



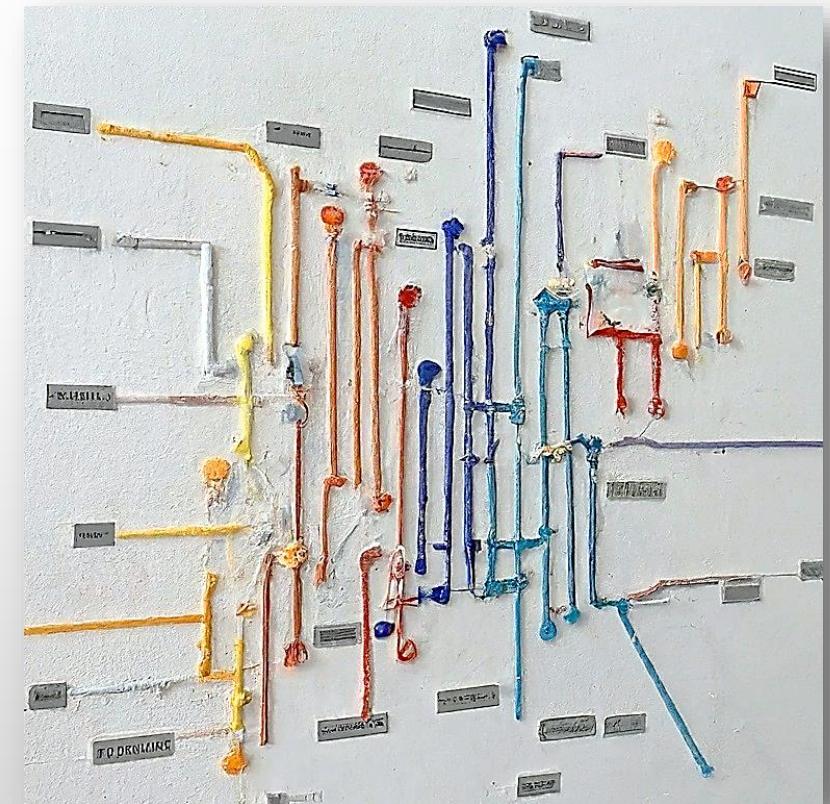
Bölüm 4: Çizge Algoritmaları

Algoritmalar

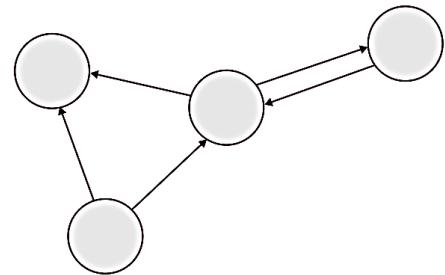


Çizge Algoritmaları

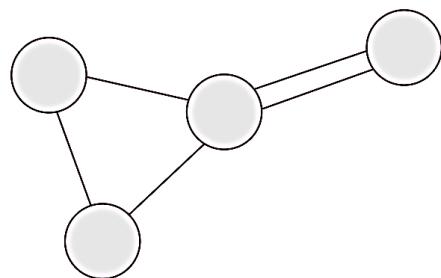
- Dünya aslında bir ağ gibidir.
 - Şehirler yollarla,
 - İnsanlar ilişkilerle,
 - Bilgisayarlar kablolarla birbirine bağlıdır.
- Çizge algoritmaları bu ağları inceler ve anlamlandırır.



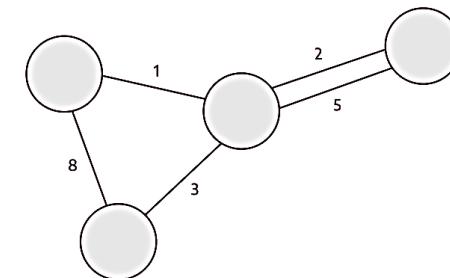
Çizge Türleri



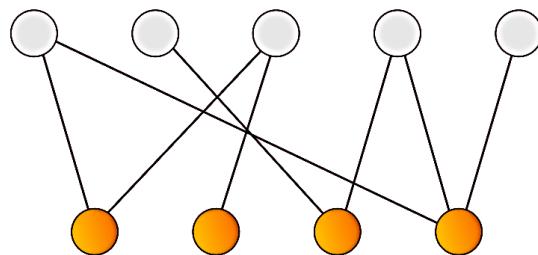
Directed graph



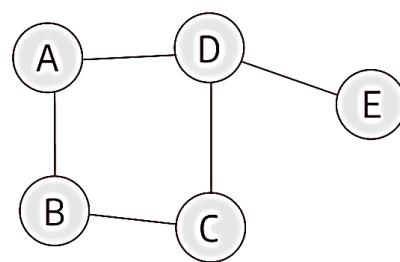
Undirected



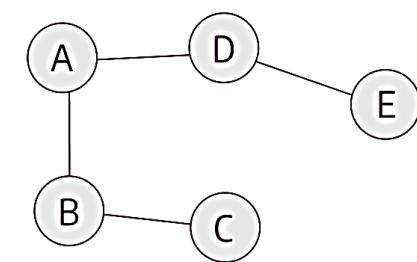
Weighted



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 şehrre 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*)

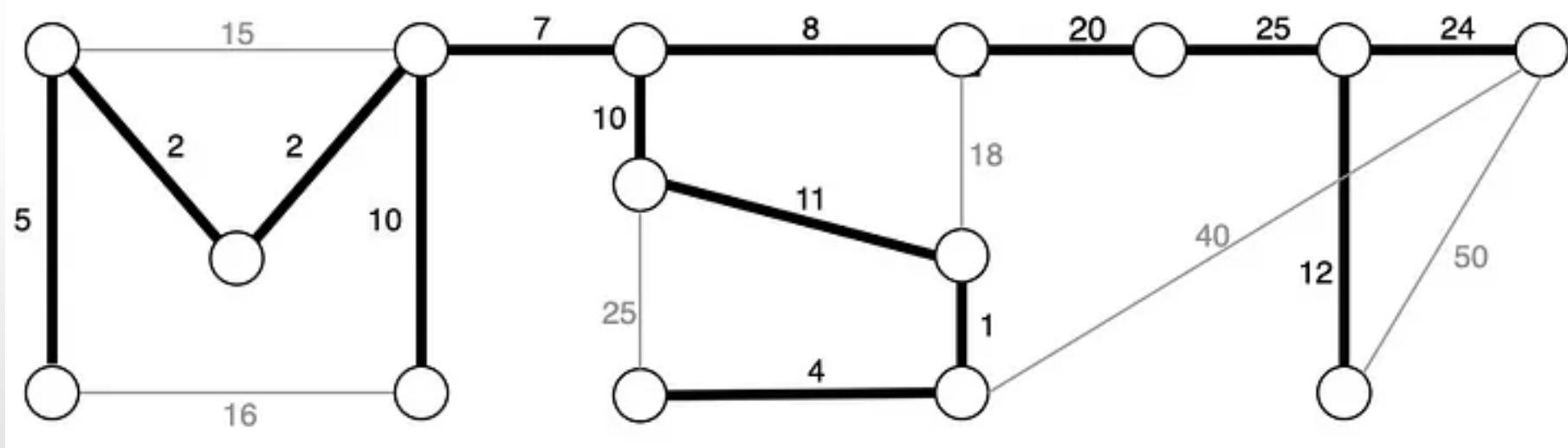


Minimum Kapsayan Ağaç Algoritmaları

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



Minimum Kapsayan Ağaç (MST)





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 oluşturmadan, 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.

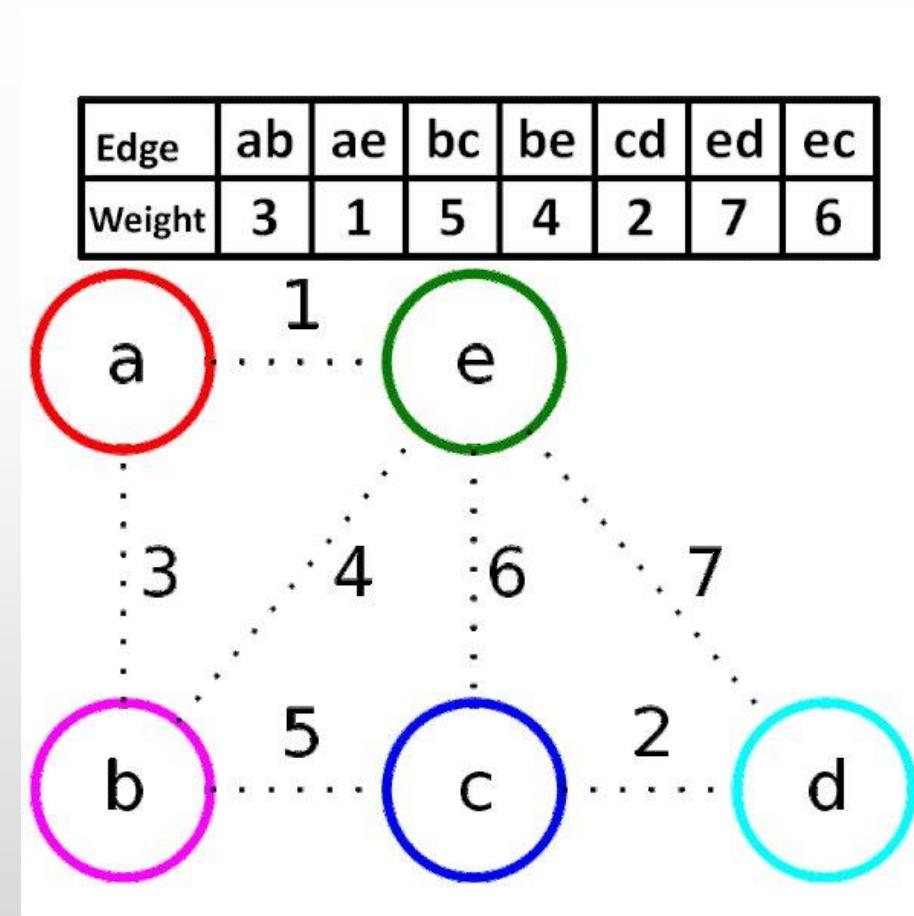


Karmaşıklık Analizi

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

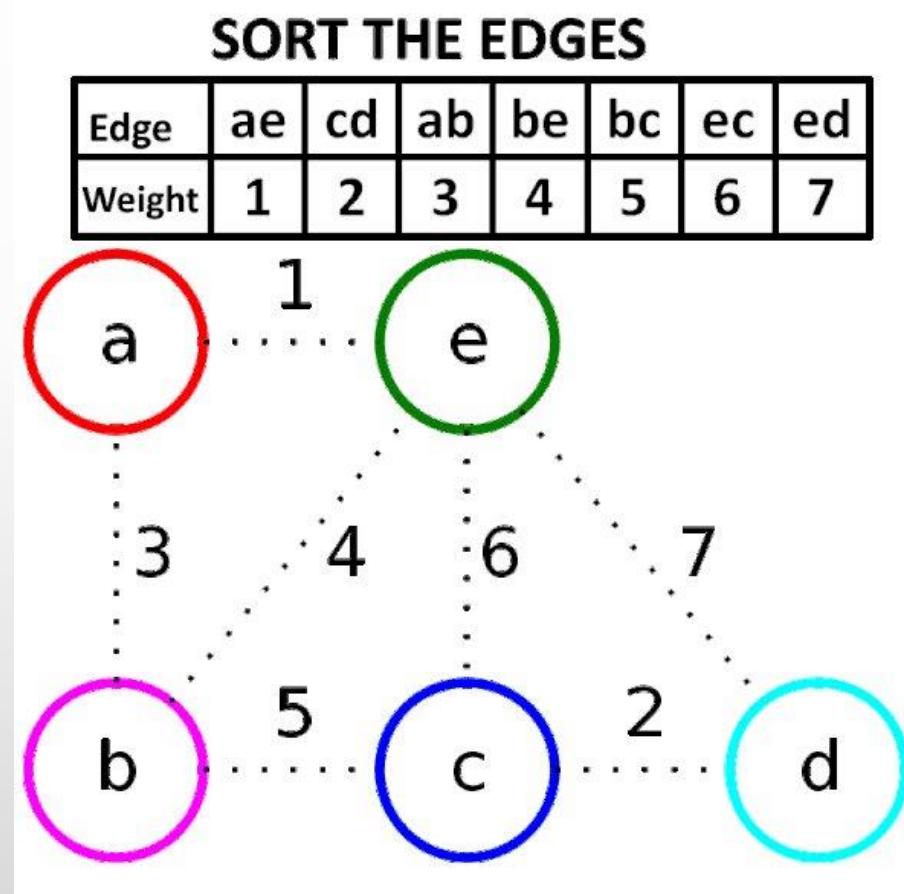


Kruskal



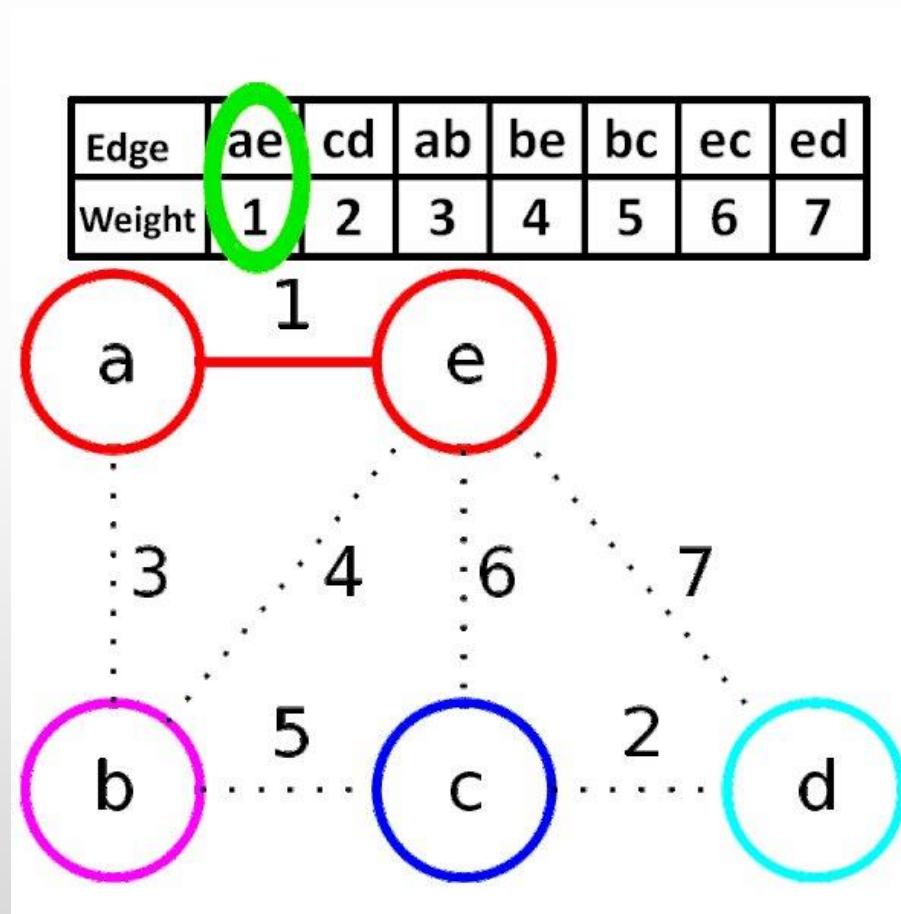


Kruskal



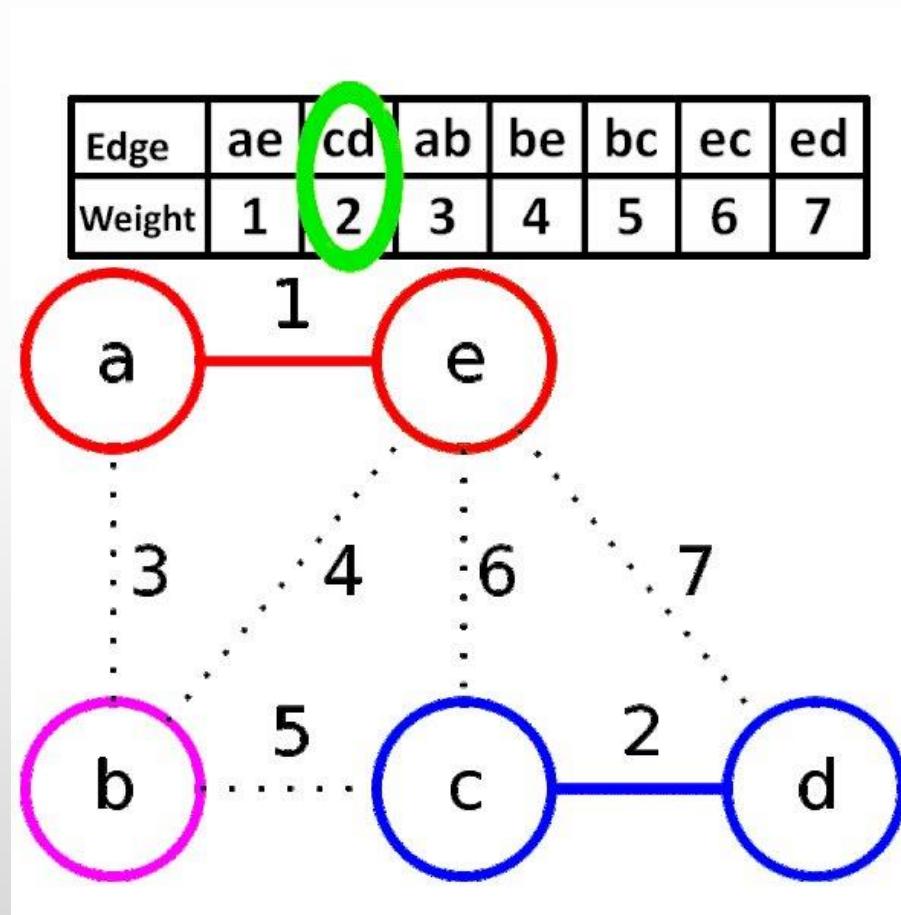


Kruskal



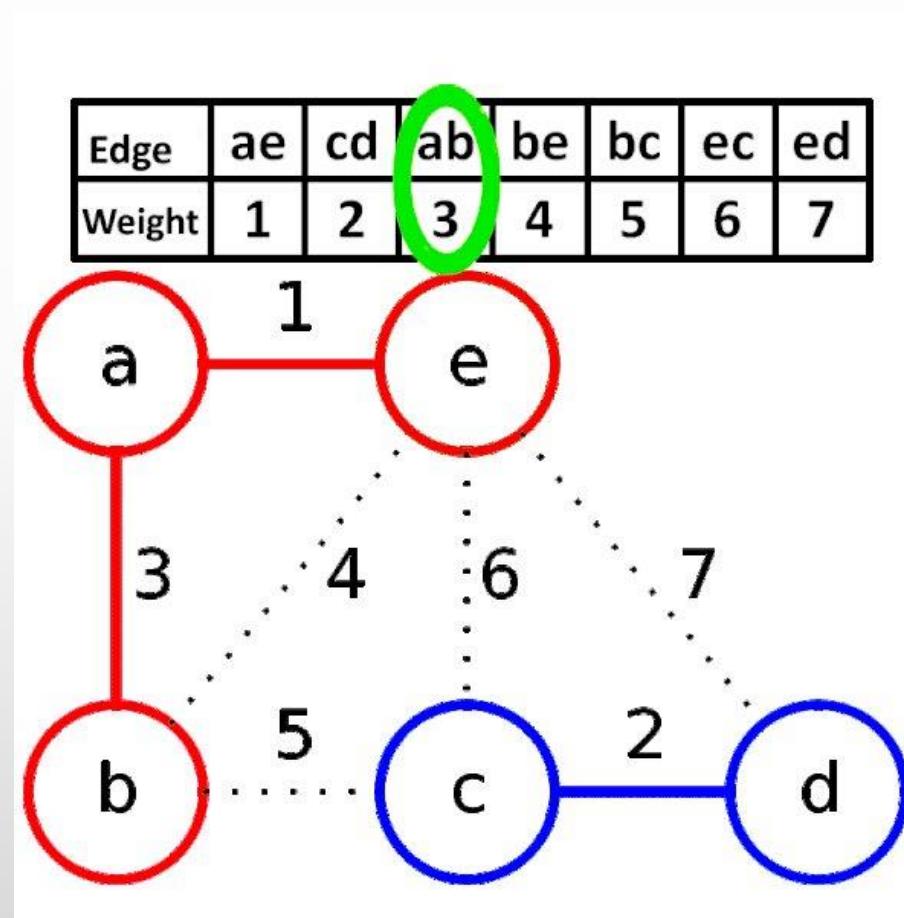


Kruskal



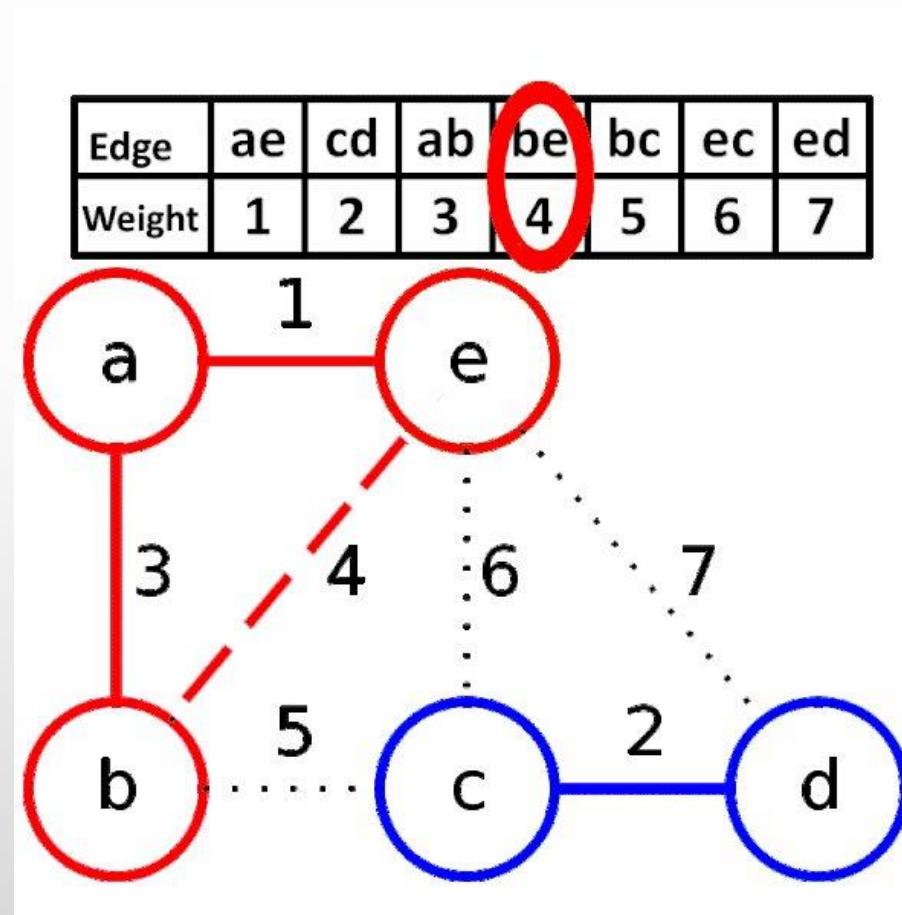


Kruskal



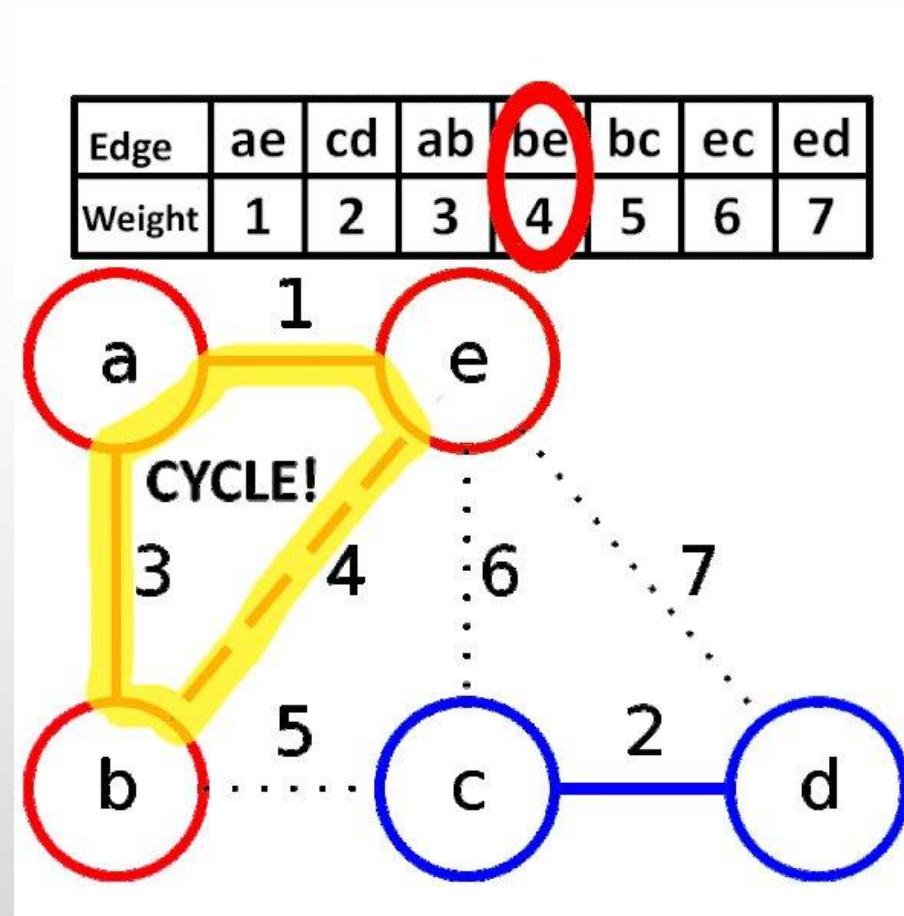


Kruskal



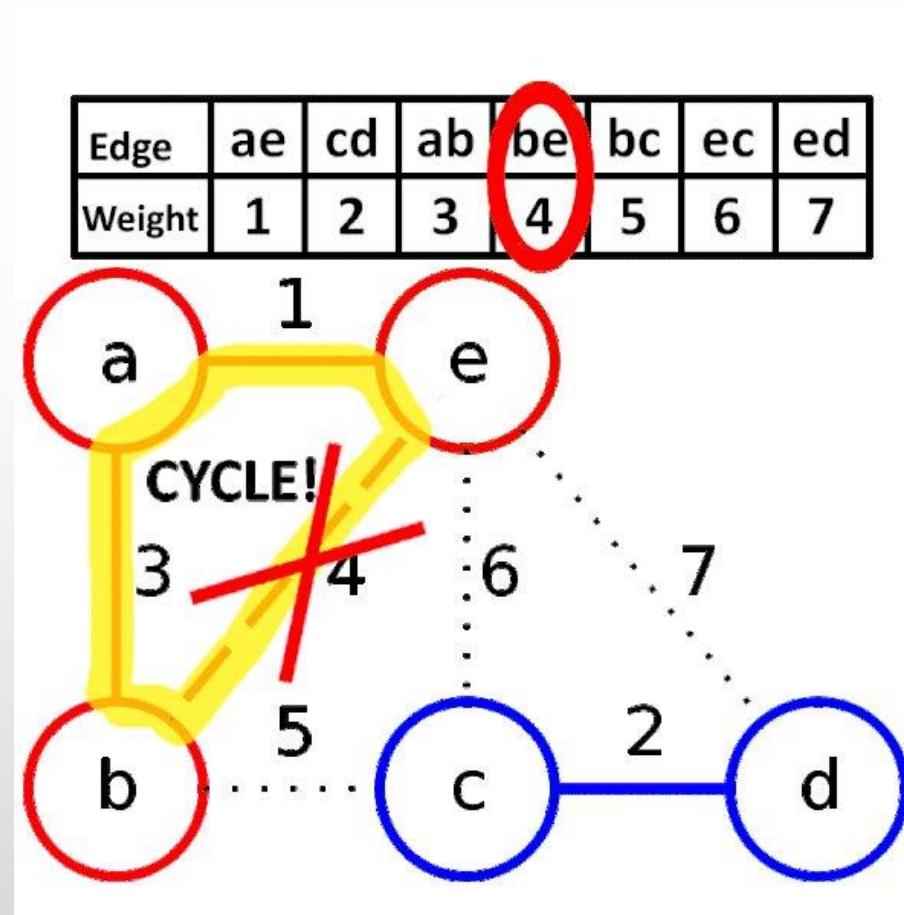


Kruskal



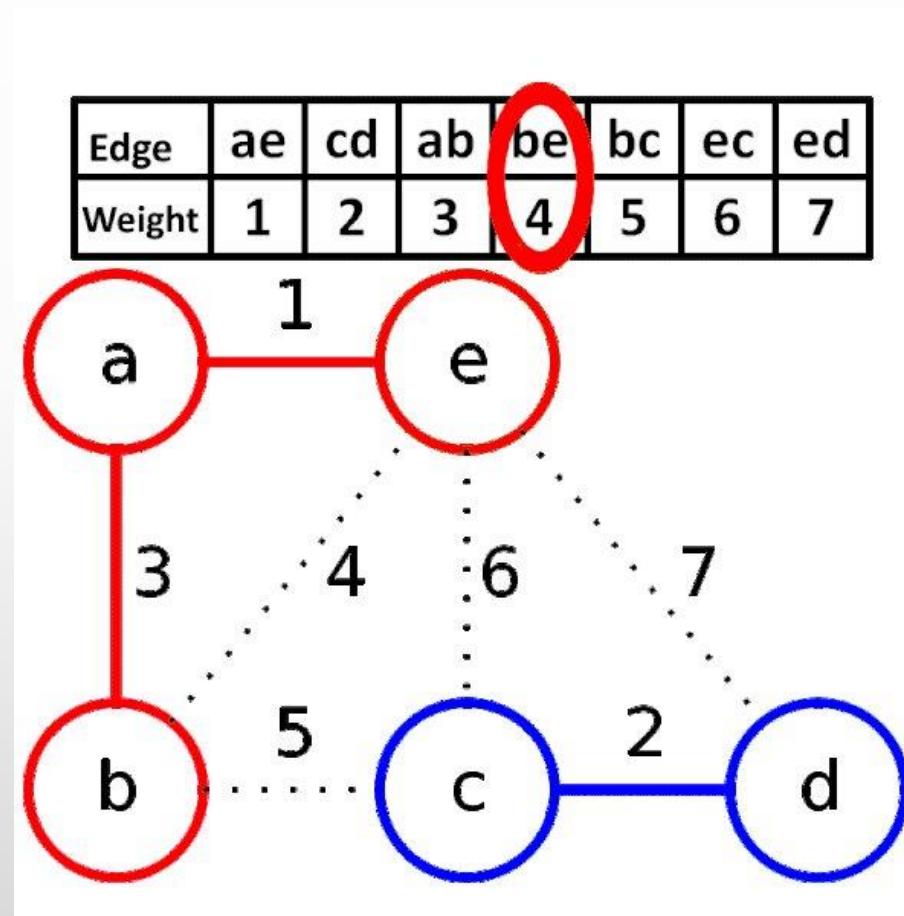


Kruskal



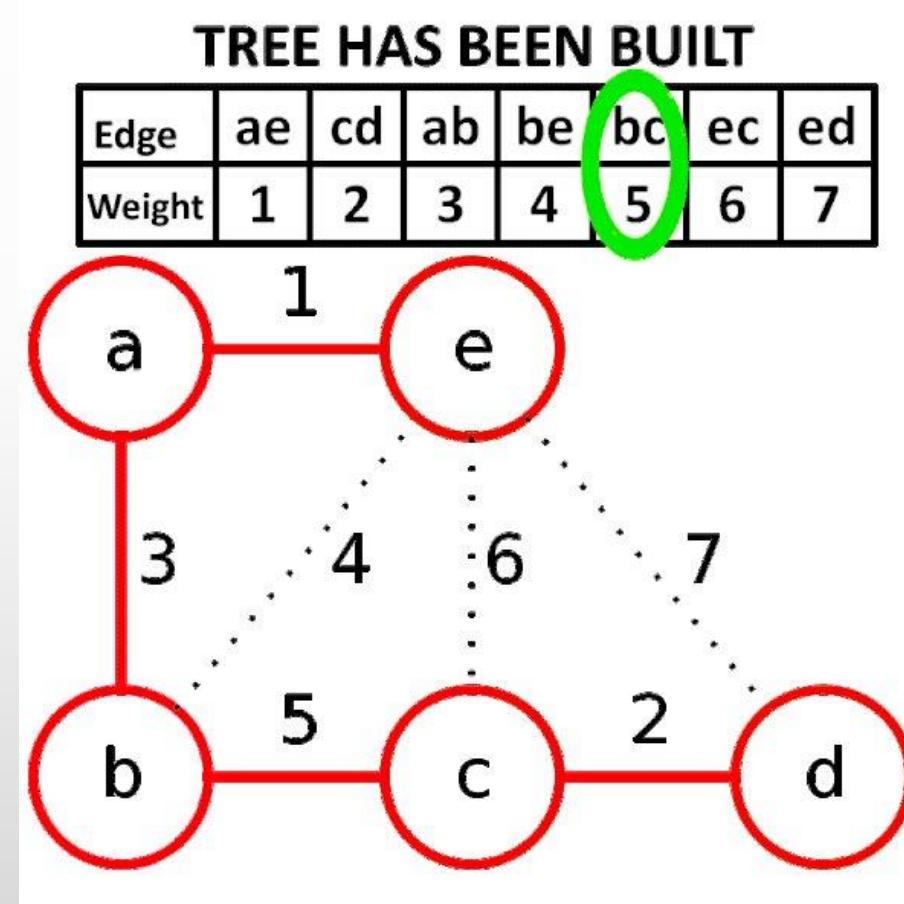


Kruskal



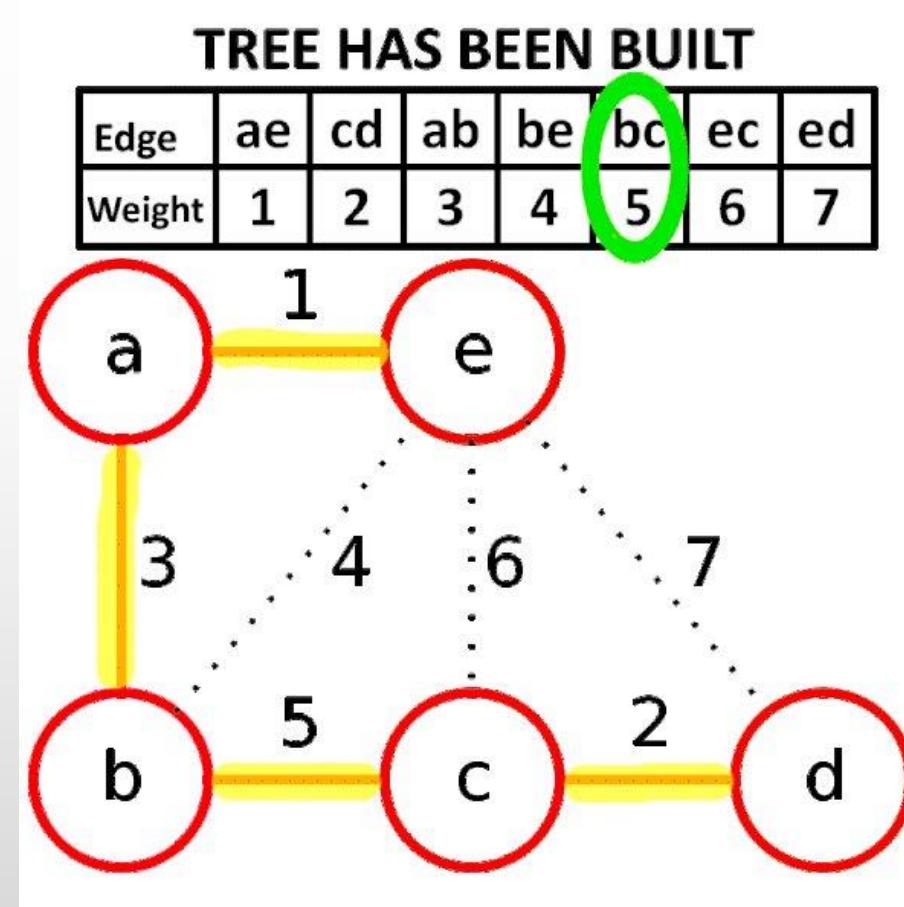


Kruskal





Kruskal

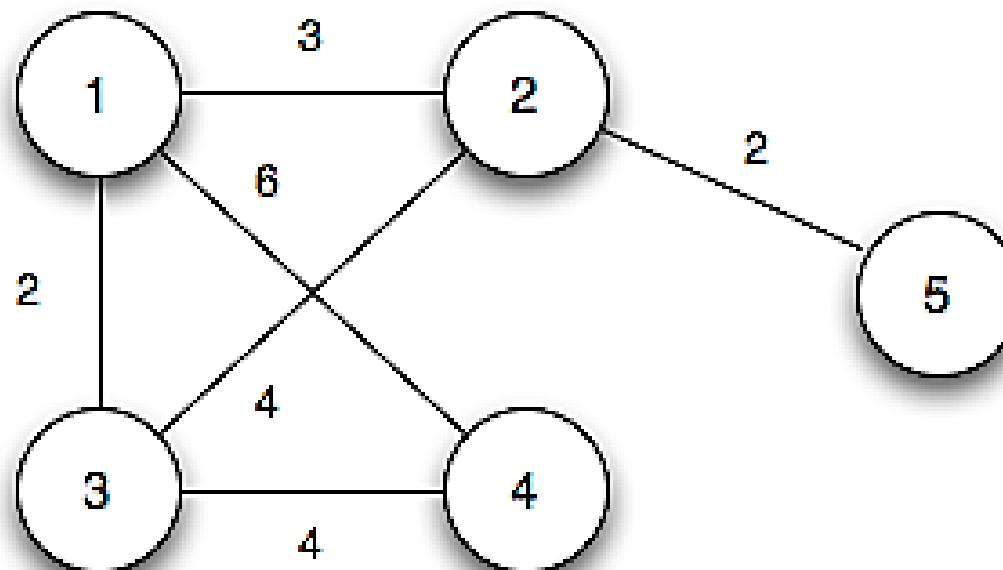






Örnek

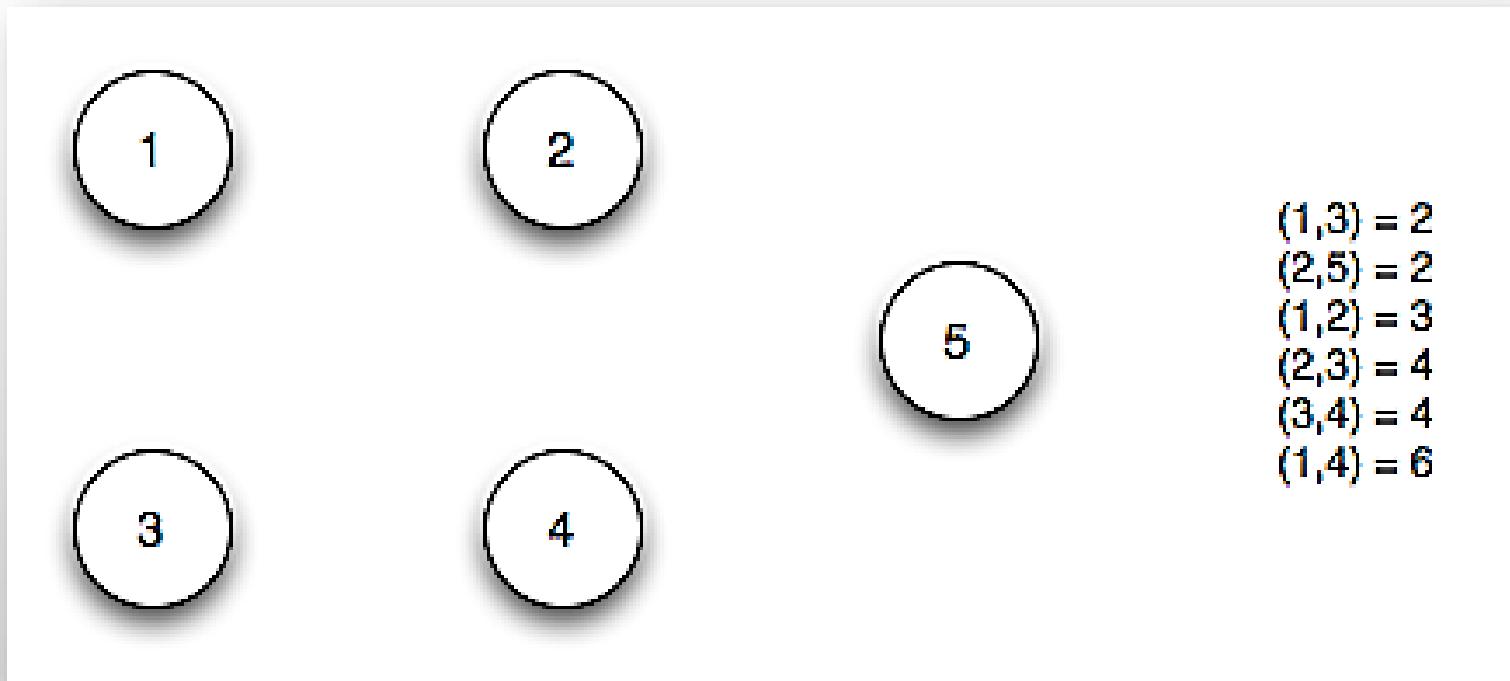
- Aşağıdaki yönsüz çizge verilsin.





İlkendirme Aşaması

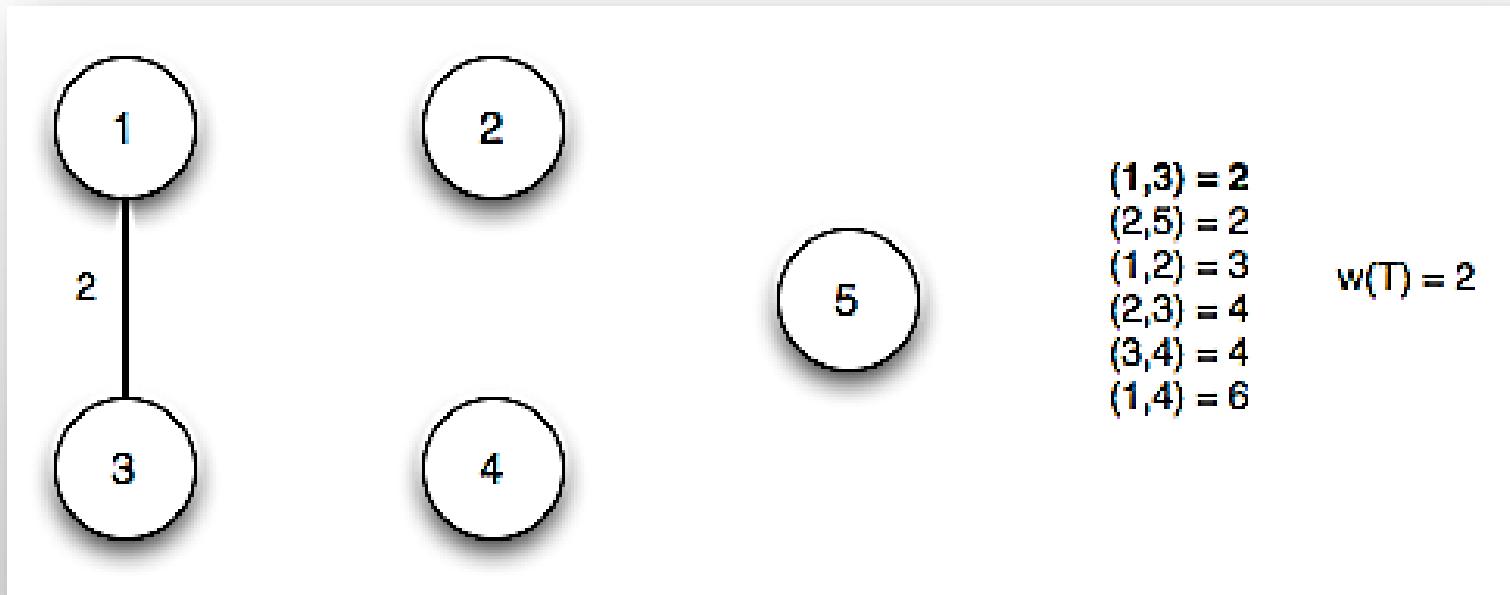
- Her bir düğüm ayrı bir ağaç yapılır.
- Kenarlar ağırlıklarına göre sıralanır.





Adım 1

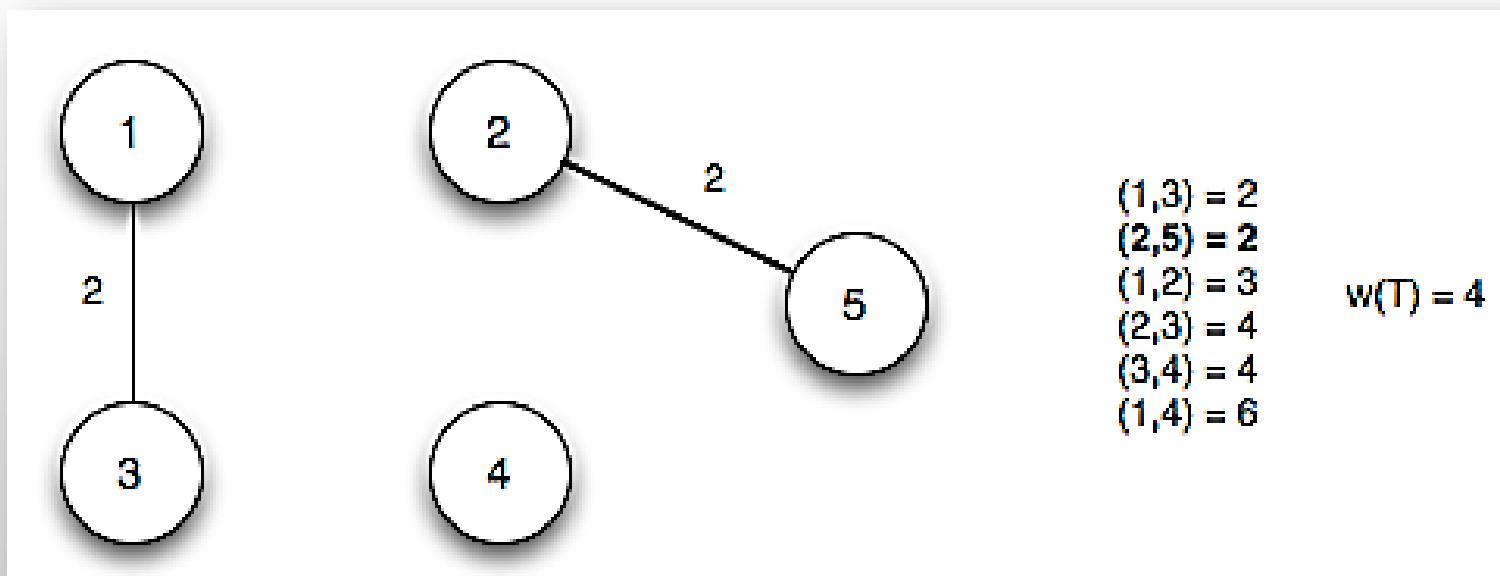
- (u_1, u_3) kenarını ekle
- v_1 ve v_3 birleştir





Adım 2

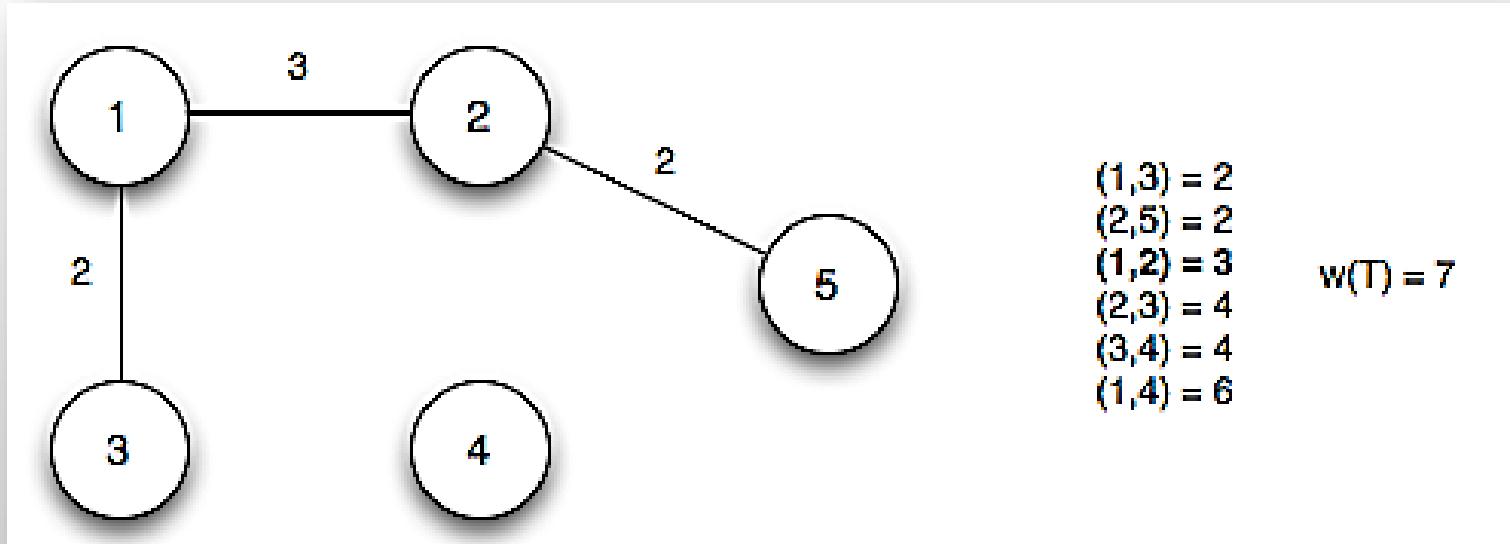
- (u_2, u_5) kenarını ekle.
- v_2 ve v_5 birleştir.





Adım 3

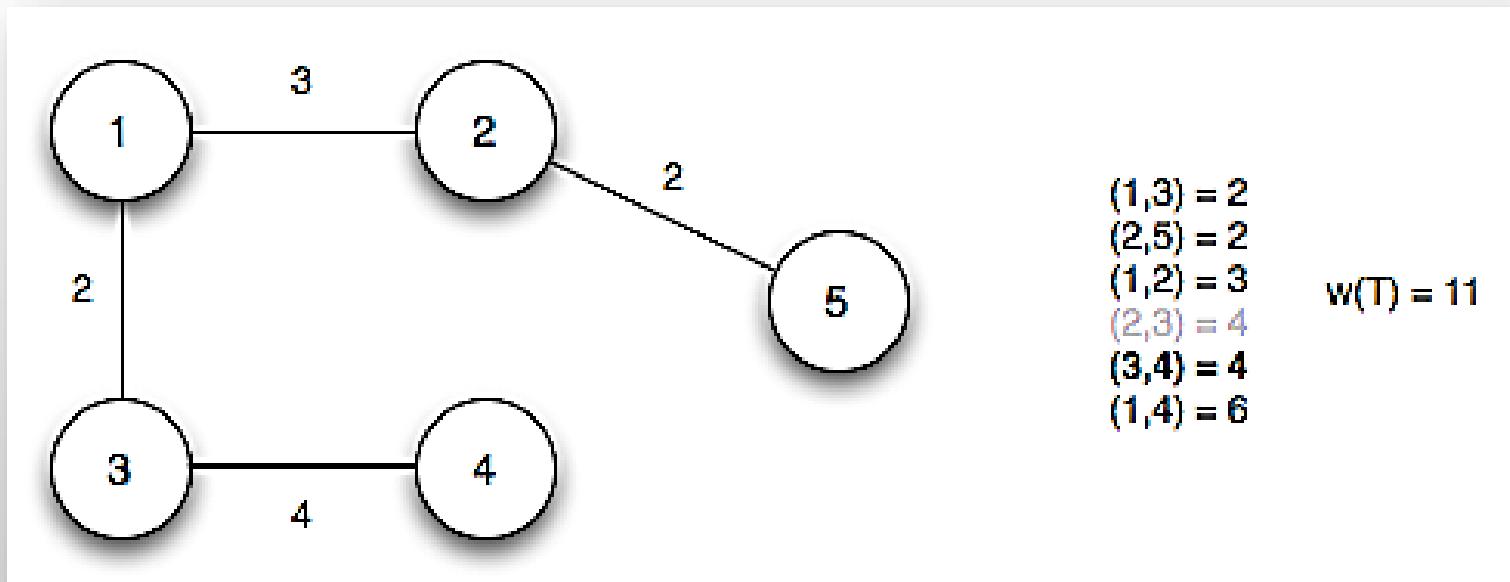
- (u_1, u_2) kenarını ekle.
- Adım 1 ve Adım 2'de oluşan iki ağaçtı birleştir.





Adım 4

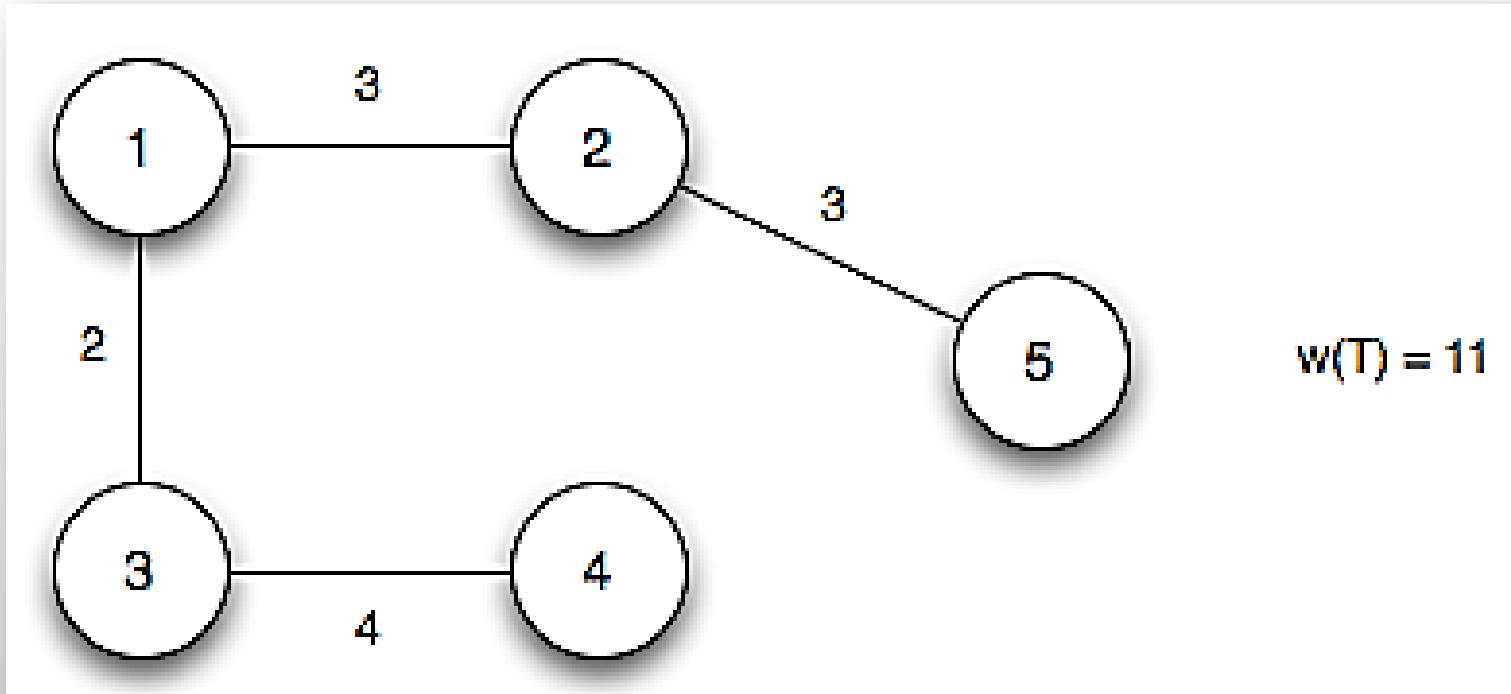
- (u_3, u_4) kenarını ekle, v_4 birleştirir.
- (u_2, u_3) ayrı ağaçları birleştirmiyor!





Son Durum

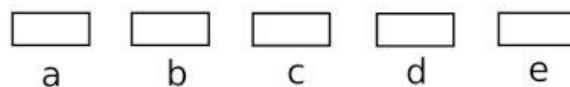
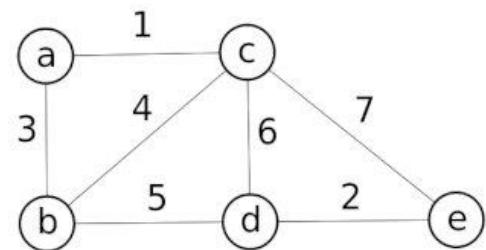
- (u_1, u_4) kenarı ayrı ağaçları birleştirmiyor!





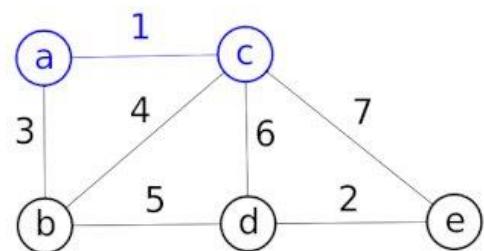


Union Find

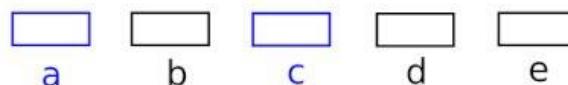




Union Find

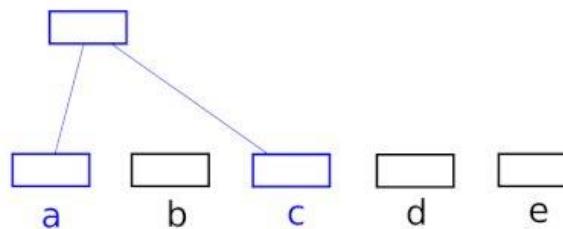
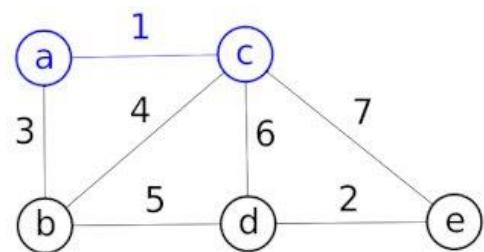


Disjoint!



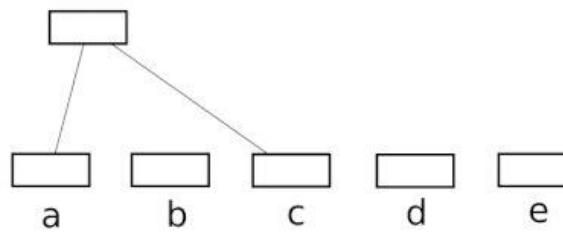
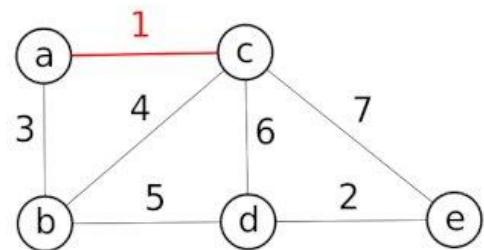


Union Find



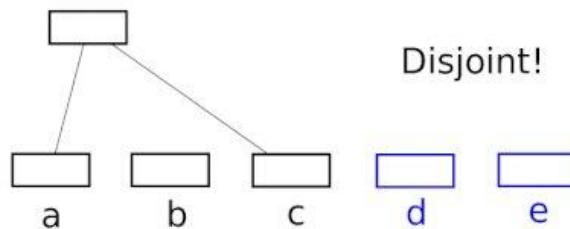
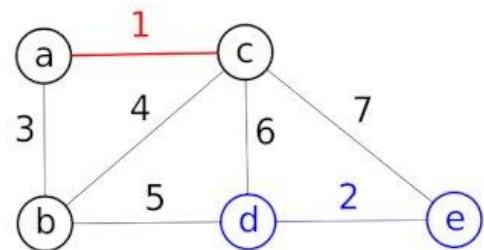


Union Find



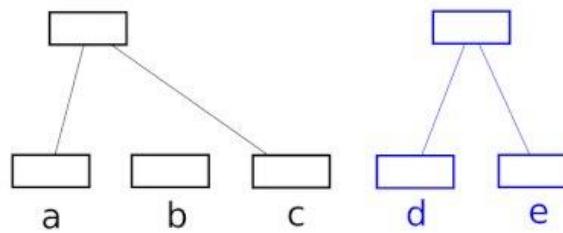
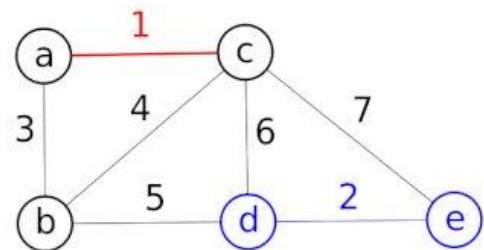


Union Find



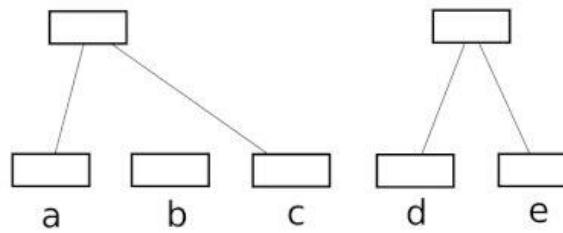
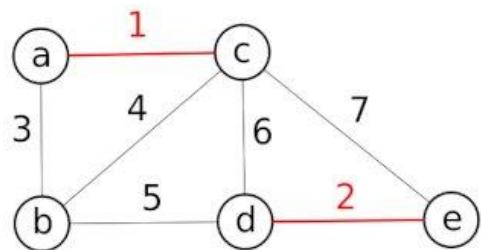


Union Find



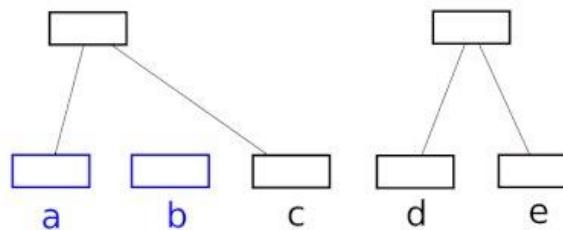
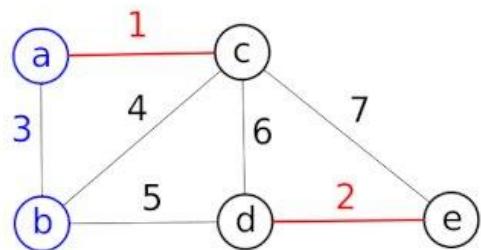


Union Find



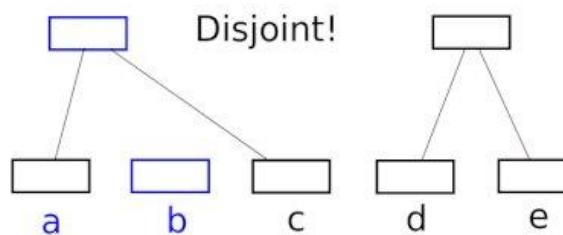
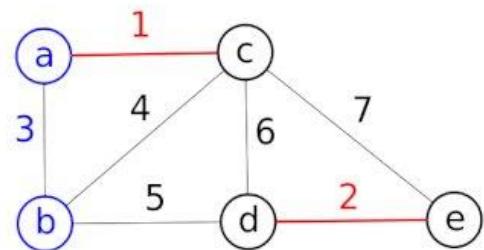


Union Find



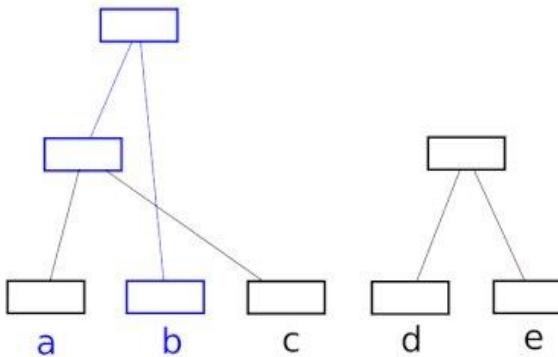
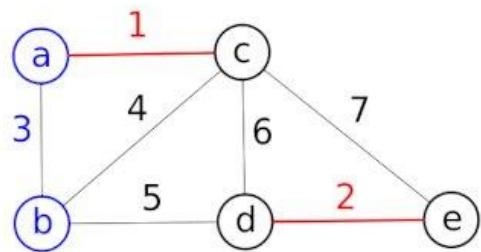


Union Find



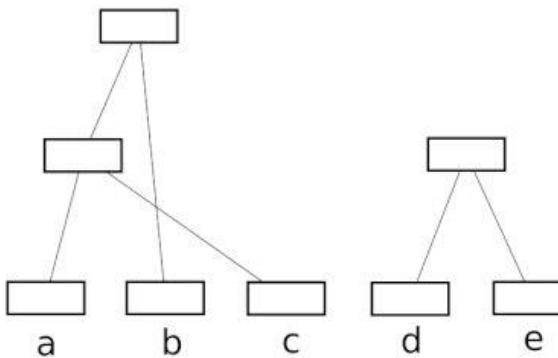
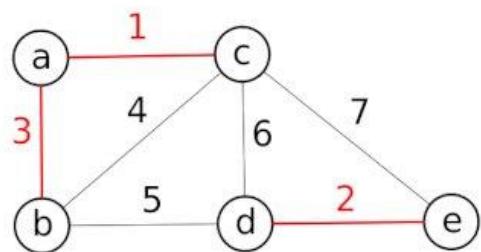


Union Find



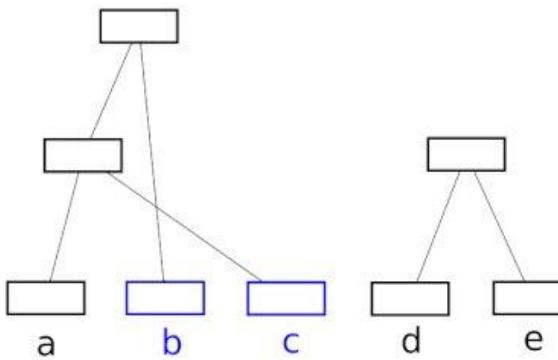
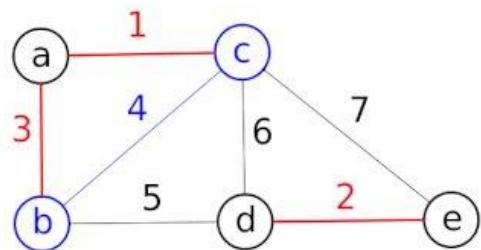


Union Find



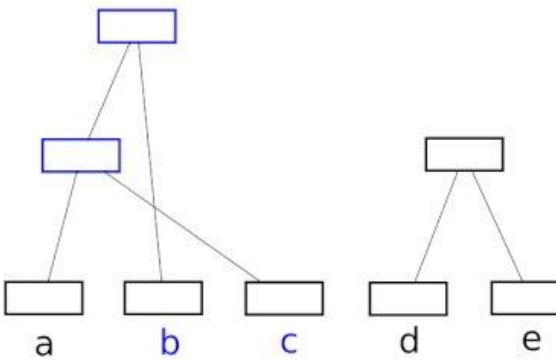
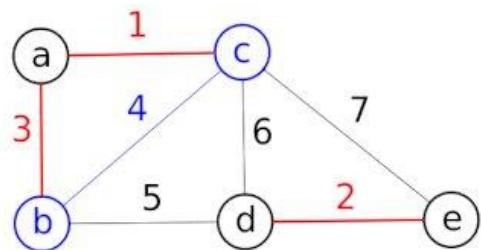


Union Find



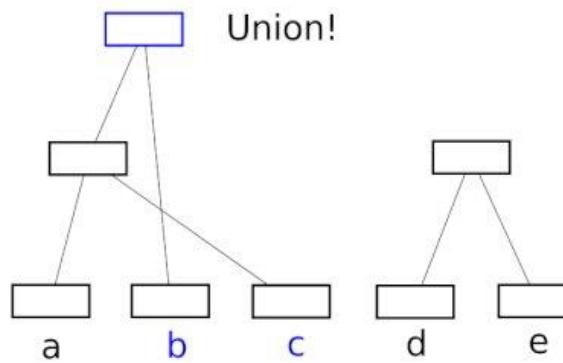
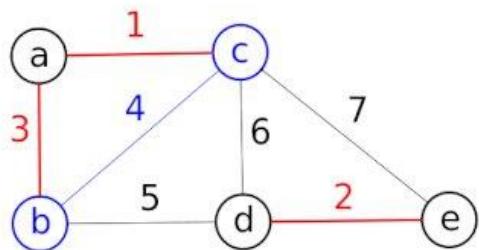


Union Find



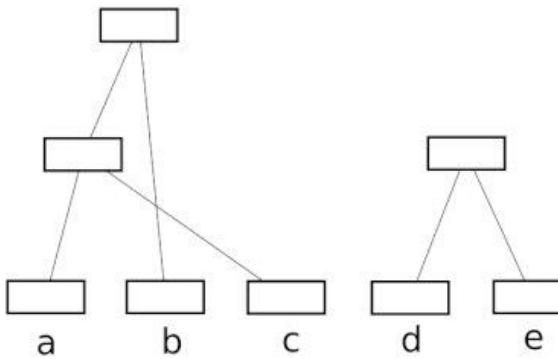
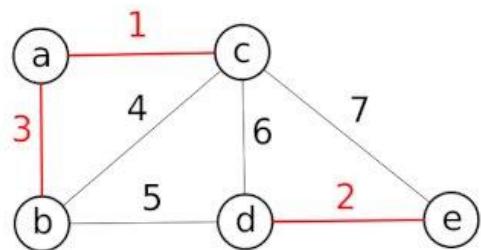


Union Find



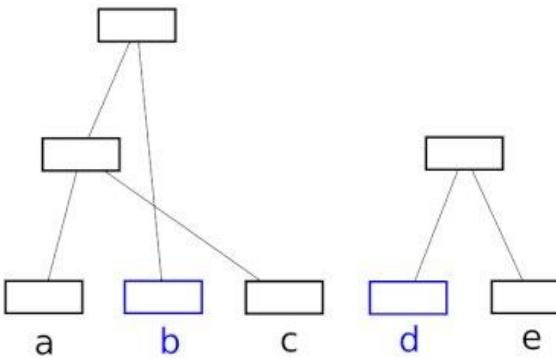
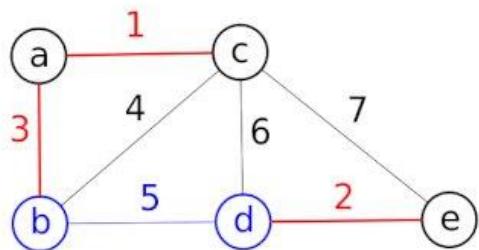


Union Find



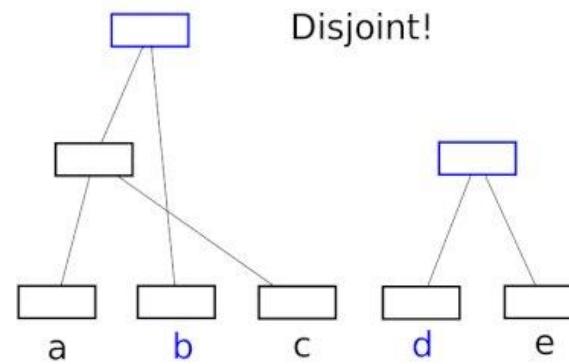
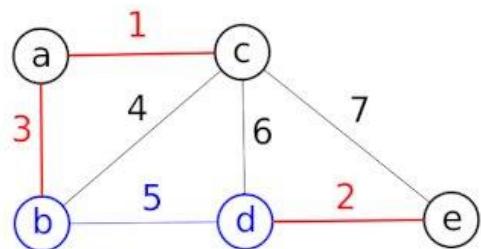


Union Find



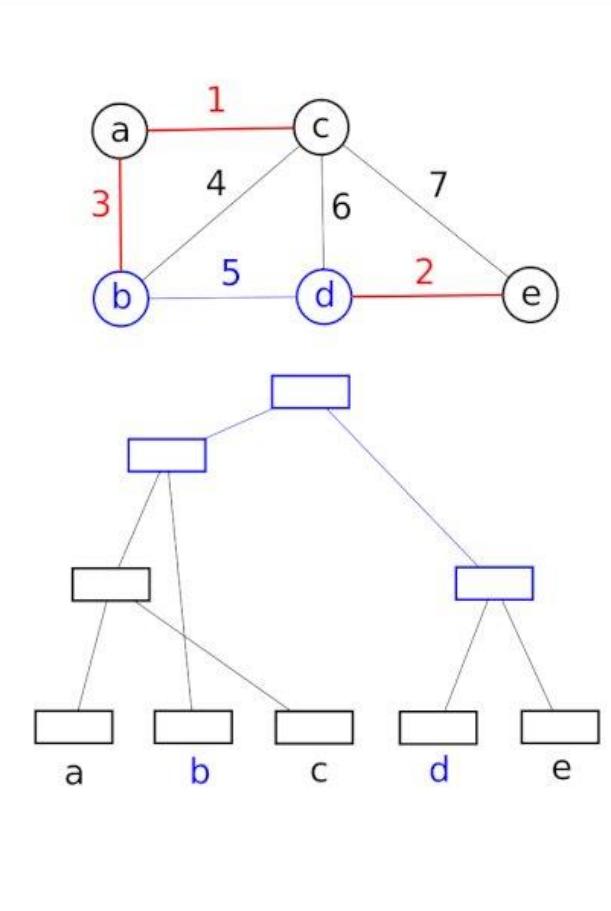


Union Find



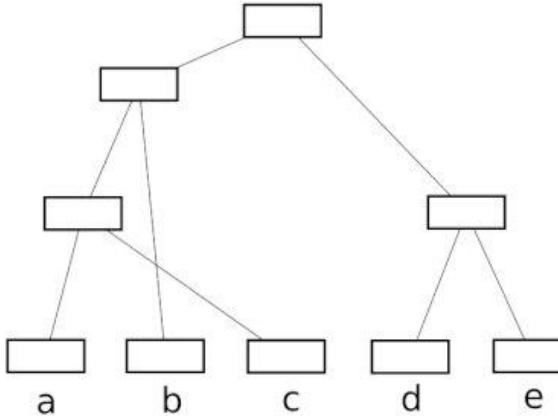
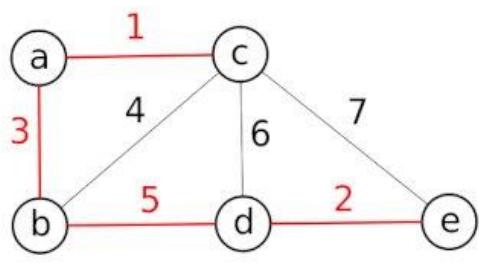


Union Find





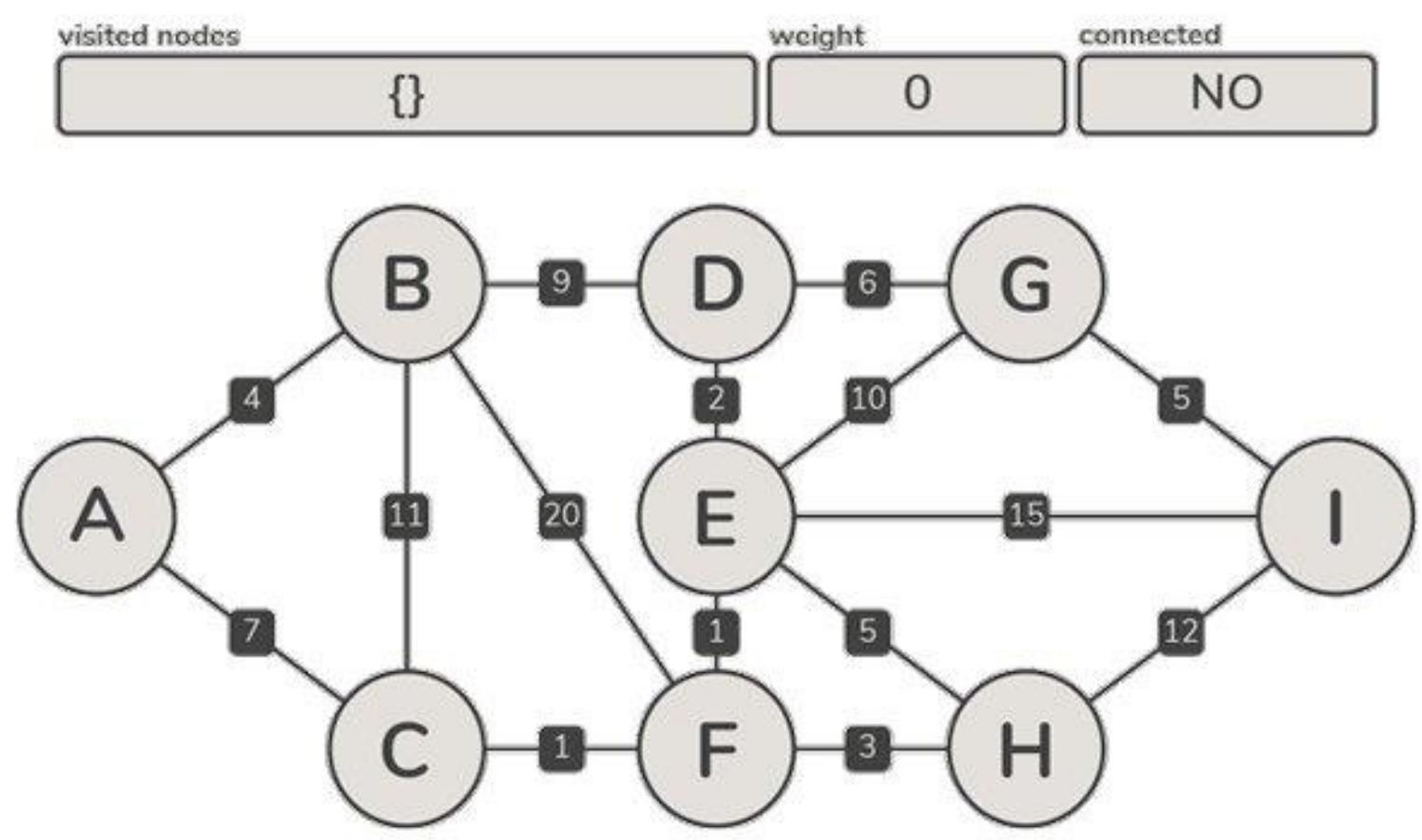
Union Find







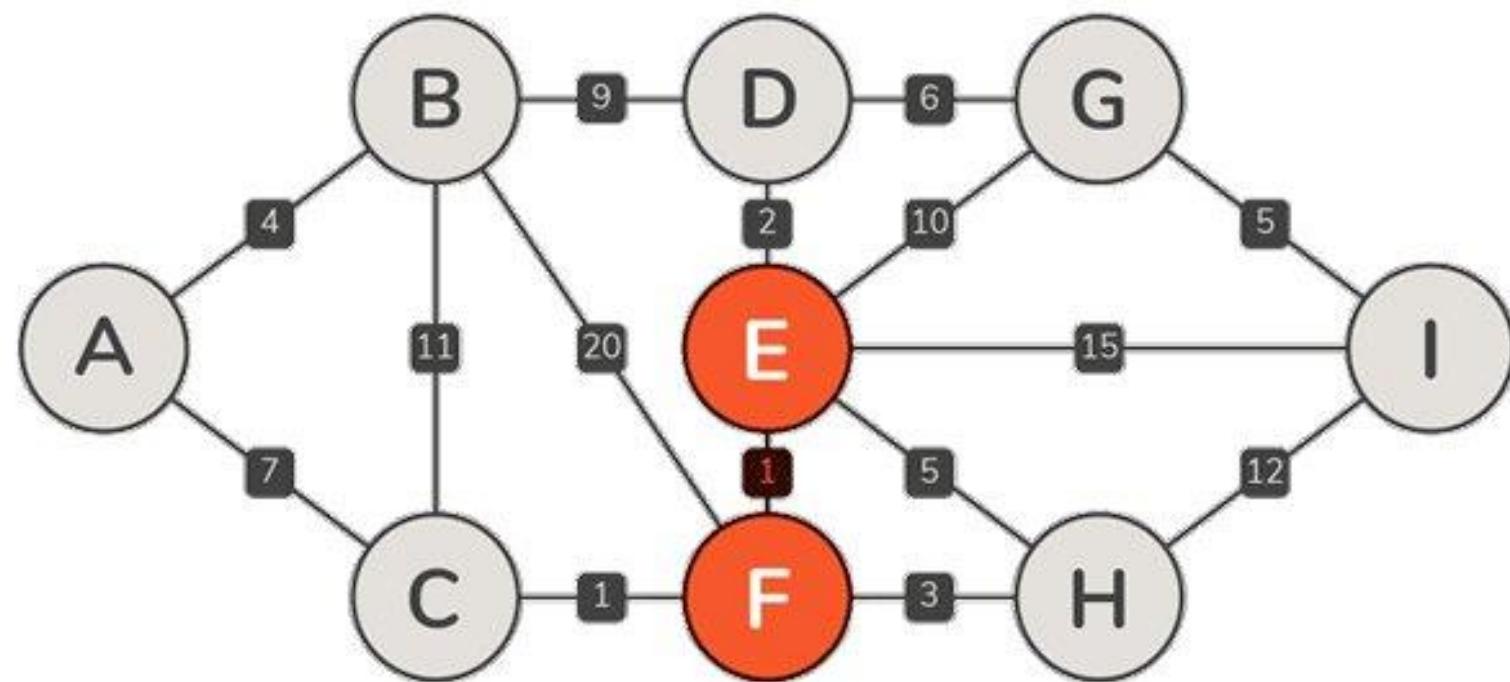
Örnek Kruskal





Örnek Kruskal

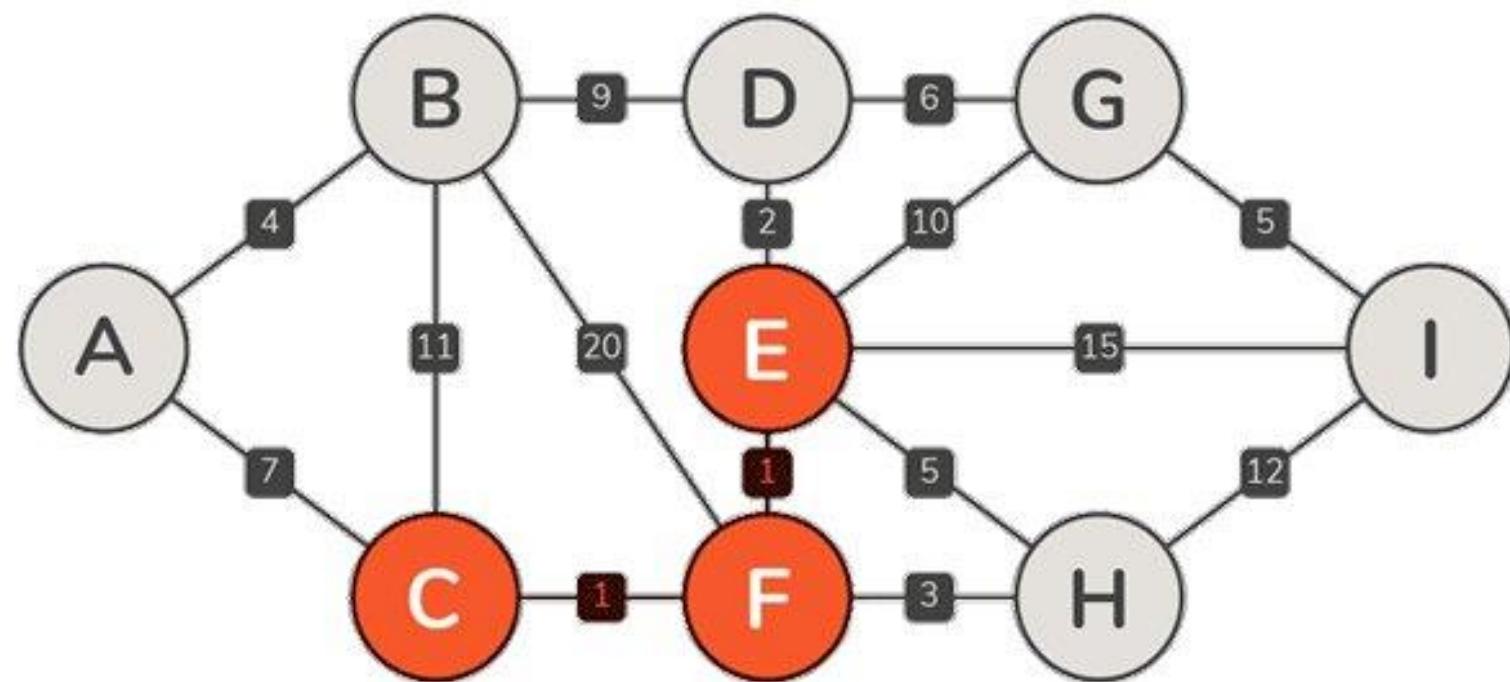
visited nodes	weight	connected
{E, F}	1	NO





Örnek Kruskal

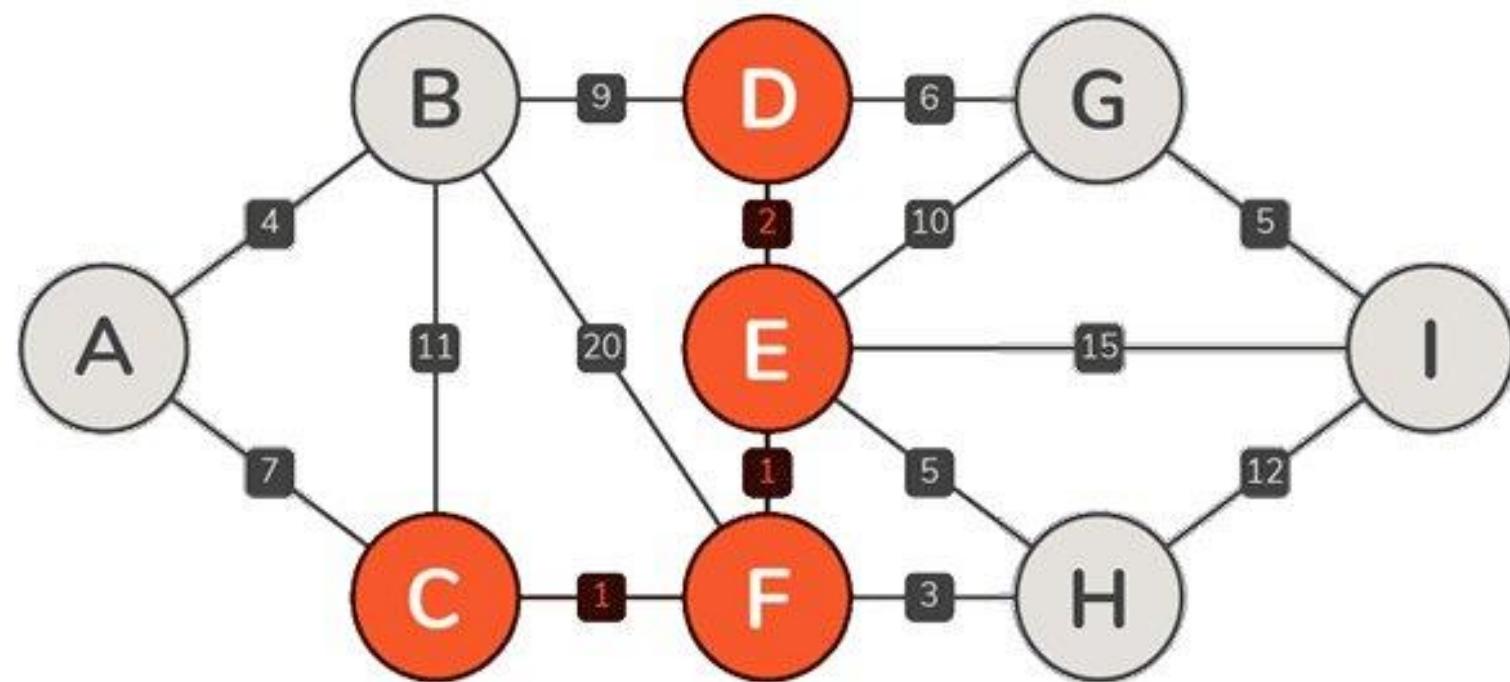
visited nodes	weight	connected
{E, F, C}	2	NO





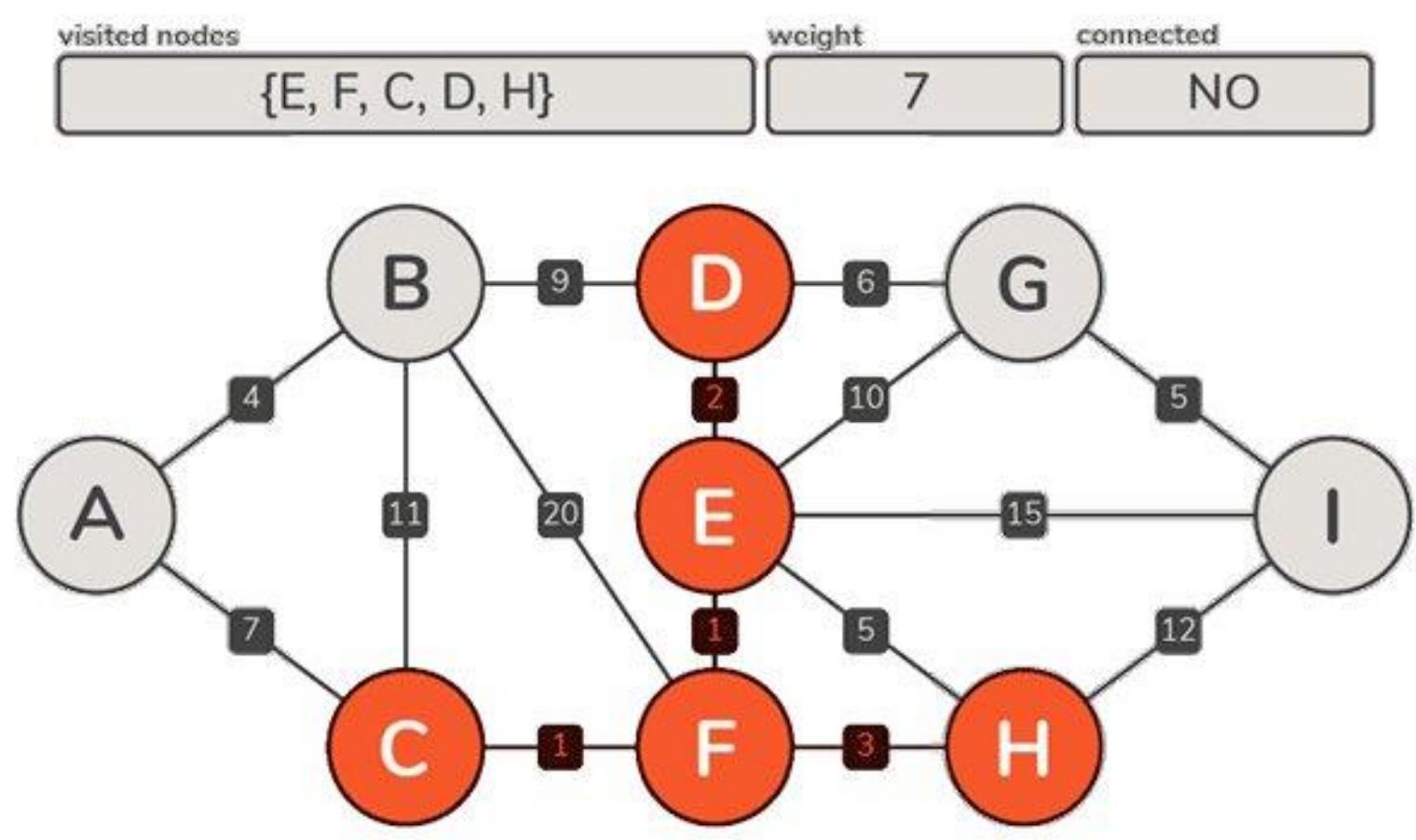
Örnek Kruskal

visited nodes	weight	connected
{E, F, C, D}	4	NO





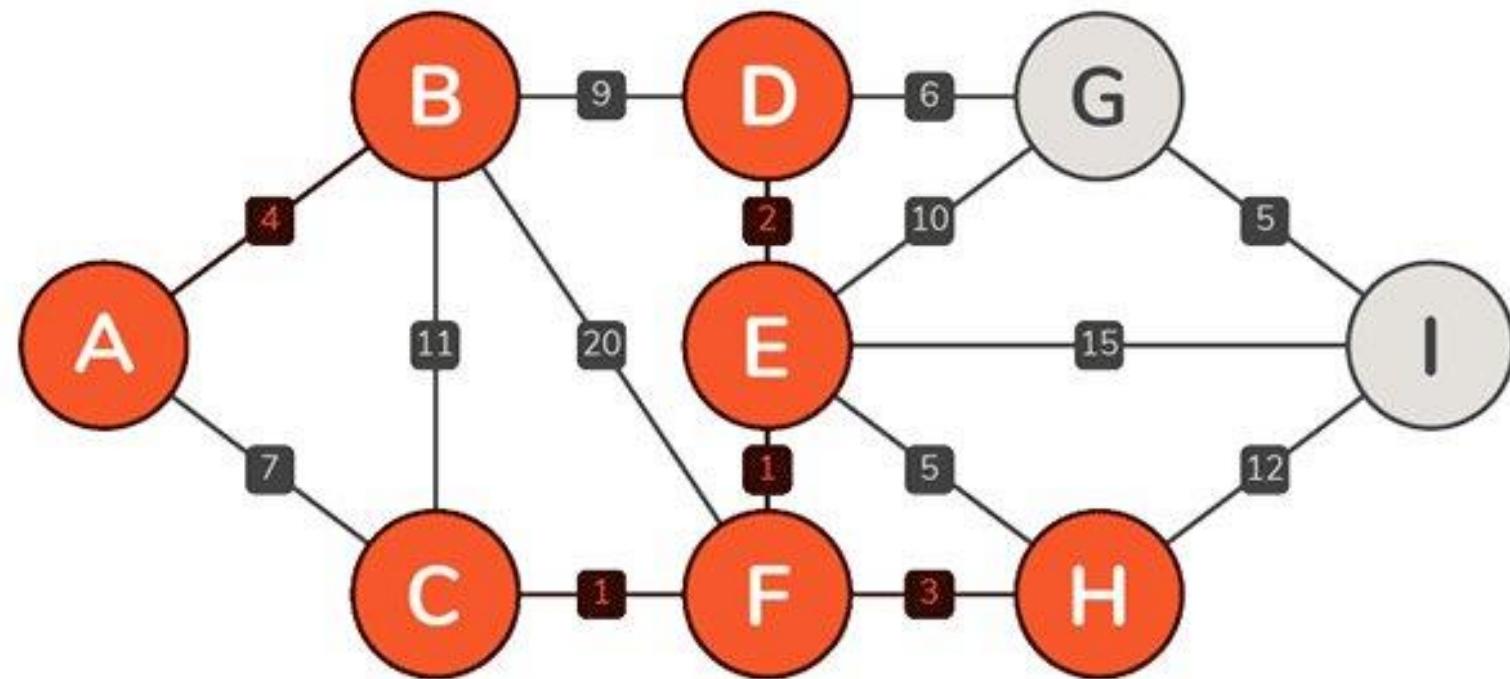
Örnek Kruskal





Örnek Kruskal

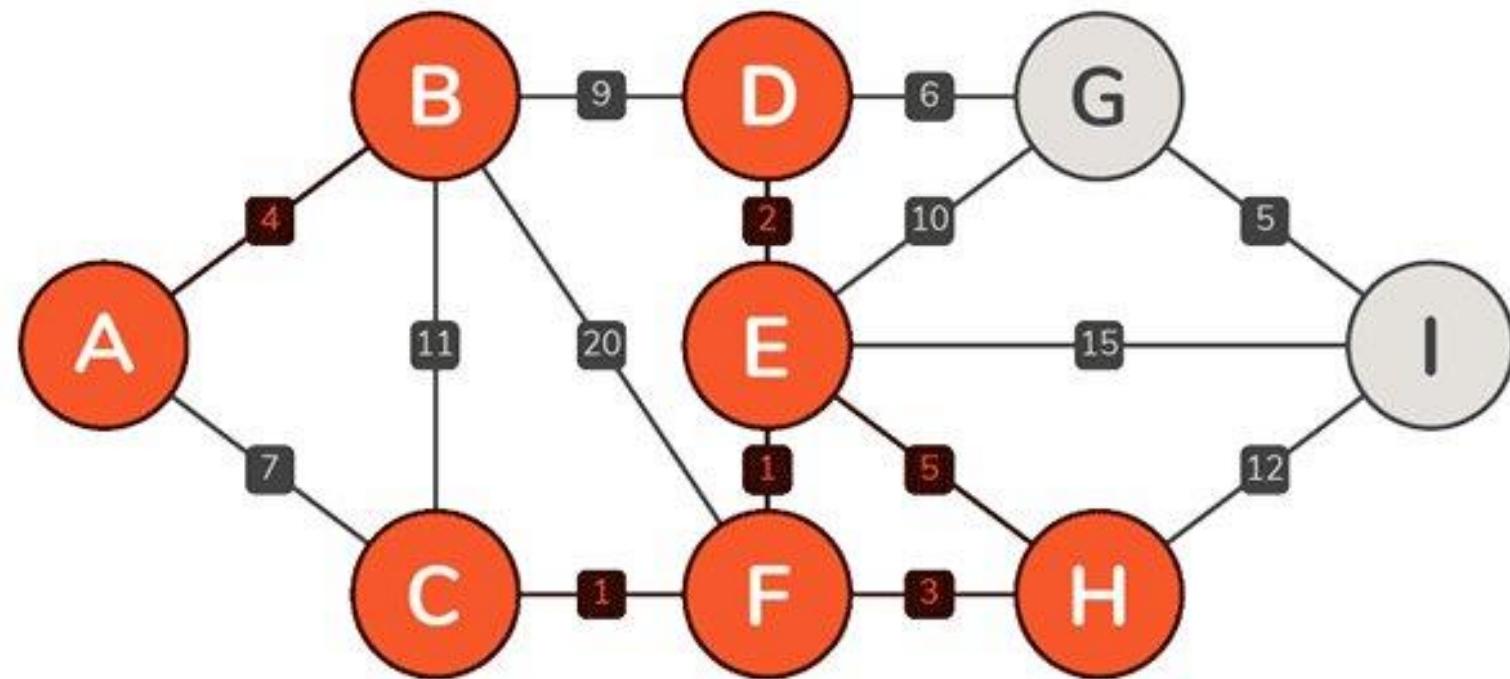
visited nodes	weight	connected
{E, F, C, D, H} {A, B}	11	NO





Örnek Kruskal

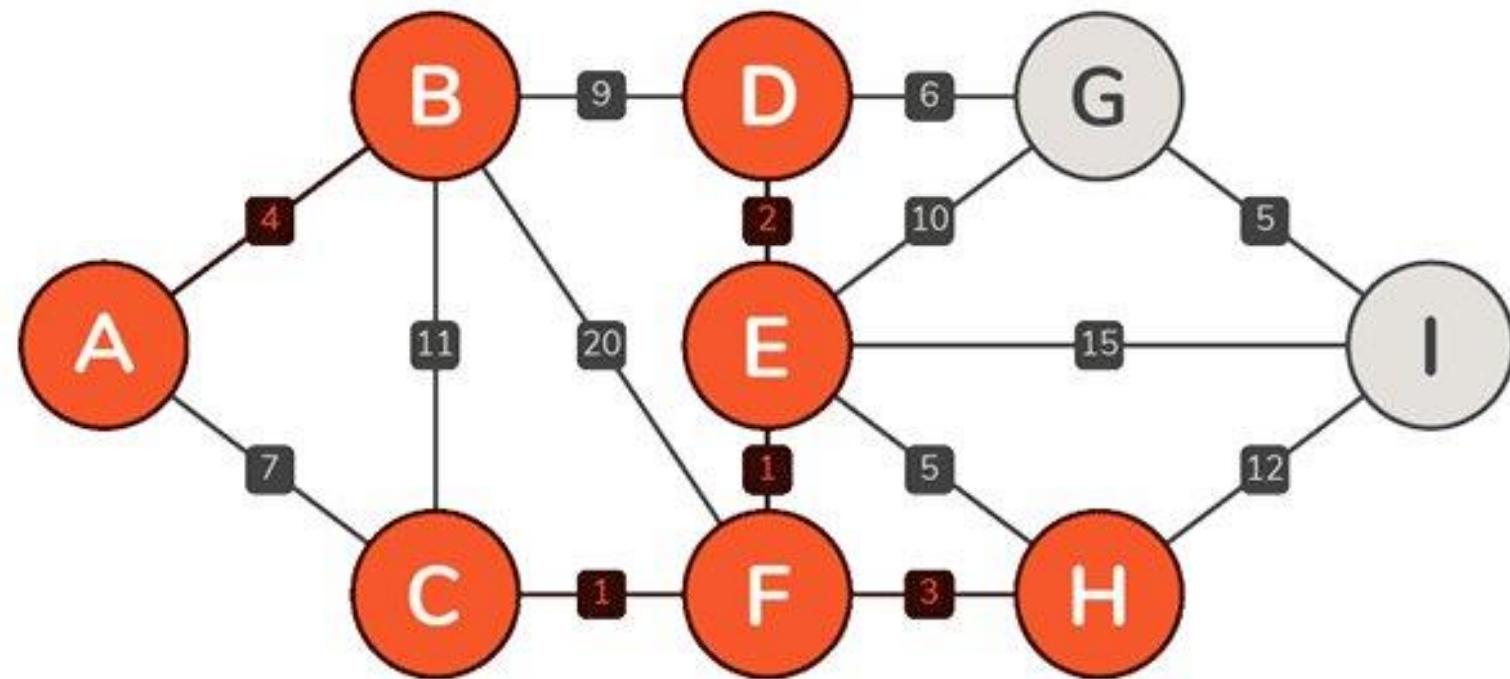
visited nodes	weight	connected
{E, F, C, D, H} {A, B}	16	NO





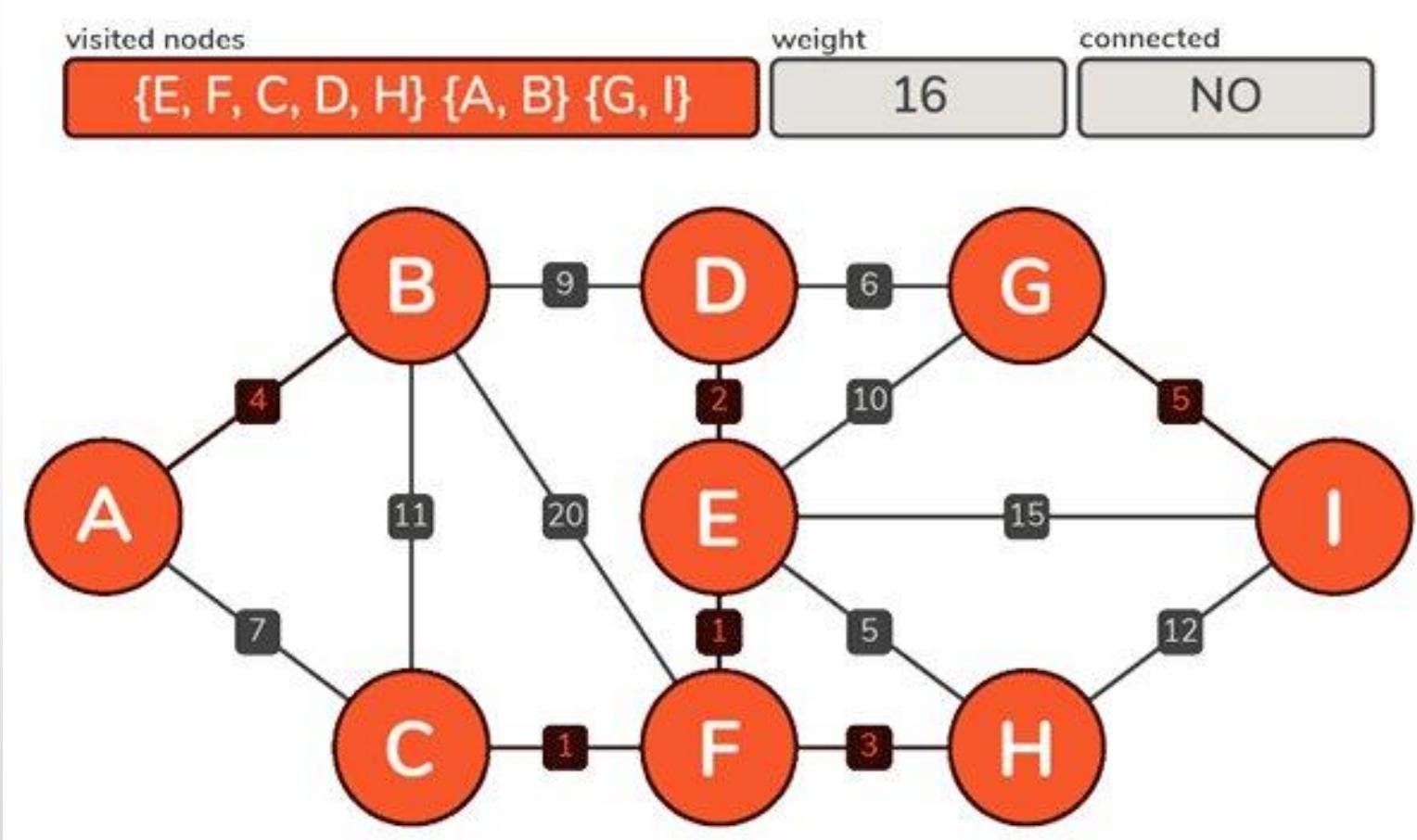
Örnek Kruskal

visited nodes	weight	connected
{E, F, C, D, H} {A, B}	11	NO





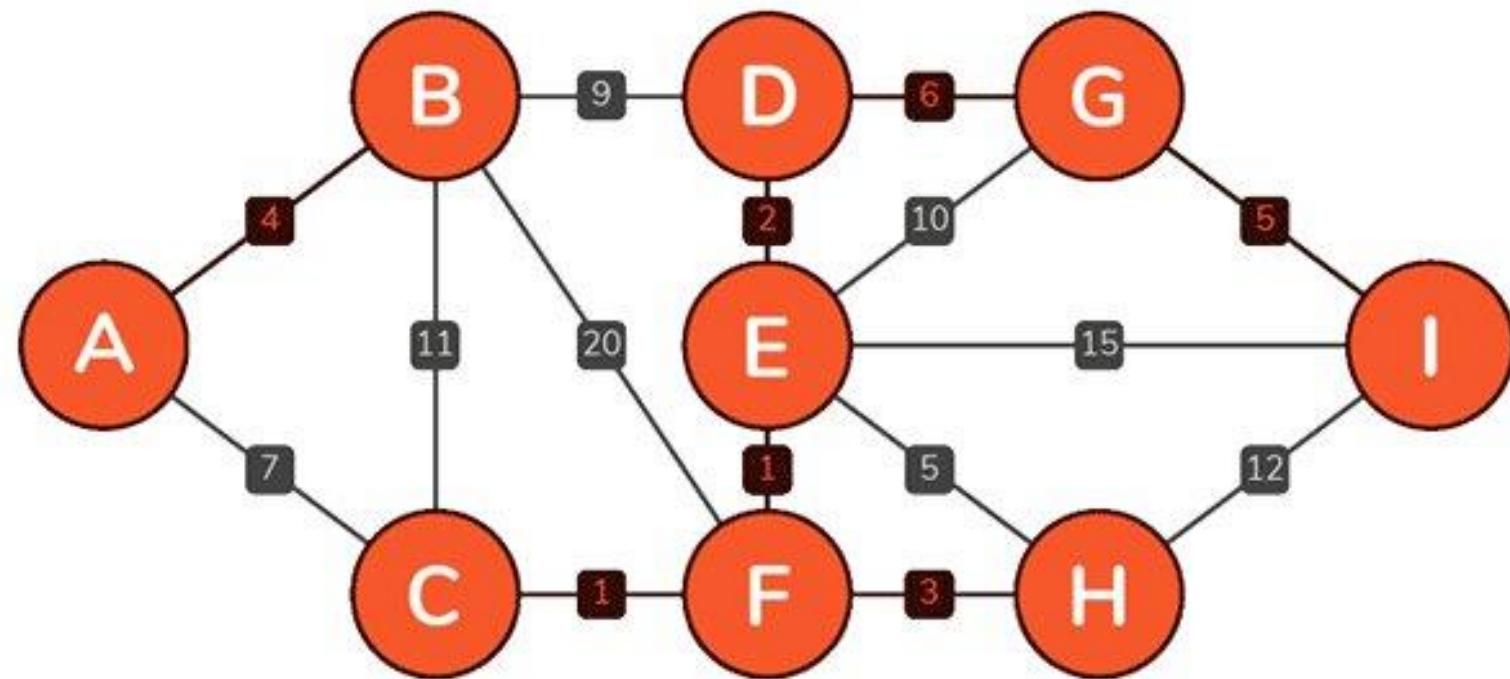
Örnek Kruskal





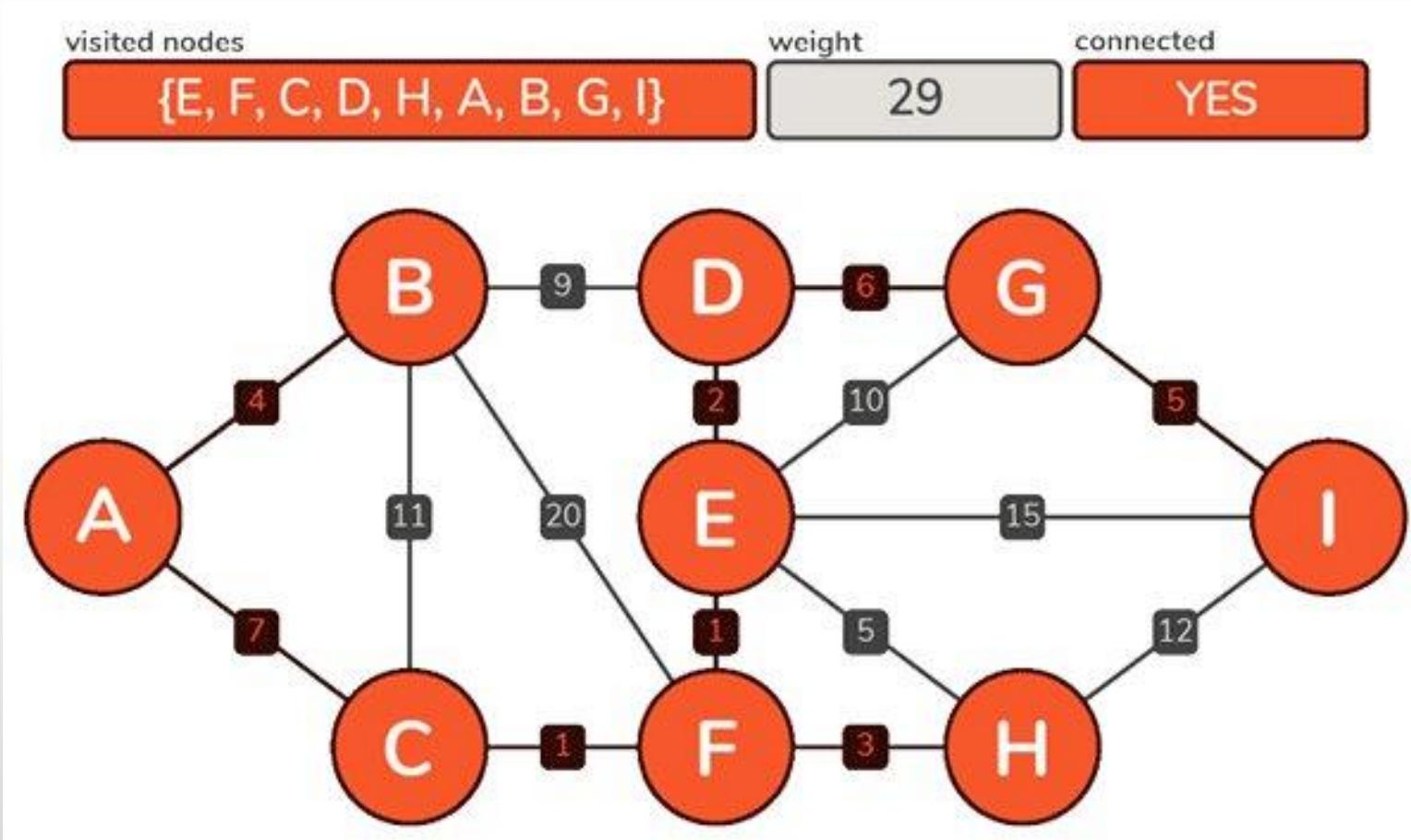
Örnek Kruskal

visited nodes	weight	connected
{E, F, C, D, H, G, I} {A, B}	22	NO





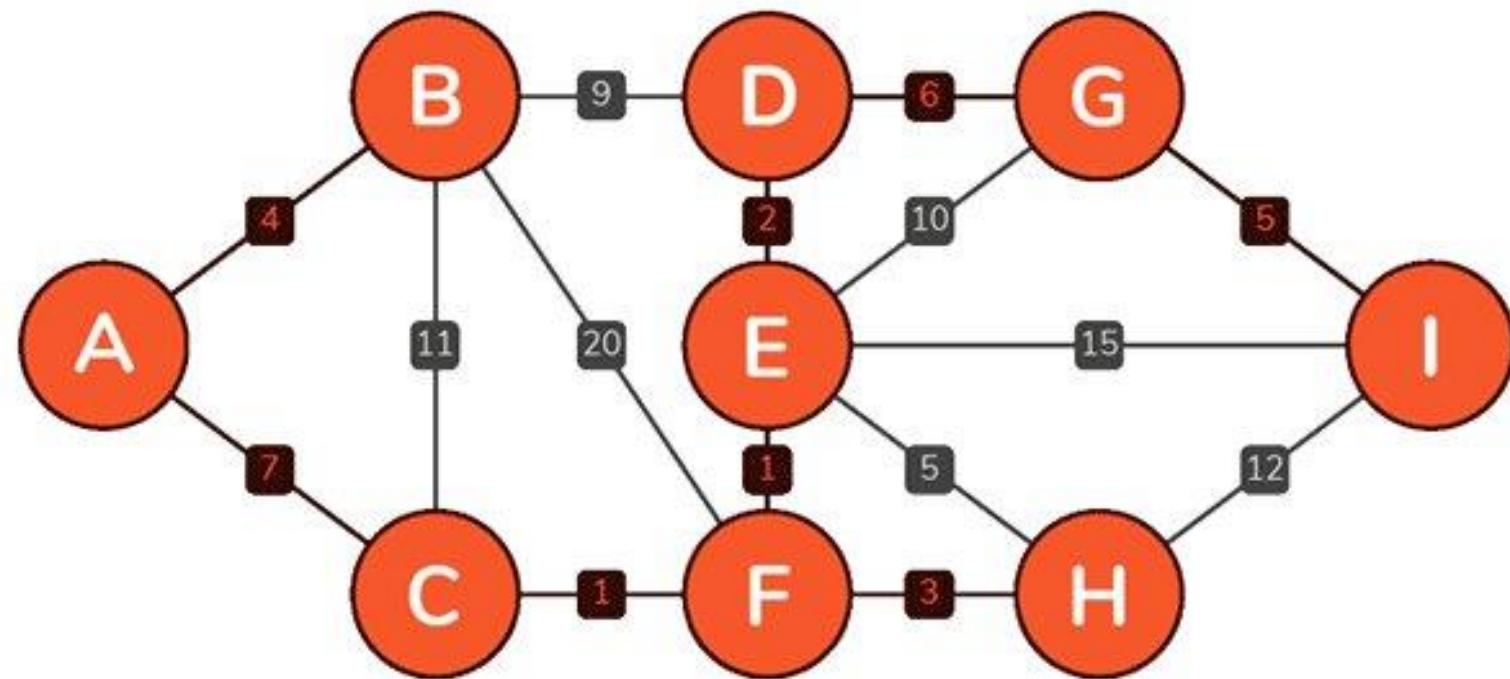
Örnek Kruskal





Örnek Kruskal

visited nodes	weight	connected
{E, F, C, D, H, A, B, G, I}	29	YES







Prim Algoritması

- Tüm düğümleri bağlayan en küçük ağırlıklı ağacı oluşturmayı amaçlar.
- Açıgözlü (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

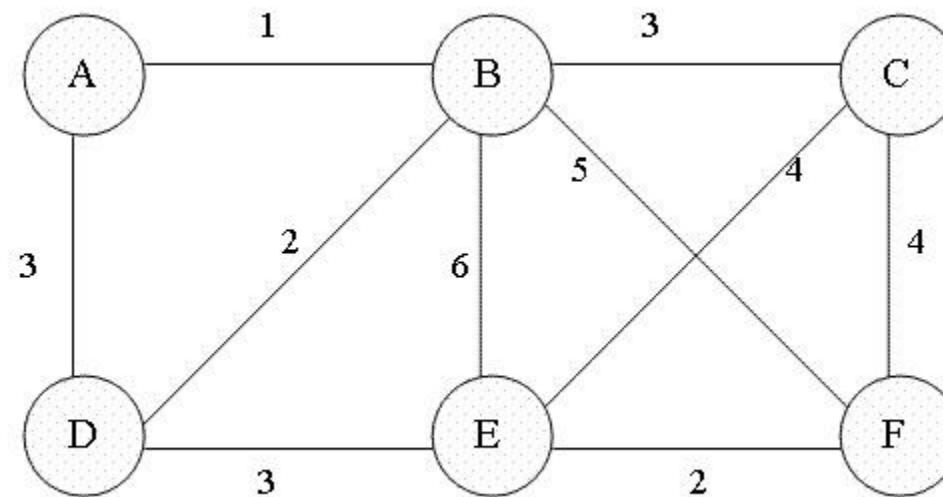
- Adım 1: Başlangıç düğümü seçilir ve bu düğüme ait olan tüm kenarlar bir öncelik kuyruğuna eklenir.
- Adım 2: Kuyruktan, ağaçta olmayan en küçük ağırlıklı kenar seçilir.
- Adım 3: Seçilen kenar ile bağlantılı olan yeni düğüm ağaç'a eklenir.
- Adım 4: Yeni eklenen düğüme bağlı kenarlar öncelik kuyruğuna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



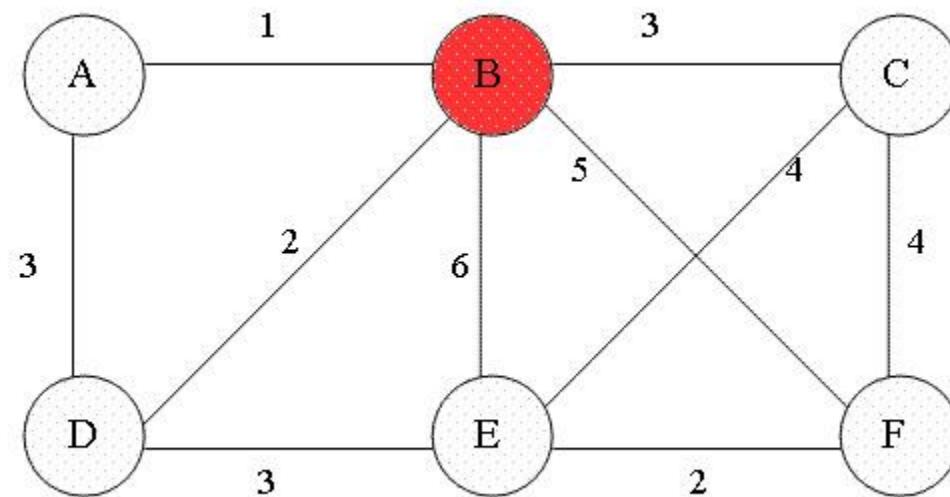
Karmaşıklık Analizi

- 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.

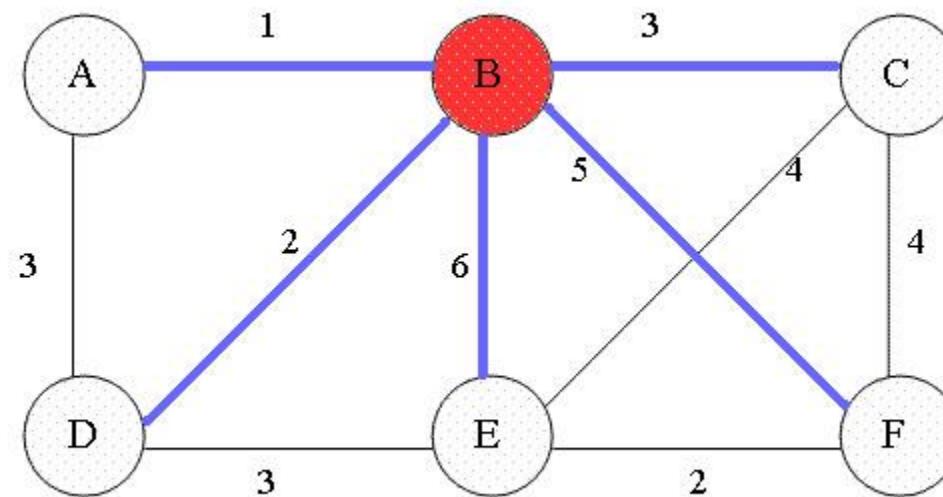
Prim



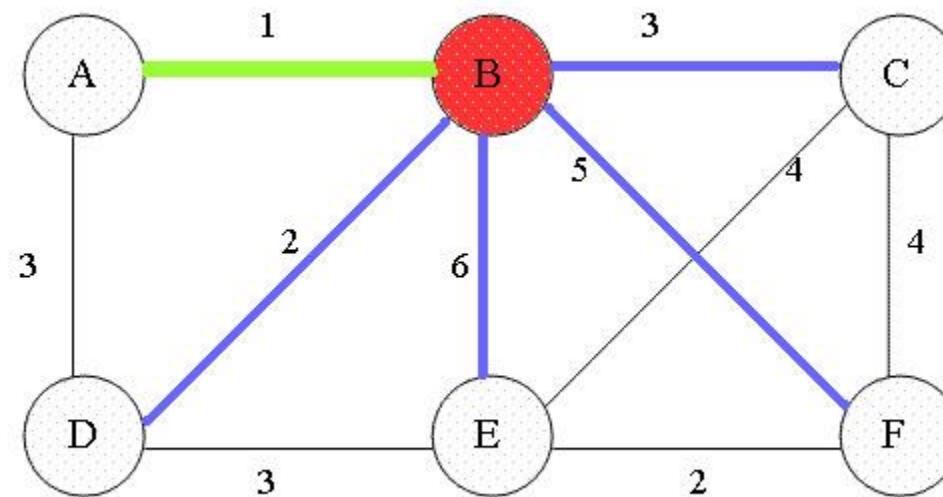
Prim



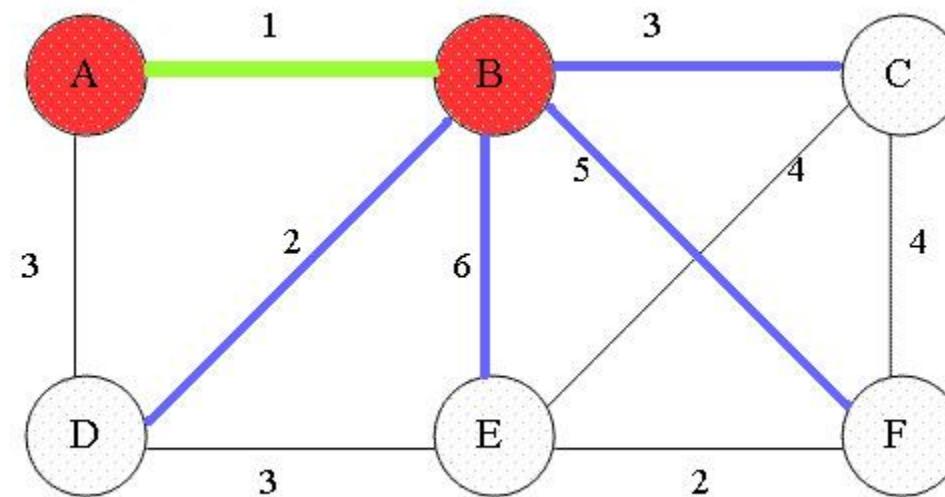
Prim



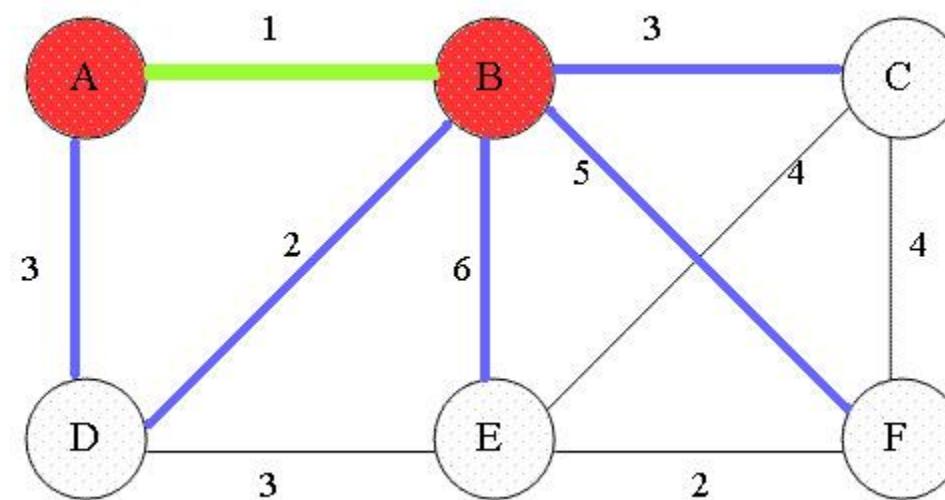
Prim



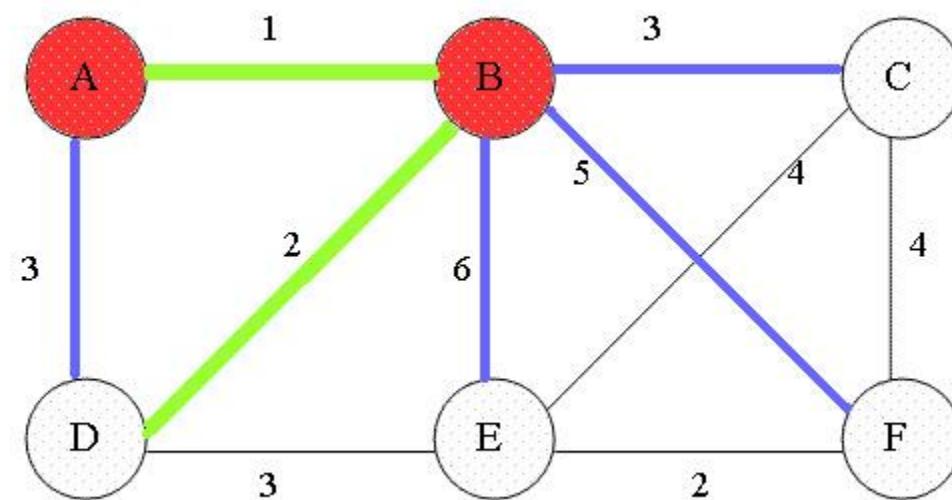
Prim



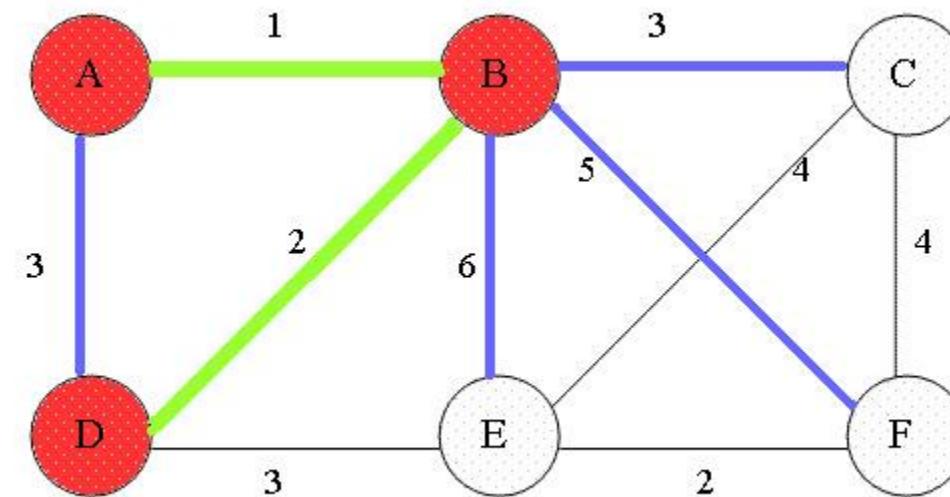
Prim



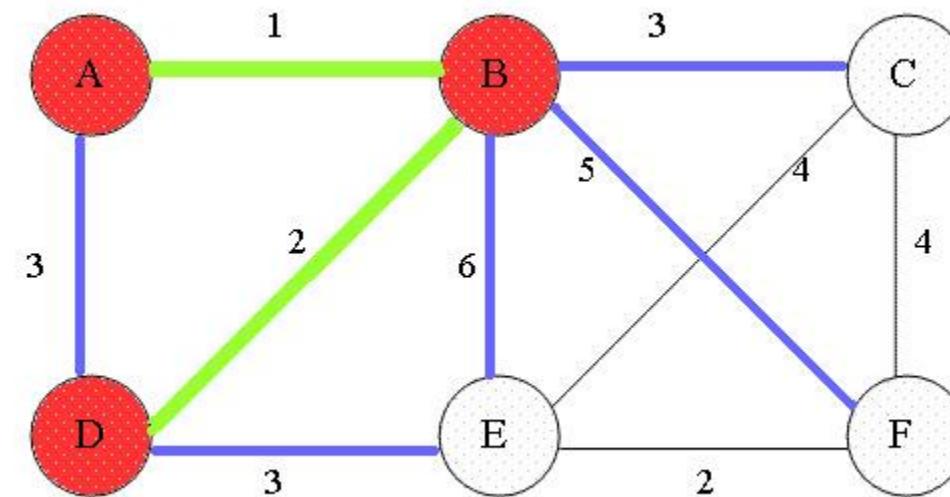
Prim



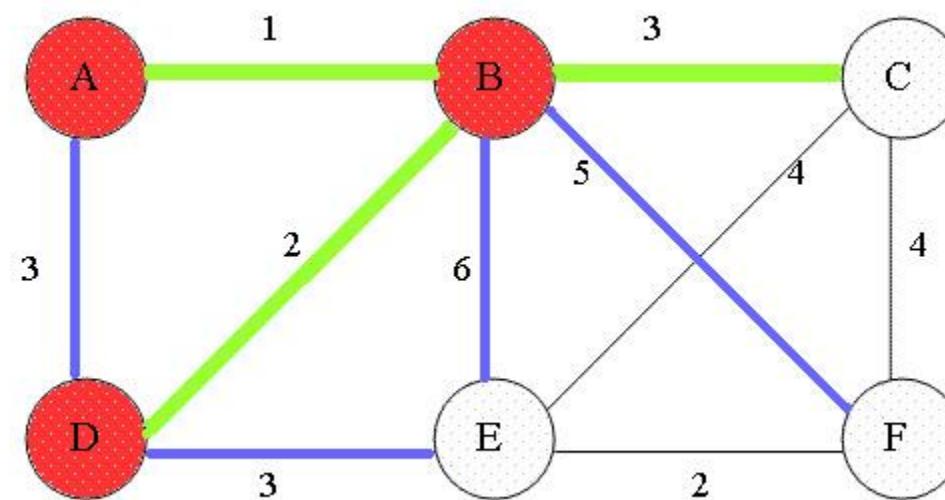
Prim



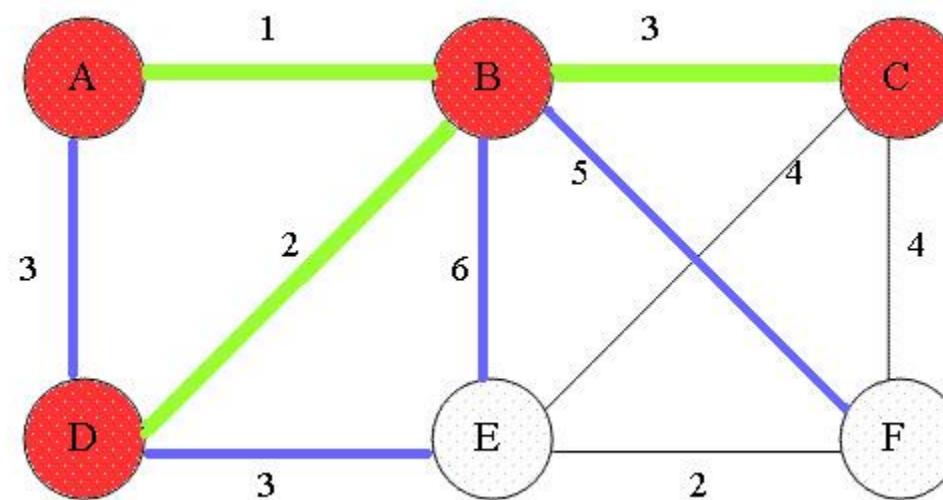
Prim



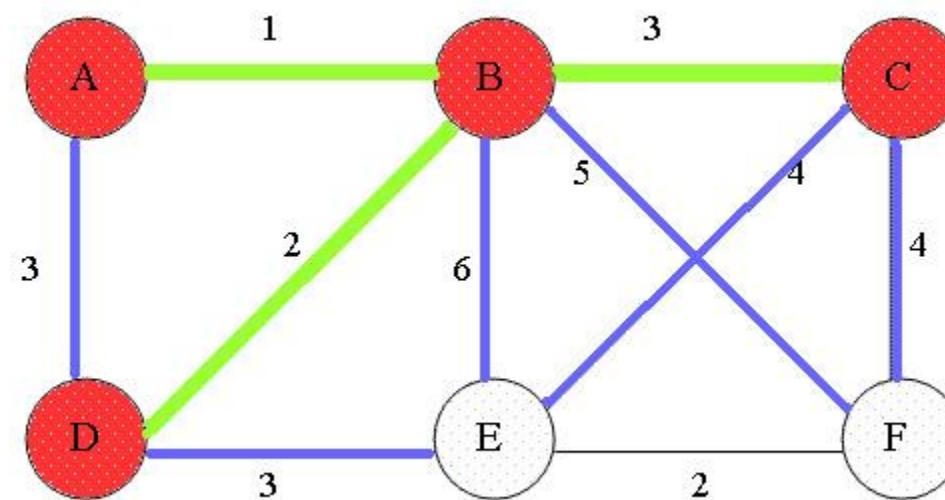
Prim



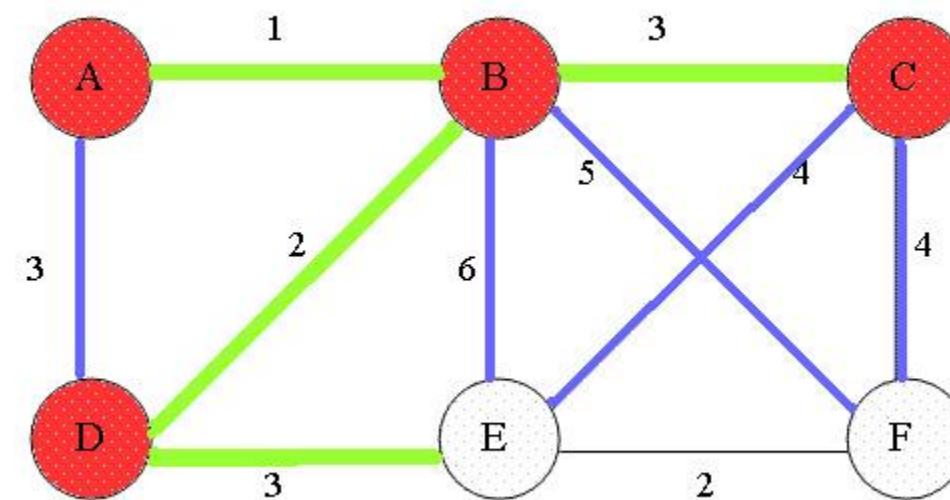
Prim



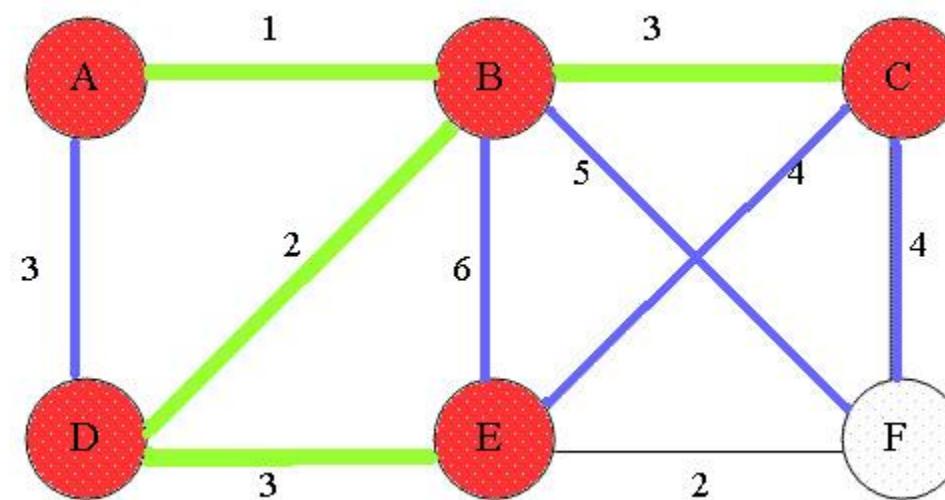
Prim



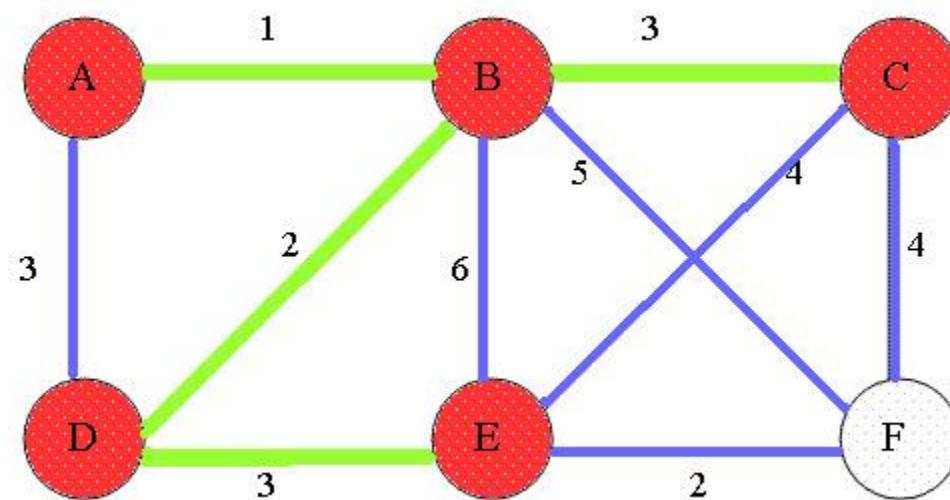
Prim



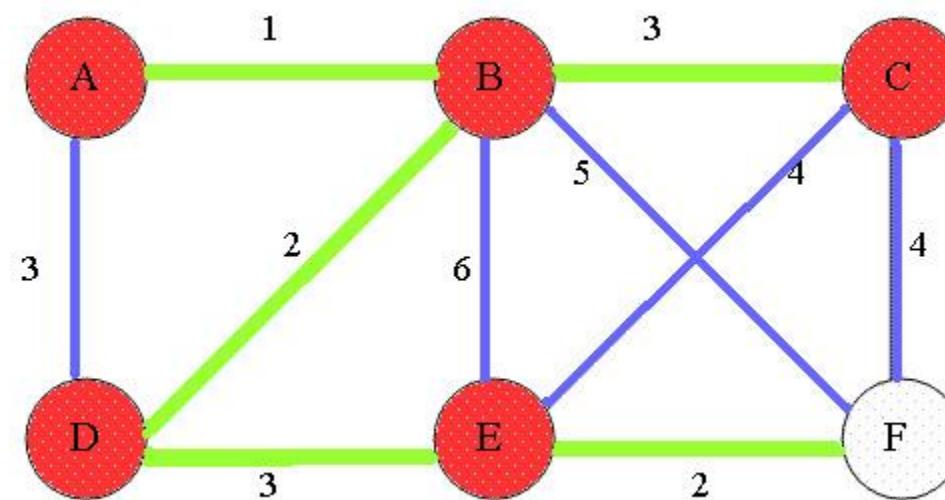
Prim



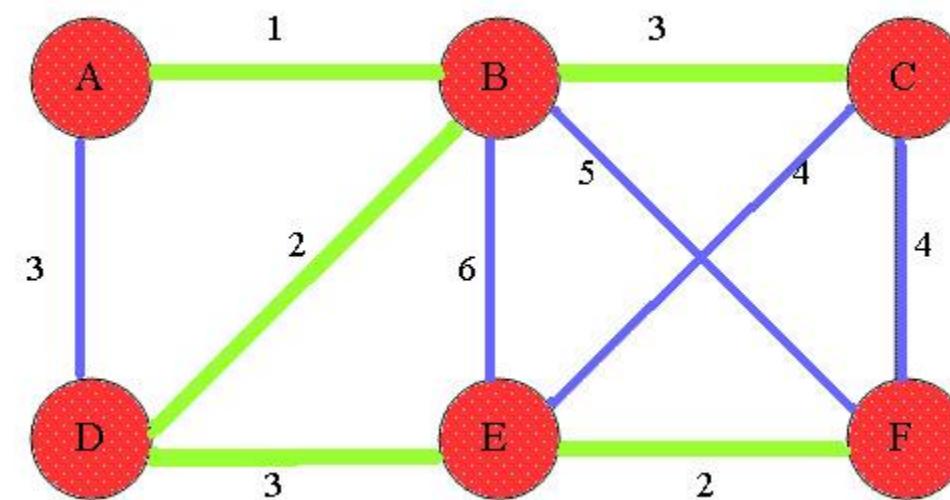
Prim



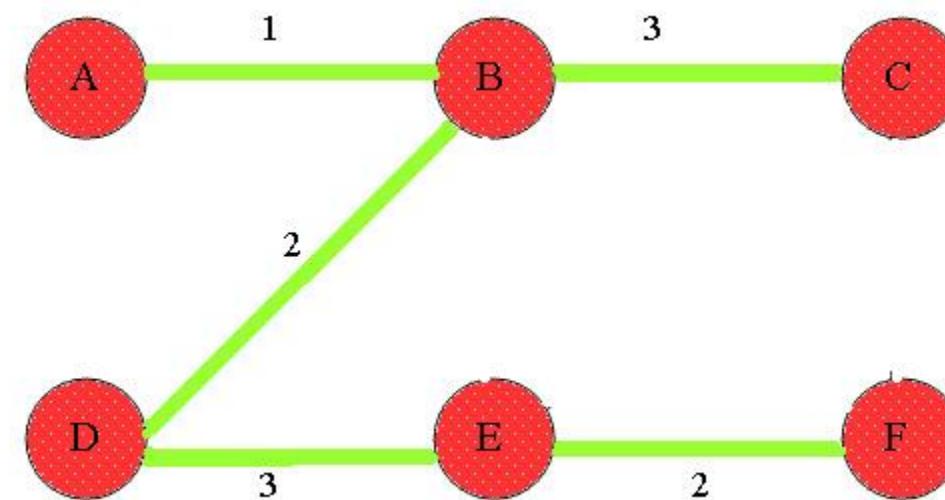
Prim



Prim



Prim

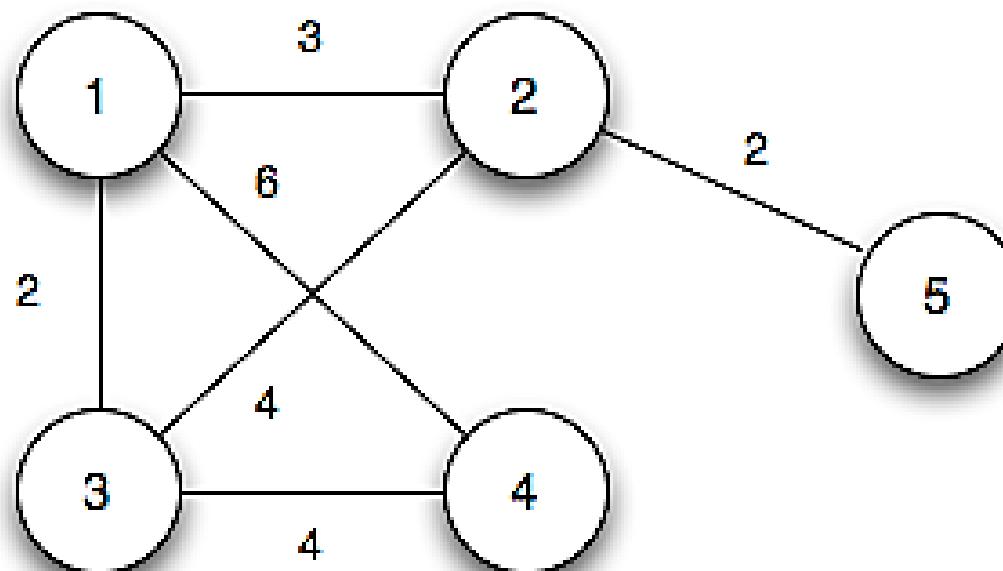






Örnek

- Aşağıdaki yönsüz çizge verilsin.





İlkendirme Aşaması

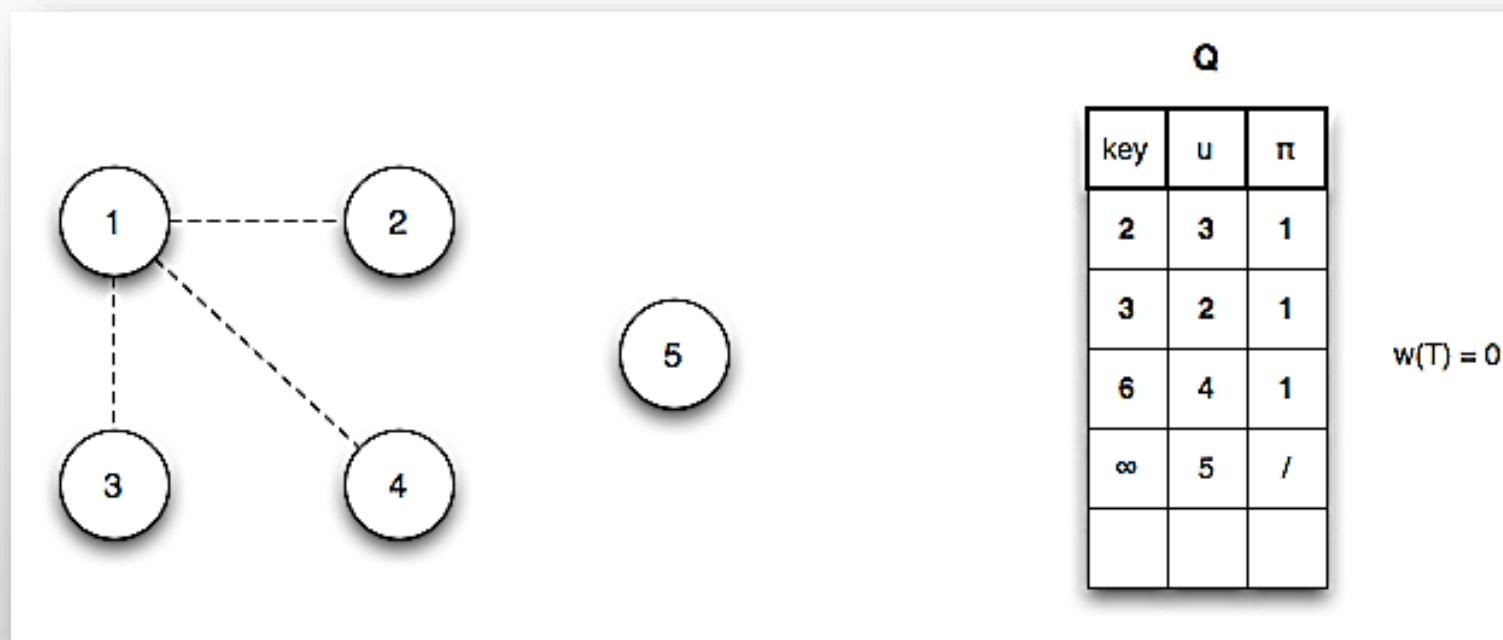
- Düğüm 1 ile kuyruk başlatılır.





Adım 1

- Kuyruktan düğüm 1'i al, Q'yu güncelle.
- $u_3.key = 2$ ((u_1, u_3)), $u_2.key = 3$ ((u_1, u_2)), $u_4.key = 6$ ((u_1, u_4))





Adım 2

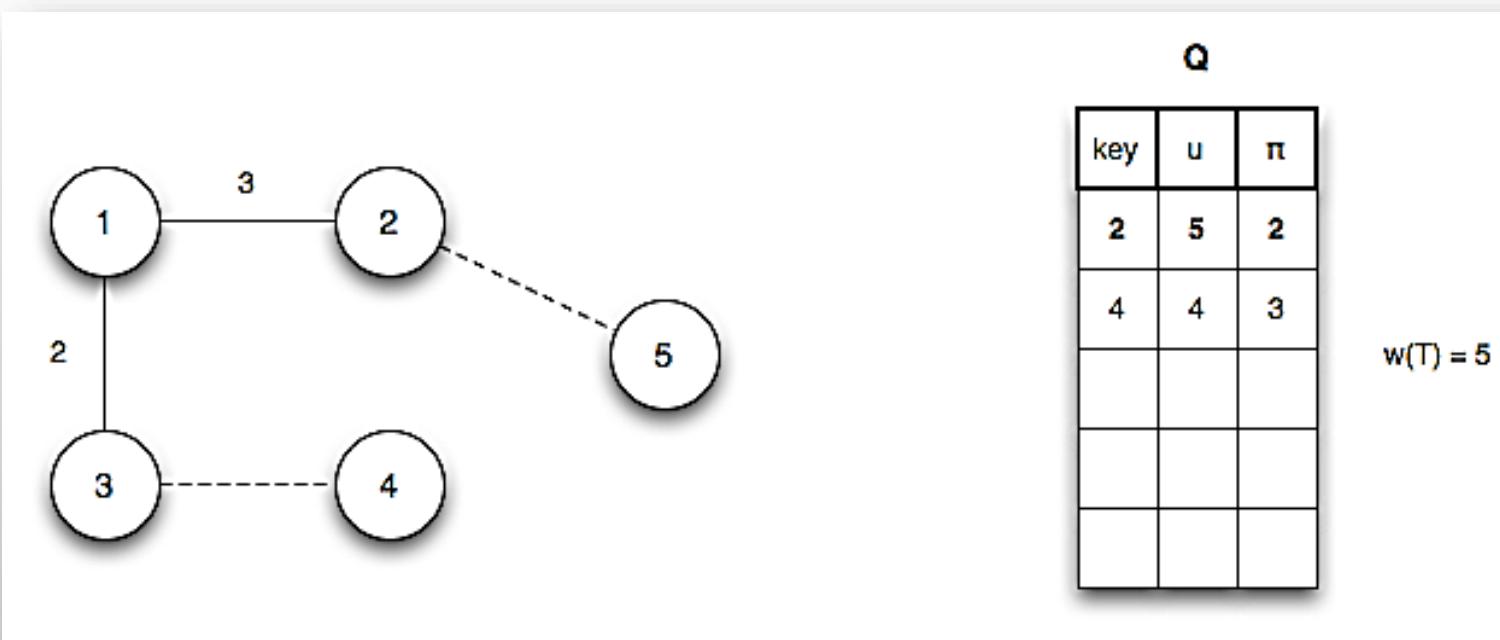
- Kuyruktan düğüm 3'ü al, T'ye (u_1, u_3) kenarını ekle. Q'yu güncelle.
- $u4.key = 4$ $((u3, u4))$





Adım 3

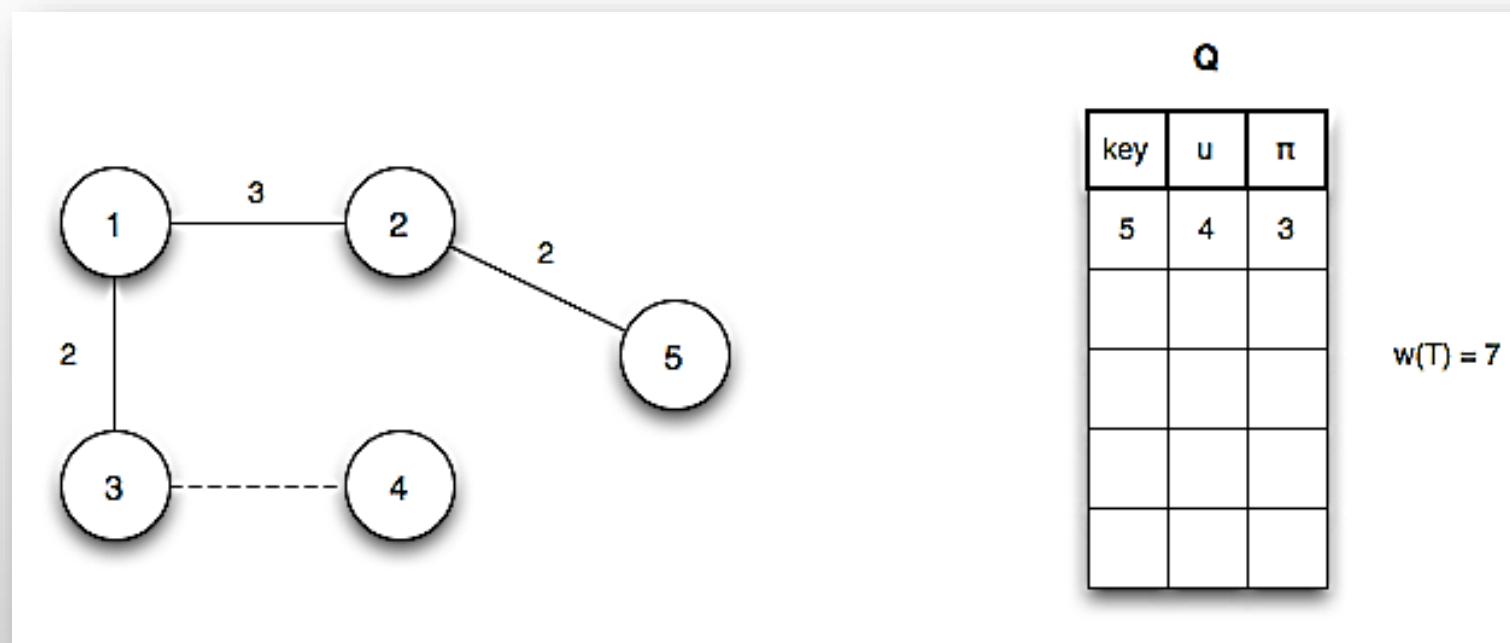
- Kuyruktan düğüm 2'yi al, T'ye (u_1, u_2) kenarını ekle. Q'yu güncelle.
- $u_5.key = 2 \ ((u_2, u_5))$





Adım 4

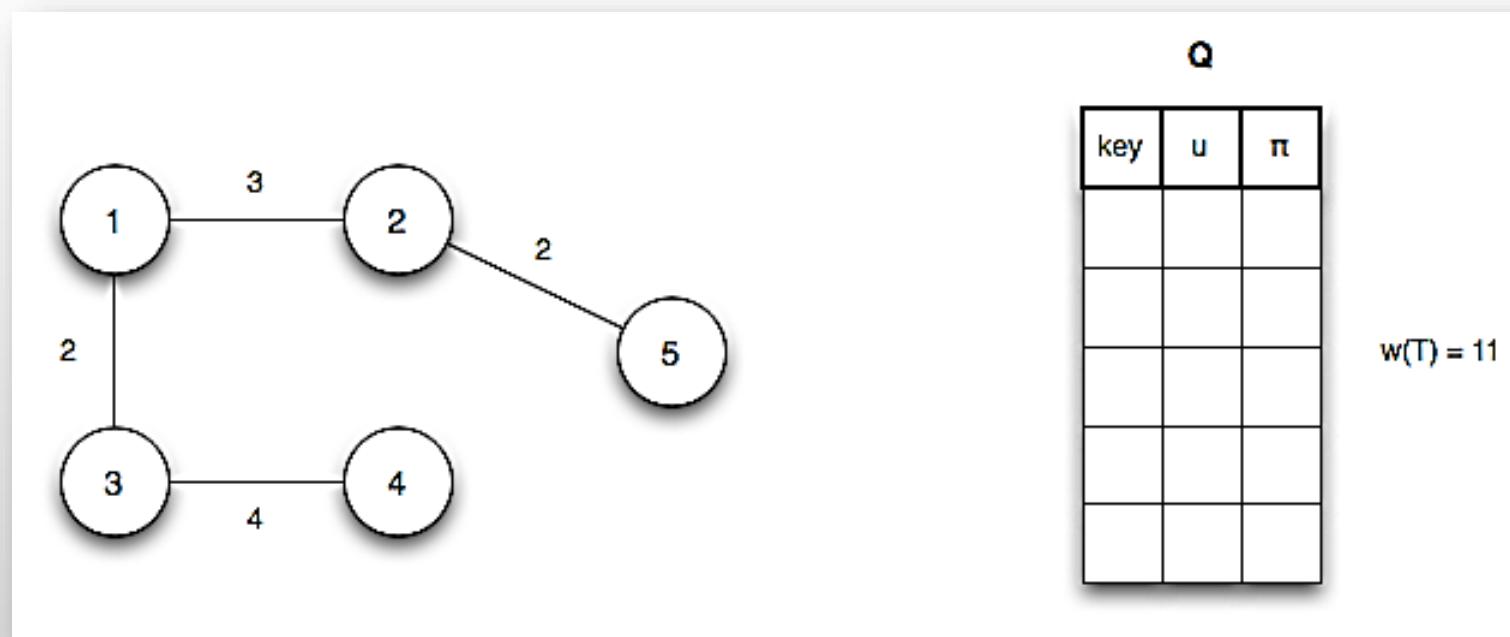
- Kuyruktan düğüm 5'i al, T'ye (u_2, u_5) kenarını ekle. Q'da güncelleme yok.





Adım 5

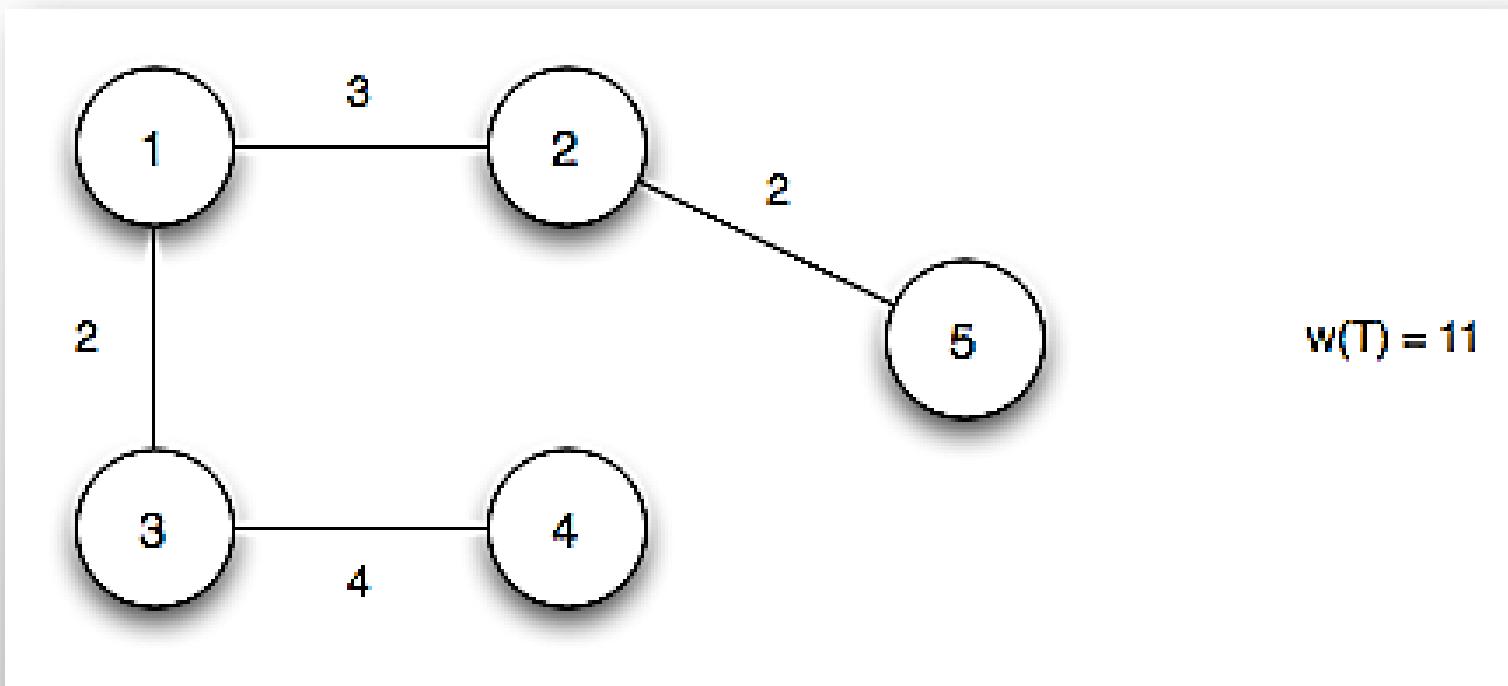
- Kuyruktan düğüm 4'ü al, T'ye (u_3, u_4) kenarını ekle. Q'da güncelleme yok.





Son Durum

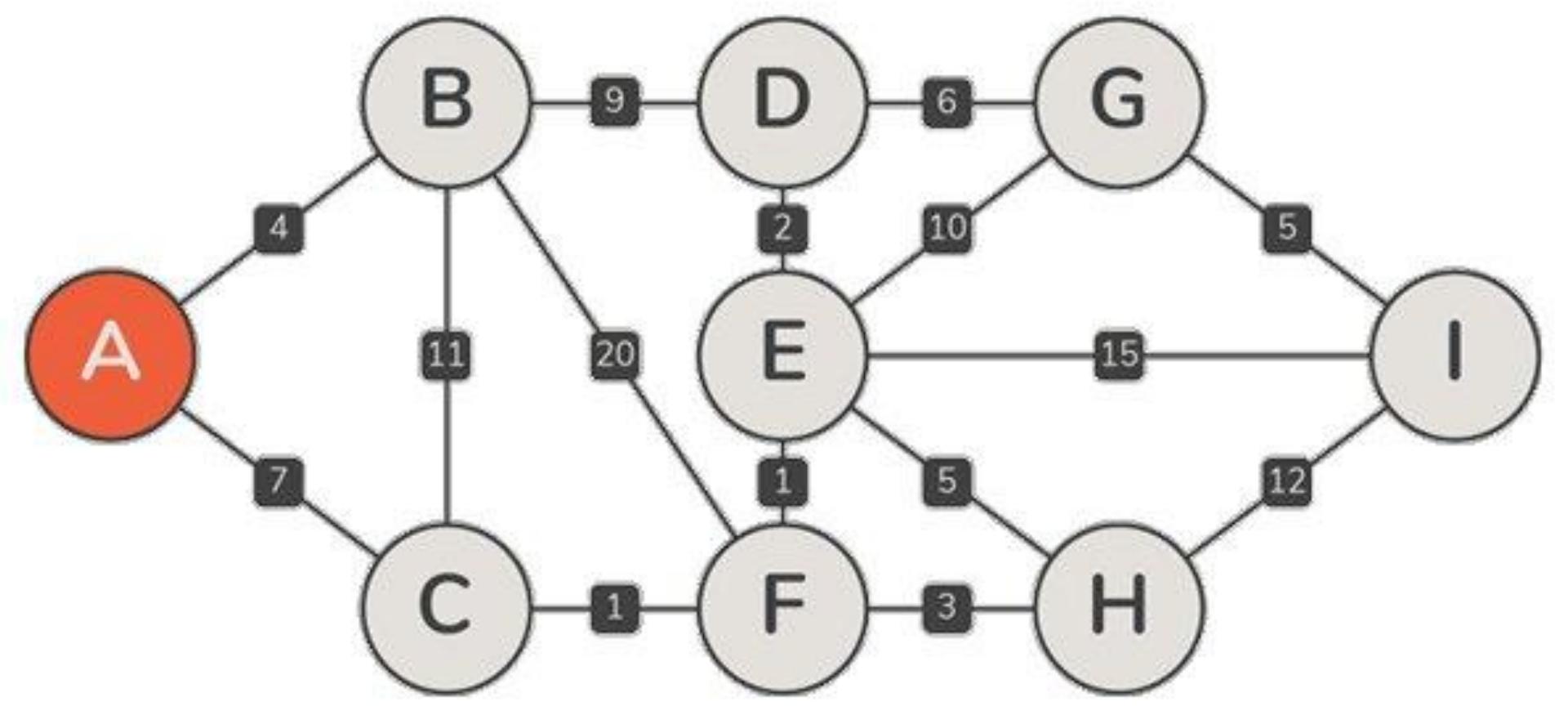
- $Q = \emptyset$





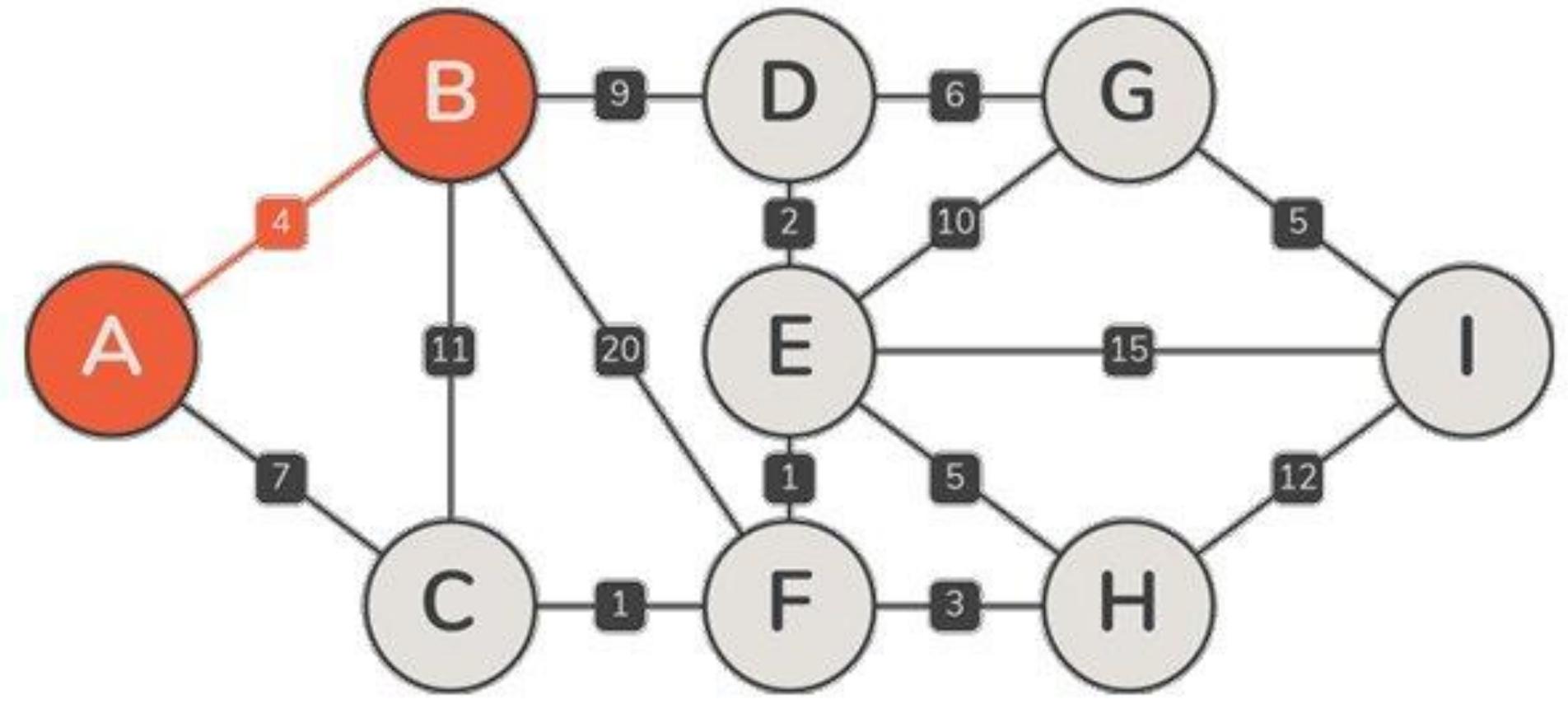


Örnek Prim



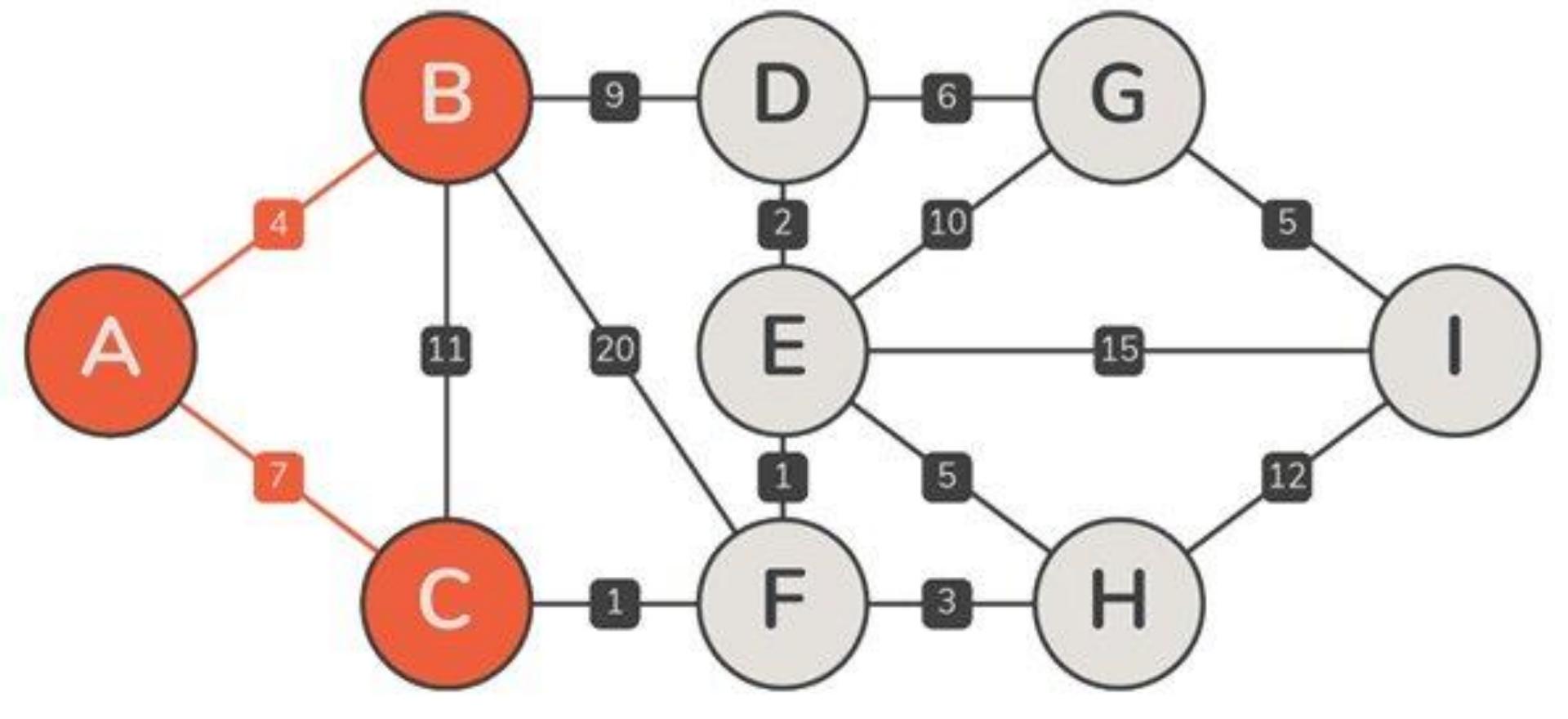


Örnek Prim



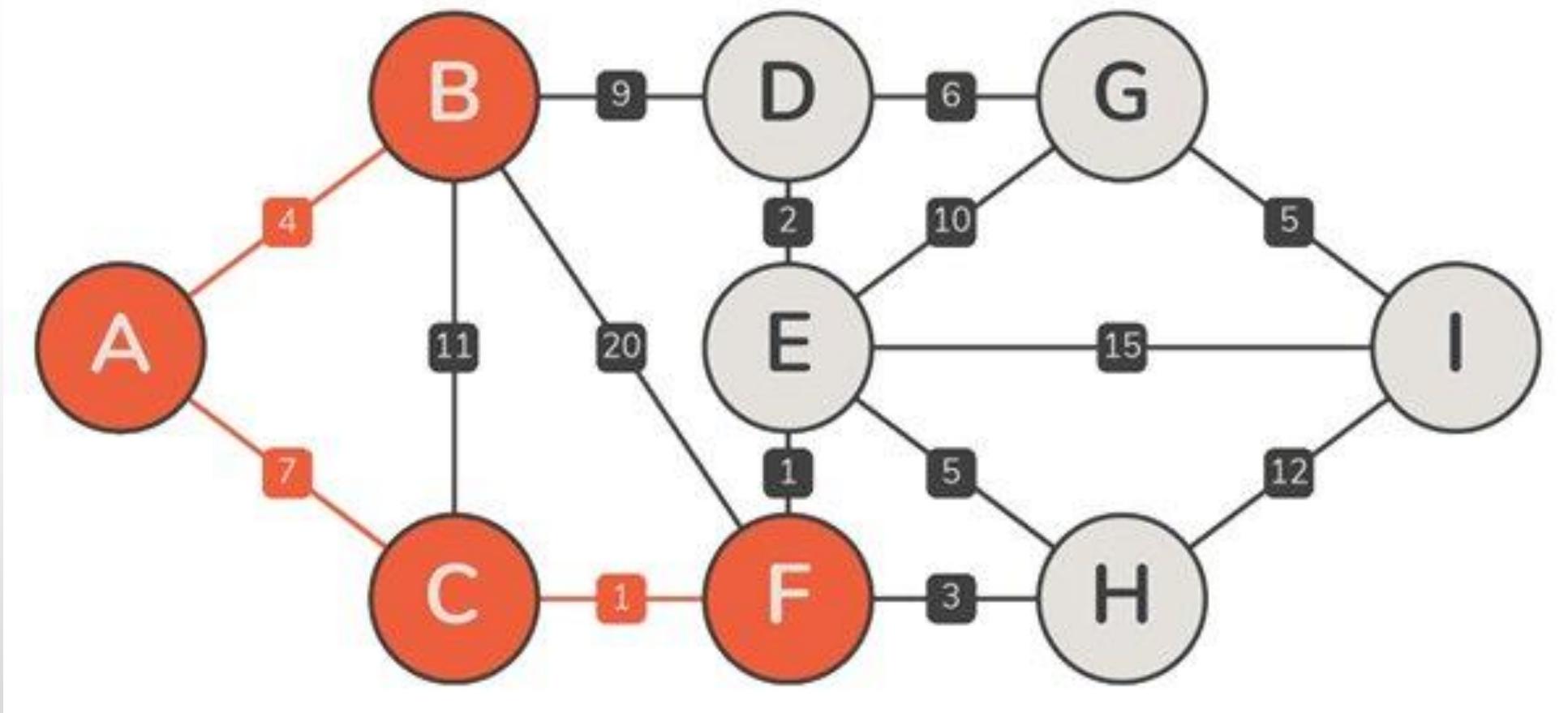


Örnek Prim



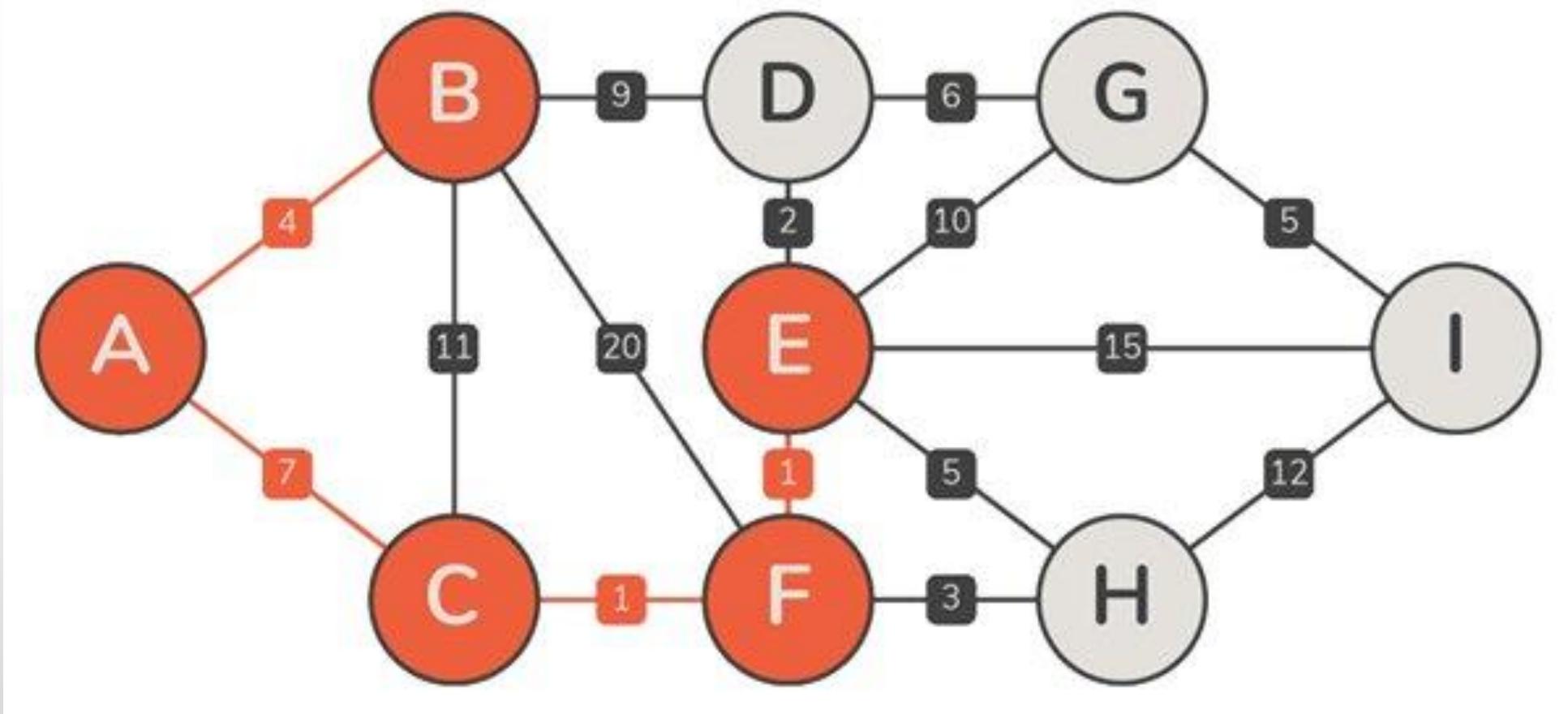


Örnek Prim



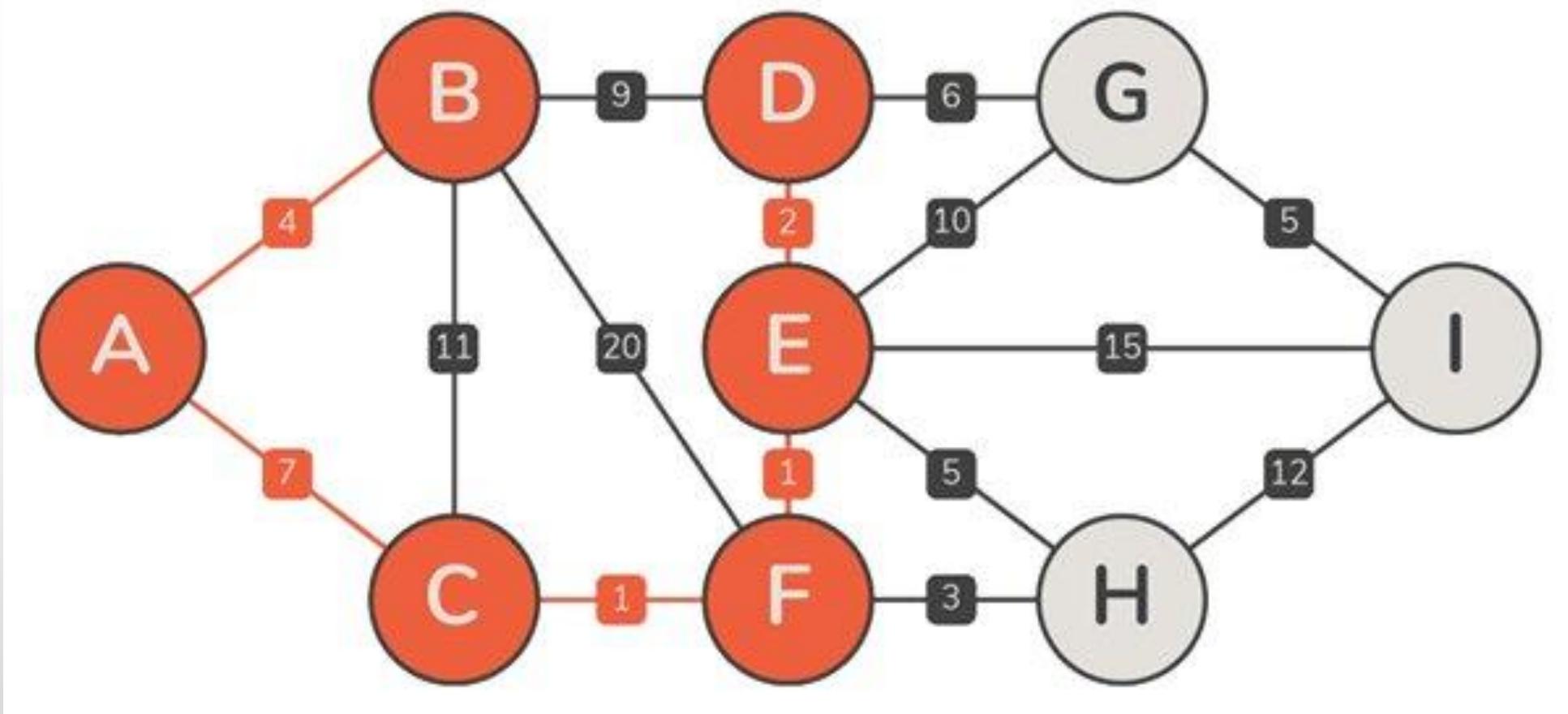


Örnek Prim



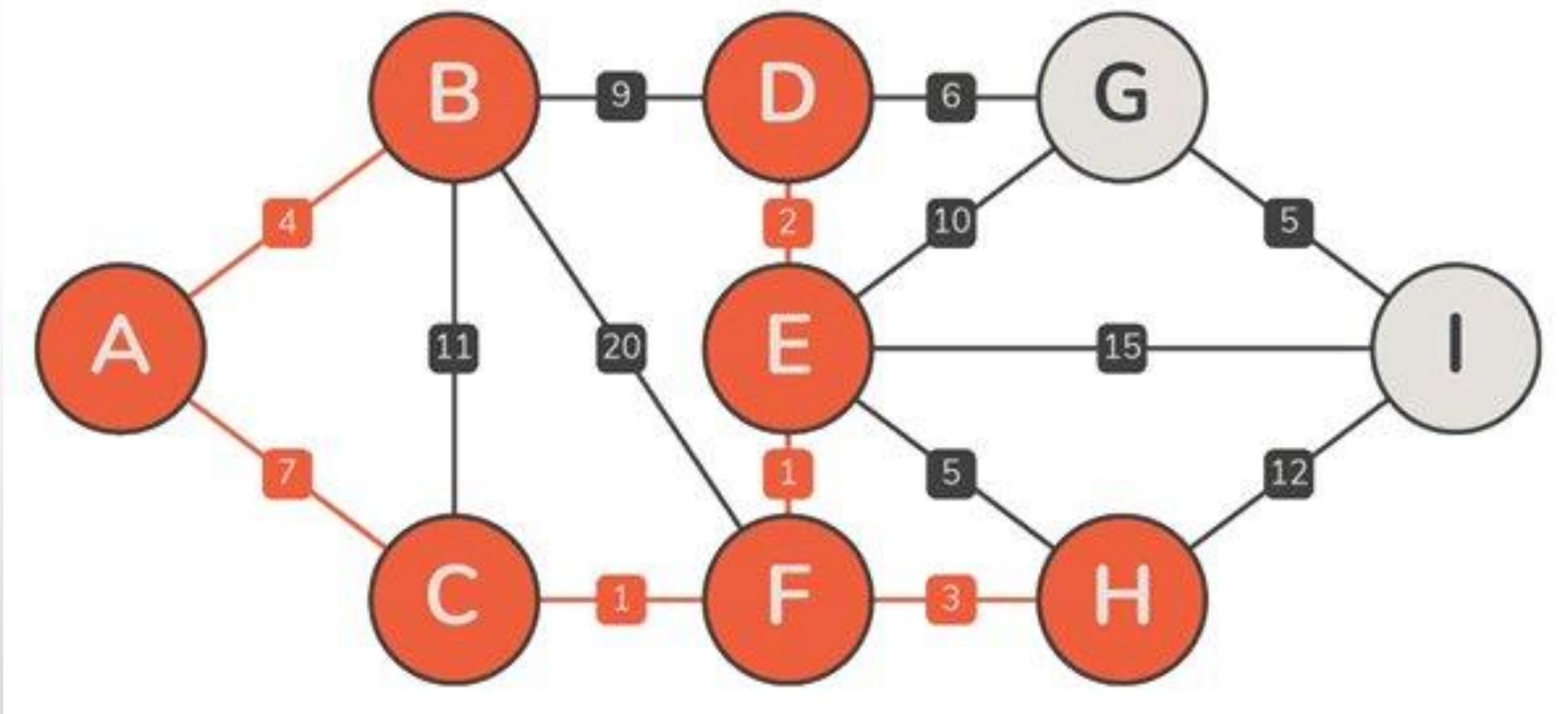


Örnek Prim



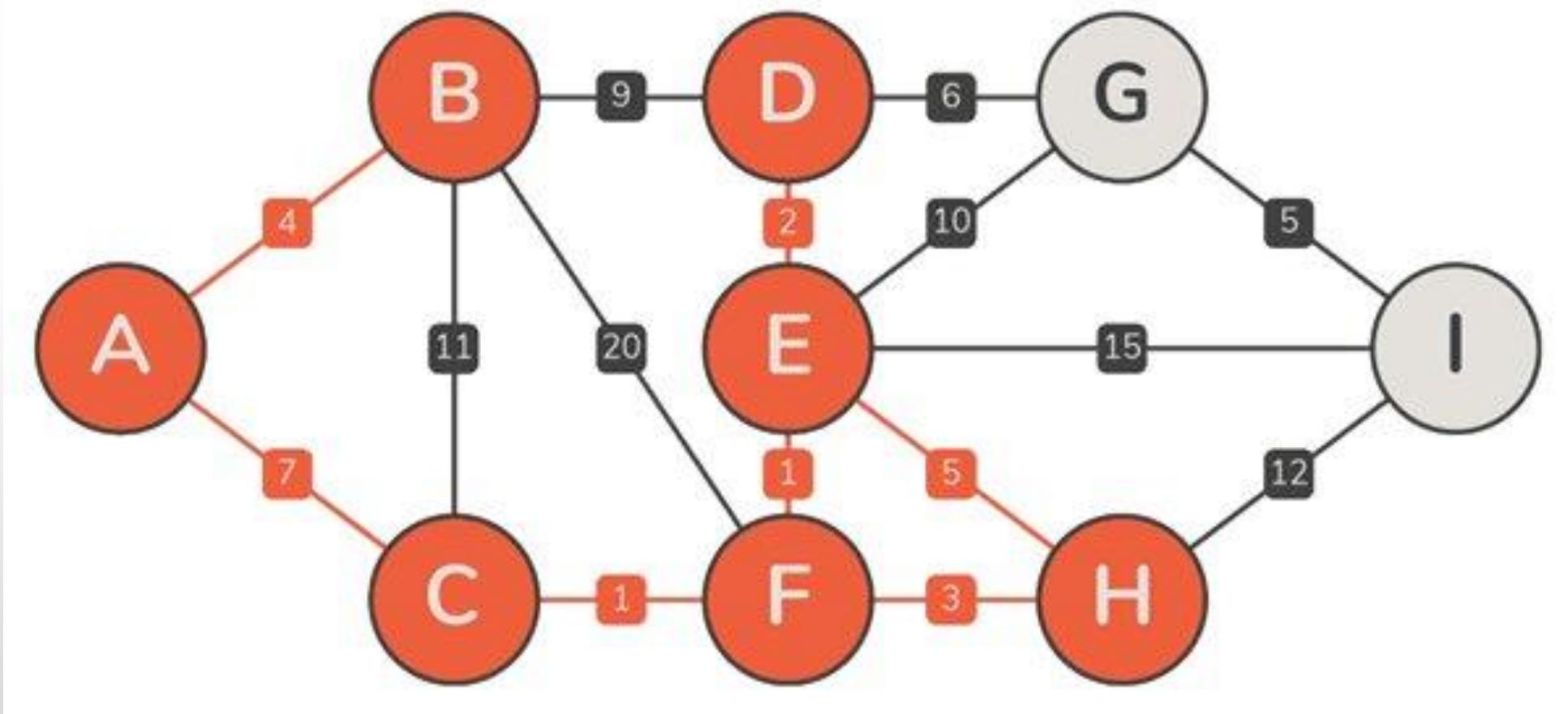


Örnek Prim



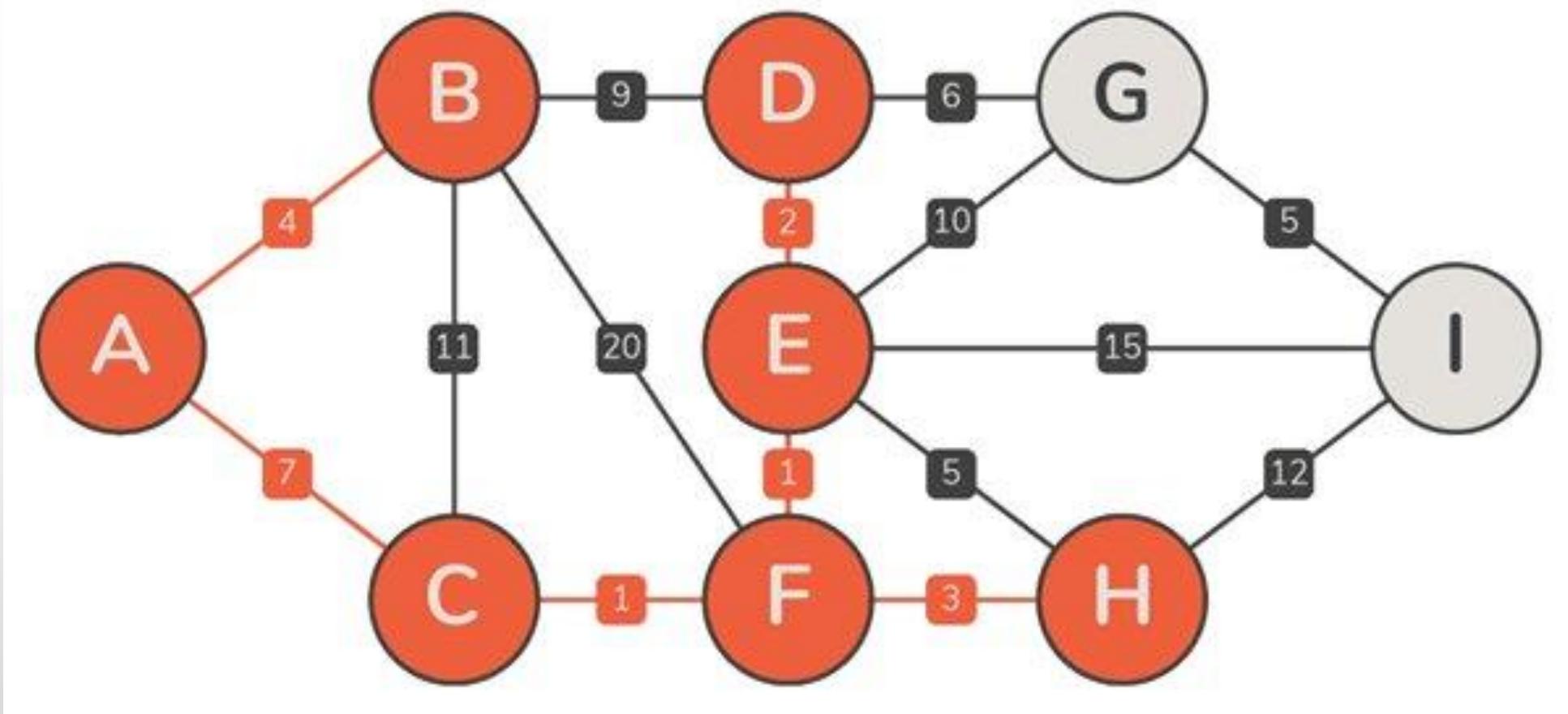


Örnek Prim



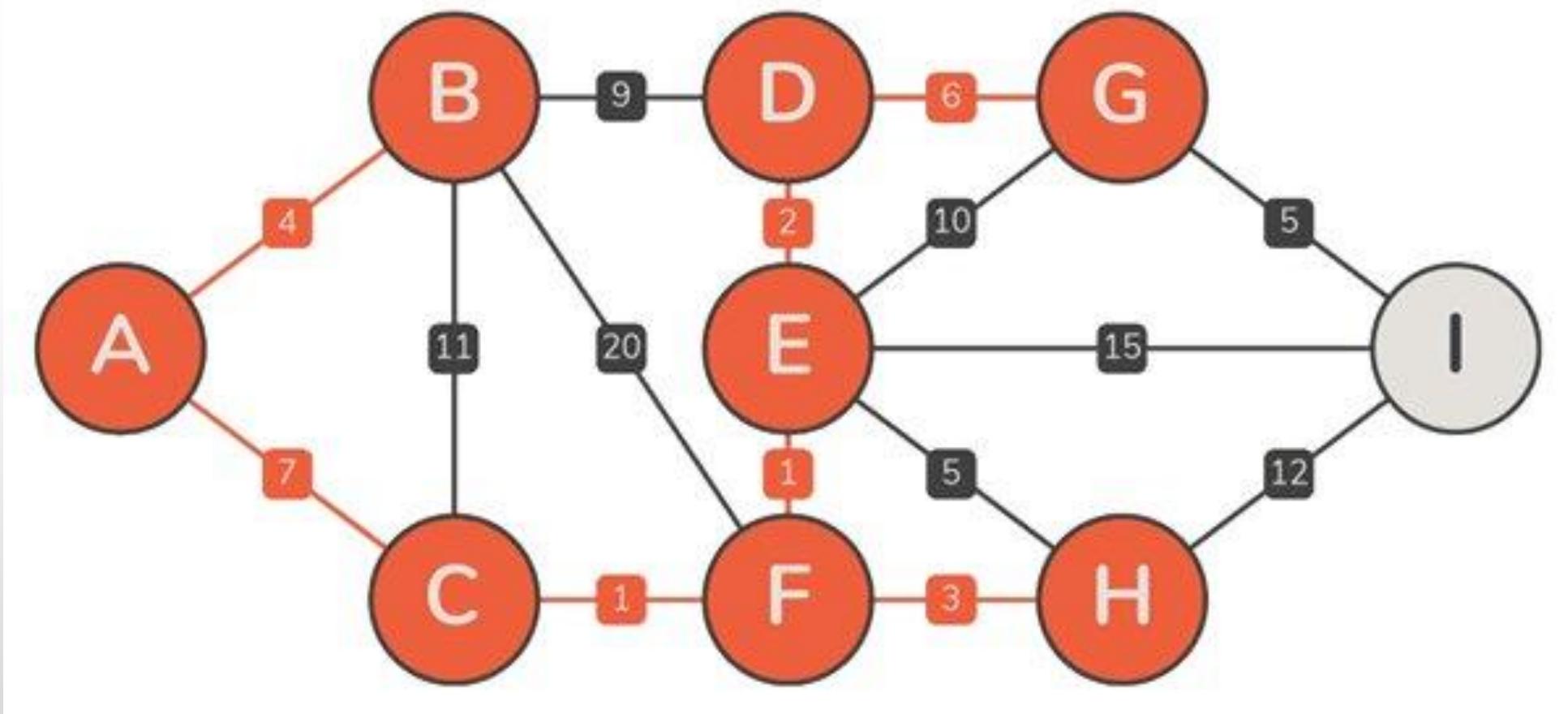


Örnek Prim



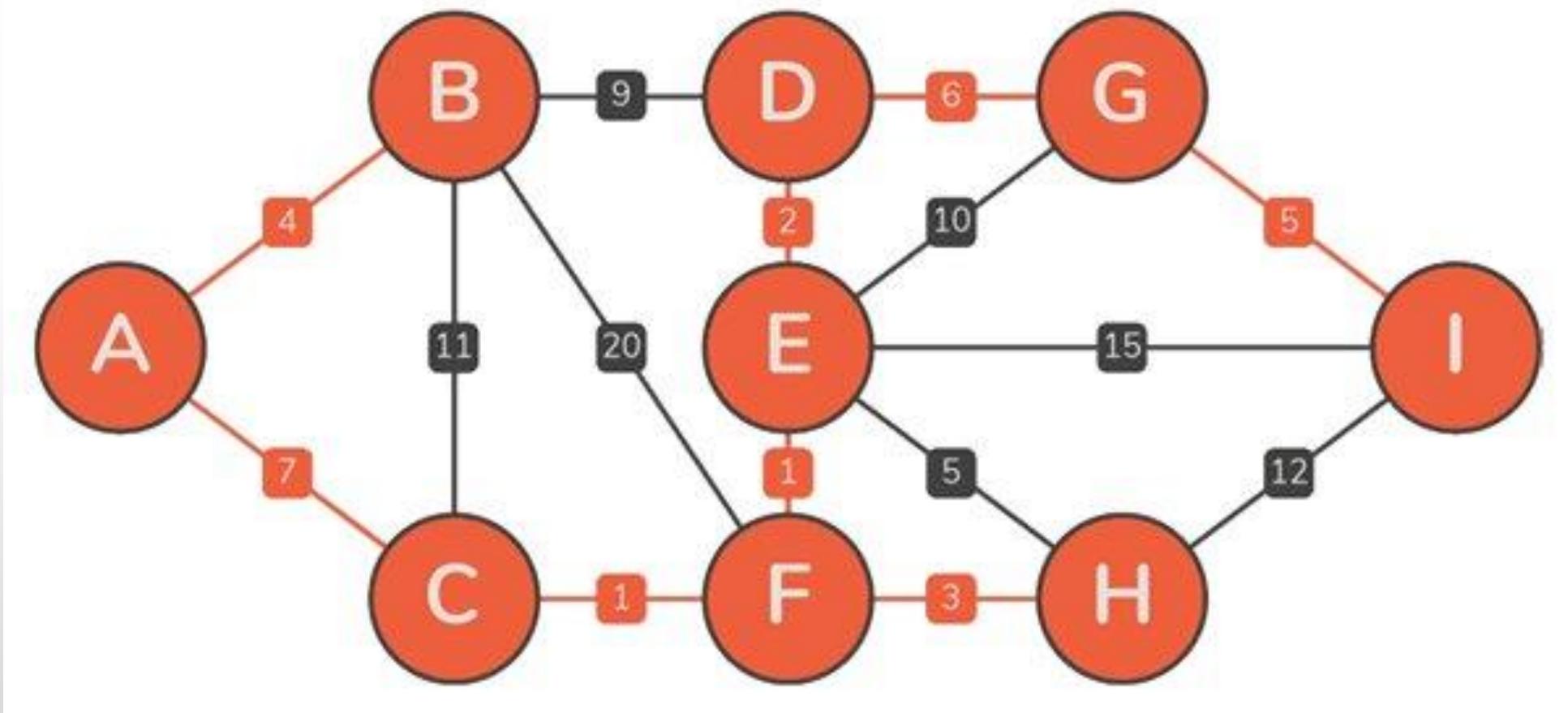


Örnek Prim





Örnek Prim







Boruvska Algoritması

- Minimum kapsayan ağaçları bulmak için kullanılır.
- Çizgenin tüm düğümlerini kapsayan ve ağırlığı minimum alt çizgeyi bulur.
- Nasıl Çalışır?
 - Başlangıçta her düğüm birbirinden bağımsız kabul edilir.
 - Her adımda,
 - en küçük ağırlıklı kenar seçilir ve
 - bu kenarın bağladığı iki düğüm birleştirilir.
 - Bu işlem, tüm düğümler birleşinceye kadar devam eder.



Algoritma Adımları

- Tüm kenarları ağırlıklarına göre sırala.
- Başlangıçta her düğümü tek başına bir küme olarak ele al.
- En küçük ağırlıklı kenarı seç ve bu kenarı bağlayan düğümleri birleştir.
- Her adımda oluşan alt küme ağaçının ağırlığını güncelle.
- Tüm düğümler birleşinceye kadar 3 ve 4. adımları tekrarla.

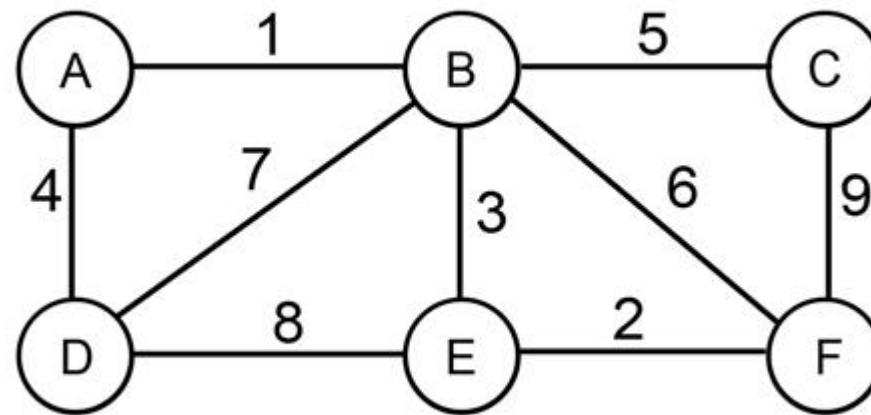


Özellikleri

- Ağırlık sıralaması yaparak minimum ağaçları bulur.
- Greedy (Açgözlü) bir algoritmadır.
- Karmaşıklığı $O(E \log V)$ 'dir,
 - E kenar sayısı,
 - V düğüm sayısıdır.

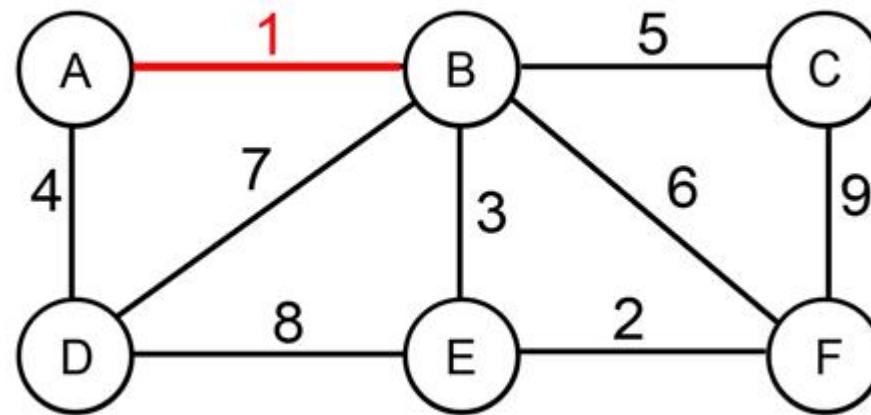


Örnek Boruvka



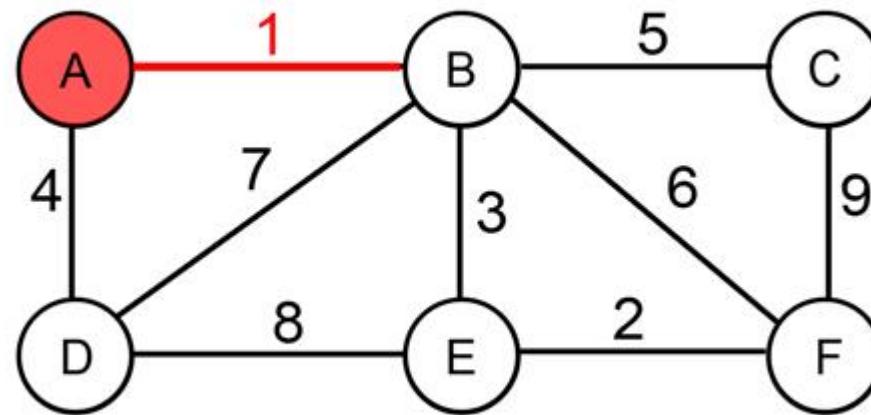


Örnek Boruvka



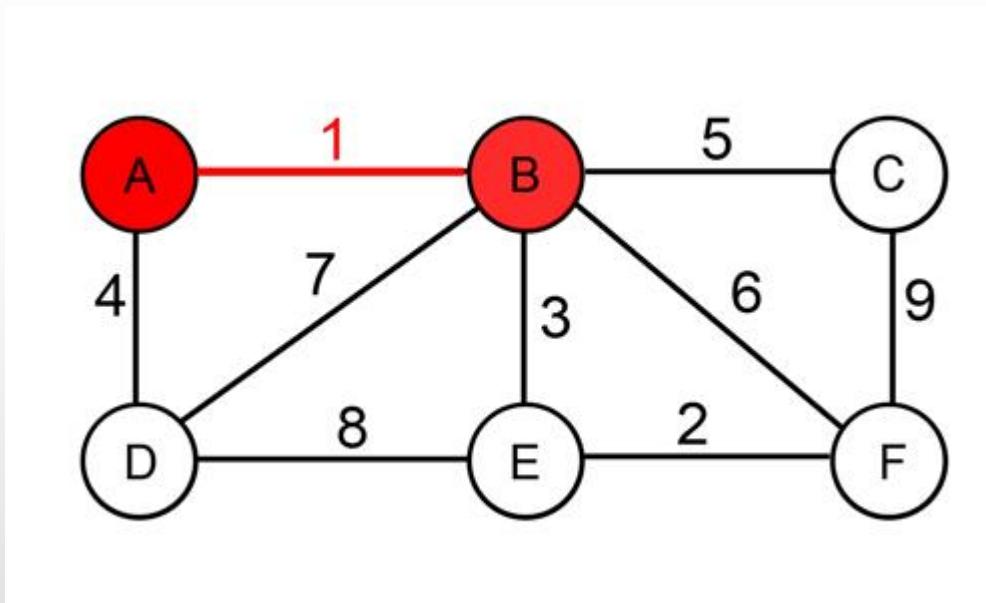


Örnek Boruvka



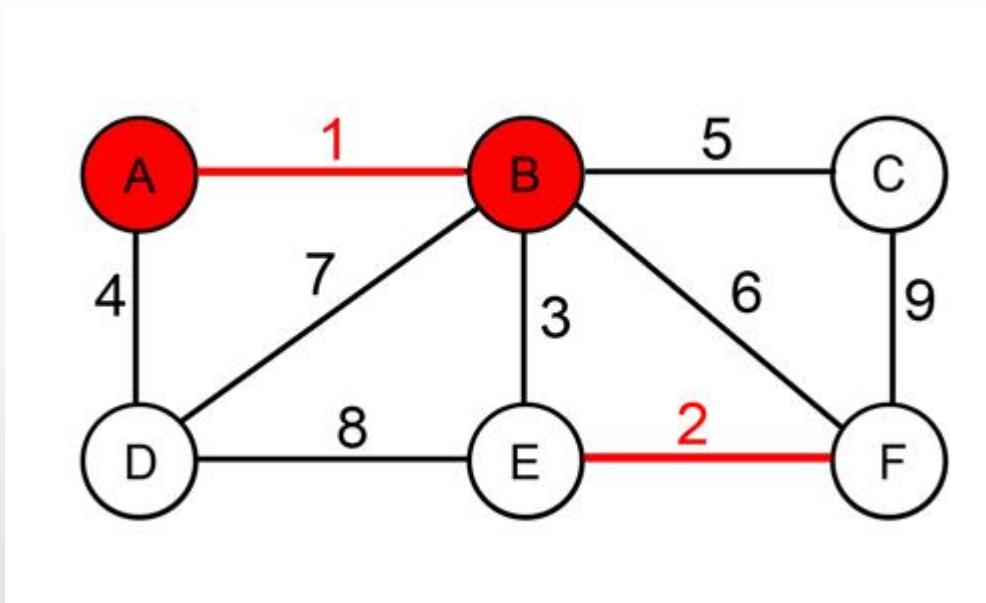


Örnek Boruvka



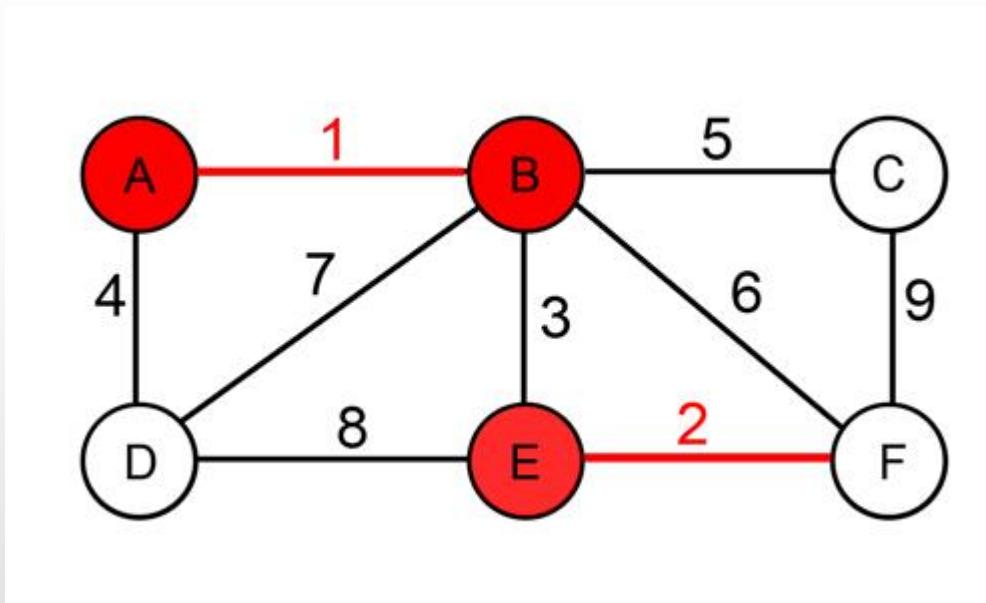


Örnek Boruvka



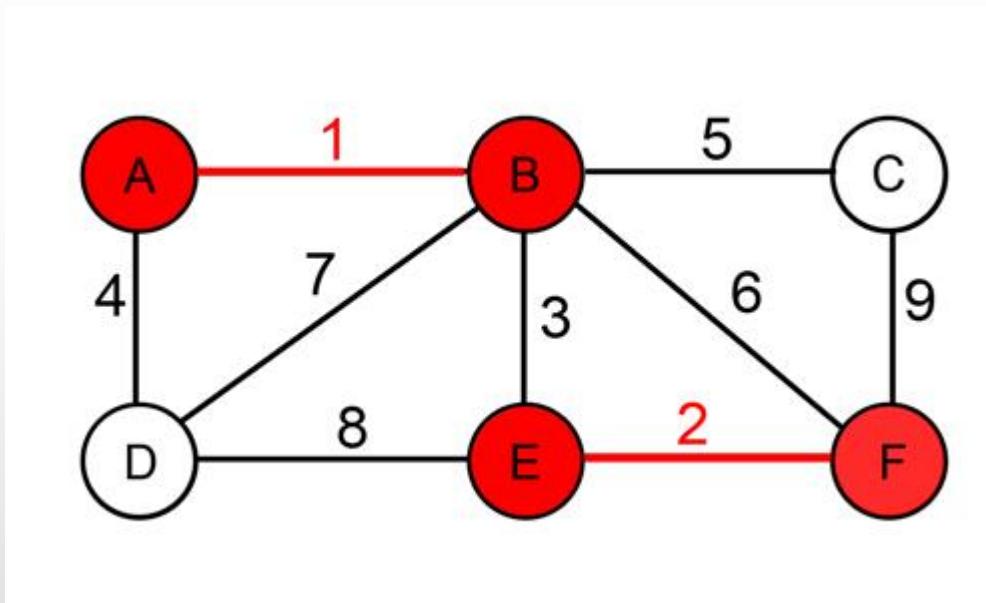


Örnek Boruvka



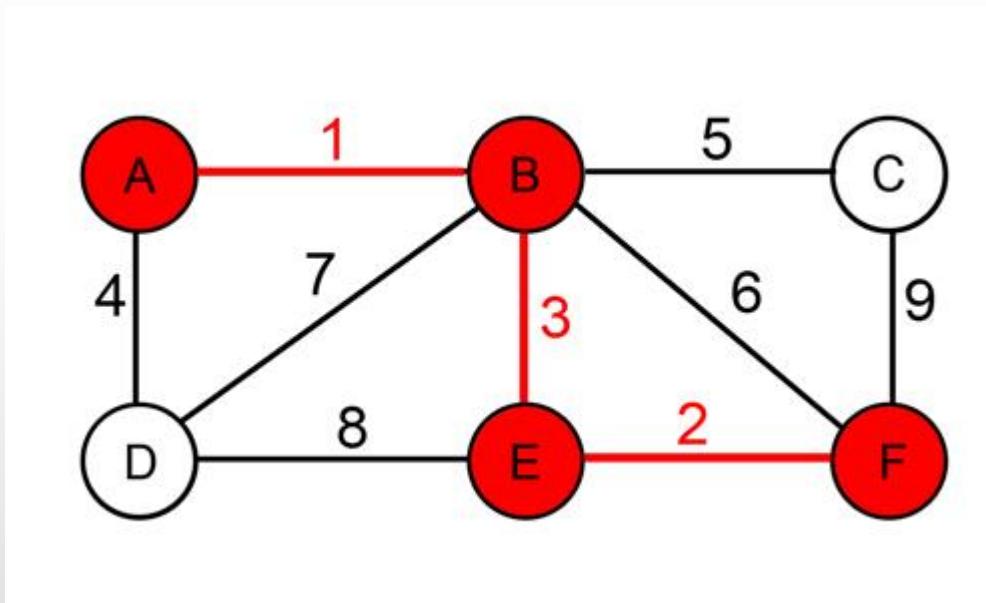


Örnek Boruvka



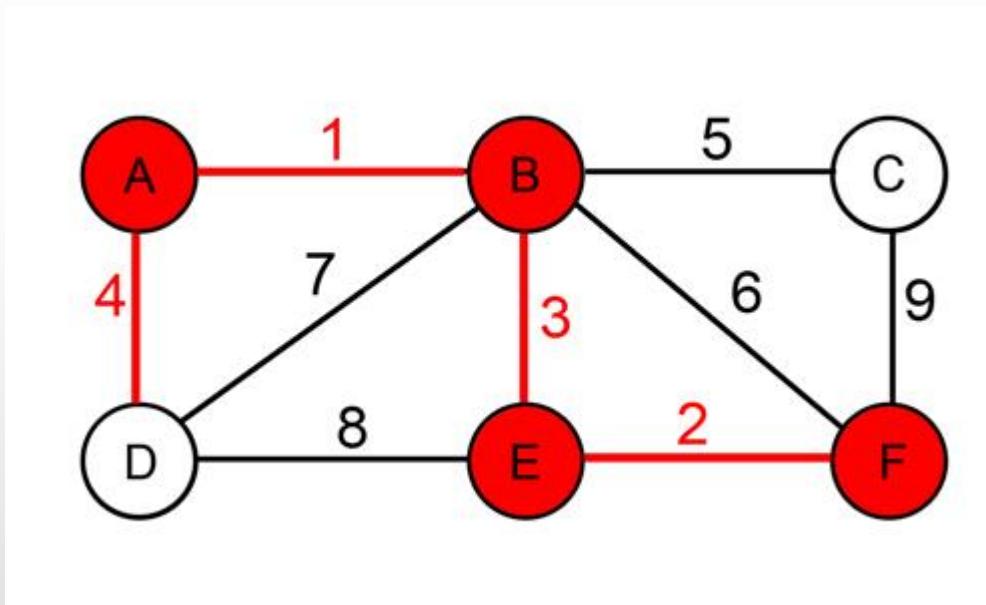


Örnek Boruvka



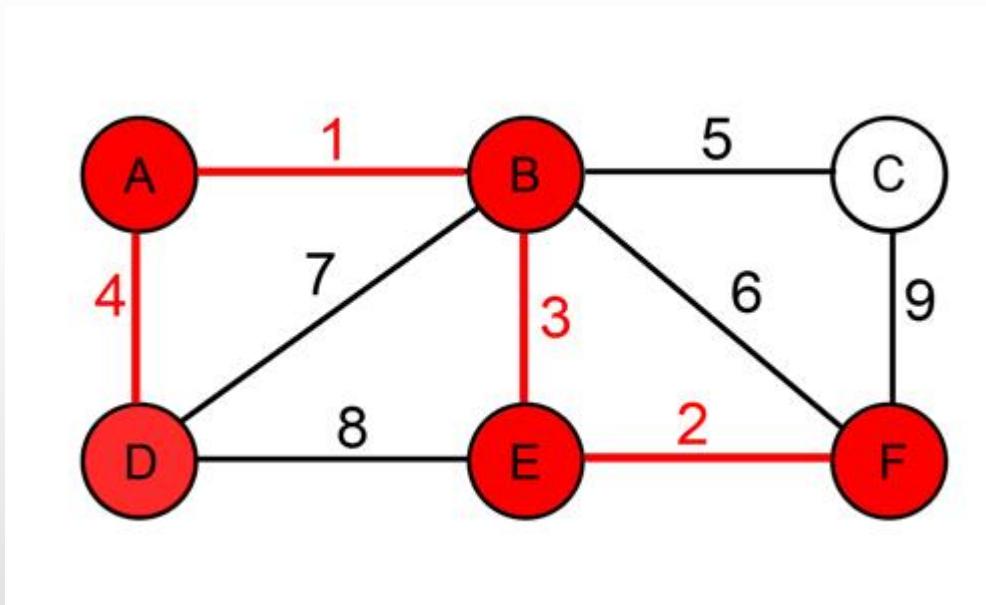


Örnek Boruvka



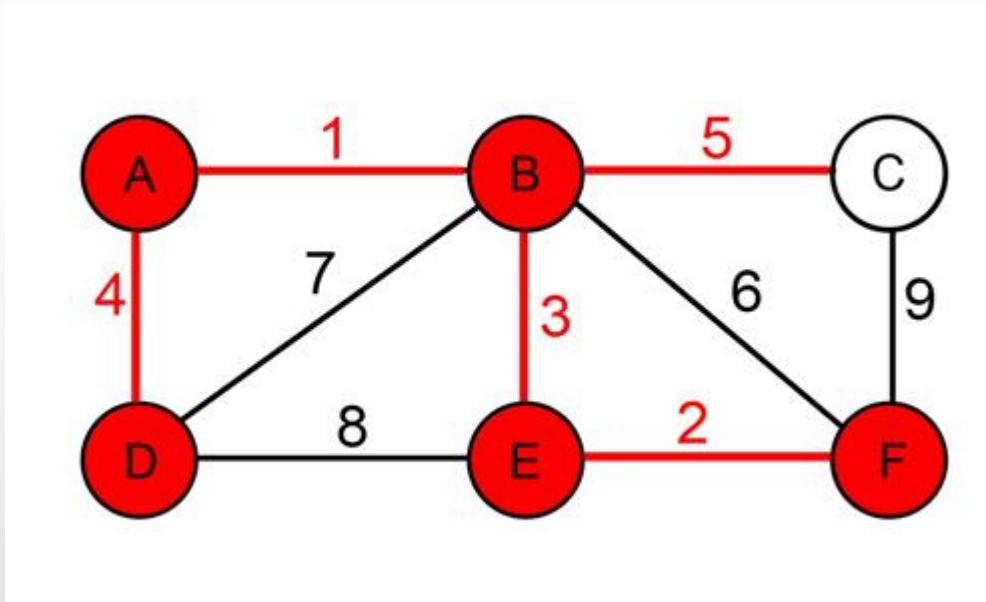


Örnek Boruvka



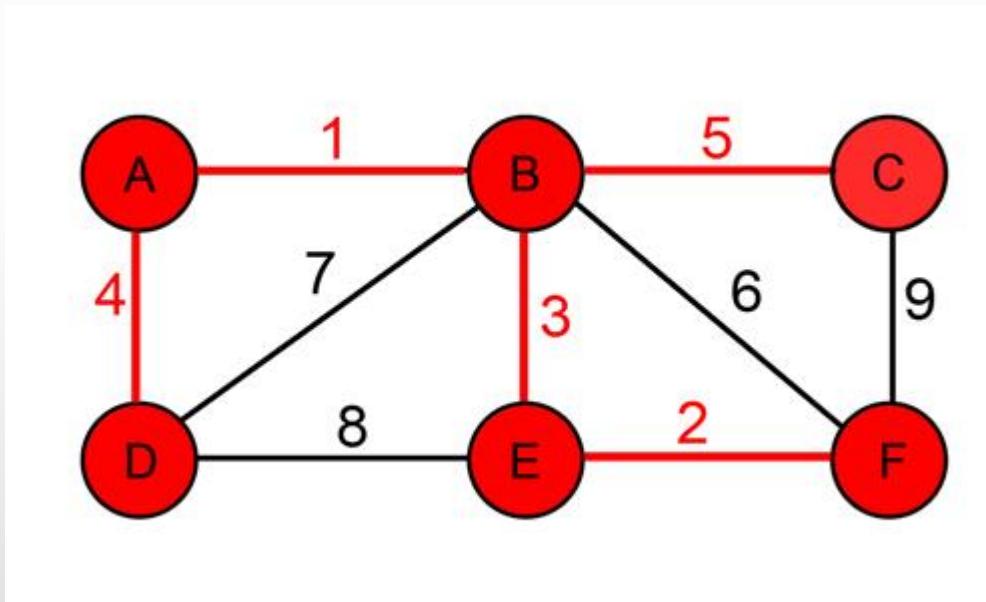


Örnek Boruvka



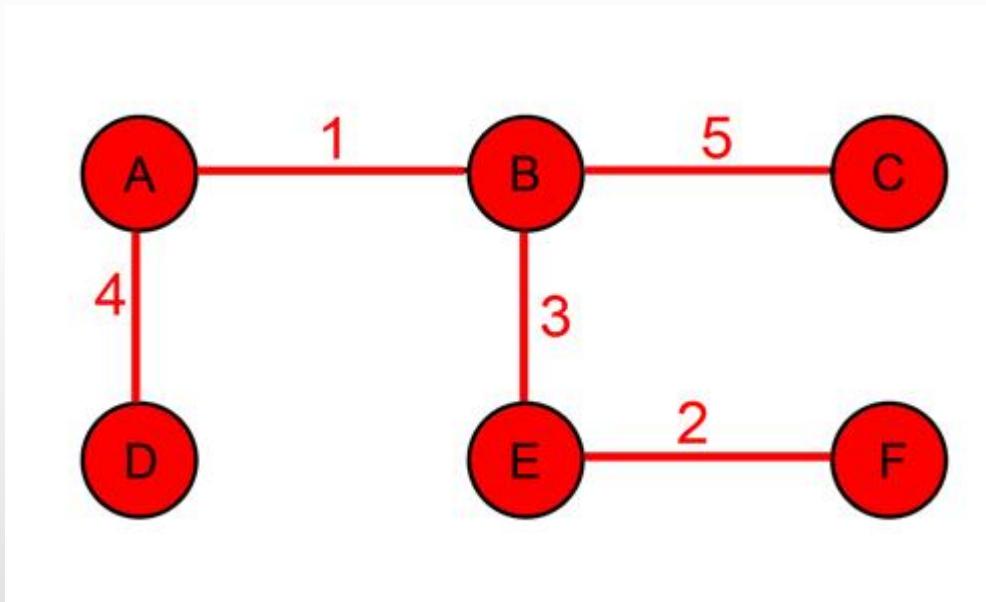


Örnek Boruvka





Örnek Boruvka







Sollin Algoritması

- Başlangıçta her düğüm birbirinden bağımsız olarak kabul edilir.
- Her adımda,
 - her düğüm için en düşük ağırlıklı kenar seçilir ve
 - bu kenarın bağladığı düğümler birleştirilir.
- Bu işlem, tüm düğümler birleşinceye kadar devam eder.



Algoritma Adımları

- Başlangıçta her düğümü tek başına bir küme olarak ele al.
- Her düğüm için, komşu kenarlar arasından en düşük ağırlıklı olanı seç.
- Seçilen kenarların her iki ucundaki düğümleri birleştir.
- Her adımda oluşan alt küme ağaçının ağırlığını güncelle.
- Tüm düğümler birleşinceye kadar 2, 3 ve 4. adımları tekrarla.

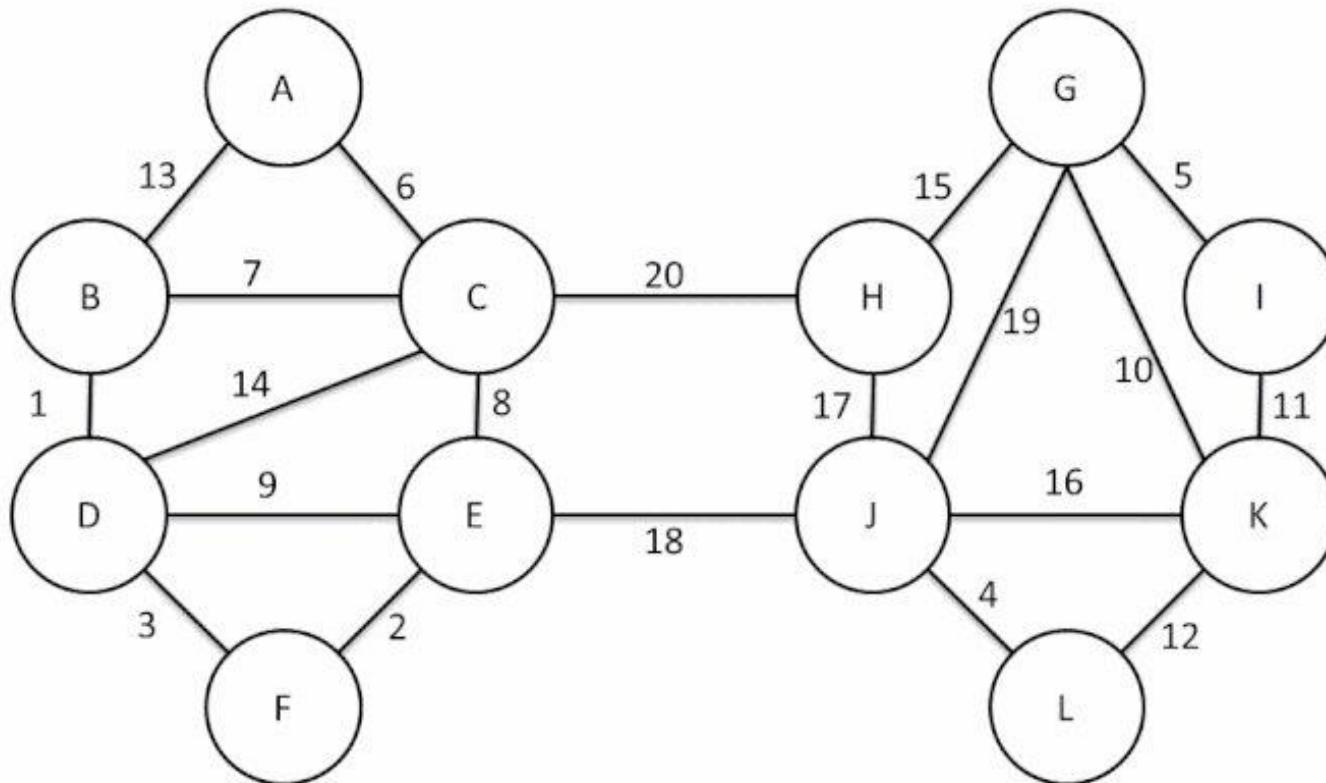


Özellikleri

- Paralel işlem gücünden yararlanabilir.
- Diğer algoritmalarla kıyasla daha hızlı çalışabilir.
- Zaman karmaşıklığı $O(E + V \log V)$ 'dir,
 - E kenar sayısı,
 - V düğüm sayısıdır.

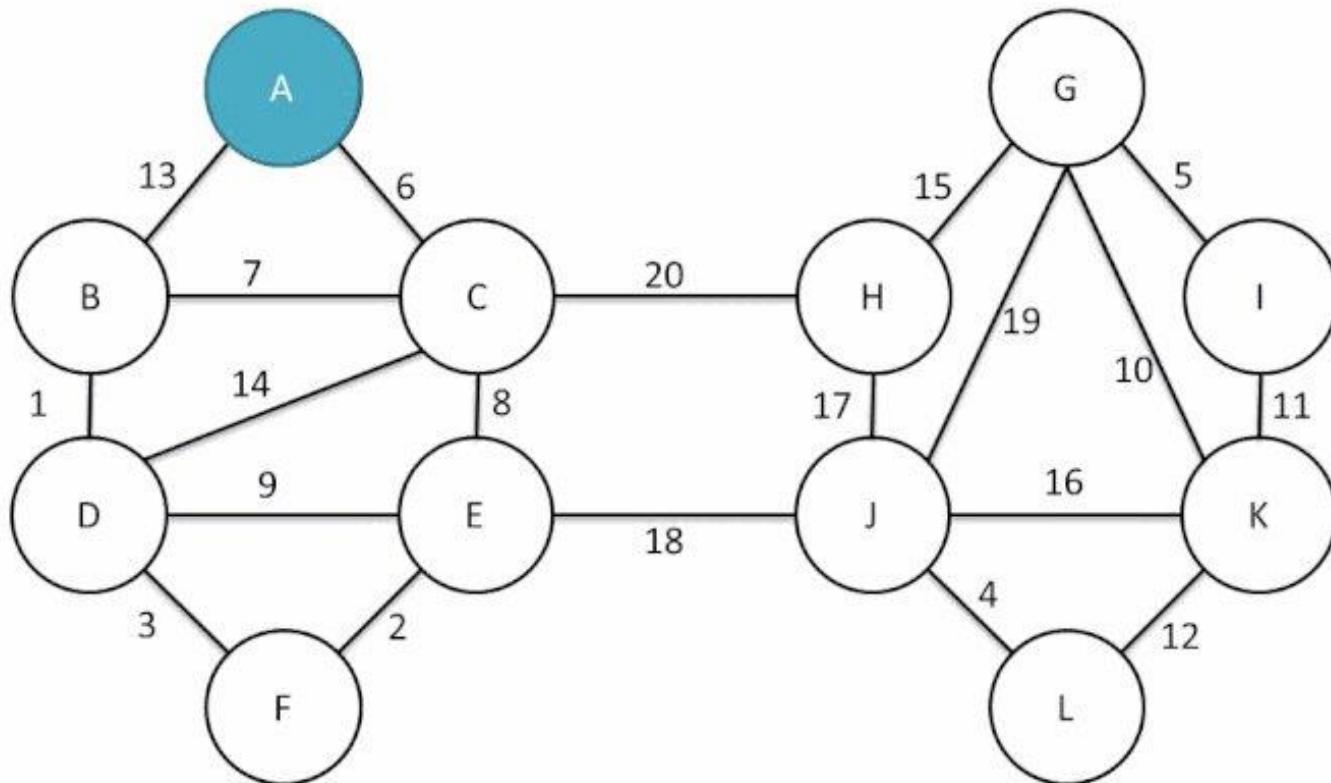


Boruvka's Algorithm



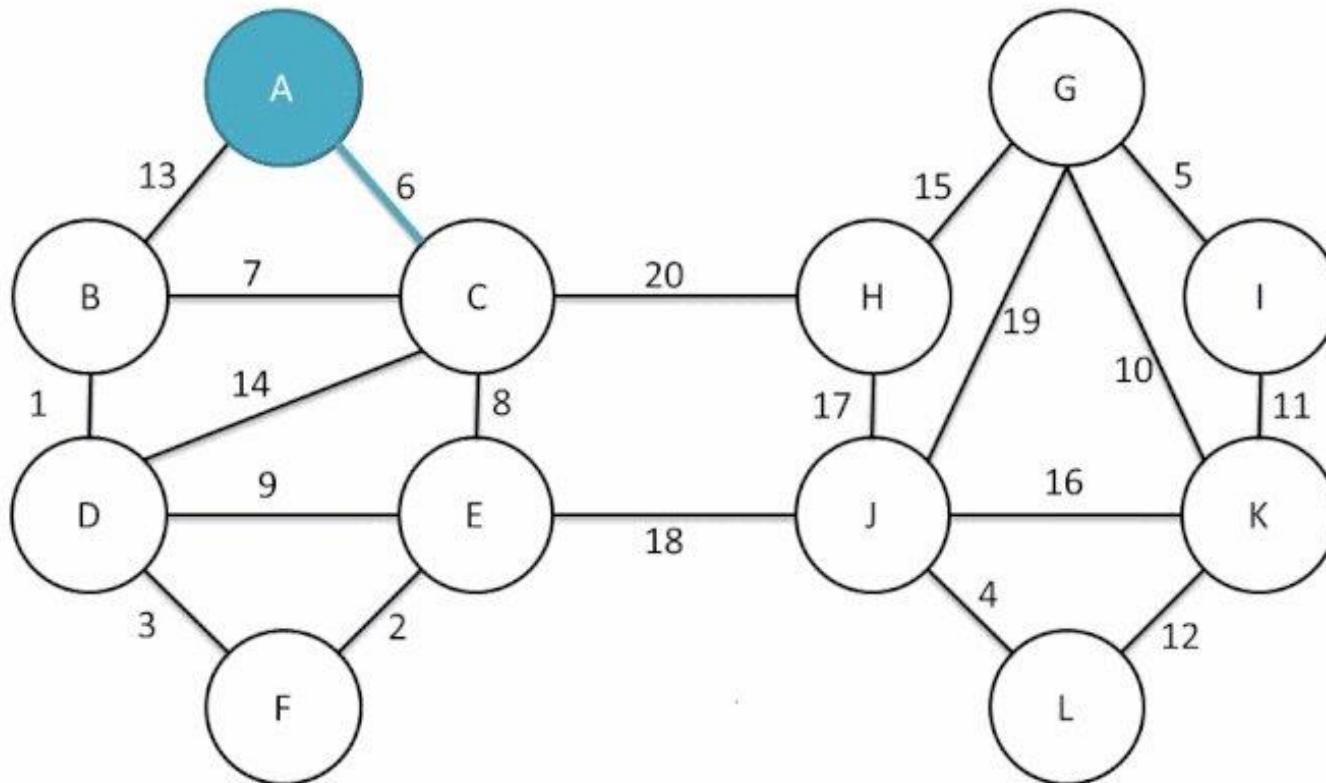


Boruvka's Algorithm



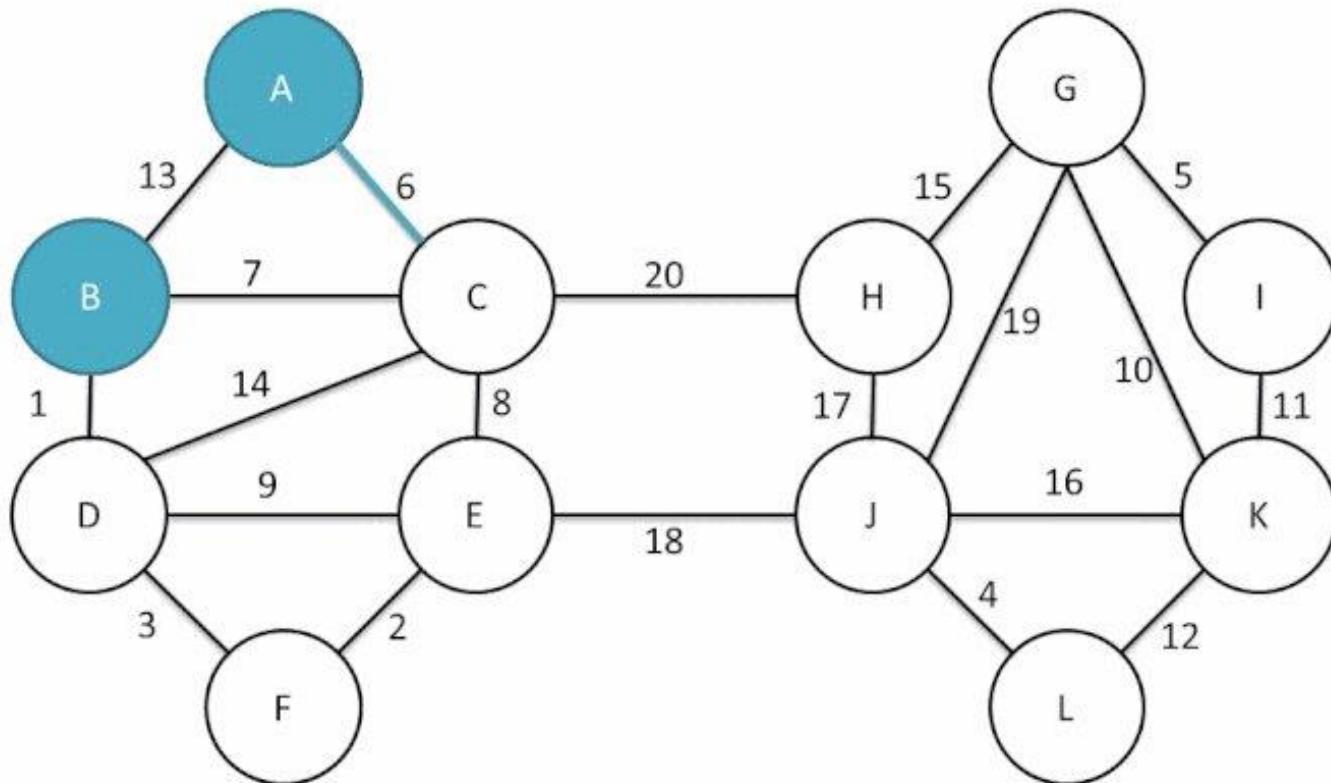


Boruvka's Algorithm



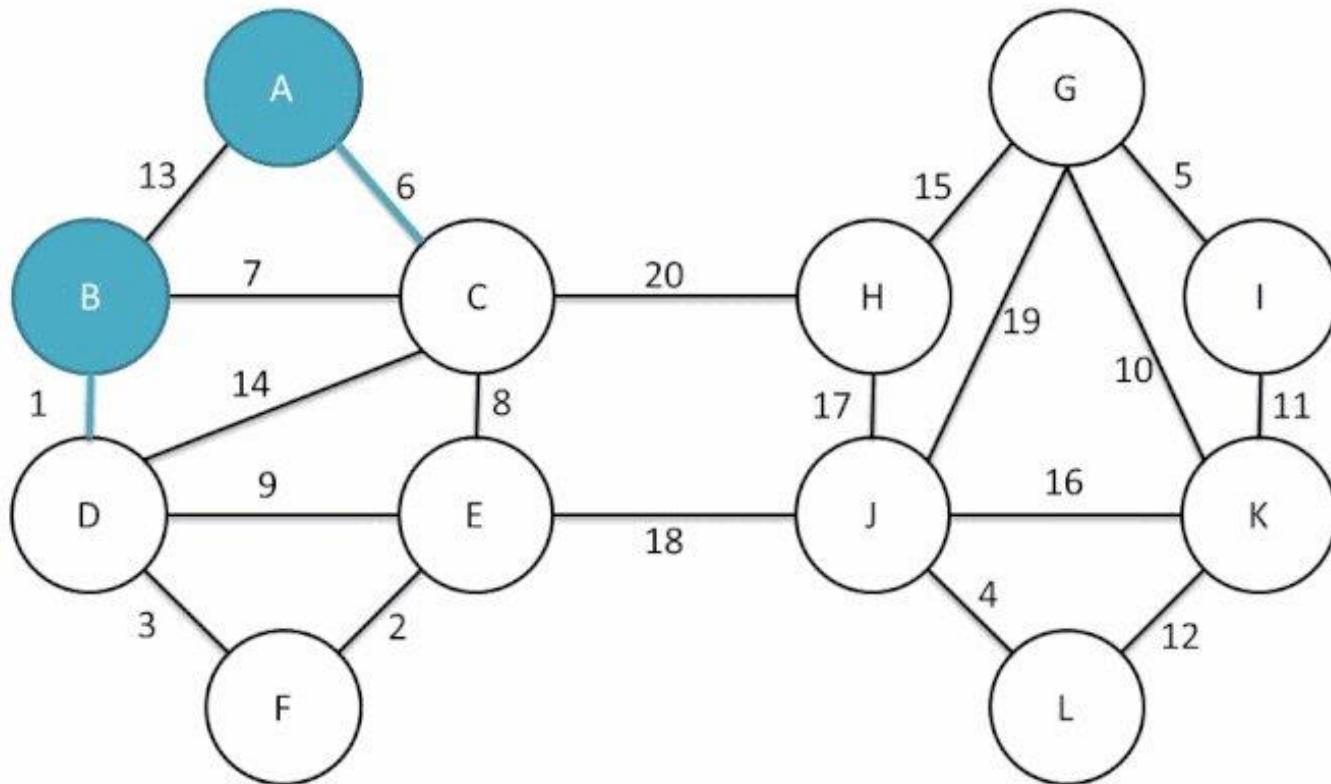


Boruvka's Algorithm



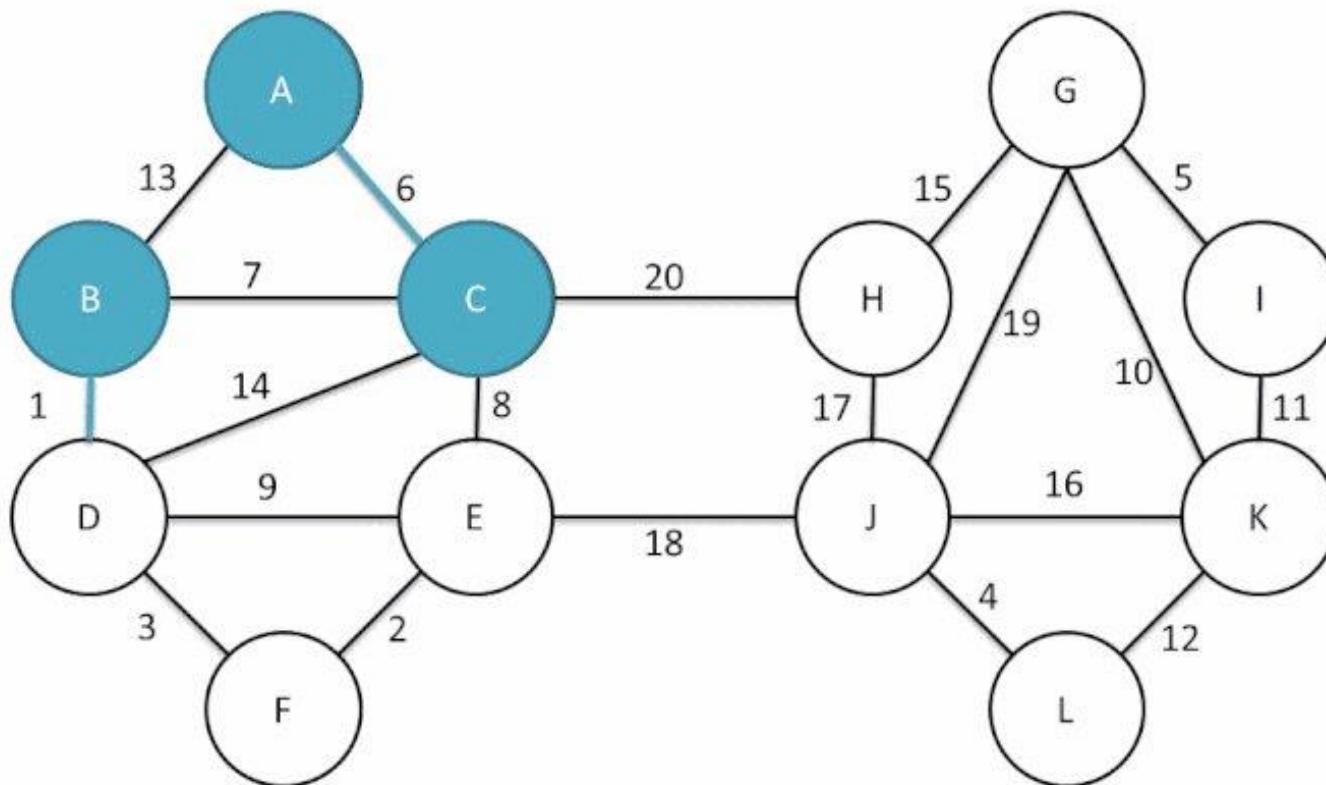


Boruvka's Algorithm



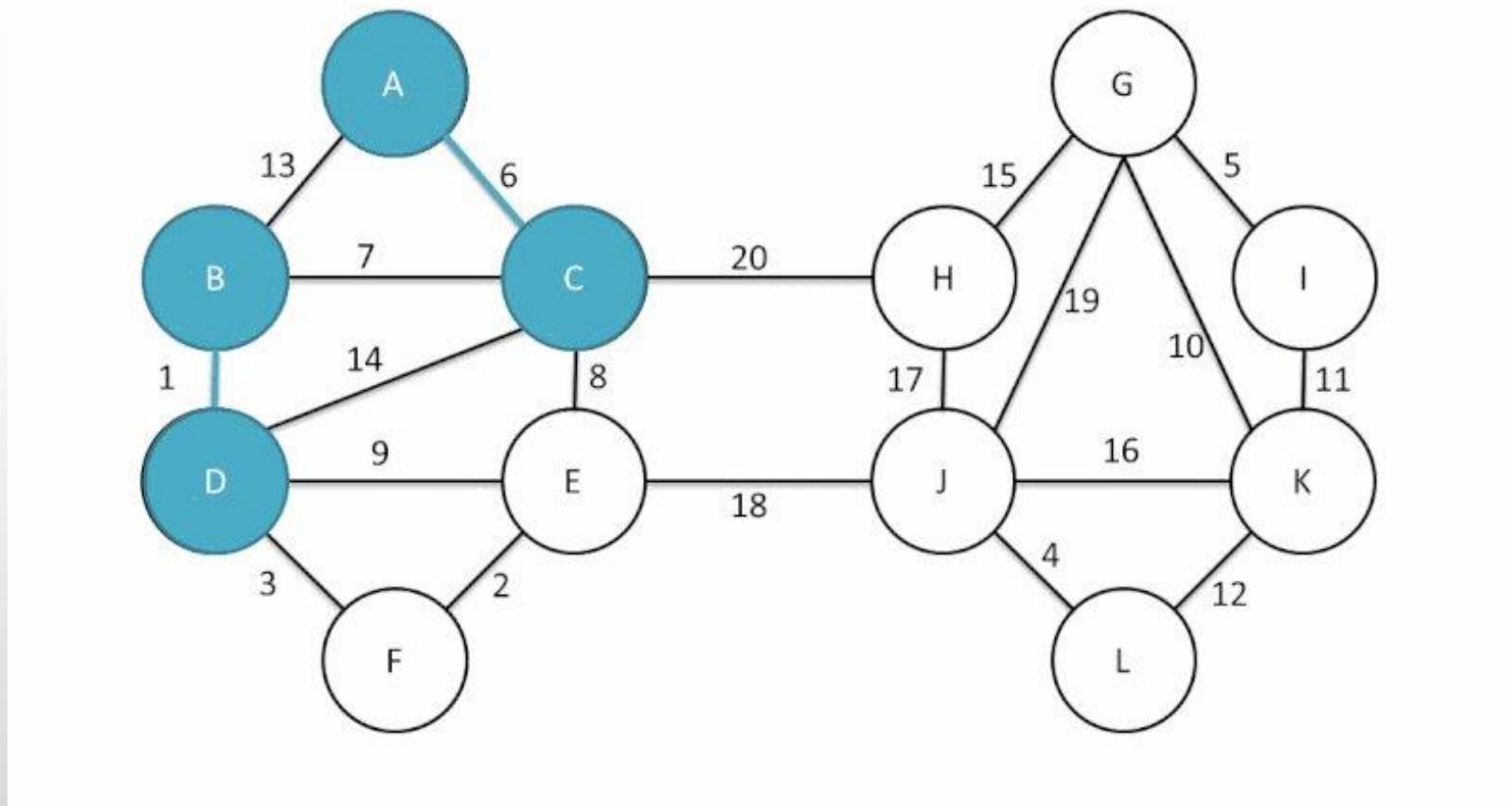


Boruvka's Algorithm



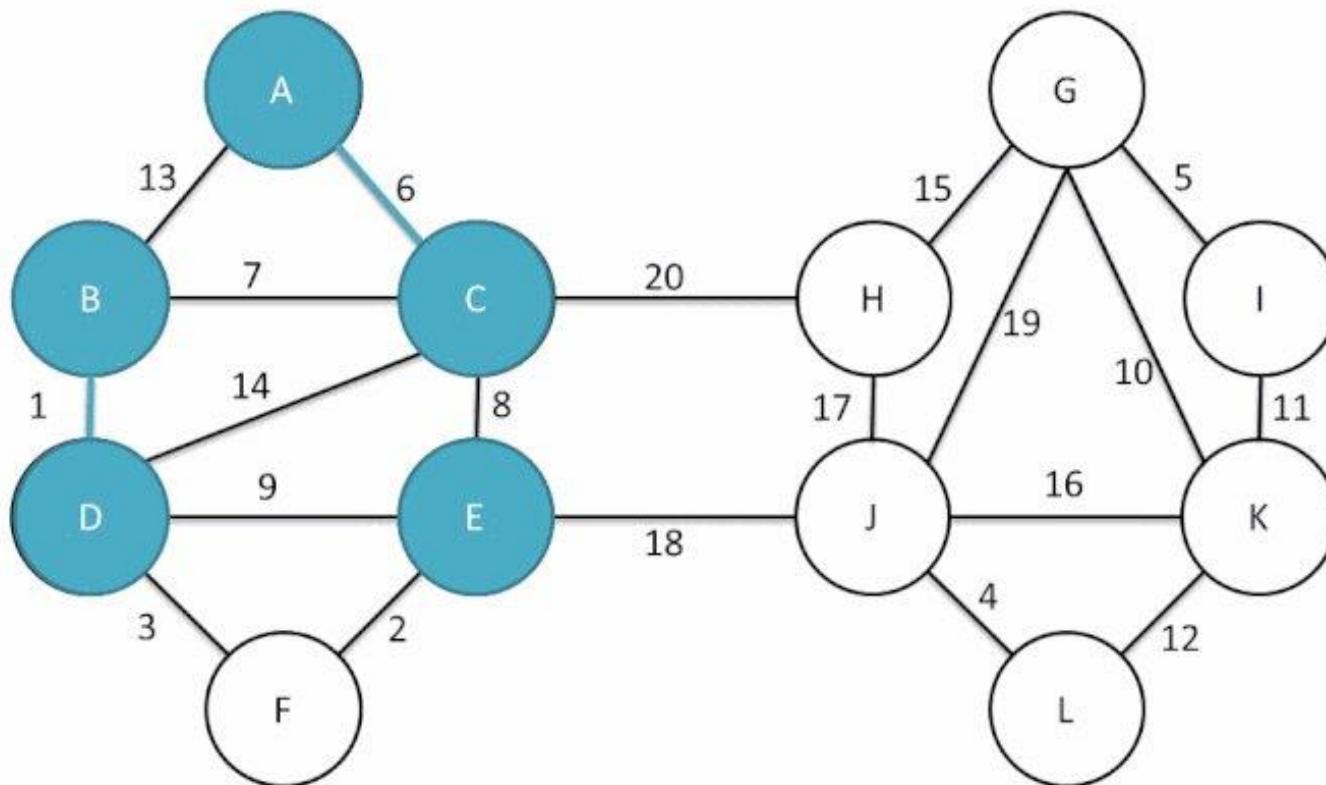


Boruvka's Algorithm



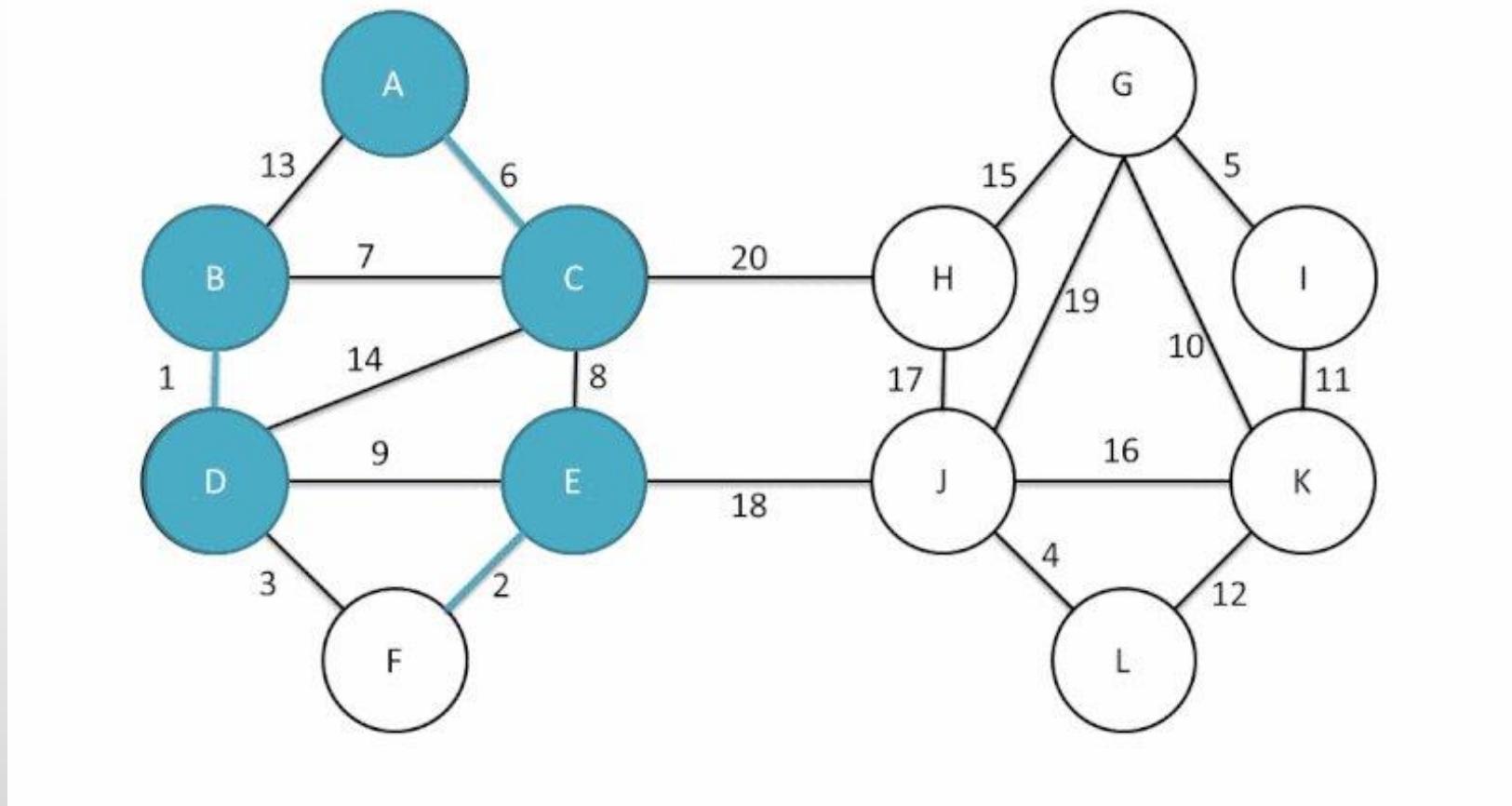


Boruvka's Algorithm



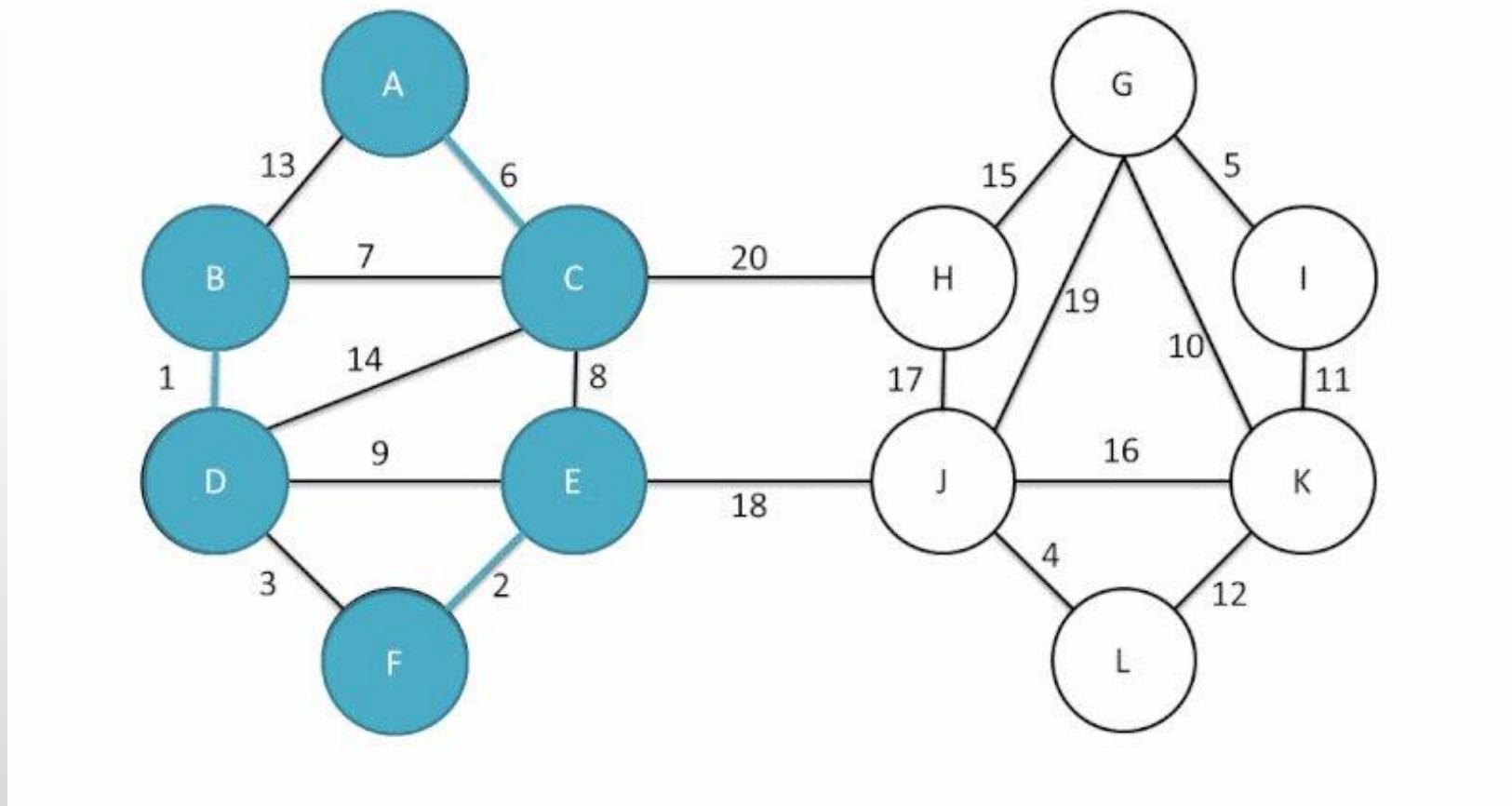


Boruvka's Algorithm



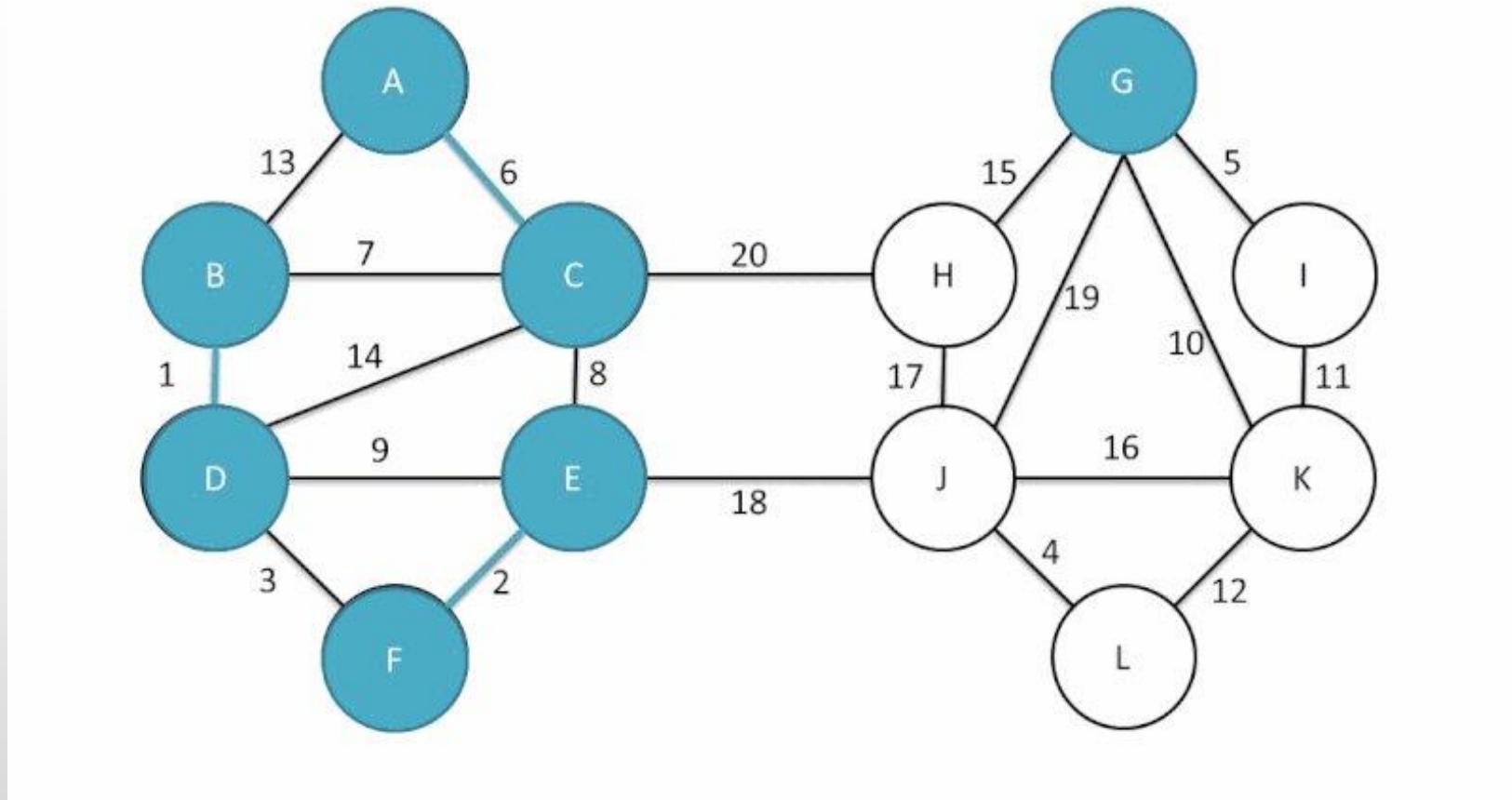


Boruvka's Algorithm



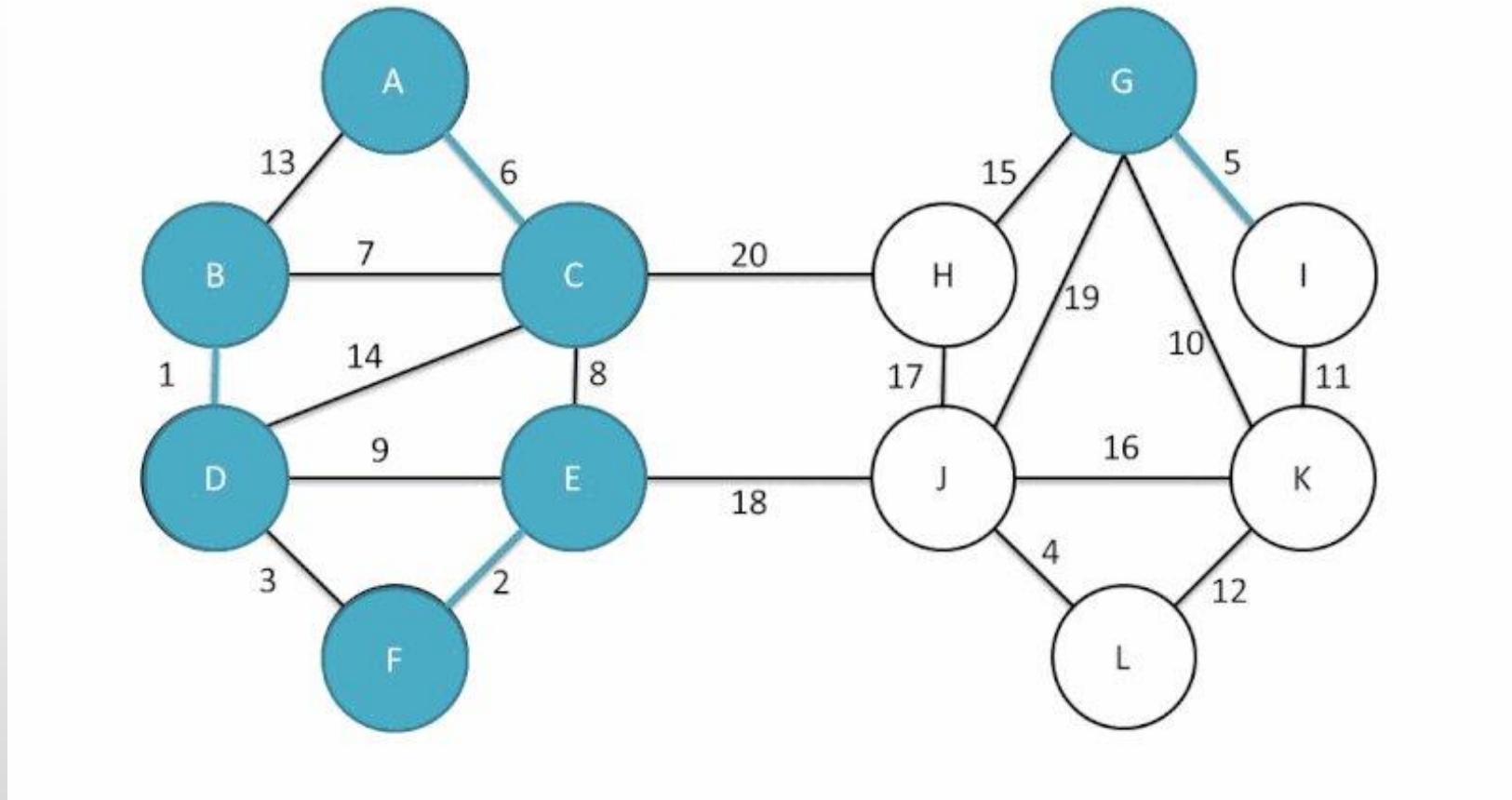


Boruvka's Algorithm



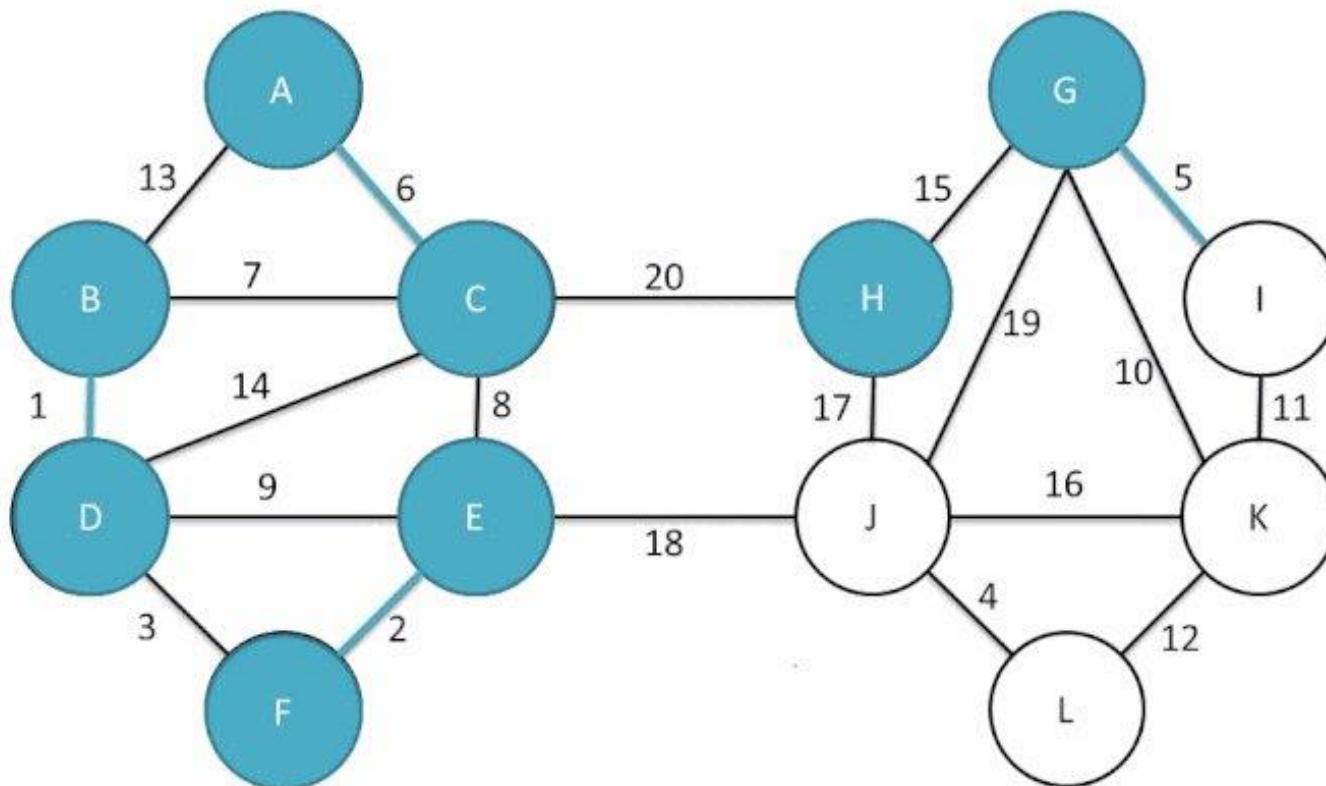


Boruvka's Algorithm



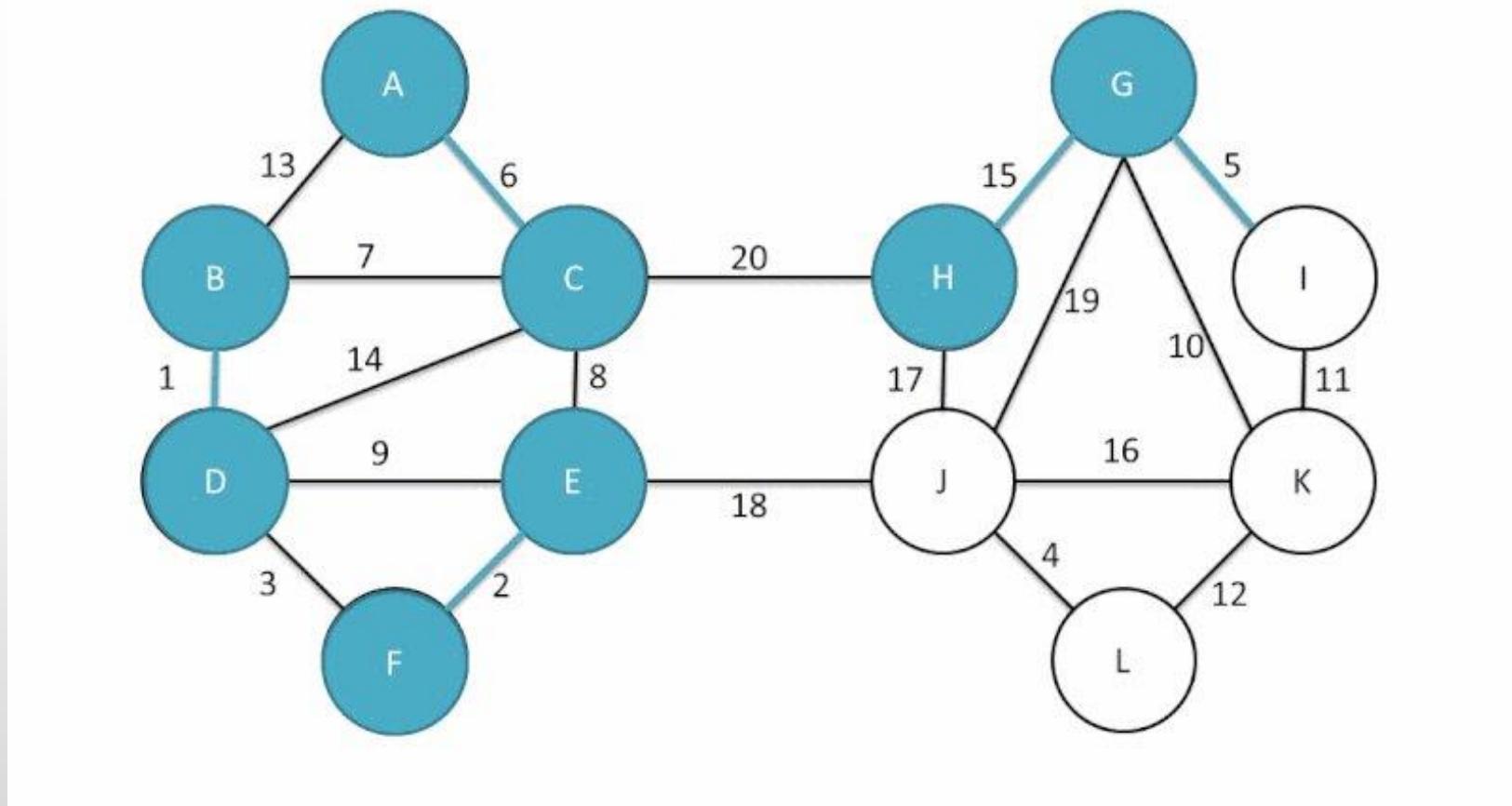


Boruvka's Algorithm



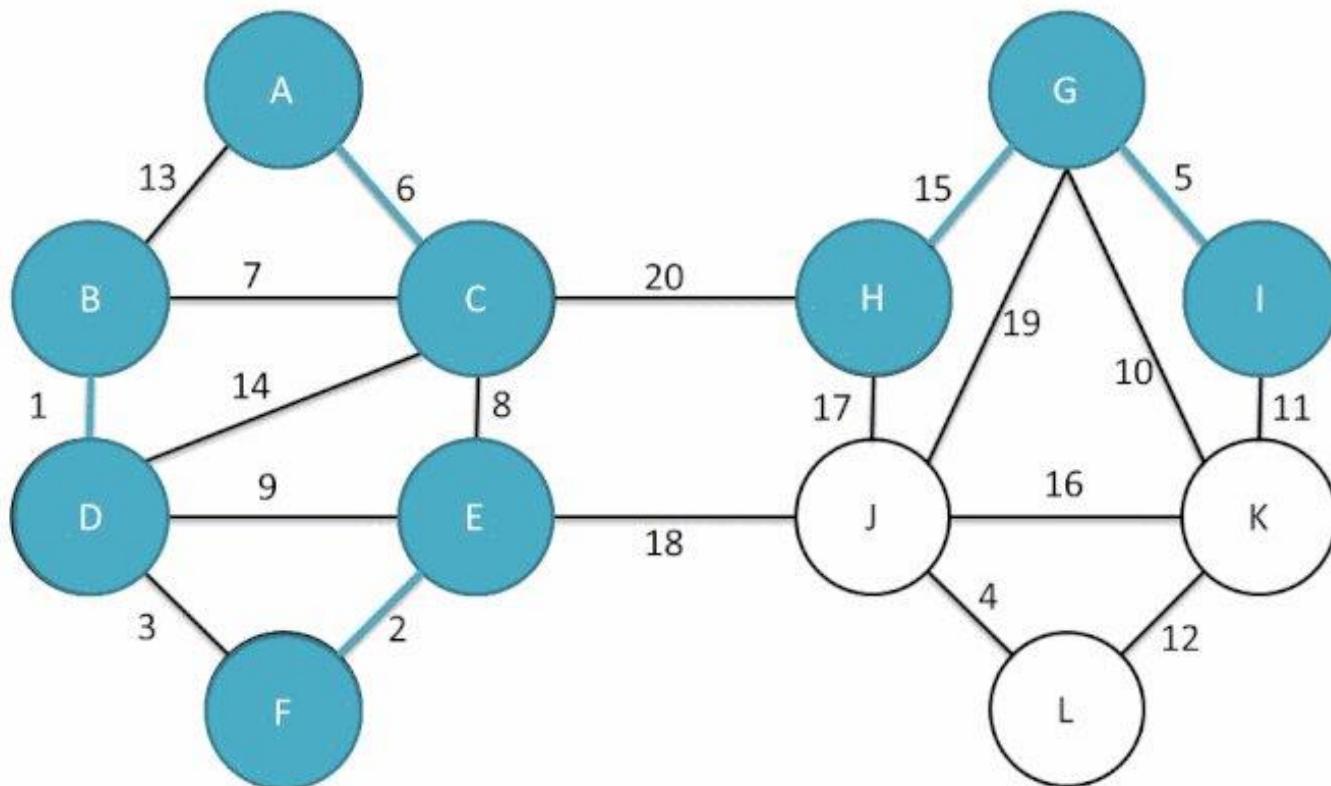


Boruvka's Algorithm



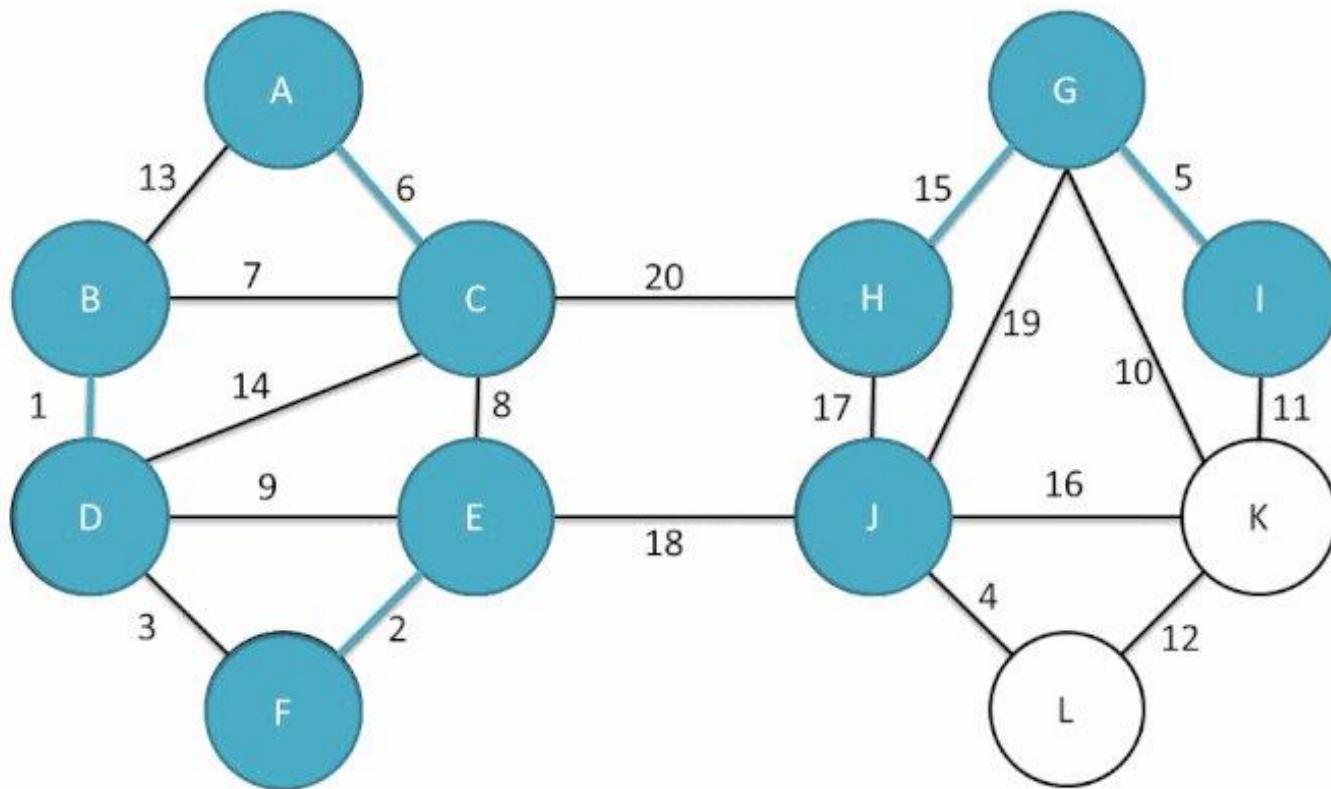


Boruvka's Algorithm



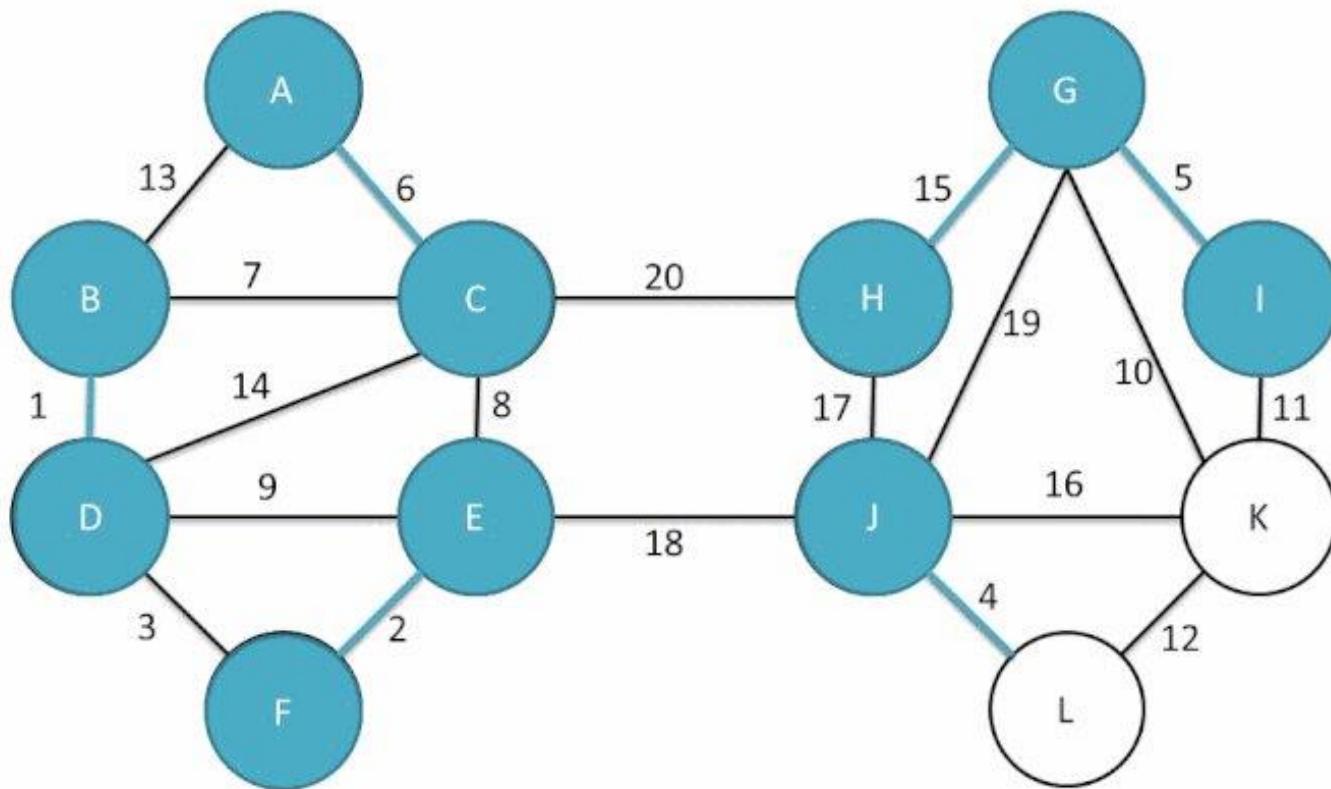


Boruvka's Algorithm



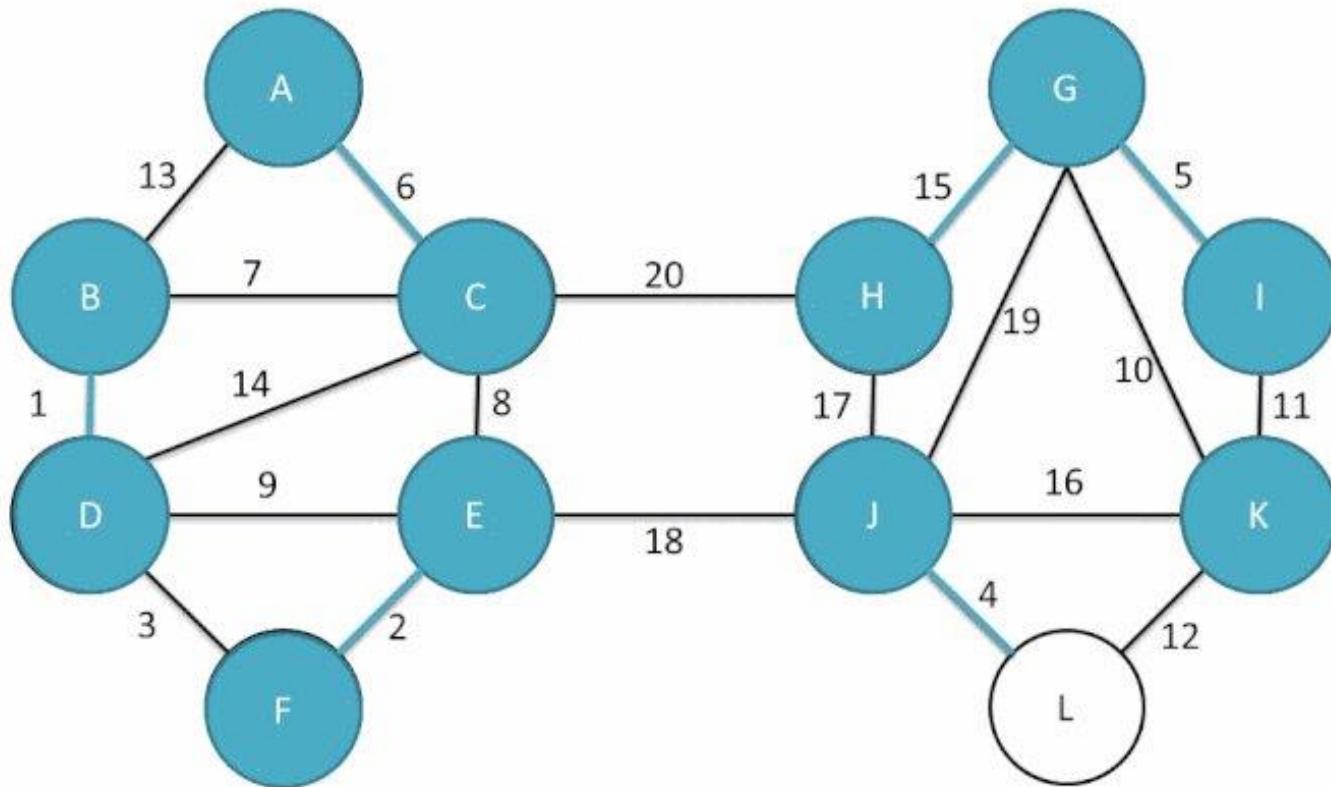


Boruvka's Algorithm



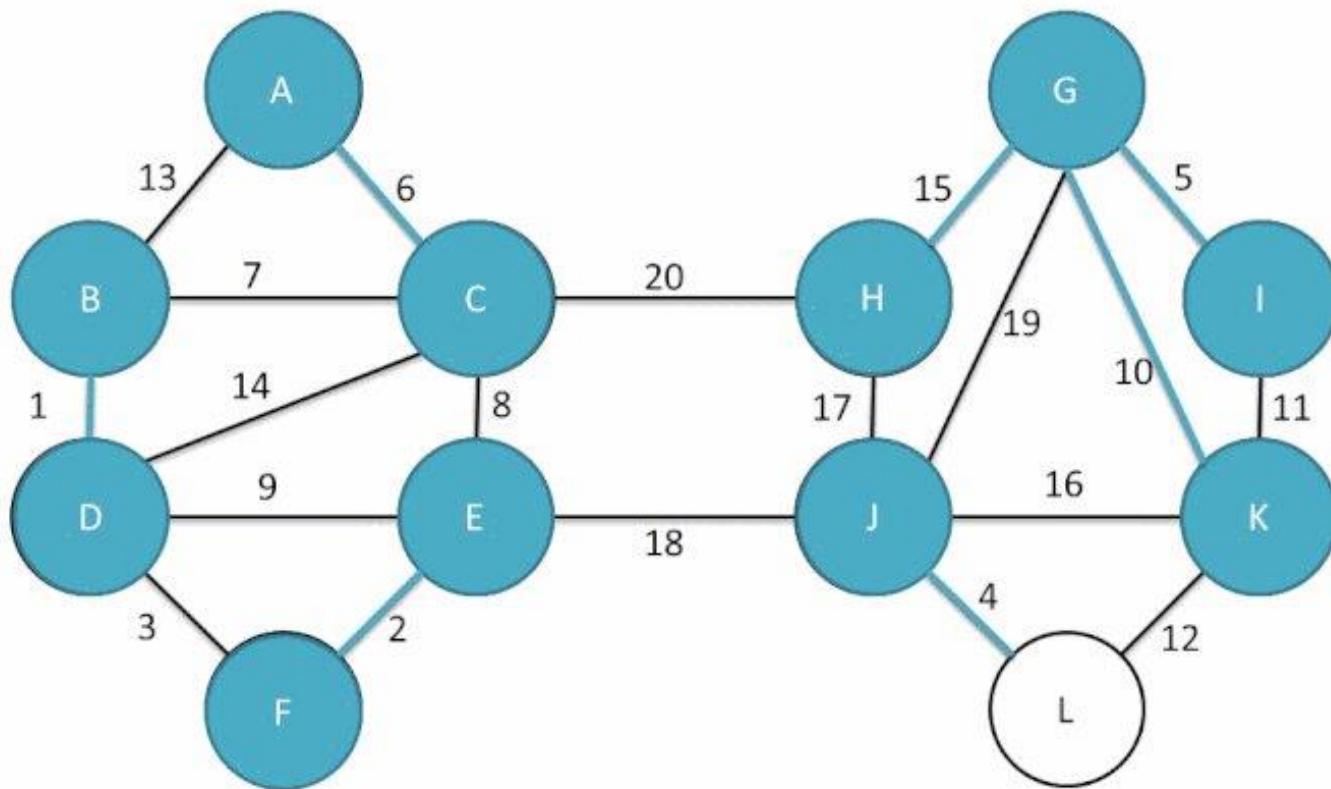


Boruvka's Algorithm



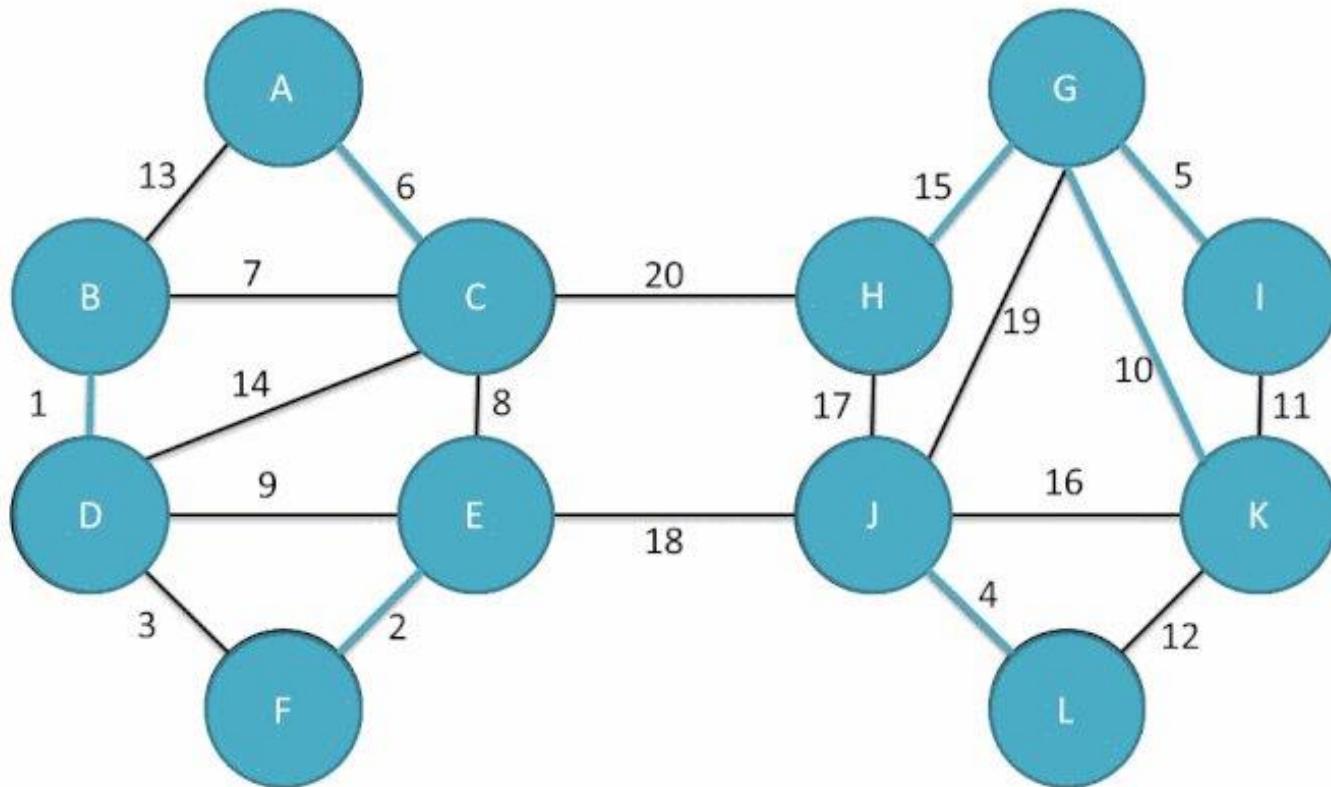


Boruvka's Algorithm



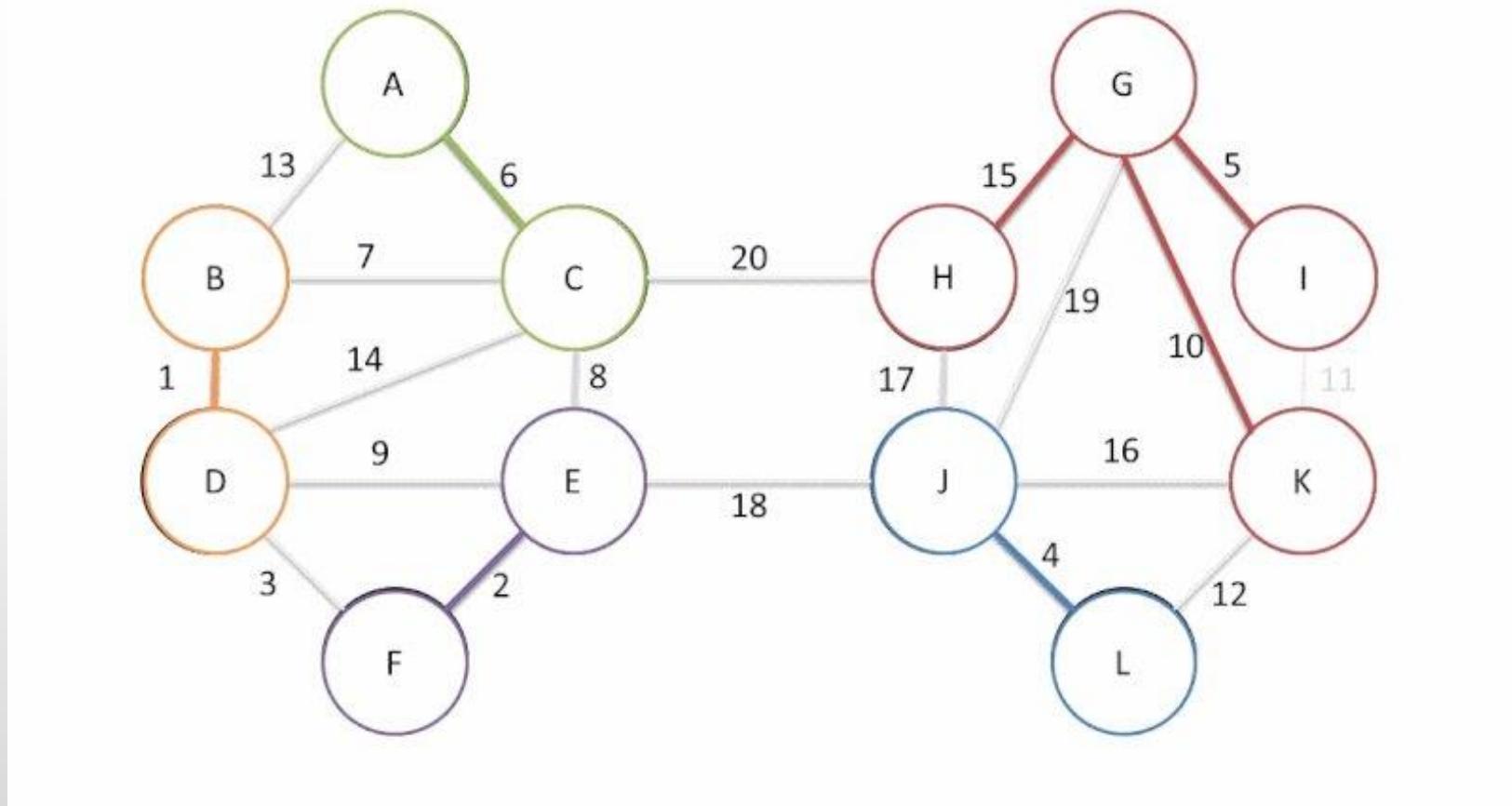


Boruvka's Algorithm



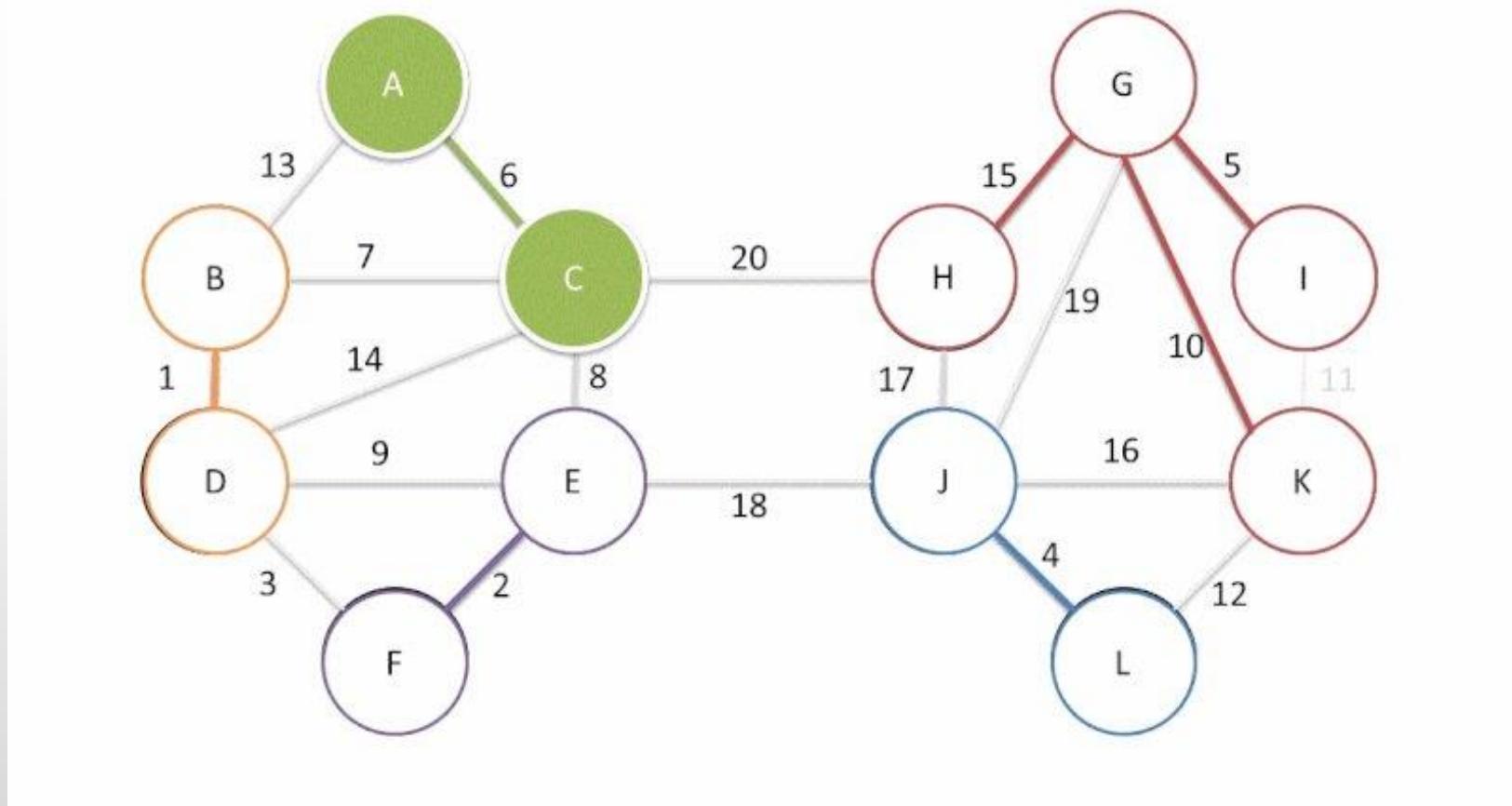


Boruvka's Algorithm



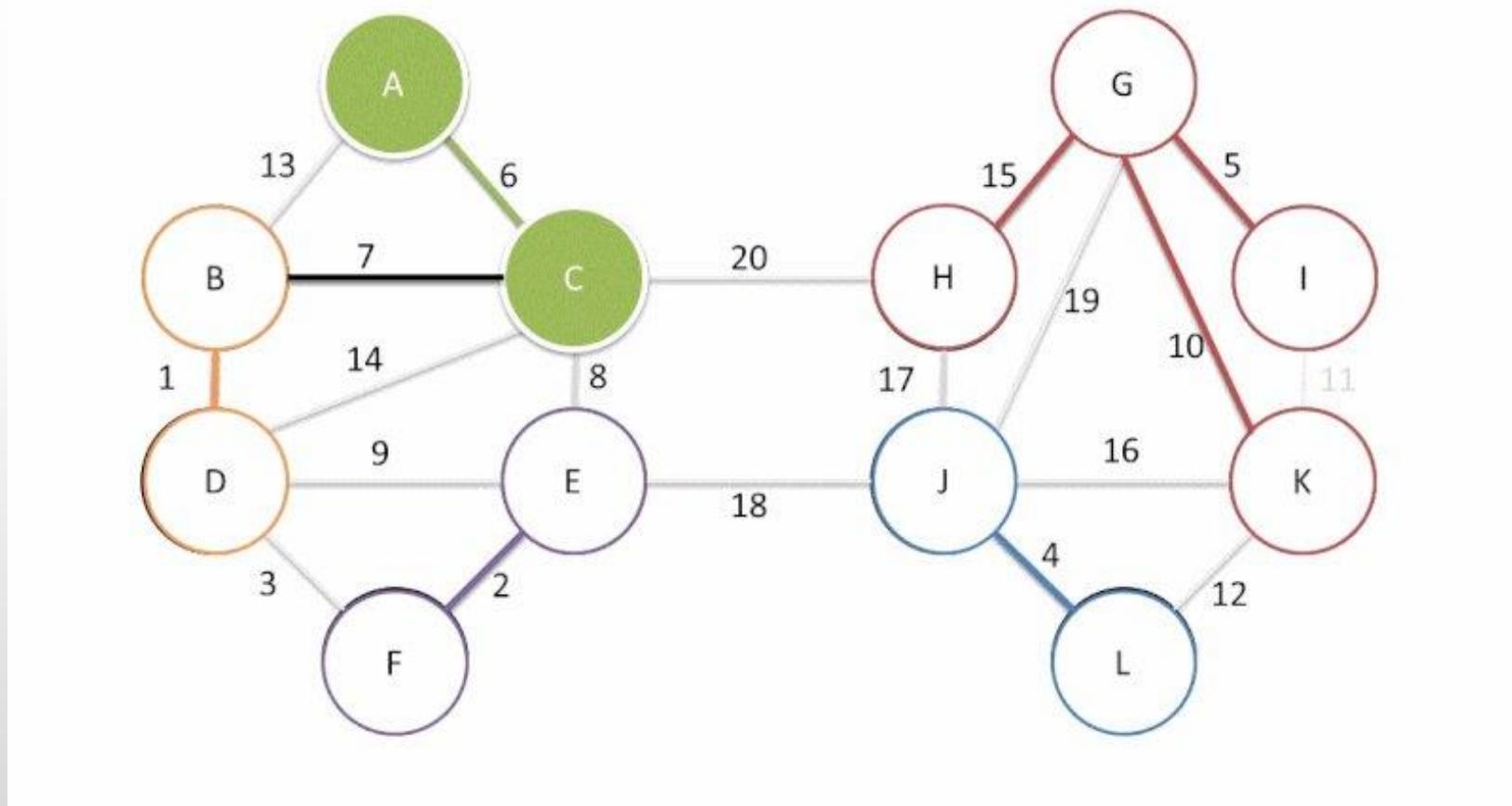


Boruvka's Algorithm



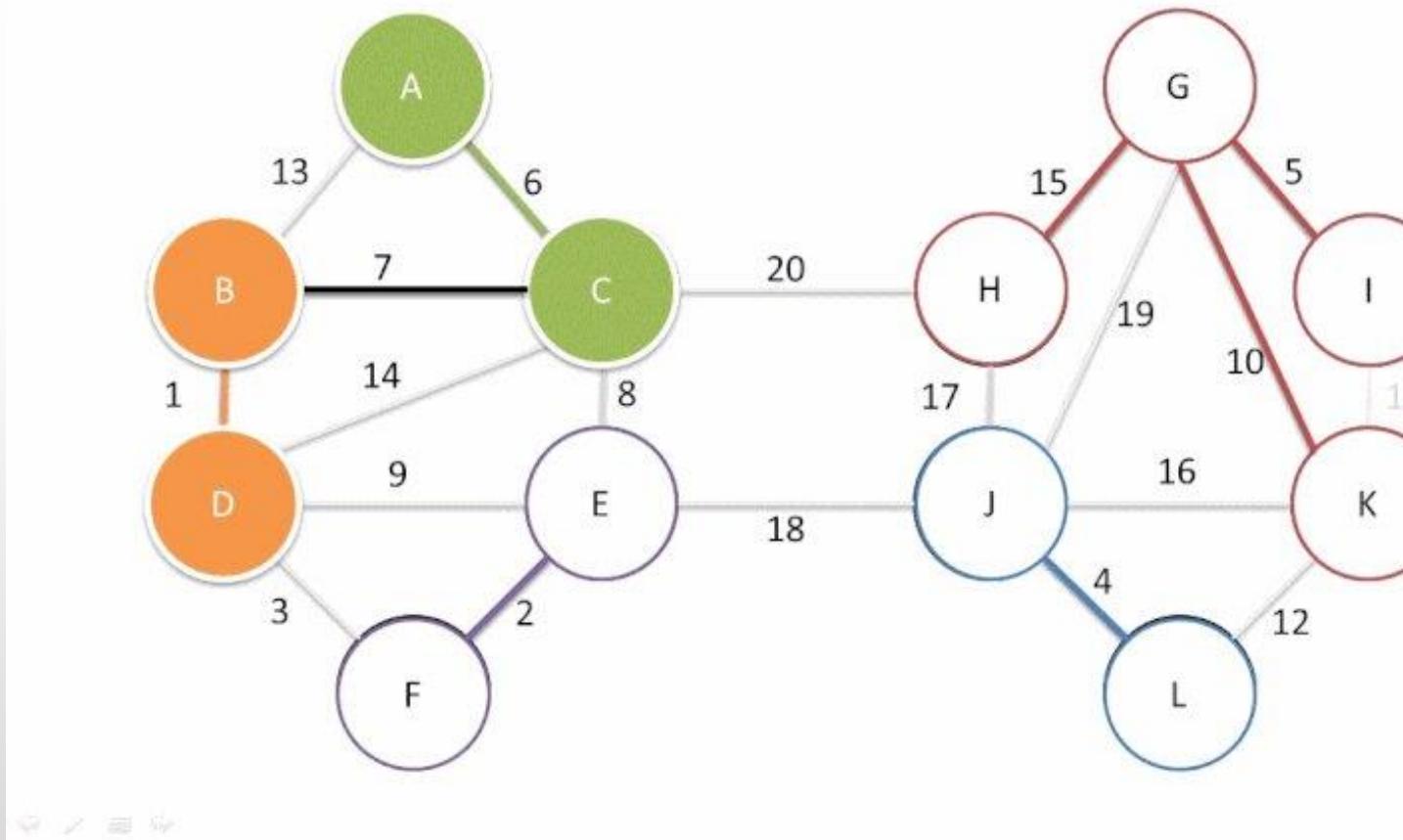


Boruvka's Algorithm



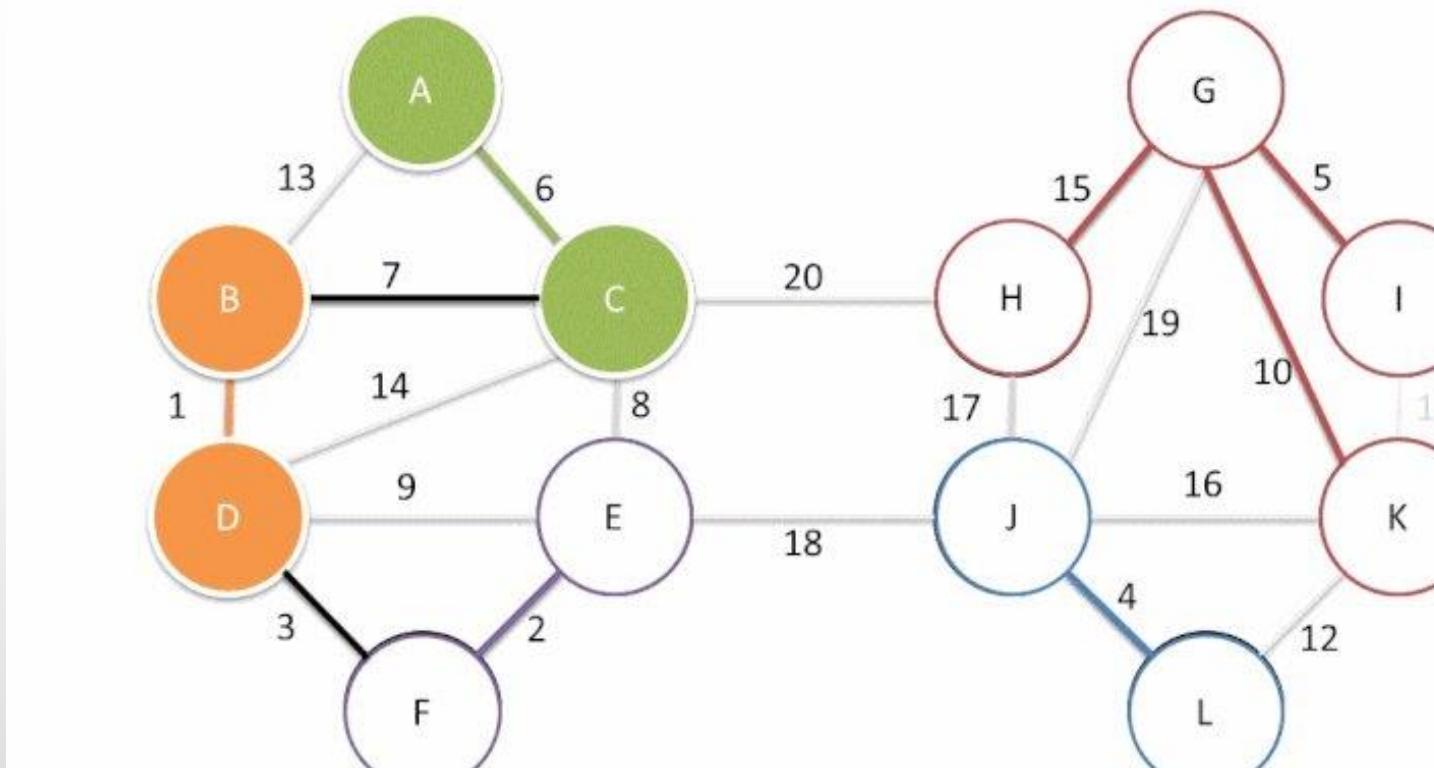


Boruvka's Algorithm



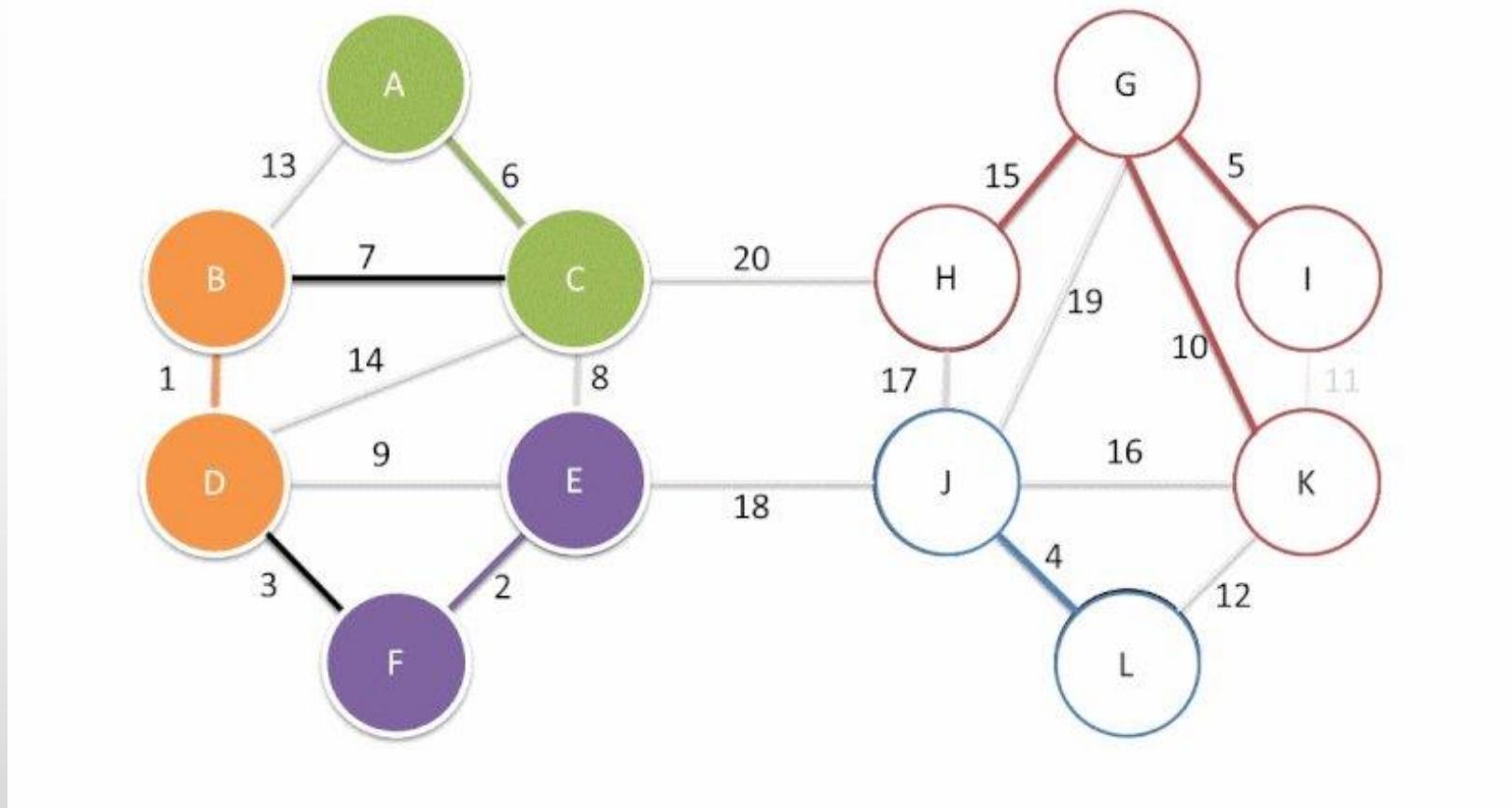


Boruvka's Algorithm



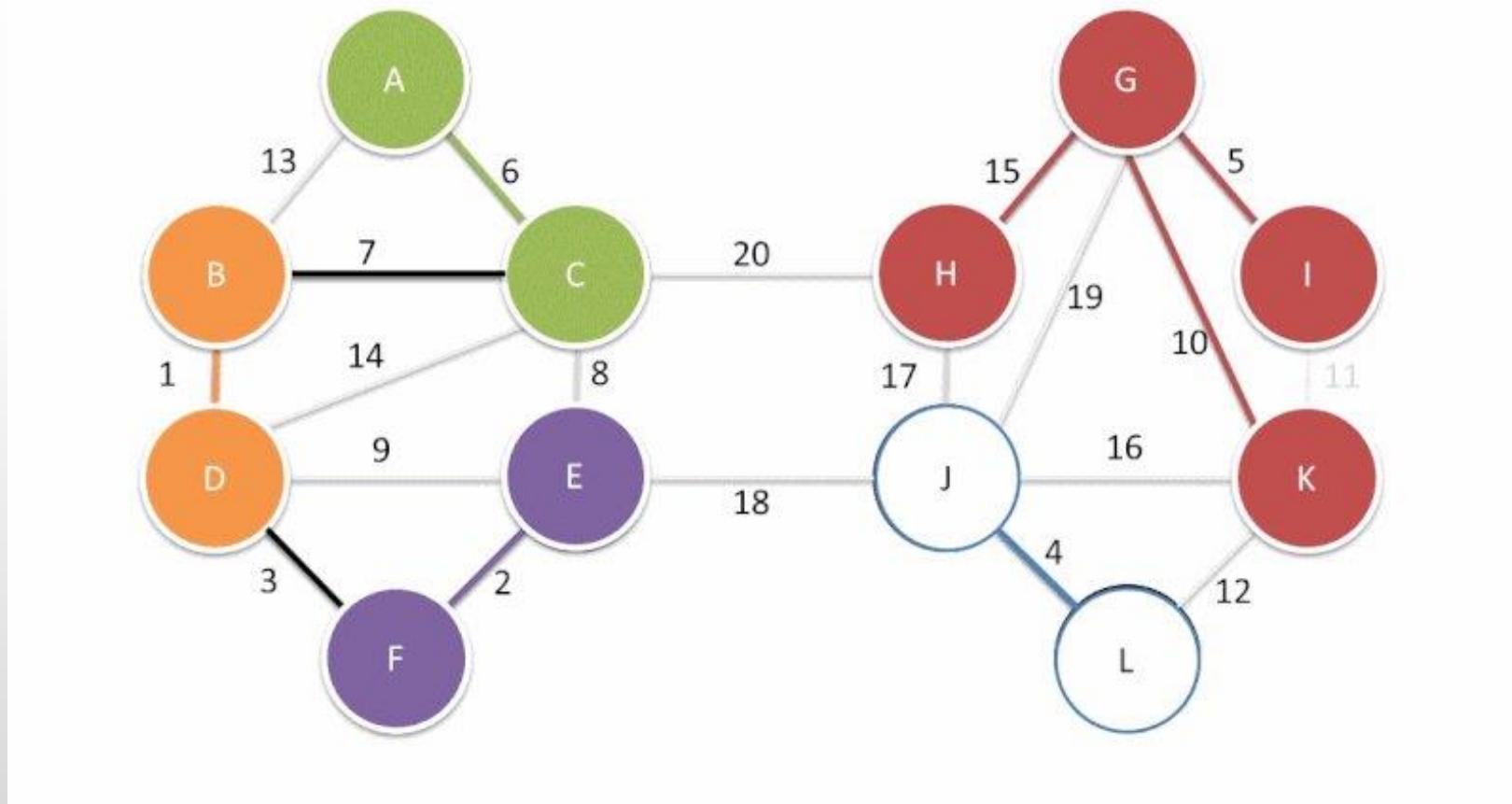


Boruvka's Algorithm



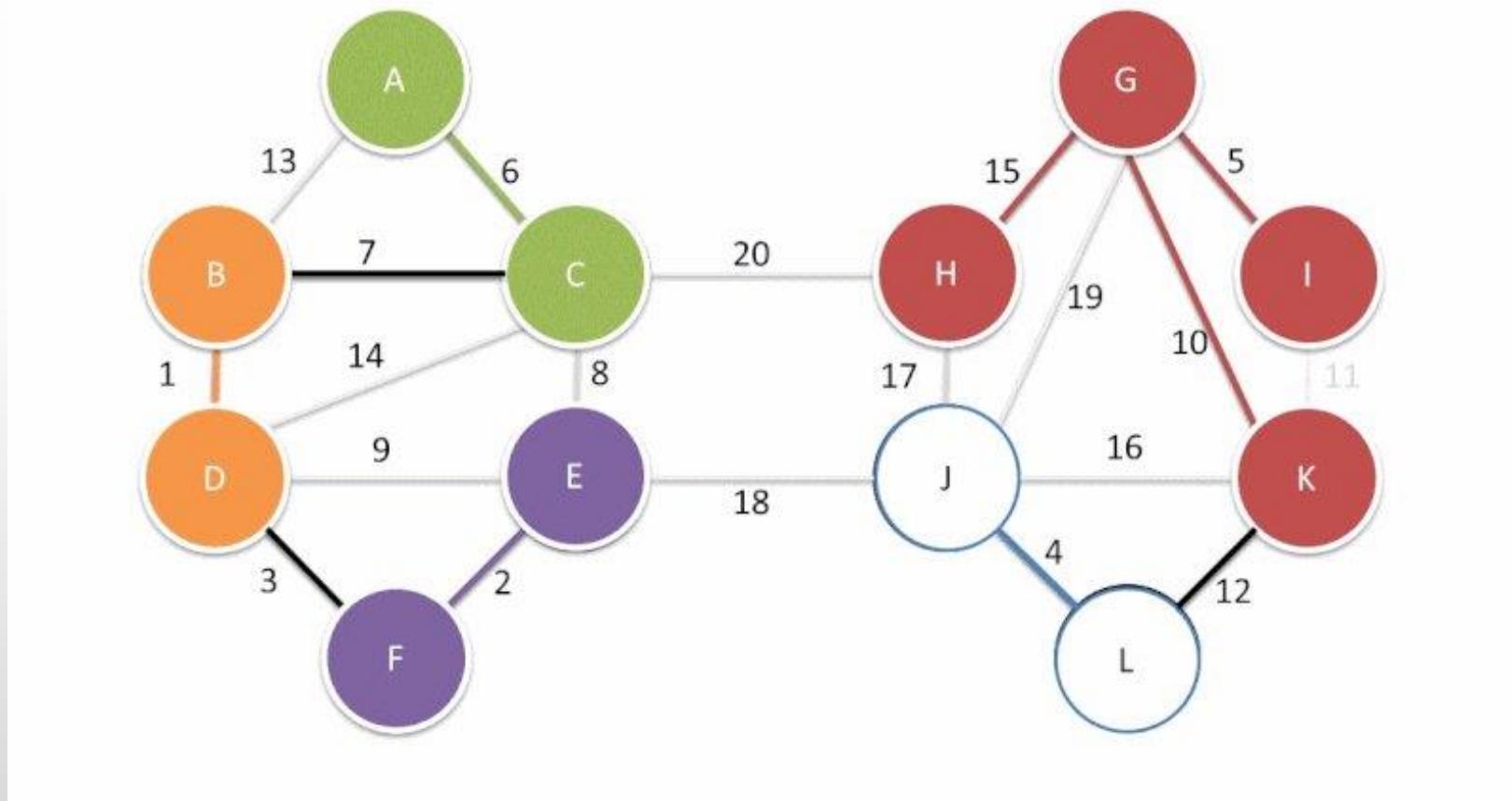


Boruvka's Algorithm



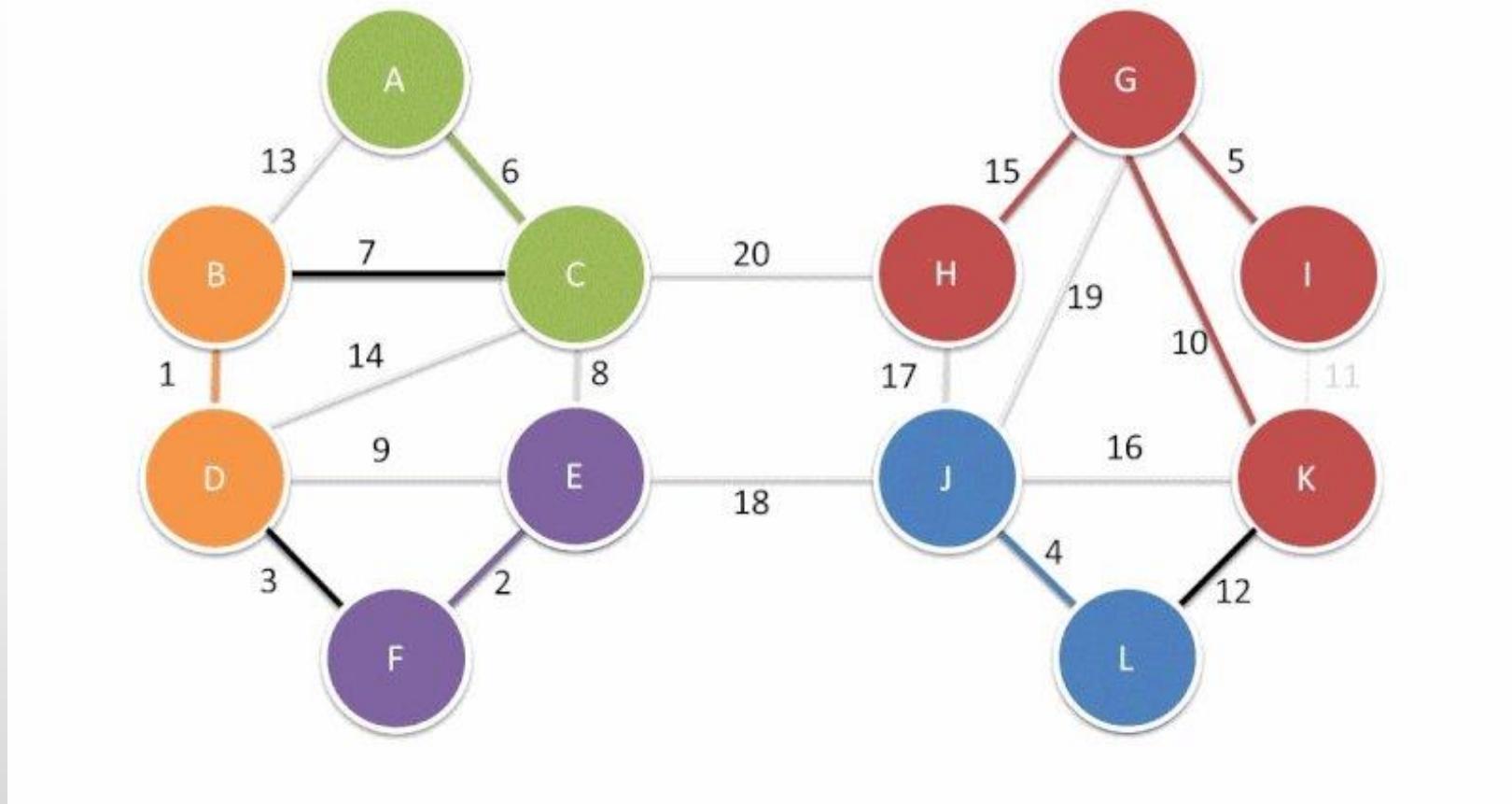


Boruvka's Algorithm



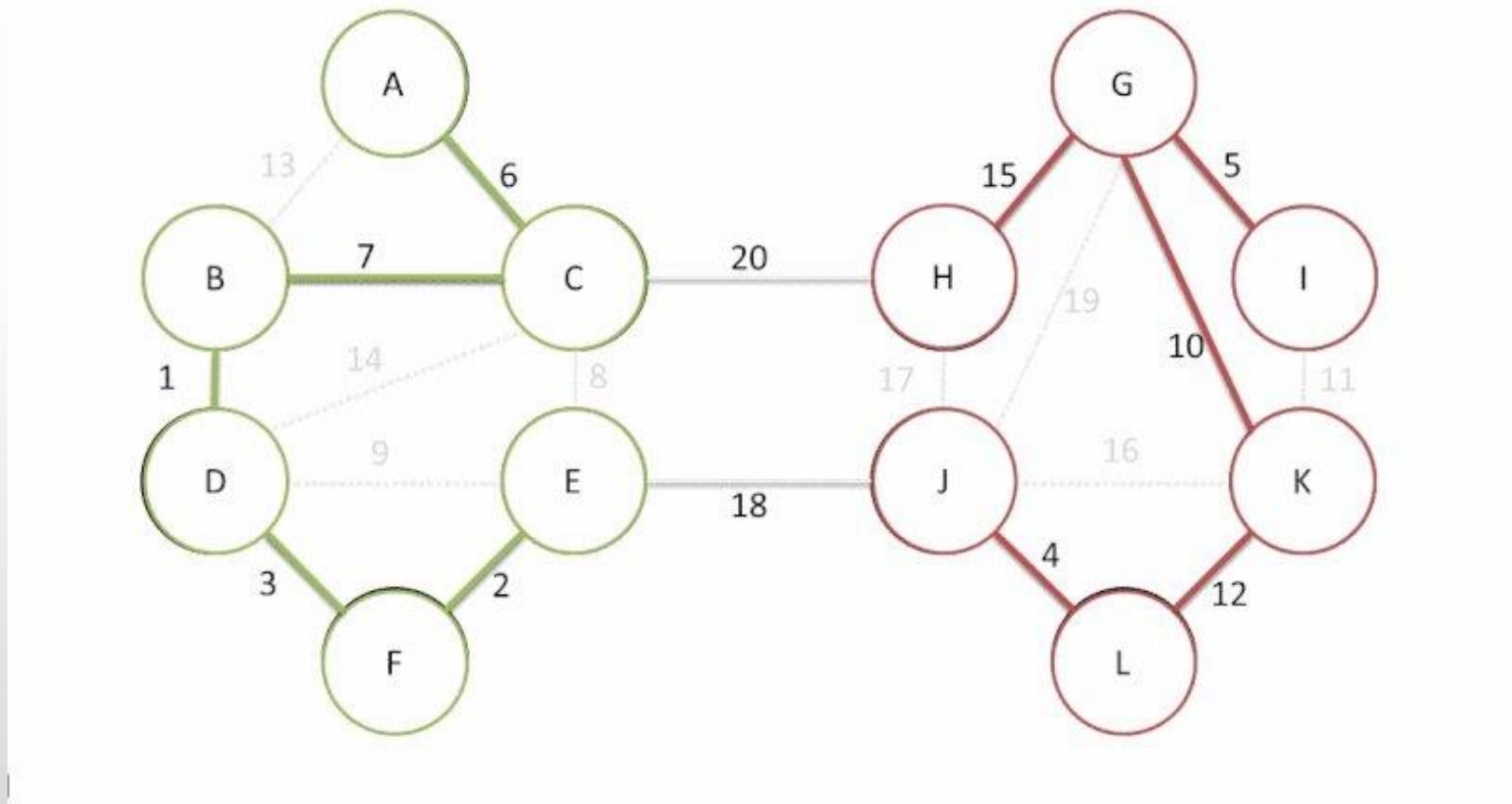


Boruvka's Algorithm



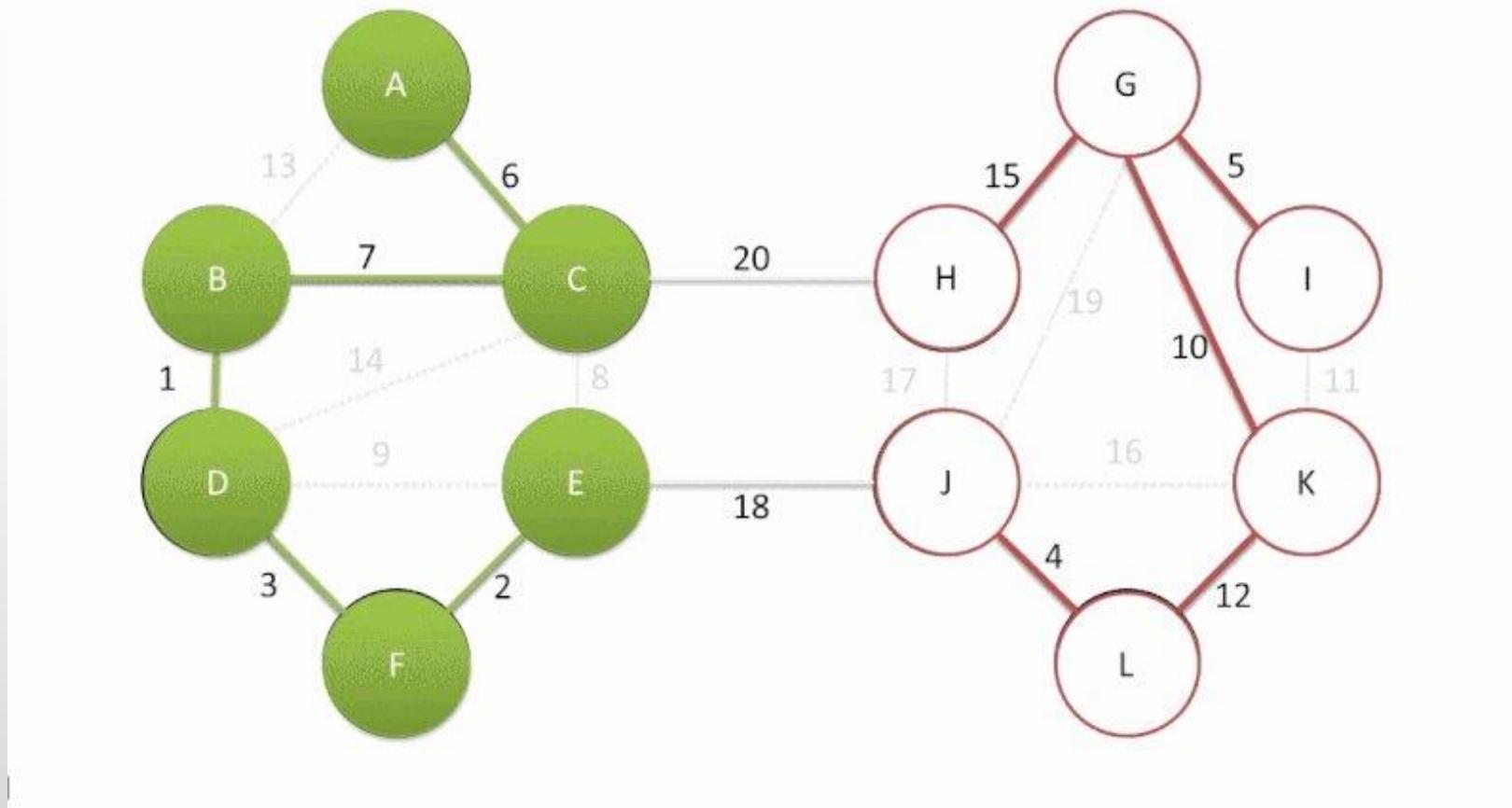


Boruvka's Algorithm



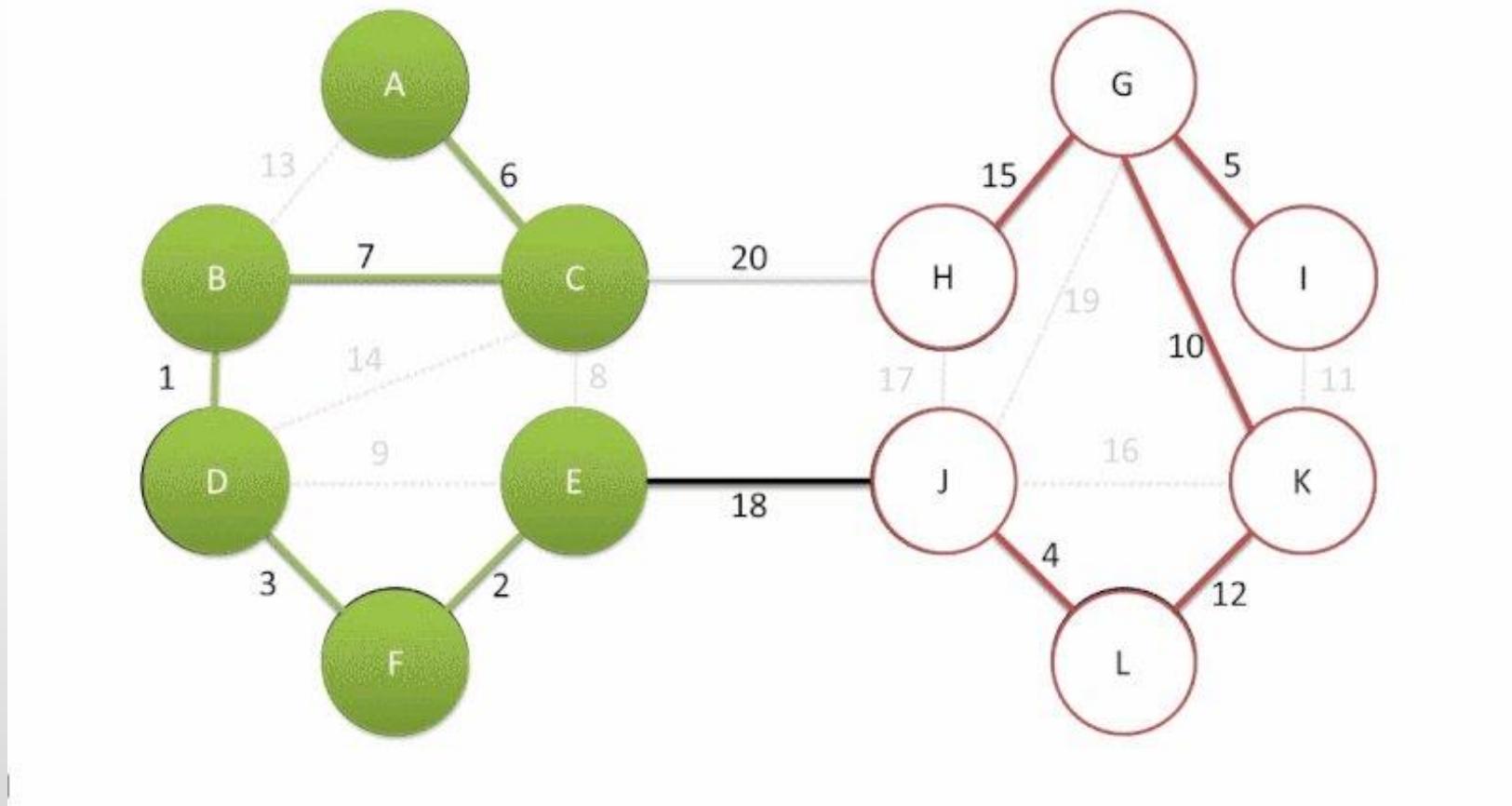


Boruvka's Algorithm



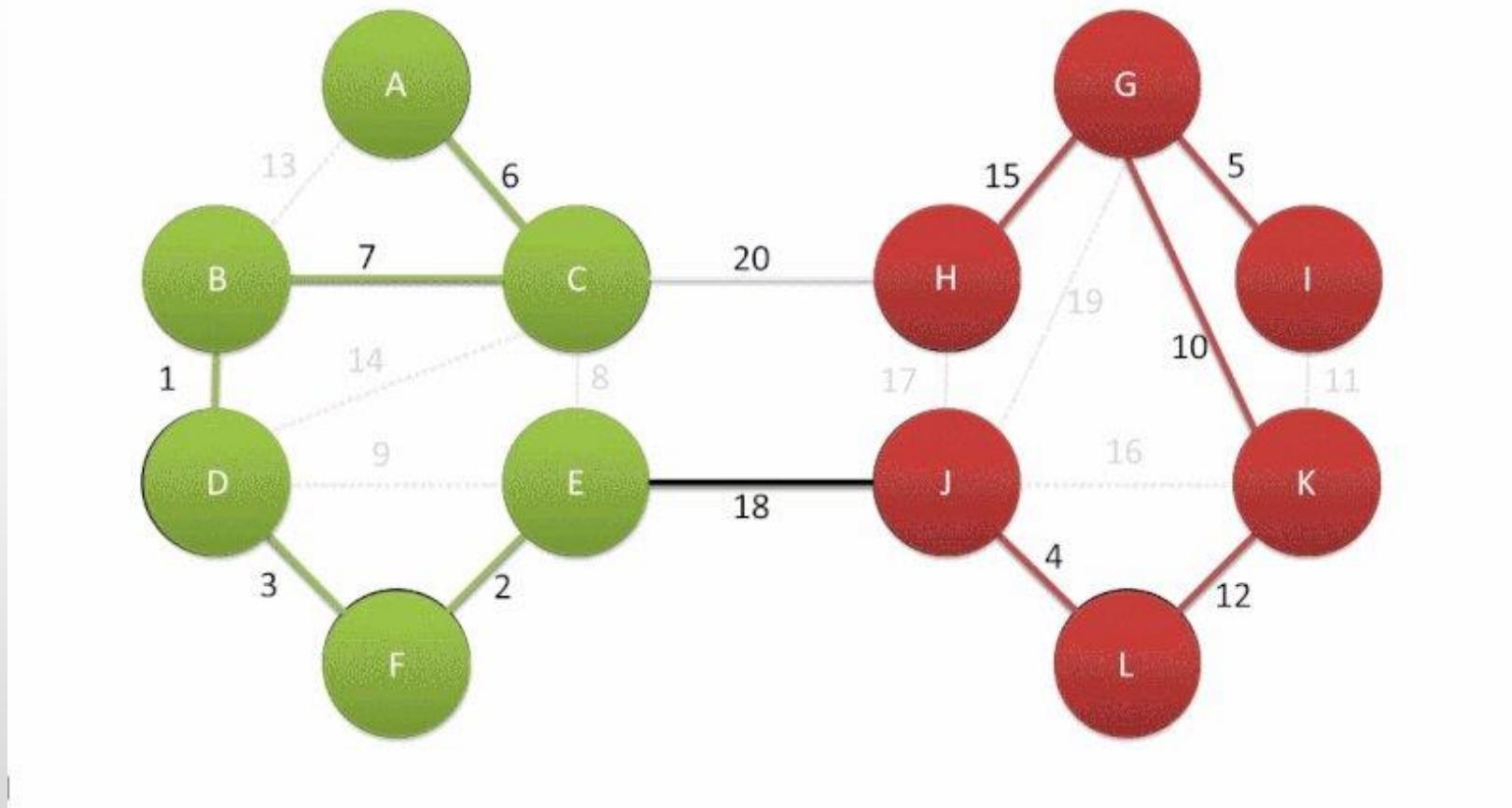


Boruvka's Algorithm



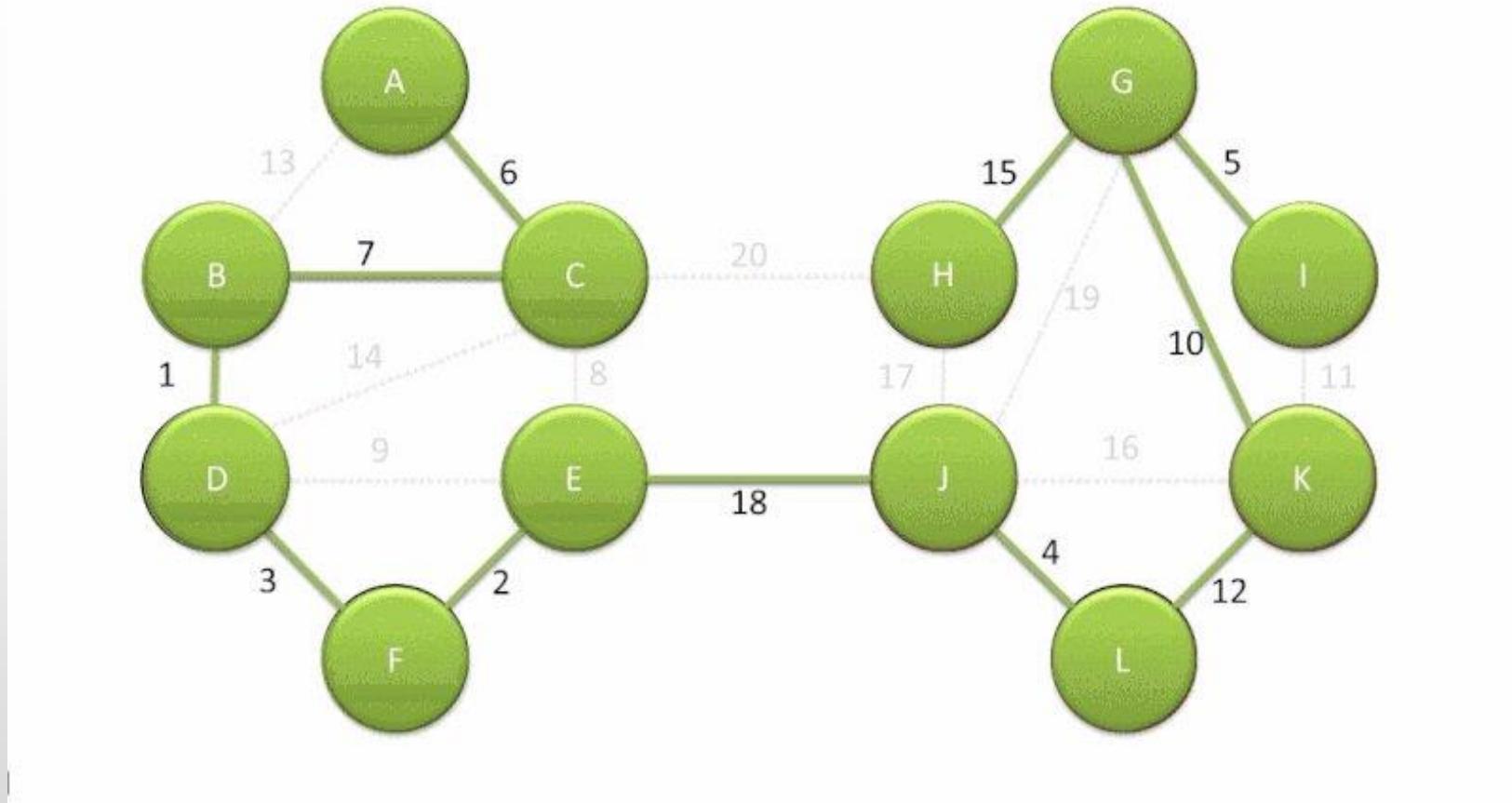


Boruvka's Algorithm





Boruvka's Algorithm







Reverse Delete Algorithm

- Tüm kenarları ağırlığına göre azalan sırayla sırala.
- En yüksek ağırlığa sahip kenarı sil.
- Eğer çizge hala bağlı değilse, silinen kenarı geri ekle.
- Tüm kenarlar silinene kadar 2. ve 3. adımları tekrarla.

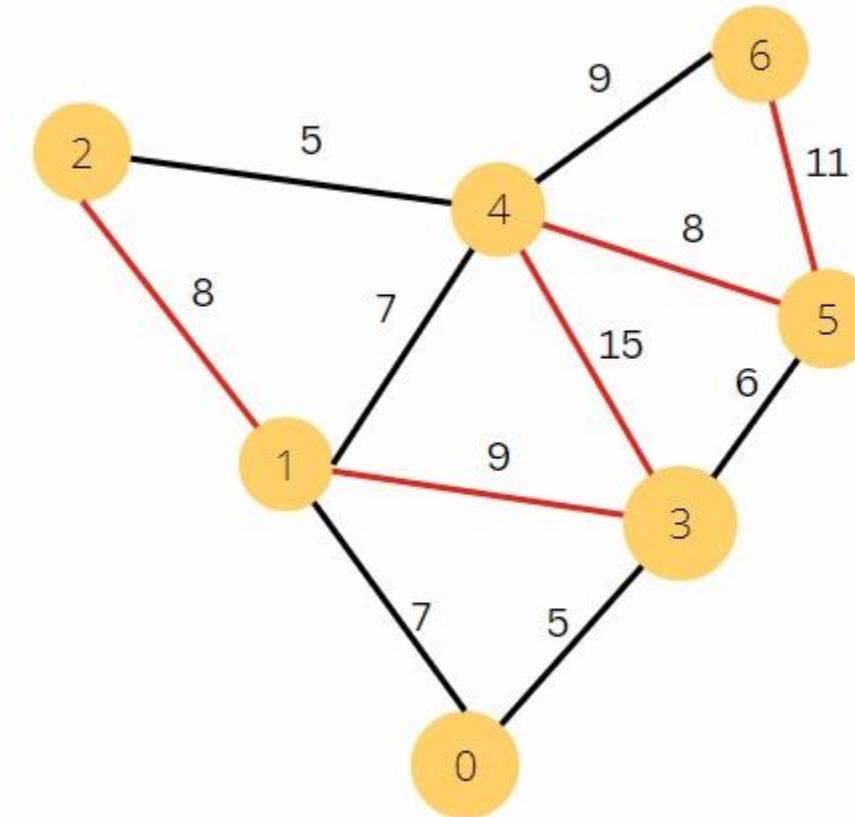


Ters Silme Algoritması Özellikleri

- Minimum kapsayan ağacı bulur.
- Greedy (Açgözlü) bir algoritmadır.
- Ağırlık sıralaması yaparak kenarları siler ve çizgenin bağlılığını korur.

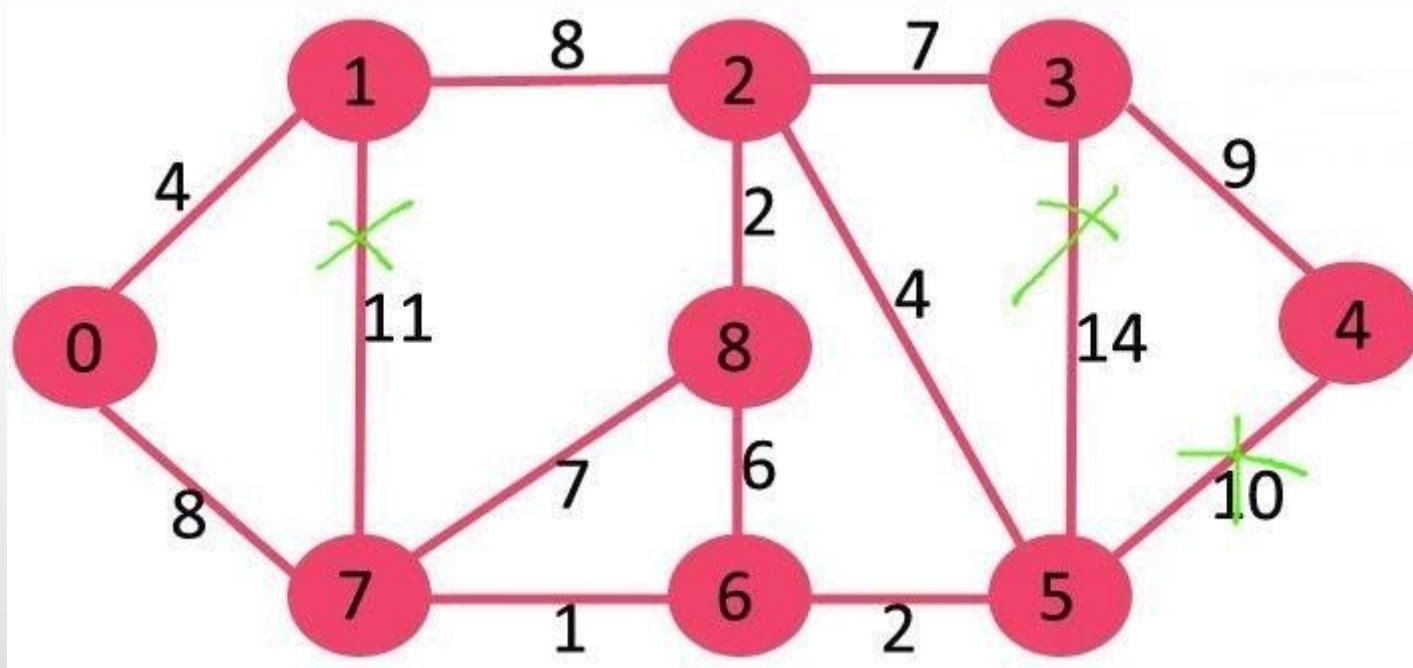


Ters Silme





Ters Silme





SON