DİJSKTRA ALGORİTMASI

Atakan AVCILAR (2211502060)
Çağrı DEMİRKIRAN (2211502023)
Ebrar BULUT (2211502059)

İÇERİK

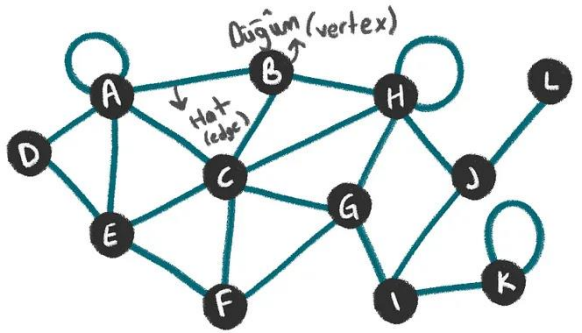
- ✓Graf Nedir?
- ✓Dijsktra Algoritması Nedir?
- ✓Dijsktra Algoritması Örnekleri

GRAF NEDİR?

Bir grafı kısaca, düğüm(vertex) adı verilen noktalar ve bu noktaların arasındaki hat(edge) adı verilen bağlantılardan oluşan topluluk olarak tanımlayabiliriz. Grafların birçok çeşidi olsa da şimdilik şu üçünü anlamamız bizim için yeterli olacaktır:

1.Yönsüz Graf (Undirected Graph)

Bu graf çeşidinde düğümler arasındaki hatların yönü yoktur. Yönsüz ilişkiye Facebook'daki arkadaşlık ilişkilerini örnek verebiliriz. A kişisi B kişisi ile arkadaş olduğunda B kişisi de A kişisi ile arkadaş olmuş olur.



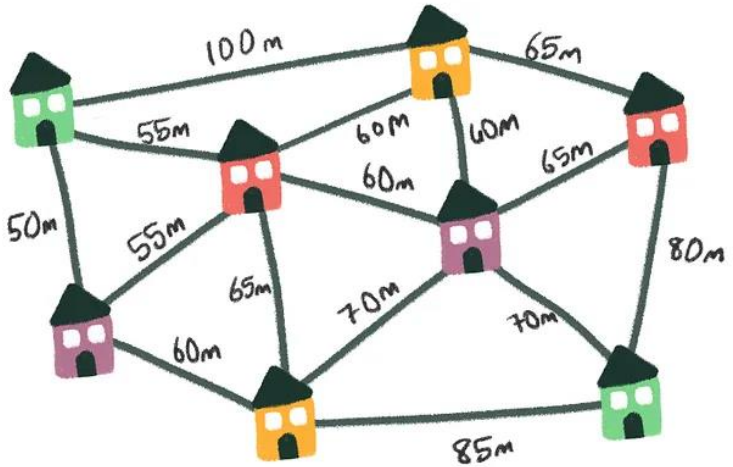
GRAF NEDİR?

2. Yönlü Graf (Directed Graf)

Bu graf çeşidinde düğümler arası yöne dayalı bir ilişki vardır. A noktası B'ye bağlı diye B de ona bağlıdır anlamını çıkaramayız. Buna Twitter'daki takipleşme ilişkisini örnek gösterebiliriz.

3. Ağırlıklı Graf (Weighted Graph)

Bu graf çeşidinde hatların bir değeri vardır. Hatlar maliyet, uzunluk, zaman gibi özelliklerine göre ağırlıklandırılır. Buna örnek olarak bir mahalle krokisi çizebilir ve yol uzunluğuna göre hatları ağırlıklandırabiliriz.



DİJKSTRA ALGORİTMASI

Günümüzde bilgisayar bilimlerinin temel taşlarından biri olan graf teorisi, çeşitli problemleri modellemek ve çözmek için kullanılan güçlü bir araçtır. Graf teorisinin sunduğu en önemli problemlerden biri, iki nokta arasındaki en kısa yolun bulunmasıdır. Bu problem, Dijkstra algoritması ile etkili bir şekilde çözülebilir.

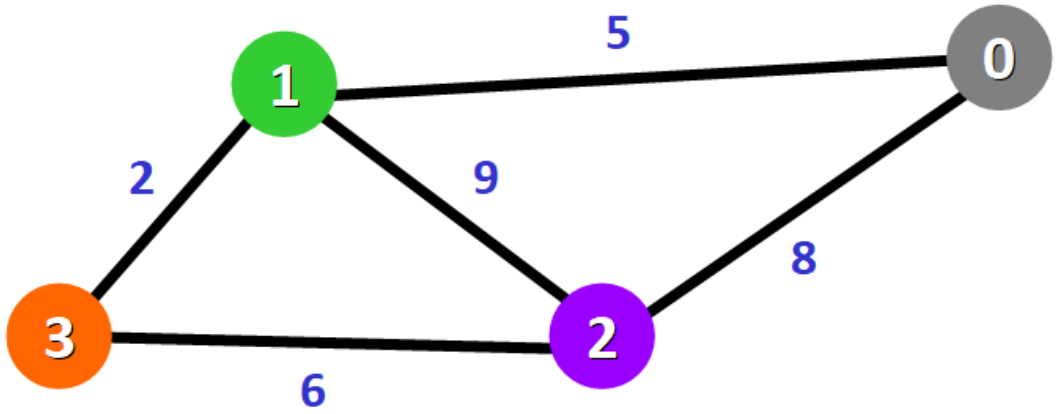
Çalışma Prensibi

Dijkstra algoritması, bir başlangıç düğümünden diğer tüm düğümlere olan en kısa yol mesafelerini bulmak için kullanılır. Temel olarak, algoritma her adımda henüz işlenmemiş düğümler arasından en kısa mesafeye sahip olanı seçer ve bu düğümü işler. Seçilen düğümün komşularının mesafelerini günceller ve ardından bir sonraki adıma geçer. Bu işlem, hedef düğüme ulaşılıncaya kadar veya tüm düğümler işlenene kadar devam eder.

DİJKSTRA ALGORİTMASI

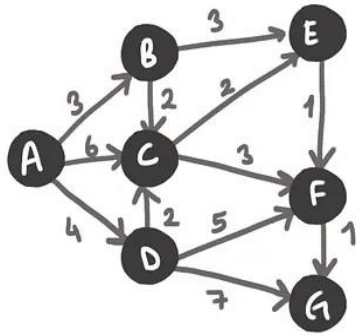
Terminoloji

- **Düğüm (Node):** Bir ağın bir noktasını temsil eder.
- **Kenar (Edge):** İki düğüm arasındaki bağı gösterir.
- **Ağırlık (Weight):** İki düğüm arasındaki mesafeyi veya maliyeti belirtir.
- **En Kısa Yol (Shortest Path):** İki düğüm arasındaki en küçük toplam ağırlığa sahip yol.



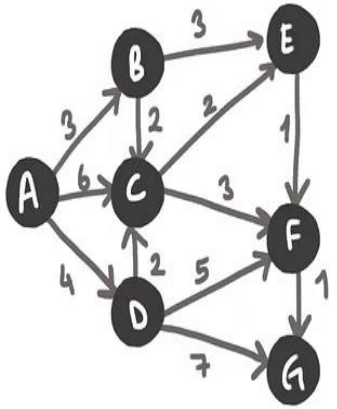
DİJKSTRA ALGORİTMA ÇÖZÜMÜ

Algoritmanın adımlarına bakarsak ilk olarak yapacağımız şey, A düğümü hariç düğümleri bir matrisin sütunlarına dizmek olacaktır.



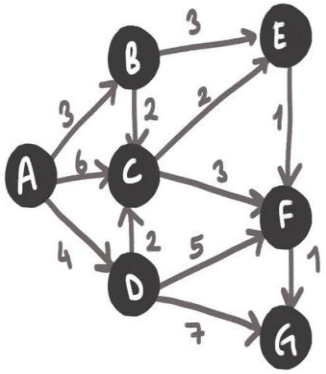
	B	C	D	E	F	G

B'ye 3, C'ye 6 ve D'ye 4 birim uzaklıkta. E, F ve G ile direkt bağlantısı olmadığından şimdilik onlara sonsuz uzaklıkta olduğunu düşünerek sonsuz yazıyoruz.



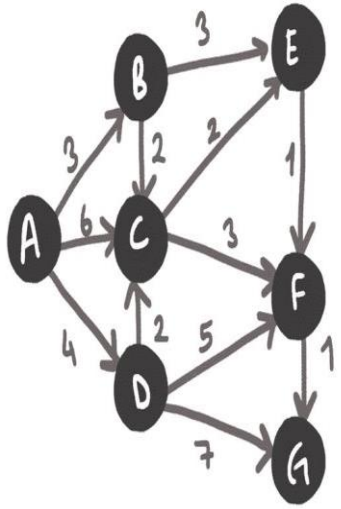
	B	C	D	E	F	G
A	3	6	4	∞	∞	∞

İlk değerden daha kısa bir mesafe bulduğumuzda matrisi güncelliyoruz. Burada dikkat etmemiz gereken çok önemli bir şey var. Yeni kaynağa olan uzaklıkları hesaplarken ilk kaynağı unutmamalıyız.

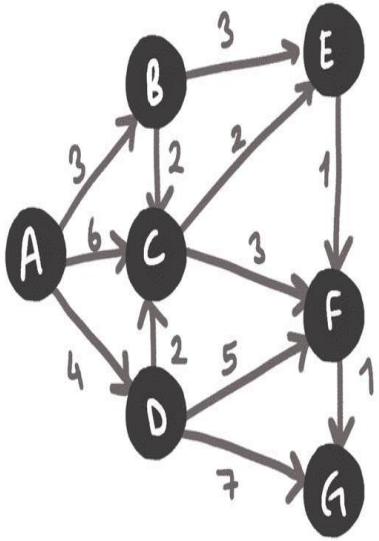


	B	C	D	E	F	G
A	3	6	4	∞	∞	∞

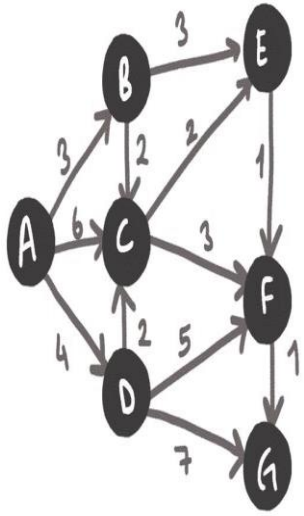
DİJKSTRA ALGORİTMA ÇÖZÜMÜ



	B	C	D	E	F	G
A	3	6	4	∞	∞	∞
B		5	4	6	∞	∞

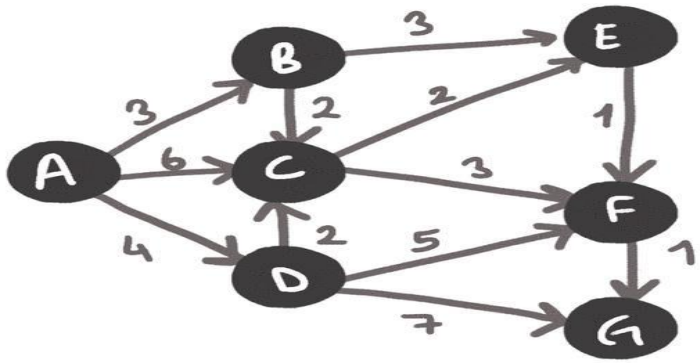


	B	C	D	E	F	G
A	3	6	4	∞	∞	∞
B		5	4	6	∞	∞
D		~~6~~ 5		6	9	11

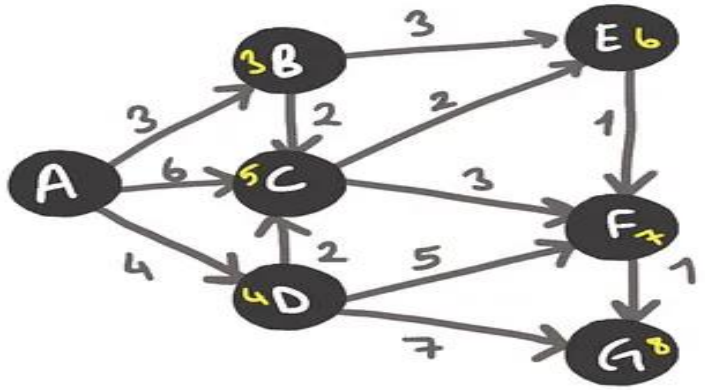


	B	C	D	E	F	G
A	3	6	4	∞	∞	∞
B		5	4	6	∞	∞
D		5		6	9	11
C				~~7~~ 6	8	11

DİJKSTRA ALGORİTMA ÇÖZÜMÜ



	B	C	D	E	F	G
A	3	6	4	∞	∞	∞
B		5	4	6	∞	∞
D		5		6	9	11
C				6	8	11
E					7	11



	B	C	D	E	F	G
A	3	6	4	∞	∞	∞
B		5	4	6	∞	∞
D		5		6	9	11
C				6	8	11
E					7	11
F						8

DİJKSTRA ALGORİTMASININ ZAYIF YÖNÜ

Algoritma ne yazık ki eksi (-) değer taşıyan bir kenar bulunması halinde başarılı çalışmaz. Bunun sebebi eksi (-) değerdeki kenarın sürekli olarak mevcut durumdan daha iyi bir sonuç üretmesi ve algoritmanın hiçbir zaman için kararlı hale gelememesidir.

DİJKSTRA ALGORİTMASI PSEUDO KOD

Function dijkstra(graph, source):

//**Değişkenleri başlatıyoruz**

V = graph.length //**Düğüm sayısı**

dist[V] = {INF, INF, ..., INF} //**Kaynaktan her düğüme en kısa mesafeyi saklayan dizinin elemanları sonsuzda başlatıldı**

visited[V] = {false, false, ..., false} // **Ziyaret edilen düğümlerin takibini sağlayan dizinin elemanları false olarak başlatıldı.**

dist[source] = 0 // **Kaynağın kendisine uzaklığı sıfır olduğu için sıfır alıyoruz**

DİJKSTRA ALGORİTMASI PSEUDO KOD

// En kısa yolları bulmak için ana döngü

for count = 0 to V - 1:

//Henüz işlenmemiş düğümler kümesinden minimum uzaklıkta olan köşeyi seçin

u = minDistance(dist, visited)

//Seçilen köşeyi ziyaret edildi olarak işaretle

visited[u] = true

//Seçilen köşenin komşu köşelerinin mesafelerini güncelle

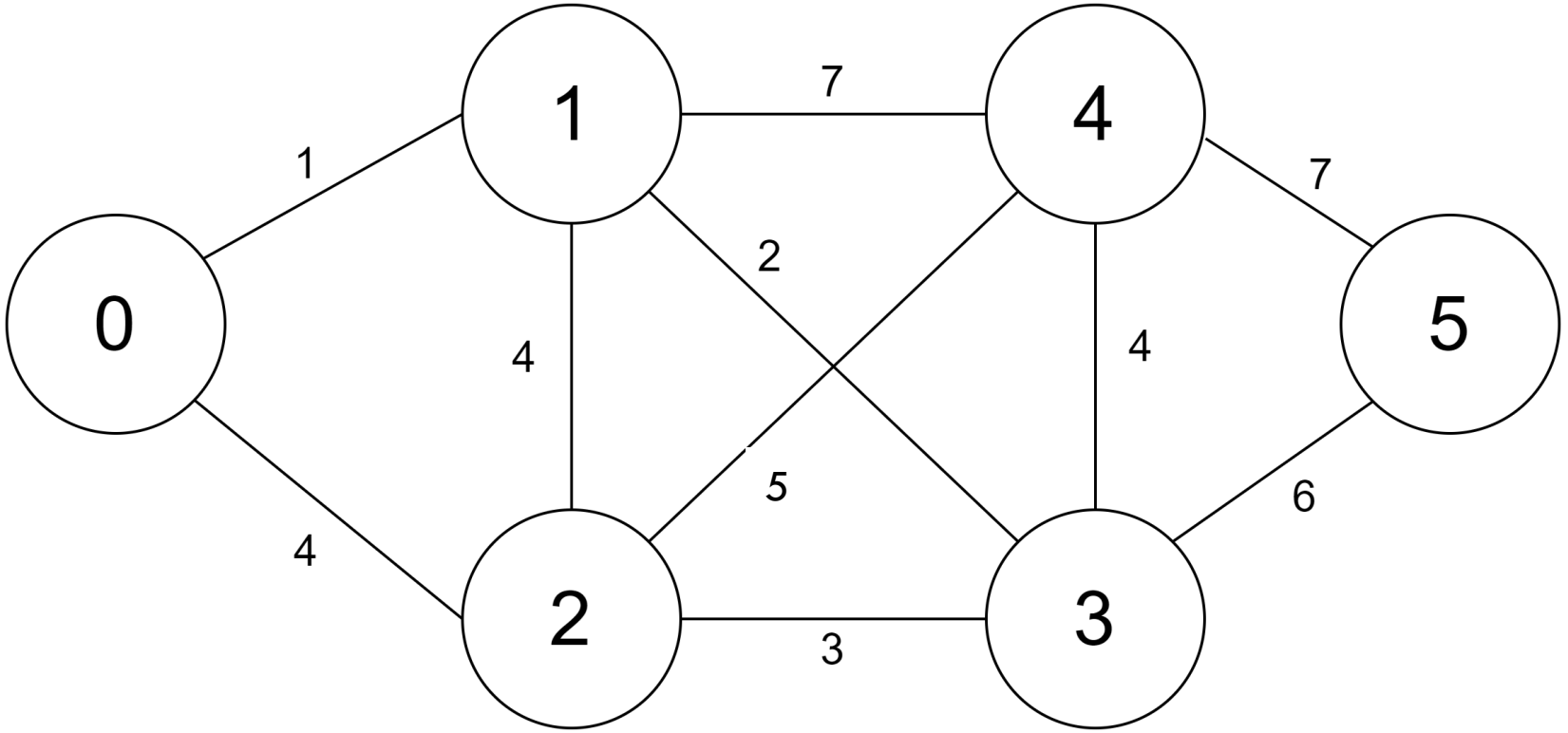
for v = 0 to V - 1: if not visited[v] and graph[u][v] != 0 and dist[u] != INF and dist[u] + graph[u][v] < dist[v]:

dist[v] = dist[u] + graph[u][v]

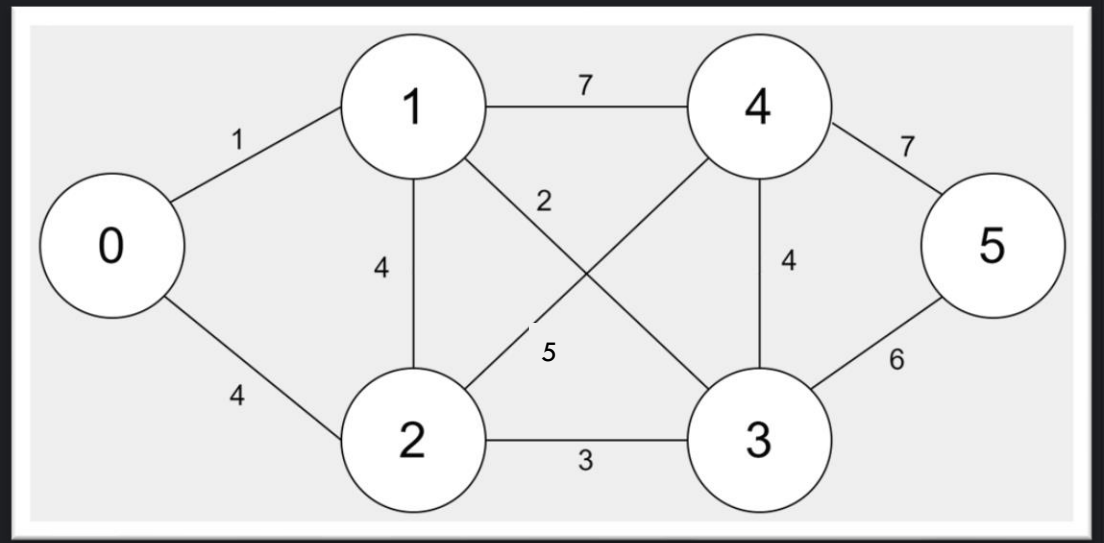
DİJKSTRA ALGORİTMASI PSEUDO KOD

```
// En kısa mesafeleri yazdır

printSolution(dist)

Function minDistance(dist, visited): min = INF //Sonsuza olan minimum mesafeyi başlat

minIndex = -1 // minIndex'i -1 olarak başlat

 for v = 0 to length(dist) - 1:

        if not visited[v] and dist[v] < min:

            min = dist[v]

            minIndex = v

    return minIndex

Function printSolution(dist):

// Kaynaktan her düğüme en kısa mesafeleri yazdırın

    for i = 0 to length(dist) - 1:

        print i + " \t\t " + dist[i]
```

DİJKSTRA ALGORİTMASI KOD ÖRNEGİ

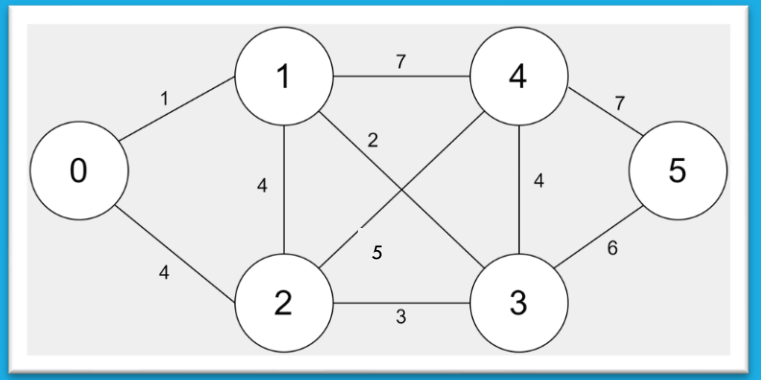


```java
public void dijsktra(int[][] graph, int src) {
    int[] dist = new int[graph.length];// En kısa mesafe dizisi
    boolean[] visited = new boolean[graph.length]; // Ziyaret edilip edilmediğini gösteren bir dizi
    // başlangıç için tüm mesafleri sonsuz yapıyoruz
    for (int i = 0; i < graph.length; i++) {
        dist[i] = Integer.MAX_VALUE;
        visited[i] = false;
    }
    //başlangıç düğümü için mesafeyi 0 ayarladık
    dist[src] = 0;
    //tüm düğümleri ziyaret edene kadar
    for (int count = 0; count < graph.length; count++) {
        // en kısa mesafeli düğümü bul
        int em = minDistance(dist, visited);
        //bu düğümü ziyaret ettiğimizi işaretle
        visited[em] = true;
        //şimdiye kadar ziyaret edilmemiş olan ve em'den geçen düğümlere olan mesafeti günncele
        for (int j = 0; j < graph.length; j++) {
            if (visited[j] == false && graph[em][j] != 0 && dist[em] != Integer.MAX_VALUE && dist[em] + graph[em][j] < dist[j]) {
                dist[j] = dist[em] + graph[em][j];
            }
        }
    }
    printGraph(dist, src);
}
```



```java
public int minDistance(int[] dist, boolean[] visited) {
    int min = Integer.MAX_VALUE; // Başlangıçta minimum mesafeyi integerin max değeri kabul ediyoruz.
    int index = -1; // Minimum mesafeli düğümün indisini -1 olarak başlatıyoruz
    for (int i = 0; i < dist.length; i++) {
        if (visited[i] == false && dist[i] <= min) { // Daha önce ziyaret edilmemiş ve daha küçük bir mesafeye sahipse
            min = dist[i]; // Minimum mesafeyi güncelliyoruz
            index = i; // Minimum mesafeli düğümün indisini güncelliyoruz
        }
    }
    return index;
}
```

DİJKSTRA ALGORİTMASI KOD ÖRNEGİ

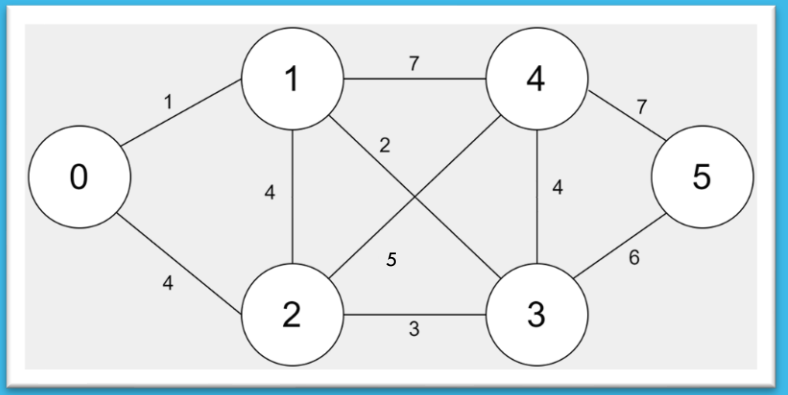


```java
public void printGraph(int[] dist, int src) {
    System.out.println("Düğüm \t\t En Kısa Mesafe");
    for (int i = 0; i < dist.length; i++) {
        System.out.println(src + " -> " + i + " \t\t " + dist[i]);
    }
}
```

Düğüm	En Kısa Mesafe
0 -> 0	0
0 -> 1	1
0 -> 2	4
0 -> 3	3
0 -> 4	7
0 -> 5	9

```java
public static void main(String[] args) {
    int[][] graph = {
            {0, 1, 4, 0, 0, 0},
            {1, 0, 4, 2, 7, 0},
            {4, 4, 0, 3, 5, 0},
            {0, 2, 3, 0, 4, 6},
            {0, 7, 5, 4, 0, 7},
            {0, 0, 0, 6, 7, 0}
    };
    dijkstraAlgorithm da = new dijkstraAlgorithm();
    da.dijsktra(graph, src: 0);
}
```

DİJKSTRA ALGORİTMASI KOD ÖRNEGİ



0	1	2	3	4	5
~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~1~~
	0	0	1	1	3
				3	

0	1	2	3	4	5
~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	F
T	T	T	T	T	

0	1	2	3	4	5
0	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	1	4	3	~~8~~	9
				7	

count = 0

i = 0

min = 0

index = 0

em = 0

j = 1, 2

dist[1] = 1

dist[2] = 4

count = 1

i = 1

min = 1

index = 1

em = 1

j = 3, 4

dist[3] = 1+2 = 3

dist[4] = 1+7 = 8

count = 2

i = 2

min = 4

index = 2

em = 2

j = 4

dist[4] = 4+5 = 9 ×

count = 3

i = 3

min = 3

index = 3

em = 3

j = 4, 5

dist[4] = 3+4 = 7 ✓

dist[5] = 3+6 = 9

count = 4

i = 4

min = 7

index = 4

em = 4

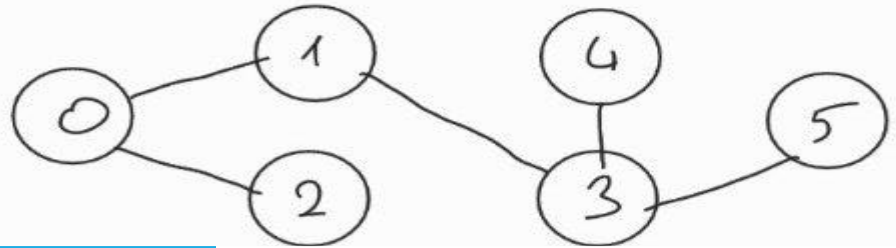
j = 5

dist[5] = 7+7 = 14 ×

0	1	2	3	4	5
-1	0	0	1	3	3

0	1	2	3	4	5
T	T	T	T	T	T

0	1	2	3	4	5
0	1	4	3	7	9



KAYNAKÇA

- https://medium.com/t%C3%BCrkiye/graf-teorisi-4-en-k%C4%B1sa-yol-problemi-322a648c864e
- https://medium.com/@alifurkangokce/dijkstra-algoritmas%C4%B1-en-k%C4%B1sa-yol-bulman%C4%B1n-anahtar%C4%B1-7fc022a3c028#:~:text=Dijkstra%20algoritmas%C4%B1%2C%20bir%20ba%C5%9Flang%C4%B1%C3%A7%20d%C3%BC%C4%9F%C3%BCm%C3%BCnden,se%C3%A7er%20ve%20bu%20d%C3%BC%C4%9F%C3%BCm%C3%BC%20i%C5%9Fler
- https://medium.com/t%C3%BCrkiye/graf-teorisine-giri%C5%9F-c90cbdf9564c
- https://bilgisayarkavramlari.com/2010/05/13/dijkstra-algoritmasi-2/
- https://www.youtube.com/watch?v=t2d-XYuPfg0
- https://www.youtube.com/watch?v=MeiKA0uai0c&t=1513s
- https://www.youtube.com/watch?v=SZXXnB7vSm4
- https://chat.openai.com/