****

**MESLEK YÜKSEKOKULU**

**Bölüm Adı:** Bilgisayar Teknolojileri

**Program Adı:** Bilgisayar Programcılığı

**Ders Adı:**Yapay Zekâ

**Proje Adı:** En kısa yol bulunma uygulaması

**Hazırlayan**

**Adı - Soyadı:** Canberk Ayhan

**Okul Numarası:** 18MY03032

**Öğretim Görevlisi:** Nilgün İNCEREİS

# 

İçindekiler

[Yapay zeka nedir ? 2](#_Toc38555348)

[Evet ama zeka nedir ? 2](#_Toc38555349)

[İnsan zekasıyla ilişkilendirmeye bağlı olmayan sağlam bir zeka tanımı yok mu? 2](#_Toc38555350)

[Yapay zeka araştırması ne zaman başladı? 2](#_Toc38555351)

[En kısa yol algoritmaları 3](#_Toc38555352)

[DIJKSTRA ve FLOYD algoritmaları 3](#_Toc38555353)

[FLOYD ALGORİTMASI 3](#_Toc38555354)

[Dijkstra Algoritması 7](#_Toc38555355)

[İşte algoritmanın açıklaması: 11](#_Toc38555356)

[Projem 12](#_Toc38555357)

# Yapay zeka nedir ?

Akıllı makineler, özellikle de akıllı bilgisayar programları yapma bilim ve mühendisliğidir. İnsan zekasını anlamak için bilgisayar kullanmadaki benzer görevle ilgilidir, ancak AI, biyolojik olarak gözlemlenebilir yöntemlerle sınırlamak zorunda değildir.

## Evet ama zeka nedir ?

Zeka, dünyadaki hedeflere ulaşma yeteneğinin hesaplayıcı bir parçasıdır. Değişen çeşitler ve zeka dereceleri insanlarda, birçok hayvanda ve bazı makinelerde görülür.

## İnsan zekasıyla ilişkilendirmeye bağlı olmayan sağlam bir zeka tanımı yok mu?

Henüz değil. Sorun şu ki, genel olarak ne tür hesaplama prosedürlerini akıllı olarak adlandırmak istediğimizi henüz belirleyemiyoruz. Zekanın bazı mekanizmalarını anlıyoruz, diğerlerini değil.

## Yapay zeka araştırması ne zaman başladı?

İkinci Dünya Savaşı'ndan sonra, birkaç kişi bağımsız olarak akıllı makineler üzerinde çalışmaya başladı. İngiliz matematikçi Alan Turing ilk olabilir. 1947'de bir konferans verdi. Ayrıca yapay zekanın en iyi inşaat makineleri yerine programlama bilgisayarlarıyla araştırılmasına karar veren ilk kişi olabilir. 1950'lerin sonunda, yapay zeka konusunda birçok araştırmacı vardı ve bunların çoğu çalışmalarını programlama bilgisayarlarına dayandırıyordu.

## En kısa yol algoritmaları

* En Kısa Yol probleminde, ulaştırma şebekesindeki **Kaynak** ile **Varış Noktası** arasındaki en kısa yol bulunmaya çalışılır. Amaç, şebeke üzerindeki iki düğüm noktası arasındaki en kısa uzunluğu ve rotayı bulmaktır.
* Mesela karayolları haritası üzerinde seçilen iki şehir arasındaki en kısa gidiş rotasını bulmak gibi. Günlük hayatta ve özellikle Mühendislik Uygulamalarında sıkça karşılaşılan bir problemdir.

## DIJKSTRA ve FLOYD algoritmaları

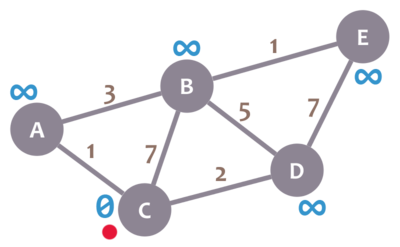
* Dijkstra algoritması belirtilen bir düğümden belirtilen başka bir düğüme gidebilmek için en kısa yolu ve rotayı bulmaya yarar.
* Floyd algoritması ise belirli bir graf için her hangi bir düğümden her hangi bir düğüme en kısa yolu ve rotayı bulmaya yarar.
* Floyd algoritmasını, dijkstra algoritmasının daha gelişmişi olduğunu düşüne biliriz.

# Dijkstra Algoritması

Dijkstra Algoritması, bir düğüm (hangisini seçerseniz) ile grafikteki diğer tüm düğümler arasındaki en kısa yolu hesaplamanıza olanak tanır. Bu sayfanın sonunda algoritmanın bir açıklamasını bulacaksınız, ancak algoritmayı açıklanmış bir örnekle inceleyelim! Grafikteki C düğümü ile diğer düğümler arasındaki en kısa yolu hesaplayalım:

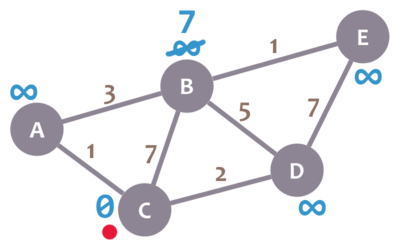
# graph

Algoritma yürütme sırasında, her düğümü C düğümüne (seçilen düğümümüz) minimum mesafesi ile işaretleriz. C düğümü için bu mesafe 0'dır. Geri kalan düğümler için hala minimum mesafeyi bilmediğimizden, sonsuzluk olmaya başlar (∞):

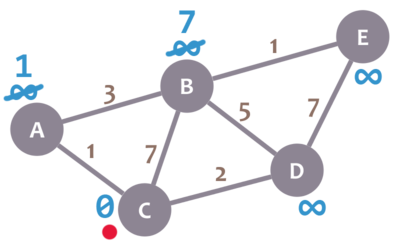


Ayrıca mevcut bir düğümümüz olacak. Başlangıçta, C (seçilen düğümümüz) olarak ayarladık. Görüntüde, mevcut düğümü kırmızı bir nokta ile işaretliyoruz.

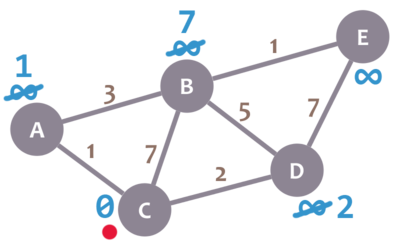
Şimdi, mevcut düğümümüzün (A, B ve D) komşularını belirli bir sırada kontrol etmiyoruz. B ile başlayalım. Geçerli düğümün minimum mesafesini (bu durumda, 0) mevcut düğümümüzü B'ye bağlayan kenarın ağırlığı ile ekliyoruz (bu durumda, 7) ve 0 + 7 = 7 elde ediyoruz Bu değeri minimum B (sonsuzluk) mesafesi ile karşılaştırıyoruz; en düşük değer, B'nin minimum mesafesi olarak kalan değerdir (bu durumda, 7 sonsuzdan daha azdır):



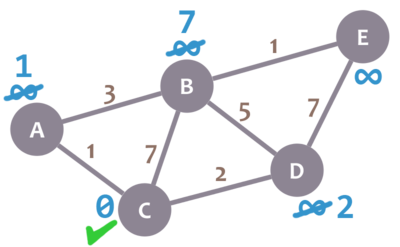
Çok uzak çok iyi. Şimdi, komşu A'yı kontrol edelim. 1'i elde etmek için 0'ı (mevcut C düğümümüzün minimum mesafesi, mevcut düğümümüz) 1 (mevcut düğümümüzü A ile bağlayan kenarın ağırlığı) ekliyoruz. 1'i A'nın minimum mesafesi ile karşılaştırıyoruz. (sonsuz) ve en küçük değeri bırakın:,



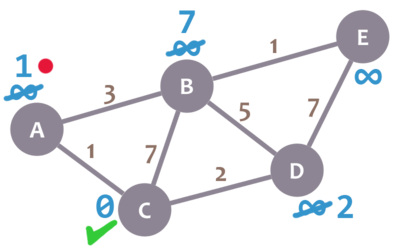
Tamam aynı prosedürü tekrarlayın.



Harika. C'nin tüm komşularını kontrol ettik. Bu nedenle onu ziyaret edilmiş olarak işaretliyoruz. Ziyaret edilen düğümleri yeşil bir onay işaretiyle temsil edelim:

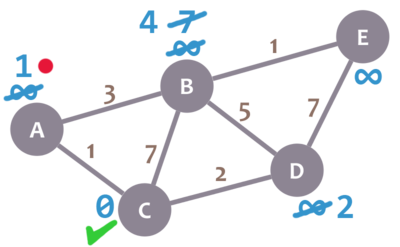


Şimdi yeni bir akım düğümü seçmemiz gerekiyor. Bu düğüm, en küçük minimum mesafeye sahip görünmeyen düğüm olmalıdır (yani, en küçük sayıya sahip olan ve onay işareti olmayan düğüm). Bu A. Kırmızı nokta ile işaretleyelim:

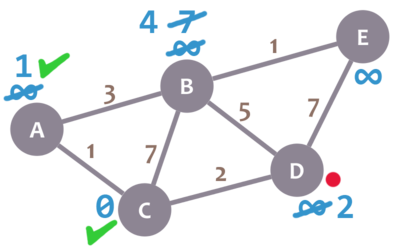


Ve şimdi algoritmayı tekrarlıyoruz. Ziyaret ettiğimiz düğümleri göz ardı ederek mevcut düğümümüzün komşularını kontrol ediyoruz. Bu, yalnızca B'yi kontrol ettiğimiz anlamına gelir.

B için, 4 elde etmek üzere 1 (A'nın minimum mesafesi, mevcut düğümümüz) 3 (A ve B'yi bağlayan kenarın ağırlığı) ekliyoruz. 4'ü minimum B (7) mesafesi ile karşılaştırıyoruz ve en küçük değer: 4.



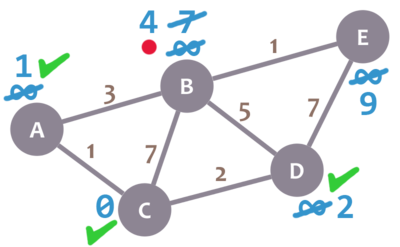
Daha sonra, A'yı ziyaret edilmiş olarak işaretleriz ve yeni bir mevcut düğüm seçeriz: D, mevcut en küçük mesafeye sahip ziyaret edilmeyen düğüm.



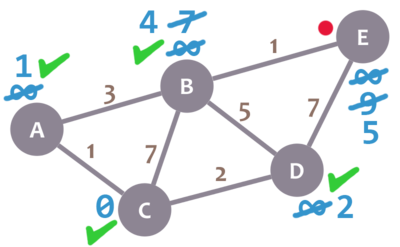
Algoritmayı tekrar tekrar yapıyoruz. Bu sefer B ve E'yi kontrol ediyoruz.

B için 2 + 5 = 7 elde ederiz. Bu değeri B'nin minimum mesafesi (4) ile karşılaştırır ve en küçük değeri (4) bırakırız. E için 2 + 7 = 9 elde ediyoruz, minimum E (sonsuz) mesafesi ile karşılaştırıyoruz ve en küçük olanı bırakıyoruz (9).

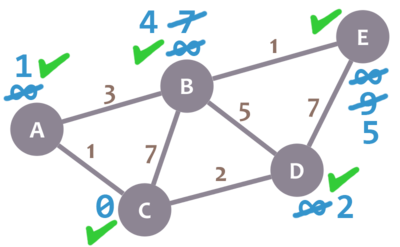
D'yi ziyaret edildi olarak işaretliyoruz ve mevcut düğümümüzü B olarak ayarladık.



Sadece E'yi kontrol etmeliyiz. E'nin minimum mesafesinden (9) daha az olan 4 + 1 = 5, bu yüzden 5'i terk ediyoruz. Sonra B'yi ziyaret edilmiş olarak işaretliyoruz ve E'yi mevcut düğüm olarak ayarlıyoruz.



E'nin ziyaret edilmemiş komşuları yok, bu yüzden hiçbir şey kontrol etmemize gerek yok. Ziyaret edildi olarak işaretliyoruz.



Bölünmemiş düğümler olmadığı için işimiz bitti! Artık her bir düğümün minimum mesafesi aslında bu düğümden C düğümüne (ilk düğüm olarak seçtiğimiz düğüm) minimum mesafeyi temsil eder!

## İşte algoritmanın açıklaması:

Seçtiğiniz başlangıç ​​düğümünü geçerli mesafe 0, geri kalanını ise sonsuz işaretleyin.

En küçük akım mesafesine sahip ziyaret edilmemiş düğümü mevcut düğüm C olarak ayarlayın.

Mevcut düğümünüz C'nin her bir komşusu N için: C'nin mevcut mesafesini C-N'yi bağlayan kenarın ağırlığına ekleyin. N'nin mevcut mesafesinden daha küçükse, N'nin yeni geçerli mesafesi olarak ayarlayın.

Geçerli düğüm C'yi ziyaret edildi olarak işaretleyin.

Ziyaret edilmeyen düğümler varsa, 2. adıma gidin.

# Projem

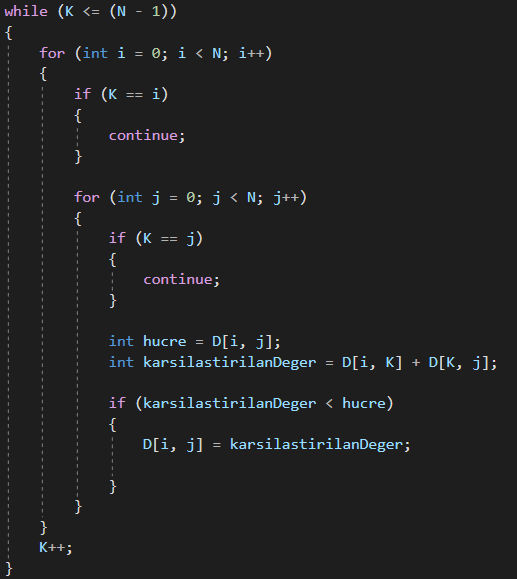
# 

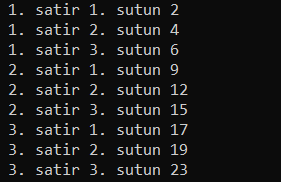


Projemi çalıştırdığımızda bize düğüm sayımızı soruyor oraya düğüm sayımızı giriyoruz.



Sonsuz değeri soruyor sonsuz değerimizi giriyoruz.





Matris sutun ve satır elemanlarını teker teker soruyor bunları dolduruyoruz.

# 



Floyd uygulayıp elde edilen matrisi gösteriyor.

# 



Bize Başlangıç düğümünü ve varış düğümünü soruyor.

Onları doldurduktan sonra



Bize düğümler arasında en kısa yolu veriyor.

# Kaynakça

https://burakbayramli.github.io/dersblog/algs/dijks/dijkstra\_algoritmasi\_ile\_en\_kisa\_yol.html

https://www.muhendisbeyinler.net/dijkstra-algoritmasi/

https://www.algoritmauzmani.com/algoritmalar/dijkstra-algoritmasi-nedir-dijkstra-ornekli-anlatim-c-kodu/

http://forum.cagataycebi.com/index.php?topic=1808.0

https://www.buraksenyurt.com/post/floyd-warshall-algoritmasi-ile-en-kisa-yolu-bulmak