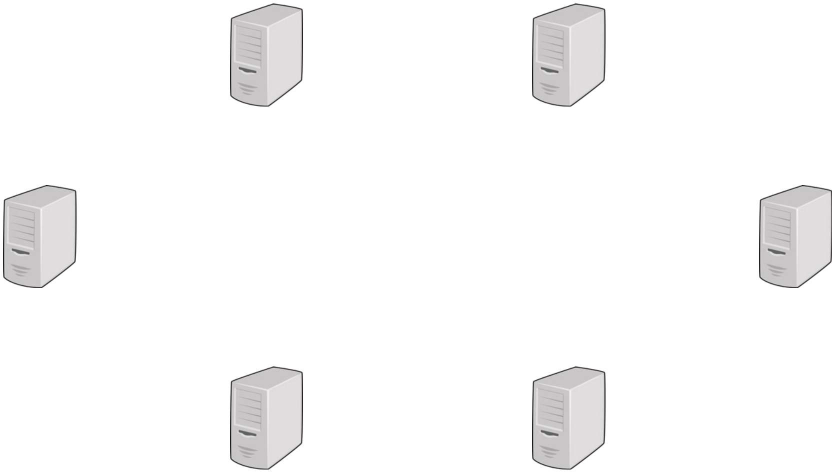
SPANNING TREES

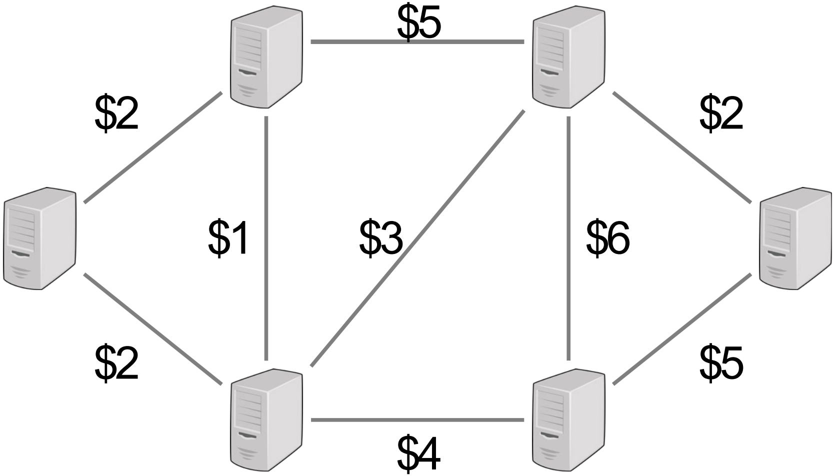
Alexander S. Kulikov
Steklov Institute of Mathematics at St. Petersburg Russian Academy of Sciences

Outline

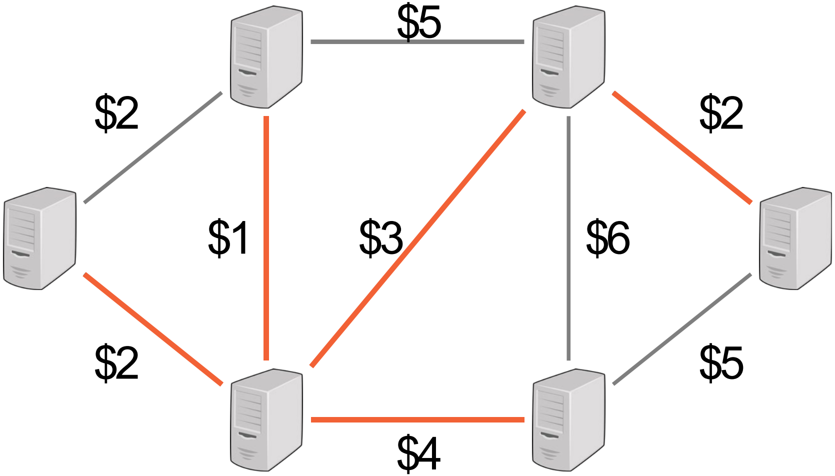
Connecting Computers by Wires



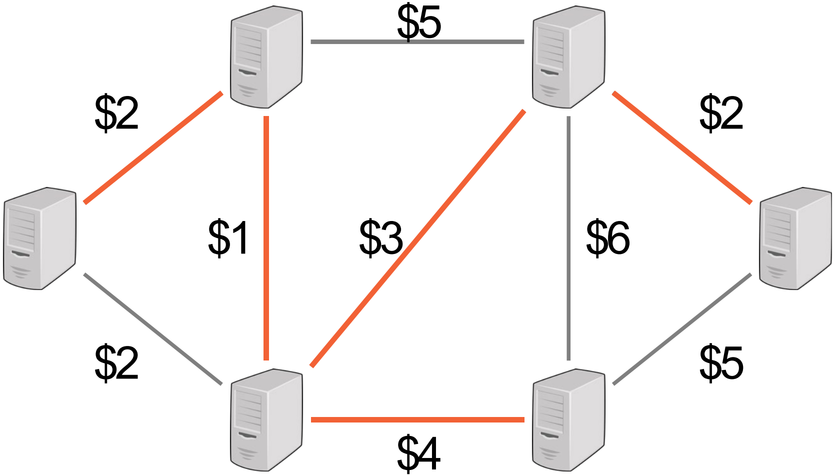
Connecting Computers by Wires



Connecting Computers by Wires

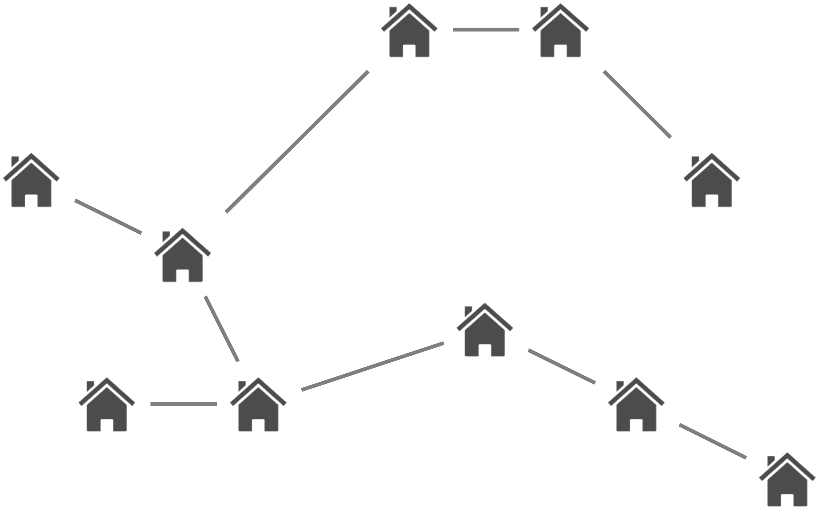


Connecting Computers by Wires



Building Roads

Building Roads



Minimum spanning tree (MST)

Input: A connected, undirected graph $G = (V, E)$ with positive edge weights.

Output: A subset of edges $E' \subseteq E$ of minimum total weight such that the graph (V, E') is connected.

Remark

The set E' always forms a tree.

Properties of Trees

- A tree is an undirected graph that is connected and acyclic.
- A tree on n vertices has $n - 1$ edges.
- Any connected undirected graph $G(V, E)$ with $|E| = |V| - 1$ is a tree.
- An undirected graph is a tree iff there is a unique path between any pair of its vertices.

Outline

This lesson

Two efficient greedy algorithms for the minimum spanning tree problem.

Kruskal’s algorithm

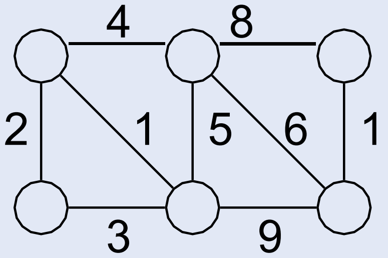
repeatedly add the next lightest edge if this doesn’t produce a cycle

Prim’s algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

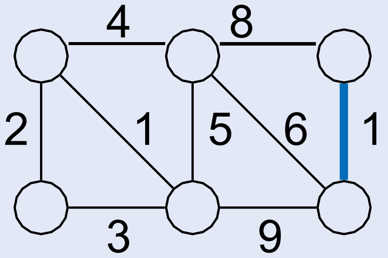


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal’s algorithm

repeatedly add the next lightest edge if this doesn’t produce a cycle

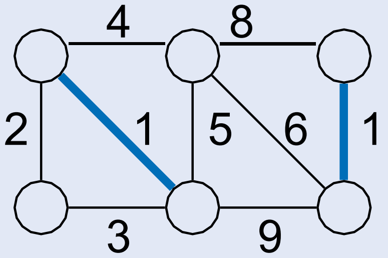


Prim’s algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

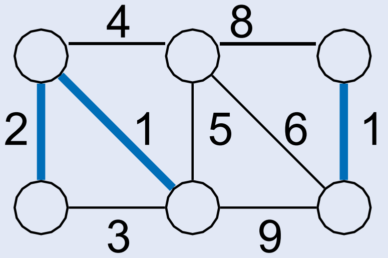


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

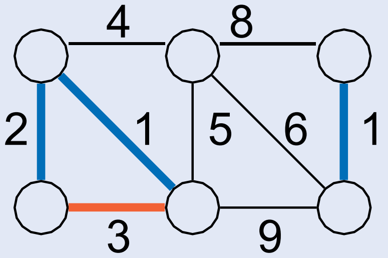


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

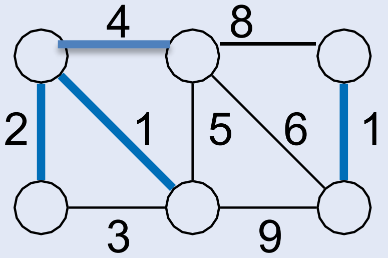


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal’s algorithm

repeatedly add the next lightest edge if this doesn’t produce a cycle

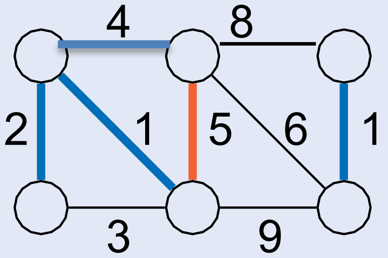


Prim’s algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

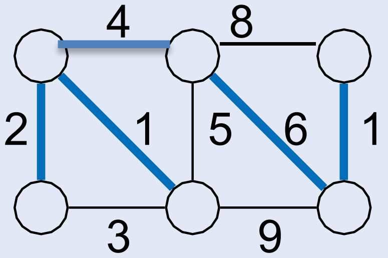


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle

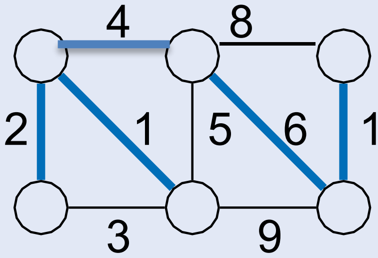


Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge

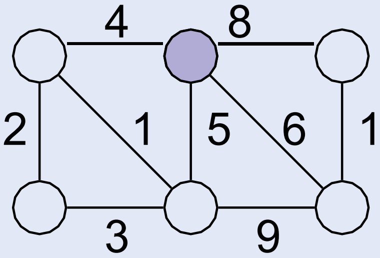
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



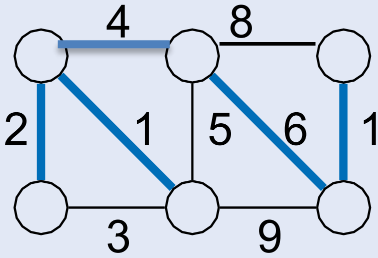
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



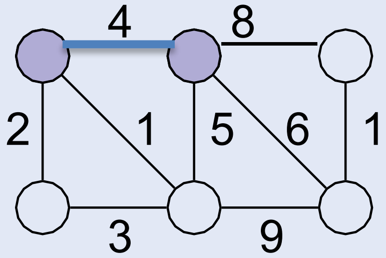
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



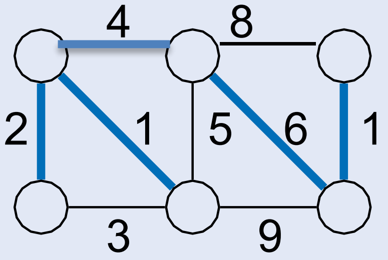
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



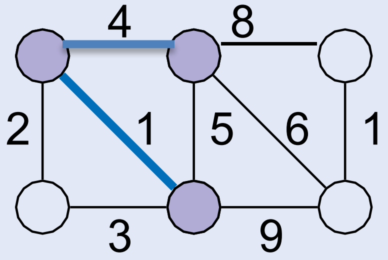
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



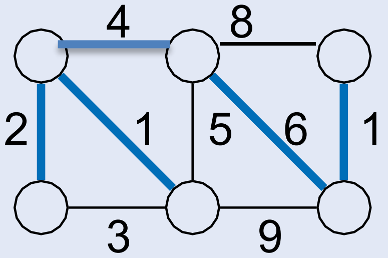
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



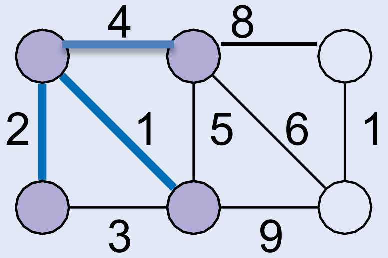
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



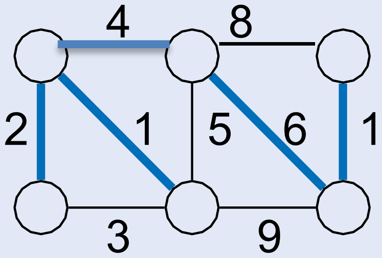
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



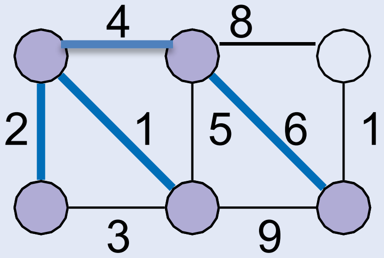
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



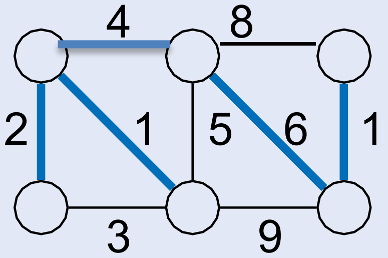
Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



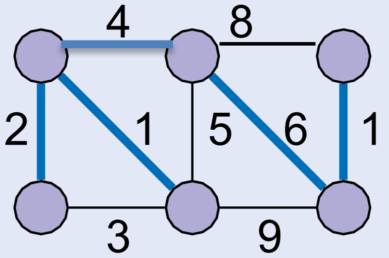
Kruskal's algorithm

repeatedly add the next lightest edge if this doesn't produce a cycle



Prim's algorithm

repeatedly attach a new vertex to the current tree by a lightest edge



Outline

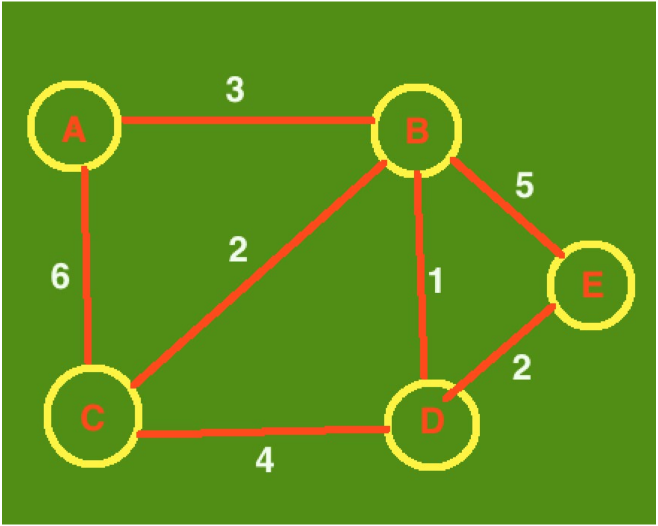
Kruskal(G)

```
for all u ∈ V:
  MakeSet(v)
X ← empty set
sort the edges E by weight
for all {u, v} ∈ E in non-decreasing
  weight order:
  if Find(u) ≠ Find(v):
    add {u, v} to X
    Union(u, v)
return X
```

Outline

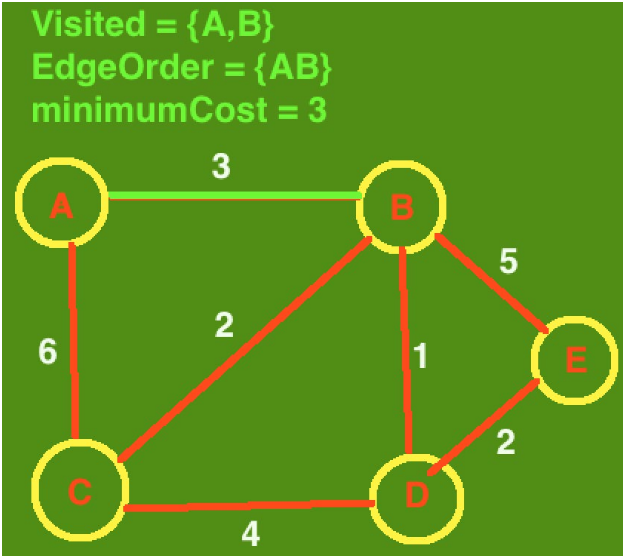
Örnek

Başlangıç düğümü seçilir (A). A’dan gidilebilecek kenarlara bakılır. {AB, AC}



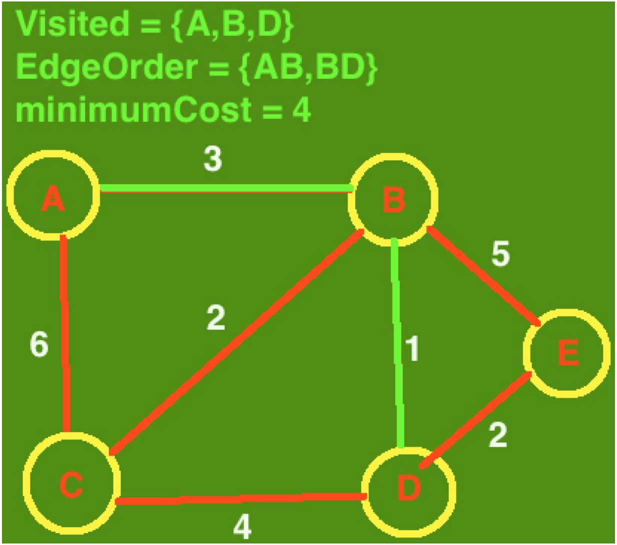
Örnek

{AB, AC} kenarlarından en küçük olan seçilir: AB. B düğümü *visited* olarak işaretlendi.



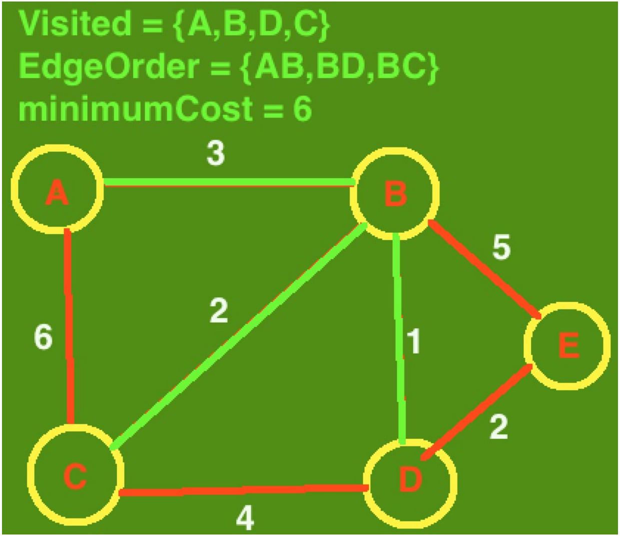
Örnek

Gidilebilecek {BD, BC, BE, AC} kenarlarından en küçük olan seçilir: BD. D düğümü *visited* olarak işaretlendi.



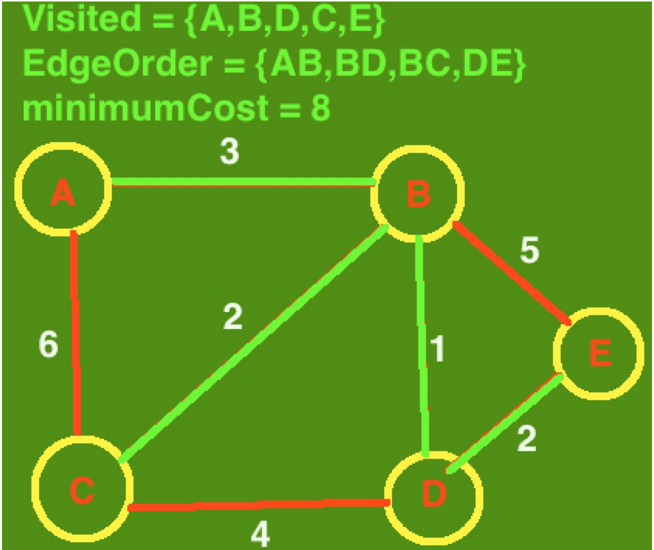
Örnek

Gidilebilecek {BC=DE, DC, BE, AC} kenarlarından en küçük olan seçilir: BC. C düğümü *visited* olarak işaretlendi.



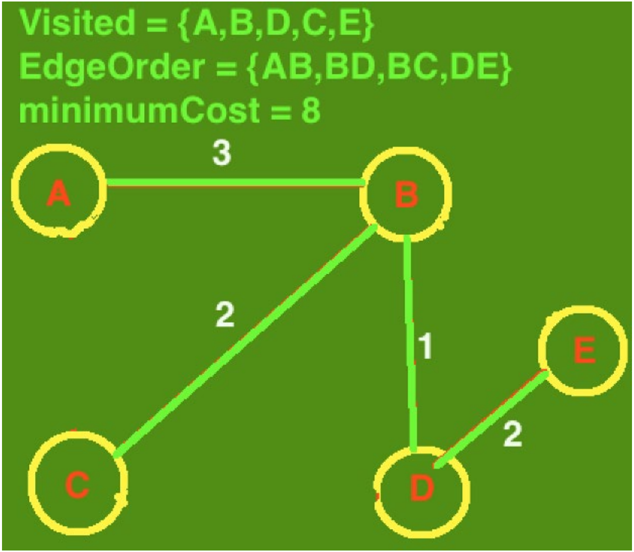
Örnek

Gidilebilecek {DE, DC, BE, AC} kenarlarından en küçük olan seçilir: DE. E düğümü *visited* olarak işaretlendi. Tüm düğümlere ulaşıldı.

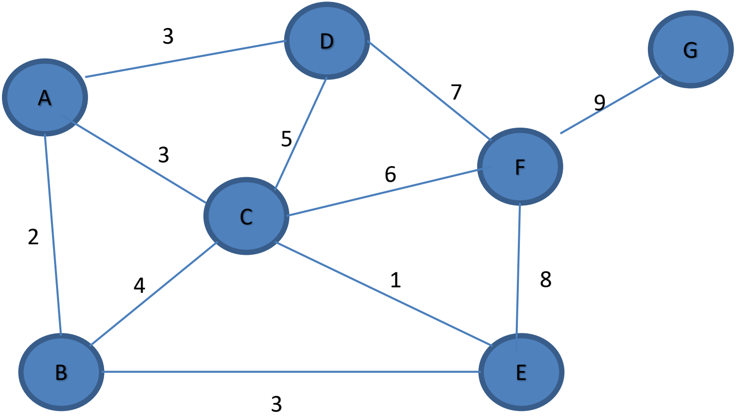


Örnek

Elde edilen MST:

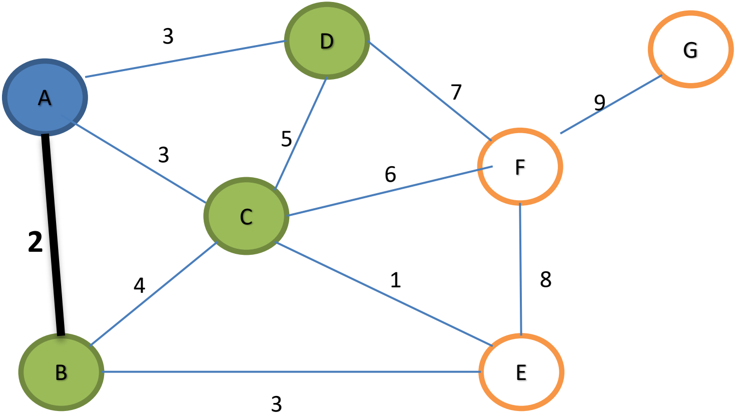


Örnek

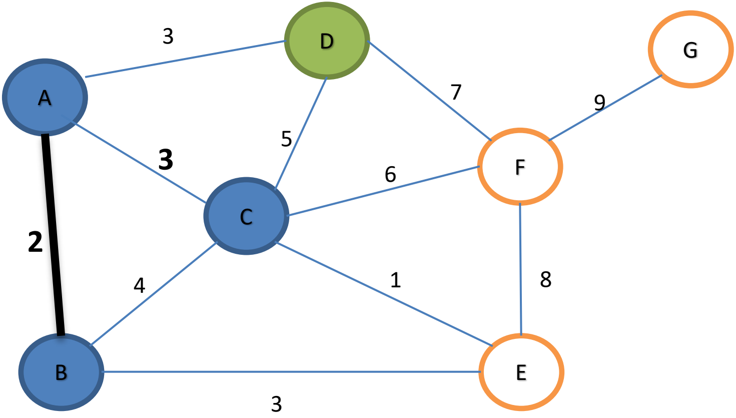


Örnek:

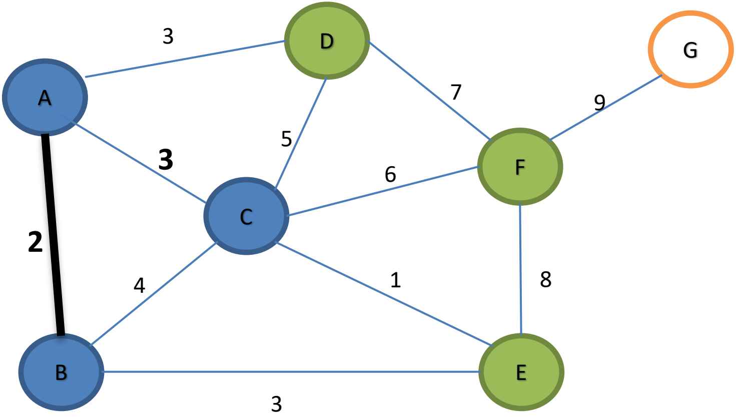
Bir başlangıç noktası seçilir (A). A düğümünden gidilebilen düğümlere bakılır ve en kısası seçilir.



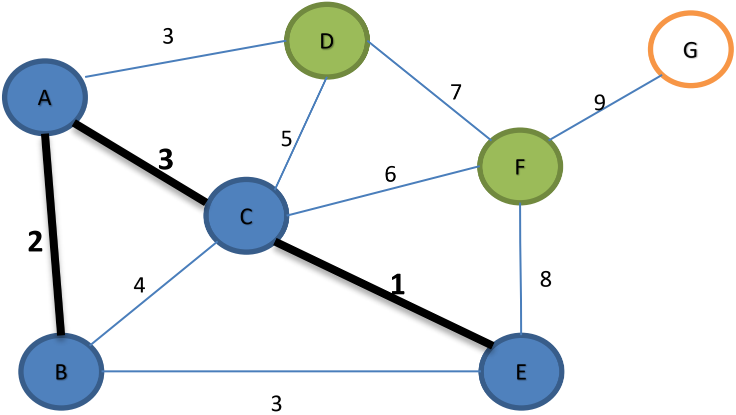
Güncellenen düğümler arasından en küçük ağırlıkla gidileni bul. (C veya E veya D). Bir tanesini seç. (C) .



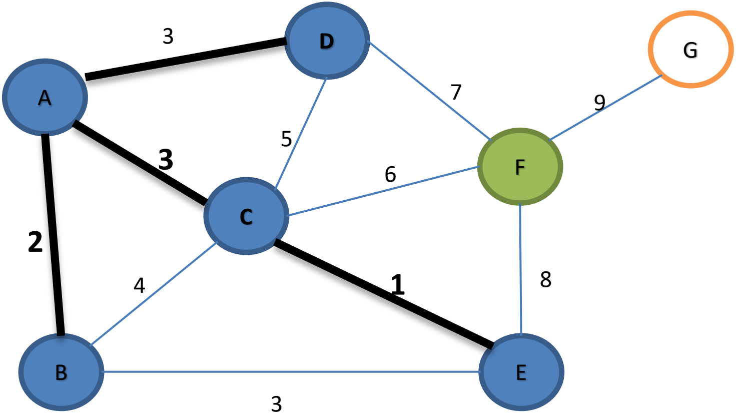
Güncellenen düğümler arasından en küçük ağırlıklı şekilde gidileni bul. (E)



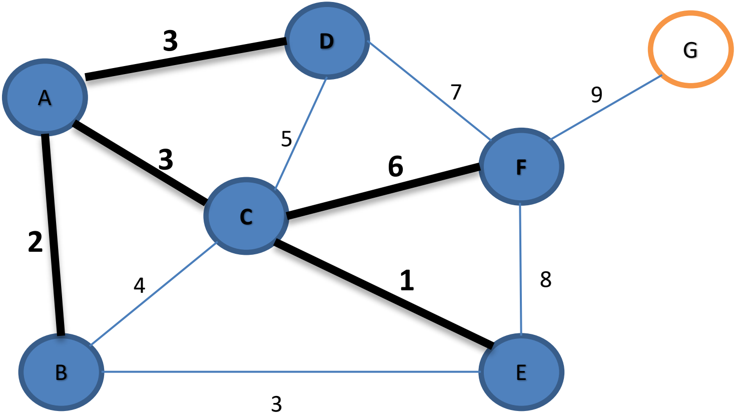
Güncellenen düğümler arasından en küçük ağırlıklı şekilde gidilebilecekleri bul. (D ve F). En küçük ağırlıkla gidileceği bul. (D)



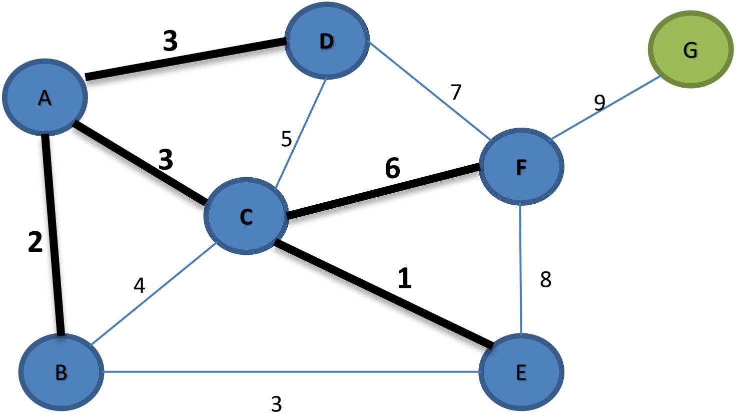
Güncellenen düğümler arasından en küçük ağırlıklı şekilde gidilebilecekleri bul. (D ve F). En küçük ağırlıkla gidileceği bul. (D)



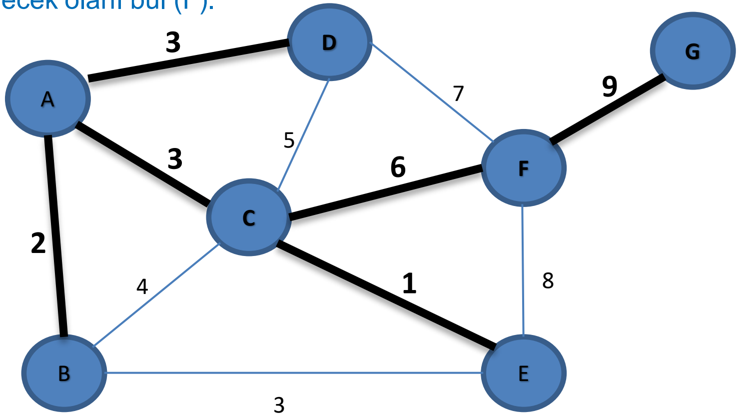
Daha önce gidilmeyen düğümler arasından, en küçük ağırlıkla gidilebilecek olanı bul (F).



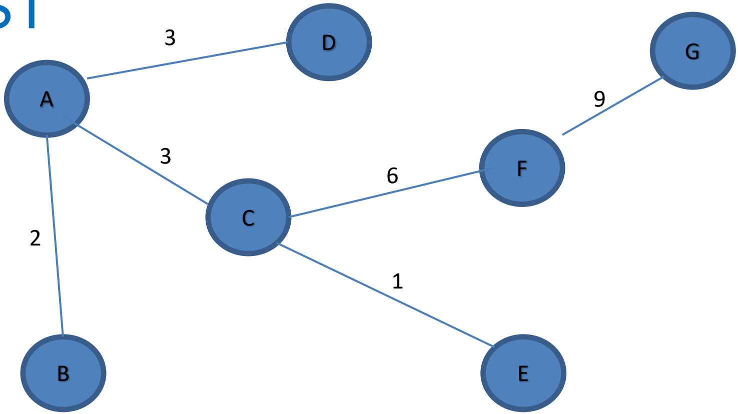
Daha önce gidilmeyen düğümler arasından, en küçük ağırlıkla gidilebilecek olanı bul (F).



Daha önce gidilmeyen düğümler arasından, en küçük ağırlıkla gidilebilecek olanı bul (F).



MST



Visited = {A, B, C, E, D, F, G},
Total Edge Weight: 2 + 3 + 3 + 1+ 6+ 9 = 24

Prim's Algorithm

Prim(G)

```
for all u ∈ V:
  cost[u] ← ∞, parent[u] ← nil
pick any initial vertex u0
cost[u0] ← 0
PrioQ ← MakeQueue(V)     {priority is cost}
while PrioQ is not empty:
  v ← ExtractMin(PrioQ)
  for all {v, z} ∈ E:
    if z ∈ PrioQ and cost[z] > w(v, z):
      cost[z] ← w(v, z), parent[z] ← v
      ChangePriority(PrioQ, z, cost[z])
```

Summary

Kruskal: repeatedly add the next lightest edge if this doesn't produce a cycle; use disjoint sets to check whether the current edge joins two vertices from different components

Prim: repeatedly attach a new vertex to the current tree by a lightest edge; use priority queue to quickly find the next lightest edge