Veri Yapıları

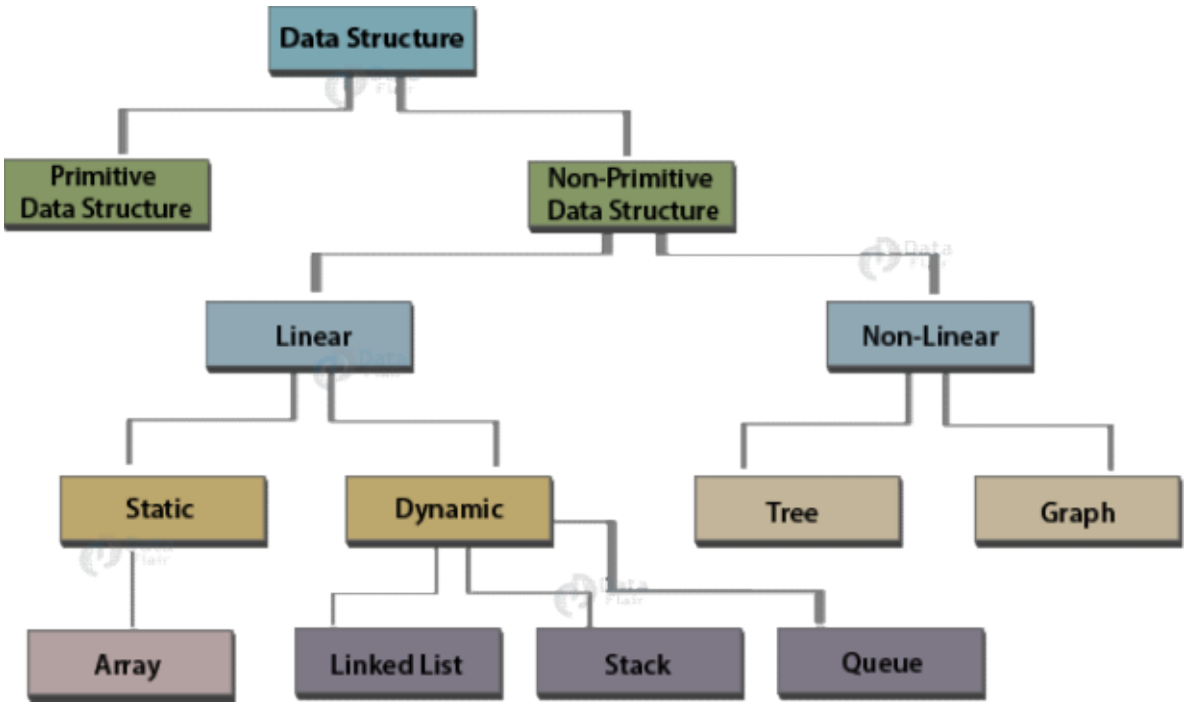
- **Veri Yapıları Nedir?**
  Bilgisayar bilimlerinde, veri yapısı, verimli erişim ve değişiklik yapmamızı sağlayan bir veri organizasyonu, yönetimi ve depolama biçimidir.

- **Veri Yapıları Neden Gereklidir?**
  Bilgisayar algoritmalarındaki karmaşıklık arttıkça, veri kullanım miktarı da dolaylı olarak artmaktadır, bunun sonucunda uygulamaların performansı etkilenebilir ve bazı endişelere yol açabilir. Veri yapılarını kullanarak performans kayıplarını ve karmaşıklığı engellemeye çalışılır.

Veri Yapıları İkiye Ayrılır:

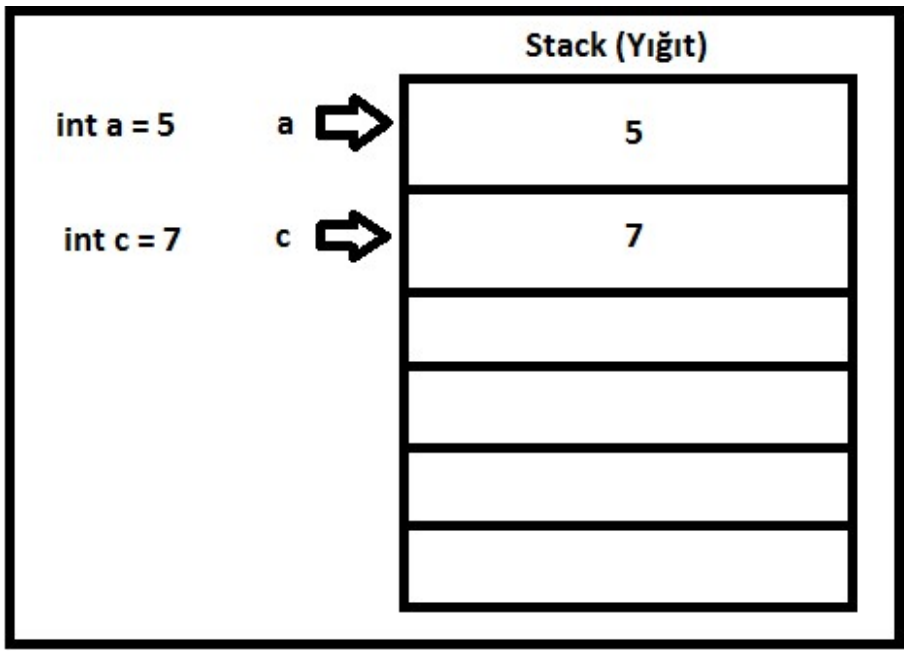
1.İlkel Veri Yapıları(Primitive)

2.İlkel Olmayan Veri Yapıları(Non-Primitive)



- **İlkel Veri Yapıları**

  İlkel veri tipleri, değerleri yığın üzerinde tutulan tiplerdir. Örneğin; long, int ve double gibi. İlkel tiplere değer girilmediğinde varsayılan değer atanır.



- **İlkel Olmayan Veri Yapıları**

İlkel olmayan veri yapıları iki türe ayrılır

1.Doğrusal Veri Yapıları
2.Doğrusal Olmayan Veri Yapıları

- **Doğrusal Veri Yapıları**

Doğrusal veri yapıları verilerin sıralı bir şekilde dizilmesi ve düzenlenmesidir. Bu amaçla kullanılan veri yapıları ; "Dizi(Array), Bağlantılı Liste(Linked List), Yığın(Stack) ve Kuyruk(Queue)". Bu veri yapısında bir veri diğer veriye doğrusal olarak bağlanır.

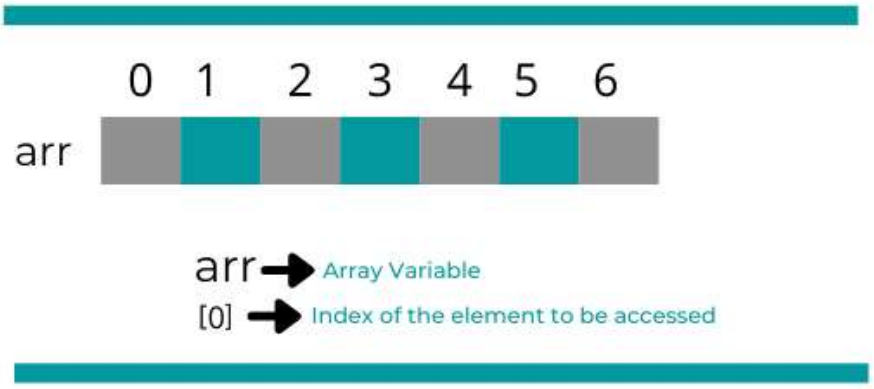
1. Dizi(Array)

Dizi, bitişik olarak bellekte tutulan bir veri yapısıdır. Dizilerde sadece aynı türe sahip

veriler saklanabilir. Dizilerin boyutu sabittir. Tanımlanırken dizi boyutu atanır ve daha sonra değiştirilemez. Kullanılmayan dizi elemanları bellekte "NULL" olarak yer kaplar.

Çoğu programlama dilinde dizinin ilk elemanı 0. indiste tutulur.

Elemanlar bitişik olarak tutulduğu içinde dizi içinde dolaşmak oldukça hızlı bir şekilde gerçekleşir.



2. Bağlantılı Liste(Linked List)

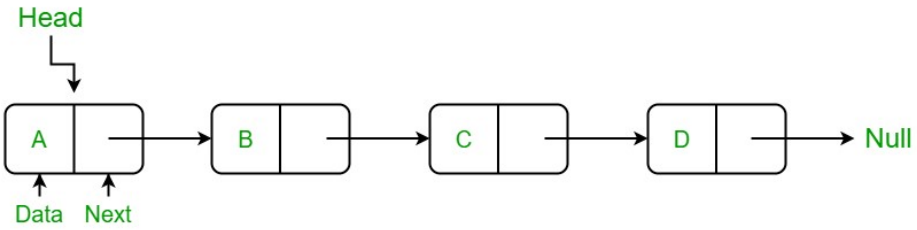
Bağlantılı liste düğüm (node) ismi verilen veri tutan yapılardan oluşur. Yeni bir veri ekleneceği zaman yeni bir düğüm oluşturulur ve listeye bağlanır. Eleman silineceği zaman bağ koparılır eleman silinir ve kopan bağ tekrar kurulur.

Düğümde herhangi bir elemana ulaşmak için listenin oluşum şekline göre listede gezilmesi gerekir.

- **Tek Yönlü Bağlantılı Liste(Singly Linked List)**

Düğüm (node) içerisinde iki farklı değişken olur. Bu değişkenlerden biri bilgi saklarken ikinci değişken bir işaretçidir (pointer) ve diğer düğümler ile bağlantı kurulmasını sağlar.

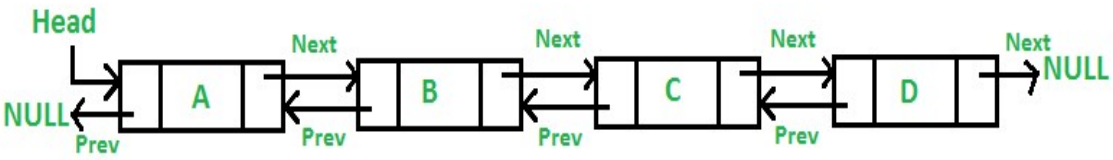
Listenin son elemanı (tail)'nın işaretçisi NULL'ü işaret eder.



- **Çift Yönlü Bağlantılı Liste(Doubly Linked List)**

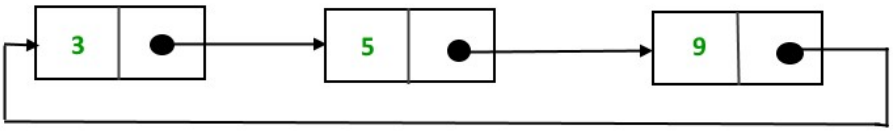
Düğüm içerisinde üç farklı değişken bulunur. Bunlardan biri bilgi saklarken diğer ikisi işaretçi görevi görür. İşaretçilerden biri kendinden önceki düğümü işaret ederken diğeri ise kendinden sonraki düğümü işaret eder.

İşaretçilerin son elemanları NULL'ü işaret eder. İki işaretçi olduğu için iki adet düğümğün işaretçisi NULL'ü işaret eder.



- **Dairesel Liste(Circular List)**

Tek yönlü veya çift yönlü listelerin son düğümler(tail), ilk düğümlerine(head) bağlanırsa dairesel listeler oluşur. Böylelikle son düğümden ilk düğüme ve ilk düğümden son düğüme kolaylıkla geçilebilir.

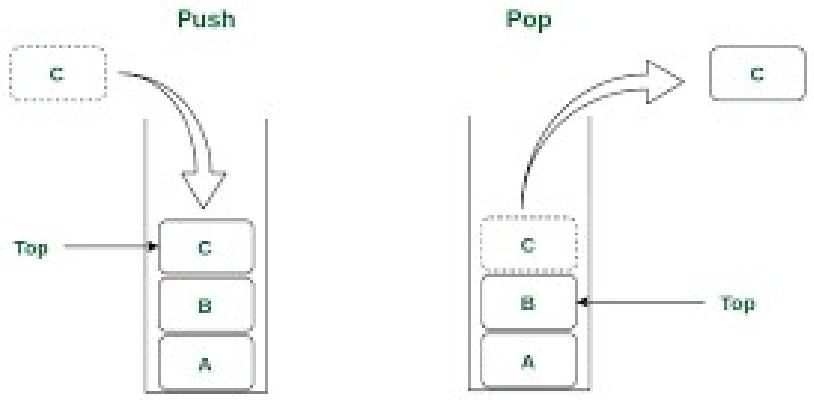


3.Yığın(Stack)

Yığın veri yapısında en son gelen eleman ilk erişilebilen elemandır. Verilere sadece tek taraftan erişim sağlanır.

Bu veri yapısında Last-In-First-Out (LIFO) prensibi vardır. Yani son giren eleman, ilk çıkar.

Yığın yapısı dizi veya bağlantılı liste kullanılarak oluşturulabilir.



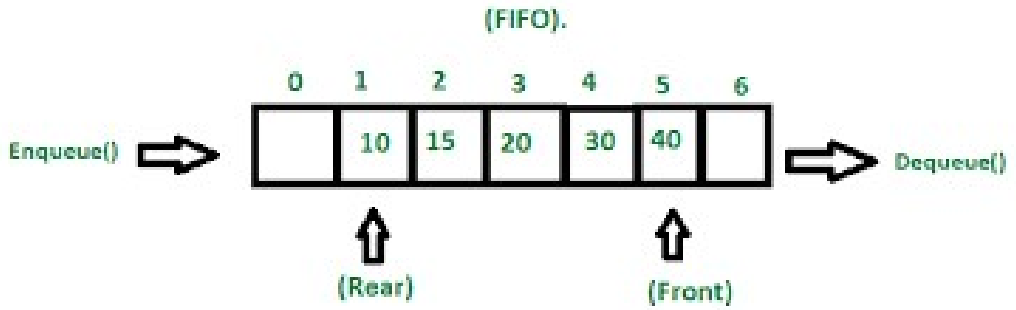
Stack Data Structure

4.Kuyruk(Queue)

Bu veri yapısında ilk gelen eleman ilk erişilen eleman manasına gelir. Verilere sadece tek taraftan erişim sağlanır.

Bu erişimde First-In-First-Out (FIFO) prensibi vardır. Yani ilk giren eleman, ilk çıkar.

Kuyruk yapısı dizi veya bağlantılı liste kullanılarak oluşturulabilir.



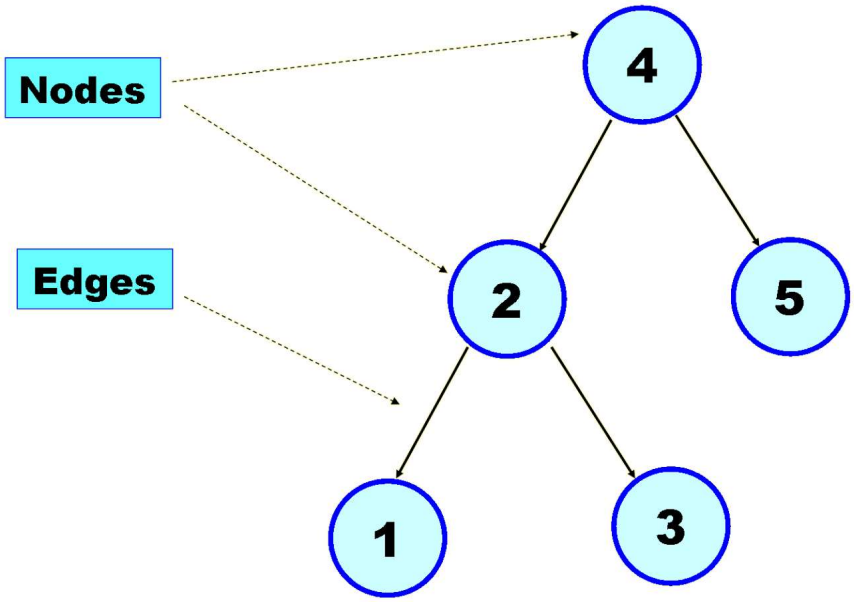
- **Doğrusal Olmayan Veri Yapıları**

Doğrusal olamayan veri yapıları verilerin sıralı olmayan bir şekilde dizilmesi ve düzenlenmesidir. Bu amaçla kullanılan veri yapıları; "Ağaç(Tree) ve Graf(Graph)". Bu veri yapısında elemanlar rastgele düzenlenmiştir.

1.Agaç(Tree)

Verilerin birbirine ağaç yapısına benzer bir şekilde bağlanmasına ağaç veri yapısı denir.

Bir ağacın her elemanına düğüm(node) denir. Düğümler birbirlerine dal ile bağlanır.



Ağaç yapısı hiyerarşi içerir. Bu yapının en tepesindeki düğüm kök(root), en ucundaki çocuğu olmayan düğüm yaprak(leaf), çocuğu olan düğümler ebeveyn(parent) çocuklar ise çocuk(child) olarak adlandırılır.

Bir düğüme kadar giderken geçilen düğümler o düğümün atalarıdır.

Bir düğüme kadar giderken geçilen düğümler listesine yol(path) denir.

Düğümün derinliği(depth), o düğümden kök düğüme kadar olan yolun uzaklığıdır.

Düğümün yüksekliği(height), o düğümden kendisiyle ilişkili en uzak yaprak düğüme kadar giden yolun uzunluğudur.

Düğümün düzey/seviyesi(level), kök ve ilgili düğüm arasında bulunan düğümlerin sayısına eşittir.

Bir düğüm'ün derecesi(degree), çocuk sayısına eşittir.

Ağaçlar döngü içermezler.

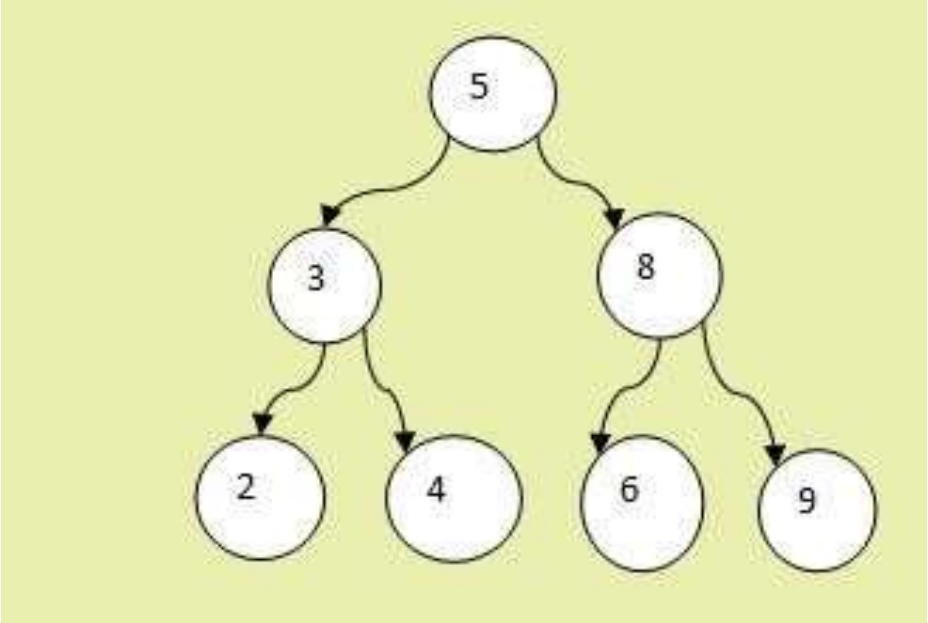
Ağaç Türleri

1.İkili Arama Ağacı(Binary Search Tree)
2.Adelson-Velsky Ağacı(AVL)
3.Huffman Ağacı

- ### İkili Arama Ağacı(Binary Search Tree)

İkili ağaçların (Binary Tree) özel bir hali olan ikili arama ağaçlarında, düğümlerde duran bilgilerin birbirine göre küçüklük büyüklük ilişkisi bulunmalıdır.

İkili arama ağacı, her düğümün solundaki koldan ulaşılabilecek bütün verilerin düğümün değerinden küçük, sağ kolundan ulaşılabilecek verilerin değerinin o düğümün değerinden büyük olmasını şart koşar.



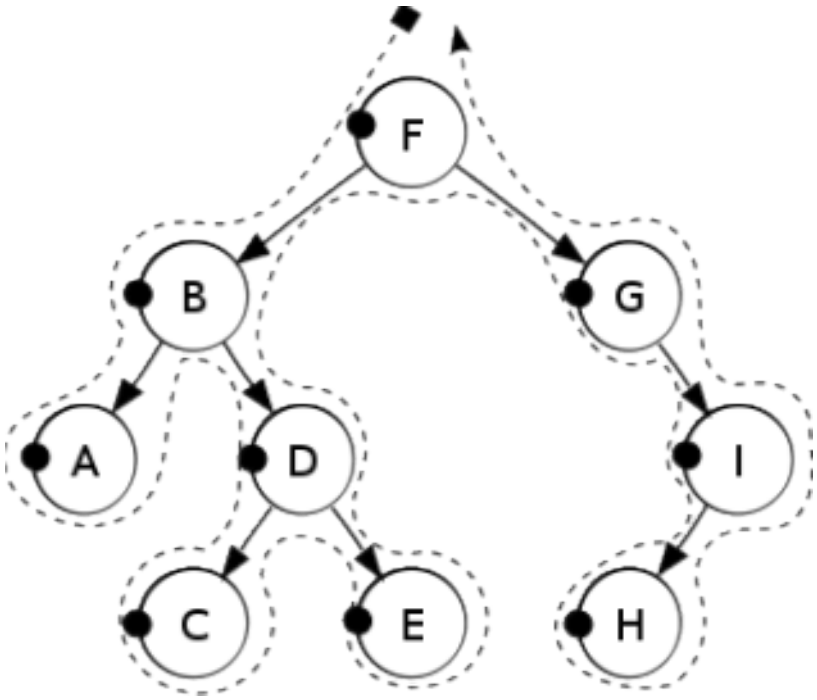
Ağaç Üzerinde Dolaşma Türleri

Ağaç üzerinde pek çok farklı dolaşma türü olsa da biz 3 tanesine değineceğiz.

1.Pre-Order Traversal
- Kökü ziyaret et
- Sol alt ağacı dolaş
- Sağ alt ağacı dolaş

2.In-order Traversal
- Sol alt ağacı dolaş
- Kökü ziyaret et
- Sağ alt ağacı dolaş

3.Post-order Traversal
- Sol alt ağacı dolaş
- Sağ alt ağacı dolaş
- Kökü ziyaret et



Pre-Order Traversal : F, B, A, D, C, E, G, I, H
In-order Traversal : A, B, C, D, E, F, G, H, I
Post-order Traversal : A, C, E, D, B, H, I, G, F

- Adelson-Velsky Ağacı(AVL)

AVL Ağaçları sürekli olarak dengeli olan ikili arama ağacıdır.

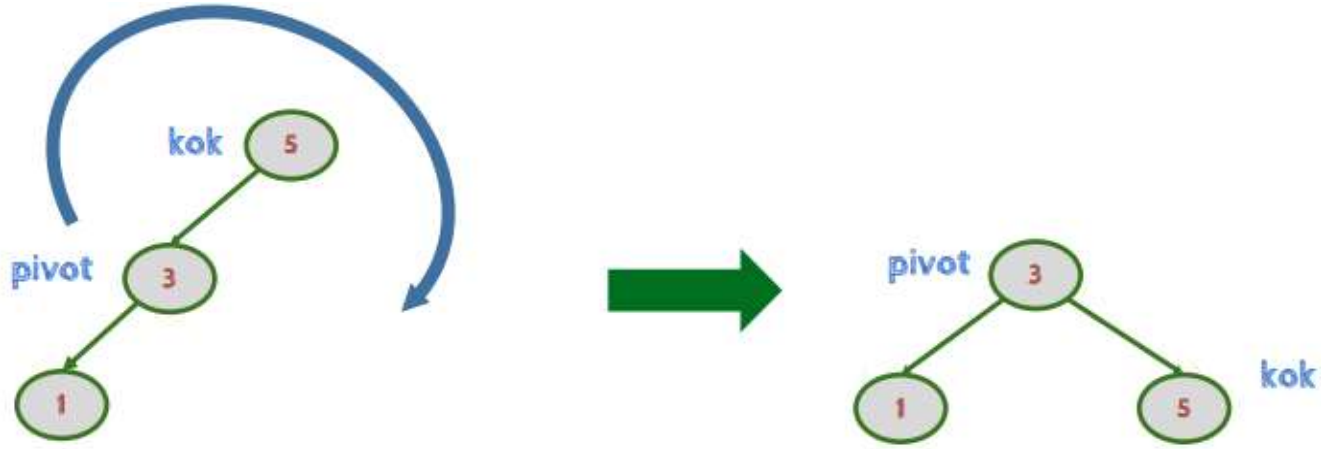
Bir düğümün kolları arasındaki derinlik farkı 2 ise bu durumda dengeleme işlemi yapılır. Şayet fark 2'den az ise bu durumda bir dengeleme işlemine gerek yoktur.

Ağaca eğer bir düğüm eklenecek veya silinecekse bu iki işlem sonrasında ağaç dengeli mi diyr bakılır. Eğer dengesiz ise gerekli döndürme işlemleri yapılır.

Dengesiz ağacı dengeleme işleminde dört durum vardır:

- Solun solu ise sağa döndürme
- Sağın sağı ise sola döndürme
- Solun sağı ise önce sola sonra sağa döndürme
- Sağın sol ise önce sağa sonra sola döndürme

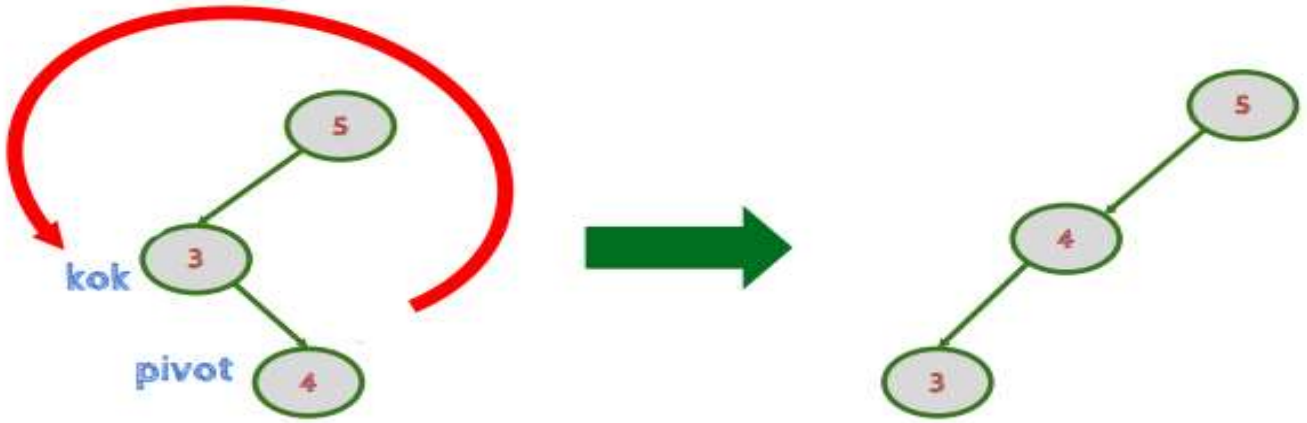
Dengeleme İşlemi : Solun Solu



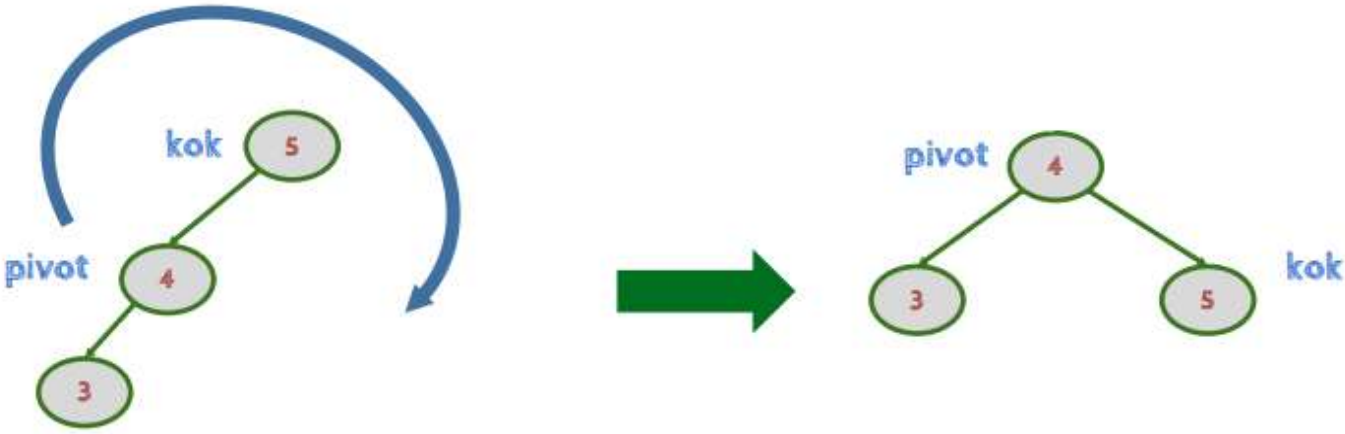
Dengeleme İşlemi : Sağın sağı

Dengeleme İşlemi : Solun sağı

1.Adım: Sola döndürme

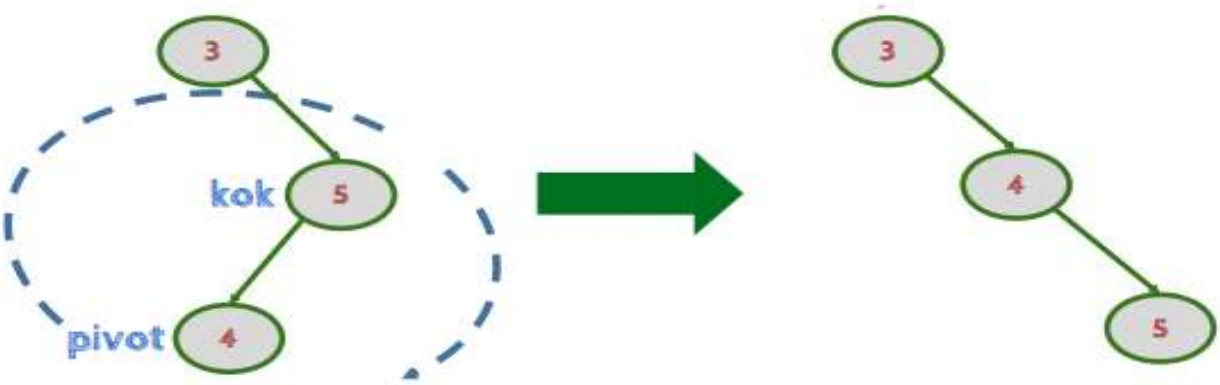


2.Adım: Sağa döndürme

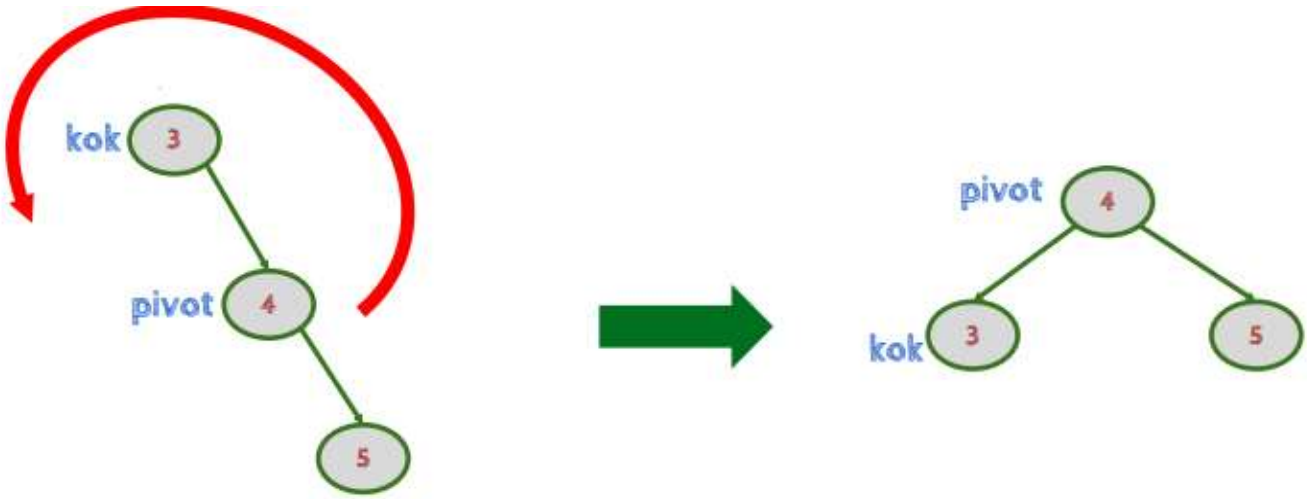


Dengeleme İşlemi: Sağın Solu

1.Adım: Sağa döndürme



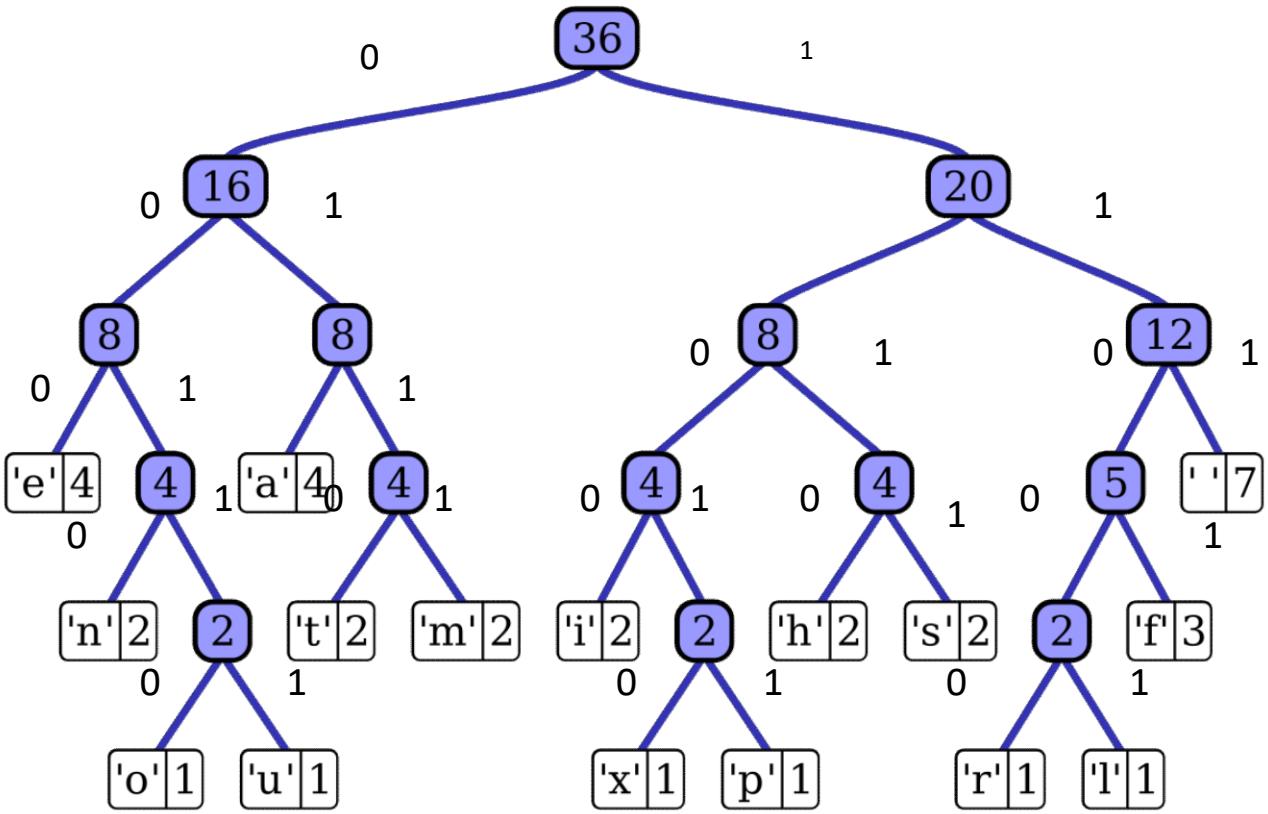
2.Adım: Sola döndürme



- **Huffman Tree**

Veriyi en düşük maliyetle ve kayıpsız sıkıştırmak için, karakterleri kullanım sıklıklarına göre sıralarız . En sık kullanılan karaktere en kısa kodu, en az kullanılan karaktere en uzun kodu verirsek, sabit kod uzunluğuna göre daha kısa bir çıktı üretebiliriz.

Bu kodlama sisteminde ağaç kullanmamızın sebebi ise peş peşe gelen kodlamalar eğer ağaç kullanmaz isek birbirlerine benzeyebilirler. Bu karışıklığı ortadan kaldırmak için Huffman Ağacı kullanılır.



Yukarıda görüldüğü gibi her harf ve kaç kez geçtikleri bilgileri kullanılarak bir huffman ağacı oluşturulmuştur. Oluşan ağaçta sola giden dallara 0 sağa giden dallara 1 yazılır ve harfler kodlanır bu sonuçta:

e 000
a 010
n 0010
o 00110
u 00111
t 0110
m 0111
' ' 111

i 1000
x 10010
p 10011
h 1010
s 1011
f 1101
r 11000
l 11001

2.Graflar

Graf, bir olay veya ifadenin düğüm ve çizgiler kullanılarak gösterilmesi şeklidir.

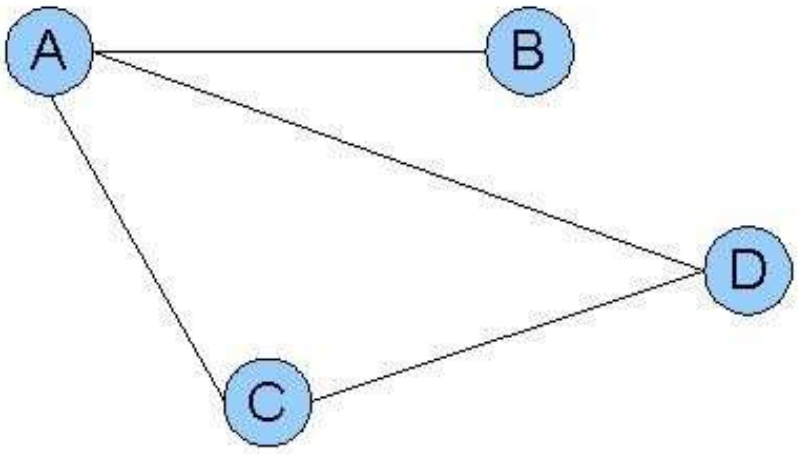
Grafta bulunan varlıklar düğümler ile ifade edilmekte, bu varlıklar arasındaki ilişkiler ise graftaki bağlantılar ile ifade edilmektedir.

Bağlantılara sayısal ağırlıklar ve yönler atanabilir.

Graflarda bir düğümden diğer düğümlere olan en kısa yol için dijkstra algortması kullanılır.

Graftaki bütün düğümleri dolaşmak için gereken en kısa yolu bulmak için prim veya kruskal algoritması kullanılır.

Graflarda Welsh-Powell algoritması kullanılarak graf renklendirme yapılır.

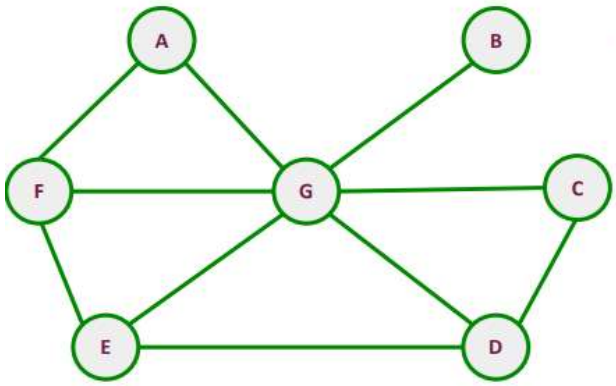


1.Graf Renklendirme

Graf renklendirme, graf üzerinde birbirine komşu olan düğümlere farklı renk atama işlemidir; amaç, en az sayıda renk kullanılarak tüm düğümlere komşularından farklı birer renk vermektir.

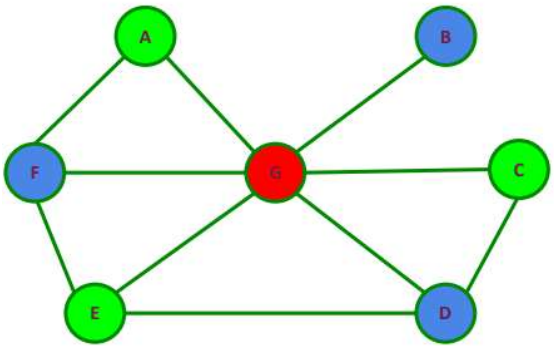
Renklendirmede kullanılan toplam renk sayısı kromatik (chromatik) sayı olarak adlandırılır.

Adım 1: Dügümler derecelerine göre büyükten küçüğe doğru sıralanır.



Düğüm	Derece
G	6
D	3
E	3
F	3
C	2
A	2
B	1

Adım 2: En yüksek dereceden başlanarak renk atanır. Bir sonraki düğüm başka bir renk atanarak ilerlenir. Komşusu ile aynı renk oluyor ise renk değiştirilir.



Düğüm	Derece
G	6
D	3
E	3
F	3
C	2
A	2
B	1

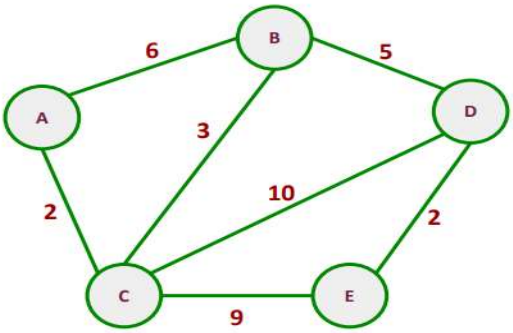
Kırmızı: G
Mavi: D,F,B
Yeşil A,E,C

Kromatik sayı: 3

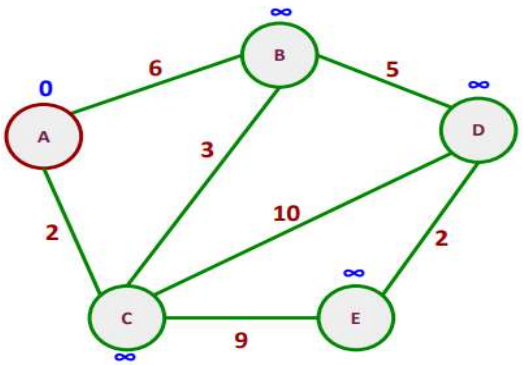
2.Dijsktra Algoritması

Dijkstra'nın algoritması belirli bir başlangıç noktasına göre en kısa yolu belirleyen bir algoritmadır Bir düğümden, yani başlangıç düğümünden, diğer tüm düğümlere olan en kısa yolu belirler Ağırlıklı ve yönlü graflar için geliştirilmiş olup graf üzerindeki her bir kenarın ağırlığı sıfır veya sıfırdan büyük sayıdır.

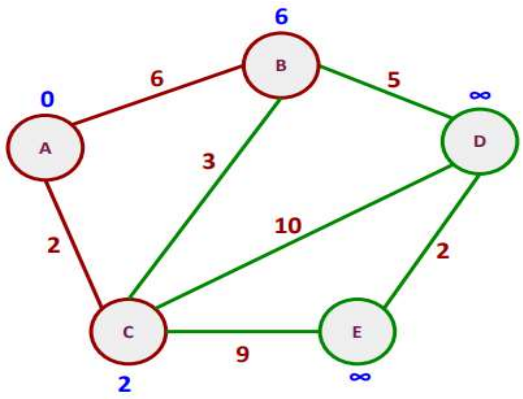
Dijkstrada bir başlangıç noktası seçilir. Daha sonra o noktanın komşulara olan uzalığı yazılır. Komşulardan daha yakın olan yani daha az maliyetli olan düğüme gidilir ve o düğümden o düğümün komşularına olan uzaklığı not alınır. Ama burada dikkat edilmesi gereken nokta tablonun yukarısından gelen maliyeti eklemeyi unutmamaktır. Bu gezme işlemi bütün düğümleri gezene kadar devam eder.



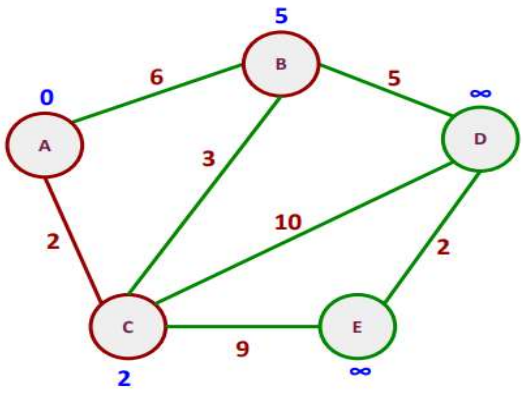
	A	B	C*	D	E



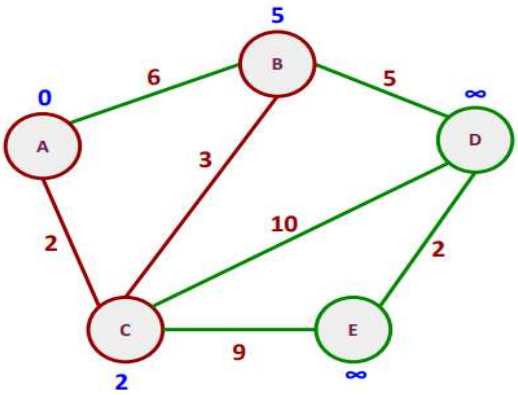
	A	B	C*	D	E
	0	∞	∞	∞	∞



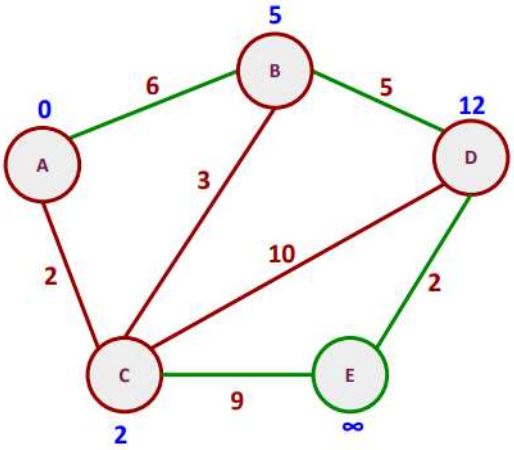
	A*	B	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞



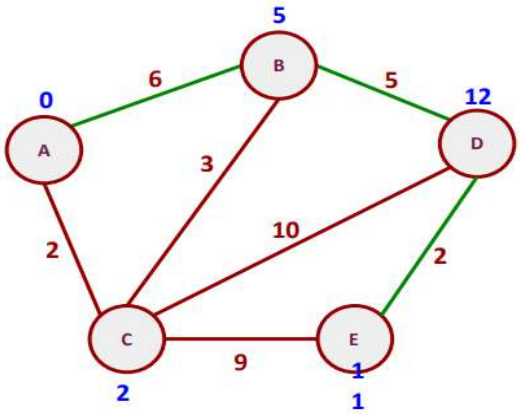
	A*	B	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0		2 (A)		



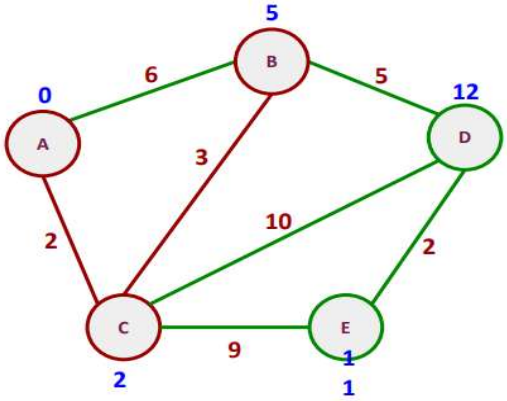
	A*	B	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)		



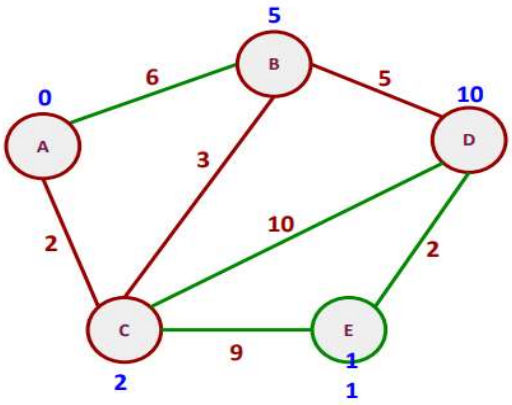
	A*	B	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	



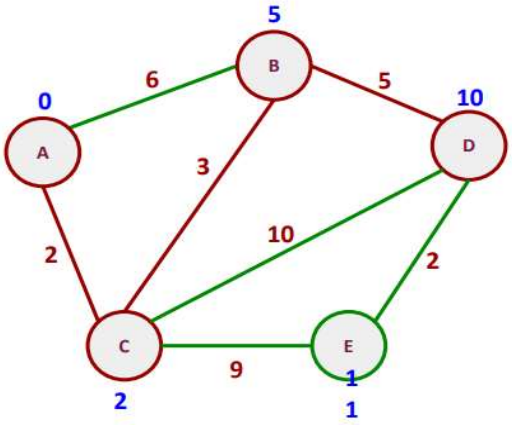
	A*	B	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	11 (A-C-E)



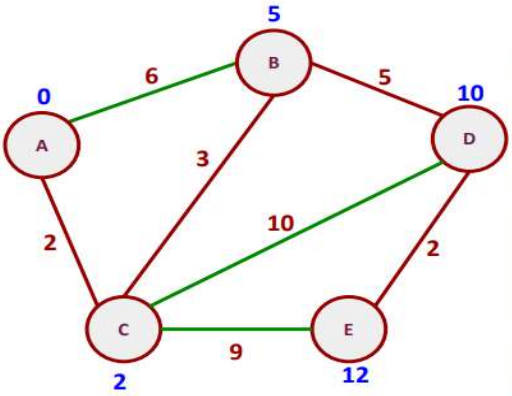
	A*	B	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	11 (A-C-E)
B	0	5 (A-C)	2 (A)		



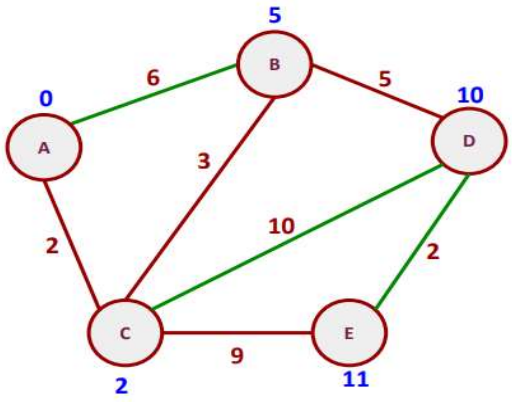
	A*	B*	C*	D	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	11 (A-C-E)
B	0	5 (A-C)	2 (A)	10 (A-C-B-D)	11 (A-C-E)



	A*	B*	C*	D*	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	11 (A-C-E)
B	0	5 (A-C)	2 (A)	10 (A-C-B-D)	11 (A-C-E)
D	0	5 (A-C)	2 (A)	10 (A-C-B-D)	



	A*	B*	C*	D*	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	11 (A-C-E)
B	0	5 (A-C)	2 (A)	10 (A-C-B-D)	11 (A-C-E)
D	0	5 (A-C)	2 (A)	10 (A-C-B-D)	



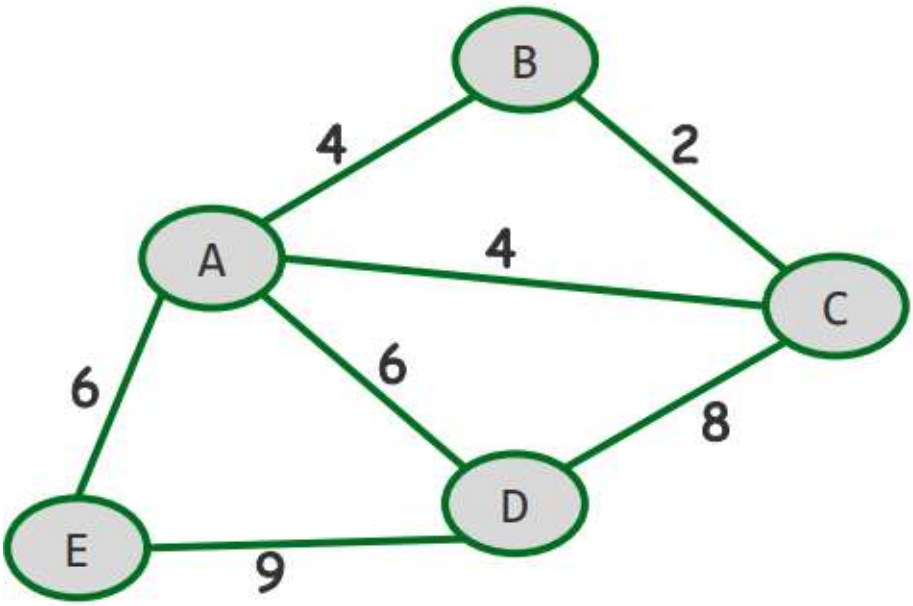
	A*	B*	C*	D*	E
	0	∞	∞	∞	∞
A	0	6 (A)	2 (A)	∞	∞
C	0	5 (A-C)	2 (A)	12 (A-C-D)	11 (A-C-E)
B	0	5 (A-C)	2 (A)	10 (A-C-B-D)	11 (A-C-E)
D	0	5 (A-C)	2 (A)	10 (A-C-B-D)	11 (A-C-E)

3.En Kısa Yol Ağacı

En kısa yol ağacı bütün ağacı en az maliyetle dolaşmamızı sağlar. 2 farklı yolu vardır.

- Prim

Bir başlangıç düğümü seçilir. Daha sonra en az maliyetli komşuya gidilir. Gidilebilecek yollara bakılır ve en az maliyetli yola gidilir. Burada dikkat edilmesi gereken en önemli konu gezilen düğümlerin bir döngü oluşturmamasıdır.



A noktasını başlangıç noktası varsayalım. Üç komşusu var ikisi 4 diğeri ise 6 maliyetlidir. Bu yüzden ikinci düğümümüz B düğümüdür.

Daha sonra B'den 2 maliyetli bir yol A'dan ise 4 ve 6 maliyetli yollar vardır. O yüzden 2 maliyetli yolla devam ederiz. C düğümüne ilerledik.

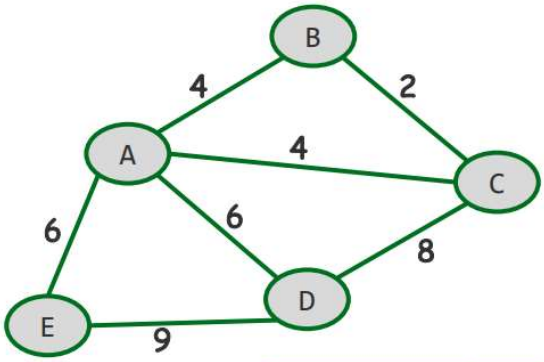
Şimdi ise C'den 8 maliyetli, A'dan ise 6 maliyetli iki yol vardır. Burada A'dan istediğimiz yolu kullanabiliriz. Çünkü hiçbir yol döngü oluşturmuyor. 6 maliyetli E yoluna ilerledik.

Ziyaret etmediğimiz sadece D düğümü kaldı onu ziyaret edebileceğimiz 3 yol var. Biri

A'dan 6, ikincisi C'den 8, üçüncüsü E'den 9 maliyetli. Burada seçmemiz gereken yol A'dan 6 maliyetli olandır ve böylece Prim ağacı oluşmuştur.

- **Kruskal**

Kruskalda kenarlar maliyeti en düşükten en yükseğe olacak şekilde sıralanır. Daha sonra sıra sıra kenarlar döngü yapıp yapmadığı dikkate alınarak çizilir. Böylece Kruskal ağacı oluşmuş olur.



Kenarlar	B-C	A-B	A-C	A-D	A-E	C-D	D-E
Maliyet	2	4	4	6	6	8	9