# Gezgin Kargo Problemi

Programlama Labaratuvarı 2

BLM210

2019-2020

Hilal Özkan 180201017
Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi
*hilal_ozkan1200@hotmail.com*

Edmond Vujici 170201127
Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi
edmondvujic6@gmail.com

***Abstract - Özet***

**Gezgin kargo probleminde amaç, Kocaeli'nden başlanarak kullanıcının uğraması gerektiği şehirlerden geçip en kısa yolu bularak tekrar başladığı şehire (Kocaeli) gelmesidir. Bu proje gezgin satıcı probleminin uyarlanmasıdır çünkü gezgin kargo probleminde ziyaret edilen bir şehir tekrar ziyaret edilebilir. Bu proje kapsamında bulunan kısa yollar sıralı bir şekilde *tum_yollar.txt* dosyasına yazdırılmıştır ve bu yollardan beş tanesi seçilip temsili bir harita üzerinde resmedilmiştir.**

## I. Anahtar Kelimeler

Graph Teorisi, En Kısa Yol Bulma, Dijkstra Algoritması, Nöron Ağı, Dosyaya Yazdırma, GraphStream Kütüphanesi, Alternatif Yol Bulma, Gezgin Satıcı Problemi, Algortima Geliştirme, Java Swing, Komşuluk Listesi.

## II. Giriş

Gezgin Satıcı Problemimde amaç, bir satıcının bulunduğu şehirden başlayarak her şehre sadece bir kez uğradıktan sonra başladığı şehre dönebilmesi için en kısa yolun bulunmasıdır. Bu problem polinom zamanda çözülmesi mümkün olmadığından NP (non-polynomial problem) sınıfına girmektedir. Bu projedeki problem, Gezgin Satıcı Probleminden uyarlanarak aşağıda tanımlanmıştır.

Bir nakliye firmasının en az maliyetli taşıma ağını yapması ve internet ağ trafiği protokolü problemleri kullanım alanlarına örnek verilebilir.

Projeyi yapan kişi için veri yapıları ve veri modellerini anlamak, graf yapısının kullanılması ve algoritma mantığını kullanarak bir probleme çözüm sağlayabilmesi amaçlanmaktadır.

Merkezi Kocaeli'nde kurulan yeni bir kargo firması, siparişlerini en kısa yoldan yerlerine ulaştırmayı amaçlamaktadır. Bu amaç doğrultusunda bize başvurulmaktadır.

## III. Temel Kavramlar

Proje gelişiminde: Programlama dili olarak Java dili kullanılmıştır. Geliştirme ortamı olarak **InteliJ IDEA Community Edition** kullanılmıştır. Projeyi çalıştırmak için projenin klasörüne *sehir.txt* dosyasını ekleyiniz ve InteliJ'i çalıştırınız. **Mutlaka GraphStream kütüphanesi Maven üzerinden indirilmelidir (org.graphstream1.3)**. Programı Run butonuna basarak çalıştırınız.

## IV. Yalancı Kod

### *A.* Sınıf DjikstraEnKisaYol

- **Prosedür EnKisaYolHesapla(*Dugum kaynakDugum*)**
  - kaynakDugummün uzaklığı sıfır yap.
  - OncelikKuyruğu tanımla.
  - OncelikKuyruguna kaynakDugum'ü ekle.
  - kaynakDugum'ü ziyaret et.
  - OncelikKuyrugu boş değilken
    - suankiDugum'e oncelikKuyrugun en üstteki elemanını ata.
    - Kenarları gezerken mevcut dugumun komşuları al ve dugume Kenar sınıfının hedefDugum al.
      - Eğer v dugumu ziyaret edilmiş değil ise yeniUzaklığa mevcut dugumun uzaklığı ve kenar ağırılığının toplamını ata.
      - Eğer yeniUzaklık v düğümünün uzaklığından daha küçük ise öncelikKuyrugundan v düğümünü sil.
      - V düğümünün uzaklığına yeniUzaklik at.
      - V 'nin öncekine mevcutDugumu ata.
    - OncelikKuyruguna v düğümünü ekle.
  - MevcutDugumu ziyaret edilmiş olarak düzelt.
  - ÖncelikDugumunu boşalt.

*V.* **Prosedür ArrayList<String> getEnKisaYol (*hedefDugum*)**

- Yol isimli ArrayList tanımla.
- Dugumleri gezerken
  - yol ArrayListine ekle.
- yol'u tersine çevir.
- yol'u döndür.

*A.* **Prosedür AlternatifYol (*Dugum start, List <String> gezilecekler*)**

- AlternatifRota ArrayList'i tanımla.
- gidis adlı int tanımlatıp sıfır ata.
- DjakstraEnKisaYol tipinde alternatif obje oluştur.
- Son dugum tanıplayıp null ata.
- int tipinde AlternatifUzakliklar ArrayLis'i tanımla.
- gezilecek listesi boş değilken
  - Alternatif obje üzerinden start dügümünden en kısa yolu bul.
  - Gezilecekleri gezerken
    - AlternatifUzaklıklara uzakliklari ekle.
    - Mesafe yazdir.
    - Yolu yazdir.
  - AlternatifRotalara yolları ekle.
  - Düğümleri gezerken
    - düğümlerin uzaklıklarını sonsuz olarak düzlet.
    - düğümleri ziyaret edilmemiş olarak düzelt.
    - Düğümlerin öncekilerini boşalt.
  - Gidis' e sıfıncı AlternatifUzaklıkları ata.
  - start'a gezileceklerin sıfırıncı duğümü ata.
  - Gezileceklerden sıfırncı elemani sil.
  - Eğer gezileceleceklerin büyüklüğü bir ise son düğüme son şehiri ata.
- Düğümleri boşalt.
- AlternatifUzaklıklar ArrayList'i boşalt.
- Alternatif obje üzerinden son dugume kadar en kısa yolu hesapla.
- Uzaklık yazdir.
- Yolu yazdır.
- Mesafleri topla ve yazdir.
- AlternatifRotaya dönüş yolunu ekle.
- Düğümleri tekrar boşalt.
- Rotaları yazdir.

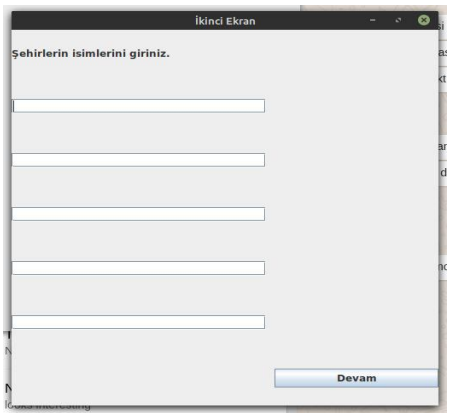
## VI. YÖNTEM

Proje kapsamında ilk olarak sehir.txt dosyasını okutup sehirleri sehir_atama fonksiyonu ile AnaSehir classının içerisine atadık daha sonra AnaSehir sınıfında her şehirin KomsuSehir class tipinde nesne oluşturarak ArrayList tanımladık. Böylece komsu_atama fonksiyonunda komşu şehirleri ana şehirin ArrayListine atadık.

Bu proje kullandığımız şehirler için AnaSehir ve KomsuSehir classlarını oluşturduk. Algoritması için Kenar, Dugum ve DijkstraEnKısaYol classları kullanıldı. Yazılımı kullanıcıya bir arayüz halinde sunmak için GirişEkranı, İkinciEkran ve UcuncuEkran frame sınıflarını kullandık.

Giriş Ekranında kullanıcıya kaç şehir gezmek istediği sorusunu sorarak gezilecek şehirlerin sayısını kullanıcıdan aldık.


İkinici ekranda kullanıcının verdiği şehir sayısı kadar kullanıcıdan gezilecek şehirlerin isimlerini aldık.




Asıl algoritmayı UcuncuEkran classında çalıştırarak en kısa yolu ve tum yolları bularak dosyaya yazdırmasını sağladık.

En kısa yolu bulmak için şehir isimlerini düğümlere atadık. Kullanıcıdan hangi şehirleri gezmek istediğini İkinciEkranda girmesini istedik. Girilen şehirler tek tek **IV B** prosedürüne atıp Dijkstra algortimasını uyguladık. Dijkstra algoritmasını uygulama adımları:

Başladığımız düğüme ilk düğüm denir. Y düğümünün maliyeti, ilk düğümden Y'ye olan maliyete denir. Dijkstra algoritmasında bazı başlangıç mesafe değerlerinin atılması gereklidir ve en kısa yolu bu mesafeler üzerinden adım adım geliştirmeye çalışılacaktır.

1. Tüm düğümleri ziyaret edilmemiş olarak işaretlenir. Ziyaret edilmemiş küme adı verilen tüm ziyaret edilmemiş düğümlerin bir kümesi oluşturulur.

2. Her düğüme geçici bir maliyet değeri atılır: ilk düğümümüz için sıfır ve diğer tüm düğümler için sonsuz ayarlanır.

3. Mevcut düğüm için, tüm ziyaret edilmemiş komşuları bulunur ve maliyeti hesaplanır komşu olmayanlara belirsiz (sonsuz) değeri atılır. Yeni hesaplanan geçici maliyeti mevcut değerle karşılaştırılır ve daha küçük olan atanır. Örneğin, mevcut düğüm A (6 maliyetli) işaretlenmişse ve onu komşu B ile bağlayan kenarın maliyeti 2 ise, B'den A'ya olan maliyeti 6+ 2 = 8 olacaktır. 8'den küçük bir maliyet hesaplanınca yerine atanır. Aksi takdirde, geçerli değer korunur.

4. Mevcut düğümün ziyaret edilmemiş tüm komşuları kontrol edildiğinde, mevcut düğüm ziyaret edilmiş olarak işaretlenir ve ziyaret edilmemiş komşuların olduğu kümeden çıkarılır.

5. Hedef düğüm ziyaret edildi olarak işaretlenmişse algoritma biter.

6. Aksi takdirde, en küçük mesafe ile işaretlenmiş, ziyaret edilmemiş düğüm seçilir, yeni "geçerli düğüm" olarak ayarlanır ve 3. adıma geri dönülür.

Djikstra Algoritmanın projemizde uygulama adımları aşağıdadır:

a) İlk önce başlangıç şehirini Kocaeli olarak atadık.

b) Kullanıcıdan aldığımız gezilecek şehirlerin dizilimini değiştirerek farklı permütasyonlarını elde ettik. Örneğin: kullanıcı Edirne-Bursa-Adana şehirlerini gezmek istediğinde Bursa-Adana-Edirne, Edirne-Adana-Bursa, Adana-Bursa-Edirne... gibi.

c) Elde ettiğimiz her permütasyon bir yola denk geldi. Yani oluşturulan dizilimler farklı yolları oluşturdu.

d) Kocaeli'nden başlayarak dizilimdeki gidilecek ilk şehir için Kocaeli ile arasındaki mesafeyi ve rotayı Dijkstra algortiması ile bulduk. Ve dizilimlerden hedef olan düğümü ziyaret edildikten sonra çıkardık.

e) Bu iki şehir arasındaki mesafeyi kaydetip hedef olan şehirimizi kaynak şehir yaptık.

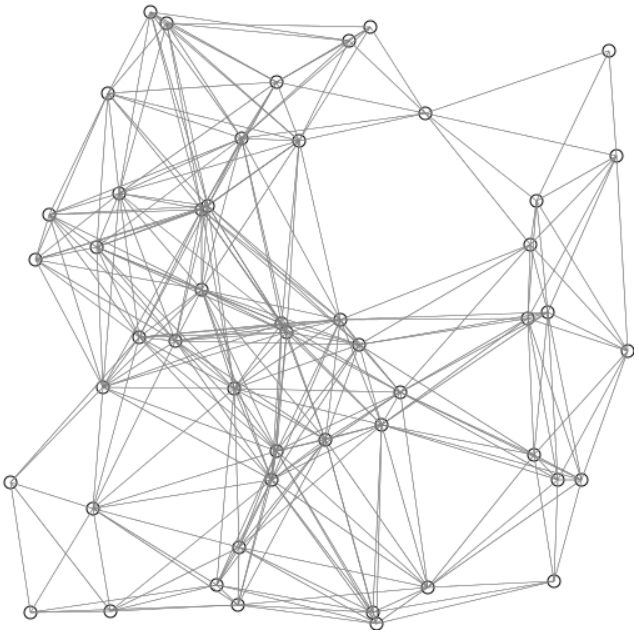
f) Gezileceklerde yeni oluşan kaynak şehir için dizilimde gidilecek ikinci şehiri hedef düğümü seçtik.

g) Gezilecekler listesi bitine kadar aynı algortimayı birden fazla kez kullandık. Burada asıl önemli olan nokta Dijkstra algoritmasını kaynak ve hedef şehiri değiştirerek birden fazla kullanmamızdır ve bu şekilde gezilecek şehirler rotasını tamamlamamızdır.

h) Her permütasyon için bir yol elde ettikten ve bu yollardaki gezilecekler listesi boşaldıktan sonra elde ettiğimiz şehirleri mesafelerine göre küçükten büyüğe sıraladık.

i) Bu yollar arasında 0. indisteki yolu en kısa yol olarak kullanıcıya sunduk. Bazı alternatif yollarda elde edilen toplam mesafe eşit olabilmektedir fakat şehirlerin gezilme sıraları farklıdır.

Düğümler şehir ve aralarındaki kenarları mesafe olarak kabul edilirse projemizdeki temsili norön ağını gösterebiliriz.



Alternatif yolları ***tum_yollar.txt*** dosyasına sıralı bir şekilde yazdırdıktan sonra, eğer gezilecek şehir sayısı bir ise bir yolu, iki ise iki yolu (dizilim sayısından dolayı) şehir sayısı üçten fazla ise beş yolu çizdirerek kullanıcaya sunduk. Yolların çizdirilmesi için Maven'den indirmiş olduğumuzu GrapStream kütüphanesi kullanarak Türkiye haritasını temsil edecek bir graf oluşturduk. ***tum_yollar.txt*** dosyasına yazdırdığımız yollardan ilk beş tanesini seçip kullanıcıya beş ayrı yolu gösteren graf sunduk. Böylece kullanıcı bu beş grafı karşılaştırarak kendisine en uygununu seçebilir.

## VII. Zorlandığımız Noktalar

Algoritma seçmede hangi algoritmayı kullanacağımıza karar vermede sorun yaşadık. Proje dokümanında algoritmaları tek tek inceledikten sonra Dijkstra'nın Algoritmasını kullanacağımıza karar verdik. İlk başta yüz yüze projeyi geliştirmeye çalışıyorduk ama dünyanın mevcut durumu (COVID-19 krizi) yüz yüze takım olarak geliştirmemize engel oldu. Bundan dolayı başka yöntemlere başvurduk. Discord uygulamasını kullanmaya

başladık ve bağlantı sıkıntıları olmasına rağmen başarılı bir şekilde projemizi geliştirdik.

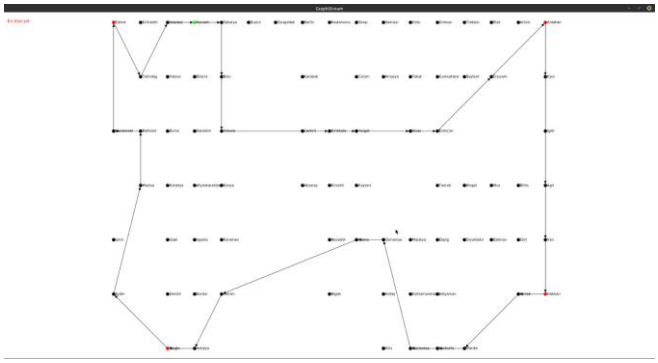
Algoritmanın görsel olarak göstermemizde bir kaç gün seçeneklerimizi inceledikten sonra en uygun olan GraphStream kütüphanesini seçtik. GraphStream'de temsili Türkiye haritasını yaparken hangi şehrin hangi konuma geleceğini tartıştık. Görev dağılımı ise, her gün beraber Discord üzerinden konuştuğumuzdan dolayı eşittir.

## VIII. SONUÇ

Kullanıcı dört tane şehir girdi, bu şehirler ; Ardahan, Edirne, Muğla ve Hakkari'dir.

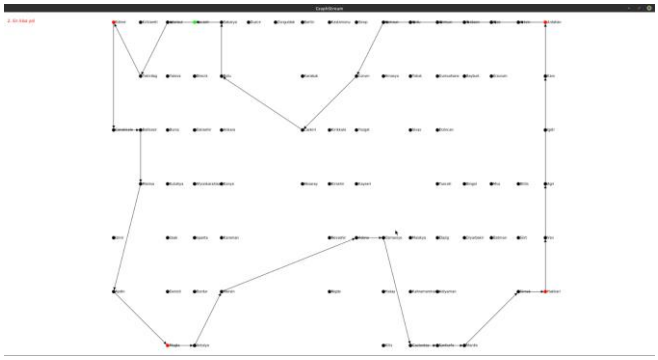
***tum_yol.txt*** dosyasının bir kısmı :

1. yol : [Kocaeli, Sakarya, Bolu, Ankara, Kirikkale, Yozgat, Sivas, Erzincan, Erzurum, Ardahan, Kars, Agri, Van, Hakkari, Sirnak, Mardin, Sanliurfa, Gaziantep, Osmaniye, Adana, Mersin, Antalya, Mugla, Aydin, Manisa, Balikesir, Canakkale, Edirne, Tekirdag, Istanbul, Kocaeli,]



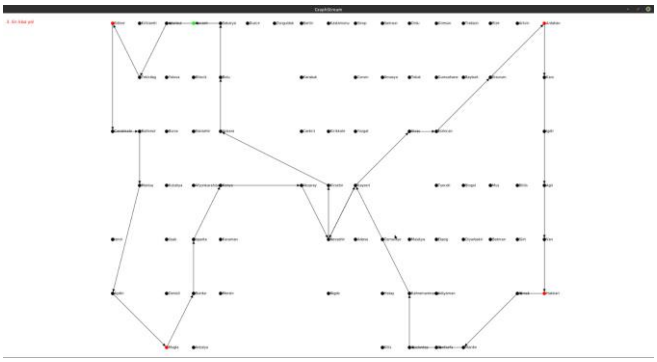
Uzaklık : 5045

2. yol : [Kocaeli, Istanbul, Tekirdag, Edirne, Canakkale, Balikesir, Manisa, Aydin, Mugla, Antalya, Mersin, Adana, Osmaniye, Gaziantep, Sanliurfa, Mardin, Sirnak, Hakkari, Van, Agri, Kars, Ardahan, Artvin, Rize, Trabzon, Giresun, Ordu, Samsun, Corum, Cankiri, Bolu, Sakarya, Kocaeli,]

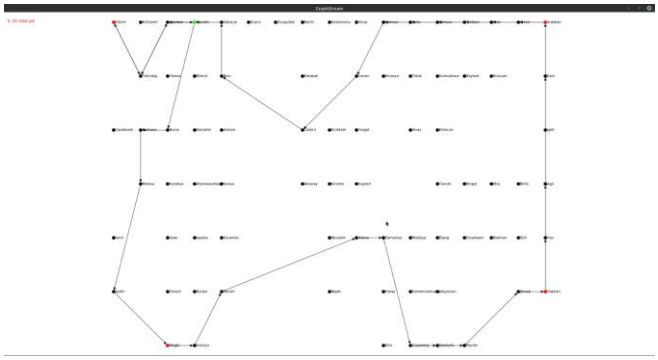


Uzaklık : 5087

3. yol : [Kocaeli, Istanbul, Tekirdag, Edirne, Canakkale, Balikesir, Manisa, Aydin, Mugla, Burdur, Isparta, Konya, Aksaray, Nevsehir, Kayseri, Sivas, Erzincan, Erzurum, Ardahan, Kars, Agri, Van, Hakkari, Sirnak, Mardin, Sanliurfa, Gaziantep, Kahramanmaras, Kayseri, Nevsehir, Kirsehir, Ankara, Bolu, Sakarya, Kocaeli,]
Uzaklık : 5294



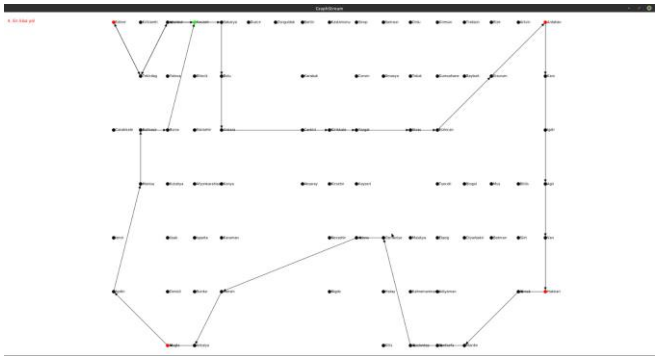
4. yol : [Kocaeli, Istanbul, Tekirdag, Edirne, Tekirdag, Istanbul, Kocaeli, Sakarya, Bolu, Ankara, Kirikkale, Yozgat, Sivas, Erzincan, Erzurum, Ardahan, Kars, Agri, Van, Hakkari, Sirnak, Mardin, Sanliurfa, Gaziantep, Osmaniye, Adana, Mersin, Antalya, Mugla, Aydin, Manisa, Balikesir,



Bursa, Kocaeli,]

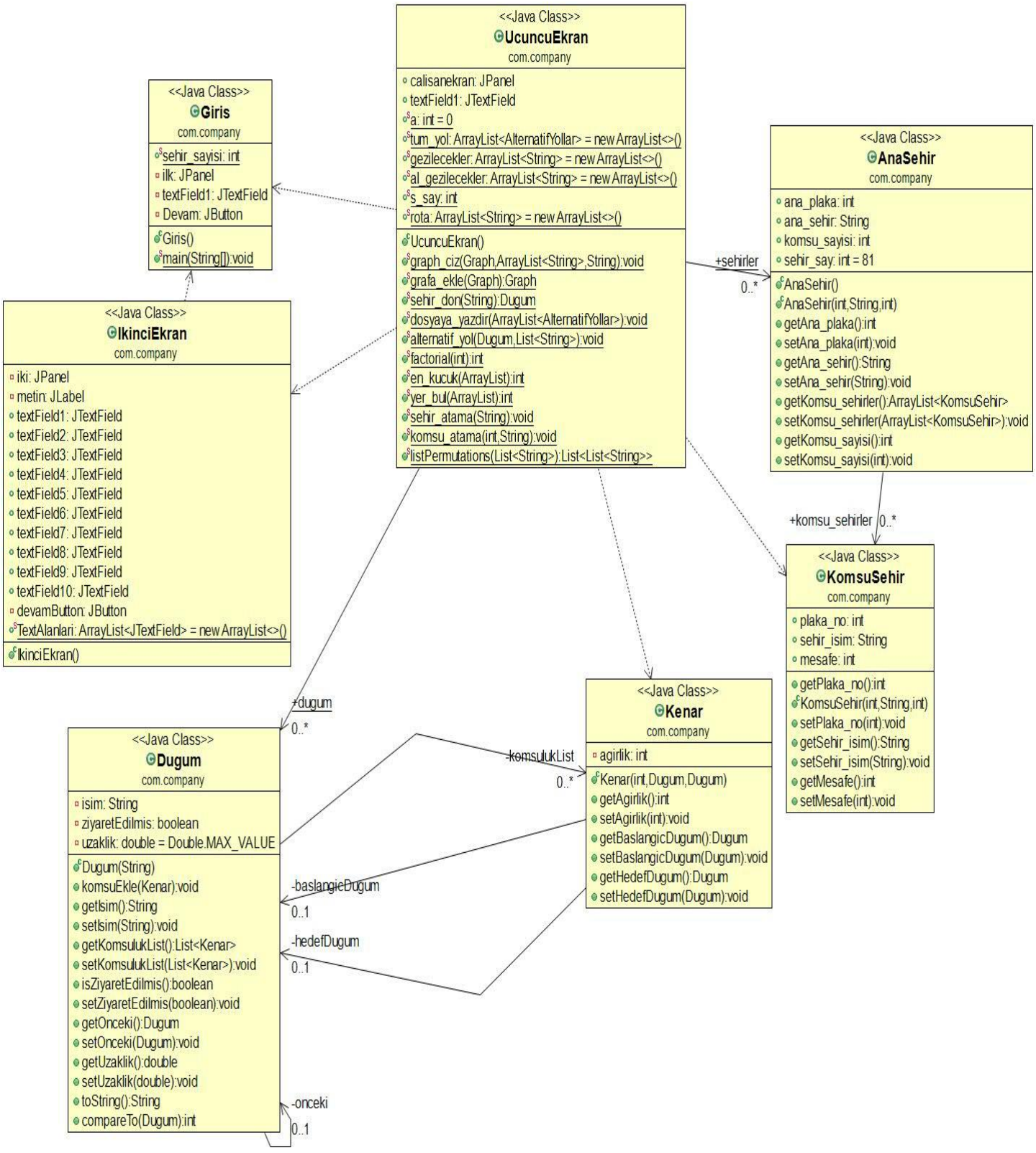
Uzaklık : 5296

5. yol : [Kocaeli, Bursa, Balikesir, Manisa, Aydin, Mugla, Antalya, Mersin, Adana, Osmaniye, Gaziantep, Sanliurfa, Mardin, Sirnak, Hakkari, Van, Agri, Kars, Ardahan, Artvin, Rize, Trabzon, Giresun, Ordu, Samsun, Corum, Cankiri, Bolu, Sakarya, Kocaeli, Istanbul, Tekirdag, Edirne, Tekirdag, Istanbul, Kocaeli, ]



Uzaklık : 5338

# UML Diagramı



## IX. KAYNAKÇA


1) http://graphstream-project.org/doc/Tutorials/Graph-Visualisation/
(Erişim Tarihi : 28.03.2020)

2)https://data.graphstream-project.org/api/gs-core/current/org/graphstream/graph/Graph.html
(Erişim Tarihi: 28.03.2020)

3)http://graphstream-project.org/doc/Tutorials/Storing-retrieving-and-displaying-data-in-graphs/
(Erişim Tarihi : 27.03.2020)

4) http://graphstream-project.org/doc/FAQ/The-graph-viewer/How-do-I-dynamically-change-color-and-size-in-the-viewer/
(Erişim Tarihi : 27.03.2020)

5) http://www.ahmetsayar.com/lecturenotes/
(Erişim Tarihi : 13.03.2020)

6)https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html
(Erişim Tarihi : 25.03.2020)

7) https://www.baeldung.com/java-graphs
(Erişim Tarihi : 22.03.2020)

8)https://www.java67.com/2015/03/how-to-remove-duplicates-from-arraylist.html
(Erişim Tarihi : 20.03.2020)

9)https://www.geeksforgeeks.org/print-all-permutation-of-a-string-using-arraylist/
(Erişim Tarihi : 26.03.2020)

10)https://www.geeksforgeeks.org/arraylist-of-arraylist-in-java/
(Erişim Tarihi : 10.03.2020)

11)https://stackoverflow.com/questions/11071211/convert-jtextfield-input-into-an-integer
(Erişim Tarihi : 23.03.2020)

12)https://www.geeksforgeeks.org/passing-and-returning-objects-in-java/
(Erişim Tarihi : 19.03.2020)

13) https://java2blog.com/dijkstra-java/
(Erişim Tarihi : 20.03.2020)

14)http://graphstream-project.org/doc/Tutorials/GraphStream-Maven/
(Erişim Tarihi : 27.03.2020)