

Bölüm 14: Dal Sınır Algoritmaları

Algoritmalar

Durum uzayı ağacı

Sınırlama ve budama

FIFO, LIFO, en düşük maliyet

0/1 sırt çantası

Gezgin satıcı

Tam sayılı programlama

Maksimum klik

Geri izleme ile karşılaştırma

GİRİŞ

Öğrenme Çıktıları

Paradigma

Dal sınır yönteminin durum uzayı ağacı, dallanma, sınırlama ve budama adımlarını açıklamak; geri izlemeden farkını söylemek.

Sınır tasarımı

En büyükleme için üst, en küçükleme için alt sınır kurmak; sınırın geçerli (iyimser) olmasının neden şart olduğunu gerekçelendirmek.

Arama stratejileri

FIFO, LIFO ve en düşük maliyet (en iyi önce) stratejilerini aynı örnekte izlemek; bellek ve düğüm sayısı açısından karşılaştırmak.

Uygulamalar

0/1 sırt çantası, gezgin satıcı, tam sayılı doğrusal programlama ve maksimum klik problemlerini dal sınır ile adım adım çözmek.

GİRİŞ

En İyi Seçenekleri Ele Almak

- Karmaşık problemleri çözmek, bazen bir **yol ayrımında** doğru seçeneği bulmaya benzer.
- **Dal sınır** (branch and bound) algoritmaları bu problemler için akıllı bir yaklaşım sunar: olası çözüm yollarını inceleyerek **en iyi seçeneği** bulmaya çalışır.
- Hedef: **kombinatoriyal optimizasyon** problemlerinde **kesin** (optimal) çözüm. Tüm adayları denemek ise üstel ya da faktöriyel sürer.
- Fikir: bir dalın en fazla ne kadar iyi olabileceğini **önceden tahmin et**; yeterince iyi değilse o dala hiç girme.

n	2^n alt küme	$(n-1)!$ tur
5	32	24
10	1 024	362 880
20	1 048 576	$\approx 1,2 \cdot 10^{17}$
40	$\approx 1,1 \cdot 10^{12}$	$\approx 2,0 \cdot 10^{46}$

Sırt çantası 2^n alt kümeyi, gezgin satıcı $(n-1)!$ turu dener: kaba kuvvet hızla imkânsızlaşır.

Temel Kavramlar

Durum uzayı ağacı

Kök = hiçbir karar verilmemiş durum; her seviye bir karar (ör. "eşya i alınsın mı?"); yapraklar tam çözümler.

Dallanma

Bir düğümün alt problemlerini (çocuklarını) üretmek. Genişletilen düğüme **E-düğüm** (expanding node) denir.

Sınır

Düğümün altındaki **hiçbir** çözümün geçemeyeceği iyimser değer: en büyüklemede üst, en küçüklemede alt sınır.

Canlı düğüm

Üretilmiş ama henüz genişletilmemiş, umut vaat eden düğüm. Hepsi bir **kuyruk** yapısında bekler.

Ölü düğüm

Genişletilmiş, olursuz ya da sınırı yetersiz olduğu için budanmış düğüm; bir daha ele alınmaz.

eniyi (incumbent)

Şimdiye kadar bulunan en iyi tam çözümün değeri. Budama kararları bu değere göre verilir.

Genel Şema (En Büyükleme)

```
int enIyi = -∞;
kuyruk.ekle(kök);
while (!kuyruk.bosMu()) {
    Dugum u = kuyruk.cikar(); // strateji
    if (sinir(u) <= enIyi) continue;
    for (Dugum c : u.cocuklar()) {
        if (!olurluMu(c)) continue;
        if (tamCozumMu(c) && deger(c) > enIyi)
            enIyi = deger(c); // yeni aday
        if (sinir(c) > enIyi)
            kuyruk.ekle(c); // canlı düğüm
    }
}
return enIyi;
```

1. **Seç:** kuyruktan bir canlı düğüm çıkar. Hangisinin çıkacağını **strateji** belirler (FIFO, LIFO, en iyi sınır).
2. **Tekrar sına:** düğüm beklerken eniyi büyümüş olabilir; sınırı artık yetersizse at.
3. **Dallan:** çocukları üret; olursuzları ele.
4. **Güncelle:** tam çözüm daha iyiye eniyi'yi yükselt.
5. **Sınırla:** yalnızca sınırı eniyi'yi geçebilen çocukları kuyruğa koy.

En küçüklemede yönler ters çevrilir: alt sınır kullanılır, $\text{alt}(c) \geq \text{enIyi}$ ise budanır.

Sınır (Bound) Nasıl Seçilir?

Geçerli (iyimser) olmalı

En büyüklemede $\text{sınır}(u) \geq$ alt ağaçtaki her çözüm; en küçüklemede $\text{alt}(u) \leq$ her çözüm. Aksi hâlde optimum budanabilir ve sonuç **yanlış** olur.

Sıkı olmalı

Gerçek değere ne kadar yakınsa o kadar çok dal budanır. Sonsuz büyük bir "sınır" geçerlidir ama hiçbir şey budamaz.

Ucuz hesaplanmalı

Sınır her düğümde hesaplanır. Çok pahalı bir sınır, budadığı düğümlerden daha fazla zaman harcatabilir.

Gevşetme (relaxation) ile kurulur

Zor bir kısıtı kaldırıp kolay problemi çöz: kesirli sırt çantası, LP gevşetmesi, en ucuz çıkış kenarları.

Başarı büyük ölçüde iyi bir sınır tahminine bağlıdır. En küçüklemede **alt sınır**, en iyi senaryoda elde edilebilecek minimum değerdir; zayıf bir alt sınır arama süresini uzatır.

Bir Düğüm Ne Zaman Budanır?

1. Olursuzluk

Düğüm bir kısıtı ihlal ediyor: ağırlık > kapasite, aynı şehre ikinci ziyaret, LP'nin çözümü yok. Altında hiçbir geçerli çözüm yoktur.

2. Sınır

$\text{sinir}(u) \leq \text{enlyi}$ (en büyükleme) ya da $\text{alt}(u) \geq \text{enlyi}$ (en küçükleme). Altındaki çözümler en iyiyi geçemez.

3. Tamlık

Düğüm tam bir çözüm (yaprak) ya da gevşetmenin çözümü zaten olurlu (ör. LP çözümü tam sayı). Değeri enlyi ile karşılaştırılır; dallanmaya gerek kalmaz.

Geri izleme yalnızca 1. kuralı (ve tamlığı) kullanır. Dal sınırı ondan ayıran şey **2. kuraldır**: düğüm geçerli olsa bile "yeterince iyi değilse" kesilir.

Arama Stratejileri: Sıradaki Canlı Düğüm Hangisi?

Strateji	Kuyruk	Nasıl gezer?	Artı / eksi
FIFO dal sınır	kuyruk (<code>ArrayDeque</code>)	seviye seviye, genişlik öncelikli (BFS)	sığ çözümleri erken bulur; canlı düğüm sayısı üstel büyüyebilir
LIFO dal sınır	yığın (ya da özyineleme)	derinlik öncelikli (DFS)	bellek $O(\text{derinlik})$; erken bir tam çözüm enli'yi hızla yükseltir
En düşük maliyet (LC, en iyi önce)	öncelikli kuyruk (<code>PriorityQueue</code>)	sınırı en umut verici düğüm önce	genellikle en az düğümü genişletir; kuyruk bellek tüketir, her işlem $O(\log k)$

- Üç strateji de **aynı optimumu** bulur; değişen, üretilen düğüm sayısı ve bellektir.
- LC aramada maliyet fonksiyonu $\hat{c}(u) = \text{sınır değeri (ya da } f(u) + \hat{g}(u) \text{ tahmini)}$ olur; A* araması bu fikrin yol bulmadaki karşılığıdır.

Geri İzleme ve Dal Sınır

Özellik	Geri izleme (Bölüm 13)	Dal sınır
Problem türü	karar / arama: bir ya da tüm geçerli çözümler (N-vezir, Sudoku)	optimizasyon: en iyi çözüm (en büyük kazanç, en kısa tur)
Budama ölçütü	olurluluk: kısıt ihlal ediliyor mu?	olurluluk + sınır : enliyi'yi geçebilir mi?
Gezinme	her zaman DFS (özyineleme)	DFS, BFS ya da en iyi önce
Ek bilgi	yalnızca kısmi çözüm	kısmi çözüm + sınır fonksiyonu + enliyi
Bellek	$O(\text{derinlik})$	DFS'te $O(\text{derinlik})$; BFS/LC'de kuyruk boyu kadar
En kötü durum	üstel	yine üstel; pratikte çok daha az düğüm

Sınır testi olmayan bir TSP ya da maksimum klik araması aslında geri izlemedir; tek bir sınır satırı eklenince dal sınırı dönüşür (ilerideki slaytlar).

Avantajlar ve Dezavantajlar

Avantajlar

- + Karmaşık optimizasyon problemlerinde **kesin en iyi** çözümü bulur.
- + Arama alanını daraltır, gereksiz olasılıkları eler.
- + En iyi rotayı ya da seçimi bulmayı gerektiren problemlerde idealdir.
- + Çözümleri ağaç olarak temsil edilebilen çok çeşitli problemlere uygulanır.
- + Erken durdurulursa elde iyi bir çözüm ve onun sınırı (kalite garantisi) bulunur.

Dezavantajlar

- Çok fazla olasılık içeren durumlarda çözüm süresi artabilir: en kötü durumda **üstel** zaman.
- FIFO ve en iyi önce stratejilerinde bellek de üstel büyüyebilir.
- Başarı, **iyi bir sınır tahminine** bağlıdır; zayıf sınır neredeyse hiç budamaz.
- Probleme özgü sınır ve dallanma kuralı tasarlamak gerekir; uygulaması geri izlemeden karmaşıktır.

0/1 SIRT ÇANTASI

Örnek: 0/1 Sirt Çantası Problemi

- n eşya: kazanç p_i , ağırlık w_i , kapasite W . Her eşya **ya alınır ya alınmaz** ($x_i \in \{0, 1\}$).
- Amaç: $\sum w_i x_i \leq W$ koşuluyla $\sum p_i x_i$ 'yi **en büyükmek**.
- Eşyalar **p_i/w_i oranına göre azalan** sırada dizilir; sınır hesabı bu sırayı kullanır.
- Durum uzayı ağacı ikilidir: i. seviyede sol dal "eşya i alındı" (1), sağ dal "alınmadı" (0). Tam ağaç $2^5 - 1 = 31$ düğüm.

Eşya	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Kapasite $W = 16$.

Oranla açgözlü seçim 1, 2, 4'ü alır: kazanç **80**.
Optimum ise 1 ve 3: kazanç **90** (ağırlık 12).
Açgözlü yetmez; tümünü denemek pahalı → dal sınırı.

Üst Sınır: Kesirli Sirt Çantası Gevşetmesi

- Gevşetme: "eşya bölünemez" kısıtını kaldır. **Kesirli** sırt çantasını açgözlü yöntem (Bölüm 12) en iyi şekilde çözer.
- u düğümünde (kazanç k , ağırlık a , seviye i): kalan eşyaları oran sırasıyla, sığdığı kadar **tam** ekle; ilk sığmayan eşyanın **kesirli** parçasını ekle:

$$\text{sinir}(u) = k + \sum_{j=i+1}^{m-1} p_j + (W - a - \sum w_j) \cdot p_m / w_m \quad (m: \text{ilk sığmayan eşya})$$

Neden geçerli?

Her 0/1 çözüm aynı zamanda bir kesirli çözümdür. Kesirli optimum $\geq$ 0/1 optimum, yani sınır asla gerçek en iyiyi küçük göstermez.

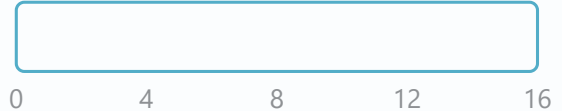
Kök için

$40 + 30 = 70$ (ağırlık 7); eşya 3 sığmaz, kalan 9 birim $\rightarrow 9 \cdot 50/10 = 45$. **sinir(kök) = 115**. $a > W$ ise düğüm olursuzdur (sınır 0).

sinir(kök): Kodu Adım Adım İzle

```
▶ double sinir(Dugum u) {  
▶   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   while (j < n && toplamAg + w[j] <= W) {  
7       toplamAg += w[j];  
8       sonuc += p[j];  
9       j++;  
10  }  
11  if (j < n)                        // kesirli parça  
12      sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite $W = 16$



henüz eşya yok

Kök düğüm: seviye -1 , kazanç 0 , ağırlık 0 .
 $0 \leq 16$ olduğundan düğüm **olurlu**.

Adım 1 / 9

DEĞİŞKENLER

$u = A$ (kök)

$j = -$

$toplamAg = -$

$sonuc = -$

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {
2   if (u.agirlik > W) return 0;    // olursuz
3   ▶ double sonuc = u.kazanc;
4   ▶ int toplamAg = u.agirlik;
5   ▶ int j = u.seviye + 1;
6   while (j < n && toplamAg + w[j] <= W) {
7     toplamAg += w[j];
8     sonuc += p[j];
9     j++;
10  }
11  if (j < n)                        // kesirli parça
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];
13  return sonuc;
14 }
```

kapasite W = 16



j = 0: sırada eşya 1

Sınır mevcut kazançtan başlar; **j** bir sonraki eşyayı gösterir (indis 0 = eşya 1).

Adım 2 / 9

DEĞİŞKENLER

u = A (kök)

j = 0

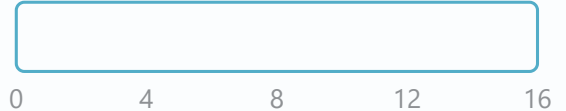
toplamAg = 0

sonuc = 0

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {
2   if (u.agirlik > W) return 0;    // olursuz
3   double sonuc = u.kazanc;
4   int toplamAg = u.agirlik;
5   int j = u.seviye + 1;
6   ► while (j < n && toplamAg + w[j] <= W) {
7     toplamAg += w[j];
8     sonuc += p[j];
9     j++;
10  }
11  if (j < n)                        // kesirli parça
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];
13  return sonuc;
14 }
```

kapasite $W = 16$



$0 + 2 \leq 16$?

$0 + w[0] = 2 \leq 16 \rightarrow$ eşya 1 **tamamen** sığar.

Adım 3 / 9

DEĞİŞKENLER

$u = A$ (kök)

$j = 0$

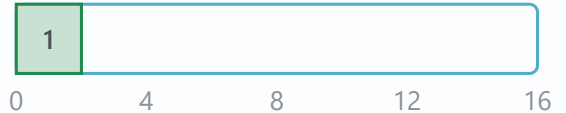
$toplamAg = 0$

$sonuc = 0$

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {  
2   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   while (j < n && toplamAg + w[j] <= W) {  
7     toplamAg += w[j];  
8     sonuc += p[j];  
9     j++;  
10  }  
11  if (j < n)                      // kesirli parça  
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite $W = 16$



eşya 1 eklendi

toplamAg = 2, sonuc = 40, j = 1.

Adım 4 / 9

DEĞİŞKENLER

u = A (kök)

j = 1

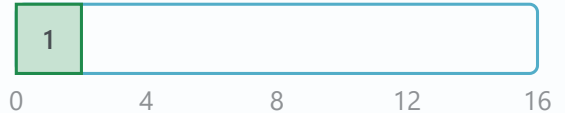
toplamAg = 2

sonuc = 40

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {  
2   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   ► while (j < n && toplamAg + w[j] <= W) {  
7     toplamAg += w[j];  
8     sonuc += p[j];  
9     j++;  
10  }  
11  if (j < n)                        // kesirli parça  
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite $W = 16$



$2 + 5 \leq 16$?

$2 + w[1] = 7 \leq 16 \rightarrow$ eşya 2 de tamamen sığar.

Adım 5 / 9

DEĞİŞKENLER

$u = A$ (kök)

$j = 1$

$toplamAg = 2$

$sonuc = 40$

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {  
2   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   while (j < n && toplamAg + w[j] <= W) {  
7     toplamAg += w[j];  
8     sonuc += p[j];  
9     j++;  
10  }  
11  if (j < n)                      // kesirli parça  
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite $W = 16$



eşya 2 eklendi

$\text{toplamAg} = 7, \text{sonuc} = 70, j = 2.$

Adım 6 / 9

DEĞİŞKENLER

$u = A$ (kök)

$j = 2$

$\text{toplamAg} = 7$

$\text{sonuc} = 70$

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {  
2   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   ► while (j < n && toplamAg + w[j] <= W) {  
7     toplamAg += w[j];  
8     sonuc += p[j];  
9     j++;  
10  }  
11  if (j < n)                        // kesirli parça  
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite W = 16



$$7 + 10 = 17 > 16$$

$7 + w[2] = 17 > 16 \rightarrow$ eşya 3 tamamen sığmaz, döngü biter.

Adım 7 / 9

DEĞİŞKENLER

u = A (kök)

j = 2

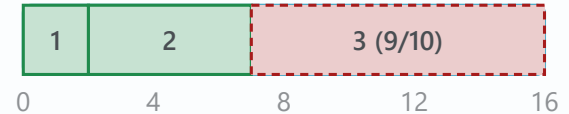
toplamAg = 7

sonuc = 70

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {  
2   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   while (j < n && toplamAg + w[j] <= W) {  
7     toplamAg += w[j];  
8     sonuc += p[j];  
9     j++;  
10  }  
11  ▶ if (j < n)                // kesirli parça  
12  ▶   sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite $W = 16$



kalan 9 birim: eşya 3'ün 9/10'u

Kalan kapasite 9: eşya 3'ün **9/10'u** $\rightarrow 9 \cdot 50/10 = 45$ eklenir, sonuc = 115.

Adım 8 / 9

DEĞİŞKENLER

$u = A$ (kök)

$j = 2$

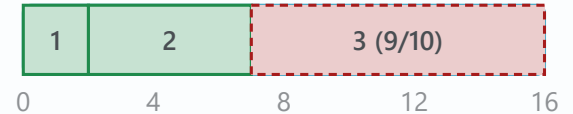
toplamAg = 7

sonuc = 115

sinir(kök): Kodu Adım Adım İzle

```
1 double sinir(Dugum u) {  
2   if (u.agirlik > W) return 0;    // olursuz  
3   double sonuc = u.kazanc;  
4   int toplamAg = u.agirlik;  
5   int j = u.seviye + 1;  
6   while (j < n && toplamAg + w[j] <= W) {  
7     toplamAg += w[j];  
8     sonuc += p[j];  
9     j++;  
10  }  
11  if (j < n)                        // kesirli parça  
12    sonuc += (W - toplamAg) * (double) p[j] / w[j];  
13  return sonuc;  
14 }
```

kapasite $W = 16$



sinir(A) = 115

sinir(A) = 115: bu düğümün altındaki hiçbir çözüm 115'ten fazla kazandıramaz.

Adım 9 / 9

DEĞİŞKENLER

$u = A$ (kök)

$j = 2$

$toplamAg = 7$

$sonuc = 115$

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı

kök

A 0 / 0
s = 115

eşya 1
(40, 2)

eşya 2
(30, 5)

eşya 3
(50, 10)

eşya 4
(10, 5)

Kök **A**: hiçbir eşya seçilmedi (kazanç 0, ağırlık 0). Kesirli sınır **s = 115**; eniyi = 0. Kuyruk: A (115).

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı

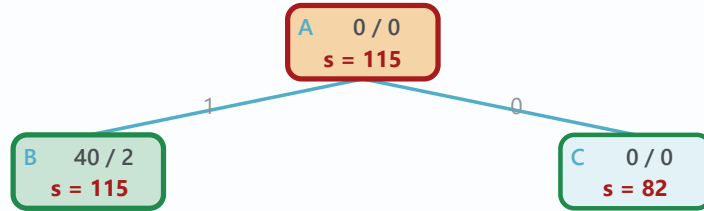
kök

eşya 1
(40, 2)

eşya 2
(30, 5)

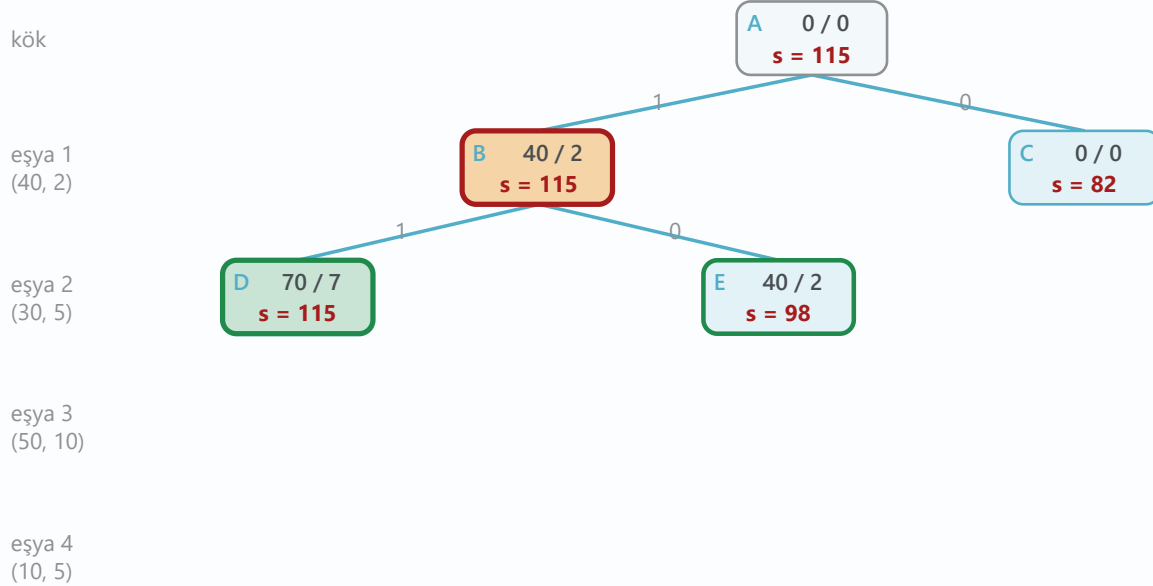
eşya 3
(50, 10)

eşya 4
(10, 5)



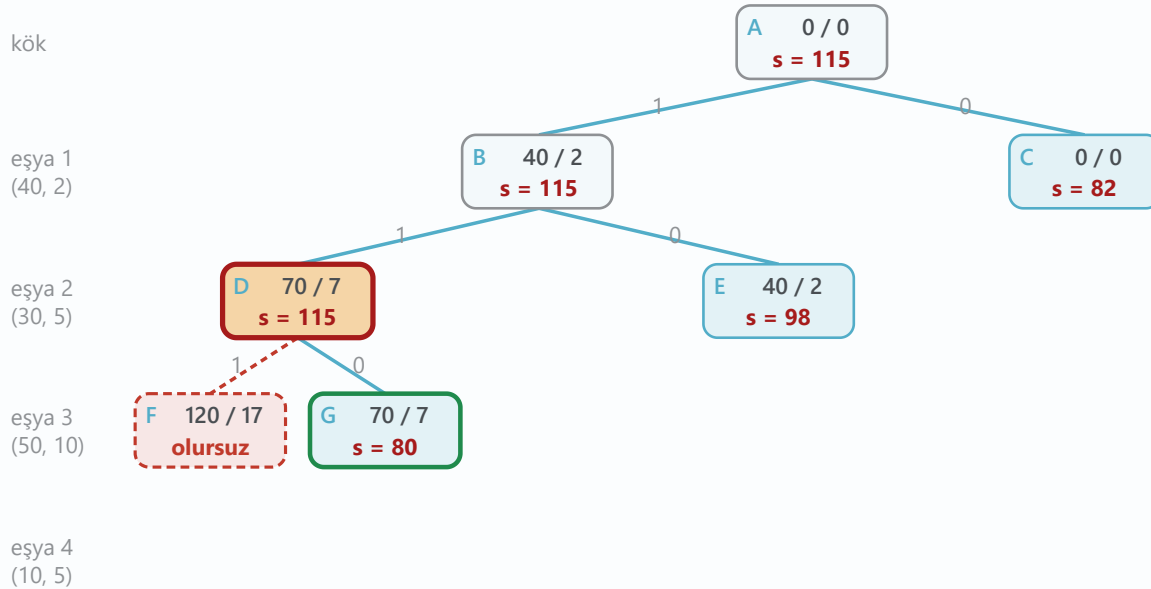
A çıkarıldı (en büyük sınır) ve dallandı. **B** (eşya 1 alındı): **eniyi = 40**, $s = 115 \rightarrow$ kuyruğa. **C** (eşya 1 yok): $s = 82 \rightarrow$ kuyruğa. Kuyruk: B (115), C (82).

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı



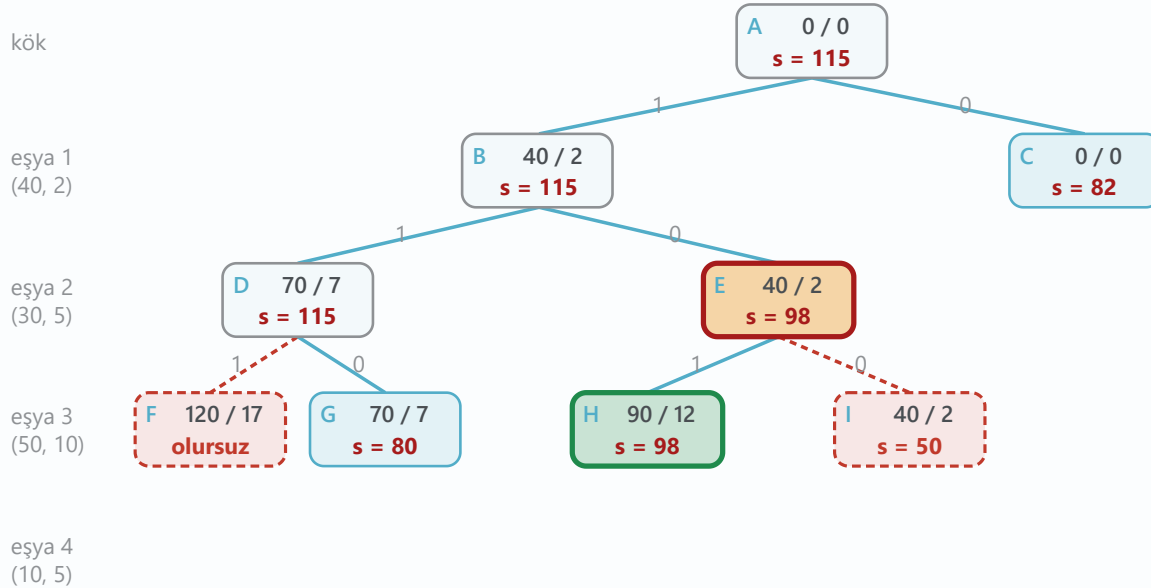
B çıkarıldı (en büyük sınır) ve dallandı. **D** (eşya 2 alındı): **eniyi = 70**, s = 115 → kuyruğa. **E** (eşya 2 yok): s = 98 → kuyruğa. Kuyruk: D (115), E (98), C (82).

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı



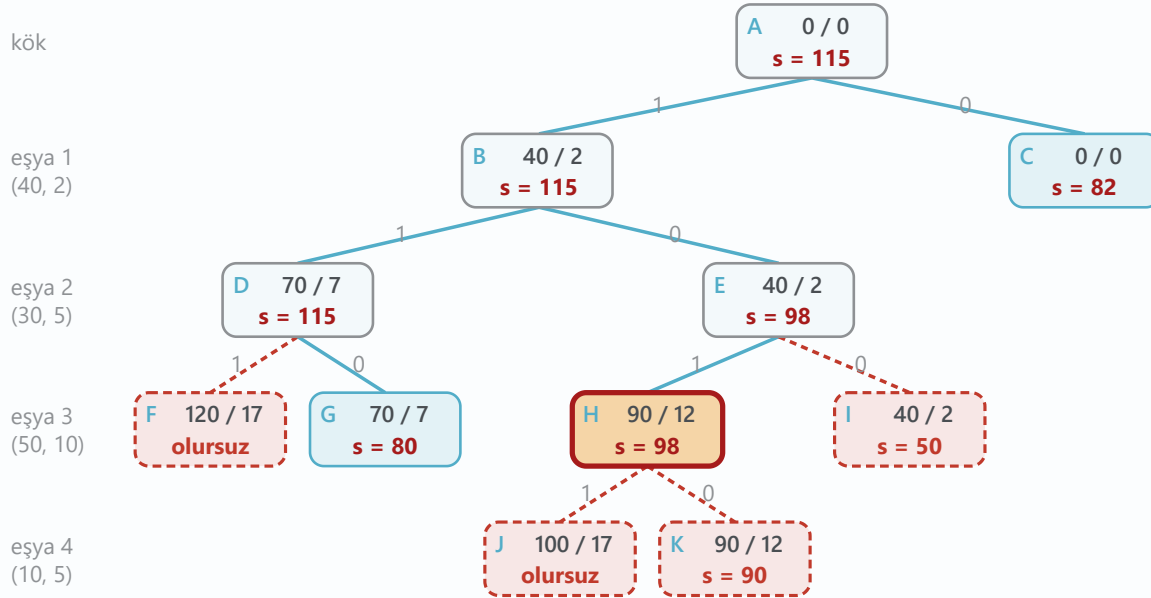
D çıkarıldı (en büyük sınır) ve dallandı. **F** (eşya 3 alındı): ağırlık $17 > 16 \rightarrow$ **olursuz**. **G** (eşya 3 yok): $s = 80 \rightarrow$ kuyruğa. Kuyruk: E (98), C (82), G (80).

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı



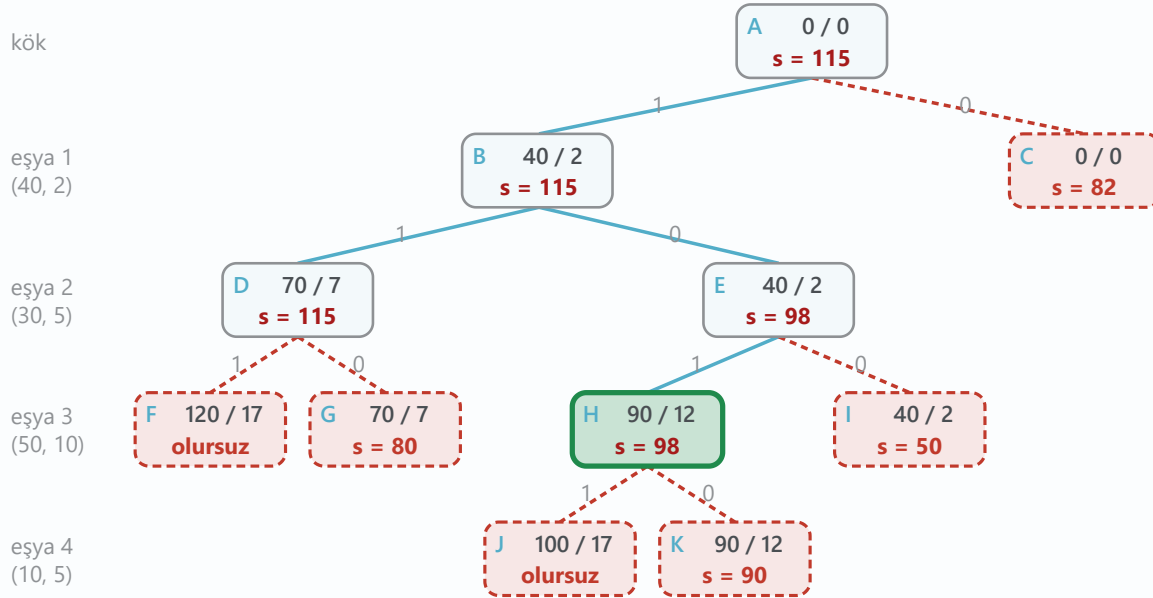
E çıkarıldı (en büyük sınır) ve dallandı. **H** (eşya 3 alındı): **eniyi = 90**, $s = 98 \rightarrow$ kuyruğa. **I** (eşya 3 yok): $s = 50 \leq 90 \rightarrow$ budandı. Kuyruk: H (98), C (82), G (80).

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı



H çıkarıldı (en büyük sınır) ve dallandı. **J** (eşya 4 alındı): ağırlık $17 > 16 \rightarrow$ **olursuz**. **K** (eşya 4 yok): $s = 90 \leq 90 \rightarrow$ budandı. Kuyruk: C (82), G (80).

En İyi Önce (LC) Dal Sınır: Durum Uzayı Ağacı



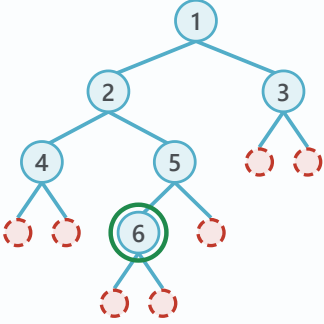
C çıkarıldı: $s = 82 \leq \text{eniyi} = 90 \rightarrow$ budandı. **G** çıkarıldı: $s = 80 \leq \text{eniyi} = 90 \rightarrow$ budandı. Kuyruk boş: **en iyi kazanç 90** (H: eşya 1 ve 3). Tam ağacın 31 düğümünden yalnızca 11'i üretildi.

En İyi Önce Dal Sınır: Java

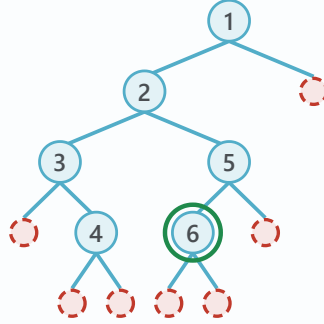
```
int dalSindir() {
    PriorityQueue<Dugum> kuyruk = new PriorityQueue<>((
        (a, b) -> Double.compare(b.sinir, a.sinir));
    Dugum kok = new Dugum(-1, 0, 0);
    kok.sinir = sinir(kok);
    kuyruk.add(kok); int enIyi = 0;
    while (!kuyruk.isEmpty()) {
        Dugum u = kuyruk.poll(); // en büyük sınır
        if (u.sinir <= enIyi || u.seviye == n - 1) continue;
        int i = u.seviye + 1;
        // sol çocuk: i. eşyayı al
        Dugum al = new Dugum(i, u.kazanc + p[i],
            u.agirlik + w[i]);
        if (al.agirlik <= W && al.kazanc > enIyi)
            enIyi = al.kazanc;
        al.sinir = sinir(al);
        if (al.sinir > enIyi) kuyruk.add(al);
        // sağ çocuk: i. eşyayı alma
        Dugum alma = new Dugum(i, u.kazanc, u.agirlik);
        alma.sinir = sinir(alma);
        if (alma.sinir > enIyi) kuyruk.add(alma);
    }
    return enIyi;
}
```

- **Dugum** : seviye, kazanç, ağırlık, sınır (seçilen eşyalar için bir **boolean[]** ya da bit maskesi eklenebilir).
- Karşılaştırmacı ters: **en büyük sınır** önce çıkar.
- **poll** sonrası test, kuyrukta beklerken **eskiyen** düğümleri atar (örnekte C ve G).
- FIFO için **ArrayDeque** + **add()** / **poll()** , LIFO için **push()** / **pop()** yeterlidir; kodun geri kalanı aynıdır.
- En kötü durum $O(2^n)$ düğüm, düğüm başına $O(n)$ sınır hesabı.

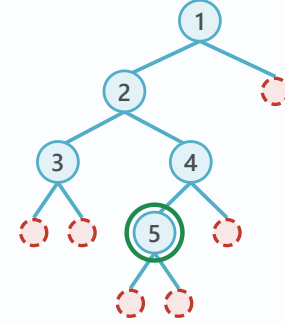
Aynı Örnekte Üç Strateji



FIFO: 13 düğüm, 6 genişletme



LIFO: 13 düğüm, 6 genişletme



En iyi önce: 11 düğüm, 5 genişletme

Numaralar genişletme sırasıdır; küçük daireler üretilip genişletilmeyen düğümler (kırmızı kesikli: budanan), halka: optimum (90).

Yöntem	Üretilen düğüm	Genişletilen	En büyük kuyruk
Kaba kuvvet (tam ağaç)	31	15	—
FIFO dal sınır	13	6	3
LIFO dal sınır (önce "al")	13	6	3
En iyi önce (LC)	11	5	3

Gezgin Satıcı Problemi (TSP): Alt Sınır

- Her şehri bir kez ziyaret edip başlangıca dönen **en kısa tur** (en küçükleme, NP-zor).
- Düğüm = 0'dan başlayan kısmi yol; çocuklar = ziyaret edilmemiş bir sonraki şehir.
- Alt sınır:** henüz **çıkılmamış** her şehirden (bulunulan şehir + ziyaret edilmemişler) en az bir kenarla çıkılmak zorundadır:

	0	1	2	3	min
0	—	3	2	15	2
1	3	—	21	5	3
2	2	21	—	8	2
3	15	5	8	—	5

4 şehirli simetrik uzaklık matrisi d.

$$\text{alt}(\text{yol}) = \text{maliyet}(\text{yol}) + \text{enKucuk}[\text{son şehir}] + \sum_{v \text{ ziyaret edilmemiş}} \text{enKucuk}[v]$$

Kök: $2 + 3 + 2 + 5 = 12$. Yol $0 \rightarrow 1$: $3 + 3 + (2 + 5) = 13$. Her gerçek turun maliyeti bu toplamdan küçük olamaz, bu yüzden sınır geçerlidir.

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```
▶ void tsp(int sehir, int maliyet, int sayac) {  
  2  if (sayac == n) {           // tur tamam  
  3      int tur = maliyet + d[sehir][0];  
  4      if (tur < enIyi) enIyi = tur;  
  5      return;  
  6  }  
  7  for (int s = 0; s < n; s++) {  
  8      if (ziyaret[s]) continue;  
  9      int yeni = maliyet + d[sehir][s];  
 10      ziyaret[s] = true;  
 11      if (altSinir(s, yeni) < enIyi) // sınır  
 12          tsp(s, yeni, sayac + 1);  
 13      ziyaret[s] = false;      // geri al  
 14  }  
 15 }
```

0
a=12

Şehir 0'dan başlanır. Kökün alt sınırı = 2 + 3 + 2 + 5 = **12**.

Adım 1 / 13

ÇAĞRI YIĞINI

tsp(0, 0, 1)

DEĞİŞKENLER

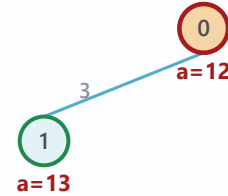
sehir = 0 maliyet = 0 sayac = 1 s = -
yeni = - alt = - enIyi = ∞

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ▶ if (altSinir(s, yeni) < enIyi) // sınır
12    ▶    tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```



0 → 1: yeni = 0 + 3 = 3; alt = 13 < ∞ → **dallan.**

Adım 2 / 13

ÇAĞRI YIĞINI

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 0

maliyet = 0

sayac = 1

s = 1

yeni = 3

alt = 13

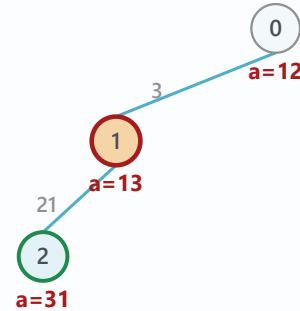
enIyi = ∞

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) {           // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ▶ if (altSinir(s, yeni) < enIyi) // sınır
12    ▶    tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false;      // geri al
14  }
15 }

```



1 → 2: yeni = 3 + 21 = 24; alt = 31 < ∞ → **dallan.**

Adım 3 / 13

ÇAĞRI YIĞINI

tsp(1, 3, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 1

maliyet = 3

sayac = 2

s = 2

yeni = 24

alt = 31

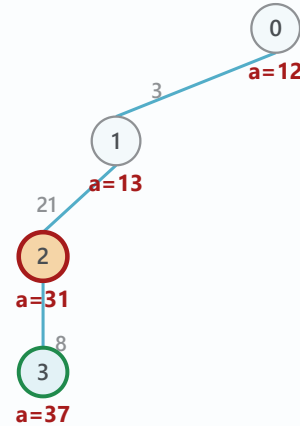
enIyi = ∞

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) {           // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ▶ if (altSinir(s, yeni) < enIyi) // sınır
12    ▶    tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false;      // geri al
14  }
15 }

```



2 → 3: yeni = 24 + 8 = 32; alt = 37 < ∞ → **dallan.**

Adım 4 / 13

ÇAĞRI YIĞINI

tsp(2, 24, 3)

tsp(1, 3, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 2 maliyet = 24 sayac = 3 s = 3

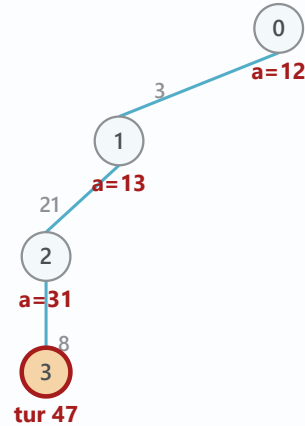
yeni = 32 alt = 37 enIyi = ∞

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     ▶ int tur = maliyet + d[sehir][0];
4     ▶ if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s] continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    if (altSinir(s, yeni) < enIyi) // sınır
12      tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```



Tur $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0 = 32 + 15 = 47 < \infty \rightarrow \text{enIyi} = 47$.

Adım 5 / 13

ÇAĞRI YIĞINI

tsp(3, 32, 4)

tsp(2, 24, 3)

tsp(1, 3, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 3

maliyet = 32

sayac = 4

s = -

yeni = -

alt = -

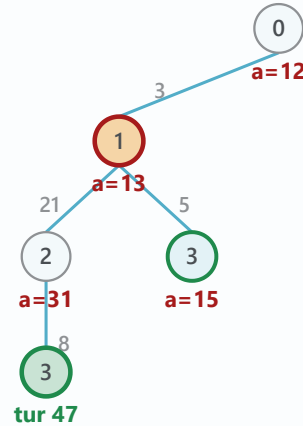
enIyi = 47

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ▶ if (altSinir(s, yeni) < enIyi) // sınır
12    ▶    tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```



1 → 3: yeni = 3 + 5 = 8; alt = 15 < 47 → **dallan**.

Adım 6 / 13

ÇAĞRI YIĞINI

tsp(1, 3, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 1

maliyet = 3

sayac = 2

s = 3

yeni = 8

alt = 15

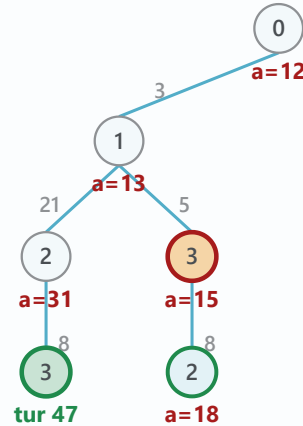
enIyi = 47

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) {           // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ▶ if (altSinir(s, yeni) < enIyi) // sınır
12    ▶    tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false;      // geri al
14  }
15 }

```



3 → 2: yeni = 8 + 8 = 16; alt = 18 < 47 → **dallan**.

Adım 7 / 13

ÇAĞRI YIĞINI

tsp(3, 8, 3)

tsp(1, 3, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 3

maliyet = 8

sayac = 3

s = 2

yeni = 16

alt = 18

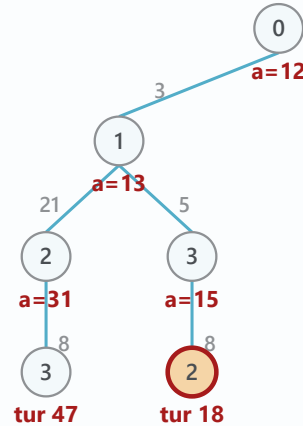
enIyi = 47

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     ▶ int tur = maliyet + d[sehir][0];
4     ▶ if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s] continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    if (altSinir(s, yeni) < enIyi) // sınır
12      tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```



Tur 0 → 1 → 3 → 2 → 0 = 16 + 2 = **18** < 47 → enİyi = 18.

Adım 8 / 13

ÇAĞRI YIĞINI

tsp(2, 16, 4)

tsp(3, 8, 3)

tsp(1, 3, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 2 maliyet = 16 sayac = 4 s = -

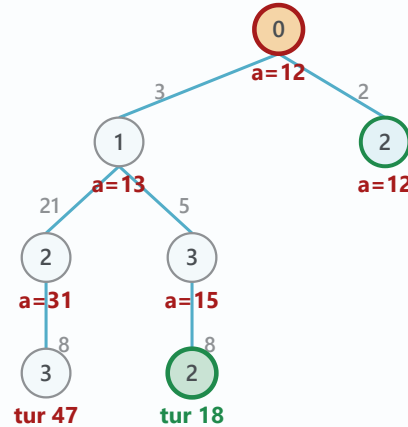
yeni = - alt = - enIyi = 18

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) {           // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ▶ if (altSınir(s, yeni) < enIyi) // sınır
12    ▶    tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false;       // geri al
14  }
15 }

```



0 → 2: yeni = 0 + 2 = 2; alt = 12 < 18 → **dallan**.

Adım 9 / 13

ÇAĞRI YIĞINI

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 0

maliyet = 0

sayac = 1

s = 2

yeni = 2

alt = 12

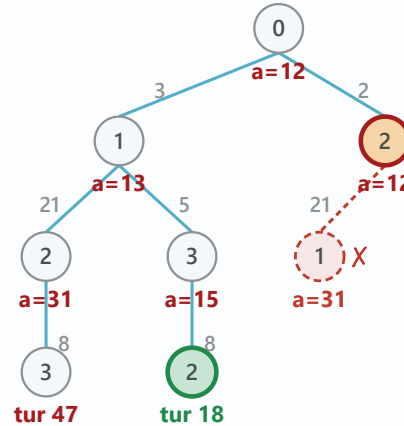
enIyi = 18

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ► if (altSinir(s, yeni) < enIyi) // sınır
12      tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```



2 → 1: yeni = 23; alt = 31 ≥ enIyi = 18 → **budandı**.

Adım 10 / 13

ÇAĞRI YIĞINI

tsp(2, 2, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 2

maliyet = 2

sayac = 2

s = 1

yeni = 23

alt = 31

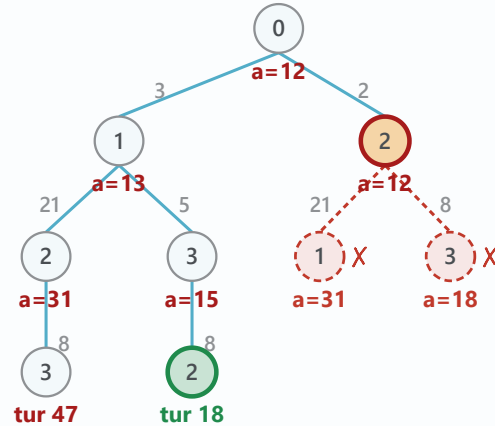
enIyi = 18

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ► if (altSinir(s, yeni) < enIyi) // sınır
12      tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```



2 → 3: yeni = 10; alt = 18 ≥ enIyi = 18 → **budandı.**

Adım 11 / 13

ÇAĞRI YIĞINI

tsp(2, 2, 2)

tsp(0, 0, 1)

DEĞİŞKENLER

sehir = 2

maliyet = 2

sayac = 2

s = 3

yeni = 10

alt = 18

enIyi = 18

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

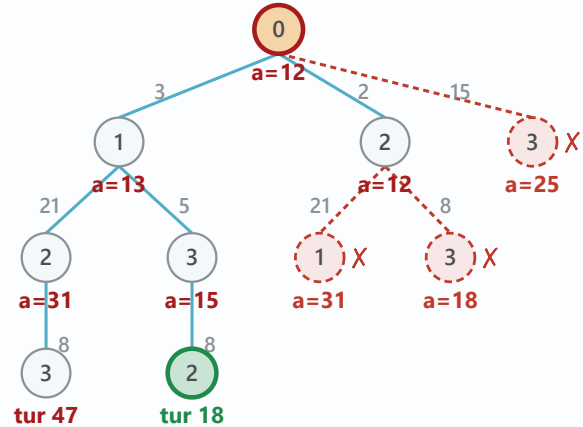
```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    ► if (altSinir(s, yeni) < enIyi) // sınır
12      tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```

ÇAĞRI YIĞINI

tsp(0, 0, 1)



0 → 3: yeni = 15; alt = 25 ≥ enIyi = 18 → **budandı**.

Adım 12 / 13

DEĞİŞKENLER

sehir = 0 maliyet = 0 sayac = 1 s = 3
 yeni = 15 alt = 25 enIyi = 18

tsp(0, 0, 1): Sınırlı Derinlik Öncelikli Arama

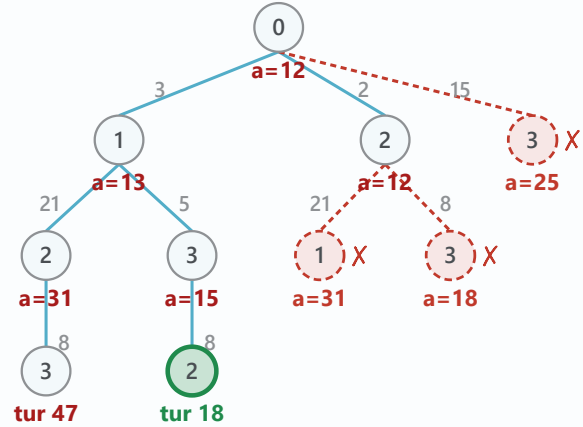
```

1 void tsp(int sehir, int maliyet, int sayac) {
2   if (sayac == n) { // tur tamam
3     int tur = maliyet + d[sehir][0];
4     if (tur < enIyi) enIyi = tur;
5     return;
6   }
7   for (int s = 0; s < n; s++) {
8     if (ziyaret[s]) continue;
9     int yeni = maliyet + d[sehir][s];
10    ziyaret[s] = true;
11    if (altSinir(s, yeni) < enIyi) // sınır
12      tsp(s, yeni, sayac + 1);
13    ziyaret[s] = false; // geri al
14  }
15 }

```

ÇAĞRI YİĞİNİ

boş



Bitti: en kısa tur $0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 0 = 18$; 16 yerine 10 düğüm.

Adım 13 / 13

DEĞİŞKENLER

sehir = 0

maliyet = 0

sayac = 1

s = -

yeni = -

alt = -

enIyi = 18

Daha Büyük Örnek: 6 Şehir

	0	1	2	3	4	5	min
0	—	29	20	21	16	31	16
1	29	—	15	29	28	40	15
2	20	15	—	15	14	25	14
3	21	29	15	—	12	18	12
4	16	28	14	12	—	22	12
5	31	40	25	18	22	—	18

- Budamasız geri izleme tüm $(n-1)! = 120$ turu dener; **altSınır** testiyle aynı sonuç çok daha az düğümle bulunur.
- Kök alt sınırı: $16 + 15 + 14 + 12 + 12 + 18 = \mathbf{87}$; optimum **115**.
- En iyi tur: $\mathbf{0} \rightarrow \mathbf{1} \rightarrow \mathbf{2} \rightarrow \mathbf{3} \rightarrow \mathbf{5} \rightarrow \mathbf{4} \rightarrow \mathbf{0} = 29 + 15 + 15 + 18 + 22 + 16 = 115$.

Sürüm	Üretilen düğüm	tsp çağrısı
Budamasız (geri izleme)	326	326
En ucuz çıkış sınırı	210	130

Daha sıkı sınırlar

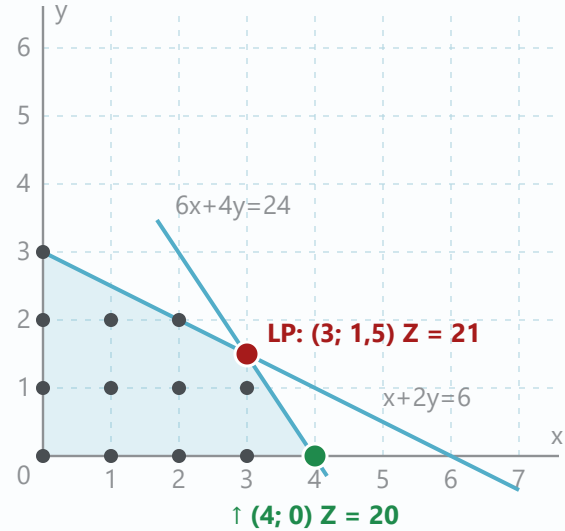
Simetrik TSP'de her şehrin **iki** en kısa kenarının toplamının yarısı; ya da satır/sütun **indirgenmiş maliyet matrisi** (Little ve ark., 1963). Sıkı sınır daha çok budar ama her düğümde daha pahalıdır.

Tam sayılı Doğrusal Programlama (ILP)

- Doğrusal programın tüm değişkenleri **tam sayı** olmalı. Dal sınırın en yaygın endüstriyel kullanımı (Land ve Doig, 1960).
- **Sınır:** tam sayı koşulunu kaldır, **LP gevşetmesini** çöz. LP optimumu, ILP optimumu için üst sınırdır.
- **Dallanma:** kesirli bir değişken ($y = 1,5$) için iki alt problem: $y \leq 1$ ve $y \geq 2$. Aradaki kesirli bölge atılır, hiçbir tam sayı nokta kaybolmaz.

Örnek: en büyükle $Z = 5x + 4y$; $6x + 4y \leq 24$, $x + 2y \leq 6$, $x, y \geq 0$ tam sayı.

İkinci örnek: $Z = 3x + 2y$; $x + y \leq 4$, $x \leq 3$, $y \leq 2$. Kaba kuvvet $0 \leq x \leq 3$, $0 \leq y \leq 2$ ızgarasındaki 12 noktanın hepsini dener $\rightarrow (3, 1)$, $Z = 11$. Bu örnekte LP gevşetmesi de $(3, 1)$ verir: dal sınır **kökte** biter.



Gölge: LP olurlu bölgesi; noktalar: tam sayı çözümler.

ILP: LP Gevşetmeleriyle Dallanma

$$x=3, y=1,5$$

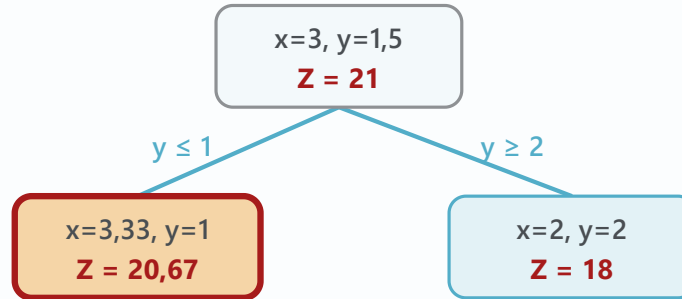
$$Z = 21$$

Kök: tam sayı koşulu gevşetilir (LP gevşetmesi). LP optimumu $(3; 1,5)$, $Z = 21 \rightarrow$ üst sınır. y kesirli $\rightarrow y$ üzerinde dallan.

Adım 1 / 5

Dikkat: LP optimumu $(3; 1,5)$ yuvarlanırsa $(3; 1) \rightarrow Z = 19$ ya da $(3; 2) \rightarrow$ olursuz olur. Gerçek optimum $(4; 0)$ yuvarlamayla bulunamaz.

ILP: LP Gevşetmeleriyle Dallanma

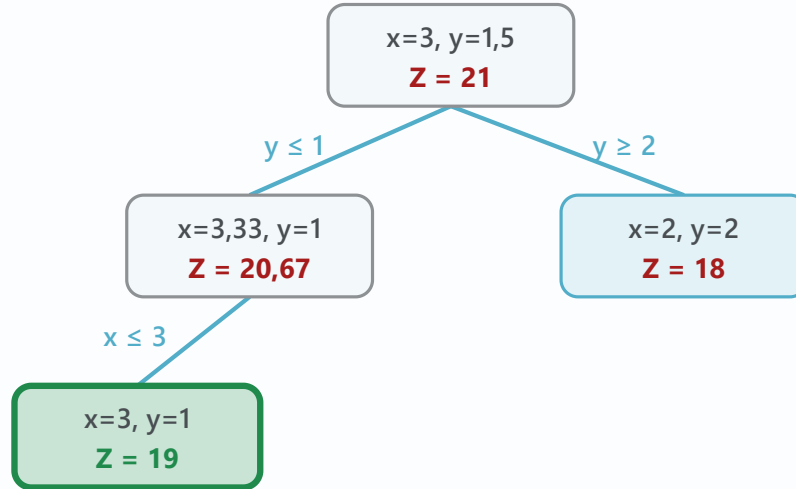


Dal $y \leq 1$: LP optimumu $(3,33; 1)$, $Z = 20,67$. x kesirli $\rightarrow x \leq 3$ ve $x \geq 4$ dalları açılır.

Adım 2 / 5

Dikkat: LP optimumu $(3; 1,5)$ yuvarlanırsa $(3; 1) \rightarrow Z = 19$ ya da $(3; 2) \rightarrow$ olursuz olur. Gerçek optimum $(4; 0)$ yuvarlamayla bulunamaz.

ILP: LP Gevşetmeleriyle Dallanma

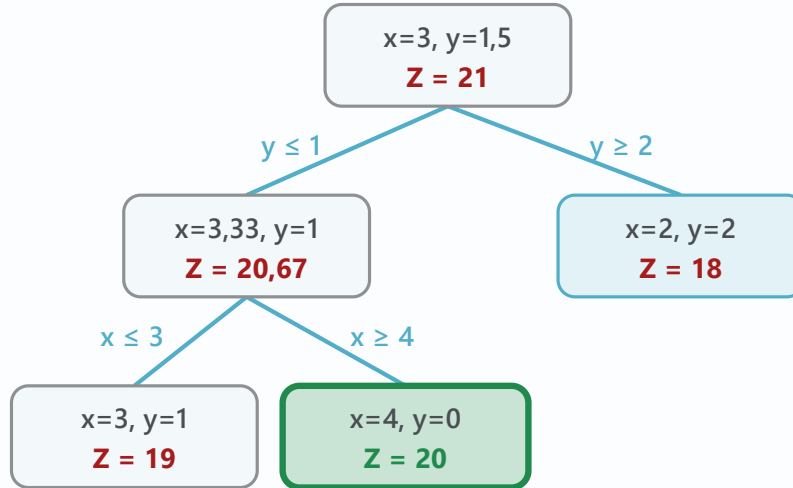


Dal $x \leq 3$: LP çözümü (3; 1) **tam sayı** → ilk aday çözüm, eniyi = 19.

Adım 3 / 5

Dikkat: LP optimumu (3; 1,5) yuvarlanırsa (3; 1) → $Z = 19$ ya da (3; 2) → olursuz olur. Gerçek optimum (4; 0) yuvarlamayla bulunamaz.

ILP: LP Gevşetmeleriyle Dallanma

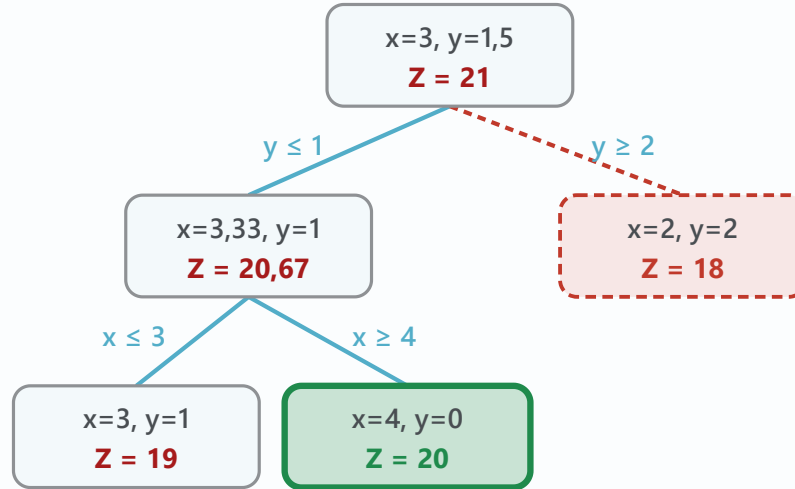


Dal $x \geq 4$: LP çözümü (4; 0) tam sayı, $Z = 20 > 19 \rightarrow$ **enişi = 20**.

Adım 4 / 5

Dikkat: LP optimumu (3; 1,5) yuvarlanırsa (3; 1) $\rightarrow Z = 19$ ya da (3; 2) $\rightarrow$ olursuz olur. Gerçek optimum (4; 0) yuvarlamayla bulunamaz.

ILP: LP Gevşetmeleriyle Dallanma

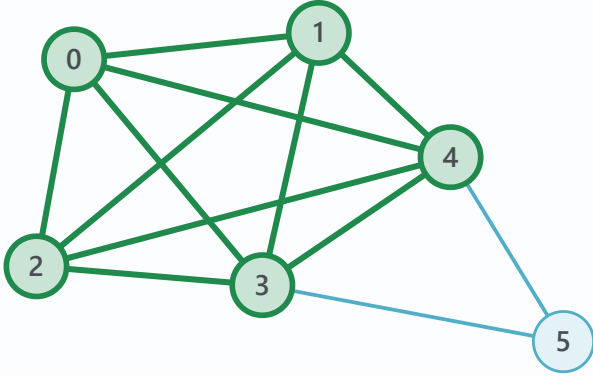


Dal $y \geq 2$: LP sınırı $Z = 18 \leq 20 \rightarrow$ **budandı**. Optimum: $x = 4, y = 0, Z = 20$.

Adım 5 / 5

Dikkat: LP optimumu $(3; 1,5)$ yuvarlanırsa $(3; 1) \rightarrow Z = 19$ ya da $(3; 2) \rightarrow$ olursuz olur. Gerçek optimum $(4; 0)$ yuvarlamayla bulunamaz.

Maksimum Klik (Maximum Clique)



6 düğümlü örnek çizge; en büyük klik {0, 1, 2, 3, 4}.

- **Klik:** her iki düğümü komşu olan düğüm kümesi.
- **Sınır:** $|K| + |aday| \leq |enIyi|$ ise mevcut klik ne kadar büyütülse de enIyi'yi geçemez.

```
void klik(List<Integer> K, int bas) {
    if (K.size() > enIyi.size())
        enIyi = new ArrayList<>(K);
    List<Integer> aday = new ArrayList<>();
    for (int v = bas; v < n; v++)
        if (uygun(v, K)) aday.add(v); // K'ye komşu
    if (K.size() + aday.size() <= enIyi.size())
        return; // sınır
    for (int v : aday) {
        K.add(v);
        klik(K, v + 1);
        K.remove(K.size() - 1); // geri al
    }
}
```

Sürüm	klik çağrısı
Sınırsız geri izleme	36
$ K + aday $ sınırlı	17

Dal Sınırla Çözülen Diğer Problemler

Hamilton yolu

Her düğümü tam bir kez ziyaret eden yol; ağırlıklı sürümü TSP'ye çok benzer sınırlarla çözülür.

Boole sağlanabilirlik (SAT)

Değişkenlere doğruluk değeri ata; çelişen kısmi atamaları buda (DPLL). En büyük SAT için sınır: sağlanabilecek en fazla tümce.

Çizelgeleme

En büyük yükü (makespan) ya da gecikmeyi en küçükke; sınır: kalan işlerin en iyi durumdaki yükü.

Kutu paketleme

Eşyaları en az kutuya yerleştir; alt sınır $\lceil \sum \text{boyut} / \text{kutu kapasitesi} \rceil$.

Maksimum kapsama

En fazla öğeyi kapsayan k küme seç; sınır: kalan en büyük kümelerin ekleyebileceği öğe sayısı.

Atama problemi

n işi n kişiye en düşük maliyetle ata; alt sınır: atanmamış her satırın en küçük maliyeti.

Not: Maksimum akış gibi polinom zamanlı algoritması olan (Bölüm 4) problemlerde dal sınıra gerek yoktur; yöntem NP-zor problemler için anlamlıdır.

ÖZET

Özet

Konu	Hatırlanacaklar
Paradigma	durum uzayı ağacı; seç → dallan → sınırla → buda; eniyi en iyi tam çözümü tutar
Sınır	geçerli (iyimser), sıkı, ucuz; gevşetme ile kurulur. En büyüklemede üst, en küçüklemede alt sınır
Budama	olursuzluk, sınır $\leq$ eniyi (ya da alt $\geq$ eniyi), tam çözüm / tam sayı LP
Stratejiler	FIFO (BFS, kuyruk), LIFO (DFS, yığın, az bellek), LC / en iyi önce (öncelikli kuyruk, az düğüm)
Sırt çantası	kesirli sırt çantası üst sınırı; örnekte sınır(kök) = 115, optimum 90; 31 yerine 11 düğüm
TSP	alt sınır = maliyet + çıkılmamış şehirlerin en ucuz çıkış kenarları; 4 şehirde 16 → 10 düğüm
ILP / klik	LP gevşetmesi + kesirli değişkende dallanma; klikte $ K + aday \leq eniyi $ sınırı
Karmaşıklık	en kötü durumda üstel; pratik başarı sınırın kalitesine bağlı

Etkileşimli Simülatörler

NP-Tamlık ve Hesaplamalı Zorluk

P, NP, NP-zor ve NP-tam sınıflarını, polinom indirgemeyi ve SAT/TSP örneklerini inceleyin; doğru/yanlış sorularıyla kendinizi sınavın. Dal sınırın neden NP-zor problemler için kullanıldığını görün.

Gezgin Satıcı Problemi

En fazla 10 rastgele şehir üretin; hızlı en yakın komşu sezgiselini, tüm turları deneyen kaba kuvvet ile karşılaştırın. Dal sınırın budadığı aramanın kaba kuvvete göre ne kazandırdığını düşünün.

ÖZET

Sık Yapılan Hatalar

Geçersiz (kötümser) sınır

Sırt çantasında yalnızca tam sığan eşyaları toplamak üst sınır değildir; optimumu içeren dal budanabilir.

Yönü karıştırmak

En büyüklemede $\text{sinir} \leq \text{enliyi}$, en küçüklemede $\text{alt} \geq \text{enliyi}$ ise budanır. Tersini yazmak ya optimumu kaçıır ya hiç budamaz.

Kuyruktan çıkınca tekrar sınamamak

Düğüm eklendiğinde umut vaat ediyordu; beklerken enliyi büyüdüyse artık gereksizdir (örnekte C ve G).

Eşyaları sıralamamak

Kesirli sınır, eşyalar p_i/w_i 'ye göre azalan sırada değilse geçerli bir üst sınır vermez.

enliyi'yi yalnızca yapraklarda güncellemek

Sırt çantasında her olurlu düğüm bir çözümdür; erken güncelleme daha çok budama sağlar.

Geri izleme ile aynı sanmak

Sınır testi olmayan kod (ör. budamasız TSP araması) dal sınır değildir; yalnızca tüm ağacı gezer.

ÖZET

Örnek Problemler

Sırt çantası: $(p, w) = (20, 2), (30, 5), (35, 7), (12, 3); W = 9$ (oranlar 10, 6, 5, 4).

1. Kökün kesirli üst sınırını hesaplayın.
2. Oranla açgözlü seçim ve optimum kazanç nedir?
3. "Eşya 1 alınmadı" düğümünün $(0, 0)$ sınırı nedir? Optimum bulunduktan sonra budanır mı?
4. 4 şehirli TSP örneğinde $0 \rightarrow 3 \rightarrow 1$ yolunun alt sınırı nedir?
5. Maksimum klikte $K = \{3, 4\}$, aday = $\{5\}$, $|eniyi| = 5$ iken ne olur?
6. Belleği çok kısıtlı bir sistemde hangi arama stratejisini seçersiniz?

► Cevapları göster

Alıştırmalar

1. Sırt çantası kodunu FIFO kuyrukla yazın; bölümdeki örnekte kaç düğüm üretilir?
2. `tsp` metoduna en iyi turu kaydeden `int[] enIyiYol` ekleyin.
3. 4 şehirli örneğin kökünde "iki en kısa kenar toplamının yarısı" sınırını hesaplayın.
4. Maliyet matrisi $\begin{bmatrix} 9 & 2 & 7 \\ 6 & 4 & 3 \\ 5 & 8 & 1 \end{bmatrix}$ olan atama probleminin kök alt sınırı ve optimumu nedir?
5. Kesirli sırt çantası sınırının her zaman geçerli olduğunu kanıtlayın.
6. İkinci ILP örneği ($Z = 3x + 2y$; $x + y \leq 4$, $x \leq 3$, $y \leq 2$) LP gevşetmeli dal sınırla kaç düğümde çözülür?

► Cevaplar / ipuçları

Kaynaklar

- E. Horowitz, S. Sahni ve S. Rajasekaran, *Computer Algorithms*, 2. baskı, Silicon Press, 2008 (Bölüm 8: Branch-and-Bound).
- R. Neapolitan, *Foundations of Algorithms*, 5. baskı, Jones & Bartlett, 2015 (Bölüm 6).
- A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 3. baskı, Pearson, 2012 (Bölüm 12.2).
- A. H. Land ve A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems", *Econometrica* 28(3), 1960.
- J. D. C. Little, K. G. Murty, D. W. Sweeney ve C. Karel, "An Algorithm for the Traveling Salesman Problem", *Operations Research* 11(6), 1963.
- R. Carraghan ve P. M. Pardalos, "An Exact Algorithm for the Maximum Clique Problem", *Operations Research Letters* 9(6), 1990.

Son

Bölüm 14: Dal Sınır Algoritmaları

Dersin sonu · [Terim sözlüğü](#)