

**Adı:** Ayhan Metehan

**Soyadı:** Aksoy

**Numarası:** 18MY03019

**Konu:** Minimum yolun bulunması problem raporu

**İÇİNDEKİLER**

**1. Giriş ...................................................................................................**

**2. Genel Bilgiler........................................................................................**

**2.1 Yapay Zekada Minimum Yolun Bulunması Problemi……………..**

**2.2 Minimum Yolun Bulunması Uygulaması Tasarımı………………….**

**2.3 Algoritma………………………………………………………………………………**

**2.4 Program Kodları……………………………………………………………………..**

**3. Özgeçmiş……………………………………………………………………………………………**

**4. Kaynakça ……………………………………………………………………………………………**

**5. Ekler……………………………………………………………………………………………………**

**1.Giriş**

Projemin konusu en kısa yolun bulunması problemi. Projedeki amacım iki nokta arasındaki en kısa yolun bulunabilmesi için uygulama geliştirmek. Uygulamamı yapmak için çeşitli kaynaklar kullandım. Bu projeyi tasarlarken uygulama ortamı olarak Netbeansi dil olarak da Javayı kullandım.

**2. Genel Bilgiler**

Yapay Zeka da en kısa yol problemi, bir grafikte iki köşe arasında bir yol bulma problemidir, öyle ki kurucu kenarlarının ağırlıklarının toplamı en aza indirilir. *Mümkün olan en kısa sürede iki nokta arasındaki en kısa yolu bulmamız gerekir.* En Kısa Yol Problemi’nde kullanılan algoritmalardan söz edecek olursak;

-Djikstra Algoritması

-Bellman Ford Algoritması

-Floyd-Warshall Algoritması

-Johnson Algoritması

-A Search Algoritması

-Dinitz Algoritması

-Tarjan Algoritması

-GorAlgoritması

-Dial Algoritması

Algoritmalardan bazılarına kısaca göz atacak olursak;

Djikstra Algoritması: Single Source, SinglePair ve SingleDestination problemlerini çözmek için kullanılır. Döngüye girme ihtimalinden dolayı negatif ağırlıklı bağlantıları kabul etmez. Basit bir algoritmadır. Buna karşı performansı yüksektir.

Bellman Ford Algoritması: Single Source problemlerini kenar ağırlıkları negatif olduğu durumlarda da çozebılmektedır.

A Search Algoritması: İki nokta arasındaki en kısa yolu aramayı hızlandırmak için sezgisel yöntemler kullanarak çözebilmektedir.

Floyd-Warshall Algoritması: Allpairs probleminin çözümünde kullanılır.

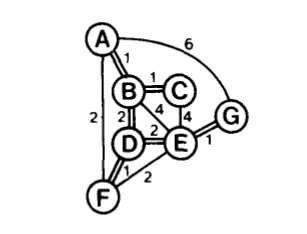
Johnson Algoritması: Floyd WarshallAlgoritması’ndaki gibi allpairs problemini çözmeye yarar. Sparse(dağınık) ve yönlü graflar için çözün sunar.

**Graf Teorisi**

Graf Teori yani Cizge Kuramı, Çizelgeleri yani grafları inceleyen matematik dalıdır. Graflarnode (ya da vertice) yani düğümler ve edge yani bağlantılardan oluşan bir çeşit ağ yapısıdır.

Kısa Yolun Bulunması Problemi

Kısa Yolun Bulunması Problemi, Graf Teorisi içerisinde önemli bir problemdir. Graf içerisinde belirtilen node’lar arasındaki en kısa yolu bulmayı amaçlar. Bu özelliğiyle Gezgin Satıcı Problemi’ne de benzer. Ancak Gezgin Satıcı’dan iki noktada ayrılır.Birincisi, hedef düğüme giderken tüm düğümlere uğramak zorunda değildir. İkincisi de hedefe vardıktan sonra başlangıca dönüş yoktur.

En kısa yollar döngüler içeremez, ancak negatif bağlantı değerleri bulundurabilirler. Bununla birlikte n tane düğümün olduğu bir grafta en kısa yol n-1 kenardan daha fazla kenara sahip olamaz.

Kaynak nodun diğer tüm node’lara olan en kısa yollarını bularak Graf Teorisi içerisinde yer alan En Kısa Yol Ağacı’nı da oluşturabiliriz.

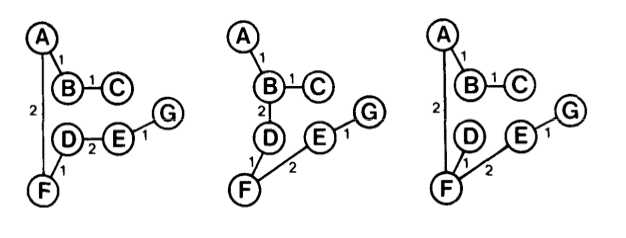
**Minimum Kapsamlı Ağaç (Minimum Spanning Tree -MST-)**

MST algoritması kenar ağırlıklarına (weights) sahip olan bir çizit (graph) yapısı içinde minimal ve kapsayan ağacı (MST) bulan algoritmaya verilen isimdir. Mesela alttaki çizit içinde

MST öyle bir bağlantı yapısıdır ki baştan sona, herhangi bir noktadan (node) bir diğerine geçiş yapılabilsin ve bu tüm yolların toplamı en minimal olsun. Dikkat, herhangi bir noktadan diğerine giden yol en az olsun demiyoruz, bu durumda problem en kısa yol (shortestpath) problemi olurdu. Burada göreceğimiz kapsayan ağacın toplamının minimal olmasıdır.

Kapsayan ağaç (spanningtree) kavramını tanımlamak gerekirse, bir çizitin kapsayan ağacı orijinal çizitin tüm noktalarına sahip olmalıdır, ağaç içinde hiçbir döngü (cycle) olmamalıdır. Döngü derken bir noktadan diğerine atlaya atlaya giderken bizi dönüp tekrar aynı yere getirebilecek türden "kapalı devre" tur bir döngüden bahsediyoruz- bu mümkün olmamalıdır. Ayrıca çizit bağlantılı olmalıdır, yani bir kısmı diğer kısmından kopuk bir çizit üzerinde MST bulunamaz.

Minimum kapsayıcı ağaç ise bu tür pek çok alternatif ağaçların içinde en az ağırlıklı olanıdır. Not, bir çizitin MST çözümü özgün (unique) olmayabilir, aynı ağırlıkta birden fazla değişik ağaç mümkündür. Mesela üstteki çizit için mümkün MST'ler altta görülüyor,



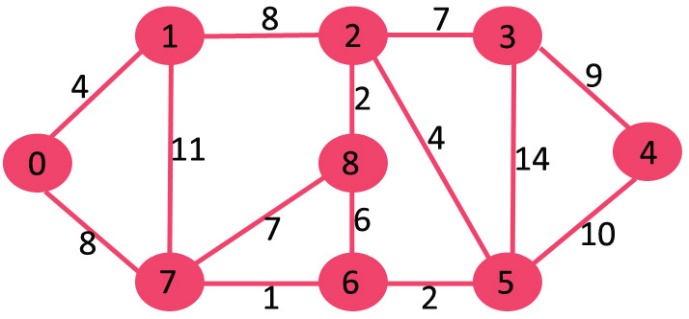
Kontrol edilebilir, üstteki her ağacın kenar toplamı 8'dır. Bu ağaçların her biri bir MST olarak kabul edilebilir.

Uygulama bağlamında MST bulma algoritmasının ne kadar kullanışlı olacağı görülüyor herhalde; mesela elektrik hatları, telefon iletişim hatları tasarlarken MST kullanılabilir, toplam bağlantısı en az olan bir ağ yapısı her iki durumda da kullanışlı olur. Biyolojik, kimyasal ağların analizinde bile MST kullanılmaktadır.

Dijkstra'nın algoritması,Primin minimum yayılan ağaç algoritmasına çok benzer . Prim'in MST'si gibi, verilen kaynak kök olarak bir *SPT (en kısa yol ağacı)* üretiyoruz . İki set tutuyoruz, bir set en kısa yol ağacına dahil olan köşeleri içeriyor, diğer set henüz henüz en kısa yol ağacına dahil olmayan köşeleri içeriyor. Algoritmanın her adımında, diğer kümede (henüz dahil edilmemiştir) olan ve kaynaktan minimum mesafeye sahip bir tepe noktası buluruz.

Aşağıda, Dijkstra'nın algoritmasında, tek bir kaynak tepe noktasından verilen grafikteki diğer tüm köşelere en kısa yolu bulmak için kullanılan ayrıntılı adımlar verilmiştir.  
Algoritma  
**1)** En kısa yol ağacına dahil olan köşeleri izleyen, yani kaynaktan minimum mesafesi hesaplanan ve sonlandırılmış olan bir set *sptSet* (en kısa yol ağacı seti) oluşturun. Başlangıçta, bu küme boştur.  
**2)** Giriş grafiğindeki tüm köşelere bir mesafe değeri atayın. Tüm mesafe değerlerini INFINITE olarak başlat. Kaynak köşe noktası için uzaklık değerini önce seçilecek şekilde 0 olarak atayın. **3)** iken *sptSet* yaptığı tüm köşeleri dahil değildir

.... **a)***sptSet* içinde olmayan bir köşe u seçinve minimum mesafe değerine sahiptir.  
.... **b)** u *sptSet'e* ekleyin .  
.... **c)** Tüm bitişik u köşelerinin mesafe değerini güncelleyin. Mesafe değerlerini güncellemek için tüm bitişik köşelerde yineleme yapın. Her bitişik köşe v için, u'nun (kaynaktan) mesafe değeri ve uv kenarının ağırlığı toplamı v'nin uzaklık değerinden düşükse, v'nin mesafe değerini güncelleyin.

Aşağıdaki örnekle anlayalım:  
[](https://www.geeksforgeeks.org/wp-content/uploads/Fig-11.jpg)

Ayarlanan *sptSet* başlangıçta boştur ve köşelere atanan mesafeler {0, INF, INF, INF, INF, INF, INF, INF, INF} olup, burada INF sonsuzdur. Şimdi tepe noktasını minimum mesafe değeriyle seçin. *Köşe* 0 alınır, *sptSet* içine *ekleyin* . Böylece *sptSet* {0} olur. *SptSet değerine* 0 *ekledikten* sonra , bitişik köşelerinin uzaklık değerlerini güncelleyin. 0'ın bitişik köşeleri 1 ve 7'dir. 1 ve 7 arasındaki mesafe değerleri 4 ve 8 olarak güncellenir. Aşağıdaki alt çizgi, köşeleri ve mesafe değerlerini gösterir, yalnızca sonlu mesafe değerlerine sahip köşeler gösterilir. SPT'de bulunan köşeler yeşil renkte gösterilir.

[](https://www.geeksforgeeks.org/wp-content/uploads/MST1.jpg)

Minimum mesafe değeri olan ve zaten SPT'ye dahil olmayan (sptSET'te değil) tepe noktasını seçin. Tepe noktası 1 seçilir ve sptSet'e eklenir. Böylece sptSet artık {0, 1} oluyor. Bitişik köşelerin 1 mesafe değerlerini güncelleyin. Köşe 2'nin mesafe değeri 12 olur.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ2.jpg)

Minimum mesafe değeri olan ve zaten SPT'ye dahil olmayan (sptSET'te değil) tepe noktasını seçin. Köşe 7 seçilir. Böylece sptSet artık {0, 1, 7} oluyor. Bitişik köşe 7'nin mesafe değerlerini güncelleyin. Tepe 6 ve 8'in mesafe değeri sonlu olur (sırasıyla 15 ve 9).  
[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ3.jpg)

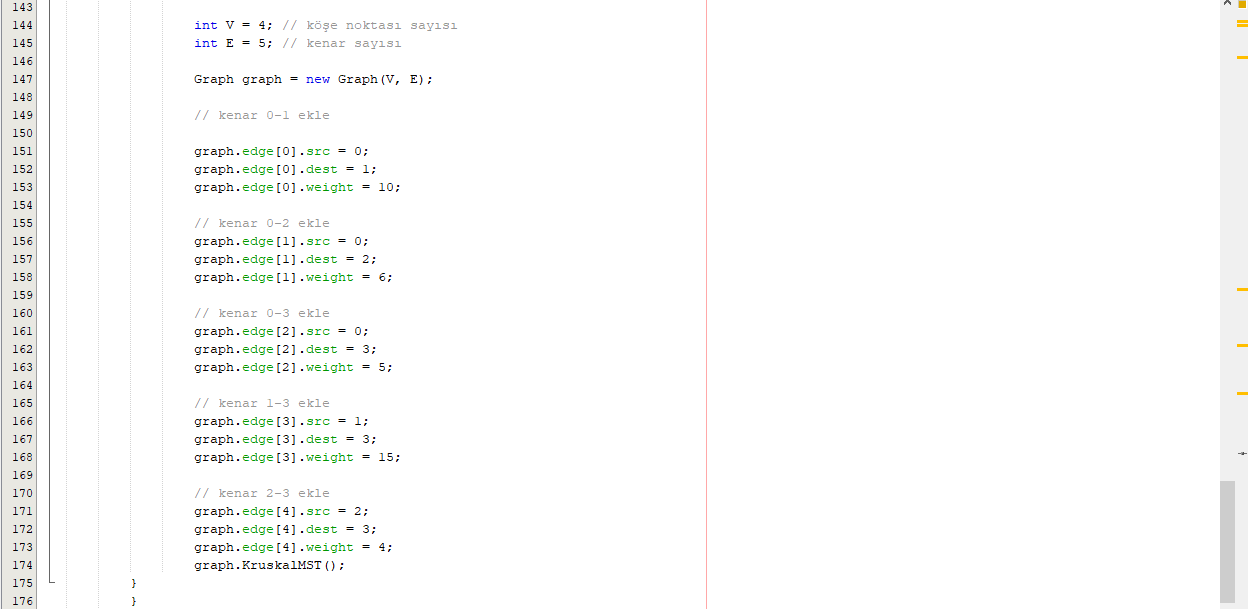
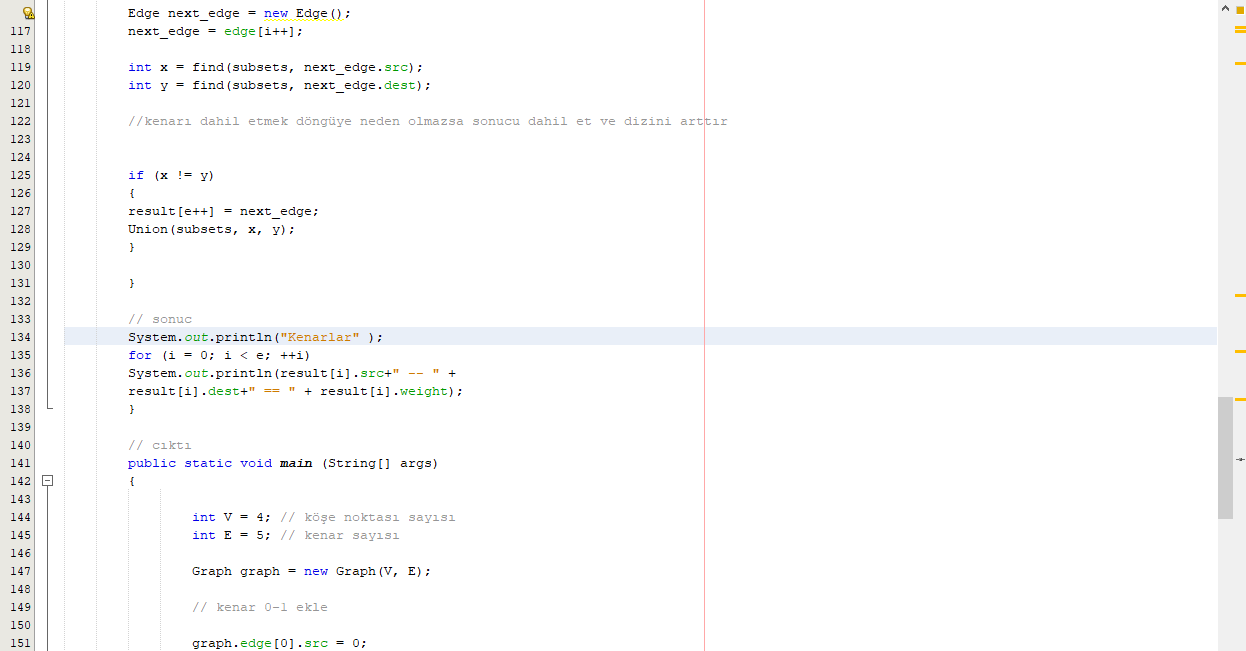
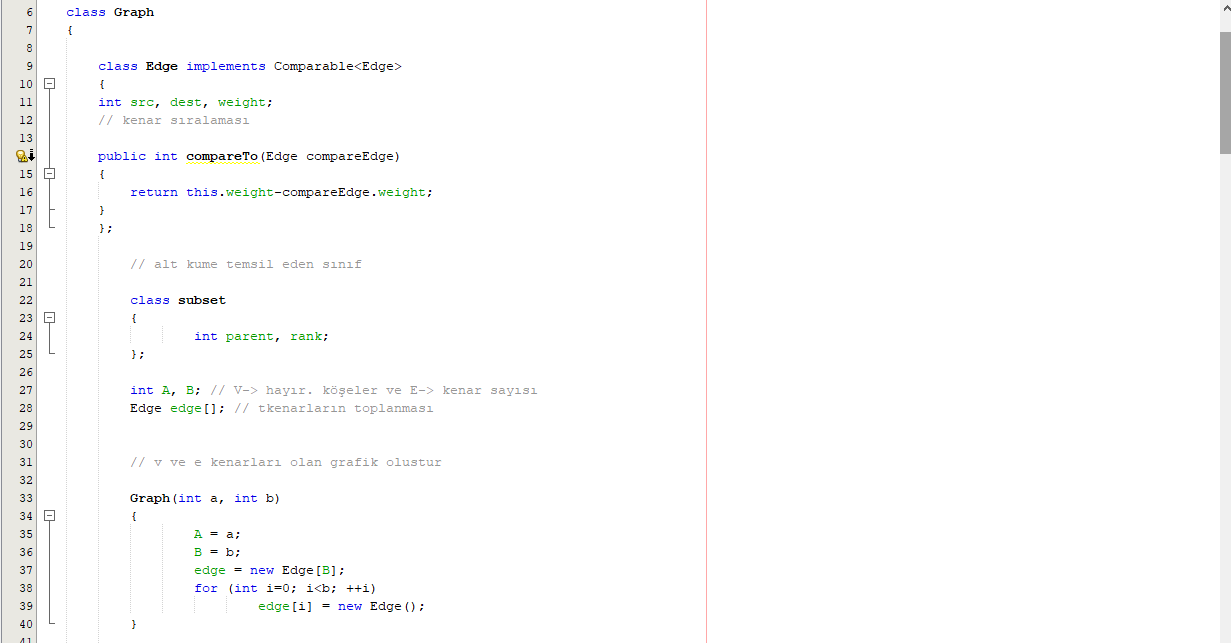
Minimum mesafe değeri olan ve zaten SPT'ye dahil olmayan (sptSET'te değil) tepe noktasını seçin. Köşe 6 seçilir. Böylece sptSet artık {0, 1, 7, 6} oluyor. Bitişik köşelerin 6 mesafe değerlerini güncelleyin. Köşe 5 ve 8'in mesafe değeri güncellenir.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ4.jpg)

*SptSet* verilen grafiğin tüm köşelerini *içerene* kadar yukarıdaki adımları tekrarlıyoruz . Son olarak, aşağıdaki En Kısa Yol Ağacı'nı (SPT) elde ederiz.

[](https://www.geeksforgeeks.org/wp-content/uploads/DIJ5.jpg)

**Program Kodları(Ekler):**



**Kaynakça:**

<https://www.tutorialspoint.com/>

<https://en.wikipedia.org/>

<https://www.geeksforgeeks.org/>

<https://www.hackerearth.com/>

<https://www.researchgate.net/>