

Bölüm 4.4: Ağ Akışı

Algoritmalar

Akış ağları

Artık ağ ve artış yolu

Kesimler

Maks-akış min-kesim

Ford-Fulkerson

Edmonds-Karp

Dinic

İtme-yeniden etiketleme

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

Öğrenme Çıktıları

Modelleme

Bir problemi kaynak, hedef ve kapasiteli kenarlardan oluşan bir akış ağı olarak kurmak; akışın kapasite ve korunum kısıtlarını yazmak.

Artık ağ ve kesim

Artık ağı, artış yolunu ve darboğazı hesaplamak; s–t kesiminin kapasitesini bulmak ve maks-akış min-kesim teoremini uygulamak.

Algoritmalar

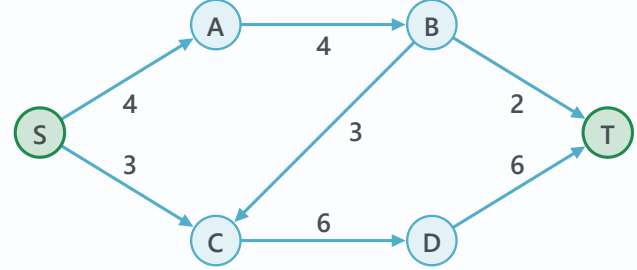
Ford-Fulkerson, Edmonds-Karp, Dinic ve itme–yeniden etiketleme algoritmalarını bir örnek ağ üzerinde adım adım çalıştırmak.

Analiz

$O(E \cdot |f^*|)$, $O(V \cdot E^2)$ ve $O(V^2 \cdot E)$ sınırlarının nereden geldiğini açıklamak; algoritmaları karşılaştırıp uygun olanı seçmek.

Maksimum Akış (Maximum Flow) Problemi

- Yönlü çizgede bir **kaynak** (source, s) ve bir **hedef** (sink, t) düğüm bulunur.
- Her kenara bir **kapasite** (capacity) $c(u, v) \geq 0$ atanır; o kenardan birim zamanda geçebilecek en fazla miktar.
- Amaç: kapasiteleri aşmadan kaynaktan hedefe gönderilebilecek **maksimum akış** miktarını bulmak.



Kenar etiketleri kapasitedir. Bu ağda maksimum akış 7'dir.

Sezgi: boru hattında su, yol ağında trafik, bilgisayar ağında veri paketleri. Her boru (kenar) en fazla kapasitesi kadar taşır; ara kavşaklarda (düğümlerde) madde birikmez.

AKIŞ AĞLARI

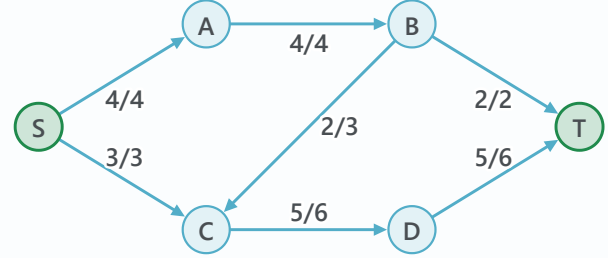
Akış (Flow) ve Kısıtları

Akış, her kenara bir sayı $f(u, v)$ atayan bir fonksiyondur:

1. **Kapasite kısıtı:** $0 \leq f(u, v) \leq c(u, v)$.
2. **Akış korunumu:** s ve t dışındaki her v için giren akış = çıkan akış.

Akışın değeri: $|f| = \sum f(s, v) - \sum f(v, s)$

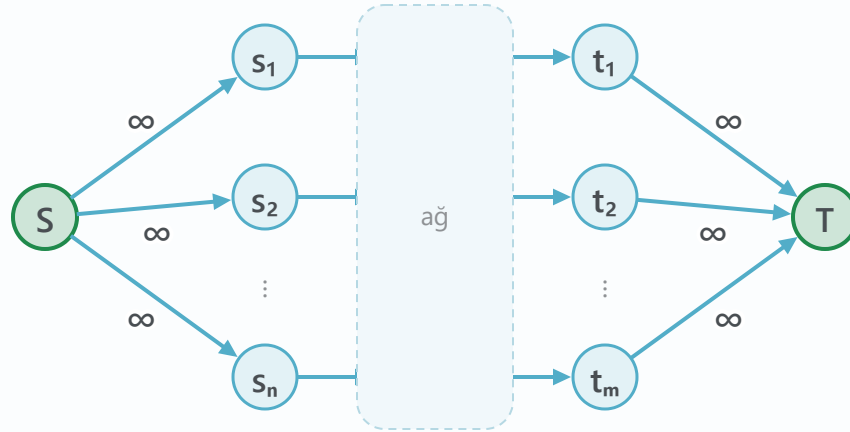
(kaynaktan çıkan net akış = hedefe giren net akış).



Etiket: akış/kapasite. C'ye $3 + 2 = 5$ girer, 5 çıkar. $|f| = 4 + 3 = 7$.

Bir kenarda $f = c$ ise kenar **doymuştur** (saturated): üzerinden daha fazla akış gönderilemez.

Birden Çok Kaynak ve Hedef



- Birden çok kaynak ($s_1 \dots s_n$) ve hedef ($t_1 \dots t_m$) varsa yeni bir **süper kaynak** S ve **süper hedef** T eklenir.
- $S \rightarrow s_i$ ve $t_j \rightarrow T$ kenarlarına **sınırsız (∞) kapasite** verilir; problem tek kaynaklı, tek hedefli maksimum akışa dönüşür.

Artık Ağ (Residual Network)

- Artık ağ G_f , mevcut akış f altında **hâlâ gönderilebilecek** akışı gösterir.
- **İleri kenar:** $c_f(u, v) = c(u, v) - f(u, v)$ — kapasiteden akış çıkarılır; akış kapasiteden azsa kalan kapasite pozitifdir.
- **Ters kenar:** $c_f(v, u) = f(u, v)$ — gönderilmiş akış **geri alınabilir** (iptal edilebilir).
- Yalnızca $c_f > 0$ olan kenarlar G_f 'ye girer; artış yolları bu ağda aranır.

Akış ağı: $f/c = 3/5$



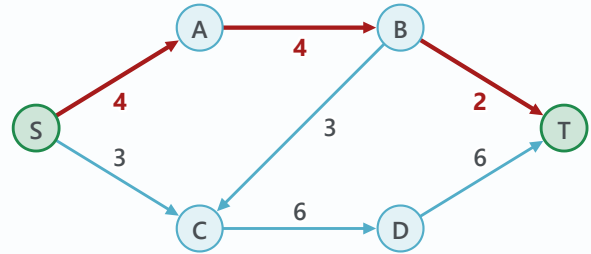
Artık ağ: $5 - 3 = 2$ ileri, 3 geri



Düz ok: ileri (kalan) kapasite · kesikli ok: ters kenar. Ters kenarlar olmadan algoritma ilk yanlış seçimini düzeltemez.

Artış Yolu (Augmenting Path)

- Artık ağda **kaynaktan hedefe giden** yönlü bir yoldur; kaynaktan hedefe **ek akış** sağlar.
- Yoldaki her kenarın artık kapasitesi pozitifdir: kenarlar kapasitelerinden **daha az** akış taşır (ya da ters kenardır).
- **Darboğaz** (bottleneck): $c_f(p) = \min c_f(u, v)$, $(u, v) \in p$. Yol boyunca bu kadar akış eklenir.
- Genellikle **DFS** veya **BFS** ile bulunur; artış yolu kalmayınca akış maksimumdur.

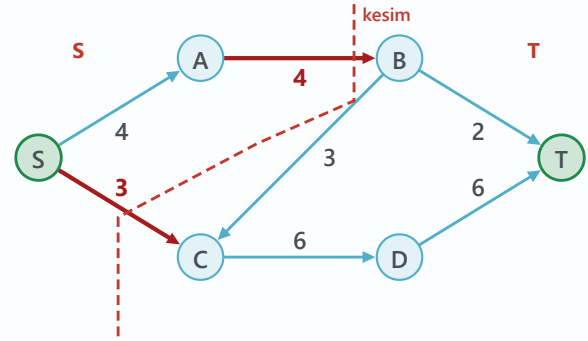


$$p = S \rightarrow A \rightarrow B \rightarrow T: \min(4, 4, 2) = 2.$$

KESİMLER

Kesimler (Cuts)

- **s-t kesimi** (S, T): düğümleri $s \in S$ ve $t \in T$ olacak biçimde ikiye bölen ayrımdır; kesilen kenarlar çizgenin bağlantısını keser.
- **Kesim kapasitesi**: $c(S, T) = \sum c(u, v)$, $u \in S$, $v \in T$ — yalnızca **S'den T'ye** giden kenarlar sayılır.
- Her akış için kesimden geçen net akış $|f|$ 'ye eşittir; bu yüzden $|f| \leq c(S, T)$.
- **Minimum kesim** (min-cut): kapasitesi minimum kesim (kenar *sayısı* değil, kapasite *toplamı*).



$S = \{S, A\}$; $c(S, T) = 4 + 3 = 7$ (minimum).

Maksimum Akış – Minimum Kesim Teoremi

Bir f akışı için aşağıdakiler **denktir** (Ford ve Fulkerson, 1956):

1. f bir maksimum akıştır.
2. Artık ağ G_f 'de hiçbir artış yolu yoktur.
3. Bir (S, T) kesimi için $|f| = c(S, T)$.

Sonuç: maksimum akış değeri = minimum kesim kapasitesi. Min kesim, son artık ağda s 'ten **ulaşılabilen düğümler** S alınarak bulunur.

S tarafı (örnek ağ)	$c(S, T)$
{S}	7
{S, A}	7
{S, A, B}	8
{S, A, B, C, D}	8
{S, C}	10
{S, B, D}	18

16 kesimin en küçüğü 7'dir; bulunan akışın değeri de 7 olduğu için bu akış maksimumdur.

Ağ Akış Algoritmaları (Network Flow)

Amaç: kısıtlar altında kaynaktan hedefe mümkün olan en fazla akışı sağlamak.

Algoritma	Yaklaşım	Süre	Not
Ford-Fulkerson (1956)	herhangi bir artış yolu (çoğunlukla DFS)	$O(E \cdot f^*)$	basit; süre akış değerine bağlı
Edmonds-Karp (1972)	en kısa artış yolu (BFS)	$O(V \cdot E^2)$	Ford-Fulkerson'ın polinom zamanlı hâli
Dinic (1970)	seviye çizgesi + engelleyici akış	$O(V^2 \cdot E)$	Edmonds-Karp'tan hızlı; birim kapasitede çok verimli
İtme-yeniden etiketleme (1988)	ön akış + yükseklikler, yerel işlemler	$O(V^2 \cdot E)$	yol aramaz; FIFO sürümü $O(V^3)$

$|f^*|$: maksimum akışın değeri. Ford-Fulkerson'ın süresi tam sayı kapasiteler için geçerlidir.

Ford-Fulkerson Yöntemi

- L. R. Ford Jr. ve D. R. Fulkerson tarafından geliştirilmiştir (1956).
- Ağırlıklı (kapasiteli) yönlü çizgede iki düğüm arasındaki maksimum akışı bulur: kaynaktan hedefe, belirli kapasiteli borularla gönderilebilecek **en fazla suyu** modelleyen bir çizge problemi.
- **İlkeler:** ağ yönlü bir çizgedir; her kenarın bir kapasitesi vardır; başlangıçta tüm akışlar **0**'dır; artış yolu bulma adımı tekrarlanarak maksimum akışa ulaşılır.
- Artış yolunun *nasıl* bulunacağını belirtmediği için bir **yöntemdir**: Ford-Fulkerson genellikle **DFS**, Edmonds-Karp ise **BFS** kullanır.

Ford-Fulkerson: Algoritma Adımları

0. Tüm kenarlarda $f(u, v) = 0$; artık ağ = özgün ağ.

Başlangıçta akış sıfırdır, her kenarın artık kapasitesi kendi kapasitesidir.

Adım 1 / 5

Ford-Fulkerson: Algoritma Adımları

0. Tüm kenarlarda $f(u, v) = 0$; artık ağ = özgün ağ.

Başlangıçta akış sıfırdır, her kenarın artık kapasitesi kendi kapasitesidir.

1. Artık ağda kaynaktan hedefe bir **artış yolu** p bul ($c_f > 0$ olan kenarlar).

Yol DFS ile (Ford-Fulkerson) ya da BFS ile (Edmonds-Karp) aranır.

Ford-Fulkerson: Algoritma Adımları

0. Tüm kenarlarda $f(u, v) = 0$; artık ağ = özgün ağ.

Başlangıçta akış sıfırdır, her kenarın artık kapasitesi kendi kapasitesidir.

1. Artık ağda kaynaktan hedefe bir **artış yolu** p bul ($c_f > 0$ olan kenarlar).

Yol DFS ile (Ford-Fulkerson) ya da BFS ile (Edmonds-Karp) aranır.

2. Darboğazı hesapla: $\delta = \min c_f(u, v), (u, v) \in p$.

Yol boyunca gönderilebilecek en fazla akış, yolun en dar kenarıdır.

Ford-Fulkerson: Algoritma Adımları

0. Tüm kenarlarda $f(u, v) = 0$; artık ağ = özgün ağ.

Başlangıçta akış sıfırdır, her kenarın artık kapasitesi kendi kapasitesidir.

1. Artık ağda kaynaktan hedefe bir **artış yolu** p bul ($c_f > 0$ olan kenarlar).

Yol DFS ile (Ford-Fulkerson) ya da BFS ile (Edmonds-Karp) aranır.

2. Darboğazı hesapla: $\delta = \min c_f(u, v), (u, v) \in p$.

Yol boyunca gönderilebilecek en fazla akış, yolun en dar kenarıdır.

3. Yol boyunca δ kadar akış gönder: $f(u, v) += \delta$, ters kenar için $f(v, u) -= \delta$; $|f| += \delta$.

Ters kenarlar güncellenir; yoldaki en az bir kenar doyar.

Ford-Fulkerson: Algoritma Adımları

0. Tüm kenarlarda $f(u, v) = 0$; artık ağ = özgün ağ.

Başlangıçta akış sıfırdır, her kenarın artık kapasitesi kendi kapasitesidir.

1. Artık ağda kaynaktan hedefe bir **artış yolu** p bul ($c_f > 0$ olan kenarlar).

Yol DFS ile (Ford-Fulkerson) ya da BFS ile (Edmonds-Karp) aranır.

2. Darboğazı hesapla: $\delta = \min c_f(u, v), (u, v) \in p$.

Yol boyunca gönderilebilecek en fazla akış, yolun en dar kenarıdır.

3. Yol boyunca δ kadar akış gönder: $f(u, v) += \delta$, ters kenar için $f(v, u) -= \delta$; $|f| += \delta$.

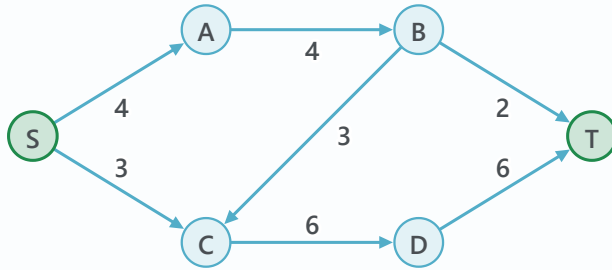
Ters kenarlar güncellenir; yoldaki en az bir kenar doyar.

4. Artış yolu **kalmayana** kadar 1–3'ü tekrarla; $|f|$ maksimumdur.

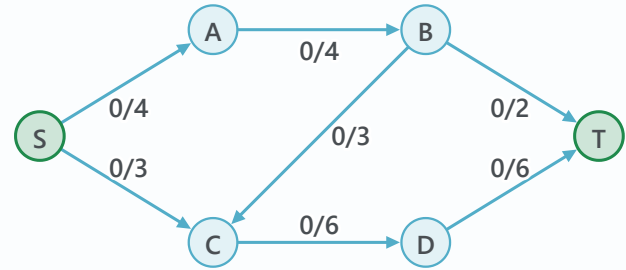
Döngü, hedefe ulaşamadığında (artış yolu kalmadığında) biter; maks-akış min-kesim teoremi sonucun en iyi olduğunu garanti eder.

Ford-Fulkerson Örneği 1

Artık ağ G_f



Akış $f(f/c)$



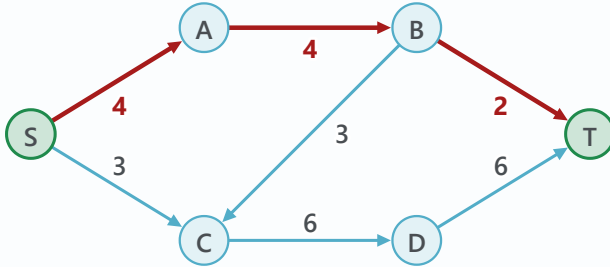
$|f| = 0$

Başlangıç: tüm akışlar 0, artık ağ özgün ağın aynısı.

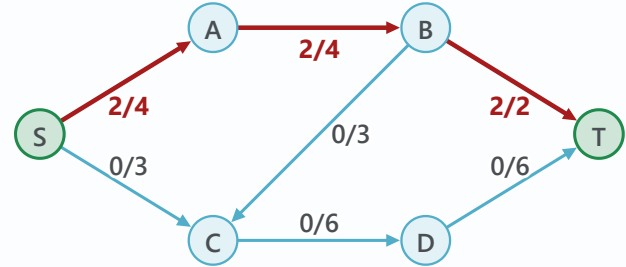
Adım 1 / 5

Ford-Fulkerson Örneği 1

Artık ağ: yol p



Akış: +2



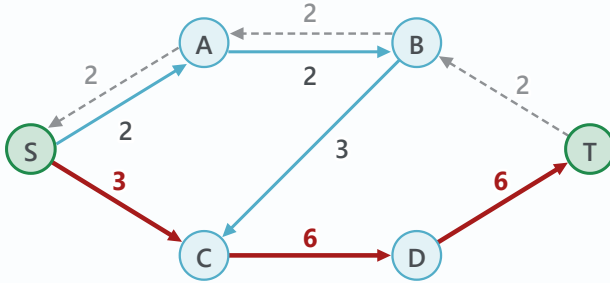
$|f| = 2$

İlk artış yolu $p = \{S, A, B, T\}$: darboğaz **2** birim. $|f| = 2$.

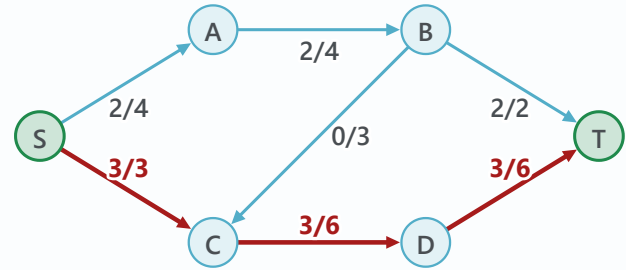
Adım 2 / 5

Ford-Fulkerson Örneği 1

Artık ağ: yol p



Akış: +3



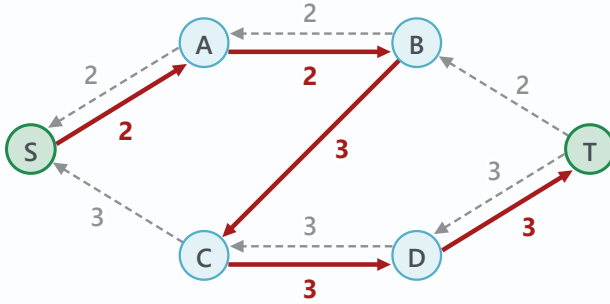
$|f| = 5$

İkinci artış yolu $p = \{S, C, D, T\}$: darboğaz **3** birim. $|f| = 5$.

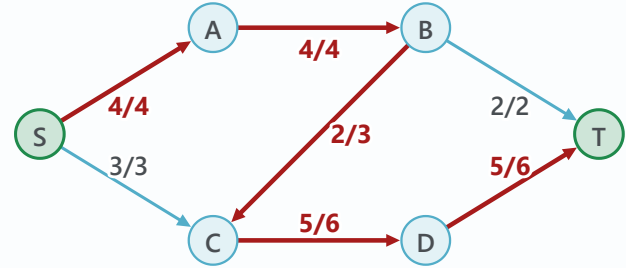
Adım 3 / 5

Ford-Fulkerson Örneği 1

Artık ağ: yol p



Akış: +2



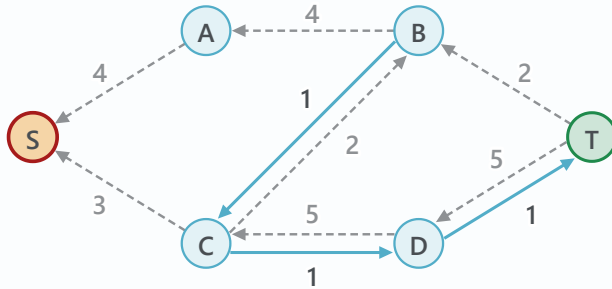
$|f| = 7$

Üçüncü artış yolu $p = \{S, A, B, C, D, T\}$: darboğaz **2** birim. $|f| = 7$.

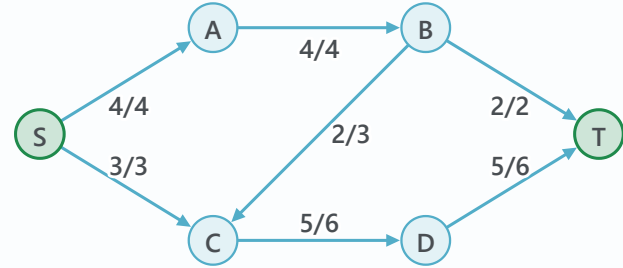
Adım 4 / 5

Ford-Fulkerson Örneği 1

Son artık ağ



Maksimum akış



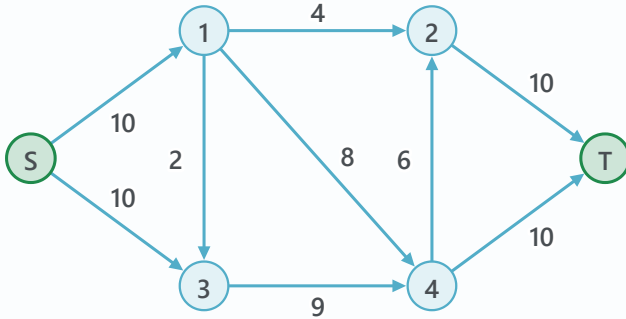
$|f| = 7$

S'den çıkan iki kenar da doydular: artık ağda S'den hiçbir yere gidilemez. **Maksimum akış 7;**
min kesim ($\{S\}$, diğerleri) = 4 + 3.

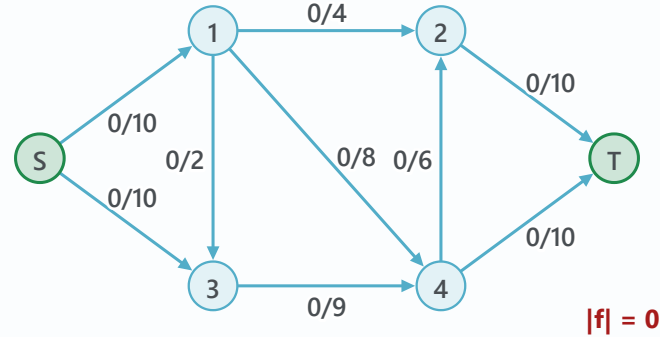
FORD-FULKERSON

Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Artık ağ G_f



Akış $f(f/c)$



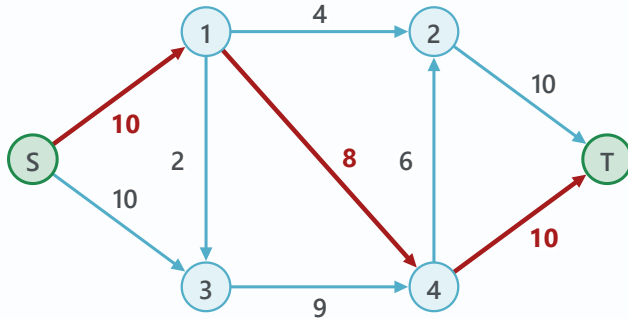
$|f| = 0$

Kaynak S, hedef T. Başlangıçta tüm akışlar 0.

Adım 1 / 7

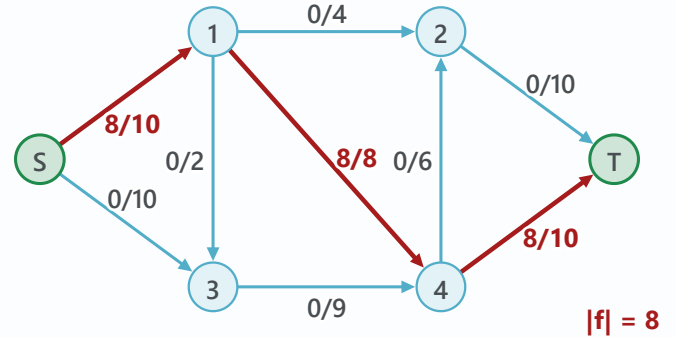
Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Artık ağ: yol p (+8)



$p = S \rightarrow 1 \rightarrow 4 \rightarrow T$: darboğaz $1 \rightarrow 4$ (8). $|f| = 8$.

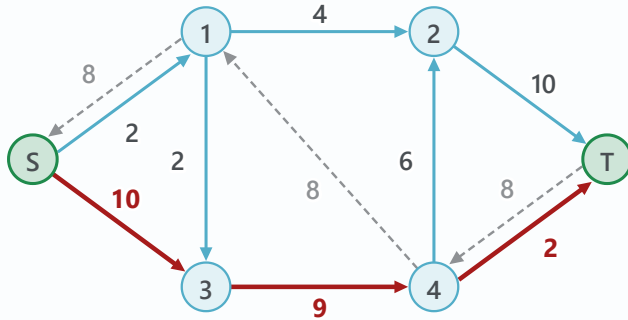
Akış f



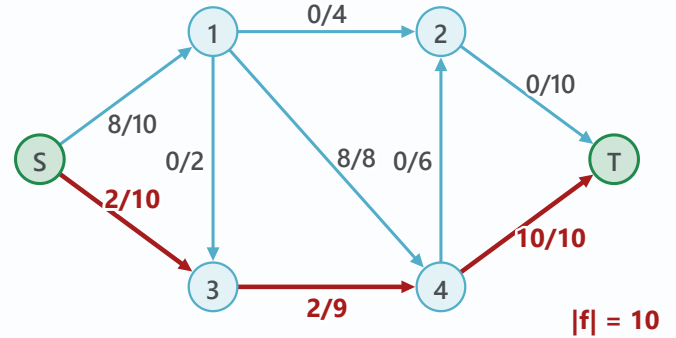
Adım 2 / 7

Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Artık ağ: yol p (+2)



Akış f

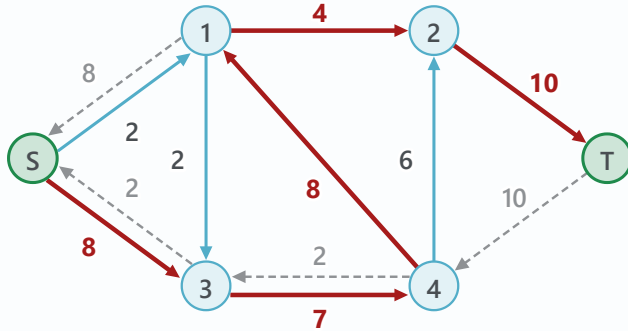


$p = S \rightarrow 3 \rightarrow 4 \rightarrow T$: 4 $\rightarrow T$ 'de yalnızca 2 kaldı. $|f| = 10$.

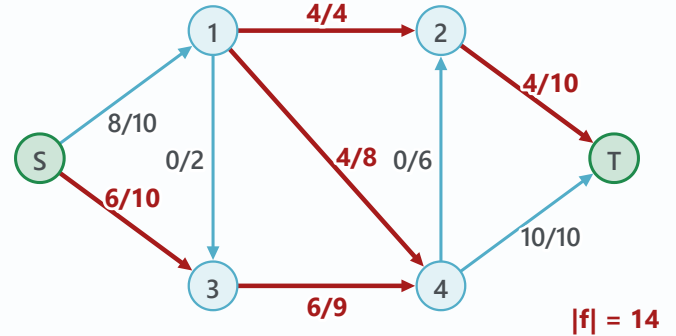
Adım 3 / 7

Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Artık ağ: yol p (+4)



Akış f

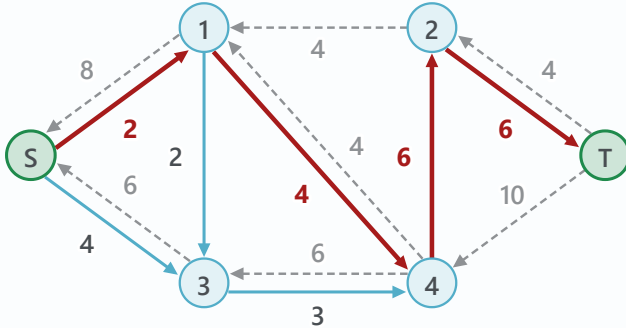


$p = S \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow T$: $\mathbf{4 \rightarrow 1}$ **ters kenarı** kullanılır; $1 \rightarrow 4$ akışı 8'den 4'e düşer, 1'den çıkan 4 birim $1 \rightarrow 2$ 'ye yönlendirir. $|f| = 14$.

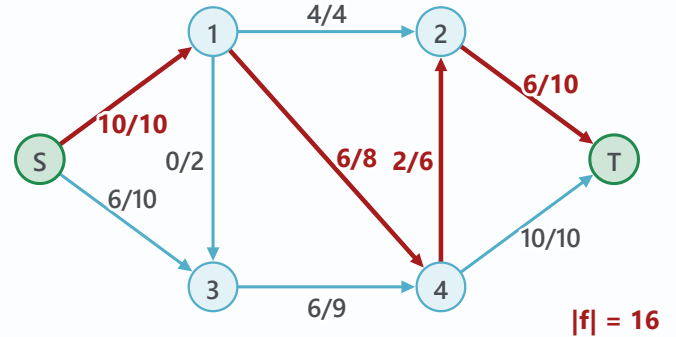
FORD-FULKERSON

Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Artık ağ: yol p (+2)



Akış f

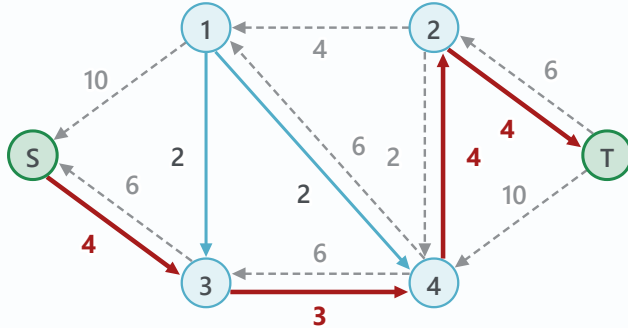


$p = S \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow T$: $S \rightarrow 1$ 'de 2 birim kalmıştı. $|f| = 16$.

Adım 5 / 7

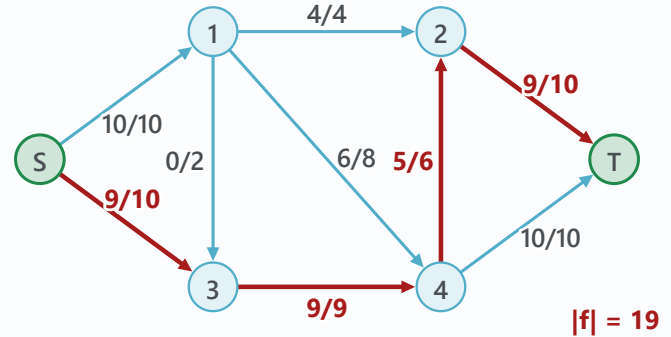
Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Artık ağ: yol p (+3)



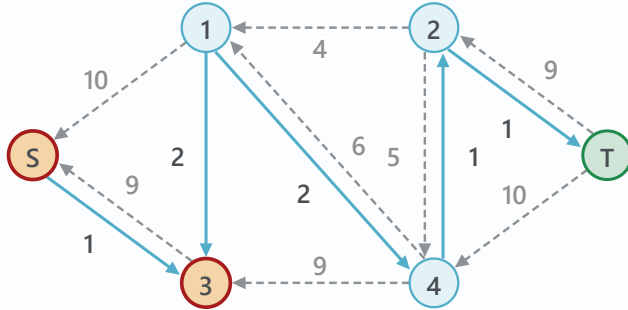
$p = S \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow T$: darboğaz $3 \rightarrow 4$ ($9 - 6 = 3$ kaldı). $|f| = 19$.

Akış f

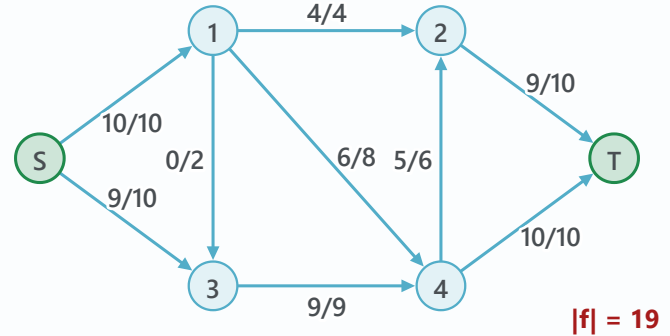


Ford-Fulkerson Örneği 2: Ters Kenar Kullanımı

Son artık ağ



Maksimum akış



Artık ağda S'den yalnızca 3'e ulaşılır $\rightarrow$ min kesim ($\{S, 3\}$, diğerleri) = $10 + 9 = 19 = |f^*|$.

Ters Kenarlar Neden Gerekli?

Yalnızca ileri kenarlarla (açgözlü)

1. $S \rightarrow 1 \rightarrow 4 \rightarrow T$: 8
2. $S \rightarrow 3 \rightarrow 4 \rightarrow T$: 2
3. $S \rightarrow 1 \rightarrow 2 \rightarrow T$: 2
4. $S \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow T$: 6

Toplam **18**. S'den yalnızca 3 ve 4'e gidilebiliyor; 4'ün çıkışları dolu. Takıldı!

Artık ağ (ters kenarlı)

İlk seçim $1 \rightarrow 4$ 'e 8 birim gönderdi. $4 \rightarrow 1$ ters kenarı bu kararın bir kısmını **geri almayı** sağlar: 1'den gelen akış 2'ye, 3'ten gelen akış 4'e yönlenir.

Sonuç **19** = min kesim. Yol hangi sırayla seçilirse seçilsin artık ağ maksimuma ulaşır.

Ters kenar, gerçekte "geriye akış göndermek" değil, daha önce gönderilmiş akışı **iptal etmek** demektir: $f(u, v) - = \delta$.

Ford-Fulkerson: Sözde Kod

```
FORD_FULKERSON(G, kaynak, hedef):
    akış[u, v] = 0, her bir (u, v) için E içinde           // başlangıçta 0
    maksimumAkış = 0
    döngü artırıcı yol var iken:
        yol = DFS(G, kaynak, hedef, akış)                // c - akış > 0 kenarlarla
        delta = ∞                                           // yoldaki en küçük kalan kapasite
        v = hedef
        döngü v ≠ kaynak iken:
            u = yol.ata[v]
            delta = min(delta, kapasite[u, v] - akış[u, v])
            v = u
        v = hedef                                           // yoldaki akışları güncelle
        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
```

Java: DFS ile Artış Yolu

```
boolean dfs(int u, int hedef,
            boolean[] ziyaret, int[] ata) {
    ziyaret[u] = true;
    if (u == hedef) return true;
    for (int v = 0; v < V; v++) {
        if (!ziyaret[v] && artik[u][v] > 0) {
            ata[v] = u;
            if (dfs(v, hedef, ziyaret, ata))
                return true;
        }
    }
    return false; // bu daldan hedefe yol yok
}
```

- `artik[u][v]` : artık kapasite matrisi. Başlangıçta kapasiteler; ters kenarlar 0.
- Hedefe ulaşılnca `true` döner; `ata[]` yolu tersten verir.
- Akış eklenirken `artik[u][v] -= d` ve `artik[v][u] += d` yapılır: ters kenar kendiliğinden oluşur.
- Komşu matrisinde bir DFS $O(V^2)$, komşu listesinde $O(V + E)$ sürer.

Ford-Fulkerson: Karmaşıklık Analizi

- Çalışma süresi ağın yapısına ve **kapasitelere** bağlıdır.
- Bir artış yolu bulmak (DFS/BFS) ve akışı güncellemek: $O(E)$.
- Kapasiteler tam sayı ise her artış yolu akışı **en az 1 birim** artırır $\rightarrow$ en fazla $|f^*|$ tekrar.
- Toplam: $O(E \cdot |f^*|)$; E kenar sayısı, $|f^*|$ maksimum akış değeri.

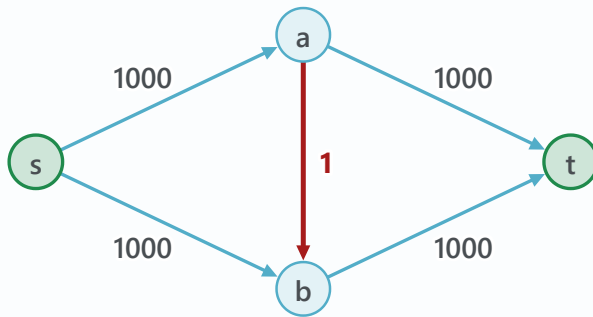
Tam sayıllık (integrality)

Tüm kapasiteler tam sayı ise akış hep tam sayı kalır ve tam sayı değerli bir maksimum akış bulunur. Eşleştirme problemlerinde bu özellik kullanılır.

Uyarı

$O(E \cdot |f^*|)$ **polinom değildir**: $|f^*|$ sayının büyüklüğüne bağlıdır. İrrasyonel kapasitelerde yöntem sonlanmayabilir bile.

Kötü Yol Seçimi: 2000 Artış



- Maksimum akış 2000 ($s \rightarrow a \rightarrow t$ ve $s \rightarrow b \rightarrow t$).
- Kötü bir yol seçimi her seferinde ortadaki kenarı kullanır: $s \rightarrow a \rightarrow b \rightarrow t$, sonra ters kenarla $s \rightarrow b \rightarrow a \rightarrow t$... Her yol yalnızca **1 birim** ekler → **2000 artış**.
- Edmonds-Karp (BFS) en kısa yolları seçer: **2 artış** yeter.
- Kapasite 10^9 olsaydı DFS'li sürüm milyarlarca adım atabilirdi; BFS bu bağımlılığı kaldırır.

Edmonds-Karp Algoritması

- Ağırlıklı yönlü çizgede iki düğüm arasındaki maksimum akışı bulur; **Ford-Fulkerson yönteminin bir türevidir.**
- Jack Edmonds ve Richard Karp tarafından geliştirilmiştir (1972; Dinic 1970'te bağımsız olarak).
- Artış yollarını **BFS** (genişlik öncelikli arama) ile bulur: her seferinde **kenar sayısı en az** olan artış yolu seçilir.
- Edmonds-Karp BFS'yi artış yolu bulmak için kullanır; Dinic ise BFS'yi daha fazla akış mümkün mü diye bakmak ve **seviye çizgesi** kurmak için kullanır.

Adımlar: (1) BFS ile kaynaktan hedefe en kısa artış yolunu bul. (2) Yol boyunca darboğaz kadar akış gönder, ters kenarları güncelle. (3) BFS hedefe ulaşamayana kadar tekrarla.

Edmonds-Karp: Karmaşıklık $O(V \cdot E^2)$

- Her tur bir BFS ve bir yol güncellemesidir: $O(E)$.
- BFS en kısa yolu seçtiği için s'ten her düğüme olan artık ağ uzaklığı **hiç azalmaz**.
- Her artışta en az bir kenar **kritik** olur (doyar). Aynı kenarın yeniden kritik olabilmesi için uzaklığı en az 2 artmalıdır; uzaklık $\leq V$ olduğundan her kenar en fazla $\sim V/2$ kez kritik olur.
- Artış sayısı $O(V \cdot E) \rightarrow$ toplam $O(V \cdot E) \cdot O(E) = O(V \cdot E^2)$; E kenar, V düğüm sayısı.

Süre artık $|f^*|$ 'ye bağlı değildir: kapasiteler ne kadar büyük olursa olsun algoritma polinom zamanda biter.

Java: BFS ile Artış Yolu

```
boolean bfs(int kaynak, int hedef, int[] ata) {
    boolean[] ziyaret = new boolean[V];
    Queue<Integer> kuyruk = new LinkedList<>();
    kuyruk.add(kaynak);
    ziyaret[kaynak] = true;
    ata[kaynak] = -1;
    while (!kuyruk.isEmpty()) {
        int u = kuyruk.poll();
        for (int v = 0; v < V; v++)
            if (!ziyaret[v] && artik[u][v] > 0) {
                kuyruk.add(v);
                ata[v] = u;
                ziyaret[v] = true;
            }
    }
    return ziyaret[hedef];
}
```

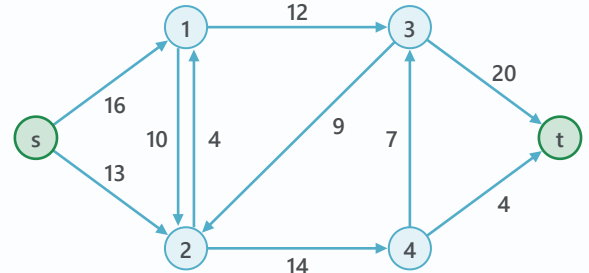
- Kuyruk kaynaktan başlar; artık kapasitesi pozitif ve ziyaret edilmemiş her komşu kuyruğa eklenir.
- `ata[v] = u` ile BFS ağacı saklanır; hedef ziyaret edildiyse bulunan yol **en az kenarlı** yoldur.
- Ford-Fulkerson'dan tek farkı bu fonksiyondur; ana döngü aynıdır.

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2     int maksAkis = 0;
3     int[] ata = new int[V];
4     while (bfs(s, t, ata)) {
5         int akis = Integer.MAX_VALUE;
6         for (int v = t; v != s; v = ata[v])
7             akis = Math.min(akis, artik[ata[v]][v]);
8         for (int v = t; v != s; v = ata[v]) {
9             int u = ata[v];
10            artik[u][v] -= akis; // ileri
11            artik[v][u] += akis; // ters
12        }
13        maksAkis += akis;
14    }
15    return maksAkis;
16 }

```



Başlangıç: artık ağ = kapasiteler, **maksAkis = 0**.

Adım 1 / 14

DEĞİŞKENLER

maksAkis = 0

yol = -

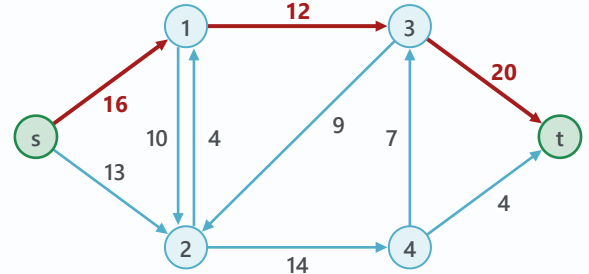
akis = -

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   ► while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



bfs en kısa yolu buldu: $s \rightarrow 1 \rightarrow 3 \rightarrow t$ (3 kenar).

Adım 2 / 14

DEĞİŞKENLER

maksAkis = 0

yol = s13t

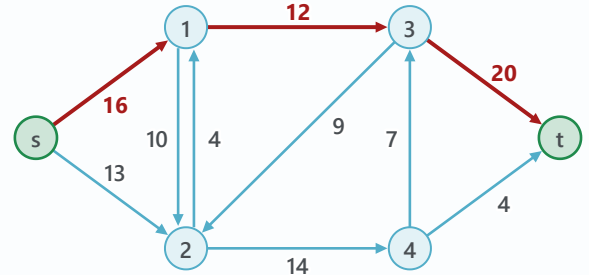
akis = -

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     ▶ for (int v = t; v != s; v = ata[v])
7     ▶   akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



Hedefte geriye: $\text{akis} = \min(16, 12, 20) = 12$.

Adım 3 / 14

DEĞİŞKENLER

`maksAkis = 0`

`yol = s13t`

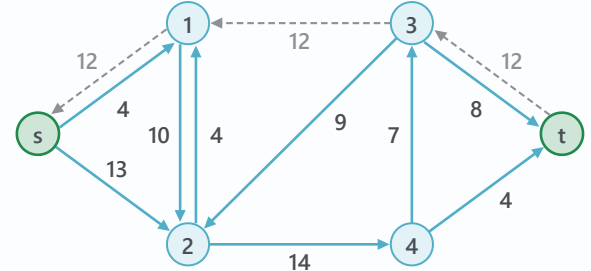
`akis = 12`

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



Yoldaki ileri kenarlardan 12 çıkar, ters kenarlara 12 eklenir (kesikli oklar).

Adım 4 / 14

DEĞİŞKENLER

maksAkis = 0

yol = s13t

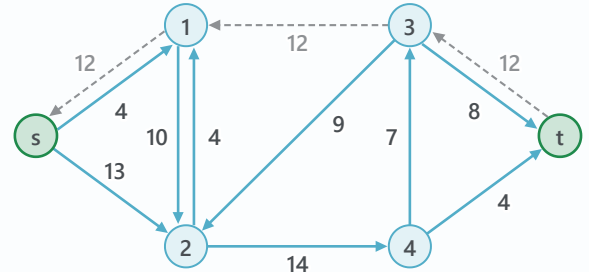
akis = 12

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



maksAkis = 0 + 12 = **12**.

Adım 5 / 14

DEĞİŞKENLER

maksAkis = 12 yol = **s13t**

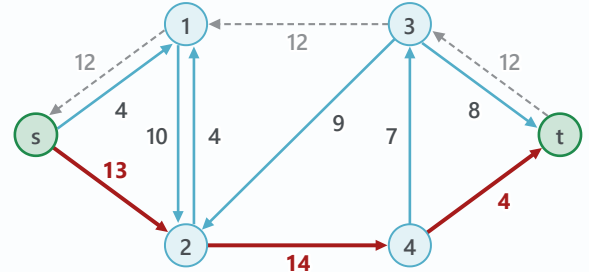
akis = 12

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   ► while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



bfs en kısa yolu buldu: $s \rightarrow 2 \rightarrow 4 \rightarrow t$ (3 kenar).

Adım 6 / 14

DEĞİŞKENLER

maksAkis = 12

yol = s24t

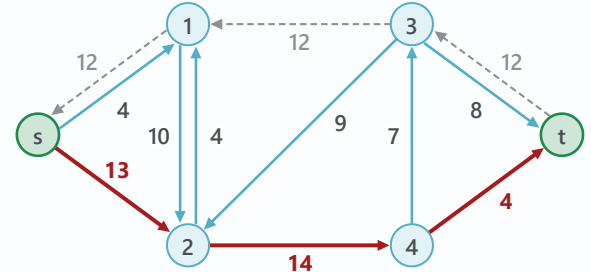
akis = -

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



Hedefte geriye: $\text{akis} = \min(13, 14, 4) = 4$.

Adım 7 / 14

DEĞİŞKENLER

maksAkis = 12

yol = s24t

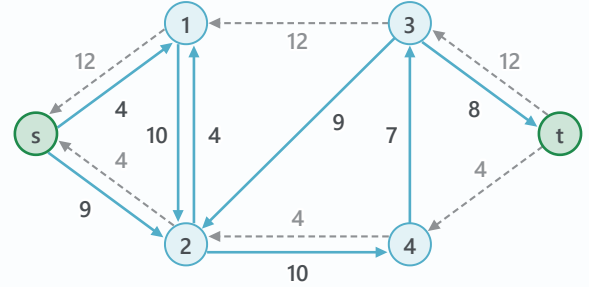
akis = 4

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



Yoldaki ileri kenarlardan 4 çıkar, ters kenarlara 4 eklenir (kesikli oklar).

Adım 8 / 14

DEĞİŞKENLER

maksAkis = 12

yol = s24t

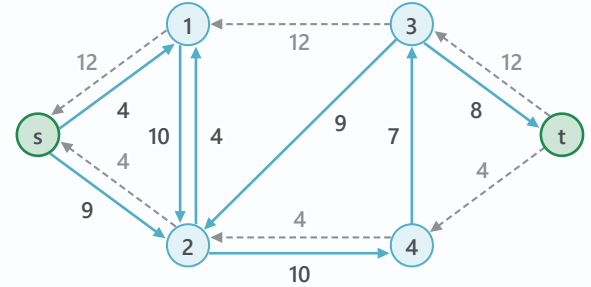
akis = 4

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



maksAkis = 12 + 4 = **16**.

Adım 9 / 14

DEĞİŞKENLER

maksAkis = 16 yol = **s24t**

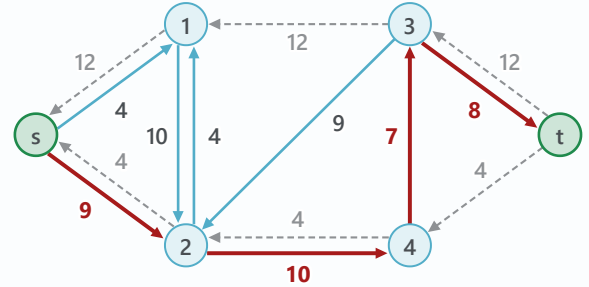
akis = 4

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   ▶ while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



bfs en kısa yolu buldu: $s \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow t$ (4 kenar).

Adım 10 / 14

DEĞİŞKENLER

maksAkis = 16

yol = s243t

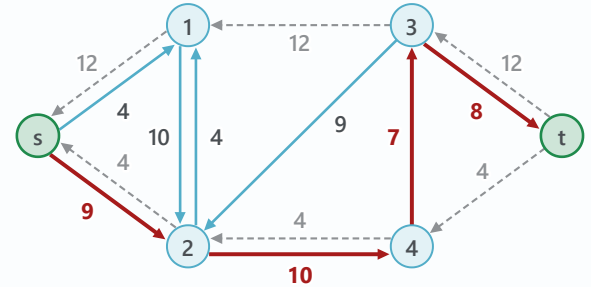
akis = -

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     ▶ for (int v = t; v != s; v = ata[v])
7       ▶ akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



Hedefte geriye: $akis = \min(9, 10, 7, 8) = 7$.

Adım 11 / 14

DEĞİŞKENLER

maksAkis = 16

yol = s243t

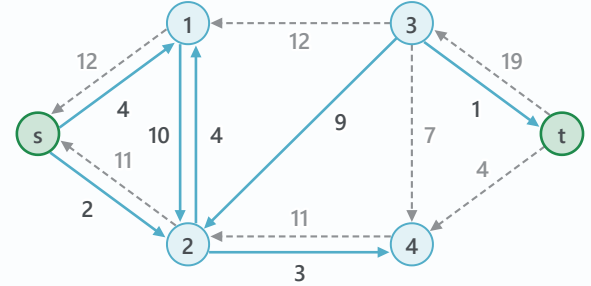
akis = 7

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



Yoldaki ileri kenarlardan 7 çıkar, ters kenarlara 7 eklenir (kesikli oklar).

Adım 12 / 14

DEĞİŞKENLER

maksAkis = 16

yol = s243t

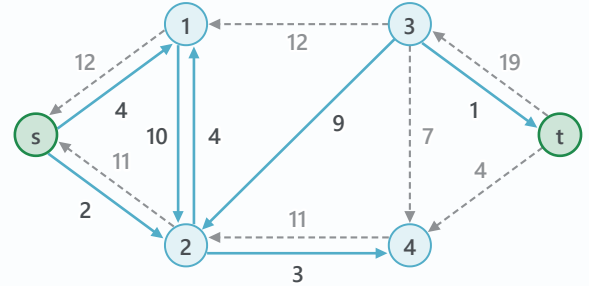
akis = 7

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  return maksAkis;
16 }

```



maksAkis = 16 + 7 = **23**.

Adım 13 / 14

DEĞİŞKENLER

maksAkis = 23

yol = **s243t**

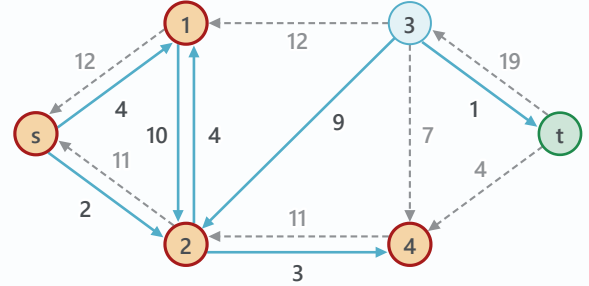
akis = 7

edmondsKarp(s, t): Kodu Adım Adım İzle

```

1 int edmondsKarp(int s, int t) {
2   int maksAkis = 0;
3   int[] ata = new int[V];
4   ▶ while (bfs(s, t, ata)) {
5     int akis = Integer.MAX_VALUE;
6     for (int v = t; v != s; v = ata[v])
7       akis = Math.min(akis, artik[ata[v]][v]);
8     for (int v = t; v != s; v = ata[v]) {
9       int u = ata[v];
10      artik[u][v] -= akis; // ileri
11      artik[v][u] += akis; // ters
12    }
13    maksAkis += akis;
14  }
15  ▶ return maksAkis;
16 }

```



BFS t'ye ulaşamadı (s'ten yalnızca s, 1, 2, 4 ulaşılabilir). **return 23**.

Adım 14 / 14

DEĞİŞKENLER

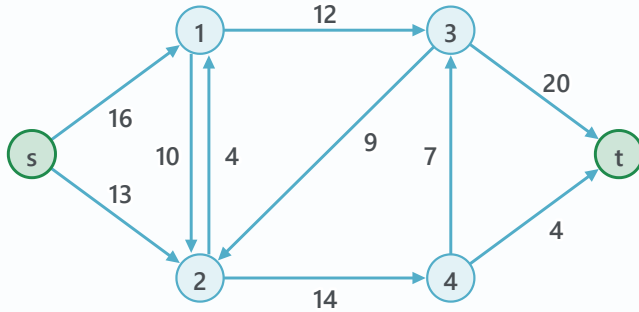
maksAkis = 23

yol = yok

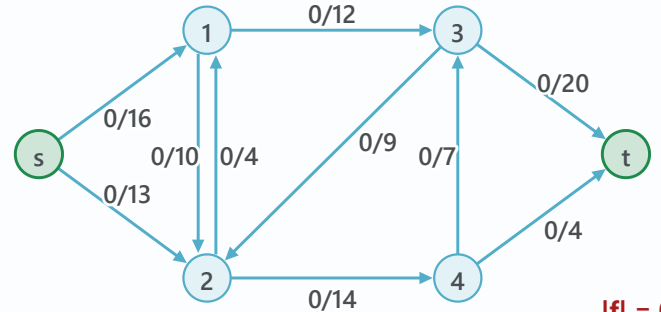
akis = 7

Edmonds-Karp Örneği

Artık ağ



Akış f

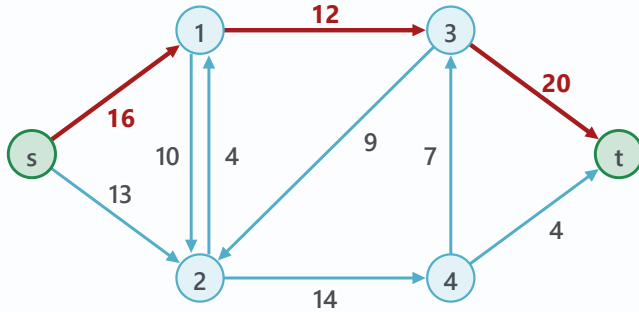


$|f| = 0$

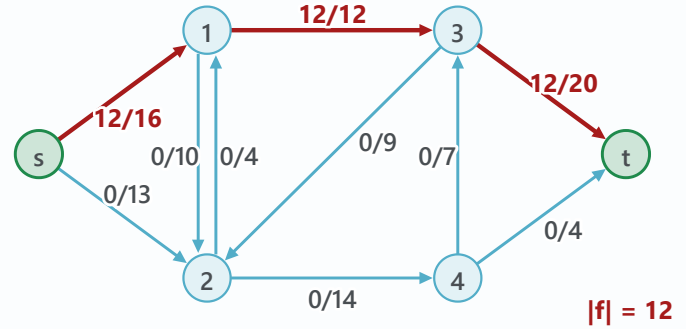
Ağ verilsin. Üst sınır: $|f^*| \leq \mathbf{24}$ (s'den çıkan kapasite $16 + 13 = 29$, t'ye giren $20 + 4 = 24$; küçüğü).

Edmonds-Karp Örneği

Artık ağ: p_1



Akış f

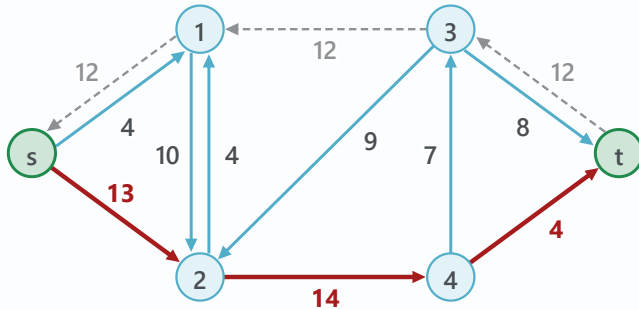


Adım 1: $c(1, 3)$ yüzünden $c_f(p_1) = \mathbf{12}$ olan $p_1 = \langle s, 1, 3, t \rangle$ artış yolu seçilir. $|f| = 12$.

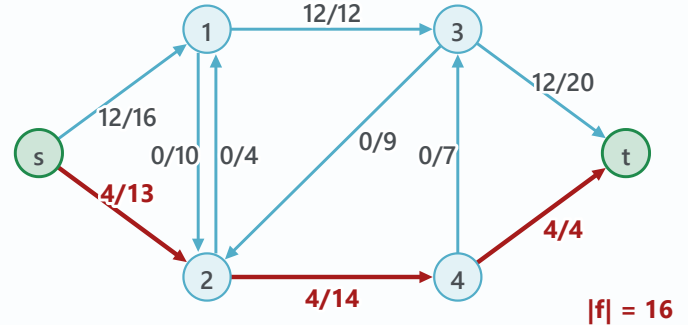
Adım 2 / 5

Edmonds-Karp Örneği

Artık ağ: p_2



Akış f

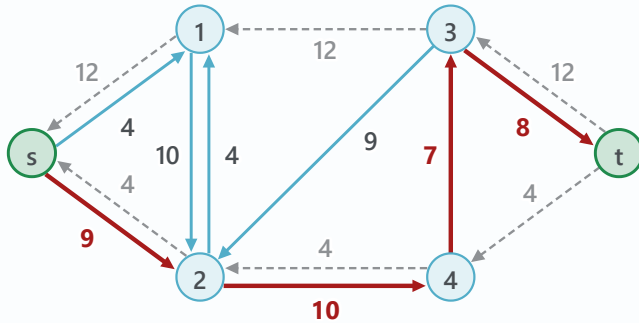


Adım 2: $c(4, t)$ yüzünden $c_f(p_2) = 4$ olan $p_2 = \langle s, 2, 4, t \rangle$ artış yolu seçilir. $|f| = 16$.

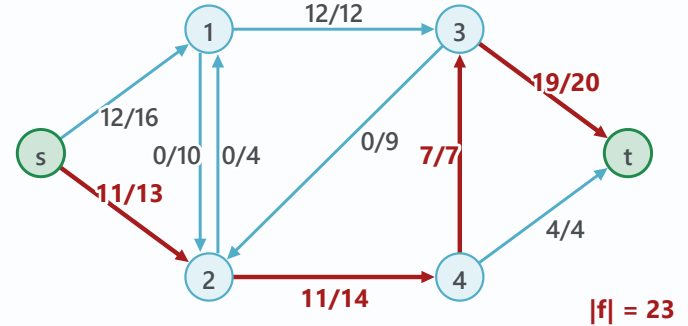
Adım 3 / 5

Edmonds-Karp Örneği

Artık ağ: p_3



Akış f

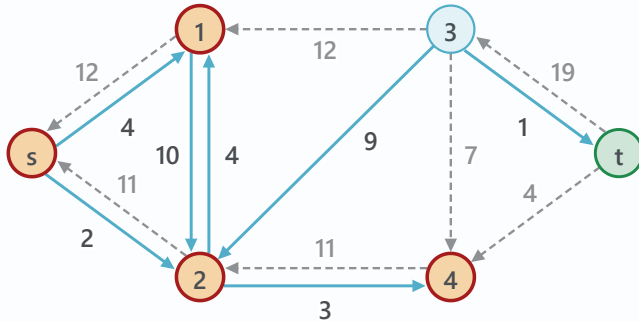


Adım 3: $c(4, 3)$ yüzünden $c_f(p_3) = 7$ olan $p_3 = \langle s, 2, 4, 3, t \rangle$ artış yolu seçilir. $|f| = 23$.

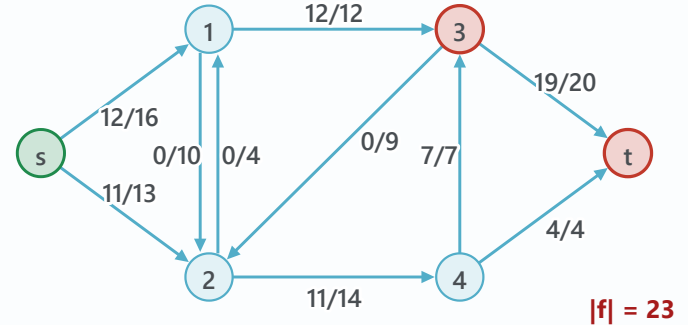
Adım 4 / 5

Edmonds-Karp Örneği

Son artık ağ



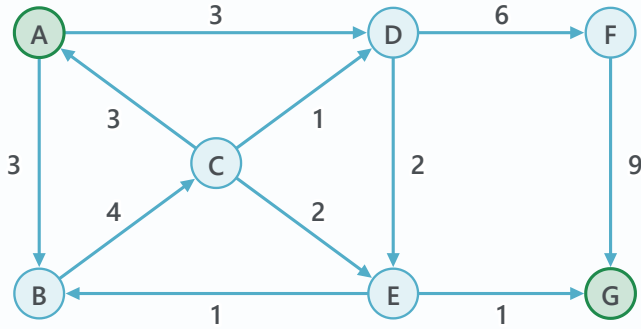
Son akış ve min kesim



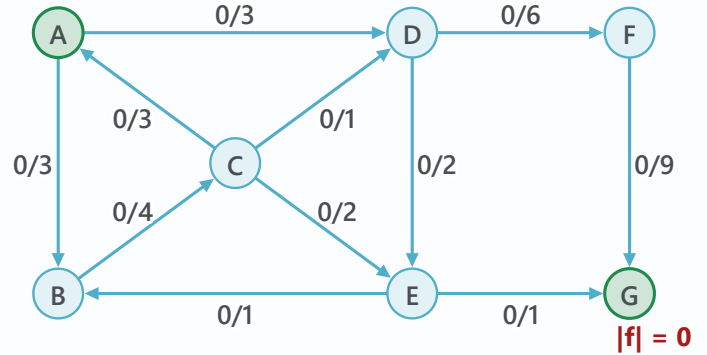
Son durum: artık ağda s'ten yalnızca 1, 2, 4'e ulaşılır. Kesim ($\{s, 1, 2, 4\}, \{3, t\}$) (kırmızı: T tarafı): $12 + 7 + 4 = \mathbf{23} = |f^*|$ (t'ye giren akış: $19 + 4$).

Edmonds-Karp Örneği 2

Artık ağ



Akış f

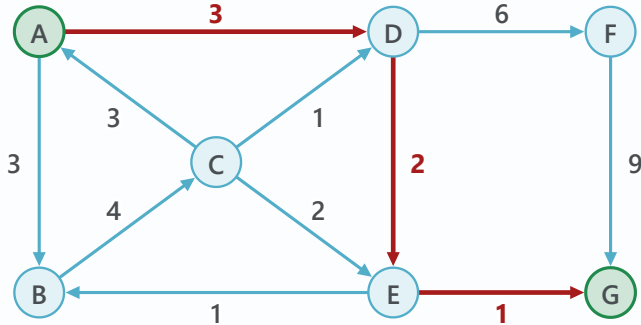


Kaynak A, hedef G. Başlangıçta tüm akışlar 0.

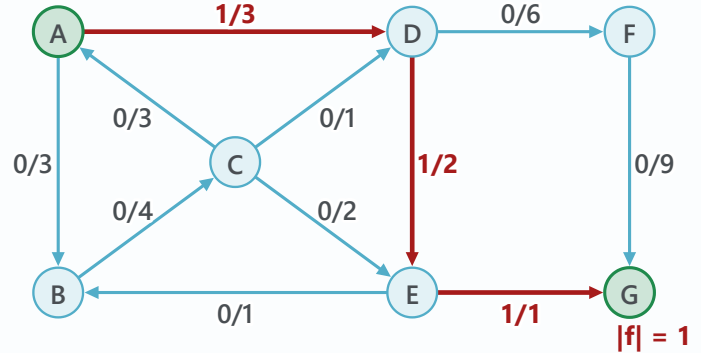
Adım 1 / 6

Edmonds-Karp Örneği 2

Artık ağ: yol (+1)



Akış f

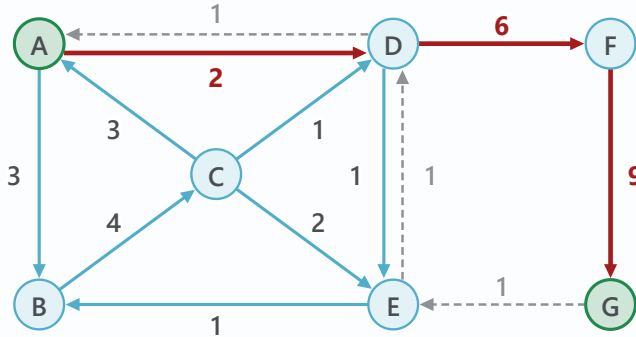


$p = \{A, D, E, G\}$: $E \rightarrow G$ (1) darboğaz. Akış = 1.

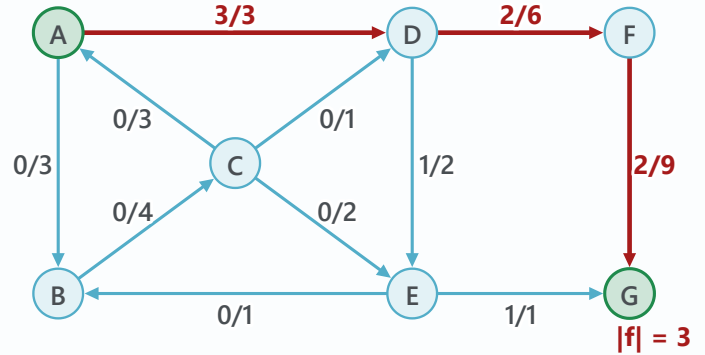
Adım 2 / 6

Edmonds-Karp Örneği 2

Artık ağ: yol (+2)



Akış f

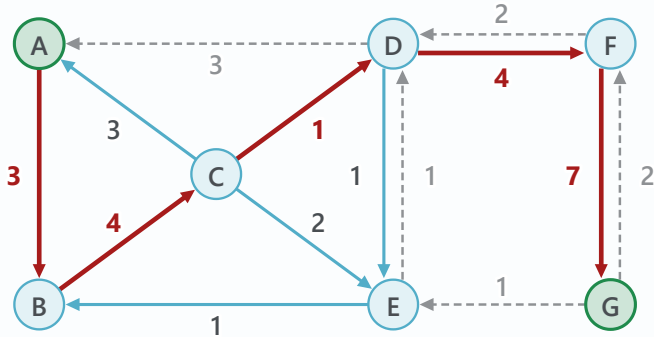


$p = \{A, D, F, G\}$: A $\rightarrow$ D'de 2 kaldı. Akış = 1 + 2 = 3.

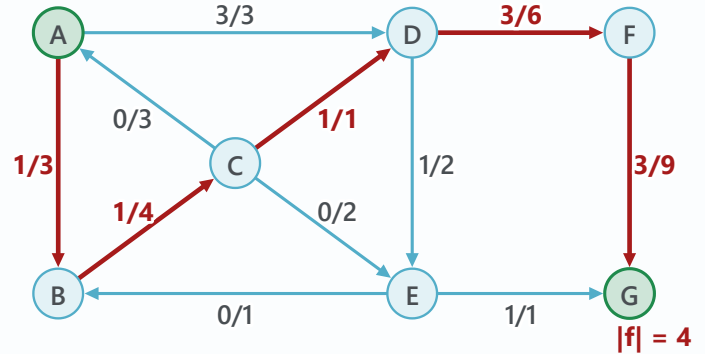
Adım 3 / 6

Edmonds-Karp Örneği 2

Artık ağ: yol (+1)



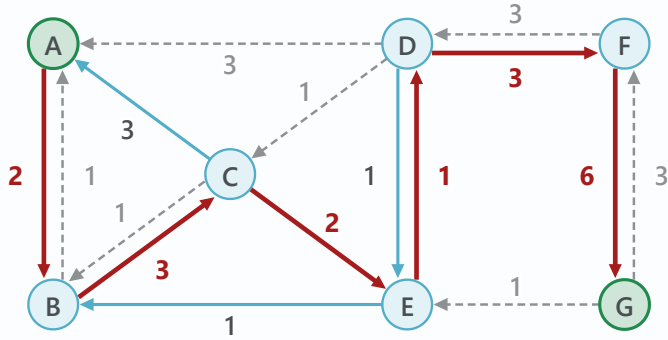
Akış f



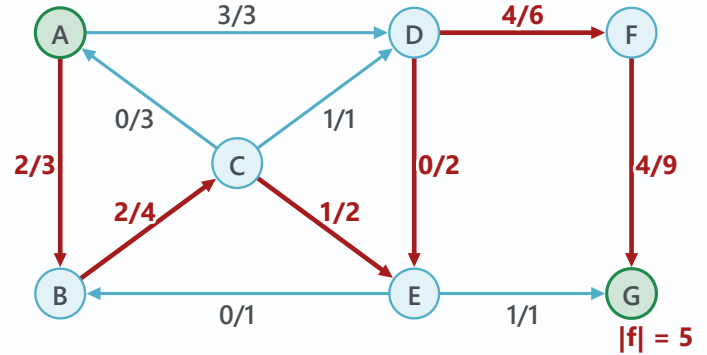
$p = \{A, B, C, D, F, G\}$: $A \rightarrow D$ doydular, BFS 5 kenarlı yolu bulur; $C \rightarrow D$ (1) darboğaz. Akış = 4.

Edmonds-Karp Örneği 2

Artık ağ: yol (+1)



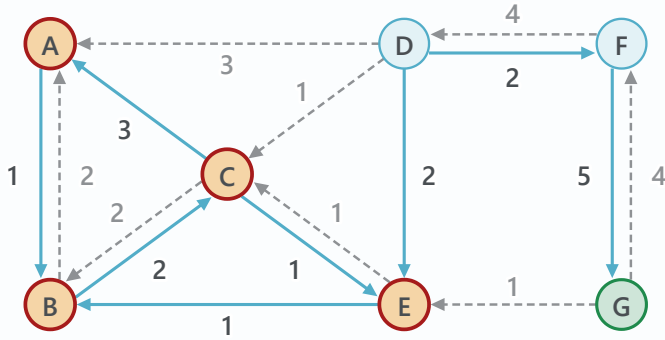
Akış f



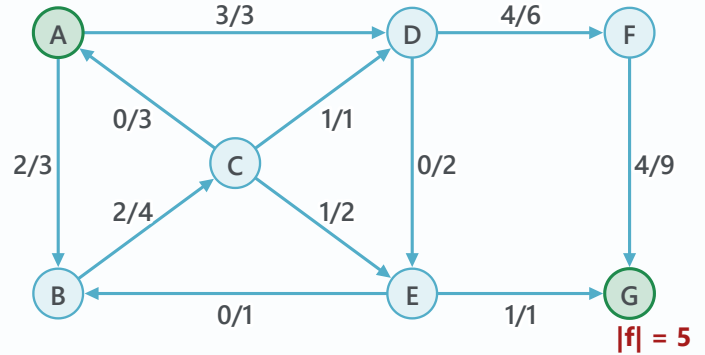
$p = \{A, B, C, E, D, F, G\}$: **E → D ters kenarı** D → E akışını 1'den 0'a indirir. Akış = 5.

Edmonds-Karp Örneği 2

Son artık ağ



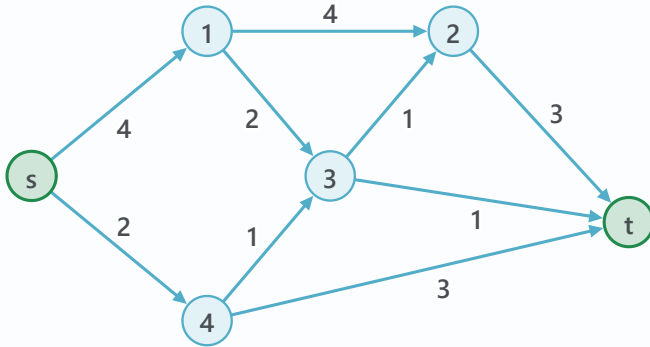
Maksimum akış



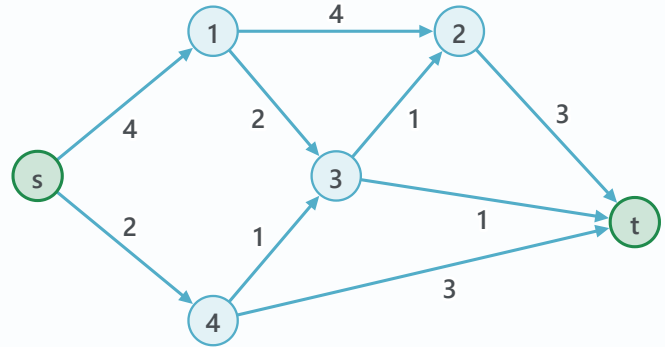
BFS G'ye ulaşamaz; A'dan A, B, C, E ulaşılabilir. Kesim kapasitesi $A \rightarrow D$ 3 + $C \rightarrow D$ 1 + $E \rightarrow G$ 1
 $= 5 = |f^*|$.

Edmonds-Karp Örneği 3: Artık Ağ Üzerinde

Ağ (kapasiteler)



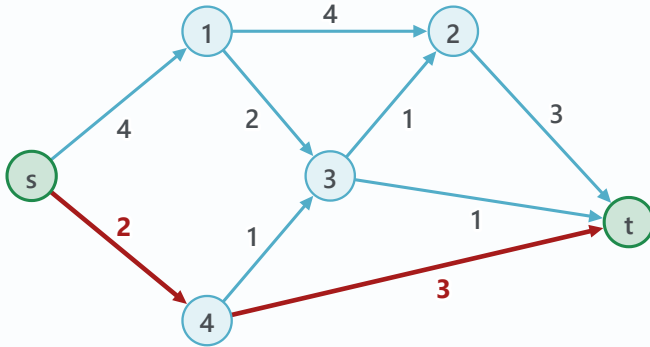
Artık ağ



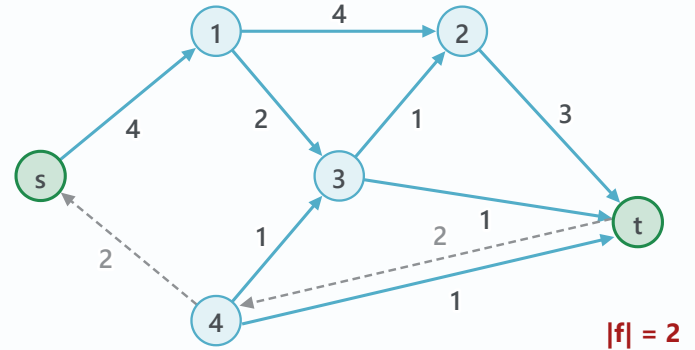
s 'ten t 'ye; başlangıçta artık ağ = kapasiteler. En kısa yolların hepsi 3 kenarlı: $s \rightarrow 4 \rightarrow t$, $s \rightarrow 1 \rightarrow 2 \rightarrow t$, $s \rightarrow 1 \rightarrow 3 \rightarrow t$.

Edmonds-Karp Örneği 3: Artık Ağ Üzerinde

Önce: yol



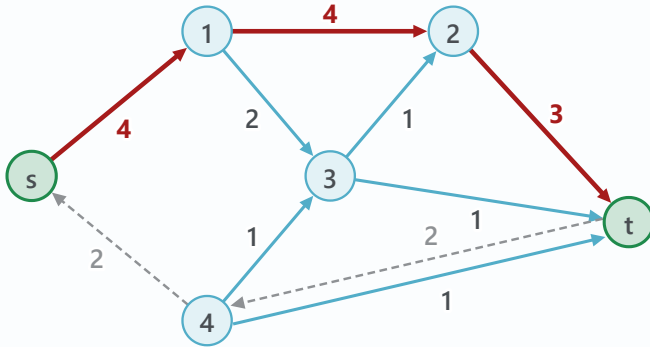
Sonra: +2



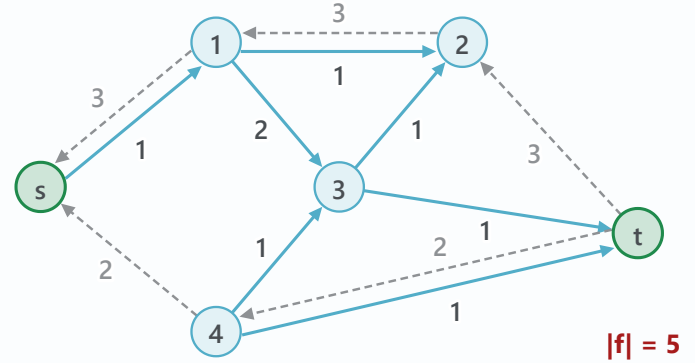
$s \rightarrow 4 \rightarrow t$ yolu: darboğaz $s \rightarrow 4$ (2). **2 birim itilir**; $s \rightarrow 4$ kalanı 0, $4 \rightarrow s$ ters kenarı 2.

Edmonds-Karp Örneği 3: Artık Ağ Üzerinde

Önce: yol



Sonra: +3

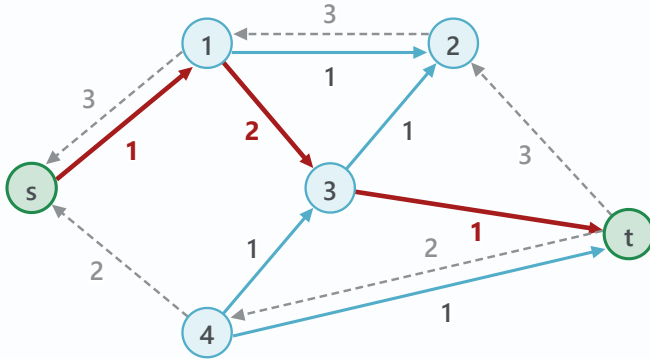


$s \rightarrow 1 \rightarrow 2 \rightarrow t$ yolu: darboğaz $2 \rightarrow t$ (3). **3 birim** itilir.

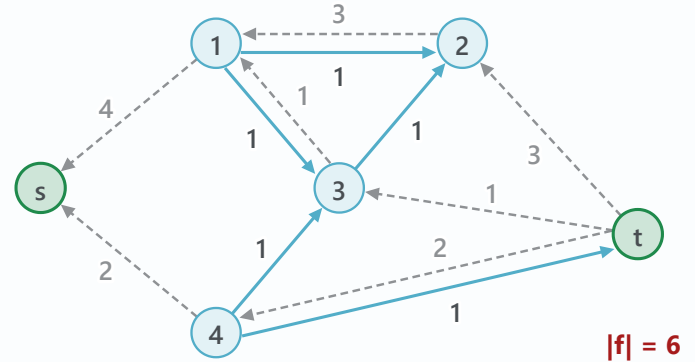
Adım 3 / 5

Edmonds-Karp Örneği 3: Artık Ağ Üzerinde

Önce: yol



Sonra: +1

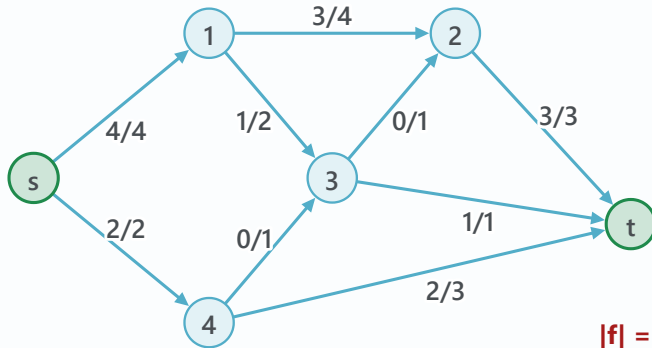


$s \rightarrow 1 \rightarrow 3 \rightarrow t$ yolu: $s \rightarrow 1$ 'de 1 kalmıştı. **1 birim itilir.**

Adım 4 / 5

Edmonds-Karp Örneği 3: Artık Ağ Üzerinde

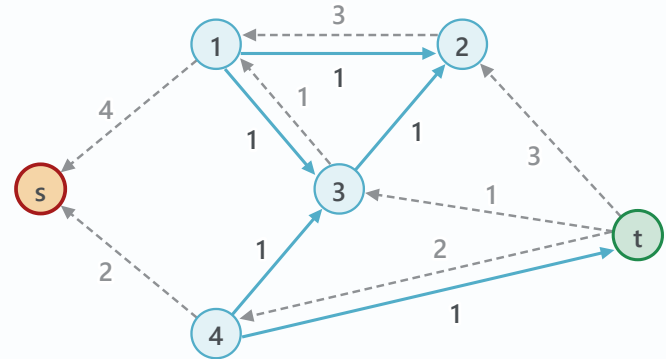
Son akış (f/c)



$|f| = 6$

s'ten çıkan kenarlar doydular → yol yok. $|f^*| = 2 + 3 + 1 = 6$: 3 birim $s \rightarrow 1 \rightarrow 2 \rightarrow t$, 1 birim $s \rightarrow 1 \rightarrow 3 \rightarrow t$, 2 birim $s \rightarrow 4 \rightarrow t$.

Son artık ağ



Dinic Algoritması

- Bir çizgede maksimum akışı bulur (Y. Dinitz, 1970); kapasiteli (ağırlıklı) çizge üzerinde çalışır.
- **Seviye çizgesi** (level graph): artık ağda BFS ile her düğümün seviyesi (kaynağa uzaklığı) bulunur; yalnızca $\text{seviye}(v) = \text{seviye}(u) + 1$ olan kenarlar tutulur.
- **Engelleyici akış** (blocking flow): seviye çizgesinde her s - t yolunda en az bir doymuş kenar bırakan akış; DFS ile bulunur.
- Akış ters kenarlarla güncellenir; hedef seviye çizgesinde ulaşılamaz olana kadar fazlar tekrarlanır.

Edmonds-Karp her BFS'den sonra **tek** yol kullanır; Dinic aynı seviye çizgesindeki **tüm** en kısa yolları tek fazda doyurur.

Dinic: Algoritma Adımları

1. Başlangıçta tüm kenarların akışına 0 ata.

Artık ağ özgün ağla aynıdır.

Adım 1 / 5

Dinic: Algoritma Adımları

1. Başlangıçta tüm kenarların akışına 0 ata.

Artık ağ özgün ağla aynıdır.

2. Kaynaktan BFS ile her düğümün **seviyesini** (kaynağa uzaklık) belirle. Hedefin seviyesi yoksa dur. BFS, artık kapasitesi pozitif kenarları kullanır; hedefe ulaşamıyorsa akış maksimumdur.

Dinic: Algoritma Adımları

1. Başlangıçta tüm kenarların akışına 0 ata.

Artık ağ özgün ağla aynıdır.

2. Kaynaktan BFS ile her düğümün **seviyesini** (kaynağa uzaklık) belirle. Hedefin seviyesi yoksa dur.
BFS, artık kapasitesi pozitif kenarları kullanır; hedefe ulaşamıyorsa akış maksimumdur.

3. DFS ile her adımda **bir seviye ileri** giden artış yolları bul.

Yalnızca $\text{seviye}(v) = \text{seviye}(u) + 1$ olan kenarlar izlenir; çıkmaz kenarlar bir daha denenmez.

Dinic: Algoritma Adımları

1. Başlangıçta tüm kenarların akışına 0 ata.

Artık ağ özgün ağla aynıdır.

2. Kaynaktan BFS ile her düğümün **seviyesini** (kaynağa uzaklık) belirle. Hedefin seviyesi yoksa dur. BFS, artık kapasitesi pozitif kenarları kullanır; hedefe ulaşamıyorsa akış maksimumdur.

3. DFS ile her adımda **bir seviye ileri** giden artış yolları bul.

Yalnızca $seviye(v) = seviye(u) + 1$ olan kenarlar izlenir; çıkmaz kenarlar bir daha denenmez.

4. Her yoldaki minimum kapasiteyi toplam akışa ekle; kenarların ve ters kenarların kapasitesini güncelle.

Yol kalmayınca bu fazın engelleyici akışı bulunmuştur.

Dinic: Algoritma Adımları

1. Başlangıçta tüm kenarların akışına 0 ata.

Artık ağ özgün ağla aynıdır.

2. Kaynaktan BFS ile her düğümün **seviyesini** (kaynağa uzaklık) belirle. Hedefin seviyesi yoksa dur. BFS, artık kapasitesi pozitif kenarları kullanır; hedefe ulaşamıyorsa akış maksimumdur.

3. DFS ile her adımda **bir seviye ileri** giden artış yolları bul.

Yalnızca $seviye(v) = seviye(u) + 1$ olan kenarlar izlenir; çıkmaz kenarlar bir daha denenmez.

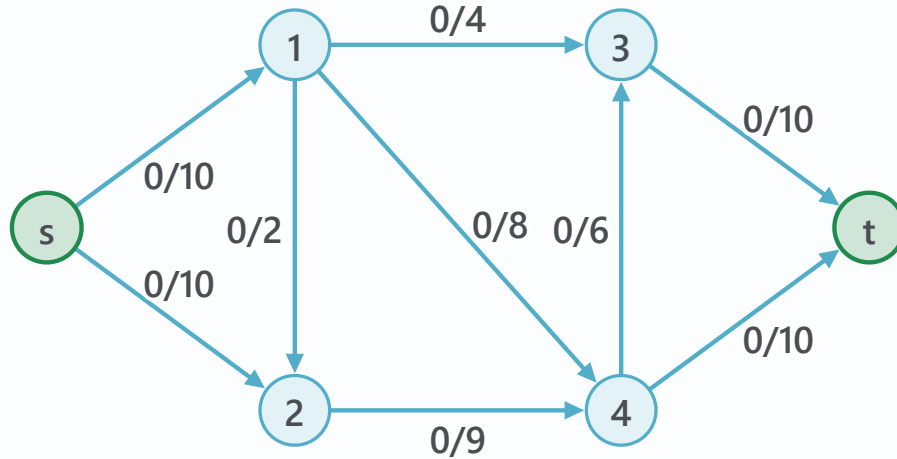
4. Her yoldaki minimum kapasiteyi toplam akışa ekle; kenarların ve ters kenarların kapasitesini güncelle.

Yol kalmayınca bu fazın engelleyici akışı bulunmuştur.

5. 2. adıma dön ve işlemleri tekrarla.

Her fazda s-t uzaklığı en az 1 artar; bu yüzden en fazla $V - 1$ faz vardır.

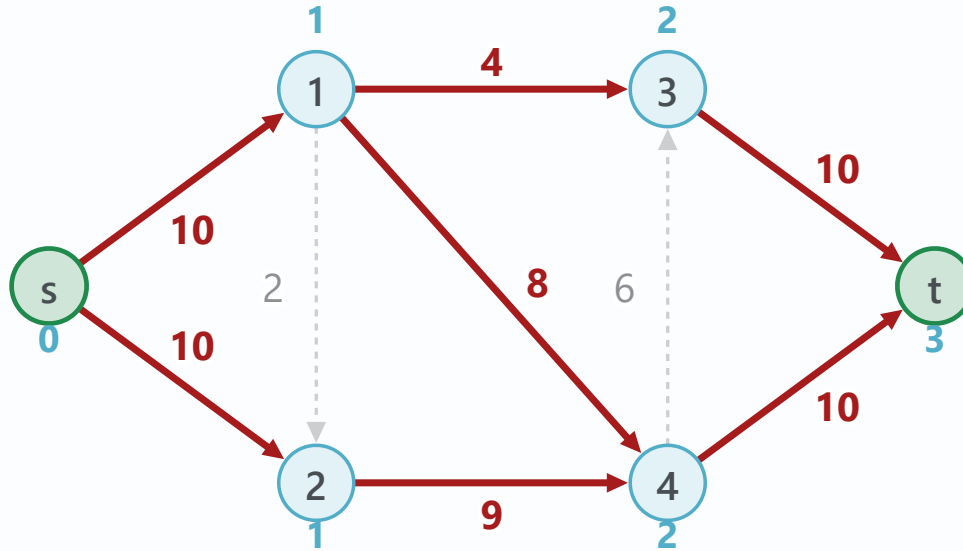
Dinic Örneği: Fazlar



$|f| = 0$

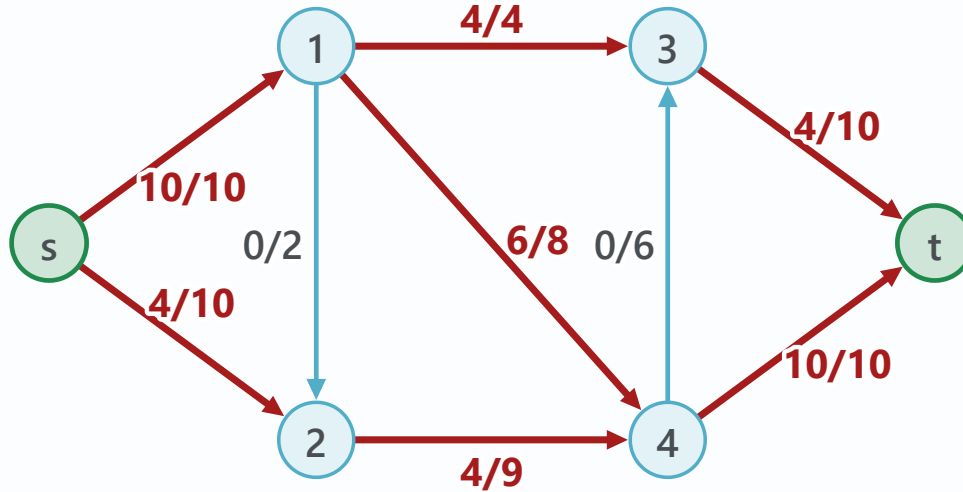
Ağ (akış/kapasite). Başlangıçta tüm akışlar 0; artık ağ = kapasiteler.

Dinic Örneği: Fazlar



Faz 1 – BFS: seviyeler $s = 0$; $1, 2 = 1$; $3, 4 = 2$; $t = 3$. Soluk kenarlar ($1 \rightarrow 2$, $4 \rightarrow 3$) seviye atlamaz, kullanılmaz.

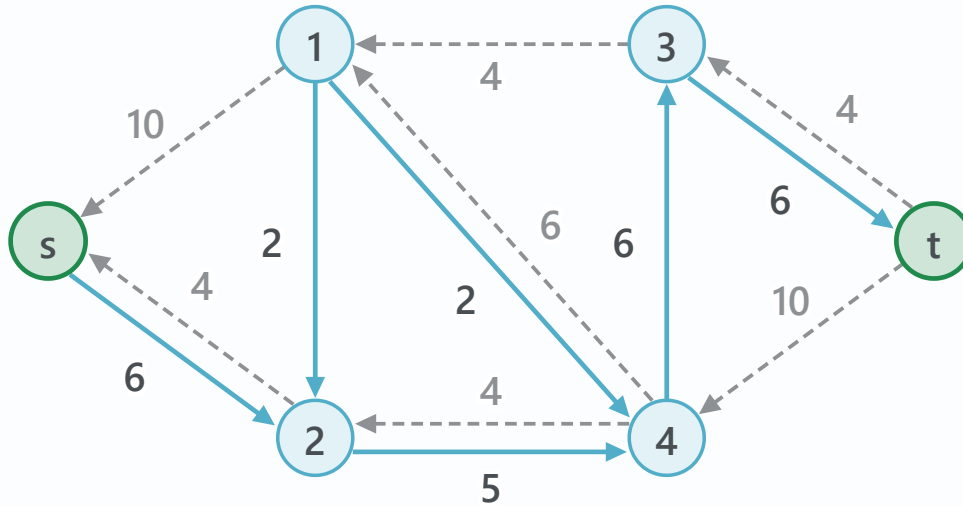
Dinic Örneği: Fazlar



$|f| = 14$

Faz 1 – engelleyici akış (DFS): $\{s, 1, 3, t\}$ 4 birim, $\{s, 1, 4, t\}$ 6 birim, $\{s, 2, 4, t\}$ 4 birim $\rightarrow |f| = 14$.

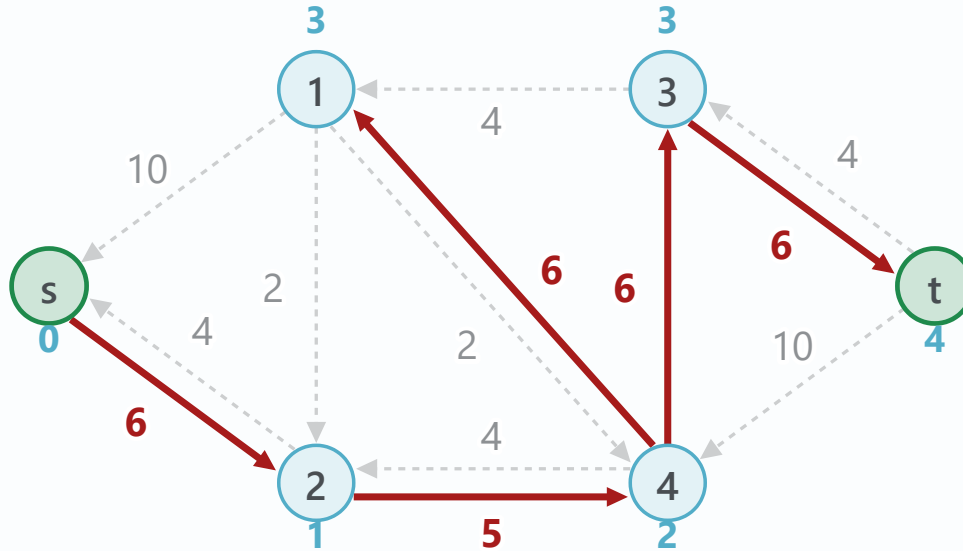
Dinic Örneği: Fazlar



$$|f| = 14$$

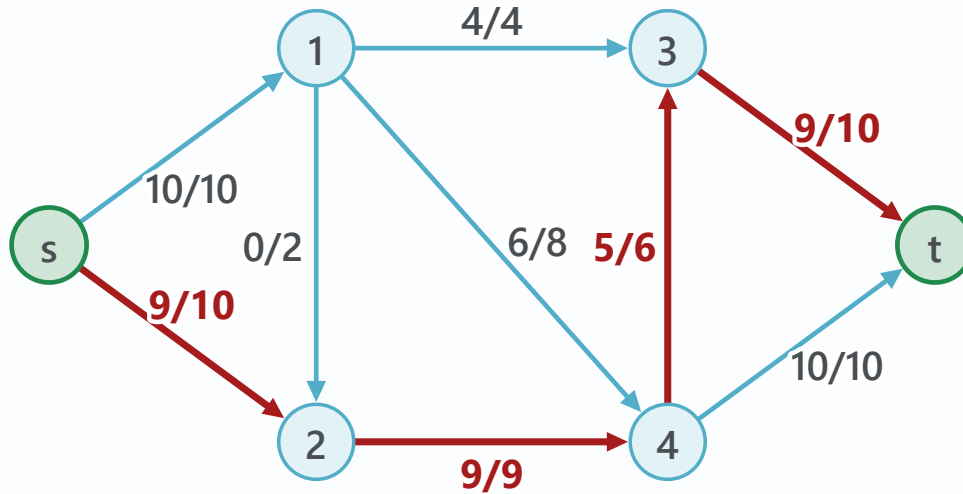
Faz 1 sonrası artık ağ (akış çizgesi G_f): doymuş kenarlar yalnızca ters yönde (kesikli) kalır.

Dinic Örneği: Fazlar



Faz 2 – BFS: $s = 0$; $2 = 1$; $4 = 2$; $1, 3 = 3$; $t = 4$. $s-t$ uzaklığı 3'ten 4'e çıktı. $4 \rightarrow 1$ kenarı çıkmaz sokaktır.

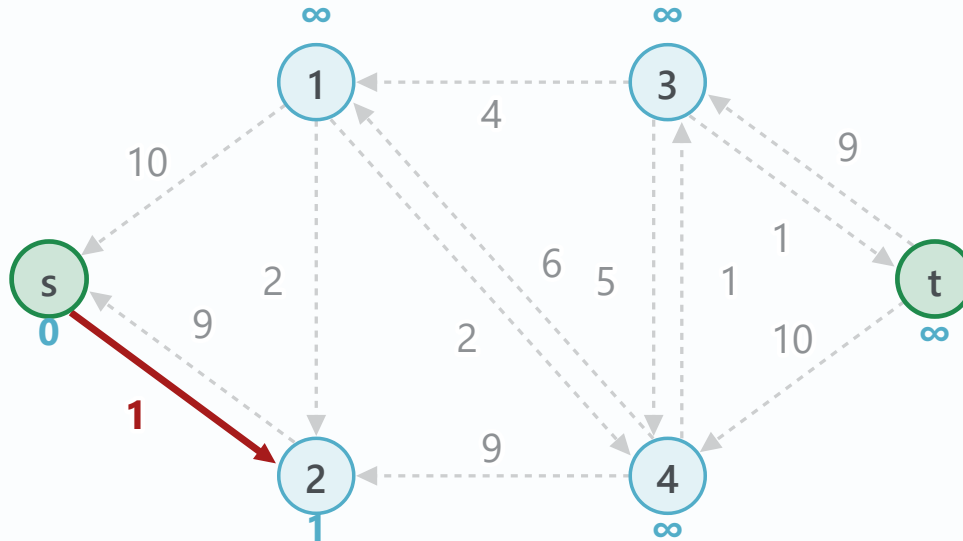
Dinic Örneği: Fazlar



$|f| = 19$

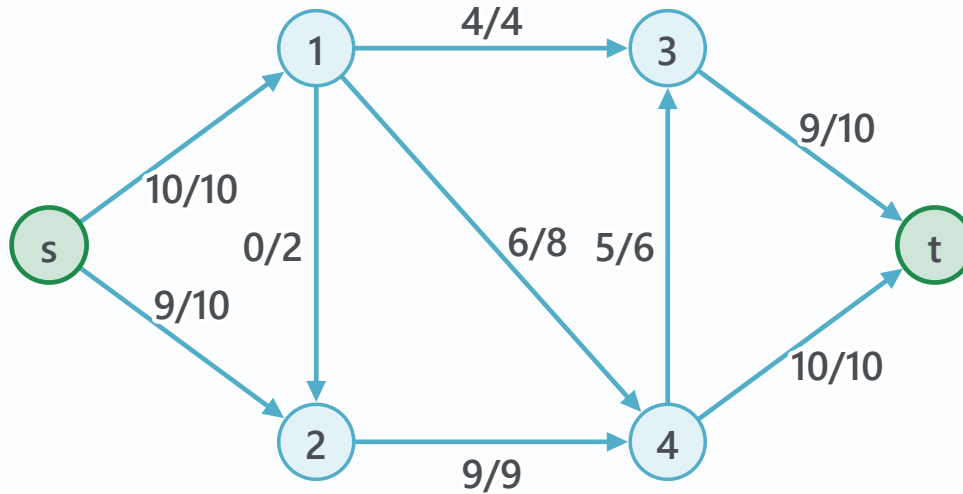
Faz 2 – engelleyici akış: $\{s, 2, 4, 3, t\}$ 5 birim (darboğaz $2 \rightarrow 4$) $\rightarrow |f| = 14 + 5 = 19$.

Dinic Örneği: Fazlar



Faz 3 – BFS: yalnızca s (0) ve 2 (1) ulaşılabilir; t 'ye yol yok $\rightarrow$ algoritma biter. Min kesim ($\{s, 2\}$, diğerleri) = $10 + 9 = 19$.

Dinic Örneği: Fazlar



$|f| = 19$

Sonuç: maksimum akış **19**; yalnızca **2 faz** (2 + 1 BFS) yetti. Aynı ağda Ford-Fulkerson 5 artış yolu kullanmıştı.

Dinic: Sözde Kod

```

DINIC(G, kaynak, hedef):
    akış = [ ]           // her kenar için 0
    maksimumAkış = 0
    döngü BFS(G, kaynak, hedef, akış) iken:
        // seviye çizgesi kuruldu
        döngü engelleyici akış var iken:
            delta = DFS(G, kaynak, hedef,
                        akış, seviye, ∞)
            maksimumAkış += delta
    döndür maksimumAkış

BFS(G, kaynak, hedef, akış):
    seviye[v] = -1, her v için
    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)
    döndür seviye[hedef] ≥ 0

```

```

DFS(G, u, hedef, akış, seviye, lim):
    eğer u == hedef: döndür lim
    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(lim, kalan))
                eğer delta > 0:
                    akış[u, v] += delta
                    akış[v, u] -= delta    // ters
                döndür delta
    döndür 0

```

Java: Seviyeli DFS ve Kaldığı Yer Dizisi

```
int dfs(int u, int t, int akis, int[] basla) {
    if (u == t) return akis;
    for (; basla[u] < komsu[u].size(); basla[u]++) {
        Kenar k = komsu[u].get(basla[u]);
        if (seviye[k.hedef] == seviye[u] + 1 && k.akis < k.kapasite) {
            int d = dfs(k.hedef, t, Math.min(akis, k.kapasite - k.akis), basla);
            if (d > 0) {
                k.akis += d;           // ileri kenar
                komsu[k.hedef].get(k.tersIndis).akis -= d; // ters kenar
                return d;
            }
        }
    }
    return 0; // u'dan t'ye seviyeli yol kalmadı
}
```

- Kenarlar komşu listesinde tutulur; her kenar ters kenarının indisini (`tersIndis`) bilir.
- `basla[u]` : u'da denenecek sıradaki kenar (*current arc*). Çıkmaz kenar bir daha denenmez; bir fazdaki tüm DFS'ler toplam $O(V \cdot E)$.
- Her fazın başında `basla` sıfırlanır, seviyeler BFS ile yeniden hesaplanır.

Dinic: Karmaşıklık $O(V^2 \cdot E)$

- Her faz bir BFS gerektirir: $O(V + E)$.
- Her fazdan sonra s–t uzaklığı **kesin olarak artar** ve uzaklık $\leq V - 1$ olduğundan en kötü durumda **V faz** olabilir.
- Bir fazdaki engelleyici akış: her artış yolu $O(V)$ uzunluktadır ve en fazla E yol bulunur; kaldığı yer dizisiyle faz başına $O(V \cdot E)$.
- Toplam: $O(V) \cdot O(V \cdot E) = O(V^2 \cdot E)$.

Birim kapasiteler

Tüm kapasiteler 1 ise $O(E \cdot \sqrt{E})$; iki parçalı eşleştirme gibi birim ağırlarda $O(E \cdot \sqrt{V})$ (Hopcroft-Karp).

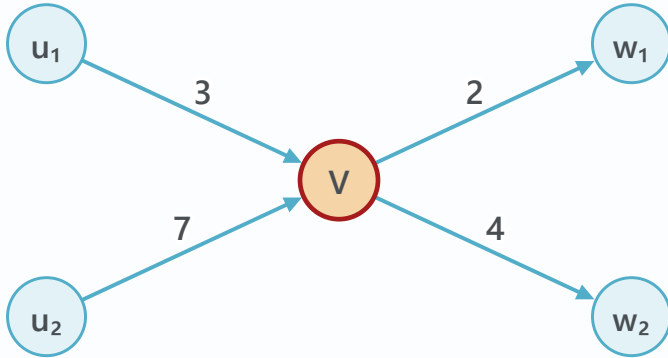
Pratikte

Yarışma programlamada en çok kullanılan maksimum akış algoritmasıdır: basit ve genellikle sınırının çok altında çalışır.

İtme ve Yeniden Etiketleme (Push-Relabel)

- A. Goldberg ve R. Tarjan (1988). Artış yolu **aramaz**; düğümler üzerinde yerel işlemlerle çalışır.
- Kaynaktan çıkan tüm kenarları doldurarak **aşırı akış** başlatır; akış, korunum kısıtını geçici olarak bozan bir **ön akıştır** (preflow).
- Her düğümün bir **yükseklik** (height) etiketi vardır; akış yalnızca **yüksekten bir alçağa** doğru itilir (push), su gibi.
- Bir düğüm fazlasını itemiyorsa yüksekliği artırılır (**relabel**).
- Aşırı akışı olan düğüm kalmayınca ön akış gerçek bir akıştır; **hedefteki aşırı akış** maksimum akışı verir.

Ön Akış ve Aşırı Akış (Excess Flow)



$$e(V) = (3 + 7) - (2 + 4) = 4$$

- **Aşırı akış:** $e(v)$ = giren toplam akış – çıkan toplam akış.
- **Ön akış:** kapasite kısıtına uyar, korunum yerine $e(v) \geq 0$ ister (s hariç).
- $e(v) > 0$ olan düğüm (s ve t dışında) **aktif** düğümdür; algoritma aktif düğüm kalmayınca biter.

Yükseklik Fonksiyonu ve Başlangıç

Başlangıç:

- Tüm kenarların akışı 0.
- $h(s) = |V|$ (düğüm sayısı); diğer tüm düğümler $h = 0$.
- s 'ten çıkan her kenar **doldurulur**: $f(s, v) = c(s, v)$, $e(v) = c(s, v)$.

Değişmezler:

- $h(s) = |V|$ ve $h(t) = 0$ hiç değişmez.
- Artık ağdaki her (u, v) kenarı için **$h(u) \leq h(v) + 1$** .
- Bu yüzden $h(u)$, u 'dan t 'ye artık ağ uzaklığının bir **alt sınırıdır**.

$h(s) = |V|$ seçilir, çünkü s 'ten t 'ye artık ağda yol olsaydı uzunluğu $\leq |V| - 1$ olurdu ve $h(s) \leq |V| - 1$ gerekirdi. Bu yüzden artık ağda hiçbir zaman $s-t$ yolu yoktur; sonunda akış maksimumdur.

İtme (Push) ve Yeniden Etiketleme (Relabel)

İtme: push(u, v)

Koşul: $e(u) > 0$, $c_f(u, v) > 0$ (kenarda kapasite ile akış arasında fark var) ve **$h(u) = h(v) + 1$** .

$\delta = \min(e(u), c_f(u, v))$ kadar akış itilir: $e(u) -= \delta$, $e(v) += \delta$. Kenar doyarsa *doyuran*, aksi hâlde *doyurmayan* itme.

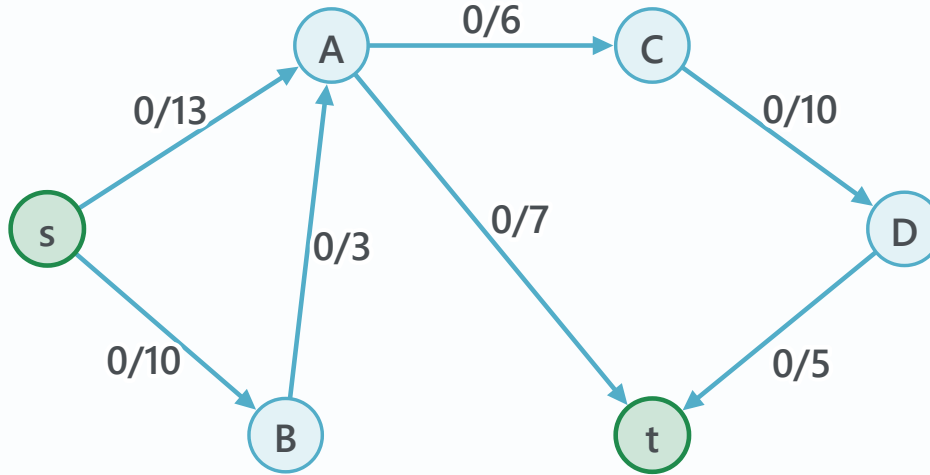
Yeniden etiketleme: relabel(u)

Koşul: $e(u) > 0$ ama artık ağdaki her (u, v) için $h(u) \leq h(v)$; yani hiçbir komşuya itilemiyor.

$h(u) = 1 + \min h(v)$, (u, v) artık ağda. Böylece en az bir itme mümkün olur.

Hedefe ulaşamayan fazla, yükseklik $|V|$ 'yi aşınca **kaynağa geri** itilir. Akış yüksekten alçağa itilir; yükseklik yalnızca artar.

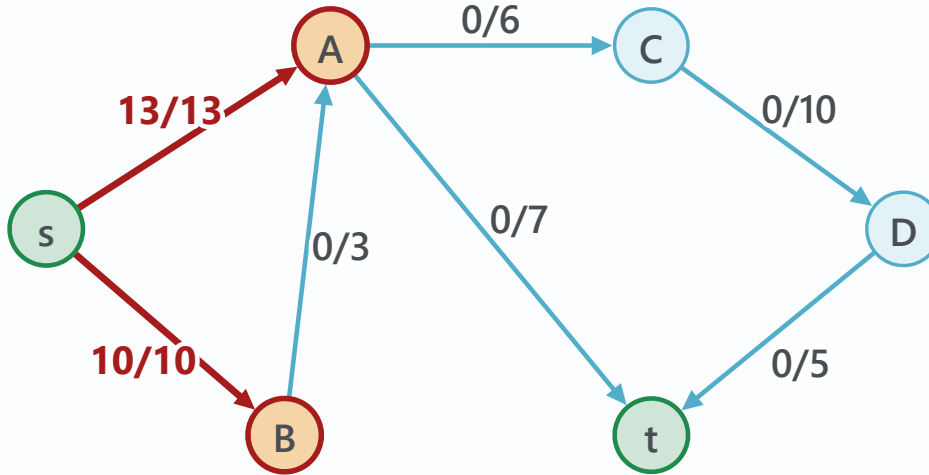
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	0	–
A	0	0
B	0	0
C	0	0
D	0	0
t	0	0

Ağ verilsin (akış/kapasite). 6 düğüm var: $|V| = 6$.

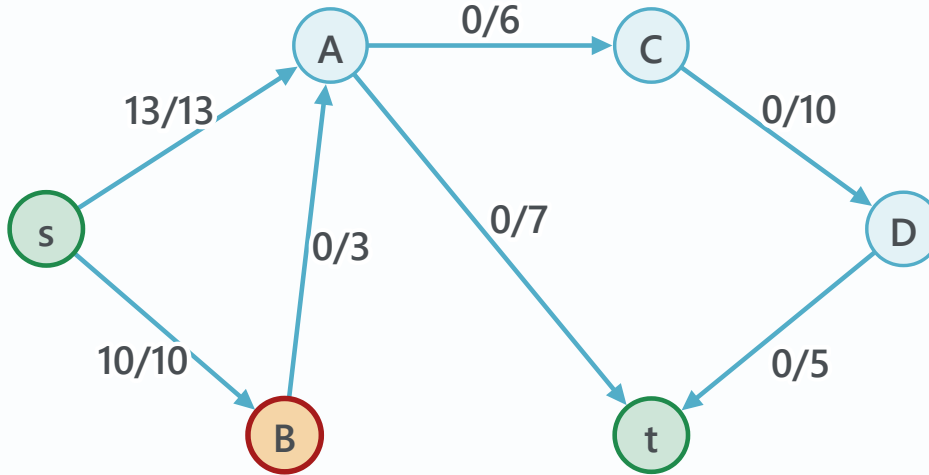
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	0	13
B	0	10
C	0	0
D	0	0
t	0	0

Başlangıç: $h(s) = 6$; $s \rightarrow A$ ve $s \rightarrow B$ doldurulur. $e(A) = 13$, $e(B) = 10$.

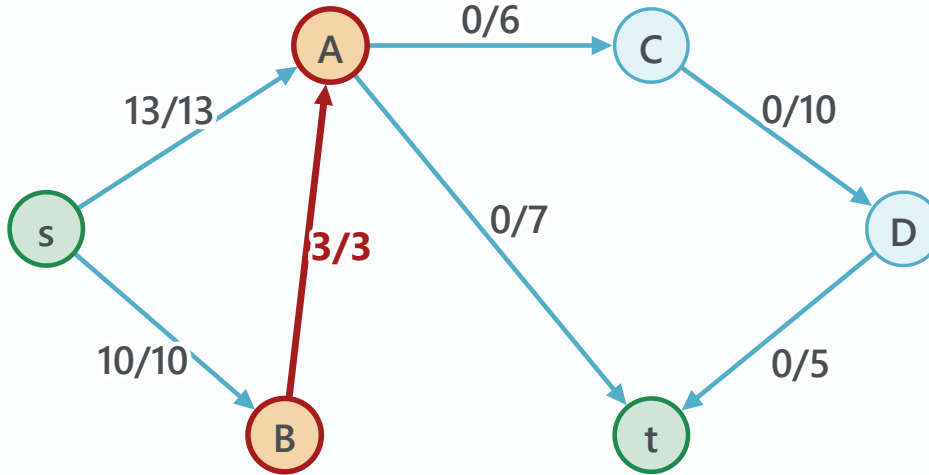
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	0	13
B	1	10
C	0	0
D	0	0
t	0	0

B ele alınır. A ile aynı yükseklikte (0) olduğundan fazlasını A'ya gönderemez → **relabel**: $h(B) = 1 + \min(h(A) = 0, h(s) = 6) = 1$.

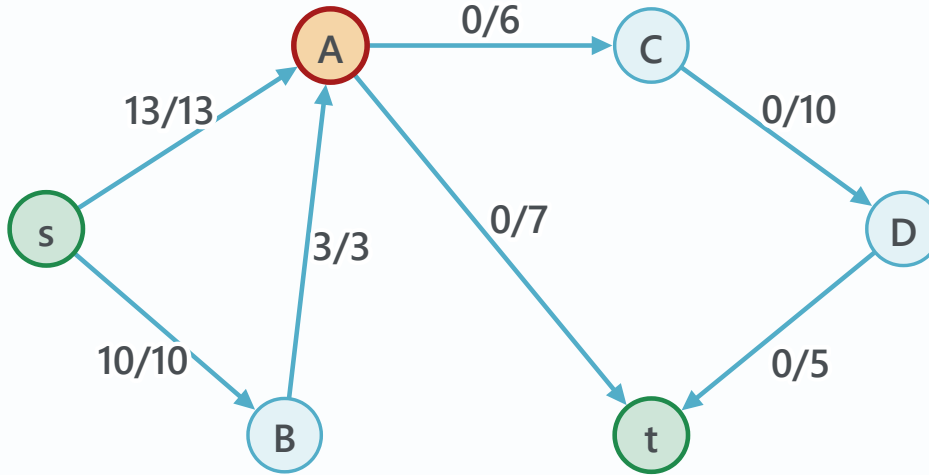
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	-
A	0	16
B	1	7
C	0	0
D	0	0
t	0	0

B şimdi fazlasını A'ya itebilir: $\delta = \min(10, 3) = 3$. $e(A) = 16$, $e(B) = 7$.

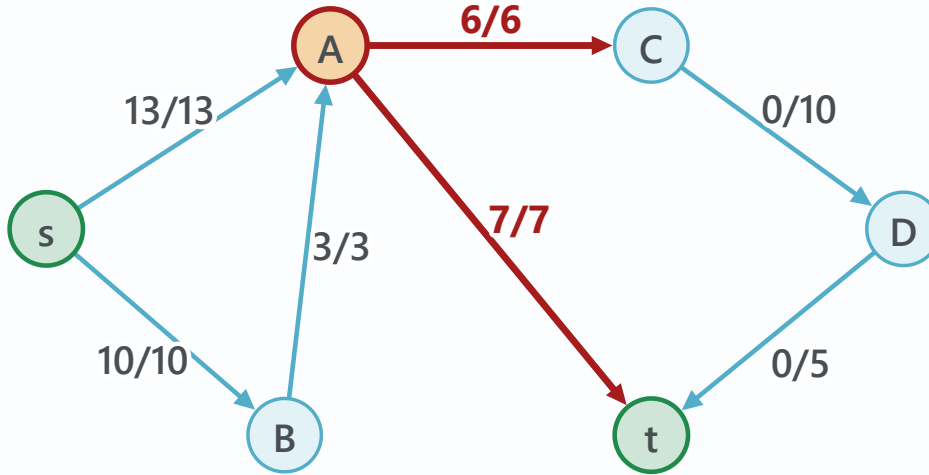
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	1	16
B	1	7
C	0	0
D	0	0
t	0	0

A ele alınır. İtebileceği alçak komşu yok $\rightarrow$ relabel: $h(A) = 1 + \min(h(C) = 0, h(t) = 0, h(B) = 1, h(s) = 6) = 1$.

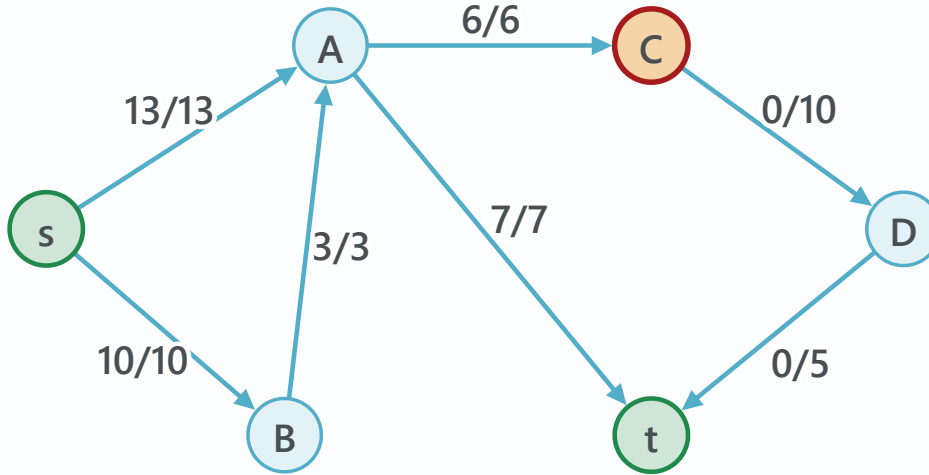
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	1	3
B	1	7
C	0	6
D	0	0
t	0	7

A şimdi C'ye 6 ve hedefe 7 birim iter (iki kenar da doyar). $e(A) = 3$, $e(C) = 6$, hedefte 7.

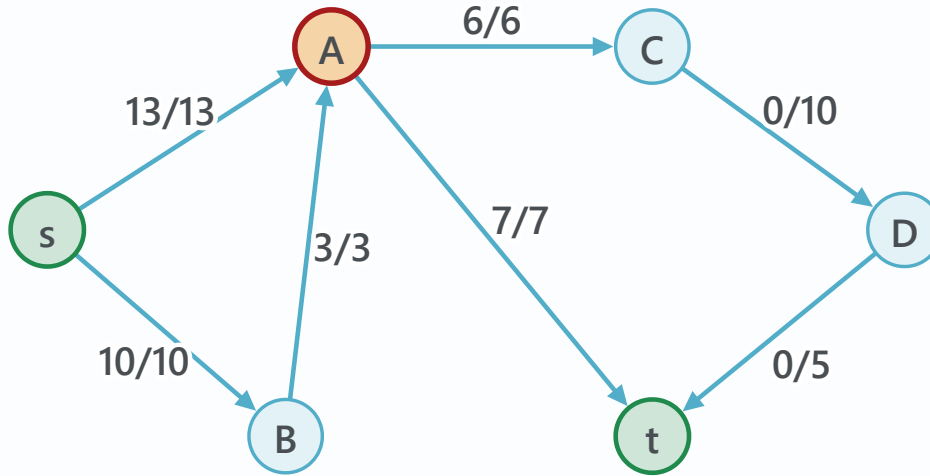
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	1	3
B	1	7
C	1	6
D	0	0
t	0	7

C relabel edilir: $h(C) = 1 + h(D) = 1$. (C'nin 6 birim fazlası şimdilik bekler; aktif düğümler istenen sırada seçilebilir.)

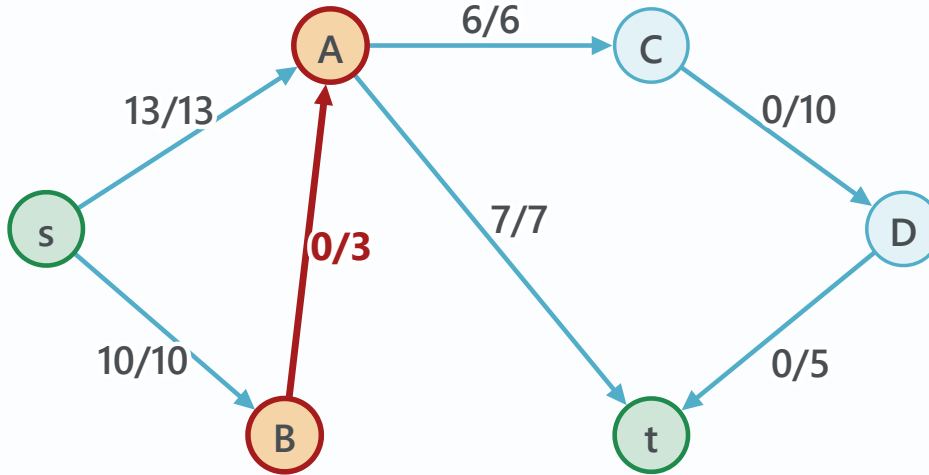
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	2	3
B	1	7
C	1	6
D	0	0
t	0	7

A relabel edilir: artık ağda yalnızca B ($h = 1$) ve s ($h = 6$) kaldı $\rightarrow h(A) = 2$.

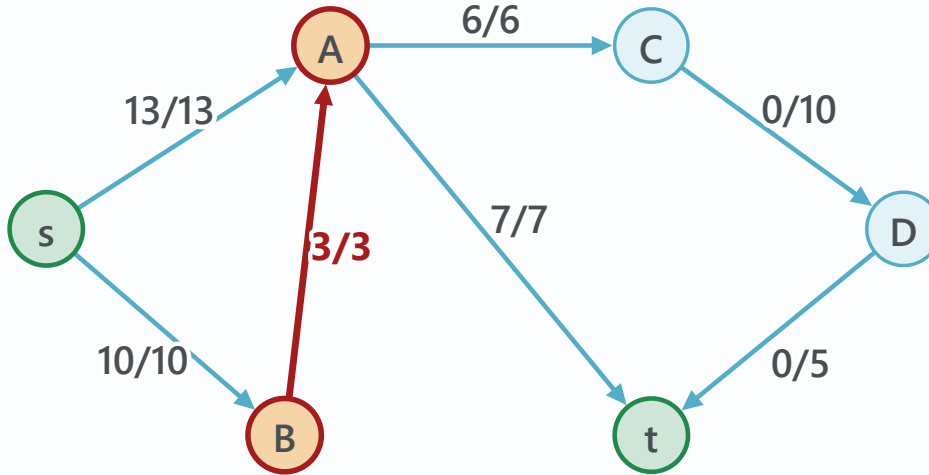
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	2	0
B	1	10
C	1	6
D	0	0
t	0	7

A'nın yüksekliği B'den fazla: 3 birimi B'ye ($B \rightarrow A$ akışını iptal ederek) iter. $e(A) = 0$, $e(B) = 10$.

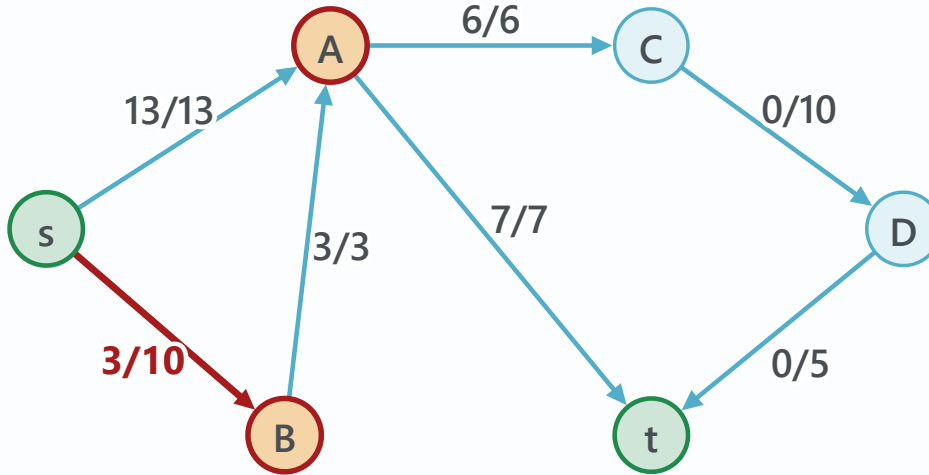
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	2	3
B	3	7
C	1	6
D	0	0
t	0	7

B'nin gidebileceği başka kenar yok $\rightarrow$ relabel $h(B) = 3$; 3 birimi tekrar A'ya iter. $e(A) = 3$, $e(B) = 7$.

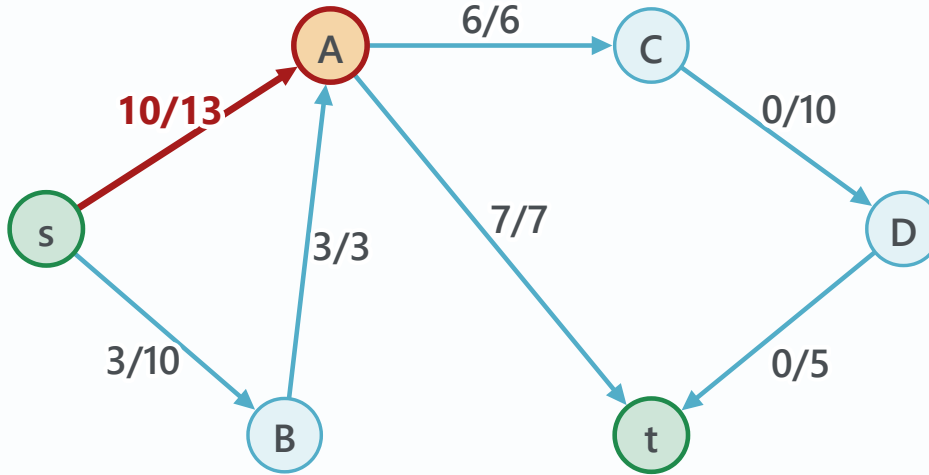
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	6	3
B	7	0
C	1	6
D	0	0
t	0	7

A ile B arasında relabel-push sürer ($h(A) = 4$, $h(B) = 5$, $h(A) = 6$, $h(B) = 7$). B'nin yüksekliği kaynağı geçince: 3 birimi A'ya, **7 birimi kaynağa** iter.

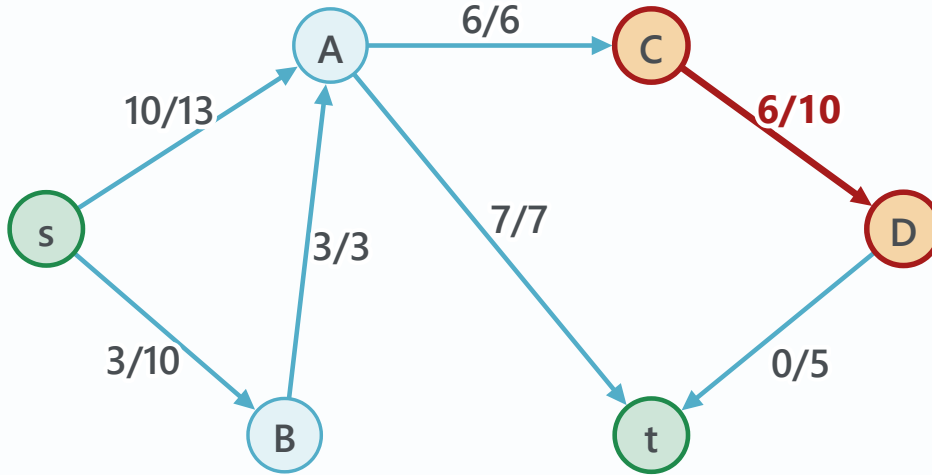
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	7	0
B	7	0
C	1	6
D	0	0
t	0	7

A relabel: $h(A) = 1 + \min(h(s) = 6, h(B) = 7) = 7$; 3 birimi **kaynağa** iter. Artık A ve B'de fazla yok.

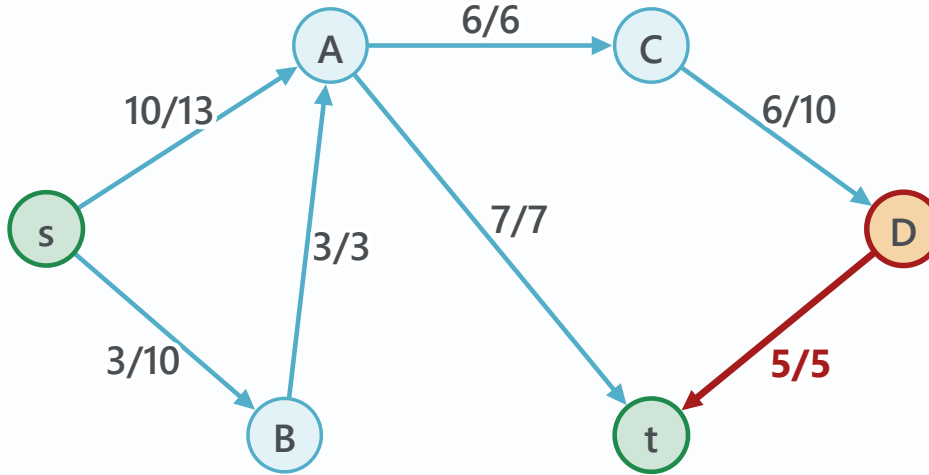
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	7	0
B	7	0
C	1	0
D	0	6
t	0	7

C ele alınır: fazlası 6 birim D'ye itilir ($h(C) = 1 > h(D) = 0$).

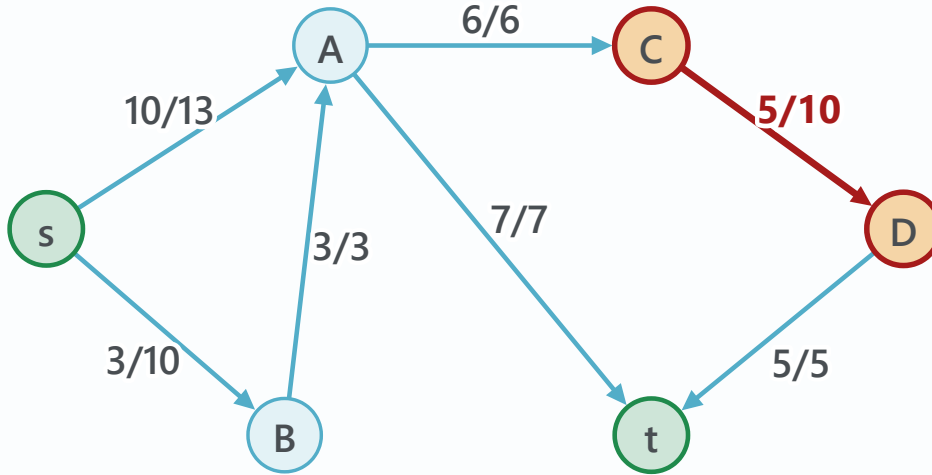
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	7	0
B	7	0
C	1	0
D	1	1
t	0	12

D relabel ($h = 1$) ve hedefe 5 birim iter. D'de hâlâ **1** birim fazla var.

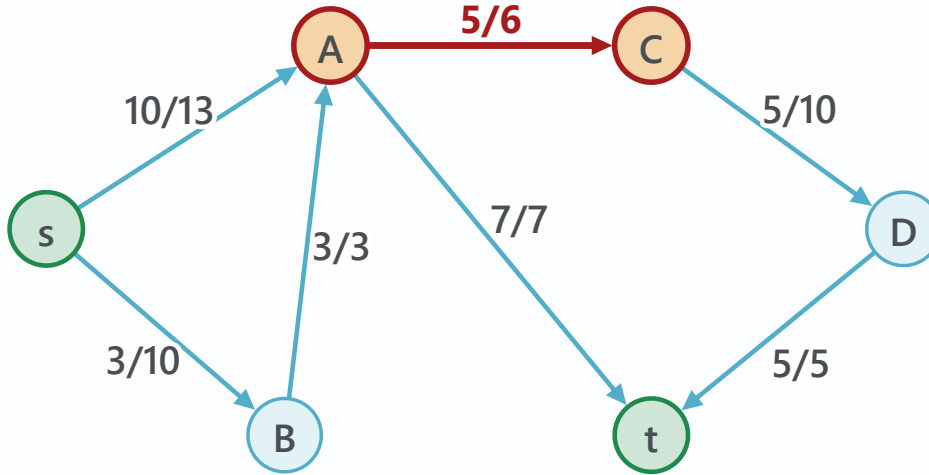
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	7	0
B	7	0
C	1	1
D	2	0
t	0	12

D relabel: hedef kenarı dolu, tek komşu C ($h = 1$) $\rightarrow h(D) = 2$. 1 birimi C'ye iter.

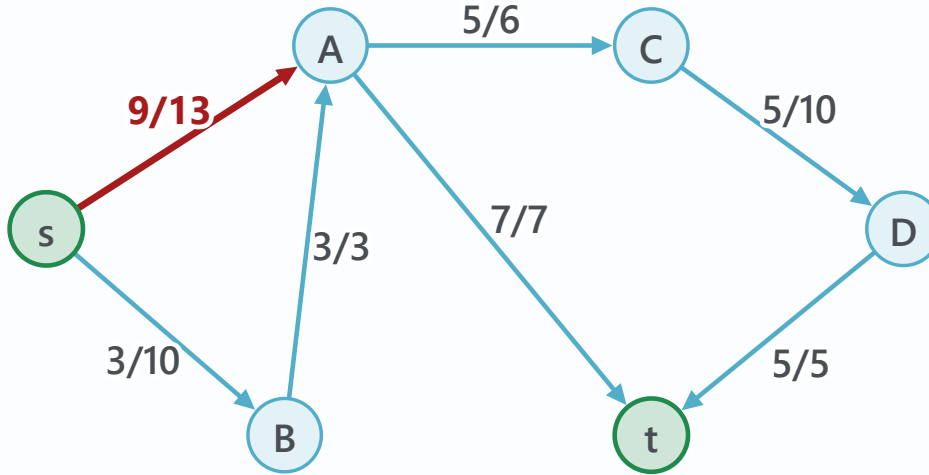
İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	–
A	7	1
B	7	0
C	8	0
D	8	0
t	0	12

C'nin yüksekliği A'yı (7) geçene kadar C–D arasında relabel–push sürer (C: 3, 5, 7, 8; D: 4, 6, 8). C 1 birim fazlayı A'ya iter.

İtme-Yeniden Etiketleme Örneği



Düğüm	h	e
s	6	-
A	7	0
B	7	0
C	8	0
D	8	0
t	0	12

A 1 birim fazlayı kaynağa iter. Aktif düğüm kalmadı: **maksimum akış = hedefe giren = 7 + 5 = 12** = kaynaktan çıkan 9 + 3.

İtme-Yeniden Etiketleme: Sözde Kod

```
PUSH_RELABEL(G, kaynak, hedef):  
    yükseklik[kaynak] = |V|  
    yükseklik[v] = 0, v ≠ kaynak  
    aşırı[v] = 0, her v için  
    akış[e] = 0, her kenar e için  
    // ön akış: kaynak kenarlarını doldur  
    her bir (kaynak, v) ∈ E için:  
        c = kapasite[kaynak, v]  
        akış[kaynak, v] = c  
        akış[v, kaynak] = -c  
        aşırı[v] = c  
        aşırı[kaynak] -= c
```

```
döngü aşırı[u] > 0 olan u varken:  
    // u ≠ kaynak, u ≠ hedef  
    eğer kalan(u,v) > 0 ve  
        yükseklik[u] == yükseklik[v]+1:  
        // İtme  
        delta = min(aşırı[u], kalan(u,v))  
        akış[u,v] += delta  
        akış[v,u] -= delta  
        aşırı[u] -= delta  
        aşırı[v] += delta  
    değilse:  
        // Yeniden etiketleme  
        m = ∞  
        her bir (u, v) için G[u] içinde:  
            eğer akış[u,v] < kapasite[u,v]:  
                m = min(m, yükseklik[v])  
        yükseklik[u] = m + 1  
    döndür aşırı[hedef]  
    // kalan = kapasite - akış
```

Java: push ve relabel

```
boolean it(Dugum d) {  
    for (Kenar k : d.kenarlar) {  
        if (d.yukseklık > k.hedef.yukseklık  
            && k.akis ≠ k.kapasite) {  
            int akis = Math.min(  
                k.kapasite - k.akis, d.asiri);  
            d.asiri -= akis;  
            k.hedef.asiri += akis;  
            k.akis += akis;  
            tersKenariGuncelle(k, akis);  
            return true;    // itme yapıldı  
        }  
    }  
    return false;  
}
```

```
void etiketle(Dugum d) {  
    int minY = Integer.MAX_VALUE;  
    for (Kenar k : d.kenarlar) {  
        // yalnızca artık ağdaki kenarlar  
        if (k.akis ≠ k.kapasite  
            && k.hedef.yukseklık < minY) {  
            minY = k.hedef.yukseklık;  
            d.yukseklık = minY + 1;  
        }  
    }  
}
```

Ana döngü: aktif düğüm varken `if (!it(d)) etiketle(d);` — sonuç hedefin aşırı akışıdır. Ters kenarlar negatif akışla tutulur (kapasite 0).

İtme-Yeniden Etiketleme: Karmaşıklık

İşlem	Toplam sayı	Gerekçe
Yeniden etiketleme	$O(V^2)$	her düğümün yüksekliği $\leq 2 V - 1$ ve yalnızca artar
Doyuran itme	$O(V \cdot E)$	bir kenar yeniden doyabilmek için iki uçtaki yükseklik ≥ 2 artmalı
Doyurmayan itme	$O(V^2 \cdot E)$	potansiyel fonksiyonu $\sum h(v)$ (aktif v) ile

- Genel algoritma en kötü durumda **$O(V^2 \cdot E)$** ; bir relabel en fazla derece kadar kenarı tarar (toplam $O(V \cdot E)$).
- **FIFO** seçimiyle $O(V^3)$, **en yüksek etiket** seçimiyle $O(V^2 \cdot \sqrt{E})$.
- Pratikte *gap* ve *global relabeling* sezgileriyle en hızlı genel amaçlı maksimum akış yöntemlerindendir.

Algoritmaların Karşılaştırması

Algoritma	Temel işlem	Süre	Örnek ağda ($ f^* = 19$)
Ford-Fulkerson	herhangi bir artış yolu (DFS)	$O(E \cdot f^*)$	5 artış yolu (Örnek 2'deki sıra)
Edmonds-Karp	en kısa artış yolu (BFS)	$O(V \cdot E^2)$	4 artış yolu, 5 BFS
Dinic	seviye çizgesi + engelleyici akış	$O(V^2 \cdot E)$	2 faz, 3 BFS
İtme–yeniden etiketleme	yerel push / relabel	$O(V^2 \cdot E)$, FIFO $O(V^3)$	yol aramaz

Seçim

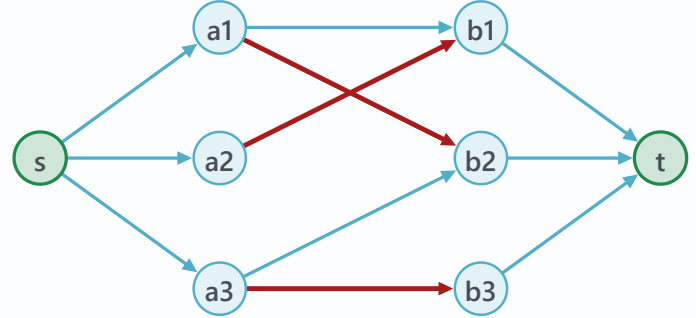
Küçük tam sayı kapasiteler: Ford-Fulkerson yeterli. Genel amaç: Dinic. Çok büyük, yoğun ağlar: itme–yeniden etiketleme.

Ortak sonuç

Hepsi aynı $|f^*|$ değerini bulur; akışın kenarlara dağılımı farklı olabilir. Min kesim son artık ağdan okunur.

Uygulamalar

- **İki parçalı eşleştirme:** çalışan–iş, öğrenci–proje atama (birim kapasiteler).
- **Kenar ayırık yollar** (Menger): birim kapasiteli maksimum akış.
- **Görüntü bölütleme:** ön plan/arka plan ayrımı minimum kesimle.
- **Ağ güvenilirliği:** ağı bölmek için kesilmesi gereken en küçük toplam kapasite.
- **Ulaşım ve lojistik:** boru hattı, trafik, havayolu çizelgeleme, proje seçimi.



Maksimum akış 3 = en büyük eşleştirme: a_1-b_2 , a_2-b_1 , a_3-b_3 .

ÖZET

Özet

Konu	Hatırlanacaklar
Akış	$0 \leq f \leq c$; ara düğümlerde giren = çıkan; $ f $ = s'ten çıkan net akış
Artık ağı	ileri $c - f$, ters f ; artış yolu G_f 'de $s \rightarrow t$ yolu, darboğaz = en küçük c_f
Kesim	$c(S, T)$ = S'den T'ye kapasiteler; maksimum akış = min kesim; S = son G_f 'de s'ten ulaşılanlar
Ford-Fulkerson	herhangi bir artış yolu; $O(E \cdot f^*)$, tam sayı kapasitelerde sonlanır
Edmonds-Karp	BFS ile en kısa yol; $O(V \cdot E^2)$
Dinic	BFS seviye çizgesi + DFS engelleyici akış; $\leq V$ faz; $O(V^2 \cdot E)$
Push-relabel	$h(s) = V $, ön akış; yüksekten alçağa it, itemezsen etiketle; $O(V^2 \cdot E)$

Etkileşimli Simülatörler

Ağ Akışı Simülatörü

4–12 düğümlü rastgele kapasiteli bir ağ üretin, düğümleri sürükleyin; Ford-Fulkerson (DFS) ve Edmonds-Karp (BFS) ile her artış yolunu ve eklenen akışı adım adım izleyip iki yöntemi aynı ağ üzerinde karşılaştırın.

Önerilen deneme

Aynı ağda önce Ford-Fulkerson'ı, sonra Edmonds-Karp'ı çalıştırın: bulunan maksimum akış aynı, artış yollarının sayısı ve sırası farklı mı? Günlükteki yolları sayın.

ÖZET

Sık Yapılan Hatalar

Ters kenarları unutmak

Yalnızca ileri kenarlarla aramak açgözlü bir algoritmadır ve maksimumu kaçırabilir (örnekte $18 < 19$).

Darboğazı kapasiteyle hesaplamak

Darboğaz, kapasitelerin değil **artık kapasitelerin** minimumudur: $c - f$ (ya da ters kenarda f).

Min kesimi kenar sayısı sanmak

Min kesim en az **kenarı** değil, en küçük **kapasite toplamını** keser.

$T \rightarrow S$ kenarlarını saymak

Kesim kapasitesine yalnızca S 'den T 'ye giden kenarlar girer; geri yöndekiler eklenmez.

Ford-Fulkerson'ı polinom sanmak

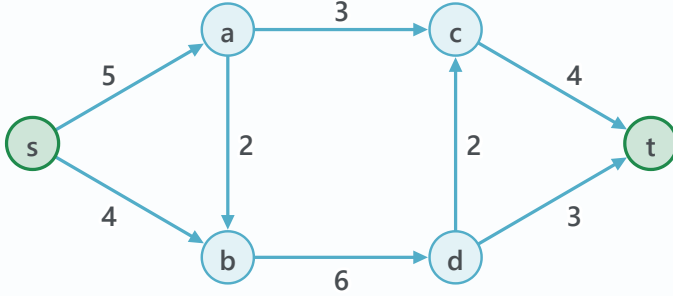
$O(E \cdot |f^*|)$ kapasitelerin değerine bağlıdır; BFS (Edmonds-Karp) bu bağımlılığı kaldırır.

İtme yönünü karıştırmak

Akış yüksekten alçağa, yalnızca $h(u) = h(v) + 1$ iken itilir; başlangıçta $h(s) = |V|$ olmalıdır.

ÖZET

Örnek Problemler



1. $|f^*|$ için kaynak ve hedef tarafından üst sınır nedir?
2. Edmonds-Karp hangi artış yollarını sırayla bulur? (BFS komşuları alfabetik dener.)
3. Maksimum akış ve bir minimum kesim nedir?
4. Son durumda $c_f(s, a)$ ve $c_f(c, d)$ kaçtır?
5. Push-relabel başlangıcında h ve e değerleri nedir?

► Cevapları göster

ÖZET

Alıştırmalar

1. Bir kenarın kapasitesi 1 artırılırsa maksimum akış en fazla ne kadar artar?
2. Tüm kapasiteler k ile çarpılırsa maksimum akış ve minimum kesim nasıl değişir?
3. Düğüm kapasiteleri $c(v)$ varsa problem nasıl modellenir?
4. Son artık ağdan minimum kesimi bulan bir Java metodu yazın.
5. "Kötü durum" ağında DFS'li Ford-Fulkerson ve Edmonds-Karp en fazla kaç artış yapar?
6. s 'ten t 'ye kenar ayrık yol sayısı maksimum akışla nasıl bulunur?

► Cevaplar / ipuçları

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, MIT Press, 2022 (Bölüm 24: Maximum Flow).
- Kleinberg ve Tardos, *Algorithm Design*, Pearson, 2006 (Bölüm 7: Network Flow).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011 (Bölüm 6.4: Maximum Flow).
- L. R. Ford ve D. R. Fulkerson, "Maximal Flow Through a Network", *Canadian Journal of Mathematics* 8, 1956.
- J. Edmonds ve R. M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", *Journal of the ACM* 19(2), 1972.
- E. A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation", *Soviet Math. Doklady* 11, 1970.
- A. V. Goldberg ve R. E. Tarjan, "A New Approach to the Maximum-Flow Problem", *Journal of the ACM* 35(4), 1988.

Son

Bölüm 4.4: Ağ Akışı

Sonraki bölüm: Bölüm 5: Dizgi Algoritmaları · [Terim sözlüğü](#)