Bölüm 7: Kuyruklar

C ile Veri Yapıları ve Algoritmalar

FIFO · Kuyruk ADT · Doğrusal kuyruk · Dairesel kuyruk · Bağlı kuyruk · Deque · Zamanlama · BFS

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

FIFO'yu tanımlamak

Kuyruğu $Q = \{a_0, \ldots, a_{n-1}\}$ olarak tanımlayıp ön ve arka uçları göstermek.

Doğrusal kuyruğun sorununu görmek

Önde boşalan hücrelerin neden kullanılamadığını ve kaydırmanın maliyetini açıklamak.

Dairesel kuyruk yazmak

`% MAX` ile sarma yapıp dolu/boş ayrımını üç yöntemle çözmek.

Bağlı kuyruk kurmak

`on` ve `arka` işaretçileriyle O(1) ekleme/çıkarma yapmak.

Deque'yi tanımak

İki uçtan ekleme ve çıkarma yapan çift uçlu kuyruğu dairesel dizide gerçeklemek.

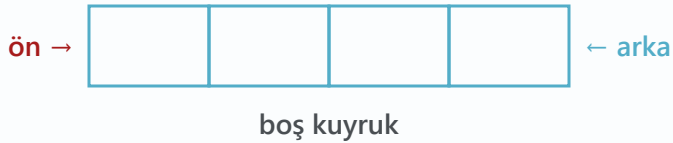
Uygulamak

Zaman dilimli iş zamanlama benzetiminde kuyruğu kullanmak; iki yığınla kuyruk kurmak.

KUYRUK ADT

Kuyruk (Queue): İlk Giren İlk Çıkar

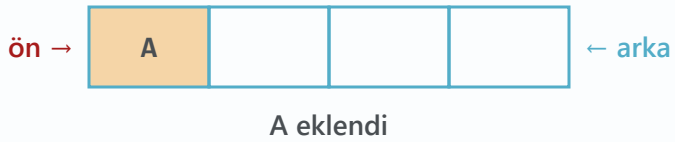
Q = {a_0, a_1, …, a_{n-1}} kuyruğunda, yığından farklı olarak işlenecek ilk öğe a_0'dır (**ön**, front); a_{n-1} kuyruğun **arka**sında (rear) yer alır. Ekleme arkadan, silme önden yapılır: **FIFO** (first-in-first-out).



Boş kuyruk: ön ve arka aynı yerde.

Kuyruk (Queue): İlk Giren İlk Çıkar

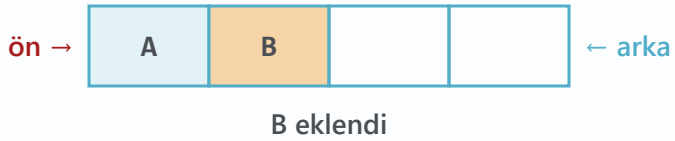
Q = {a_0, a_1, …, a_{n-1}} kuyruğunda, yığından farklı olarak işlenecek ilk öğe a_0'dır (**ön**, front); a_{n-1} kuyruğun **arka**sında (rear) yer alır. Ekleme arkadan, silme önden yapılır: **FIFO** (first-in-first-out).



A arkaya eklendi.

Kuyruk (Queue): İlk Giren İlk Çıkar

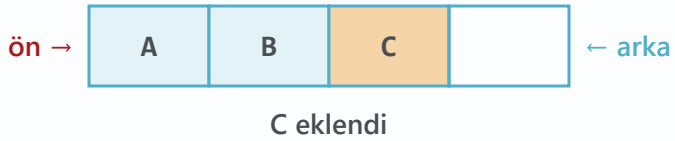
Q = {a_0, a_1, …, a_{n-1}} kuyruğunda, yığından farklı olarak işlenecek ilk öğe a_0'dır (**ön**, front); a_{n-1} kuyruğun **arka**sında (rear) yer alır. Ekleme arkadan, silme önden yapılır: **FIFO** (first-in-first-out).



B arkaya eklendi.

Kuyruk (Queue): İlk Giren İlk Çıkar

Q = {a_0, a_1, …, a_{n-1}} kuyruğunda, yığından farklı olarak işlenecek ilk öğe a_0'dır (**ön**, front); a_{n-1} kuyruğun **arka**sında (rear) yer alır. Ekleme arkadan, silme önden yapılır: **FIFO** (first-in-first-out).

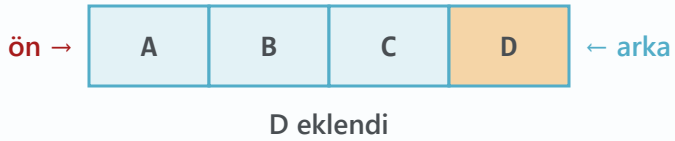


C arkaya eklendi.

KUYRUK ADT

Kuyruk (Queue): İlk Giren İlk Çıkar

Q = {a_0, a_1, …, a_{n-1}} kuyruğunda, yığından farklı olarak işlenecek ilk öğe a_0'dır (**ön**, front); a_{n-1} kuyruğun **arka**sında (rear) yer alır. Ekleme arkadan, silme önden yapılır: **FIFO** (first-in-first-out).

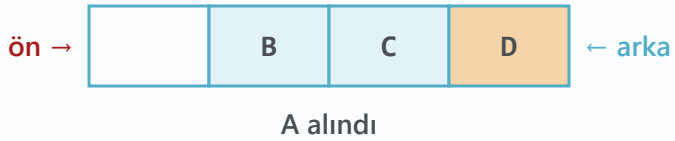


D arkaya eklendi.

Adım 5 / 6

Kuyruk (Queue): İlk Giren İlk Çıkar

Q = {a_0, a_1, …, a_{n-1}} kuyruğunda, yığından farklı olarak işlenecek ilk öğe a_0'dır (**ön**, front); a_{n-1} kuyruğun **arka**sında (rear) yer alır. Ekleme arkadan, silme önden yapılır: **FIFO** (first-in-first-out).



İlk alınan, ilk eklenen **A** olur: **FIFO**. Öndeki hücre boşaldı.

KUYRUK ADT

Soyut Veri Tipi: Kuyruk

İşlem	Anlamı	Hata durumu
`kuyruk_olustur(max)`	En çok max öğelik boş kuyruk	—
`kuyruk_dolu(Q)`	Yer kalmadıysa doğru	—
`kuyruk_ekle(Q, x)` · enqueue	x'i **arkaya** ekler	Dolu ise taşma
`kuyruk_bos(Q)`	Öğe yoksa doğru	—
`kuyruk_al(Q)` · dequeue	**Öndeki** öğeyi çıkarıp döndürür	Boş ise alt taşma
`kuyruk_on(Q)` · peek	Öndekini çıkarmadan okur	Boş ise hata

Yığın

Tek uç: ekle ve al **üstte**. LIFO.

Kuyruk

İki uç: ekle **arkada**, al **önde**. FIFO.

Kuyruğun Kullanıldığı Yerler

İşletim sistemi

Çalışmaya hazır süreçler, yazıcı işleri, disk istekleri sırayla kuyrukta bekler.

Tampon (buffer)

Klavye girdisi, ağ paketleri, ses akışı: üretici ile tüketici arasında dairesel tampon.

Genişlik öncelikli arama

BFS ve ağacın düzey sırası gezinmesi: işlenen düğümün çocukları arkaya eklenir, sıradaki düğüm önden alınır.

Benzetim (simülasyon)

Banka gişesi, trafik ışığı, iş zamanlama: gelen müşteri sıraya girer.

Mesaj kuyrukları

Sunucular arasında işler sırayla ve kaybolmadan aktarılır.

Öncelikli kuyruk

“Kuyruk” adını taşır ama FIFO değildir: en öncelikli öğe önce çıkar.

KUYRUK ADT

Arkadan Ekle, Önden Al



Çıkan sıra, giren sırayla **aynıdır**. Yığında tam tersiydi.

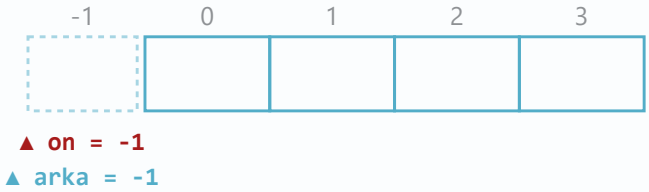
Dizi ile Doğrusal Kuyruk

```c
#define MAX_QUEUE_SIZE 100
element kuyruk[MAX_QUEUE_SIZE];
int on = -1;        /* ilk öğenin bir öncesi */
int arka = -1;      /* son öğe */

void kuyruk_ekle(int *arka, element item) {
    if (*arka >= MAX_QUEUE_SIZE - 1)   /* yer var mı? */
        kuyruk_dolu();          /* uyar ve çık */
    kuyruk[++*arka] = item;     /* önce artır, sonra yaz */
}
element kuyruk_al(int *on, int arka) {
    if (*on == arka)
        kuyruk_bos();           /* boşsa uyar ve çık */
    return kuyruk[++*on];       /* artır ve öndekini döndür */
}
```

- `on` ilk öğeyi değil, **bir öncesini** gösterir; `arka` son öğeyi gösterir.
- Boş: **on == arka** (başta ikisi de −1).
- Dolu: **arka == MAX − 1**, önde boş yer olsa bile.
- `kuyruk_al` `arka`'yı değiştirmez; bu yüzden onu **değerle** alır.
- `kuyruk_dolu` / `kuyruk_bos`: yığındaki gibi mesaj + `exit`.

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



Başta **on = arka = −1**: kuyruk boş. MAX_QUEUE_SIZE = 4. Sırayla ekle A, B, C; iki kez al; ekle D, E.

Adım 1 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

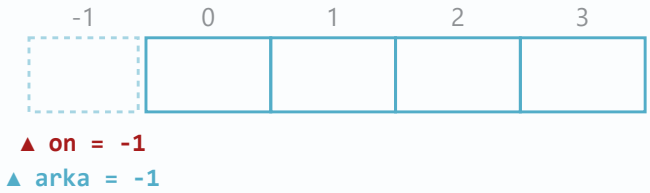
on = **−1** arka = **−1**

ÇAĞRI YIĞINI

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



-1 ≥ 3? Hayır.

Adım 2 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
▶    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

on = **-1** | *arka = **-1** | item = **A**

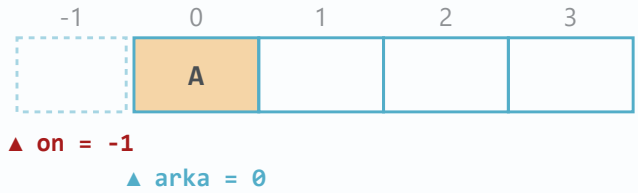
ÇAĞRI YIĞINI

kuyruk_ekle(&arka, A)

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



++*arka → 0; **kuyruk[0] = A**.

Adım 3 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
▶    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

on = **-1** | ***arka = 0** | item = **A**

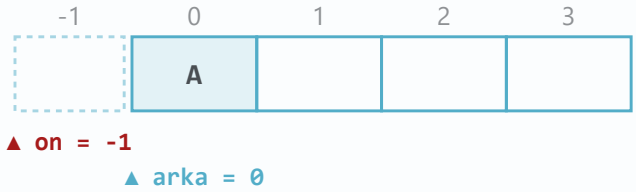
ÇAĞRI YIĞINI

kuyruk_ekle(&arka, A)

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



0 ≥ 3? Hayır.

Adım 4 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
▶    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

on = **-1** | *arka = **0** | item = **B**

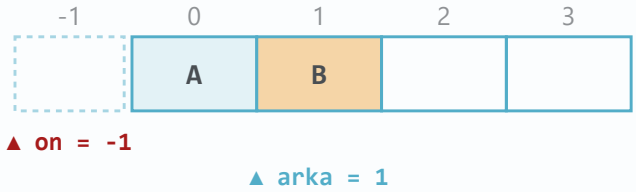
ÇAĞRI YIĞINI

kuyruk_ekle(&arka, B)

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



++*arka → 1; **kuyruk[1] = B**.

Adım 5 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
▶    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

`on = -1` `*arka = 1` `item = B`

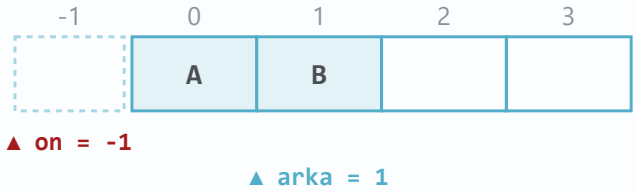
ÇAĞRI YIĞINI

`kuyruk_ekle(&arka, B)`

`main()`

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



1 ≥ 3? Hayır.

Adım 6 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
▶    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

on = **-1** | *arka = **1** | item = **C**

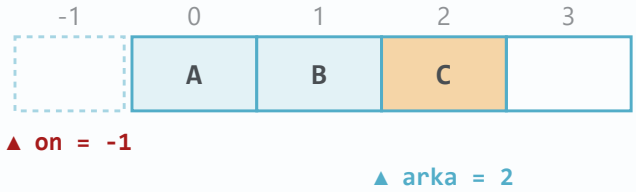
ÇAĞRI YIĞINI

kuyruk_ekle(&arka, C)

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



++*arka → 2; **kuyruk[2] = C**.

Adım 7 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
▶    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

on = **-1** | *arka = **2** | item = **C**

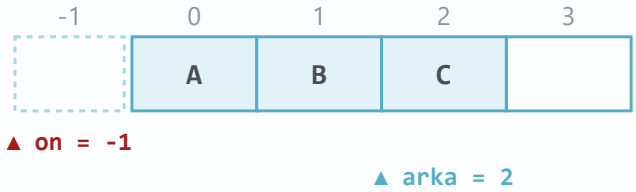
ÇAĞRI YIĞINI

kuyruk_ekle(&arka, C)

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



on == arka (-1 == 2)? Hayır, boş değil.

Adım 8 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
▶    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

`*on = -1` `arka = 2`

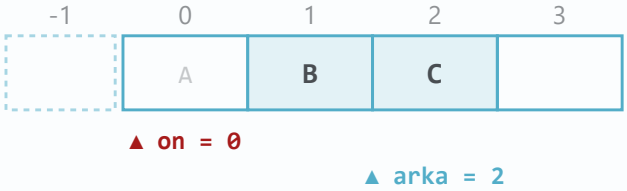
ÇAĞRI YIĞINI

`kuyruk_al(&on, arka)`

`main()`

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



++*on → 0; kuyruk[0] = **A** döner. Hücre artık kullanılmıyor.

Adım 9 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
▶    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

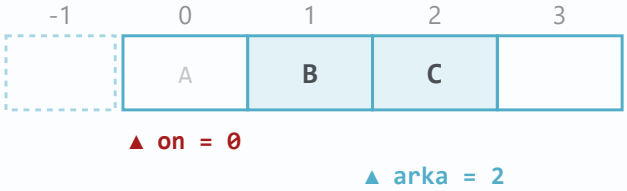
*on = 0 | arka = 2 | donen = A

ÇAĞRI YIĞINI

kuyruk_al(&on, arka)
main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



on == arka (0 == 2)? Hayır, boş değil.

Adım 10 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
▶    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

`*on = 0` `arka = 2`

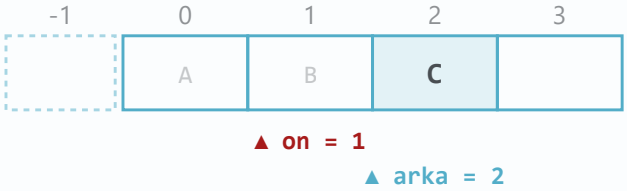
ÇAĞRI YIĞINI

`kuyruk_al(&on, arka)`

`main()`

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



++*on → 1; kuyruk[1] = **B** döner. Hücre artık kullanılmıyor.

Adım 11 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
▶    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

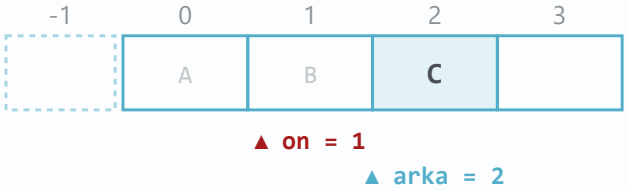
*on = 1 | arka = 2 | donen = B

ÇAĞRI YIĞINI

kuyruk_al(&on, arka)
main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



2 ≥ 3? Hayır.

Adım 12 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
▶    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

`on = 1` `*arka = 2` `item = D`

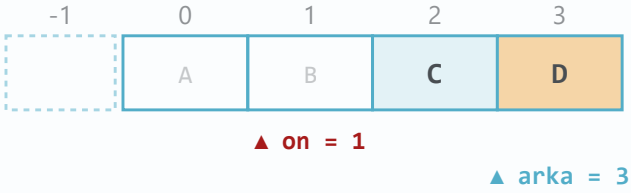
ÇAĞRI YIĞINI

`kuyruk_ekle(&arka, D)`

`main()`

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



++*arka → 3; **kuyruk[3] = D**.

Adım 13 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
▶    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

on = **1** | *arka = **3** | item = **D**

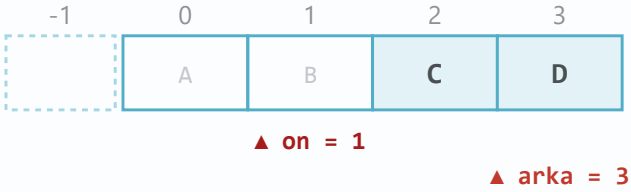
ÇAĞRI YIĞINI

kuyruk_ekle(&arka, D)

main()

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



3 ≥ 3 → dolu sayılır! Oysa 0 ve 1 numaralı hücreler boş.

Adım 14 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
▶    if (*arka >= MAX_QUEUE_SIZE - 1)
3      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

`on = 1` `*arka = 3` `item = E`

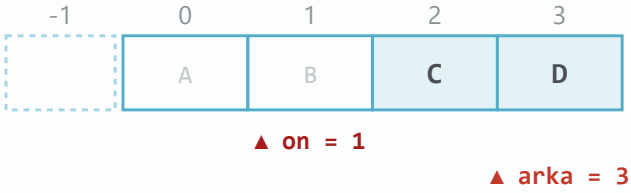
ÇAĞRI YIĞINI

`kuyruk_ekle(&arka, E)`

`main()`

ÇIKTI

Kod İzleme: Doğrusal Kuyruk ve “Sahte Doluluk”



`kuyruk_dolu()` : program sonlanır. **Doğrusal kuyruğun sorunu**: arka sona dayanınca öndeki boş yerler kullanılamaz.

Adım 15 / 15

```
1  void kuyruk_ekle(int *arka, element item) {
2    if (*arka >= MAX_QUEUE_SIZE - 1)
▶      kuyruk_dolu();   /* "Kuyruk Dolu!!!", exit */
4    kuyruk[++*arka] = item;
5  }
6  element kuyruk_al(int *on, int arka) {
7    if (*on == arka)
8      kuyruk_bos();    /* "Kuyruk Bos!!!", exit */
9    return kuyruk[++*on];
10 }
```

DEĞİŞKENLER

`on = 1` `*arka = 3` `item = E`

ÇAĞRI YIĞINI

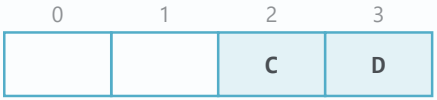
`kuyruk_ekle(&arka, E)`

`main()`

ÇIKTI

```
Kuyruk Dolu!!!
```

Önde Boşalan Yer: Kaydırmak mı?



on = 1, arka = 3: “dolu”

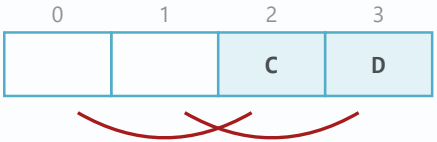
Önde iki boş hücre var ama arka sona dayandı.

Adım 1 / 3

HOCADAN “Kaydırma! Bilgisayar bilimlerinde en kötü kelime! Yüksek maliyet demek!”

Kaydırmalar aşırı zaman alır; çözüm olarak **dairesel kuyruk** geliştirilmiştir: kuyruğun başı ile sonu birleştirilir, önde boşalan yerler arkadaymış gibi kullanılır.

Önde Boşalan Yer: Kaydırmak mı?



her öğeyi başa kaydır

Kaydırma: n öğenin her biri bir kez taşınır → **O(n)**.

HOCADAN "Kaydırma! Bilgisayar bilimlerinde en kötü kelime! Yüksek maliyet demek!"

Kaydırmalar aşırı zaman alır; çözüm olarak **dairesel kuyruk** geliştirilmiştir: kuyruğun başı ile sonu birleştirilir, önde boşalan yerler arkadaymış gibi kullanılır.

Önde Boşalan Yer: Kaydırmak mı?



on = −1, arka = 1

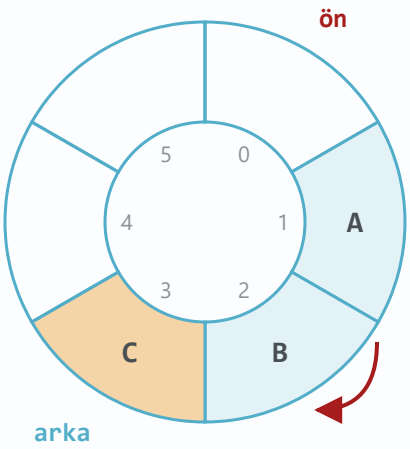
Yer açıldı; ama her “dolu”da bunu yapmak ekleme işlemini O(n) yapar.

Adım 3 / 3

HOCADAN “Kaydırma! Bilgisayar bilimlerinde en kötü kelime! Yüksek maliyet demek!”

Kaydırmalar aşırı zaman alır; çözüm olarak **dairesel kuyruk** geliştirilmiştir: kuyruğun başı ile sonu birleştirilir, önde boşalan yerler arkadaymış gibi kullanılır.

Dairesel Kuyruk



- Dizinin sonu başına bağlanmış gibi düşünülür: 5'ten sonra 0 gelir.
- İlerleme: `arka = (arka + 1) % MAX`, `on = (on + 1) % MAX`.
- `on` yine ilk öğenin **bir öncesini** gösterir; başta `on = arka = 0`.
- Boş: **on == arka**. Dolu: **(arka + 1) % MAX == on**.
- Bu yüzden MAX hücrenin **biri boş kalır**: en çok MAX − 1 öğe.

Sarma (wrap-around): `% MAX`

arka	arka + 1	(arka + 1) % 6	Anlamı
3	4	4	normal ilerleme
4	5	5	son hücre
5	6	**0**	**sarma**: başa dön
0	1	1	yeniden normal

Geri gitmek

Bir geri adım: `(i - 1 + MAX) % MAX`. C'de `-1 % 6` −1 verir; `+ MAX` eklemek negatif sonucu önler (deque bunu kullanır).

Öğe sayısı

`(arka - on + MAX) % MAX`. Örnek: on = 4, arka = 1, MAX = 6 → (1 − 4 + 6) % 6 = 3.

Dairesel Kuyruk Şekli: A…F ile Sarma



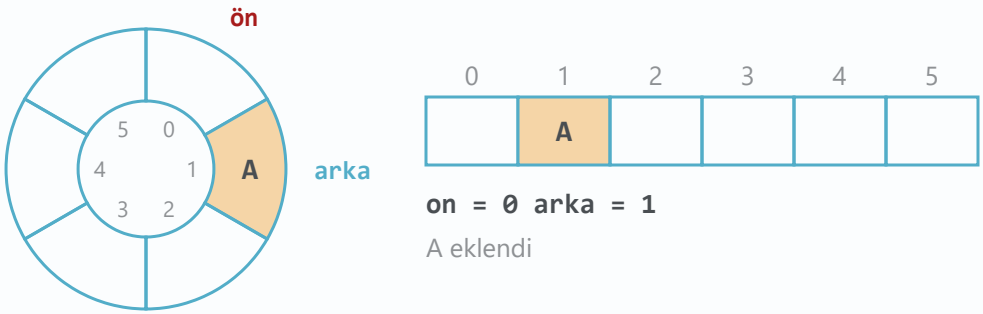


`on = 0 arka = 0`

başlangıç: on = arka = 0

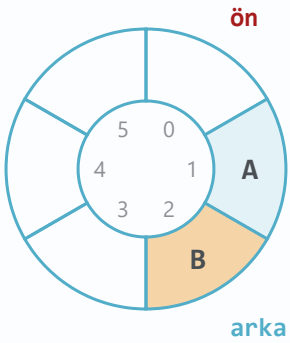
Boş kuyruk: **on == arka** = 0.

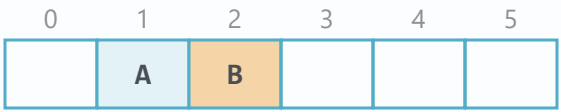
Dairesel Kuyruk Şekli: A...F ile Sarma



A eklendi: arka = 1.

Dairesel Kuyruk Şekli: A...F ile Sarma



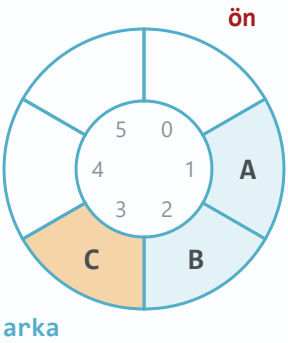


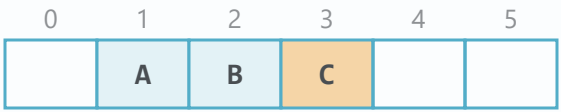
`on = 0 arka = 2`

B eklendi

B eklendi: arka = 2.

Dairesel Kuyruk Şekli: A...F ile Sarma



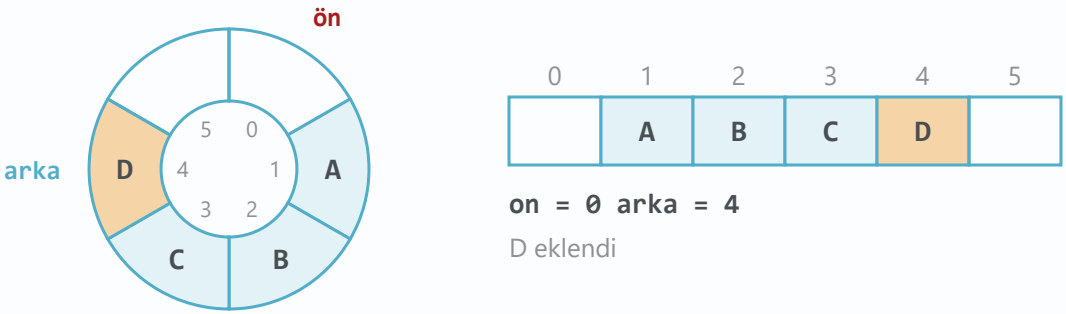


`on = 0 arka = 3`

C eklendi

C eklendi: arka = 3.

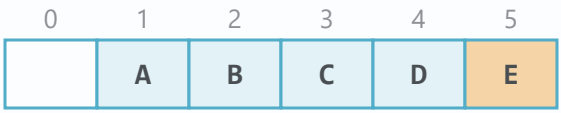
Dairesel Kuyruk Şekli: A...F ile Sarma



D eklendi: arka = 4.

Dairesel Kuyruk Şekli: A...F ile Sarma



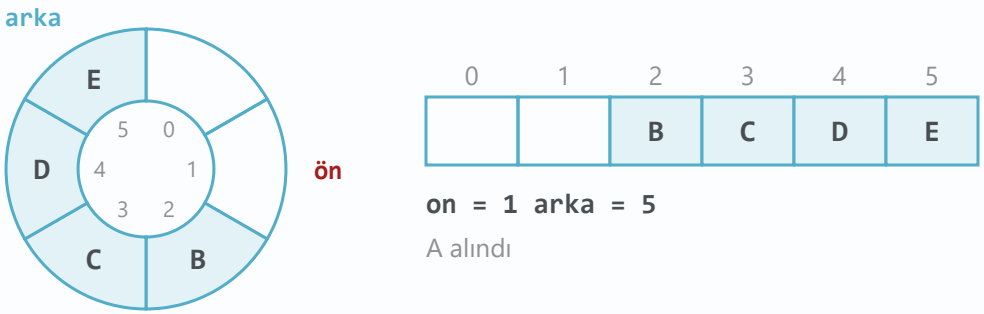


`on = 0 arka = 5`

E eklendi

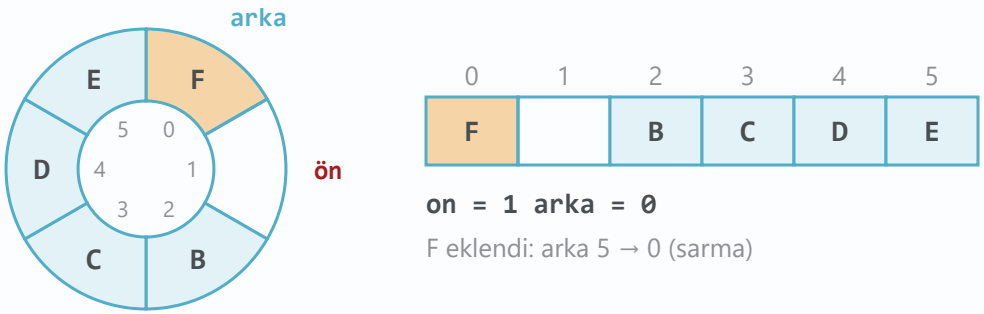
E eklendi: arka = 5.

Dairesel Kuyruk Şekli: A...F ile Sarma



A alındı: on = 1. Artık 0 ve 1 numaralı hücrelerin ikisi de boş.

Dairesel Kuyruk Şekli: A…F ile Sarma



F, arka = (5 + 1) % 6 = **0**'a yazıldı: önde boşalan yer arkadaymış gibi kullanıldı. Kuyruk şimdi **dolu**: (0 + 1) % 6 == on. 6 hücrede 5 öğe.

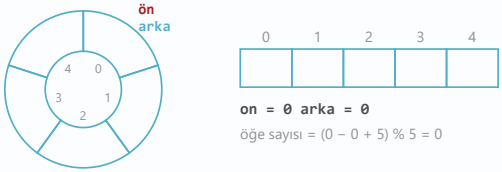
Dairesel Kuyruk: C Kodu

```c
void kuyruk_ekle(int on, int *arka, element item) {
    *arka = (*arka + 1) % MAX_QUEUE_SIZE; /* ilerlet */
    if (on == *arka) kuyruk_dolu();       /* dolu mu? */
    kuyruk[*arka] = item;                 /* yerleştir */
}
element kuyruk_al(int *on, int arka) {
    if (*on == arka) kuyruk_bos();        /* boş mu? */
    *on = (*on + 1) % MAX_QUEUE_SIZE;     /* ilerlet */
    return kuyruk[*on];                   /* döndür */
}
```

İnce bir nokta: `kuyruk_ekle` önce `*arka`'yı ilerletip **sonra** doluluğa bakar. `kuyruk_dolu` programı bitirdiği için sorun olmaz; ama hata döndüren bir sürümde `arka` bozulur (on'a eşit kalır ve kuyruk **boş** görünür). Güvenli sürümde yeni konum önce geçici bir değişkende denenir (Horowitz–Sahni de `rear`'ı geri alır).

HOCADAN "Dünyadaki bilgisayar mühendisliği alanında en iyi üniversite hangisi? Harvard mı? Öyleyse "Harvard Üniversitesi'nde okuyorum da, eğer başaramazsam atılacağım. Daha kötü bir okula gitmek zorunda kalacağım." diye okumak lazım."

Kod İzleme: Güvenli Dairesel Kuyruk



MAX_QUEUE_SIZE = 5, **on = arka = 0**. Ekle A, B, C, D, E; al, al; ekle E, F, G.

Adım 1 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

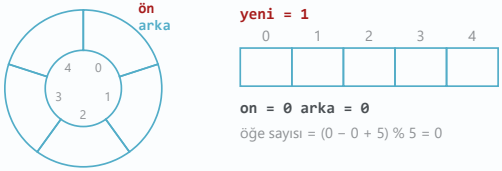
`on = 0` `arka = 0`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



yeni = (0 + 1) % 5 = **1**; 1 ≠ on, yer var.

Adım 2 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 0` `*arka = 0` `yeni = 1` `x = A`

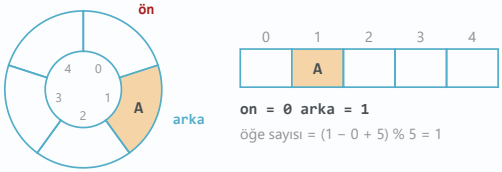
ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, A)`

`main()`

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



arka = 1, kuyruk[1] = A.

Adım 3 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
▶    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

on = 0 | *arka = 1 | yeni = 1 | x = A

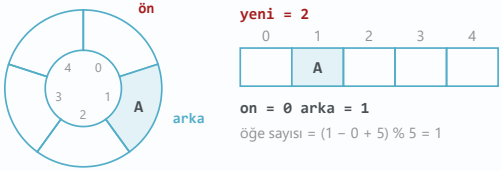
ÇAĞRI YIĞINI

kuyruk_ekle(on, &arka, A)

main()

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



yeni = (1 + 1) % 5 = **2**; 2 ≠ on, yer var.

Adım 4 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 0` `*arka = 1` `yeni = 2` `x = B`

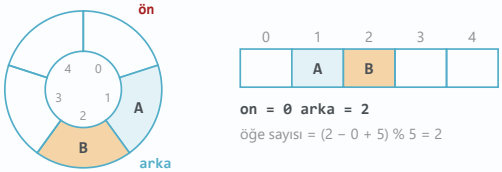
ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, B)`

`main()`

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



arka = 2, kuyruk[2] = B.

Adım 5 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
▶    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

on = 0 | *arka = 2 | yeni = 2 | x = B

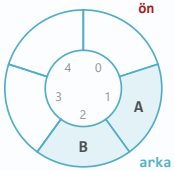
ÇAĞRI YIĞINI

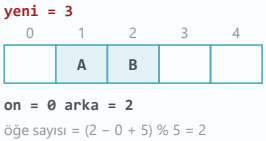
kuyruk_ekle(on, &arka, B)

main()

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk





yeni = (2 + 1) % 5 = **3**; 3 ≠ on, yer var.

Adım 6 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 0` `*arka = 2` `yeni = 3` `x = C`

ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, C)`

`main()`

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



arka = 3, kuyruk[3] = C.

Adım 7 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
▶    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

on = 0 | *arka = 3 | yeni = 3 | x = C

ÇAĞRI YIĞINI

kuyruk_ekle(on, &arka, C)
main()

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



yeni = (3 + 1) % 5 = **4**; 4 ≠ on, yer var.

Adım 8 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 0` `*arka = 3` `yeni = 4` `x = D`

ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, D)`

`main()`

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



arka = 4, kuyruk[4] = D.

Adım 9 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
▶    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

on = 0 | *arka = 4 | yeni = 4 | x = D

ÇAĞRI YIĞINI

kuyruk_ekle(on, &arka, D)
main()

ÇIKTI

Kod İzleme: Güvenli Dairesel Kuyruk



yeni = (4 + 1) % 5 = 0 == on → **dolu**, false döner. `arka` 4 olarak **korunur**. 5 hücrede 4 öğe.

Adım 10 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 0` `*arka = 4` `yeni = 0` `x = E`

ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, E)`

`main()`

ÇIKTI

```
E: dolu
```

Kod İzleme: Güvenli Dairesel Kuyruk



on == arka (0 == 4)? Hayır.

Adım 11 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
▶    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`*on = 0` `arka = 4`

ÇAĞRI YIĞINI

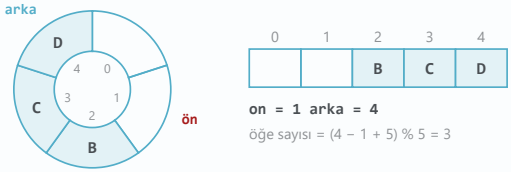
`kuyruk_al(&on, arka, &x)`

`main()`

ÇIKTI

```
E: dolu
```

Kod İzleme: Güvenli Dairesel Kuyruk



on = 1, ***x = kuyruk[1] = A**.

Adım 12 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
▶    *on = (*on + 1) % MAX_QUEUE_SIZE;
▶    *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

*on = 1 | arka = 4 | *x = A

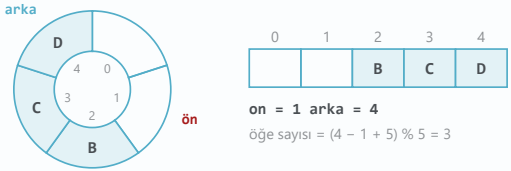
ÇAĞRI YIĞINI

kuyruk_al(&on, arka, &x)
main()

ÇIKTI

```
E: dolu  al: A
```

Kod İzleme: Güvenli Dairesel Kuyruk



on == arka (1 == 4)? Hayır.

Adım 13 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
▶    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`*on = 1` `arka = 4`

ÇAĞRI YIĞINI

`kuyruk_al(&on, arka, &x)`

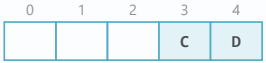
`main()`

ÇIKTI

```
E: dolu  al: A
```

Kod İzleme: Güvenli Dairesel Kuyruk





on = 2 arka = 4

öğe sayısı = (4 − 2 + 5) % 5 = 2

on = 2, ***x = kuyruk[2] = B**.

Adım 14 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
▶    *on = (*on + 1) % MAX_QUEUE_SIZE;
▶    *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

*on = 2 | arka = 4 | *x = B

ÇAĞRI YIĞINI

kuyruk_al(&on, arka, &x)

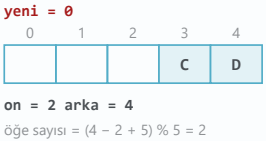
main()

ÇIKTI

```
E: dolu  al: A  al: B
```

Kod İzleme: Güvenli Dairesel Kuyruk





yeni = (4 + 1) % 5 = **0** (sarma!); 0 ≠ on, yer var.

Adım 15 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 2` `*arka = 4` `yeni = 0` `x = E`

ÇAĞRI YIĞINI

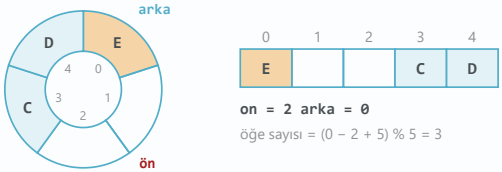
`kuyruk_ekle(on, &arka, E)`

`main()`

ÇIKTI

```
E: dolu  al: A  al: B
```

Kod İzleme: Güvenli Dairesel Kuyruk



arka = 0, kuyruk[0] = E.

Adım 16 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
▶    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

on = **2** | *arka = **0** | yeni = **0** | x = **E**

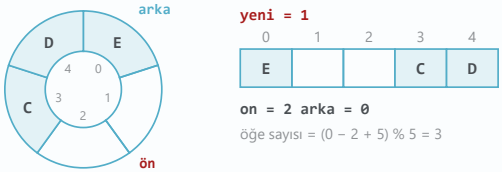
ÇAĞRI YIĞINI

kuyruk_ekle(on, &arka, E)
main()

ÇIKTI

```
E: dolu  al: A  al: B
```

Kod İzleme: Güvenli Dairesel Kuyruk



yeni = (0 + 1) % 5 = **1**; 1 ≠ on, yer var.

Adım 17 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 2` `*arka = 0` `yeni = 1` `x = F`

ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, F)`

`main()`

ÇIKTI

```
E: dolu  al: A  al: B
```

Kod İzleme: Güvenli Dairesel Kuyruk



arka = 1, kuyruk[1] = F.

Adım 18 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
2    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
3    if (yeni == on) return false;        /* dolu */
▶    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;       /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

on = **2** | *arka = **1** | yeni = **1** | x = **F**

ÇAĞRI YIĞINI

kuyruk_ekle(on, &arka, F)

main()

ÇIKTI

```
E: dolu  al: A  al: B
```

Kod İzleme: Güvenli Dairesel Kuyruk



yeni = (1 + 1) % 5 = 2 == on → **dolu**, false döner. `arka` 1 olarak **korunur**. 5 hücrede 4 öğe.

Adım 19 / 19

```
1  bool kuyruk_ekle(int on, int *arka, element x) {
▶    int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
▶    if (yeni == on) return false;       /* dolu */
4    *arka = yeni;  kuyruk[yeni] = x;
5    return true;
6  }
7  bool kuyruk_al(int *on, int arka, element *x) {
8    if (*on == arka) return false;      /* boş */
9    *on = (*on + 1) % MAX_QUEUE_SIZE;
10   *x = kuyruk[*on];  return true;
11 }
```

DEĞİŞKENLER

`on = 2` `*arka = 1` `yeni = 2` `x = G`

ÇAĞRI YIĞINI

`kuyruk_ekle(on, &arka, G)`

`main()`

ÇIKTI

```
E: dolu  al: A  al: B  G: dolu
```

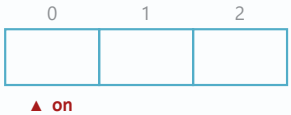
Dolu/Boş Ayrımı: Üç Yöntem

Yöntem	Boş	Dolu	Kapasite	Bedeli
1. Bir hücre boş bırakmak	`on == arka`	`(arka+1)%MAX == on`	MAX − 1	Bir hücre israf
2. Sayaç tutmak	`sayac == 0`	`sayac == MAX`	MAX	Her işlemde sayaç güncellemesi
3. Bayrak tutmak	`on == arka` ve son işlem al	`on == arka` ve son işlem ekle	MAX	Her işlemde bayrak

Sorunun kökü: yalnız `on` ve `arka` ile MAX + 1 farklı doluluk durumu (0, 1, …, MAX öğe) ayırt edilmek istenir, ama iki indisin farkı yalnız MAX farklı değer alabilir. Ya bir durumdan vazgeçilir (yöntem 1) ya da ek bilgi tutulur (yöntem 2, 3).

Sayaçlı sürüm ayrıca öğe sayısını O(1)'de verir ve `arka`'yı hiç tutmaz: `arka = (on + sayac − 1) % KAP`.

Kod İzleme: Sayaçlı Dairesel Kuyruk



sayac = 0
KAP = 3

KAP = 3, on = 0, sayac = 0. Ekle 1, 2, 3, 4; al; ekle 4; al.

Adım 1 / 10

```
1  bool ekle(Kuyruk *k, int x) {
2    if (k->sayac == KAP) return false;        /* dolu */
3    k->veri[(k->on + k->sayac) % KAP] = x;
4    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

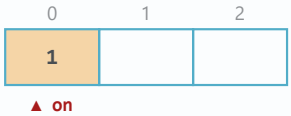
`k.on = 0` `k.sayac = 0`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Sayaçlı Dairesel Kuyruk



Yer var; konum = (on + sayac) % 3 = **0**, veri[0] = 1; sayac = 1.

Adım 2 / 10

```
1  bool ekle(Kuyruk *k, int x) {
▶    if (k->sayac == KAP) return false;        /* dolu */
▶    k->veri[(k->on + k->sayac) % KAP] = x;
▶    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

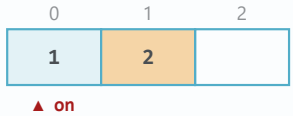
k→on = **0** | k→sayac = **1** | x = **1**

ÇAĞRI YIĞINI

- ekle(&k, 1)
- main()

ÇIKTI

Kod İzleme: Sayaçlı Dairesel Kuyruk



Yer var; konum = (on + sayac) % 3 = **1**, veri[1] = 2; sayac = 2.

Adım 3 / 10

```
1  bool ekle(Kuyruk *k, int x) {
▶    if (k->sayac == KAP) return false;        /* dolu */
▶    k->veri[(k->on + k->sayac) % KAP] = x;
▶    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

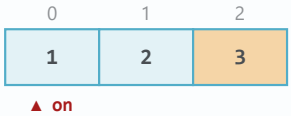
k→on = **0** | k→sayac = **2** | x = **2**

ÇAĞRI YIĞINI

- **ekle(&k, 2)**
- main()

ÇIKTI

Kod İzleme: Sayaçlı Dairesel Kuyruk



Yer var; konum = (on + sayac) % 3 = **2**, veri[2] = 3; sayac = 3.

Adım 4 / 10

```
1  bool ekle(Kuyruk *k, int x) {
▶    if (k->sayac == KAP) return false;        /* dolu */
▶    k->veri[(k->on + k->sayac) % KAP] = x;
▶    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

k→on = **0** | k→sayac = **3** | x = **3**

ÇAĞRI YIĞINI

- **ekle(&k, 3)**
- main()

ÇIKTI

Kod İzleme: Sayaçlı Dairesel Kuyruk



sayac == KAP → dolu. Üç hücrenin **üçü de** dolu: hiç hücre israf edilmedi.

Adım 5 / 10

```
1  bool ekle(Kuyruk *k, int x) {
▶    if (k->sayac == KAP) return false;        /* dolu */
3    k->veri[(k->on + k->sayac) % KAP] = x;
4    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

`k→on = 0` `k→sayac = 3` `x = 4`

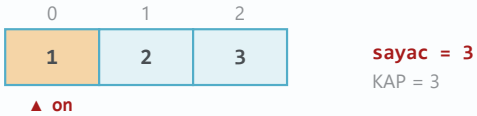
ÇAĞRI YIĞINI

- `ekle(&k, 4)`
- `main()`

ÇIKTI

```
ekle 4: dolu
```

Kod İzleme: Sayaçlı Dairesel Kuyruk



Boş değil; ***x = veri[0] = 1**.

Adım 6 / 10

```
1  bool ekle(Kuyruk *k, int x) {
2    if (k->sayac == KAP) return false;        /* dolu */
3    k->veri[(k->on + k->sayac) % KAP] = x;
4    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
▶    if (k->sayac == 0) return false;          /* boş */
▶    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

`k→on = 0` `k→sayac = 3` `*x = 1`

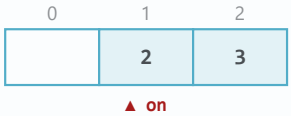
ÇAĞRI YIĞINI

- `al(&k, &x)`
- `main()`

ÇIKTI

```
ekle 4: dolu
```

Kod İzleme: Sayaçlı Dairesel Kuyruk



on = 1, sayac = 2.

Adım 7 / 10

```
1  bool ekle(Kuyruk *k, int x) {
2    if (k->sayac == KAP) return false;        /* dolu */
3    k->veri[(k->on + k->sayac) % KAP] = x;
4    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
▶    k->on = (k->on + 1) % KAP;
▶    k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

`k→on = 1` `k→sayac = 2` `*x = 1`

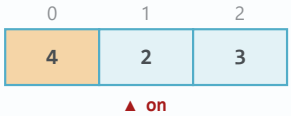
ÇAĞRI YIĞINI

- `al(&k, &x)`
- `main()`

ÇIKTI

```
ekle 4: dolu  al: 1
```

Kod İzleme: Sayaçlı Dairesel Kuyruk



Yer var; konum = (on + sayac) % 3 = **0** (sarma), veri[0] = 4; sayac = 3.

Adım 8 / 10

```
1  bool ekle(Kuyruk *k, int x) {
▶    if (k->sayac == KAP) return false;        /* dolu */
▶    k->veri[(k->on + k->sayac) % KAP] = x;
▶    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

k→on = **1** | k→sayac = **3** | x = **4**

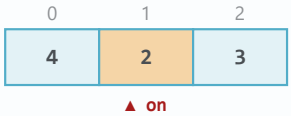
ÇAĞRI YIĞINI

- **ekle(&k, 4)**
- main()

ÇIKTI

```
ekle 4: dolu  al: 1
```

Kod İzleme: Sayaçlı Dairesel Kuyruk



Boş değil; ***x = veri[1] = 2**.

Adım 9 / 10

```
1  bool ekle(Kuyruk *k, int x) {
2    if (k->sayac == KAP) return false;        /* dolu */
3    k->veri[(k->on + k->sayac) % KAP] = x;
4    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
▶    if (k->sayac == 0) return false;          /* boş */
▶    *x = k->veri[k->on];
10   k->on = (k->on + 1) % KAP;
11   k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

`k→on = 1` `k→sayac = 3` `*x = 2`

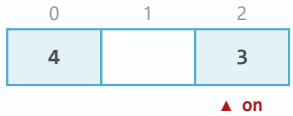
ÇAĞRI YIĞINI

- `al(&k, &x)`
- `main()`

ÇIKTI

```
ekle 4: dolu  al: 1
```

Kod İzleme: Sayaçlı Dairesel Kuyruk



on = 2, sayac = 2.

Adım 10 / 10

```
1  bool ekle(Kuyruk *k, int x) {
2    if (k->sayac == KAP) return false;        /* dolu */
3    k->veri[(k->on + k->sayac) % KAP] = x;
4    k->sayac++;
5    return true;
6  }
7  bool al(Kuyruk *k, int *x) {
8    if (k->sayac == 0) return false;          /* boş */
9    *x = k->veri[k->on];
▶    k->on = (k->on + 1) % KAP;
▶    k->sayac--;
12   return true;
13 }
```

DEĞİŞKENLER

`k→on = 2` `k→sayac = 2` `*x = 2`

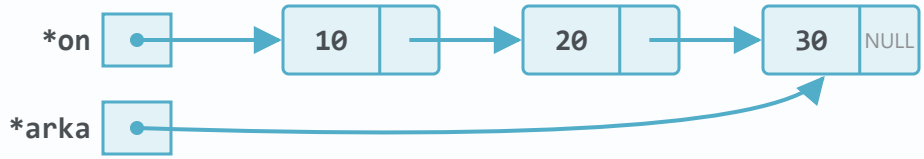
ÇAĞRI YIĞINI

- `al(&k, &x)`
- `main()`

ÇIKTI

```
ekle 4: dolu  al: 1  al: 2
```

Bağlı Liste ile Kuyruk



on ilk düğümü, arka son düğümü gösterir; son düğümün bağı NULL.

- Ekleme **arkaya**: `arka→bag = yeni; arka = yeni;` → O(1).
- Silme **önden**: `on = on→bag;` → O(1).
- Tersi (önden ekle, arkadan sil) olmaz: tek yönlü bağlıda sondan silmek için sondan bir öncekini bulmak O(n).

HOCADAN "– Ben iki derstir ne anlatıyorum? Dersi bıraktım, C anlattım, pointerları anlattım. Çok iyi bilinen pointerları bir de ben anlattım. – Ben pointerları çok iyi biliyorum, ama hoca bir şey diyor, yine de anlamadım. – Olabilir."

Kod İzleme: Bağlı `kuyruk_ekle`

*on NULL

*arka NULL

Boş kuyruk: **on = arka = NULL**. Sırayla ekle 10, 20, 30.

Adım 1 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;   /* boş kuyruk */
6   else (*arka)->bag = yeni;        /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

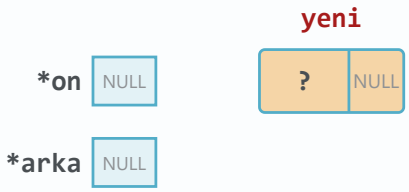
DEĞİŞKENLER

on = NULL arka = NULL

ÇAĞRI YIĞINI

main()

Kod İzleme: Bağlı `kuyruk_ekle`



malloc: yeni düğüm ayrıldı.

Adım 2 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
▶   dugum *yeni = malloc(sizeof *yeni);
▶   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

DEĞİŞKENLER

`*on = NULL` `*arka = NULL` `yeni = düğüm(?)`

ÇAĞRI YIĞINI

`kuyruk_ekle(&on, &arka, 10)`

`main()`

Kod İzleme: Bağlı `kuyruk_ekle`



deger = 10, **bag = NULL** (yeni son düğüm olacak).

Adım 3 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
▶  yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

DEĞİŞKENLER

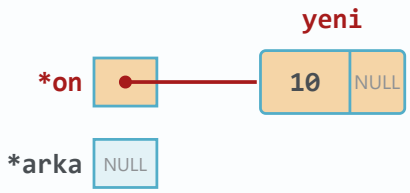
`*on = NULL` `*arka = NULL` `yeni = düğüm(10)`

ÇAĞRI YIĞINI

`kuyruk_ekle(&on, &arka, 10)`

`main()`

Kod İzleme: Bağlı `kuyruk_ekle`



***arka == NULL**: kuyruk boştu, yeni düğüm aynı zamanda **ön** olur: `*on = yeni`.

Adım 4 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
▶   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

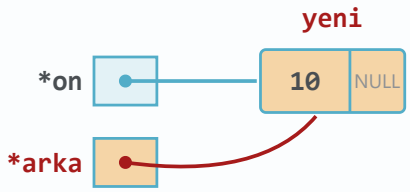
DEĞİŞKENLER

`*on` = **düğüm(10)** | `*arka` = **NULL** | `yeni` = **düğüm(10)**

ÇAĞRI YIĞINI

- **`kuyruk_ekle(&on, &arka, 10)`**
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



***arka = yeni**: arka 10'u gösterir.

Adım 5 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
▶   *arka = yeni;
8   return true;
9 }
```

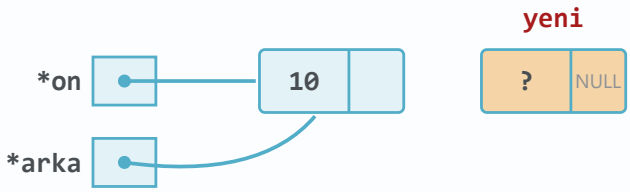
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(10)**
- `yeni` = **düğüm(10)**

ÇAĞRI YIĞINI

- `kuyruk_ekle(&on, &arka, 10)`
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



malloc: yeni düğüm ayrıldı.

Adım 6 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
▶   dugum *yeni = malloc(sizeof *yeni);
▶   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

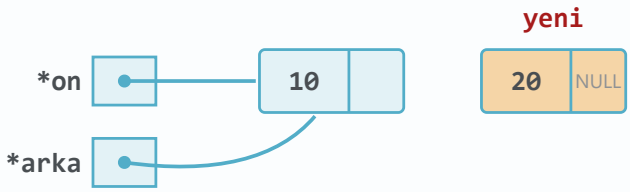
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(10)**
- `yeni` = **düğüm(?)**

ÇAĞRI YIĞINI

- **`kuyruk_ekle(&on, &arka, 20)`**
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



deger = 20, **bag = NULL** (yeni son düğüm olacak).

Adım 7 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
▶   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

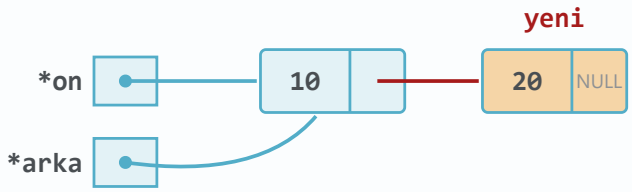
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(10)**
- `yeni` = **düğüm(20)**

ÇAĞRI YIĞINI

- `kuyruk_ekle(&on, &arka, 20)`
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



Kuyruk boş değil: **(*arka)->bag = yeni**, 10 artık 20'yi gösterir.

Adım 8 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
▶   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

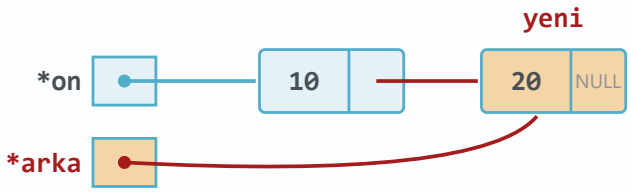
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(10)**
- `yeni` = **düğüm(20)**
- `(*arka)→bag` = **düğüm(20)**

ÇAĞRI YIĞINI

- **`kuyruk_ekle(&on, &arka, 20)`**
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



***arka = yeni**: arka 20'yi gösterir.

Adım 9 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;   /* boş kuyruk */
6   else (*arka)->bag = yeni;        /* sona bağla */
▶   *arka = yeni;
8   return true;
9 }
```

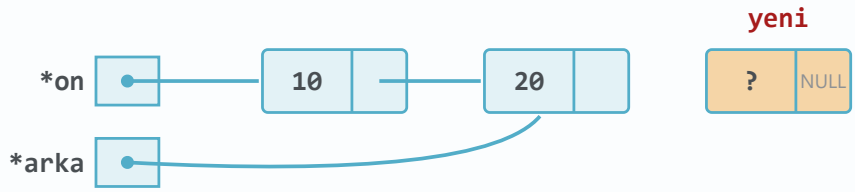
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(20)**
- `yeni` = **düğüm(20)**

ÇAĞRI YIĞINI

- `kuyruk_ekle(&on, &arka, 20)`
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



malloc: yeni düğüm ayrıldı.

Adım 10 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
▶   dugum *yeni = malloc(sizeof *yeni);
▶   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

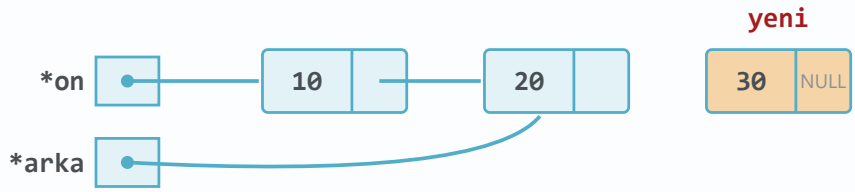
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(20)**
- `yeni` = **düğüm(?)**

ÇAĞRI YIĞINI

- `kuyruk_ekle(&on, &arka, 30)`
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



deger = 30, **bag = NULL** (yeni son düğüm olacak).

Adım 11 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
▶   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
6   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

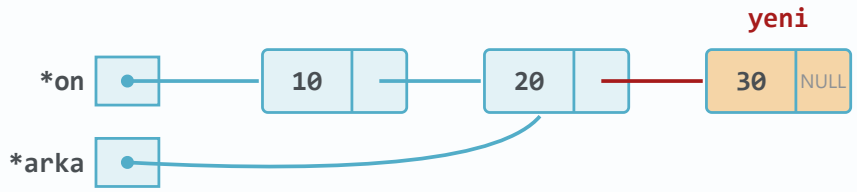
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(20)**
- `yeni` = **düğüm(30)**

ÇAĞRI YIĞINI

- `kuyruk_ekle(&on, &arka, 30)`
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



Kuyruk boş değil: **(*arka)->bag = yeni**, 20 artık 30'u gösterir.

Adım 12 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;    /* boş kuyruk */
▶   else (*arka)->bag = yeni;         /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

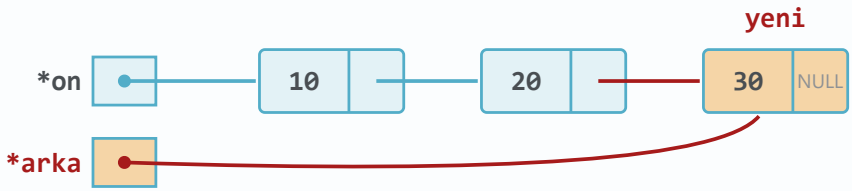
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(20)**
- `yeni` = **düğüm(30)**
- `(*arka)→bag` = **düğüm(30)**

ÇAĞRI YIĞINI

- **`kuyruk_ekle(&on, &arka, 30)`**
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



***arka = yeni**: arka 30'u gösterir.

Adım 13 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;   /* boş kuyruk */
6   else (*arka)->bag = yeni;        /* sona bağla */
▶   *arka = yeni;
8   return true;
9 }
```

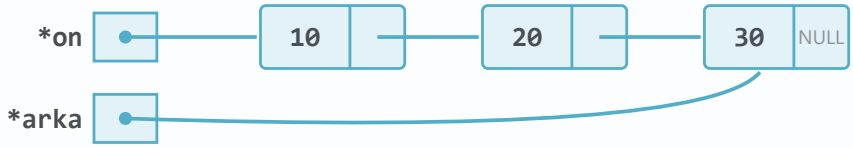
DEĞİŞKENLER

- `*on` = **düğüm(10)**
- `*arka` = **düğüm(30)**
- `yeni` = **düğüm(30)**

ÇAĞRI YIĞINI

- `kuyruk_ekle(&on, &arka, 30)`
- `main()`

Kod İzleme: Bağlı `kuyruk_ekle`



Kuyruk: 10 → 20 → 30. Her ekleme sabit iş: **O(1)**; `arka` olmasaydı sonu bulmak O(n) olurdu.

Adım 14 / 14

```
1 bool kuyruk_ekle(dugum **on, dugum **arka, int x) {
2   dugum *yeni = malloc(sizeof *yeni);
3   if (yeni == NULL) return false;
4   yeni->deger = x;  yeni->bag = NULL;
5   if (*arka == NULL) *on = yeni;   /* boş kuyruk */
6   else (*arka)->bag = yeni;        /* sona bağla */
7   *arka = yeni;
8   return true;
9 }
```

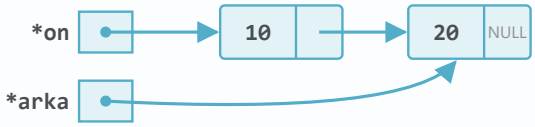
DEĞİŞKENLER

`on` = **düğüm(10)** `arka` = **düğüm(30)**

ÇAĞRI YIĞINI

main()

Kod İzleme: Bağlı `kuyruk_al`



Kuyruk: 10 → 20. Üç kez `kuyruk_al`.

Adım 1 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

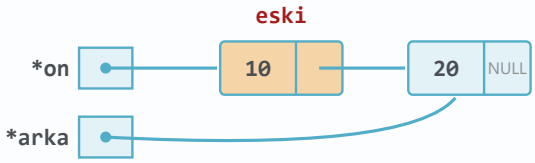
`on` = **düğüm(10)** `arka` = **düğüm(20)**

ÇAĞRI YIĞINI

main()

ÇIKTI

Kod İzleme: Bağlı `kuyruk_al`



eski = *on.

Adım 2 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
▶   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

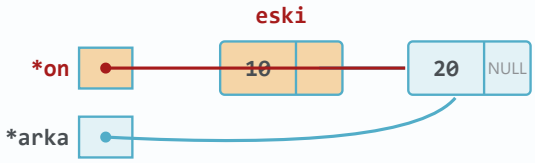
- `*on` = **düğüm(10)**
- `*arka` = **düğüm(20)**
- `eski` = **düğüm(10)**

ÇAĞRI YIĞINI

- `kuyruk_al(&on, &arka, &x)`
- `main()`

ÇIKTI

Kod İzleme: Bağlı `kuyruk_al`



***x = 10**; ***on = eski->bag** → düğüm(20).

Adım 3 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
▶   *x = eski->deger;
▶   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

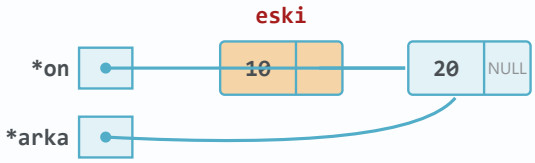
- `*on` = **düğüm(20)**
- `*arka` = **düğüm(20)**
- `eski` = **düğüm(10)**
- `*x` = **10**

ÇAĞRI YIĞINI

- `kuyruk_al(&on, &arka, &x)`
- `main()`

ÇIKTI

Kod İzleme: Bağlı `kuyruk_al`



*on ≠ NULL: arka değişmez.

Adım 4 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
▶   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

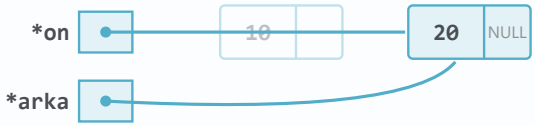
- `*on` = **düğüm(20)**
- `*arka` = **düğüm(20)**
- `eski` = **düğüm(10)**
- `*x` = **10**

ÇAĞRI YIĞINI

- `kuyruk_al(&on, &arka, &x)`
- `main()`

ÇIKTI

Kod İzleme: Bağlı `kuyruk_al`



free(eski); main 10 yazar.

Adım 5 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
▶   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

`*on` = **düğüm(20)**

`*arka` = **düğüm(20)** `*x` = **10**

ÇAĞRI YIĞINI

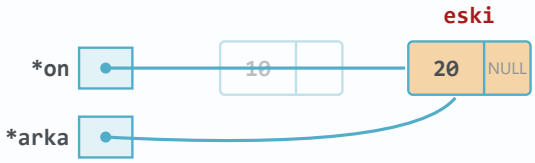
`kuyruk_al(&on, &arka, &x)`

`main()`

ÇIKTI

```
10
```

Kod İzleme: Bağlı `kuyruk_al`



eski = *on.

Adım 6 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
▶   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

- `*on` = **düğüm(20)**
- `*arka` = **düğüm(20)**
- `eski` = **düğüm(20)**

ÇAĞRI YIĞINI

- `kuyruk_al(&on, &arka, &x)`
- `main()`

ÇIKTI

```
10
```

Kod İzleme: Bağlı `kuyruk_al`



***x = 20**; ***on = eski->bag** → NULL.

Adım 7 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
▶   *x = eski->deger;
▶   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

`*on = NULL` `*arka = düğüm(20)` `eski = düğüm(20)` `*x = 20`

ÇAĞRI YIĞINI

`kuyruk_al(&on, &arka, &x)`

`main()`

ÇIKTI

```
10
```

Kod İzleme: Bağlı `kuyruk_al`



***on == NULL**: son öğe de çıktı; ***arka = NULL** yapılmazsa arka serbest bırakılacak düğümü göstermeye devam eder (sarkan işaretçi)!

Adım 8 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;      /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
▶   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

`*on = NULL` `*arka = NULL` `eski = düğüm(20)` `*x = 20`

ÇAĞRI YIĞINI

`kuyruk_al(&on, &arka, &x)`
`main()`

ÇIKTI

```
10
```

Kod İzleme: Bağlı `kuyruk_al`



free(eski); main 20 yazar.

Adım 9 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
2   dugum *eski = *on;
3   if (eski == NULL) return false;       /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
▶   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

`*on = NULL` `*arka = NULL` `*x = 20`

ÇAĞRI YIĞINI

`kuyruk_al(&on, &arka, &x)`

`main()`

ÇIKTI

```
10 20
```

Kod İzleme: Bağlı `kuyruk_al`



eski == NULL → boş, false döner.

Adım 10 / 10

```
1 bool kuyruk_al(dugum **on, dugum **arka, int *x) {
▶   dugum *eski = *on;
▶   if (eski == NULL) return false;      /* boş */
4   *x = eski->deger;
5   *on = eski->bag;
6   if (*on == NULL) *arka = NULL;  /* son öğe gitti */
7   free(eski);
8   return true;
9 }
```

DEĞİŞKENLER

`*on = NULL` `*arka = NULL` `eski = NULL`

ÇAĞRI YIĞINI

`kuyruk_al(&on, &arka, &x)`

`main()`

ÇIKTI

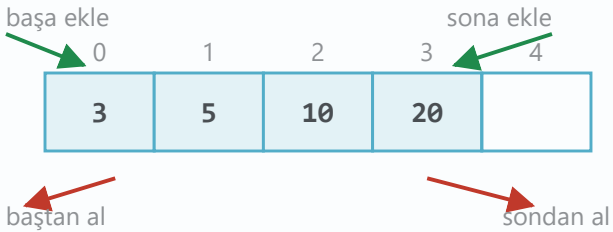
```
10 20
```

Kuyruk Gerçeklemelerinin Karşılaştırması

Ölçüt	Doğrusal dizi	Dairesel dizi	Bağlı liste
ekle / al	O(1), ama “sahte dolu”	O(1)	O(1) (malloc/free)
Kaydırma gerekir mi?	Evet: O(n)	Hayır	Hayır
Kapasite	MAX	MAX − 1 (ya da sayaçla MAX)	Bellek kadar
Ek bellek	Yok	Yok	Öğe başına bir işaretçi
Özel durum	on/arka sona dayanır	Sarma ve dolu/boş ayrımı	Boşalınca arka = NULL

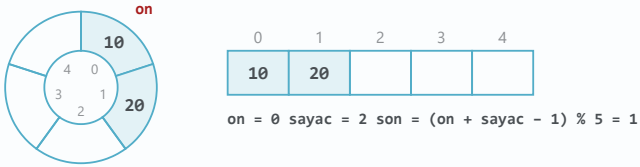
Pratikte en çok dairesel dizi (sabit kapasiteli tampon) ve büyüyen dairesel dizi kullanılır; bağlı kuyruk, öğeler zaten düğüm olarak yaşıyorsa ya da üst sınır bilinmiyorsa tercih edilir.

Çift Uçlu Kuyruk (Deque)



- Deque (*double-ended queue*): ekleme ve silme **her iki uçta**, hepsi O(1).
- Yalnız bir uç kullanılırsa **yığın**, bir uçtan ekleyip öbüründen alırsa **kuyruk** olur.
- Kısıtlı türler: girdisi kısıtlı (ekleme tek uçta), çıktısı kısıtlı (silme tek uçta).
- Kullanım: kayan pencere maksimumu, iş çalma (work stealing), geri al + yinele geçmişi.

Kod İzleme: Dairesel Dizide Deque



KAP = 5; `sona_ekle` ile 10 ve 20 kondu. Şimdi başa 5, başa 3; sonra iki kez sondan al.

Adım 1 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

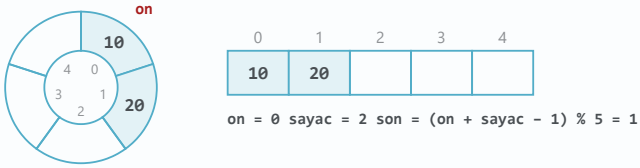
`d.on = 0` `d.sayac = 2`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



sayac = 2 < 5, yer var.

Adım 2 / 11

```
1  bool basa_ekle(Deque *d, int x) {
▶    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

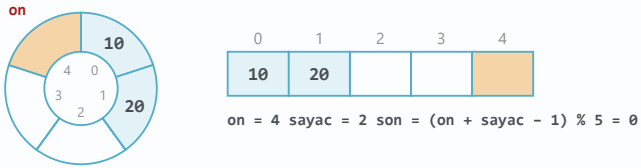
`d→on = 0` `d→sayac = 2` `x = 5`

ÇAĞRI YIĞINI

- `basa_ekle(&d, 5)`
- `main()`

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



on = (0 − 1 + 5) % 5 = 4: 0'ın bir gerisi dizinin **sonu**.

Adım 3 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
▶    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

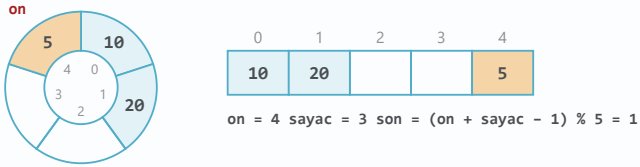
d→on = **4** | d→sayac = **2** | x = **5**

ÇAĞRI YIĞINI

- `basa_ekle(&d, 5)`
- `main()`

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



veri[4] = 5; sayac = 3.

Adım 4 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
▶    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

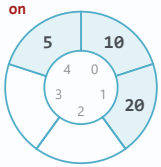
d→on = 4 | d→sayac = 3 | x = 5

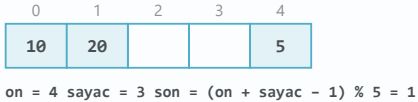
ÇAĞRI YIĞINI

basa_ekle(&d, 5)
main()

ÇIKTI

Kod İzleme: Dairesel Dizide Deque





sayac = 3 < 5, yer var.

Adım 5 / 11

```
1  bool basa_ekle(Deque *d, int x) {
▶    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

`d→on = 4` `d→sayac = 3` `x = 3`

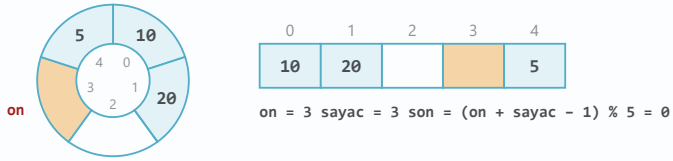
ÇAĞRI YIĞINI

`basa_ekle(&d, 3)`

`main()`

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



on = (4 − 1 + 5) % 5 = 3.

Adım 6 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
▶    d->on = (d->on - 1 + KAP) % KAP;    /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

d→on = 3 | d→sayac = 3 | x = 3

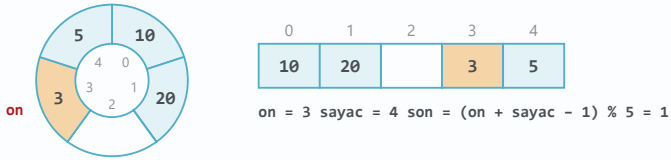
ÇAĞRI YIĞINI

basa_ekle(&d, 3)

main()

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



veri[3] = 3; sayac = 4.

Adım 7 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
▶    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

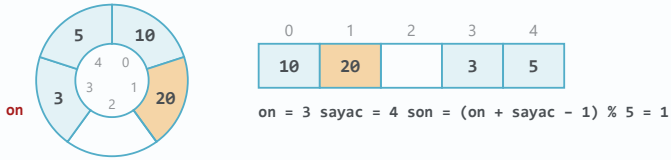
d→on = 3 | d→sayac = 4 | x = 3

ÇAĞRI YIĞINI

basa_ekle(&d, 3)
main()

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



Son konum = (3 + 4 − 1) % 5 = 1; ***x = 20**.

Adım 8 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
▶    if (d->sayac == 0) return false;
▶    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

`d→on = 3` `d→sayac = 4` `*x = 20`

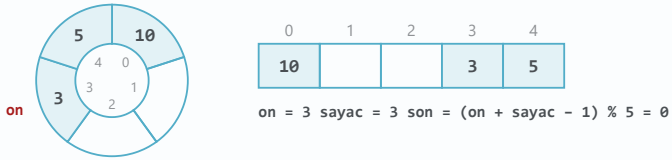
ÇAĞRI YIĞINI

`sondan_al(&d, &x)`

`main()`

ÇIKTI

Kod İzleme: Dairesel Dizide Deque



sayac = 3. (`on` değişmez.)

Adım 9 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
▶    d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

`d→on = 3` `d→sayac = 3` `*x = 20`

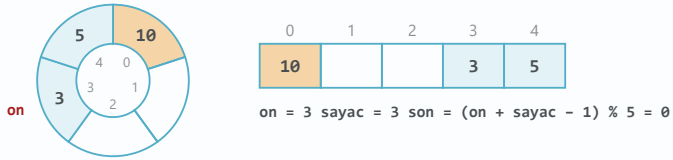
ÇAĞRI YIĞINI

- `sondan_al(&d, &x)`
- `main()`

ÇIKTI

```
20
```

Kod İzleme: Dairesel Dizide Deque



Son konum = (3 + 3 − 1) % 5 = 0; ***x = 10**.

Adım 10 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
▶    if (d->sayac == 0) return false;
▶    *x = d->veri[(d->on + d->sayac - 1) % KAP];
10   d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

`d→on = 3` `d→sayac = 3` `*x = 10`

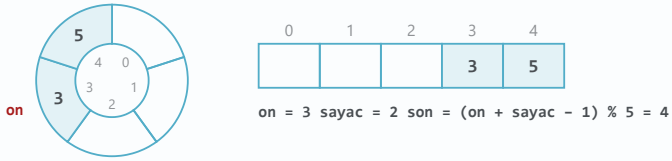
ÇAĞRI YIĞINI

- `sondan_al(&d, &x)`
- `main()`

ÇIKTI

```
20
```

Kod İzleme: Dairesel Dizide Deque



sayac = 2. (`on` değişmez.)

Adım 11 / 11

```
1  bool basa_ekle(Deque *d, int x) {
2    if (d->sayac == KAP) return false;
3    d->on = (d->on - 1 + KAP) % KAP;     /* bir geri */
4    d->veri[d->on] = x;  d->sayac++;
5    return true;
6  }
7  bool sondan_al(Deque *d, int *x) {
8    if (d->sayac == 0) return false;
9    *x = d->veri[(d->on + d->sayac - 1) % KAP];
▶    d->sayac--;  return true;
11 }
```

DEĞİŞKENLER

`d→on = 3` `d→sayac = 2` `*x = 10`

ÇAĞRI YIĞINI

`sondan_al(&d, &x)`

`main()`

ÇIKTI

```
20 10
```

Tek Dizide Birden Çok Kuyruk

Kesimlere bölme

Çoklu yığında olduğu gibi tek boyutlu bir `memory[]` dizisi kesimlere bölünür; her kuyruğun `on[i]`, `arka[i]` ve `sinir[i]` değerleri tutulur.

İki kuyruk

Yığındaki "uçlardan büyütme" hilesi kuyrukta işe yaramaz: kuyruk iki ucunda da hareket eder, kesim içinde kayar.

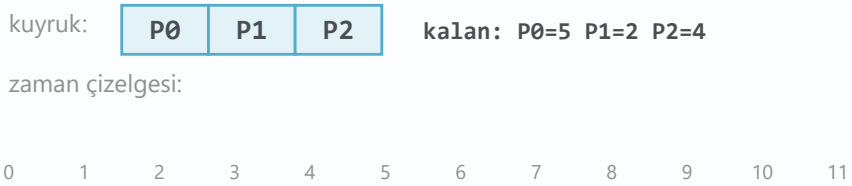
Kesim içinde dairesel

Her kesim kendi başına dairesel kuyruk gibi yönetilirse kaydırma gerekmez; ama bir kesim dolunca yine yer açmak (O(m)) gerekir.

Günümüzde

Genellikle her kuyruk için ayrı büyüyen dizi ya da bağlı liste tercih edilir.

Kod İzleme: Round Robin Zamanlama



Üç süreç: P0 = 5, P1 = 2, P2 = 4 birim iş. Zaman dilimi Q = 2 (**round robin**).

Adım 1 / 13

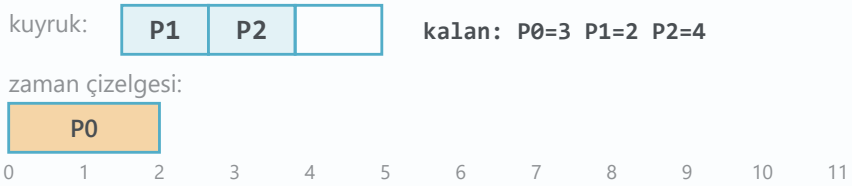
```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);           /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

DEĞİŞKENLER

`Q = 2` `zaman = 0`

ÇIKTI

Kod İzleme: Round Robin Zamanlama



Öndeki **P0** alınır, 2 birim çalışır; t = 2.

Adım 2 / 13

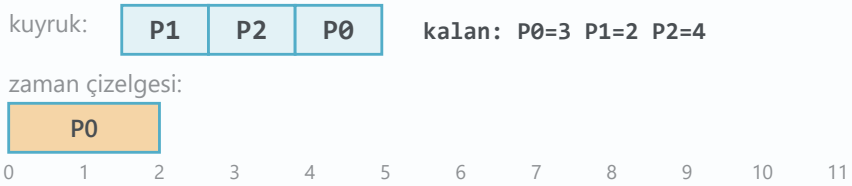
```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
▶  while (kuyruk_al(&k, &p)) {
▶    int dilim = kalan[p] < Q ? kalan[p] : Q;
▶    zaman += dilim;
▶    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

DEĞİŞKENLER

`p = 0` `dilim = 2` `zaman = 2` `kalan[0] = 3`

ÇIKTI

Kod İzleme: Round Robin Zamanlama



P0'ın 3 birimi kaldı → **kuyruğun sonuna** geri döner.

Adım 3 / 13

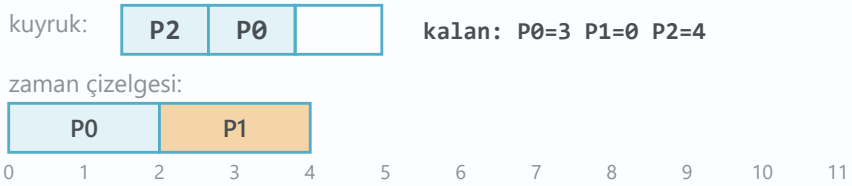
```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
▶    if (kalan[p] > 0)
▶      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

DEĞİŞKENLER

`p = 0` `dilim = 2` `zaman = 2` `kalan[0] = 3`

ÇIKTI

Kod İzleme: Round Robin Zamanlama



Öndeki **P1** alınır, 2 birim çalışır; t = 4.

Adım 4 / 13

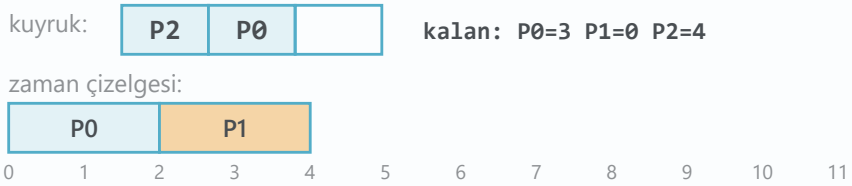
```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
▶  while (kuyruk_al(&k, &p)) {
▶    int dilim = kalan[p] < Q ? kalan[p] : Q;
▶    zaman += dilim;
▶    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

DEĞİŞKENLER

`p = 1` `dilim = 2` `zaman = 4` `kalan[1] = 0`

ÇIKTI

Kod İzleme: Round Robin Zamanlama



P1 bitti; kuyruğa dönmez.

Adım 5 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
▶      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

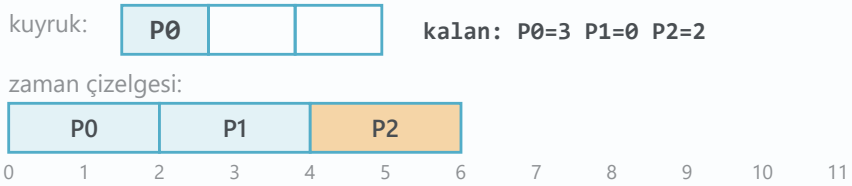
DEĞİŞKENLER

`p = 1` `dilim = 2` `zaman = 4` `kalan[1] = 0`

ÇIKTI

```
P1 biter: t=4
```

Kod İzleme: Round Robin Zamanlama



Öndeki **P2** alınır, 2 birim çalışır; t = 6.


```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
▶  while (kuyruk_al(&k, &p)) {
▶    int dilim = kalan[p] < Q ? kalan[p] : Q;
▶    zaman += dilim;
▶    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

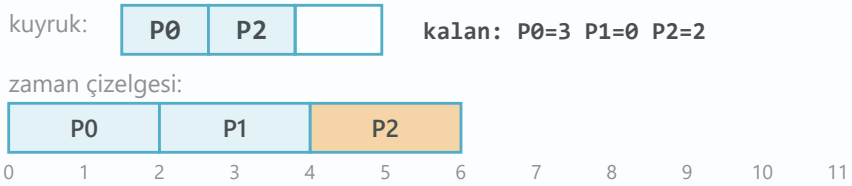
DEĞİŞKENLER

`p = 2` `dilim = 2` `zaman = 6` `kalan[2] = 2`

ÇIKTI

```
P1 biter: t=4
```

Kod İzleme: Round Robin Zamanlama



P2'nin 2 birimi kaldı → **kuyruğun sonuna** geri döner.

Adım 7 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
▶    if (kalan[p] > 0)
▶      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

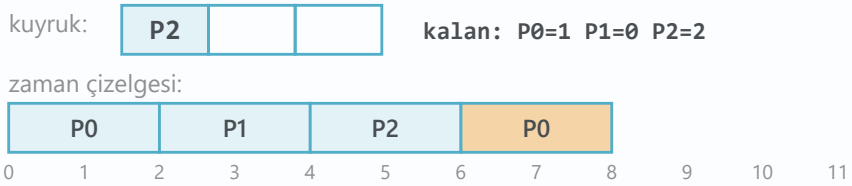
DEĞİŞKENLER

`p = 2` `dilim = 2` `zaman = 6` `kalan[2] = 2`

ÇIKTI

```
P1 biter: t=4
```

Kod İzleme: Round Robin Zamanlama



Öndeki **P0** alınır, 2 birim çalışır; t = 8.

Adım 8 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
▶  while (kuyruk_al(&k, &p)) {
▶    int dilim = kalan[p] < Q ? kalan[p] : Q;
▶    zaman += dilim;
▶    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

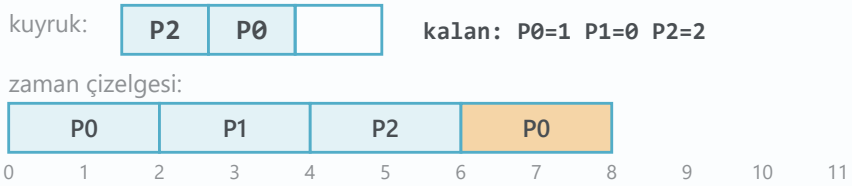
DEĞİŞKENLER

`p = 0` `dilim = 2` `zaman = 8` `kalan[0] = 1`

ÇIKTI

```
P1 biter: t=4
```

Kod İzleme: Round Robin Zamanlama



P0'ın 1 birimi kaldı → **kuyruğun sonuna** geri döner.

Adım 9 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
▶    if (kalan[p] > 0)
▶      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

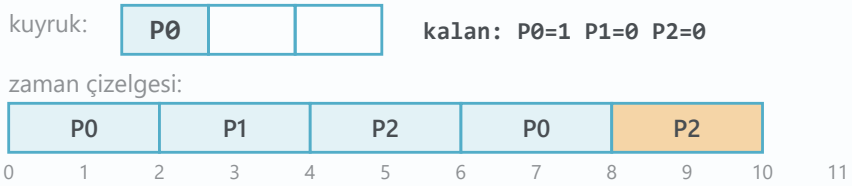
DEĞİŞKENLER

`p = 0` `dilim = 2` `zaman = 8` `kalan[0] = 1`

ÇIKTI

```
P1 biter: t=4
```

Kod İzleme: Round Robin Zamanlama



Öndeki **P2** alınır, 2 birim çalışır; t = 10.

Adım 10 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
▶  while (kuyruk_al(&k, &p)) {
▶    int dilim = kalan[p] < Q ? kalan[p] : Q;
▶    zaman += dilim;
▶    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

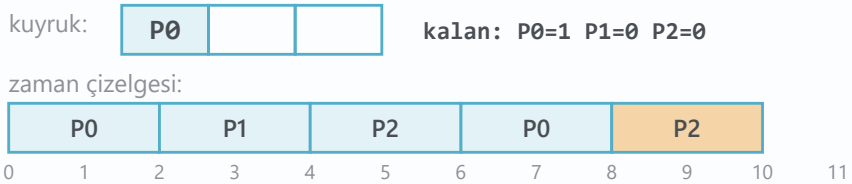
DEĞİŞKENLER

`p = 2` `dilim = 2` `zaman = 10` `kalan[2] = 0`

ÇIKTI

```
P1 biter: t=4
```

Kod İzleme: Round Robin Zamanlama



P2 bitti; kuyruğa dönmez.

Adım 11 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);            /* sona geri */
8    else
▶      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

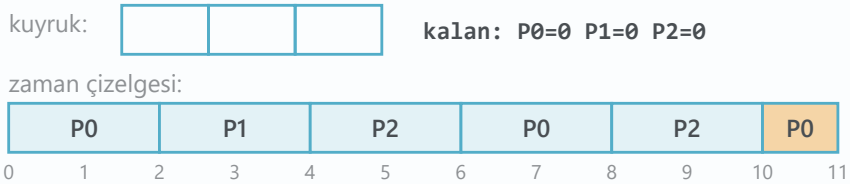
DEĞİŞKENLER

`p = 2` `dilim = 2` `zaman = 10` `kalan[2] = 0`

ÇIKTI

```
P1 biter: t=4
P2 biter: t=10
```

Kod İzleme: Round Robin Zamanlama



Öndeki **P0** alınır, 1 birim çalışır; t = 11.

Adım 12 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
▶  while (kuyruk_al(&k, &p)) {
▶    int dilim = kalan[p] < Q ? kalan[p] : Q;
▶    zaman += dilim;
▶    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);          /* sona geri */
8    else
9      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

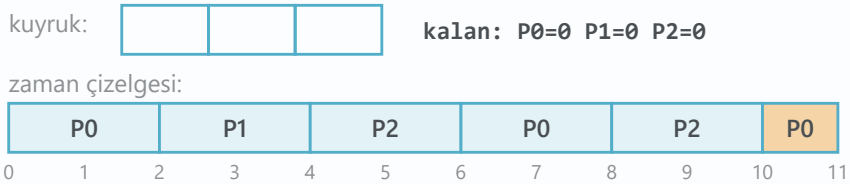
DEĞİŞKENLER

`p = 0` `dilim = 1` `zaman = 11` `kalan[0] = 0`

ÇIKTI

```
P1 biter: t=4
P2 biter: t=10
```

Kod İzleme: Round Robin Zamanlama



P0 bitti; kuyruğa dönmez.

Adım 13 / 13

```
1  /* k: süreç numaraları kuyruğu; Q: zaman dilimi */
2  while (kuyruk_al(&k, &p)) {
3    int dilim = kalan[p] < Q ? kalan[p] : Q;
4    zaman += dilim;
5    kalan[p] -= dilim;
6    if (kalan[p] > 0)
7      kuyruk_ekle(&k, p);            /* sona geri */
8    else
▶      printf("P%d biter: t=%d\n", p, zaman);
10 }
```

DEĞİŞKENLER

`p = 0` `dilim = 1` `zaman = 11` `kalan[0] = 0`

ÇIKTI

```
P1 biter: t=4
P2 biter: t=10
P0 biter: t=11
```

Kod İzleme: İki Yığınla Kuyruk



Kuyruk iki yığından oluşur. Ekle 1, 2, 3; al; ekle 4; üç kez al.

Adım 1 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = []` `cikis = []`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



ekle(1): giriş yığınına **push**, O(1).

Adım 2 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
▶    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [1]` `cikis = []`

ÇAĞRI YIĞINI

`ekle(&q, 1)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



ekle(2): giriş yığınına **push**, O(1).

Adım 3 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
▶    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [1, 2]` `cikis = []`

ÇAĞRI YIĞINI

`ekle(&q, 2)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



ekle(3): giriş yığınına **push**, O(1).

Adım 4 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
▶    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [1, 2, 3]` `cikis = []`

ÇAĞRI YIĞINI

`ekle(&q, 3)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



Çıkış yığını boş → girişteki her şey aktarılacak.

Adım 5 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
▶    if (yigit_bos(&q->cikis))        /* çıkış boşsa */
7      while (pop(&q->giris, &t))     /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [1, 2, 3]` `cikis = []`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



3 giriş'ten pop, çıkış'a push.

Adım 6 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))        /* çıkış boşsa */
▶      while (pop(&q->giris, &t))     /* hepsini aktar */
▶        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [1, 2]` `cikis = [3]`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



2 giriş'ten pop, çıkış'a push.

Adım 7 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
▶      while (pop(&q->giris, &t))    /* hepsini aktar */
▶        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [1]` `cikis = [3, 2]`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



1 giriş'ten pop, çıkış'a push.

Adım 8 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
▶      while (pop(&q->giris, &t))    /* hepsini aktar */
▶        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = []` `cikis = [3, 2, 1]`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

Kod İzleme: İki Yığınla Kuyruk



Çıkış yığınının üstü **1**: en eski öğe. FIFO sağlandı.

Adım 9 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
▶    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = []` `cikis = [3, 2]` `*x = 1`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

```
1
```

Kod İzleme: İki Yığınla Kuyruk



ekle(4): giriş yığınına **push**, O(1).

Adım 10 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
▶    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [4]` `cikis = [3, 2]`

ÇAĞRI YIĞINI

`ekle(&q, 4)`

`main()`

ÇIKTI

```
1
```

Kod İzleme: İki Yığınla Kuyruk



Çıkış yığınının üstü **2**: en eski öğe. FIFO sağlandı.

Adım 11 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
▶    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [4]` `cikis = [3]` `*x = 2`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

```
1 2
```

Kod İzleme: İki Yığınla Kuyruk



Çıkış yığınının üstü **3**: en eski öğe. FIFO sağlandı.

Adım 12 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
▶    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [4]` `cikis = []` `*x = 3`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

```
1 2 3
```

Kod İzleme: İki Yığınla Kuyruk



Çıkış yığını boş → girişteki her şey aktarılacak.

Adım 13 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
▶    if (yigit_bos(&q->cikis))        /* çıkış boşsa */
7      while (pop(&q->giris, &t))     /* hepsini aktar */
8        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = [4]` `cikis = []`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

```
1 2 3
```

Kod İzleme: İki Yığınla Kuyruk



4 giriş'ten pop, çıkış'a push.

Adım 14 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))        /* çıkış boşsa */
▶      while (pop(&q->giris, &t))     /* hepsini aktar */
▶        push(&q->cikis, t);
9    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = []` `cikis = [4]`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

```
1 2 3
```

Kod İzleme: İki Yığınla Kuyruk



Çıkış yığınının üstü **4**: en eski öğe. FIFO sağlandı. Her öğe en çok bir kez aktarıldı → **amortize O(1)**.

Adım 15 / 15

```
1  void ekle(Kuyruk2 *q, int x) {
2    push(&q->giris, x);
3  }
4  bool al(Kuyruk2 *q, int *x) {
5    int t;
6    if (yigit_bos(&q->cikis))       /* çıkış boşsa */
7      while (pop(&q->giris, &t))    /* hepsini aktar */
8        push(&q->cikis, t);
▶    return pop(&q->cikis, x);
10 }
```

DEĞİŞKENLER

`giris = []` `cikis = []` `*x = 4`

ÇAĞRI YIĞINI

`al(&q, &x)`

`main()`

ÇIKTI

```
1 2 3 4
```

Hangisi Uygun?

Problem	Yapı
Yazıcıya gönderilen belgeler	▶ göster
Metin düzenleyicide Ctrl+Z	▶ göster
Ağacı düzey düzey yazdırmak	▶ göster
İç içe parantezlerin denetimi	▶ göster
Fonksiyon çağrıları	▶ göster
Acil serviste en ağır hasta önce	▶ göster

HOCADAN "Kuyruk diye bir şey biliniyor, bunda da kuyruk kelimesi geçiyor. Ama bildiğimiz kuyrukta sondan eklenir, baştan alınır. Bu ise ondan farklı..."

Laboratuvar: Dairesel Kuyruk

Sarmal ilerleme

Girdi (sayı = ekle, al) `10, 20, 30, al, al, 40, 50, 60, 70, al` Uygula

KOD

```
 1 #define MAX_QUEUE_SIZE 6
 2 bool kuyruk_ekle(int on, int *arka,
 3                   element x) {
 4   int yeni = (*arka + 1) % MAX_QUEUE_SIZE;
 5   if (yeni == on) return false;  /* dolu */
 6   *arka = yeni;  kuyruk[yeni] = x;
 7   return true;
 8 }
 9 bool kuyruk_al(int *on, int arka,
10                 element *x) {
11   if (*on == arka) return false; /* boş */
12   *on = (*on + 1) % MAX_QUEUE_SIZE;
13   *x = kuyruk[*on];  return true;
14 }
```



`MAX_QUEUE_SIZE 6` `on 0` `arka 0`

ÇIKTI (henüz yok)

Başlangıçta kuyruk boş: `on = arka = 0`. Dizide 6 yuva var; dairesel kuyruk en çok 5 öğe tutar, çünkü bir yuva dolu ile boşu ayırmak için boş bırakılır.

ÖZET

Karmaşıklık Özeti

İşlem	Doğrusal dizi	Dairesel dizi	Bağlı	Deque (dairesel)
ekle	O(1)*	O(1)	O(1)	O(1) iki uçta
al	O(1)	O(1)	O(1)	O(1) iki uçta
ön / boş / dolu	O(1)	O(1)	O(1)	O(1)
öğe sayısı	arka − on	(arka−on+MAX)%MAX	O(n) ya da sayaç	sayac
bellek	MAX	MAX	n · (öğe + işaretçi)	KAP

* Doğrusal kuyrukta ekleme O(1)'dir ama "sahte dolu" durumunda kaydırma yapılırsa O(n) olur. Round robin benzetimi ve düzey sırası gezinme n öğe için O(n) kuyruk işlemi yapar.

ÖZET

Sık Yapılan Hatalar

1. Doğrusal kuyrukta `arka == MAX-1`'i gerçek doluluk sanmak.
2. Dairesel kuyrukta `% MAX`'ı unutup dizi dışına yazmak.
3. Önce `arka`'yı ilerletip doluysa geri almamak: kuyruk boş görünür.
4. Geri adımda `(i - 1) % MAX` yazmak: C'de negatif sonuç çıkar.
5. Bağlı kuyruktan son öğe alınınca `arka = NULL` yapmamak.
6. Boş kuyruğa ilk eklemede `on`'u güncellemeyi unutmak.

ÖZET

Alıştırmalar

1. MAX = 6, on = 4, arka = 1: kaç öğe var, hangi indislerde?
2. Bayrak yöntemiyle dairesel kuyruğun `ekle` ve `al` yordamlarını yaz.
3. İki yığınla bir kuyruk yaz; `al` işleminin amortize maliyeti nedir?
4. Kuyruğu yalnız **arka** işaretçili dairesel bağlı listeyle yaz.
5. Round robin örneğini Q = 3 ile çalıştır: süreçler ne zaman biter?

▶ **İpuçları / cevaplar**

ÖZET

Bölüm Özeti

FIFO
Ekleme arkada, alma önde; ilk giren ilk çıkar.

Doğrusal kuyruk
Basit ama arka sona dayanınca öndeki boş yer kullanılamaz; kaydırma pahalı.

Dairesel kuyruk
`% MAX` ile sarma; boş on == arka, dolu (arka+1)%MAX == on.

Dolu/boş ayrımı
Bir hücre boş bırakılır ya da sayaç veya bayrak tutulur.

Bağlı kuyruk
on ve arka işaretçisi; son öğe çıkınca arka = NULL.

Uygulamalar
Deque, iş zamanlama, tampon, iki yığınla kuyruk, BFS.

ÖZET

Kaynaklar ve Devam

- E. Horowitz, S. Sahni, S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2. bs., Bölüm 3.3 ve 4.3.
- T. H. Cormen vd., *Introduction to Algorithms*, 3. bs., Bölüm 10.1 (kuyruklar, deque alıştırmaları).
- R. Sedgewick, *Algorithms in C*, Kısım 1–4, Bölüm 4.

← Bölüm 6: Yığınlar

Ders sayfası

Bölüm 8: Yığın ve Kuyruk Uygulamaları

Son

Bölüm 7: Kuyruklar

Sonraki bölüm: Bölüm 8: Yığın ve Kuyruk Uygulamaları · Ders sayfası