GEZGİN ZEPLİN PROBLEMİ

160201079 İlknur GÖK,

160201012 Elanur OCAK

Bilgisayar Mühendisliği Bölümü

Kocaeli Üniversitesi

[ilknurgok.1@hotmail.com](mailto:ilknurgok.1@hotmail.com) , [elaocak45@gmail.com](mailto:elaocak45@gmail.com)

**Özet**

Bize verilen proje kapsamında zeplin aracı ile yapılabilecek olan şehirlere gezi işlemini en kısa yol algoritmalarından herhangi birini kullanarak problemi çözmemiz istenmiştir. Projede ilk olarak, projeyi gerçekleştirmemiz için verilen içinde şehirlere ait bilgilerin olduğu dosyalardan okuma yaptık. Sonra zeplin ile kalkış ve de iniş yapılması istenen şehirlerin plakalarını kullanıcıdan istedik. Sonrasında ise çözülmesi gerekli olan iki problemlerden hangisinin çözümünün bulunmasına dair kullanıcıya bir seçim sunduk. Seçilen probleme göre çözümünü ekrana yazdırdık. Bu çözümü yaparken de en kısa yol algoritmalarından olan Dijkstra algoritmasını kullandık.

**1.Giriş**

Projenin genel olarak konusu, zeplin aracının en düşük maliyet ve en yüksek karla ulaşımını sağlamak için en kısa yol algoritmasını (Dijkstra Algoritması) kullanmaktır.

Projenin amacı ise veri yapıları ve veri modellerini anlaşılması, graf yapısını kullanılmasını sağlayabilmektir.

Dijkstra algoritması, **ağırlıklı bir graf üzerinde** yani kenarları (edge) belli bir metrik değere göre değerleri olan herhangi iki düğüm arası en kısa mesafeyi bulmamızı sağlayan bir algoritmadır.

Algoritma, Hollandalı matematikçi ve bilgisayar bilimci **Edsger Wybe Dijkstra** tarafından bulunmuştur.

**2.Temel Bilgiler**

Proje C dilinde yazıldı ve CodeBlocks’ta hazırlandı. Dosya oluşturma kısmı Notepad’de tamamlandı.

* Code::Blocks, özgür açık kaynak kodlu bir C++ tümleşik geliştirme ortamıdır. wxWidgets tabanlı tamamen özelleştirilebilir arabirimiyle, GNU/Linux, Microsoft Windows, MacOS platformlarında sorunsuzca kullanılabilmektedir.
* Notepad++, Windows işletim sistemi içerisine gömülü olarak gelen Notepad yazılımının yerine kullanılmak üzere C++ ile saf Win32 API ve STL ile geliştirilmiş GPL ile dağıtılan açık kaynak kodlu bir kaynak kod düzenleyicisidir.

**3.Diğer Bölümler**

*3.1. Şehir bilgilerini edinme*

Projemizde ilk olarak proje ile birlikte verilen şehirlere ait lat, long, plaka kodu ve rakım bilgilerini içeren ve her şehrin komşu şehirlerinin plaka kodlarını içeren iki dosyayı okutup her şehre ait bu özelliklerin bir struct yapısına atamasını yaptık.

*3.2. Başlangıç ve bitiş şehri alma*

Zeplin aracının yolculuk yapmaya başlayacağı başlangıç ve bitiş şehrini kullanıcıdan istedik. Doğru bir plaka girişinin yapılabilmesini de kontrol ettirdik.

Sonrasında her şehrin gidilebilecek olan komşu şehirleri arasındaki lat ve long farklarını kullanarak distance dediğimiz değeri Haversine formülü ile hesapladık. Bunu yaparken aynı zamanda her şehir ile komşu şehri arasındaki rakım farkını da bularak iki şehir arasında zeplin ile yolculuk yapılıp yapılmayacağını belirleyen eğim açılarını da hesapladık ve bunları da yine bir struct yapısı ile tuttuk. Bu işlemlerden önce aldığımız başlangıç şehrine ise eğim açısını hesaplarken, kalkışta zeplinin 50 metre yükselmesi gerektiği için komşu şehirleri arasında hesapladığımız rakım farkına 50 daha ekleme yaptık.

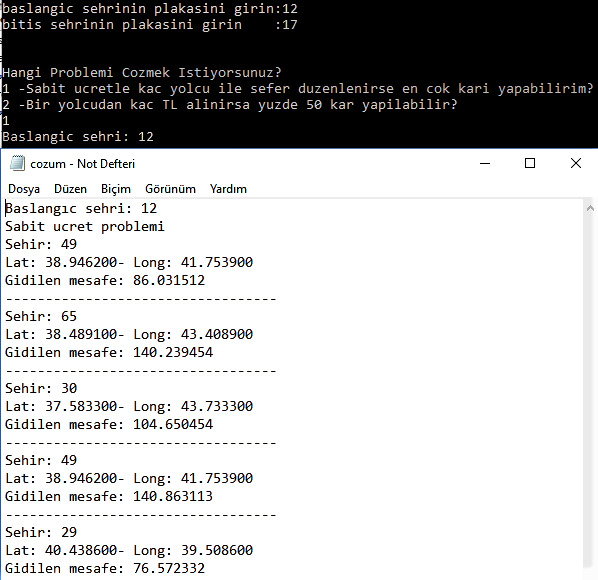
*3.3. Problemleri çözme*

Sonrasında kullanıcıdan çözülmesini istediği problemi seçmesini istedik. Seçilen probleme göre her probleme ait olan yazmış olduğumuz fonksiyonlara geçiş yaptık.

*3.3.1. Sabit ücret problemi fonksiyonu*

Kullanıcının seçtiği problem sabit ücret alındığında kaç yolcu ile sefer düzenlenirse maksimum kar elde edilir olduğunda SabitUcretProblemi() isimli fonksiyona geçiş yaptık. Buraya önceden kullanıcıdan aldığımız başlangıç ve bitiş şehirlerinin plakalarını gönderdik. Problemin çözümünü dosyaya kaydedebilmek için fonksiyonda ilk olarak dosya açma işlemini yaptık. Sonrasında yolcu sayısını 5 ile 50 arasında tutarak, her yolcu sayısına göre bulduğumuz eğimi şehirler arasında yol bulabilmek amacıyla oluşturduğumuz rotaHesaplaniyor() isimli fonksiyona başlangıç ve bitiş şehir plakları ile gönderdik. Bu fonksiyondan her yolcu sayısına göre bulmuş olduğumuz gidilen yol değerlerinin toplamını bulduk. Bu toplama göre zeplinin harcadığı maliyeti ve de yolcu sayısına göre sabit ücret ile yolcu maliyetini hesapladık. Bulduğumuz bu değerlere göre her yolcu sayısı için bir kar oranı hesapladık ve bu oranları ise bir struct yapısında tuttuk.

Hesaplanan bu kar oranlarını büyükten küçüğe doğru sıralayıp ekrana maksimum kar yapıldığındaki yolcu sayısını ve bu yolcu sayısına göre oluşan eğim ile gidilen şehir ve bilgilerini yazdırdık. Sonrasında sıralı bir şekilde diğer kar oranlarına göre de gidilen şehir ve bilgilerini ekrana yazdırdık. En sonunda ise fonksiyonun başında açmış olduğumuz dosyayı kapattık.

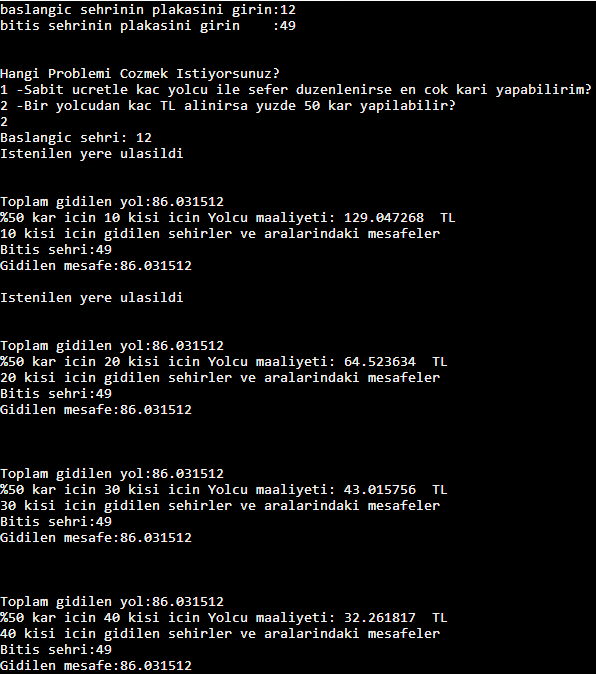


Şekil 1:Sabit Ücret Problemi Örnek Çıktı

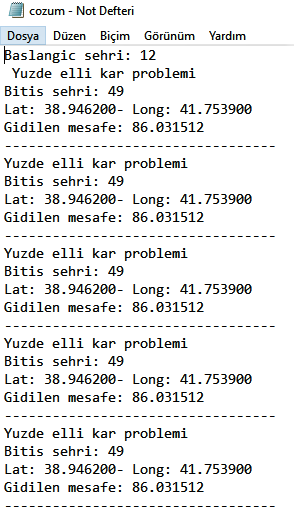
*3.3.2. Yüzde elli kar problemi fonksiyonu*

Kullanıcının seçtiği problem bir yolcudan kaç TL alınırsa yüzde 50 ara ulaşılır

problemi olduğunda YuzdeElliKarProblemi() isimli fonksiyona geçiş yaptık. Buraya önceden aldığımız başlangıç ve bitiş şehirlerinin plakalarını gönderdik. Çözümü kaydedebilmek için fonksiyonun başında ilk olarak dosya açma işlemini yaptık. Sonrasında 10,20,30,40 ve 50 sayıdaki yolcu saylarına göre her yolcu sayısına göre bulduğumuz eğimi önceden kullanıcıdan almış olduğumuz başlangıç ve bitiş şehrinin plakalarını rotaHesaplaniyor() isimli fonksiyona gönderdik. Her yolcu sayısına göre gidilen yolların toplamını bulduk. Bu toplam ile zeplinin harcadığı maliyeti hesapladık. Bu maliyet ile %50 oranında bir kar yapabilmek için her yolcu başına alınması gereken yolcu maliyetini hesapladık. Her yolcu sayısına göre (10, 20, 30, 40 ve 50) bulduğumuz bu maliyetleri ekrana yazdırdık. En sonunda ise fonksiyonun başında açtığımız dosyayı kapattık.



Şekil 2: Yüzde Elli Kar Problemi Örnek Çıktı

**

Şekil 3: Örnek Dosya Kaydı

*3.4. rotaHesaplaniyor() fonksiyonu*

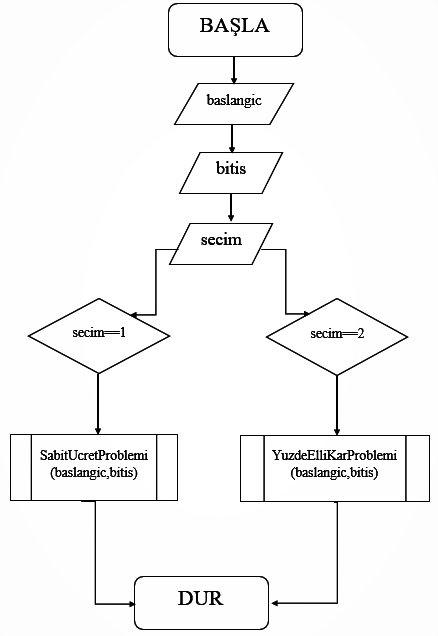
Kullanıcıdan aldığımız başlangıç ve bitiş şehirleri arasındaki yolları hesaplamak için SabitUcretProblemi() ve YuzdeElliKarProblemi() isimli fonksiyonlarda kullandığımız fonksiyondur.

Bu fonksiyonun başında yine gidilen şehirleri ve bilgilerini kaydedebilmek için dosya açma işlemini yaptık. Sonrasında fonksiyona gönderdiğimiz başlangıç şehrinin komşularını, yine fonksiyona gönderdiğimiz eğim ile kontrol ettirdik.

Komşu şehri ile arasındaki eğim, zeplin aracının eğiminden büyük veya eşit olduğu durumda hangi eğimi uygun olan şehir ile arasındaki uzaklık en kısa ise bir sonraki şehri o şehir yaptık. Bunu yaparken bu gidilecek olan şehri ise d isimli diziye, bu iki şehir arasındaki yolu path isimli diziye attık. Aynı zamanda ise push() fonksiyonuna gidilecek olan şehri atıp linked yapısı ile graf oluşturmaya çalıştık. Sonrasında bu şehir kullanıcıdan önceden aldığımız bitiş şehrine eşitse programı sonlandırdık değilse bu şehri başlangıç şehri olarak alıp tekrar rotaHesaplaniyor() fonksiyonuna gönderdik.

Eğer eğim uygun değilse veya grafControl() isimli fonksiyonla yaptığımız kontrol sonrasında (yol bulma yaparken gittiğimiz şehirleri attığımız linked list üzerinde o şehrin olup olmadığını sorgulama işlemi) o şehir varsa eğer bu şehre tekrar gidilmesini engellemek için bu şehir ile öncesinde bu şehre gelinmesini sağlayan şehri egimSifirla() isimli fonksiyona gönderdik. Burada bu iki şehir arasında tekrar tekrar gidip gelmeyi engellemek için her iki şehir arasındaki eğim değerini sıfıra eşitledik. Sonrasında başlangıç şehrinin başka bir komşusunu kontrol ettirmek için döngünün tekrar başına gönderip diğer komşuları arasında bir kontrol işlemi gerçekleştirdik.

**4.Akış Şeması**



**5. Sonuçlar**

Projemizde genel olarak bizden en kısa yol algoritması ve graf yapısını kullanmamız istendi.

Projemizi oluştururken en başta şehirler ile ilgili bilgilerin ve komşularının bulunduğu dosyaları okurken ve de bu bilgileri struct yapısına atama yapmayı sorunsuz yaptık.

Zeplin ile ilgili çözülmesi gereken iki adet problem için ayrı ayrı fonksiyonları doğru bir şekilde oluşturduk fakat SabitUcretProblemi() fonksiyonunda her yolcu sayısı için gidilen yolları ekrana yazdırırken bir sorun yaşadık.

Ayrıca rotaHesaplaniyor() fonksiyonunda bitiş şehrine gelindiğinde fonksiyonu durdurmayı ve de bazı şehirler için önceden gidilmiş olan şehirlere tekrar gidilmemesini engellemeye başaramadık.

**6.Kaynakça**

[1]. Web Site

<https://www.programmingsimplified.com/c/source-code/c-program-bubble-sort>

[2]. Web Site

<https://www.orkunucan.com/2015/05/19/haversine-formulu-ile-iki-konum-arasindaki-mesafeyi-hesaplama/>

[3]. Web Site

<http://www.programmingworldtech.com/2014/03/c-program-for-shortest-path-using.html>

[4]. Web Site

<http://ensarkarabudak.com/algoritma/dijkstra-en-kisa-yol-algoritmasi-dijkstras-shortest-path-algorithm/>