



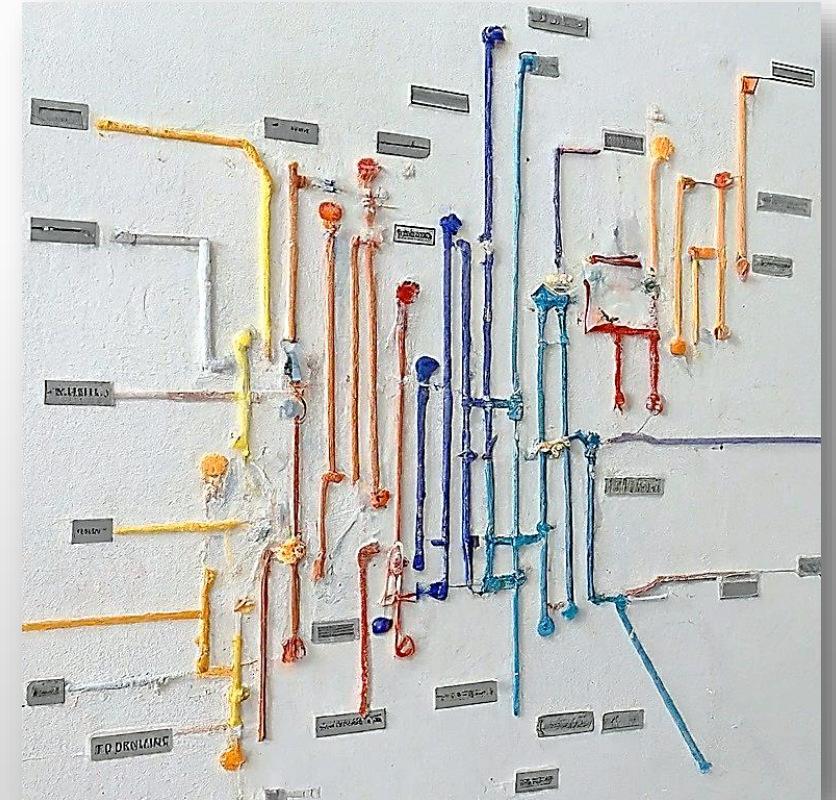
# Bölüm 4: Çizge Algoritmaları

## Algoritmalar



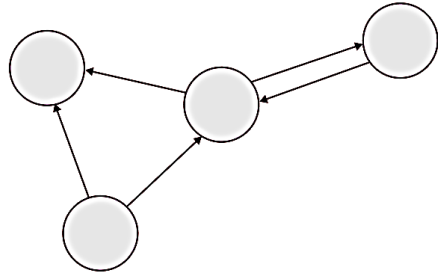
# Çizge Algoritmaları

- Dünya aslında bir ağ gibidir.
  - Şehirler yollarla,
  - İnsanlar ilişkilerle,
  - Bilgisayarlar kablolarla birbirine bağlıdır.
- Çizge algoritmaları bu ağları inceler ve anlamlandırır.

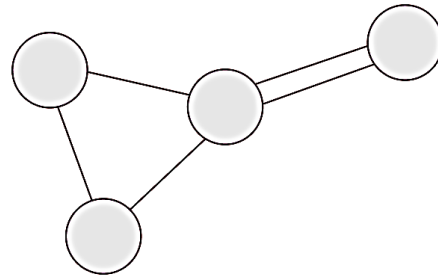




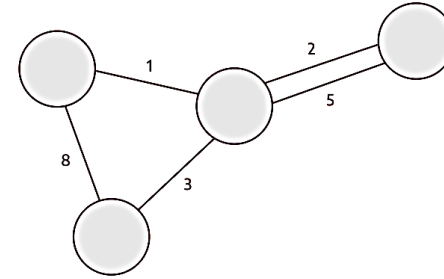
# Çizge Türleri



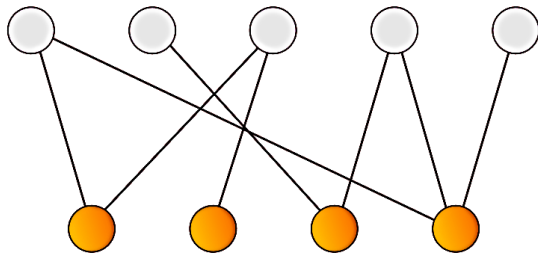
Directed graph



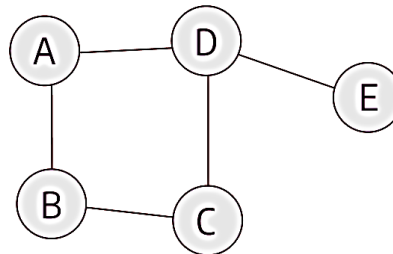
Undirected



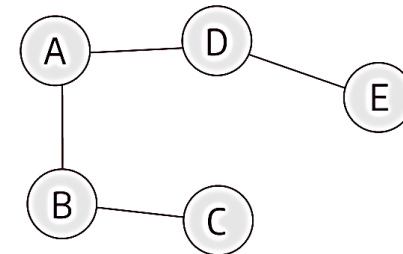
Weighted



Bipartite graph



Cyclic graph



Acyclic graph



# Çizge Algoritmaları

- Birbirine bağlı noktalar (düğüm) ve bu noktaları birleştiren çizgiler (kenar) ile temsil edilen ağ yapılarını inceler.
- Ağlarda en kısa yolu hesaplama, gruplama gibi işlemleri gerçekleştirir.
- Sosyal ağlar, harita uygulamaları, navigasyon gibi birçok alanda kullanılır.



# Çizge Algoritmalarının Çeşitleri

- Farklı çizge algoritmaları, farklı işlemler için kullanılır.
- Derinlik Öncelikli Arama (DFS):
  - Bir düğümden başlar, dallanarak tüm ağı gezer.
- Genişlik Öncelikli Arama (BFS):
  - Bir düğümden başlar, katman katman tüm ağı gezer.
- Dijkstra Algoritması:
  - Başlangıç düğümünden diğer düğümlere en kısa yolları bulur.
- Kruskal Algoritması:
  - Bir ağı minimum maliyetle birbirine bağlayan kenarları seçer.



# Çizge Algoritmaları

- DFS bir labirentten çıkış yolu ararken kullanılabilir.
- BFS bir haberin tüm şehire yayılma sürecini modelleyebilir.
- Dijkstra en kısa sürede teslimat yapmak için kullanılabilir.







# Çizge Algoritmaları

- Çizge gezinme algoritmaları (*Graph traversal*)
- En kısa yol algoritmaları (*Shortest path*)
- Minimum kapsayan ağaç algoritmaları (*Minimum spanning tree*)
- Ağ akış algoritmaları (*Network flow*)



# Minimum Kapsayan Ağaç Algoritmaları

- Çizgedeki,
  - tüm düğümleri birbirine bağlayan ve
  - toplam kenar ağırlığının en az olduğu alt ağaçtır.
- *Kruskal*, kenarları ağırlıklarına göre sıralar ve döngü oluşturmeyen kenarları seçerek ağacı oluşturur.
- *Prim*, başlangıç düğümünden başlayarak, her adımda en düşük ağırlıklı kenarı seçerek ağacı büyütür.





# Kruskal Algoritması

- Çizgede tüm düğümleri birbirine bağlayan en kısa ağırlıklı ağacı oluşturur.
- Açgözlü (greedy) bir yaklaşım kullanır.
- Joseph Kruskal tarafından geliştirilmiştir.



# Algoritma İlkeleri

- Ağırlıklı çizge üzerinde çalışır.
- Tüm düğümleri en küçük ağırlıklı kenarları kullanarak birleştirir.
- Döngü oluşturmada, minimum kapsayan ağacı oluşturur.
- Başlangıçta, her düğüm ayrı bir ağacı temsil eder.
  - Adım adım bu ağaçlar birleştirilir.



# Algoritma Adımları

- Adım 1: Çizge içindeki tüm kenarlar ağırlıklarına göre sıralanır.
- Adım 2: Sıralı kenarlar arasından en küçük ağırlıklı kenar seçilir.
- Adım 3: Seçilen kenar, döngü oluşturmuyorsa, ağaç içine eklenir.
- Adım 4: Eğer seçilen kenar, farklı ağaçlara ait düğümleri birleştiriyorsa, bu kenar ağaç içine eklenir ve ağaçlar birleştirilir.



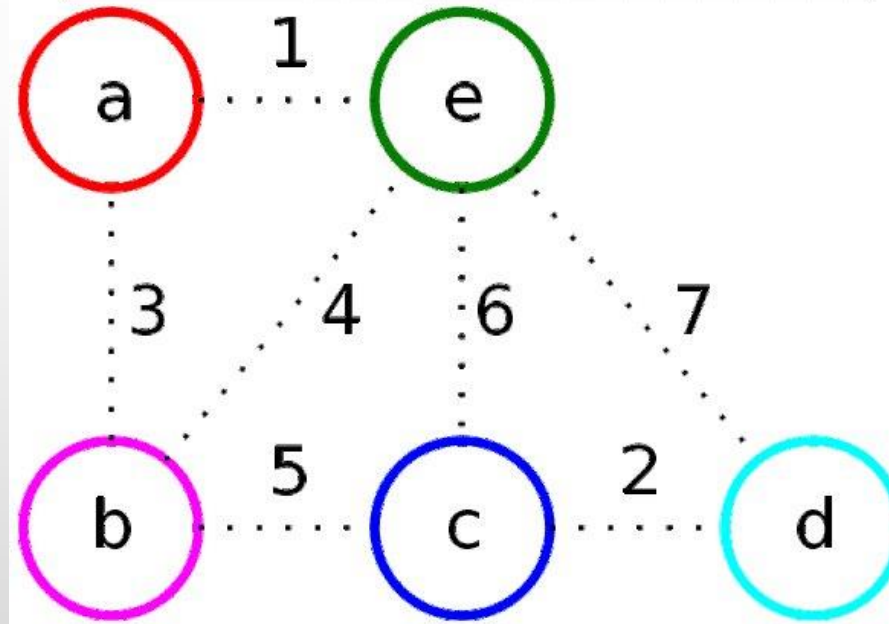
# Karmaşıklık Analizi

- Kenarların ağırlıklarına göre sıralanması:
  - $O(E \log E)$
- Birleştirme-bulma (union-find) işlemleri:
  - $O(\log E)$  (amortize edilmiş)

# Kruskal



| Edge   | ab | ae | bc | be | cd | ed | ec |
|--------|----|----|----|----|----|----|----|
| Weight | 3  | 1  | 5  | 4  | 2  | 7  | 6  |

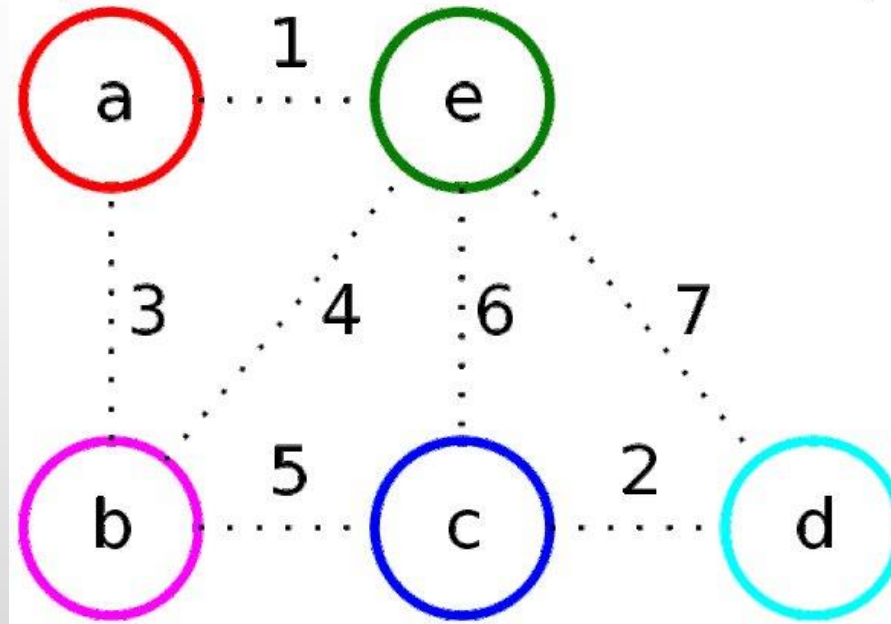


# Kruskal



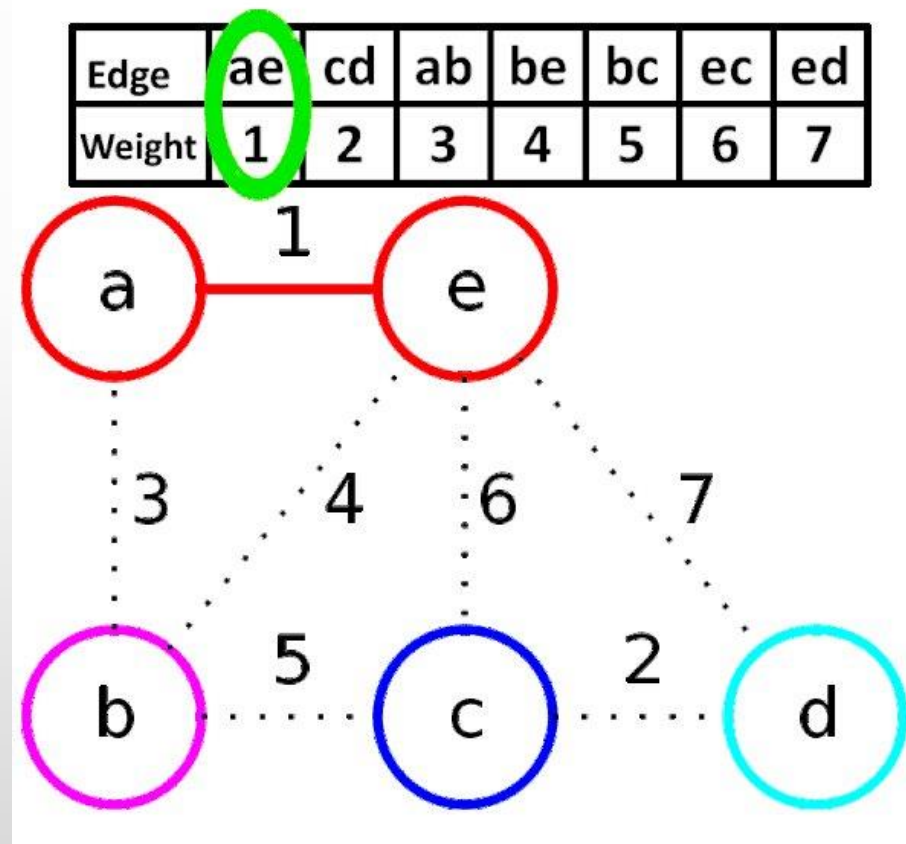
## SORT THE EDGES

| Edge   | ae | cd | ab | be | bc | ec | ed |
|--------|----|----|----|----|----|----|----|
| Weight | 1  | 2  | 3  | 4  | 5  | 6  | 7  |

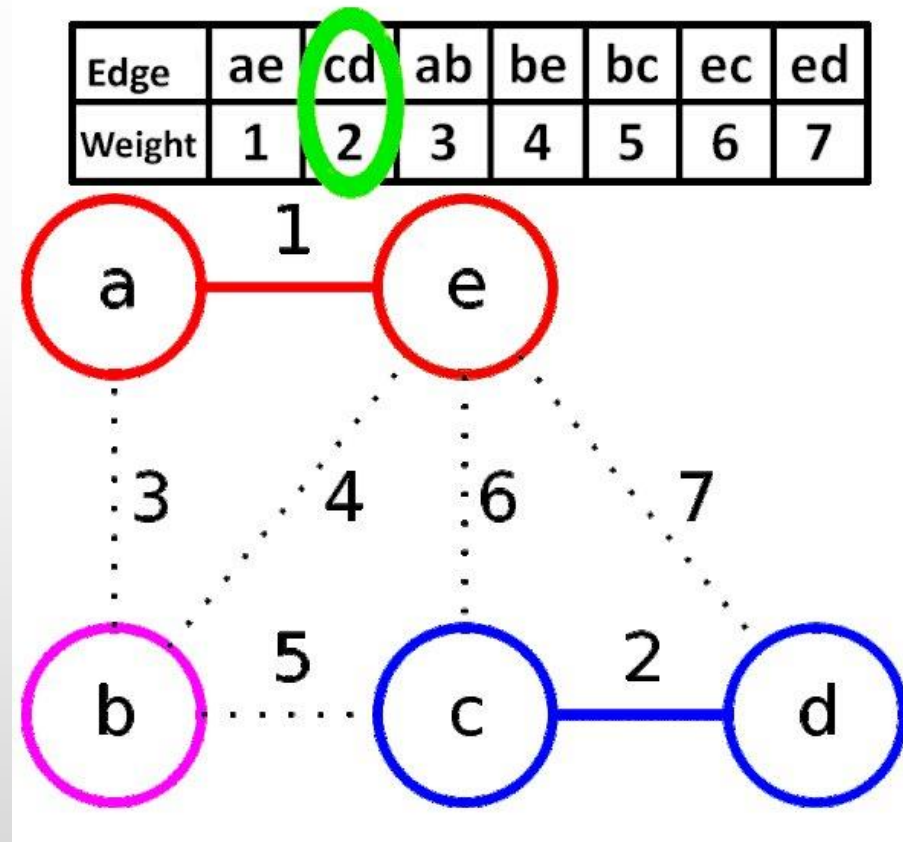




# Kruskal



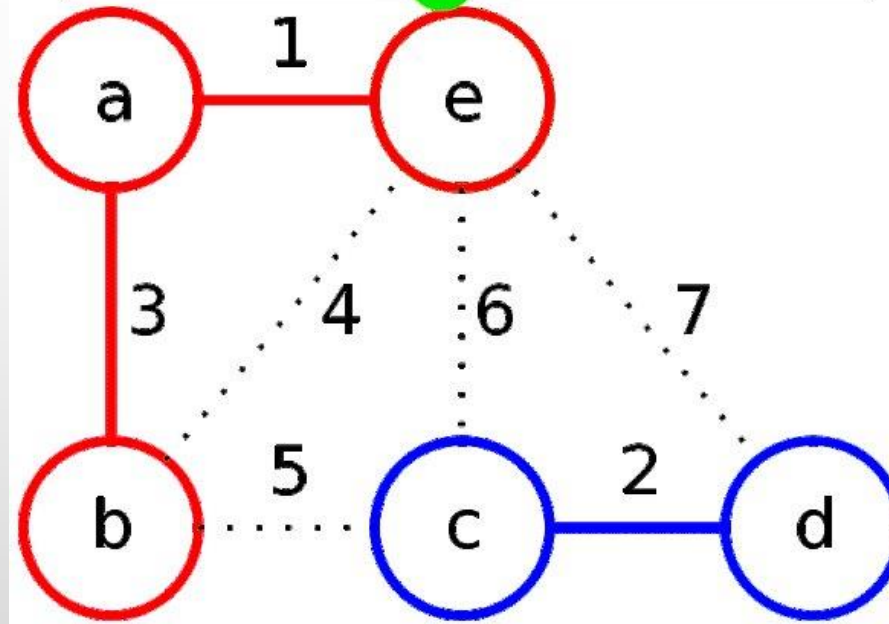
# Kruskal



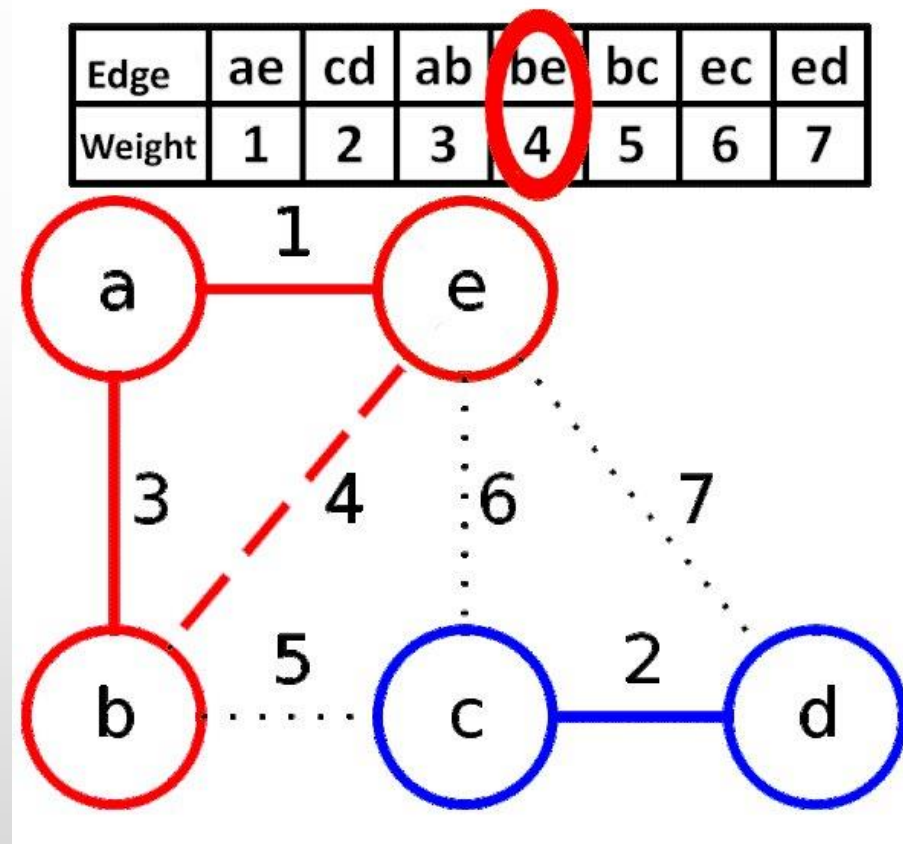
# Kruskal



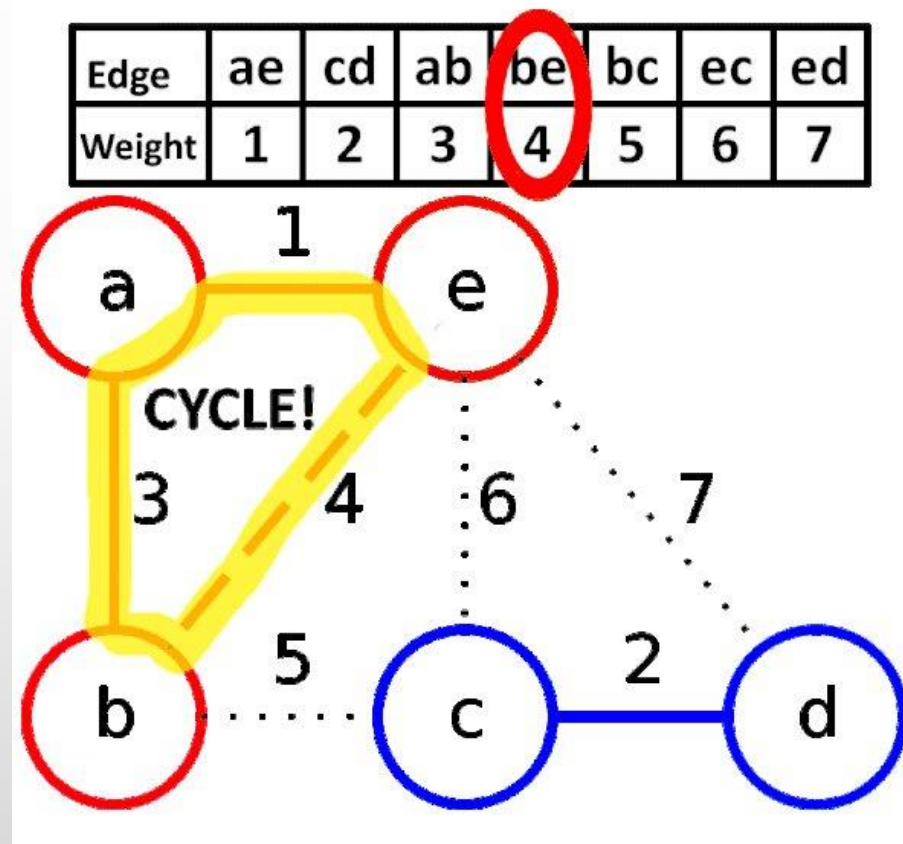
| Edge   | ae | cd | ab | be | bc | ec | ed |
|--------|----|----|----|----|----|----|----|
| Weight | 1  | 2  | 3  | 4  | 5  | 6  | 7  |



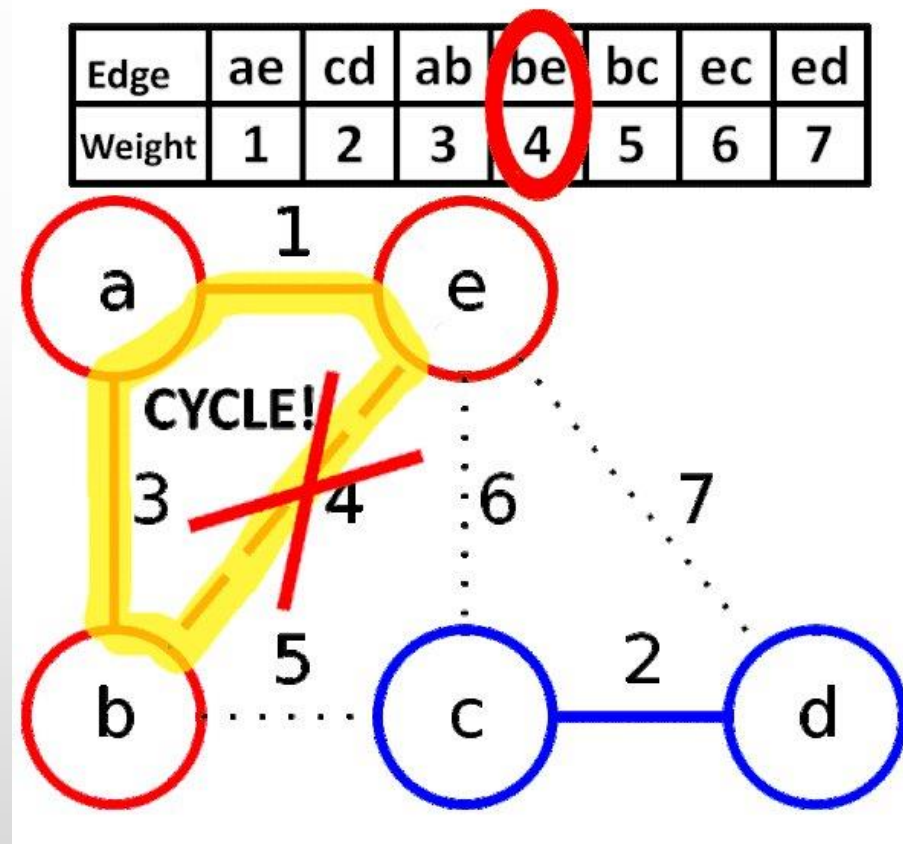
# Kruskal



# Kruskal

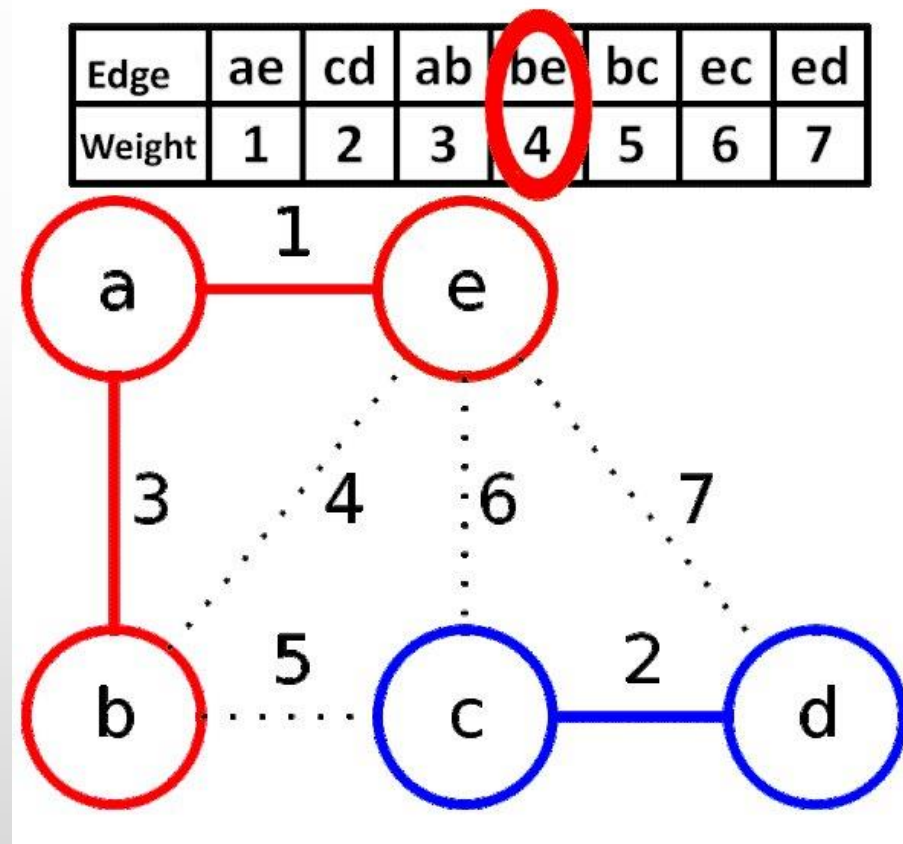


# Kruskal





# Kruskal

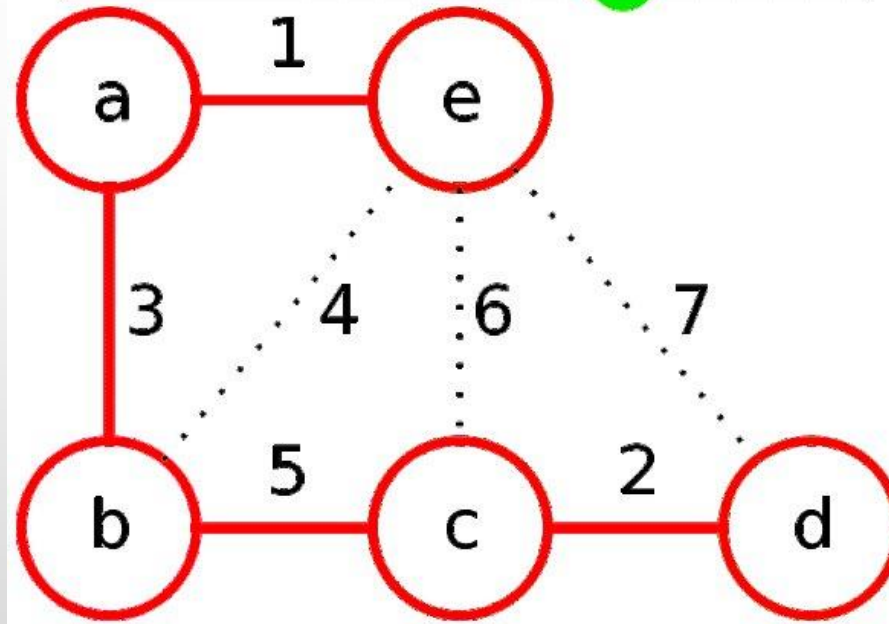


# Kruskal



**TREE HAS BEEN BUILT**

| Edge   | ae | cd | ab | be | bc | ec | ed |
|--------|----|----|----|----|----|----|----|
| Weight | 1  | 2  | 3  | 4  | 5  | 6  | 7  |

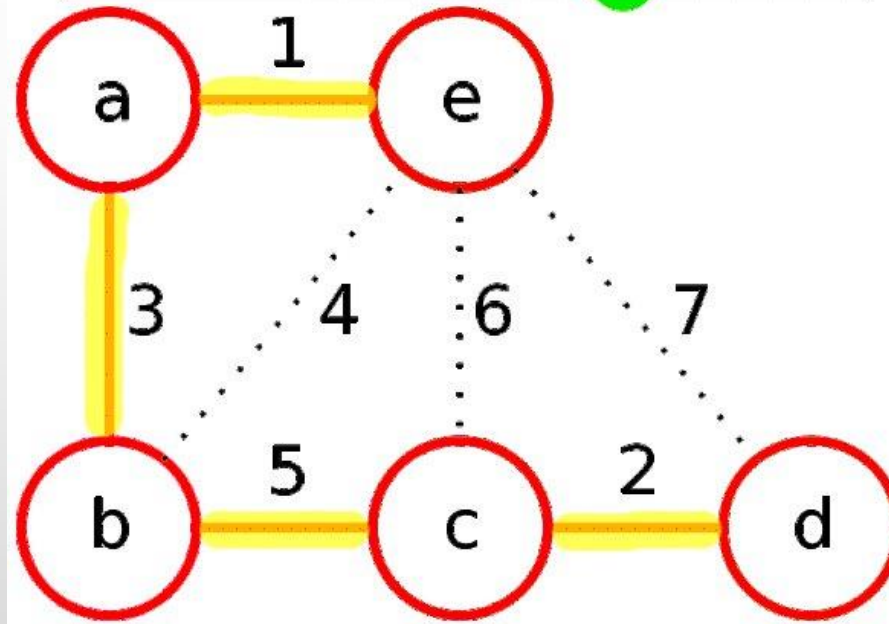


# Kruskal



**TREE HAS BEEN BUILT**

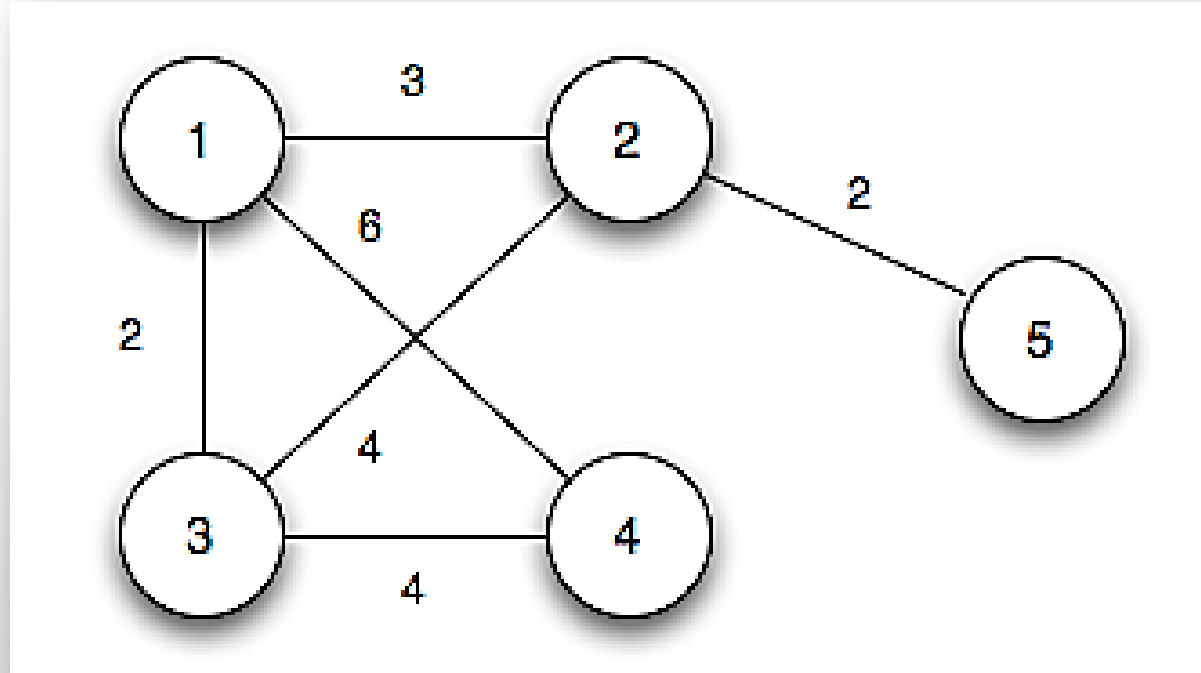
| Edge   | ae | cd | ab | be | bc | ec | ed |
|--------|----|----|----|----|----|----|----|
| Weight | 1  | 2  | 3  | 4  | 5  | 6  | 7  |





# Örnek

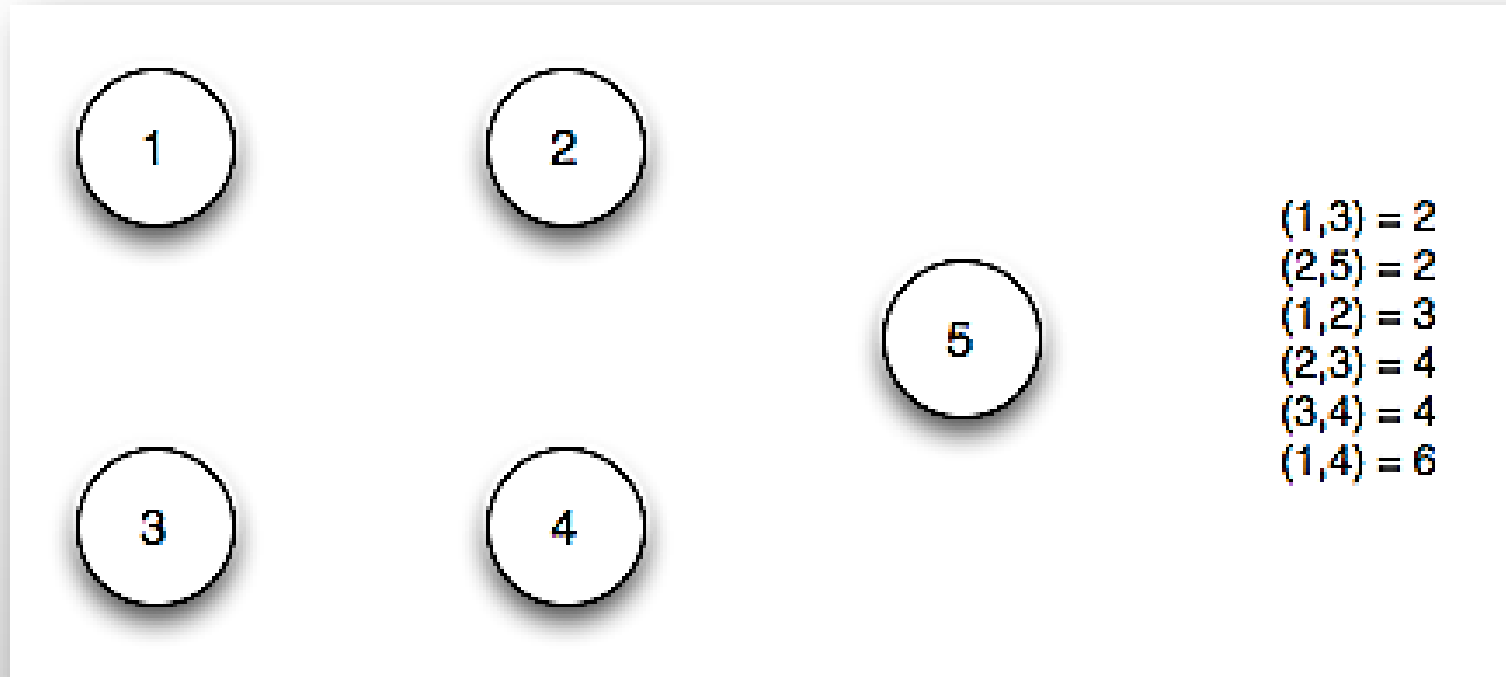
- Aşağıdaki yönsüz çizge verilsin.





# İlklendirme Aşaması

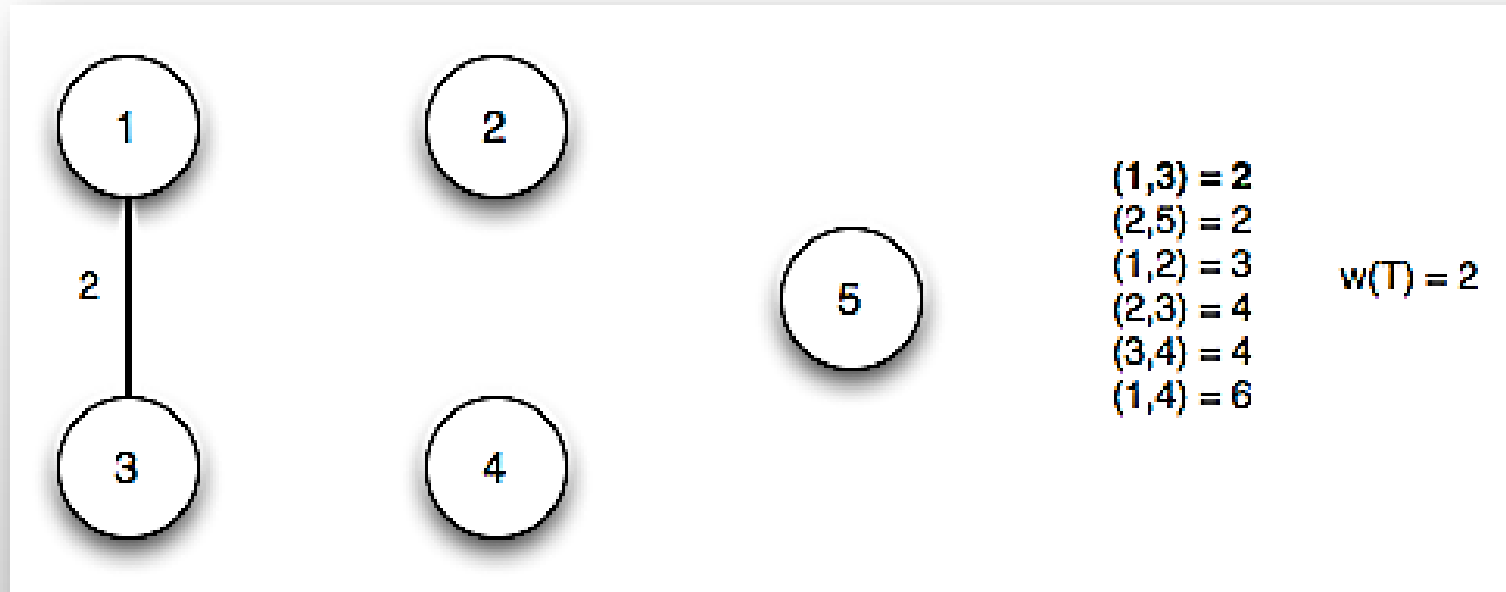
- Her bir düğüm ayrı bir ağaç yapılır.
- Kenarlar ağırlıklarına göre sıralanır.





# Adım 1

- $(u_1, u_3)$  kenarını ekle
- $v_1$  ve  $v_3$  birleştir

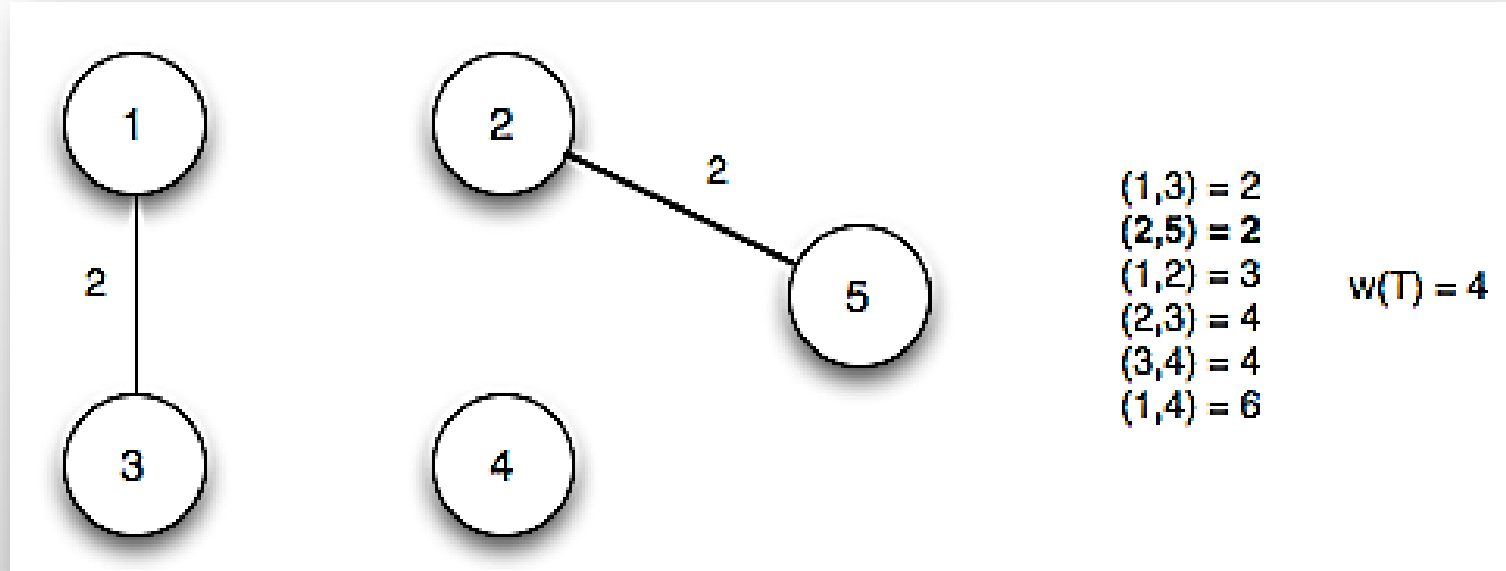






## Adım 2

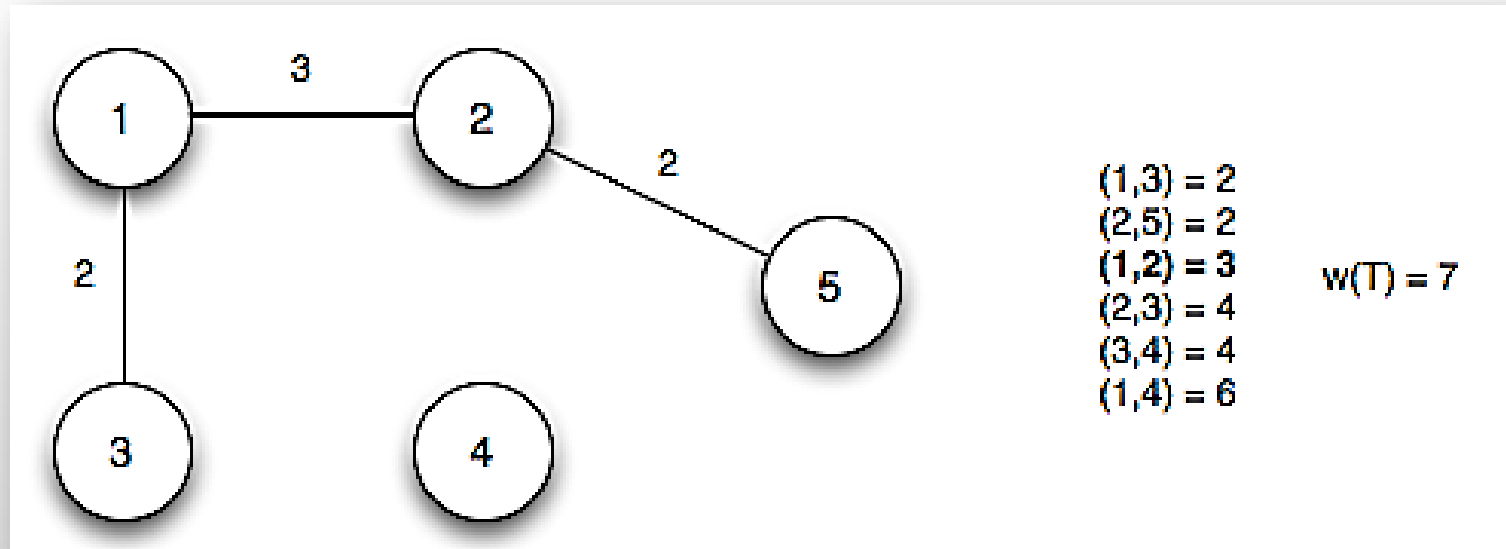
- $(u_2, u_5)$  kenarını ekle.
- $v_2$  ve  $v_5$  birleştir.





## Adım 3

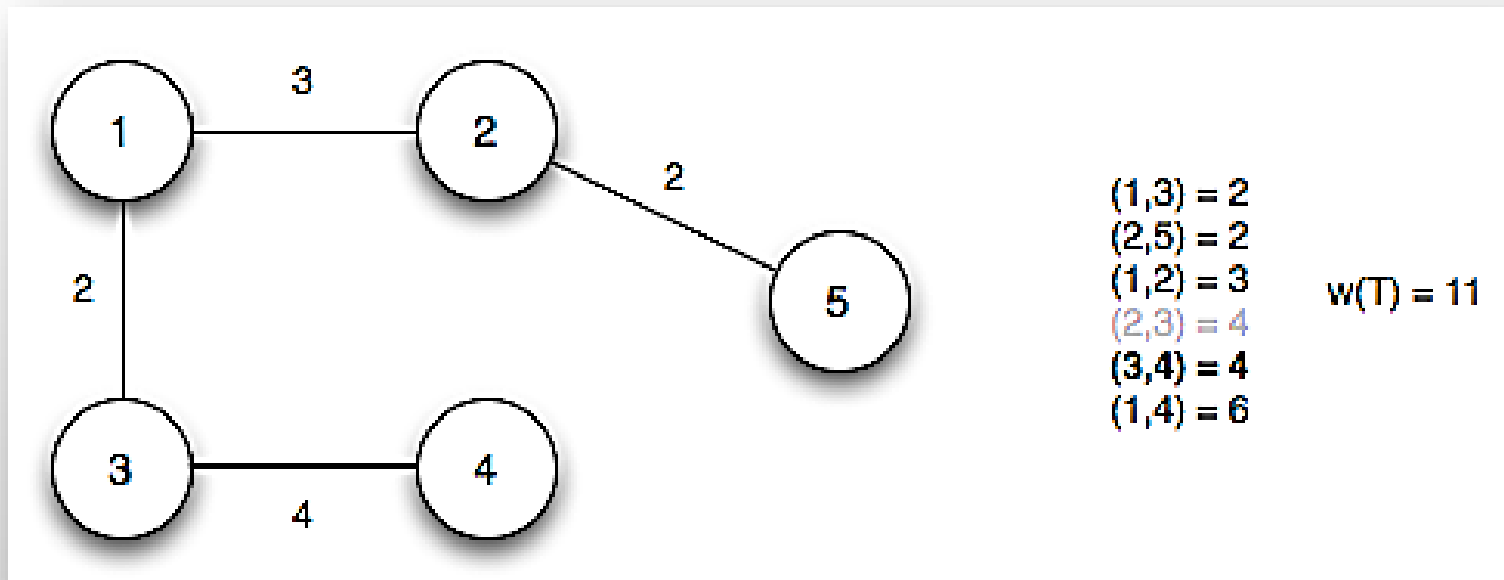
- $(u_1, u_2)$  kenarını ekle.
- Adım 1 ve Adım 2'de oluşan iki ağacı birleştir.





## Adım 4

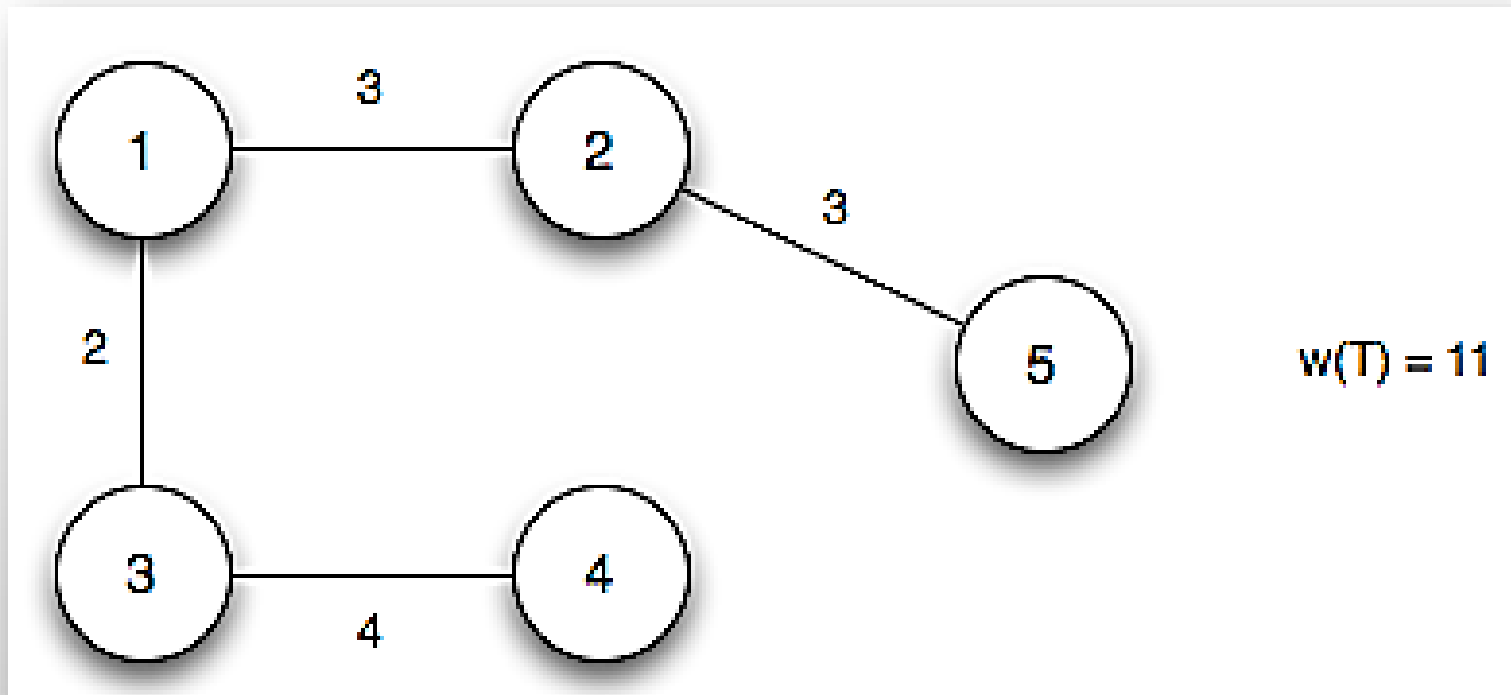
- $(u_3, u_4)$  kenarını ekle,  $v_4$  birleştir.
- $(u_2, u_3)$  ayrı ağaçları birleştirmiyor!





# Son Durum

- (u1,u4) kenarı ayrı ağaçları birleştirmiyor!







# Prim Algoritması

- Tüm düğümleri bağlayan en küçük ağırlıklı ağacı oluşturmayı amaçlar.
- Açgözlü (greedy) bir algoritmadır.
- Robert C. Prim tarafından geliştirilmiştir.





# Algoritma İlkeleri

- Ağırlıklı çizge üzerinde çalışır.
- Tüm düğümleri en küçük ağırlıklı kenarlarla birleştirir.
- Başlangıçta, bir düğüm seçilir ve ağacın başlangıç düğümü kabul edilir.
- Her adımda,
  - ağaç içinde olmayan düğümler arasından,
  - ağaçta olan en küçük ağırlıklı kenar ile yeni bir düğüm eklenir.



# Algoritma Adımları

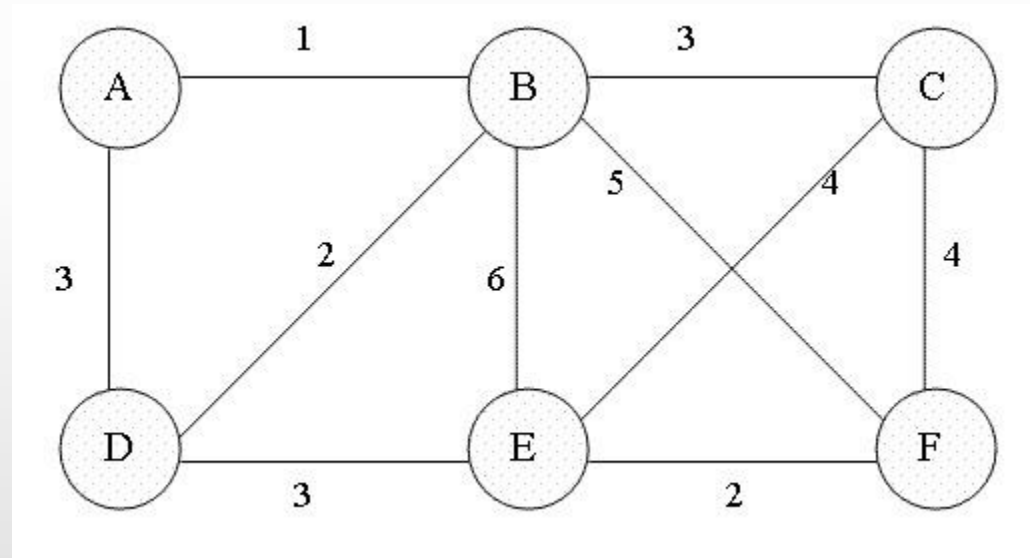
- Adım 1: Başlangıç düğümü seçilir ve bu düğüme ait olan tüm kenarlar bir öncelik kuyruğuna eklenir.
- Adım 2: Kuyruktan, ağaçta olmayan en küçük ağırlıklı kenar seçilir.
- Adım 3: Seçilen kenar ile bağlantılı olan yeni düğüm ağaca eklenir.
- Adım 4: Yeni eklenen düğüme bağlı kenarlar öncelik kuyruğuna eklenir.



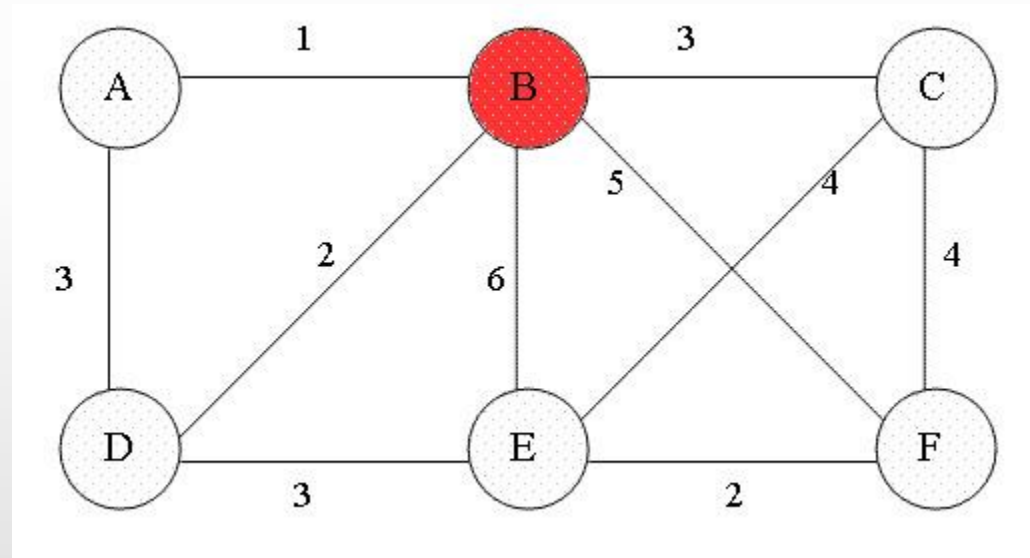
# Karmaşıklık Analizi

- Prim Algoritması'nın karmaşıklığı
  - $O(E + V \log V)$  veya
  - $O(E \log V)$  olarak ifade edilir.
- E kenar sayısını,
- V düğüm sayısını temsil eder.

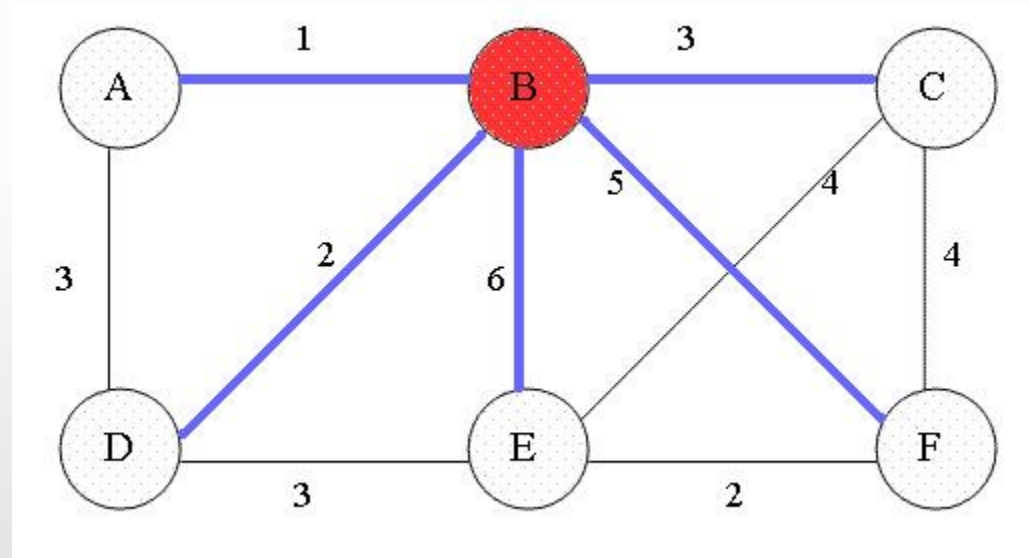
# Prim



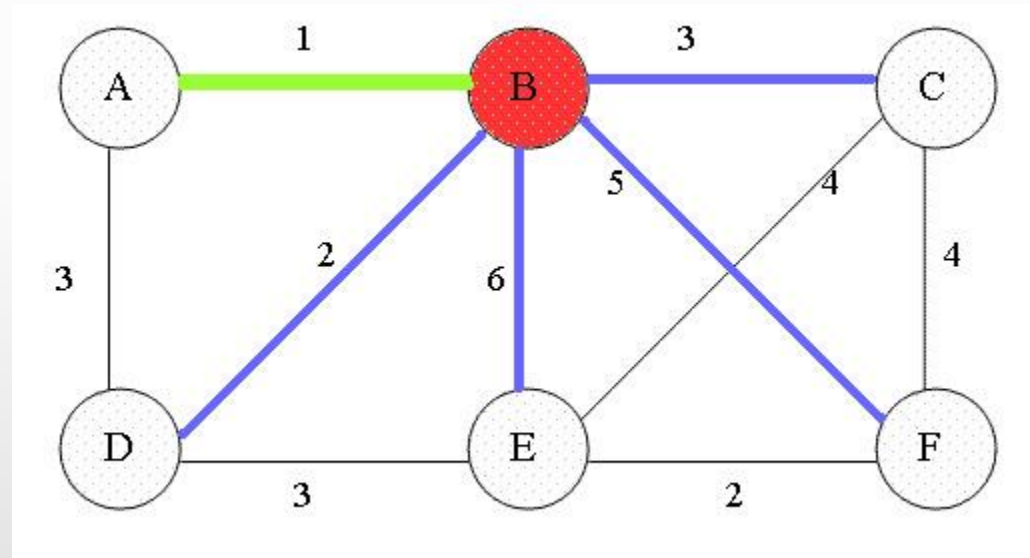
# Prim



# Prim

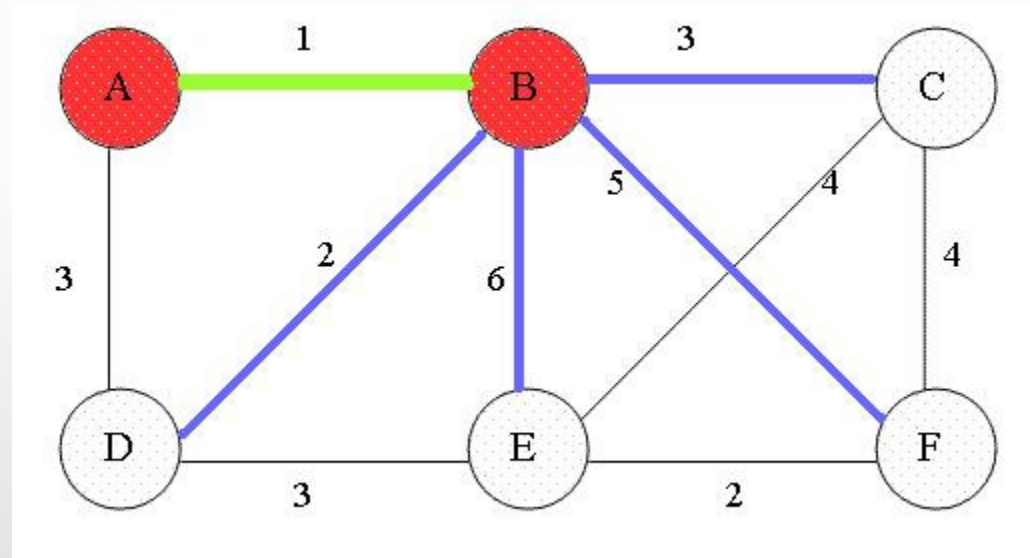


# Prim

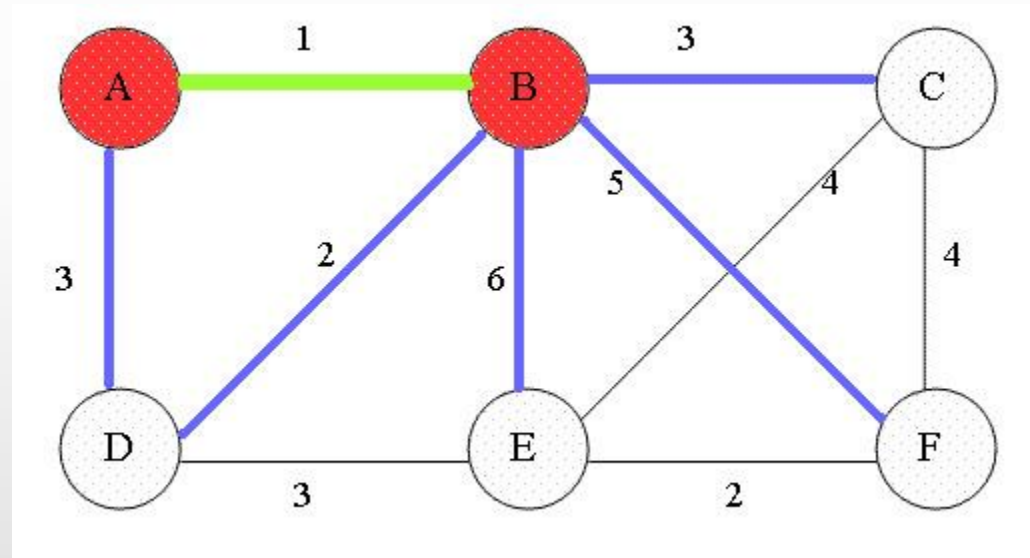




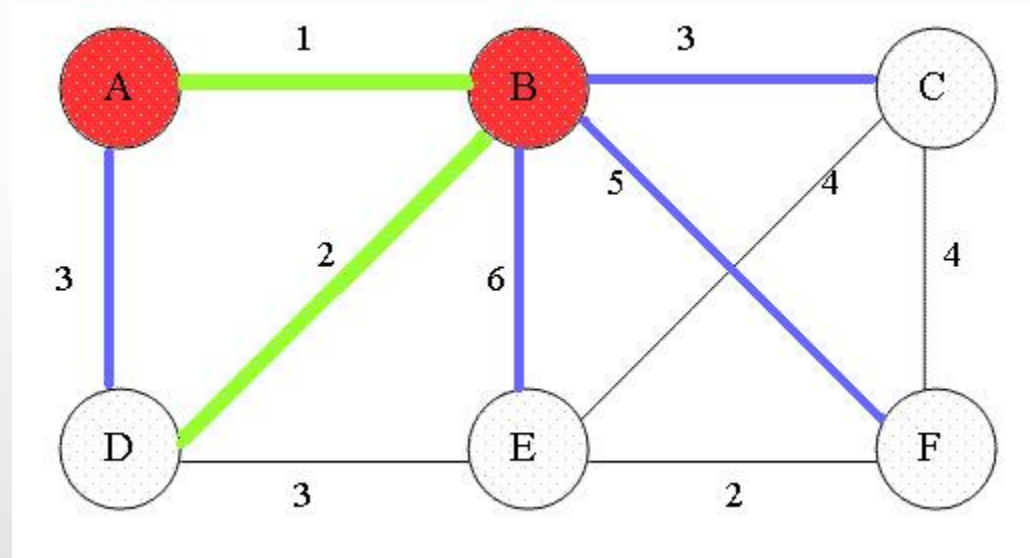
# Prim



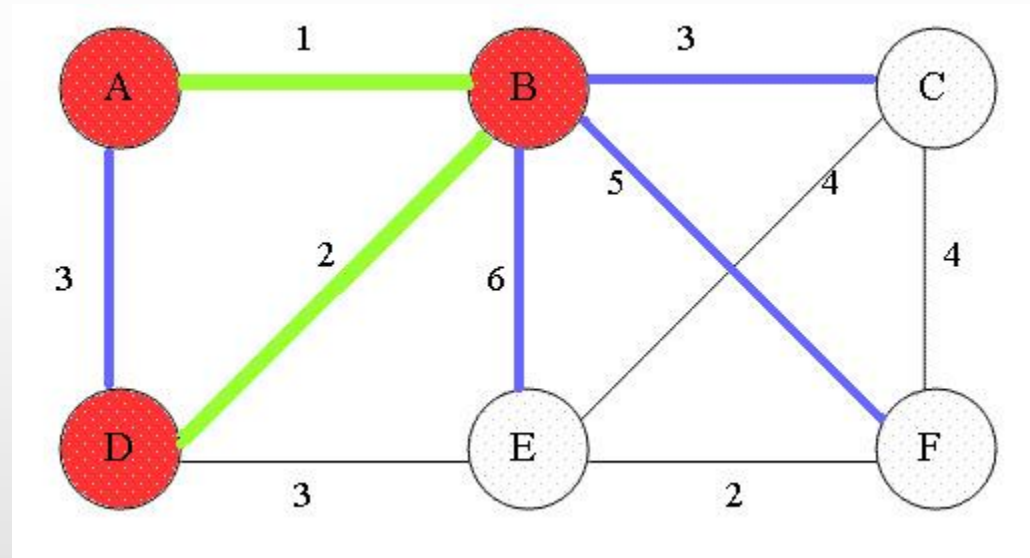
# Prim



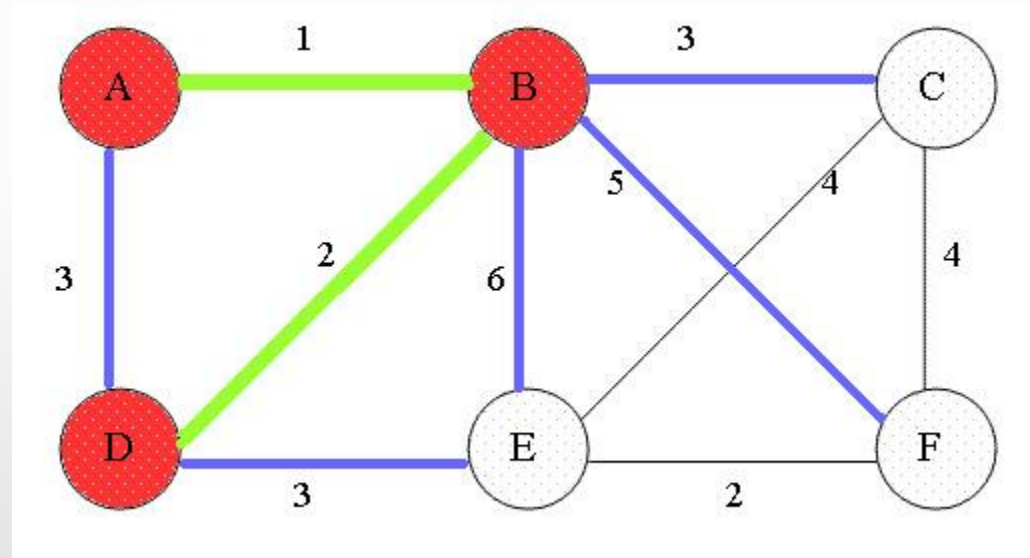
# Prim



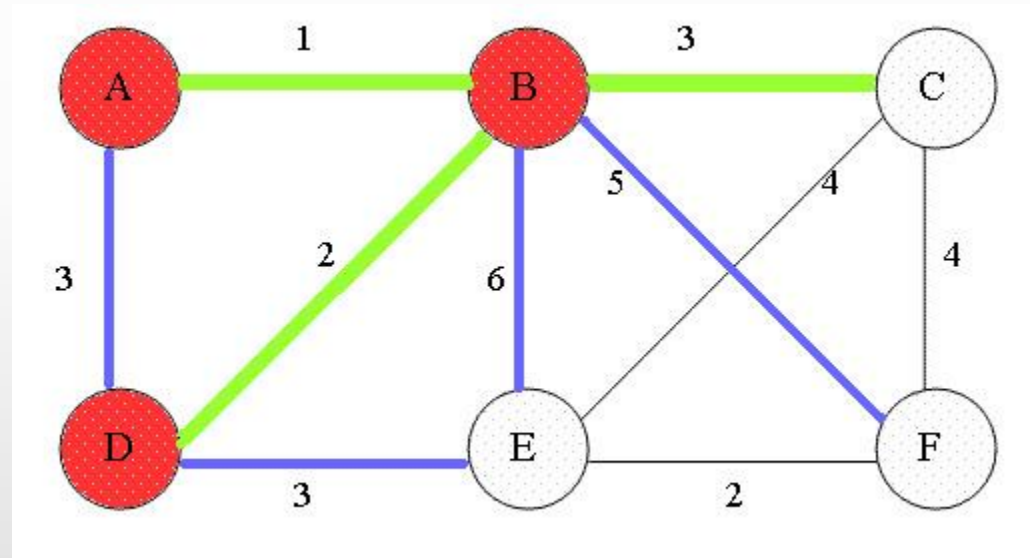
# Prim



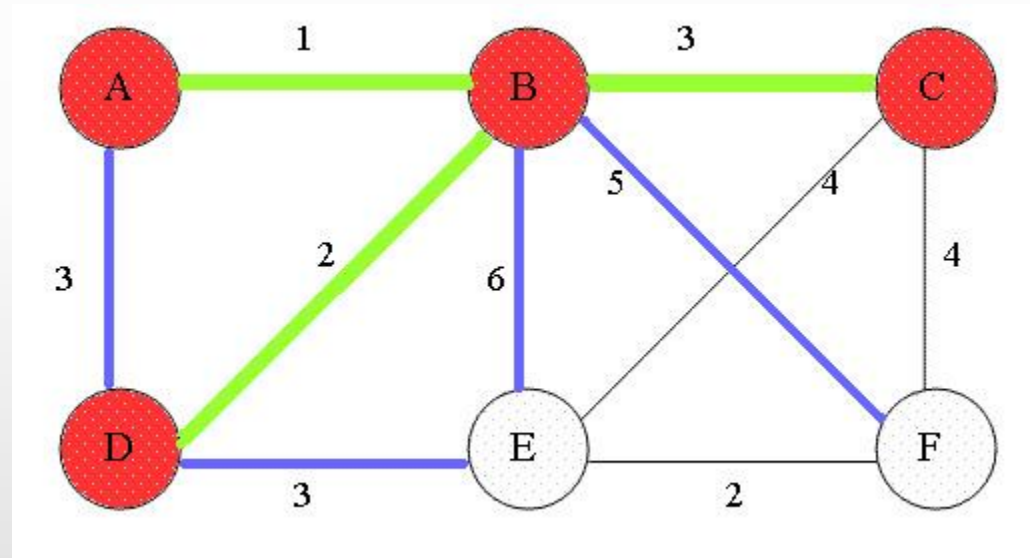
# Prim



# Prim

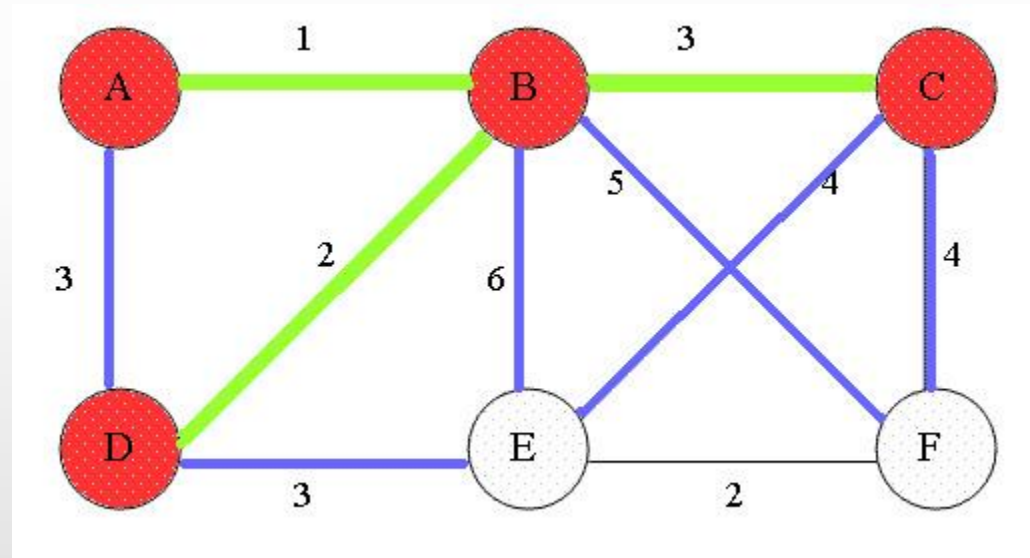


# Prim

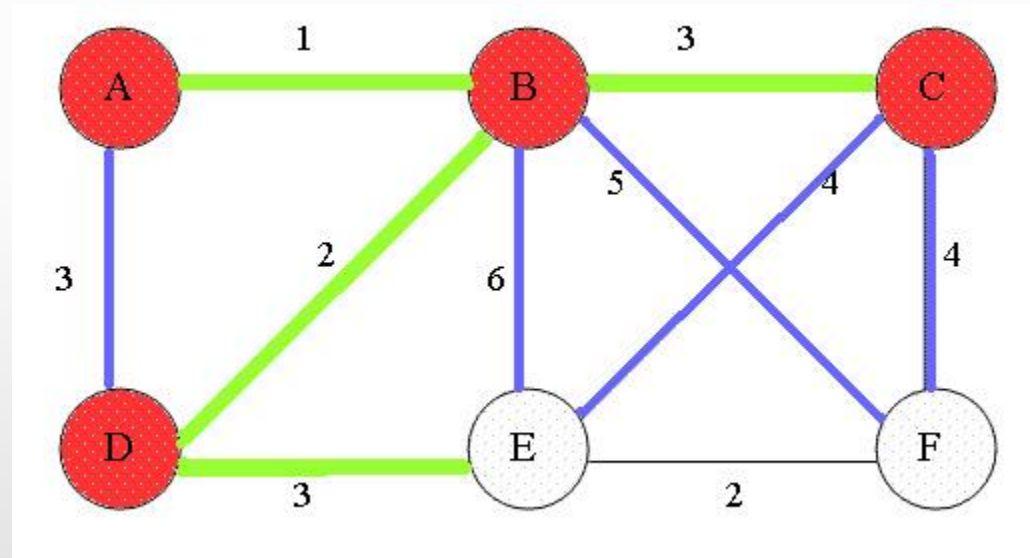




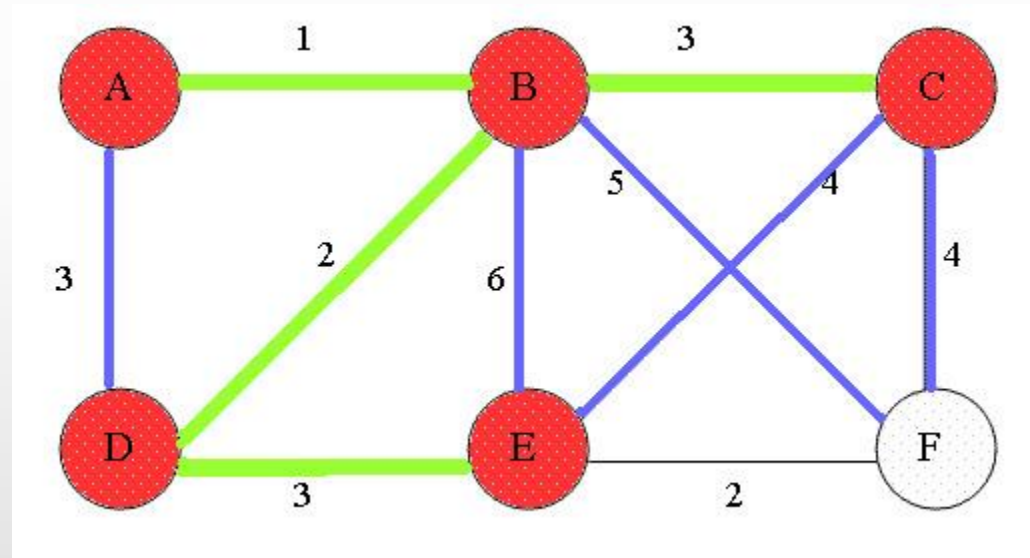
# Prim



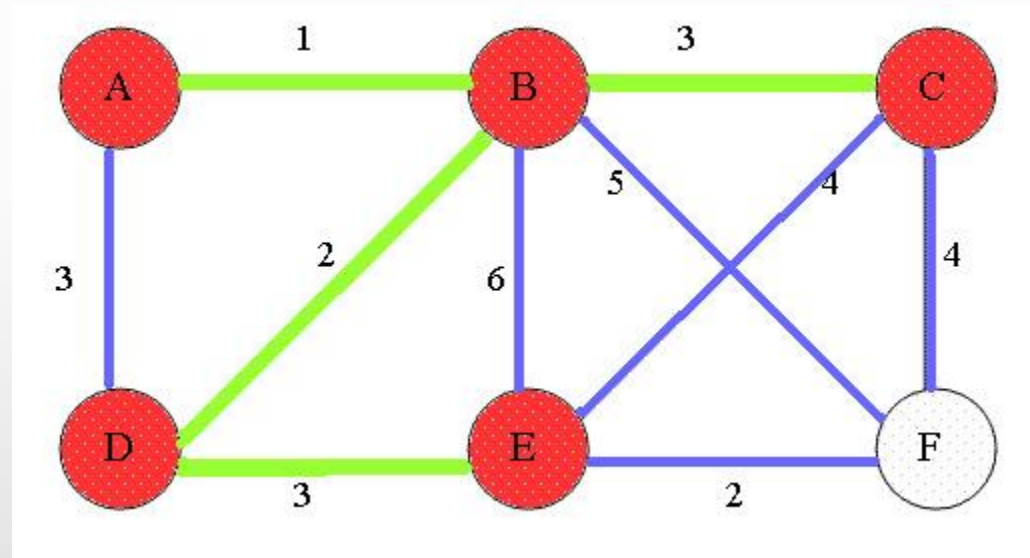
# Prim



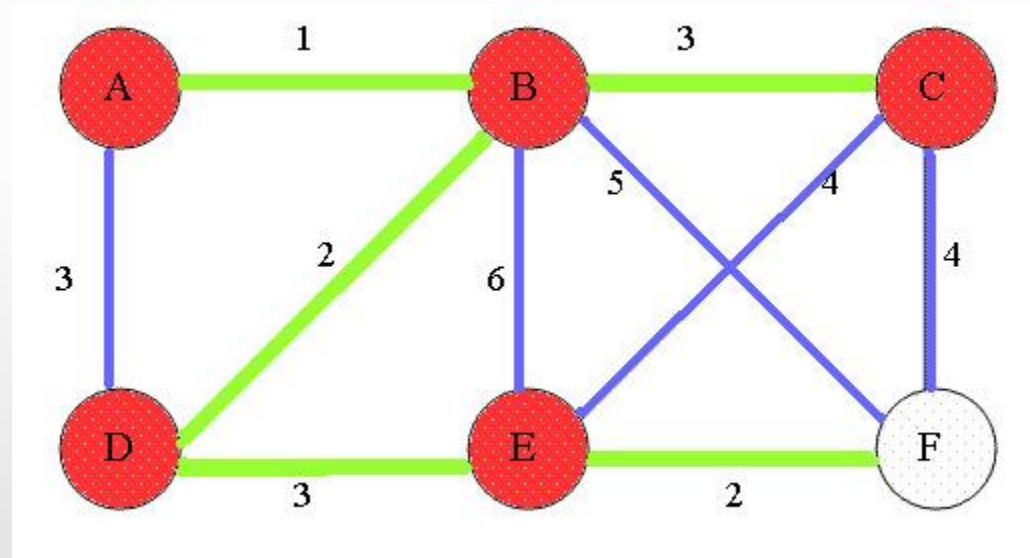
# Prim



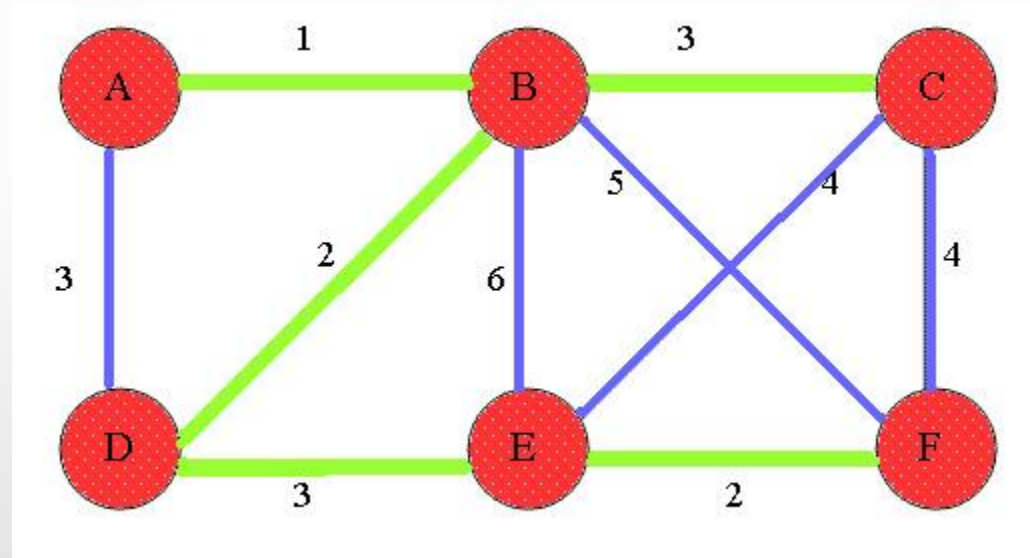
# Prim



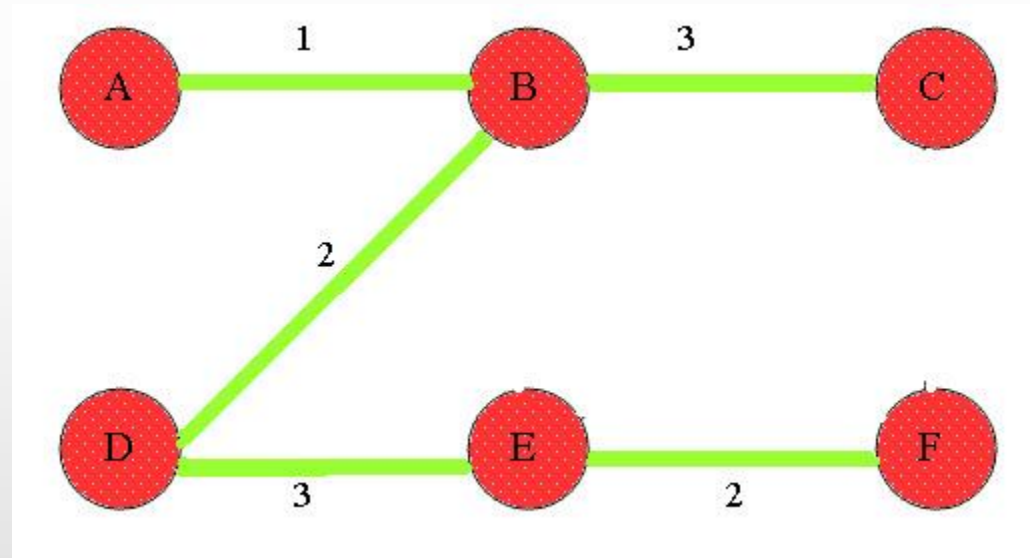
# Prim



# Prim



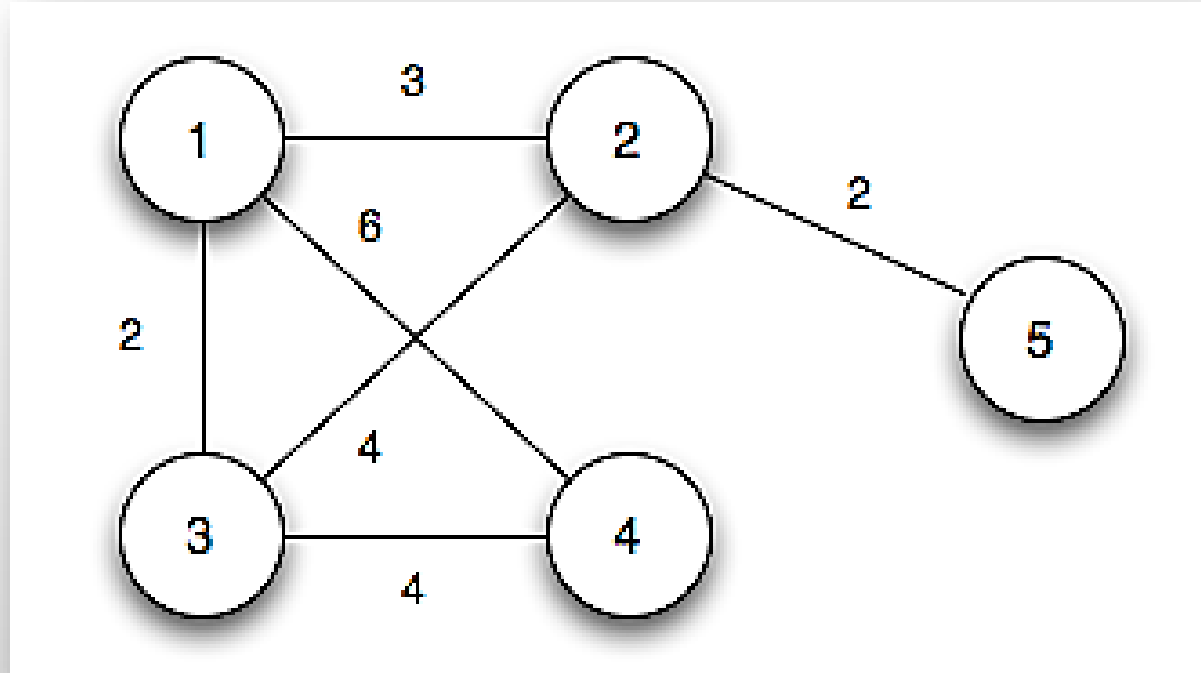
# Prim





# Örnek

- Aşağıdaki yönsüz çizge verilsin.

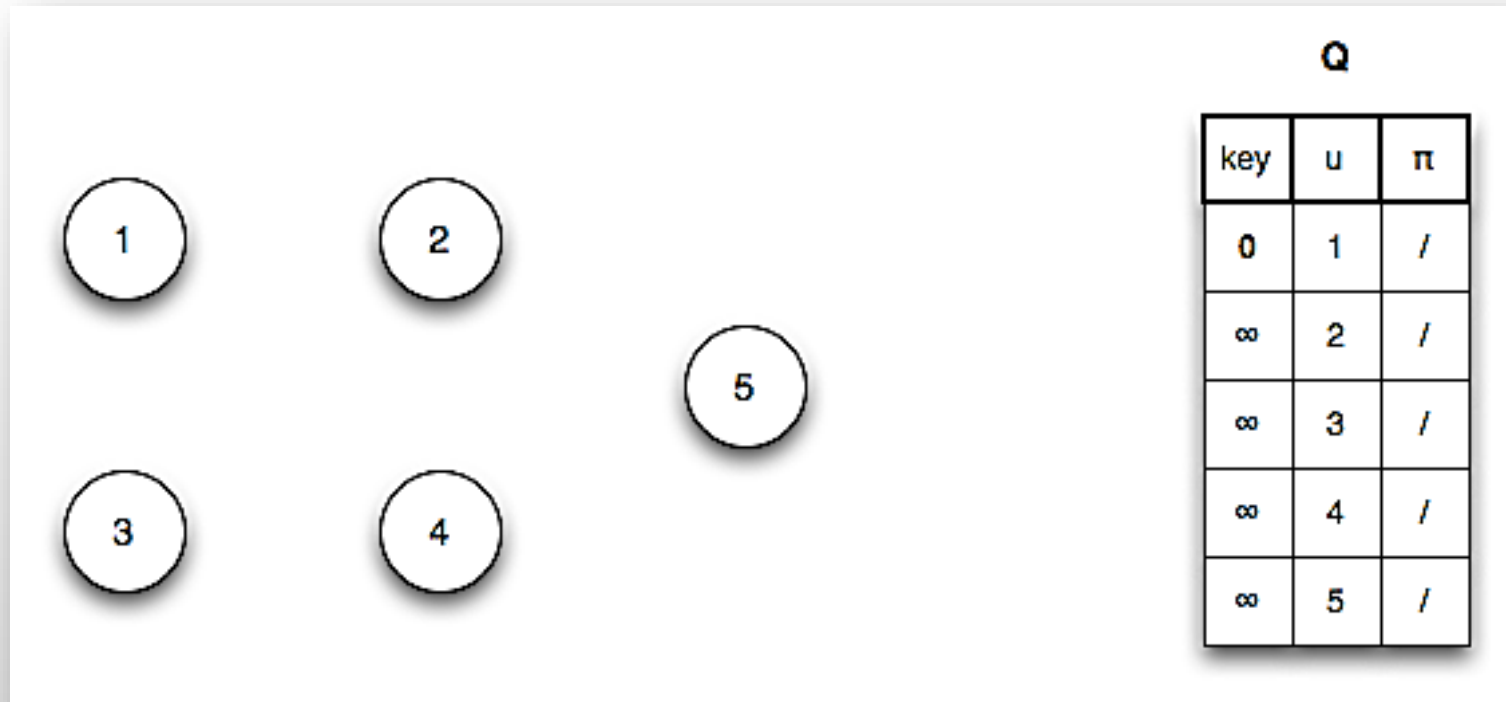






# İklendirme Aşaması

- Düğüm 1 ile kuyruk başlatılır.





# Adım 1

- Kuyruktan düğüm 1'i al, Q'yu güncelle.
- $u3.key = 2$  ((u1,u3)),  $u2.key = 3$  ((u1,u2)),  $u4.key = 6$  ((u1,u4))





## Adım 2

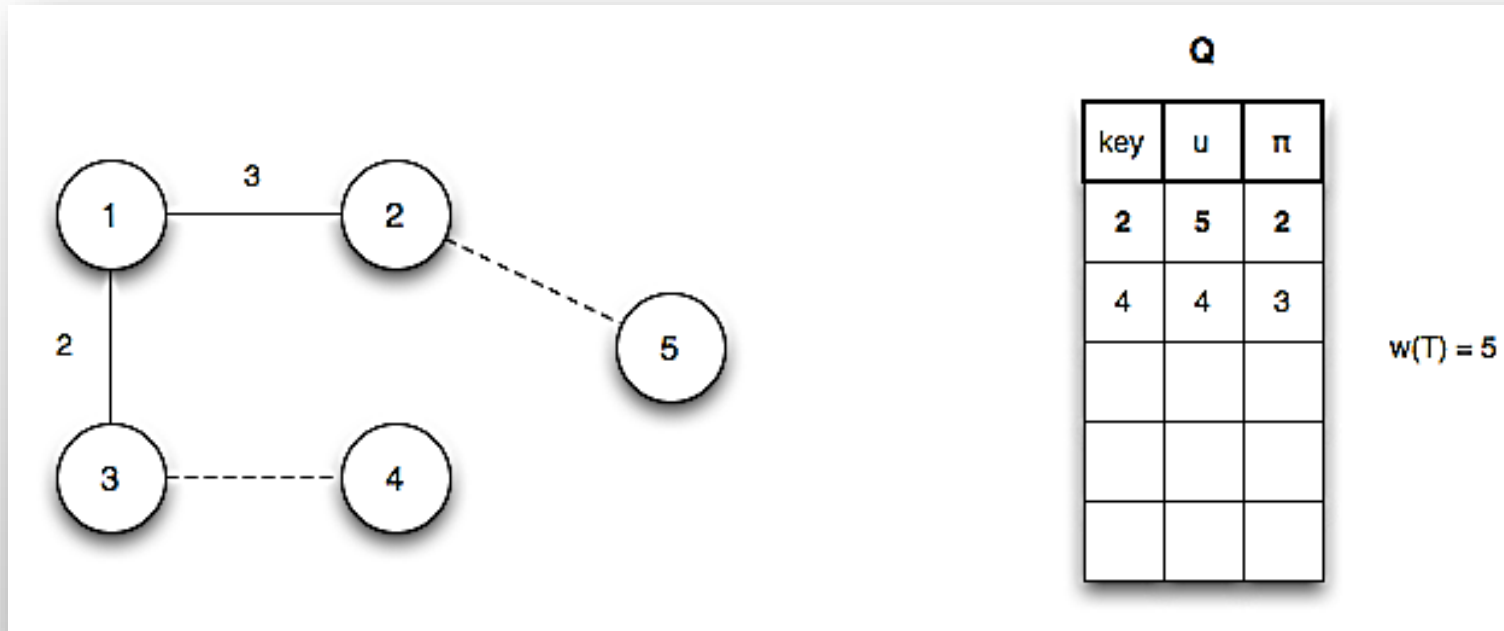
- Kuyruktan düğüm 3'ü al,  $T$ 'ye  $(u_1, u_3)$  kenarını ekle.  $Q$ 'yu güncelle.
- $u4.key = 4$  ( $(u3, u4)$ )





## Adım 3

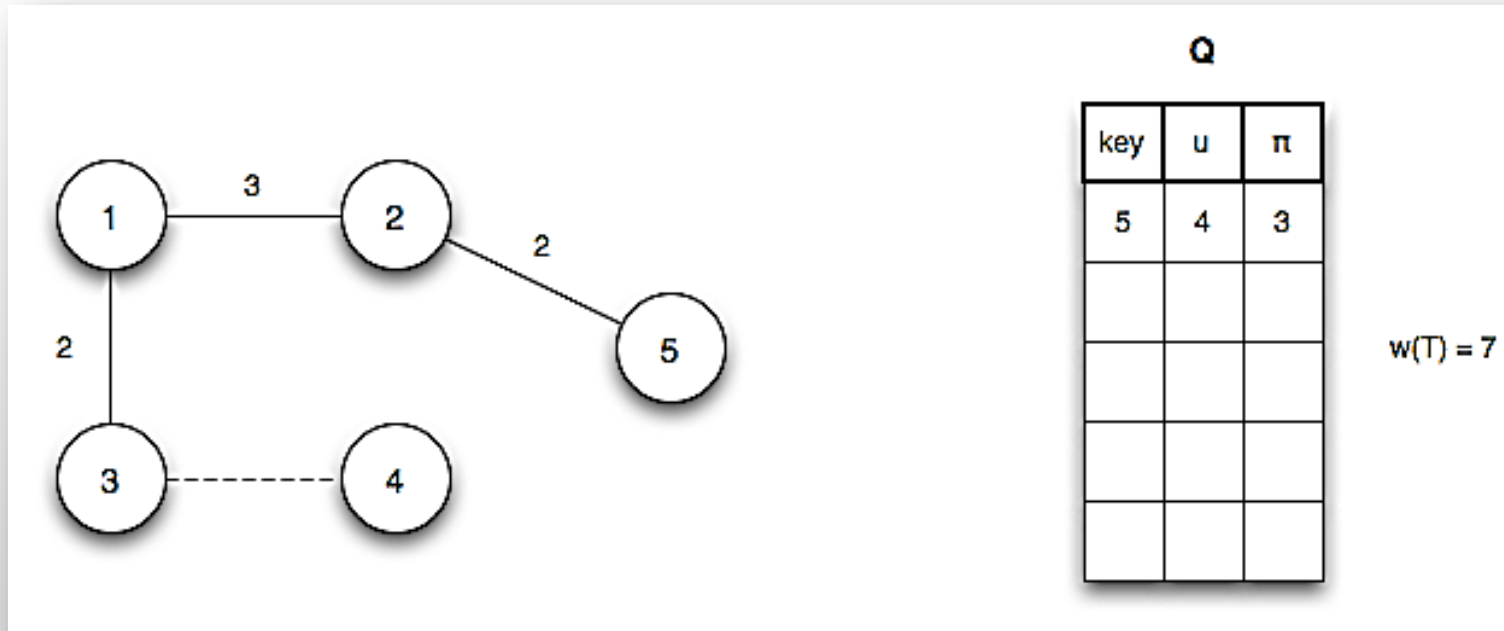
- Kuyruktan düğüm 2'yi al, T'ye  $(u_1, u_2)$  kenarını ekle. Q'yu güncelle.
- $u_5.key = 2$   $((u_2, u_5))$





## Adım 4

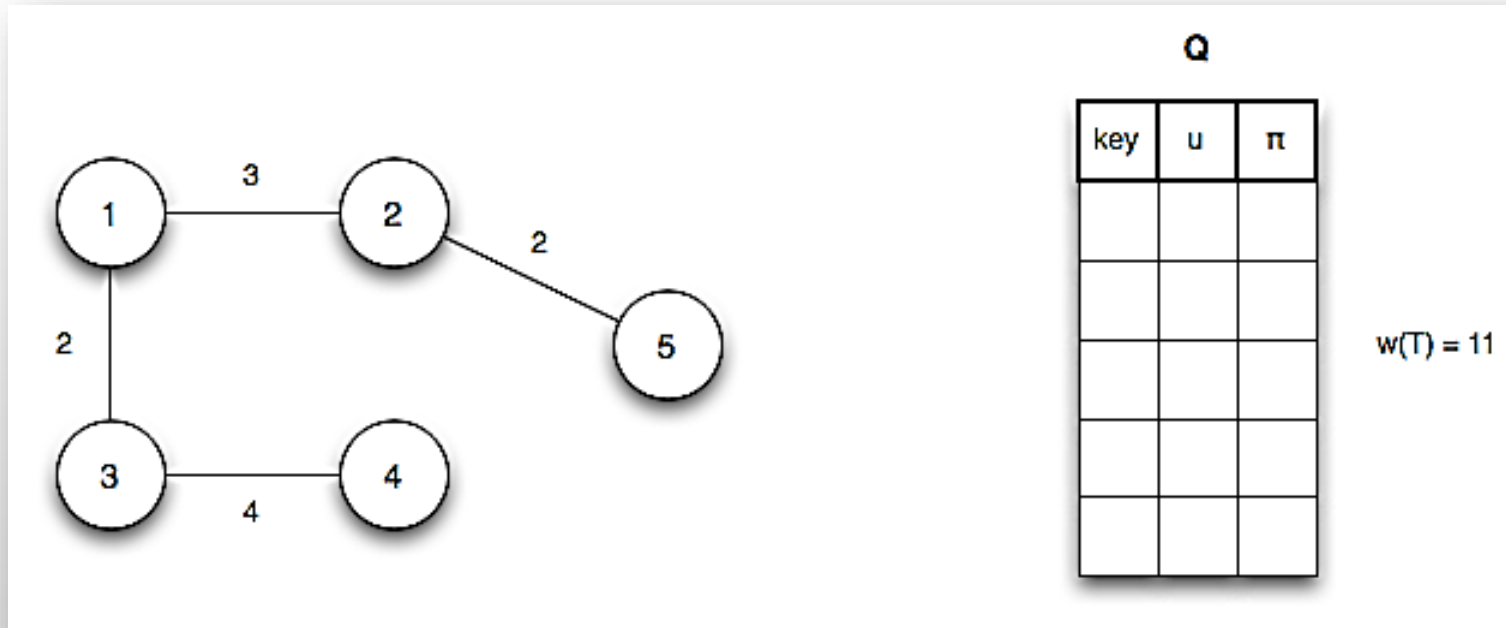
- Kuyruktan düğüm 5'i al, T'ye (u2,u5) kenarını ekle. Q'da güncelleme yok.





## Adım 5

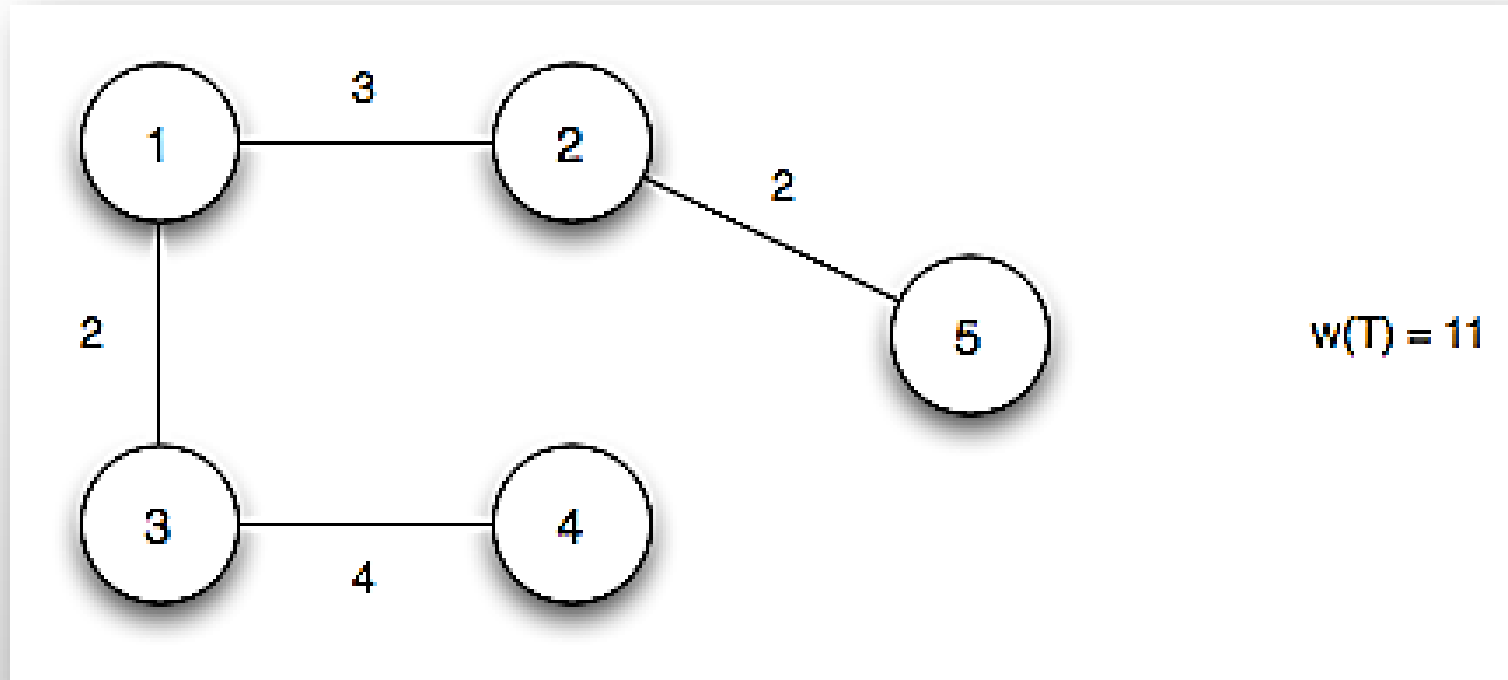
- Kuyruktan düğüm 4'ü al, T'ye (u3,u4) kenarını ekle. Q'da güncelleme yok.





# Son Durum

- $Q = \emptyset$





SON