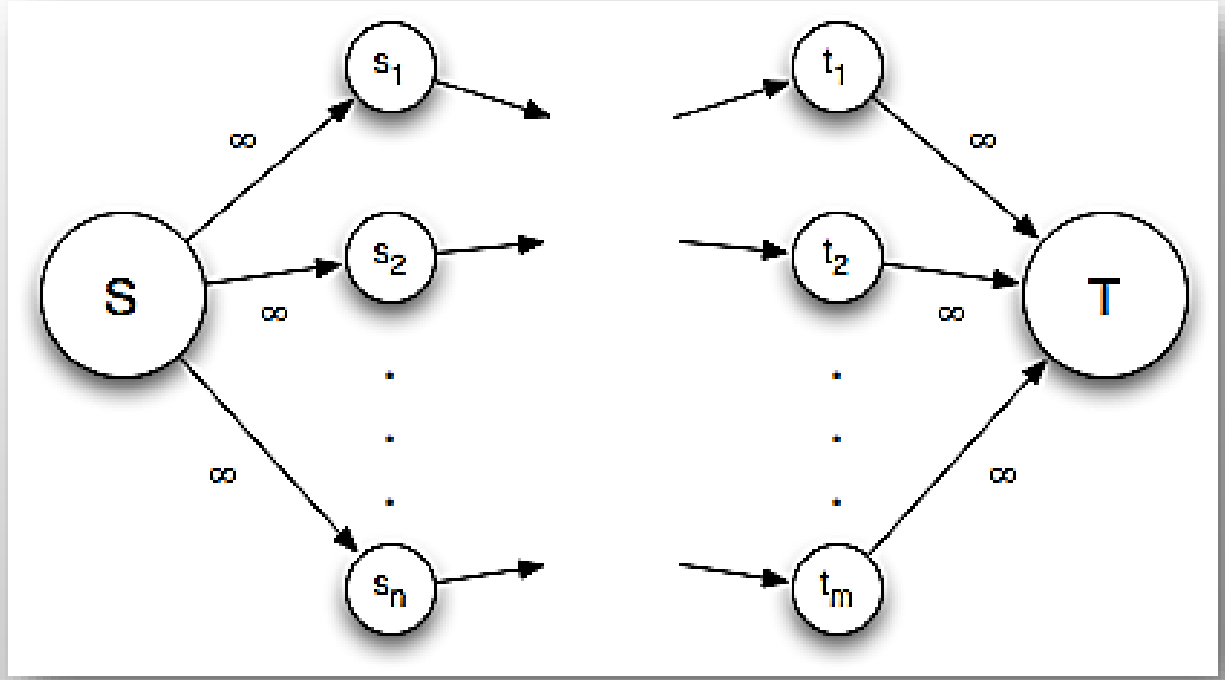
Bölüm 4: Çizge Algoritmaları

Algoritmalar

Maksimum Akış (Maximum Flow)

- Çizge içinde bir kaynak (*source*) ve bir hedef (*sink*) düğüm bulunur.
- Çizge üzerindeki kenarlara *kapasite* değerleri atanır.
- Kaynaktan hedefe maksimum miktarda *akış*ı bulmak amaçlanır.



Artık Ağlar (Residual Networks)

- Akışın hedefe ulaşması için kullanılan kapasiteyi gösterir.
- Bir kenarın kapasitesinden akışın miktarı çıkarılarak bulunur.
- Eğer kenarda daha az akış varsa, kalan kapasite pozitif olur.
- Çizge üzerinde *artış yollarını* gösterir.

Artış Yolları (Augmenting Paths)

- Çizge üzerinde kaynaktan hedefe *ek akış sağlayan* yolları ifade eder.
- Bir artış yolu, kaynaktan hedefe yönlendirilen bir yol olmalıdır.
- Çizge üzerindeki kenarların kapasitelerinden daha az akış taşımalıdır.
- Maksimum akışa ulaşmak için kullanılır.
- Genellikle BFS veya DFS algoritmaları kullanılarak bulunur.

Kesimler (Cuts)

- Bir çizgenin düğümlerini ikiye bölen bir kenar kümesidir.
- Çizgenin bağlantısını keser ve farklı bileşenlere ayırır.
- *Min-cut* (Minimum kesme):
  - En az sayıda kenarı kesecek düğüm kümesi.
- *Max-flow* (Maksimum akış):
  - Çizgeyi kaynak ve hedef arasında böler.

Ağ Akış Algoritmaları (Network Flow)

- Kaynaktan hedefe ağ akışını kısıtlamalar altında optimize eder.
- Amaç, kısıtlamalar altında mümkün olan en fazla akışı sağlamaktır.
- *Ford-Fulkerson*,
  - basit ve anlaşılır, çalışma süresi diğerlerine göre daha uzun.
- *Edmonds-Karp*,
  - *Ford-Fulkerson* algoritmasının gelişmiş hali, çalışma süresi daha kısa.
- *Dinic's*,
  - *Edmonds-Karp* algoritmasından hızlı, bazı durumlarda daha verimli.

Ford Fulkerson

- Ağırlıklı yönlü çizgede iki düğüm arasındaki maksimum akışı bulur.
- Maksimum akış problemi, kaynaktan hedefe belirli kapasiteye sahip yollarla maksimum suyun akışını modelleyen bir çizge problemidir.
- *L.R. Ford Jr.* ve *D.R. Fulkerson* tarafından geliştirilmiştir.
- *Ford-Fulkerson uses the DFS, Edmonds-Karp uses the BFS approach.*

Algoritma İlkeleri

- Ağ, yönlü bir çizge olarak temsil edilir.
- Her kenara bir kapasite değeri atanır.
- Başlangıçta, tüm akışlar sıfır olarak başlatılır.
- Artan yol (*augmenting path*) bulma adımları tekrarlanarak,
  - maksimum akış bulunur.

Algoritma Adımları

- Adım 1: Kaynaktan hedefe artan bir yol bulunur.
- Adım 2: Bulunan artan yol boyunca maksimum akışa izin verilir. Bu, akış ağındaki tüm kenarlarda artışa neden olur.
- Adım 3: Hedefe ulaşılana kadar Adım 1 ve Adım 2 tekrarlanır.

Karmaşıklık Analizi

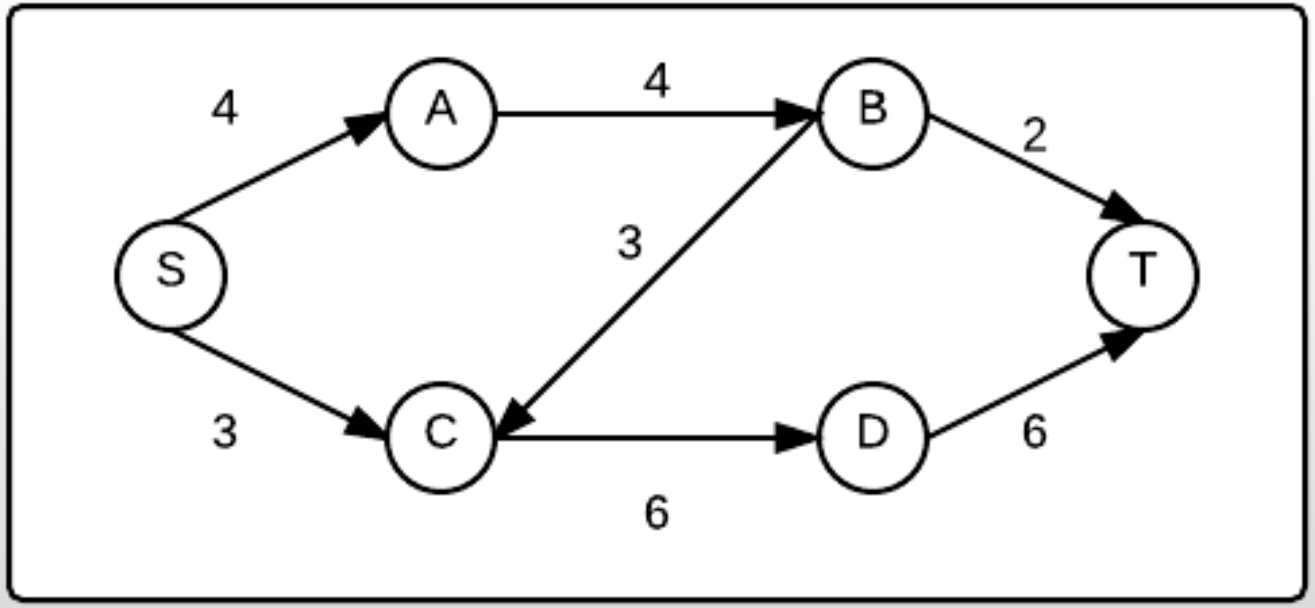
- Algoritmanın çalışma süresi, akış ağındaki yapı ve kapasitelere bağlıdır.
- Artırma yollarının bulunması için yapılan taramalar, ağdaki kenar ve düğüm sayısına bağlıdır.
- Maksimum akış değeri (f), algoritmanın kaç kez artırma yolu bulması gerektiğini belirler.

Karmaşıklık Analizi

- Akış ağının yapısına ve kapasitelerine bağlı olarak değişir.
- En kötü durumda, *O(E f)* karmaşıklığına sahiptir.
  - E kenar sayısını, f maksimum akışı temsil eder.
- *O(E f)*, bir artış yolu bulmak *O(E)*, her artış yolu en az bir birimlik akış artırır, bu nedenle en fazla *O(f)* kez.

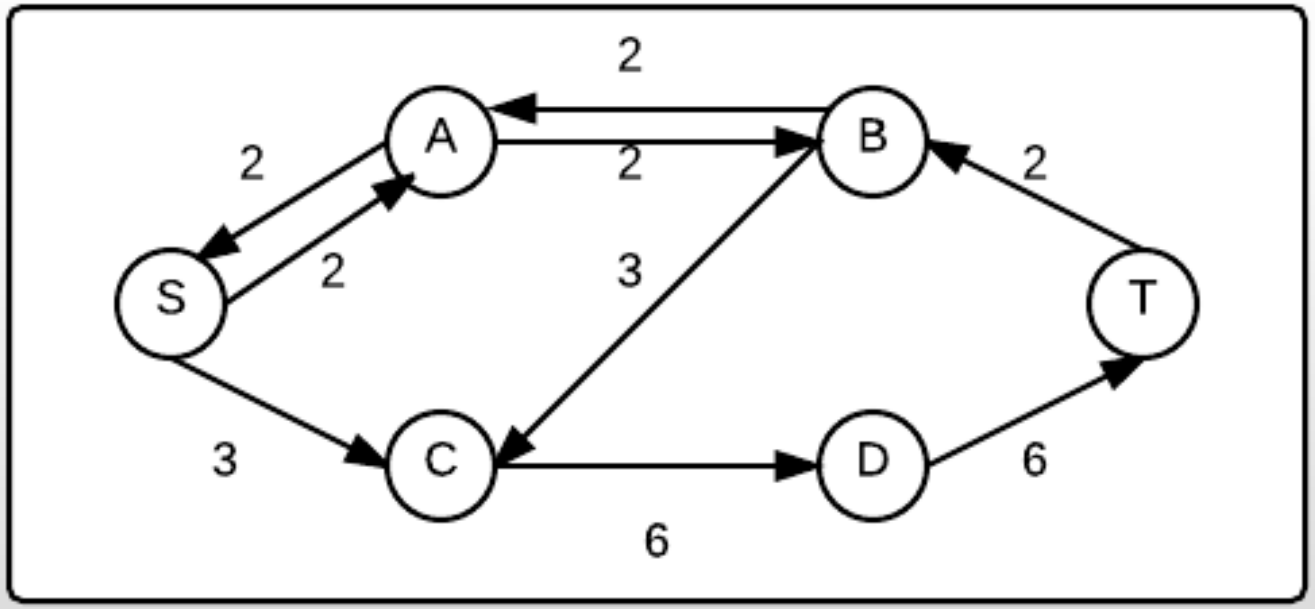
Ford Fulkerson

- Aşağıdaki çizge verilsin. p={S,A,B,T} 2 birim akış



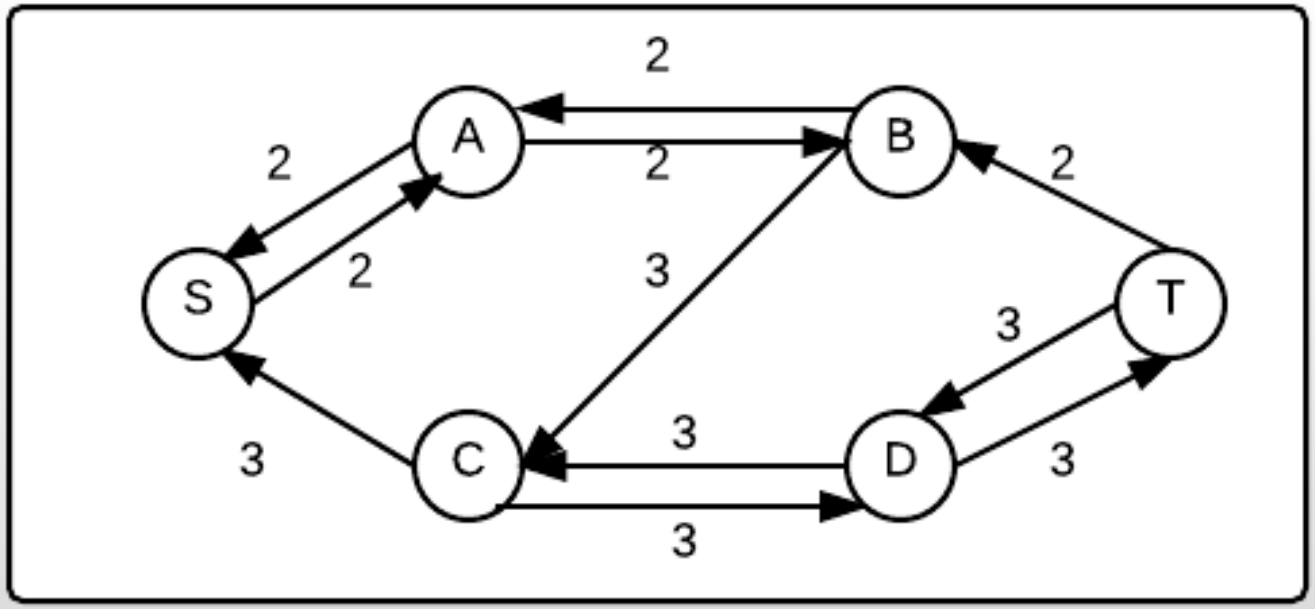
Ford Fulkerson

- p={S,C,D,T} 3 birim akış



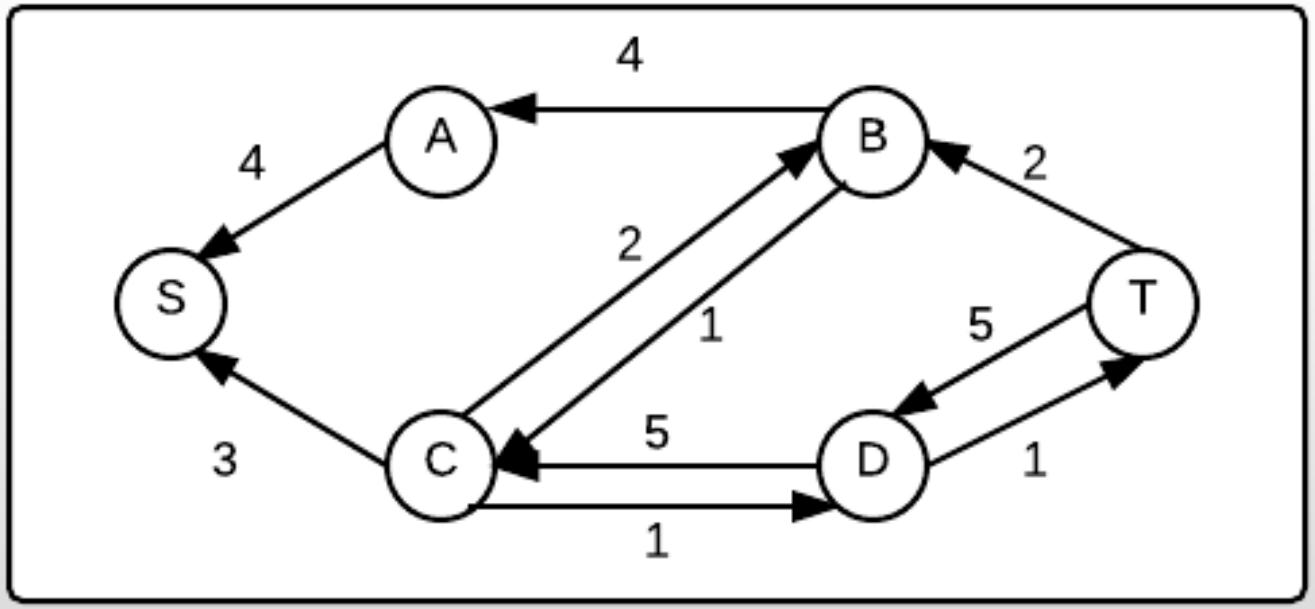
Ford Fulkerson

- p={S,A,B,C,D,T} 2 birim akış

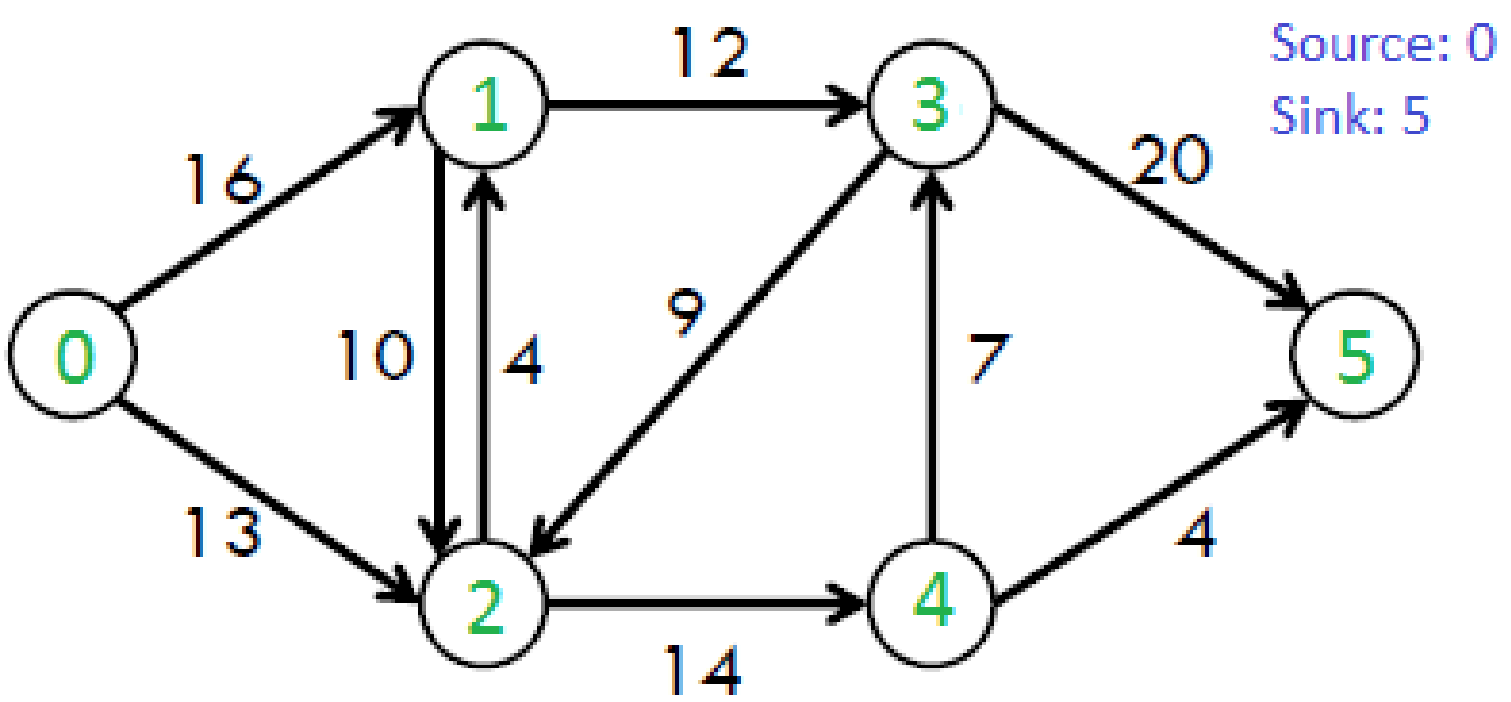


Ford Fulkerson

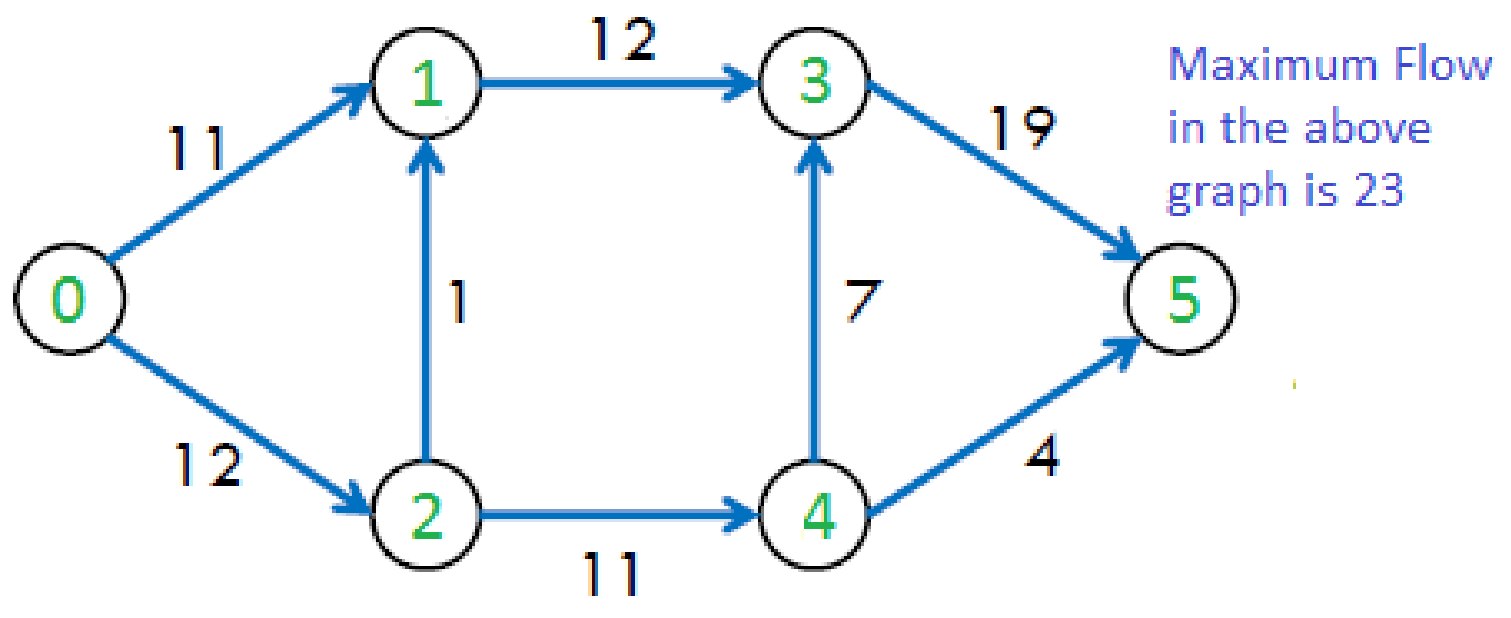
- Maksimum akış 7



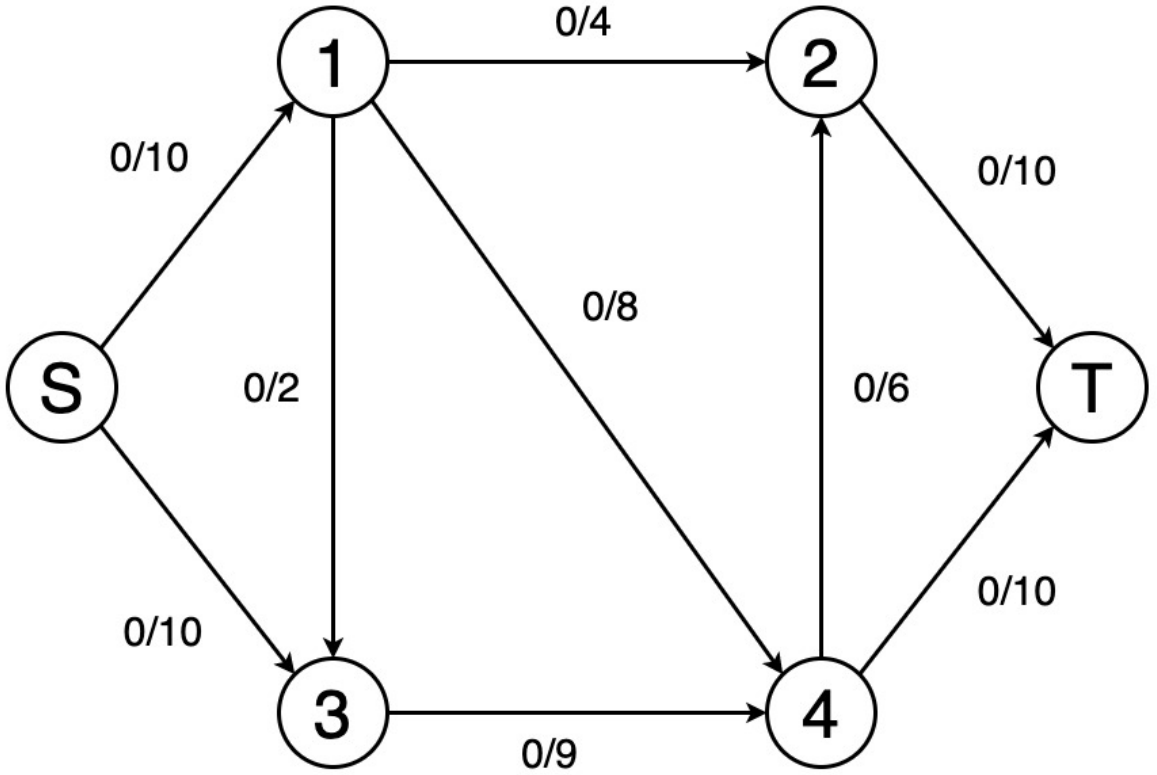
Ford Fulkerson



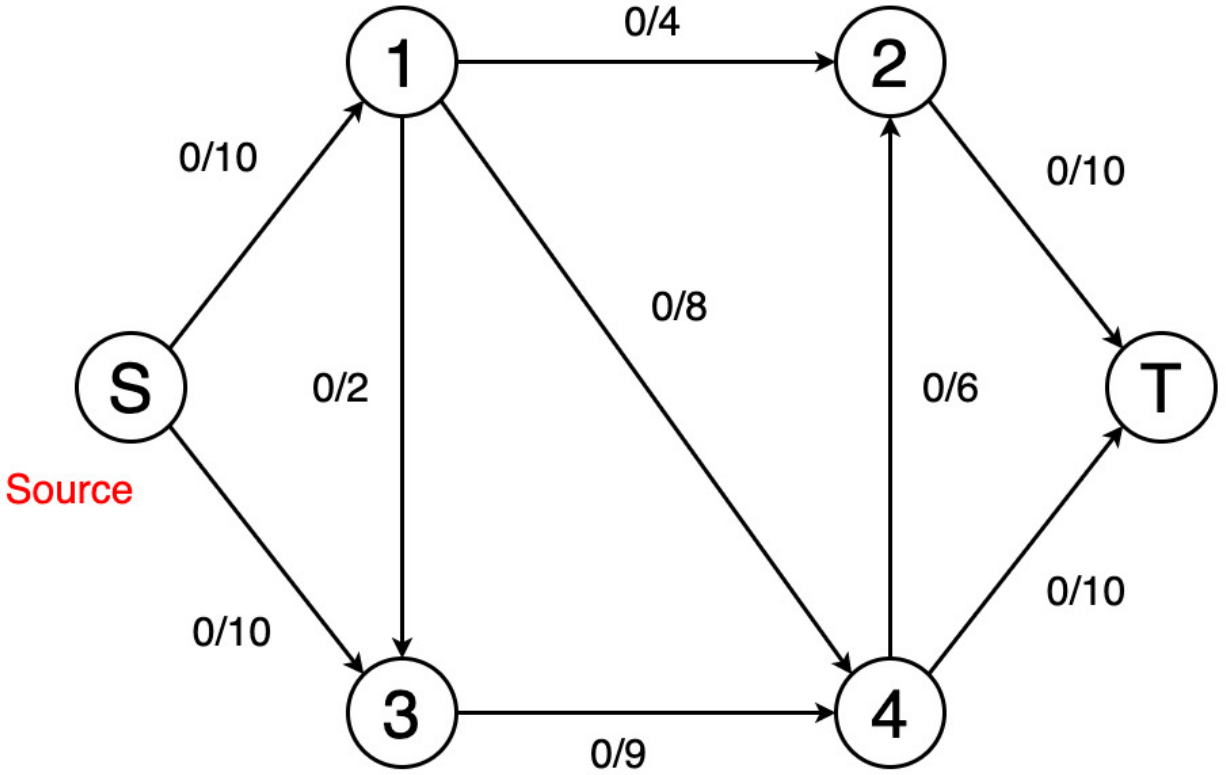
Ford Fulkerson



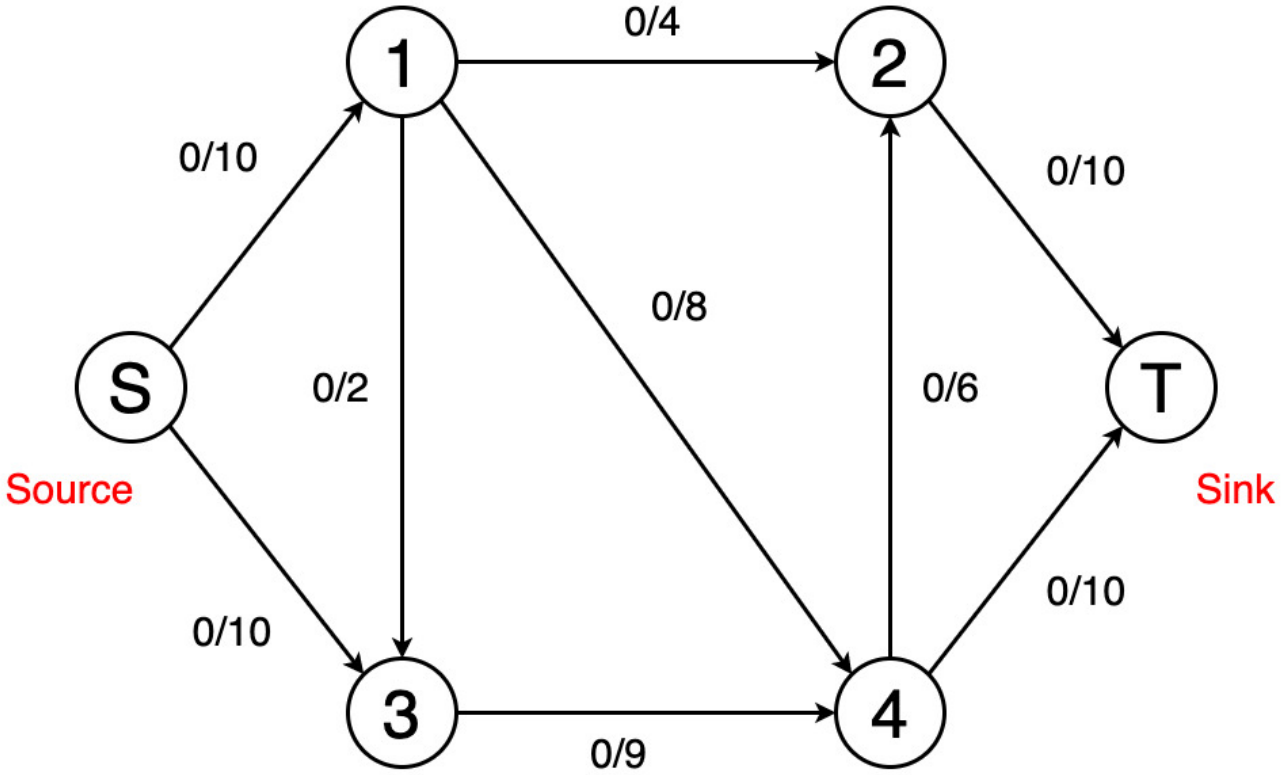
Ford Fulkerson



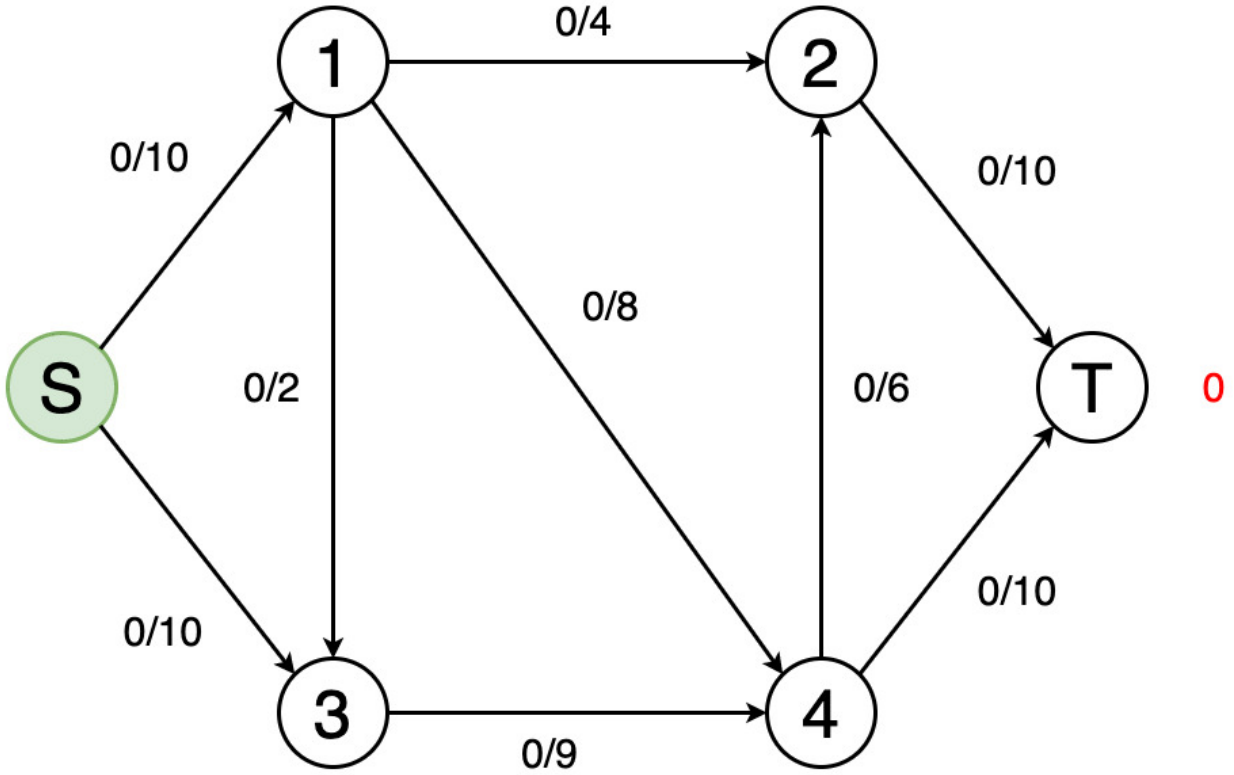
Ford Fulkerson



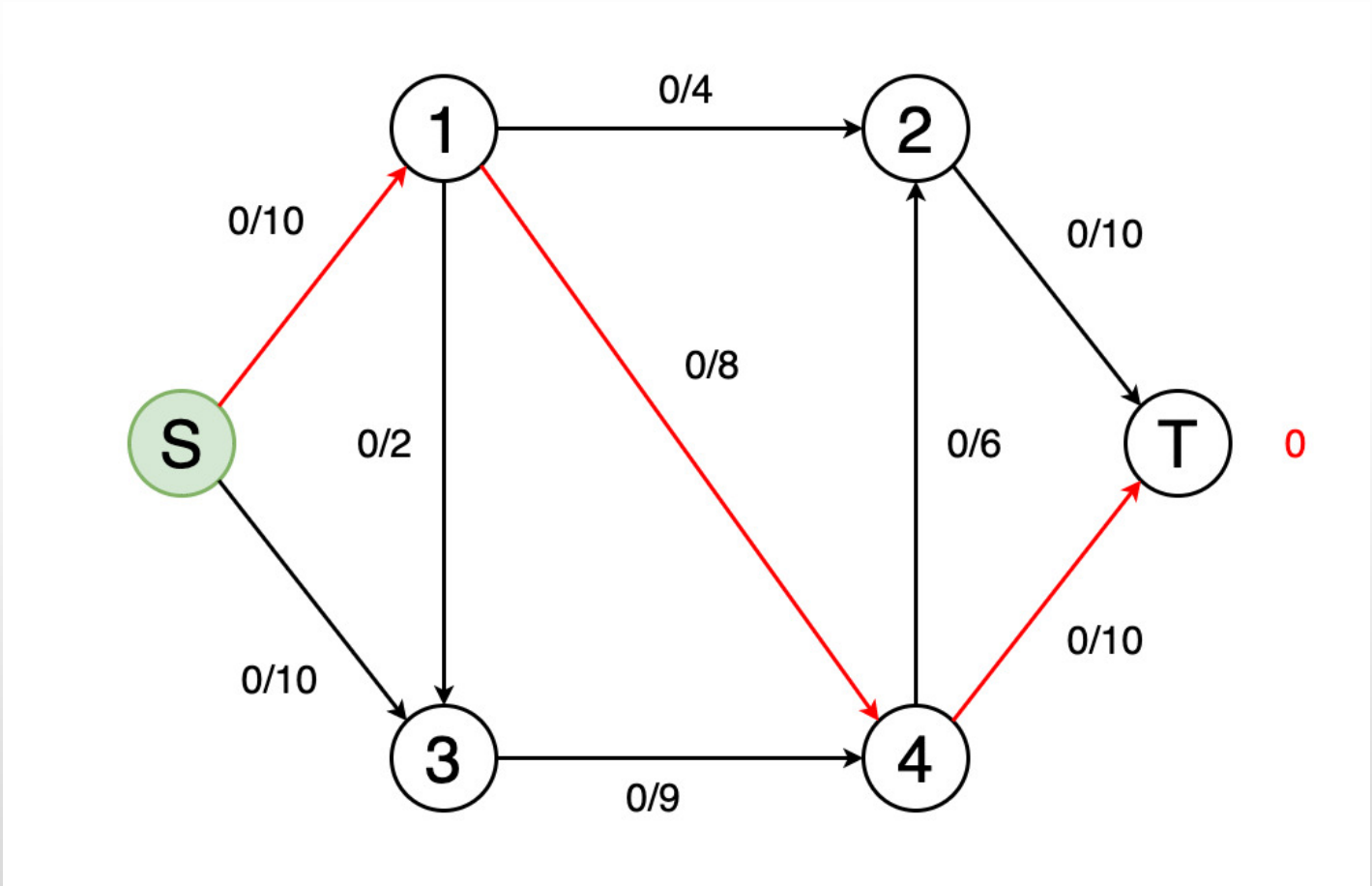
Ford Fulkerson



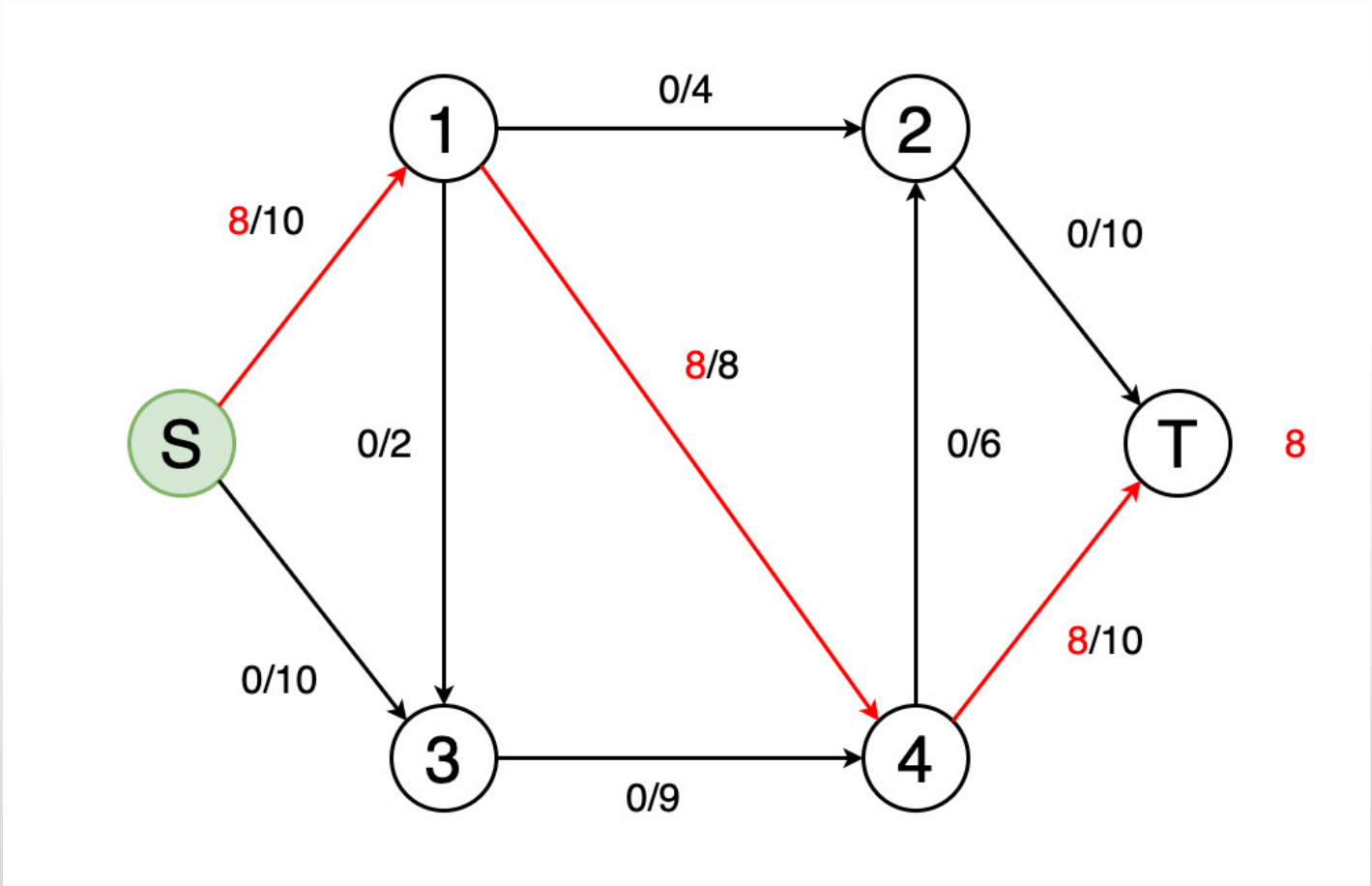
Ford Fulkerson



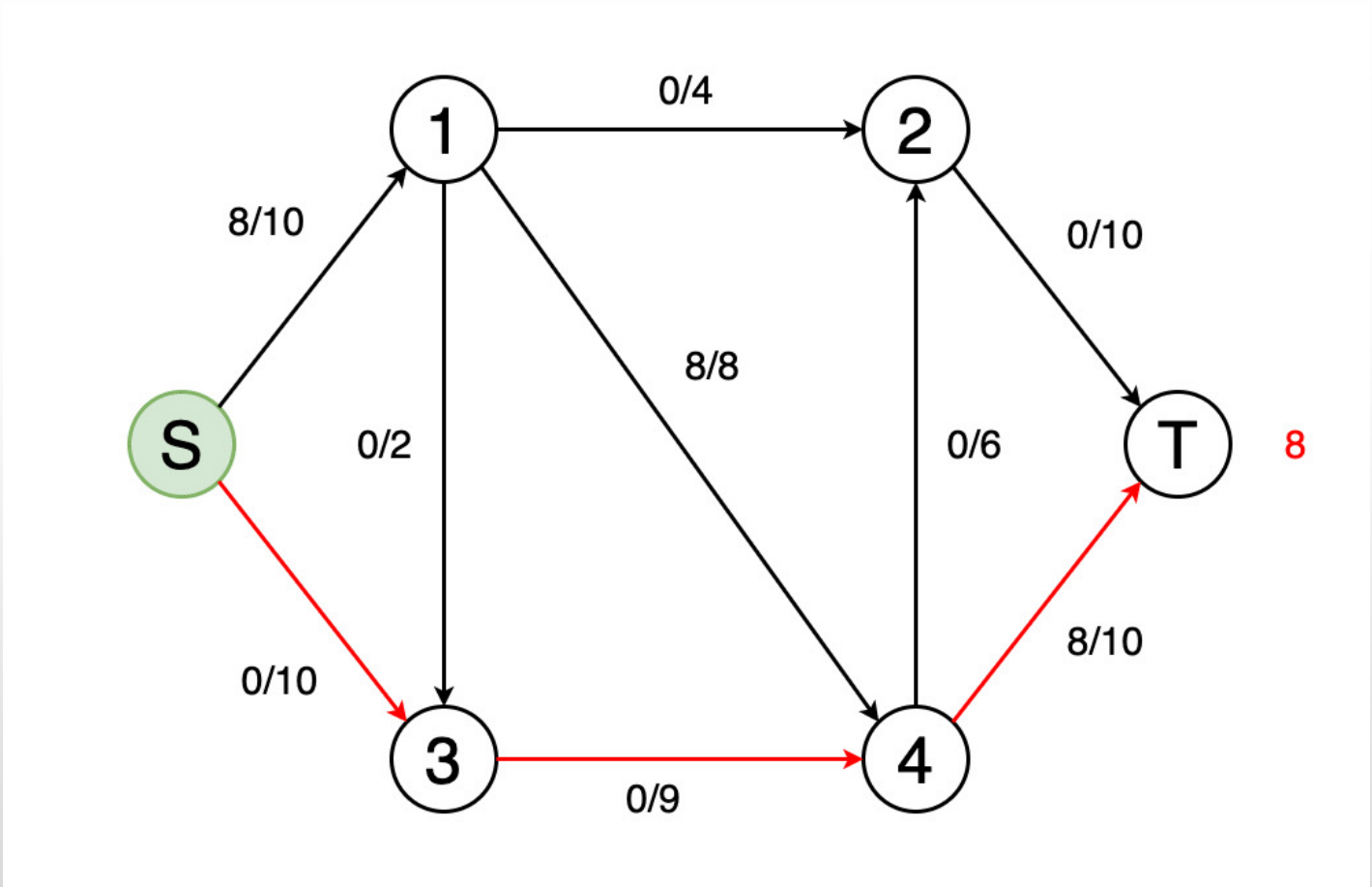
Ford Fulkerson



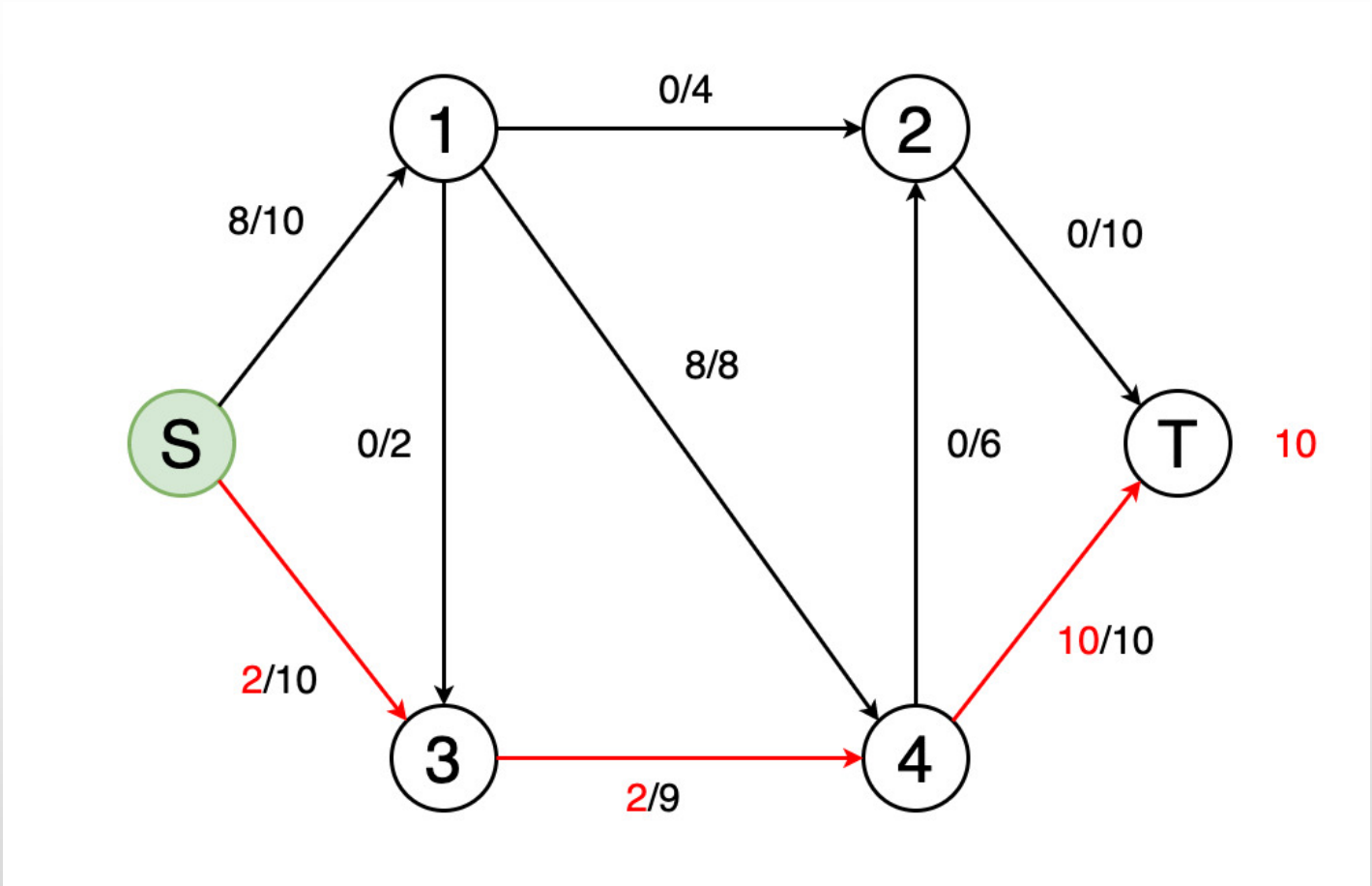
Ford Fulkerson



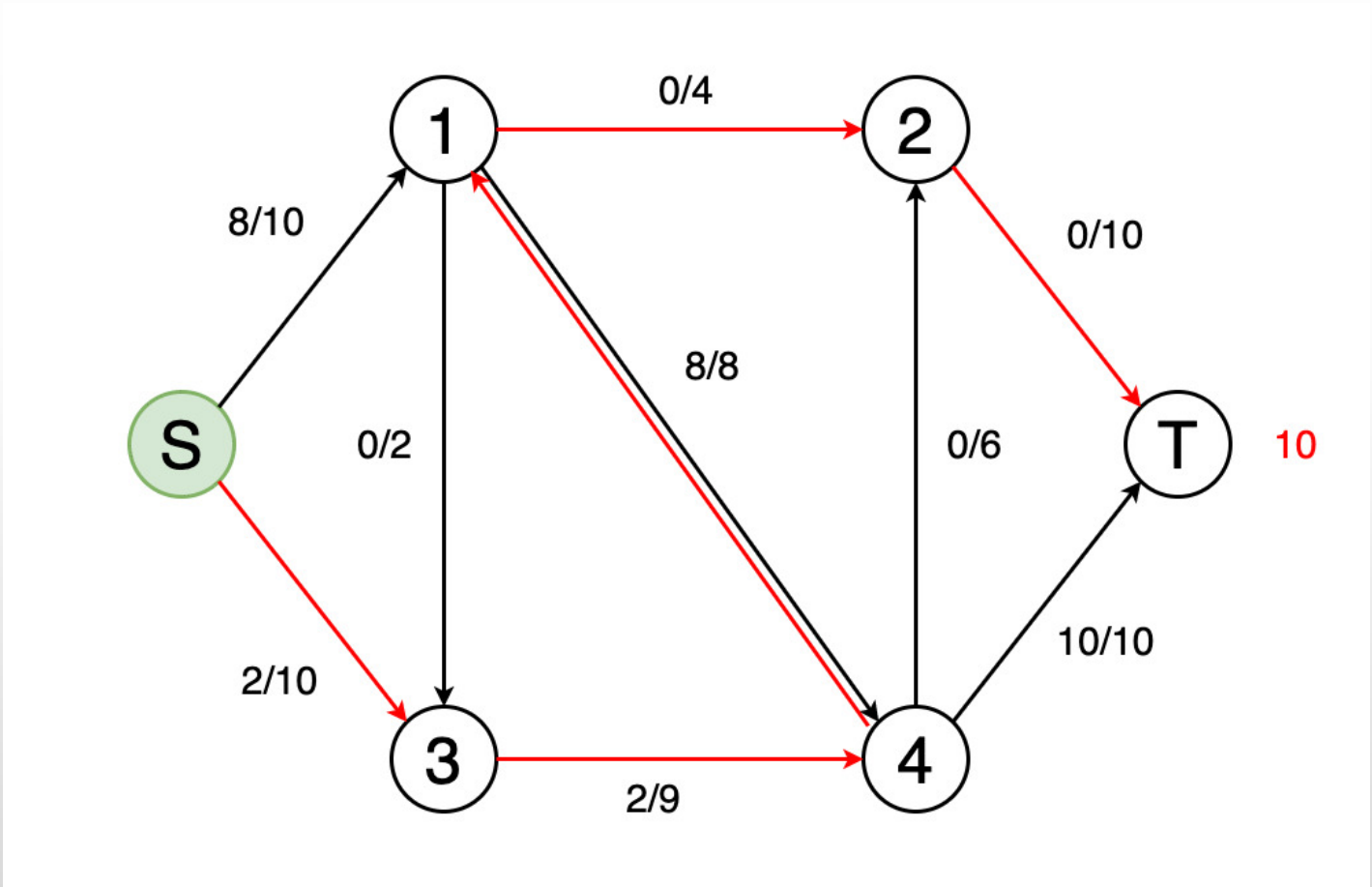
Ford Fulkerson



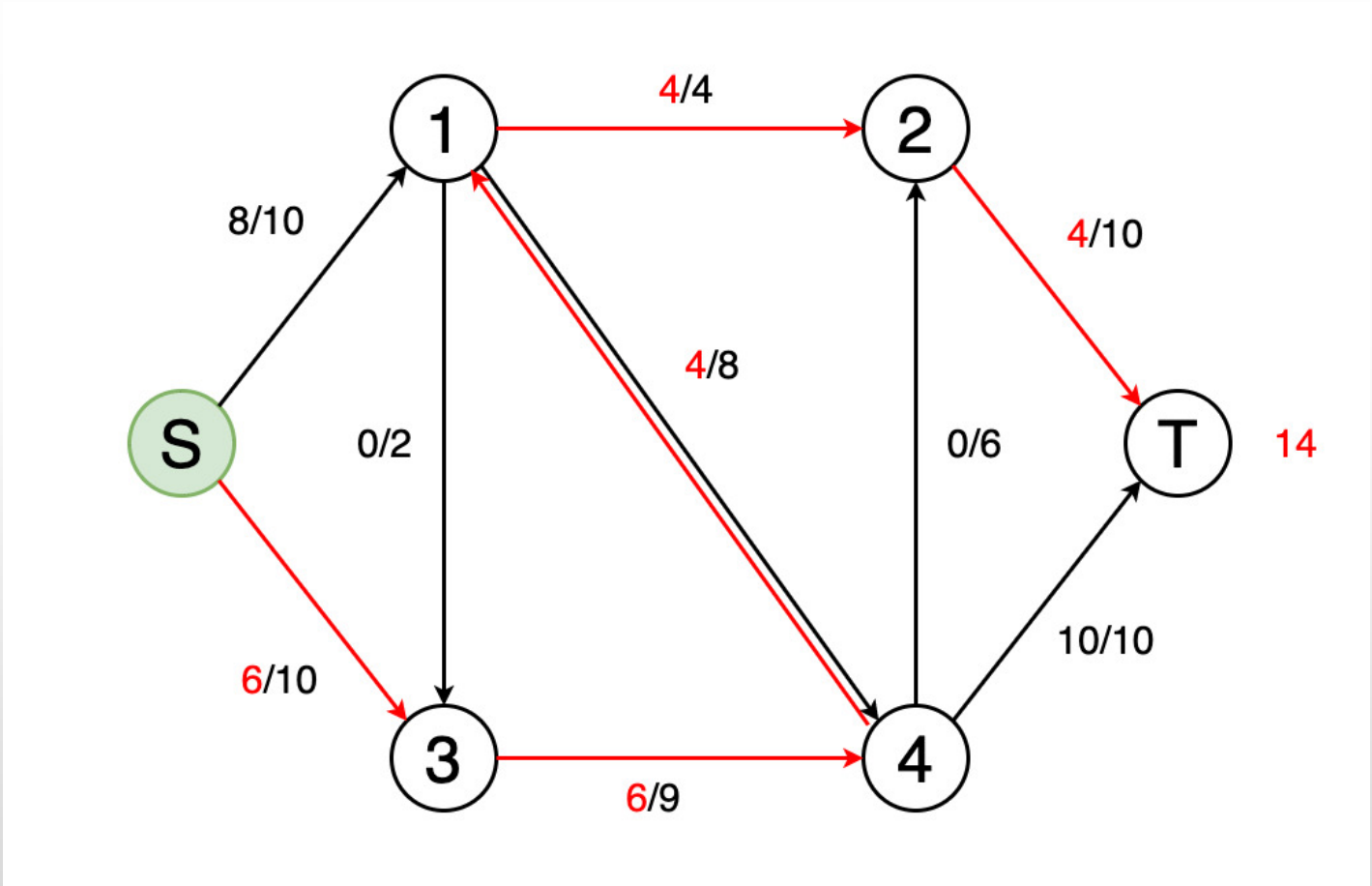
Ford Fulkerson



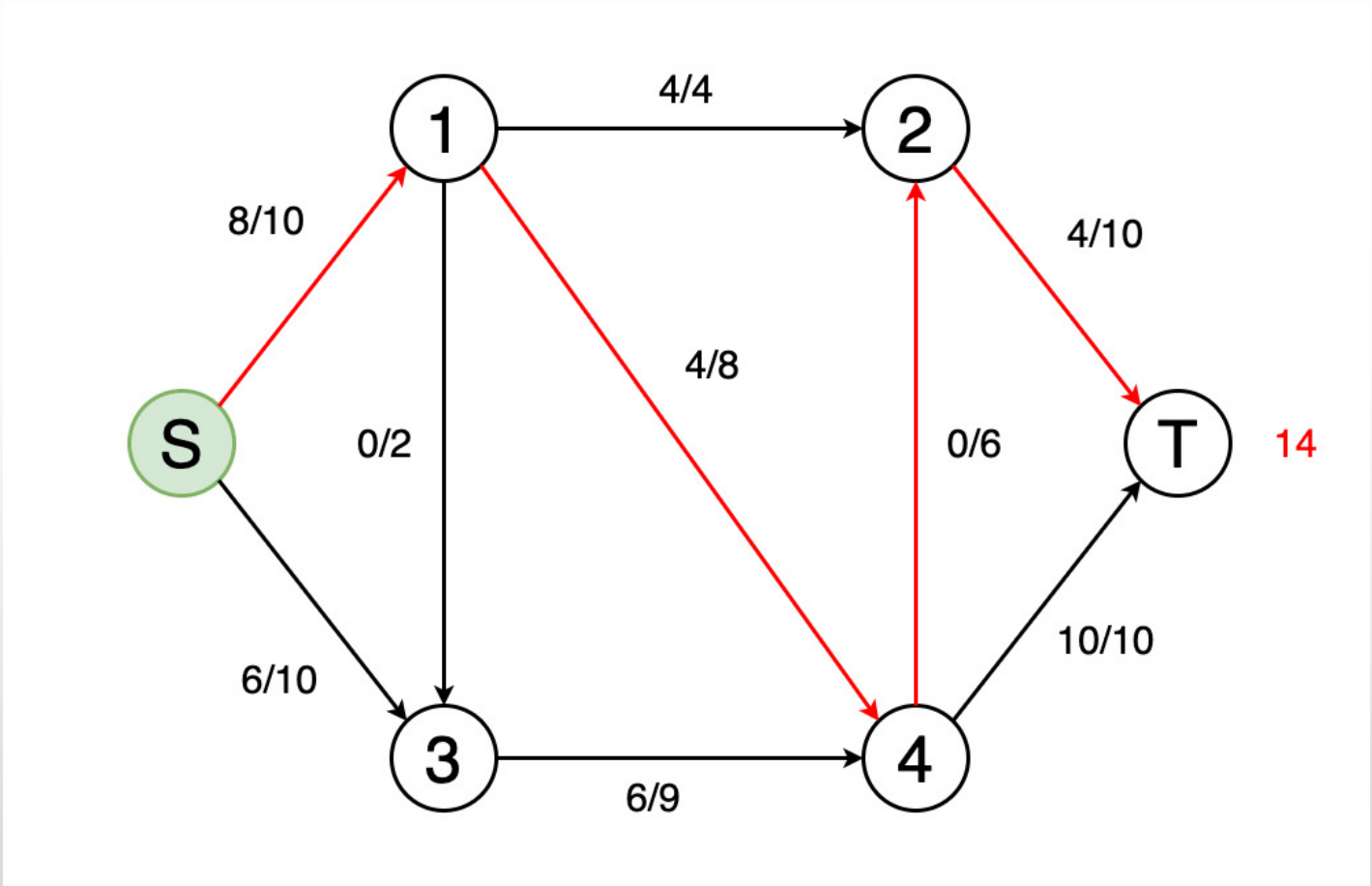
Ford Fulkerson



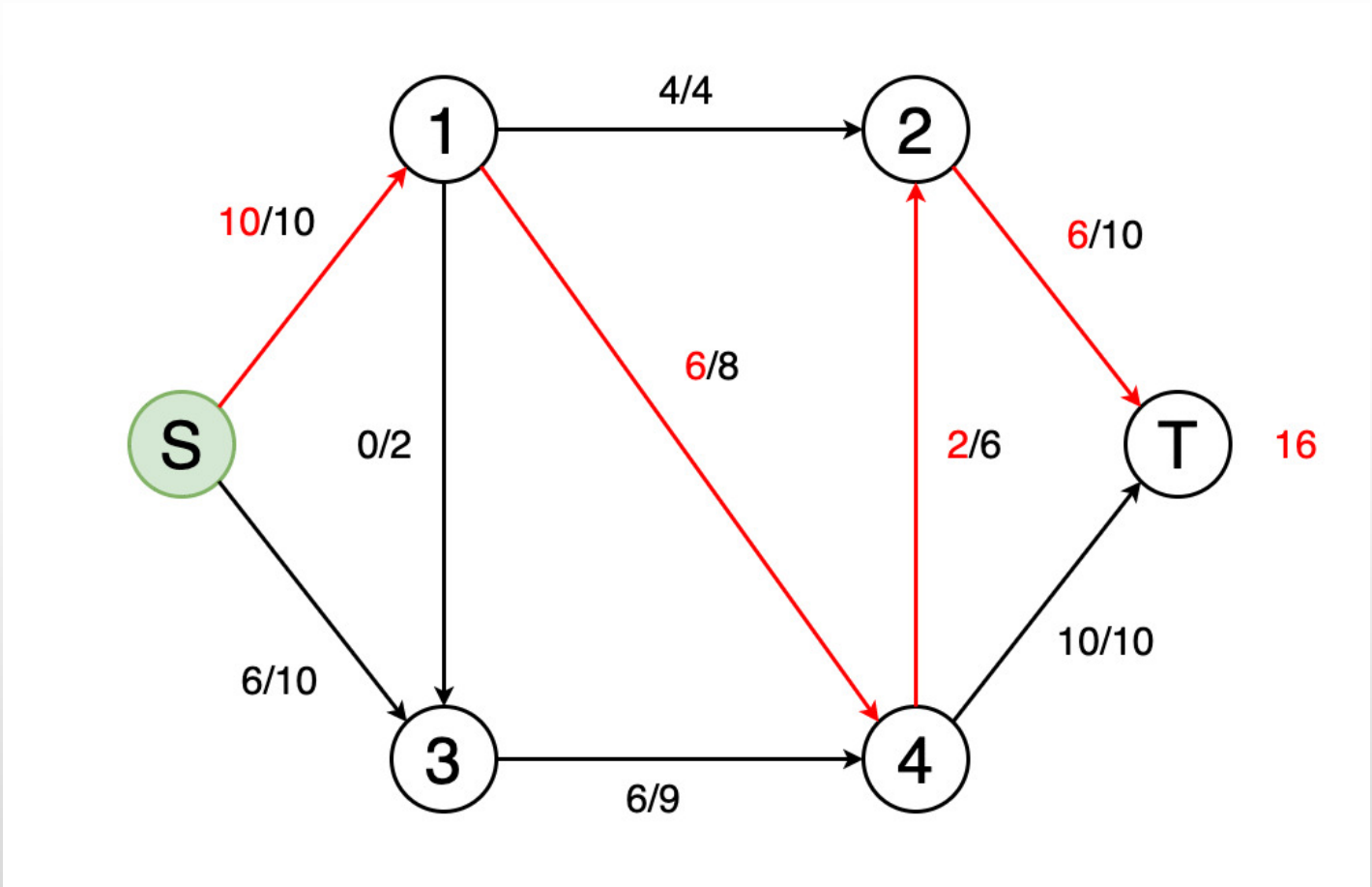
Ford Fulkerson



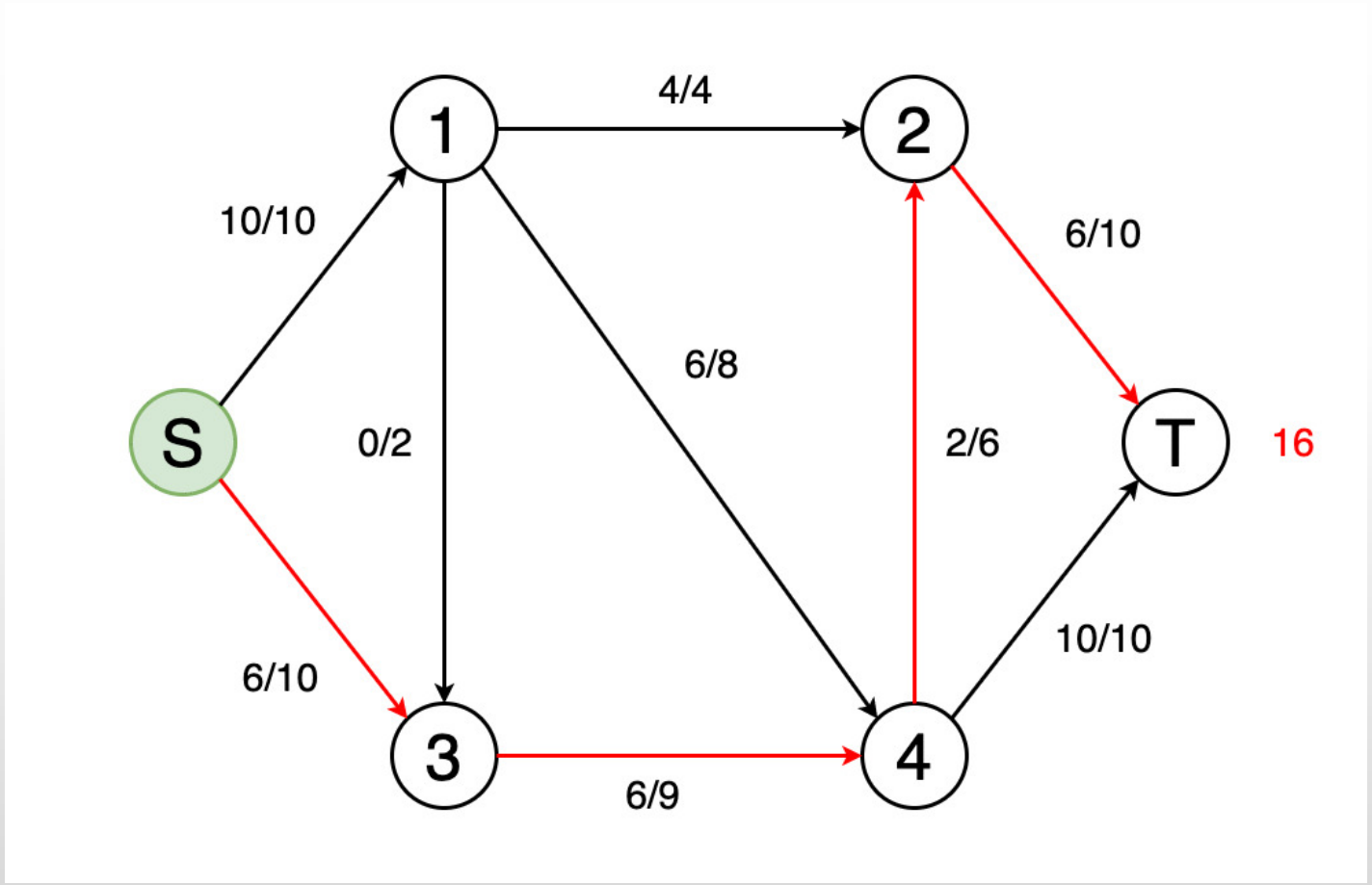
Ford Fulkerson



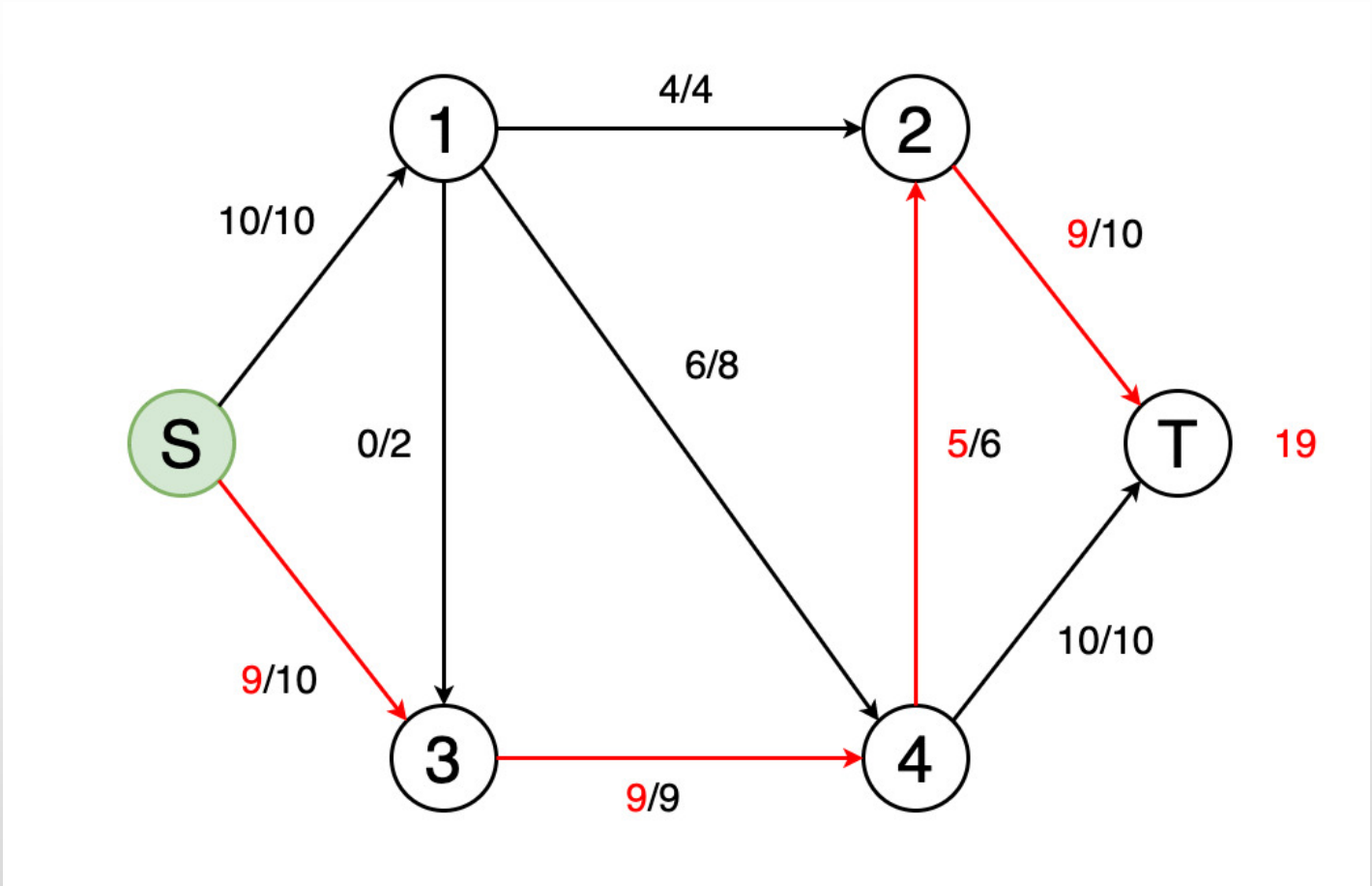
Ford Fulkerson



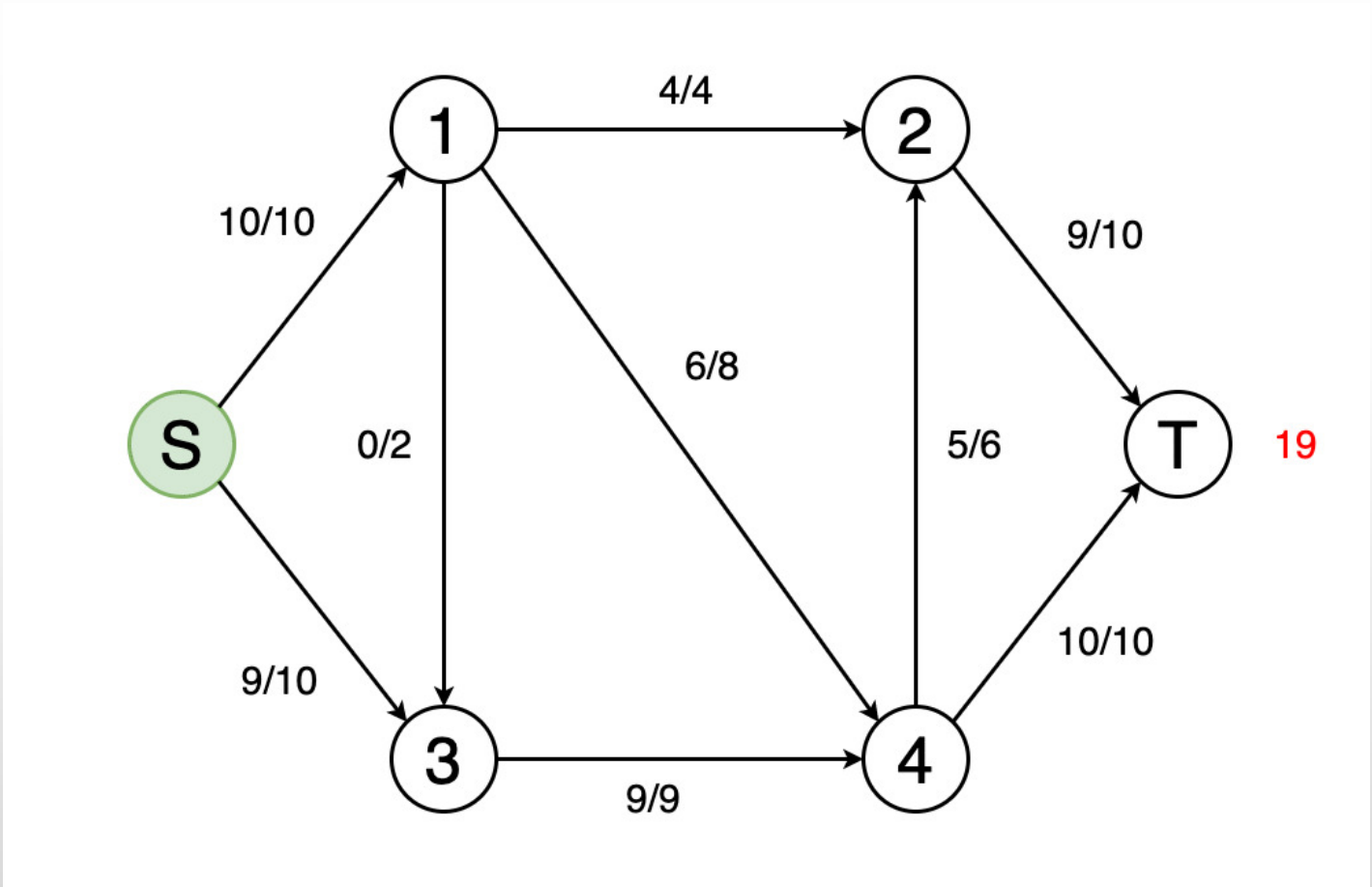
Ford Fulkerson



Ford Fulkerson



Ford Fulkerson



Algoritma Adımları

- Artırıcı yolları (kalan kapasite > 0) bulur.
- Her yolda minimum kalan kapasite kadar akış gönderir.
- Ters kenarlarla akışı günceller.
- Artırıcı yol kalmayana kadar devam eder.

Sözde Kod

FORD_FULKERSON(G, kaynak, hedef):

```
akış = [ ]  // Her kenar için akış (başlangıçta 0)
maksimumAkış = 0

her bir (u, v) için E içinde:
        akış[u, v] = 0
```

Sözde Kod (2)

```
döngü artırıcı yol var iken:
    yol = DFS(graf, kaynak, hedef, akış)
    // Yoldaki minimum kalan kapasiteyi bul
    delta = ∞
    v = hedef
    döngü v ≠ kaynak iken:
        u = yol.ata[v]
        delta = min(delta, kapasite[u, v] - akış[u, v])
        v = u
```

Sözde Kod (3)

```
        // Yoldaki akışları güncelle
        v = hedef
        döngü v ≠ kaynak iken:
                u = yol.ata[v]
                akış[u, v] += delta
                akış[v, u] -= delta  // Ters kenar
                v = u
        maksimumAkış += delta
döndür maksimumAkış, akış
```

Edmonds Karp

- Ağırlıklı yönlü çizgede iki düğüm arasındaki maksimum akışı bulur.
- Ford-Fulkerson Algoritması'nın bir türevidir.
- *Jack Edmonds* ve *Richard Karp* tarafından geliştirilmiştir.
- BFS (Breadth-First Search) kullanarak artan yolları bulur ve bu yollarda maksimum akışı uygular.
- *Edmond's Karp uses BFS to find an augmenting path and Dinic's uses BFS to check if more flow is possible and to construct level graph.*

Algoritma Adımları

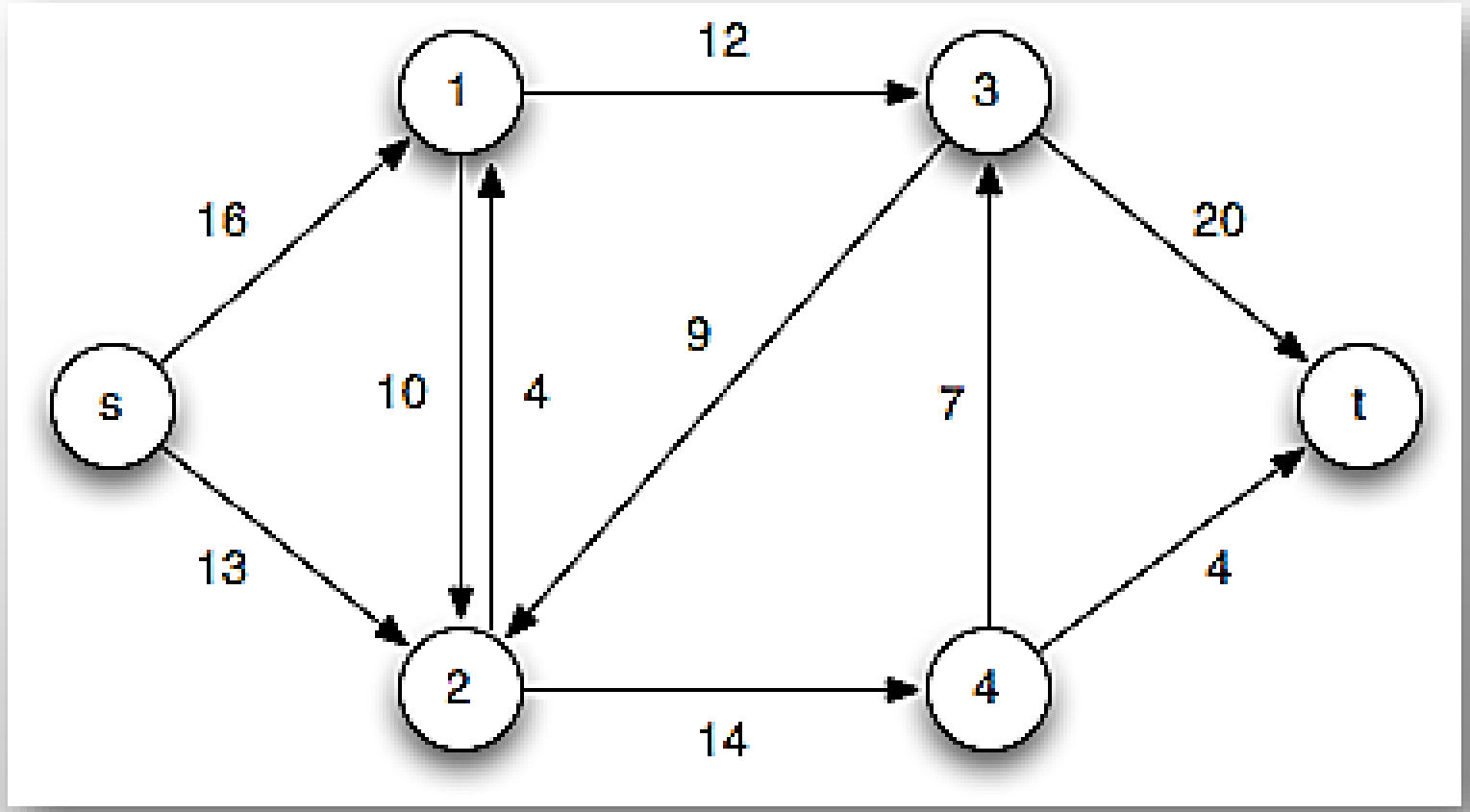
- Adım 1: BFS kullanılarak, kaynaktan hedefe olan artan yollar bulunur.
- Adım 2: Bulunan artan yollar boyunca maksimum artışa izin verilir. Bu, akış ağındaki tüm kenarlarda artışa neden olur.
- Adım 3: Hedefe ulaşılana kadar Adım 1 ve Adım 2 tekrarlanır.

Karmaşıklık Analizi

- Algoritma karmaşıklığı,
  - BFS kullanılarak artan yolların bulunmasına dayanır ve
  - $O(V E^2)$ karmaşıklığına sahiptir.
    - E kenar sayısı
    - V düğüm sayısı.
  - Her aşamada kaynaktan hedefe artış yollarını bulmak için BFS kullanır.
  - Artış yolundaki tüm kenarların kapasitelerini artırmak için tekrar BFS.
  - En kötü durumda, bu işlemler V kez gerçekleşir.

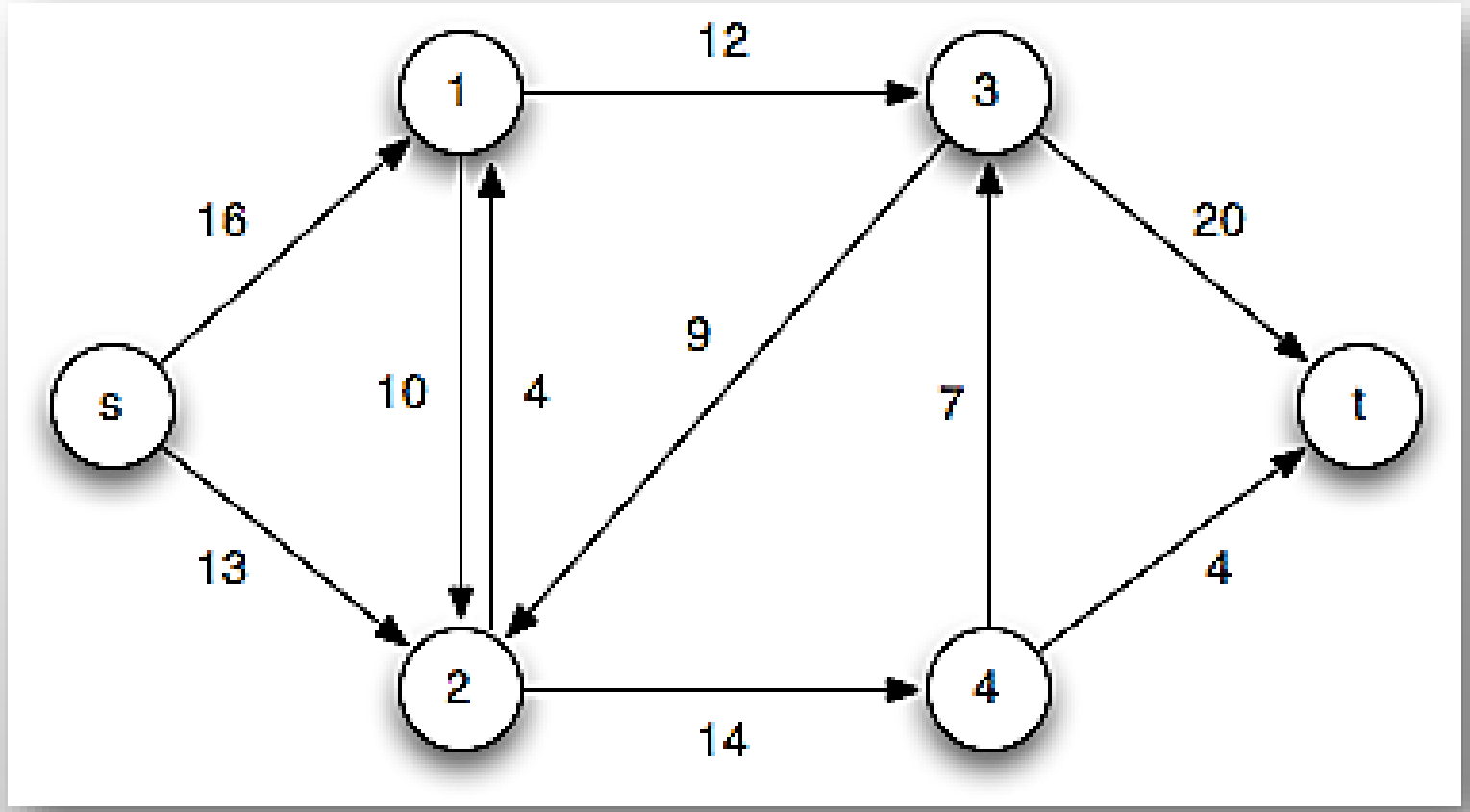
Örnek

- Aşağıdaki çizge verilmiş olsun



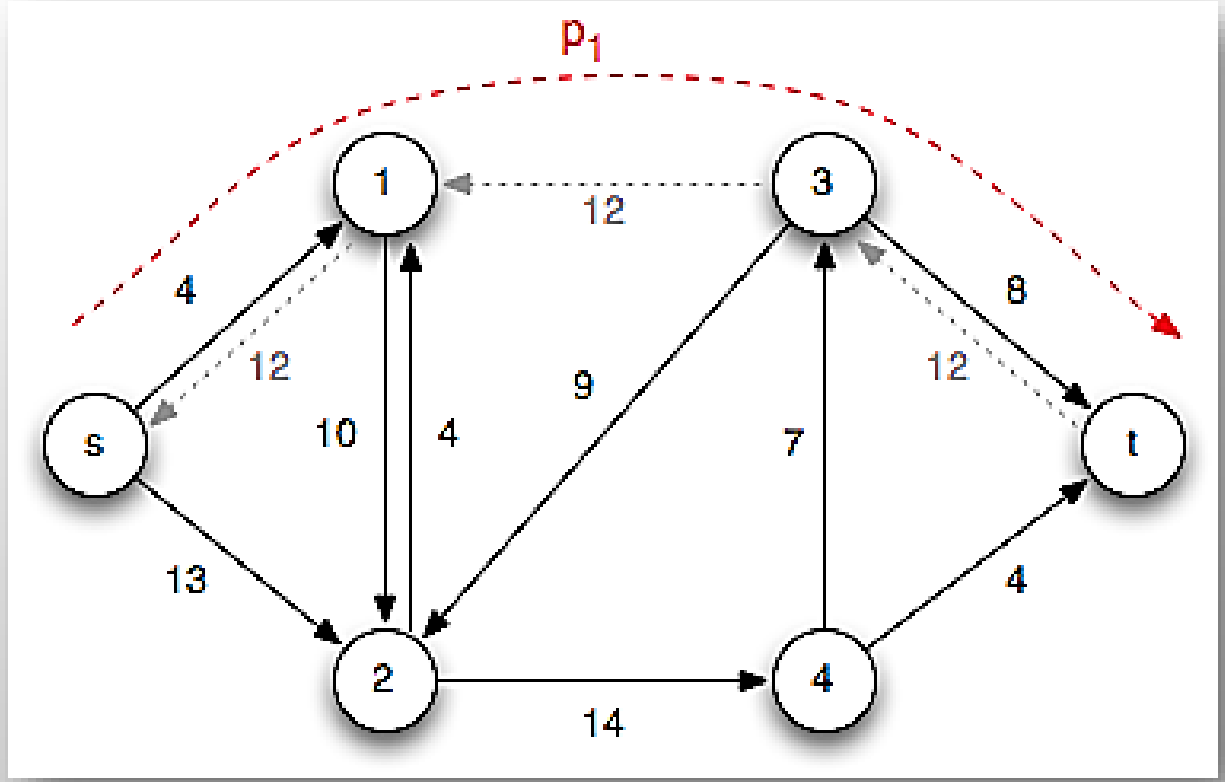
Örnek

- |f *| ≤ 24 (s'den çıkan, ya da t'ye giren kenarların ağırlıkları toplamı)



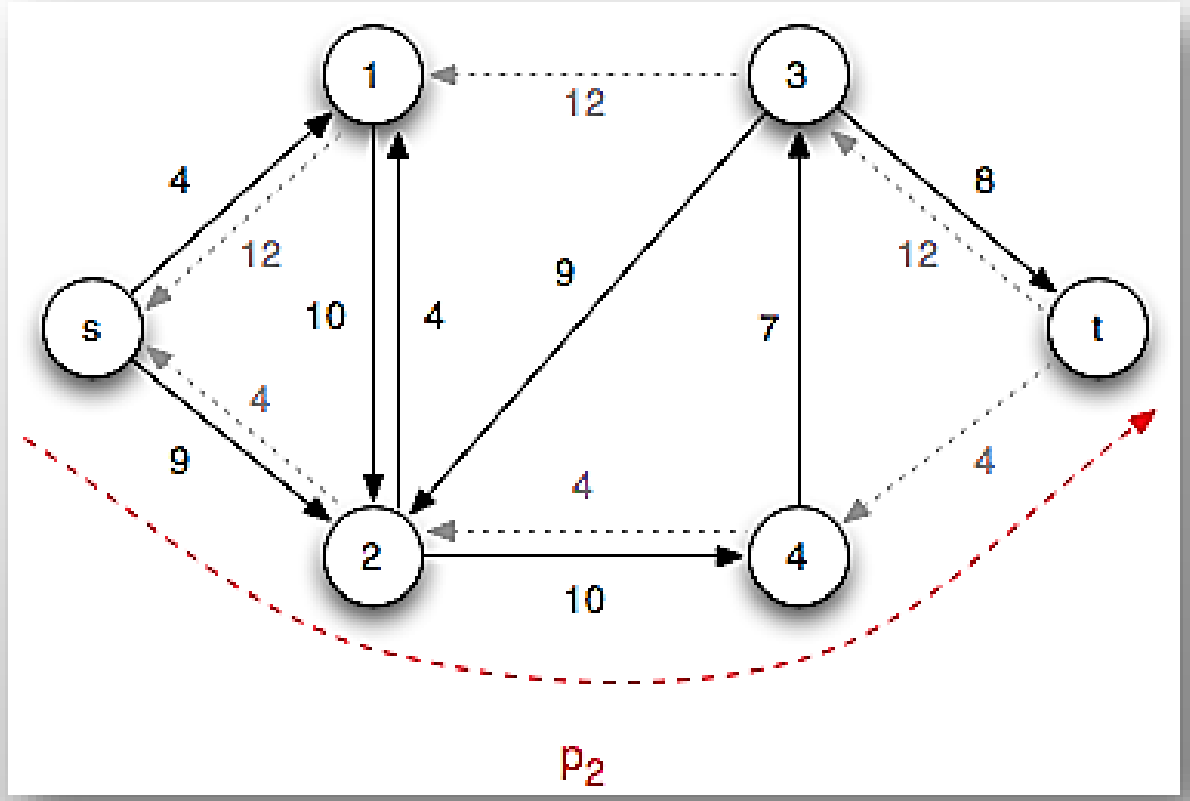
Adım 1

- c(1,3) (*residual network*) yüzünden cf(p1) = 12'ye sahip p1 = <s,1,3,t> yolu artan yol (*augmenting path*) olarak seçilir.



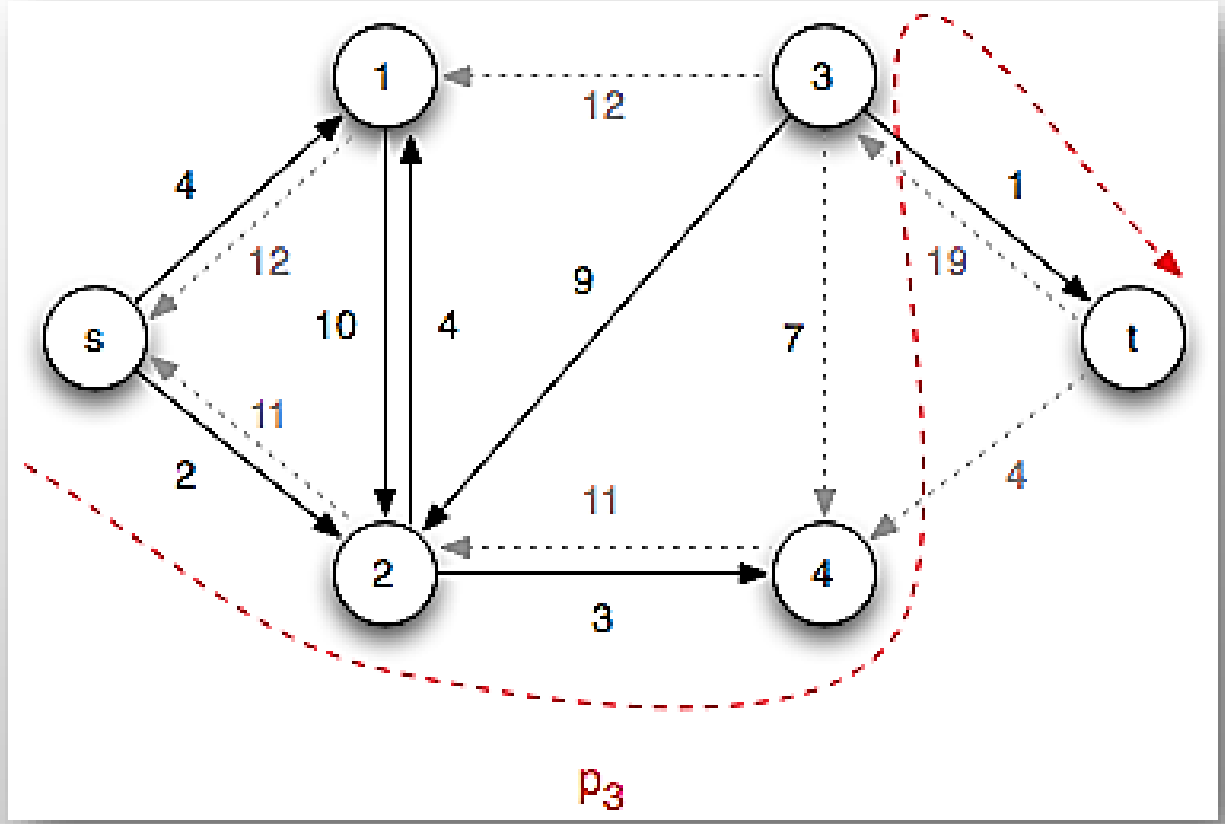
Adım 2

- c(4,t) yüzünden cf(p2) = 4'e sahip p2 = <s,2,4,t> yolu artan yol olarak seçilir.



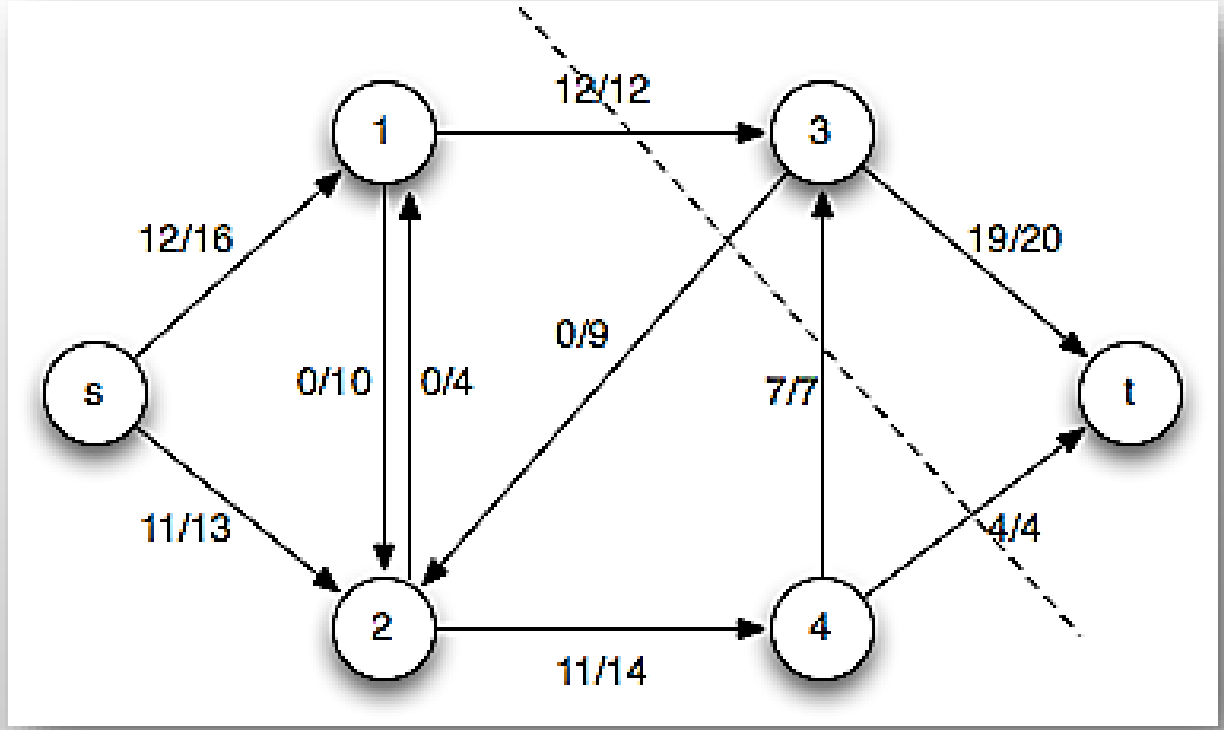
Adım 3

- c(4,3) yüzünden cf(p3) = 7'ye sahip p3 = <s,2,4,3,t> yolu artan yol olarak seçilir.



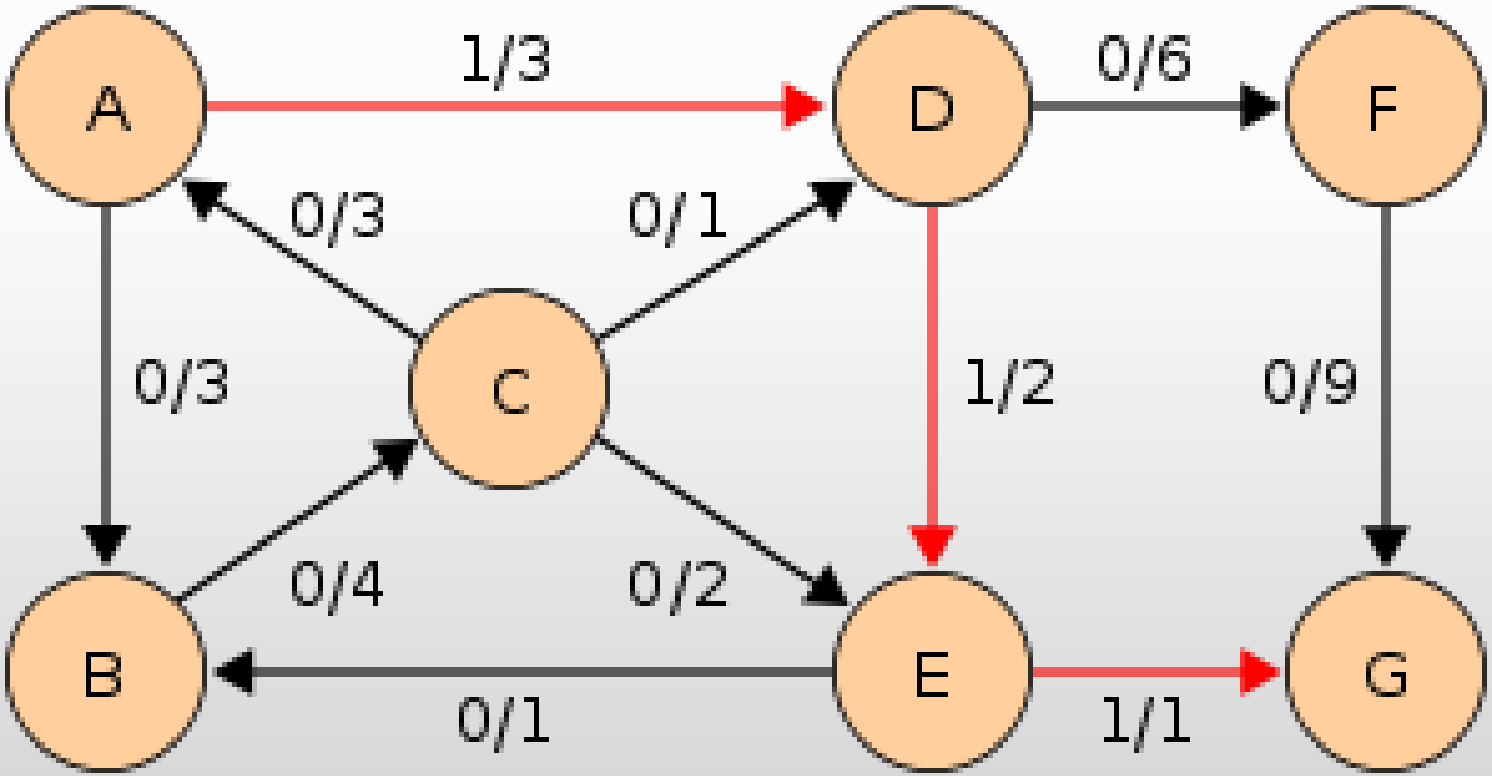
Son Durum

- Sadece 3 numaralı düğüm ek kapasiteye sahip. *final flow network with a min-cut*. |f *| = 19 + 4 = 12 + 11 = 23.



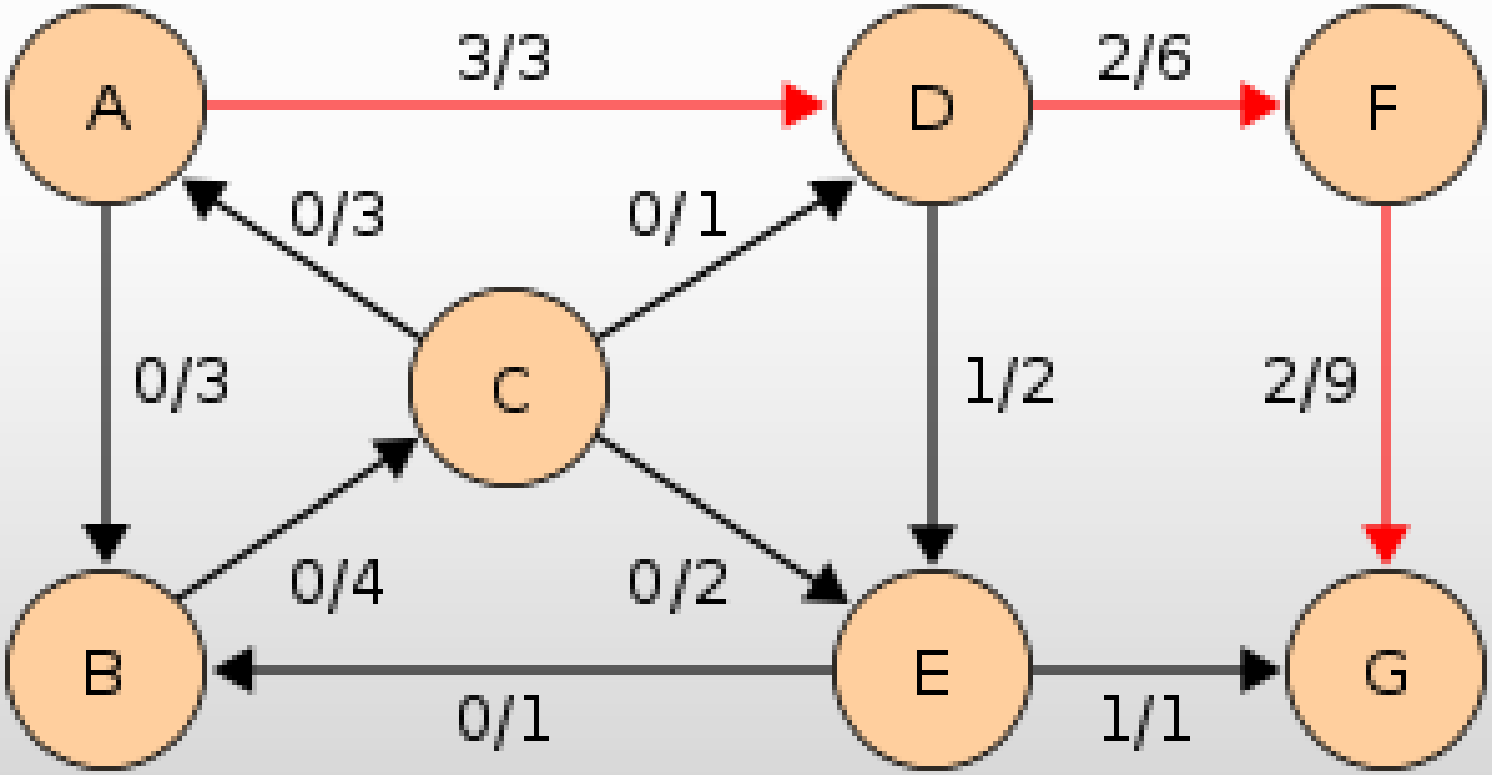
Edmonds–Karp

- p = {A,D,E,G}, flow = 1



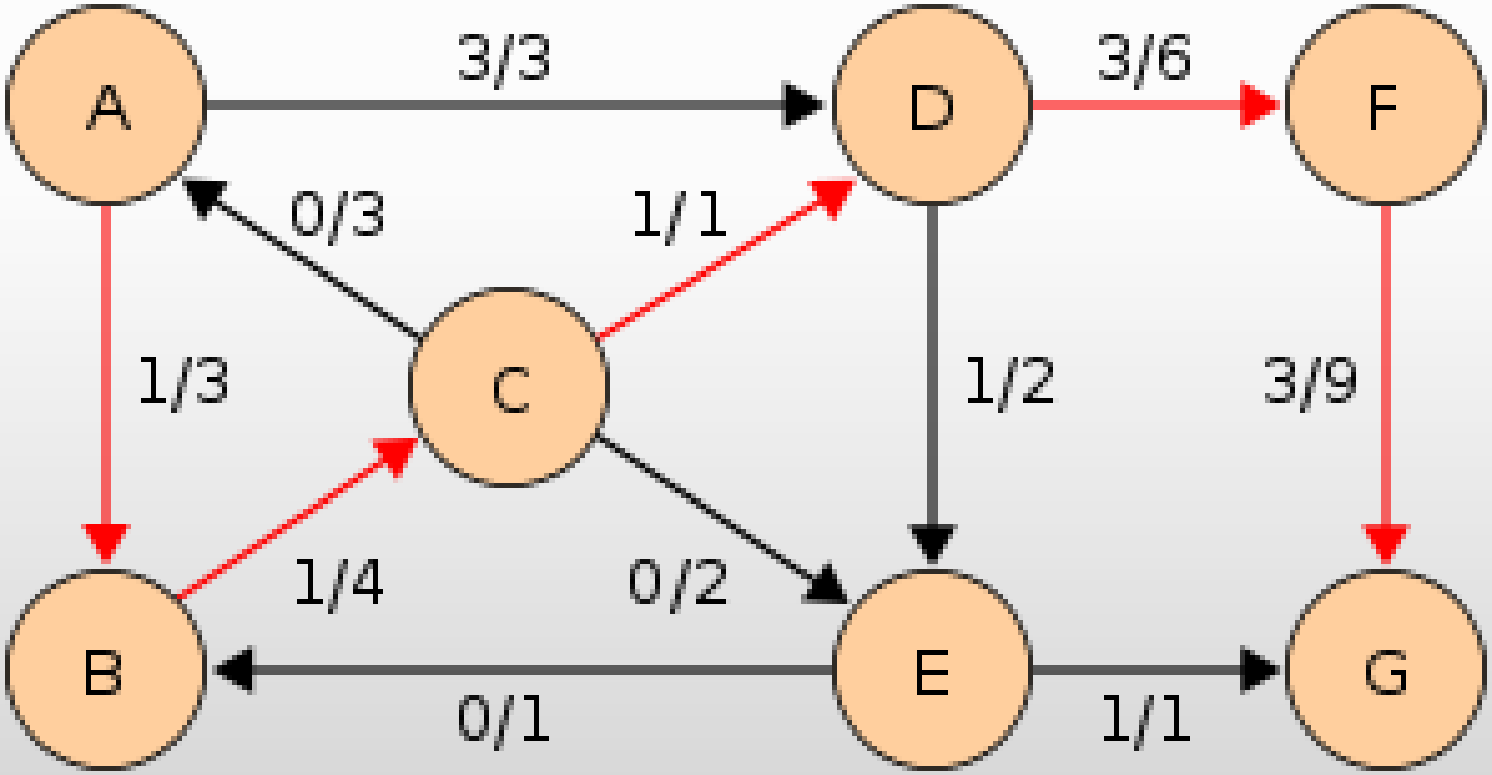
Edmonds–Karp

- p = {A,D,F,G}, flow = 2



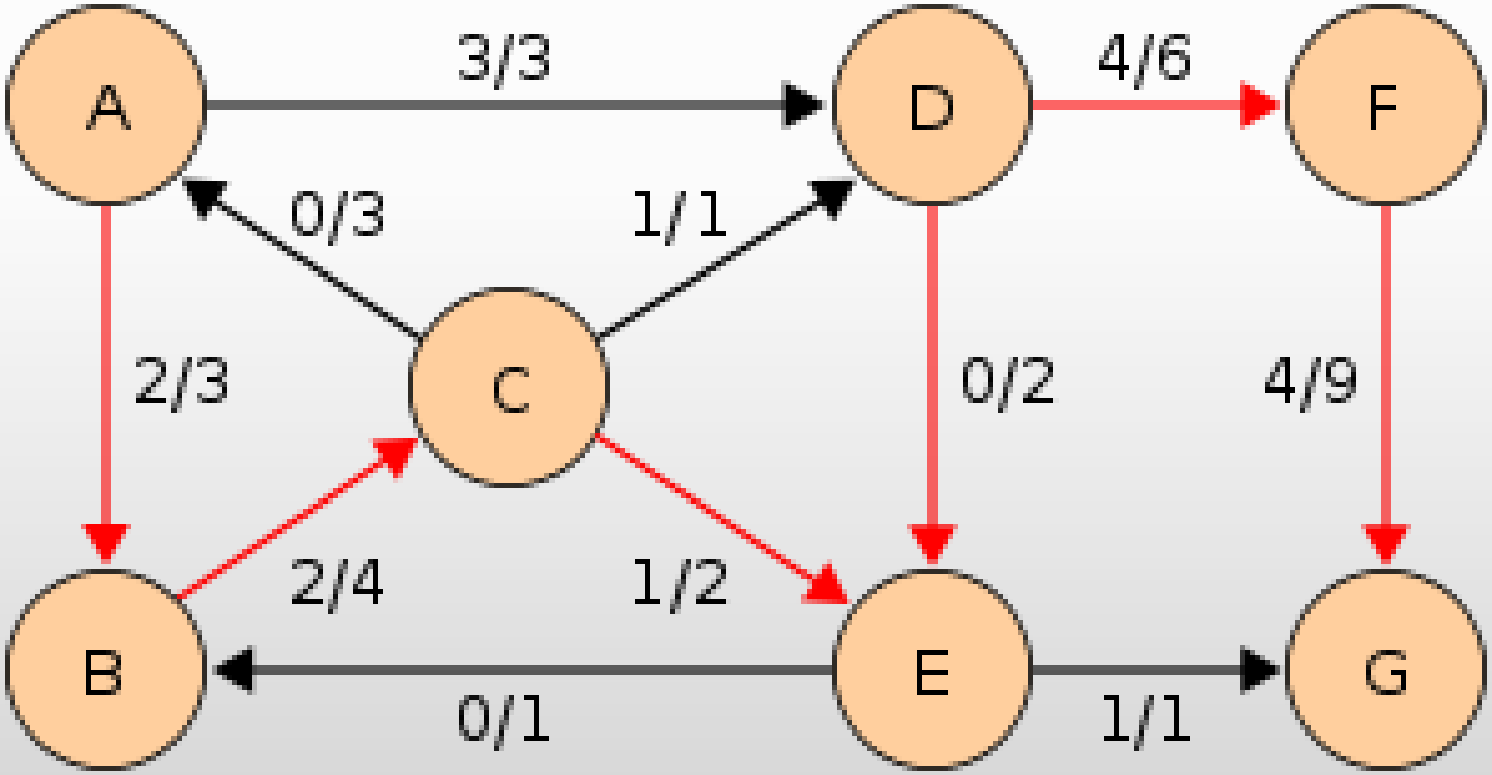
Edmonds–Karp

- p = {A,B,C,D,F,G}, flow = 1

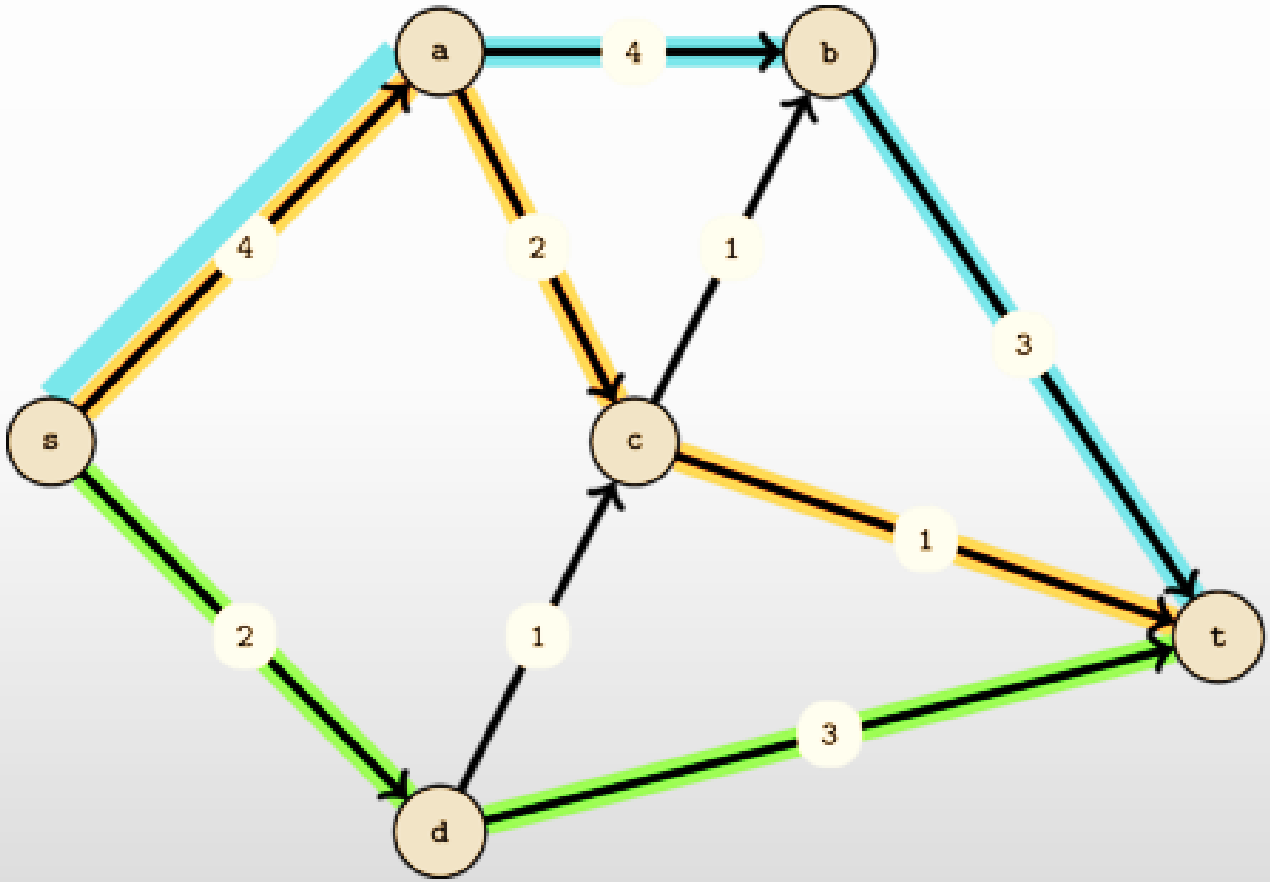


Edmonds–Karp

- p = {A,B,C,E,D,F,G}, flow = 1

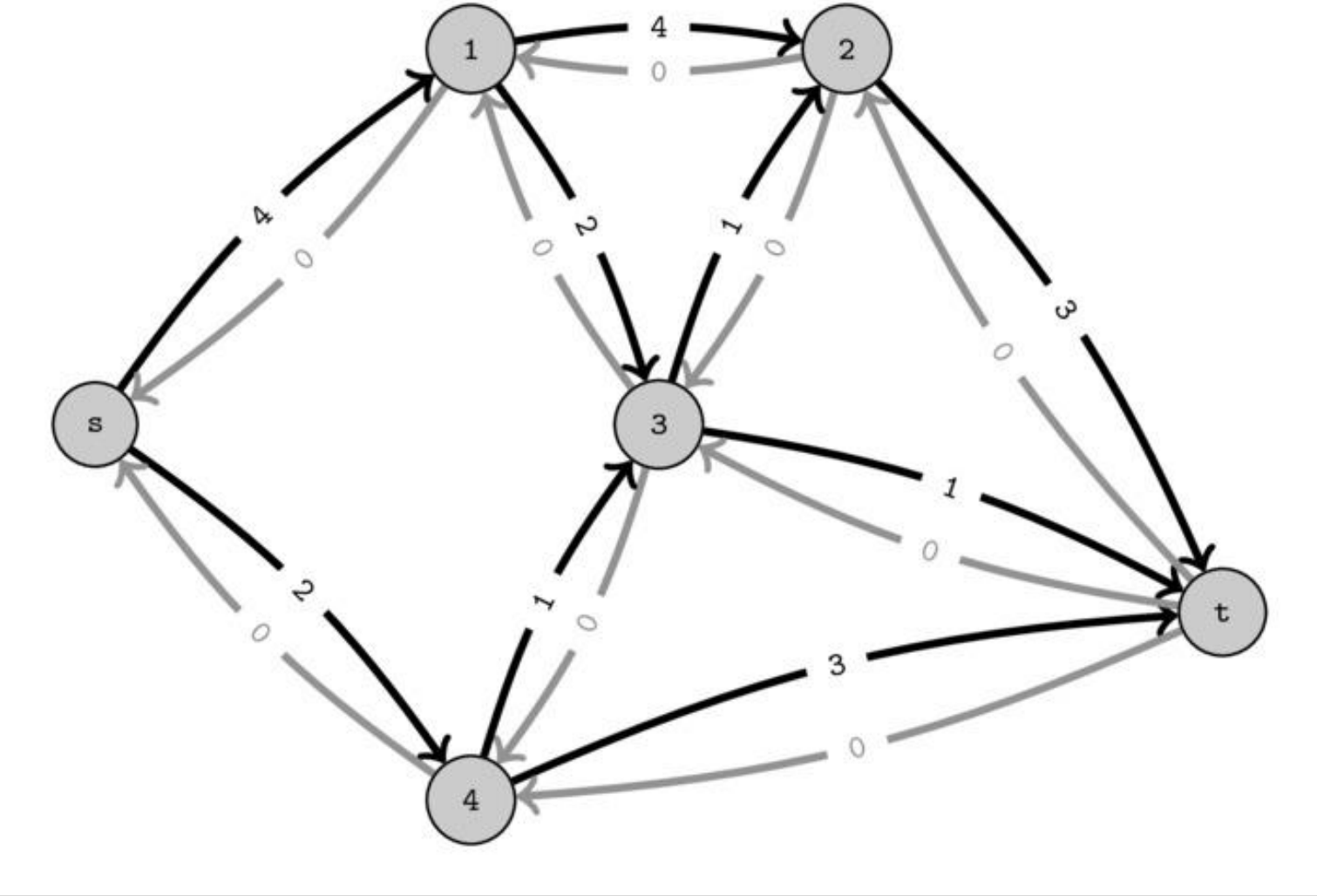


Edmonds Karp

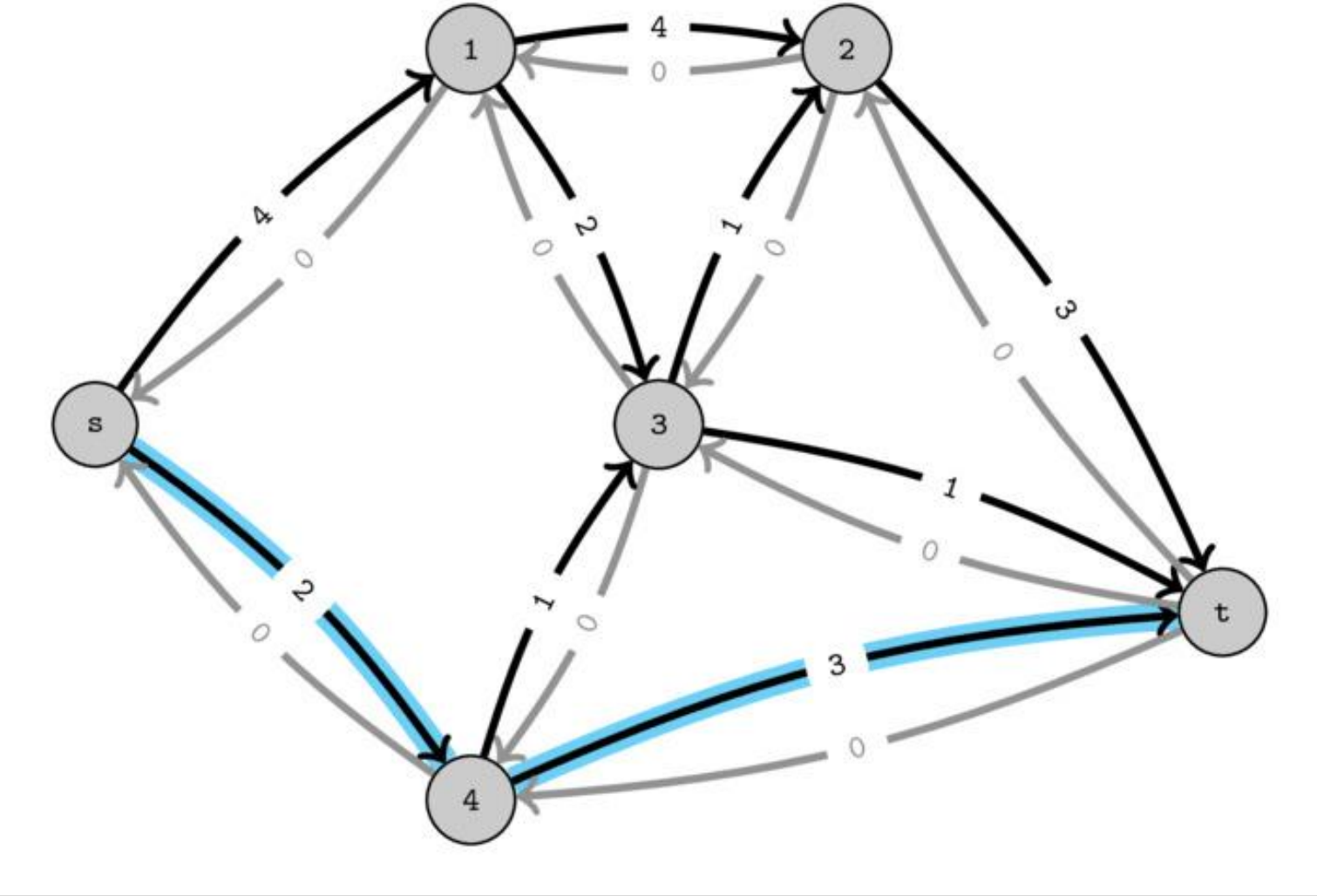


3 units of flow
1 unit of flow
2 units of flow

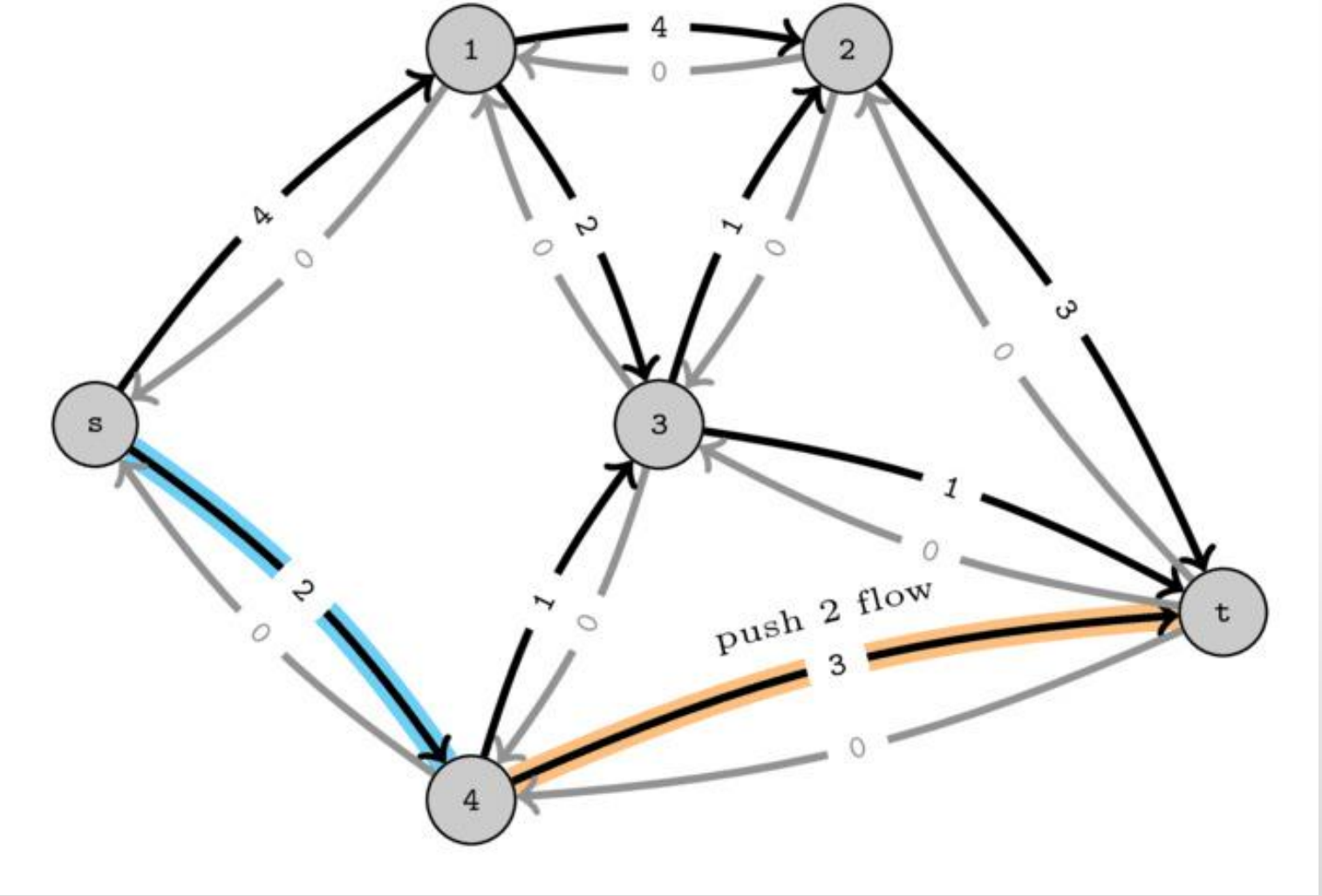
Edmonds Karp



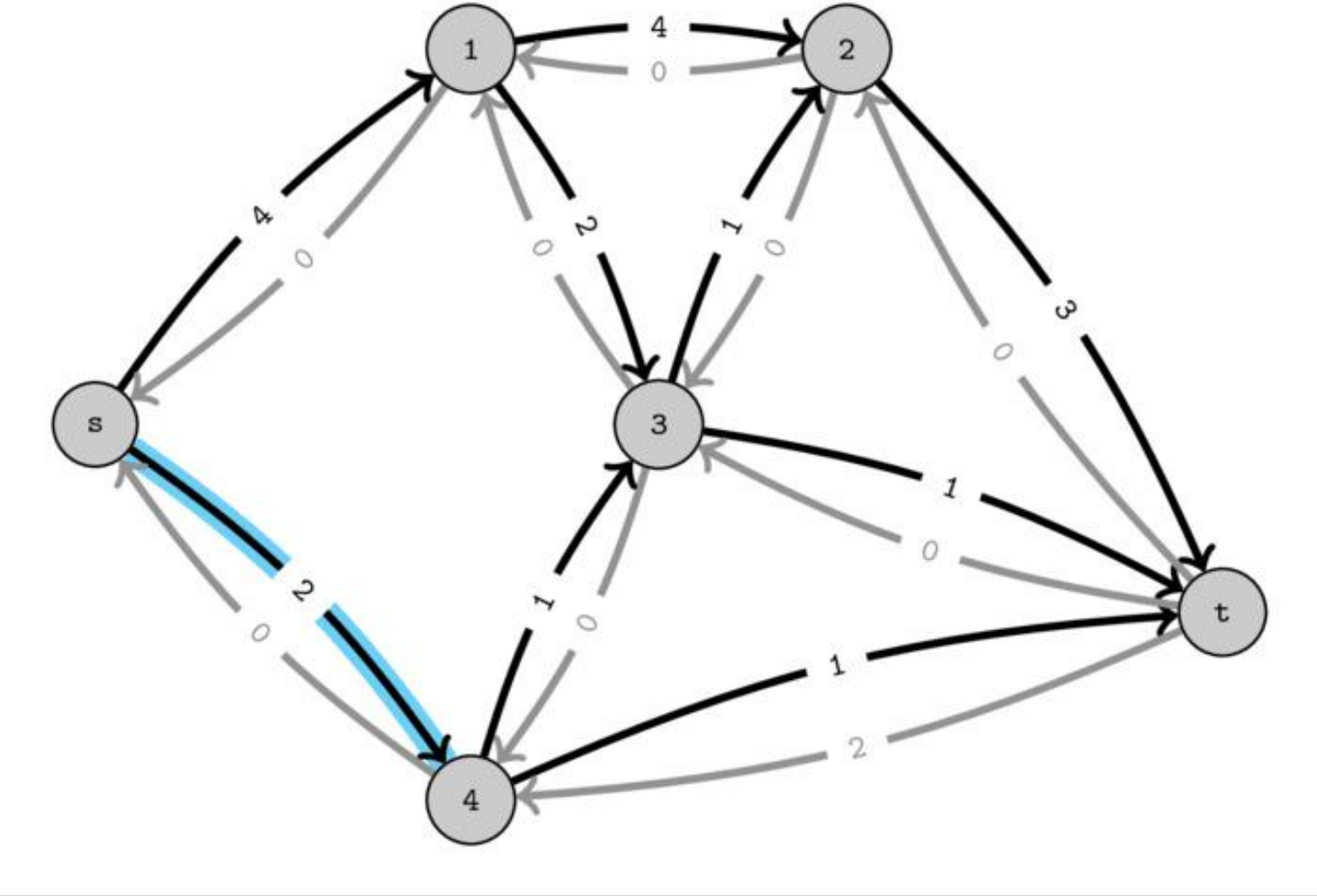
Edmonds Karp



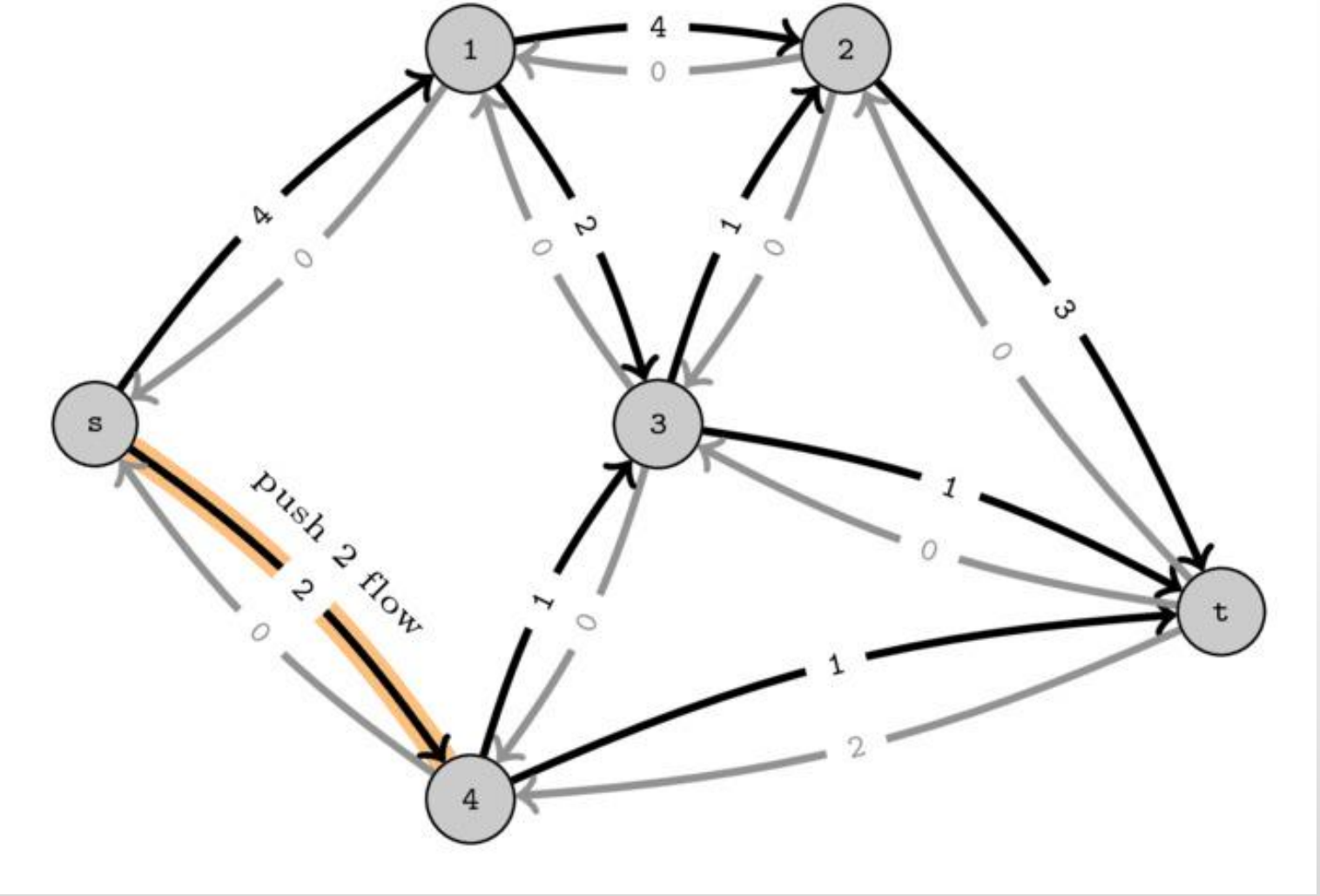
Edmonds Karp



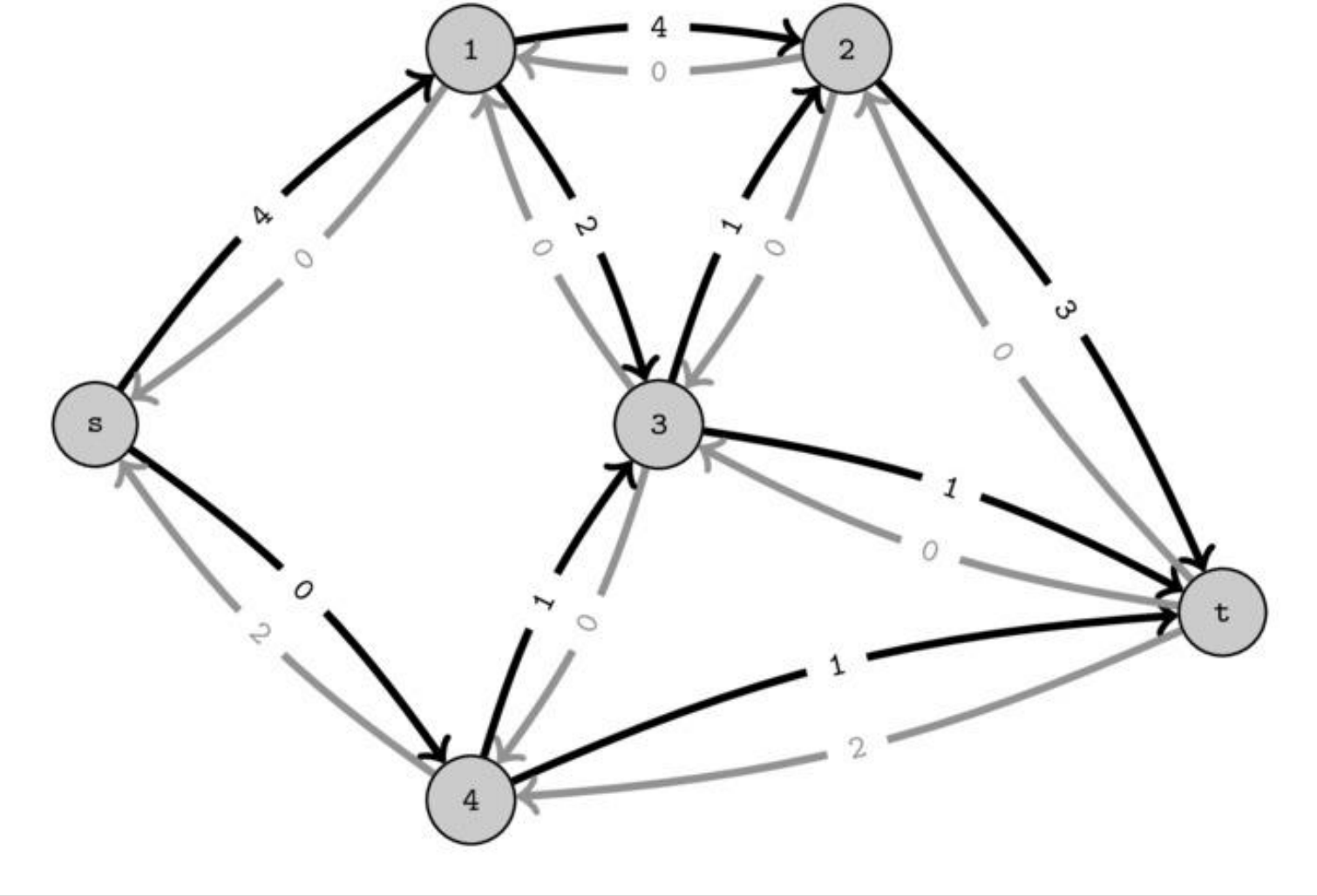
Edmonds Karp



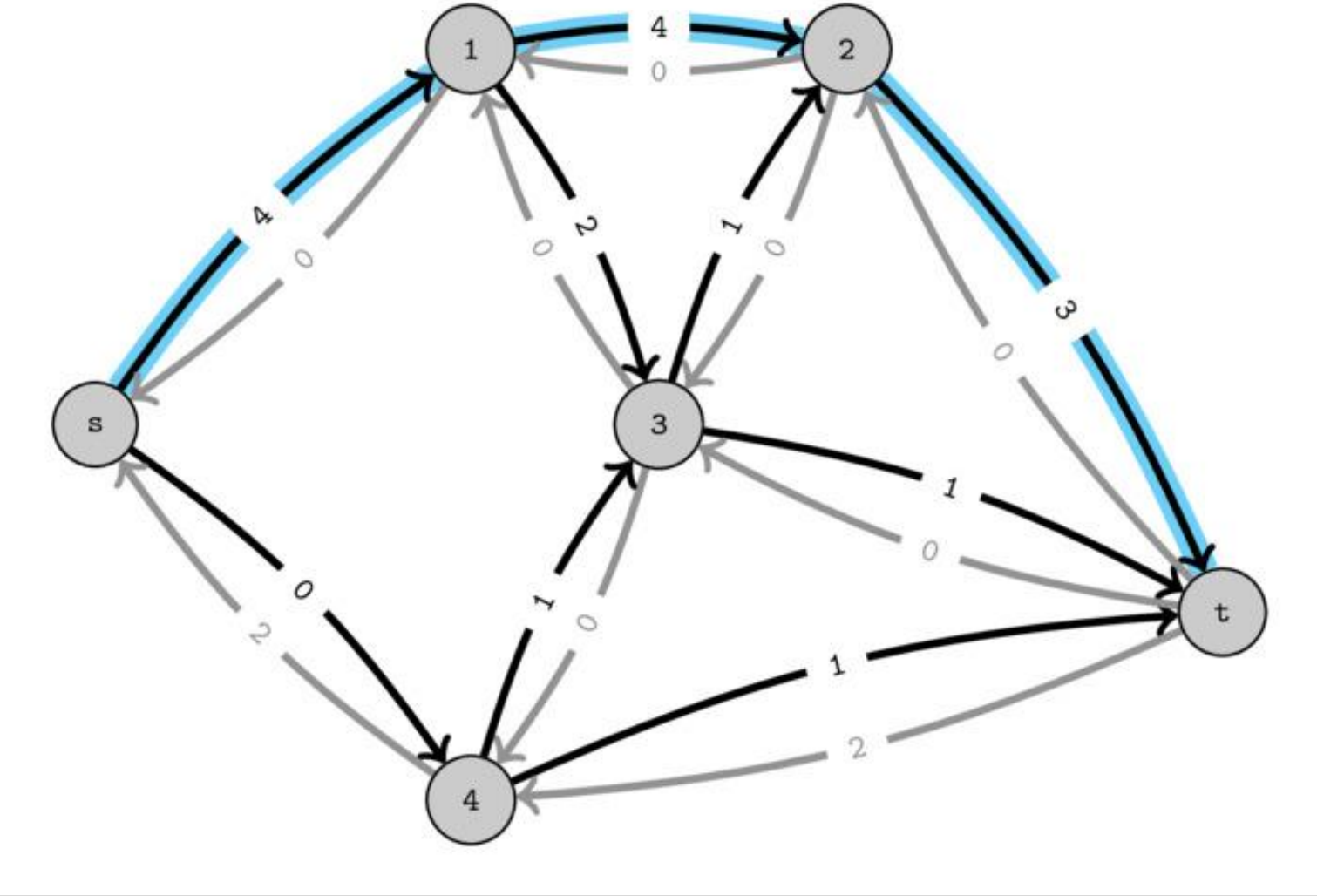
Edmonds Karp



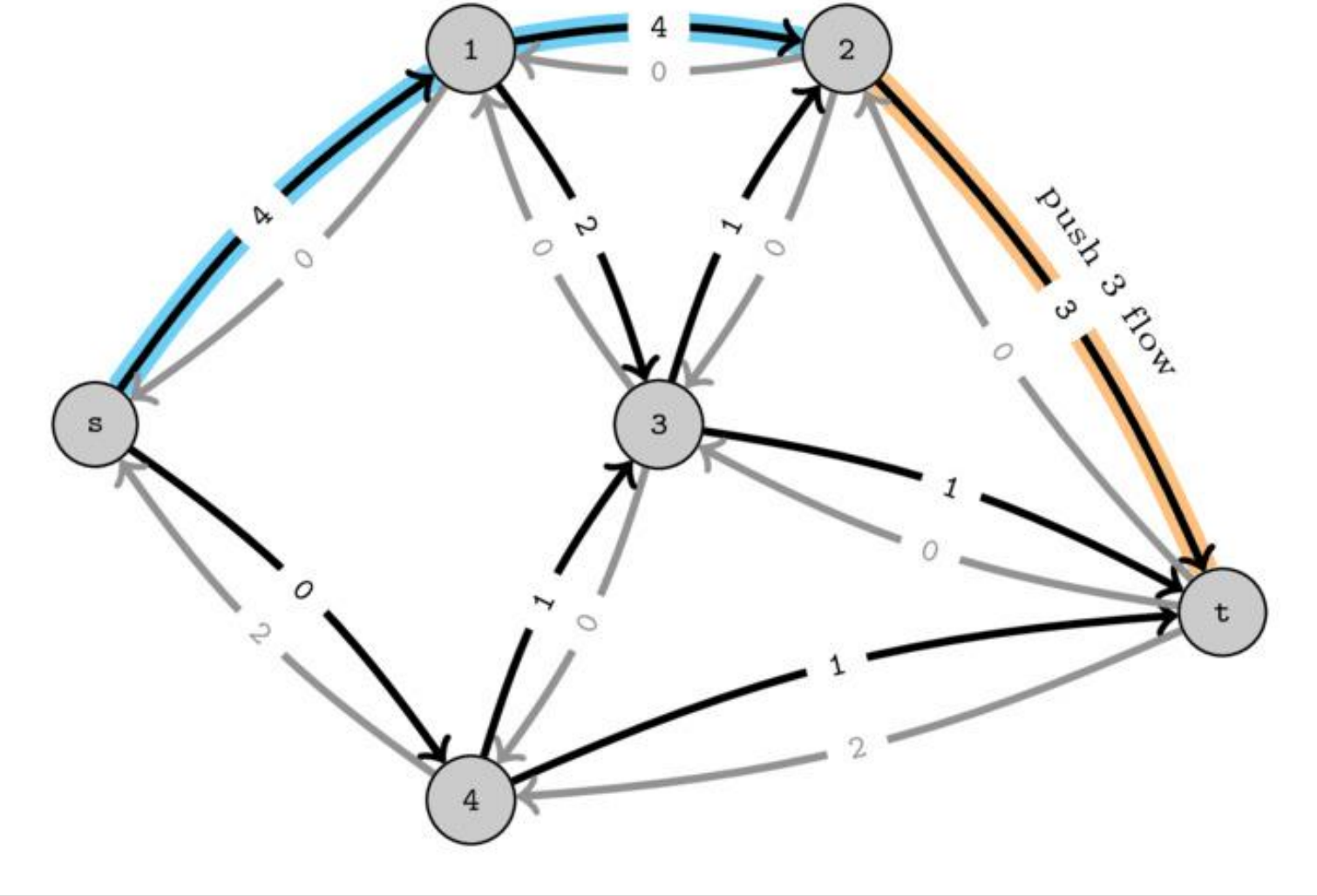
Edmonds Karp



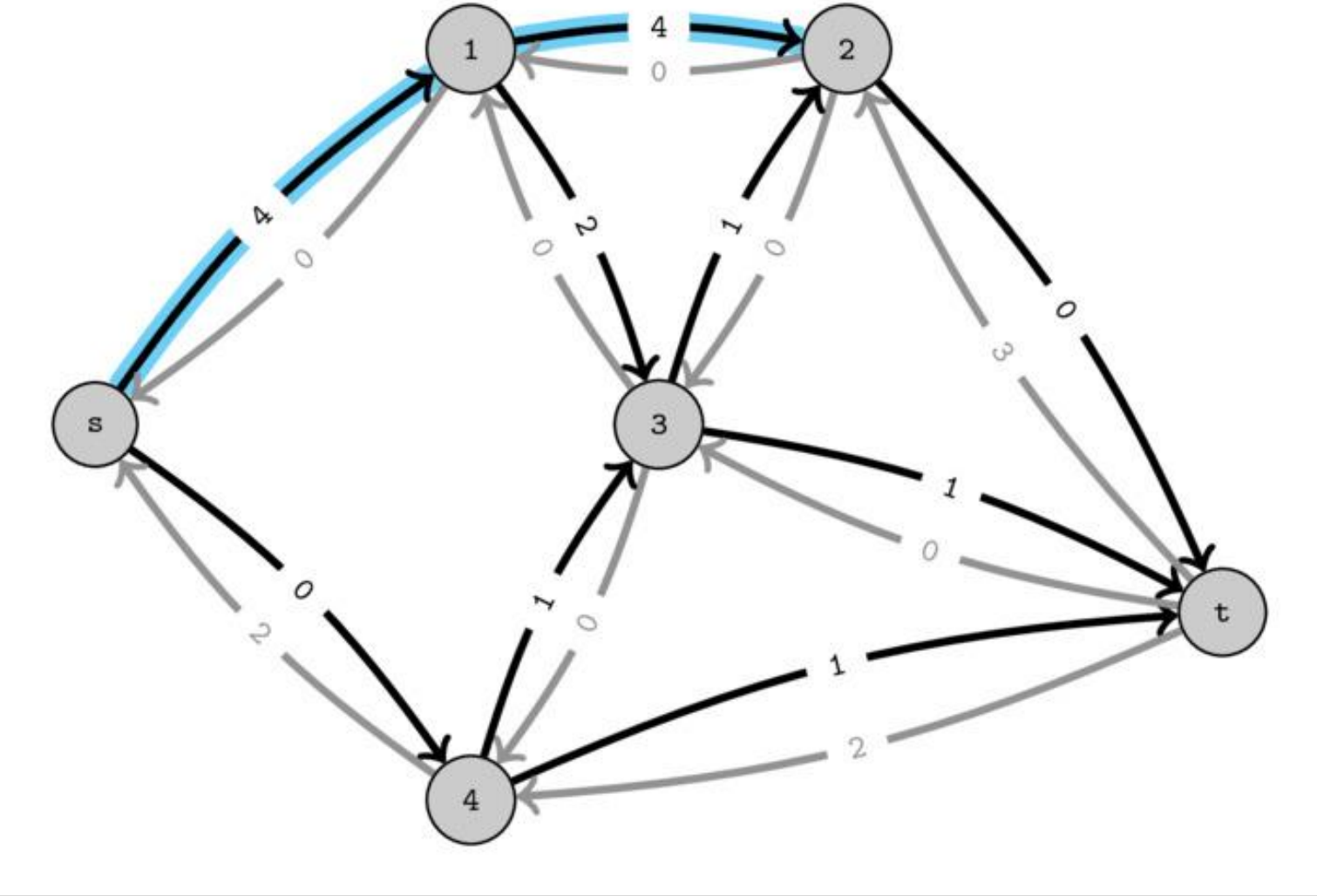
Edmonds Karp



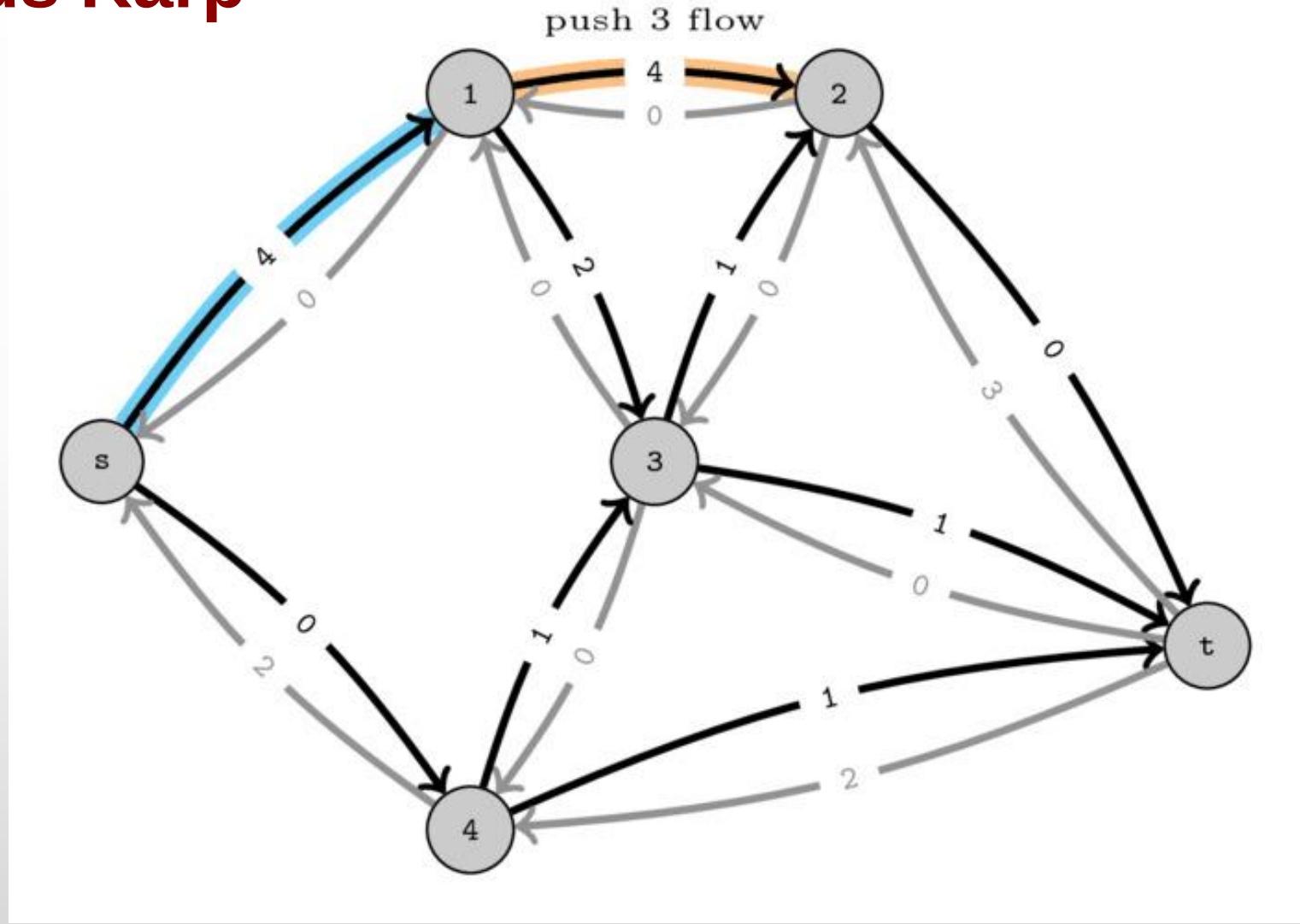
Edmonds Karp



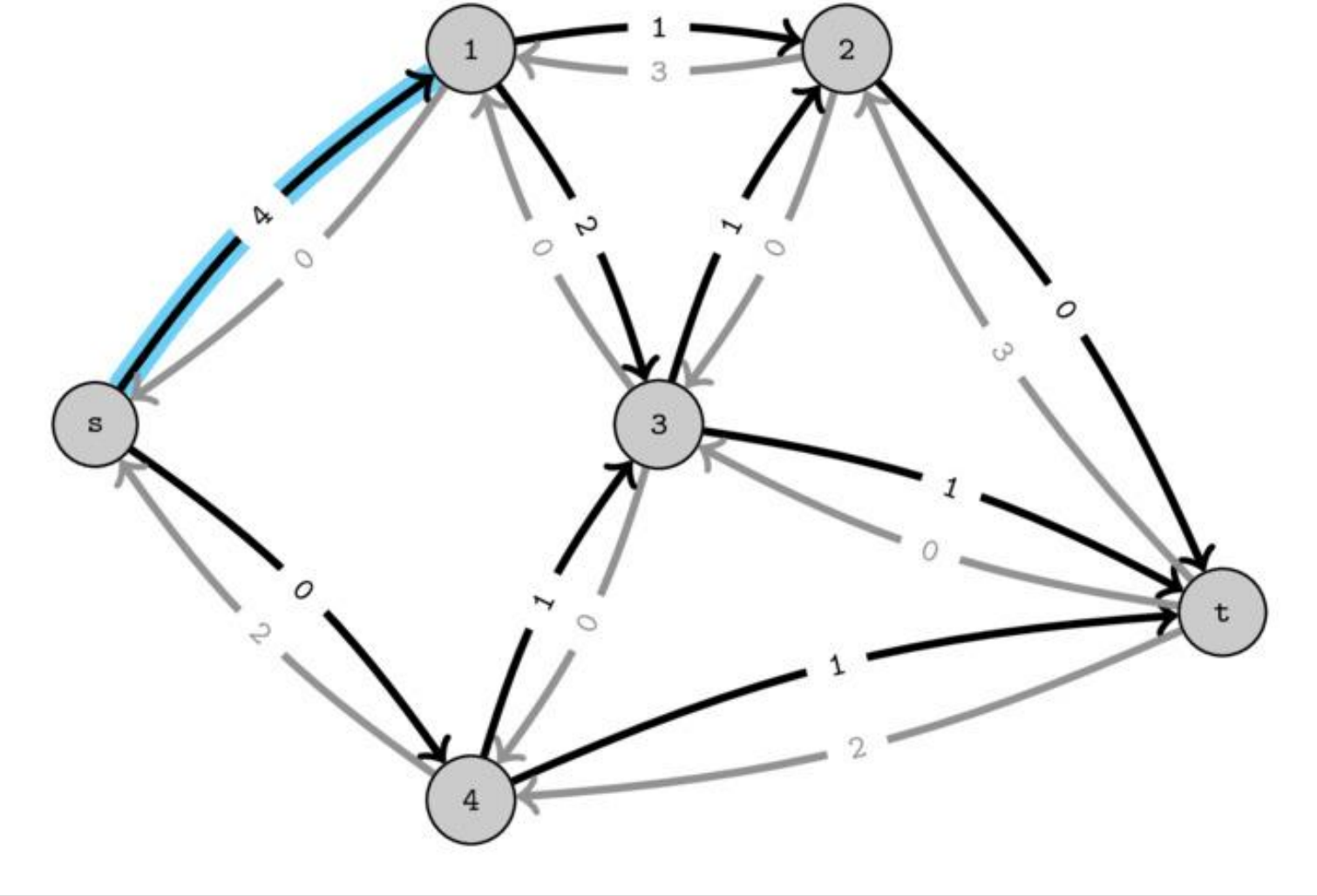
Edmonds Karp



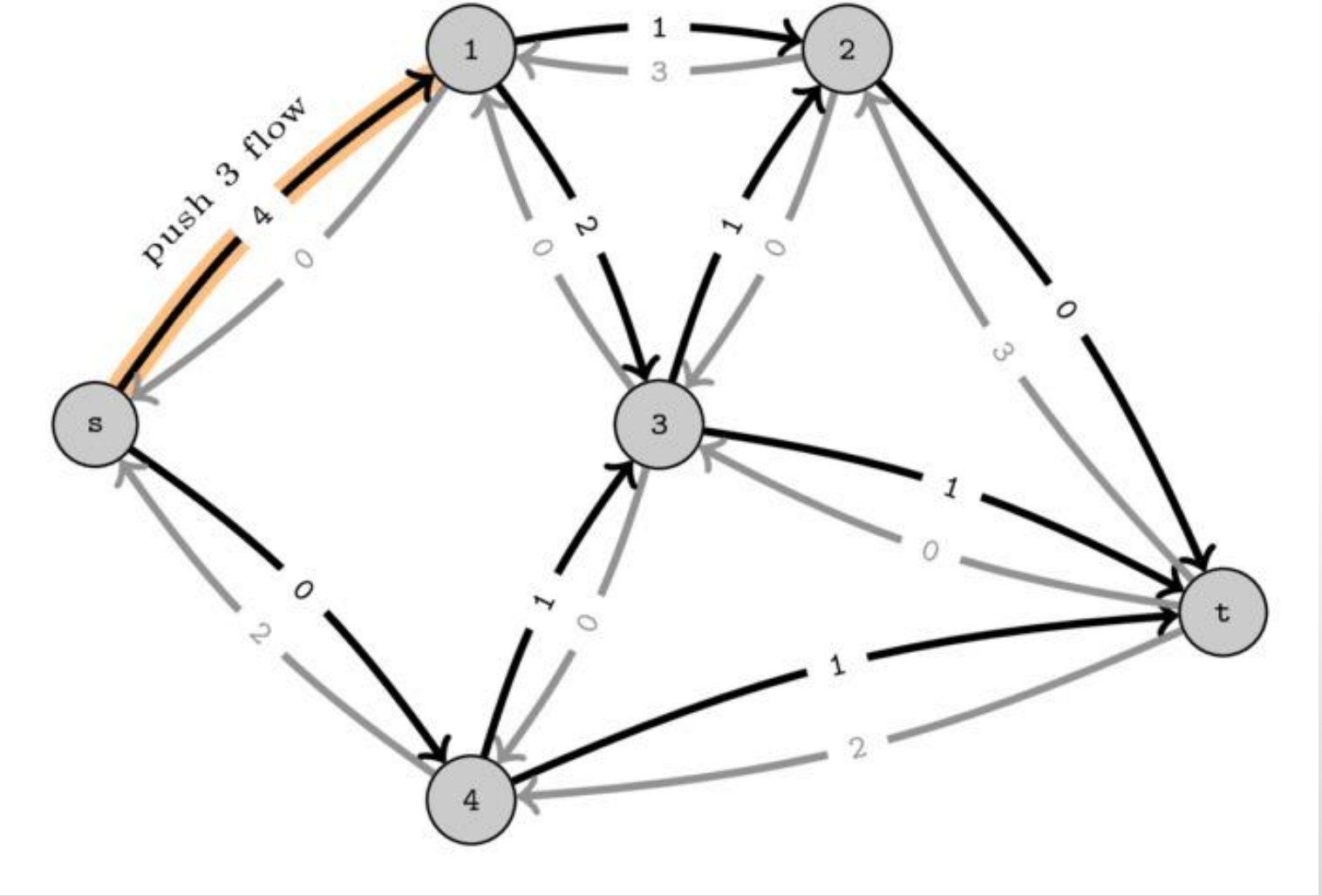
Edmonds Karp



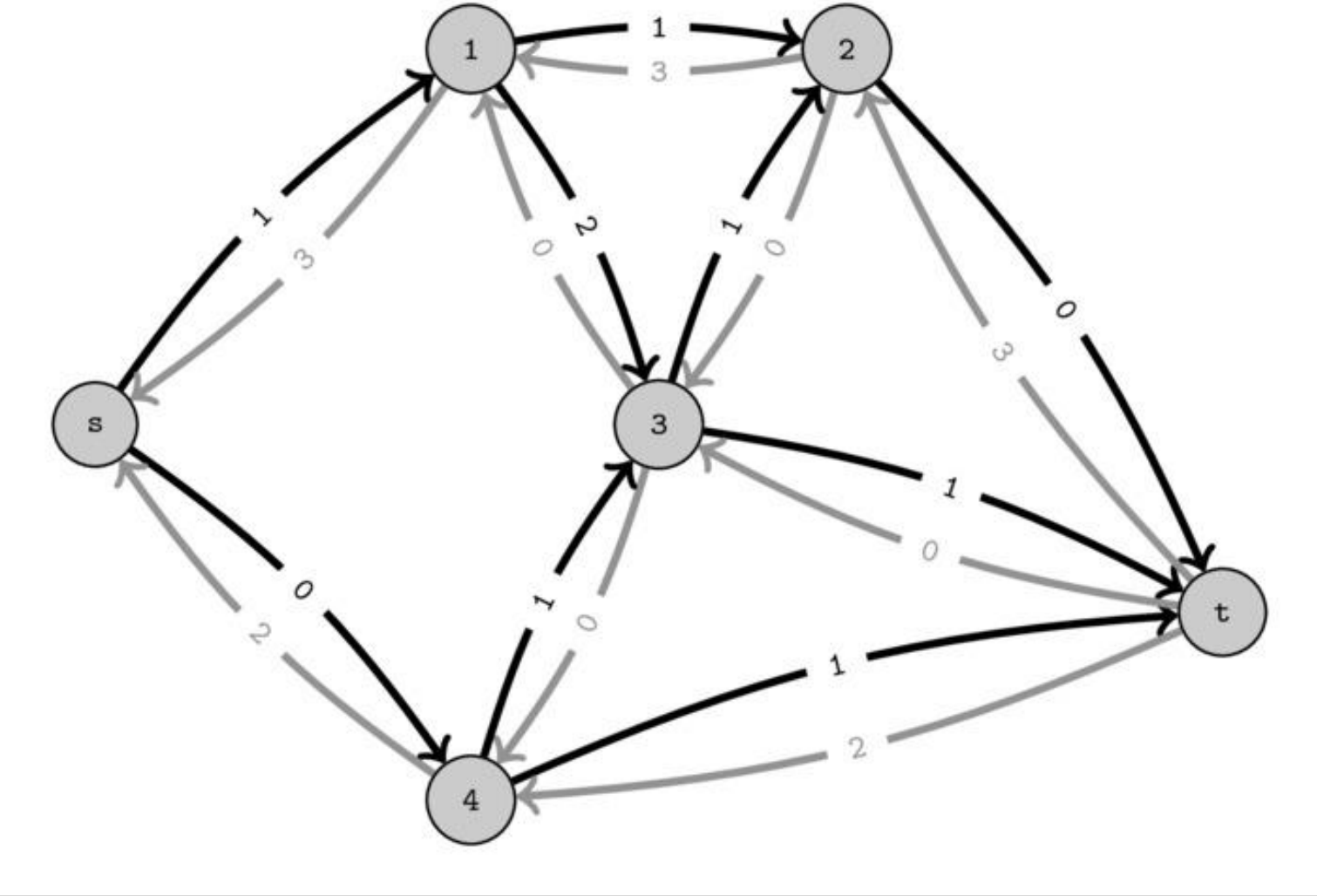
Edmonds Karp



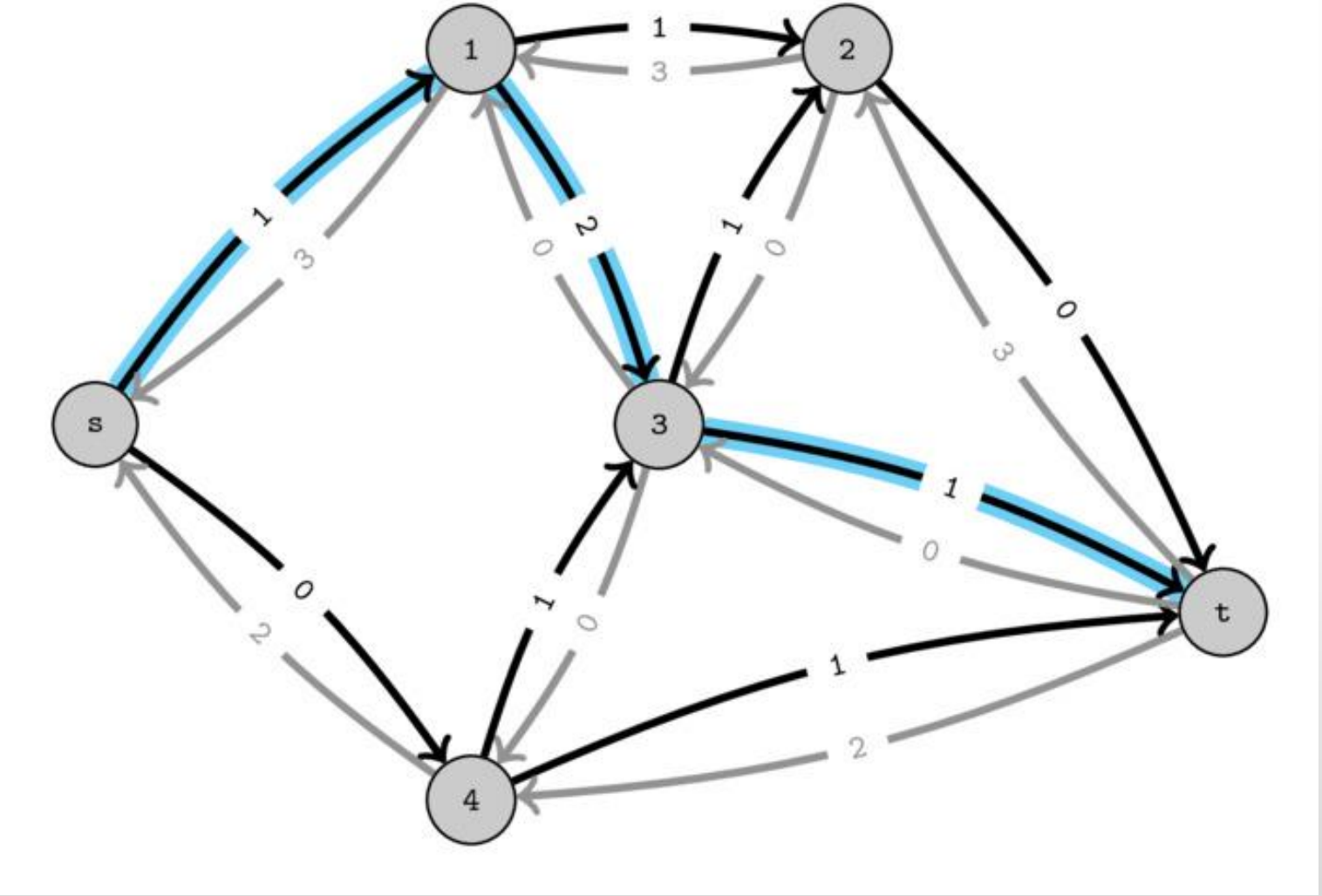
Edmonds Karp



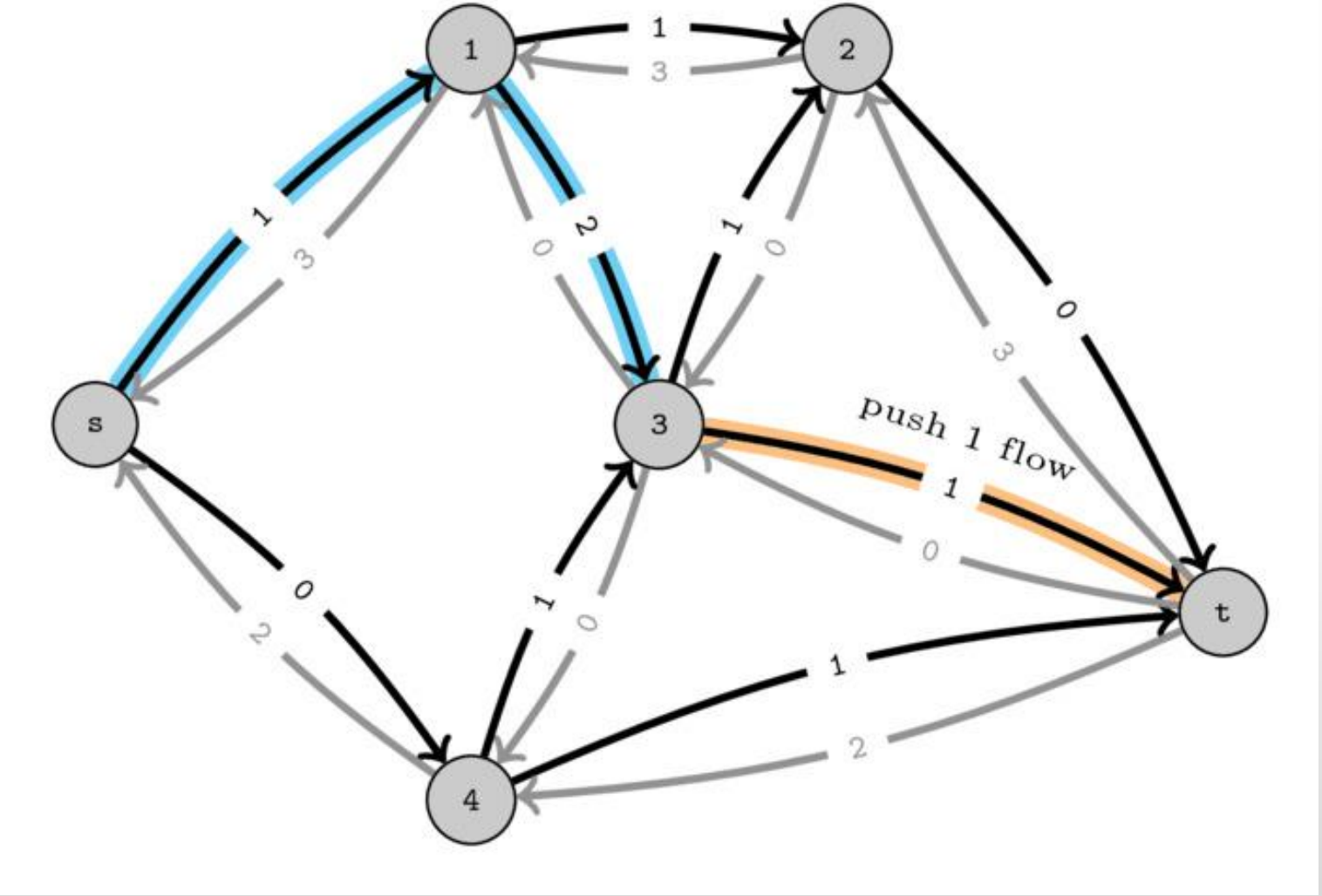
Edmonds Karp



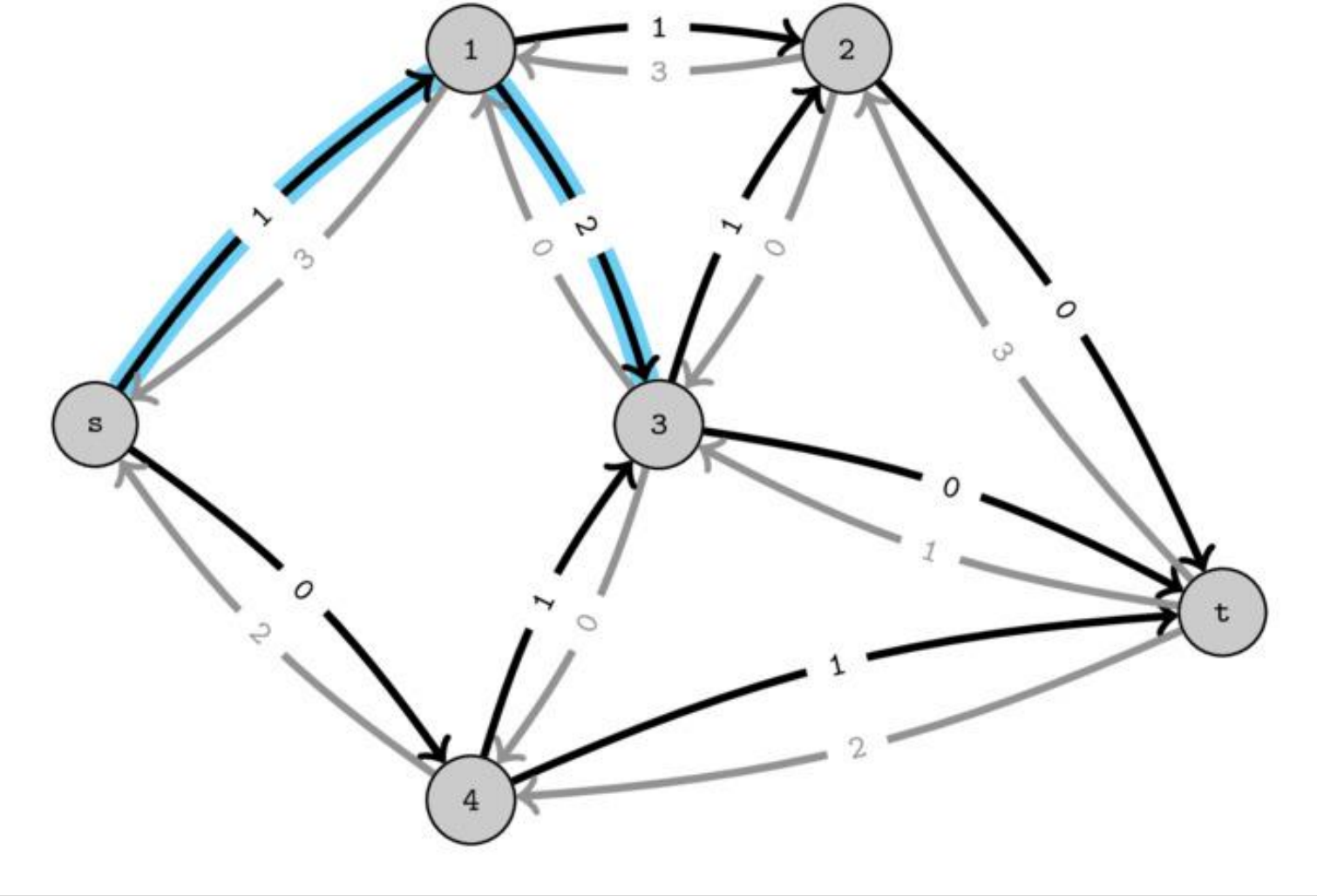
Edmonds Karp



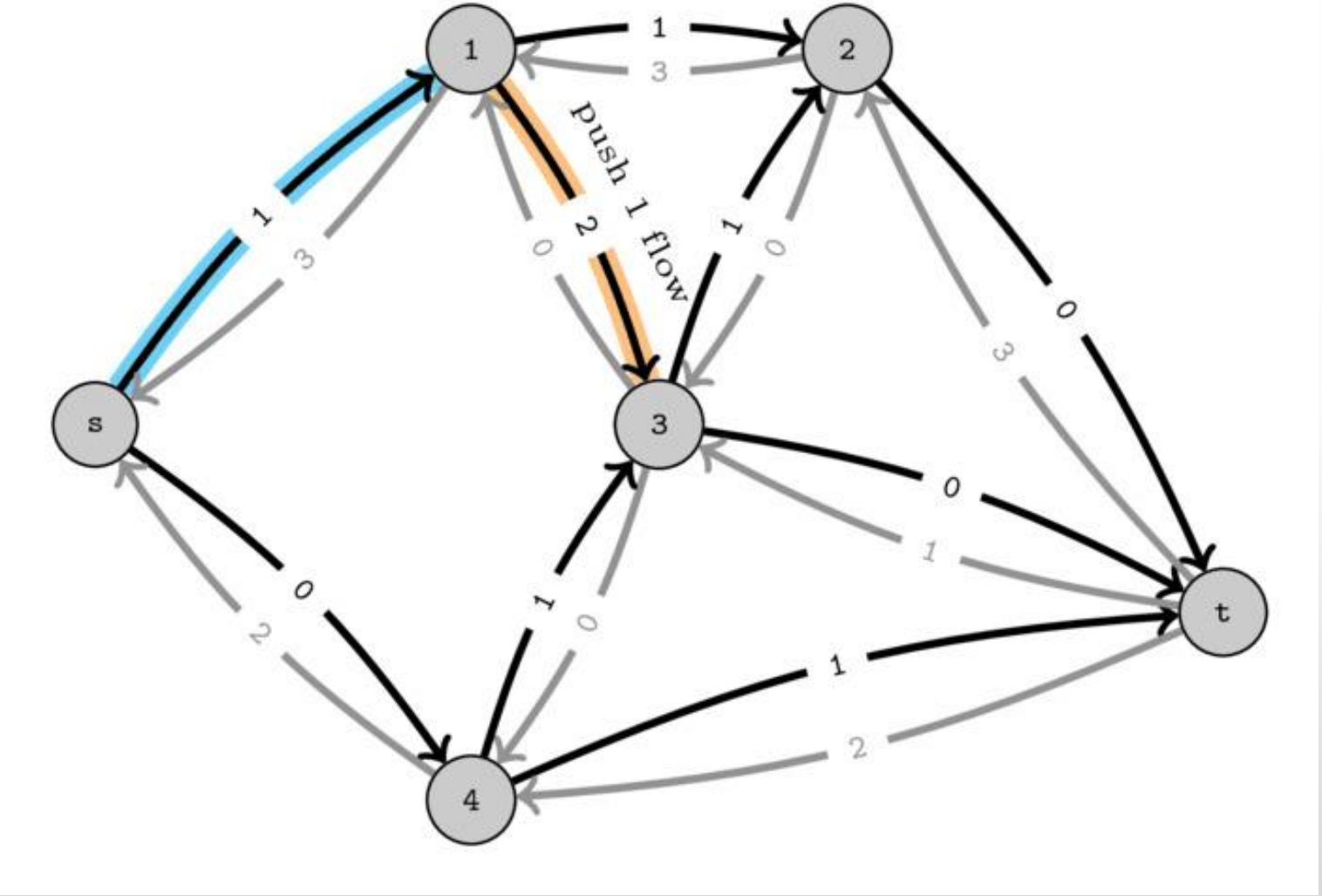
Edmonds Karp



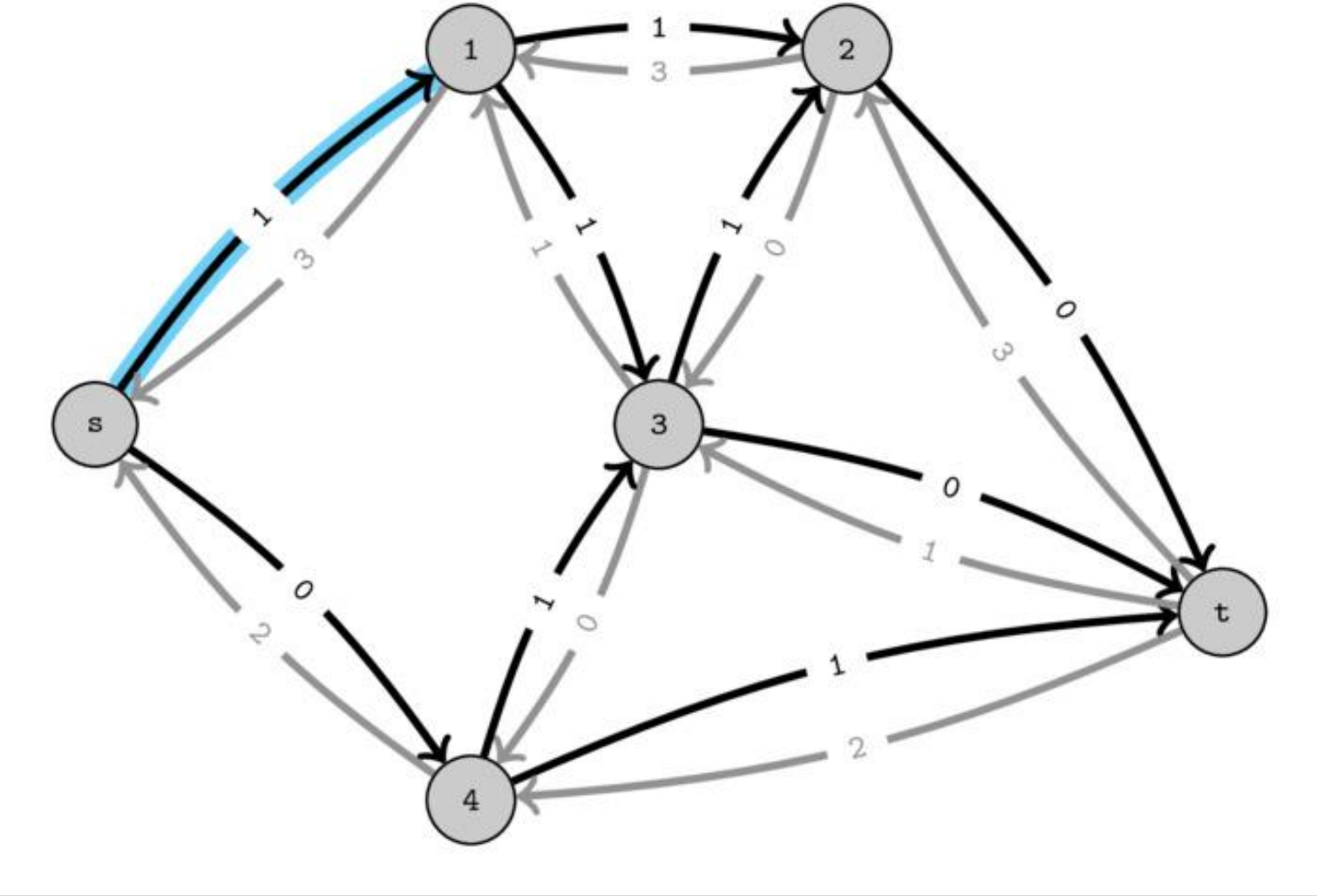
Edmonds Karp



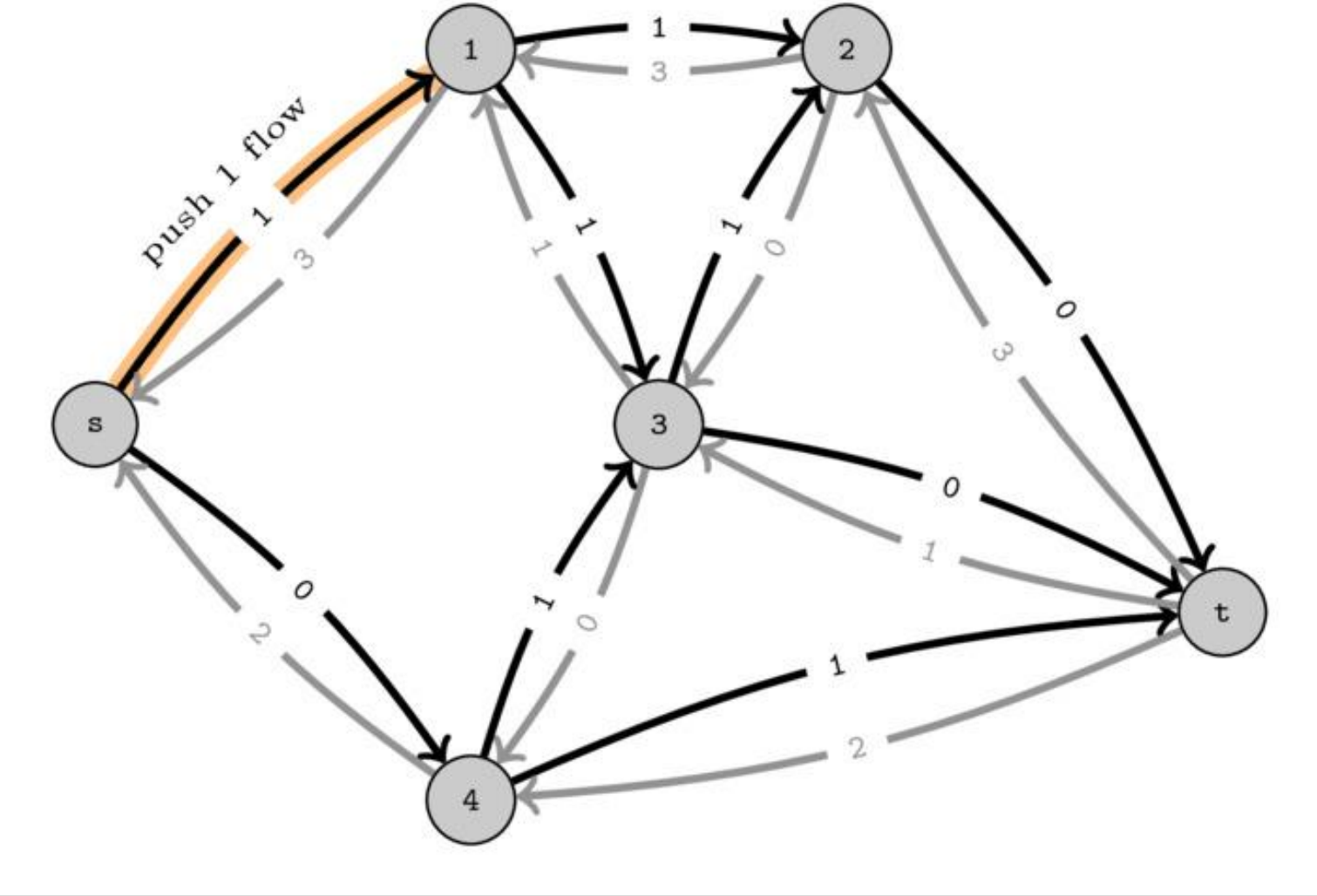
Edmonds Karp



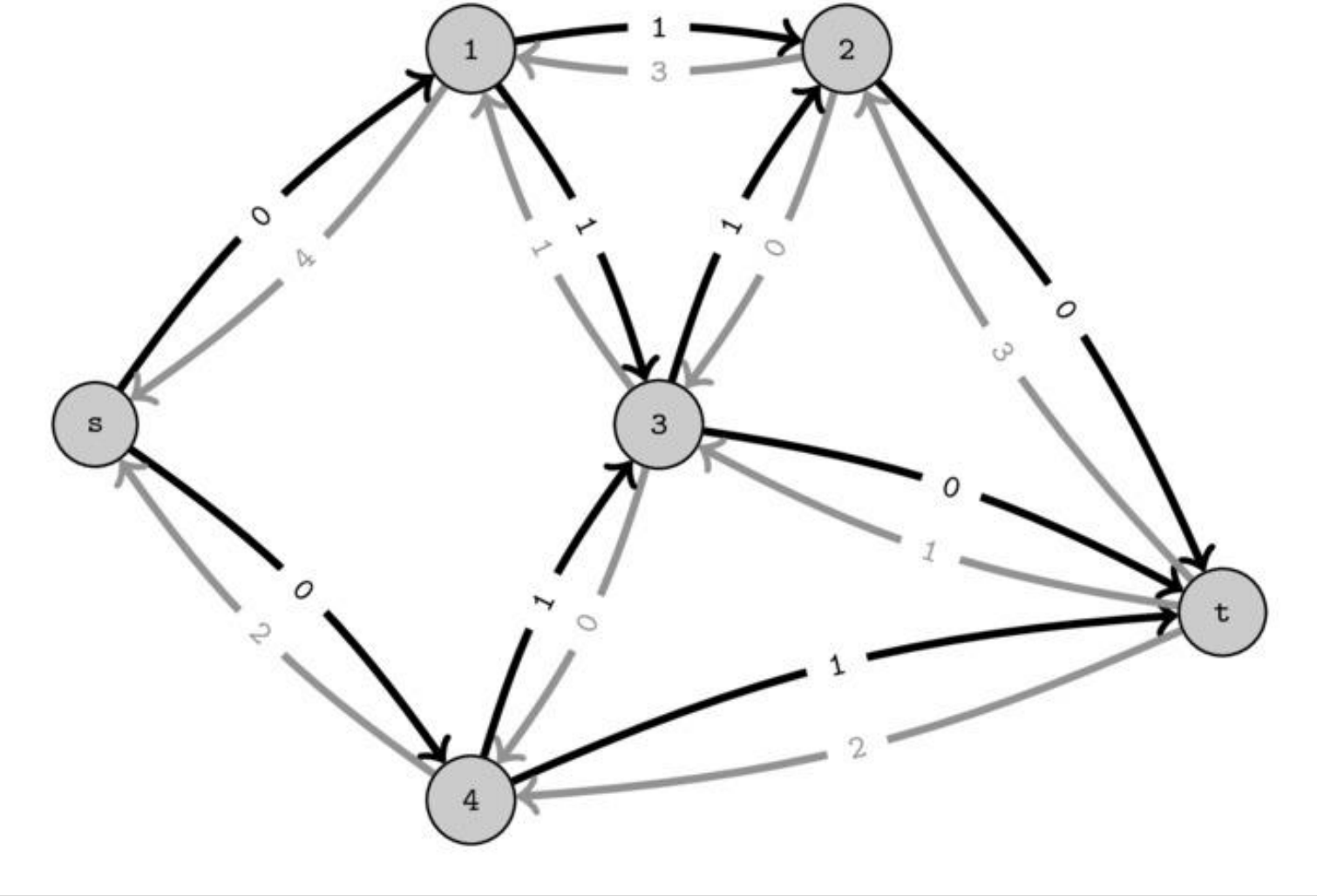
Edmonds Karp



Edmonds Karp



Edmonds Karp



Sözde Kod

EDMONDS_KARP(G, kaynak, hedef):

```
akış = [ ]  // Her kenar için akış (başlangıçta 0)
maksimumAkış = 0

her bir (u, v) için E içinde:
        akış[u, v] = 0
```

Sözde Kod (2)

```
döngü artırıcı yol var iken:
    yol = BFS(graf, kaynak, hedef, akış)
    // Yoldaki minimum kalan kapasiteyi bul
    delta = ∞
    v = hedef
    döngü v ≠ kaynak iken:
        u = yol.ata[v]
        delta = min(delta, kapasite[u, v] - akış[u, v])
        v = u
```

Sözde Kod (3)

```
        // Yoldaki akışları güncelle
        v = hedef
        döngü v ≠ kaynak iken:
                u = yol.ata[v]
                akış[u, v] += delta
                akış[v, u] -= delta  // Ters kenar
                v = u
        maksimumAkış += delta
döndür maksimumAkış, akış
```

Dinic Algoritması

- Bir çizge içinde maksimum akışı bulur.
- Ağırlıklı çizge üzerinde çalışır.
- Karmaşıklığı; $O(V^2 E)$
  - Her aşama BFS gerektirir $O(V + E)$.
  - En kötü durumda V aşama olabilir.
  - BFS tarafından bulunan artış yollarının uzunluğu $O(V)$ olabilir.

Dinic Algoritması

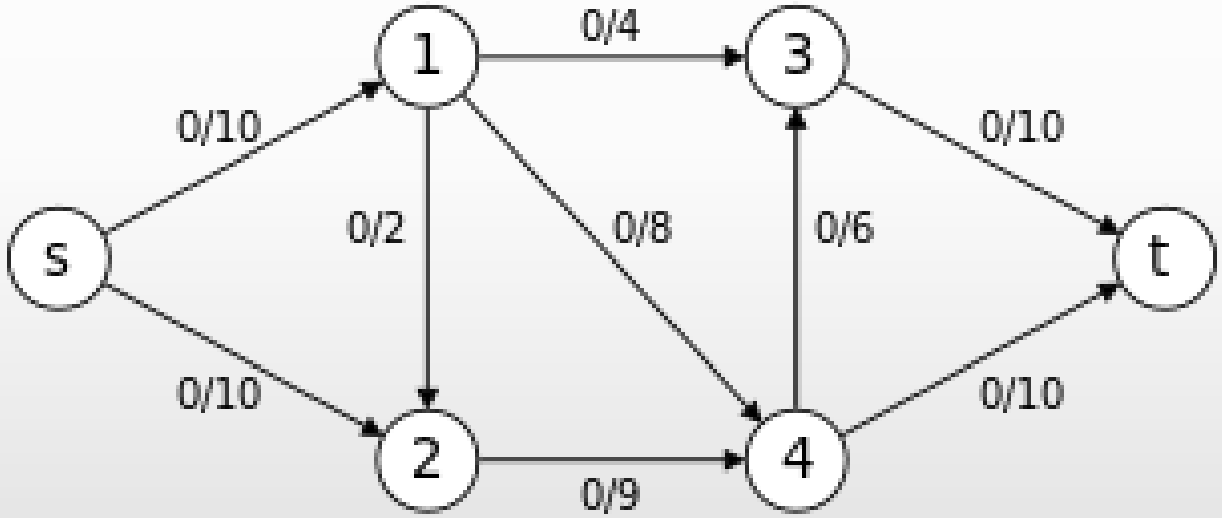
- BFS ile seviye çizgesi oluşturur (hedefe ulaşılabilen düğümler).
- DFS ile engelleyici akışları bulur.
- Ters kenarlarla akışı günceller.
- Seviye çizgesi geçersiz olana kadar devam eder.

Dinic Algoritması

- Başlangıçta çizgenin tüm kenarlarının akışlarına *0* ata.
- Kaynak düğümden başlayarak BFS ile her düğümün seviyesini (kaynağa olan uzaklık) belirle.
- DFS ile her adımda bir seviye ileri giden artan yol bul.
- Artan yollar üzerindeki minimum kapasiteyi bul, toplam ağ akışına ekle.
- Kenarların kapasitesini ve ters kenarların kapasitesini güncelle.
- 2. adıma geri dön ve işlemleri tekrarla.

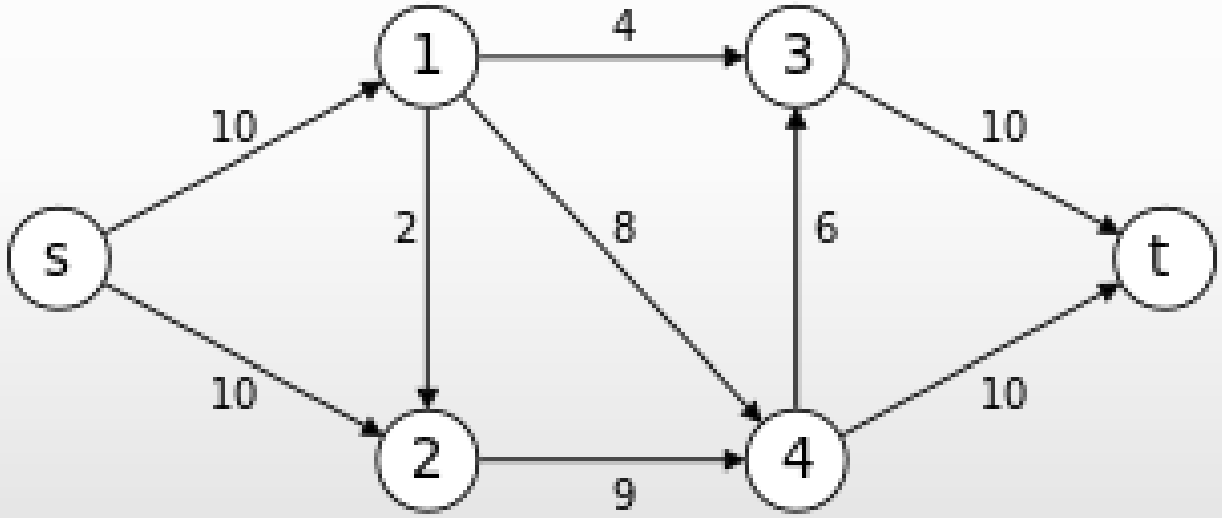
Dinic - Graph

- {s,1,3,t} 4 birim akış, {s,1,4,t} 6 birim akış, {s,2,4,t} 4 birim akış, $|f| = 14$.



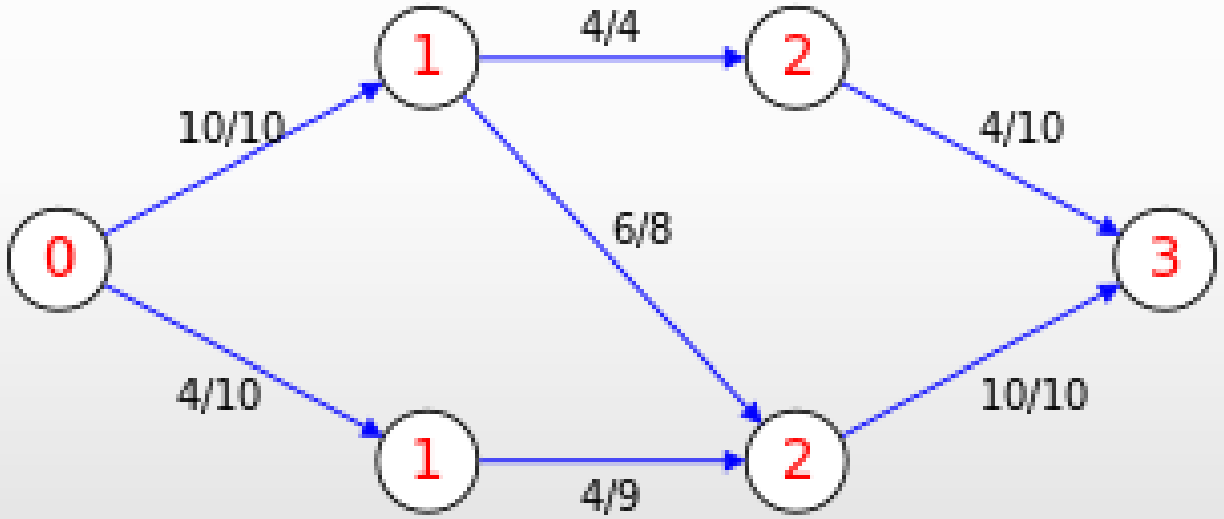
Dinic - $Graph_f$

- Flow graph - akış çizgesi



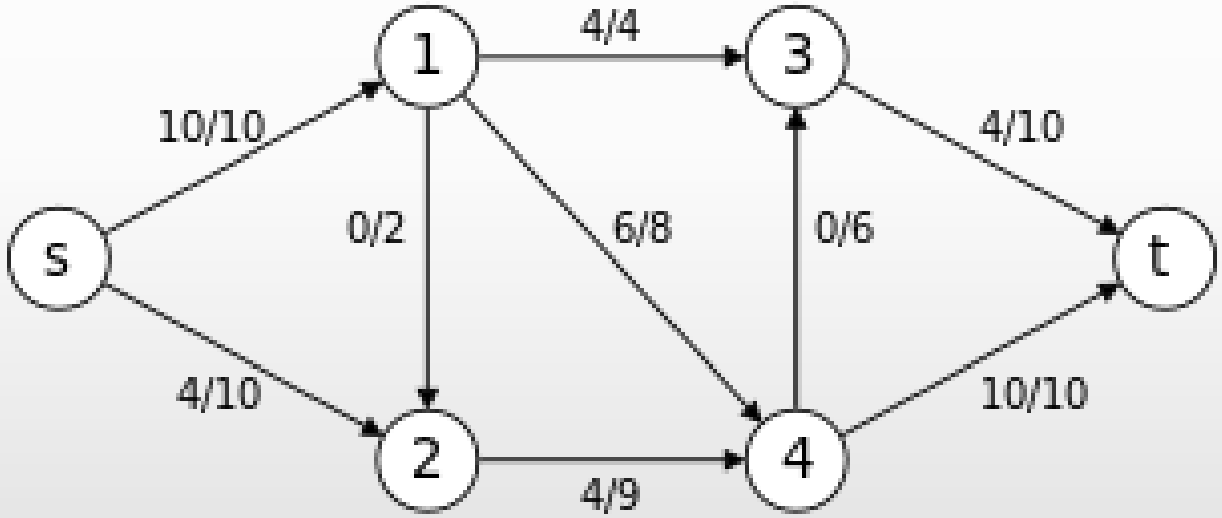
Dinic - $Graph_L$

- Level graph – seviye çizgesi



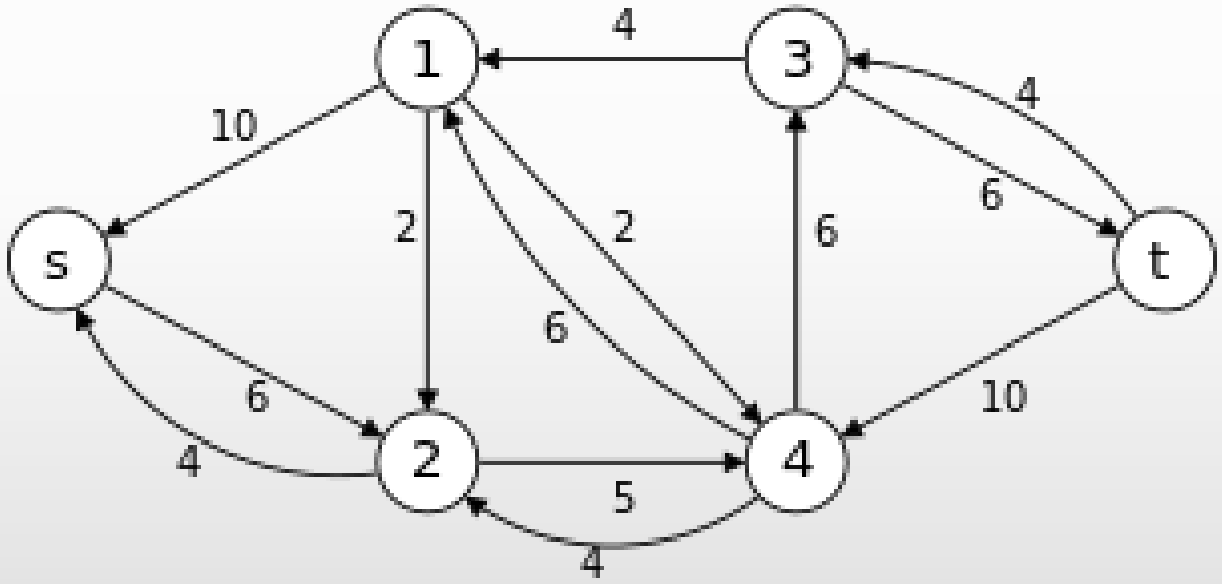
Dinic - Graph

- {s,2,4,3,t} 5 birim akış, $|f| = 14 + 5 = 19$.



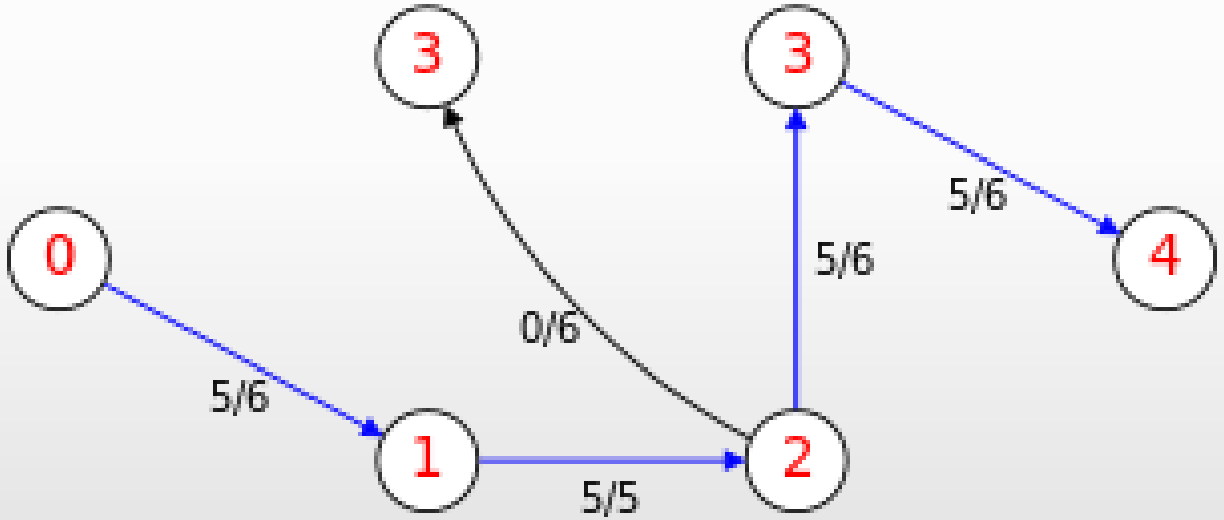
Dinic - $Graph_f$

- Flow graph - akış çizgesi



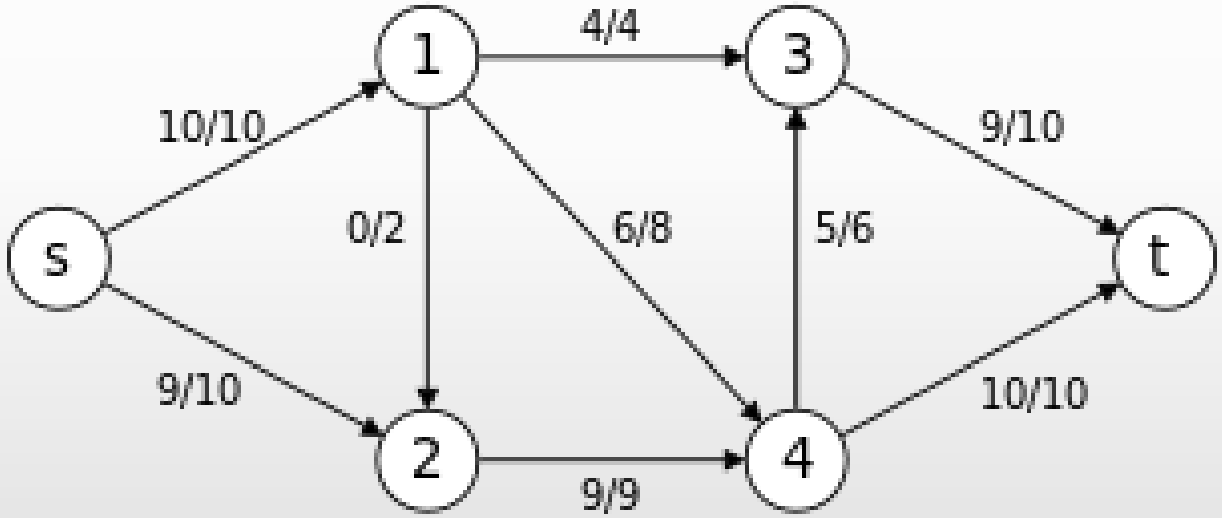
Dinic - $Graph_L$

- Level graph – seviye çizgesi



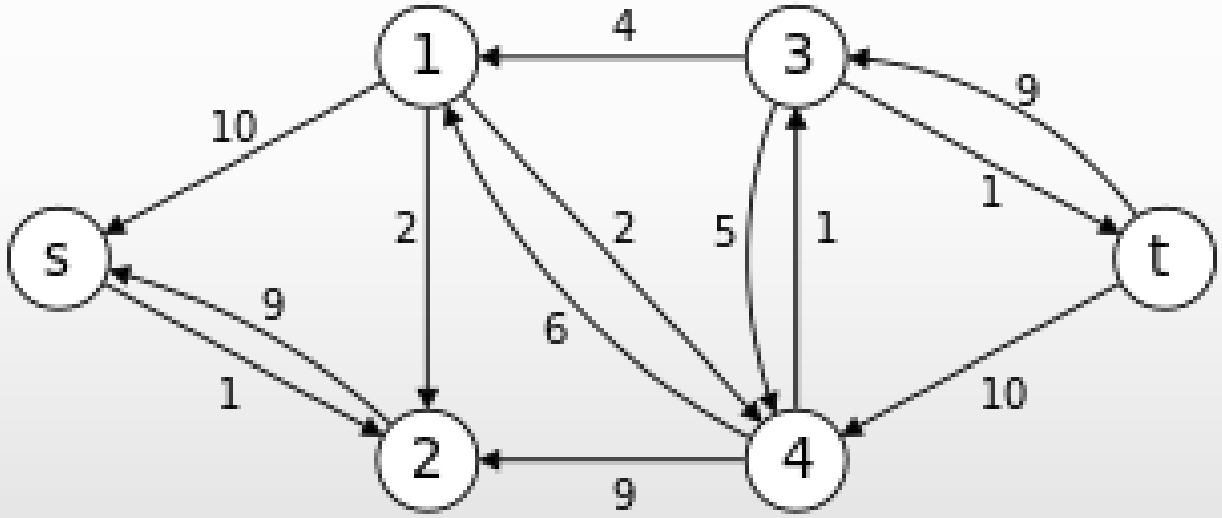
Dinic - Graph

- akış çizgesine bakılarak t'ye erişen başka yol yok. *algorithm terminates*



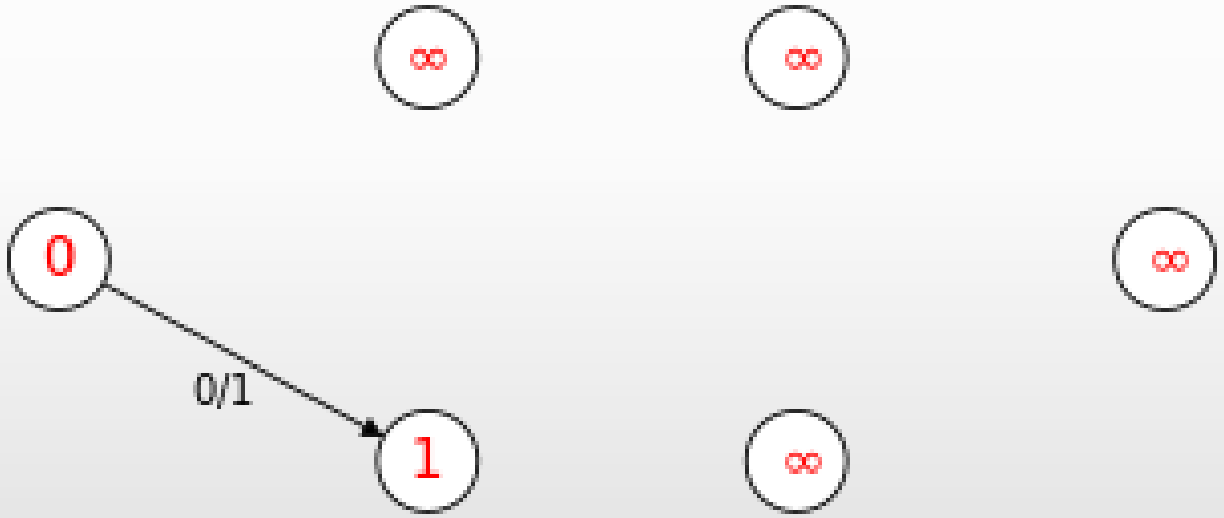
Dinic - $Graph_f$

- Flow graph - akış çizgesi



Dinic - $Graph_L$

- Level graph – seviye çizgesi



Sözde Kod

```
DINIC(G, kaynak, hedef):
        akış = [ ]  // Her kenar için akış
        maksimumAkış = 0

        döngü BFS(G, kaynak, hedef, akış) başarılı iken:
                seviye = BFS ile seviye çizgesini oluştur
                döngü engelleyici akış var iken:
                        delta = DFS(G, kaynak, hedef, akış, seviye, sonsuz)
                        maksimumAkış += delta
        döndür maksimumAkış
```

Sözde Kod (2)

```
BFS(G, kaynak, hedef, akış):
        seviye = [ ]  // Her düğüm için seviye
        seviye[kaynak] = 0
        kuyruk.ekle(kaynak)
        döngü kuyruk boş değil iken:
                u = kuyruk.çıkar()
                her bir (u, v) için G[u] içinde:
                        eğer seviye[v] < 0 ve kapasite[u, v] - akış[u, v] > 0:
                                seviye[v] = seviye[u] + 1
                                kuyruk.ekle(v)
```

Sözde Kod (3)

DFS(G, u, hedef, akış, seviye, limit):

```
    her bir (u, v) için G[u] içinde:
        kalan = kapasite[u, v] - akış[u, v]
        eğer seviye[v] == seviye[u] + 1 ve kalan > 0:
            delta = DFS(G, v, hedef, akış, seviye, min(limit, kalan))
            eğer delta > 0:
                akış[u, v] += delta
                akış[v, u] -= delta  // Ters kenar
                döndür delta
```

İtme ve Yeniden Etiketleme Algoritması

- *Push Relabel Algorithm*
- Her düğümün bir *yükseklik* değeri vardır.
- Akışı artırmak için *yükseklik* değerleri ve *kenar kapasiteleri* dikkate alınır.
- Düşükten yükseğe itme (*push*) ve yüksekten düşüğe etiketleme (*relabel*) adımları yapılır.

İtme ve Yeniden Etiketleme Algoritması

- Başlangıçta, tüm kenarların akışını sıfırlar.
- Kaynak düğümünün yüksekliğini diğer düğümlerden bir fazla yapar.
- İki adımdan oluşur.
  - İtme işlemi:
    - Yükseklik önceliğine göre düğümler arasında akışı iter.
  - Etiketleme işlemi:
    - Çizgenin yüksekliğini günceller ve
    - İtme işlemine devam eder.

İtme İşlemi (Push)

- Bir düğümün yüksekliği, bitiş düğümüne en kısa yolun uzunluğuna eşittir.
- Eğer bir kenar üzerinde potansiyel bir akış varsa (kenar kapasitesi ile akışın toplamı arasında fark varsa), akış artırılır.

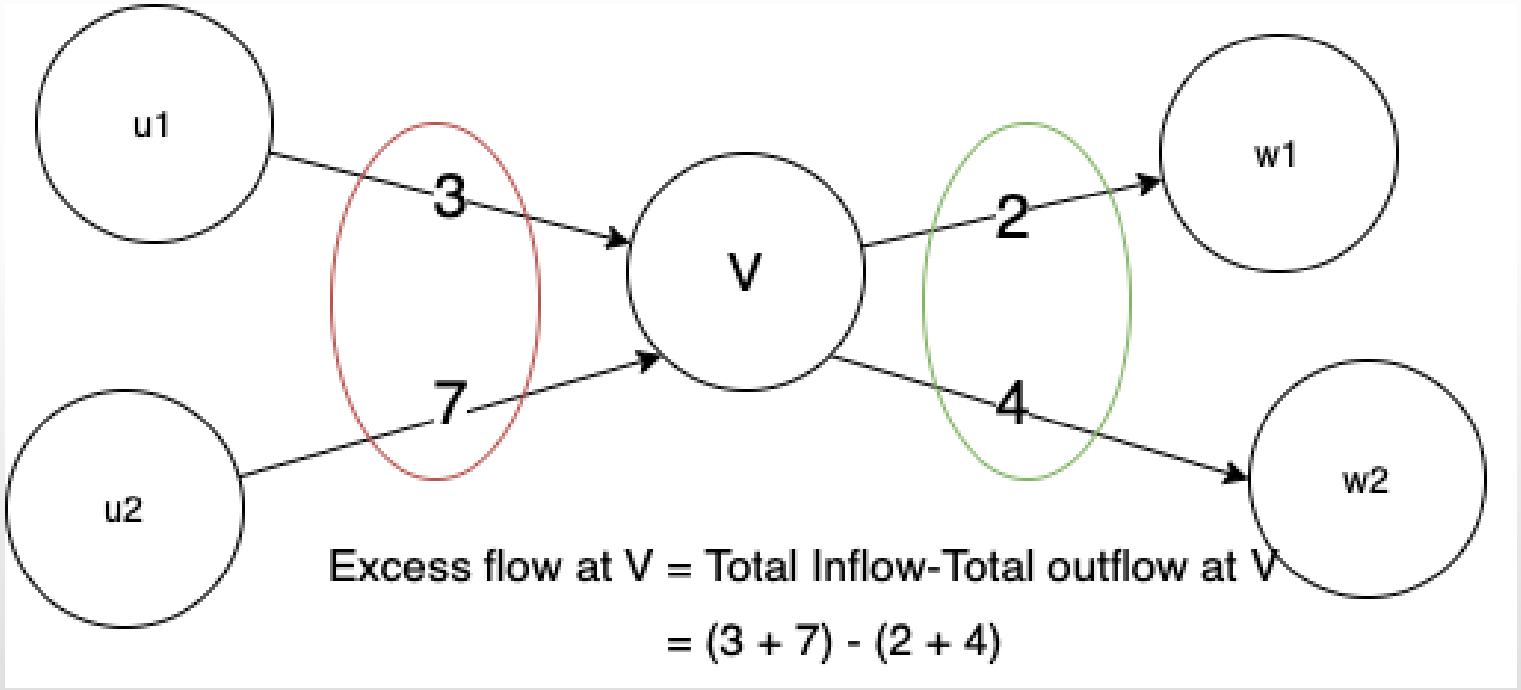
Etiketleme İşlemi (Relabel)

- Bir düğümün yüksekliği, bitiş düğümüne olan en kısa yolun uzunluğuna eşit değilse, yükseklik değeri güncellenir.
- Potansiyel bir itme işlemi için yeni bir yol bulmak için yapılır.

Algoritma Karmaşıklığı

- En kötü durumda $O(V^2 E)$ karmaşıklığa sahiptir.
- Her itme işlemi bir kenarı taramak ve potansiyel olarak akışı artırmak için birçok adım gerektirebilir. O(V)
- Etiketleme işlemi, bir düğümün yüksekliğini güncellemek için kullanılır. Her düğüm için tüm kenarları kontrol etmek zorunda kalabilir. O(V E)
- Her itme ve etiketleme işlemi, tüm düğümleri ve kenarları tarayabilir.
- Toplam işlem sayısı, tüm düğümler ve kenarlar üzerinde yapılacak işlemlerin toplamıdır.

Aşırı Akış (Excess Flow)

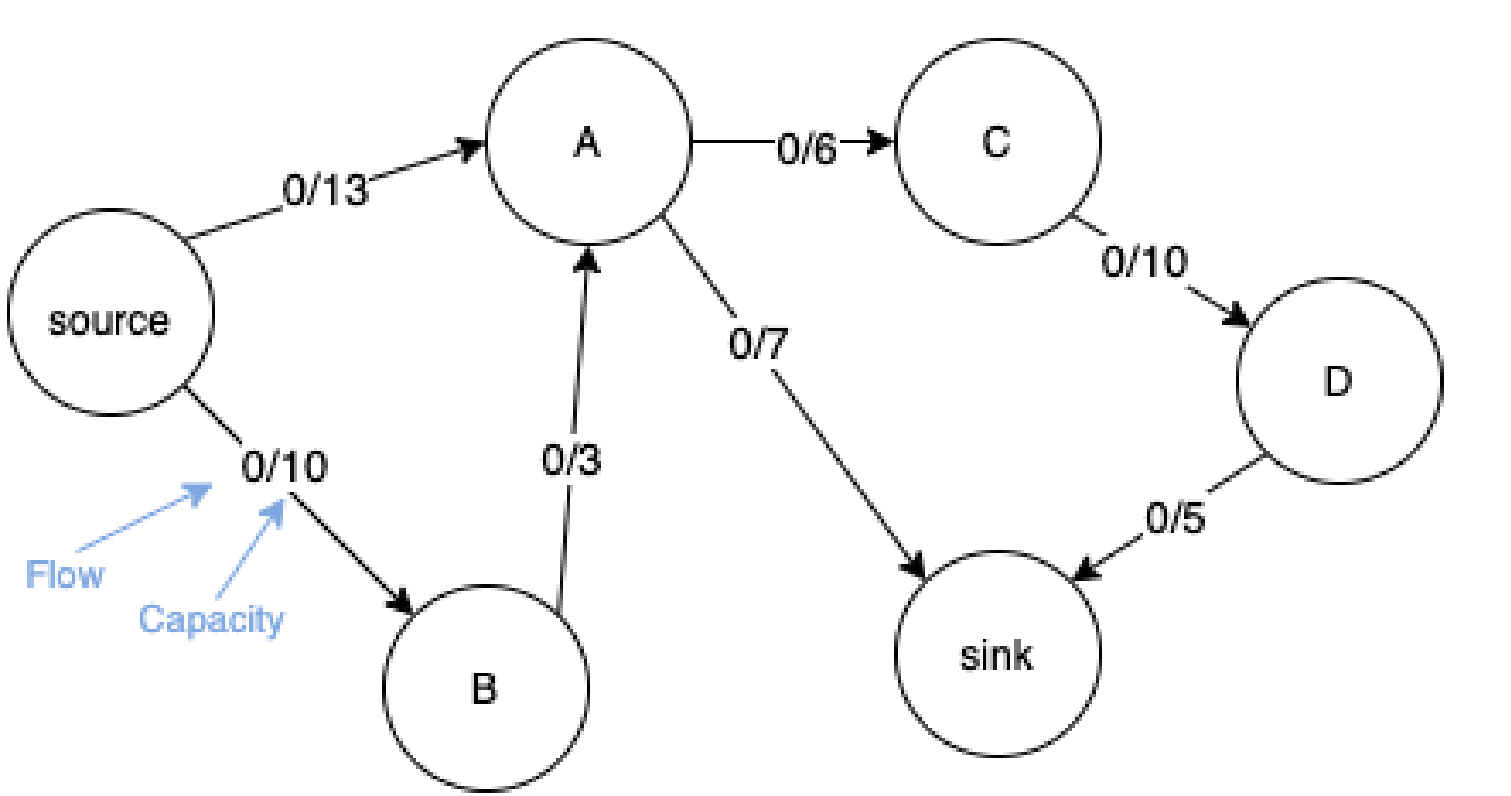


İtme ve Yeniden Etiketleme Algoritması

- Kaynaktan aşırı akış başlatır, yükseklik etiketleriyle yönlendirir.
- Aşırı akışı uygun komşulara iter (push).
- İtemiyorsa, düğümün yüksekliğini artırır (relabel).
- Hedefteki aşırı akış, maksimum akışı verir.

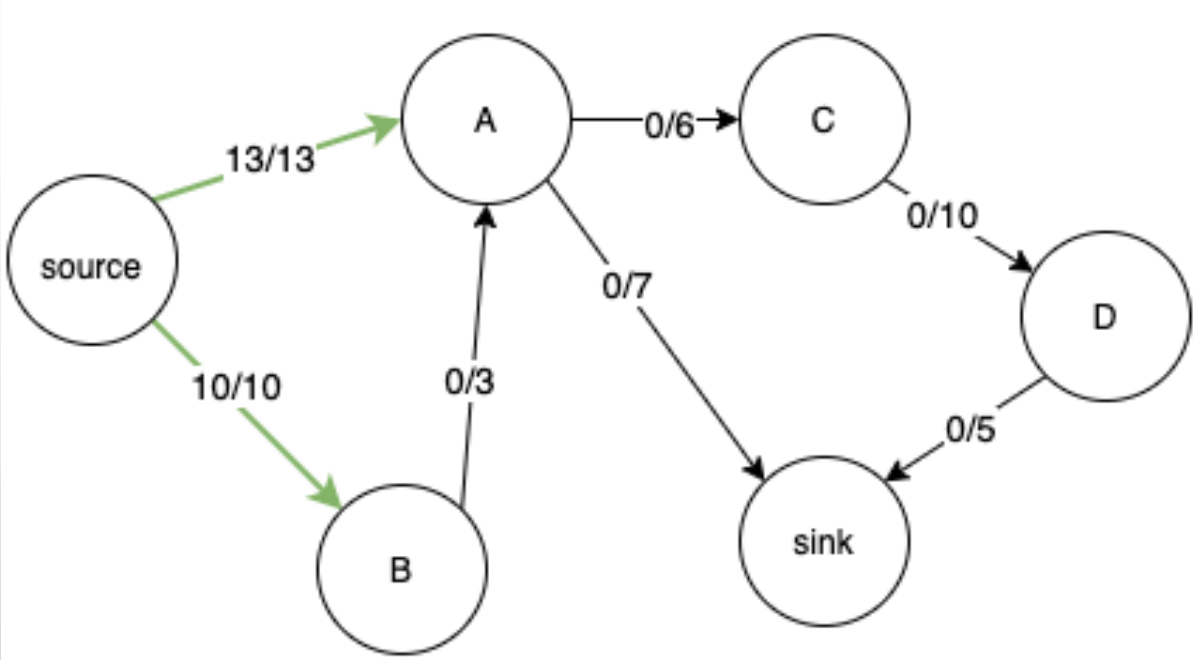
Push Relabel

- Aşağıdaki çizge verilmiş olsun.



Push Relabel

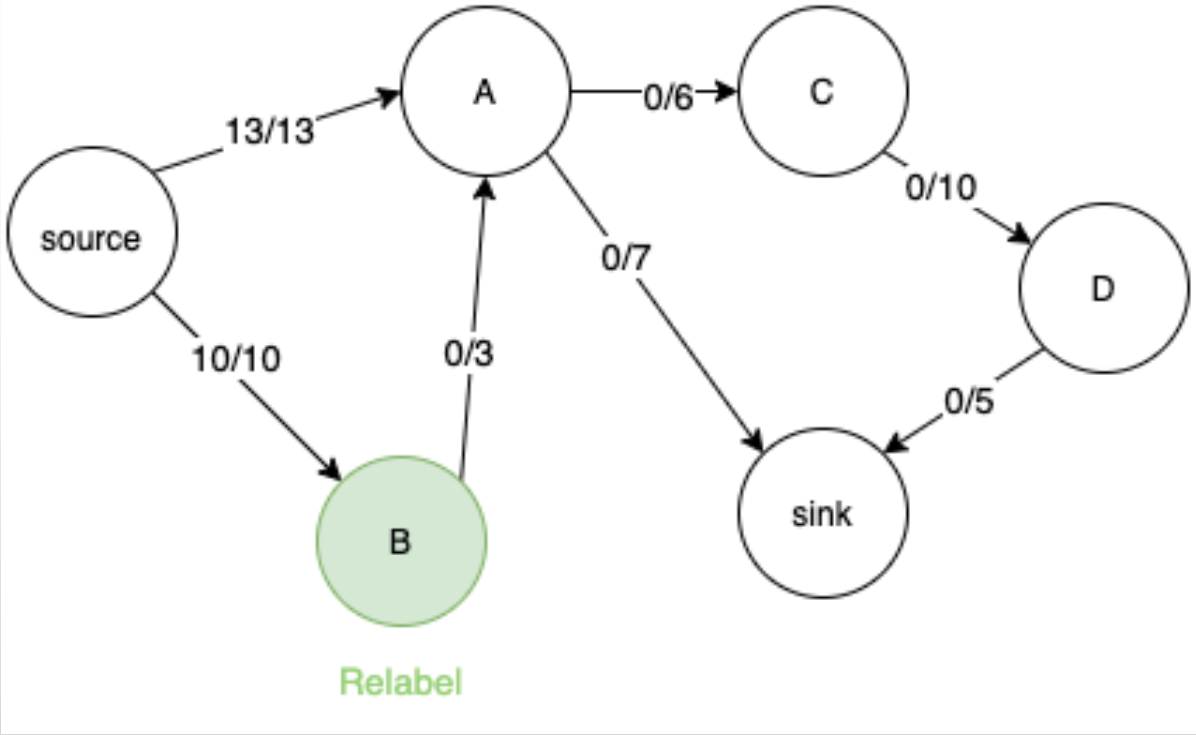
- Yükseklik ve artık akış ilk değerleri atanır.



Node	Height	Excess Flow
source	6	
A	0	13
B	0	10
C	0	
D	0	
sink	0	

Push Relabel

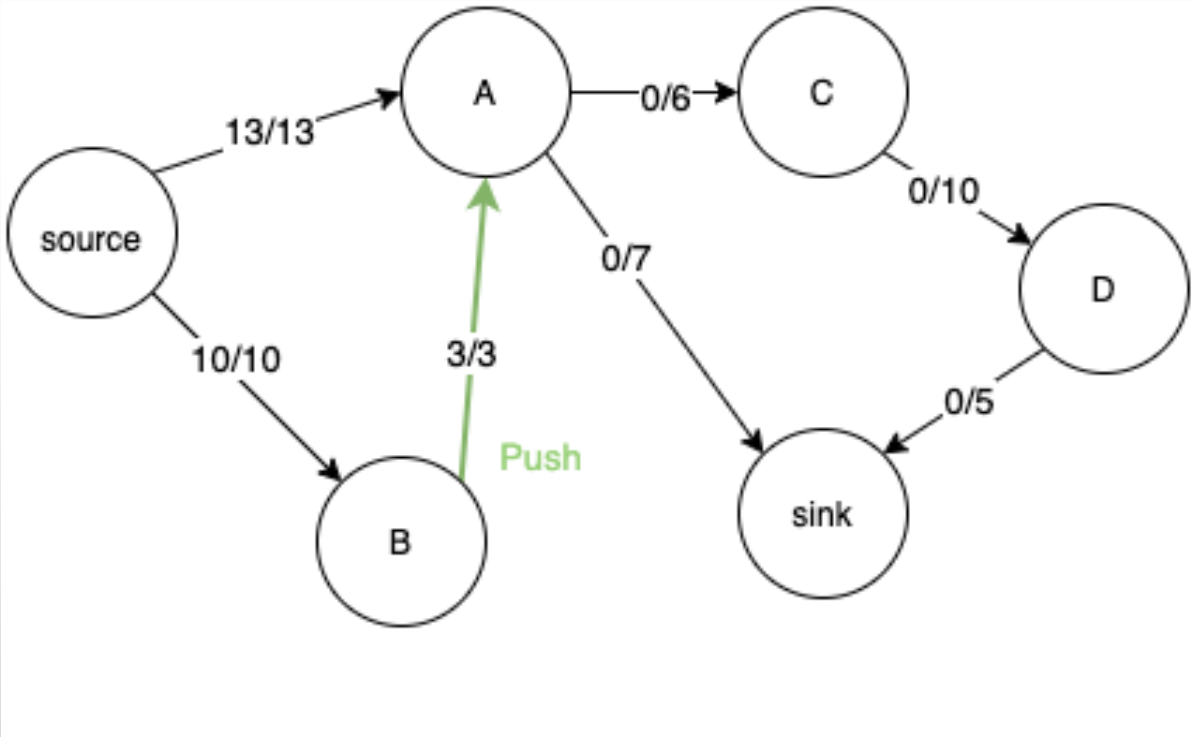
- B düğümü ele alınır. A ile aynı yüksekliğe sahip olduğundan artık akışı A ya gönderemez. Bu nedenle *relabel* yapılır.



Node	Height	Excess Flow
source	6	-
A	0	13
B	1	10
C	0	
D	0	
sink	0	-

Push Relabel

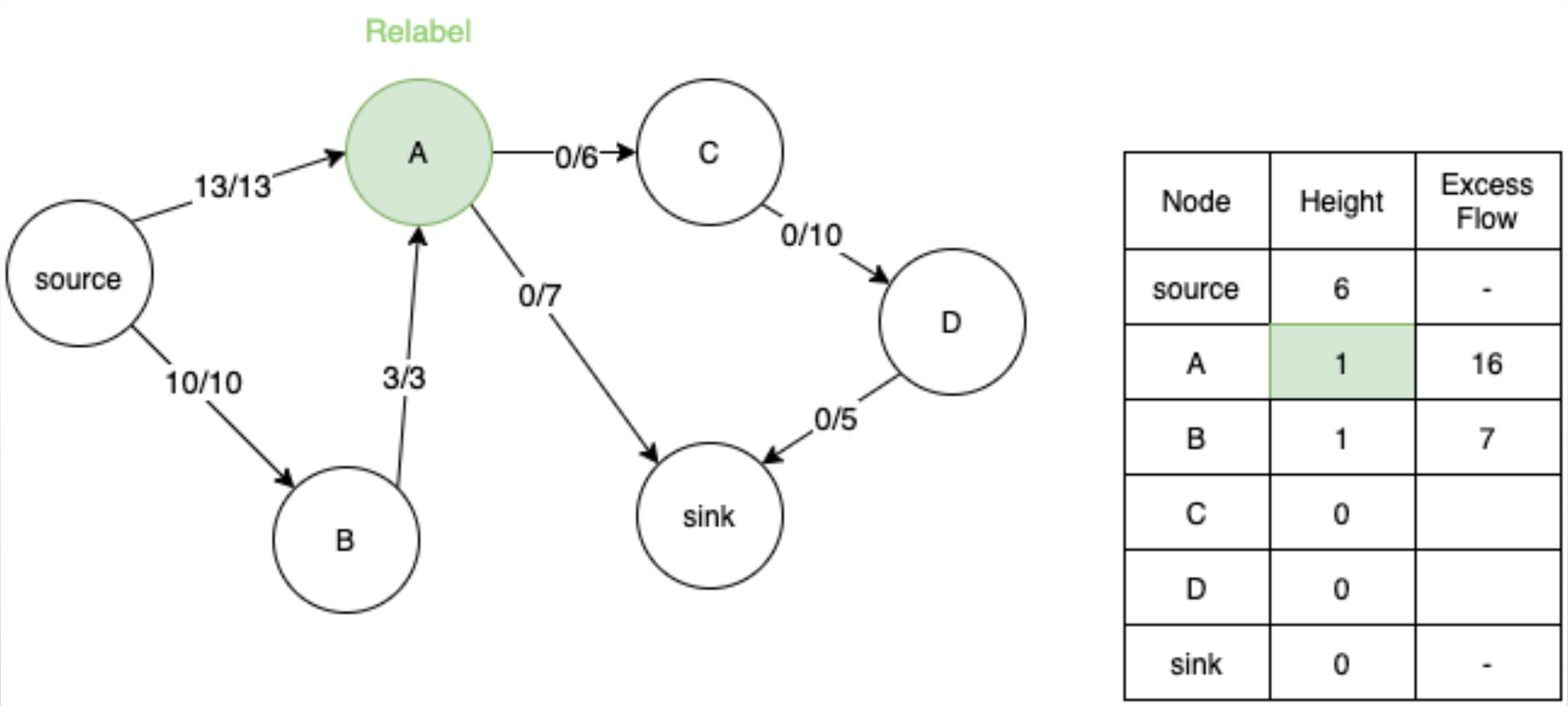
- B düğümü şimdi artık akışı A'ya gönderebilir.



Node	Height	Excess Flow
source	6	-
A	0	16
B	1	7
C	0	
D	0	
sink	0	-

Push Relabel

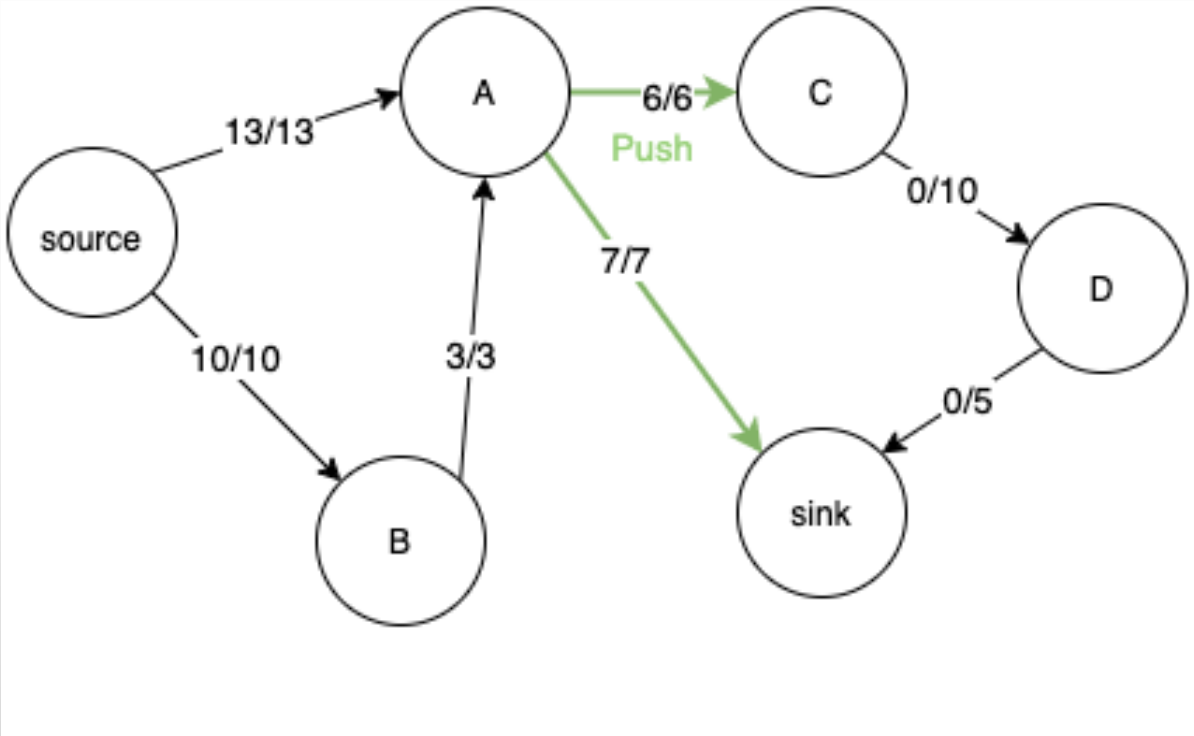
- A düğümü ele alınır. *relabel* yapılarak yüksekliği 1 atanır.



Node	Height	Excess Flow
source	6	-
A	1	16
B	1	7
C	0	
D	0	
sink	0	-

Push Relabel

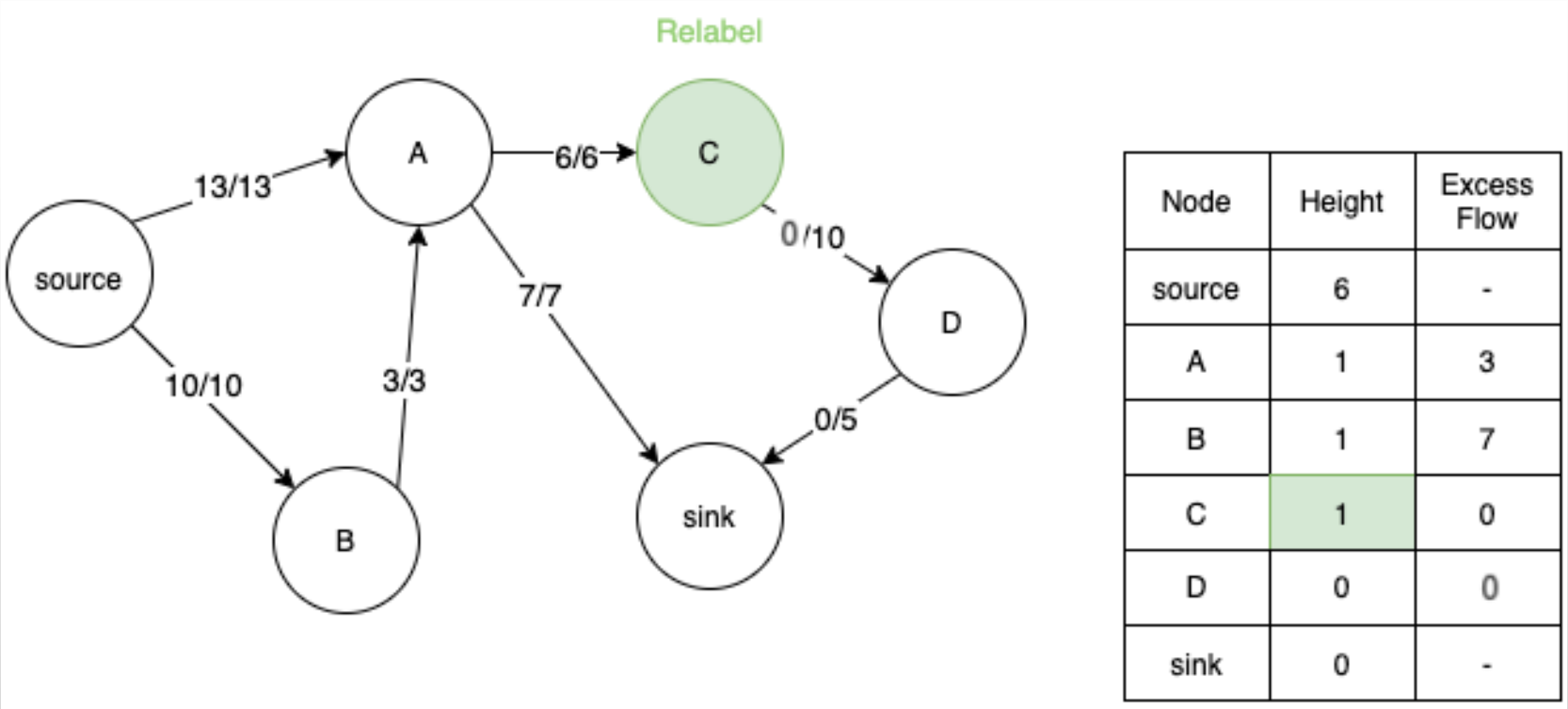
- A düğümü şimdi C ve sink'e akış yapabilir.



Node	Height	Excess Flow
source	6	-
A	1	3
B	1	7
C	0	6
D	0	
sink	0	-

Push Relabel

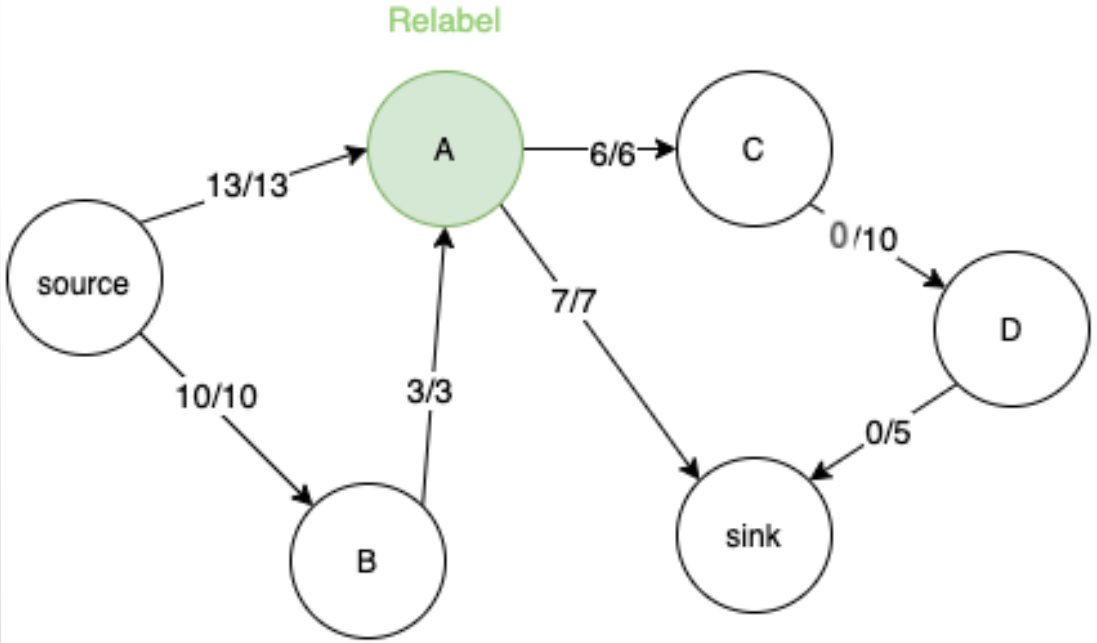
- C düğümü relabel yapılır. A ile aynı yüksekliğe sahip olduğundan A'dan C'ye akış olmaz.



Node	Height	Excess Flow
source	6	-
A	1	3
B	1	7
C	1	0
D	0	0
sink	0	-

Push Relabel

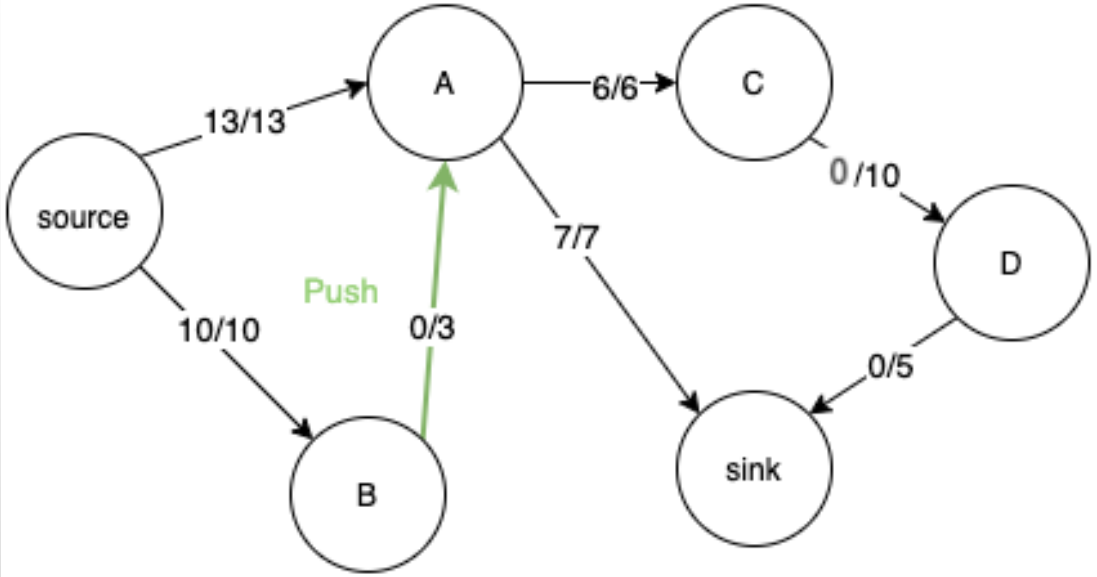
- A düğümü relabel yapılır.



Node	Height	Excess Flow
source	6	-
A	2	3
B	1	7
C	1	0
D	0	0
sink	0	-

Push Relabel

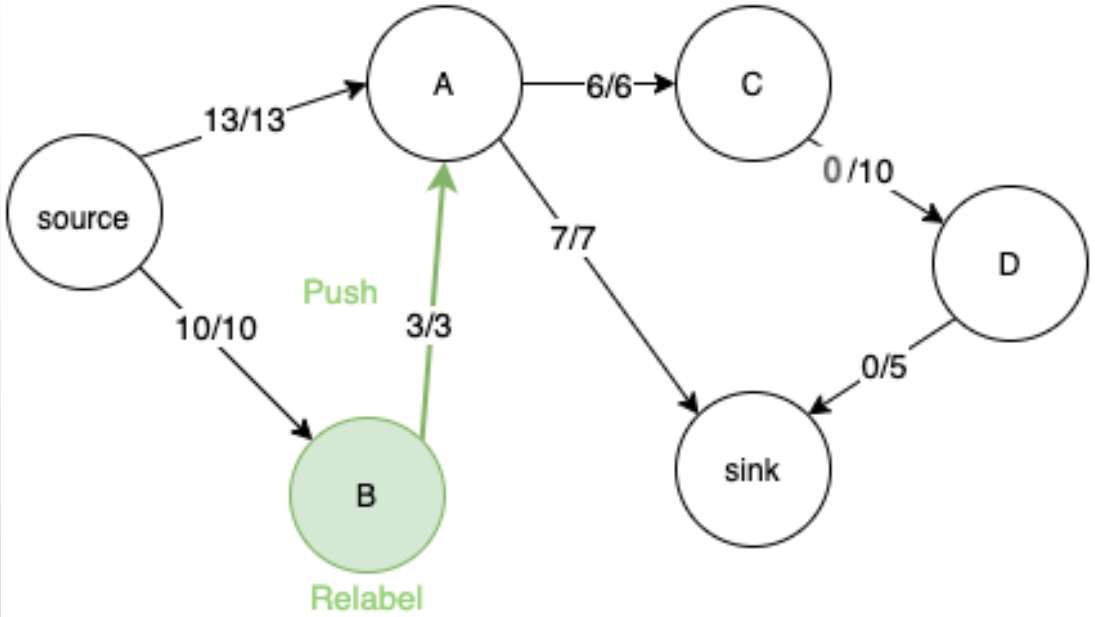
- A'nın yüksekliği B'den fazla olduğu için artık akışı B'ye gönderebilir.



Node	Height	Excess Flow
source	6	-
A	2	0
B	1	10
C	1	0
D	0	0
sink	0	-

Push Relabel

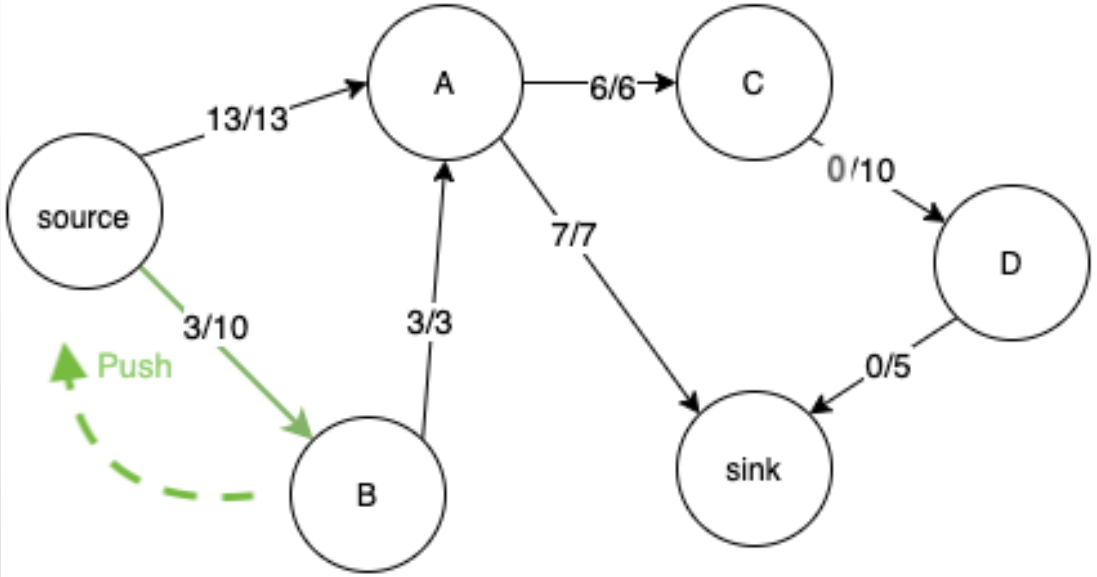
- B'nin akışı göndereceği başka kenar olmadığı için relabel yapılır.



Node	Height	Excess Flow
source	6	-
A	2	3
B	3	7
C	1	0
D	0	0
sink	0	-

Push Relabel

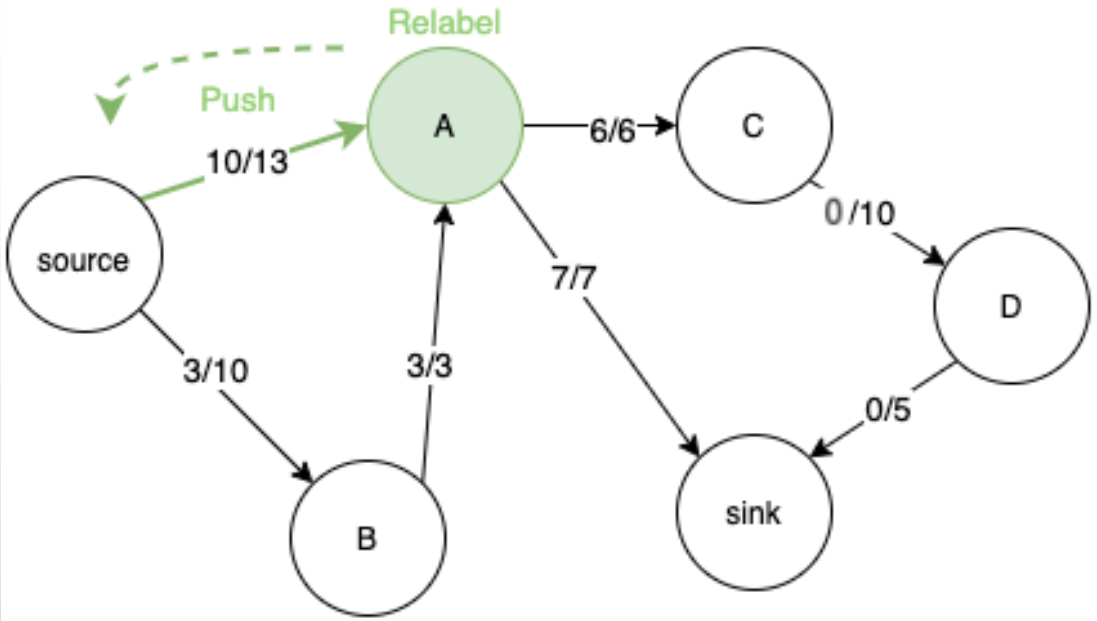
- Bu işlem B'nin yüksekliği source'tan büyük olana kadar devam eder. B şimdi artık akışı kaynak düğüme gönderir.



Node	Height	Excess Flow
source	6	-
A	6	3
B	7	0
C	1	0
D	0	0
sink	0	-

Push Relabel

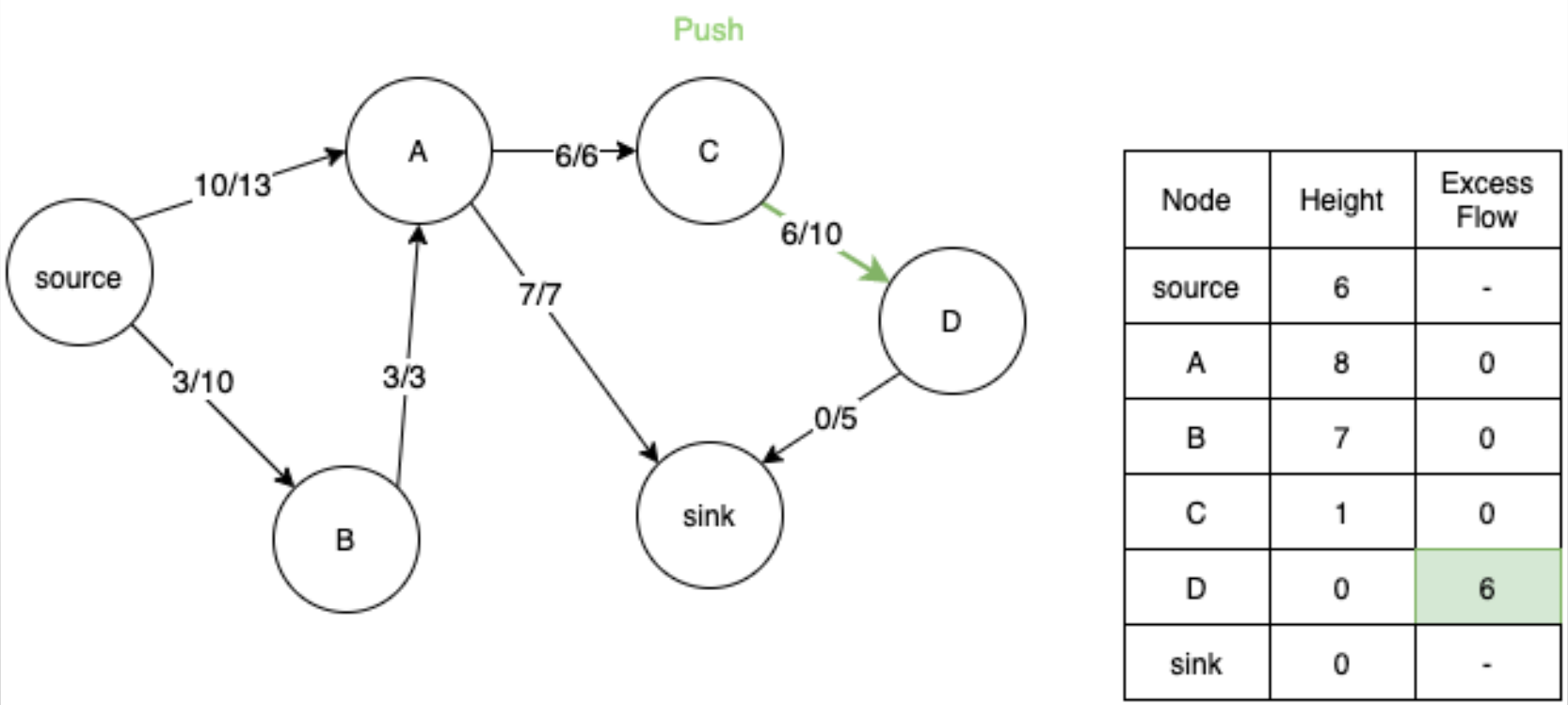
- Aynı şekilde A'nın yüksekliği kaynak düğümden fazla olduğu için artık akışı kaynak düğüme gönderir. *Now both A and B nodes have 0 extra flow*



Node	Height	Excess Flow
source	6	-
A	8	0
B	7	0
C	1	0
D	0	0
sink	0	-

Push Relabel

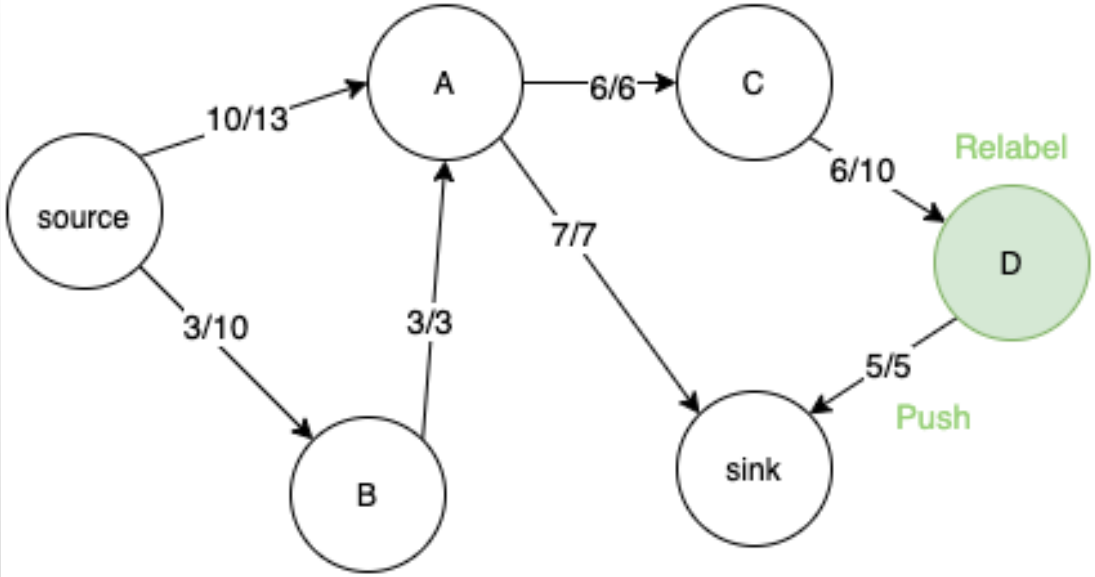
- C ele alınır. Artık akış D'ye gönderilir.



Node	Height	Excess Flow
source	6	-
A	8	0
B	7	0
C	1	0
D	0	6
sink	0	-

Push Relabel

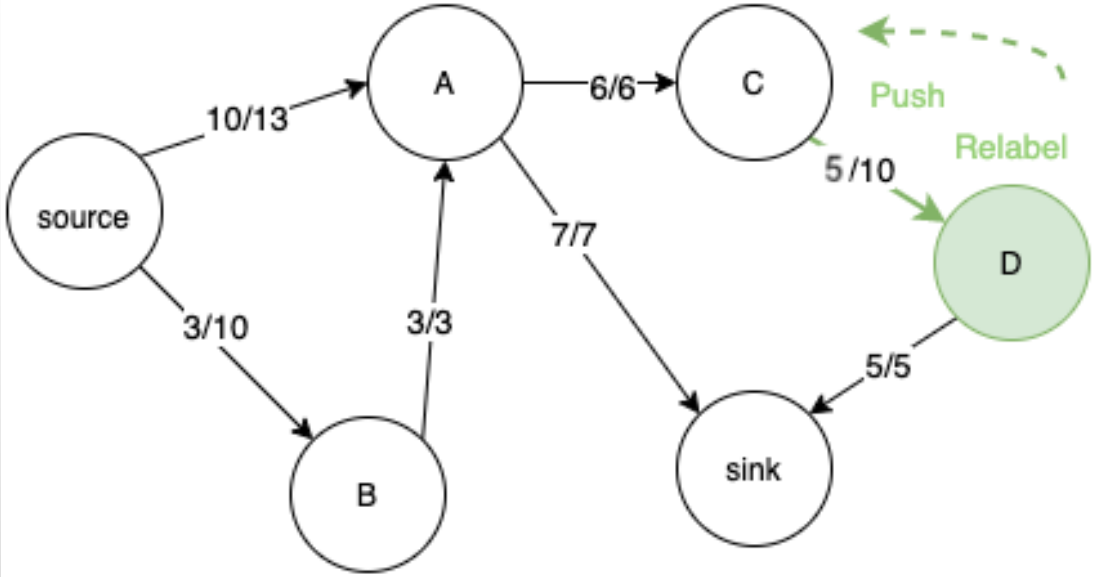
- D relabel yapılır. Artık akış sink'e gönderilir. Hala 1 birim artık akış vardır.



Node	Height	Excess Flow
source	6	-
A	8	0
B	7	0
C	1	0
D	1	1
sink	0	-

Push Relabel

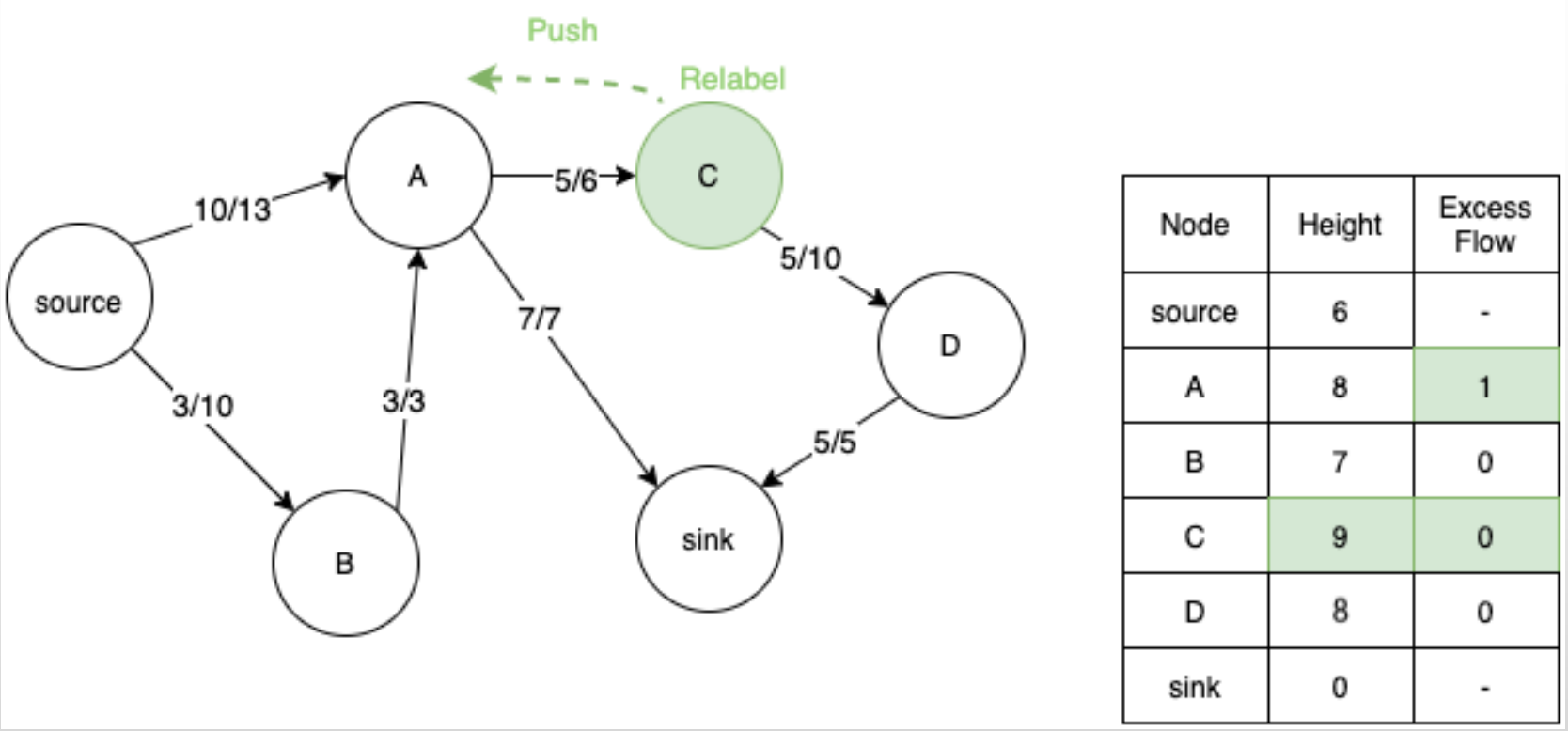
- D relabel yapılır. Artık akış C'ye gönderilir.



Node	Height	Excess Flow
source	6	-
A	8	0
B	7	0
C	1	1
D	2	0
sink	0	-

Push Relabel

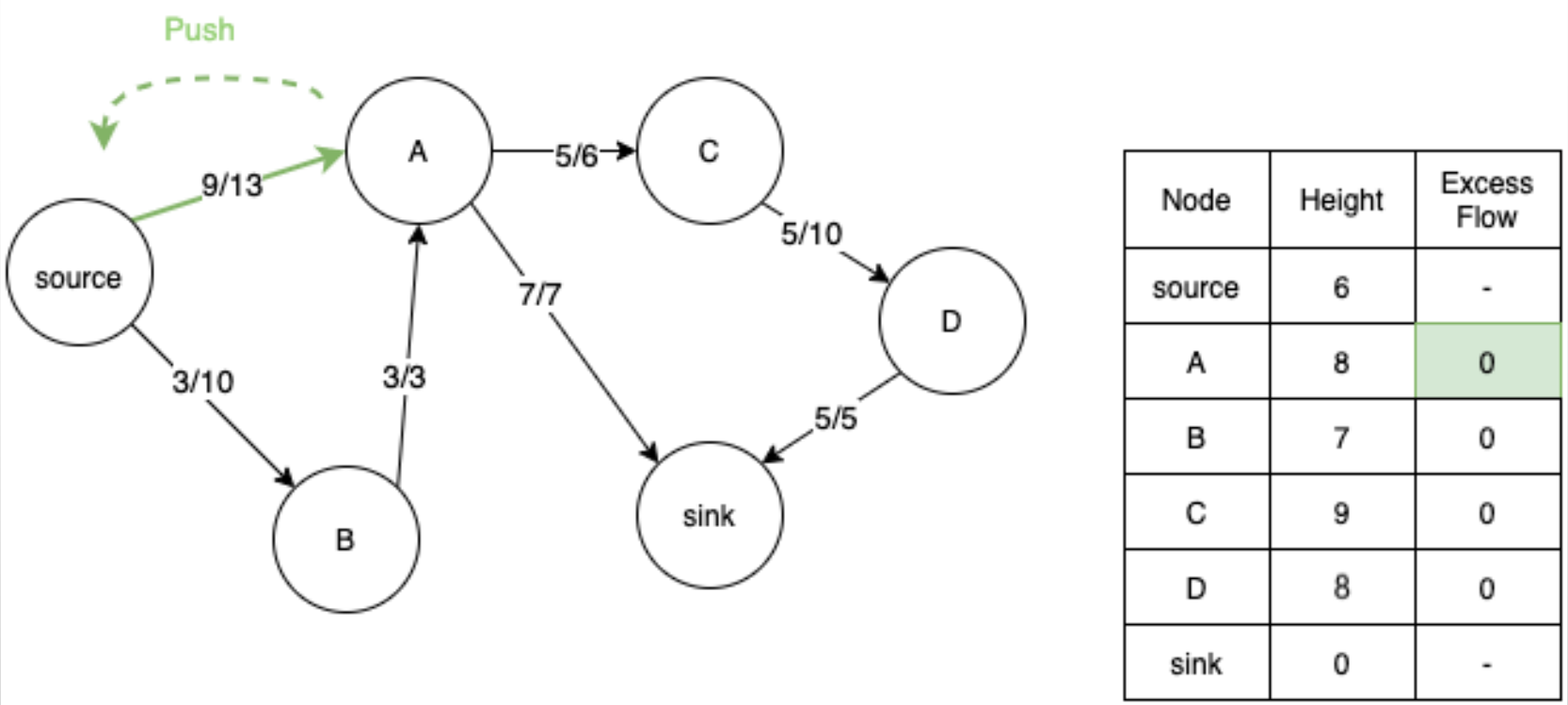
- C'nin yüksekliği A'yı geçene kadar C-D arasında relabel push işlemleri devam eder. C 1 birim ekstra akışı A'ya gönderir.



Node	Height	Excess Flow
source	6	-
A	8	1
B	7	0
C	9	0
D	8	0
sink	0	-

Push Relabel

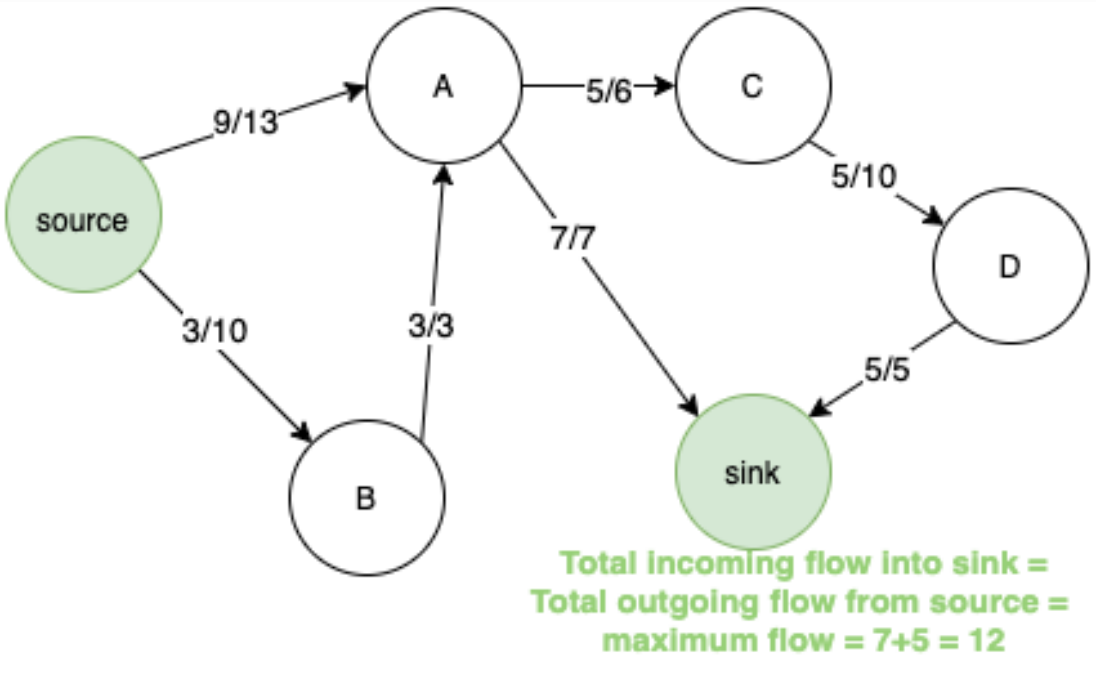
- A 1 birim artık akışı kaynak düğüme gönderir.



Node	Height	Excess Flow
source	6	-
A	8	0
B	7	0
C	9	0
D	8	0
sink	0	-

Push Relabel

- Maksimum akış kaynak düğümden çıkan ya da hedef düğüme giren trafik miktarına eşit olur.



Node	Height	Excess Flow
source	6	-
A	8	0
B	7	0
C	9	0
D	8	0
sink	0	-

Sözde Kod

PUSH_RELABEL(G, kaynak, hedef):

yükseklik[kaynak] = |V|, düğüm sayısı

yükseklik[v] = 0, her v ≠ kaynak için

aşırı[v] = 0, her v için

akış[e] = 0, her kenar e için

aşırı[kaynak] = ∞

Sözde Kod (2)

her bir kenar (kaynak,v) ∈ E için:

akış[kaynak,v] = kapasite[kaynak,v]

akış[v,kaynak] = -kapasite[kaynak,v]

aşırı[v] = kapasite[kaynak,v]

aşırı[kaynak] -= kapasite[kaynak,v]

Sözde Kod (3)

```
döngü aşırı[v] > 0 olan bir v ≠ kaynak bulunduğu sürece:
    eğer kapasite[v,u] - akış[v,u] > 0 ve yükseklik[v] > yükseklik[u]:
        delta = min(aşırı[v], kapasite[v,u] - akış[v,u]) # İtme
        akış[v,u] += delta, akış[u,v] -= delta
        aşırı[v] -= delta, aşırı[u] += delta
    değilse:
        her bir (u, v) için G[u] içinde: # Yeniden etiketleme
            eğer akış[u, v] < kapasite[u, v]:
                minimum = min(minimum, yükseklik[v])
        yükseklik[u] = minimum + 1
```

SON