GRAPH FLOWS

Algorithms and Data Structures 2
Exercise – 2023W

Martin Schobesberger, Markus Weninger, Markus Jäger,
Florian Beck, Achref Rihani

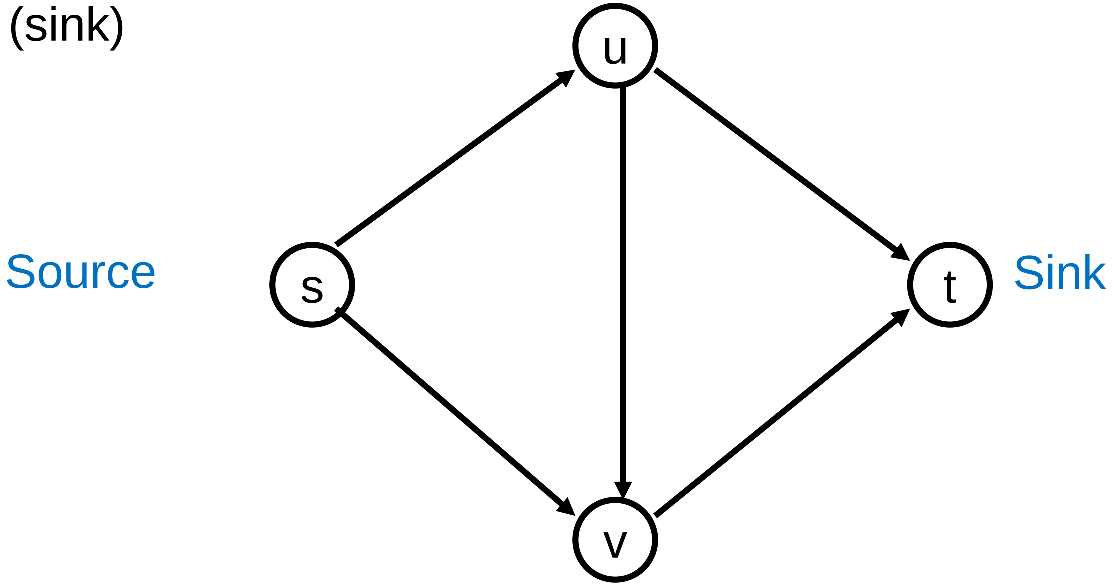
Institute of Pervasive Computing
Johannes Kepler University Linz
teaching@pervasive.jku.at

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

NETWORK FLOW DEFINITIONS

Flow graph

- directed graph with distinct vertices *s* (source) and *t* (sink)



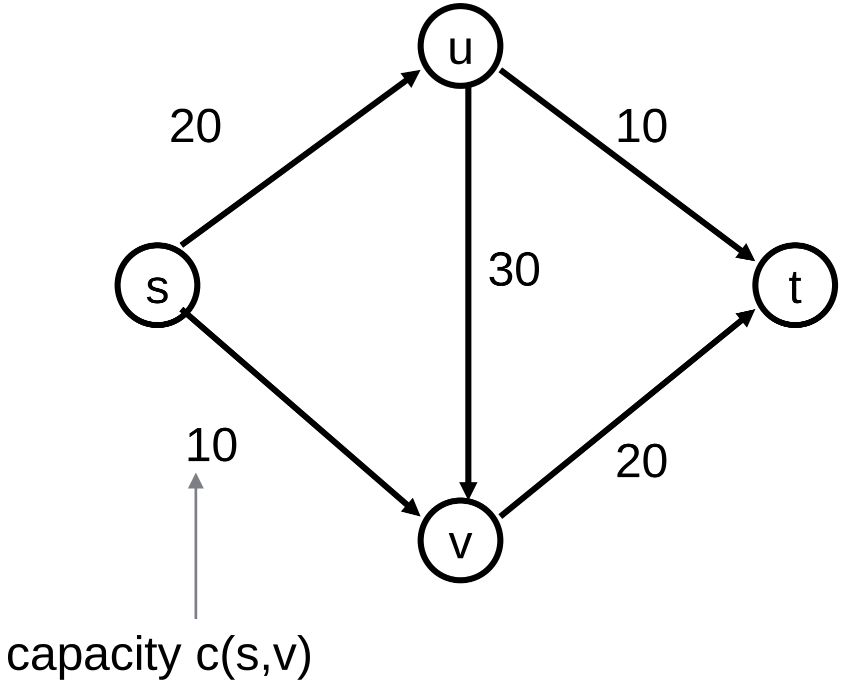
NETWORK FLOW DEFINITIONS

Flow graph

- directed graph with distinct vertices *s* (source) and *t* (sink)

Capacities

- weights on the edges: *c(u,v) >= 0*



NETWORK FLOW DEFINITIONS

Flow graph

- directed graph with distinct vertices *s* (source) and *t* (sink)

Capacities

- weights on the edges: *c(u,v) >= 0*

Flows

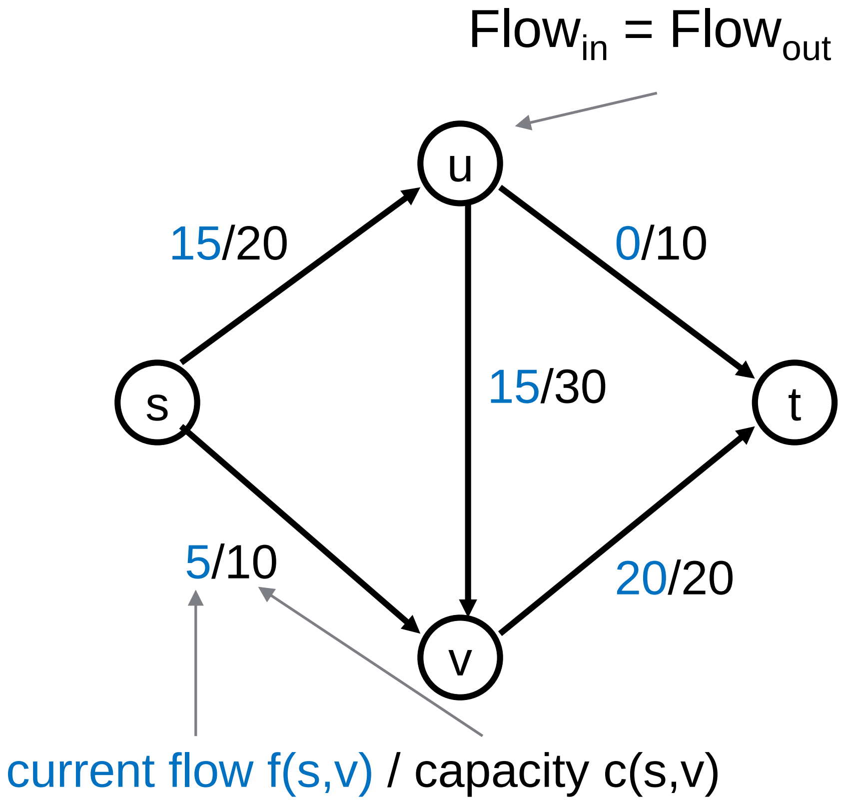
- assign flows *f(u,v)* to the edges such that
  - Capacity Condition: *0 <= f(u,v) <= c(u,v)*
  - Conservation Condition:

$$\sum_{u \in in(v)} f(u,v) = \sum_{w \in out(v)} f(v,w) \ \forall\, v \in V \setminus \{s,t\}$$

→ flow conservation: flow going into a vertex other than *s* and *t* equals the flow going out

$$\sum_{w \in out(s)} f(s,w) \ (= \sum_{u \in in(t)} f(u,t))$$

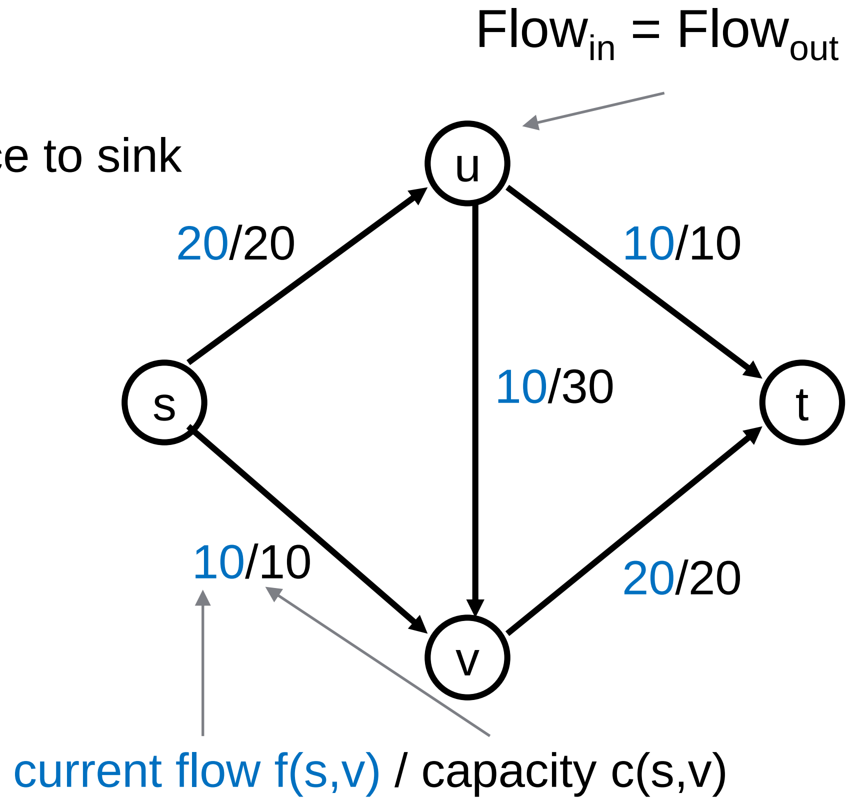
→ flow conservation: out flow of **source** must be the same as the in flow of **sink**



NETWORK FLOW DEFINITIONS

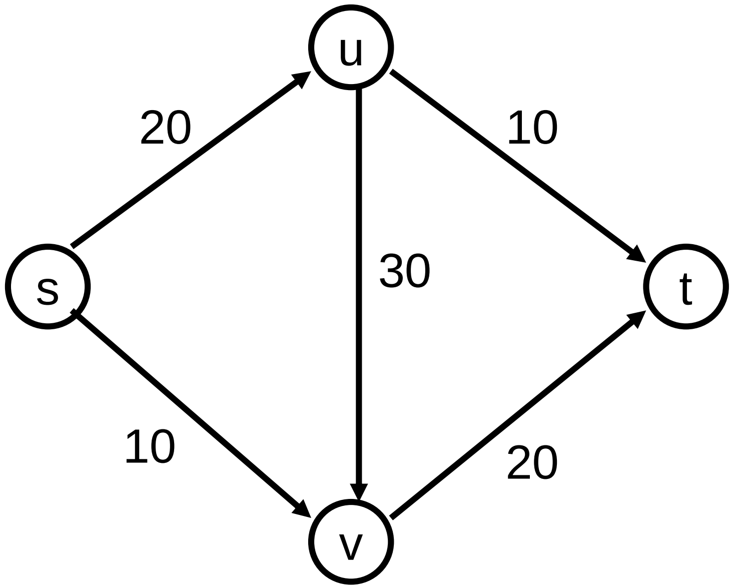
Problem

maximize $\sum_{w \in out(s)} f(s,w)$ $(= \sum_{u \in in(t)} f(u,t))$, the flow from source to sink



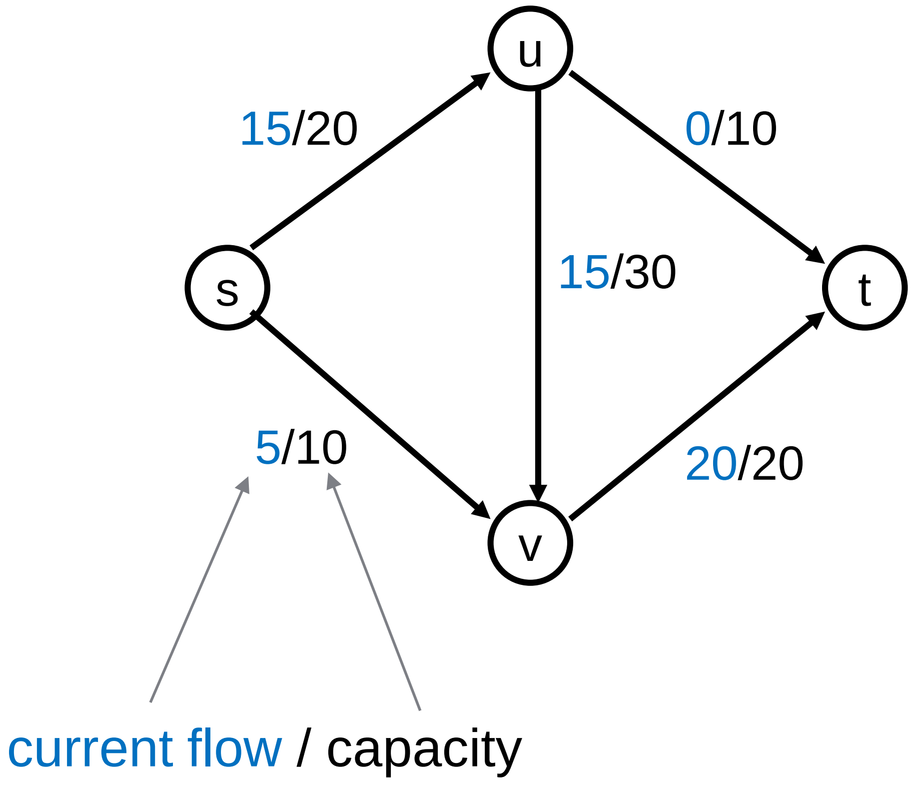
FLOW EXAMPLE 1/2

Network with capacities



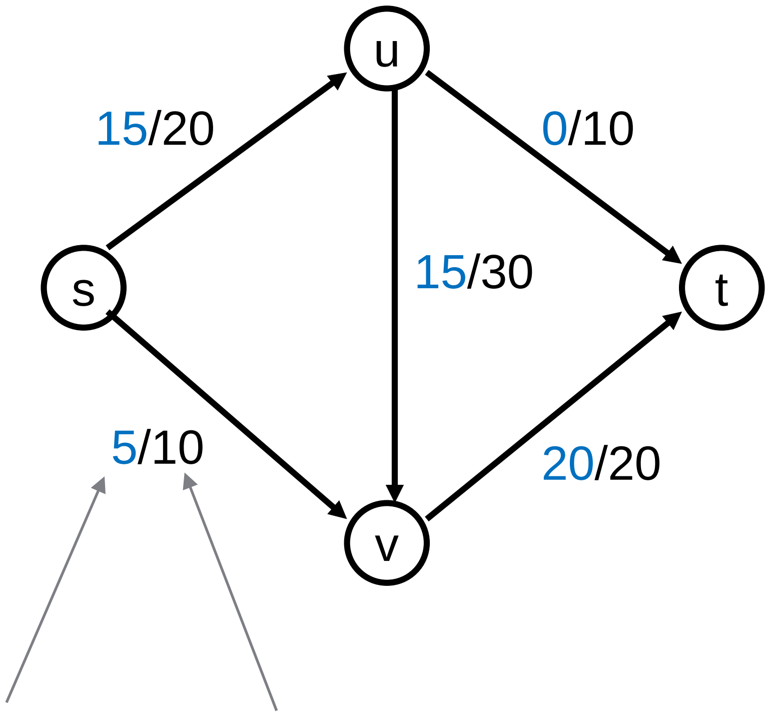
FLOW EXAMPLE 2/2

Two possible flows within the network

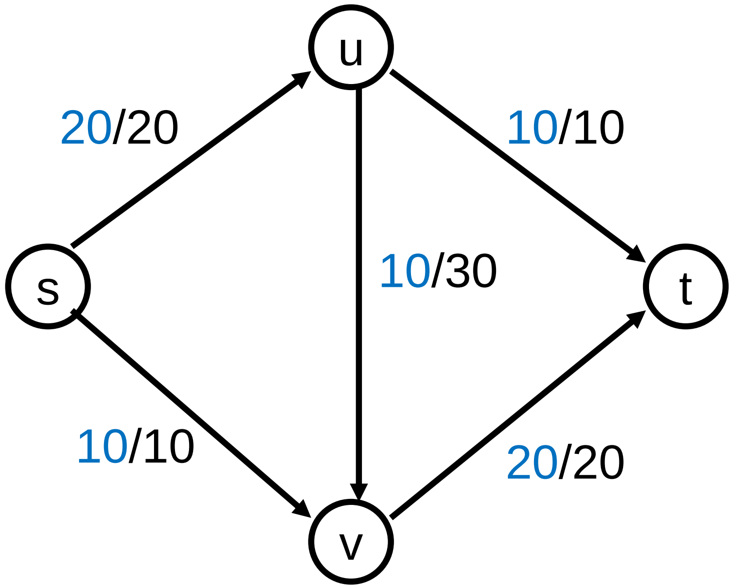


FLOW EXAMPLE 2/2

Two possible flows within the network



current flow / capacity



maximum flow = 30

RESIDUAL GRAPH

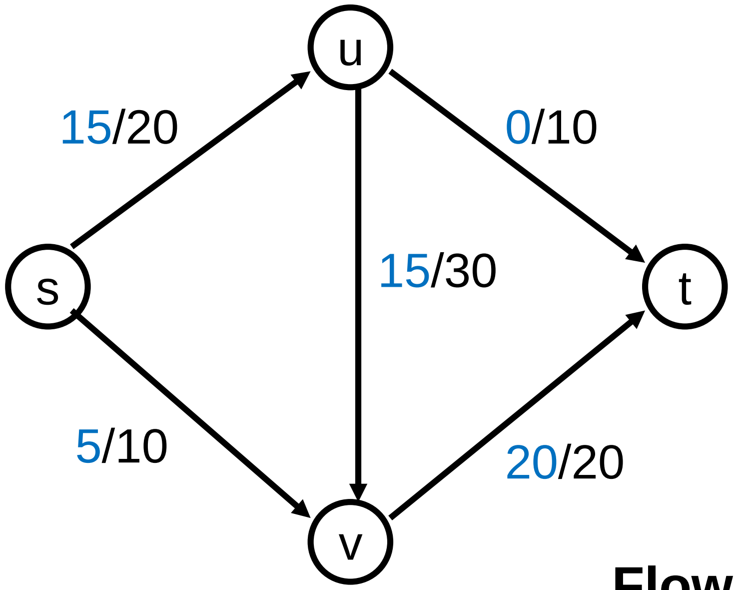
Flow graph showing the remaining capacity

Flow graph *G*

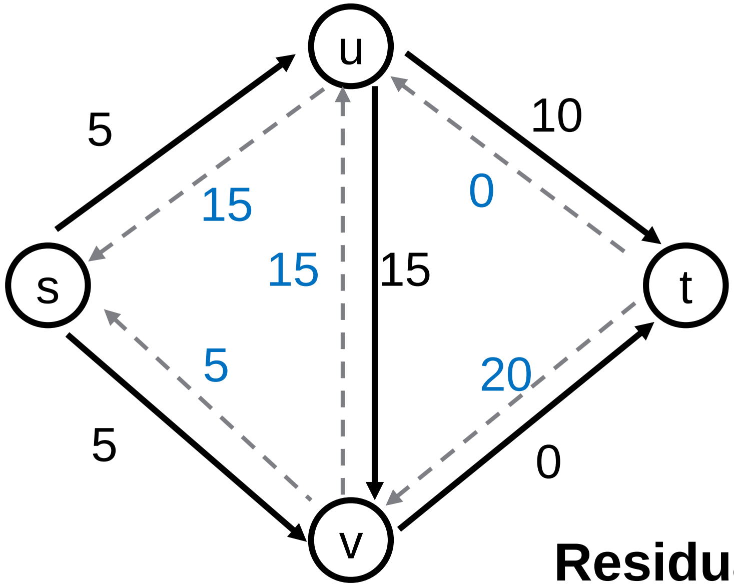
Residual graph G_R

- *G*: edge *e* from *u* to *v* with capacity *c* and flow *f*
- G_R: edge *e'* from *u* to *v* with remaining capacity $c - f$
- G_R: edge *e''* from *v* to *u* with flow *f*

Dotted Line = “Backedge” *e*
(Flow on backedges can be used to "give back" capacity to an edge)



Flow graph G

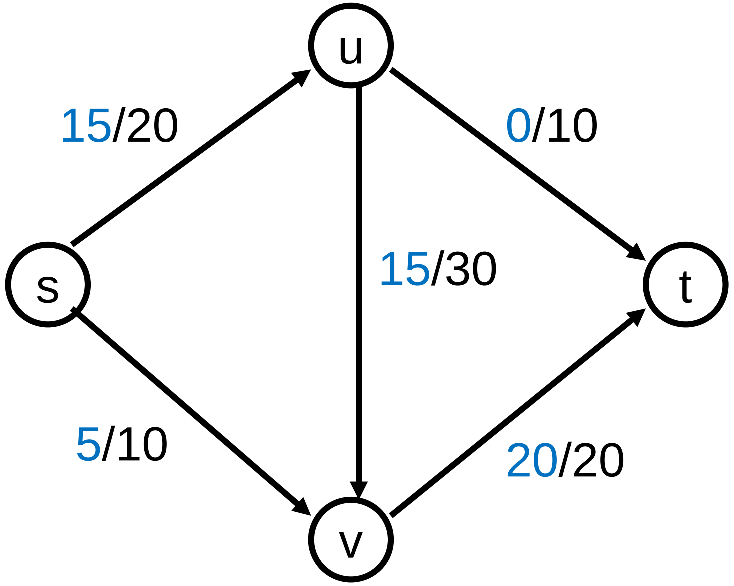


Residual graph G_R

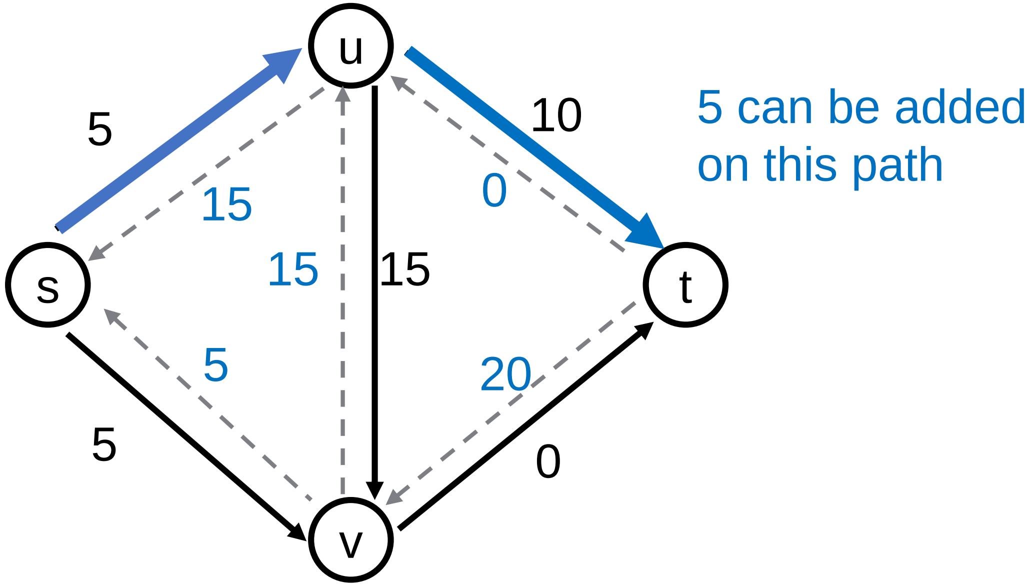
AUGMENTING PATH LEMMA

Let $P = v_1, v_2, \ldots, v_k$ be a path from s to t with **minimum** capacity $\boldsymbol{b}$ in the residual graph.

Then $\boldsymbol{b}$ units of flow can be added along the path P in the flow graph.



Flow graph

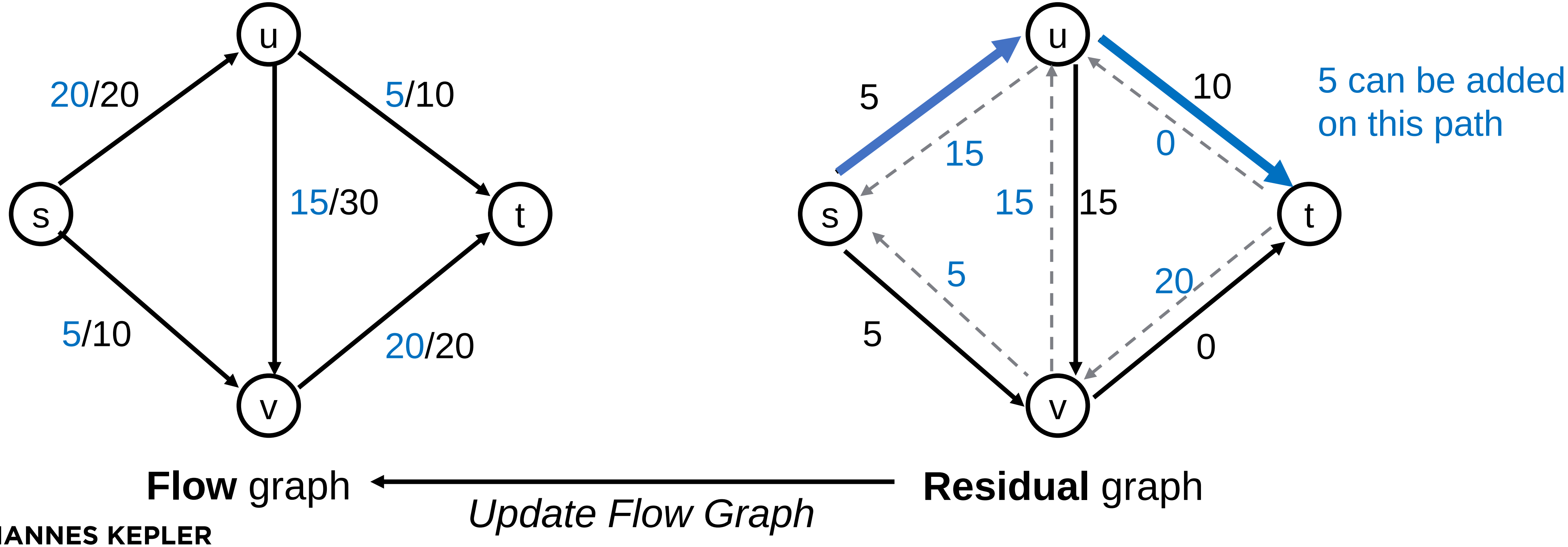


Residual graph

AUGMENTING PATH LEMMA

Let $P = v_1, v_2, \ldots, v_k$ be a path from s to t with **minimum** capacity ***b*** in the residual graph.

Then ***b*** units of flow can be added along the path P in the flow graph.



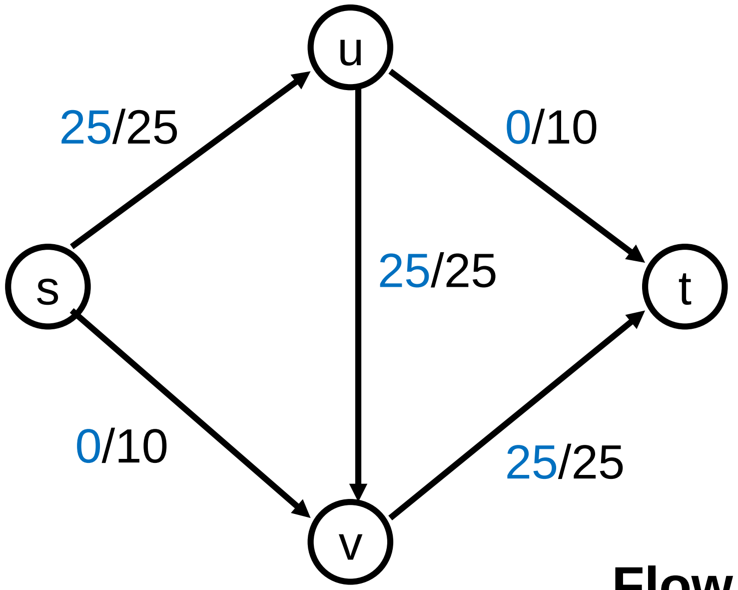
CONCEPT OF THE “BACKEDGE”

Dotted Line = “Backedge” *e*
(Flow on backedges can be used to "give back" capacity to an edge)

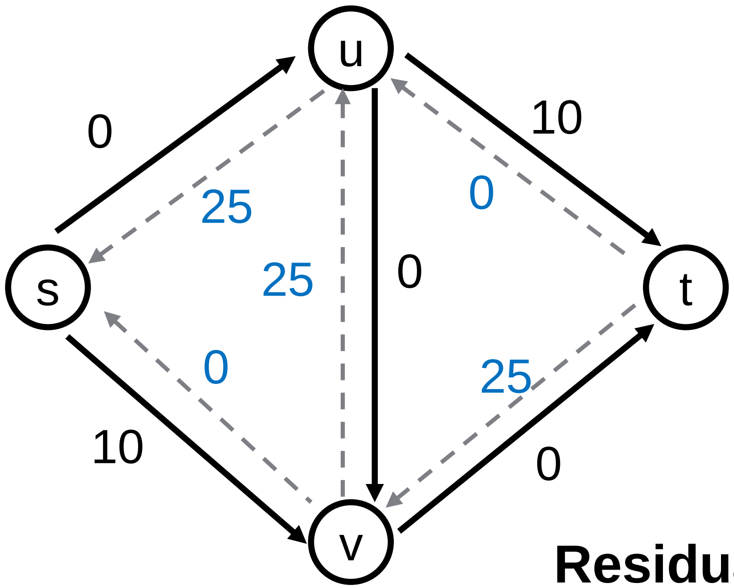
→ Edge in the residual graph which has capacity left

→ Can be used like a regular edge to find an augmenting path

→ In the **flow graph** the flow along this edge is **decreased**



Flow graph G



Residual graph G_R

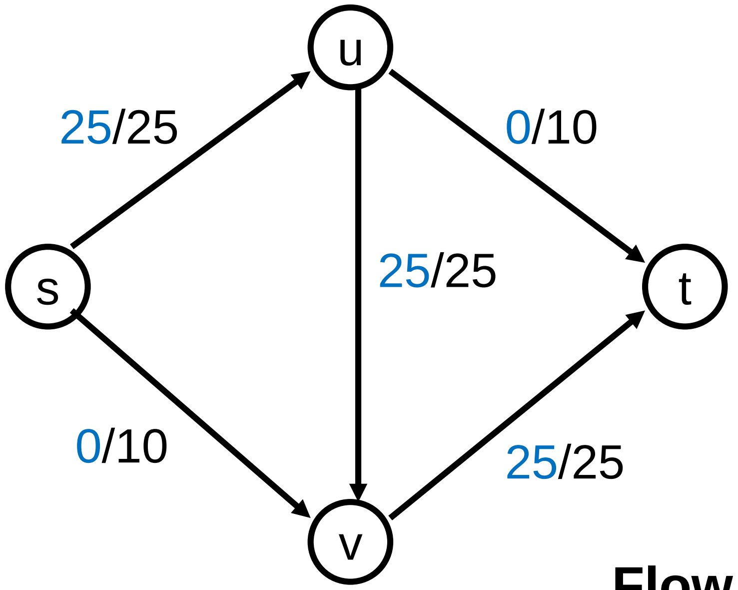
CONCEPT OF THE "BACKEDGE"

Dotted Line = "Backedge" *e*
(Flow on backedges can be used to "give back" capacity to an edge)

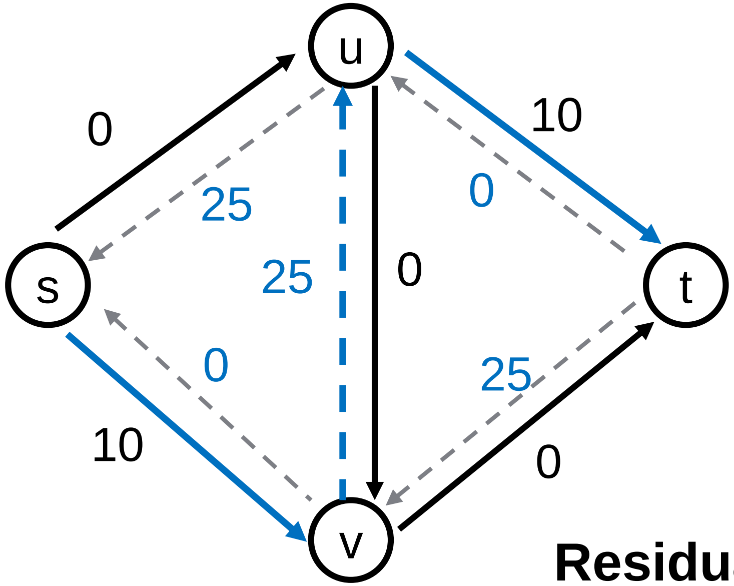
→ Edge in the residual graph which has capacity left

→ Can be used like a regular edge to find an augmenting path

→ In the **flow graph** the flow along this edge is **decreased**

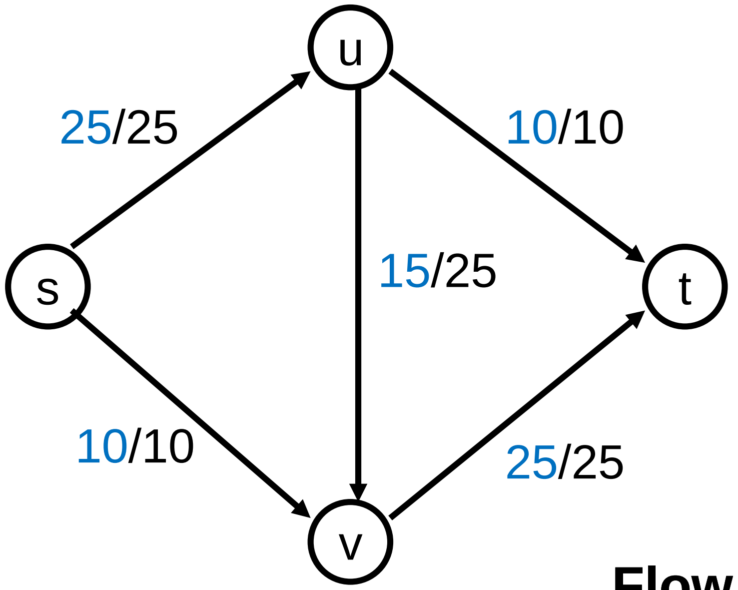


Flow graph G



Residual graph G_R

CONCEPT OF THE “BACKEDGE”



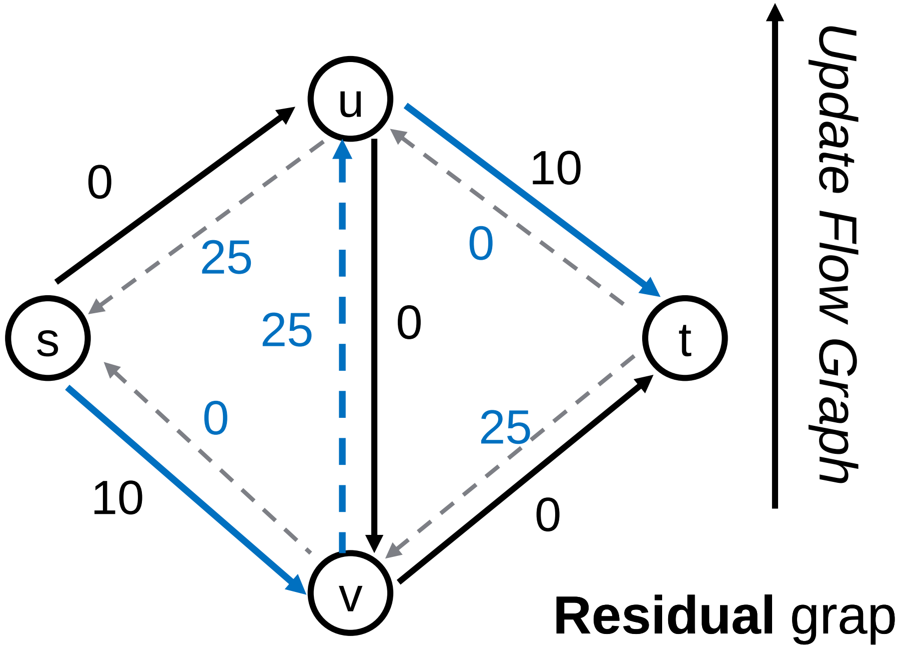
Flow graph G

Dotted Line = “Backedge” *e*
(Flow on backedges can be used to "give back" capacity to an edge)

→ Edge in the residual graph which has capacity left

→ Can be used like a regular edge to find an augmenting path

→ In the **flow graph** the flow along this edge is **decreased**



Residual graph G_R

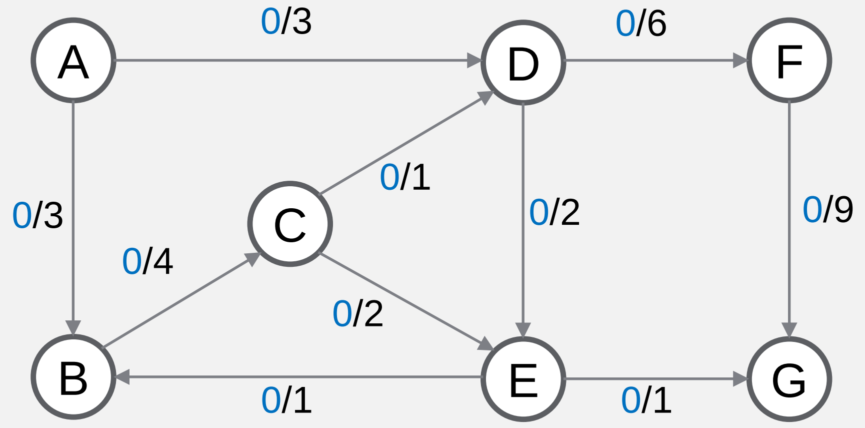
FORD-FULKERSON ALGORITHM (1956)

Pseudo code:

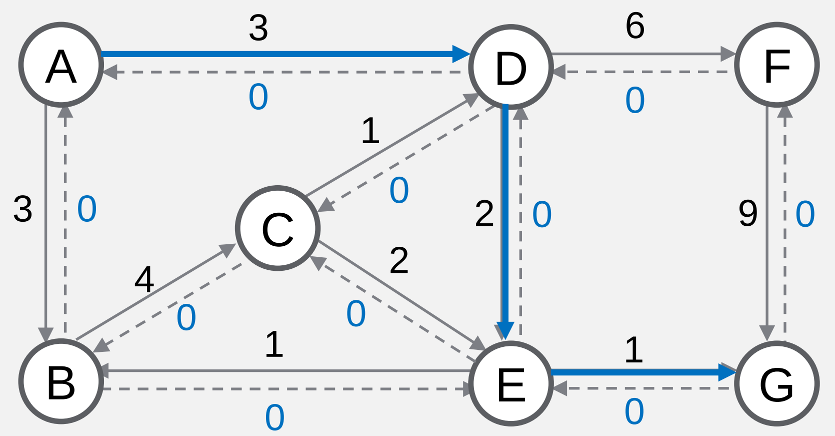
```
initialize all flows for graph G with 0
do
   construct residual graph G_R
   find a source-sink path P in G_R with capacity b > 0
   if P found: update the flow along P with b units in G
while P found
```

If the sum of the capacities of edges leaving source *S* is at most *C*, then the algorithm takes at most *C* iterations.

FORD-FULKERSON ALGORITHM STEP BY STEP

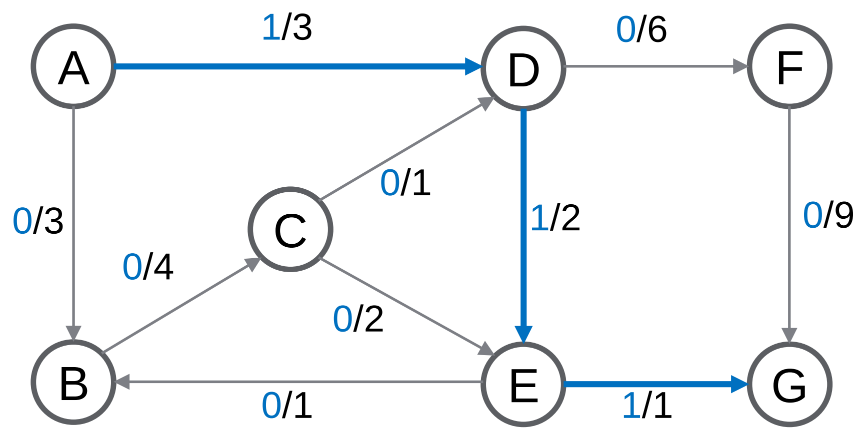


Initial graph with current flow = 0



Residual graph

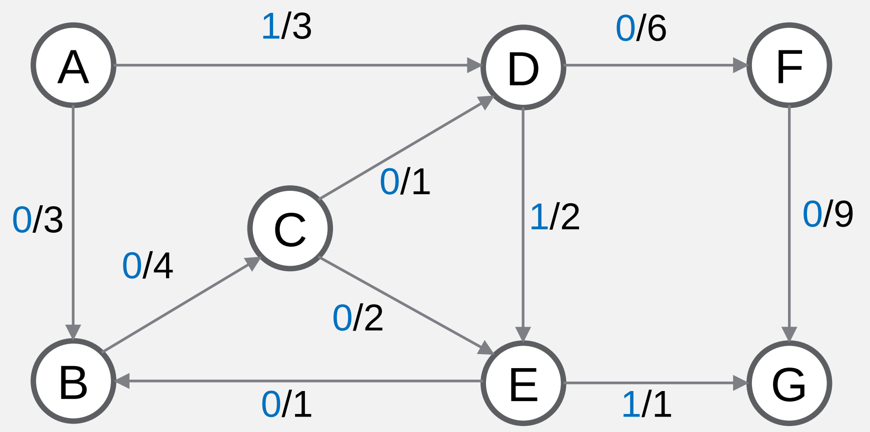
A = Source
G = Sink



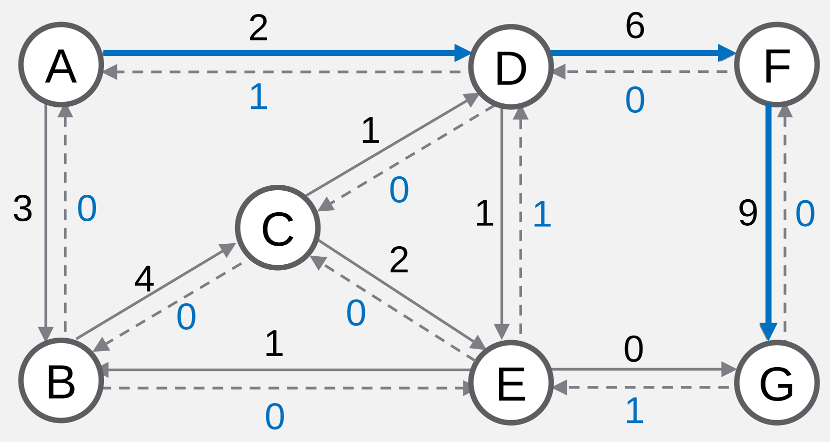
Updated graph

Flow along the path A->D->E->G increased by 1

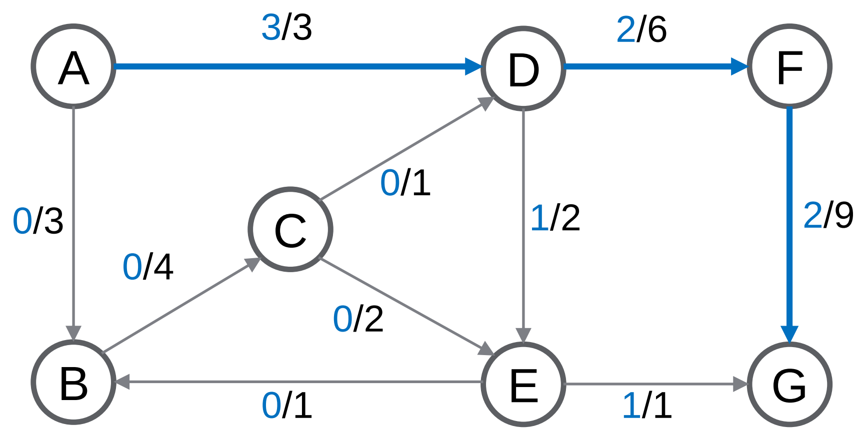
FORD-FULKERSON ALGORITHM STEP BY STEP



Intermediate graph with current flow = 1



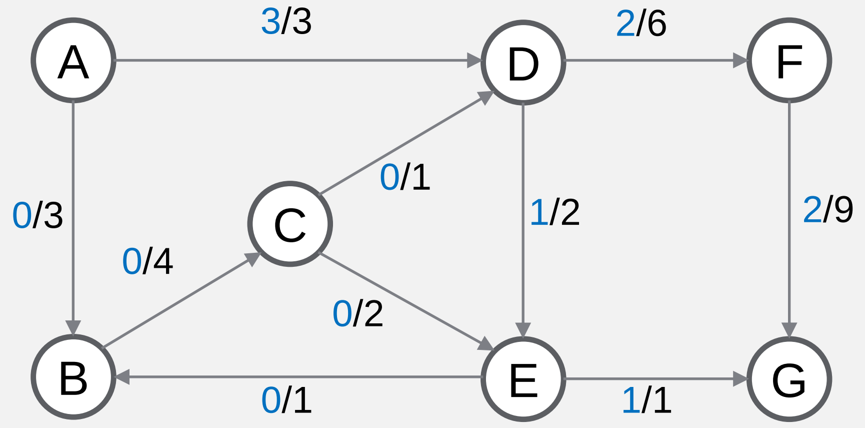
Residual graph



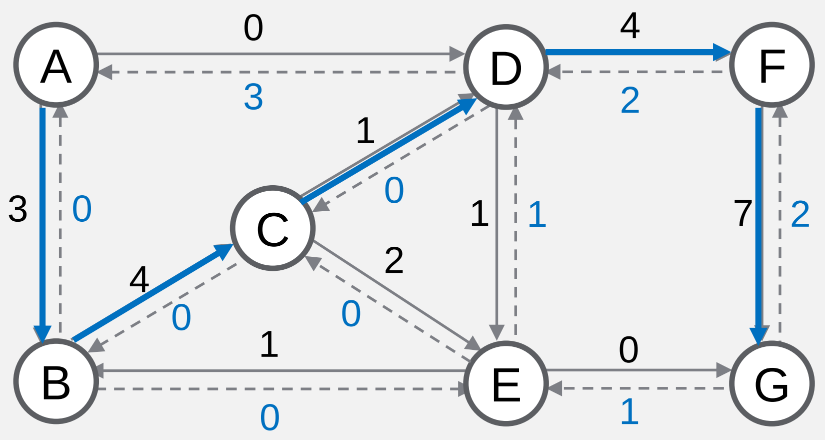
Updated graph

Flow along the path A->D->F->G increased by 2

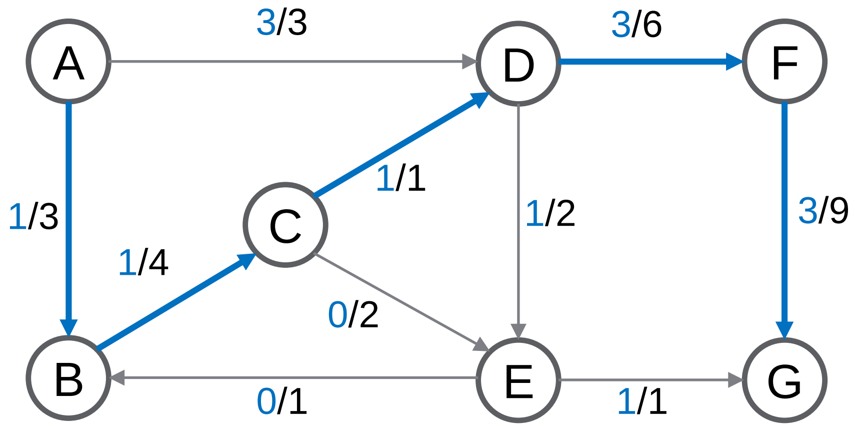
FORD-FULKERSON ALGORITHM STEP BY STEP



Intermediate graph with current flow = 3



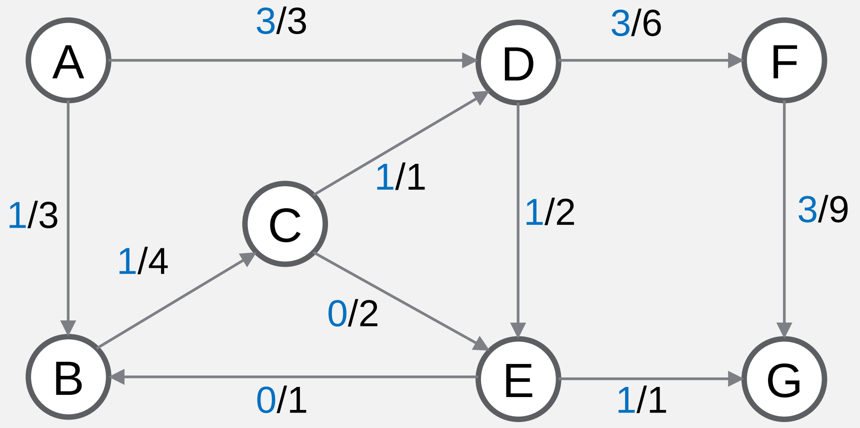
Residual graph



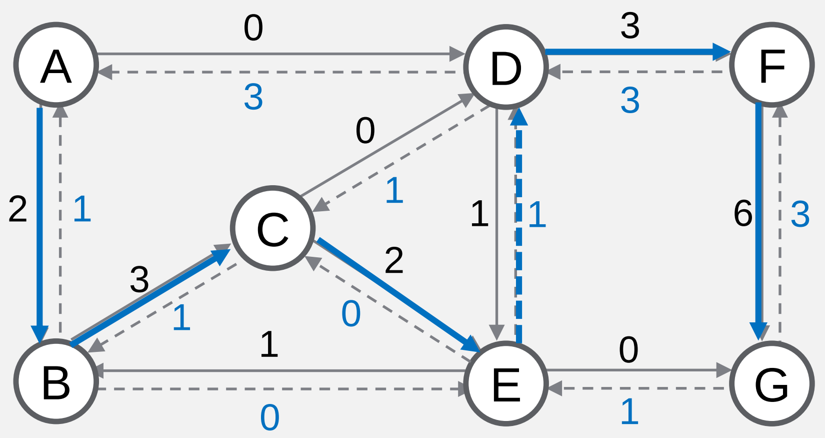
Updated graph

Flow along the path A->B->C->D->F->G increased by 1

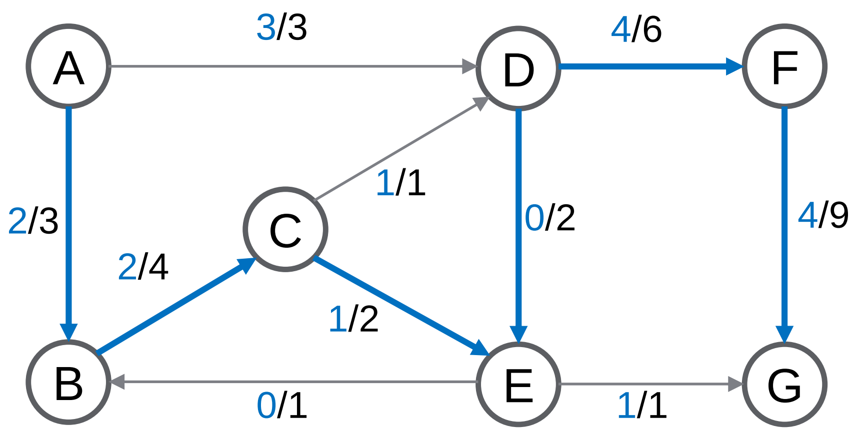
FORD-FULKERSON ALGORITHM STEP BY STEP



Intermediate graph with current flow = 4



Residual graph

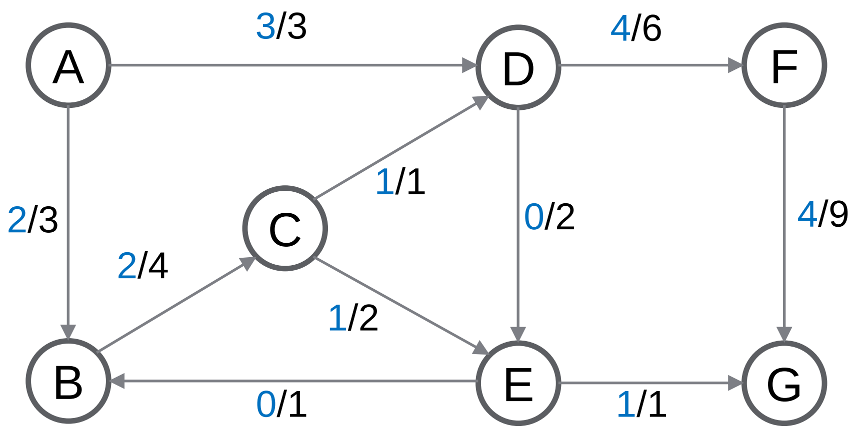


Updated graph

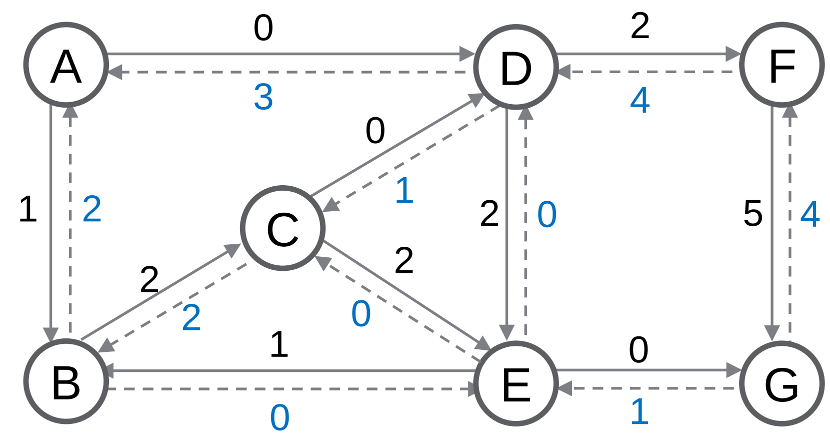
Flow along the path A->B->C->E->D->F->G increased by 1

<u>Attention</u>**: E->D is a "backedge" where flow is **decreased

FORD-FULKERSON ALGORITHM STEP BY STEP



Final graph with current/max flow = 5



Residual graph

ASSIGNMENT 05

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

GRAPH FLOWS

Algorithms and Data Structures 2
Exercise – 2023W

Martin Schobesberger, Markus Weninger, Markus Jäger,
Florian Beck, Achref Rihani

Institute of Pervasive Computing
Johannes Kepler University Linz
teaching@pervasive.jku.at

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at