Algoritma Analizi

Ders 10

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Temel veri yapıları

- Bu bölümde dinamik olarak değişen eleman kümelerinin gösteriminde kullanılan doğrusal veri yapılarını göreceğiz.
- Dinamik yapılarda belli işlemler yapılabilir.
  - Search(S,k), Insert(S,x), Delete(S,x), Minimum(S), Maximum(S), Successor(S,x), Predecessor(S,x).
- Bahsedeceğimiz veri yapıları:
  - Yığın (stack)
  - Kuyruk (queue)
  - Bağlı liste (Linked list)

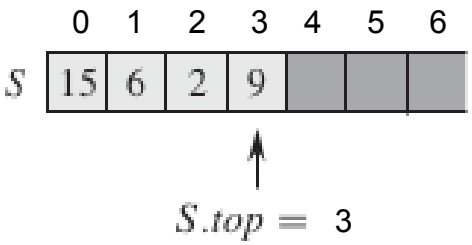
Yığın ve kuyruk yapısı

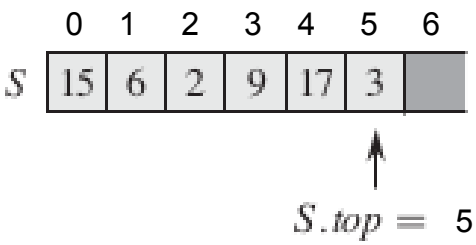
- Yığın ve kuyruk yapılarında delete (sil) işlemi ile yapıdan silinecek olan eleman önceden belirlidir.
  - Yığın yapısında silinecek olan eleman yığına en son eklenen elemandır.
  - Bu sebeple yığın yapısı “son eklenen ilk çıkar“ (LIFO) kuralına göre çalışır.
  - Kuyruk yapısında silenecek eleman sıraya ilk eklenen elemandır.
  - Bu sebeple kuyruk yapısı “ilk eklenen ilk çıkar“ (FIFO) kuralına göre çalışır.

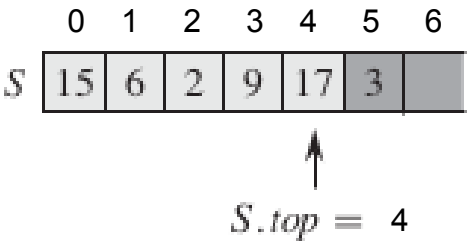
Yığın

- Yığın yapısında aşağıdaki iki işlem yapılır:
  - Yığına eleman ekleme, genellikle push terimi ile adlandılır.
  - Yığından eleman çıkarma, genellikle pop terimi ile adlandırılır.
- Toplamda en fazla n elemana sahip olabilecek bir yığın S[0..n-1] dizi yapısı kullanarak şu şekilde oluşturabilinir.
  - S.top yığına en son eklenen elemanın pozisyonudur.
  - Yığındaki elemalar S[0..S.top].
  - En alttaki eleman S[0].
  - En üstteki eleman S[S.top].

Yığın


0 1 2 3 4 5 6
S
15 6 2 9
S.top = 3


0 1 2 3 4 5 6
S
15 6 2 9 17 3
S.top = 5


0 1 2 3 4 5 6
S
15 6 2 9 17 3
S.top = 4

Yığın

- S.top = -1 olduğunda, yığın boş anlamına gelmektedir.
- S.top -1 olduğunda pop yapılırsa aşağı taşma, yığın dolduğunda yeni bir eleman push yapılır ise yukarı taşma durumu oluşur.
- Yığın üzerinde yapılacak işlemler bir sonraki slaytta gösterilmiştir.
- Bu işlemlerin her biri O(1) süre içerisinde olur.
- Yığın yapısı diziler veya bağlı listeler üzerinden oluşturulabilir.
  - Bu aşamada diziler üzerinden oluşturulmuş bir yığın yapısını inceleyeceğiz.
  - Bağlı listeler konusuna geldiğimizde bağlı listeler kullanılarak yığın yapısı oluşturabileceğimizi göreceğiz.

Yığın

- C dilinde yığın yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Yığın yapısının verisini saklamak için bir struct oluşturulmalıdır.

```
typedef struct stack
{
    int capacity;
    int top;
    int* items;
}stack;
```

  - capacity parametresi yığının maksimum büyüklüğünü belirler.
  - top parametresi yığının en üstünde bulunan elemanın indeksini belirler.
  - items dizisi yığında bulunan elemanları içerir.

Yığın

- C dilinde yığın yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Yığını oluşturmak ve parametreleri ayarlamak için:

```
stack* NewStack(int size)
{
     stack *pt = (stack*)malloc(sizeof(stack));

     pt->capacity = size;
     pt->top = -1;
     pt->items = (int*)malloc(sizeof(int) * size);

     return pt;
}
```

Yığın

- C dilinde yığın yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Yığının boş veya tamamiyle dolu olduğunu tespit etmek için:

```
int StackEmpty(stack *pt)
{
    return (pt->top == -1);
}

int StackFull(stack *pt)
{
    return (pt->top == pt->capacity-1);
}
```

Yığın

- C dilinde yığın yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Yığına yeni bir eleman eklemek için:

```
void Push(int element, stack* pt)
{
    if(StackFull(pt))
    {
        printf("Yigin doldu! Yeni eleman eklenemez!\n");
    }
    else
    {
        pt->top++;
        pt->items[pt->top] = element;
        printf("%d elemani eklendi\n",element);
    }
}
```

Yığın

- C dilinde yığın yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Yığından bie eleman çıkarmak için:

```
int Pop(stack* pt)
{
    if(StackEmpty(pt))
    {
        printf("Yigin bos!\n");
        exit(-1);
    }
    else
    {
        pt->top--;
        printf("%d elemani siliniyor.\n",pt->items[pt->top+1]);
        return pt->items[pt->top+1];
    }
}
```

Yığın

- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - C programlarında birbirini çağıran fonksiyonlar bulunduğunda bu fonksiyonların değişken ve dönüş yerleri yığın yapısında saklanır.
  - Web istemcilerinde “Geri“ tuşuna basıldığında önceki sayfaya dönebilmemiz için ziyaret edilen sayfalar yığın yapısında saklanır.
  - Birçok uygulamada “Geri al“ (Undo) operasyonu yığın kullanılarak yapılır.
    - Yapılan operasyonların her biri yığın yapısına eklenir.

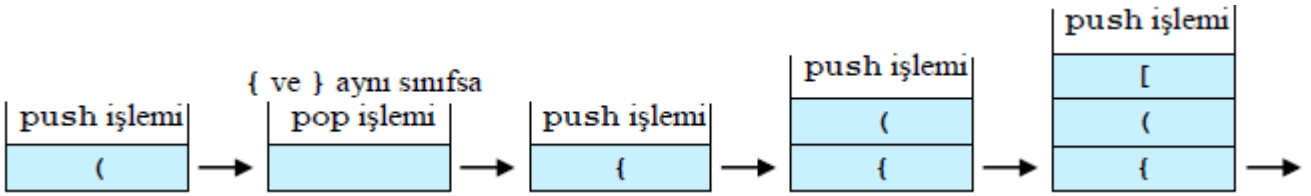
Yığın

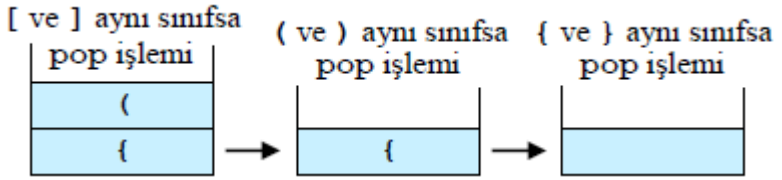
- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - C ve C++ dillerinde parantez kontrolü yığın yapısı ile yapılabilir.
    - C dilinde (, [, veya { gibi farklı parantez türleri kullanılır.
    - Bunların doğru sayıda ve sıra ile dizilmesi C dilinin syntax kurallarında belirtilmiştir.
    - Örneğin [()] Kabul edilebilir bir parantez sıralaması iken [(]) değildir.
    - Bu amaçla aşağıdaki yöntem kullanılabilir:
      - 1- Boş bir yığın oluştur.
      - 2- Dosyadaki karekterleri sonuna kadar teker teker oku.
        - 3- Eğer karakter bir parantez tipini açıyor ise yığına ekle (Push).
        - 4- Eğer karakter parantez tipini kapıyorsa
          - 5- Yığın boş ise hata bildir.
          - 6- Yığın boş değilse yığından eleman çıkar (Pop).
            - 7- Eğer çıkan eleman kapatılan parantezi açmıyosa, hata bildir.
      - 8- Dosya tamamlandığında yığın boş değilse hata bildir.

Yığın

- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - Cve C++ dillerinde parantez kontrolü yığın yapısı ile yapılabilir.

```
if(a == 2)
{
    printf("%d\n", dizi[2]);
}
```





Yığın

- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - Cve C++ dillerinde parantez kontrolü yığın yapısı ile yapılabilir.

```
if(i == 2)
{
    printf(“Merhaba Dunya}”);
```

Yığın

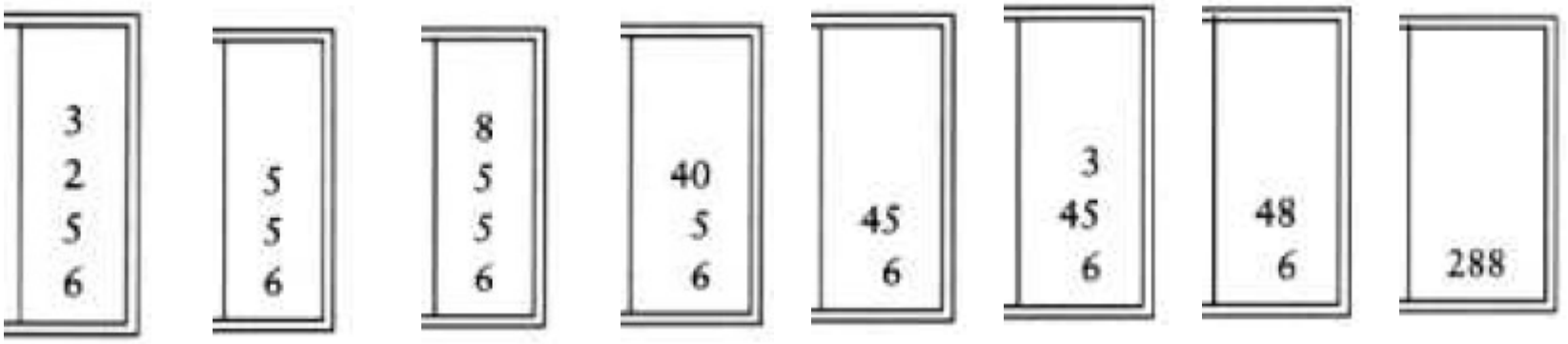
- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - Postfix notasyon ile yazılmış işlemlerin yapılması.
    - Aşağıdakine benzer bir işlem yaptığınızı farzedelim.
      - 4.99 * 1.06 + 5.99 + 6.99 * 1.06
    - Operatör öncelikleri dikkate alındığında sonuç 18.69 olmalı.
    - Operatör önceliklerine önem vermezseniz sonucu 19.37 hesaplayabilirsiniz.
    - Bu gösterime infix notasyonu denir.
    - Bu hesabı aşağıdaki şekilde yapabiliriz.
      - 4.99 ve 1.06'yı çarparak bu cevabı a1 olarak kaydederiz. Daha sonra 5.99 ve a1‘i toplayıp ve sonucu a1‘de saklarız. 6.99 ve 1.06'yı çarparız ve sonucu a2‘de kaydederiz. En son a1 ve a2‘yi toplayarak sonucu a1'de bırakırız. Bu işlem sırasını aşağıdaki gibi yazabiliriz:
        - 4.99 1.06 * 5.99 + 6.99 1.06 * +

Yığın

- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - Postfix notasyon ile yazılmış işlemlerin yapılması.
    - 4.99 1.06 * 5.99 + 6.99 1.06 * +
    - Bu gösterim postfix (veya reverse Polish) gösterim olarak bilinir ve tam olarak yukarıda açıkladığımız gibi değerlendirilir.
    - Bunu yapmanın en kolay yolu bir yığın kullanmaktır.
      - Bir sayı göründüğünde yığının üzerine itilir; bir operatör görüldüğünde, operatör yığından çıkan iki sayıya (simgelere) uygulanır ve sonuç yığına itilir.
      - Örnek: 6 5 2 3 + 8 * + 3 + *

Yığın

- Yığın birçok farklı durumda kullanılan bir veri yapısıdır.
  - Postfix notasyon ile yazılmış işlemlerin yapılması.
    - Örnek: 6 5 2 3 + 8 * + 3 + *



  - Ayrıca yığın yapısı kullanarak infix notasyonunda yazılan postfix notasyonuna çevirilebilir.

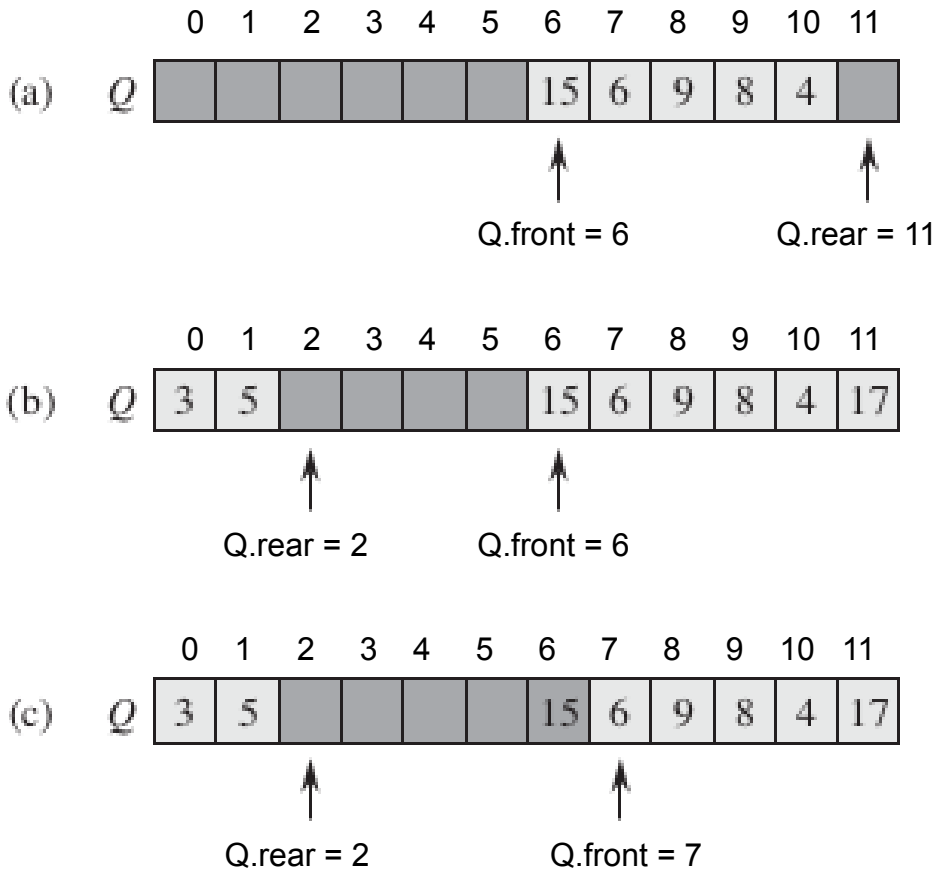
Kuyruk

- Kuyruk (veya sıra) yapısında aşağıdaki iki işlem yapılır:
  - Kuyruğa eleman ekleme, enqueue olarak adlandırılır.
  - Kuyruğa eleman çıkarma, dequeue olarak adlandırılır.
- Kuyruk yapısı ön (front) ve arka (rear) barındırır.
  - Arka, sıraya yeni eklenecek elemanın yerleştirileceği yerdir.
  - Ön, sıraya ilk eklenen elemanı işaret eder. kuyruktan eleman çıkarıldığında ön eleman sıradan ayrılır.

Kuyruk

- Kuyruk yapısı ön (front) ve arka (rear) barındırır.
  - En fazla n eleman içerebilecek Q[0..n-1] kuyruğu ile şu şekilde tanımlanabilir.
    - Q.front kuyruğa ilk eklenen elemanı işaret eder.
    - Q.rear kuyruğa yeni eklenecek elemanın yerleştirileceği pozisyonu işaret eder.
    - Sıradaki elemanlar Q.front, Q.front + 1. ... , Q.rear – 1 pozisyonlarıdır.
      - Elemanlar sirküler şekilde pozisyon alır.
  - Yeni bir eleman ekleme (enqueue) ve bir eleman çıkarma (dequeue) işlemleri O(1) süre alır.
  - Kuyruk yapısı diziler veya bağlı listeler üzerinden oluşturulabilir.
    - Bu aşamada diziler üzerinden oluşturulmuş bir kuyruk yapısını inceleyeceğiz.
    - Bağlı listeler konusuna geldiğimizde bağlı listeler kullanılarak kuyruk yapısı oluşturabileceğimizi göreceğiz.

Kuyruk


0 1 2 3 4 5 6 7 8 9 10 11
(a) Q
15 6 9 8 4
Q.front = 6
Q.rear = 11
0 1 2 3 4 5 6 7 8 9 10 11
(b) Q
3 5 15 6 9 8 4 17
Q.rear = 2
Q.front = 6
0 1 2 3 4 5 6 7 8 9 10 11
(c) Q
3 5 15 6 9 8 4 17
Q.rear = 2
Q.front = 7

Kuyruk

- C dilinde kuyruk yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - kuyruk yapısının verisini saklamak için bir struct oluşturulmalıdır.

```
typedef struct Queue {
    int front;
    int rear;
    int numberOfElements; //sıradaki eleman sayısı
    int capacity; //sıranın maksimum büyüklüğü
    int* array;
}Queue;
```

  - front parametresi ön indeksi belirtir.
  - rear parametresi arka indeksi belirtir.
  - numberOfElements parametresi kuyrukta bulunan eleman sayısını belirtir.
  - capacity parametresi kuyrukta bulunabilecek maksimum eleman sayısını belirtir.
  - array dizisi kuyruktaki elemanları içerir.

Kuyruk

- C dilinde kuyruk yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - kuyruk yapısını oluşturmak ve parametreleri ayarlamak için:

```
Queue* createQueue(int size)
{
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->capacity = size;
    queue->front = 0;
    queue->numberOfElements = 0;
    queue->rear = 0;
    queue->array = (int*)malloc(queue->capacity * sizeof(int));
    return queue;
}
```

Kuyruk

- C dilinde kuyruk yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Kuyruğun boş veya dolu olma durumunu kontrol edebilmek için:

```
int isFull(Queue* queue)
{
    return (queue->numberOfElements == queue->capacity);
}

int isEmpty(Queue* queue)
{
    return (queue->numberOfElements == 0);
}
```

Kuyruk

- C dilinde kuyruk yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Kuyruğa yeni bir eleman eklemek için:

```
void enqueue(Queue* queue, int item)
{
    if (isFull(queue))
    {
        printf("Ekleme yapilamaz, sira dolu!\n");
        return;
    }
    queue->array[queue->rear] = item;
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->numberOfElements = queue->numberOfElements + 1;
    printf("%d siraya eklendi\n", item);
}
```

Kuyruk

- C dilinde kuyruk yapısı oluşturabilmek için aşağıda belirtilen yapı ve fonksiyonlara ihtiyaç vardır:
  - Kuyruktan bir eleman çıkarmak için:

```
int dequeue(Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Sira bos! Program sonlandiriliyor!\n");
        exit(-1);
    }
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->capacity;
    queue->numberOfElements = queue->numberOfElements - 1;
    return item;
}
```

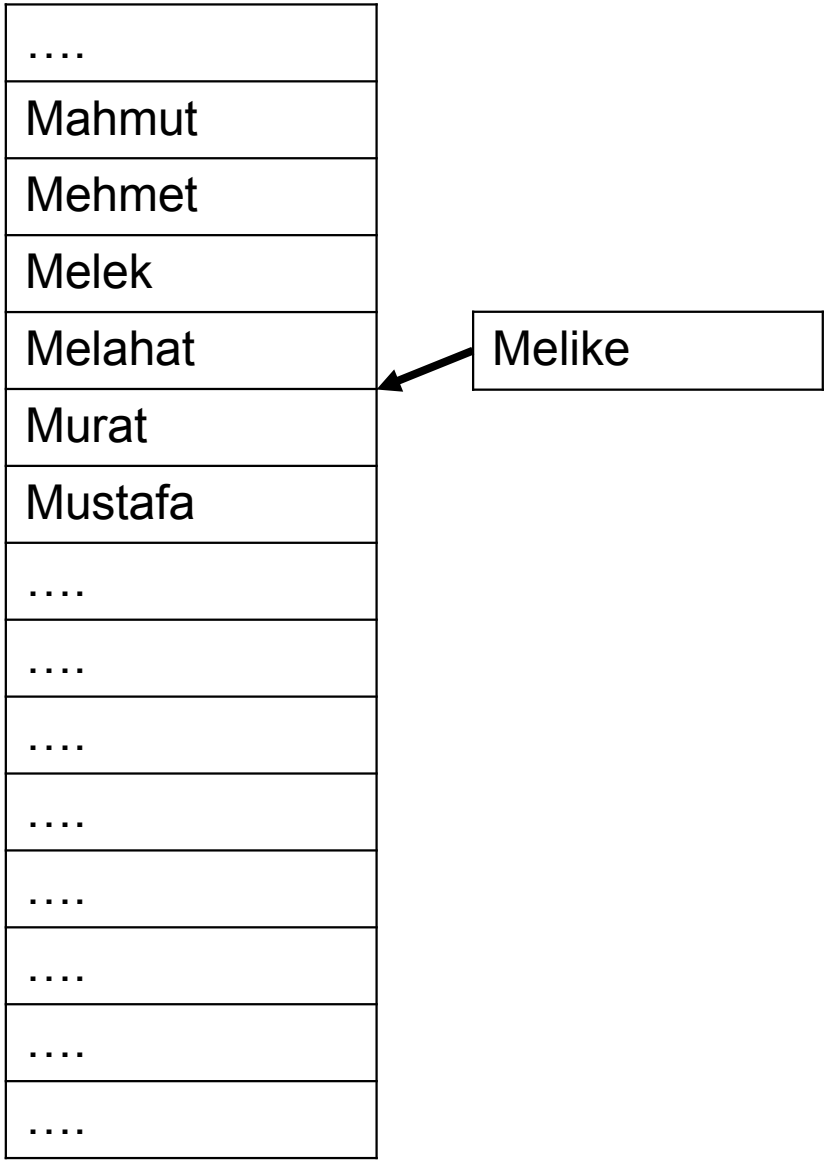
Kuyruk

- Kuyruk çalışma mantığını günlük hayatta birçok ortamda gözlemleyebiliriz.
  - Örneğin, banka gişesinde sıraya giren insanlar
- Kuyruk yapısı bilgisayar sistemlerinde birçok yerde kullanılır.
  - Bilgisayar ağları, işletim sistemleri, simülasyon sistemleri, çoklu ortam sistemleri kuyruk yapısının sıklıkla kullanıldığı ortamlar örnektir.
  - Örneğin, yazıcıya gönderdiğiniz işler kuyruk mantığı ile sıra ile yapılır.

Bağlı liste

- Birçok programda kullanılabilecek bir veri topluluğu olarak liste oluşturmak istediğimizi farzedelim.
- Bu işlemi basitçe bir dizi kullanarak halledebiliriz.
- Ancak listemizi dizi olarak oluşturduğumuzda aşağıda belirtilen durumlar gözlemlenir.
  - Dizi veri yapısında elemanlar aynı türden olur bellekte art arda sıralanırlar.
  - Dizi elemanları arasında başka elemanlar bulunamaz.
  - Dizinin sonu hariç başka bir noktasına ekleme veya bir noktadan çıkarma yapmak istediğinizde dizi elemanlarının yer değiştirmesi gerekir.
  - Dizinin büyüklüğü en başta dizi oluşturulurken tanımlanır ve daha sonra değiştirilemez.
    - Diziyi büyütmek isterseniz daha büyük bir dizi oluşturup bütün elemanları kopyalamanız gerekir.
    - Yığın ve kuyruk yapılarını dizi kullanarak oluşturduğumuzda capasite tanımlamamız gerekti.
  - Dizi büyüklüğü sonradan sorun olmaması için çok büyük bir dizi kullanırsanız bu sefer de yer israfı yapmış olursunuz.

Bağlı liste



Bağlı liste

- Bağlı listeler dizi kullanımı kaynaklı önceki slaytlarda belirttiğimiz sorunları ortadan kaldırır.
  - Bağlı listelerin elemanları bellekte birbirini takip eder şekilde bulunmak zorunda değildir.
  - Bağlı listenin her elemanı kendinden sonraki elemanın nerede olduğunun bilgisini taşır.
    - Bunu sahip olduğu işaretçi sayesinde yapar.
  - Bağlı listenin her elemanı bir miktar bilgi ve sonraki elemana işaretçi taşır.
  - Her eleman için ayrı hafıza alanı ayrılır.
  - Bu sayede bağlı liste kullanıldığında listenin boyu dinamik olarak artabilir veya azalabilir.
  - Ayrıca yeni bir eleman listenin herhangi bir noktasına kolaylıkla eklenebilir.
  - Bu sebeplerle bağlı listeler çoklukla kullanılan doğrusal veri yapısı tipidir.
  - Ayrıca, önceki derste belirttiğimiz gibi yığın veya kuyruk yapısı gibi veri yapıları bağlı listeler kullanılarak oluşturulabilir.

Bağlı liste

- Bağlı listelerin her bir elemanına düğüm ismi verilir.
- Düğümlerin sahip olduğu bağlantı sayı ve tipine göre farklı bağlı liste tipleri oluşturulabilir.
  - Tek yönlü bağlı listeler
  - Çift yönlü bağlı listeler
  - Dairesel bağlı listeler
- Bağlı listelerde aşağıda belirtilen işlemler yapılabilir
  - Listeye eleman ekleme (başa, sonra, belli noktaya)
  - Listeden eleman silme (Baştan, sondan, belli noktadan, tümünü, belli bilgiye sahip olanı)
  - Arama
  - Listeleme

Bağlı liste

- Tek yönlü bağlı listeler
  - Bu tür bağlı listelerde düğümler arası sadece bir bağ bulunur.
  - Tek yönlü bağlı liste için öncelikle bir düğüm yapısı tanımlamalıyız.

```
typedef struct node
{
    int key;
    struct node* next;
}node;
```

  - key, düğümde taşınan veridir.
  - next ise düğümün bir sonraki düğüme bağıdır.

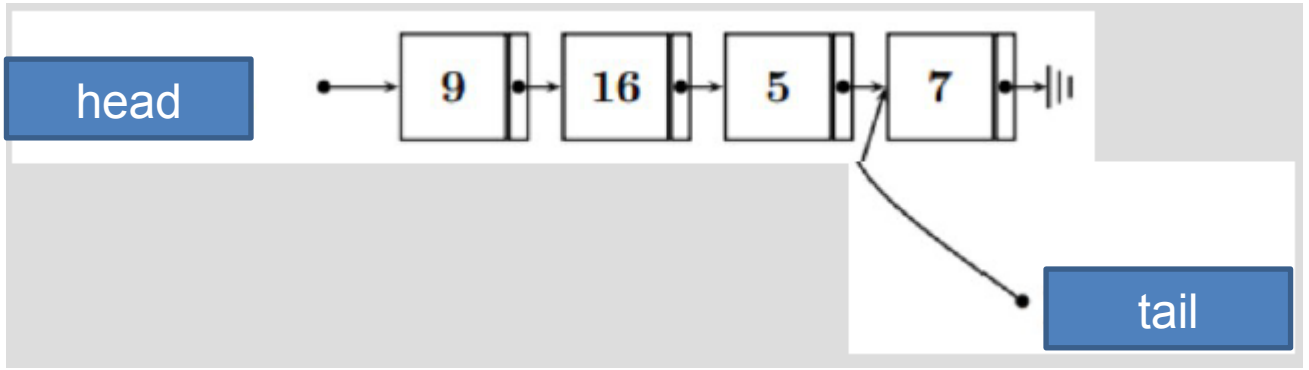
Bağlı liste

- Tek yönlü bağlı listeler
  - Daha sonra bir düğümü dinamik olarak oluşturan bir fonksiyon yazmalıyız.

```
node* CreateNode(int value)
{
    node *newNode = malloc(sizeof(node));
    newNode->key = value;
    newNode->next = NULL;
    return newNode;
}
```

Bağlı liste

- Tek yönlü bağlı listeler
  - Bağlı listemize düğüm eklemek veya bağlı listemizden düğüm silebilmek için listedeki ilk ve son düğümü hatırlayacağız.
    - İlk düğüm head, son düğüm ise tail olarak adlandırılacak.



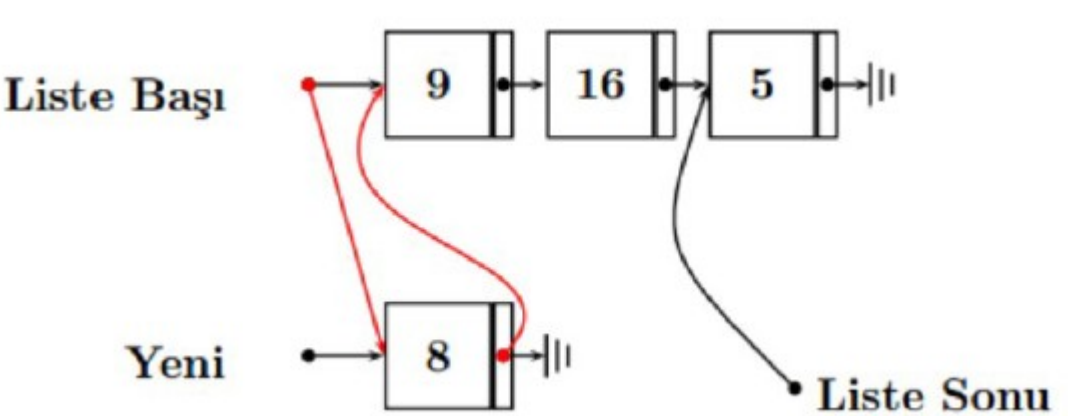
    - Listemiz head ve tail düğümlerini kullanarak baştan veya sondan ekleme/silme yapmamıza olacak verecek.

```
node *head = NULL;
node *tail = NULL;
```

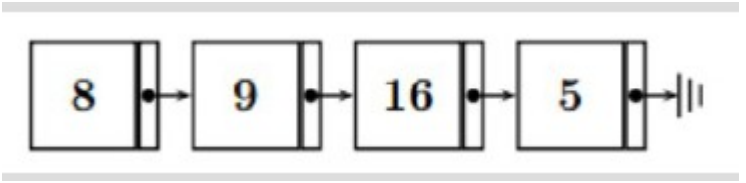
Bağlı liste

- Tek yönlü bağlı listeler
  - Listemize baştan ekleme yapmak için:

```
void ListInsertHead(node *x)
{
     if(head == NULL)
     {
          head = x;
          tail = x;
     }
     else
     {
          x->next = head;
          head = x;
     }
}
```



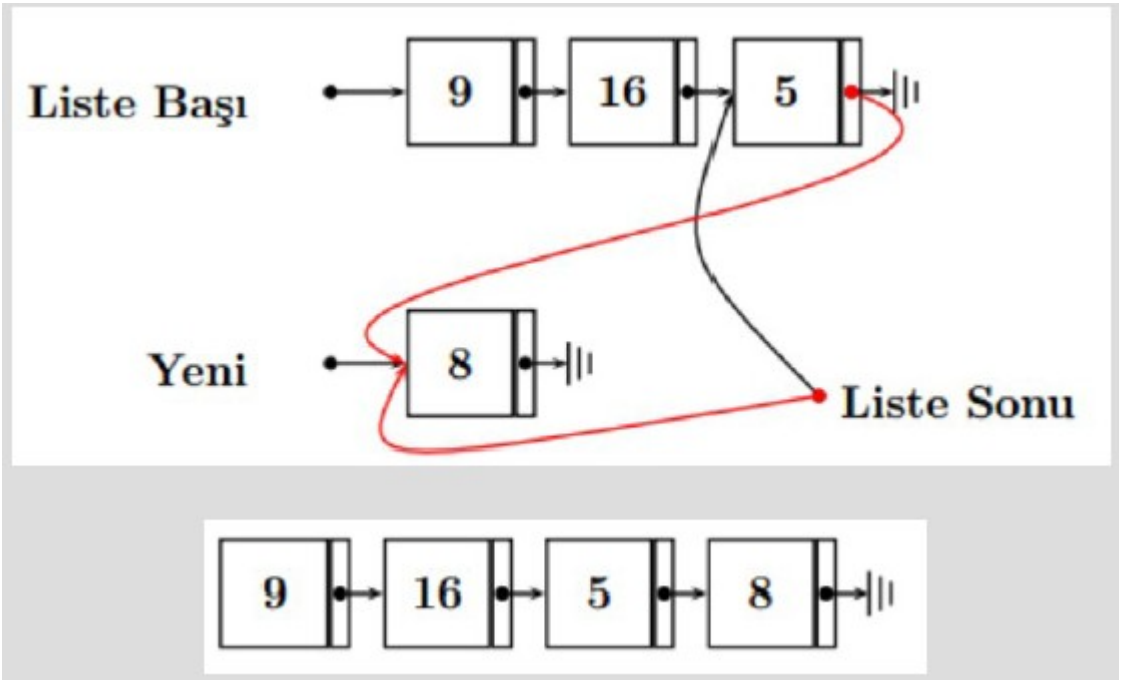
  - O(1) süre alır.



Bağlı liste

- Tek yönlü bağlı listeler
  - Listemize sondan ekleme yapmak için:

```
void ListInsertTail(node *x)
{
     if(head == NULL)
     {
          head = x;
          tail = x;
     }
     else
     {
          tail->next = x;
          tail = tail->next;
     }
}
```

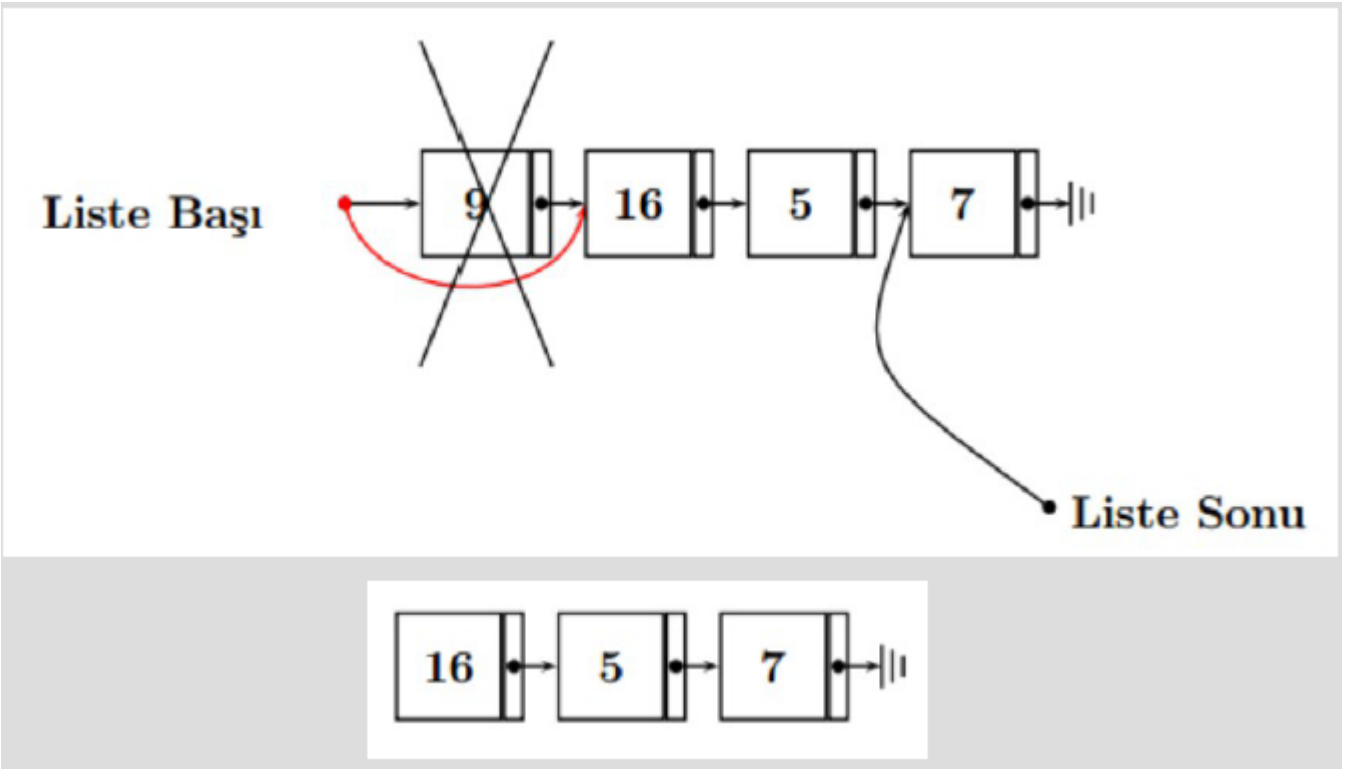


  - O(1) süre alır.

Bağlı liste

- Tek yönlü bağlı listeler
  - Listemizde baştan silme yapmak için:

```
void ListDeleteHead()
{
     if(head == NULL)
          return;
     node* x = head;
     head = head->next;
     if(head == NULL)
          tail = NULL;
     free(x);
}
```



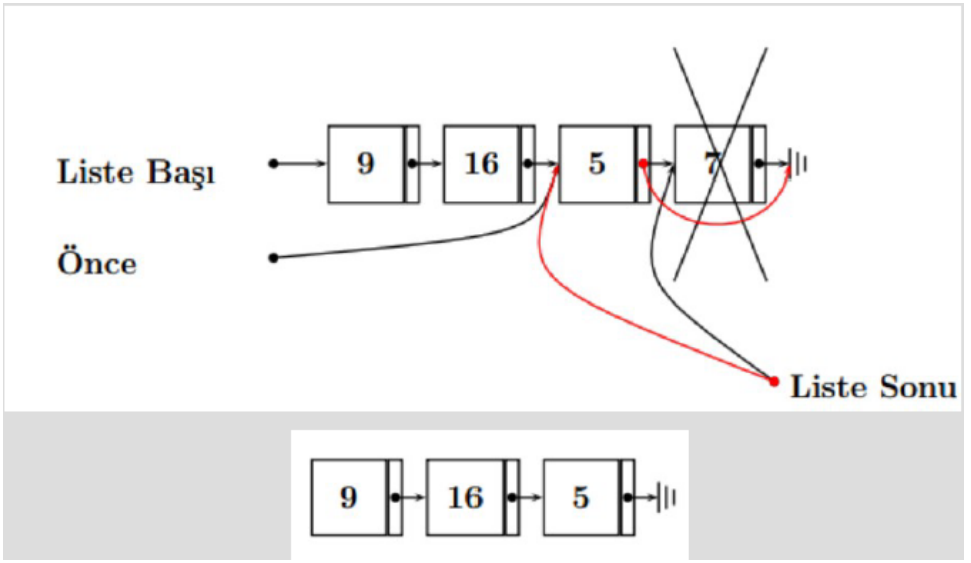
  - O(1) süre alır.

Bağlı liste

- Tek yönlü bağlı listeler
  - Listemizde sondan silme yapmak için:

```
void ListDeleteTail()
{
    if(head == NULL)
        return;

    node* x = head;
    if (head->next == NULL)
    {
        head = NULL;
        tail = NULL;
        free(x);
    }
    else
    {
        while(x->next->next != NULL)
        {
            x = x->next;
        }
        tail = x;
        free(x->next);
        x->next = NULL;
    }
}
```



  - O(n) süre alır!

Bağlı liste

- Tek yönlü bağlı listeler
  - Listemizdeki elemanlar yazdırma:

```
void DisplayList()
{
    printf("DisplayList starts.....................\n");
    node *x = head;
    while(x != NULL)
    {
        printf("%d ", x->key);
        x = x->next;
    }
    printf("\nDisplayList ends.....................\n\n\n");
}
```

  - O(n) süre alır!

Bağlı liste

- Tek yönlü bağlı listeler
  - Listemizde arama yapma:

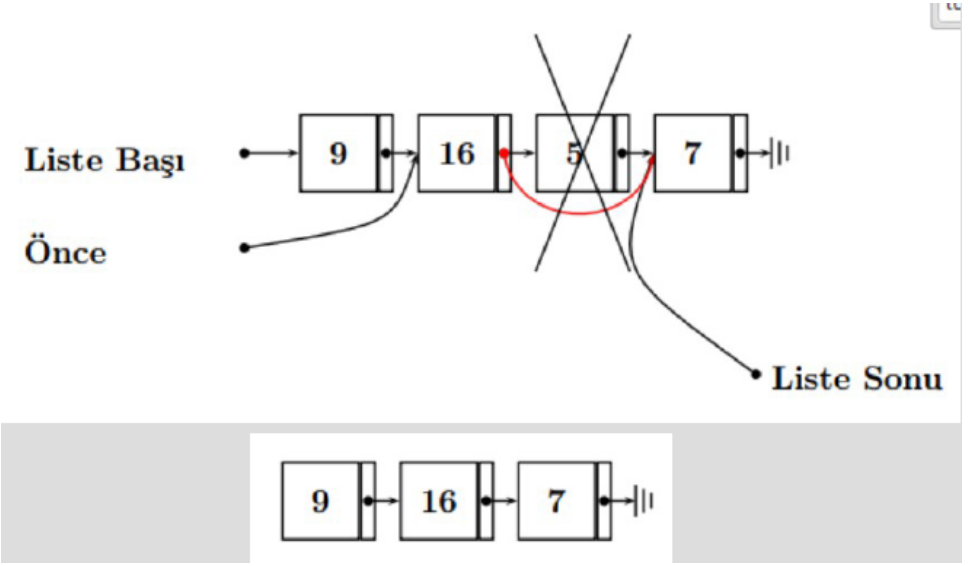
```
node* ListSearch(int k)
{
    node *x;
    x = head;
    while(x != NULL && x->key != k)
    {
        x = x->next;
    }
    return x;
}
```

  - O(n) süre alır!

Bağlı liste

- Tek yönlü bağlı listeler
  - Listemizden herhangi bir noktadan belli bir elemanı silme:

```
void ListDeleteNode(node* x)
{
    if(x == NULL)
        return;
    if(head == x)
    {
        ListDeleteHead();
    }
    else
    {
        node* temp = head;
        while(temp->next != x)
        {
            temp = temp->next;
        }
        temp->next = temp->next->next;
        free(x);
    }
}
```



  - O(n) süre alır!

Bağlı liste

- Sorular:
  - Önceki slaytlarda gördüğümüz tek yönlü bağlı liste ile yığın yapısı oluşturabilir miyiz?
    - Evet! Baştan ekler (O(1)), baştan sileriz (O(1)). Böylece hem yığına ekleme hem de yığından silme işlemleri O(1) sürede yapılır.
  - Önceki slaytlarda gördüğümüz tek yönlü bağlı liste ile kuyruk yapısı oluşturabilir miyiz?
    - Evet! Sondan ekler (O(1)), baştan sileriz (O(1)). Böylece hem kuyruğa ekleme hem de kuyruktan silme işlemleri O(1) sürede yapılır.