Lecture 16
Maximum flow

NOTICE: THIS DOCUMENT CONTAINS INFORMATION AFFECTING THE NATIONAL DEFENSE OF THE UNITED STATES WITHIN THE MEANING OF THE ESPIONAGE LAWS, TITLE 18, U.S.C., SECTIONS 793 and 794. THE TRANSMISSION OR THE REVELATION OF ITS CONTENTS IN ANY MANNER TO AN UNAUTHORIZED PERSON IS PROHIBITED BY LAW.

U. S. AIR FORCE

PROJECT RAND

RESEARCH MEMORANDUM

FUNDAMENTALS OF A METHOD FOR EVALUATING RAIL NET CAPACITIES (U)

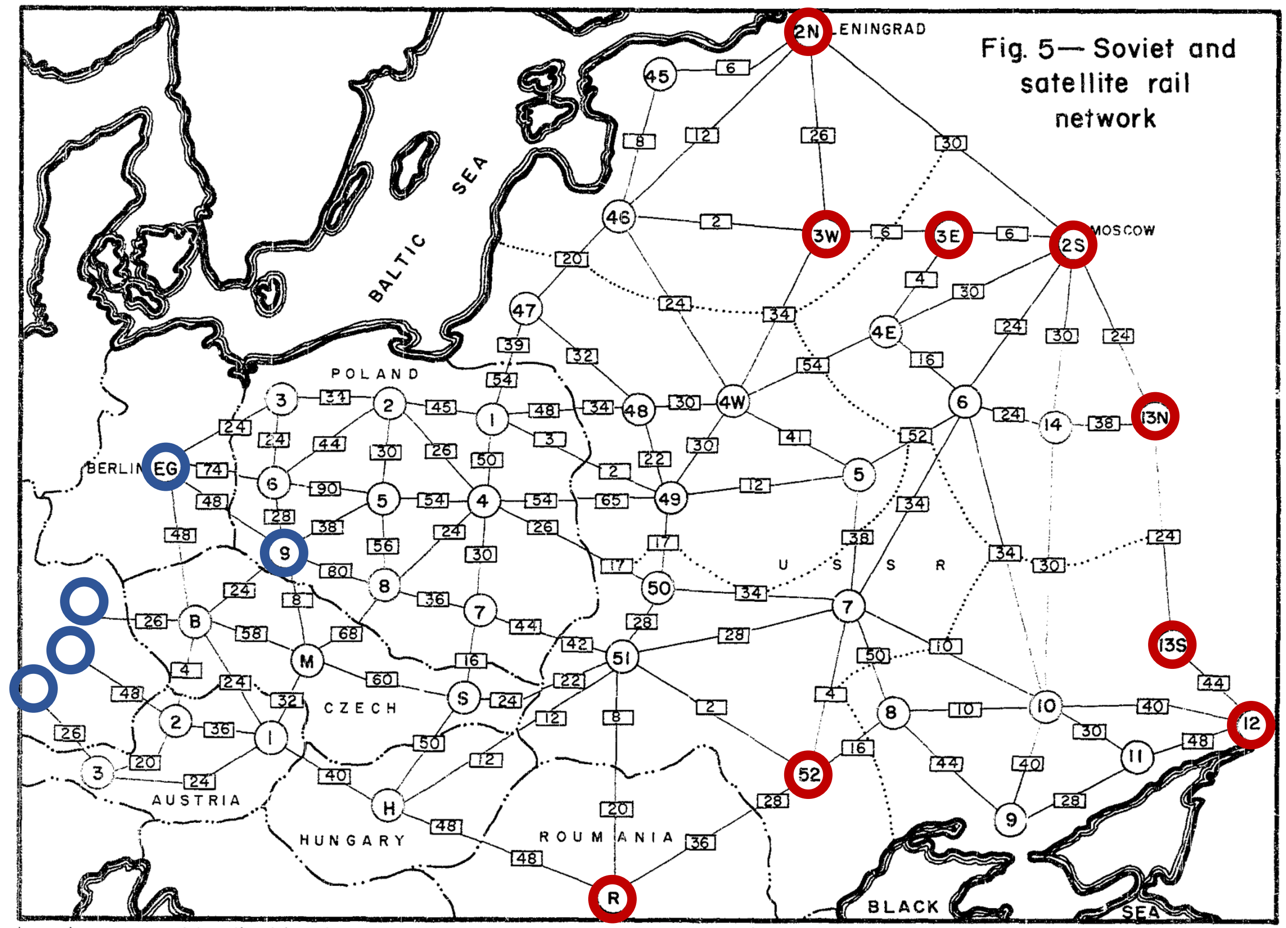
T. E. Harris
F. S. Ross

RM-1573

October 24, 1955

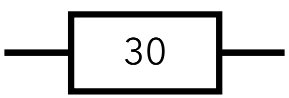
Copy No. 37

SECRET



Legend: —·—·— International boundary ········· Regional boundaries of the USSR (they are included as a matter of general information)

SECRET

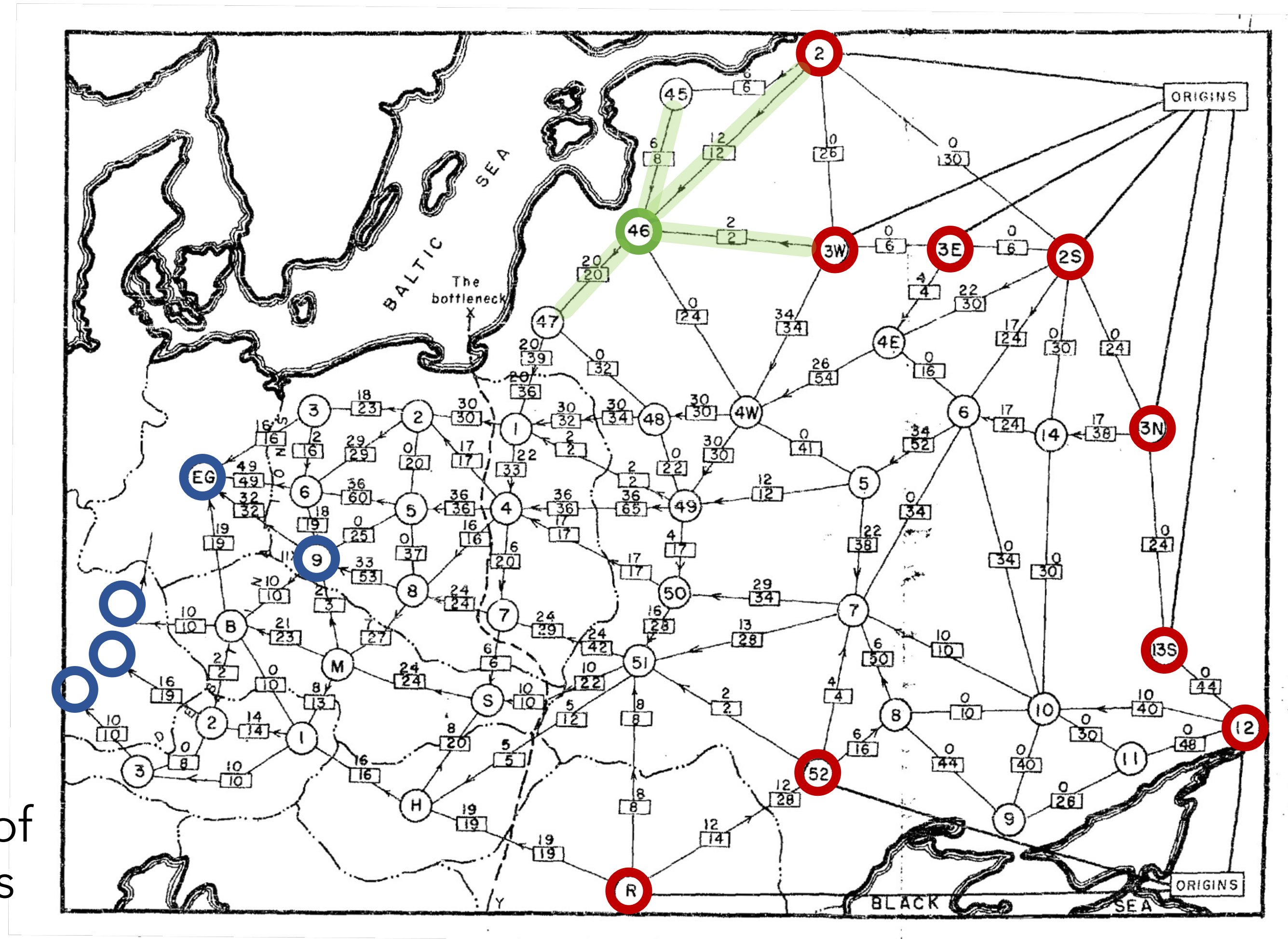


= a capacity of 30,000 tons

: source

: destination

SECRET



SECRET

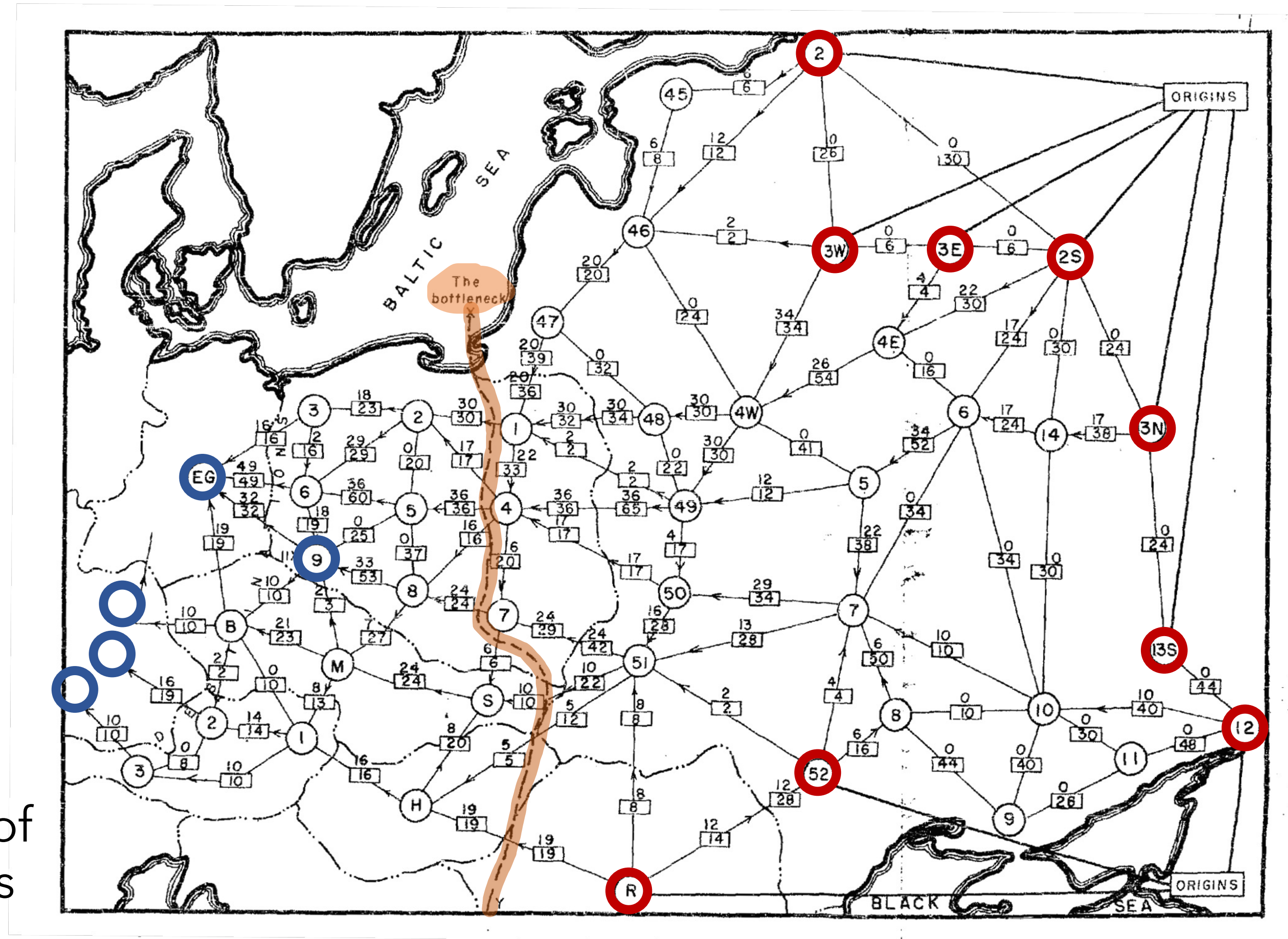
30

= a capacity of 30,000 tons

: source

: destination

SECRET



SECRET

30
= a capacity of 30,000 tons

: source
: destination

Total East → West capacity: **163,000 tons**

Optimal due to **the bottleneck**

Harris and Ross solved this problem
using a **greedy** algorithm they called "flooding"

But flooding would sometimes output **incorrect** solutions

So they approached their colleagues Ford and Fulkerson,
who devised an alg known as the **Ford-Fulkerson algorithm**

MAXIMAL FLOW THROUGH A NETWORK

L. R. FORD, JR. AND D. R. FULKERSON

Introduction. The problem discussed in this paper was formulated by T. Harris as follows:

We will see this algorithm today.

See also: "On the history of the transportation and maximum flow problems"
by Alexander Schrijver

Maximum flow

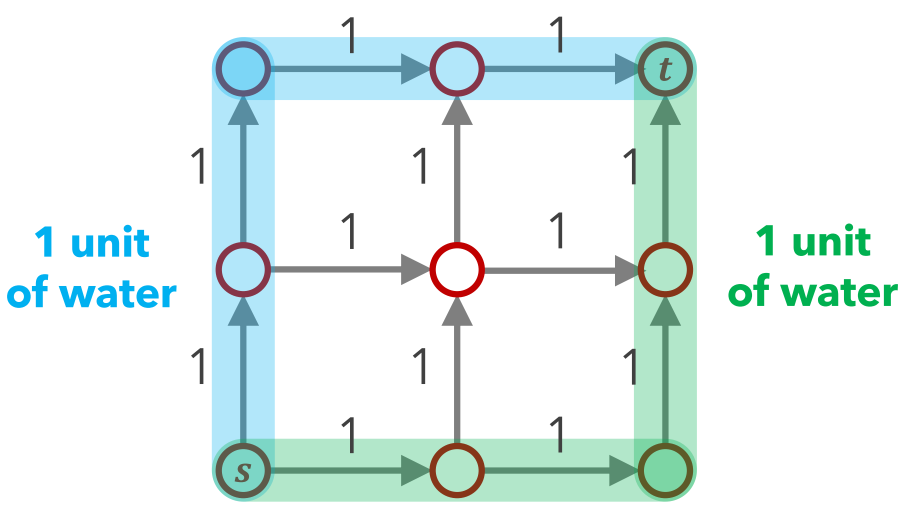
Input: 1. Directed graph $G = (V, E)$

2. One "source vertex" $s \in V$

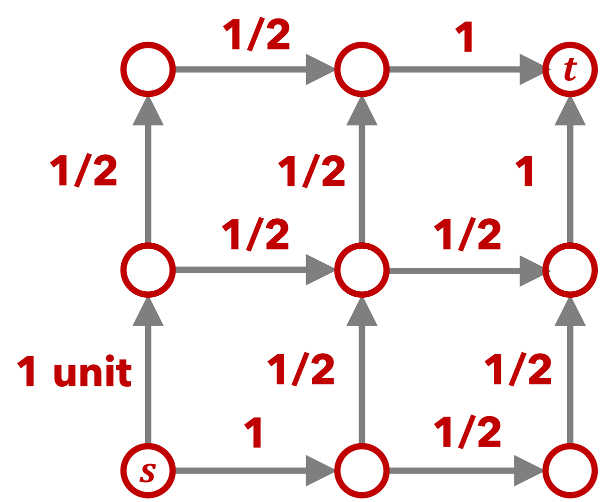
3. One "sink vertex" $t \in V$

4. For each edge $e \in E$, a "capacity" $c_e \in \mathbb{Z}^+$ (or $\mathbb{R}^+$)

Goal: Route the maximum amount of water from s to t



2 units of flow (water)

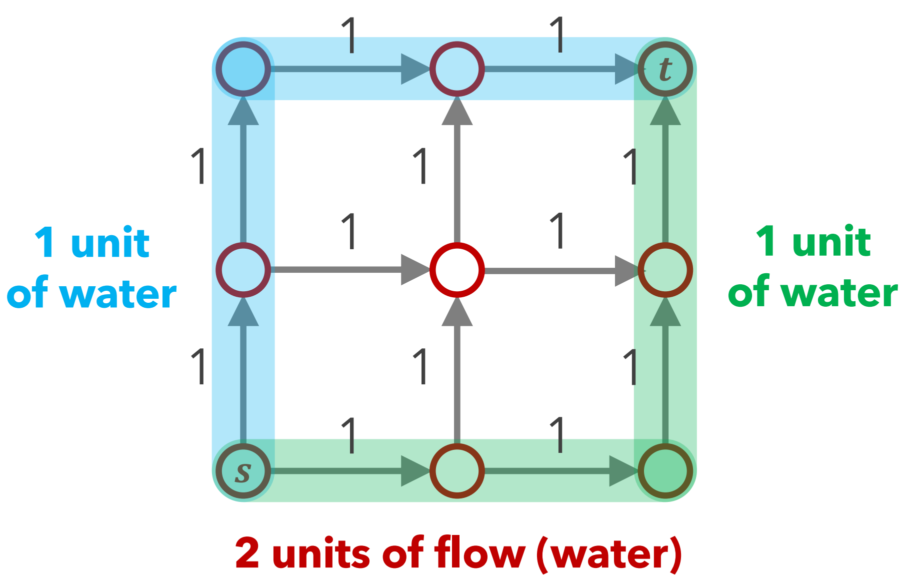


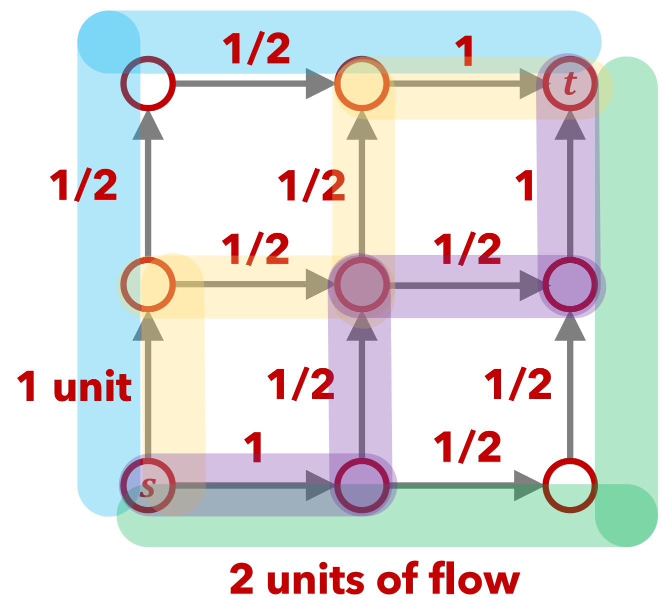
Maximum flow

Input: 1. Directed graph $G = (V, E)$

2. One "source vertex" $s \in V$
3. One "sink vertex" $t \in V$
4. For each edge $e \in E$, a "capacity" $c_e \in \mathbb{Z}^+$ (or $\mathbb{R}^+$)

Goal: Route the maximum amount of water from s to t



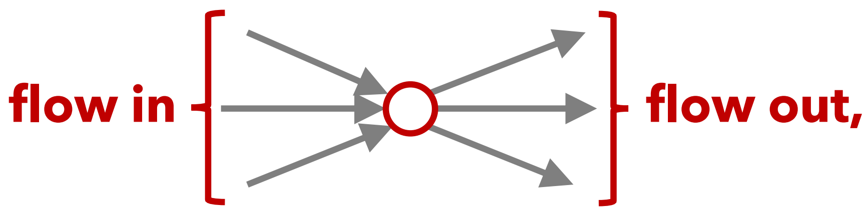


Def: A **flow** assigns a number f_e to each directed edge $e \in E$ such that

(Nonnegativity) $f_e \geq 0$

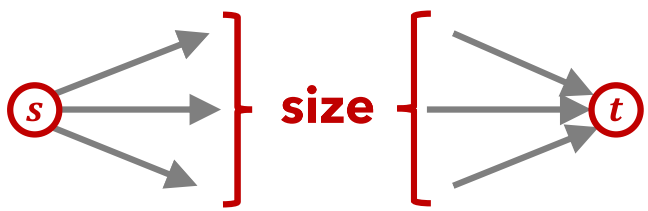
(Capacity) $f_e \leq c_e$

(Flow in = flow out) for each vertex $v \neq s, t$,



$$\sum_{u \to v} f_{u,v} = \sum_{v \to w} f_{v,w}$$

Def: The **size** of a flow f is the total quantity sent from s to t.



$$\mathbf{size}(f) = \sum_{s \to v} f_{s,v} = \sum_{v \to t} f_{v,t}$$

Maximum flow

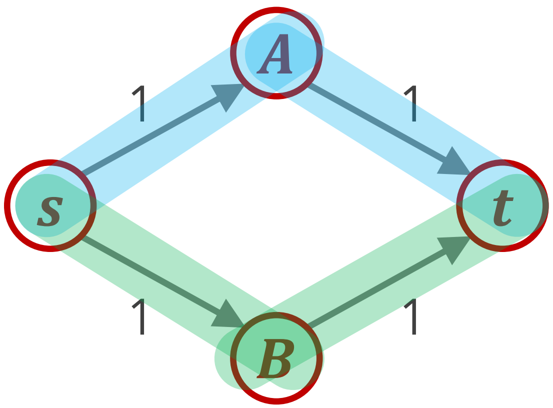
maximize $\mathbf{size}(f)$

s.t. $\{f_e\}$ is a flow

= a linear program!

Max flow algorithm: first try (Harris and Ross' "flooding" method)

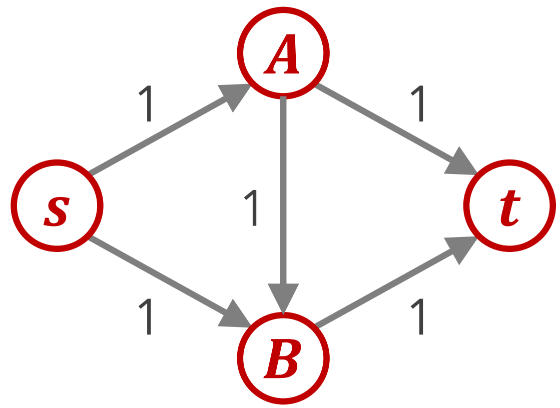
1. Find a path ***P*** from ***s*** to ***t*** which is not yet saturated
2. Send more flow along ***P***
3. Repeat

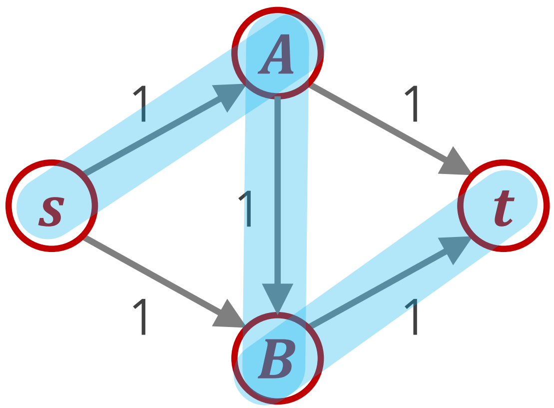


$s \to A \to t$: **1 unit**

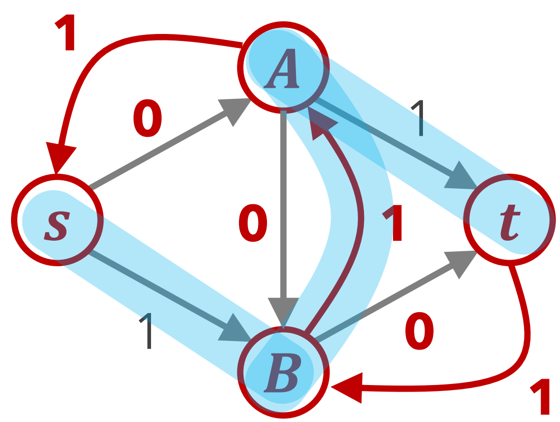
$s \to B \to t$: **1 unit**

total: **2 units**



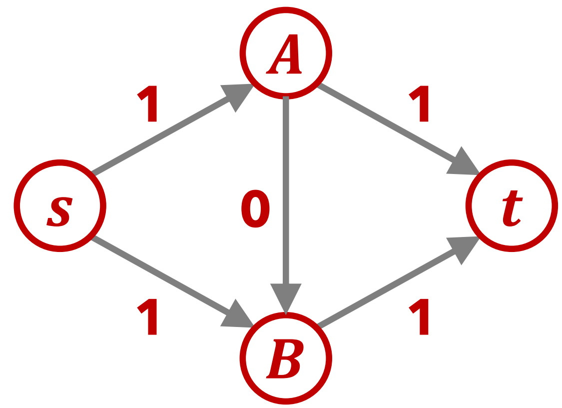


$s \rightarrow A \rightarrow B \rightarrow t$: **1 unit**



"residual graph"

$s \rightarrow B \rightarrow A \rightarrow t$: **1 unit**

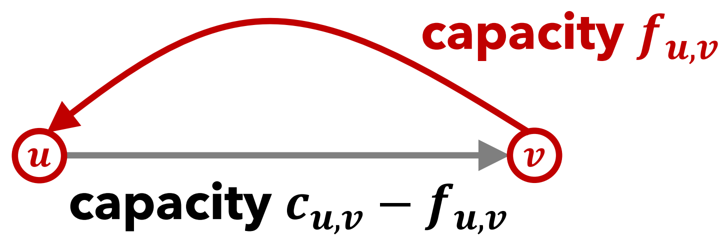


2 units total!

Def: Given a graph $\boldsymbol{G}$ and a flow $\boldsymbol{f}$ on $\boldsymbol{G}$, the residual graph $\boldsymbol{G_f}$ is as follows.

For all edges $(\boldsymbol{u}, \boldsymbol{v})$:

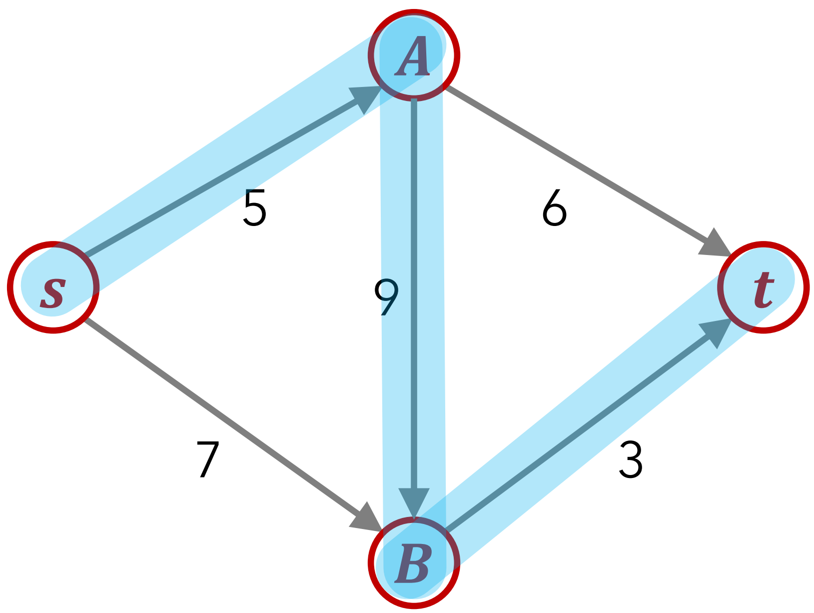
(in original graph)



(in residual graph)

Ford-Fulkerson algorithm

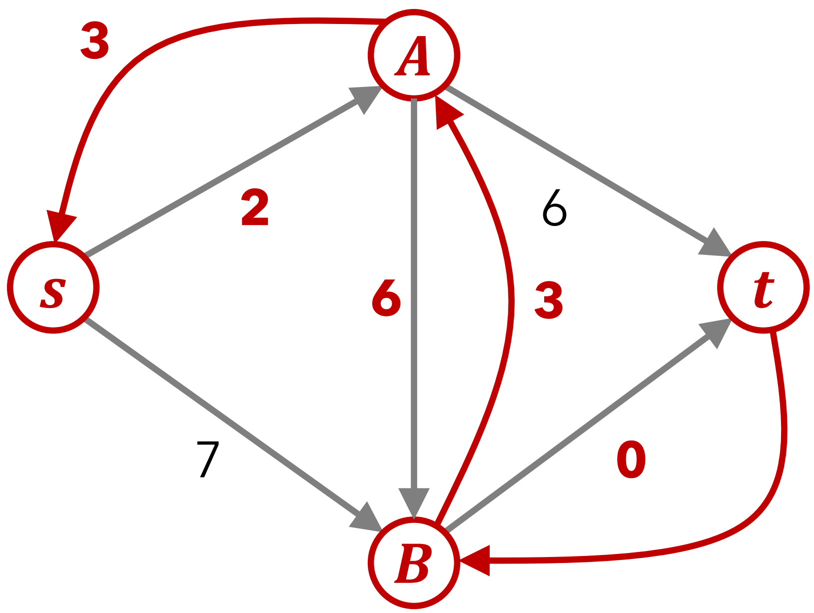
1. Find a path P from s to t **in the residual graph** which is not yet saturated
2. Send more flow along P = an **augmenting path**
3. Repeat



$s \to A \to B \to t$: **3 units**

Ford-Fulkerson algorithm

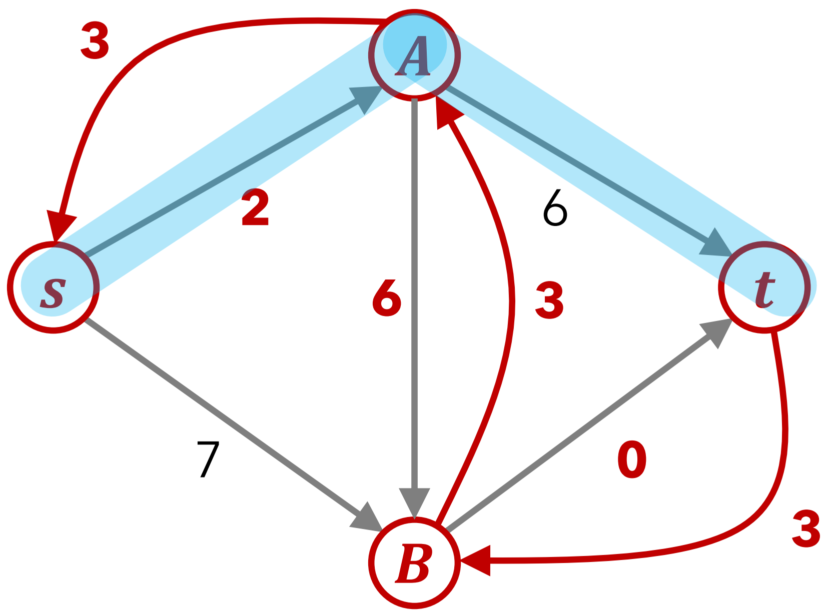
1. Find a path ***P*** from ***s*** to ***t*** **<u>in the residual graph</u>** which is not yet saturated
2. Send more flow along ***P*** = an **augmenting path**
3. Repeat



$s \to A \to B \to t$: **3 units**

Ford-Fulkerson algorithm

1. Find a path P from s to t **in the residual graph** which is not yet saturated
2. Send more flow along P = an **augmenting path**
3. Repeat

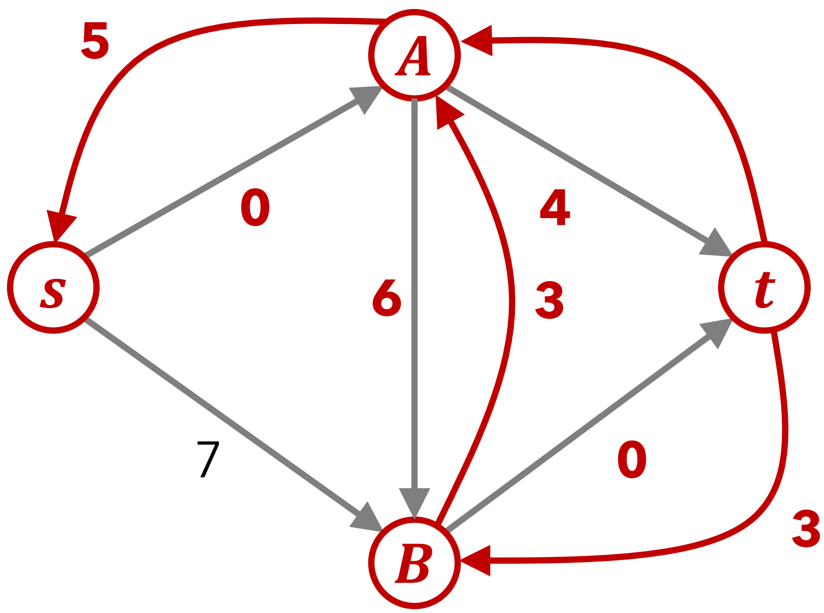


$s \to A \to B \to t$: **3 units**

$s \to A \to t$: **2 units**

Ford-Fulkerson algorithm

1. Find a path ***P*** from ***s*** to ***t*** **in the residual graph** which is not yet saturated
2. Send more flow along ***P*** = an **augmenting path**
3. Repeat

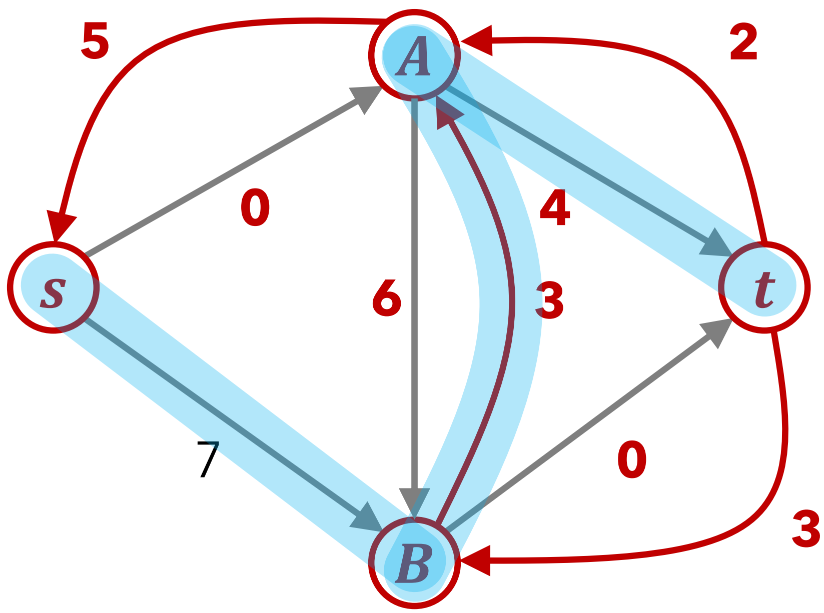


$s \to A \to B \to t$: **3 units**

$s \to A \to t$: **2 units**

Ford-Fulkerson algorithm

1. Find a path ***P*** from ***s*** to ***t*** **in the residual graph** which is not yet saturated
2. Send more flow along ***P*** = an **augmenting path**
3. Repeat



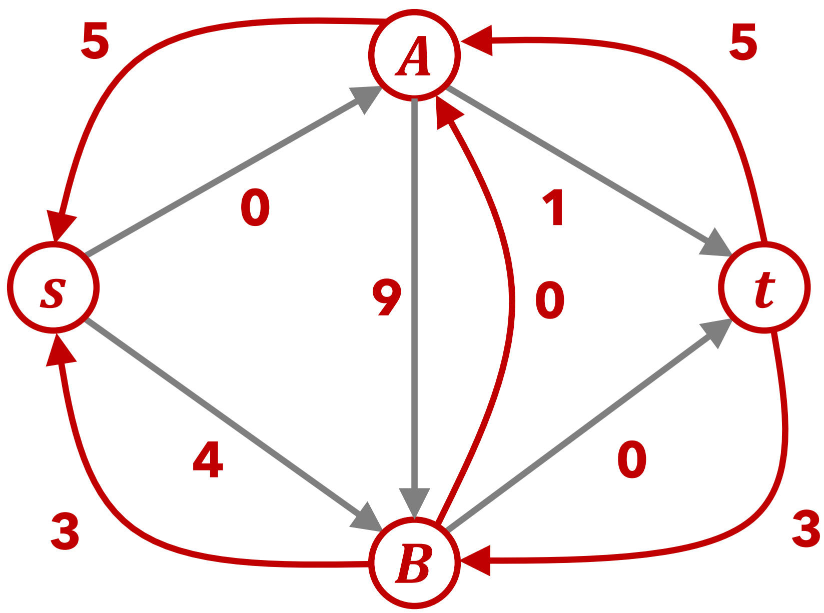
$s \to A \to B \to t$: **3 units**

$s \to A \to t$: **2 units**

$s \to B \to A \to t$: **3 units**

Ford-Fulkerson algorithm

1. Find a path ***P*** from ***s*** to ***t*** **in the residual graph** which is not yet saturated
2. Send more flow along ***P*** = an **augmenting path**
3. Repeat



$s \to A \to B \to t$: **3 units**

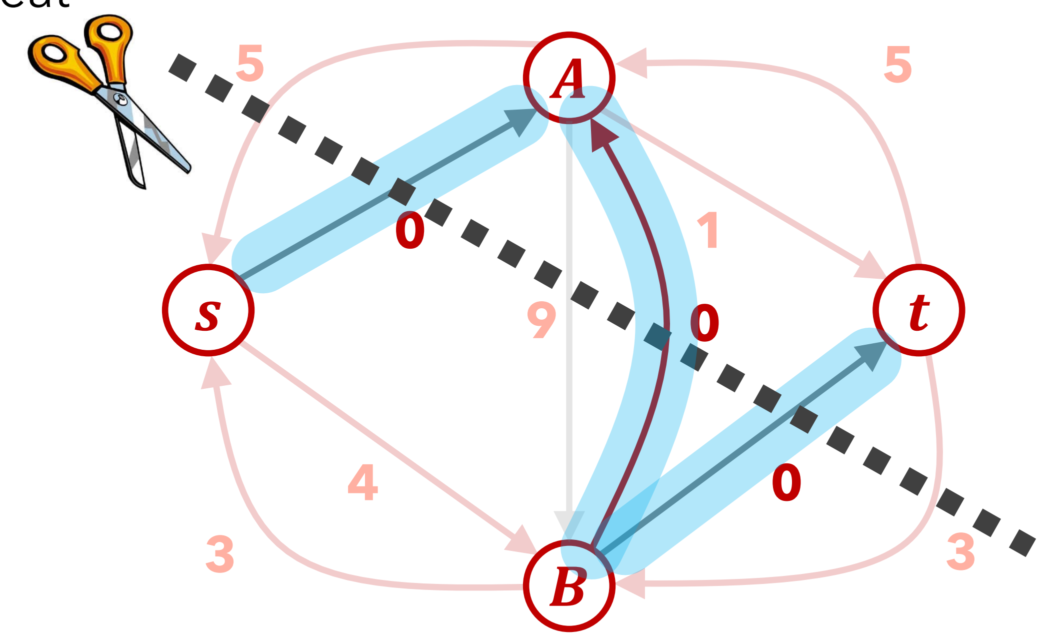
$s \to A \to t$: **2 units**

$s \to B \to A \to t$: **3 units**

total: **8 units**

Ford-Fulkerson algorithm

1. Find a path ***P*** from ***s*** to ***t*** **in the residual graph** which is not yet saturated
2. Send more flow along ***P*** = an **augmenting path**
3. Repeat



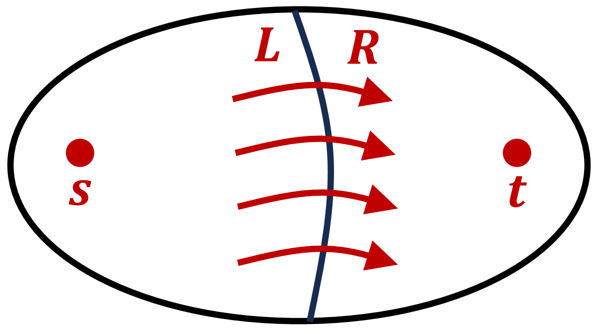
$s \to A \to B \to t$: **3 units**

$s \to A \to t$: **2 units**

$s \to B \to A \to t$: **3 units**

total: **8 units**

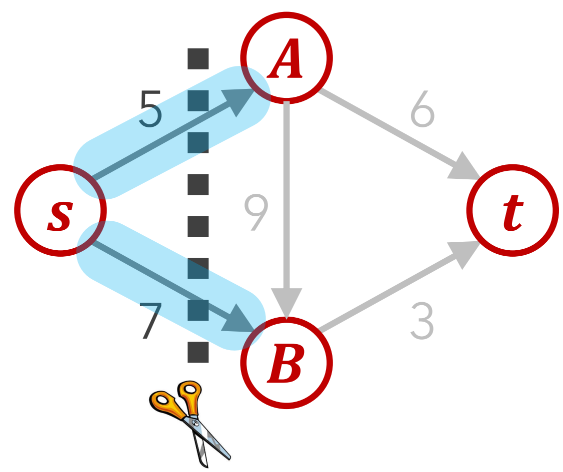
Def: An ***s*-*t* cut** is a partition $V = L \cup R$ of the vertices such that $s \in L$ and $t \in R$



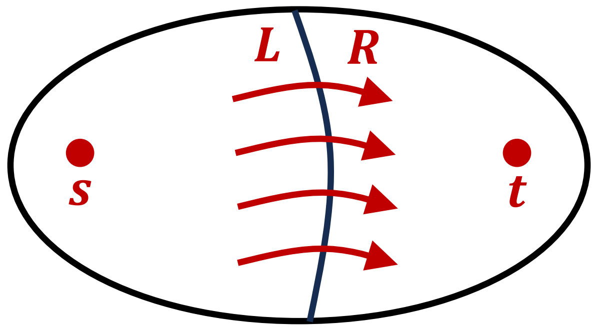
Def: The **capacity** of the **cut** is $\text{capacity}(L, R) = \sum_{\substack{u \to v \\ u \in L, v \in R}} c_{u,v}$

Thm: For any flow f and any cut (L, R),

$$\text{size}(f) \le \text{capacity}(L, R).$$



Def: An ***s-t* cut** is a partition $\boldsymbol{V = L \cup R}$ of the vertices such that $\boldsymbol{s \in L}$ and $\boldsymbol{t \in R}$



Def: The **capacity** of the **cut** is $\textbf{capacity}(L,R) = \sum_{\substack{u \to v \\ u \in L, v \in R}} c_{u,v}$

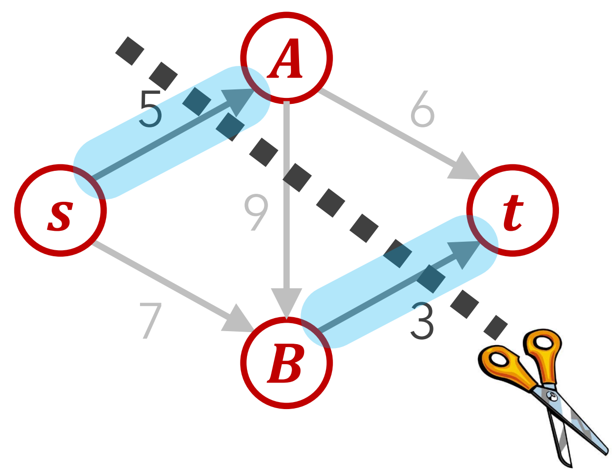
Thm: For any flow f and any cut (L,R),

$$\textbf{size}(f) \leq \textbf{capacity}(L,R).$$

Def: The **Min-cut** is the cut with **minimum** capacity.

Aka: Max-flow ≤ Min-cut

(Harris and Ross' "bottleneck")



Thm: Max-flow = Min-cut

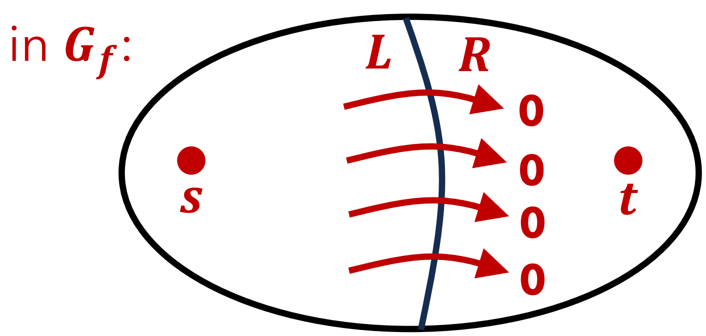
Pf: Only need to show "≥"

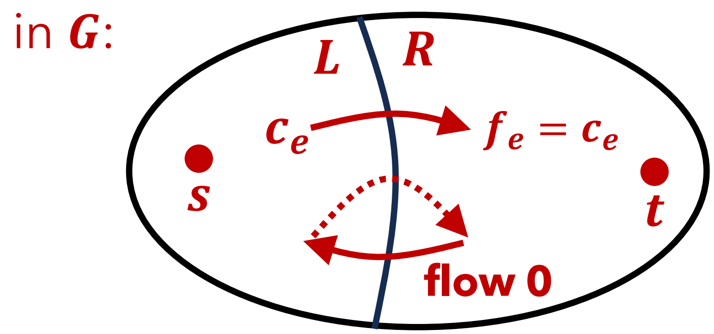
Run Ford-Fulkerson on G. Let f be the flow it outputs.

Then no $s \to t$ in residual graph G_f.

Set L = vertices reachable from s in G_f.

R = everything else.





Max-flow ≥ $\text{size}(f) = \text{capacity}(L, R)$ ≥ Min-cut. □

Thm: Ford-Fulkerson outputs a **maximum flow**.

Runtime $\approx$ # of augmenting paths $\leq \boldsymbol{U}$, where $\boldsymbol{U}$ = Max-flow

(× the time to find the paths) (in graphs of integer weights)

$\leq \mathbf{O}(m+n) \cdot \boldsymbol{U}$

Is this a good runtime?

Suppose each capacity $\boldsymbol{c_e}$ was $\leq \boldsymbol{C}$.

Then $\boldsymbol{U} \leq \boldsymbol{m} \cdot \boldsymbol{C}$.

Each $\boldsymbol{c_e}$ is a $\mathbf{log_2}(\boldsymbol{C})$–bit integer.

So $\boldsymbol{U}$ can be **exponential** in the input length!

This is a **pseudo-polynomial** algorithm

(it is polynomial in the **numerical value** of the input)

Recall: Knapsack

Runtime $\approx$ # of augmenting paths $\leq \boldsymbol{U}$, where $\boldsymbol{U}$ = Max-flow

(× the time to find the paths) (in graphs of integer weights)

$\leq \mathbf{O}(m+n) \cdot \boldsymbol{U}$

Surprise: If all the capacities are integral, then the Max-flow is integral.

(all the capacities/flows are integers)

Other algorithms:

Dinitz 1970/Edmonds-Karp 1972: Always pick the shortest augmenting path

Runs in time $\mathbf{O}(n\, m^2)$!

⋮

(many, many more)

⋮

Chen-Kyng-Liu-Peng-Gutenberg-Sachdeva 2022: $\mathbf{O}(m^{1+o(1)} \cdot \log(\boldsymbol{U}))$

(only 112 pages!)