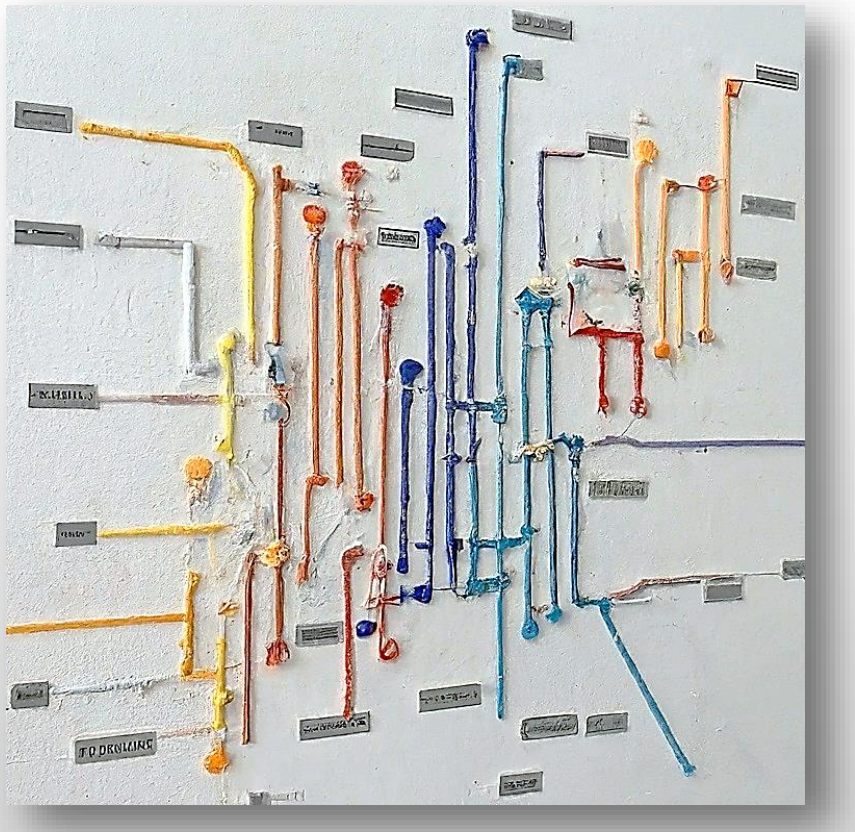
# Bölüm 4: Çizge Algoritmaları

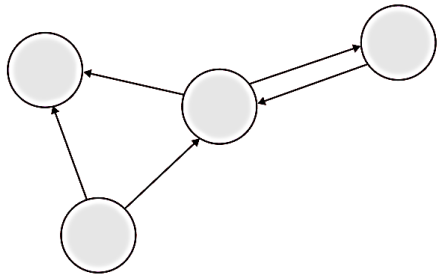
## Algoritmalar

# Çizge Algoritmaları

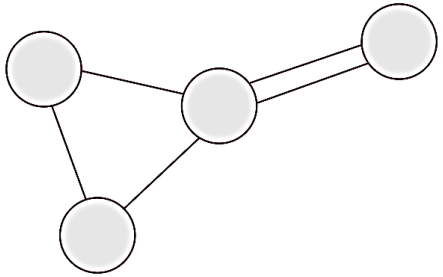
- Dünya aslında bir ağ gibidir.
  - Şehirler yollarla,
  - İnsanlar ilişkilerle,
  - Bilgisayarlar kablolarla birbirine bağlıdır.
- Çizge algoritmaları bu ağları inceler ve anlamlandırır.



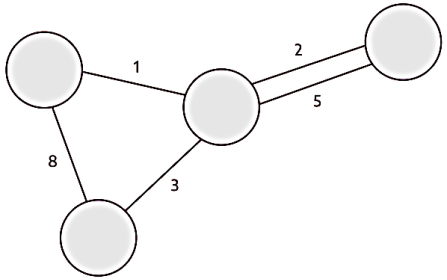
# Çizge Türleri



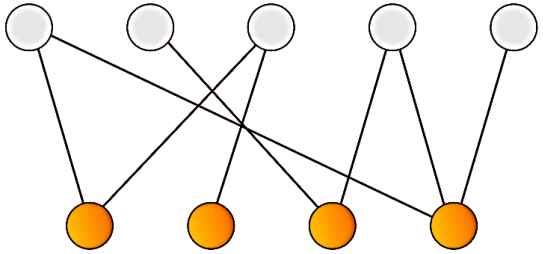
Directed graph



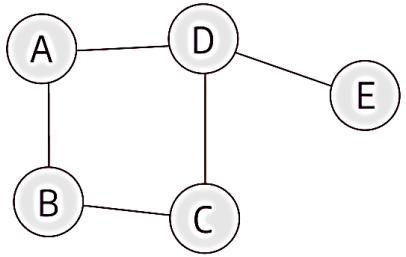
Undirected



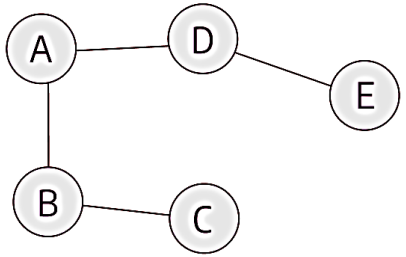

Weighted



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.

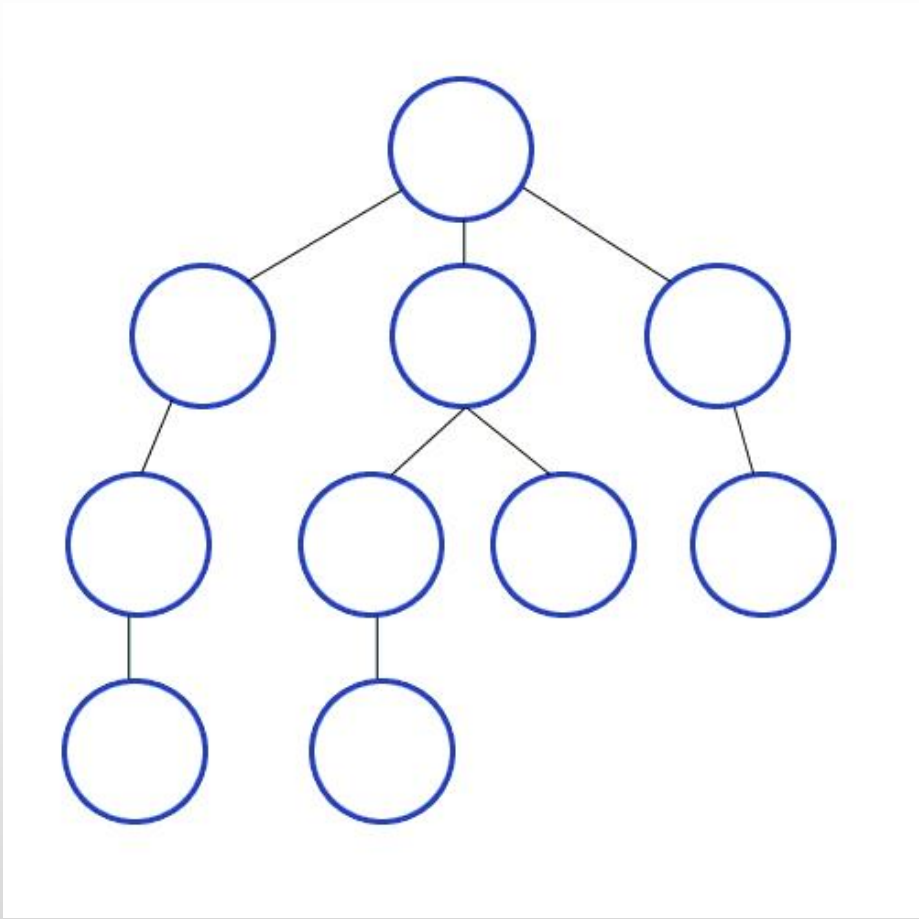
# Çizge Algoritmaları

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

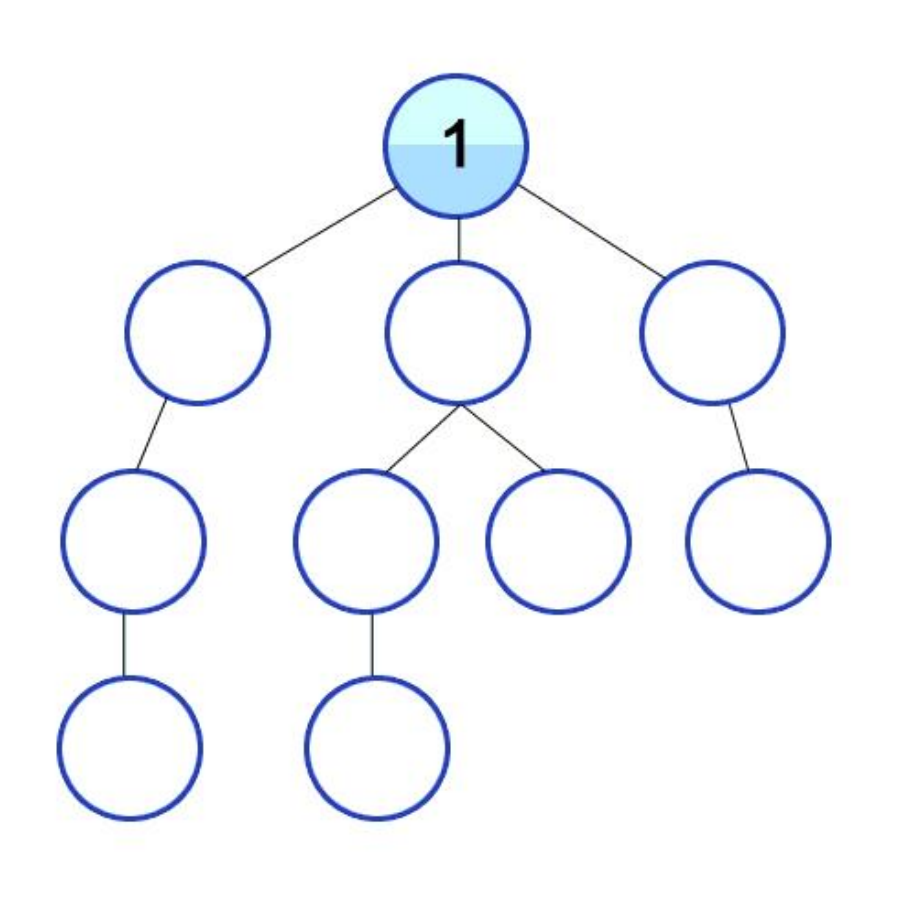
# Çizge Gezinme Algoritmaları (Graph Traversal)

- Çizgenin yapısını sistematik bir şekilde keşfetmek için kullanılır.
- İki ana kategoriye ayrılır:
  - derinlik öncelikli arama (DFS) ve
  - genişlik öncelikli arama (BFS).
- *DFS*, bir düğümden başlayarak mümkün olduğunca derinlere iner ve tüm komşularını ziyaret ettikten sonra geri döner.
- *BFS*, bir kuyruk veri yapısı kullanarak seviye seviye tüm düğümleri ziyaret eder.

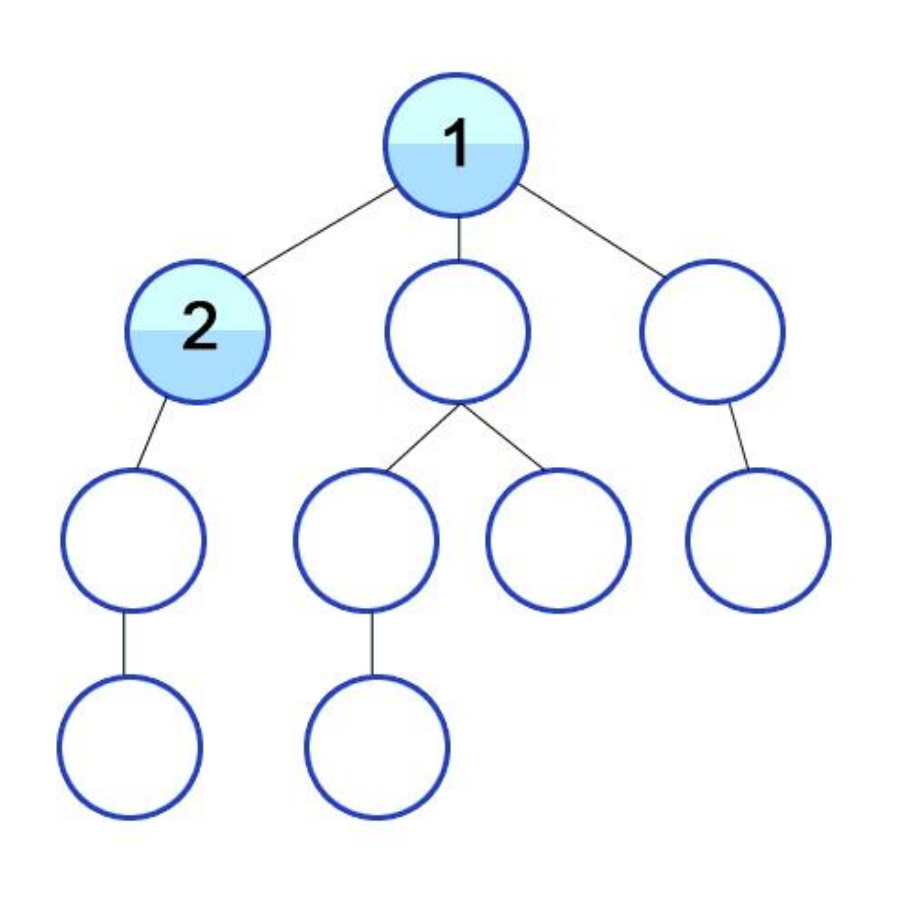
# Derinlik Öncelikli Arama



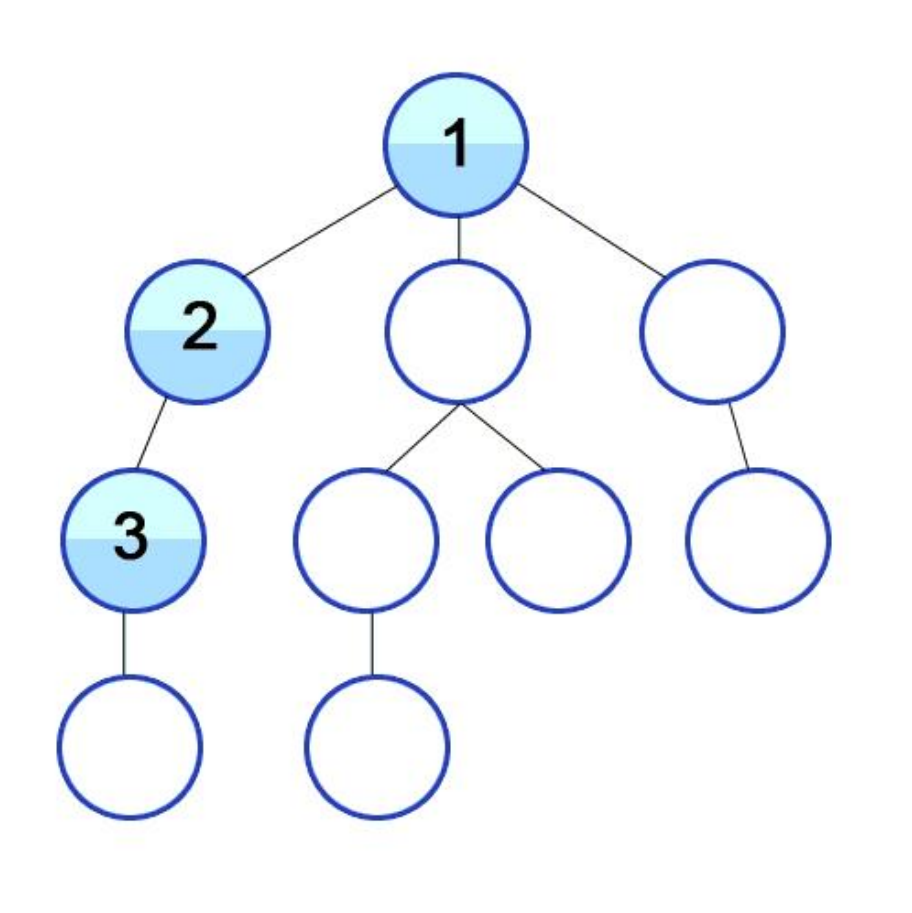
# Derinlik Öncelikli Arama



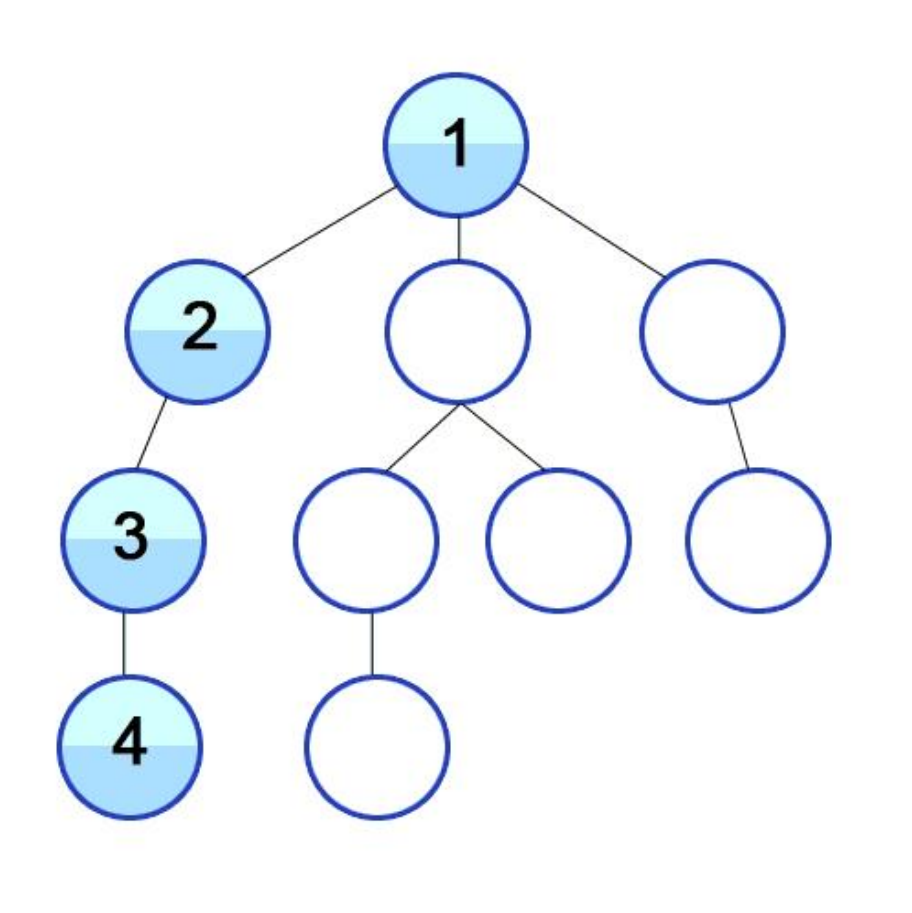
# Derinlik Öncelikli Arama



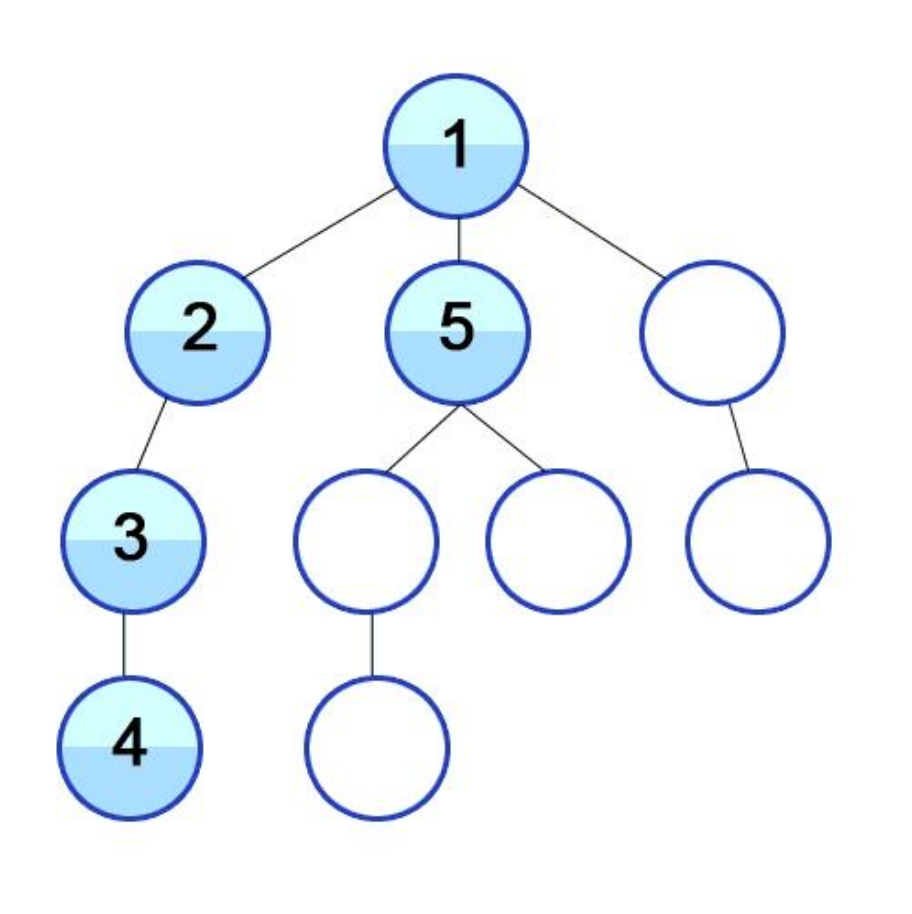
# Derinlik Öncelikli Arama



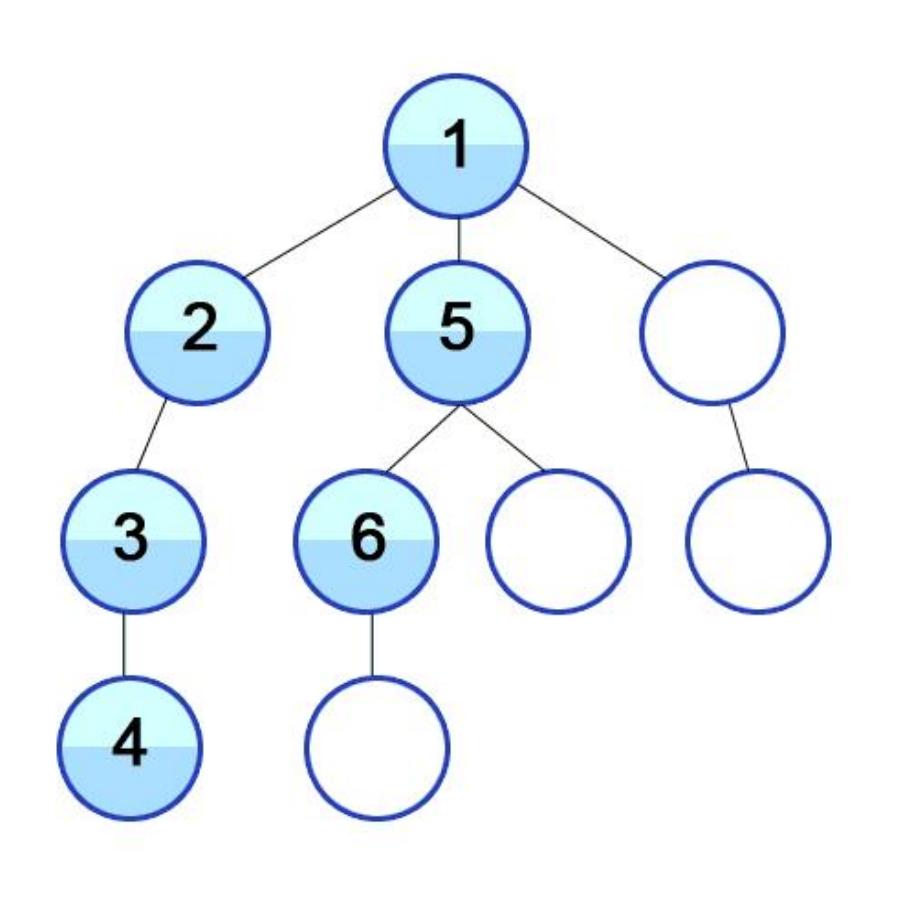
# Derinlik Öncelikli Arama



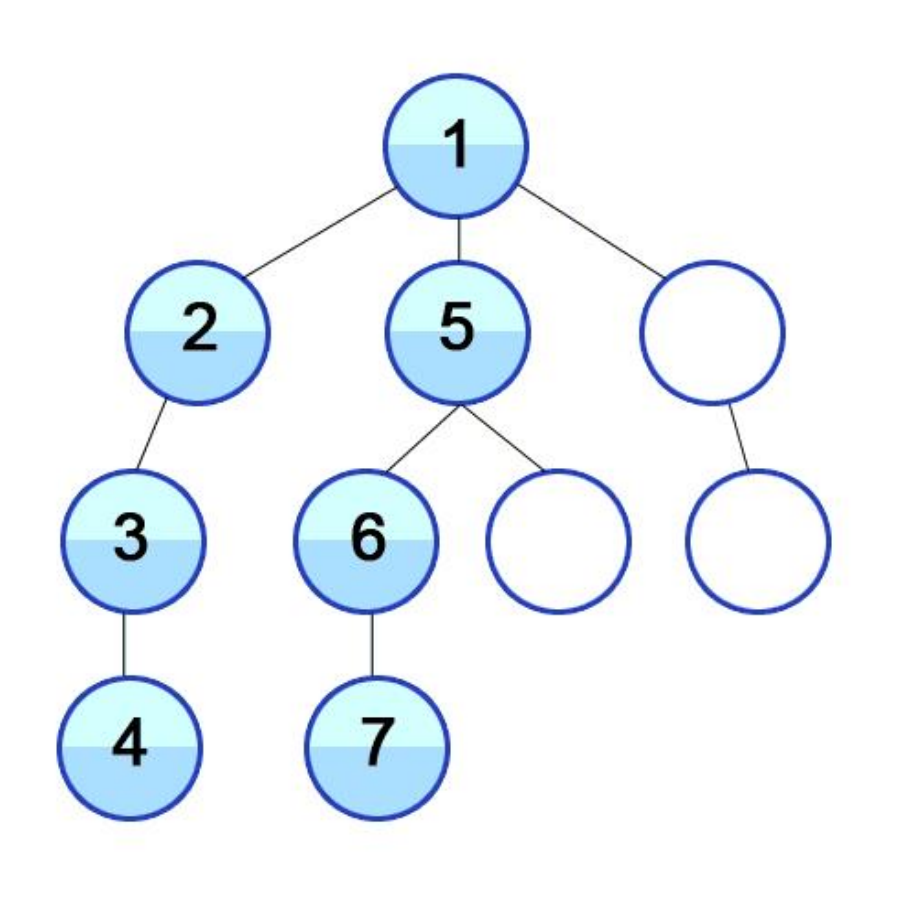
# Derinlik Öncelikli Arama



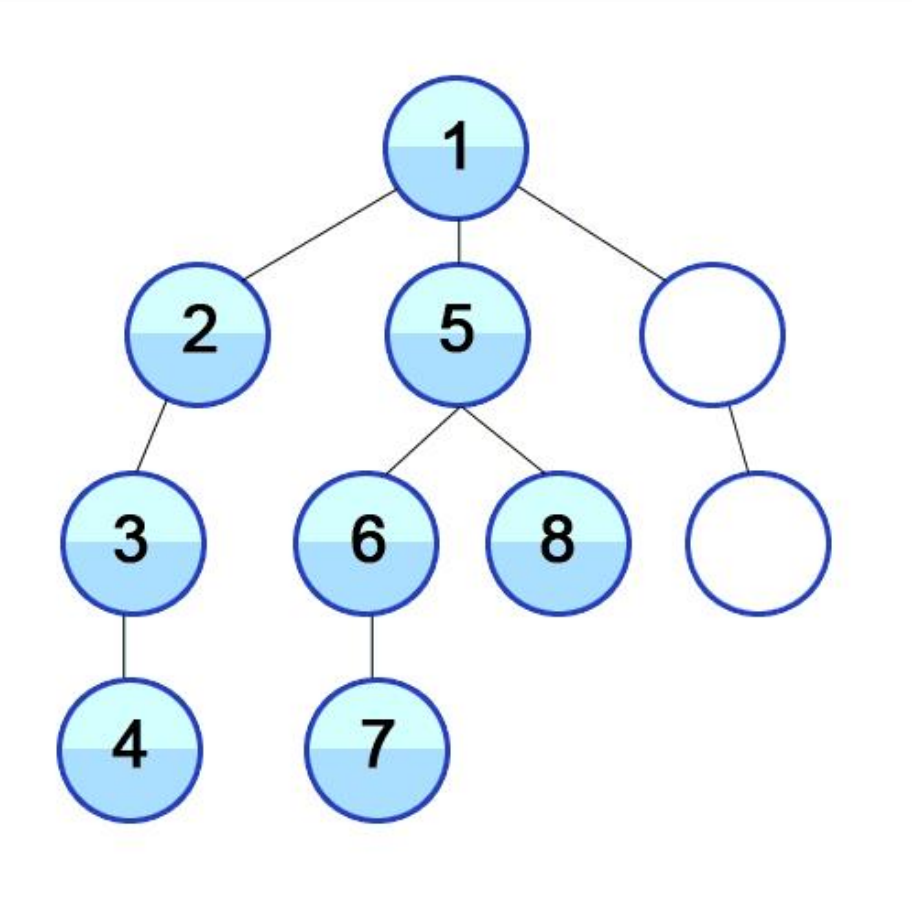
# Derinlik Öncelikli Arama



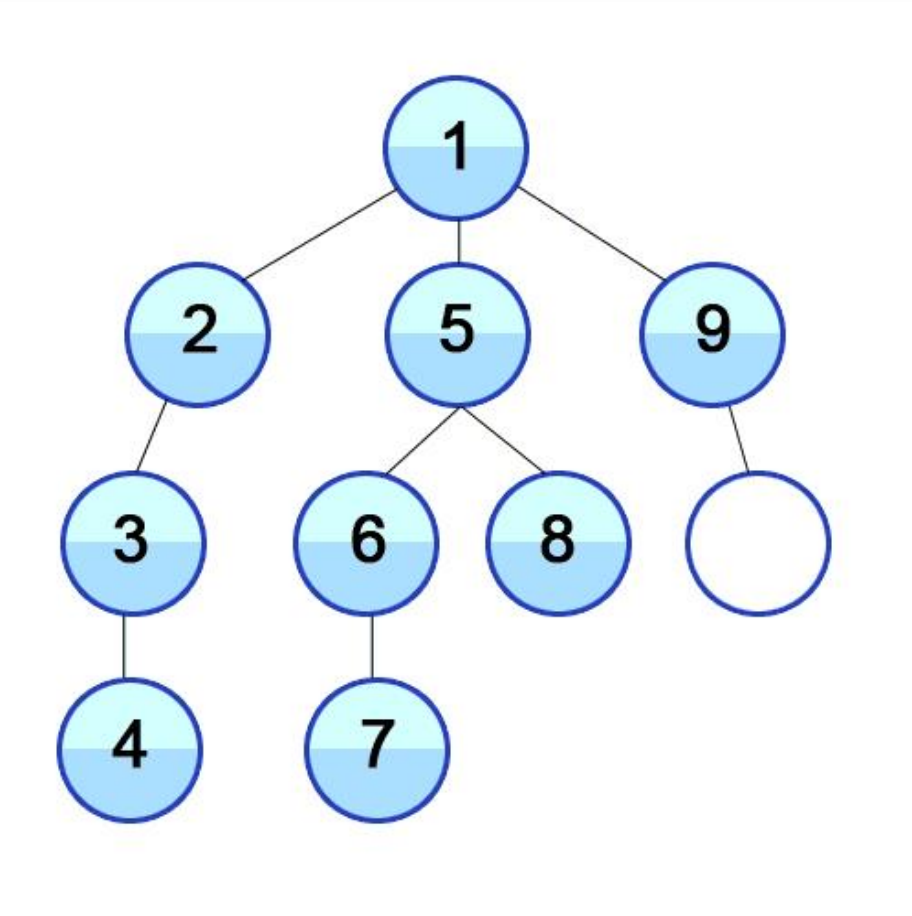
# Derinlik Öncelikli Arama



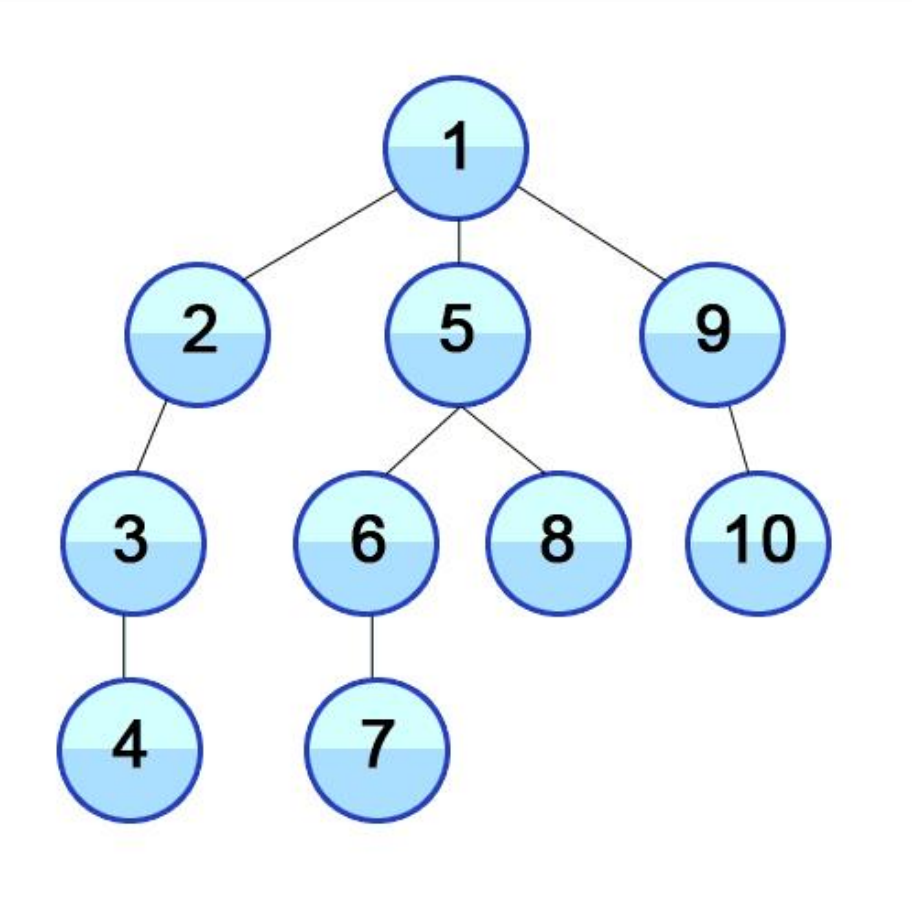
# Derinlik Öncelikli Arama



# Derinlik Öncelikli Arama



# Derinlik Öncelikli Arama



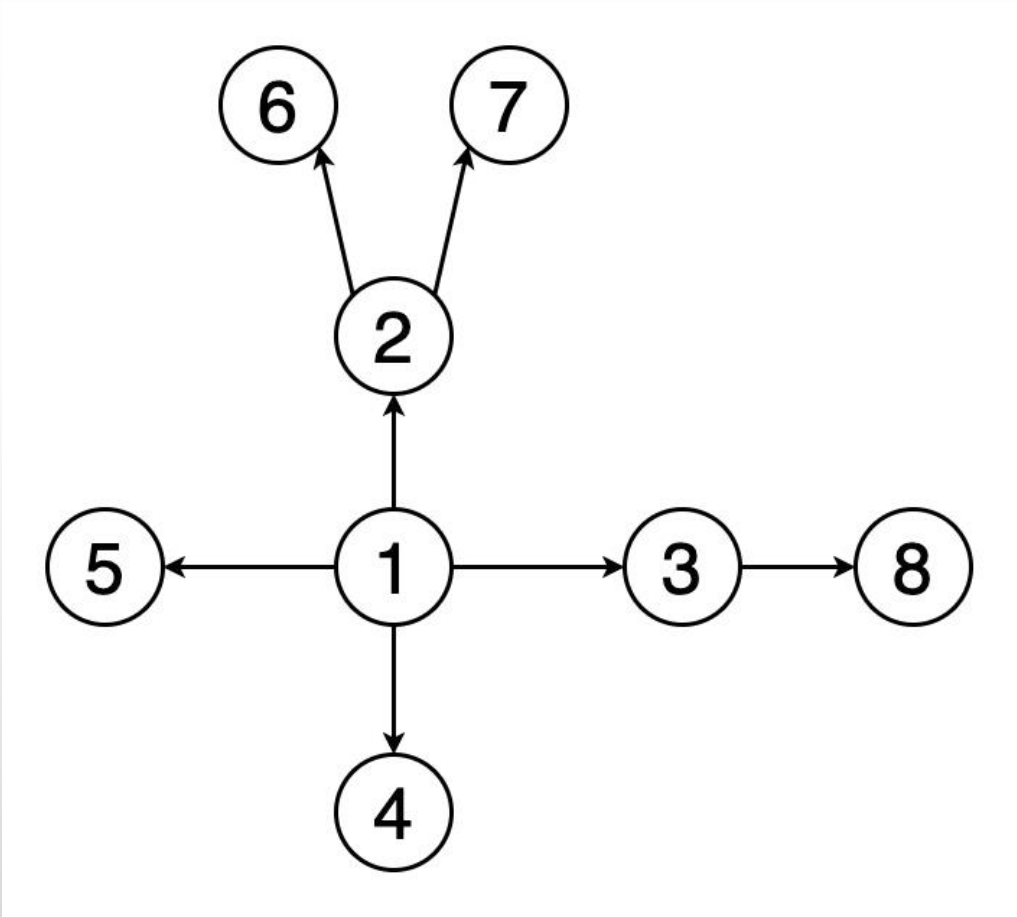
# Recursive DFS

```
procedure DFS(G, v) is
    label v as discovered
    for all directed edges from v to w that are in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

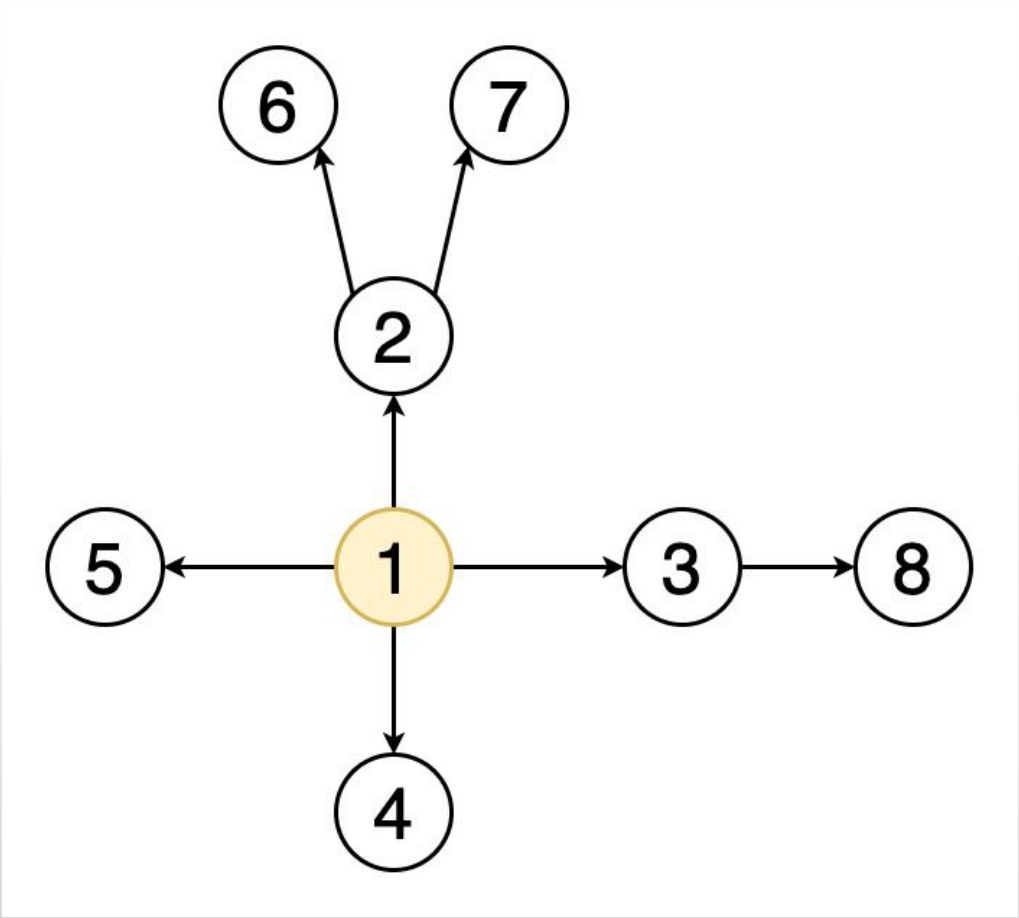
# Iterative DFS

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

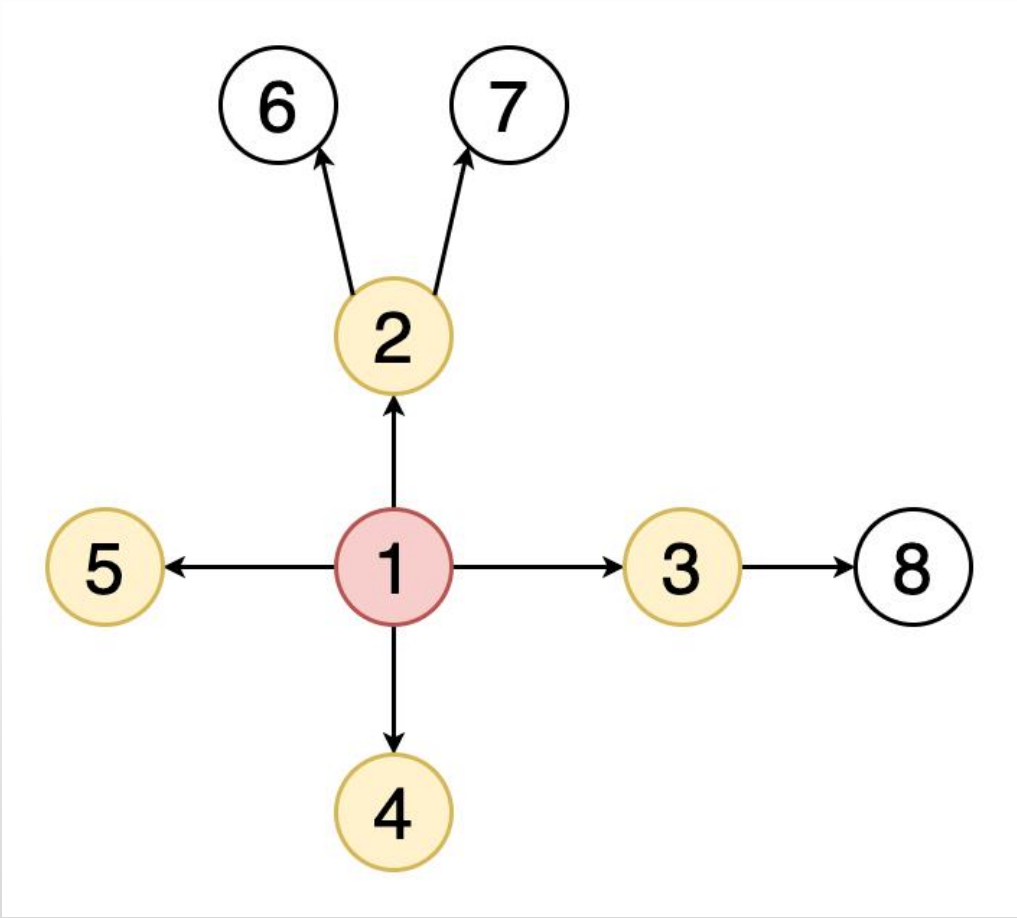
# Genişlik Öncelikli Arama



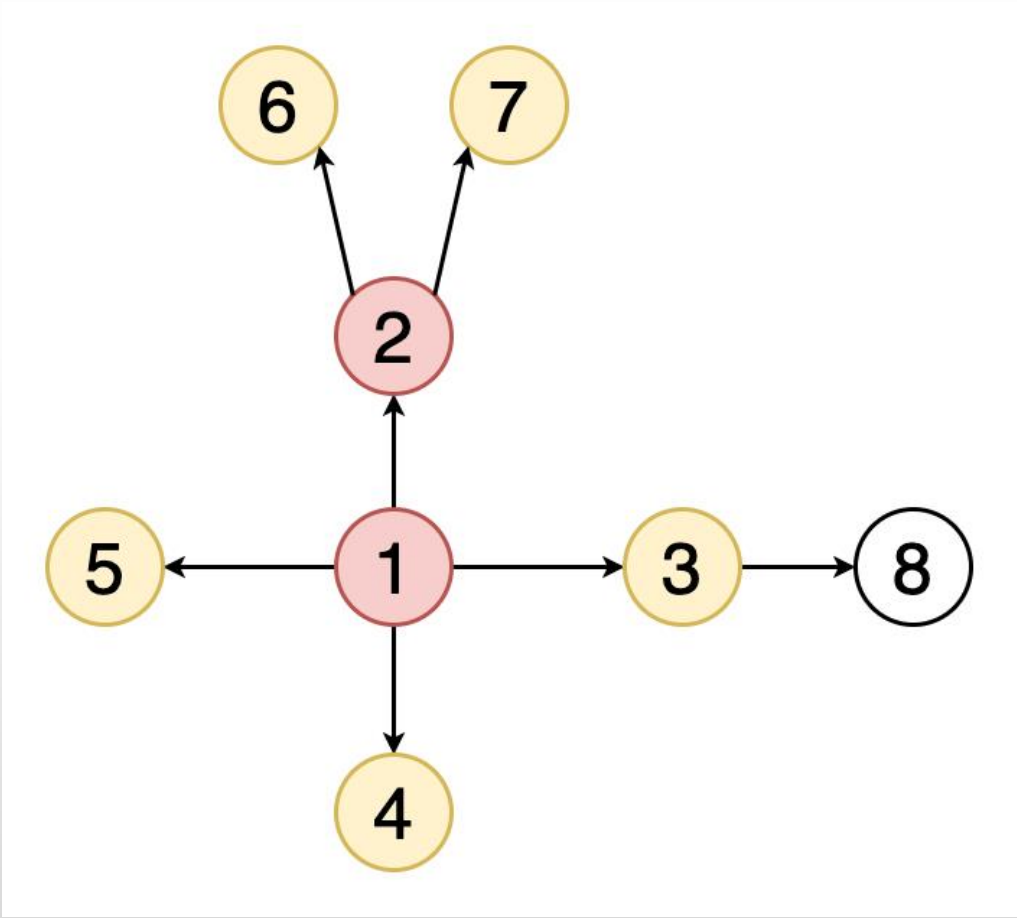
# Genişlik Öncelikli Arama



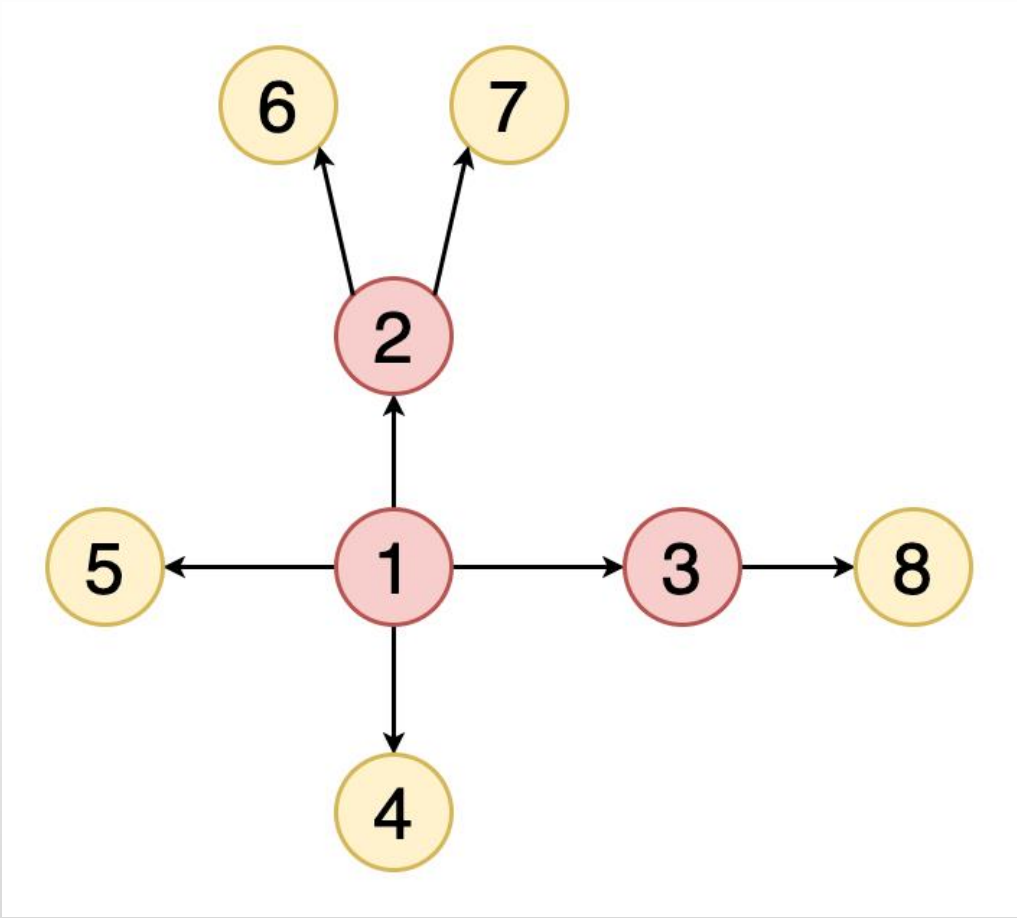
# Genişlik Öncelikli Arama



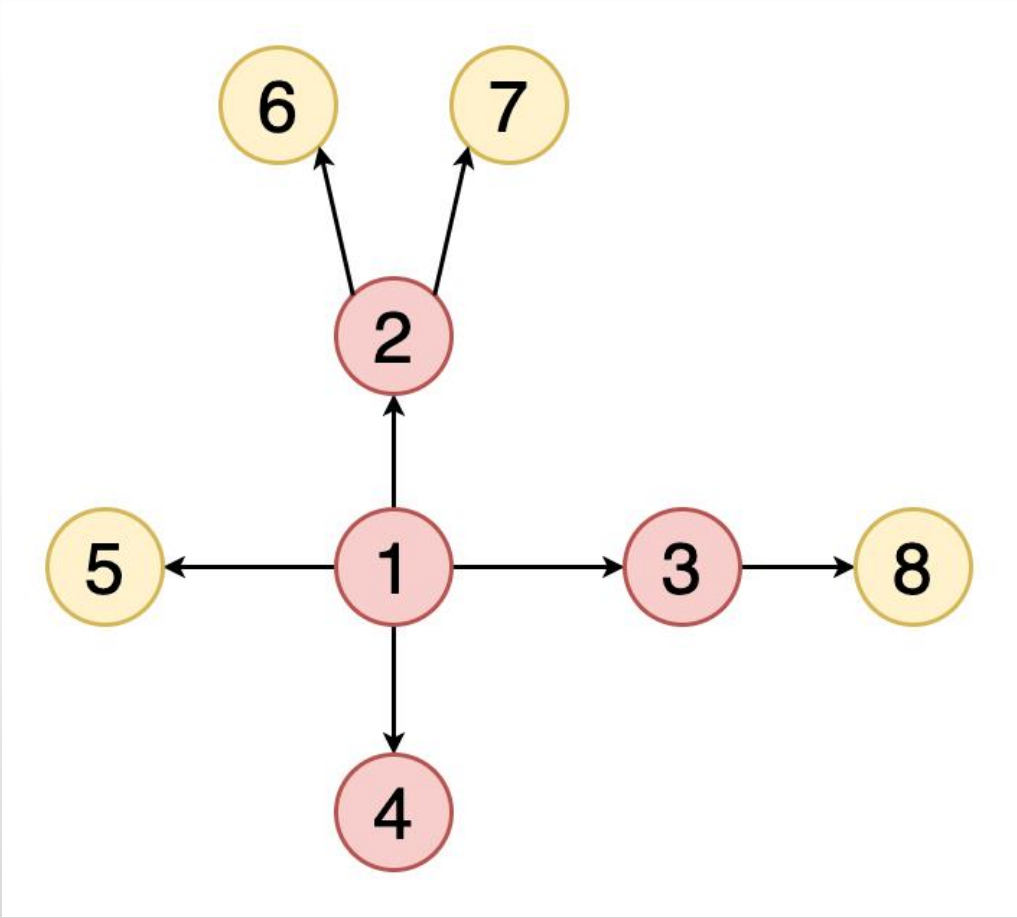
# Genişlik Öncelikli Arama



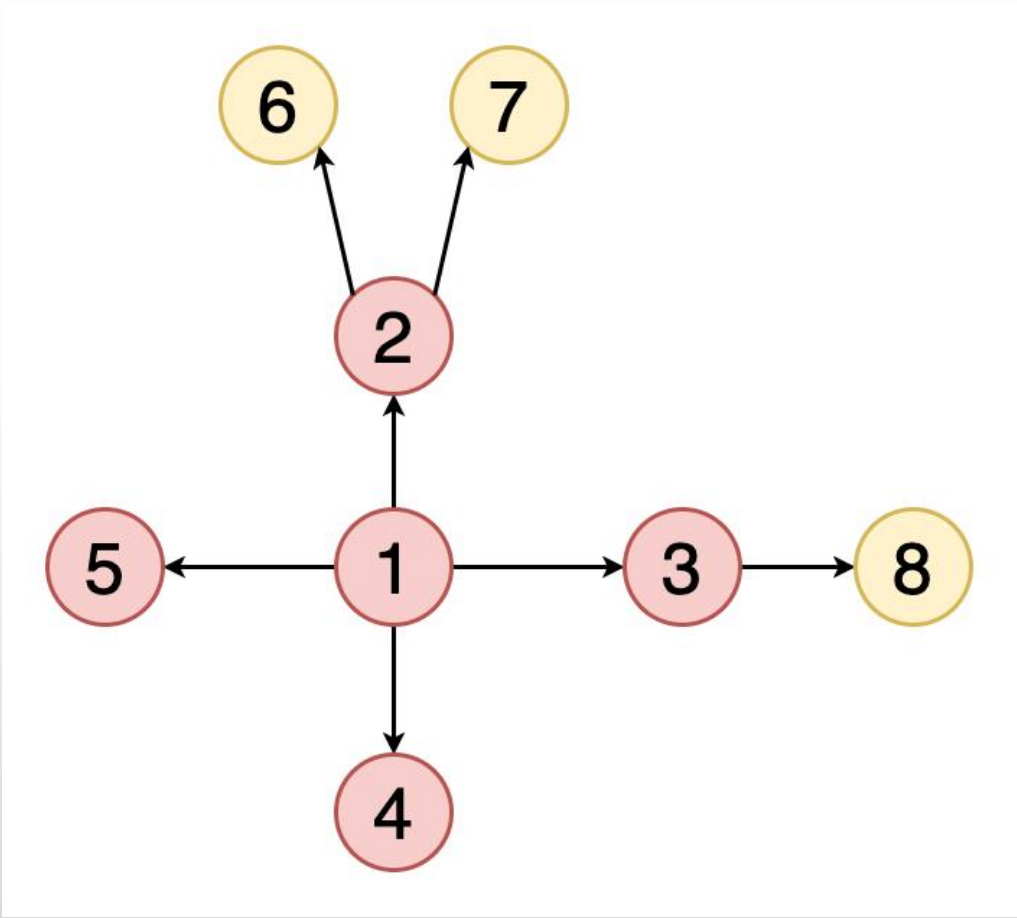
# Genişlik Öncelikli Arama



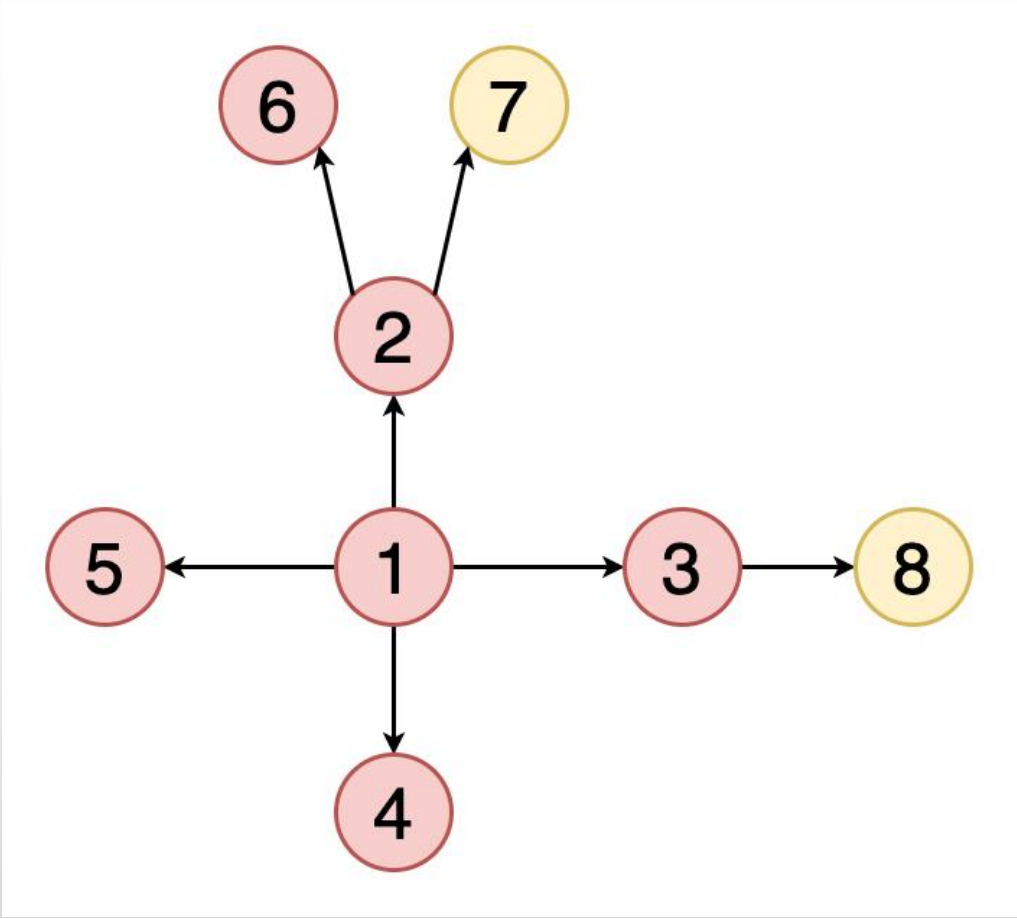
# Genişlik Öncelikli Arama



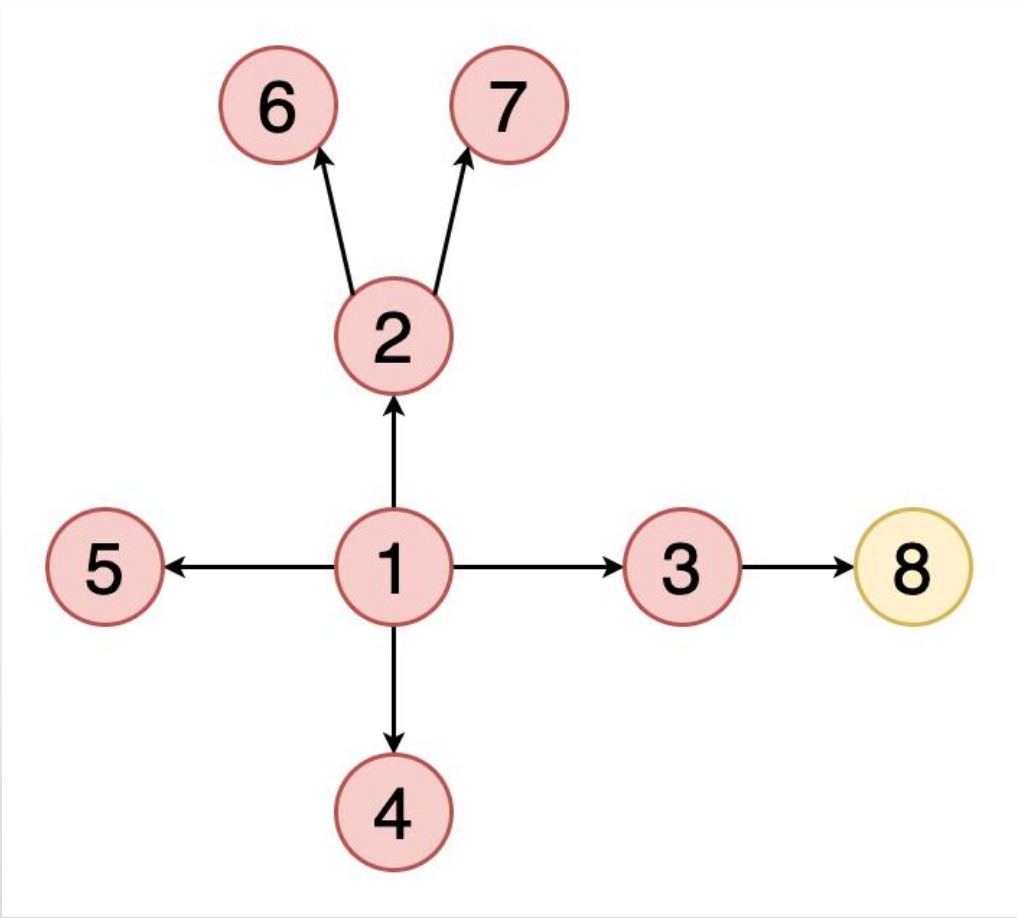
# Genişlik Öncelikli Arama



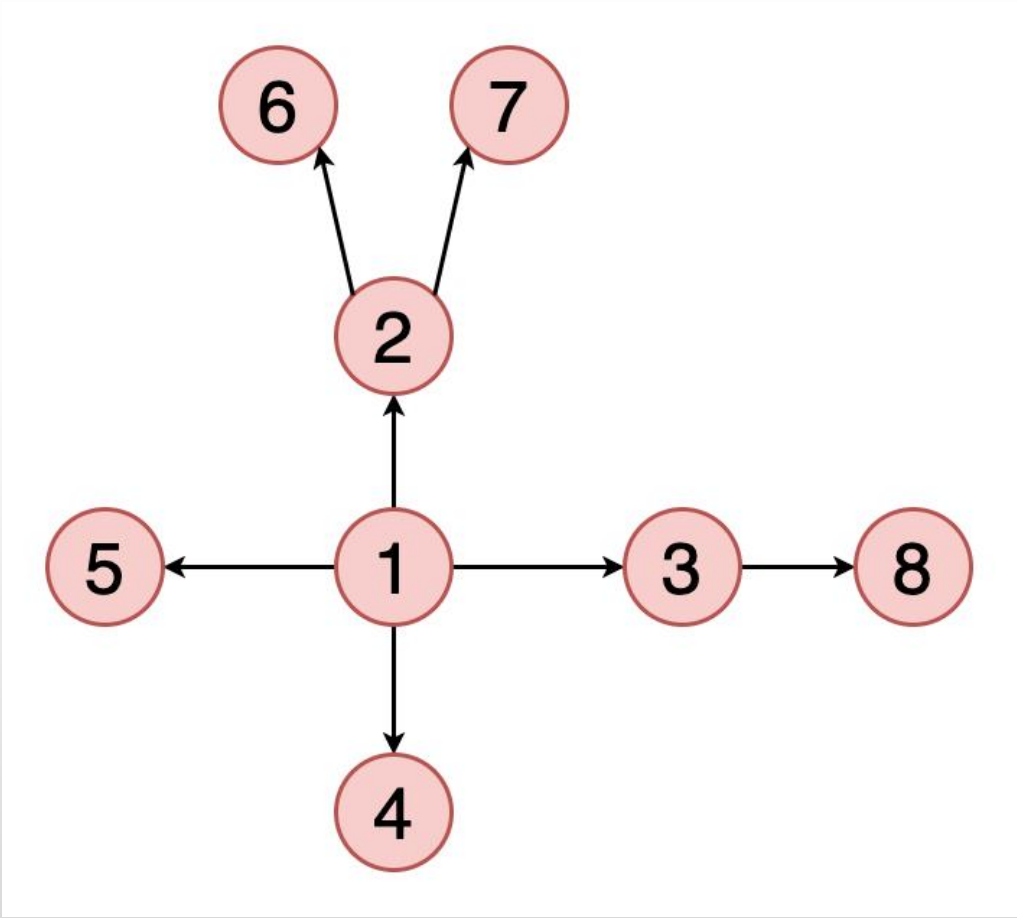
# Genişlik Öncelikli Arama



# Genişlik Öncelikli Arama



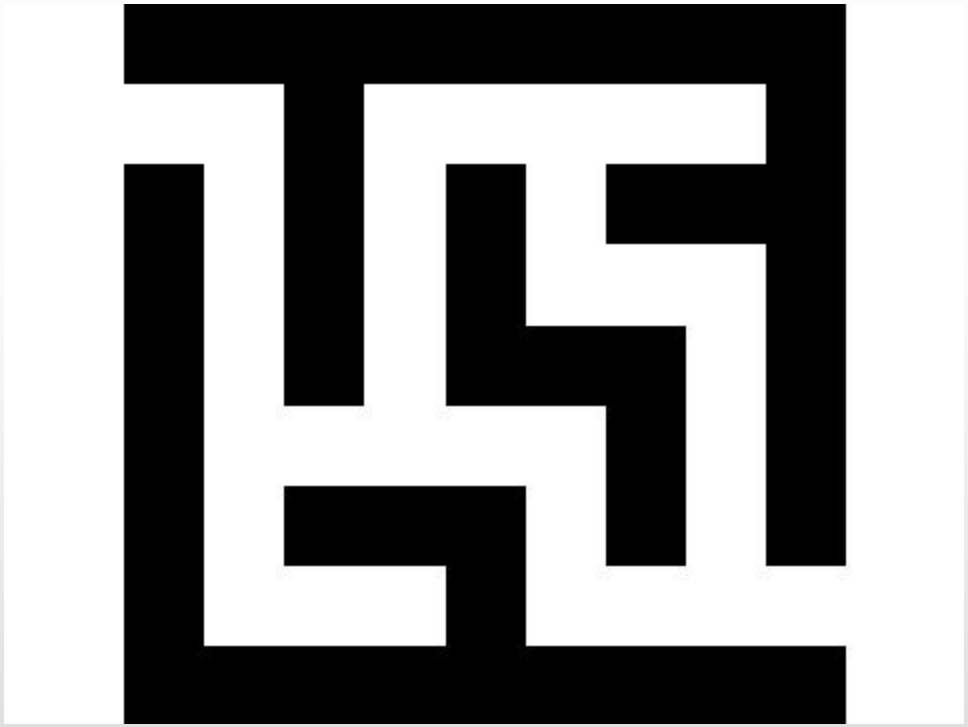
# Genişlik Öncelikli Arama



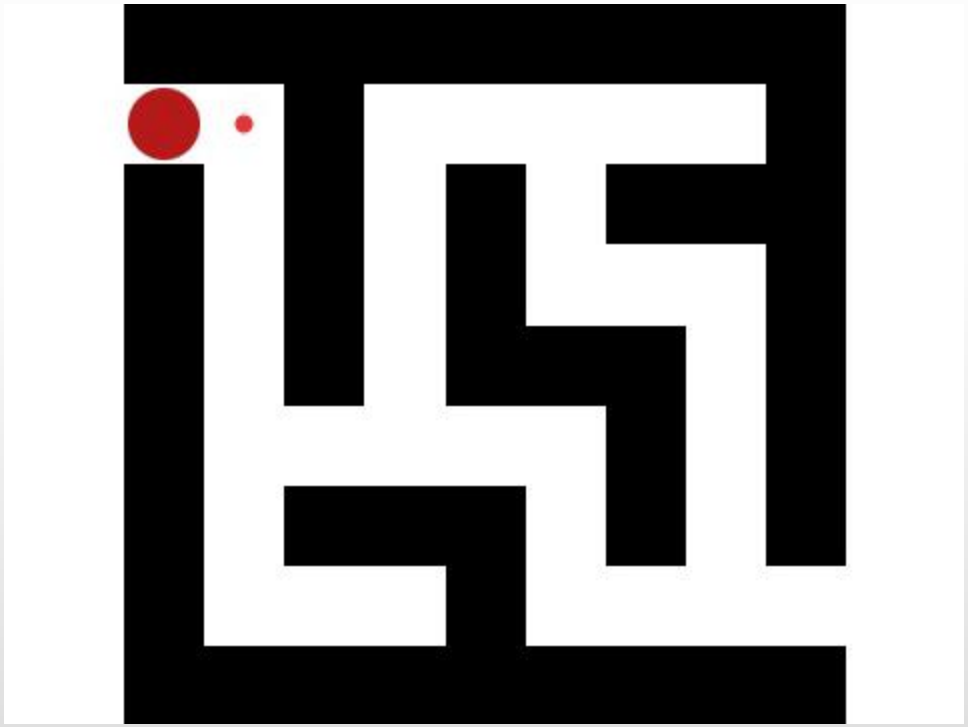
# Iterative BFS

```
procedure BFS(G, root) is
    let Q be a queue
    label root as explored
    Q.enqueue(root)
    while Q is not empty do
        v := Q.dequeue()
        if v is the goal then
            return v
        for all edges from v to w in G.adjacentEdges(v) do
            if w is not labeled as explored then
                label w as explored
                w.parent := v
                Q.enqueue(w)
```

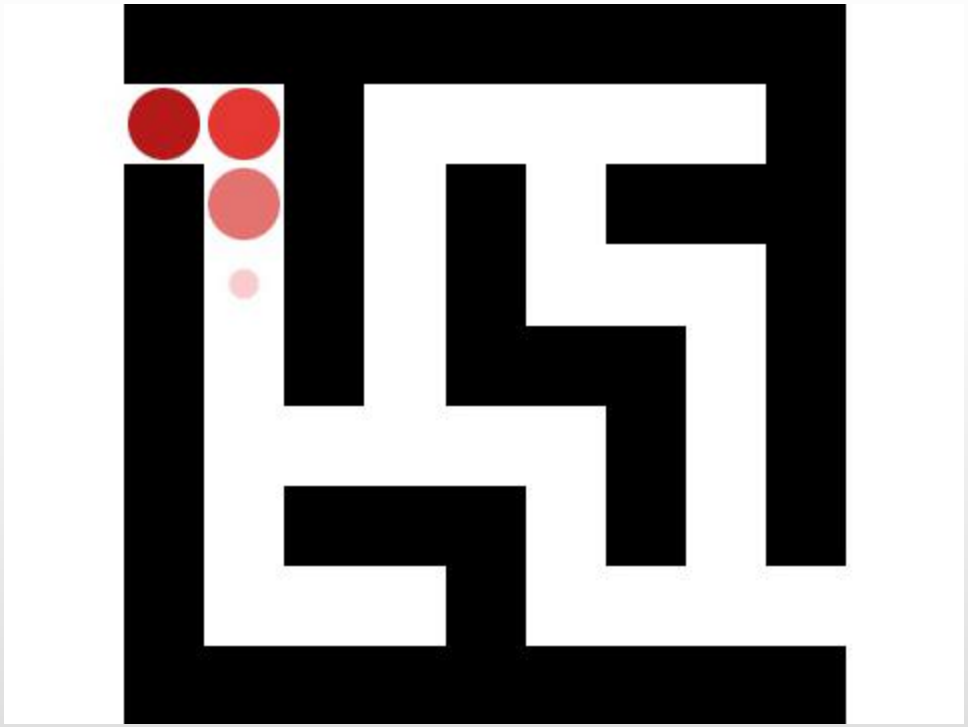
# BFS Search Way



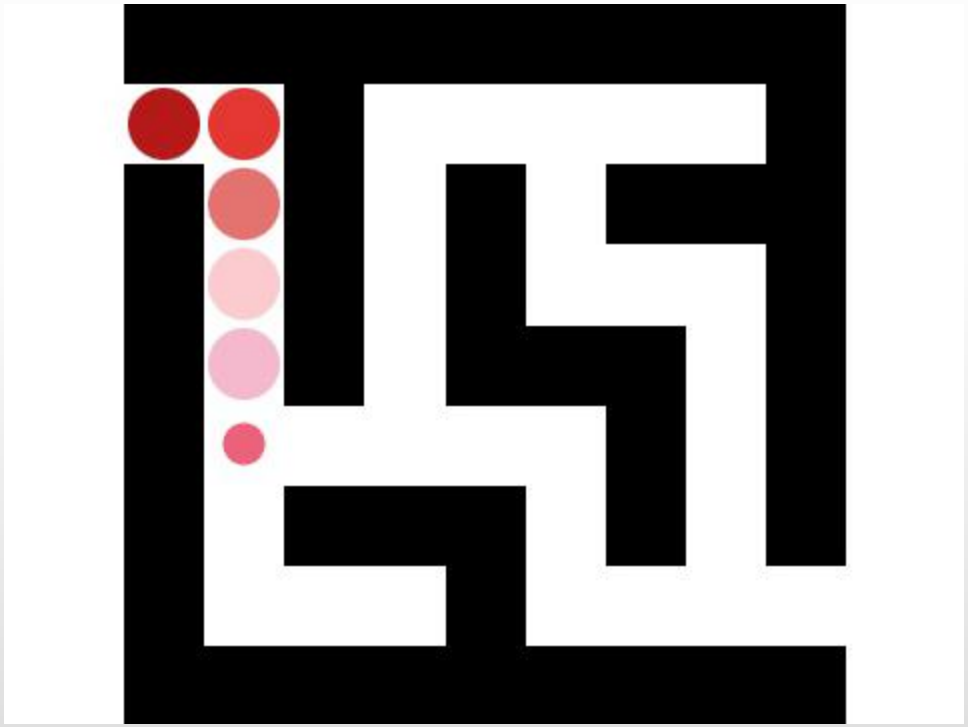
# BFS Search Way



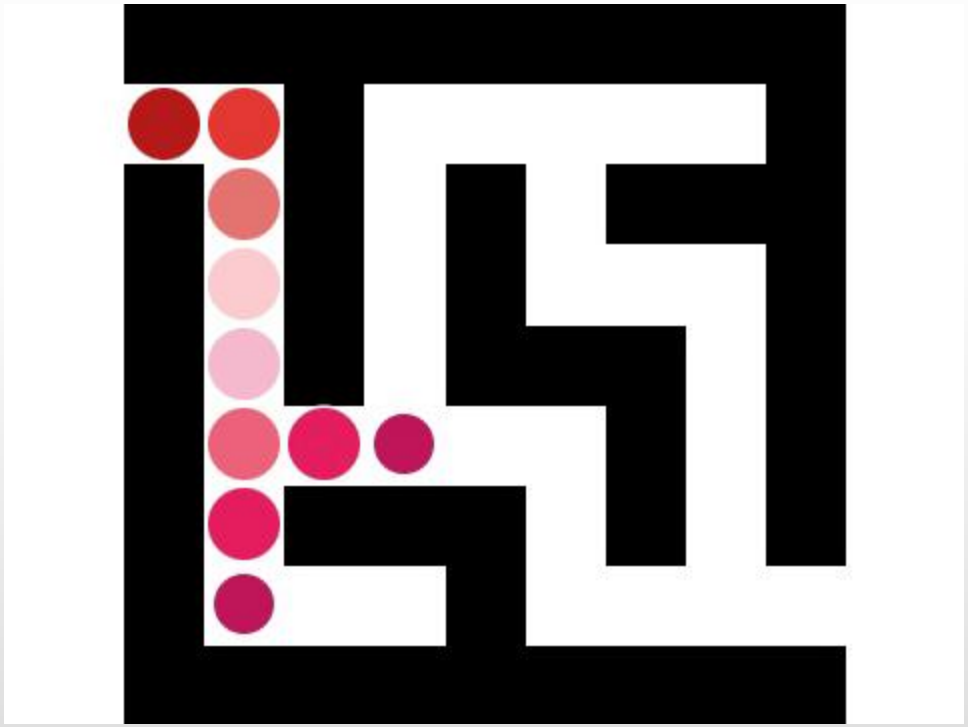
# BFS Search Way



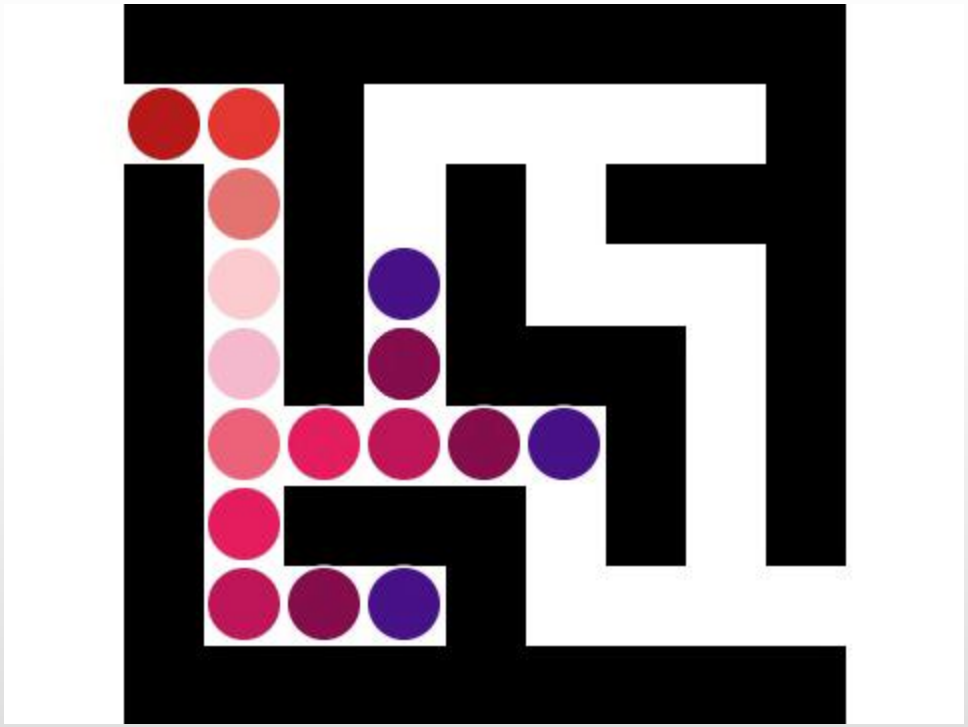
# BFS Search Way



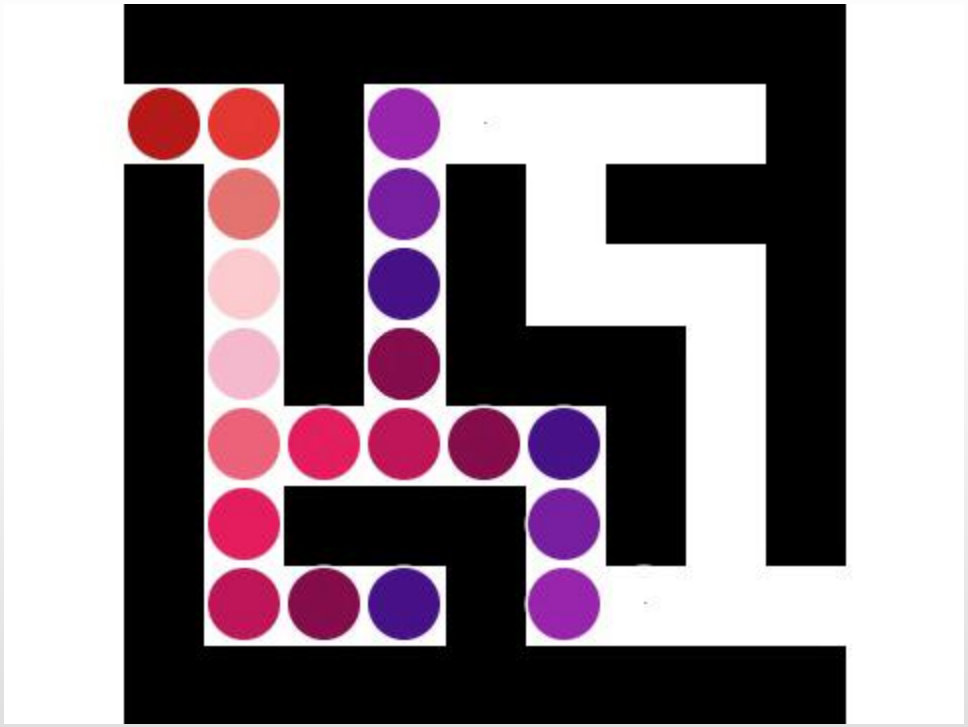
# BFS Search Way



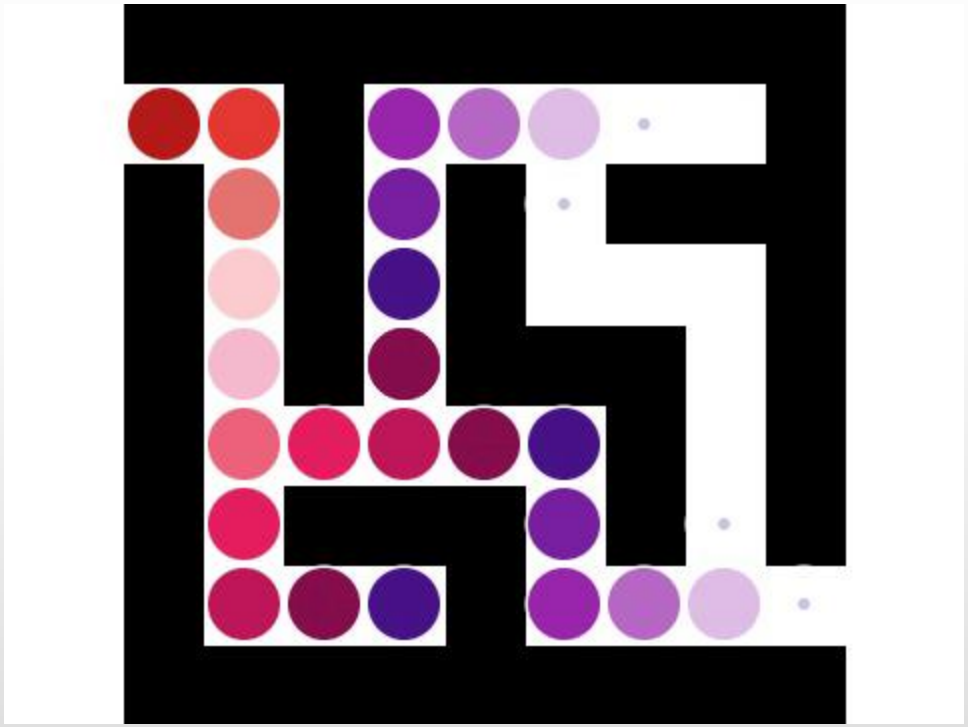
# BFS Search Way



# BFS Search Way



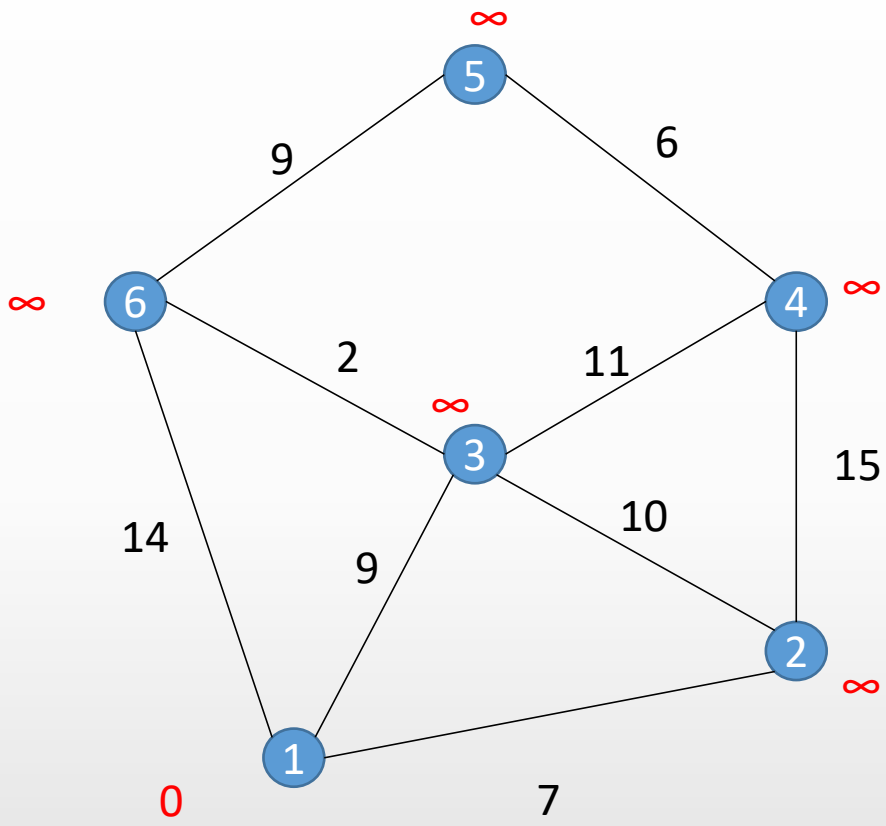
# BFS Search Way



# En Kısa Yol Algoritmaları (Shortest Path)

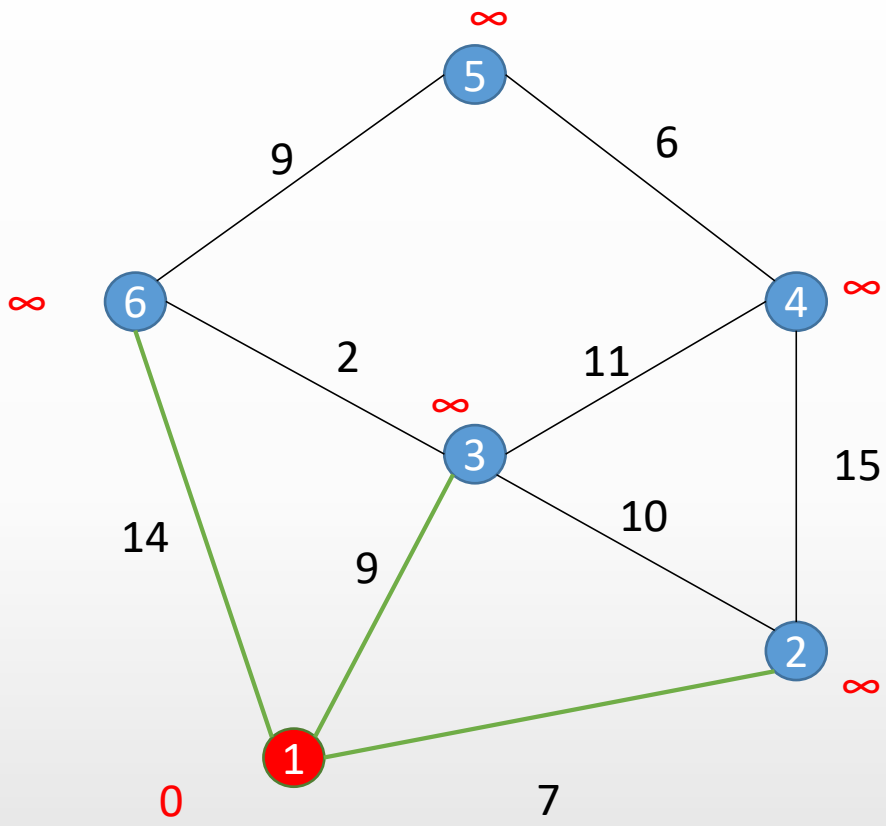
- Başlangıç düğümünden hedef düğüme giden en kısa yolu bulur.
- *Dijkstra*: Başlangıç düğümünden diğer tüm düğümlere olan en kısa yolları bulur. Ağırlıklar pozitif değer olmalıdır.
- *Bellman-Ford*: Negatif ağırlıklı kenar içeren çizgelerde kullanılabilir. Dijkstra Algoritmasından daha yavaştır.
- *A* Arama*: Sezgisel bilgiler kullanılarak aramayı hızlandırır. Hedef düğüme olan tahmini mesafeyi hesaba katar.

# Dijkstra



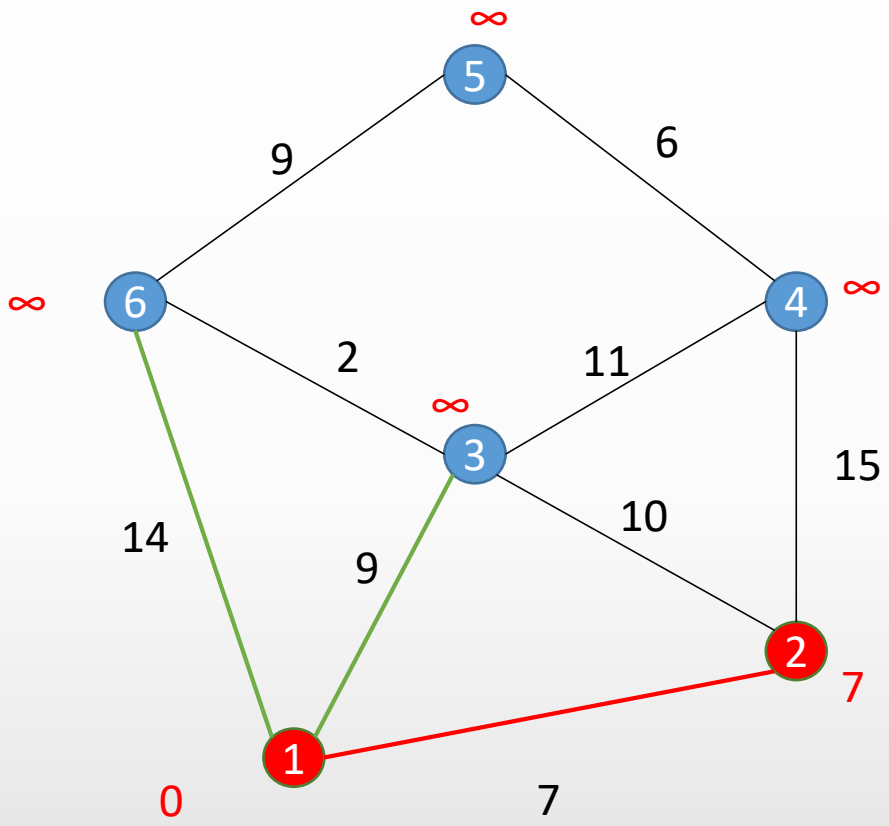

| | 1 |
|---|---|
| 1 | 0 |
| 2 | ∞ |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |

# Dijkstra



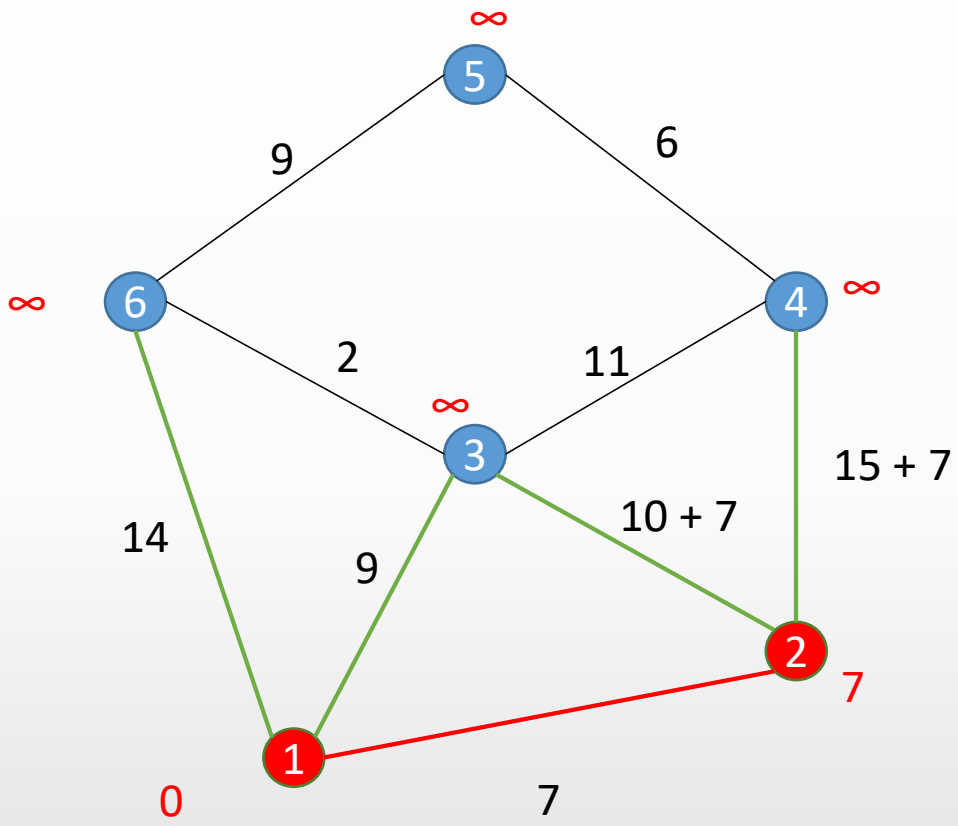

| | 1 |
|---|---|
| 1 | 0 |
| 2 | ∞ |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |

# Dijkstra



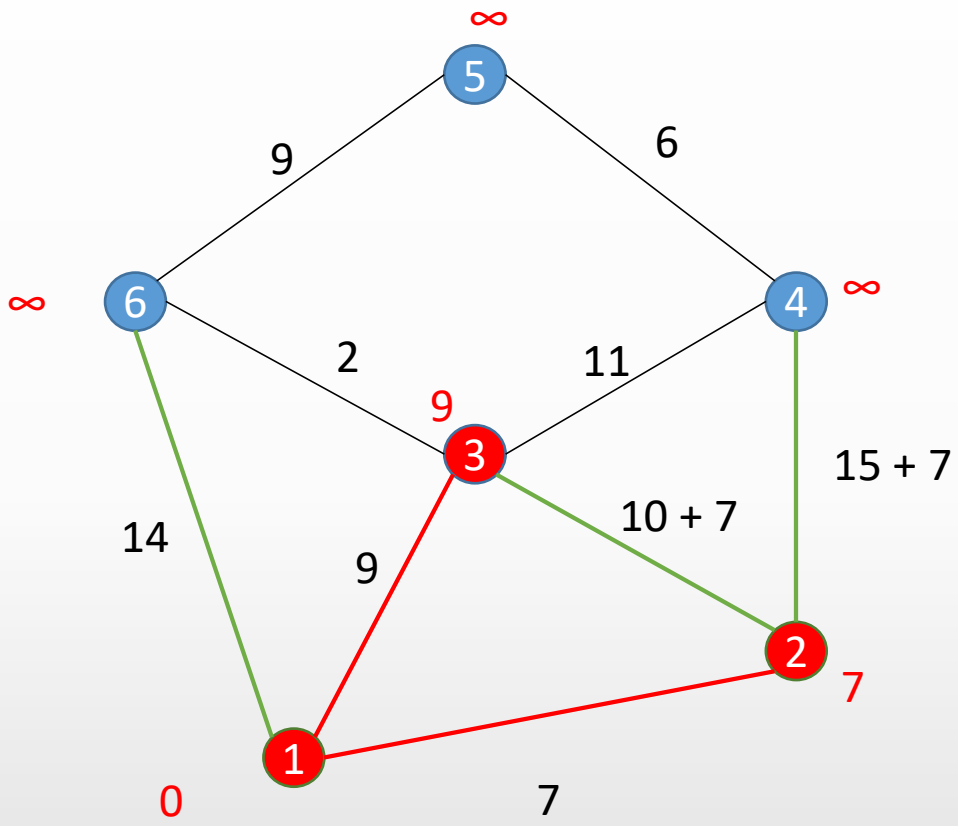

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |

# Dijkstra



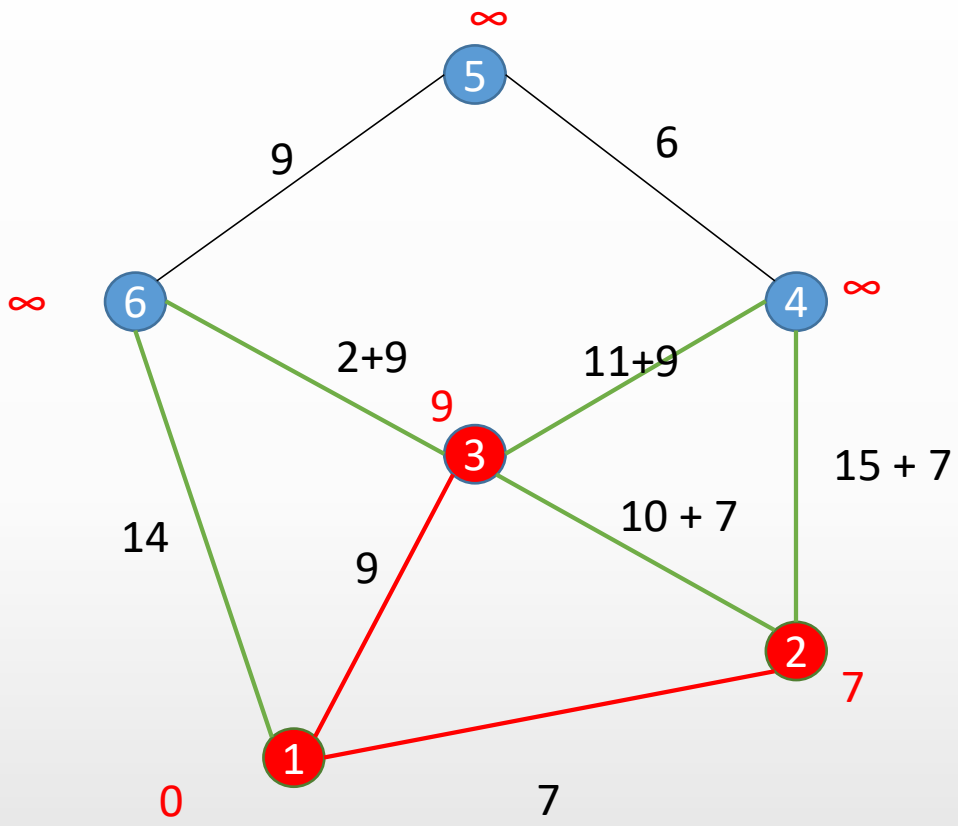

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | ∞ |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |

# Dijkstra



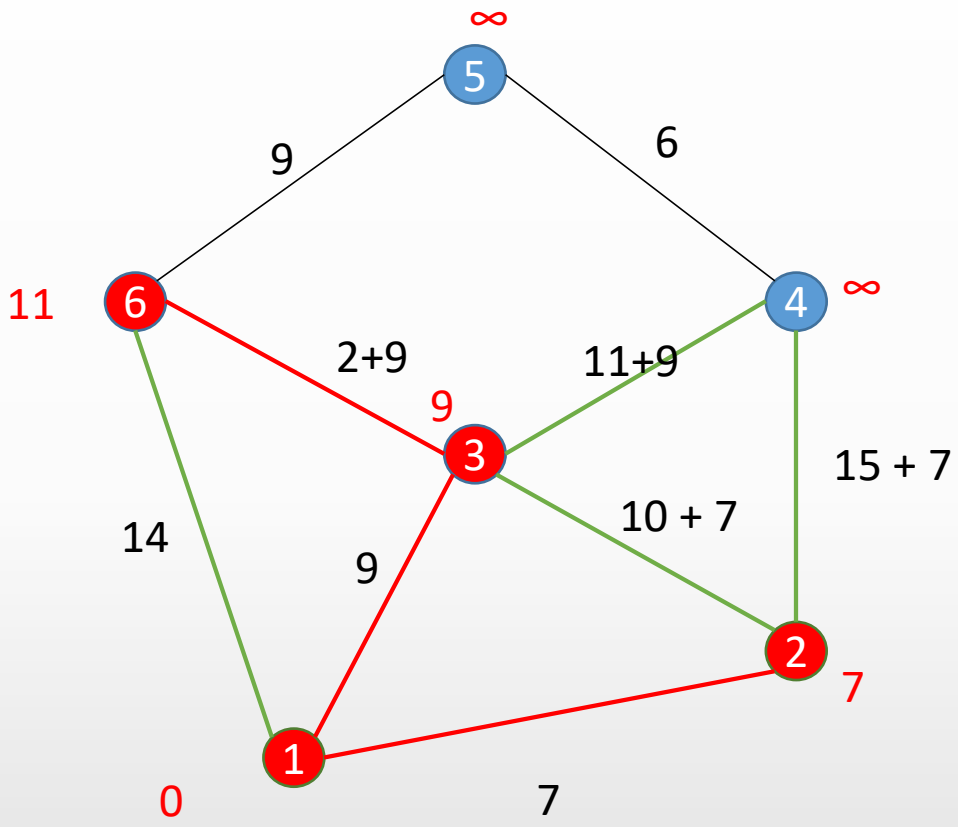

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |

# Dijkstra



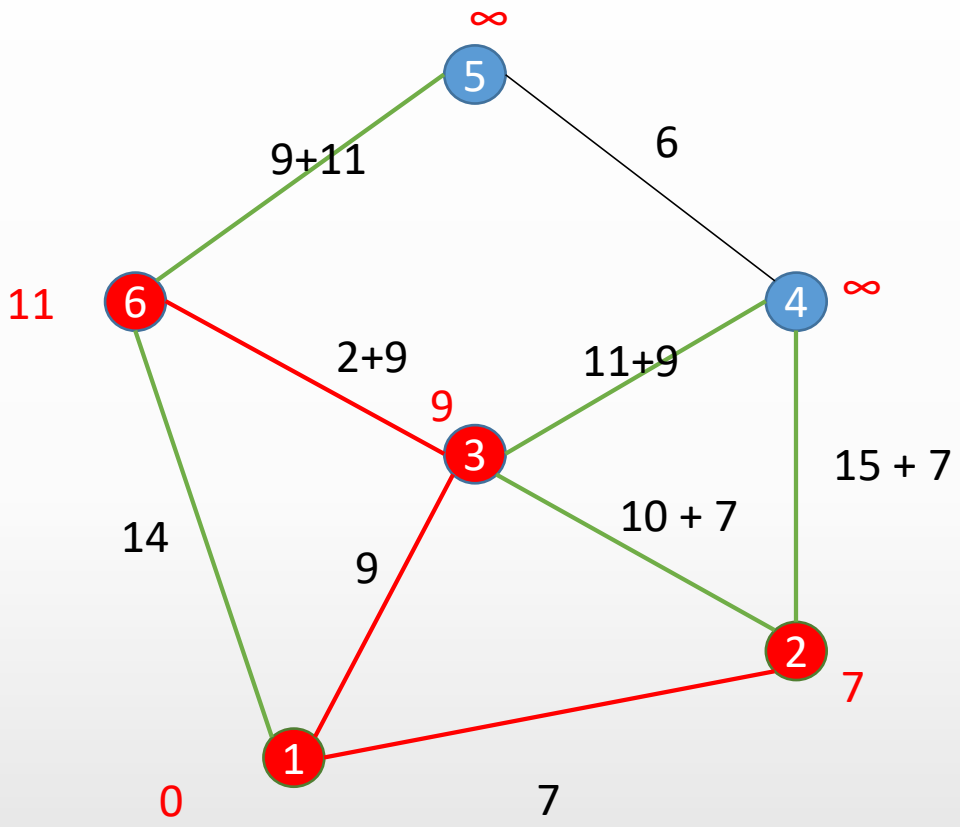

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | ∞ |
| 5 | ∞ |
| 6 | ∞ |

# Dijkstra



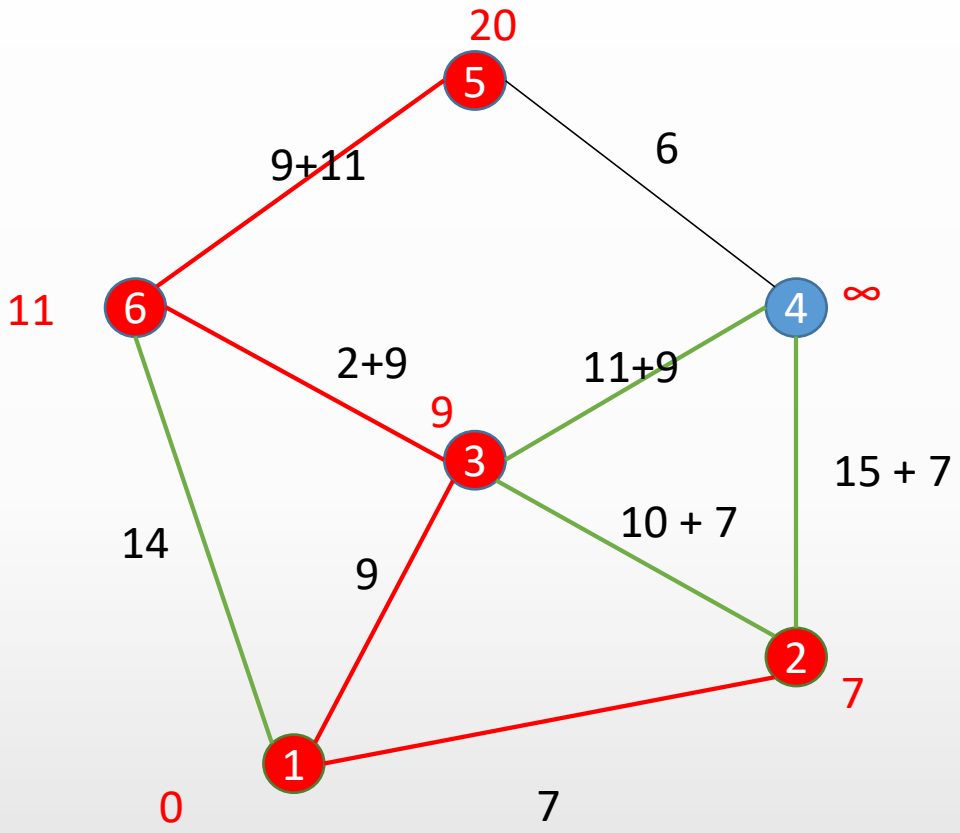

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | ∞ |
| 5 | ∞ |
| 6 | 11 |

# Dijkstra



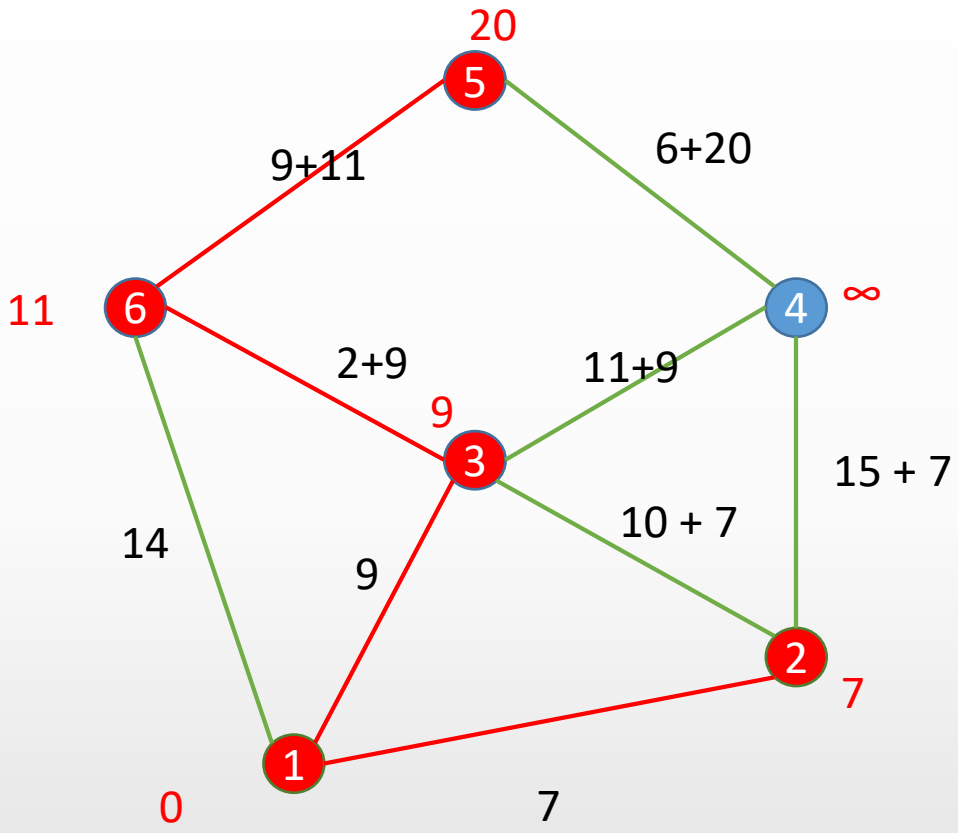

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | ∞ |
| 5 | ∞ |
| 6 | 11 |

# Dijkstra



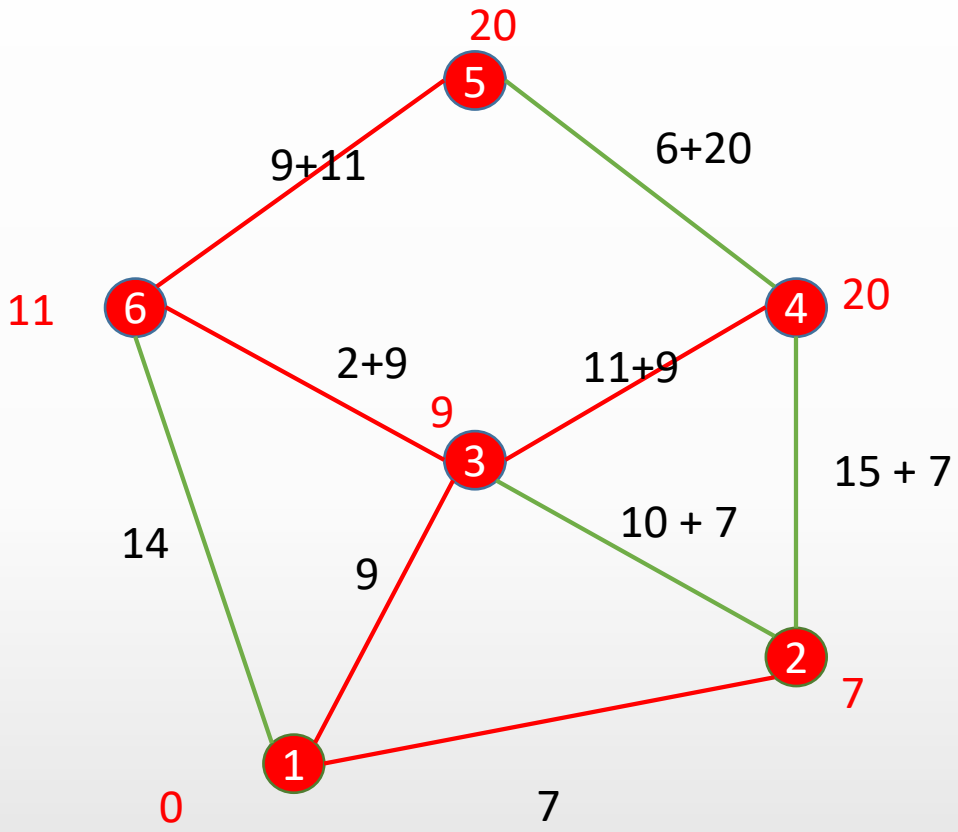

| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | ∞ |
| 5 | 20 |
| 6 | 11 |

# Dijkstra




| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | ∞ |
| 5 | 20 |
| 6 | 11 |

# Dijkstra




| | 1 |
|---|---|
| 1 | 0 |
| 2 | 7 |
| 3 | 9 |
| 4 | 20 |
| 5 | 20 |
| 6 | 11 |

# Dijkstra

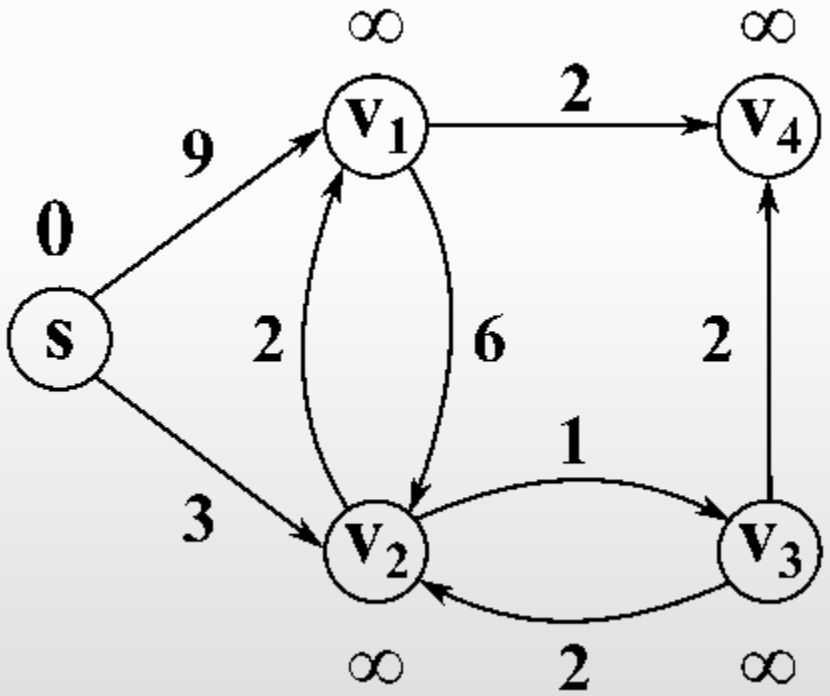
```
function Dijkstra(Graph, source):

    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0
```

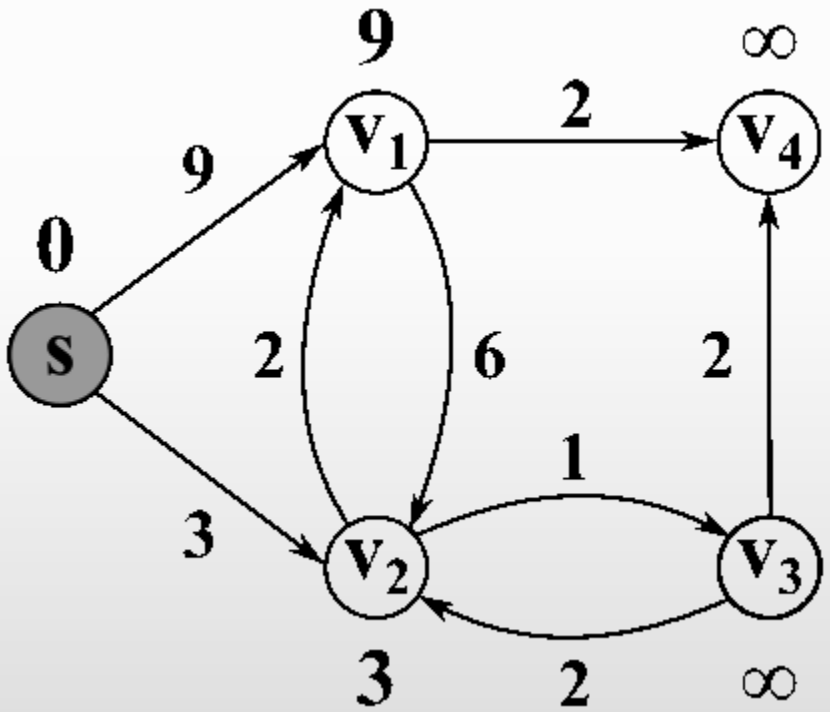
# Dijkstra

```
while Q is not empty:
    u ← vertex in Q with min dist[u]
    remove u from Q
    for each neighbor v of u still in Q:
        alt ← dist[u] + Graph.Edges(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u
return dist[], prev[]
```

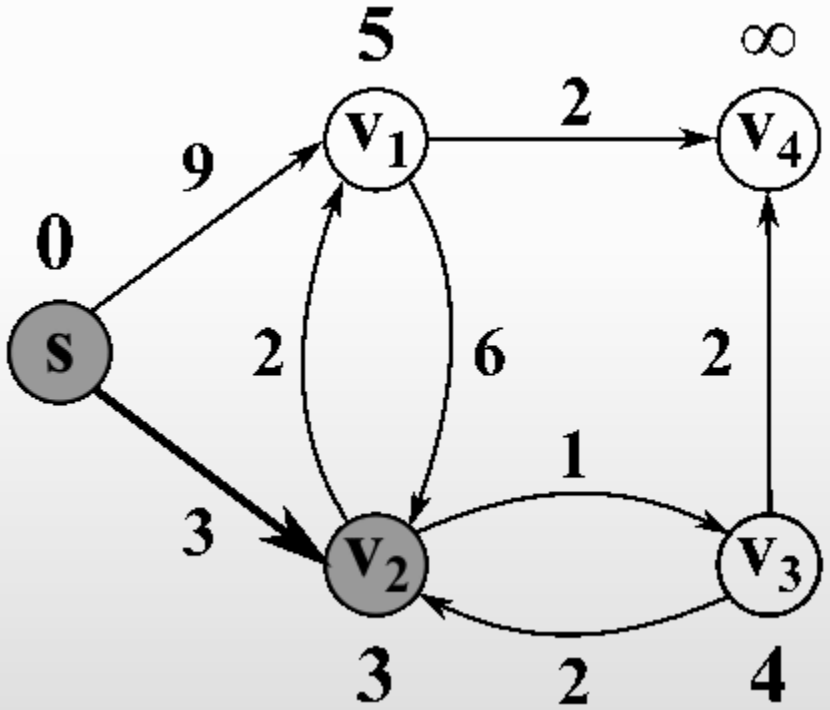
# Bellman Ford



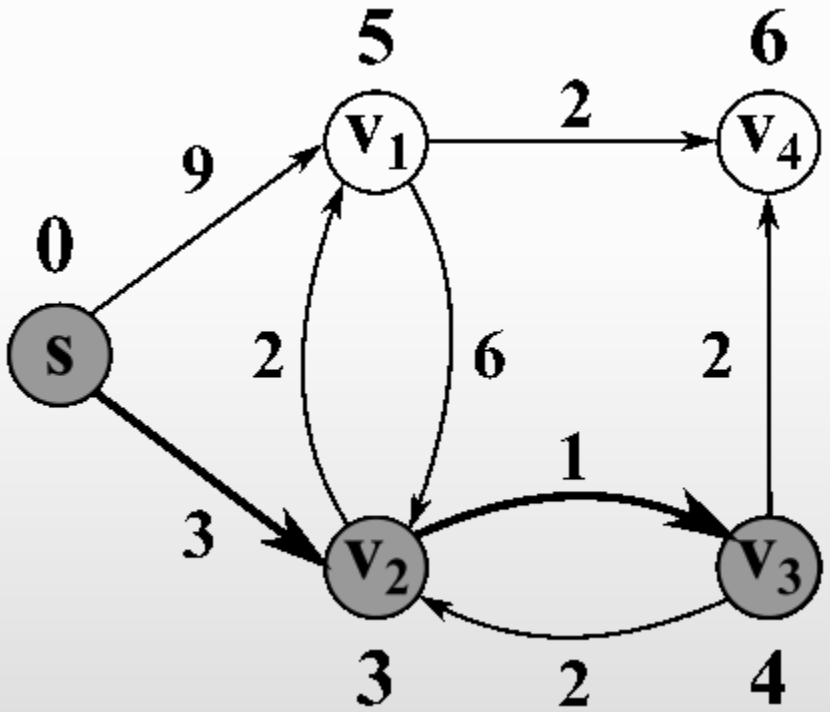
# Bellman Ford



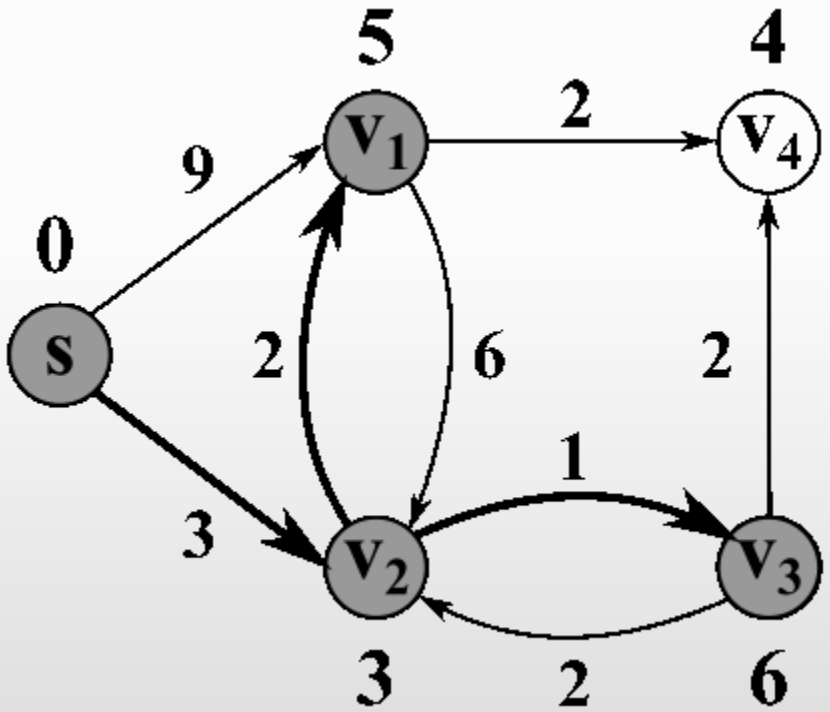
# Bellman Ford



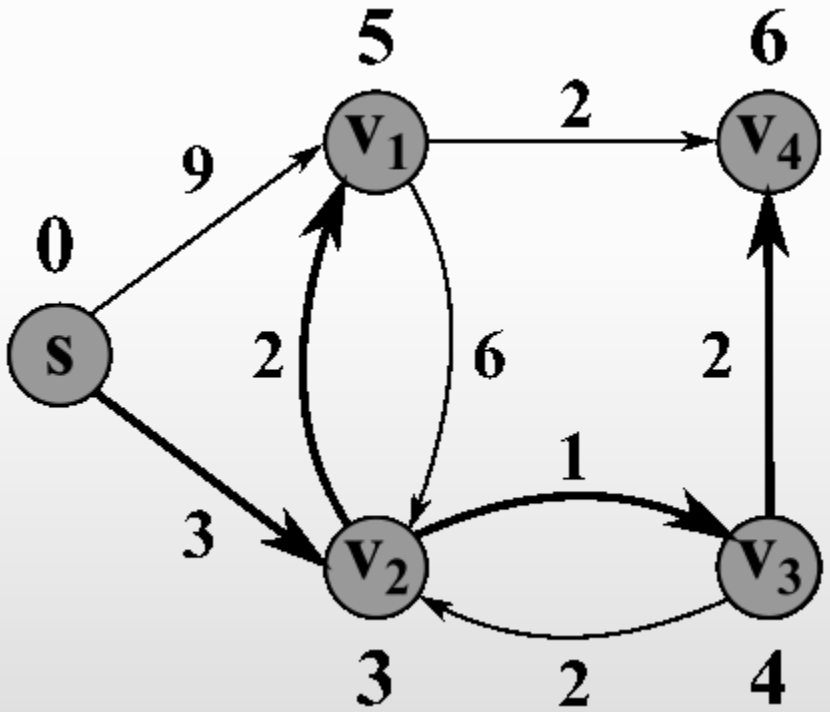
# Bellman Ford



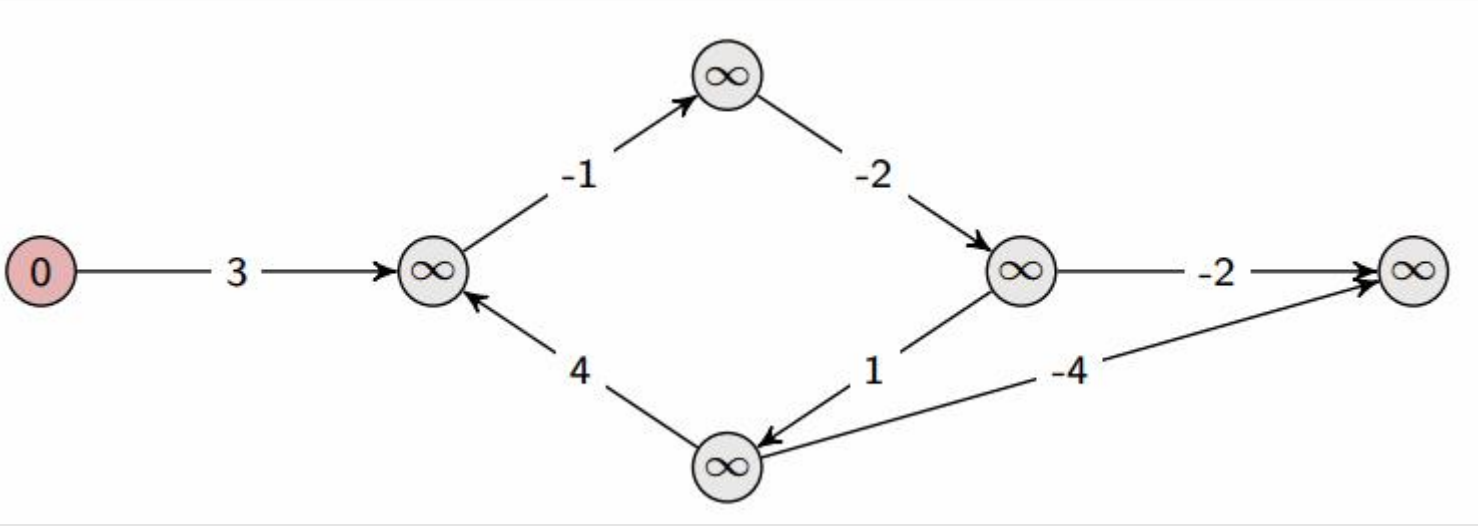
# Bellman Ford



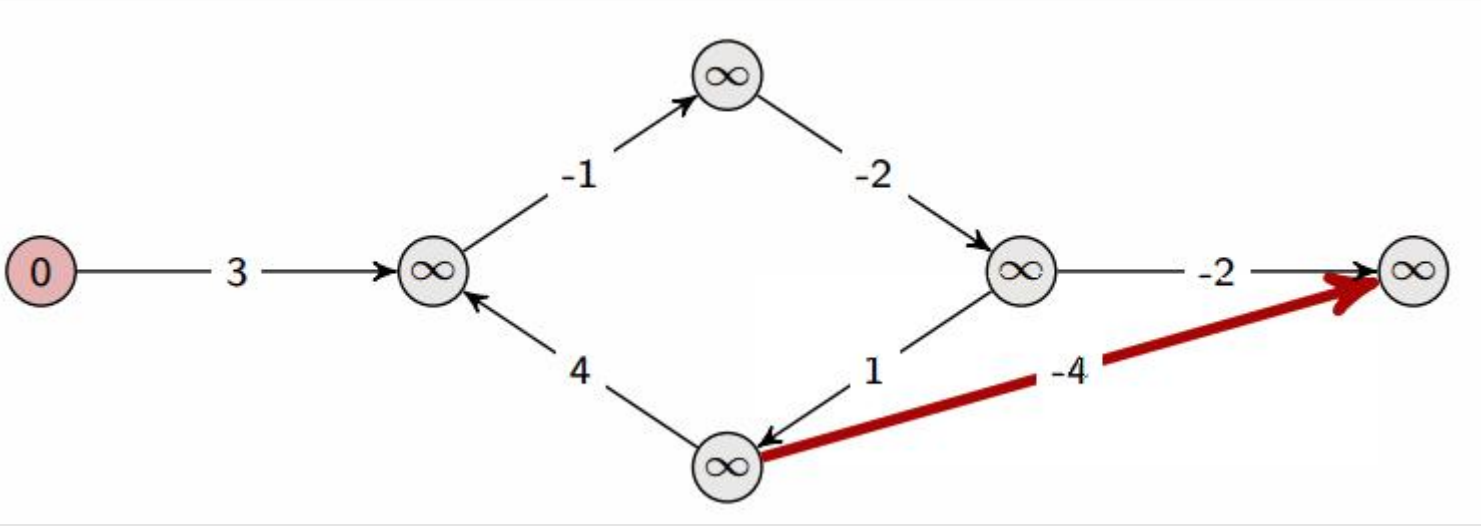
# Bellman Ford



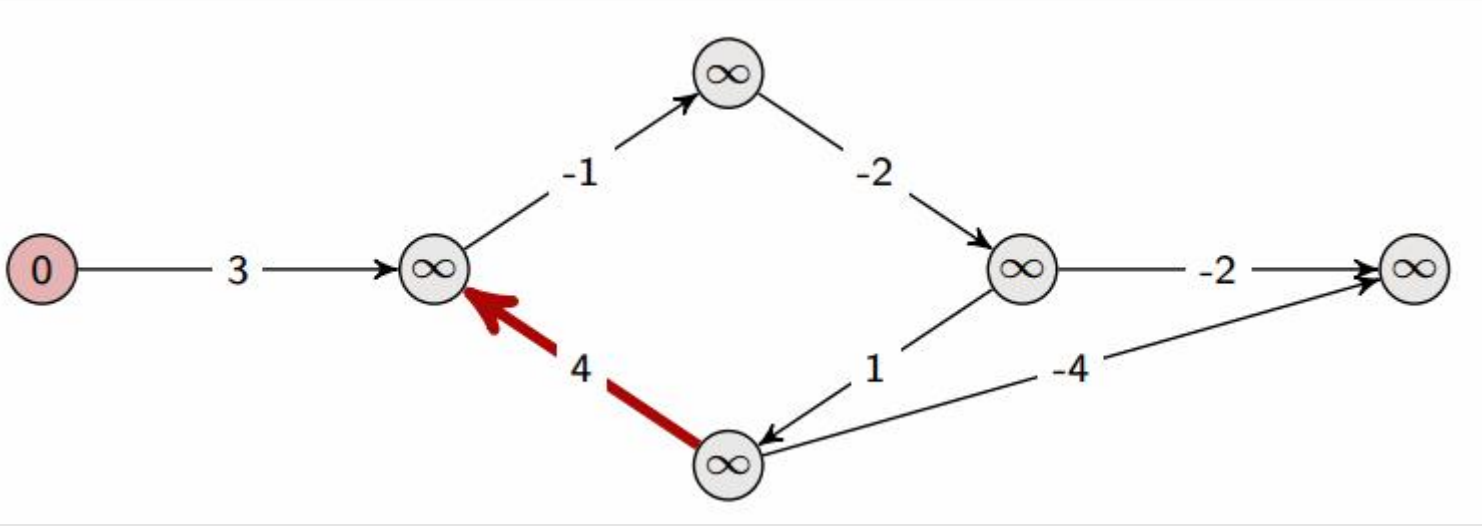
# Bellman Ford



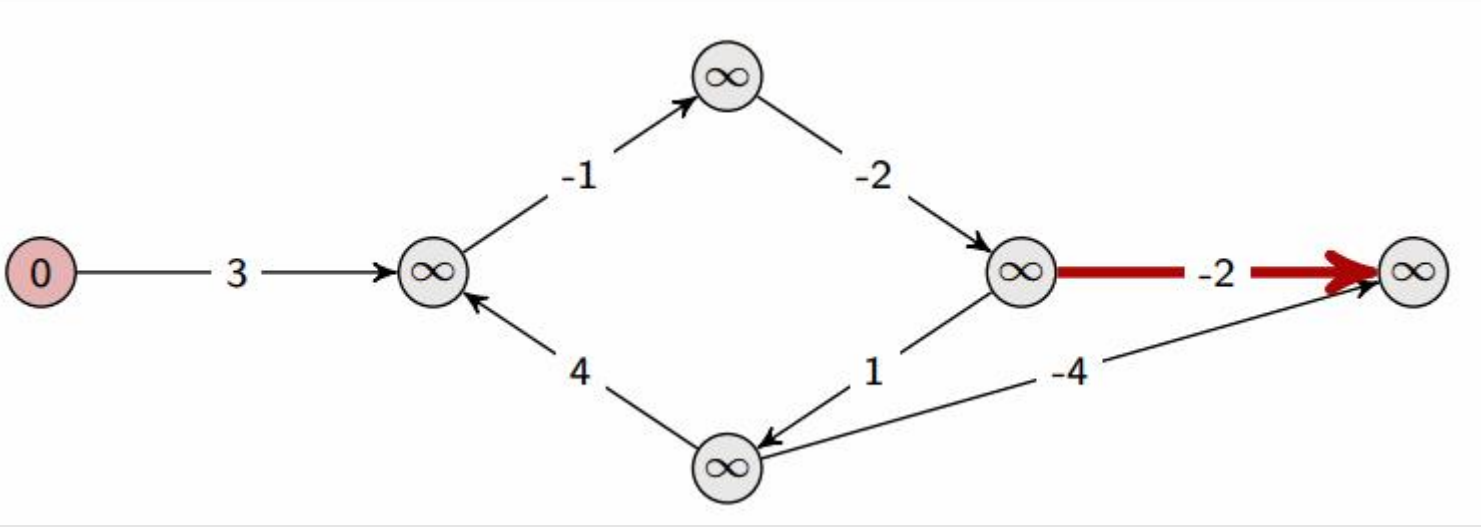
# Bellman Ford



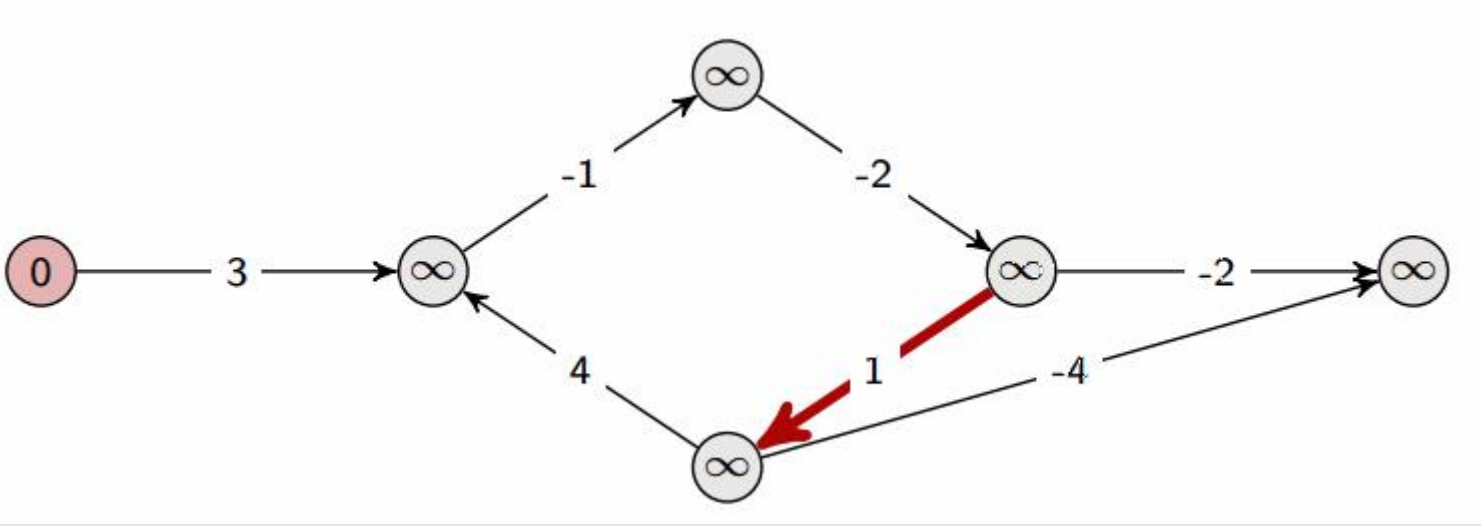
# Bellman Ford



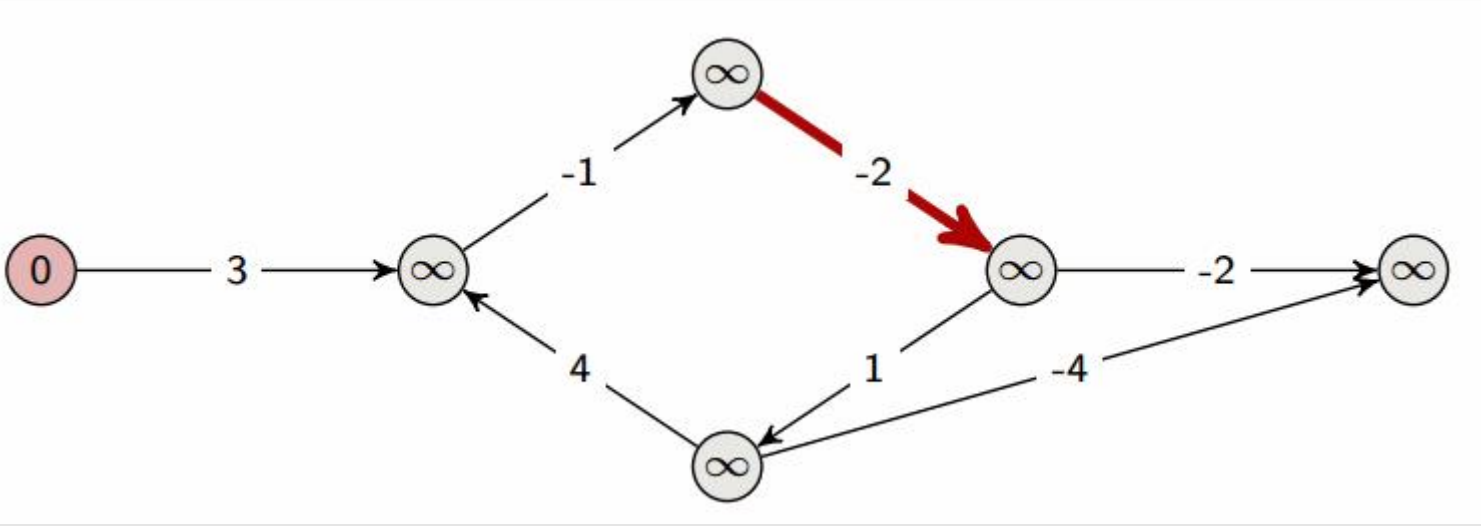
# Bellman Ford



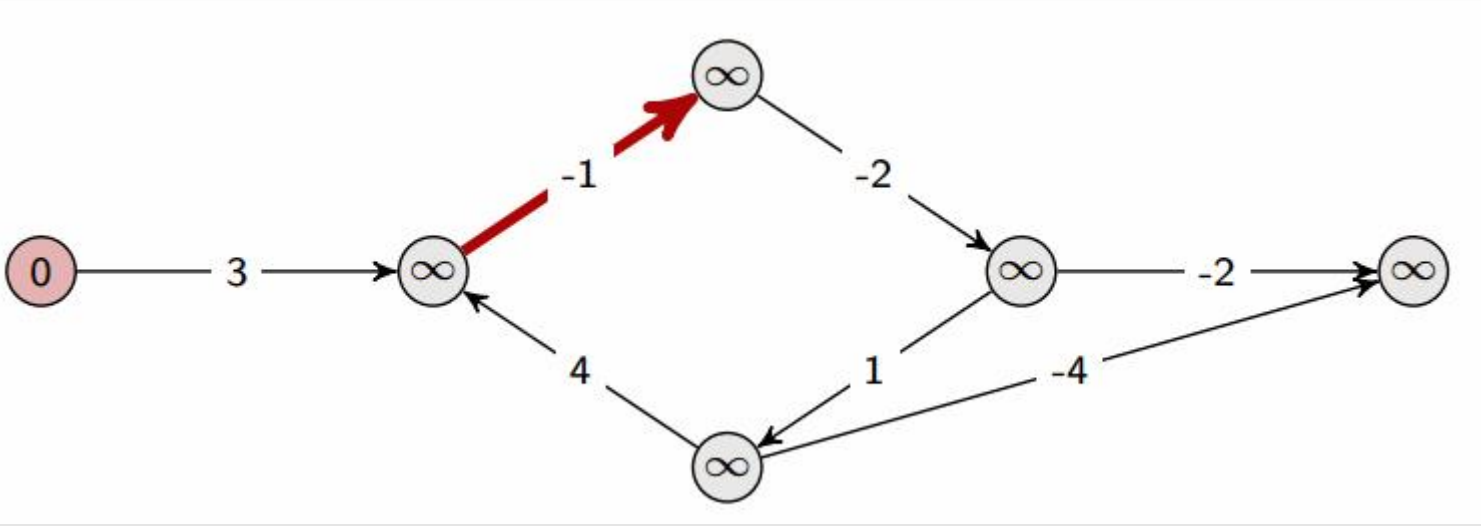
# Bellman Ford



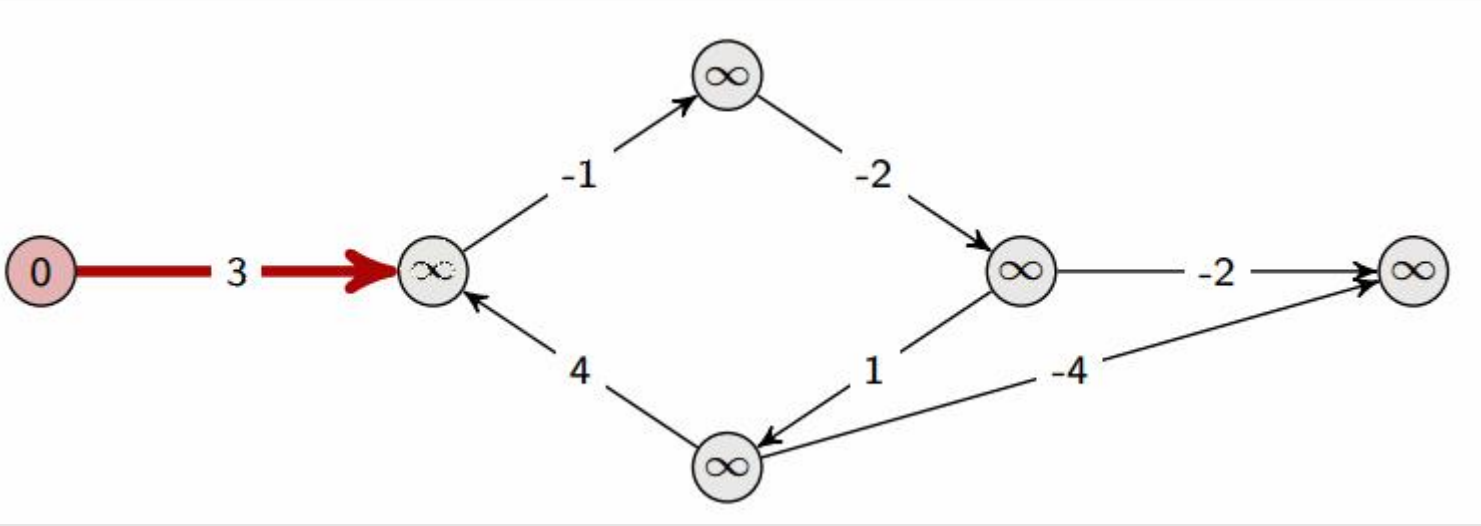
# Bellman Ford



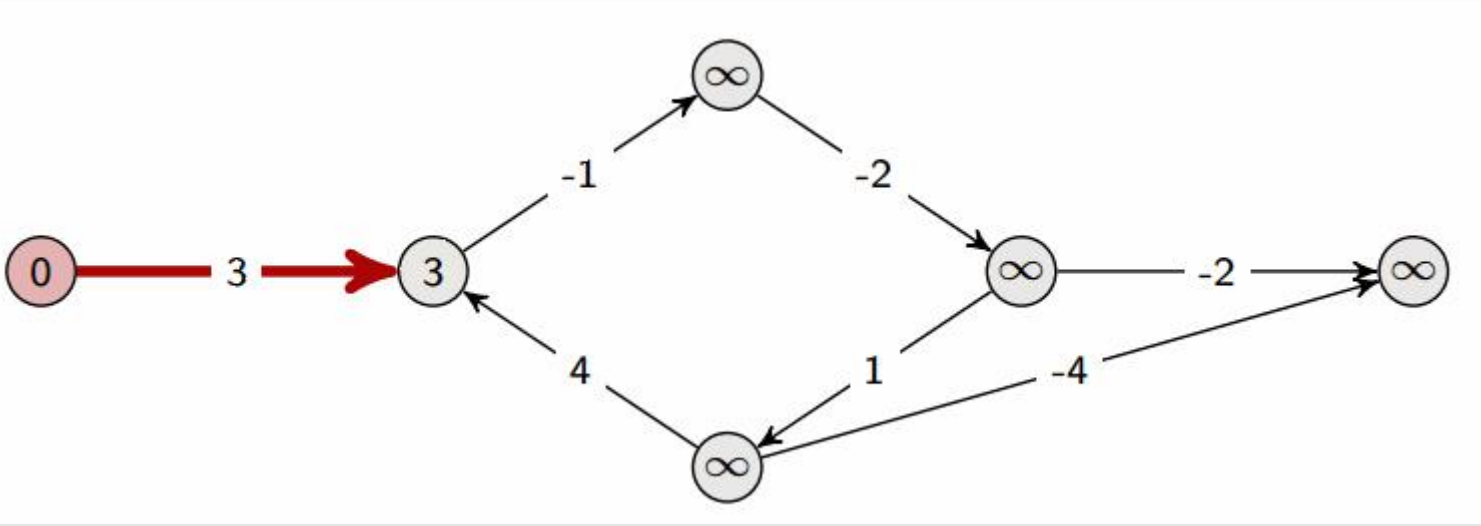
# Bellman Ford



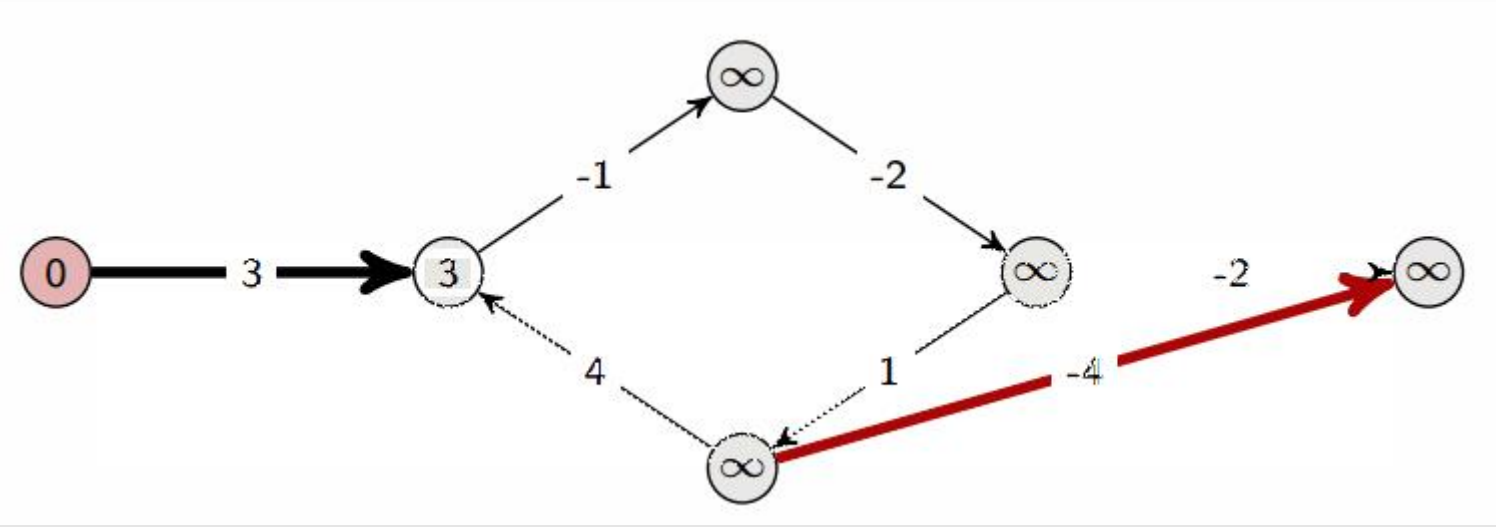
# Bellman Ford



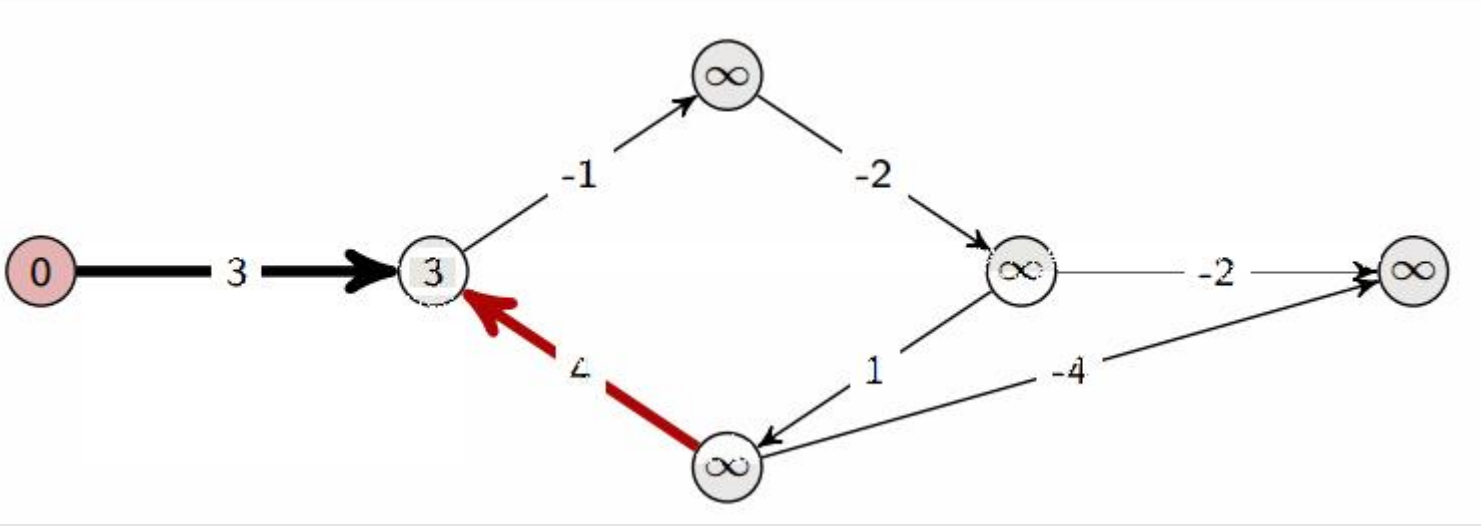
# Bellman Ford



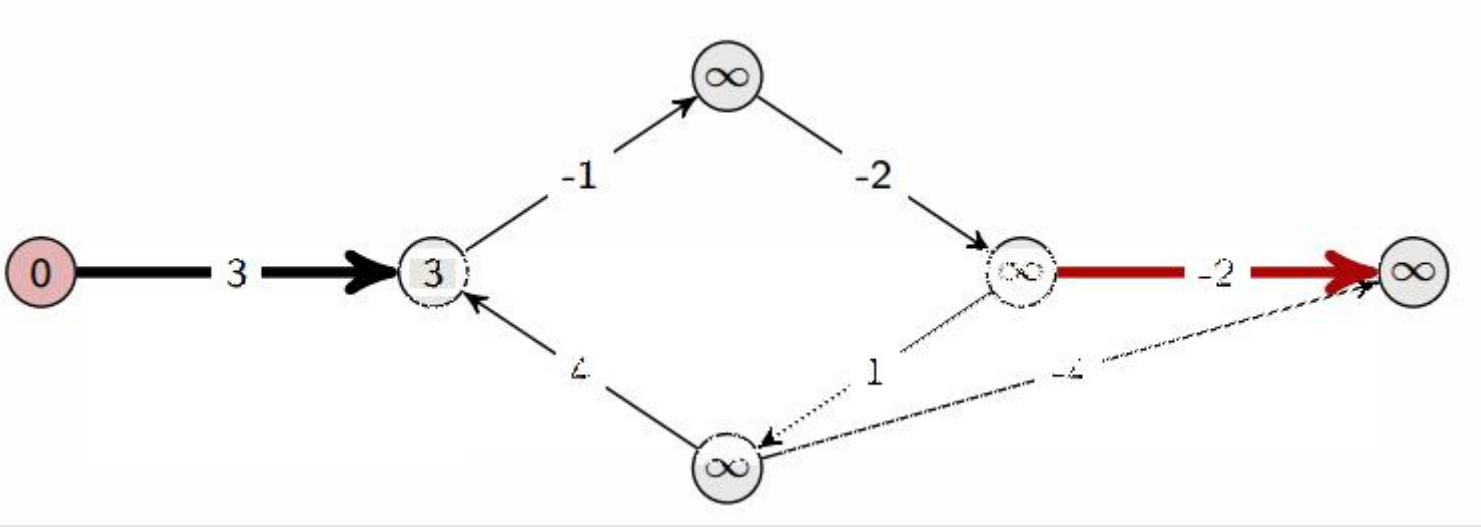
# Bellman Ford



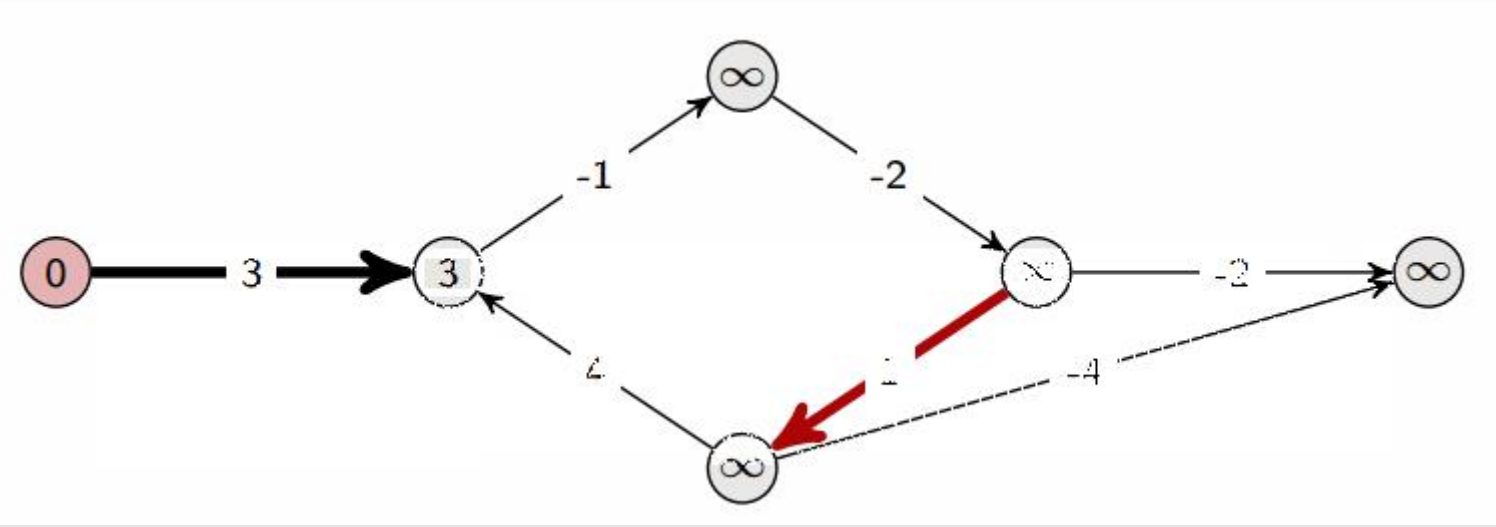
# Bellman Ford



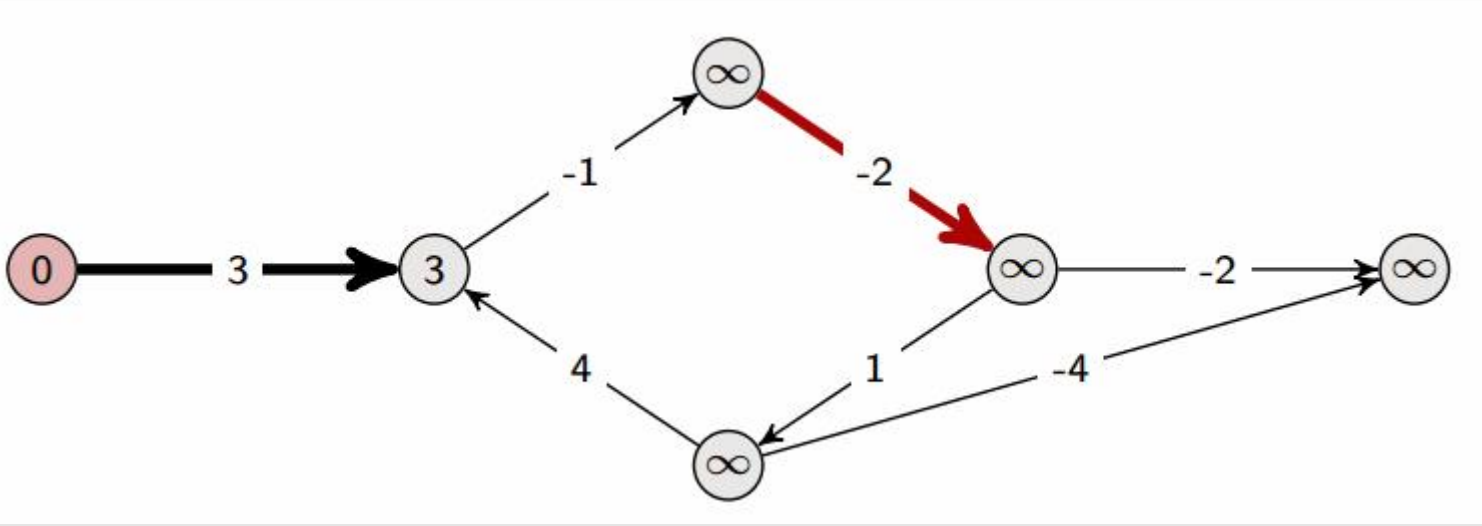
# Bellman Ford



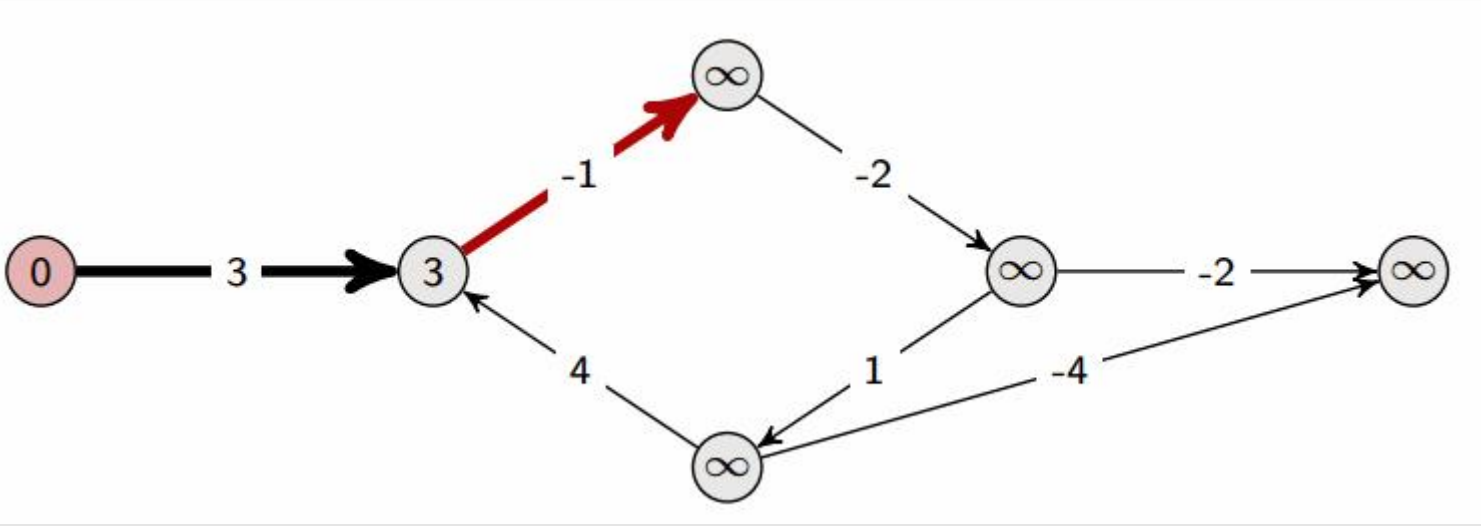
# Bellman Ford



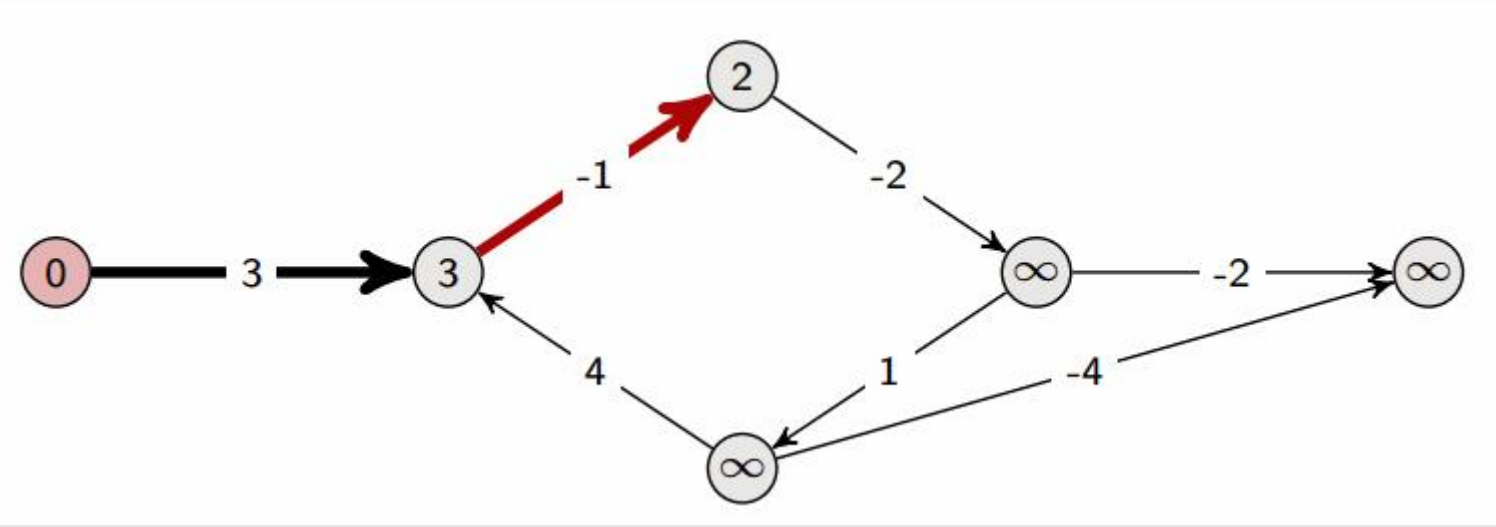
# Bellman Ford



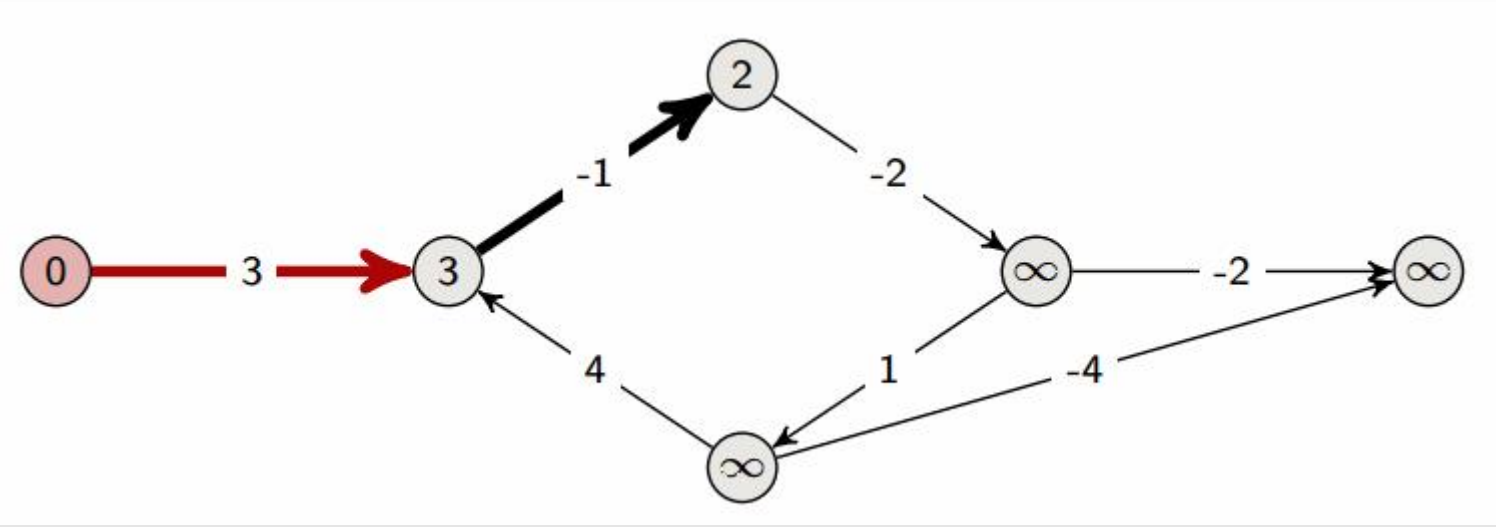
# Bellman Ford



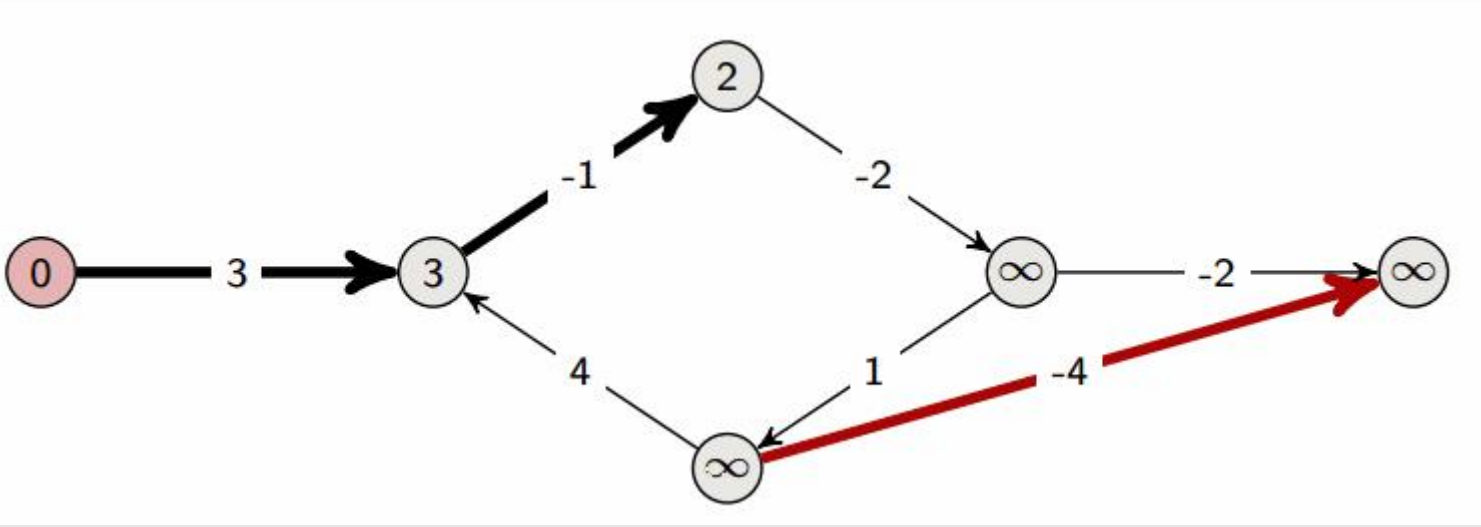
# Bellman Ford



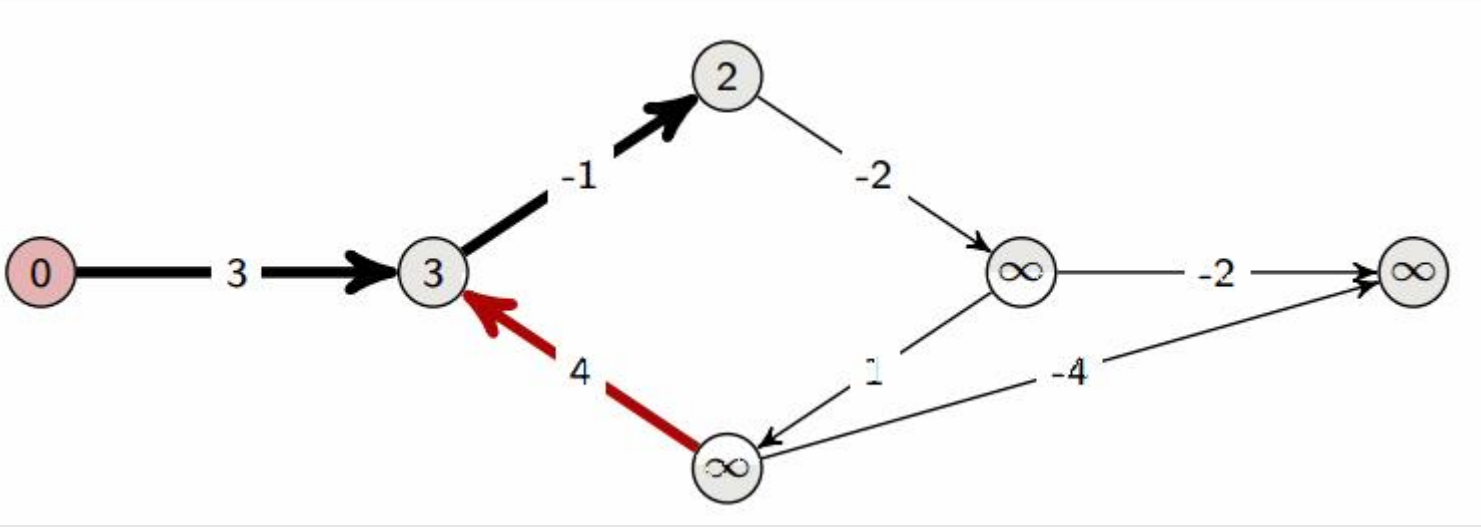
# Bellman Ford



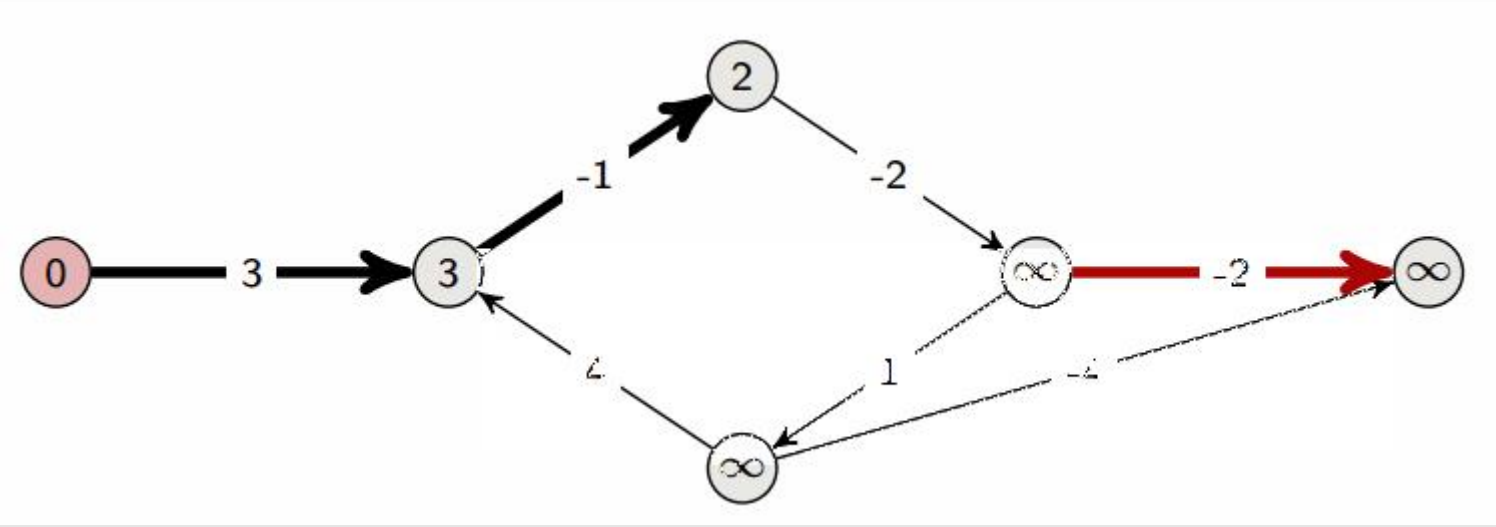
# Bellman Ford



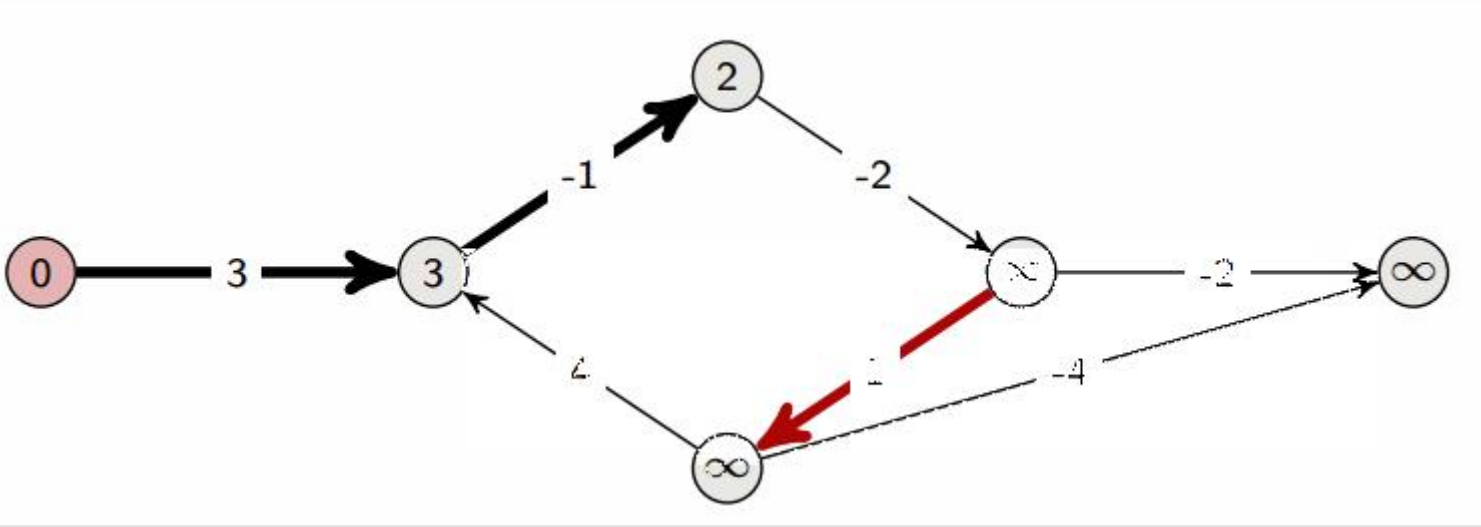
# Bellman Ford



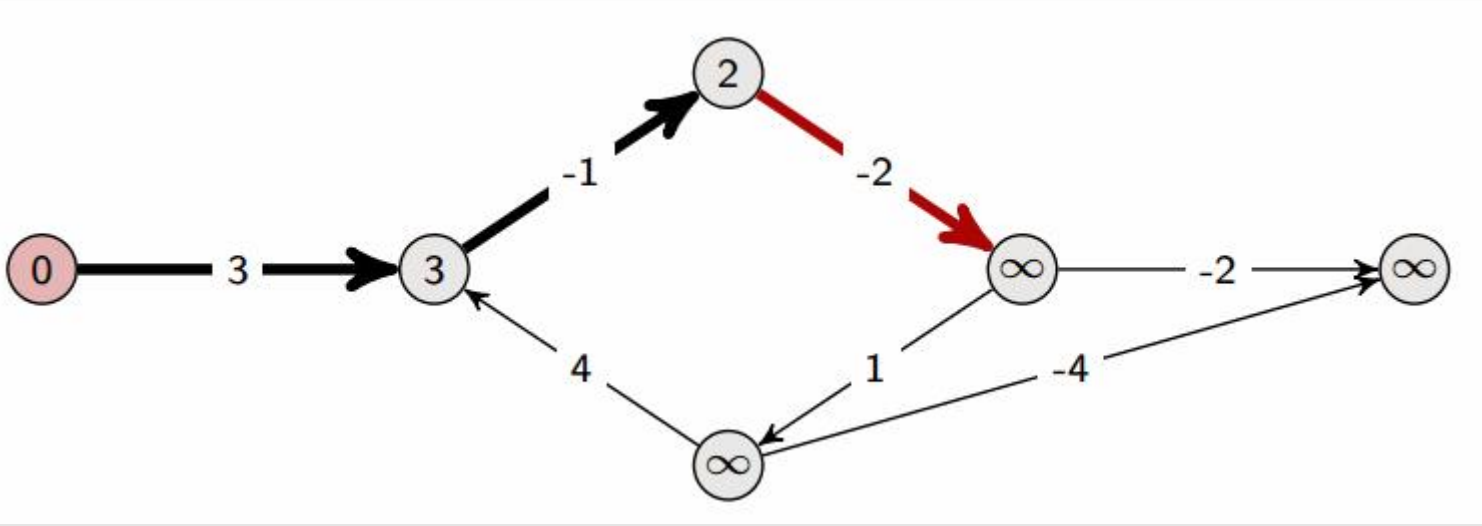
# Bellman Ford



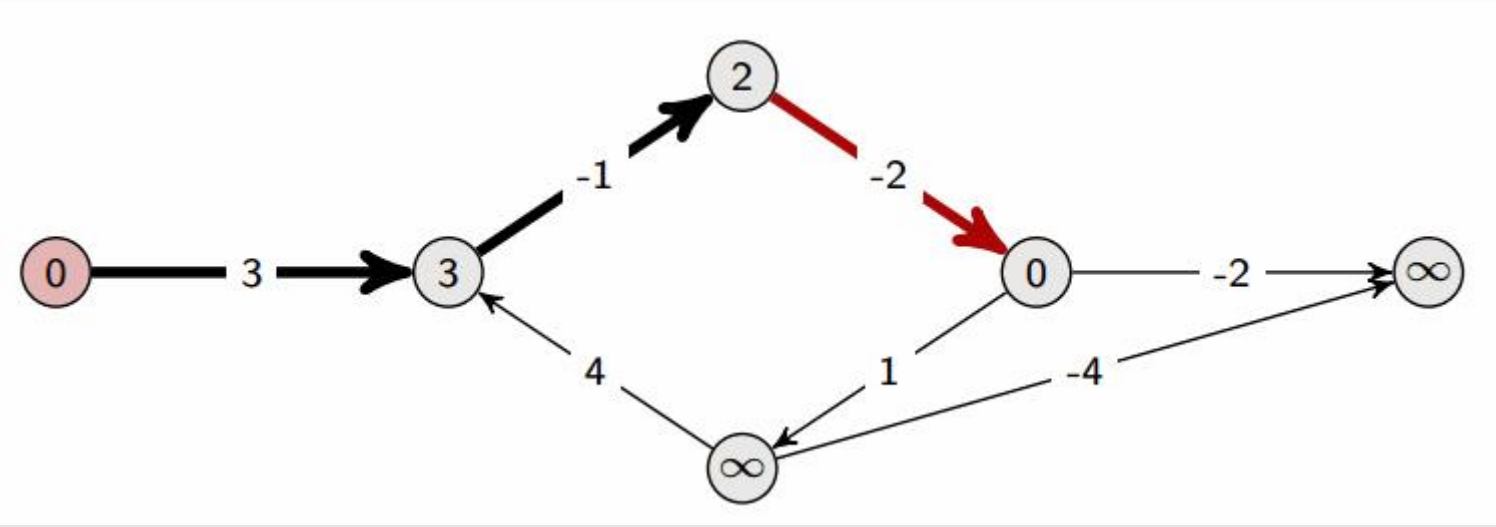
# Bellman Ford



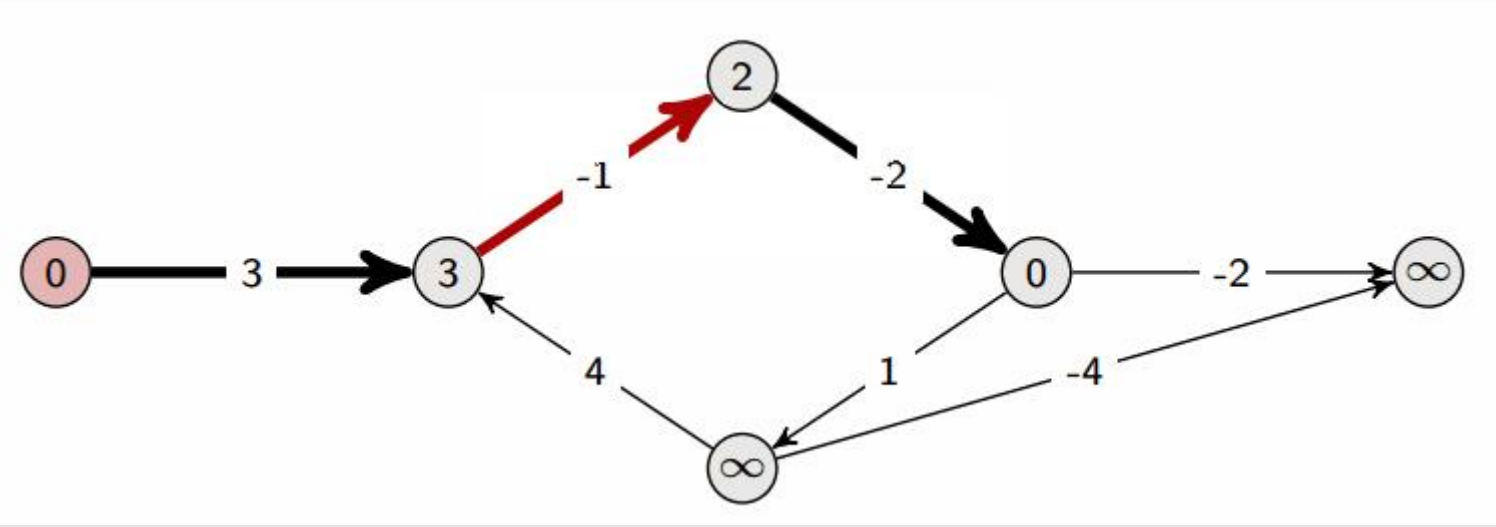
# Bellman Ford



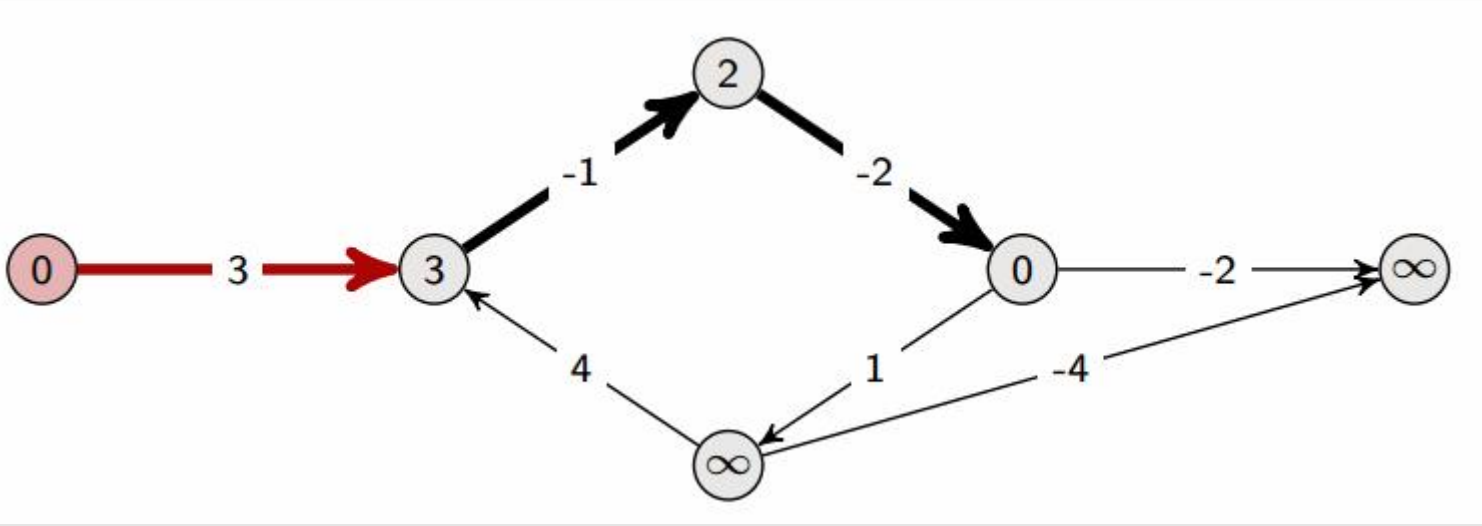
# Bellman Ford



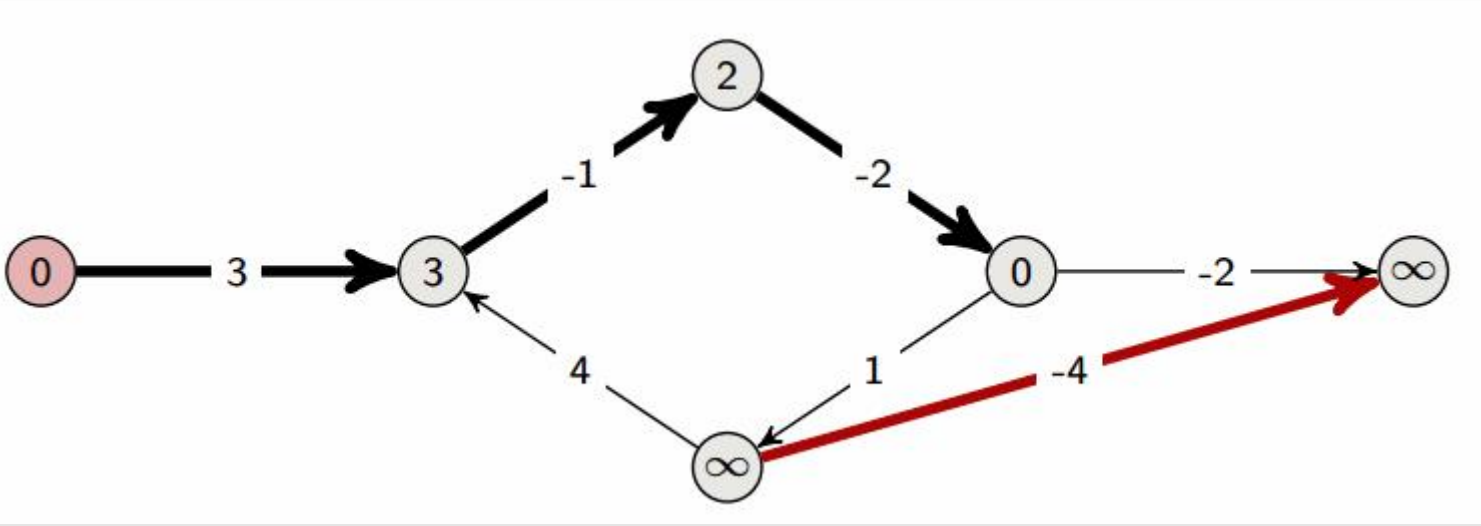
# Bellman Ford



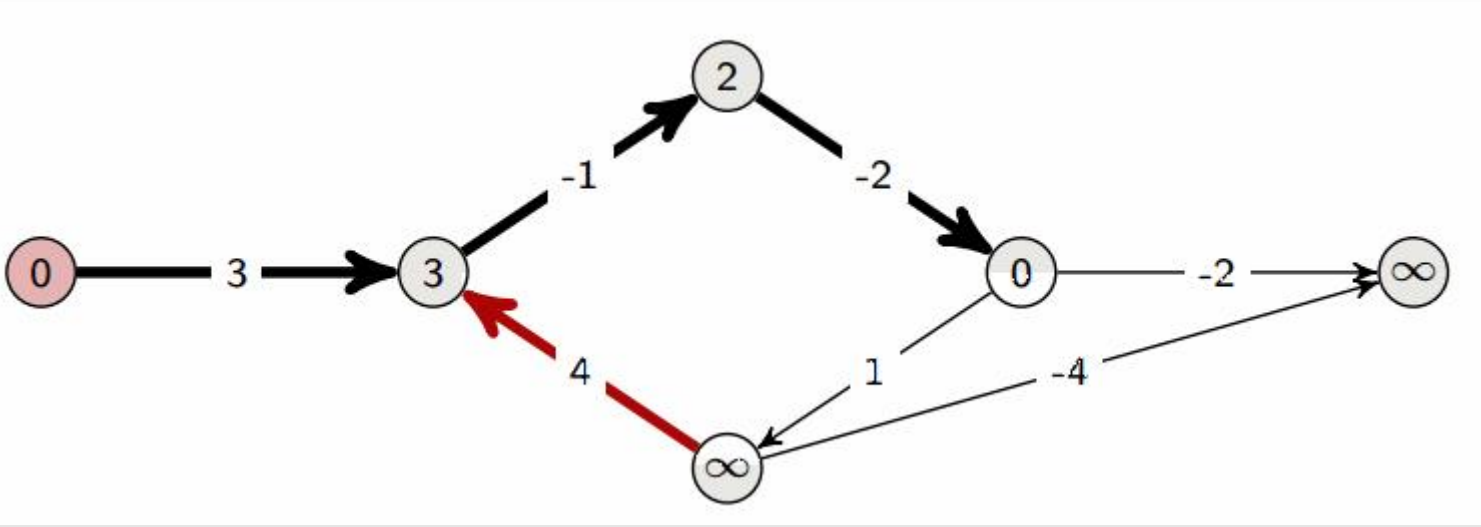
# Bellman Ford



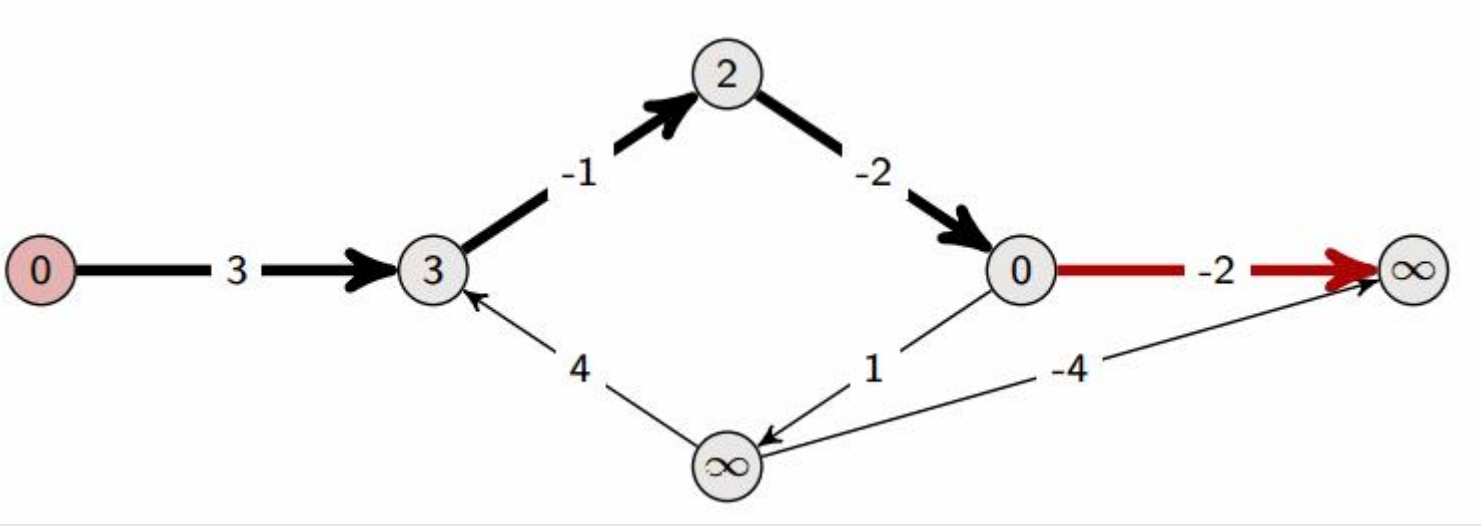
# Bellman Ford



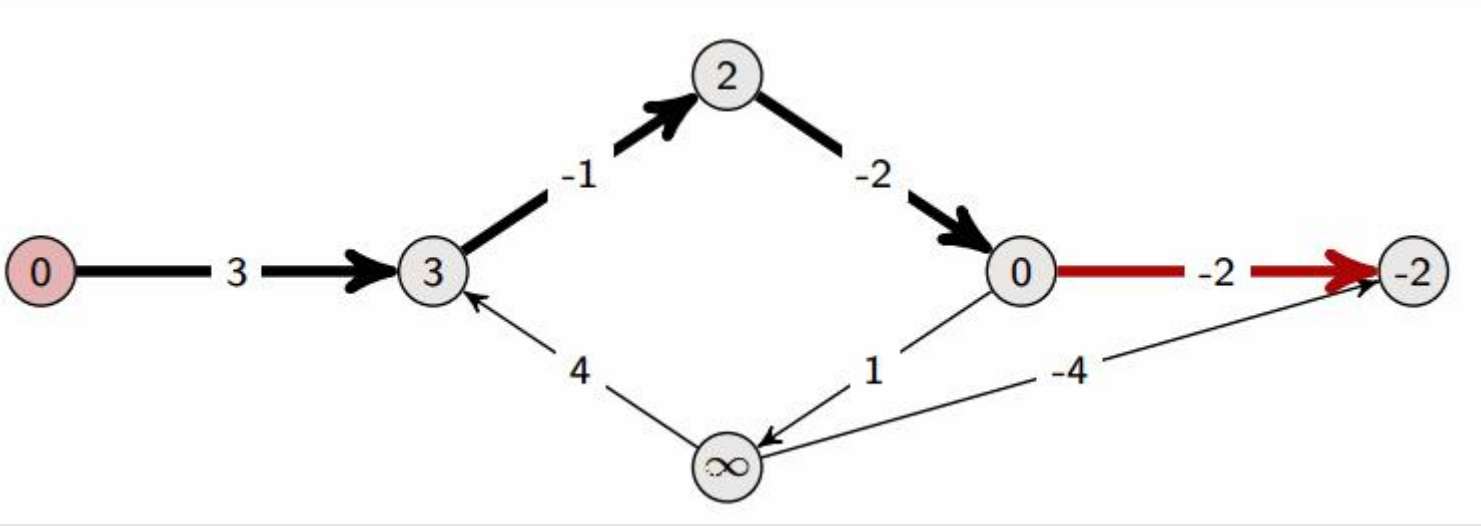
# Bellman Ford



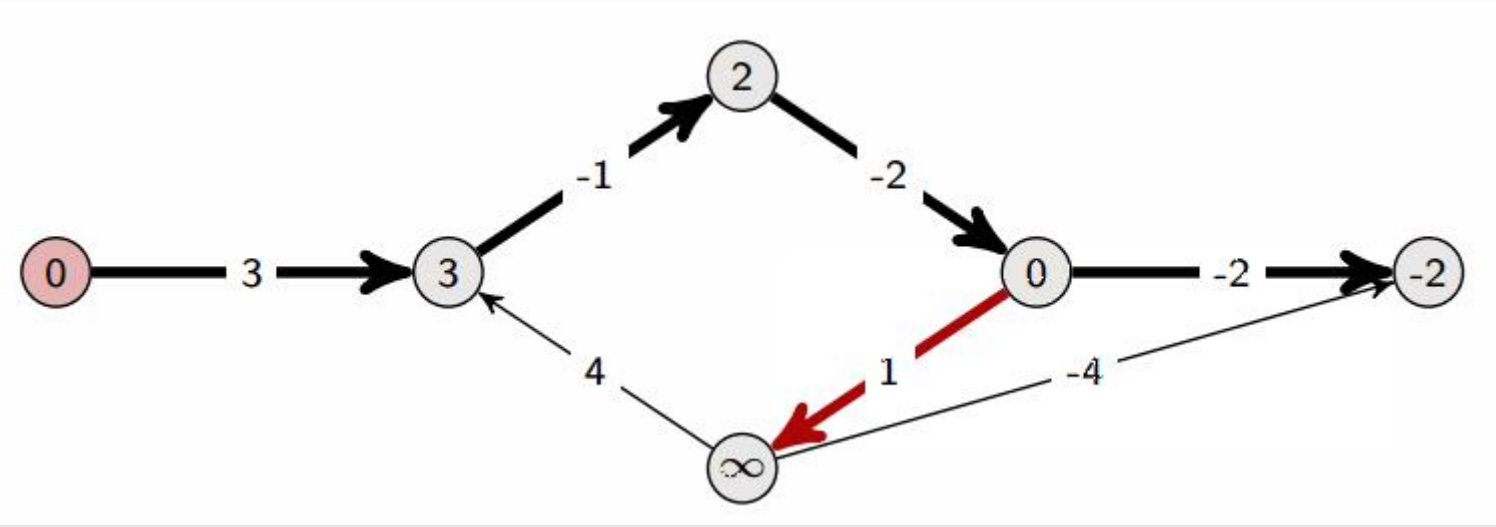
# Bellman Ford



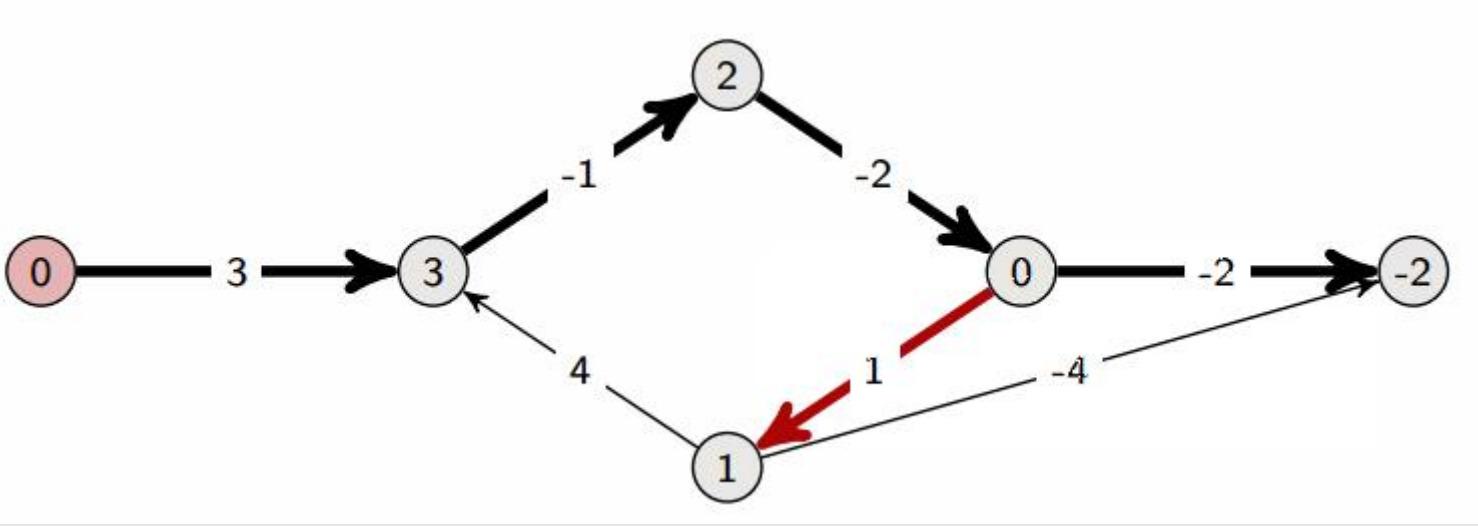
# Bellman Ford



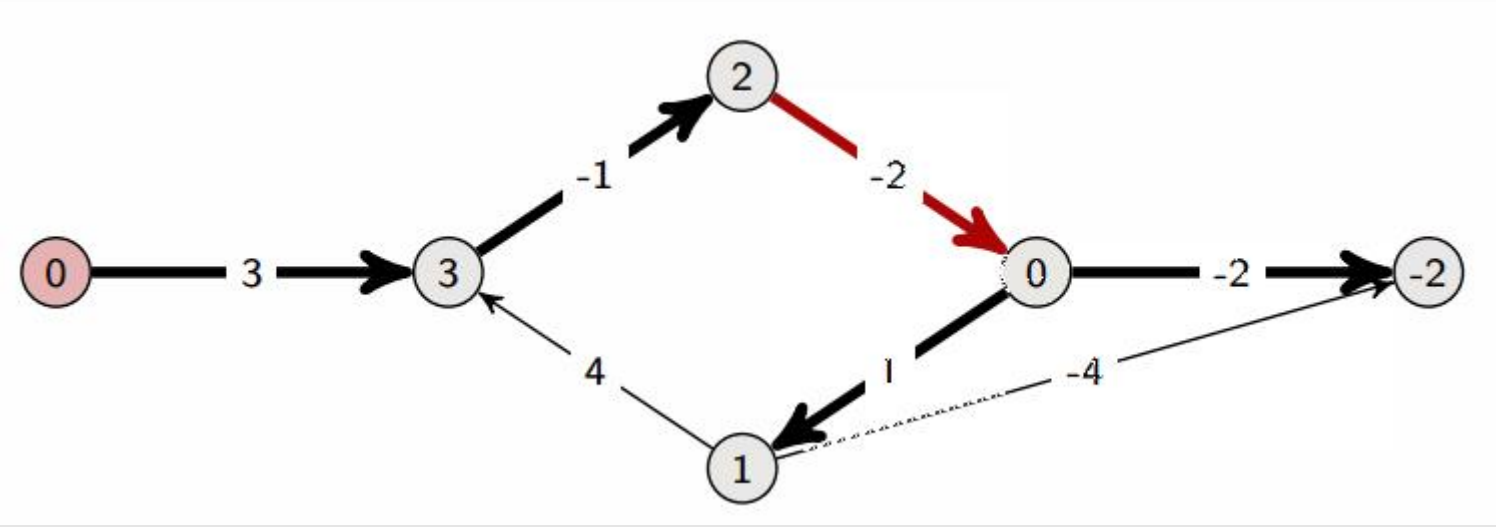
# Bellman Ford



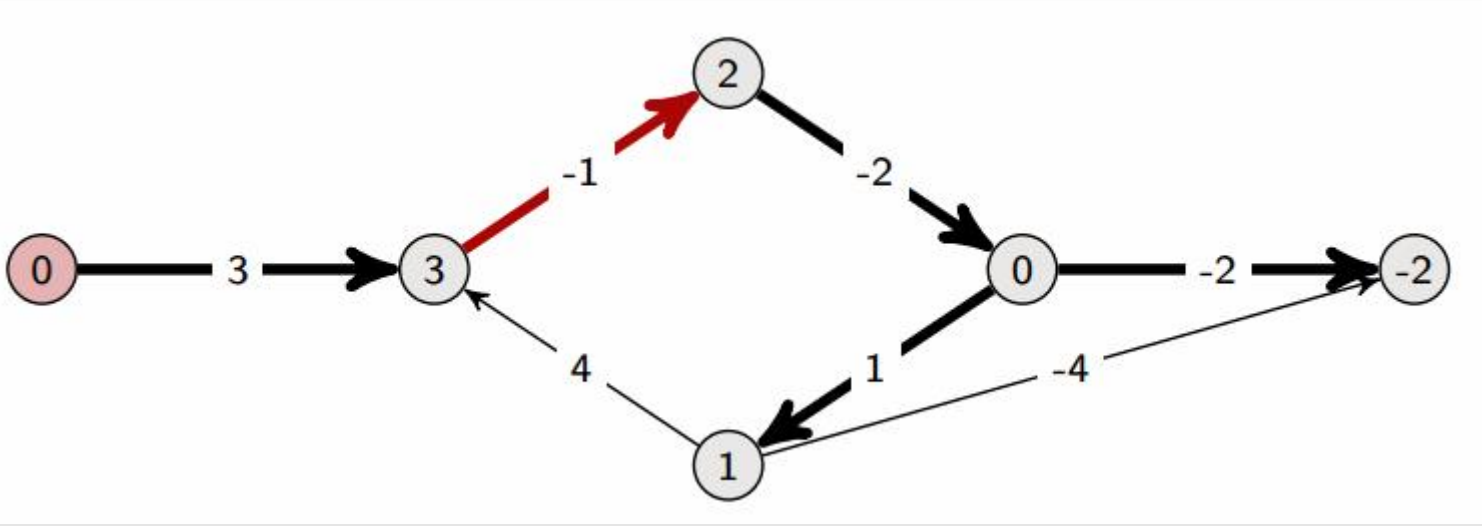
# Bellman Ford



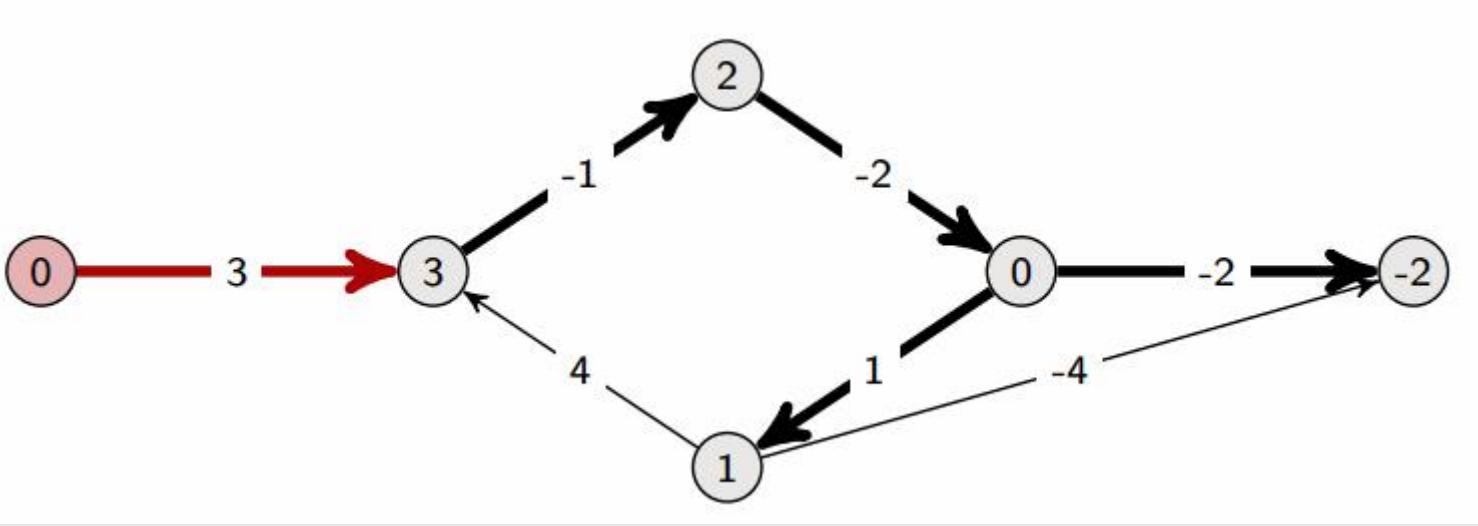
# Bellman Ford



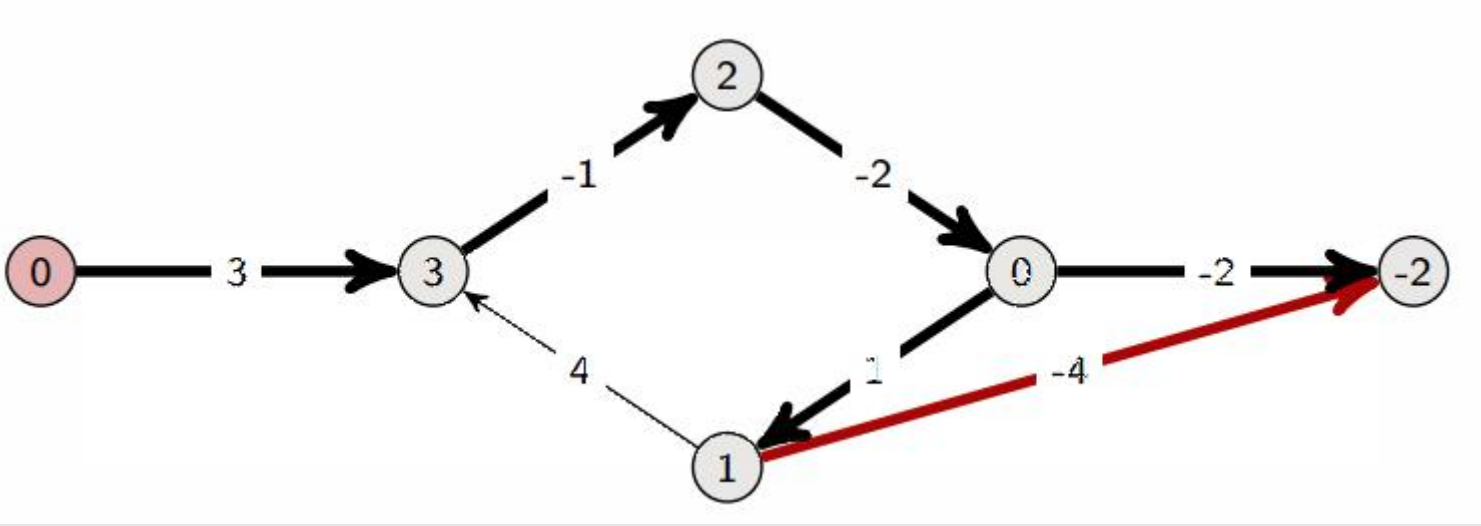
# Bellman Ford



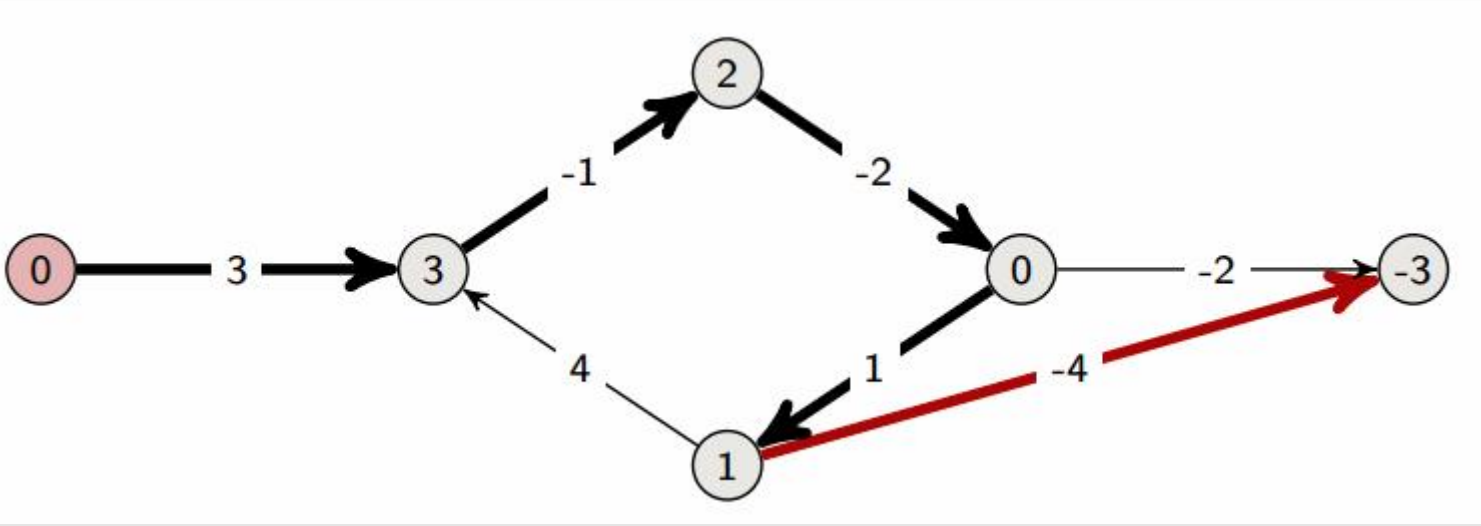
# Bellman Ford



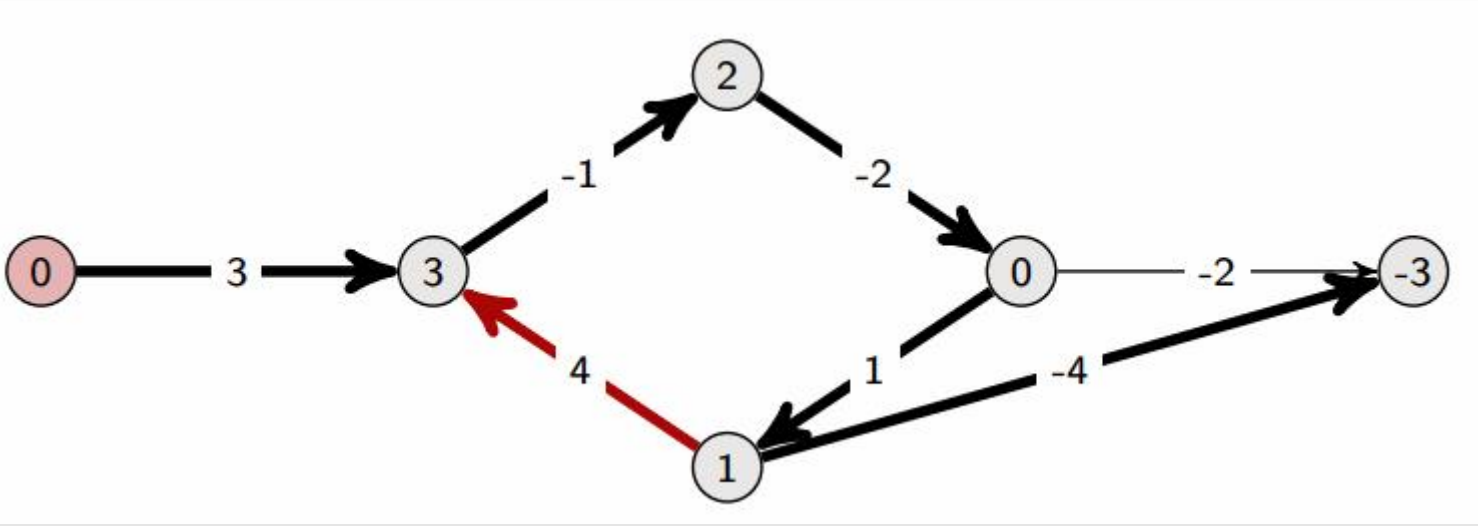
# Bellman Ford



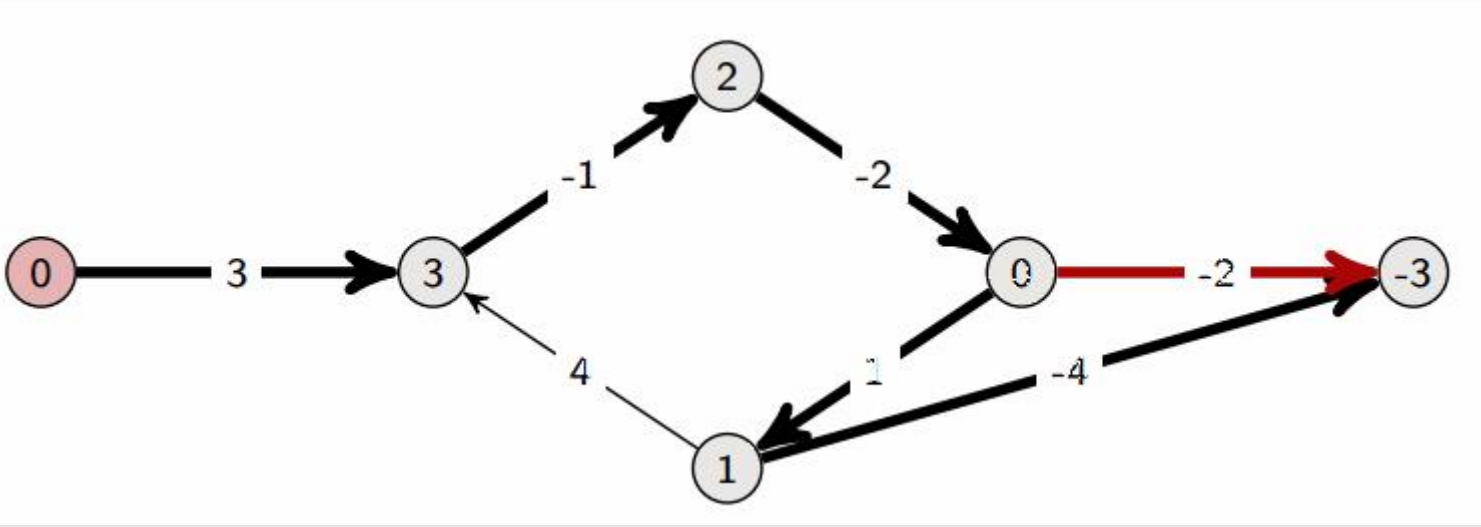
# Bellman Ford



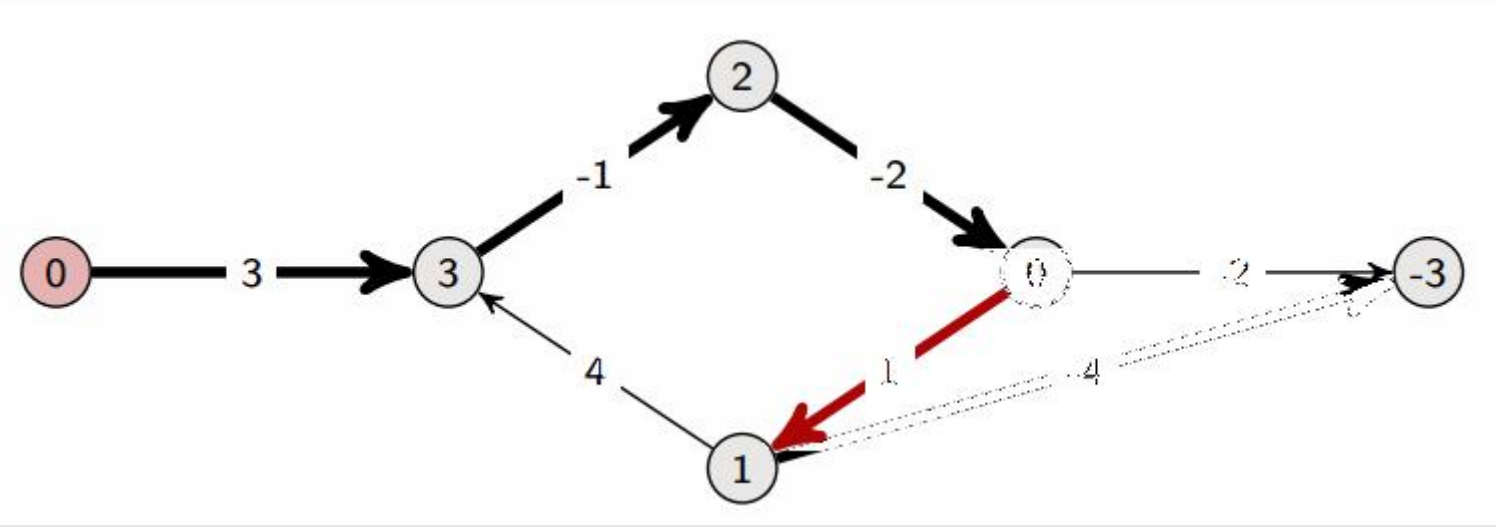
# Bellman Ford



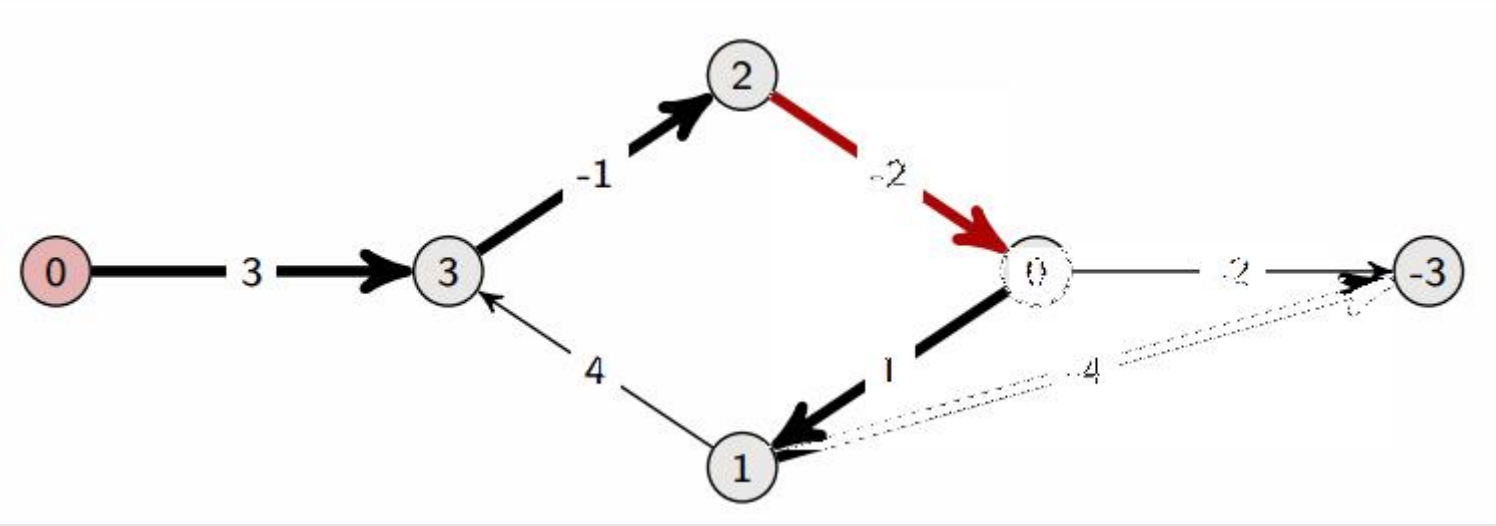
# Bellman Ford



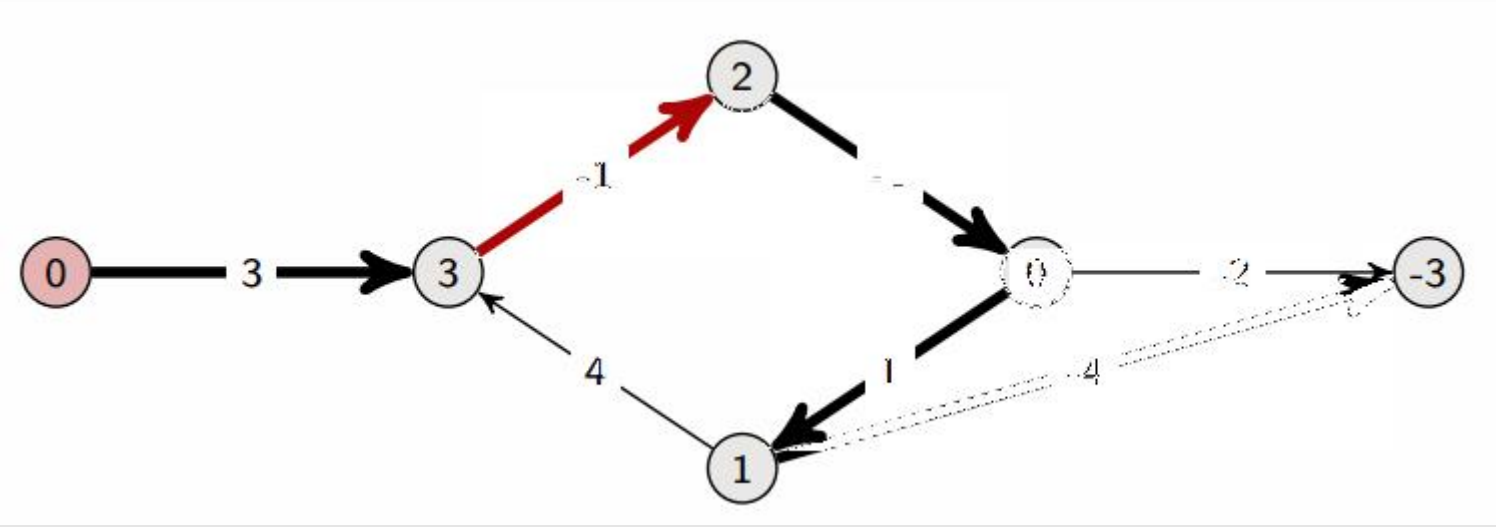
# Bellman Ford



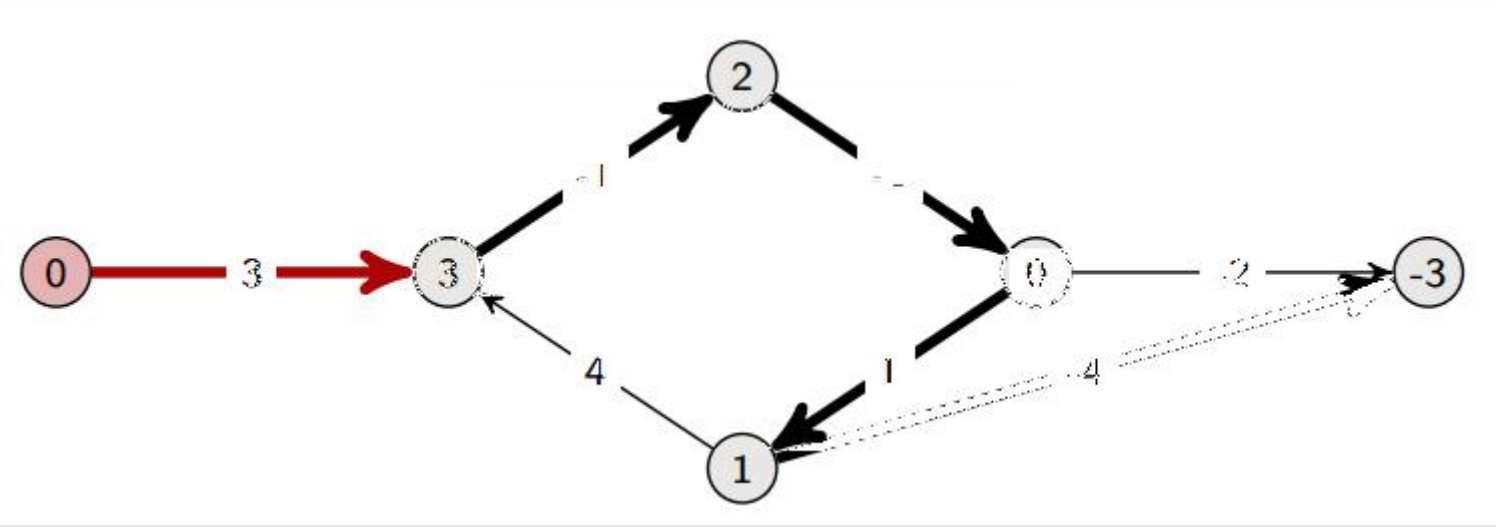
# Bellman Ford



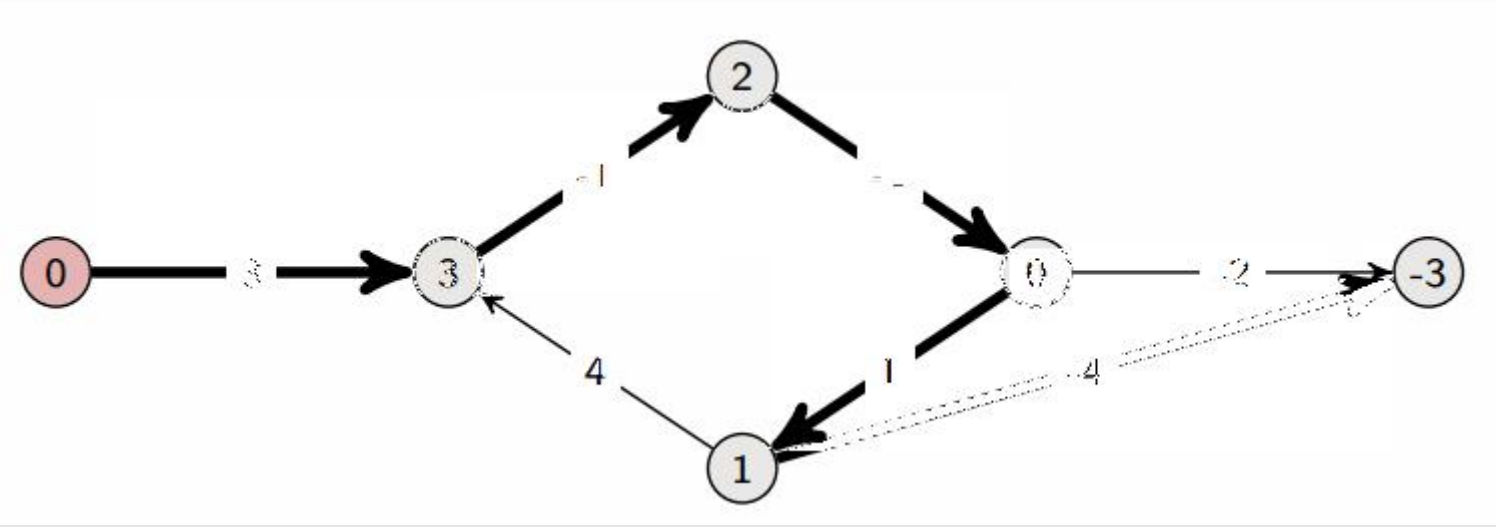
# Bellman Ford



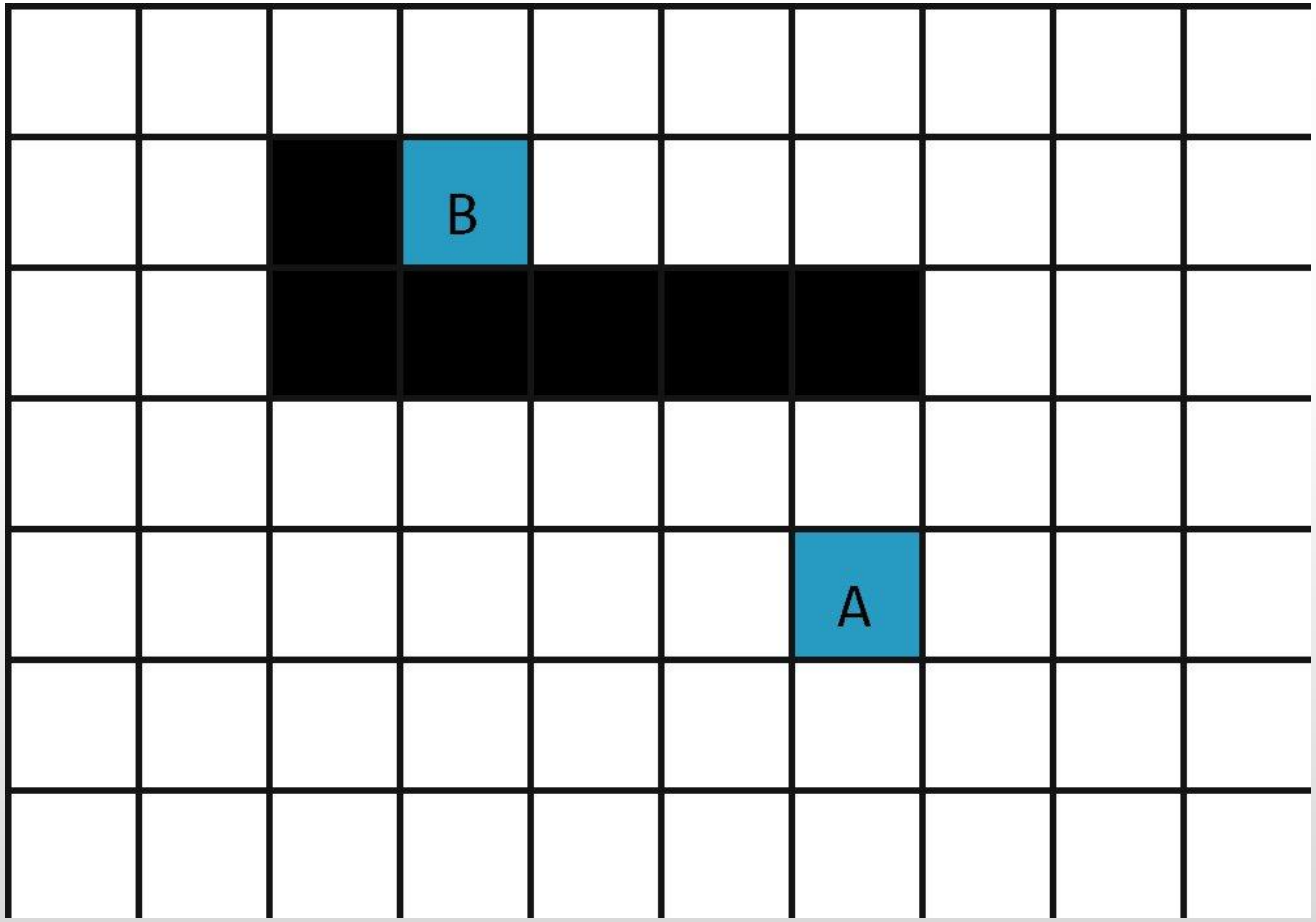
# Bellman Ford



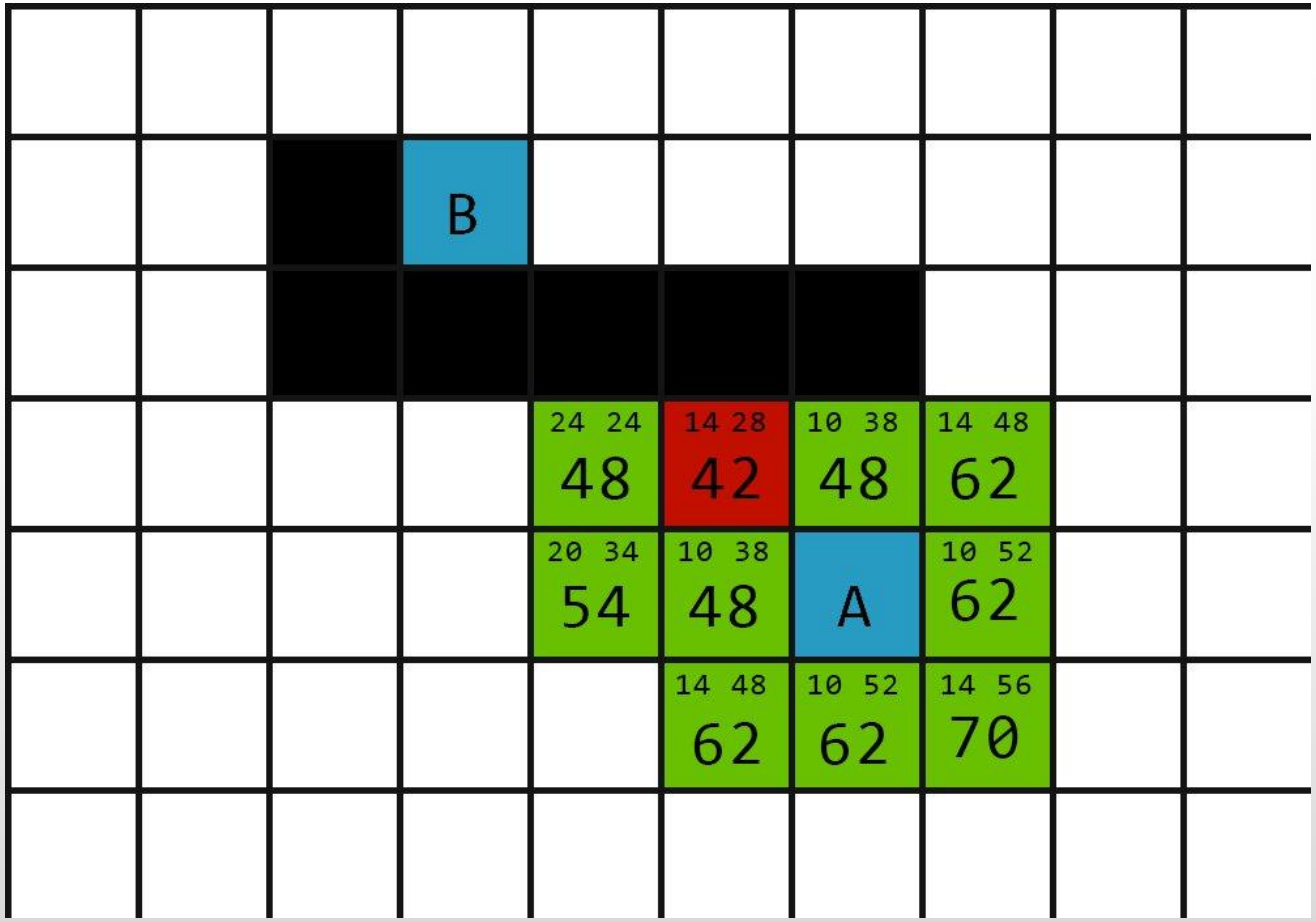
# Bellman Ford



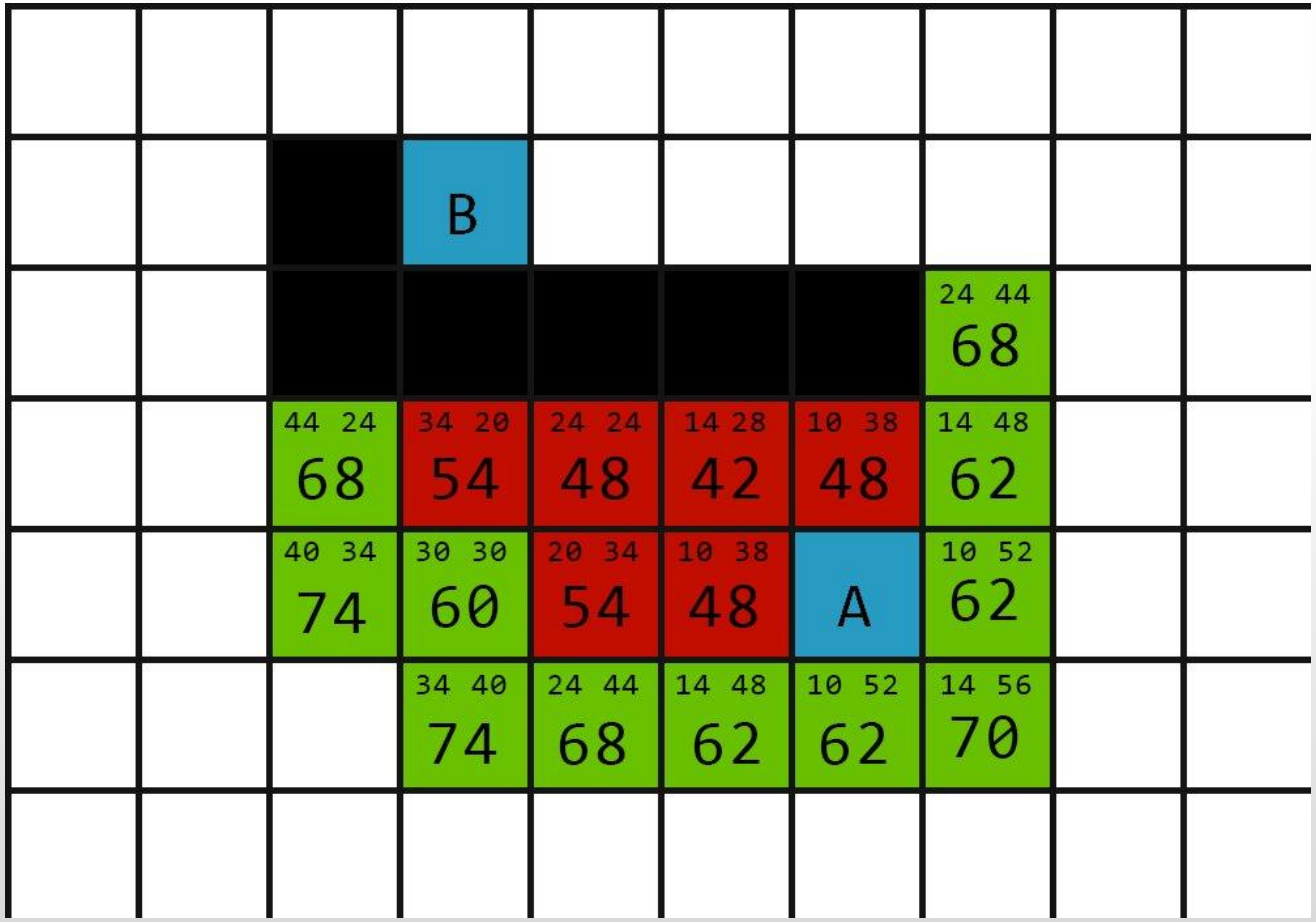
# A Star



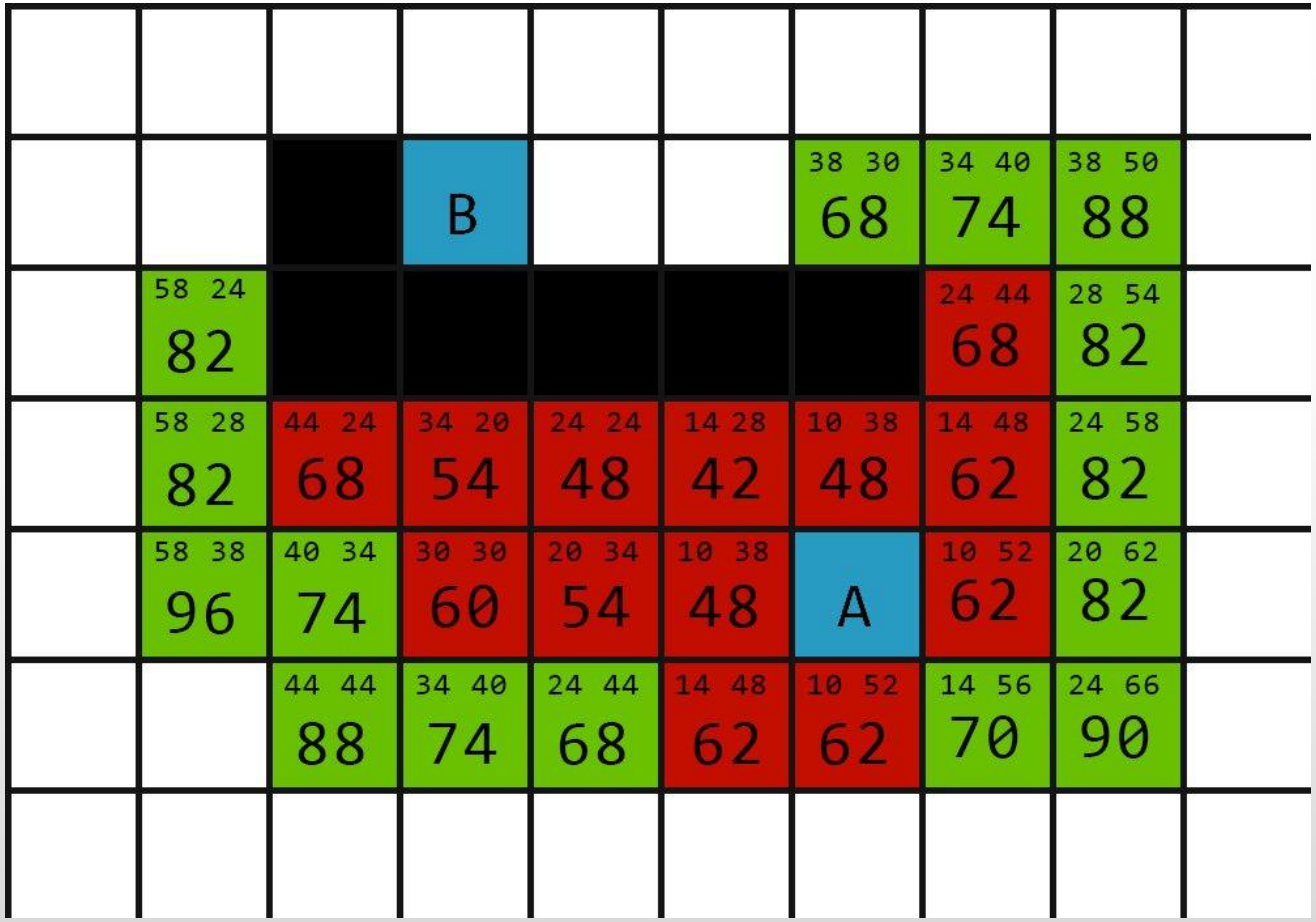
# A Star



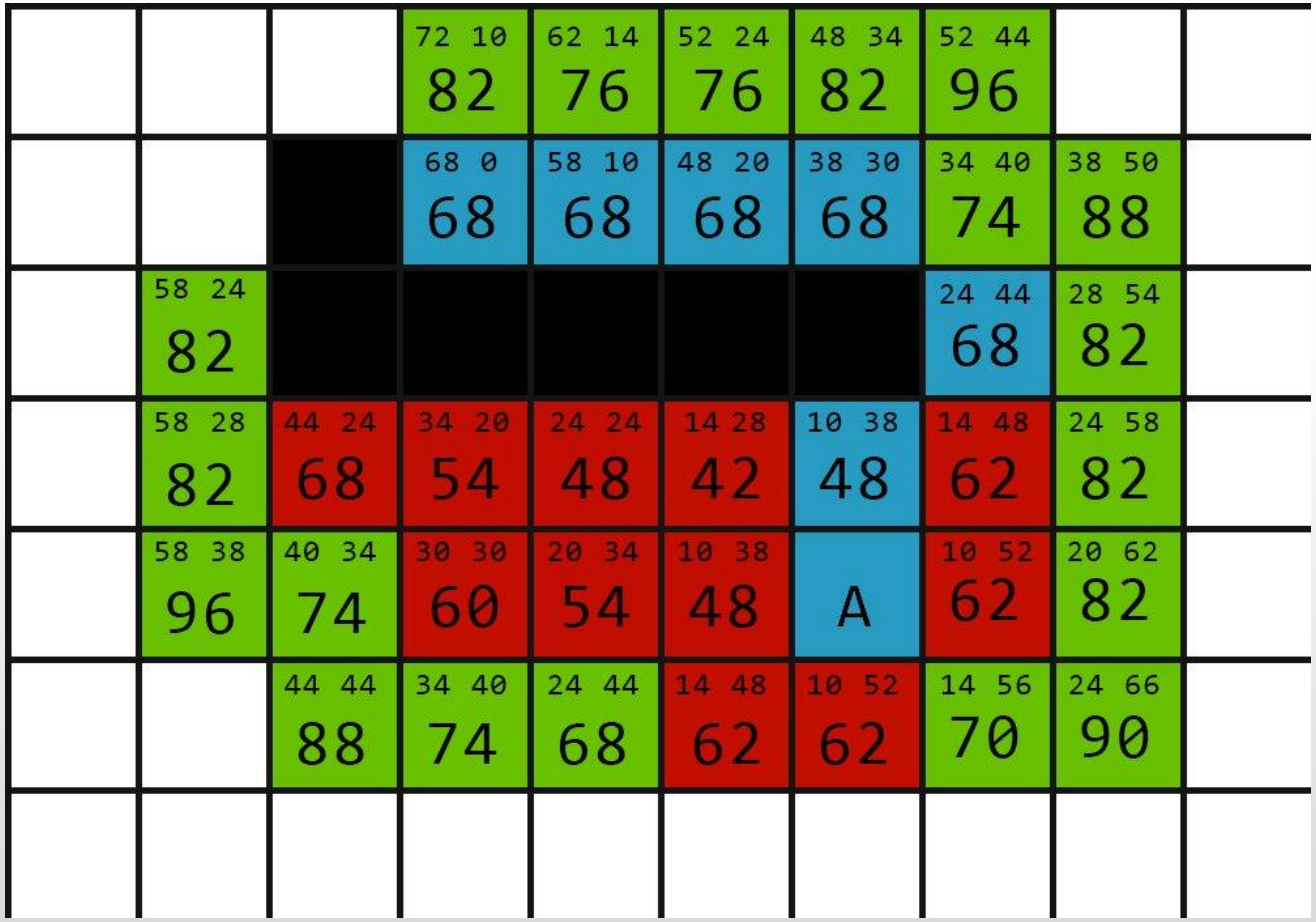
# A Star



# A Star



# A Star



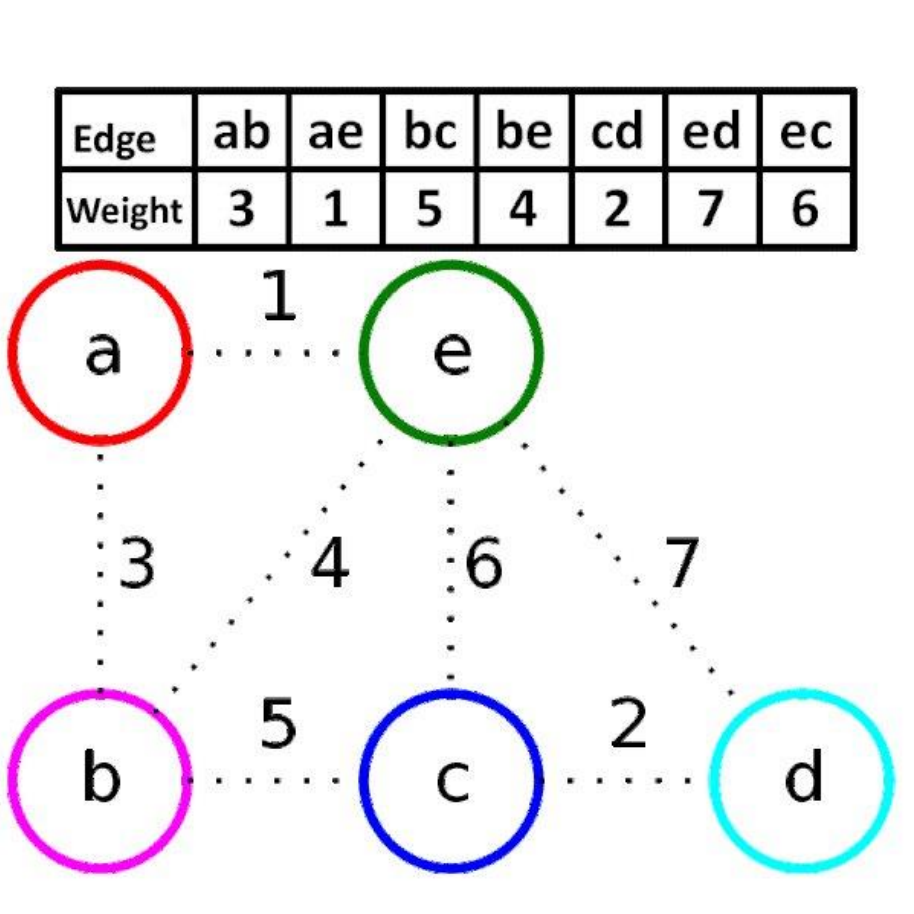
# A Star

# Minimum Yayılan Ağaç Algoritmaları

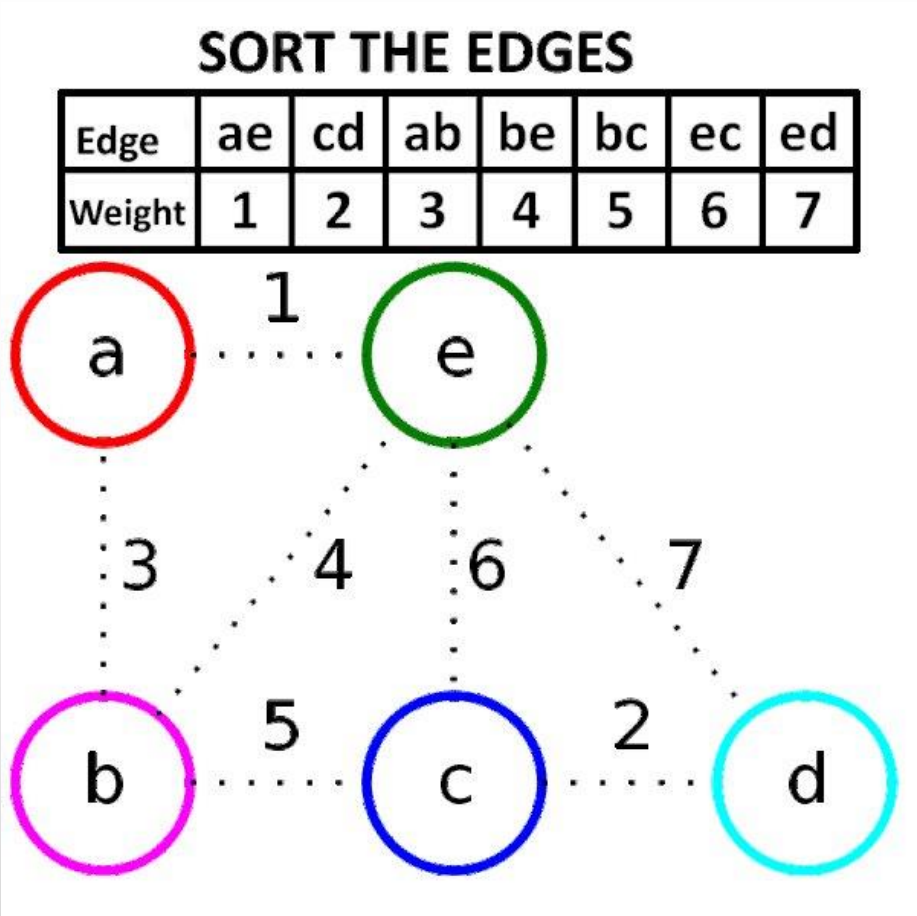
- Çizgedeki tüm düğümleri birbirine bağlayan ve toplam kenar ağırlığının en az olduğu alt ağaçtır.
- *Kruskal*, kenarları ağırlıklarına göre sıralar ve döngü oluşturmayan kenarları seçerek ağacı oluşturur.
- *Prim*, başlangıç düğümünden başlayarak, her adımda en düşük ağırlıklı kenarı seçerek ağacı büyütür.

# Kruskal

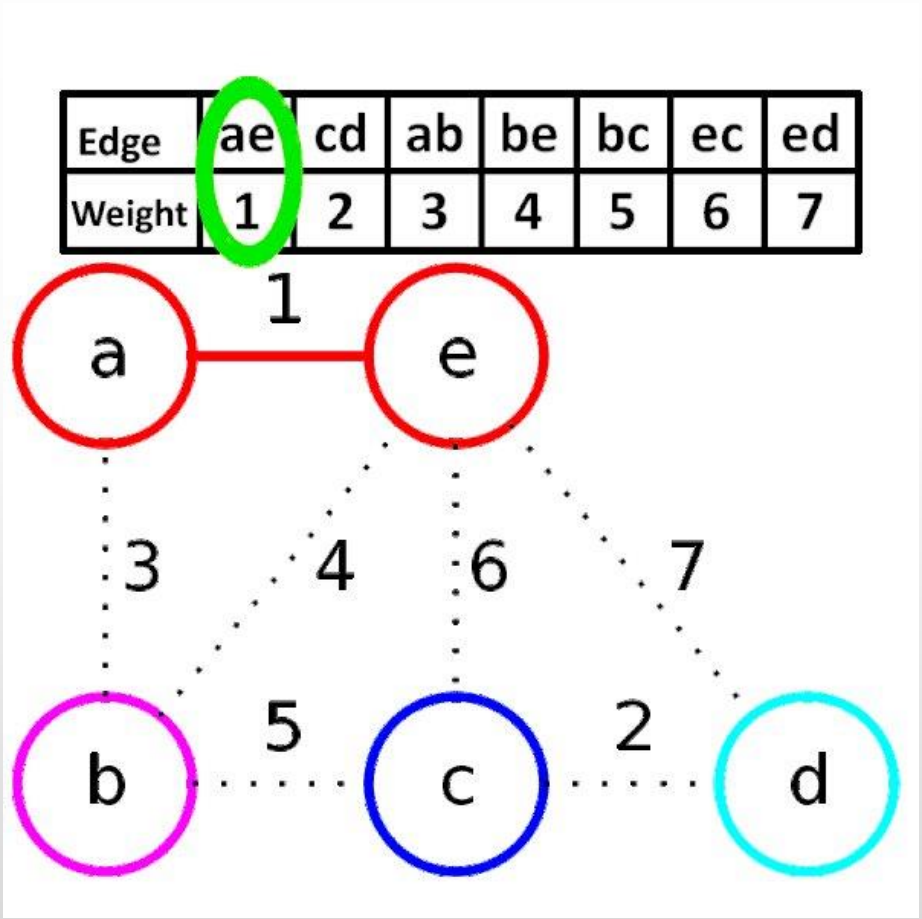
| Edge | ab | ae | bc | be | cd | ed | ec |
|---|---|---|---|---|---|---|---|
| Weight | 3 | 1 | 5 | 4 | 2 | 7 | 6 |



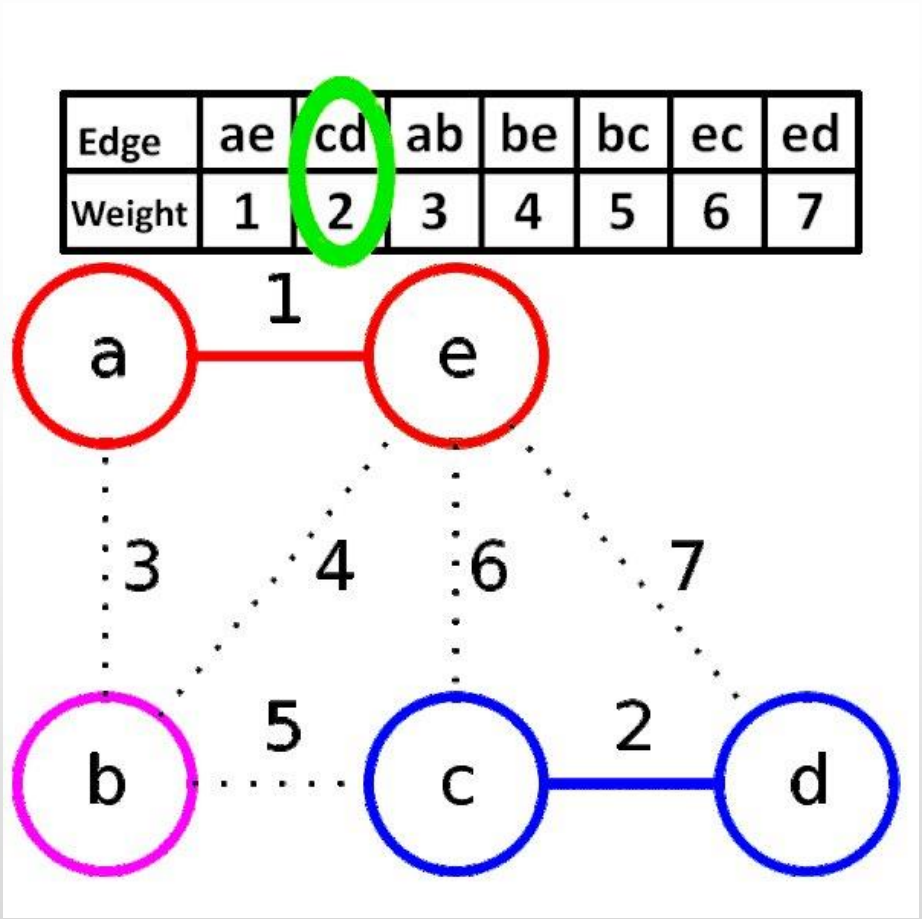
# Kruskal



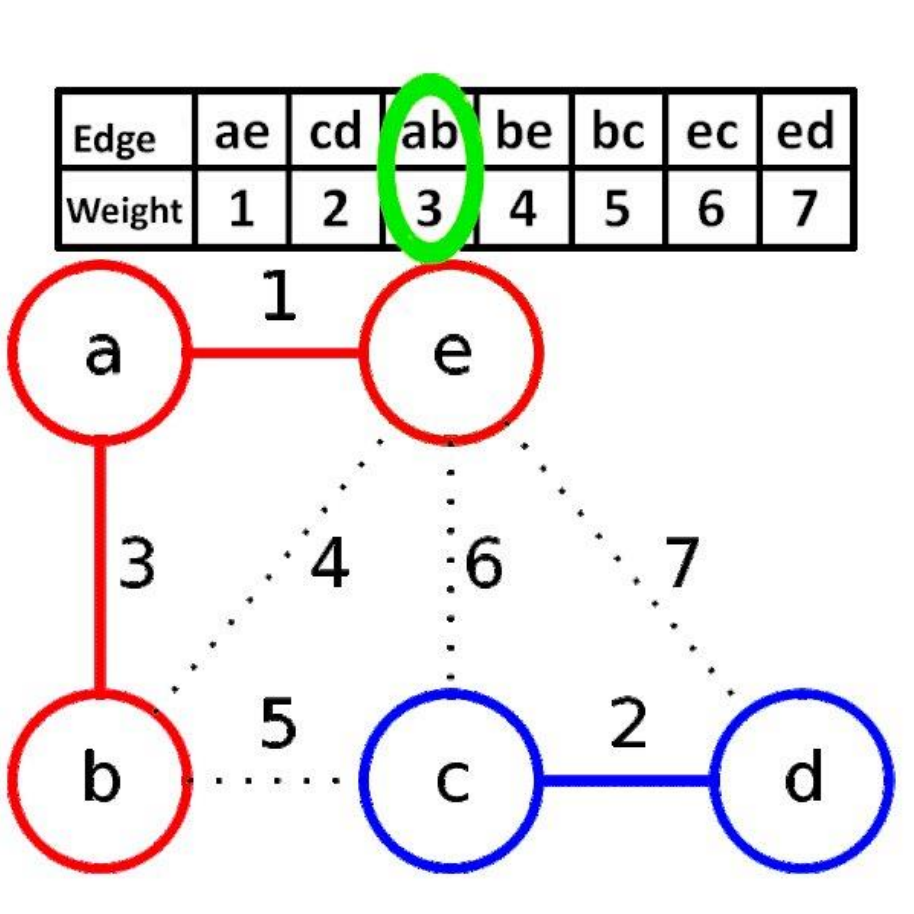
# Kruskal



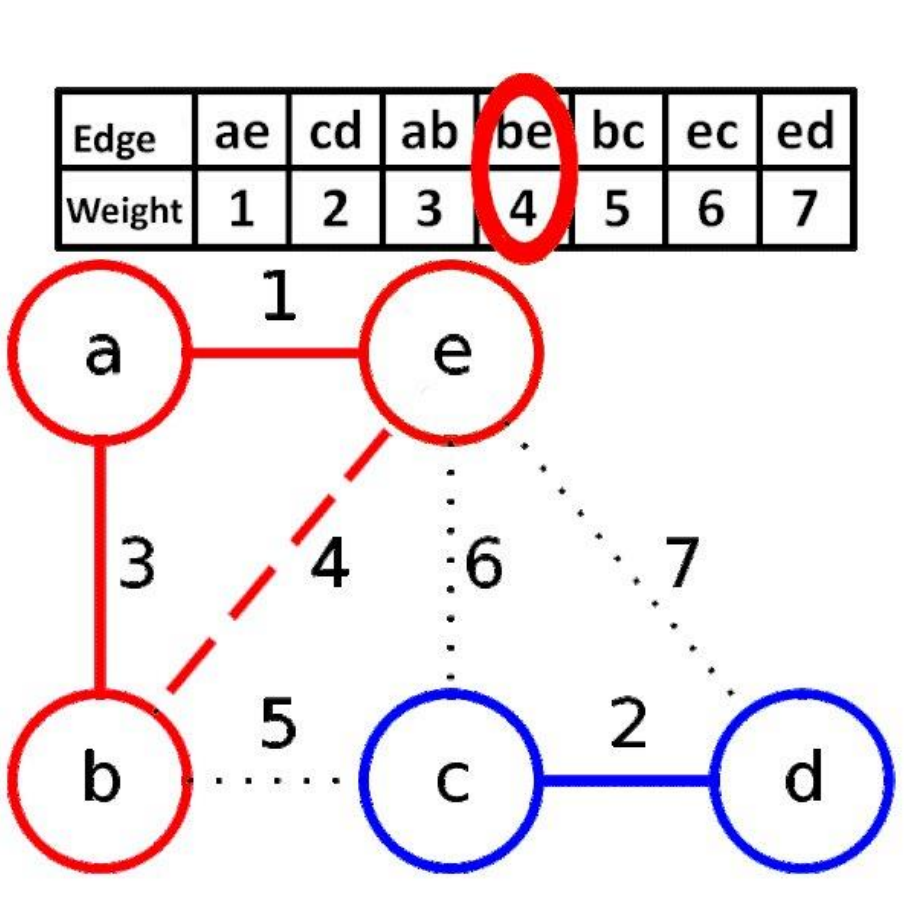
# Kruskal



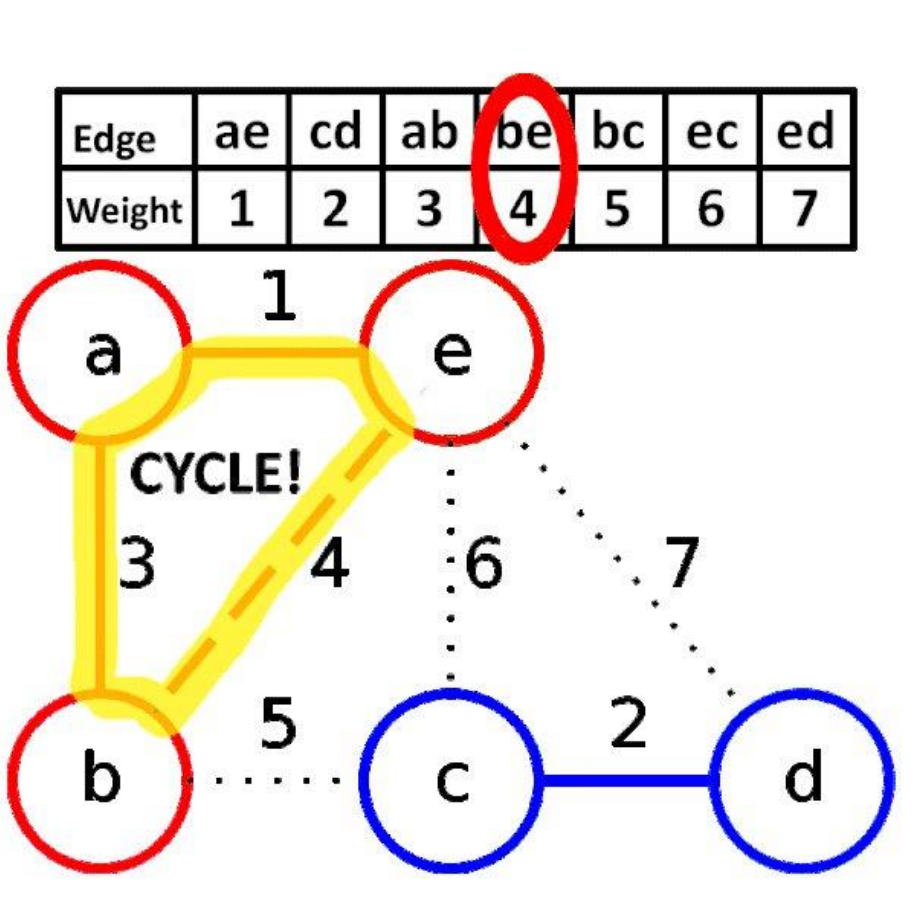
# Kruskal



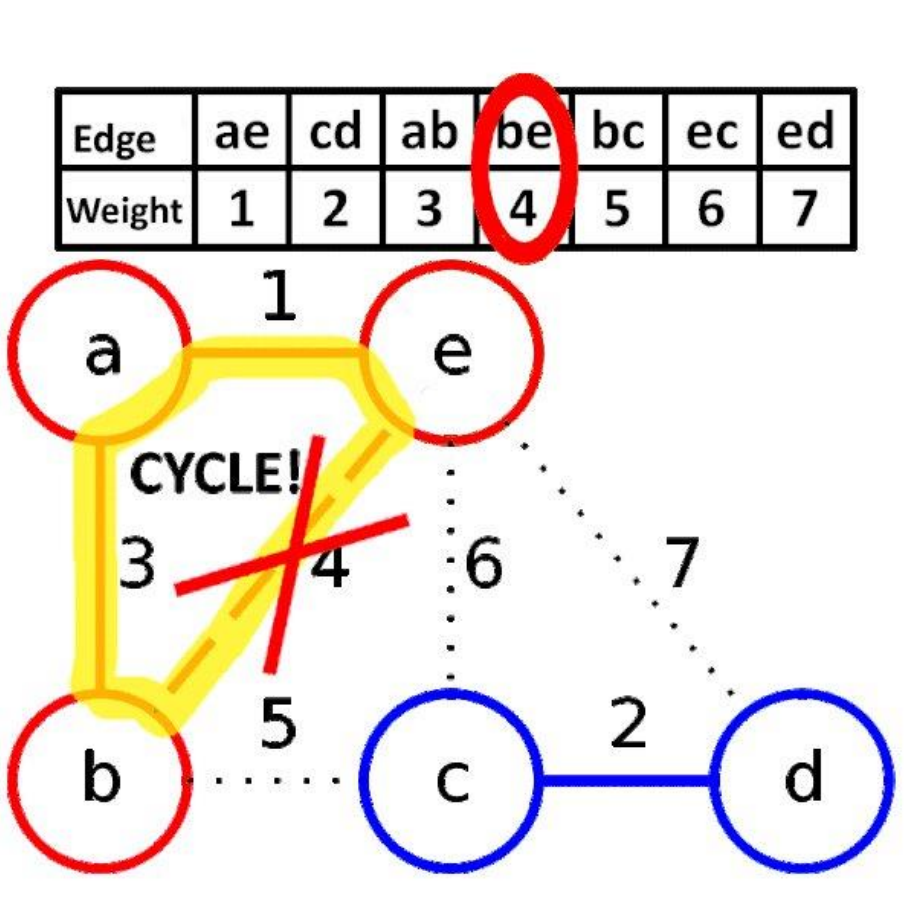
# Kruskal



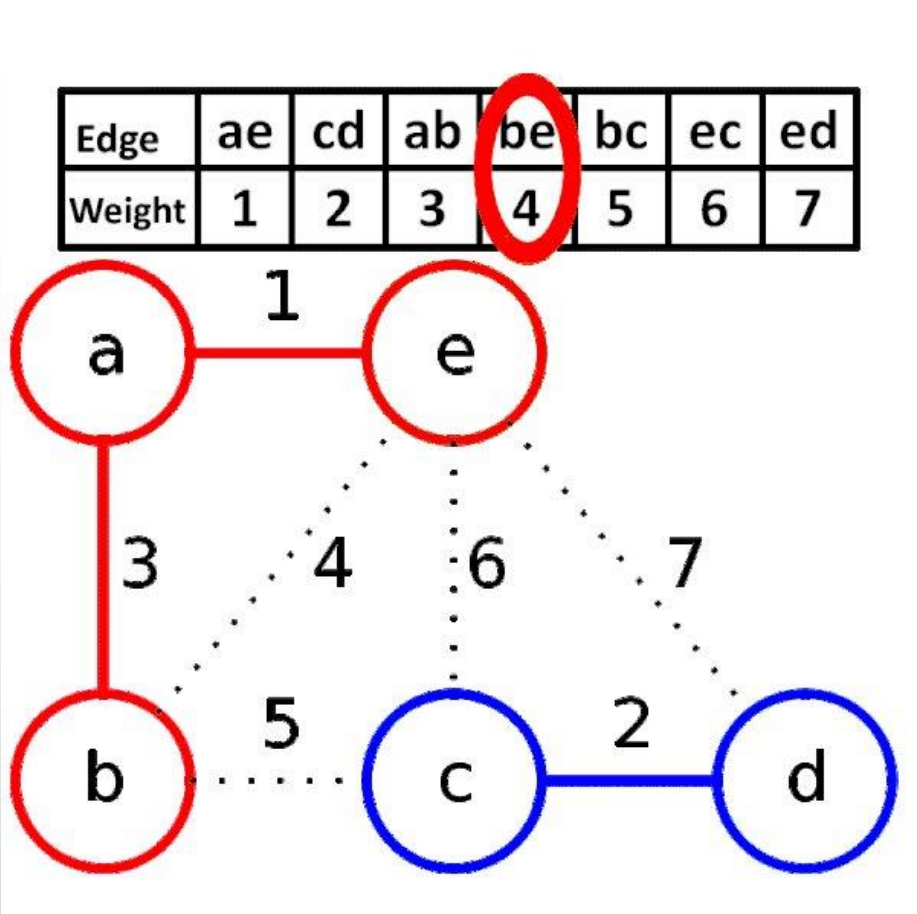
# Kruskal



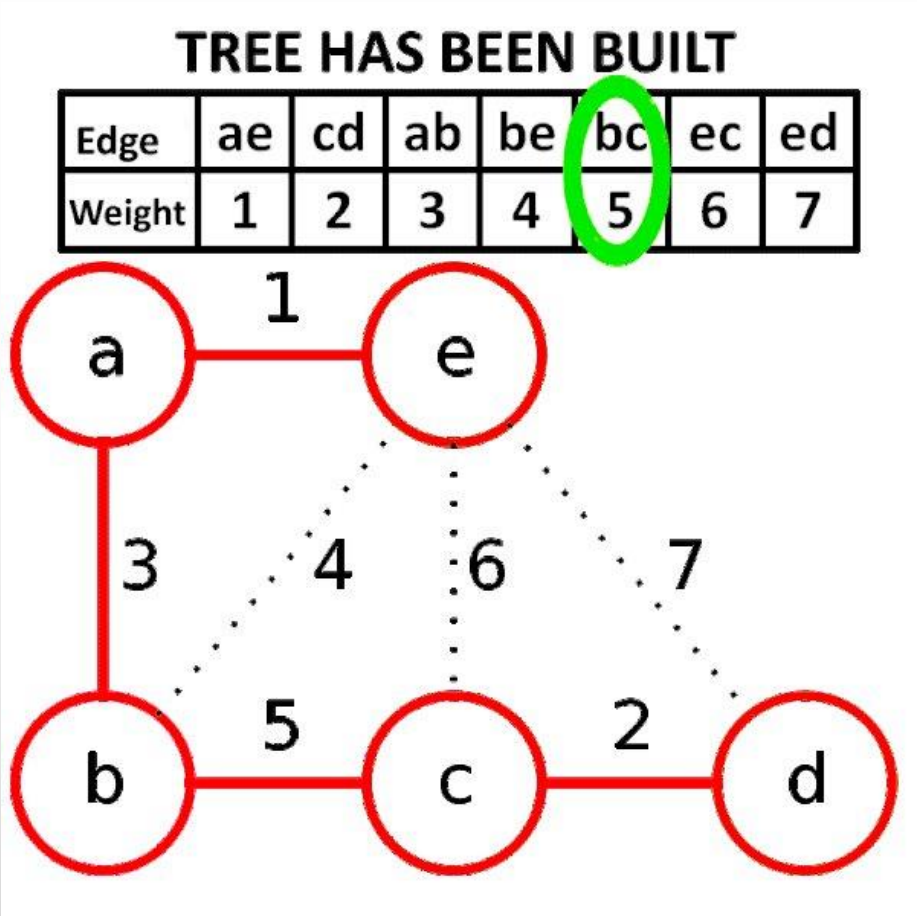
# Kruskal



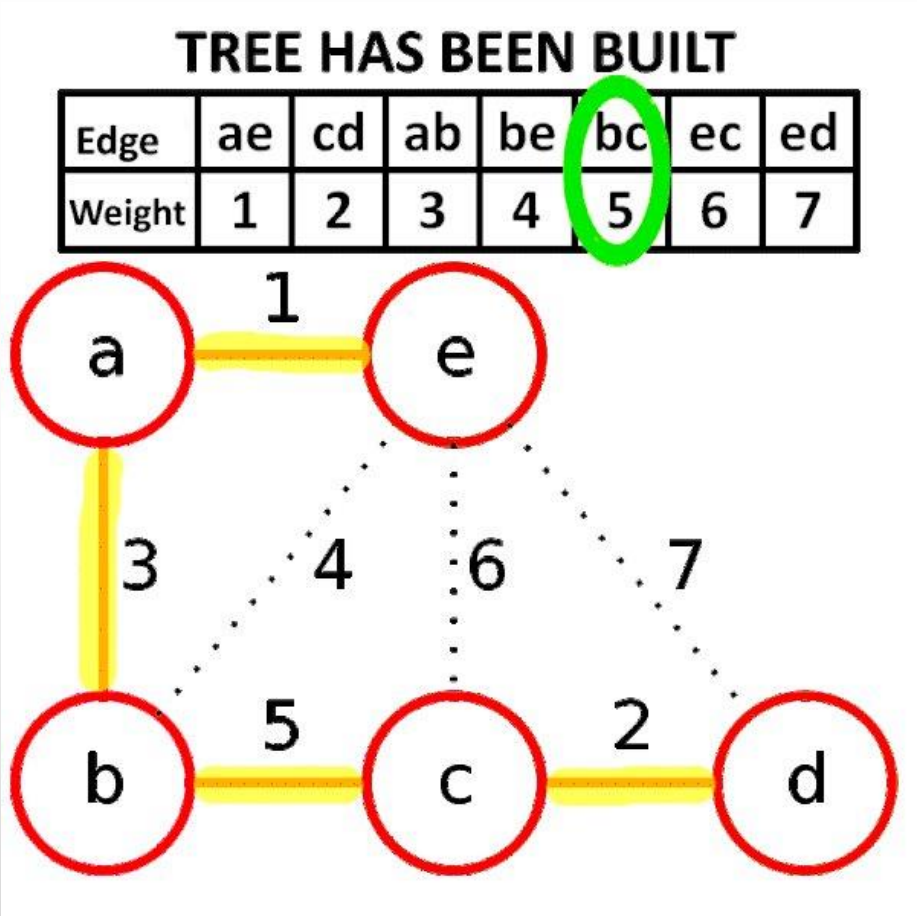
# Kruskal



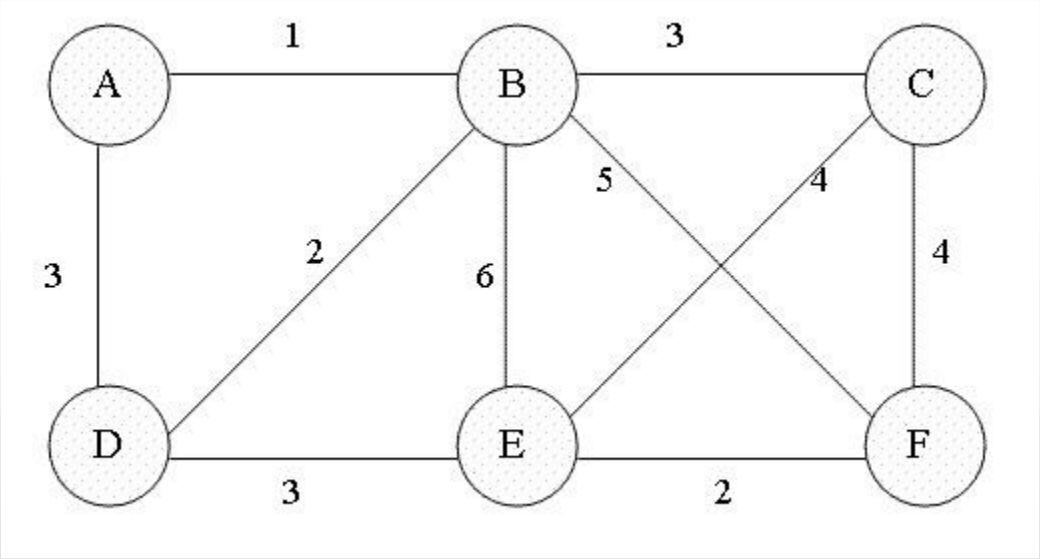
# Kruskal



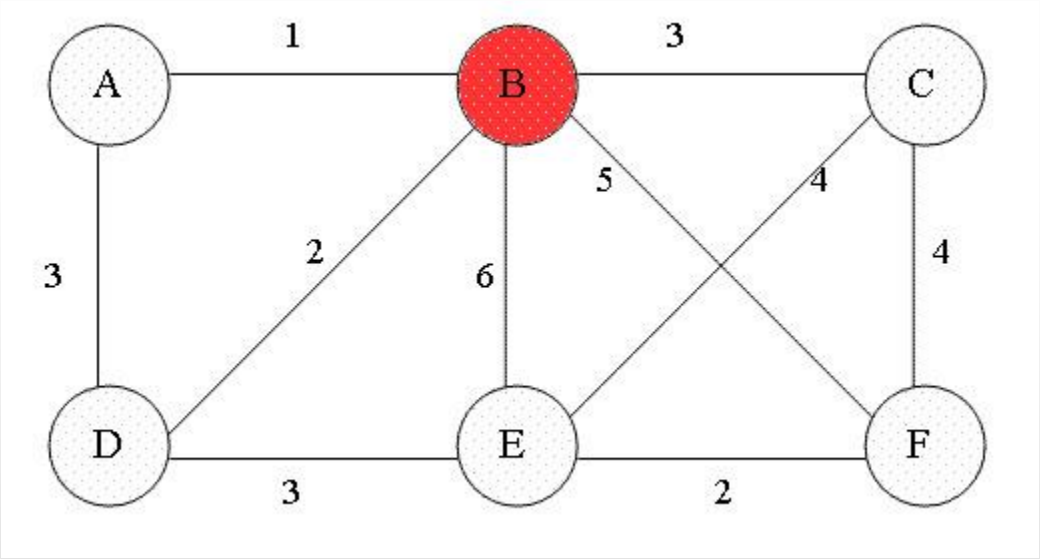
# Kruskal



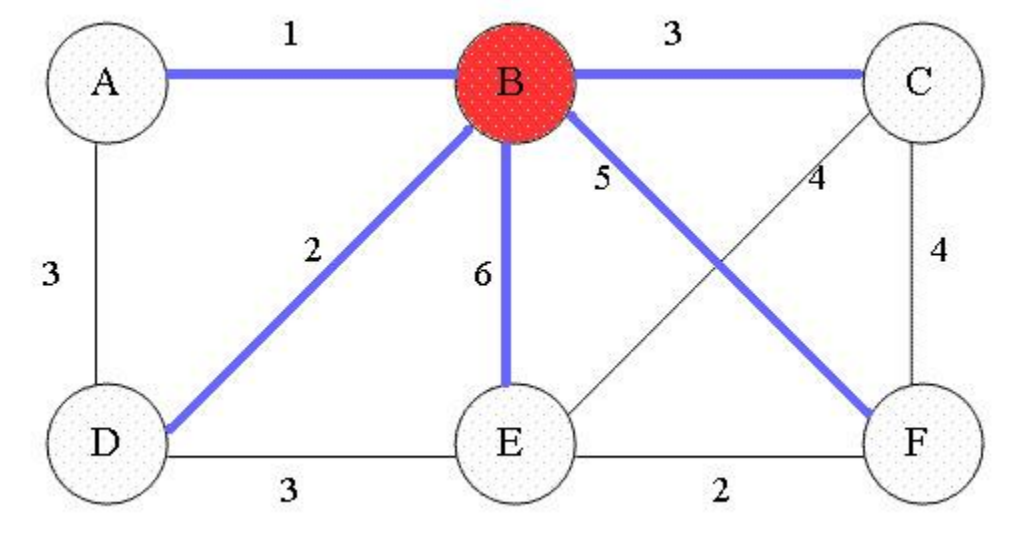
# Prim



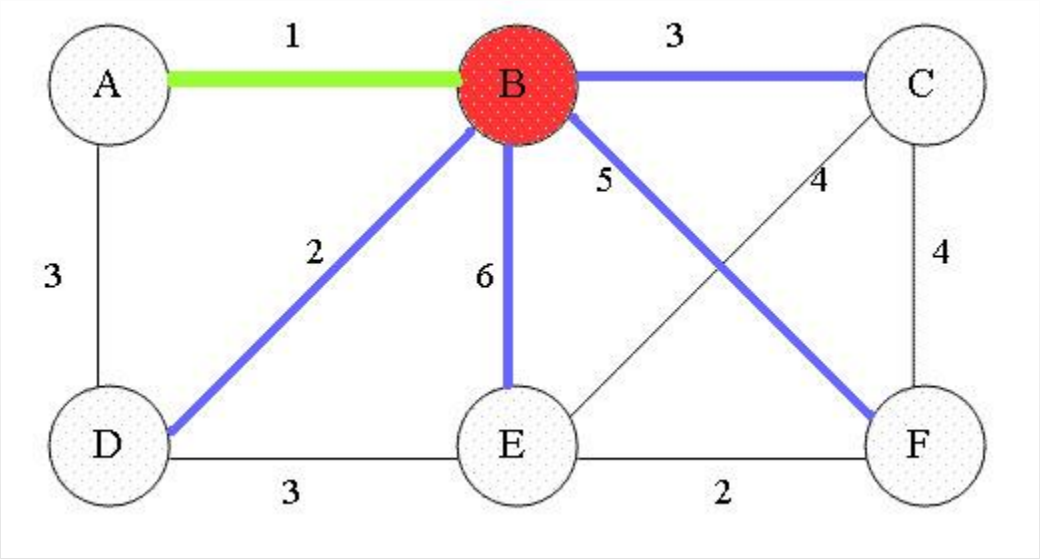
# Prim



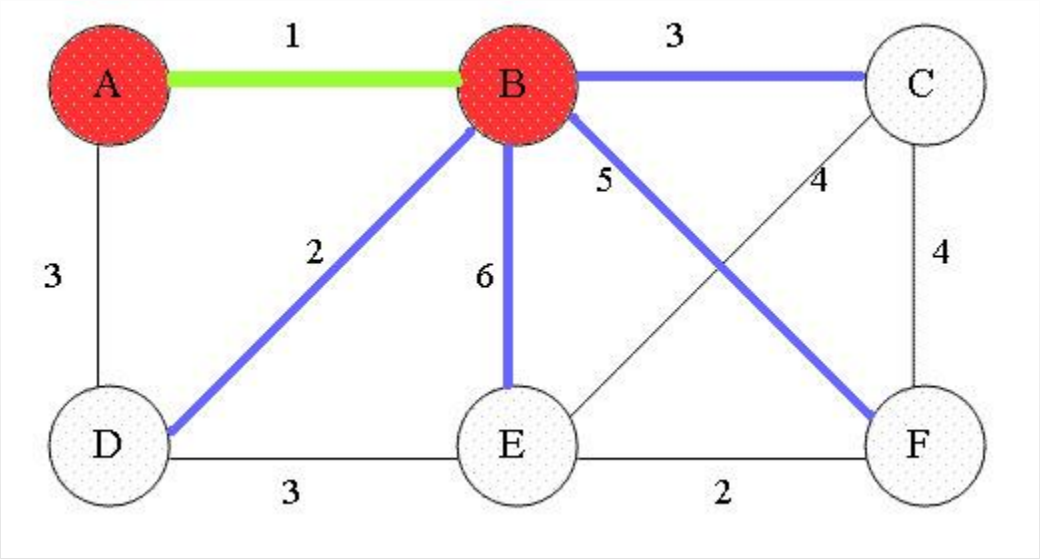
# Prim



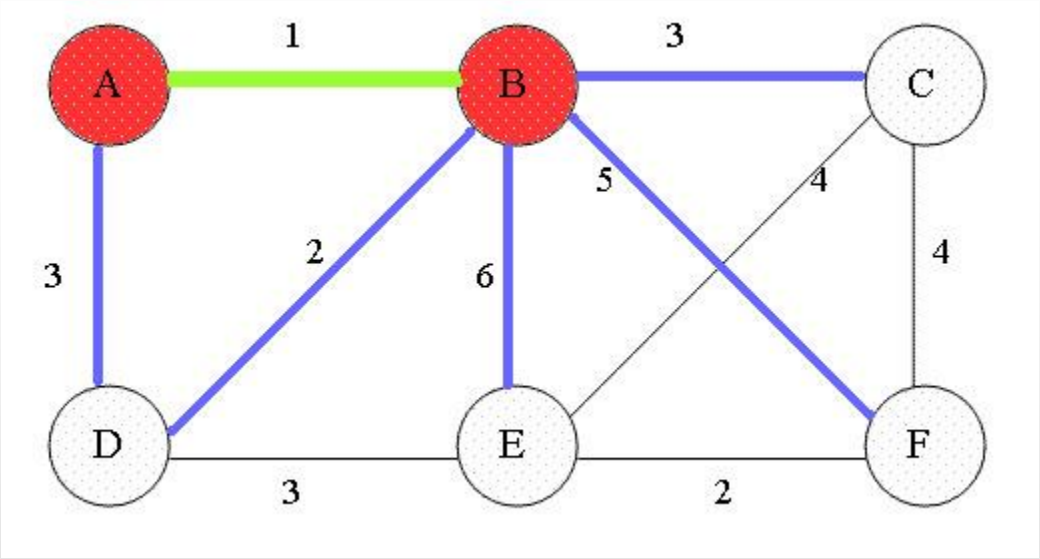
# Prim



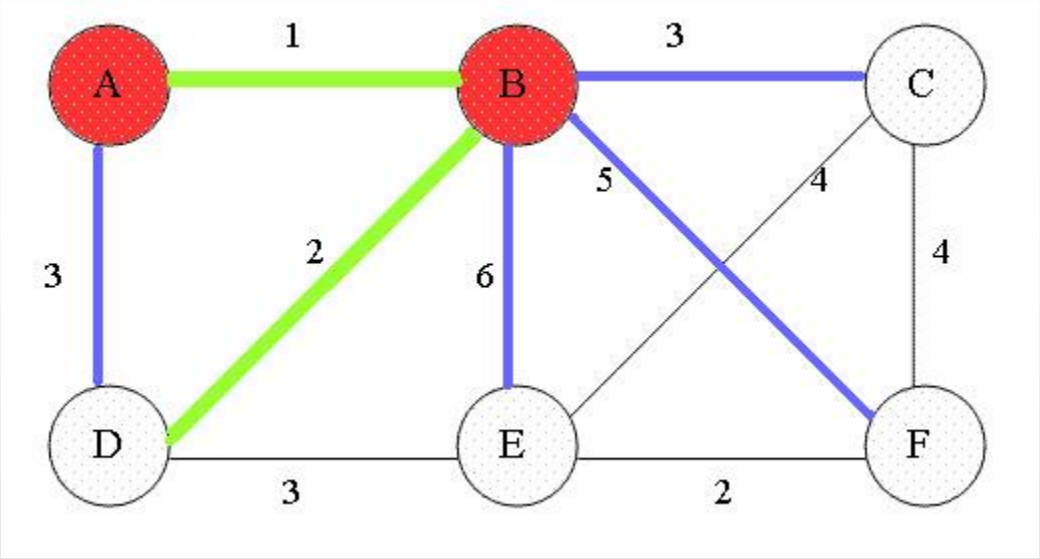
# Prim



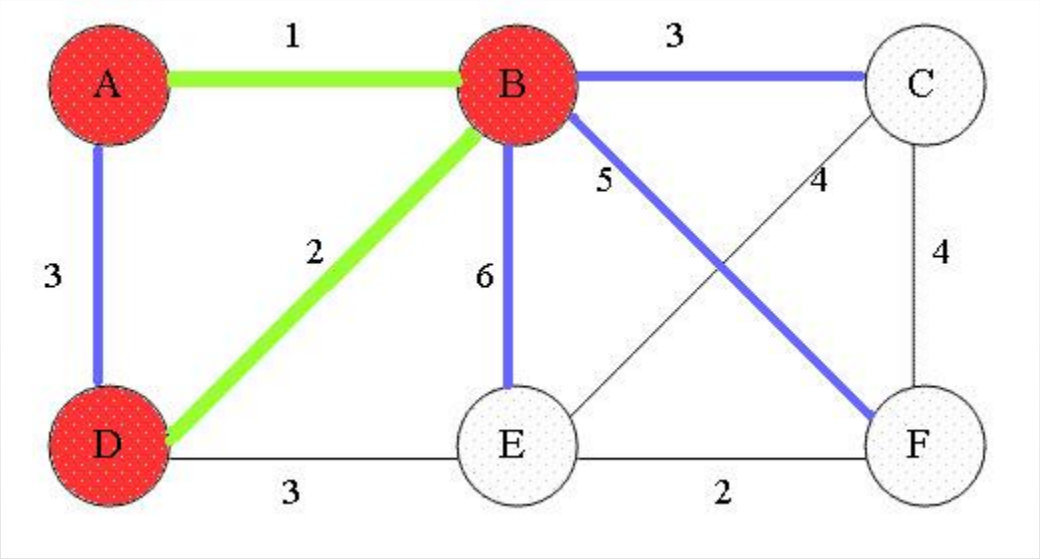
# Prim



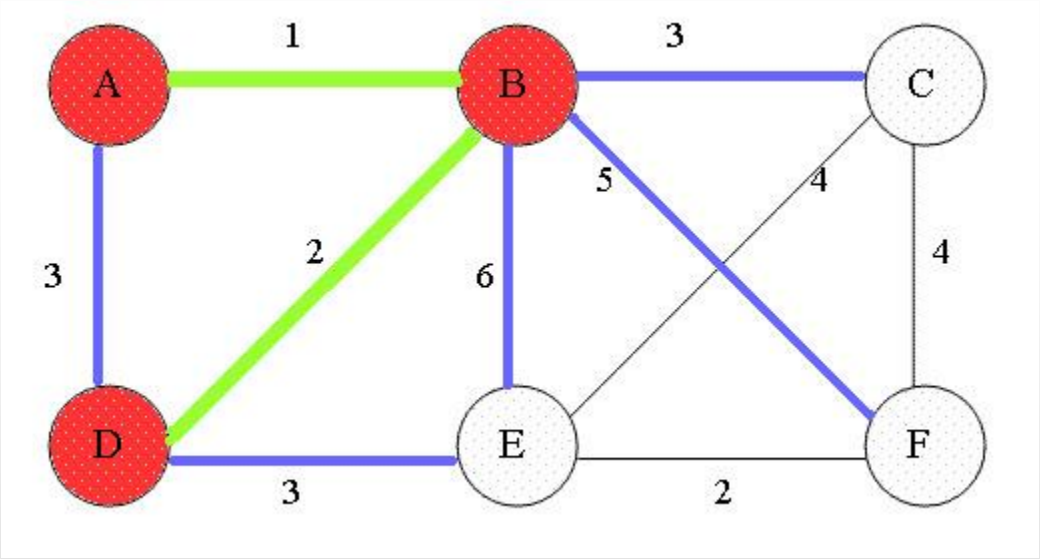
# Prim



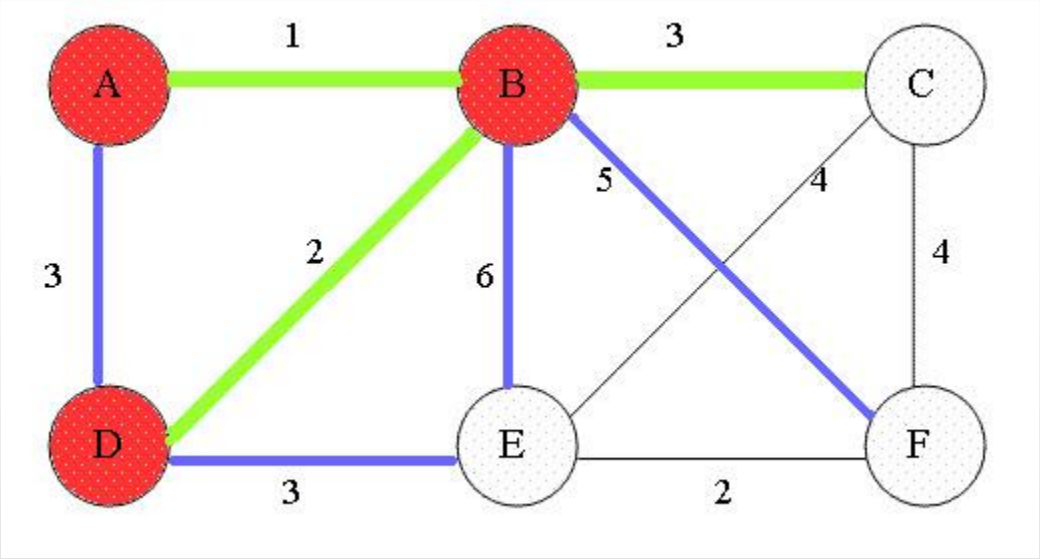
# Prim



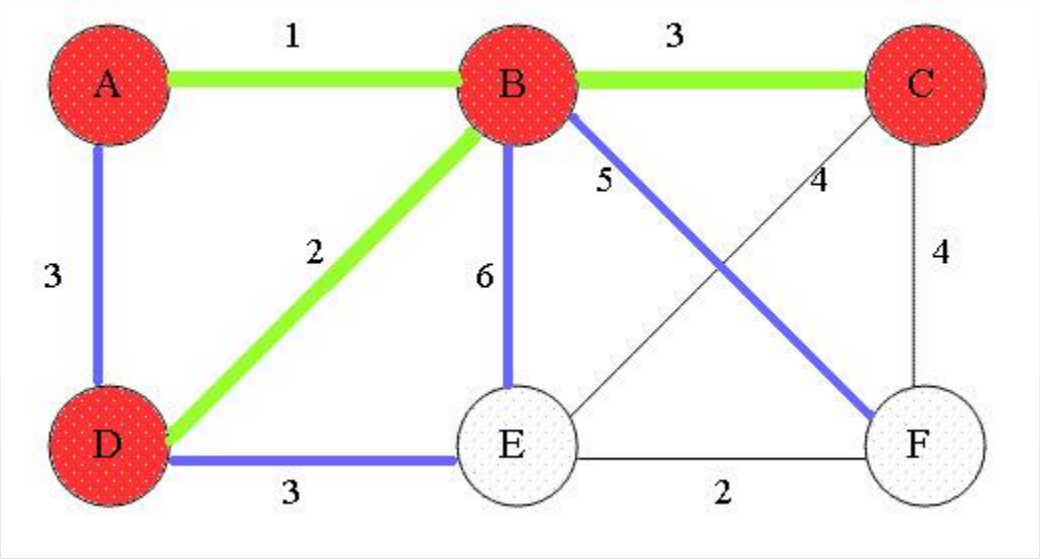
# Prim



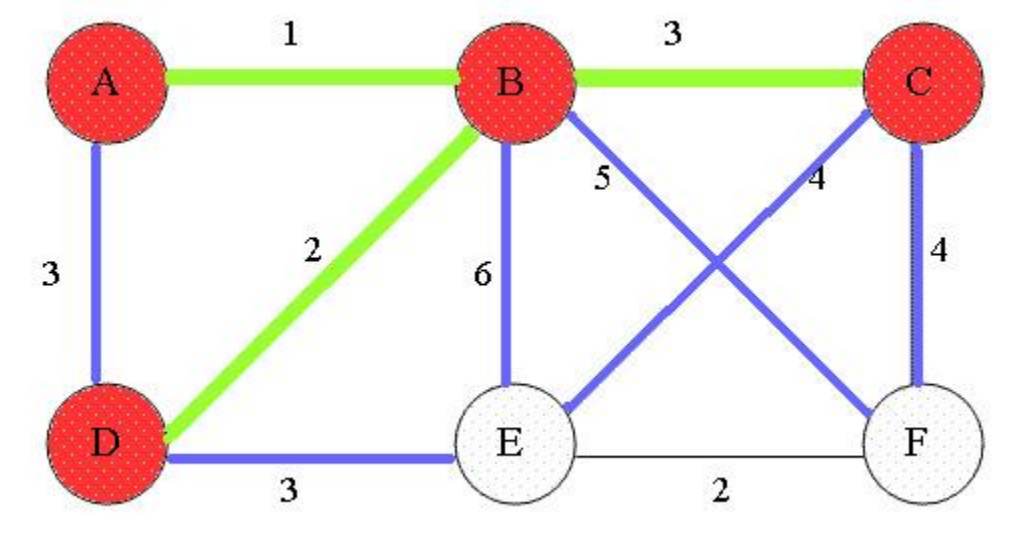
# Prim



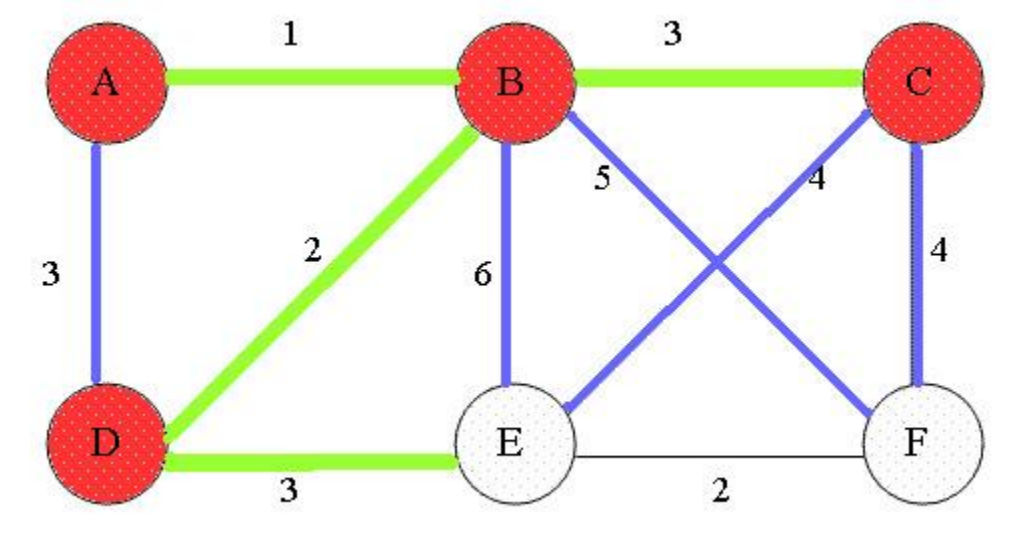
# Prim



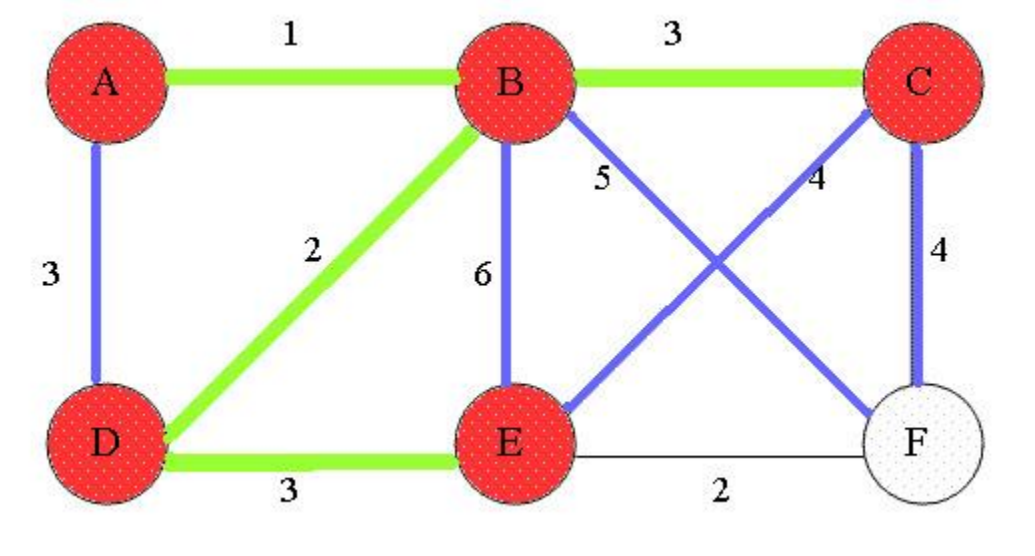
# Prim



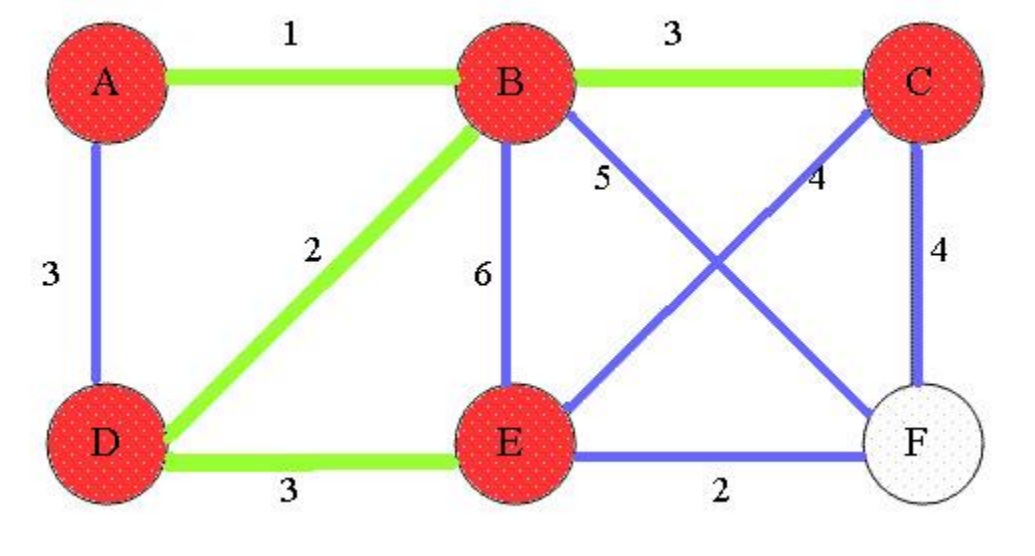
# Prim



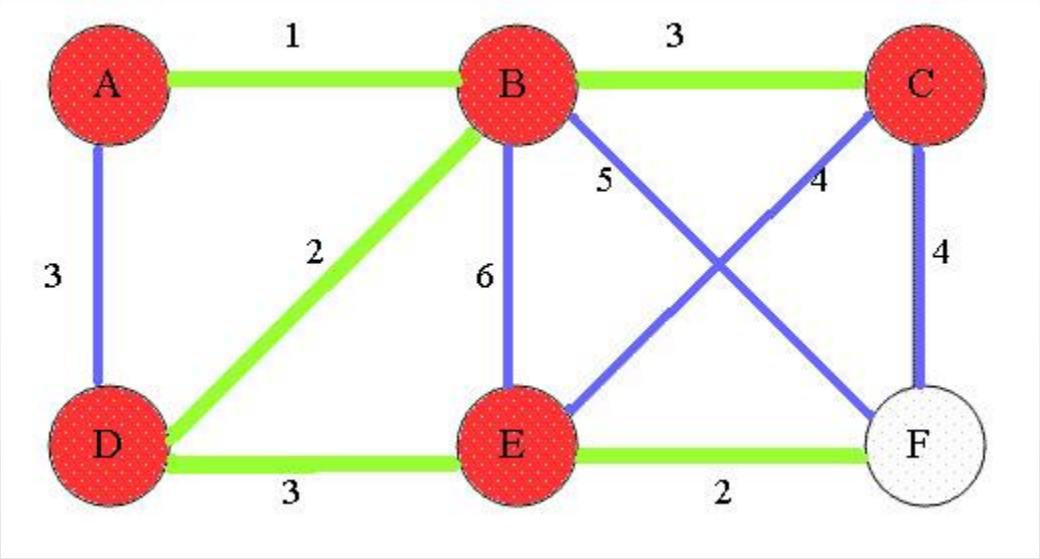
# Prim



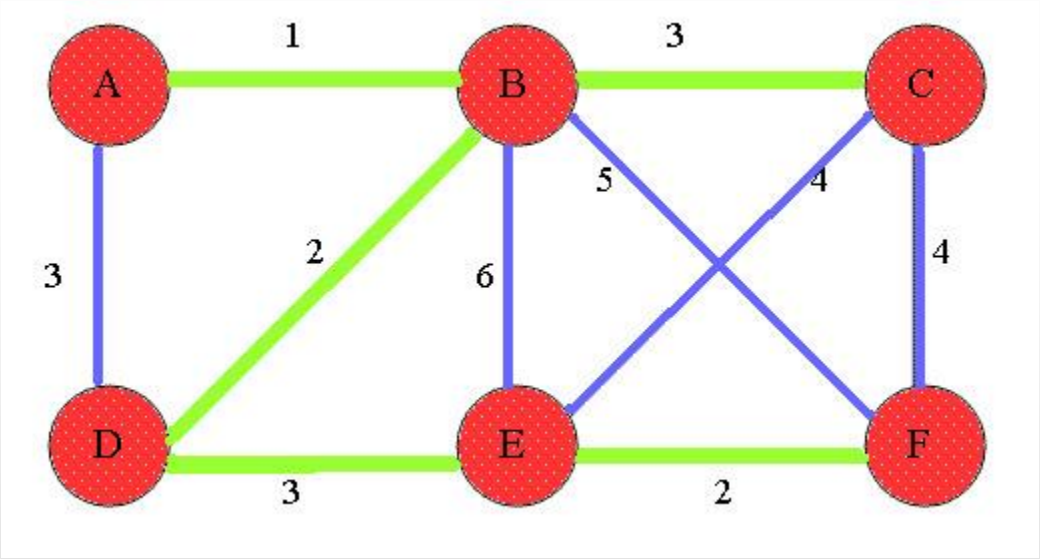
# Prim



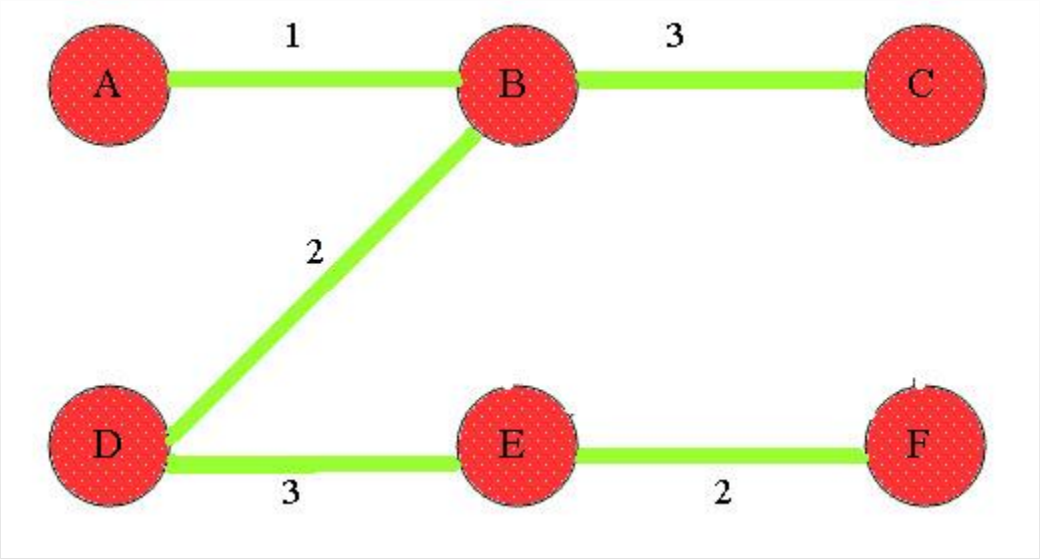
# Prim



# Prim



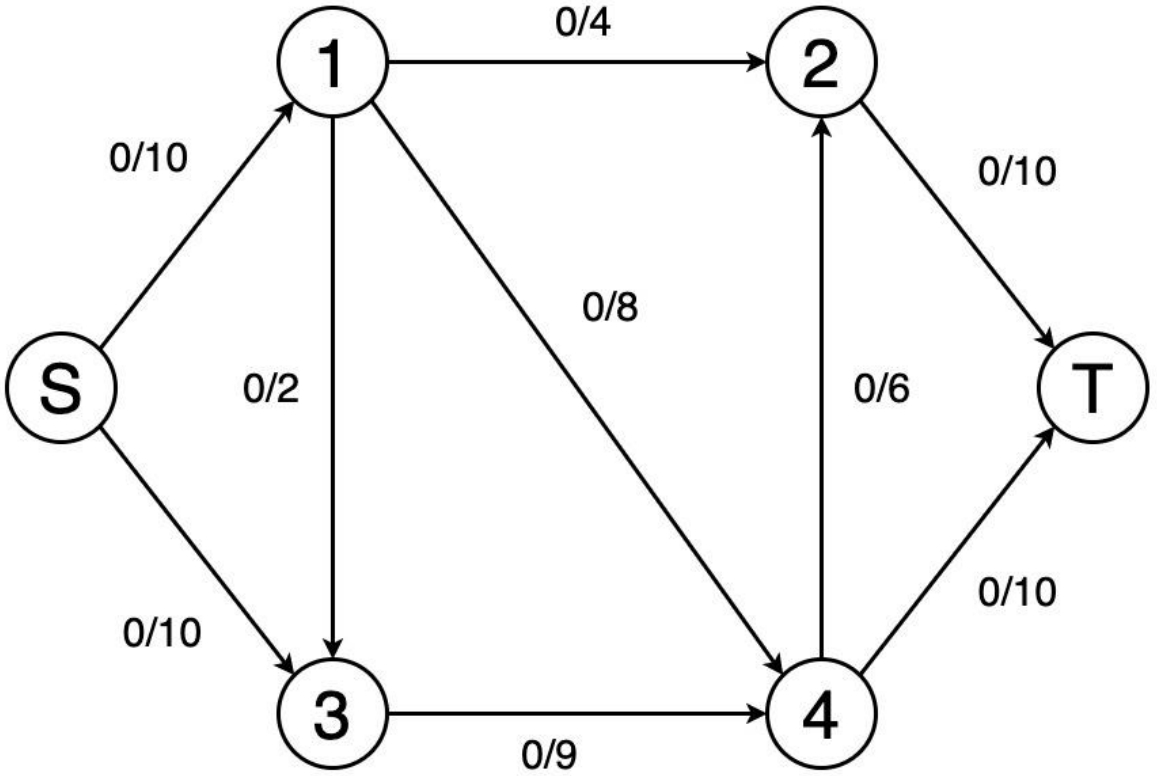
# Prim



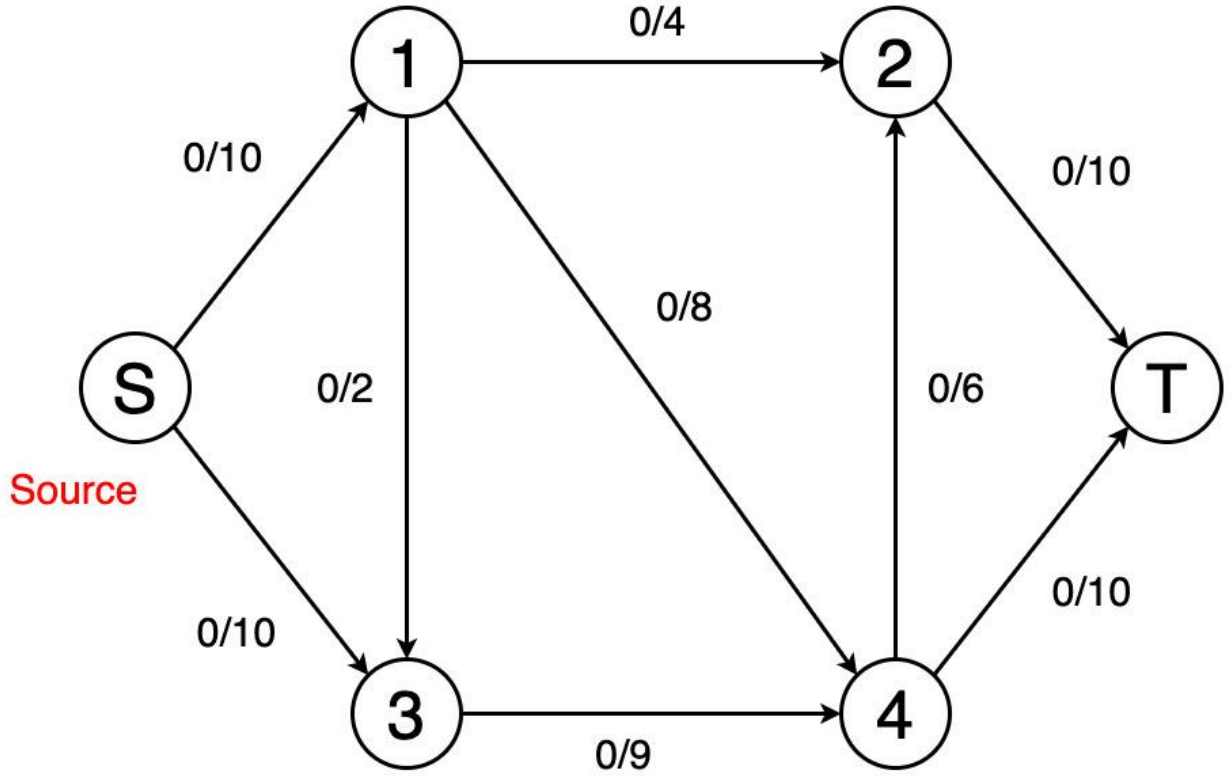
# Ağ Akış Algoritmaları (Network Flow)

- Ağ üzerinde kaynak noktasından hedef noktasına belirli kısıtlamalar altında akışın nasıl optimize edileceğini modeller.
- Amaç, kısıtlamalar altında kaynak noktasından hedef noktasına mümkün olan en fazla akışı sağlamaktır.
- *Ford-Fulkerson*, en basit ve anlaşılır algoritmalardan biridir, ancak çalışma süresi diğer algoritmalara göre daha uzun olabilir.
- *Edmonds-Karp*, *Ford-Fulkerson* algoritmasını geliştirerek çalışma süresini azaltır.
- *Dinic's*, *Edmonds-Karp* algoritmasından daha hızlı ve bazı durumlarda daha verimlidir.

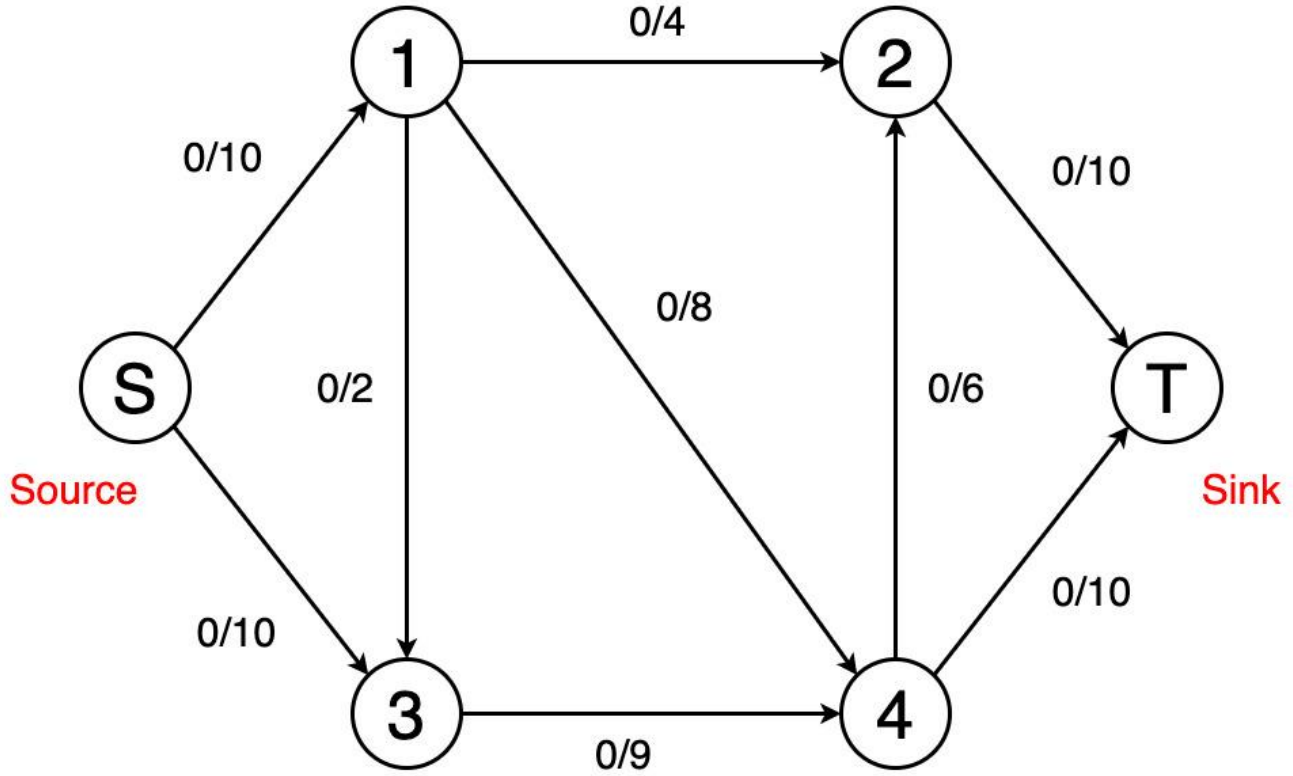
# Ford Fulkerson



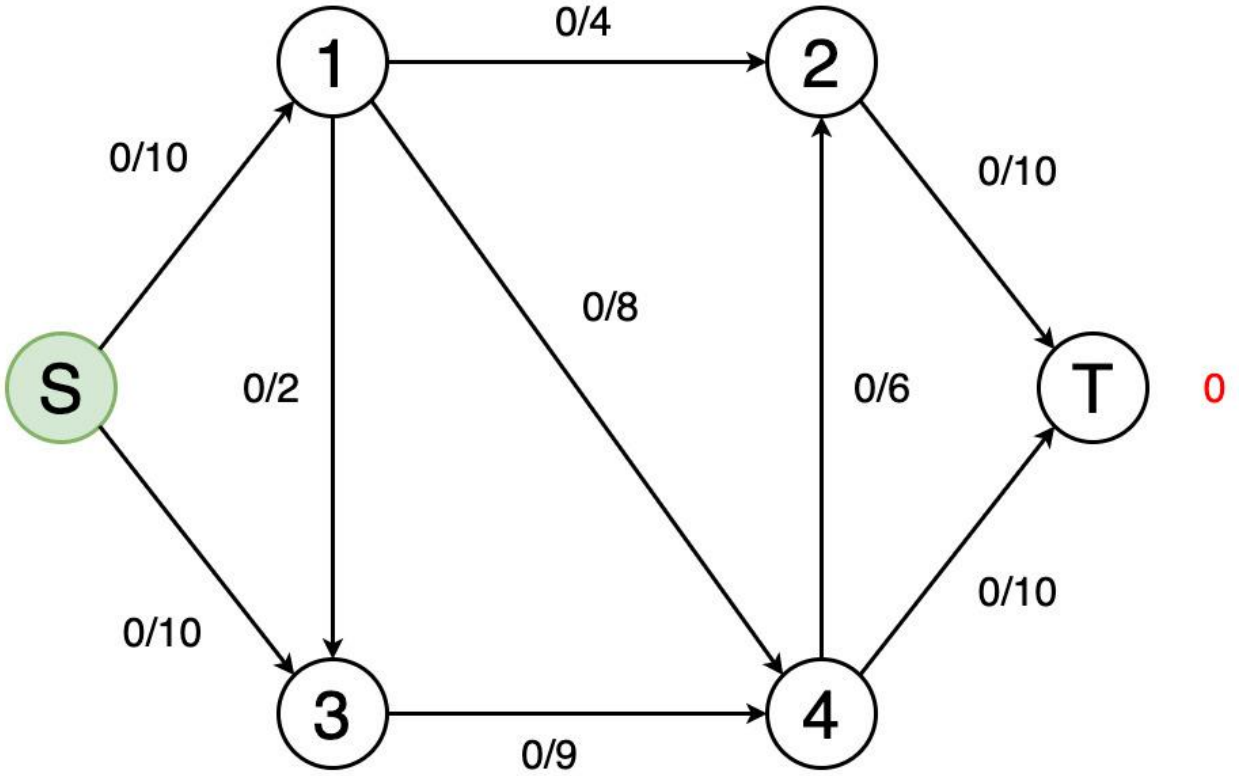
# Ford Fulkerson



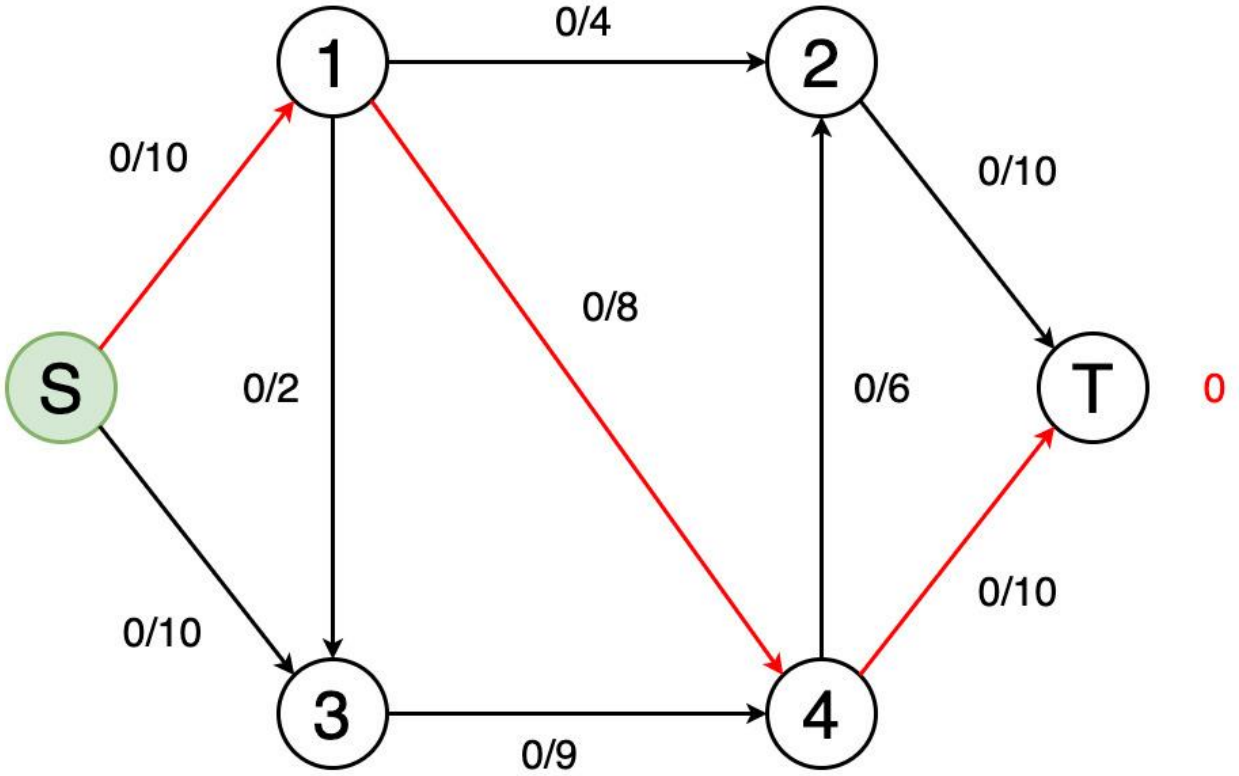
# Ford Fulkerson



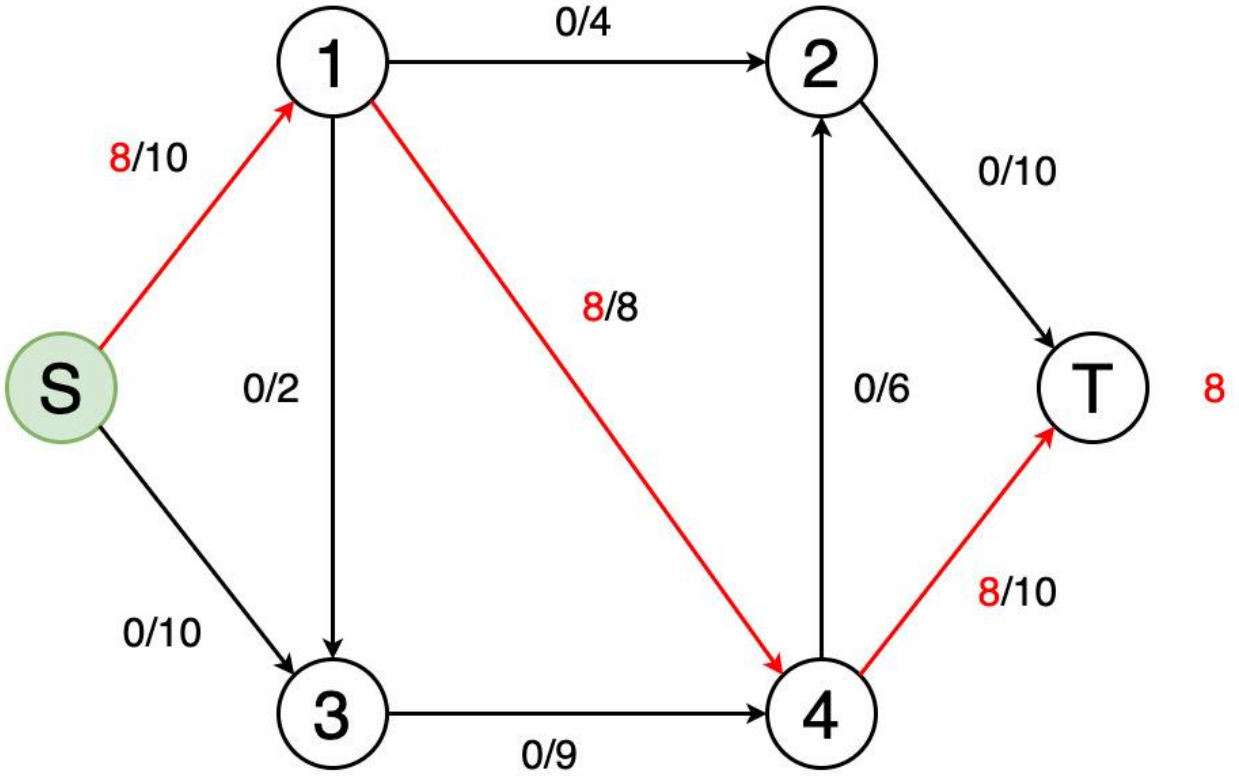
# Ford Fulkerson



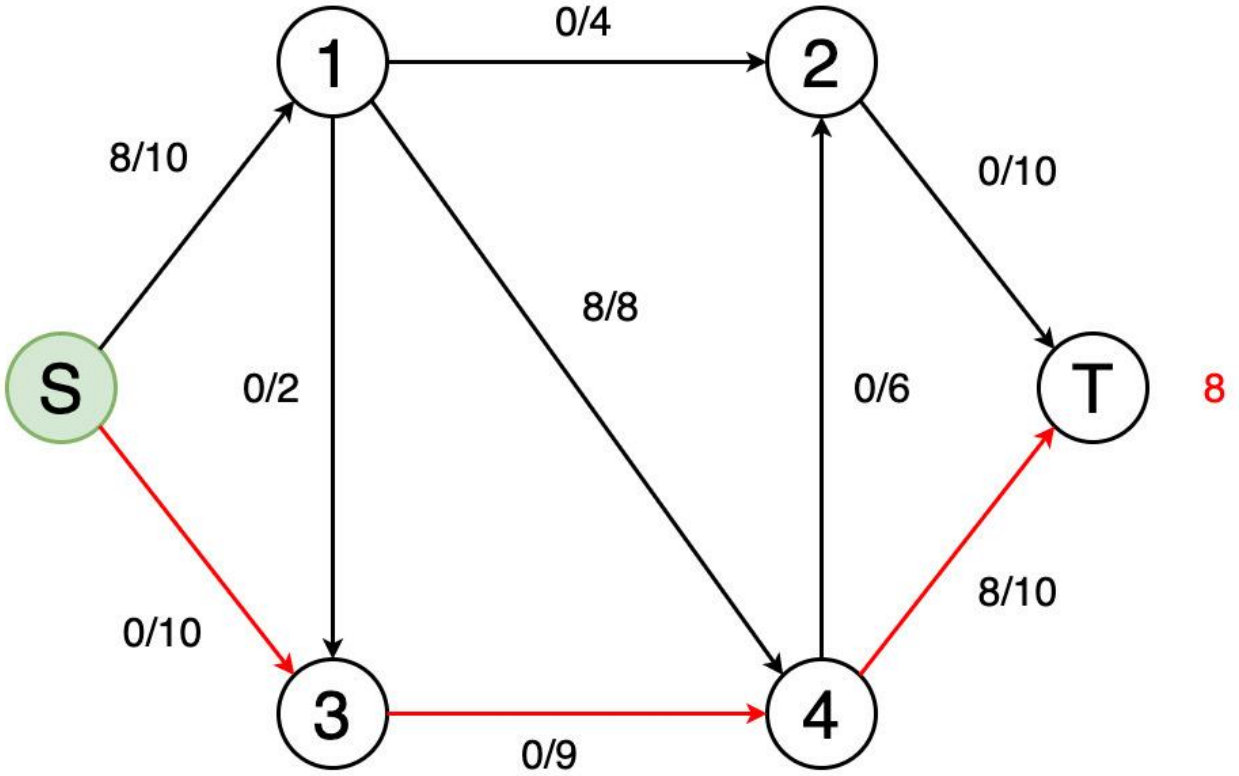
# Ford Fulkerson



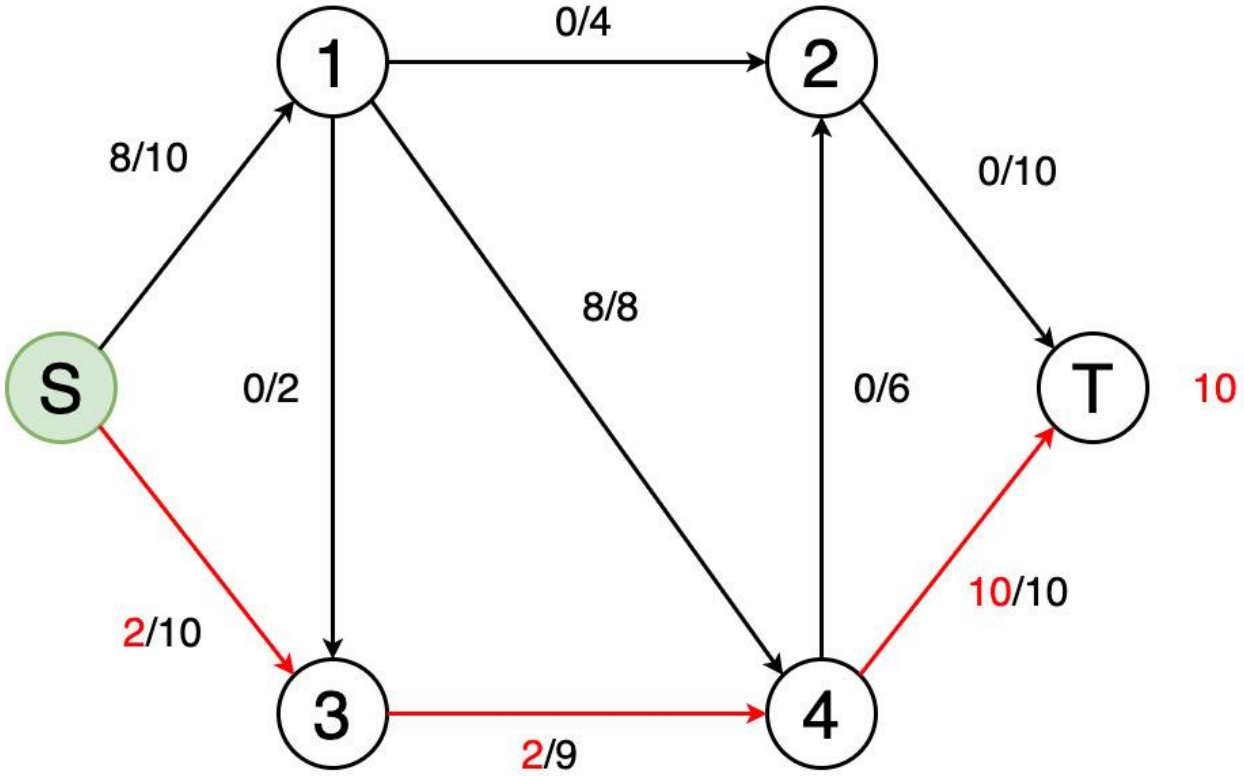
# Ford Fulkerson



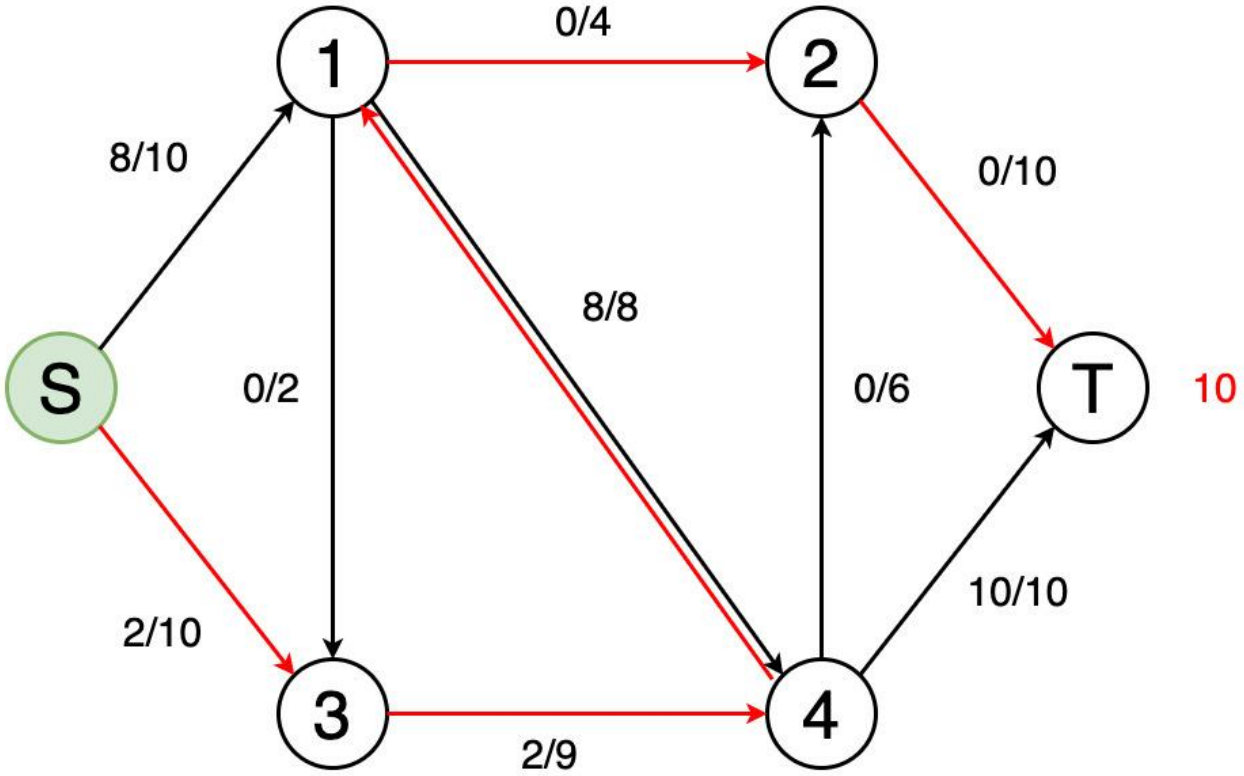
# Ford Fulkerson



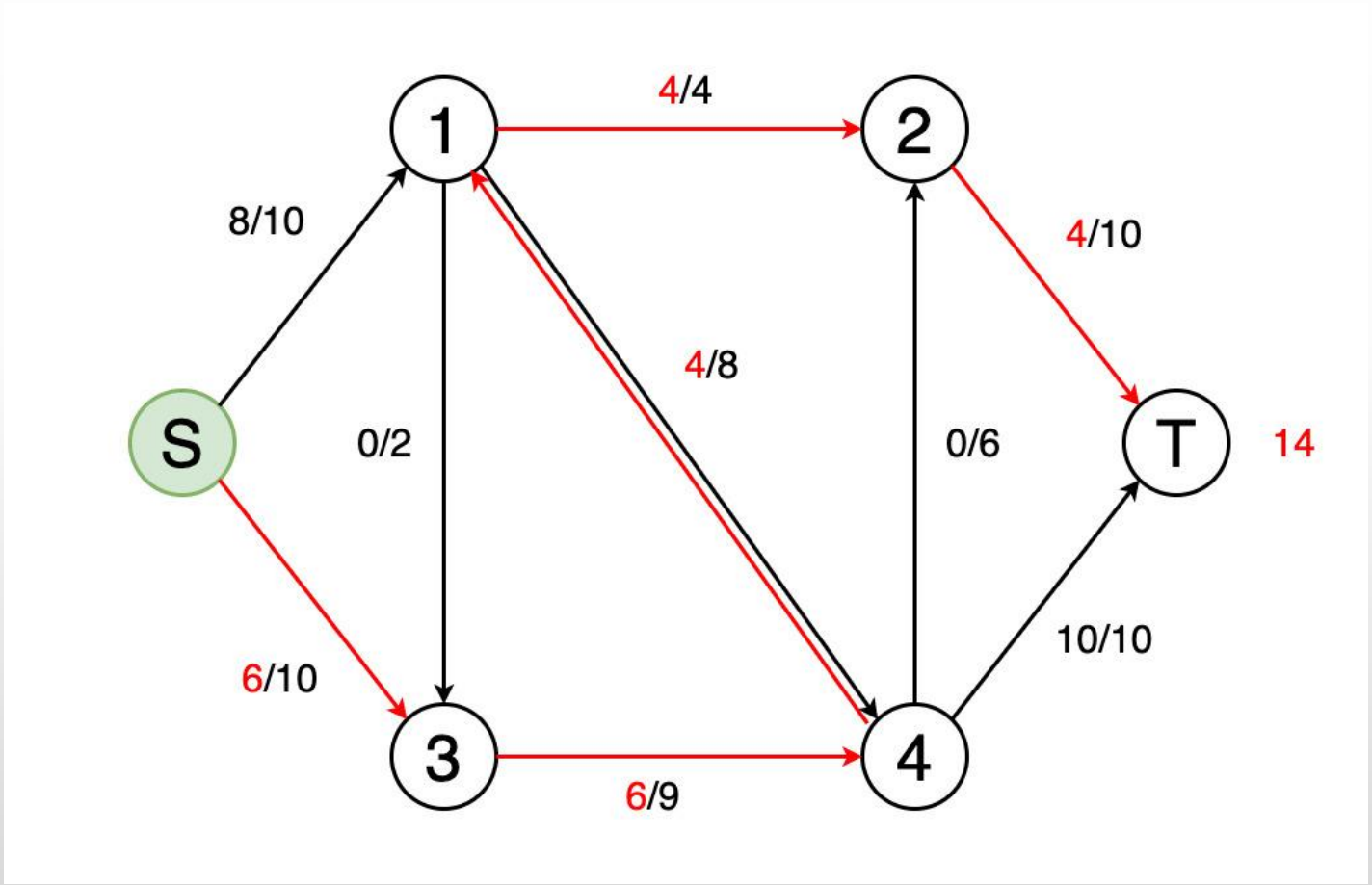
# Ford Fulkerson



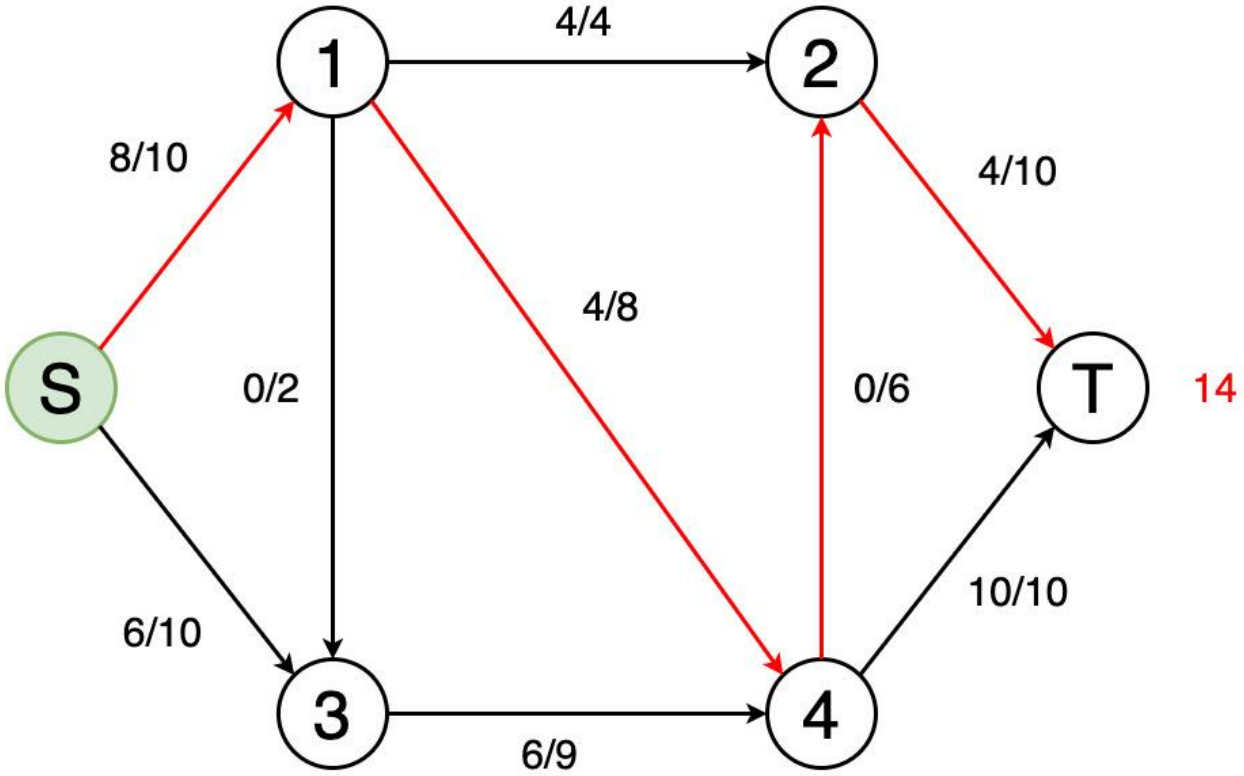
# Ford Fulkerson



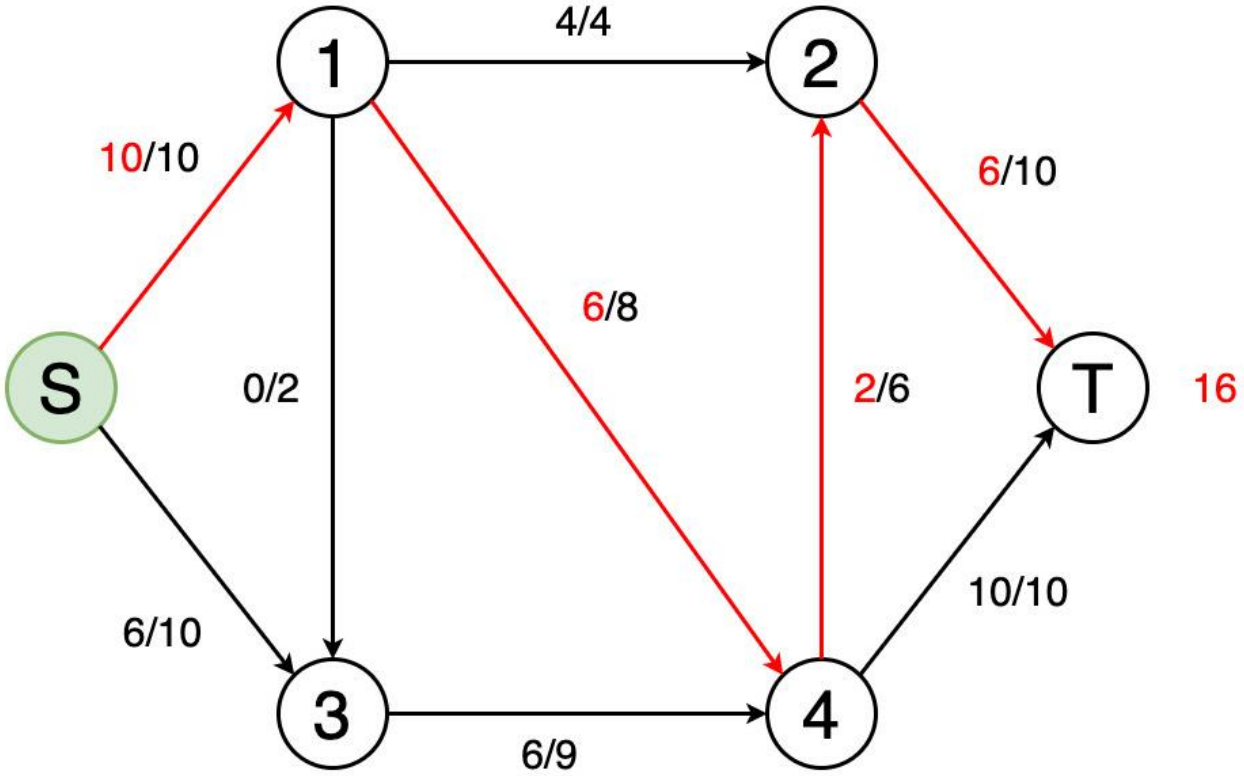
# Ford Fulkerson



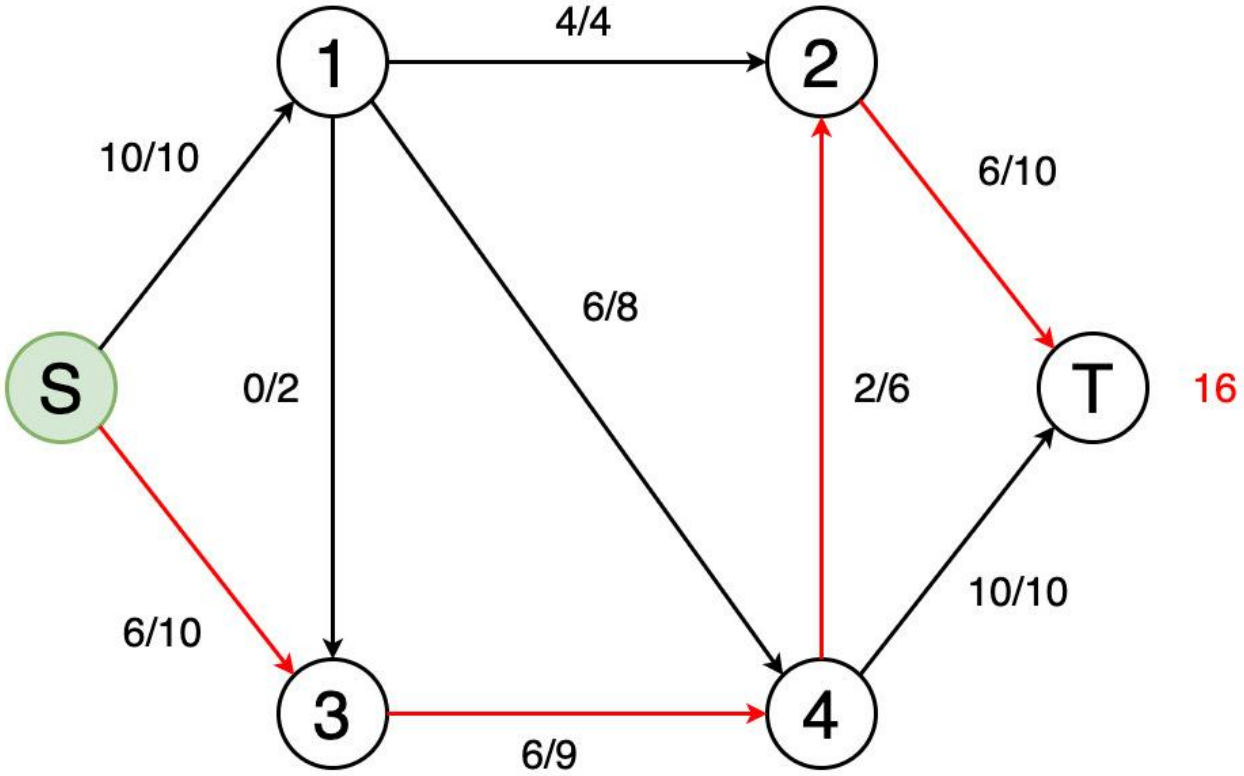
# Ford Fulkerson



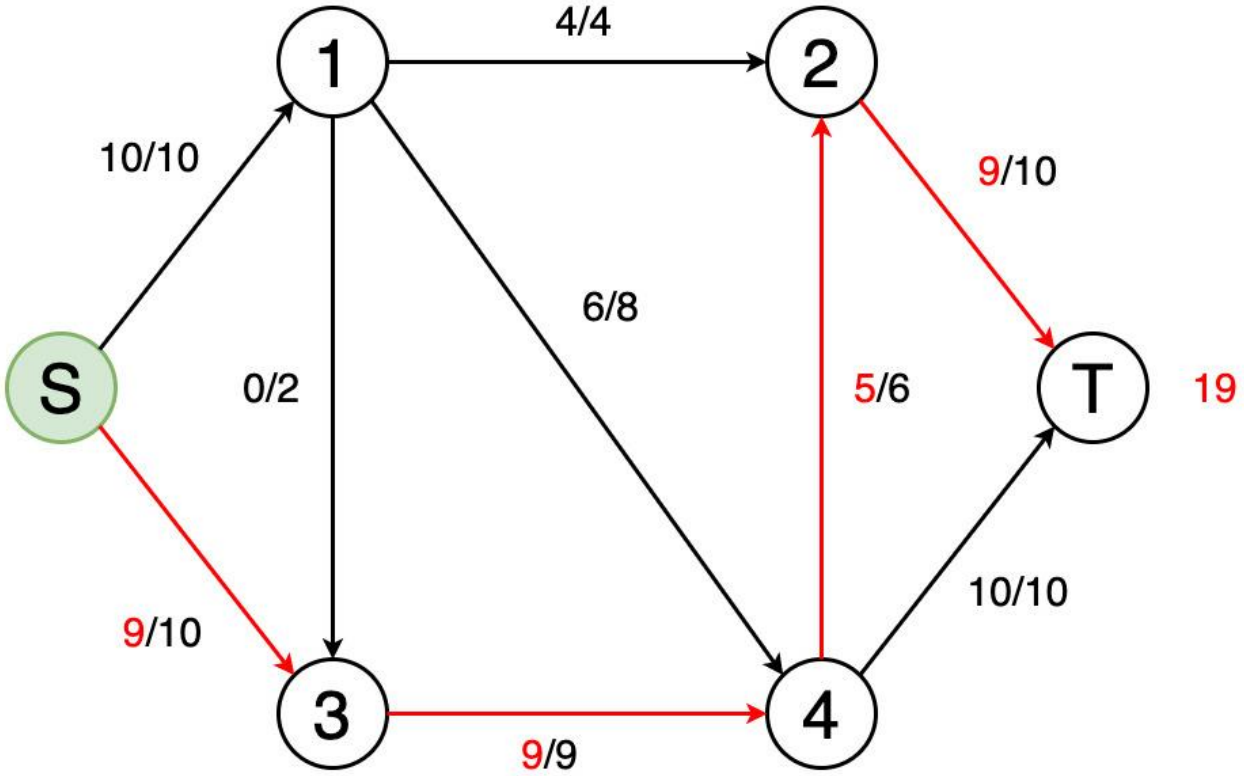
# Ford Fulkerson



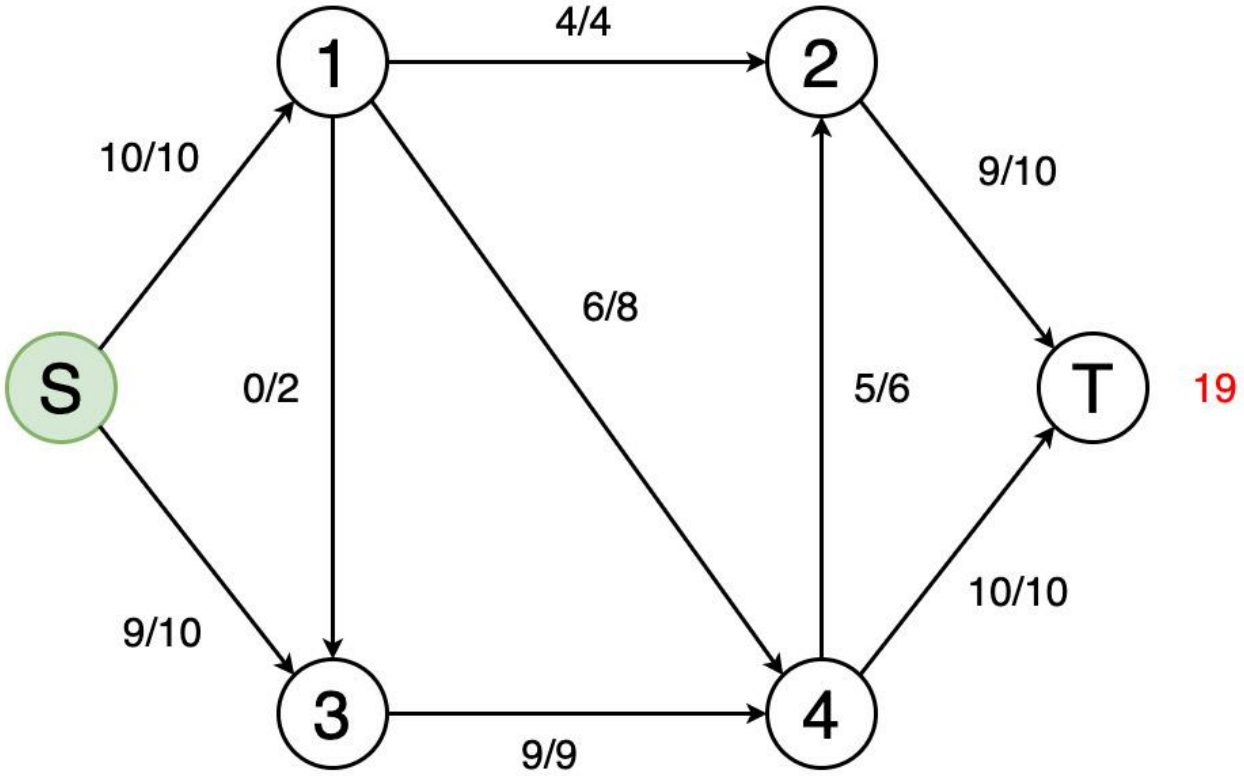
# Ford Fulkerson



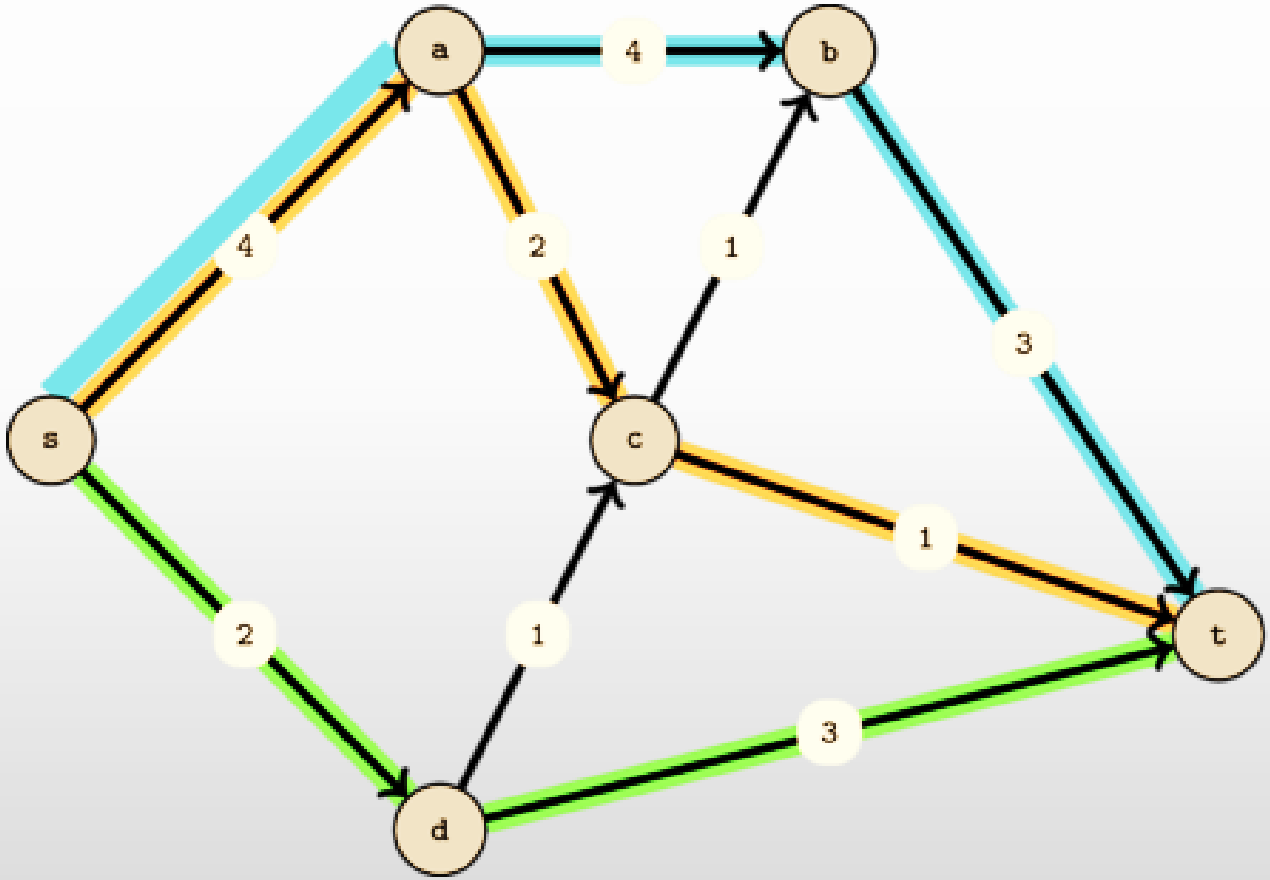
# Ford Fulkerson



# Ford Fulkerson

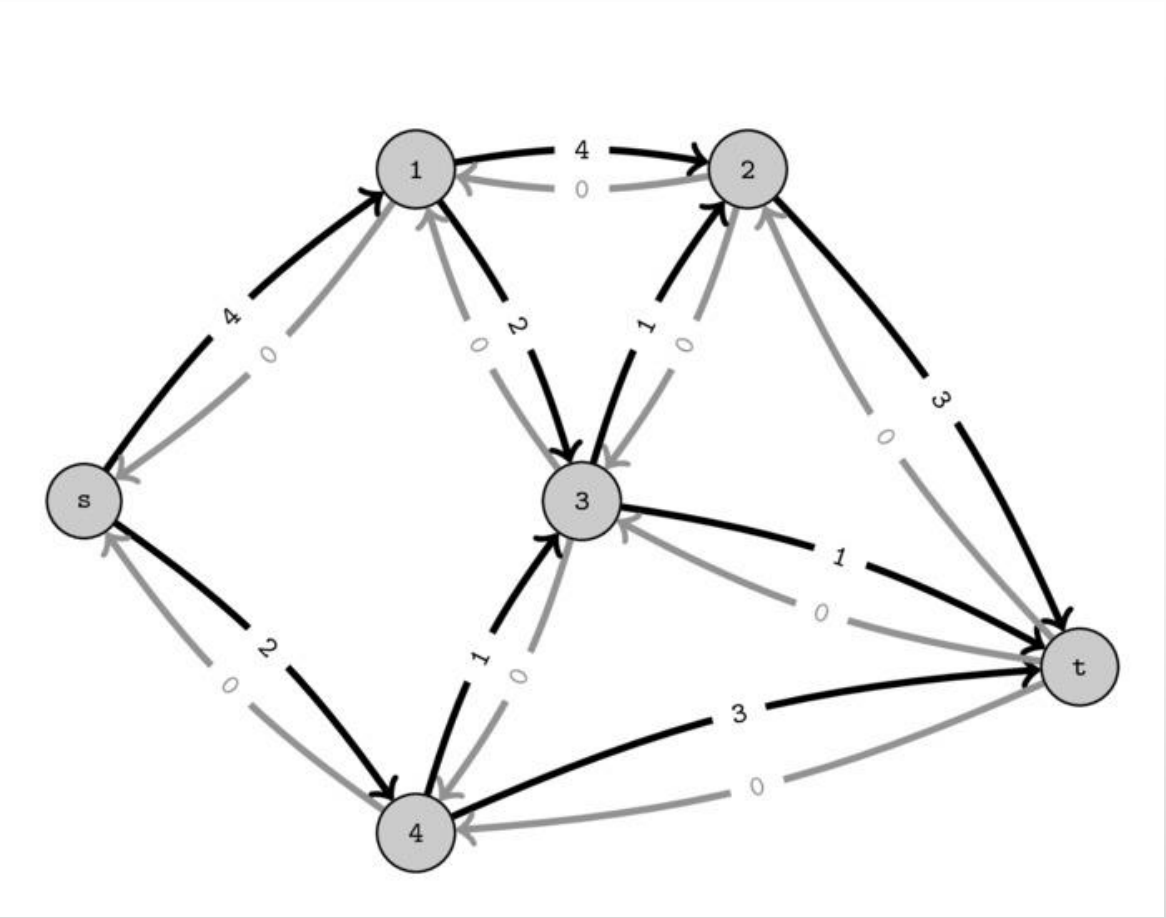


# Edmonds Karp

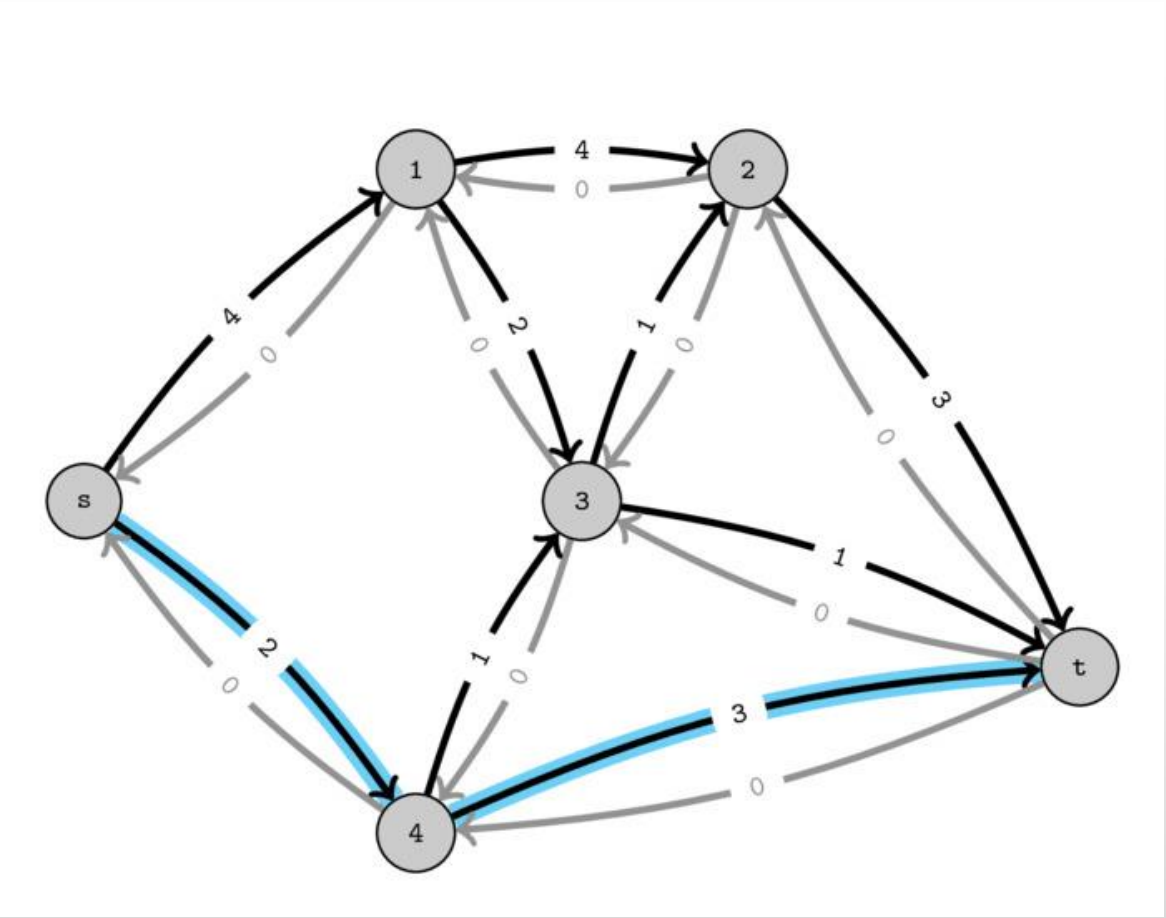



3 units of flow
1 unit of flow
2 units of flow

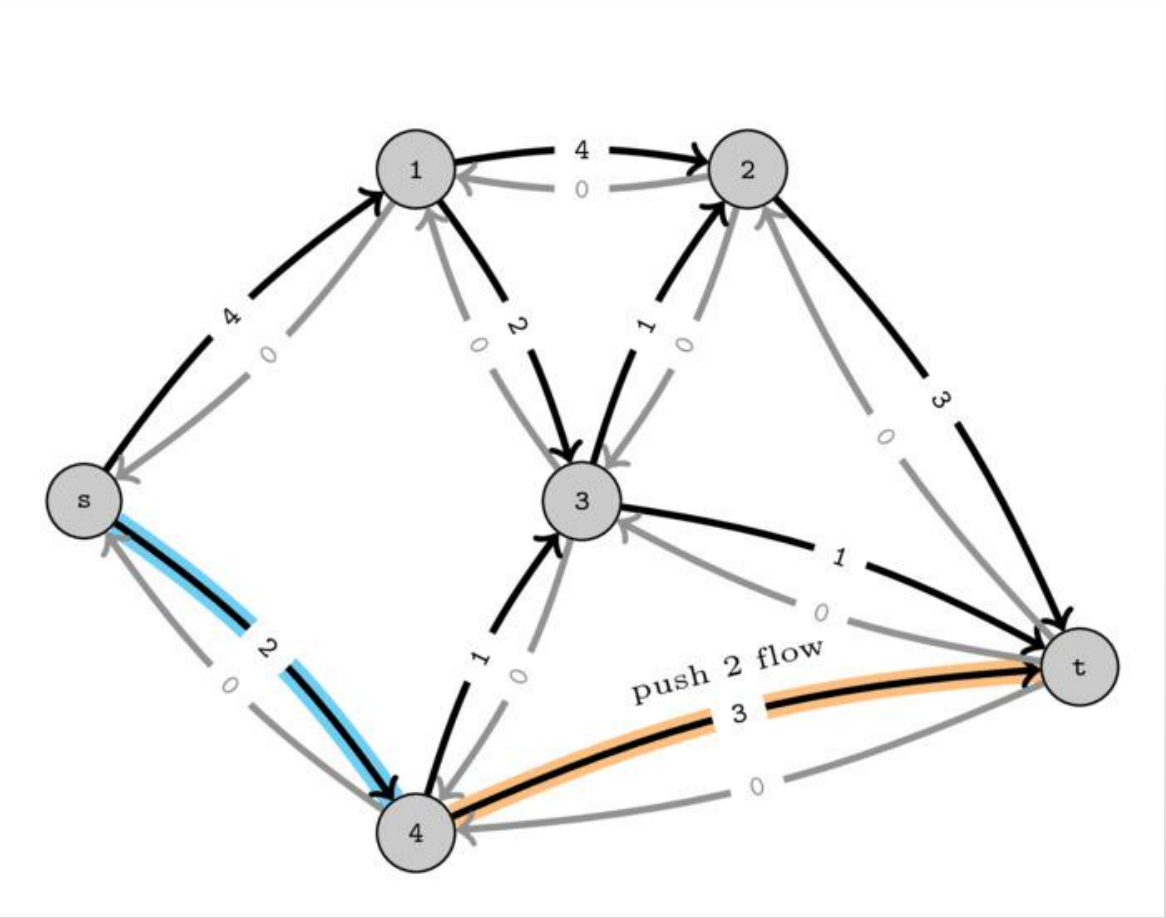
# Edmonds Karp



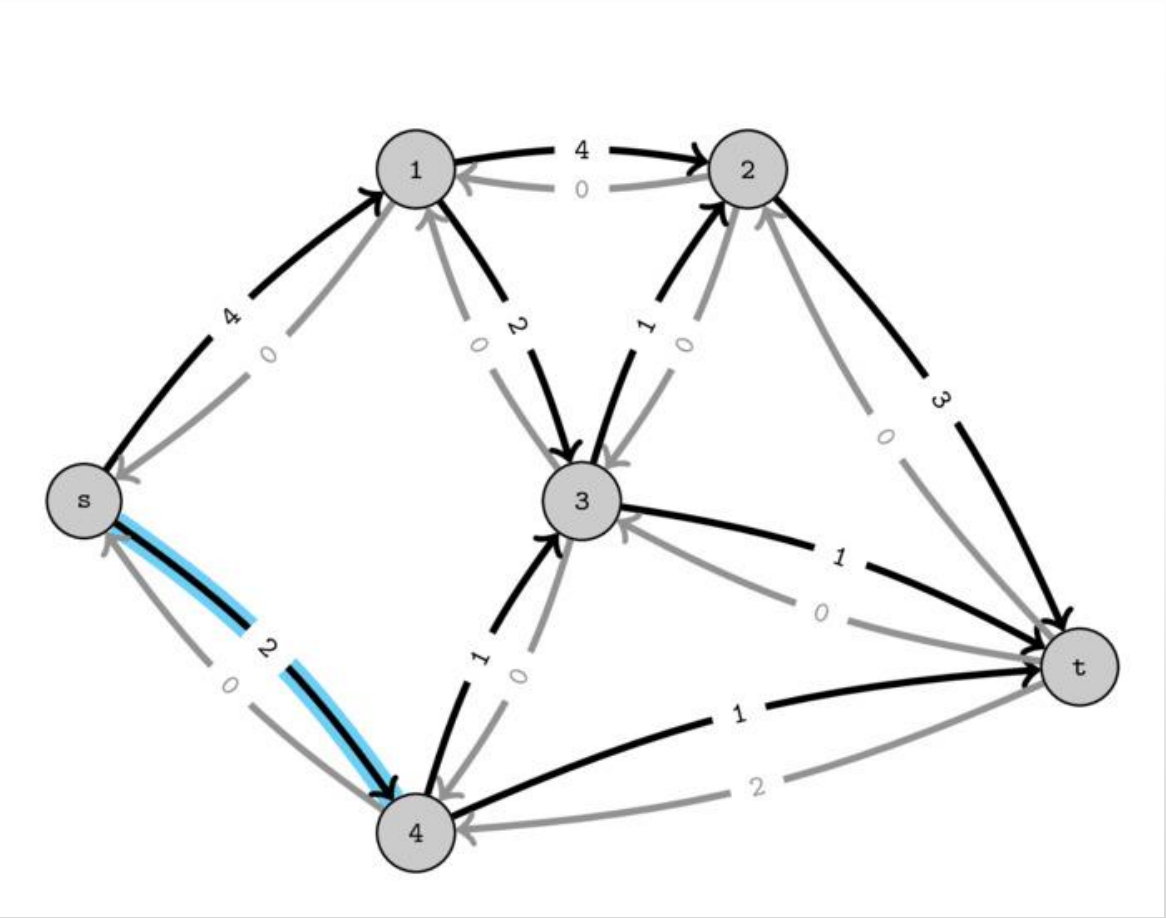
# Edmonds Karp



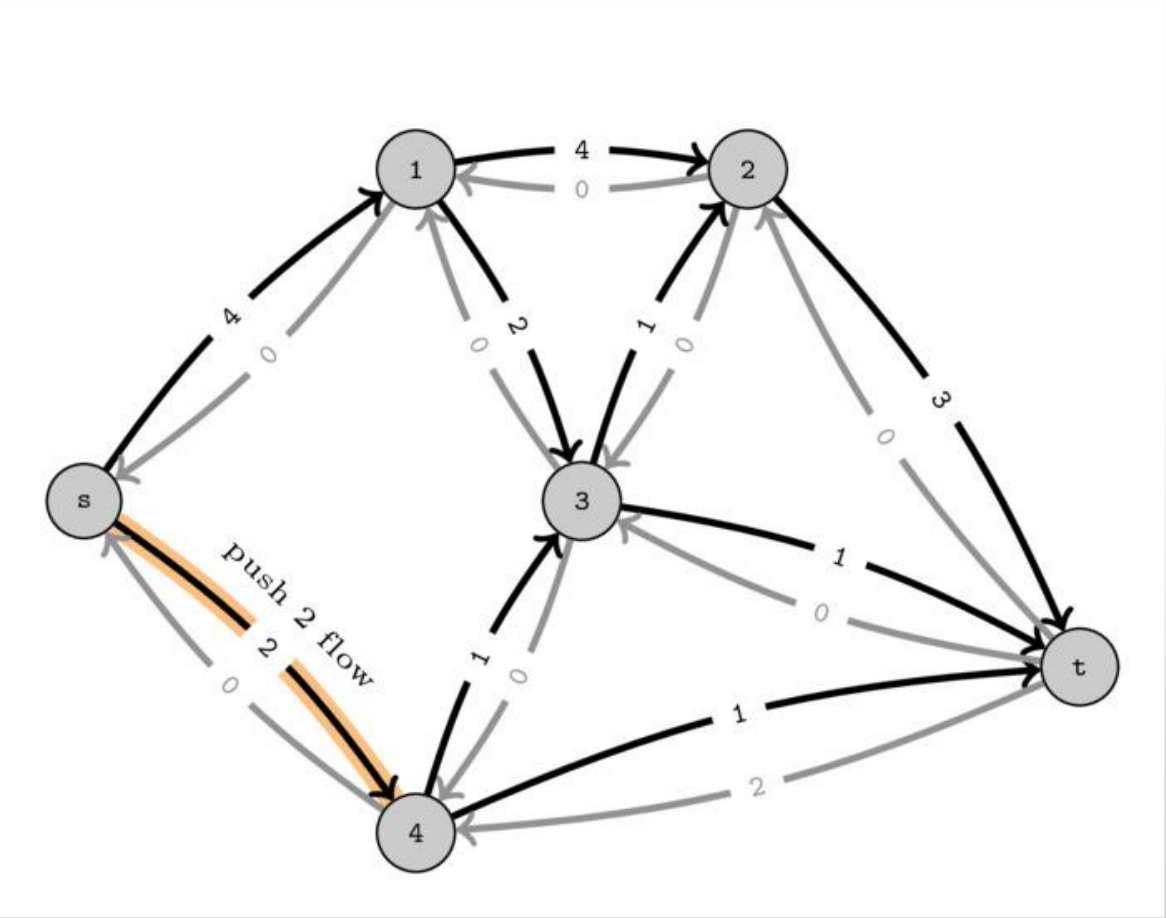
# Edmonds Karp



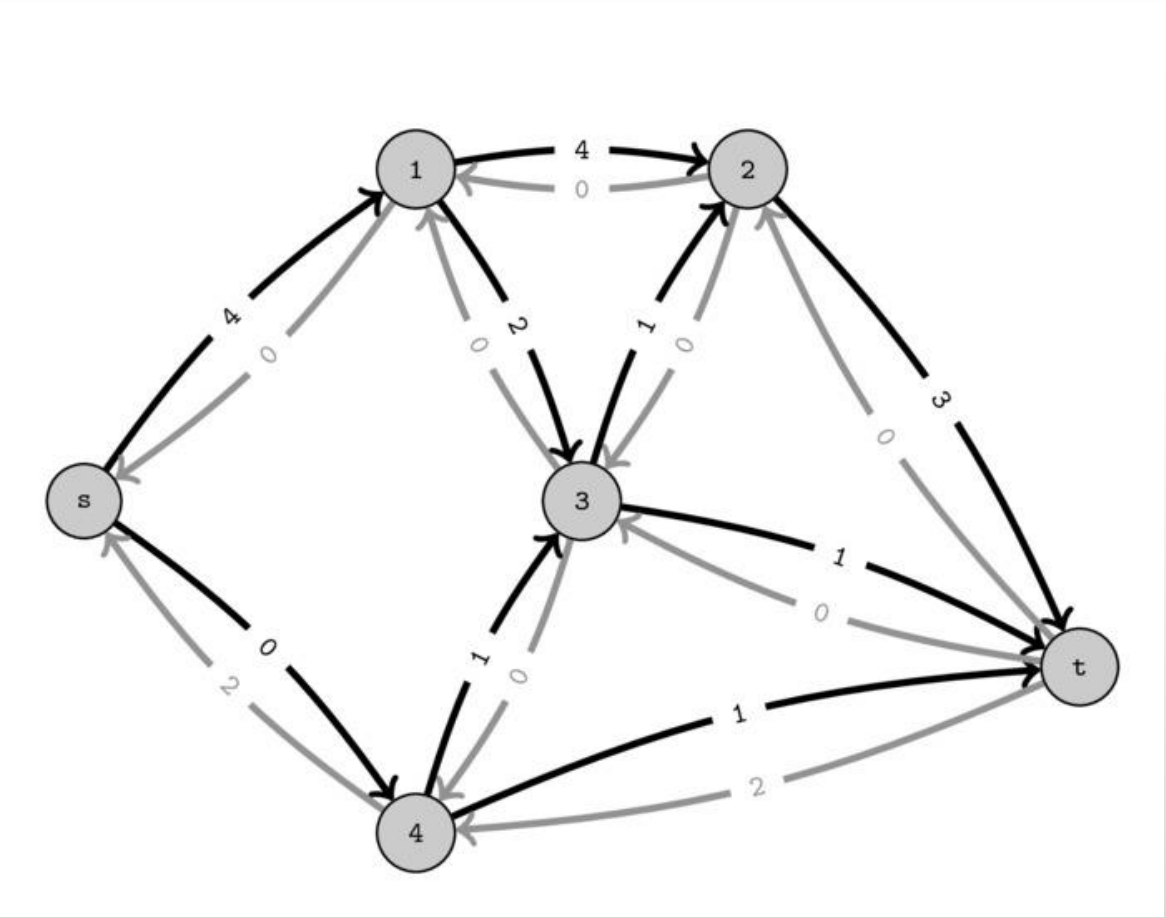
# Edmonds Karp



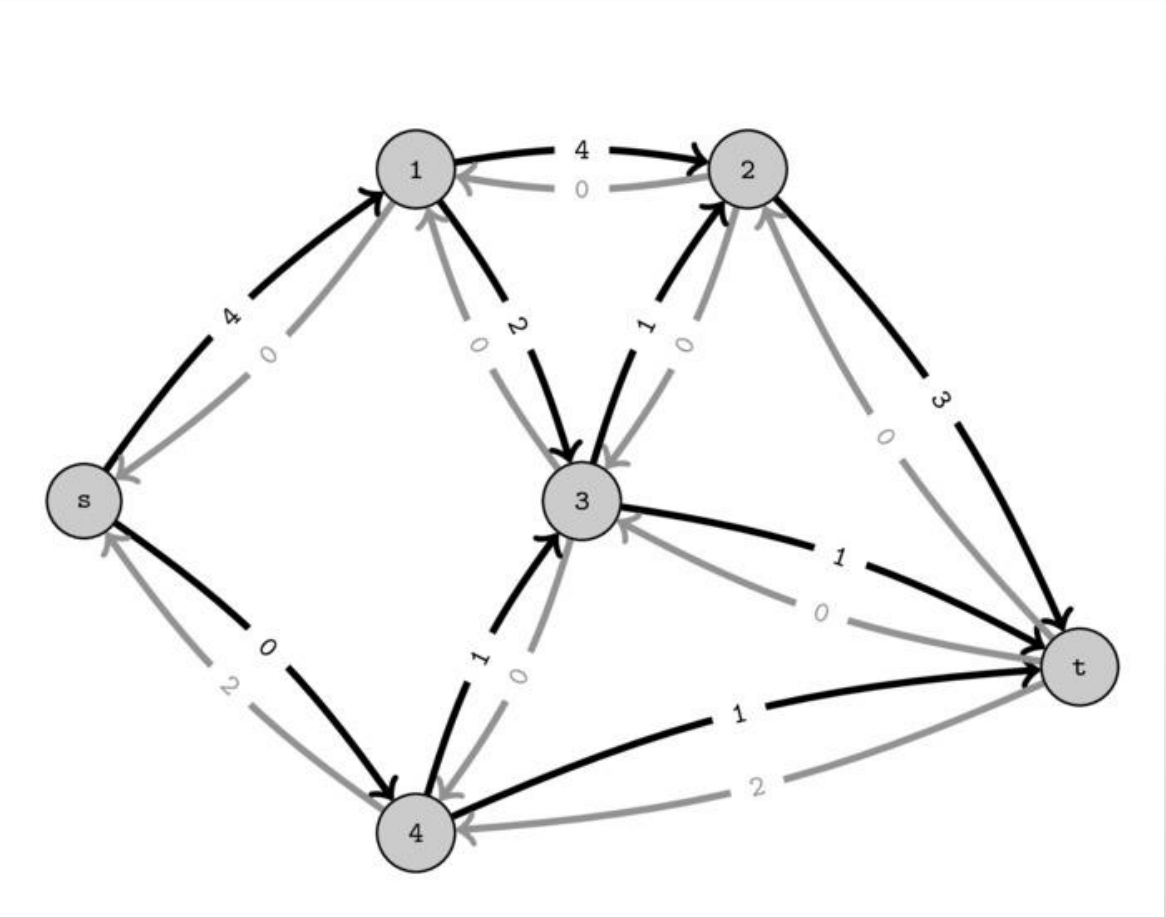
# Edmonds Karp



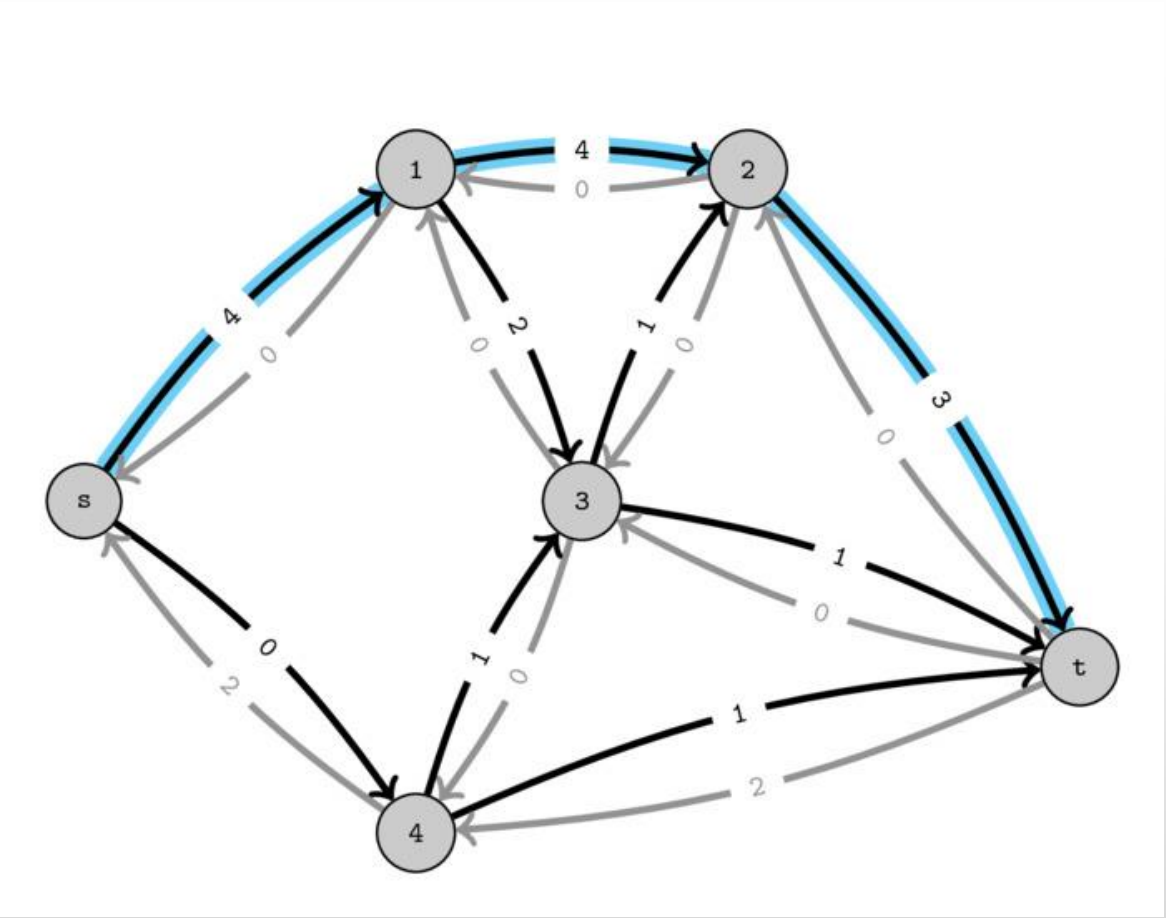
# Edmonds Karp



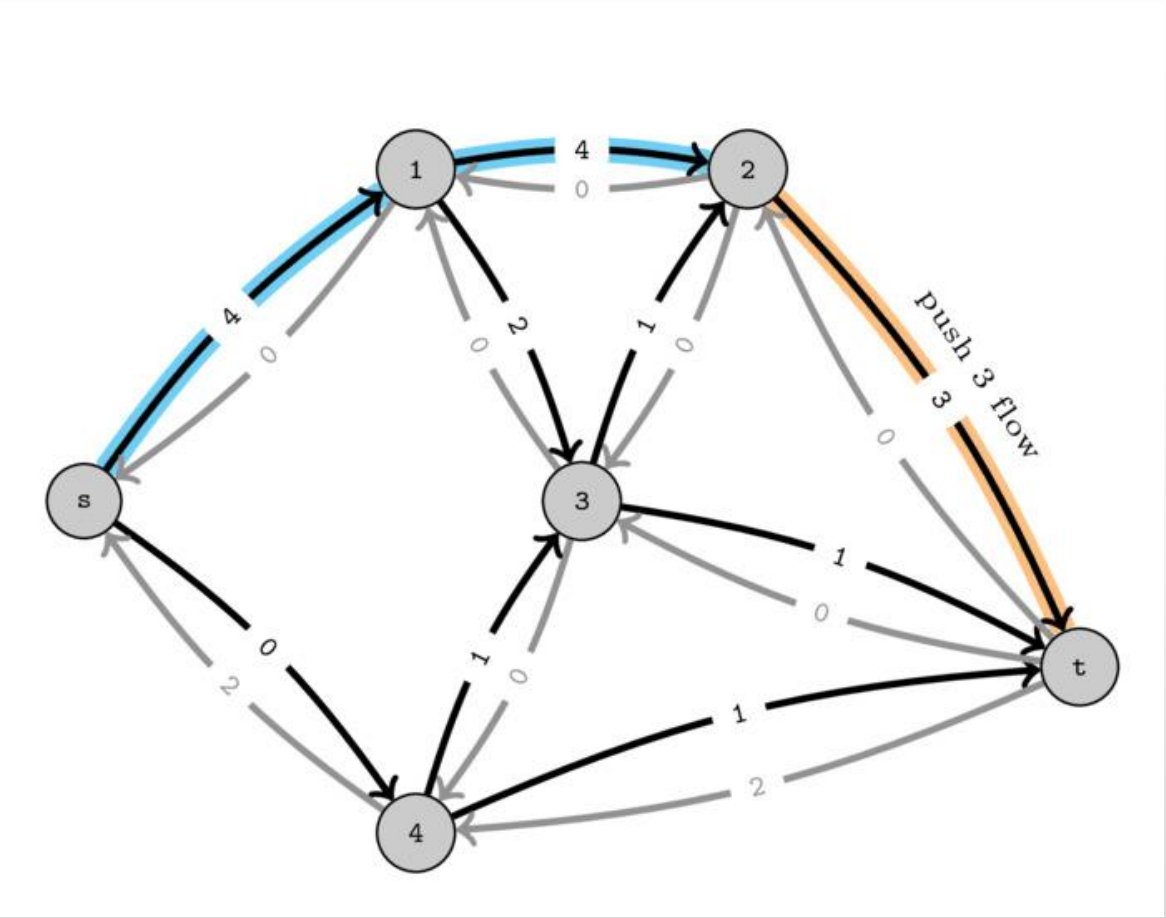
# Edmonds Karp



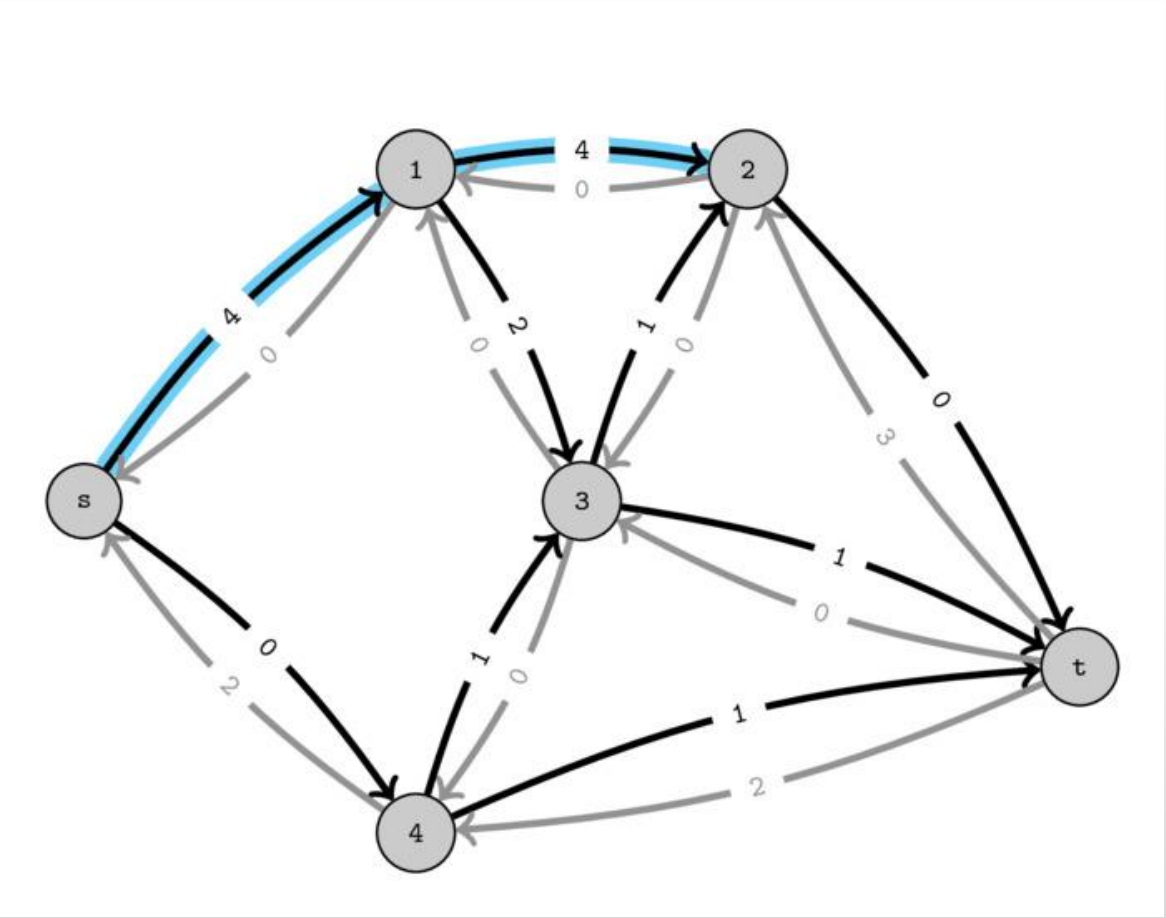
# Edmonds Karp



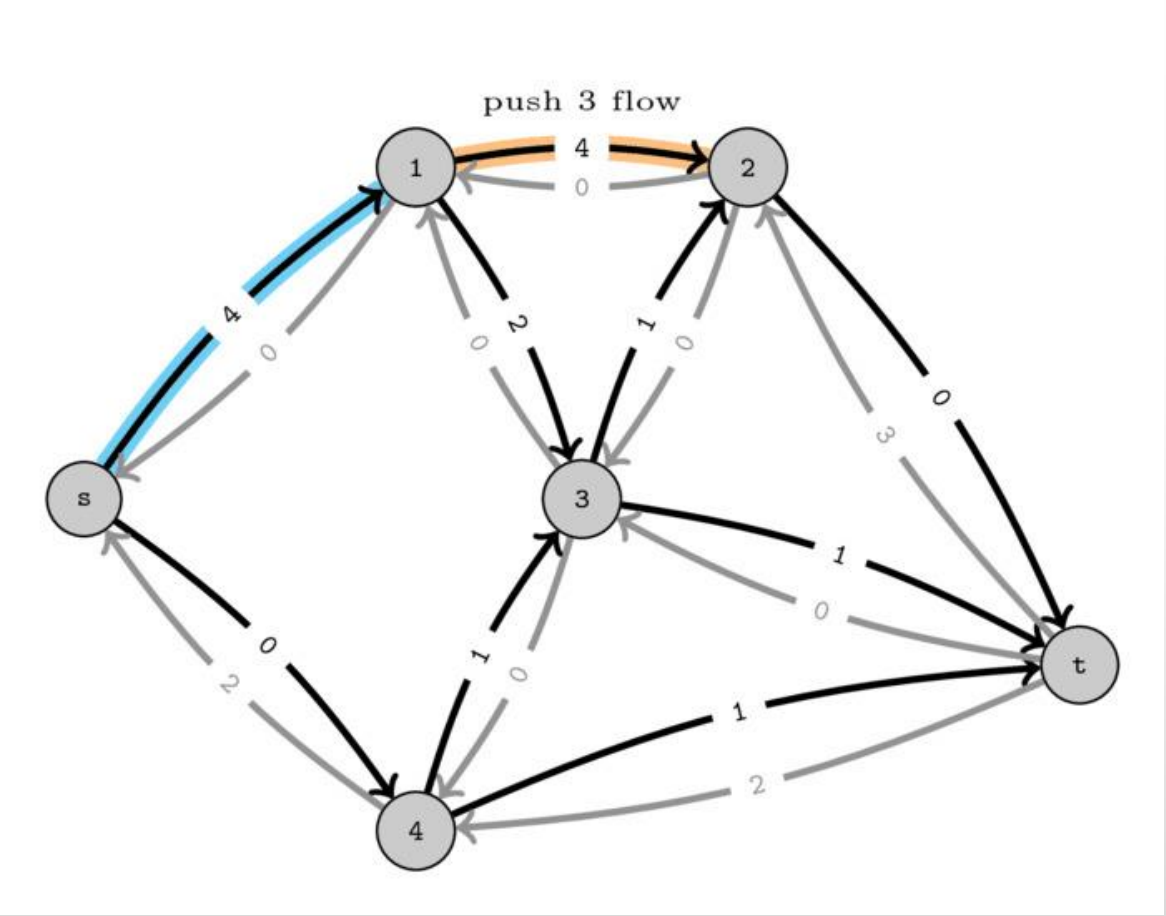
# Edmonds Karp



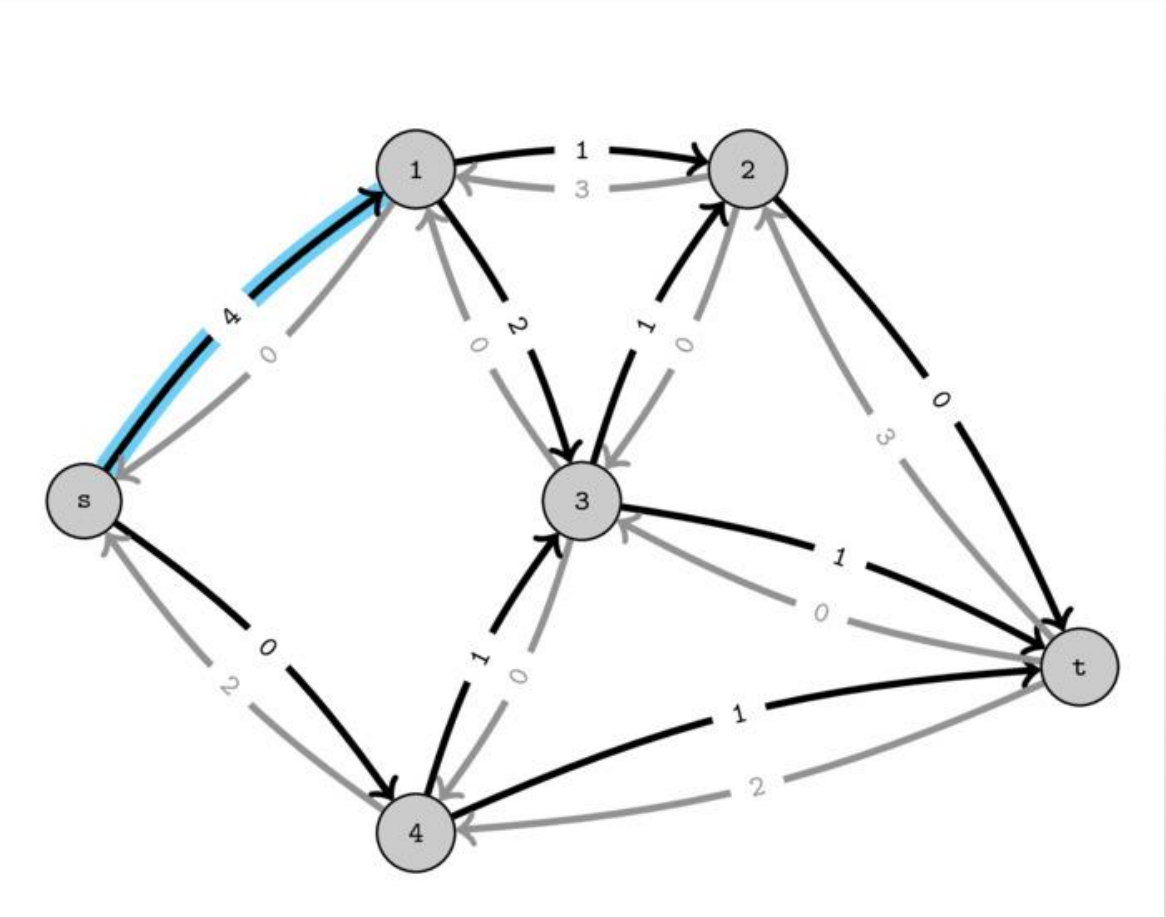
# Edmonds Karp



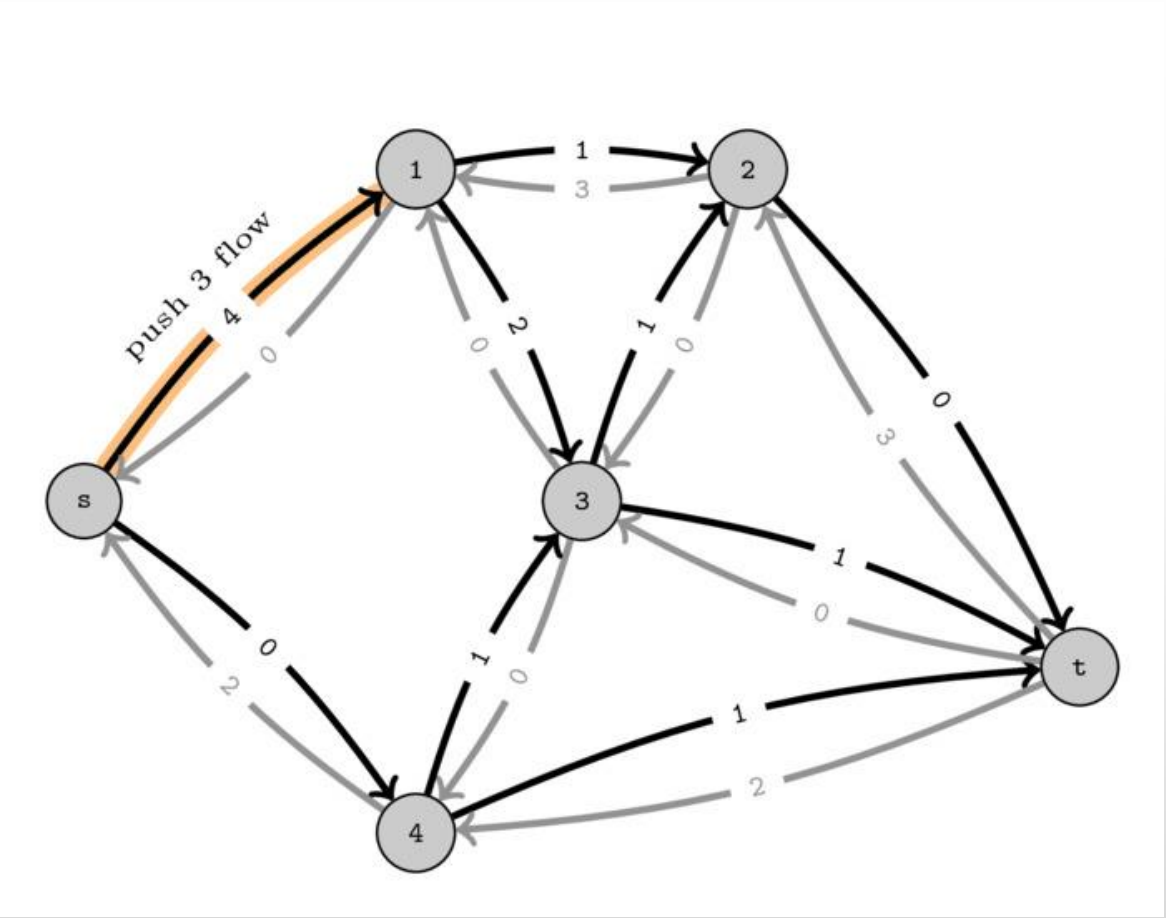
# Edmonds Karp



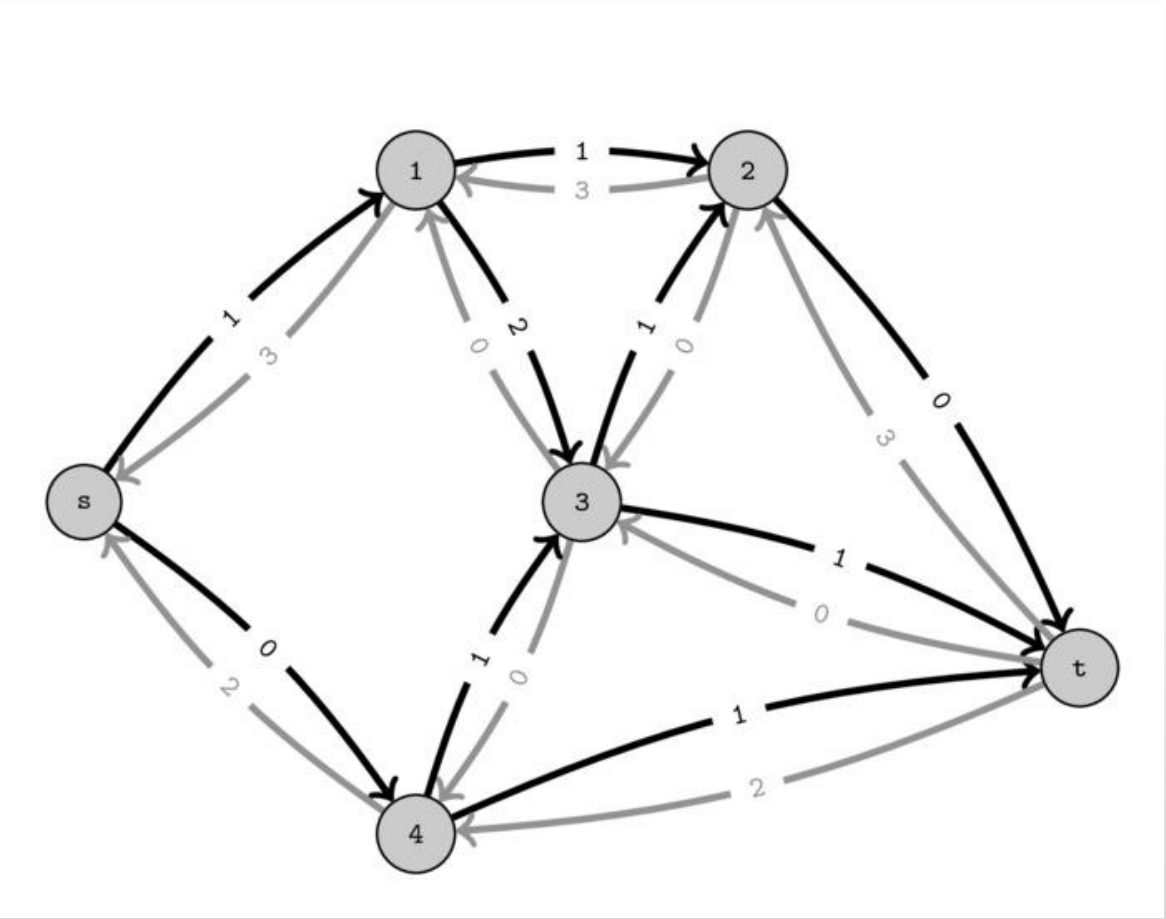
# Edmonds Karp



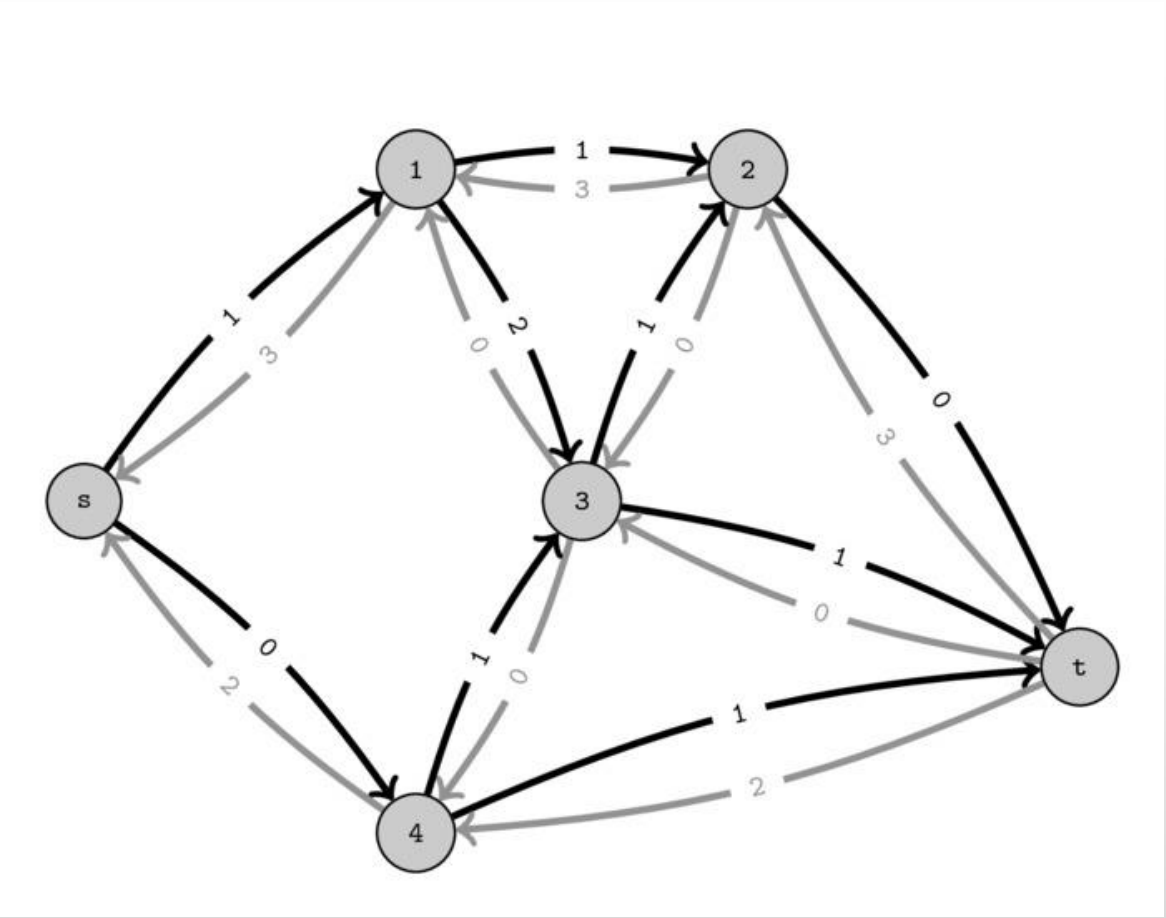
# Edmonds Karp



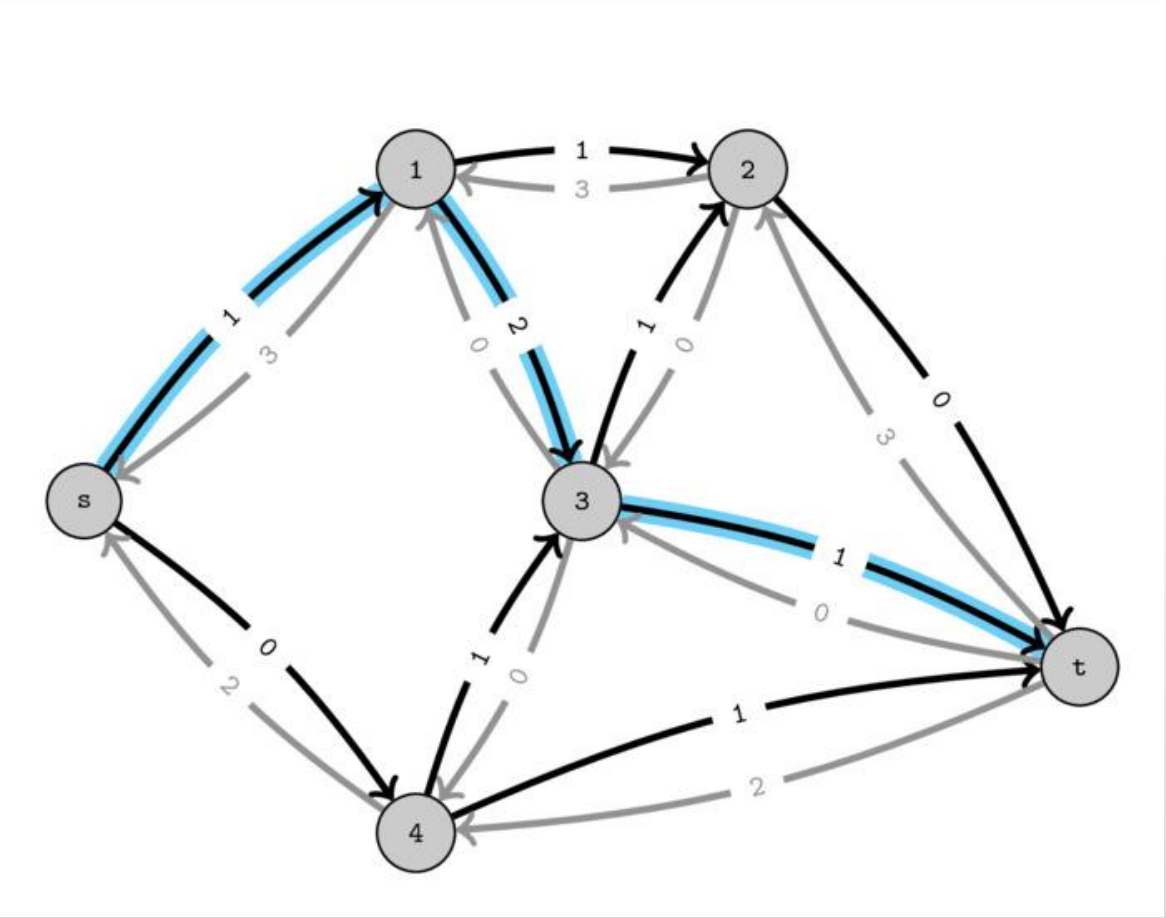
# Edmonds Karp



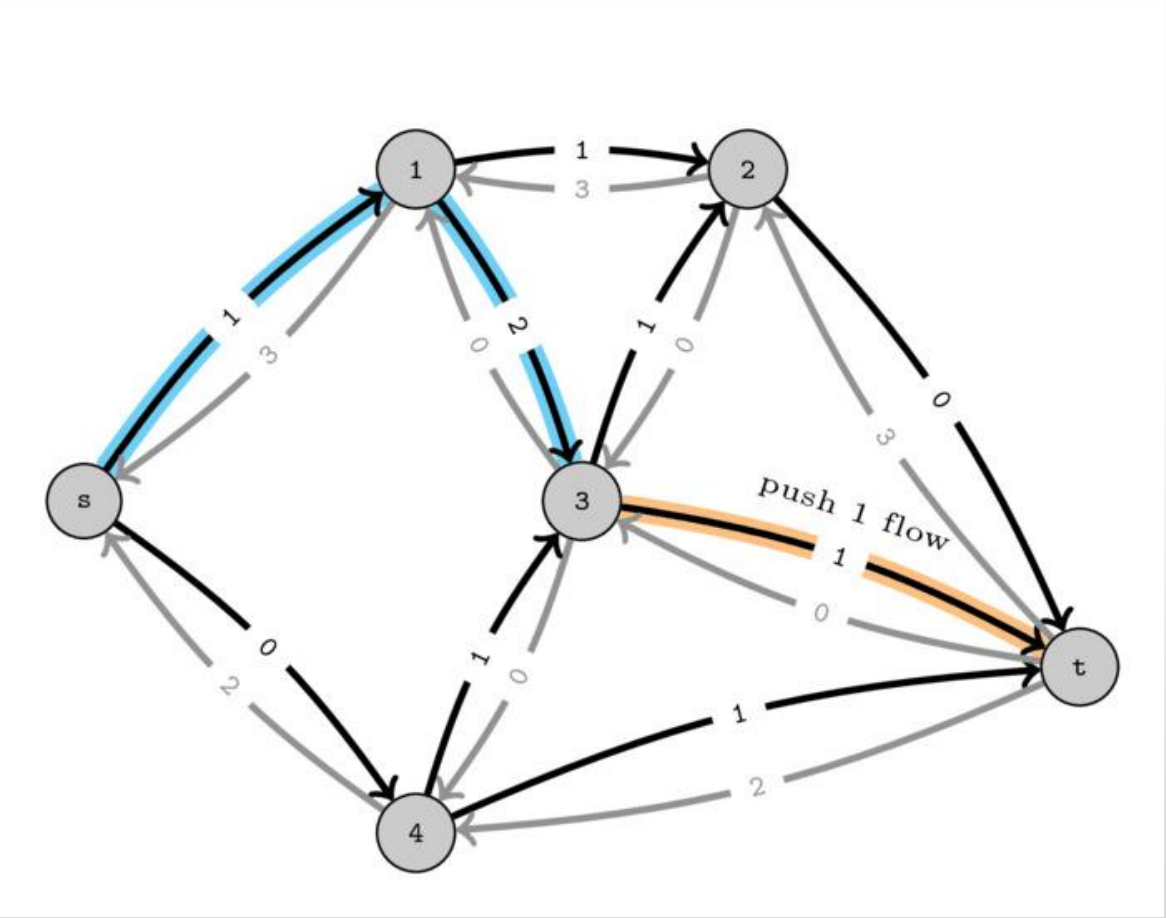
# Edmonds Karp



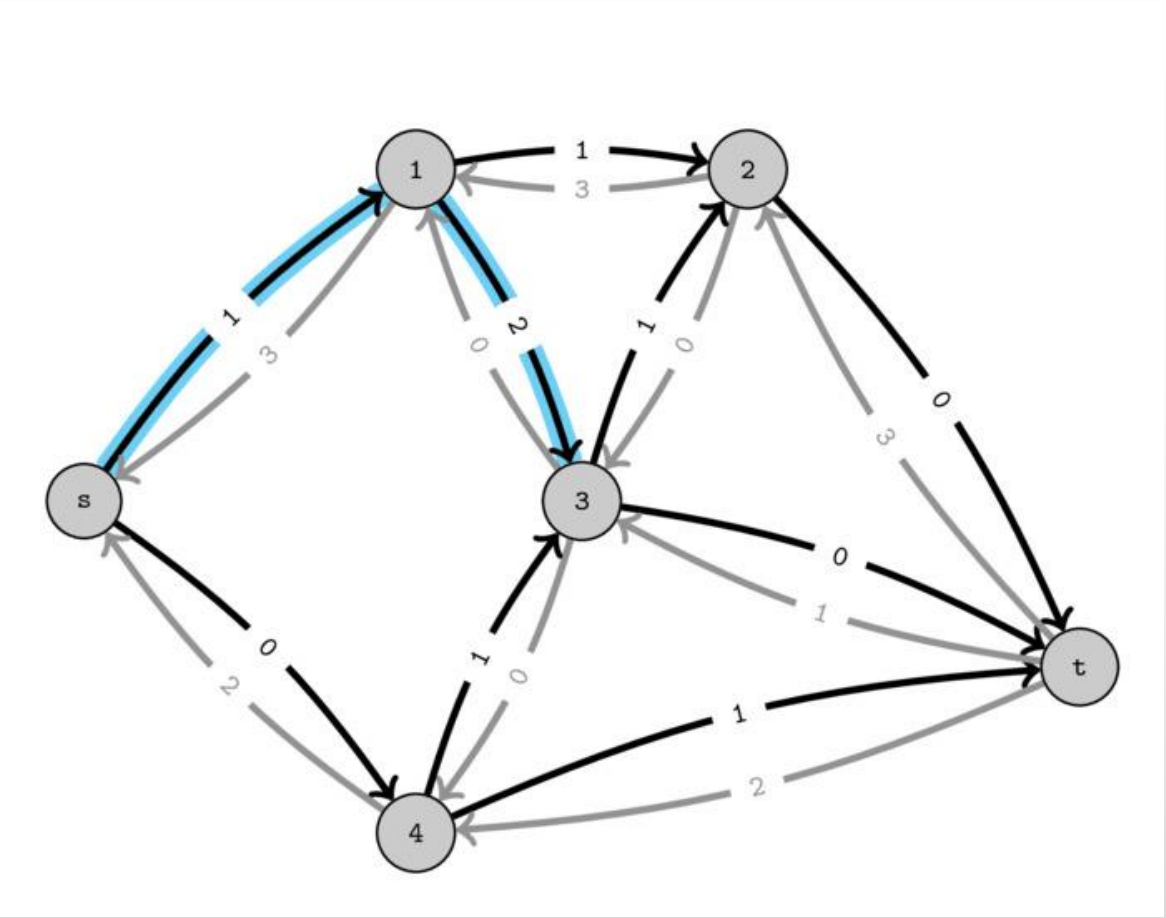
# Edmonds Karp



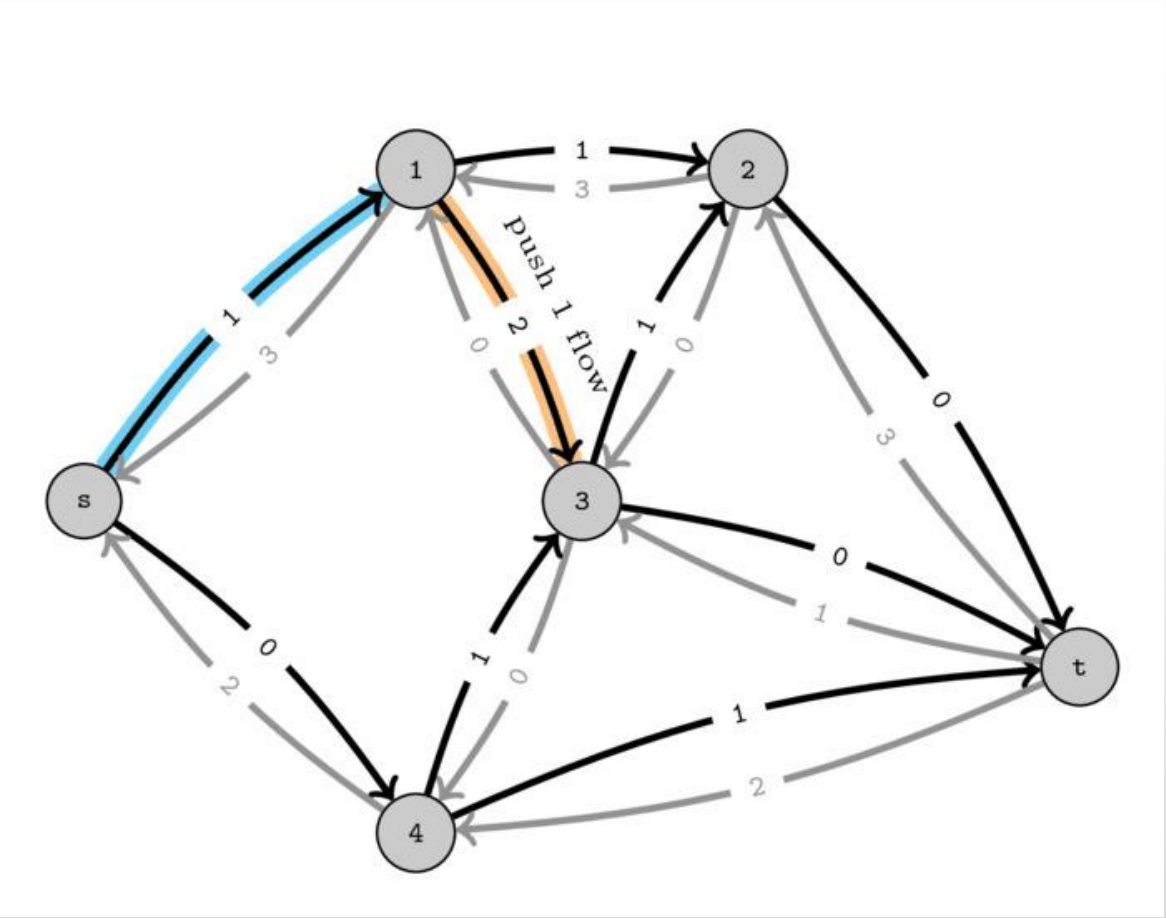
# Edmonds Karp



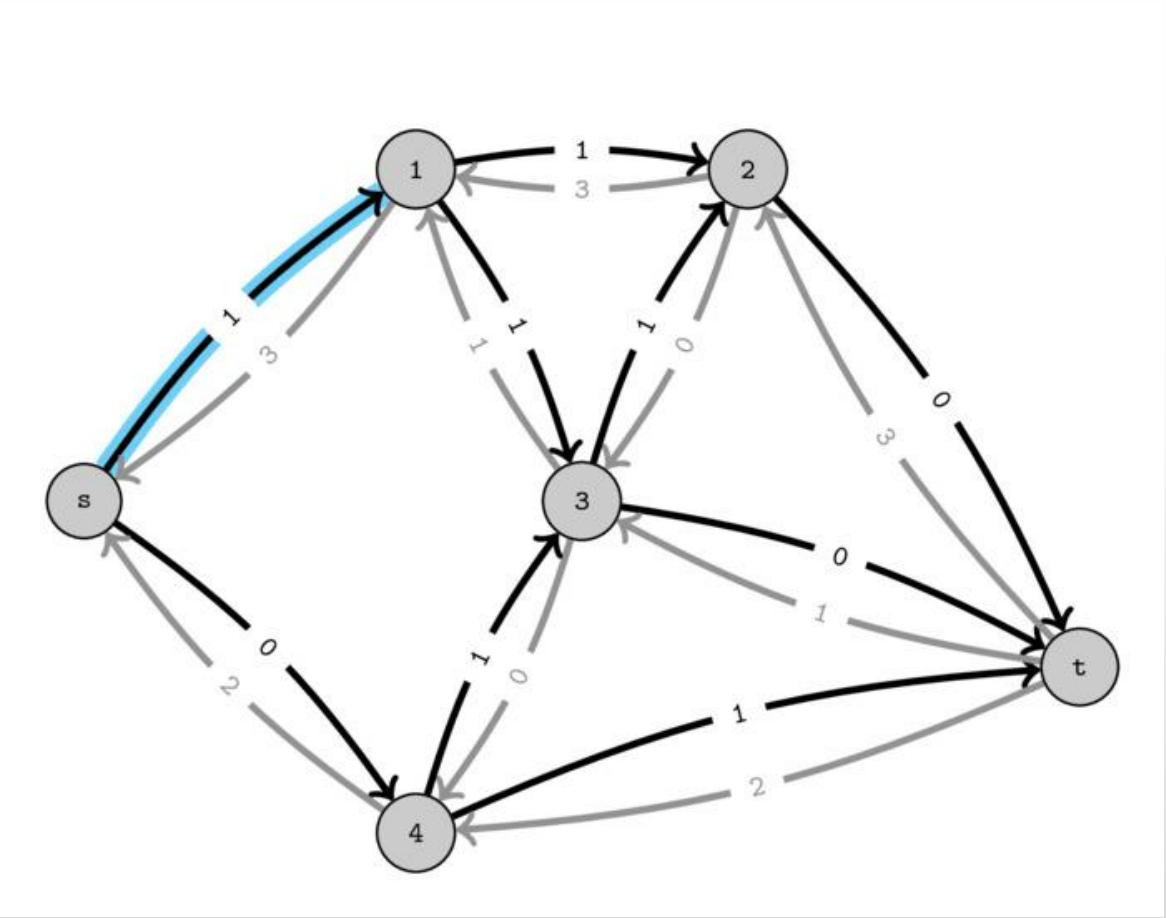
# Edmonds Karp



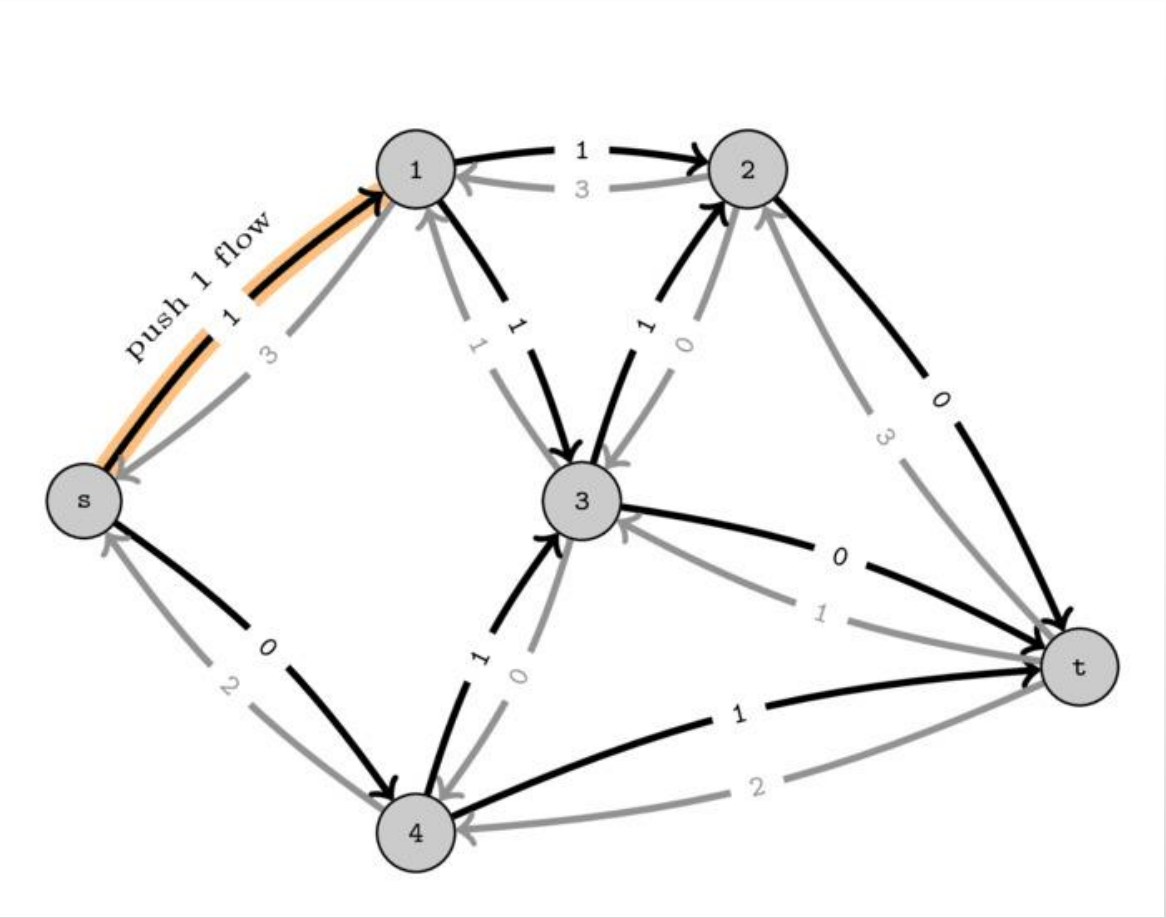
# Edmonds Karp



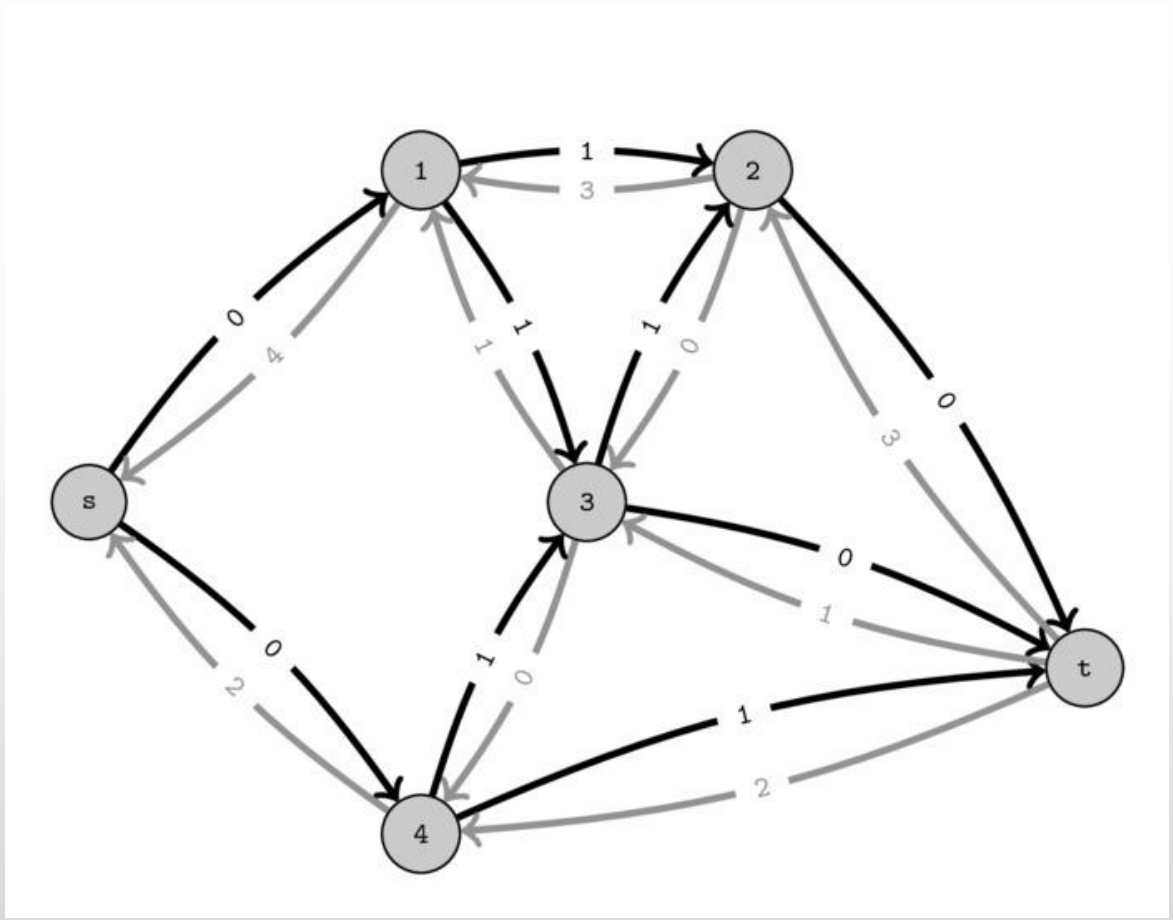
# Edmonds Karp



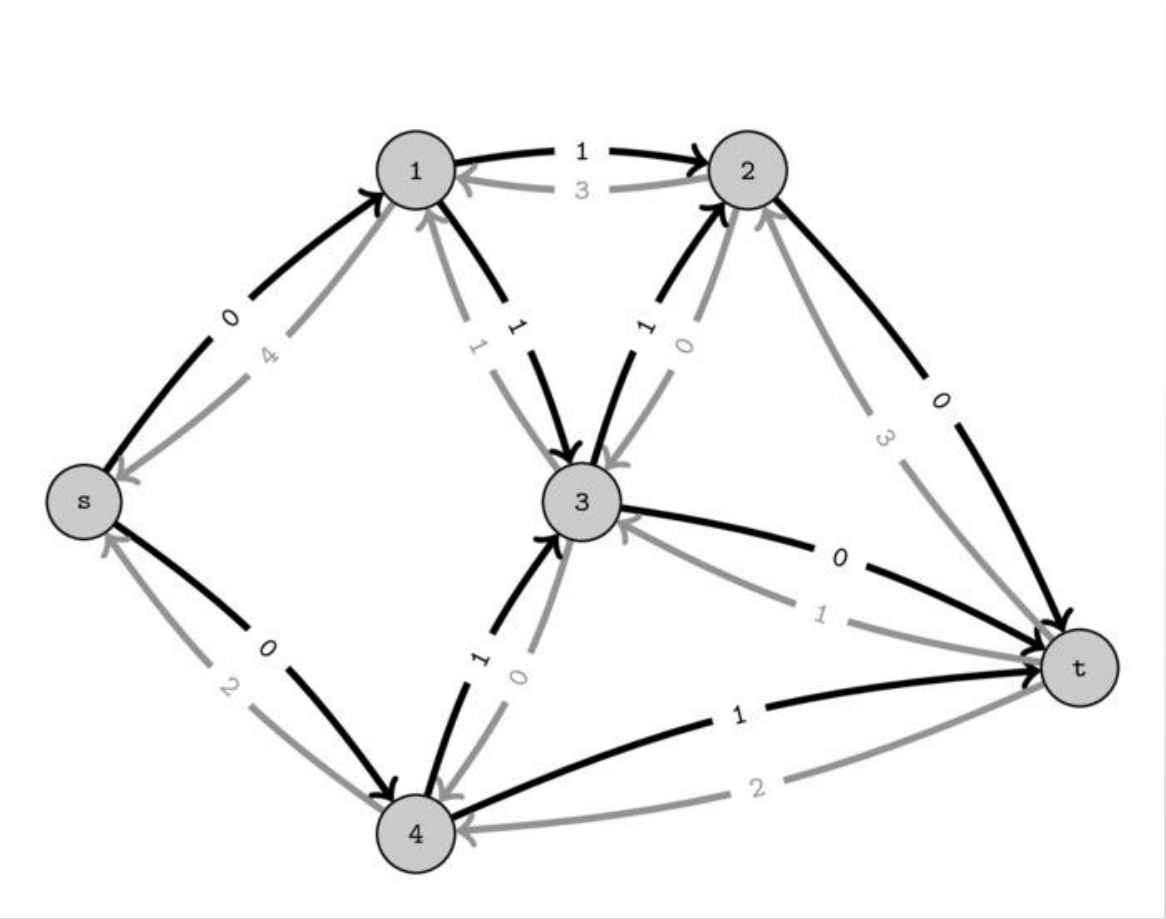
# Edmonds Karp



# Edmonds Karp

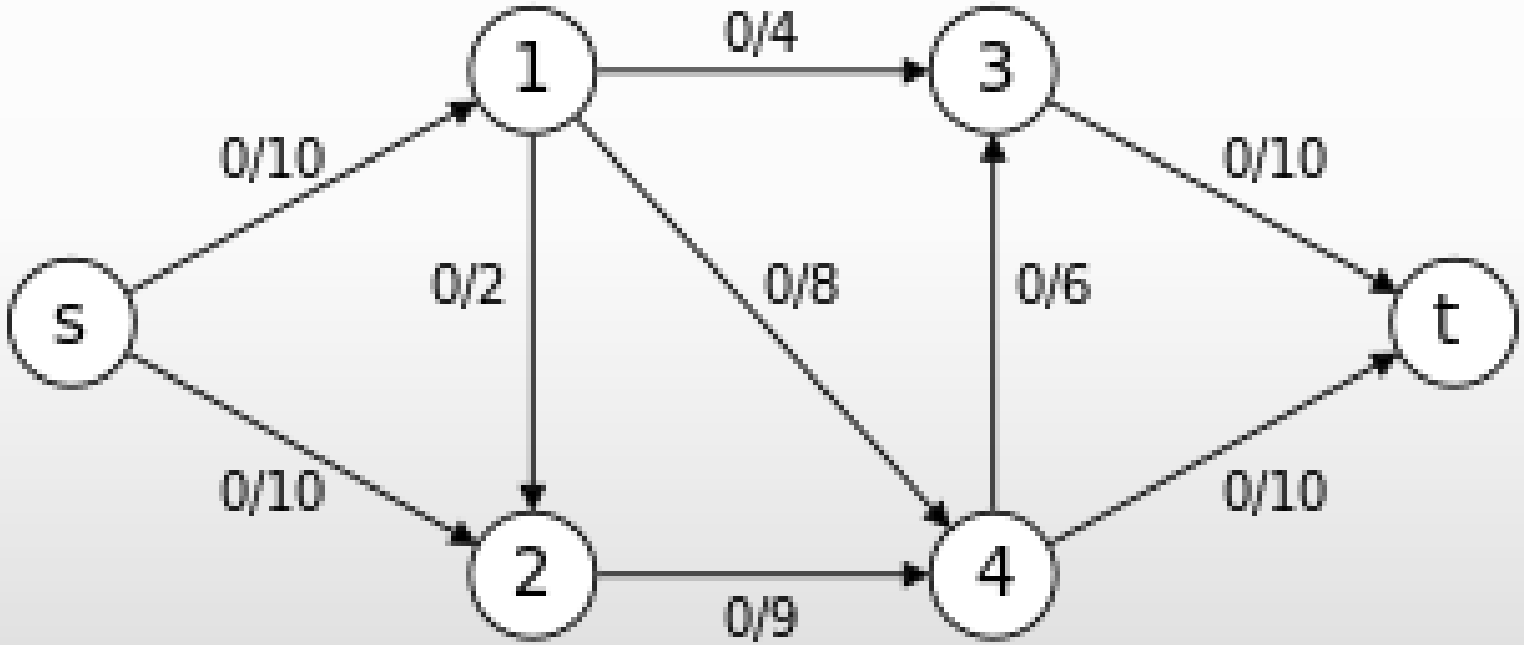


# Edmonds Karp

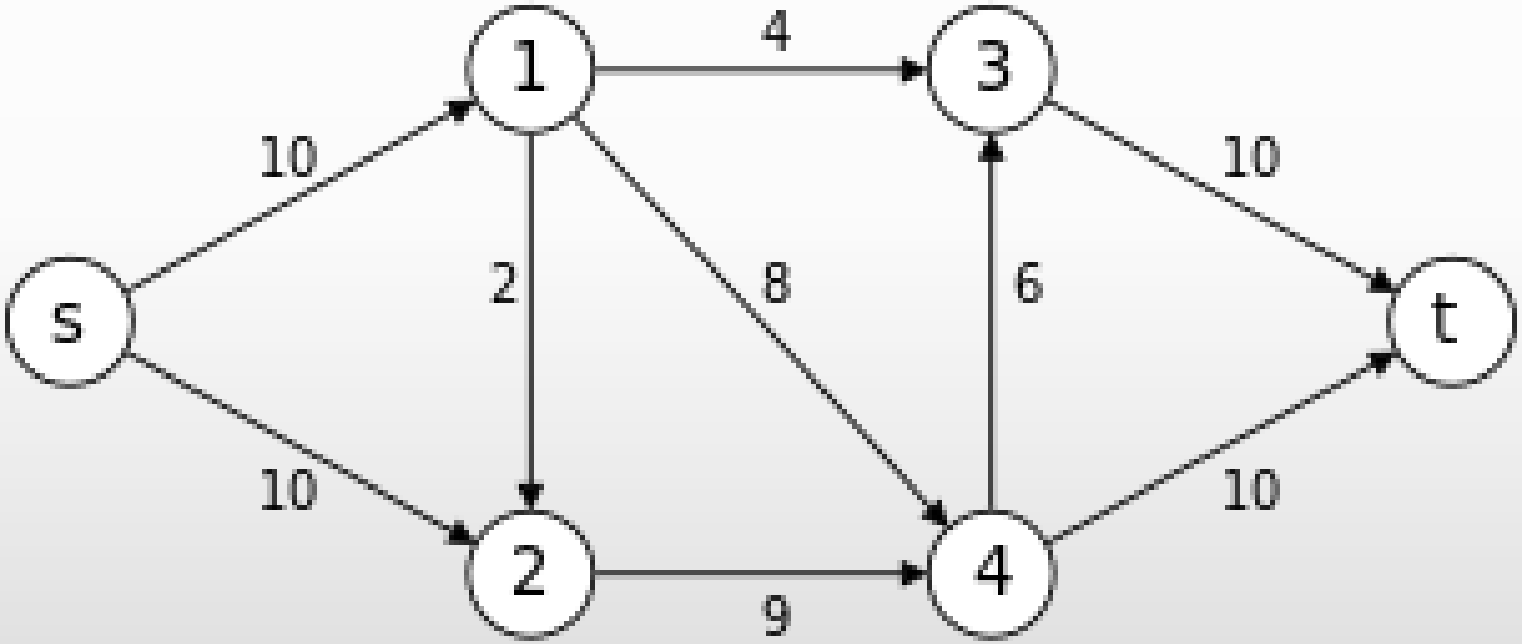


# Edmonds Karp

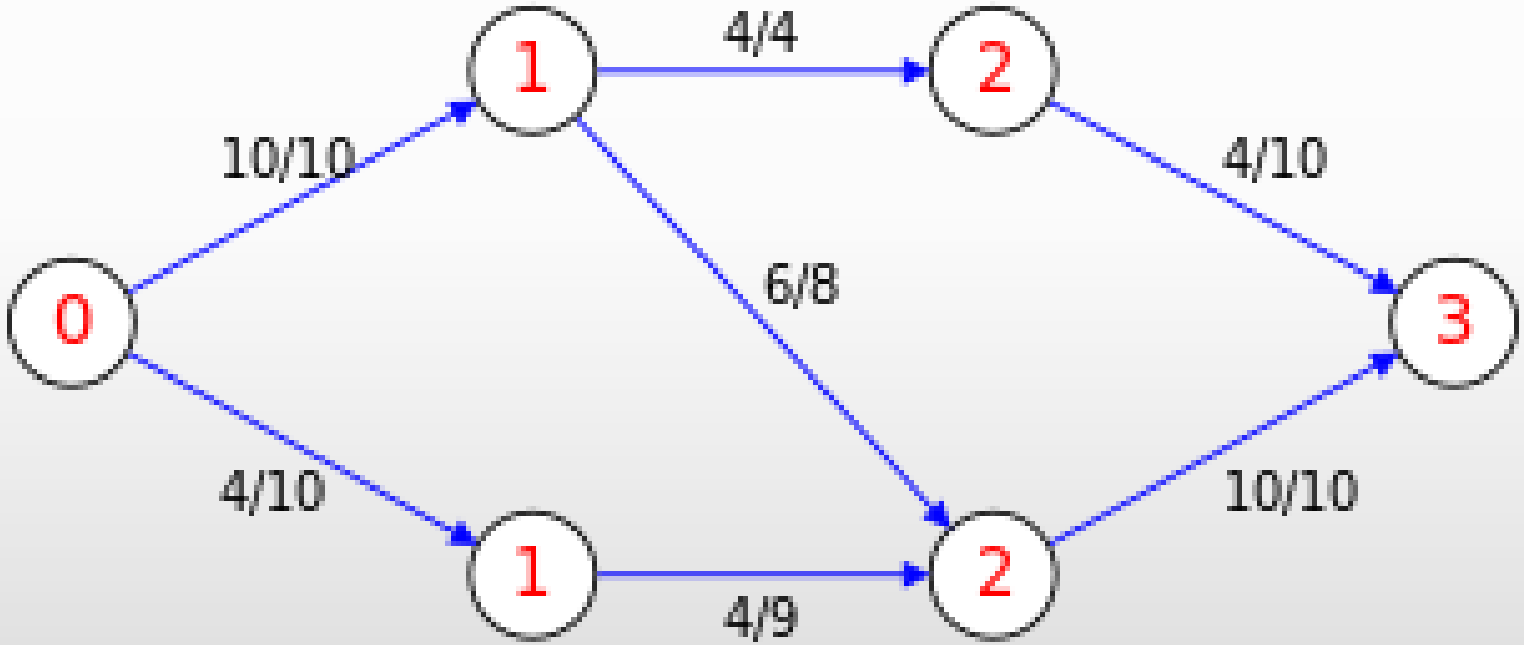
# Dinic's



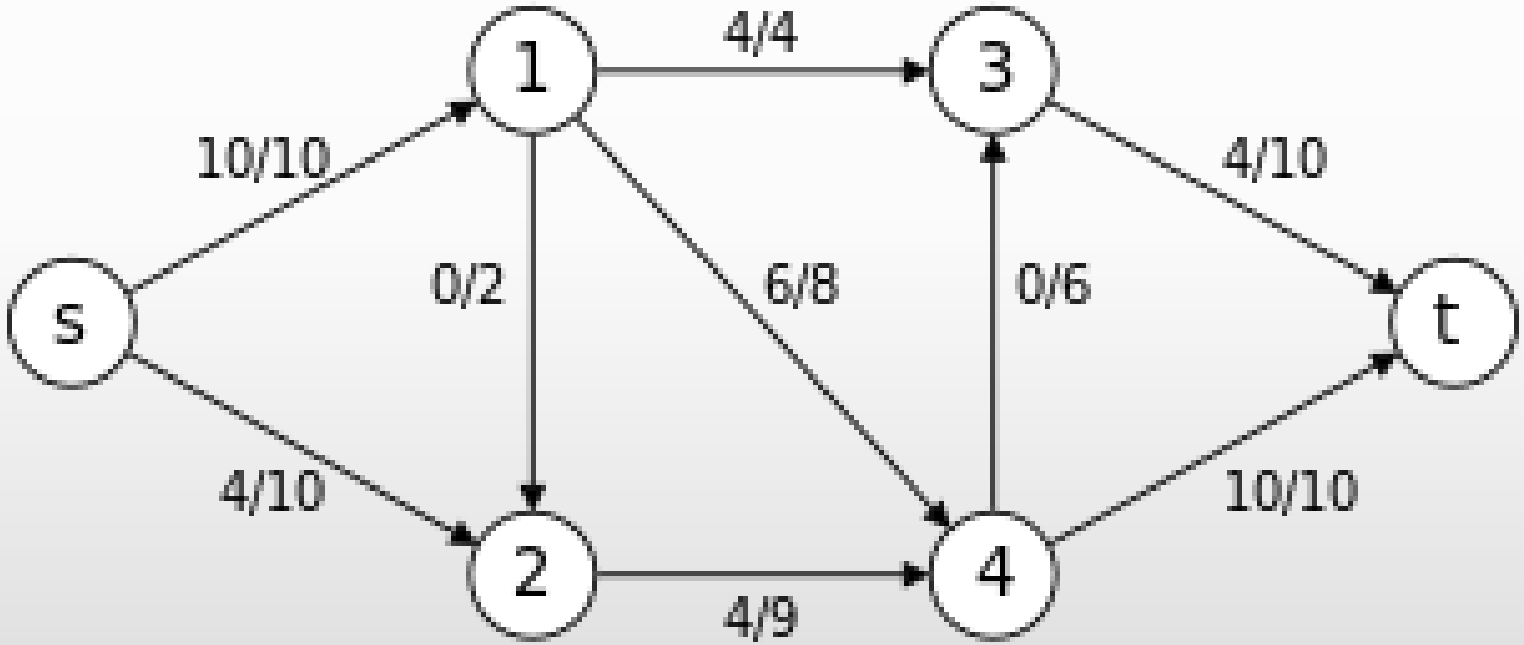
# Dinic's



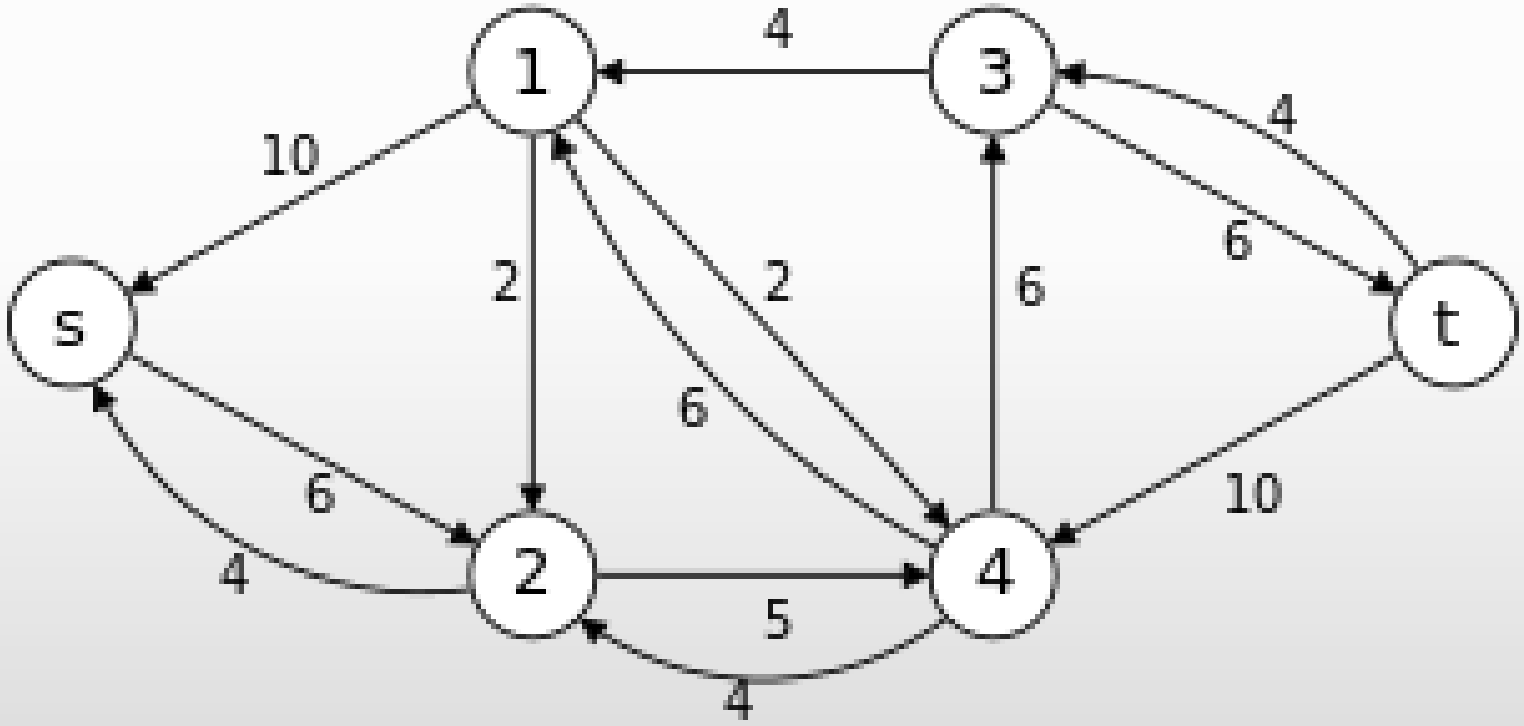
# Dinic's



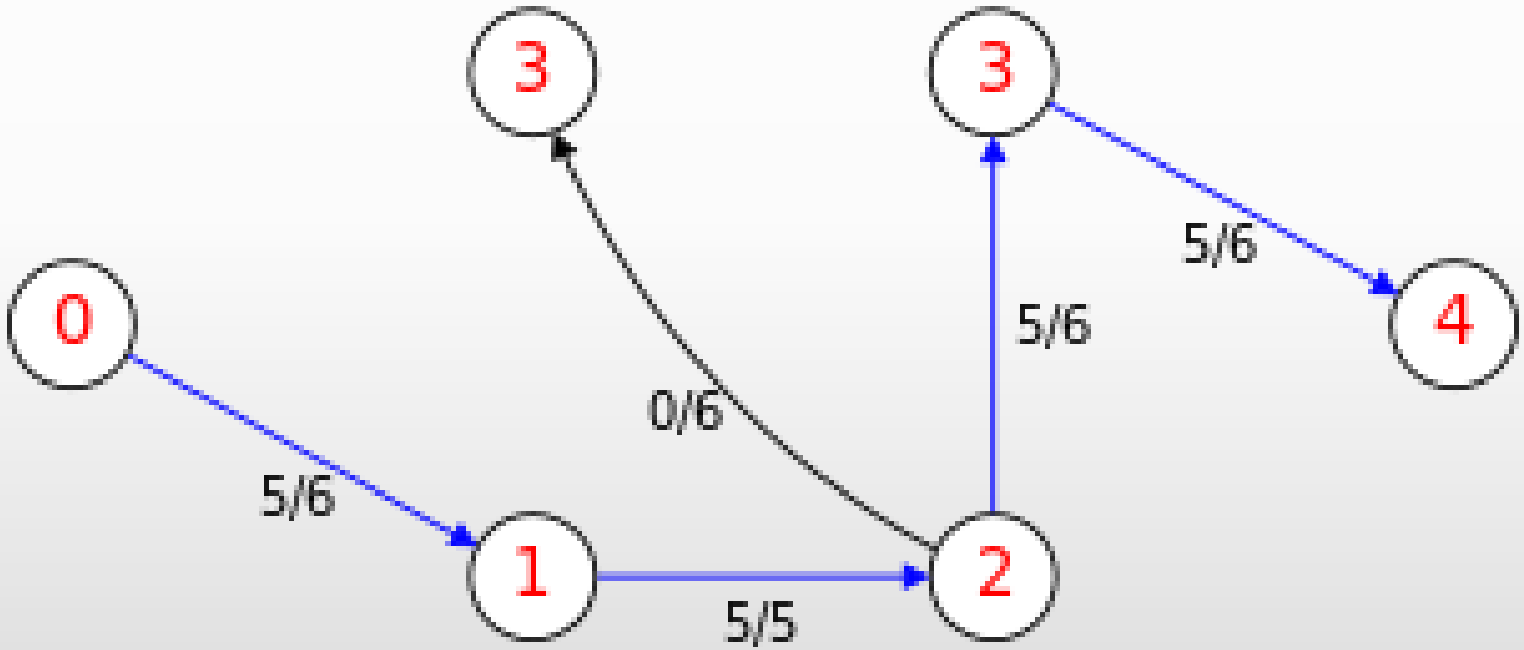
# Dinic's



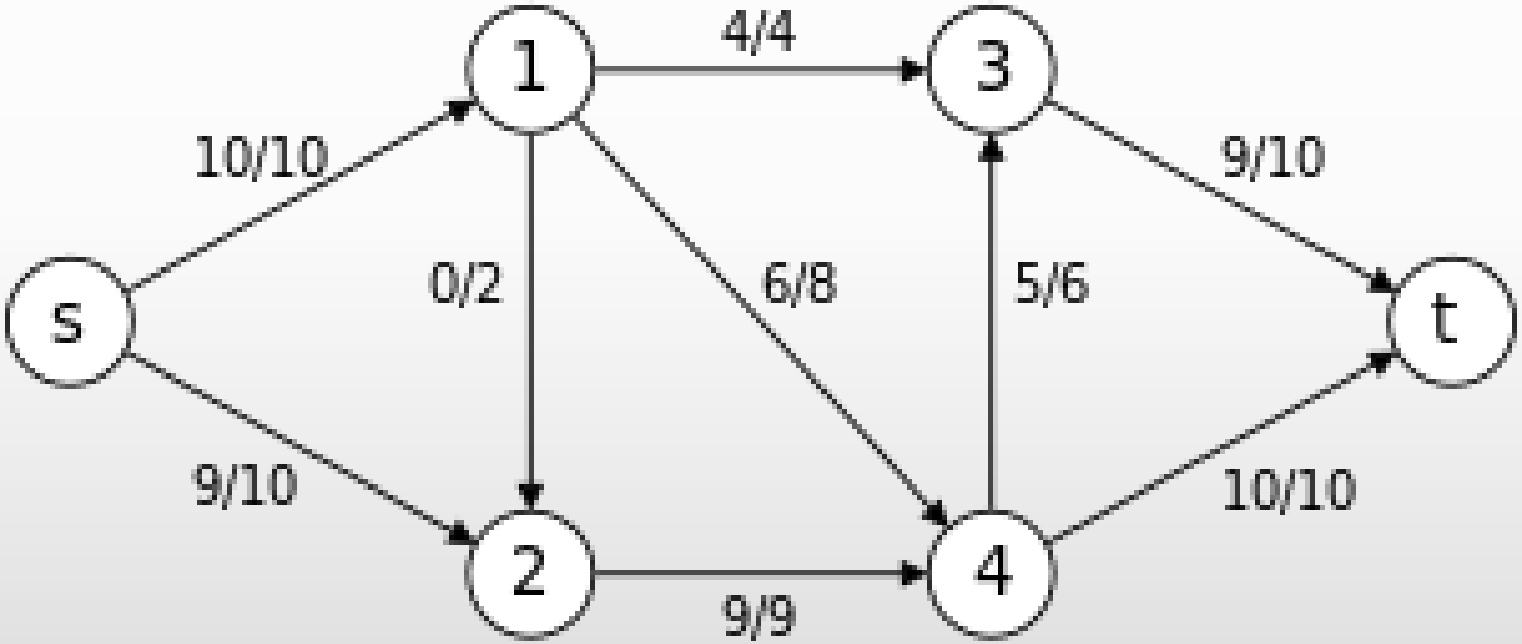
# Dinic's



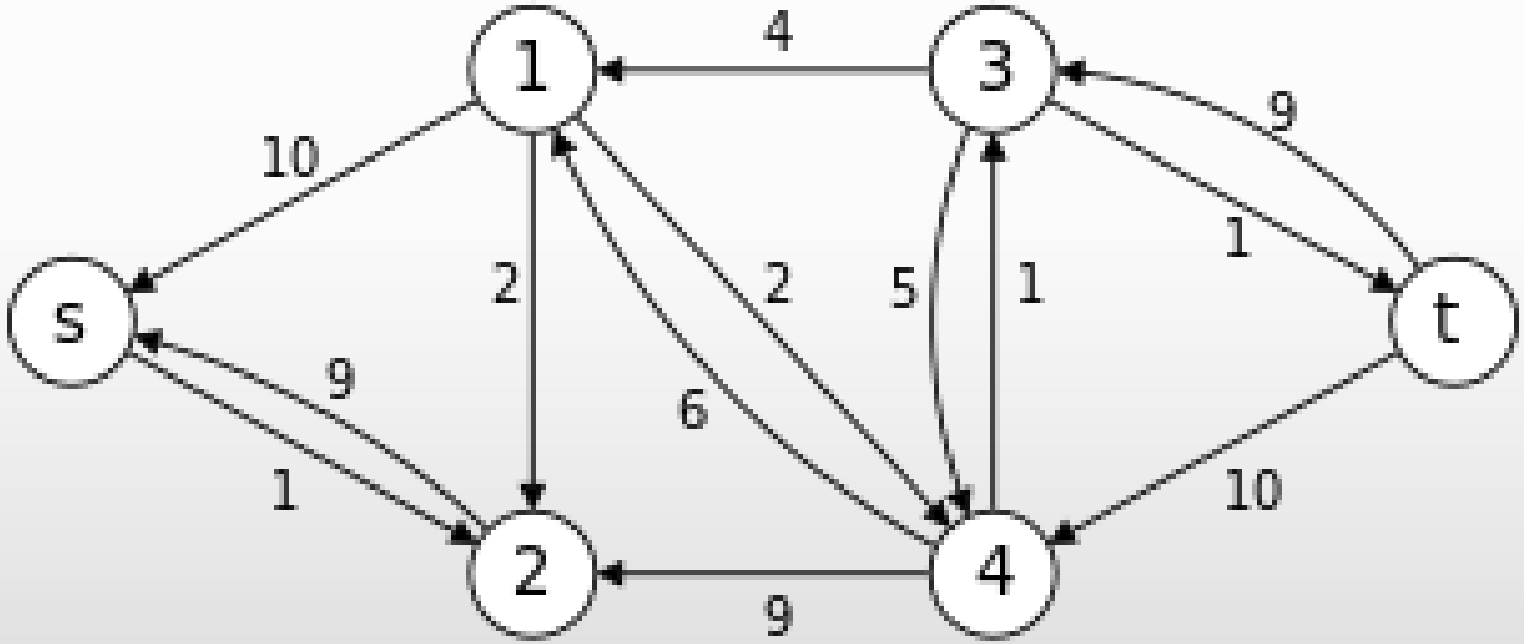
# Dinic's



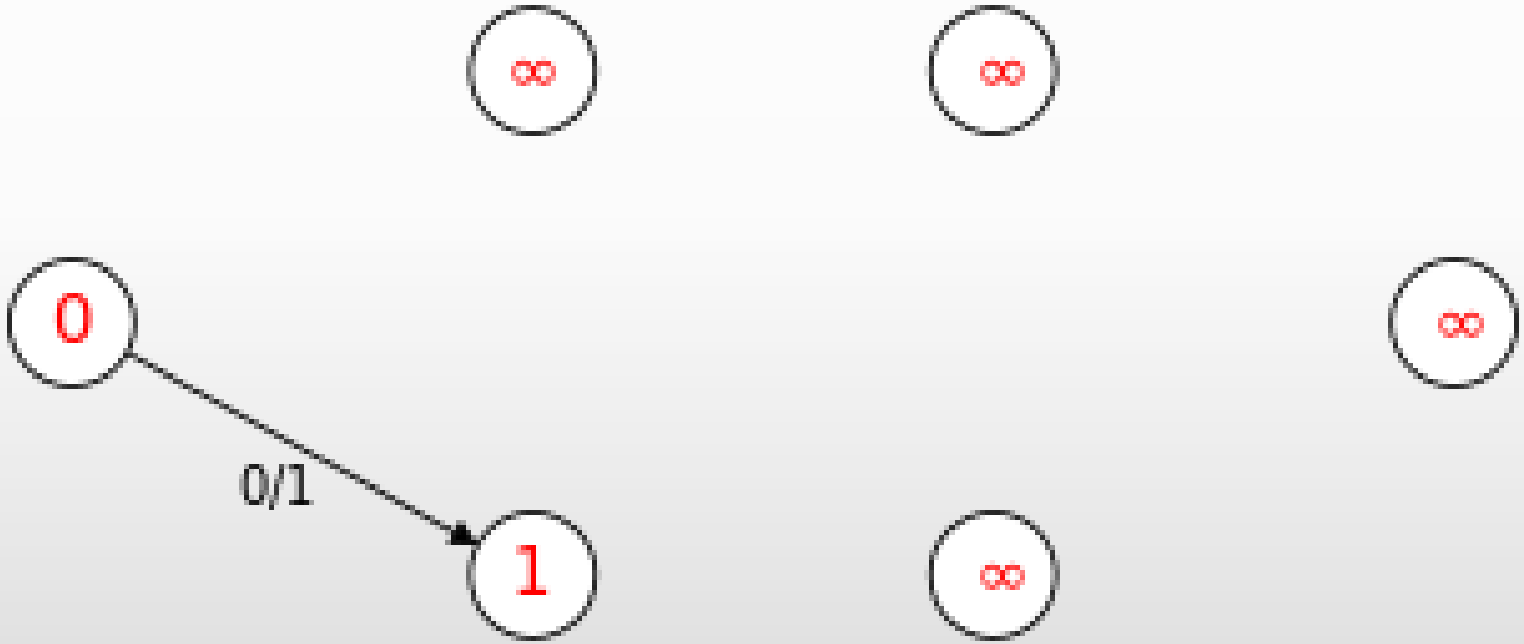
# Dinic's



# Dinic's



# Dinic's



SON