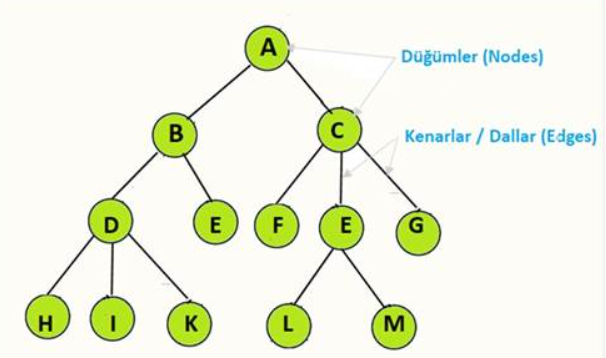
**Tree Data Srtuctures**

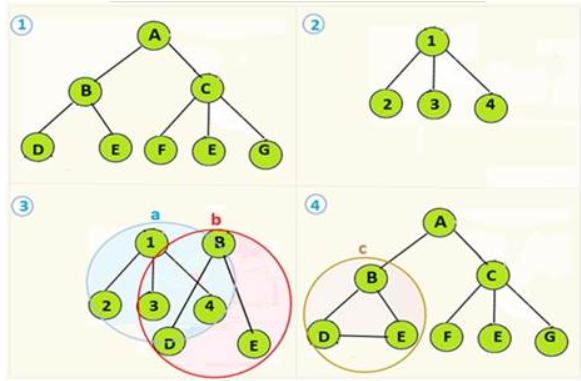
-Doğrusal olmayan hiyerarşik bir veri yapısıdır.

-Düğümler, ağacın temel elemanıdır.

-Ağaç ve düğümler arasındaki ilişki edges kullanarak oluşturulur.



-Ağaçlar döngü içermez. Mutlaka düğüm ve kenar arasında bağlantı olmalı

 3 ve 4 ağaç değil

-Ağaçlarda kök düğüm (root) bulunur.

-Düğümler arasında ebeyvn-çocuk ilişkisi vardır. (B ebeveyn D ve E çocuk)

-Çocuğu olmayan düğümler yapraktır (H,I,K,L,M)

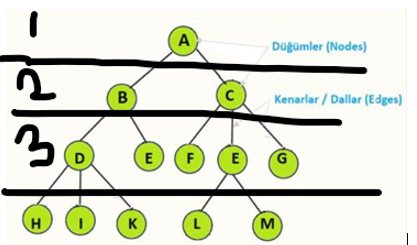
-Bir düğüme bağlı tüm alt düğümlere o düğümün descendant’ı denir

**Ağacın Derinliği:** Kök düğümün en uçtaki yaprak düğüme olan uzaklığı

**Düğümün Derinliği:** Bir düğümün kök düğüme olan uzaklığı

**Düğümün Yüksekliği:** düğümün ilişkili yaprak düğüme olan uzaklığı. Tek düğüm varsa yükseklik 0, hiç düğüm yoksa yükseklik -1

**Düzey:**



**Derece:** Düğmün çocuk sayısı

**Empyt Tree:** Hiç düğüm yoksa boş ağaçtır

**Inorder, preorder, postorder**

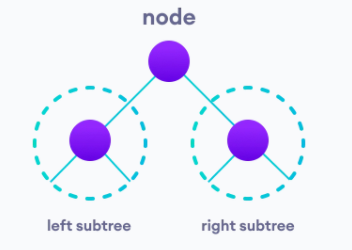
Bir ağacı okumak için tüm düğümlerden geçmeliyiz.

Stack, queue, linked list gibi veri yapılarının okunması için tek seçenek vardır. Ama ağaçlarda birden fazla seçenek vardır

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Left ve right bir düğümün sağ ve sol çocuğu olabilir, bu düğümler de kendi içinde bir ağaç barındırıyor olabilir. Ona göre düşünmeliyiz.



ok içeren bir resim

Açıklama otomatik olarak oluşturuldu

tablo içeren bir resim

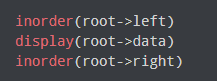
Açıklama otomatik olarak oluşturuldu

**Inorder Treaversal**

1-sol subtree

2-root node

3-sağ subtree



metin, ölçü aleti, gece göğü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Preorder Traversal**

1-Root node

2-sol subtree

3-sag subtree

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Postorder Subtree**

1-sol subtree

2-sağ subtree

3-root node

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Nerelerde Kullanılır?**

-Dosya/dizin yapılarında

-Veri tabanlarında (B-Tree ve T-Tree)

-Yapay zeka uygulamalarında gibi birçok alanda kullanılır

-Derleyiciler (sözdizim ağaçları)

-sıralama, arama, veri sıkıştırma, hardisklerin okunması

**Neden Ağaç Yapısını Kullanırız? Ağaç İşlemleri**

Diziler, bağlantılı listeler, yığın ve kuyruk ver yapıları doğrusaldır. Herhangi bir işlmei gerçekleştirmek için complexity time, veri boyutu arttıkça artar. Doğrusal veri yapılarında bu artış daha fazla olur. Ağaçlar doğrusal olmadığı için verilere daha hızlı ve kolay erişimemizi sağlar.

**Ağaç Türleri**

Aşağıda en çok kullanılan ağaçlar vardır. Her ağaç üzerinde arama, ekleme, silme algoritmaları farklıdır.

**Binary Search Tree**

İkili Arama ağacında düğümler 0 (sıfır), 1 (bir), 2 (iki) çocuğa sahip olabilir, daha fazla çocuğa sahip olamaz. Ayrıca bu ağaç veri yapısında alt düğümlerin (çocukların) bağlantıları belirli bir sırada yapılır. Örneğin önce küçük veya alfabetik olarak küçük olanlar sola, eşit ve büyük olanlar sağa bağlanır.

**Coding Tree**

Kodlama ağaç veri yapısı genel olarak bir kümedeki karakterlere, örneğin alfabedeki karakterlere kod atanması için kurulan bir ağaç şeklidir. Kodlama ağaçlarının en bileneni *Huffman Kodlaması ‘*dır. Ancak başka benzer birçok kodlama ağacı da vardır. Kodlama ağaçlarında kökten başlanıp yapraklara kadar olan yol üzerindeki bağlantı değerleri kodu vermektedir.

**Dictionary Tree**

Sözlük ağacı, adından da anlaşılacağı gibi bir sözlükte bulunan sözcüklerin tutulması için kurulan bir ağaç türüdür. Bu ağacın kurulmasındaki amaç veriler üzerinde arama işlemlerinin hızlı yapılması ve bilgisayar belleğinin verimli kullanılmasıdır. Sözlük ağaçları sözlüğün oluşturulabilmesi için bir araya gelip sözlük ormanını oluşturur. Bu ormanda alfabedeki karakter sayısı kadar ağaç vardır.

**Heap Tree**

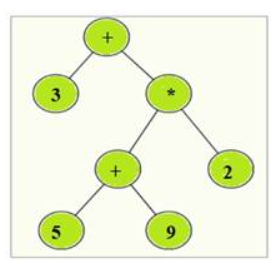
Kümeleme ağacının iki farklı türü vardır. Birincisi, çocuk düğümlerin her durumda aile düğümünden daha küçük değerlere sahip olduğu kümeleme ağaç türüdür. Bu ağaç türünde kök düğüm en büyük değere sahipken yaprak düğümler en küçük değere sahip olurlar. Bu ağaç türüne Max Heap denir. İkincisi ise çocuk düğümlerin her durumda aile düğümünden daha büyük değerlere sahip olduğu kümeleme ağaç türüdür. Bu ağaç türünde kök düğüm en küçük değere sahipken yaprak düğümler en büyük değere sahip olurlar. Bu ağaç türüne Min Heap denir.

küçük resim, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Expressin Tree (Bağlantı Ağacı)**

Bağıntı ağaçları bir ikili ağaç uygulamasıdır ve matematiksel bir bağıntının ağaç şeklinde tutulması için tanımlanmıştır. Bağıntı ağacının yapraklarında değişken veya sabit değerler tutulurken kök düğüm ve iç düğümlerde operatörler tutulur. Aşağıda *3 + ((5 + 9) \* 2)* ifadesinin bağıntı ağacı gösterilmiştir.



**Binary Trees**

İkili ağaçlar (Binary Trees) her düğümünün en fazla iki çocuk alabildiği ağaç veri yapısıdır. Buradan hareketle ikili ağaçlarda her düğümün 0 veya 1 veya 2 çocuğu olabileceğini söyleyebiliriz. İkili ağaçtaki her düğüm, veri öğesi ile birlikte bir sol ve sağ referansa sahiptir. Bu ağaçlar özellikle arama işlemlerinde daha iyi sonuç alabilmek için üretilmiştir. İkili ağaçların arama performansları dizilerden daha kötü, bağlı listelerden daha başarılıdır. Buna karşılık ikili ağaçlar ekleme (Insert) ve Silme (Delete) işlemlerinde dizilerden daha iyi sonuç vermektedirler. İkili ağaçlarda eleman eklenmesi ve silinmesi için bir sınır yoktur. Bu yönüyle ikili ağaçlar bağlı listelere benzer.

İkili ağaçlar veriler organize edilirken, hiyerarşik veriler üzerinde işlem yapılırken, arama performansına ihtiyaç duyulduğu zaman, yön bulma algoritmalarında, sıralı elemanları düzenleme gibi işlemlerde kullanılır.

