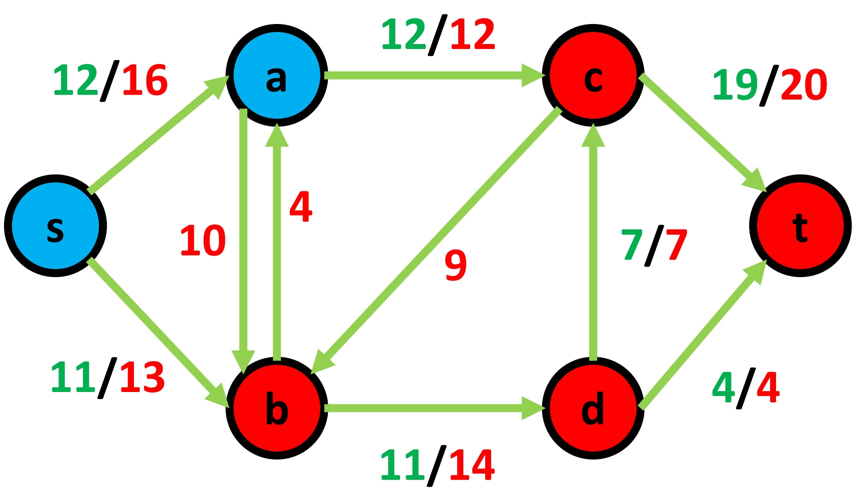
- Therefore, capacity of a <u>min-cut</u> = <u>max-flow</u> of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero



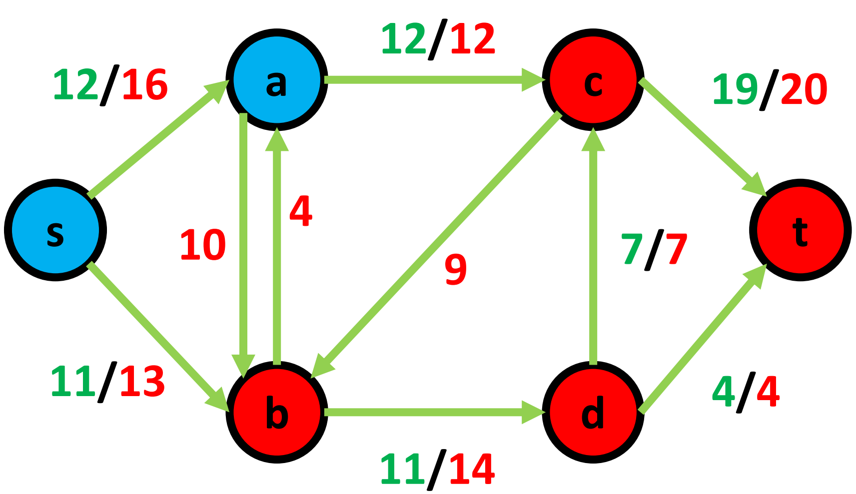
Min-Cut Max-Flow Theorem

The key to this…

- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
  - This meets the requirement above right?



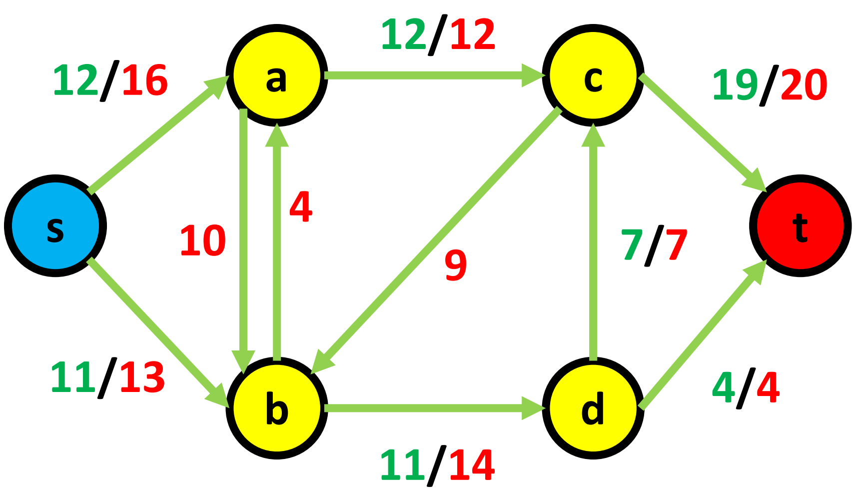
- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
  - This meets the requirement above right? NO!



Min-Cut Max-Flow Theorem

The key to this…

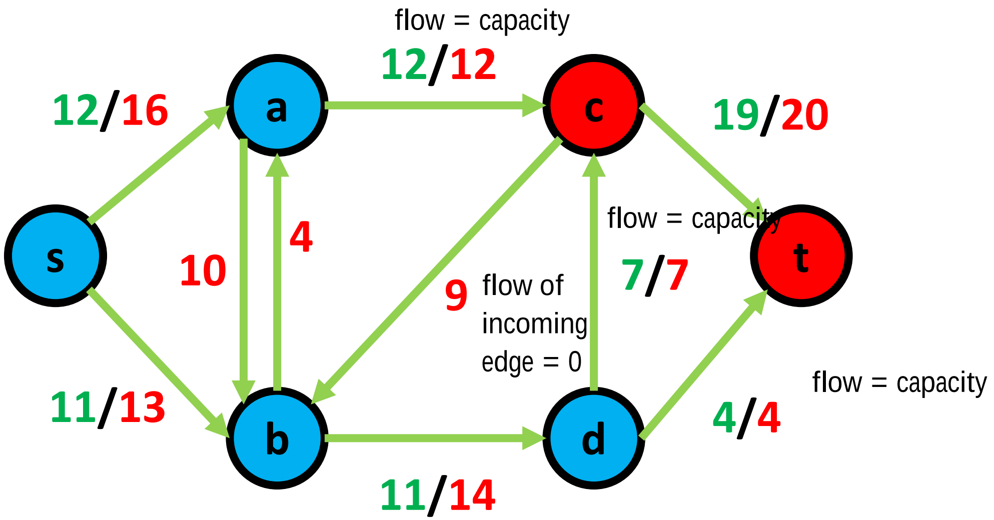
- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
  - This meets the requirement above right? Which one meet the requirement above?



Min-Cut Max-Flow Theorem

The key to this…

- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
  - This meets the requirement above right? Which one meet the requirement above? This one!

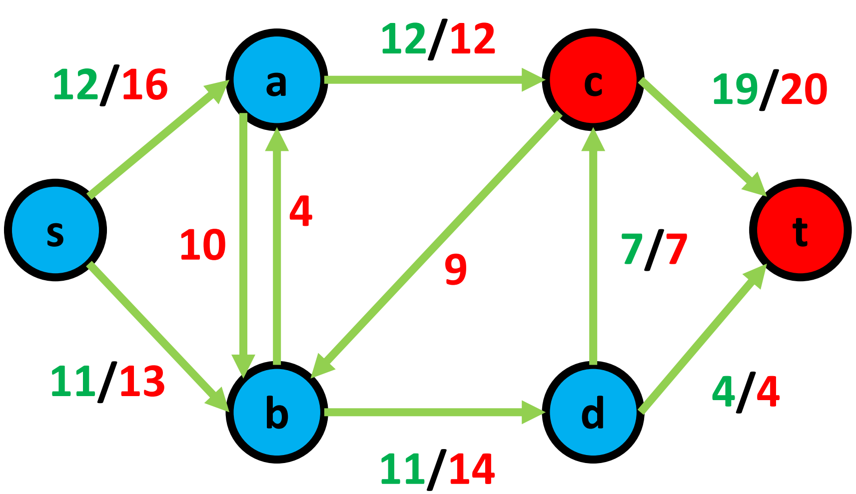


Min-Cut Max-Flow Theorem

The key to this…

MONASH University

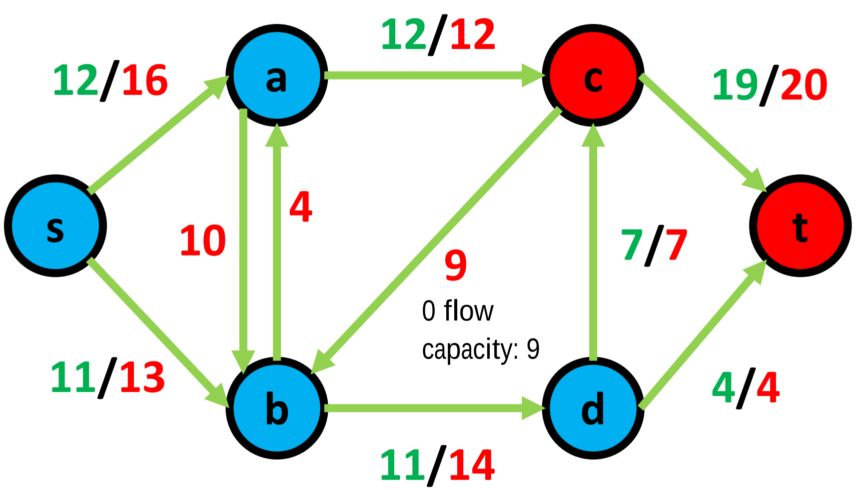
- Therefore, capacity of a <u>min-cut</u> = <u>max-flow</u> of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
  - This meets the requirement above right? Which one meet the requirement above? This one! This is called the min-cut



Min-Cut Max-Flow Theorem

The key to this…

- Therefore, capacity of a <u>min-cut</u> = <u>max-flow</u> of a network
- Ford-Fulkerson terminates when there is a cut:
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
  - This meets the requirement above right? Which one meet the requirement above? This one! This is called the min-cut



<u>Min-cut</u>

S = (s, a, b, d)

T = (c, t)

cross of cut: still have capacity and has incoming flow to return

due to Ford-Fulkerson find cut that flow on each outgoing = capacity flow on incoming edge = 0, so no longer have forward edge (outgoing edge in residual network and no incoming flow to return (purple color)

short cut

cut(S,T) set of vertices reachable by S

reachable: has a edge directly to that vertex

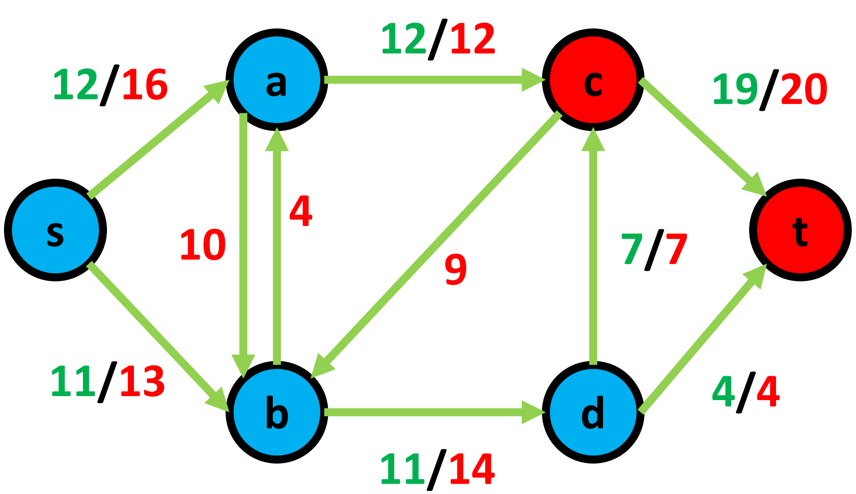
everything else falls on T

Min-Cut Max-Flow Theorem
The key to this…

from residual network, start can reach a, b, d, rest to destination, so cut between them -> Mincut

find min-cut lead to max-flow

- Therefore, capacity of a min-cut = max-flow of a network
- Ford-Fulkerson terminates when there is a cut:
  no forward edge
  - Flow on each outgoing edge cut is equal to the capacity of the edge
  - Flow on each incoming edge to the cut is zero
    no flow to be undone
    so no backward edge to return to sink
  - This meets the requirement above right? Which one meet the requirement above? This one! This is called the min-cut



Go through MUA's slides where his final network differs but still max flow of 23. Can you find such a cut?

Questions?

- We know the following…

Optimization

Finding max-flow quicker…

- We know the following…
  – Complexity is O(FE) where F is the maximum flow

- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F

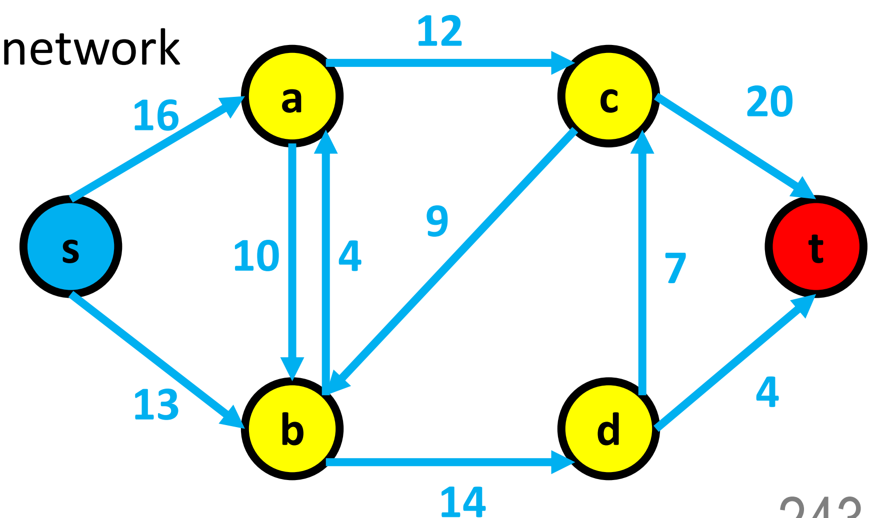
- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F

- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F

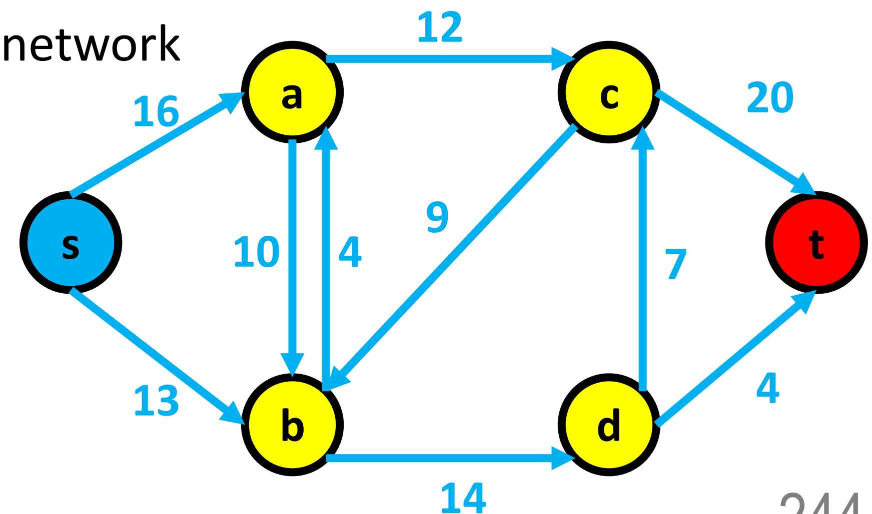
Optimization
Finding max-flow quicker…

- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!

- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!
  - But this depends on our residual network

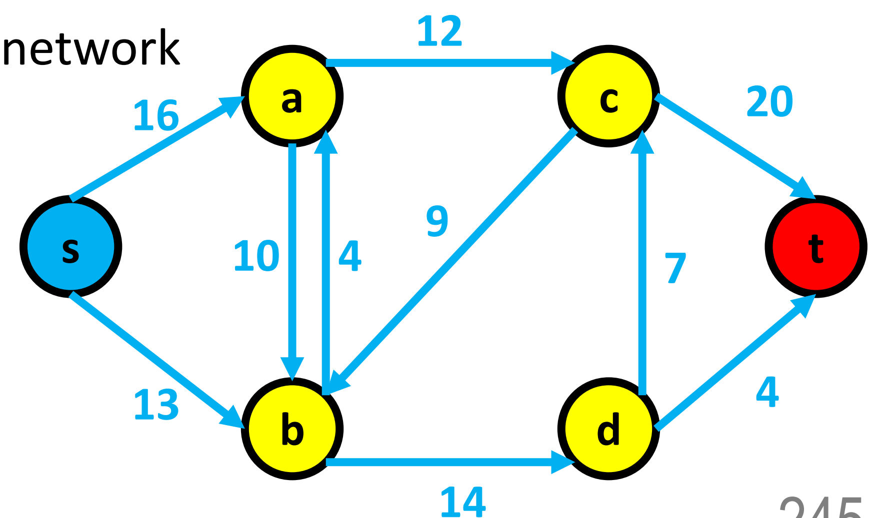


- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!
  - But this depends on our residual network
    - We used BFS/ DFS
    - Can we choose a better path?

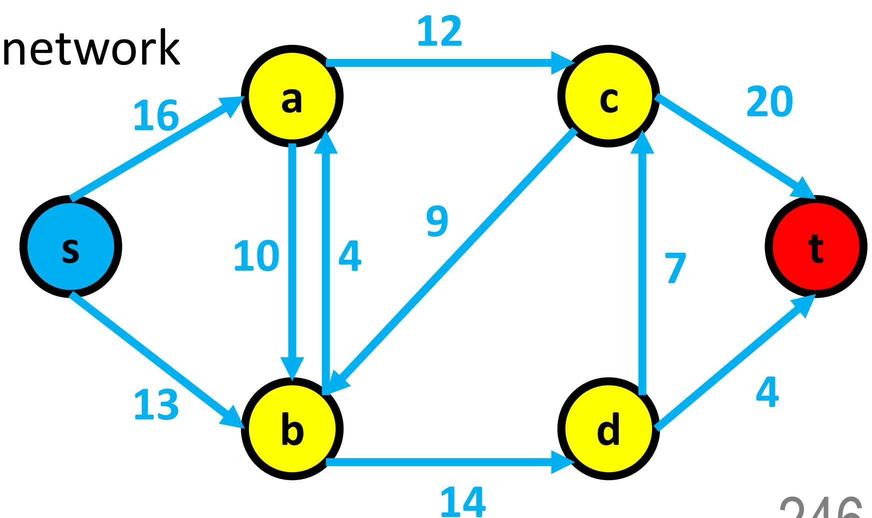


Optimization
Finding max-flow quicker…

- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!
  - But this depends on our residual network
    - We used BFS/ DFS
    - Can we choose a better path?
      - s -> b -> d -> t?

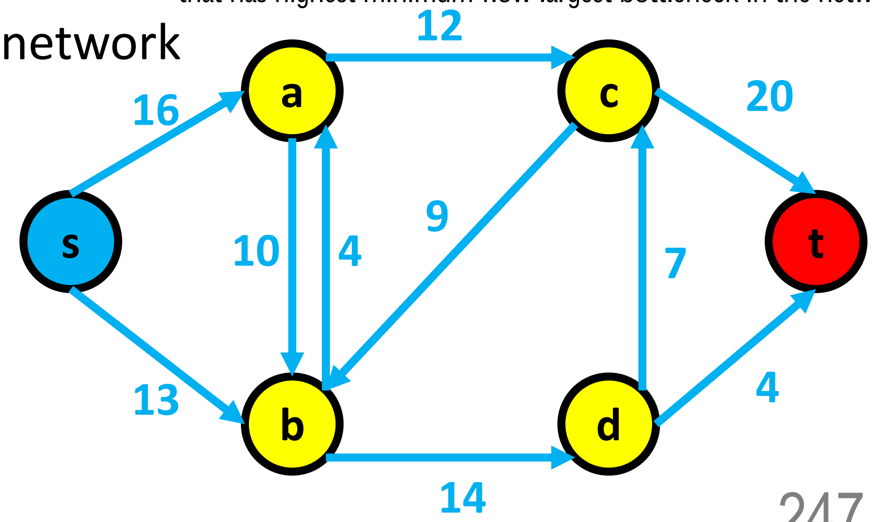


- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!
  - But this depends on our residual network
    - We used BFS/ DFS
    - Can we choose a better path?
      - s -> b -> d -> t?
      - s -> a -> c -> t?

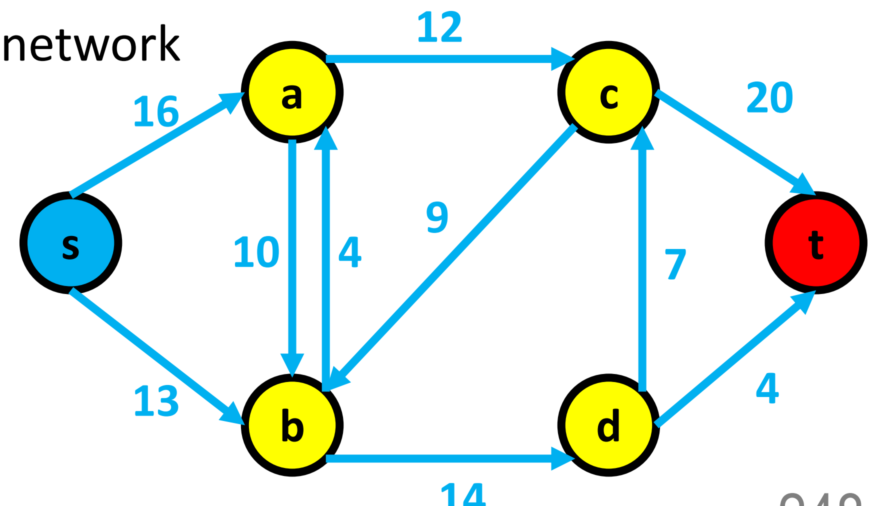


- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!
  - But this depends on our residual network
    - We used BFS/ DFS
    - Can we choose a better path?
      - s -> b -> d -> t?
      - s -> a -> c -> t?
      - s -> a -> b -> d -> t?

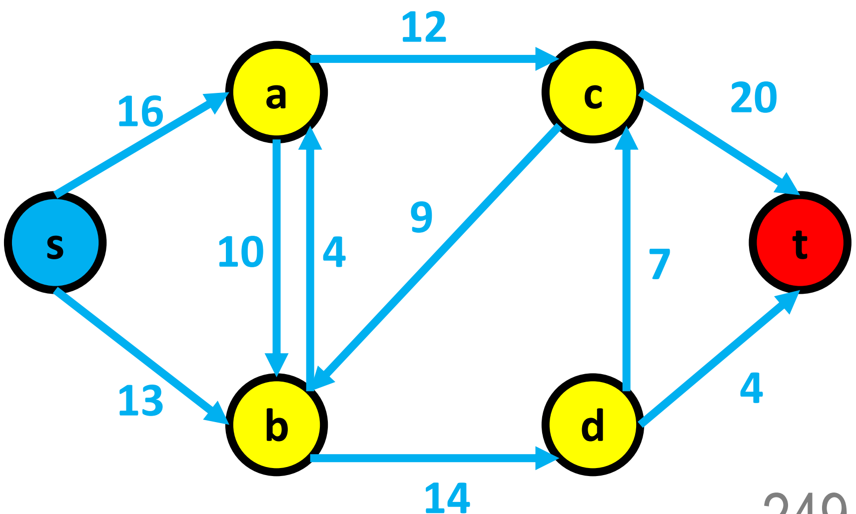
go through the possible route
that has highest minimum flow largest bottleneck in the network



- We know the following…
  - Complexity is O(FE) where F is the maximum flow
  - Each iteration would increase our flow, between 1…F until total flow is F
  - Which is faster?
    - 1,2,3,4,…,F
    - 8,13,41,…,F
    - Because we scale quicker!
  - But this depends on our residual network
    - We used BFS/ DFS
    - Can we choose a better path?
      - s -> b -> d -> t?
      - s -> a -> c -> t?
      - s -> a -> b -> d -> t?



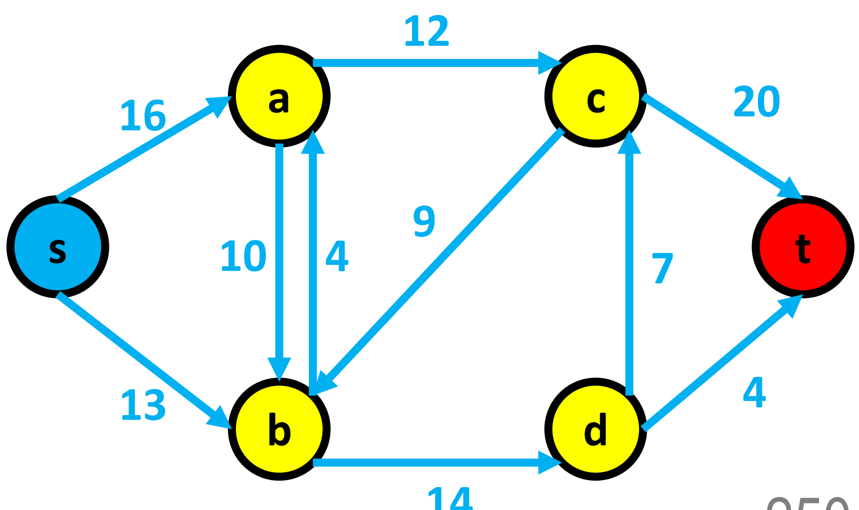
- Consider the following:



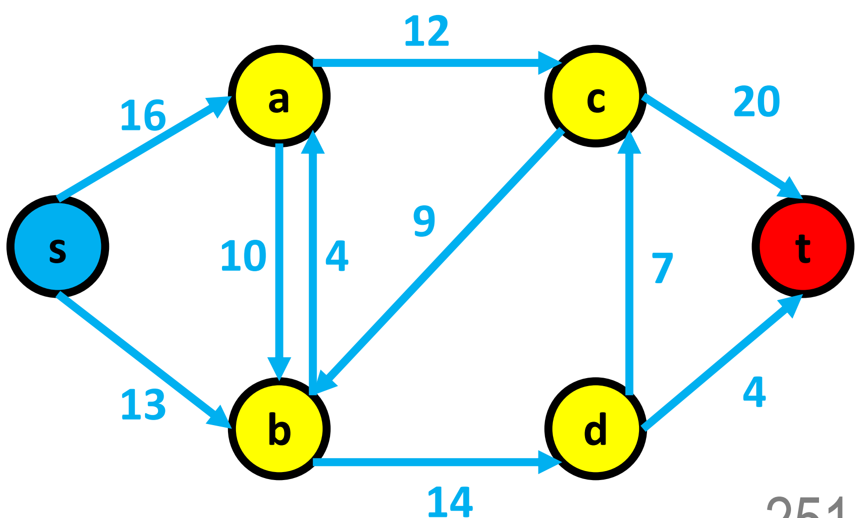
Edmond-Karp

Finding max-flow quicker…

- Consider the following:
  - Dinic/ Edmonds-Karp = choose path with fewest edge
  - Edmonds-Karp = choose path with largest bottle neck



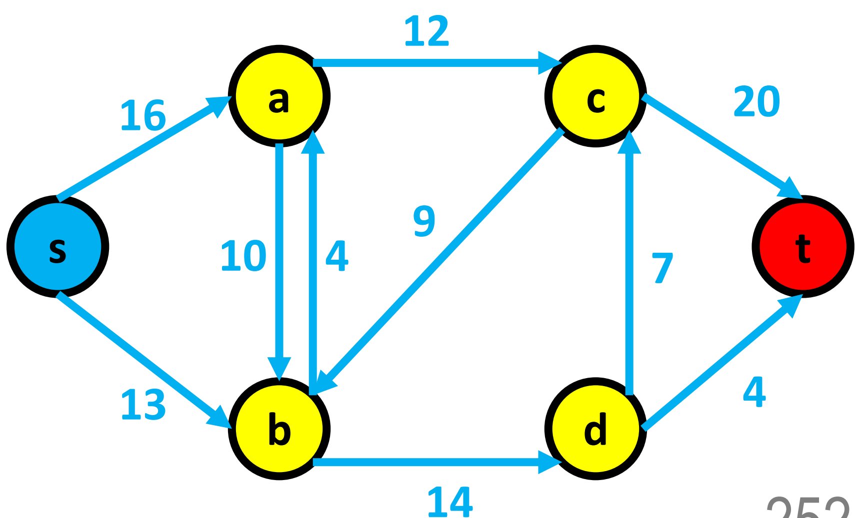
- Consider the following:
  - Dinic/ Edmonds-Karp = choose path with fewest edge
  - Edmonds-Karp = choose path with largest bottle neck
  - How?



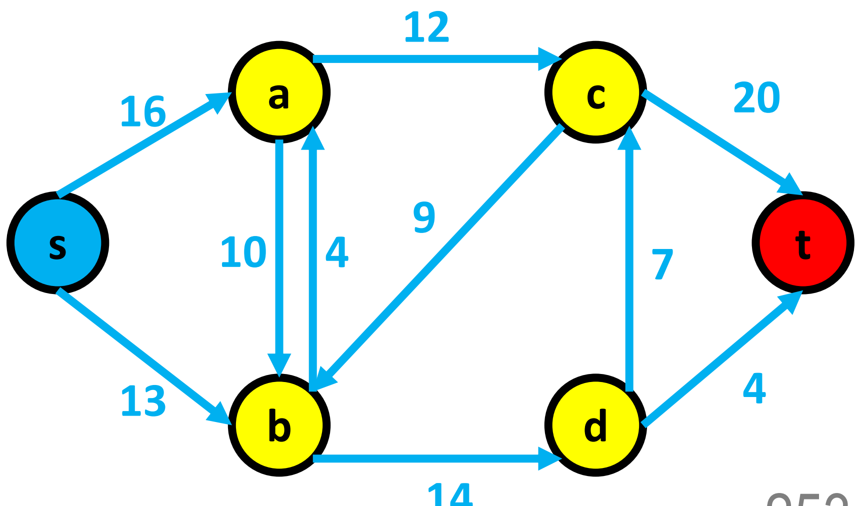
Edmond-Karp

Finding max-flow quicker…

- Consider the following:
  - Dinic/ Edmonds-Karp = choose path with fewest edge
    - Running BFS ensure this isn't it?
  - Edmonds-Karp = choose path with largest bottle neck



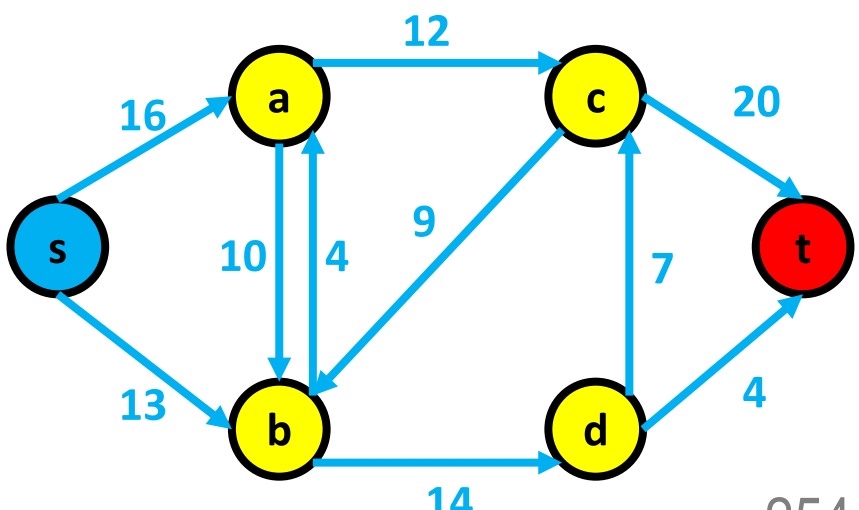
- Consider the following:
  - Dinic/ Edmonds-Karp = choose path with fewest edge
    - Running BFS ensure this isn't it?
    - Less edge, lower chance to get smaller weights
  - Edmonds-Karp = choose path with largest bottle neck



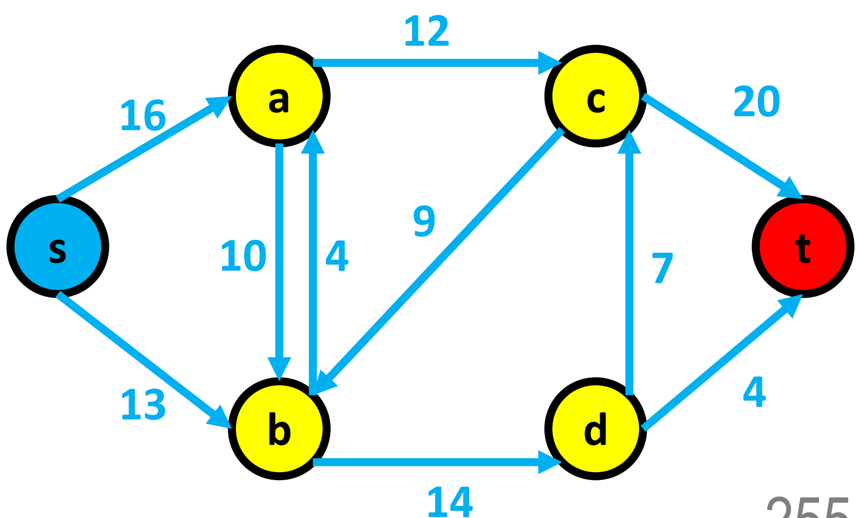
Edmond-Karp

Finding max-flow quicker…

- Consider the following:
  - Dinic/ Edmonds-Karp = choose path with fewest edge
    - Running BFS ensure this isn't it?
    - Less edge, lower chance to get smaller weights
  - Edmonds-Karp = choose path with largest bottle neck
    - Find a maximum spanning tree! convert minimum spanning tree into maximum spanning tree



- Consider the following:
  - Dinic/ Edmonds-Karp = choose path with fewest edge
    - Running BFS ensure this isn't it?
    - Less edge, lower chance to get smaller weights
  - Edmonds-Karp = choose path with largest bottle neck
    - Find a maximum spanning tree! convert minimum spanning tree in previous unit to max spanning tree by negating weight in prim's and Kruskel's -> edge with largest bottleneck
    - Recall we discuss a little in FIT2004 Tutorial 10 about bottleneck…



Questions?

Bipartite Graph

An application of network flow…

- Consider the situation…

- Arranged marriage agency…

Bipartite Graph

An application of network flow…

- Arranged marriage agency…
  - A list of men
  - A list of women

Bipartite Graph

An application of network flow…

- Arranged marriage agency…
  - A list of men
  - A list of women
  - Their preferences

Bipartite Graph
An application of network flow…

- Arranged marriage agency…
  - A list of men
  - A list of women
  - Their preferences
  - Each match you get, you earn $$$

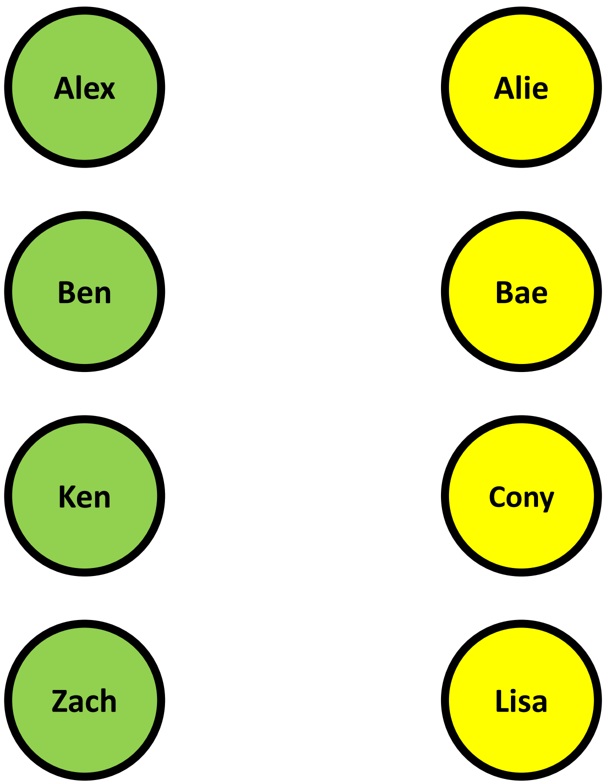
Bipartite Graph
An application of network flow…

- Arranged marriage agency…
  – A list of men
  – A list of women
  – Their preferences
  – Each match you get, you earn $$$
  – So we want the most matches!

Bipartite Graph

An application of network flow…

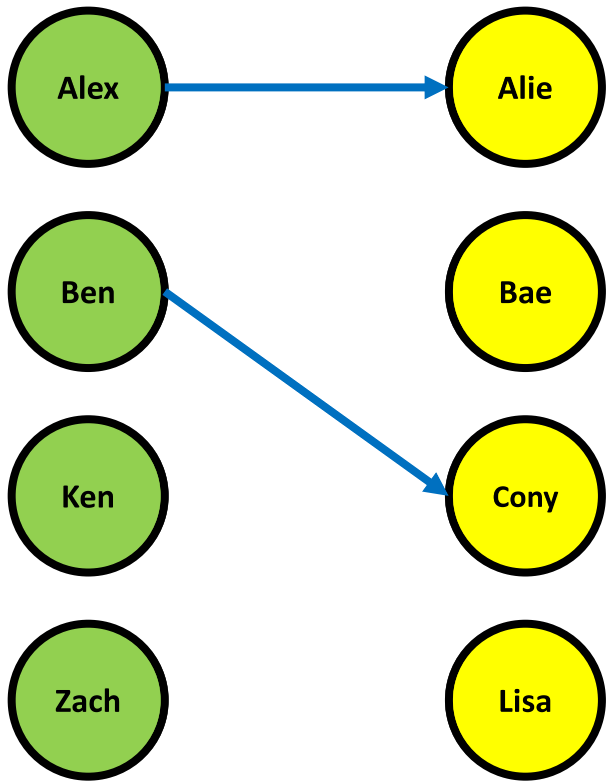
- Arranged marriage agency…



Bipartite Graph

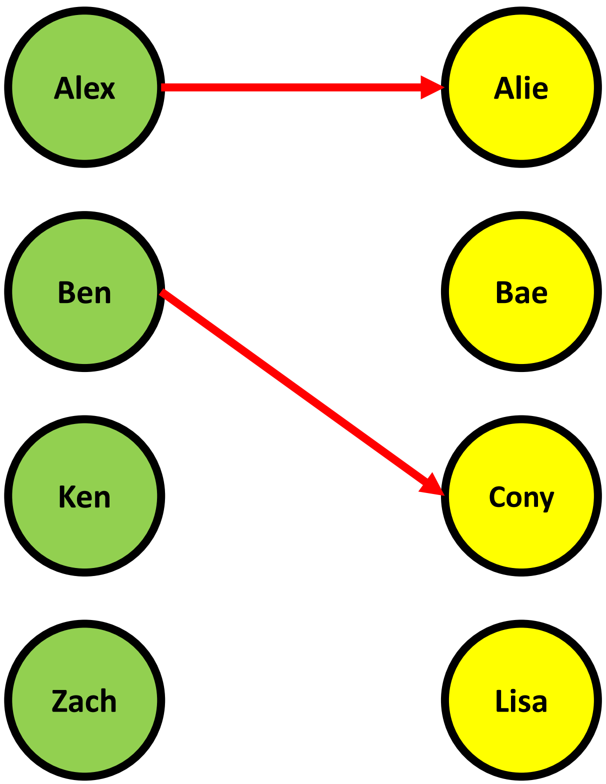
An application of network flow…

- Arranged marriage agency…
  - Can you max profit?



Bipartite Graph
An application of network flow…

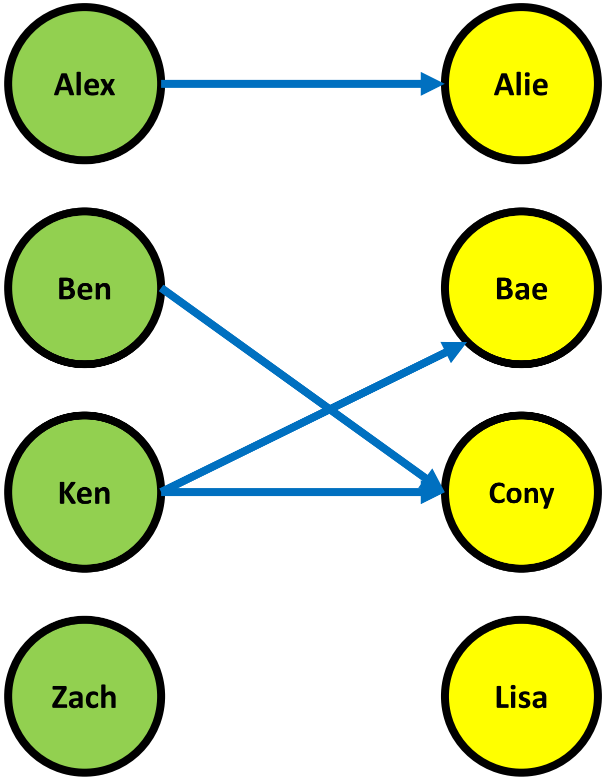
- Arranged marriage agency…
  - Can you max profit?



Bipartite Graph

An application of network flow…

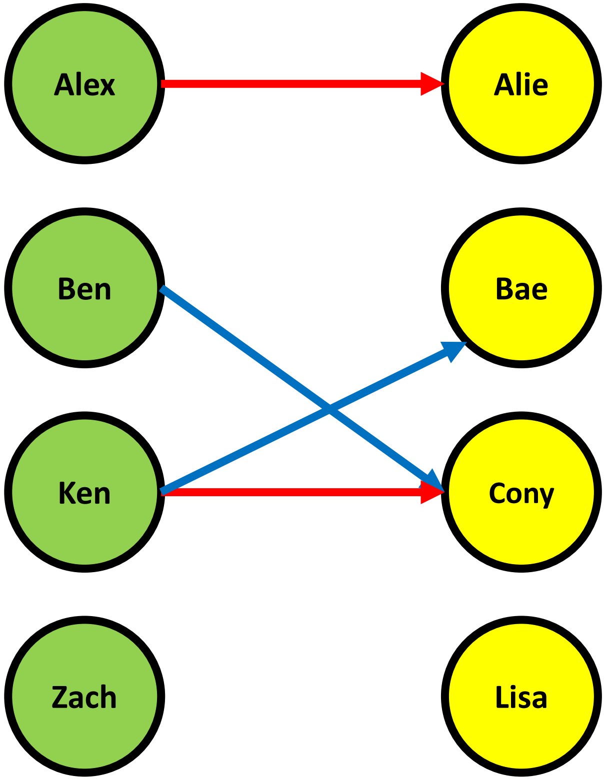
- Arranged marriage agency…
  - Can you max profit? What about now?



Bipartite Graph

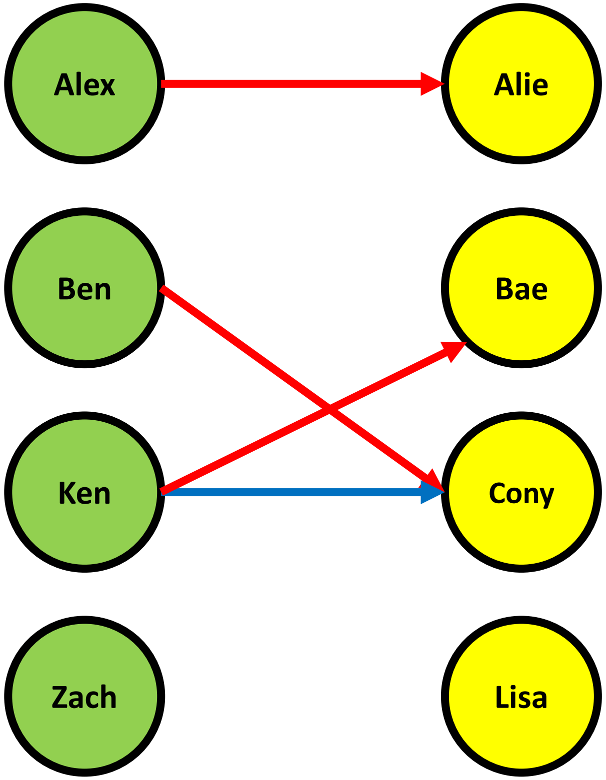
An application of network flow…

- Arranged marriage agency…
  - Can you max profit? What about now? Only 2…



Bipartite Graph
An application of network flow…

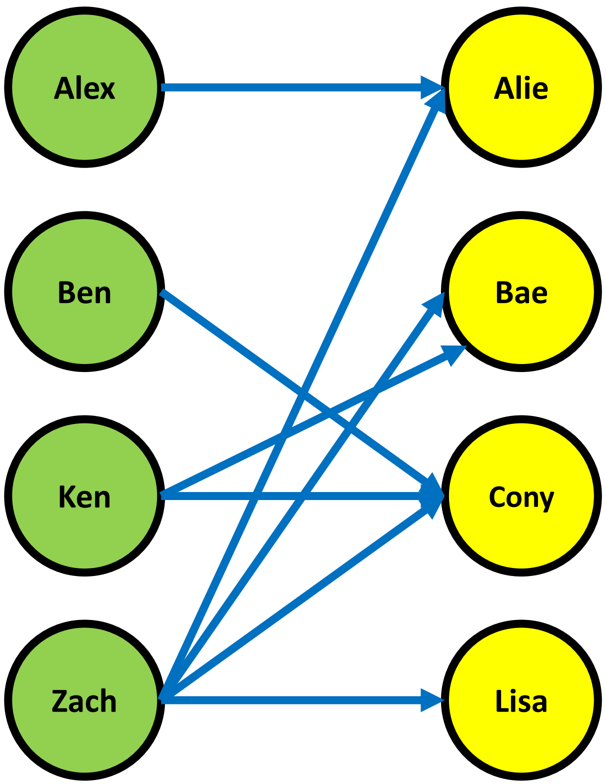
- Arranged marriage agency…
  - Can you max profit? What about now? We can do better with 3



Bipartite Graph

An application of network flow…

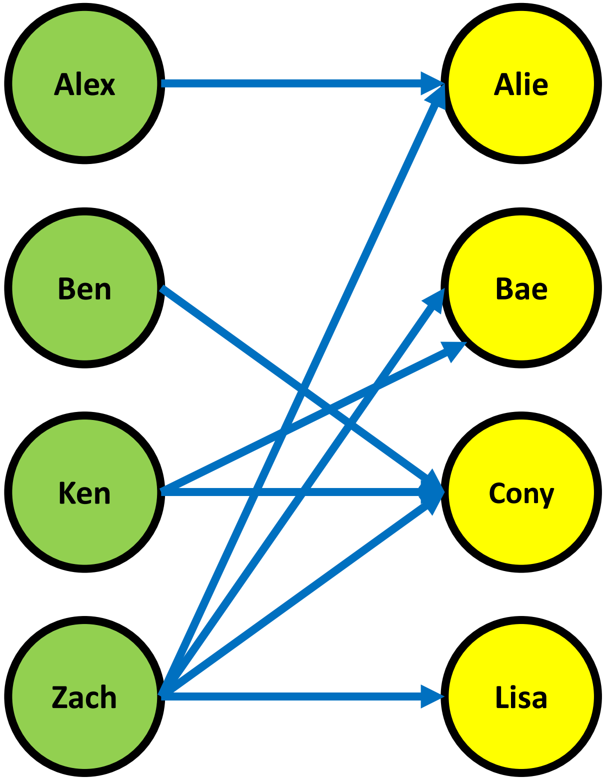
- Arranged marriage agency…
  - What about now???



Bipartite Graph

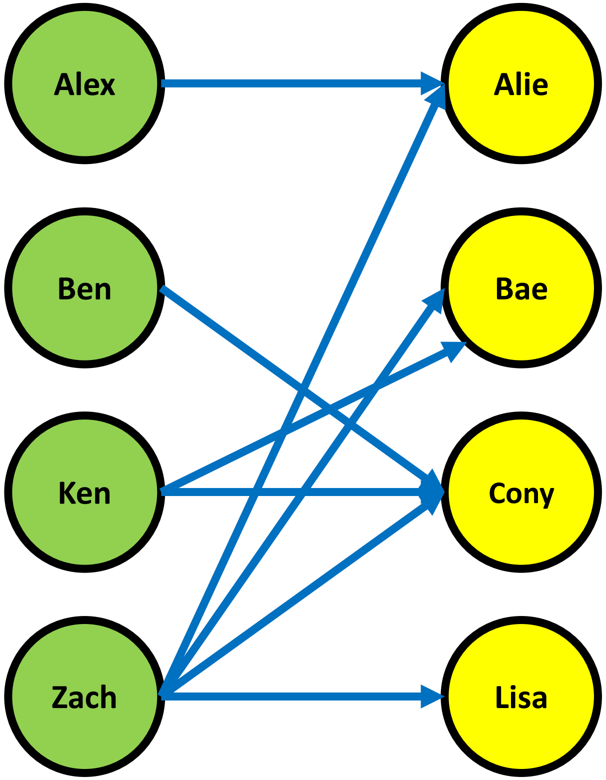
An application of network flow…

- Arranged marriage agency…
  - What about now???



Bipartite Graph
An application of network flow…

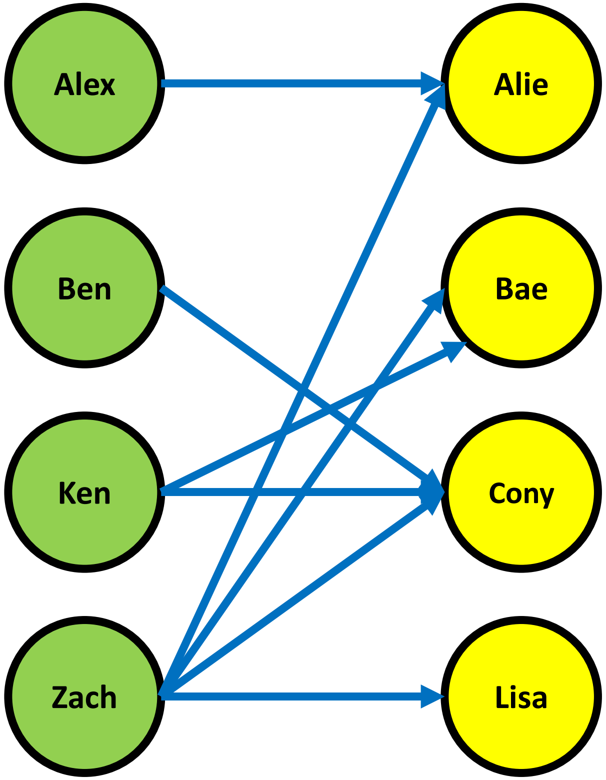
- Arranged marriage agency…
  - What about now???
  - Imagine you run a bigger agency!
    - 100s of eligible bachelor
    - 100s of eligible ladies



Bipartite Graph

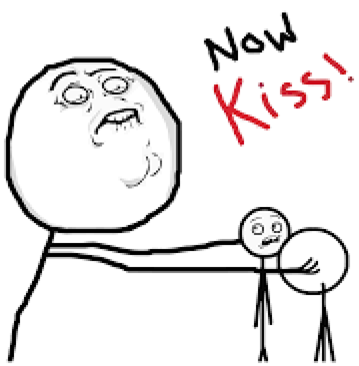
An application of network flow…

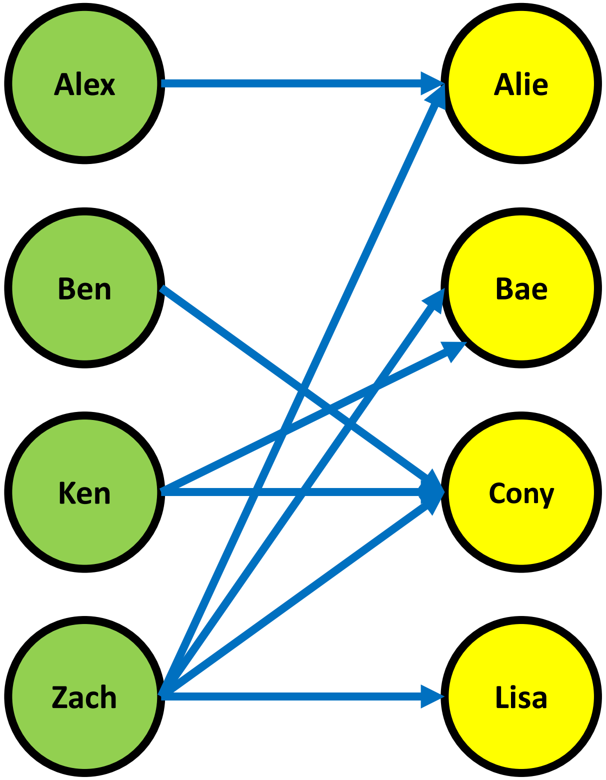
- Arranged marriage agency…
  - What about now???
  - Imagine you run a bigger agency!
    - 100s of eligible bachelor
    - 100s of eligible ladies
    - Or Tinder!



Bipartite Graph
An application of network flow…

- Arranged marriage agency…
  - What about now???
  - Imagine you run a bigger agency!
    - 100s of eligible bachelor
    - 100s of eligible ladies
    - Or Tinder!
  - What would you do?

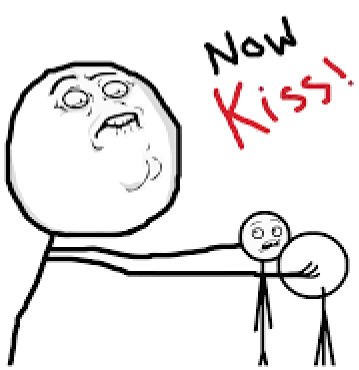


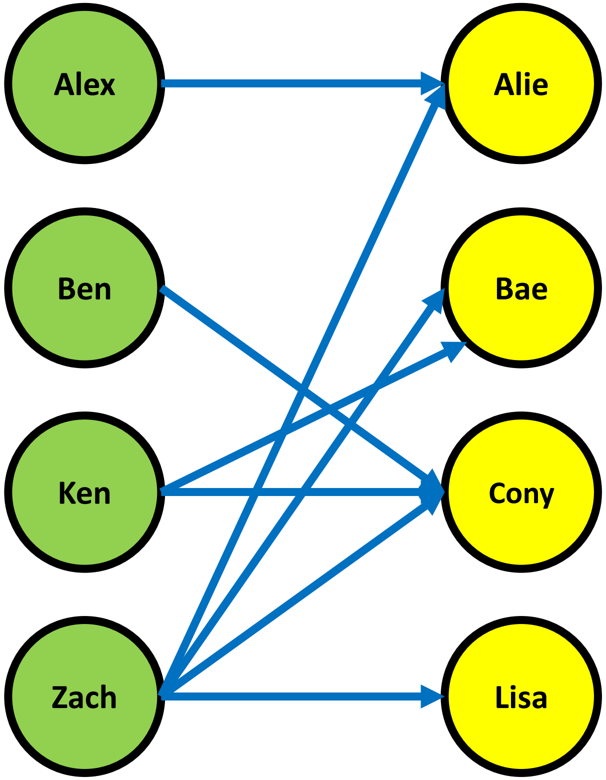


Bipartite Graph

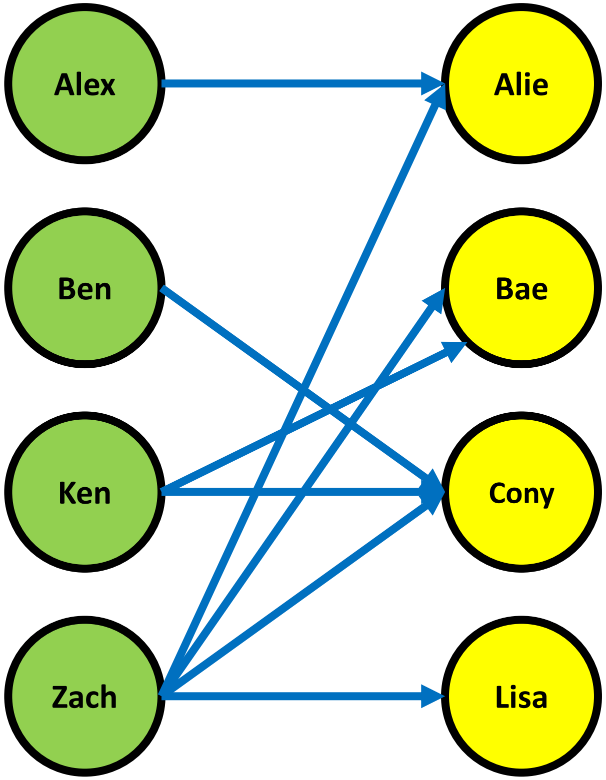
An application of network flow…

- Arranged marriage agency…
  - What about now???
  - Imagine you run a bigger agency!
    - 100s of eligible bachelor
    - 100s of eligible ladies
    - Or Tinder!
  - What would you do? Maximize the flow!





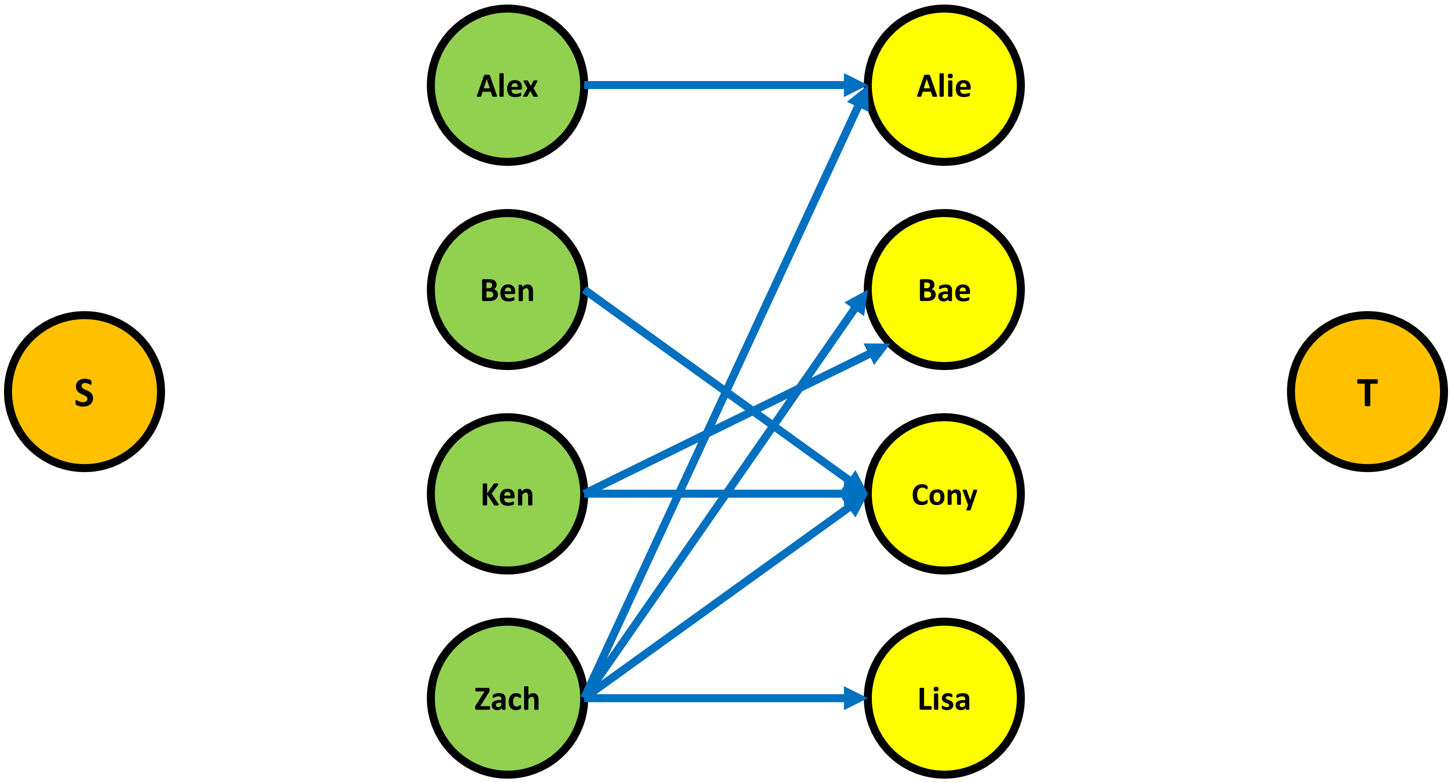
- Arranged marriage agency…
  - What would you do? Maximize the flow!



Bipartite Graph

An application of network flow…

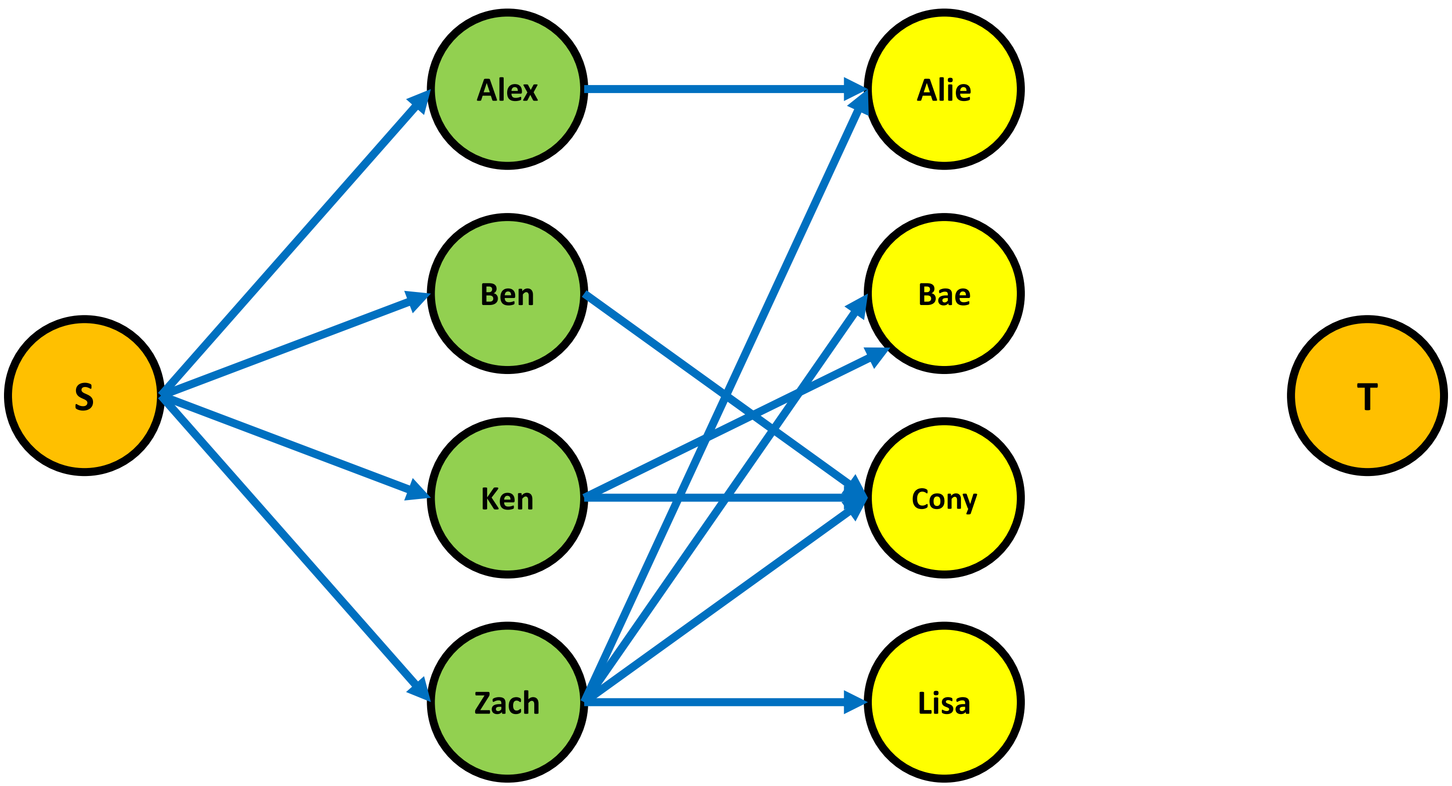
- Arranged marriage agency…
  - What would you do? Maximize the flow!



Bipartite Graph

An application of network flow…

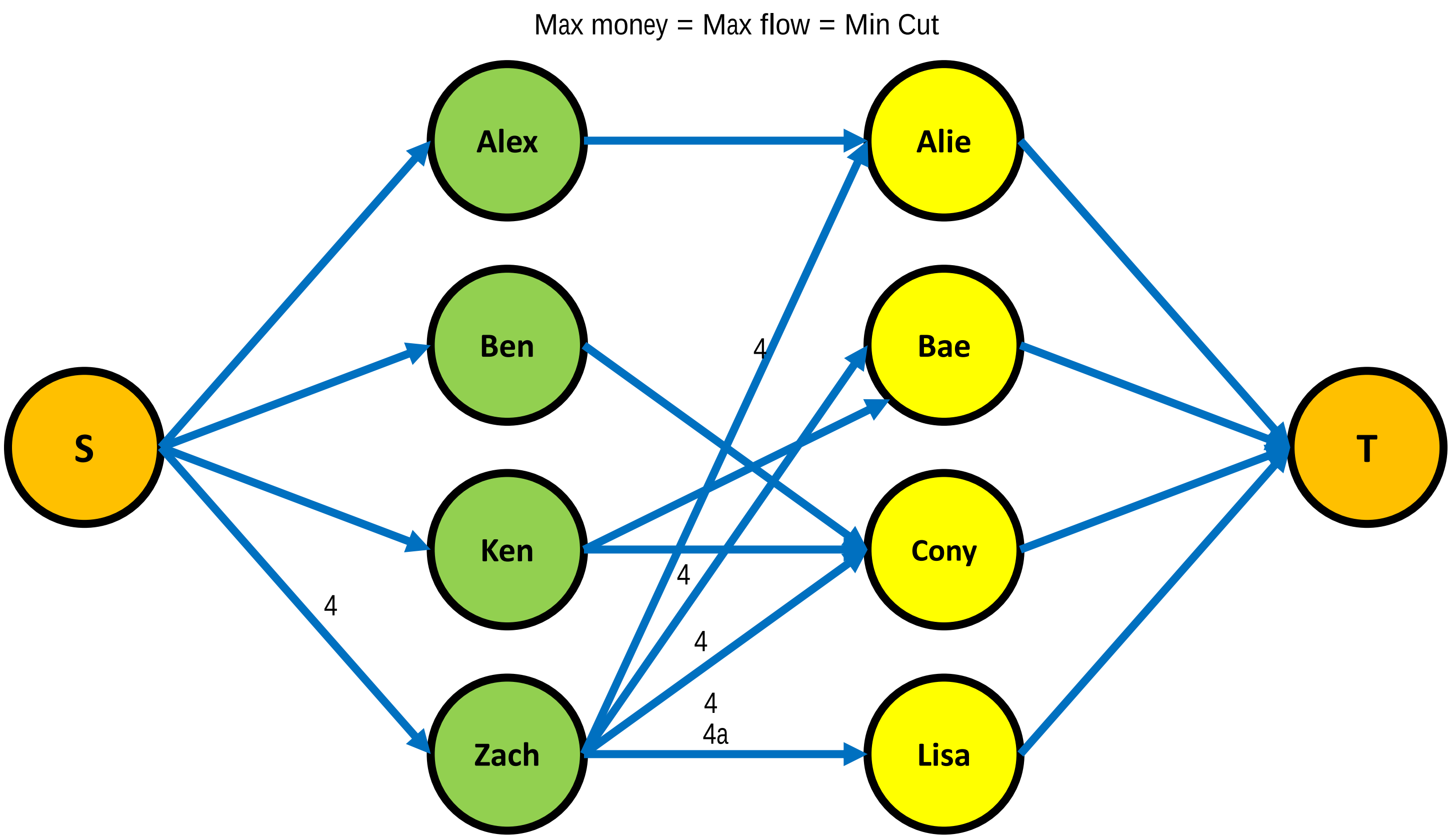
- Arranged marriage agency…
  – What would you do? Maximize the flow!



Bipartite Graph

An application of network flow…

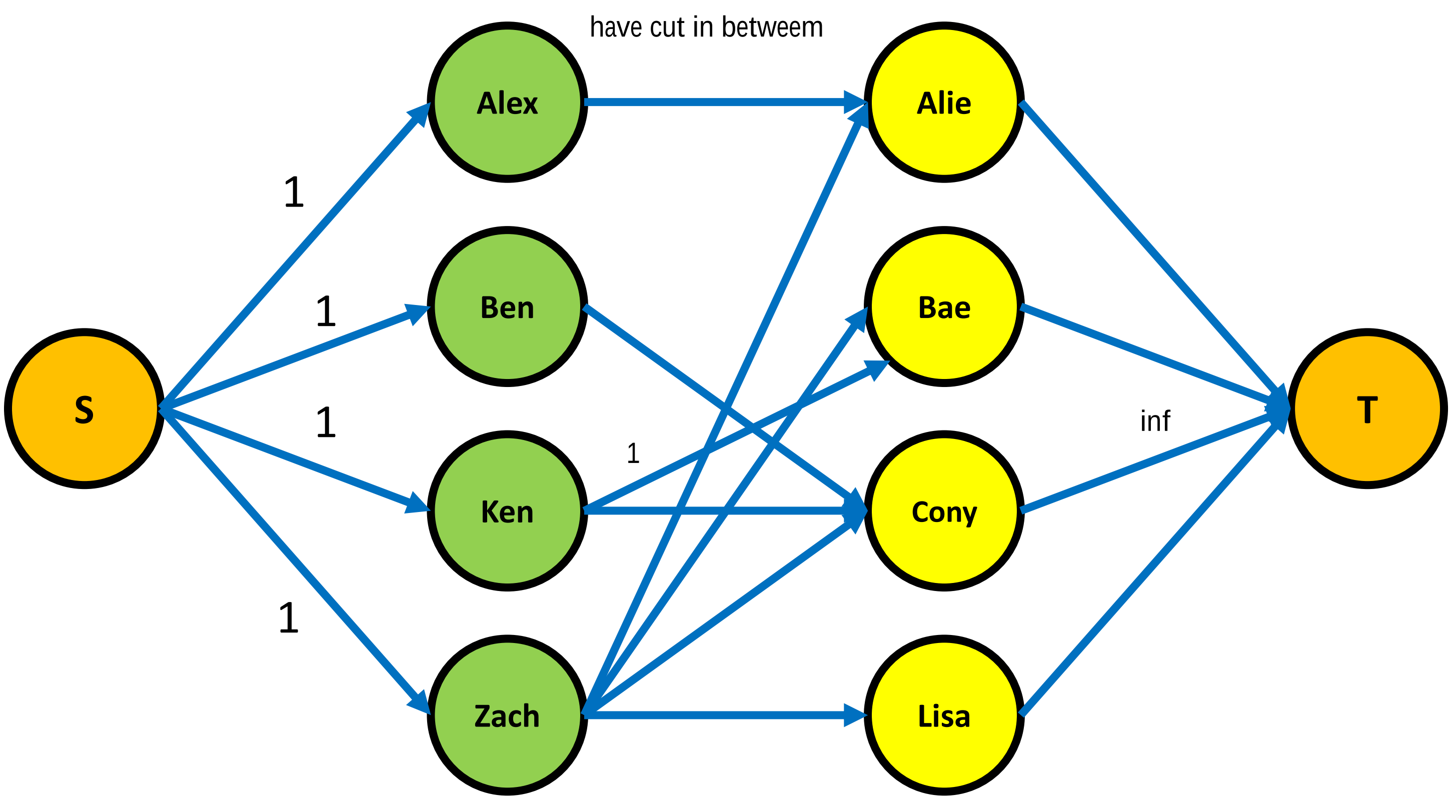
- Arranged marriage agency…
  - What would you do? Maximize the flow!



Bipartite Graph

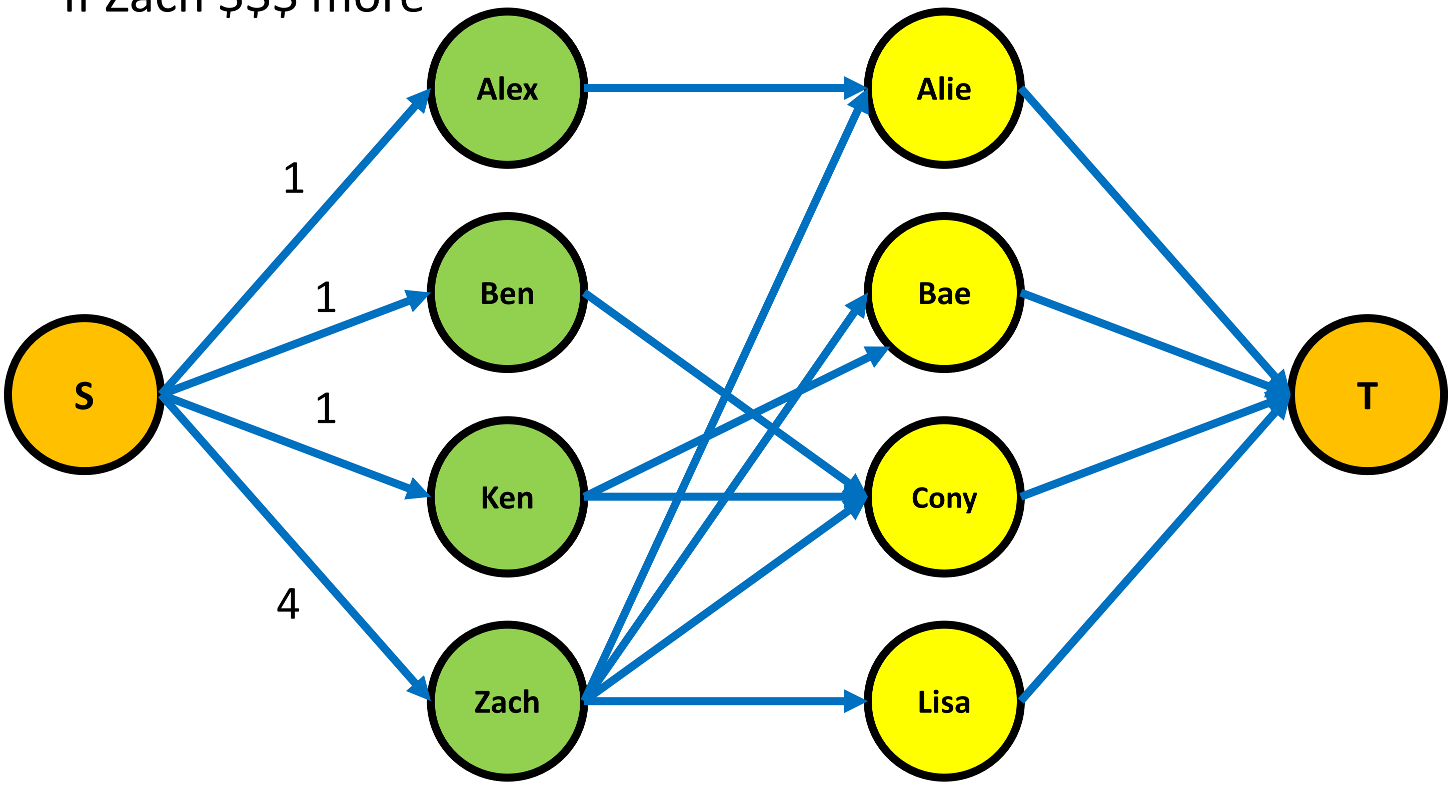
An application of network flow…

- Arranged marriage agency…
  - What would you do? Maximize the flow! … I won't finish this #lazy2draw



MONASH University

- Arranged marriage agency…
  - What would you do? Maximize the flow! … I won't finish this #lazy2draw
  - If Zach $$$ more



Questions?

Bipartite Graph

An application of network flow…

- Arranged marriage agency…
  - A list of men
  - A list of women
  - Their preferences
  - Each match you get, you earn $$$
  - So we want the most matches!

- Can you think of other applications?

Bipartite Graph

An application of network flow…

- Arranged marriage agency…
  - A list of men
  - A list of women
  - Their preferences
  - Each match you get, you earn $$$
  - So we want the most matches!

- Can you think of other applications?
  - Think of Monash

Questions?

Thank You