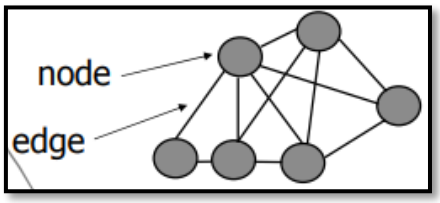
Çizgeler (Graphs)

Graf (çizge), bilgisayar dünyasında ve gerçek hayatta çeşitli sebeplerle karşılaşılan bir olay veya ifadenin düğüm ve çizgiler kullanılarak gösterilmesi şeklidir.
V (Verteks) ile gösterilen node'ları yani noktaları, E (Edge) ile gösterilenler kenarları simgeler.
Her kenar iki bilgi arasındaki ilişkiyi gösterir ve (u, v) şeklinde ifade edilir. (u, v) iki node'u gösterir.

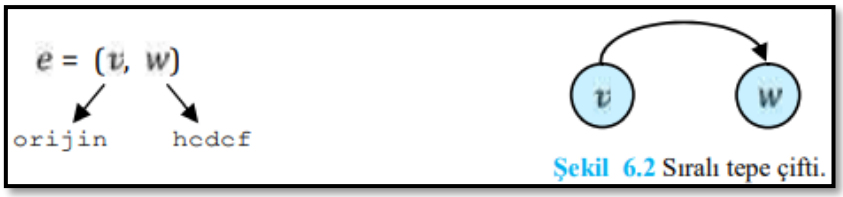


Herbir kenar iki node arasındaki uzaklığı ifade etmektedir.

Kenar Türleri

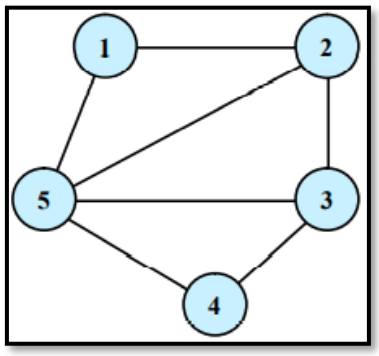
1- **Yönsüz Ayrıt (Undirected Edge)**
Çizgi şeklinde yönü belirtilmeyen ayrıtlar yönsüz ayrıtlardır. (v, w) ile (w, v) olması arasında fark yoktur.

2- **Yönlü Ayrıt (Directed Edge / digraph)**
Ok şeklinde gösterilen ayrıtlar yönlü ayrıtlardır. Birinci node ***orijin,*** ikinci node ise ***hedef*** olarak adlandırılır. Örneğin Şekil 6.2'de (v, w) ile (w, v) aynı değildir.

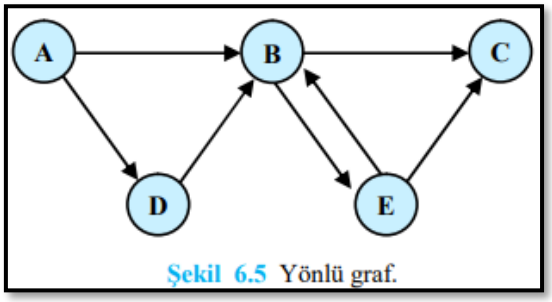


Şekil 6.2 Sıralı tepe çifti.

3- **Yönsüz Graf (Undirected Graph)**
Tüm ayrıtları yönsüz olan grafa yönsüz graf denilir. Yönsüz grafta bir node çifti arasında en fazla bir ayrıt olabilir.



4- **Yönlü Graf (Directed Graph, Digraph)**
Tüm ayrıtları yönlü olan grafa yönlü graf adı verilir. Yönlü grafta bir node çifti arasında ters yönlerde olmak üzere en fazla iki ayrıt olabilir.



Şekil 6.5 Yönlü graf.

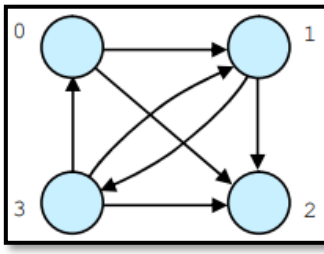
5- Self Ayrıt (Döngü –Loop)

(v, v) şeklinde gösterilen ve bir node'u kendine bağlayan ayrıttır.

Şekil 6.3 Self ayrıt.

6- Komşu Node'lar (Adjacent)

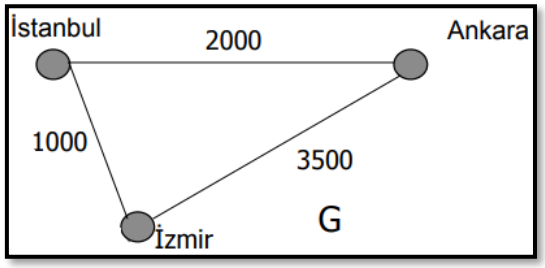
Aralarında doğrudan bağlantı (ayrıt) bulunanlar komşudur.



Tepe	Komşuları (*Adjacents*)
0	{1,2}
1	{2,3}
2	{}
3	{0,1,2}

7- Ağırlıklı Graf (Weighted Graph)

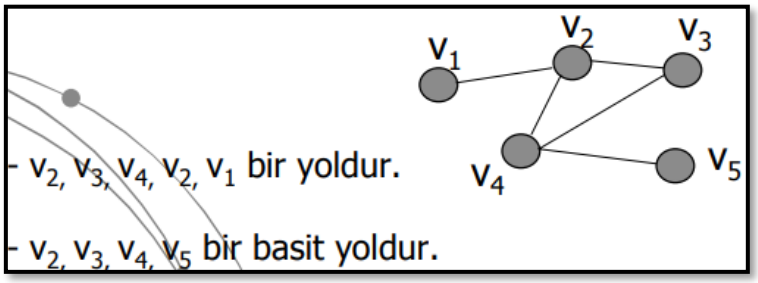
Eğer G (V, E) şeklinde gösterilen bir grafta her E kenarına bir ağırlık değeri atanmış ise ağırlıklandırılmış graf olarak adlandırılır.



8- Yol ve Basit Yol (Simple Path)

Bir yol v_1'den v_k'ya kadar sıralı node'ları (v_1, v_2), (v_2, v_3), …, $(v_k\text{-}1, v_k)$ kenarlarıyla birbirine bağlar.

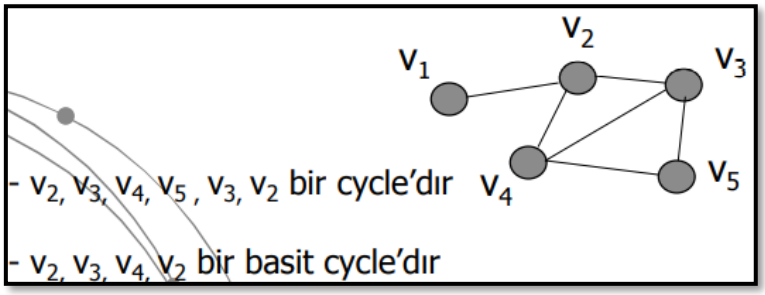
Basit yolda her bir node sadece bir kez bulunur.



9- Cycle (Çevrim) ve Basit Cycle

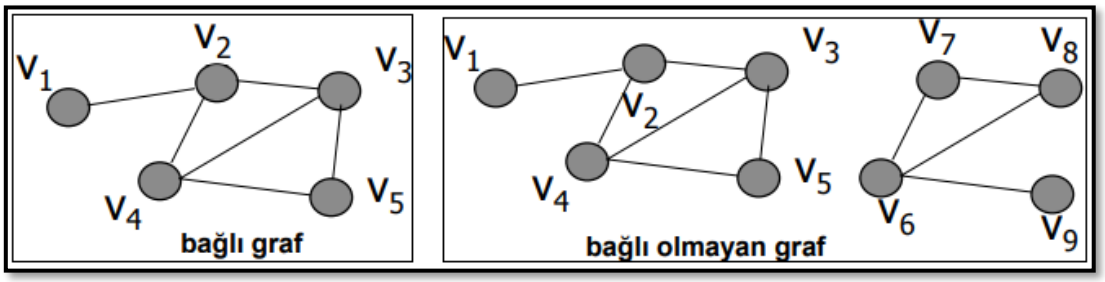
Cycle'da başlangıç ve bitiş node'ları aynıdır.

Basit cycle'da başlangıç ve bitiş node'ları hariç tüm node'lar sadece bir kez bulunur.



10- Bağlı ve Bağlı Olmayan Graf

Eğer bir graftaki tüm node'lar arasında en azından bir yol varsa **bağlı graftır.**
Bir grafta herhangi bir düğümden, herhangi başka bir düğüme her zaman için bir yol mevcut ise ona **bağlı graf** denir.
Eğer bir grafta herhangi iki node arasında yol bulunmuyorsa **bağlı olmayan graftır.**



11- Uzunluk

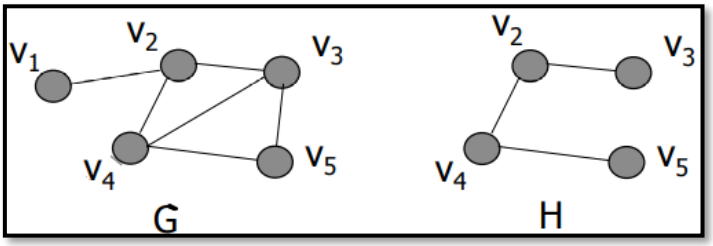
Bir yol üzerindeki ayrıtların sayısı o yolun uzunluğudur.

12- Bir Node'un Derecesi

Bir grafın node'una bağlı olan ayrıtların sayısına v'nin derecesi denir. Self-ayrıtlar 1 olarak sayılır.

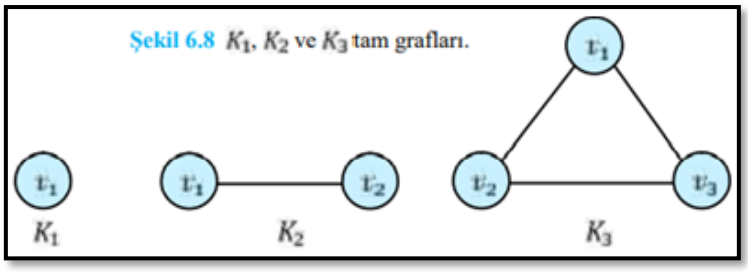
13- Alt Graf (Subgraph)

G (V, E) şeklinde gösterilen bir grafın alt grafı H(U, F) ise U ⊆ V ve F ⊆ E olur.



14- Tam (Komple) Graf

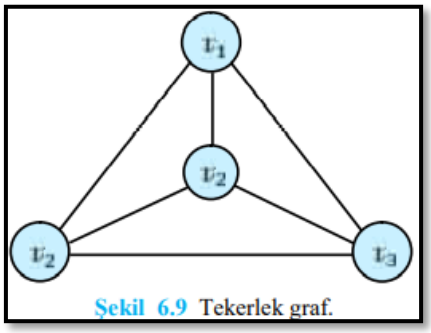
Eğer bir graftaki her iki node arasında bir kenar varsa komple graftır.



Komple grafın kenar sayısı = $\frac{n(n-1)}{2}$

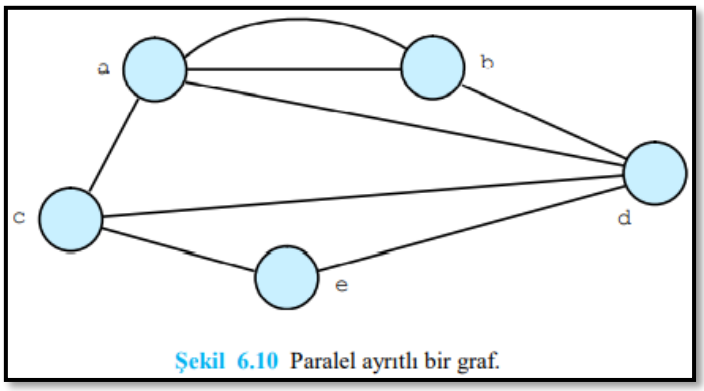
15- Tekerlek (Wheel) Graf

Bir cycle grafına ek bir node eklenerek oluşturulan ve eklenen yeni node'un, diğer bütün node'lara bağlı olduğu graftır.



16- Paralel Ayrıtlar (Parallel Edges)

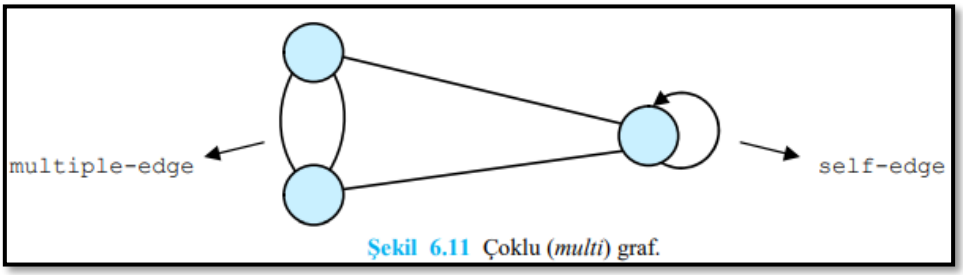
İki veya daha fazla ayrıt bir node çifti ile bağlanmıştır. Şekil 6.10'da a ve b iki paralel ayrıt ile birleşmiştir.



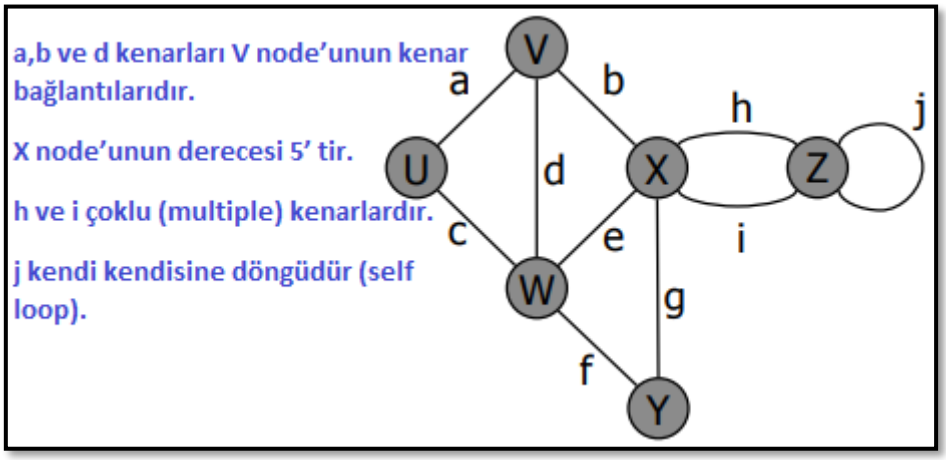
Şekil 6.10 Paralel ayrıtlı bir graf.

17- Çoklu (Multi) Graf

Multigraf iki node arasında birden fazla ayrıta sahip olan veya bir node'un kendi kendisini gösteren ayrıta sahip olan graftır.

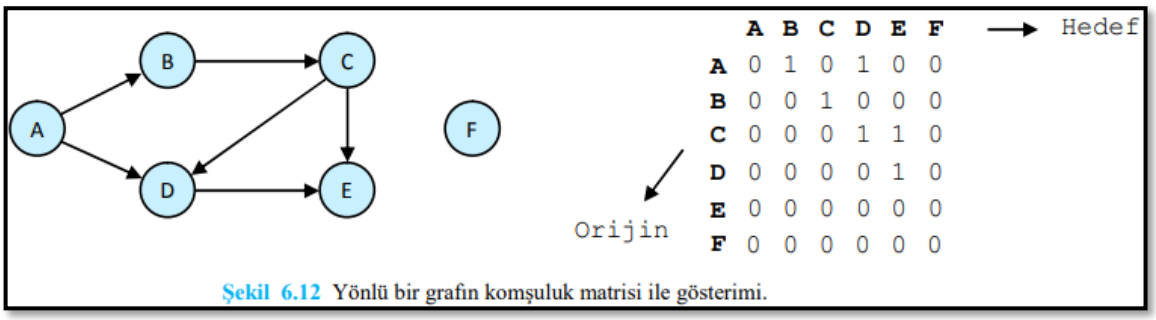


Şekil 6.11 Çoklu (*multi*) graf.



Komşuluk Matrisi (Adjacency Matrix)

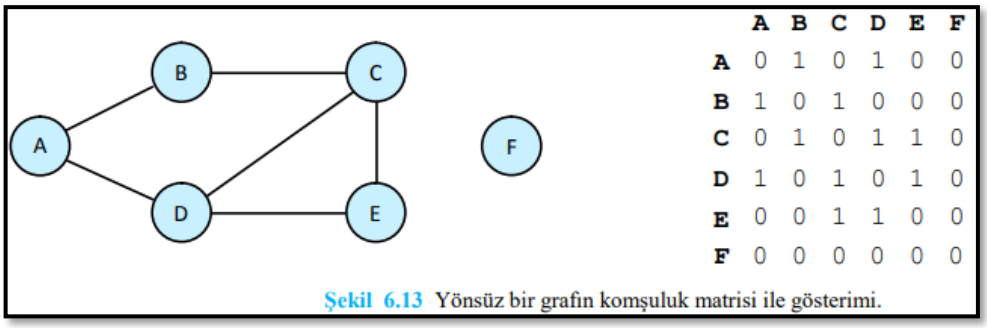
Node'lardan node'lara olan bağlantıyı boolean değerlerle ifade eden bir kare matristir.
Eğer (A, B) ∈ ise 1, diğer durumlarda ise 0 olarak işaretlenir.
Karmaşık O(v^2) olur.



	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	1	1	0
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Şekil 6.12 Yönlü bir grafın komşuluk matrisi ile gösterimi.

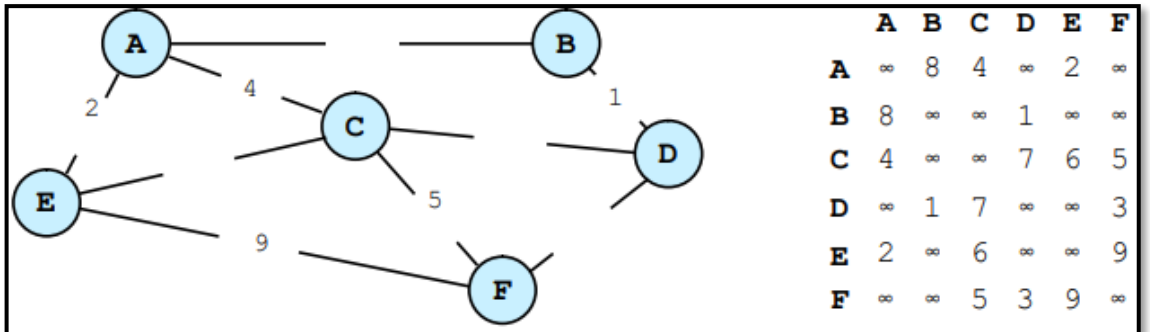
Yönlü grafların matrisi genelde simetrik olmaz fakat yönsüz graflarda bir simetriklik söz konusudur.



	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0

Şekil 6.13 Yönsüz bir grafın komşuluk matrisi ile gösterimi.

Şekil 6.14'te ağırlıklı ancak yönlendirilmemiş bir grafın komşuluk matrisi gösterilmiştir.

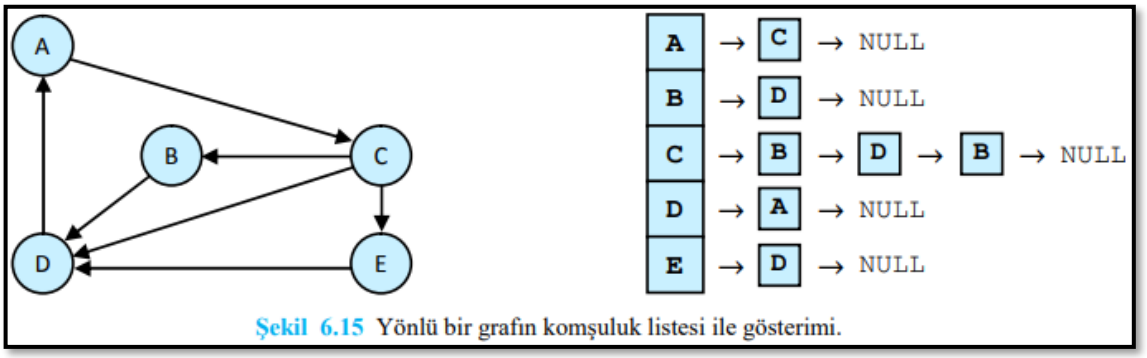


	A	B	C	D	E	F
A	∞	8	4	∞	2	∞
B	8	∞	∞	1	∞	∞
C	4	∞	∞	7	6	5
D	∞	1	7	∞	∞	3
E	2	∞	6	∞	∞	9
F	∞	∞	5	3	9	∞

Şekil 6.14 Yönlendirilmemiş ağırlıklı bir grafın komşuluk matrisi ile gösterimi.

Komşuluk Listesi (Adjacency List)

Her v node'u kendinden çıkan ayrıtları gösteren bir bağlı listeye sahiptir ve her bir node için komşu node'ların listesi tutulur.



Şekil 6.15 Yönlü bir grafın komşuluk listesi ile gösterimi.

```
#define tepe_sayisi 6 // tepe sayisinin 6 oldugu varsayiliyor
struct vertex {
        int node;
        struct vertex *nextVertex;
};
struct vertex *head[tepe_sayisi]; // 6 boyutlu, bir bagli liste basi
```

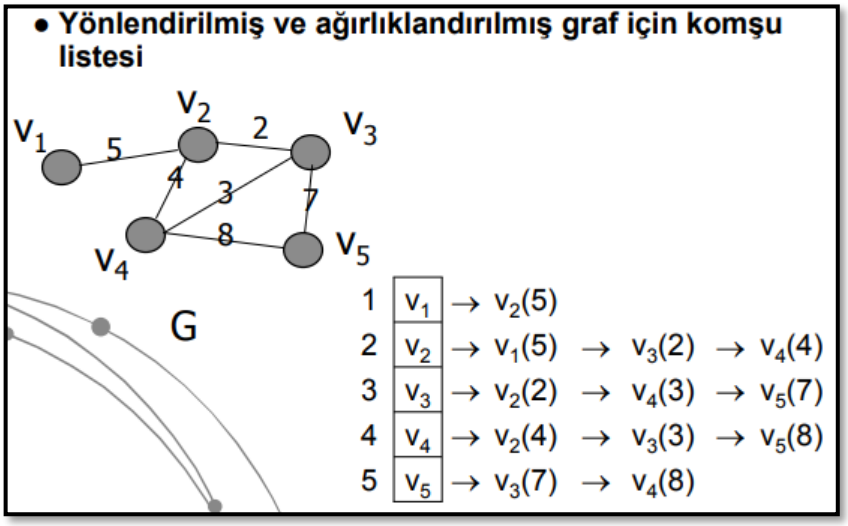
Komşuluk Matrisleri ve Komşuluk Listelerinin Avantajları-Dezavantajları

Komşuluk Matrisi

- Çok fazla alana ihtiyaç duyar. Daha az hafıza gereksinimi için seyrek (sparse) matris teknikleri kullanılmalıdır.
- Herhangi iki düğümün komşu olup olmadığına çok kısa sürede karar verilebilir.

Komşuluk Listesi

- Bir düğümün tüm komşularına hızlı bir şekilde ulaşılır.
- Daha az alana ihtiyaç duyar.
- Oluşturulması matrise göre daha zor olabilir.



Depth First Dolaşma

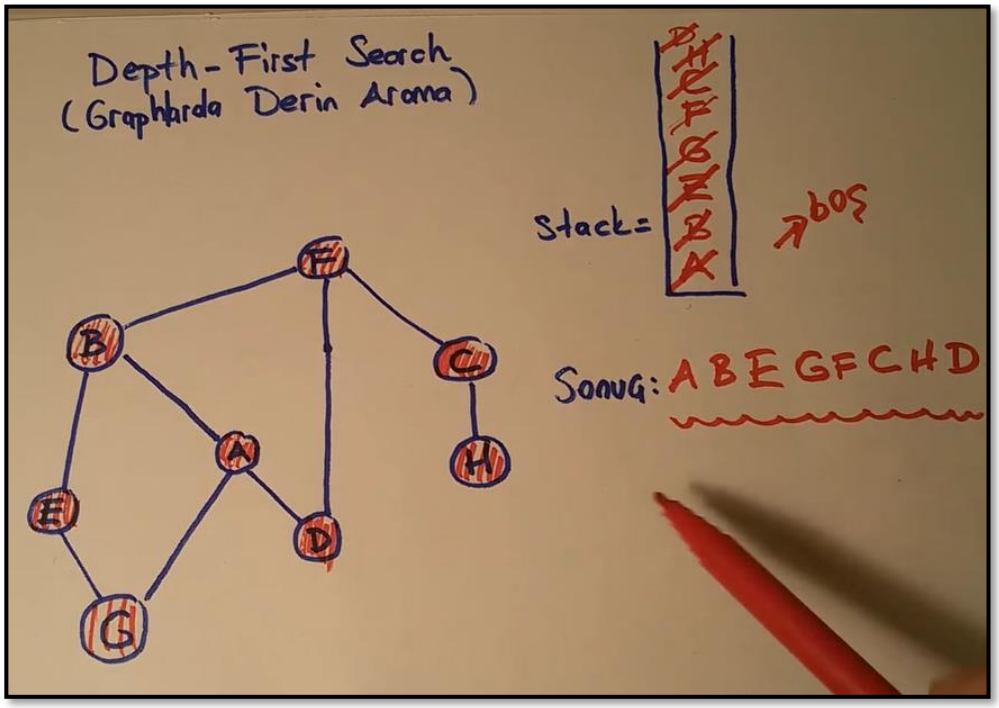
Bir node seçilir sonra o node'un bir komşusu seçilir ve ziyaret edilir. Ardından onun bir komşusu seçilir ve ard arda komşu seçimi yapılarak devam edilir. Komşu kalmadığında geri dönülür.

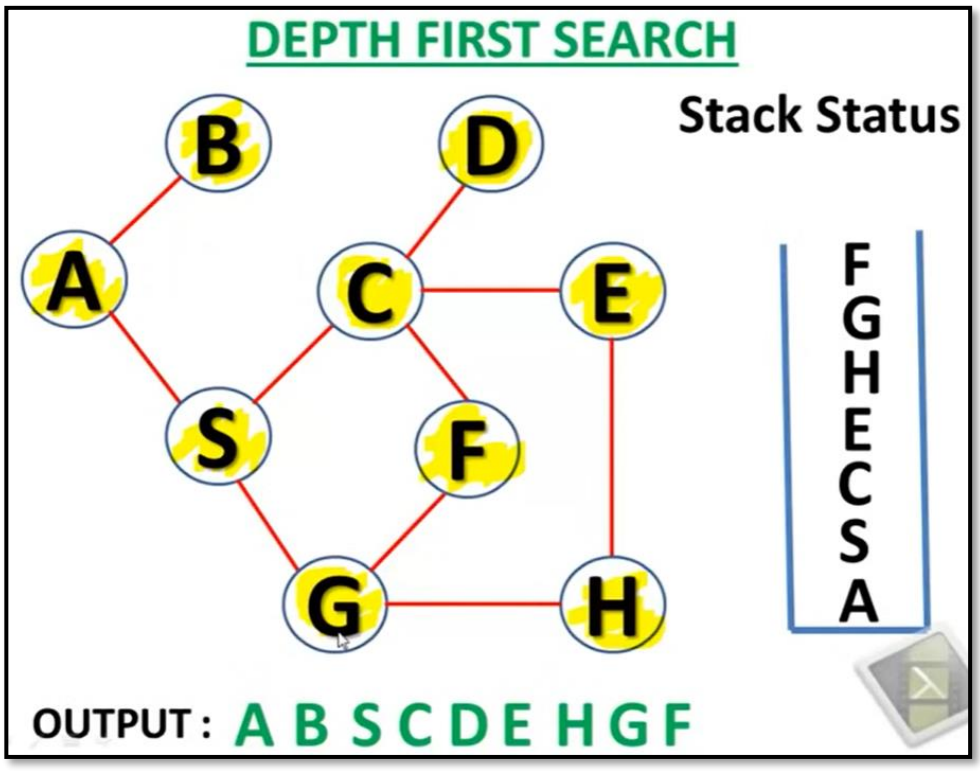
İkili ağaç veri yapısındaki ***preOrder Traversal*** gibi çalışır.

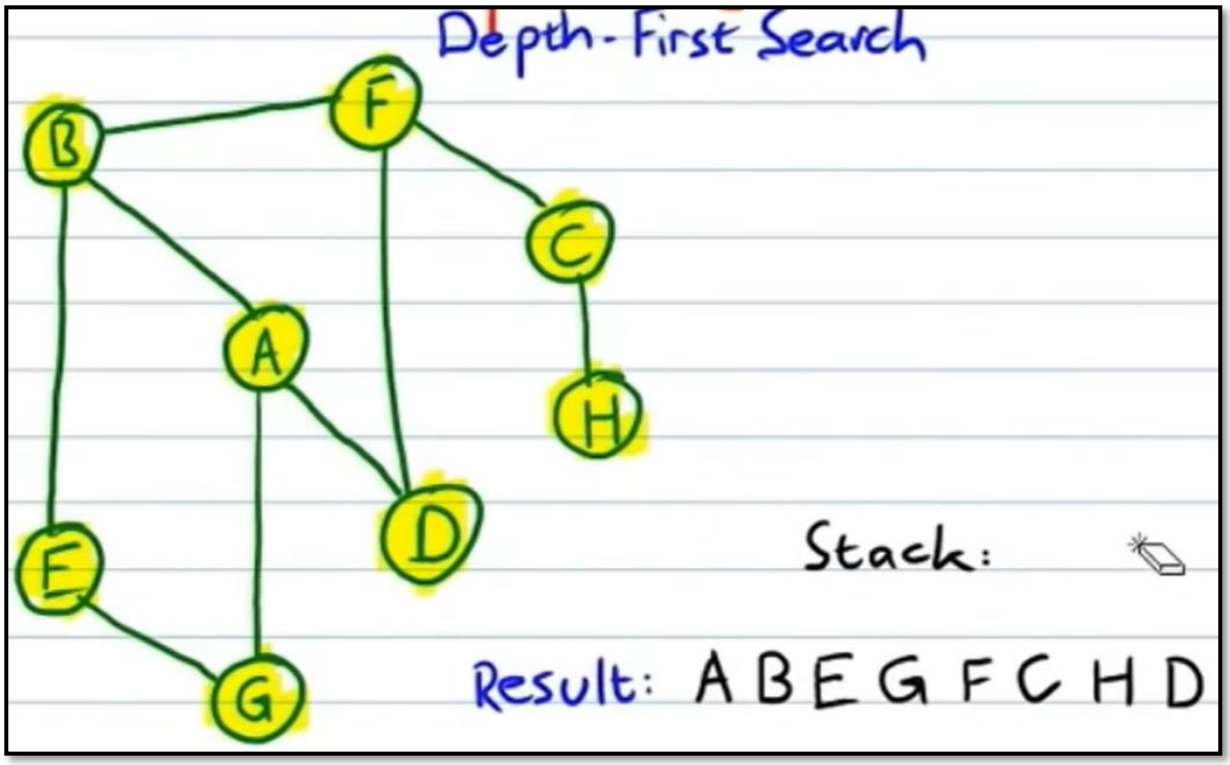
LIFO yani son gelen ilk çıkar mantığı vardır.

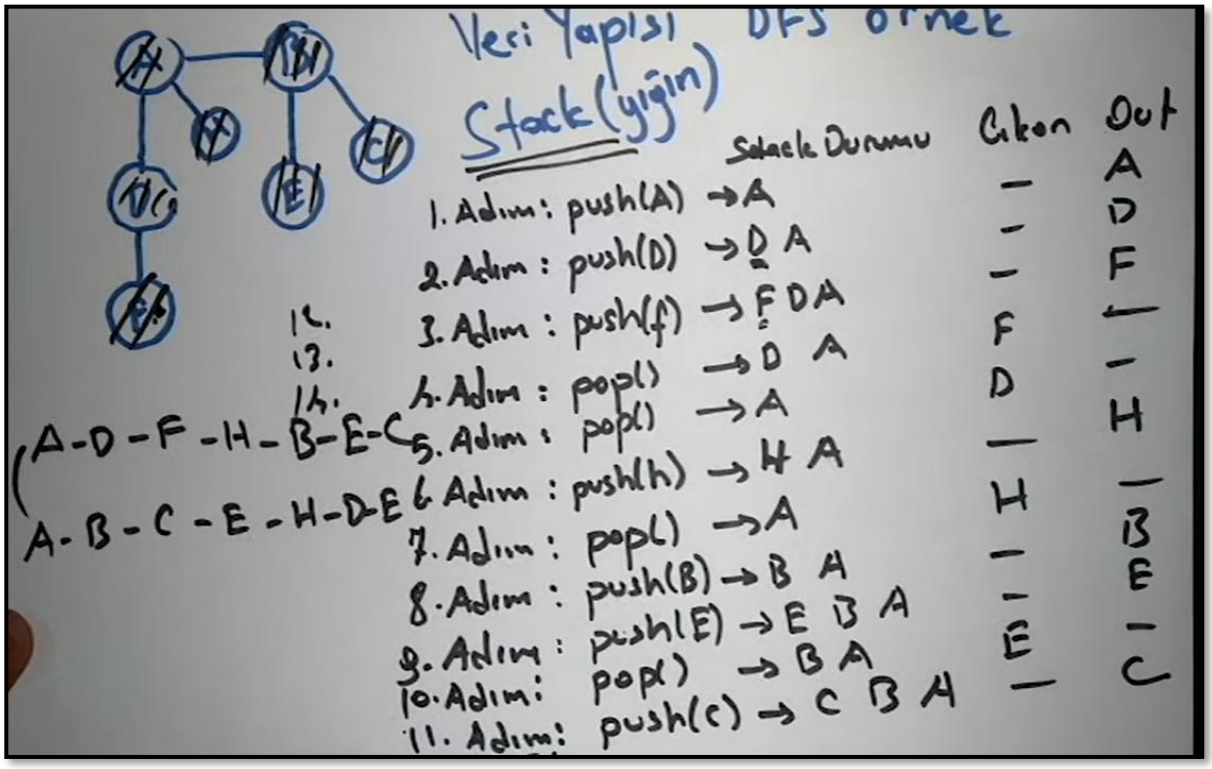
Depth First Arama İşlem Adımları

1. Önce bir başlangıç node'u seçilir ve ziyaret edilir.
2. Seçilen node'un bir komşusu seçilir ve ziyaret edilir.
3. 2.adım ziyaret edecek komşu kalmayıncaya kadar tekrar edilir.
4. Komşu kalmadığında tekrar geri dönülür ve önceki ziyaret edilmiş node'lar için adım 2 ve 3 tekrar edilir.






Depth-First Search
B
F
C
A
H
D
E
G
Stack:
Result: A B E G F C H D


Veri Yapisi DFS örnek
Stack (yigin)
Sstack Durumu
Cikan
Out
1. Adim: push(A) → A
2. Adim: push(D) → D A
3. Adim: push(f) → F D A
4. Adim: pop() → D A
5. Adim: pop() → A
6. Adim: push(h) → H A
7. Adim: pop() → A
8. Adim: push(B) → B A
9. Adim: push(E) → E B A
10. Adim: pop() → B A
11. Adim: push(c) → C B A
A-D-F-H-B-E-C
A-B-C-E-H-D-E

Breadth First Dolaşma

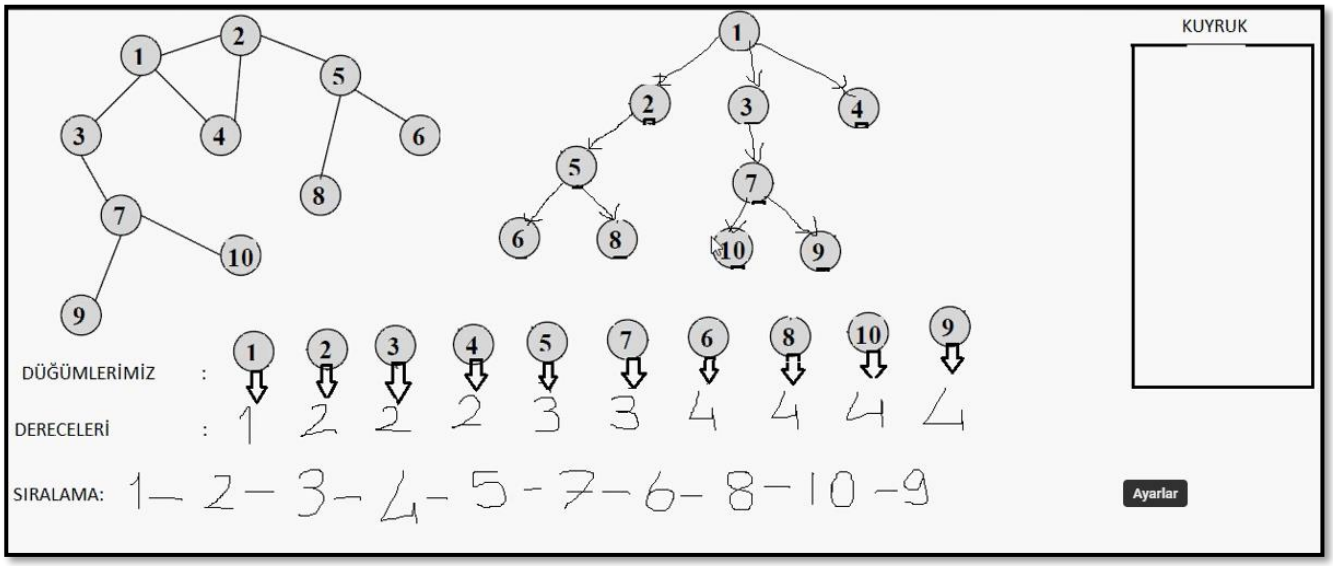
Bir node seçilir sonra o node'un sırasıyla tüm komşu node'larına gidilir ardından tüm komşu node'ların komşu node'larına gidilir.

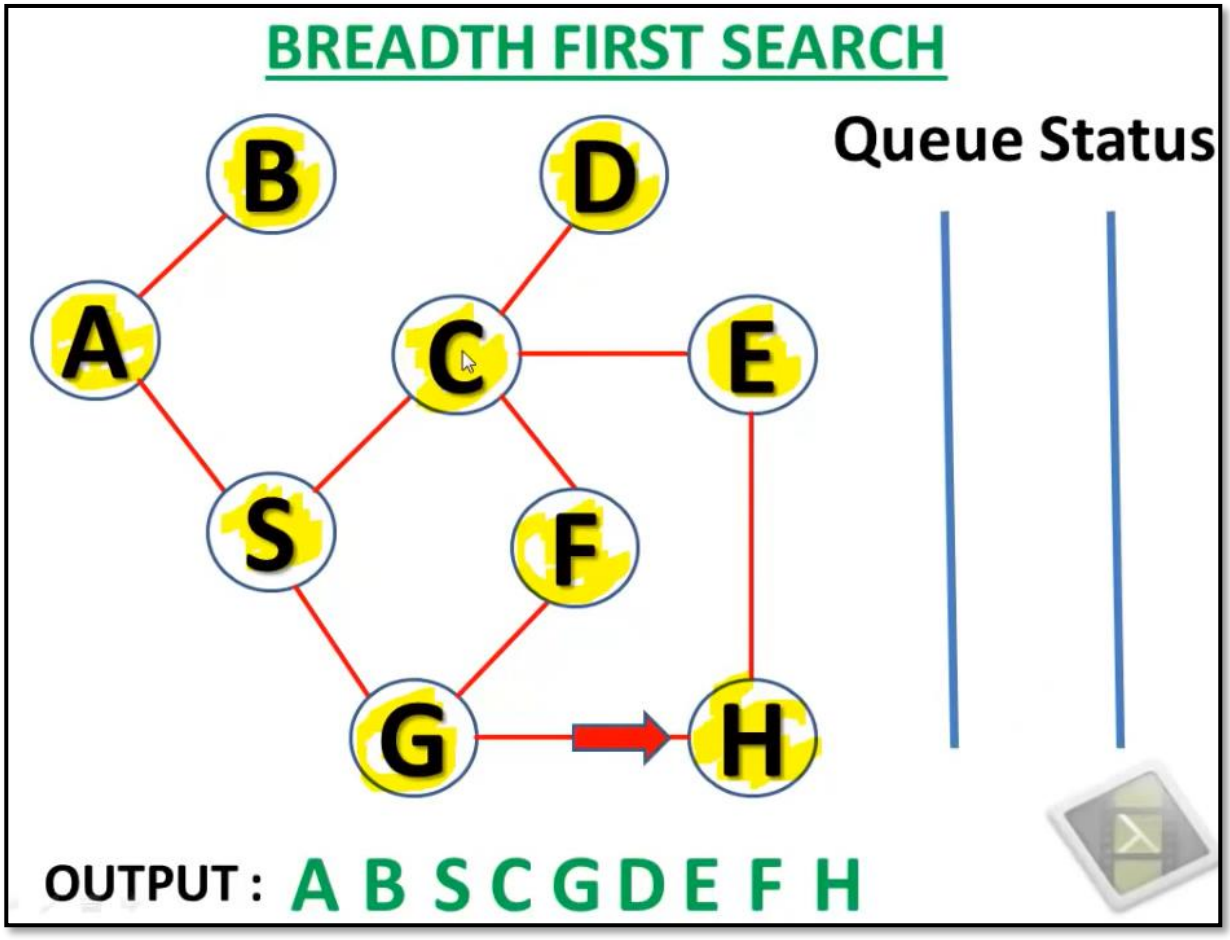
Breadth First arama ağaçlardaki ***level order*** aramaya benzer.

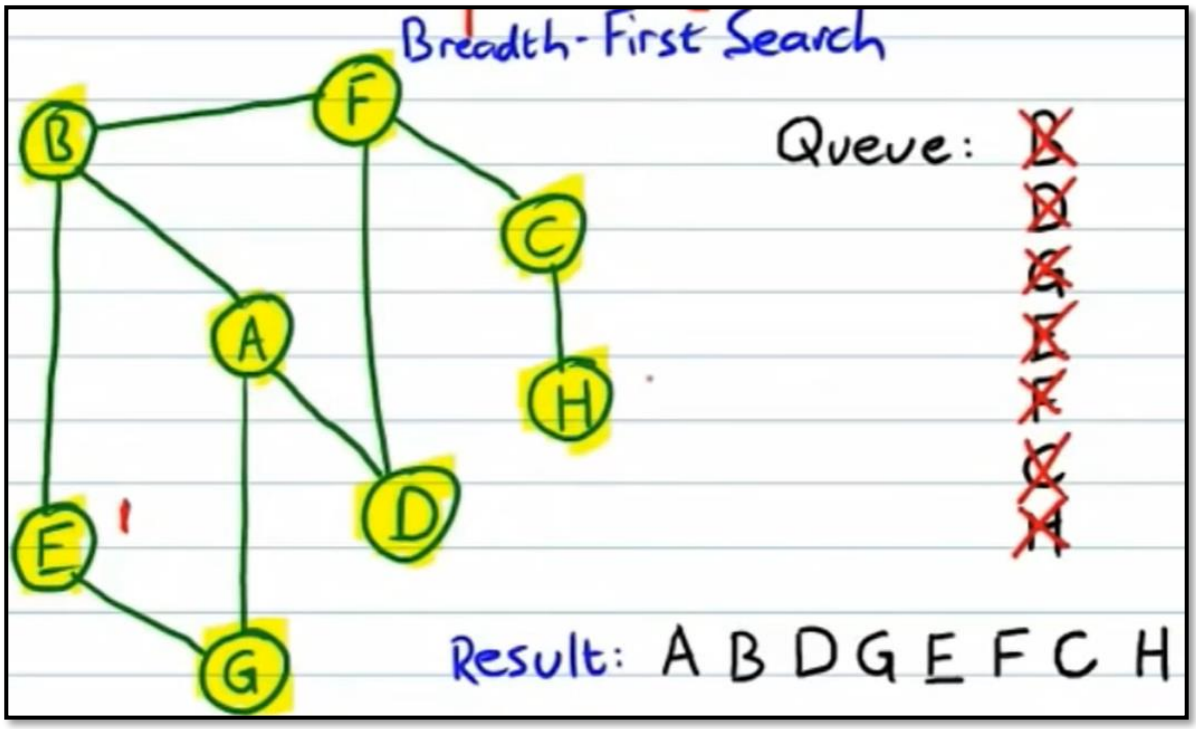
FIFO yani ilk gelen ilk çıkar mantığı vardır.

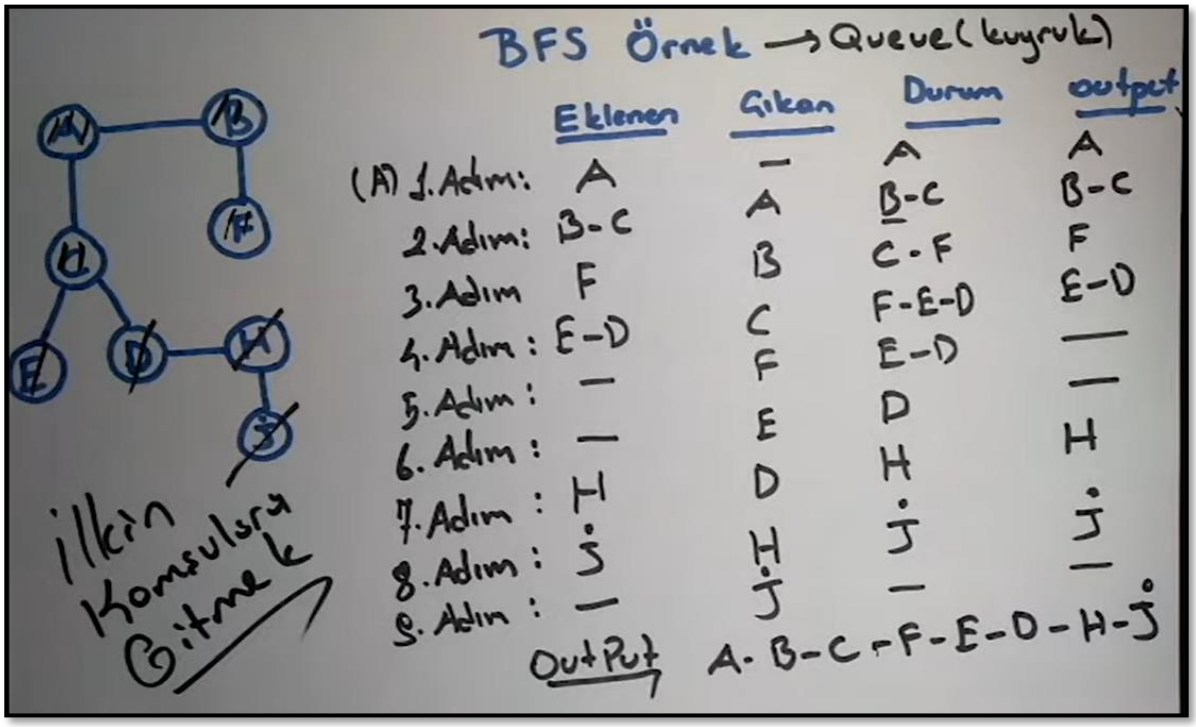
Breadth First Arama İşlem Adımları

1. Ziyaret edilmemiş komşu düğümü ziyaret et. Bu düğümü ziyaret edilmiş olarak işaretle.
2. Düğümü göster ve kuyruğa (queue) ekle.
3. Komşu kalmadığında Queue içerisindeki ilk node alınır ve 2.adıma gidilir.





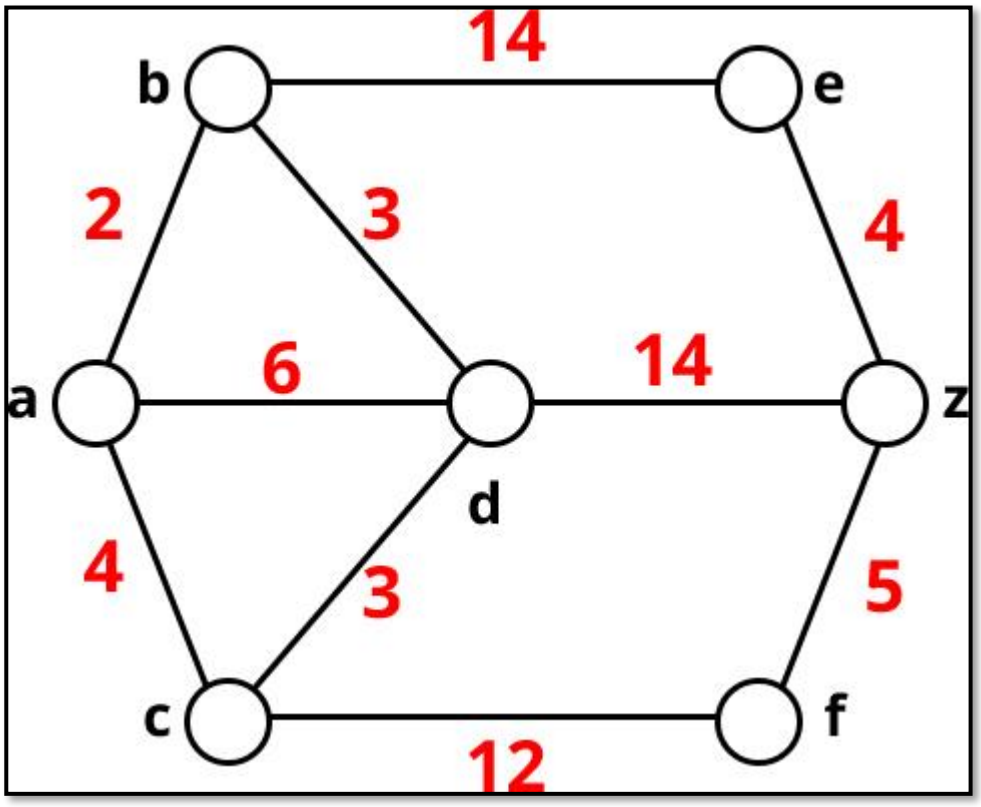

Breadth-First Search
B
F
C
A
H
D
E
G
Queue:
Result: A B D G E F C H


BFS Örnek → Queue (kuyruk)
Eklenen
Çıkan
Durum
output
(A) 1. Adım: A — A A
2. Adım: B-C A B-C
3. Adım F B C-F F
4. Adım: E-D C F-E-D E-D
5. Adım: — F E-D —
6. Adım: — E D —
7. Adım: H D H H
8. Adım: J H J J
9. Adım: — J — —
Output A-B-C-F-E-D-H-J
İlkin Komşuları Gitme

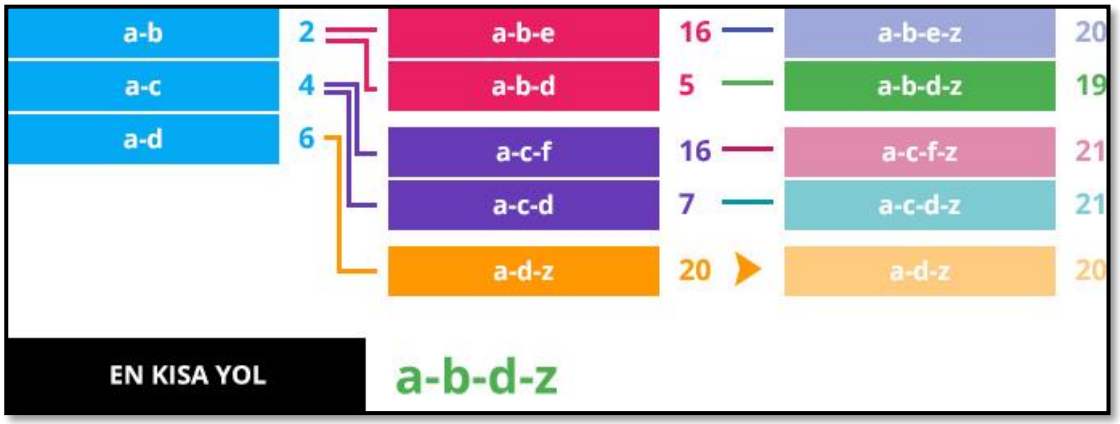
Dijkstra's Algoritması (https://www.youtube.com/watch?v=jT3c45XkPTg&t=309s)
Bu algoritmanın ağırlıkları negatif olmamalıdır.
Dijkstra algoritması, ağırlıklandırılmış çizgelerde bir başlangıç düğümü ile diğer düğümler arasındaki en kısa mesafeyi tespit etmek için kullanılır.
Dijkstra algoritmasının zaman karmaşıklığı $O(Mlog^N)$ 'dir.
Dijkstra's Algoritması'nın çalışması aşağıdaki gibidir.

1. İlk adım için hedef noktasına ***0*** diğer tüm düğümlere ***sonsuz*** atanır.
2. İkinci adımda ise seçilen her bir düğüm için kenarlar ayrı ayrı işleme girer bu işlem; **yola çıkılan düğüm + ilgili kenar maliyeti < hedef düğüm** ise hedef düğüm güncellenir.
3. Her düğüm için her kenar ayrı ayrı işleme girdiğinden performansı da **düğüm sayısı * kenar sayısı** olarak tanımlanabilir.

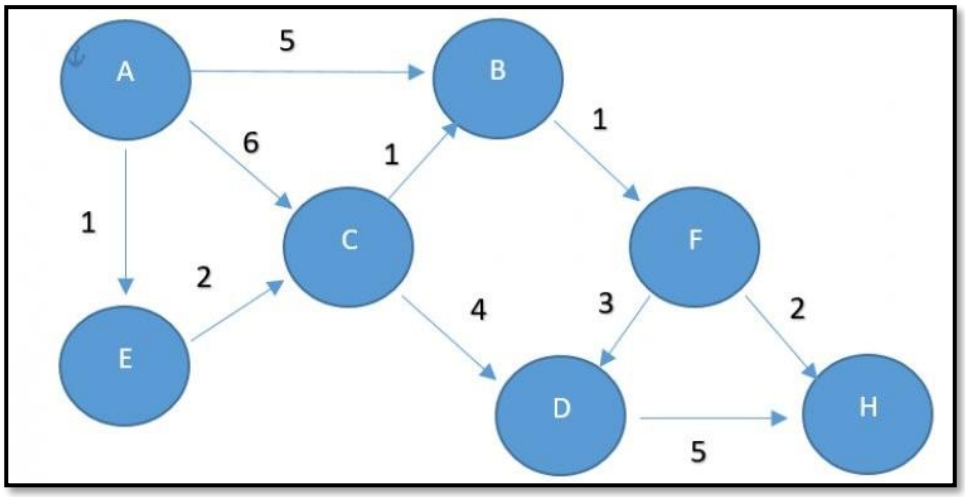
Örnek-1



Yukarıdaki şekilde belirtilen en kısa yolu bulmak için gösterimlerimizi şöyle yapabiliriz;



Örnek-2



1. Aşama

Başlangıç düğümü olarak A düğümünü seçelim.Başlangıçta diğer düğümlerin erişim imkanı olmadığı için uzaklık değerlerine sonsuz değerini atarız. Durum aşağıdaki gibi olur.

Durum	A	B	C	D	E	F	H
Başlangıç	0	∞	∞	∞	∞	∞	∞

2. Aşama

Başlangıç düğümünden başlayarak komşu olan düğümlerine olan uzaklıklarını durum tablomuzda gösteriyoruz ve en kısa olan yolun düğümünü seçerek bir sonraki düğüme geçiyoruz. Şeklimizde A düğümünün 3 düğüme komşuluğu bulunmaktadır ve en az maliyetli olan E düğümü seçiyoruz.

Durum	A	B	C	D	E	F	H
Başlangıç	0	∞	∞	∞	∞	∞	∞
A		5	6	∞	1	∞	∞

3. Aşama

E düğümünün C düğümüne komşuluğu bulunmakta. C düğümünün yeni değeri 1+2=3 oldu. C değerine E üzerinden gitmek daha az maliyetli olduğu için C değeri üzerinde güncelleme yaptık. Bir sonraki düğüme geçmek için en küçük değere sahip ve daha önceden ziyaret edilmemiş olan C'den devam edelim.

Durum	A	B	C	D	E	F	H
Başlangıç	0	∞	∞	∞	∞	∞	∞
A		5	6	∞	1	∞	∞
E(1)		5	3	∞		∞	∞

4. Aşama

C'den B ve D düğümlerine komşuluk bulunmakta. B'nin yeni değeri 3+1=4<5 olduğu için B düğümün değerini güncelliyoruz. D'nin yeni değeri 3+4=7 oldu. Durum tablosunda B düğümünün değeri en düşük olduğu için B düğümünden devam ediyoruz.

Durum	A	B	C	D	E	F	H
Başlangıç	0	∞	∞	∞	∞	∞	∞
A		5	6	∞	1	∞	∞
E(1)		5	3	∞		∞	∞
C(3)		4		7		∞	∞

5. Aşama

B düğümünün F düğümüne komşuluğu bulunmakta ve 4+1=5 den F'nin yeni değerini güncelliyoruz.

Durum	A	B	C	D	E	F	H
Başlangıç	0	∞	∞	∞	∞	∞	∞
A		5	6	∞	1	∞	∞
E(1)		5	3	∞		∞	∞
C(3)		4		7		∞	∞
B(4)						5	∞

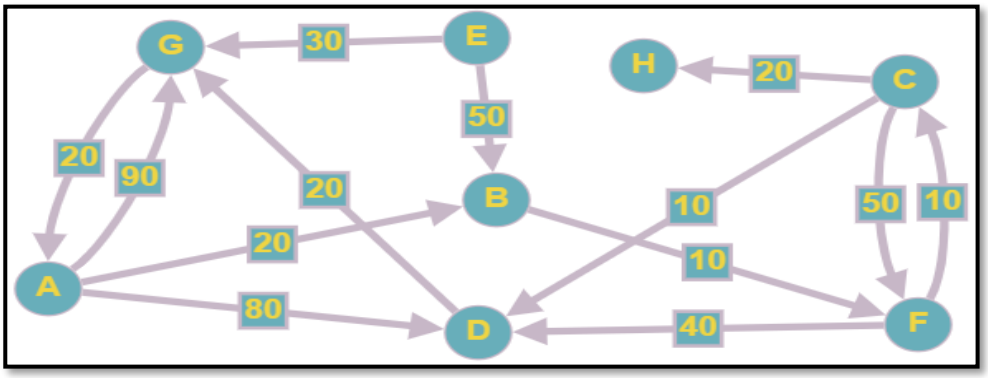
6. Aşama

F düğümü D ve H düğümüne komşuluğu bulunmakta. H düğümünün değeri 5+2=7 oldu. D düğümünün değeri 7 iken 5+3=8 olması gerekmekteydi ama 8>7 olduğundan mevcut değerini korudu.

Durum	A	B	C	D	E	F	H
Başlangıç	0	∞	∞	∞	∞	∞	∞
A		5	6	∞	1	∞	∞
E(1)		5	3	∞		∞	∞
C(3)		4		7		∞	∞
B(4)						1	∞
F(5)				7			7

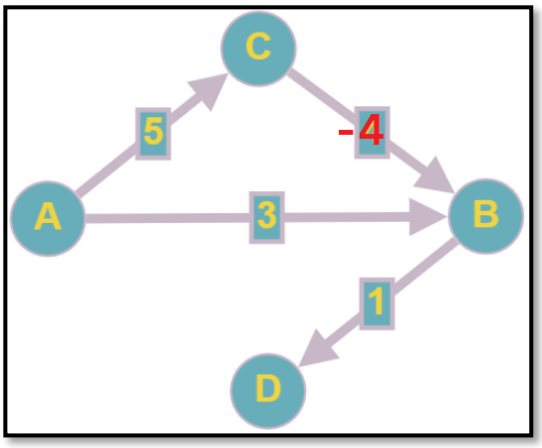
Sonuç, tüm düğümleri gezdiğimize göre A düğümünden H düğümüne en kısa yol **A->E->C->B->F->H** şeklindedir. Maliyeti ise **1+2+1+1+2=7'dir**.

Örnek-3



Gidilen Düğüm	A	B	C	D	E	F	G	H
Başlangıç	∞	∞	∞	∞	∞	∞	∞	∞
A	0	20 (A)	∞	80 (A)	∞	∞	90 (A)	∞
B	0	20 (A)	∞	80 (A)	∞	30 (B)	90 (A)	∞
F	0	20 (A)	40 (F)	70 (F)	∞	30 (B)	90 (A)	∞
C	0	20 (A)	40 (F)	50 (C)	∞	30 (B)	90 (A)	60 (C)
D	0	20 (A)	40 (F)	50 (C)	∞	30 (B)	70 (D)	60 (C)
H	0	20 (A)	40 (F)	50 (C)	∞	30 (B)	70 (D)	60 (C)
G	0	20 (A)	40 (F)	50 (C)	∞	30 (B)	70 (D)	60 (C)

Örnek-4



Gidilen Düğüm	A	B	C	D
Başlangıç	∞	∞	∞	∞
A	0	3 (A)	5 (A)	∞
B	0	3 (A)	5 (A)	4 (B)
D	0	3 (A)	5 (A)	4 (B)
C	0	5+(-4)=1 1 (C)	5 (A)	4 (B)

Özetle, bir graph'da negatif bir değer varsa Dijkstra's algoritması o graph için doğru sonuç üretmeyebilir. Çünkü en son satıra bakacak olursak B'ye 1 maliyetle gidebiliyoruz ve ordan D'ye geçerek maliyetimiz 2 oluyor ama biz yine son satıra baktığımızda maliyetin 4 olduğunu görüyoruz. Dolayısıyla bu algoritma negatif değer içerdiği için bize yanlış bir sonuç üretmiş oldu.

Bellman-Ford Algoritması

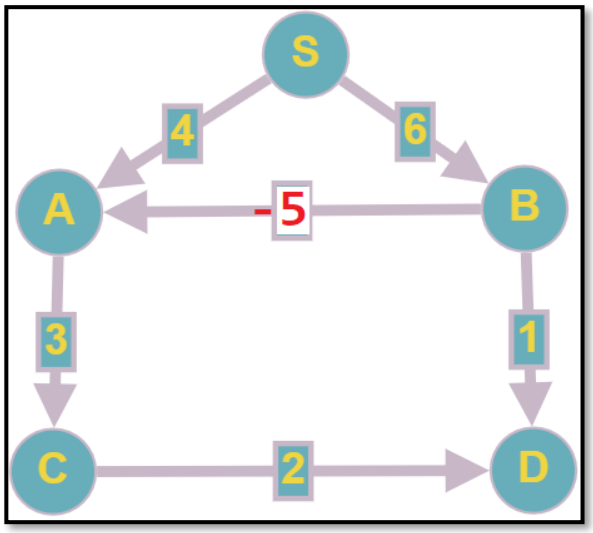
Bellman-Ford algoritmasının amacı graph üzerindeki bir node'dan hedef node'a giden en kısa yolu bulmaktır. Dijkstra algoritmasından farkı *eksi maliyetli* graph'larda, Dijkstra'nın aksine doğru sonuç üretebilmektedir.

Bellman-Ford Algoritmasının çalışması aşağıdaki gibidir.

1. İlk adım için hedef noktasına ***0*** diğer tüm düğümlere ***sonsuz*** atanır.
2. İkinci adımda ise seçilen her bir düğüm için kenarlar ayrı ayrı işleme girer bu işlem **yola çıkılan düğüm + ilgili kenar maliyeti < hedef düğüm** ise hedef düğüm güncellenir.
3. Her düğüm için her kenar ayrı ayrı işleme girdiğinden performansı da **düğüm sayısı * kenar sayısı** olarak tanımlanabilir.
4. **d(v) ≤ d(u) + w(u,v)**

|V| = n
|E| = m
O(n*m) olur.

Örnek-1



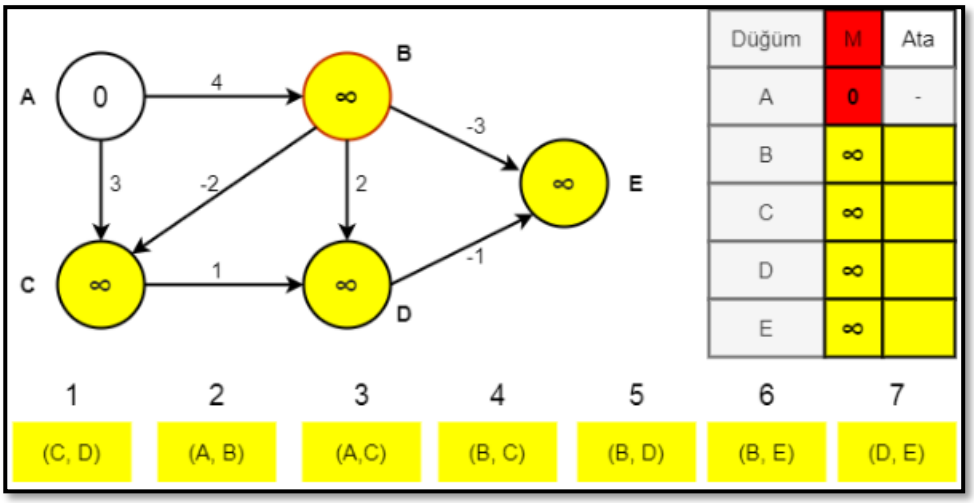
Kenar	Maliyet
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

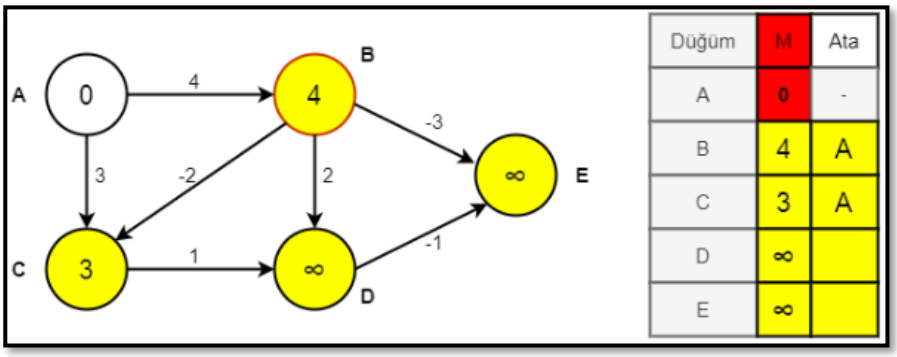
Gidilen Düğüm	S	A	B	C	D
Başlangıç		∞	∞	∞	∞
S	0	∞	∞	∞	∞
A	0	4 (S)	6 (S)	∞	∞
C	0	6+(-5)=1 1 (B)	6 (S)	7 (A)	6+1=7 7 (B)
D	0	1 (B)	6 (S)	1+3=4 4 (A)	4+2=6 6 (C)
B	0	1 (B)	6 (S)	4 (A)	6 (D)

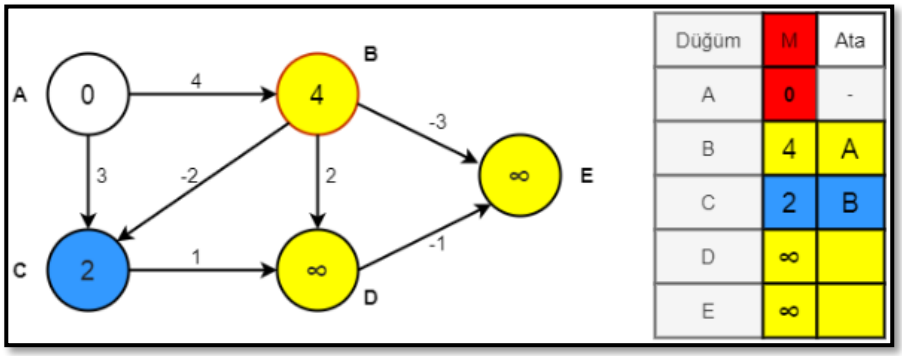
Şimdide her kenar için kontrol edelim.

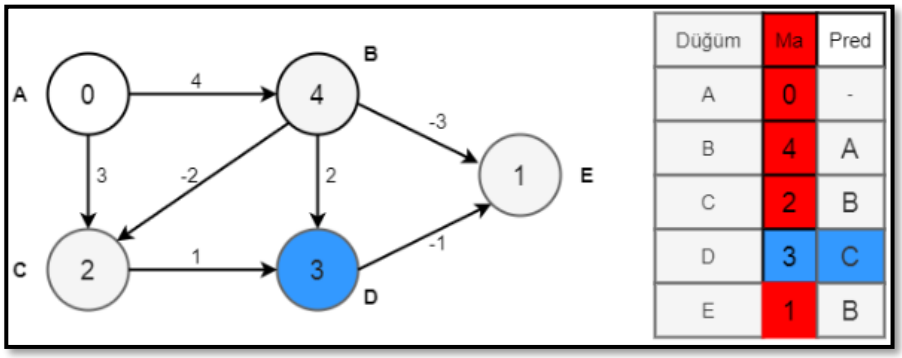
E(u, v)	d(v) ≤ d(u) + w(u, v)
E(A, C)	4 ≤ 1 + 3
E(B,A)	1 ≤ 6 + (-5)
E(B, D)	6 ≤ 6 + 1
E(C, D)	6 ≤ 4 + 2
E(S, A)	1 ≤ 0 + 4
E(S, B)	6 ≤ 6 + 0

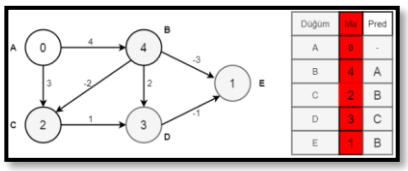
Örnek-2


A
B
C
D
E
0
∞
4
3
-2
2
-3
1
-1
Düğüm | M | Ata
A | 0 | -
B | ∞ |
C | ∞ |
D | ∞ |
E | ∞ |
1 (C, D)
2 (A, B)
3 (A,C)
4 (B, C)
5 (B, D)
6 (B, E)
7 (D, E)


A
B
C
D
E
0
4
3
∞
Düğüm | M | Ata
A | 0 | -
B | 4 | A
C | 3 | A
D | ∞ |
E | ∞ |

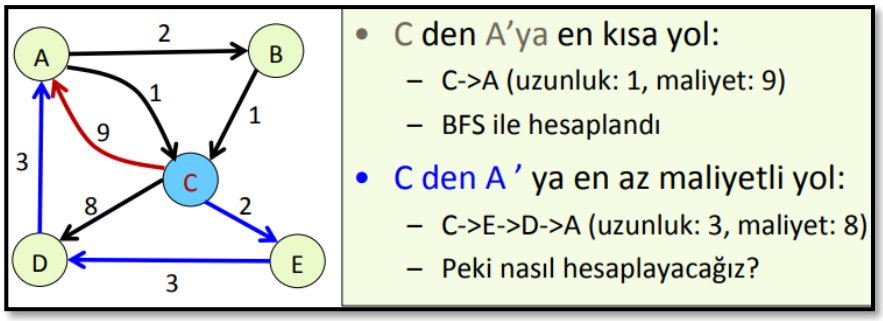

A
B
C
D
E
0
4
2
∞
Düğüm | M | Ata
A | 0 | -
B | 4 | A
C | 2 | B
D | ∞ |
E | ∞ |


A
B
C
D
E
0
4
2
1
3
Düğüm | Ma | Pred
A | 0 | -
B | 4 | A
C | 2 | B
D | 3 | C
E | 1 | B



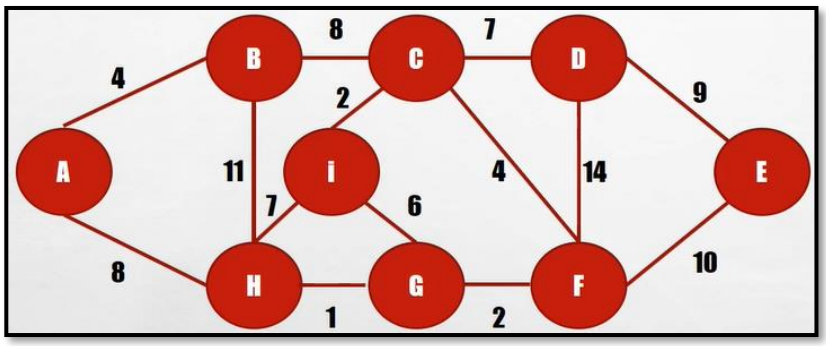
Minimum Spanning Tree (Minimum Yol Ağacı)

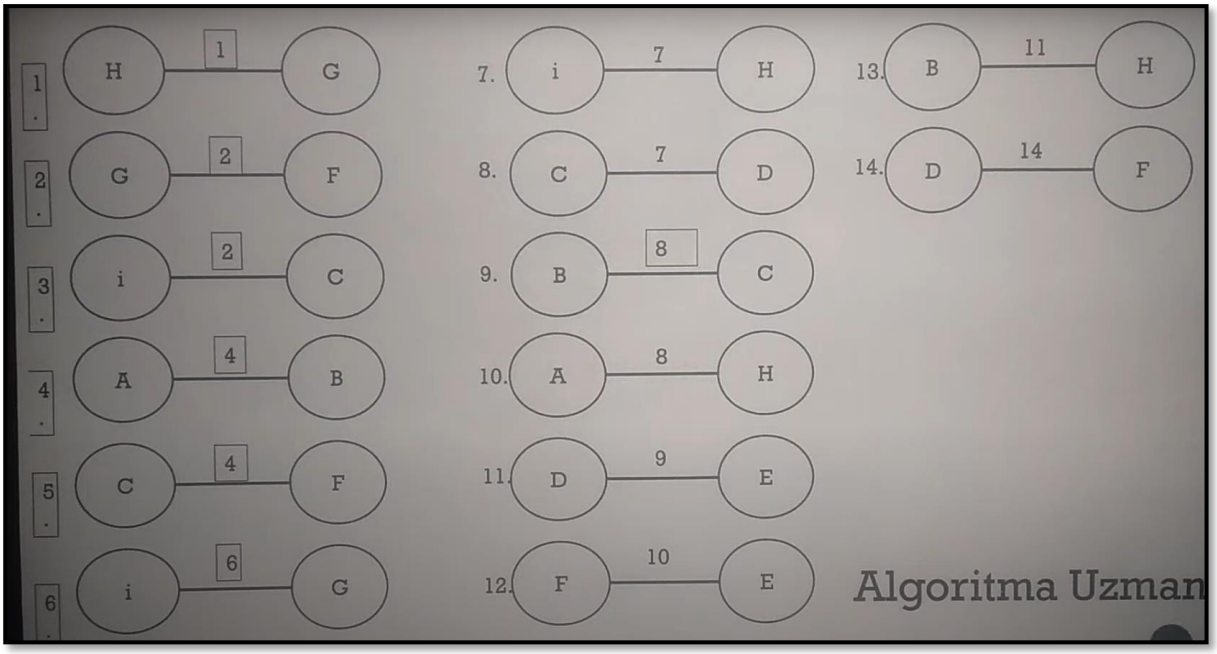
- Bir graf üzerinde birden çok yol ağacı olabilir; en az maliyetli olan en küçük yol ağacı olarak adlandırılır.
- **Dikkat!** Herhangi bir noktadan diğerine giden yol en az olsun demiyoruz, bu durumda problem en kısa yol (shortest path) problemi olurdu. Burada ağacın toplamının maliyeti minimal olmalıdır.
- Ağaç özelliği olduğu için **kapalı çevrim (çember-döngü-cycle) olmamalıdır.**
  Cycle'da başlangıç ve bitiş node'ları aynıdır.

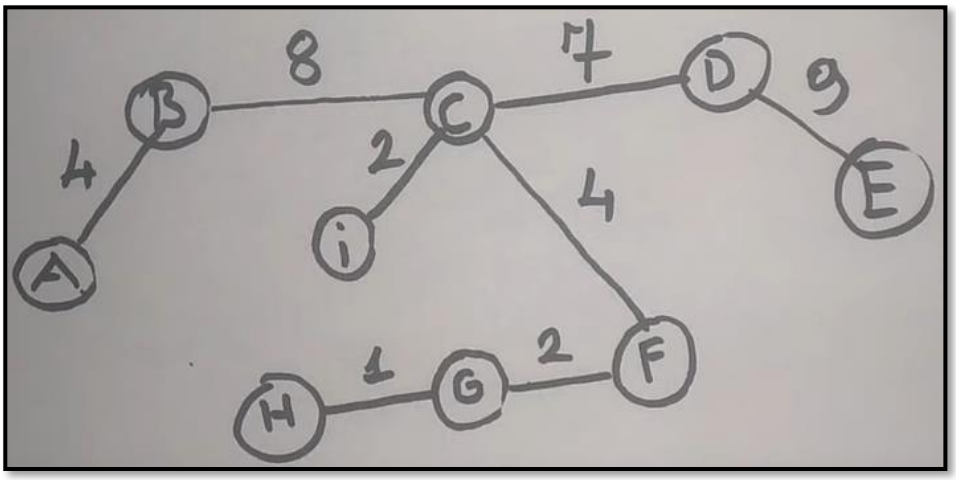


1. **Kruskal'ın Algoritması** (https://www.youtube.com/watch?v=BdKSkl02wBQ&t=4s)

- Tek tek maliyeti en az olan kenarlarla birleştirilerek düğümler arasında bağlantı olan tek bir küme oluşturulmaya çalışılır.
- **Küme birleştirme işlemine en az maliyetli olan kenardan başlanır**; daha sonra kalan kenarlar arasından en az maliyetli olanlar seçilir.
- **Kapalı çevrim (çember-döngü-cycle) olmamalıdır.**
- **O(E.logV)**, karmaşıklığına sahiptir. E kenar sayısıdır.

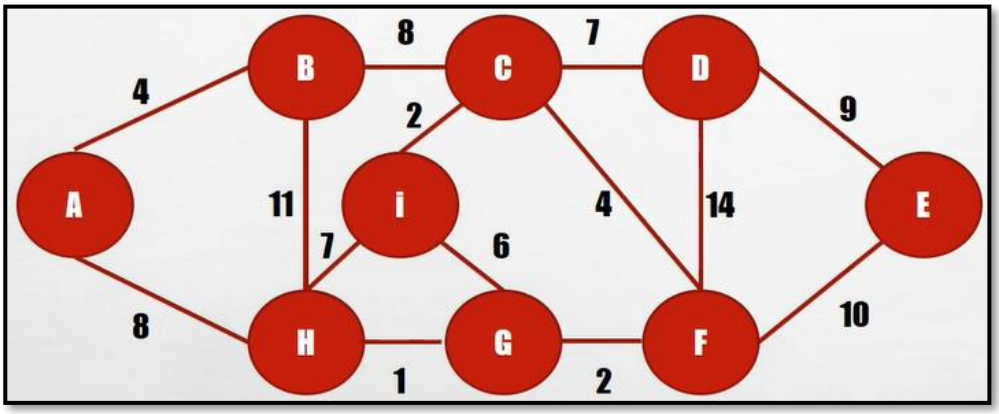


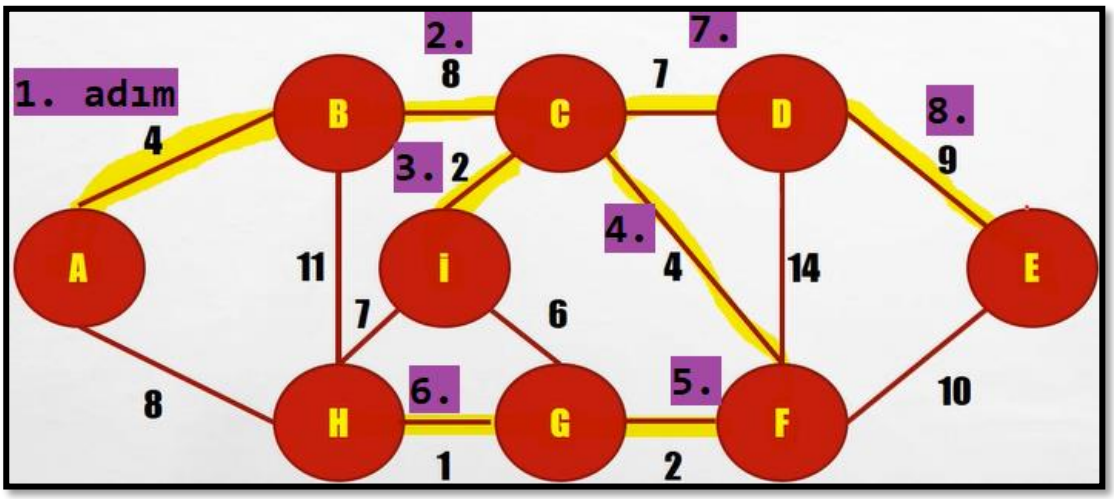




2. Prim'in Algoritması (https://www.youtube.com/watch?v=RPHc_IxNYdQ&t=264s)

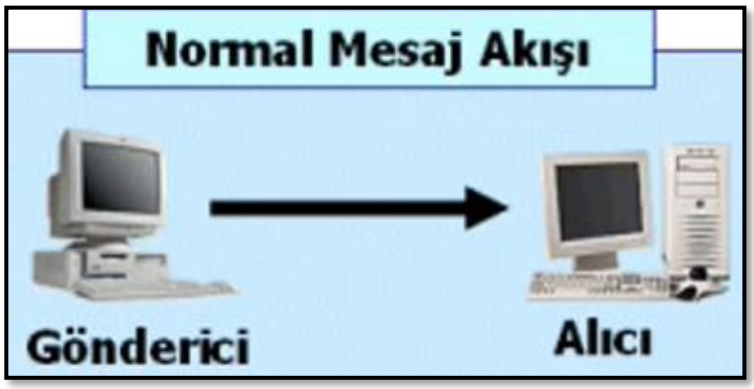
- **Cycle** oluşturmamak ve tüm düğümleri dolaşmak temel amaçtır.
- Başlangıçta, herhangi bir noktayı ağacı oluşturmaya başlamak için seç.
- Oluşturulan ağaca eklemek için, şu ana kadar oluşturulmuş ağaç üzeirnden erişilebilen ve daha **önceden ağaca katılmamış olan** en küçük ağırlıklı kenarı seç.
- Eğer bu kenarın ağaca katılması, bir çember oluşmasına sebep olmuyorsa, ağaca ekle.
- **O(v^2)** karmaşıklığına sahiptir.
- Karmaşıklığı daha az olduğu için liste veri yapısı kullanılır.





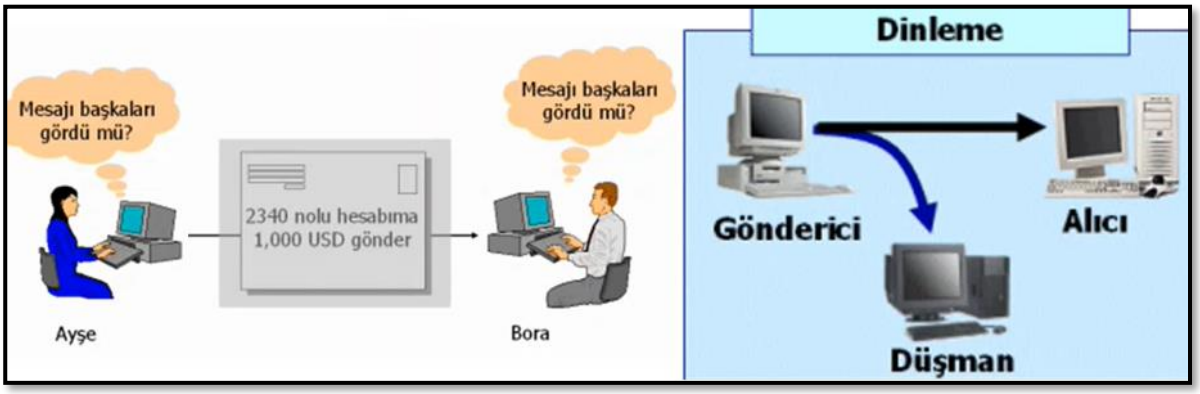
Elektronik Tehditler

- Haberleşen iki taraf, bilgisayar ağları, kablolu veya kablosuz iletişim kanalları kullanarak bir veriyi, mesajı bir taraftan diğerine iletirler.
- Elektronik ortamda haberleşen taraflar çeşitli tehditlerle karşı karşıya kalırlar.



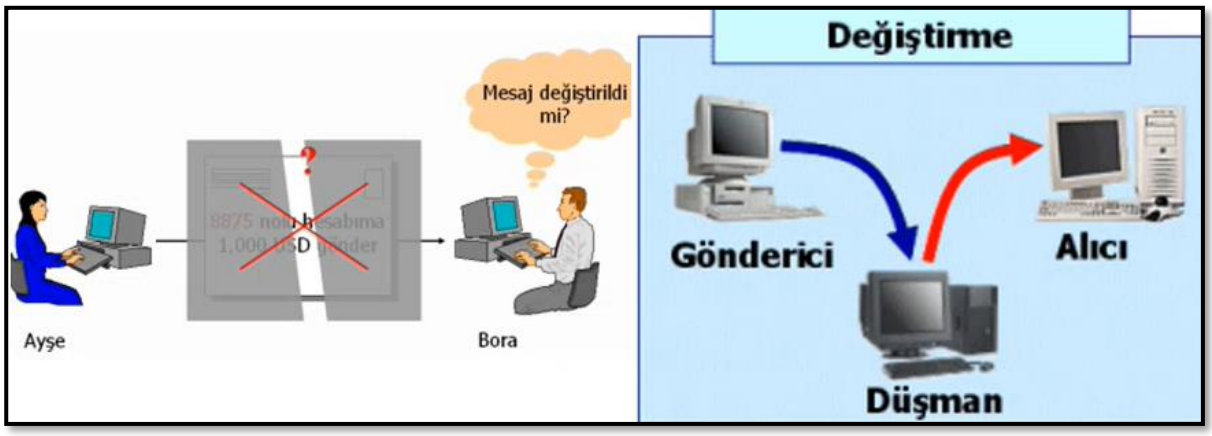
Gizlilik İhlali

- **Gizlilik:** Taşınan bilginin içeriğinin gizli kalmasıdır.
- Haberleşme kanalını dinleyen saldırgan gönderici ile alıcı arasındaki mesaj trafiğini dinleyebilir ve elde ettiği mesajları okuyarak bu haberleşmenın gizliliğini bozar.
- Bu tehdit dinleme tehdidi olarak bilinir.



Bütünlük İhlali

- **Bütünlük:** Taşınan bilginin içeriğinin yolda değiştirilememesidir.
- Haberleşmeye müdahale edip göndericinin mesajlarını değiştiren saldırgan, alıcıya giden mesajı istediği şekle sokabilir. Bu tehdit mesajın bütünlüğünü bozan değiştirme tehdididir.
- **Gizlilik ihlalinden farkı**, burda veriyi değiştiriyor lakin orda bir kopyasını alıyor.

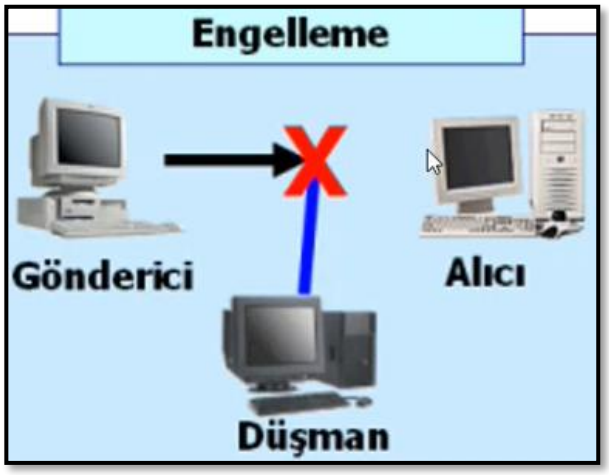


Kimlik Doğrulama İhlali

- **Kimlik Doğrulama:** Bilgiyi gönderen kişinin kimliğinin doğrulugundan emin olmaktır.
- **İnkâr Edememezlik:** Bilgiyi gönderen veya işleyen kişinin yaptığı işi sonradan inkar edememesidir.
- Mesajı gönderen veya alan tarafın bu işi yaptığını inkar etmesi söz konusu olabilir. Bu kötü niyetli girişimi boşa çıkaracak mekanizmalara ihtiyaç vardır.

Süreklilik İhlali

- **Haberleşmenin Sürekliliği:** Haberleşmenin kesintiye uğramadan yapılmasıdır.
- Saldırgan, haberleşen iki taraf arasındaki hattı veya haberleşme araçlarını kullanılmaz hale getirerek haberleşmenin sürekliliğini engellemeye çalışır.



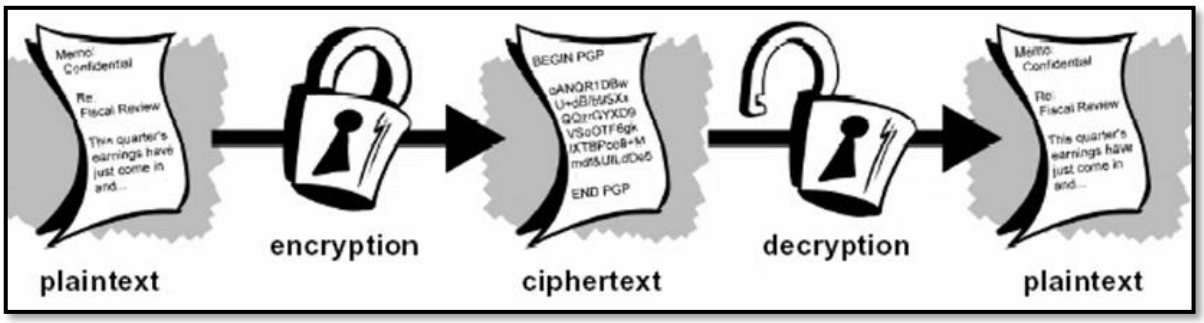
Elektronik Tedbirler

Yukarıda anlatılanlar için alınabilecek elektronik tedbirler aşağıda yer almaktadır.

- Gizlilik sağlamak için veri şifreleme yöntemleri kullanılır.
- Bütünlük sağlamak için özetleme algoritmaları, mesaj özetleri, elektronik imzalar kullanılır.
- Kimlik doğrulaması için özetleme algoritmaları, mesaj özetleri, elektronik imzalar ve sertifikalar kullanılır.
- İnkâr edememezlik için elektronik imzalar, işlem kayıtları kullanılır.
- Süreklilik için yedek sistemler, bakım, yedekleme, alternatif haberleşme kanalları kullanılır.

Şifreleme (Encryption) Nedir

- Bir açık metinin bir şifreleme algoritması yardımıyla anlaşılamaz hale getirilmesi işlemine **şifreleme (encryption)** denir.



- Şifrelenecek mesaj **plaintext (düz-metin)** olarak adlandırılır.
- **Şifreleme(encryption),** veriyi alıcının haricinde kimse okuyamayacak şekilde kodlamaktır.
- Şifrelenmiş mesaja **cipher text (şifreli-mesaj)** denir.
- **Şifre çözme (decryption)** ise şifrelenmiş veriyi çözüp eski haline getirme işlemidir.
- Veriyi şifrelerken ve çözerken kullanılan matematiksel metoda (toplama-çıkarma-çarpma-bölme ve lojik-mantıksal işlemlere) ise **şifreleme algoritması** denilmektedir.
  XOR en çok kullanılan mantıksal operatördür sebebi ise **tersine çevirme işleminin yapılabiliyor** olmasıdır.
- Şifreleme ve çözme genelde bir **anahtar (key)** kullanılarak yapılır.
- Aşağıdaki örnekte XOR mantıksal operatörüyle olarak basit bir şifreleme yapılmıştır.
  Plain text'imiz ve anahtarımız aşağıda yer aldığı gibidir.
  XOR mantıksal operatörü kullanılarak yani aynı değerse 0 farklı değerse 1 mantığı kullanılarak cipher text elde edilmiştir.
  Tekrar cipher text ve anahtar kullanılarka plain text elde edilmiştir sağda olduğu gibi.

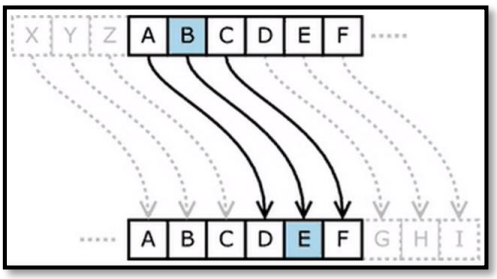
Plain text: 01101101	Cipher Text : 10111000
Anahtar : 11010101	Anahtar : 11010101
Cipher Text : 10111000	Plain Text : 01101101

Şifreleme Özellikleri

- Şifreleme yönteminde aranan bir takım özellikler vardır:
  - Şifreleme ve şifre çözme işleminin zorluğu ihtiyaç duyulan güvenlikle doğru orantılı olmalıdır.
  - **Anahtar seçimi ve şifreleme algoritması özel koşullara bağlı olmamalıdır**. Şifreleme yöntemi her türlü bilgi için aynı şekilde çalışmalıdır.
- Şifrelemede yapılan **hatalar sonraki adımlara yansımamalı ve mesajın tamamını bozmamalıdır**.
- Kullanılan algoritmanın **karıştırma özelliği** olmalıdır. Mesajın şifrelenmiş hali ile açık hali arasında ilişki kurulması çok zor olmalıdır.
- Kullanılan algoritmanın dağıtma özelliği olmalıdır. Mesajın açık hali şifreli hale gelirken içerdiği kelime ve harf grupları şifreli mesajın içinde olabildiğince dağıtılmalıdır.

Sezar Şifresi (Mono Alfabetik Şifreleme)

- Bu algoritmada; mesajdaki her karakter, başka ('anahtar' değeri kadar ötelenmiş) karakterle yer değiştirerek şifreli mesaj elde edilmektedir. Sezar yöntemi **monoalfabetik** şifrelemenin tipik bir örneğidir.
- Şifrelenecek metindeki harfler alfabede 3 harf kaydırılarak değiştirilir.



Tablo Yöntemi (Mono Alfabetik Şifreleme)

- Alfabedeki her harf başka bir harfle yer değiştirir ama bu bir kurala bağlı olmadan karışık bir şekilde yapılır.

Mono Alfabetik Şifreleme Zayıflığı

- Mono alfabetik şifreleme yöntemleri bilgisayar yardımıyla çok kısa sürede kırılabilir. Bu yöntemler kullanılan dildeki harflerin yerini değiştirir ama harflerin ***kullanım sıklığını (frekansını)*** değiştirmez.
- Örneğin Türkçe'de en çok kullanılan harf olan "a" harfi tablo yöntemi kullanılarak "c" harfi ile yer değiştirilirse elde edilecek şifreli metinde en çok tekrar eden harfin "c" olduğu görülür ve bunun "a" harfi olabileceği tahmin edilerek şifre çözülmeye başlanabilir.

Vigenere Yöntemi (Poli Alfabetik Şifreleme)

- Bu tip şifrelemede, mono alfabetik yöntemlerden farklı olarak bir harf değiştirilince her seferinde aynı harfe dönüşmez.
- Poli alfabetik şifreleme yöntemleri de bilgisayar yardımıyla ve ***frekans sayımı*** ile çok kolay ve çabuk çözülebilmektedir.
- Bu yöntemde oluşturulan tablo ve bir anahtar kelime kullanılarak şifreleme yapılır.
- Şifreleme İşlemi
  Açık Mesaj (sütun): BULUŞ MAYER İANKA RA
  Anahtar Kelime (satır): KALEM KALEM KALEM KALEM...
  Şifreli Mesaj: LUZAĞ ZAJIF UABÖM DA
  [(B + K) %26] + 65 = L

- Şifre Çözme İşlemi
  Şifreli Mesaj (tablo): LUZAĞ ZAJIF UABÖM DA
  Anahtar Kelime (satır): KALEM KALEM KALEM KALEM...
  Açık Mesaj (sütun): BULUŞ MAYER İANKA RA

Yalnızca Harflere Göre Şifreleme Yapıldığında

- **Şifreli Karakter = [{ (orijinal karakter - 65) + (anahtar karakteri - 65)} % (mod) 26] + 65**
- **Orijinal Karakter = [{ (şifreli karakter - anahtar karakteri) + 26 } % (mod) 26] + 65**

Bütün Karakterler Göre Şifreleme Yapıldığında

- **Şifreli Karakter = (orijinal karakter + anahtar karakteri) % (mod) 255**
- **Orijinal Karakter = [(şifreli karakter - anahtar karakteri) + 255] % (mod) 255**

Tek Kullanımlık Karakter Dizisi (One-time Pad)

- Bu basit şifreleme yönteminde rastgele üretilen bir karakter (harf veya rakam) dizisi kullanılarak şifreleme yapılır.
- Açık mesaj içinde yer alan her karakter, üretilen dizide karşısına denk gelen karakterle işleme sokularak şifreli mesaj elde edilir. Mesajı çözmek için rastgele dizinin bilinmesi gereklidir. Bu yönteme Vernam şifreleme yöntemi denir.
- Açık Mesaj: BULUŞMAYERIANKARA
  Rastgele Dizi : DEFRYPLCNMLJKHFGH
  Şifreli Mesaj: RLDYDOY …
- Bu yöntemin güvenliği rastgele üretilen diziye bağlıdır. Bu dizi gerçekten rastgele üretilmelidir, eğer bir kurala bağlı olarak üretilirse ve bu kural saldırgan tarafından bilinirse sistem kırılabilir.
  İlk olarak **"teletype"** makinelerinde kullanılmıştır.

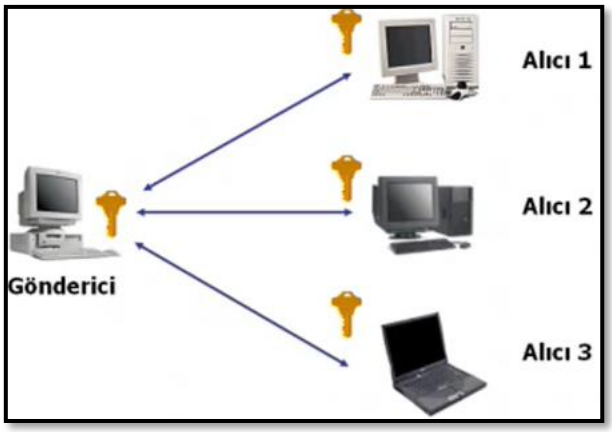
Güvenli Şifreleme Yöntemleri

1. Simetrik Kriptografi (gizli anahtarlı kriptografi sistemi) Nedir?

- Simetrik şifrelemede, şifreleme ve şifre açma işlemi aynı anahtar ile yapılır. Simetrik kriptografide bu anahtar gizli tutulmalıdır.
- Bu sistemi kullanarak haberleşen taraflar (gönderici-alıcı taraflar):
  - Aynı şifreleme algoritmasını kullanırlar.
  - Birbirine uyumlu gerçeklemeler kullanırlar.
  - Aynı anahtarı (yani yalnızca tek anahtar) kullanırlar.

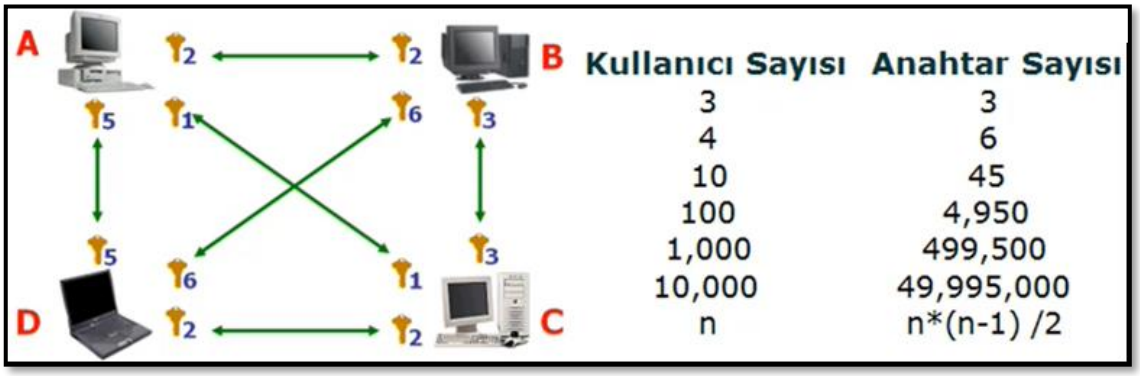
Birden – Çoğa (One to Many) Anahtar Yöntemi

Bu yöntemde haberleşen tüm taraflar aynı gizli anahtarı kullanırlar. Bu nedenle herkes birbirinin şifreli mesajlarını açabilir ve okuyabilir.



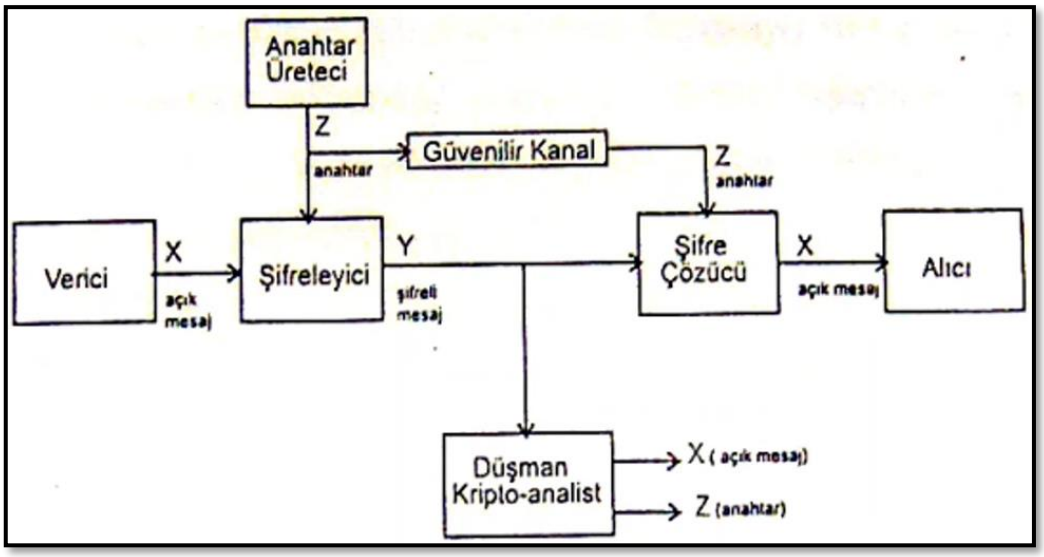
Çoktan – Çoğa (Many to Many) Anahtar Yöntemi

A-B arasında 2 numaralı anahtar, B-C arasında 3 numaralı anahtar … kullanılır.
Kullanıcı (monitör) sayısı ve anahtar sayısının kuralı aşağıda verilmiştir.



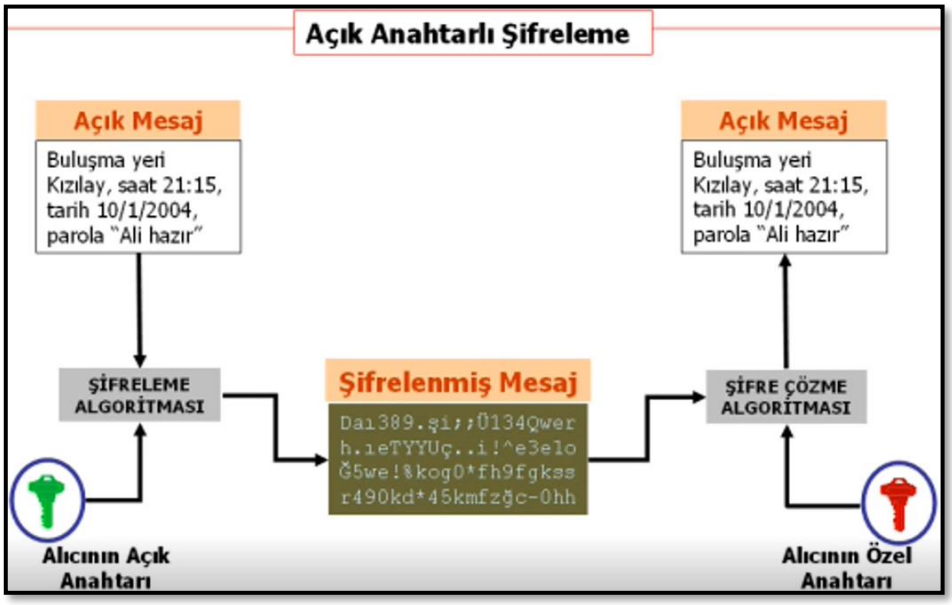
Kullanıcı Sayısı	Anahtar Sayısı
3	3
4	6
10	45
100	4,950
1,000	499,500
10,000	49,995,000
n	n*(n-1) /2

Simetrik Kriptografi Genel Görünüş



2. Asimetrik Kriptografi (Açık Anahtarlı Şifreleme) Nedir

- **Asimetrik kriptografi, simetrik kriptografiye göre daha güvenilirdir.**
- Asimetrik kriptografide, şifreleme ve şifre çözme işlemi **farklı anahtarlar** ile yapılır. Bu anahtar çiftini oluşturan anahtarlara **açık ve özel anahtar** adı verilir.
- Bu kriptografi yönteminde özel anahtar gizli tutulmalıdır fakat açık anahtar gerekli kişilere verilebilir ve başka kişilerle paylaşılabilir.
- Bu sistemi kullanarak haberleşen taraflar (gönderici-alıcı taraflar):
  - **Aynı şifreleme algoritmasını kullanırlar**.
  - **Gerekli anahtarlara erişebilirler**.
- Aşağıdaki fotoğraf incelendiğinde; gönderilecek olan veri, alıcının açık anahtarıyla şifrelenir.



- **Asimetrik kriptografi için anahtar yönetimi daha kolaydır** çünkü bir kullanıcıyla şifreli haberleşmek isteyen kişi karşı tarafın açık anahtarına ihtiyaç duyar. Bu açık anahtar kamuya açık olarak yayınlandığı için sisteme giren bir kişi için sadece bir anahtar çifti üretmek yeterli olmaktadır.

Kullanıcı Sayısı	Anahtar Çifti Sayısı
3	3
10	10
1000	1000
n	n

3. Asimetrik Kriptografik Yöntemler

Asimetrik kriptografi algoritmaları, simetrik algoritmalardan farklı olarak çözülmesi zor olan matematiksel problemlere dayanır.

Başlıca asimetrik kriptografi algoritmaları:

- RSA
- Eliptik Eğri Sistemleri
- El Gamal
- Diffie - Hellman

a. RSA Algoritması

- Açık anahtar kriptografik sistemi ve sayısal imzalama yöntemi olarak kullanılır.
- **Çarpanlarına ayırma problemi üzerine inşa edilmiştir**.
- Bileşik tam sayı olan N'i oluşturan, asal sayılar p ve q bulunur, öyleki N = P * Q 'dir. **Asal sayılar yani P ve Q büyüdükçe şifrenin kırılması zorlaşır**.
- Çok **güvenlidir** fakat fazla **hızlı değildir**.
- Anahtar oluşturma algoritması şu şekildedir:
  1. P ve Q gibi çok büyük iki asal sayı seçilir.
  2. Bu iki sayının çarpımı N = P*Q ve bu sayıların bir eksiklerinin çarpımı φ(N) = (P-1)*(Q-1) hesaplanır.
  3. 1'den büyük φ(N)'den küçük ve φ(N) ile aralarında asal bir E tamsayısı seçilir.
  4. Seçilen E tamsayısının mod φ(N)'de tersi alınır, sonuç D gibi bir tamsayıdır. [E*D mod φ(N)=1 olacak şekilde D tamsayısı belirlenir]
  5. **E** ve **N** tamsayıları **genel anahtarı**, **D** ve **N** tamsayıları ise **özel anahtarı** oluşturur.
- Şifrelerne ve deşifreleme ise şu şekildedir.

  Herhangi bir M meajı için şifreleme C = M^E (mod N)

  C şifreli metni için deşifreleme M = C^D (mod N)
- Algoritmanın kullanımı

  Ayşe, Bora'ya m mesajını şifreli göndermek için:

  m'nin e'inci üssünü alır, yani m'yi Bora'nın açık anahtarı ile şifreler:

  c = m^e (mod n)

  c'yi (şifreli mesaj) Bora'ya gönderir

  Bora c sayısının d'nci üssünü alır, yani c'nin şifresini kendi özel anahtarını kullanarak çözer:

  m = c^d (mod n)

Örnek:

- P=7 ve Q=17 gibi çok büyük iki asal sayı seçilsin.
- Bu iki asal sayının çarpımı N = P.Q=7*17;**N=119** ve bu sayıların bir eksiklerinin çarpımı φ(N)=(P-1)(Q-1)=6*16; **φ(N)=96** olarak hesaplanır.
- 1'den büyük φ(N)**(96)**'den küçük ve 96 ile aralarında asal bir **E=5** tamsayısı seçilsin.
- **5 * D mod 96 = 1** olacak şekilde **D** tamsayısı belirlenir (**D=77**).
- **E=5** ve **N=119** tamsayıları genel anahtarı, **D=77** ve **N=119** tamsayıları ise özel anahtarı oluşturur.

(5,119) anahtarları ile şifreleme, (77,119) anahtarı ile deşifreleme yapılacaktır. M açık metni 19 olarak seçilsin.

- $C=M^E \pmod N \rightarrow C=19^5 \pmod{119} \rightarrow C=66$
- $M= C^D \pmod N \rightarrow M= 66^{77} \pmod{119} \rightarrow M=19$

4. Kripto Sistemlerini Karşılaştırılması

Simetrik kriptografinin kuvvetli yönleri:

- Algoritmalar **hızlıdır**.
- Algoritmaların donanımla gerçeklenmesi kolaydır.
- "Gizlilik" güvenlik hizmetini yerine getirir.

Simetrik kriptografinin zayıf yönleri:

- Ölçeklenebilir değil.
- Emniyetli anahtar dağıtımı zor.
- "Bütünlük" ve "Kimlik Doğrulama" güvenlik hizmetlerini gerçeklemek zor.

Asimetrik kriptografinin kuvvetli yönleri:

- Anahtar yönetimi ölçeklenebilir.
- Kripto-analize karşı dirençli (kırılması zor).
- Bütünlük, kimlik doğrulama ve inkâr edememezlik güvenlik hizmetleri sağlanabilir.

Asimetrik kriptografinin zayıf yönleri :

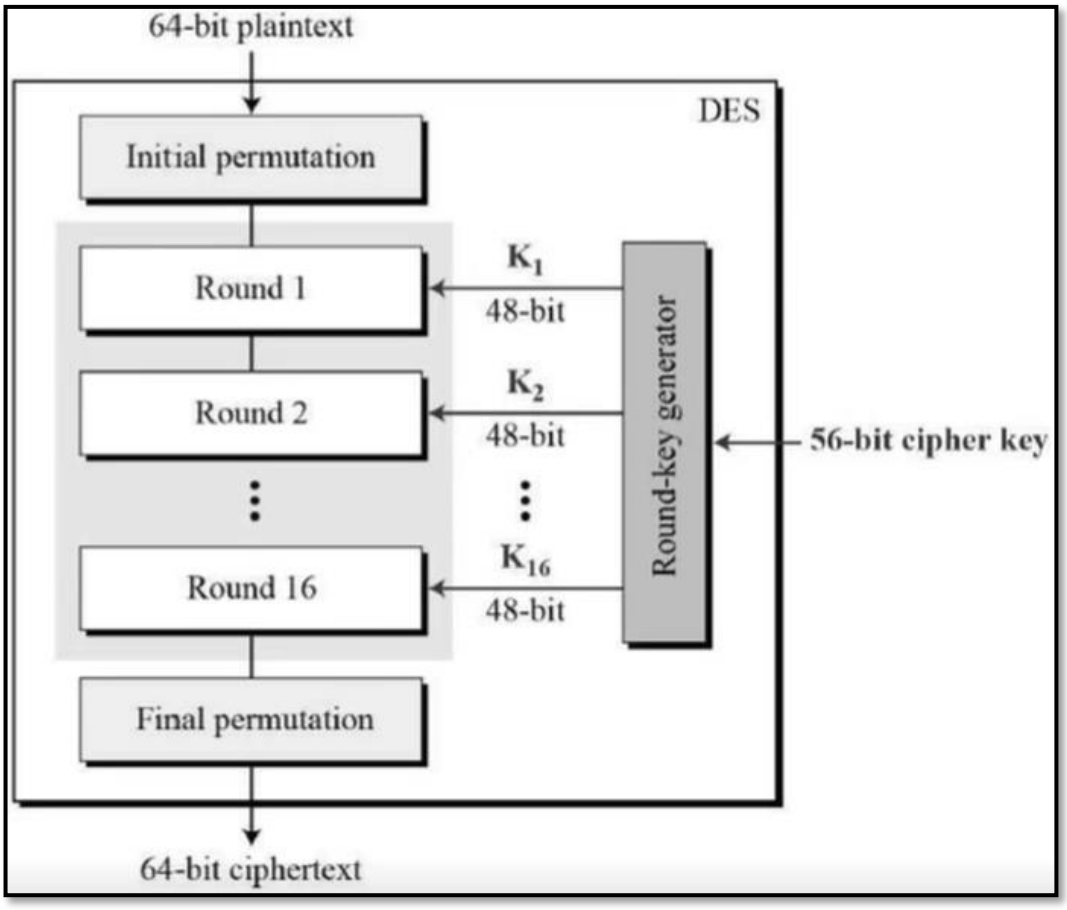
- Simetrik krıptografiye göre daha yavaştırlar.
- Anahtar uzunluğu bazı durumlar için kullanışlı değildir.

Konu	Simetrik Kriptografi	Asimetrik Kriptografi
Gizlilik	Sağlar	Sağlar
Bütünlük	--	Sağlar
Kimlik doğrulama	--	Sağlar
İnkar Edememezlik	--	Sağlar
Performans	Hızlı	Yavaş
Güvenlik	Anahtar uzunluğuna bağlı	Anahtar uzunluğuna bağlı

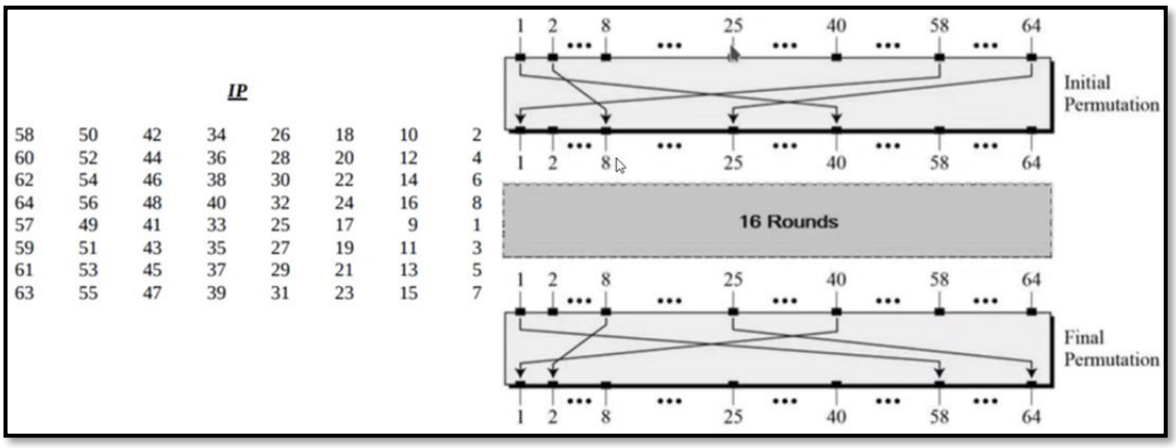
DES (Data Encryption Standart) Algoritması

- DES **ilk simetrik şifreleme** algoritmasıdır.
- **Anahtar uzunluğunun kısa olması nedeniyle yeterli güvenliği sağlayamadığı** görülmüştür.
- Veriler bloklar halinde işlenir.
- Anahtar gizlidir ve hızlı çalışırlar.
- **AES**, **DES**, **3DES** algoritmaları simetrik şifreleme algoritması örneklerindendir.
- Algoritmadaki matematiksel işlemler, bitler (0 ve 1) üzerinden gerçekleştirilir.

- DES algoritmasının **blok uzunluğu 64 bit, *anahtar uzunluğu ise 56 bittir.***
  280 bitlik bir karakterimizin olduğunu varsayalaım. 64+64+64+64 = 256
  280-256 = 24 bit
  DES algoritması 64 bitle çalışıyor ve bizim elimizde 24 bit uzunluğunda karakterimiz kaldı dolayısıyla 24 bitin önünde 40 bit uzunluğunda 0 koyarız.
- 500 bitlik **şifrelenecek metnimizin (plaintext)** olduğunu varsayalım. Bu metin 64 bitlik bloklara ayrılır, her blok birbirinden bağımsız olarak şifrelenir ve yine 64 bit uzunluğunda **şifrelenmiş metinler (cipher text)** elde edilir.
- DES algoritmasında **döngü (round) yapısı** vardır ve algoritma çalışırken bu döngü 16 kez kullanılır. Yani 64 bitlik bir verinin şifrelenmesi için 16 kez çalışması lazım. Bunun yapılmasının sebebi her seferinde daha da karmaşık hale getirmeye çalışmaktır.
- Algoritma **initial permutation (başlangıç permütasyonu)**, **döngü işlemi** ve **inverse initial permutation (ters permütasyon)** aşamalarından oluşur.
- Aşağıdaki resim için konuşacak olursak; 64 bitlik bir veri girişi var (başlangıç permütasyonu) ve 16 kez döngüye giriyor.
  56 bit 48 bite çevriliyor ve 64 bitlik plain textimiz 48 bitle işleniyor en son 64 bit olarak çıkıyor (final permütasyon).



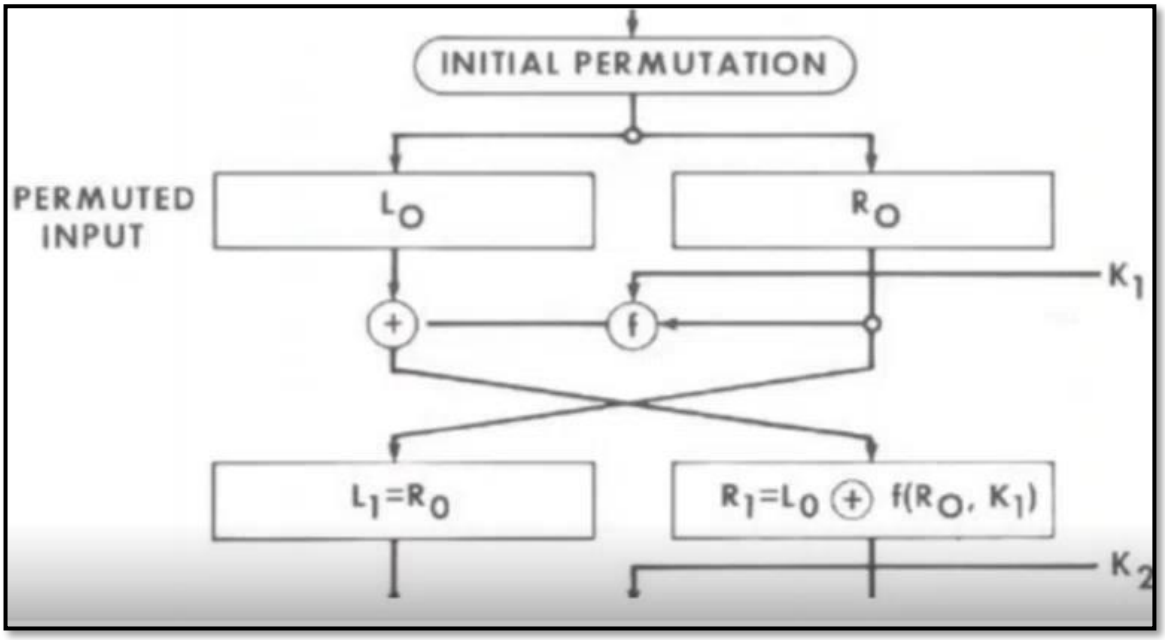
- **Başlangıç permütasyonu, DES algoritmasının ilk aşamasıdır.** Dışarıdan gelen 64 bitlik açık metindeki bitler initial (başlangıç) permütasyon tablosundaki koordinatlara göre yer değiştirirler.
  Aşağıdaki resim için konuşacak olursak 58. bitimiz, 1. bitin yerine geçiyor. 50. bitimiz 2. bitin yerine geçiyor. Böylelikle karıştırma işlemi gerçekleştiriliyor.



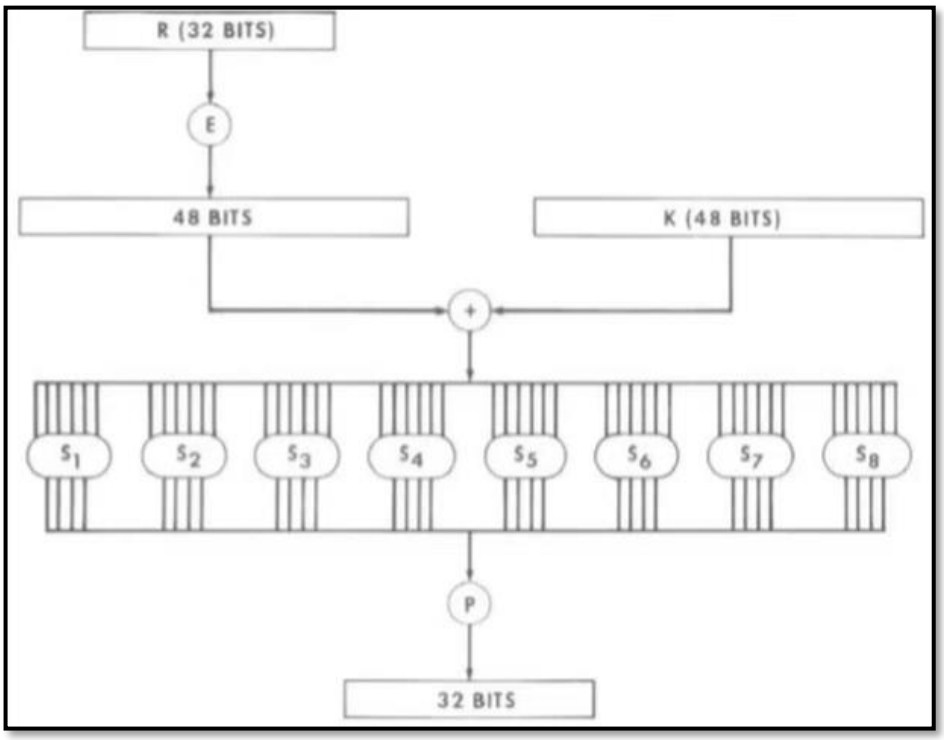
IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

- DES döngüsü **iki ana işlemden** oluşur. Bunlar **F fonksiyonu** ve **XOR** işlemleridir.
- Başlangıç permütasyonundan gelen veri **sağ ve sol şeklinde 32 bitlik iki parçaya** ayrılır.



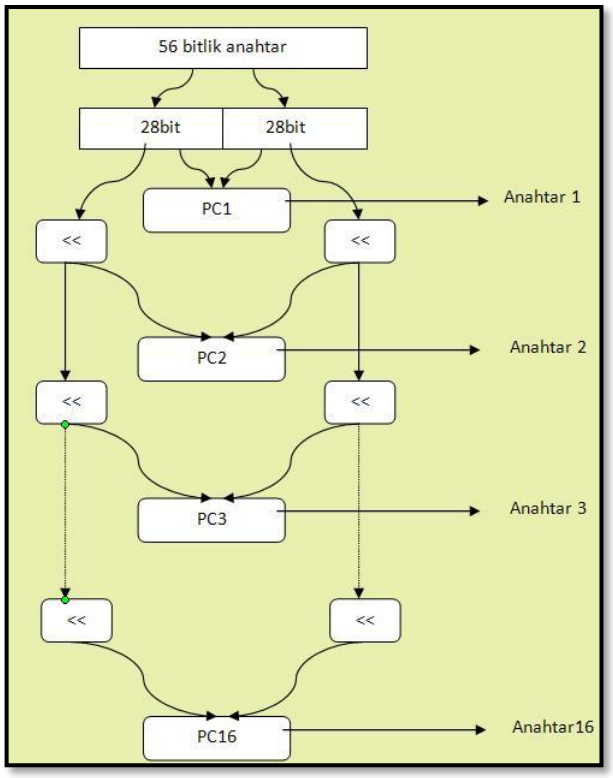
- F fonksiyonuna giren sağ 32 bitlik veriye ilk olarak **expension (genişletme) işlemi** uygulanır. Bu işlemde 32 bitlik verinin bazı bitleri tekrarlanarak 48 bitlik veri elde edilir.
- Anahtar üretici tarafından, 56 bitlik anahtar girişinden her döngü adımı için ayrı olarak kullanılacak 16 adet 48 bitlik anahtar üretilir.



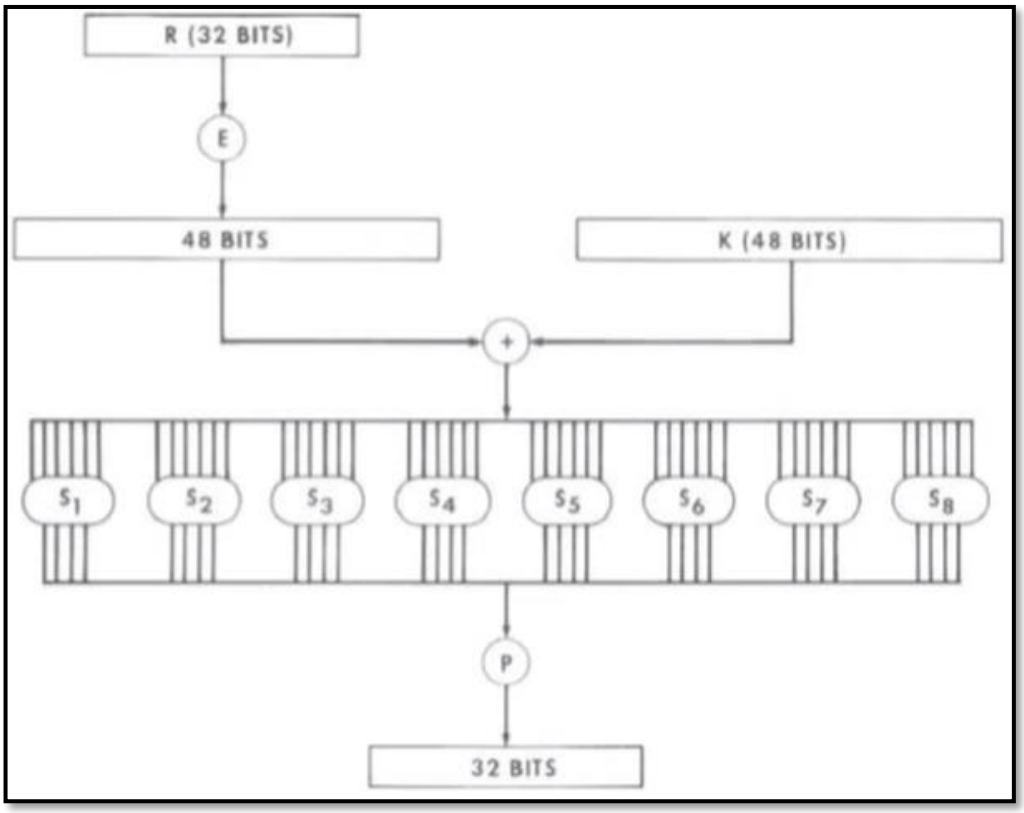
Her Adım için Farklı Bir Anahtar Üretme

Adım-1: İlk 28 bitlik bloğun en solundaki 4 bit ve ikinci 28 bitin en sağındaki 4 bit çıkarılarak ortada kalan 48 bit ile anahtar üretilir.

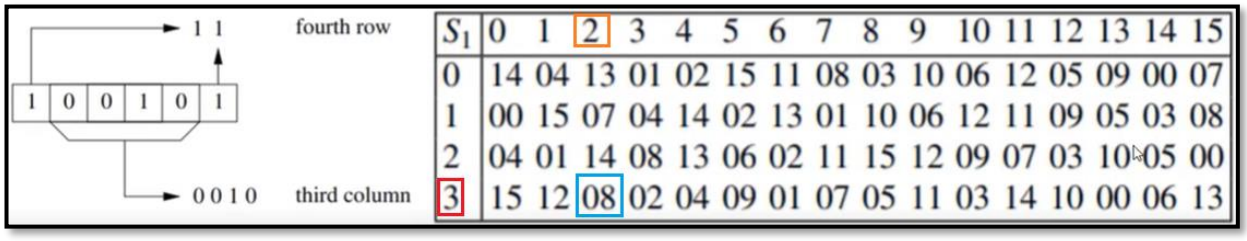
Adım-2: Her iki bloğun bitleri kendi içinde sola kaydırılır (shift). En soldaki bit bloğun en anlamsız biti olarak başa gelir. Yukarıdaki iki adım 16 kez tekrar edilerek 16 farklı anahtar üretilmiş olur.



- Genişletme işleminden gelen veri ve anahtara **XOR** işlemi uygulanır.
- Çıkan veri S kutularına girer.



- F fonksiyonun içinde 8 adet S kutusu bulunur. Bu kutuların girişi 6 bit, çıkışı ise 4 bittir (1111). Yani aşağıdaki S_1 kutusuna bakacak olursak [0-15] arasındaki sayılardan oluşmaktadır bunun sebebi 4 bitin hepsi 1111 olursa maximum 15 eğer ki 0000 olursa 0 edeceğindendir.
- Yani toplamda 48 bit girecek (8x6), 32 bit (8x4) çıkacak.
- Her S kutusu 4 satır 16 sütundan oluşur ve 64 adet 4 bitlik sayıyı barındırır.
- Örnek : $S_1(37 = 100101) = 8 = 1000$
- 37 sayısının ikilik tabandaki karşılığı 100101. İlk bit ve son bit yan yana getirildiğinde 11'i elde ederiz dolayısıyla 3'ü. **3** bize satır numarasını verir.
- 1 … 1 arasında kalan 0010 ise sütun numarasını verir bize. 0010 sayısıda **2'ye** tekabül etmektedir.
- Yani 3. satır, 2. sütundaki sayıyı elde ederiz yani **8'i.**

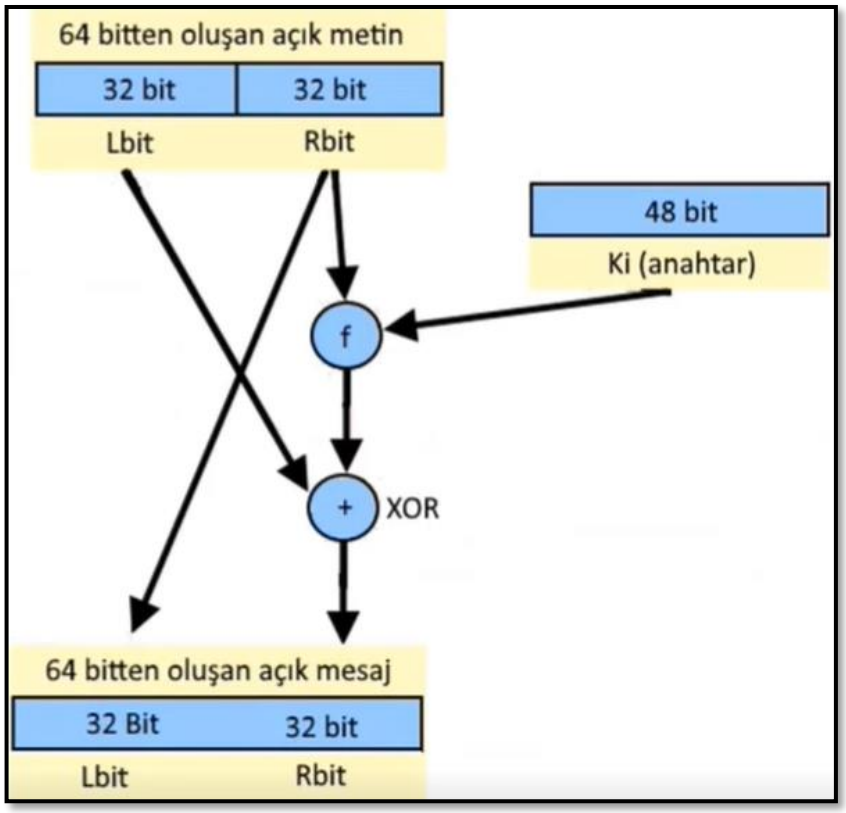


S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

- Sekiz adet S kutusunun çıkışından toplam 32 bitlik veri elde edilir.
- S kutusu expansion işleminin tersi olarak çalışır ve 48 bitlik veriyi tekrar 32 bite indirir.
- Elde edilen veriye permütasyon işlemi uygulanır.

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

- F fonksiyonundan çıkan 32 bitlik veri, sol 32 bit ile XOR işlemine girer.
- Döngü işlemi arka arkaya 16 kez tekrarlanır.

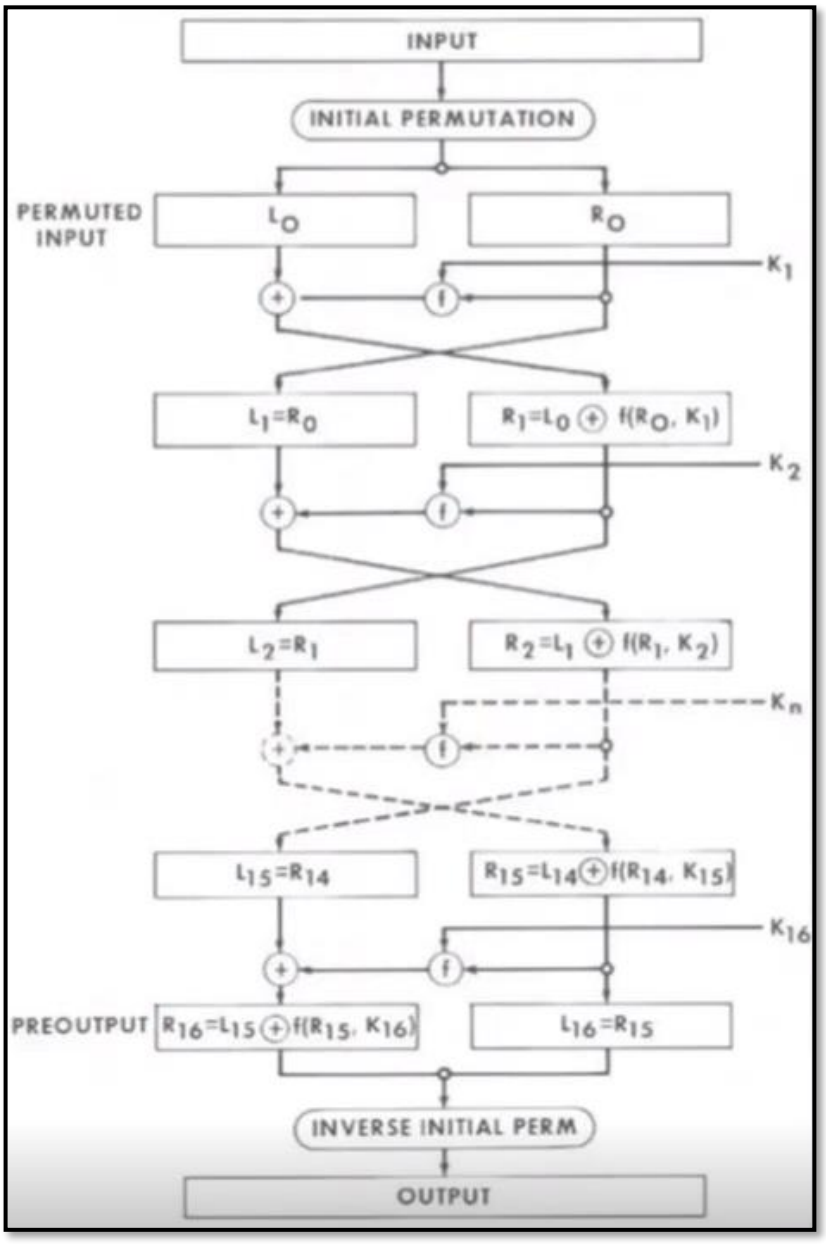


- Bu aşama DES algoritmasının son adımıdır. Başlangıç permütasyonundaki işlem tersten uygulanır ve 64 bitlik şifrelenmiş metin (cipher text) çıkışa verilir.

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

- DES algoritmasının daha zor saldırılır hale gelmesi için 128 bit anahtar uzunluğu kullanan üçlü DES uygulaması geliştirilmiştir.


INPUT
INITIAL PERMUTATION
PERMUTED INPUT
L_0
R_0
K_1
f
$L_1=R_0$
$R_1=L_0 \oplus f(R_0, K_1)$
K_2
$L_2=R_1$
$R_2=L_1 \oplus f(R_1, K_2)$
K_n
$L_{15}=R_{14}$
$R_{15}=L_{14} \oplus f(R_{14}, K_{15})$
K_{16}
PREOUTPUT
$R_{16}=L_{15} \oplus f(R_{15}, K_{16})$
$L_{16}=R_{15}$
INVERSE INITIAL PERM
OUTPUT

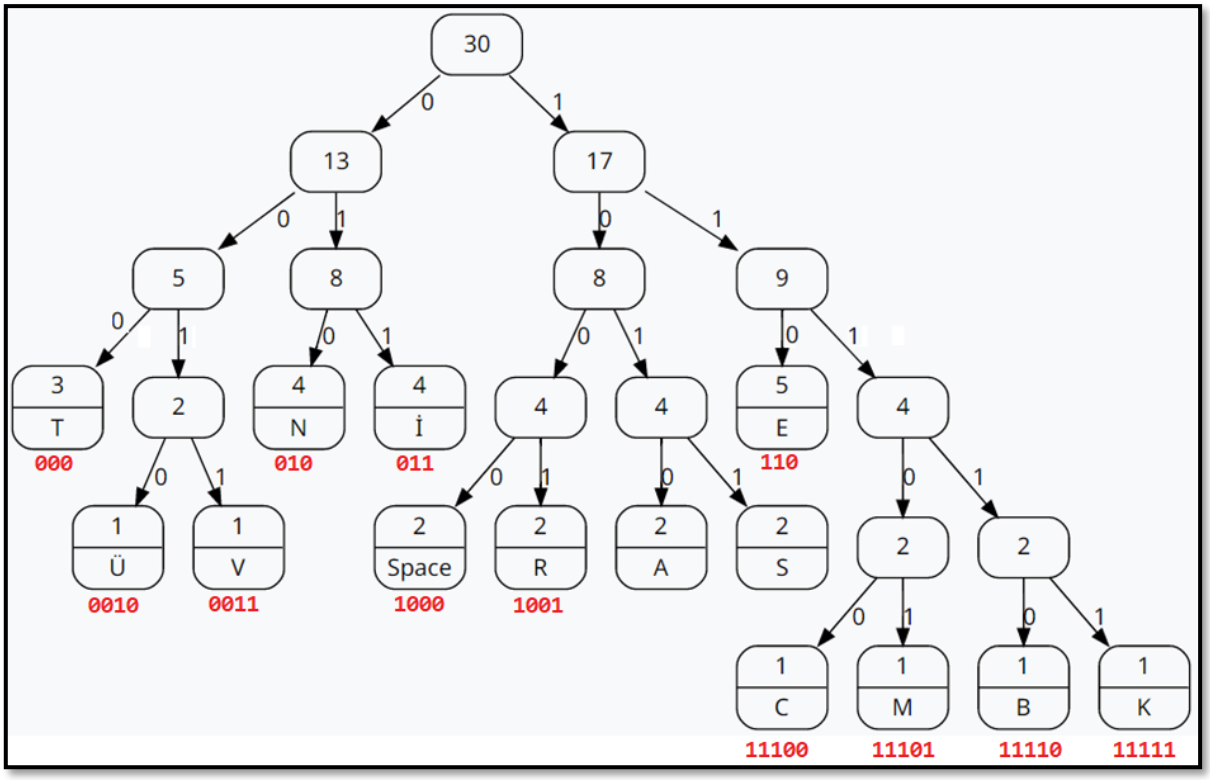
Veri Sıkıştırma

- Veri sıkıştırma, veriyi saklarken disk alanından tasarruf etmemizi sağlar.
- Veriyi iletirken zamandan tasarruf etmemizi sağlar.
- Veri binary şeklinde tutulur.
- Sıkıştırma oranı **%50-75** ise iyi bir oran anlamına gelmektedir.
- Sıkıştırma demek saklanan verinin formatını değiştirmek denilebilir.
- A-T-G-C olan insanın gen karakterlerini sıkıştıracağımızı varsayalım.
  Eğer ASCII tablosunda saklayacak olursak A harfi 41'e, G harfi 47'ye .. tekabül etmektedir.
  Binary formatında saklaycak olursak 00 A – 01 G .. tekabül etmektedir.
  Burada dikkat etmemiz gereken şey A'yı ASCCII tablosunda 64 bitlik formatta ifade ederken, binary formatında ise 16 bit olarak ifade etmekteyiz. Arada 4 kat fark vardır bu da alandan tasarruf etmemizi sağlar.
- Sıkıştırma algoritmaları veriyi her zaman sıkıştırmaz bazen genişletebilir.
- wwwwaaadexxxxxx stringini sıkıştırmak için hangi karakterden kaç tane olduğunu yazmamız gerekmektedir.
  w4a3d1e1x6

Huffman Algoritması

- En çok kullanılan karakterler daha az bitle temsil edilir. Çünkü verinin hafızdaki boyutunun küçük olması amaçlanıyor.
  Az kullanılan karakterler daha fazla bitle temsil edilir.
- Karmaşıklığı **n.logn**
- Algoritma oluşturulurken önce yapraklar oluşuyor. Oluşum yapraklardan köke doğru devam ediyor.
- Mesela **"NECMETTİN ERBAKAN ÜNİVERSİTESİ"**
  Öncelikle frekans yani harflerin tekrarlanma sıklığı hesaplanır.

Karakter	A	B	C	E	İ	K	M	N	R	S	T	Ü	V	Space	TOPLAM
Frekans	2	1	1	5	4	1	1	4	2	2	3	1	1	2	30
Bit Sayısı	4 bit	5 bit	5 bit	2 bit	3 bit	5 bit	5 bit	3 bit	4 bit	4 bit	4 bit	5 bit	5 bit	4 bit	58 bit
Toplam Bit Sayısı	2*4=8 bit	1*5=5 bit	1*5=5 bit	5*2=10 bit	4*3=12 bit	1*5=5 bit	1*5=5 bit	4*3=12 bit	2*4=8 bit	2*4=8 bit	3*4=12 bit	1*5=5 bit	2*4=8 bit	2*4=8 bit	108 bit



Ağaç Yapısı Nasıl Oluşturulur ?

1. **Her sembol için bir yaprak düğüm ekle.**
2. **Sırada, birden fazla düğüm kaldığı sürece döngüler halinde:**
   - **En az sıklıkla kullanılan iki düğüm alınır.**
   - **Yani bir iç düğüm oluşturulur ve değer olarak iki düğümün toplamı alınır.**
   - **Yeni düğüm ağaca eklenir.**
3. **Döngü bitip tek düğüm kaldıysa o düğüm kök düğüm yapılır.**

En düşük frekans değerine sahip olan verilerden başlanır yapraklar oluşturulmaya.
Yaprak düğümler sayesinde oluşturulan kök düğümler her zaman minimum değere sahip olmayı hedefler bu yüzden yapraklar oluşturulurken en küçük frekans değerlikli veriler öncelikli kullanılır.
Ağacın her sol edge'i 0, her sağ edge'i ise 1 olarak ifade edilir.
Her bir düğümün bit değerini kökten başlayarak bulunulan düğüme kadar hesaplarız.

LZW Coding

- Bilgisayar bilimlerinde kullanılan **kayıpsız sıkıştırma** (lossless compression) algoritmalarından birisidir.
- Algoritma, sıkıştırılacak metin içerisinde harf harf ilerleyerek, mümkün olan en fazla harfi içeren kelimeyi sözlüğe eklemeye çalışmakta ve bu sırada da sözlükteki karşılığı ile metni değiştirmektedir. Böylelikle sıkıştırma işlemi gerçekleşmiş olur. Örneğin 4 harf uzunluğunda bir kelimeyi sözlüğe eklemeyi başardıysak, karşı tarafa gönderilen mesajda tek bir sembol bu dört harflik mesajı içerecektir.
- Yollanan mesaj içerisinde harflerden oluşan bir sözlük bulunmakla birlikte çok harfli kelimeleri içeren sözlük dinamik olarak oluşturulur ve açan taraf da bu sözlüğü yine dinamik bir şekilde oluşturarak mesajı açar.
- Karmaşıklığı **O(n) 'dir.**

Karakter	s+c	Var mı?	String Mevcut	Output	Sözlük	String Son
b	b	Var	" "			b
a	ba	Yok	b	1	ba/5	a
n	an	Yok	a	0	an/6	n
a	na	Yok	n	3	na/7	a
n	an	Var	a			an
a	ana	Yok	an	6	ana/8	a
_	a_	Yok	a	0	a_/9	_
b	_b	Yok	_	4	_b/10	b
a	ba	Var	b			ba
n	ban	Yok	ba	5	ban/11	n
d	nd	Yok	n	3	nd/12	d
a	da	Yok	d	2	da/13	a
n	an	Var	a			an
a	ana	Var	an			ana
				8		

Pseudo Code

```
string s = "";
char c;
while (okunacak karakter oldukça){
   if(s+c sözlükte varsa){
      s = s + c;
   }
   else{
      s'nin sözlükteki index'i output edilir
      s+c ifadesi sözlüğe eklenir
      s=c
   }
   output s sözlük index'i
}
```

```
banana_bandana (14 BYTE)
1 0 3 6 0 4 5 3 2 8 (10 BYTE)
```

Sözlük	
a	0
b	1
d	2
n	3
_	4

berber_berbere

Sözlük	İndeks
b	0
e	1
r	2
_	3

berber_berbere : 14 bayt
012423771 : 9 bayt

%35'lik sıkıştırma sağlanmıştır.

Karakter	Karakter + Harf Grubu	Sözlükte Var/Yok	Çıktı	Sözlüğe Eklenen/ İndeks	Mevcut Harf Grubu
b	b	Var	-	-	b
e	be	Yok	0	be/4	e
r	er	Yok	1	er/5	r
b	rb	Yok	2	rb/6	b
e	be	Var	-	-	be
r	ber	Yok	4	ber/7	r
_	r_	Yok	2	r_/8	_
b	_b	Yok	3	_b/9	b
e	be	Var	-	-	be
r	ber	Var	-	-	ber
b	berb	Yok	7	berb/10	b
e	be	Var	-	-	be
r	ber	Var	-	-	ber
e	bere	Yok	7	bere/11	e
-	-	-	1	-	-

012423771

Sözlük	İndeks
b	0
e	1
r	2
_	3

Gelen İndeks	Gelen + Mevcut İndeks	Sözlükte Var/Yok	Çıktı	Sözlüğe Eklenen/İndeks	Mevcut İndeks
0	0 = b	Var	-	-	0
1	01 = be	Yok	b	be/4	1
2	12 = er	Yok	e	er/5	2
4	24 = rbe	Yok	r	rbe/6	4
2	42 = ber	Yok	be	ber/7	2
3	23 = r_	Yok	t	r_/8	3
7	37 = _ber	Yok	_	_ber/9	7
7	77 = berber	Yok	ber	berber/10	7
1	71 = bere	Yok	ber	bere/11	1
			e		

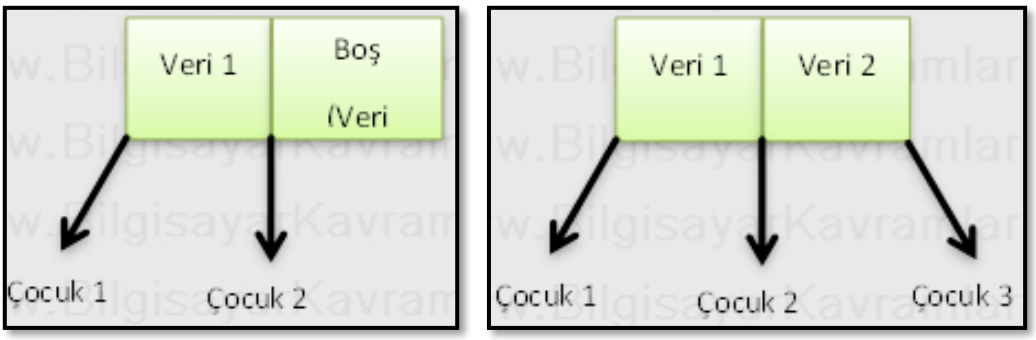
2-3 Ağaçları (https://www.youtube.com/watch?v=jtDjhk1HZFU)

- 2-3 ağacının amacı sürekli olarak dengeli (balanced) tutmaktır.
  Ağaçta her yaprak aynı seviyededir.
- BST'ye göre daha etkili arama ve daha düşük update maliyetlerine sahiptir.
- Arama $\log_3$ tabanında gerçekleştirilir. 3 tabanında gösterilmesinin sebebi 3 tane çocuğun olmasıdır.
- 2-3 Tree için; Binary Tree için;
  $2 \leq$ çocuk sayısı < 4 --> 2, 3 $x \leq$ çocuk sayısı $< 2x$
  $1 \leq$ anahtar sayısı < 3 --> 1, 2 $x-1 \leq$ anahtar sayısı $< 2x-1$

Silme Koşulları

1- Anahtarın silinmesi, bir düğümün tutması gereken minimum anahtar sayısını ihlal etmez.
2- Anahtar silinmesi, bir düğümün tutması gereken minimum anahtar sayısını ihlal eder. İlk önce sol kardeş düğüme bakılır ve minimum sayıdan fazla anahtara sahipse bu düğümden bir anahtar alınır. Aksi takdirde hem sağ kardeş düğüme bakılır. Sağ kardeş düğümde minimum sayıdan fazla düğüm varsa sağ kardeşten alınır.
3- Silme işlemlerinde düğümlere ait çocuk sayılarında denge bozulmuyorsa sadece kurala göre döner.(Soldaki en büyük veya sağdaki en küçük düğüm alınarak.)
4- Sol veya sağa göre işlem yapıldığında öncelik çocuk sayısı fazla olandadır.

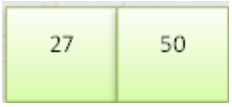
Düğümlerin isimlendirilmesi aşağıdaki şekilde yapılır:
2 düğümleri (2 nodes) : 2 adet çocuğu ve bir veri elemanı bulunan düğüm yapısıdır.
3 düğümleri (3 nodes) : 3 adet çocuğu ve iki veri elemanı bulunan düğüm yapısıdır.



2-3 Ağacına Ekleme İşlemi

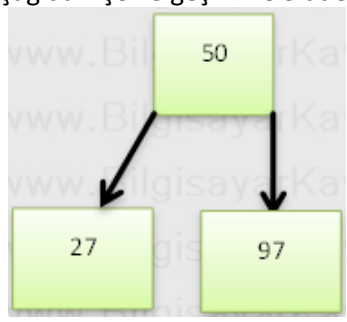
- Ağacımıza ekleyeceğimiz sayılar: 50, 27, 97, 52, 19, 11, 111
  Bu sırayla verilen değerleri ağaca ekleyelim:

İlk sayı olan 50'nin eklenmesi sonucunda ağaçta tek bir düğüm ve bu düğümde tek bir veri bulunuyor. Bu tip düğüme **2 düğümü** ismi verilmektedir ve iki çocuğu da boştur (null).

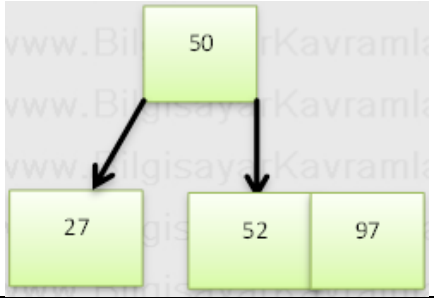


Düğümlerin içerisinde sıra bulunması gerektiği için, 27 değeri, 50 değerinin soluna yerleştirilmiştir. Bu kural bundan sonraki ekleme işlemlerinde de geçerli olacaktır. Yani düğüm içerisindeki değerler kendi aralarında sıralı olacak ve herhangi bir değerin solundaki çocuklarının değeri, kendi değerinden küçük ve sağındaki değeri kendi değerinden büyük olacaktır.
Ardından gelen ekleme değeri, 97'dir. Bu değer düğüme eklenince, aslında aşağıdaki gibi bir yapı elde edilir:

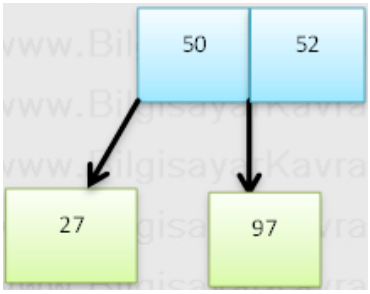
Ancak ne yazık ki 2-3 ağaçlarında, 3 adet değeri tek bir düğümde bulundurma şansımız yoktur. Bu yüzden yukarıdaki bu hayali yapı bölünerek aşağıdaki şekle geçilir ve 3 adet 2 tipinde düğümümüz olur:



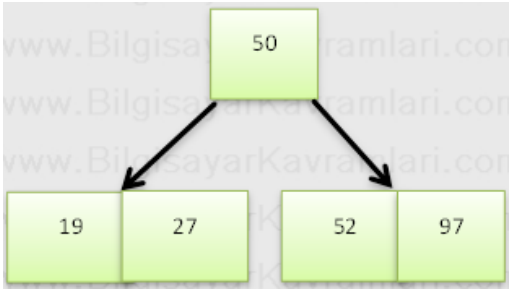
Yeni gelen değerimiz 52'dir:



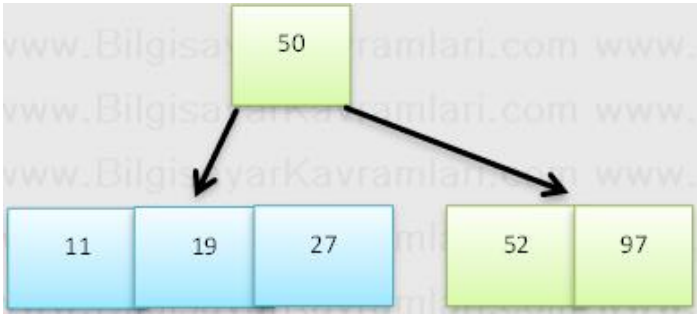
Yukarıdaki yeni şekilde, gelen 52 değeri, 50'den büyük olduğu için bu düğümün sağına sonra da 97'den küçük olduğu için bu düğümün soluna yerleştirilir.
52 değerinin, 50 değerinin sağına yerleştirilmemesinin sebebi, bu durumda ağaçta iki veri ve iki çocuk bulunan aşağıdaki hayali ağacın oluşmasıdır:



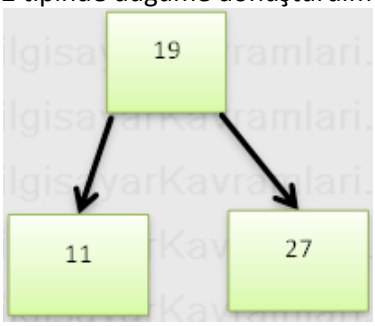
Yukarıdaki bu hayali ağaç ne yazık ki 2-3 ağaçlarında yer almaz. Tekrar hatırlatacak olursak 2-3 ağaçlarında **2 düğümlerinin 2 çocuk ve bir verisi**, **3 düğümlerinin ise 3 çocuk ve 2 verisi bulunur**. Yukarıdaki kök düğüm ise 2 veri ve 2 çocuk bulundurmaktadır. Bu durum kabul edilemez.
Yeni eklenen değerimiz 19:



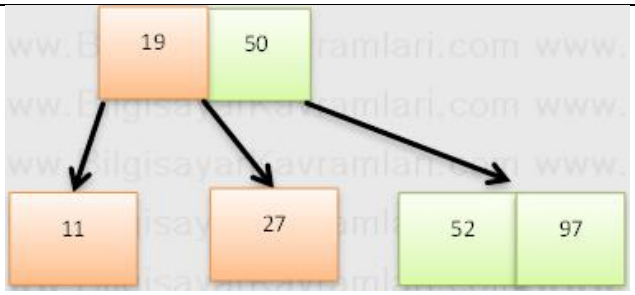
Yukarıdaki bu yeni ağaç yapısı, bir önceki adımda açıklandığı gibi, 50'nin yanına yeni bir veri eklenememesinden dolayı, 50'nin solundaki düğüme eklenmesi sonucunda elde edilmiştir.
Sıradaki eklenecek olan veri 11'dir. Öncelikle verinin eklenmek isteneceği yer, 50 değerindeki kök düğümünün soludur.



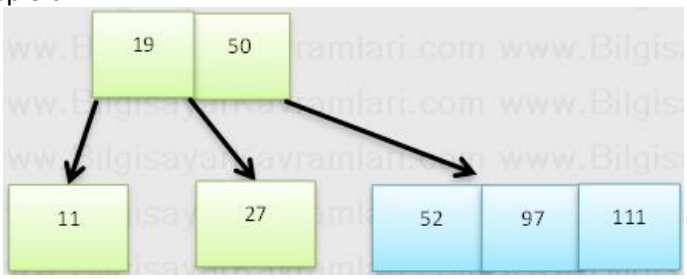
Yukarıdaki hayali ağaç, ne yazık ki 2-3 ağaçları tarafından kabul edilemez. Dolayısıyla üç veri içeren bu düğümü tek başına ele aldığımızda (ki aynı durum, örneğimizde 97 eklenirken de yaşanmış ve 3 adet veri yan yana elde edilmesin diye 3 adet 2 tipinde düğüme dönüştürülmüştü) bölünme işlemi gerçekleştirilir:



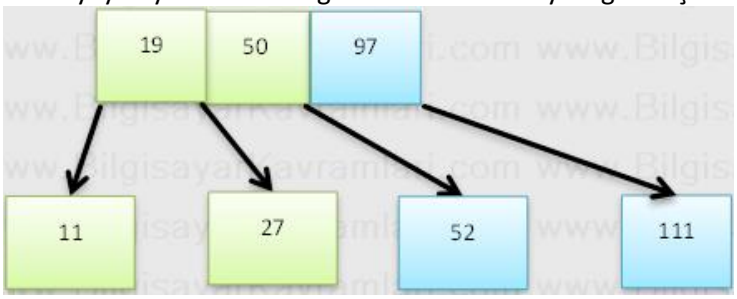
Yukarıdaki bu alt ağacı, bir önceki ağacımız ile birleştirirsek:



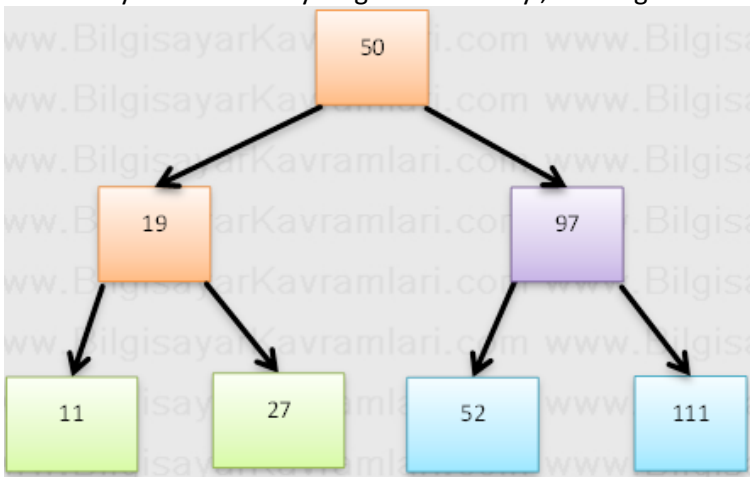
Elde ettiğimiz yeni ağaç, yukarıdaki şekilde iki ağacın birleştirilmiş hali olur.
Son olarak ekleyeceğimiz değer 111 olsun. Bu değer eklendiğinde ağaçta bir zincirleme reaksiyon oluşur ve birden fazla düğümün yukarıya doğru bölünmesi gerekir. Öncelikle 111 değeri 50'den büyük olduğu için, ağaçta 50'nin sağındaki gösterici (pointer) tarafından gösterilen düğüme yönlendirilir ve aşağıdaki hayali ağacın oluşmasına sebep olur:



Elbette yukarıdaki bu 3 veriyi yan yana tutan düğümü kabul edemeyeceğimiz için bu düğümü bölüyoruz:



Bu sefer kök düğümdeki 3 veriyi kabul edemeyeceğimizden dolayı, kök düğümü de bölüyoruz:

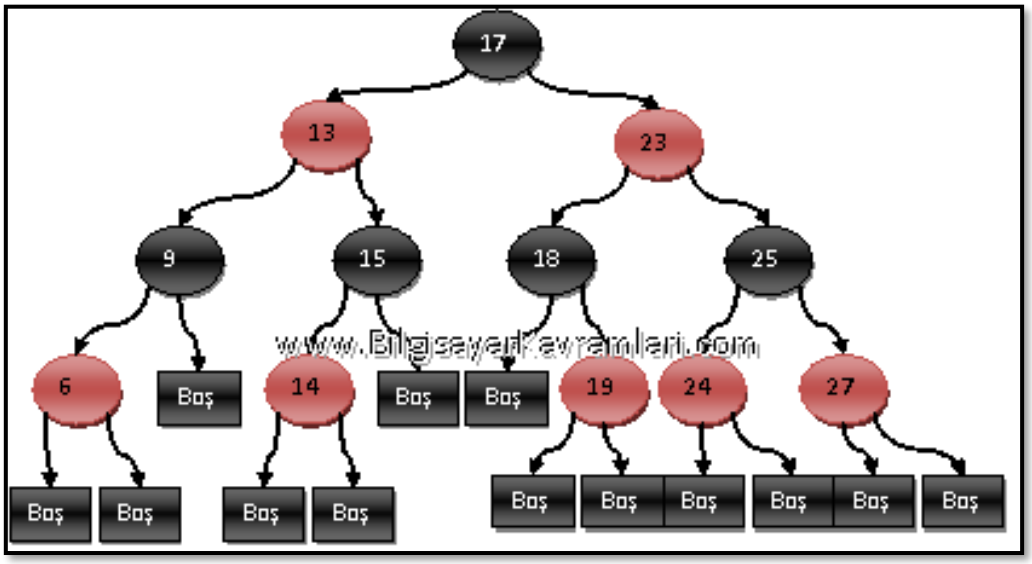


Son bölünmeden sonra toplam 7 adet 2 tipinde düğüm elde edilmiştir. Ayrıca yukarıdaki ekleme işlemleri boyunca her adıma bakıldığında ağacın dengeli bir ağaç olduğu (bütün yaprak düğümlerinin (leaf nodes) aynı seviyede olduğu) görülebilir.
Aslında yukarıdaki isimlendirmeler, **B-Ağacının (B-Tree)** özel bir şekli olarak düşünülebilir. Yani, düğüm boyutu 2 olan bir B-Ağacında, her düğümde ya iki ya da tek eleman bulunacak ve bulunan eleman sayısının bir fazlası kadar çocuk bulunacaktır.
2-3 Ağaçları, AA ağaçlarının (AA-trees), **eş şekillisi (isometry)** olarak düşünülebilir.

Red Black Tree (Kırmızı Siyah Ağaç)

- İkili arama ağaçları gibi çalışır yani herhangi bir düğümün solunda kendinden küçük ve sağında kendinden büyük veriler durur. Aşağıda düğümlerin taşıması gereken özellikler yer almaktadır:
  1. Kök (root) düğüm her zaman siyahtır.
  2. Bütün yapraklar (leaf nodes) siyahtır.
  3. Herhangi bir kırmızı düğümün bütün çocukları siyahtır.
  4. Herhangi bir node'dan yaprak düğüme gidilen bütün yollarda eşit sayıda siyah düğüm bulunur.

- Ağacın aynı seviyedeki düğümleri aynı renktir. Ayrıca ağaçtaki renklendirme kökten başlayarak, siyah – kırmızı – siyah – kırmızı sıralamasıyla değişmektedir.
- En uzun yol (kökten yaprağa) en kısa yolun uzunluğunun iki katından fazla değildir.
- En kısa yol, siyah node'lardan oluşur.
- En uzun yol, siyah ve kırmızı node'ların alternatiflerinden oluşur.
- Arama işlemi dengeli olan BST ile aynıdır karmaşıklığı **O(logn) 'dir.**
- Ekleme veyahut silme işlemi gerekebilir onlarda da **O(logn)** yapılır.

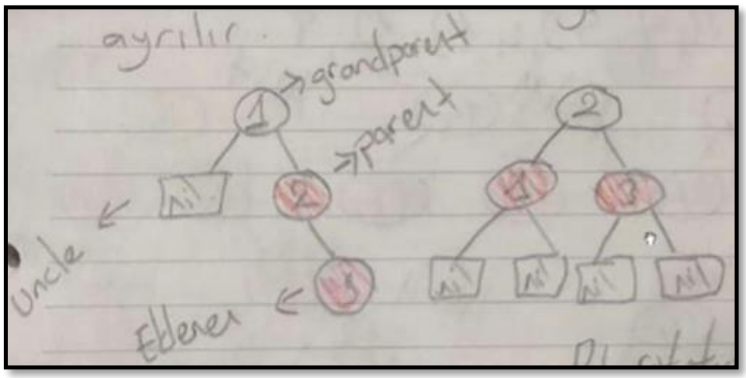


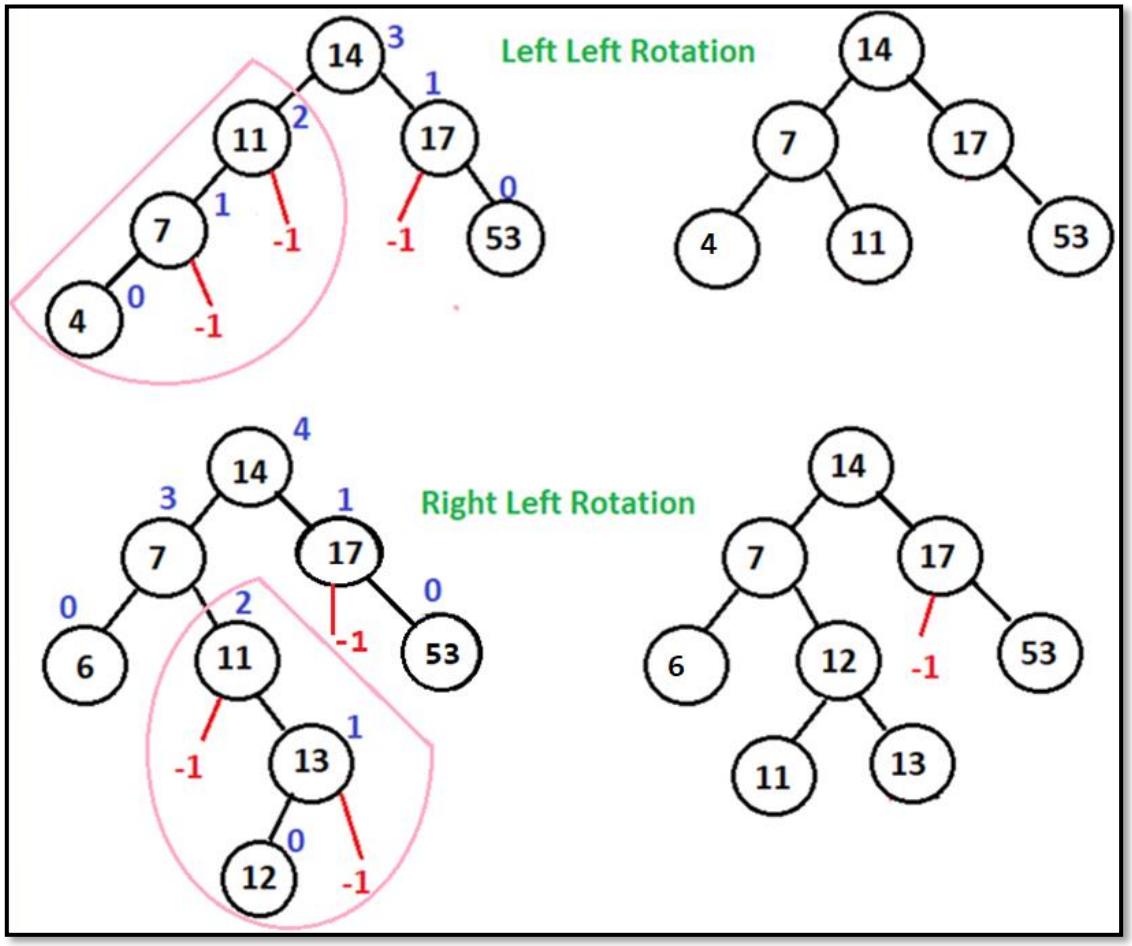
- Yukarıda görülen örnek ağaçta, kök düğüm siyah, yapraktaki boş düğümler (null) siyah ve bu düğümler dışındaki her düğümün çocukları, kendisinin ters rengindedir. Örneğin kırmızı olan 13 düğümünün çocukları siyah, siyah olan 25 düğümünün çocukları ise kırmızıdır. Bu anlamda kutu ile gösterilen yaprak düğümler göz ardı edilirse, aynı seviyedeki düğümler aynı renkte olmaktadır.

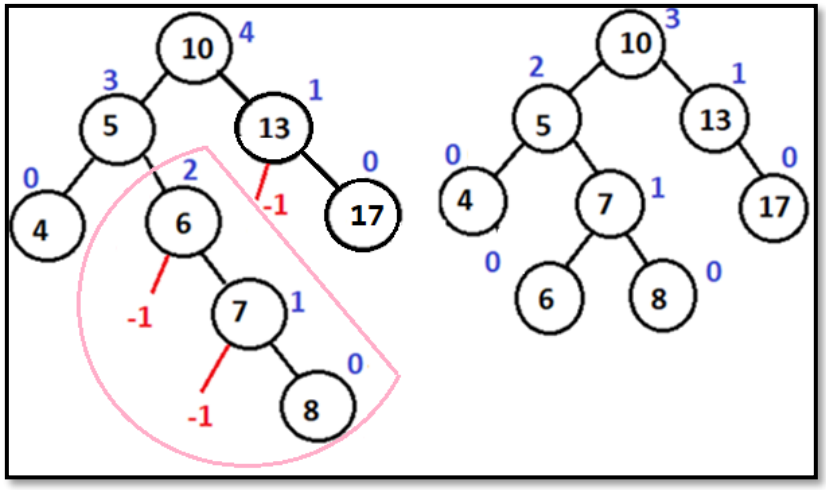
- Ekleme işlemi normal bir ikili arama ağacına (BST) ekler gibi başlar. Bu işlem sırasında yeni düğümün kırmızı olacağını kabul ederiz.
  Ekleme işlemi sırasında uyulması gereken 3 temel kural vardır:
  1. Kök düğüm (root node) her zaman için siyahtır.
  2. Herhangi bir düğümden, yapraklara kadar uzanan herhangi bir yolda, eşit sayıda siyah düğüm bulunur.
  3. Bir kırmızı düğümün, kırmızı çocuğu bulunamaz.

  Yukarıdaki istenmeyen durumlardan birisi oluştuğunda, ağaçtaki düğümlerin rengi değiştirilir ya da değiştirilemiyorsa ağaçta dengelemek için döndürme (rotation) işlemi yapılır.

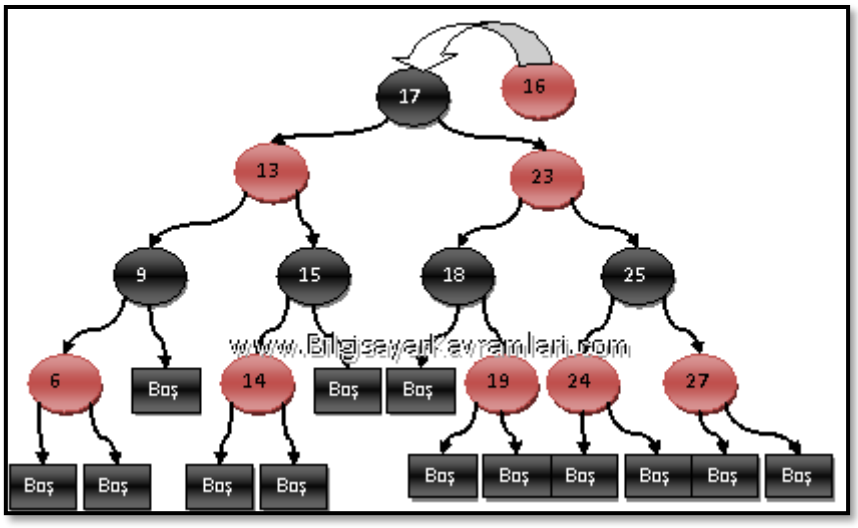
  Rotasyon
  - Hedef, ağacın yüksekliğini azaltmaktır.
  - Büyük alt ağaçlar kısalır, kısa alt ağaçlar uzar.
    Left – rotate ve right – rotate olarak ikliye ayrılır.


grandparent
parent
Uncle
Ebeveyn

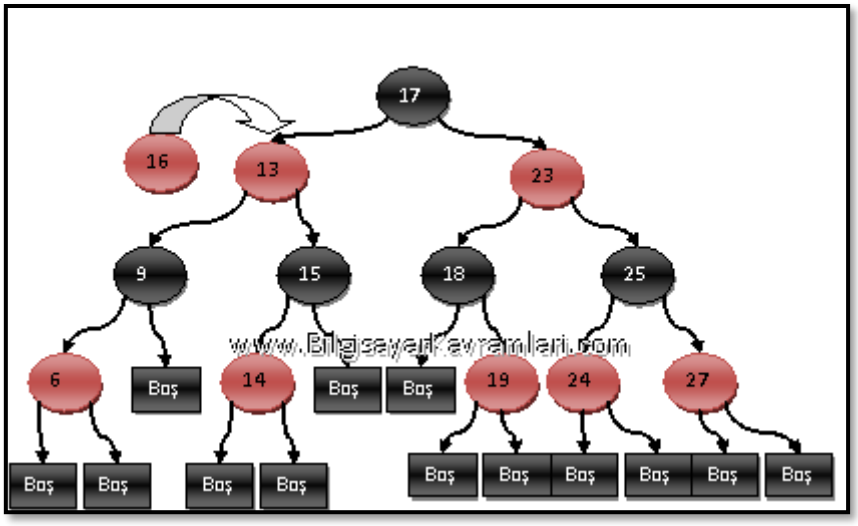

Left Left Rotation
14
11
17
7
53
4
Right Left Rotation
14
7
17
6
11
53
12
13


10
5
13
4
6
17
7
8

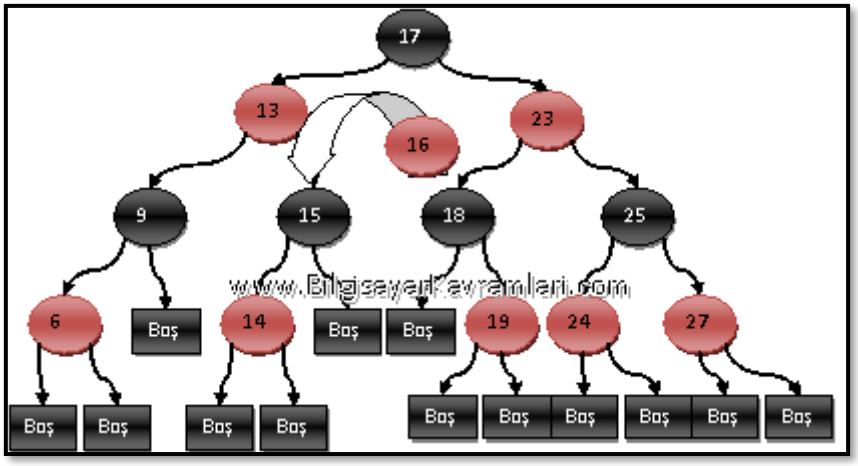
Ekleme işleminde istenmeyen durumlardan birisi oluştuğunda, ağaçtaki düğümlerin rengi değiştirilir ya da değiştirilemiyorsa ağaçta dengelemek için döndürme (rotation) işlemi yapılır.



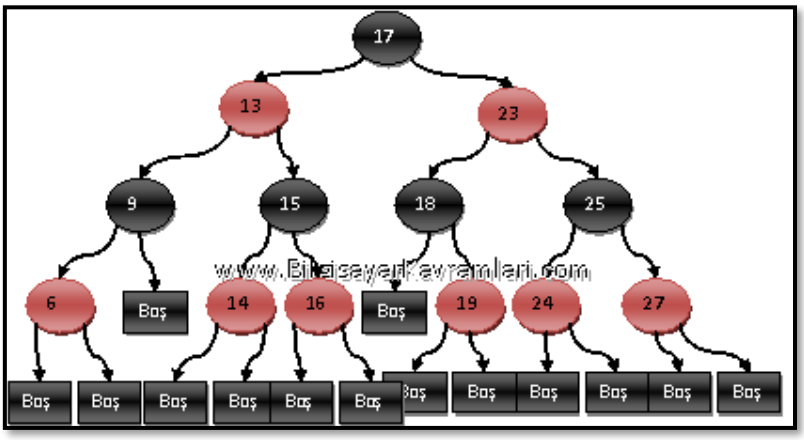
Örneğin, yukarıdaki şekilde gösterildiği üzere 16 sayısını ağaca eklemek isteyelim.



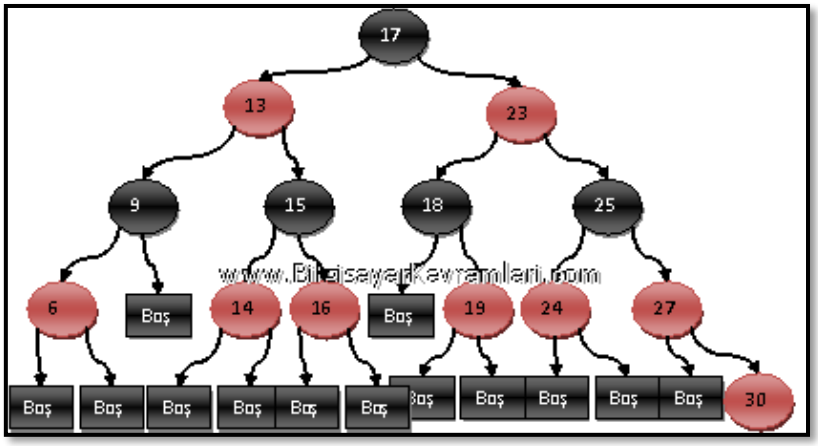
16 değeri, 17'den küçük olduğu için, klasik bir ikili arama ağacında hareket eder gibi , ağacın sol tarafında devam edilecek ve 13 ile karşılaştırılacak.



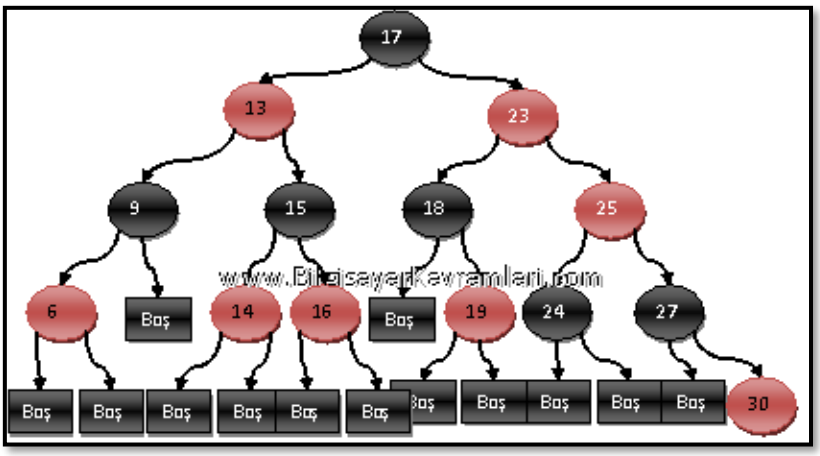
13'ten büyük olduğu için sağa bakılıyor. Ve 15'ten de büyük olduğu için 15'in sağına ekleniyor.



Görüldüğü üzere yeni eklenen 16, 15'ten büyük ve sağındaki boş yere yerleştirilmiştir. Bu yerleştirme sonucunda iki ardışık kırmızı durumu oluşmadığı için sorun yoktur. Yani bir siyah düğüm altına kırmızı düğüm eklenmiştir. Şimdi örnek olarak sırasıyla 30 ve 32 sayılarını ekleyelim.

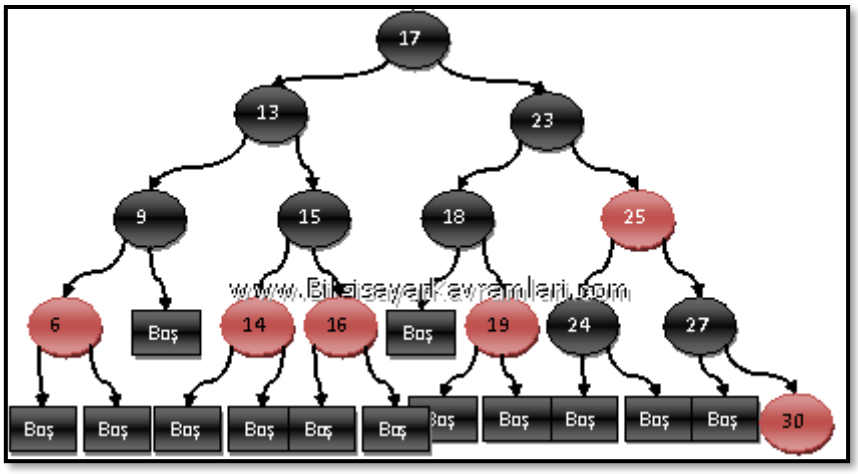


Ağaca yukarıdaki şekilde 30 değeri eklenince, ağacın en sağına yerleşmekte ve istenmeyen bir durum olan iki kırmızı arka arkaya gelmektedir. Çözüm olarak ağaçtaki düğümlerin rengi değiştirilmelidir.

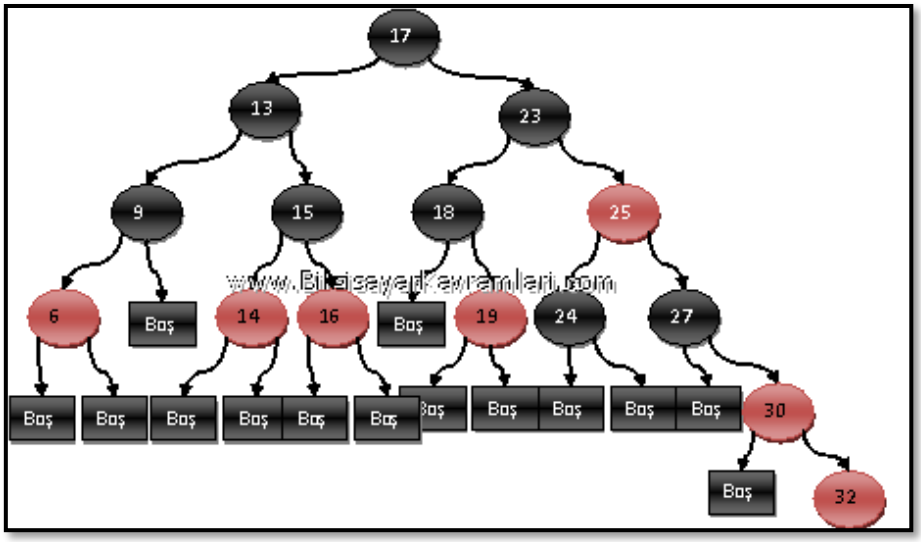


Öncelikle, 30'un hemen üzerinde bulunan 27 düğümü, iki kırmızı düğüm arka arkaya olamayacağı için siyaha çevrilir. Siyaha çevrilen 27 numaralı düğümün büyük babası 23 numaralı düğümdür. Dolayısıyla 27 numaralı düğümün amcası 18 numaralı düğüm olur. 25 numaralı düğümün kırmızıya çevrilmesi, 23 numaralı büyük baba için problem oluşturur çünkü bir kırmızı düğümün çocukları siyah olmalıdır. Aynı zamanda problem 17 numaralı düğümden yaprağa kadar giden yolda da eşit miktarda siyah düğüm bulunmalıdır. Yukarıdaki örnekte 23 numaralı düğümün siyah kalması durumunda, 17 numaralı düğümden gidilebilen sağ yol ile sol yol arasında düğüm sayıları farklı olmaktadır.

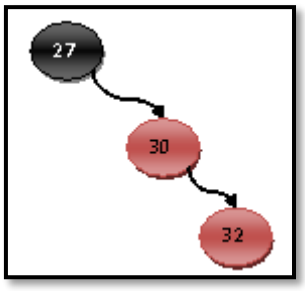
Çözüm olarak 17 numaralı düğüm kırmızı yapılabilir ancak bu durumda da kök düğümün kırmızı olamayacağı kuralı ile ihtilafa düşülür. Dolayısıyla bu adımda çözüm 13 ve 23 numaralı düğümlerin siyah yapılmasıdır.



Ağacın çalışma şeklini daha iyi anlayabilmek için ağaca bir de 32 değerinde bir düğüm eklemeyi deneyelim:

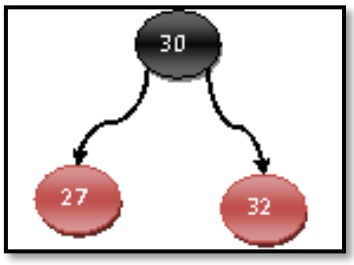


Yukarıdaki örnekte görüldüğü üzere, 32 değeri, bir önceki adımda eklediğimi 30 değerinin sağına gelmektedir. Hemen dikkat edilebilecek bir problem, 30-32 ikilisinin arka arkaya kırmızı olmasıdır. Bu durumda 30 düğümünün siyah yağılması veya 32 düğümünün siyah yapılması problemi çözmez çünkü kökten yapraklara kadar giden yolda eşit sayıda siyah düğüm bulunmalıdır. Bu yola yeni bir siyah düğüm eklenmesi bu dengeyi bozar. Çözüm olarak saat yönünün tersine döndürme işlemi uygulanıp çocukların siyah yapılması gerekir.
Bu durumu aşağıdaki şekil üzerinden açıklamaya çalışalım:

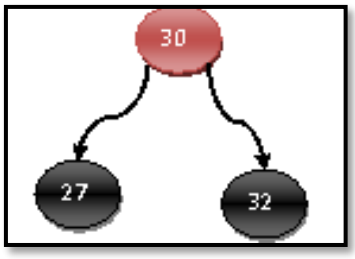


Yukarıdaki 2 problem bulunuyor:

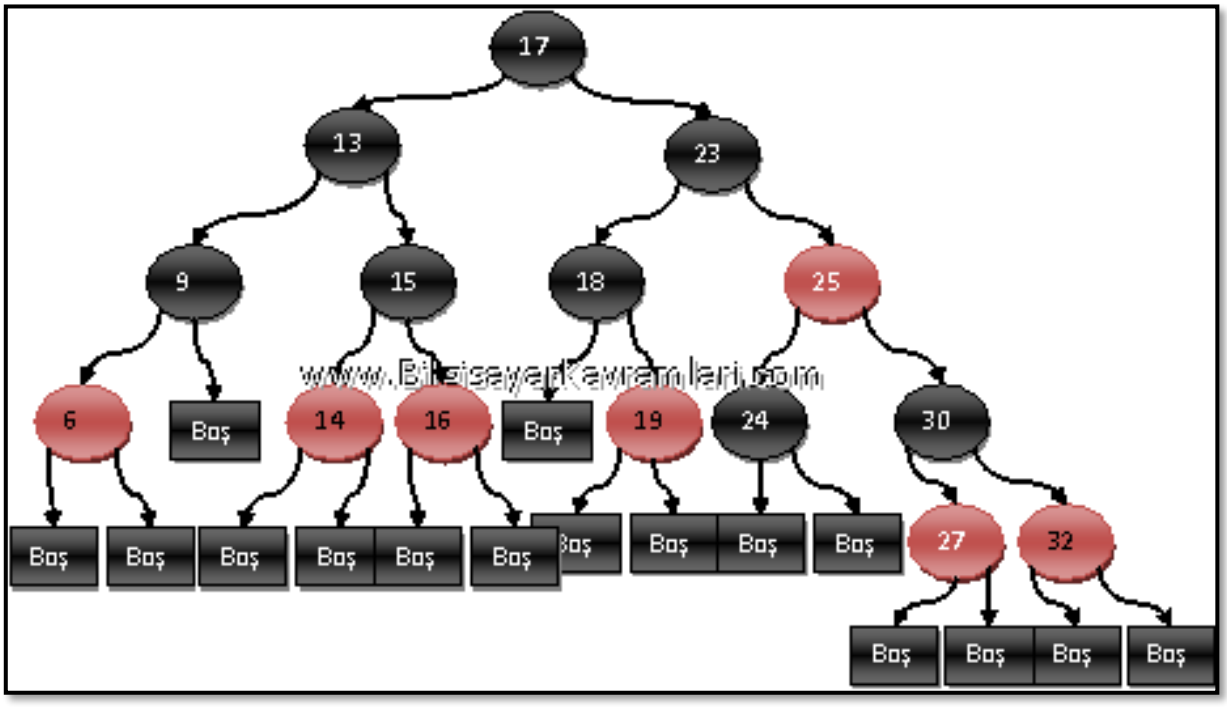
- İki kırmızı düğüm arka arkaya
- Yolda tek siyah düğüm bulunmasına izin verilmiş, yapraklara kadar giden ikinci bir siyah düğüm çıkarma hakkımız yok.



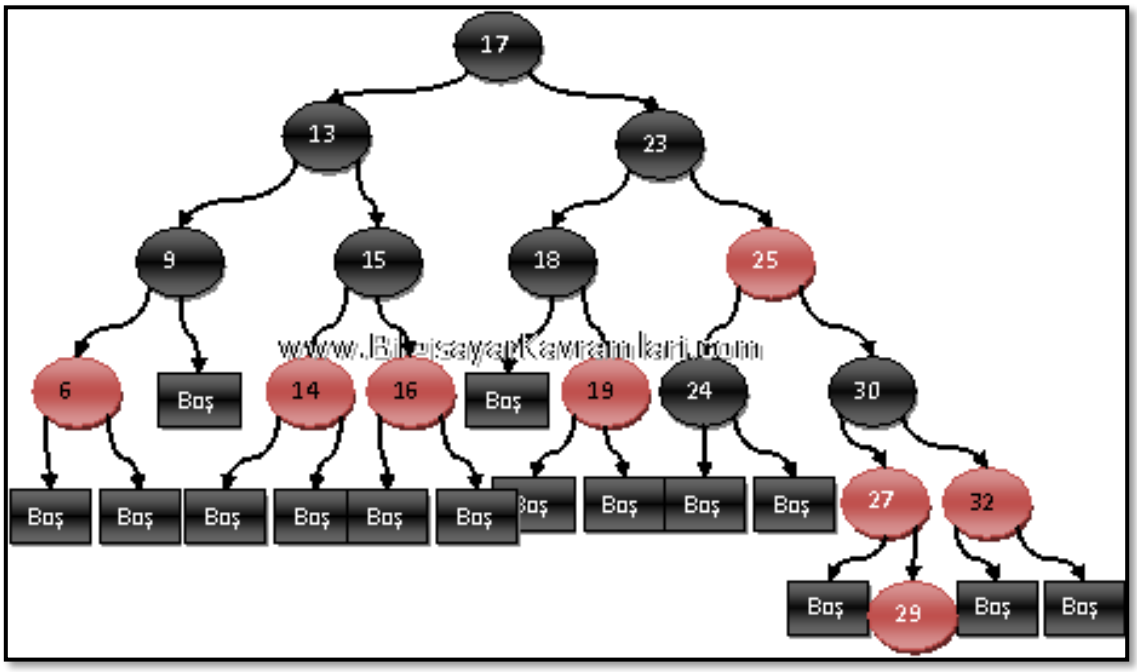
Yukarıdaki yeni halinde, kural bozulmaksızın hem tek siyah ile yaprağa ulaşılıyor hem de iki kırmızı ihlalinden kurtulunuyor.
Benzer bir çözüm aşağıdaki şekilde olabilirdi:



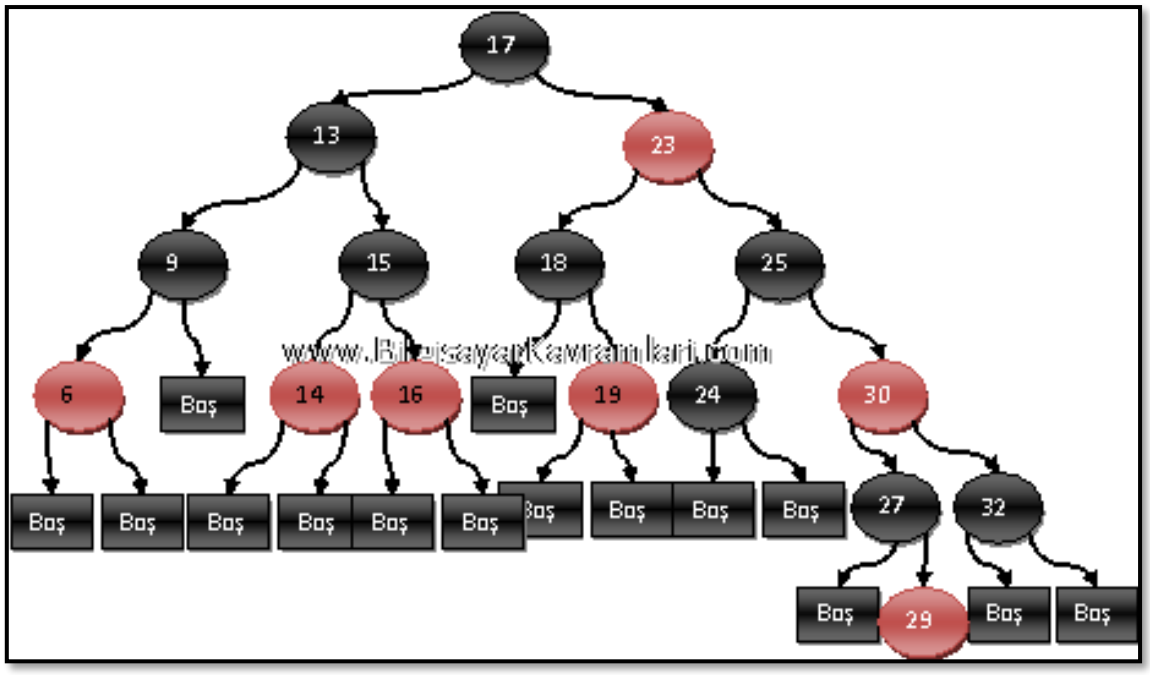
Ancak yukarıdaki bu yeni halimiz ağacın bütünüyle uyuşmamaktadır. Yani döndürme işleminin yapıldığı ağacın üst düğümü 25 olduğu ve kırmızı olduğu için bu ikinci çözüm yeni bir kırmızı kırmızı komşuluğu doğuracaktır. Dolayısıyla ilk örnekte olduğu gibi problemi çözebiliriz:



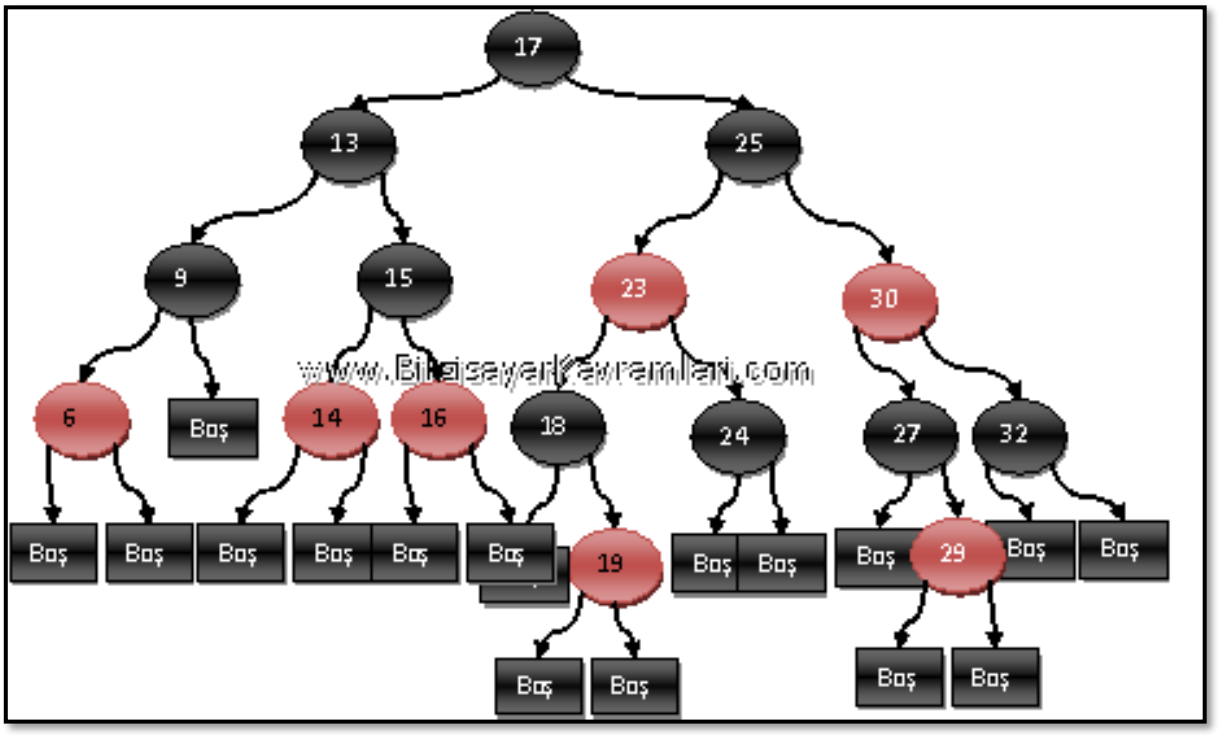
Yukarıdaki yeni halimizde kuralların tamamına uyarak ağaca yeni sayıyı eklemiş bulunuyoruz. Benzer bir durumu oluşturup daha yukarıdan döndürme (rotation) işleminin nasıl yapıldığını, ağaca 29 sayısını ekleyerek görebiliriz. Bakın 29 sayısı eklenince de yapılan işlem aslında aynıdır sadece daha büyük bir döndürme işlemi gerçekleştirilmiş olacaktır:



İki kırmızı durumu oluşuyor. Bu noktada artık ikisinden birisinin siyah olması sorunu çözmez çünkü yaprağa kadar olan yolda her halükarda bir fazla siyah düğüm oluşur.
Çözüm olarak sırasıyla kırmızı-siyah dönüşümü yapıyoruz:



Görüldüğü üzere köke kadar giden yolda kırmızı-kırmızı durumunu engellemek için düğümler bir kırmızı bir de siyah sırasına dönüştürüldü. Şimdi problem 23 numaralı düğümün sağ ve sol taraflarındaki yol uzunluğunun farklı olması. Yani 23 numaralı düğümün sol tarafındaki problem çözülürse 17 numaralı düğüm için bir problem yok. Döndürme işlemi gerçekleştiriyoruz:



Yukarıda görüldüğü üzere 25 numaralı düğüm23 ve 30 numaralı düğümler arasındayken, bu düğümlerin atası durumuna geçmiştir.