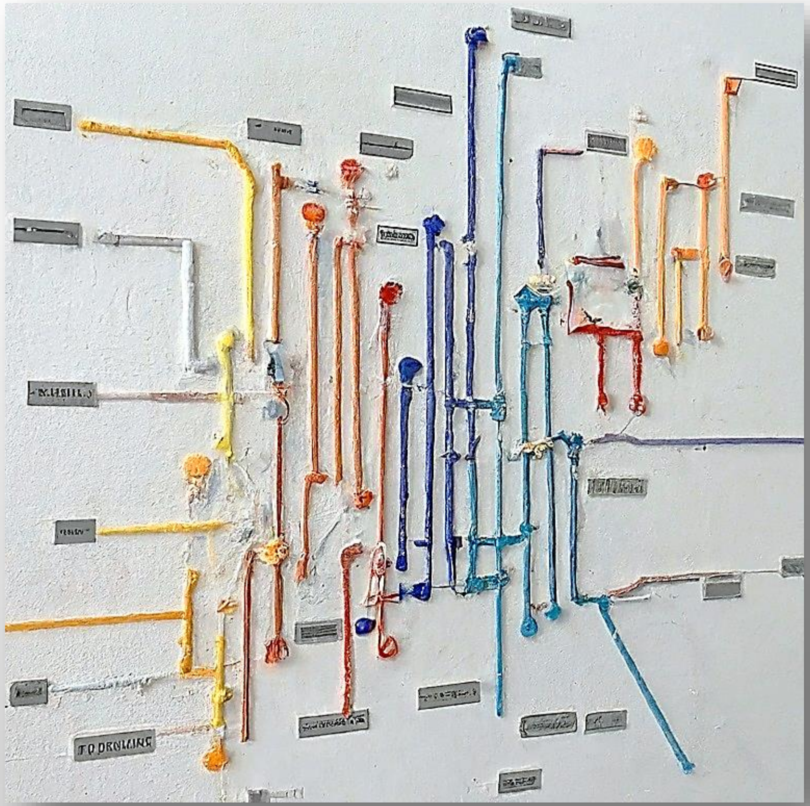
Bölüm 4: Çizge Algoritmaları

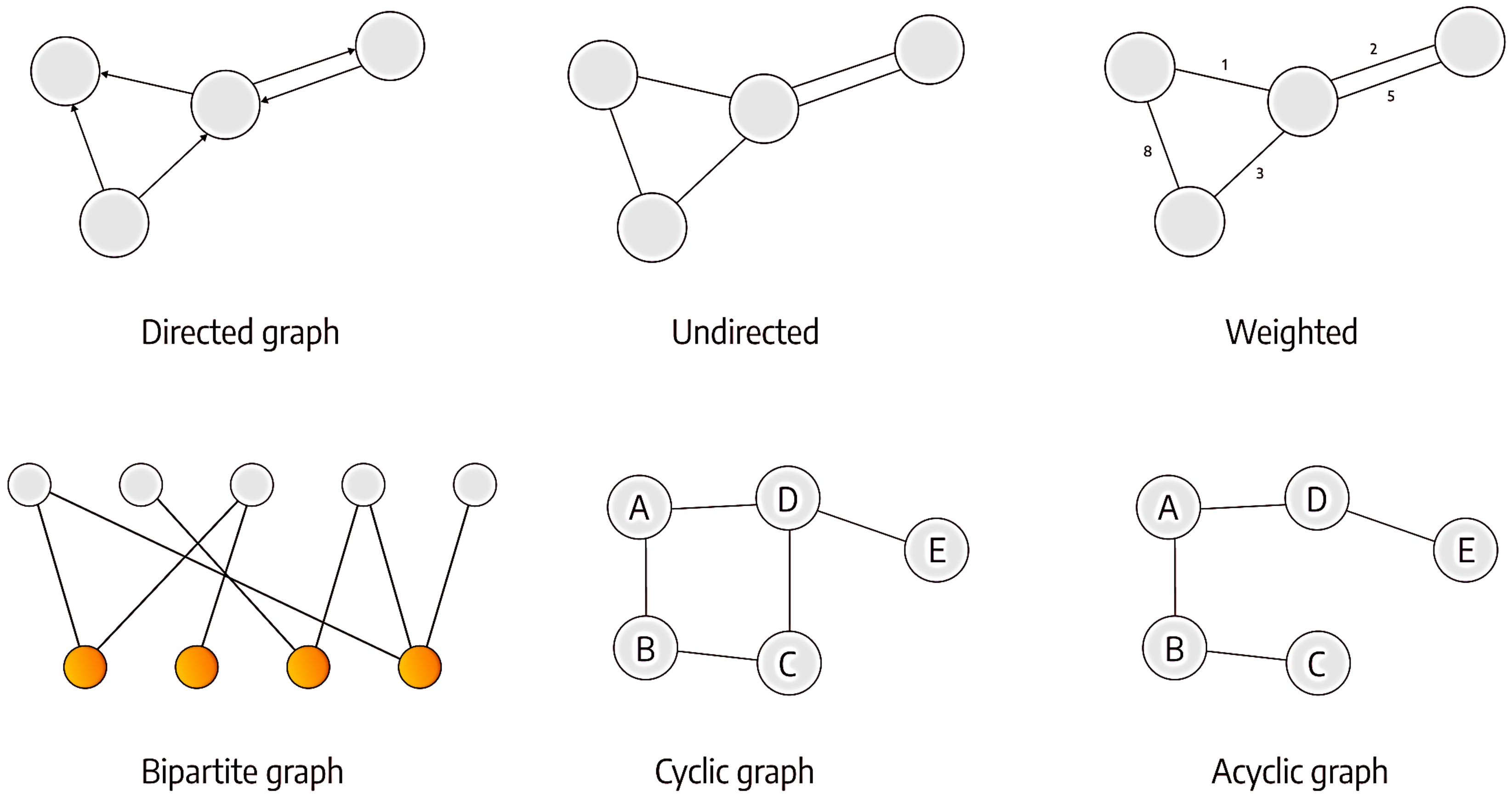
Algoritmalar

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.

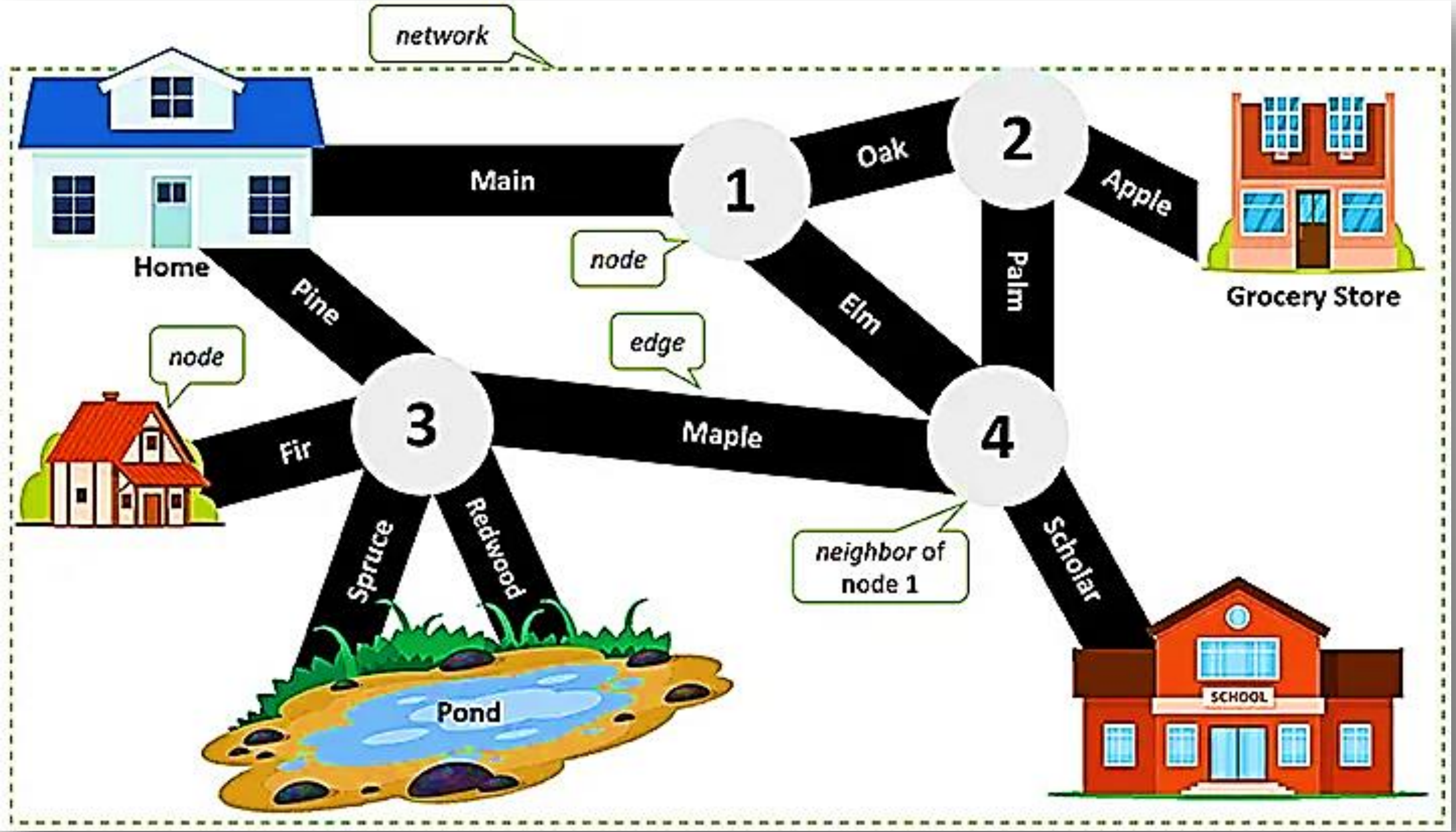
Çizge Algoritmaları

- 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



network
Home
Main
1
Oak
2
Apple
Grocery Store
node
Pine
Elm
Palm
edge
node
3
Fir
Maple
4
Spruce
Redwood
Scholar
neighbor of
node 1
Pond
SCHOOL

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.

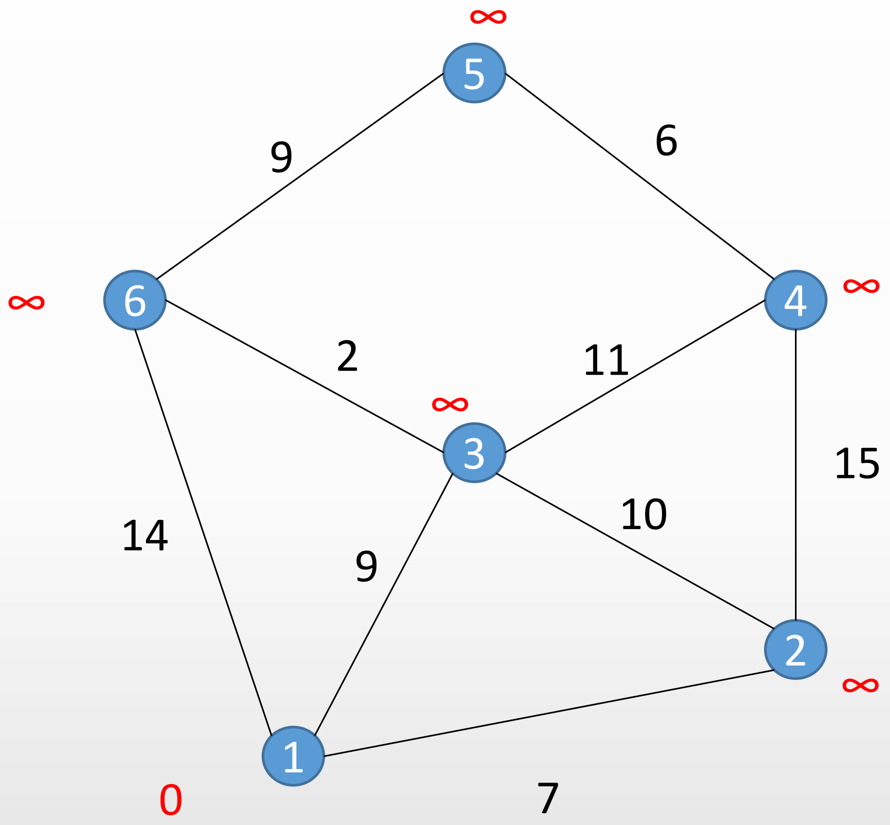
Algoritma Adımları

- 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

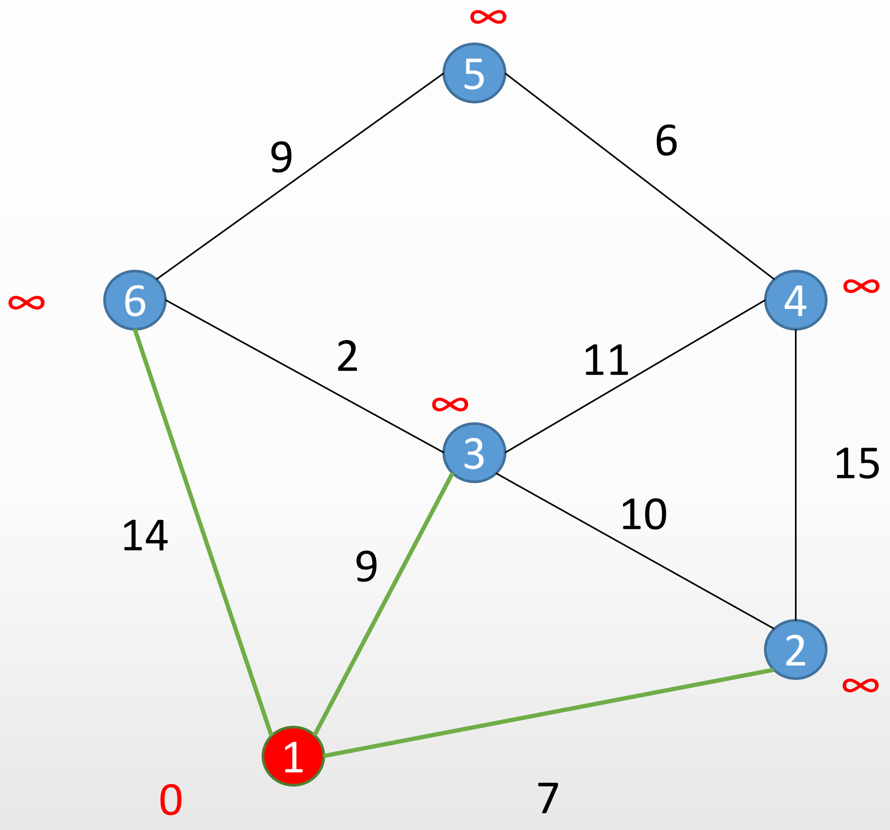
- En kısa mesafeli düğümü seçmek için;
  - Dizi temsili kullanılırsa;
    - $O(V^2)$ karmaşıklığına sahiptir.
  - Öncelik kuyruğu (*Priority Queue*) kullanılırsa;
    - karmaşıklık $O((V + E)\log V)$ olur.

Dijkstra



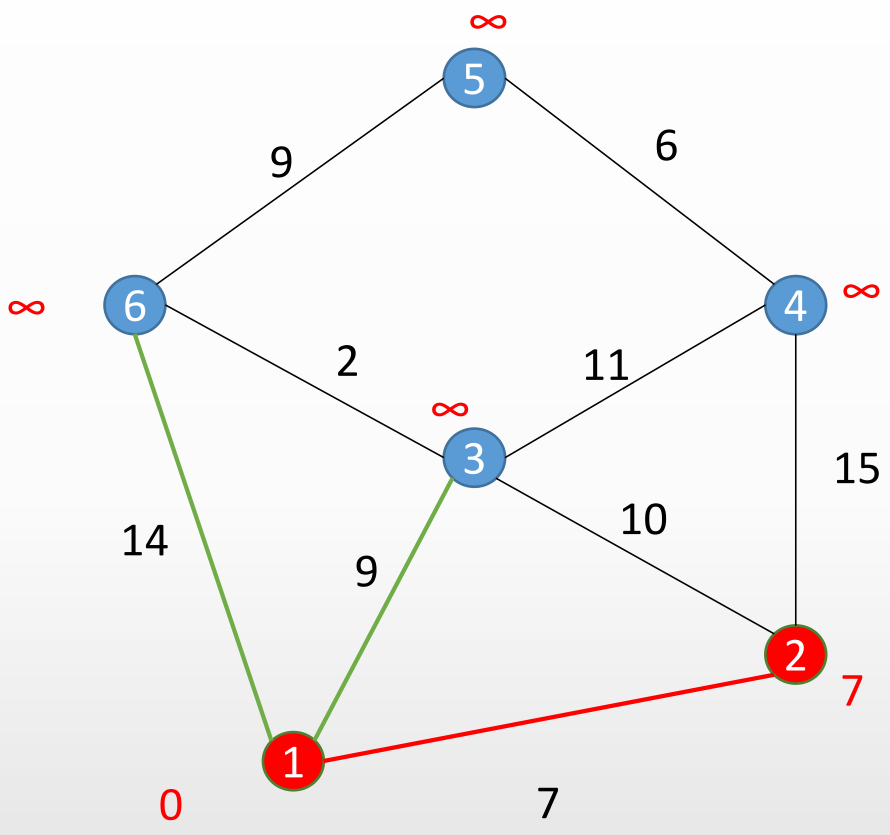
	1
1	0
2	∞
3	∞
4	∞
5	∞
6	∞

Dijkstra



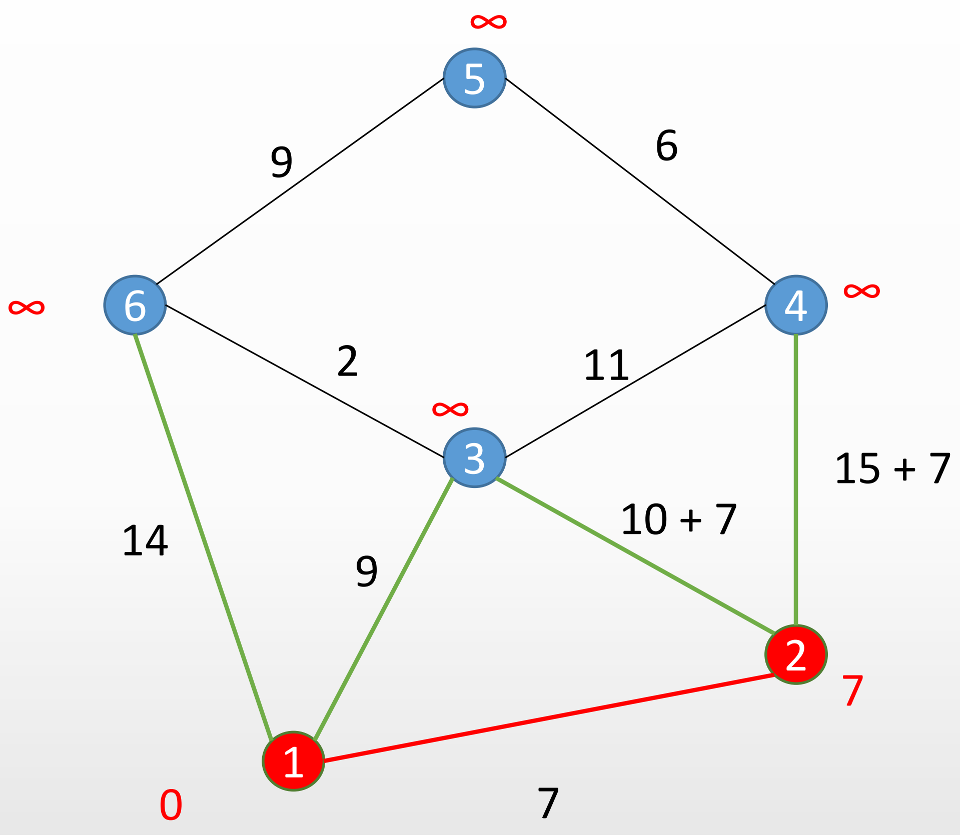
	1
1	0
2	∞
3	∞
4	∞
5	∞
6	∞

Dijkstra



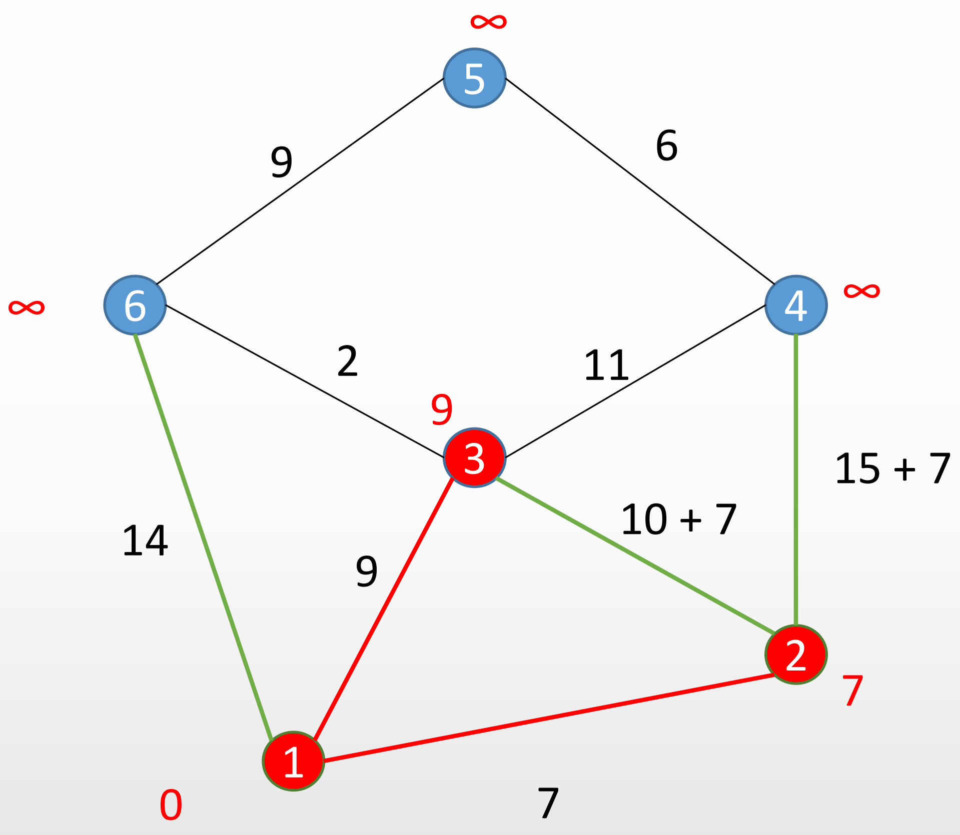
	1
1	0
2	7
3	∞
4	∞
5	∞
6	∞

Dijkstra



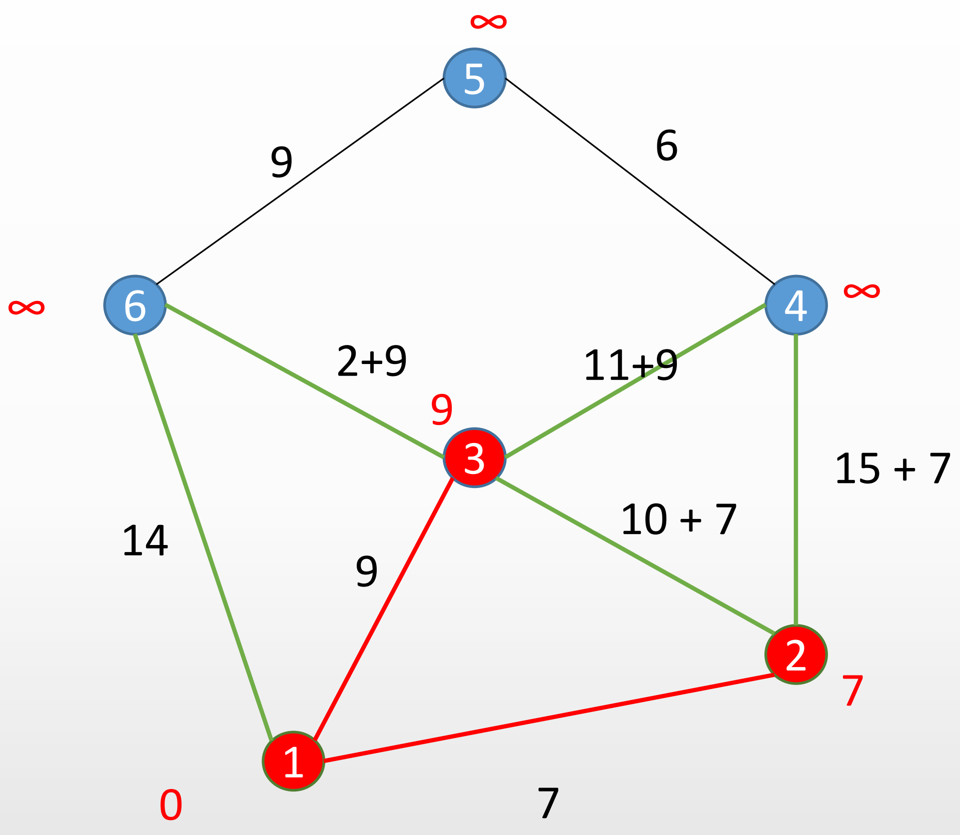
	1
1	0
2	7
3	∞
4	∞
5	∞
6	∞

Dijkstra



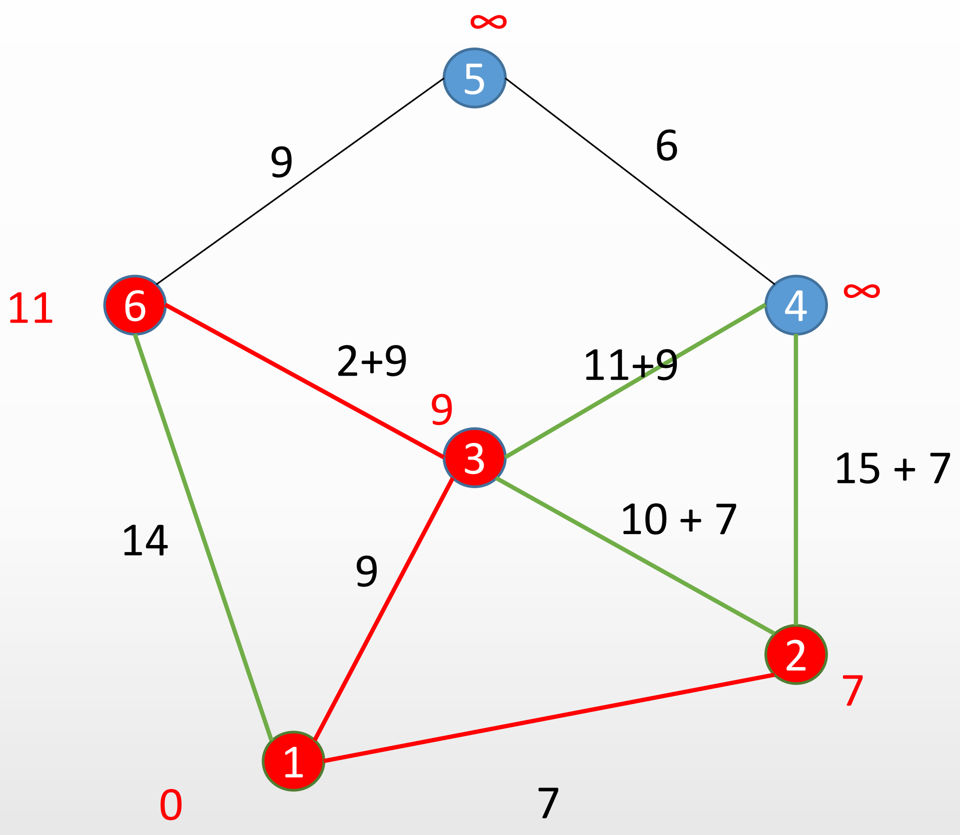
	1
1	0
2	7
3	9
4	∞
5	∞
6	∞

Dijkstra



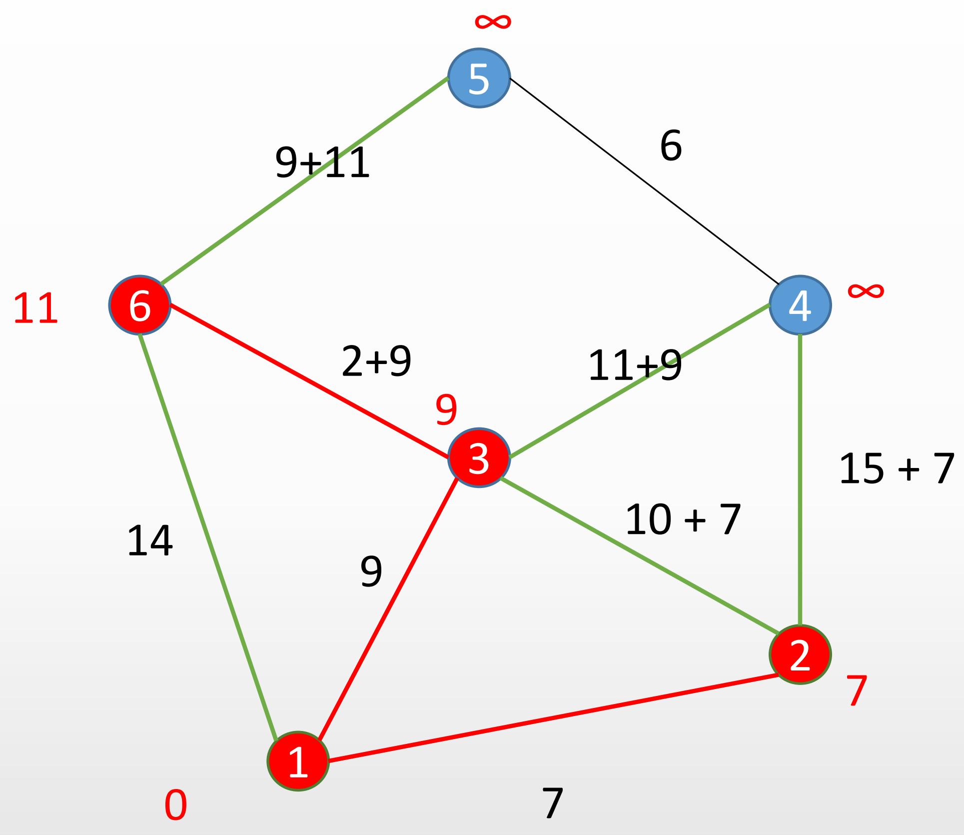
	1
1	0
2	7
3	9
4	∞
5	∞
6	∞

Dijkstra



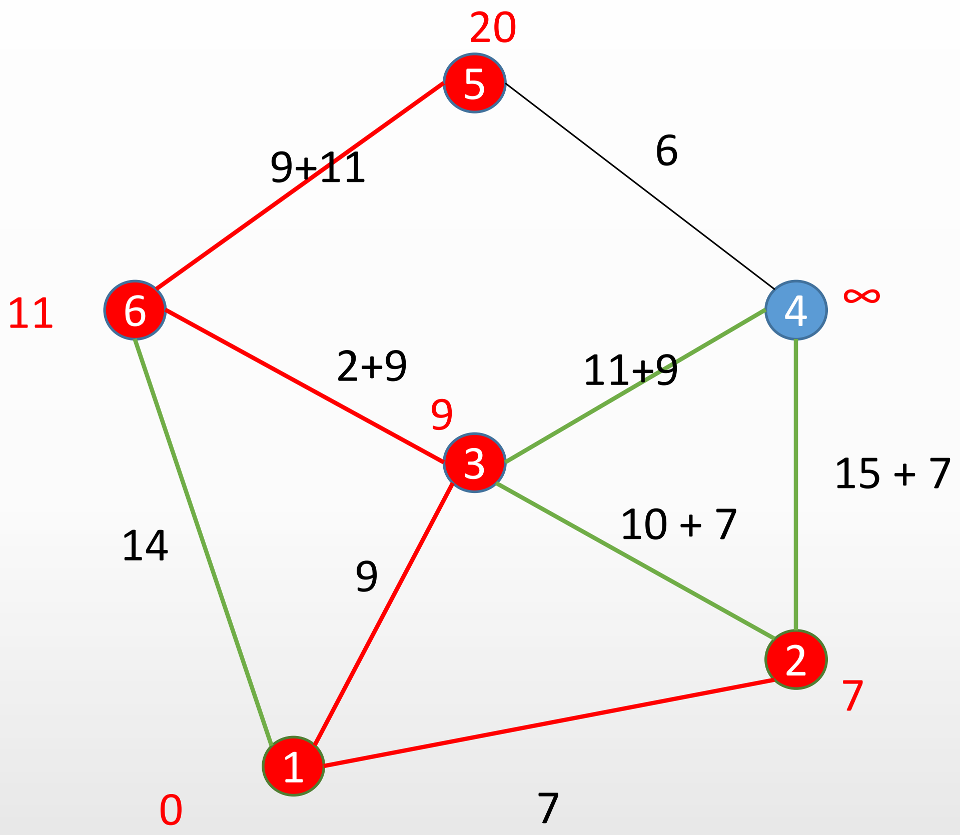
	1
1	0
2	7
3	9
4	∞
5	∞
6	11

Dijkstra



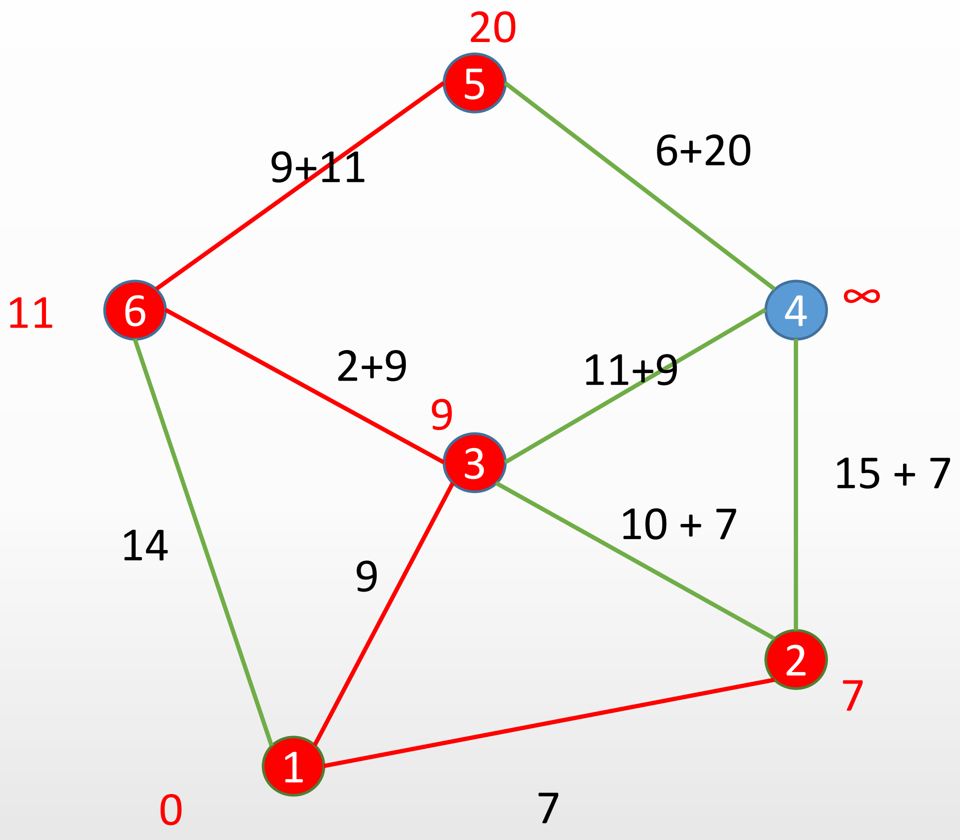
	1
1	0
2	7
3	9
4	∞
5	∞
6	11

Dijkstra



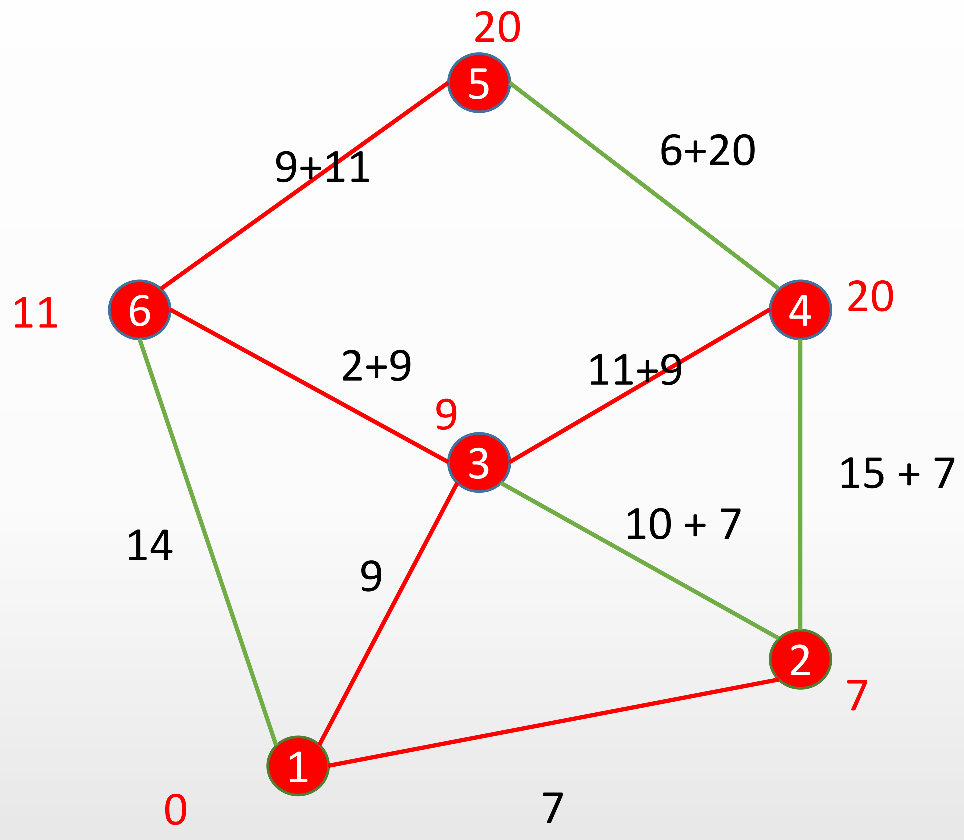
	1
1	0
2	7
3	9
4	∞
5	20
6	11

Dijkstra



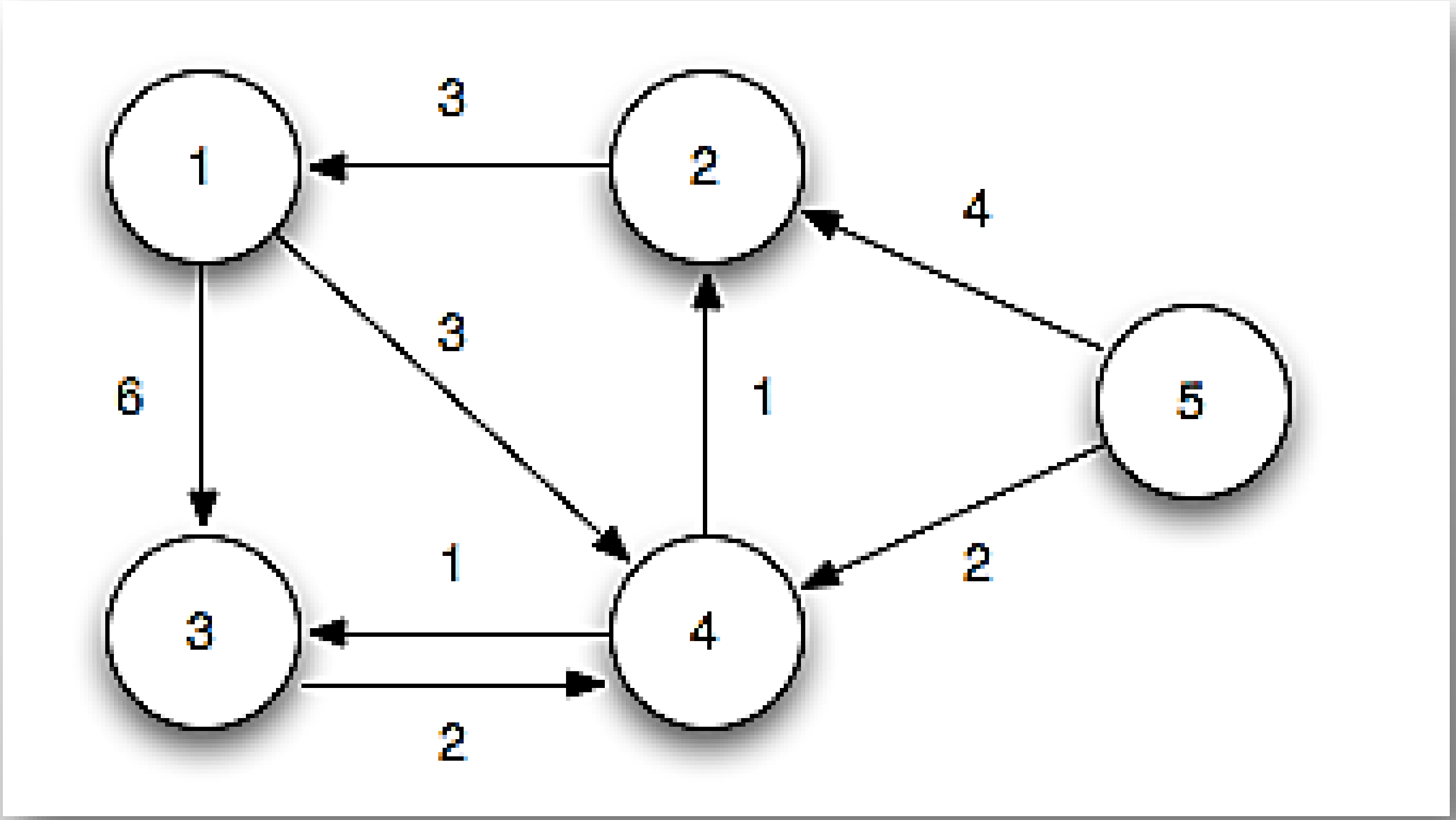
	1
1	0
2	7
3	9
4	∞
5	20
6	11

Dijkstra



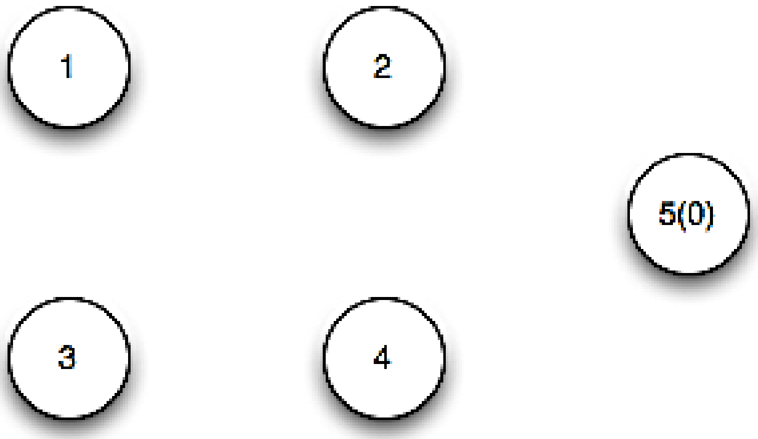
	1
1	0
2	7
3	9
4	20
5	20
6	11

Örnek



İlklendirme Aşaması

- Kaynak düğüm 5'in uzaklık değerine 0, diğerlerine ∞ atanır. S = ∅ olarak başlatılır.



Q

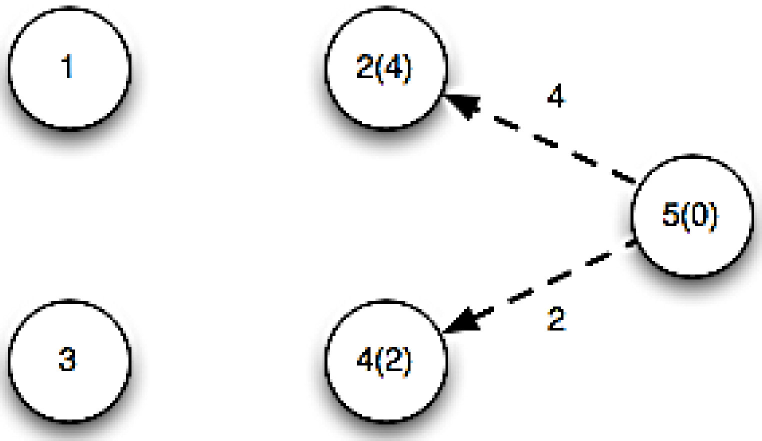
d	u	π
0	5	/
∞	1	/
∞	2	/
∞	3	/
∞	4	/

S

u	d	π

Iteration 1

- Düğüm 5 kuyruktan alınır, 0 uzaklık ile S kümesine konur. (u5,u2) ve (u5,u4) kenarları incelenerek kısa yollar hesaplanır. (relax)



Q

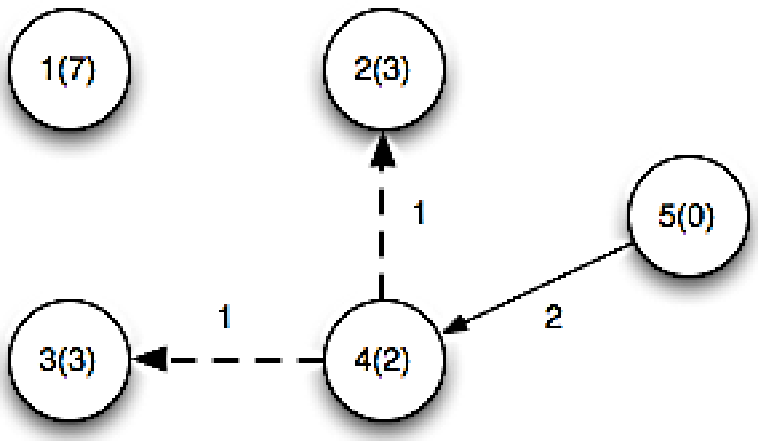
d	u	π
2	4	5
4	2	5
∞	1	/
∞	3	/

S

u	d	π
5	0	/

Iteration 2

- Düğüm 4 kuyruktan alınır, ve 2 uzaklık ile S kümesine konur. (u4,u2) ve (u4,u3) kenarları incelenerek kısa yollar hesaplanır. (relax) (u4,u2) kenarı daha kısa bir yol bulur.



Q

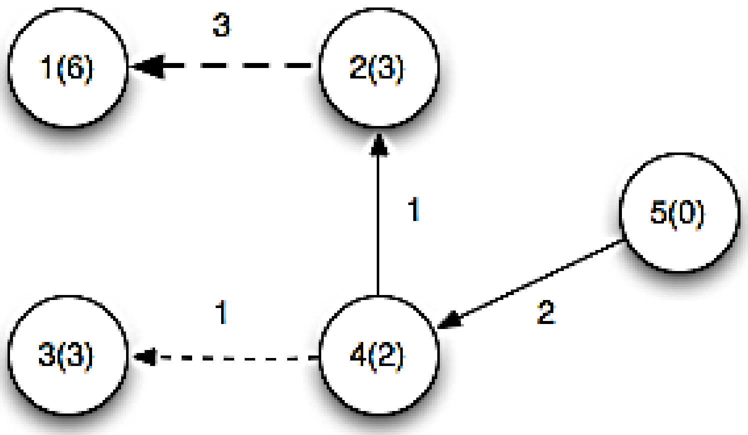
d	u	π
3	2	4
3	3	4
∞	1	/

S

u	d	π
5	0	/
4	2	5

Iteration 3

- Düğüm 2 kuyruktan alınır, 3 uzaklık ile S kümesine konur. (u2,u1) kenarı incelenerek en kısa yollar hesaplanır. (relax)



Q

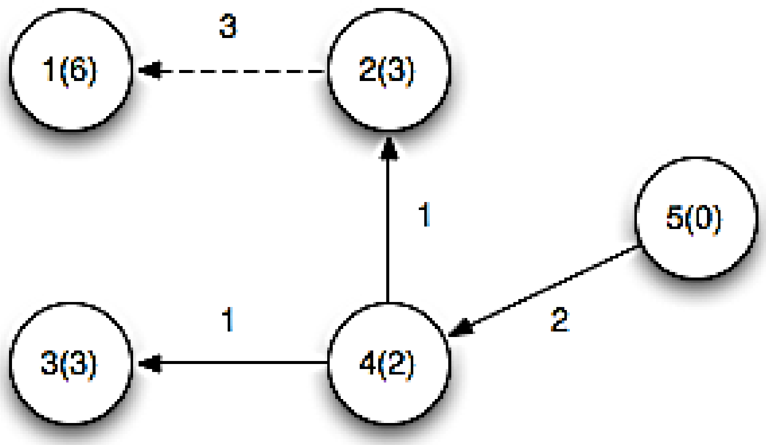
d	u	π
3	3	4
6	1	2

S

u	d	π
5	0	/
4	2	5
2	3	4

Iteration 4

- Düğüm 3 kuyruktan alınır, 3 uzaklık ile S kümesine konur. İncelenecek kenar yok. (no relax)



Q

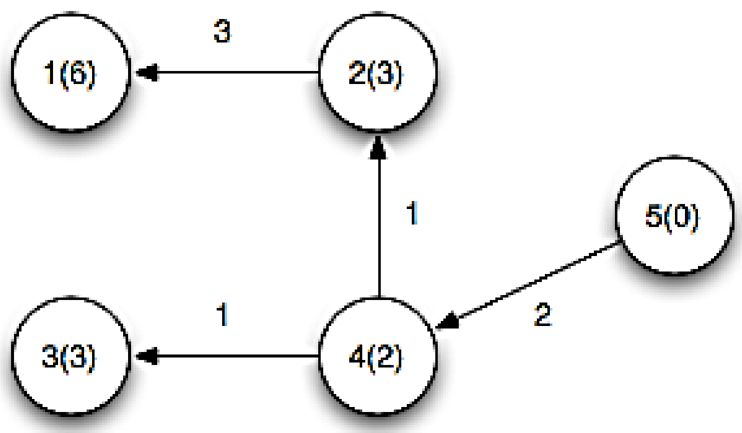
d	u	π
6	1	2

S

u	d	π
5	0	/
4	2	5
2	3	4
3	3	4

Iteration 5

- Düğüm 1 kuyruktan alınır, 6 uzaklık ile S kümesine konur. İncelenecek kenar yok. (no relax)



Q

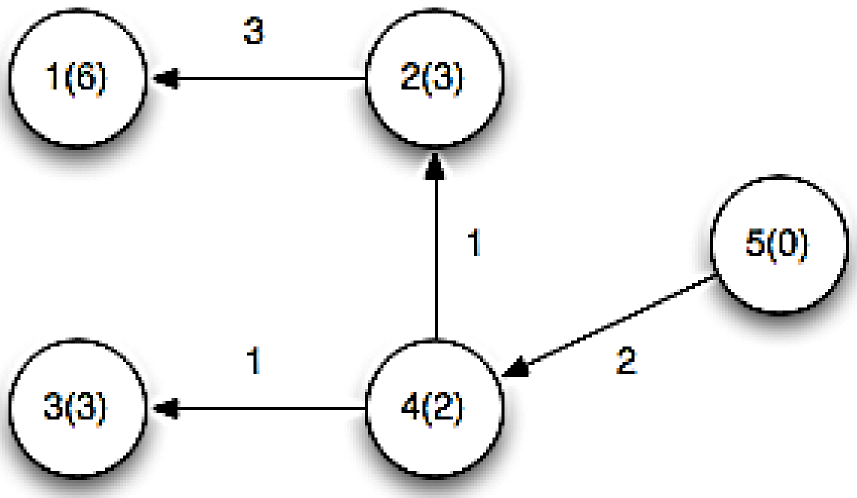
d	u	π

S

u	d	π
5	0	/
4	2	5
2	3	4
3	3	4
1	6	2

Son Durum

- Düğüm 5'ten diğer düğümlere olan en kısa yollar



	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.

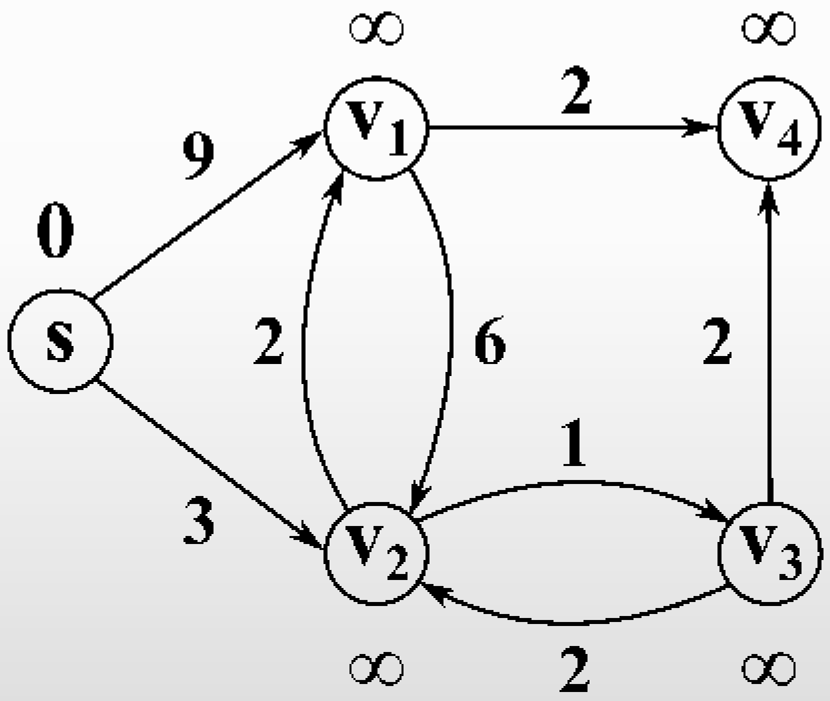
Algoritma Adımları

- 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.

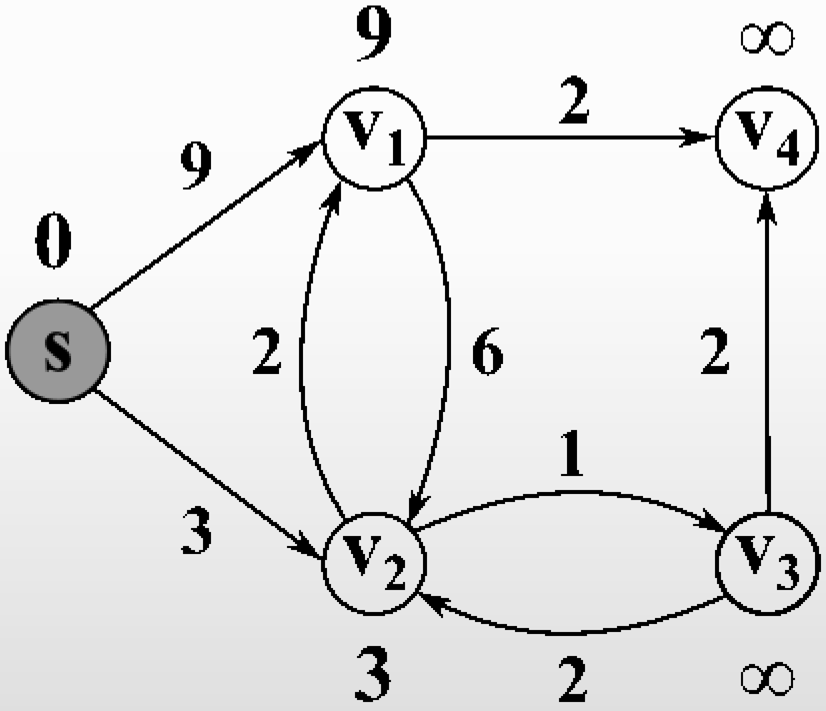
Karmaşıklık Analizi

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

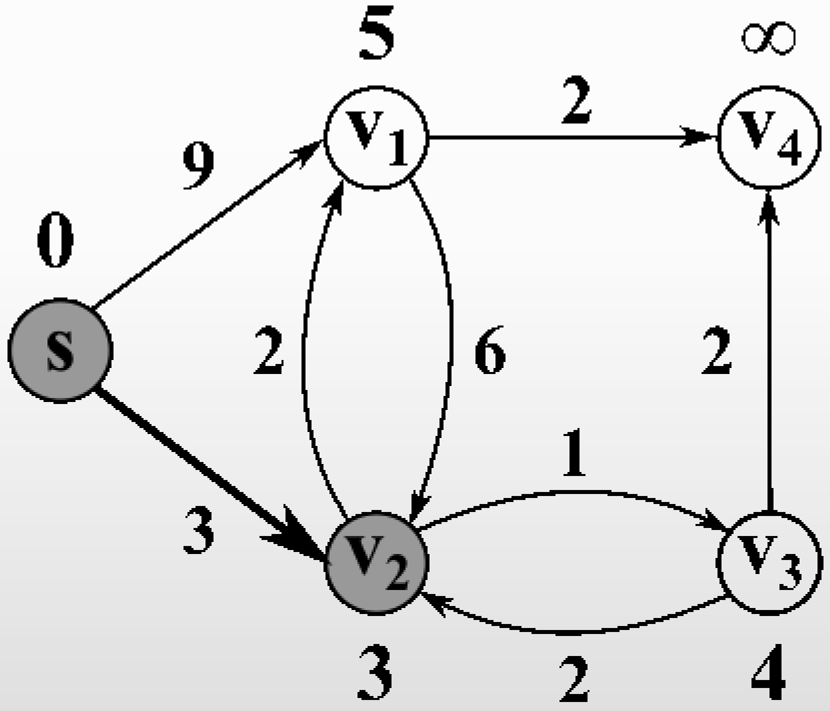
Bellman Ford



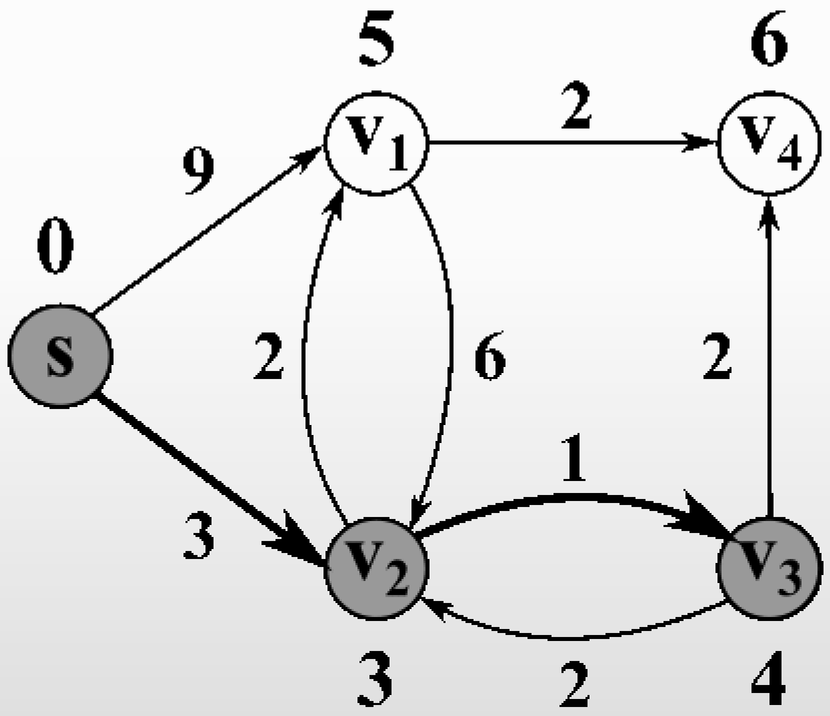
Bellman Ford



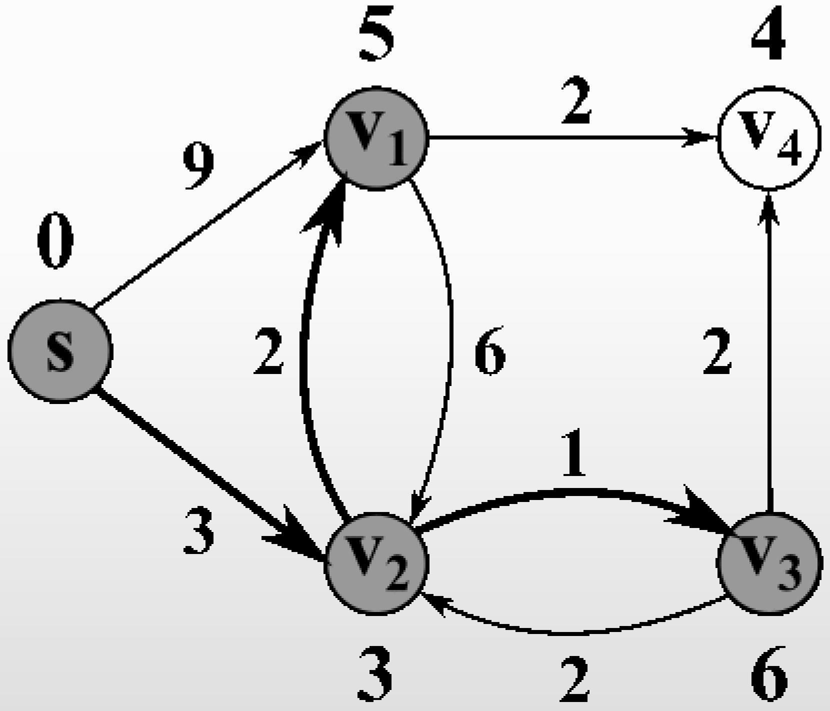
Bellman Ford



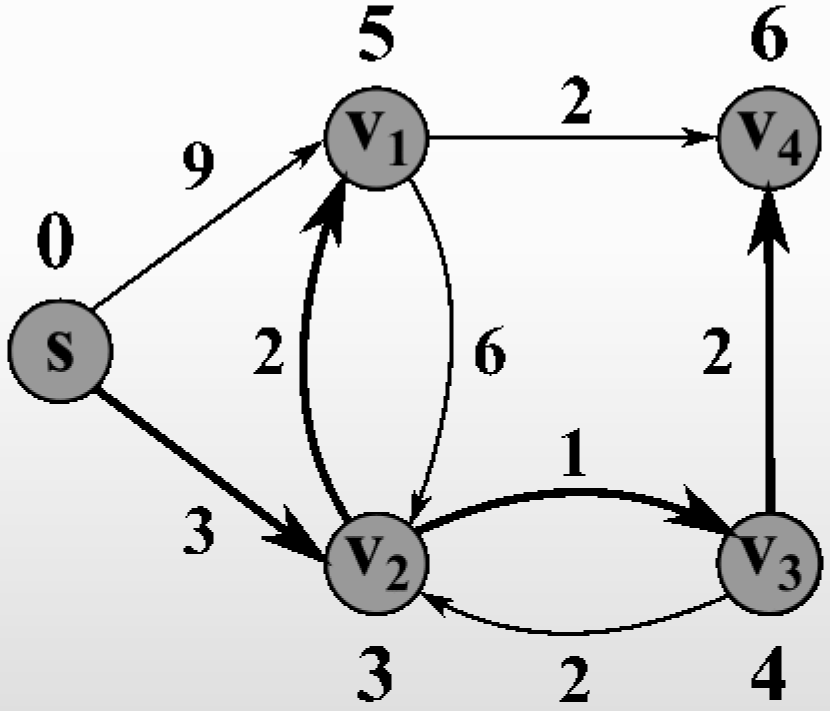
Bellman Ford



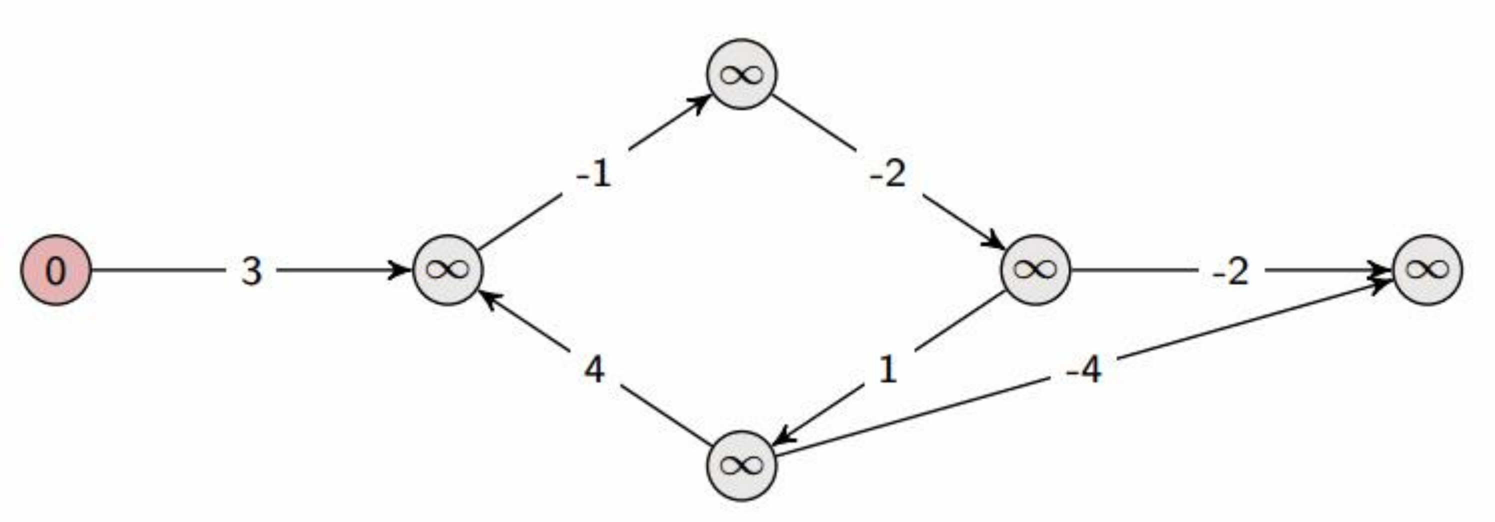
Bellman Ford



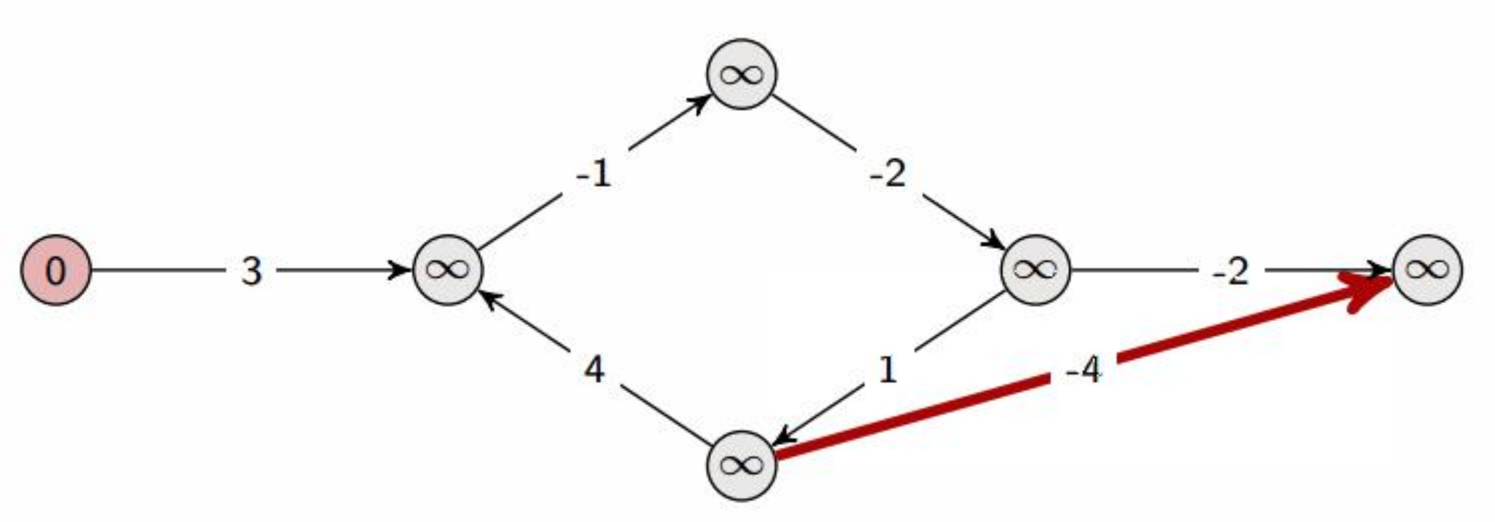
Bellman Ford



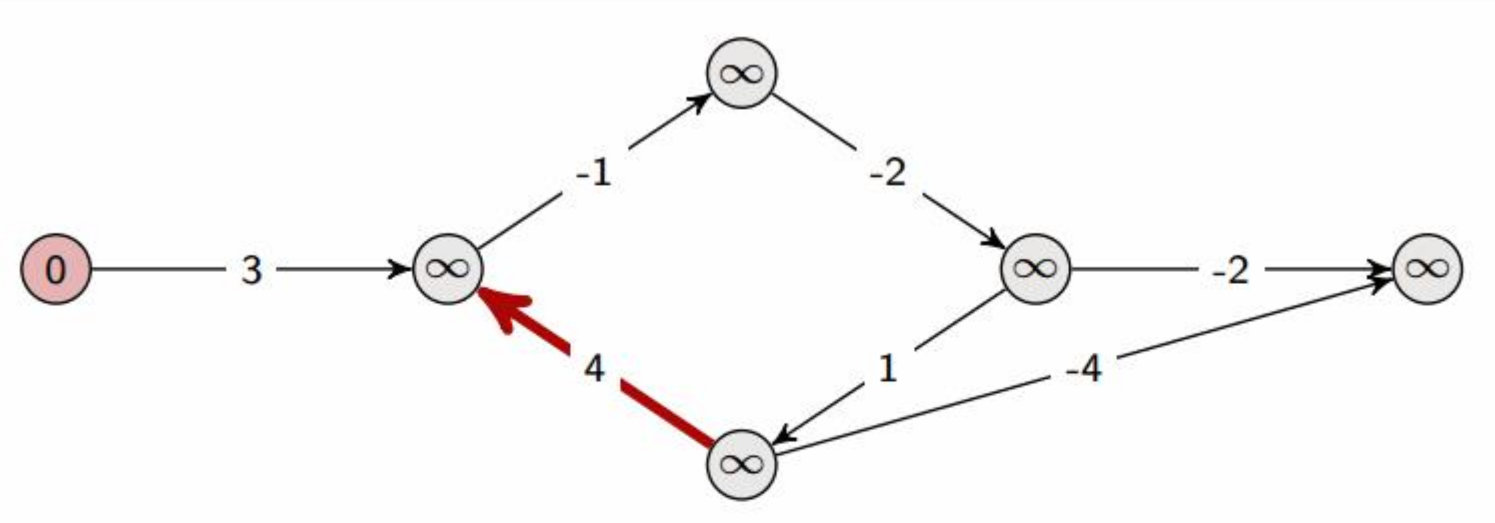
Bellman Ford



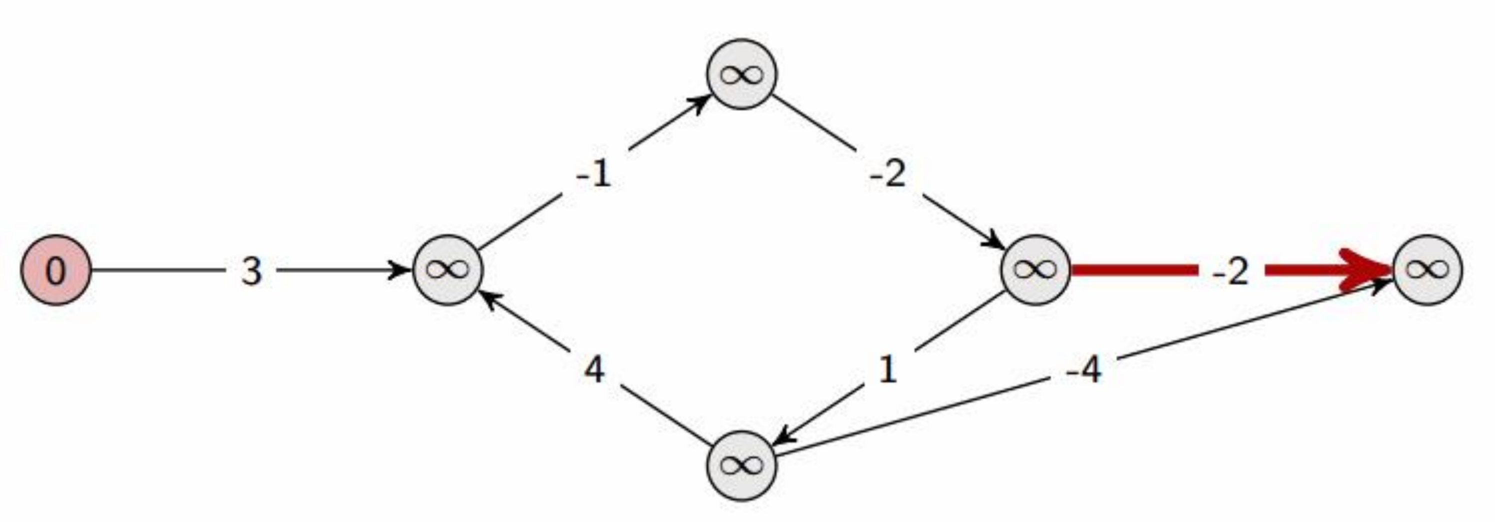
Bellman Ford



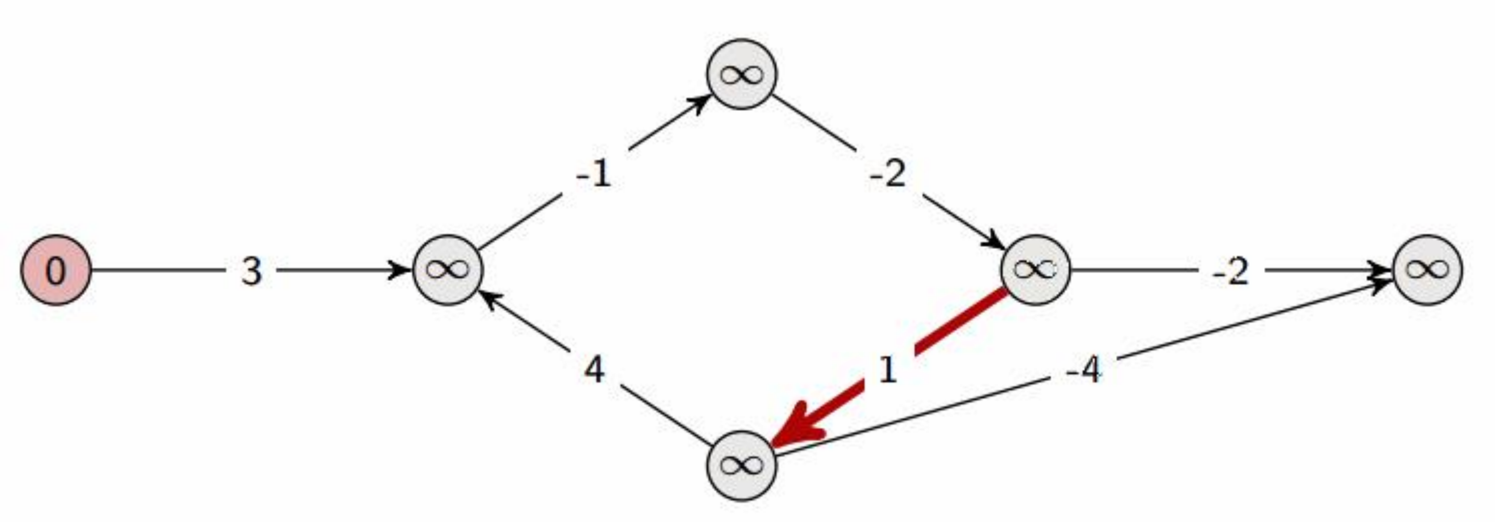
Bellman Ford



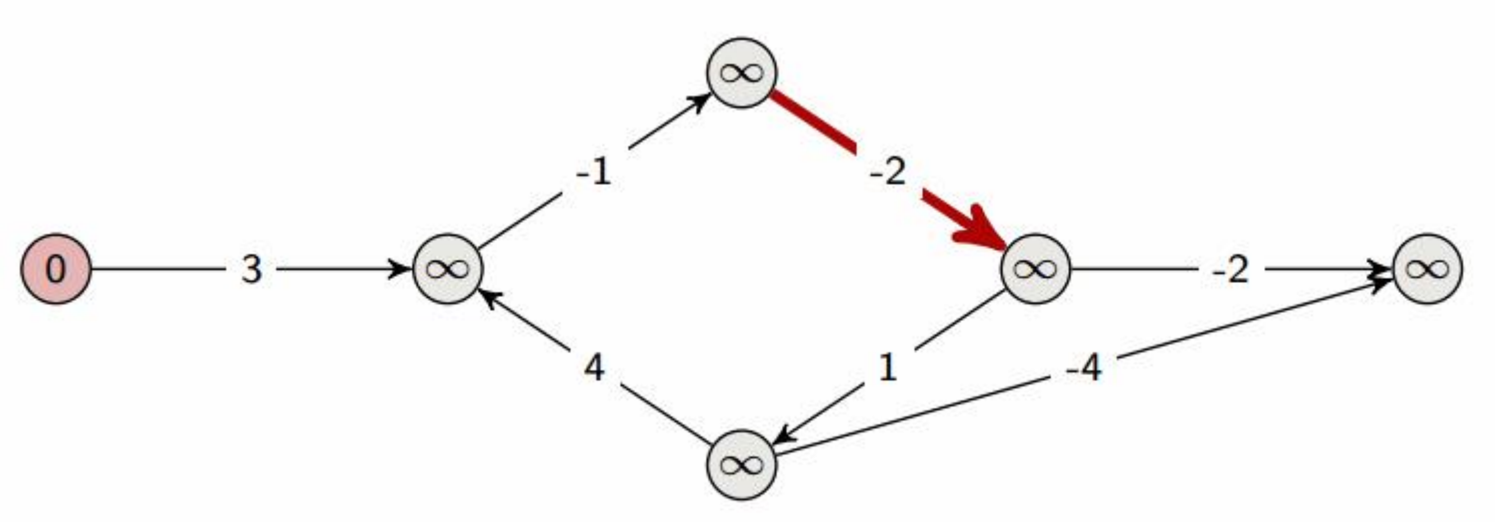
Bellman Ford



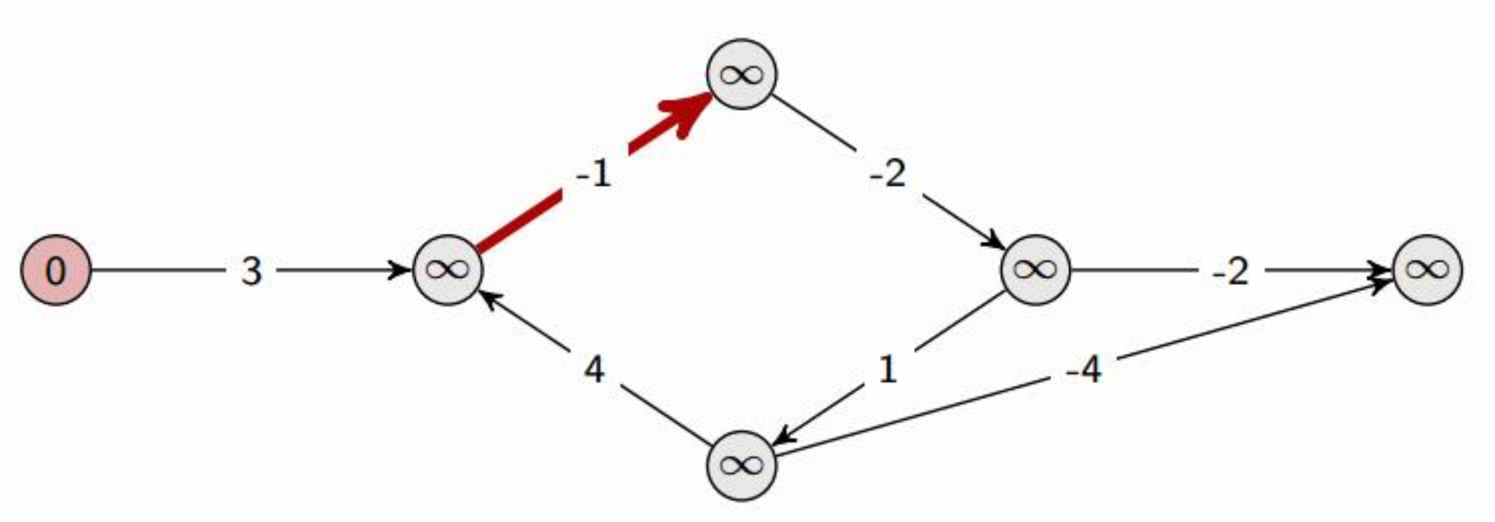
Bellman Ford



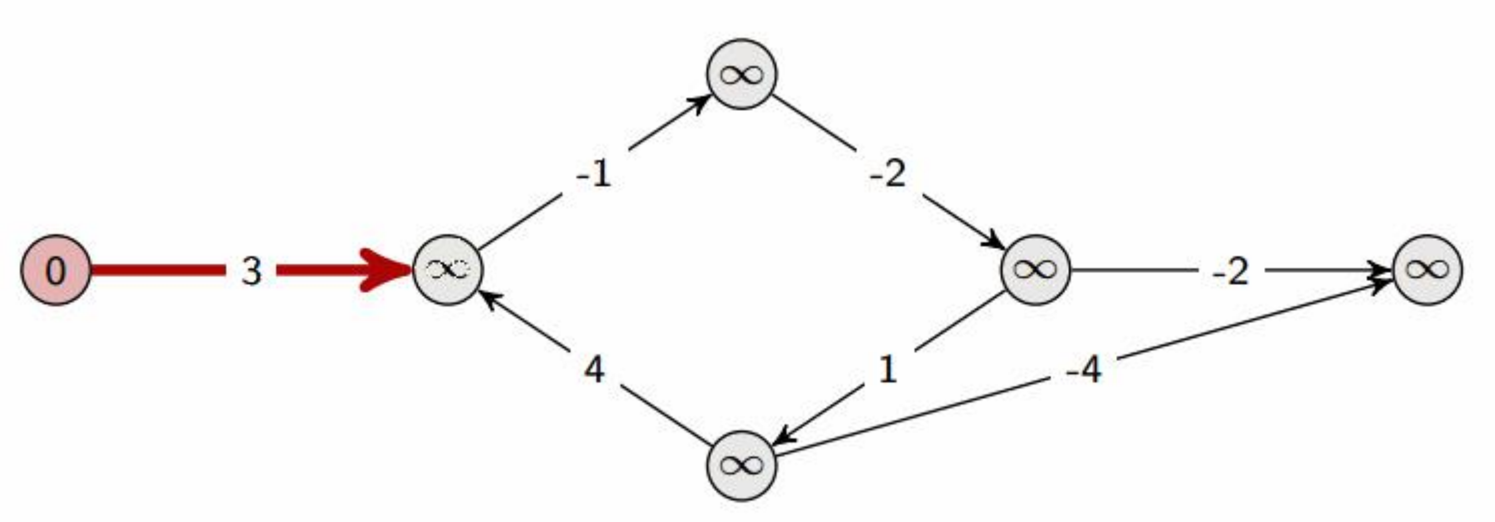
Bellman Ford



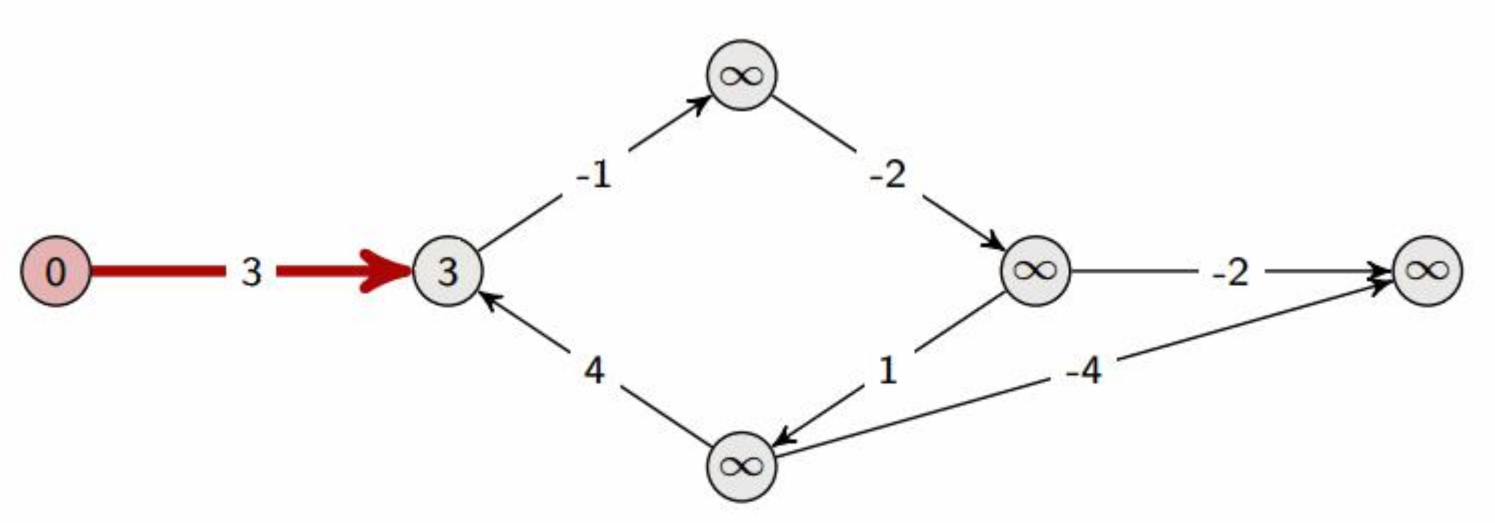
Bellman Ford



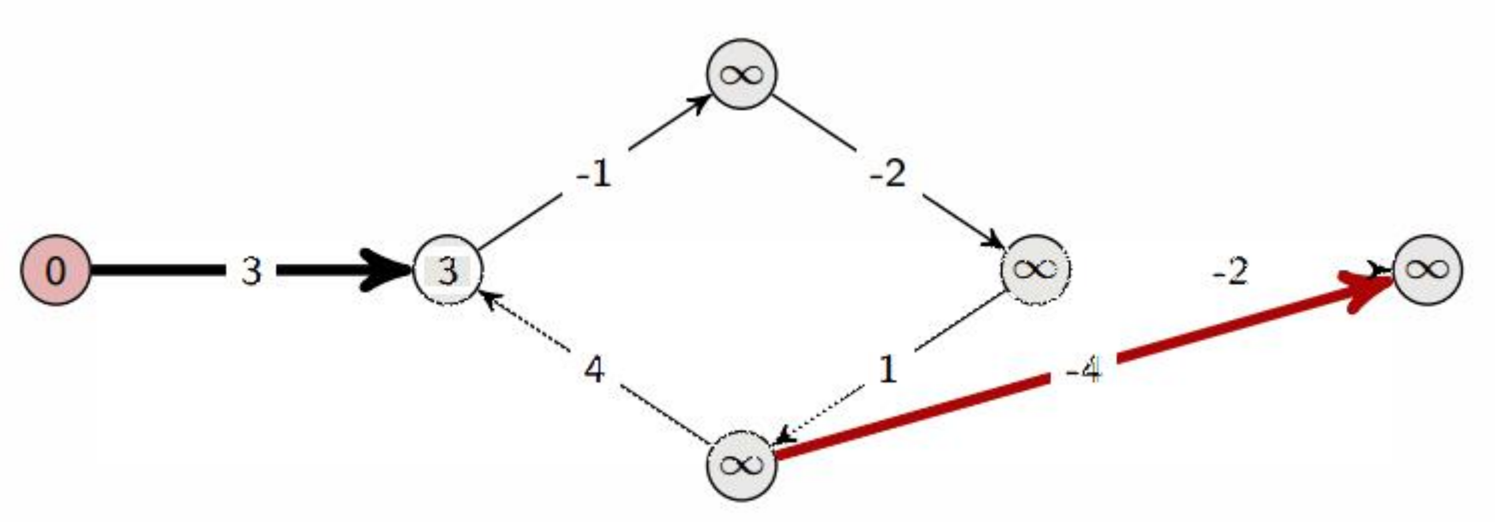
Bellman Ford



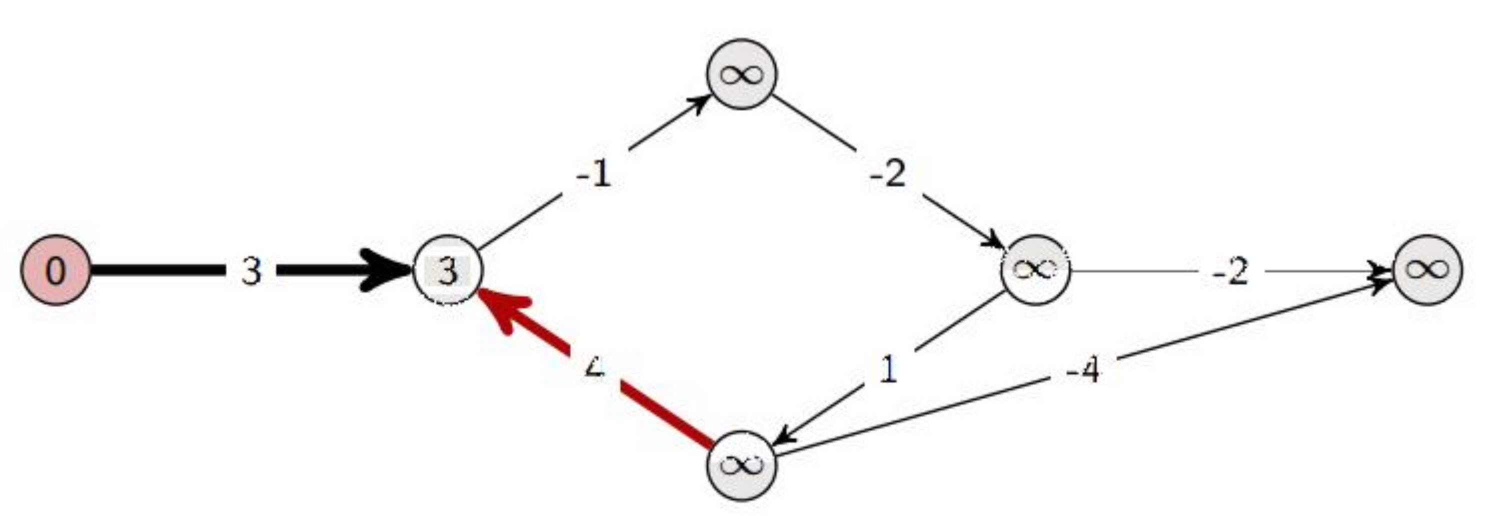
Bellman Ford



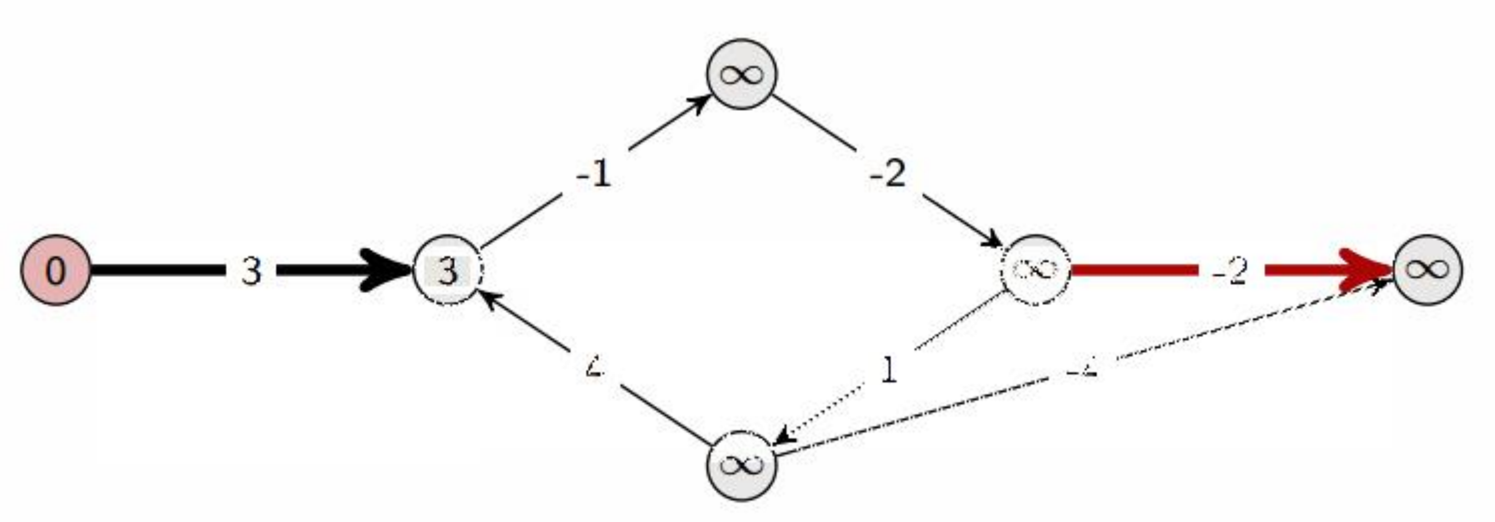
Bellman Ford



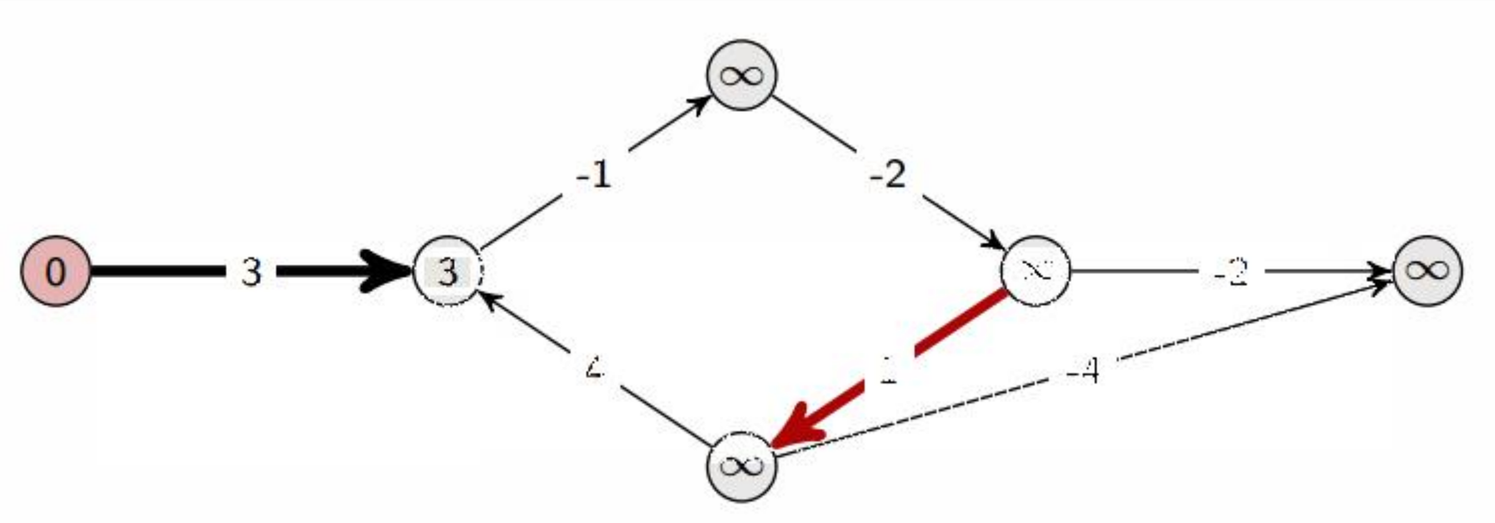
Bellman Ford



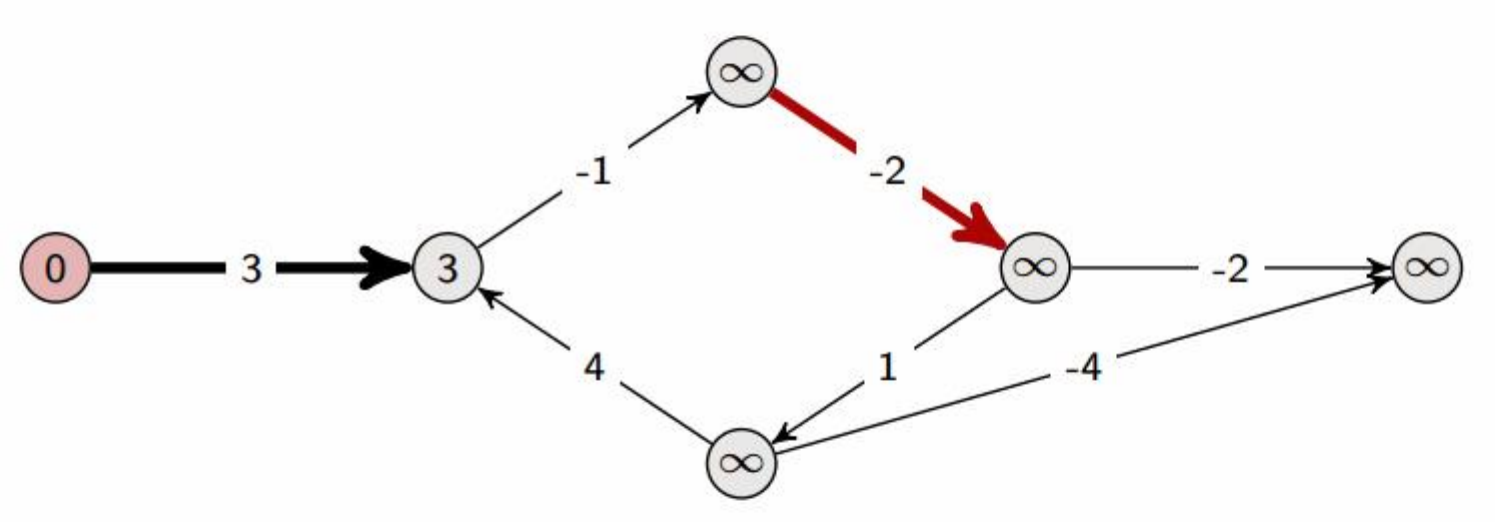
Bellman Ford



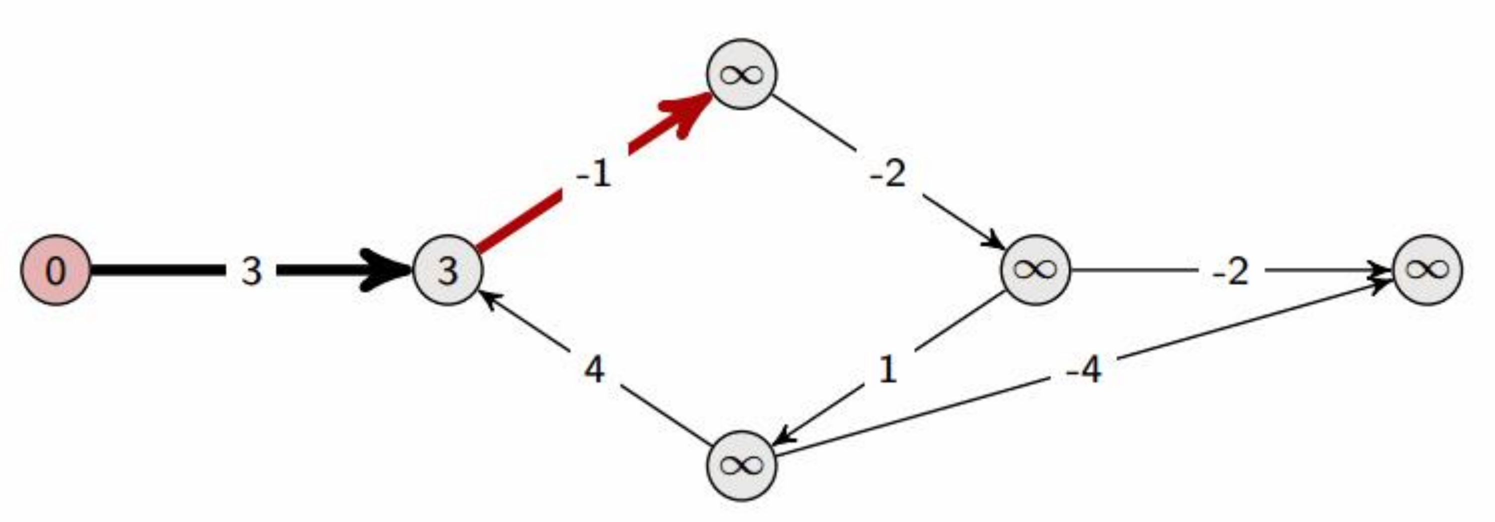
Bellman Ford



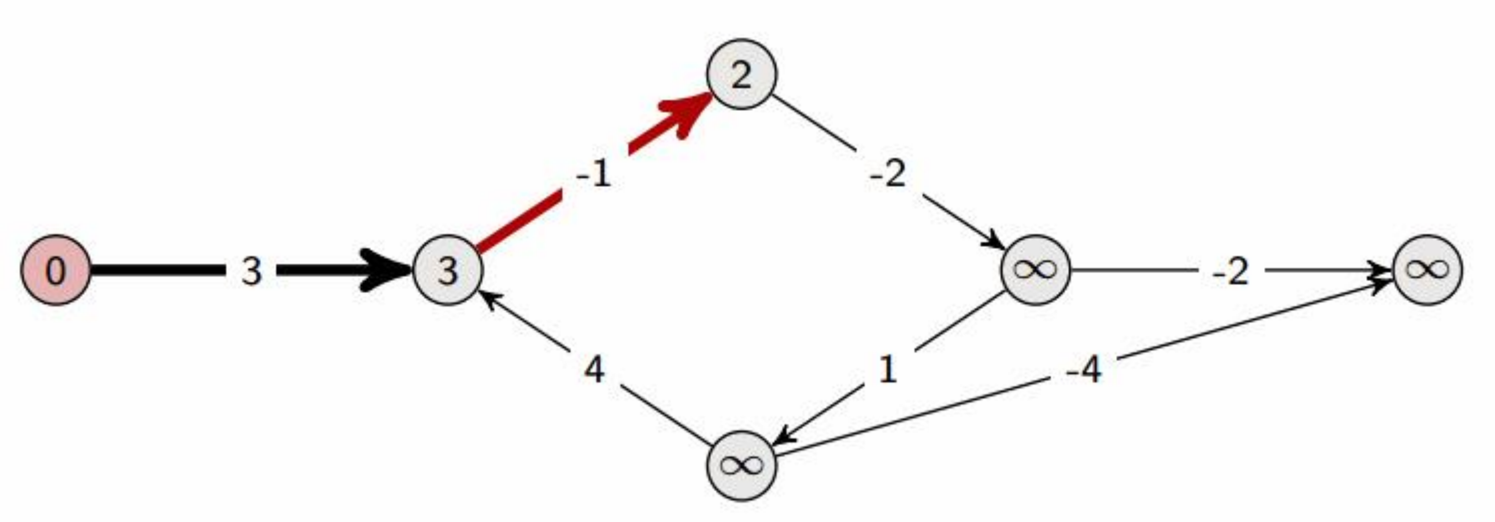
Bellman Ford



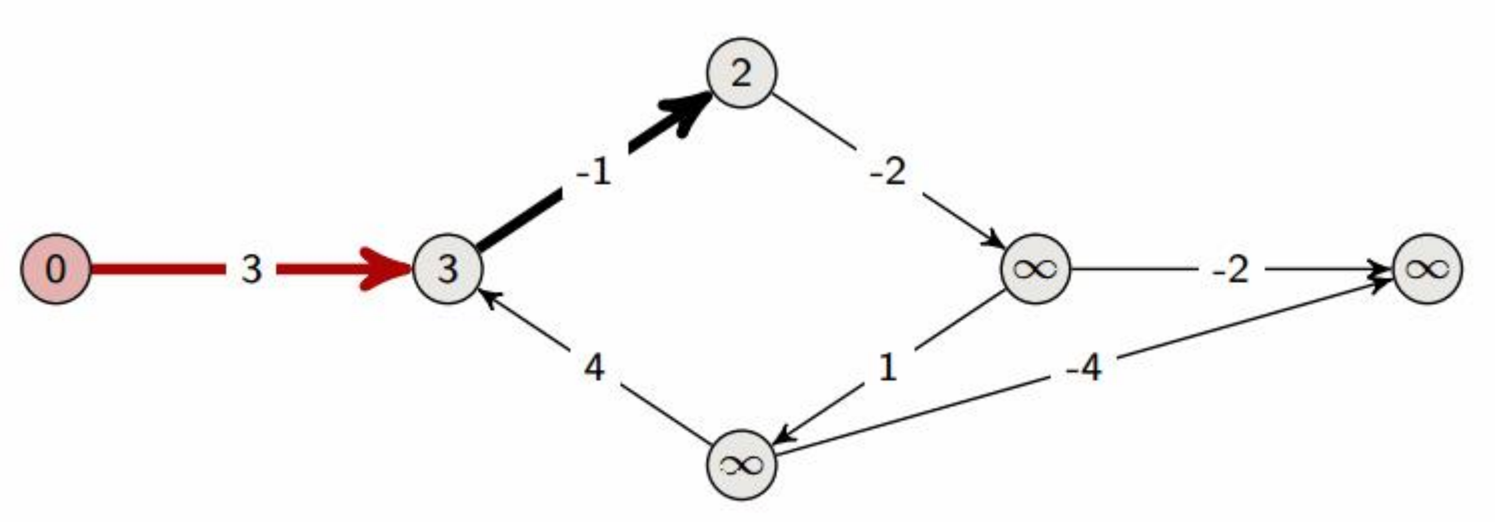
Bellman Ford



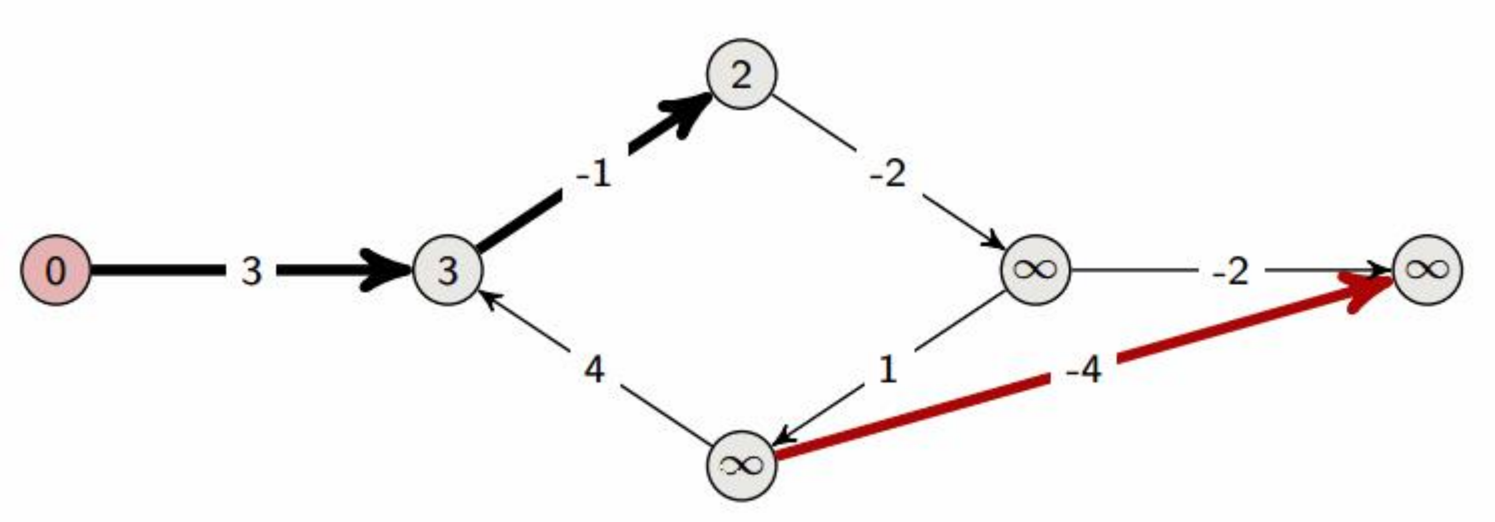
Bellman Ford



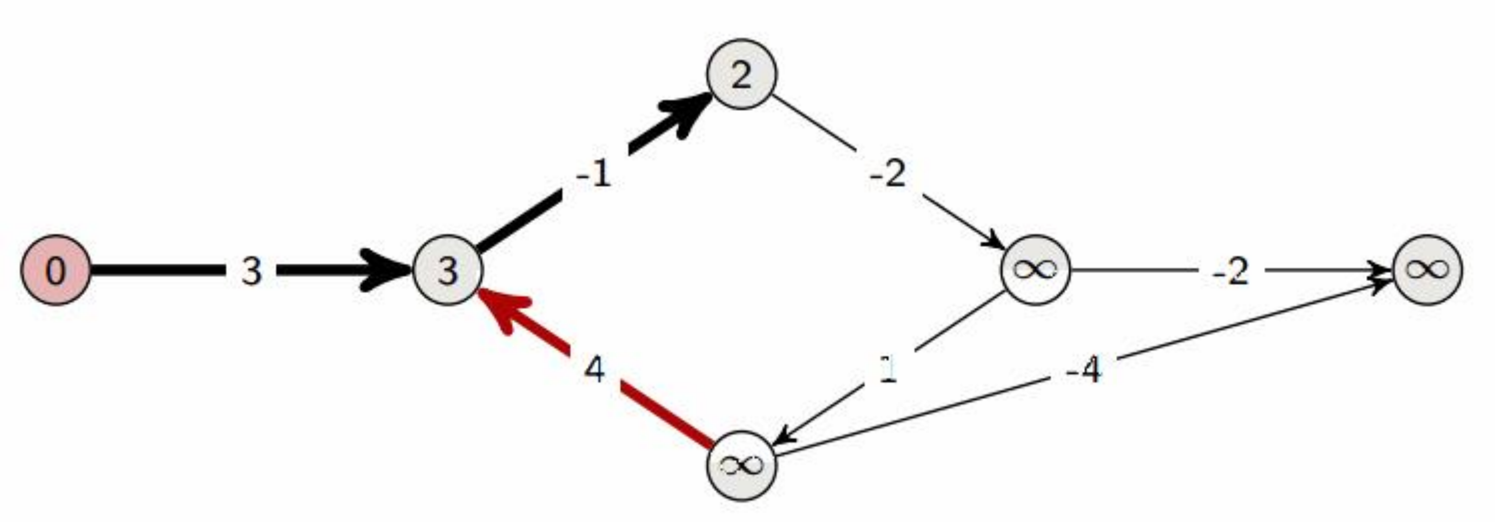
Bellman Ford



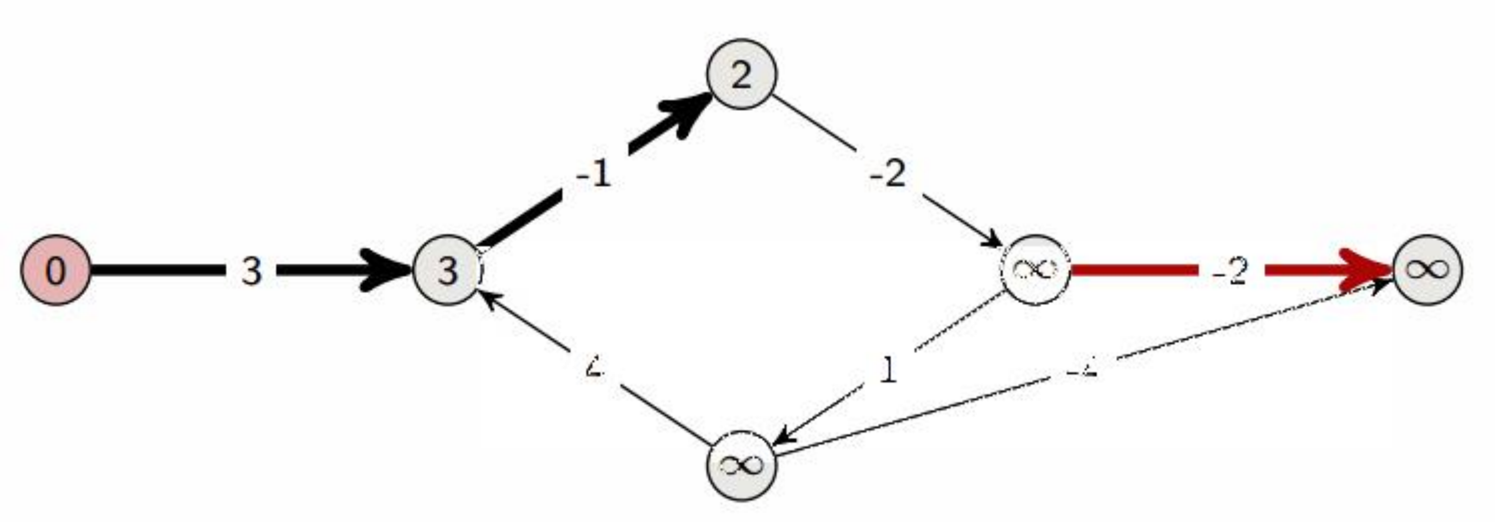
Bellman Ford



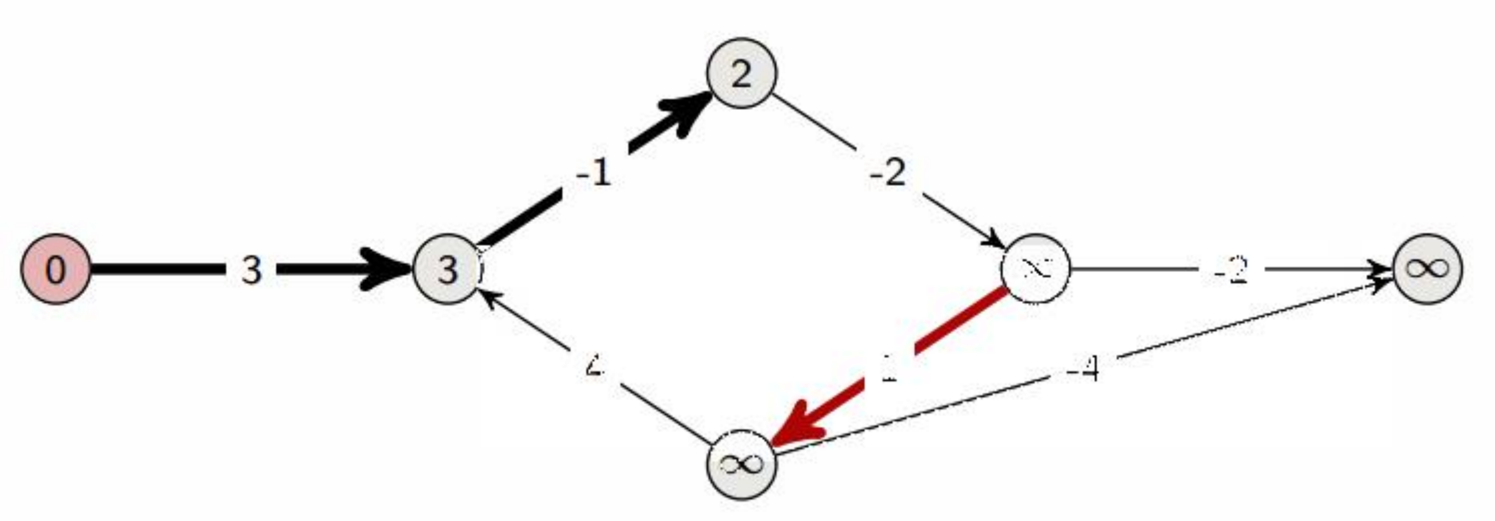
Bellman Ford



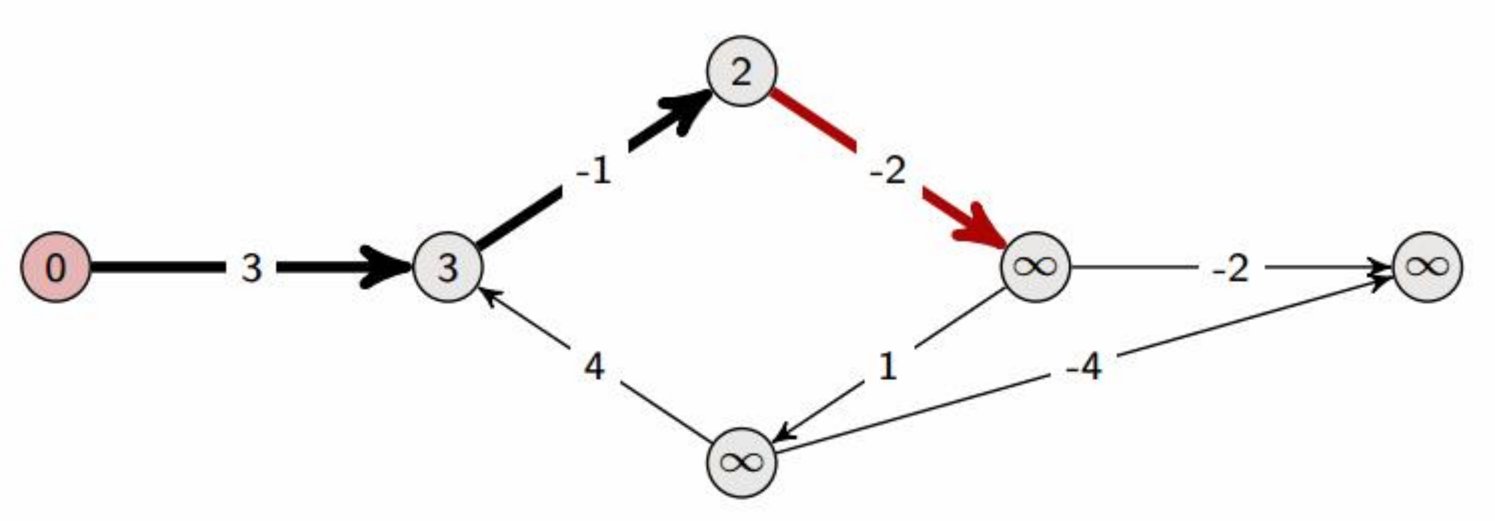
Bellman Ford



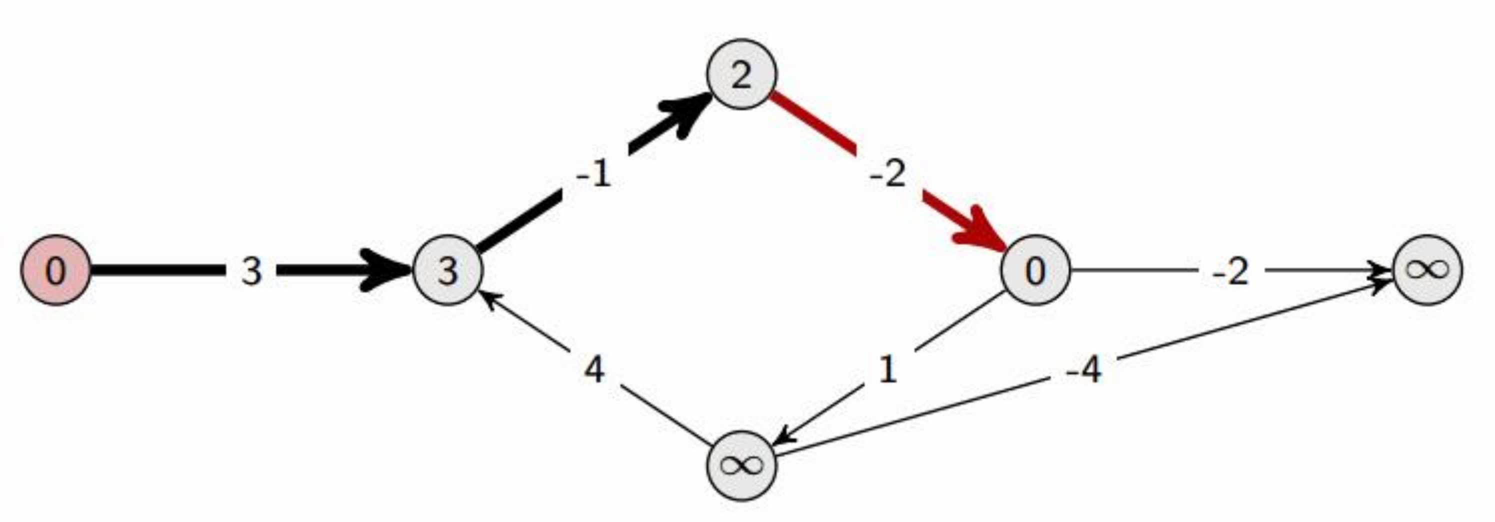
Bellman Ford



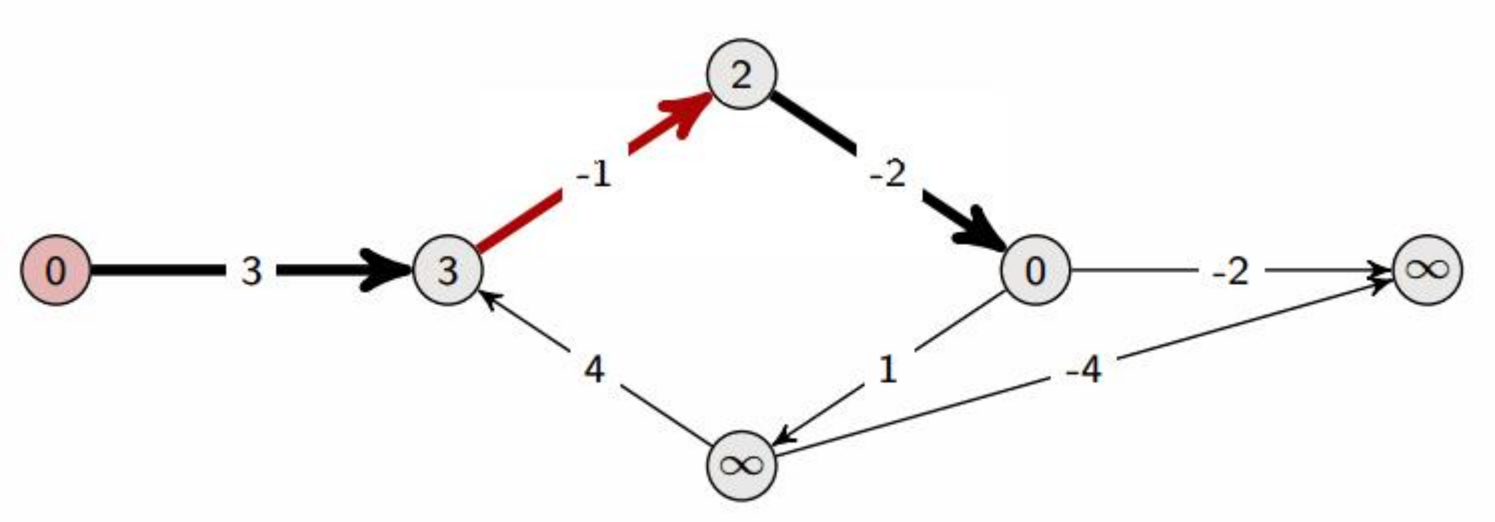
Bellman Ford



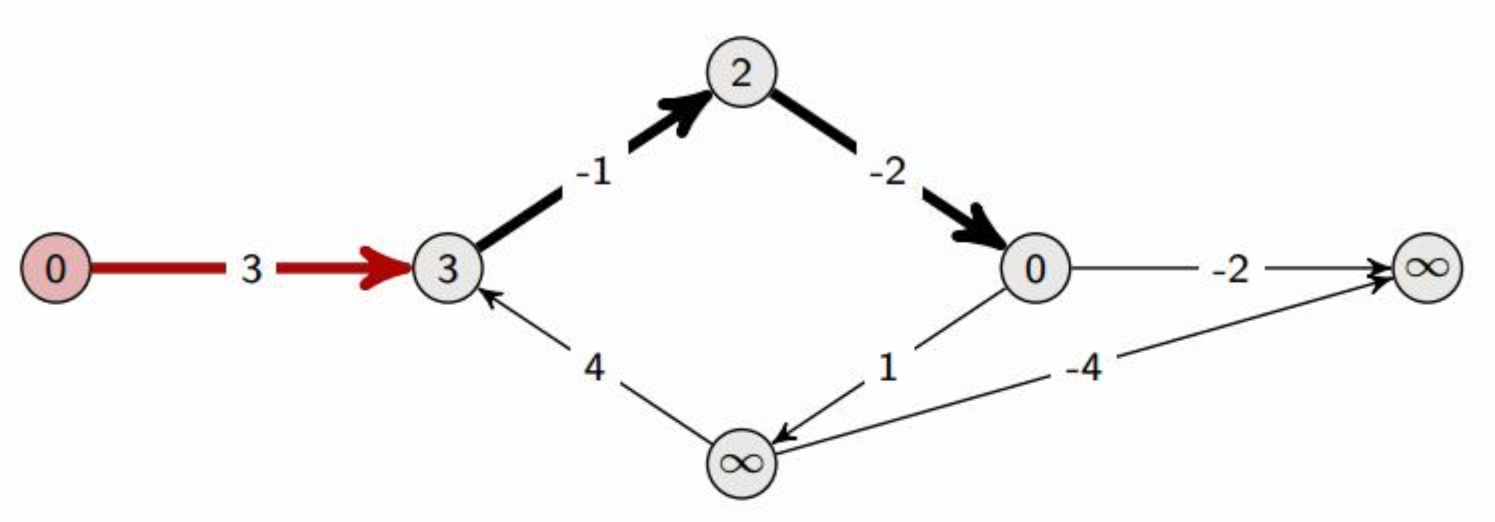
Bellman Ford



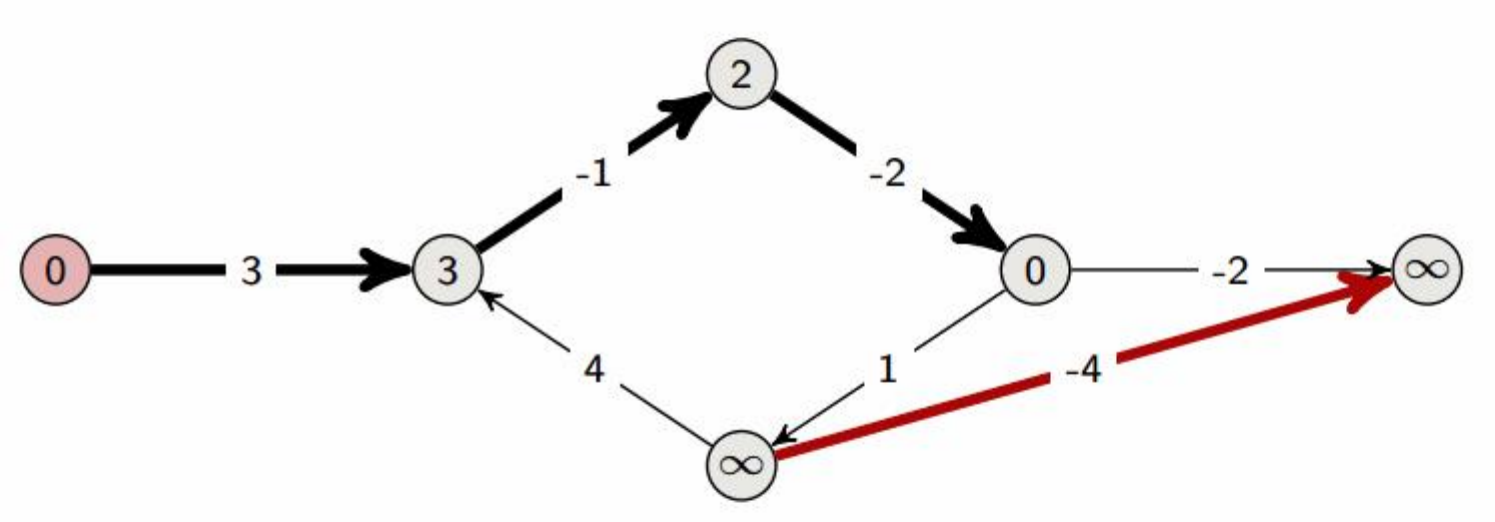
Bellman Ford



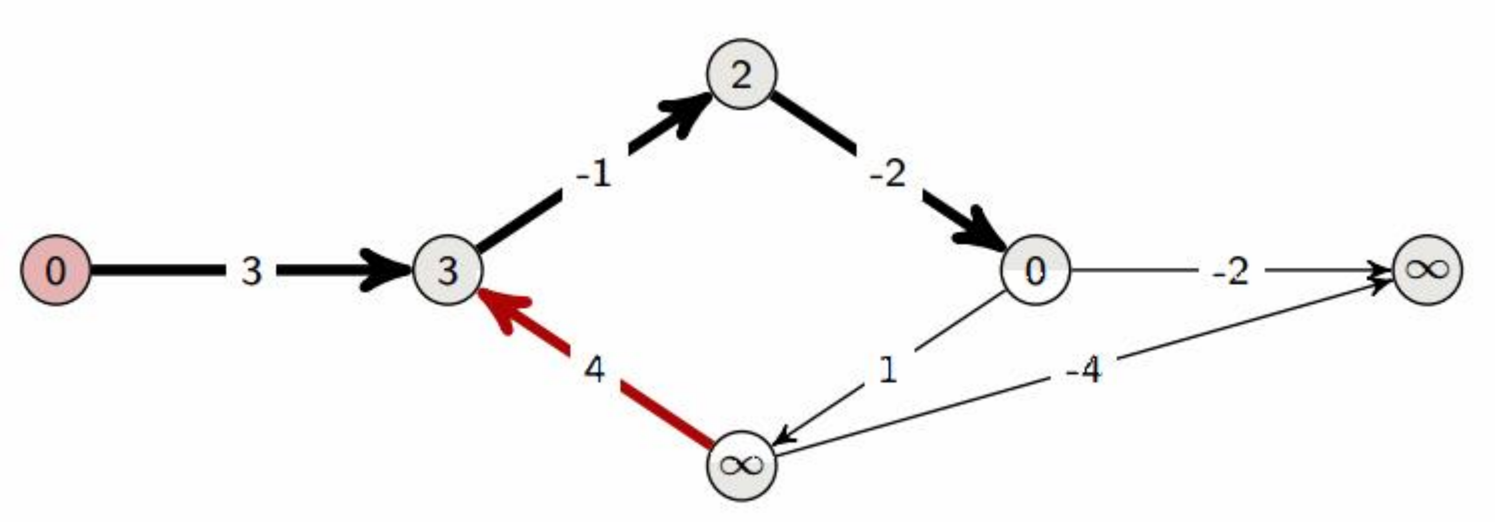
Bellman Ford



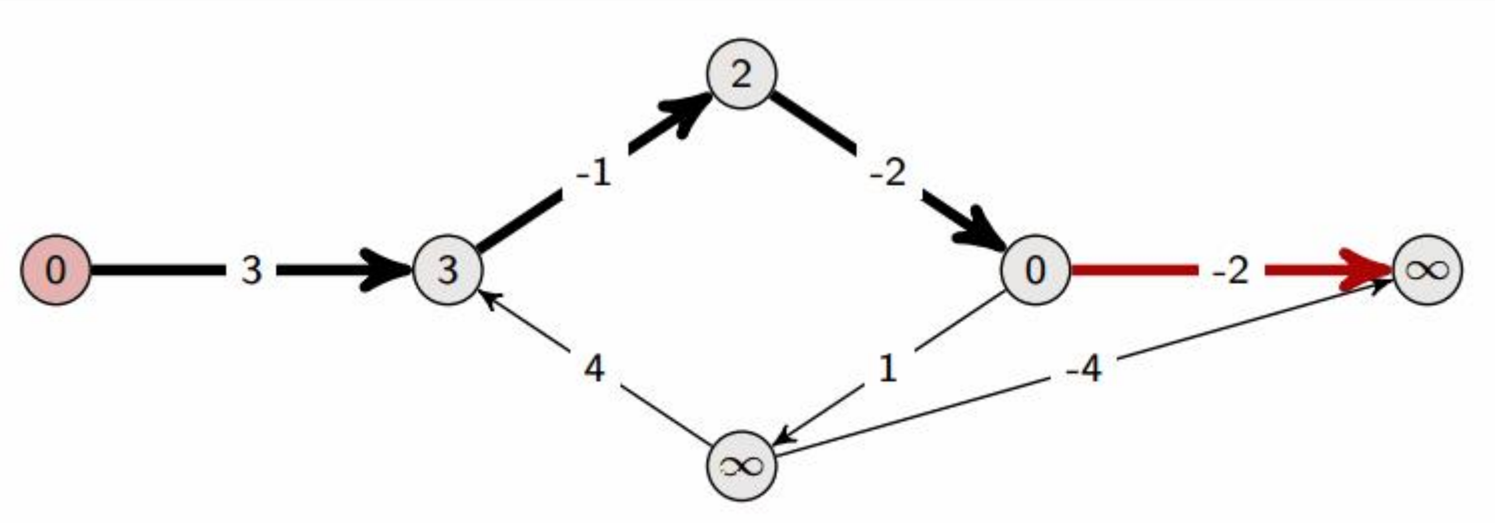
Bellman Ford



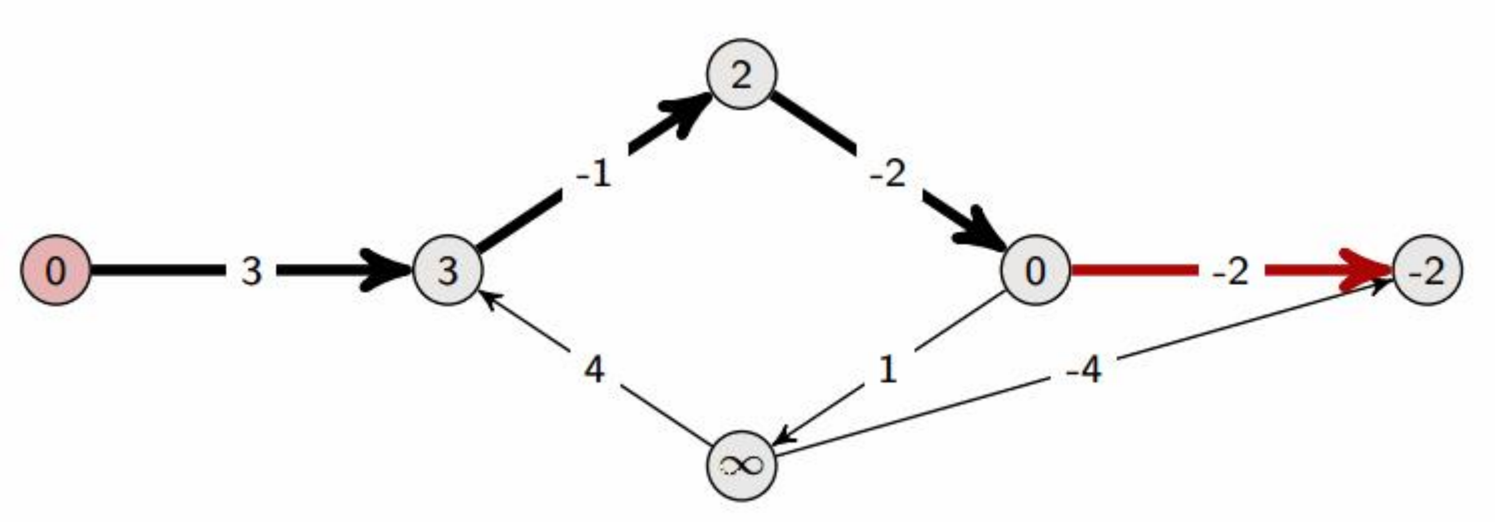
Bellman Ford



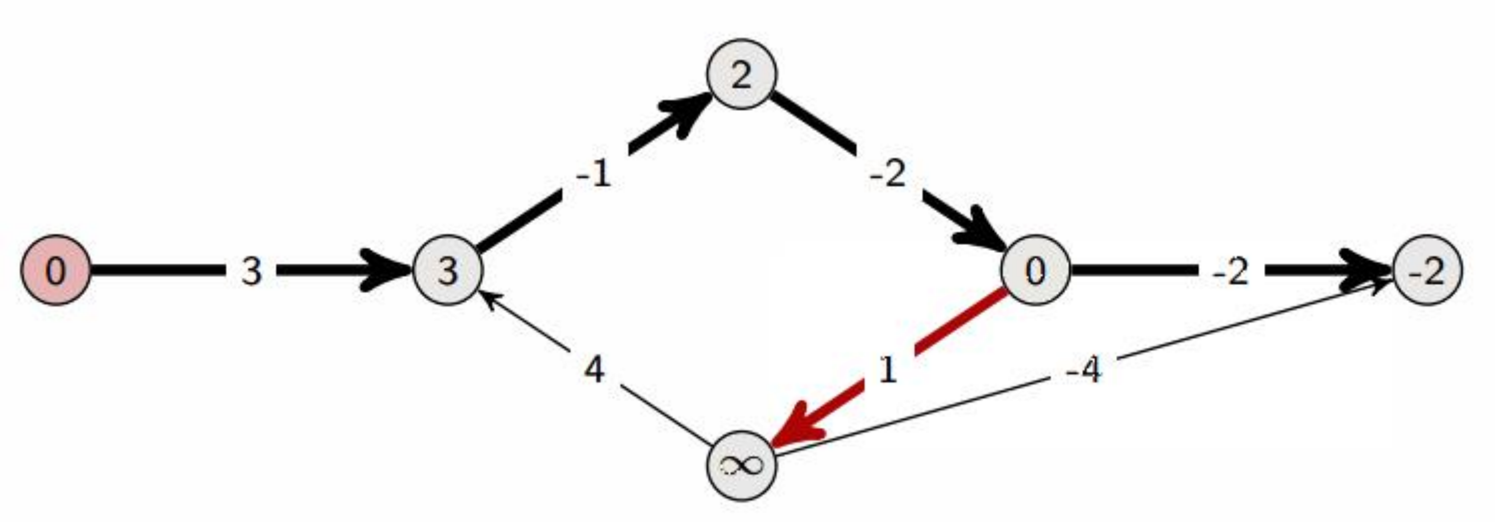
Bellman Ford



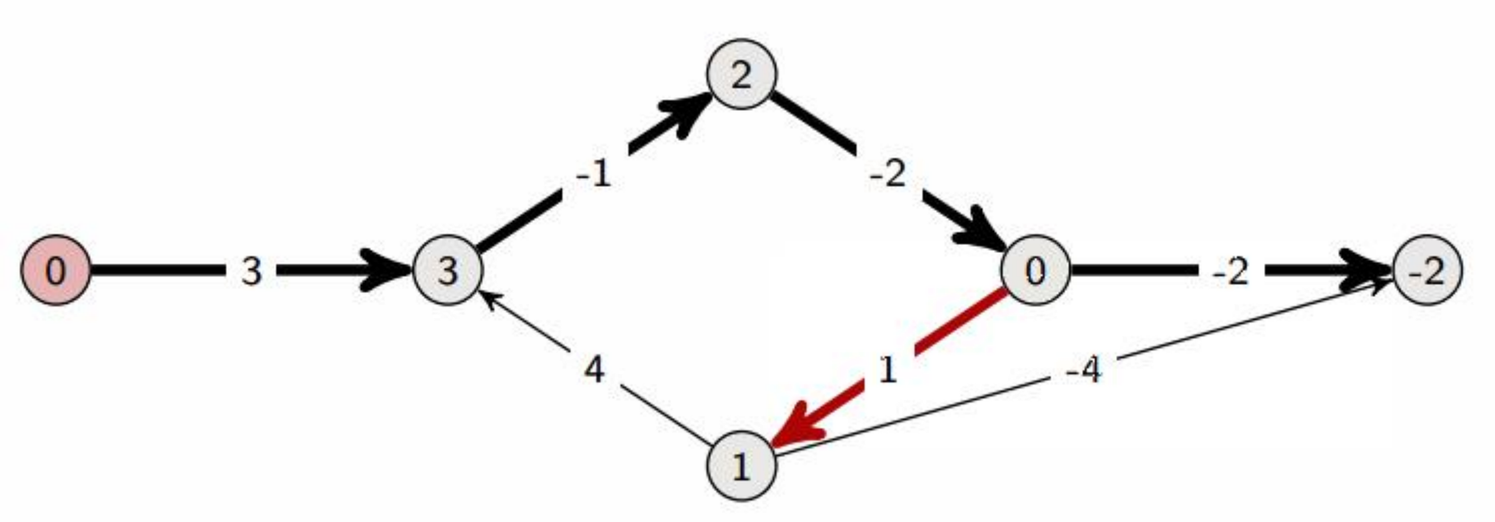
Bellman Ford



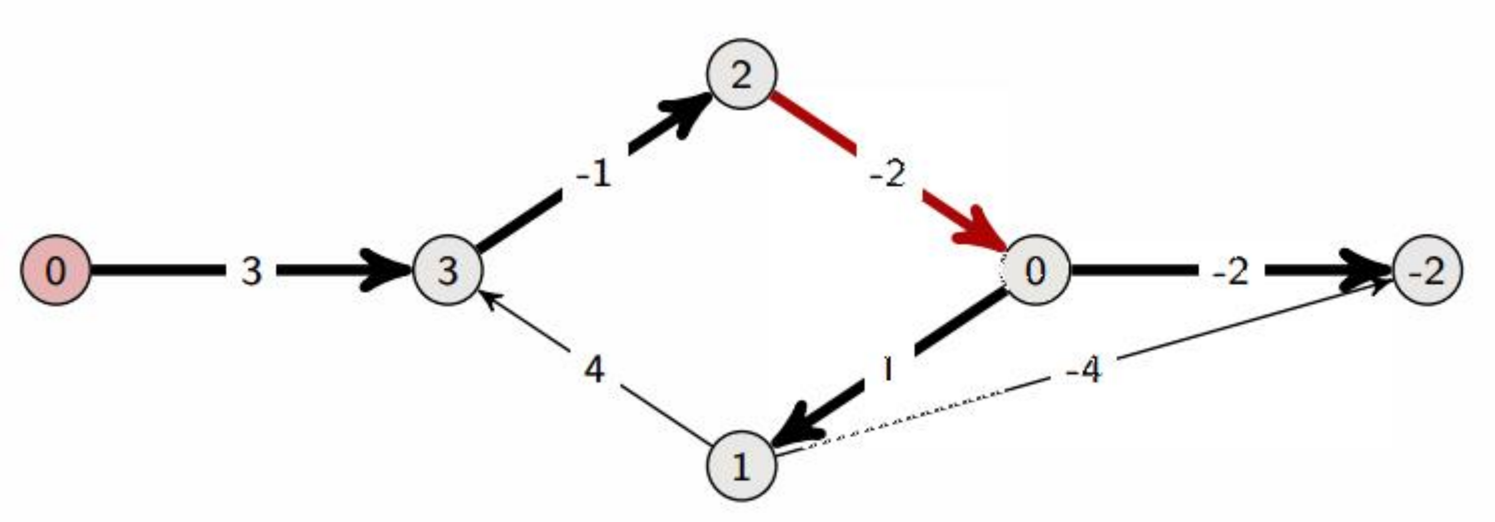
Bellman Ford



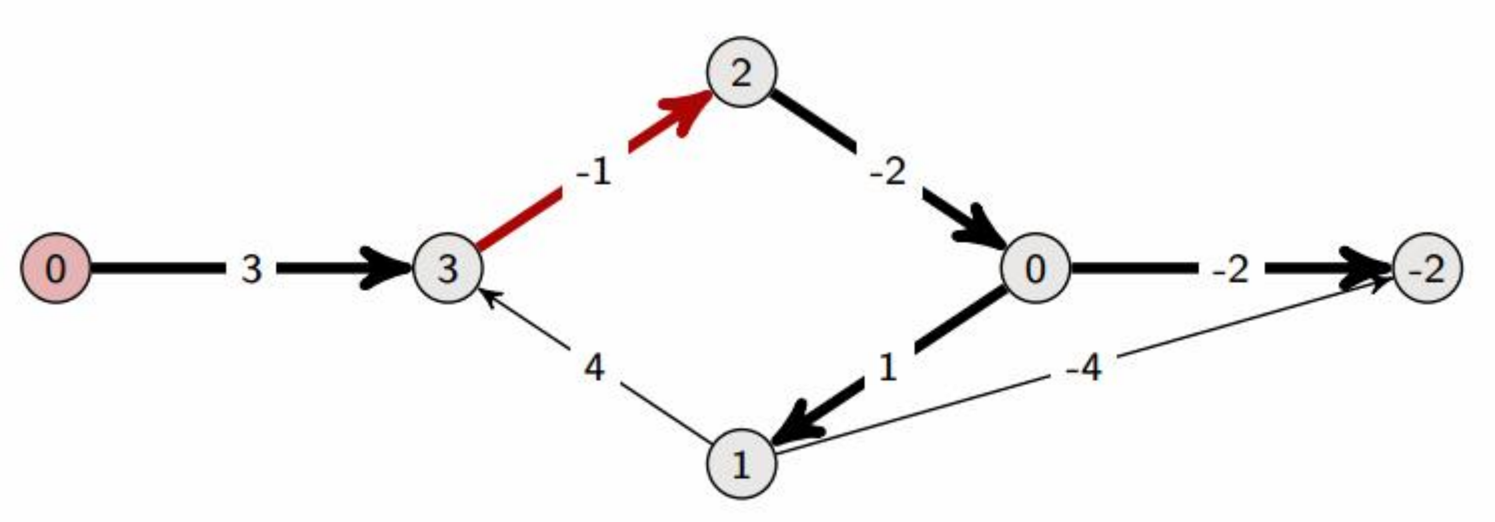
Bellman Ford



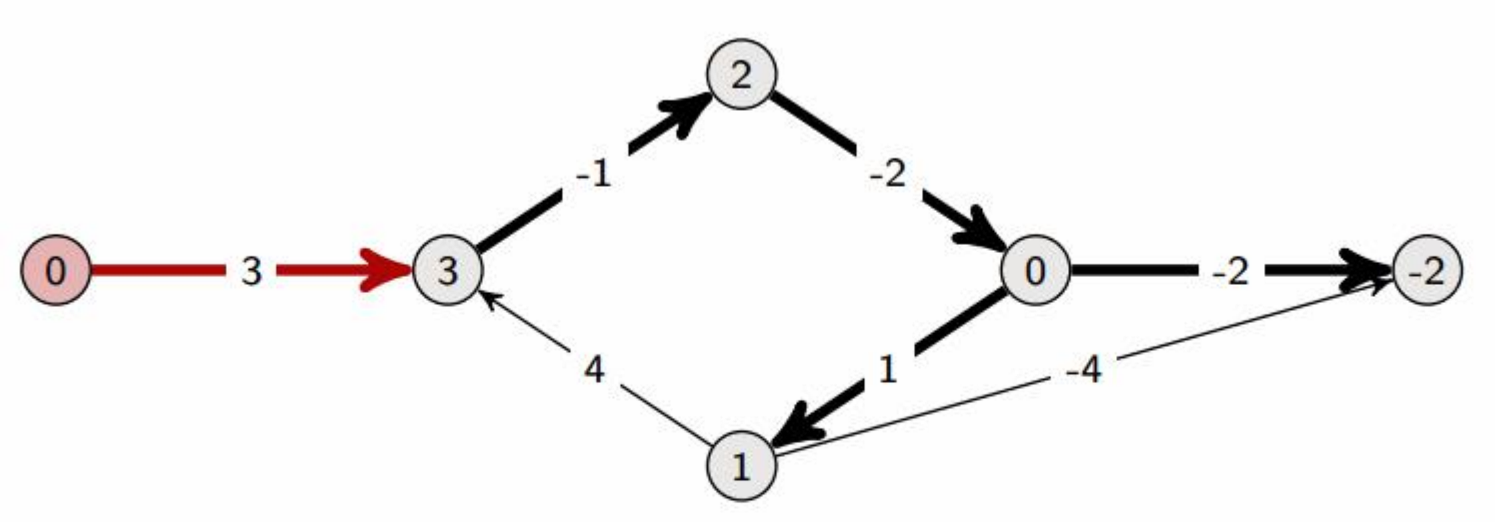
Bellman Ford



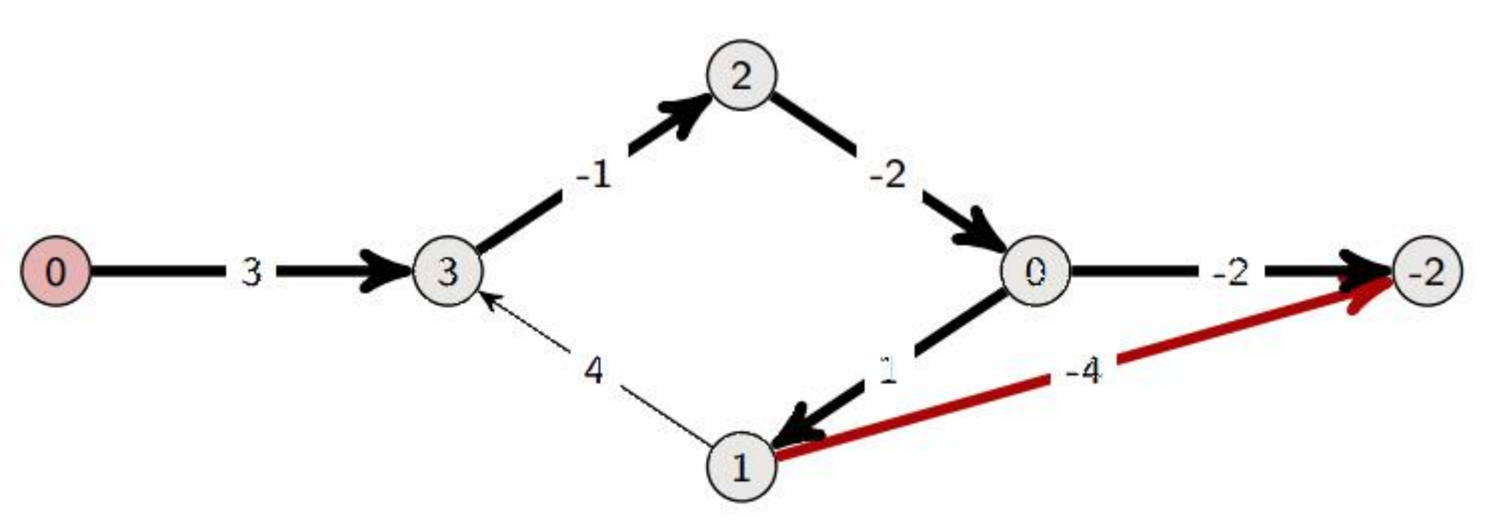
Bellman Ford



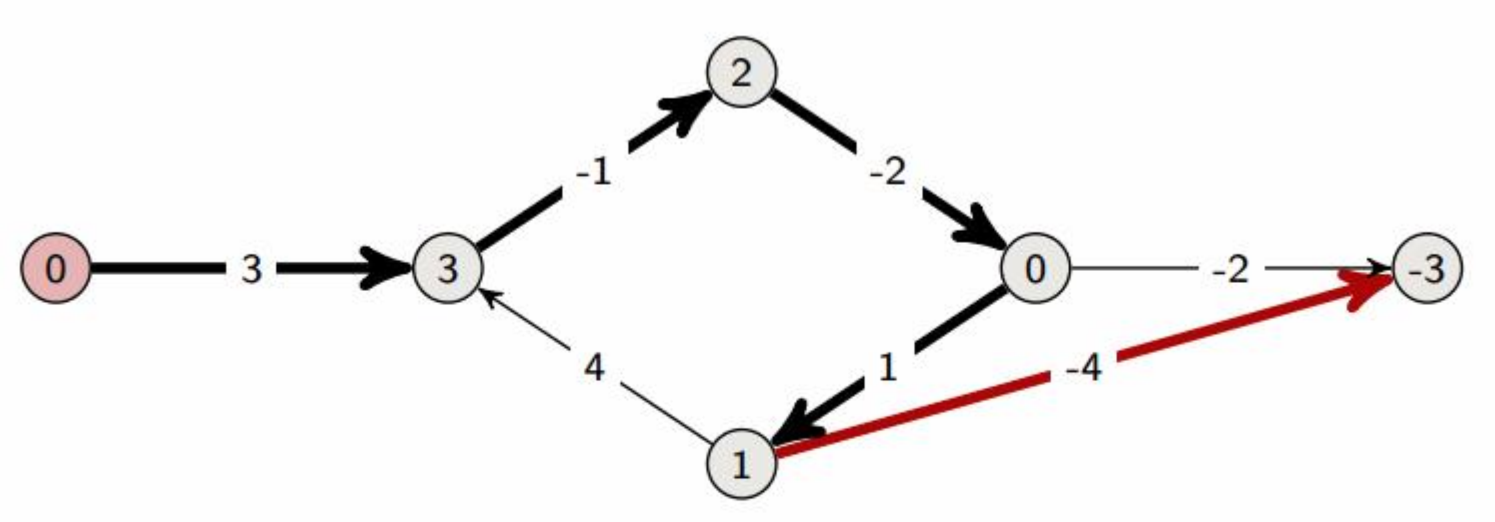
Bellman Ford



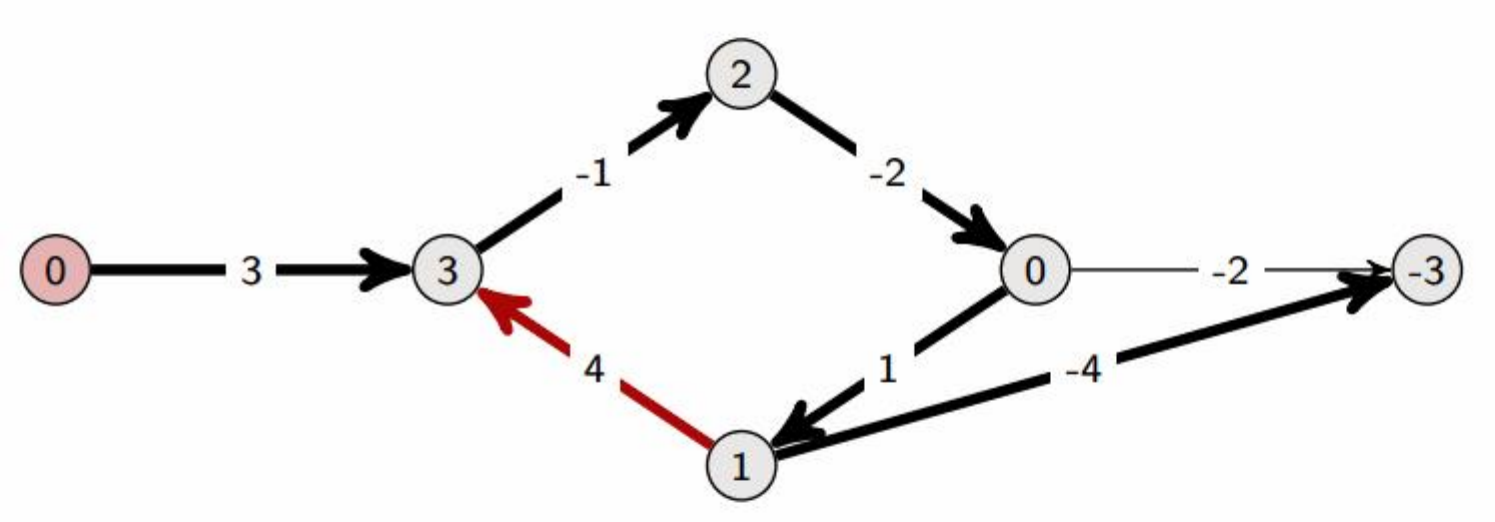
Bellman Ford



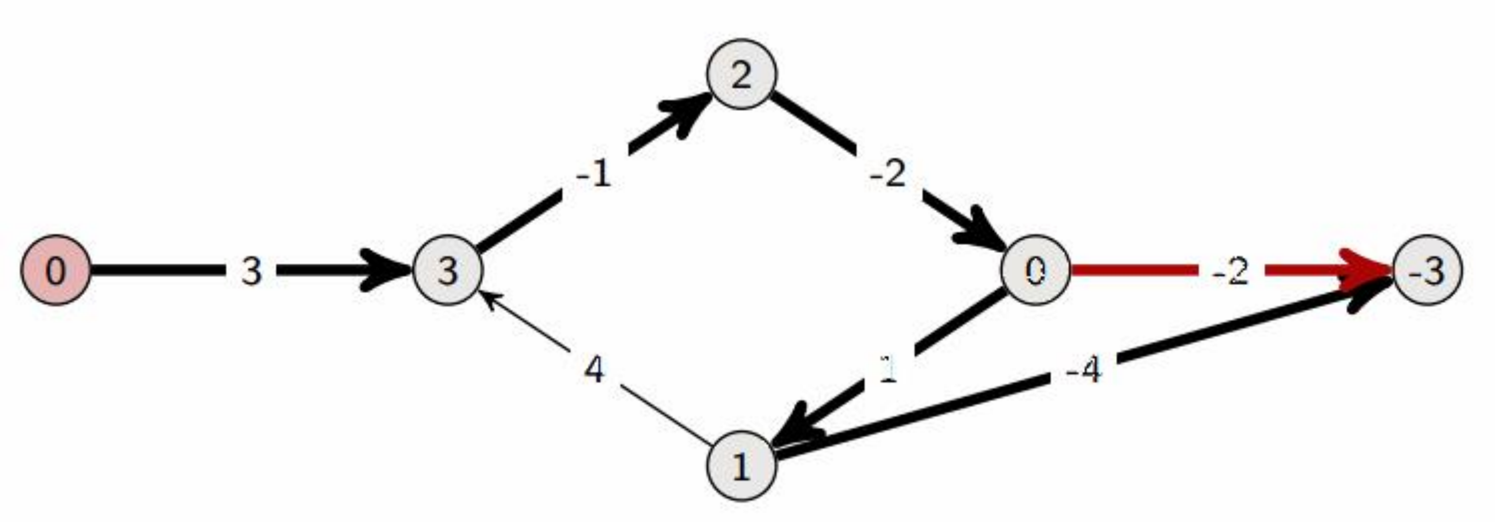
Bellman Ford



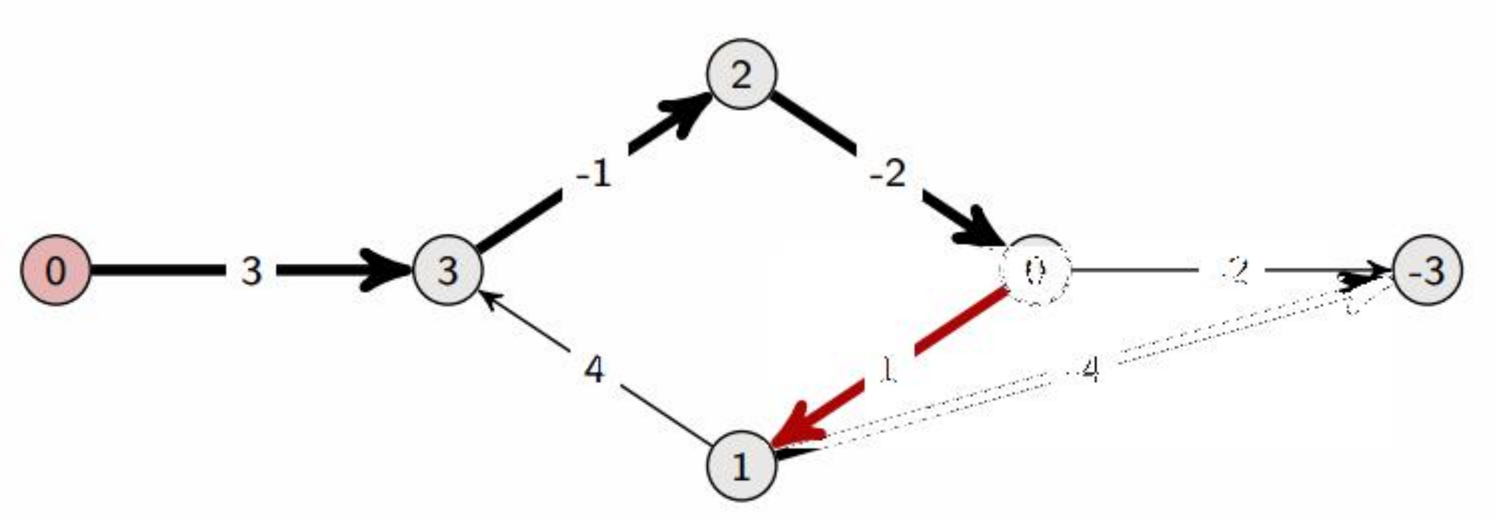
Bellman Ford



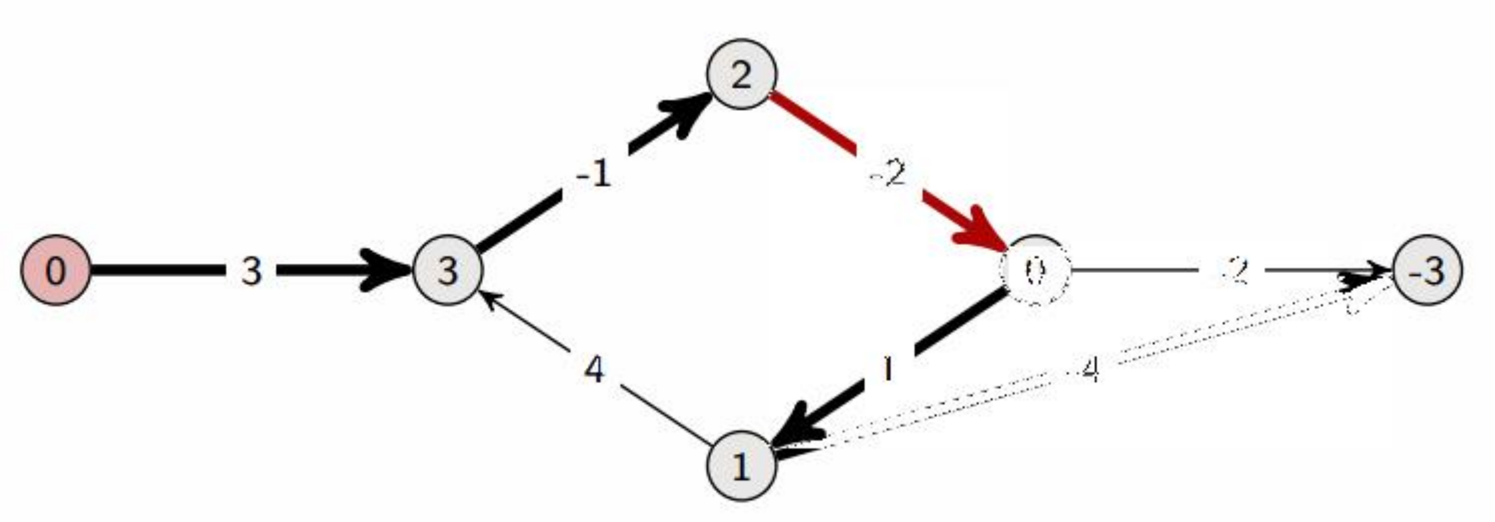
Bellman Ford



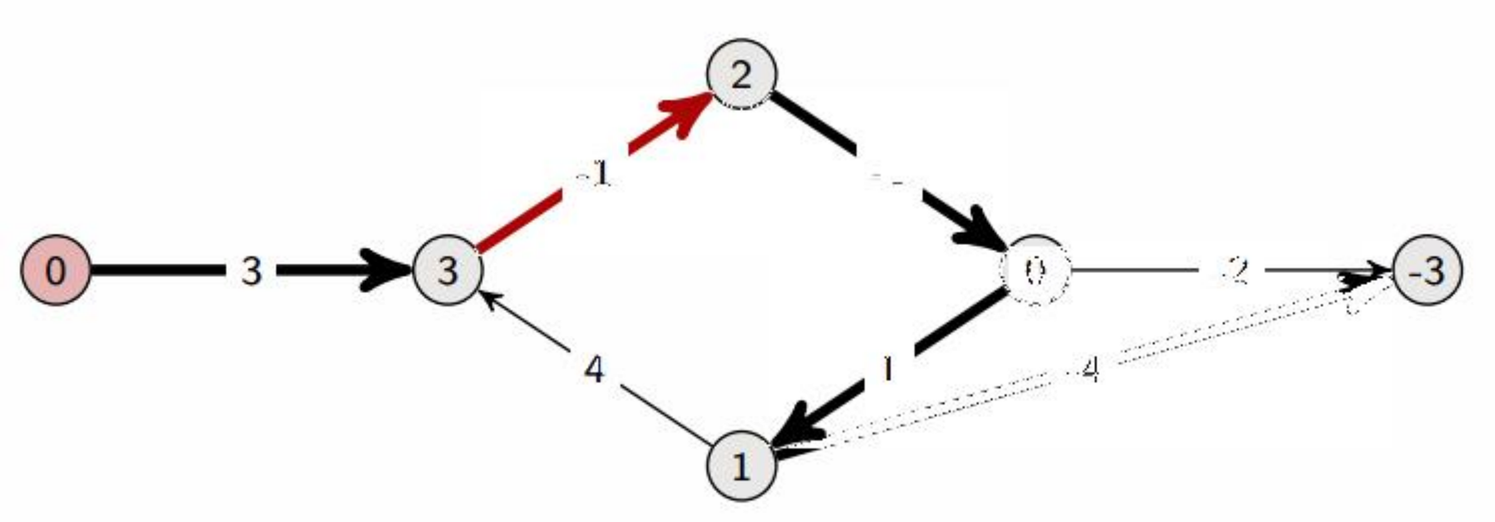
Bellman Ford



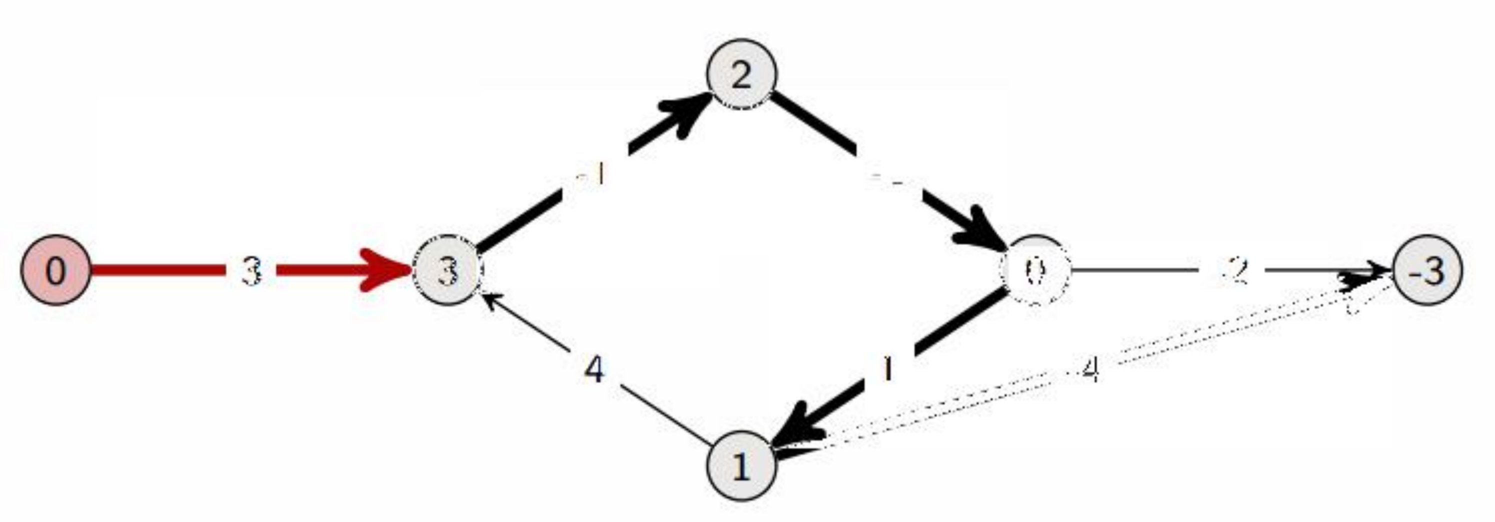
Bellman Ford



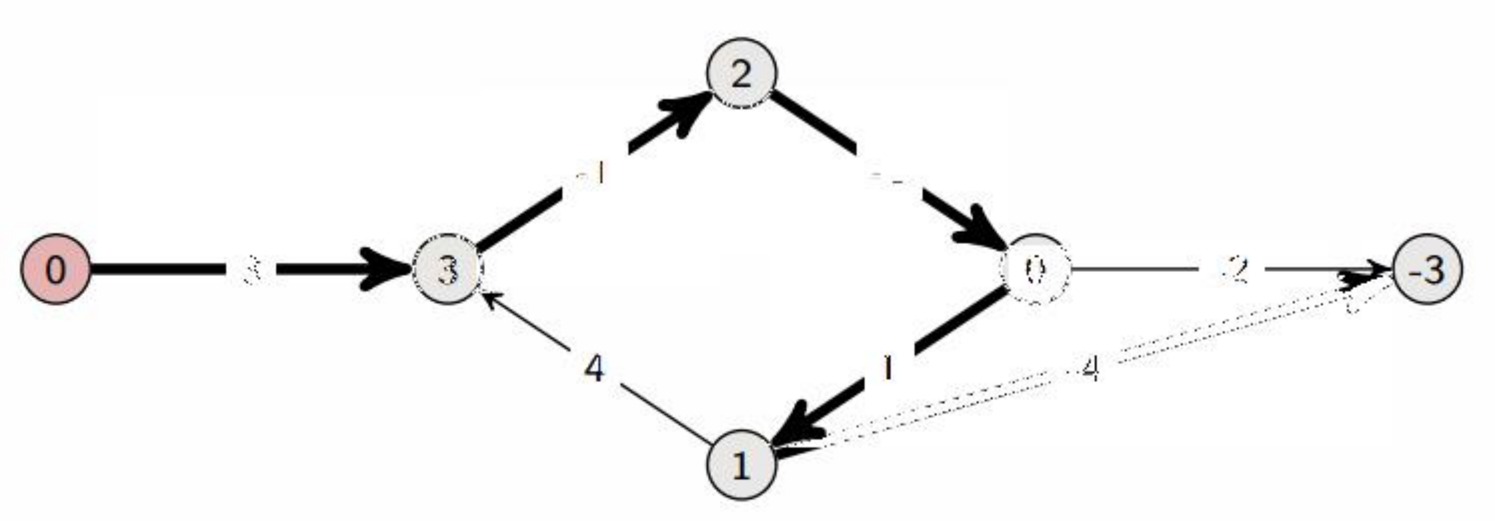
Bellman Ford



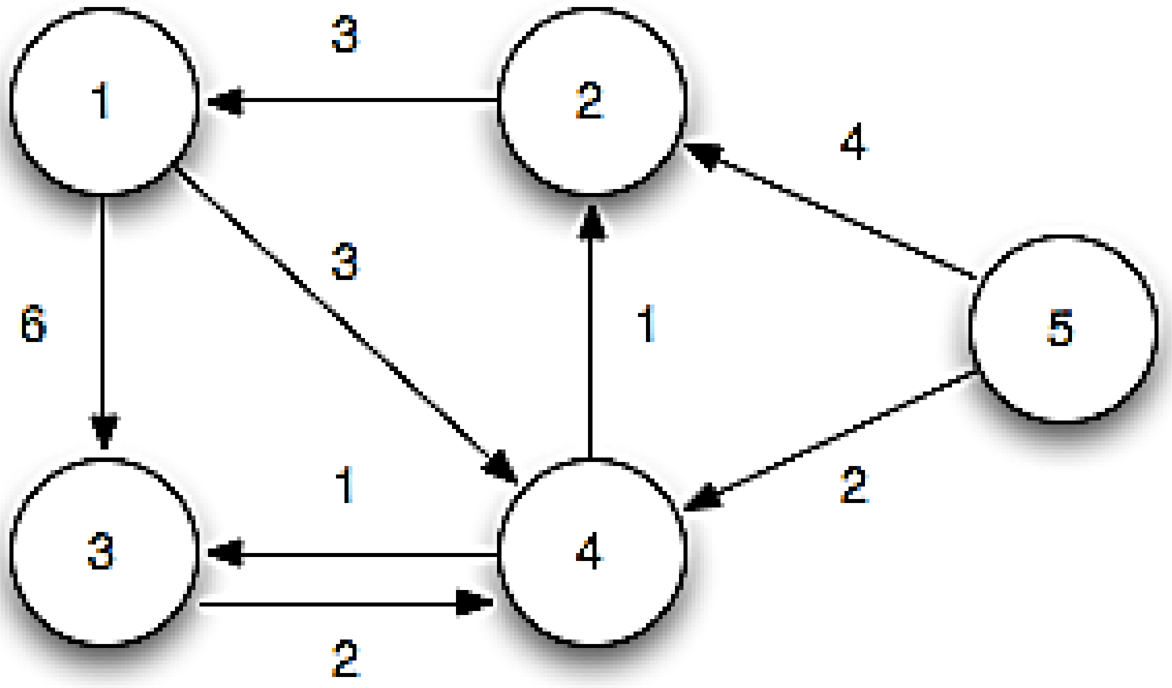
Bellman Ford



Bellman Ford



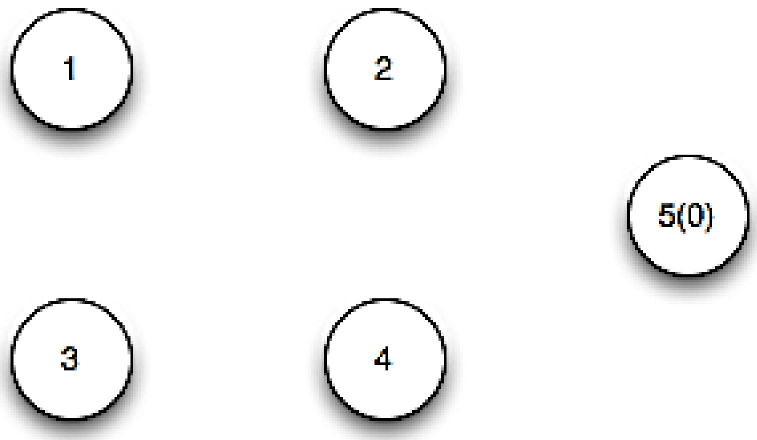
Örnek



(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

İlklendirme Aşaması

- Kaynak düğüm 5'in uzaklık değerine 0, diğerlerine ∞ atanır.

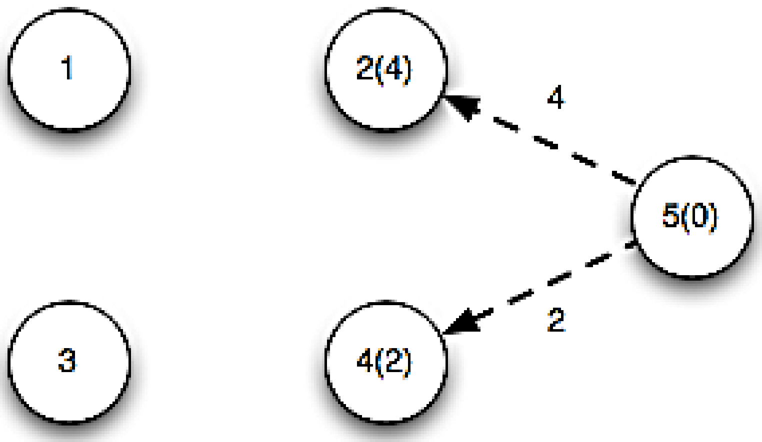


(1,3) = 6
(1,4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	∞	∞	∞	∞	0
π	/	/	/	/	/

Iteration 1

- (u5,u2) ve (u5,u4) kenarları incelenir. (relax) en kısa yollar sırasıyla 2 ve 4 olarak güncellenir.

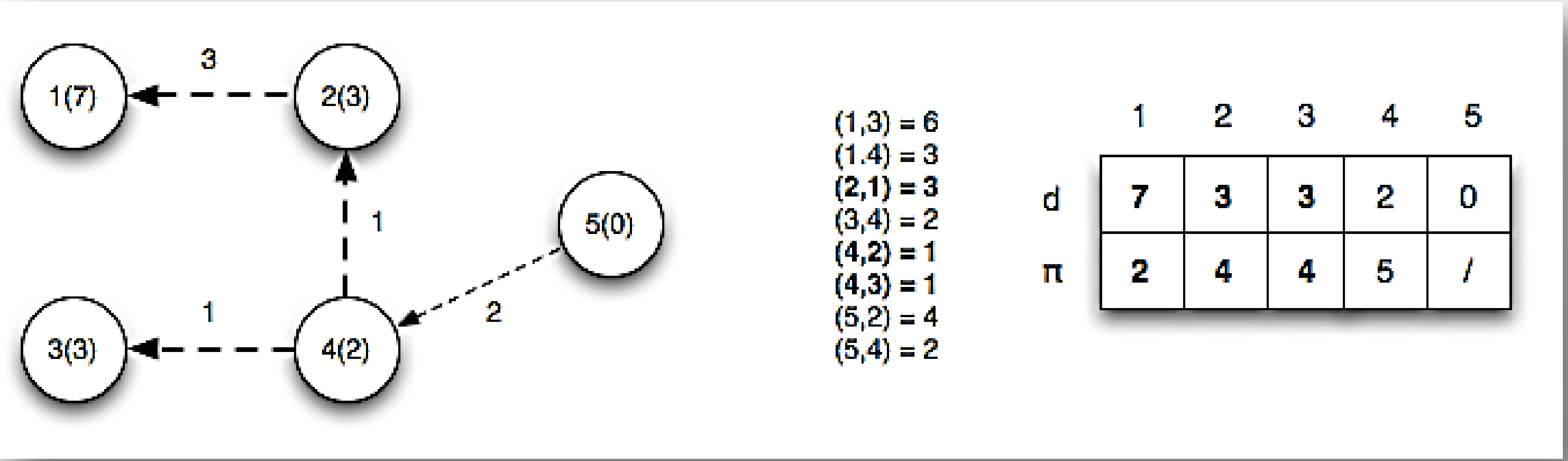


(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	∞	4	∞	2	0
π	/	5	/	5	/

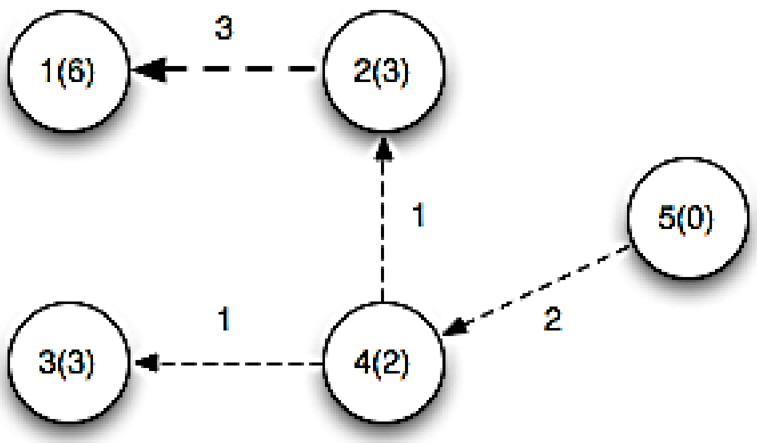
Iteration 2

- (u2,u1), (u4,u2) ve (u4,u3) kenarları incelenir. (relax) en kısa yollar sırasıyla 1, 2, 4 olarak güncellenir. (u4,u2) kenarı daha kısa bir yol bulur.



Iteration 3

- (u2,u1) kenarı (bir önceki adımda düğüm 2'ye daha kısa bir yol bulunduğu için) incelenir. (relax)

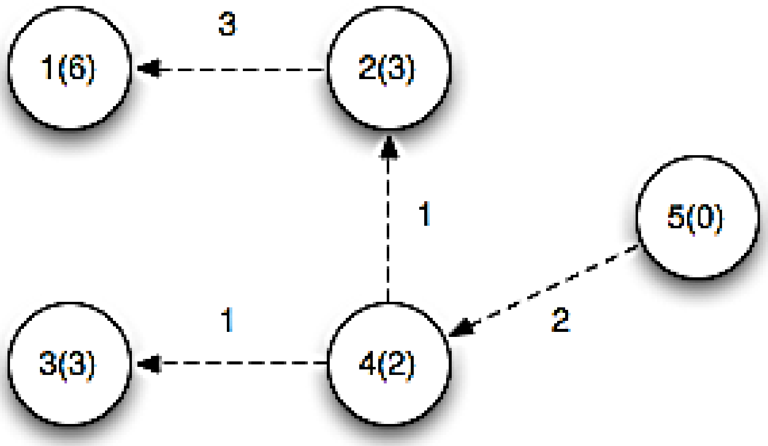


(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

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

Iteration 4

- Daha kısa bir yol bulunamadı. (No edges relax)

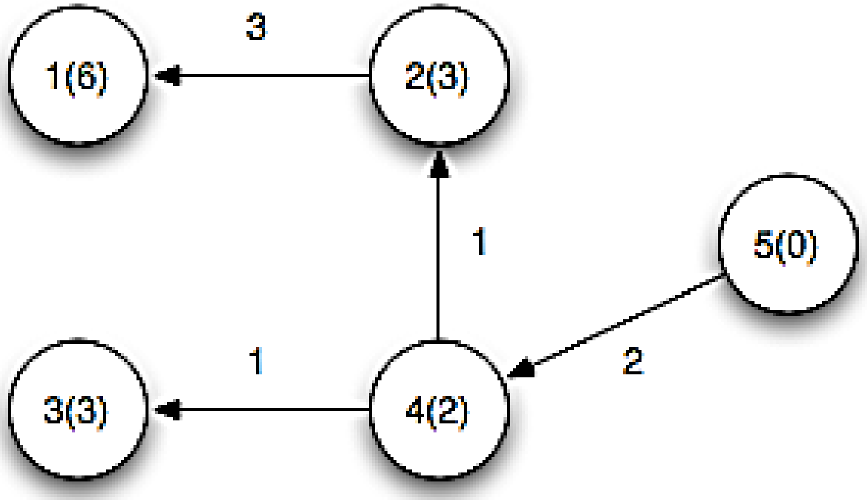


(1,3) = 6
(1.4) = 3
(2,1) = 3
(3,4) = 2
(4,2) = 1
(4,3) = 1
(5,2) = 4
(5,4) = 2

	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

Son Durum

- Düğüm 5'ten diğer düğümlere olan en kısa yollar



	1	2	3	4	5
d	6	3	3	2	0
π	2	4	4	5	/

Negatif Döngü Konrolü

- Her kenar için son bir defa inceleme (*relaxation*) yapılır.
- Eğer kısa yol bulunursa negatif döngü vardır.

$$v3.d > u1.d + w(1,3) \Rightarrow 4 \not> 6 + 6 = 12 \checkmark$$
$$v4.d > u1.d + w(1,4) \Rightarrow 2 \not> 6 + 3 = 9 \checkmark$$
$$v1.d > u2.d + w(2,1) \Rightarrow 6 \not> 3 + 3 = 6 \checkmark$$
$$v4.d > u3.d + w(3,4) \Rightarrow 2 \not> 3 + 2 = 5 \checkmark$$
$$v2.d > u4.d + w(4,2) \Rightarrow 3 \not> 2 + 1 = 3 \checkmark$$
$$v3.d > u4.d + w(4,3) \Rightarrow 3 \not> 2 + 1 = 3 \checkmark$$
$$v2.d > u5.d + w(5,2) \Rightarrow 3 \not> 0 + 4 = 4 \checkmark$$
$$v4.d > u5.d + w(5,4) \Rightarrow 2 \not> 0 + 2 = 2 \checkmark$$

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.

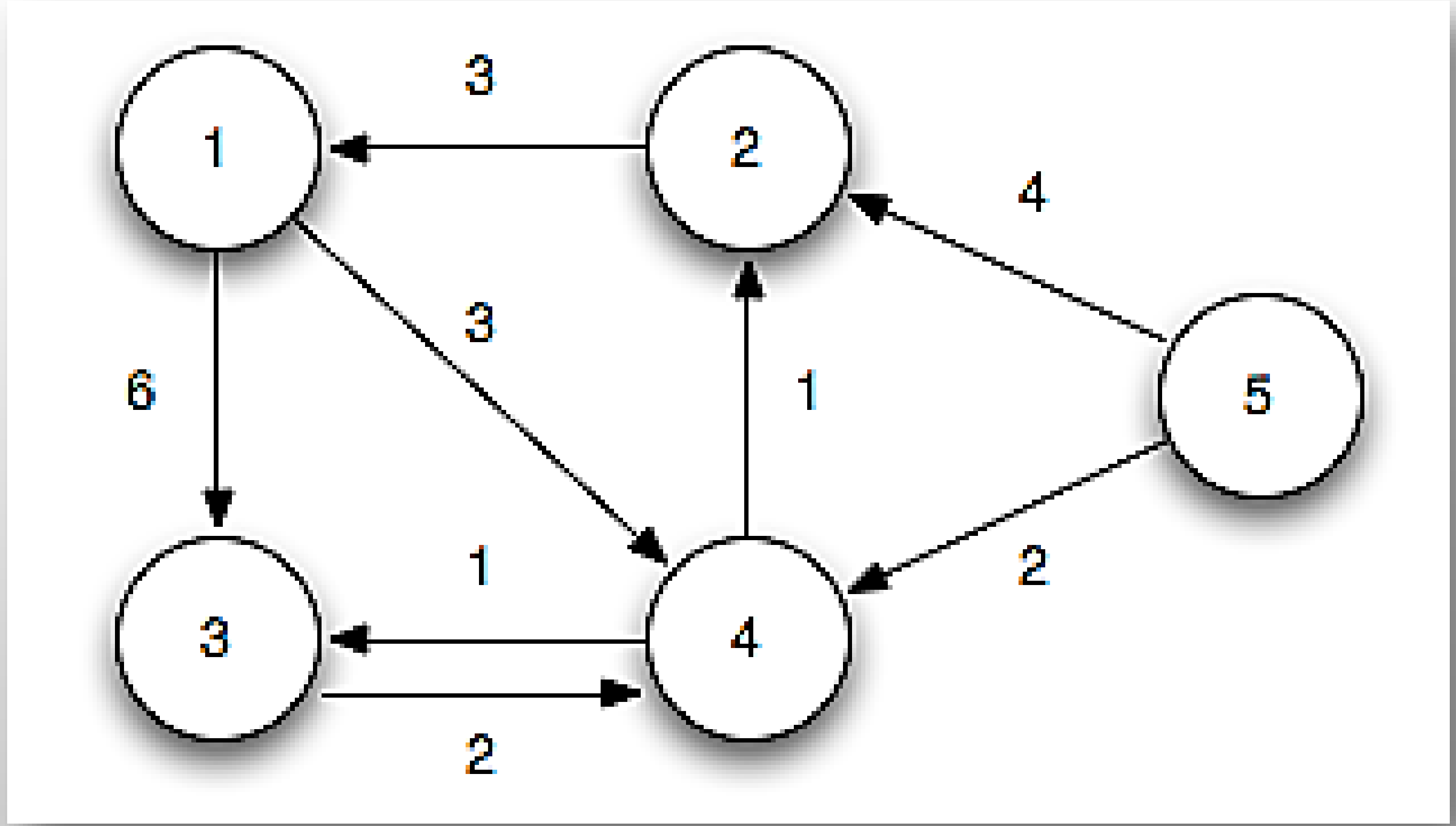
Algoritma Adımları

- 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

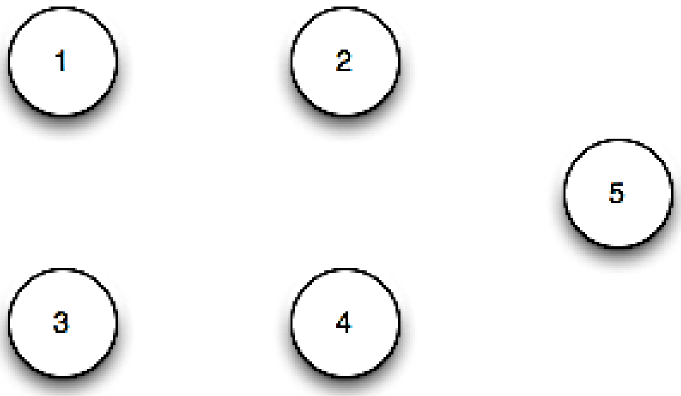
- Floyd-Warshall Algoritması'nın karmaşıklığı $O(V^3)$ şeklindedir.
- V düğüm sayısını temsil eder.
- 3 adet iç içe for döngüsü kullanılır.
    - ilk döngü tüm ara düğümleri gezer
    - ikinci döngü tüm kaynak düğümleri gezer.
    - son döngü tüm hedef düğümleri gezer.

Örnek



İlklendirme Aşaması

- (k = 0)
- kenar ağırlıklarına göre D matrisi doldurulur.



D

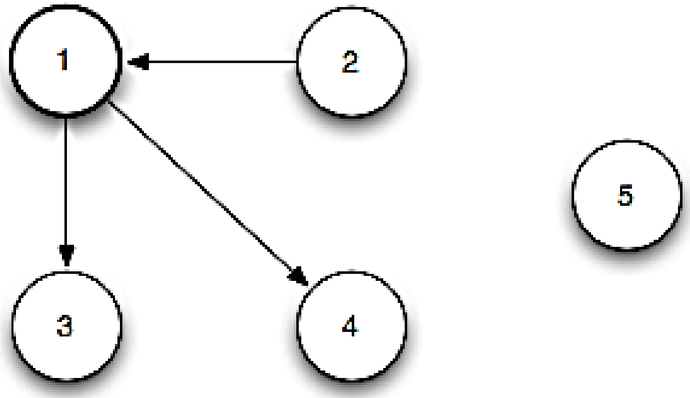
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	∞	∞	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

Π

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

Iteration 1

- (k = 1)
- Düğüm 1 üzerinden $2 \leadsto 3$ ve $2 \leadsto 4$ kısa yolları bulundu.



D

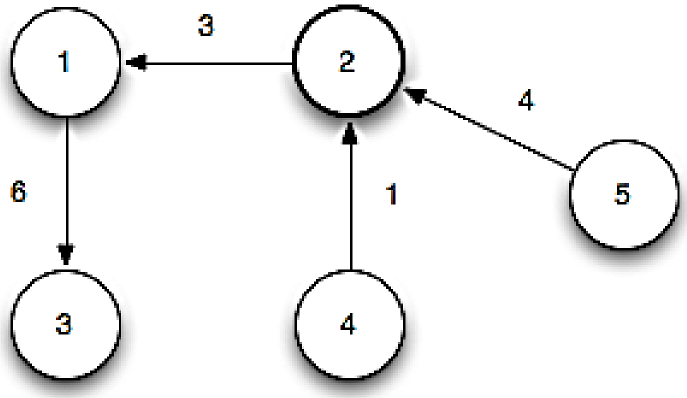
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

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)
- Düğüm 2 üzerinden 4 ↝ 1, 5 ↝ 1, ve 5 ↝ 3 kısa yolları bulundu.



D

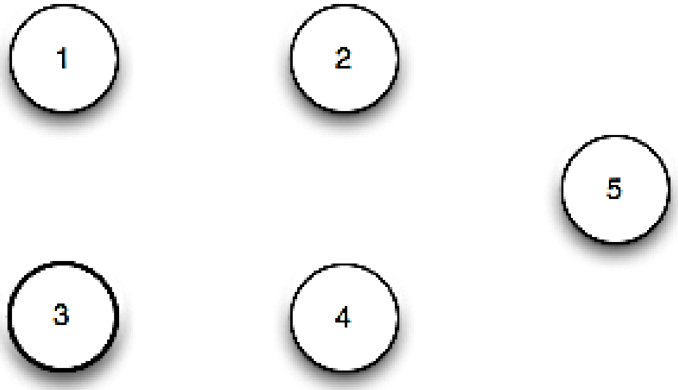
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	**4**	1	1	0	∞
5	**7**	4	**13**	2	0

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)
- Düğüm 3 üzerinden kısa yol bulunamadı.



D

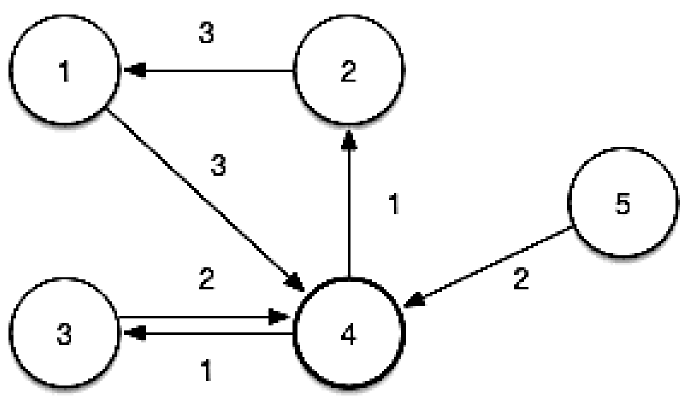
	1	2	3	4	5
1	0	∞	6	3	∞
2	3	0	9	6	∞
3	∞	∞	0	2	∞
4	4	1	1	0	∞
5	7	4	13	2	0

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)
- Düğüm 4 üzerinden 1 ↝ 2, 1 ↝ 3, 2 ↝ 3, 3 ↝ 1, 3 ↝ 2, 5 ↝ 1, 5 ↝ 2, 5 ↝ 3, ve 5 ↝ 4 kısa yolları bulundu.



D

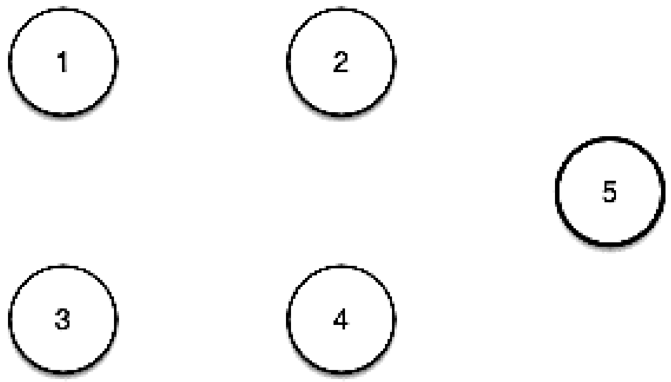
	1	2	3	4	5
1	0	4	4	3	∞
2	3	0	7	6	∞
3	6	3	0	2	∞
4	4	1	1	0	∞
5	6	3	3	2	0

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)
- Düğüm 5 üzerinden kısa yol bulunamadı.



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	/

Son Durum

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.

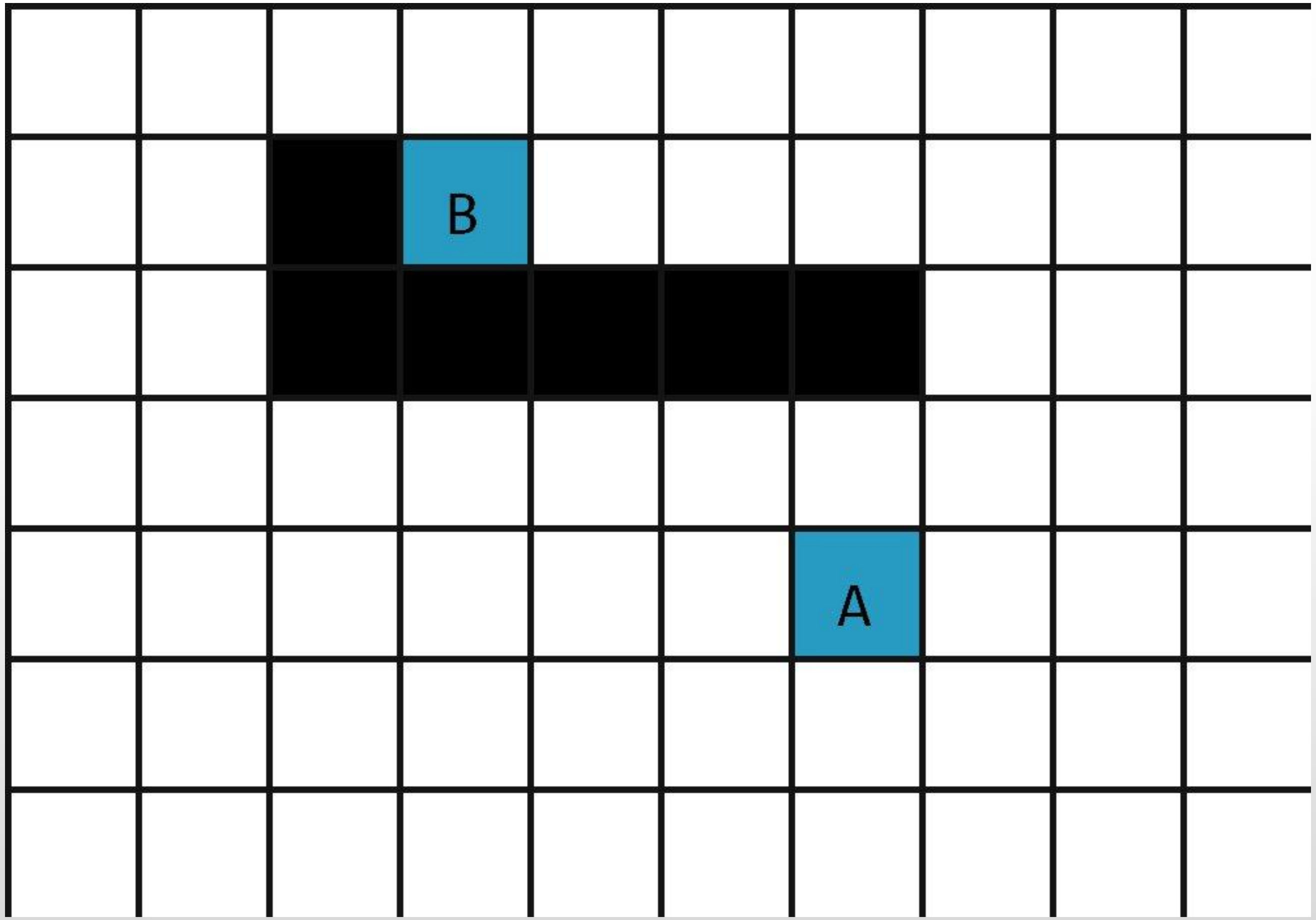
Algoritma Adımları

- 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.

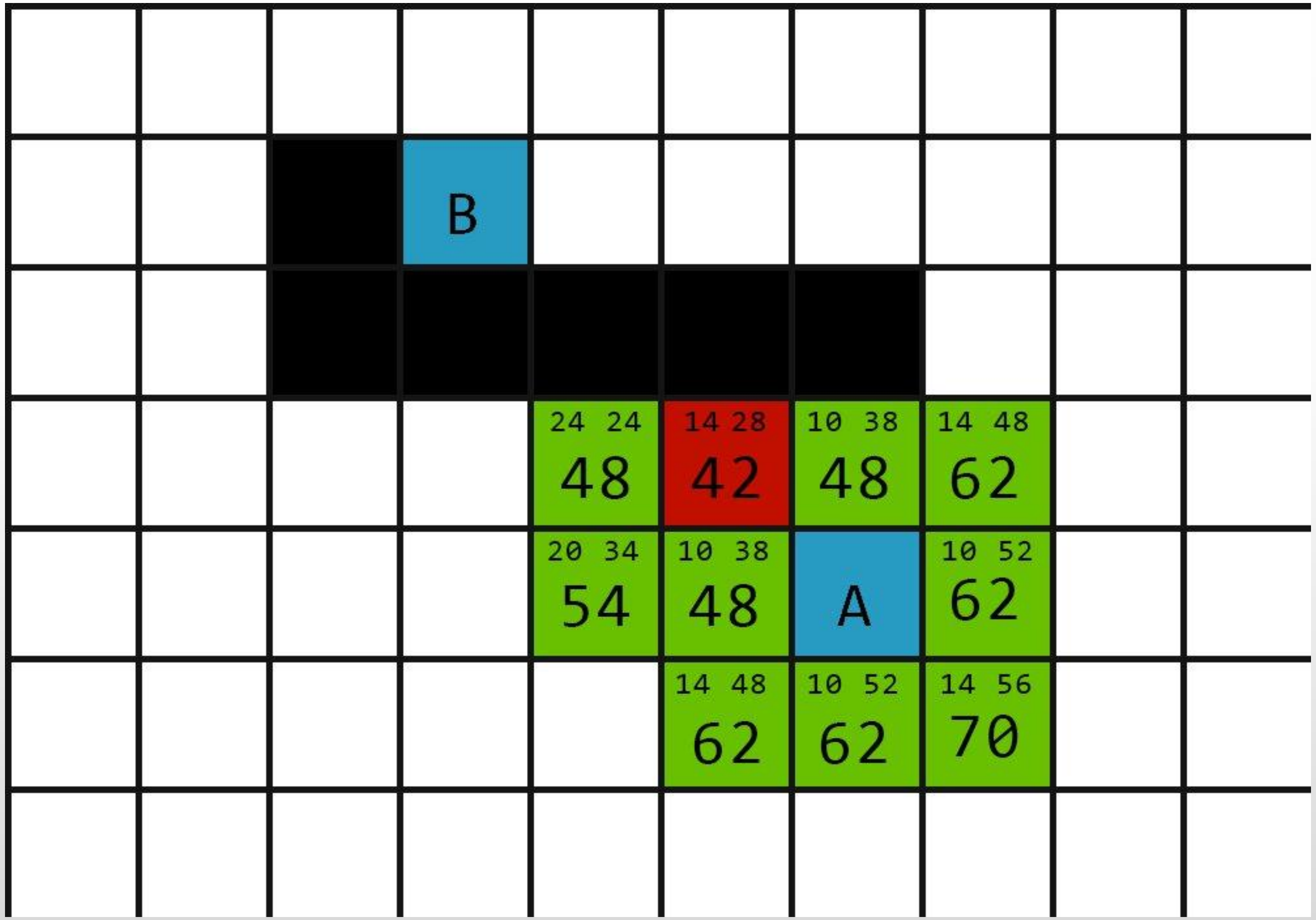
Algoritma Karmaşıklığı

- 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

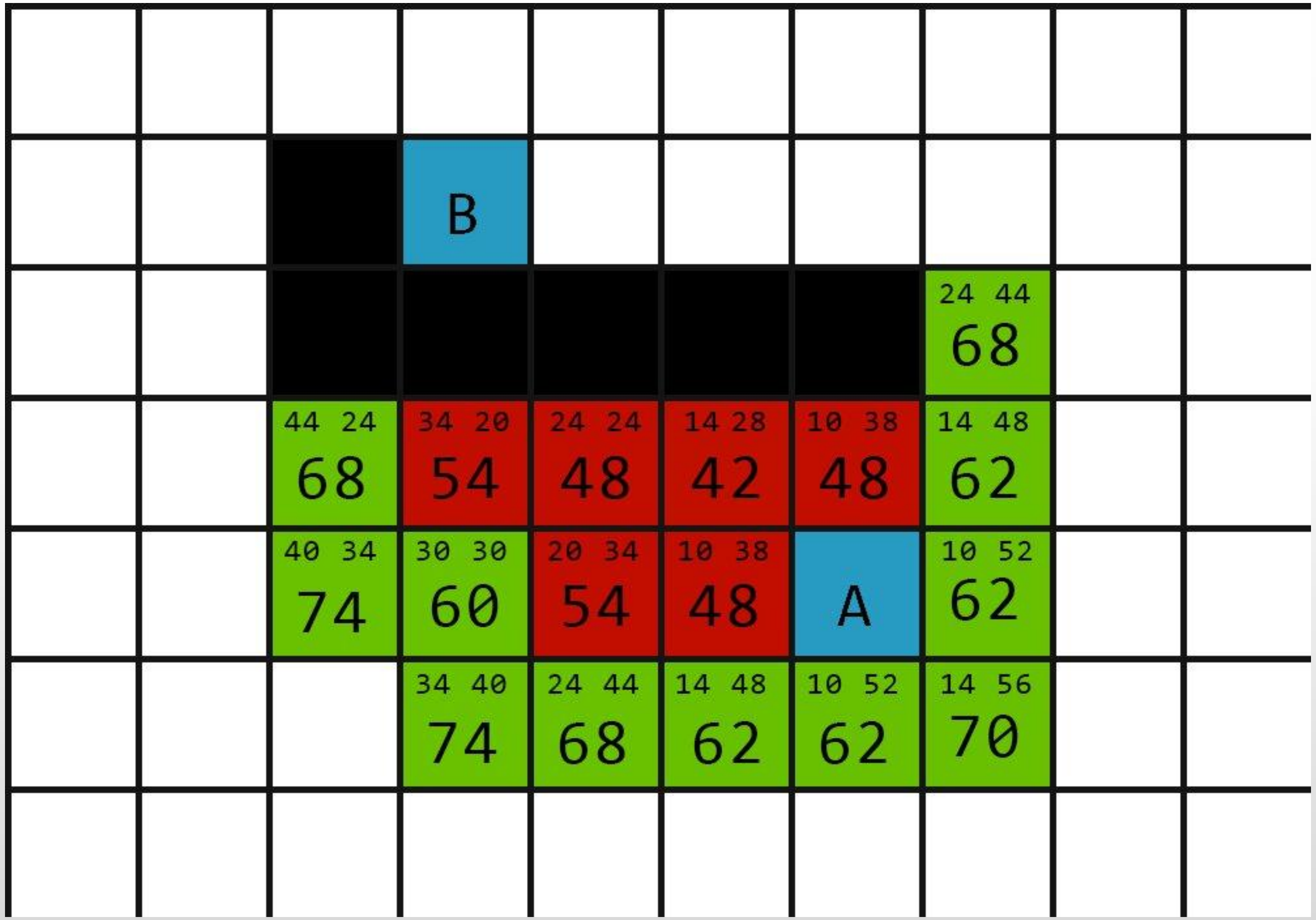


A Star



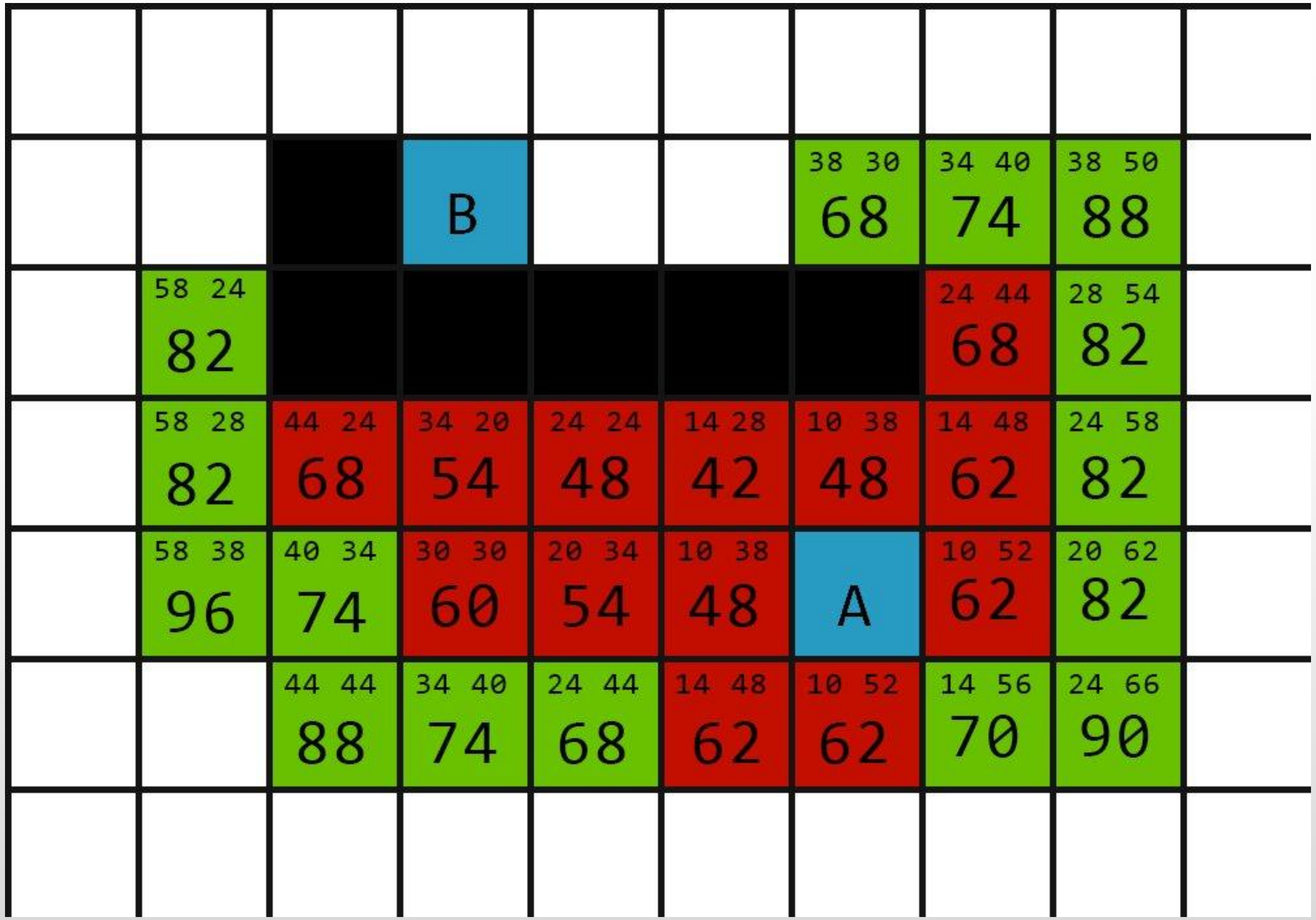
B
24 24
48
14 28
42
10 38
48
14 48
62
20 34
54
10 38
48
A
10 52
62
14 48
62
10 52
62
14 56
70

A Star

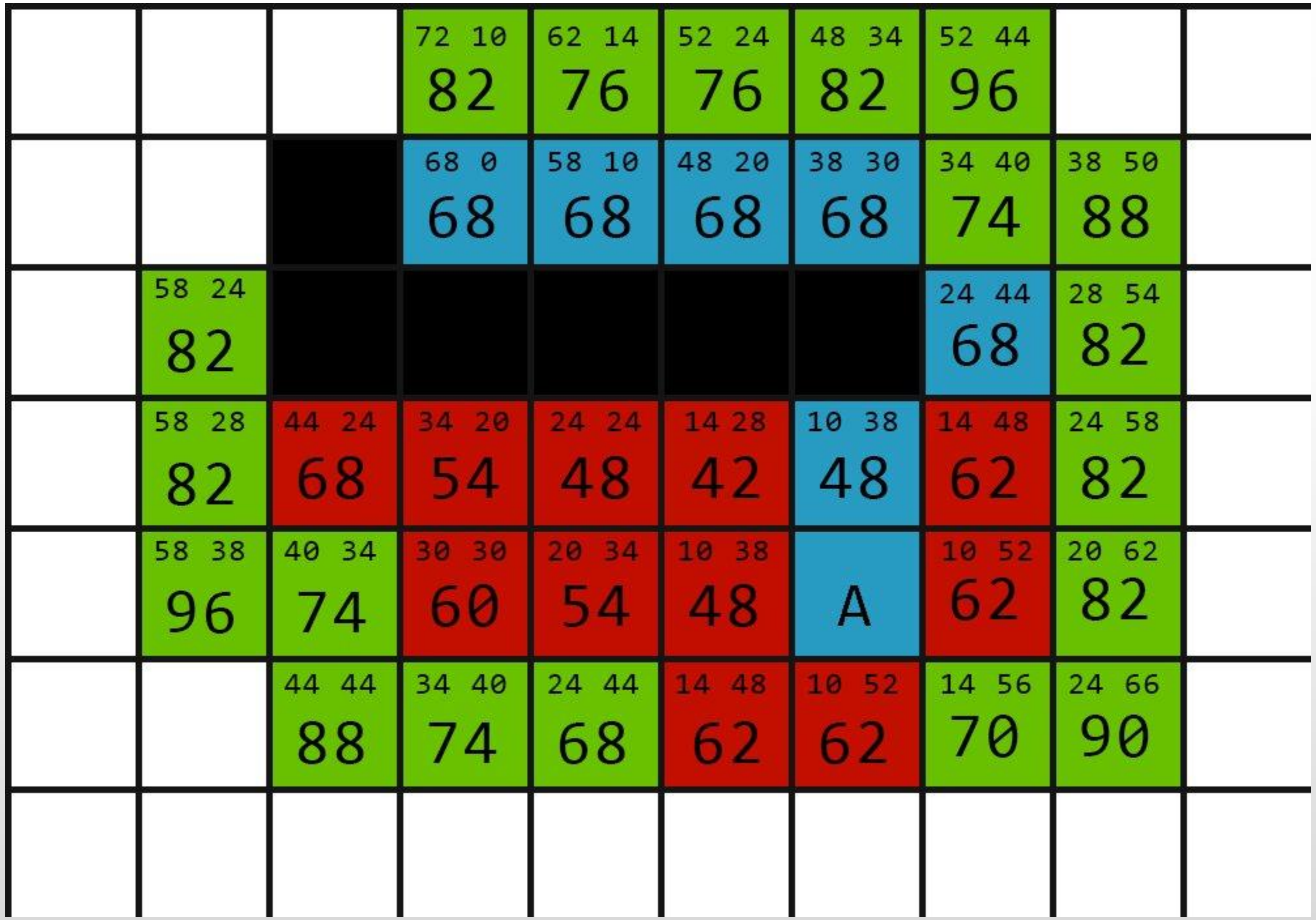


B
24 44
68
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
40 34
74
30 30
60
20 34
54
10 38
48
A
10 52
62
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70

A Star


B
38 30 68
34 40 74
38 50 88
58 24 82
24 44 68
28 54 82
58 28 82
44 24 68
34 20 54
24 24 48
14 28 42
10 38 48
14 48 62
24 58 82
58 38 96
40 34 74
30 30 60
20 34 54
10 38 48
A
10 52 62
20 62 82
44 44 88
34 40 74
24 44 68
14 48 62
10 52 62
14 56 70
24 66 90

A Star



72 10 82
62 14 76
52 24 76
48 34 82
52 44 96
68 0 68
58 10 68
48 20 68
38 30 68
34 40 74
38 50 88
58 24 82
24 44 68
28 54 82
58 28 82
44 24 68
34 20 54
24 24 48
14 28 42
10 38 48
14 48 62
24 58 82
58 38 96
40 34 74
30 30 60
20 34 54
10 38 48
A
10 52 62
20 62 82
44 44 88
34 40 74
24 44 68
14 48 62
10 52 62
14 56 70
24 66 90

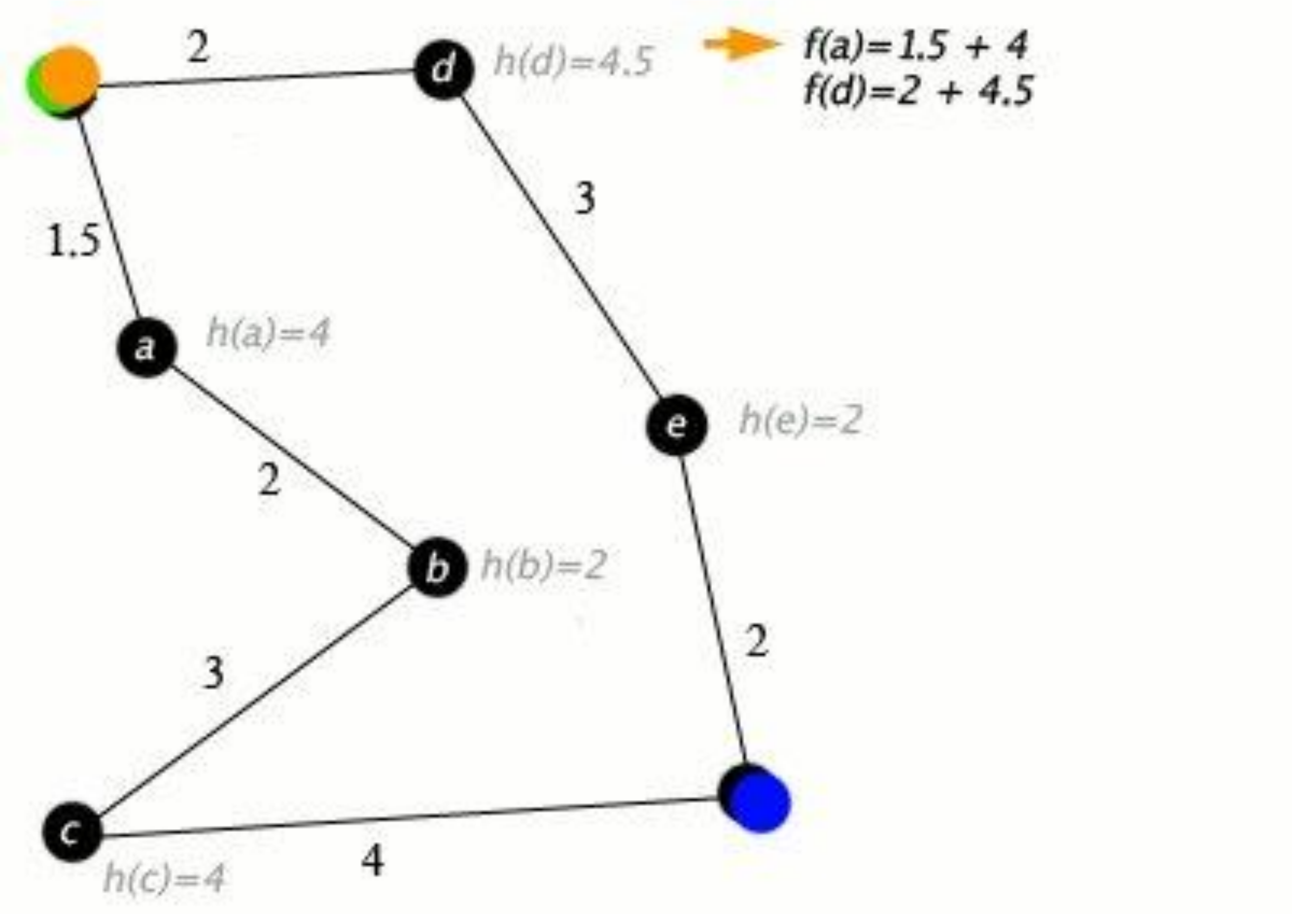
A* Arama Nasıl Çalışır?

- Her düğümün başlangıç düğümünden ulaşım maliyetini ("g-maliyet") ve mevcut düğümden hedef düğüme tahmini bir ulaşım maliyetini ("h-maliyet" veya sezgisel) dikkate alır.
- Sezgisel tahminlere göre hedefe yakın görünen düğümleri önceliklendirir.

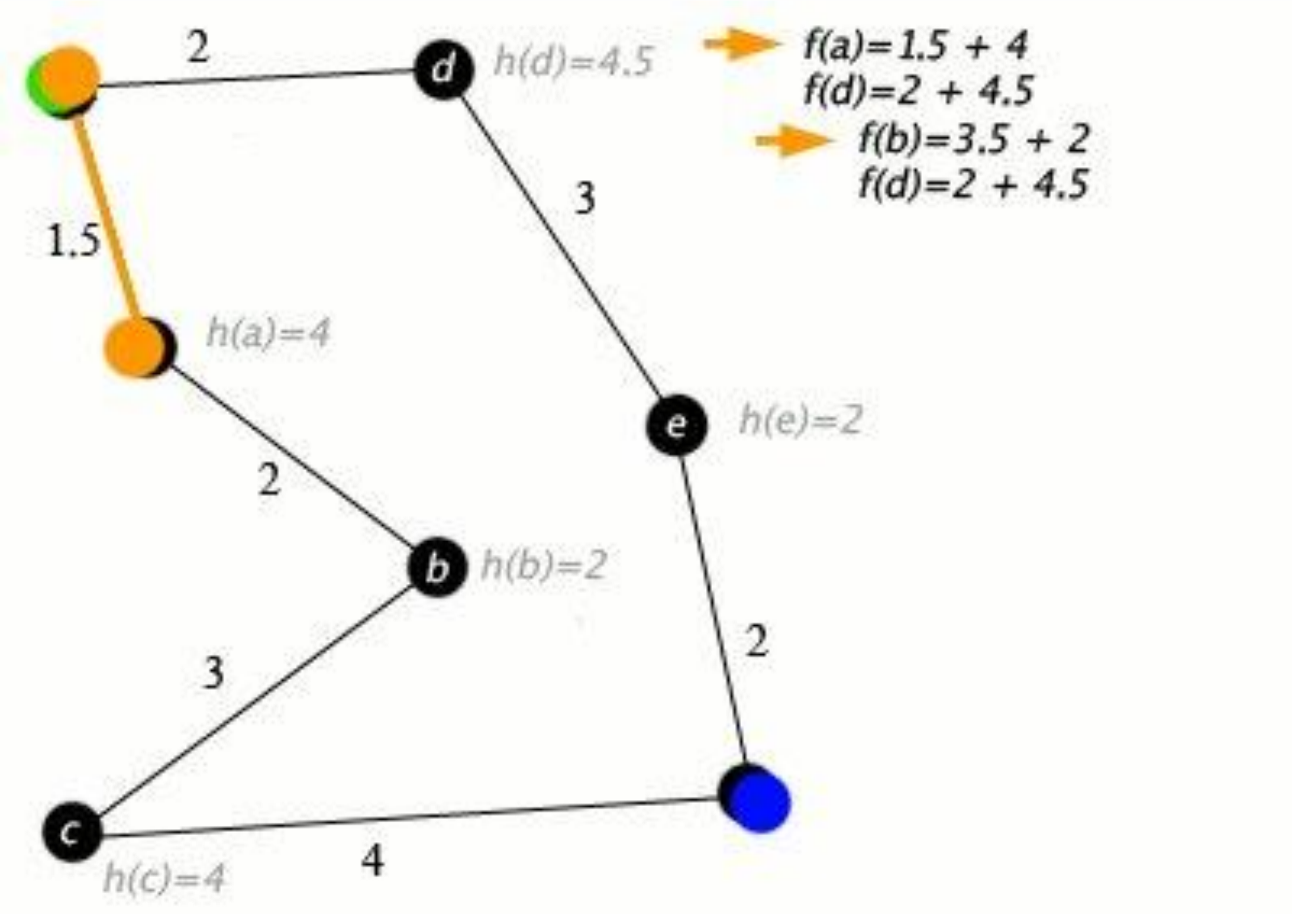
A* Arama

- Başlangıç düğümünü seç ve açık düğüm listesine ekle.
- Listeden en düşük f() + g() maliyetine sahip düğümü seç ve genişlet.
- Genişletilen düğüm,
    - hedef düğüm ise, çözüm bulundu.
    - değilse, hala genişletilecek düğümler var.
- Her bir sonraki düğüm için g ve f maliyetlerini güncelle, listeye ekle.
- Tekrar 2. adıma dön.

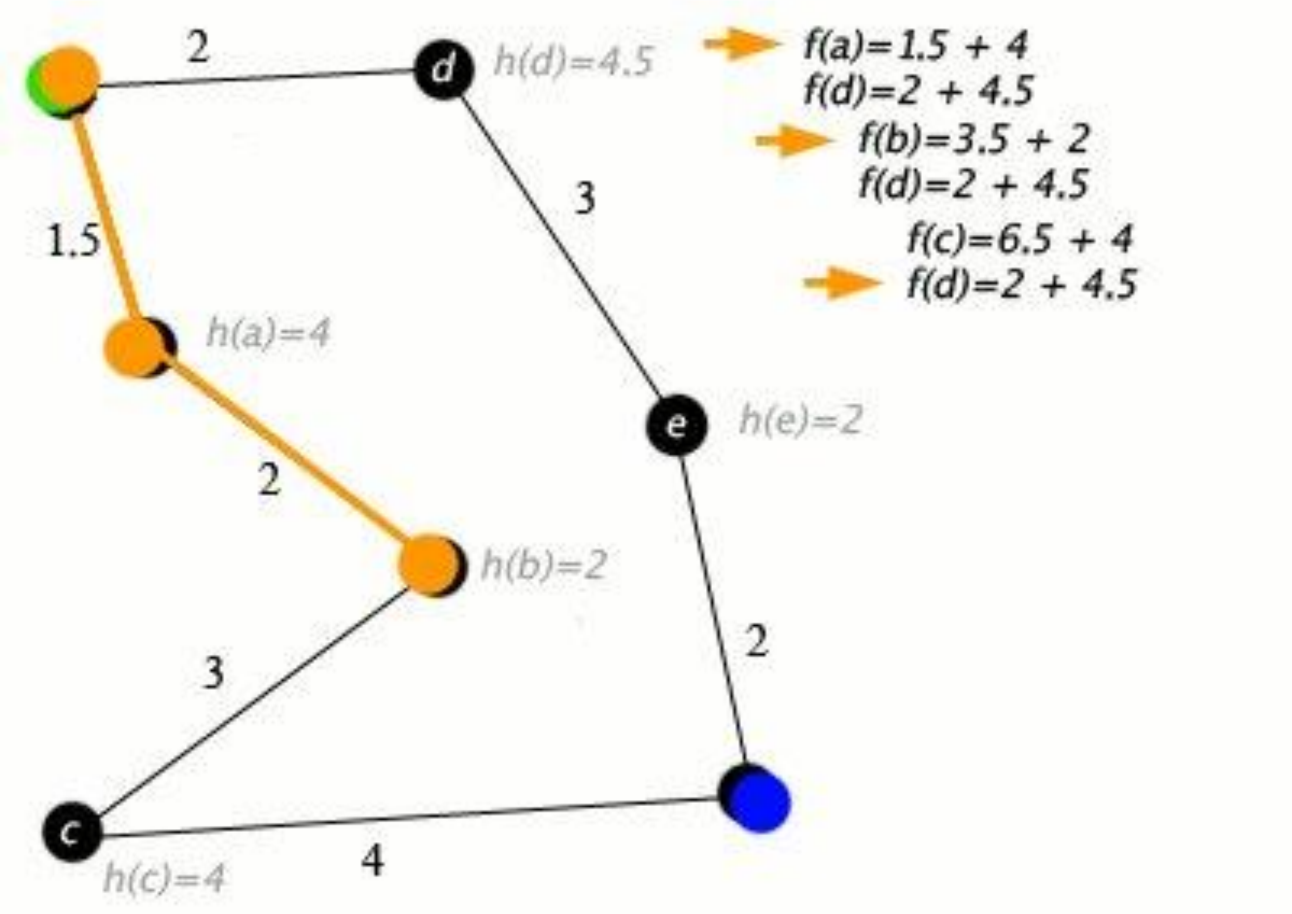
A Star Search



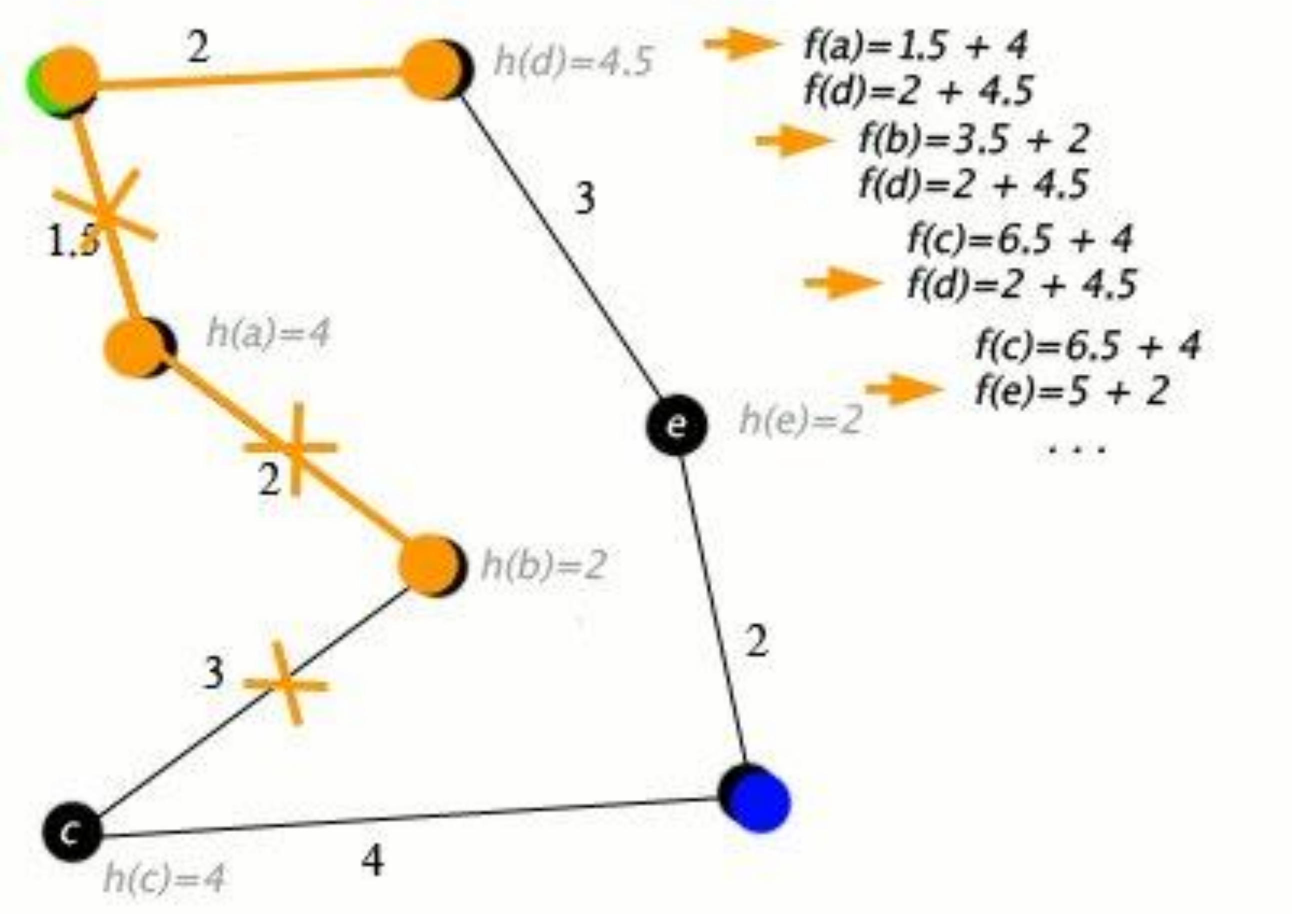
A Star Search



A Star Search



A Star Search



SON