Minimaks ve Alfa-Beta Budaması

Bölüm 2 · Algoritmik Oyun Kuramı

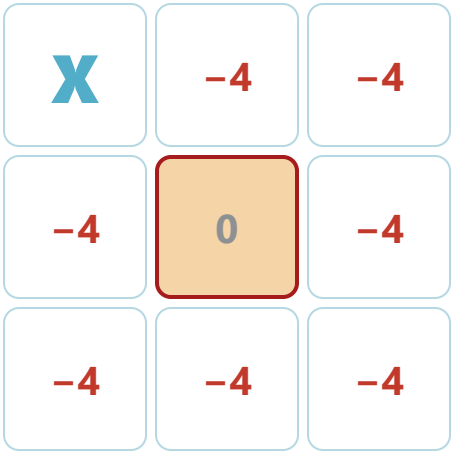
Oyun ağacı | Minimaks | Nim | XOX'ta minimaks | Alfa-beta budaması | Hamle sıralaması | Dörtlü Bağla | Derinlik sınırlı arama

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_02_Minimaks_ve_Budama.html

GİRİŞ

Oyunla Başlangıç: YZ'ye Karşı XOX



Yeni oyun: insan başlar | **Yeni oyun: YZ başlar** | **YZ oynasın**

Örnek: sol üst köşeye X kondu. YZ (O) her boş kareyi tam ağaçla değerlendirdi; sayılar YZ'nin gözünden değer (+ kazanır, 0 berabere, − kaybeder). Seçim: **merkez kare (0)**. Merkez kare beraberliği garanti eder; iki taraf da hatasız oynarsa oyun berabere biter. Diğer karelerin hepsi kaybettirir. (59.705 düğüm)

Bir kareye tıklanınca insan oyuncunun taşı konur. YZ oynamadan önce her boş kareye o kareyi seçerse oyunun sonunda ne alacağını yazar (+ kazanır, 0 berabere, − kaybeder); turuncu kare YZ'nin seçimidir. Hamleyi **YZ oynasın** düğmesi yapar.

Merak sorusu: YZ bu sayıları nereden biliyor? Köşeyle açılan oyunda merkez dışındaki her cevap neden kaybettiriyor?

GİRİŞ

Öğrenme Çıktıları

Oyun ağacı

Konumu düğüm, hamleyi kenar olarak çizmek; MAX ve MIN katmanlarını, yaprakları ve faydayı tanımak; ağaç boyutunu b^d ile kestirmek.

Minimaks

Değerleri yapraktan köke elle yükseltmek; kökün seçtiği hamleyi ve nedenini söylemek; kodu satır satır izlemek.

Nim ve XOX

Nim'de minimaksın P/N konumlarıyla ve XOR kuralıyla aynı sonucu verdiğini; XOX'ta derinlik cezasının neden gerektiğini göstermek.

Alfa-beta budaması

α ve β'yı tanımlamak, her düğümde [α, β] aralığını izlemek, α ≥ β olduğunda budanan dalları bulmak ve sonucun değişmediğini açıklamak.

Hamle sıralaması

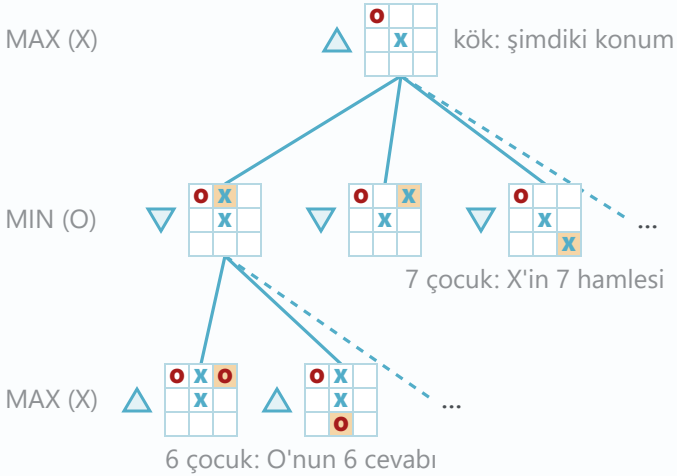
En iyi ve en kötü sıralamada değerlendirilen yaprak sayısını ($b^{d/2}$ ile b^d) karşılaştırmak.

Derinlik sınırlı arama

Dörtlü Bağla için değerlendirme fonksiyonu yazmak; ufuk etkisini, yinelemeli derinleştirmeyi ve geçiş tablosunu açıklamak.

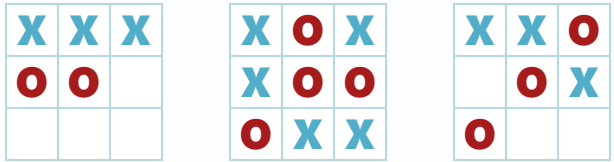
Oyun Ağacı: Konumlar ve Hamleler

- **Düğüm** bir konumdur (tahta ve sıradaki oyuncu); **kenar** bir hamledir. **Kök** şimdiki konumdur.
- İki oyuncu sırayla oynar, şans yoktur, iki taraf da tahtayı görür: **mükemmel bilgili**, **sıfır toplamlı** oyun.
- Kökte sırası olan oyuncuya **MAX** denir (▲): değeri büyütmek ister. Rakibi **MIN**'dir (▽): değeri küçültmek ister.
- Katmanlar sırayla MAX ve MIN olur; bir katman inmek bir yarım hamle (bir oyuncunun bir hamlesi) demektir.
- Aynı konuma farklı sıralarla gelinebilir; ağaçta bu konum birden çok düğüm olarak görünür.



Yapraklar ve Fayda

- **Yaprak** (oyun sonu): kazanan belli ya da tahta dolu; çocuğu yoktur.
- Her yaprağın bir **faydası** vardır. XOX'ta MAX'ın (X) gözünden: X kazanırsa **+1**, berabere **0**, O kazanırsa **−1**.
- Oyun sıfır toplamlıdır: O'nun faydası X'inkinin eksisidir ($u_O = -u_X$). Bu yüzden ağaçta tek bir sayı yeter.
- İç düğümlerin değeri henüz bilinmez; minimaks onları yapraklardan hesaplar.



Fayda yalnız yapraklarda tanımlıdır. “Merkez iyi bir karedir” gibi bir yargı ancak ağaç yapraklara kadar aranınca sayıya dönüşür.

OYUN AĞACI

Dallanma Etkeni ve Derinlik

- **Dallanma etkeni** b: bir konumdaki yasal hamle sayısı (ortalama ya da en çok). **Derinlik** d: oyunun yarım hamle cinsinden uzunluğu.
- Her düğümün b çocuğu varsa d derinlikte b^d yaprak olur: **ağaç boyutu ≈ b^d**. Derinlik bir artınca iş b katına çıkar.

Oyun	b	d (yarım hamle)	Ağaç ya da konum sayısı
XOX	≤ 9	≤ 9	9! = 362.880 üst sınır; gerçekte 255.168 oyun, 549.946 düğüm
Dörtlü Bağla	≤ 7	≤ 42	4.531.985.219.092 ≈ $4{,}5 \cdot 10^{12}$ yasal konum (Tromp)
Satranç	≈ 35	≈ 80	$35^{80} \approx 10^{123}$; Shannon'un (1950) kaba tahmini ≈ 10^{120} oyun
Go (19×19)	≈ 250	≈ 150	≈ $2{,}1 \cdot 10^{170}$ yasal konum (Tromp, 2016)

XOX'un tüm ağacı bir saniyenin altında aranır. Dörtlü Bağla'da bile tam arama için akıllı yöntemler gerekir; satranç ve Go'da arama belli bir derinlikte durdurulur.

Minimaks Değeri

Bir s konumunun **minimaks değeri**, iki taraf da en iyi oynarsa MAX'ın oyunun sonunda alacağı faydadır:

```
minimaks(s) =
  fayda(s)                          s oyun sonuysa
  max  minimaks(sonuç(s, a))        sıra MAX'taysa
   a
  min  minimaks(sonuç(s, a))        sıra MIN'deyse
   a
```

- Tanım özyinelemelidir: arama **derinlik öncelikli** iner, değerler yapraktan köke **yükselir**.
- Kökte MAX, değeri en büyük çocuğa giden hamleyi seçer.

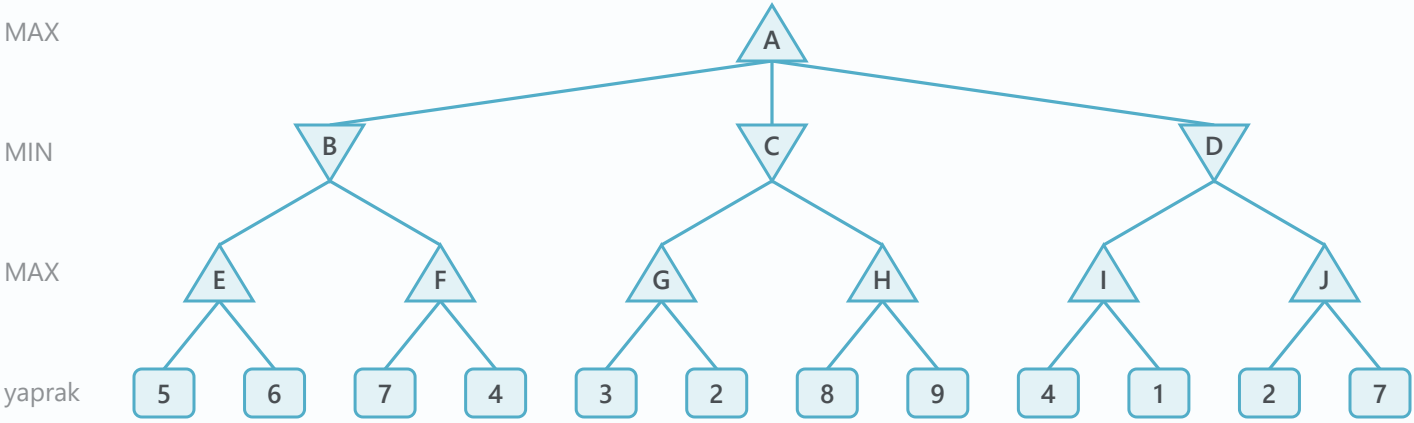
Maliyet

- Zaman: her düğüm bir kez → $O(b^d)$.
- Bellek: yalnız kökten o anki düğüme giden yol ve kardeş bilgisi → $O(b \cdot d)$.
- Rakip hatasız oynarsa minimaks hamlesi en iyisidir; rakip hata yaparsa MAX en az minimaks değerini alır.

MİNİMAKS

Kökün Değeri Ne Olur?

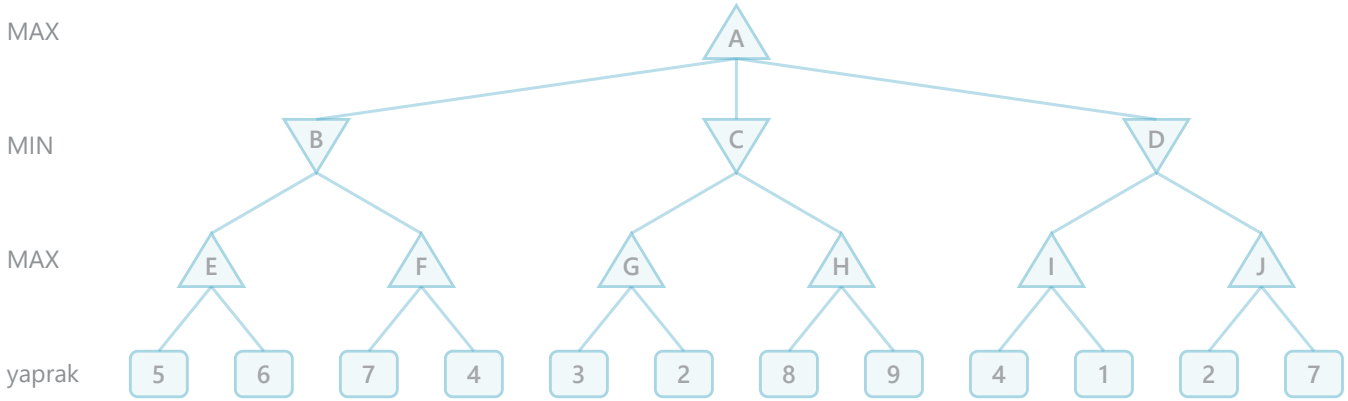
Üç katmanlı ağaçta kök MAX, ikinci katman MIN, üçüncü katman MAX. Kökün değeri kaçtır ve MAX hangi hamleyi seçer?



Cevabı göster

MİNİMAKS

Minimaks Adım Adım



Başlangıç: değerler yapraklarda. Adım düğmesiyle derinlik öncelikli arama başlar; değerler yapraktan köke yükselir.

Adım **0 / 32** · Değerlendirilen yaprak: **0 / 12**

Geri | Adım | Oynat | Sıfırla

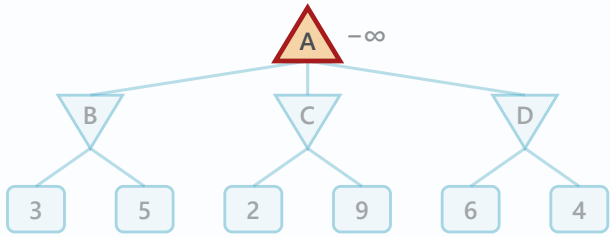
Turuncu: şu an işlenen düğüm · kırmızı çerçeve: yığındaki düğümler · gri sayı: henüz kesinleşmemiş değer · kalın kenar: kökün seçtiği hamle.

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI





İlk çağrı: kök A bir MAX düğümü, deger = -inf.

Adım 1 / 14

DEĞİŞKENLER

`n = "A"` `maks = True` `deger = -inf`

ÇAĞRI YIĞINI

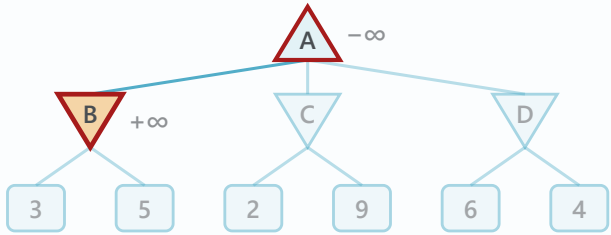
`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
▶    deger = -inf if maks else inf
8    for c in T[n]:
9      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI





minimaks("B") çağrılır: MIN, deger = inf.

Adım 2 / 14

DEĞİŞKENLER

`n = "B"` `maks = False` `deger = inf`

ÇAĞRI YIĞINI

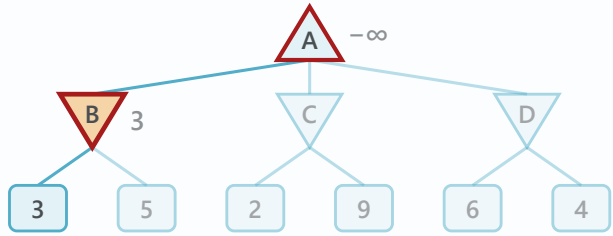
`minimaks("B", False)`

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
▶        deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI



minimaks(3) bir yaprak: 3 döner. min ile deger = 3.

Adım 3 / 14

DEĞİŞKENLER

`n = "B"` `maks = False` `c = 3` `v = 3` `deger = 3`

ÇAĞRI YIĞINI

`minimaks("B", False)`

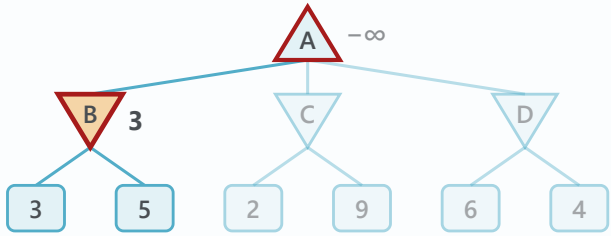
`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
▶        deger = min(deger, v)
▶    return deger
15 print(minimaks("A", True))
```

ÇIKTI





minimaks(5) bir yaprak: 5 döner. min ile deger = 3; B bitti, 3 döner.

Adım 4 / 14

DEĞİŞKENLER

`n = "B"` `maks = False` `c = 5` `v = 5` `deger = 3`

ÇAĞRI YIĞINI

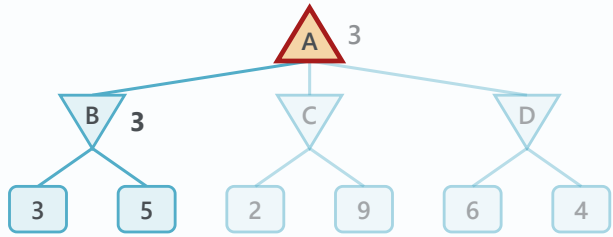
`minimaks("B", False)`

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
▶        deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI



B çağrısı 3 döndürdü. max ile deger = 3.

Adım 5 / 14

DEĞİŞKENLER

`n = "A"` `maks = True` `c = "B"` `v = 3` `deger = 3`

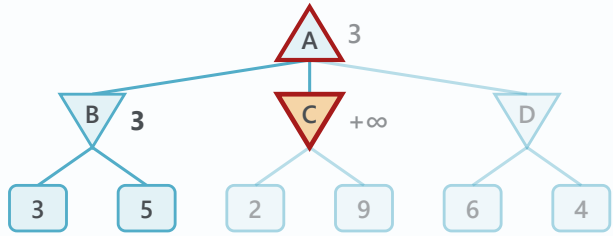
ÇAĞRI YIĞINI

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI



minimaks("C") çağrılır: MIN, deger = inf.

Adım 6 / 14

DEĞİŞKENLER

`n = "C"` `maks = False` `deger = inf`

ÇAĞRI YIĞINI

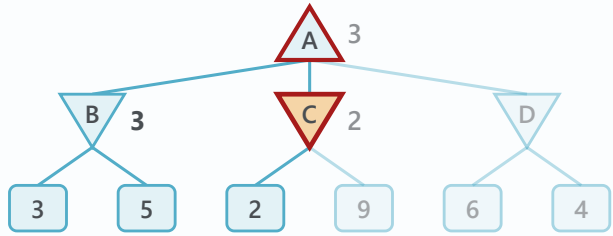
`minimaks("C", False)`

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
▶        deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI



minimaks(2) bir yaprak: 2 döner. min ile deger = 2.

Adım 7 / 14

DEĞİŞKENLER

`n = "C"` `maks = False` `c = 2` `v = 2` `deger = 2`

ÇAĞRI YIĞINI

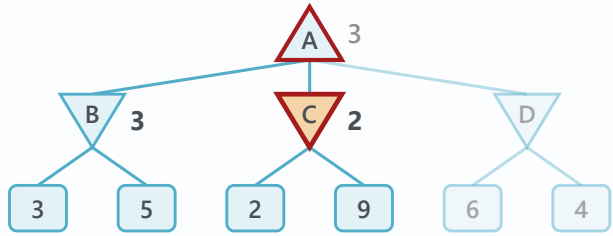
`minimaks("C", False)`

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
▶        deger = min(deger, v)
▶    return deger
15 print(minimaks("A", True))
```

ÇIKTI



minimaks(9) bir yaprak: 9 döner. min ile deger = 2; C bitti, 2 döner.

Adım 8 / 14

DEĞİŞKENLER

`n = "C"` `maks = False` `c = 9` `v = 9` `deger = 2`

ÇAĞRI YIĞINI

`minimaks("C", False)`

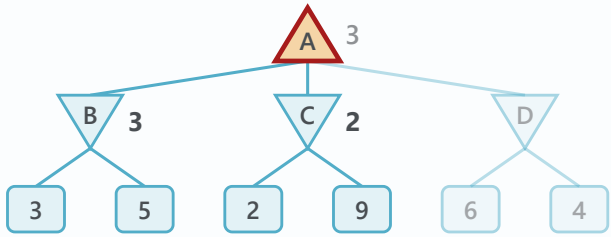
`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
▶        deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI





C çağrısı 2 döndürdü. max ile deger = 3.

Adım 9 / 14

DEĞİŞKENLER

`n = "A"` `maks = True` `c = "C"` `v = 2` `deger = 3`

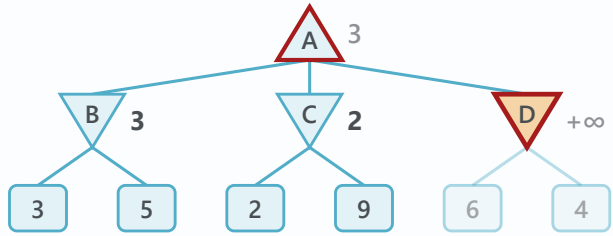
ÇAĞRI YIĞINI

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI



minimaks("D") çağrılır: MIN, deger = inf.

Adım 10 / 14

DEĞİŞKENLER

`n = "D"` `maks = False` `deger = inf`

ÇAĞRI YIĞINI

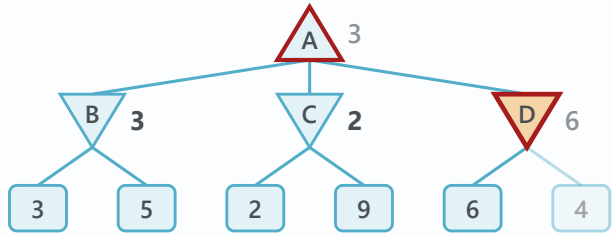
`minimaks("D", False)`

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
▶        deger = min(deger, v)
14   return deger
15 print(minimaks("A", True))
```

ÇIKTI



minimaks(6) bir yaprak: 6 döner. min ile deger = 6.

Adım 11 / 14

DEĞİŞKENLER

`n = "D"` `maks = False` `c = 6` `v = 6` `deger = 6`

ÇAĞRI YIĞINI

`minimaks("D", False)`

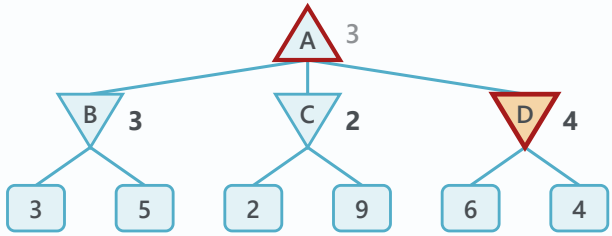
`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
▶        deger = min(deger, v)
▶    return deger
15 print(minimaks("A", True))
```

ÇIKTI





minimaks(4) bir yaprak: 4 döner. min ile deger = 4; D bitti, 4 döner.

Adım 12 / 14

DEĞİŞKENLER

`n = "D"` `maks = False` `c = 4` `v = 4` `deger = 4`

ÇAĞRI YIĞINI

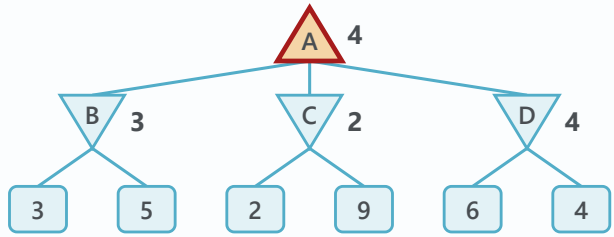
`minimaks("D", False)`

`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = minimaks(c, not maks)
10     if maks:
▶        deger = max(deger, v)
12     else:
13       deger = min(deger, v)
▶    return deger
15 print(minimaks("A", True))
```

ÇIKTI



D çağrısı 4 döndürdü. max ile deger = 4; A bitti, 4 döner.

Adım 13 / 14

DEĞİŞKENLER

`n = "A"` `maks = True` `c = "D"` `v = 4` `deger = 4`

ÇAĞRI YIĞINI

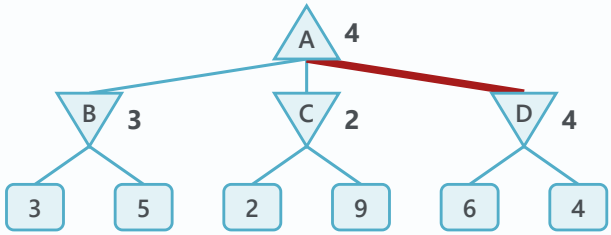
`minimaks("A", True)`

Kod İzleme: Minimaks Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def minimaks(n, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = minimaks(c, not maks)
10     if maks:
11       deger = max(deger, v)
12     else:
13       deger = min(deger, v)
14   return deger
▶ print(minimaks("A", True))
```

ÇIKTI

```
4
```



Kökün değeri 4: A, D'ye giden hamleyi seçer. 6 yaprağın hepsi okundu.

Adım 14 / 14

DEĞİŞKENLER

`sonuç = 4`

ÇAĞRI YIĞINI

boş

Minimaks, Geriye Doğru Çıkarım ve Maksimin

- Minimaks, **geriye doğru çıkarımın** iki kişilik, sıfır toplamlı özel hâlidir: genel oyunda her düğümde sıradaki oyuncu *kendi* kazancını büyütür; sıfır toplamda B'nin kazancı A'nınkinin eksisi olduğundan "B büyütür" = "A'nınkini küçültür".
- **Maksimin**: MAX'ın garanti edebileceği en iyi değer, $\max_s \min_t u(s, t)$. **Minimaks**: MIN'in MAX'ı tutabileceği en düşük değer, $\min_t \max_s u(s, t)$.
- Mükemmel bilgili sonlu oyunda ikisi eşittir ve saf stratejiler yeter (Zermelo, 1913; Kuhn, 1953). Ağaçtaki minimaks değeri bu ortak değerdir.

Eşzamanlı oyunda fark

Yazı-Tura'da (eşleşirse A +1, eşleşmezse −1) saf stratejilerde maksimin −1, minimaks +1'dir: aralık kapanmaz.

von Neumann (1928): karışık stratejilere izin verilince her sonlu iki kişilik sıfır toplamlı oyunda maksimin = minimaks (**minimaks teoremi**). Yazı-Tura'da ortak değer 0'dır.

MİNİMAKS

Minimaksın Varsayımı: Hatasız Rakip

Ne garanti eder?

MAX, rakip ne yaparsa yapsın en az minimaks değerini alır. Rakip hata yaparsa daha fazlasını alabilir, daha azını asla.

Ne garanti etmez?

Zayıf bir rakibe karşı en çok kazancı getiren hamleyi bulmaz: değeri eşit iki hamleden rakibe tuzak kuranı ayırt etmez.

Şans varsa

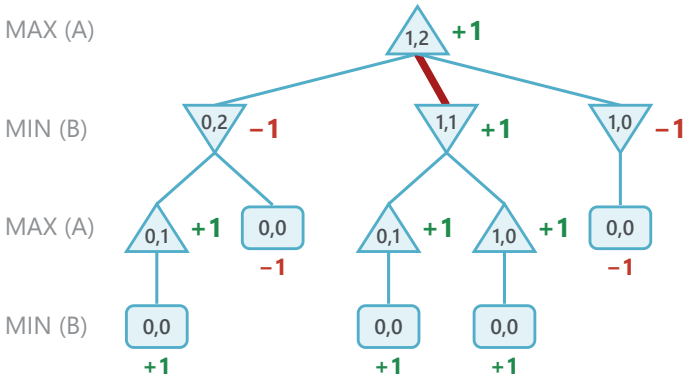
Tavla gibi zarlı oyunlarda ağaca **şans düğümleri** eklenir ve orada ortalama alınır (expectiminimaks); budama zorlaşır.

Sıfır toplamlı iki kişilik oyunda “rakibim en iyi cevabı verir” varsayımı güvenli bir alt sınır verir: minimaks hamlesi, en kötü durumdaki kazancı en büyükleyen hamledir.

NİM

Nim'in Oyun Ağacı

Konum: **(1, 1)** **(1, 2)** **(3)** **(1, 3)** **(2, 2)** **Değerleri gizle**



Kök (1, 2): değer **+1**, sıradaki A kazanır → **N konumu**. Kazandıran hamle: **(1, 1)**. Nim toplamı 1 ⊕ 2 = 3 ≠ 0; XOR'u 0 yapan hamle aynısı ✓.

Sırası gelen tek bir yığından en az bir jeton alır, son jetonu alan kazanır. Düğümde yığın boyları; +1: A (ilk oynayan) kazanır. Nim oyunu →

P ve N Konumları = Minimaks Değeri

- **N konumu**: sıradaki oyuncu kazanır. **P konumu**: bir önceki oyuncu (sıradaki değil) kazanır.
- Kurallar: hamlesi olmayan konum P'dir; bir konum N'dir ⇔ en az bir hamle P'ye gider; P'dir ⇔ her hamle N'ye gider.
- Değer *sıradaki oyuncunun* gözünden yazılırsa (+1 kazanır, −1 kaybeder): `değer(s) = max`$_a$ `-değer(sonuç(s, a))`. Bu, minimaksın tek fonksiyonla yazılışıdır (**negamaks**).
- "En az bir çocuk −1 ise +1" kuralı, N tanımının ta kendisidir: N ⇔ +1, P ⇔ −1.

Konum	Sıradakine değer	Tür
(0, 0)	−1	P
(1, 0)	+1	N
(1, 1)	−1	P
(2, 0)	+1	N
(1, 2)	+1	N
(2, 2)	−1	P
(1, 2, 3)	−1	P

NİM

Minimaks ve Bouton'un XOR Kuralı

Bouton (1901): konum P'dir ⇔ yığınların nim toplamı (XOR) 0'dır. Kazandıran hamle XOR'u 0 yapan hamledir. Minimaksın tam ağaçta bulduğu hamleyle karşılaştırma:

Konum	XOR	Minimaksın kazandıran hamlesi	XOR'u 0 yapan hamle	Tam ağaç
(1, 2)	3	→ (1, 1)	→ (1, 1) ✓	12 düğüm
(1, 3)	2	→ (1, 1)	→ (1, 1) ✓	28 düğüm
(2, 3)	1	→ (2, 2)	→ (2, 2) ✓	86 düğüm
(1, 2, 3)	0	yok: P konumu	yok ✓	447 düğüm
(3, 4, 5)	2	→ (1, 4, 5)	→ (1, 4, 5) ✓	1.038.768 düğüm

İki yöntem her konumda aynı hamleyi bulur; fark maliyettedir. Minimaksın ağacı jeton sayısıyla üstel büyür, XOR kuralı yığın sayısı kadar işlemle biter. Oyuna özgü bir kuram bulunduğunda aramaya gerek kalmaz; bulunamadığında minimaks her zaman çalışır.

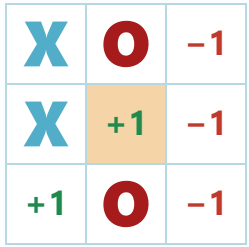
XOX'ta Değer Ölçeği ve Derinlik Cezası

- ±1 ölçeğinde hızlı kazanç ile yavaş kazanç aynı değeri alır. Sağdaki konumda X sol altla *hemen* kazanır, merkezle iki hamle sonra kazanır: ikisi de +1.
- Eşitlikte ilk denenen kare seçilir; merkez önce denenirse YZ hazır kazancı bırakıp oyunu uzatır. Kaybederken de en çabuk kaybedeni seçebilir.
- **Derinlik cezası**: d kökten itibaren hamle sayısıysa

```
X kazanırsa  +(10 − d)
O kazanırsa  −(10 − d)
berabere      0
```

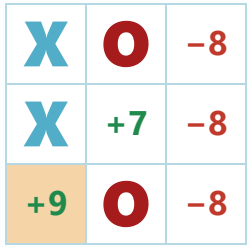
Hızlı kazanç ve geç kayıp tercih edilir: +9 > +7, −6 > −8.

±1 ölçeği: merkez seçilir



merkez de sol alt da +1

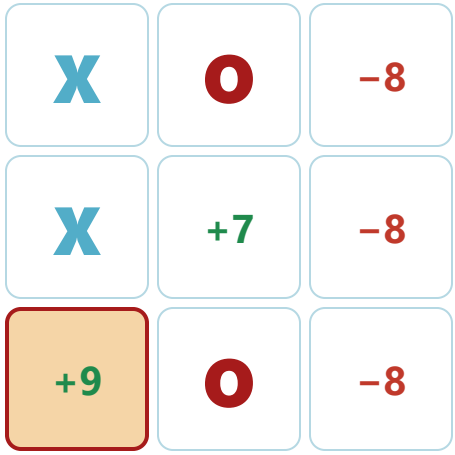
derinlik cezası: sol alt seçilir



+9 hemen kazanç, +7 iki hamle sonra

XOX'TA MİNİMAKS

XOX Laboratuvarı



Sıra: X (MAX) | Derinlik cezası: açık | En iyi hamleyi oyna

Boş tahta | Çatal | Engel | Ceza örneği

Sıra X'te (MAX, en büyüğü seçer): **sol alt kare, +9**. Sol alt kare üçlüyü tamamlar: hemen kazanır.

Minimaks: **113** düğüm (0 ms) · Alfa-beta: **32** düğüm (0,1 ms) · kök değeri ikisinde de **+9**

Bir kareye tıklanınca boş → X → O → boş döner; böylece istenen konum kurulur. Sayılar X'in gözünden minimaks değeri: MAX (X) en büyüğü, MIN (O) en küçüğü seçer; turuncu kare seçimdir. Gerekçe değerlerden sonra kurallarla açıklanır.

Hamlenin Gerekçesi

Sıra X: kazanır

X	X	+9
O	O	0
−8	−8	−8

üçlüyü tamamlar

Sıra O: engeller

X	X	0
+8	O	+8
+8	+8	+8

tek berabere kare

Sıra X: çatal kurar

X	O	0
+7	X	0
+7	0	O

iki tehdit kurar

Sıra O: berabere tutar

X	+4	+4
+4	0	+4
+4	+4	+4

diğerleri kaybettirir

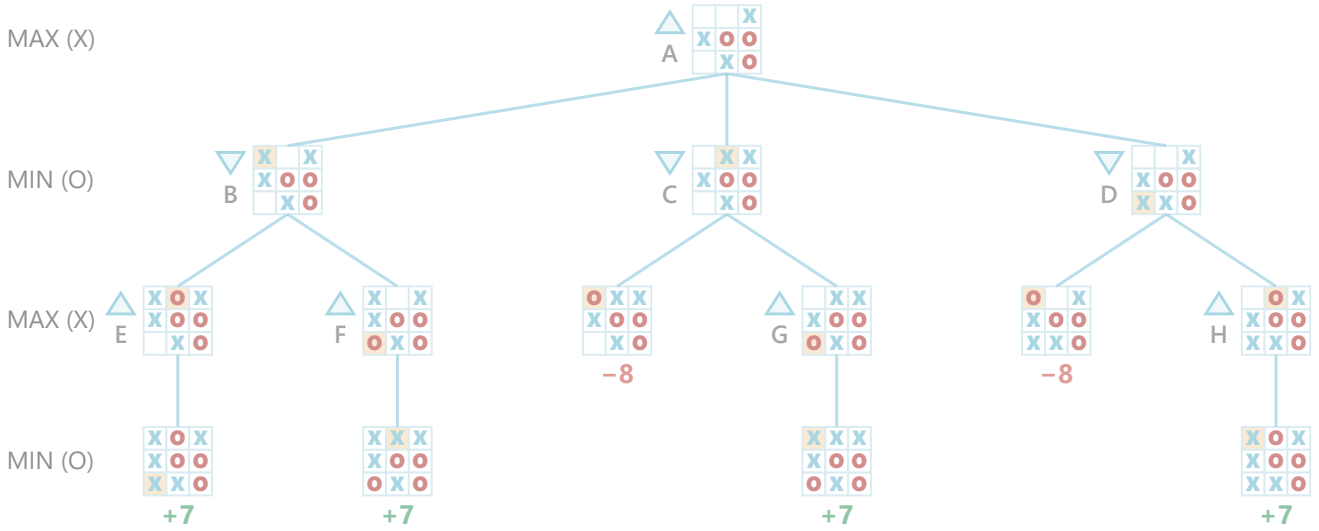
- Seçimi yalnız değerler yapar; gerekçe, seçilen hamlenin ne işe yaradığını sırayla sorarak söylenir:
- 1) Üçlüyü tamamlıyor mu? → **kazanır**. 2) Rakibin tek hamlelik tehdidini kapatıyor mu? → **engeller**.
- 3) Aynı anda iki tehdit kuruyor mu? → **çatal kurar**. 4) Yoksa değerin işareti: kazanmayı zorlar / berabere tutar / kaybı geciktirir.

Tek tek kural yazmak yerine

"Önce kazan, sonra engelle, sonra çatal kur" gibi kurallarla da iyi bir XOX oyuncusu yazılabilir. Minimaks bu kuralların hiçbirini bilmez; hepsini ağacı arayarak kendisi bulur. Aynı program Dörtlü Bağla'da da çalışır, kurallar çalışmaz.

XOX'TA MİNİMAKS

Orta Oyun Konumunun Tam Ağacı



Başlangıç: X'in sırası, üç boş kare. Her düğüm bir konum; turuncu kare o konuma götüren hamle. Değerler oyun sonlarından köke yükselir.

Adım **0 / 22** · Değerlendirilen yaprak: **0 / 6**

Geri | Adım | Oynat | Sıfırla

Boş Tahtanın Değeri: Beraberlik

- Boş tahtadan tam minimaks araması **549.946** düğüm ziyaret eder; bunların **255.168**'i yaprak, yani farklı oyunlardır.
- X'in dokuz ilk hamlesinin hepsinin değeri **0**: hiçbir açılış kazancı zorlamaz, iki taraf da hatasız oynarsa oyun **berabere** biter.
- Değerler eşit olduğundan seçimi sıralama belirler; laboratuvardaki YZ eşitlikte merkez, köşeler, kenarlar sırasını kullanır.
- O'nun işi daha zordur: X köşeyle açarsa O'nun tek berabere tutan cevabı merkezdir; diğer yedi cevabın her biri kaybettirir.
- Aynı konumlar tekrar tekrar aranır: farklı konum sayısı yalnız **5.478**'dir.

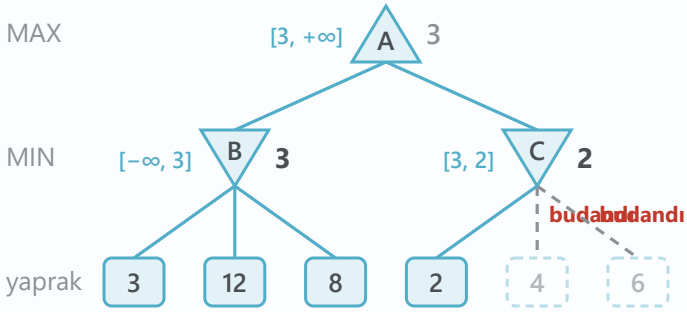
0	0	0
0	0	0
0	0	0

her ilk hamlenin değeri 0

ALFA-BETA BUDAMASI

Bazı Dalları Okumaya Gerek Yok

- A (MAX) ilk çocuğu B'den **3**'ü garanti etti: B = min(3, 12, 8) = 3.
- C'nin ilk yaprağı **2**. C bir MIN düğümü olduğundan değeri **en fazla 2** olur.
- A, 3 varken en fazla 2 veren C'yi asla seçmez: C'nin kalan yaprakları (4 ve 6) ne olursa olsun kökün değeri değişmez.
- Bu iki yaprak hiç okunmaz: **budanır**. Satrançta bir "yaprak" derin bir alt ağaçtır; budama milyonlarca düğümü kurtarır.



Alfa-beta budaması minimaksla **aynı kök değerini ve aynı hamleyi** bulur, ama sonucu etkileyemeyecek dalları atlar.

ALFA-BETA BUDAMASI

α ve β

α (alfa): MAX'ın alt sınırı

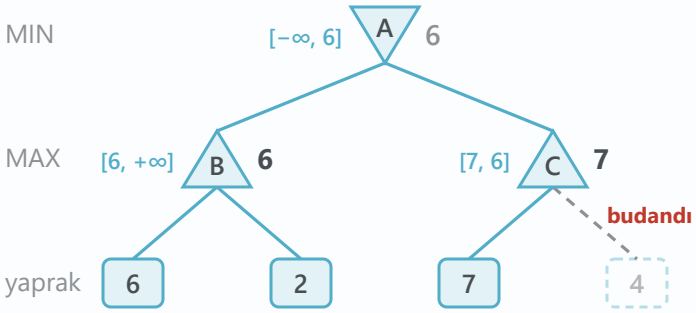
Kökten bu düğüme giden yol üzerinde MAX'ın *şimdiden garanti ettiği* en iyi değer. Başta −∞. Yalnız MAX düğümlerinde büyür: α = max(α, değer).

β (beta): MIN'in üst sınırı

Aynı yol üzerinde MIN'in *şimdiden garanti ettiği* en iyi (en küçük) değer. Başta +∞. Yalnız MIN düğümlerinde küçülür: β = min(β, değer).

- Her düğüm [α, β] aralığını **ebeveyninden alır** ve çocuklarına **aynen aktarır**; ebeveyne geri dönen yalnız değerdir.
- Bir düğümün değeri bu aralığın dışına düşerse sonucu etkilemez: α'dan küçükse MAX onu seçmez, β'dan büyükse MIN onu seçmez.
- Aralık boşalırsa (**α ≥ β**) düğümün kalan çocuklarına bakmaya gerek kalmaz.

Budama Koşulu: α ≥ β



Kök MIN, B'den 6'yı garanti etti (β = 6). C bir MAX düğümü; ilk yaprağı 7 → α = 7 ≥ β = 6. C en az 7 verir, MIN C'yi seçmez: yaprak 4 budanır.

	MIN düğümünde	**MAX düğümünde**
Güncellenen	β = min(β, değer)	α = max(α, değer)
Budama	β ≤ α olunca	α ≥ β olunca
Anlamı	bu düğüm en fazla β verir; yukarıdaki MAX α'yı zaten garanti ediyor	bu düğüm en az α verir; yukarıdaki MIN β'yı zaten garanti ediyor
Döndürülen	bir üst sınır (gerçek değer ≤ β)	bir alt sınır (gerçek değer ≥ α)

İki durum aynı koşuldur: α ≥ β. Bazı kitaplar MIN düğümündekine α kesmesi, MAX düğümündekine β kesmesi der.

Budama Sonucu Neden Değiştirmez?

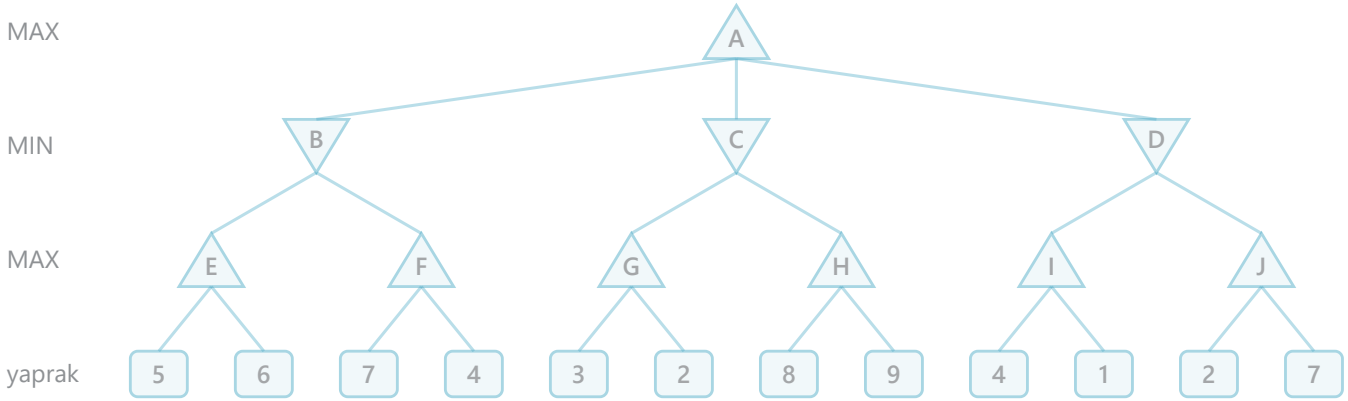
- Alfa-beta, [α, β] aralığıyla çağrıldığında şunu sağlar (v gerçek minimaks değeri, r döndürülen değer):

$\alpha < v < \beta$ ise	$r = v$ (kesin değer)
$v \leq \alpha$ ise	$r \leq \alpha$ (yalnız "yeterince kötü" olduğu bilinir)
$v \geq \beta$ ise	$r \geq \beta$ (yalnız "yeterince iyi" olduğu bilinir)

- Kök $(-\infty, +\infty)$ aralığıyla çağrılır; v bu aralığın her zaman içindedir, bu yüzden kökün değeri **kesindir**.
- Budanan bir dalın gerçek değeri ne olursa olsun, ebeveynin max ya da min işleminin sonucunu değiştiremez: MAX için $\leq \alpha$ olan bir değer α'yı geçemez, MIN için $\geq \beta$ olan bir değer β'nın altına inemez.
- İç düğümlerde döndürülen sayı bir sınır olabilir; kökte seçilen hamle yine minimaksın seçtiği hamledir.

ALFA-BETA BUDAMASI

Alfa-Beta Adım Adım



Başlangıç: kök A'da α = −∞, β = +∞. Adım düğmesiyle derinlik öncelikli arama başlar; budanan dallar kesik çizgiyle gösterilir.

Adım **0 / 26** · Değerlendirilen yaprak: **0 / 12**

Geri | **Adım** | **Oynat** | **Sıfırla**

Düğümün solunda [α, β] aralığı, sağında değeri · kesik çizgi ve soluk düğüm: budanan dal · sayaç: değerlendirilen yaprak.

ALFA-BETA BUDAMASI

Minimaks ve Alfa-Beta: Aynı Ağaç

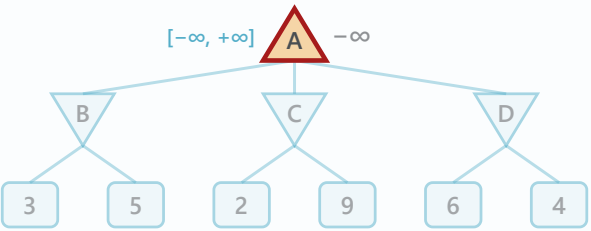
Ağaç	Minimaks	Alfa-beta	Kök değeri	Budanan ya da sıra
Adımlayıcıdaki ağaç (12 yaprak)	12 yaprak	7 yaprak	6 ve 6	F'nin 4'ü; H ve J'nin tamamı
Kod izlemedeki ağaç (6 yaprak)	6 yaprak	5 yaprak	4 ve 4	C'nin 9'u
Elle çözülecek ağaç (8 yaprak)	8 yaprak	5 yaprak	7 ve 7	E'nin 1'i; G'nin tamamı
XOX, boş tahta (tam arama)	549.946 düğüm	7.865 düğüm	0 ve 0	merkez, köşeler, kenarlar sırasıyla
Dörtlü Bağla, başlangıç, derinlik 7	960.793 düğüm	5.086 düğüm	+8 ve +8	ortadan dışa sırayla

Her satırda kök değeri ve seçilen hamle aynı; değişen yalnız okunan yaprak ya da ziyaret edilen düğüm sayısı. Kazanç ağaç büyüdükçe artar: küçük ağaçta birkaç yaprak, XOX'ta 70 kat, Dörtlü Bağla'da 189 kat.

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
14     if a >= b:
15       break                     # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



İlk çağrı: kök A bir MAX düğümü, deger = -inf, a = -inf, b = inf.

Adım 1 / 14

DEĞİŞKENLER

`n = "A"` `a = -inf` `b = inf` `maks = True` `deger = -inf`

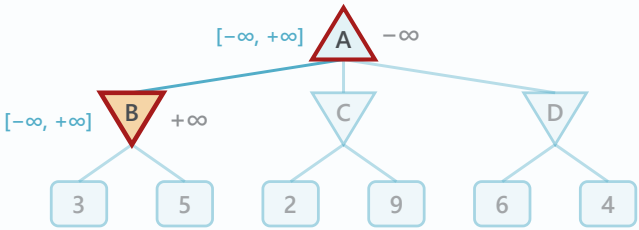
ÇAĞRI YIĞINI

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
▶    deger = -inf if maks else inf
8    for c in T[n]:
9      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
14     if a >= b:
15       break                     # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta("B") çağrılır: MIN, deger = inf; a ve b üst çağrıdan gelir: [-inf, inf].

Adım 2 / 14

DEĞİŞKENLER

`n = "B"` `a = -inf` `b = inf` `maks = False` `deger = inf`

ÇAĞRI YIĞINI

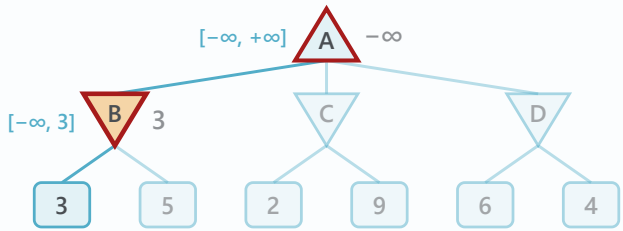
`alfabeta("B", -inf, inf, False)`

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
▶        deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                    # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta(3) bir yaprak: 3 döner. min ile deger = 3, b = 3; a < b, devam.

Adım 3 / 14

DEĞİŞKENLER

`n = "B"` `a = -inf` `b = 3` `maks = False` `c = 3` `v = 3` `deger = 3`

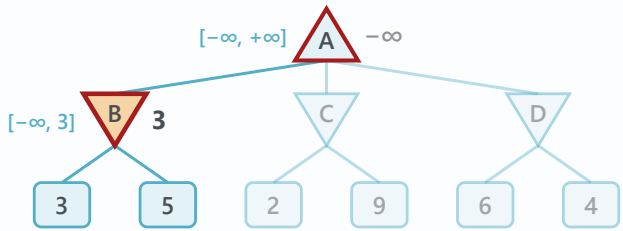
ÇAĞRI YIĞINI

```
alfabeta("B", -inf, inf, False)
alfabeta("A", -inf, inf, True)
```

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
▶        deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                    # budama
▶    return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta(5) bir yaprak: 5 döner. min ile deger = 3, b = 3; a < b, devam; B bitti, 3 döner.

Adım 4 / 14

DEĞİŞKENLER

`n = "B"` `a = -inf` `b = 3` `maks = False` `c = 5` `v = 5` `deger = 3`

ÇAĞRI YIĞINI

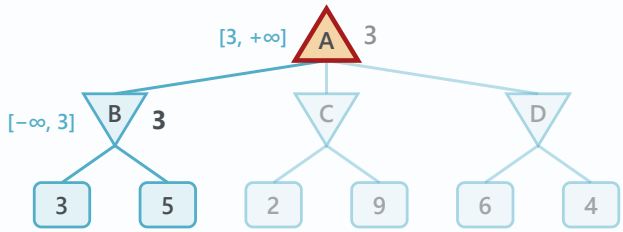
`alfabeta("B", -inf, inf, False)`

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
▶        deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                    # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



B çağrısı 3 döndürdü. max ile deger = 3, a = 3; a < b, devam.

Adım 5 / 14

DEĞİŞKENLER

`n = "A"` `a = 3` `b = inf` `maks = True`
`c = "B"` `v = 3` `deger = 3`

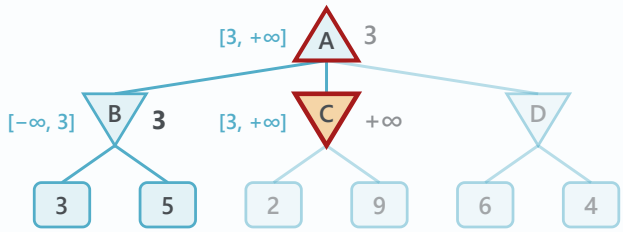
ÇAĞRI YIĞINI

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
▶    deger = -inf if maks else inf
8    for c in T[n]:
9      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
14     if a >= b:
15       break                     # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta("C") çağrılır: MIN, deger = inf; a ve b üst çağrıdan gelir: [3, inf].

Adım 6 / 14

DEĞİŞKENLER

`n = "C"` `a = 3` `b = inf` `maks = False` `deger = inf`

ÇAĞRI YIĞINI

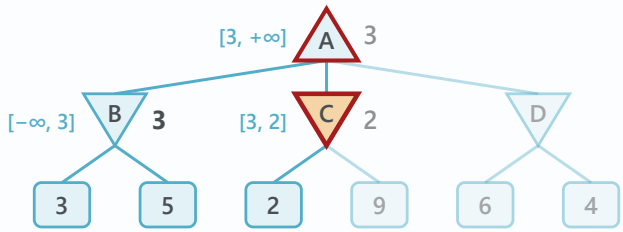
`alfabeta("C", 3, inf, False)`

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
▶        deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                    # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta(2) bir yaprak: 2 döner. min ile deger = 2, b = 2; a ≥ b.

Adım 7 / 14

DEĞİŞKENLER

`n = "C"` `a = 3` `b = 2` `maks = False` `c = 2` `v = 2` `deger = 2`

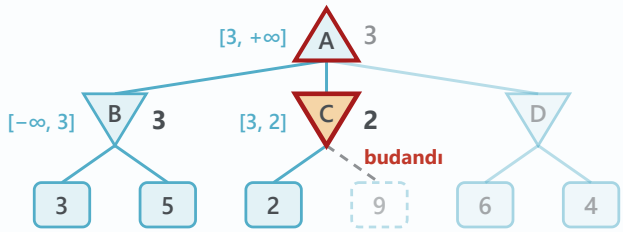
ÇAĞRI YIĞINI

```
alfabeta("C", 3, inf, False)
alfabeta("A", -inf, inf, True)
```

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
▶        break                   # budama
▶    return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



a = 3 ≥ b = 2: break. C en fazla 2 verir, A zaten 3 buldu; yaprak 9 hiç okunmaz.

Adım 8 / 14

DEĞİŞKENLER

`n = "C"` `a = 3` `b = 2` `maks = False` `c = 9` `deger = 2`

ÇAĞRI YIĞINI

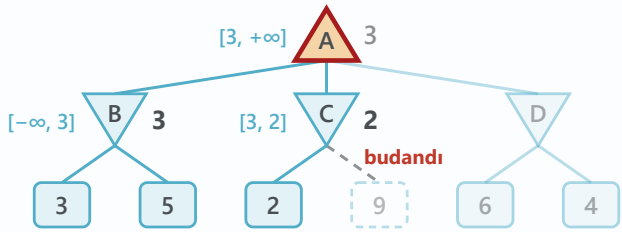
`alfabeta("C", 3, inf, False)`

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
▶        deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                    # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



C çağrısı 2 döndürdü. max ile deger = 3, a = 3; a < b, devam.

Adım 9 / 14

DEĞİŞKENLER

`n = "A"` `a = 3` `b = inf` `maks = True`
`c = "C"` `v = 2` `deger = 3`

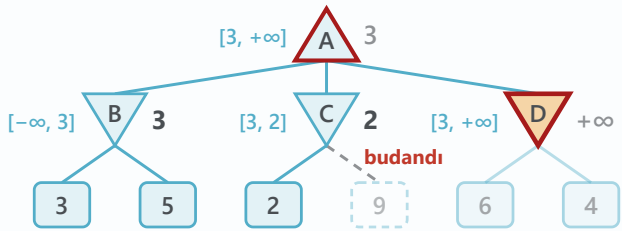
ÇAĞRI YIĞINI

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
▶7   deger = -inf if maks else inf
8    for c in T[n]:
9      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
14     if a >= b:
15       break                     # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta("D") çağrılır: MIN, deger = inf; a ve b üst çağrıdan gelir: [3, inf].

Adım 10 / 14

DEĞİŞKENLER

`n = "D"` `a = 3` `b = inf` `maks = False` `deger = inf`

ÇAĞRI YIĞINI

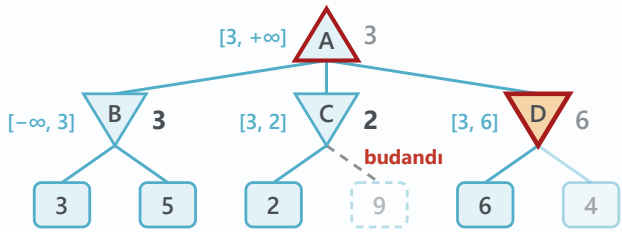
`alfabeta("D", 3, inf, False)`

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
▶        deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                    # budama
16   return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta(6) bir yaprak: 6 döner. min ile deger = 6, b = 6; a < b, devam.

Adım 11 / 14

DEĞİŞKENLER

`n = "D"` `a = 3` `b = 6` `maks = False` `c = 6` `v = 6` `deger = 6`

ÇAĞRI YIĞINI

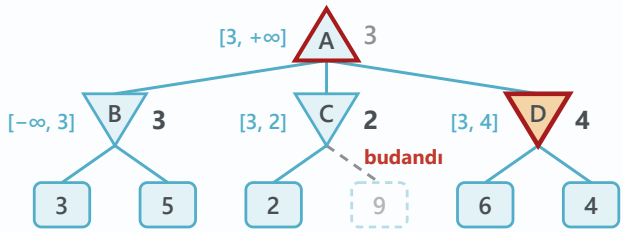
`alfabeta("D", 3, inf, False)`

`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
▶        deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                     # budama
▶    return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



alfabeta(4) bir yaprak: 4 döner. min ile deger = 4, b = 4; a < b, devam; D bitti, 4 döner.

Adım 12 / 14

DEĞİŞKENLER

`n = "D"` `a = 3` `b = 4` `maks = False` `c = 4` `v = 4` `deger = 4`

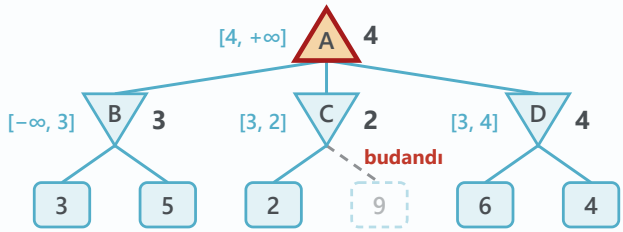
ÇAĞRI YIĞINI

```
alfabeta("D", 3, inf, False)
alfabeta("A", -inf, inf, True)
```

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
▶      v = alfabeta(c, a, b, not maks)
10     if maks:
▶        deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
▶      if a >= b:
15       break                     # budama
▶    return deger
17 print(alfabeta("A", -inf, inf, True))
```

ÇIKTI



D çağrısı 4 döndürdü. max ile deger = 4, a = 4; a < b, devam; A bitti, 4 döner.

Adım 13 / 14

DEĞİŞKENLER

`n = "A"` `a = 4` `b = inf` `maks = True` `c = "D"` `v = 4` `deger = 4`

ÇAĞRI YIĞINI

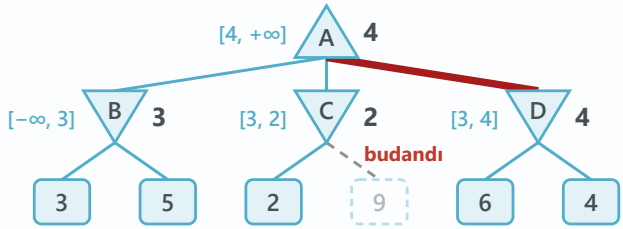
`alfabeta("A", -inf, inf, True)`

Kod İzleme: Alfa-Beta Fonksiyonu

```
1  from math import inf
2  T = {"A": ["B", "C", "D"], "B": [3, 5],
3       "C": [2, 9], "D": [6, 4]}
4  def alfabeta(n, a, b, maks):
5    if isinstance(n, int):        # yaprak
6      return n
7    deger = -inf if maks else inf
8    for c in T[n]:
9      v = alfabeta(c, a, b, not maks)
10     if maks:
11       deger = max(deger, v); a = max(a, deger)
12     else:
13       deger = min(deger, v); b = min(b, deger)
14     if a >= b:
15       break                     # budama
16   return deger
▶ print(alfabeta("A", -inf, inf, True))
```

ÇIKTI

```
4
```



Kökün değeri 4: A, D'ye giden hamleyi seçer. 6 yaprağın 5'i okundu, yaprak 9 budandı.

Adım 14 / 14

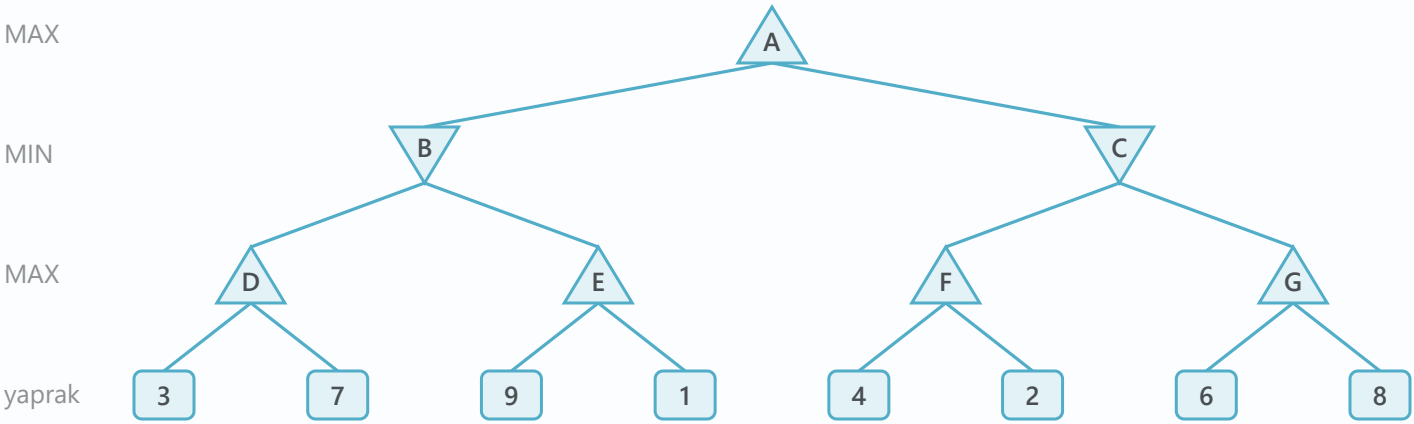
DEĞİŞKENLER

`sonuç = 4`

ÇAĞRI YIĞINI

boş

Elle Alfa-Beta: Soru



1. Kökün değeri

Minimaksla kökün değeri kaçtır? MAX hangi hamleyi seçer?

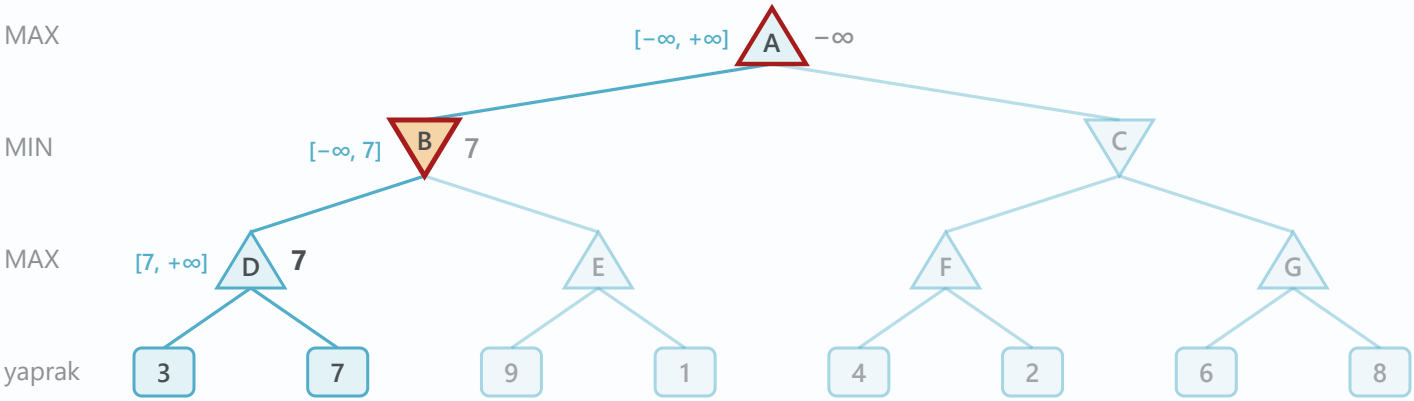
2. Aralıklar

Çocuklar soldan sağa aranırken her iç düğüme hangi [α, β] aralığı gelir?

3. Budananlar

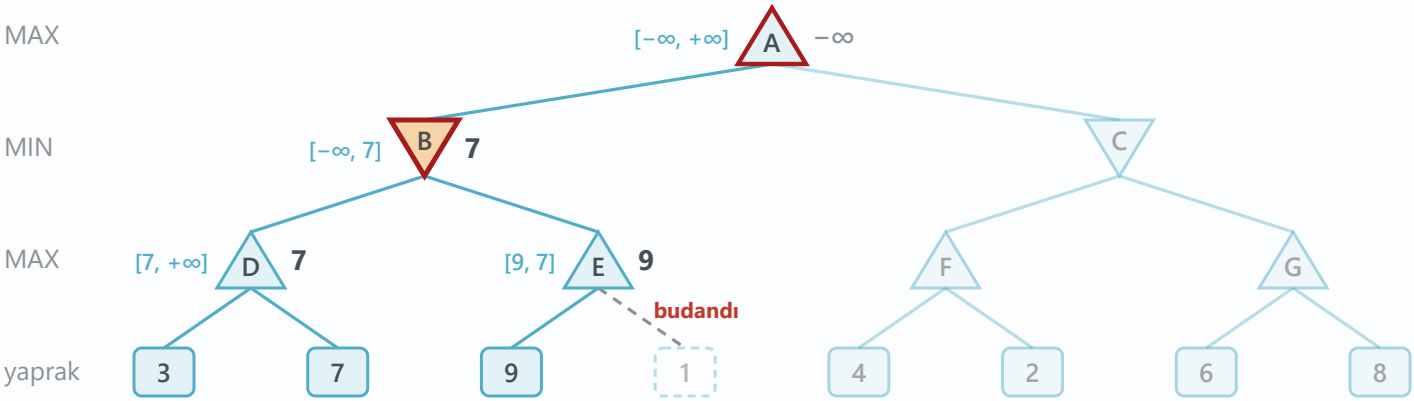
Hangi yapraklar hiç okunmaz? Kaç yaprak değerlendirilir?

Elle Alfa-Beta: Çözüm



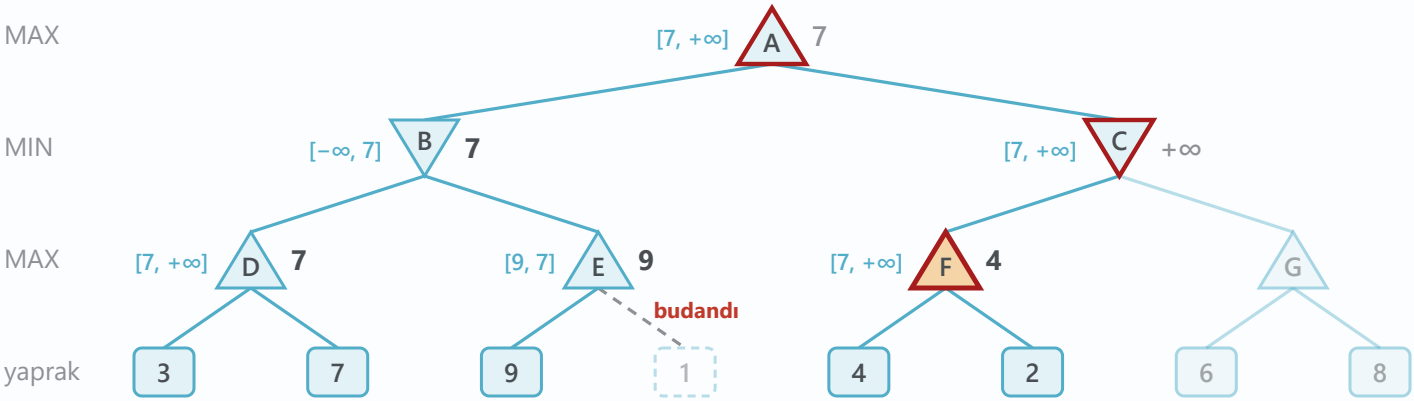
B'ye [−∞, +∞] gelir. D (MAX) 3'ü ve 7'yi okur: D = 7. B (MIN) β = 7 olur.

Elle Alfa-Beta: Çözüm



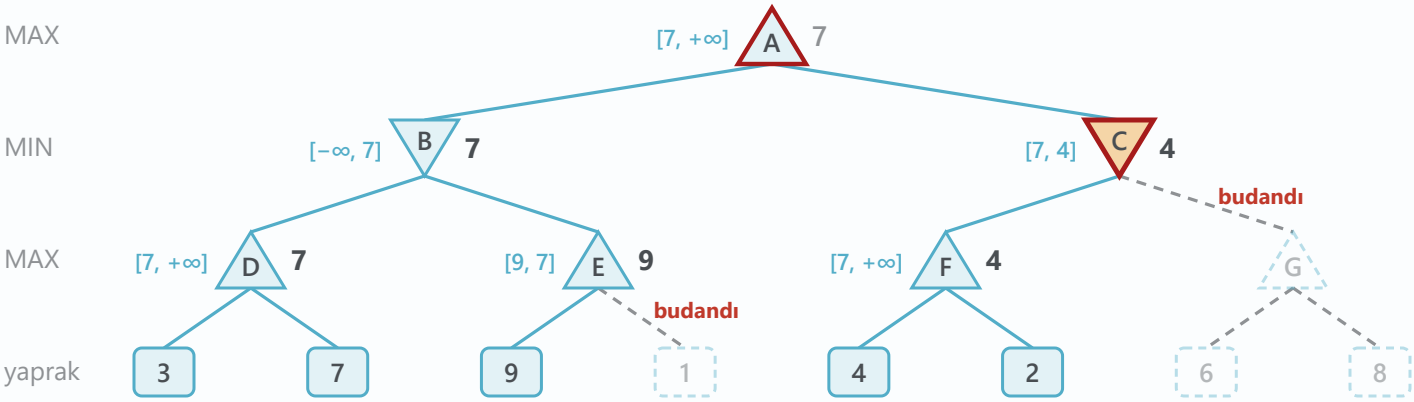
E'ye [−∞, 7] gelir. İlk yaprak 9 → α = 9 ≥ β = 7: E en az 9 verir, B onu seçmez; yaprak 1 budanır. B = min(7, 9) = 7.

Elle Alfa-Beta: Çözüm



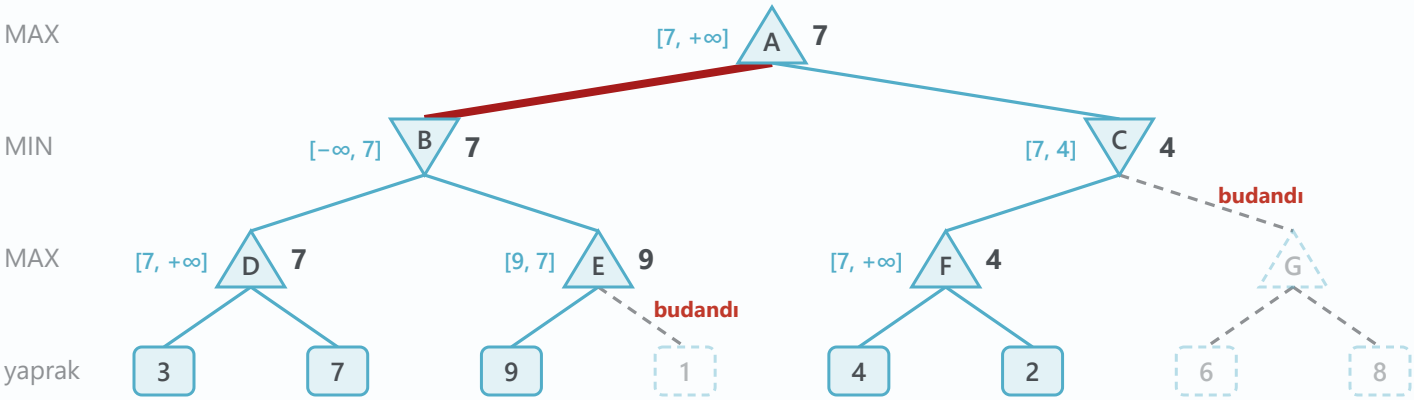
Kökte α = 7. C'ye [7, +∞] gelir, F'ye de aynı aralık. F (MAX) 4'ü ve 2'yi okur: F = 4 (α 7 olarak kalır).

Elle Alfa-Beta: Çözüm



C'de β = min(+∞, 4) = 4 ≤ α = 7: C en fazla 4 verir, kök onu seçmez. G'nin tamamı (6 ve 8) budanır.

Elle Alfa-Beta: Çözüm



Kök = max(7, 4) = 7: MAX, B'yi seçer. 8 yapraktan 5'i okundu, 3'ü budandı. Minimaksla aynı sonuç.

Hamle Sıralaması

- Alfa-betanın ne kadar budayacağı çocukların **hangi sırayla** denendiğine bağlıdır; kök değeri sıraya bağlı değildir.
- **En iyi sıra** (MAX'ta en büyük, MIN'de en küçük çocuk önce): değerlendirilen yaprak $b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1 \approx b^{d/2}$ (Knuth ve Moore, 1975).
- **En kötü sıra**: hiç budama olmaz, b^d yaprağın hepsi okunur.
- Sonuç: aynı sürede yaklaşık **iki kat derinlik**; etkin dallanma etkeni b'den √b'ye iner.
- İyi sıra nereden gelir? Oyun bilgisi (Dörtlü Bağla'da orta sütun, XOX'ta merkez ve köşeler, satrançta önce taş alan hamleler) ve bir önceki aramanın en iyi hamlesi.

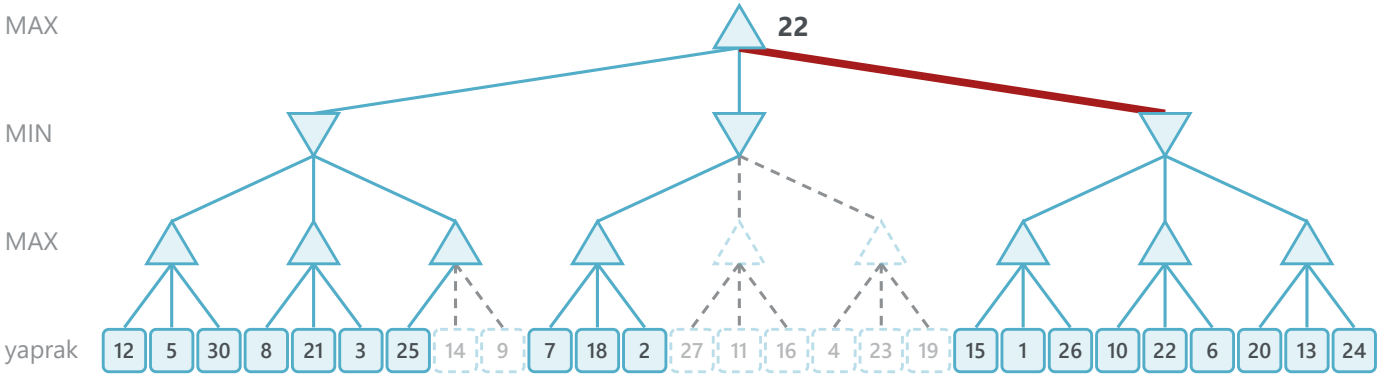
d	35^d	En iyi sıra
2	1.225	69
4	1.500.625	2.449
6	1.838.265.625	85.749
8	$\approx 2{,}25 \cdot 10^{12}$	3.001.249

b = 35 (satranç ortalaması) için okunan yaprak sayısı.

HAMLE SIRALAMASI

Sıralama Laboratuvarı

Özgün sıra | Karıştır | En iyi sıra | En kötü sıra | Yeni değerler



Özgün sıra: değerlendirilen yaprak **19 / 27**, kökün değeri **22**. Kuram: en iyi sırada $3^2 + 3^1 - 1 = 11$, sırasız minimaksta 27 yaprak.

Derinlik 3, dallanma 3: 27 yaprak. Düğme yalnız çocukların sırasını değiştirir (ağaç aynı ağaçtır); alfa-beta her seferinde baştan çalışır. Soluk yapraklar ve kesik kenarlar budanan dallardır. En iyi sırada 11 yaprak okunur, en kötü sırada budama hiç olmaz.

XOX'ta Alfa-Beta: Düğüm Sayıları

Boş tahtadan tam arama	Ziyaret edilen düğüm	Süre	Kök değeri
Minimaks (sıra fark etmez)	549.946	73 ms	0
Alfa-beta, kareler 0–8 sırasıyla (soldan sağa, yukarıdan aşağı)	20.866	3,8 ms	0
Alfa-beta, önce merkez, sonra köşeler, sonra kenarlar	7.865	1,3 ms	0
Geçiş tablolu minimaks (her konum bir kez)	5.478	12 ms	0

Yeniden hesapla

- Sayılar sayfa açılınca bu tarayıcıda hesaplanır; derinlik cezalı ölçek kullanılır. ±1 ölçeğiyle alfa-beta 18.297 ve 7.275 düğüm ziyaret eder: budama değer ölçeğine de bağlıdır.
- İyi sıralama düğüm sayısını 2,7 kat azaltır; minimaksa göre kazanç yaklaşık 70 kattır.
- Geçiş tablosu aynı konumu ikinci kez aramaz; alfa-betayla birlikte kullanıldığında tabloya değerin kesin mi, sınır mı olduğu da yazılır.

Dörtlü Bağla: Tam Arama Neden Olmaz?

- 7 sütun, 6 satır; taş seçilen sütunda en alttaki boş yere düşer. Yatay, dikey ya da çapraz dört taşını ilk dizen kazanır.
- Dallanma etkeni en çok 7, oyun en çok 42 yarım hamle. Yasal konum sayısı **4.531.985.219.092** ≈ $4{,}5 \cdot 10^{12}$ (John Tromp).
- Oyun 1988'de birbirinden bağımsız olarak **James D. Allen** ve **Victor Allis** tarafından çözüldü: ilk oyuncu ortadaki sütundan başlarsa kazanmayı garanti eder.
- Bu çözüm kaba kuvvetle değil, oyuna özgü kurallar ve çok güçlü aramalarla bulundu. Bir tarayıcıda, bir saniyede böyle bir çözüm yapılamaz.

Ne yapılır?

Arama belli bir **derinlikte durdurulur**. Oradaki konumlar oyun sonu olmadığından faydaları bilinmez; yerine bir **değerlendirme fonksiyonu** konumun ne kadar iyi olduğunu tahmin eder.

Başlangıç konumunda derinlik 7 bile minimaksla 960.793 düğüm demektir.

Derinlik Sınırlı Arama

```
H(s, d) =
  fayda(s)            s oyun sonuysa
  E(s)                d = 0 ise
  max  H(sonuç(s, a), d − 1)   MAX
   a
  min  H(sonuç(s, a), d − 1)   MIN
   a
```

Laboratuvardaki fayda: Kırmızı (MAX) k. yarım hamlede kazanırsa +(1000 − k), Mavi kazanırsa −(1000 − k), tahta dolarsa 0. 1000 yakınındaki sayılar sütun düğmelerinde “kazanç” ya da “kayıp” diye gösterilir.

- **Kesme testi**: oyun bittiyse ya da derinlik sınırına gelindiyse inmeyi bırak.
- **E(s)** kesin değer değil, tahmindir: konum MAX için iyi görünüyorsa büyük, kötü görünüyorsa küçük.
- Oyun sonu E'nin her değerinden büyük ya da küçük olmalı: kazanç ±1000 civarı, E en fazla birkaç yüz.
- Derinlik cezası burada da var: hızlı kazanç ve geç kayıp tercih edilir.
- Alfa-beta aynen çalışır; sınırda E değerleri yaprak gibi kullanılır.

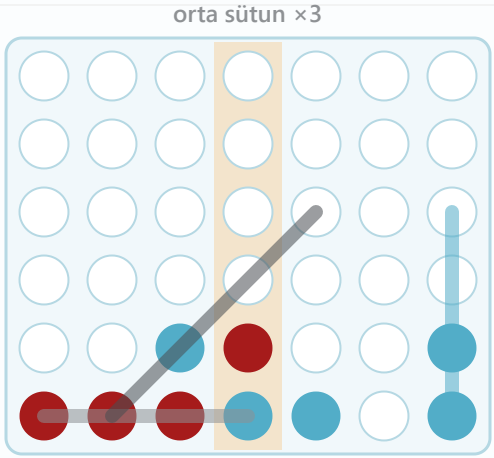
Değerlendirme Fonksiyonu

Tahtada yan yana dört kareden oluşan **69 pencere** var: 24 yatay, 21 dikey, 24 çapraz. Her pencere bir puan alır:

Penceredeki taşlar	p(w)
3 kırmızı + 1 boş	+5
2 kırmızı + 2 boş	+2
3 mavi + 1 boş	−5
2 mavi + 2 boş	−2
iki renk birden, tek taş ya da boş	0

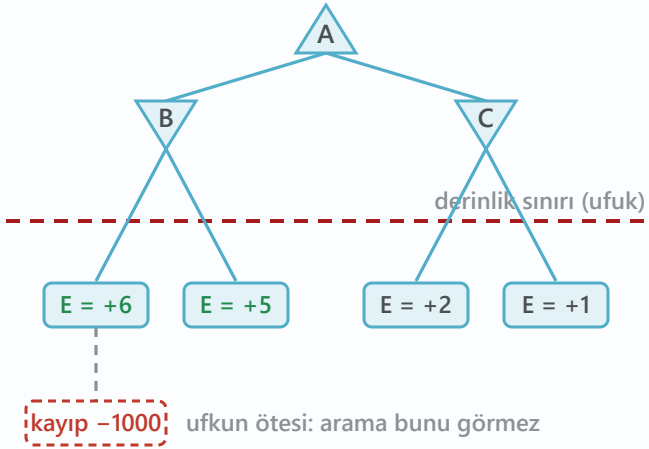
$$E(s) = \Sigma\, p(w) + 3 \cdot (K_{orta} - M_{orta})$$

K_{orta}, M_{orta}: orta sütundaki kırmızı ve mavi taş sayısı. Orta sütun en çok pencereye girer.



Ufuk Etkisi

- Arama derinlik sınırında durduğu için sınırın hemen ötesindeki olayları **göremez**: buna **ufuk etkisi** denir.
- Şekilde B'nin altındaki konum E = +6 ile en iyi görünür; oysa bir hamle sonra kaçınılmaz bir kayıp vardır.
- Daha sinsi hâli: kaçınılmaz bir kaybı ufkun ötesine iten anlamsız hamleler (satrançta boş yere şah çekmek). Kayıp ortadan kalkmaz, yalnız görünmez olur.
- Çareler: derinliği artırmak, hareketli konumlarda aramayı sürdürmek (**sakinlik araması**), tehdit varken bir katman daha inmek.
- Dörtlü Bağla'da derinlik 1 ya da 2 ile oynayan YZ iki yönlü tehditleri göremez ve yenilebilir.



Laboratuvardaki Arama

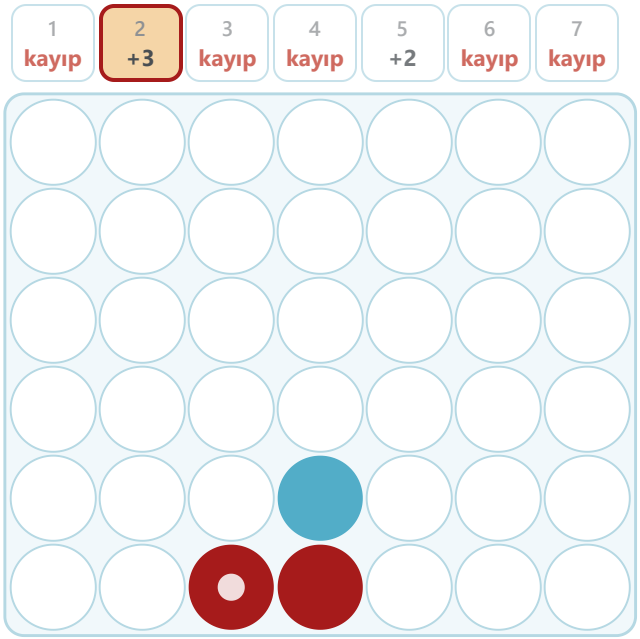
```
from math import inf
def alfabeta(s, d, a, b, maks):
  if s.bitti():                # oyun sonu
    return s.fayda()           # ±(1000 − k) ya da 0
  if d == 0:
    return E(s)                # değerlendirme
  deger = -inf if maks else inf
  for m in sirala(s.hamleler()):
    v = alfabeta(s.oyna(m), d-1, a, b, not maks)
    if maks:
      deger = max(deger, v); a = max(a, deger)
    else:
      deger = min(deger, v); b = min(b, deger)
    if a >= b:
      break
  return deger
```

```
def en_iyi_hamle(s, d):      # kök: MAX
  a, en = -inf, None
  for m in sirala(s.hamleler()):
    v = alfabeta(s.oyna(m), d-1,
                 a, inf, False)
    if v > a:
      a, en = v, m
  return en
```

- Oyundan bağımsızdır: yalnız `bitti`, `fayda`, `hamleler`, `oyna` ve `E` oyuna özgüdür.
- Kökte α kökün çocukları arasında taşınır: ilk çocuktan sonra kalanlar daralan pencereyle aranır.
- Laboratuvar her sütunun puanını göstermek için ayrıca tam minimaks da çalıştırır ve iki sonucu karşılaştırır.

DÖRTLÜ BAĞLA

Dörtlü Bağla Laboratuvarı



Yeni oyun: insan başlar | YZ başlar | YZ oynasın

– | Derinlik 5 | + | Sıra: ortadan dışa

Örnek konum: Kırmızı 4. ve 3. sütunlara, Mavi 4. sütuna oynadı. YZ'nin seçimi: **sütun 2** (+3). Bu sütunda derinlik sınırı içinde zorunlu bir kazanç ya da kayıp yok; ufuktaki konumların değerlendirme puanı (açık ikili ve üçlüler, orta sütun) en iyi olan sütun seçildi. 5 sütun derinlik sınırı içinde kayba götürüyor.

Minimaks: **18.347** düğüm, 9,2 ms · Alfa-beta: **1.391** düğüm, 0,6 ms · aynı değer, aynı sütun ✓

Sütun düğmesine tıklanınca insan oyuncunun taşı düşer. Düğmelerdeki sayılar YZ'nin gözünden o sütunun minimaks puanı; turuncu sütun seçimdir. Derinlik 1–7 arası. Dörtlü Bağla oyunu →

Ortadan Dışa Sıralamanın Etkisi

Derinlik	Minimaks	Alfa-beta (soldan sağa)	Alfa-beta (ortadan dışa)	Kök değeri
1	8	8	8	+3
2	57	33	21	0
3	400	200	76	+6
4	2.801	966	175	0
5	19.608	4.340	627	+6
6	137.257	16.409	1.285	0
7	960.793	67.549	5.086	+8

- Boş tahtadan aramada ziyaret edilen düğüm sayısı; sayılar sayfa açılınca bu tarayıcıda yeniden hesaplanır.
- Minimaks yaklaşık 7^d ile büyür. Alfa-beta derinlik 7'de soldan sağa sırayla 14 kat, **ortadan dışa** (4, 3, 5, 2, 6, 1, 7) sırayla 189 kat az düğüm ziyaret eder.
- Kök değeri üç yöntemde de aynı: sıra yalnız maliyeti değiştirir. Orta sütun en çok dörtlüye girdiği için iyi hamleler çoğunlukla ortadadır; önce onlar denenince α ve β erken daralır.

DÖRTLÜ BAĞLA

Yinelemeli Derinleştirme ve Geçiş Tablosu

Yinelemeli derinleştirme

- Önce derinlik 1, sonra 2, 3, … süre bitene kadar arama yapılır; süre bitince son tamamlanan aramanın hamlesi oynanır.
- Tekrarlanan iş azdır: son katman diğerlerinin toplamından büyüktür (b = 7 için toplam iş yaklaşık 7/6 katı).
- Bir önceki aramanın en iyi hamlesi önce denenir: hamle sıralaması kendiliğinden iyileşir.

Geçiş tablosu

- Aynı konuma farklı hamle sıralarıyla ulaşılır (transpozisyon). Konum bir kez aranır, sonucu hash tablosuna yazılır.
- Kayıt: değer, arandığı derinlik, değerin kesin mi alt/üst sınır mı olduğu, en iyi hamle.
- XOX'ta 549.946 düğüm yerine 5.478 konum. Satranç programları konumu Zobrist (1970) karmasıyla anahtara çevirir.

Güçlü oyun programlarının iskeleti: yinelemeli derinleştirme + alfa-beta + iyi hamle sıralaması + geçiş tablosu + iyi bir değerlendirme fonksiyonu.

KAPANIŞ

Sık Yapılan Hatalar

α ve β'yı yanlış yöne aktarmak

Aralık ebeveynden çocuğa aynen iner; ebeveyne yalnız değer döner. Çocukta güncellenen α ya da β ebeveyne taşınmaz. MAX yalnız α'yı, MIN yalnız β'yı günceller.

Budanan dalı hesaba katmak

Budanan yaprakların değeri okunmaz; elle çözümde onları min ya da max işlemine sokmak yanlıştır. Kesilen düğümün döndürdüğü sayı da kesin değer değil, sınırdır.

Bakış açısı ve işaret hatası

E(s) hep aynı oyuncunun gözünden yazılmalı (burada Kırmızı/MAX). YZ ikinci oyuncuysa en küçüğü seçer; negamaks yazılırsa her katmanda işaret çevrilir.

Derinlik cezasını unutmak

Bütün kazançlar aynı değeri alırsa YZ hazır kazancı erteler, kaybederken de en çabuk kaybedeni seçebilir. Fayda, kazancın kaç hamle sonra geldiğini de içermeli.

KAPANIŞ

Gerçek Dünyada Oyun Ağacı Araması

Deep Blue (1997)

IBM'in bilgisayarı, dünya şampiyonu Kasparov'u altı partilik maçta 3½–2½ yendi. Özel satranç yongalarında paralel alfa-beta, saniyede yaklaşık 200 milyon konum.

Satranç motorları

Stockfish gibi programlar hâlâ alfa-beta türevi arama kullanır: yinelemeli derinleştirme, geçiş tablosu, güçlü budama sezgileri; 2020'den beri sinir ağıyla değerlendirme (NNUE).

Go ve MCTS

Go'da b ≈ 250 ve iyi bir E(s) yazmak çok zor. Monte Carlo ağaç araması (Coulom, 2006; UCT: Kocsis ve Szepesvári, 2006) rastgele oyunlarla değerlendirir. AlphaGo, MCTS'yi derin sinir ağlarıyla birleştirdi ve Mart 2016'da Lee Sedol'u 4–1 yendi.

Ortak fikir değişmedi: rakibin en iyi cevabını varsayarak ileriye bakmak. Değişen, ağacın ne kadarının arandığı ve yaprakların nasıl değerlendirildiğidir.

KAPANIŞ

Özet

Oyun ağacı

Düğüm konum, kenar hamle; MAX ▲ büyütür, MIN ▽ küçültür. Boyut ≈ b^d.

Minimaks

Değerler yapraktan köke: MAX katmanında max, MIN katmanında min. Hatasız rakibe karşı en iyi hamle; $O(b^d)$ zaman.

Nim ve XOX

Nim'de minimaks = P/N = XOR kuralı. XOX'ta değer 0; derinlik cezası hızlı kazanç ve geç kayıp seçtirir.

Alfa-beta

[α, β] aralığı yukarıdan iner; α ≥ β olunca kalan çocuklar budanır. Kök değeri ve hamle değişmez.

Hamle sıralaması

En iyi sırada ≈ $b^{d/2}$ yaprak, en kötü sırada b^d. Ortadan dışa sıra Dörtlü Bağla'da 189 kat kazandırır.

Derinlik sınırı

Değerlendirme fonksiyonu tahmin eder; ufuk etkisi, yinelemeli derinleştirme ve geçiş tablosu.

KAPANIŞ

Kaynaklar

- S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, 4. baskı, Pearson, 2020 (Bölüm 5: Adversarial Search and Games).
- D. E. Knuth, R. W. Moore, "An analysis of alpha-beta pruning", *Artificial Intelligence* 6(4), 293–326, 1975.
- C. E. Shannon, "Programming a computer for playing chess", *Philosophical Magazine* 41(314), 1950.
- J. von Neumann, "Zur Theorie der Gesellschaftsspiele", *Mathematische Annalen* 100, 1928 · H. W. Kuhn, "Extensive games and the problem of information", *Contributions to the Theory of Games* II, 1953.
- C. L. Bouton, "Nim, a game with a complete mathematical theory", *Annals of Mathematics* 3, 1901–1902.
- L. V. Allis, *A Knowledge-Based Approach of Connect-Four*, yüksek lisans tezi, Vrije Universiteit Amsterdam, 1988 · J. Tromp, John's Connect Four Playground (konum sayısı).
- J. Tromp, G. Farnebäck, "Combinatorics of Go", *Computers and Games*, 2006.
- J. Tromp, "The Number of Legal Go Positions", *Computers and Games*, 2016.
- A. L. Zobrist, "A new hashing method with application for game playing", Tech. Rep. 88, University of Wisconsin, 1970.
- M. Campbell, A. J. Hoane, F. Hsu, "Deep Blue", *Artificial Intelligence* 134, 57–83, 2002.
- R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search", *Computers and Games*, 2006 · L. Kocsis, C. Szepesvári, "Bandit based Monte-Carlo planning", *ECML*, 2006.
- D. Silver vd., "Mastering the game of Go with deep neural networks and tree search", *Nature* 529, 484–489, 2016.

Son

Bölüm 2: Minimaks ve Alfa-Beta Budaması

Önceki: Bölüm 2: Oyunlar ve Temsiller
Sonraki bölüm: Bölüm 3: Nash Dengesi
Çalışma soruları · Minimaks XOX · Dörtlü Bağla · Terim sözlüğü