

Bölüm 12: Açgözlü Algoritmalar

Algoritmalar

Açgözlü yaklaşım

Para üstü

Etkinlik seçimi

İş sıralama

Boruvka

Union-Find

Kruskal

Huffman kodlama

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Yaklaşım

Açgözlü (greedy) stratejiyi tanımlamak; açgözlü seçim özelliği ve optimal alt yapının ne zaman doğru sonuç garantilediğini açıklamak.

Klasik problemler

Para üstü, etkinlik seçimi, iş sıralama ve kesirli sırt çantası problemlerini açgözlü olarak çözmek; başarısız olduğu durumlara karşı örnek vermek.

Kapsayan ağaçlar

Borůvka ve Kruskal algoritmalarını Union-Find ile uygulamak, adım adım izlemek ve $O(E \log V)$ karmaşıklığını gerekçelendirmek.

Veri sıkıştırma

Huffman ağacını kurmak, önek-özgür kodları türetmek, bir metni kodlayıp çözmek ve kazancı bit olarak hesaplamak.

Adım Adım Kazananlar

- **Açgözlü algoritmalar** (greedy algorithms), her adımda **o an en iyi görünen** seçeneği seçerek ilerler.
- Tıpkı bir **define arayan** gibi: her aşamada en fazla kazancı sağlayacak seçeneği tercih eder.
- Basamaklı bir yaklaşımdır: problemi bütün olarak ele almaz, **mevcut en iyi seçeneğe** odaklanır.
- Verilen bir karar **geri alınmaz**; geri izleme (backtracking) yoktur.
- Yolculuk planlarken her kavşakta en kısa görünen yolu seçmek gibi.

Kazanç ve risk

Hızlı ve basit bir yöntem sunar; kod yapısı genellikle kısa ve anlaşılırdır.

Ancak **en iyi (optimal) çözümü bulmayı garanti etmez**; çoğu durumda iyi bir yaklaşık çözüm üretir.

Açgözlü Paradigma: Genel Şablon

```
Cozum acgozlu(Aday[] adaylar) {  
    Cozum s = new Cozum();    // boş çözüm  
    sirala(adaylar);          // ölçüte göre  
    for (Aday a : adaylar) {  
        if (s.uygunMu(a))    // kısıt bozulmaz  
            s.ekle(a);        // geri alınmaz  
    }  
    return s;  
}
```

1. **Uygun çözümler** kümesini belirle (hangi seçimler kısıtları bozmaz?).
2. Bir **açgözlü ölçüt** seç: en ucuz, en erken biten, en kârlı, en sık...
3. Ölçüte göre **yerel olarak en iyi** adayı seç; uygunsa çözüme ekle.
4. Çözüm tamamlanana kadar tekrarla.

Çoğu açgözlü algoritmanın maliyeti **sıralama** ya da **öncelikli kuyruk** işlemlerinden gelir: tipik olarak **$O(n \log n)$** .

Ne Zaman Doğru Sonuç Verir?

Açgözlü seçim özelliği

(Greedy-choice property) Yerel olarak en iyi seçimi içeren **en az bir** en iyi çözüm vardır. İlk açgözlü seçim hiçbir zaman "yanlış yola" sokmaz.

Optimal alt yapı

(Optimal substructure) Seçim yapıldıktan sonra kalan alt problemin en iyi çözümü, ilk seçimle birlikte tüm problemin en iyi çözümünü verir.

Kanıt tekniği: değiştirme argümanı

(Exchange argument) Herhangi bir en iyi çözüm O alınır. O , açgözlü seçimi içermiyorsa, O 'daki bir öge açgözlü seçimle **değiştirilir**; çözüm uygun kalır ve daha kötü olmaz. Böylece açgözlü seçimi içeren bir en iyi çözüm bulunur; tümevarımla tüm adımlar için geçerlidir.

İki özellik de sağlanmıyorsa açgözlü algoritma **yalnızca bir sezgisel** (heuristic) yöntemdir: hızlıdır ama sonucu en iyiden kötü olabilir.

Avantajlar ve Dezavantajlar

Avantajlar

- + Anlaması ve uygulaması kolay; kod kısa.
- + Hızlı: çoğunlukla sıralama + tek geçiş, $O(n \log n)$.
- + Az bellek: yalnızca mevcut çözüm tutulur.
- + Doğru problemde (MST, Huffman, etkinlik seçimi) en iyi sonucu verir.
- + Zor problemlerde iyi bir yaklaşık çözüm ya da başlangıç noktası sağlar.

Dezavantajlar

- En iyi çözümü **garanti etmez** (para üstü {1, 3, 4}, 0/1 sırt çantası).
- Doğruluğu her problem için ayrıca **kanıtlanmalıdır**.
- Ölçüt seçimi kritik: yanlış ölçüt yanlış sonuç verir.
- Verilen karar geri alınmaz; hatayı düzeltme şansı yoktur.

Para Üstü Problemi (Coin Change)

Belirli bir tutarı **en az sayıda** madeni parayla ödemek.

Açgözlü ölçüt: kalan tutardan büyük olmayan **en büyük** parayı seç; tutar sıfır olana kadar tekrarla.

```
// para[] büyükten küçüğe sıralı
static int paraUstu(int[] para, int tutar) {
    int adet = 0;
    for (int p : para) {
        while (tutar >= p) {
            tutar -= p;
            adet++;
        }
    }
    return adet;
}
```

Her zaman doğru mu?

{50, 25, 10, 5, 1} gibi **kanonik** para sistemlerinde evet: $68 = 50 + 10 + 5 + 1 + 1 + 1$.

{4, 3, 1} sisteminde 6 için açgözlü $4 + 1 + 1$ (3 para) verir; en iyisi $3 + 3$ (2 para). Genel durumda **dinamik programlama** gerekir. Zaman: $O(k + \text{adet})$, k = para türü sayısı.

Para Üstü: Adım Adım

madeni paralar

50

25

10

5

1

seçilenler

kalan tutar

68

para sayısı: 0

Örnek 1: paralar {50, 25, 10, 5, 1}, tutar **68**. En büyük paradan başla.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 1

Kalan tutara sığan en büyük para **50** → kalan $68 - 50 = 18$.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 2

Kalan tutara sığan en büyük para **10** → kalan $18 - 10 = 8$.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 3

Kalan tutara sığan en büyük para **5** → kalan $8 - 5 = 3$.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 4

Kalan tutara sığan en büyük para **1** → kalan $3 - 1 = 2$.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 5

Kalan tutara sığan en büyük para **1** → kalan $2 - 1 = 1$.

Para Üstü: Adım Adım

madeni paralar	50	25	10	5	1	
seçilenler	50	10	5	1	1	1
kalan tutar	0	para sayısı: 6				

Kalan tutara sığan en büyük para **1** → kalan $1 - 1 = 0$. Toplam **6 para** — bu para sistemi için en iyi sonuç.

Para Üstü: Adım Adım

madeni paralar



seçilenler

kalan tutar



para sayısı: 0

Örnek 2: paralar {4, 3, 1}, tutar **6**. En büyük paradan başla.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 1

Kalan tutara sığan en büyük para **4** → kalan $6 - 4 = 2$.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 2

Kalan tutara sığan en büyük para **1** → kalan $2 - 1 = 1$.

Para Üstü: Adım Adım

madeni paralar

4

3

1

seçilenler

4

1

1

kalan tutar

0

para sayısı: 3

Kalan tutara sığan en büyük para **1** → kalan $1 - 1 = 0$.

Para Üstü: Adım Adım

madeni paralar



seçilenler



kalan tutar



para sayısı: 2

en iyi: 2 para

Açgözlü $4 + 1 + 1 = 3$ para verdi; oysa $3 + 3 = 2$ para yeterli. Yerel en iyi seçim (4) küresel en iyiyi kaçırdı.

Deneyin: Açgözlü ve En İyi Para Üstü

Paralar

1, 3, 4

Tutar

6

hesapla

TL kuruş

1, 7, 10

Açgözlü

4

1

1

3 para

En iyi (DP)

3

3

2 para

Açgözlü **en iyi değil**: 3 para, en iyi 2 para (3 + 3).

En iyi sonuç dinamik programlama ile bulunur: $enAz[t] = 1 + \min enAz[t - p]$. Paralar 1–999 arası tam sayı, en fazla 8 tür; tutar en fazla 999. Enter = hesapla.

AÇGÖZLÜ YAKLAŞIM

Açgözlü, Dinamik Programlama, Böl ve Fethet

	Açgözlü	Dinamik programlama	Böl ve fethet
Karar	her adımda tek, yerel en iyi seçim; geri alınmaz	tüm seçenekler denenir, alt sonuçlar tabloda	problem bağımsız parçalara bölünür
Alt problemler	seçimden sonra tek alt problem kalır	örtüşen alt problemler	örtüşmeyen alt problemler
Doğruluk	açgözlü seçim özelliği gerekli	optimal alt yapı yeterli	birleştirme adımına bağlı
Maliyet	genelde $O(n \log n)$	genelde polinom, daha yavaş, ek bellek	özyineleme bağıntısı ile
Örnek	Kruskal, Huffman, etkinlik seçimi	0/1 sırt çantası, genel para üstü	birleştirme sıralaması

Açgözlü yaklaşım, DP'nin "her adımda yalnızca bir seçeneğe bakan" özel durumu gibi düşünülebilir: seçimin doğru olduğu kanıtlanabiliyorsa tablo gerekmez.

Etkinlik Seçimi (Activity Selection)

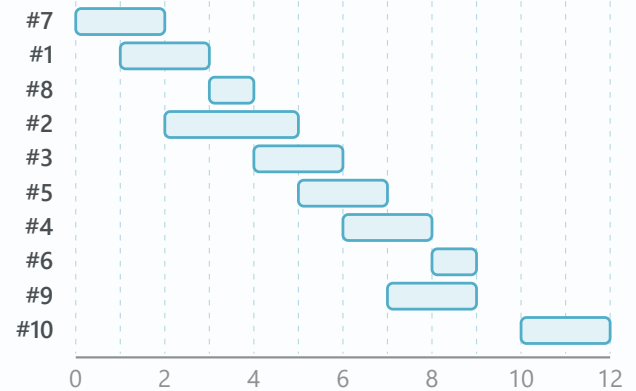
- Her etkinliğin bir **başlangıç** s_i ve **bitiş** f_i zamanı var; aynı anda tek etkinlik yapılabilir (tek salon).
- Amaç: birbiriyle çakışmayan **en fazla sayıda** etkinliği seçmek. $[s_i, f_i)$ ile $[s_j, f_j)$ çakışmaz $\Leftrightarrow s_j \geq f_i$ ya da $s_i \geq f_j$.
- Açgözlü ölçüt: en erken biten** etkinliği seç, onunla çakışanları at, kalanlarla tekrarla.

Etkinlik	1	2	3	4	5	6	7	8	9	10
başlangıç	1	2	4	6	5	8	0	3	7	10
bitiş	3	5	6	8	7	9	2	4	9	12

Zaman: sıralama $O(n \log n)$ + tek geçiş $O(n)$. Etkinlikler zaten bitişe göre sıralıysa **$O(n)$** .

sec(e): Kodu Adım Adım İzle

```
1 static void sec(Etkinlik[] e) {  
    ► Arrays.sort(e, (x, y) -> x.bitis -  
      y.bitis);  
3     int i = 0; // son seçilen  
4     System.out.print(e[i].id + " ");  
5     for (int j = 1; j < e.length; j++) {  
6         if (e[j].bas >= e[i].bitis) {  
7             System.out.print(e[j].id + " ");  
8             i = j;  
9         }  
10    }  
11 }
```



Etkinlikler **bitiş zamanına göre** sıralandı (yukarıdan aşağı). Kesikli çizgi: son seçilenin bitışı.

Adım 1 / 11

DEĞİŞKENLER

i = -

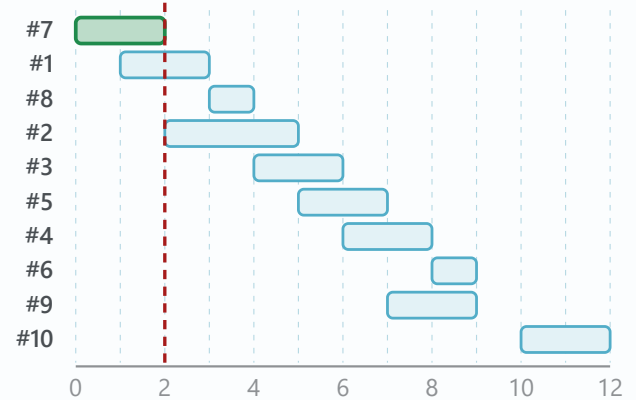
j = -

e[i].bitis = -

çıktı = -

sec(e): Kodu Adım Adım İzle

```
1 static void sec(Etkinlik[] e) {  
2     Arrays.sort(e, (x, y) -> x.bitis -  
   y.bitis);  
3     int i = 0; // son seçilen  
4     System.out.print(e[i].id + " ");  
5     for (int j = 1; j < e.length; j++) {  
6         if (e[j].bas >= e[i].bitis) {  
7             System.out.print(e[j].id + " ");  
8             i = j;  
9         }  
10    }  
11 }
```



En erken biten etkinlik (#7, [0, 2)) **her zaman** seçilir.

Adım 2 / 11

DEĞİŞKENLER

i = 0

j = -

e[i].bitis = 2

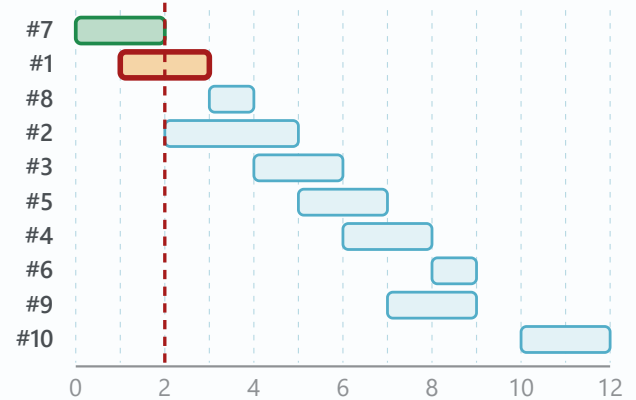
çıktı = 7

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     System.out.print(e[i].id + " ");
6     for (int j = 1; j < e.length; j++) {
7         if (e[j].bas >= e[i].bitis) {
8             System.out.print(e[j].id + " ");
9             i = j;
10        }
11    }

```



#1 [1, 3): başlangıç 1 < 2 → çakışıyor, **atlandı**.

Adım 3 / 11

DEĞİŞKENLER

i = 0

j = 1

e[i].bitis = 2

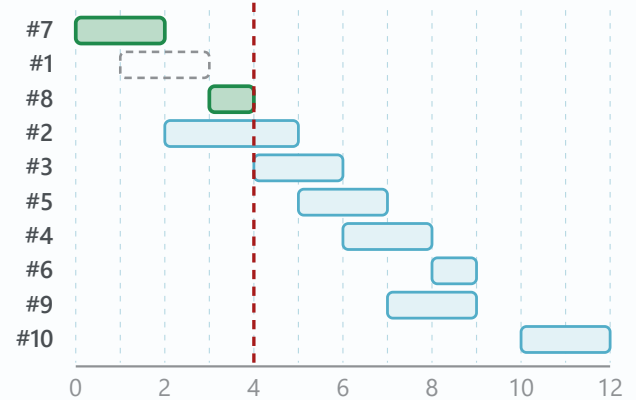
çıktı = 7

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     for (int j = 1; j < e.length; j++) {
6         if (e[j].bas >= e[i].bitis) {
7             System.out.print(e[j].id + " ");
8             i = j;
9         }
10    }
11 }

```



#8 [3, 4): başlangıç $3 \geq 2 \rightarrow$ **seçildi**, $i = 2$.

Adım 4 / 11

DEĞİŞKENLER

$i = 2$

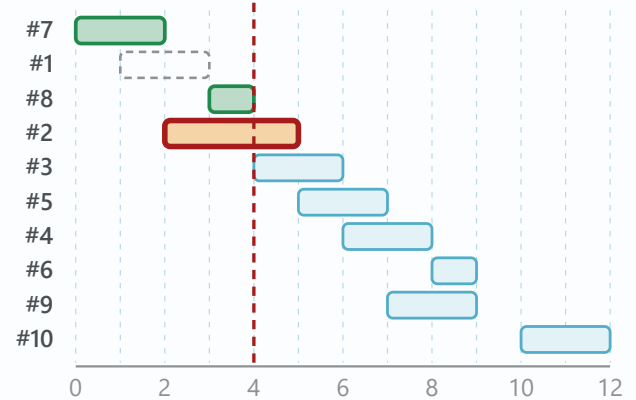
$j = 2$

$e[i].bitis = 4$

çıktı = 7 8

sec(e): Kodu Adım Adım İzle

```
1 static void sec(Etkinlik[] e) {  
2     Arrays.sort(e, (x, y) -> x.bitis -  
3         y.bitis);  
4     int i = 0; // son seçilen  
5     System.out.print(e[i].id + " ");  
6     for (int j = 1; j < e.length; j++) {  
7         if (e[j].bas >= e[i].bitis) {  
8             System.out.print(e[j].id + " ");  
9             i = j;  
10        }  
11    }  
12 }
```



#2 [2, 5): başlangıç 2 < 4 → çakışıyor, **atlandı**.

Adım 5 / 11

DEĞİŞKENLER

i = 2

j = 3

e[i].bitis = 4

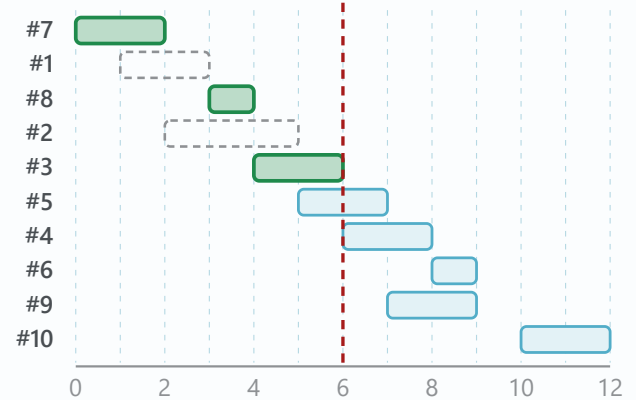
çıktı = 7 8

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     for (int j = 1; j < e.length; j++) {
6         if (e[j].bas >= e[i].bitis) {
7             System.out.print(e[j].id + " ");
8             i = j;
9         }
10    }
11 }

```



#3 [4, 6): başlangıç $4 \geq 4 \rightarrow$ **seçildi**, $i = 4$.

Adım 6 / 11

DEĞİŞKENLER

$i = 4$

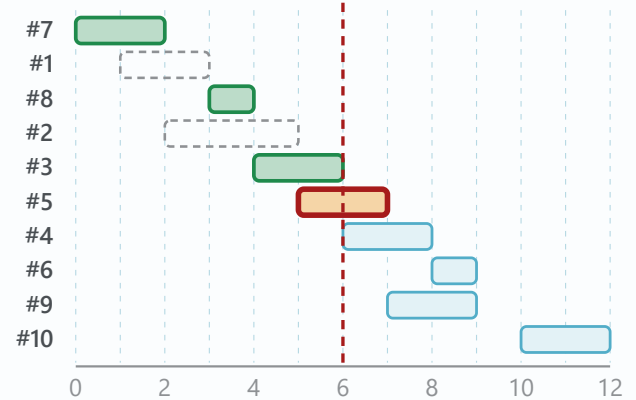
$j = 4$

$e[i].bitis = 6$

çıktı = 7 8 3

sec(e): Kodu Adım Adım İzle

```
1 static void sec(Etkinlik[] e) {  
2     Arrays.sort(e, (x, y) -> x.bitis -  
3         y.bitis);  
4     int i = 0; // son seçilen  
5     System.out.print(e[i].id + " ");  
6     for (int j = 1; j < e.length; j++) {  
7         if (e[j].bas >= e[i].bitis) {  
8             System.out.print(e[j].id + " ");  
9             i = j;  
10        }  
11    }  
12 }
```



#5 [5, 7): başlangıç 5 < 6 → çakışıyor, **atlandı**.

Adım 7 / 11

DEĞİŞKENLER

i = 4

j = 5

e[i].bitis = 6

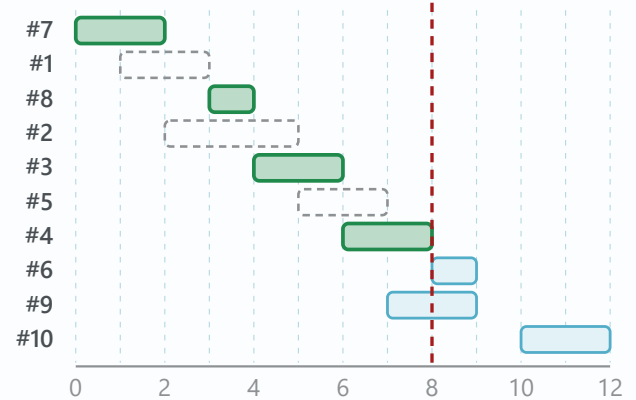
çıktı = 7 8 3

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     System.out.print(e[i].id + " ");
6     for (int j = 1; j < e.length; j++) {
7         if (e[j].bas >= e[i].bitis) {
8             System.out.print(e[j].id + " ");
9             i = j;
10        }
11    }

```



#4 [6, 8): başlangıç $6 \geq 6 \rightarrow$ **seçildi**, $i = 6$.

Adım 8 / 11

DEĞİŞKENLER

$i = 6$

$j = 6$

$e[i].bitis = 8$

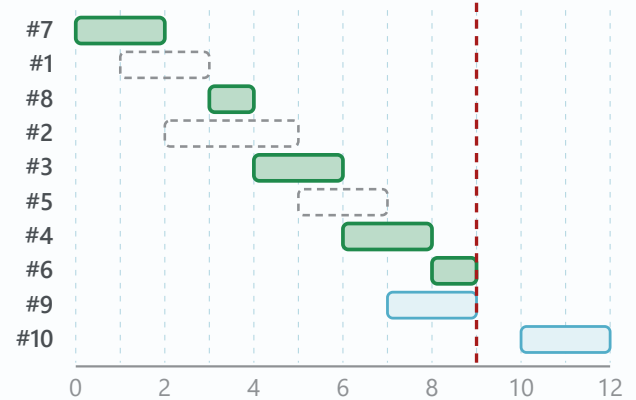
çıktı = 7 8 3 4

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     System.out.print(e[i].id + " ");
6     for (int j = 1; j < e.length; j++) {
7         if (e[j].bas >= e[i].bitis) {
8             System.out.print(e[j].id + " ");
9             i = j;
10        }
11    }

```



#6 [8, 9): başlangıç 8 ≥ 8 → **seçildi**, i = 7.

Adım 9 / 11

DEĞİŞKENLER

i = 7

j = 7

e[i].bitis = 9

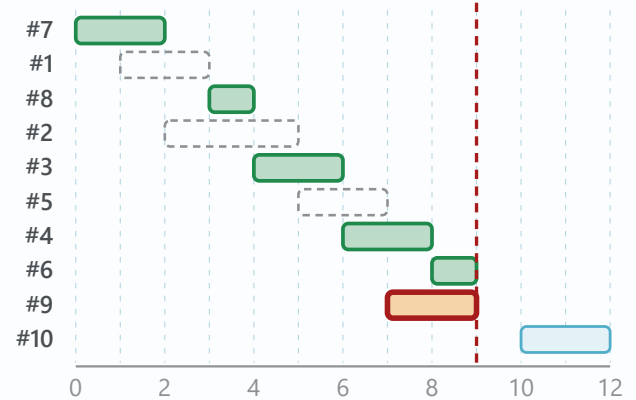
çıktı = 7 8 3 4 6

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     System.out.print(e[i].id + " ");
6     for (int j = 1; j < e.length; j++) {
7         if (e[j].bas >= e[i].bitis) {
8             System.out.print(e[j].id + " ");
9             i = j;
10        }
11    }

```



#9 [7, 9): başlangıç 7 < 9 → çakışıyor, **atlandı**.

Adım 10 / 11

DEĞİŞKENLER

i = 7

j = 8

e[i].bitis = 9

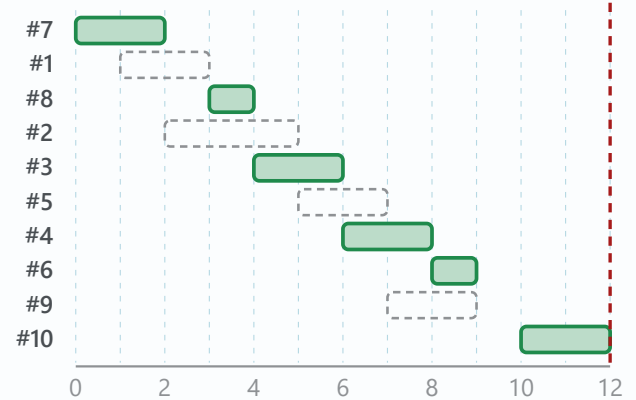
çıktı = 7 8 3 4 6

sec(e): Kodu Adım Adım İzle

```

1 static void sec(Etkinlik[] e) {
2     Arrays.sort(e, (x, y) -> x.bitis -
3         y.bitis);
4     int i = 0; // son seçilen
5     System.out.print(e[i].id + " ");
6     for (int j = 1; j < e.length; j++) {
7         if (e[j].bas >= e[i].bitis) {
8             System.out.print(e[j].id + " ");
9             i = j;
10        }
11    }

```



#10 [10, 12): başlangıç $10 \geq 9 \rightarrow$ **seçildi**, $i = 9$.

Sonuç: **6 etkinlik** (7, 8, 3, 4, 6, 10).

Adım 11 / 11

DEĞİŞKENLER

$i = 9$

$j = 9$

$e[i].bitis = 12$

çıktı = 7 8 3 4 6 10

Neden En Erken Biten?

Akla gelen diğer ölçütler en iyi sonucu **garanti etmez** (turuncu: açgözlünün seçtiği, yeşil: en iyi çözüm):

En erken başlayan
açgözlü 1 · en iyi 3



En kısa süren
açgözlü 1 · en iyi 2



"En az çakışan" ölçütünün de karşı örneği vardır.

Değiştirme argümanı

a_1 en erken biten etkinlik, O ise bir en iyi çözüm olsun. O'nun ilk etkinliği $b \neq a_1$ ise b yerine a_1 konur: $f(a_1) \leq f(b)$ olduğundan O'nun diğer etkinlikleriyle çakışma oluşmaz ve etkinlik sayısı aynı kalır.

Demek ki a_1 'i içeren bir en iyi çözüm vardır. a_1 ile çakışanlar atılınca kalan problem aynı yapıdadır (optimal alt yapı).

Son Tarihli İş Sıralama (Job Sequencing)

- Her iş **1 birim** sürer, bir **son tarihi** (deadline) ve bir **kârı** vardır.
- Kâr, iş yalnızca son tarihine kadar bitirilirse kazanılır; aynı anda tek iş yapılır.
- Amaç: **toplam kârı** en büyük yapmak.
- Açgözlü ölçüt:** işleri kâra göre **azalan** sırayla ele al; her işi son tarihinden önceki **en geç boş** zaman dilimine koy; boş dilim yoksa işi atla.
- En geç dilimi seçmek, erken dilimleri son tarihi daha yakın işlere bırakır.

İş	son tarih	kâr
1	4	20
2	1	10
3	1	40
4	1	30
5	3	50
6	2	30

Zaman: sıralama $O(n \log n)$ + slot arama $O(n \cdot d_{\max}) \rightarrow$ en kötü **$O(n^2)$** . Union-Find ile slot arama neredeyse $O(n)$ 'e iner.

isSiralais, 6): Kodu Adım Adım İzle

```
1 static int isSiralais[] is, int n) {  
    ▶ Arrays.sort(is, (a, b) -> b.kar - a.kar);  
    ▶ int[] slot = new int[n];  
    ▶ Arrays.fill(slot, -1);  
    ▶ int toplam = 0;  
    6 for (int i = 0; i < n; i++) {  
    7     int j = Math.min(n, is[i].sonTarih) -  
        1;  
    8     for (; j >= 0; j--) {  
    9         if (slot[j] == -1) {  
10             slot[j] = is[i].id;  
11             toplam += is[i].kar;  
12             break;  
13         }  
14     }  
15 }  
16 return toplam;  
17 }
```

kâra göre sıralı işler

İş 5 $t \leq 3$ kâr 50	İş 3 $t \leq 1$ kâr 40	İş 4 $t \leq 1$ kâr 30	İş 6 $t \leq 2$ kâr 30	İş 1 $t \leq 4$ kâr 20	İş 2 $t \leq 1$ kâr 10
------------------------------	------------------------------	------------------------------	------------------------------	------------------------------	------------------------------

slot (zaman dilimi)

1	2	3	4	5	6
—	—	—	—	—	—

İşler **kâra göre azalan** sıralandı (eşit kârılı İş 4 ve İş 6 özgün sırasını korur). Tüm slotlar boş (-1).

Adım 1 / 8

DEĞİŞKENLER

i = -

j = -

toplam = 0

isSiralama(is, 6): Kodu Adım Adım İzle

```
1 static int isSiralama(Is[] is, int n) {  
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);  
3     int[] slot = new int[n];  
4     Arrays.fill(slot, -1);  
5     int toplam = 0;  
6     for (int i = 0; i < n; i++) {  
7         ▶ int j = Math.min(n, is[i].sonTarih) -  
8           1;  
9         ▶ for (; j >= 0; j--) {  
10            ▶ if (slot[j] == -1) {  
11                ▶ slot[j] = is[i].id;  
12                ▶ toplam += is[i].kar;  
13            }  
14        }  
15    }  
16    return toplam;  
17 }
```

kâra göre sıralı işler

İş 5 $t \leq 3$ kâr 50	İş 3 $t \leq 1$ kâr 40	İş 4 $t \leq 1$ kâr 30	İş 6 $t \leq 2$ kâr 30	İş 1 $t \leq 4$ kâr 20	İş 2 $t \leq 1$ kâr 10
------------------------------	------------------------------	------------------------------	------------------------------	------------------------------	------------------------------

slot (zaman dilimi)

1	2	3	4	5	6
—	—	İş 5	—	—	—

İş 5 (son tarih 3, kâr 50): en geç uygun slot **3** boş → yerleştirildi. toplam = 50.

Adım 2 / 8

DEĞİŞKENLER

i = 0

j = 2

toplam = 50

isSiralama(is, 6): Kodu Adım Adım İzle

```
1 static int isSiralama(Is[] is, int n) {  
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);  
3     int[] slot = new int[n];  
4     Arrays.fill(slot, -1);  
5     int toplam = 0;  
6     for (int i = 0; i < n; i++) {  
7         ▶ int j = Math.min(n, is[i].sonTarih) -  
8           1;  
9         ▶ for (; j >= 0; j--) {  
10            ▶ if (slot[j] == -1) {  
11                ▶ slot[j] = is[i].id;  
12                ▶ toplam += is[i].kar;  
13            }  
14        }  
15    }  
16    return toplam;  
17 }
```

kâra göre sıralı işler

İş 5 $t \leq 3$ kâr 50	İş 3 $t \leq 1$ kâr 40	İş 4 $t \leq 1$ kâr 30	İş 6 $t \leq 2$ kâr 30	İş 1 $t \leq 4$ kâr 20	İş 2 $t \leq 1$ kâr 10
------------------------------	------------------------------	------------------------------	------------------------------	------------------------------	------------------------------

slot (zaman dilimi)

1	2	3	4	5	6
İş 3	—	İş 5	—	—	—

İş 3 (son tarih 1, kâr 40): en geç uygun slot **1** boş → yerleştirildi. toplam = 90.

Adım 3 / 8

DEĞİŞKENLER

i = 1

j = 0

toplam = 90

isSiralais, 6): Kodu Adım Adım İzle

```
1 static int isSiralais[] is, int n) {  
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);  
3     int[] slot = new int[n];  
4     Arrays.fill(slot, -1);  
5     int toplam = 0;  
6     for (int i = 0; i < n; i++) {  
7         int j = Math.min(n, is[i].sonTarih) -  
8             1;  
9         for (; j >= 0; j--) {  
10             if (slot[j] == -1) {  
11                 slot[j] = is[i].id;  
12                 toplam += is[i].kar;  
13                 break;  
14             }  
15         }  
16     }  
17     return toplam;  
}
```

kâra göre sıralı işler

İş 5 $t \leq 3$ kâr 50	İş 3 $t \leq 1$ kâr 40	İş 4 $t \leq 1$ kâr 30	İş 6 $t \leq 2$ kâr 30	İş 1 $t \leq 4$ kâr 20	İş 2 $t \leq 1$ kâr 10
------------------------------	------------------------------	------------------------------	------------------------------	------------------------------	------------------------------

slot (zaman dilimi)

1	2	3	4	5	6
İş 3	—	İş 5	—	—	—

İş 4 (son tarih 1, kâr 30): slot 1 **dolu** (İş 3) ve daha erken slot yok → iş **atlanır**.

Adım 4 / 8

DEĞİŞKENLER

i = 2

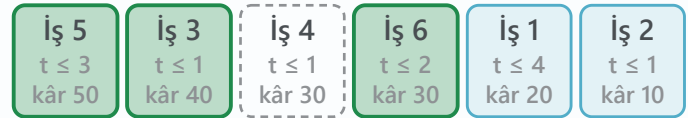
j = 0

toplam = 90

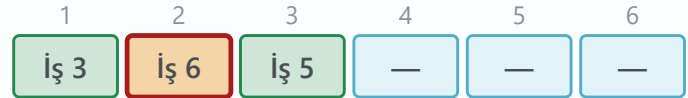
isSiralais, 6): Kodu Adım Adım İzle

```
1 static int isSiralais[] is, int n) {  
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);  
3     int[] slot = new int[n];  
4     Arrays.fill(slot, -1);  
5     int toplam = 0;  
6     for (int i = 0; i < n; i++) {  
7         int j = Math.min(n, is[i].sonTarih) -  
8             1;  
9         for (; j >= 0; j--) {  
10             if (slot[j] == -1) {  
11                 slot[j] = is[i].id;  
12                 toplam += is[i].kar;  
13                 break;  
14             }  
15         }  
16     }  
17     return toplam;  
18 }
```

kâra göre sıralı işler



slot (zaman dilimi)



İş 6 (son tarih 2, kâr 30): en geç uygun slot **2** boş → yerleştirildi. toplam = 120.

Adım 5 / 8

DEĞİŞKENLER

i = 3

j = 1

toplam = 120

isSiralama(is, 6): Kodu Adım Adım İzle

```
1 static int isSiralama(Is[] is, int n) {  
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);  
3     int[] slot = new int[n];  
4     Arrays.fill(slot, -1);  
5     int toplam = 0;  
6     for (int i = 0; i < n; i++) {  
7         int j = Math.min(n, is[i].sonTarih) -  
8             1;  
9         for (; j >= 0; j--) {  
10             if (slot[j] == -1) {  
11                 slot[j] = is[i].id;  
12                 toplam += is[i].kar;  
13             }  
14         }  
15     }  
16     return toplam;  
17 }
```

kâra göre sıralı işler



slot (zaman dilimi)



İş 1 (son tarih 4, kâr 20): en geç uygun slot 4 boş → yerleştirildi. toplam = 140.

Adım 6 / 8

DEĞİŞKENLER

i = 4

j = 3

toplam = 140

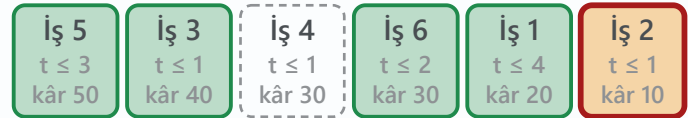
isSiralama(is, 6): Kodu Adım Adım İzle

```

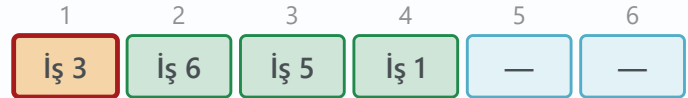
1 static int isSiralama(Is[] is, int n) {
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);
3     int[] slot = new int[n];
4     Arrays.fill(slot, -1);
5     int toplam = 0;
6     for (int i = 0; i < n; i++) {
7         int j = Math.min(n, is[i].sonTarih) -
            1;
8         for (; j >= 0; j--) {
9             ► if (slot[j] == -1) {
10                 slot[j] = is[i].id;
11                 toplam += is[i].kar;
12                 break;
13             }
14         }
15     }
16     return toplam;
17 }

```

kâra göre sıralı işler



slot (zaman dilimi)



İş 2 (son tarih 1, kâr 10): slot 1 **dolu** (İş 3) ve daha erken slot yok → iş **atlanır**.

Adım 7 / 8

DEĞİŞKENLER

i = 5

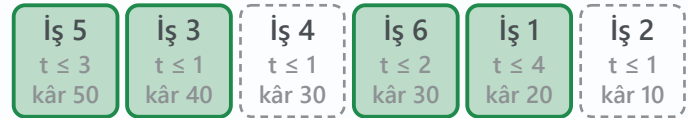
j = 0

toplam = 140

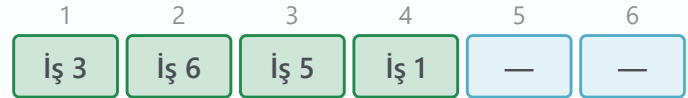
isSiralais, 6): Kodu Adım Adım İzle

```
1 static int isSiralais[] is, int n) {  
2     Arrays.sort(is, (a, b) -> b.kar - a.kar);  
3     int[] slot = new int[n];  
4     Arrays.fill(slot, -1);  
5     int toplam = 0;  
6     for (int i = 0; i < n; i++) {  
7         int j = Math.min(n, is[i].sonTarih) -  
8             1;  
9         for (; j >= 0; j--) {  
10             if (slot[j] == -1) {  
11                 slot[j] = is[i].id;  
12                 toplam += is[i].kar;  
13                 break;  
14             }  
15         }  
16     }  
17     return toplam;  
18 }
```

kâra göre sıralı işler



slot (zaman dilimi)



Sonuç: slotlar 1–4 = İş 3, İş 6, İş 5, İş 1 → **toplam kâr 140**. İş 4 ve İş 2 yapılamaz.

Adım 8 / 8

DEĞİŞKENLER

i = 6

j = -

toplam = 140

SIRT ÇANTASI

Kesirli Sırt Çantası (Fractional Knapsack)

Kapasite **W = 50**. Nesneler bölünebiliyorsa açgözlü ölçüt: **kâr / ağırlık oranı** en büyük olanı al.

Nesne	kâr	ağırlık	oran
1	60	10	6
2	100	20	5
3	120	30	4

Kesirli: 1 + 2 tümüyle, 3'ün 20/30'u $\rightarrow 60 + 100 + 80 = \mathbf{240}$ (en iyi). Zaman $O(n \log n)$.

0/1 sırt çantasında başarısız

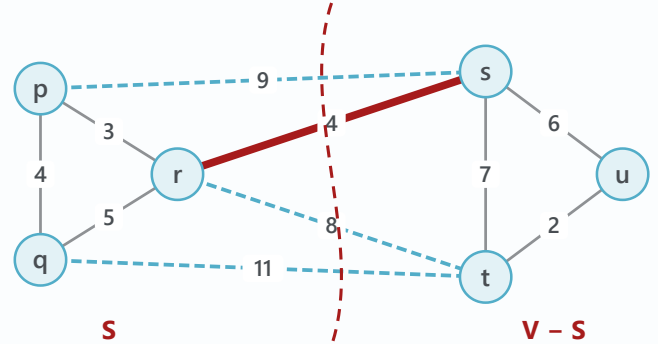
Nesneler bölünemiyorsa aynı ölçüt 1 ve 2'yi alır: **160** (30 kg; 3 sığmaz).

En iyi çözüm $2 + 3 = \mathbf{220}$. 0/1 sırt çantası açgözlü seçim özelliğini sağlamaz; **dinamik programlama** ile $O(n \cdot W)$ zamanda çözülür.

MINİMUM KAPSAYAN AĞAÇ

Minimum Kapsayan Ağaç ve Kesim Özelliği

- **Minimum kapsayan ağaç** (minimum spanning tree, MST): bağlı, ağırlıklı, yönsüz bir çizgenin tüm düğümlerini **en az toplam ağırlıklı**la bağlayan ağaç; $V - 1$ kenar.
- **Kesim özelliği** (cut property): düğümler S ve $V - S$ diye ikiye ayrılırsa, bu kesimi geçen **en hafif kenar** bir MST'de yer alır.
- Borůvka, Kruskal ve Prim bu özelliği farklı biçimlerde kullanan **açgözlü** algoritmalarıdır.
- Ağırlıklar farklıysa MST **tektir**.



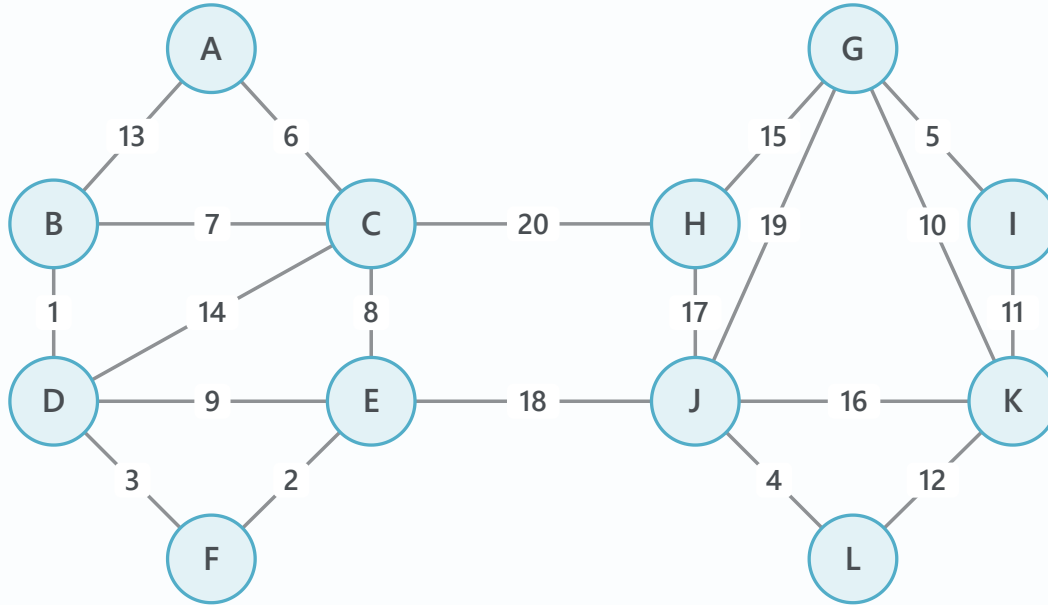
Kesimi geçen kenarlar: 9, 4, 8, 11. En hafifi r-s (4) MST'dedir.

Borůvka'nın Algoritması

- Çizge teorisinde kullanılan bir **minimum kapsayan ağaç** algoritmasıdır (Otakar Borůvka, 1926).
 - Başlangıçta her düğüm bir **alt ağaç** (bileşen) oluşturur.
 - Her turda, **her alt ağacın** dışarı çıkan **en az maliyetli kenarı** bulunur ve bu kenarlar eklenerek alt ağaçlar birleştirilir.
 - **Açgözlü:** her bileşen için yerel olarak en iyi kenarı seçer (kesim özelliği: bileşen ile geri kalan arasındaki en hafif kenar).
 - Her turda en az bir kenar eklenir ve ağaç büyür.
1. **Adım 1:** Her düğüm bir alt ağaç olarak başlar.
 2. **Adım 2:** Her alt ağaçtan, dışarı çıkan en az maliyetli kenar seçilir.
 3. **Adım 3:** Seçilen kenarlarla alt ağaçlar birleştirilir.
 4. **Adım 4:** Alt ağaçlar tek bir ağaçta toplanana kadar tekrarlanır.

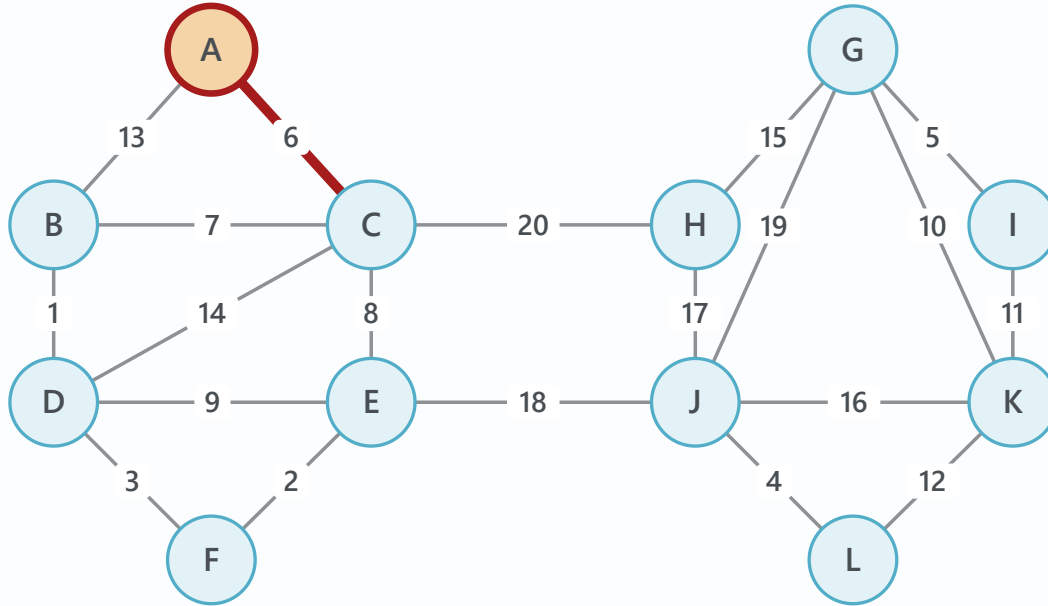
Çalışma zamanı: **$O(E \log V)$** — E kenar, V düğüm sayısı.

Boruvka Adım Adım



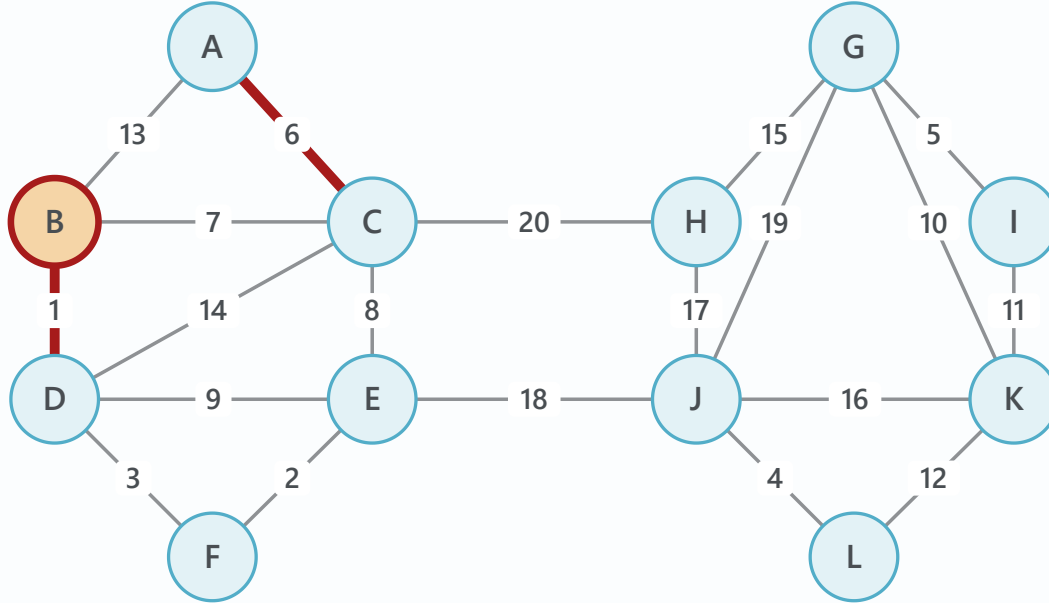
Başlangıç: her düğüm tek başına bir alt ağaç → **12 bileşen**, 0 kenar.

Boruvka Adım Adım



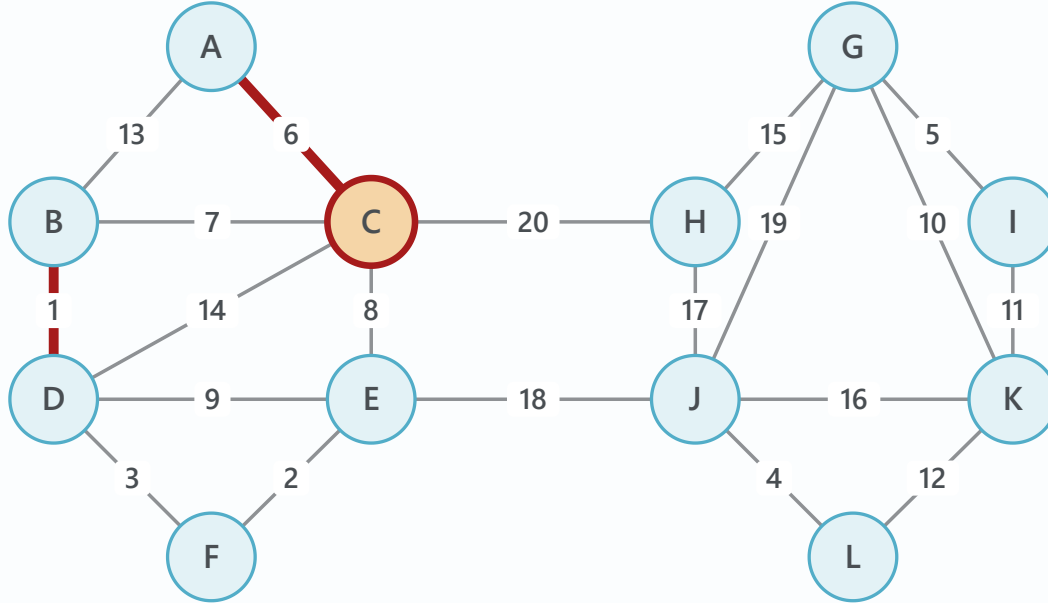
Tur 1 · düğüm **A**: dışarı çıkan en ucuz kenar **A–C (6)**.

Boruvka Adım Adım



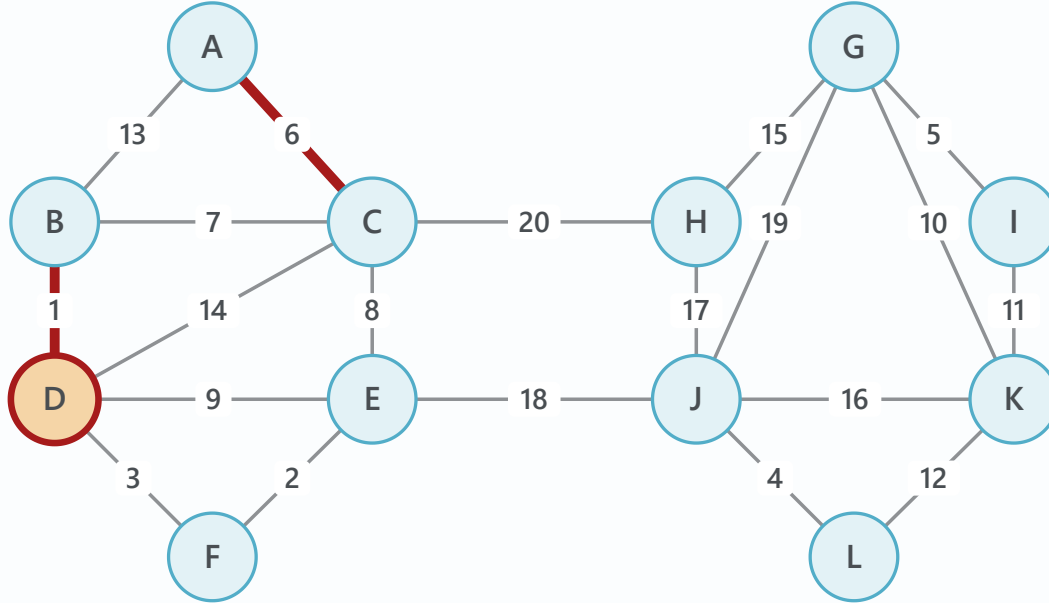
Tur 1 · düğüm **B**: dışarı çıkan en ucuz kenar **B-D (1)**.

Boruvka Adım Adım



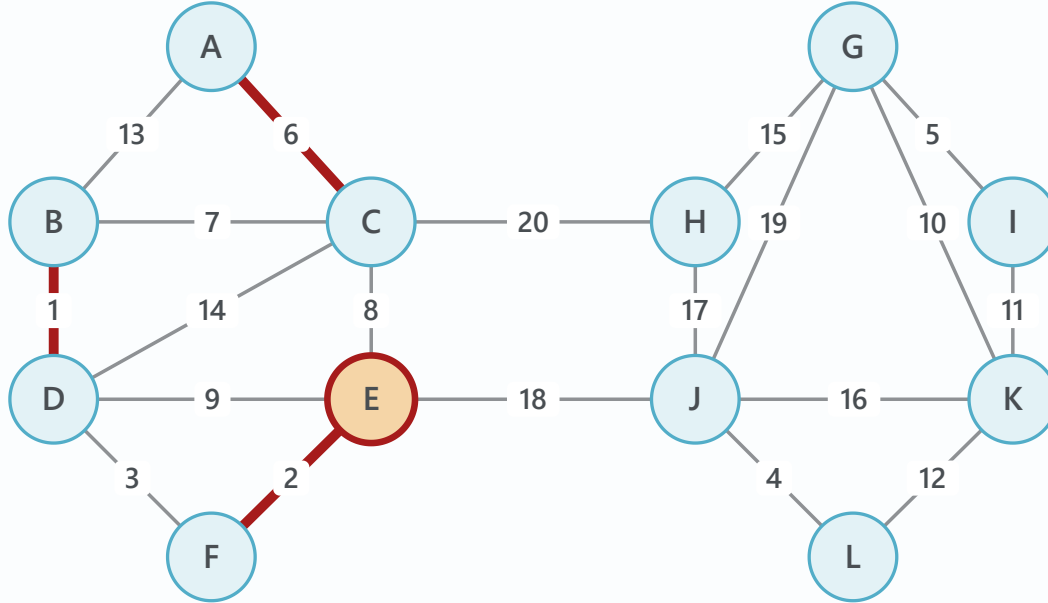
Tur 1 · düğüm **C**: dışarı çıkan en ucuz kenar **A-C (6)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



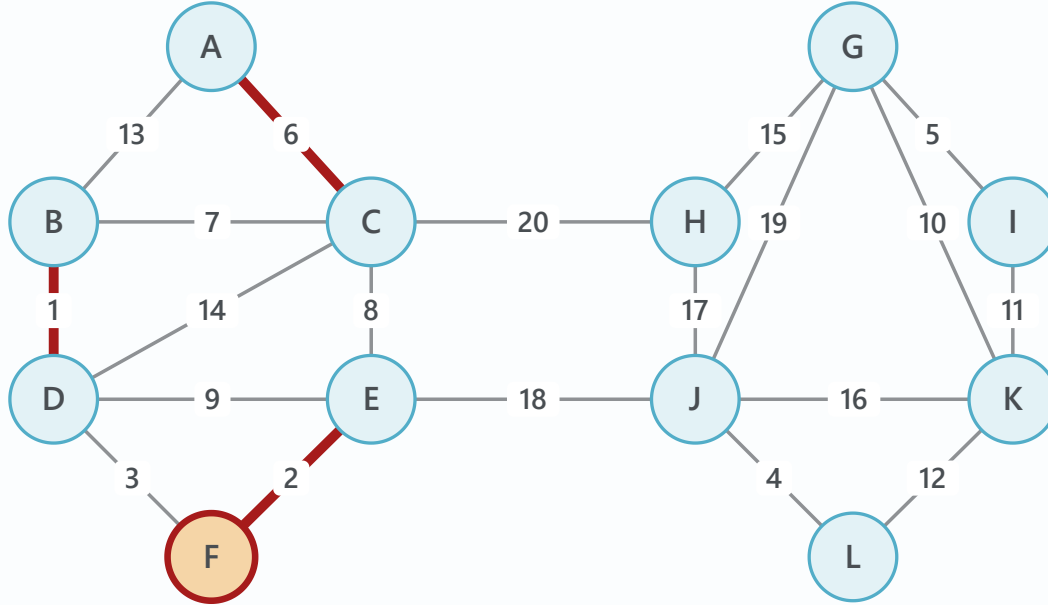
Tur 1 · düğüm **D**: dışarı çıkan en ucuz kenar **B-D (1)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



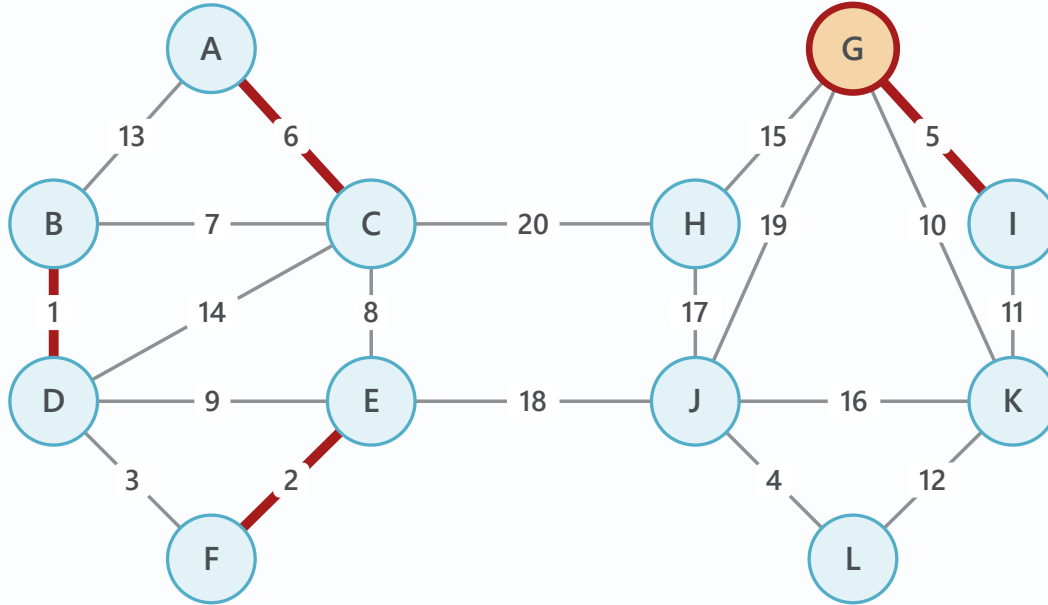
Tur 1 · düğüm **E**: dışarı çıkan en ucuz kenar **E-F (2)**.

Boruvka Adım Adım



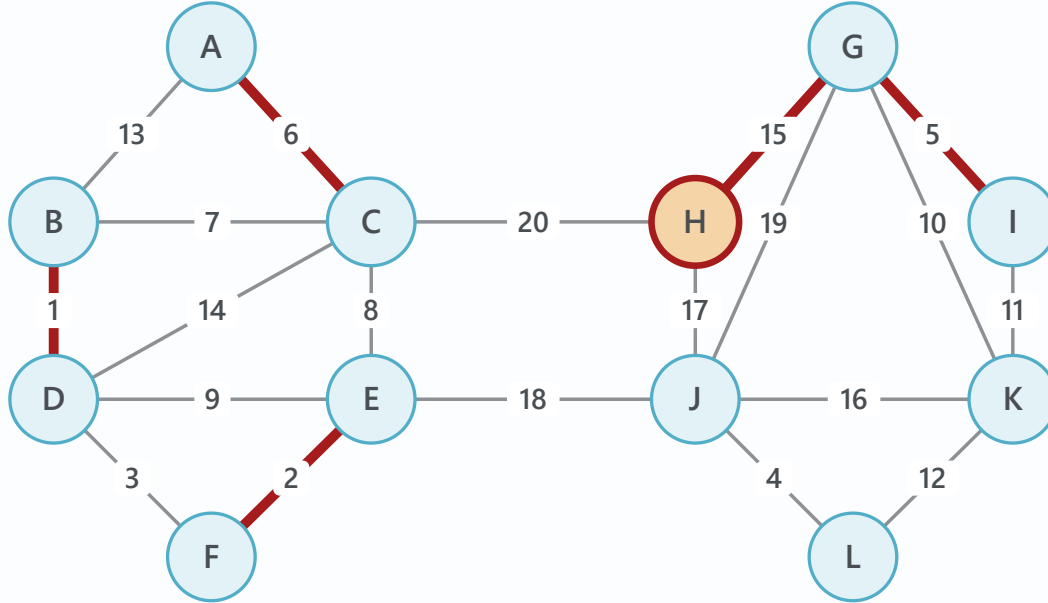
Tur 1 · düğüm **F**: dışarı çıkan en ucuz kenar **E-F (2)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



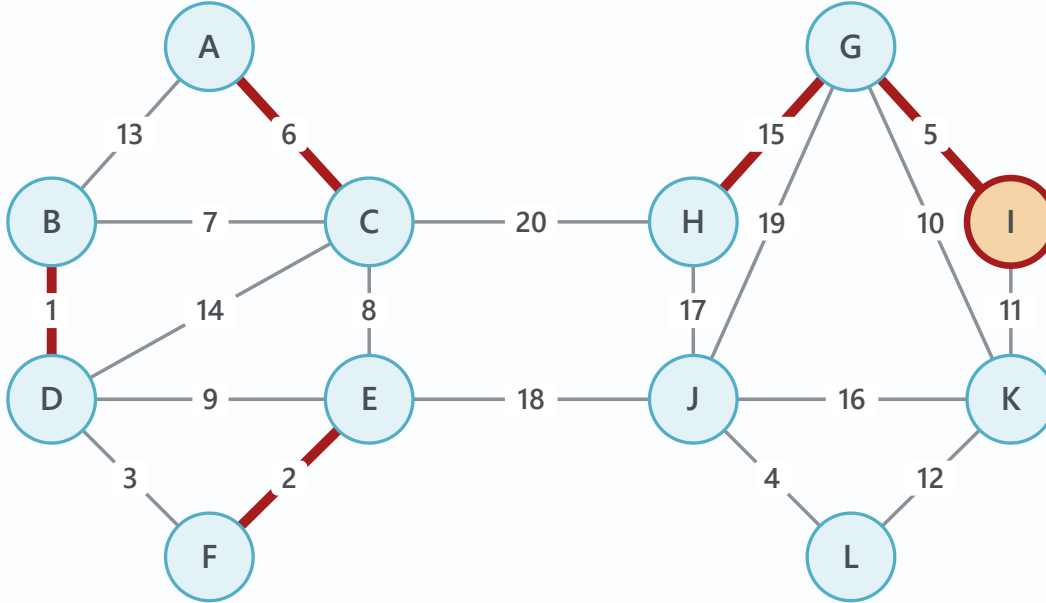
Tur 1 · düğüm **G**: dışarı çıkan en ucuz kenar **G-I (5)**.

Boruvka Adım Adım



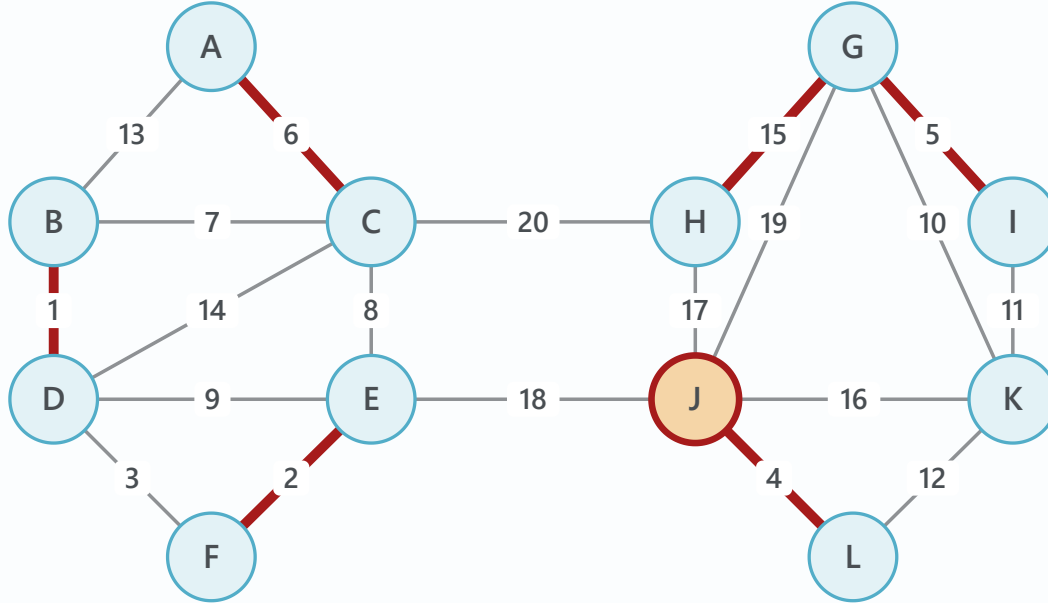
Tur 1 · düğüm **H**: dışarı çıkan en ucuz kenar **G-H (15)**.

Borůvka Adım Adım



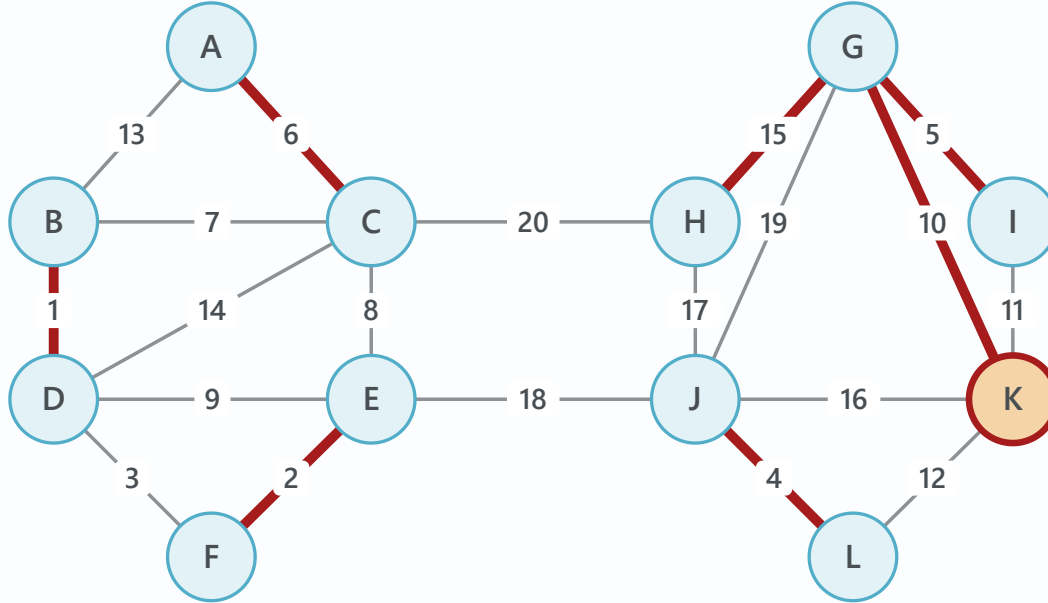
Tur 1 · düğüm I: dışarı çıkan en ucuz kenar **G-I (5)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



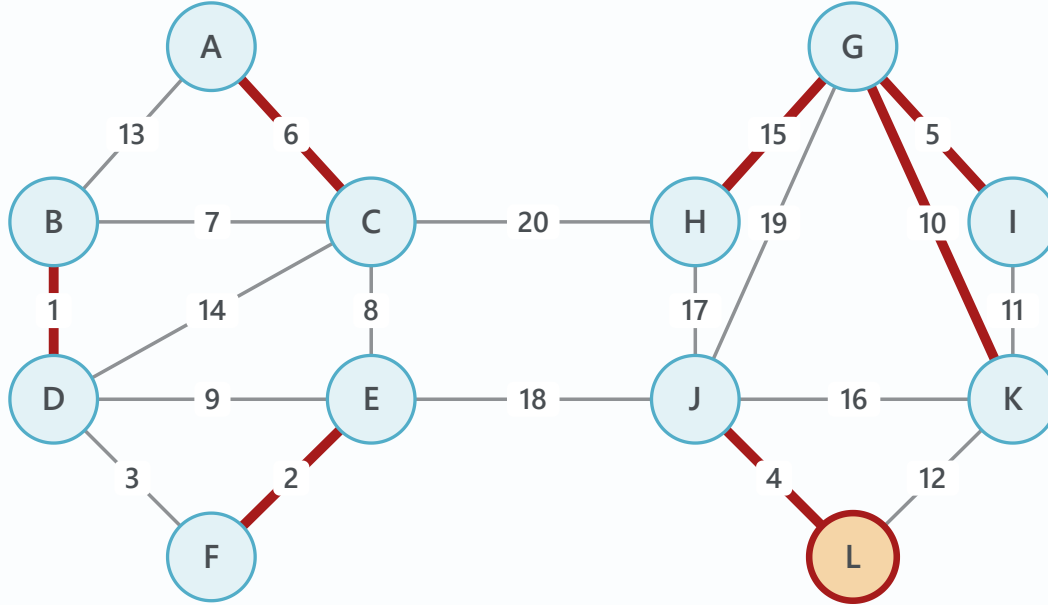
Tur 1 · düğüm J: dışarı çıkan en ucuz kenar J-L (4).

Boruvka Adım Adım



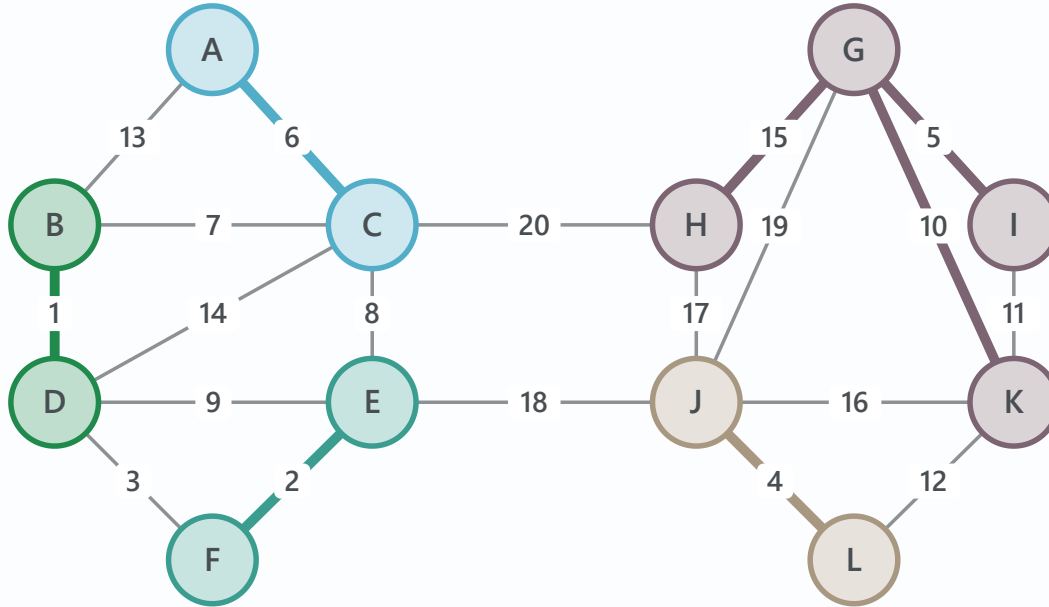
Tur 1 · düğüm **K**: dışarı çıkan en ucuz kenar **G–K (10)**.

Borůvka Adım Adım



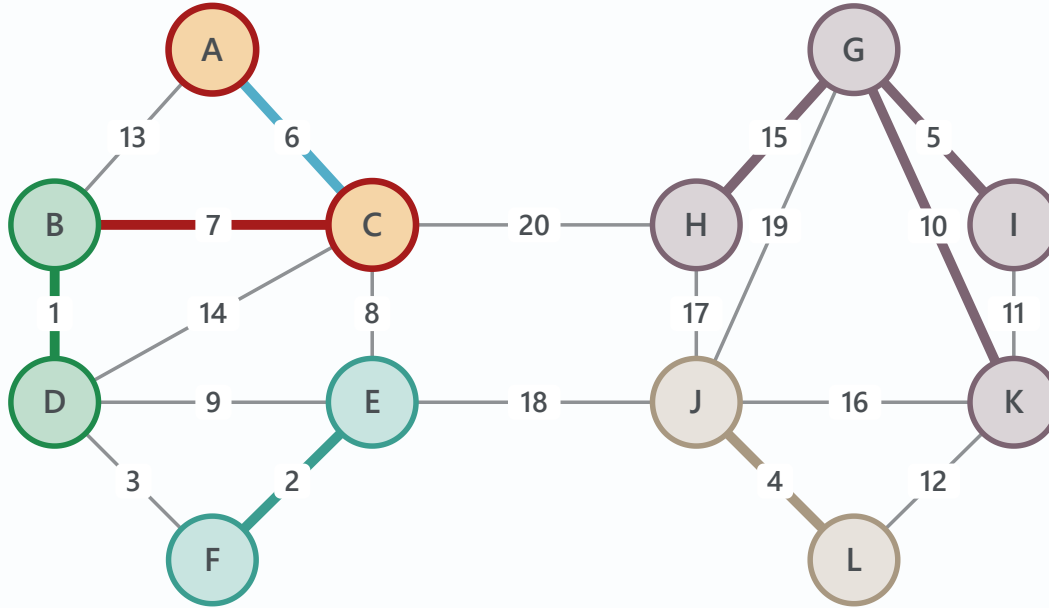
Tur 1 · düğüm **L**: dışarı çıkan en ucuz kenar **J-L (4)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



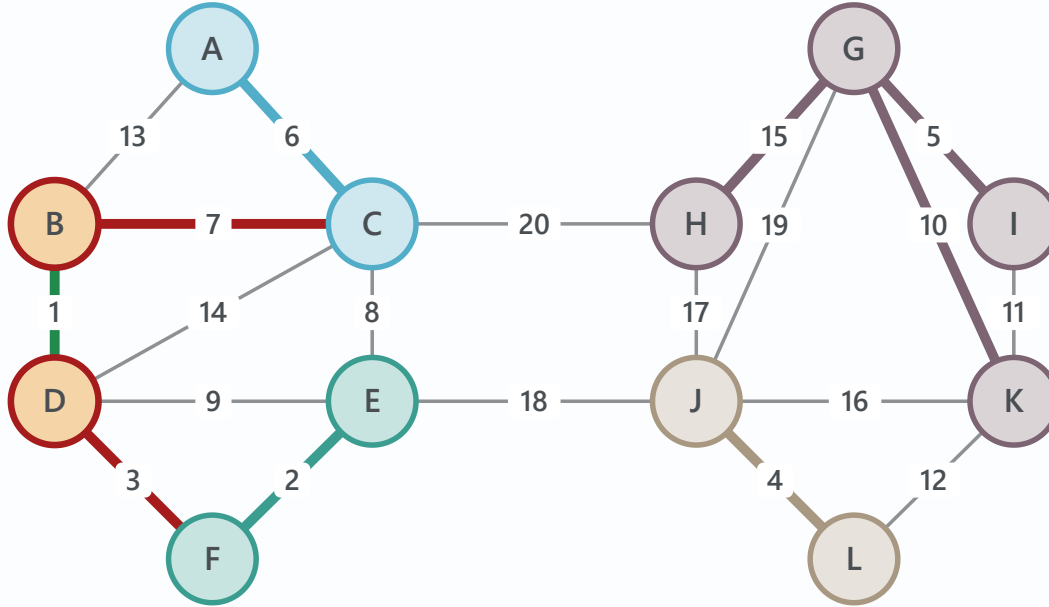
Tur 1 sonu: seçilen 7 farklı kenar eklendi → **5 bileşen**: {A,C}, {B,D}, {E,F}, {G,H,I,K}, {J,L}. Ağırlık: 43.

Boruvka Adım Adım



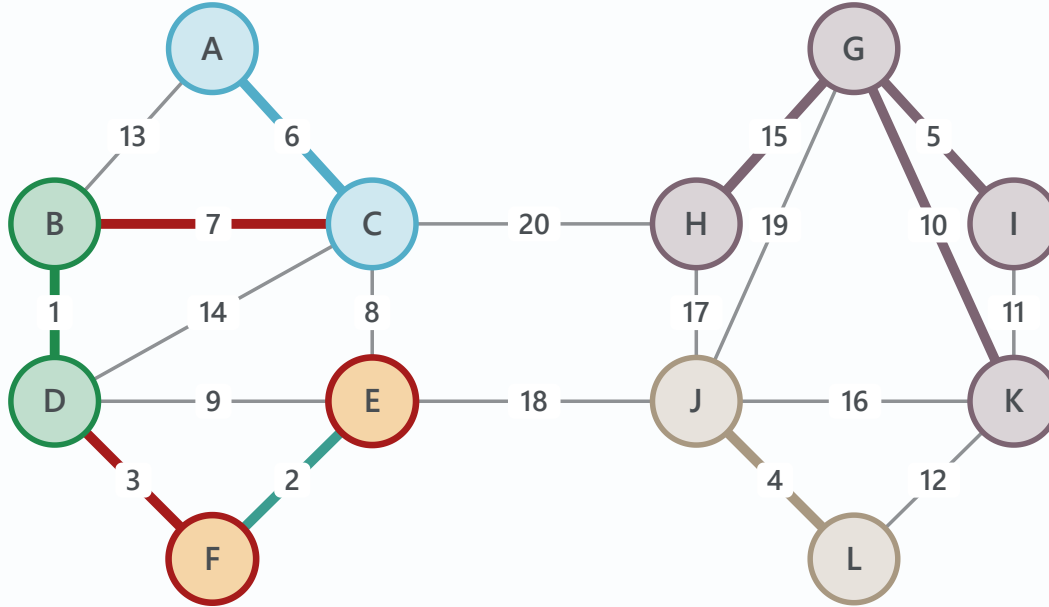
Tur 2 · bileşen {A, C}: dışarı çıkan en ucuz kenar **B–C (7)**.

Boruvka Adım Adım



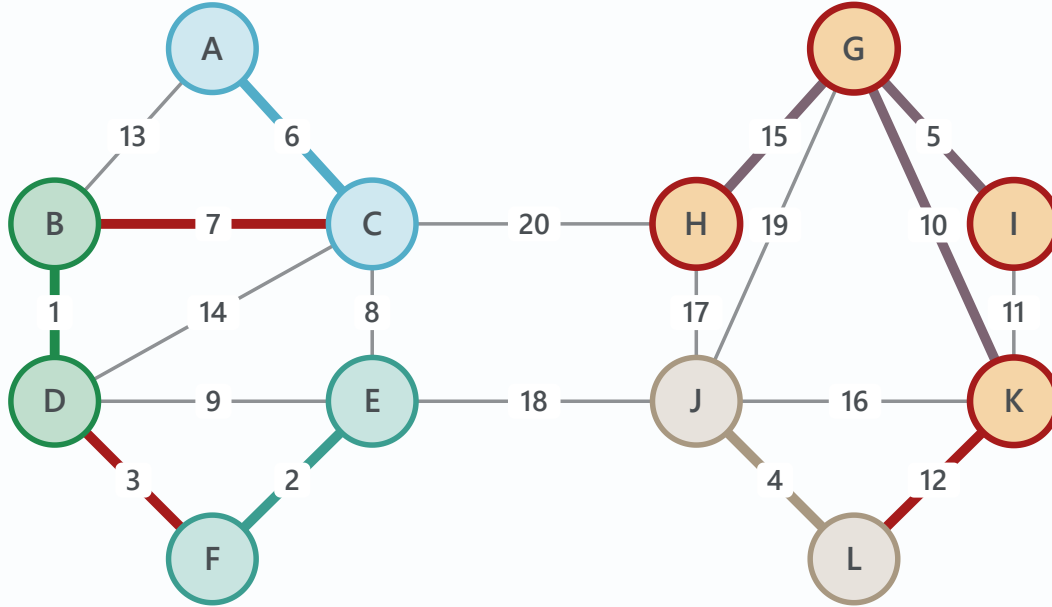
Tur 2 · bileşen {**B**, **D**}: dışarı çıkan en ucuz kenar **D-F (3)**.

Boruvka Adım Adım



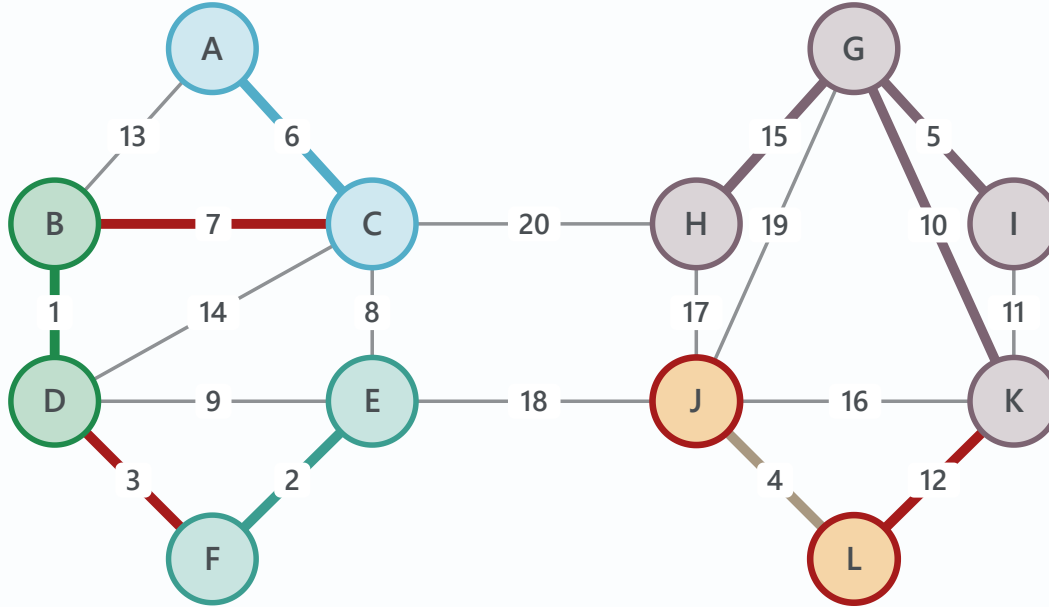
Tur 2 · bileşen {E, F}: dışarı çıkan en ucuz kenar **D-F (3)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



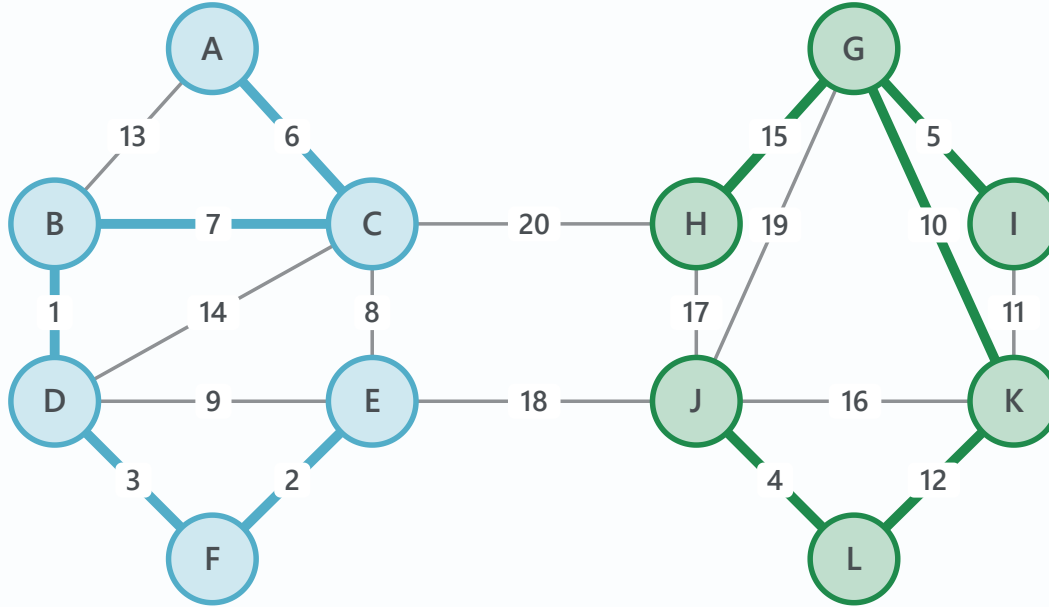
Tur 2 · bileşen {G, H, I, K}: dışarı çıkan en ucuz kenar K-L (12).

Boruvka Adım Adım



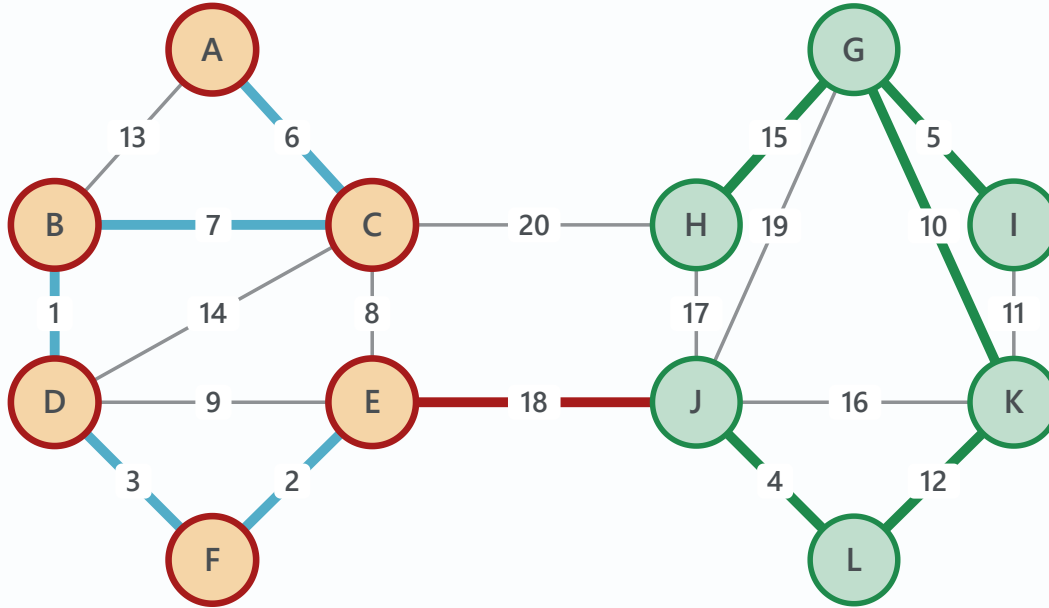
Tur 2 · bileşen {J, L}: dışarı çıkan en ucuz kenar **K-L (12)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



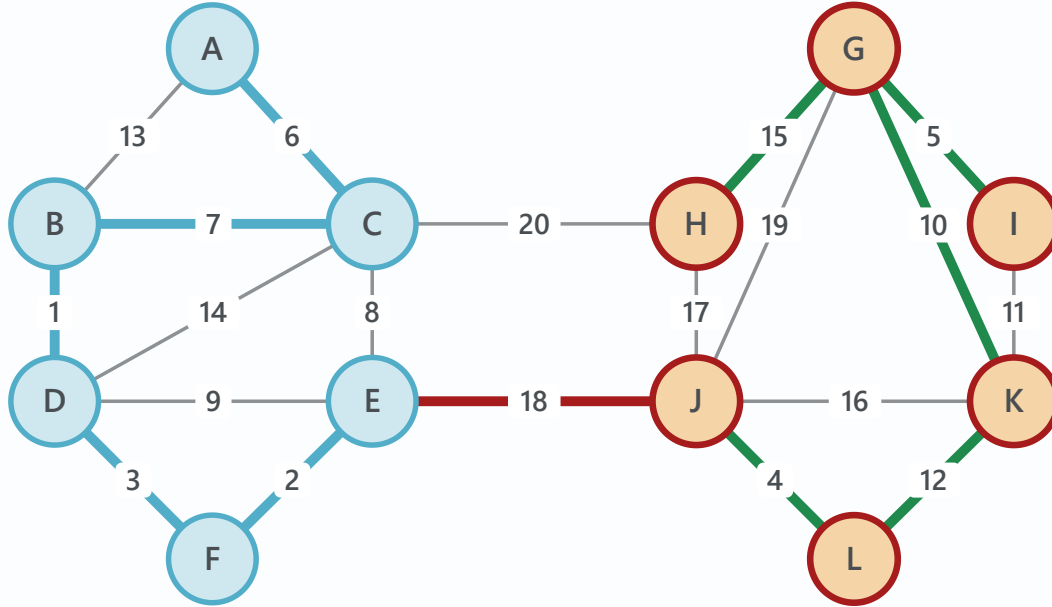
Tur 2 sonu: seçilen 3 farklı kenar eklendi → **2 bileşen**: {A,B,C,D,E,F}, {G,H,I,J,K,L}. Ağırlık: 65.

Boruvka Adım Adım



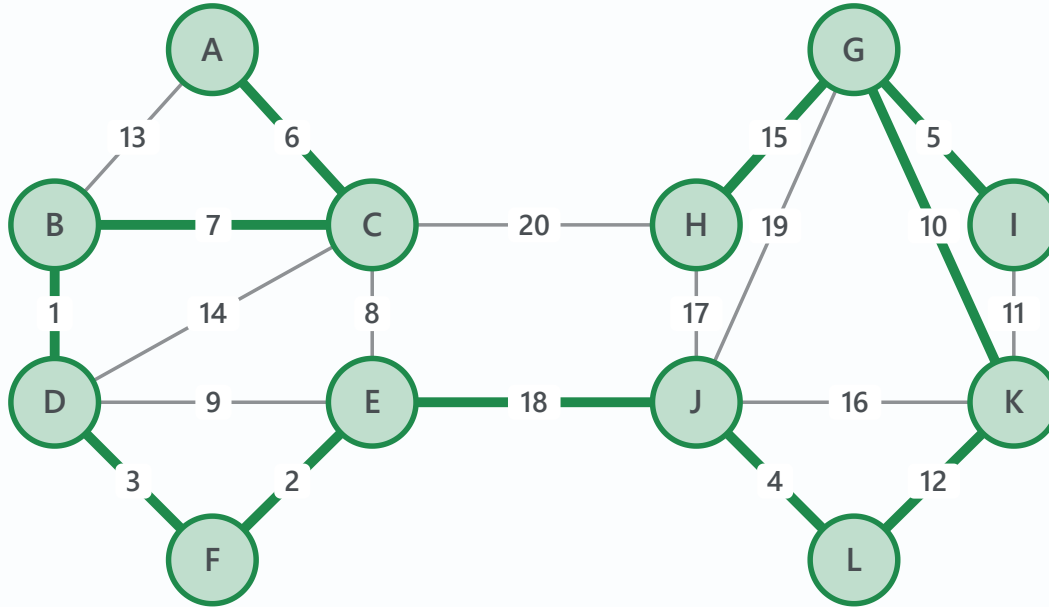
Tur 3 · bileşen {A, B, C, D, E, F}: dışarı çıkan en ucuz kenar E-J (18).

Boruvka Adım Adım



Tur 3 · bileşen {G, H, I, J, K, L}: dışarı çıkan en ucuz kenar **E-J (18)** — bu kenar zaten seçilmişti.

Boruvka Adım Adım



Tur 3 sonu: tek bileşen kaldı. MST: **11 kenar** ($V - 1$), toplam ağırlık **83**. Yalnızca **3 tur** yetti.

Boruvka: Java Kodu

```
static int boruvka(int n, int[][] kenar) {
    AyrikKume kume = new AyrikKume(n);
    int bilesen = n, toplam = 0;
    while (bilesen > 1) {
        int[] enUcuz = new int[n];    // kök -> kenar no
        Arrays.fill(enUcuz, -1);
        for (int i = 0; i < kenar.length; i++) {
            int a = kume.bul(kenar[i][0]), b = kume.bul(kenar[i][1]);
            if (a == b) continue;    // bileşen içi kenar
            int w = kenar[i][2];
            if (enUcuz[a] < 0 || w < kenar[enUcuz[a]][2]) enUcuz[a] = i;
            if (enUcuz[b] < 0 || w < kenar[enUcuz[b]][2]) enUcuz[b] = i;
        }
        for (int c = 0; c < n; c++) {    // seçilenleri ekle
            if (enUcuz[c] < 0) continue;
            int[] k = kenar[enUcuz[c]];
            if (kume.bul(k[0]) != kume.bul(k[1])) {    // iki kez eklenmesin
                kume.birlestir(k[0], k[1]);
                toplam += k[2]; bilesen--;
            }
        }
    }
    return toplam;
}
```

Borůvka: Analiz

Tur sayısı $\leq \log_2 V$

Her bileşen en az bir başka bileşenle birleşir; bu yüzden her turda bileşen sayısı **en az yarıya** iner. Örnekte $12 \rightarrow 5 \rightarrow 2 \rightarrow 1$: turlarda 7, 3 ve 1 kenar eklendi.

Bir tur = $O(E)$

Her kenar bir kez taranır (Union-Find işlemleri neredeyse sabit). Toplam: **$O(E \log V)$** .

Eşit ağırlıklar

İki bileşen eşit ağırlıklı farklı kenarlar seçerse döngü oluşabilir. Eşitlik **tutarlı** bozulmalı: örn. (ağırlık, kenar no) çifti karşılaştırılır.

Paralleleştirilebilir

Bir turdaki bileşenler birbirinden bağımsız en ucuz kenarını arar; bu yüzden paralel ve dağıtık MST algoritmalarının temelidir.

Union-Find (Ayrık Kümeler) Veri Yapısı

```
class AyrikKume {
    int[] ebeveyn, rutbe;
    AyrikKume(int n) {
        ebeveyn = new int[n];
        rutbe = new int[n];
        for (int i = 0; i < n; i++) ebeveyn[i] = i;
    }
    int bul(int x) {                // yol sıkıştırma
        if (ebeveyn[x] != x)
            ebeveyn[x] = bul(ebeveyn[x]);
        return ebeveyn[x];
    }
    void birlestir(int x, int y) {
        int kx = bul(x), ky = bul(y);
        if (kx == ky) return;
        if (rutbe[kx] < rutbe[ky]) {
            int t = kx; kx = ky; ky = t;
        }
        ebeveyn[ky] = kx;           // rütbeğe göre
        if (rutbe[kx] == rutbe[ky]) rutbe[kx]++;
    }
}
```

- **Amaç:** küme birleştirme ve küme bulma işlemlerini hızlı yapmak.
- **Union(x, y)** → **birlestir** : x ve y'nin kümelerini birleştirir (bir kök değerinin çocuğu olur).
- **Find(x)** → **bul** : x'in bulunduğu kümenin **kök** ögesini bulur.
- Her küme bir ağaçtır; **ebeveyn[kök] = kök**.
- **Yol kısaltma + rütbe (rank) göre birleştirme** ile işlem başına amortize **$O(\alpha(n))$** ; α pratikte ≤ 4 .

Kruskal Algoritması

- **Amaç:** ağırlıklı, bağlı bir çizgenin minimum kapsayan ağacını bulmak.
 - **Açgözlü yaklaşım:** her adımda kalan kenarlar arasından **en küçük ağırlıklı** olanı seçerek ilerler.
 - Döngü denetimi **Union-Find** ile yapılır: uçların kökleri aynıysa kenar döngü oluşturur.
1. Tüm kenarları ağırlıklarına göre **sırala**.
 2. En küçük ağırlıklı kenarı seç.
 3. Kenar mevcut ormanda **döngü oluşturmuyorsa** ekle (iki kümeyi birleştir), oluşturunca at.
 4. $V - 1$ kenar eklenene (ya da kenarlar bitene) kadar tekrarla.

Doğruluk: eklenen kenar, iki farklı bileşeni ayıran kesimin en hafif kenarıdır (kesim özelliği).

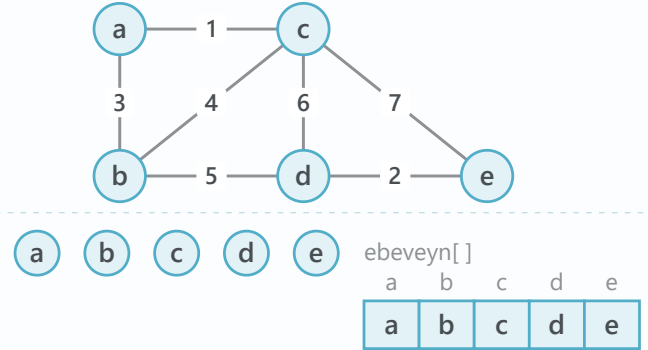
Zaman: sıralama $O(E \log E) = O(E \log V) + E \cdot O(\alpha(V))$.

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Kenarlar ağırlığa göre sıralandı: 1, 2, 3, 4, 5, 6, 7. Her düğüm kendi kümesinin kökü (ebeveyn[x] = x).

Adım 1 / 11

DEĞİŞKENLER

kenar = -

bul(u) = -

bul(v) = -

toplam = 0

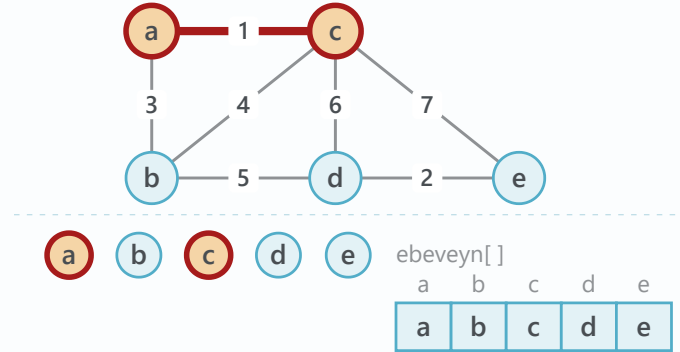
sayac = 0

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         ► if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Kenar **a-c (1)**: $\text{bul}(a) = a$, $\text{bul}(c) = c \rightarrow$ kökler **farklı**, döngü oluşmaz.

Adım 2 / 11

DEĞİŞKENLER

kenar = (a, c, 1)

bul(u) = a

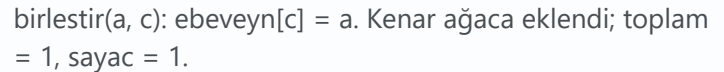
bul(v) = c

toplam = 0

sayac = 0

```

graph LR
    a((a)) --- 3 --- b((b))
    a --- 1 --- c((c))
    b --- 4 --- d((d))
    b --- 5 --- e((e))
    c --- 6 --- d
    c --- 7 --- e
    d --- 2 --- e
    style a fill:#add8e6,stroke:#000,stroke-width:1px
    style b fill:#add8e6,stroke:#000,stroke-width:1px
    style c fill:#add8e6,stroke:#000,stroke-width:1px
    style d fill:#add8e6,stroke:#000,stroke-width:1px
    style e fill:#add8e6,stroke:#000,stroke-width:1px
    linkStyle 1 stroke:#008000,stroke-width:2px
  
```



DEĞİŞKENLER

$$\text{bul}(v) = \mathbf{c}$$

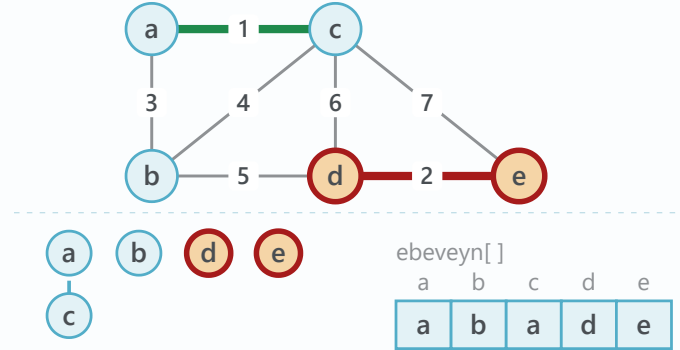
```
sayac = 1
```

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         ► if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Kenar **d-e (2)**: $\text{bul}(d) = d$, $\text{bul}(e) = e \rightarrow$ kökler **farklı**, döngü oluşmaz.

Adım 4 / 11

DEĞİŞKENLER

kenar = (d, e, 2)

bul(u) = d

bul(v) = e

toplam = 1

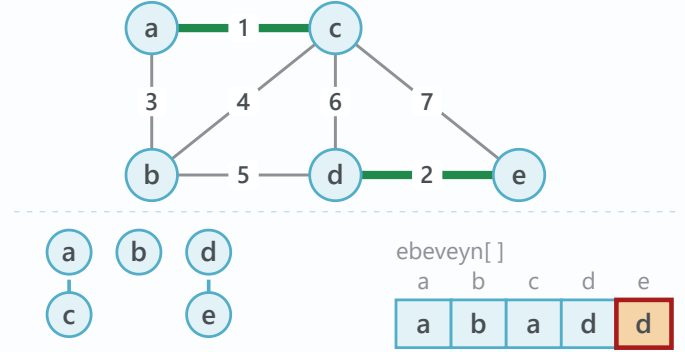
sayac = 1

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2   Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3   AyrikKume kume = new AyrikKume(n);
4   int toplam = 0, sayac = 0;
5   for (int[] k : kenar) {
6     int u = k[0], v = k[1], w = k[2];
7     if (kume.bul(u) != kume.bul(v)) {
8       ▶ kume.birlestir(u, v);
9       ▶ toplam += w;
10      ▶ sayac++;
11      if (sayac == n - 1) break;
12    }
13  }
14  return toplam;
15 }

```



birlestir(d, e): ebeveyn[e] = d. Kenar ağaca eklendi; toplam = 3, sayac = 2.

Adım 5 / 11

DEĞİŞKENLER

kenar = (d, e, 2)

bul(u) = d

bul(v) = e

toplam = 3

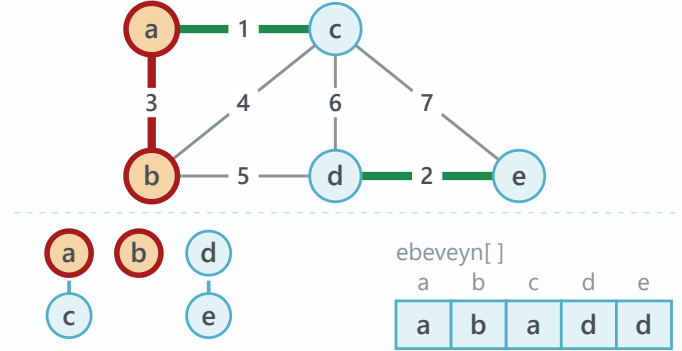
sayac = 2

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         ► if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Kenar **a-b (3)**: $\text{bul}(a) = a$, $\text{bul}(b) = b \rightarrow$ kökler **farklı**,
döngü oluşmaz.

Adım 6 / 11

DEĞİŞKENLER

kenar = (a, b, 3)

bul(u) = a

bul(v) = b

toplam = 3

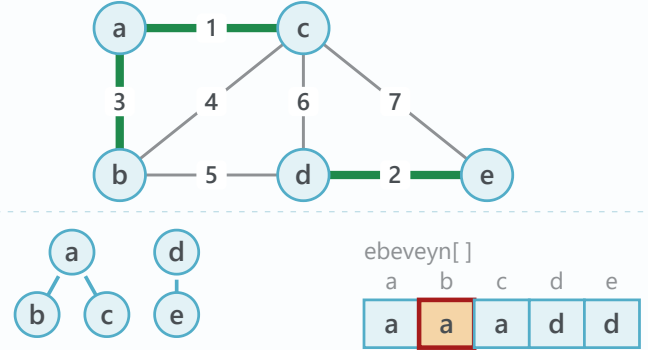
sayac = 2

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



birlestir(a, b): ebeveyn[b] = a. Kenar ağaca eklendi; toplam = 6, sayac = 3.

Adım 7 / 11

DEĞİŞKENLER

kenar = (a, b, 3)

bul(u) = a

bul(v) = b

toplam = 6

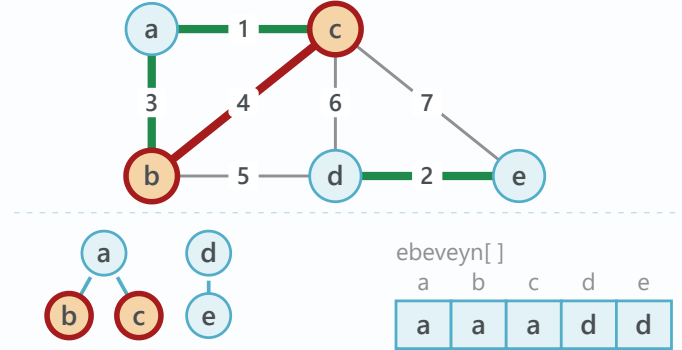
sayac = 3

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         ► if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Kenar **b-c (4)**: $\text{bul}(b) = \text{bul}(c) = a \rightarrow$ **aynı küme**; eklenirse döngü oluşur, atlanır.

Adım 8 / 11

DEĞİŞKENLER

kenar = (b, c, 4)

bul(u) = a

bul(v) = a

toplam = 6

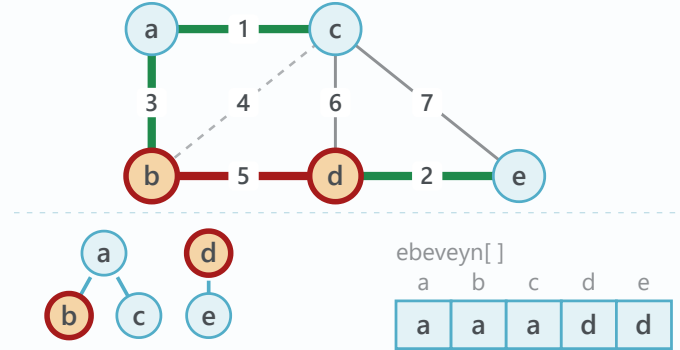
sayac = 3

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         ► if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Kenar **b-d (5)**: $\text{bul}(b) = a$, $\text{bul}(d) = d \rightarrow$ kökler **farklı**, döngü oluşmaz.

Adım 9 / 11

DEĞİŞKENLER

kenar = (b, d, 5)

bul(u) = a

bul(v) = d

toplam = 6

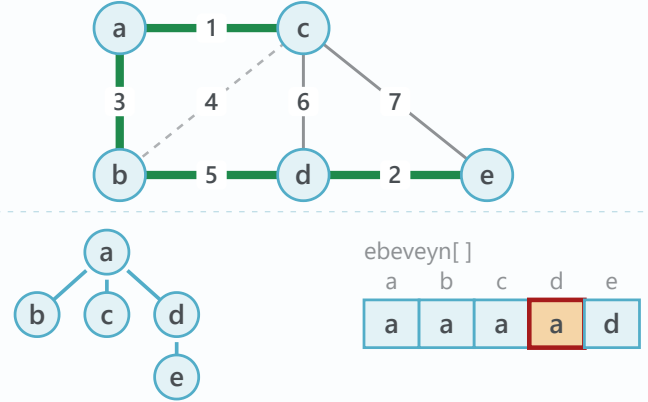
sayac = 3

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         if (kume.bul(u) != kume.bul(v)) {
8             ▶ kume.birlestir(u, v);
9             ▶ toplam += w;
10            ▶ sayac++;
11            ▶ if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



birlestir(b, d): ebeveyn[d] = a. Kenar ağaca eklendi; toplam = 11, sayac = 4. sayac = n - 1 → döngüden çıkılır.

Adım 10 / 11

DEĞİŞKENLER

kenar = (b, d, 5)

bul(u) = a

bul(v) = d

toplam = 11

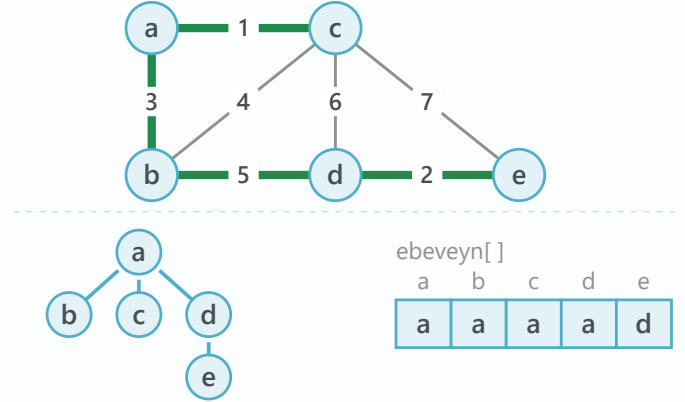
sayac = 4

kruskal(5, kenar): Kodu Adım Adım İzle

```

1 static int kruskal(int n, int[][] kenar) {
2     Arrays.sort(kenar, (x, y) -> x[2] - y[2]);
3     AyrikKume kume = new AyrikKume(n);
4     int toplam = 0, sayac = 0;
5     for (int[] k : kenar) {
6         int u = k[0], v = k[1], w = k[2];
7         if (kume.bul(u) != kume.bul(v)) {
8             kume.birlestir(u, v);
9             toplam += w;
10            sayac++;
11            if (sayac == n - 1) break;
12        }
13    }
14    return toplam;
15 }

```



Sonuç: a-c, d-e, a-b, b-d → **toplam 11**. c-d (6) ve c-e (7) hiç incelenmedi.

Adım 11 / 11

DEĞİŞKENLER

kenar = -

bul(u) = -

bul(v) = -

toplam = 11

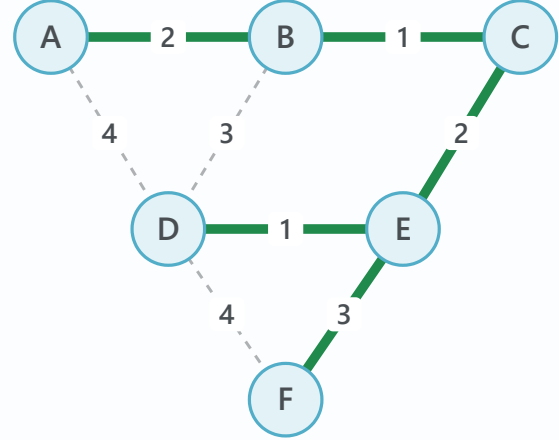
sayac = 4

KRUSKAL

Örnek: Altı Düğümlü Çizge

Kenarlar: (A,B) 2, (A,D) 4, (B,C) 1, (B,D) 3, (C,E) 2, (D,E) 1, (D,F) 4, (E,F) 3. Sıralı işleme:

kenar	w	karar
B-C	1	eklendi (toplam 1)
D-E	1	eklendi (toplam 2)
A-B	2	eklendi (toplam 4)
C-E	2	eklendi (toplam 6)
B-D	3	reddedildi: döngü
E-F	3	eklendi (toplam 9)
A-D	4	incelenmez (5 kenar tamam)
D-F	4	incelenmez (5 kenar tamam)



Ağaç: $B-C \rightarrow D-E \rightarrow A-B \rightarrow C-E \rightarrow E-F$,
toplam 9 (5 kenar = $V - 1$).

KRUSKAL

Kruskal, Prim ve Borůvka

	Kruskal	Prim	Borůvka
Açgözlü seçim	tüm çizgedeki en hafif güvenli kenar	ağaçtan çıkan en hafif kenar	her bileşenden çıkan en hafif kenar
Büyüme	orman → ağaç	tek ağaç, düğüm düğüm	orman, turlarla birleşme
Veri yapısı	sıralama + Union-Find	öncelikli kuyruk	Union-Find
Zaman	$O(E \log V)$	$O(E \log V)$ (ikili heap)	$O(E \log V)$
Uygun olduğu	seyrek çizge, kenar listesi	yoğun çizge, komşuluk	paralel / dağıtık

Üçü de aynı çizgede aynı MST'yi bulur (ağırlıklar farklıysa MST tektir). Borůvka örneğinde Kruskal da 83 verir.

Huffman Kodlama

- **Kayıpsız veri sıkıştırma** için kullanılır (David Huffman, 1952).
- **Değişken uzunluklu** kodlar kullanır: **sık** kullanılan karakterlere **kısa**, seyrek olanlara uzun kodlar atanır.
- **Önek-özgür** (prefix-free) kodlar üretir: hiçbir kod, başka bir kodun öneki olamaz → ayırıcı gerekmeden tek anlamlı çözülür.
- Açgözlü adım: **en düşük frekanslı iki** ağacı birleştir.

Önek-özgür neden önemli?

Kodlar $a = 0$, $b = 01$, $c = 1$ olsun. "01" → "ab" mi, "b" mi? Belirsiz: 0, 01'in önekidir. Huffman kodları ikili ağacın **yapraklarında** durduğu için hiçbir kod diğerinin öneki olamaz.

Kullanım: ZIP/DEFLATE, gzip, PNG, JPEG ve MP3'ün entropi kodlama aşaması.

Huffman Kodlama: Adımlar

- Adım 1:** Her karakterin **frekansı** hesaplanır.
- Adım 2:** Frekanslara göre bir **Huffman ağacı** oluşturulur: her karakter bir yaprak; en küçük iki frekanslı düğüm birleştirilerek yeni bir iç düğüm yapılır.
- Adım 3:** Ağaçta her karaktere karşılık gelen kodlar atanır: sol kenar 0, sağ kenar 1.
- Adım 4:** Veri, oluşturulan Huffman kodlarıyla kodlanır.

Örnek frekanslar (toplam 25 karakter):

karakter	F	O	R	G	E	T
frekans	2	3	4	4	5	7

n karakter için $n - 1$ birleştirme; öncelikli kuyruk ile **$O(n \log n)$** .

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         Dugum sol = kuyruk.poll();  
8         Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```

F	O	E	T	G	R
2	3	5	7	4	4

Her karakter için frekansıyla bir yaprak oluşturulup öncelikli kuyruğa (min-heap) eklenir.

Adım 1 / 12

DEĞİŞKENLER

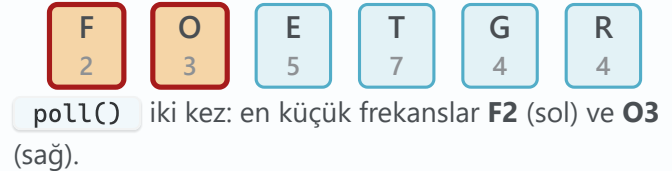
kuyruk = [F2, O3, G4, R4, E5, T7]

sol = -

sag = -

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         ▶ Dugum sol = kuyruk.poll();  
8         ▶ Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Adım 2 / 12

DEĞİŞKENLER

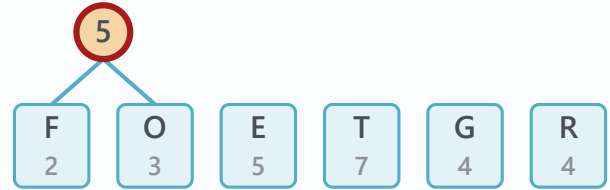
kuyruk = [G4, R4, E5, T7]

sol = F2

sag = O3

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         Dugum sol = kuyruk.poll();  
8         Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Yeni iç düğüm $5 = 2 + 3$; çocukları F2 ve O3, kuyruğa geri eklenir.

Adım 3 / 12

DEĞİŞKENLER

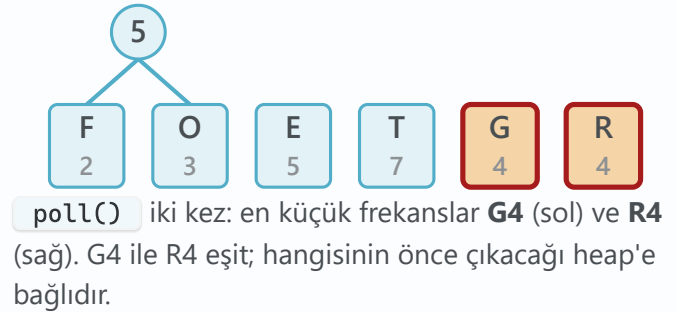
kuyruk = [G4, R4, (5), E5, T7]

sol = F2

sag = O3

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         ▶ Dugum sol = kuyruk.poll();  
8         ▶ Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Adım 4 / 12

DEĞİŞKENLER

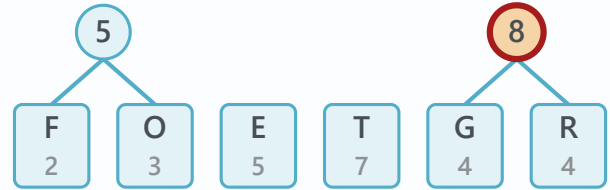
kuyruk = [(5), E5, T7]

sol = G4

sag = R4

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         Dugum sol = kuyruk.poll();  
8         Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Yeni iç düğüm $8 = 4 + 4$; çocukları G4 ve R4, kuyruğa geri eklenir.

Adım 5 / 12

DEĞİŞKENLER

kuyruk = [(5), E5, T7, (8)]

sol = G4

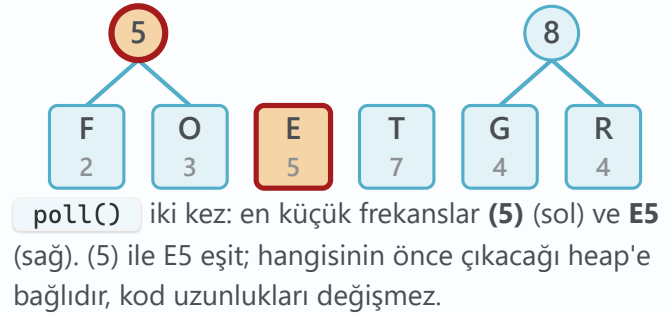
sag = R4

Huffman Ağacını Kur: Kodu Adım Adım İzle

```

1 static Dugum huffman(char[] h, int[] f) {
2     PriorityQueue<Dugum> kuyruk =
3         new PriorityQueue<>((a, b) -> a.f - b.f);
4     for (int k = 0; k < h.length; k++)
5         kuyruk.add(new Dugum(h[k], f[k]));
6     while (kuyruk.size() > 1) {
7         ▶ Dugum sol = kuyruk.poll();
8         ▶ Dugum sag = kuyruk.poll();
9         Dugum ata = new Dugum('\0', sol.f + sag.f);
10        ata.sol = sol;
11        ata.sag = sag;
12        kuyruk.add(ata);
13    }
14    return kuyruk.poll();
15 }

```



Adım 6 / 12

DEĞİŞKENLER

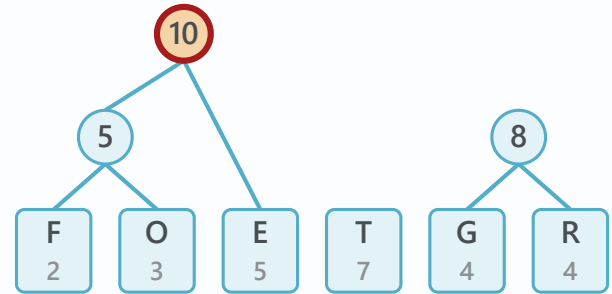
kuyruk = [T7, (8)]

sol = (5)

sag = E5

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         Dugum sol = kuyruk.poll();  
8         Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Yeni iç düğüm $10 = 5 + 5$; çocukları (5) ve E5, kuyruğa geri eklenir.

Adım 7 / 12

DEĞİŞKENLER

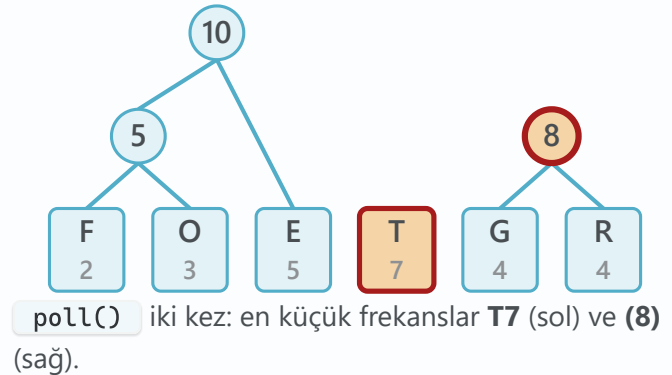
kuyruk = [T7, (8), (10)]

sol = (5)

sag = E5

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         ▶ Dugum sol = kuyruk.poll();  
8         ▶ Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Adım 8 / 12

DEĞİŞKENLER

kuyruk = [(10)]

sol = T7

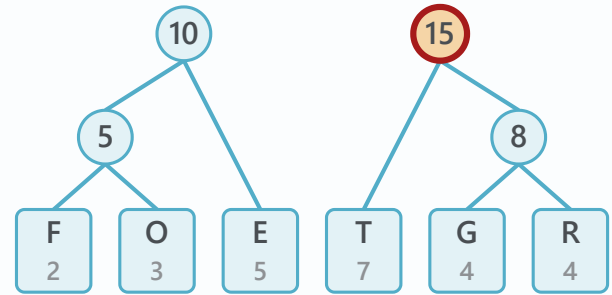
sag = (8)

Huffman Ağacını Kur: Kodu Adım Adım İzle

```

1 static Dugum huffman(char[] h, int[] f) {
2     PriorityQueue<Dugum> kuyruk =
3         new PriorityQueue<>((a, b) -> a.f - b.f);
4     for (int k = 0; k < h.length; k++)
5         kuyruk.add(new Dugum(h[k], f[k]));
6     while (kuyruk.size() > 1) {
7         Dugum sol = kuyruk.poll();
8         Dugum sag = kuyruk.poll();
9         Dugum ata = new Dugum('\0', sol.f + sag.f);
10        ata.sol = sol;
11        ata.sag = sag;
12        kuyruk.add(ata);
13    }
14    return kuyruk.poll();
15 }

```



Yeni iç düğüm $15 = 7 + 8$; çocukları T7 ve (8), kuyruğa geri eklenir.

Adım 9 / 12

DEĞİŞKENLER

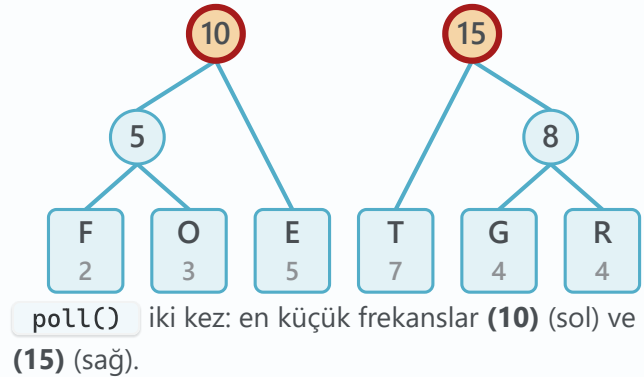
kuyruk = [(10), (15)]

sol = T7

sag = (8)

Huffman Ağacını Kur: Kodu Adım Adım İzle

```
1 static Dugum huffman(char[] h, int[] f) {  
2     PriorityQueue<Dugum> kuyruk =  
3         new PriorityQueue<>((a, b) -> a.f - b.f);  
4     for (int k = 0; k < h.length; k++)  
5         kuyruk.add(new Dugum(h[k], f[k]));  
6     while (kuyruk.size() > 1) {  
7         ▶ Dugum sol = kuyruk.poll();  
8         ▶ Dugum sag = kuyruk.poll();  
9         Dugum ata = new Dugum('\\0', sol.f + sag.f);  
10        ata.sol = sol;  
11        ata.sag = sag;  
12        kuyruk.add(ata);  
13    }  
14    return kuyruk.poll();  
15 }
```



Adım 10 / 12

DEĞİŞKENLER

kuyruk = []

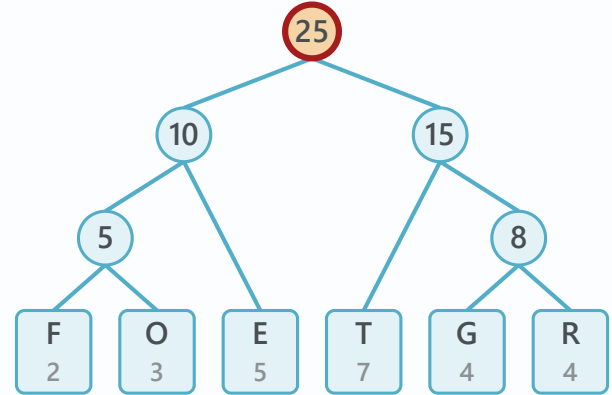
sol = **(10)**sag = **(15)**

Huffman Ağacını Kur: Kodu Adım Adım İzle

```

1 static Dugum huffman(char[] h, int[] f) {
2     PriorityQueue<Dugum> kuyruk =
3         new PriorityQueue<>((a, b) -> a.f - b.f);
4     for (int k = 0; k < h.length; k++)
5         kuyruk.add(new Dugum(h[k], f[k]));
6     while (kuyruk.size() > 1) {
7         Dugum sol = kuyruk.poll();
8         Dugum sag = kuyruk.poll();
9         Dugum ata = new Dugum('\0', sol.f + sag.f);
10        ata.sol = sol;
11        ata.sag = sag;
12        kuyruk.add(ata);
13    }
14    return kuyruk.poll();
15 }

```



Yeni iç düğüm **25** = **10** + **15**; çocukları (10) ve (15), kuyruğa geri eklenir. Kuyruқта tek düğüm kaldı: kök.

Adım 11 / 12

DEĞİŞKENLER

kuyruk = [(25)]

sol = (10)

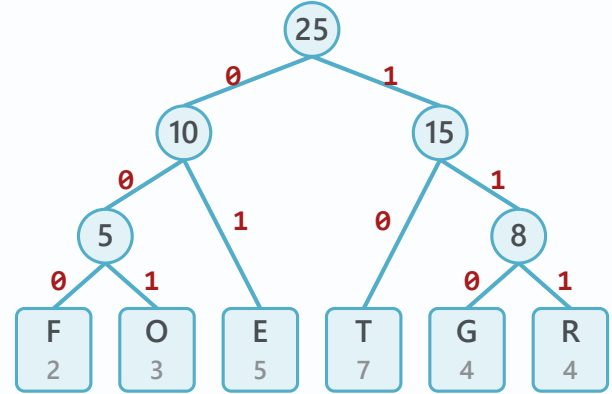
sag = (15)

Huffman Ağacını Kur: Kodu Adım Adım İzle

```

1 static Dugum huffman(char[] h, int[] f) {
2     PriorityQueue<Dugum> kuyruk =
3         new PriorityQueue<>((a, b) -> a.f - b.f);
4     for (int k = 0; k < h.length; k++)
5         kuyruk.add(new Dugum(h[k], f[k]));
6     while (kuyruk.size() > 1) {
7         Dugum sol = kuyruk.poll();
8         Dugum sag = kuyruk.poll();
9         Dugum ata = new Dugum('\0', sol.f + sag.f);
10        ata.sol = sol;
11        ata.sag = sag;
12        kuyruk.add(ata);
13    }
14    return kuyruk.poll();
15 }

```



Döngü biter, kök (25) döndürülür. Sol kenar 0, sağ kenar 1: F=000, O=001, E=01, T=10, G=110, R=111.

Adım 12 / 12

DEĞİŞKENLER

kuyruk = [(25)]

sol = (10)

sag = (15)

Kodlar, Kodlama ve Çözme

kar.	frekans	kod	uzunluk	bit
F	2	000	3	6
O	3	001	3	9
E	5	01	2	10
T	7	10	2	14
G	4	110	3	12
R	4	111	3	12
toplam				63

- Sabit uzunluk: 6 karakter $\rightarrow$ 3 bit, $25 \times 3 = \mathbf{75}$ **bit**. Huffman: **63 bit** (%16 kazanç, ortalama 2,52 bit).
- **Kodlama:** "FORTE" $\rightarrow$ 000 001 111 10 01 = 13 bit (sabit kodla 15).
- **Çözme:** kökten başla, 0 $\rightarrow$ sol, 1 $\rightarrow$ sağ; yaprağa gelince karakteri yaz ve köke dön:
 $1100110 \rightarrow 110 \mid 01 \mid 10 \rightarrow$ "GET".
- Sol ve sağ çocuğun yeri keyfidir (ör. T sağ çocuk yapılabilir): kodlar değişir ama kod **uzunlukları** ve 63 bit değişmez.

Huffman Neden En İyi?

Açgözlü seçim

En düşük frekanslı iki karakter x ve y 'nin, en derin seviyede **kardeş** olduğu bir en iyi kod ağacı vardır: daha derindeki bir karakterle yer değiştirmek toplam biti artırmaz (değiştirme argümanı).

Optimal alt yapı

x ve y , frekansı $f(x) + f(y)$ olan tek bir "z" karakteri yapılırsa, küçülen problemin en iyi ağacından z açılarak asıl problemin en iyi ağacı elde edilir.

Maliyet

Toplam bit = $\sum \text{frekans}(c) \cdot \text{derinlik}(c)$.
Huffman, önek-özgür kodlar arasında bunu **en küçük** yapar.

Karmaşıklık

Frekans sayımı $O(m)$ (m = metin uzunluğu);
ağaç $O(n \log n)$; kodlama $O(m)$. Ağaç (ya da kod tablosu) da çözücüye gönderilmelidir.

Diğer Açgözlü Algoritmalar

Dijkstra ve Prim

Her adımda uzaklığı / bağlantı maliyeti en küçük düğümü kalıcı olarak seçer (Bölüm 4). Negatif kenarsız çizgede en iyisidir.

Çizge boyama

Düğümleeri sırayla, komşularının kullanmadığı en küçük renkle boyar. Hızlı ama renk sayısı en az olmayabilir (NP-zor).

Kutulara yerleştirme

(Bin packing) İlk sığan / en iyi sığan: nesneyi sığdığı ilk kutuya koyar. Azalan sırayla ilk sığan en fazla $\approx 11/9 \cdot \text{OPT} + 1$ kutu kullanır.

En büyük kapsama

(Maximum coverage) Her adımda en çok yeni öğe kapsayan kümeyi seçer: en iyinin en az $(1 - 1/e) \approx \%63$ 'ünü garanti eder.

NP-zor problemlerde açgözlü algoritmalar, kanıtlanmış oranlı **yaklaşım algoritmaları** (Bölüm 7) olarak değerlidir.

ÖZET

Özet

Problem	Açgözlü ölçüt	Zaman	En iyi mi?
Para üstü	sığan en büyük para	$O(k + \text{adet})$	kanonikse
Etkinlik seçimi	en erken biten	$O(n \log n)$	evet
İş sıralama	en kârlı iş, en geç boş slot	$O(n^2)$	evet
Kesirli sırt çantası	en büyük kâr/ağırlık	$O(n \log n)$	evet (0/1: hayır)
Boruvka	her bileşenin en ucuz dış kenarı	$O(E \log V)$	evet
Kruskal	döngü kurmayan en hafif kenar	$O(E \log V)$	evet
Huffman	en düşük frekanslı iki ağaç	$O(n \log n)$	evet
Union-Find	bul / birleştir, yol kısaltma + rütbeye göre birleştirme → amortize $O(\alpha(n))$		

Etkileşimli Simülatörler

Minimum Kapsayan Ağaç

Rastgele bağlı ağırlıklı bir çizge üretin; Prim ve Kruskal algoritmalarını adım adım, yan yana çalıştırıp seçilen kenarları ve toplam ağırlığı karşılaştırın.

Dizgi Sıkıştırma

RLE (Run Length Encoding) ve LZW sıkıştırmasını adım adım izleyin, sıkıştırma oranını görün. Huffman bu sayfada yok; sözlük tabanlı LZW ile frekans tabanlı Huffman'ı karşılaştırmak için kullanın.

ÖZET

Sık Yapılan Hatalar

Açgözlüyü kanıtsız doğru saymak

{1, 3, 4} paralarıyla 6: açgözlü 3 para, en iyi 2. Her problemde açgözlü seçim özelliği gösterilmelidir.

Yanlış ölçüt

Etkinlik seçiminde en erken **başlayan** ya da en **kısa** etkinlik ölçütü yanlıştır; doğru ölçüt en erken **biten**.

0/1 sırt çantasında oran

Kesirli problemde en iyi olan oran ölçütü, bölünemeyen nesnelerde başarısızdır ($160 < 220$).

Döngü denetimi

Kruskal'da $ebeveyn[u] \neq ebeveyn[v]$ yetmez; **kökler** karşılaştırılmalı: $bul(u) \neq bul(v)$.

Borůvka'da eşitlik

Eşit ağırlıklarda tutarsız seçim döngü kurabilir; aynı kenar iki bileşence seçilince iki kez eklenmemelidir.

Huffman'da ağaçsız çözme

Kod tablosu ya da ağaç gönderilmezse veri çözülemez. Tek karakterli metinde kök yapaktır: koda en az 1 bit verilmelidir.

ÖZET

Örnek Problemler

1. $\{1, 5, 10, 25\}$ paralarıyla 63 açgözlü olarak kaç parayla ödenir? $\{1, 7, 10\}$ ile 14 için açgözlü ve en iyi sonuç nedir?
2. Etkinlikler (s, f) : $(1,4), (3,5), (0,6), (5,7), (3,9), (5,9), (6,10), (8,11), (8,12), (2,14), (12,16)$. En fazla kaç etkinlik seçilir?
3. Frekanslar $a:5, b:9, c:12, d:13, e:16, f:45$. Huffman kod uzunluklarını ve toplam biti bulun.
4. Borůvka örnek çizgesine Kruskal uygulayın: kenarlar hangi sırayla eklenir?
5. $V = 1000$ düğümlü bir çizgede Borůvka en fazla kaç tur sürer?

► Cevapları göster

ÖZET

Alıştırmalar

1. İşler (id, son tarih, kâr): (a,2,100), (b,1,19), (c,2,27), (d,1,25), (e,3,15). Açgözlü iş sıralamasını uygulayın.
2. $W = 10$, nesneler (kâr, ağırlık): (10,5), (40,4), (30,6), (50,3). Kesirli sırt çantasının değeri nedir?
3. "ABRACADABRA" için Huffman kodlaması kaç bit tutar? Sabit uzunluklu kodla karşılaştırın.
4. Kruskal'ın tüm kenarlar aynı ağırlıkta olduğunda ne ürettiğini ve $O(E \log V)$ maliyetinin hangi adımdan geldiğini açıklayın.
5. Açgözlü para üstünün en iyi sonuç vermediği, en küçük tutarlı bir para sistemi bulun ve gerekçelendirin.

► Cevaplar / ipuçları

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 15: Greedy Algorithms; Bölüm 19: Disjoint Sets; Bölüm 21: MST).
- Kleinberg ve Tardos, *Algorithm Design*, 2006 (Bölüm 4: Greedy Algorithms).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011 (4.3 MST, 5.5 Data Compression).
- D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the IRE* 40(9), 1952.
- J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proc. AMS* 7(1), 1956.
- O. Borůvka, "O jistém problému minimálním", *Práce Moravské přírodovědecké společnosti* 3, 1926.
- R. E. Tarjan, "Efficiency of a Good But Not Linear Set Union Algorithm", *Journal of the ACM* 22(2), 1975.

Son

Bölüm 12: Açgözlü Algoritmalar

Sonraki bölüm: Bölüm 13: Geri İzleme Algoritmaları · [Terim sözlüğü](#)