

Network Flow Problem

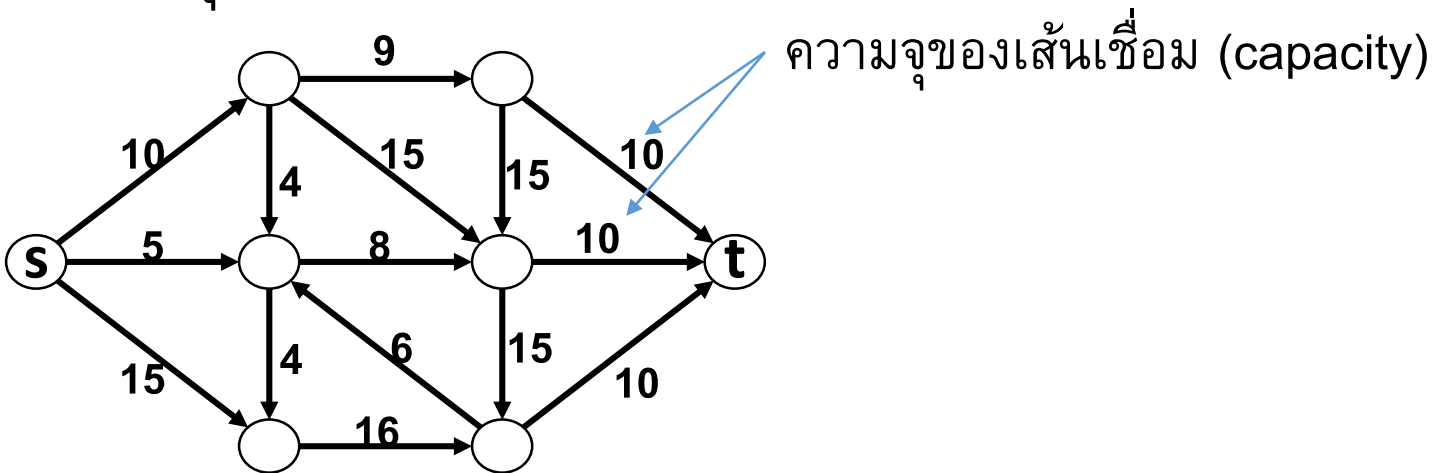
Network Flow Problem

ในทฤษฎีกราฟ flow network หรืออาจจะรู้จักในชื่อ transportation network คือกราฟแบบมีทิศทาง (Directed graph) ที่แต่ละเส้นเชื่อมจะมีความจุ (Capacity) และแต่ละเส้นเชื่อมจะรองรับ flow โดยปริมาณของ flow บนเส้นเชื่อมจะไม่สามารถเกินความจุของเส้นเชื่อม

Directed graph ของปัญหานี้จะถูกเรียกว่า network ซึ่ง network จะจำลองโมเดลของการจราจรบนถนน, การไหลของน้ำในท่อ, กระแสไฟฟ้าในวงจร เป็นต้น

Flow network

- เป็นการจำลองการไหลของวัตถุผ่านเส้นเชื่อม
- เป็นกราฟแบบมีทิศทาง Digraph $G = (V, E)$ ที่
 - มีโหนด source $s \in V$ และ sink $t \in V$ (ไม่มีเส้นเชื่อมคู่ขนาน ไม่มีเส้นเชื่อมเข้า s และไม่มีเส้นเชื่อมออกจาก t)
 - มีความจุ capacity $c(e)$ ของแต่ละเส้นเชื่อม $e \in E$ และเป็นค่าความจุที่ไม่ติดลบ

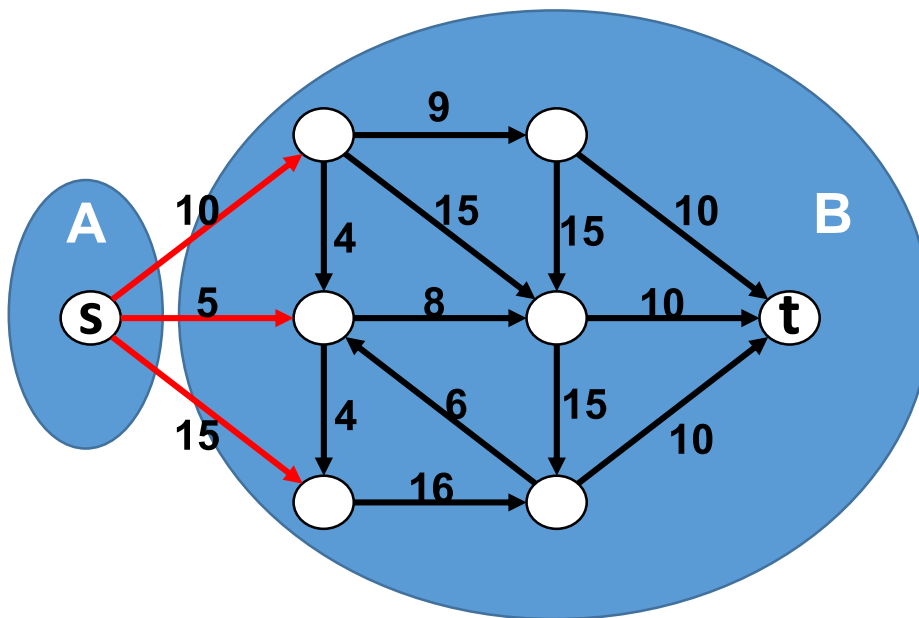


Minimum cut problem

นิยาม st-cut (cut) เป็นการแบ่งโหนดออกเป็นเซต (A,B) ที่มี $s \in A$ และ $t \in B$

นิยาม ความจุของ cut คือผลรวมความจุของเส้นเชื่อมที่ออกจาก A ไป B

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



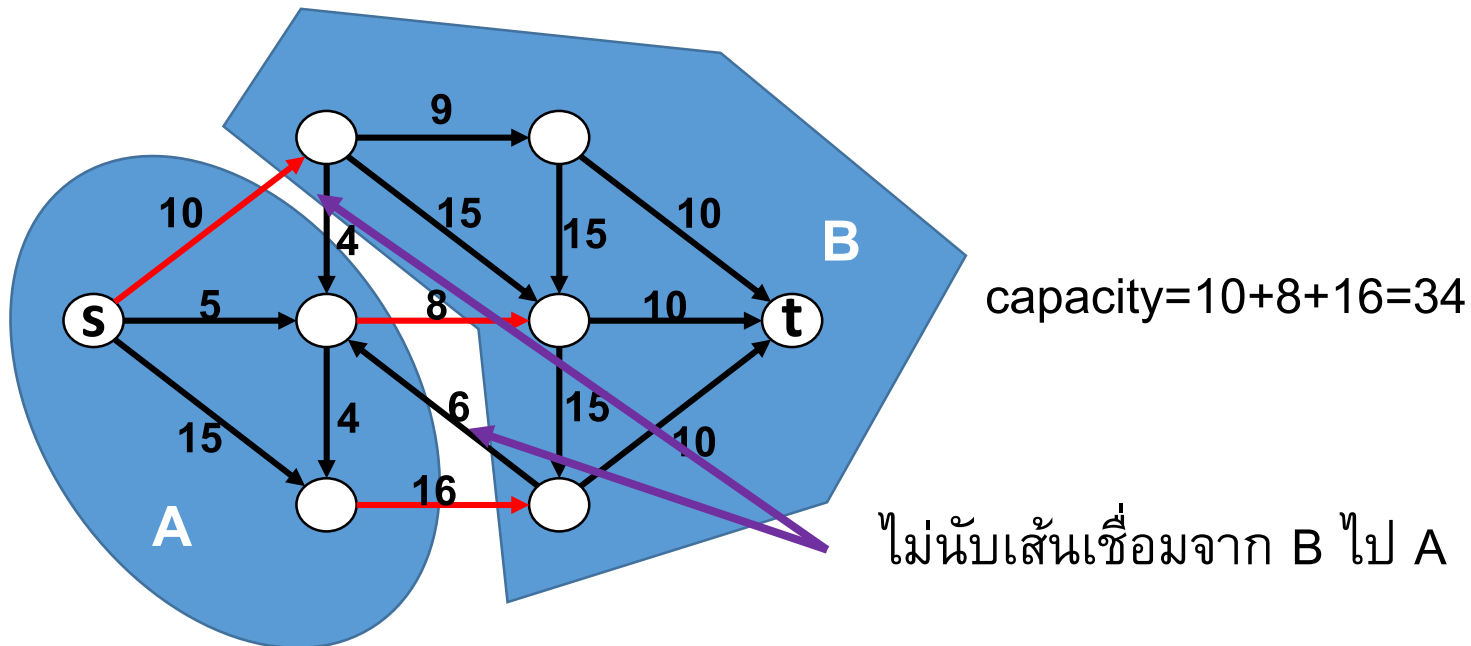
$$capacity = 10 + 5 + 15 = 30$$

Minimum cut problem

นิยาม st-cut (cut) เป็นการแบ่งโหนดออกเป็นเซต (A,B) ที่มี $s \in A$ และ $t \in B$

นิยาม ความจุของ cut คือผลรวมความจุของเส้นเชื่อมที่ออกจาก A ไป B

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



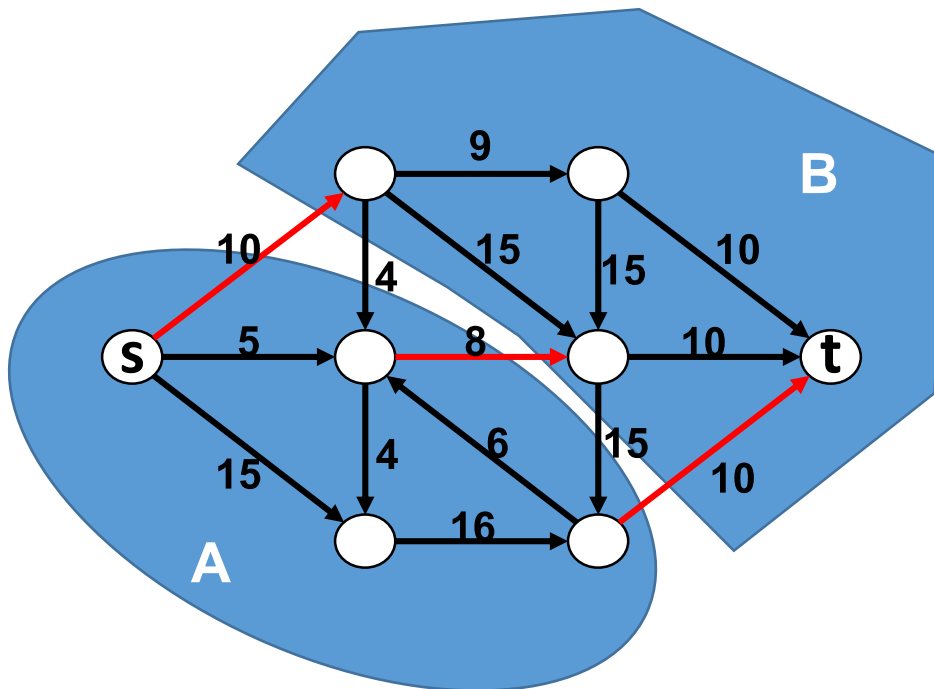
Minimum cut problem

นิยาม st-cut (cut) เป็นการแบ่งโหนดออกเป็นเซต (A,B) ที่มี $s \in A$ และ $t \in B$

นิยาม ความจุของ cut คือผลรวมความจุของเส้นเชื่อมที่ออกจาก A ไป B

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

Min-cut problem หา cut ที่มีความจุต่ำที่สุด



$$capacity = 10 + 8 + 10 = 28$$

Maximum flow problem

นิยาม st-flow (flow) f คือฟังก์ชัน $V \times V \rightarrow \mathbb{R}$ ที่สอดคล้องกับคุณสมบัติต่อไปนี้

- สำหรับแต่ละเส้นเชื่อม $e \in E$:

$$0 \leq f(e) \leq c(e)$$

[capacity]

- สำหรับแต่ละเส้นเชื่อม $e \in E$:

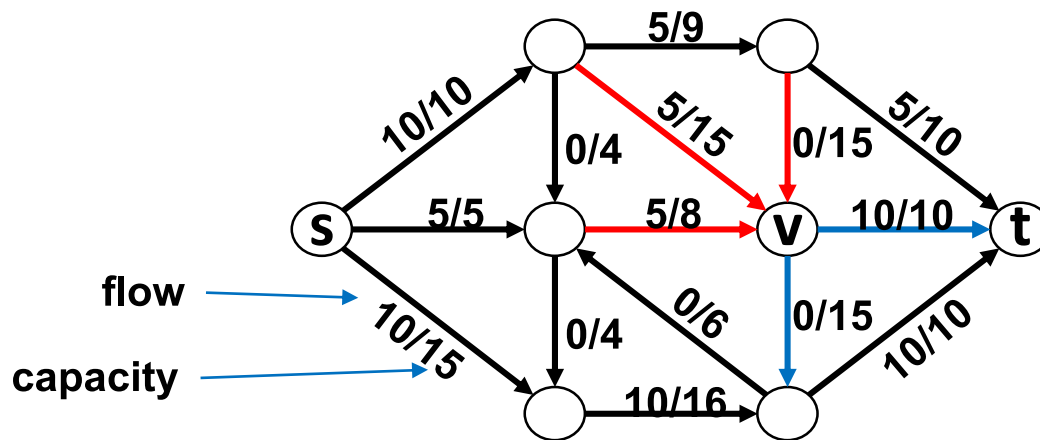
$$f(u, v) = -f(v, u)$$

[skew symmetry]

- สำหรับแต่ละโหนด $v \in V - \{s, t\}$:

$$\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

[flow conservation]



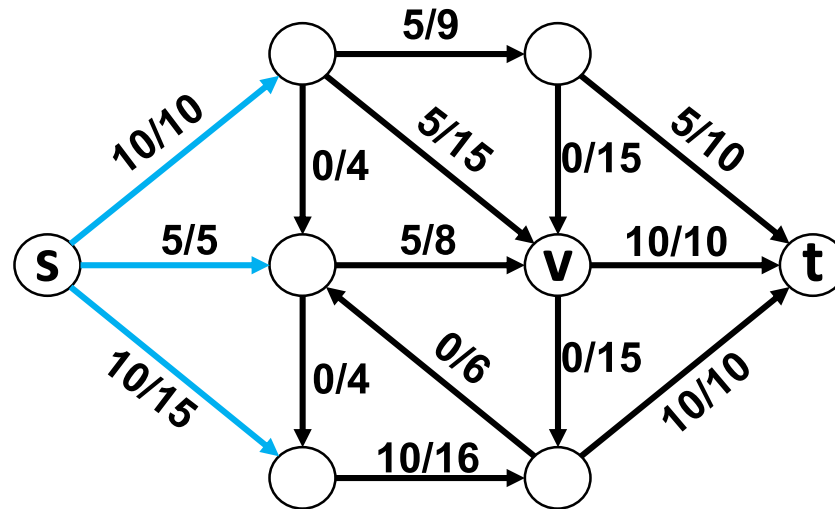
inflow at v = 5+5+0 = 10

outflow at v = 10+0 = 10

Maximum flow problem

นิยาม ค่าของ flow f คือ $val(f) = \sum_{e \text{ out of } s} f(e)$

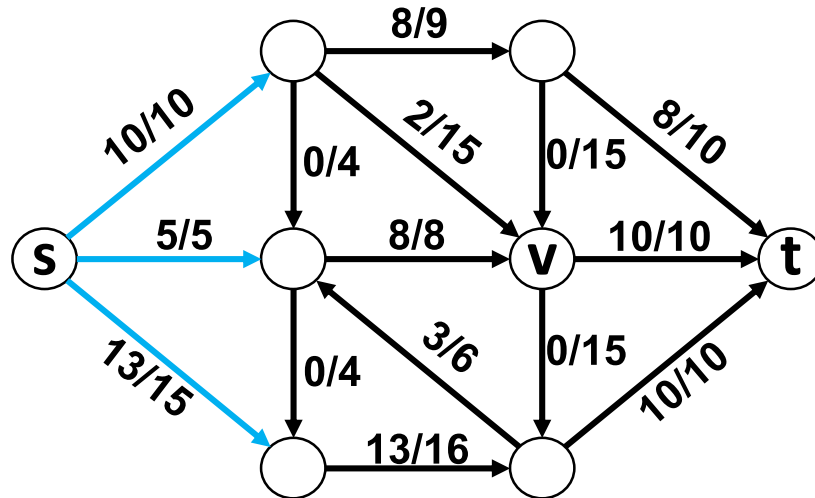
value=5+10+10=25



Maximum flow problem

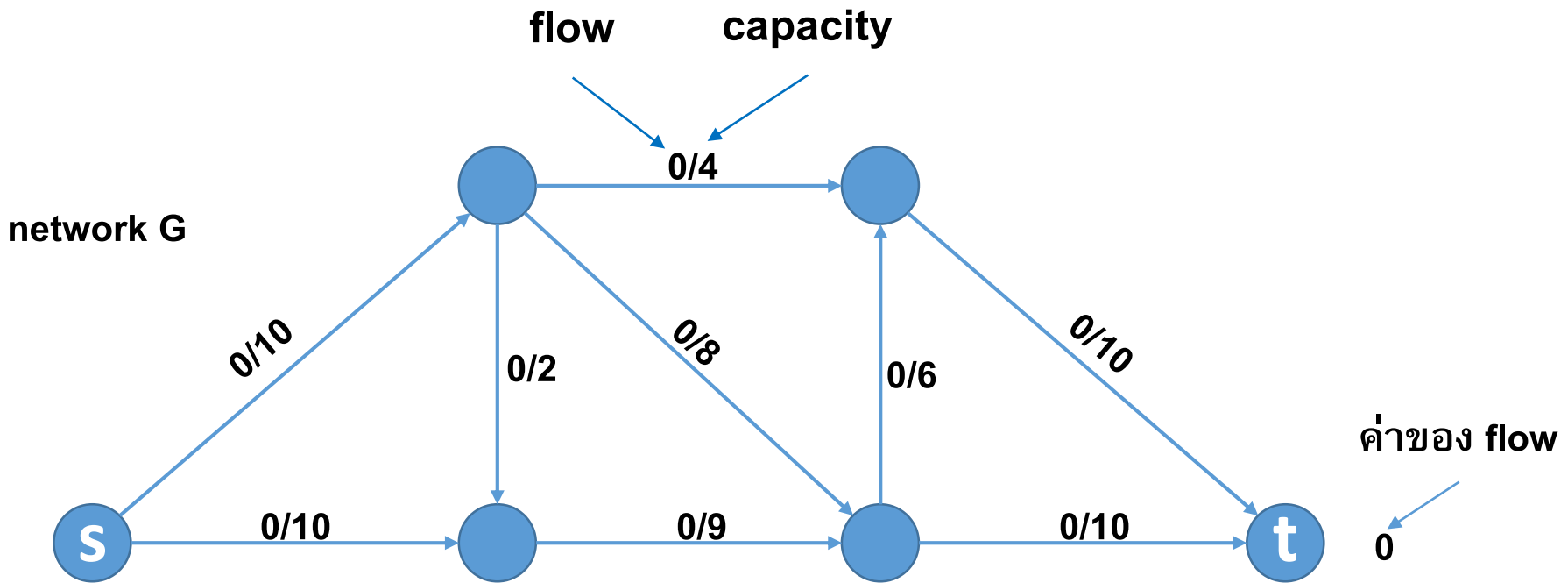
Max-flow problem หา flow ที่มีค่ามากที่สุด

$$\text{value} = 5 + 10 + 13 = 28$$



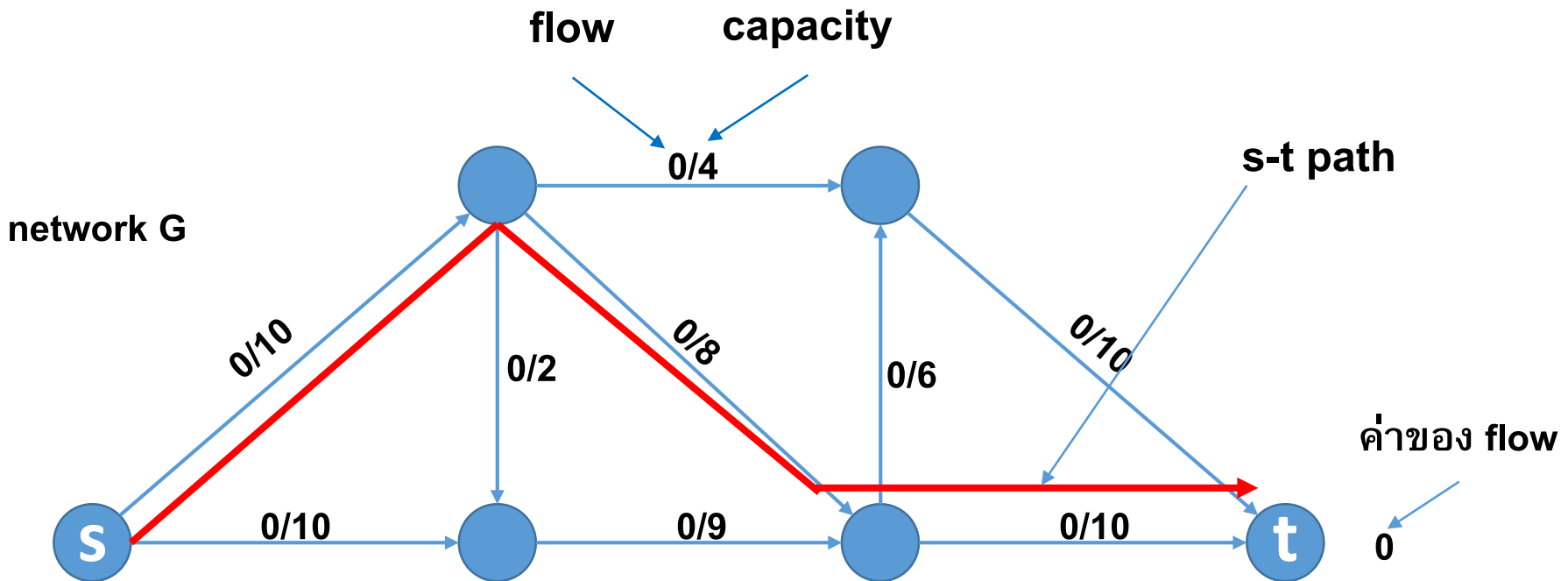
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



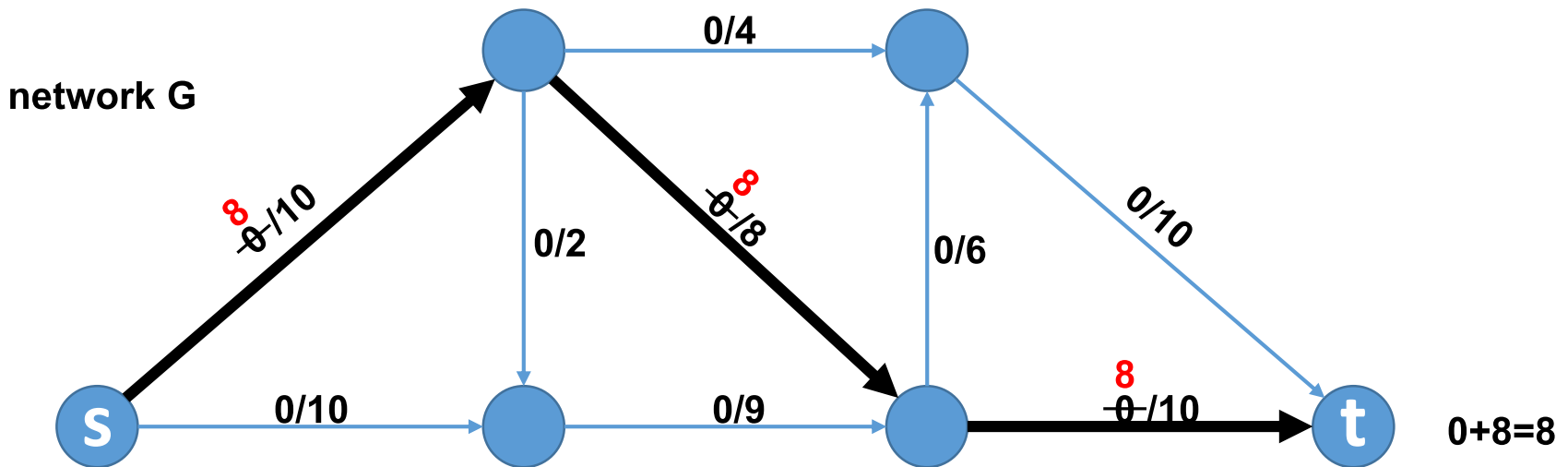
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



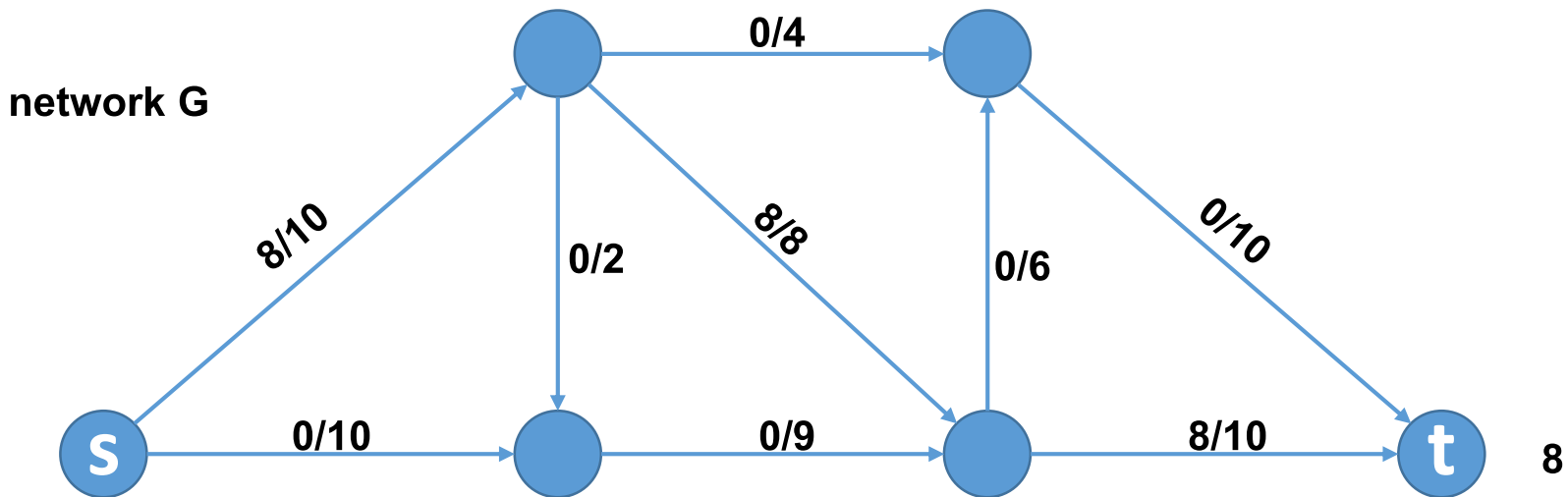
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



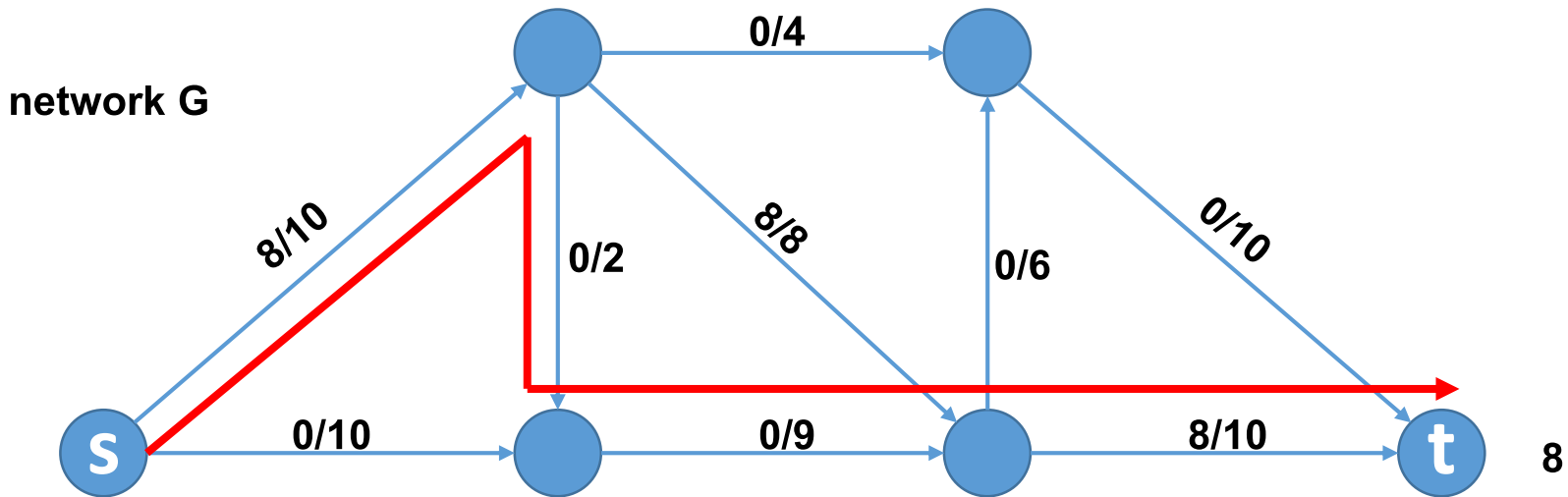
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



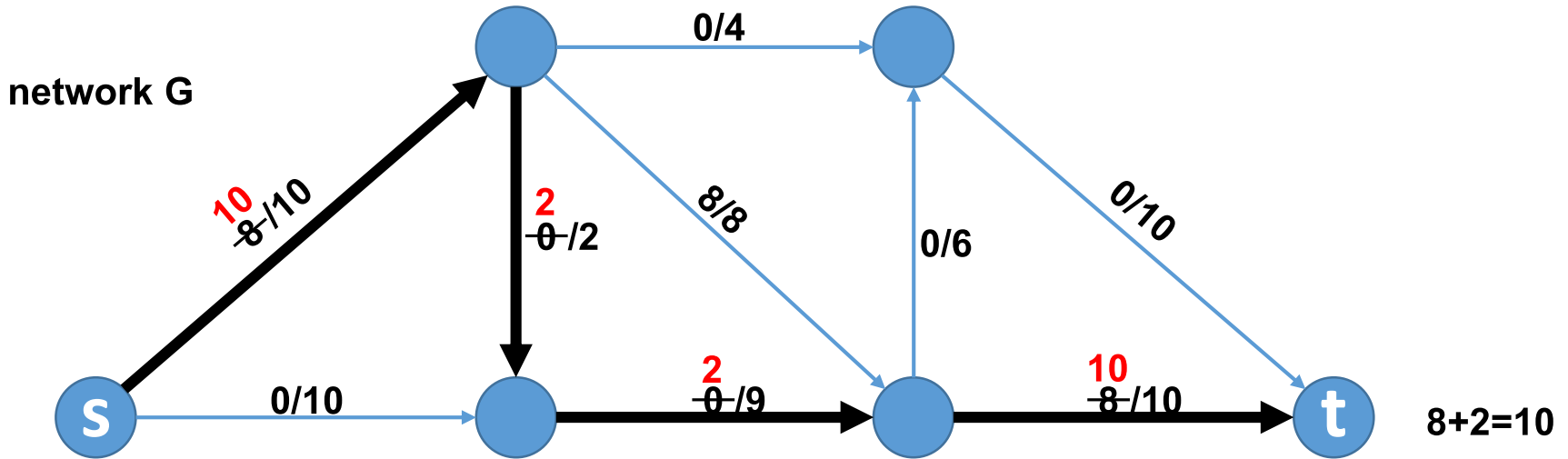
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



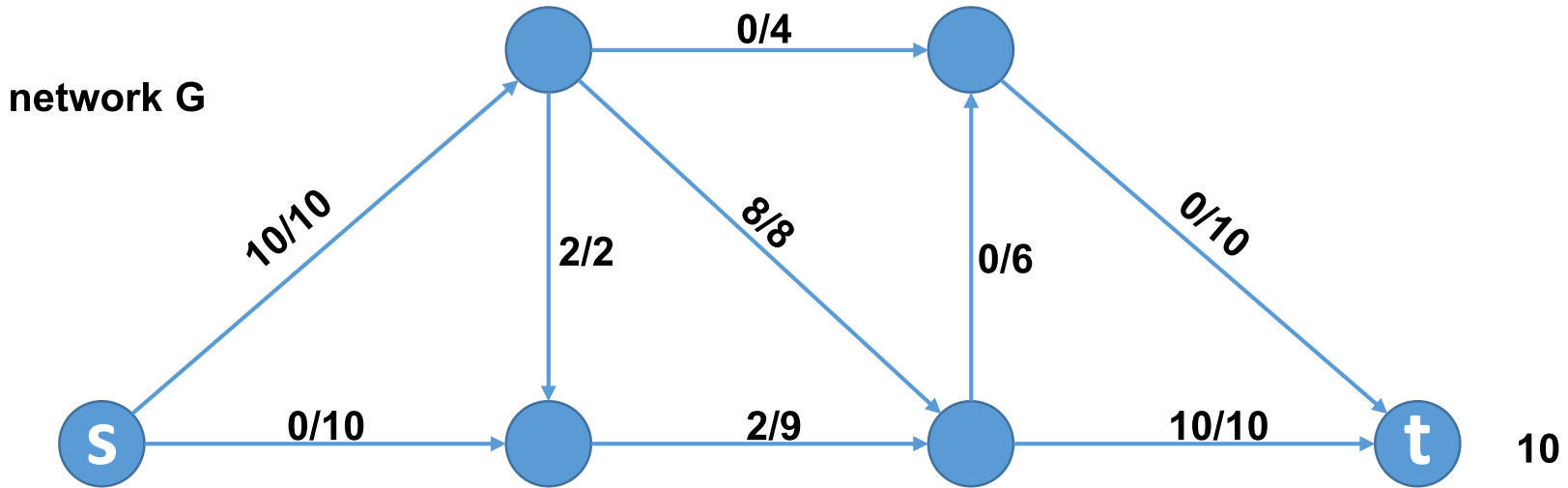
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



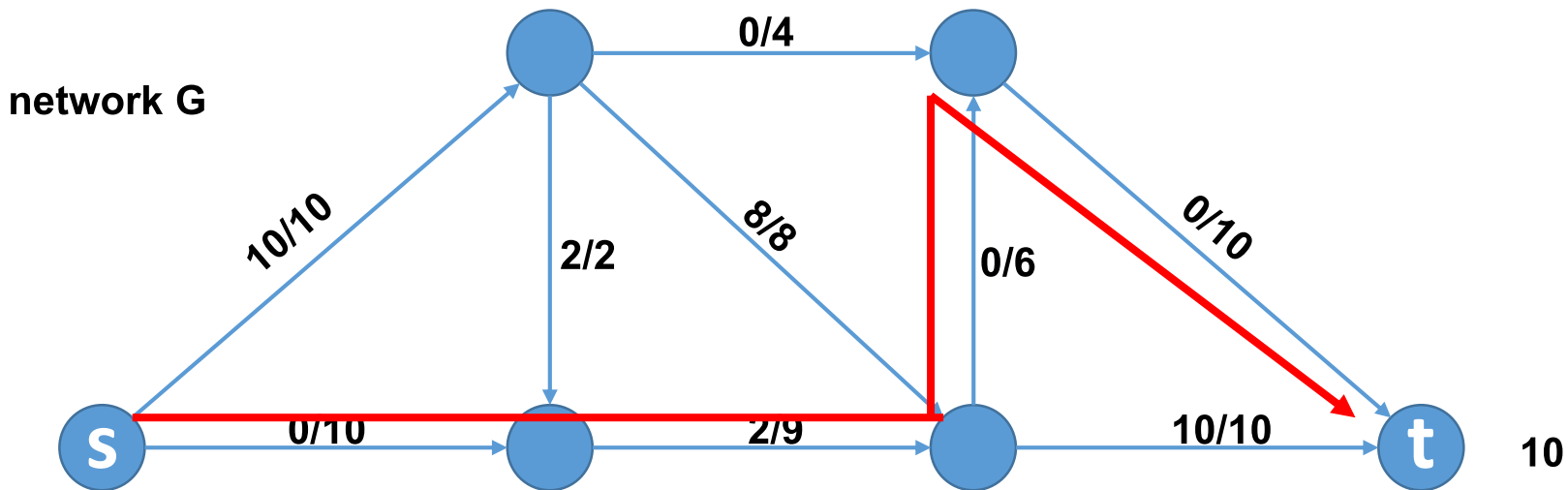
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



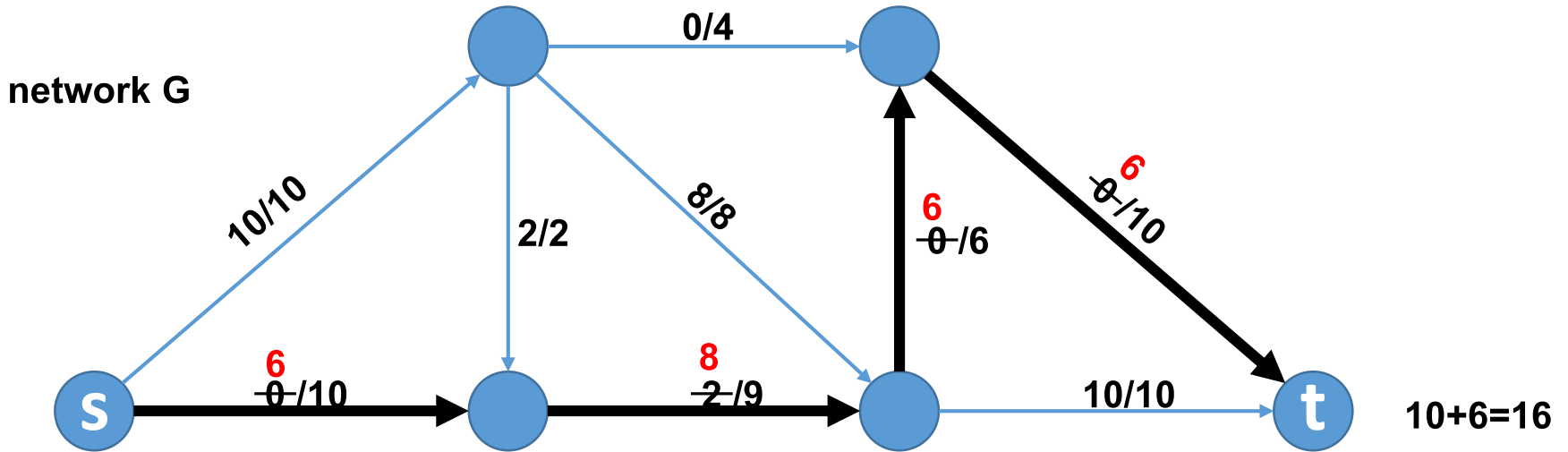
Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



Greedy algorithm (ลอง)

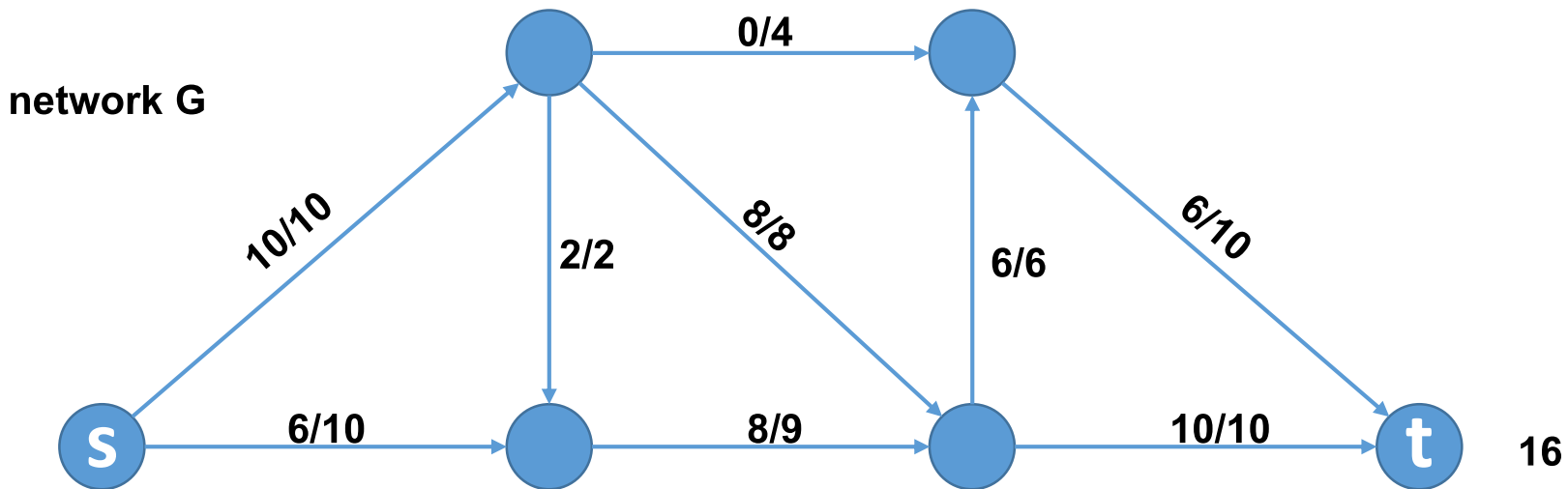
- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้

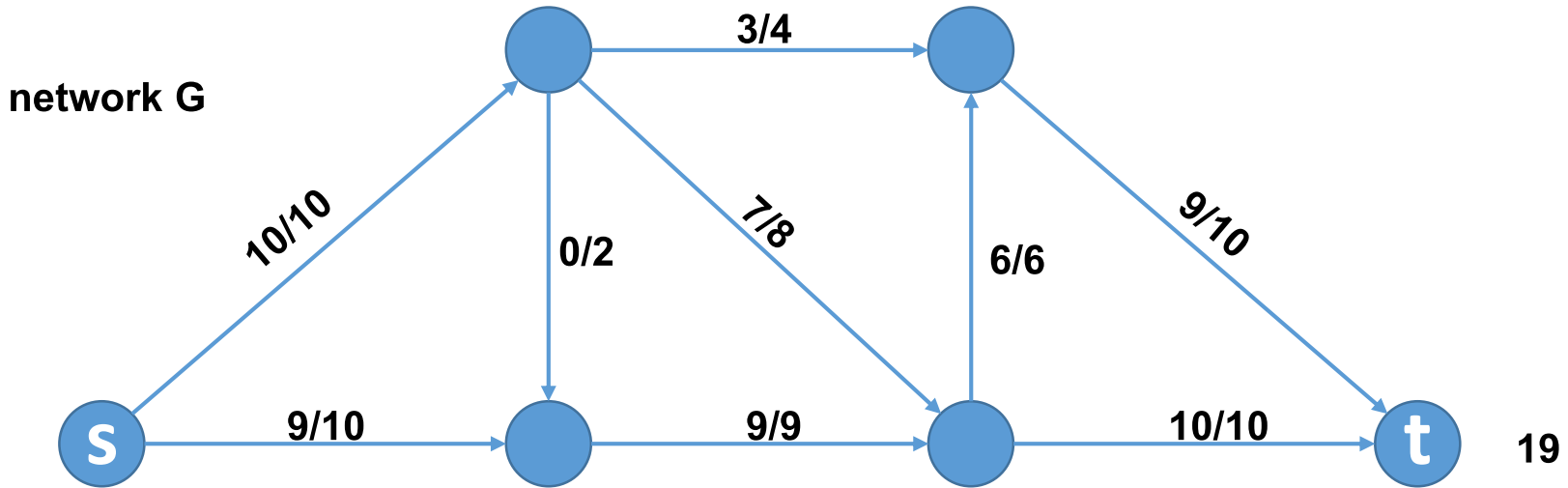
ทำเสร็จด้วย flow = 16



Greedy algorithm (ลอง)

- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า $f(e) = 0$
- หา s-t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้

แต่ **max-flow = 19**



Greedy

หา st-path ที่แต่ละเส้นเชื่อมมี $\text{Flow } f(e) \leq \text{Capacity } c(e)$
จากนั้น augment flow ตาม path นั้น

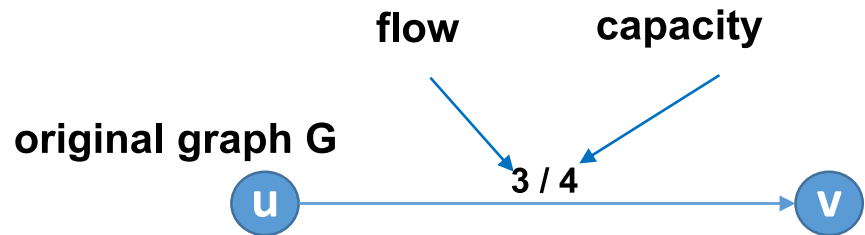
ลักษณะการทำงานของ Greedy คือ ทำซ้ำจนหา path เพิ่มไม่ได้

ข้อเสียคือ การที่จะทำให้ได้ดีที่สุดต้อง backtrack

Residual graph

Original edge: $e = (u, v) \in E$

- Flow $f(e)$
- Capacity $c(e)$

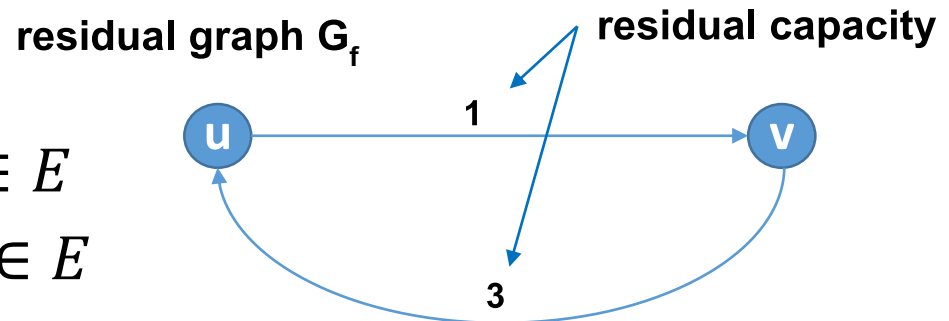


Residual edge ย้อนกลับกับ flow ที่ถูกส่ง

- $e = (u, v)$ และ $e^R = (v, u)$

- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



Residual graph

Residual graph: $G_f = (V, E_f)$

- Residual edge มีค่าเป็น residual capacity ที่เป็นบวก
- $E_f = \{e: f(e) < c(e)\} \cup \{e^R: f(e) > 0\}$

Augmenting path

นิยาม augmenting path คือ simple s-t path P ใน residual graph G_f

นิยาม bottleneck capacity ของ augmenting P คือ minimum residual capacity ของเส้นเชื่อมใดๆ ใน P

คุณสมบัติที่สำคัญ: ให้ f เป็น flow และให้ P เป็น augmenting path ใน G_f แล้ว f' เป็น flow และ $val(f') = val(f) + bottleneck(G_f, P)$

```
AUGMENT( $f, c, P$ )  
 $b =$  bottleneck capacity of path  $P$   
FOREACH edge  $e \in P$   
    IF ( $e \in E$ )  $f(e) = f(e) + b$   
    ELSE        $f(e^R) = f(e^R) - b$   
RETURN  $f$ 
```


Ford-Fulkerson algorithm

Ford-Fulkerson augmenting path algorithm

- เริ่มต้นด้วย $f(e) = 0$ สำหรับทุกเส้นเชื่อม $e \in E$
- หา augmenting path P ใน residual graph G_f
- Augment (เพิ่ม) flow ไปตาม path P
- ทำซ้ำจนทำไม่ได้

FORD-FULKERSON(G, s, t, c)

FOREACH edge $e \in E: f(e) = 0$

$G_f =$ residual graph

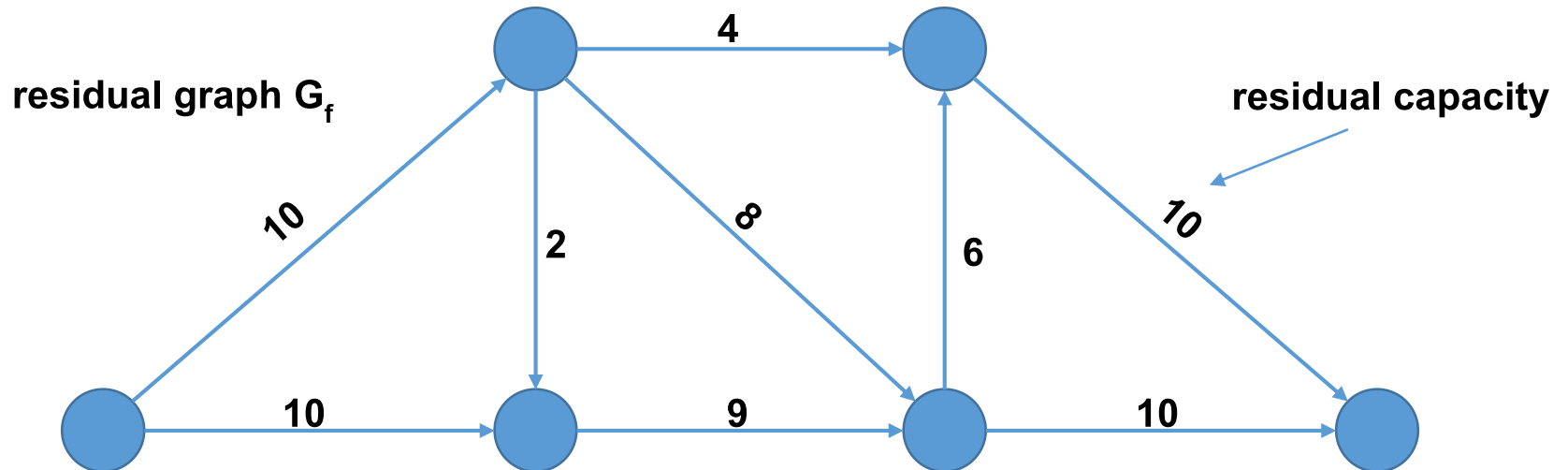
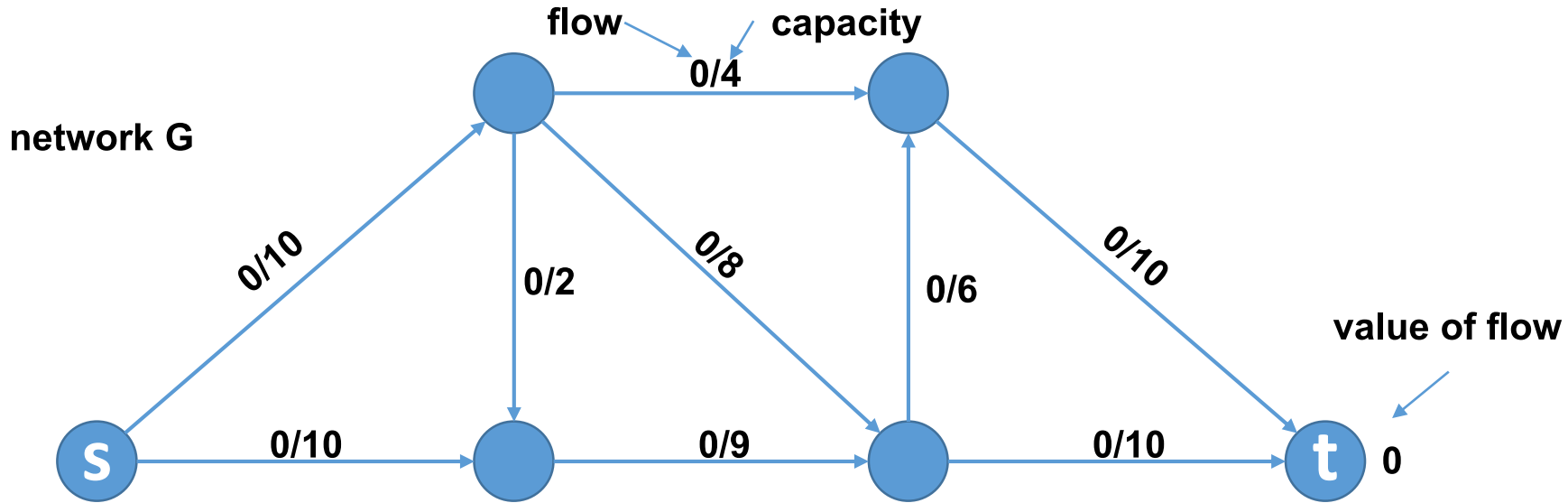
while (there exists an augmenting path P in G_f)

$f =$ AUGMENT(f, c, P)

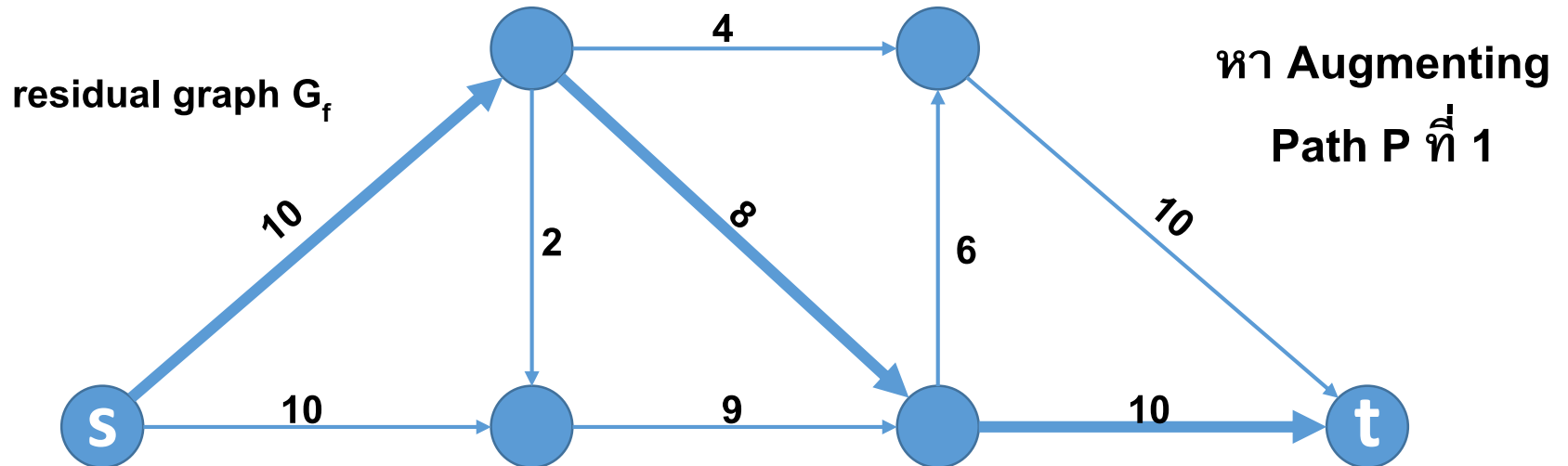
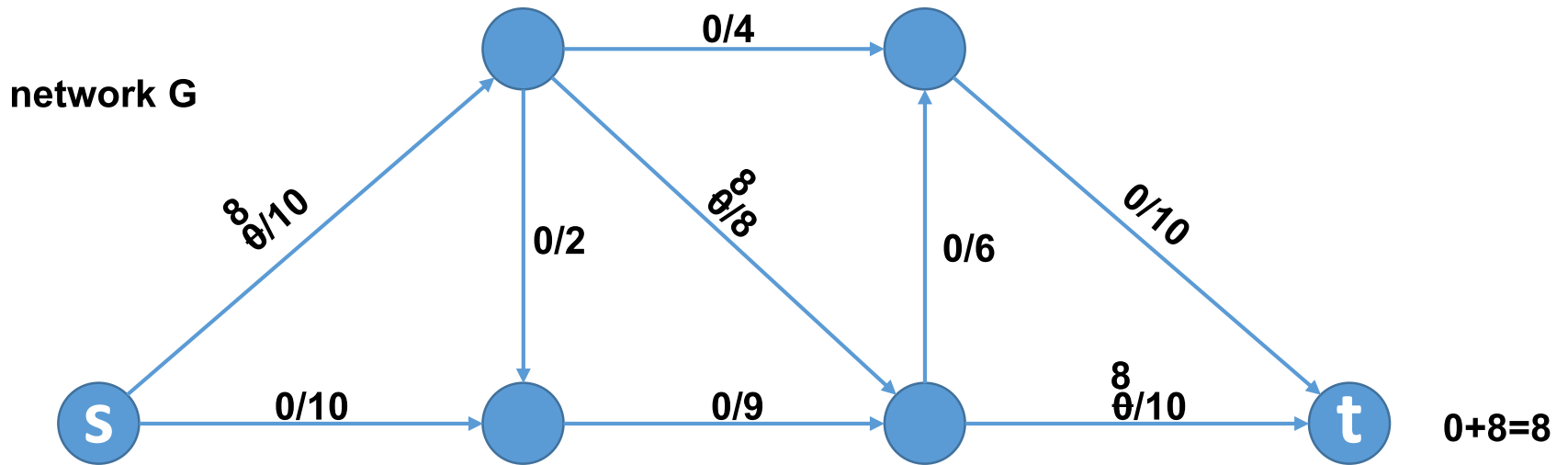
Update G_f

return f

Ford-Fulkerson algorithm demo



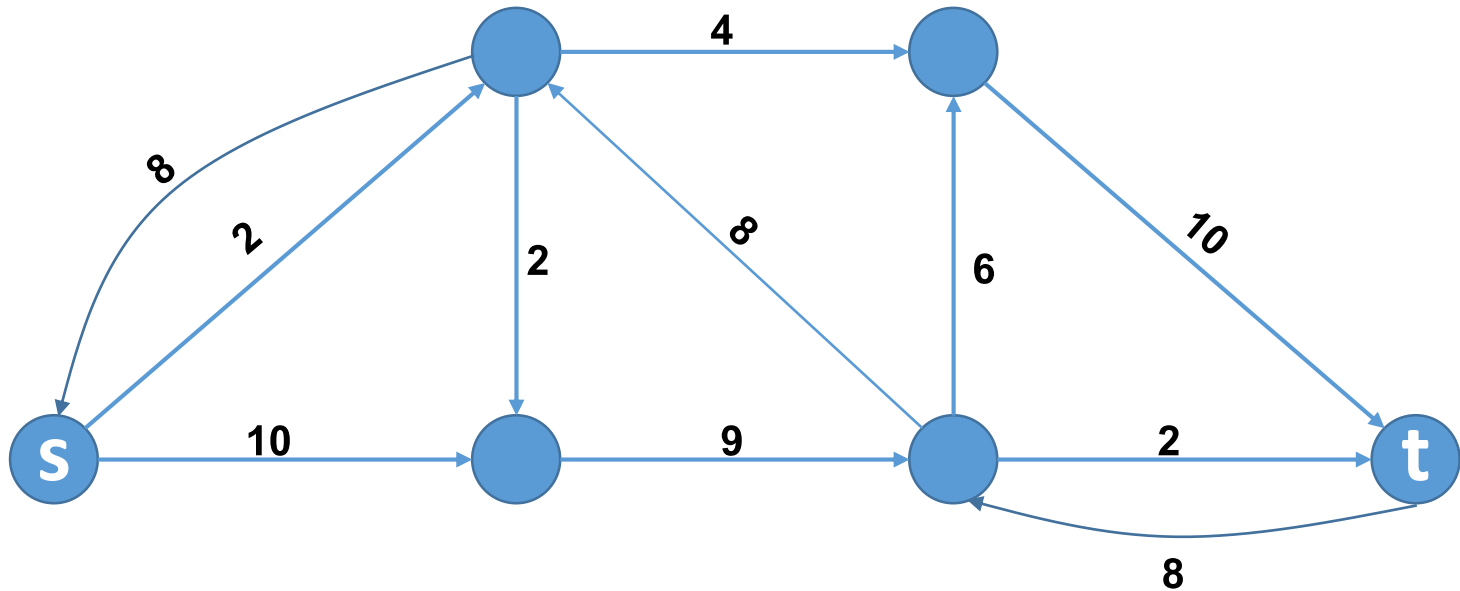
Ford-Fulkerson algorithm demo



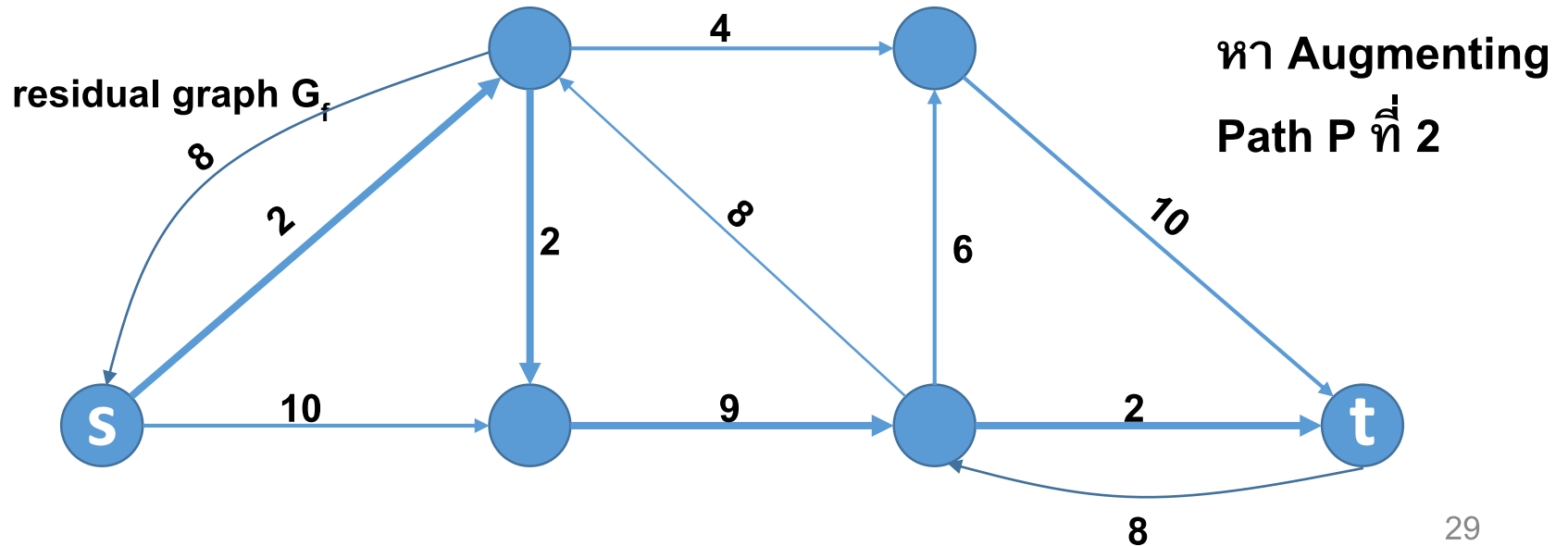
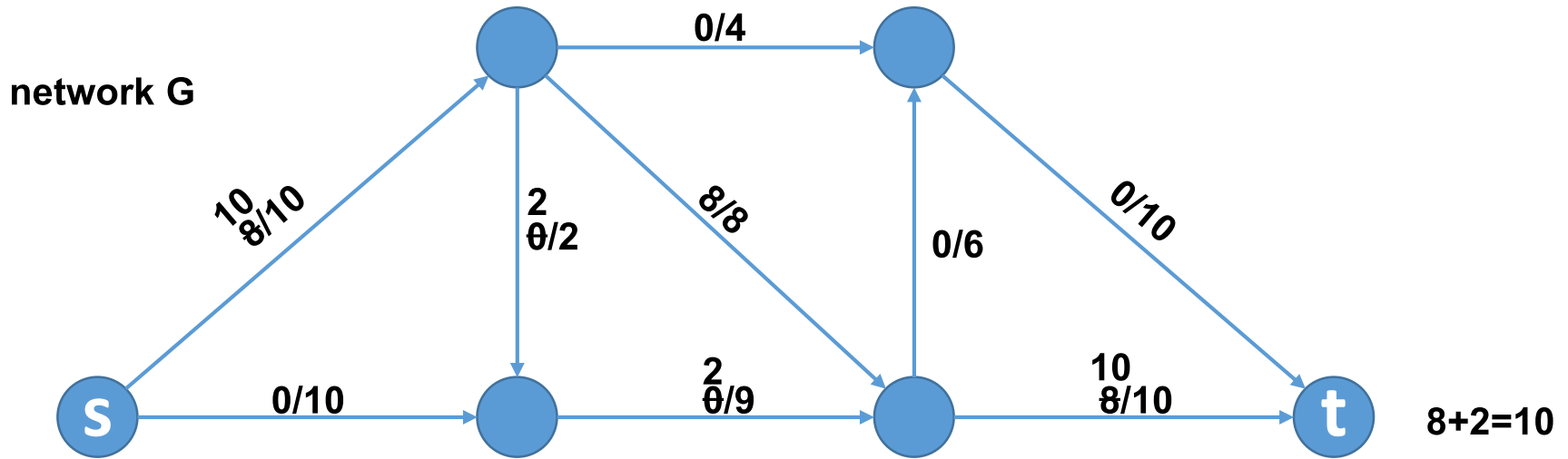
Ford-Fulkerson algorithm demo

- หลังจากการเลือกเส้นทาง ที่ 1 จะได้ residual graph G_f ที่ปรับปรุงแล้วดังนี้

residual graph G_f



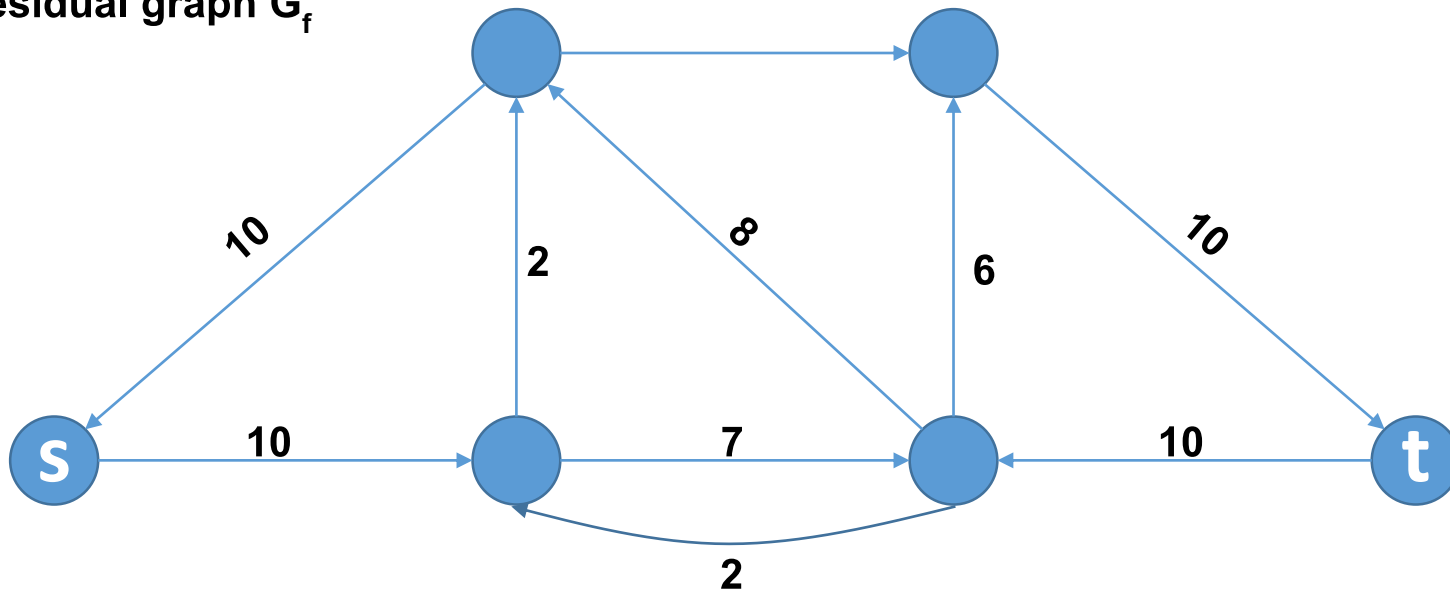
Ford-Fulkerson algorithm demo



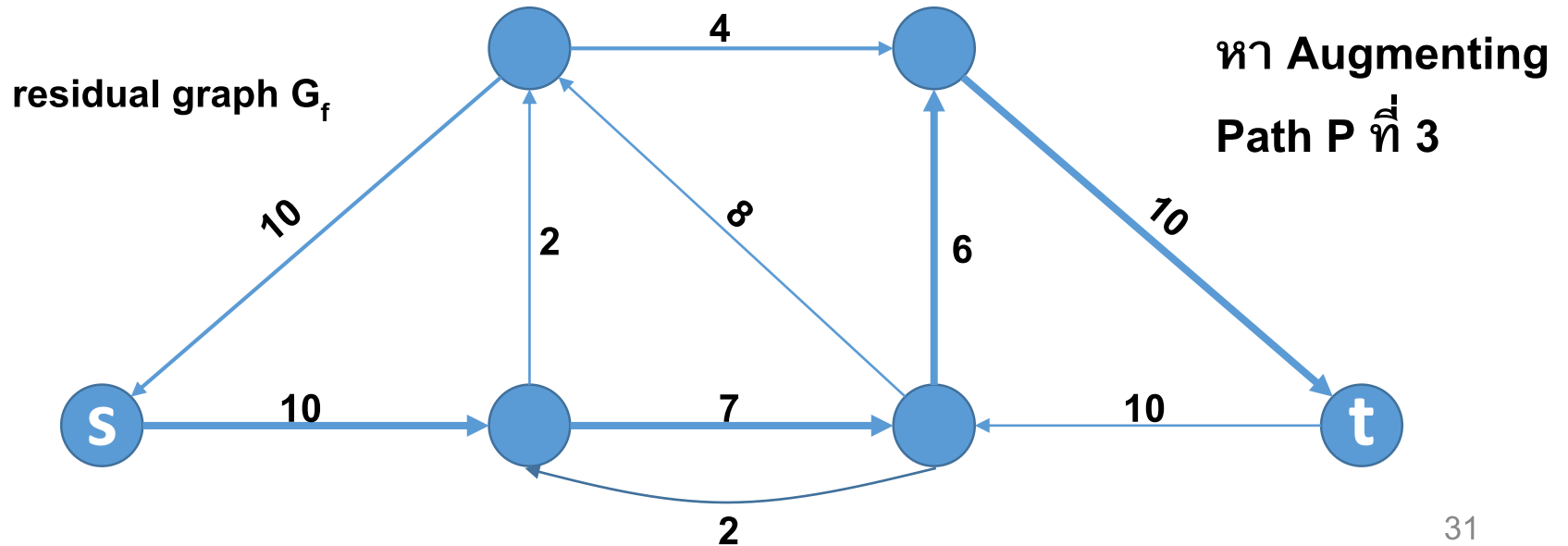
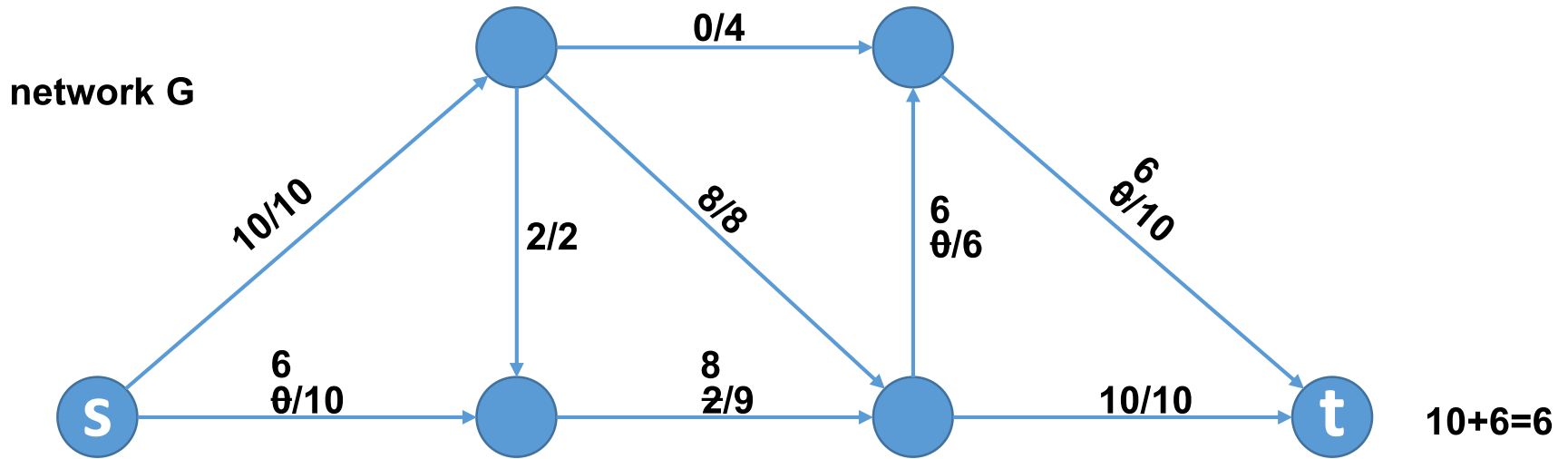
Ford-Fulkerson algorithm demo

- หลังจากการเลือกเส้นทาง ที่ 2 จะได้ residual graph G_f ที่ปรับปรุงแล้วดังนี้

residual graph G_f

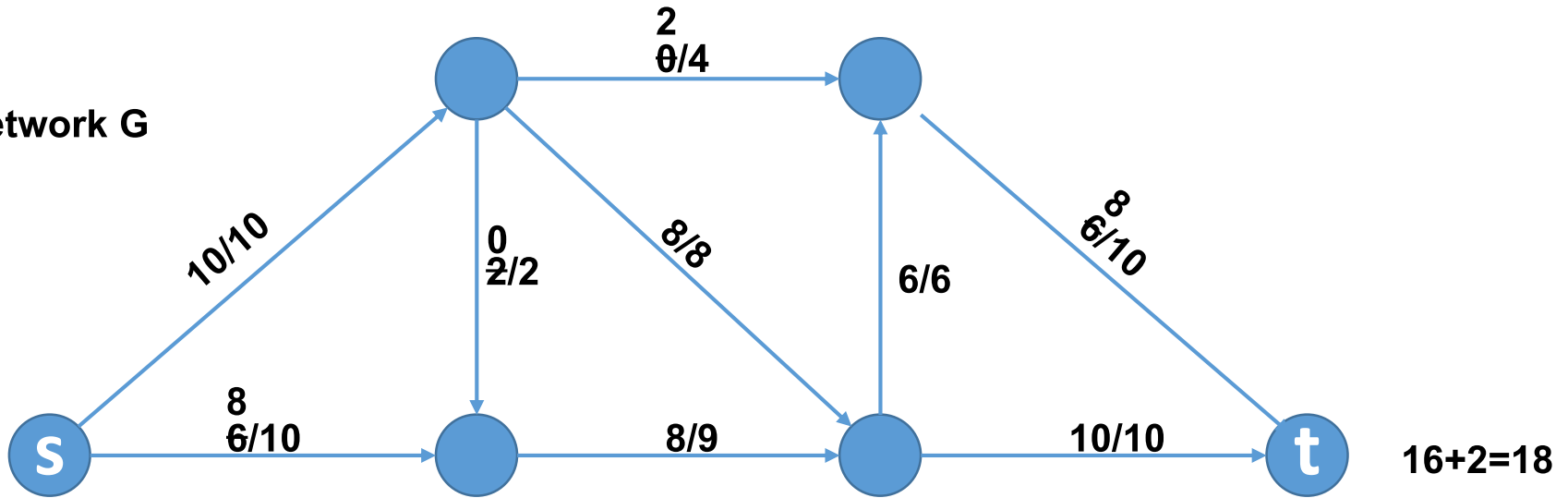


Ford-Fulkerson algorithm demo

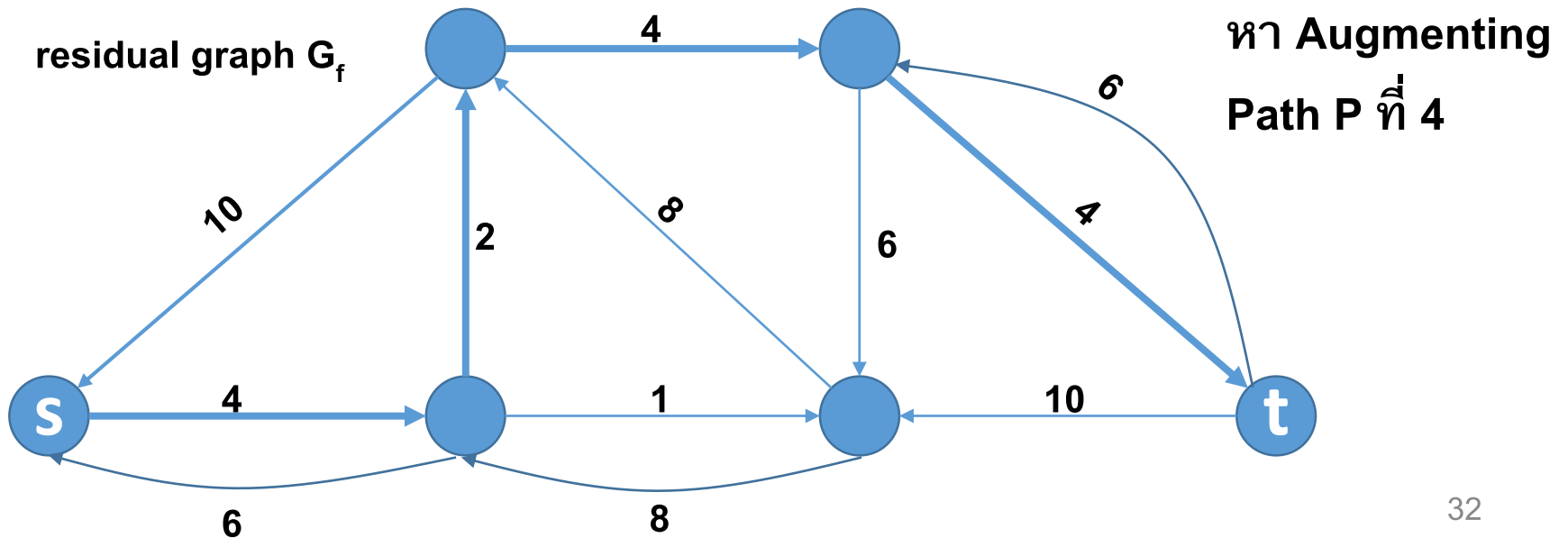


Ford-Fulkerson algorithm demo

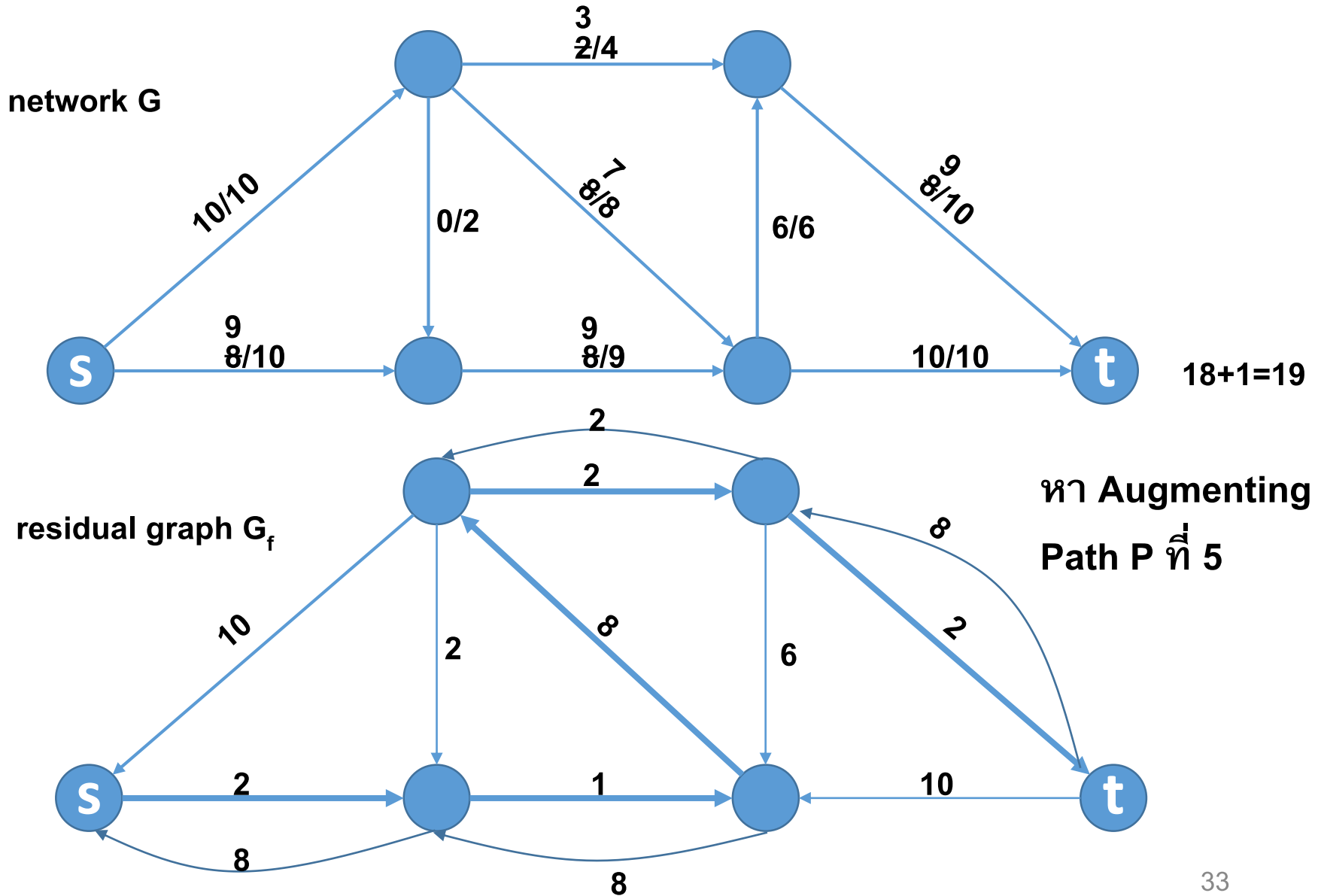
network G



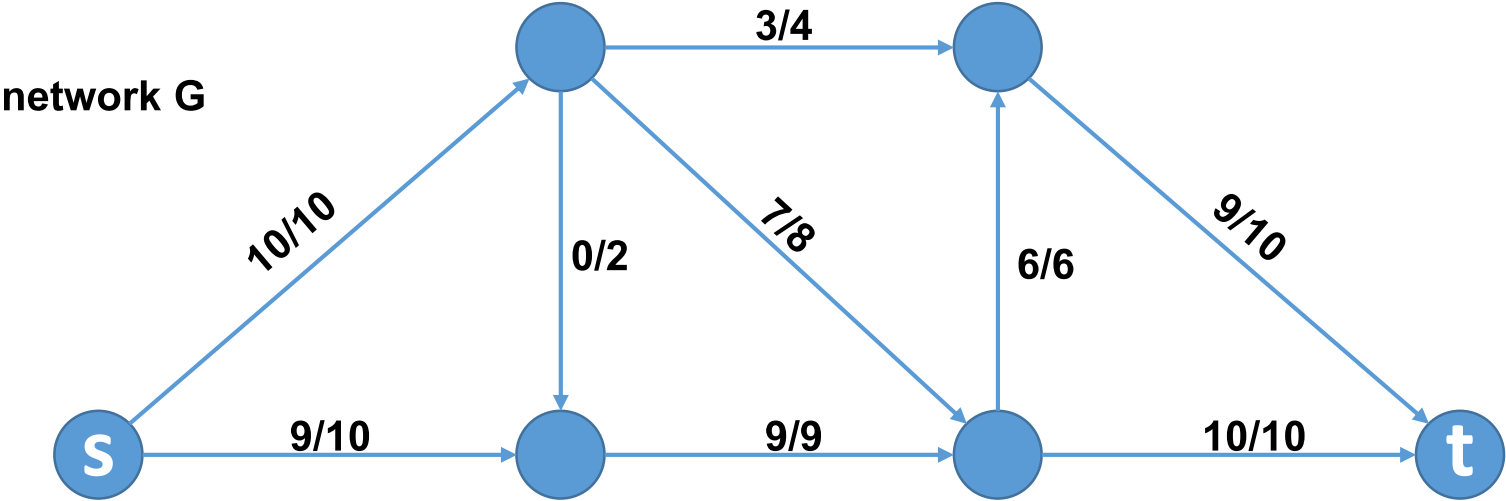
residual graph G_f



Ford-Fulkerson algorithm demo



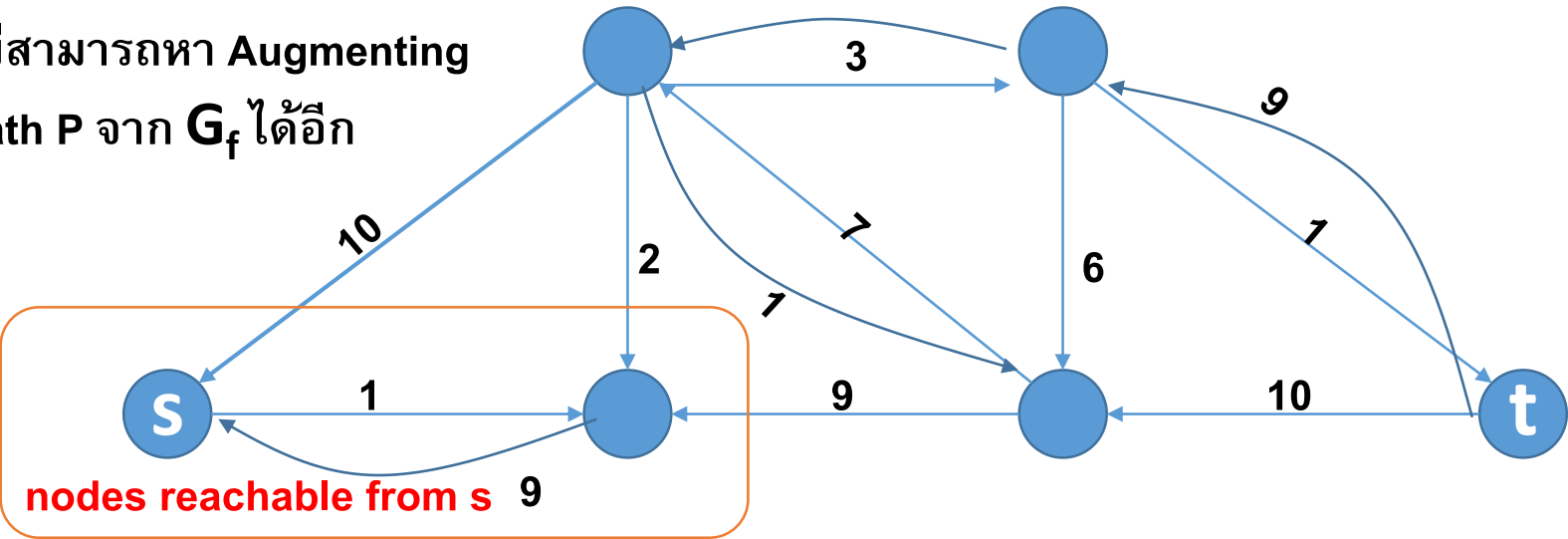
Ford-Fulkerson algorithm demo



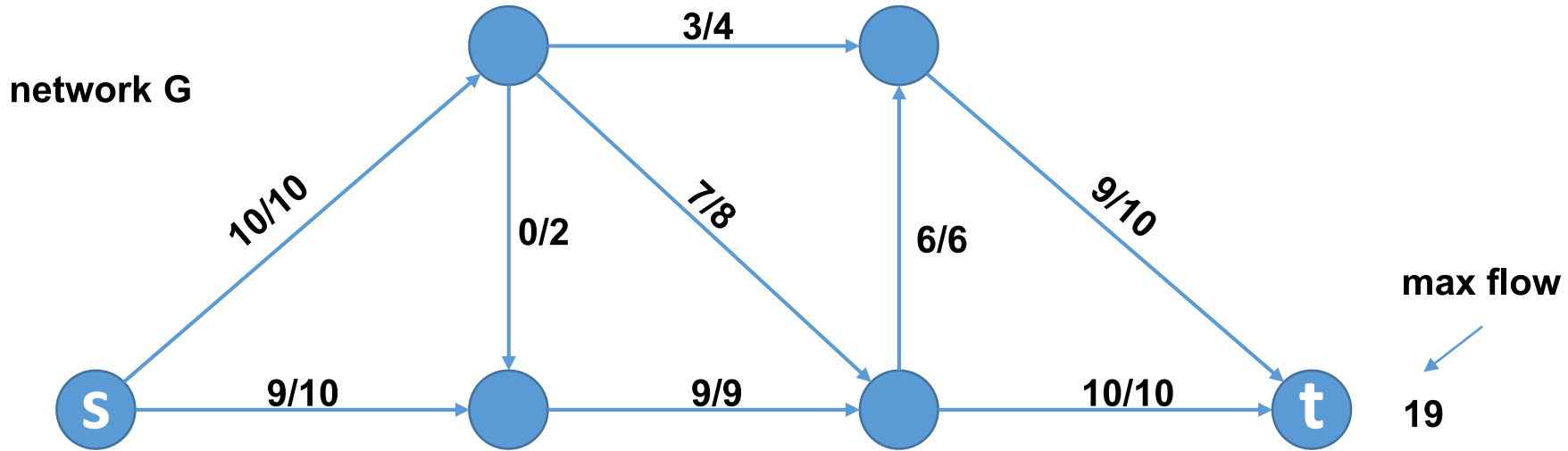
max flow
19

- residual graph G_f ที่ปรับปรุงแล้วดังนี้

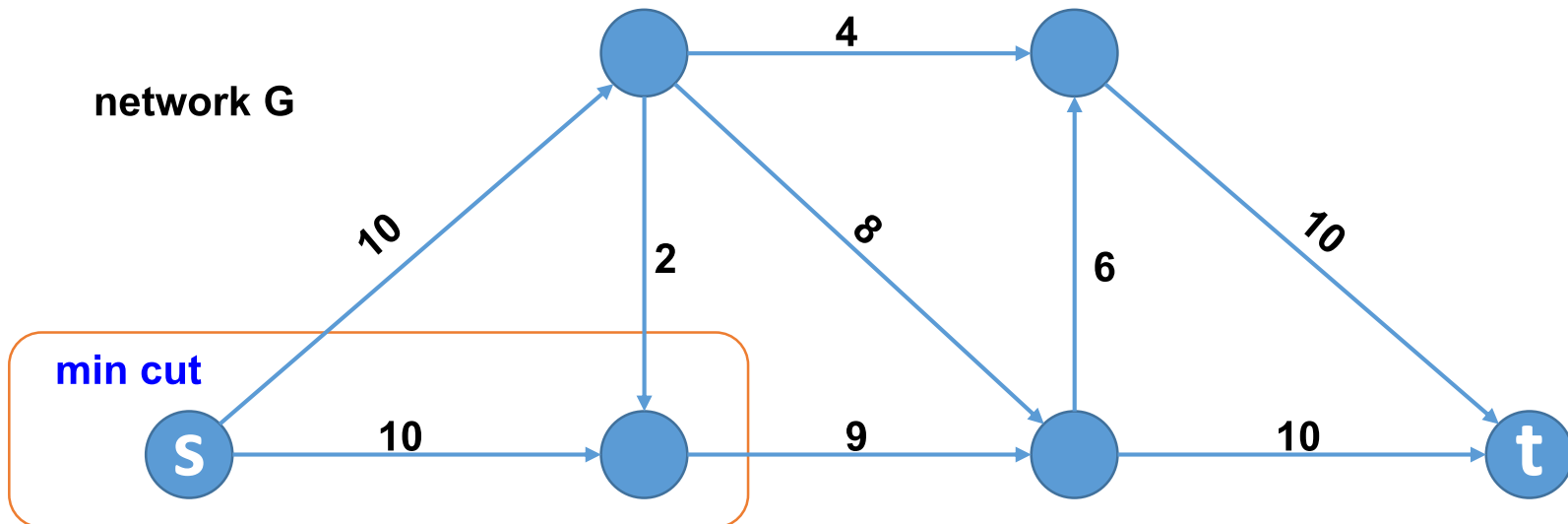
ไม่สามารถหา Augmenting Path P จาก G_f ได้อีก



Max-flow min-cut Theorem



จงหา cut ที่มีความจุต่ำที่สุด ของกราฟ G



Max-flow min-cut Theorem

Augmenting path theorem: flow f จะเป็น maximum flow ก็ต่อเมื่อ
ไม่มี augmenting paths

Max-flow min-cut theorem: ค่าของ max-flow = ความจุของ min-cut

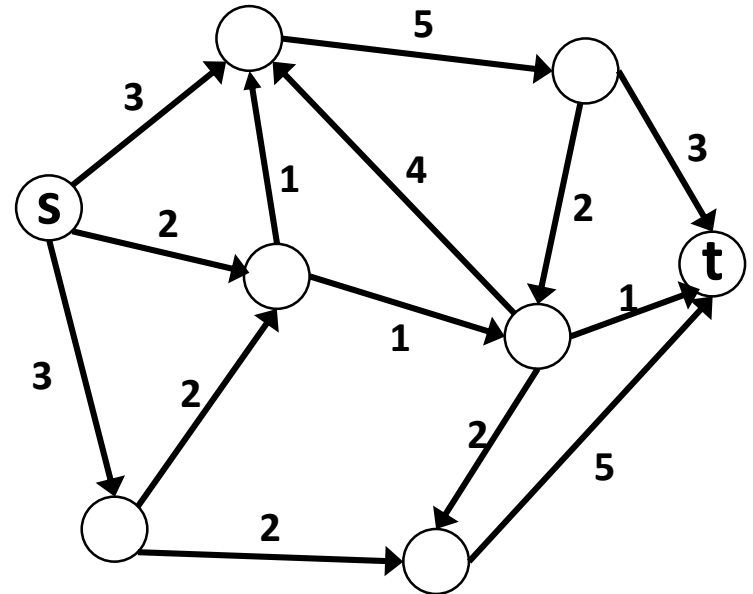
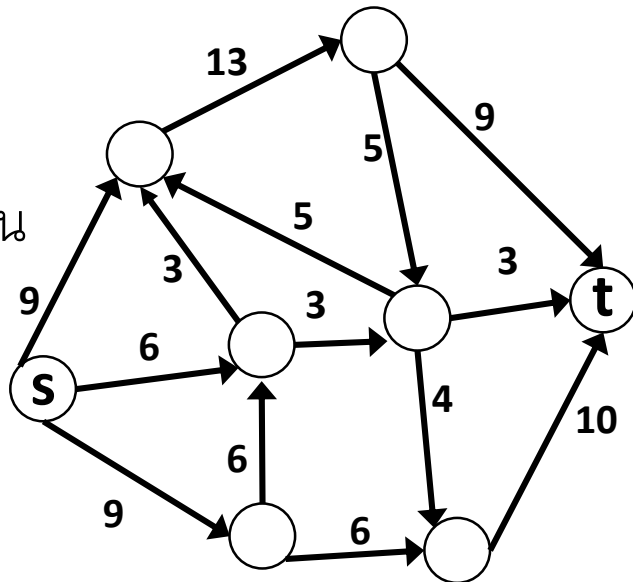
เงื่อนไข 3 ข้อต่อไปนี้เทียบเท่ากันสำหรับ flow f ใดๆ

- จะมี cut(A,B) ที่ $cap(A,B) = val(f)$
- f เป็น max-flow
- ไม่มี augmenting path เมื่อเทียบกับ f

Exercise 1

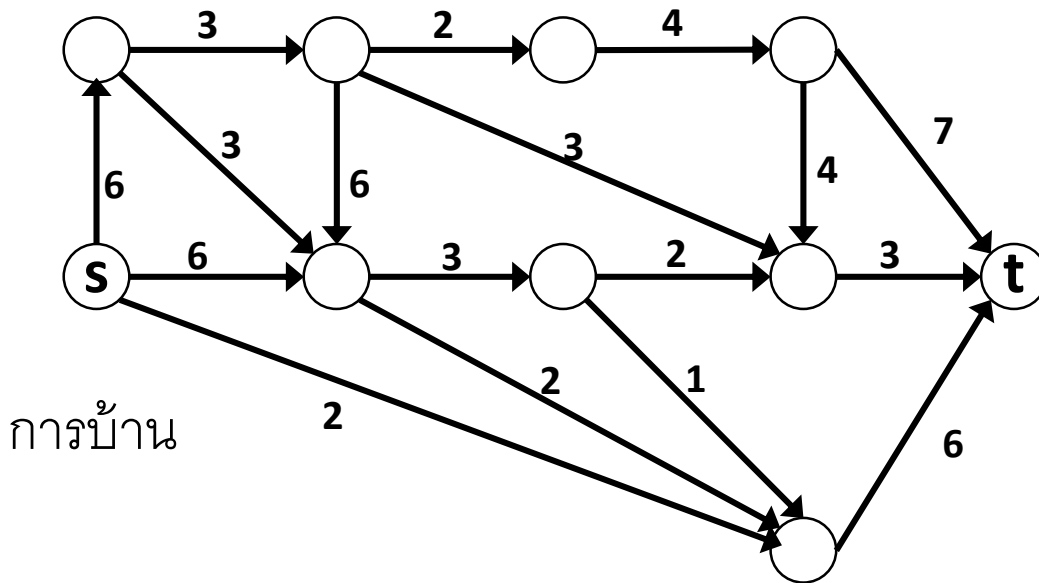
- ให้หาค่า max flow ที่ได้ โดยวิธีการหา Augmenting paths และการสร้าง Residual graph

การบ้าน



Exercise 2

- ให้หาค่า max flow ที่ได้ โดยวิธีการหา Augmenting paths และการสร้าง Residual graph



Ford-Fulkerson Augmenting Path Algorithm

```
while(there exists an augmenting path){  
    Find augmenting path P  
    Compute bottleneck capacity of P  
    Augment flow along P  
}
```

คำถาม

Q: การทำเช่นนี้ทำให้ได้ max flow ใช่หรือไม่

A: ใช่

Ford-Fulkerson Algorithm: Analysis

Assumption: ให้ความจุมีค่าเป็นจำนวนเต็ม

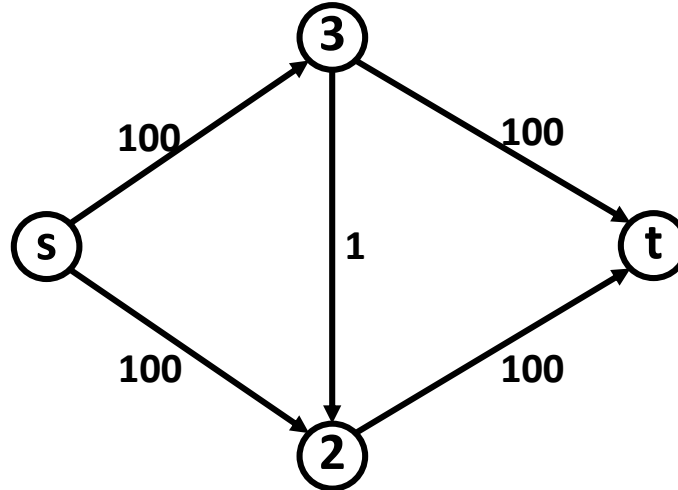
ในแต่ละรอบการทำงานจะมีการหา **st-path** ซึ่งใช้เวลาในการทำงาน **$O(N+M)$**

เราจะส่ง **flow** อย่างน้อย 1 หน่วยผ่าน **path** นี้

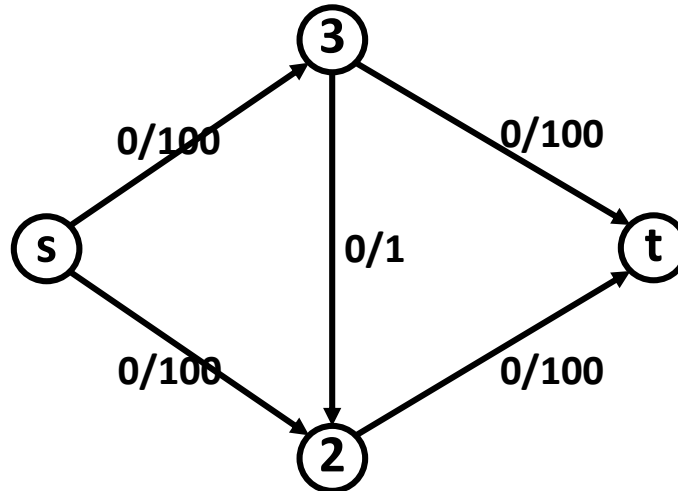
ถ้า **max-flow** มีค่าเป็น **f^*** แล้วเวลาในการทำงานของอัลกอริทึมมีค่าเป็น **$O((N+M)*|f^*|)$**

Choosing Good Augmenting Path

residual graph

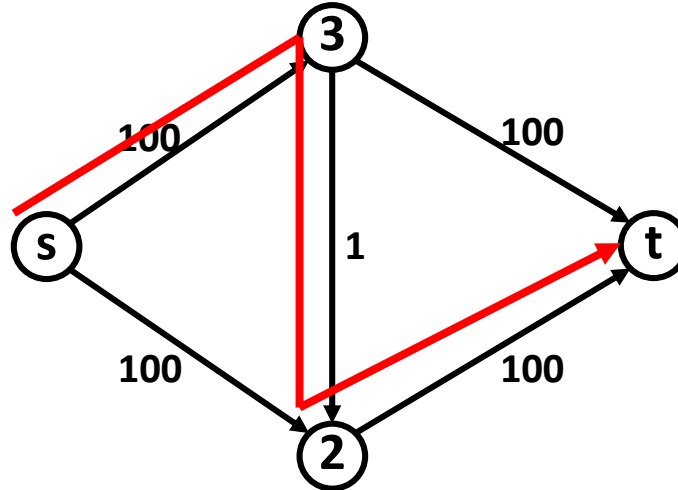


original graph

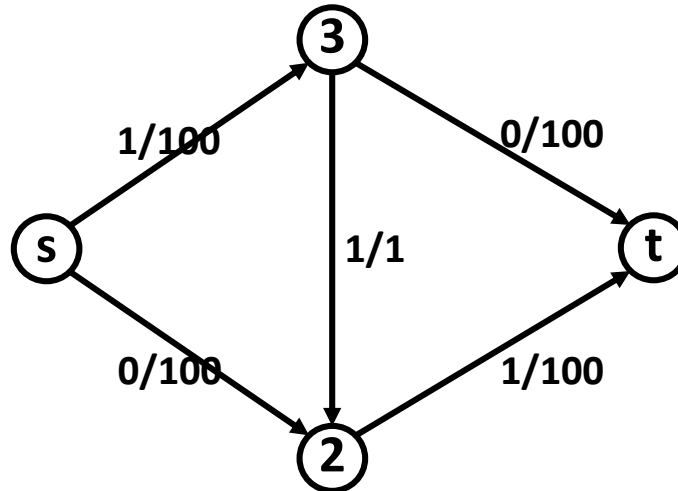


Choosing Good Augmenting Path

residual graph

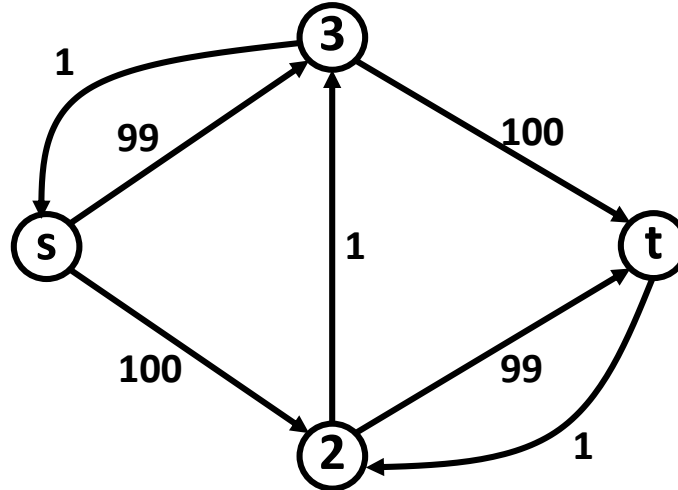


original graph

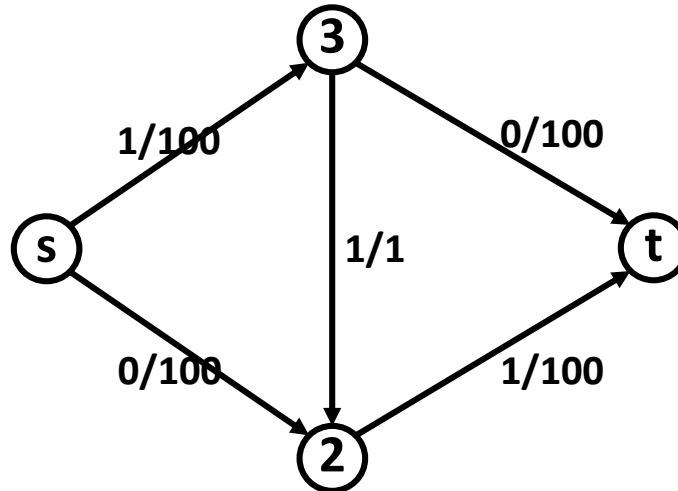


Choosing Good Augmenting Path

residual graph

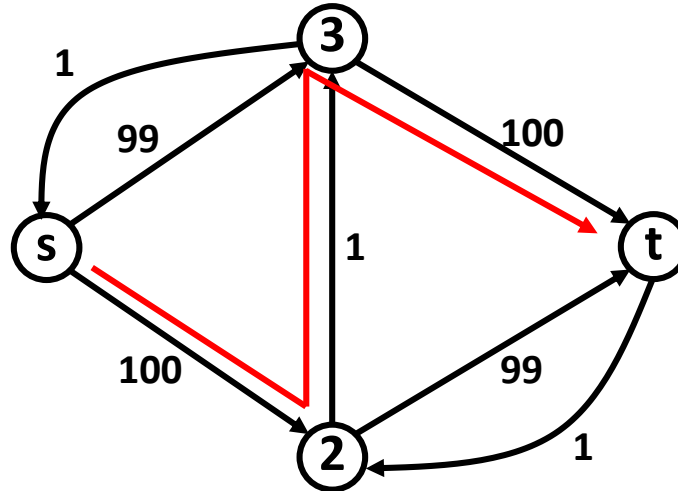


original graph

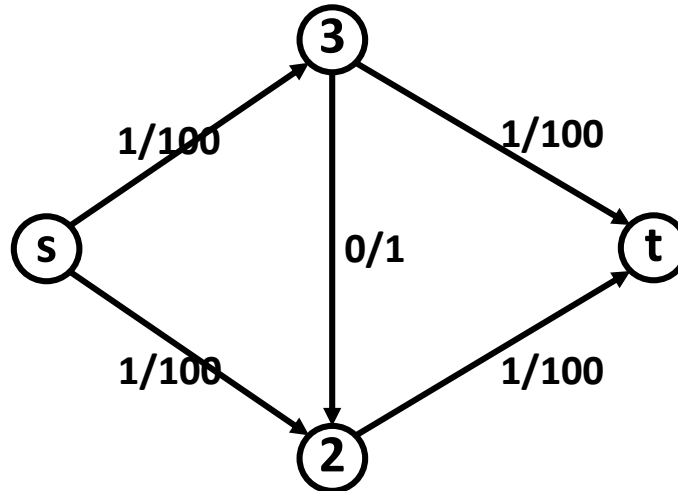


Choosing Good Augmenting Path

residual graph



original graph



หากเลือกไม่ได้ต้องทำถึง 200 รอบ

Choosing Good Augmenting Path

ควรระวังในการเลือก augmenting path

- การเลือกบางวิธีทำให้ได้ exponential algorithm
- การเลือกบางวิธีทำให้ได้ polynomial algorithm

ออกแบบรูปแบบของ augmenting path ที่อยากได้

- หา augmenting path ได้อย่างมีประสิทธิภาพ
- ทำจำนวนรอบที่น้อย

เลือก augmenting path ด้วยวิธี

- ใช้จำนวนเส้นเชื่อมที่น้อยที่สุด (shortest path)
- เลือกเส้นที่ bottleneck capacity ใหญ่ที่สุด (fattest path)

Shortest augmenting path

หาได้ง่าย สามารถใช้ BFS ได้

หา augmenting path ที่มีจำนวนเส้นเชื่อมน้อยที่สุด

ในแต่ละรอบความยาวของ shortest augmenting path จะเพิ่มขึ้น

- ความยาวเพิ่มไม่เกิน E
- ไม่เกิน EV augmenting path ทั้งหมด
- ดังนั้นเวลาในการทำงานเป็น $O(E^2V)$

Fattest augmenting path

หา augmenting path ที่มี bottleneck capacity ที่มีค่ามากที่สุด

ส่ง flow ไปยัง sink

solve โดยใช้ dijkstra-style (Priority-first search) algorithm

การหา fattest path ใช้ $O(E \log V)$ ต่อการ augment โดยใช้ binary heap

การหา min-cut

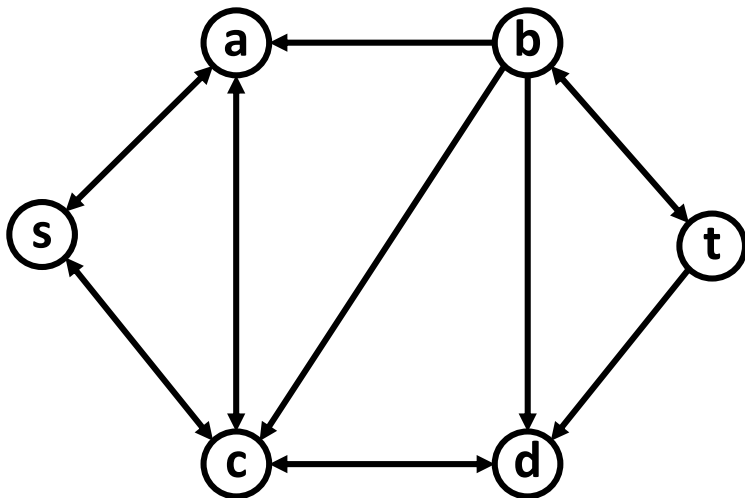
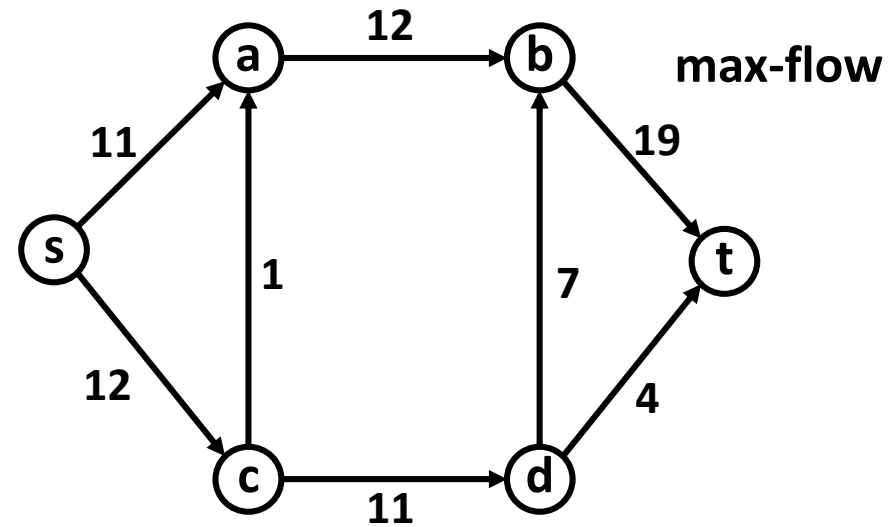
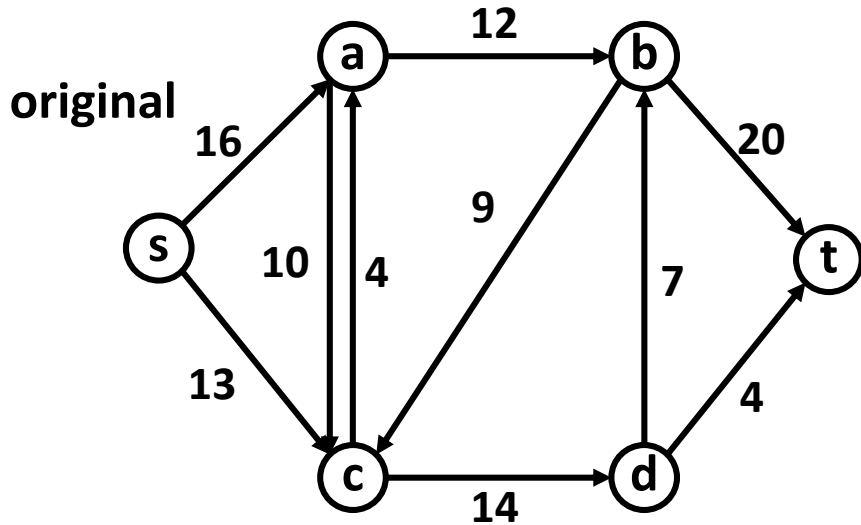
- เรารู้ว่า max flow – min cut
- ตอนนี้เรารู้วิธีการหา max flow

Q: เราจะหา min-cut ได้อย่างไร

A: เราจะใช้ residual graph

การทำ min-cut

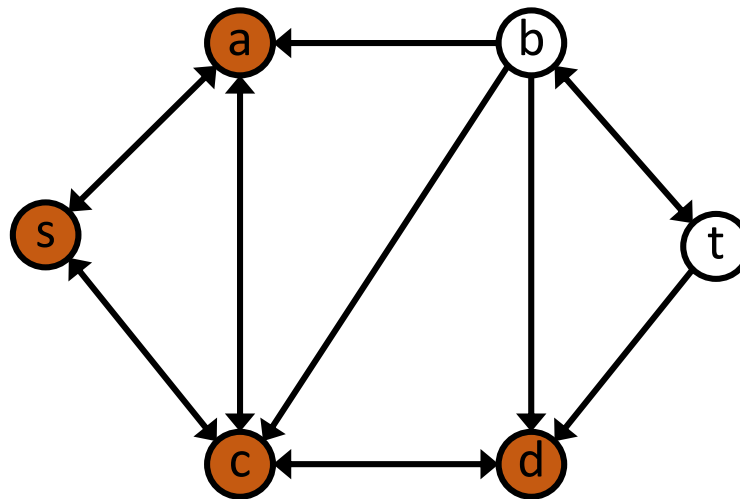
- เราจะลบ max flow ออกจาก original graph



แสดงแค่ topology ของ residual graph
อย่าลืมเพิ่มเส้นย้อนกลับ

การหา min-cut

- mark ทุกโหนดที่ไปถึงจาก s
 - เรียก set ของโหนดที่ไปถึงจาก s ว่า A



แยกโหนดเหล่านี้ออกจากกลุ่ม

เส้นเชื่อมที่วิ่งจาก A ไปยัง V-A คือ cut นั้นเอง

การหา min-cut

- พิจารณาใน original graph เพื่อให้ cut

