

# COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

## LECTURE 30

### NETWORK FLOW II :

### MAXIMUM FLOWS AND MINIMUM CUTS

OCT 21, 2026

|

ROHIT VAISH

# MAXIMUM FLOW PROBLEM

an **instance** of max flow problem is defined by:

a directed graph  $G = (V, E)$

a **source** vertex  $s \in V$

a **sink** vertex  $t \in V$

a non negative and integral **capacity**  $u_e$  for each edge  $e \in E$

# MAXIMUM FLOW PROBLEM

Feasible Solutions : **Flows**

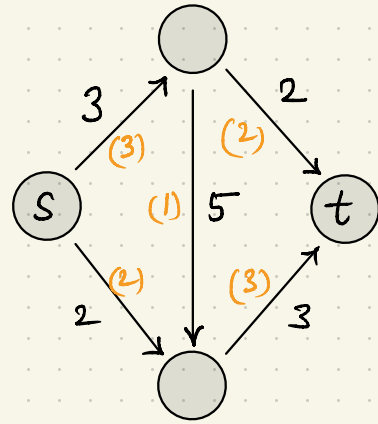
A nonnegative vector  $\{f_e\}_{e \in E}$  Satisfying

1. **Capacity constraints**

$$f_e \leq u_e \quad \forall e \in E$$

2. **Conservation constraints**

$$\text{flow in} = \text{flow out} \quad \forall v \neq s, t$$



# IDEA # 1 : NAIVE GREEDY APPROACH

# IDEA # 1 : NAIVE GREEDY APPROACH

Compute a sequence of flows with progressively higher values

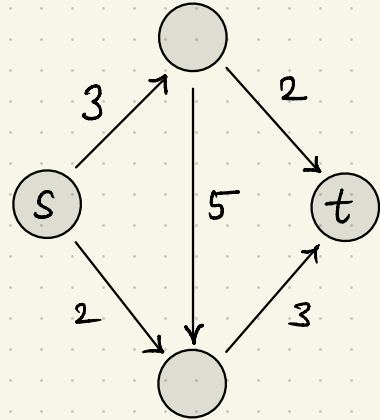
- ① initialize  $f_e = 0 \quad \forall e$
- ② repeat:
  - \* search for an  $s-t$  path  $P$  such that  $f_e < u_e \quad \forall e \in P$ .

// possible in  $O(|E|)$  time via BFS / DFS

\* if no such path, **return** current flow  $\{f_e\}_{e \in E}$

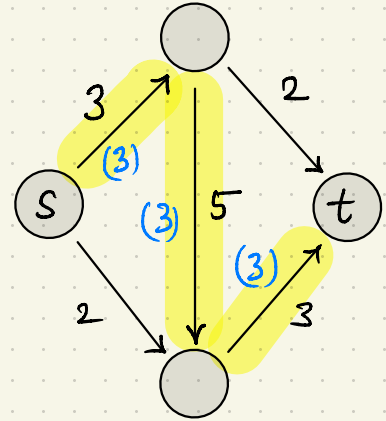
\* else, let  $\Delta := \min_{e \in P} (u_e - f_e)$

└ for all edges  $e \in P$ , increase  $f_e$  by  $\Delta$ .



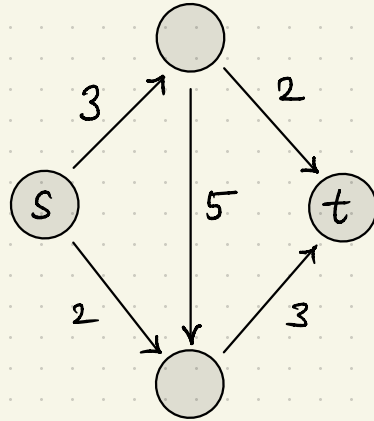
# IDEA # 1 : NAIVE GREEDY APPROACH

Could fail to return a max flow.

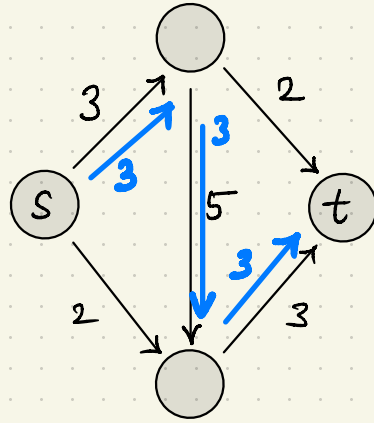


IDEA # 2 : NAIVE GREEDY + "UNDO"

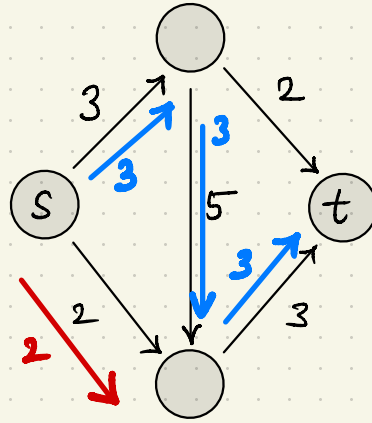
## IDEA # 2 : NAIVE GREEDY + "UNDO"



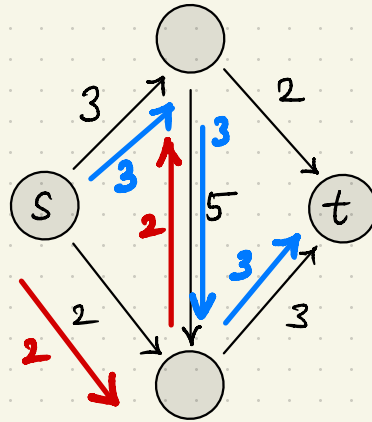
## IDEA # 2 : NAIVE GREEDY + "UNDO"



## IDEA # 2 : NAIVE GREEDY + "UNDO"

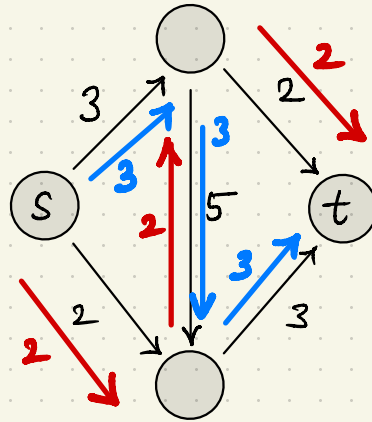


## IDEA # 2 : NAIVE GREEDY + "UNDO"

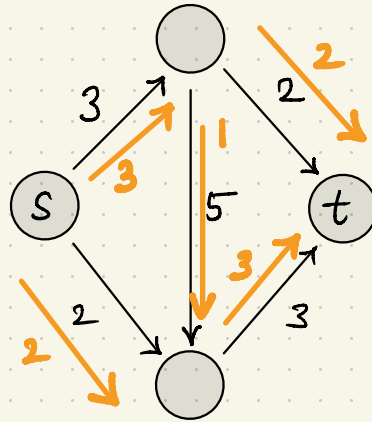


undo previous flow

## IDEA # 2 : NAIVE GREEDY + "UNDO"



## IDEA # 2 : NAIVE GREEDY + "UNDO"



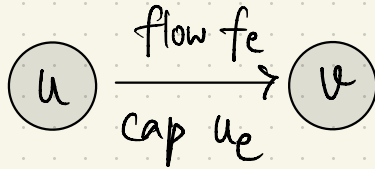
max flow !

## IDEA # 2 : NAIVE GREEDY + "UNDO"

Let  $f$  be a feasible flow in  $G$ .

Residual graph  $G_f$

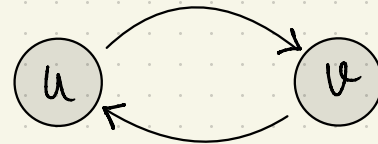
in  $G$



in  $G_f$

scope for  
pushing further

residual cap =  $u_e - f_e$



residual cap =  $f_e$

scope for "undo"

# FORD - FULKERSON ALGORITHM

# FORD - FULKERSON ALGORITHM

① initialize  $f_e = 0 \quad \forall e$

② repeat :

\* search for an  $s-t$  path  $P$  in  $G_f$   
such that every edge in  $P$  has positive residual capacity

// possible in  $O(|E|)$  time via BFS/DFS

\* if no such path, return current flow  $\{f_e\}_{e \in E}$

\* else, let  $\Delta := \min_{e \in P} e$ 's residual capacity in  $G_f$

└ for all  $e \in G$  with forward edges in  $P$ , add  $\Delta$  to  $f_e$ .  
for all  $e \in G$  with reverse edges in  $P$ , subtract  $\Delta$  from  $f_e$ .

# TERMINATION OF FORD - FULKERSON

# TERMINATION OF FORD - FULKERSON

Claim 1: The algorithm maintains the invariant that  $\{f_e\}_{e \in E}$  is a flow.

Claim 2: The algorithm maintains the invariant that  $\{f_e\}_{e \in E}$  is integral.

Claim 3:  $\Delta \geq 1$ .

Claims 1, 2, 3 and the fact that finite amount of capacity exiting  $s$   
 $\Rightarrow$  termination in a finite number of iterations.

# TERMINATION OF FORD - FULKERSON

For integral capacities,

Ford - Fulkerson terminates in finite time with a valid flow.

# TERMINATION OF FORD - FULKERSON

For integral capacities,

Ford - Fulkerson terminates in finite time with a valid flow.

Does the algorithm return a max flow?

# TERMINATION OF FORD - FULKERSON

For integral capacities,

Ford - Fulkerson terminates in finite time with a valid flow.

Does the algorithm return a max flow?



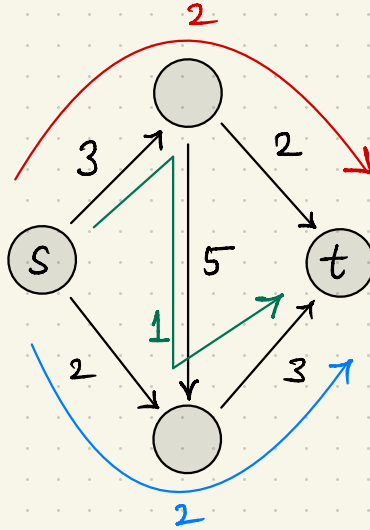
HOW DO WE KNOW WHEN WE ARE DONE?

# OPTIMALITY CONDITIONS

How DO WE KNOW WHEN WE ARE DONE?

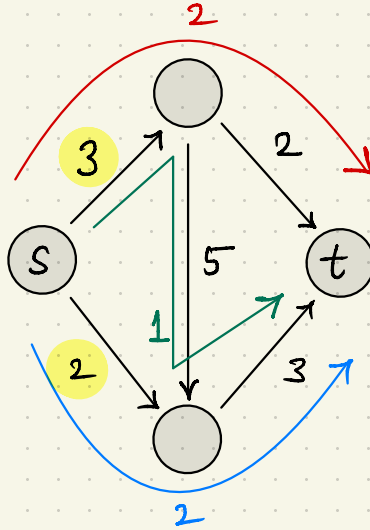
# OPTIMALITY CONDITIONS

How DO WE KNOW WHEN WE ARE DONE?



# OPTIMALITY CONDITIONS

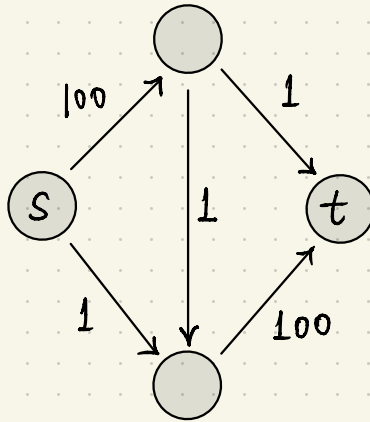
How DO WE KNOW WHEN WE ARE DONE?



easy!

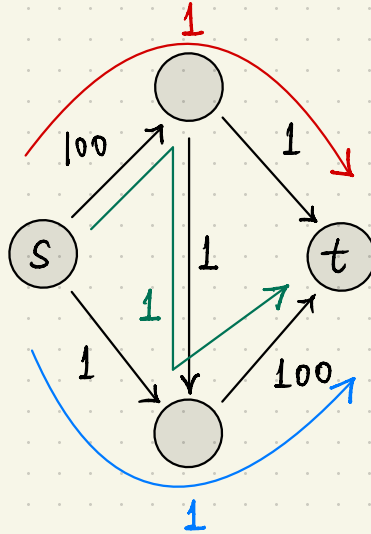
# OPTIMALITY CONDITIONS

How DO WE KNOW WHEN WE ARE DONE?



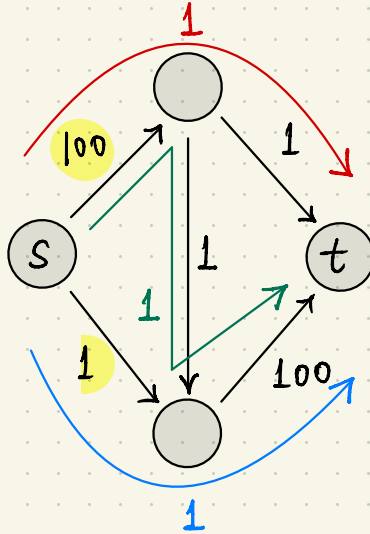
# OPTIMALITY CONDITIONS

How DO WE KNOW WHEN WE ARE DONE?



# OPTIMALITY CONDITIONS

How DO WE KNOW WHEN WE ARE DONE?

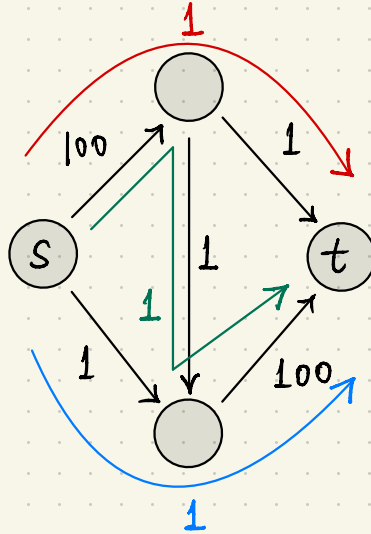


$101 \gg 3$

Could this be  
a max flow?

# OPTIMALITY CONDITIONS

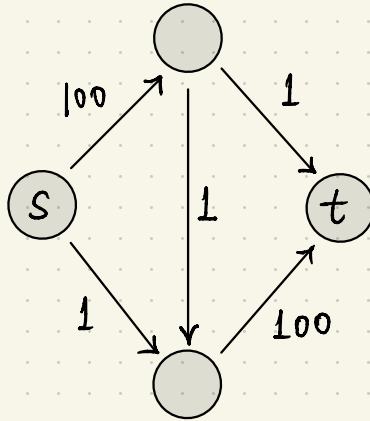
How DO WE KNOW WHEN WE ARE DONE?



Identify a **bottleneck** subset of vertices

# OPTIMALITY CONDITIONS

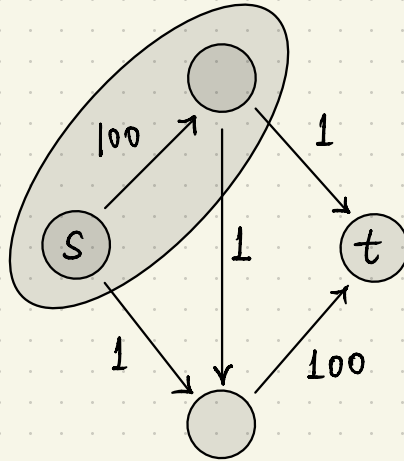
How DO WE KNOW WHEN WE ARE DONE?



Identify a **bottleneck** subset of vertices

# OPTIMALITY CONDITIONS

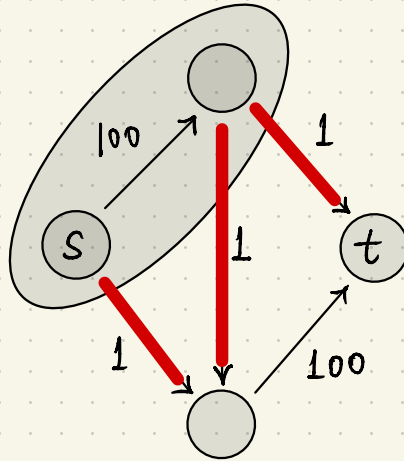
How DO WE KNOW WHEN WE ARE DONE?



Identify a **bottleneck** subset of vertices

# OPTIMALITY CONDITIONS

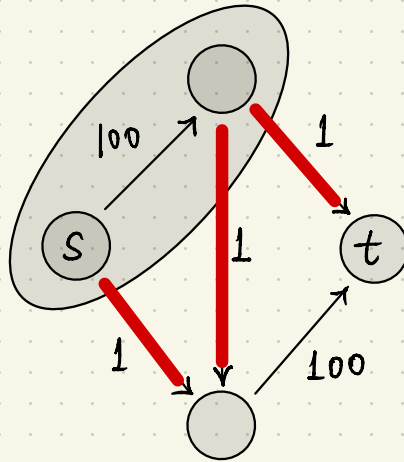
How DO WE KNOW WHEN WE ARE DONE?



Identify a **bottleneck** subset of vertices

# OPTIMALITY CONDITIONS

How DO WE KNOW WHEN WE ARE DONE?



All flow going into the sink has to cross this set.



Identify a **bottleneck** subset of vertices

# OPTIMALITY CONDITIONS

A general recipe :

# OPTIMALITY CONDITIONS

A general recipe :

- ① (Structure) Identify "optimality conditions".
- ② (Algorithm) Design an algorithm that terminates with the optimality conditions satisfied.

# OPTIMALITY CONDITIONS

A general recipe :

- ① (Structure) Identify "optimality conditions".
- ② (Algorithm) Design an algorithm that terminates with the optimality conditions satisfied.

# OPTIMALITY CONDITIONS

A general recipe :

① (Structure) Identify "optimality conditions".

Claim : If  $G_f$  has no  $s-t$  path, then  $f$  is a max flow in  $G$

② (Algorithm) Design an algorithm that terminates with the optimality conditions satisfied.

# OPTIMALITY CONDITIONS

A general recipe :

① (Structure) Identify "optimality conditions".

Claim : If  $G_f$  has no  $s-t$  path, then  $f$  is a max flow in  $G$

② (Algorithm) Design an algorithm that terminates  
with the optimality conditions satisfied.

# OPTIMALITY CONDITIONS

A general recipe :

① (Structure) Identify "optimality conditions".

Claim : If  $G_f$  has no  $s-t$  path, then  $f$  is a max flow in  $G$

② (Algorithm) Design an algorithm that terminates with the optimality conditions satisfied.

Ford-Fulkerson algorithm

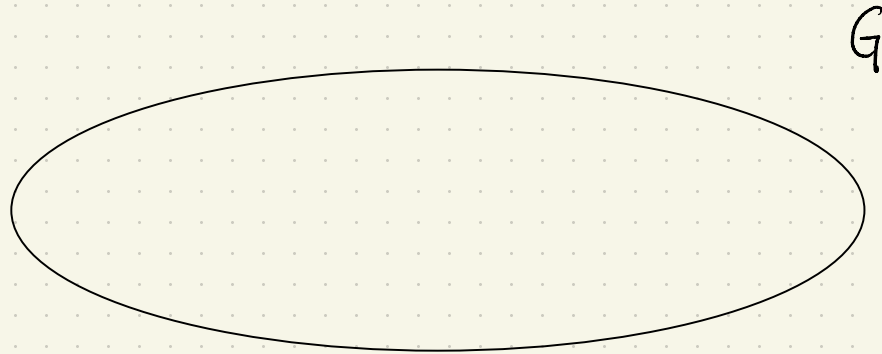
$(s,t)$ -CUT

## $(s,t)$ -CUT

An  $(s,t)$ -cut of a graph  $G=(V,E)$  is a partition of  $V$  into sets  $A, B$  with  $s \in A$  and  $t \in B$ .

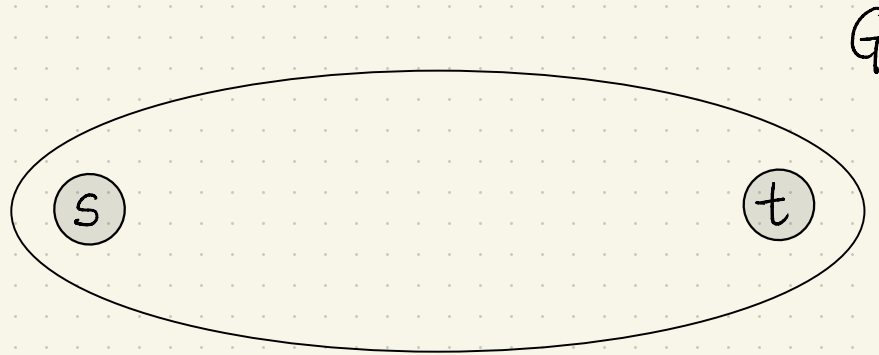
## $(s,t)$ -CUT

An  $(s,t)$ -cut of a graph  $G=(V,E)$  is a partition of  $V$  into sets  $A, B$  with  $s \in A$  and  $t \in B$ .



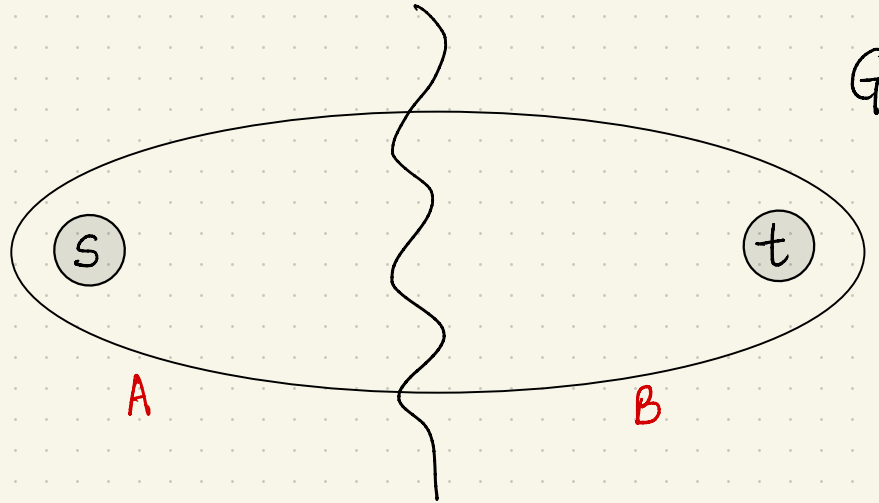
## $(s,t)$ -CUT

An  $(s,t)$ -cut of a graph  $G = (V, E)$  is a partition of  $V$  into sets  $A, B$  with  $s \in A$  and  $t \in B$ .



## $(s,t)$ -CUT

An  $(s,t)$ -cut of a graph  $G = (V, E)$  is a partition of  $V$  into sets  $A, B$  with  $s \in A$  and  $t \in B$ .



## $(s,t)$ -CUT

An  $(s,t)$ -cut of a graph  $G = (V, E)$  is a partition of  $V$  into sets  $A, B$  with  $s \in A$  and  $t \in B$ .

