**GEZGİN ZEPLİN PROBLEMİ**

Ali Recep KARACA, Mehmet Fırat KÖMÜRCÜ

Bilgisayar Mühendisliği Bölümü

[recepkaraca@yandex.com](mailto:recepkaraca@yandex.com) , [mehmetfiratkomurcu@hotmail.com](mailto:mehmetfiratkomurcu@hotmail.com)

**Özet**

*Bu proje; başlangıç ve bitiş şehirleri kullanıcı tarafından alınan, iki şehir arasındaki en kısa maliyetli yolu bulmakta olup, bu bulunan yollar üzerindeki şehirlerin tüm komşuları için açı hesabı yapmakta ve bu açıları zeplin içerisindeki yolcu sayısıyla karşılaştırarak ilgili şehre gidilip gidilemeyeceğini hesaplamaktadır. Bu hesaplar neticesinde bizden bu projede istenen; sabit ücret alınarak maksimum kar elde etme problemi ve bir yolcudan ne kadar ücret alınırsa yüzde elli kar elde edileceği problemleri çözümlenmektedir. Yapılan tüm bu adımlar, projede bizden istenen; zeplin aracının özelliklerine göre çözümlenmektedir.*

**1. Problemin Tanımlanması**

Bu projede çözümlenmesi gereken sorun; başlangıç ve bitiş şehirleri kullanıcı tarafından alınan iki şehir arasındaki yolları hesaplayarak, bu yollar içerisinden, proje için bizlerden istenen sabit ücret problemi ve yüzde elli kar probleminin çözülebilmesi adına en uygun olanının seçilip, üzerinde bazı matematiksel işlemler yapılıp, kullanıcıya en iyi yolların harita üzerinde grafiksel olarak gösterilmesi ve diğer mevcut tüm yolların en karlıdan zararlıya doğru sıralanmasıdır.

**2. Giriş**

Verilen projenin konusu; iki şehir arasındaki en kısa yolun bulunması ve bu yolun projede bizlerden istenen sabit ücretle kar ve yüzde elli kar için gereken ücret miktarı problemleri için kullanılarak istenilen sorunlara çözüm bulunulmasıdır.

Projede, enlem ve boylam bilgisi kullanarak iki şehir arasındaki uzaklığı bulma probleminin çözümlenmesi için aşağıdaki “Haversine” adı verilen formülden yararlanılmıştır:

Denklemde arasındaki mesafe bulunacak olan şehirler için; lat1; ilk şehrin enlemi, lat2; ikinci şehrin enlemi, lon1; ilk şehrin boylamı, lon2 ise; ikinci şehrin boylamı anlamına gelmektedir. “toRadians()” ifadesi ise kendinden önceki ifadenin radyan cinsine çevrilmesi gerektiğini ifade etmektedir.

R = 6371000 (1)

φ1 = lat1.toRadians() (2)

φ2 = lat2.toRadians() (3)

Δφ = (lat2-lat1).toRadians() (4)

Δλ = (lon2-lon1).toRadians() (5)

a = sin(Δφ/2) \* sin(Δφ/2) + cos(φ1) \* cos(φ2) \*

sin(Δλ/2) \* sin(Δλ/2) (6)

c = 2 \* atan2(sqrt(a), sqrt(1-a)) (7)

d = R \* c; (8)

Burada (8) numaralı denklemde bulunan d değeri bize, ilk şehrimiz ile ikinci şehrimiz arasındaki mesafeyi vermektedir.

Verilen projenin amacı; kullanıcı tarafından girilen iki şehir arasındaki en kısa yol bulma problemi içerisinde graf yapısı kullanılması gerektiğinden, öğrencilerin daha önce derslerde teorik olarak almış oldukları graf yapısının pratikte bir örnek üzerinde uygulanmasını sağlamak ve öğrencileri bu konu üzerinde deneyim kazanmasını sağlamaktır.

**3. Temel Bilgiler**

Projeyi gerçekleştirme aşamasında Java programlama dili kullanılmıştır. Ayrıca; yapmış olduğumuz proje; Java programlama dili kullanılarak nesneye yönelik programlama mantığında oluşturulmuştur.

Yazılan bu program Windows 10 üzerindeki NetBeans 8.2 derleyicisi üzerinde derlenmiş olup, derleme aşamasında herhangi bir hatayla karşılaşılmamıştır. Ayrıca derlenen bu program yine Windows 10 üzerinde çalıştırılmış ve programın çalışması sırasında yine herhangi bir hatayla karşılaşılmamıştır.

**4. Genel Yapı**

Yaptığımız projede ilk olarak; kullanıcıdan arasındaki mesafe ve yollar bulunmak üzere başlangıç ve varış şehirlerinin plaka kodları alınmaktadır. Ardından, projeyi geliştirme aşamasında bizlere verilmiş olan komşuluk dosyasından bilgiler programa aktarılmış olup 81x81 boyutunda bir komşuluk matrisi oluşturulmuştur. Burada; komşuluk dosyasından aktarılan veriler ışığında, komşu olan şehirlerin ilgili satır ve sütun değerleri “1”; komşu olmayan şehirlerin satır ve sütün değerleri “0” olarak atanmıştır. Ardından, yine bize proje geliştirme aşamasında sunulmuş olan enlem ve boylam bilgileri programa bir matris biçiminde aktarılmış olup ileride kullanılmak üzere depolanmıştır.

Proje geliştirme aşamasında bizlere dosya olarak sunulmamış olsa da, projedeki plaka kodlarının kullanıcı tarafından daha iyi anlaşılabilmesi adına, internet üzerinden tüm plaka kodlarının hangi şehre karşılık geldiği bulunmuş olup, bulunan bu şehirler bir “text” dosyasına yazdırılmış ve program içerisine bir matris olarak dahil edilmiştir.

Projede yukarıda bahsedilmiş olan komşuluk matrisi üzerinde; iki şehrin komşu olduğunu anlamak için kullanmış olduğumuz “1” değerleri bize proje geliştirme aşamasında sunulmuş olunan enlem ve boylam bilgileri kullanılarak, daha önce yine bu dokümanda bahsedilmiş olunan “Haversine” formülü kullanılarak uzaklık bilgilerine dönüştürülmüş olup, ileride bu uzaklıklar kullanılarak en kısa yol bulma probleminin çözümlenmesi amaçlanmıştır.

Yukarıdaki tüm işlemler sonunda elde edilmiş olunan komşuluk matrisi; zeplin içerisine binen her yolcu, zeplinin dönüş derecesini 1 derece düşürmekte olduğundan, aradaki şehirlerin açılarının zeplinin gidebileceği açı aralığında olup olmadığına bakılması adına “hesaplar” isimli fonksiyona gönderilmiştir. Yine bu “hesaplar” fonksiyonu içerisinde; şehirlerin sadece yatay uzaklığı alınmaması adına, yüksekliğin de uzaklık hesabı içerisine katılması sağlanmıştır. Bu işlem 5 kişi den başlayarak 50 kişi ye kadar (Bu değerler bizlere proje kapsamında verilmiştir.) tüm yolcu sayıları için ayrı ayrı hesaplanmaktadır.

“hesaplar” fonksiyonu içerisinde ilk olarak; komşuluk matrisi üzerindeki tüm komşu şehirler için, şehirlerin rakım değerlerine bakılarak yükseklik farkı bilgisi bulunmaktadır. Ardından bu yükseklik farkı ve şehirler arasındaki yatay mesafe farkı kullanılarak şehirlerin arasındaki gerçek mesafe farkı bulunmaktadır. Burada; eğer şehirler arasındaki açı, fonksiyon içerisine gönderilmiş olan kişi sayısı kullanılarak hesaplanan zeplinin dönüş açısından büyük bulunursa bu iki şehir gidilemez olarak işaretlenmektedir, ardından komşuluk matrisi üzerine uzaklık bilgisi silinerek, yerine “0” değeri yazılarak iki şehrin komşu olması engellenmiş olmaktadır. Ardından; zeplinin şehrin 50 metre yüksekte gitmesi gerektiğinden, kalkış, varış gibi durumlarda zeplinin hangi şehirlere inebileceği ve kaç derece açıyla inmesi gerektiği yine bu fonksiyon içerisinde hesaplanmaktadır. Bu fonksiyon içerisinde yapılan tüm hesaplar “Main” fonksiyonu içerisine gönderilerek ana komşuluk matrisinin bu hesaplardan etkilenmesi sağlanmaktadır.

Komşuluk matrisi, “Dijkstra” isimli sınıf içerisindeki, yine aynı isimli “dijkstra” isimli fonksiyona gönderilmiştir. Burada ilk olarak; şehirler arasındaki tüm yollar maliyet matrisine; eğer yol “0” değeriyse “9999” yani ulaşılamaz değeri olarak, eğer değilse graf içerisindeki sayı direkt olarak maliyet matrisine aktarılmaktadır. Daha sonra bu maliyet matrisi kullanılarak, şehirler arasındaki tüm olası yollar bulunmaya çalışılmaktadır. Ardından bulunan bu yollar içerisinden en kısa yol seçilerek “yolUzaklik” isimli matrise aktarılarak bu matris geri “main” fonksiyonuna gönderilmektedir. Tüm bu işlemler sırasında en kısa yol bulma algoritmalarından birisi olan “Dijkstra” algoritması kullanılmıştır.

Tüm bu işlemler sonrasında, sabit ücret probleminin çözümlenmesi adına; “ConstantPaymentProblem” isimli sınıftan bir nesne üretilmiştir ve bu nesne içerisinde bulunan “constantPayment” metodu bu nesne üzerinden çağrılmıştır. Bu fonksiyon içerisinde kişi sayısı, kişi başı ücret ve şehirler arasındaki toplam mesafe bilgileri kullanılarak gelir ve gider hesapları yapılmıştır. Hesaplanan tüm kar değerleri bir diziye atılarak bu yollar arasında sıralama yapılması amaçlanmıştır ve bulunan bu yollar konsol ekranında en karlıdan zararlıya doğru sıralanmıştır ve hangi yollardan geçileceği, toplam mesafe gibi bilgiler de kullanıcıya verilmiştir. Bulunan bu yollar arasından en karlı olanı ise grafiksel olarak gösterilmesi sağlanmıştır. Bunun için “Java” programlama dili içerisinde bulunan “JFrame” özelliğinden yararlanılmıştır. “JFrame” formu içerisine Google Haritalar servisini kullanmakta olan rota bilgisi eklenmiştir. Burada; Google Haritalar servisi kullanılırken “Google Static Maps” servisinden yararlanılmıştır. Ardından; bulunan bu yol üzerindeki tüm şehirler arasındaki mesafeler, şehirlerin enlem, boylam gibi bilgiler “constant\_payment.txt” isimli dosyaya yazılmaktadır.

Yine yukarıda olduğu gibi, projede bizlerden istenen bir diğer problem olan yüzde elli kar problemi için ilk olarak; problemin çözümlenmesi adına ilk olarak giderin yüzde ellisi alınarak ve bu sayı gider ile toplanarak elde edilmesi gereken kar miktarı bulunmuştur. Burada bir denklem kurularak yolculardan kişi başı kaç lira alınırsa yüzde elli kar edileceği bulunmuş olup bulunan tüm bu değerler bir diziye aktarılarak depolanmıştır. Bulunan tüm bu karlar içerisinden en iyi sonucu veren değerler küçükten büyüğe sıralanarak konsol ekranında kullanıcıya gösterilmiştir. Sonuçlar arasında en iyi olanı ise yine “Java” programlama dili içerisinde bulunan “JFrame” içerisine Google Haritalar hizmeti kullanılarak görsel olarak kullanıcıya gösterilmiştir. Ayrıca yine bu ekranda kaç kişiden kaç para ücret alınması gerektiği de kullanıcıya gösterilmiştir. Yine burada da, bulunan en iyi yol için; aradaki tüm şehirlerin birbirlerine uzaklıkları, enlem ve boylam gibi bilgileri “fifty\_percent.txt” isimli bir dosyaya yazdırılmıştır.

Tüm bu problemler için; plaka kodları kullanıcı için karmaşık gelebileceğinden dolayı, oluşturmuş olduğumuz plaka-şehir dosyasından çekilen tüm şehir isimleri, plaka kodları yerine kullanılmış olup, kullanıcının şehir isimleriyle muhatap olması sağlanmıştır ve program daha kullanılabilir bir seviyeye getirilmiştir.

Ayrıca tüm program içerisinde, her sınıfın ne kadar süre çalıştığı “main” metodu içerisinde kullanıcıya gösterilmiş olup, programın hızı hakkında bilgi edinilmesi amaçlanmıştır. Projedeki bu süre hesaplama kısmında, kullanıcının giriş yaparken harcadığı sürenin gerçek verileri saptırmaması adına, kullanıcı giriş yaparken geçen süreler programın çalışma süresi içerisine dahil edilmemiş olup, programın daha doğru bir çalışma süresi vermesi sağlanmıştır.

**5. Proje Hakkında Diğer Veriler**

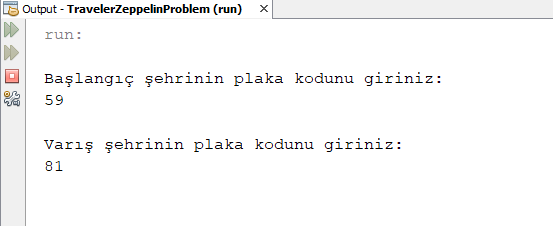
**5.1 Fonksiyonlar ve İşlevleri**

|  |  |
| --- | --- |
| Fonksiyon | İşlevi |
| Main | Kullanıcıdan başlangıç ve bitiş şehirlerini almak, komşuluk matrisinin elemanlarını dosyadan çekmek, şehirler arasındaki yatay uzaklığı hesaplamak, diğer fonksiyonların çalışmalarını sağlamak, ekranda diğer fonksiyonların çalışma sürelerini göstermek |
| hesaplar | Main fonksiyonu tarafından gönderilmiş olan komşuluk matrisi üzerinde işlemler yaparak, zepline binecek olan kişi sayısına göre zeplinin gidip gidemeyeceği şehirleri bulmak, ayrıca zeplinin kalkışı ve inişi sırasında yapılması gereken gerekli hesaplamaları yaparak varış ve kalkış yapılabilecek şehirleri belirlemek |
| dijkstra | Kullanıcı tarafından girilen şehirler arasındaki en kısa yolu bulmak adına; tüm olası yolları inceleyerek, komşuluk matrisindeki değerleri dikkate alarak kullanıcının girmiş olduğu bu iki şehir arasındaki en kısa yolu bulmak, bu yolu geri “main” fonksiyona göndermek |
| constantPayment | Projede çözümlenmesi gereken bir problem olan sabit ücret problemini çözümlemek, bunun için gider-gelir hesabı yapmak, bu problemin çözümü için bulunan bütün yolları en karlıdan zararlıya doğru sıralayarak kullanıcıya göstermek ve en iyi yolu grafiksel olarak ekrana basmak |
| fiftyPercent | Projede çözümlenmesi gereken bir problem olan yüzde elli kar probleminin çözümlenmesi için gerekli olan denklemleri çözümlemek, çözümlenen bu denklemler sonrasında açığa çıkan en iyi yolları en iyiden en kötüye sıralayarak kullanıcıya göstermek, yollar arasında bulunan en iyi yolu grafiksel olarak ekrana basmak |

*Tablo 1*: Fonksiyonlar ve İşlevleri

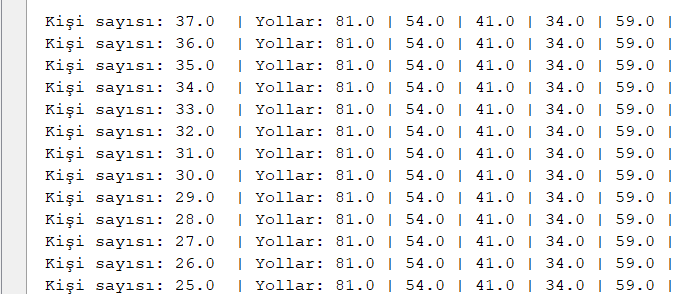
**5.2 Programın Çalışması Sırasından Bazı Görseller**

Kullanıcının şehir girişi:



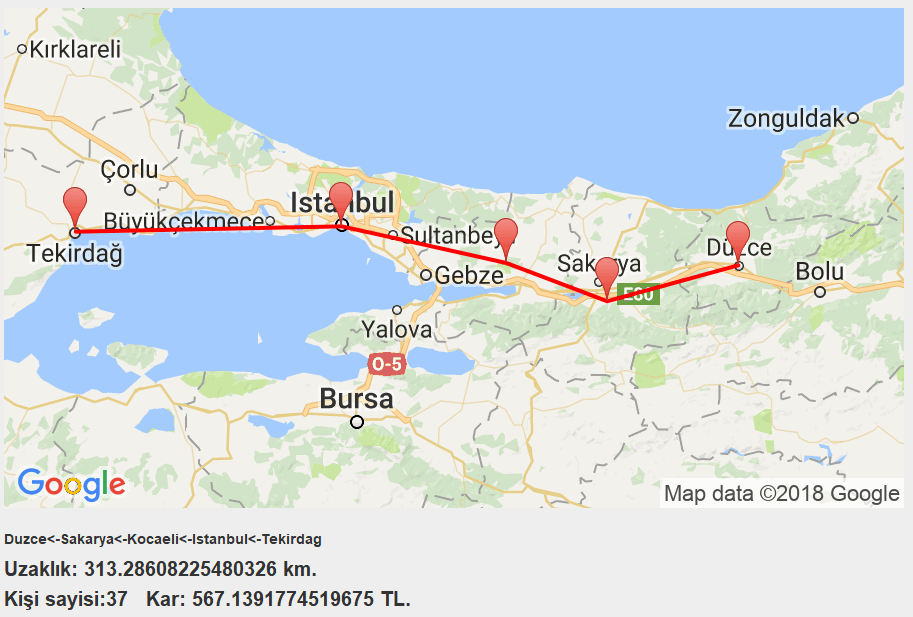
*Şekil 1:* Kullanıcı Veri Girişi

Konsol üzerinde yolların gösterilmesi:



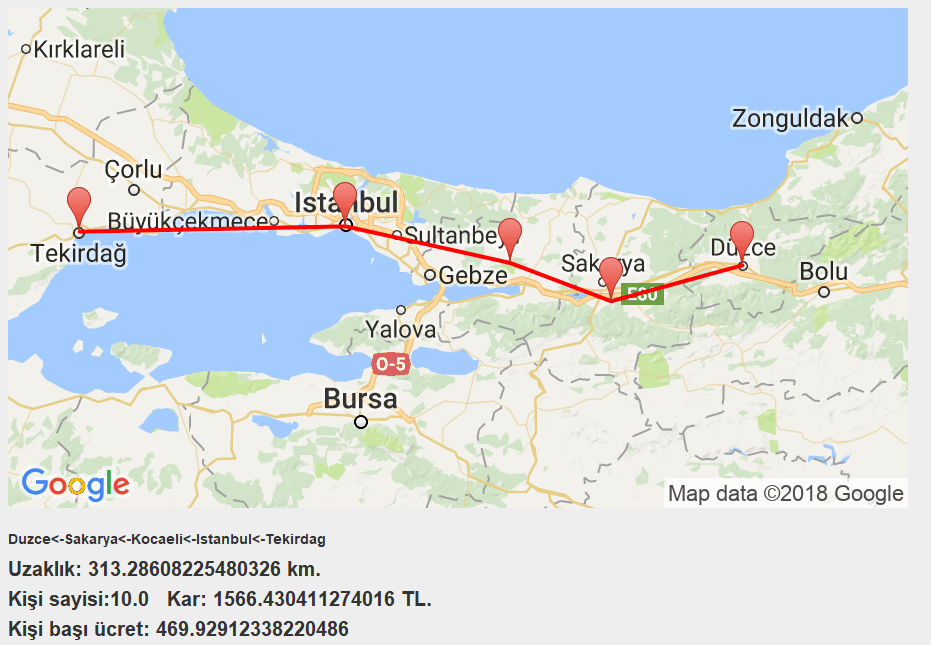
*Şekil 2:* Yolların Gösterilmesi

Sabit ücret probleminin grafiksel olarak gösterilmesi:



*Şekil 3*: Grafiksel Gösterim 1

Yüzde elli kar probleminin grafiksel gösterilmesi:



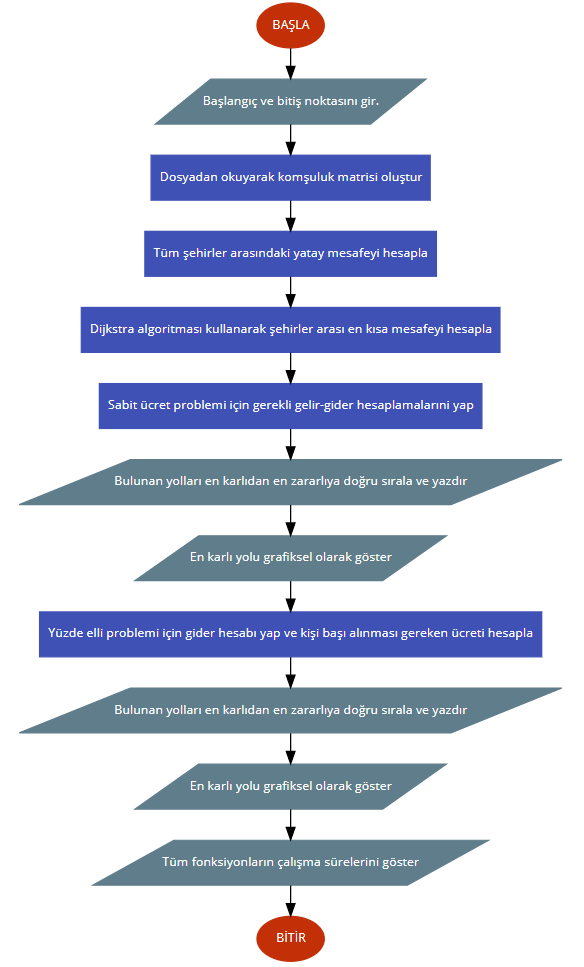
*Şekil 4*: Grafiksel Gösterim 2

**6. Sonuçlar**

* Türkiye’nin şehirlerini içeren latitude, altitude (rakım) ve komşuluk bilgilerini içeren bir dosya hazırlanmıştır. (lat long altitude bilgileri için verilen dosyadan yararlanılmıştır.
* Her şehrin komşu şehrine gidip gidemeyeceği tespit edilmiştir. Bunun tespiti için kendi rakım bilgisi değerlendirilerek eğim (derece) kriterini sağlayıp sağlamadığına bakılmıştır. Eğim hesaplanırken yükseklik değeri metre, yatay mesafe kilometre olarak alınmıştır.
* Yol uzunlukları yükseklik bilgisi kilometreye dönüştürülerek hesaplanmıştır.
* Her şehir düğüm, gidilebilir komşu düğümler ve hesaplanan ağırlık değeri bağlantıların kenarı olacak şekilde graf yapısı tanımlanmıştır. Geliştirilen algoritma graf üzerinde çalışacak şekilde ayarlanmıştır. Algoritma olarak Dijkstra en kısa yol bulma algoritmasından faydalanılmıştır.
* Taşınabilir yolcu sayısına göre eğim değişeceği için graf yapısında komşuluklar ve ağırlıklar değişebilmektedir. Bu sebeple yolcu sayısına göre graf yapısı değiştirilebilmektedir.
* Graf için girdi olarak başlangıç ve bitiş şehri seçilmektedir. Seçilen şehirler arasında graf üzerinde en kısa yol algoritması işletilerek yolcu sayısına göre farklı çözümler sunulmaktadır.
* Konum ve güzergâh bilgileri ayrı ayrı harita olarak gösterilmektedir.
* Sabit ücretle kaç yolcu ile sefer düzenlenirse en çok kârı yapabilirim probleminde; 5 ile 50 arasındaki yolcu sayıları kadar olan yolcu sayısı için en iyi çözümü veren yol (harita üzerinde) ve girdi olarak alınan şehirler arasındaki toplam yollar ayrı ayrı çıktı olarak verilmektedir. Çıktı üzerinde maksimum kâr hesaplanarak en iyi sonuca göre sıralanmaktadır.
* Bir yolcudan kaç TL alınırsa yüzde 50 kâr yapılabilir probleminde; 10, 20, 30, 40, 50 yolcu sayıları için en iyi çözümü veren yol (harita üzerinde) ve girdi olarak alınan şehirler arasındaki toplam yollar ayrı ayrı çıktı olarak verilmektedir. Çıktı üzerinde kâr miktarı yüzde 50 olacak şekilde bir yolcudan alınması gereken para miktarı hesaplanmaktadır.
* Her iki problem için lat, long ve komşular arasındaki mesafeler bir dosyaya yazılmaktadır.
* Program içerisinde çalışma süreleri gösterilmektedir.

Yukarıdaki değerlendirmeler ışığında, yapmış olduğumuz program, proje sırasında bizlerden istenen tüm adımları yerine getirmekte olup, hiçbir eksik fonksiyonu bulunmamaktadır. Program, belirlenen hedefe tam anlamıyla ulaşmaktadır.

**7. Akış Şeması**



*Şekil 5*: Akış Şeması

**8. Kaynakça**

<http://bilgisayarkavramlari.sadievrenseker.com/2010/05/13/dijkstra-algoritmasi-2/> (Access Date: 26.02.2018)

<https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm> (Access Date: 26.02.2018)

<https://www.tutorialspoint.com/java/java_files_io.htm> (Access Date: 26.02.2018)

<https://gelecegiyazanlar.turkcell.com.tr/konu/android/egitim/android-101/javada-dosya-islemleri> (Access Date: 26.02.2018)

<https://www.tutorialspoint.com/swing/swing_jframe.htm> (Access Date: 03.03.2018)

<https://www.youtube.com/watch?v=jUdIAgJ7JKo> (Access Date: 03.03.2018)

<https://www.youtube.com/watch?v=Zh_CJxYi0wU> (Access Date: 03.03.2018)

<https://www.movable-type.co.uk/scripts/latlong.html> (Access Date: 03.03.2018)

<https://en.wikipedia.org/wiki/Haversine_formula> (Access Date: 03.03.2018)