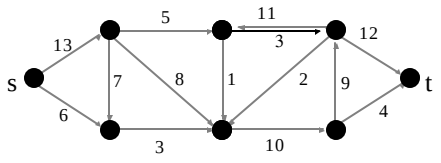
## Max Flow in a Network

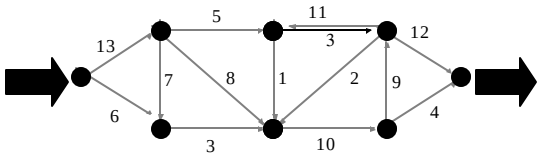
- Input: Flow Network






## Max Flow in a Network

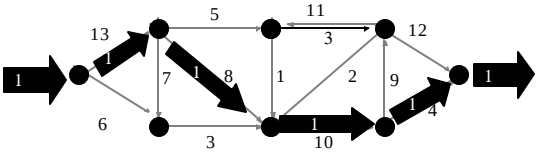
- Output: Maximum flow that can be pumped through the network from s to t.






## The rules

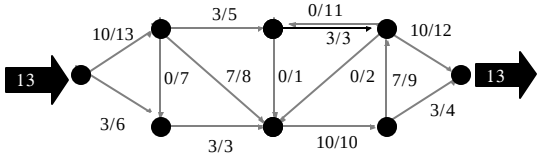
1. The flow along an edge cannot exceed its capacity
2. The total flow into a node must equal the total flow out of a node.






## Feasible Flow! But is it a Max Flow?

1. The flow along an edge cannot exceed its capacity
2. The total flow into a node must equal the total flow out of a node.





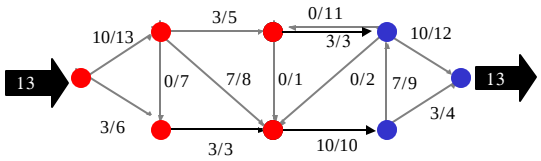

## Max Flow-Min Cut Theorem

- The maximum flow of a network is equal to the minimum capacity of any cut in the network.



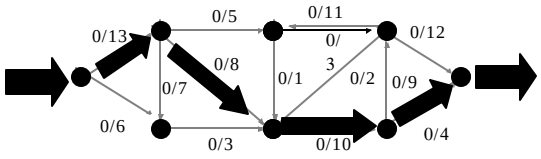
## Feasible Flow! But is it a Max Flow? YES

Here is a matching cut!



## Augmenting path algorithm

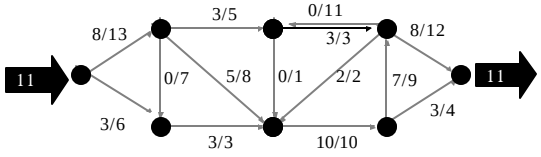
Find an augmenting path: An s => t path in which the capacity of each edge exceeds its flow. Push more flow through.




CS140 26-7

## Augmenting Path in network doesn't always work!

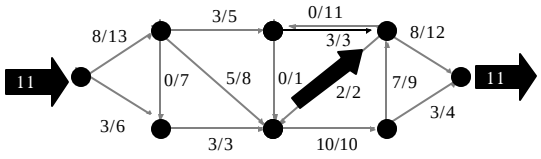
We can get stuck! What should we do?




CS140 26-8

## Augmenting Path in network doesn't always work!

What should we do? Reroute!




CS140 26-9

## One more idea

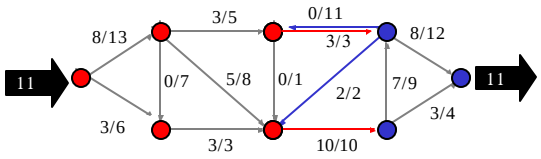
Given a network and a flow:

The flow across a cut (R,B) is

$\Sigma f(<u,v>) - \Sigma f(<u,v>)$

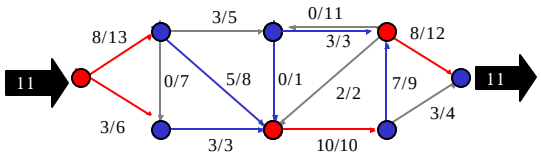
CS140 26-10

## Flow across (R,B)




CS140 26-11

## Flow across (R,B)




CS140 26-12

## Flow across a cut

- Conservation of flow
- The net flow across any cut equals the network flow

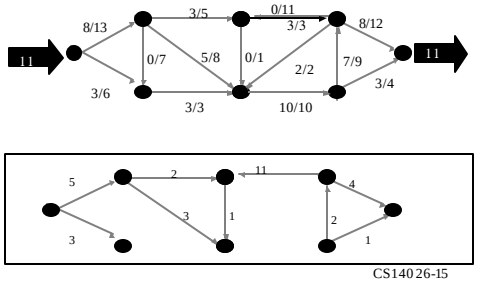
CS140 26-13

## Flow across a cut

- Conservation of flow + max flow/min cut theorem
- If flow is max then for minimum capacity cut (R,B): $\Sigma$ f(<u,v>)=0

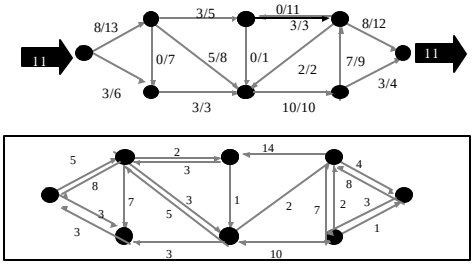
CS140 26-14

## Build residual graph
1. Add e if flow(e)<capacity(e)



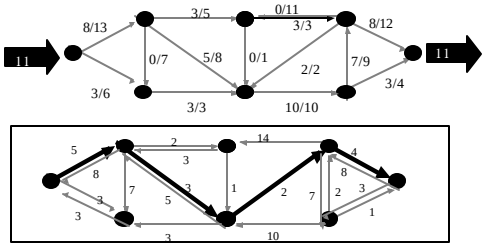

CS140 26-15

## Build residual graph
2. Add reverse(e) if flow(e)>0



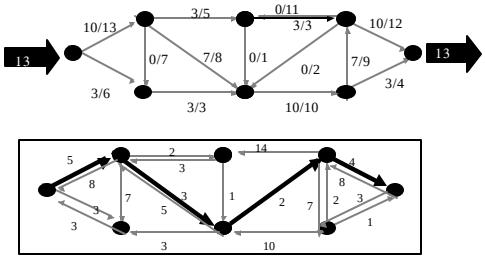

CS140 26-16

## Find s→t path in the residual graph




CS140 26-17

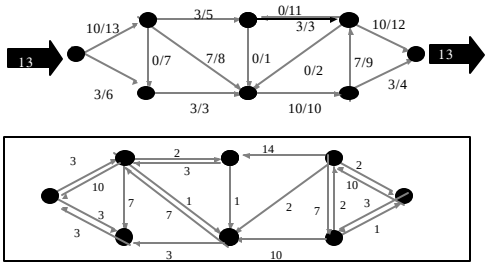
## Augment the s→t path in the network




CS140 26-18

And then …

Repeat until residual graph has no s –t path






## Augmenting Path Method (Ford-Fulkerson)

- **Is it correct?**
- Is it efficient?



## Claim

- The flow in a network is a maximum flow if and only if there is no path from s to t in the corresponding residual graph.



## Proof of Claim

The flow in a network is a maximum flow.

⇓

There is no path from s to t in the corresponding residual graph.



## Proof of Claim

The flow in a network is a maximum flow.

⇓

There is no path from s to t in the corresponding residual graph.

Proof: If there is an s to t path in the residual graph then there is an augmenting path in the network and we can increase the flow.



## Proof of Claim

The flow in a network is a maximum flow.

⇑

There is no path from s to t in the corresponding residual graph.

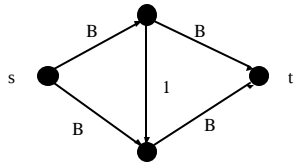
Follows from previous argument.

## Augmenting Path Method (Ford-Fulkerson)

- Is it correct?
- **Is it efficient?**

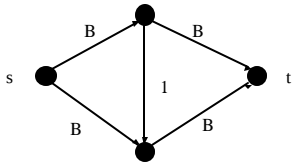


## Problems: How many augmentations could this take?



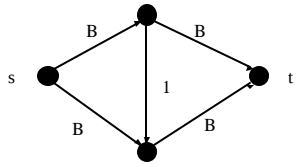



## B Augmentations – What is size of input?






## B Augmentations Input has size n + m lg(B)






## Edmonds-Karp

- Choose shortest s to t path in residual graph to augment

- Total augmentations is O(mn)



## Edmonds-Karp

- Build residual graph
- If $s \rightarrow t$ path does not exist then return current flow
- Find shortest $s \rightarrow t$ path
- Augment flow in network
- Repeat

## Reductions to Network Flow Problem

- Bipartite Matching $\propto$ Network Flow



## Matching

- Let G=(V,E) be a graph.
- E' $\subseteq$ E is a matching if every vertex of V is incident to at most one edge of E'.



## Matching Example

G E'



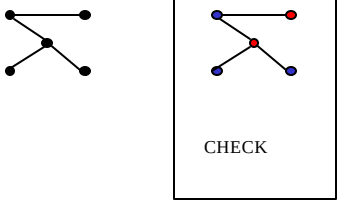
## Bipartite Graph

- Let G=(V,E) be a graph.
- G is bipartite if V can be partitioned into $V_1$ and $V_2$ such that no pair of vertices in $V_i$ (i=1,2) have an edge.



## Bipartite Example

G G

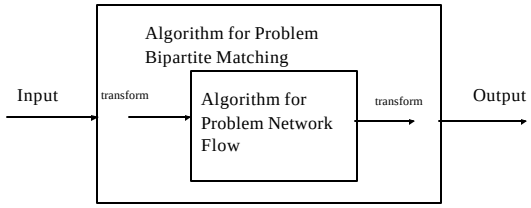





## Bipartite Matching

- Input: Bipartite graph G
- Output: A maximum matching of G

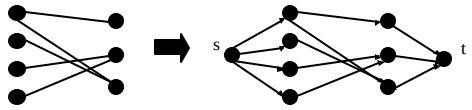
## Bipartite Matching ∝ Network Flow






## Bipartite Matching ∝ Network Flow

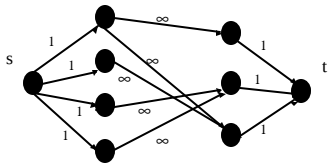
- Transform Input – Step 1





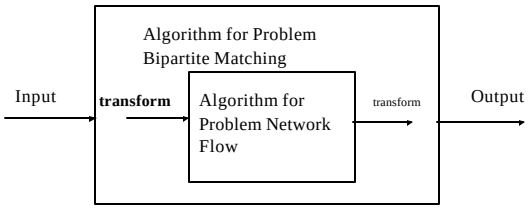

## Bipartite Matching ∝ Network Flow

- Transform Input – Step 2





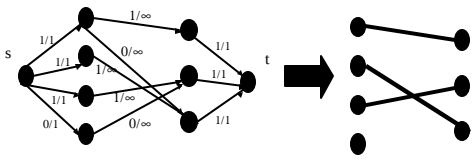

## Bipartite Matching ∝ Network Flow






## Bipartite Matching ∝ Network Flow

- Transform output






## Reduction

- Is it correct?
- Is it efficient?