BLM267

Bölüm 8: Kuyruklar

C Kullanarak Veri Yapıları, İkinci Baskı
Reema Thareja

- Kuyruklara Giriş
- Kuyrukların Dizi Gösterimi
- Kuyrukların Bağlantılı Gösterimi
- Kuyruk Türleri
  - Dairesel Kuyruklar
  - Kuyruklar
  - Öncelikli Kuyruklar
  - Çoklu Kuyruklar
- Kuyrukların Uygulamaları

Kuyruklara Giriş

- Kuyruk kavramını aşağıda verilen benzetmelerle açıklayalım.
  - Yürüyen merdivende hareket eden insanlar. Yürüyen merdivene ilk binen insanlar, yürüyen merdivenden ilk inen kişiler olacaktır.
  - Otobüs bekleyen insanlar. Sırada duran ilk kişi otobüse ilk binen kişi olacaktır.
  - Bir sinema salonunun bilet gişesinin dışında duran insanlar. Sıradaki ilk kişi bileti ilk alacak ve böylece biletten ilk çıkan kişi olacaktır.
  - Bagajlar konveyör bantlarında tutulur. İlk yerleştirilen çanta diğer uçtan ilk çıkan çanta olacaktır.
  - Gişeli bir köprüde sıralanmış arabalar. Köprüye ilk ulaşan araba, ilk ayrılan olacak.

Kuyruklara Giriş

- Tüm bu örneklerde ilk pozisyondaki öğenin önce sunulduğunu görüyoruz.
- Aynı durum kuyruk veri yapısı için de geçerlidir.
- Kuyruk, ilk eklenen elemanın ilk çıkarılan eleman olduğu bir FIFO (İlk Giren, İlk Çıkar) veri yapısıdır.
- Bir kuyruktaki elemanlar ARKA adı verilen bir uçtan eklenir ve ÖN adı verilen diğer uçtan çıkarılır.
- Kuyruklar diziler veya bağlı listeler kullanılarak uygulanabilir.
- Bu bölümde, bu veri yapılarının her birini kullanarak kuyrukların nasıl uygulandığını göreceğiz.

Kuyrukların Dizi Gösterimi

- Kuyruklar doğrusal diziler kullanılarak kolayca temsil edilebilir.
- Daha önce de belirtildiği gibi, her kuyruğun sırasıyla silme ve ekleme işlemlerinin yapılabileceği konumu gösteren ön ve arka değişkenleri vardır.
- Bir kuyruğun dizi gösterimi Şekil 8.1'de gösterilmiştir.
- Kuyruklardaki İşlemler
- Şekil 8.1'de ön = 0 ve arka = 5.
- Diyelim ki değeri 45 olan bir eleman daha eklemek istiyoruz, o zaman rear 1 artırılır ve değer rear'ın işaret ettiği konumda saklanır.
- Eklemeden sonraki kuyruk Şekil 8.2'de gösterildiği gibi olacaktır. Burada ön = 0 ve arka = 6'dır.
- Her yeni bir elementin eklenmesi gerektiğinde aynı işlemi tekrarlıyoruz.

Kuyrukların Dizi Gösterimi

- Kuyruktan bir elemanı silmek istediğimizde front'un değeri artırılacaktır.
- Silmeler yalnızca kuyruğun bu ucundan yapılır.
- Silinme işleminden sonraki kuyruk Şekil 8.3'teki gibi olacaktır.
- Burada ön = 1 ve arka = 6

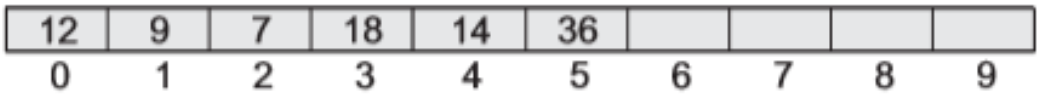

Figure 8.1 Queue

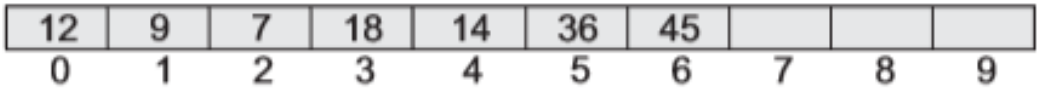

Figure 8.2 Queue after insertion of a new element

Figure 8.3 Queue after deletion of an element

Kuyrukların Dizi Gösterimi

- Ancak bir kuyruğa eleman eklemeden önce taşma koşullarını kontrol etmemiz gerekir.
- Zaten dolu olan bir kuyruğa bir eleman eklemeye çalıştığımızda taşma meydana gelir.
- rear = MaX – 1 olduğunda, MaX kuyruğun boyutunu ifade eder, taşma durumumuz vardır.
- MaX – 1 olarak yazdığımızı unutmayın çünkü indeks 0'dan başlıyor.
- Benzer şekilde, bir öğeyi kuyruktan silmeden önce, alt taşma koşullarını kontrol etmeliyiz.
- Bir kuyruktan boş olan bir öğeyi silmeye çalıştığımızda alt taşma durumu oluşur.
- Eğer front = –1 ve rear = –1 ise kuyrukta eleman yok demektir.
- Şimdi bir kuyruğa eleman ekleme ve kuyruktan eleman silme algoritmalarını gösteren Şekil 8.4 ve 8.5'e bakalım.

Kuyrukların Dizi Gösterimi

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```

Figure 8.4 Algorithm to insert an element in a queue

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```

Figure 8.5 Algorithm to delete an element from a queue

Kuyrukların Dizi Gösterimi

- Şekil 8.4 bir kuyruğa eleman ekleme algoritmasını göstermektedi
- Adım 1'de, ilk önce taşma koşulunu kontrol ederiz. Adım 2'de, kuyruğun boş olup olmadığını kontrol ederiz.
- Eğer kuyruk boş ise hem ön hem de arka sıfırlanır, böylece yeni değer 0. konumda saklanabilir.
- Aksi takdirde, kuyrukta zaten bazı değerler varsa, rear, dizideki bir sonraki konuma işaret edecek şekilde artırılır.
- Adım 3'te değer, rear'ın işaret ettiği konumdaki kuyrukta saklanır.
- Şekil 8.5, bir öğeyi bir kuyruktan silmek için algoritmayı gösterir. 1. Adımda, alt taşma koşulunu kontrol ederiz.
- Eğer front = –1 veya front > rear ise alt taşma meydana gelir.
- Ancak, kuyrukta bazı değerler varsa, ön değer artırılır ve artık kuyruktaki bir sonraki değere işaret eder.

Kuyrukların Dizi Gösterimi

```
void insert()
{
        int num;
        printf("\n Enter the number to be inserted in the queue : ");
        scanf("%d", &num);
        if(rear == MAX-1)
        printf("\n OVERFLOW");
        else if(front == -1 && rear == -1)
        front = rear = 0;
        else
        rear++;
        queue[rear] = num;
}
int delete_element()
{
        int val:
        if(front == -1 || front>rear)
        {
                printf("\n UNDERFLOW");
                return -1;
        }
        else
        {
                val = queue[front];
                front++;
                if(front > rear)
                front = rear = -1;
                return val;
        }
}
```

Kuyrukların Dizi Gösterimi

```
int peek()
{
        if(front==-1 || front>rear)
        {
                printf("\n QUEUE IS EMPTY");
                return -1;
        }
        else
        {
                return queue[front];
        }
}
void display()
{
        int i;
        printf("\n");
        if(front == -1 || front > rear)
        printf("\n QUEUE IS EMPTY");
        else
        {
                for(i = front;i <= rear;i++)
                printf("\t %d", queue[i]);
        }
}
```

Kuyrukların Bağlantılı Gösterimi

- Bir dizinin nasıl oluşturulduğunu gördük.
- Bu kuyruk oluşturma tekniği kolay olmasına rağmen, dezavantajı dizinin sabit bir boyuta sahip olacak şekilde bildirilmesi gerekliliğidir.
- Kuyrukta 50 elemana yer ayırdığımızda ve bu yer 20-25 yeri zor kullandığında, o zaman alanın yarısı israf olacaktır.
- Ve eğer giderek büyüyebilecek bir kuyruk için daha az bellek konumu ayırırsak, o zaman çok sayıda yeniden ayırma işlemi yapılması gerekecek, bu da çok fazla yük yaratacak ve çok fazla zaman tüketecektir.
- Eğer kuyruk çok küçükse veya maksimum boyutu önceden biliniyorsa, kuyruğun dizi uygulaması verimli bir uygulama verir.
- Ancak dizinin boyutu önceden belirlenemiyorsa, diğer alternatif, yani bağlantılı gösterim kullanılır.

Kuyrukların Bağlantılı Gösterimi

- n elemanlı bir kuyruğun bağlantılı gösteriminin depolama gereksinimi O(n)'dir ve işlemler için tipik zaman gereksinimi O(1)'dir.
- Bağlantılı kuyrukta her elemanın iki bölümü vardır; biri verileri depolar, diğeri ise bir sonraki elemanın adresini depolar.
- Bağlı listenin START işaretçisi FRONT olarak kullanılır.
- Burada, kuyruktaki son elemanın adresini saklayacak olan REAR adlı başka bir işaretçiyi de kullanacağız.
- Tüm eklemeler arka uçtan, tüm silmeler ise ön uçtan yapılacaktır.
- Eğer FRONT = REAR = NULL ise bu kuyruğun boş olduğunu gösterir.
- Bir kuyruğun bağlantılı gösterimi Şekil 8.6'da gösterilmiştir.

Kuyrukların Bağlantılı Gösterimi

- Bağlantılı Kuyruklarda İşlemler
- Bir kuyruğun iki temel işlemi vardır: ekleme ve silme.
- Ekleme işlemi, bir elemanı kuyruğun sonuna ekler ve silme işlemi, bir elemanı kuyruğun başından veya başından kaldırır.
- Bunun dışında kuyruğun ilk elemanının değerini döndüren bir peek işlemi daha var.
- Ekleme İşlemi
- Ekleme işlemi bir elemanı bir kuyruğa eklemek için kullanılır.
- Yeni eleman kuyruğun son elemanı olarak eklenir.
- Şekil 8.7'de gösterilen bağlantılı kuyruğu ele alalım.

Kuyrukların Bağlantılı Gösterimi

- Değeri 9 olan bir eleman eklemek için öncelikle FRONT=NULL olup olmadığını kontrol ederiz.
- Eğer koşul sağlanıyorsa kuyruk boştur.
- Yani yeni bir node için bellek ayırıyoruz, DATA kısmına değeri, NEXT kısmına ise NULL değerini depoluyoruz.
- Yeni düğüm daha sonra hem FRONT hem de REAR olarak adlandırılacaktır.
- Ancak, eğer FRONT != NULL ise, o zaman yeni düğümü bağlı kuyruğun arka ucuna ekleyeceğiz ve bu yeni düğüme REAR adını vereceğiz.
- Böylece güncellenen kuyruk Şekil 8.8'deki gibi olur.

Kuyrukların Bağlantılı Gösterimi

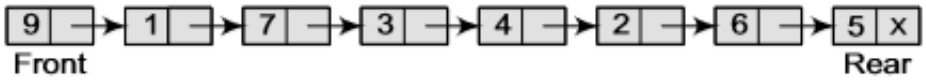

Figure 8.6 Linked queue

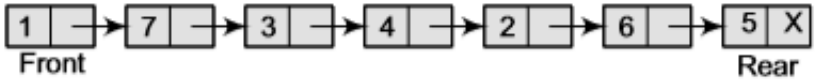

Figure 8.7 Linked queue

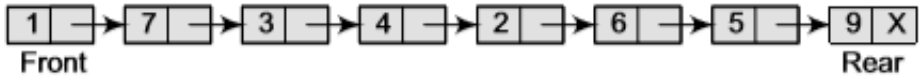

Figure 8.8 Linked queue after inserting a new node

```
Step 1: Allocate memory for the new node and name
        it as PTR
Step 2: SET PTR -> DATA = VAL
Step 3: IF FRONT = NULL
            SET FRONT = REAR = PTR
            SET FRONT -> NEXT = REAR -> NEXT = NULL
        ELSE
            SET REAR -> NEXT = PTR
            SET REAR = PTR
            SET REAR -> NEXT = NULL
        [END OF IF]
Step 4: END
```

Figure 8.9 Algorithm to insert an element in a linked queue

Kuyrukların Bağlantılı Gösterimi

- Şekil 8.9, bağlantılı bir kuyruğa bir eleman eklemek için kullanılan algoritmayı göstermektedir.
- 1. Adımda yeni düğüm için bellek tahsis edilir.
- Adım 2'de yeni düğümün DATA kısmı, düğümde depolanacak değerle başlatılır.
- 3. Adımda yeni düğümün bağlı kuyruğun ilk düğümü olup olmadığını kontrol ediyoruz.
- Bu, FRONT = NULL olup olmadığını kontrol ederek yapılır. Eğer durum buysa, yeni düğüm FRONT ve REAR olarak etiketlenir.
- Ayrıca düğümün NEXT kısmında (aynı zamanda FRONT ve REAR düğümüdür) NULL saklanır.
- Ancak, yeni düğüm listedeki ilk düğüm değilse, bağlı kuyruğun ARKA ucuna (veya kuyruğun son düğümüne) eklenir.

Kuyrukların Bağlantılı Gösterimi

- Silme İşlemi
- Silme işlemi, bir kuyruğa ilk eklenen öğeyi, yani adresi FRONT'ta saklanan öğeyi silmek için kullanılır.
- Ancak değeri silmeden önce FRONT=NULL olup olmadığını kontrol etmeliyiz çünkü eğer durum buysa kuyruk boştur ve daha fazla silme işlemi yapılamaz.
- Zaten boş olan bir kuyruktan bir değeri silmeye çalışılırsa, bir alt taşma mesajı yazdırılır.
- Şekil 8.10'da gösterilen kuyruğu ele alalım.

Kuyrukların Bağlantılı Gösterimi

- Silme İşlemi
- Bir elemanı silmek için öncelikle FRONT=NULL olup olmadığını kontrol ederiz.
- Eğer koşul yanlış ise FRONT'un işaret ettiği ilk düğümü sileriz.
- FRONT artık bağlı kuyruğun ikinci elemanına işaret edecektir.
- Böylece güncellenen kuyruk Şekil 8.11'deki gibi olur.

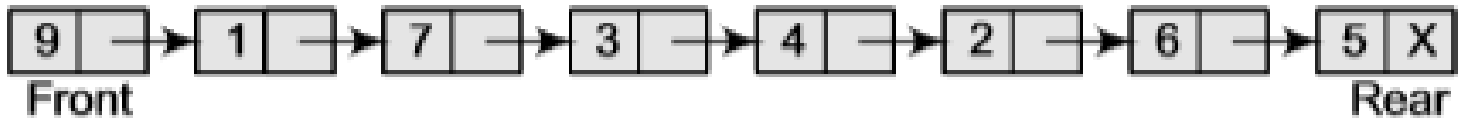

Figure 8.10 Linked queue

Figure 8.11 Linked queue after deletion of an element

Kuyrukların Bağlantılı Gösterimi

- Silme İşlemi
- Şekil 8.12, bağlantılı bir kuyruktan bir öğeyi silmek için kullanılan algoritmayı göstermektedir.
- 1. Adımda öncelikle alt taşma koşulunu kontrol ediyoruz.
- Eğer koşul doğru ise uygun bir mesaj görüntülenir, aksi takdirde 2. Adımda FRONT'u işaret eden bir PTR işaretçisi kullanırız.
- Adım 3'te, FRONT'un sıradaki düğüme işaret etmesi sağlanır. Adım 4'te, PTR tarafından işgal edilen bellek boş havuza geri verilir.

```
Step 1: IF FRONT = NULL
            Write "Underflow"
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = FRONT
Step 3: SET FRONT = FRONT -> NEXT
Step 4: FREE PTR
Step 5: END
```

Figure 8.12 Algorithm to delete an element from a linked queue

Kuyrukların Bağlantılı Gösterimi

```
struct node
{
     int data;
     struct node *next;
};
struct queue
{
     struct node *front;
     struct node *rear;
};
struct queue *q;
void create_queue(struct queue *);
struct queue *insert(struct queue *,int);
struct queue *delete_element(struct queue *);
struct queue *display(struct queue *);
int peek(struct queue *);
```

Kuyrukların Bağlantılı Gösterimi

```
void create_queue(struct queue *q)
{
     q->rear = NULL;
     q->front = NULL;
}
struct queue *insert(struct queue *q,int val)
{
     struct node *ptr;
     ptr = (struct node*)malloc(sizeof(struct node));
     ptr->data = val;
     if(q->front == NULL)
     {
          q->front = ptr;
          q->rear = ptr;
          q->front->next = q->rear->next = NULL;
     }
     else
     {
          q->rear->next = ptr;
          q->rear = ptr;
          q->rear->next = NULL;
     }
     return q;
}
```

Kuyrukların Bağlantılı Gösterimi

```
struct queue *display(struct queue *q)
{
     struct node *ptr;
     ptr = q->front;
     if(ptr == NULL)
          printf("\n QUEUE IS EMPTY");
     else
     {
          printf("\n");
          while(ptr!=q->rear)
          {
               printf("%d\t", ptr->data);
               ptr = ptr->next;
          }
          printf("%d\t", ptr->data);
     }
     return q;
}
```

Kuyrukların Bağlantılı Gösterimi

```
struct queue *delete_element(struct queue *q)
{
     struct node *ptr;
     ptr = q->front;
     if(q->front == NULL)
          printf("\n UNDERFLOW");
     else
     {
          q->front = q->front->next;
          printf("\n The value being deleted is : %d", ptr->data);
          free(ptr);
     }
     return q;
}
int peek(struct queue *q)
{
     if(q->front==NULL)
     {
          printf("\n QUEUE IS EMPTY");
          return -1;
     }
     else
          return q->front->data;
}
```

Kuyruk Türleri

- Bir kuyruk veri yapısı aşağıdaki tiplere sınıflandırılabilir:
- 1. Dairesel Sıra
- 2. Bu nedenle
- 3. Öncelikli Sıra
- 4. Çoklu Kuyruk
- Bu kuyrukların her birini aşağıdaki bölümlerde ayrıntılı olarak ele alacağız.

Dairesel Kuyruklar

- Doğrusal kuyruklarda, eklemelerin yalnızca REAR adı verilen bir uçtan yapılabildiğini ve silmelerin her zaman FRONT adı verilen diğer uçtan yapıldığını şimdiye kadar tartıştık.
- Şekil 8.13'te görülen kuyruğa bakın.
- Burada ÖN = 0 ve ARKA = 9.

13 Linear queue

Dairesel Kuyruklar

- Şimdi başka bir değer girmek isterseniz bu mümkün olmayacaktır çünkü kuyruk tamamen dolmuştur.
- Değerin girilebileceği boş alan bulunmamaktadır.
- İki ardışık silmenin yapıldığı bir senaryoyu ele alalım. Kuyruk daha sonra Şekil 8.14'te gösterildiği gibi verilecektir.
- Burada ÖN = 2 ve ARKA = 9.

Figure 8.14 Queue after two successive deletions

Dairesel Kuyruklar

- Şekil 8.14'te gösterilen kuyruğa yeni bir eleman eklemek istediğimizi varsayalım.
- Boş alan olmasına rağmen, taşma koşulu hala mevcuttur çünkü REAR = MAX – 1 koşulu hala geçerlidir.
- Bu, doğrusal bir kuyruğun önemli bir dezavantajıdır. Bu sorunu çözmek için iki çözümümüz var.
- Öncelikle elemanları sola kaydırarak boş alanın verimli bir şekilde doldurulmasını ve kullanılmasını sağlayın.
- Ancak bu çok zaman alıcı olabilir, özellikle de sıra oldukça büyük olduğunda. İkinci seçenek dairesel bir sıra kullanmaktır.
- Dairesel sırada ilk endeks son endeksin hemen ardından gelir.
- Kavramsal olarak dairesel kuyruğu Şekil 8.15'te gösterildiği gibi düşünebilirsiniz.
- Dairesel sıra yalnızca FRONT = 0 ve REAR = Max – 1 olduğunda dolacaktır.
- Dairesel kuyruk, doğrusal kuyruk gibi aynı şekilde uygulanır.
- Tek fark, ekleme ve silme işlemlerini yapan kodda olacaktır.

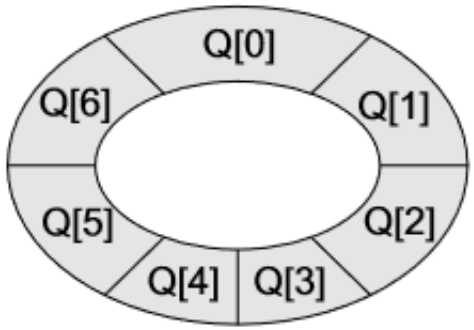

Figure 8.15 Circular queue

Dairesel Kuyruklar

- **Ekleme için şimdi aşağıdaki üç koşulu kontrol etmemiz gerekiyor:**
- **Eğer FRONT = 0 ve REAR = MAX – 1 ise, dairesel kuyruk doludur. Bu noktayı gösteren Şekil 8.16'da verilen kuyruğa bakın.**
- **Eğer REAR != MAX – 1 ise, REAR artırılacak ve değer Şekil 8.17'de gösterildiği gibi eklenecektir.**
- **Eğer FRONT != 0 ve REAR = MAX – 1 ise, bu kuyruğun dolu olmadığı anlamına gelir. Bu yüzden, REAR = 0 olarak ayarlayın ve Şekil 8.18'de gösterildiği gibi yeni elemanı oraya ekleyin.**

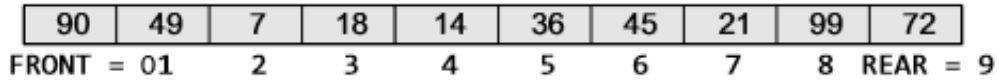

Figure 8.16 Full queue

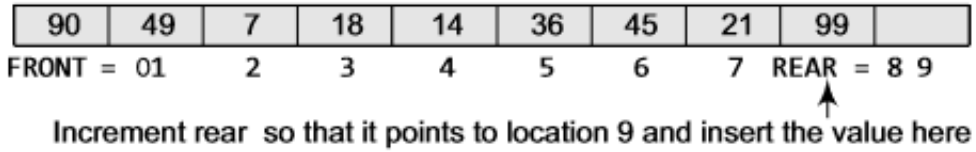

Figure 8.17 Queue with vacant locations

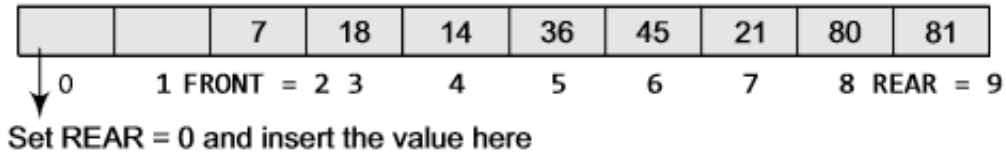

Figure 8.18 Inserting an element in a circular queue

Dairesel Kuyruklar

- Dairesel bir kuyruğa eleman ekleme algoritmasını gösteren Şekil 8.19'a bakalım.
- Adım 1'de taşma koşulunu kontrol ediyoruz. Adım 2'de iki kontrol yapıyoruz.
- İlk olarak kuyruğun boş olup olmadığına, ikinci olarak da ÖN tarafın önünde belirli sayıda boş yer varken ARKA tarafın maksimum kapasiteye ulaşıp ulaşmadığına bakılır.
- Adım 3'te değer, REAR'ın işaret ettiği konumdaki kuyrukta saklanır.

```
Step 1: IF FRONT = 0 and Rear = MAX - 1
            Write "OVERFLOW"
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Figure 8.19 Algorithm to insert an element in a circular queue

Dairesel Kuyruklar

```
void insert()
{
        int num;
        printf("\n Enter the number to be inserted in the queue : ");
        scanf("%d", &num);
        if(front==0 && rear==MAX-1)
                printf("\n OVERFLOW");
        else if(front==-1 && rear==-1)
        {
                front=rear=0;
                queue[rear]=num;
        }
        else if(rear==MAX-1 && front!=0)
        {
                rear=0;
                queue[rear]=num;
        }
        else
        {
                rear++;
                queue[rear]=num;
        }
}
```

Dairesel Kuyruklar

```
int delete_element()
{
        int val;
        if(front==-1 && rear==-1)
         {
                printf("\n UNDERFLOW");
                return -1;
         }
        val = queue[front];
        if(front==rear)
                front=rear=-1;
        else
        {
                if(front==MAX-1)
                        front=0;
                else
                        front++;
        }
        return val;
}
```

Kuyruklar

- Deque (telaffuzu ‘deck’ veya ‘dequeue’), elemanların her iki ucundan eklenebildiği veya silinebildiği bir listedir.
- Baş-kuyruk bağlı liste olarak da bilinir çünkü elemanlar listenin ön (baş) veya arka (kuyruk) ucuna eklenebilmekte veya çıkarılabilmektedir.
- Ancak ortada hiçbir eleman eklenemez ve silinemez. Bilgisayarın belleğinde, bir deque dairesel bir dizi veya dairesel çift bağlantılı bir liste kullanılarak uygulanır.
- Bir deque'de, deque'nin her iki ucunu işaret eden LEFT ve RIGHT adında iki işaretçi tutulur.
- Bir deque'deki elemanlar SOL uçtan SAĞ uca kadar uzanır ve dairesel olduğundan Dequeue[N–1]'i Dequeue[0] takip eder.
- Şekil 8.24'te gösterilen deque'leri göz önünde bulundurun.

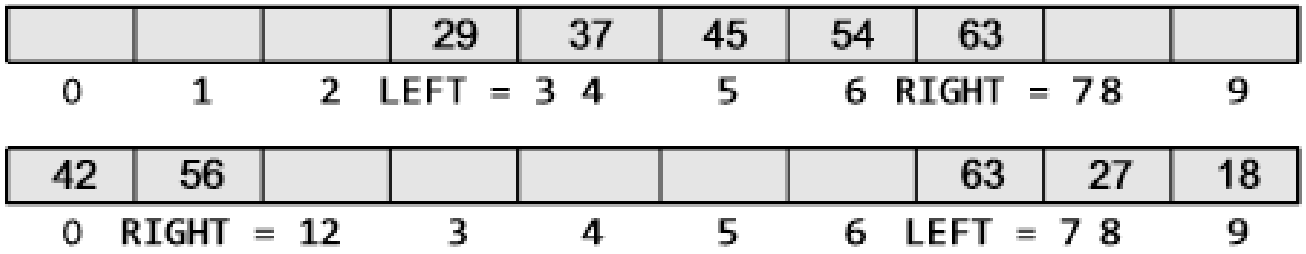

Figure 8.24 Double-ended queues

Kuyruklar

- Çift uçlu kuyruğun iki çeşidi vardır. Bunlar şunları içerir:
- Giriş kısıtlı kuyruktan çıkarma: Bu kuyruktan çıkarmada, eklemeler yalnızca uçlardan birinde yapılabilirken, silmeler her iki uçtan da yapılabilir.
- Çıktı kısıtlı kuyruktan çıkarma: Bu kuyruktan çıkarmada, silmeler yalnızca uçlardan birinde yapılabilirken, eklemeler her iki uçta da yapılabilir.

Öncelikli Kuyruklar

- Öncelik kuyruğu, her bir elemana bir öncelik atanan bir veri yapısıdır.
- Elemanların hangi sırayla işleneceğini belirlemek için elemanın önceliği kullanılacaktır.
- Öncelikli kuyruğun öğelerinin işlenmesine ilişkin genel kurallar şunlardır:
- Daha yüksek önceliğe sahip bir öğe, daha düşük önceliğ sahip bir öğeden önce işlenir.
- Aynı önceliğe sahip iki öğe, ilk gelen ilk hizmet esasına göre işlenir.

Öncelikli Kuyruklar

- Öncelikli bir kuyruk, bir öğenin kuyruktan çıkarılması gerektiğinde en yüksek önceliğe sahip olanın ilk alındığı değiştirilmiş bir kuyruk olarak düşünülebilir. Öğenin önceliği çeşitli faktörlere göre ayarlanabilir.
- Öncelik kuyrukları, işletim sistemlerinde en yüksek önceliğe sahip işlemleri ilk yürütmek için yaygın olarak kullanılır.
- İşlemin önceliği, tamamen yürütülmesi için ihtiyaç duyduğu CPU zamanına göre ayarlanabilir.
- Örneğin, ilk işlemin tamamlanması için 5 ns, ikinci işlemin tamamlanması için 4 ns ve üçüncü işlemin tamamlanması için 7 ns'ye ihtiyaç duyan üç işlem varsa, ikinci işlem en yüksek önceliğe sahip olacak ve dolayısıyla ilk yürütülecek işlem olacaktır.
- Ancak CPU zamanı önceliği belirleyen tek faktör değildir, aksine birçok faktörden sadece biridir.
- Bir diğer faktör ise bir sürecin diğerine göre önemidir. Aynı anda iki süreci çalıştırmamız gerekirse, bir süreç çevrimiçi sipariş rezervasyonuyla ilgiliyse ve ikincisi stok detaylarının yazdırılmasıyla ilgiliyse, o zaman çevrimiçi rezervasyon daha önemlidir ve önce yürütülmelidir.

Öncelikli Kuyruklar

- Öncelikli Kuyruğun Uygulanması
- Öncelik kuyruğunu uygulamanın iki yolu vardır.
- Elemanları saklamak için sıralı bir liste kullanabiliriz, böylece bir eleman çıkarıldığında kuyrukta en yüksek önceliğe sahip elemanı aramak zorunda kalmayız ya da eklemelerin her zaman listenin sonuna yapılmasını sağlamak için sıralanmamış bir liste kullanabiliriz.
- Listeden bir elemanın çıkarılması gerektiğinde, en yüksek önceliğe sahip eleman aranacak ve çıkarılacaktır.
- Sıralanmış bir listenin listeye bir eleman eklemesi O(n) zaman alırken, bir elemanı silme işlemi yalnızca O(1) zaman alır.
- Buna karşılık, sıralanmamış bir listenin bir öğe eklemesi O(1) zaman, listeden bir öğeyi silmesi ise O(n) zaman alacaktır.
- Pratikte, bu tekniklerin her ikisi de verimsizdir ve genellikle bu iki yaklaşımın bir karışımı benimsenir ve bu da yaklaşık O(log n) zaman veya daha az sürer.

Öncelikli Kuyruklar

- Öncelikli Kuyruğun Bağlantılı Gösterimi
- Bilgisayar belleğinde, bir öncelik kuyruğu diziler veya bağlı listeler kullanılarak temsil edilebilir.
- Bir öncelikli kuyruk, bağlantılı liste kullanılarak uygulandığında, listenin her düğümü üç bölümden oluşur: (a) bilgi veya veri bölümü, (b) öğenin öncelik numarası ve (c) sonraki öğenin adresi.
- Sıralı bir bağlı liste kullanıyorsak, önceliği daha yüksek olan eleman, önceliği daha düşük olan elemandan önce gelir.
- Şekil 8.25'te gösterilen öncelikli kuyruğu ele alalım.

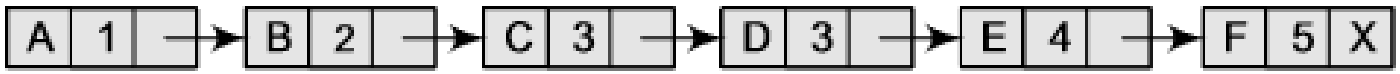

Figure 8.25 Priority queue

Öncelikli Kuyruklar

- Öncelik numarası ne kadar düşükse öncelik o kadar yüksek demektir.
- Örneğin, A ve B olmak üzere iki öğe varsa ve A'nın öncelik numarası 1, B'nin öncelik numarası 5 ise, A, B'den daha yüksek önceliğe sahip olduğundan B'den önce işlenecektir.
- Şekil 8.25'teki öncelik kuyruğu, altı elemandan oluşan sıralı bir öncelik kuyruğudur.
- Sıradan bakıldığında A'nın E'den önce eklenip eklenmediğini veya E'nin A'dan önce sıraya girip girmediğini anlayamıyoruz çünkü liste FCFS'ye göre sıralanmamış.
- Burada, daha yüksek önceliğe sahip öğe, daha düşük öncelig sahip öğeden önce gelir.
- Ancak C'nin D'den önce kuyruğa eklendiğini kesin olarak söyleyebiliriz çünkü iki eleman aynı önceliğe sahip olduğunda elemanlar FCFS prensibine göre düzenlenir ve işlenir.

Öncelikli Kuyruklar

- Ekleme Öncelik sırasına yeni bir eleman eklenmesi gerektiğinde, yeni elemanın önceliğinden daha düşük önceliğe sahip bir düğüm bulana kadar tüm listeyi taramamız gerekir.
- Yeni düğüm, daha düşük önceliğe sahip düğümün önüne eklenir.
- Ancak yeni elemanla aynı önceliğe sahip bir eleman varsa, yeni eleman o elemandan sonra eklenir.
- Örneğin, Şekil 8.26'da gösterilen öncelikli kuyruğu ele alalım.

Figure 8.26 Priority queue

Öncelikli Kuyruklar

- Eğer data = F ve öncelik numarası = 4 olan yeni bir eleman eklememiz gerekirse, bu eleman öncelik numarası 5 olan D'den önce eklenecektir; bu, yeni elemanın önceliğinden daha düşük bir önceliğe sahiptir.
- Böylece öncelik sırası Şekil 8.27'deki gibi olur.

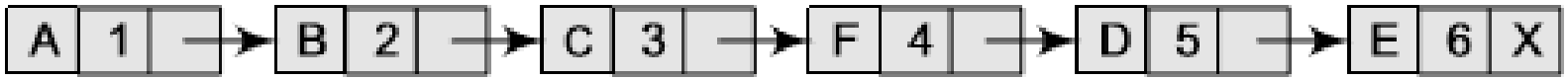

Figure 8.27 Priority queue after insertion of a new node

Öncelikli Kuyruklar

- Ancak, data = F ve öncelik numarası = 2 olan yeni bir elemanımız varsa, o zaman eleman B'den sonra eklenecektir, çünkü bu iki elemanın da önceliği aynıdır ancak eklemeler Şekil 8.28'de gösterildiği gibi FCFS temelinde yapılır.

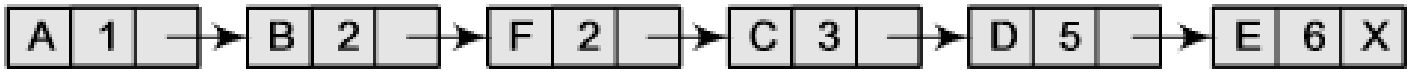

Figure 8.28 Priority queue after insertion of a new node

- Silme: Bu durumda silme işlemi çok basit bir işlemdir. Listenin ilk düğümü silinecek ve o düğümün verileri ilk önce işlenecektir.

Öncelikli Kuyruklar

- Öncelik kuyruğunu uygulamak için diziler kullanıldığında, her öncelik numarası için ayrı bir kuyruk tutulur.
- Bu kuyrukların her biri dairesel diziler veya dairesel kuyruklar kullanılarak uygulanacaktır.
- Her bir kuyruğun kendine ait ÖN ve ARKA göstergeleri olacak.
- Bu amaçla her kuyruğa aynı miktarda alanın tahsis edileceği iki boyutlu bir dizi kullanıyoruz.
- Aşağıda verilen öncelik sırasının iki boyutlu gösterimine bakın.
- Her kuyruğun FRONT ve REAR değerleri verildiğinde, Şekil 8.29'da gösterildiği gibi iki boyutlu matris oluşturulabilir.

FRONT	REAR
3	3
1	3
4	5
4	1

	1	2	3	4	5
1			A		
2	B	C	D		
3				E	F
4	I			G	H

Figure 8.29 Priority queue matrix

Öncelikli Kuyruklar

- FRONT[K] ve REAR[K], K satırının ön ve arka değerlerini içerir; burada K öncelik numarasıdır.
- Burada satır ve sütun indekslerinin 0'dan değil 1'den başladığını varsaydığımızı unutmayın.
- Elbette programlama yaparken bu tür varsayımlarda bulunmayacağız.
- Ekleme
- Öncelik kuyruğuna K önceliğine sahip yeni bir öğe eklemek için, öğeyi K satırının son ucuna ekleyin; burada K, satır numarası ve o öğenin öncelik numarasıdır.
- Örneğin, öncelik numarası 3 olan bir R elemanı eklememiz gerekirse, öncelik sırası Şekil 8.30'da gösterildiği gibi verilecektir.

FRONT	REAR
3	3
1	3
4	1
4	1

	1	2	3	4	5
1			A		
2	B	C	D		
3	R			E	F
4	I			G	H

Figure 8.30 Priority queue matrix after insertion of a new element

Öncelikli Kuyruklar

- Silme
- Bir elemanı silmek için, ilk boş olmayan kuyruğu buluruz ve ardından ilk boş olmayan kuyruğun ön elemanını işleriz.
- Öncelikli kuyruğumuzda ilk boş olmayan kuyruğun önceliği 1 olan kuyruktur ve ön elemanı A'dır, dolayısıyla A silinecek ve önce işlenecektir.
- Teknik terimlerle, FRONT[K] != NULL olacak şekilde en küçük K'ye sahip öğeyi bulun.

Öncelikli Kuyruklar

```
struct node *insert(struct node *start)
{
        int val, pri;
        struct node *ptr, *p;
        ptr = (struct node *)malloc(sizeof(struct node));
        printf("\n Enter the value and its priority : " );
        scanf( "%d %d", &val, &pri);
        ptr->data = val;
        ptr->priority = pri;
        if(start==NULL || pri < start->priority )
        {
                ptr->next = start;
                start = ptr;
        }
        else
        {
                p = start;
                while(p->next != NULL && p->next->priority <= pri)
                        p = p->next;
                ptr->next = p->next;
                p->next = ptr;
        }
        return start;
}
```

Öncelikli Kuyruklar

```
struct node *delete(struct node *start)
{
        struct node *ptr;
        if(start == NULL)
        {
                printf("\n UNDERFLOW" );
                return;
        }
        else

        {
                ptr = start;
                printf("\n Deleted item is: %d", ptr->data);
                start = start->next;
                free(ptr);
        }
        return start;
}
```

Öncelikli Kuyruklar

```
void display(struct node *start)
{
        struct node *ptr;
        ptr = start;
        if(start == NULL)
                printf("\nQUEUE IS EMPTY" );
        else
        {
                printf("\n PRIORITY QUEUE IS : " );
                while(ptr != NULL)
                {
                        printf( "\t%d[priority=%d]", ptr->data, ptr->priority );
                        ptr=ptr->next;
                }
        }
}
```

Çoklu Kuyruklar

- Bir dizi kullanarak kuyruk uyguladığımızda dizinin boyutunun önceden bilinmesi gerekir.
- Eğer kuyruğa daha az alan tahsis edilirse, sık sık taşma durumuyla karşılaşılacaktır.
- Bu sorunla başa çıkabilmek için, diziye daha fazla alan tahsis etmek amacıyla kodun değiştirilmesi gerekecektir.
- Kuyruk için çok fazla alan ayırmamız durumunda, bu durum tamamen hafıza israfına yol açacaktır.
- Dolayısıyla taşma sıklığı ile ayrılan alan arasında bir denge söz konusudur.
- Dolayısıyla bu problemle başa çıkmanın daha iyi bir çözümü, birden fazla kuyruğa sahip olmak veya aynı dizide yeterli büyüklükte birden fazla kuyruğa sahip olmaktır.
- Şekil 8.31 bu kavramı göstermektedir.

Çoklu Kuyruklar

- Şekilde, QUEUE A ve QUEUE B olmak üzere iki kuyruğu temsil etmek için QUEUE[n] dizisi kullanılmıştır.
- n değeri, her iki kuyruğun toplam boyutunun hiçbir zaman n'yi aşmayacağı şekildedir.
- Bu kuyruklarda işlem yaparken bir noktaya dikkat etmek önemlidir: KUYRUK A soldan sağa doğru büyürken, KUYRUK B aynı anda sağdan sola doğru büyüyecektir.
- Kavramı birden fazla kuyruğa genişleterek, bir kuyruk aynı dizideki n sayıda kuyruğu temsil etmek için de kullanılabilir.
- Yani, eğer bir QUEUE[n] varsa, o zaman her QUEUE I'e b[i] ve e[i] indeksleriyle sınırlanmış eşit miktarda alan tahsis edilecektir.
- Bu durum Şekil 8.32'de gösterilmiştir.

Çoklu Kuyruklar

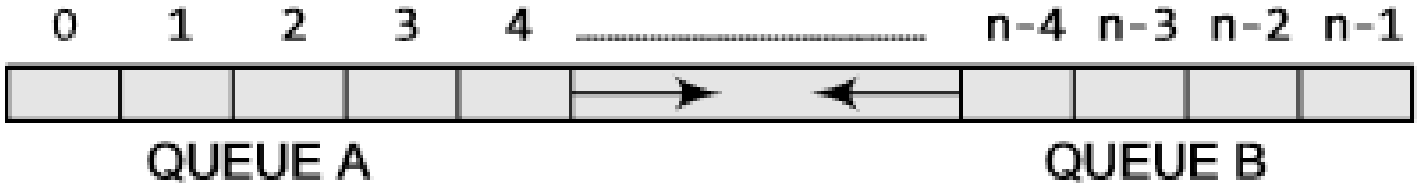

Figure 8.31 Multiple queues

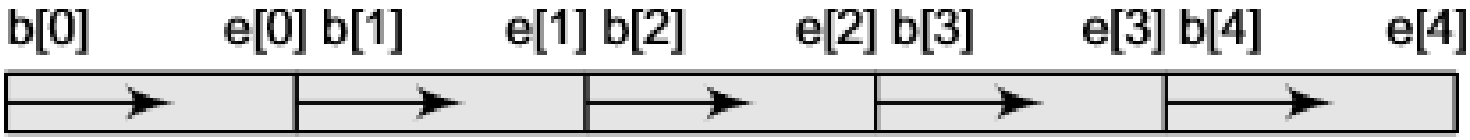

Figure 8.32 Multiple queues