Bölüm 3: Özyineleme, İkili Arama ve Geri İzleme

C ile Veri Yapıları ve Algoritmalar

Yığın çerçevesi | Kuyruk özyinelemesi | İkili arama | Hanoi kuleleri | Permütasyon | Satisfiability | n-vezir | At gezisi

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Özyinelemeli düşünmek

Bir problemi taban durum + küçülen alt problem olarak ifade etmek; özyineleme ile döngüyü karşılaştırmak.

Yığını izlemek

Her çağrının yığın çerçevesini (parametreler, yereller, dönüş adresi) çizmek.

İkili arama

Yinelemeli ve özyinelemeli ikili aramayı yazmak; O(log n) sınırını göstermek.

Yineleme bağıntısı

Hanoi için $h(n) = 2h(n-1) + 1$ bağıntısını kurup çözmek; permütasyon için $O(n \cdot n!)$.

Geri izleme

Kaba kuvvet ile geri izlemeyi ayırmak; SAT, n-vezir, alt küme üretimi.

Sezgisel yöntem

At gezisinde Warnsdorff kuralının geri izlemeye göre neden hızlı olduğunu açıklamak.

ÖZYİNELEME

Özyinelemeli Algoritmalar (Recursive Algorithms)

Bütündeki çözüm mantığı **alt problemlerde de geçerliyse** özyinelemeli algoritma kullanılabilir.

Doğrudan özyineleme

Fonksiyon kendini çağırır: `fakt` → `fakt`.

Dolaylı özyineleme

Çağırdığı bir fonksiyon onu geri çağırır: `A` → `B` → `A`.

Taban durum

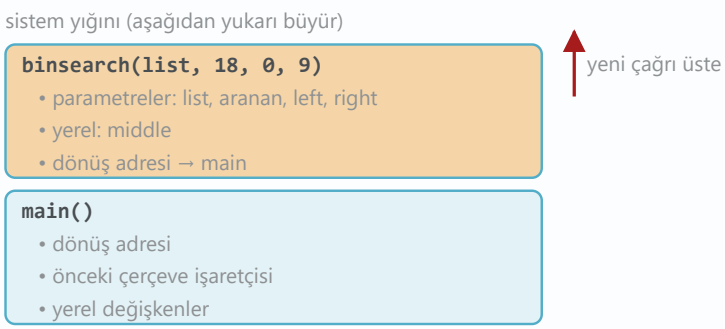
Özyinelemesiz çözülen en küçük örnek. Yoksa çağrılar sonsuza gider (yığın taşar).

Özyinelemeli adım

Problemi **daha küçük** bir örneğe indirger; her çağrı taban duruma yaklaşmalı.

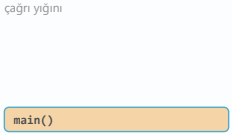
HOCADAN *"'Bu özyinelemeli çözüm çok bellek yemez mi?' Bu soru için erken. Önce kodu yazmak öğrenilir; algoritma karmaşıklığı hesaplanabildiğinde ancak böyle denebilir. Özyinelemeli kod yazmak öğrenilmeden bu yargıya varılmaz."*

Yığın Çerçevesi (Stack Activation Frame)



- Fonksiyon çağrılınca **sistem yığını** kullanılır.
- Denetim fonksiyona geçmeden önce bir hazırlık yapılır: nereye dönüleceği, parametrelerin saklanması…
- Bu iletişim–denetim yapısına **yığın çerçevesi** denir.
- Başlangıçta yığında iki değer vardır: **dönüş adresi** ve **önceki çerçeveye işaretçi**.
- Her özyinelemeli çağrı yeni bir çerçeve demektir → derinlik d ise ek bellek O(d).

Kod İzleme: `fakt(3)` ve çağrı yığını



main, `fakt(3)`'ü çağırır: yeni çerçeve açılacak.

Adım 1 / 9

```
1  long fakt(int n) {
2    if (n <= 1)
3      return 1;              /* taban durum */
4    return n * fakt(n - 1);
5  }
6
7  int main(void) {
▶    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

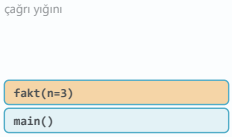
`n = –`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



fakt(3): n = 3 > 1, taban durum değil.

Adım 2 / 9

```
1  long fakt(int n) {
▶    if (n <= 1)
3      return 1;              /* taban durum */
4    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

`n = 3`

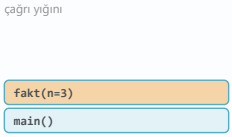
ÇAĞRI YIĞINI

`fakt(3)`

`main()`

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



`3 * fakt(2)` : çarpma bekler; önce fakt(2) çağrılır.

Adım 3 / 9

```
1  long fakt(int n) {
2    if (n <= 1)
3      return 1;              /* taban durum */
▶    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

`n = 3`

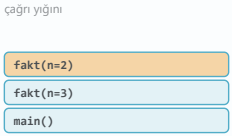
ÇAĞRI YIĞINI

`fakt(3)`

`main()`

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



fakt(2): n = 2 > 1, taban durum değil.

Adım 4 / 9

```
1  long fakt(int n) {
▶    if (n <= 1)
3      return 1;              /* taban durum */
4    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

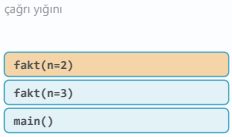
`n = 2`

ÇAĞRI YIĞINI

- **fakt(2)**
- fakt(3)
- main()

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



`2 * fakt(1)` : çarpma bekler; önce fakt(1) çağrılır.

Adım 5 / 9

```
1  long fakt(int n) {
2    if (n <= 1)
3      return 1;              /* taban durum */
▶    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

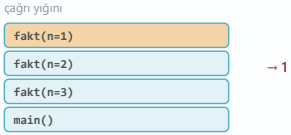
`n = 2`

ÇAĞRI YIĞINI

- **fakt(2)**
- fakt(3)
- main()

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



fakt(1): **taban durum** → return 1. Yığın en derin hâlde (4 çerçeve).

Adım 6 / 9

```
1  long fakt(int n) {
▶    if (n <= 1)
▶      return 1;              /* taban durum */
4    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

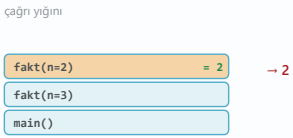
`n = 1`

ÇAĞRI YIĞINI

- **`fakt(1)`**
- `fakt(2)`
- `fakt(3)`
- `main()`

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



fakt(2) kaldığı yerden devam: 2 · 1 = **2**, çerçeve kapanır.

Adım 7 / 9

```
1  long fakt(int n) {
2    if (n <= 1)
3      return 1;              /* taban durum */
▶    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

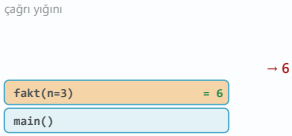
`n = 2` `fakt(n-1) = 1`

ÇAĞRI YIĞINI

- **fakt(2)**
- fakt(3)
- main()

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



fakt(3) kaldığı yerden devam: 3 · 2 = **6**, çerçeve kapanır.

Adım 8 / 9

```
1  long fakt(int n) {
2    if (n <= 1)
3      return 1;              /* taban durum */
▶    return n * fakt(n - 1);
5  }
6
7  int main(void) {
8    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

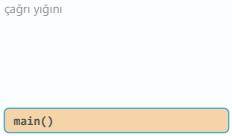
`n = 3` `fakt(n-1) = 2`

ÇAĞRI YIĞINI

- **fakt(3)**
- main()

ÇIKTI

Kod İzleme: `fakt(3)` ve çağrı yığını



main 6 yazar. Aynı n değişkeni her çerçevede **ayrı** bir kopyadır.

Adım 9 / 9

```
1  long fakt(int n) {
2    if (n <= 1)
3      return 1;               /* taban durum */
4    return n * fakt(n - 1);
5  }
6
7  int main(void) {
▶    printf("%ld\n", fakt(3));
9    return 0;
10 }
```

DEĞİŞKENLER

`n = –`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
6
```

Özyineleme mi, Döngü mü?

```c
/* ozyinelemeli */
long fakt(int n) {
  return n <= 1 ? 1 : n * fakt(n - 1);
}

/* dongulu */
long fakt_d(int n) {
  long f = 1;
  while (n > 1)
    f *= n--;
  return f;
}
```

	Özyinelemeli	Döngülü
Okunabilirlik	Tanıma çok yakın	Durum değişkenleri gerekir
Ek bellek	O(derinlik) yığın	O(1)
Çağrı maliyeti	Her adımda çerçeve	Yok
Risk	Derin özyinelemede yığın taşması	Sonsuz döngü
Doğal olduğu yer	Ağaç, böl-yönet, geri izleme	Düz tarama, sayaçlar

Her özyinelemeli algoritma, çağrı yığınını **açık bir yığınla** taklit ederek döngüye çevrilebilir (ve tersi). Seçim; açıklık, bellek ve hız arasındaki bir dengedir.

Horowitz–Sahni 1.2.2

ÖZYİNELEME

Kuyruk Özyinelemesi (Tail Recursion)

Özyinelemeli çağrı fonksiyonun **son işiyse** (dönüşten sonra yapılacak iş kalmıyorsa) buna kuyruk özyinelemesi denir.

```c
long fakt_k(int n, long acc) {
  if (n <= 1) return acc;
  return fakt_k(n - 1, n * acc);  /* son is: cagri */
}
/* ilk cagri: fakt_k(n, 1) */
```



fakt kuyruk özyinelemeli değil

`n * fakt(n-1)` : dönüşten sonra çarpma bekler, çerçeve saklanmalıdır.

Birikimli parametre

`acc` kısmi sonucu taşır; dönüşte yapılacak iş kalmaz.

Derleyici

Kuyruk çağrısı bir **atlama**ya çevrilebilir (O(1) yığın). C standardı bunu **garanti etmez**; gcc/clang optimizasyonla çoğu zaman yapar.

İkili Arama (Binary Search)

Sıralı bir dizide değer arıyoruz: bulunursa **indisini**, bulunamazsa **−1** döndüren yordam. Her adımda ortadaki öğeye bakılır ve aralığın yarısı atılır.

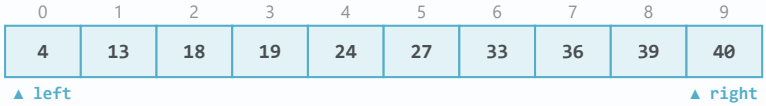


Test dizisi

```
#define COMPARE(x, y) ((x) < (y) ? -1 : (y) < (x) ? 1 : 0)
```

HOCADAN *"list[i] değil, i döndürülür. Neden? Hata değeri olan −1 ile dizi öğesi olabilecek bir −1 karışmasın diye."*

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



`binsearch(list, 18, 0, 9)` : aralık tüm dizi.

Adım 1 / 7

```
▶ int binsearch(int list[], int aranan,
▶               int left, int right) {
3   while (left <= right) {
4     int middle = (left + right) / 2;
5     switch (COMPARE(list[middle], aranan)) {
6       case -1: left = middle + 1; break;
7       case 0: return middle;
8       case 1: right = middle - 1;
9     }
10  }
11  return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 0` `right = 9`

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



0 ≤ 9: 1. tur. middle = (0 + 9) / 2 = **4**, list[4] = 24.

Adım 2 / 7

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶    while (left <= right) {
▶      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 0` `right = 9` `middle = 4`

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



24 > 18 → COMPARE = 1: sağ yarı atılır, **right = 3**.

Adım 3 / 7

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
▶        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 0` `right = 3` `middle = 4`

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



0 ≤ 3: 2. tur. middle = (0 + 3) / 2 = **1**, list[1] = 13.

Adım 4 / 7

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶    while (left <= right) {
▶      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 0` `right = 3` `middle = 1`

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



13 < 18 → COMPARE = −1: sol yarı atılır, **left = 2**.

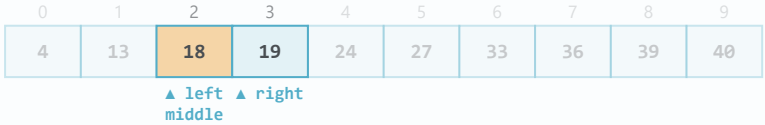
Adım 5 / 7

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
▶        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 2` `right = 3` `middle = 1`

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



2 ≤ 3: 3. tur. middle = (2 + 3) / 2 = **2**, list[2] = 18.

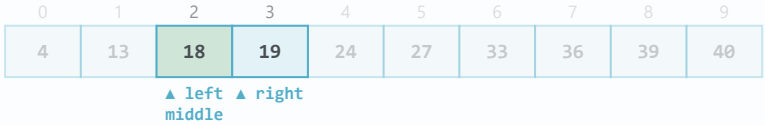
Adım 6 / 7

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶    while (left <= right) {
▶      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 2` `right = 3` `middle = 2`

Kod İzleme: yinelemeli `binsearch(list, 18, 0, 9)`



COMPARE = 0 → **return 2**. 3 turda bulundu.

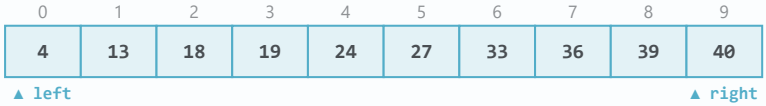
Adım 7 / 7

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
▶        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 18` `left = 2`
`right = 3` `middle = 2`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



`binsearch(list, 25, 0, 9)` : aralık tüm dizi.

Adım 1 / 8

```
▶ int binsearch(int list[], int aranan,
▶               int left, int right) {
3   while (left <= right) {
4     int middle = (left + right) / 2;
5     switch (COMPARE(list[middle], aranan)) {
6       case -1: left = middle + 1; break;
7       case 0: return middle;
8       case 1: right = middle - 1;
9     }
10  }
11  return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 0` `right = 9`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



0 ≤ 9: 1. tur. middle = (0 + 9) / 2 = **4**, list[4] = 24.

Adım 2 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶    while (left <= right) {
▶      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 0` `right = 9` `middle = 4`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



24 < 25 → COMPARE = −1: sol yarı atılır, **left = 5**.

Adım 3 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
▶        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 5` `right = 9` `middle = 4`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



5 ≤ 9: 2. tur. middle = (5 + 9) / 2 = **7**, list[7] = 36.

Adım 4 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶    while (left <= right) {
▶      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 5` `right = 9` `middle = 7`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



36 > 25 → COMPARE = 1: sağ yarı atılır, **right = 6**.

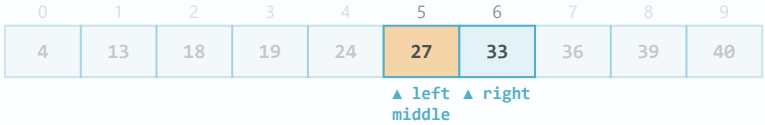
Adım 5 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
▶        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 5` `right = 6` `middle = 7`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



5 ≤ 6: 3. tur. middle = (5 + 6) / 2 = **5**, list[5] = 27.

Adım 6 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶   while (left <= right) {
▶     int middle = (left + right) / 2;
5     switch (COMPARE(list[middle], aranan)) {
6       case -1: left = middle + 1; break;
7       case 0: return middle;
8       case 1: right = middle - 1;
9     }
10  }
11  return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 5` `right = 6` `middle = 5`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



left = 5 > right = 4: aralık boş

27 > 25 → COMPARE = 1: sağ yarı atılır, **right = 4**.

Adım 7 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
▶        case 1: right = middle - 1;
9      }
10   }
11   return -1;
12 }
```

DEĞİŞKENLER

`aranan = 25` `left = 5` `right = 4` `middle = 5`

Kod İzleme: `binsearch(list, 25, 0, 9)` → −1



left = 5 > right = 4: aralık boş

left > right: döngü biter → **return −1** (3 tur).

Adım 8 / 8

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
▶    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle], aranan)) {
6        case -1: left = middle + 1; break;
7        case 0: return middle;
8        case 1: right = middle - 1;
9      }
10   }
▶    return -1;
12 }
```

DEĞİŞKENLER

`aranan` = **25** `left` = **5** `right` = **4**

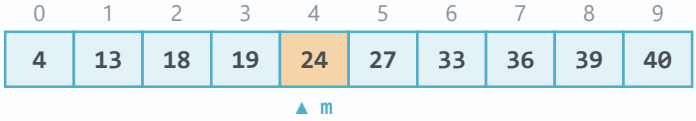
Özyinelemeli İkili Arama

```
int binsearch(int list[], int x, int l, int r) {
  int m;
  if (l > r) return -1;          /* aralik bos */
  m = (l + r) / 2;
  switch (COMPARE(list[m], x)) {
    case -1: return binsearch(list, x, m + 1, r);
    case 0:  return m;
    case 1:  return binsearch(list, x, l, m - 1);
  }
  return -1;                     /* ulasilmaz */
}
```

- Döngü yerine aralığı küçülterek **kendini çağırır**.
- Taban durumlar: bulundu (`case 0`) ya da aralık boş (`l > r`). Kısa olsun diye x = aranan, l/r = left/right, m = middle.
- Her iki özyinelemeli çağrı da **kuyruk çağrısıdır**: sonucu olduğu gibi döndürür.

Sık yapılan hata: özyinelemeli sürümde `if` yerine `while` yazmak. Her dal `return` ile çıktığı için döngü bir kez döner ve sonuç doğrudur, ama yanıltıcıdır: burada `if` olmalı.

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



`l=0 r=9 m=4`

Yeni çerçeve: l = 0, r = 9 → m = **4** (list[4] = 24).

Adım 1 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
▶    if (l > r) return -1;          /* aralik bos */
▶    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
6      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
8      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

DEĞİŞKENLER

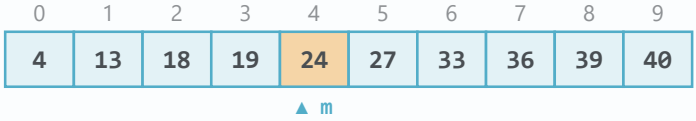
`l = 0` `r = 9` `m = 4`

ÇAĞRI YIĞINI

`binsearch(0, 9)`

`main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



`l=0 r=9 m=4`

24 > 18 → case 1: binsearch(list, 18, 0, 3) çağrılır.

Adım 2 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
3    if (l > r) return -1;          /* aralik bos */
4    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
6      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
▶      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

DEĞİŞKENLER

`l = 0` `r = 9` `m = 4`

ÇAĞRI YIĞINI

`binsearch(0, 9)`

`main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



l=0 r=3 m=1

l=0 r=9 m=4

Yeni çerçeve: l = 0, r = 3 → m = **1** (list[1] = 13).

Adım 3 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
▶    if (l > r) return -1;          /* aralik bos */
▶    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
6      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
8      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

DEĞİŞKENLER

`l = 0` `r = 3` `m = 1`

ÇAĞRI YIĞINI

- **`binsearch(0, 3)`**
- `binsearch(0, 9)`
- `main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



l=0 r=3 m=1

l=0 r=9 m=4

13 < 18 → case −1: binsearch(list, 18, 2, 3) çağrılır.

Adım 4 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
3    if (l > r) return -1;          /* aralik bos */
4    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
▶      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
8      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

DEĞİŞKENLER

`l = 0` `r = 3` `m = 1`

ÇAĞRI YIĞINI

- **`binsearch(0, 3)`**
- `binsearch(0, 9)`
- `main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



l=2 r=3 m=2

l=0 r=3 m=1

l=0 r=9 m=4

Yeni çerçeve: l = 2, r = 3 → m = **2** (list[2] = 18).

Adım 5 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
▶    if (l > r) return -1;          /* aralik bos */
▶    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
6      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
8      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

DEĞİŞKENLER

`l = 2` `r = 3` `m = 2`

ÇAĞRI YIĞINI

- **`binsearch(2, 3)`**
- `binsearch(0, 3)`
- `binsearch(0, 9)`
- `main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



l=2 r=3 m=2 →2

l=0 r=3 m=1

l=0 r=9 m=4

list[2] = 18: **case 0** → **return 2**.

Adım 6 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
3    if (l > r) return -1;          /* aralik bos */
4    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
6      case -1: return binsearch(list, x, m + 1, r);
▶      case 0:  return m;
8      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

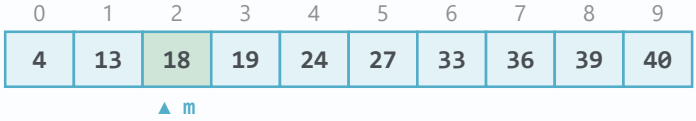
DEĞİŞKENLER

`l = 2` `r = 3` `m = 2`

ÇAĞRI YIĞINI

- **binsearch(2, 3)**
- binsearch(0, 3)
- binsearch(0, 9)
- main()

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



Dönen 2 olduğu gibi yukarı iletilir (case −1 çerçevesi kapanır).

Adım 7 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
3    if (l > r) return -1;          /* aralik bos */
4    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
▶      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
8      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                     /* ulasilmaz */
11 }
```

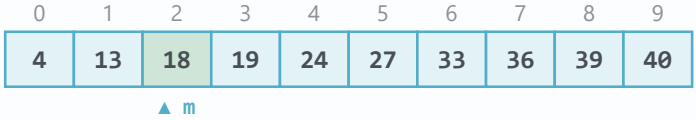
DEĞİŞKENLER

`l = 0` `r = 3` `m = 1`

ÇAĞRI YIĞINI

- `binsearch(0, 3)`
- `binsearch(0, 9)`
- `main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



`l=0 r=9 m=4 →2`

Dönen 2 olduğu gibi yukarı iletilir (case 1 çerçevesi kapanır).

Adım 8 / 9

```
1  int binsearch(int list[], int x, int l, int r) {
2    int m;
3    if (l > r) return -1;           /* aralik bos */
4    m = (l + r) / 2;
5    switch (COMPARE(list[m], x)) {
6      case -1: return binsearch(list, x, m + 1, r);
7      case 0:  return m;
▶      case 1:  return binsearch(list, x, l, m - 1);
9    }
10   return -1;                      /* ulasilmaz */
11 }
```

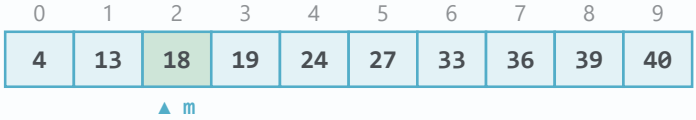
DEĞİŞKENLER

`l = 0` `r = 9` `m = 4`

ÇAĞRI YIĞINI

- `binsearch(0, 9)`
- `main()`

Kod İzleme: özyinelemeli `binsearch(list, 18, 0, 9)`



main'e **2** döner: "2'yi case 1'e döndürür, sonra da main'e döner." Derinlik 3 → O(log n) yığın.

Adım 9 / 9

```
▶ int binsearch(int list[], int x, int l, int r) {
2   int m;
3   if (l > r) return -1;          /* aralik bos */
4   m = (l + r) / 2;
5   switch (COMPARE(list[m], x)) {
6     case -1: return binsearch(list, x, m + 1, r);
7     case 0:  return m;
8     case 1:  return binsearch(list, x, l, m - 1);
9   }
10  return -1;                     /* ulasilmaz */
11 }
```

DEĞİŞKENLER

`sonuç = 2`

ÇAĞRI YIĞINI

`main()`

İkili Aramanın Karmaşıklığı

Her turda aralık **en az yarıya** iner: n → n/2 → n/4 → … → 1.

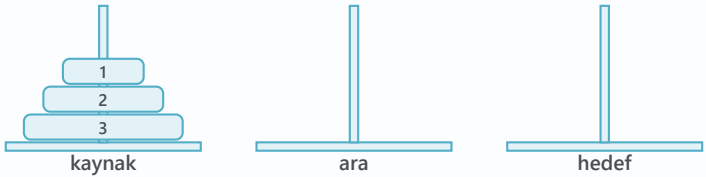
$$T(n) = T(n/2) + c \Rightarrow T(n) = O(\log n)$$

- En fazla $\lfloor \log_2 n \rfloor + 1$ tur.
- Yinelemeli sürüm ek bellek: **O(1)**.
- Özyinelemeli sürüm: derinlik kadar çerçeve → **O(log n)**.
- Ön koşul: dizi **sıralı** olmalı ve indisle O(1) erişilebilmeli (bağlı listede işe yaramaz).

n	en çok tur	doğrusal arama
10	4	10
1.000	10	1.000
1.000.000	20	1.000.000
1.000.000.000	30	1.000.000.000

Örnek dizide (n = 10) en çok 4 tur: 18 için 3 tur yetti.

Hanoi Kuleleri Problemi



Kaynak kulede n disk var; her disk üzerindekinden büyüktür. Amaç: hepsini **hedef** kuleye taşımak, **ara** kuleden yardım alarak.

1. Her seferinde **tek** disk taşınır.
2. Bir disk kendinden **küçük** bir diskin üzerine konamaz.

Özyinelemeli çözüm: En büyük diski hedefe taşıyabilmek için üstündeki n−1 diskin ara kulede olduğu bir an olmalıdır: n−1 disk kaynak→ara, en büyük disk kaynak→hedef, n−1 disk ara→hedef. **h(n) = h(n−1) + 1 + h(n−1)**.

HOCADAN *"Öyle problemler var ki, ancak özyinelemeli yolla çözülebilir."*

Kuramsal olarak her özyineleme açık bir yığınla döngüye çevrilebilir; vurgu, özyinelemenin bu tür problemlerde tek **doğal** ifade olmasıdır.

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 3: önce üstteki 2 disk A → B (hedef C yardımcı olur).

Adım 1 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
▶      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

ÇAĞRI YIĞINI

- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 2: önce üstteki 1 disk A → C (hedef B yardımcı olur).

Adım 2 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
▶      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

ÇAĞRI YIĞINI

- **`hanoi(2,A,C,B)`**
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 1: taban durum. Disk 1: **A** → **C**.

Adım 3 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
▶      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

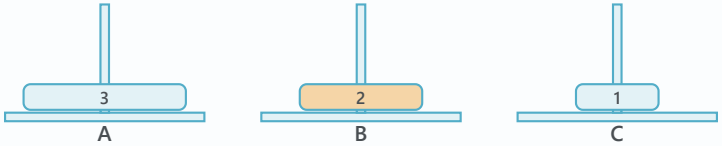
ÇAĞRI YIĞINI

- **`hanoi(1,A,B,C)`**
- `hanoi(2,A,C,B)`
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



Geri dönüldü; en büyük disk 2: **A** → **B**. Sonra 1 disk C → B.

Adım 4 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
3    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
▶      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

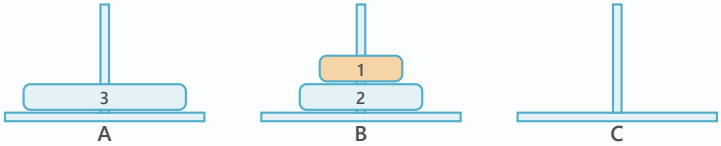
ÇAĞRI YIĞINI

- **`hanoi(2,A,C,B)`**
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
A -> B
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 1: taban durum. Disk 1: **C** → **B**.

Adım 5 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
▶      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

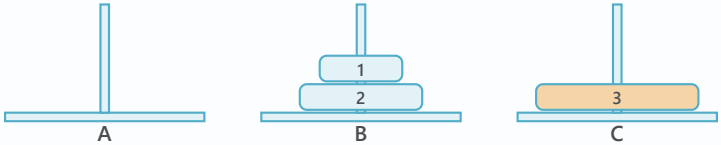
ÇAĞRI YIĞINI

- **`hanoi(1,C,A,B)`**
- `hanoi(2,A,C,B)`
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
A -> B
C -> B
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



Geri dönüldü; en büyük disk 3: **A** → **C**. Sonra 2 disk B → C.

Adım 6 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
3    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
▶      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

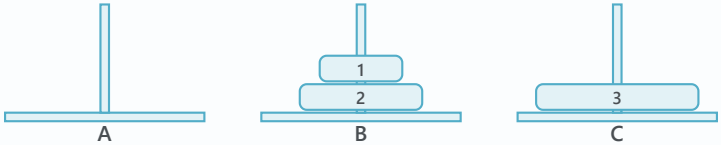
ÇAĞRI YIĞINI

- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
A -> B
C -> B
A -> C
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 2: önce üstteki 1 disk B → A (hedef C yardımcı olur).

Adım 7 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
▶      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

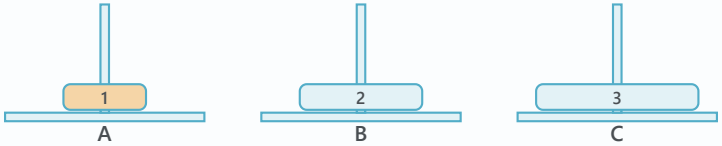
ÇAĞRI YIĞINI

- **`hanoi(2,B,A,C)`**
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
A -> B
C -> B
A -> C
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 1: taban durum. Disk 1: **B** → **A**.

Adım 8 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
▶      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

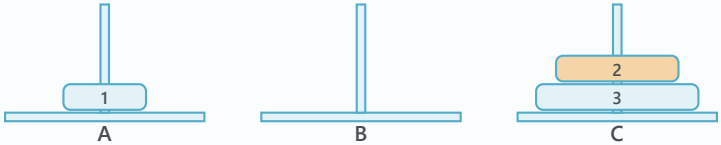
ÇAĞRI YIĞINI

- **`hanoi(1,B,C,A)`**
- `hanoi(2,B,A,C)`
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
A -> B
C -> B
A -> C
B -> A
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



Geri dönüldü; en büyük disk 2: **B** → **C**. Sonra 1 disk A → C.

Adım 9 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
3    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
▶      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

ÇAĞRI YIĞINI

- **`hanoi(2,B,A,C)`**
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C      A -> C
A -> B      B -> A
C -> B      B -> C
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



n = 1: taban durum. Disk 1: **A** → **C**.

Adım 10 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
▶    if (n == 1)
▶      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
11 /* main: hanoi(3, "A", "B", "C"); */
```

ÇAĞRI YIĞINI

- **`hanoi(1,A,B,C)`**
- `hanoi(2,B,A,C)`
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

Kod İzleme: `hanoi(3, "A", "B", "C")`



Bitti: $7 = 2^3 - 1$ hamle, 7 çağrı. Yığın en çok n = 3 çerçeve derinleşti.

Adım 11 / 11

```
1  void hanoi(int n, char *kaynak, char *ara,
2             char *hedef) {
3    if (n == 1)
4      printf("%s -> %s\n", kaynak, hedef);
5    else {
6      hanoi(n - 1, kaynak, hedef, ara);
7      printf("%s -> %s\n", kaynak, hedef);
8      hanoi(n - 1, ara, kaynak, hedef);
9    }
10 }
▶ /* main: hanoi(3, "A", "B", "C"); */
```

ÇAĞRI YIĞINI

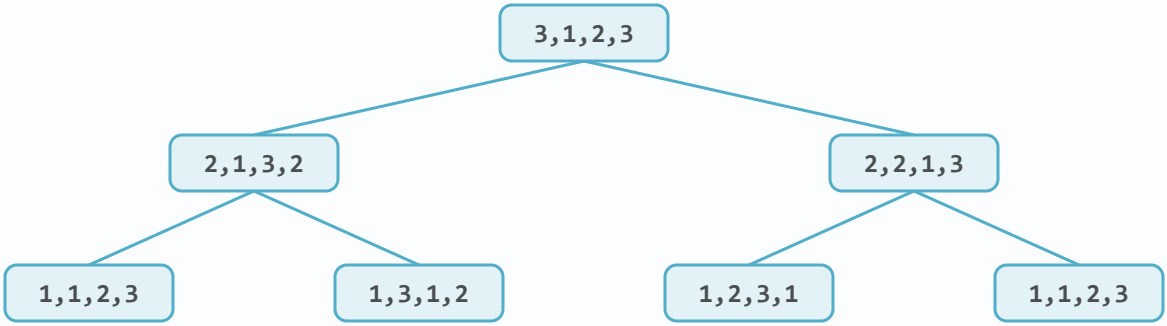
`main()`

ÇIKTI

```
A -> C      A -> C      A -> C
A -> B      B -> A
C -> B      B -> C
```

Ödev: `hanoi(3, "1", "2", "3")` Yığın Çerçeveleri

Her kutu bir çağrının çerçevesidir: `n, kaynak, ara, hedef`. Sol çocuk ilk özyinelemeli çağrı, sağ çocuk ikincisi. Yığının herhangi bir andaki içeriği, kökten o düğüme giden yoldur.



▶ **Hamle sırası (kök ortada gezinme)**

Yineleme Bağıntısını Çözmek: h(n) = 2·h(n−1) + 1

```
h(1) = 1
h(n) = 2·h(n−1) + 1
     = 4·h(n−2) + 2 + 1
     = 8·h(n−3) + 4 + 2 + 1
     = …
     = 2^(n−1)·h(1) + 2^(n−2) + … + 1
     = 2^n − 1
```

n	hamle
1	1
2	3
3	7
4	15
10	1.023
20	1.048.575
64	18.446.744.073.709.551.615

64 disk: $2^{64} - 1 \approx 1{,}8 \cdot 10^{19}$ hamle. Saniyede bir hamleyle ≈ **585 milyar yıl**. Algoritma $O(2^n)$ zaman, $O(n)$ yığın kullanır ve bundan iyisi yoktur: $2^n - 1$ hamle gerekli ve yeterlidir.

Permütasyon Problemi [O(n!)]

n ≥ 1 öğeli bir kümenin **tüm dizilişlerini** yazdırmak istiyoruz: n! diziliş vardır. {a, b, c} → abc, acb, bac, bca, cab, cba.

Özyinelemeli fikir

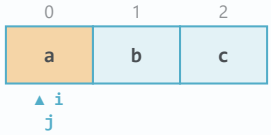
{a, b, c, d} kümesinin dizilişleri:

- a'yı izleyen {b, c, d} dizilişleri
- b'yi izleyen {a, c, d} dizilişleri
- c'yi izleyen {a, b, d} dizilişleri
- d'yi izleyen {a, b, c} dizilişleri

Takasla gerçekleme

i. konuma sırayla her öğe **takasla** getirilir, kalan n−i−1 konum özyinelemeli dizilir, sonra **geri takasla** dizi eski hâline döndürülür (geri izleme!).

Kod İzleme: `perm("abc", 0, 3)`



i = 0, j = 0: kendisiyle takas → **abc**; perm(i = 1) çağrılır.

Adım 1 / 11

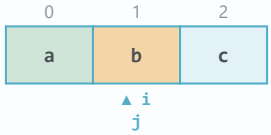
```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

ÇAĞRI YIĞINI

- **perm(i=1)**
- perm(i=0)
- main()

ÇIKTI

Kod İzleme: `perm("abc", 0, 3)`



i = 1, j = 1: kendisiyle takas → **abc**; perm(i = 2) çağrılır.

Adım 2 / 11

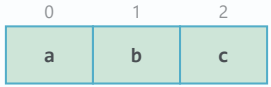
```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

ÇAĞRI YIĞINI

- **perm(i=2)**
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

Kod İzleme: `perm("abc", 0, 3)`



i = n: diziliş hazır → **abc** yazılır.

Adım 3 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
▶    if (i == n) {                    /* bir dizilis */
▶      for (j = 0; j < n; j++) printf("%c", list[j]);
▶      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

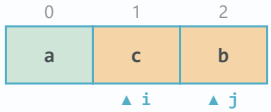
ÇAĞRI YIĞINI

- **perm(i=3)**
- perm(i=2)
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
```

Kod İzleme: `perm("abc", 0, 3)`



i = 1, j = 2: list[1] ↔ list[2] → **acb**; perm(i = 2) çağrılır.

Adım 4 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

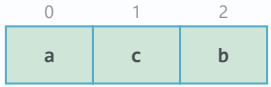
ÇAĞRI YIĞINI

- **perm(i=2)**
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
```

Kod İzleme: `perm("abc", 0, 3)`



i = n: diziliş hazır → **acb** yazılır.

Adım 5 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
▶    if (i == n) {                    /* bir dizilis */
▶      for (j = 0; j < n; j++) printf("%c", list[j]);
▶      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

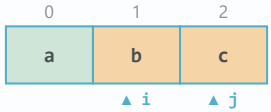
ÇAĞRI YIĞINI

- **perm(i=3)**
- perm(i=2)
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
```

Kod İzleme: `perm("abc", 0, 3)`



Geri takas: list[1] ↔ list[2] → **abc** (eski hâl).

Adım 6 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
▶        SWAP(list[i], list[j], temp);
11     }
12 }
```

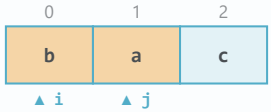
ÇAĞRI YIĞINI

- **perm(i=1)**
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
```

Kod İzleme: `perm("abc", 0, 3)`



i = 0, j = 1: list[0] ↔ list[1] → **bac**; perm(i = 1) çağrılır.

Adım 7 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

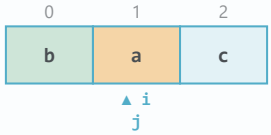
ÇAĞRI YIĞINI

- **perm(i=1)**
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
```

Kod İzleme: `perm("abc", 0, 3)`



i = 1, j = 1: kendisiyle takas → **bac**; perm(i = 2) çağrılır.

Adım 8 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

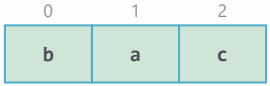
ÇAĞRI YIĞINI

- **perm(i=2)**
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
```

Kod İzleme: `perm("abc", 0, 3)`



i = n: diziliş hazır → **bac** yazılır.

Adım 9 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
▶    if (i == n) {                    /* bir dizilis */
▶      for (j = 0; j < n; j++) printf("%c", list[j]);
▶      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

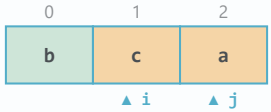
ÇAĞRI YIĞINI

- **perm(i=3)**
- perm(i=2)
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
bac
```

Kod İzleme: `perm("abc", 0, 3)`



i = 1, j = 2: list[1] ↔ list[2] → **bca**; perm(i = 2) çağrılır.

Adım 10 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

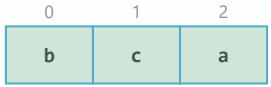
ÇAĞRI YIĞINI

- **perm(i=2)**
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
bac
```

Kod İzleme: `perm("abc", 0, 3)`



i = n: diziliş hazır → **bca** yazılır.

Adım 11 / 11

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
▶    if (i == n) {                    /* bir dizilis */
▶      for (j = 0; j < n; j++) printf("%c", list[j]);
▶      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

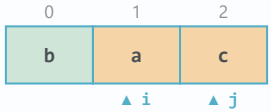
ÇAĞRI YIĞINI

- **perm(i=3)**
- perm(i=2)
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
bac
bca
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



Geri takas: list[1] ↔ list[2] → **bac** (eski hâl).

Adım 1 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
▶        SWAP(list[i], list[j], temp);
11     }
12 }
```

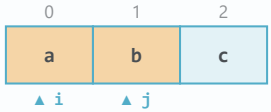
ÇAĞRI YIĞINI

- **perm(i=1)**
- perm(i=0)
- main()

ÇIKTI

```
abc         bca
acb
bac
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



Geri takas: list[0] ↔ list[1] → **abc** (eski hâl).

Adım 2 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
▶        SWAP(list[i], list[j], temp);
11     }
12 }
```

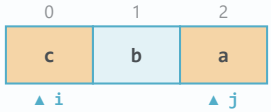
ÇAĞRI YIĞINI

- **perm(i=0)**
- main()

ÇIKTI

```
abc         bca
acb
bac
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



i = 0, j = 2: list[0] ↔ list[2] → **cba**; perm(i = 1) çağrılır.

Adım 3 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

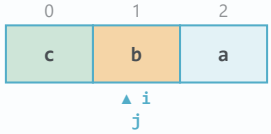
ÇAĞRI YIĞINI

- **perm(i=1)**
- perm(i=0)
- main()

ÇIKTI

```
abc         bca
acb
bac
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



i = 1, j = 1: kendisiyle takas → **cba**; perm(i = 2) çağrılır.

Adım 4 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

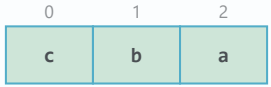
ÇAĞRI YIĞINI

- **perm(i=2)**
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc          bca
acb
bac
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



i = n: diziliş hazır → **cba** yazılır.

Adım 5 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
▶    if (i == n) {                    /* bir dizilis */
▶      for (j = 0; j < n; j++) printf("%c", list[j]);
▶      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

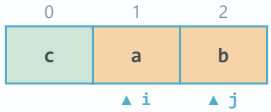
ÇAĞRI YIĞINI

- **perm(i=3)**
- perm(i=2)
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc
acb
bac
bca
cba
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



i = 1, j = 2: list[1] ↔ list[2] → **cab**; perm(i = 2) çağrılır.

Adım 6 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
▶      for (j = i; j < n; j++) {
▶        SWAP(list[i], list[j], temp);
▶        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

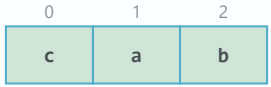
ÇAĞRI YIĞINI

- **perm(i=2)**
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc         bca
acb         cba
bac
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



i = n: diziliş hazır → **cab** yazılır.

Adım 7 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
▶    if (i == n) {                    /* bir dizilis */
▶      for (j = 0; j < n; j++) printf("%c", list[j]);
▶      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
10       SWAP(list[i], list[j], temp);
11     }
12 }
```

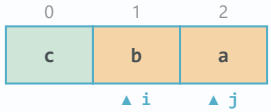
ÇAĞRI YIĞINI

- **perm(i=3)**
- perm(i=2)
- perm(i=1)
- perm(i=0)
- main()

ÇIKTI

```
abc         bca
acb         cba
bac         cab
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



Geri takas: list[1] ↔ list[2] → **cba** (eski hâl).

Adım 8 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
▶        SWAP(list[i], list[j], temp);
11     }
12 }
```

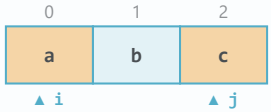
ÇAĞRI YIĞINI

- **perm(i=1)**
- perm(i=0)
- main()

ÇIKTI

```
abc         bca
acb         cba
bac         cab
```

Kod İzleme: `perm("abc", 0, 3)` (devam)



Geri takas: list[0] ↔ list[2] → **abc** (eski hâl).

Adım 9 / 9

```
1  void perm(char *list, int i, int n) {
2    int j, temp;
3    if (i == n) {                    /* bir dizilis */
4      for (j = 0; j < n; j++) printf("%c", list[j]);
5      printf("\n");
6    } else
7      for (j = i; j < n; j++) {
8        SWAP(list[i], list[j], temp);
9        perm(list, i + 1, n);
▶        SWAP(list[i], list[j], temp);
11     }
12 }
```

ÇAĞRI YIĞINI

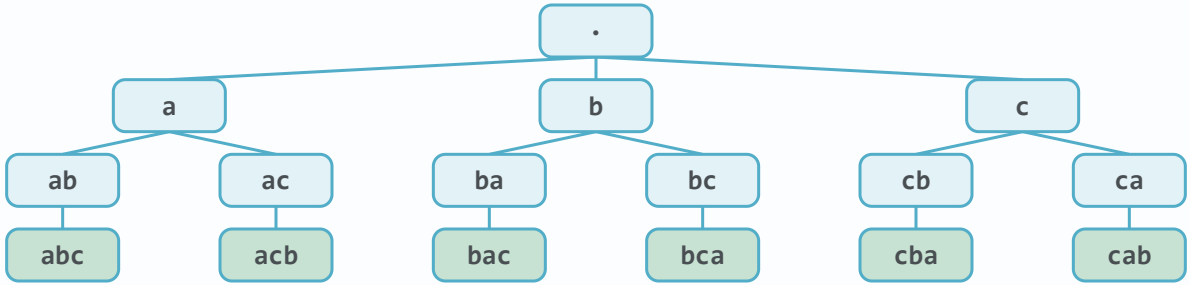
- **`perm(i=0)`**
- `main()`

ÇIKTI

```
abc        bca
acb        cba
bac        cab
```

PERMÜTASYON · ANALİZ

Özyineleme Ağacı ve Karmaşıklık



Kök: sabitlenmiş önek yok · her düzey bir konumu sabitler · yapraklar: çıktıdaki sırayla 6 diziliş

Çağrı sayısı

1 + n + n(n−1) + … + n! çağrı; yaprak sayısı n!.

Zaman

Her yaprakta n karakter yazılır → **O(n·n!)**. Sık görülen kısa yazım O(n!) bu n çarpanını gözden kaçırır.

Neden geri takas?

Kardeş dallar aynı başlangıç dizisini görmeli; ikinci SWAP diziyi onarır.

GERİ İZLEME

Kaba Kuvvet ve Geri İzleme (Backtracking)

Kaba kuvvet (brute force)

Olası tüm adaylar üretilir, her biri sınanır. Bilgisayarın gücüne güvenir; aday sayısı 2^n, n! … olabilir.

Geri izleme

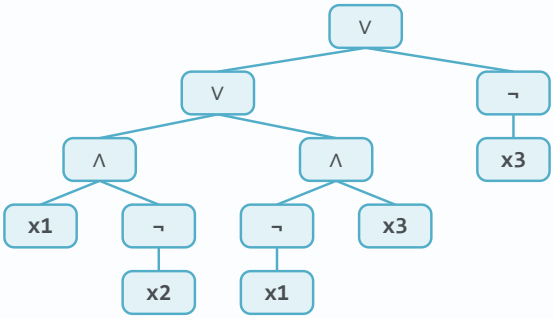
Çözüm **adım adım** kurulur; kısmi çözüm bir kuralı bozduğu anda o dal **budanır**, son seçim geri alınır, sıradaki denenir.

```
void geri_izle(kismi_cozum) {
  if (tam_cozum_mu) { yaz; return; }
  for (her aday secim)
    if (kurallara_uygun) { sec; geri_izle(...); secimi_geri_al; }
}
```

HOCADAN *"Daha iyi çözüm aramak: Veri yapılarıyla problem çözerken seçilen veri yapısından asla tam emin olunmamalı. Sürekli yazılmalı, tartışılmalı; "Daha iyi bir çözüm yöntemi vardır da biz bulamıyoruz." denmeli."*

GERİ İZLEME

Satisfiability (SAT) Problemi



(x1 ∧ ¬x2) ∨ (¬x1 ∧ x3) ∨ ¬x3

x1, x2, … true/false alan mantıksal değişkenler; ∧, ∨, ¬ işleçler. **Soru:** formülü **true** yapan bir atama var mı?

- 3 değişken → $2^3 = 8$ olası atama: {F,F,F}, {F,F,T}, …
- n değişken → 2^n durum; kaba kuvvet **$O(2^n)$**. m işlemciye eşit bölünse bile $O(2^n/m)$: hâlâ üstel.
- SAT, polinom zamanlı algoritması **bilinmeyen** NP-tam bir problemdir. 2-SAT polinom zamanda çözülür, 3-SAT NP-tamdır.

HOCADAN *"Keşke Computational Theory (hesaplama kuramı) dersi de alınmış olsa; problemler orada sınıflandırılıyor."*

GERİ İZLEME

SAT: Kök Sonda Gezinmeyle Değerlendirme

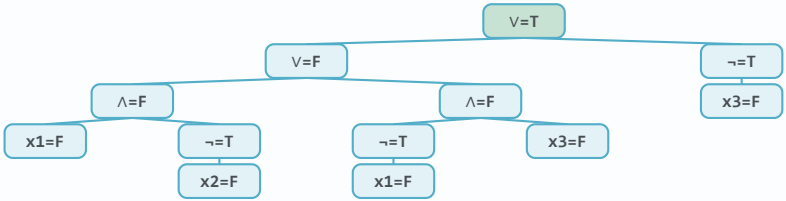
```
typedef enum { DEGIL, VE, VEYA, DOGRU, YANLIS } mantik;
typedef struct agac *agac_gost;
struct agac {
  agac_gost solcocuk, sagcocuk;
  mantik veri;
  short deger;
};

void kok_sonda_degerlendir(agac_gost L) {
  if (L) {
    kok_sonda_degerlendir(L->solcocuk);
    kok_sonda_degerlendir(L->sagcocuk);
    switch (L->veri) {
      case DEGIL: L->deger = !L->sagcocuk->deger; break;
      case VE:   L->deger = L->solcocuk->deger
                          && L->sagcocuk->deger; break;
      case VEYA: L->deger = L->solcocuk->deger
                          || L->sagcocuk->deger; break;
      case DOGRU:  L->deger = 1; break;
      case YANLIS: L->deger = 0; break;
    }
  }
}
```

- **Kök-sonda** (postorder): önce iki çocuk, sonra düğümün kendisi.
- ¬ düğümünün tek işleneni **sağ** çocuktadır.
- Yapraklar: her atamada değişkenlerin değeri `DOGRU` / `YANLIS` olarak yerleştirilir.

`enum { not, and, or, true, false }` gibi adlar C99'un `<stdbool.h>` makrolarıyla ve C23 anahtar sözcükleriyle çakışır; `typedef struct agac {…};` da ad vermeyen bir typedef'tir. Burada ASCII ve çakışmasız adlar kullandık.

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 000: ¬x3 = T → kök **true: yazdır**.

Adım 1 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

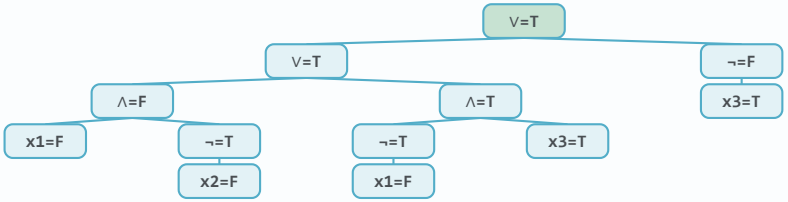
DEĞİŞKENLER

`m = 0` `x1 = 0` `x2 = 0` `x3 = 0` `kok = T`

ÇIKTI

```
000
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 001: ¬x1 ∧ x3 = T → kök **true: yazdır**.

Adım 2 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

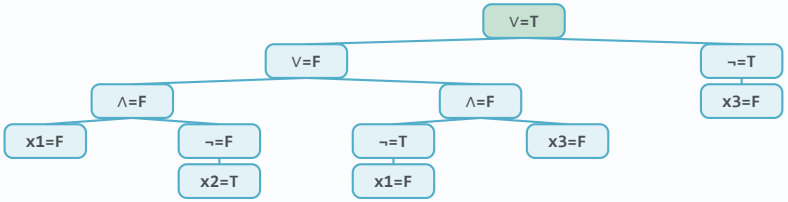
DEĞİŞKENLER

`m = 1` `x1 = 0` `x2 = 0` `x3 = 1` `kok = T`

ÇIKTI

```
000
001
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 010: ¬x3 = T → kök **true: yazdır**.

Adım 3 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

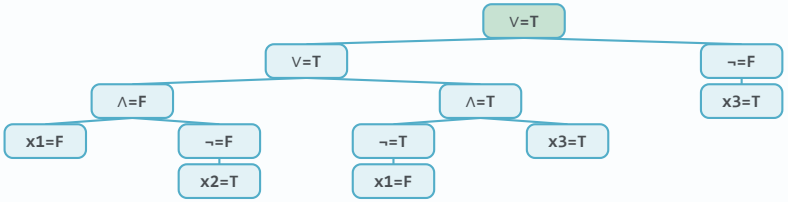
DEĞİŞKENLER

`m = 2` `x1 = 0` `x2 = 1` `x3 = 0` `kok = T`

ÇIKTI

```
000
001
010
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 011: ¬x1 ∧ x3 = T → kök **true: yazdır**.

Adım 4 / 9

```
 1 int m, x1, x2, x3;
 2 for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

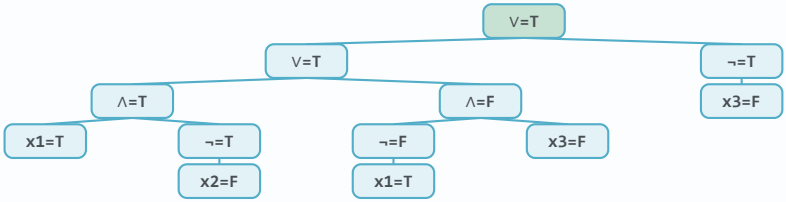
DEĞİŞKENLER

`m = 3` `x1 = 0` `x2 = 1` `x3 = 1` `kok = T`

ÇIKTI

```
000        011
001
010
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 100: ¬x3 = T → kök **true: yazdır**.

Adım 5 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

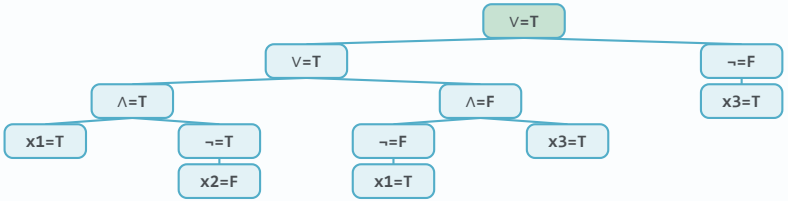
DEĞİŞKENLER

`m = 4` `x1 = 1` `x2 = 0` `x3 = 0` `kok = T`

ÇIKTI

```
000
001
010
011
100
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 101: x1 ∧ ¬x2 = T → kök **true: yazdır**.

Adım 6 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

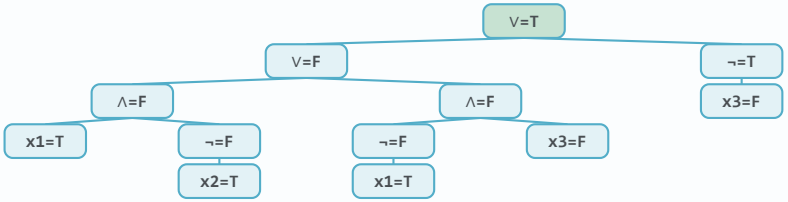
DEĞİŞKENLER

`m = 5` `x1 = 1` `x2 = 0` `x3 = 1` `kok = T`

ÇIKTI

```
000        011
001        100
010        101
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 110: ¬x3 = T → kök **true: yazdır**.

Adım 7 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {     /* 2^3 atama */
     x1 = (m >> 2) & 1;
     x2 = (m >> 1) & 1;
     x3 = m & 1;
     yapraklari_yerlestir(kok, x1, x2, x3);
     kok_sonda_degerlendir(kok);
     if (kok->deger)
       printf("%d%d%d\n", x1, x2, x3);
10 }
```

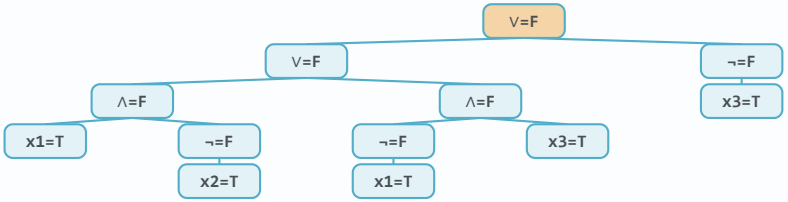
DEĞİŞKENLER

`m = 6` `x1 = 1` `x2 = 1` `x3 = 0` `kok = T`

ÇIKTI

```
000        011        110
001        100
010        101
```

Kod İzleme: 2^3 atamanın hepsini denemek



Atama 111: hiçbir terim T değil → kök **false**.

Adım 8 / 9

```
1  int m, x1, x2, x3;
2  for (m = 0; m < 8; m++) {    /* 2^3 atama */
       x1 = (m >> 2) & 1;
       x2 = (m >> 1) & 1;
       x3 = m & 1;
       yapraklari_yerlestir(kok, x1, x2, x3);
       kok_sonda_degerlendir(kok);
       if (kok->deger)
9        printf("%d%d%d\n", x1, x2, x3);
10 }
```

DEĞİŞKENLER

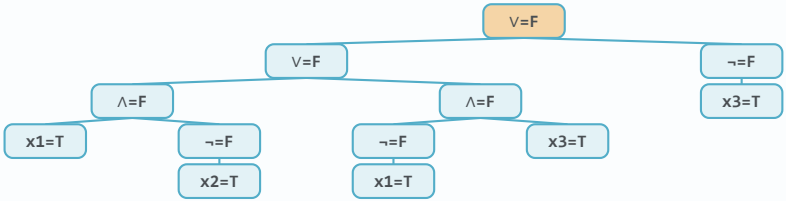
`m = 7` `x1 = 1` `x2 = 1` `x3 = 1` `kok = F`

ÇIKTI

```
000
001
010
011
100
101
110
```

GERİ İZLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: 2^3 atamanın hepsini denemek



8 atamanın **7**'si formülü sağlar; yalnız x1 = x2 = x3 = 1 sağlamaz. İlk çözümde `return` eden bir yordam 000'da durdu.

Adım 9 / 9

```
1  int m, x1, x2, x3;
▶ for (m = 0; m < 8; m++) {     /* 2^3 atama */
3    x1 = (m >> 2) & 1;
4    x2 = (m >> 1) & 1;
5    x3 = m & 1;
6    yapraklari_yerlestir(kok, x1, x2, x3);
7    kok_sonda_degerlendir(kok);
8    if (kok->deger)
9      printf("%d%d%d\n", x1, x2, x3);
10 }
```

DEĞİŞKENLER

`m = 8` `bulunan = 7`

ÇIKTI

```
000
001
010
011
100
101
110
```

GERİ İZLEME

SAT Türleri ve Ön Hesaplama (Precomputing)

CIRCUIT-SAT

Verilen bir mantık devresinin çıkışını 1 yapacak bir giriş ataması var mı?

Formula SAT

Yukarıdaki gibi ∧, ∨, ¬ ile kurulmuş bir önerme formülü.

3-CNF-SAT

Her biri 3 değişken içeren ∨ alt cümlelerinin (clause) ∧'si: (x1 ∨ ¬x1 ∨ ¬x2) ∧ (x3 ∨ x2 ∨ x4) ∧ (¬x1 ∨ ¬x3 ∨ ¬x4).

Ön hesaplama

İşlem ağacı **bir kez** kurulur; her atamada yalnız yapraklar değişir. Defalarca yapılacak iş için bir kez hazırlanan veri yapısı verimi çok artırır.

HOCADAN *"Google'a yazılır, tak bulunur! İşte hızlı algoritma: Knuth–Morris–Pratt. Algoritma analizi kitaplarındaki greedy algoritmalar, branch and bound ve precomputing konuları mutlaka çalışılmalı!"*

"I haven't failed, I've found 10,000 ways that don't work." — Thomas Edison'a atfedilen söz.

GERİ İZLEME

n-Vezir (n-Queens) Problemi

n×n satranç tahtasına n veziri, **hiçbiri diğerini tehdit etmeyecek** biçimde yerleştirmek: aynı satır, sütun ve çaprazda iki vezir olamaz.

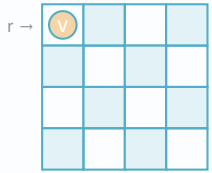
- Her satıra bir vezir → çözüm bir `q[0..n−1]` dizisi.
- Kaba kuvvet: n^n aday (satır başına bir vezirle; n = 8 için 16,7 milyon).
- Geri izleme: satır satır ilerlenir; uygun sütun yoksa bir önceki satıra dönülür.
- Çapraz testi: |q[i] − c| == r − i.

```
int q[8];  /* q[r]: r. satirdaki vezirin sutunu */

int uygun(int r, int c) {
  int i;
  for (i = 0; i < r; i++)
    if (q[i] == c || abs(q[i] - c) == r - i)
      return 0;          /* ayni sutun ya da capraz */
  return 1;
}
```

Horowitz–Sahni; Sedgewick. n = 4 için 2, n = 8 için 92 çözüm vardır.

Kod İzleme: `yerlestir(0, 4)`



(0, 0) uygun → q[0] = 0, sonraki satıra geçilir.

Adım 1 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

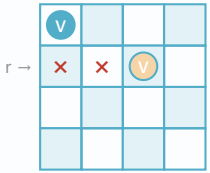
DEĞİŞKENLER

`r = 0` `c = 0` `q = [0, -, -, -]`

ÇAĞRI YIĞINI

- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Sütun 0, 1 çakışıyor (×); **(1, 2) uygun** → q[1] = 2, sonraki satıra geçilir.

Adım 2 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

DEĞİŞKENLER

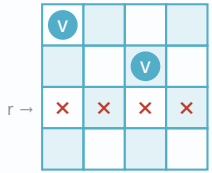
`r = 1` `c = 2` `q = [0, 2, –, –]`

ÇAĞRI YIĞINI

- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

GERİ İZLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `yerlestir(0, 4)`



Sütun 0, 1, 2, 3 çakışıyor; satır 2'ye vezir konamaz → **return 0**: geri dönülür, bir üst satır sıradaki sütunu dener.

Adım 3 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
4    for (c = 0; c < n; c++)
5      if (uygun(r, c)) {
6        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
▶    return 0;                        /* geri don */
10 }
```

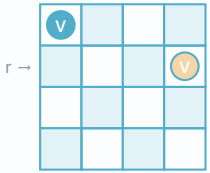
DEĞİŞKENLER

`r = 2` `q = [0, 2, -, -]`

ÇAĞRI YIĞINI

- **`yerlestir(2)`**
- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



(1, 3) uygun → q[1] = 3, sonraki satıra geçilir.

Adım 4 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

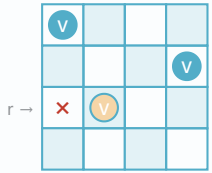
DEĞİŞKENLER

`r = 1` `c = 3` `q = [0, 3, –, –]`

ÇAĞRI YIĞINI

- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Sütun 0 çakışıyor (×); **(2, 1) uygun** → q[2] = 1, sonraki satıra geçilir.

Adım 5 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

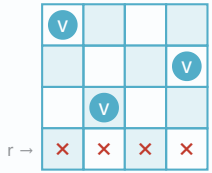
DEĞİŞKENLER

`r = 2` `c = 1` `q = [0, 3, 1, -]`

ÇAĞRI YIĞINI

- `yerlestir(2)`
- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Sütun 0, 1, 2, 3 çakışıyor; satır 3'e vezir konamaz → **return 0**: geri dönülür, bir üst satır sıradaki sütunu dener.

Adım 6 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
4    for (c = 0; c < n; c++)
5      if (uygun(r, c)) {
6        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
▶    return 0;                        /* geri don */
10 }
```

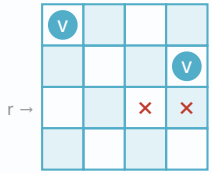
DEĞİŞKENLER

`r = 3` `q = [0, 3, 1, -]`

ÇAĞRI YIĞINI

- **`yerlestir(3)`**
- `yerlestir(2)`
- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Sütun 2, 3 çakışıyor; satır 2'ye vezir konamaz → **return 0**: geri dönülür, bir üst satır sıradaki sütunu dener.

Adım 7 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
4    for (c = 0; c < n; c++)
5      if (uygun(r, c)) {
6        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
▶    return 0;                        /* geri don */
10 }
```

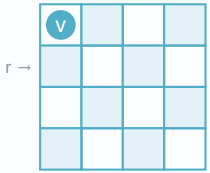
DEĞİŞKENLER

`r = 2` `q = [0, 3, -, -]`

ÇAĞRI YIĞINI

- **`yerlestir(2)`**
- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Kalan sütun yok; satır 1'e vezir konamaz → **return 0**: geri dönülür, bir üst satır sıradaki sütunu dener.

Adım 8 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
4    for (c = 0; c < n; c++)
5      if (uygun(r, c)) {
6        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
▶    return 0;                        /* geri don */
10 }
```

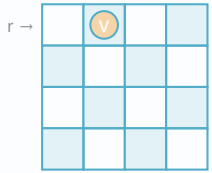
DEĞİŞKENLER

`r = 1` `q = [0, –, –, –]`

ÇAĞRI YIĞINI

- **`yerlestir(1)`**
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



(0, 1) uygun → q[0] = 1, sonraki satıra geçilir.

Adım 9 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

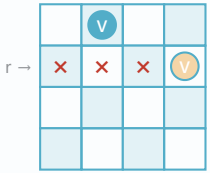
DEĞİŞKENLER

`r = 0` `c = 1` `q = [1, -, -, -]`

ÇAĞRI YIĞINI

- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Sütun 0, 1, 2 çakışıyor (×); **(1, 3) uygun** → q[1] = 3, sonraki satıra geçilir.

Adım 10 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

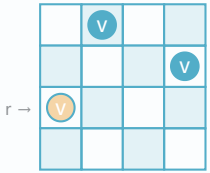
DEĞİŞKENLER

`r = 1` `c = 3` `q = [1, 3, –, –]`

ÇAĞRI YIĞINI

- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



(2, 0) uygun → q[2] = 0, sonraki satıra geçilir.

Adım 11 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

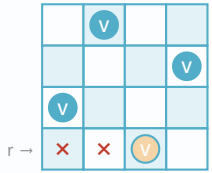
DEĞİŞKENLER

`r = 2` `c = 0` `q = [1, 3, 0, -]`

ÇAĞRI YIĞINI

- `yerlestir(2)`
- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



Sütun 0, 1 çakışıyor (×); **(3, 2) uygun** → q[3] = 2, sonraki satıra geçilir.

Adım 12 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
3    if (r == n) return 1;              /* cozum */
▶    for (c = 0; c < n; c++)
▶      if (uygun(r, c)) {
▶        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

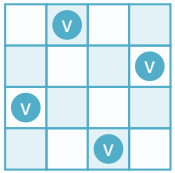
DEĞİŞKENLER

`r = 3` `c = 2` `q = [1, 3, 0, 2]`

ÇAĞRI YIĞINI

- `yerlestir(3)`
- `yerlestir(2)`
- `yerlestir(1)`
- `yerlestir(0)`
- `main()`

Kod İzleme: `yerlestir(0, 4)`



r = n: dört vezir yerleşti → **çözüm q = [1, 3, 0, 2]**. 8 yerleştirme yaptık; kaba kuvvet $4^4 = 256$ aday.

Adım 13 / 13

```
1  int yerlestir(int r, int n) {
2    int c;
▶   if (r == n) return 1;              /* cozum */
4    for (c = 0; c < n; c++)
5      if (uygun(r, c)) {
6        q[r] = c;
7        if (yerlestir(r + 1, n)) return 1;
8      }
9    return 0;                    /* geri don */
10 }
```

DEĞİŞKENLER

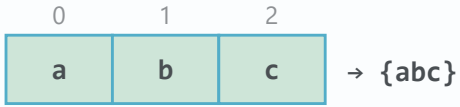
`r = 4` `q = [1, 3, 0, 2]`

ÇAĞRI YIĞINI

- **yerlestir(4)**
- yerlestir(3)
- yerlestir(2)
- yerlestir(1)
- yerlestir(0)
- main()

GERİ İZLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: al a, al b, al c → **{abc}**.

Adım 1 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

DEĞİŞKENLER

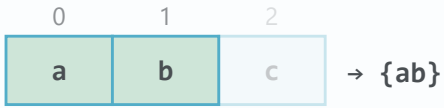
`sec = [1, 1, 1]` `i = 3`

ÇIKTI

```
{abc}
```

GERİ İZLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: al a, al b, alma c → **{ab}**.

Adım 2 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

DEĞİŞKENLER

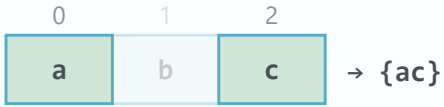
`sec = [1, 1, 0]` `i = 3`

ÇIKTI

```
{abc}
{ab}
```

GERİ İZLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: al a, alma b, al c → **{ac}**.

Adım 3 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

DEĞİŞKENLER

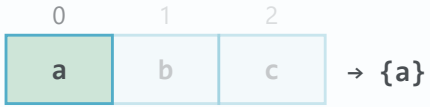
`sec = [1, 0, 1]` `i = 3`

ÇIKTI

```
{abc}
{ab}
{ac}
```

GERİ İZLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: al a, alma b, alma c → **{a}**.

Adım 4 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

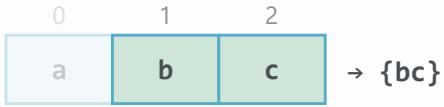
DEĞİŞKENLER

`sec = [1, 0, 0]` `i = 3`

ÇIKTI

```
{abc}       {a}
{ab}
{ac}
```

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: alma a, al b, al c → **{bc}**.

Adım 5 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

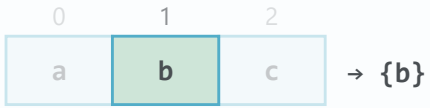
DEĞİŞKENLER

`sec = [0, 1, 1]` `i = 3`

ÇIKTI

```
{abc}       {a}
{ab}        {bc}
{ac}
```

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: alma a, al b, alma c → **{b}**.

Adım 6 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

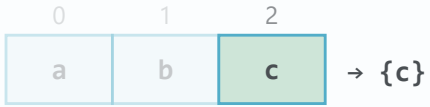
DEĞİŞKENLER

`sec = [0, 1, 0]` `i = 3`

ÇIKTI

```
{abc}       {a}
{ab}        {bc}
{ac}        {b}
```

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: alma a, alma b, al c → **{c}**.

Adım 7 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

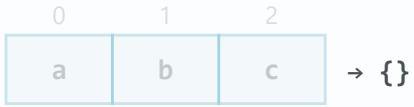
DEĞİŞKENLER

`sec = [0, 0, 1]` `i = 3`

ÇIKTI

```
{abc}   {a}   {c}
{ab}    {bc}
{ac}    {b}
```

Kod İzleme: `altkume("abc", 3, 0, sec)`



Kararlar: alma a, alma b, alma c → **{}**.

Adım 8 / 8

```
1  void altkume(char *s, int n, int i,
2               int sec[]) {
3    int j;
▶    if (i == n) {                  /* karar bitti */
▶      printf("{");
▶      for (j = 0; j < n; j++)
▶        if (sec[j]) printf("%c", s[j]);
▶      printf("}\n");
9      return;
10   }
11   sec[i] = 1; altkume(s, n, i + 1, sec);
12   sec[i] = 0; altkume(s, n, i + 1, sec);
13 }
```

DEĞİŞKENLER

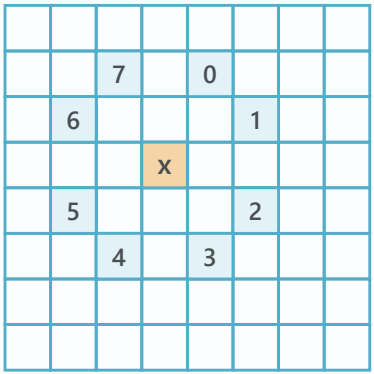
`sec = [0, 0, 0]` `i = 3`

ÇIKTI

```
{abc}   {a}   {c}
{ab}    {bc}  {}
{ac}    {b}
```

GERİ İZLEME

Satranç Tahtasında At Gezisi (Knight's Tour)



At, her kareden **bir kez** geçmek şartıyla tüm kareleri gezecek. Hareket L biçimli: 2 yatay + 1 dikey ya da 2 dikey + 1 yatay.

Hamle	0	1	2	3	4	5	6	7
Δi	−2	−1	+1	+2	+2	+1	−1	−2
Δj	+1	+2	+2	+1	−1	−2	−2	−1

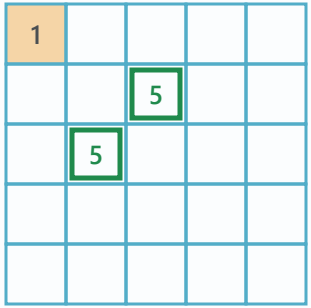
- **Geri izleme:** hamle denenir; çıkmaza girilince son hamle geri alınır, sıradaki denenir. Her zaman çözüm bulur, ama kaba kuvvet niteliğindedir: 8×8'de çok yavaş.
- **Sezgisel yöntem** (heuristic): "akıllıca" seçim; optimale yakın çözer, garanti vermez (mayın tarlası gibi).

Geri izleme "senaryoyu sondan başa geri sarmak" değildir: kısmi çözümü ileri kurar, çıkmazda son seçimi geri alır.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



1. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 5, 5 → en küçük pozitif: **5**, (1, 2) seçilir.

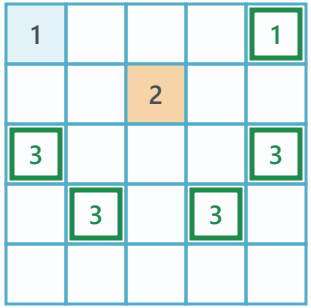
Adım 1 / 9

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



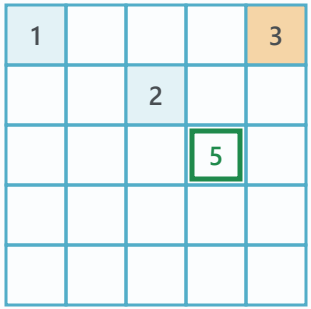
2. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 1, 3, 3, 3, 3 → en küçük pozitif: **1**, (0, 4) seçilir.

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



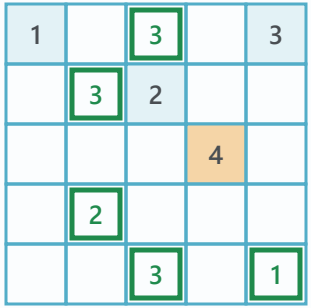
3. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 5 → en küçük pozitif: **5**, (2, 3) seçilir.

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



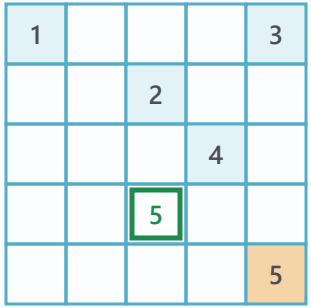
4. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 1, 3, 2, 3, 3 → en küçük pozitif: **1**, (4, 4) seçilir.

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



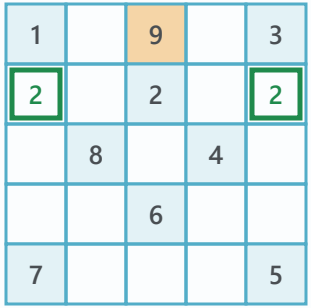
5. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 5 → en küçük pozitif: **5**, (3, 2) seçilir.

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



9. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 2, 2 → en küçük pozitif: **2**, (1, 4) seçilir.

Adım 6 / 9

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.



16. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 1, 4 → en küçük pozitif: **1**, (4, 2) seçilir.

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.

1	14	9	20	3
24	19	2	15	10
13	8	0	4	21
18	23	6	11	16
7	12	17	22	5

24. kare. Adaylar (yeşil sayı = oradan sonraki hamle sayısı): 0 → son boş kare tek aday: **(2, 2)** seçilir.

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

GERİ İZLEME

Warnsdorff'un Sezgisel Kuralı (1823)

"Atın bir sonraki gideceği yer, **ilerleme sayısı sıfır olmayan ama en az olan** karedir." İlerleme sayısı: o kareden doğrudan gidilebilecek boş kare sayısı. Önce köşeler ve kenarlar tüketilir, çıkmaz sonraya kalmaz.

1	14	9	20	3
24	19	2	15	10
13	8	25	4	21
18	23	6	11	16
7	12	17	22	5

25 kare, hiç geri dönmeden gezildi. Warnsdorff her adımda yalnız 8 komşuya bakar: toplam iş $O(n^2)$.

Adım 9 / 9

Kural adını H. C. von Warnsdorff'tan alır. Eşitlikte hamle tablosundaki sıra (0…7) kullanıldı.

Laboratuvar: İkili Arama

18 aranıyor

Girdi (aranan: artan sıralı sayılar) `18: 4, 13, 18, 19, 24, 27, 33, 36, 39, 40` Uygula

KOD

```
1  int binsearch(int list[], int aranan,
2                int left, int right) {
3    while (left <= right) {
4      int middle = (left + right) / 2;
5      switch (COMPARE(list[middle],
6                      aranan)) {
7        case -1: left = middle + 1; break;
8        case 0: return middle;
9        case 1: right = middle - 1;
10     }
11   }
12   return -1;
13 }
```



`aranan 18` `left 0` `right 9` `tur 0`

ÇIKTI (henüz yok)

`binsearch(list, 18, 0, 9)` : arama aralığı tüm dizi. Her turda aralığın ortasındaki öğe aranan ile karşılaştırılır ve aralık yarıya iner.

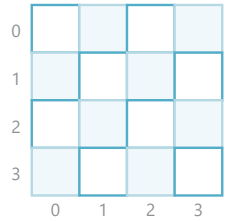
Laboratuvar: n-Vezir ve Geri İzleme

4-vezir

Girdi (tahta boyutu n, 1–8) 4 Uygula

KOD

```
 1 int q[8];
 2
 3 int yerlestir(int r, int n) {
 4   int c;
 5   if (r == n) return 1;          /* çözüm */
 6   for (c = 0; c < n; c++)
 7     if (uygun(r, c)) {
 8       q[r] = c;
 9       if (yerlestir(r + 1, n)) return 1;
10     }
11   return 0;                    /* geri dön */
12 }
13 /* uygun(r, c): q[0..r-1] ile aynı sütun
14    ya da çapraz yoksa 1 */
```



n 4 | r 0 | q [] | çağrılar –

ÇIKTI (henüz yok)

`yerlestir(0, 4)` : her satıra bir vezir konur. Bir satırda uygun sütun yoksa çağrı 0 döner ve bir önceki satır sıradaki sütunu dener (geri izleme).

ÖZET

Karmaşıklık Özeti

Algoritma	Zaman	Yığın / ek bellek	Not
`fakt` (özyinelemeli)	$\Theta(n)$	$O(n)$	kuyruk sürümü $O(1)$'e indirilebilir
İkili arama (yinelemeli)	$O(\log n)$	$O(1)$	dizi sıralı olmalı
İkili arama (özyinelemeli)	$O(\log n)$	$O(\log n)$	kuyruk özyinelemesi
Hanoi	$\Theta(2^n)$	$O(n)$	$2^n - 1$ hamle
Permütasyon	$\Theta(n \cdot n!)$	$O(n)$	$n!$ diziliş
Alt küme üretimi	$\Theta(n \cdot 2^n)$	$O(n)$	2^n alt küme
SAT (kaba kuvvet)	$O(2^n \cdot \lvert F \rvert)$	$O(\lvert F \rvert)$	$\lvert F \rvert$: formül ağacının boyu
n-vezir (geri izleme)	$O(n!)$ üst sınır	$O(n)$	budama pratikte çok daha az
At gezisi (Warnsdorff)	$O(n^2)$	$O(n^2)$	sezgisel; geri izleme üstel

ÖZET

Sık Yapılan Hatalar

1. Taban durumu unutmak ya da ona hiç ulaşmayan adım yazmak → yığın taşması.

2. Özyinelemeli çağrının sonucunu `return` etmemek: `binsearch( ... )` yazıp değeri atmak.

3. İkili aramayı sırasız dizide ya da bağlı listede kullanmak.

4. Bulunan öğenin **değerini** döndürmek: −1 değeriyle karışır; indis döndürülmeli.

5. Geri izlemede seçimi geri almayı (ikinci SWAP, `sec[i] = 0`) unutmak.

6. Üstel algoritmayı "makine hızlı" diye büyük n ile çalıştırmak.

Alıştırmalar

1. Dizide en büyük öğeyi bulan özyinelemeli yordamı yazınız. Yığın derinliği ne olur?
2. `binsearch` 1000 öğeli dizide en çok kaç tur döner? Aranan yoksa?
3. hanoi(4, …) kaç kez `printf` çağırır, kaç kez kendini çağırır?
4. perm("abcd", 0, 4) çıktısının ilk 6 satırını yazınız.
5. (x1 ∨ x2) ∧ (¬x1 ∨ ¬x2) formülünü sağlayan atamaları kaba kuvvetle bulunuz.
6. 5-vezir için geri izleme ile ilk çözümü bulunuz (q dizisi).

▶ **Cevapları / ipuçlarını göster**

ÖZET

Bölüm Özeti

Özyineleme

Taban durum + küçülen adım. Her çağrı bir yığın çerçevesi; kuyruk özyinelemesi döngüye çevrilebilir.

İkili arama

Sıralı dizide aralığı yarılar: O(log n). İndis döndürülür, −1 hata demektir.

Hanoi

$h(n) = 2h(n-1) + 1 = 2^n - 1$. Çağrı ağacı = yığının tarihçesi.

Permütasyon

Takas + özyineleme + geri takas: n! çıktı, O(n·n!).

Geri izleme

Kısmi çözüm kurulur, kural bozulunca budanır ve geri alınır: SAT, n-vezir, alt küme.

Sezgisel

Warnsdorff: en az ilerleme sayılı kare. Daha iyi çözüm aramaktan vazgeçilmez.

ÖZET

Kaynaklar

- Horowitz, Sahni, Anderson-Freed, *Fundamentals of Data Structures in C*, 2. baskı, Bölüm 1.2 (özyinelemeli algoritmalar).
- Cormen ve diğ., *Introduction to Algorithms*, 3. baskı: Bölüm 4 (yineleme bağıntıları), Bölüm 34 (NP-tamlık, SAT).
- Sedgewick, *Algorithms in C*, Bölüm 5 (özyineleme).

← Bölüm 2: Diziler ve Yapılar

Ders sayfası

Bölüm 4: Bağlı Listeler

Son

Bölüm 3: Özyineleme, İkili Arama ve Geri İzleme

Sonraki bölüm: Bölüm 4: Bağlı Listeler · Ders sayfası