

Veri Yapısı ve Veri Modeli

- Veri yapısı (Data Structure) verinin veya bilginin bellekte tutulma şeklini veya düzenini gösterir.
- Tüm programlama dillerinin, genel olarak, tamsayı, kesirli sayı, karakter ve sözcük saklanması için temel veri yapıları vardır. Bir program değişkeni bile basit bir veri yapısı olarak kabul edilebilir.
- Veri modeli (Data Model), verilerin birbirleriyle ilişkisel veya sırasal durumunu gösterir; problemin çözümü için kavramsal bir yaklaşım yöntemidir denilebilir.
- Bilgisayar ortamında uygulanacak tüm matematik ve mühendislik problemleri bir veri modeline yaklaştırılarak veya yeni veri modelleri tanımlaması yapılarak çözülebilmektedir.

1- Veri Yapılarına Genel Bakış (Doğrusal Veri Yapıları)

Veri yapısı, bir bilgisayarda verileri etkin bir şekilde kullanabilmesi için düzenlemenin özel bir yoludur. Fikir, farklı görevlerin mekan ve zaman karmaşıklıklarını azaltmaktır. Aşağıda bazı popüler doğrusal veri yapılarına genel bir bakış yer almaktadır.

1. Dizi (Array)

2. Bağlantılı Liste (Linked List)

3. Yığın (Stack)

4. Sıra (Queue)

Dizi (Array)

Dizi, bitişik konumlarda homojen elemanları saklamak için kullanılan bir veri yapısıdır. Veri saklamadan önce bir dizinin boyutu sağlanmalıdır.

Dizinin boyutu n olsun.

Accessing Time: $O(1)$ [Bu mümkün, çünkü elemanlar
bitişik konumlarda saklanır]

Ardışık Arama için Arama Süresi: $O(n)$:

İkili Arama için $O(\log n)$ [Dizi sıralanırsa]

Ekleme Zamanı: $O(n)$ [En kötü durum, yerleştirme sırasında meydana gelir.

Bir dizinin başlangıcında olur ve
tüm öğelerin kaydırılmasını gerektirir]

Silme Süresi: $O(n)$ [En kötü durum silme sırasında ortaya çıkar.

Bir dizinin başlangıcında olur ve
tüm öğelerin kaydırılmasını gerektirir]

Bağlı liste (Linked List)

Bağlantılı bir liste, her bir elemanın ayrı bir nesne olduğu doğrusal bir veri yapısıdır (diziler gibi). Bir listenin her bir elemanı (yani düğüm) iki maddeden oluşur - veri ve bir sonraki düğüme referans.

Bağlantılı Liste Türleri:

1. **Tek Bağlantılı Liste:** Bu bağlantılı listede, her düğüm listedeki bir sonraki düğümün adresini veya referansını saklar ve son düğüm bir sonraki adres veya NULL olarak referansı vardır. Örneğin 1-> 2-> 3-> 4-> NULL
2. **Çift Bağlantılı Liste:** Bu Bağlantılı listede, her düğümle ilişkili iki referans vardır, referanslardan biri bir sonraki düğüme ve bir tanesi bir önceki düğüme. Bu veri yapısının avantajı, her iki yönde de geçebilmemiz ve silmek için önceki düğüme açık bir şekilde erişmemize gerek olmamasıdır. Örneğin. BOŞ <-1 <-> 2 <-> 3-> NULL
3. **Dairesel Bağlantılı Liste:** Dairesel bağlantılı liste, tüm düğümlerin bir daire oluşturacak şekilde bağlandığı bağlantılı bir listedir. Sonunda *NULL* yoktur. Dairesel bir bağlı liste, tek bir dairesel bağlantı listesi veya iki kat dairesel bağlantı listesi olabilir. Bu veri yapısının avantajı, herhangi bir düğümün başlangıç düğümü olarak yapılabilmesidir. Bu, bağlantılı listedeki dairesel sıranın uygulanmasında kullanışlıdır. Örneğin. 1-> 2-> 3-> 1 [Son düğümün bir sonraki göstericisi birinciye işaret ediyor]

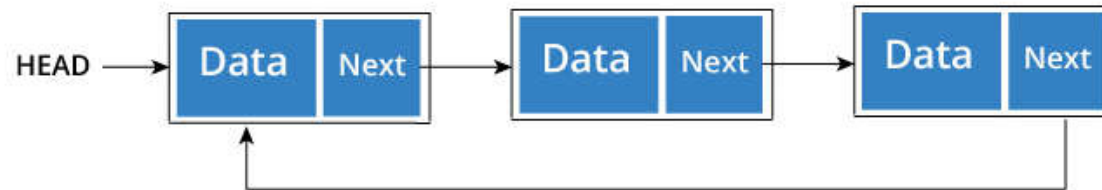
Bağlı liste (Linked List)



Tek Bağlantılı Liste(Singly Linked List)



Çift Bağlantılı Liste(Doubly Linked List)



Dairesel Bağlantılı Liste(Circular Linked List)

Bağlı liste (Linked List)

Bir elemanın erişim zamanı: $O(n)$
Bir elemanın arama süresi : $O(n)$
Bir Elementin Eklenmesi : $O(1)$ [Elemanı eklemek istediğimiz pozisyondaysak]
Bir Elemanın Silinmesi : $O(1)$ [Silinecek Düğüm öncesi Düğümünün adresini biliniyorsa]

➤ Tek Bağlı Listeler (*One Way Linked List*)

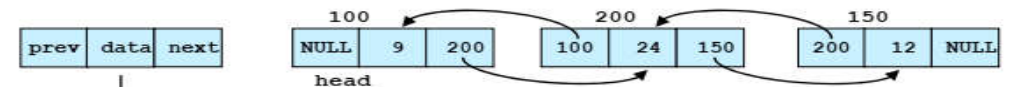
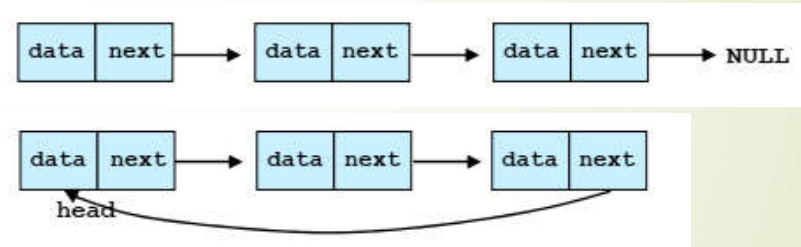
• Tek Bağlı Doğrusal Listeler (*One Way Linear List*)

• Tek Bağlı Dairesel Listeler (*One Way Circular List*)

➤ Çift Bağlı listeler (*Double Linked List*)

• Çift Bağlı Doğrusal Listeler (*Double Linked Linear List*)

• Çift Bağlı Dairesel Listeler (*Double Linked Circular List*)



Şekil 2.7 Çift bağlı liste yapısı ve mantıksal gösterimi.



Şekil 2.10 Çift bağlı dairesel listeler.

Yığın (Stack)

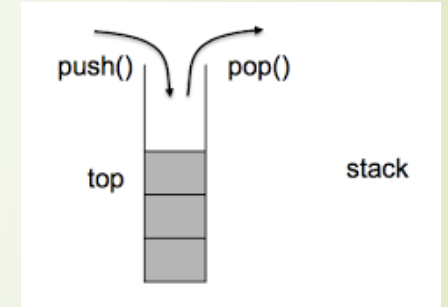
Bir yığın veya **FILO (ilk giren son çıkar)**, iki temel işlemle bir öge koleksiyonu görevi gören soyut bir veri türüdür: koleksiyona bir öge ekleyen *push* ve eklenen son ögeyi kaldıran *pop* ‘tan oluşan mekanizmadır. Yığın halinde hem push hem de pop işlemi, yığının en üstünde bulunan aynı uçta gerçekleşir. Stack, doğrusal artan bir veri yapısı olup; insert (push) ve delete (pop) işlemleri, listenin sadece “top” adı verilen bir ucunda yani stack’in en üstünden gerçekleştirilir.

Ekleme: $O(1)$

Silme: $O(1)$

Erişim Süresi: $O(n)$ [En Kötü Durum]

Ekleme ve Silme işlemine bir uçta izin verilir.



Örnek: Yığınlar, işlev çağrılarını sürdürmek için kullanılır (en son çağrılan işlev önce yürütmeyi bitirmelidir), özyinelemeyi her zaman yığınların yardımıyla kaldırabiliriz. Yığınlar ayrıca, bir sözcüğü ters çevirmemiz, dengeli parantez olup olmadığının kontrol edilmesi ve en son yazdığınız kelimenin geri alma işlemini kullandığınızda ilk çıkardığınız editörlerde de kullanılır. Benzer şekilde, web tarayıcılarında geri işlevsellik uygulamak için.

Yığın (Stack)

Infix notasyonu: Alışa geldiğimiz ifadeler infix şeklindedir. Operatörlerin işlenecek operandlar arasına yerleştirildiği gösterim biçimidir. Bu gösterimde operatör önceliklerinin değiştirilebilmesi için parantez kullanılması şarttır. Örneğin infix notasyonundaki $2+4*6$ ifadesi $2+24=26$ ile sonuçlanır. Aynı ifadede + operatörüne öncelik verilmesi istenirse parantezler kullanılır; $(2+4)*6$. Böylece ifade 36 ile sonuçlandırılır.


Prefix notasyonu: Prefix notasyonunda (PN, polish notation) operatörler, operandlarından önce yazılır. Örneğin $2+4*6$ ifadesi infix notasyonundadır ve prefix notasyonunda $+2*46$ şeklinde gösterilir. Benzer biçimde $(2+4)*6$ ifadesi $*+246$ şeklinde gösterilir. Görüldüğü gibi prefix notasyonunda işlem önceliklerinin sağlanması için parantezlere ihtiyaç duyulmamaktadır.

Postfix notasyonu: Postfix notasyonunda (RPN, reverse polish notation) ise önce operandlar ve ardından operatör yerleştirilir. Aynı örnek üzerinden devam edersek; infix notasyonundaki $2+4*6$ ifadesi prefix notasyonunda $2\ 4\ 6\ *\ +$ şeklinde, benzer biçimde $(2+4)*6$ ifadesi de $2\ 4\ +\ 6\ *$ şeklinde gösterilir. Yine prefix'te olduğu gibi bu gösterimde de parantezlere ihtiyaç duyulmamaktadır.

Yığın (Stack)

Infix, Postfix, Prefix:

Infix $((A / (B ^ C)) - ((D * E) - (A * C)))$

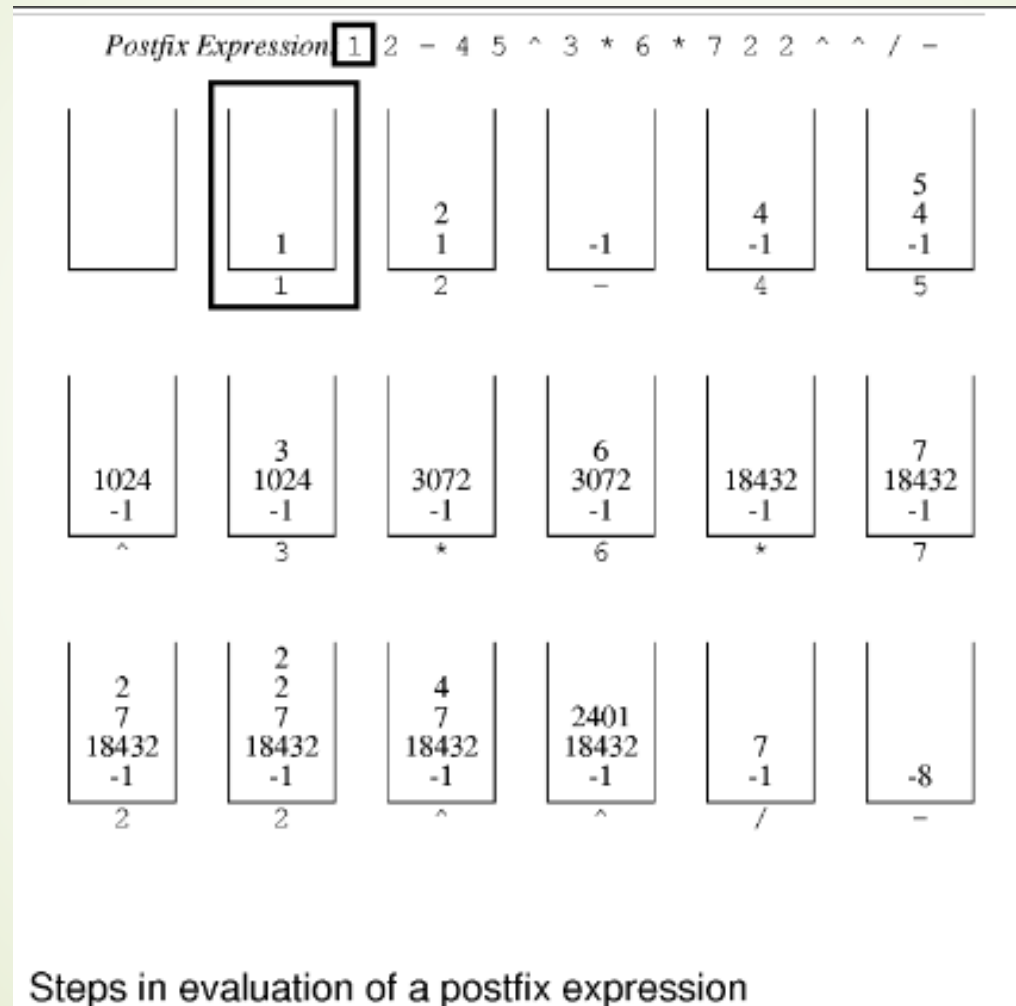


Postfix $A B C ^ / D E * A C * - -$



Prefix $- / A ^ B C - * D E * A C$

Yığın (Stack)



Kuyruk (Queue)

Sıra, basit veri yapılarından birisidir. Buna göre bir sıraya ilk giren ilk çıkar (FIFO , first in first out fifo). Bazı kaynaklarda kuyruk kelimesi de kullanılır. Basitçe bir gişe önündeki bilet kuyruğu veya bilet sırası olarak düşünülebilir. Bu yapının olmazsa olmaz iki adet fonksiyonu bulunur:

Enqueue and Dequeue

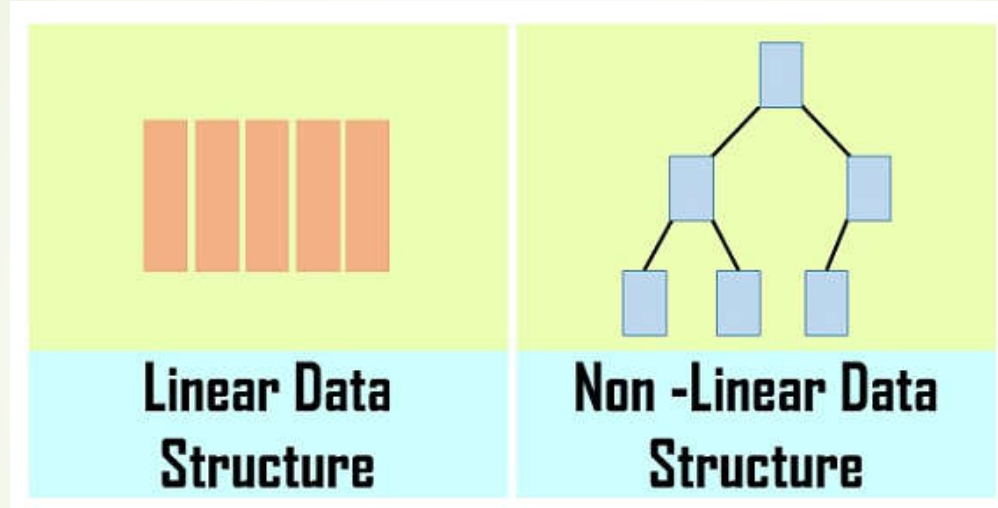
- Queue operations: **Enqueue** and **Dequeue**
- Like lines in a store, a queue has a **front** and a **rear**.
- **Enqueue** – insert an element at the **rear** of the queue
- **Dequeue** – remove an element from the **front** of the queue



Komutlar	Sıranın durumu
enqueue(10)	10
enqueue(20)	10 -> 20
enqueue(30)	10 -> 20 -> 30
dequeue()	20 -> 30
dequeue()	30
dequeue()	Boş

2- Veri Yapılarına Genel Bakış

(Doğrusal Olmayan Veri Yapıları-Binary Tree - Binary Search Tree)



Veri yapısı, verilerin tek unsurları arasında var olan mantıksal ilişkinin yorumlanması olarak tanımlanabilir. Doğrusal ve doğrusal olmayan veri yapısı, ilkel olmayan veri yapısı altında gelen veri yapısının alt sınıflamasıdır. Aralarındaki en önemli fark, doğrusal veri yapısının verileri bir sıraya göre düzenlemesi ve bir tür düzeni izlemesidir. Oysa, doğrusal olmayan veri yapısı verileri sıralı bir şekilde düzenlememektedir. Doğrusal veri yapısı, tek seviyeli bir veri yapısıdır, doğrusal olmayan veri yapıları ise çok seviyeli veri yapısıdır. Veri yapısı önceden verinin nasıl organize edildiğini, erişildiğini, ilişkilendirildiğini ve işlendiğini açıklar.

2- Veri Yapılarına Genel Bakış

(Doğrusal Olmayan Veri Yapıları-Binary Tree - Binary Search Tree)

Karşılaştırma Tablosu

Karşılaştırma için temel	Doğrusal veri yapısı	Doğrusal olmayan veri yapısı
Temel	Veri maddeleri, elemanların bitişik olarak eklendiği bir düzende düzenlenmiştir.	Verileri sıralanmış bir düzende düzenler ve veri elemanları arasında bir ilişki vardır.
Verilerin dolaşılması	Veri öğelerine bir defada erişilebilir (tek çalışma).	Bir seferde veri elemanlarının çaprazlanması mümkün değildir.
Uygulama kolaylığı	Daha basit	karmaşık
İlgili seviyeler	Tek seviye	Çoklu seviye
Örnekler	Dizi, sıra, yığın, bağlantılı liste vb.	Ağaç ve grafik.
Hafıza kullanımı	Etkisiz	etkili

2- Veri Yapılarına Genel Bakış

(Doğrusal Olmayan Veri Yapıları-Binary Tree - Binary Search Tree)

Binary Search Tree

Recent Articles on Binary Search Tree !

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

