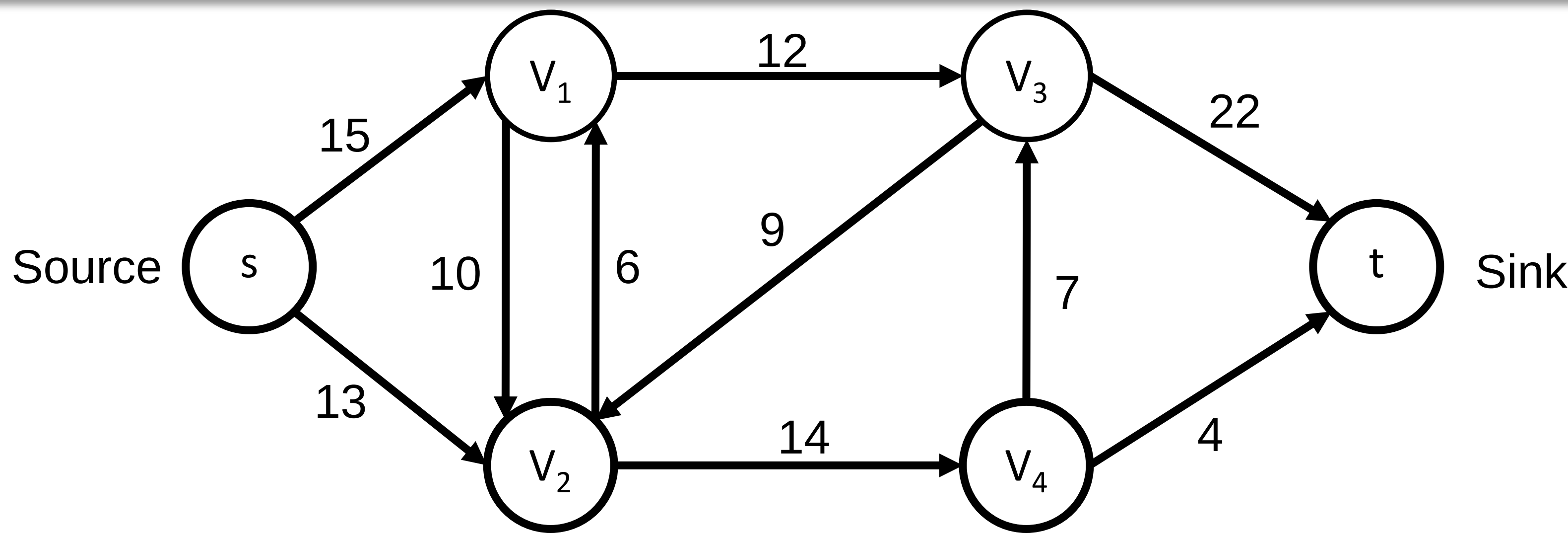
Network Flow

Overview and Motivation

- พิจารณา connected (integer) weighted and directed graph เป็นเครือข่ายการเชื่อมต่อกันของท่อเมื่อเส้นเชื่อมคือท่อและโหนดคือจุดที่ท่อแยกกัน(ข้อต่อ)
- เส้นเชื่อมแต่ละเส้นจะมีน้ำหนักแทน**ความจุ**ของท่อ และมีโหนดพิเศษ 2 โหนด **source s** และ **sink t**
- **Maximum flow** จาก source s ไป sink t จินตนาการว่าเราต้องการปล่อยน้ำให้ไหลในท่อ เราต้องการรู้ว่าปริมาณน้ำมากที่สุดที่สามารถไหลผ่านท่อได้เป็นเท่าไร

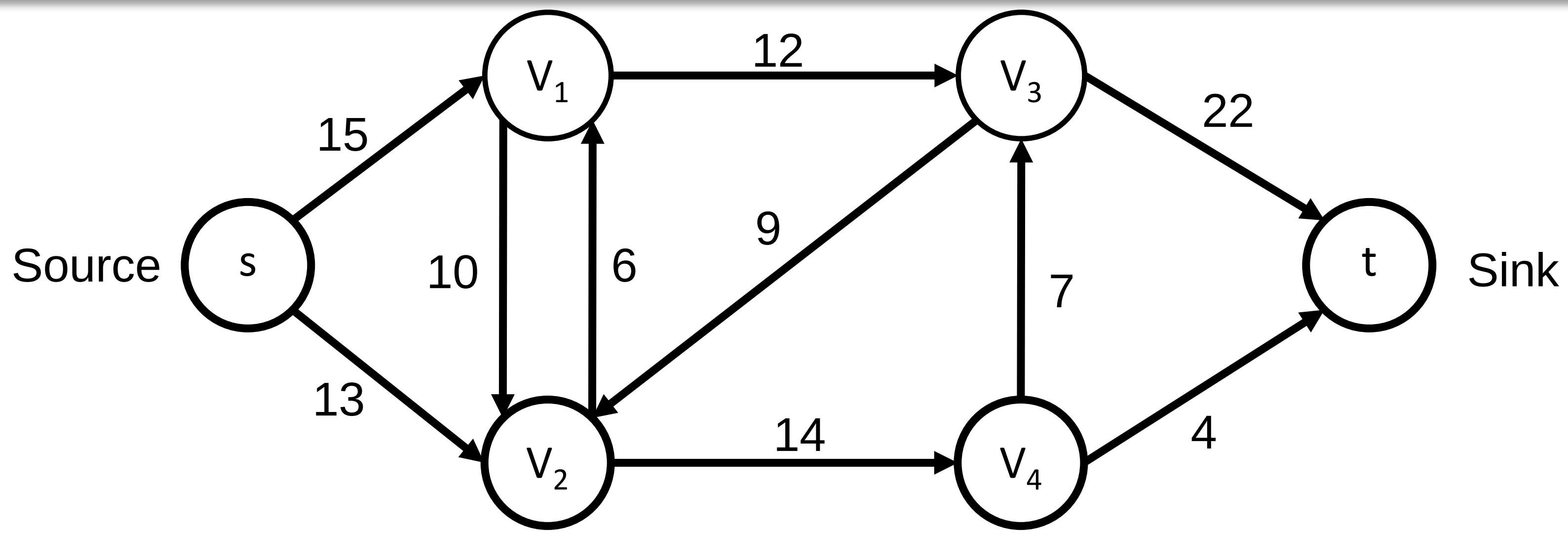
หาปริมาณการไหลสูงสุดจาก s ไป t

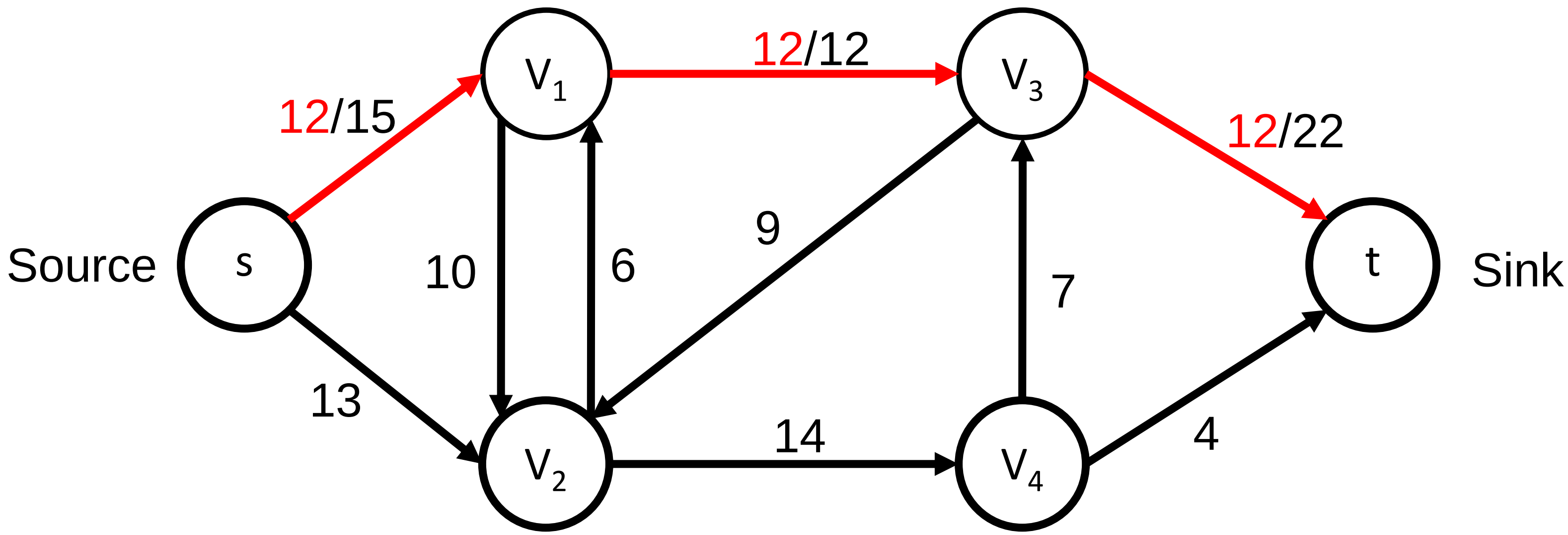


- เราต้องการส่งน้ำจาก s **<u>ต่อเนื่อง</u>**ไป t ให้ได้มากสุด หา st–path
- เราก็ค่อยๆ ส่งไป จากเดิมมีเลขบนเส้นเช่น 15 นั่นคือความจุ
- ถ้าเราส่งน้ำผ่าน x หน่วยก็จะเป็น x/15 ทั้งนี้ส่งน้ำได้ไม่เกินความจุ
- Flow/Capacity

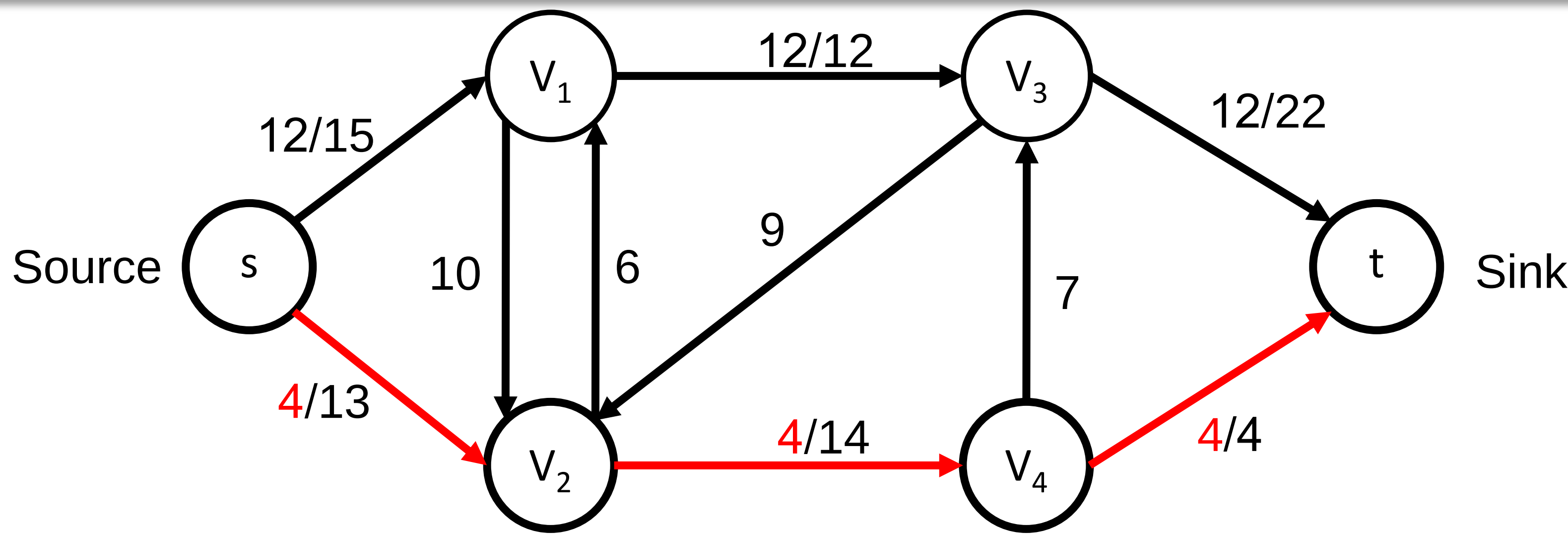
- การไหล กำหนดให้ f(u,v) คือปริมาณการไหลจากโหนด u ไปโหนด v
  - ปริมาณการไหลในเส้นเชื่อมต้องไม่เกินความจุ
  - ปริมาณการไหลเข้าของโหนดใดๆ ต้องเท่ากับปริมาณการไหลออก ทั้งนี้ยกเว้น s กับ t
  - เราต้องการให้โหนด s ปล่อยออกอย่างเดียว และโหนด t รับอย่างเดียว
  - ดังนั้น $|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$ เพราะว่าโหนดอื่นๆ รับมาเท่าไรต้องปล่อยออกหมด

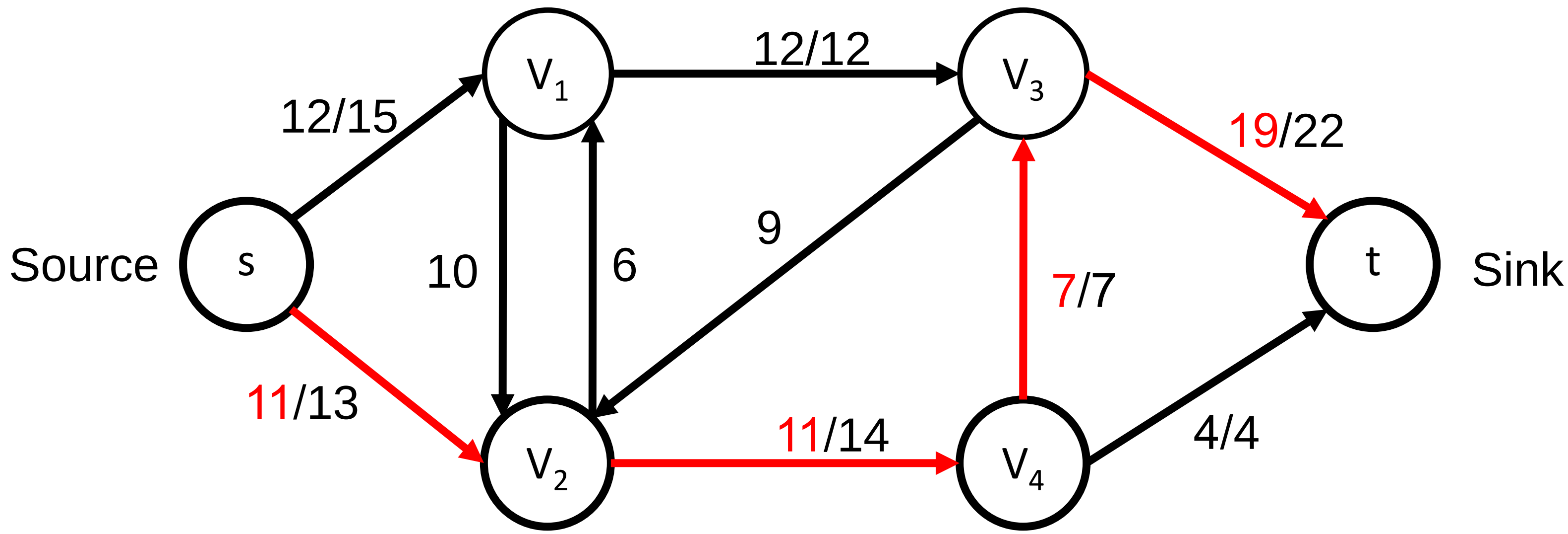
หาปริมาณการไหลสูงสุดจาก s ไป t

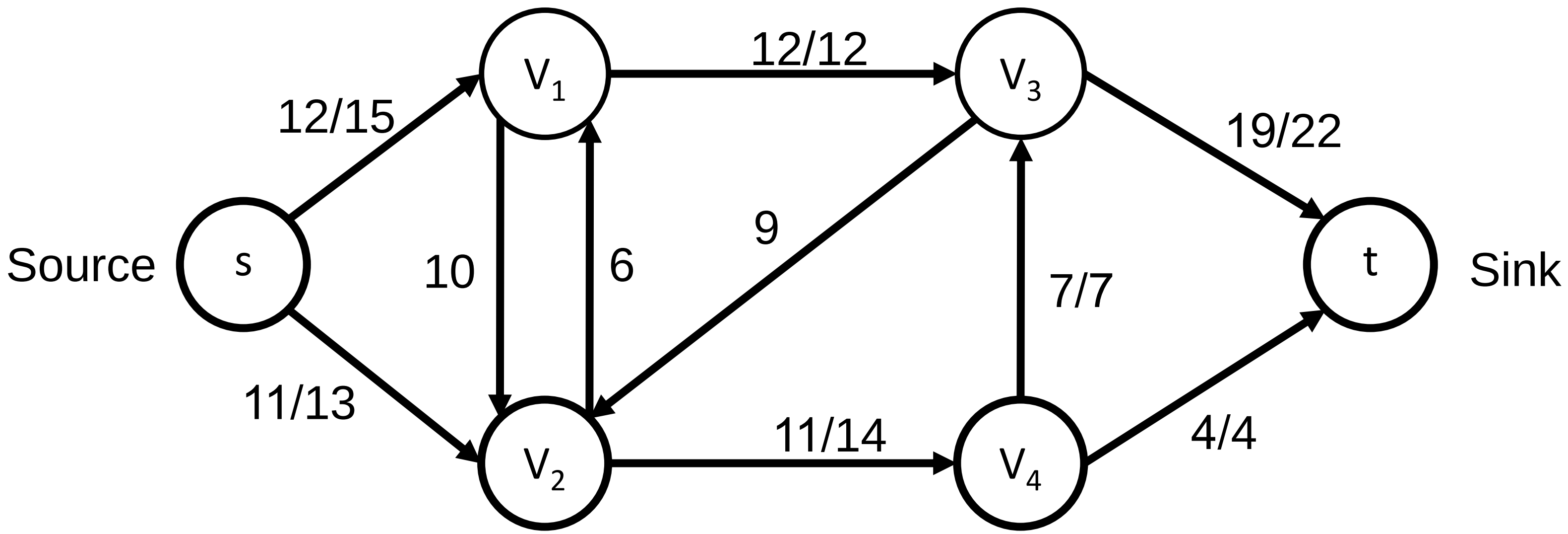




หาปริมาณการไหลสูงสุดจาก s ไป t



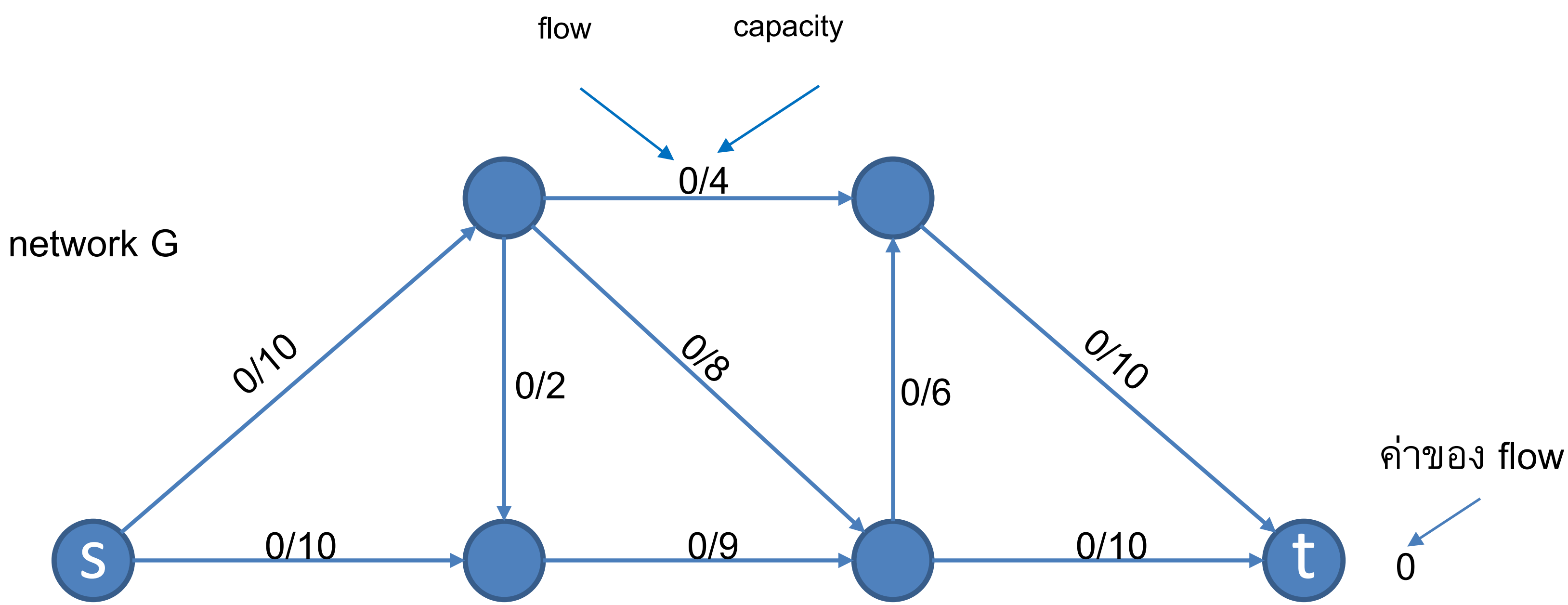




- สรุปส่ง flow ได้ 23 หน่วย
- จะนั่งนับแต่ละรอบ หรือว่าดูข้อสังเกตที่โหนด s หรือโหนด t

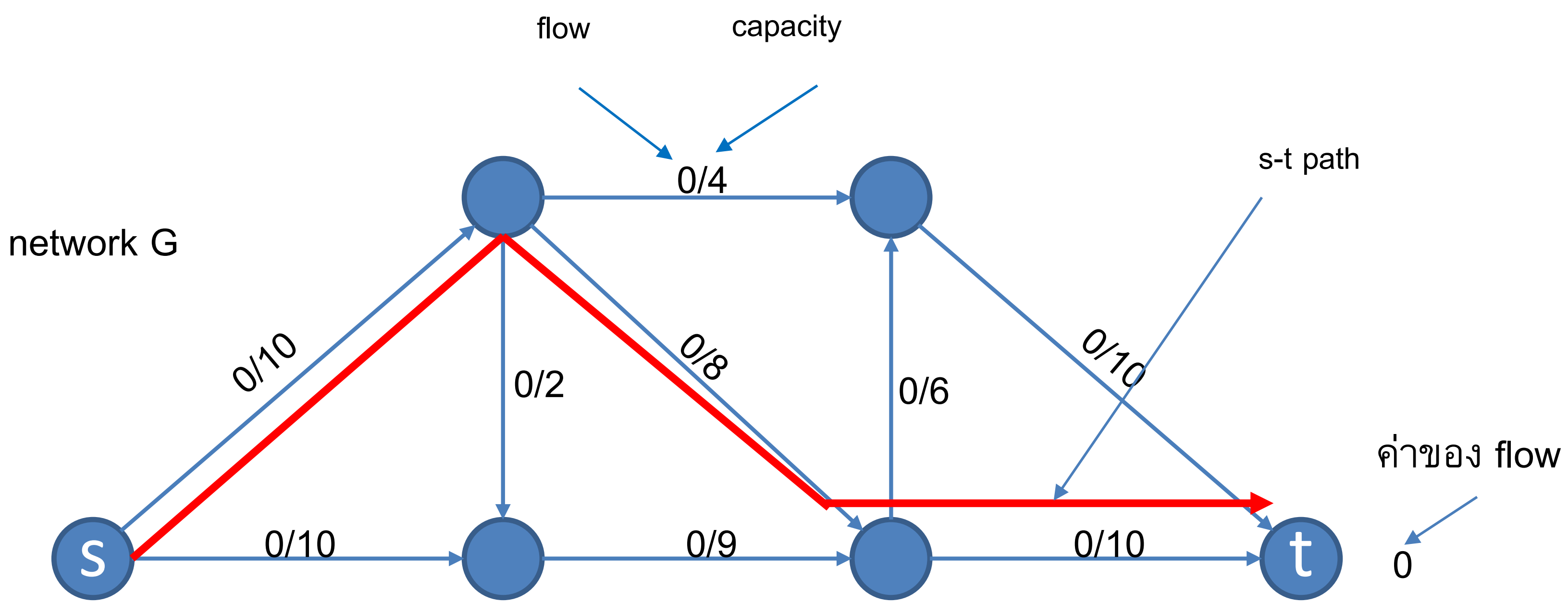
Greedy algorithm (ลองอีกตัวอย่าง)

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



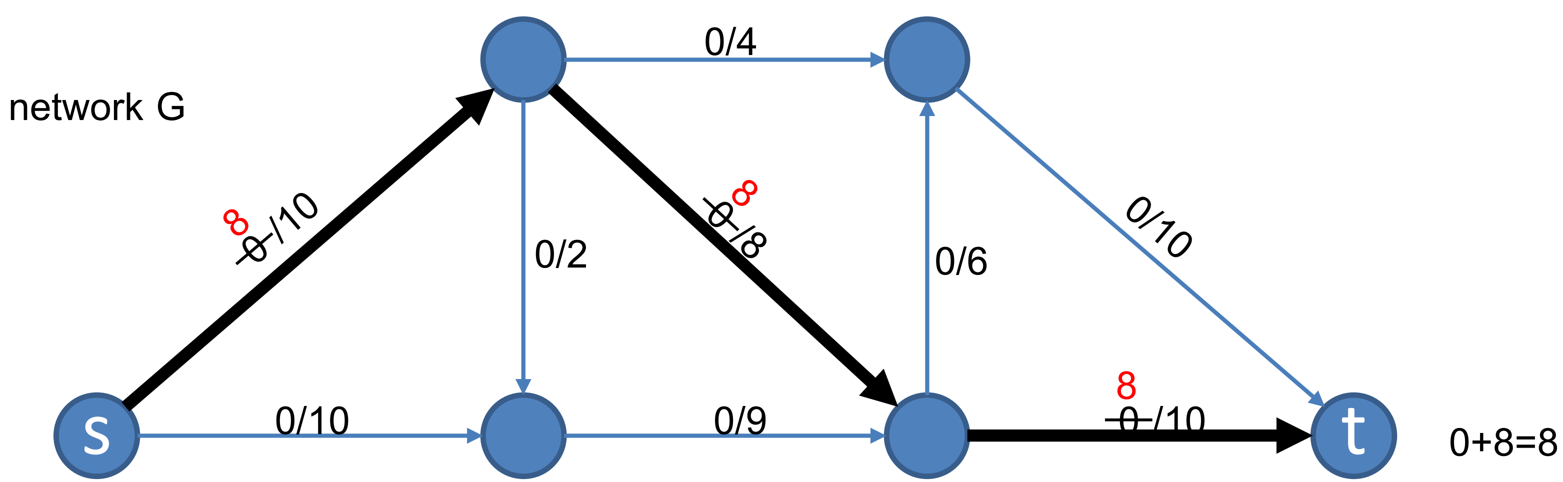
Greedy algorithm (ลอง)

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



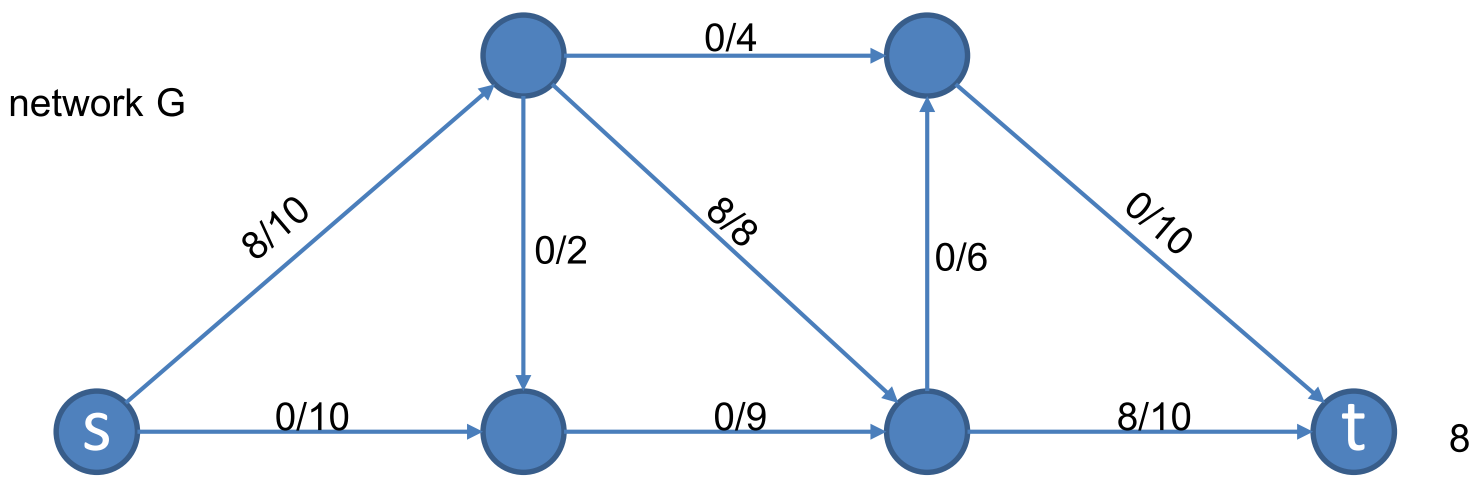
Greedy algorithm (ลอง)

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



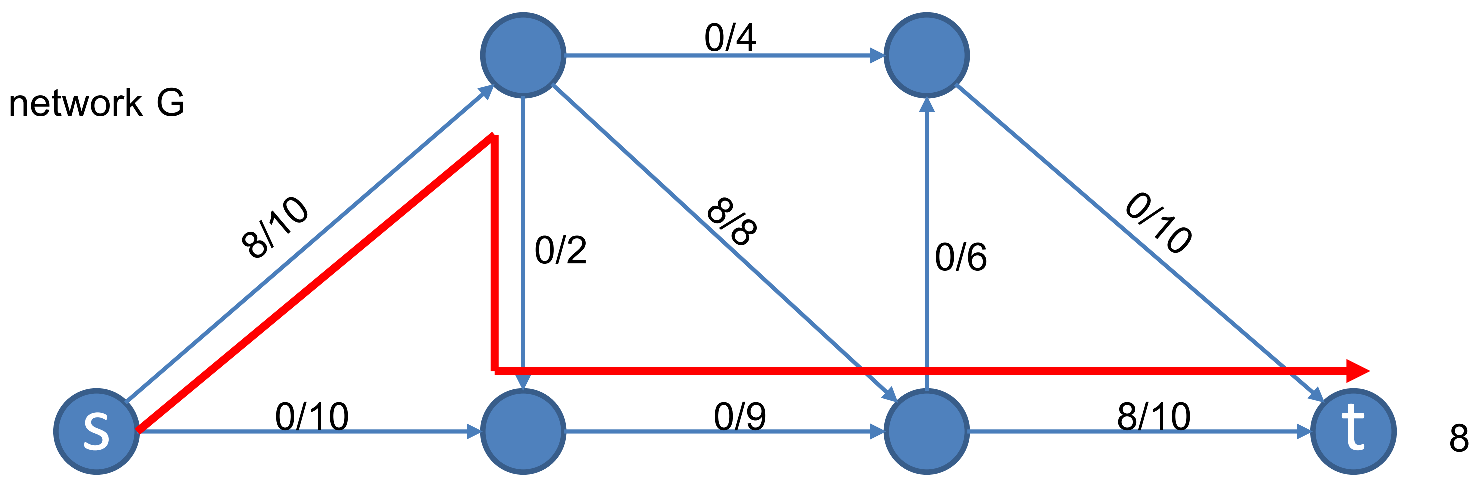
Greedy algorithm (ลอง)

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



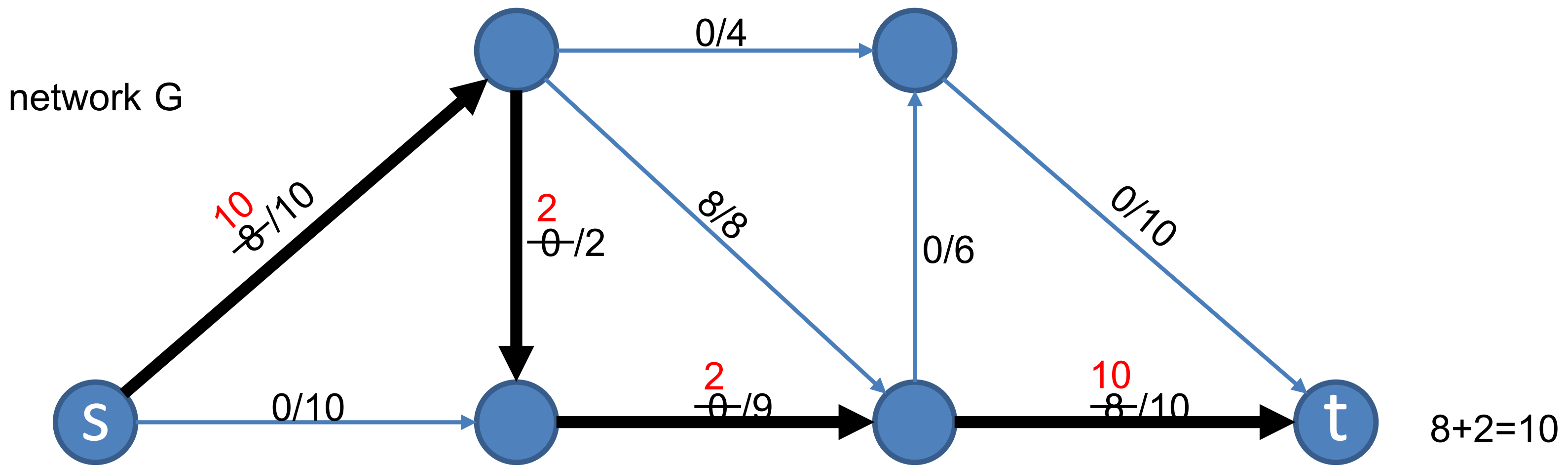
Greedy algorithm (ลอง)

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



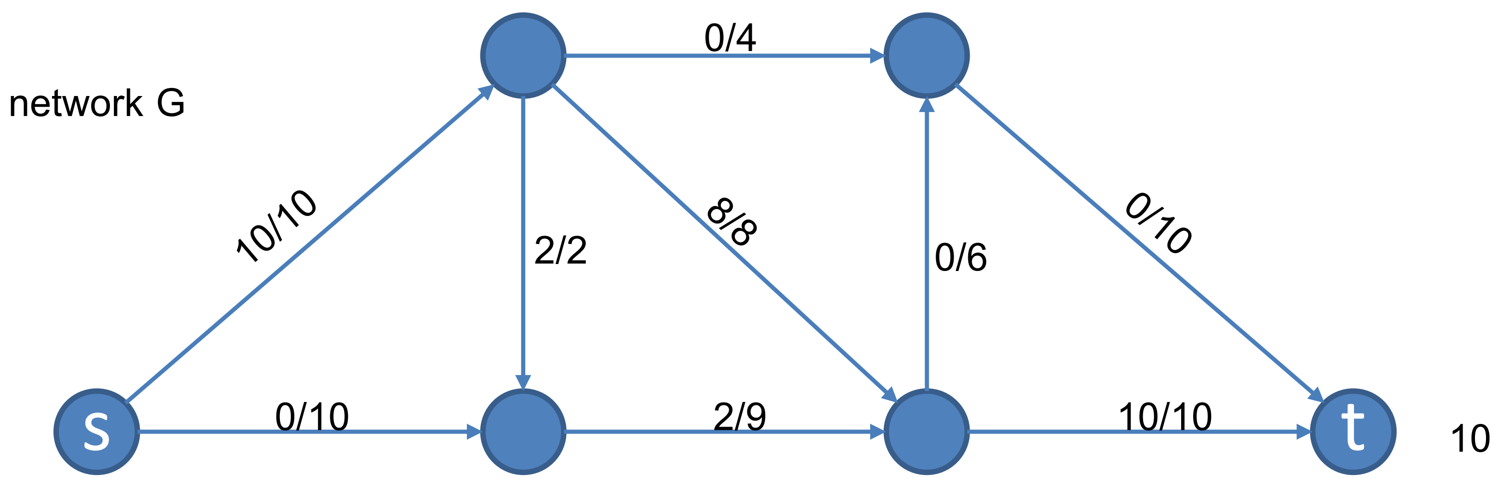
Greedy algorithm (ลอง)

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



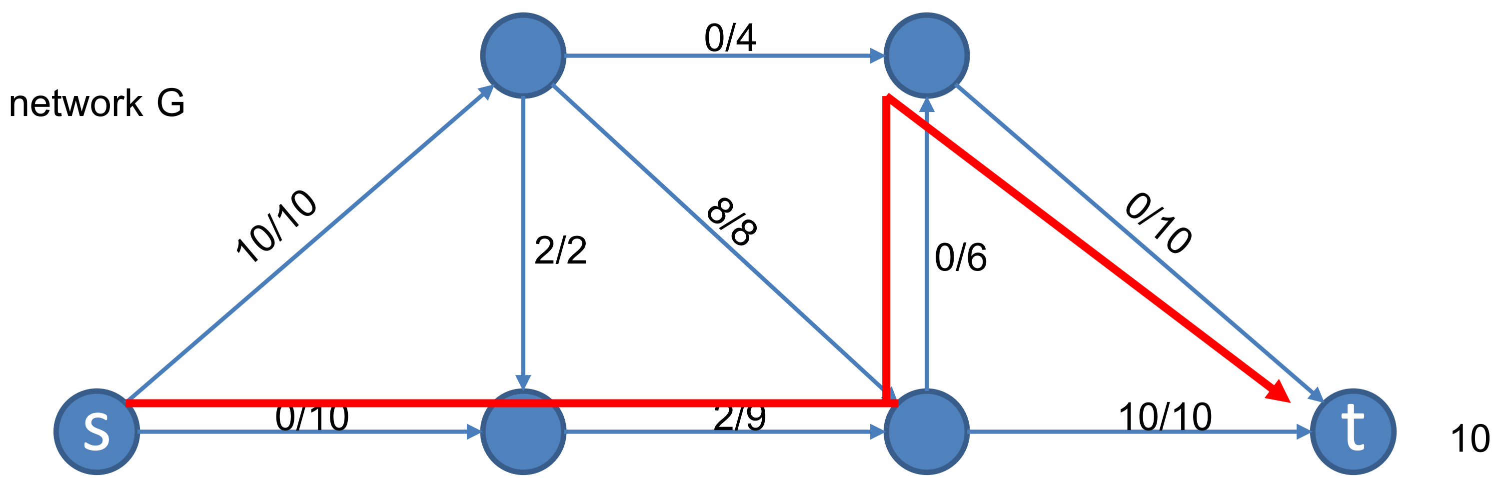
Greedy algorithm (ลอง)

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



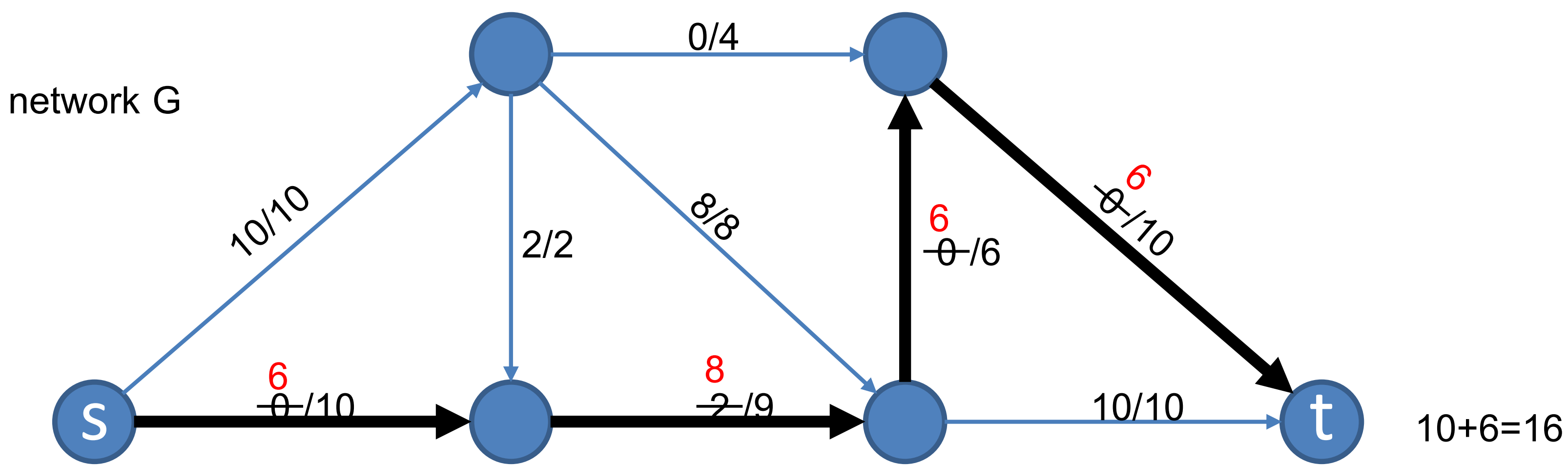
Greedy algorithm (ลอง)

- 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)

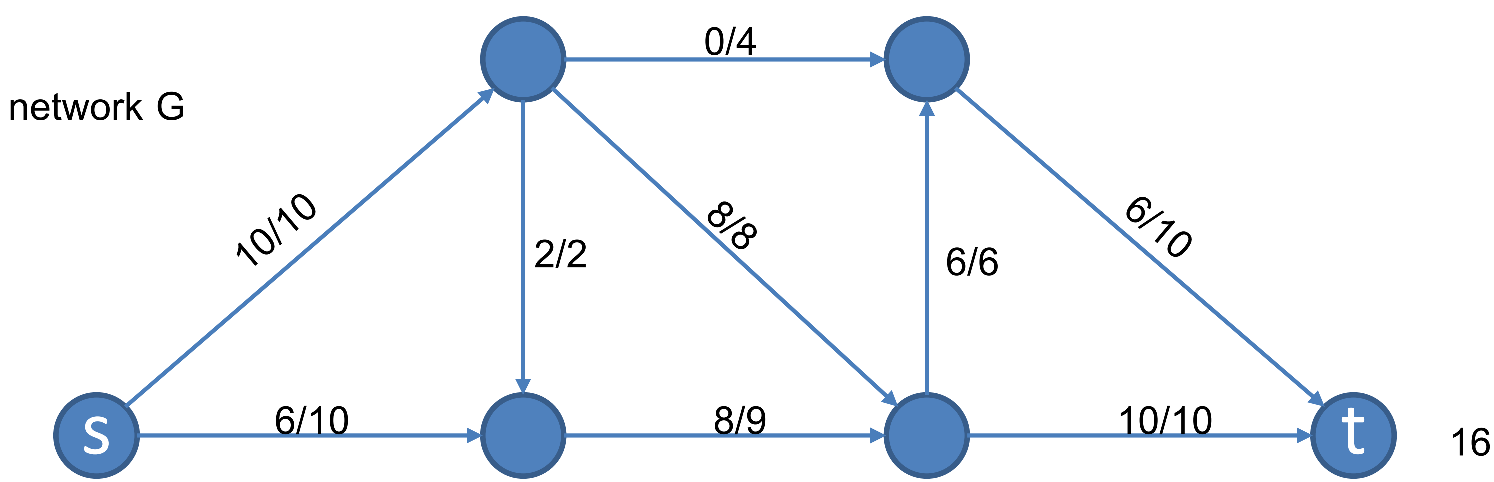
- เริ่มต้นให้ทุกเส้นเชื่อม $e \in E$ มีค่า f(e) = 0
- หา s–t path P ที่แต่ละเส้นเชื่อมมี $f(e) < c(e)$
- เพิ่ม (augment) flow ไปตามเส้นทาง P
- ทำซ้ำจนทำไม่ได้



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

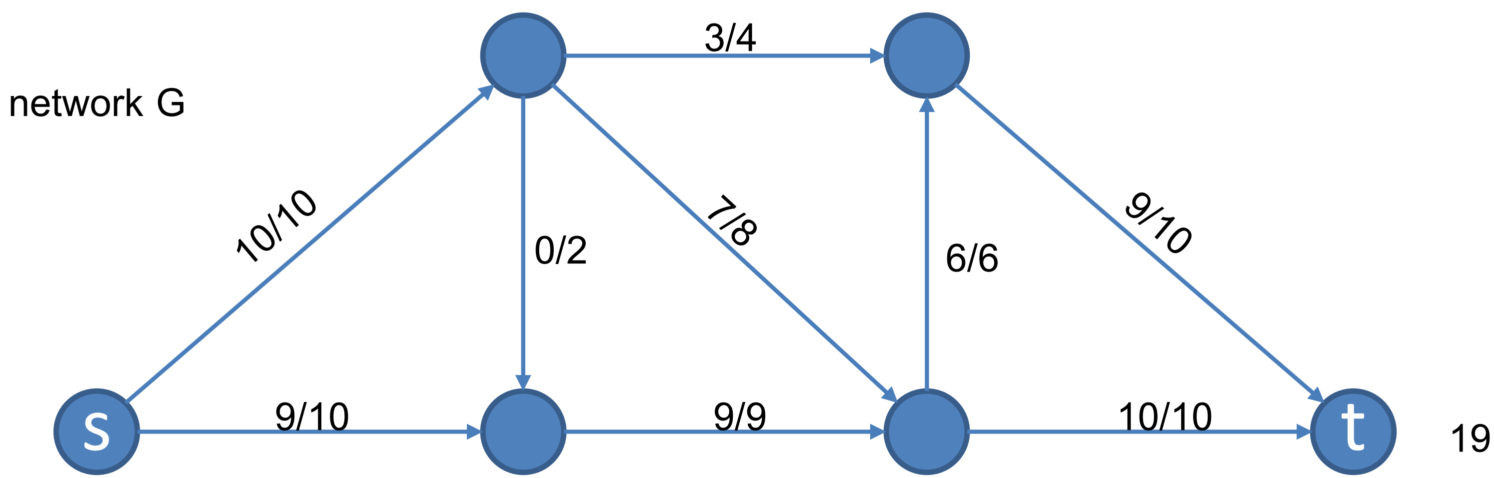
ทำเสร็จด้วย flow = 16



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 ไม่ work

- Greedy ไม่ work ในบางกรณี
- ทำไม
- เพราะว่าเวลาเราเลือกเส้นทาง st path แล้ว เติม flow แล้ว แต่ผิด เราไม่สามารถยกเลิกหรือเปลี่ยนแปลงได้
- ทำอย่างไรดี
- มีคนเสนอ graph ใหม่ที่เรียกว่า กราฟคงเหลือ Residual Graph

Residual graph

- กำหนดให้ความจุคงเหลือ (residual capacity) ของเส้นเชื่อม (u,v) ในเครือข่ายที่มีการไหลคือ

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

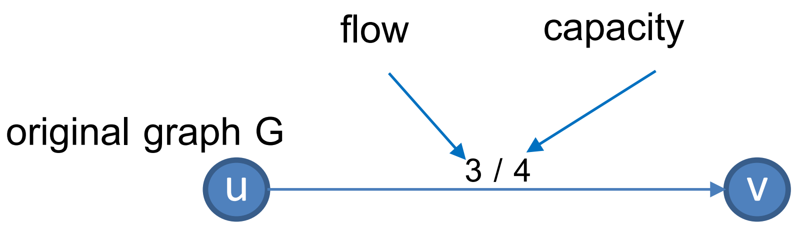
- กำหนดให้กราฟคงเหลือ (Residual graph) G’ ของ G ที่มีการไหล f คือ
  - กราฟที่มีโหนดเหมือน G
  - มีเส้นเชื่อม (u,v) ที่มีความจุ $c_f(u,v)$ ถ้า $c_f(u,v) > 0$
  - มีเส้นเชื่อม (v,u) ที่มีความจุ $f(u,v)$ ถ้า $f(u,v) > 0$

นั่นคือจะมีเส้นเชื่อม weight เท่า flow ที่ผ่านสวนทาง ส่วนเส้นเชื่อมปกติ weight เท่าที่เหลือ

Residual graph

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

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

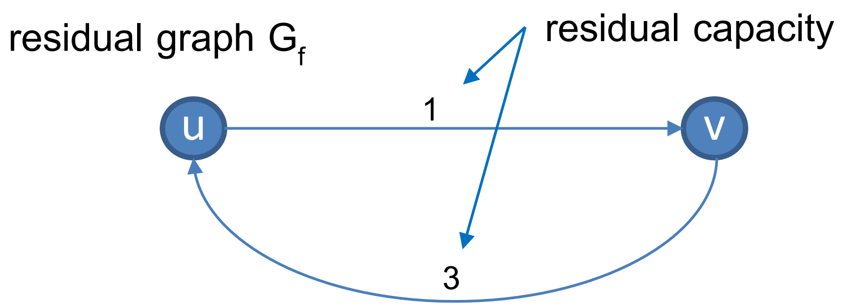


Residual edge ย้อนกลับกับ flow ที่ถูกส่ง

- $e = (u, v)$ และ $e^R = (v, u)$

- Residual capacity:

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



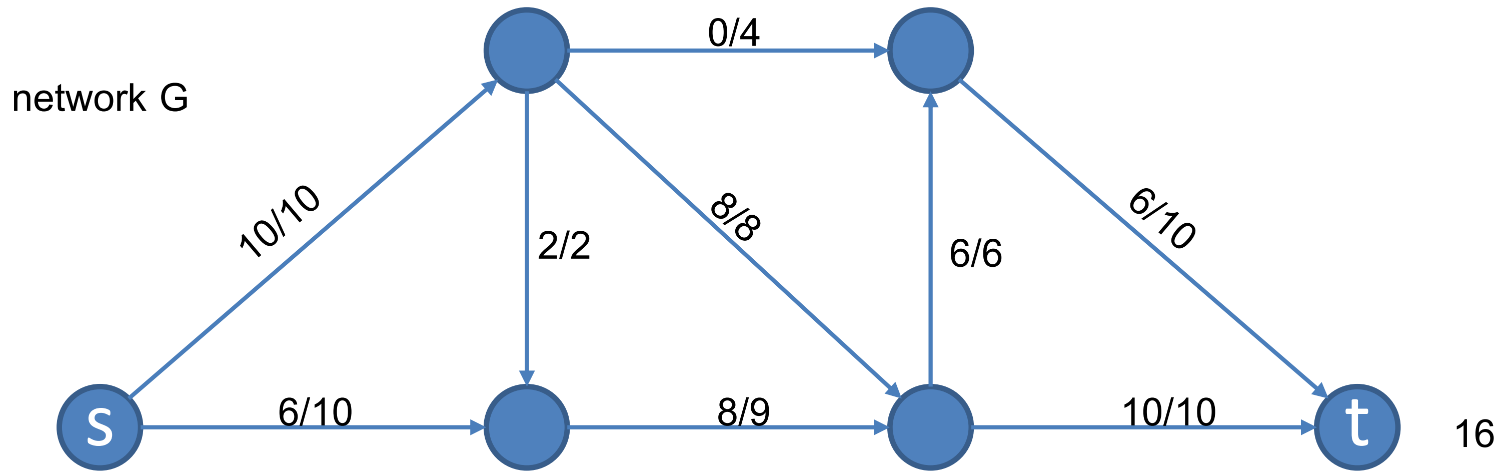
Augmenting path

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

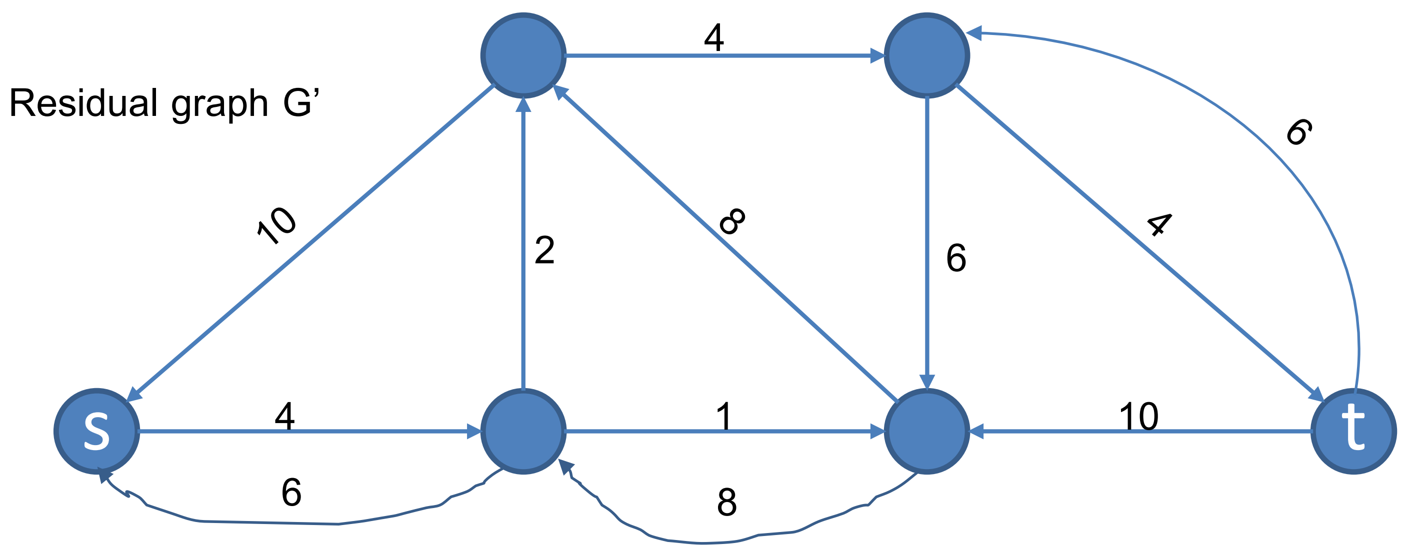
นิยาม bottleneck capacity ของ augmenting P คือ minimum residual capacity ของเส้นเชื่อมใดๆ ใน P

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

$$
\begin{array}{l}
AUGMENT(f, c, P) \\
b = bottleneck\ capacity\ of\ path\ P \\
FOREACH\ edge\ e \in P \\
\quad IF(e \in E)\ \ f(e) = f(e) + b \\
\quad ELSE \qquad f(e^R) = f(e^R) - b \\
RETURN\ f
\end{array}
$$



network G
0/4
10/10
2/2
8/8
6/6
6/10
s
6/10
8/9
10/10
t
16



Residual graph G'
4
6
10
2
8
6
4
s
4
1
10
t
6
8

Ford Fulkerson's algorithm

- หนึ่งในวิธีแก้ปัญหา Max Flow คือ algorithm ของ Ford Fulkerson ซึ่งสร้างโดย Lester Randolph Ford คนเดียวกับ Bellman Ford's algorithm และ Delbert Ray Fulkerson

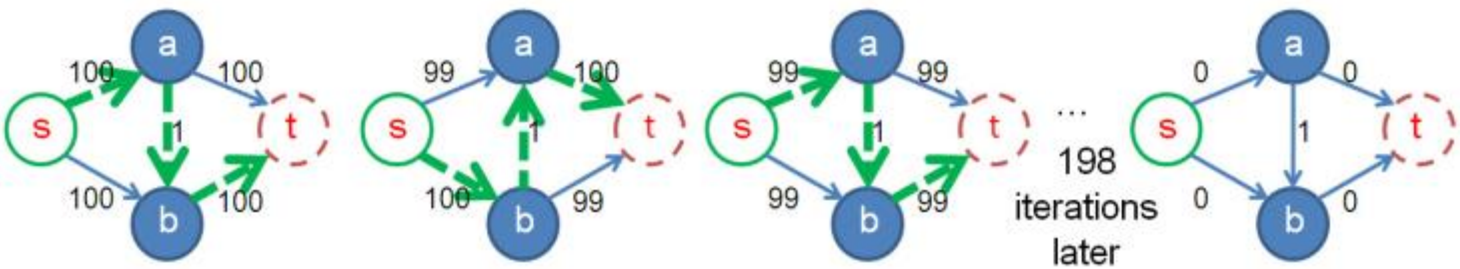
```
setup directed residual graph with edge capacity=original graph weights
mf = 0
while (there exists an augmenting path p from s to t) {
  // p is a path from s to t that pass through +ve edges in residual graph
  augment/send flow f along the path p (s -> ... -> i -> j -> ... t)
    1. find f, the minimum edge weight along the path p
    2. decrease capacity of forward edges(e.g. i->j) along path p by f
    3. increase capacity of backward edges(e.g. j->i) along path p by f
  mf += f // we can send a flow of size f from s to t, increase mf
}
output mf
```

- Ford Fulkerson's algorithm ทำงานเป็นรอบ โดยในแต่ละรอบจะหา augmenting path p: path จาก s ไป t ที่ผ่านเส้นเชื่อมที่มีน้ำหนักเป็นบวกใน residual graph
- หลังจากหา augmenting path p ที่มี f เป็นน้ำหนักของเส้นเชื่อมที่ต่ำที่สุดใน p (bottleneck edge) Ford Fulkerson's algorithm จะทำขั้นตอนสำคัญสองอย่างคือ ลดความจุของเส้นเชื่อมขาไป (i–>j) และเพิ่มความจุของเส้นเชื่อมขากลับ (j–>i) ตาม path p ด้วยค่า f
- หลังจากนั้นจะทำจนกระทั่งไม่สามารถหา augmenting path จาก s ไป t ได้อีกนั่นหมายความว่า total flow ที่ได้เป็น maximum flow

- เหตุผลในการลดค่าความจุของเส้นเชื่อมขาไปนั้น ตรงไปตรงมาเนื่องจากเราส่ง flow ผ่าน augmenting path p ซึ่งจะไปลดความจุที่เหลือ (remaining residual capacities) ของ forward edges ที่ถูกใช้ใน p

- เหตุผลในการเพิ่มค่าความจุของเส้นเชื่อมขากลับ คือทำให้ในรอบต่อไปในอนาคต เราสามารถยกเลิกบางส่วนของความจุที่ถูกใช้ไปแล้วจาก forward edge ที่ถูกใช้ไปไม่ถูกต้องก่อนหน้านี้ได้

- มีวิธีการหลายอย่างในการหา augmenting s–t path ในที่นี้เราจะใช้ DFS
- Ford Fulkerson's algorithm ที่ implement โดยใช้ DFS อาจจะทำงานในเวลา O(|f*|E) เมื่อ |f*| คือค่า Maximum flow ทั้งนี้เป็นเพราะเราอาจจะมีกราฟดังนี้



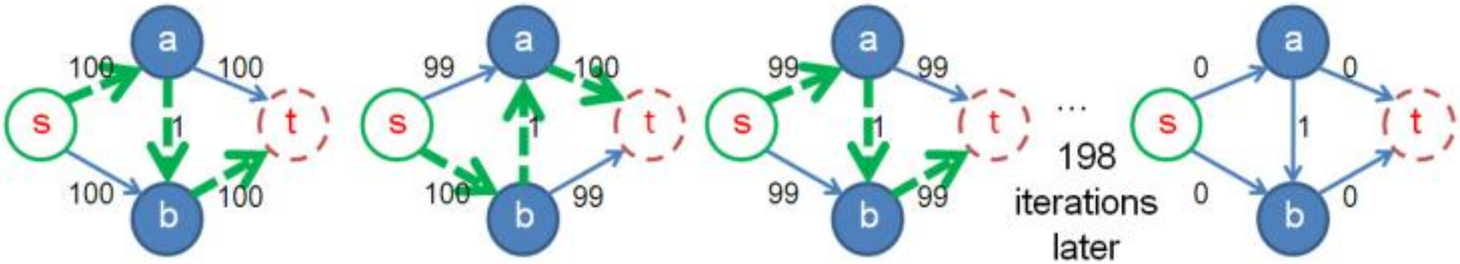
- มี augmenting path s–>a–>b–>t และ s–>b–>a–>t ที่ลดความจุของเส้นเชื่อมขาไปเพียง 1 หน่วย ทำให้ในกรณีแย่สุดจะต้องทำงานอย่างน้อย |f*| รอบและ DFS ทำงานใน O(E)ซึ่งถ้า |f*| มีค่ามากนี่ไม่ดีเลย

Edmonds Karp's Algorithm

- วิธีการสร้างที่ดีกว่า Ford Fulkerson's algorithm คือการใช้ BFS ในการหา shortest path ในด้านจำนวนชั้นที่ห่างกันของ s และ t อัลกอริทึมนี้ค้นพบโดย Jack Edmonds และ Richard Manning Karp เลยชื่อ Edmonds Karp algorithm

- มันทำงานใน $O(VE^2)$ เนื่องจากมันสามารถถูกพิสูจน์ได้ว่าหลังจาก $O(VE)$ BFS รอบแล้วทุก augmenting path จะเต็มหมดและเนื่องจาก BFS ทำงานใน $O(E)$ ใน flow graph เวลาทั้งหมดเลยเป็น $O(VE^2)$

- Edmond Karp's algorithm นั้นใช้ 2 s–t path นั่นคือ s–>a–>t (2 hop และส่ง 100 หน่วย) และ s–>b–>t (2 hop และส่ง 100 หน่วย)



- code ของ Edmond Karp's algorithm ต่อไปนี้ เราจะใช้ Adjacency Matrix ที่ชื่อว่า res ที่มีขนาด O(V^2) ที่เก็บ residual capacities ของแต่ละเส้นเชื่อม ใน version นี้ทำงานใน O(VE) BFS รอบ x O(V^2) ต่อ BFS เนื่องจากใช้ Adjacency Matric = O(V^3E) ซึ่งทำงานทันสำหรับกราฟขนาดเล็ก

```
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
#define MAX_V 40
#define INF 1000000000
int res[MAX_V][MAX_V], mf, f, s, t; // global variables
vi p;   // p stores the BFS spanning tree from s
void augment(int v, int minEdge) {
// traverse BFS spanning tree from s to t
  if (v == s) { f = minEdge; return; }
// record minEdge in a global variable f
  else if (p[v] != -1) {
    augment(p[v], min(minEdge, res[p[v]][v])); //recursive
    res[p[v]][v] -= f;
    res[v][p[v]] += f;
  }    // update
}
```

```
int main() {
  int V, k, vertex, weight;
  scanf("%d %d %d", &V, &s, &t);
  memset(res, 0, sizeof res);
  for (int i = 0; i < V; i++) {
    scanf("%d", &k);
    for (int j = 0; j < k; j++) {
      scanf("%d %d", &vertex, &weight);
      res[i][vertex] = weight;
    }
  }
```

```
  mf = 0; // mf stands for max_flow
  while (1) {      // O(VE^2) (actually O(V^3E) Edmonds Karp's algorithm
    f = 0;
    // run BFS, compare with the original BFS shown in Section 4.2.2
    vi dist(MAX_V, INF); dist[s] = 0; queue<int> q; q.push(s);
    p.assign(MAX_V, -1);           // record the BFS spanning tree, from s to t!
    while (!q.empty()) {
      int u = q.front(); q.pop();
      if (u == t) break;      // immediately stop BFS if we already reach sink t
      for (int v = 0; v < MAX_V; v++)                 // note: this part is slow
        if (res[u][v] > 0 && dist[v] == INF)
          dist[v] = dist[u] + 1, q.push(v), p[v] = u;
    }
    augment(t, INF);     // find the min edge weight `f' along this path, if any
    if (f == 0) break;      // we cannot send any more flow (`f' = 0), terminate
    mf += f;                // we can still send a flow, increase the max flow!
  }
  printf("%d\n", mf);                // this is the max flow value
  return 0;
}
```

https://bit.ly/2Wtn5g0

ก่อนเรียก augment(t,INF) นั้นมีการสร้าง BFS จาก s ไป t ก่อน โดยพิจารณาว่ามีเส้นเชื่อมไหมจาก res[u][v]>0 ถึงไปได้

```
vi dist(MAX_V, INF);
dist[s] = 0;
queue<int> q;
q.push(s);
p.assign(MAX_V, -1);          // record the BFS spanning tree, from s to t!
while (!q.empty()) {
    int u = q.front(); q.pop();
    if (u == t) break;     // immediately stop BFS if we already reach sink t
    for (int v = 0; v < MAX_V; v++)                // note: this part is slow
        if (res[u][v] > 0 && dist[v] == INF)
          dist[v] = dist[u] + 1, q.push(v), p[v] = u;
}
```

- จะหยุดทำเมื่อถึง t แล้ว สิ่งที่ได้คือ ลำดับ parent จาก t ย้อนไปถึง s จากนั้นจึงเรียก augment(t,INF)

```
void augment(int v, int minEdge) {
// traverse BFS spanning tree from s to t
  if (v == s) { f = minEdge; return; }
// record minEdge in a global variable f
  else if (p[v] != -1) {
    augment(p[v], min(minEdge, res[p[v]][v])); //recursive
    res[p[v]][v] -= f;
    res[v][p[v]] += f;
  }    // update
}
```

- augment จะทำการหาค่า minEdge จาก t ย้อนไป s
- จากนั้น updateค่า res[u][v] และ res[v][u] ของ path จาก t ย้อนไป s

- ตัวอย่างข้อมูล

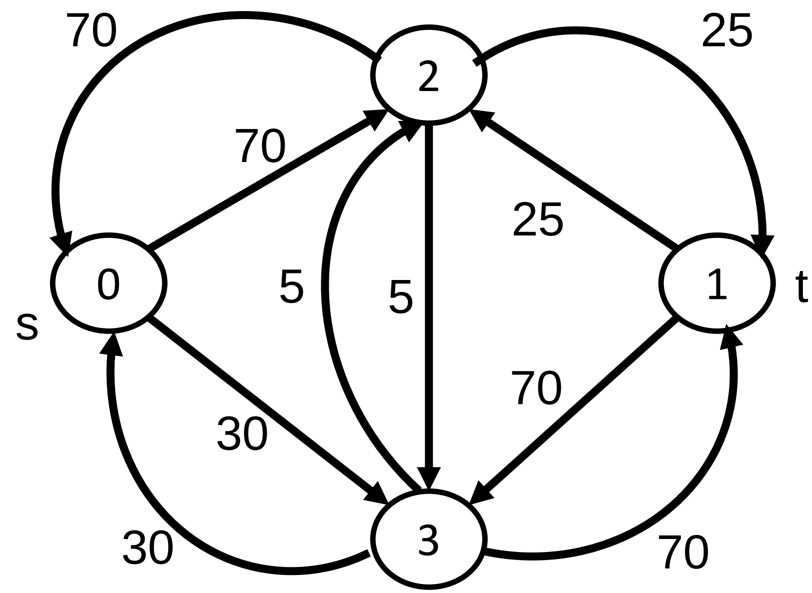
4 0 1

2 2 70 3 30

2 2 25 3 70

3 0 70 3 5 1 25

3 0 30 2 5 1 70



- ตัวอย่างข้อมูล

5 1 0

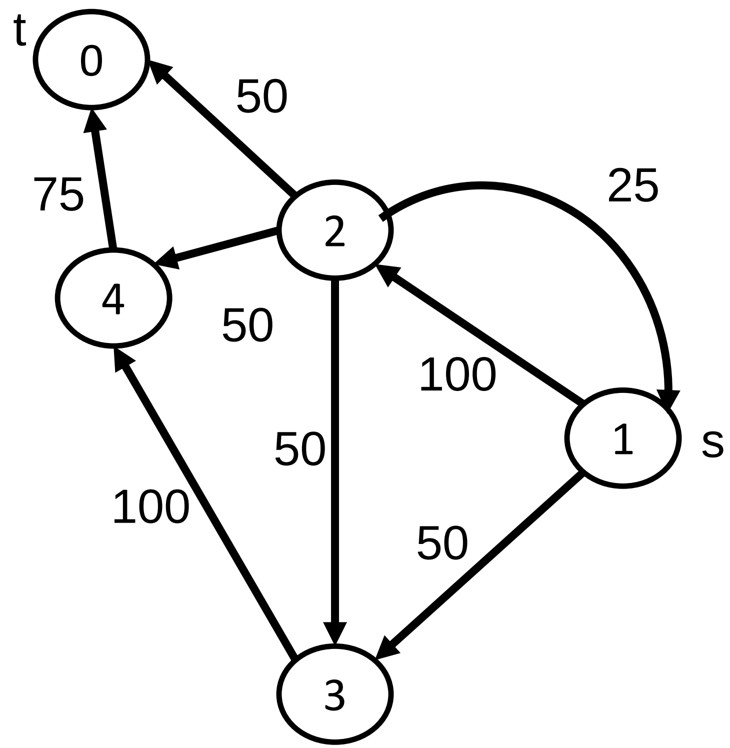
0

2 2 100 3 50

3 3 50 4 50 0 50

1 4 100

1 0 75



Flow graph modeling

- เรามี code ของ Edmond Karp's algorithm แล้ว ซึ่ง code นี้สามารถแก้ปัญหา Network Flow แบบธรรมดาได้ เวลาเจอโจทย์สิ่งที่เราต้องทำคือ
- มองก่อนว่าปัญหานี้ว่าเป็น Network Flow Problem หรือไม่ (จะคล่องขึ้นถ้าแก้ปัญหา Network Flow Problem บ่อยๆ)
- สร้าง flow graph ที่เหมาะสม (ปรับจาก code ก่อนหน้า กำหนด residual matrix, s และ t ให้เหมาะสม)
- รัน Edmond Karp's algorithm