

**MESLEK YÜKSEK OKULU**

**Bölüm Adı : Bilgisayar Teknolojileri**

**Program Adı : Bilgisayar Programcılığı**

**Ders : Yapay Zeka**

**Proje Adı : Minimum Yolun Bulunması**

**Öğr. Gör. : Nilgün İNCEREİS**

**Ozan Uğuz**

**18MY03014**

**İÇİNDEKİLER**

* **Giriş \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**

**1.Proje Hakkında Genel Bilgiler \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**

* **Genel Bilgiler \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**

**1.Yapay Zekâ Nedir? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**

**-Yapay Zekânın Faydaları \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 3**

**-Yapay Zekânın Zararları \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 4**

**2.En Kısa Yol Algoritması \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 4**

**-Farklı Algoritma Örnekleri Ve Açıklamaları \_\_\_\_\_\_\_\_\_\_ 5**

**3.Proje Tasarımı \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 7**

**-Algoritma Kodları \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 8**

**-Proje Sonuç Açıklama \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 10**

* **Kaynakça \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 11**

**Giriş**

**Proje Hakkında Genel Bilgiler**

Proje Yapımında kullanmış olduğum **Floyd-Warshall** algoritması **Graph** yapılarında boğumlar arasındaki en kısa yolların bulunmasında kullanılmaktadır.

Gerçek hayat örnekleri düşünüldüğünde **Regular Expression**, **Network Routing**, **Dynamic Programming**, yönsüz graph'ların iki parçalı graph'lar dönüştürülmesi ve daha birçok alanda kullanıldığına şahit oluruz. Algoritmanın matematiksel çalışmasına bakıldığında boğumların birbirlerine olan yakınlıklarını ele alan matrisleri kullandığını görürüz.

İşte **Floyd-Warshall** algoritması bir boğumdan diğer bir boğuma gitmek için kullanılabilecek en kısa yolların çıkartılmasında devreye girerek karar vermemizi kolaylaştırır.

1. **Yapay Zekâ Nedir ?**

**Yapay zekâ** bilgisayarın veya bilgisayara bağlı bir robotun insanlara özgü hareketleri kendi iradesini ve aklını kullanarak gerçekleştirme yeteneğidir. Yapay zekâ genellikle insana ait düşünme ve analiz etme kabiliyetinin robota verilmiş halidir. Yapay zekâlar öğrendiği şeyleri asla unutmaz ve bunu tekrar eder. Bu sayede öğrendiği yeni bilgileri daha da gelişmiş hale getirerek insanlığın hizmetine sunar. Yapay zekânın bir diğer işleviyse insanların çözemediği bazı karmaşık sorunları insanlar gibi düşünerek çözer.

**Yapay zekâ** genelde bilgisayar biliminde kullanılsa da tarım, matematik, tıp gibi birçok alanda da kullanılır. Yapay zekâlar bazı şeyleri gözlemler, sonra da önceden belirlenmiş parametreler temelinde o şeyi anlamaya ve tanımaya çalışır. Karşısındaki sorunu çözmeye çalışabilir ve buna tepki verebilir.

* 1. **Yapay Zekânın Faydaları**
* Hata yapma şansları yoktur.
* Uzay Biliminde yeni gök cisimleri keşfetmek için kullanılabilir. Bunlar makine olduğu için basıncın ve uzaydaki atmosferin bu robotlara herhangi bir zararı olmayacaktır.
* Bu robotlar insanların sahip olduğu bazı sınırlamaların üstesinden gelmek amacıyla da kullanılabilir. Az önceki maddedeki gibi uzay atmosferine ve basıncına dayanabileceği gibi yeraltı basıncına karşı da kullanılarak büyük fayda da sağlanabilir.
* Akıllı telefonlardaki bazı uygulamalar yapay zekâya örnek gösterilebilir. Bunlara Apple’ın Siri’si ve neredeyse tüm akıllı telefonlarda bulunan GPS ve harita uygulamaları buna örnek gösterilebilir.
  1. **Yapay Zekânın Zararları**
* Yapım, onarım gibi bazı gereksinimleri ciddi miktarda maliyetlidir.
* Programlarının değişen hayat koşullarına göre güncellenmesi ve kaybedilen kod veya verilerin tekrar yüklenmesi masraflıdır.
* Stephen Hawking ve Bill Gates gibi önemli insanların bu konu hakkında olumsuz görüşleri bulunmaktadır.
* İnsanlardaki ön sezgi ve bazı şeyleri yargılayabilme yeteneği onlarda bulunmaz.

1. **En Kısa Yol Algoritması**

Bir grafikte düğümler arasındaki, örneğin ağ yollarını temsil edebilecek en kısa yolları bulmak için kullanılan bir algoritmadır. 1956 yılında bilgisayar bilimcisi olan Edsger W. Dijkstra tarafından tasarlanmış ve üç yıl sonra yayınlanmıştır.

Bir çok varyantta algoritmalar bulunmaktadır. Dijkstra’nın orijinal algoritmasında verilen iki düğüm arasındaki en kısa yolu buluruz. Daha yaygın bir varyantta ise tek bir düğüm ‘kaynak’ düğüm olarak saptanır ve kaynak düğümden en kısa yol ağacını üreterek, grafikteki diğer tüm düğümlere giden en kısa yollar bulunur.

Grafikteki belirli bir kaynak düğüm için algoritma bu düğüm ve diğer her bir düğüm arasındaki en kısa yolu bulur. Hedef düğüme giden en kısa yol belirlendikten sonra algoritmayı durdurarak tek bir düğümden tek bir hedef düğüme en kısa yolları bulmak için de kullanılabilir. Örneğin grafiğimizde ki her düğüm bir şehri temsil ediyor diyelim. Kenar bağlantıları da şehirler arasında ki doğrudan sürüş mesafesini temsil ediyor olsun. Bir şehrin diğer tüm şehirler ile arasında ki en kısa rotayı bulmak için Dijkstra algoritması kullanılabilir. En kısa yol algoritması aynı zamanda yaygın olarak ağ yönlendirme protokollerinde de kullanılmaktadır. Özellikle IS-IS ( Ara Sistemden Orta Sisteme ) ve OSPF ( Önce En Kısa Yolu Aç ). Aynı zamanda Johnson gibi diğer algoritmalarda altyordam olarak kullanılır.

* **Floyd-Warshall Algoritması :** Bilgisayar bilimlerinde, Floyd-Warshall algoritması, pozitif veya negatif kenar ağırlıkları olan (ancak negatif döngüleri olmayan) ağırlıklı bir grafikte en kısa yolları bulmak için bir algoritmadır. Algoritmanın tek bir çalıştırılması, tüm köşe çiftleri arasındaki en kısa yolların uzunluklarını (toplam ağırlıkları) bulur. Yolların ayrıntılarını kendileri döndürmese de, algoritmada basit değişikliklerle yolları yeniden yapılandırmak mümkündür. Algoritmanın versiyonları ayrıca bir ilişkinin R veya (Schulze oylama sistemi ile bağlantılı olarak) ağırlıklı bir grafikteki tüm köşe çiftleri arasındaki en geniş yolları geçişli olarak kapatmak için de kullanılabilir.
* **Bellman – Ford Algortiması :**  Tek bir kaynak tepe noktasından ağırlıklı bir digrafideki diğer tüm köşelere en kısa yolları hesaplayan bir algoritmadır. Aynı sorun için Dijkstra'nın algoritmasından daha yavaştır, ancak kenar ağırlıklarının bazılarının negatif sayılar olduğu grafikleri işleyebildiğinden daha çok yönlüdür. Algoritma ilk olarak Alfonso Shimbel (1955) tarafından önerildi, ancak bunun yerine sırasıyla 1958 ve 1956'da yayınlanan Richard Bellman ve Lester Ford Jr. Edward F. Moore da 1957'de aynı algoritmayı yayınladı ve bu nedenle bazen Bellman-Ford-Moore algoritması olarak da adlandırılıyor.
* **Johnson Algoritması :** 1977 yılında Donald B. Johnson tarafından sunulmuştur. Bellman Ford, Reweighting ve Dijkstra Algoritması tabanlı, Tüm çiftler problemini çözmek için kullanılan bir algoritmadır. Seyrek (dağınık) ve yönlendirilmiş (yönlü) graflar için kullanılan güzel bir çözüm yoludur. Bağlantıların negatif olduğunu izin verir ve bu en önemli özelliğidir. Pozitif ağırlıklara güncellemektedir. Floyd Warshall’a benziyorum fakat O (V3) olan çalışma süresi Johnson Algoritması’nın tercih edilmesine neden olur. Ayrıca Floyd Marshall daha sık graflardatercih edilirken, Johnson seyrek graflarda daha iyi. Ağırlıkların pozitif olması gerekir Dijkstra’ın Kullanımı daha iyi performans sağlar.
* **A \* Arama Algoritması :** A \* ("A-yıldızı" olarak telaffuz edilir), tamlığı, optimitesi ve optimum verimliliği nedeniyle bilgisayar bilimlerinde sıklıkla kullanılan bir grafik geçişi ve yol arama algoritmasıdır. En büyük pratik dezavantaj, O (b ^ d) uzay karmaşıklığıdır, çünkü üretilen tüm düğümleri hafızada saklar. Bu nedenle, pratik seyahat yönlendirme sistemlerinde, genellikle daha iyi performans elde etmek için grafiği önceden işleyebilen algoritmalarla ve belleğe bağlı yaklaşımlarla daha iyi performans gösterir; ancak, A \* çoğu durumda hala en iyi çözümdür.

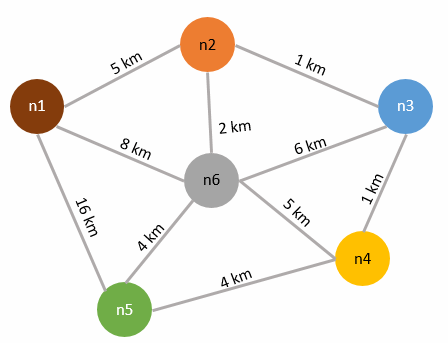
Stanford Araştırma Enstitüsü'nden Peter Hart, Nils Nilsson ve Bertram Raphael algoritmayı ilk kez 1968'de yayınladı. Edsger Dijkstra'nın 1959 algoritmasının bir uzantısı olarak görülebilir. A \* araştırmasına rehberlik etmek için sezgisel tarama kullanarak daha iyi performans elde eder***.***

* **Tarjan'ın Güçlü Bağlantı Bileşenleri Algoritması :** Tarjan'ın algoritması, grafik teorisinde yönlendirilmiş bir grafiğin güçlü bir şekilde bağlı bileşenlerini bulmak için kullanılan bir algoritmadır. Kosaraju algoritması ve yola dayalı güçlü bileşen algoritması da dahil olmak üzere alternatif yöntemler için sınırlanan zamana uygun olarak lineer zamanda çalışır.

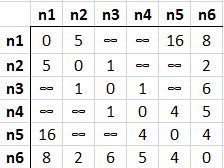
Algoritmanın temel fikri şudur: önce derinlik araması, rasgele bir başlangıç ​​düğümünden başlar (ve daha sonra ilk derinlik ilk aramaları henüz bulunmayan düğümlerde gerçekleştirilir). Her zaman olduğu gibi, derinlik ilk aramada, arama, grafiğin her düğümünü tam olarak bir kez ziyaret ederek daha önce ziyaret edilmiş olan düğümleri tekrar ziyaret etmeyi reddeder. Bu nedenle, arama ağaçlarının toplanması grafiğin genişleyen bir ormanıdır. Güçlü bir şekilde bağlanan bileşenler bu ormanın belirli alt ağaçları olarak geri kazanılacaktır. Bu alt ağaçların köklerine kuvvetle bağlı bileşenlerin "kökleri" denir. Güçlü bir şekilde bağlanmış bir bileşenin herhangi bir düğümü, arama tarafından bulunan bir bileşenin ilk düğümü olursa, kök olarak işlev görebilir.

1. **Proje Tasarımı**

Öncelikle boğumlarımıza aşağıdaki gibi numaralar verelim ve ilk olarak yakınlık matrisimizi oluşturalım. (Yakınlık matrisinin ilk versiyonu boğumların komşu boğumlar ile arasındaki mesafelerini tanımlamaktadır)



Matrisimizin ilk hali aşağıdaki gibi olacaktır.



Bazı hücrelerde sonsuzluk sembolü, bazı hücrelerde ise sıfır değeri var. İki boyutlu bu matris boğumların en yakın diğer boğuma olan mesafelerini göstermekte. Bir boğumun kendisiyle arasındaki mesafe 0, doğrudan bağlı olmadığı bir boğumlar arasındaki mesafe ise sonsuz sembolü ile işaret edilmekte. Örneğin n1 boğumundan n3 ve n4 boğumlarına doğrudan bir hat olmadığı için sonsuz sembolü kullanıldı. Algoritmanın becerisi sonsuz sembollerini eritmek ve hatta sayısal değer alan hücrelerde olabilecek daha kısa mesafeler var ise bunları matris üzerinde güncellemektir.

Örneğin n1'den n3'e direkt gidişimiz olmadığından sonsuz olarak işaretlenmiş durumda. Oysaki n1->n2->n3 şeklinde bir ulaşım var. Yani n2 üzerinden geçiş yaparak n3'e varabiliriz. Elbette n3'e varmak için n6 üzerinden de hareket edebiliriz. Yani n1->n6->n3 şeklinde bir güzergâh da söz konusu olabilir. Hatta n1->n5->n4->n3 şeklinde de gidebiliriz.

İşte matrisimizi bu şekilde algoritma içerisinde işleterek nihai haline getirmemiz gerekiyor. Tahmin edeceğiniz üzere bu, çok da uğraşmak isteyeceğimiz türden bir iş değil. Bu yüzden zaten kod yolu ile ilgili algoritmayı çalıştırmayı tercih etmekteyiz. Aşağıda algoritmanın kullanımına ilişkin bir kod parçası yer almaktadır.

* 1. **Algoritma Kodları**

using System;

using System.Linq;

namespace FloydWarshallCode

{

    class Program

    {

        static void Main(string[] args)

        {

            double[][] proximityMatrix = PrepareFirstState();

            Solve(ref proximityMatrix);

            Dump(proximityMatrix);

        }

        public static void Solve(ref double[][] matrix)

        {

            int size = matrix.Count();

            for (int i = 0; i < size; i++)

            {

                for (int j = 0; j < size; j++)

                {

                    for (int k = 0; k < size; k++)

                    {

                        matrix[j][k] = Math.Min(matrix[j][k], matrix[j][i] + matrix[i][k]);

                    }

                }

            }

        }

        private static double[][] PrepareFirstState()

        {

            double[][] matrix = new double[6][]{

                new double[6],

                new double[6],

                new double[6],

                new double[6],

                new double[6],

                new double[6]

            };

            matrix[0][0] = 0;

            matrix[0][1] = 5;

            matrix[0][2] = double.PositiveInfinity;

            matrix[0][3] = double.PositiveInfinity;

            matrix[0][4] = 16;

            matrix[0][5] = 8;

            matrix[1][0] = 5;

            matrix[1][1] = 0;

            matrix[1][2] = 1;

            matrix[1][3] = double.PositiveInfinity;

            matrix[1][4] = double.PositiveInfinity;

            matrix[1][5] = 2;

            matrix[2][0] = double.PositiveInfinity;

            matrix[2][1] = 1;

            matrix[2][2] = 0;

            matrix[2][3] = 1;

            matrix[2][4] = double.PositiveInfinity;

            matrix[2][5] = 6;

            matrix[3][0] = double.PositiveInfinity;

            matrix[3][1] = double.PositiveInfinity;

            matrix[3][2] = 1;

            matrix[3][3] = 0;

            matrix[3][4] = 4;

            matrix[3][5] = 5;

            matrix[4][0] = 16;

            matrix[4][1] = double.PositiveInfinity;

            matrix[4][2] = double.PositiveInfinity;

            matrix[4][3] = 4;

            matrix[4][4] = 0;

            matrix[4][5] = 4;

            matrix[5][0] = 8;

            matrix[5][1] = 2;

            matrix[5][2] = 6;

            matrix[5][3] = 5;

            matrix[5][4] = 4;

            matrix[5][5] = 0;

            return matrix;

        }

        public static void Dump(double[][] matrix)

        {

            int size = matrix.Count();

            for (int i = 0; i < size; i++)

            {

                for (int j = 0; j < size; j++)

                {

                    Console.Write("{0}\t", matrix[i][j]);

                }

                Console.WriteLine();

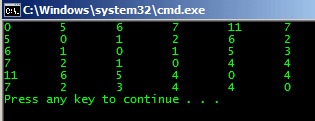
            }

        }

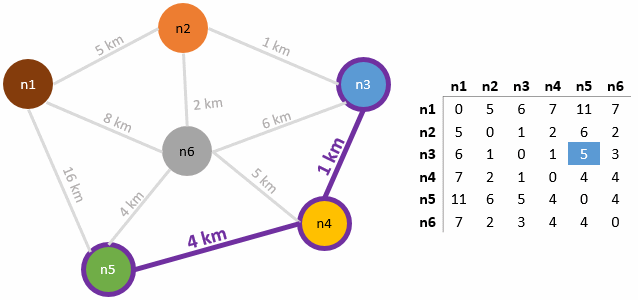
    }

}

Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuç matrisini elde ederiz.



Buna göre bir noktadan bir noktaya gidilebilecek en kısa mesafeler bulunmuştur. Örneğin **n3 noktasından n5 noktasına gitmek istediğimizde en kısa güzergâh 5km** **uzunluğunda olup n3->n4->n5 rotası** şeklindedir. Diğer alternatif yollara bakıldığında gerçekten de en kısa mesafenin bu olduğu açıkça görülebilir.



**Uygulama Visual Studio ortamında C# dili ile hazırlanmıştır.**

**Kaynakça**

<https://www.bilgiustam.com/yapay-zeka-nedir/>

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

<https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm>

<https://slideplayer.biz.tr/slide/2742392/>

<https://blog.emretopcuoglu.com/dijkstra-en-kisa-yol-algoritmasi/>