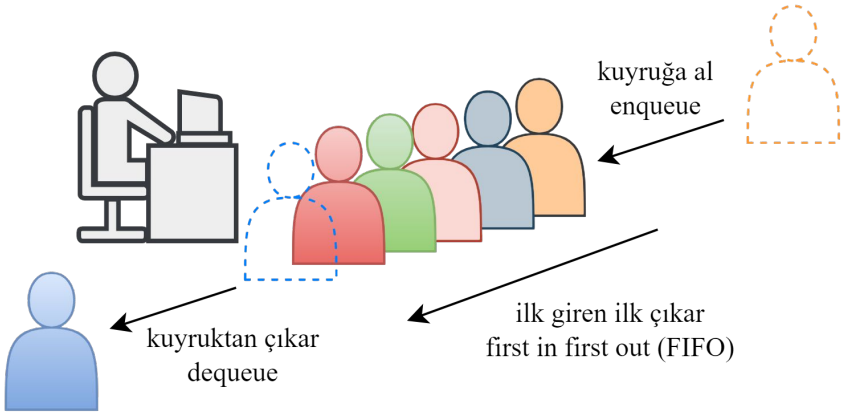
Kuyruk (Queue) Veri Yapısı

Dr. Hakan Temiz

Kuyruk

- Kuyruk veri yapısı **İlk Giren İlk Çıkar (First in First Out - FIFO)** prensibine uygun şekilde tasarlanmış doğrusal veri yapılarıdır. Yığınlarda, koleksiyona verilen son eleman ilk çıkarken, kuyruklarda koleksiyona ilk giren eleman ilk çıkar.
- Bu yapı, gerçek hayattaki kuyrukları andırır. Örneğin, bir markette ödeme kasasına gelindiğinde, kasadan ilk geçen kuyruktaki ilk kişidir.
- Kuyruktan öğe çıkarmaya **dequeue** (çıkar) ve kuyruğa almaya da **enqueue** (ekle) denir. Hem kuyruk hem yığında ekleme sondan yapılır; fakat çıkarma kuyrukta baştan, yığında sondan yapılır. Aralarındaki en temel fark budur.



Kuyruk - Türler

- **Basit Kuyruk:** FIFO Yapısını uygular. Öğe, sadece kuyruğun arkasına eklenebilir ve sadece önden çıkarılabilir.
- **Dairesel Kuyruk:** FIFO yapısını kavramsal bir daire üzerinde uygular.
- **Çift Uçlu Kuyruk (Dequeue):** Öğe ekleme ve çıkarma iki uçtan da yapılabilir. İki türü vardır:
  - **Giriş Kısıtlı Kuyruk:** Öğe, sadece bir uçtan eklenir; çıkarma iki uçtan da yapılabilir.
  - **Çıktı Kısıtlı Kuyruk:** Öğe, iki uçtan da eklenebilir; çıkarma sadece bir uçtan yapılabilir.
- **Öncelik Kuyruğu :** Öğelere kendilerine atanan önceliğe göre erişilir. İki türü vardır:
  - **Artan Öncelikli Kuyruk:** Öğeler, öncelik değerlerinin artan sırasına göre düzenlenir. En küçük öncelik değerine sahip öğe ilk olarak çıkarılır.
  - **Azalan Öncelik Kuyruğu:** Öğeler öncelik değerlerinin azalan sırasına göre düzenlenir. En büyük önceliğe sahip öğe ilk olarak çıkarılır.

Kuyruk - İşlemler

- **enqueue()** öğeyi kuyruk sonuna ekler. insert() adı da verilir. Türkçe ekle() ismi verilebilir.
- **dequeue()** ilk sıradaki öğeyi kuyruktan çıkarır. Türkçe, çek() veya çıkar() adı verilebilir.
- **peek()** veya **front()** kuyruğun en önündeki öğeyi silmeden verir. Türkçe, göster() veya ilk() adları verilebilir.
- **rear()** çıkarmadan, kuyruk sonundaki elemanı verir. Türkçe, son() adı verilebilir.
- **isEmpty()** kuyruk boş ise True, aksi durumda False döndürür. Türkçe, bosMu() adı verilebilir.
- **isFull()** kuyruk dolu ise True, aksi durumda False döndürür. Türkçe, doluMu() adı verilebilir.
- Tipik bir kuyrukta, öğe sayısını tutan ve **size** veya **boyut** diye adlandırılabilen bir değişken tutulur. Öğe ekleme ve kaldırma işlemleri sonrası değeri artırılır veya eksiltilir.

Kuyruk - İşlemler - enqueue()

enqueue() işlevi, kuyruk sonuna öğe eklemeyi sağlar. Kuyruk dolu ise Taşma Hatası (overflow) alınır. Algoritma şu şekilde gerçekleştirilebilir:

- Kuyruğa eleman eklemeden önce kuyruğun dolu olup olmadığı kontrol edilir.
- Kuyruk doluysa (yani, kuyruğun uzunluğu (size) = = kapasite), Taşması Hatası (overflow) meydana gelir; eleman eklenemez.
- Aksi durumda, size += 1 olur ve yeni öğe kuyruğun sonuna eklenir.
- Kapasite dolana kadar kuyruğa öğe eklemeye devam edilebilir.

Kuyruk - İşlemler - dequeue()

dequeue() işlemi, kuyruk sonundan öğe çıkarır. Kuyruk boş ise, buna Alt Taşma (Underflow) durumu denir. Algoritma şu şekilde gerçekleştirilebilir:

- Elemanı kuyruktan çıkarmadan önce kuyruğun boş olup olmadığı kontrol edilir.
- Kuyruk boş ise Alt Taşma (Underflow) meydana gelir ve kuyruktan hiçbir eleman kaldırılamaz.
- Aksi takdirde, size - = 1 olur; en öndeki öğe kuyruktan alınır; ön elemana referans bir sonraki elemana atanır ve çıkarılan öğe döndürülür (return).

Kuyruk - İşlemler - peek() veya front()

peek() veya front() işlevi, kuyruğun en önündeki öğeyi, kuyruktan çıkarmadan verir. Bu işlev şu şekilde gerçekleştirilebilir:

- Kuyruğun boş olup olmadığı kontrol edilir.
- Kuyruk boş ise herhangi bir öğe gösterilemez.
- Aksi takdirde, en öndeki öğe kuyruktan çıkarılmadan döndürülür (return).

Kuyruk - İşlemler - isEmpty()

isEmpty(), kuyruğun boş olup olmadığını kontrol etmeyi sağlayan işlevdir. Kuyruk boş ise True; değilse False değeri üretir. Böyle bir işlevin algoritması şu şekilde olabilir:

- Kuyruğun eleman sayısını tutan değişkenin değeri 0 ise, yani size == 0, True; aksi takdirde False değeri döndürülür.

Kuyruk - İşlemler - isFull()

isFull() işlevi, kuyruk doluysa True; aksi durumda False değeri üretir. Böyle bir işlevin algoritması şöyle olabilir:

- Kuyruğun eleman sayısını tutan değişkenin değeri kapasite ile aynı ise, yani size == kapasite, True; aksi takdirde False değeri döndürülür.

Kuyruk - Karmaşıklık

İşlem	Zaman Karmaşıklığı	Alan Karmaşıklığı
enqueue()	O(1)	O(1)
dequeue()	O(1)	O(1)
front() veya peek()	O(1)	O(1)
rear()	O(1)	O(1)
isEmpty()	O(1)	O(1)
isFull()	O(1)	O(1)
len()	O(1)	O(1)

Kuyruk - Avantajlar

- **Basitlik:** Kuyruklar, basit ve anlaşılması kolay bir veri yapısıdır.
- **Verimlilik:** Kuyruktaki işlemler genellikle O(1) karmaşıklığa sahip olduğundan, verimlidir. Büyük miktarlardaki veri kolaylıkla işlenebilir.
- **Çoklu Görev Dağılımına Uygunluk:** Kuyruklar, bir hizmet veya kaynağın çoklu istemci tarafından kullanılmasına imkan verir.
- **Hız:** Kuyruklar veri süreçleri arası iletişimin hızlı yapılmasına olanak tanır.
- **Diğer Veri Yapılarına Altyapı:** Kuyruklar, diğer veri yapılarının gerçekleştirilmesine olanak tanır

Kuyruk - Dezavantajlar

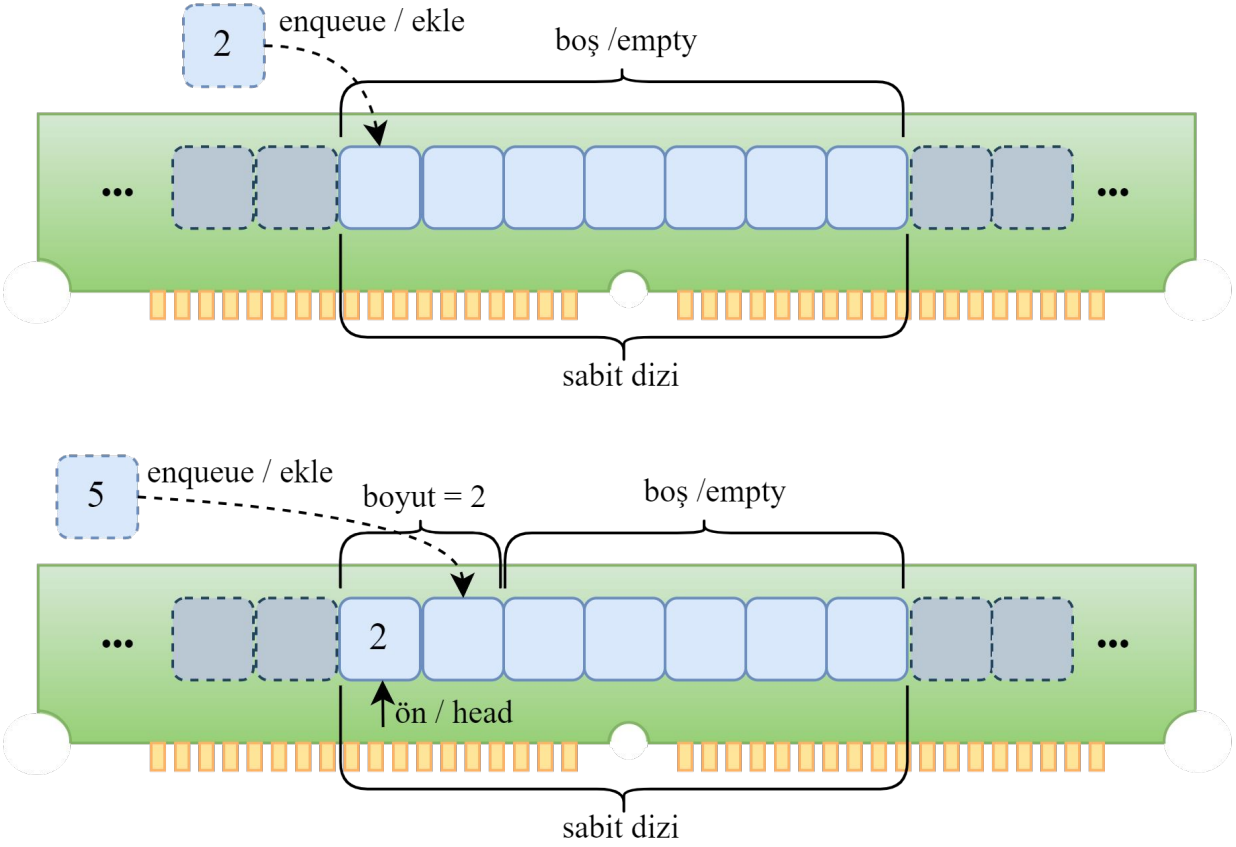
- **Sınırlı erişim:** Yığındaki daima en öndeki öğelere erişilebilir; öndeki işlenmeden (kaldırılmadan) sonraki elemanlara erişilemez. Dizilerdeki gibi rastgele erişim mümkün değildir. Bir öğenin O(n) zamanda aranır. Ayrıca, ortalara eleman eklemek veya kaldırmak maliyetlidir.
- **Dizilerde Uygulama Zorluğu:** Dizi üzerinde gerçekleştirimde, boyut önceden belirlenmek zorundadır. Bu durum gereksiz veya yetersiz bellek ayrılmasına neden olabilir. Sonradan, kuyruk boyutunu değiştirmek çok maliyetlidir.
- **Taşma olasılığı:** Sınırlı bir bellek alanında gerçekleştirilen bir kuyruğa, kapasitesinden fazla öğe yüklenmesi taşma hatasına ve veri kaybına neden olur.

Kuyruk - Dizilerde Gerçekleştirim

- Başlangıcı (önü) Sabit Kuyruk
- Başlangıcı (önü) Dinamik Kuyruk
- Çevrimsel Kuyruk

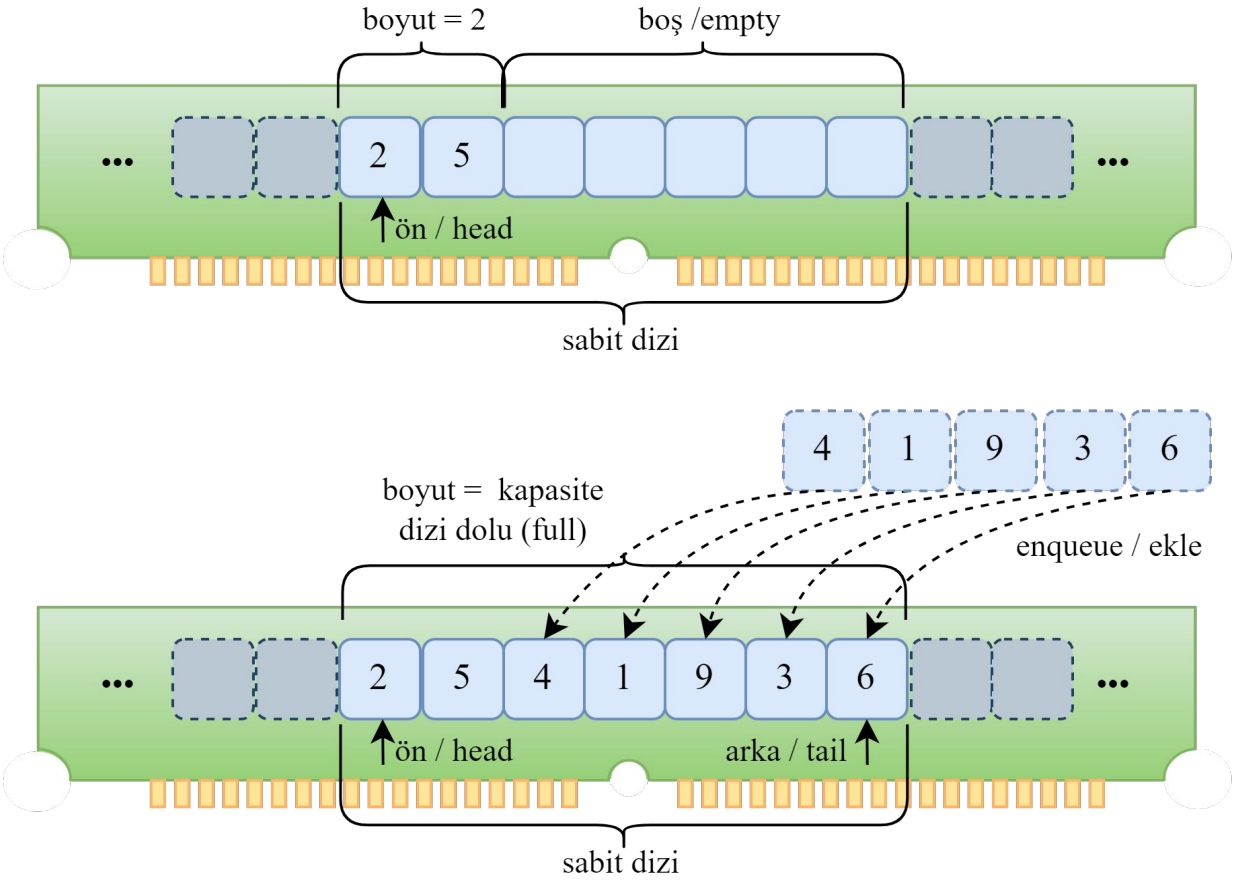
Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Sabit

Öğe ekleme



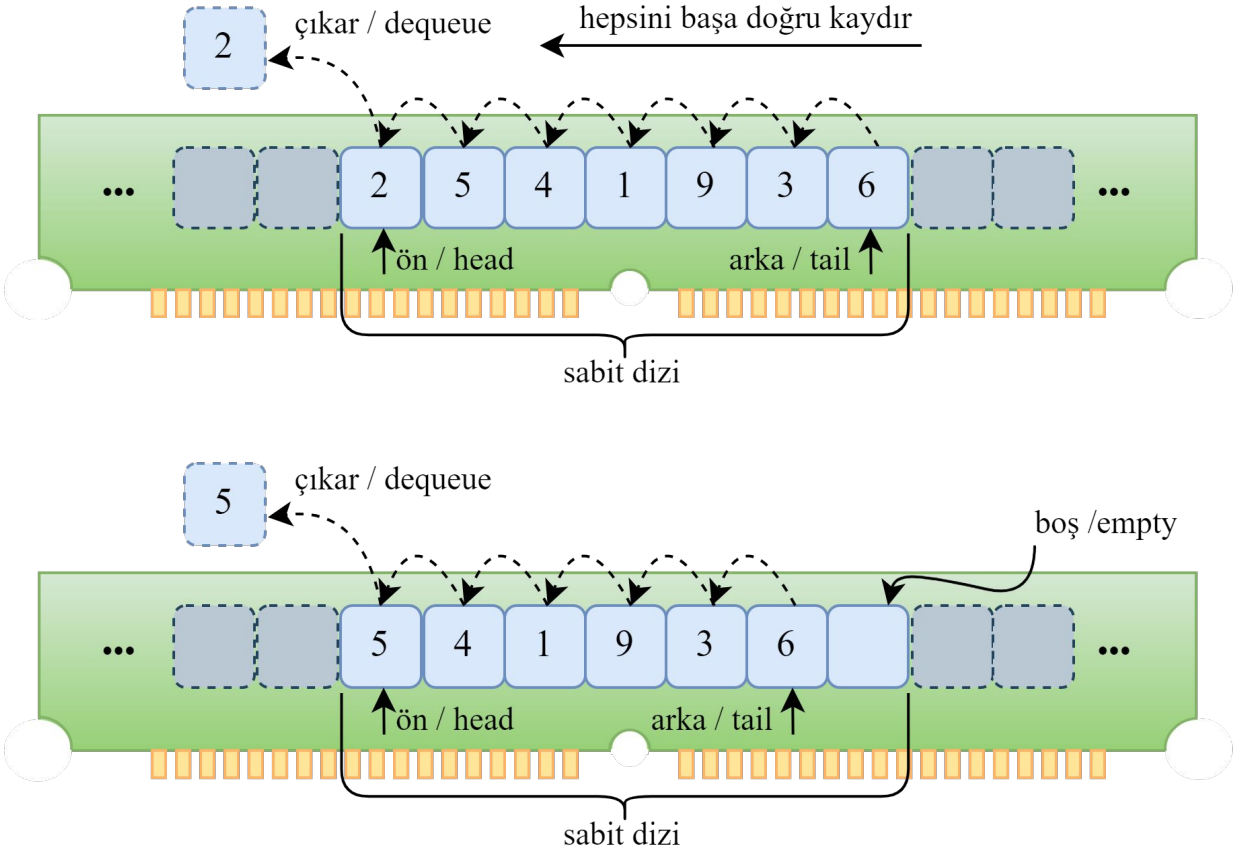
Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Sabit

Öğe ekleme



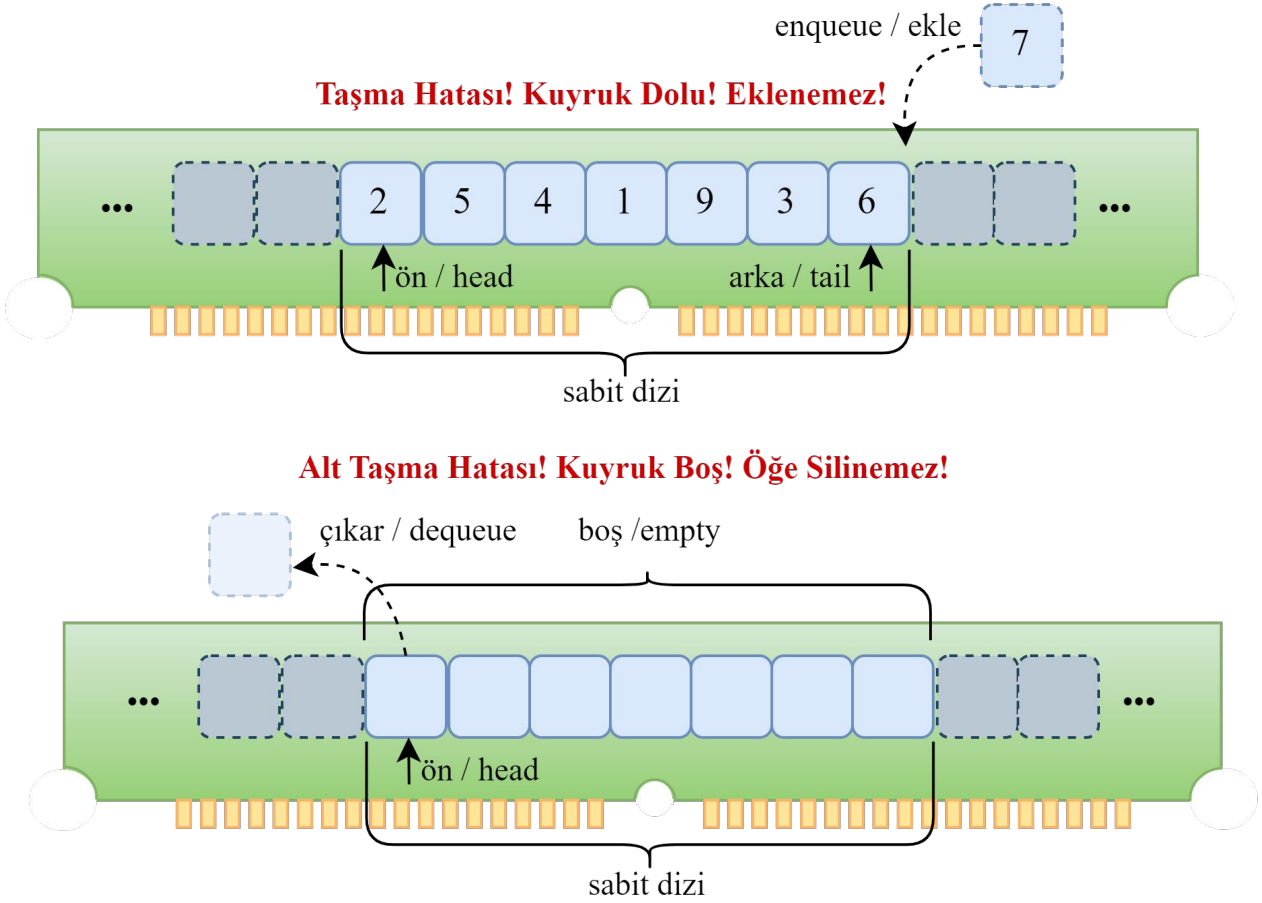
Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Sabit

Öğe Çıkarma



Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Sabit

Taşma Hataları

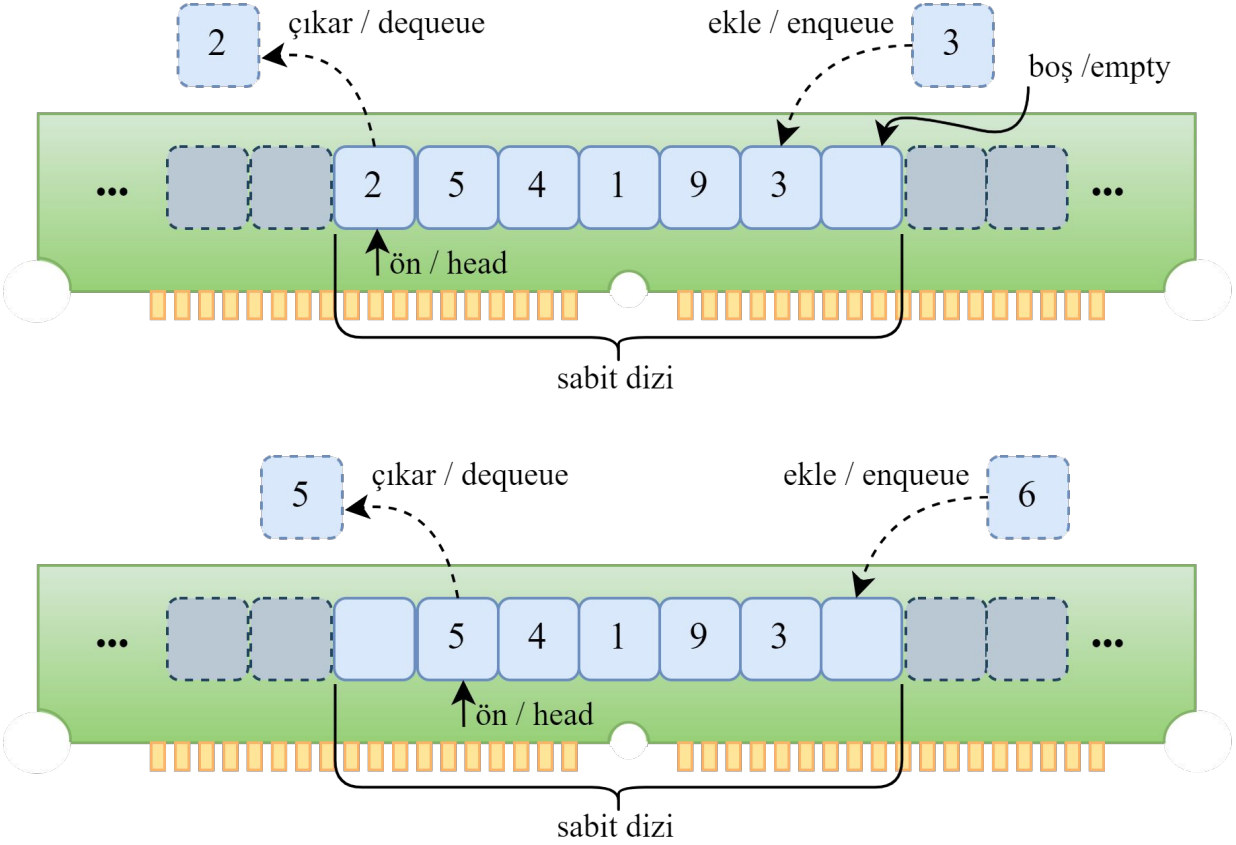


Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Dinamik

Ekleme sondan.

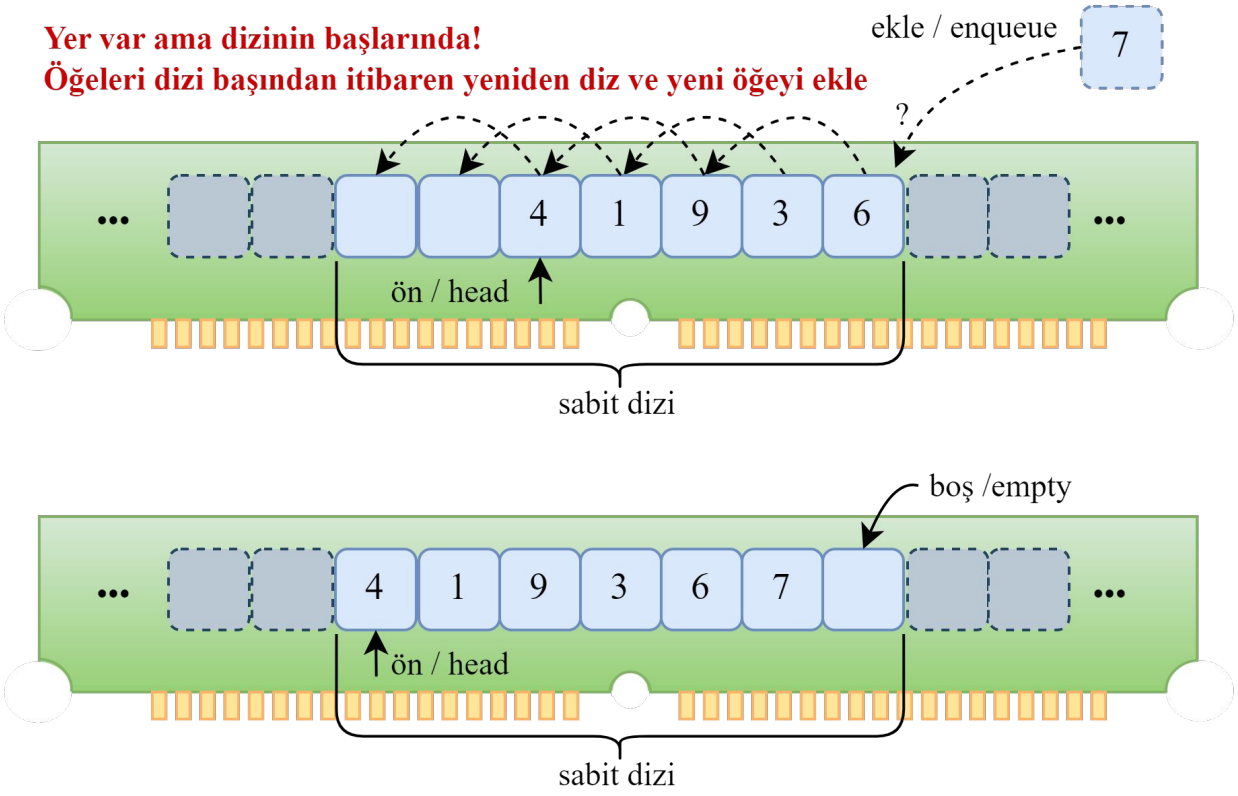
Çıkarma dizinin başından.

Kuyruk başlangıcı ilerletilir.



Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Dinamik

Kuyruk sonu dizi sonuna gelirse ve dizinin önünde yer varsa kuyruk dizi başlangıcından itibaren yeniden yerleştirilir. Öğe, sona eklenir..



Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Dinamik

```
class DinamikBaslangicliKuyruk: #Dizi üzerinde Kuyruk yapısı
    def __init__(self, kapasite):
    self.kapasite = kapasite # toplam kapasite
    self.baslangic = 0 # kuyruğun önü; başlangıç indeksi.
    self.boyut = 0 # (size) başlangıçta, kuyrukta eleman yok.
    self.kuyruk = [None] * self.kapasite # yığını saklandığı dizi

  def bastan_diz(self,):
    print("Kuyruk dizi sonuna ulaştı. Yeniden düzenleniyor")
    for i in range(self.boyut):
      self.kuyruk[i] = self.kuyruk[self.baslangic + i]
      self.kuyruk[self.baslangic + i] = None
    self.baslangic = 0 # indexi başa al.

  def enqueue(self, veri):
    if self.isFull():
      print("Taşma Hatası. Kuyruk dolu. Eleman eklenemez!")
      return False # ekleme başarısız manasında
    else:
      # kuyruk arkası dizi sonunda ve başta yer varsa.
      # Öğeleri başlangıçtan itibaren yeniden yerleştir.
      if self.boyut != self.kapasite and \
        self.baslangic + self.boyut == self.kapasite:
        # kuyruğu dizinin başından itibaren yeniden diz
        self.bastan_diz()
      self.kuyruk[self.baslangic + self.boyut] = veri
      self.boyut += 1
      print(f"{veri} kuyruğa eklendi.")
      self.yazdir()
      return True # eklemenin sorunsuz yapıldığı anlamında
```

```
  def len(self, ):
      return self.boyut
  def isEmpty(self, ):
      return self.boyut == 0
  def isFull(self, ):
      return self.boyut == self.kapasite
  def yazdir(self,):
      print("Başlangıç: ", self.baslangic, "\tBoyut: ", self.boyut)
      print("Kuyruk: ", self.kuyruk)

  def dequeue(self, ):
    if self.isEmpty():
      print("Alt Taşma Hatası. Kuyruk boş. Eleman çıkarılamaz!")
      return None #
    else:
      oge = self.kuyruk[self.baslangic]
      self.kuyruk[self.baslangic] = None
      self.baslangic += 1
      self.boyut  -= 1
      print(f"{oge} kuyruktan çıkarıldı.")
      if self.boyut == 0: # son eleman çıkarıldı demektir.
        self.baslangic = 0 # başlangıcı 0 indeksine al.
      self.yazdir()
      return oge  # öğeyi döndür
  def peek(self, ):
    if not self.isEmpty():
      return self.kuyruk[self.baslangic]
    else:
      return None
```

Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Dinamik

Kuyruğa "A", "B" ve "C" verilerini ekleyelim (enqueue):

```
kuyruk = DinamikBaslangicliKuyruk(3) # Uzunluk 3
kuyruk.enqueue("A")
kuyruk.enqueue("B")
kuyruk.enqueue("C")
print (f"En öndeki öğe: {kuyruk.peek()}")
```

```
A kuyruğa eklendi.
Başlangıç:  0    Boyut:  1
Kuyruk:  ['A', None, None]
B kuyruğa eklendi.
Başlangıç:  0    Boyut:  2
Kuyruk:  ['A', 'B', None]
C kuyruğa eklendi.
Başlangıç:  0    Boyut:  3
Kuyruk:  ['A', 'B', 'C']
En öndeki öğe: A
```

```
kuyruk.kapasite # Kuyruğun toplam boyu nedir?
```

3

```
kuyruk.len() # Kuyruğun uzunluğu (öğe sayısı)
```

3

```
kuyruk.enqueue("D")# aşma hatası(Overflow)!
```

```
Taşma Hatası. Kuyruk dolu. Eleman eklenemez!
```

```
kuyruk.dequeue() # Önden bir eleman (A) çekelim.
```

```
A kuyruktan çıkarıldı.
Başlangıç:  1    Boyut:  2
Kuyruk:  [None, 'B', 'C']
'A'
```

```
# kuyruk dizi sonuna erişti! dizinin başından itibaren
# yeniden yerleştirdikten sonra öğeyi ekle.
kuyruk.enqueue("D")
```

```
Kuyruk dizi sonuna ulaştı. Yeniden düzenleniyor
D kuyruğa eklendi.
Başlangıç:  0    Boyut:  3
Kuyruk:  ['B', 'C', 'D']
```

Kuyruk - Dizilerde Gerçekleştirim - Başlangıcı Dinamik

```
kuyruk.dequeue()# kuyruktan öğe çıkar.
kuyruk.dequeue()# kuyruktan öğe çıkar.
B kuyruktan çıkarıldı.
Başlangıç:  1     Boyut:  2
Kuyruk:  [None, 'C', 'D']
C kuyruktan çıkarıldı.
Başlangıç:  2     Boyut:  1
Kuyruk:  [None, None, 'D']
'C'
```

```
kuyruk.enqueue("E")# dizi sonu! baştan diz; sonra ekle.
Kuyruk dizi sonuna ulaştı. Yeniden düzenleniyor
E kuyruğa eklendi.
Başlangıç:  0     Boyut:  2
Kuyruk:  ['D', 'E', None]
```

```
kuyruk.dequeue() # bundan sonra tek öğe (E) kalacak
kuyruk.dequeue() # kuyruk artık tamamen boş
D kuyruktan çıkarıldı.
Başlangıç:  1     Boyut:  1
Kuyruk:  [None, 'E', None]
E kuyruktan çıkarıldı.
Başlangıç:  0     Boyut:  0
Kuyruk:  [None, None, None]
'E
```

```
kuyruk.enqueue("F") # F kuyruğun başına eklendi
F kuyruğa eklendi.
Başlangıç:  0     Boyut:  1
Kuyruk:  ['F', None, None]
```

```
kuyruk.dequeue() # F silindi, hiç öğe yok artık.
kuyruk.dequeue() # Alt Taşma Hatası!
F kuyruktan çıkarıldı.
Başlangıç:  0     Boyut:  0
Kuyruk:  [None, None, None]
Alt Taşma Hatası. Kuyruk boş. Eleman çıkarılamaz!
```

```
kuyruk.boyut
0
```

Kuyruk - Dizilerde Gerçekleştirim - Çevrimsel

- Kuyruk sonu dizi sonuna vardığında yeniden düzenlemek yerine, yer varsa, dizi başındaki ilk müsait indekse öğe eklenir.
- Kuyruk önü referansı dizi sonuna ulaştığında tekrar dizinin başına (0. indeks) referansta bulunur.
- **enqueue()** ve **dequeue()** metodları farklı; **bastan_diz()** ‘e gerek yok.

```
def enqueue(self, veri):
    if self.isFull():
        print("Taşma Hatası. Kuyruk dolu. Eleman
              eklenemez!")
        return False # ekleme başarısız
    else:
        musait = (self.baslangic + self.boyut) %
                                        self.kapasite
        self.kuyruk[musait] = veri
        self.boyut += 1
        print(f"{veri} kuyruğa eklendi.")
        self.yazdir()
        return True # eklemenin sorunsuz
```

```
def dequeue(self, ):
    if self.isEmpty():
        print("Alt Taşma Hatası. Kuyruk boş. Eleman
              çıkarılamaz!")
        return None #
    else:
        oge = self.kuyruk[self.baslangic]
        self.kuyruk[self.baslangic] = None
        self.baslangic  = (self.baslangic + 1) %
                                        self.kapasite
        self.boyut  -= 1
        print(f"{oge} kuyruktan çıkarıldı.")
        if self.boyut == 0: # son eleman çıkarıldı
            self.baslangic = 0 # baş indeks = 0.
        self.yazdir()
        return oge  # öğeyi döndür
```

Kuyruk - Bağlı Liste ile Gerçekleştirim

```
class BagliListeKuyruk:# Kuyruk (Queue) Sınıfı
  """Bağlı Listede Kuyruk "Queue""""

  class KuyrukDugumu:
    """Kuyruk için bağlı liste düğümü"""
    def __init__(self, veri): # Sınıfın inşaası
        self.veri = veri
        self.sonraki = None

   def __init__(self): # Kuyruk Sınıfı İnşaası
      self.kok = None # kuyruk boş
      self.tail = None # kuyruğun sonu boş

  def isEmpty(self):
     # Kuyruk boş ise True, değilse False
      return True if self.kok is None else False

  # üstteki öğeyi çıkarmadan göster/ver
  def peek(self):
      if self.isEmpty():
          return None # öğe yok
      return self.kok.veri
```

```
  def enqueue(self, veri): # Kuyruğa öğe ekle
      oge = self.KuyrukDugumu(veri)
      if self.isEmpty():
        self.kok = oge
        self.tail = oge
      else:
        self.tail.sonraki = oge
        self.tail = oge
      print(f"{veri} kuyruğa eklendi.")

  def dequeue(self): # bir öğe çek
      if (self.isEmpty()):
          return None
      oge = self.kok
      self.kok = self.kok.sonraki
      cekilen = oge.veri
      return cekilen
```

Kuyruk - Bağlı Liste ile Gerçekleştirim

```
# Kuyruğu oluştur ve "A", "B", "C" ile doldur.
Kuyruk = BagliListeKuyruk()
Kuyruk.enqueue("A")
Kuyruk.enqueue("B")
Kuyruk.enqueue("C")
print (f"{Kuyruk.dequeue()} kuyruktan çekildi")
print (f"En öndeki öğe: {Kuyruk.peek()}")
```

```
A kuyruğa eklendi.
B kuyruğa eklendi.
C kuyruğa eklendi.
A kuyruktan çekildi
En öndeki öğe: B
```