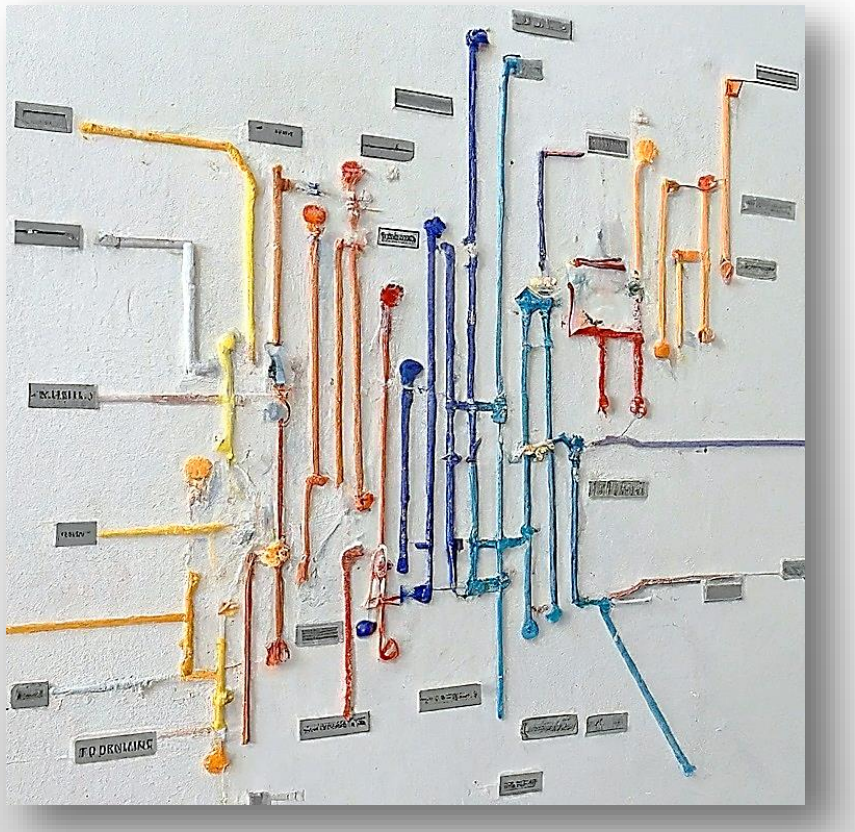
Bölüm 4: Çizge Algoritmaları

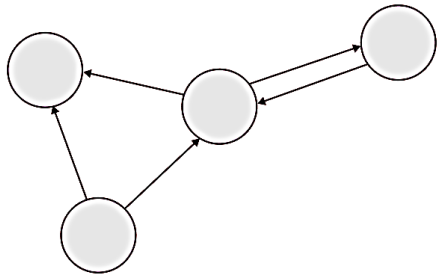
Algoritmalar

Çizge Algoritmaları

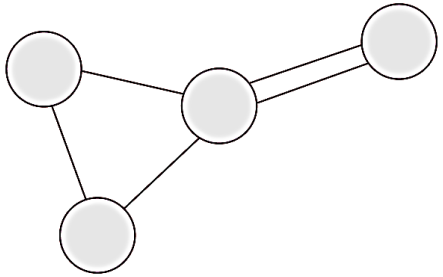
- Dünya aslında bir ağ gibidir.
  - Şehirler yollarla,
  - İnsanlar ilişkilerle,
  - Bilgisayarlar kablolarla birbirine bağlıdır.
- Çizge algoritmaları bu ağları inceler ve anlamlandırır.



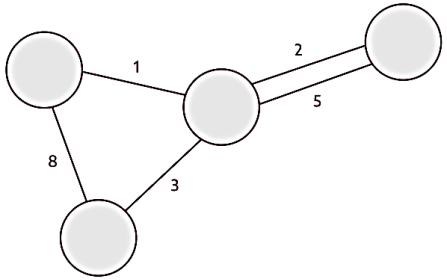
Çizge Türleri



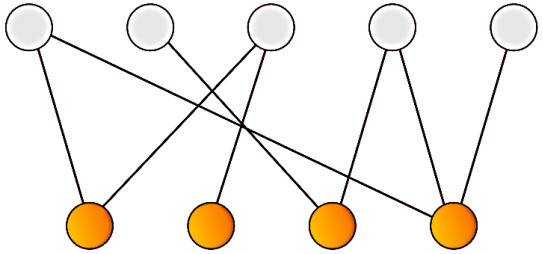
Directed graph



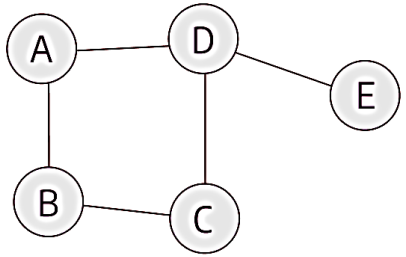
Undirected



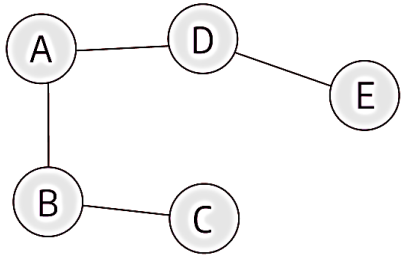
Weighted



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.

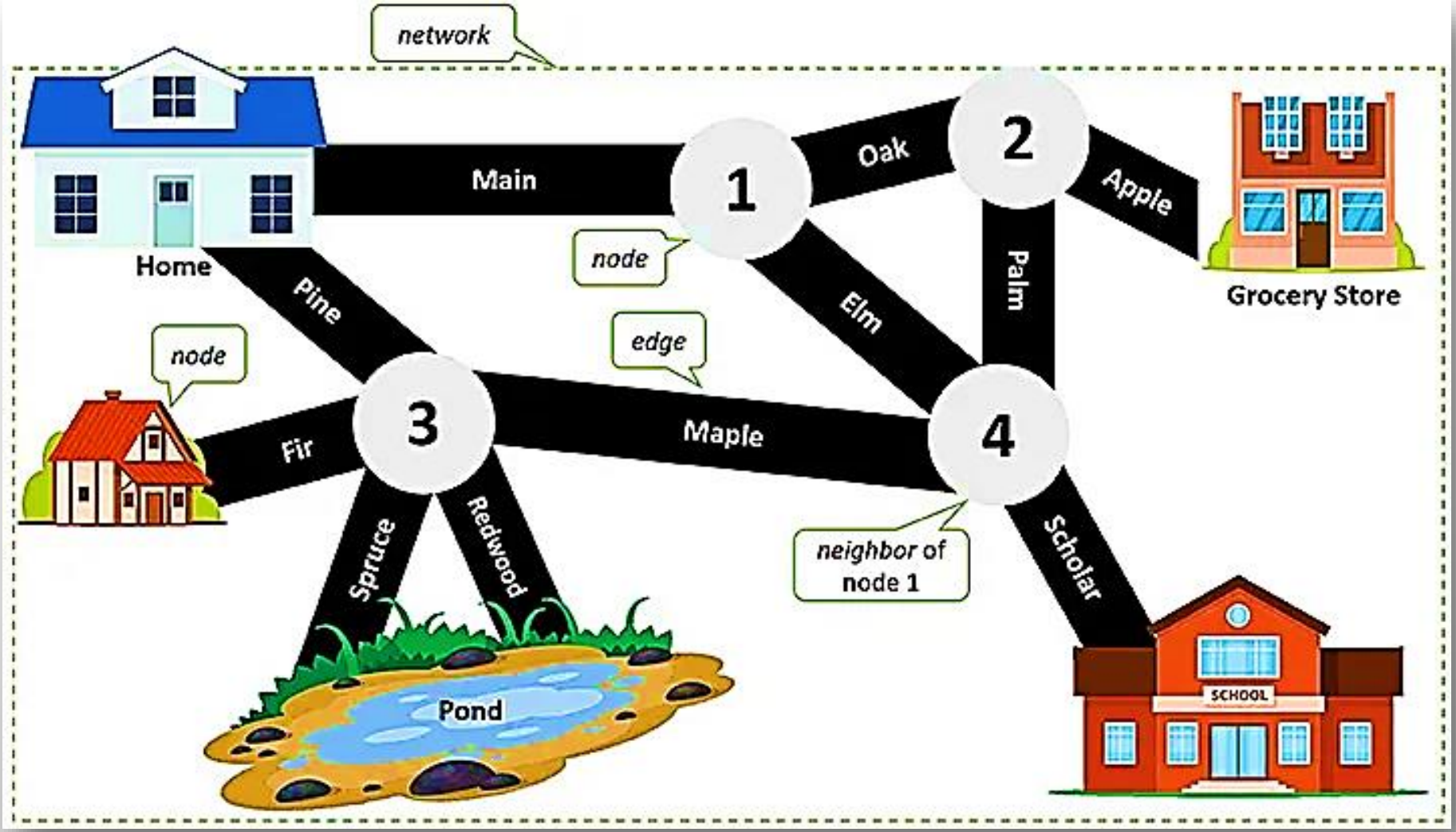
Çizge Algoritmaları

- DFS bir labirentten çıkış yolu ararken kullanılabilir.
- BFS bir haberin tüm şehre 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



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 yavaştır.
- Floyd-Warshall:
  - Tüm çiftler arasındaki en kısa yolları bulur.
  - Negatif ağırlıklı çizgelerde kullanılabilir.

En Kısa Yol Algoritmaları (Shortest Path)

- *A* Arama*:
  - Sezgisel bilgiler kullanılarak aramayı hızlandırır.
  - Hedef düğüme olan tahmini mesafeyi hesaba katar.
- BFS:
  - Ağırlıksız çizgeler üzerinde çalışır.

Dijkstra

- Tek kaynaktan diğer tüm düğümlere olan en kısa yolu bulur.
- 1956 yılında Edsger W. Dijkstra tarafından geliştirilmiştir.
- Pozitif ağırlıklı kenarlardan oluşan çizgelerde çalışır.

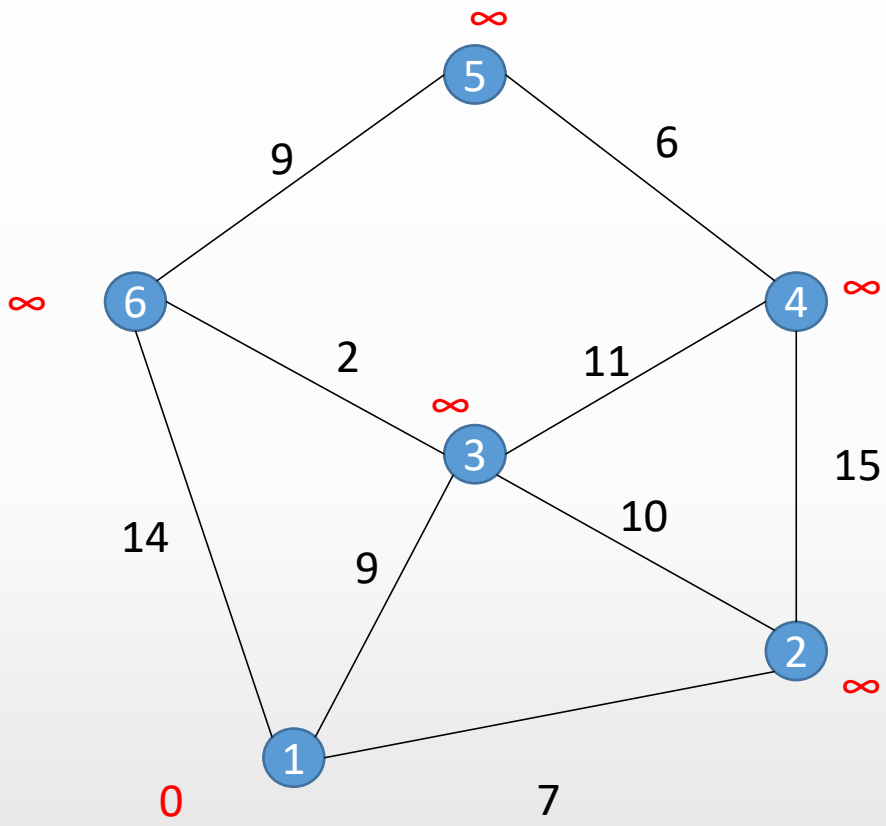
Algoritma Adımları

- Adım 1: Başlangıç düğümü seçilir ve uzaklık değeri 0 atanır. Diğer düğümlere sonsuz uzaklık atanır.
- Adım 2: Başlangıç düğümünden başlayarak, henüz işlenmemiş komşu düğümlere olan uzaklıklar hesaplanır.
- Adım 3: Daha kısa bir yol varsa, düğümün uzaklığı güncellenir.
- Adım 4: İşlenen düğümler işaretlenir ve bir sonraki düğüm seçilir.
- Adım 5: Tüm düğümler işlenene kadar Adım 2'den 4'e kadar tekrarlanır.

Algoritma Karmaşıklığı

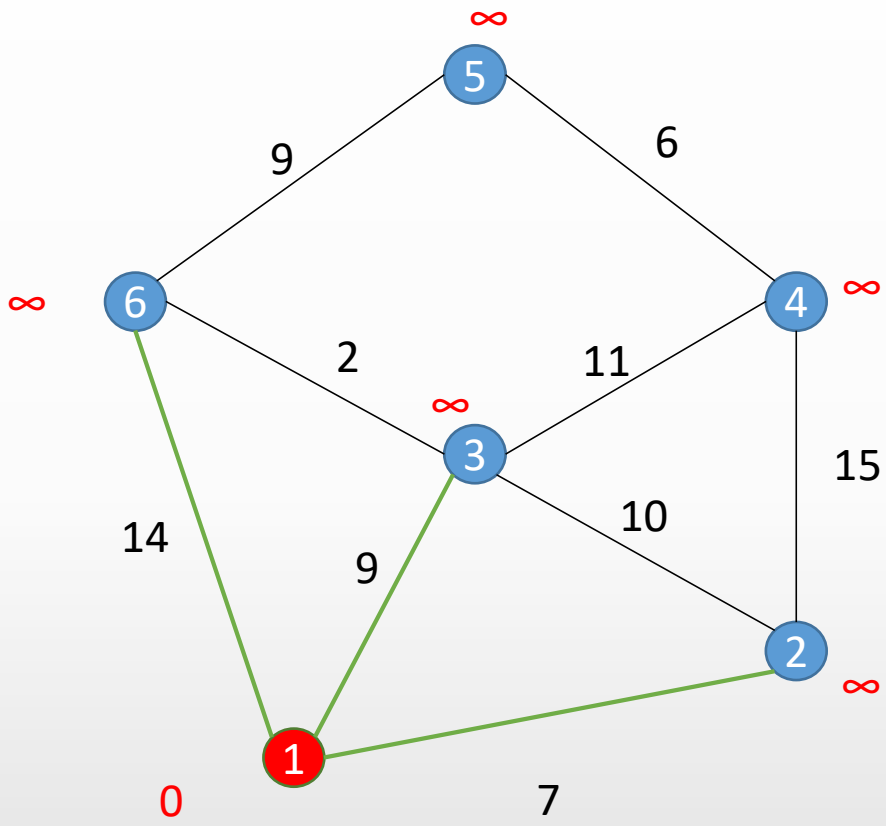
- En kısa mesafeli düğümü seçmek için;
  - Dizi temsili kullanılırsa;
    - $O(V^2)$ karmaşıklığına sahiptir.
  - Öncelik kuyruğu (*Priority Queue*) kullanılırsa;
    - karmaşıklık $O((V + E)logV)$ olur.

Dijkstra



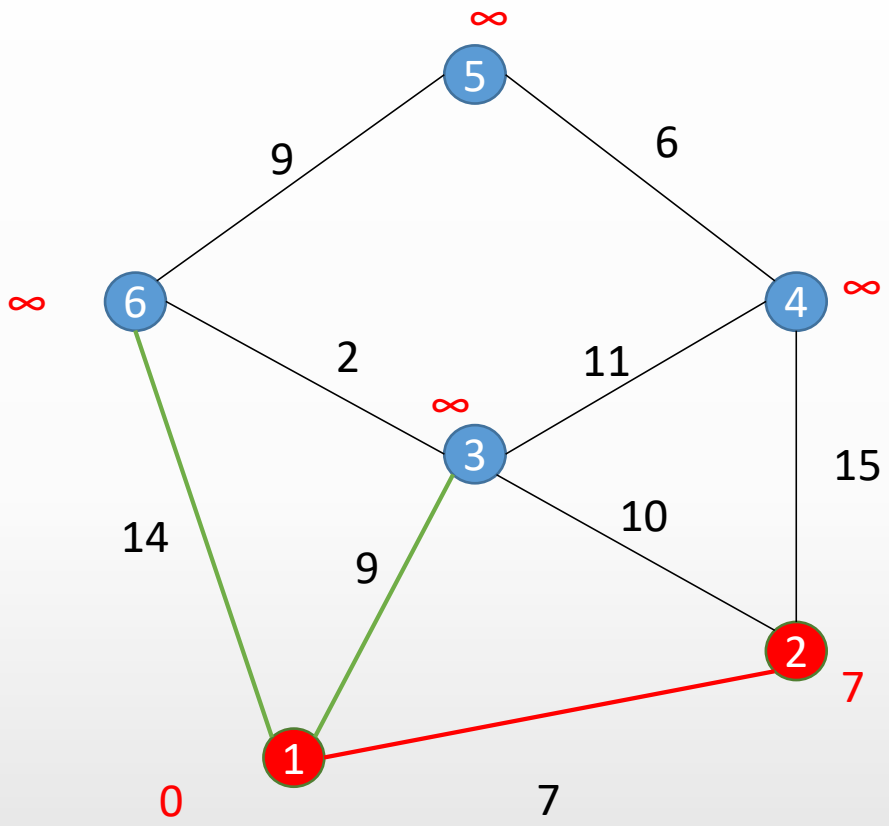
	1
1	0
2	∞
3	∞
4	∞
5	∞
6	∞

Dijkstra



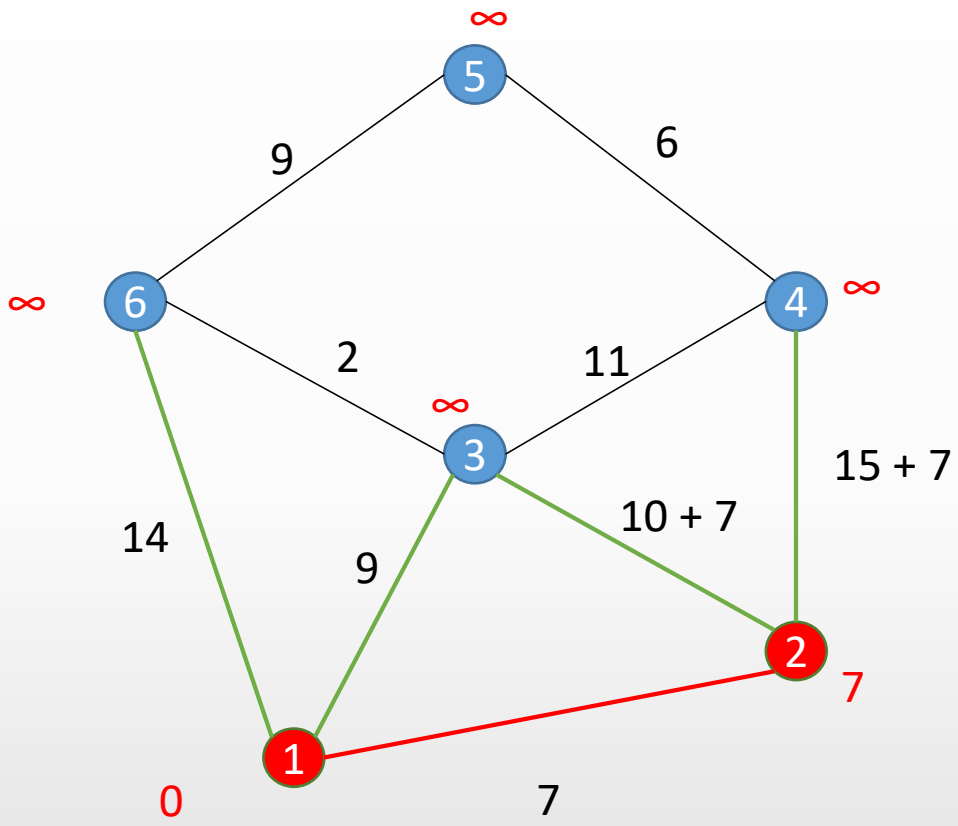
	1
1	0
2	∞
3	∞
4	∞
5	∞
6	∞

Dijkstra



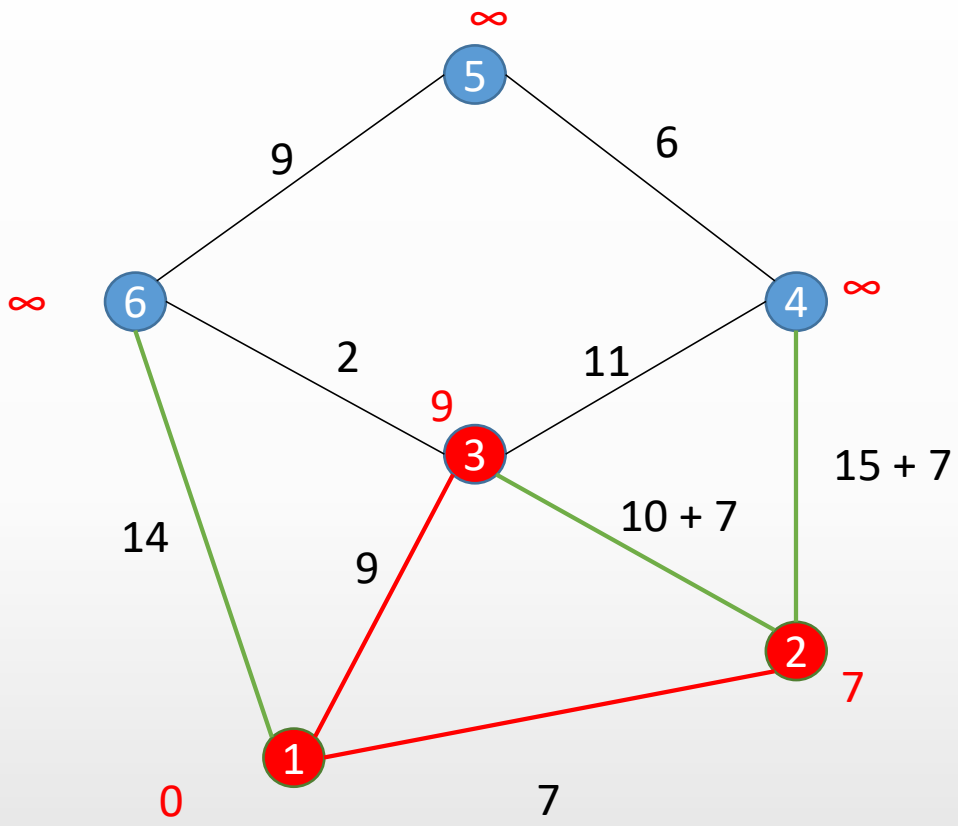
	1
1	0
2	7
3	∞
4	∞
5	∞
6	∞

Dijkstra



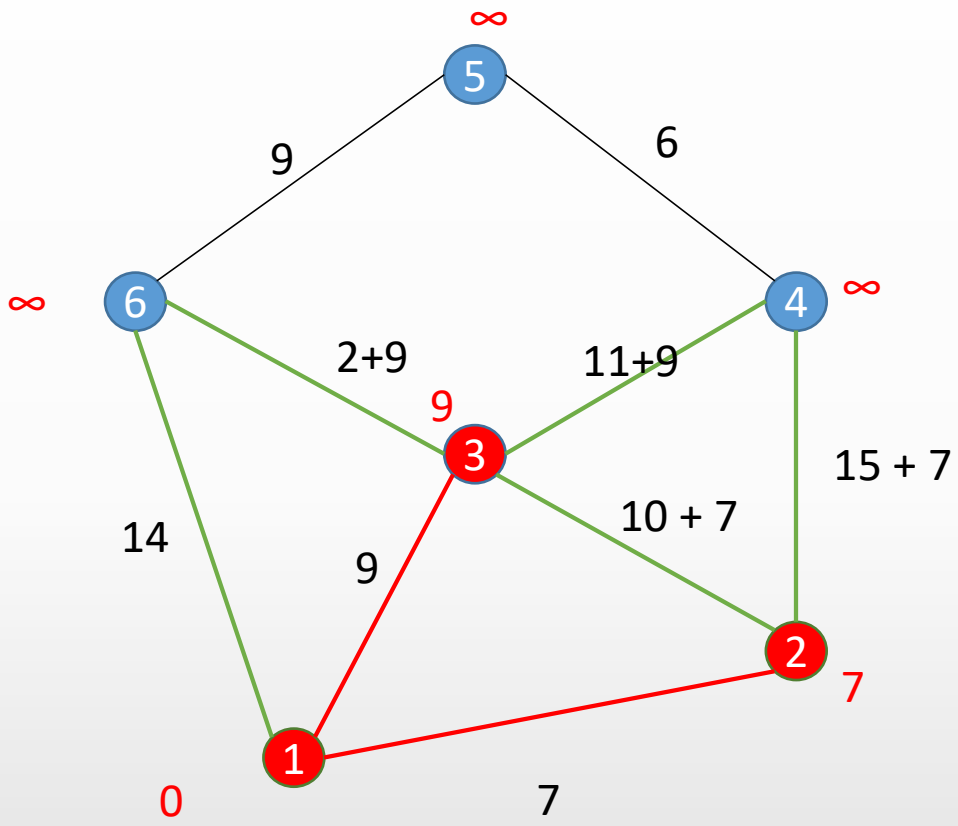
	1
1	0
2	7
3	∞
4	∞
5	∞
6	∞

Dijkstra



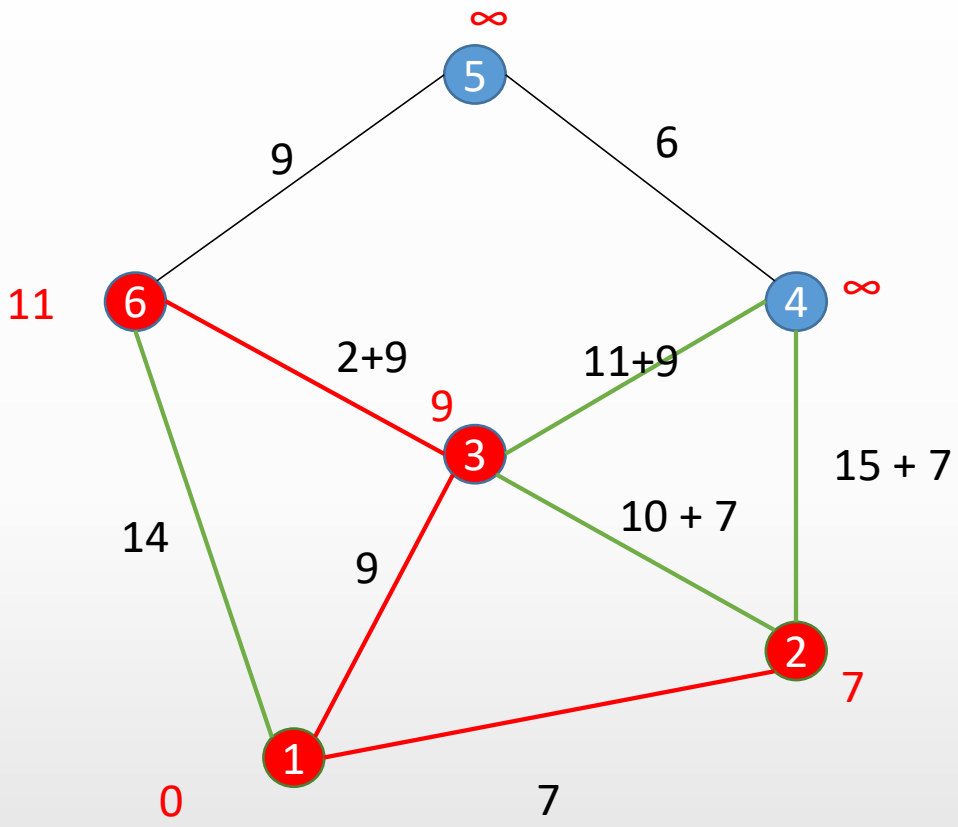
	1
1	0
2	7
3	9
4	∞
5	∞
6	∞

Dijkstra



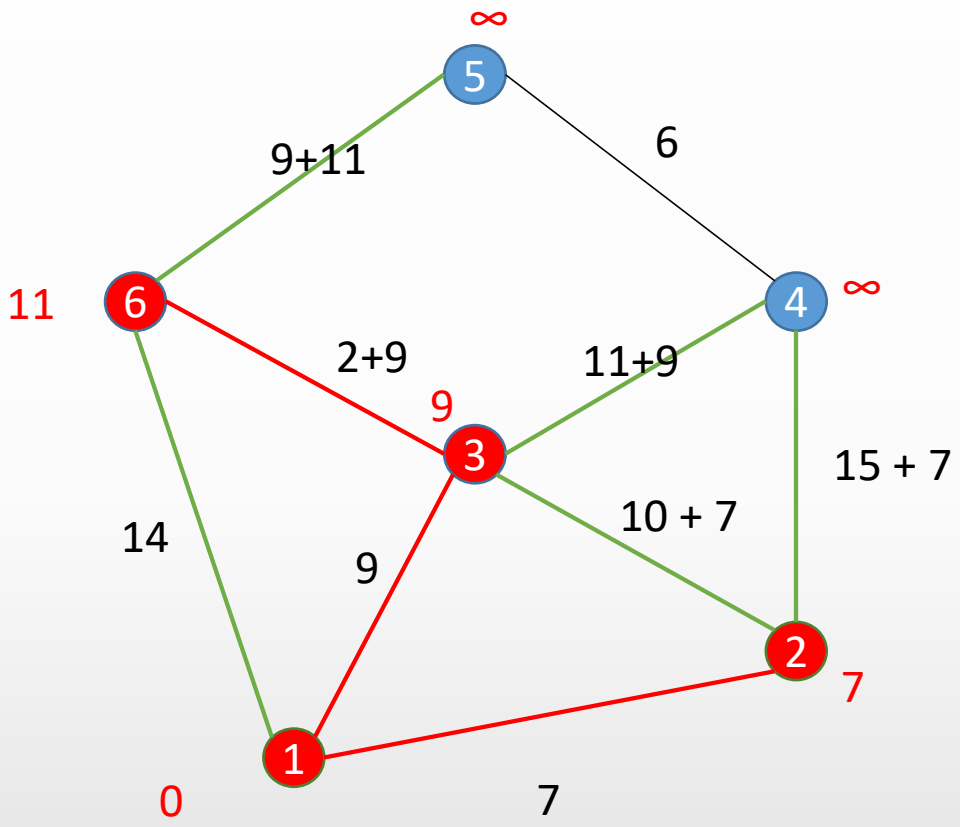
	1
1	0
2	7
3	9
4	∞
5	∞
6	∞

Dijkstra



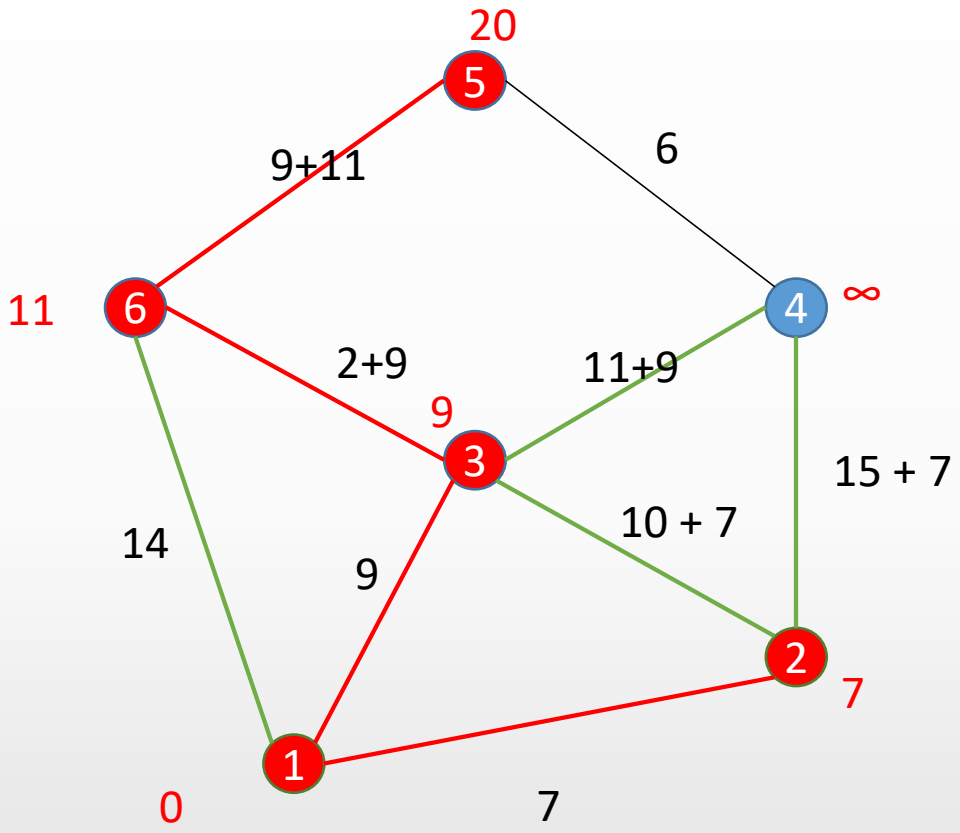
	1
1	0
2	7
3	9
4	∞
5	∞
6	11

Dijkstra



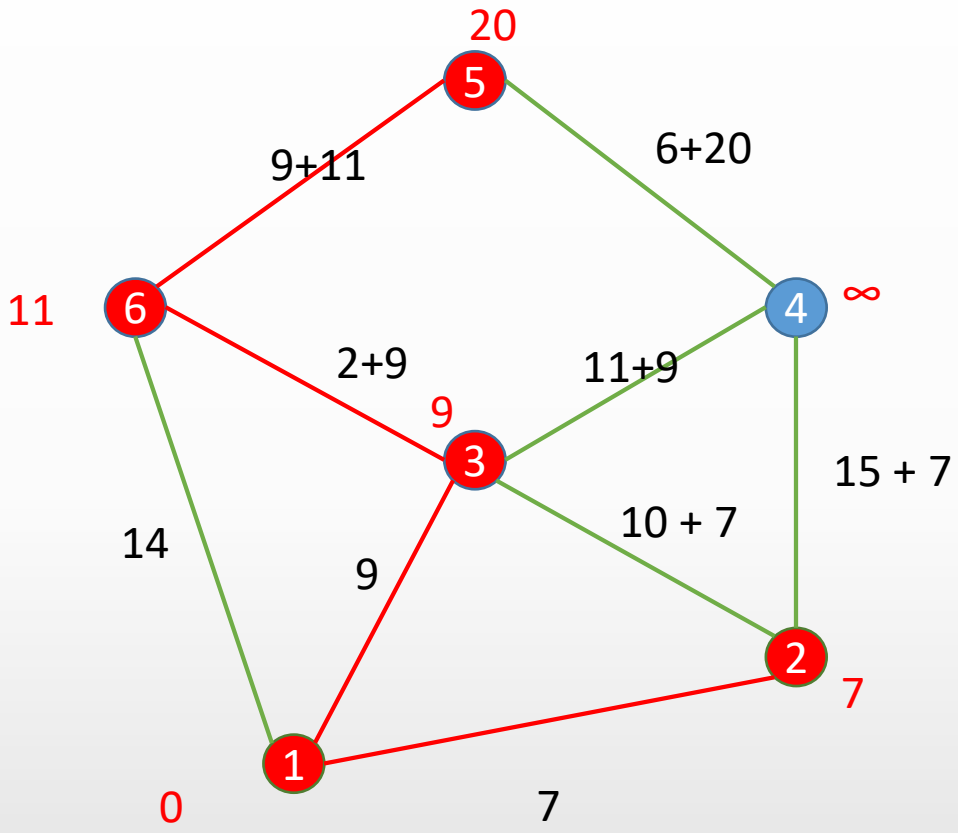
	1
1	0
2	7
3	9
4	∞
5	∞
6	11

Dijkstra



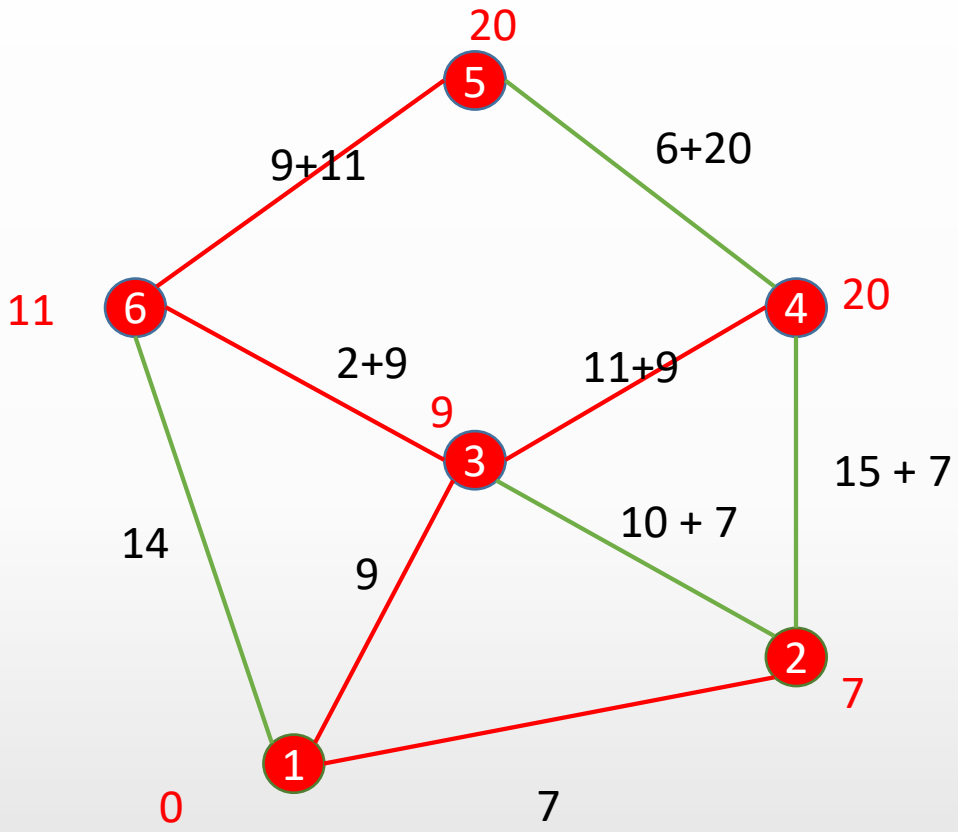
	1
1	0
2	7
3	9
4	∞
5	20
6	11

Dijkstra



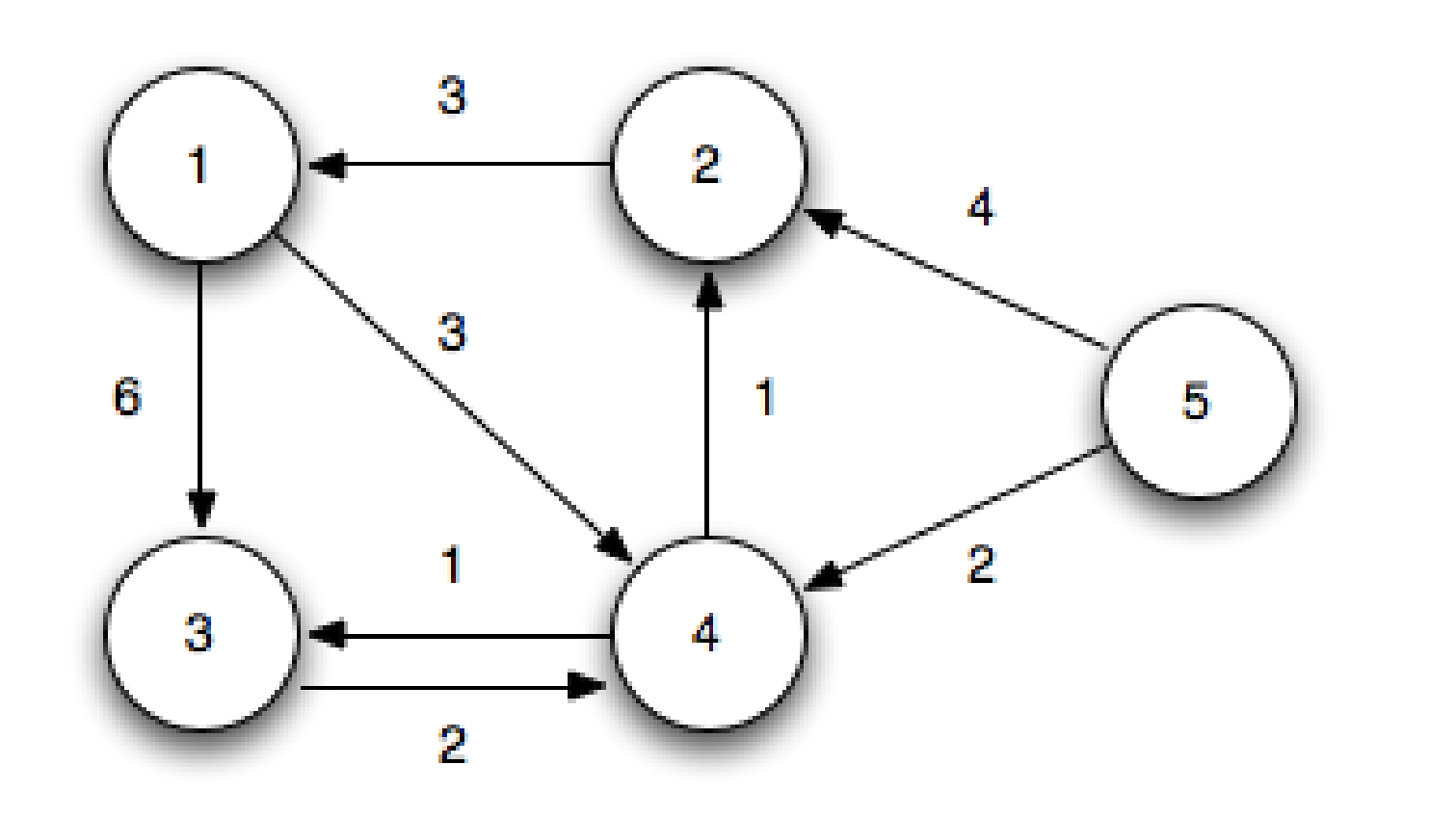
	1
1	0
2	7
3	9
4	∞
5	20
6	11

Dijkstra



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

Örnek



- Using vertex 5 as the source (setting its distance to 0), we initialize all the other distances to ∞, set S = ∅, and place all the vertices in the queue (resolving ties by lowest vertex number)

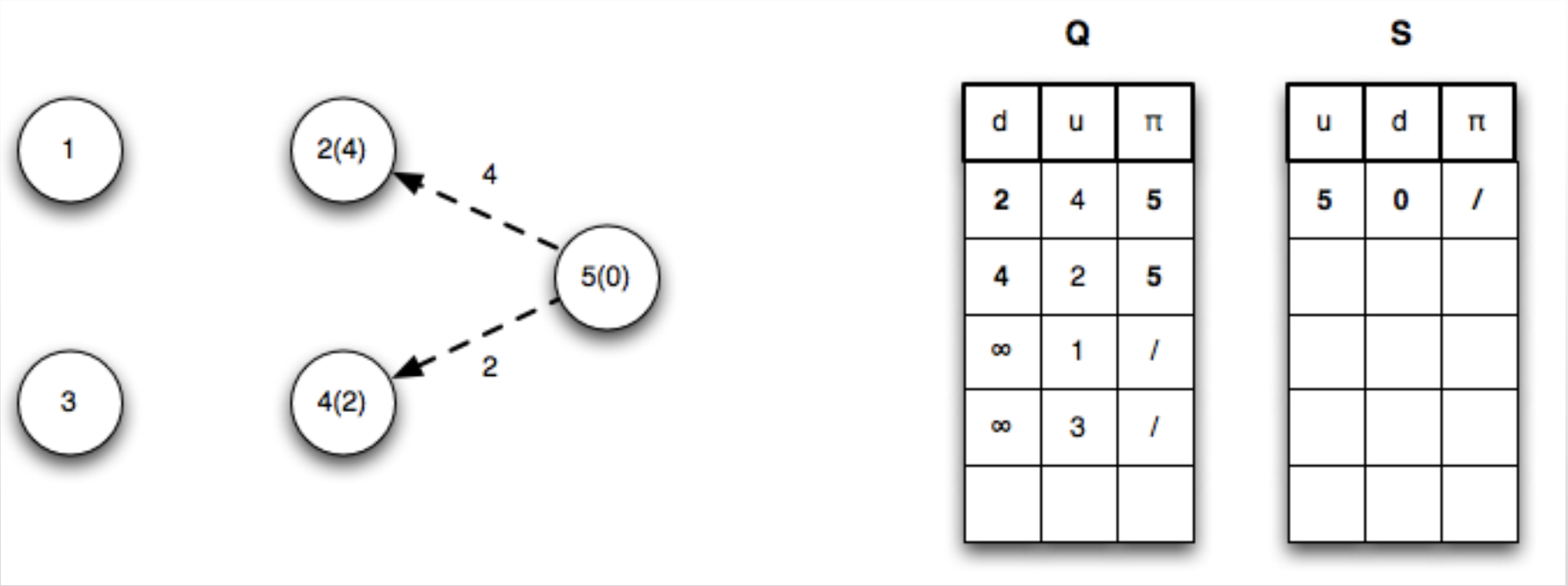
Q

d	u	π
0	5	/
∞	1	/
∞	2	/
∞	3	/
∞	4	/

S

u	d	π

- Iteration 1: Dequeue vertex 5 placing it in S (with a distance 0) and relaxing edges (u5,u2) and (u5,u4) then reprioritizing the queue



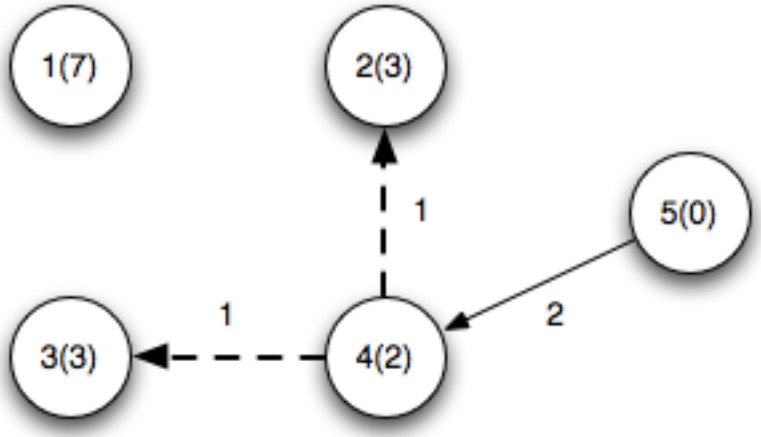
Q

d	u	π
2	4	**5**
4	2	**5**
∞	1	/
∞	3	/

S

u	d	π
5	**0**	/

- Iteration 2: Dequeue vertex 4 placing it in S (with a distance 2) and relaxing edges (u4,u2) and (u4,u3) then reprioritizing the queue. Note edge (u4,u2) finds a shorter path to vertex 2 by going through vertex 4



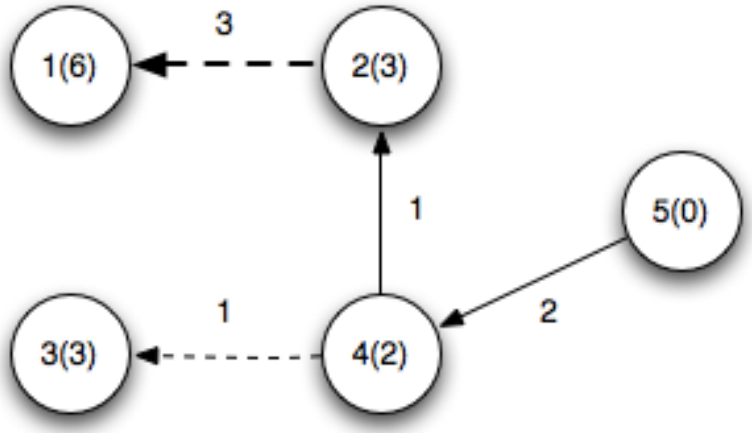
Q

d	u	π
3	2	**4**
3	3	**4**
∞	1	/

S

u	d	π
5	0	/
4	**2**	**5**

- Iteration 3: Dequeue vertex 2 placing it in S (with a distance 3) and relaxing edge (u2,u1) then reprioritizing the queue.



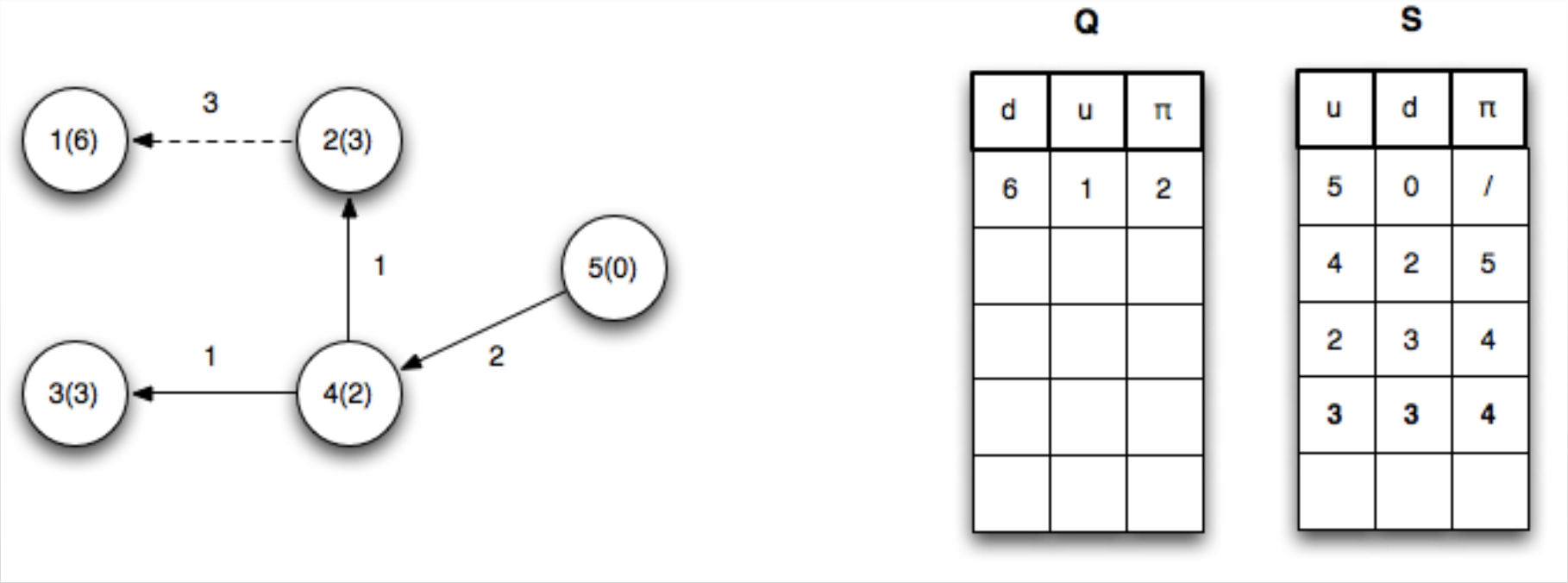
Q

d	u	π
3	3	4
6	1	**2**

S

u	d	π
5	0	/
4	2	5
2	**3**	**4**

- Iteration 4: Dequeue vertex 3 placing it in S (with a distance 3) and relaxing no edges then reprioritizing the queue.



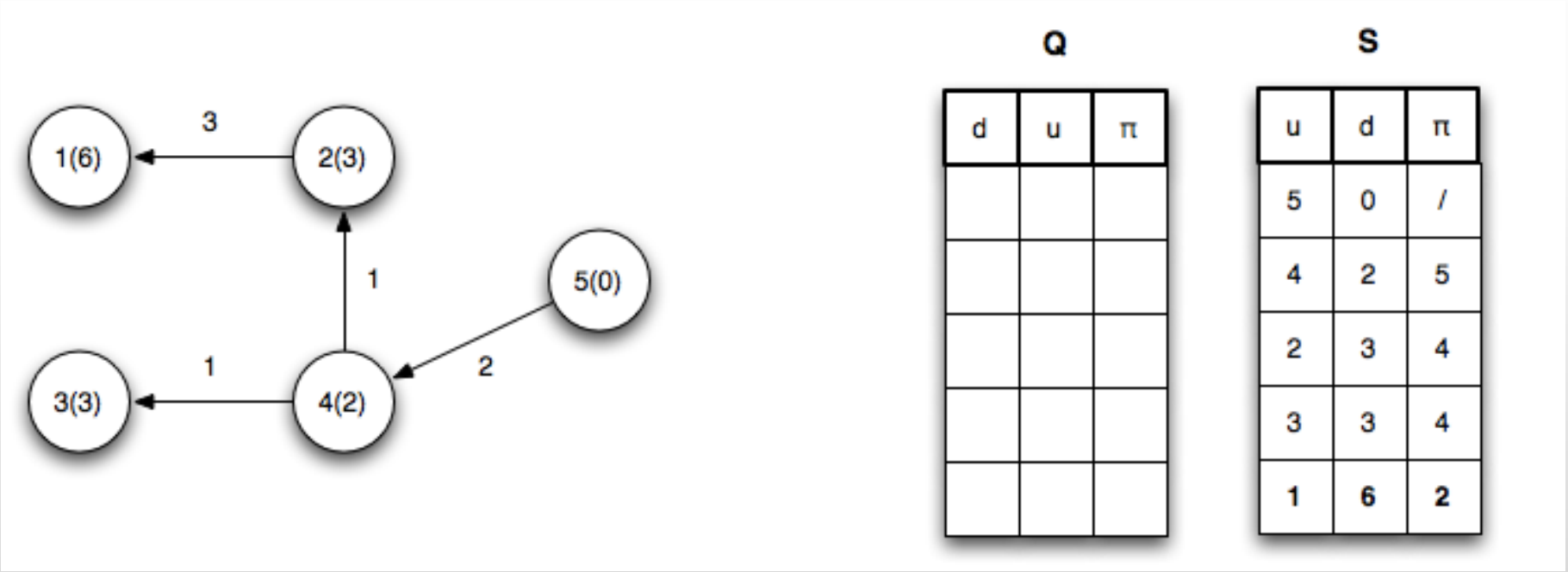
Q

d	u	π
6	1	2

S

u	d	π
5	0	/
4	2	5
2	3	4
3	**3**	**4**

- Iteration 5: Dequeue vertex 1 placing it in S (with a distance 6) and relaxing no edges.



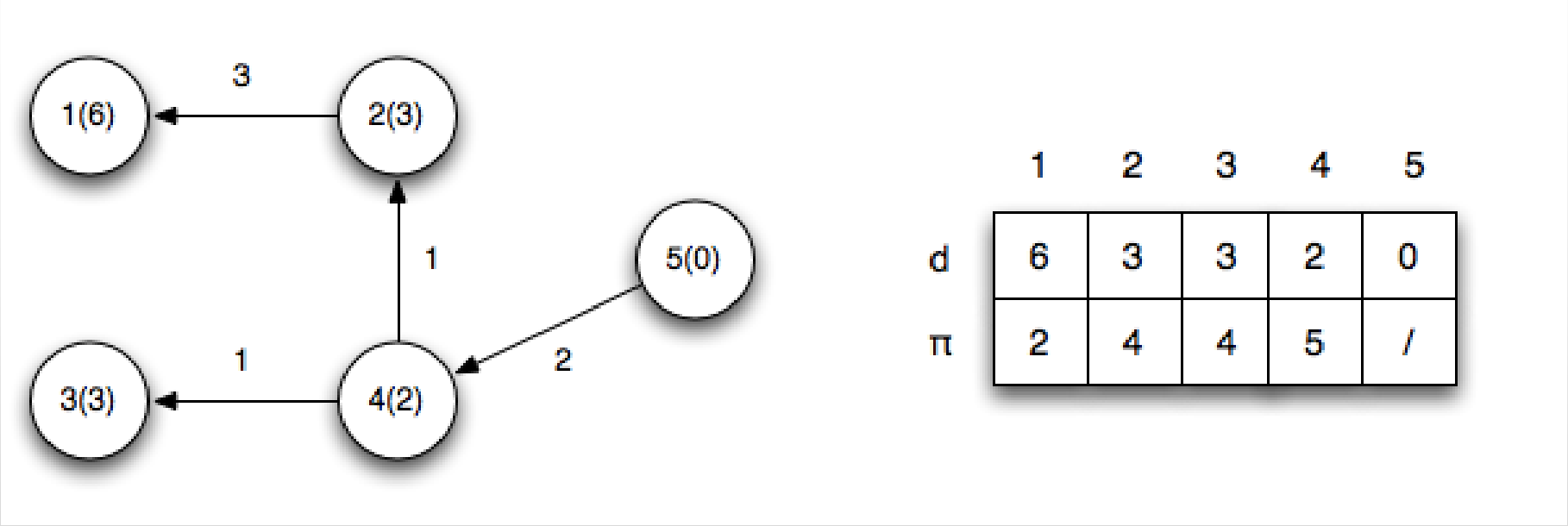
Q

d	u	π

S

u	d	π
5	0	/
4	2	5
2	3	4
3	3	4
1	**6**	**2**

- The final shortest paths from vertex 5 with corresponding distances is



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

Bellman Ford

- Tek bir kaynaktan diğer tüm düğümlere en kısa yolu bulmak için kullanılır.
- 1958 yılında Richard Bellman ve Lester Ford Jr. tarafından geliştirilmiştir.
- Negatif ağırlıklı kenarları işleyebilir.
- Negatif ağırlıklı döngüleri bulabilir.

Algoritma İlkeleri

- Her düğüm için en kısa yol tahminlerini tutan bir dizi kullanır.
- Başlangıçta tüm düğümlerin en kısa yol tahminleri sonsuz atar.
- Çizge üzerindeki tüm kenarlar teker teker incelenir ve
  - her bir düğüm için en kısa yol tahminleri güncellenir.

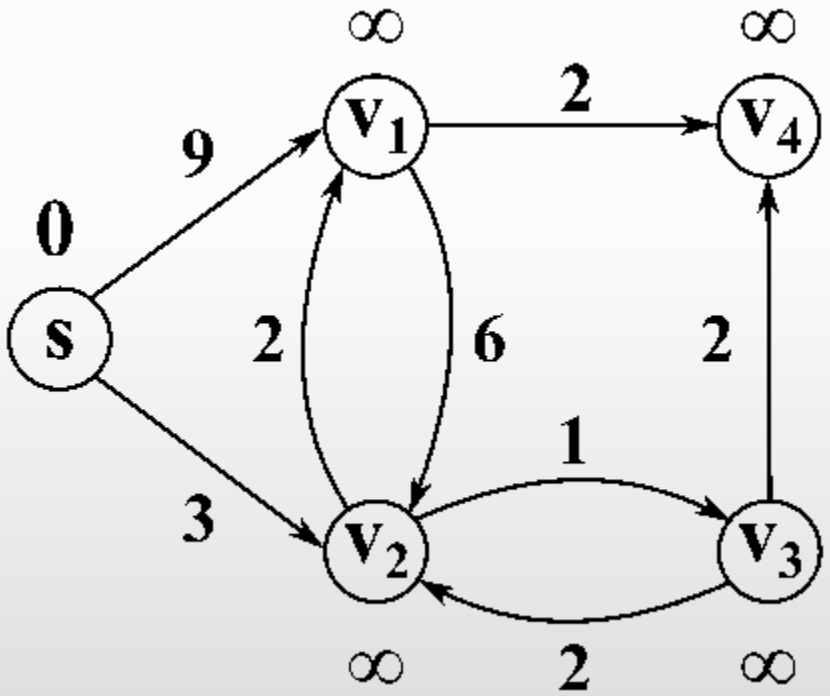
Algoritma Adımları

- Adım 1: Başlangıç düğümü seçilir ve bu düğüme uzaklık 0 atanır. Diğer düğümlere sonsuz uzaklık atanır.
- Adım 2: Tüm kenarlar tek tek incelenir ve düğümler arasındaki uzaklıklar güncellenir. Negatif döngü kontrolü için tüm kenarlar bir kez daha incelenir.
- Adım 3: Eğer bir düğümün uzaklığı güncellenirse, bu düğümün komşularının uzaklıkları da güncellenir.
- Adım 4: Eğer negatif döngü bulunursa, algoritma bu döngüyü tespit eder.

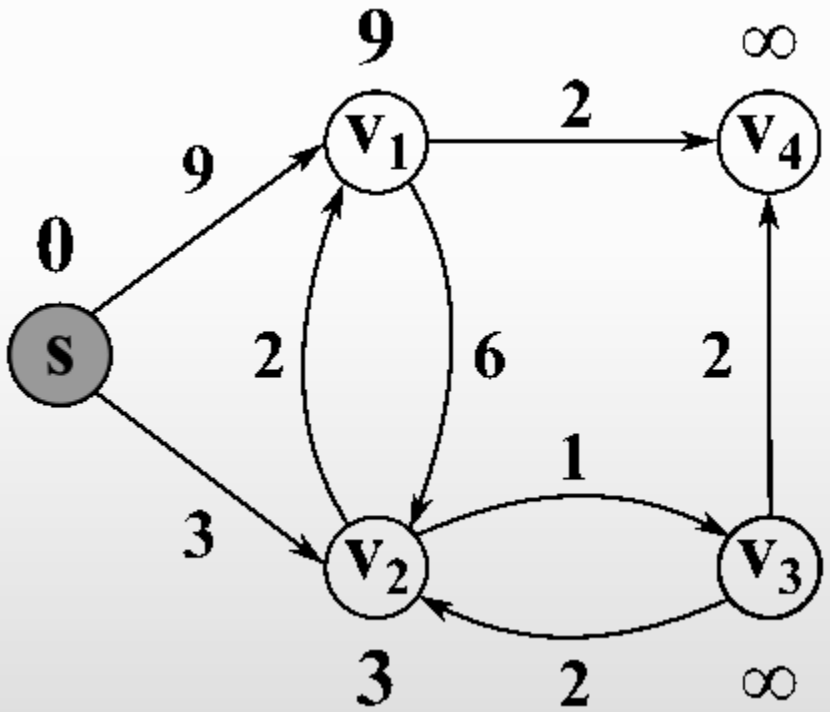
Karmaşıklık Analizi

- Çizge üzerindeki tüm kenarları V-1 kez inceler.
- *O(V*E)* karmaşıklığına sahiptir.

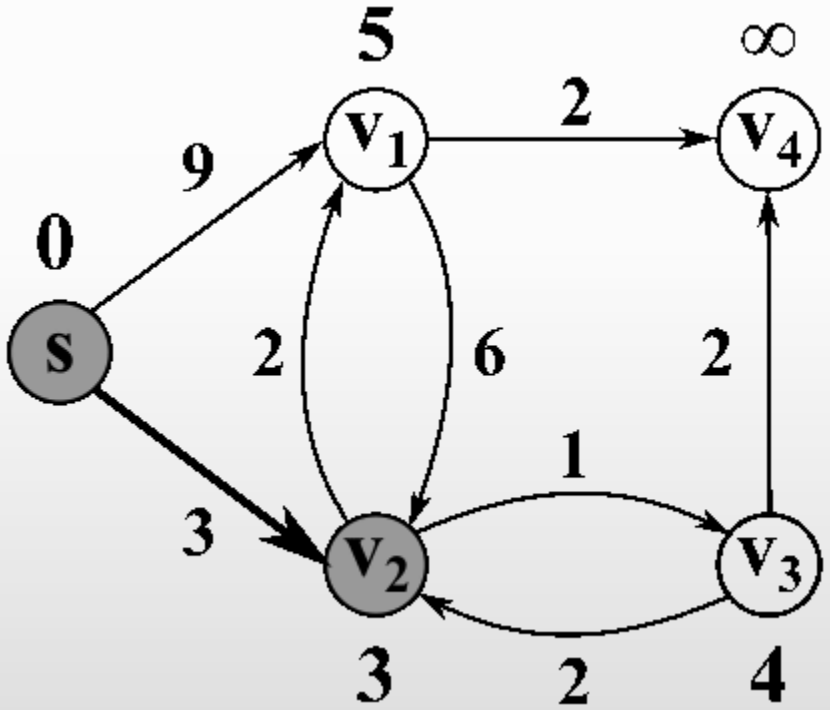
Bellman Ford



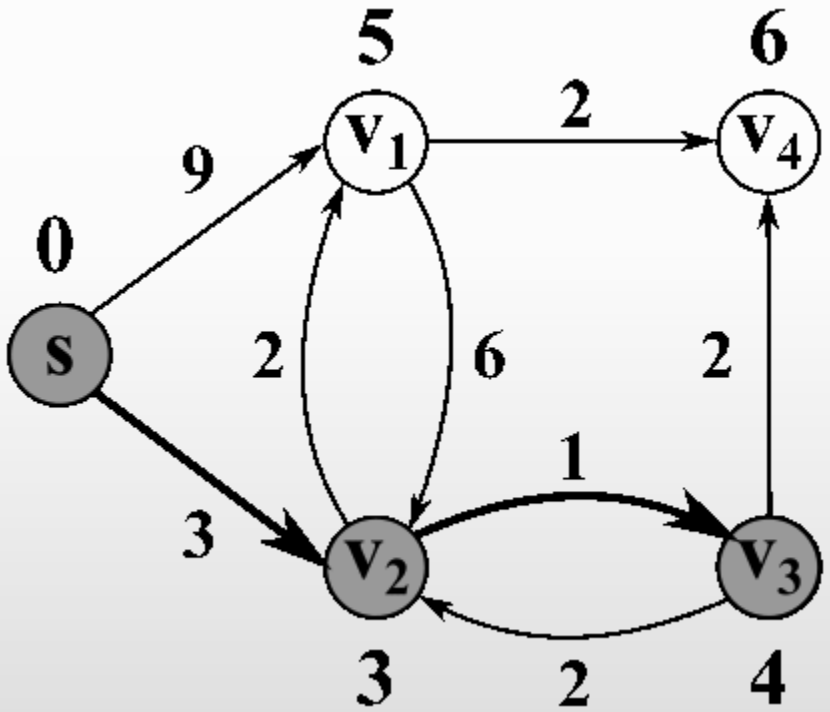
Bellman Ford



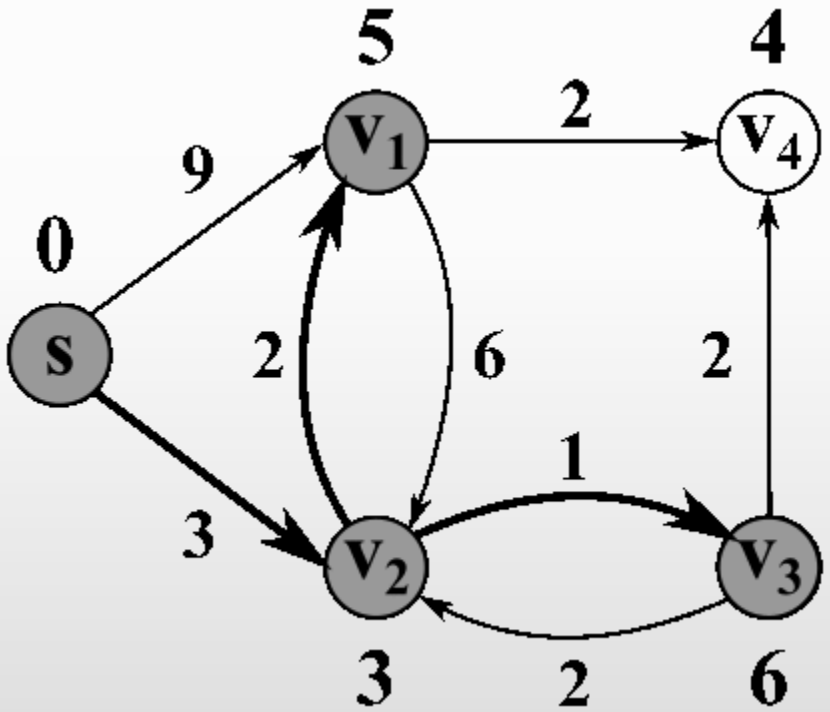
Bellman Ford



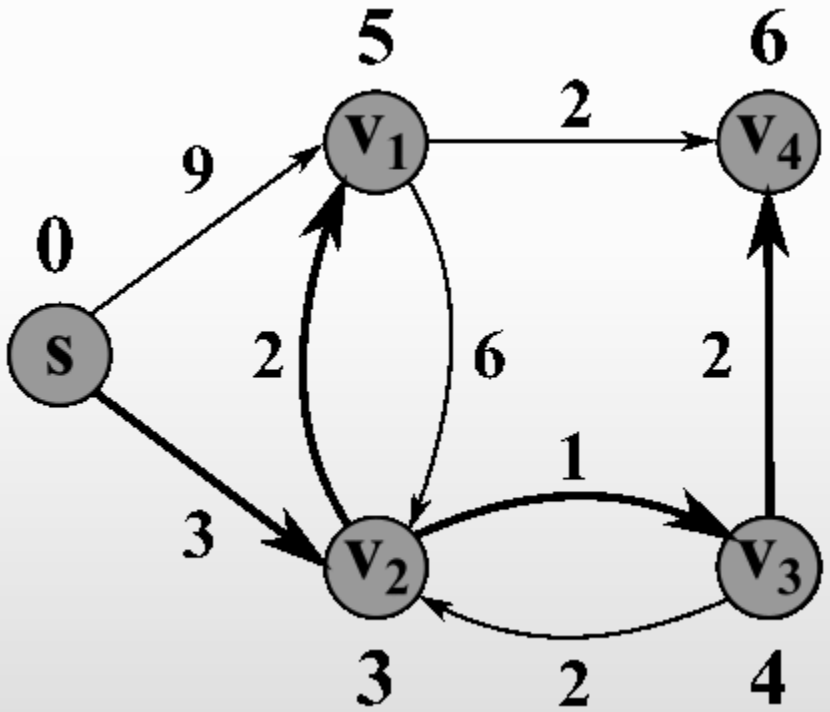
Bellman Ford



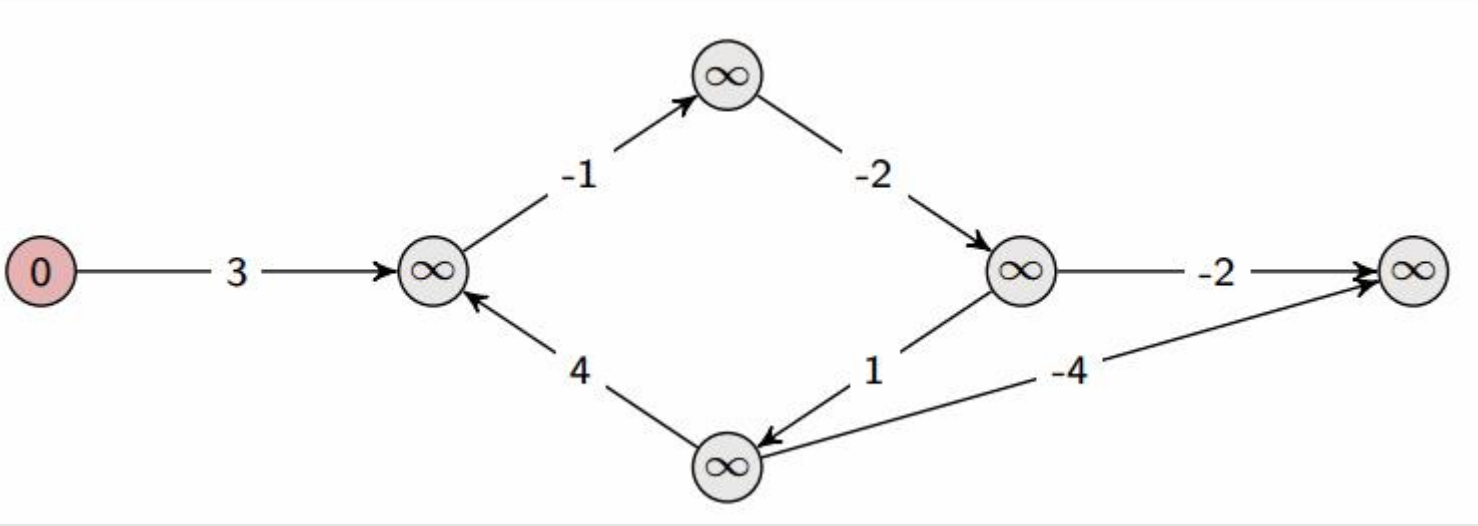
Bellman Ford



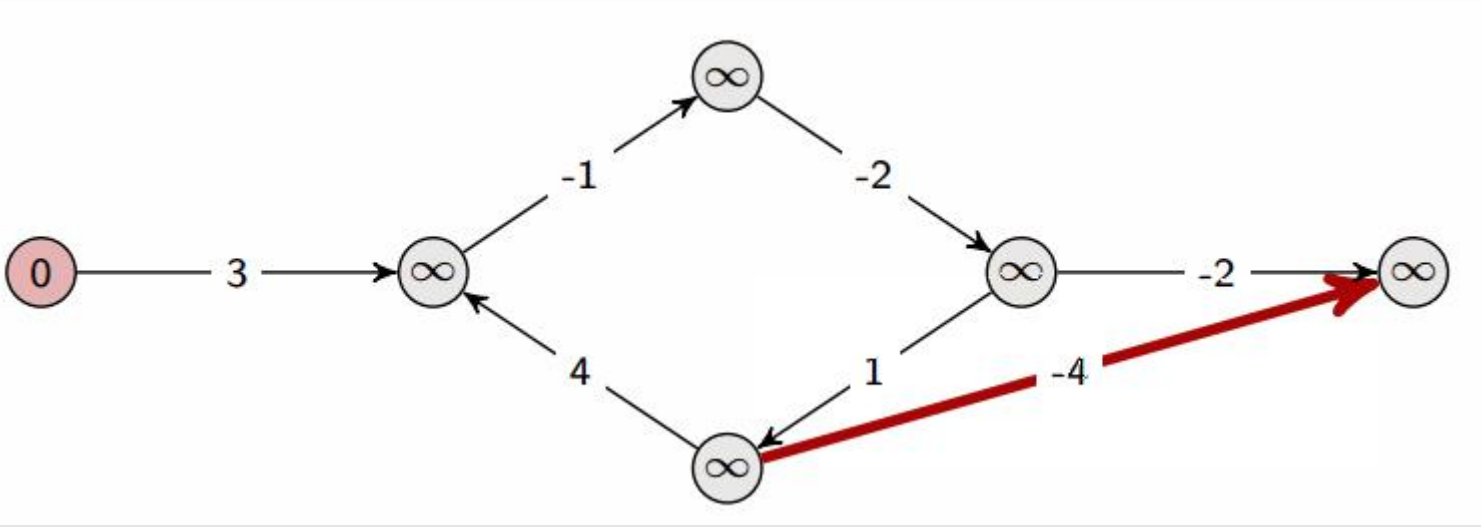
Bellman Ford



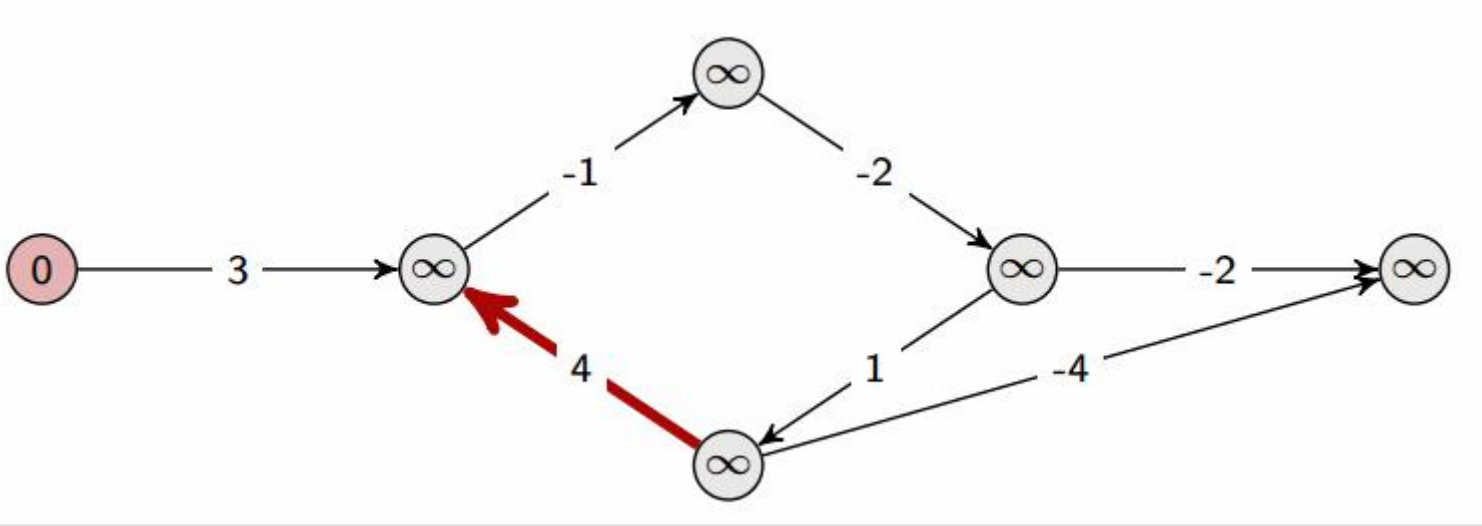
Bellman Ford



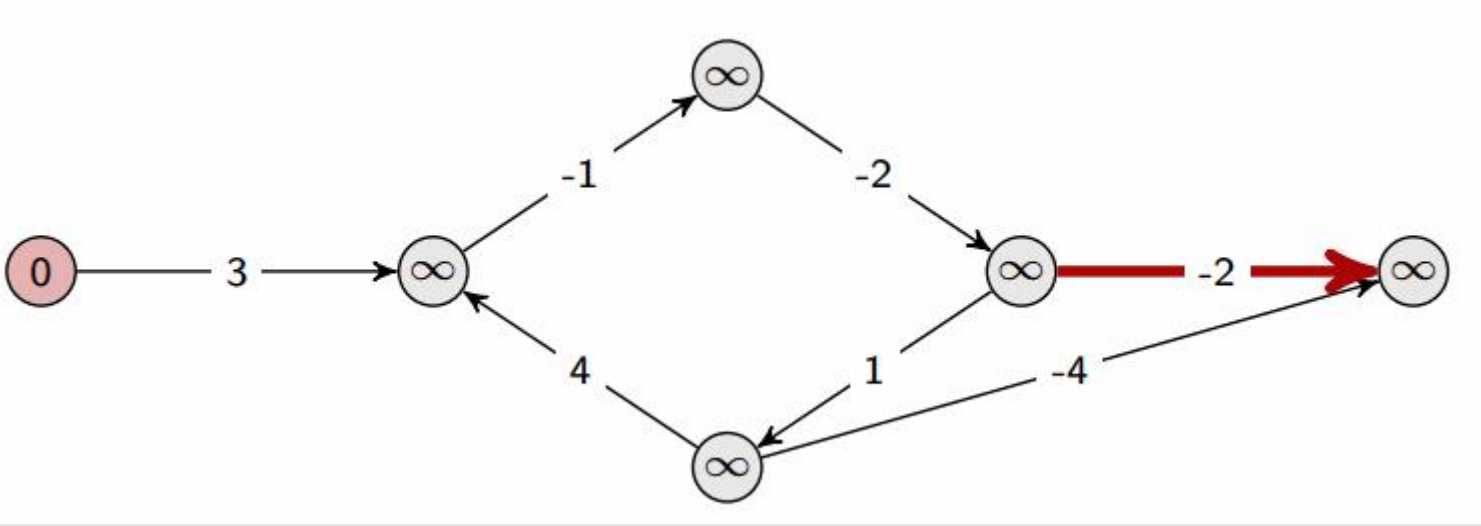
Bellman Ford



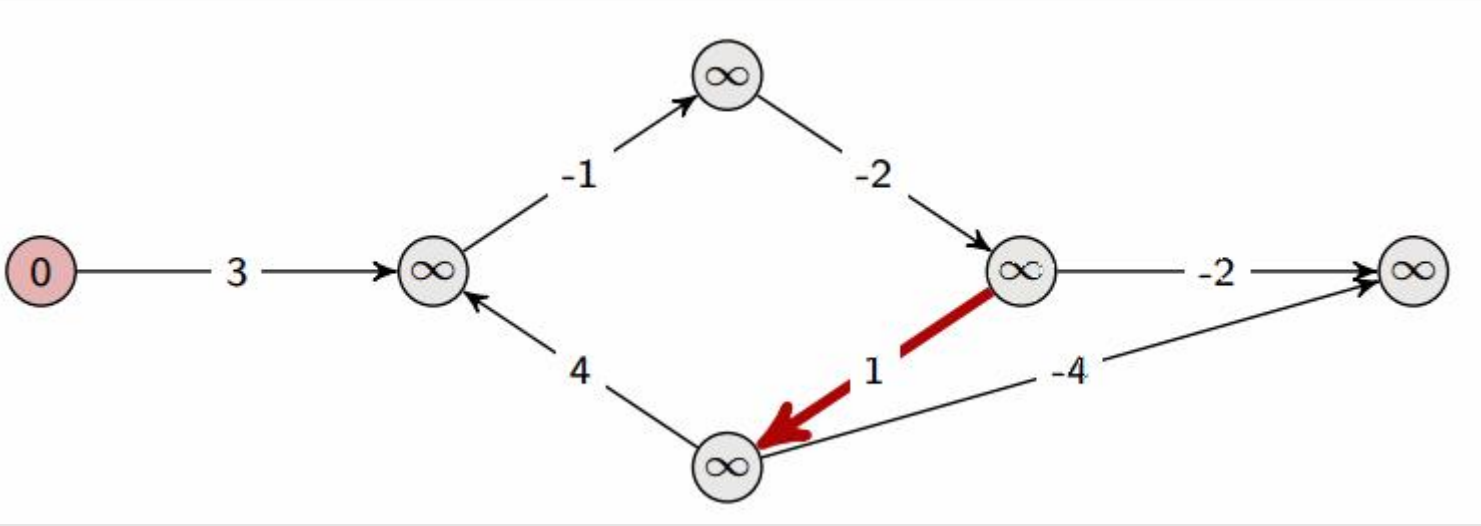
Bellman Ford



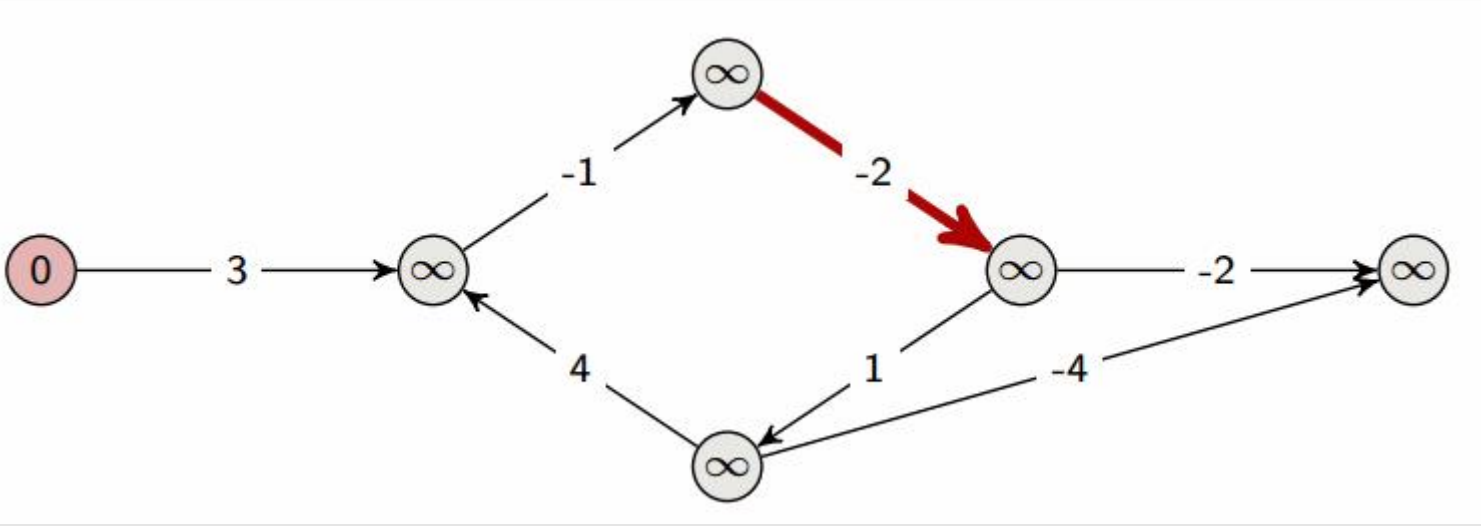
Bellman Ford



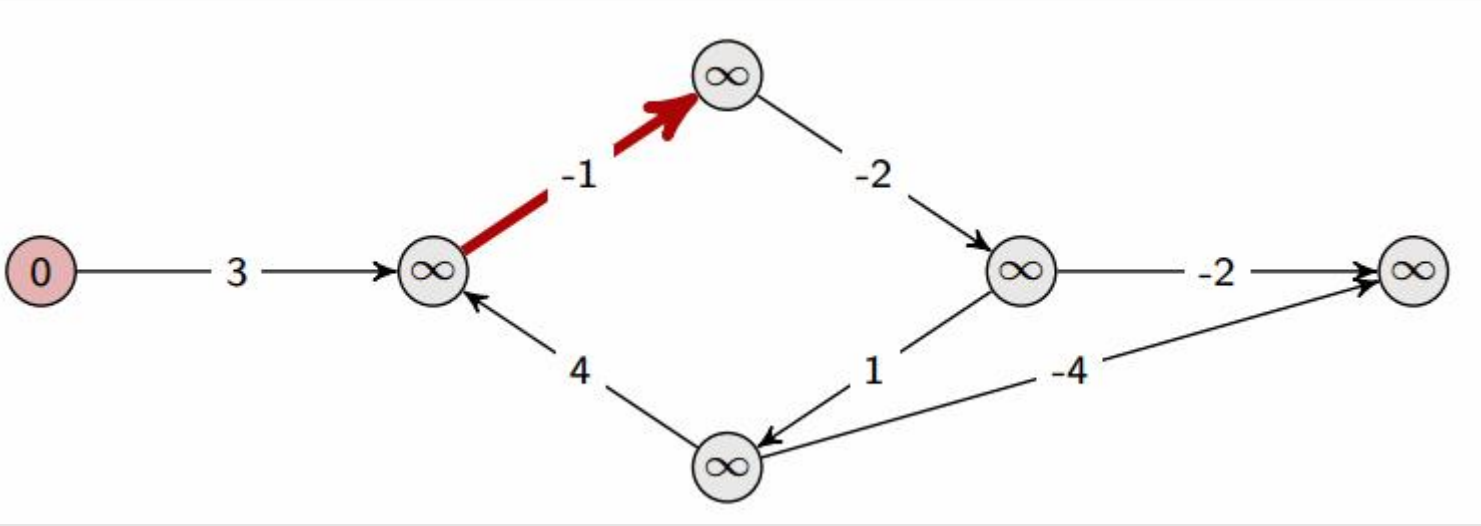
Bellman Ford



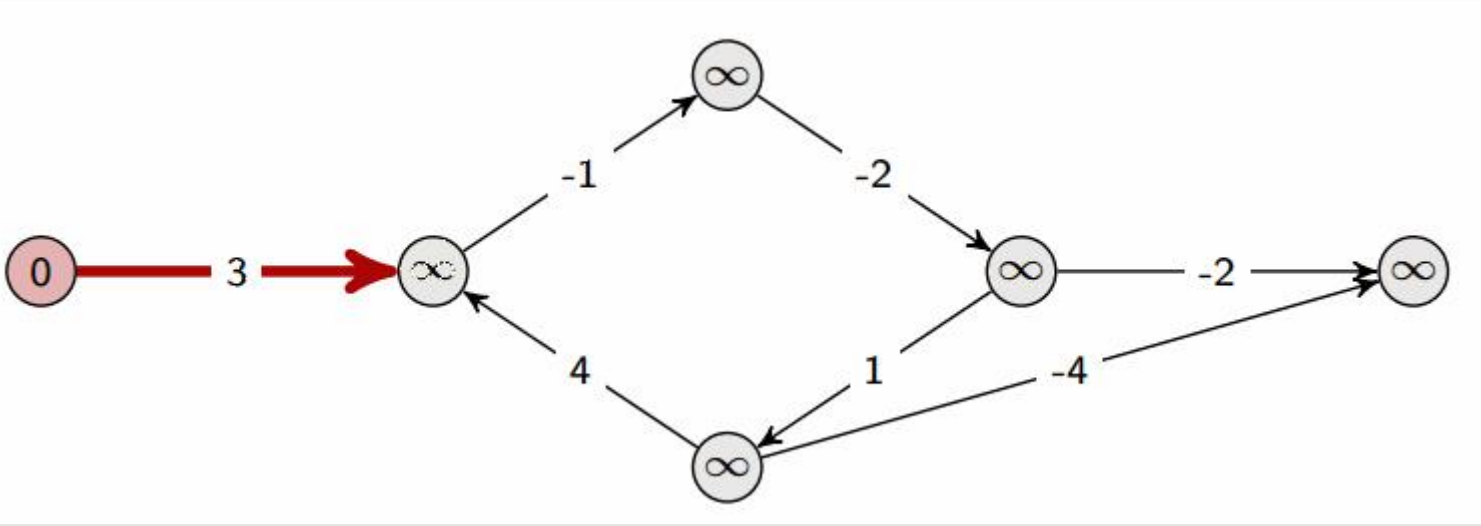
Bellman Ford



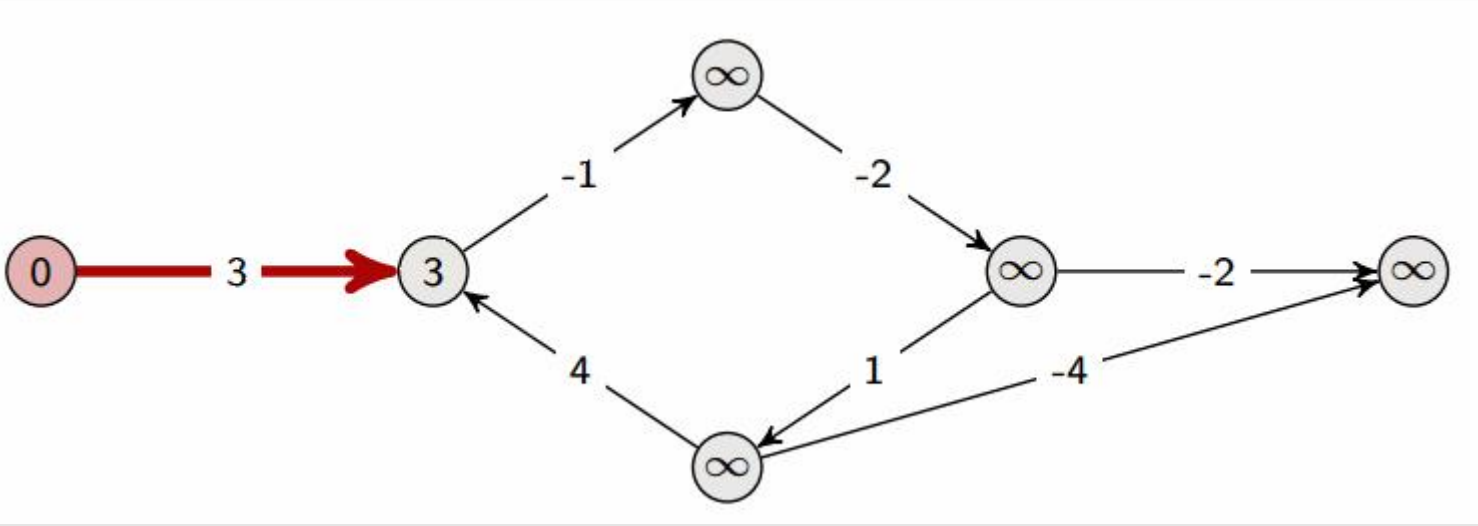
Bellman Ford



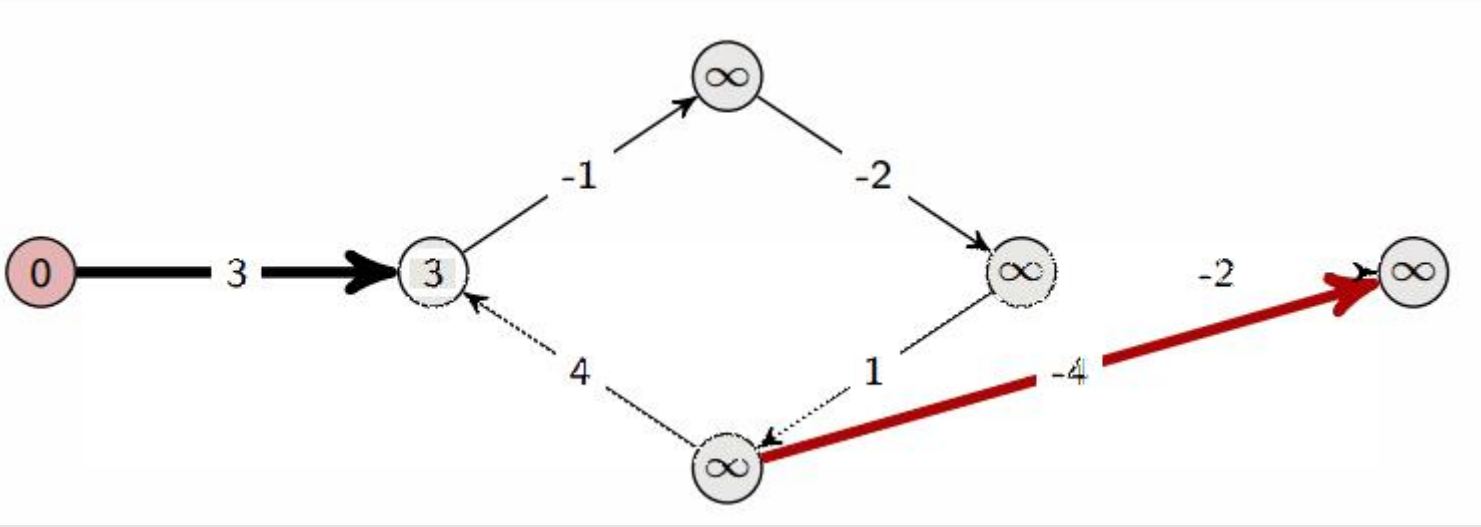
Bellman Ford



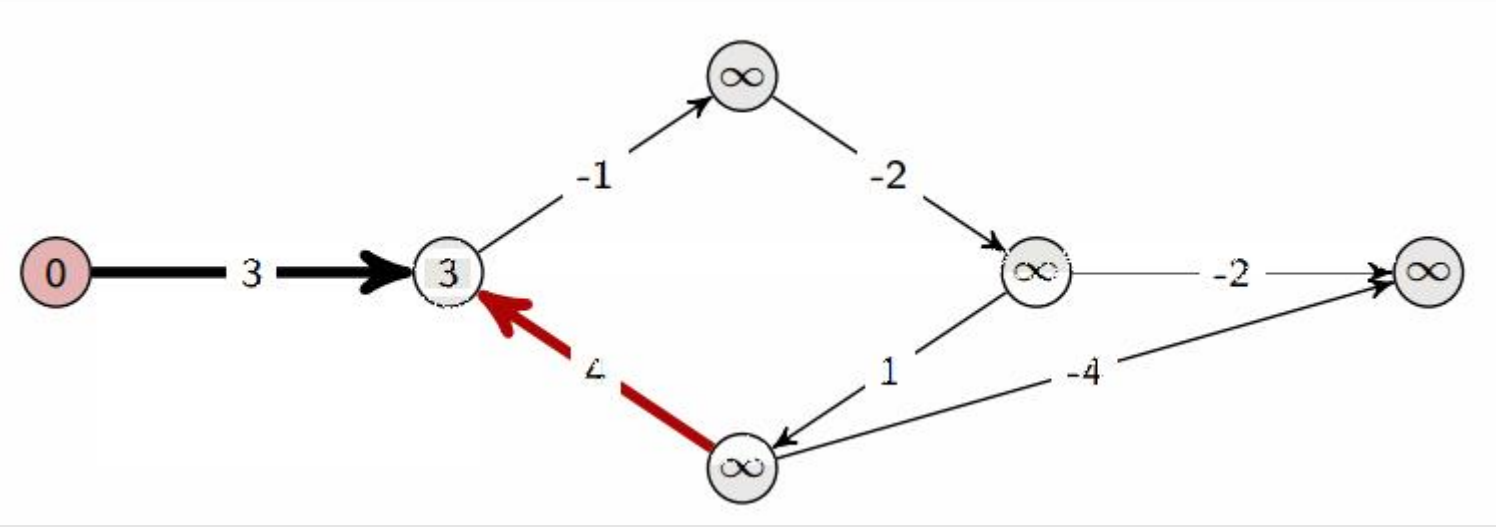
Bellman Ford



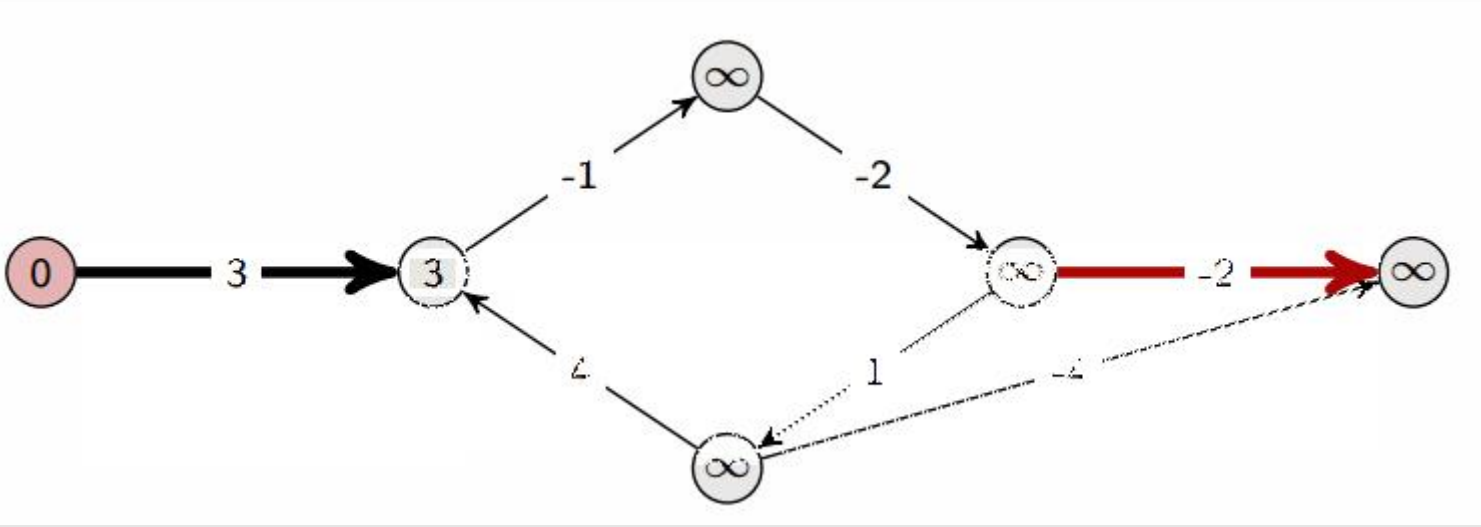
Bellman Ford



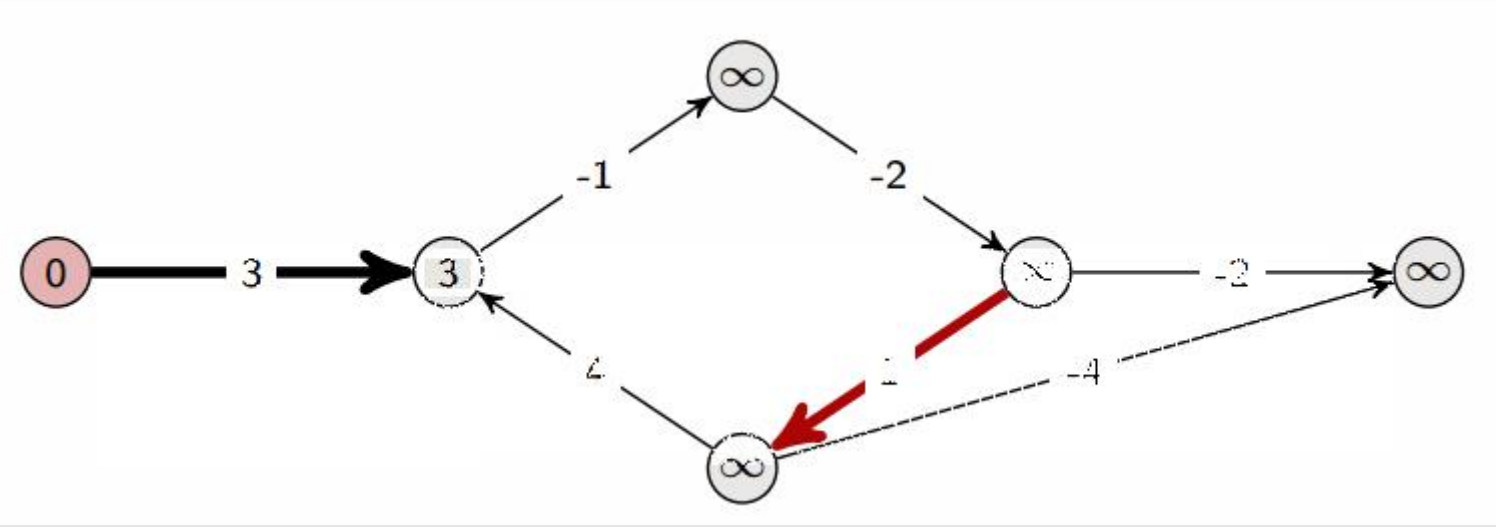
Bellman Ford



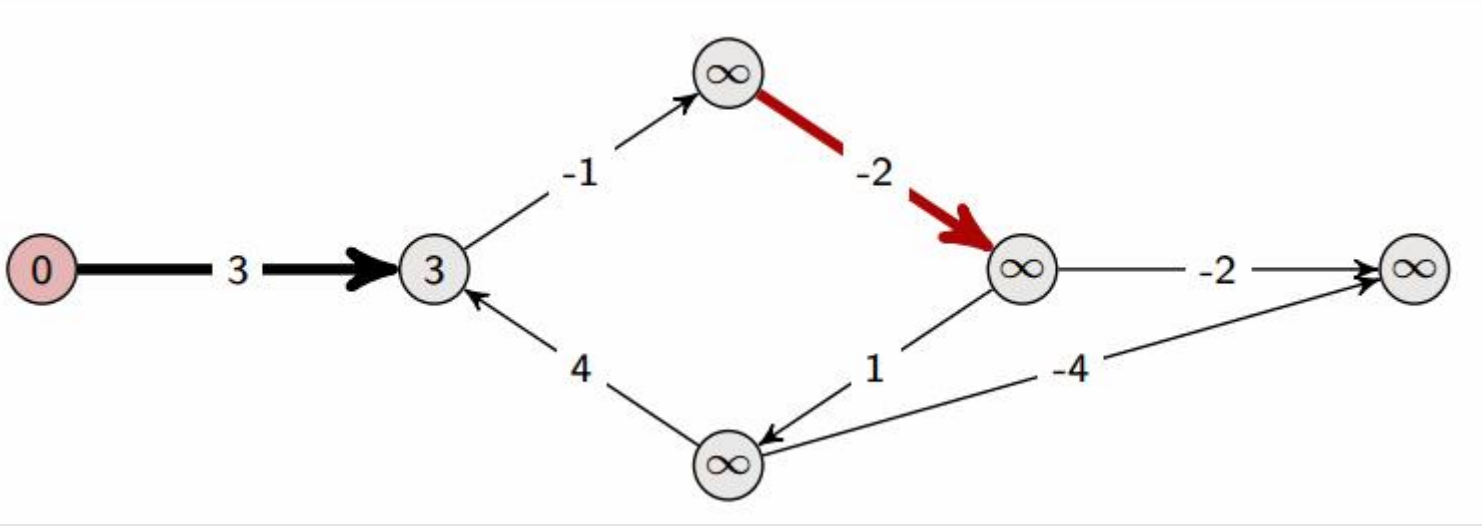
Bellman Ford



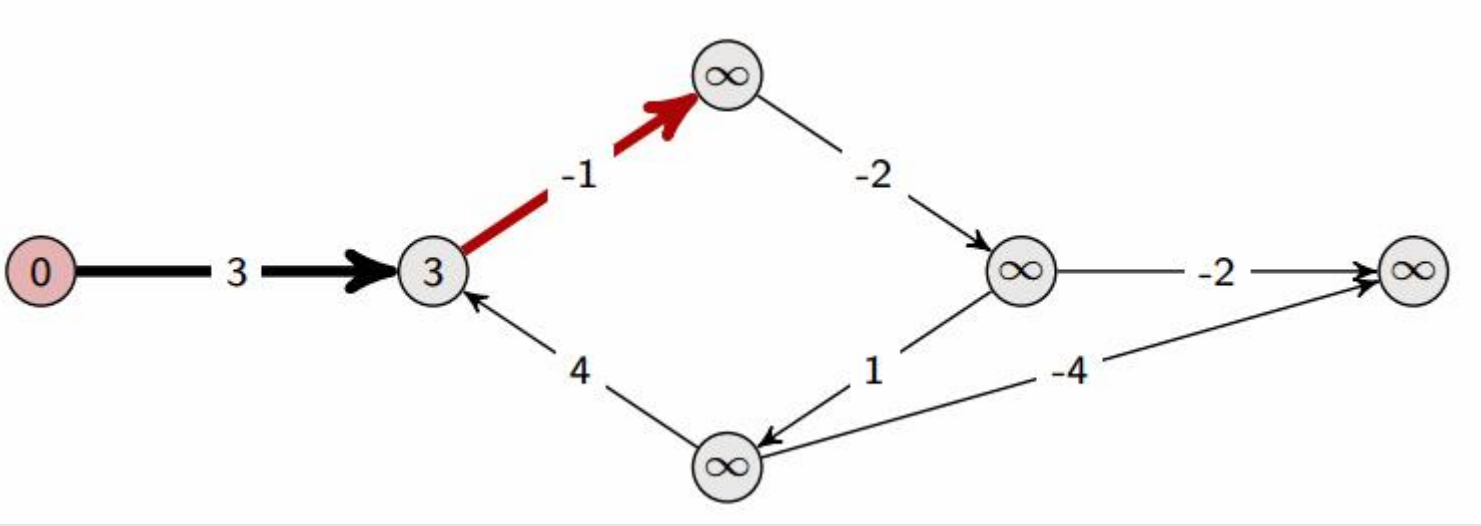
Bellman Ford



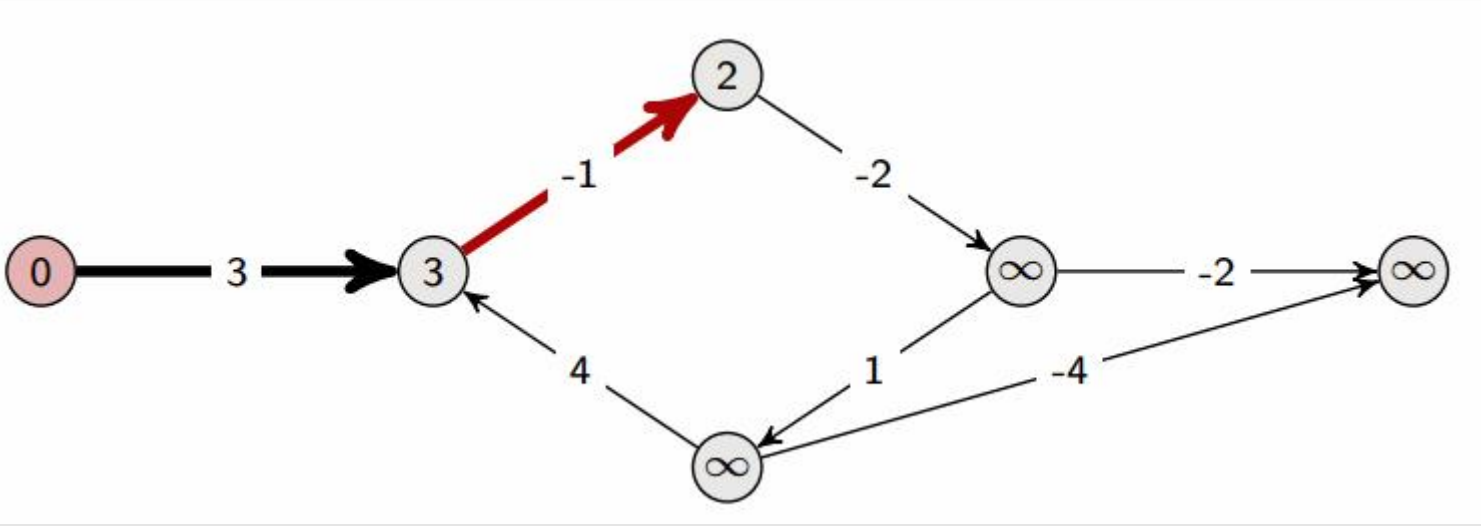
Bellman Ford



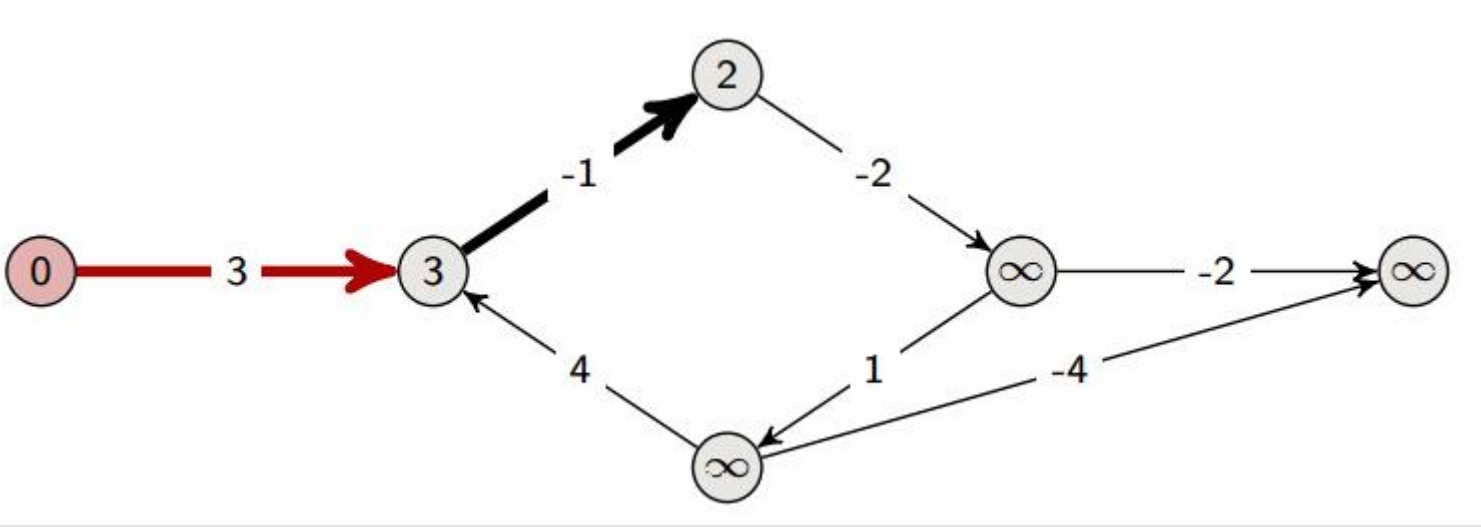
Bellman Ford



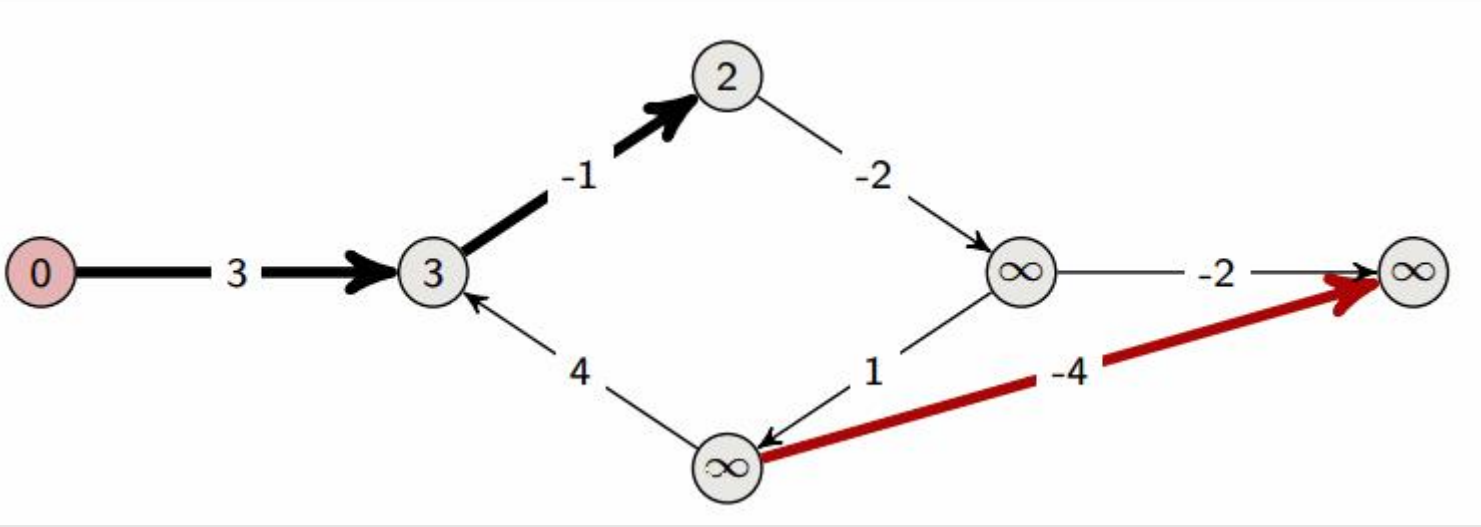
Bellman Ford



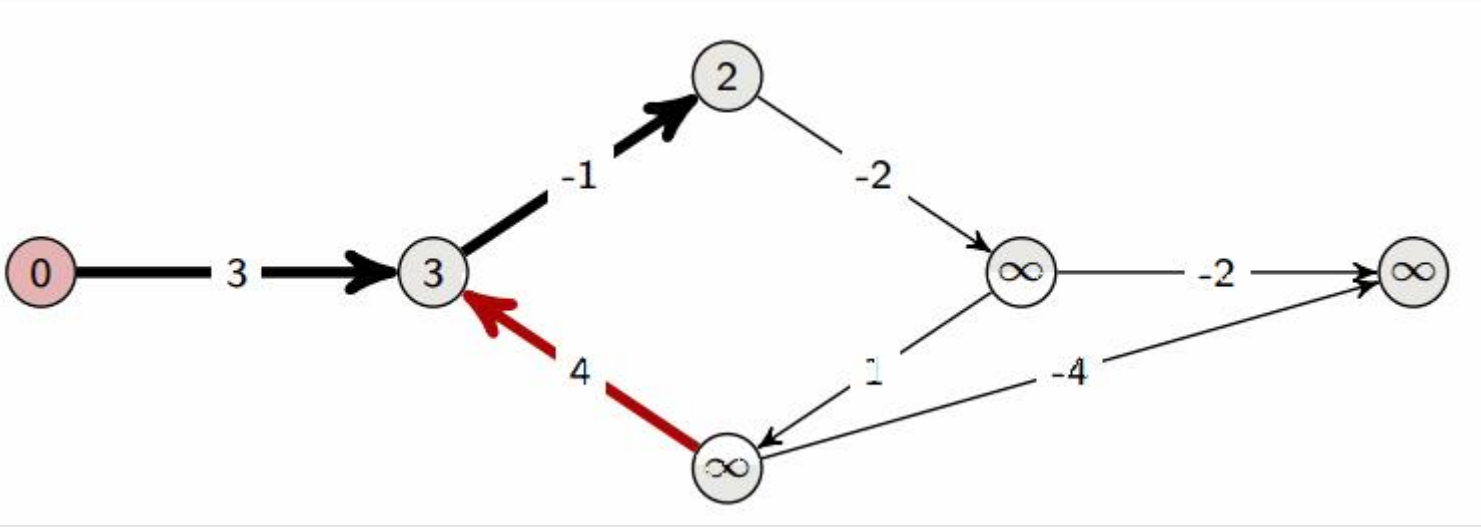
Bellman Ford



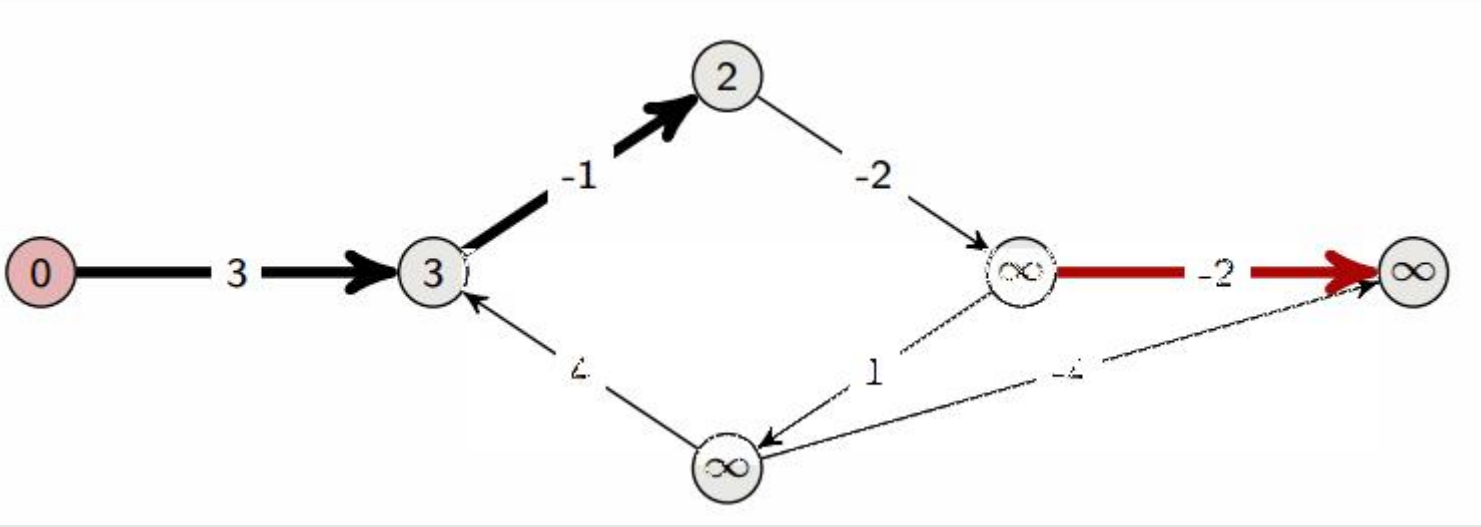
Bellman Ford



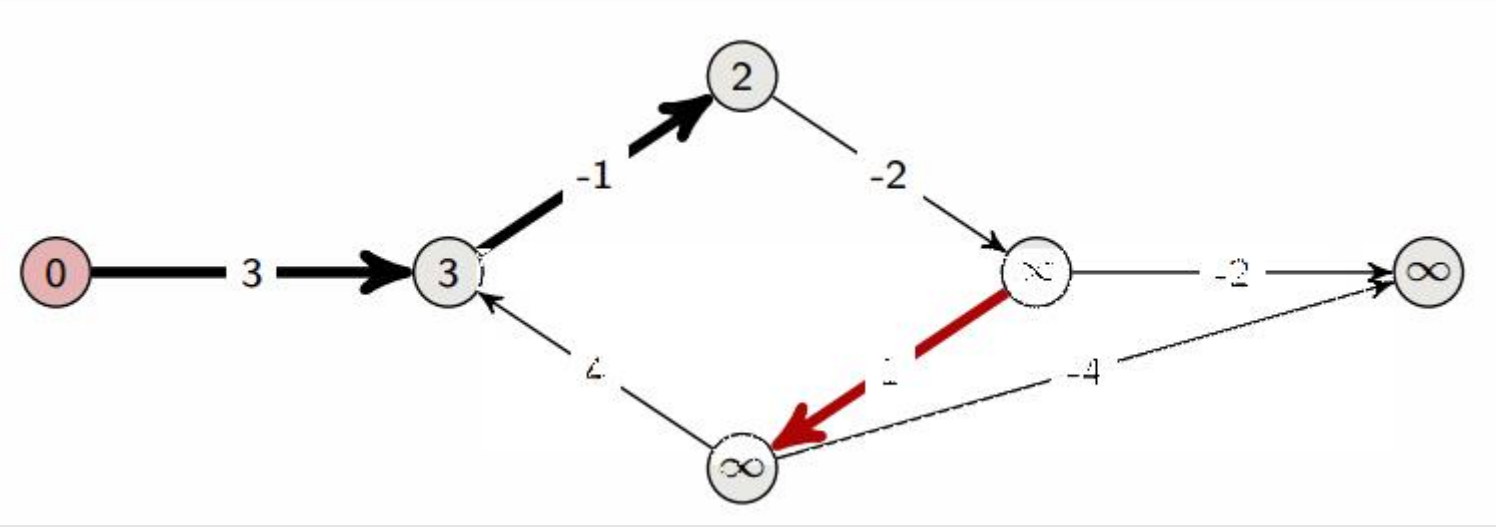
Bellman Ford



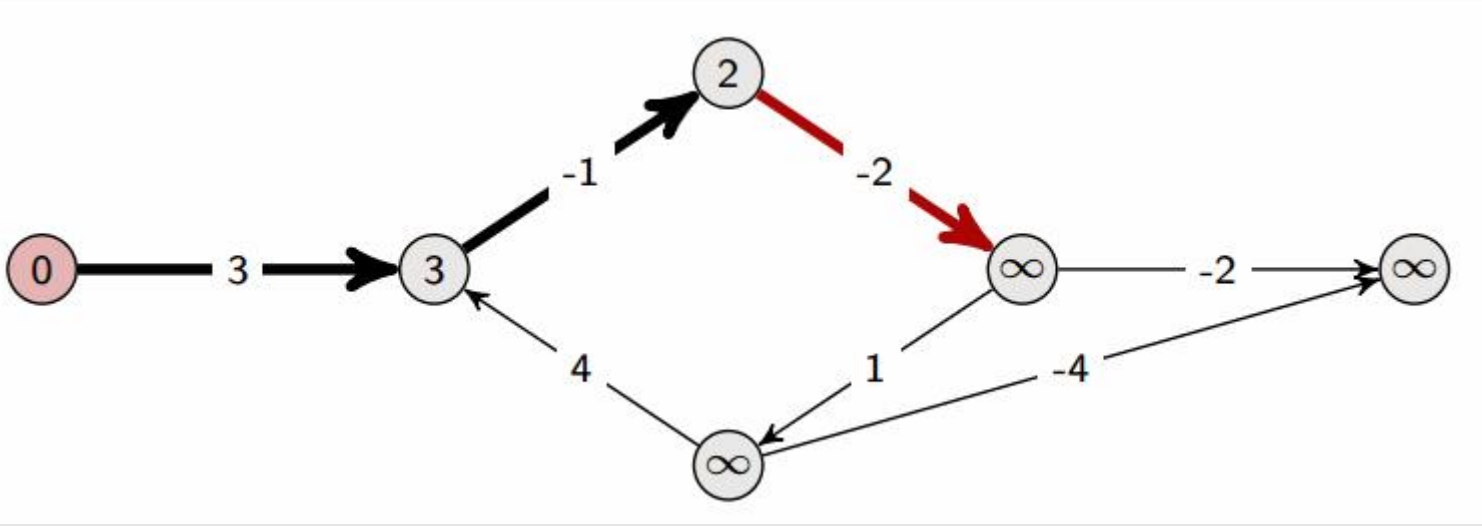
Bellman Ford



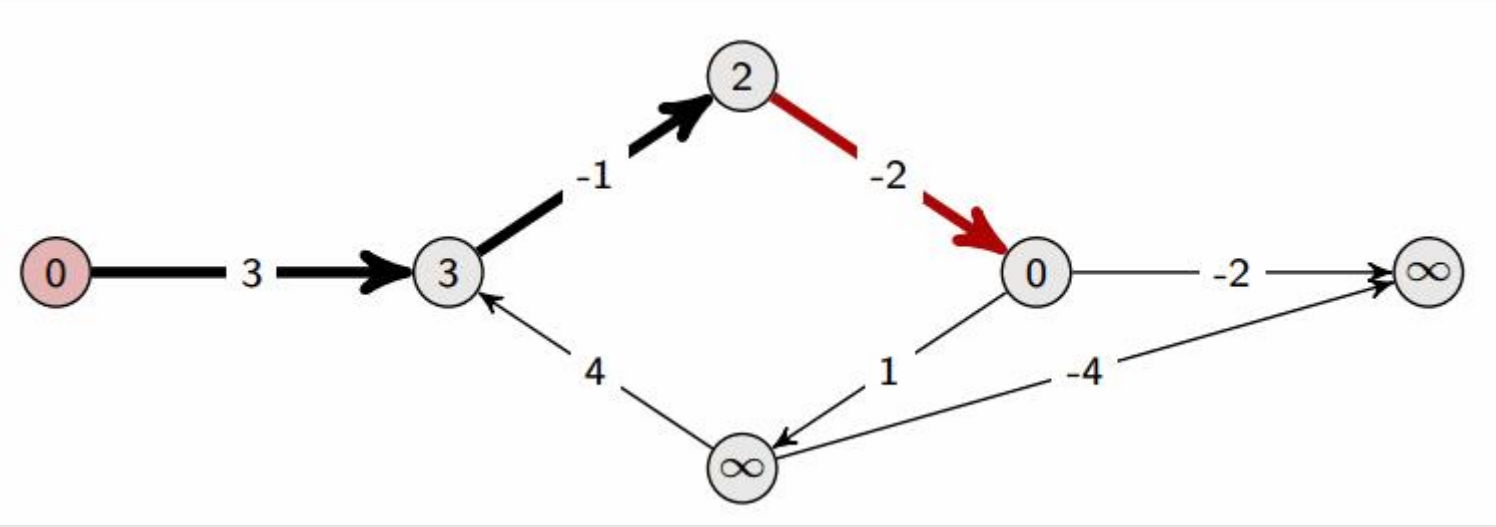
Bellman Ford



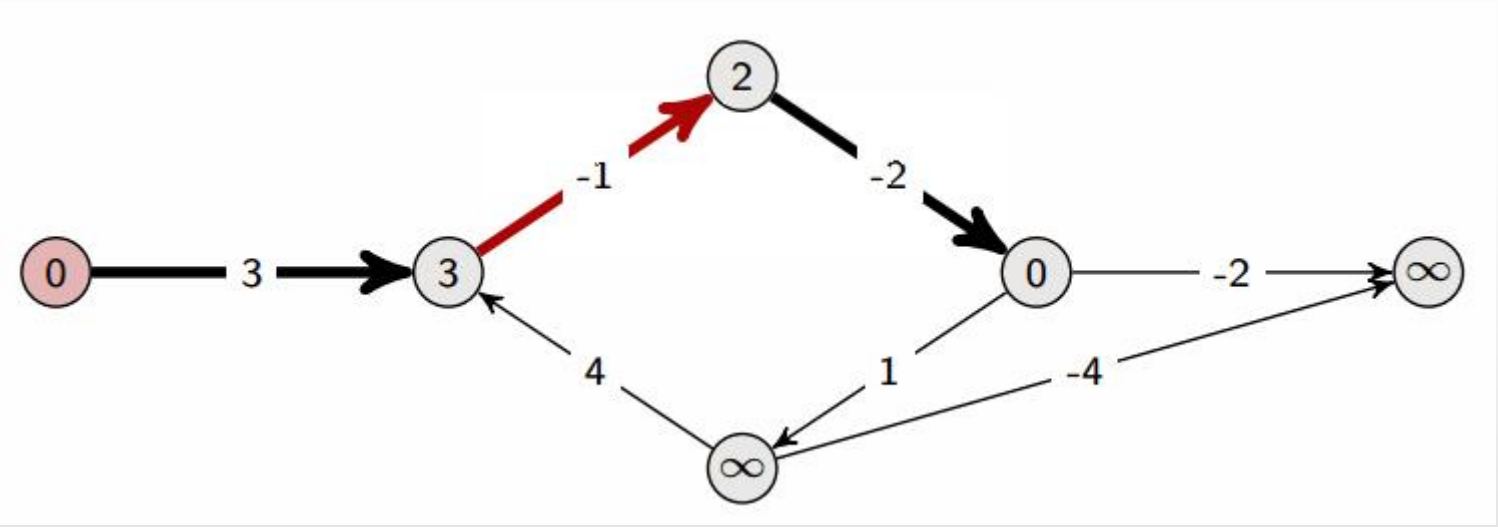
Bellman Ford



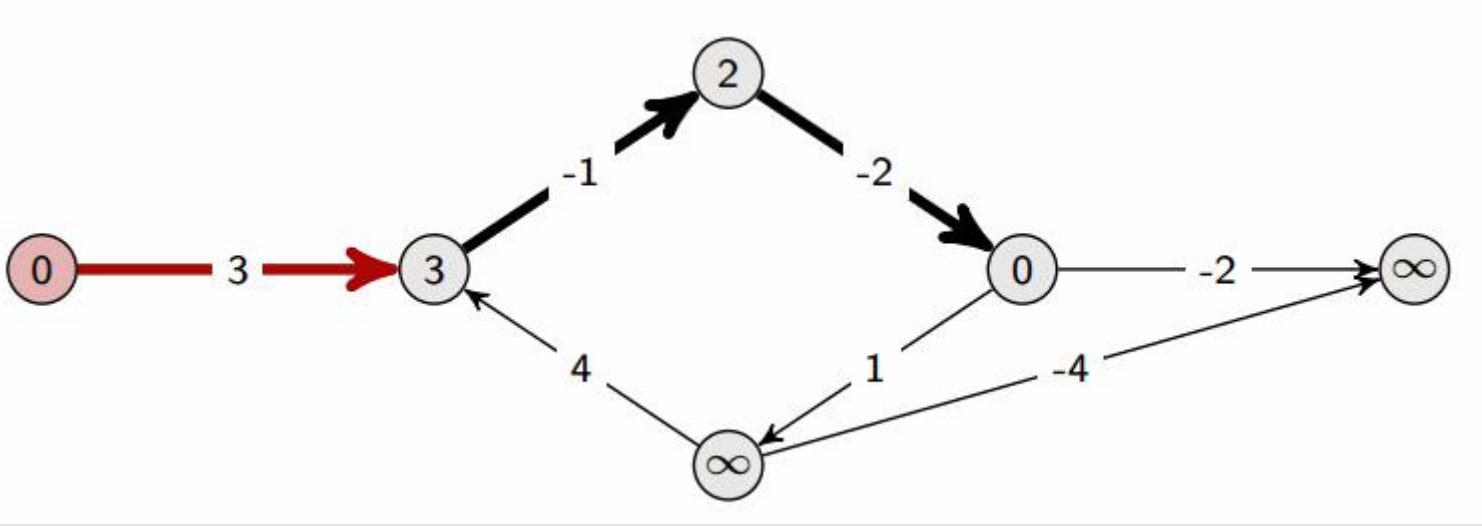
Bellman Ford



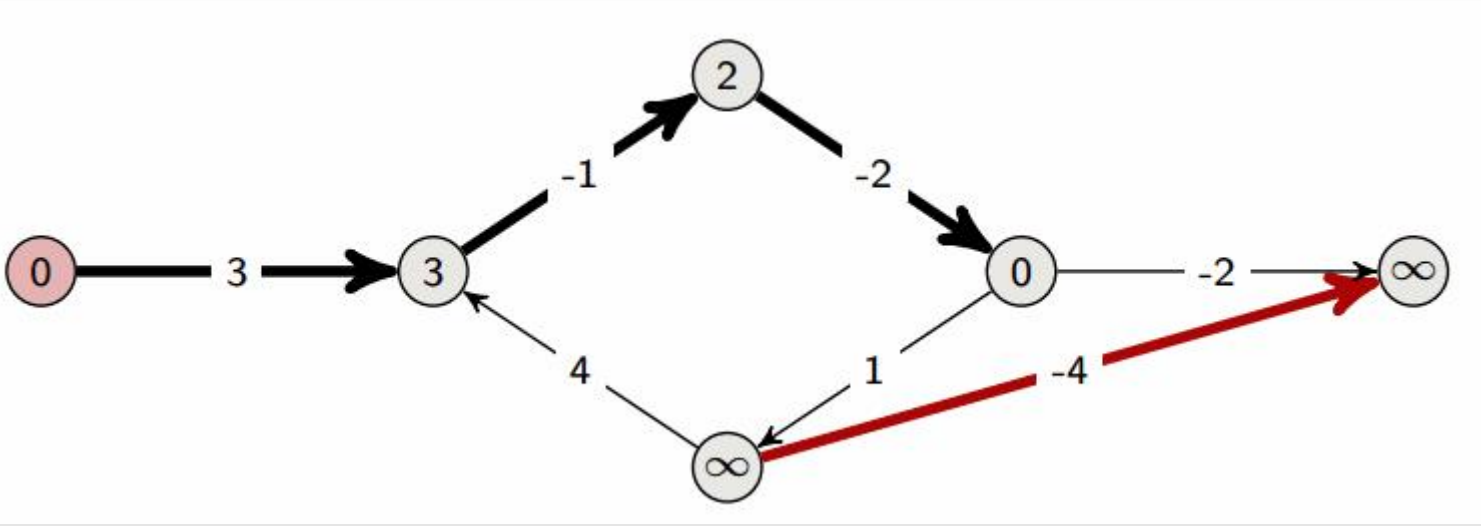
Bellman Ford



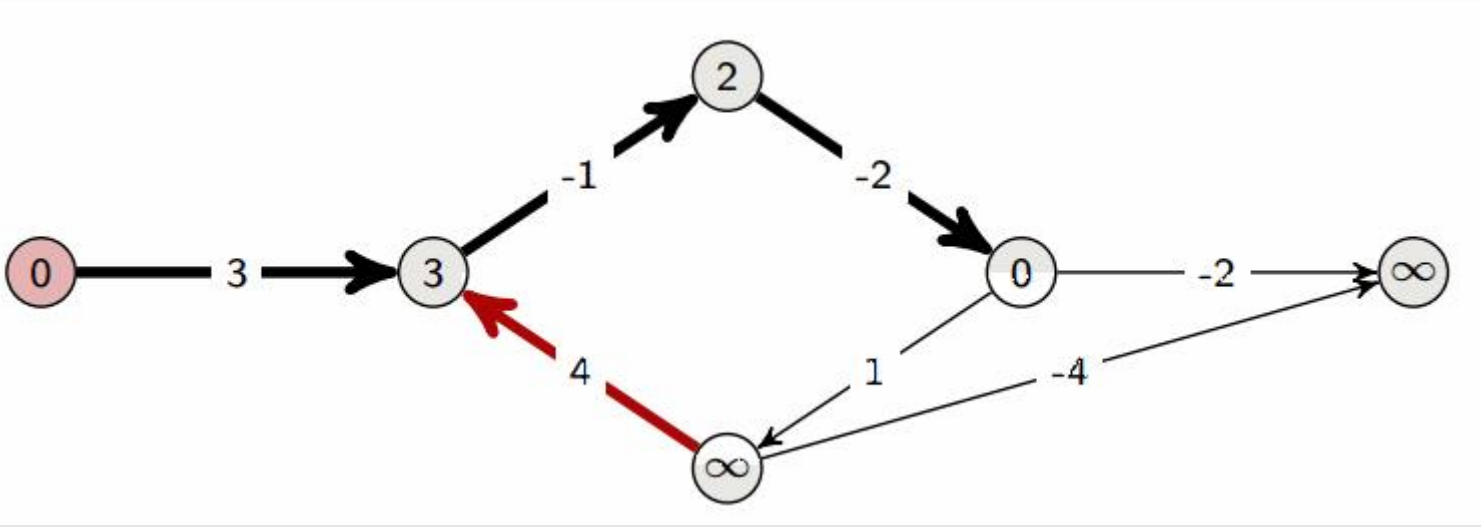
Bellman Ford



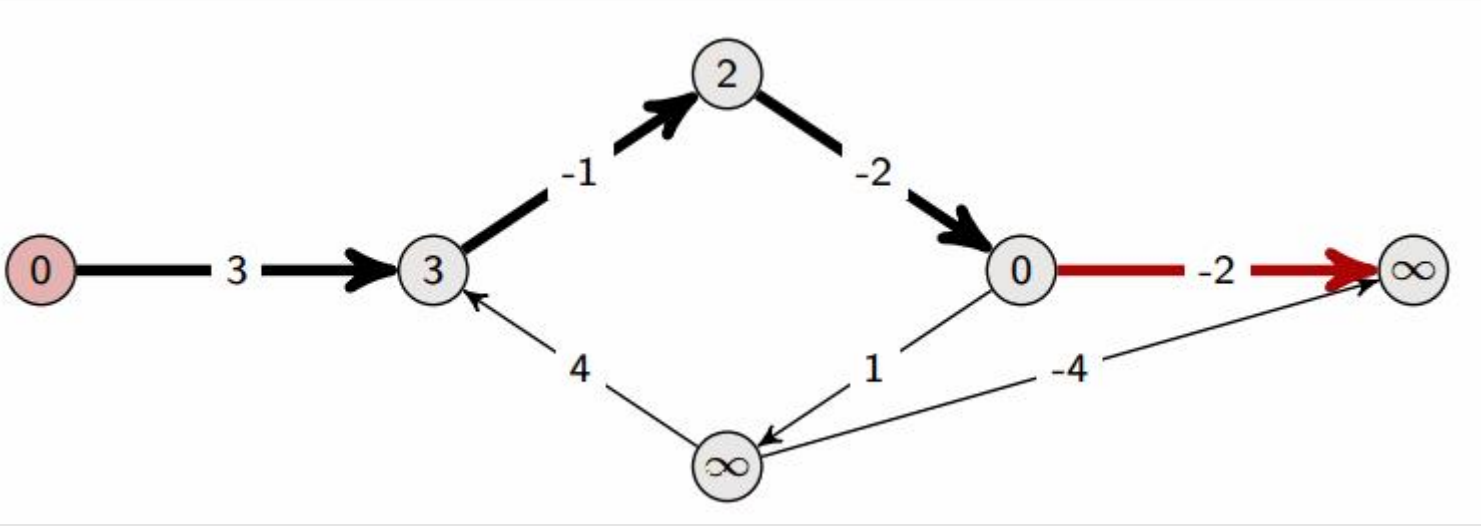
Bellman Ford



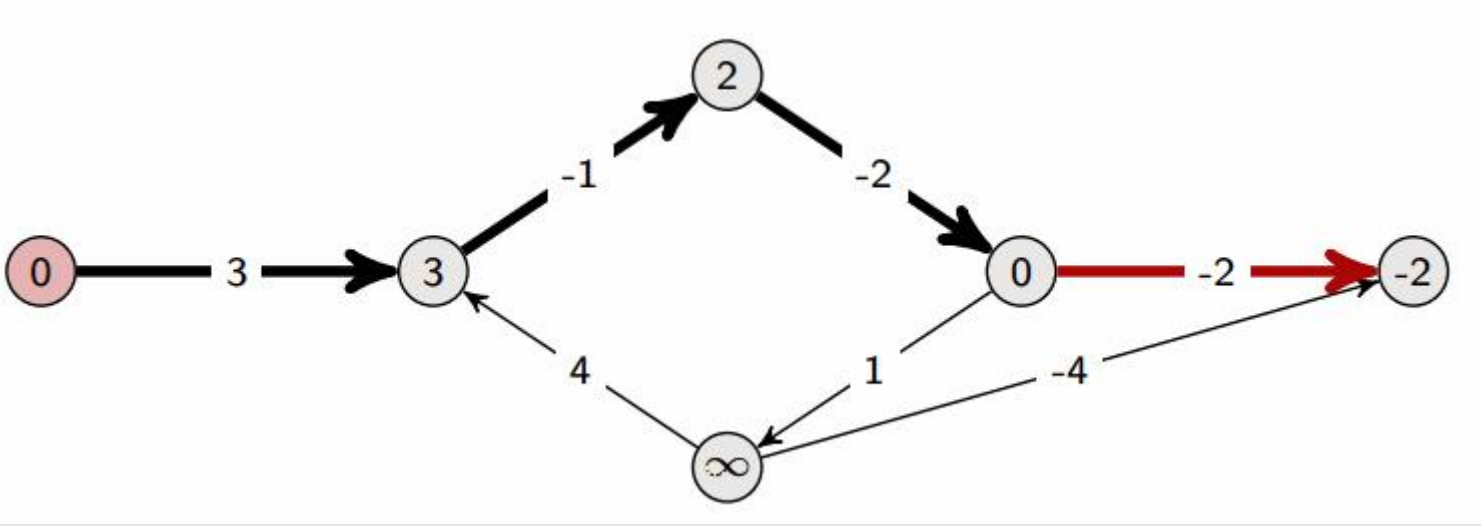
Bellman Ford



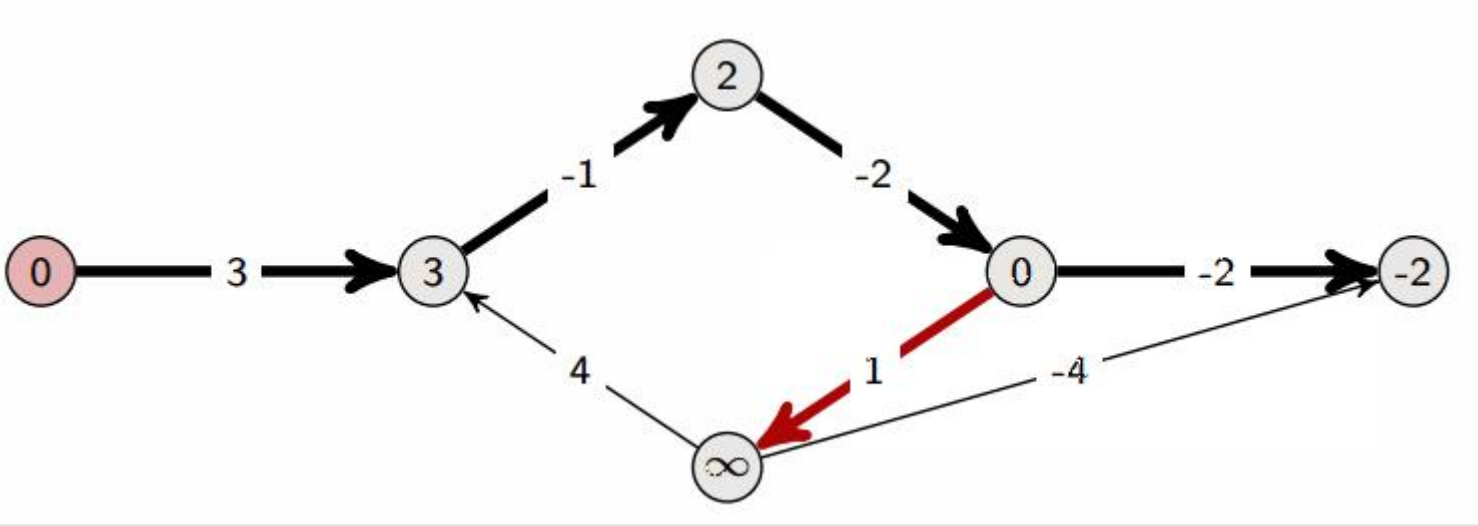
Bellman Ford



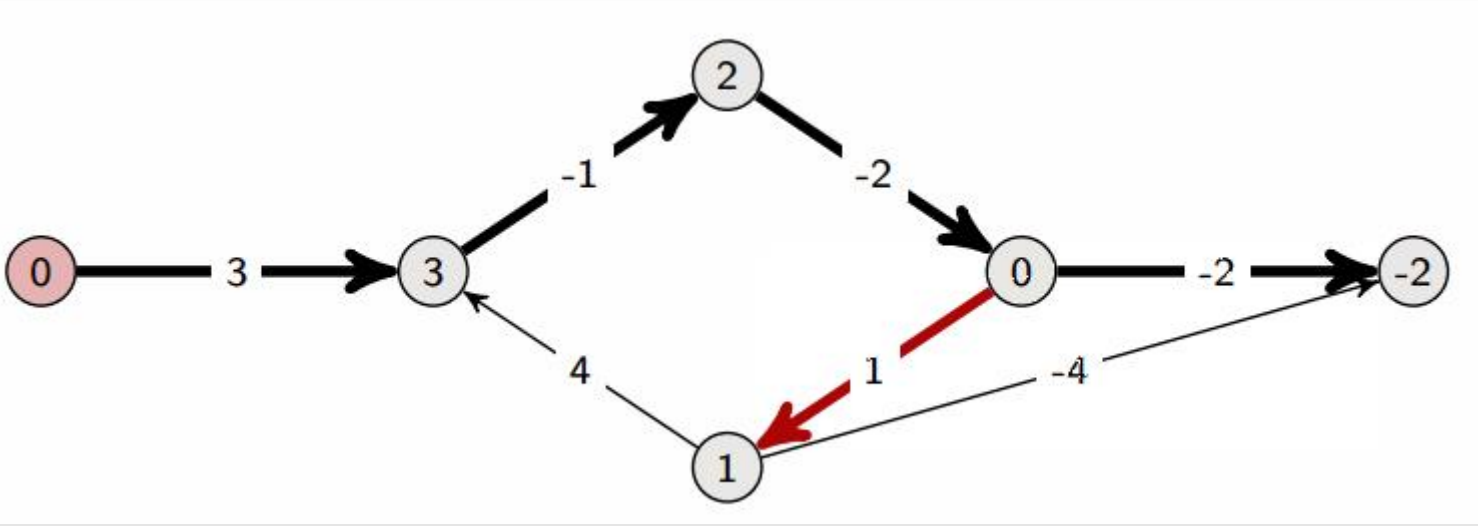
Bellman Ford



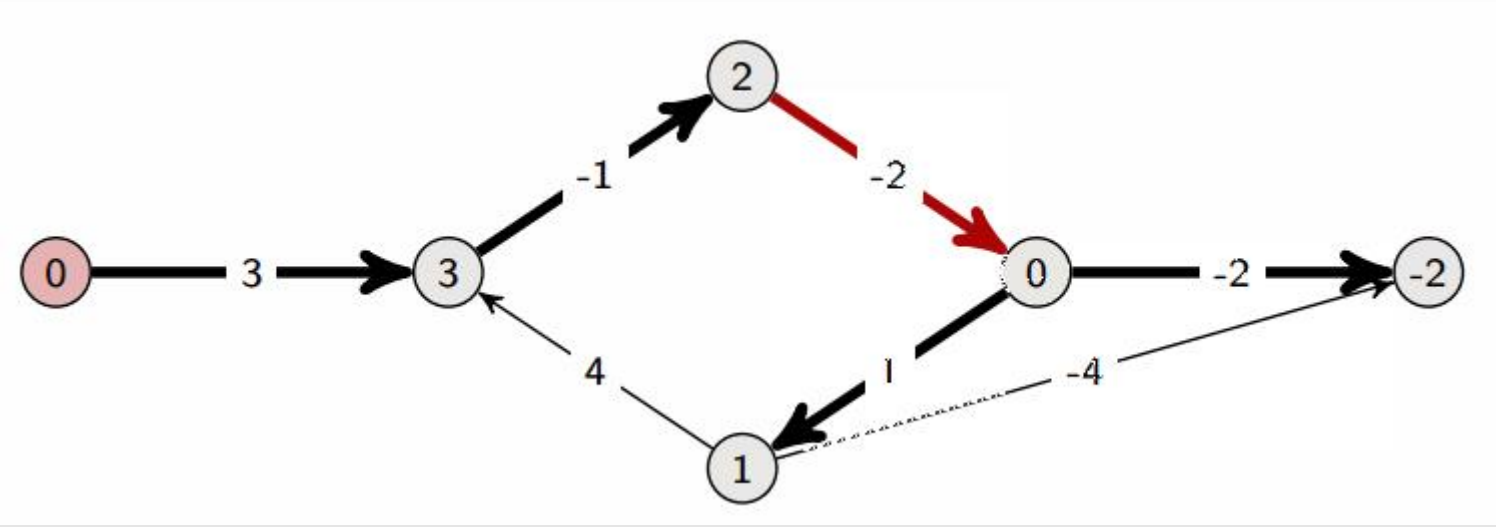
Bellman Ford



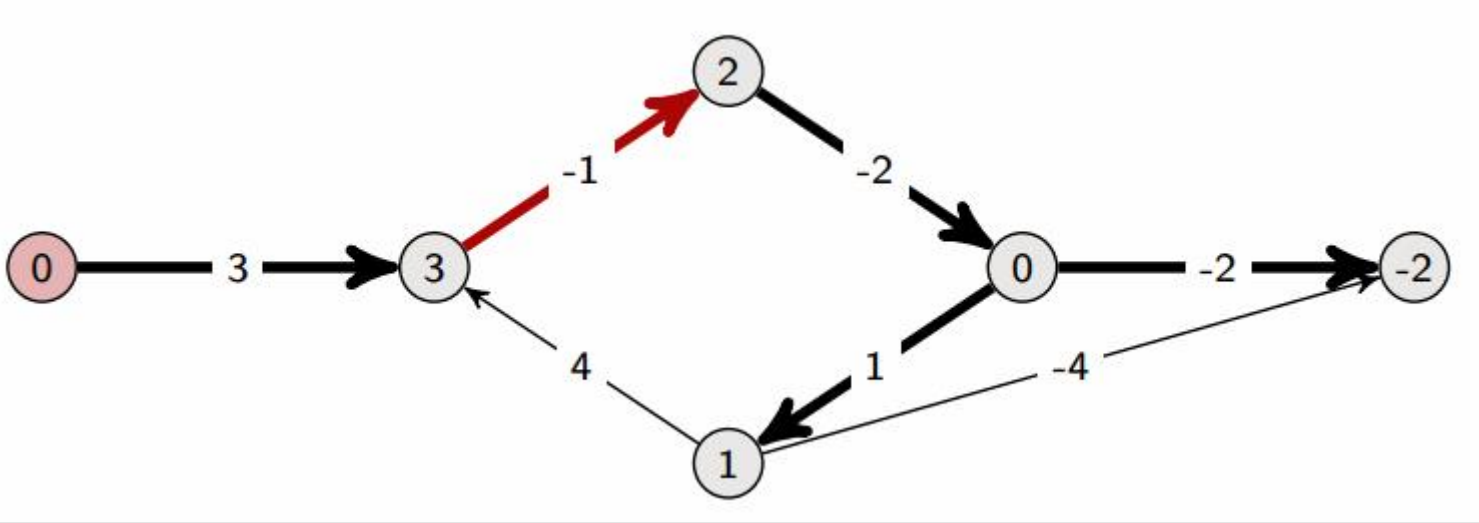
Bellman Ford



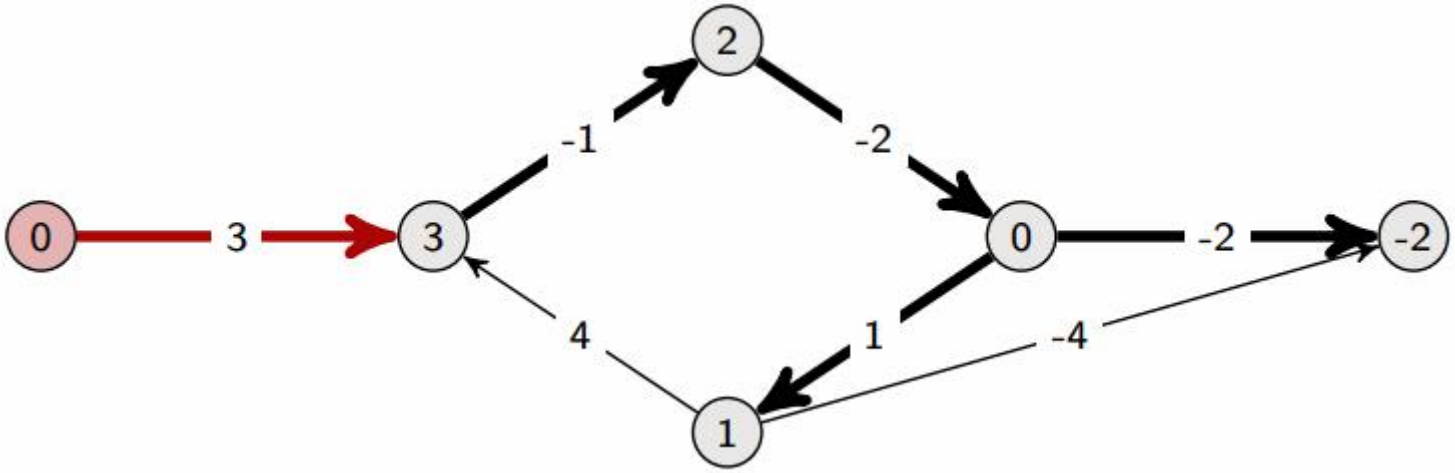
Bellman Ford



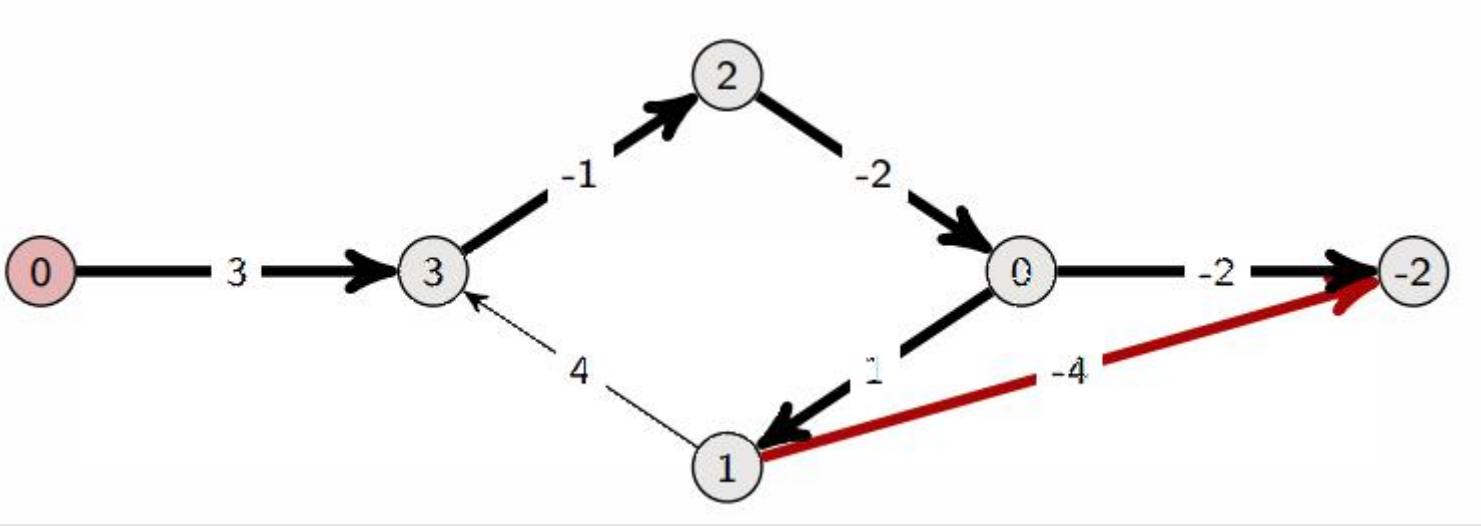
Bellman Ford



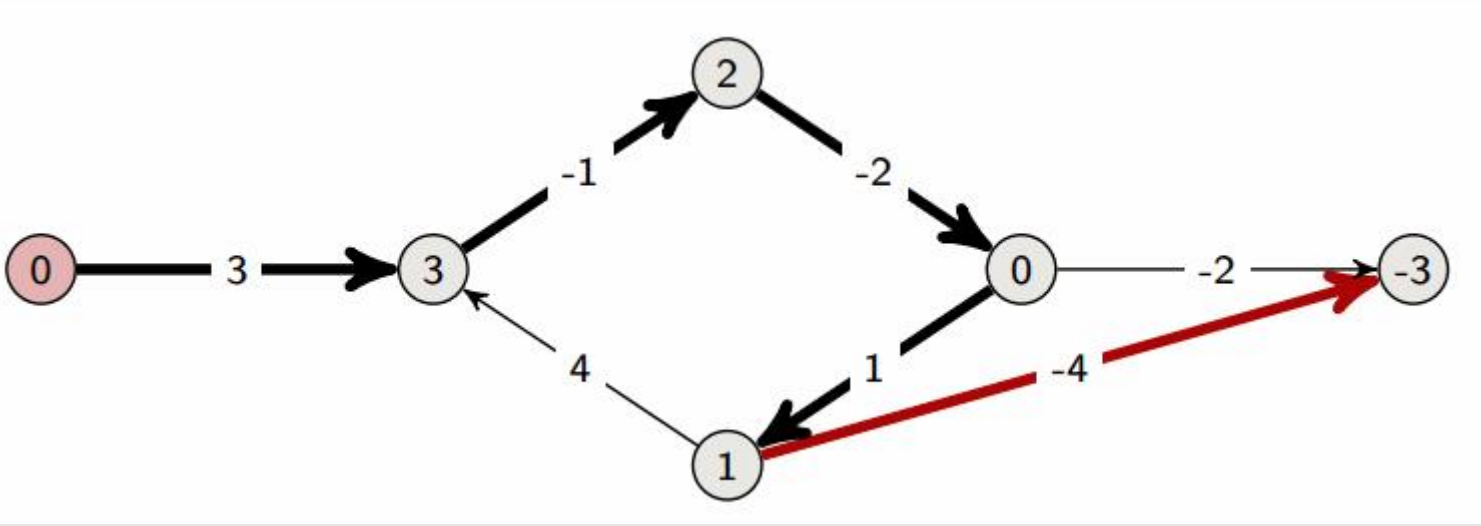
Bellman Ford



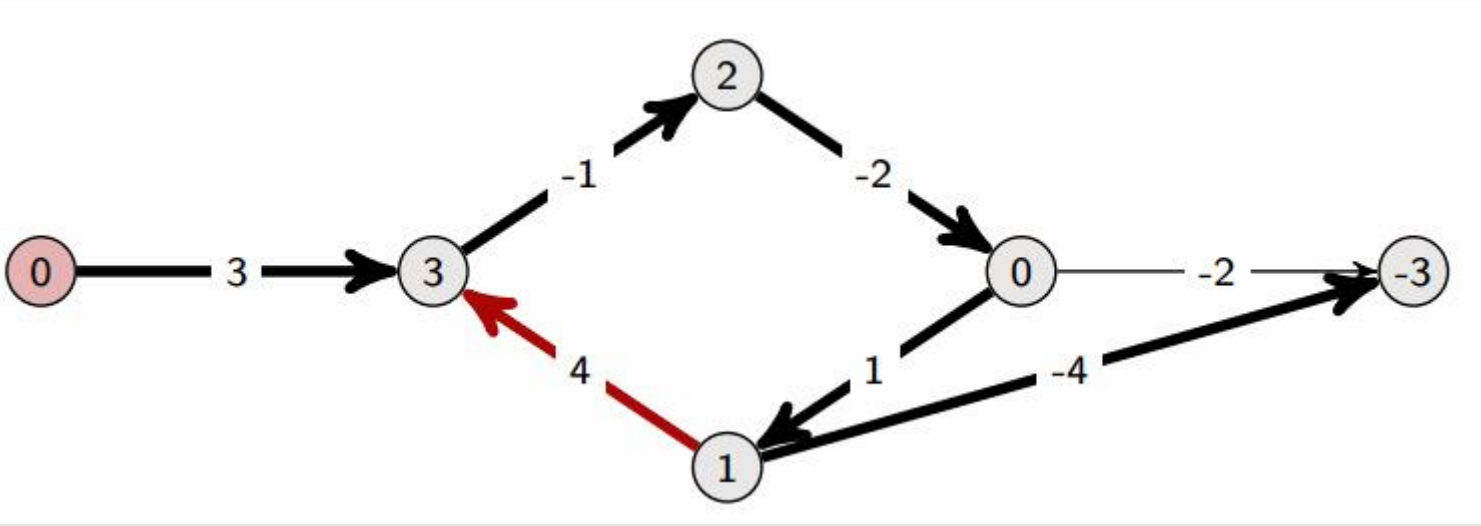
Bellman Ford



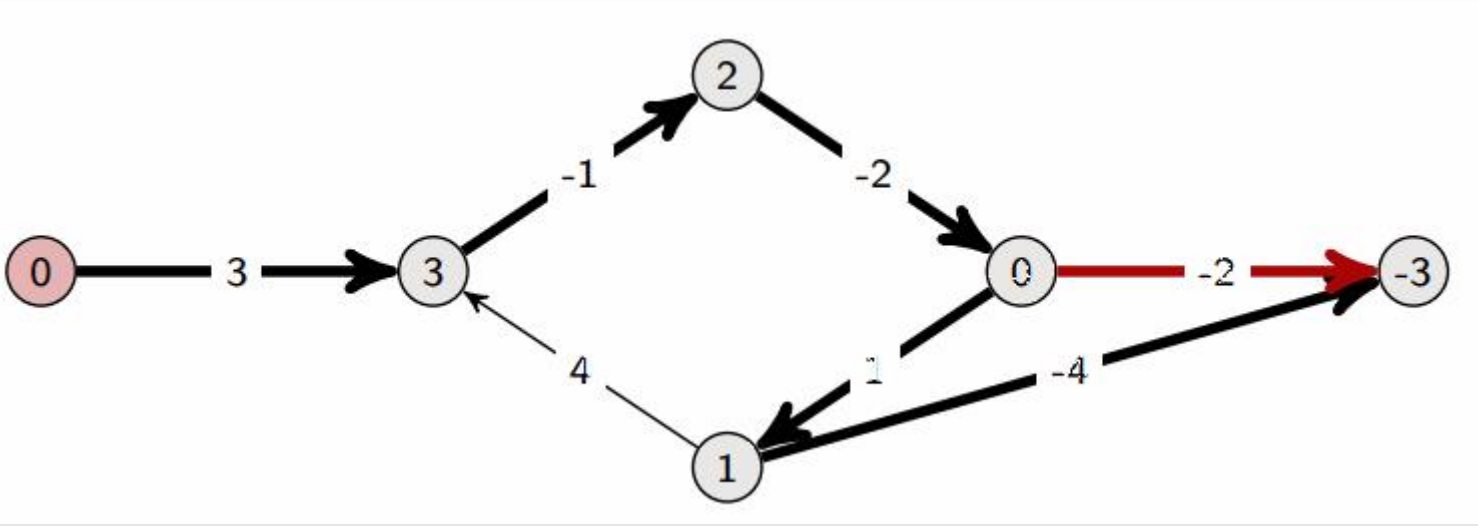
Bellman Ford



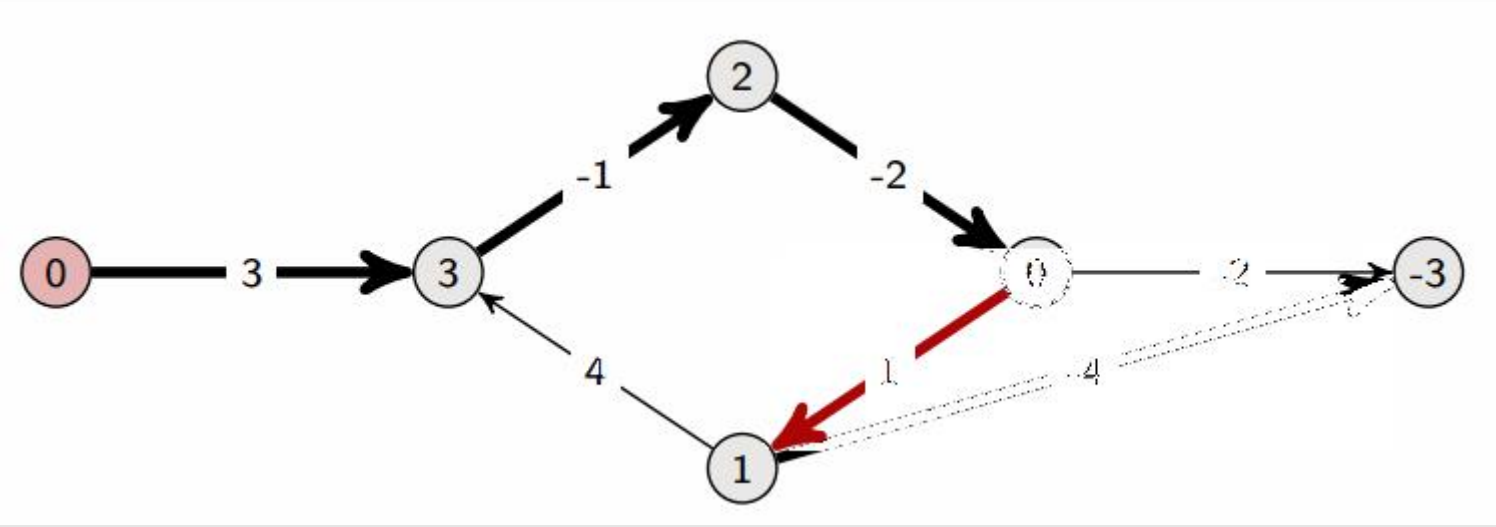
Bellman Ford



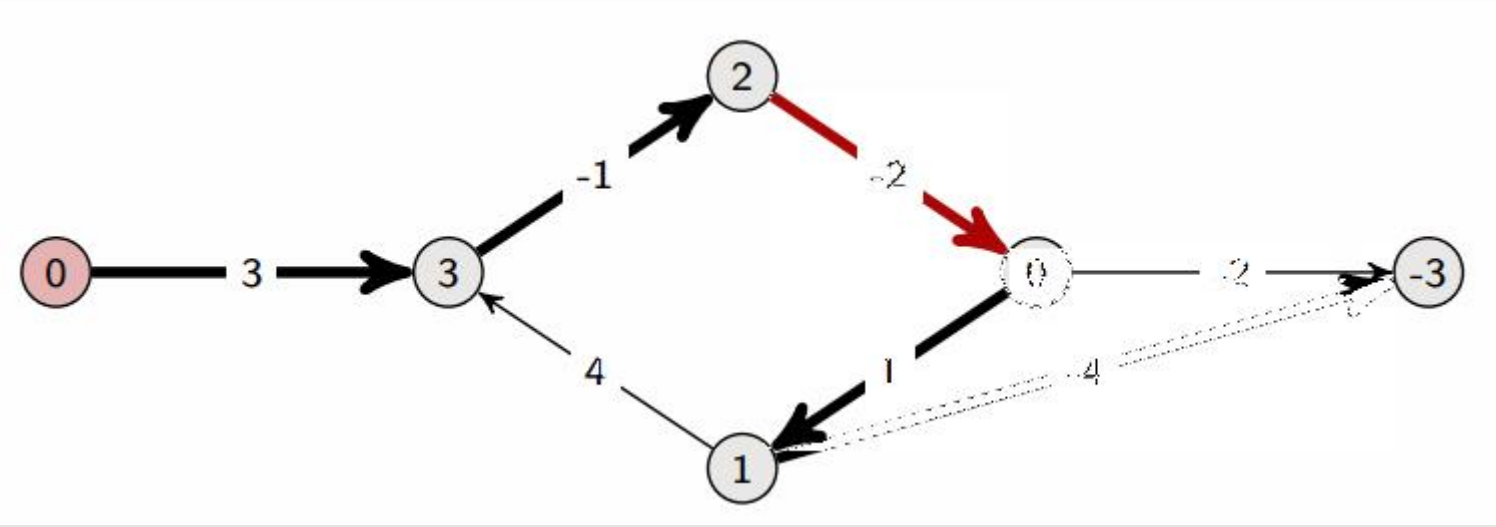
Bellman Ford



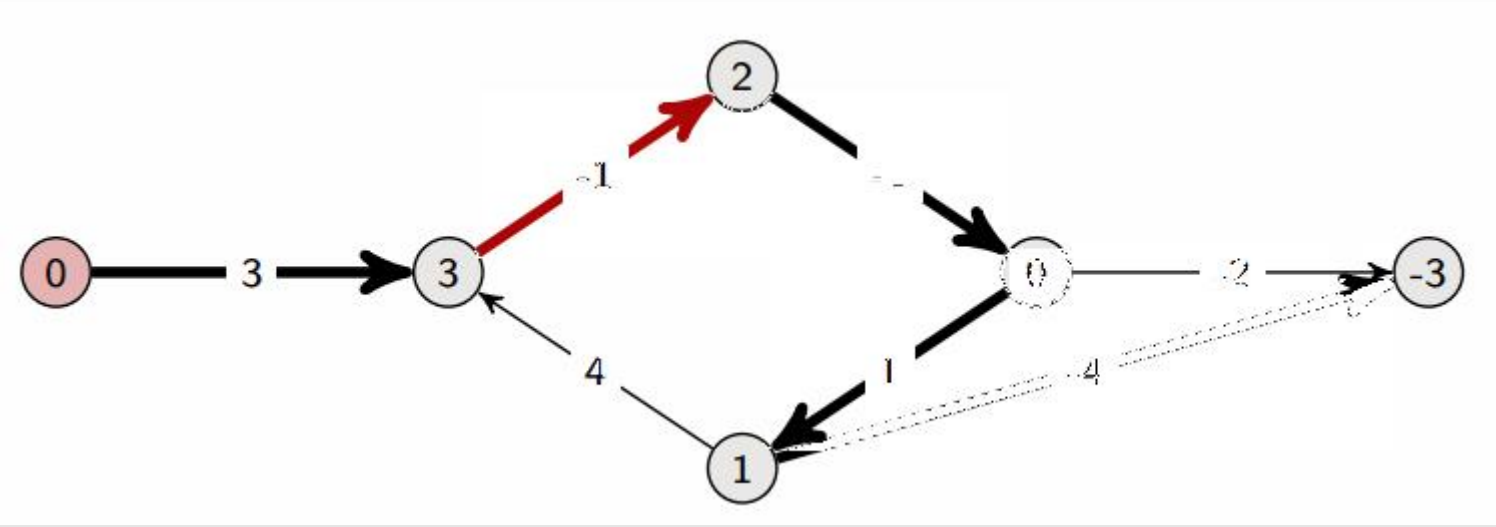
Bellman Ford



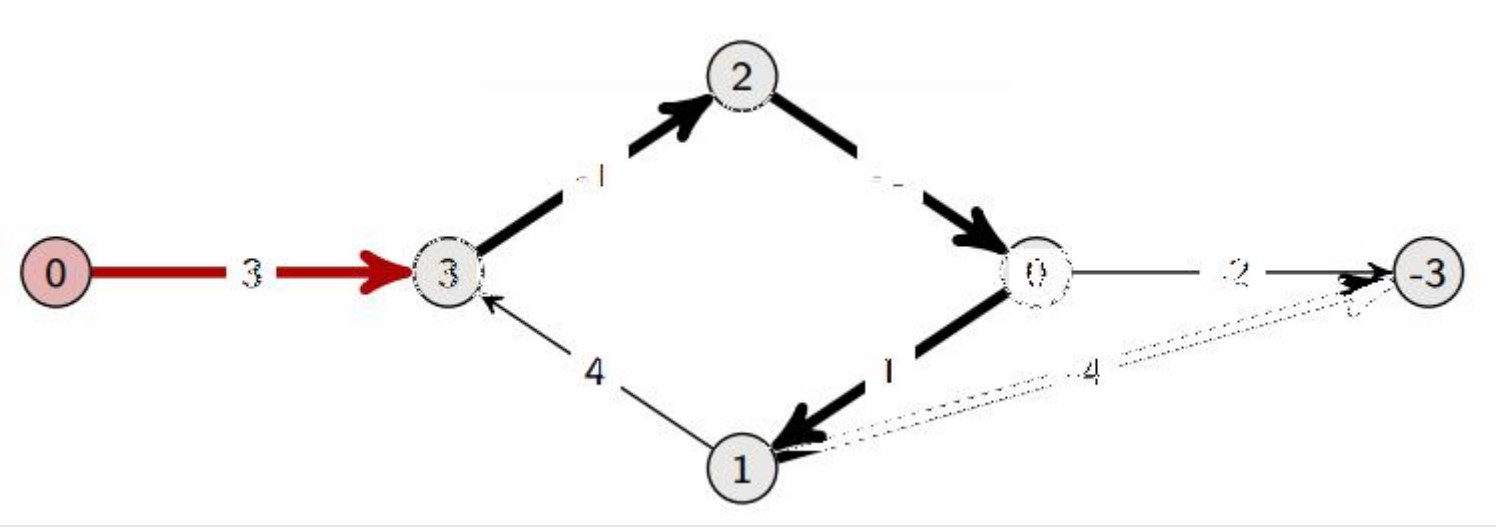
Bellman Ford



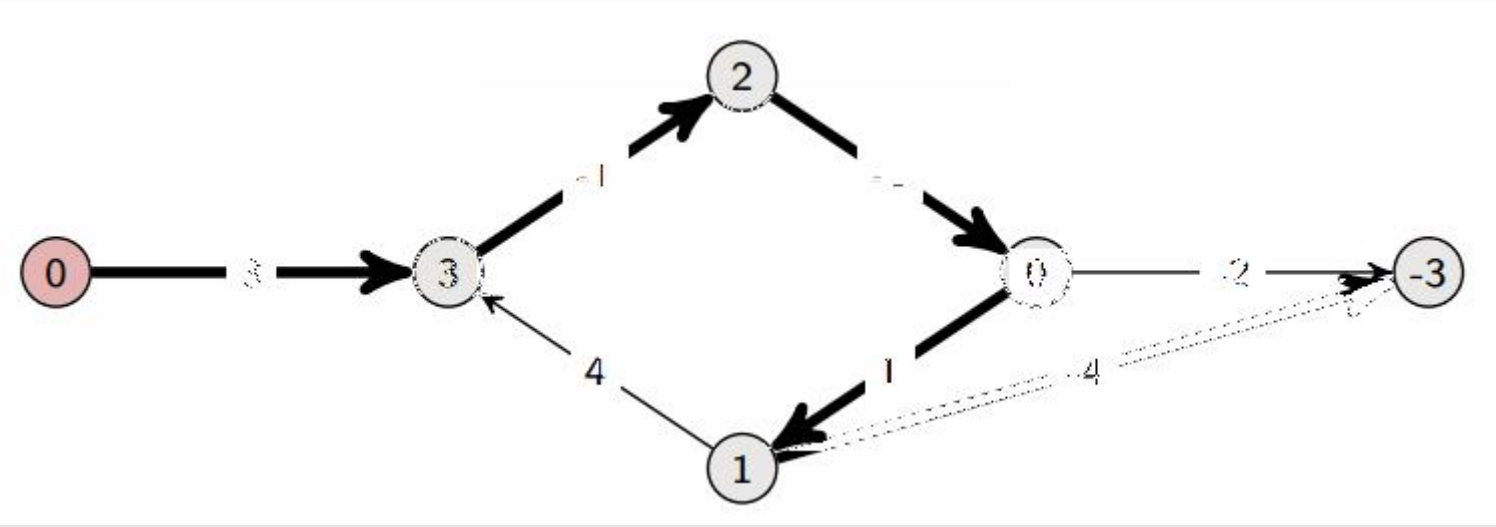
Bellman Ford



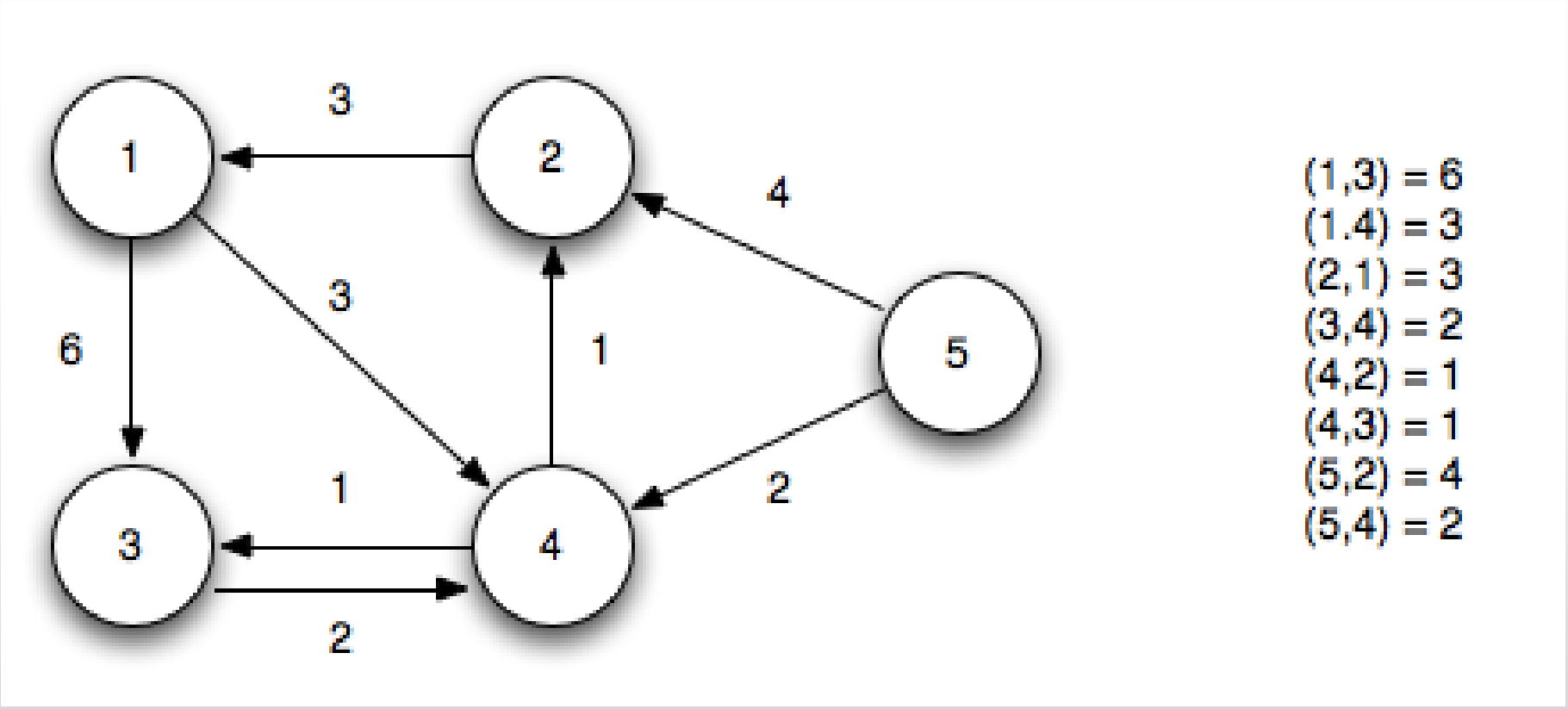
Bellman Ford



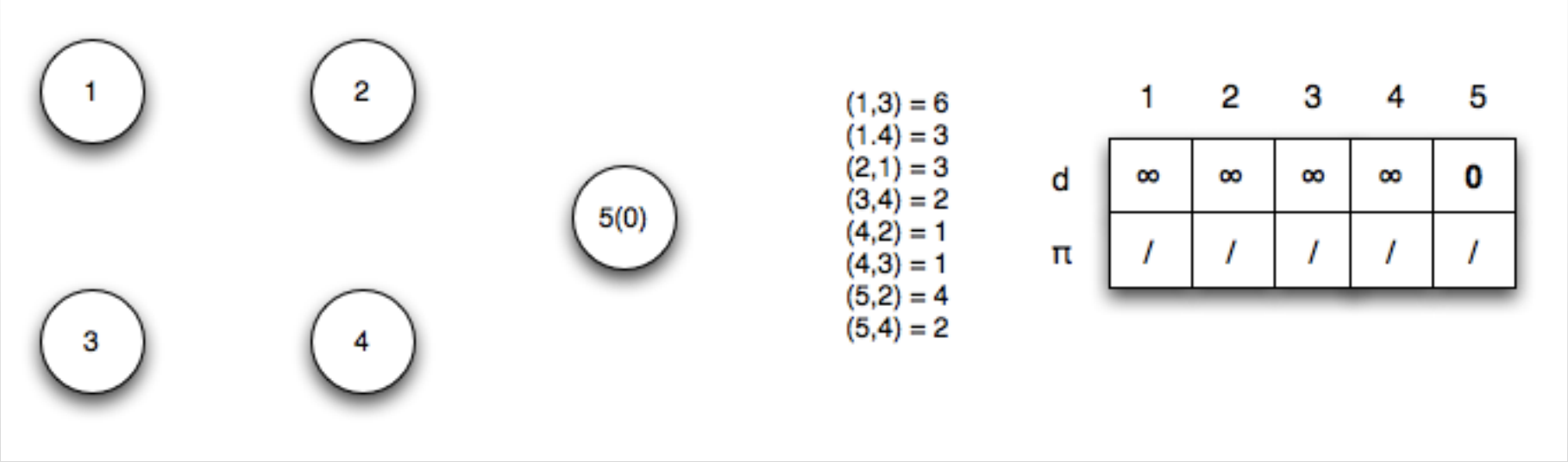
Bellman Ford



Örnek

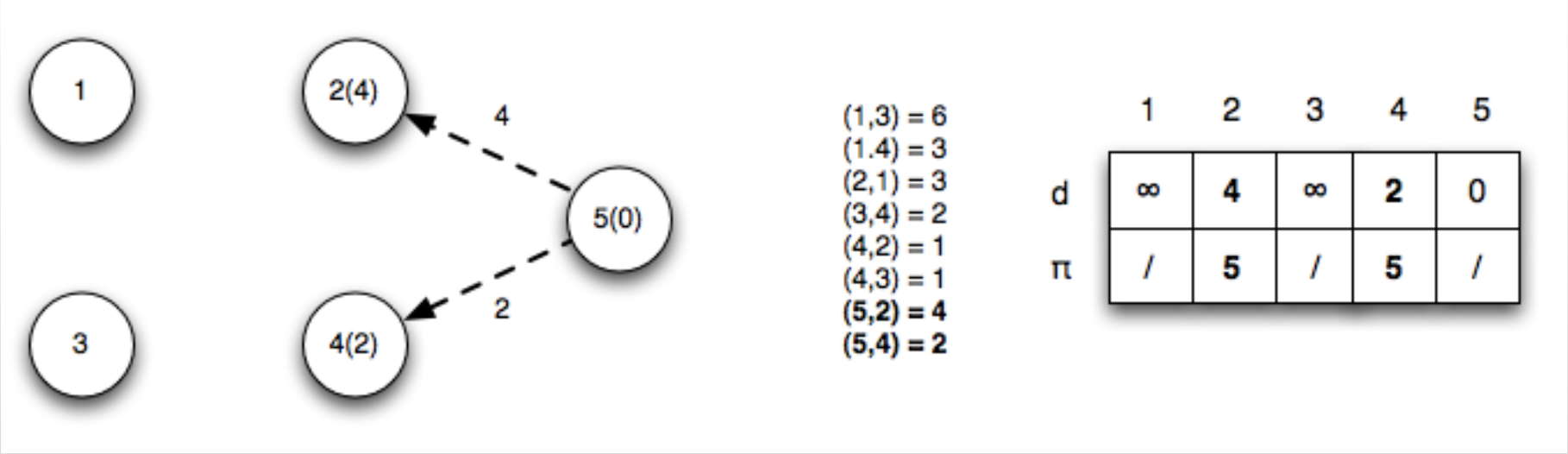


- Using vertex 5 as the source (setting its distance to 0), we initialize all the other distances to ∞.



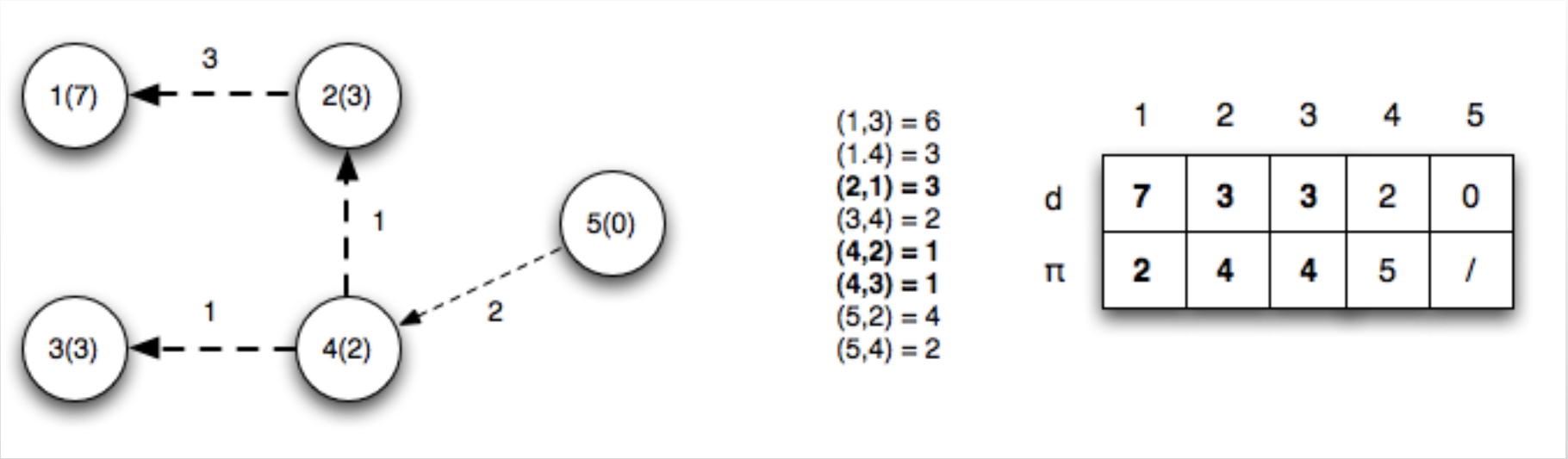
	1	2	3	4	5
d	∞	∞	∞	∞	**0**
π	/	/	/	/	/

- Iteration 1: Edges (u5,u2) and (u5,u4) relax updating the distances to 2 and 4

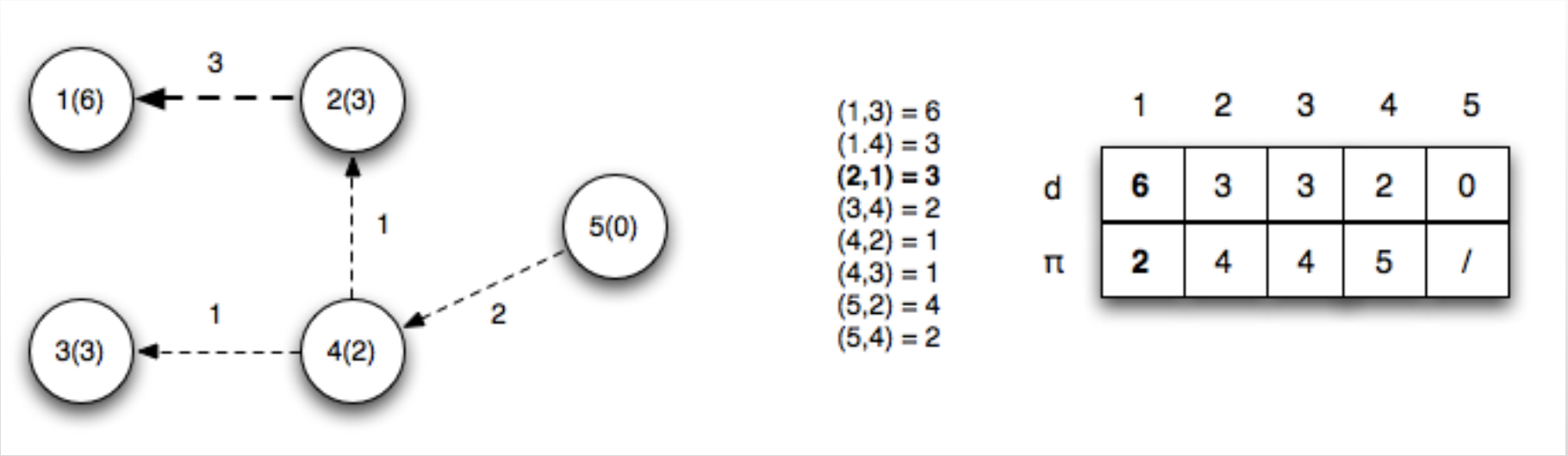


	1	2	3	4	5
d	∞	**4**	∞	**2**	0
π	/	**5**	/	**5**	/

- Iteration 2: Edges (u2,u1), (u4,u2) and (u4,u3) relax updating the distances to 1, 2, and 4 respectively. Note edge (u4,u2) finds a shorter path to vertex 2 by going through vertex 4

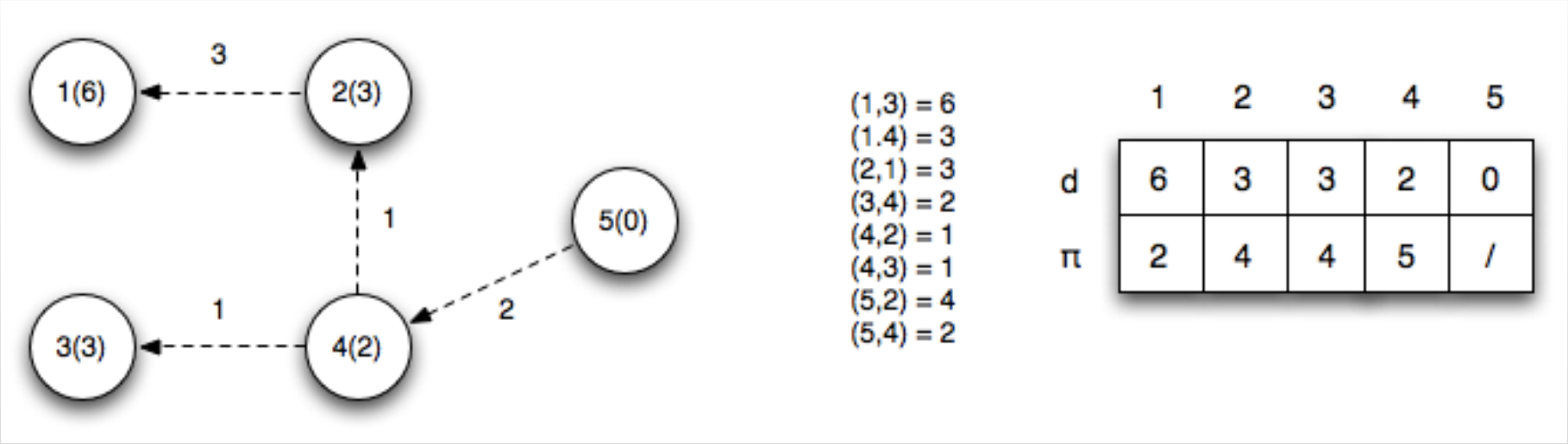


- Iteration 3: Edge (u2,u1) relaxes (since a shorter path to vertex 2 was found in the previous iteration) updating the distance to 1



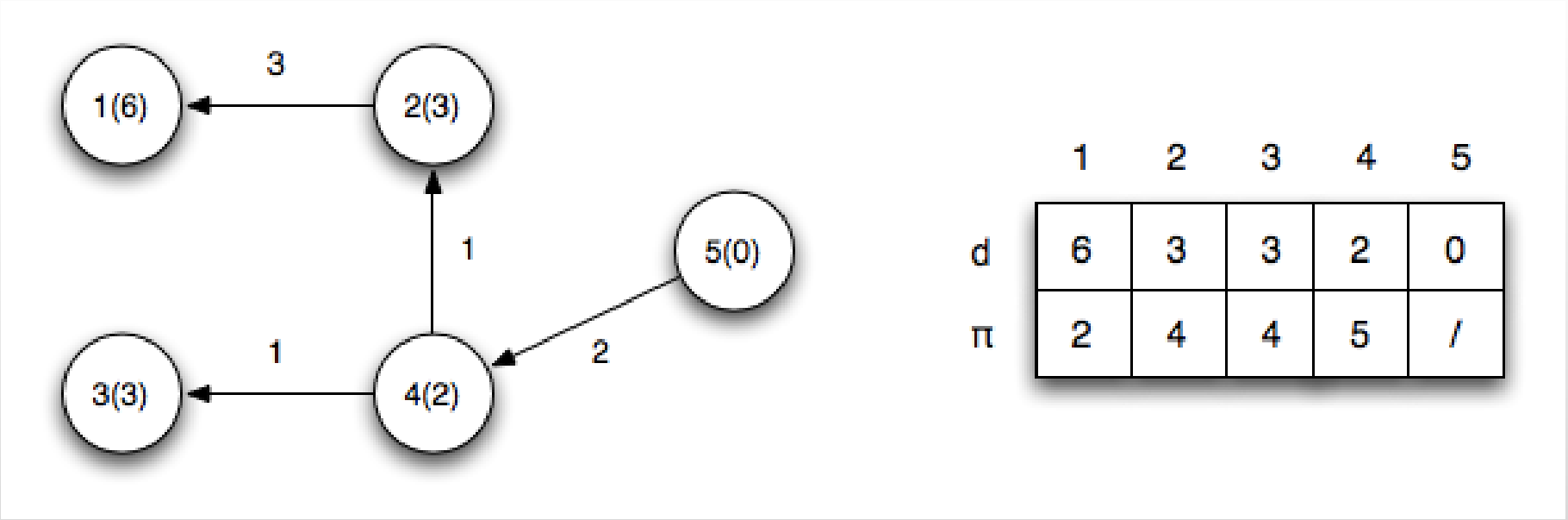
	1	2	3	4	5
d	**6**	3	3	2	0
π	**2**	4	4	5	/

- Iteration 4: No edges relax



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

- The final shortest paths from vertex 5 with corresponding distances is



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

- Negative cycle checks: We now check the relaxation condition one additional time for each edge. If any of the checks pass then there exists a negative weight cycle in the graph.

$$v_3.d > u_1.d + w(1,3) \Rightarrow 4 \not> 6 + 6 = 12 \checkmark$$

$$v_4.d > u_1.d + w(1,4) \Rightarrow 2 \not> 6 + 3 = 9 \checkmark$$

$$v_1.d > u_2.d + w(2,1) \Rightarrow 6 \not> 3 + 3 = 6 \checkmark$$

$$v_4.d > u_3.d + w(3,4) \Rightarrow 2 \not> 3 + 2 = 5 \checkmark$$

$$v_2.d > u_4.d + w(4,2) \Rightarrow 3 \not> 2 + 1 = 3 \checkmark$$

$$v_3.d > u_4.d + w(4,3) \Rightarrow 3 \not> 2 + 1 = 3 \checkmark$$

$$v_2.d > u_5.d + w(5,2) \Rightarrow 3 \not> 0 + 4 = 4 \checkmark$$

$$v_4.d > u_5.d + w(5,4) \Rightarrow 2 \not> 0 + 2 = 2 \checkmark$$

Floyd Warshall

- Tüm düğüm çiftleri arasındaki en kısa yolları bulur.
- 1959'da Robert Floyd tarafından bulunmuştur.
- 1962'de Stephen Warshall tarafından geliştirilmiştir.
- Negatif ağırlıklı kenarlar ve döngülerle başa çıkabilir.

Algoritma İlkeleri

- Dinamik programlama yöntemini kullanır.
- Bir matris kullanarak tüm düğümler arasındaki en kısa mesafeleri bulur.
- Bellman-Ford ve Dijkstra tek kaynaktan düğümlere en kısa yolları bulur.
- Floyd Warshall, tüm çiftler arasındaki en kısa yolları hesaplar.

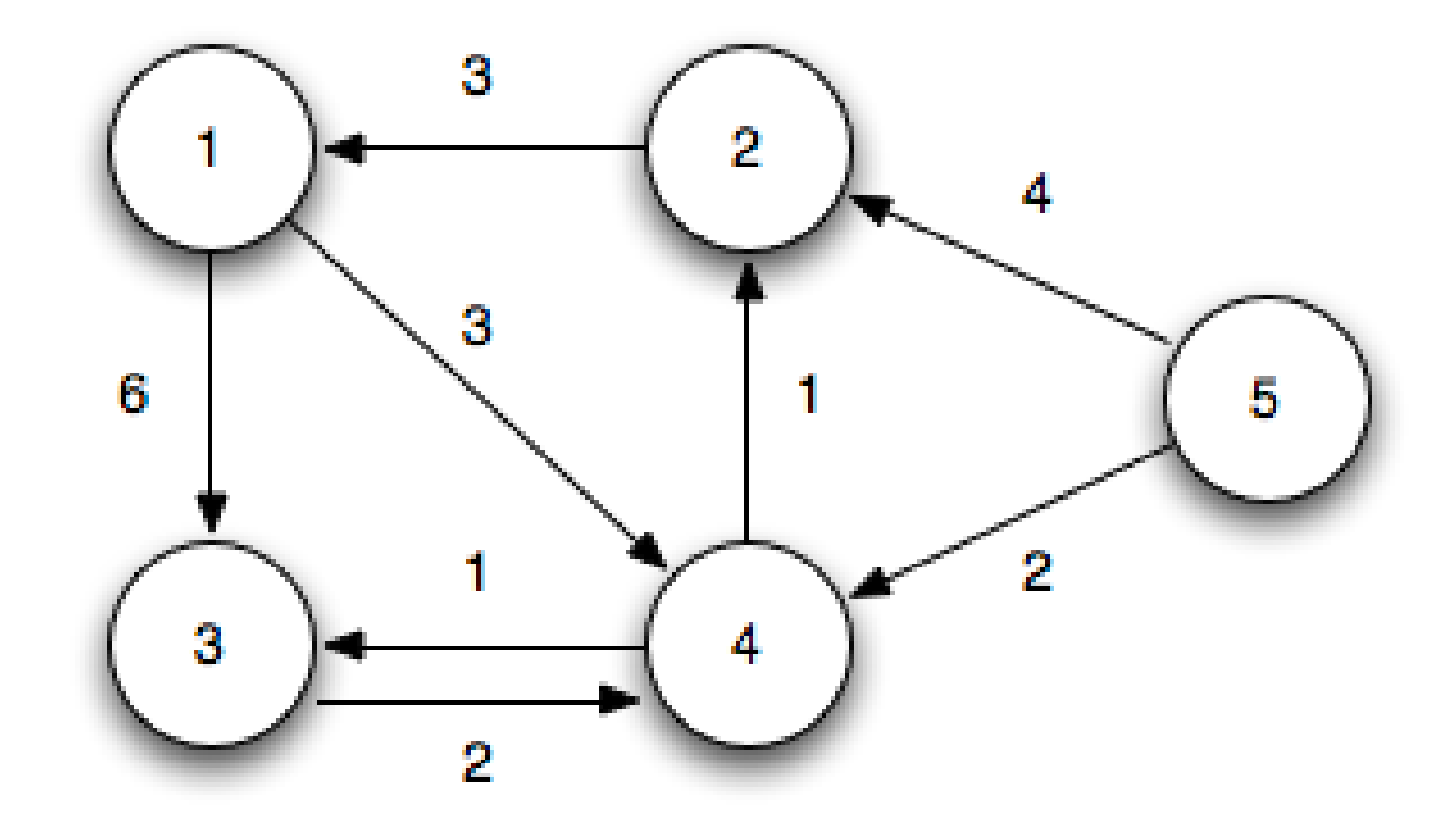
Algoritma Adımları

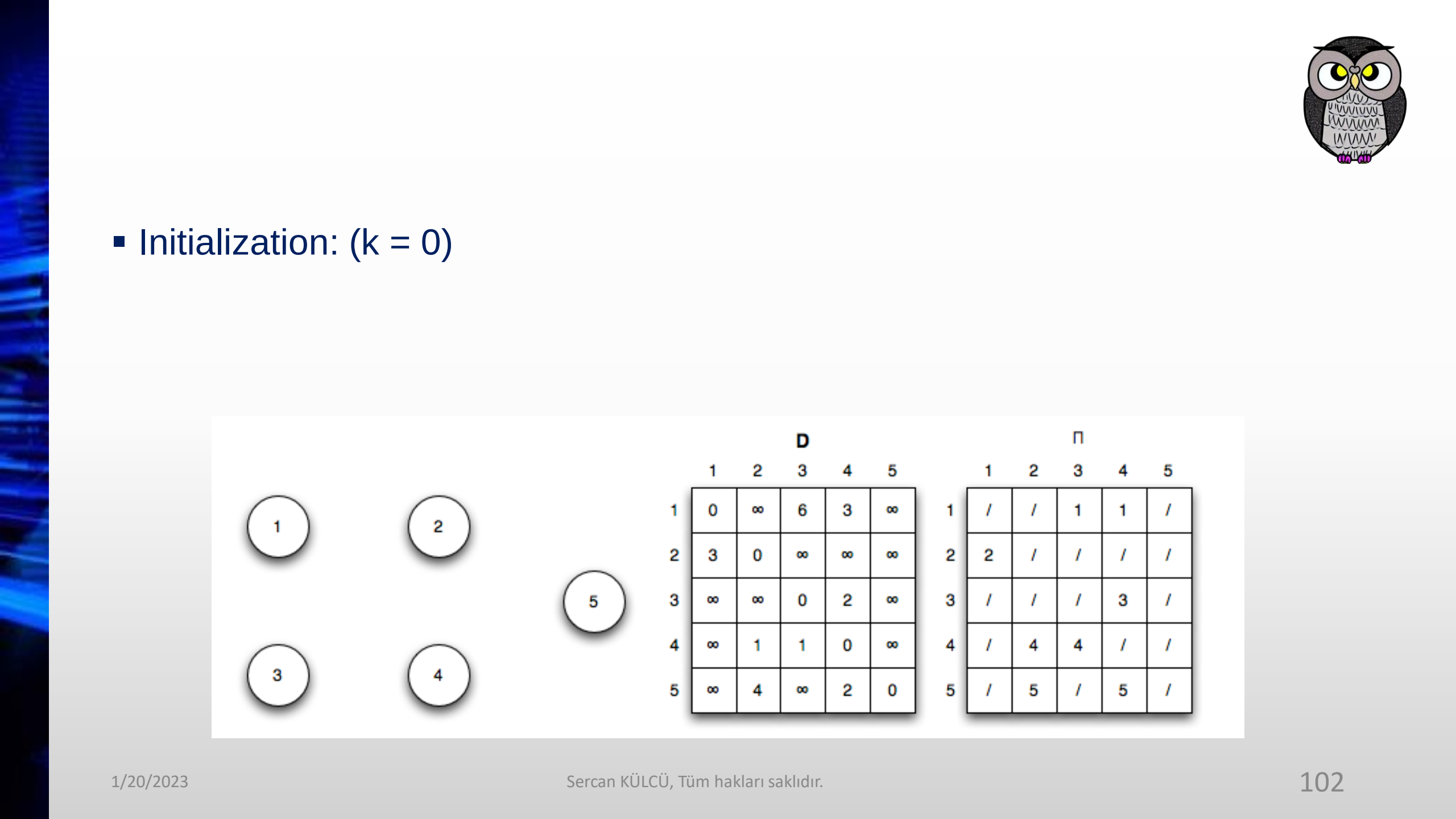
- Adım 1: Her bir çift düğüm arasındaki ağırlıklar, doğrudan kenarlarla belirtilir. Eğer iki düğüm arasında doğrudan bir kenar yoksa, uzaklık sonsuz kabul edilir.
- Adım 2: Her bir düğüm çifti için, tüm ara düğümler sırayla incelenir.
- Adım 3: Ara düğümler üzerinden geçerek, yeni yolun uzunluğu hesaplanır ve mevcut en kısa yol uzunluğu ile karşılaştırılır.
- Adım 4: Yeni bulunan en kısa yollar, matrise kaydedilir.

Karmaşıklık Analizi

- Floyd-Warshall Algoritması'nın karmaşıklığı $O(V^3)$ şeklindedir.
- V düğüm sayısını temsil eder.
- 3 adet iç içe for döngüsü kullanılır.
  - ilk döngü tüm ara düğümleri gezer
  - ikinci döngü tüm kaynak düğümleri gezer.
  - son döngü tüm hedef düğümleri gezer.

Örnek





- Initialization: (k = 0)

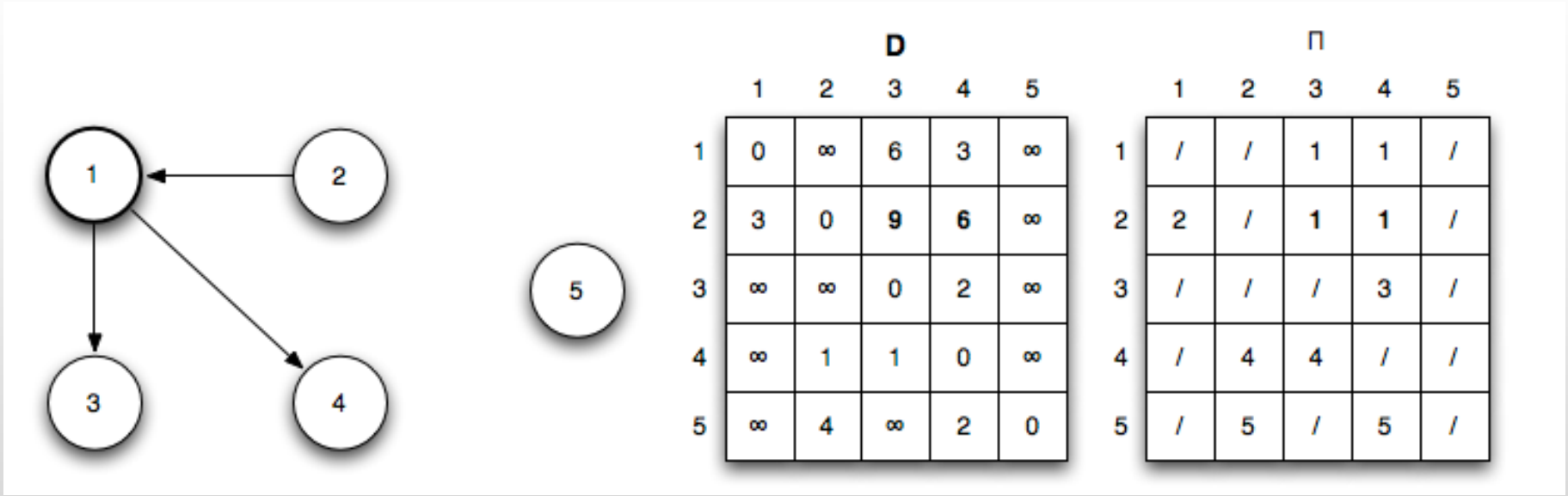
D

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

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	/	/	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/

- Iteration 1: (k = 1) Shorter paths from 2 ⇝ 3 and 2 ⇝ 4 are found through vertex 1



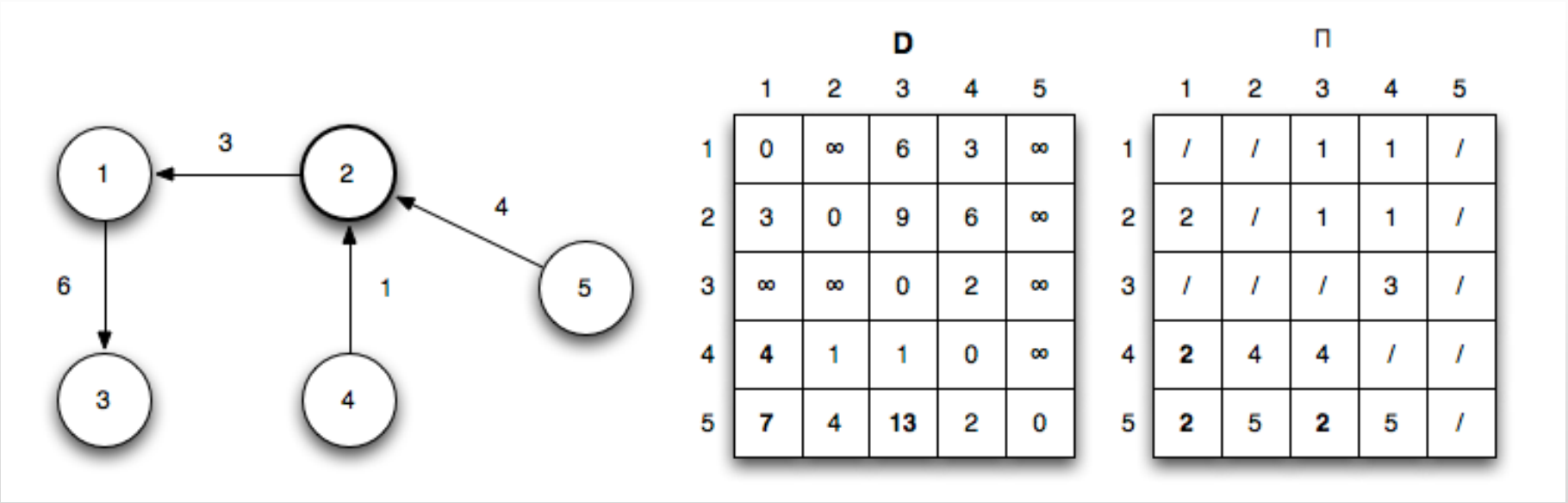
D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	**9**	**6**	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	**1**	**1**	/
3	/	/	/	3	/
4	/	4	4	/	/
5	/	5	/	5	/

- Iteration 2: (k = 2) Shorter paths from 4 ⇝ 1, 5 ⇝ 1, and 5 ⇝ 3 are found through vertex 2



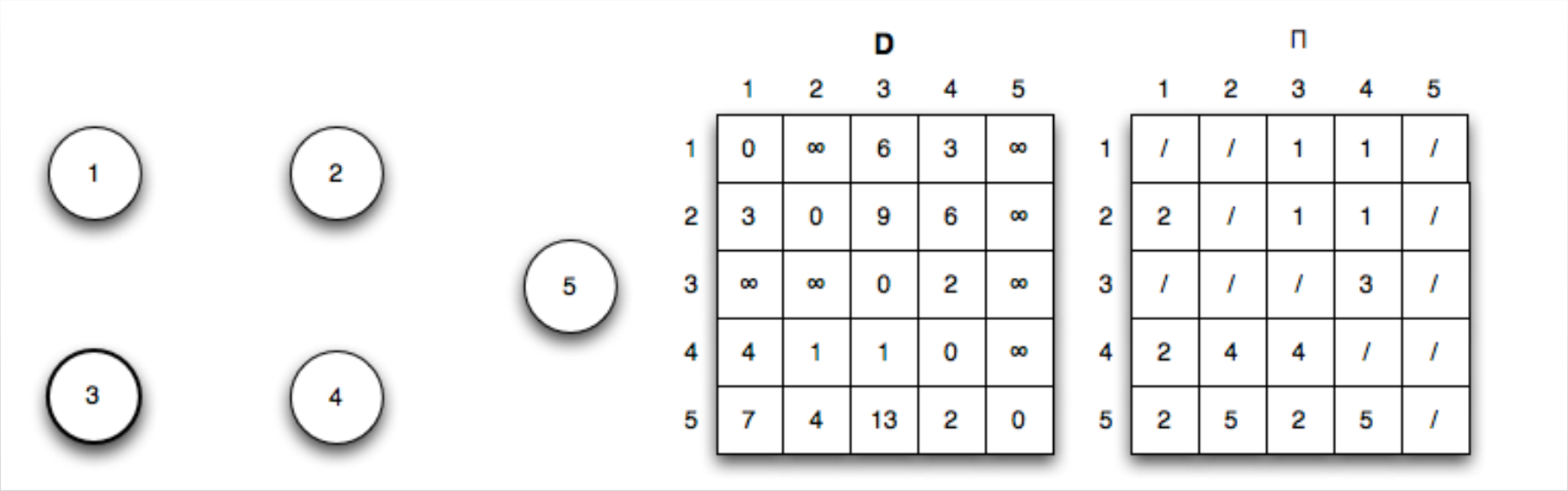
D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	**4**	1	1	0	∞
5	**7**	4	**13**	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	**2**	4	4	/	/
5	**2**	5	**2**	5	/

- Iteration 3: (k = 3) No shorter paths are found through vertex 3



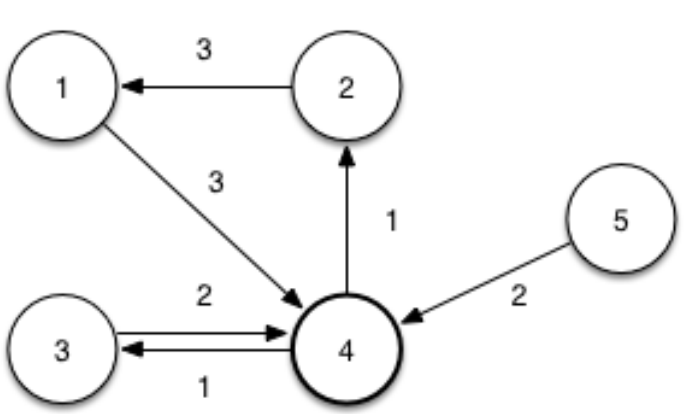
D

	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	4	1	1	0	∞
5	7	4	13	2	0

Π

	1	2	3	4	5
1	/	/	1	1	/
2	2	/	1	1	/
3	/	/	/	3	/
4	2	4	4	/	/
5	2	5	2	5	/

- Iteration 4: (k = 4) Shorter paths from 1 ⇝ 2, 1 ⇝ 3, 2 ⇝ 3, 3 ⇝ 1, 3 ⇝ 2, 5 ⇝ 1, 5 ⇝ 2, 5 ⇝ 3, and 5 ⇝ 4 are found through vertex 4



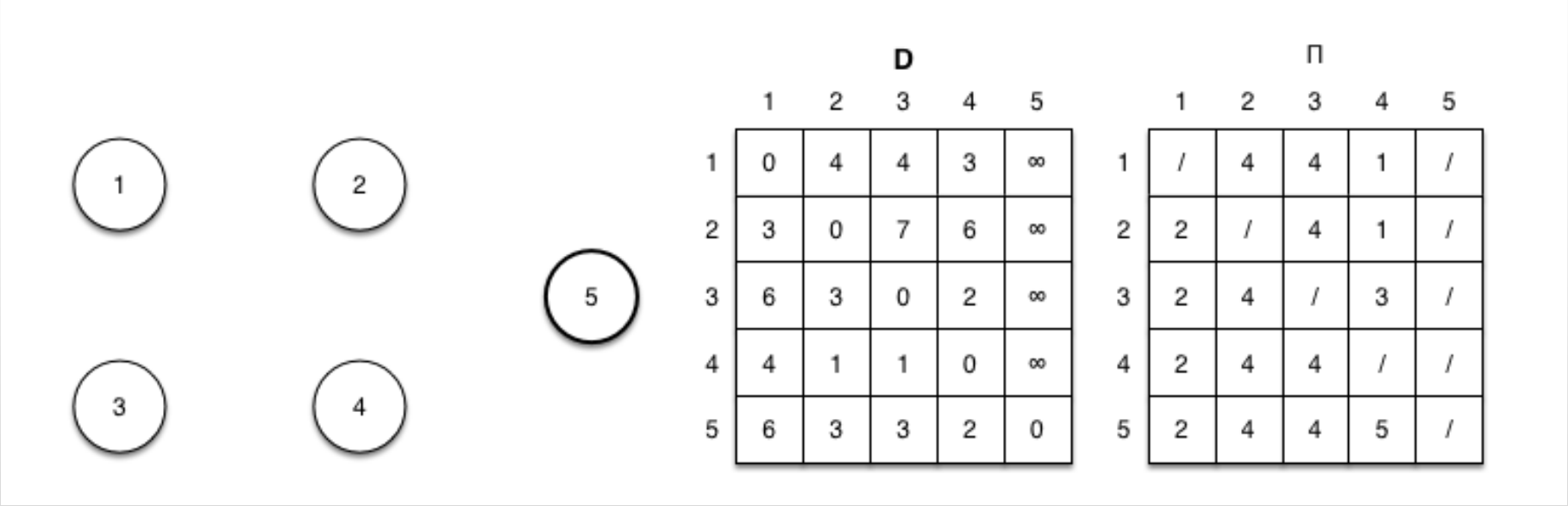
D

	1	2	3	4	5
1	0	**4**	**4**	3	∞
2	3	0	**7**	6	∞
3	**6**	**3**	0	2	∞
4	4	1	1	0	∞
5	**6**	**3**	**3**	2	0

Π

	1	2	3	4	5
1	/	**4**	**4**	1	/
2	2	/	**4**	1	/
3	**2**	**4**	/	3	/
4	2	4	4	/	/
5	2	**4**	**4**	5	/

- Iteration 5: (k = 5) No shorter paths are found through vertex 5



D

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

Π

	1	2	3	4	5
1	/	4	4	1	/
2	2	/	4	1	/
3	2	4	/	3	/
4	2	4	4	/	/
5	2	4	4	5	/

- The final shortest paths for all pairs is given by

D

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

Π

	1	2	3	4	5
1	/	4	4	1	/
2	2	/	4	1	/
3	2	4	/	3	/
4	2	4	4	/	/
5	2	4	4	5	/

A* (A Star) Algoritması

- İki nokta arasındaki en kısa yolu bulan bir bilgi arama algoritmasıdır.
- 1968'de Peter Hart, Nils Nilsson, Bertram Raphael tarafından geliştirildi.
- Genişlik öncelikli arama (Breadth-First Search) ile en iyi ilk arama (Best-First Search) algoritmalarının kombinasyonunu kullanır.
- Düzgün çalışması için doğru bir tahmin fonksiyonu gereklidir.
- Düğümlerin sayısı arttıkça karmaşıklığı artar.

Algoritma İlkeleri

- Her bir düğüm için tahmin (heuristic) değeri kullanır.
- Bu tahmin, düğümün hedefe olan tahmini mesafesini belirtir.
- Her adımda,
  - komşu düğümler arasından,
  - hedef ile arasındaki gerçek maliyet ve tahmini maliyet toplamı en küçük olan seçilir.
- Bu özellik sayesinde, algoritma hedefe doğru hareket ederken, aynı zamanda en az maliyetli yolu seçmeye çalışır.

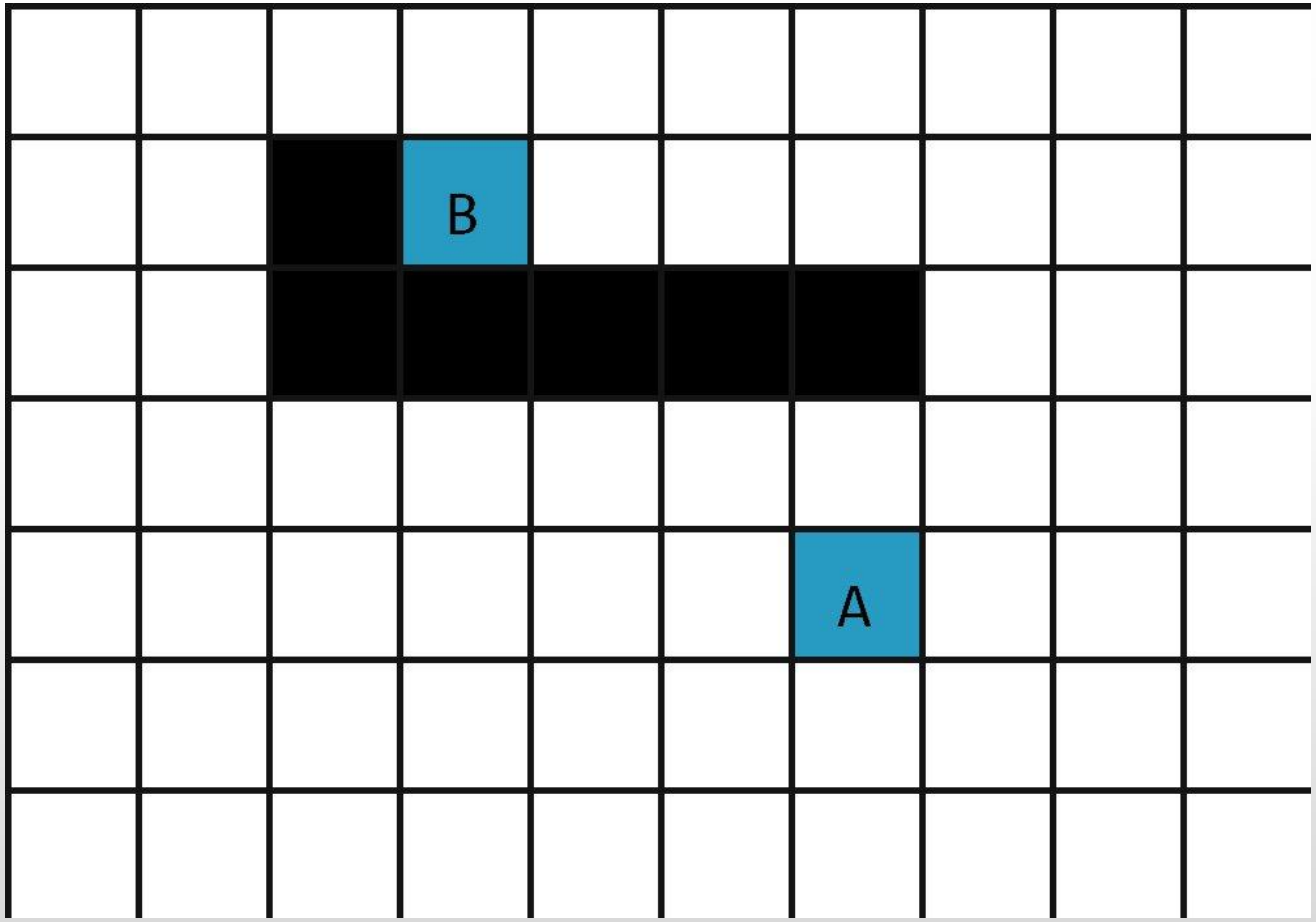
Algoritma Adımları

- Adım 1: Başlangıç düğümü seçilir ve bu düğüme uzaklık 0 atanır. Diğer düğümlere sonsuz uzaklık atanır.
- Adım 2: Mevcut düğümün komşuları incelenir ve her birinin tahmini maliyeti hesaplanır.
- Adım 3: Komşu düğümler arasından, gerçek maliyet ve tahmini maliyetin toplamı en küçük olan düğüm seçilir.
- Adım 4: Seçilen düğüm, şu ana kadar bulunan en uygun yolun bir parçası olarak kaydedilir.

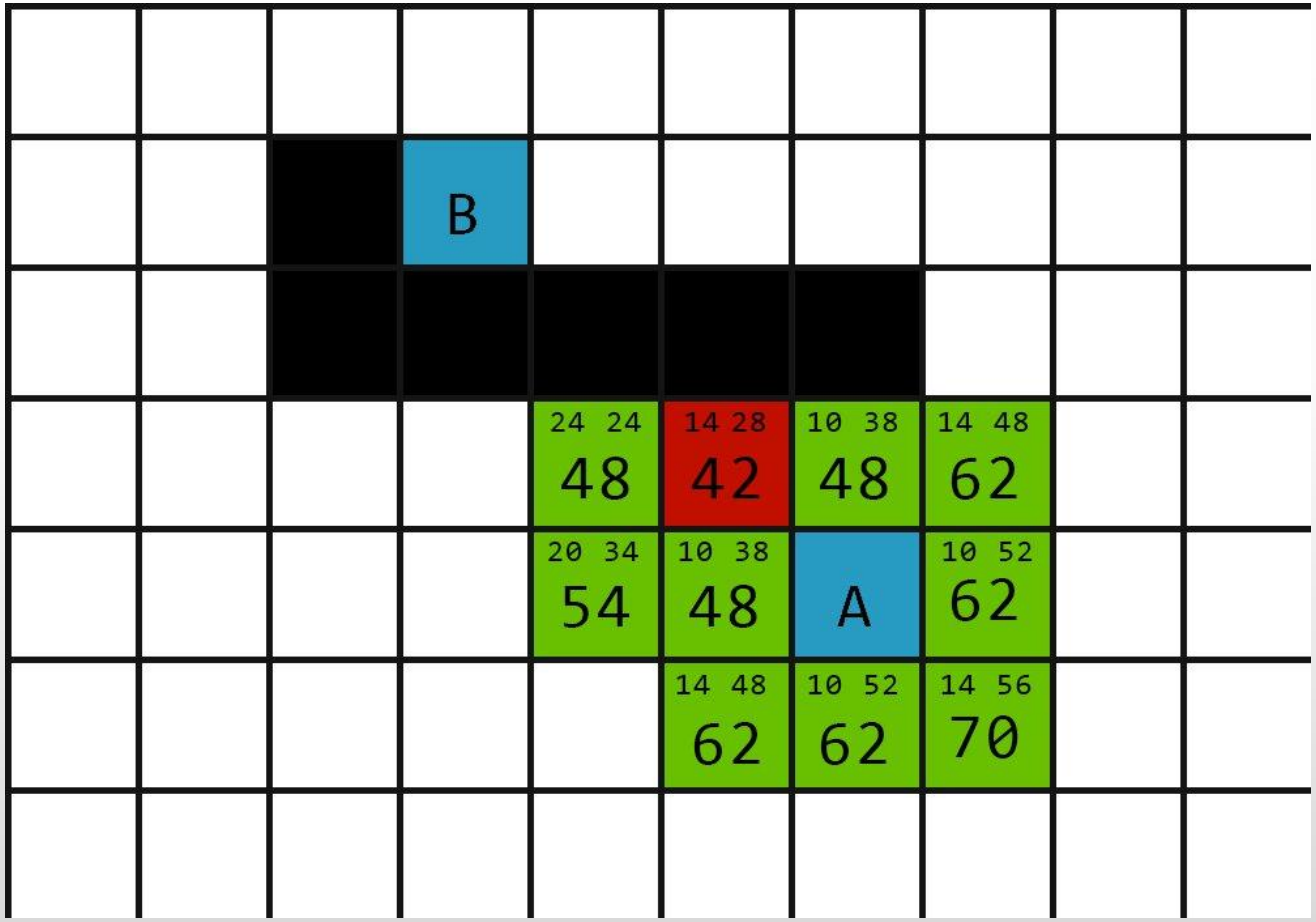
Algoritma Karmaşıklığı

- Eğer tahmin fonksiyonu gerçek maliyeti tam olarak tahmin ediyorsa,
  - karmaşıklık $O(b^d)$ şeklinde ifade edilir.
  - b çizgenin dallanma faktörünü,
  - d ise hedef düğüme olan maksimum derinliği temsil eder.

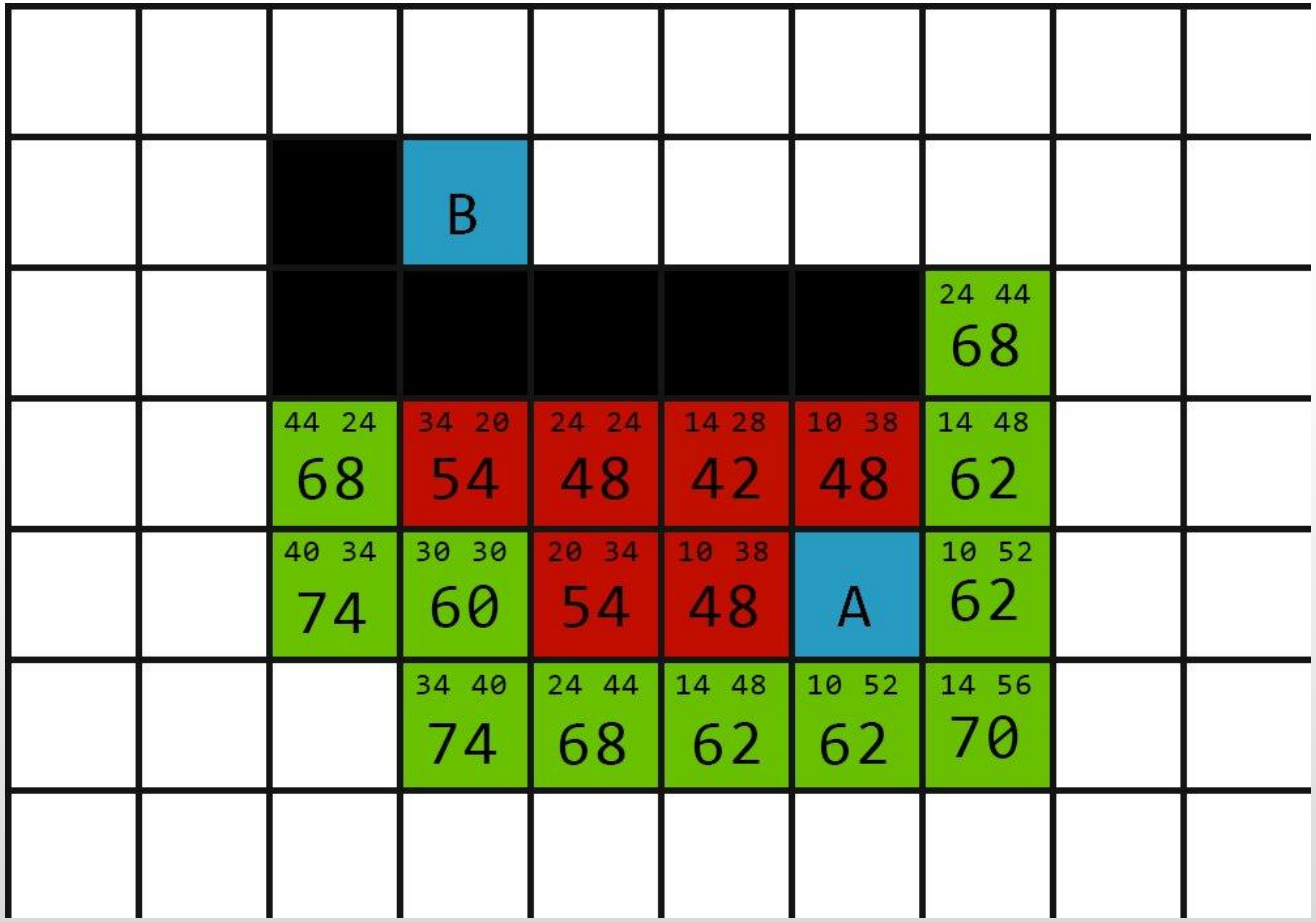
A Star



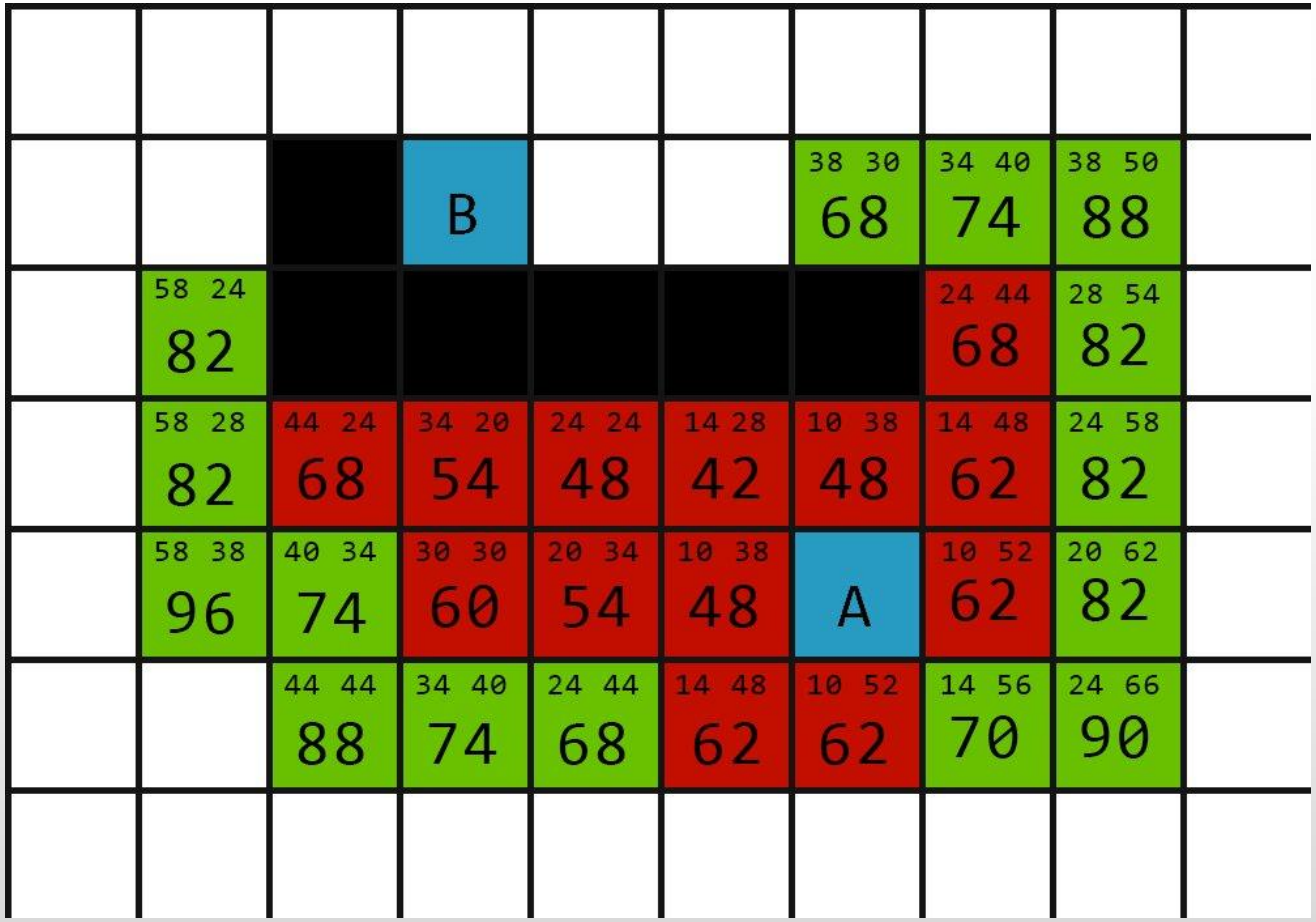
A Star



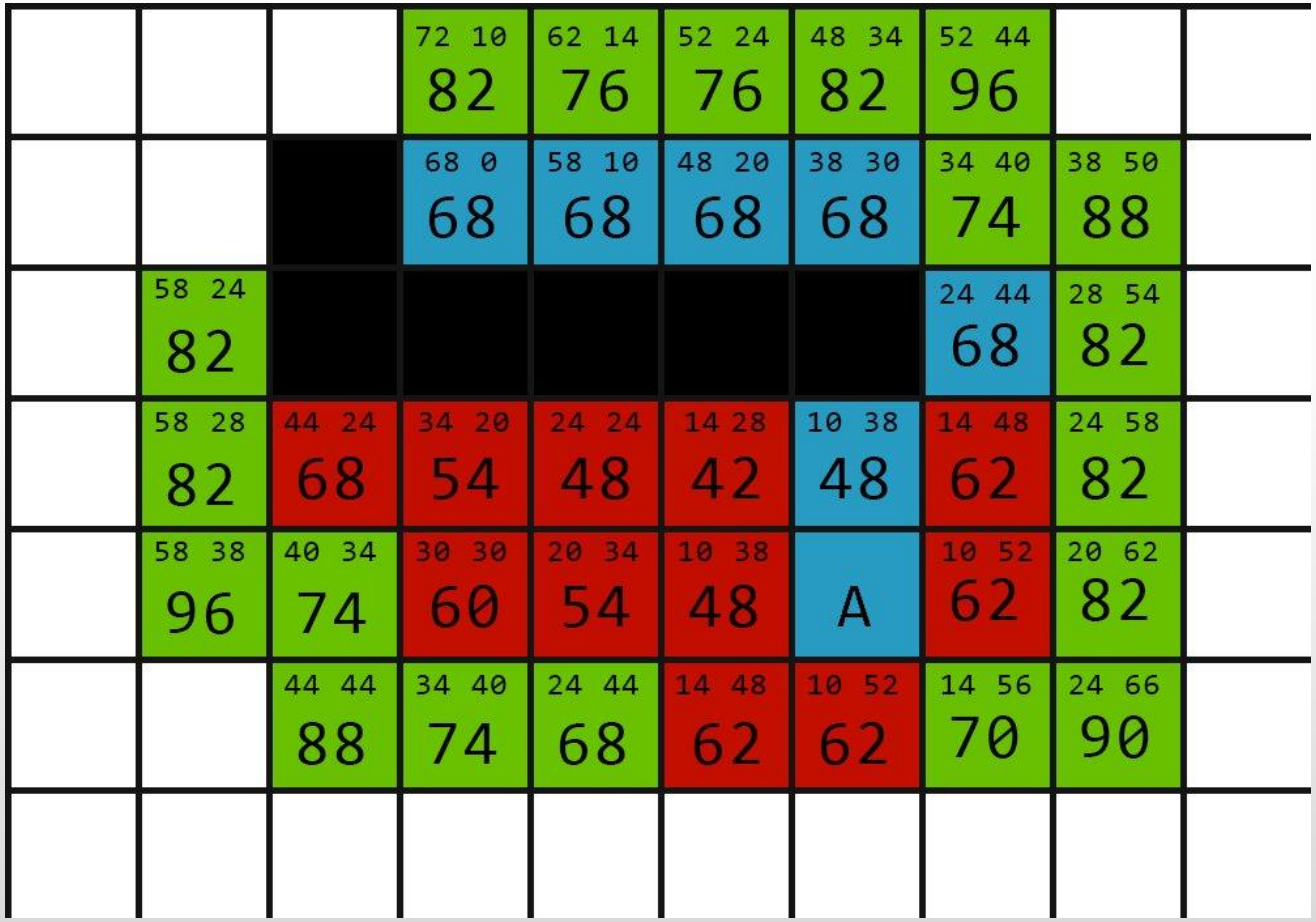
A Star



A Star



A Star



SON