Bölüm 2: Oyunlar ve Temsiller

Algoritmik Oyun Kuramı

Temel kavramlar | Normal ve genişletilmiş form | Sınıflandırma | Mahkûmun İkilemi | Klasik 2×2 oyunlar | 2/3 ortalaması oyunu | Kombinatoryal oyunlar ve minimaks | Geriye doğru çıkarım | Mekanizma tasarımına bakış

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_02_Oyunlar_ve_Temsiller.html

TEMEL KAVRAMLAR

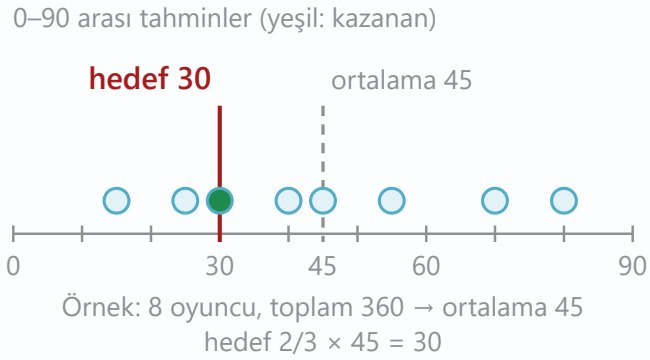
Oyunla Başlangıç: 2/3 Ortalaması Oyunu

Oyunun kuralı

Herkes 0 ile 90 arasında gizlice bir sayı (tam ya da ondalıklı) yazar; tüm tahminlerin ortalamasının 2/3'üne en yakın tahmin kazanır (eşitlikte ödül paylaşılır).

Merak sorusu

- Sizce bu sınıfta ortalamanın 2/3'ü kaç çıkar?
- Herkes sizin kadar dikkatli düşünürse hangi sayı kazanır?



[2/3 Ortalaması Oyunu →]

TEMEL KAVRAMLAR

Öğrenme Çıktıları

Kavramlar

Oyuncu, strateji, kazanç, bilgi, rasyonellik ve ortak bilgiyi bir örnek üzerinde tanımlamak.

Temsil ve sınıflandırma

Normal form (matris) ve genişletilmiş form (ağaç); oyunları zamanlama, anlaşma, toplam kazanç, simetri, bilgi ve ufuk boyutlarında ayırt etmek.

Denge ve verimlilik

Nash dengesini sezgisel olarak açıklamak; Mahkûmun İkilemi'nde ihanetin baskın olduğunu ve dengenin Pareto optimal olmadığını göstermek.

Klasik oyunlar

Altı klasik 2×2 oyunun saf dengelerini en iyi tepkilerle bulmak; çatışma ve koordinasyon yapısını yorumlamak.

Algoritmalar

Seviye-k akıl yürütmeyi açıklamak; minimaks ve Nim'in XOR stratejisiyle küçük oyunları elle ve kodla çözmek.

Sıralı oyunlar

Geriye doğru çıkarımla alt oyun mükemmel dengeyi bulmak; inandırıcı olmayan tehdidi ve sonlu ile sonsuz ufuk farkını yorumlamak.

TEMEL KAVRAMLAR

Oyun Kuramı (Game Theory) Nedir?

- Karar vericilerin **stratejik etkileşim** içinde olduğu durumları inceleyen matematiksel bir araçtır: benim en iyi kararım, senin ne yapacağına bağlıdır.
- Karar vericilere **oyuncu** (ajan) denir; nasıl davrandıklarını ve sonucun nasıl şekillendiğini açıklamaya çalışır.
- Oyuncuların **rasyonel** ve kendi çıkarlarına odaklı hareket ettiği varsayılır.
- Ekonomi, siyaset, biyoloji, psikoloji ve bilgisayar biliminde yaygın kullanılır.

Ekonomi
Şirketlerin fiyat rekabeti, açık artırmalar, pazarlık.

Bilgisayar ağları
Her paketin/sürücünün kendi yolunu seçtiği bencil yönlendirme.

Yapay zekâ
Çoklu ajan sistemleri, oyun oynayan programlar, öğrenen ajanlar.

Algoritmik oyun kuramı iki soru ekler: Dengeyi *hesaplamak* ne kadar zor? Bencil ajanlar arasında iyi sonuç veren *kuralları* (algoritma, protokol, açık artırma) nasıl tasarlarız?

Kısa Tarihçe: Oyun Kuramından Algoritmik Oyun Kuramına

Yıl	Gelişme
1913	Zermelo: satranç gibi sonlu, mükemmel bilgili oyunlarda bir taraf kazanmayı ya da iki taraf beraberliği garanti eder.
1928	von Neumann: iki kişilik sıfır toplamlı oyunlar için minimaks teoremi.
1944	von Neumann ve Morgenstern, *Theory of Games and Economic Behavior*: oyun kuramı ayrı bir alan olur.
1950	Nash: her sonlu oyunda karışık stratejilerde bir denge vardır. Aynı yıl RAND'daki Flood–Dresher deneyi; Tucker Mahkûmun İkilemi hikâyesini anlatır.
1961	Vickrey: ikinci fiyat açık artırmasında dürüst teklif baskındır.
1994	Nash, Harsanyi ve Selten'e Nobel ekonomi ödülü.
1999	Koutsoupias–Papadimitriou (en kötü denge oranı; sonradan Anarşinin Bedeli adını aldı) ve Nisan–Ronen (algoritmik mekanizma tasarımı): algoritmik oyun kuramı doğar.
2006	Genel oyunlarda Nash dengesini hızlı bulan bir yöntem bilinmiyor: problem PPAD-tam denen zor problemler sınıfındadır (Daskalakis–Goldberg–Papadimitriou; iki oyuncu: Chen–Deng).
2012, 2020	Nobel: Roth ve Shapley (kararlı eşleştirme, piyasa tasarımı); Milgrom ve Wilson (açık artırma tasarımı).

TEMEL KAVRAMLAR

Bir Oyunun Bileşenleri

Oyuncular

Karar alan kişi veya kurumlar.

$N = \{1, \ldots, n\}$

Eylemler / Stratejiler

Her oyuncunun seçebileceği eylemler kümesi.

$S_i = \{\text{Taş}, \text{Kağıt}, \text{Makas}\}$

Kazanç (payoff)

Her sonucun oyuncu için değeri; *herkesin* seçimine bağlıdır.

$u_i(s_1, \ldots, s_n)$

Bilgi

Kim neyi, ne zaman biliyor? Rakibin hamlesini görüyor musun?

satranç ≠ poker

Kurallar ve sıra

Eşzamanlı mı, sırayla mı? Kazançları eylemlere bağlayan kurallar.

normal / genişletilmiş form

Sonuç

Tüm oyuncuların seçimlerinin bir araya gelmesiyle oluşur.

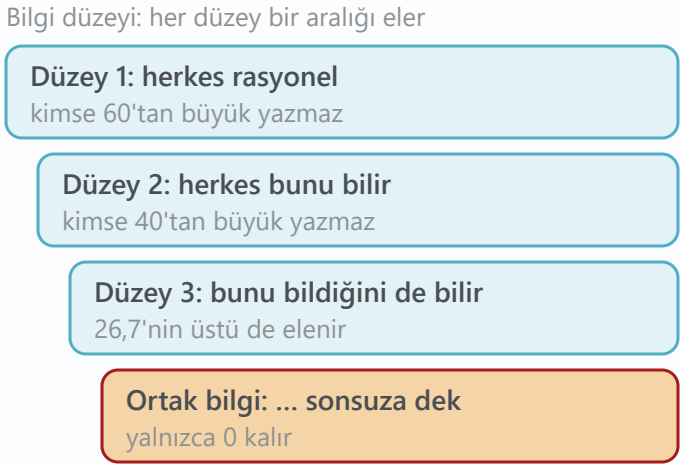
$s = (s_1, \ldots, s_n)$

Oyun: oyuncuların etkileşime girdiği, her birinin seçiminin herkesin kazancını etkilediği durum.

TEMEL KAVRAMLAR

Oyuncular ve Rasyonellik

- **Rasyonel**: Her oyuncu, inançlarına göre kendi beklenen faydasını maksimize eden seçimi yapar.
- **Kendi çıkarı**: Oyuncu kendi fayda fonksiyonunu izler; başkalarının kazancı ancak kendi faydasına girerse önemlidir.
- **Ortak bilgi** (common knowledge): Herkes rasyonel, herkes bunu biliyor, herkes herkesin bildiğini biliyor, ...

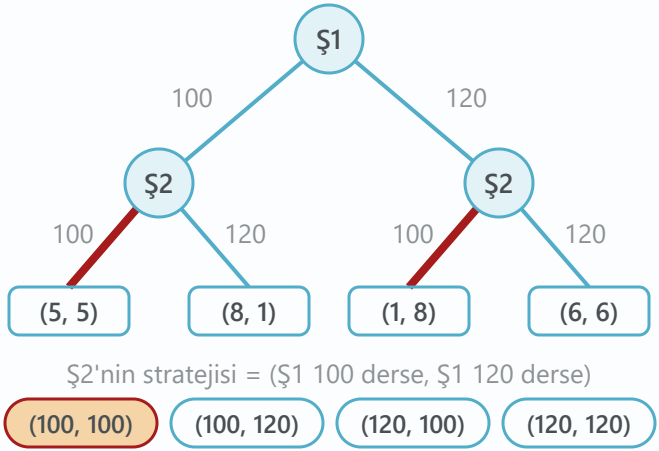


Rasyonel ≠ bencil: fayda fonksiyonu adalet, özgecilik ya da itibar kaygılarını da içerebilir. Sonuç, hiçbir oyuncunun tek başına kontrol edemediği bir **etkileşimin** ürünüdür.

TEMEL KAVRAMLAR

Eylem ve Strateji

- **Strateji**: Oyuncunun seçtiği eylem planı. Sıralı oyunlarda, *her* olası durumda ne yapacağını söyleyen *tam plan*dır.
- Sağda Ş2, Ş1'in fiyatını görüp yanıt veriyor. Ş2'nin 2 karar düğümü var ve her birinde 2 eylemden birini seçer: 2 · 2 = **4 strateji**, oysa yalnız 2 eylem var (3 düğüm olsaydı 2 · 2 · 2 = 8).
- Strateji, diğerlerinin davranışına ilişkin **beklentilere** dayanır.



Saf strateji (pure)

Tek bir planı kesin olarak seçmek.

`"her zaman Taş"`

Karışık strateji (mixed)

Saf stratejiler üzerinde olasılık dağılımı.

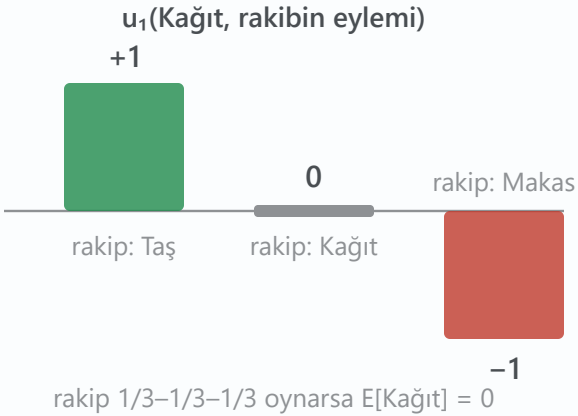
`(Taş 1/3, Kağıt 1/3, Makas 1/3)`

Strateji profili: her oyuncudan bir strateji, $s = (s_1, \ldots, s_n)$. Sonucu profil belirler.

TEMEL KAVRAMLAR

Kazanç (Payoff) ve Fayda Fonksiyonu

- Oyuncunun, **tüm oyuncuların** seçtiği strateji profiline bağlı olarak elde ettiği değer; fayda fonksiyonu u_i ile ölçülür.
- Oyuncular stratejilerini kazançlarını maksimize etmek için seçer.
- Para, puan, hapis yılı (eksi işaretle) ya da memnuniyet düzeyi olabilir.
- Karışık stratejilerde oyuncu **beklenen** kazancını hesaplar.



Aynı eylemin (Kağıt) kazancı rakibin eylemine göre +1, 0 ya da −1 olur: kazanç bir **profilin** fonksiyonudur, u_1(Kağıt, Taş) = +1.

Sıralı (Ordinal) ve Kardinal Fayda

Sıralı fayda yalnızca tercih sırasını söyler (Opera ≻ Futbol ≻ Evde; ≻: "tercih edilir"). Sayıları sırayı bozmadan değiştirmek (3, 2, 0 yerine 30, 20, 0) saf strateji analizini, baskınlığı ve saf dengeleri değiştirmez. **Kardinal fayda** (von Neumann–Morgenstern) farkların büyüklüğünü de taşır: piyango ve karışık stratejiler **beklenen fayda** ile karşılaştırıldığı için sayıların kendisi önemlidir.

Örnek

Ayşe için kesin Futbol mu, yoksa ½ Opera + ½ Evde piyangosu mu?

Fayda (Opera, Futbol, Evde)	Piyango	Kesin Futbol	Seçim
(3, 2, 0)	½·3 + ½·0 = 1,5	2	Futbol
(3, 1, 0): aynı sıra	1,5	1	**Piyango**
2u + 1 = (7, 5, 1)	4	5	Futbol

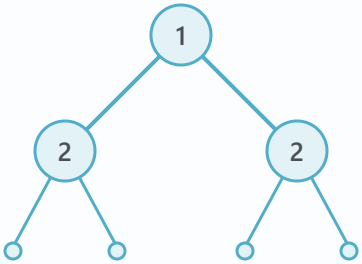
Sıra aynı kalsa da sayılar değişince piyango kararı değişebilir; a·u + b (a > 0) dönüşümü ise kararı korur.

Sık hata: "Kazanç 4, kazanç 2'nin iki katı iyidir" demek ya da karışık strateji hesabında sıralı sayıları kullanmak. Beklenen kazanç hesabı, faydaların kardinal (vNM) olduğunu varsayar; yalnızca sıra biliniyorsa saf strateji analiziyle yetinilir.

TEMEL KAVRAMLAR

Bilgi: Kim Neyi Biliyor?

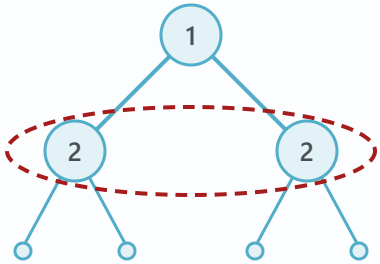
- Her oyuncunun oyun, kazançlar ve yapılan hamleler hakkında sahip olduğu bilgi; stratejiyi ve sonucu doğrudan etkiler.



2, 1'in seçimini görür

Mükemmel bilgi

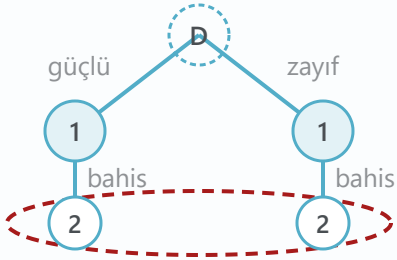
satranç, XOX, Nim



kesikli: bilgi kümesi (2 ayırt edemez)

Eşzamanlı hamle

TKM, fiyat rekabeti



D: Doğa kartı dağıtır; 2 eli görmez

Mükemmel olmayan bilgi

poker, açık artırma

Kesikli çizgi **bilgi kümesi**dir: oyuncu bu düğümlerden hangisinde olduğunu bilemez. **Mükemmel bilgi**, geçmiş hamlelerin hepsini görmek demektir; geleceği bilmek değil.

Örnek: Poker ve Blöf Stratejileri

Oyun kurulumu (Texas Hold'em)

- Her oyuncunun 2 kapalı kartı var; ortadaki açık kartlarla en iyi eli oluşturmaya çalışır.
- Rakibin kapalı kartlarını göremezsin: bilgi gizlidir.
- Güçlü bir el gördüğünde bile, rakiplerin ellerini bahislerinden tahmin etmeye çalışırsın.

Oyun kuramı yönü

- Rakibin davranışı (bahis, bekleme) ona dair **bilgi** taşır.
- **Blöf**, elin gücünü gizlemek için yanıltıcı sinyal göndermektir.
- Yalnızca güçlü elde bahis yapan oyuncu kolayca okunur.

Blöf aslında bir **karışık strateji**dir: zayıf elde de belirli bir olasılıkla bahis yapmak, rakibin sizi okumasını engeller. Bilgisayar poker programları (ör. Libratus, 2017) bu dengeleri hesaplayarak profesyonelleri yendi.

Normal Form (Stratejik Form)

- Oyuncuların **eşzamanlı** (ya da birbirinin seçimini bilmeden) karar verdiği durumları modeller.
- Oyuncular kümesi **N = {1, …, n}**; her oyuncunun strateji kümesi $\mathbf{S_i}$.
- **Strateji profili** (strategy profile): $s = (s_1, \ldots, s_n)$.
- **Fayda fonksiyonu**: $u_i : S_1 \times \ldots \times S_n \to \mathbb{R}$.
- İki oyuncuda **kazanç matrisi**: satır = oyuncu 1, sütun = oyuncu 2, hücre = (u_1, u_2).

Oyuncu 1 \ Oyuncu 2	Sol	Sağ
Üst	(4 , 3)	(1 , 2)
Alt	(3 , 1)	(2 , 4)

S_1 = {Üst, Alt}, S_2 = {Sol, Sağ}

vurgulu profil s = (Üst, Sağ): $u_1 = 1$, $u_2 = 2$

$G = (N, S_1, S_2, u_1, u_2)$, $N = \{1, 2\}$

Örnek: Taş-Kağıt-Makas Normal Formda

- İki oyuncu (n = 2), **aynı anda** Taş, Kağıt ya da Makas seçer. Taş makası, makas kağıdı, kağıt taşı yener.

Oyuncu 1 \ Oyuncu 2	Taş	Kağıt	Makas
Taş	(0 , 0)	(−1 , 1)	(1 , −1)
Kağıt	(1 , −1)	(0 , 0)	(−1 , 1)
Makas	(−1 , 1)	(1 , −1)	(0 , 0)

Kazanan +1
Kaybeden −1
Beraberlik 0
Toplam her hücrede 0 (sıfır toplamlı oyun)

Hiçbir hücrede iki oyuncu da seçiminden memnun kalmaz: hep biri seçimini değiştirmek ister.

OYUNLARIN TEMSİLİ

Taş-Kağıt-Makas'ta Strateji Seçimi

1. Rastgele seçim

Her turda her sembolü 1/3 olasılıkla seç.
Hiç sömürülemez: rakip ne yaparsa yapsın beklenen kazanç 0.

2. Geçmişe göre seçim

Rakip son turda Taş dediyse bu tur Kağıt de.
Örüntülü rakibe karşı kazandırır; ama bu kural da bir örüntüdür ve rakip onu sömürebilir.

3. Dengesiz karışık strateji

Ör. yarı zamanda Kağıt, yarı zamanda Makas.
Rakip hep Makas derse rakibin beklenen kazancı **+1/2**, seninki −1/2 olur: bu karışım **sömürülebilir**.

Basit bir oyunda bile strateji seçimi önemlidir. Tahmin edilemez olmak yetmez; doğru olasılıklarla tahmin edilemez olmak gerekir.

Sıfır Toplamlı Oyunlar

- Her profilde kazançların toplamı sıfır: $\mathbf{u_1(s) + u_2(s) = 0}$. Birinin kazancı, diğerinin kaybıdır.
- Toplam sabitse (ör. bir puan paylaşılıyorsa) aynı analiz geçerli: *sabit toplamlı* oyun.
- Örnek: TKM, Yazı-Tura, satranç (+1 / 0 / −1). Değil: Mahkûmun İkilemi, fiyat rekabeti.
- Tek matris yeter: satır oyuncusu en kötü durumunu en iyi yapar (**maksimin**); sütun oyuncusu satırın en iyi durumunu en küçük yapar (**minimaks**). (Oyun ağacındaki **minimaks arama**, aynı fikri sıralı oyunda yapraktan köke uygulayan algoritmadır.)

Eşleştiren \ Ayrıştıran	Yazı	Tura
Yazı	(1 , −1)	(−1 , 1)
Tura	(−1 , 1)	(1 , −1)

$u_2 = -u_1$ olduğundan yalnız u_1 yazılır

	Yazı	Tura
Yazı	1	−1
Tura	−1	1

A

Sıfır toplamlı oyunda **ortak çıkar yoktur**: saf rekabet. Sıfır toplamlı *olmayan* oyunlarda ise herkesin kazandığı (ya da kaybettiği) sonuçlar olabilir; işbirliği ile rekabet bir aradadır, iletişim ve müzakere sonucu değiştirebilir (ticaret müzakeresi, iklim anlaşmaları).

Örnek: Fiyat Rekabeti Oyunu

- İki şirket aynı pazarda benzer ürün satıyor; aynı anda fiyat belirliyor: 100 TL ya da 120 TL.
- Ucuz olan müşteriyi kapar; ikisi de pahalıysa kâr marjı yüksek kalır.
- Hücreler kâr birimi: (Şirket 1, Şirket 2).

	Şirket 2: 100 TL	**Şirket 2: 120 TL**
Şirket 1: 100 TL	**(5, 5)**	(8, 1)
Şirket 1: 120 TL	(1, 8)	(6, 6)

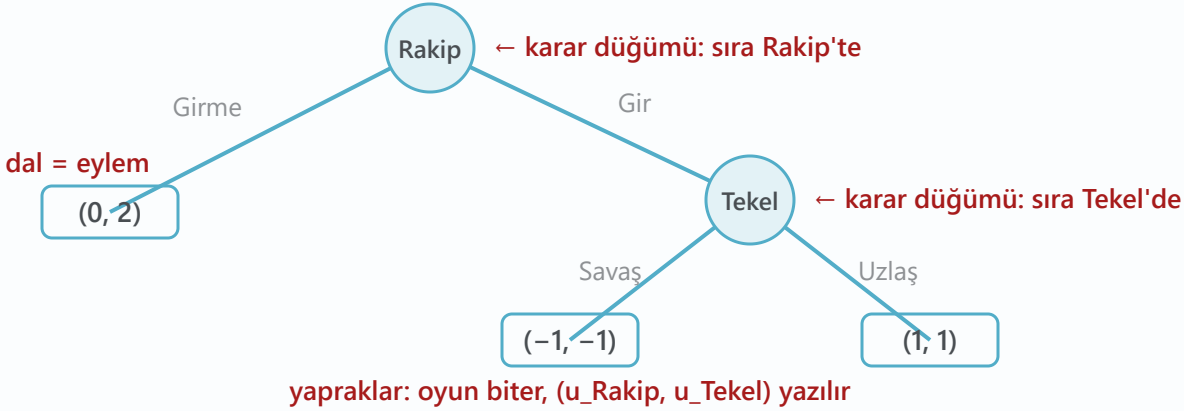
İkisi de 100 TL derse 5'er, ikisi de 120 TL derse 6'şar birim kâr.

Karşı taraf 100 derse 5 > 1, 120 derse 8 > 6: her şirket için **100 TL baskın**. Denge (100, 100) → 5'er kâr; oysa (120, 120) ikisine de 6 verirdi. Bu, Mahkûmun İkilemi'nin yapısıdır.

OYUNLARIN TEMSİLİ

Genişletilmiş Form (Oyun Ağacı)

- Kararların **sırasını** ve her anda kimin neyi bildiğini gösteren **oyun ağacı**; sıralı etkileşimleri (satranç, pazarlık, piyasaya giriş) doğal biçimde modeller.



Piyasaya giriş oyunu: Rakip (giren firma) pazara girer ya da girmez; girerse Tekel (mevcut firma) savaşır ya da uzlaşır. Ağaç, sırayı ve her karar anında kimin oynadığını açıkça gösterir.

Normal Formdan Oyun Ağacına: Adım Adım

Ş1 \ Şirket 2	100 TL	120 TL
100	(5, 5)	(8, 1)
120	(1, 8)	(6, 6)

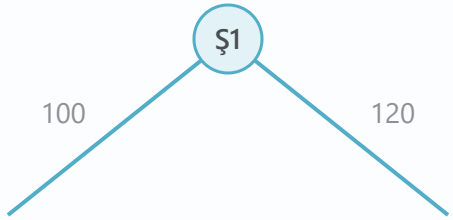
normal form

Başlangıç: fiyat rekabeti oyununun normal formu. Aynı oyunu bir **oyun ağacı** olarak çizeceğiz.

Normal Formdan Oyun Ağacına: Adım Adım

	Şirket 2	
Ş1	100 TL	120 TL
100	(5, 5)	(8, 1)
120	(1, 8)	(6, 6)

normal form

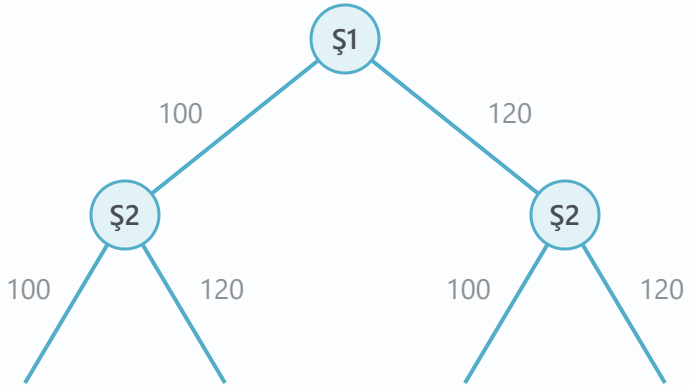


Kök düğüm: Şirket 1'in kararı. Her dal bir eylemdir: 100 TL ya da 120 TL.

Normal Formdan Oyun Ağacına: Adım Adım

	Şirket 2	
Ş1	100 TL	120 TL
100	(5, 5)	(8, 1)
120	(1, 8)	(6, 6)

normal form



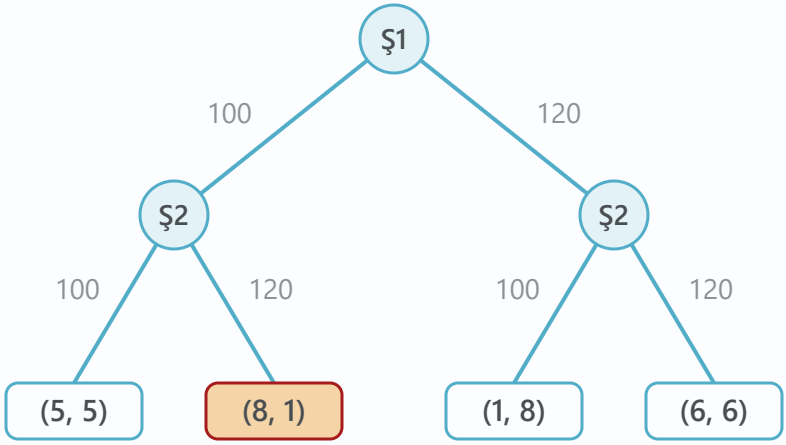
Her dalın sonunda Şirket 2'nin karar düğümü var; o da 100 ya da 120 seçer.

Normal Formdan Oyun Ağacına: Adım Adım

Şirket 2

Ş1	100 TL	120 TL
100	(5, 5)	(8, 1)
120	(1, 8)	(6, 6)

normal form

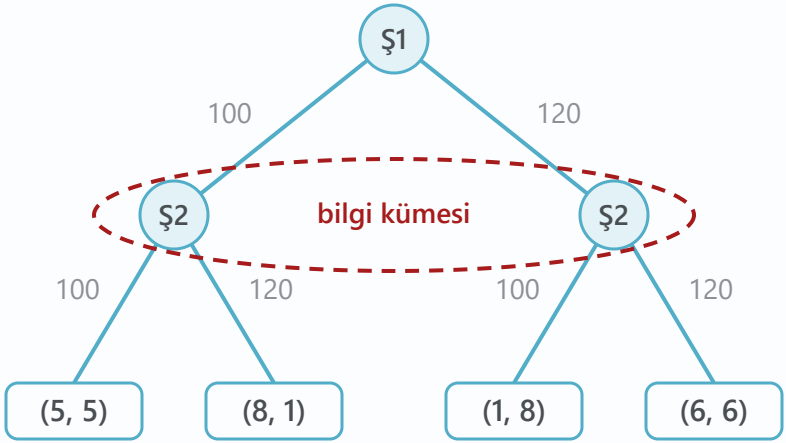


Yapraklar sonuçlardır: dört yol, matrisin dört hücresine karşılık gelir. Örn. 100–120 yolu (8, 1) hücresi.

Normal Formdan Oyun Ağacına: Adım Adım

Ş1 \ Şirket 2	100 TL	120 TL
100	(5, 5)	(8, 1)
120	(1, 8)	(6, 6)

normal form

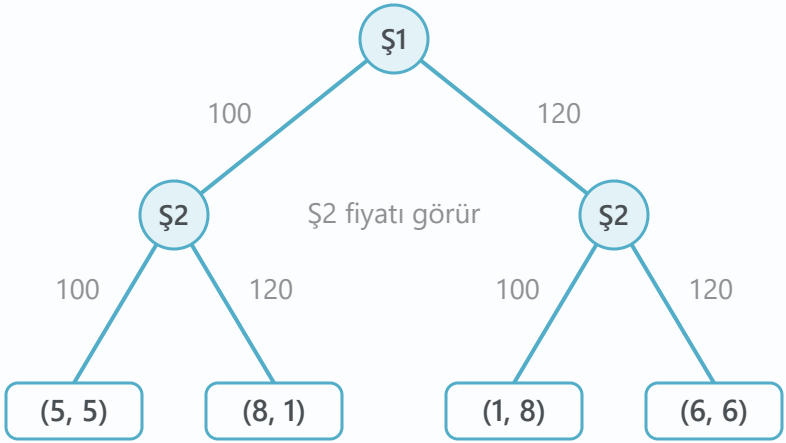


Ama iki şirket fiyatı **aynı anda** belirliyor: Ş2, Ş1'in seçimini görmez. Kesikli **bilgi kümesi** bunu gösterir; ağaç matrisle aynı oyundur.

Normal Formdan Oyun Ağacına: Adım Adım

Şirket 2		
Ş1	100 TL	120 TL
100	(5, 5)	(8, 1)
120	(1, 8)	(6, 6)

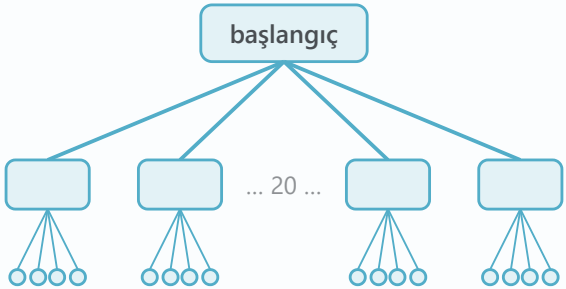
normal form



Bilgi kümesini kaldırırsak Ş2 önce Ş1'in fiyatını görür: artık **sıralı** bir oyun; geriye doğru çıkarımla çözülür.

Örnek: Satranç Bir Genişletilmiş Form Oyunudur

- İki oyuncu sırayla hamle yapar; her hamle bir dal, her konum bir düğümdür.
- **Mükemmel bilgi**: iki oyuncu da tahtayı ve tüm geçmiş hamleleri görür.
- Sonuç galibiyet, yenilgi **ya da beraberlik**tir (pat, üç kez tekrar, 50 hamle kuralı).
- Zermelo (1913): Bu tür sonlu oyunlarda ya bir taraf kazanmayı ya da iki taraf beraberliği garanti edebilir; ama ağaç çok büyük olduğu için hangisi olduğu bilinmiyor.



Beyaz: 20 hamle · Siyah: 20 cevap

$20 \times 20 = 400$ konum; tüm ağaç $\approx 10^{120}$ oyun

Satranç programları ağacın yalnızca bir kısmını arar (minimaks + budama). XOX, Nim ve “21'i Söyleme” gibi küçük oyunların ağacı ise tamamen çözülebilir.

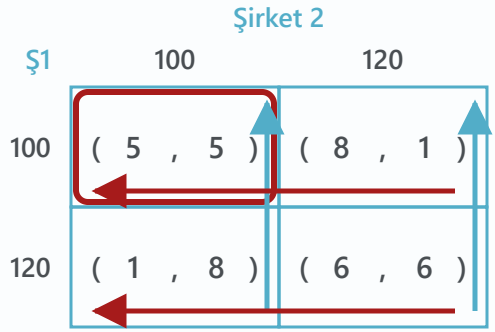
Oyunların Sınıflandırılması

Boyut	Bir uç	Diğer uç	Sorulan soru	Örnek
Zamanlama	Eşzamanlı	Sıralı	Rakibin hamlesini görerek mi oynuyorum?	kapalı zarf teklifi / piyasaya giriş
Anlaşma	İşbirlikçi	İşbirlikçi olmayan	Bağlayıcı anlaşma yapılabilir mi?	ortak girişim / Mahkûmun İkilemi
Toplam kazanç	Sıfır toplamlı	Sıfır toplamlı olmayan	Birinin kazancı diğerinin kaybı mı?	penaltı / iklim anlaşması
Simetri	Simetrik	Asimetrik	Oyuncuların yerini değiştirince oyun aynı mı?	Geyik Avı / Cinsiyetler Savaşı
Bilgi	Mükemmel bilgi	Mükemmel olmayan bilgi	Geçmiş hamlelerin hepsini görüyor muyum?	Dörtlü Bağla / poker
Ufuk	Sonlu	Sonsuz	Oyunun belirli bir sonu var mı?	Kırkayak / sürekli fiyat rekabeti

- **Tam bilgi** (complete) ayrı bir kavramdır: herkesin kazanç fonksiyonu herkesçe bilinir; bilinmiyorsa **eksik bilgi** vardır (Bayesçi oyunlar). **Simetri:** $u_1(a, b) = u_2(b, a)$; kazanç kimin oynadığına değil, stratejiye bağlı. **Ufuk:** sonsuz ufukta gelecek kazançlar $\delta \in (0, 1)$ ile tartılır: $\Sigma\, \delta^t\, u_t$.
- **Anlaşma:** işbirlikçi oyunda analiz birimi koalisyondur (Shapley değeri, çekirdek), işbirlikçi olmayanda tek oyuncu (Nash dengesi). Satranç: sıralı, işbirlikçi olmayan, sıfır toplamlı, mükemmel bilgili, sonlu.

Denge ve Nash Dengesi

- **En iyi tepki** (best response): diğer oyuncuların stratejileri (kısaca s_{-i}) sabitken oyuncu i'ye en yüksek kazancı veren strateji.
- **Nash dengesi**: herkesin stratejisinin diğerlerine en iyi tepki olduğu profil (s* ile gösterilir); hiç kimse *tek taraflı* sapmak istemez (Nash, 1950).



mavi ↑: Şirket 1 saparak kazanır · kırmızı ←: Şirket 2 sapar
okların hepsi (100, 100)'e akar: kimse ayrılmak istemez

Ne söyler?

Diğerleri sabitken sapmak kazandırmaz; kimse pişman değildir. Sapma okları dengeye çıkar, dengeden çıkmaz.

Ne söylemez?

Dengenin herkes için iyi olduğunu: (120, 120) ikisine de 6 verirdi. Mahkûmun İkilemi de aynı yapıdadır.

Mahkûmun İkilemi (Prisoner's Dilemma)

- İki şüpheli (Sam ve John) ayrı odalarda sorgulanıyor; birbirleriyle konuşamıyorlar.
- Her birinin iki seçeneği var: **sessiz kalmak** ya da suç ortağını ele verip **itiraf etmek**.

Durum	Sam	John
İkisi de sessiz	1 yıl	1 yıl
Sam itiraf eder, John sessiz	serbest	3 yıl
Sam sessiz, John itiraf eder	3 yıl	serbest
İkisi de itiraf eder	2 yıl	2 yıl

Terim birliği: **İşbirliği** = suç ortağıyla işbirliği = **sessiz kalmak**. **İhanet** = itiraf etmek (polisle işbirliği). Kısaca: "Sessiz" ve "İhanet".

Mahkûmun İkilemi oyunu

MAHKÛMUN İKİLEMİ

Sam ve John'un Kazanç Matrisi

- Kazanç = −(hapis yılı): az ceza, büyük kazanç.
- Her hücrede ilk sayı **Sam'in** (satır), ikinci sayı **John'un** (sütun) kazancı.
- İkisi de sessiz: ikisi de −1.
- Sam sessiz, John ihanet: Sam −3, John 0.

	John: Sessiz	**John: İhanet**
Sam: Sessiz	a (−1, −1)	b (−3, 0)
Sam: İhanet	c (0, −3)	d (−2, −2)

Hücreler a, b, c, d olarak adlandırıldı.

Soru: Her mahkûm kendi cezasını en aza indirmek isterse ne olur?

MAHKÛMUN İKİLEMİ

İhanet Neden Baskın? Durum Analizi

Durum 1: John sessiz kalırsa

Sam sessiz: −1 (hücre a) · Sam ihanet: 0 (hücre c) ⇒ **ihanet daha iyi** (0 > −1).

MAHKÛMUN İKİLEMİ

İhanet Neden Baskın? Durum Analizi

Durum 1: John sessiz kalırsa

Sam sessiz: −1 (hücre a) · Sam ihanet: 0 (hücre c) ⇒ **ihanet daha iyi** (0 > −1).

Durum 2: John ihanet ederse

Sam sessiz: −3 (hücre b) · Sam ihanet: −2 (hücre d) ⇒ **ihanet daha iyi** (−2 > −3).

İhanet Neden Baskın? Durum Analizi

Durum 1: John sessiz kalırsa

Sam sessiz: -1 (hücre a) · Sam ihanet: 0 (hücre c) ⇒ **ihanet daha iyi** ($0 > -1$).

Durum 2: John ihanet ederse

Sam sessiz: -3 (hücre b) · Sam ihanet: -2 (hücre d) ⇒ **ihanet daha iyi** ($-2 > -3$).

John **ne yaparsa yapsın** ihanet daha az ceza getiriyor: ihanet **baskın strateji**. John'un olasılıklarını bilmek gerekmez; “ortalama cezayı azaltır” gerekçesi yanlıştır.

MAHKÛMUN İKİLEMİ

İhanet Neden Baskın? Durum Analizi

Durum 1: John sessiz kalırsa

Sam sessiz: −1 (hücre a) · Sam ihanet: 0 (hücre c) ⇒ **ihanet daha iyi** (0 > −1).

Durum 2: John ihanet ederse

Sam sessiz: −3 (hücre b) · Sam ihanet: −2 (hücre d) ⇒ **ihanet daha iyi** (−2 > −3).

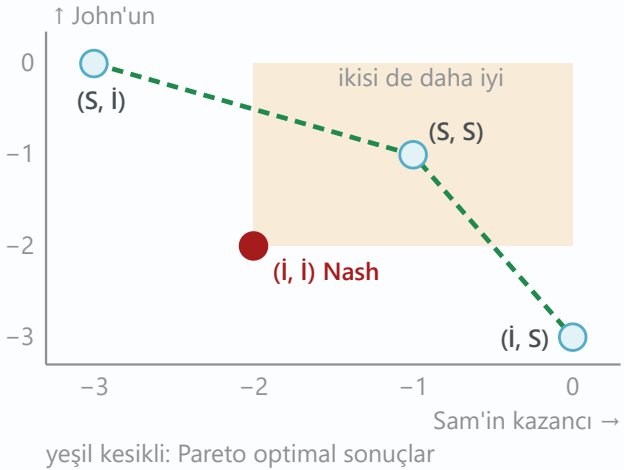
John **ne yaparsa yapsın** ihanet daha az ceza getiriyor: ihanet **baskın strateji**. John'un olasılıklarını bilmek gerekmez; "ortalama cezayı azaltır" gerekçesi yanlıştır.

John da aynı ikilemi yaşar; onun için de ihanet baskın. Sonuç hücre **d**: **(−2, −2)**, yani 2'şer yıl. Oysa ikisi de sessiz kalsaydı (hücre a) **(−1, −1)** alacaklardı: itiraf "ikisi için en iyi sonuç" **değildir**; bireysel en iyi seçim ikisini birden kötüleştirir.

İkilemin Mesajı: Bireysel Rasyonellik, Ortak Kötü Sonuç

- Oyunun tek Nash dengesi (İhanet, İhanet) = (−2, −2).
- (Sessiz, Sessiz) = (−1, −1) ikisi için de daha iyi: denge **Pareto optimal değil**.
- Anlaşma yapsalar bile bağlayıcı değil (işbirlikçi olmayan oyun): her biri sözünü bozmaya teşviklidir.

Mesaj: Herkesin kendi çıkarına rasyonel davranması, toplum için iyi sonucu garanti etmez.



yeşil kesikli: Pareto optimal sonuçlar

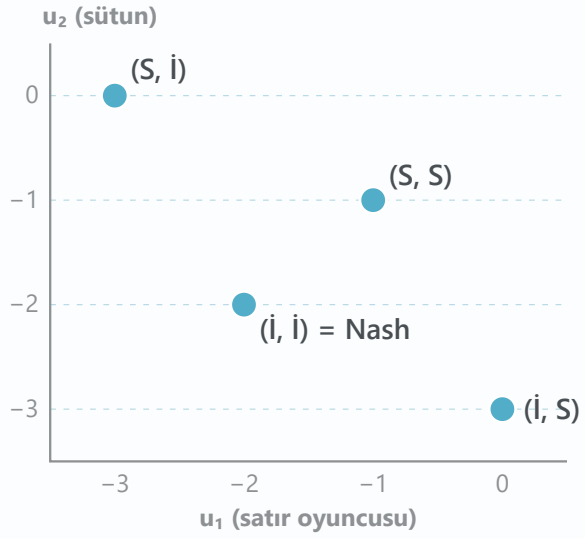
Nerede görülür?

Fiyat savaşları, silahlanma yarışı, iklim anlaşmaları, sporda doping, ortak kaynakların aşırı kullanımı.

Çıkış yolları

Tekrarlanan oyun ve itibar (Tit-for-Tat), sözleşme ve teşvik tasarımı.

Pareto Optimallik: Kazanç Uzayında Bakış



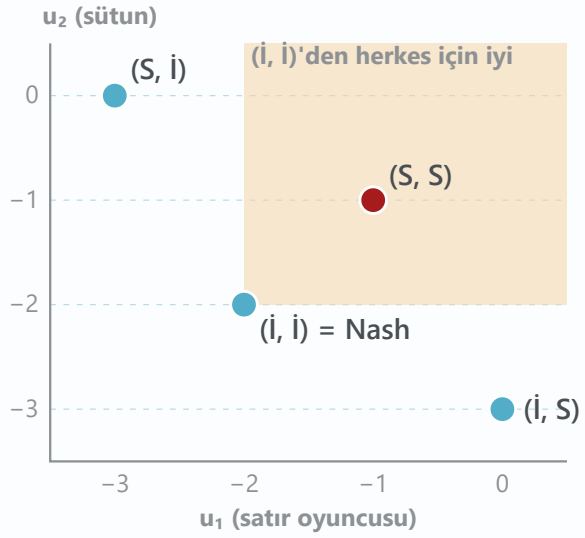
Her sonuç bir nokta: (u_1, u_2). Mahkûmun İkilemi'nin dört sonucu (S: Sessiz, İ: İhanet), kazanç = −(hapis yılı).

Adım 1 / 3

- **Pareto baskınlık:** x, y'yi baskılar ⇔ herkes için u(x) ≥ u(y) ve en az biri için >.
- **Pareto optimal:** hiçbir sonuç onu baskılamaz; kimseyi kötüleştirmeden kimse iyileştirilemez.
- Grafikte: noktanın sağ üst çeyreğinde başka sonuç yoksa Pareto optimaldir.
- Pareto optimallik **adalet değildir**: (S, İ) de Pareto optimaldir ama biri 3 yıl yatar.

Denge (kararlılık) ile verimlilik ayrı sorulardır; aradaki kayıp Anarşinin Bedeli ile ölçülür.

Pareto Optimallik: Kazanç Uzayında Bakış

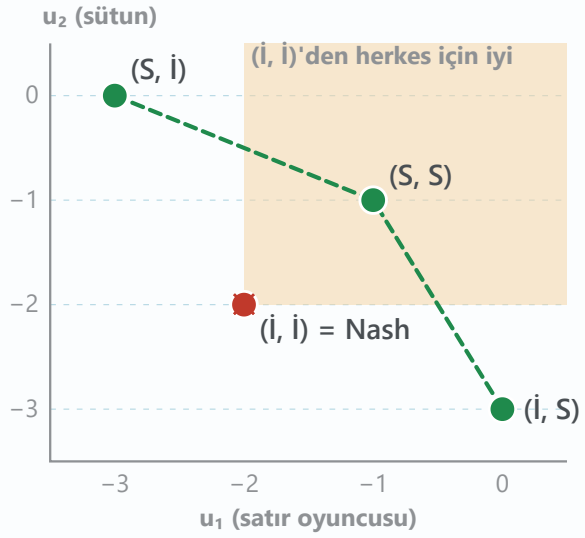


Nash dengesi (İ, İ) = (−2, −2). Turuncu bölgede ikisine de daha iyi bir sonuç var: **(S, S)**.

- **Pareto baskınlık:** x, y'yi baskılar ⇔ herkes için u(x) ≥ u(y) ve en az biri için >.
- **Pareto optimal:** hiçbir sonuç onu baskılamaz; kimseyi kötüleştirmeden kimse iyileştirilemez.
- Grafikte: noktanın sağ üst çeyreğinde başka sonuç yoksa Pareto optimaldir.
- Pareto optimallik **adalet değildir**: (S, İ) de Pareto optimaldir ama biri 3 yıl yatar.

Denge (kararlılık) ile verimlilik ayrı sorulardır; aradaki kayıp Anarşinin Bedeli ile ölçülür.

Pareto Optimallik: Kazanç Uzayında Bakış



Pareto optimal: (S, S), (S, İ), (İ, S) (yeşil). Tek Nash dengesi olan (İ, İ) ise Pareto baskılanır.

- **Pareto baskınlık:** x, y'yi baskılar ⇔ herkes için u(x) ≥ u(y) ve en az biri için >.
- **Pareto optimal:** hiçbir sonuç onu baskılamaz; kimseyi kötüleştirmeden kimse iyileştirilemez.
- Grafikte: noktanın sağ üst çeyreğinde başka sonuç yoksa Pareto optimaldir.
- Pareto optimallik **adalet değildir**: (S, İ) de Pareto optimaldir ama biri 3 yıl yatar.

Denge (kararlılık) ile verimlilik ayrı sorulardır; aradaki kayıp Anarşinin Bedeli ile ölçülür.

KLASİK 2×2 OYUNLAR

Altı Klasik 2×2 Oyun ve Saf Dengeleri

Oyun	Eylemler (1 / 2)	(1, 1)	(1, 2)	(2, 1)	(2, 2)	Saf Nash dengeleri
Yazı-Tura Eşleştirme	Yazı / Tura	(1, −1)	(−1, 1)	(−1, 1)	(1, −1)	yok
Tavuk	Kır / Düz git	(0, 0)	(−1, 1)	(1, −1)	(−10, −10)	(Kır, Düz git), (Düz git, Kır)
Cinsiyetler Savaşı	Opera / Futbol	(3, 2)	(0, 0)	(0, 0)	(2, 3)	(Opera, Opera), (Futbol, Futbol)
Geyik Avı	Geyik / Tavşan	(4, 4)	(0, 3)	(3, 0)	(3, 3)	(Geyik, Geyik), (Tavşan, Tavşan)
Saf Koordinasyon	Parti / Ev	(10, 10)	(0, 0)	(0, 0)	(5, 5)	(Parti, Parti), (Ev, Ev)
Bedava Para	Yazı / Tura	(1, 1)	(0, 0)	(0, 0)	(0, 0)	(Yazı, Yazı) kesin; (Tura, Tura) zayıf

(1, 2): satır oyuncusu 1. eylemi, sütun oyuncusu 2. eylemi oynar; hücrede (satır oyuncusunun, sütun oyuncusunun) kazançları.

Saf dengeyi bulmak: her sütunda satır oyuncusunun, her satırda sütun oyuncusunun **en iyi tepkisini** işaretle; iki işaretli hücre saf Nash dengesidir.

KLASİK 2×2 OYUNLAR

Altı Oyunun Yorumu: Çatışma mı, Koordinasyon mu?

Yazı-Tura Eşleştirme

Çıkarlar tamamen zıt; her hücrede kaybeden sapar. Kestirilebilen her eğilim sömürülür: tek çıkış rastgele oynamaktır (karışık strateji).

Tavuk

Anti-koordinasyon: farklı eylem istenir. İki denge birer sürücüyü kayırır; geri adım atamayacağını inandırıcı biçimde gösteren (taahhüt) kazanır.

Cinsiyetler Savaşı

Buluşmak ortak çıkar, nerede buluşulacağı çatışmadır. Baskın strateji yok; çözüm iletişim, gelenek ya da sırayla seçmektir.

Geyik Avı

Güven ikilemi: (Geyik, Geyik) Pareto optimal ve dengedir, ama Tavşan her durumda 3 garanti eder. Mahkûmun İkilemi'nden farkı: işbirliği dengedir.

Saf Koordinasyon

Çıkar çatışması yok; kötü denge (Ev, Ev) de kendini sürdürür. Oyuncular belirgin olana, odak noktasına (Schelling) yönelir.

Bedava Para

(Yazı, Yazı) kesin denge: sapan kaybeder. (Tura, Tura) zayıf denge: sapmak kazandırmaz ama kaybettirmez de; Yazı zayıf baskındır.

Seviye-k Düşünme: 45 → 30 → 20 …

Düşünme seviyesi	Varsayım	Tahmin
Seviye 0	Herkes rastgele yazar, ortalama 45	—
Seviye 1	Diğerleri seviye 0	2/3·45 = 30
Seviye 2	Diğerleri seviye 1	2/3·30 = 20
Seviye 3	Diğerleri seviye 2	≈ 13,3
Seviye ∞	Herkes tam rasyonel	**0**

- “Ortalama 45 olur, 2/3'ü 30” akıl yürütmesi yalnızca **seviye 1**'dir.
- Başkalarının da 30 diyeceğini düşünen daha düşük (ör. 20) yazar; herkesin fazla düşük yazacağını düşünen biraz yukarı kayabilir.
- En iyi tahmin, diğerlerinin kaçıncı seviyede düşündüğüne dair inancınıza bağlıdır.
- Seviye, bir oyuncunun kaç adım düşündüğüdür; ortak bilginin kaç kat “herkes biliyor” olduğunu sayan bilgi düzeyiyle karıştırılmamalı.

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Seviye 0 rastgele yazar (ortalama 45). Seviye k, herkesi seviye k−1 sanıp onun 2/3'ünü yazar.

Adım 1 / 10

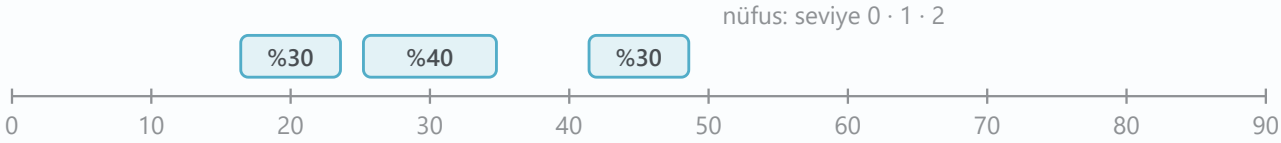
```
def tahmin(k, L0=45):     # seviye-k
    g = L0
    for _ in range(k):
        g = g * 2 / 3
    return g
pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
ort = sum(p * tahmin(k) for k, p in pay.items())
hedef = ort * 2 / 3
for k in range(5):
    print(k, round(tahmin(k), 1),
          round(abs(tahmin(k) - hedef), 1))
```

DEĞİŞKENLER

`L0 = 45`

ÇIKTI

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



İnanç: sınıfın %30'u seviye 0, %40'ı seviye 1, %30'u seviye 2 düşünüyor.

Adım 2 / 10

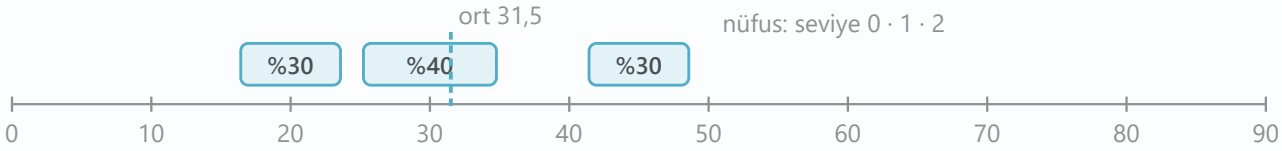
```
1  def tahmin(k, L0=45):     # seviye-k
2      g = L0
3      for _ in range(k):
4          g = g * 2 / 3
5      return g
▶ pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7  ort = sum(p * tahmin(k) for k, p in pay.items())
8  hedef = ort * 2 / 3
9  for k in range(5):
10     print(k, round(tahmin(k), 1),
11           round(abs(tahmin(k) - hedef), 1))
```

DEĞİŞKENLER

`pay = {0: 0.3, 1: 0.4, 2: 0.3}`

ÇIKTI

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Beklenen ortalama: 0,3·45 + 0,4·30 + 0,3·20 = **31,5**.

Adım 3 / 10

```
1  def tahmin(k, L0=45):      # seviye-k
2      g = L0
3      for _ in range(k):
4          g = g * 2 / 3
5      return g
6  pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
▶  ort = sum(p * tahmin(k) for k, p in pay.items())
8  hedef = ort * 2 / 3
9  for k in range(5):
10     print(k, round(tahmin(k), 1),
11           round(abs(tahmin(k) - hedef), 1))
```

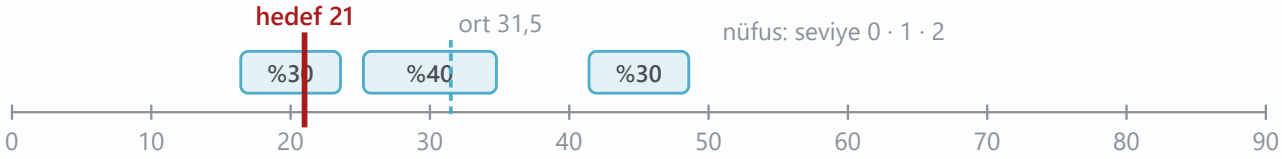
DEĞİŞKENLER

`pay = {0: 0.3, 1: 0.4, 2: 0.3}`

`ort = 31.5`

ÇIKTI

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Kazanan sayı ortalamanın 2/3'ü: **hedef = 21**.

Adım 4 / 10

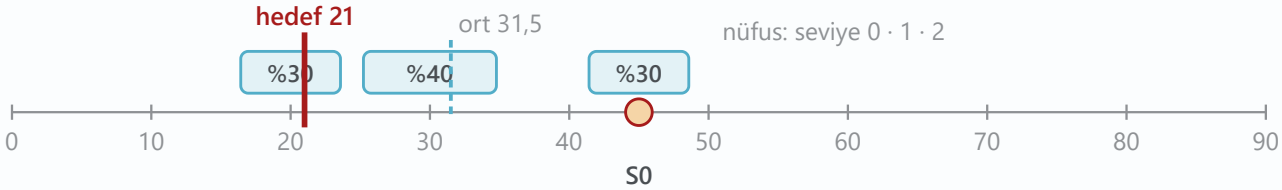
```
1  def tahmin(k, L0=45):     # seviye-k
2      g = L0
3      for _ in range(k):
4          g = g * 2 / 3
5      return g
6  pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7  ort = sum(p * tahmin(k) for k, p in pay.items())
▶  hedef = ort * 2 / 3
9  for k in range(5):
10     print(k, round(tahmin(k), 1),
11           round(abs(tahmin(k) - hedef), 1))
```

DEĞİŞKENLER

`ort = 31.5` `hedef = 21.0`

ÇIKTI

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Seviye 0: tahmin 45, hedefe uzaklık 24. Ortalamayı yazmak çok yüksek.

Adım 5 / 10

```
1 def tahmin(k, L0=45):     # seviye-k
2     g = L0
3     for _ in range(k):
4         g = g * 2 / 3
5     return g
6 pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7 ort = sum(p * tahmin(k) for k, p in pay.items())
8 hedef = ort * 2 / 3
▶ for k in range(5):
▶     print(k, round(tahmin(k), 1),
▶           round(abs(tahmin(k) - hedef), 1))
```

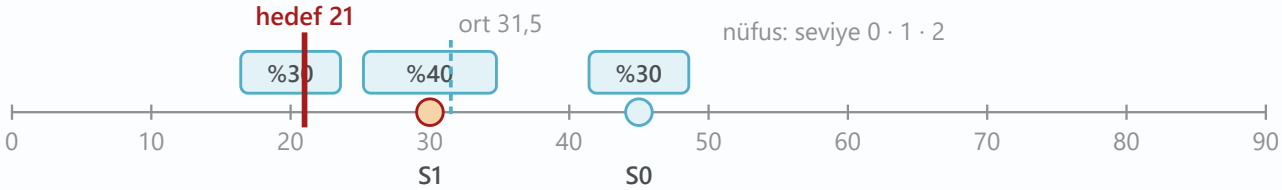
DEĞİŞKENLER

`k = 0` `tahmin = 45.0` `uzaklik = 24.0`

ÇIKTI

```
0 45 24.0
```

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Seviye 1: tahmin 30, hedefe uzaklık 9. Klasik “ortalama 45 → 30” cevabı: hâlâ uzak.

Adım 6 / 10

```
1 def tahmin(k, L0=45):     # seviye-k
2     g = L0
3     for _ in range(k):
4         g = g * 2 / 3
5     return g
6 pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7 ort = sum(p * tahmin(k) for k, p in pay.items())
8 hedef = ort * 2 / 3
▶ for k in range(5):
▶     print(k, round(tahmin(k), 1),
▶           round(abs(tahmin(k) - hedef), 1))
```

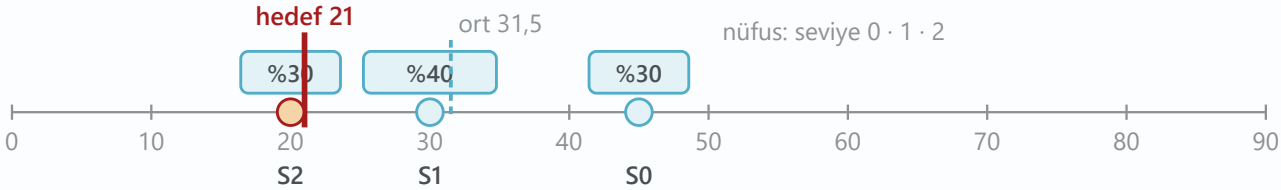
DEĞİŞKENLER

`k = 1` `tahmin = 30.0` `uzaklik = 9.0`

ÇIKTI

```
0 45 24.0
1 30.0 9.0
```

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Seviye 2: tahmin 20, hedefe uzaklık 1. Hedefe en yakın: bu inançla **en iyi tepki**.

Adım 7 / 10

```
1 def tahmin(k, L0=45):     # seviye-k
2     g = L0
3     for _ in range(k):
4         g = g * 2 / 3
5     return g
6 pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7 ort = sum(p * tahmin(k) for k, p in pay.items())
8 hedef = ort * 2 / 3
▶ for k in range(5):
▶     print(k, round(tahmin(k), 1),
▶           round(abs(tahmin(k) - hedef), 1))
```

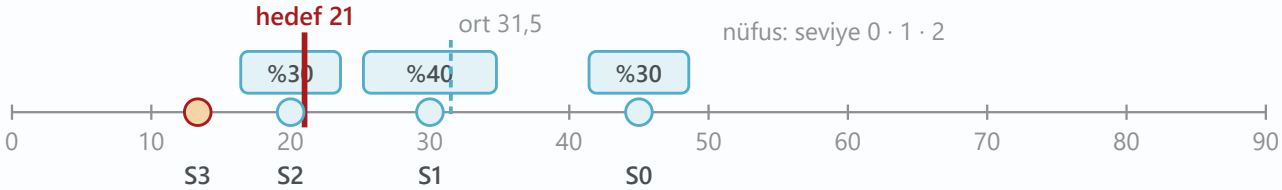
DEĞİŞKENLER

`k = 2` `tahmin = 20.0` `uzaklik = 1.0`

ÇIKTI

```
0 45 24.0
1 30.0 9.0
2 20.0 1.0
```

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Seviye 3: tahmin 13,3, hedefe uzaklık 7,7. Bir seviye fazla düşünmek de kaybettirir.

Adım 8 / 10

```
1 def tahmin(k, L0=45):     # seviye-k
2     g = L0
3     for _ in range(k):
4         g = g * 2 / 3
5     return g
6 pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7 ort = sum(p * tahmin(k) for k, p in pay.items())
8 hedef = ort * 2 / 3
▶ for k in range(5):
▶     print(k, round(tahmin(k), 1),
▶           round(abs(tahmin(k) - hedef), 1))
```

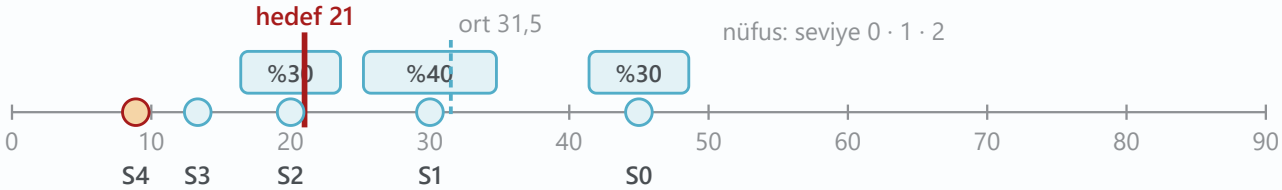
DEĞİŞKENLER

`k = 3` `tahmin = 13.33` `uzaklik = 7.67`

ÇIKTI

```
0 45 24.0
1 30.0 9.0
2 20.0 1.0
3 13.3 7.7
```

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Seviye 4: tahmin 8,9, hedefe uzaklık 12,1. Tam rasyonel cevaba (0) yaklaştıkça uzaklık artıyor.

Adım 9 / 10

```
1 def tahmin(k, L0=45):     # seviye-k
2     g = L0
3     for _ in range(k):
4         g = g * 2 / 3
5     return g
6 pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7 ort = sum(p * tahmin(k) for k, p in pay.items())
8 hedef = ort * 2 / 3
▶ for k in range(5):
▶     print(k, round(tahmin(k), 1),
▶           round(abs(tahmin(k) - hedef), 1))
```

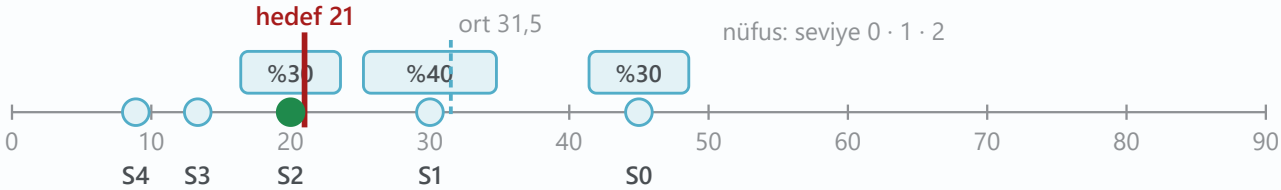
DEĞİŞKENLER

`k = 4` `tahmin = 8.89` `uzaklik = 12.11`

ÇIKTI

```
0 45 24.0
1 30.0 9.0
2 20.0 1.0
3 13.3 7.7
4 8.9 12.1
```

Kod İzleme: Seviye-k Akıl Yürütme ve En İyi Tahmin



Sonuç: bu nüfusa karşı 20 yazan kazanır; Nash dengesi 0 ise kaybeder. En iyi tepki **inanca** bağlıdır.

Adım 10 / 10

```
1  def tahmin(k, L0=45):      # seviye-k
2      g = L0
3      for _ in range(k):
4          g = g * 2 / 3
5      return g
6  pay = {0: 0.3, 1: 0.4, 2: 0.3}   # inanç
7  ort = sum(p * tahmin(k) for k, p in pay.items())
8  hedef = ort * 2 / 3
▶ for k in range(5):
10     print(k, round(tahmin(k), 1),
11           round(abs(tahmin(k) - hedef), 1))
```

DEĞİŞKENLER

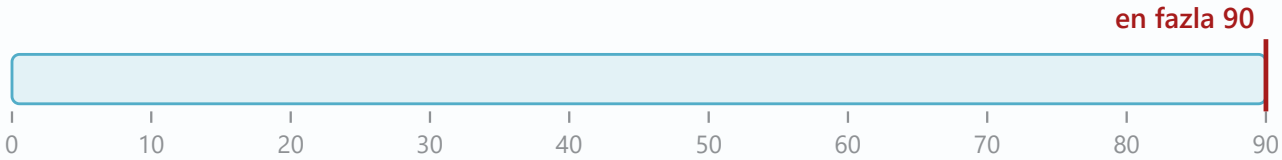
`en_iyi = seviye 2` `tahmin = 20.0`

ÇIKTI

```
0 45 24.0
1 30.0 9.0
2 20.0 1.0
3 13.3 7.7
4 8.9 12.1
```

2/3 ORTALAMASI OYUNU

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Başlangıç: her tahmin 0 ile 90 arasında. Hiçbir şey elenmedi.

Adım 1 / 8

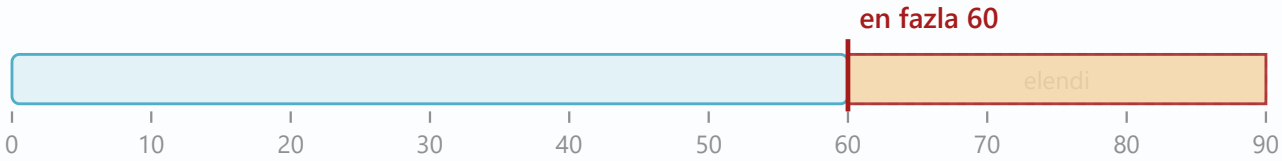
```
ust = 90                  # olası en büyük tahmin
for tur in range(1, 7):
    # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
    ust = ust * 2 / 3
    print(tur, round(ust, 1))
# ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

DEĞİŞKENLER

`ust = 90`

ÇIKTI

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Ortalama en fazla 90 olabilir, hedef (2/3 × ortalama) en fazla 60. 60'ın üstündeki her tahmin, hiçbir durumda 60 yazmaktan iyi değildir → elenir.

Adım 2 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
▶     ust = ust * 2 / 3
▶     print(tur, round(ust, 1))
6 # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

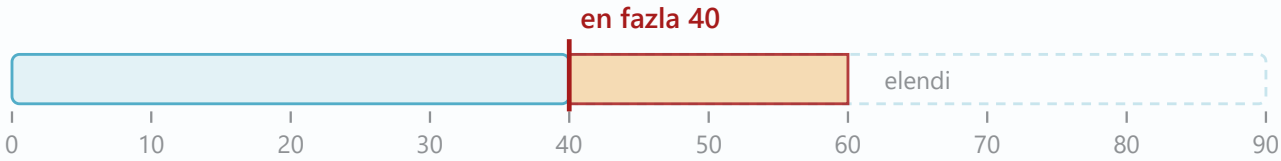
DEĞİŞKENLER

`tur = 1` `ust = 60.0`

ÇIKTI

```
1 60.0
```

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Herkes rasyonel ve bunu herkes biliyor: kimse 60'ı aşmaz. Ortalama ≤ 60 ⇒ hedef ≤ 40. 40'ın üstü elenir.

Adım 3 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
      ust = ust * 2 / 3
      print(tur, round(ust, 1))
6 # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

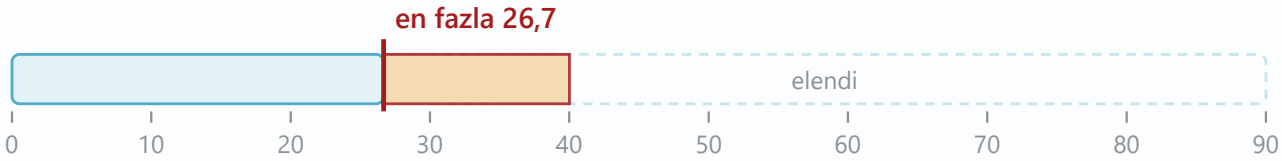
DEĞİŞKENLER

`tur = 2` `ust = 40.0`

ÇIKTI

```
1 60.0
2 40.0
```

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Bir tur daha: ortalama ≤ 40 ⇒ hedef ≤ 26,7. “Herkesin, herkesin rasyonel olduğunu bildiğini” bilmek gerekir.

Adım 4 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
▶     ust = ust * 2 / 3
▶     print(tur, round(ust, 1))
6 # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

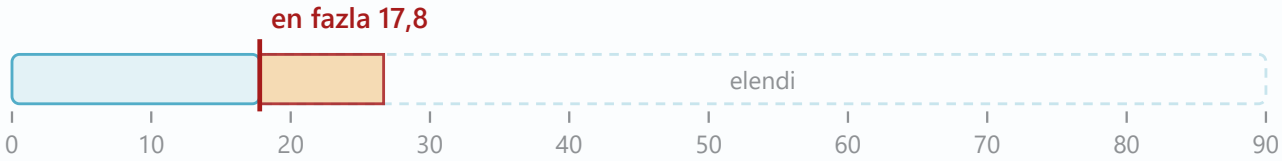
DEĞİŞKENLER

`tur = 3` `ust = 26.67`

ÇIKTI

```
1 60.0
2 40.0
3 26.7
```

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Ortalama ≤ 26,7 ⇒ hedef ≤ 17,8. Her turda üst sınır 2/3 ile çarpılıyor.

Adım 5 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
▶     ust = ust * 2 / 3
▶     print(tur, round(ust, 1))
6 # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

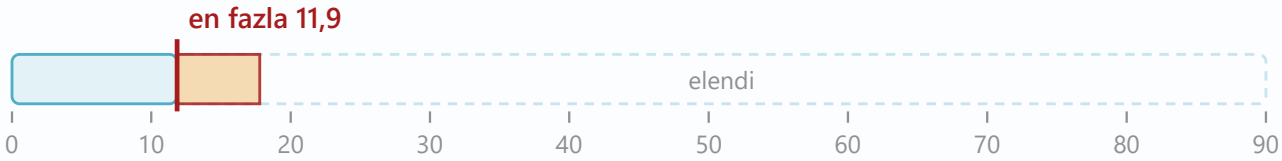
DEĞİŞKENLER

`tur = 4` `ust = 17.78`

ÇIKTI

```
1 60.0
2 40.0
3 26.7
4 17.8
```

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Ortalama ≤ 17,8 ⇒ hedef ≤ 11,9. Aralık geometrik olarak küçülüyor.

Adım 6 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
▶     ust = ust * 2 / 3
▶     print(tur, round(ust, 1))
6 # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

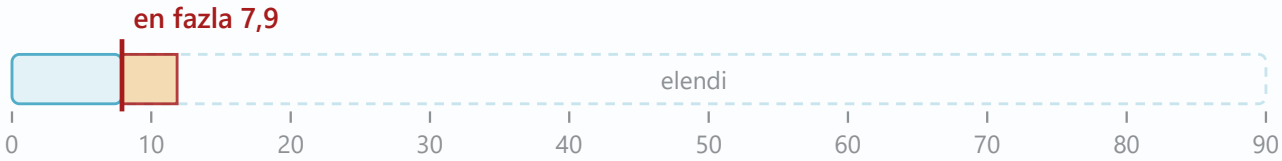
DEĞİŞKENLER

`tur = 5` `ust = 11.85`

ÇIKTI

```
1 60.0
2 40.0
3 26.7
4 17.8
5 11.9
```

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Ortalama ≤ 11,9 ⇒ hedef ≤ 7,9. k tur sonra üst sınır $90 \cdot (2/3)^k$.

Adım 7 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
      ust = ust * 2 / 3
      print(tur, round(ust, 1))
6 # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

DEĞİŞKENLER

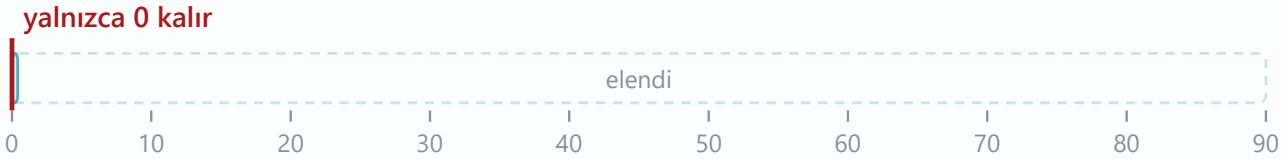
`tur = 6` `ust = 7.9`

ÇIKTI

```
1 60.0
2 40.0
3 26.7
4 17.8
5 11.9
6 7.9
```

2/3 ORTALAMASI OYUNU

Yinelemeli Eleme: 90 → 60 → 40 → … → 0



Süreç sonsuza dek sürer ve üst sınır 0'a yakınsar: yinelemeli elemeden sağ çıkan tek profil **herkesin 0 söylemesi**dir.

Adım 8 / 8

```
1 ust = 90                # olası en büyük tahmin
2 for tur in range(1, 7):
3     # ortalama ≤ ust  ⇒  hedef ≤ (2/3)·ust
4     ust = ust * 2 / 3
5     print(tur, round(ust, 1))
▶ # ust = 90·(2/3)^k → 0 : tek denge herkes 0
```

DEĞİŞKENLER

`tur = …` `ust = 0 (limit)`

ÇIKTI

```
1 60.0
2 40.0
3 26.7
4 17.8
5 11.9
6 7.9
```

2/3 ORTALAMASI OYUNU

Tek Nash Dengesi: Herkes 0 Söyler

Denge olduğunu doğrula

- Herkes 0 derse ortalama 0, hedef 0: herkes kazanır (ödül paylaşılır).
- Biri $x > 0$'a sapsa ortalama x/n, hedef $2x/(3n)$ olur; diğerlerinin 0'ı hedefe daha yakın kalır → sapan kaybeder.
- Başka hiçbir profil yinelemeli elemeden sağ çıkmaz: denge **tektir**. (Yalnız tam sayılara izin verilseydi $n \geq 4$ oyuncuda "herkes 1" de denge olurdu; ondalıklı tahmin bu ikinci dengeyi ortadan kaldırır.)

Peki gerçekte?

- Deneylerde kazanan sayı 0 çıkmaz: 1997'de Financial Times okurlarıyla yapılan 0–100 sürümünde kazanan tahmin **13** idi.
- Çünkü herkesin tam rasyonel olduğu ve bunun **ortak bilgi** olduğu varsayımı gerçekte tam tutmaz.

Dikkat: Bu oyunda Nash dengesini belirlemek zor değildir; denge tek ve saf stratejilidir (herkes 0). Zor olan, *gerçek* oyuncular karşısında en iyi tepkiyi tahmin etmektir. Oyunu tekrarlayınca tahminler dengeye doğru iner.

Oyuna Dönüş: 2/3 Ortalaması Oyunu

Kuramın cevabı: 0

Adım	Anlamlı en büyük tahmin
başlangıç	90
1	2/3 × 90 = 60
2	2/3 × 60 = 40
3	26,7
4	17,8
…	→ 0

Her adım bir kat daha “herkes rasyonel” varsayımı ekler; sonsuz adımın sonu tek Nash dengesi: herkes 0.

Gerçekte: birkaç adım

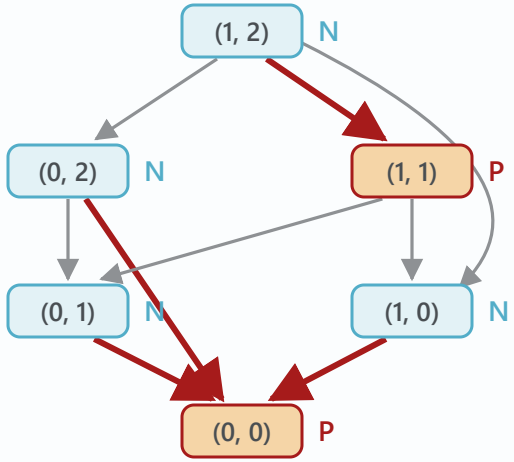
Seviye	Varsayım	Tahmin
0	rastgele, ortalama 45	45
1	diğerleri seviye-0	30
2	diğerleri seviye-1	20
3	diğerleri seviye-2	13,3

Nagel'in (1995) deneylerinde ilk tur tahminleri 1–2 düşünme adımının çevresinde yığıldı; oyun tekrarlandıkça tahminler 0'a doğru indi.

Kazanmak için 0'ı değil, sınıfın kaç adım düşündüğünü bilmek gerekir: sınıf seviye-1 oynar ve ortalama 30 çıkarsa hedef 20 olur, yani seviye-2 kazanır.

Kombinatoryal Oyunlar ve Zermelo Teoremi

- İki oyuncu **sırayla** oynar; mükemmel bilgi, şans yok, oyun sonlu adımda biter.
- **Normal oyun kuralı**: son hamleyi yapan kazanır (Nim; 21'i Söyleme'de ise 21'i söyleyen kaybeder). XOX ve satrançta beraberlik de var.
- **Zermelo (1913)**: Böyle bir oyunda ya bir taraf kazanmayı ya da iki taraf beraberliği garanti eder.
- **Strateji çalma** (Hex): beraberlik yoktur; ikinci oyuncunun kazanan stratejisi olsaydı ilk oyuncu bir taş fazlasıyla onu oynayabilirdi, bu yüzden ilk oyuncu kazanır.
- Her konum **N** (sıradaki kazanır) ya da **P** (önceki kazanır): P'den her hamle N'ye, N'den en az bir hamle P'ye gider.

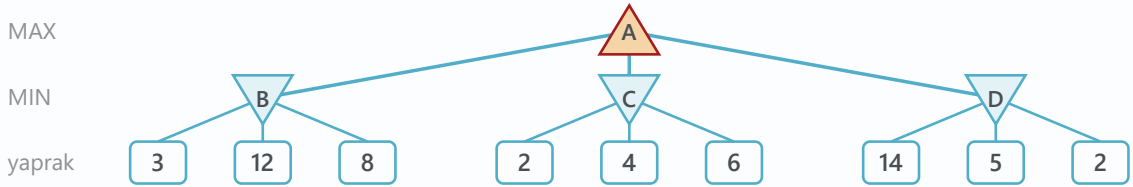


P (kırmızı): sıradaki kaybeder · kalın ok: kazandıran

İki yığınlı Nim (1 ve 2 jeton): sırası gelen tek bir yığından en az bir jeton alır, son jetonu alan kazanır.

Etiketler yapraktan köke hesaplanır: (0, 0)'da hamle yok → P. Bu, minimaksın özel hâlidir.

Kod İzleme: Oyun Ağacında Minimaks



Kök A'da sıra MAX'ta (büyük değer ister); B, C, D'de MIN (küçük ister). Yapraklar oyun sonu kazançları.

Adım 1 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
8      v = minimax(c, not maxi)
9      best = max(best, v) if maxi else min(best, v)
10   return best
▶ print(minimax("A", True))
```

DEĞİŞKENLER

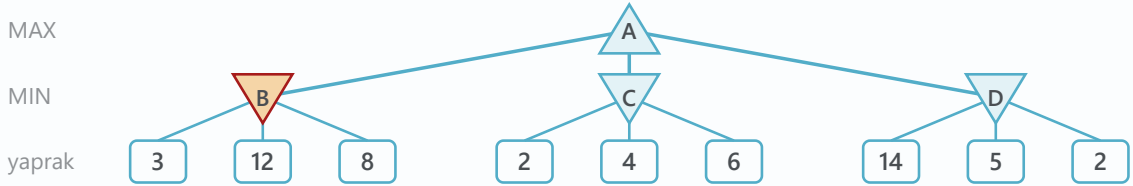
`n = "A"` `maxi = True`

ÇAĞRI YIĞINI

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



B bir MIN düğümü: best = 99 ile başla, çocukları sırayla değerlendir.


```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
▶    best = -99 if maxi else 99
▶    for c in T[n]:
8      v = minimax(c, not maxi)
9      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "B"` `maxi = False` `best = 99`

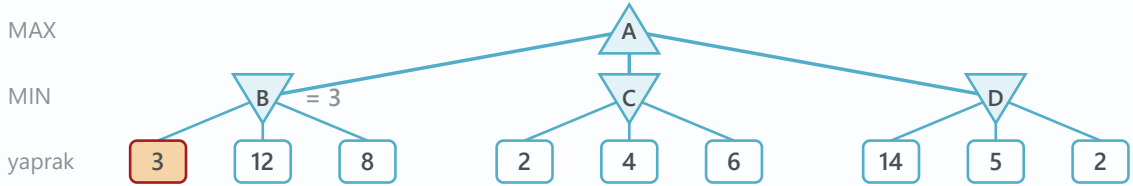
ÇAĞRI YIĞINI

`minimax("B", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 3 (int → hemen döner). best = min(99, 3) = **3**.

Adım 3 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "B"` `c = 3` `v = 3` `best = 3`

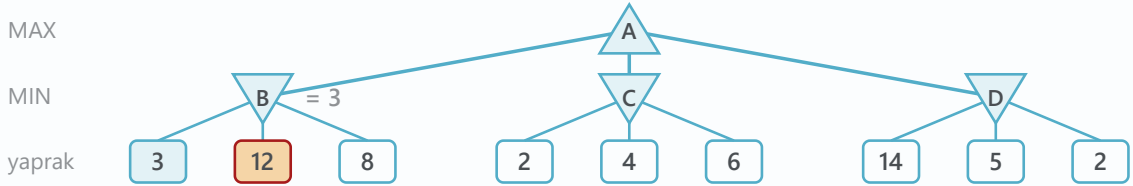
ÇAĞRI YIĞINI

`minimax("B", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 12: min(3, 12) = **3**.


```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "B"` `c = 12` `v = 12` `best = 3`

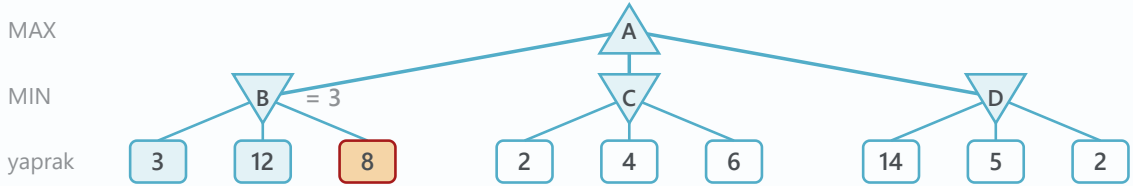
ÇAĞRI YIĞINI

`minimax("B", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 8: min(3, 8) = **3**.

Adım 5 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "B"` `c = 8` `v = 8` `best = 3`

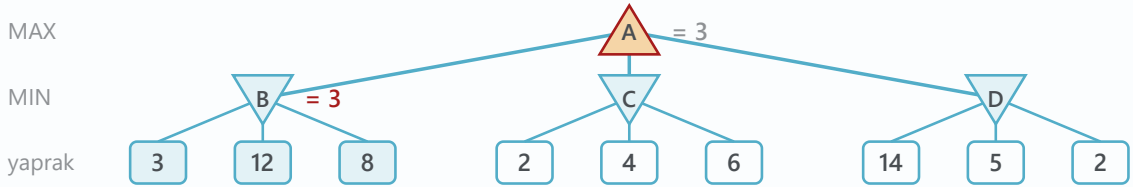
ÇAĞRI YIĞINI

`minimax("B", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



B değeri 3 olarak döner. A'da: max(−99, 3) = **3**.

Adım 6 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
8      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
▶    return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

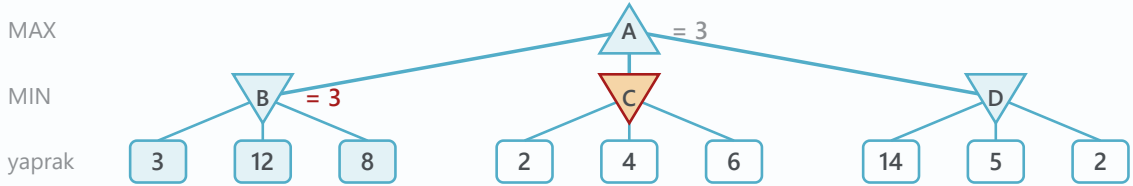
`n = "A"` `maxi = True` `c = "B"` `v = 3` `best = 3`

ÇAĞRI YIĞINI

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



A'nın sıradaki çocuğu C (MIN). Yeniden best = 99.


```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
▶    best = -99 if maxi else 99
▶    for c in T[n]:
8      v = minimax(c, not maxi)
9      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

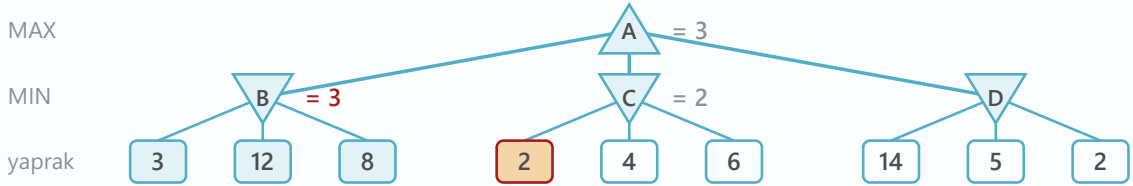
`n = "C"` `maxi = False` `best = 99`

ÇAĞRI YIĞINI

```
minimax("C", False)
minimax("A", True)
```

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 2 (int → hemen döner). best = min(99, 2) = **2**.

Adım 8 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "C"` `c = 2` `v = 2` `best = 2`

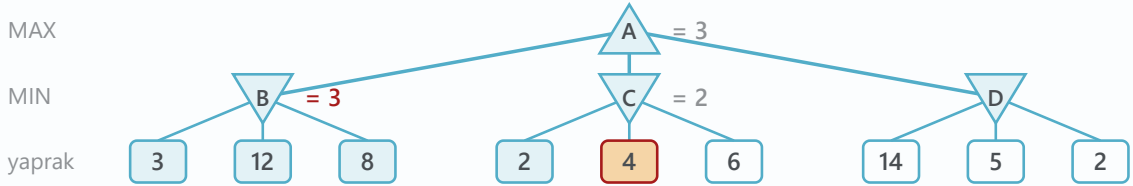
ÇAĞRI YIĞINI

`minimax("C", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 4: min(2, 4) = **2**.


```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "C"` `c = 4` `v = 4` `best = 2`

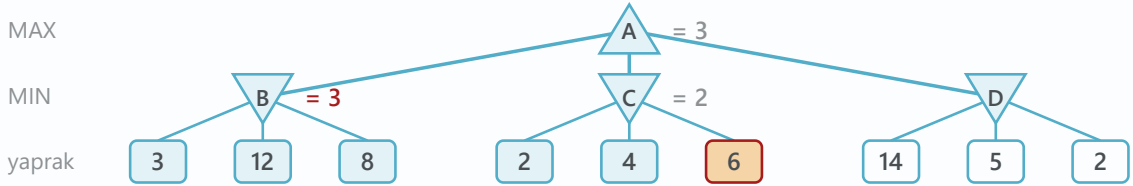
ÇAĞRI YIĞINI

`minimax("C", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 6: min(2, 6) = **2**.


```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "C"` `c = 6` `v = 6` `best = 2`

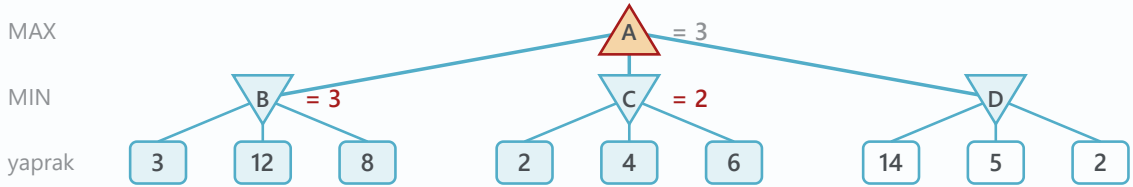
ÇAĞRI YIĞINI

`minimax("C", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



C değeri 2 olarak döner. A'da: max(3, 2) = **3**. C, A için B'den kötü.

Adım 11 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
8      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
▶    return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

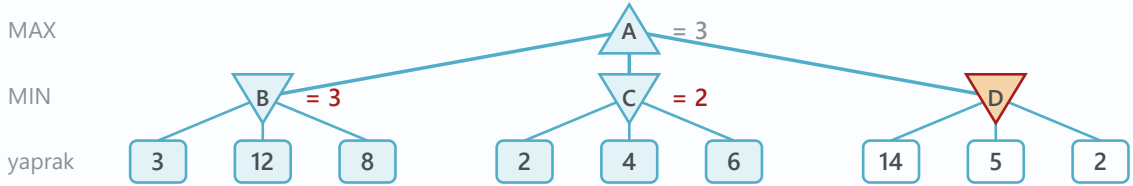
`n = "A"` `maxi = True` `c = "C"` `v = 2` `best = 3`

ÇAĞRI YIĞINI

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



A'nın sıradaki çocuğu D (MIN). Yeniden best = 99.

Adım 12 / 17

```
1 T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2      "C": [2, 4, 6], "D": [14, 5, 2]}
3 def minimax(n, maxi):
4   if isinstance(n, int):          # yaprak
5     return n
▶   best = -99 if maxi else 99
▶   for c in T[n]:
8     v = minimax(c, not maxi)
9     best = max(best, v) if maxi else min(best, v)
10  return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "D"` `maxi = False` `best = 99`

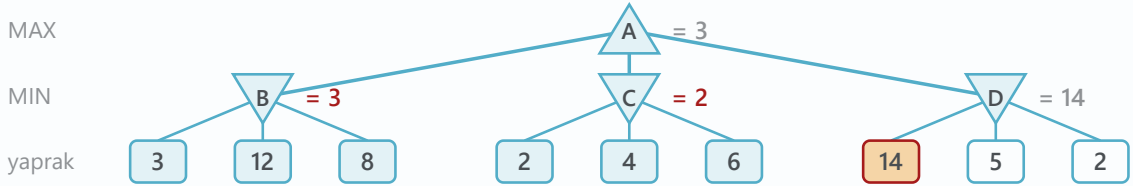
ÇAĞRI YIĞINI

`minimax("D", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 14 (int → hemen döner). best = min(99, 14) = **14**.

Adım 13 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

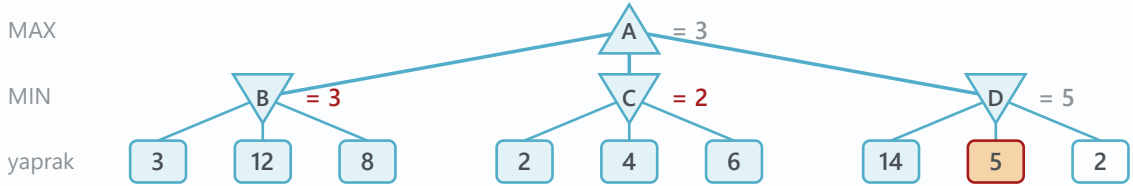
`n = "D"` `c = 14` `v = 14` `best = 14`

ÇAĞRI YIĞINI

```
minimax("D", False)
minimax("A", True)
```

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 5: min(14, 5) = **5**.

Adım 14 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
▶      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
10   return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

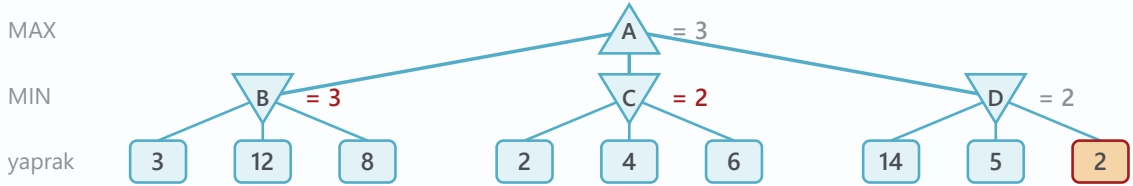
`n = "D"` `c = 5` `v = 5` `best = 5`

ÇAĞRI YIĞINI

```
minimax("D", False)
minimax("A", True)
```

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Yaprak 2: min(5, 2) = **2**.

Adım 15 / 17

```
1 T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2      "C": [2, 4, 6], "D": [14, 5, 2]}
3 def minimax(n, maxi):
4   if isinstance(n, int):          # yaprak
5     return n
6   best = -99 if maxi else 99
7   for c in T[n]:
▶     v = minimax(c, not maxi)
▶     best = max(best, v) if maxi else min(best, v)
10  return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

`n = "D"` `c = 2` `v = 2` `best = 2`

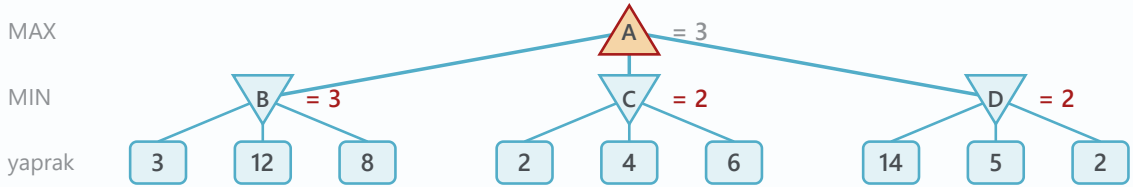
ÇAĞRI YIĞINI

`minimax("D", False)`

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



D değeri 2 olarak döner. A'da: max(3, 2) = **3**. D'deki 14 cazip görünse de MIN 2'yi seçer.

Adım 16 / 17

```
1  T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2       "C": [2, 4, 6], "D": [14, 5, 2]}
3  def minimax(n, maxi):
4    if isinstance(n, int):          # yaprak
5      return n
6    best = -99 if maxi else 99
7    for c in T[n]:
8      v = minimax(c, not maxi)
▶      best = max(best, v) if maxi else min(best, v)
▶    return best
11 print(minimax("A", True))
```

DEĞİŞKENLER

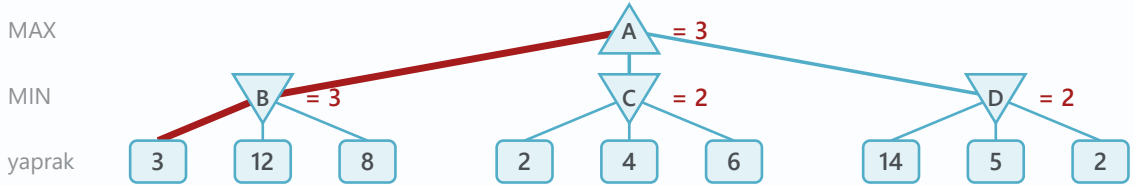
`n = "A"` `maxi = True` `c = "D"` `v = 2` `best = 3`

ÇAĞRI YIĞINI

`minimax("A", True)`

ÇIKTI

Kod İzleme: Oyun Ağacında Minimaks



Oyunun değeri 3: MAX B'yi seçer, MIN ne yaparsa yapsın en az 3 garanti. 9 yaprağın hepsine bakıldı.

Adım 17 / 17

```
1 T = {"A": ["B", "C", "D"], "B": [3, 12, 8],
2      "C": [2, 4, 6], "D": [14, 5, 2]}
3 def minimax(n, maxi):
4   if isinstance(n, int):          # yaprak
5     return n
6   best = -99 if maxi else 99
7   for c in T[n]:
8     v = minimax(c, not maxi)
9     best = max(best, v) if maxi else min(best, v)
▶   return best
▶ print(minimax("A", True))
```

DEĞİŞKENLER

`n = "A"` `best = 3`

ÇAĞRI YIĞINI

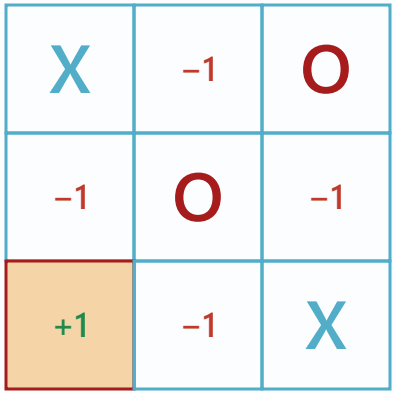
boş

ÇIKTI

```
3
```

XOX: Tüm Ağaç Aranabilir, Sonuç Beraberlik

- XOX'un ağacı küçük: tüm oyunları sayan program **255.168** farklı oyun ve **5.478** erişilebilir konum bulur.
- Boş tahtada minimaks değeri **0**: iki taraf da hatasız oynarsa oyun **berabere**.
- Sağdaki konumda X yalnız sol alt kareyle kazanır: O'nun çaprazını keser ve aynı anda iki tehdit kurar (+1). Diğer her karede O kazanır (−1).
- Satrançta tam arama imkânsız ($\approx 10^{120}$ oyun): derinlik sınırı, değerlendirme fonksiyonu ve alfa-beta budaması.



Boş karelerde: o hamleden sonraki minimaks değeri

sıra X'te

+1: X kazanır

−1: O kazanır

sol alt: O'yu keser ve çift tehdit kurar

Minimaks XOX rakibi bu değerleri hesaplayarak oynar: onu yenemezsiniz, en iyi ihtimalle berabere kalırsınız. Alfa-beta budaması ve derinlik sınırlı arama: Minimaks ve Alfa-Beta Budaması.

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

Kural: iki oyuncu sırayla, 1'den başlayarak 1, 2 ya da 3 ardışık sayı söyler; 21'i söyleyen kaybeder. Taban durum: son söylenen 20 ise sıradaki oyuncu 21'i söylemek zorunda → **kaybeder** (Y).

Adım 1 / 16

```
HEDEF = 21
kazanir = {HEDEF - 1: False}   # c=20: kayıp
for c in range(HEDEF - 2, -1, -1):
    kazanir[c] = any(not kazanir[c + k]
        for k in (1, 2, 3) if c + k < HEDEF)
kontrol = [c for c in kazanir if not kazanir[c]]
print(sorted(kontrol))
```

DEĞİŞKENLER

`HEDEF = 21` `kazanir[20] = False`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 19: 20 söylenerek rakip Y konumuna itilir → c = 19 **K**.

Adım 2 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 19` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 18: 20 söylenerek rakip Y konumuna itilir → c = 18 **K**.

Adım 3 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 18` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 17: 20 söylenerek rakip Y konumuna itilir → c = 17 **K**.

Adım 4 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 17` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 16: 17, 18, 19 hepsi K (rakip kazanır) → c = 16 **Y**. Yeni kontrol sayısı!

Adım 5 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 16` `kazanir[c] = False`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 15: 16 söylenerek rakip Y konumuna itilir → c = 15 **K**.

Adım 6 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 15` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 14: 16 söylenerek rakip Y konumuna itilir → c = 14 **K**.

Adım 7 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 14` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 13: 16 söylenerek rakip Y konumuna itilir → c = 13 **K**.

Adım 8 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 13` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 12: 13, 14, 15 hepsi K (rakip kazanır) → c = 12 **Y**. Yeni kontrol sayısı!

Adım 9 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 12` `kazanir[c] = False`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 11…9: aynı örüntü, hepsi bir sonraki Y'ye (12) ulaşabilir → K.

Adım 10 / 16

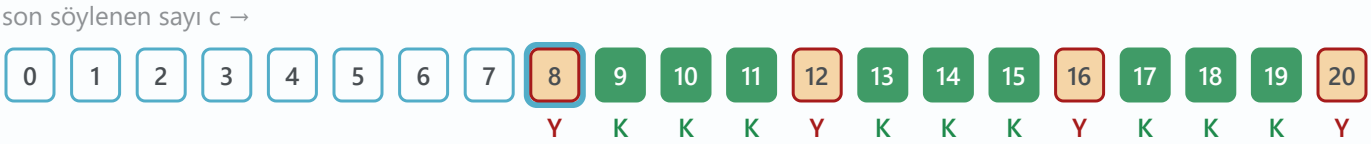
```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 9` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 8: 9, 10, 11 hepsi K (rakip kazanır) → c = 8 **Y**. Yeni kontrol sayısı!

Adım 11 / 16

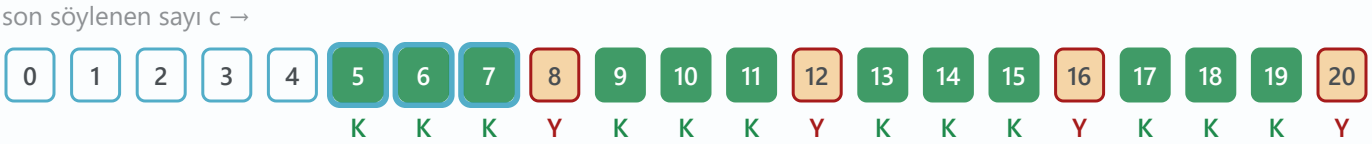
```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 8` `kazanir[c] = False`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 7…5: aynı örüntü, hepsi bir sonraki Y'ye (8) ulaşabilir → K.

Adım 12 / 16

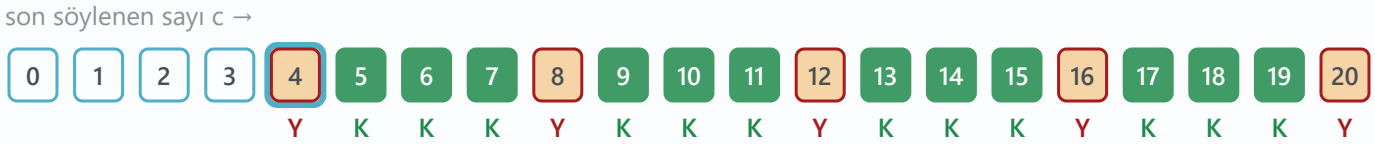
```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 5` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 4: 5, 6, 7 hepsi K (rakip kazanır) → c = 4 **Y**. Yeni kontrol sayısı!

Adım 13 / 16

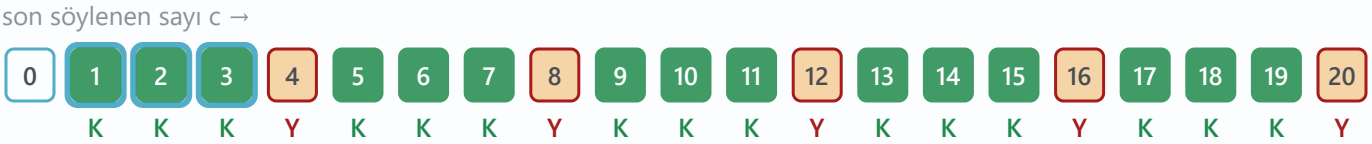
```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 4` `kazanir[c] = False`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 3…1: aynı örüntü, hepsi bir sonraki Y'ye (4) ulaşabilir → K.

Adım 14 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 1` `kazanir[c] = True`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme

son söylenen sayı c →

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Y	K	K	K	Y	K	K	K	Y	K	K	K	Y	K	K	K	Y	K	K	K	Y

K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

c = 0 (oyun başı) Y: ilk oyuncu kaybeder. İkinci oyuncu hep 4, 8, 12, 16, 20'ye tamamlar.

Adım 15 / 16

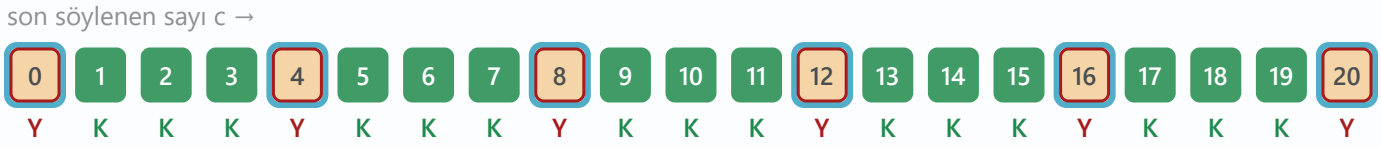
```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
▶ for c in range(HEDEF - 2, -1, -1):
▶     kazanir[c] = any(not kazanir[c + k]
▶         for k in (1, 2, 3) if c + k < HEDEF)
6 kontrol = [c for c in kazanir if not kazanir[c]]
7 print(sorted(kontrol))
```

DEĞİŞKENLER

`c = 0` `kazanir[c] = False`

ÇIKTI

Kod İzleme: Geriye Doğru Çıkarım — 21'i Söyleme



K: sırası gelen kazanır · Y (kırmızı): sırası gelen kaybeder, rakibe bırakılacak kontrol sayısı

Y konumları 4'ün katları: rakibe hep 4k bırak. Rakip x sayı söylerse sen 4 − x söyle.

Adım 16 / 16

```
1 HEDEF = 21
2 kazanir = {HEDEF - 1: False}   # c=20: kayıp
3 for c in range(HEDEF - 2, -1, -1):
4     kazanir[c] = any(not kazanir[c + k]
5         for k in (1, 2, 3) if c + k < HEDEF)
▶ kontrol = [c for c in kazanir if not kazanir[c]]
▶ print(sorted(kontrol))
```

DEĞİŞKENLER

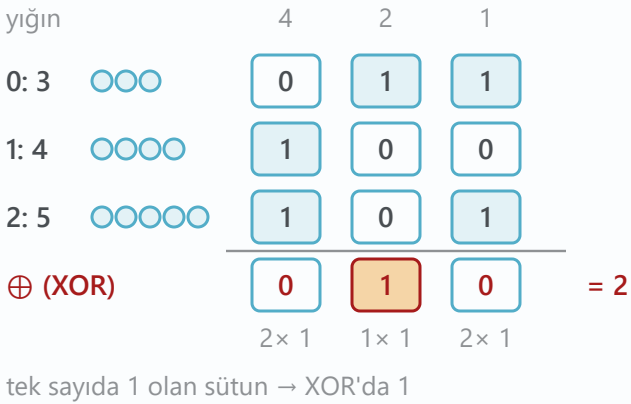
`kontrol = [0, 4, 8, 12, 16, 20]`

ÇIKTI

```
[0, 4, 8, 12, 16, 20]
```

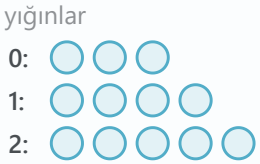
Nim: XOR ile Tam Çözülen Oyun

- **Kurallar**: birkaç jeton yığını; sırası gelen *tek* bir yığından en az bir jeton alır. Son jetonu alan kazanır.
- **Nim toplamı**: yığın boylarının bit bit XOR'u (eldesiz ikili toplama): $3 \oplus 4 \oplus 5 = 2$.
- **Bouton (1901)**: Konum kaybedendir (P) ⇔ nim toplamı 0.
- Neden? XOR 0 iken her hamle bir yığının en az bir bitini değiştirir → XOR ≠ 0. XOR = x ≠ 0 iken, x'in en soldaki 1 bitini taşıyan yığını h ⊕ x'e indirmek XOR'u 0 yapar.



Birçok başka sıralı oyun da Nim'e çevrilerek çözülebilir.

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



ikili	4	2	1
3	0	1	1
4	1	0	0
5	1	0	1
XOR	0	0	0	0

her sütunda 1'lerin sayısı çiftse XOR = 0

Konum [3, 4, 5]. Sıra bizde: nim toplamını (bit bit XOR) hesaplayalım.

Adım 1 / 7

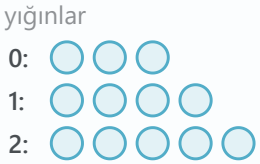
```
yiginlar = [3, 4, 5]
x = 0
for h in yiginlar:
    x ^= h              # nim toplamı (XOR)
for i, h in enumerate(yiginlar):
    if h ^ x < h:       # küçültülebilir mi?
        yiginlar[i] = h ^ x
        print("yığın", i, ":", h, "->", h ^ x)
        break
print(yiginlar)
```

DEĞİŞKENLER

`yiginlar = [3, 4, 5]` `x = 0`

ÇIKTI

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



	ikili	4	2	1
	3	0	1	1
	4	1	0	0
	5	1	0	1
XOR		3	0	1	1

her sütunda 1'lerin sayısı çiftse XOR = 0

x = 0 XOR 3 = **3** (ikili 000 ⊕ 011 = 011).

Adım 2 / 7

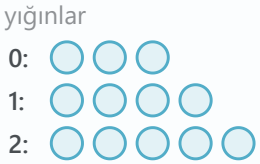
```
1  yiginlar = [3, 4, 5]
2  x = 0
▶  for h in yiginlar:
▶      x ^= h              # nim toplamı (XOR)
5  for i, h in enumerate(yiginlar):
6      if h ^ x < h:       # küçültülebilir mi?
7          yiginlar[i] = h ^ x
8          print("yığın", i, ":", h, "->", h ^ x)
9          break
10 print(yiginlar)
```

DEĞİŞKENLER

`yiginlar = [3, 4, 5]` `h = 3` `x = 3`

ÇIKTI

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



	ikili	4	2	1
	3	0	1	1
	4	1	0	0
	5	1	0	1
XOR	7	1	1	1

her sütunda 1'lerin sayısı çiftse XOR = 0

x = 3 XOR 4 = **7** (ikili 011 ⊕ 100 = 111).

Adım 3 / 7

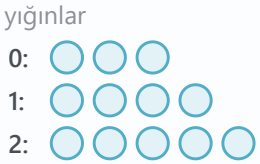
```
1  yiginlar = [3, 4, 5]
2  x = 0
▶  for h in yiginlar:
▶      x ^= h              # nim toplamı (XOR)
5  for i, h in enumerate(yiginlar):
6      if h ^ x < h:       # küçültülebilir mi?
7          yiginlar[i] = h ^ x
8          print("yığın", i, ":", h, "->", h ^ x)
9          break
10 print(yiginlar)
```

DEĞİŞKENLER

`yiginlar = [3, 4, 5]` `h = 4` `x = 7`

ÇIKTI

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



	ikili	4	2	1
	3	0	1	1
	4	1	0	0
	5	1	0	1
XOR	2	0	1	0

her sütunda 1'lerin sayısı çiftse XOR = 0

x = 7 XOR 5 = **2** (ikili 111 ⊕ 101 = 010).

Adım 4 / 7

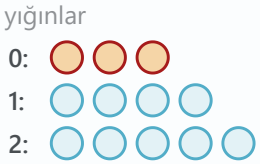
```
1  yiginlar = [3, 4, 5]
2  x = 0
▶  for h in yiginlar:
▶      x ^= h              # nim toplamı (XOR)
5  for i, h in enumerate(yiginlar):
6      if h ^ x < h:       # küçültülebilir mi?
7          yiginlar[i] = h ^ x
8          print("yığın", i, ":", h, "->", h ^ x)
9          break
10 print(yiginlar)
```

DEĞİŞKENLER

`yiginlar = [3, 4, 5]` `h = 5` `x = 2`

ÇIKTI

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



	ikili	4	2	1
	3	0	1	1
	4	1	0	0
	5	1	0	1
XOR	2	0	1	0

her sütunda 1'lerin sayısı çiftse XOR = 0

Yığın 0: 3 XOR 2 = 1 < 3 → bu yığından 2 jeton alınabilir. (XOR'un en soldaki 1 biti, 3'ün de 1 biti.)

Adım 5 / 7

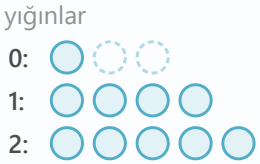
```
1  yiginlar = [3, 4, 5]
2  x = 0
3  for h in yiginlar:
4      x ^= h              # nim toplamı (XOR)
▶  for i, h in enumerate(yiginlar):
▶      if h ^ x < h:       # küçültülebilir mi?
7          yiginlar[i] = h ^ x
8          print("yığın", i, ":", h, "->", h ^ x)
9          break
10 print(yiginlar)
```

DEĞİŞKENLER

`yiginlar = [3, 4, 5]` `x = 2` `i = 0` `h = 3` `h^x = 1`

ÇIKTI

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



	ikili	4	2	1
	1	0	0	1
	4	1	0	0
	5	1	0	1
XOR	0	0	0	0

her sütunda 1'lerin sayısı çiftse XOR = 0

Yığın 0: 3 → 1. Yeni konum [1, 4, 5]: nim toplamı artık 0.


```
1  yiginlar = [3, 4, 5]
2  x = 0
3  for h in yiginlar:
4      x ^= h              # nim toplamı (XOR)
5  for i, h in enumerate(yiginlar):
6      if h ^ x < h:       # küçültülebilir mi?
           yiginlar[i] = h ^ x
           print("yığın", i, ":", h, "->", h ^ x)
           break
10 print(yiginlar)
```

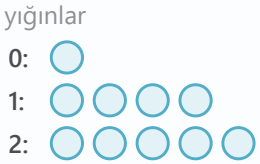
DEĞİŞKENLER

`yiginlar = [1, 4, 5]` `x = 2` `i = 0` `h = 3`

ÇIKTI

```
yığın 0 : 3 -> 1
```

Kod İzleme: Nim'de Kazandıran Hamle (XOR)



	ikili	4	2	1
	1	0	0	1
	4	1	0	0
	5	1	0	1
XOR	0	0	0	0

her sütunda 1'lerin sayısı çiftse XOR = 0

Rakip XOR = 0 konumunda: her hamlesi toplamı bozar, biz hep yeniden 0 yaparız → son jetonu biz alırız.

Adım 7 / 7

```
1 yiginlar = [3, 4, 5]
2 x = 0
3 for h in yiginlar:
4     x ^= h            # nim toplamı (XOR)
5 for i, h in enumerate(yiginlar):
6     if h ^ x < h:     # küçültülebilir mi?
7         yiginlar[i] = h ^ x
8         print("yığın", i, ":", h, "->", h ^ x)
9         break
▶ print(yiginlar)
```

DEĞİŞKENLER

`yiginlar = [1, 4, 5]` `x = 2`

ÇIKTI

```
yığın 0 : 3 -> 1
[1, 4, 5]
```

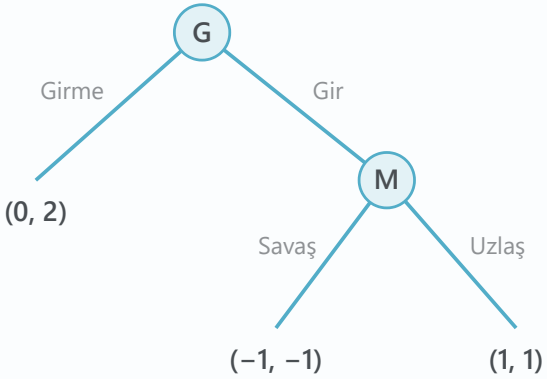
Alt Oyun ve Alt Oyun Mükemmel Denge

- **Alt oyun** (subgame): tek bir karar düğümünden başlayıp altındaki her şeyi içeren (bilgi kümelerini bölmeyen) oyun parçası.
- **Alt oyun mükemmel denge** (subgame perfect equilibrium, SPE): **her alt oyunda** Nash dengesi oluşturan strateji profili (Selten, 1965).
- Sıradan Nash dengesi, hiç ulaşılmayan düğümlerde **inandırıcı olmayan tehditlere** izin verebilir; SPE bunları eler.
- Sonlu ve mükemmel bilgili oyunlarda SPE **geriye doğru çıkarımla** (backward induction) bulunur (Kuhn/Zermelo).

Geriye doğru çıkarım

1. Yalnızca yapraklara bağlı son karar düğümlerini bul.
2. Orada karar veren oyuncunun en iyi dalını seç.
3. Düğümü, seçilen yaprağın kazancıyla değiştir.
4. Köke ulaşana kadar tekrarla.

Örnek: İnandırıcı Olmayan Tehdit



G (giren firma, Rakip) önce seçer; M (mevcut firma, Tekel), yalnızca G girerse karar verir. Normal formda iki saf Nash dengesi vardır (sağdaki tablo).

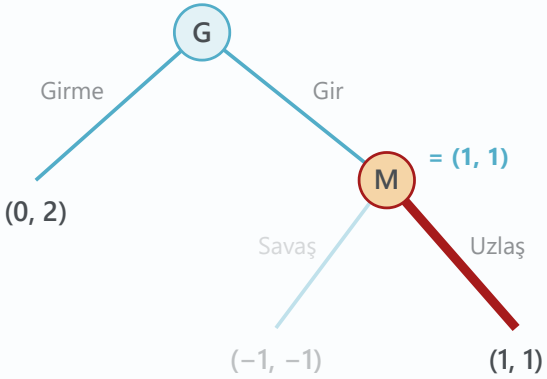
Adım 1 / 3

G \ M	Savaş	Uzlaş
Gir	(−1, −1)	(1, 1)
Girme	(0, 2)	(0, 2)

Satır: G, sütun: M. Turuncu = en iyi yanıt; çerçeveli hücre = Nash dengesi.

- (Gir, Uzlaş): SPE, geriye doğru çıkarımın sonucu.
- (Girme, Savaş): Nash dengesi, ama M'nin "girersen savaşırım" tehdidi **inandırıcı değil**: G girdiğinde savaşmak M'ye −1, uzlaşmak 1 verir.

Örnek: İnandırıcı Olmayan Tehdit



Son düğüm M: Savaş → −1, Uzlaş → 1 ⇒ **Uzlaş**. M'nin düğümünün değeri (1, 1).

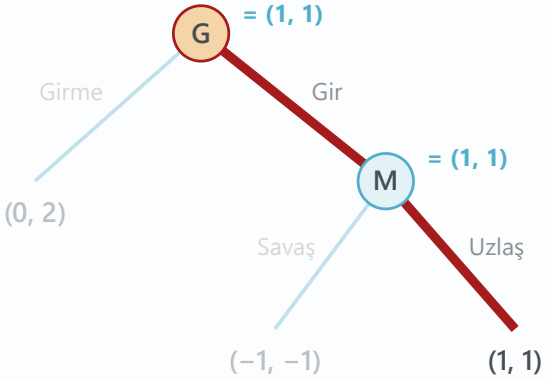
Adım 2 / 3

G \ M	Savaş	Uzlaş
Gir	(−1, −1)	(1, 1)
Girme	(0, 2)	(0, 2)

Satır: G, sütun: M. Turuncu = en iyi yanıt; çerçeveli hücre = Nash dengesi.

- (Gir, Uzlaş): SPE, geriye doğru çıkarımın sonucu.
- (Girme, Savaş): Nash dengesi, ama M'nin "girersen savaşırım" tehdidi **inandırıcı değil**: G girdiğinde savaşmak M'ye −1, uzlaşmak 1 verir.

Örnek: İnandırıcı Olmayan Tehdit



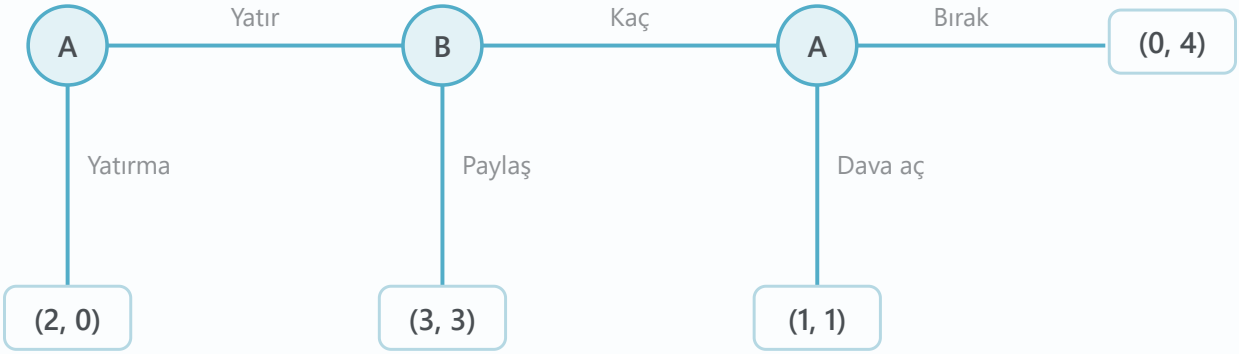
Kök G: Girme → 0, Gir → 1 (M'nin uzlaşacağını bilerek) ⇒ **Gir**. SPE: (Gir, Uzlaş), sonuç (1, 1).

G \ M	Savaş	Uzlaş
Gir	(−1, −1)	(1, 1)
Girme	(0, 2)	(0, 2)

Satır: G, sütun: M. Turuncu = en iyi yanıt; çerçeveli hücre = Nash dengesi.

- (Gir, Uzlaş): SPE, geriye doğru çıkarımın sonucu.
- (Girme, Savaş): Nash dengesi, ama M'nin "girersen savaşırım" tehdidi **inandırıcı değil**: G girdiğinde savaşmak M'ye −1, uzlaşmak 1 verir.

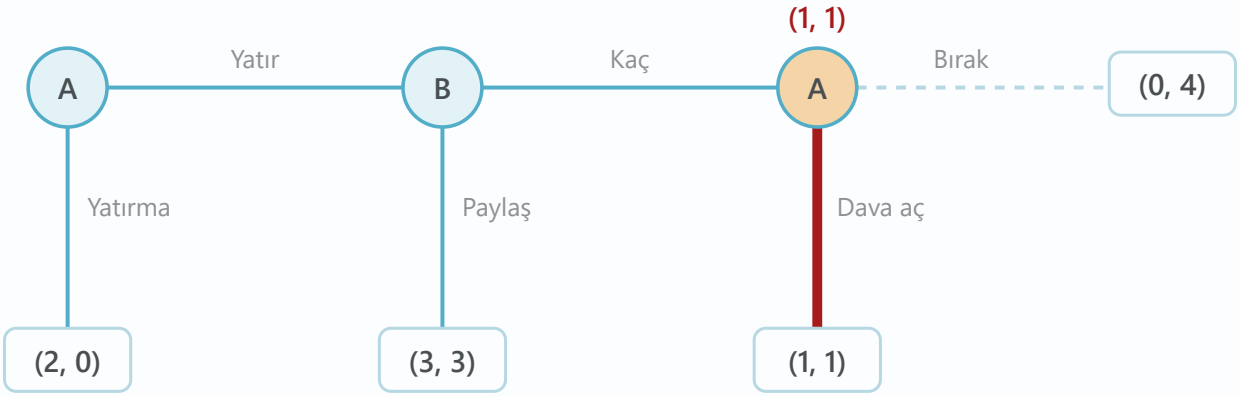
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Ayşe yatırım yapar ya da yapmaz. Yaparsa Bora kârı paylaşır ya da parayla kaçar. Kaçarsa Ayşe dava açar ya da bırakır. Çözüm **sondan** başlar.

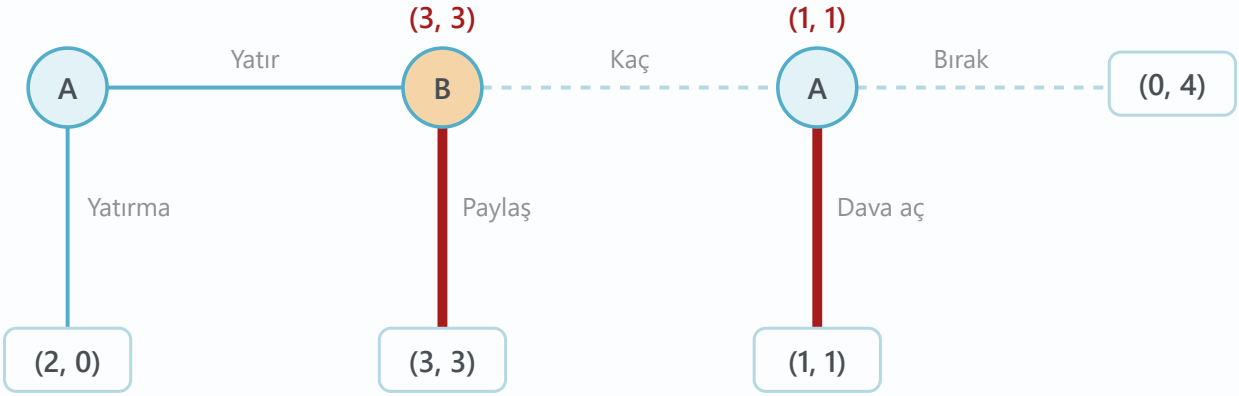
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Son karar düğümü (Ayşe, Bora kaçtıktan sonra): Dava aç 1 > Bırak 0 → **Dava aç**. Düğümün değeri (1, 1).

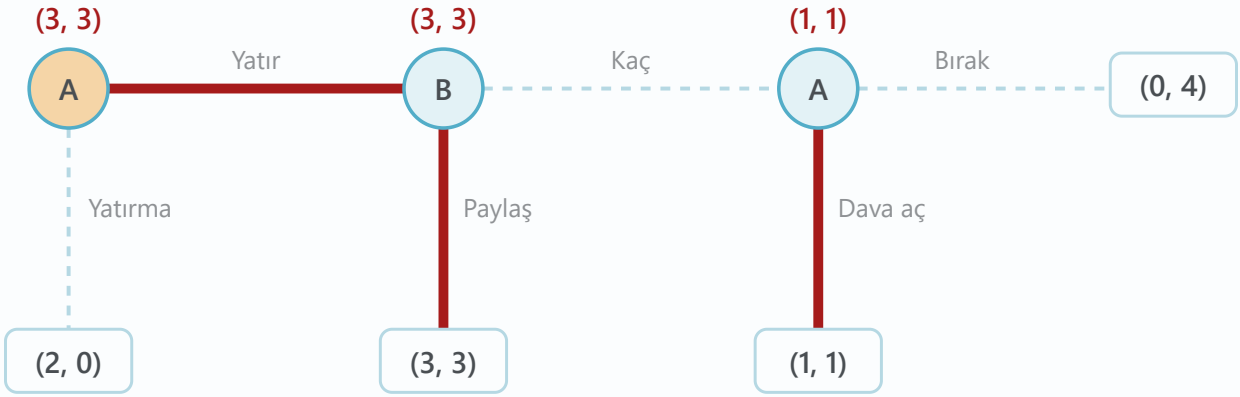
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Bora, kaçarsa dava edileceğini bilir: Paylaş 3 > Kaç 1 → **Paylaş**. Bora'nın düğümünün değeri (3, 3).

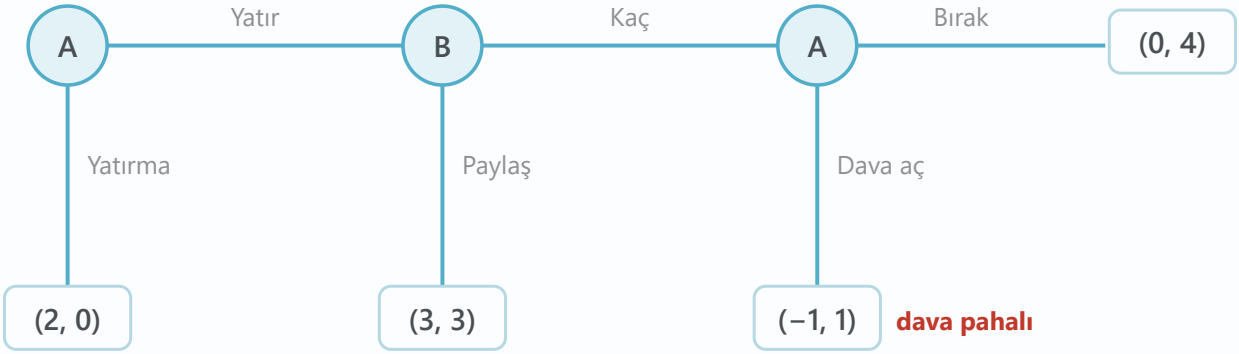
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Kök: Yatır 3 > Yatırma 2 → **Yatır**. Alt oyun mükemmel denge: Ayşe (Yatır, Dava aç), Bora Paylaş; sonuç **(3, 3)**.

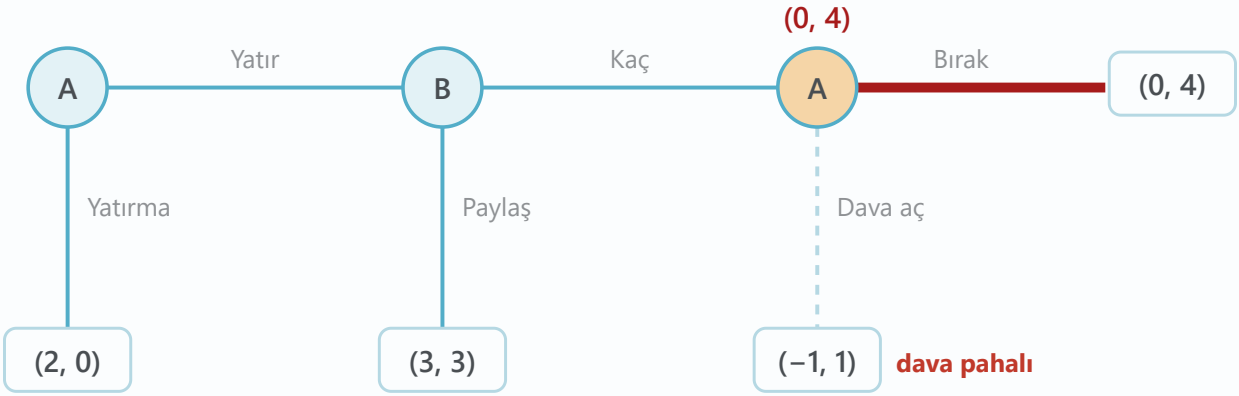
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Şimdi dava pahalı: dava açan Ayşe −1 alır. Ağaç aynı, yalnız bir yaprak değişti. Yeniden sondan başlanır.

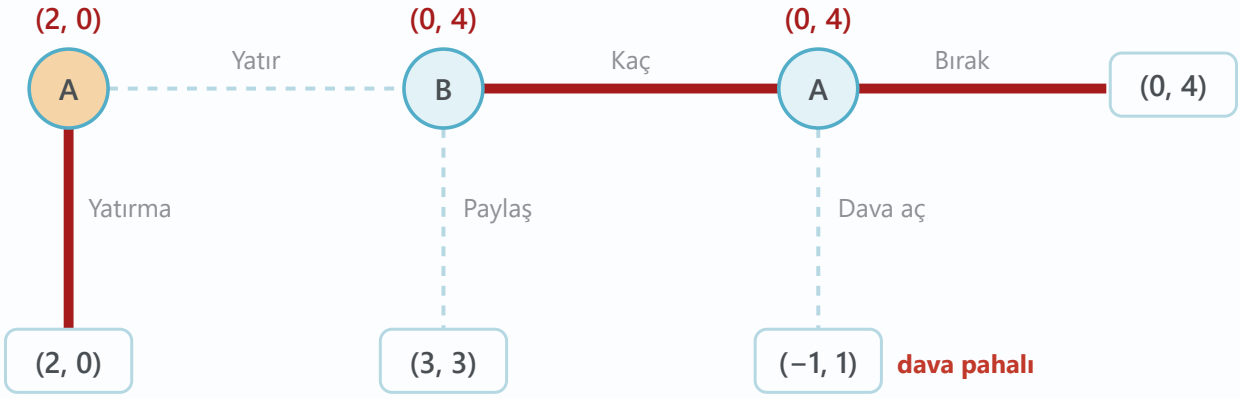
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Son düğüm: Bırak 0 > Dava aç −1 → **Bırak**. Düğümün değeri (0, 4).

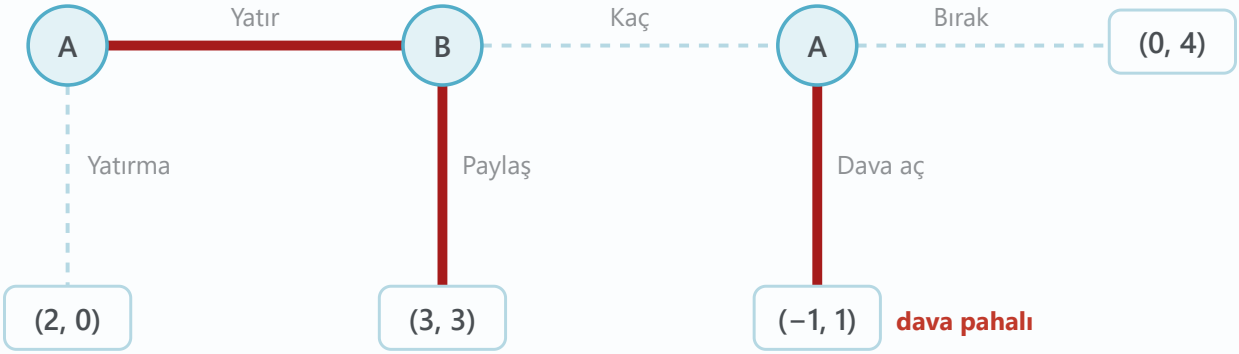
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

Bora: Kaç 4 > Paylaş 3 → **Kaç**. Kök: Yatırma 2 > 0 → **Yatırma**. Sonuç (2, 0) ikisi için de (3, 3)'ten kötü.

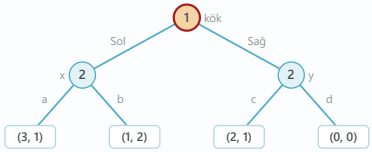
Adım Adım: Geriye Doğru Çıkarım (Yatırım Oyunu)



A: Ayşe (yatırımcı), B: Bora (girişimci); yaprakta (Ayşe, Bora)

"Kaçarsan dava açarım" planıyla (Yatır, Paylaş) yine bir **Nash dengesidir**: kimse tek başına sapmaz. Ama tehdit inandırıcı değil ($-1 < 0$): alt oyun mükemmel denge değil.

Kod İzleme: `cozum(d)`



Kökte oyuncu 1 (indis 0), x ve y'de oyuncu 2 (indis 1) seçer. cozum(kök) çağrılır.

Adım 1 / 10

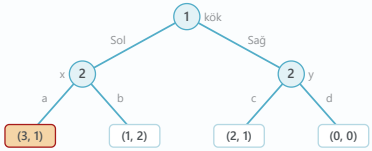
```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
▶      ad, oyuncu, dallar = d
▶      en_iyi = None
6      for hamle, cocuk in dallar:
7          v = cozum(cocuk)
8          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
9              en_iyi, secim = v, hamle
10     print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                  ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
▶  print(cozum(kok))
```

DEĞİŞKENLER

`ad = kök` `oyuncu = 0` `en_iyi = None`

ÇIKTI

Kod İzleme: `cozum(d)`



Sol dalı: cozum(x) → ilk dal a, yaprak (3, 1) olduğu gibi döner.

Adım 2 / 10

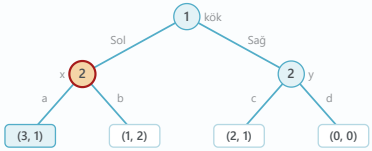
```
1  def cozum(d):
▶      if isinstance(d, tuple):          # yaprak: kazanç
▶          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
▶      for hamle, cocuk in dallar:
▶          v = cozum(cocuk)
8          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
9              en_iyi, secim = v, hamle
10     print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                  ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

DEĞİŞKENLER

`ad = x` `oyuncu = 1` `hamle = a` `v = (3, 1)`

ÇIKTI

Kod İzleme: `cozum(d)`



x'te en_iyi henüz yok: en_iyi = (3, 1), secim = a.

Adım 3 / 10

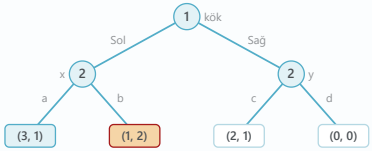
```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
6      for hamle, cocuk in dallar:
7          v = cozum(cocuk)
▶          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
▶              en_iyi, secim = v, hamle
10     print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                  ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

DEĞİŞKENLER

`ad = x` `oyuncu = 1` `v = (3, 1)` `en_iyi = (3, 1)` `secim = a`

ÇIKTI

Kod İzleme: `cozum(d)`



İkinci dal b: (1, 2). Oyuncu 2 kendi kazancına bakar: v[1] = 2 > en_iyi[1] = 1.

Adım 4 / 10

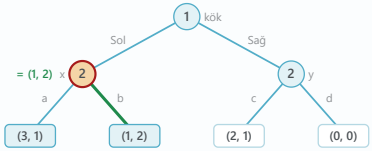
```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
▶      for hamle, cocuk in dallar:
▶          v = cozum(cocuk)
▶          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
9              en_iyi, secim = v, hamle
10     print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                  ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

DEĞİŞKENLER

`ad = x` `oyuncu = 1` `hamle = b` `v = (1, 2)` `en_iyi = (3, 1)`

ÇIKTI

Kod İzleme: `cozum(d)`



secim = b; “x b” yazılır, x'in değeri (1, 2). Oyuncu 1'in en sevdiği (3, 1) ulaşılamaz oldu.

Adım 5 / 10

```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
6      for hamle, cocuk in dallar:
7          v = cozum(cocuk)
8          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
▶              en_iyi, secim = v, hamle
▶      print(ad, secim)
▶      return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                   ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

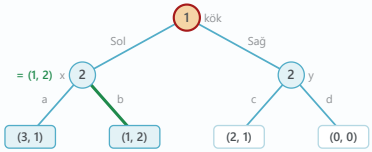
DEĞİŞKENLER

`ad = x` `en_iyi = (1, 2)` `secim = b`

ÇIKTI

```
x b
```

Kod İzleme: `cozum(d)`



Kökte Sol'un değeri (1, 2): en_iyi = (1, 2), secim = Sol.

Adım 6 / 10

```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
6      for hamle, cocuk in dallar:
▶          v = cozum(cocuk)
▶          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
▶              en_iyi, secim = v, hamle
10     print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                   ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

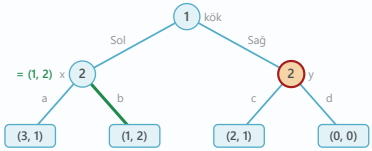
DEĞİŞKENLER

`ad = kök` `oyuncu = 0` `hamle = Sol` `v = (1, 2)` `en_iyi = (1, 2)` `secim = Sol`

ÇIKTI

```
x b
```

Kod İzleme: `cozum(d)`



Sağ dalı: cozum(y). c → (2, 1), d → (0, 0); oyuncu 2 için 1 > 0, secim = c.

Adım 7 / 10

```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
▶      for hamle, cocuk in dallar:
▶          v = cozum(cocuk)
▶          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
▶              en_iyi, secim = v, hamle
10     print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                  ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

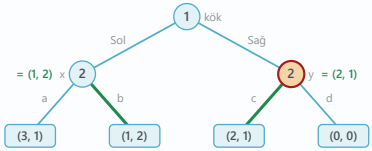
DEĞİŞKENLER

`ad = y` `oyuncu = 1` `en_iyi = (2, 1)` `secim = c`

ÇIKTI

```
x b
```

Kod İzleme: `cozum(d)`



“y c” yazılır; y'nin değeri (2, 1) döner.

Adım 8 / 10

```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
6      for hamle, cocuk in dallar:
7          v = cozum(cocuk)
8          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
9              en_iyi, secim = v, hamle
▶      print(ad, secim)
▶      return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                   ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

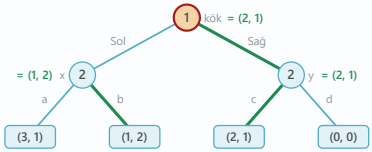
DEĞİŞKENLER

`ad = y` `en_iyi = (2, 1)` `secim = c`

ÇIKTI

```
x b
y c
```

Kod İzleme: `cozum(d)`



Kökte oyuncu 1 karşılaştırır: Sağ 2 > Sol 1 → secim = Sağ. “kök Sağ” yazılır.

Adım 9 / 10

```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
6      for hamle, cocuk in dallar:
7          v = cozum(cocuk)
▶          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
▶              en_iyi, secim = v, hamle
▶      print(ad, secim)
11     return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                   ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
14 print(cozum(kok))
```

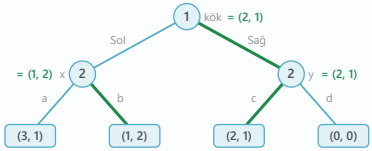
DEĞİŞKENLER

`ad = kök` `oyuncu = 0` `v = (2, 1)` `en_iyi = (2, 1)` `secim = Sağ`

ÇIKTI

```
x b
y c
kök Sağ
```

Kod İzleme: `cozum(d)`



Sonuç (2, 1). SPE: (Sağ; x'te b, y'de c). Ulaşılmayan x'teki b de stratejinin parçasıdır.

Adım 10 / 10

```
1  def cozum(d):
2      if isinstance(d, tuple):          # yaprak: kazanç
3          return d
4      ad, oyuncu, dallar = d
5      en_iyi = None
6      for hamle, cocuk in dallar:
7          v = cozum(cocuk)
8          if en_iyi is None or v[oyuncu] > en_iyi[oyuncu]:
9              en_iyi, secim = v, hamle
10     print(ad, secim)
▶      return en_iyi
12 kok = ['kök', 0, [('Sol', ['x', 1, [('a', (3, 1)), ('b', (1, 2))]]),
13                   ('Sağ', ['y', 1, [('c', (2, 1)), ('d', (0, 0))]])]]
▶ print(cozum(kok))
```

DEĞİŞKENLER

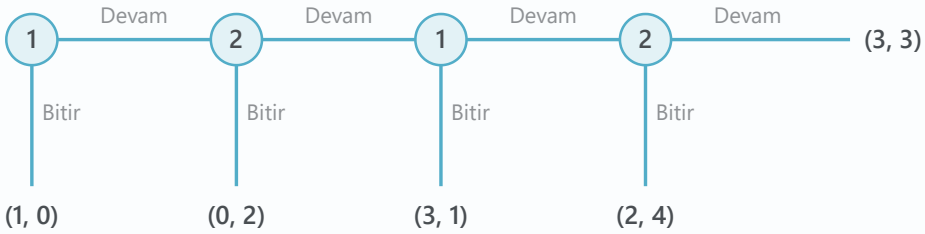
`sonuç = (2, 1)`

ÇIKTI

```
x b
y c
kök Sağ
(2, 1)
```

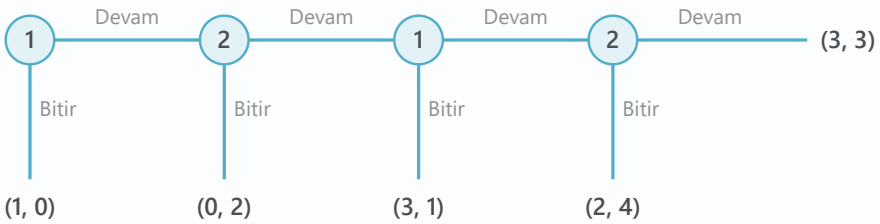
Kırkayak Oyunu (Centipede Game)

- İki oyuncu sırayla karar verir (Rosenthal, 1981). Her düğümde oyuncu ya **Bitir** der ve o anki paylaşım gerçekleşir, ya da **Devam** der ve toplam büyür.
- Toplam kazanç büyür (1 → 2 → 4 → 6; sonda da 6), ama devam eden oyuncu, rakibi bir sonraki adımda bitirirse daha az alır.



Düğümdeki sayı karar veren oyuncudur. Yapraklarda (Oyuncu 1, Oyuncu 2) kazançları; en sağda herkes devam ederse (3, 3).

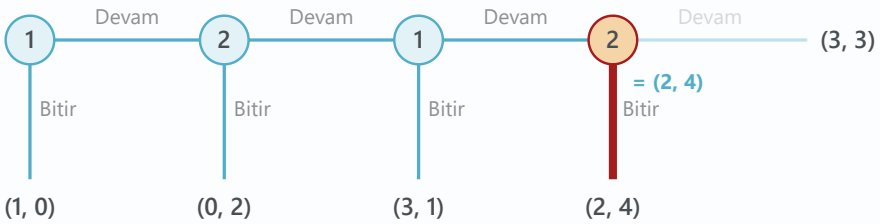
Kırkayak: Geriye Doğru Çıkarım Adım Adım



Düğüm	Karar veren	Bitir	Devamın değeri	Karar	Düğüm değeri
4	Oyuncu 2	(2, 4)	?	?	?
3	Oyuncu 1	(3, 1)	?	?	?
2	Oyuncu 2	(0, 2)	?	?	?
1	Oyuncu 1	(1, 0)	?	?	?

Oyuncu 1 başlar; her düğümde oyuncu **Bitir** (aşağı) ya da **Devam** (sağa) der. Son karar düğümünden (4. düğüm, Oyuncu 2) başlayıp geriye gideceğiz.

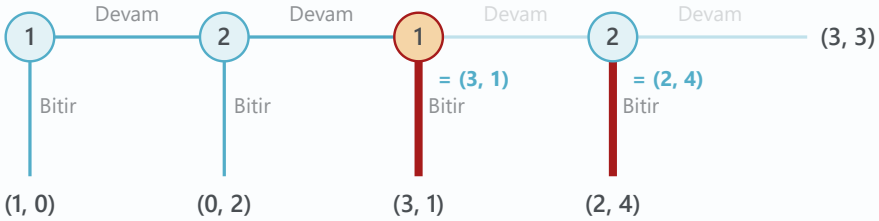
Kırkayak: Geriye Doğru Çıkarım Adım Adım



Düğüm	Karar veren	Bitir	Devamın değeri	Karar	Düğüm değeri
4	Oyuncu 2	(2, 4)	(3, 3)	**Bitir**	(2, 4)
3	Oyuncu 1	(3, 1)	?	?	?
2	Oyuncu 2	(0, 2)	?	?	?
1	Oyuncu 1	(1, 0)	?	?	?

4. düğüm, Oyuncu 2: Bitir → 4, Devam → 3 (devamın değeri (3, 3)). 4 > 3 ⇒ **Bitir**; düğümün değeri (2, 4).

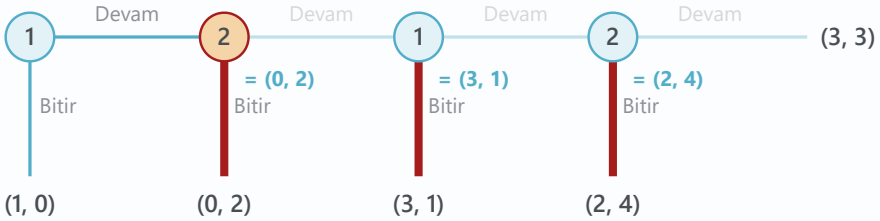
Kırkayak: Geriye Doğru Çıkarım Adım Adım



Düğüm	Karar veren	Bitir	Devamın değeri	Karar	Düğüm değeri
4	Oyuncu 2	(2, 4)	(3, 3)	**Bitir**	(2, 4)
3	Oyuncu 1	(3, 1)	(2, 4)	**Bitir**	(3, 1)
2	Oyuncu 2	(0, 2)	?	?	?
1	Oyuncu 1	(1, 0)	?	?	?

3. düğüm, Oyuncu 1: Bitir → 3, Devam → 2 (devamın değeri (2, 4)). 3 > 2 ⇒ **Bitir**; düğümün değeri (3, 1).

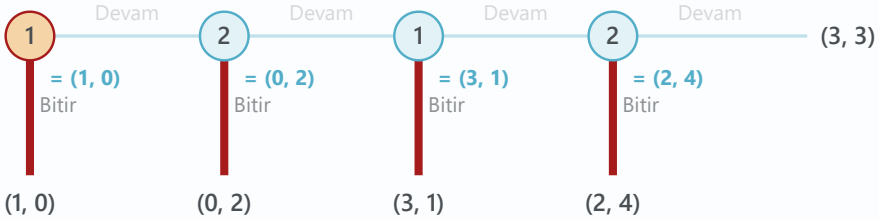
Kırkayak: Geriye Doğru Çıkarım Adım Adım



Düğüm	Karar veren	Bitir	Devamın değeri	Karar	Düğüm değeri
4	Oyuncu 2	(2, 4)	(3, 3)	**Bitir**	(2, 4)
3	Oyuncu 1	(3, 1)	(2, 4)	**Bitir**	(3, 1)
2	Oyuncu 2	(0, 2)	(3, 1)	**Bitir**	(0, 2)
1	Oyuncu 1	(1, 0)	?	?	?

2. düğüm, Oyuncu 2: Bitir → 2, Devam → 1 (devamın değeri (3, 1)). 2 > 1 ⇒ **Bitir**; düğümün değeri (0, 2).

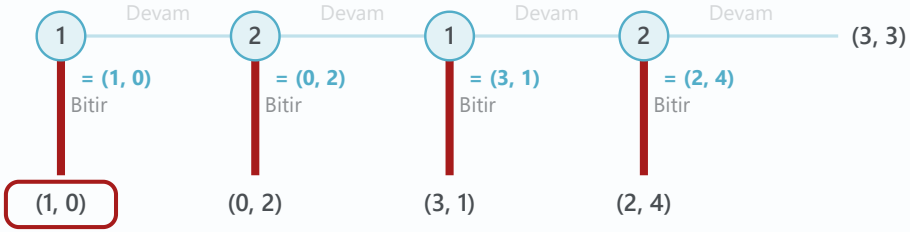
Kırkayak: Geriye Doğru Çıkarım Adım Adım



Düğüm	Karar veren	Bitir	Devamın değeri	Karar	Düğüm değeri
4	Oyuncu 2	(2, 4)	(3, 3)	**Bitir**	(2, 4)
3	Oyuncu 1	(3, 1)	(2, 4)	**Bitir**	(3, 1)
2	Oyuncu 2	(0, 2)	(3, 1)	**Bitir**	(0, 2)
1	Oyuncu 1	(1, 0)	(0, 2)	**Bitir**	(1, 0)

1. düğüm, Oyuncu 1: Bitir → 1, Devam → 0 (devamın değeri (0, 2)). 1 > 0 ⇒ **Bitir**; düğümün değeri (1, 0).

Kırkayak: Geriye Doğru Çıkarım Adım Adım



Düğüm	Karar veren	Bitir	Devamın değeri	Karar	Düğüm değeri
4	Oyuncu 2	(2, 4)	(3, 3)	**Bitir**	(2, 4)
3	Oyuncu 1	(3, 1)	(2, 4)	**Bitir**	(3, 1)
2	Oyuncu 2	(0, 2)	(3, 1)	**Bitir**	(0, 2)
1	Oyuncu 1	(1, 0)	(0, 2)	**Bitir**	(1, 0)

Alt oyun mükemmel denge: her oyuncu **her düğümde Bitir**. Oyun ilk hamlede (1, 0) ile biter; oysa (3, 3) ikisi için de daha iyiydi.

Kırkayak Oyunu: Teori ve Deney

Teori (SPE)

Her oyuncu her düğümde Bitir der; oyun ilk hamlede (1, 0) ile biter. Oyunun herhangi bir Nash dengesinde de sonuç aynıdır.

Verimsizlik

(3, 3) her iki oyuncu için de (1, 0)'dan iyidir: SPE sonucu **Pareto optimal değildir**. Düğüm sayısı arttıkça kayıp büyür.

Deney

Laboratuvarda oyuncuların çoğu ilk düğümde bitirmez; birkaç tur devam eder (McKelvey ve Palfrey, 1992).

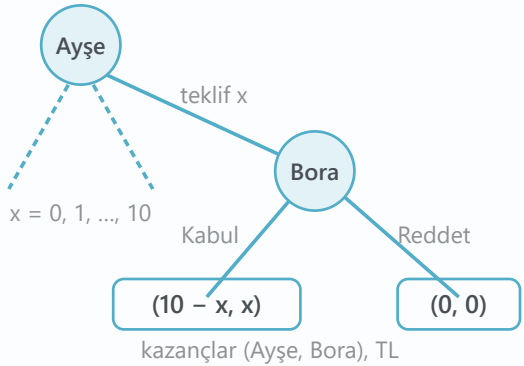
Neden?

Geriye doğru çıkarım, rasyonelliğin **ortak bilgi** olmasını ve uzun bir akıl yürütme zincirini gerektirir; rakibin biraz "iyi niyetli" olma ihtimali bile devam etmeyi kârlı kılar.

Kırkayak, sonlu tekrarlanan Mahkûmun İkilemi ile aynı mantığı paylaşır: son turdan geriye bakınca işbirliği çözülür. Sonsuz ufukta ise işbirliği dengede kalabilir.

Últimatom Oyunu: Pazarlıkta Geriye Doğru Çıkarım

- **Kural:** Ayşe 10 TL'nin x TL'sini Bora'ya önerir (tam TL). Bora **kabul** ederse paylaşım (10 − x, x) olur, **reddederse** ikisi de 0 alır (Güth, Schmittberger ve Schwarze, 1982).
- **Geriye doğru çıkarım:** Bora'nın düğümünde x > 0 için kabul (x) retten (0) iyidir. Ayşe bunu bilerek en küçük pozitif teklifi verir: **SPE sonucu (9, 1)**. (Bora x = 0'da kayıtsız; 0'ı da kabul ediyorsa (10, 0) da bir SPE'dir.)
- **İnandırıcı olmayan tehdit:** "4 TL'den azını reddederim" sözü bir Nash dengesini destekler, ama 1 TL önerildiğinde reddetmek Bora'ya 0 < 1 verir.



Teklif (toplamın oranı)	Kuram (yalnız para)	Deneylerde
En sık teklif	en küçük birim	%40–50
%20'nin altı	kabul edilir	yaklaşık yarısı reddedilir

Sık hata: Deney sonucunu "oyuncular geriye doğru çıkarım yapamıyor" diye okumak. Ret, adalet kaygısının fayda fonksiyonunda yer aldığını gösterir; teklif verenler de bunu bekleyip cömert davranır (Camerer, 2003).

Sonsuz Tekrarlı Mahkûmun İkilemi: Tetik Stratejisi

Kinci tetik stratejisi (grim trigger): Sessiz başla; rakip bir kez ihanet ederse sonsuza dek İhanet. Aynı tek atışlık oyun tekrarlanınca sonucu belirleyen, oyunun **ne zaman biteceğinin bilinip bilinmediğidir**.

Sonlu ve bilinen ufuk

Son turda gelecek yoktur, ceza da yoktur: orada İhanet baskındır. Geriye doğru çıkarım bunu bir önceki tura, oradan ilk tura taşır; her turda İhanet (kırkayaktaki gibi).

Sonsuz ufuk (her tur δ olasılıkla sürer)

Bugünkü ihanetin kazancı, gelecekteki kalıcı cezanın kaybıyla tartılır. δ yeterince büyükse (oyuncular sabırlıysa) sapmak kârlı değildir ve işbirliği dengede kalır.

Ceza inandırıcıdır, çünkü (İhanet, İhanet) tek atışlık oyunun dengesidir: iki kinci tetik stratejisi birlikte **alt oyun mükemmel dengedir**.

Sık Yapılan Hatalar: Sıralı Oyunlar

Kökten başlamak
İlk hamle, sonraki hamlelere verilecek tepkiye bağlıdır; çözüm her zaman son karar düğümlerinden başlar.

Yanlış kişinin kazancı
Her düğümde yalnız *orada karar veren* oyuncunun sayısı karşılaştırılır (Bora'nın düğümünde ikinci sayı).

Stratejiyi yol sanmak
Strateji, ulaşılmayan düğümler dahil her düğüm için seçimi içerir: Ayşe'nin stratejisi (Yatır, Dava aç).

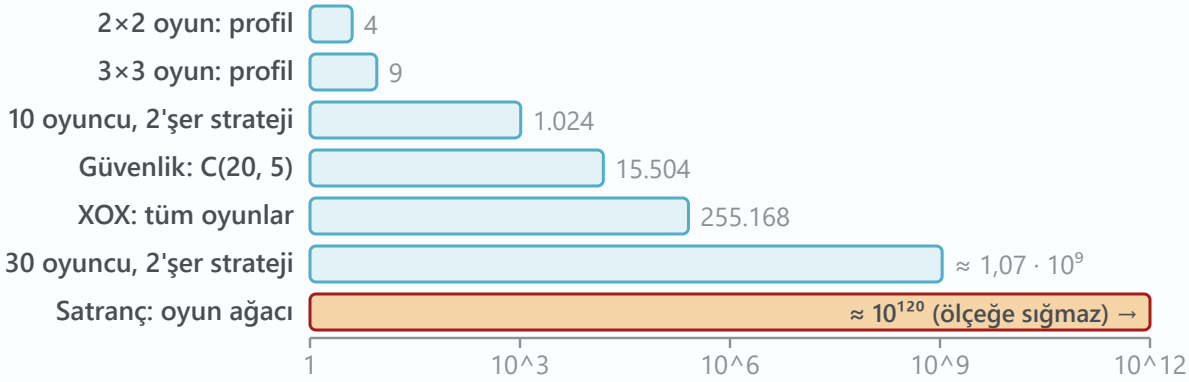
Nash = alt oyun mükemmel sanmak
Her alt oyun mükemmel denge Nash dengesidir; tersi doğru değil: inandırıcı olmayan tehdit içeren Nash dengeleri vardır.

Eşzamanlıyı sıralı çözmek
Hamleler görülmüyorsa (Mahkûmun İkilemi) ağaçta bilgi kümesi vardır; geriye doğru çıkarım doğrudan uygulanmaz, matris çözülür.

Söz = taahhüt sanmak
"Dava açarım" demek yetmez; tehdidin uygulanması o anda kârlı olmalı ya da önceden bağlanmalı (sözleşme, teminat).

ALGORİTMİK BAKIŞ

Algoritmik Oyun Kuramı Neden “Algoritmik”?



Büyüklük

Oyunun tanımı bile üstel büyür: n oyuncu × 2 strateji → 2^n profil. Kaba kuvvet tarama ölçeklenmez.

Karmaşıklık

Nash dengesi hep var, ama onu bulmak iki oyunculu oyunlarda bile hesaplama açısından zordur.

Yapı ve tasarım

Özel yapı verimlilik getirir: sıfır toplamlıda doğrusal programlama, Nim'de XOR. Kurallar da algoritmadır.

ALGORİTMİK BAKIŞ

Algoritmik Oyun Kuramının Dört Sorusu

1. Hesaplama: denge bulunabilir mi?

Sıfır toplamlı oyunlar doğrusal programlamayla hızlı çözülür. Genel oyunlarda Nash dengesini hızlı bulan bir yöntem bilinmiyor; bu problem **PPAD-tam** denen zor problemler sınıfındadır. Hex'i büyük tahtalarda çözmek çok zordur (**PSPACE-tam**).

2. Verimlilik: bencillik ne kaybettirir?

Anarşinin Bedeli = en kötü dengenin maliyeti / en iyi sonucun maliyeti. Mahkûmun İkileminde hapis yılını maliyet sayarsak denge toplam 4 yıl, (Sessiz, Sessiz) 2 yıl: oran 4/2 = 2.

3. Tasarım: hangi kurallar?

Mekanizma tasarımı, oyuncuların özel bilgisini dürüstçe bildirmesini sağlayan (teşvik uyumlu) kuralları arar: açık artırmalar, eşleştirme, oylama.

4. Dinamik: oyuncular dengeye nasıl ulaşır?

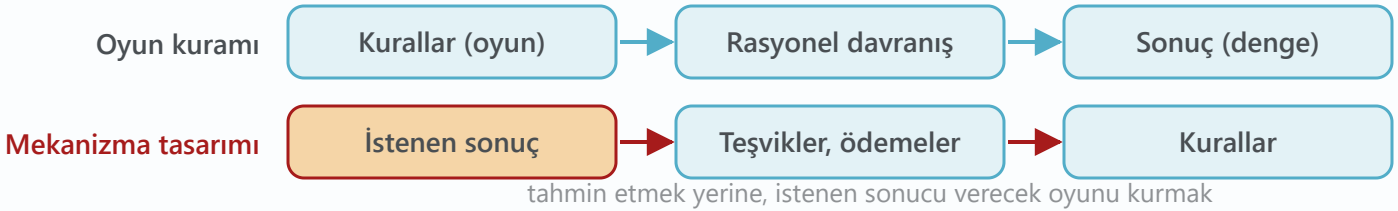
Tekrarlanan ve çevrim içi oyunlarda bilgi zamanla gelir; oyuncular geçmişten öğrenerek stratejisini uyarlar.

Dört sorunun ortak dili: oyunların sınıflandırılması, klasik oyunların dengeleri ve sıralı oyunlarda geriye doğru çıkarım.

MEKANİZMA TASARIMINA BAKIŞ

Mekanizma Tasarımı: Tersine Oyun Kuramı

- Oyun kuramı verilen bir oyunun sonucunu tahmin eder; **mekanizma tasarımı** istenen sonucu verecek kuralları ve teşvikleri tasarlar.
- Zorluk: Ajanların özel bilgisi (değerleme, konum, tercih) tasarımcıya kapalıdır; ajanlar yalan söyleyebilir.



Para ile

Tasarımcı ödeme alabilir; ödemeler doğru söylemeyi teşvik eder.

`açık artırmalar, reklam yerleşimi`

Para olmadan

Teşvikler yalnızca kuralla sağlanır; genellikle hedeften biraz ödün verilir.

`seçimler, oda takası, tesis yeri`

MEKANİZMA TASARIMINA BAKIŞ

Gerçek Dünya Uygulamaları

Arama reklamları
Reklam alanları her aramada **genelleştirilmiş ikinci fiyat** (GSP) açık artırmasıyla dağıtılır. Çok alanlı GSP'de dürüst teklif baskın değildir.

Spektrum açık artırmaları
Devletler frekans lisanslarını tasarlanmış açık artırmalarla satar (ABD'de 1994'ten beri).

Okul yerleştirme
Öğrenci önerili Gale–Shapley algoritmasında öğrencinin gerçek tercihlerini yazması en iyisidir; sonuç kararlı bir eşleştirmedir.

Oda ve vardiya takası
Herkes elindekini getirir; takas döngüleriyle kimse eskisinden kötü bir şey almaz ve para kullanılmaz.

Trafik ve yönlendirme
Her sürücü kendi en hızlı yolunu seçer; bencil seçimlerin toplam gecikmeye maliyeti Anarşinin Bedeli ile ölçülür.

Platformlar ve teşvikler
Yoğunluk fiyatlandırması sürücüleri yoğun saatlere çeker; Bitcoin madencileri işlemleri ödül için doğrular.

Ortak fikir: kurallar öyle tasarlanır ki herkes **kendi çıkarına** davransa bile sistem iyi çalışır.

Özet: Kavram Haritası

Kavram	Kısa tanım	Örnek
Normal / genişletilmiş form	Matris (eşzamanlı) ya da sırayı ve bilgiyi gösteren oyun ağacı	TKM, fiyat rekabeti; satranç
Sınıflandırma	Zamanlama, anlaşma, toplam kazanç, simetri, bilgi, ufuk	Satranç: sıralı, sıfır toplamlı, mükemmel bilgili
Baskın strateji, Nash dengesi	Karşı taraf ne yaparsa yapsın daha iyi; tek taraflı sapma kârsız	İhanet; (İhanet, İhanet)
Pareto optimallik	Kimseyi kötüleştirmeden iyileştirilemez	Denge (−2, −2) değil
Klasik 2×2 oyunlar	Çatışma ya da koordinasyon; saf dengeler en iyi tepkiyle bulunur	Yazı-Tura: yok; Geyik Avı: iki
Yinelemeli eleme, seviye-k	Baskılananları tekrar tekrar sil; inanca göre en iyi tahmin	2/3 oyunu: 90 → 60 → 40 → … → 0
Minimaks, Nim	Yapraktan köke en iyi seçim; P/N konumları	Ağaç değeri 3; XOX berabere; Nim XOR
Alt oyun mükemmel denge	Her alt oyunda Nash; geriye doğru çıkarımla bulunur	Piyasaya giriş; kırkayak (1, 0)
Kinci tetik stratejisi	Sonsuz tekrarda ihaneti kalıcı cezayla caydırmak	Sabırlı oyuncularda (δ büyük) işbirliği

Örnek Problemler (1/2)

1. Fiyat rekabeti oyununda baskın stratejiler ve denge nedir? Denge Pareto optimal mi?
2. Oyun 0–100 aralığında ve hedef “ortalamanın yarısı” olsaydı yinelemeli eleme nasıl ilerlerdi, denge ne olurdu?
3. 2/3 oyununda (0–90) sınıfın %20'si seviye 0, %80'i seviye 1 düşünüyorsa hangi seviye kazanır?
4. (A, A) = (4, 1), (B, B) = (1, 4), diğer hücreler (0, 0). Saf dengeler hangileri? Oyun hangi klasik oyuna benzer?
5. Altı klasik 2×2 oyundan hangileri sıfır toplamlı, hangileri simetrik?

▶ **Cevapları göster**

ÖZET

Örnek Problemler (2/2)

6. Nim konumu [2, 5, 6]. Sıra sizde: kazanan konum mu? Kazandıran hamle hangisi?
7. MAX kökünün üç MIN çocuğu var: B = [7, 2, 9], C = [4, 6], D = [1, 8]. Oyunun değeri ve MAX'ın seçimi nedir?
8. "16'yı Söyleme": sırayla 1–3 ardışık sayı söylenir, 16'yı söyleyen kaybeder. Kontrol sayıları nedir? Kim kazanır?
9. Kırkayak'ta en sağdaki kazanç (5, 5) olsaydı alt oyun mükemmel denge ne olurdu?
10. Piyasaya giriş oyununda (Girme, Savaş) neden alt oyun mükemmel değildir?

▶ **Cevapları göster**

ÖZET

Etkileşimli Oyunlar

2/3 oyunu dışındakiler iki kişilik, sıralı, **mükemmel bilgili** ve şanssız oyunlardır: kuramsal olarak geriye doğru çıkarımla çözülebilirler, ama ağaç çoğu zaman çok büyüktür.

2/3 Ortalaması Oyunu
Gerçek tahminlerle yinelemeli eleme.

Minimaks XOX
Tüm ağacı arayan rakip: en iyi ihtimalle berabere.

Nim
XOR ile tamamen çözülen sıralı oyun.

21'i Söyleme
Kaybettiren sayıdan geriye doğru akıl yürütme.

Dörtlü Bağla
1988'de çözüldü: mükemmel oyunda ilk oyuncu kazanır.

Hex
Beraberlik yok; strateji çalma: ilk oyuncu kazanır.

Othello (Reversi)
8×8 tahtada mükemmel oyun beraberlik (2023).

Mangala
Hamle ağacında ileriye bakma.

Satranç
Zermelo: sonuç belirli ama bilinmiyor.

Dama
2007'de çözüldü: mükemmel oyunda beraberlik.

Tilki ve Kazlar
Asimetrik roller, mükemmel bilgi.

Dokuz Taş
Klasik strateji oyunu; mükemmel bilgi.

Kaynaklar (1/2)

- Nisan, Roughgarden, Tardos ve Vazirani (ed.), *Algorithmic Game Theory*, Cambridge University Press, 2007 (Bölüm 1).
- Roughgarden, *Twenty Lectures on Algorithmic Game Theory*, Cambridge University Press, 2016.
- Osborne, *An Introduction to Game Theory*, Oxford University Press, 2004 (Bölüm 2–5) · Shoham ve Leyton-Brown, *Multiagent Systems*, Cambridge University Press, 2009 (Bölüm 3 ve 5).
- Nash, "Equilibrium points in n-person games", *PNAS* 36(1), 1950 · von Neumann, "Zur Theorie der Gesellschaftsspiele", *Mathematische Annalen* 100, 1928.
- Bouton, "Nim, a game with a complete mathematical theory", *Annals of Mathematics* 3, 1901–1902.
- Nagel, "Unraveling in guessing games: An experimental study", *American Economic Review* 85(5), 1995.
- R. W. Rosenthal, "Games of perfect information, predatory pricing and the chain-store paradox", *J. Economic Theory* 25, 1981 · R. D. McKelvey, T. R. Palfrey, "An experimental study of the centipede game", *Econometrica* 60(4), 1992.
- W. Güth, R. Schmittberger, B. Schwarze, "An experimental analysis of ultimatum bargaining", *J. Economic Behavior & Organization* 3(4), 1982 · C. F. Camerer, *Behavioral Game Theory*, Princeton University Press, 2003.
- B. Skyrms, *The Stag Hunt and the Evolution of Social Structure*, Cambridge University Press, 2004.
- S. Reisch, "Hex ist PSPACE-vollständig", *Acta Informatica* 15, 1981 · J. Schaeffer vd., "Checkers is solved", *Science* 317(5844), 2007.

ÖZET

Kaynaklar (2/2): Ek Kitaplar, Ders Sayfaları, Araçlar

Ek kitaplar

- Karlin ve Peres, *Game Theory, Alive*, American Mathematical Society, 2017. Kombinatoryal oyunlar, Nim, 2×2 oyunlar.
- Easley ve Kleinberg, *Networks, Crowds, and Markets*, Cambridge University Press, 2010. Oyunlar, ağlar ve piyasalar.
- Hearn ve Demaine, *Games, Puzzles, and Computation*, A K Peters, 2009. Oyunların hesaplama karmaşıklığı.

Ders sayfaları

- Roughgarden, CS364A (Stanford): Twenty Lectures kitabının dersi.
- Shah, CSC304 (Toronto) · Procaccia, 15-896 (CMU)
- Dinitz, AGT (Johns Hopkins) · Hoefer, AGT (Frankfurt)

Araçlar ve video

- Gambit: oyun ağaçları ve matris oyunlarında denge hesaplayan açık kaynak araçlar.
- Nashpy: iki oyunculu oyunlar için Python kütüphanesi.
- Game Theory 101 (YouTube oynatma listesi).

Son

Bölüm 2: Oyunlar ve Temsiller

Sonraki bölüm: Bölüm 3: Nash Dengesi

Çalışma soruları · Terim sözlüğü