T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

GRAF TEORİSİNE DAYALI WEB ARAYÜZLÜ YOL PROBLEMİ UYGULAMASI

(Yüksek Lisans Tezi)

Tezi Hazırlayan: **Vural TARIM**

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ BİLİM DALI

GRAF TEORİSİNE DAYALI WEB ARAYÜZLÜ YOL PROBLEMİ UYGULAMASI

(Yüksek Lisans Tezi)

Tezi Hazırlayan:
Vural TARIM
Öğrenci No:
050820015

Danışman:
Dr. Rifat ÇÖLKESEN

YEMİN METNİ

Sunduğum Yüksek Lisans Tezimi, Akademik Etik İlkelerine bağlı kalarak, hiç kimseden akademik ilkelere aykırı bir yardım almaksızın bizzat kendimin hazırladığına and içerim.

2../8.../12.../2007

Aday: Vural TARIM

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜ
TEZLİ YÜKSEK LİSANS TEZ SINAV TUTANAĞI

02/08/2007

Enstitümüz Bilgisayar Mühendisliği Anabilim dalı Bilgisayar Mühendisliği Bilim dalı yüksek lisans öğrencilerinden 050820015 numaralı Vural TARIM'a "Beykent Üniversitesi Lisansüstü Eğitim - Öğretim ve Sınav Yönetmeliği"nin ilgili maddesine göre hazırlayarak, Enstitümüze teslim ettiği **"Graf Teorisine Dayalı Web Arayüzlü Yol Problemi"** tezini, Yönetim Kurulumuzun 11.05.2007 tarih ve 2007/06 sayılı toplantısında seçilen ve Fakülte binasında toplanan biz jüri üyeleri huzurunda, ilgili yönetmeliğin (c) bendi gereğince (60) dakika süre ile aday tarafından savunulmuş ve sonuçta adayın projesi hakkında **oyçokluğu/~~oybirliği~~** ile **Kabul/~~Red veya Düzeltme~~** kararı verilmiştir.

İşbu tutanak, 1 nüsha olarak hazırlanmış ve Enstitü Müdürlüğü'ne sunulmak üzere tarafımızdan düzenlenmiştir.

DANIŞMAN
Dr. Rifat ÇÖLKESEN

ÜYE
Prof.Dr. Esat HAMZAOĞLU

ÜYE
PROF.DR. ALİ OKATAN

Tutanağı Tanzim Eden
Juri Başkanı
Dr. Rifat ÇÖLKESEN

TEŞEKKÜR

Tezimin konusunun belirlenmesi ve hazırlanmasında bana yardımlarını esirgemeyen Sayın Dr. Rifat ÇÖLKESEN'e ve aileme teşekkür ederim.

Vural TARIM

GRAF TEORİSİNE DAYALI WEB ARAYÜZLÜ YOL PROBLEMİ UYGULAMASI

Tezi Hazırlayan: Vural TARIM

Özet

En kısa yol problemi, yöneylem araştırması, bilgisayar bilimleri ve yapay zeka gibi alanlarda en önemli problemlerden biridir. Bir çok optimizasyon problemi, en kısa yol problemi olarak formüle edilebilir. Bu çalışmada, problem graf teorisi aracılığıyla, araştırılıp çözüm yöntemleri incelenmiştir. Çalışmanın son bölümünde Türkiye karayolları için, çözüm algoritmalarından Dijkstra ile, Web Tabanlı bir ASP.Net uygulaması yazılmıştır. Ayrıca Google Maps API yardımıyla bulunan çözüm görselleştirilmiştir

Anahtar Kelimeler: Graf, En kısa yol, Türkiye karayolları, Dijkstra.

A WEB-BASED PATH APPLICATION ON GRAPH ALGORITHMS

Presented by: Vural TARIM

Abstract

It can be said that the shortest path problem is one of the most important generic problems in fields such as operations research, computer science and artificial intelligence. Any general combinatorial optimization problem can be formulated as a shortest path problem. In this thesis, the problem and its solution ways are studied by graph theory. In the last section, a web based application is implemented for Turkish highways. Also for presentation layer, Google Maps API is used for showing route paths.

Key Words: Graph, The Shortest Path, Turkey Highway Map, Dijkstra.

İÇİNDEKİLER

V.BÖLÜM
EN KISA YOL PROBLEMİ UYGULAMASI

GİRİŞ

En kısa yol problemi, bir noktadan başka bir noktaya giderken en kısa yolun tercih edilmesi böylece maliyetin azaltılmasını amaçlayan bir yaklaşımdır. Bilişim teknolojilerinin gelişmesiyle, müşteri taleplerinin zamanında ve en az maliyetle karşılanması problemi, bu yaklaşımın etkinliğini arttırmış, bu nedenle de pek çok uygulaması geliştirilmiştir.

Bir çok problem, düğüm ve kenarlardan oluşan graflar yardımıyla modellenebilir. Şehirler arasındaki uçuşların gösterildiği hava yolu sistemi, karayolu ulaşım sistemi buna örnek olarak gösterilebilir. Eğer problem uzaklıklarla ilgiliyse yollar kilometre birimiyle, kar-zarar gibi değerleri içeriyorsa yollar para birimi ile isimlendirilir. Bunlara *ağırlıklı graf* denir. Bir çok problem ağırlıklı graflarla temsil edilebilir. En kısa yol problemi de ağırlıklı garflara bir örnektir.

Algoritmik olarak en kısa yol probleminin çözümü için kullanılan ilk algoritma Alman bilim adamı Dijkstra tarafından 1959 da bulunmuştur. Algoritmanın temel yaklaşımı, başlangıç düğümü ile birinci düğümün arasındaki en kısa yolun bulunması daha sonra başlangıç ile ikinci düğüm arasındaki en kısa yolun bulunması böylece tekrarlı olarak başlangıç ile diğer tüm düğümler arasındaki en kısa yolların bulunmasına kadar devam eder. Dijkstra dan sonra Bellman-Ford, Floyd–Warshall gibi pek çok algoritma geliştirilmiş ve çalışılmıştır.

I. BÖLÜM

1.1. Kavramlar ve Tanımlar

Graflar Euler (Königsberg Köprü Problemi), Kirchoff (elektrik ağları) ve Cayley (organik kimyasal izomerlerin sıralanması) tarafından değişik alanlarda keşfedilmiştir. Graflar, bilgisayar bilimlerinde kullanılan tümleşik yapılardır. *Listeler, ağaçlar, yönlü çevrimsiz graflar, akış diyagramları, kontrol akış grafları* ve *düzlemsel graflar* bilgisayar bilimlerinde geniş çaplı kullanılan graf örneklerindendir.

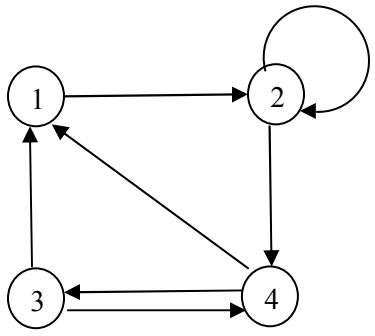
Genellikle pratik problemlerin birçoğu, grafların bazı sıralama problemleri olarak ele alınabilir. Tüm şehirler arasında en ucuz maliyetli yol bulma problemi olan gezgin satıcı problemi, komşu şehirlerin aynı renkte olmaması şartıyla renklendirme problemi, Edirne' den Ardahan' a gidilecek en kısa yolu bulma ya da Web sayfaları arasında dolaşma verilebilecek örneklerdendir.

Tanım 1.1: Graf

Bir *graf* $G = (V, E)$, sınırlı bir V köşe kümesi ve sınırlı E kenarlarından oluşur.

Tanım 1.2: Yönlü Graf

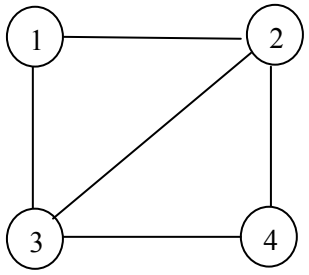
Yönlü graf $u, v \in E$ olmak üzere, (u, v) düğüm çiftlerinin sıralanmış bir kümesidir.



Şekil.1.1 Yönlü graf örneği

Tanım 1.3: Yönsüz Graf

Yönsüz graf, u, v Є E olmak üzere, (u,v) düğüm çiftlerinin sıralanmamış bir kümesidir.



Şekil.1.2 Yönsüz graf örneği

Tanım 1.4: Yol

Bir düğümden diğerine gidilmesi için izlenecek düğümlere *yol (path)* denir.

Tanım 1.5: Basit Yol

Bir yolda birinci düğüm ile sonuncu düğümden başka tekrarlama yoksa o yola *basit yol (simple path)* denir.

Tanım 1.6: Çevrim

Bir yolda birinci düğümle sonuncu düğüm aynı ise bu yola *çevrim(cycle)* denir.

Tanım 1.7: Acylic Graf

Acylic, çevrim(cycle) içermeyen graftır.

Tanım 1.8: Bağlı Graf

Graf yapısındaki her düğümden diğer düğümlerin hepsine bir yol varsa o graf *bağlıdır (connected)*.

Tanım 1.9: Komşu Düğüm

Graf yapısı içerisindeki iki düğüm arası bir kenarla birleştirebiliyorsa o iki düğüm *komşudur* denir.

Tanım 1.10: Maliyetli Graf

Graf yapısında kenarlar değer alabilir, bu değerler grafın yapısına katılabilir. Bir G grafı üzerindeki kenarların değerleri eşit değilse ve her biri farklı bir değer alabiliyorsa bu tür graflara *maliyetli graf (weighted graph)* denir. Bütün kenarların maliyeti aynı ise bu maliyetli graf olarak adlandırılmaz.

Tanım 1.11: Tamamlanmış Graf

Eğer graf üzerindeki tüm düğümler birbirlerine bağlanmışsa bu tür graflara *tamamlanmış graf (complete graf)* denir. Graf üzerindeki düğümlerin tümünün dereceleri eşittir.

Tanım 1.12: Düğüm Derecesi

Düğüme komşu olan düğümlerin sayısına d*üğümün derecesi* denir.

Tanım 1.13: Grafın Derecesi

En büyük düğüm derecesi *grafın derecesidir*.

Tanım.1.14: İlmik

Aynı düğümden başlayıp, aynı düğümde biten kenarlara *ilmik (loop)* denir.

II. BÖLÜM: GRAFIN BİLGİSAYARDA GÖSTERİMİ

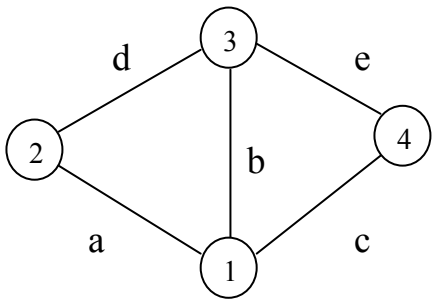
Graf bilgisayarda çeşitli şekillerde gösterilebilir. Gösterimin biçimi problemin türüne göre belirlenir. Bunlardan en efektif ve yalın olanı grafın matris şeklindeki gösterimidir. Diğer yandan problemin türüne ve gereksinimlere göre diğer gösterimlerde daha iyi sonuç verebilirler. Gösterim yöntemlerine bitişiklik matrisi, komşuluk matrisi vb. örnek gösterilebilir.[1]

2.1. Bitişiklik (Incident) Matrisi

Bu matris (nxm) büyüklüğündedir. Burada n düğüm sayısını, m ise düğümler arası bağları ifade eder. Yönsüz grafların gösteriminde bu matrisin elemanları 0 ve 1 değerlerini almaktadır. Matrisin her elemanı aşağıdaki bağıntıyla belirlenir.

$$a(i,j)=\begin{cases}1 & \text{i düğümü j kenarıyla bağlantılı ise}\\ 0 & \text{aksi durumda}\end{cases}$$

Örneğin, Şekil.2.1'de verilen grafın ilişki matrisi Tablo.2.1'de görülmektedir.



Şekil 2.1 Örnek graf

[1] **Çölkesen, Rıfat** Veri Yapıları ve Algoritmalar, Papatya Yayıncılık, 1. Basım 2002. S. 322

Tablo 2.1 Örnek grafın ilişki matrisi

Kenar / Düğüm No	**a**	**b**	**c**	**d**	**e**
1	1	1	1	0	0
2	1	0	0	1	0
3	0	1	0	1	1
4	0	0	1	0	1

Yönlü grafların gösteriminde ise ilişki matrisinin elemanları, kenarların yönüne bağlı olarak (-1, 0, 1) değerleri almaktadır.

$$a(i,j)=\begin{cases} 1 & \text{i düğümü j kenarıyla bağlantılı ise} \\ -1 & \text{i düğümü j kenarıyla ters bağlantılı ise} \\ 0 & \text{i düğümü j kenarıyla bağlantılı değilse} \end{cases}$$

Bazı durumlarda graflar ilmikler barındırır. Bu durumda A matrisini iki alt matrise parçalamak gerekir. Bu matrislerin elemanları aşağıdaki gibidir:

$$a_+(i,j)=\begin{cases} 1 & \text{i düğümü j kenarının başlangıcı ise} \\ 0 & \text{aksi durumda} \end{cases}$$

$$a_-(i,j)=\begin{cases} 1 & \text{i düğümü j kenarının sonu ise} \\ 0 & \text{aksi durumda} \end{cases}$$

İlişki matrisi kenarlar arasında bilgi içermediğinden bilgisayarlı modellemede çok da tercih edilmez. Ayrıca bellekte (*nxn*)' lik yer kaplar. Bu nedenle algoritmik açıdan en kötü graf temsilidir. [2]

[2] **Nabiyev, Vasıf** Teoriden Uygulamalara Algoritmalar, Seçkin Yayıncılık, 1. Basım 2007, s. 294.

2.2. Komşuluk(Adjacency) Matrisi

Komşuluk matrisi, (*n x n*) büyüklüğünde matris olup matris düğümleri arasındaki ilişkileri gösterir.

$$b(i,j)=\begin{cases}1 & \text{i düğümü j düğümü ile ilişkili ise}\\ 0 & \text{aksi durumda}\end{cases}$$

Düğümlere ağırlık konduğunda matrisinde 1 yerine her düğümün ağırlık değerleri yazılmaktadır. Komşuluk matrisinin tercih edilme sebeplerinden biri, matrisin tek bir sorgulamada düğümleri birleştiren kenarlar hakkında da bilginin elde edilmesidir. Dezavantajı ise bellekte n^2 lik yer kaplamasıdır. Şekil 2.1 için komşuluk matrisi aşağıda Tablo.2.2' de verilmiştir.

Tablo 2.2. Komşuluk matrisi

Düğüm No	***1***	***2***	***3***	***4***
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Komşuluk matrisi simetrik olduğundan graf ilişkilerini komşuluk matrisinin sadece üst üçgen matris kısmı ile ifade ederek bellekten tasarruf edilebilir. [3]

2.3. Düğüm Çiftlerinin Listesi

Belleğin verimli kullanımı açısından grafların en uygun gösterimi, *m* sayıda kenarlara uygun gelen düğüm çiftlerinin listesi biçimidir. Bu şekilde grafın gösterimi bellekte *2m*'lik yer kaplamaktadır. Örneğin Şekil.2.1' deki düğüm çiftlerinin tablosu aşağıdaki gibidir.

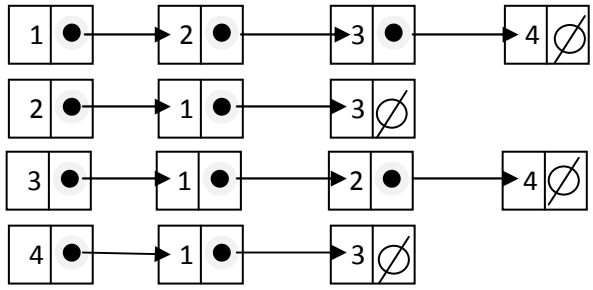
[3] **Nabiyev**, a.g.k., s.296.

Tablo 2.3. Düğüm çiftleri listesi

1	2
1	3
1	4
2	3
3	4

2.4. Listeler

Listeler bilgisayarda gösterim açısından programlamada çok kolaylık sağlamaktadırlar. Listelerin düzenli olması arama sonucunda erişimi hızlandırmaktadır. Bu yöntemin diğer olumlu yanı, bilgisayar belleği müsait oldukça yeni bağlantıların yapılabilmesidir Örneğin Şekil.2.1 deki grafın listelerle ifadesi Şekil.2.2' de verilmiştir.[4] [5]



Şekil 2.2. Düğüm Çiftleri Listesi

[4] **Nabiyev**, a.g.k., s.297.
[5] **Nabiyev**, a.g.k., s.297.

III. BÖLÜM: GRAFLARDA ARAMA ALGORİTMALARI

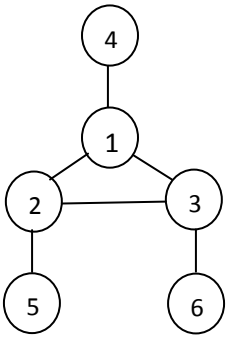
3.1. Derinine Arama/Dolaşma Algoritması

Derinine Arama/Dolaşma Algoritması, 1972 yılında R.E.Tarjan tarafından önerilmiştir. Derinine arama algoritmasında kök olarak aramanın başlanması gereken herhangi bir düğüm ele alınır. Arama ağacında fazla dallanmadan kökten gidilebilecek en uzak (derin) düğüme kadar ilerlenir. Genel olarak bu yöntemde iki durum söz konusudur:

- Göz önüne alınan düğüm daha önceden ele alınmışsa ebeveyn düğüme geri dönülerek diğer çocuk düğümleri incelenir.

- Yeni göz önüne alınan düğüme daha önce gelinmişse bu düğüm, ebeveyn kabul edilerek bir sonraki derinliklerdeki düğümlere gidilmeye çalışılır.

Bu yöntem özyineli bir biçimde çalıştığından ayrıca ilgi çekmektedir. Derinene doğru ilerlerken ziyaret edilen düğümler bir yığına yerleştirilmektedir. Çocuğu olmayan en son düğüm ziyaret edilerek yığına yerleştirme işi sonlanır. Bu kez son eklenen düğüm yığından çekilerek geri dönüş yapılır. Daha önce ziyaret edilen düğüme döndükten sonra o düğümün ziyaret edilmemiş çocuklarından işleme devam edilir. Derinine arama algoritması aynı zamanda graflarda döngüleri ve açılım ağaçlarını belirlemede kullanılmaktadır. Aşağıda Şekil.3.1'deki graf örneği üzerinde derinine arama işlemini gösterelim.[1]



Şekil 3.1. Graf Örneği

[1] **Nabiyev**, a.g.k., s.309.

Şekil 3.2(a) daki durumu adımlarla açıklayalım:

1.adım: Bu graf üzerinde ziyaret edilecek ilk düğüm 1 düğümü olarak seçilsin.

2.adım: Derine doğru ilerlendiğinden 1 düğümünün ilk komşusu olan 2 düğümüne gidilir.

3.adım: 2 düğümünün komşularına bakılır. 2 düğümünün ilk komşusu olan 1 düğümü daha önce ziyaret edildiğinden bir sonraki 3 düğümüne gidilir.

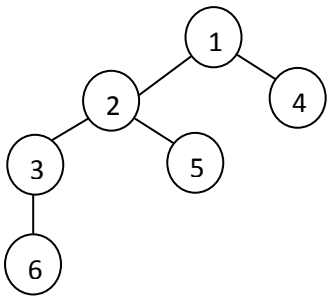
4.adım: 3 düğümü yığına eklenir.

5.adım: Bu düğümden ziyaret edilmeyen 6 düğümü bulunur. 6 düğümünün komşuları olmadığından tekrar 3 düğümüne oradan da geriye 2 düğümüne dönülür.

6.adım: 2 düğümünün ziyaret edilmeyen komşusu 5 düğümüdür. Artık bunun çocukları olmadığından bu düğümden ebeveyni olan 2 düğümüne, oradan da kök düğüme (yani 1 düğümüne) geri dönülerek ziyaret edilmemiş yeni düğümler aranır.

7.adım: Bu kez 4 düğümüne gidilir. Böylece düğümlere göre gezinme tamamlanmış olur.

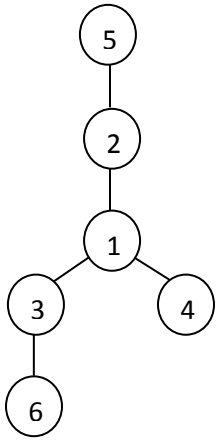
Şekil 3.2.(b)'de de başlangıç düğümünün 5 olduğu varsayılarak derinene arama algoritması uygulanmıştır. Şekil 3.2 (a) ve (b) deki sonuçlar açılım ağaçlarına karşılık gelmektedir.



(a) Kök 1 seçilirse

Ziyaret Sırası : 1,2,3,6,5,4

Çıkış Değerleri : 6,3,5,2,4,1



(b) Kök 5 seçilirse

Ziyaret Sırası : 5,2,1,3,6,4

Çıkış Değerleri : 6,3,4,1,2,5

Şekil 3.2. Derinine aramada düğümlerin ziyaret ve elde ediliş sırası

Şekil 3.2. (a) ve (b) den derinine aramada başlangıç düğüme göre aynı graf için farklı derinliklerde aramalar görülmektedir. 1 düğümü seçildiğinde 3 derinliğinde, 5 düğümü seçildiğinde ise 4 derinliğinde aramalar yapılmaktadır. Görüldüğü gibi başlangıç durumlarına göre küçük farklarla tüm yollar bulunabilir.[2]

Algoritmanın Kaba Kodu:[3]

```
DerinineArama( dugum d )
{
d.dugumuziyaretet;
      for(i=0;i<dugumsayi;i++)
      {
            if(dugumziyaretedilmemisse)
            {
                  DerinineArama(dugum i);
            }
      }
}
```

Algoritmada DerinineArama() fonksiyonunun çağırılma sayısı $O(N)$ dir. $G(V,E)$ grafında DerinineArama() fonksiyonunun döngü içerisinde komşuluk listesine göre gezinirken harcanan toplam zaman $O(E)$ dir. Dolayısıyla algoritmanın gerektirdiği zaman karmaşıklığı $O(V+E)$ olarak bulunur. Algoritma özyinelemeli olduğundan derinine arama algoritmasının çok büyük graflarda denetlenmesi zor olabilir. Bu nedenle çoğu zaman *sınırlı derinine arama yöntemi* kullanılmaktadır.

3.2. Enine Arama Algoritması

*Enine Arama Algoritması'*nda verilen kök düğüme komşu olan tüm düğümler birinci derinlikte incelenir. Belirli bir derinlikte bütün düğümler tek tek incelendikten sonra bir sonraki düğümün değerleri incelenir. Enine arama her zaman köke en yakın çözümü verir. Ama çözüme ulaşılana kadar o derinlikteki tüm düğümler ziyaret edileceği için problemin

[2] **Nabiyev**, a.g.k., s.309.
[3] **Nabiyev**, a.g.k., s.310.

çözümü daha derin olduğunda ziyaret edilen düğümler sayısı artacağından, enine arama etkin olmamaktadır. Enine arama algoritmasını yukarıda verilen graf (Şekil 3.1) için inceleyelim.

Şekil 3.3(a) daki durumu enine arama algoritması ile anlatalım:

1.adım: Kök düğüm olarak 1 düğümü seçilsin.

2.adım: Seçilen bu düğüm kuyruğa eklenir.

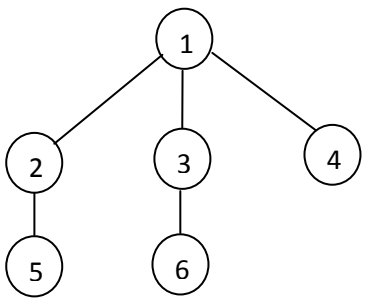
3.adım: 1 düğümünün komşuları olan 2, 3 ve 4 düğümleri peş peşe ziyaret edilir ve kuyruğa eklenir.

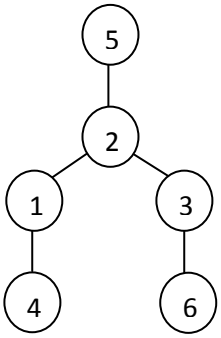
4.adım: İlk değer olan 2 düğümü kuyruktan çıkarılarak arama listesine konur ve komşusu olan 5 düğümü kuyruğa eklenir.

5.adım: Bir sonraki 3 düğümü ve onun komşusu olan 6 düğümü kuyruğa eklenir.

6.adım: Sıradaki düğüm 4 düğümüdür onun ve kuyruktaki diğer düğümlerin komşuları bulunmadığından sırasıyla bu düğümler kuyruktan çıkarılarak enine arama tamamlanmış olur.

Şekil 3.3 (b) de ise 5 başlangıç düğümü için enine arama gösterilmiştir.

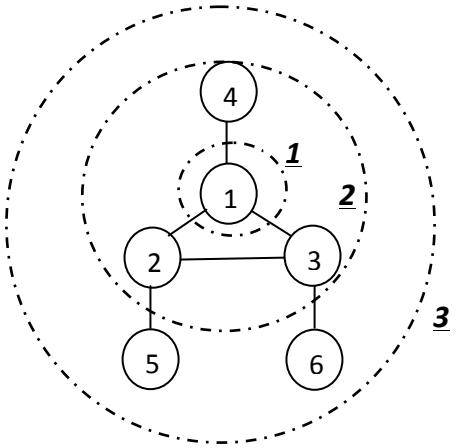




(a) Ziyaret Sırası: ((1),(2,3,4),((5),(6))) (b) Ziyaret Sırası: ((5),(2),(1,3),((4),(6)))

Şekil 3.3. Enine arama örneği

Enine aramanın ağaç üzerinde kapsadığı düğümler, kökten yayılan dalgalara benzemektedir. Yukarıdaki Şekil 3.1' deki graf için dalgaların yayılımı geometrik olarak Şekil 3.4' de gösterilmiştir. Aynı dalga üzerinde yer alan düğümlerin kök düğüme olan uzaklıkları aynıdır. Her dalga bir seviyeye karşılık gelmektedir.



Şekil 3.4. Enine Arama yönteminin dalga yayılımı[4]

Enine aramanın mantığını ve amacını açıkladıktan sonra, algoritmayı aşağıdaki şekilde verebiliriz. Algoritmanın uygulanabilmesi için enine düğüm ziyaretini gösteren L listesine ve komşulukları tutan dizine gerek duyulmaktadır. Bu yapılar kuyruk veri türüne uygun olmaktadır.

Enine Arama Algoritması:[5]

1- **Adım:** Bir başlangıç düğümü seçilir ve bu düğüm işaretlenir.
2- **Adım:** Bu düğümün komşuları sırası ile T listesine yazılır.
3- **Adım:** Düğümler teker teker ziyaret edilir ve işaretlenerek L listesine atılır.
4- **Adım:** T Listesindeki ilk düğüme gidilir ve işaretlendikten sonra T listesinden silinir.
5- **Adım:** Ele alınan (silinen) düğümün komşuları T listesine eklenir(bunu yaparken komşu düğüm daha önce ziyaret edilmişse bunu L den bakarak eklemeyeceğiz.
6- **Adım:** L listesine bu düğümler eklenir (aynı şekilde ziyaret edilen düğümler listeye eklenmeyecek).
7- **Adım:** İterasyonlar T listesinde tüm düğümler silinene kadar devam ettirilir.

[4] **Nabiyev**, a.g.k., s.311.
[5] **Nabiyev**, a.g.k., s.312.

Algoritmada her düğüm kuyruğa yalnız bir kere konmaktadır. Bu nedenle en dıştaki döngü en çok $|V| = d$ kez icra edilmektedir. Her kenar içteki döngüde bir kez incelenmekte ve en çok k kez tekrarlanmaktadır. Bu yüzden algoritmanın gerektirdiği zaman karmaşıklığı $O(d + k)$ olarak bulunur.[6]

3.3. Greedy Yöntemi

Greedy Yöntemi, graf üzerinde belirli bir konuda en iyi sonucu bulabilmek amacıyla dolaşma yapılırken bir sonraki düğümü belirlemek için kullanılan bir karar verme/seçme yöntemidir. Greedy yönteminde o andaki seçenekler içerisinden en iyi görüneni seçilir. Bunun için gerekli seçim kriteri bölgesel/yerel değerlendirmelere göre yapılır ancak seçilenin global olarak sistem için en iyi seçim olacağı öngörülür.

Greedy yöntemi her zaman için en iyi sonuca götürmeyebilir. Ancak büyük çoğunlukla en iyi sonucu vermektedir. Greedy yöntemi graf üzerine geliştirilmiş olan birçok problemin çözümünde, bir sonraki düğümün belirlenmesinde seçme unsuru olarak kullanılmaktadır. Bilgisayar biliminde graf veri modeline yaklaştırılan problemlerde geniş bir kullanım alanı vardır. Örneğin en kısa yol ağacı ve en kısa yol bulan algoritmalarda kullanılmaktadır.[7]

Algoritmanın Kaba Kodu:[8]

```
set Greedy (AdayKumesi){
   cozum:= yeni Kume( );
   while (AdayKumesi geçerli ise (boş değilse) )
   {
      seçilenAday := lokal seçim kriterlerine, aday kümesinden bir
                     elaman seç, bunu kümesinden sil ve değerini dön
      if (seçilenAday iyi bir seçim ise )
          çözüm kümesine ekle
      if (problem çözüldü mü )
          return cozumKumesi
   }

   return null  // Çözüm bulunmuyorsa
}
```

[6] **Nabiyev**, a.g.k., s.311.
[7] **Çölkesen**, a.g.k., s.349.
[8] **http://en.wikipedia.org/wiki/Greedy_algorithm**

IV. BÖLÜM

4.1. En Kısa Yol Probleminin Tanımı

En kısa yol problemi (the shortest path problem), verilen iki düğüm arasındaki maliyeti en düşük olan bir yol bulma problemidir. Daha formal olarak, bir maliyetli graf (V düğüm kümesi, E kenar kümesi ve $f: E \rightarrow R$ değer kümesi olmak üzere) , her $v \in V$ olmak üzere bir v düğümü verilmiş olsun. Problem, v den $v' \in V$ ye toplam maliyetin $(\sum_{p \in P} f(p))$ minimum olduğu yolun bulunmasıdır. [1]

4.2. En Kısa Yol Problemlerinde Kavramlar ve Tanımlar

En kısa yol problemi dört türe ayrılabilir. Bunların çözümlerinde değişik yaklaşımlar kullanılmaktadır.

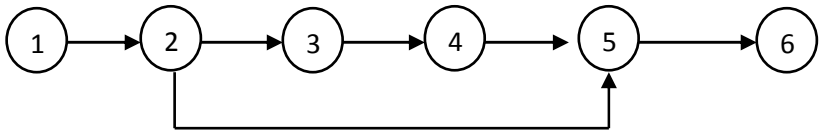
1. Tek bir hedefe en kısa yolların bulunması (Single-destination shortest path).
2. Bir başlangıç düğümünden, diğer düğümlere en kısa yolların bulunması (Single-source shortest path).
3. Verilen iki düğüm arasındaki en kısa yolun bulunması (Single-pair shortest path)..
4. Tüm düğüm çifti için en kısa yolun bulunması (All-pairs shortest path).[2]

Teorem 4.2.1

En kısa yolların alt yolları da en kısa yollardır.

İspat:

Eğer bazı alt yollar en kısa değilse, daha kısa alt yollar alınarak daha kısa toplam yol oluşturulur.



Şekil 4.1 Alt Yolun Varlığı

Şekil 4.1'den görüldüğü gibi 2 ve 5 düğümleri arasında iki yol vardır. Burada 1-2-5-6 en kısa yoldur.

[1] http://en.wikipedia.org/wiki/Shortest_path_problem
[2] **Çölkesen**, a.g.k., s.353.

Teorem 4.2.2

En kısa yol çözümlerinde döngüler olmaz.

İspat:

n düğümlü bir grafda en kısa yol $(n-1)$ kenardan oluşur, daha fazla kenardan oluşamaz çünkü düğüm sayısı n dir.

4.3. Problemin Çözümünde Kullanılan Yöntemler

Graf üzerinde iki düğüm arasındaki en kısa yolu bulmak için çeşitli algoritmalar mevcuttur. Problemin çözümünde ilk akla gelen, tüm yolların uzunluklarının hesaplanıp karşılaştırdıktan sonra en kısasının bulunmasına dayalı kaba kuvvet yaklaşımının uygulanmasıdır. Fakat düğüm sayısı artığında işlem zamanı çok artacağından etkin değildir. Bu nedenle problem için tekrarlı hesaplamalardan vazgeçilerek her adımda çözüme yaklaştıran bir yönteme ihtiyaç duyulmaktadır.

4.3.1. Dijkstra Algoritması

Eğer tüm kenar maliyetleri sıfıra eşit veya büyükse, tek-kaynaklı (single source) problemlerin çözümünde kullanılır. Algoritmada çalışma zamanını (run-time) kötüleştirmeksizin, verilen düğümden diğer tüm düğümlere en kısa yollar hesaplanır.

Dijkstra Algoritması, adını Alman bilim adamı Edsger Dijkstra dan alan bir Greedy yöntemidir. Yönlü ve pozitif ağırlıklar için en kısa yol problemlerinde kullanılır.

Algoritmanın Kaba Kodu:[3]

```
function Dijkstra(G, w, s)
   for each vertex v in V[G]           // Initializations
         d[v] := infinity              // Unknown distance function from s to v
         previous[v] := undefined
   d[s] := 0                           // Distance from s to s

   Q := V[G]                           // Set of all unvisited vertices
   while Q is not an empty set         // The algorithm itself
         u := Extract_Min(Q)           // Remove best vertex from priority queue
        for each edge (u,v) outgoing from u
```

[3] http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Algoritmanın Kaba Kodu (önceki sayfadan devam)

```
if d[u] + w(u,v) < d[v]     // Relax (u,v)
   d[v] := d[u] + w(u,v)
   previous[v] := u
```

Algoritmada yer alan $u = Extract\_Min(Q)$ işlemi, Q kümesindeki u düğümüne en küçük $d[u]$ değerinin aranması anlamındadır. Bu düğüm işlemden sonra Q dan silinir.

$d[v]$ uzaklık değerinin, s kaynak düğümünden, v hedef düğümüne giden en kısa yol maliyetine eşit olması sağlanmalıdır. Bu işleme *relaxation* denir. Bu yukarıdaki şu şekilde ifade edilir:

Eğer $d[u] + w(u,v) < d[v]$ ise $d[v] = d[u] + w(u,v)$ dir.

4.3.1.1. Yürütme Zamanı

En basit bir Dijkstra algoritması uygulamasında Q kümesinin düğümleri bağlı listeler (linked list) veya diziler (array) şeklinde ele alınır. Extract_Min(Q) işlemi basitçe Q kümesinin tüm düğümleri için doğrusal bir arama yapar. Bu durumda yürütme zamanı $O(|V|2 + |E|)$ dır.

Seyrek graflarda (köşe sayısı $|V^2|$ den daha azdır) graf, komşuluk matrisi kullanılarak ayrıca Extract_Min fonksiyonu için *binary heap* veya *Fibonacci heap* kullanarak çok daha etkin bir şekilde temsil edilebilir. Bu durumda binary heap dahil olmak üzere algoritmanın ihtiyaç duyduğu yürütme zamanı $O((|E| + |V|)log|V|)$ dir. Fibonacci heap kullanıldığında ise gerekli olan zaman $O(|E| + |V| log|V|)$ şeklinde daha da iyileşir.[4]

4.3.1.2. Algoritmanın Çalışma Şekli

Algoritma bitene kadar her v düğümü için s ve v düğümleri arasında en az maliyete sahip $d[v]$'nin bulunmasına odaklanılır. Çalışmaya, başlangıç düğümünün maliyeti sıfır $(d[s] = 0)$, diğer tüm düğümlerin maliyetleri sonsuz $(d[v] = \infty)$ alınarak başlanır. Algoritma

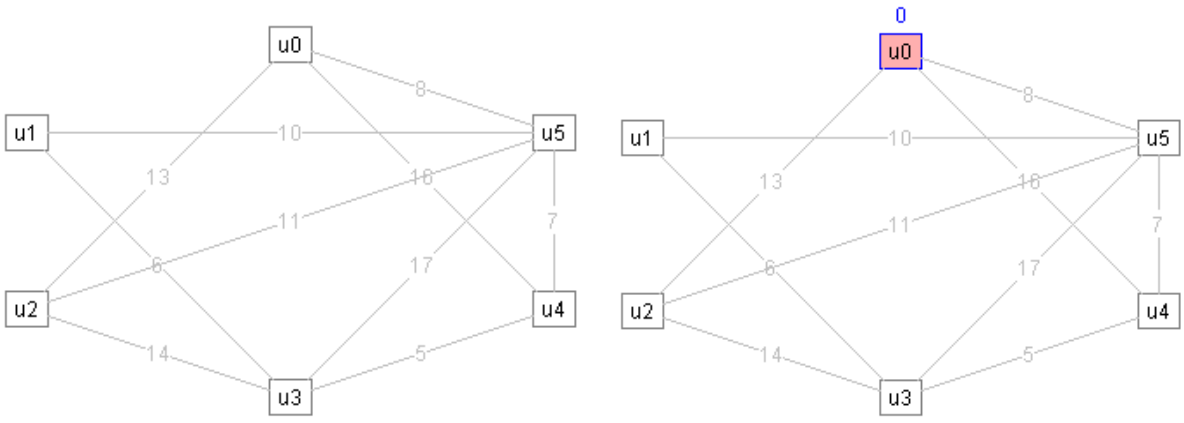
[4] http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

bittiğinde $d[v]$, s'den v'ye en kısa yol maliyetine sahip olur. Eğer aralarında bir yol yoksa d[v]'nin değeri sonsuz olacaktır.

Relaxation notasyonu, en kısa yolun tahmini ile sarmal gerilim yayının uzunluğu arasındaki benzetimden gelir. Başlangıçta, en kısa yol maliyeti, sıkıştırılmış yaydaki gibi abartılıdır. Daha kısa yollar bulunduğunda tahmini maliyet küçülmeye başlar, aynı serbest bırakılan yaydaki gerilimin küçülmesi gibi. Böylece eğer varsa en kısa yol bulunduğunda yay da rahat bırakılmış, kendi uzunluğuna gelmiş olacaktır.

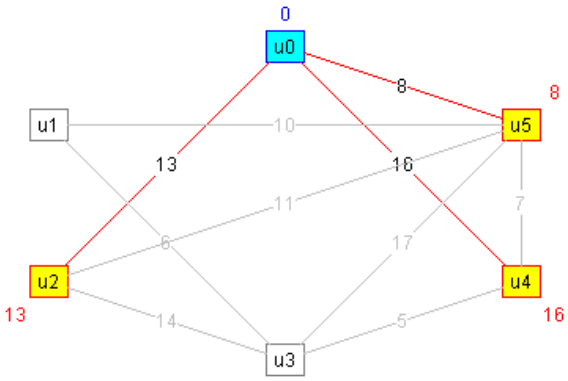
Algoritmada iki ana küme vardır. S ve Q. S kümesi, en düşük $d[v]$ yol maliyetine sahip tüm düğümleri içermektedir. Diğer düğümler ise Q kümesinde bulunurlar. S, başlangıçta boş kümedir. Her adımda Q'dan S'ye bir düğüm taşınır. Bu düğüm en düşük $d[u]$ maliyetine sahip olanlar arasından seçilir. Her u düğümü S'e taşındığında algoritma, ele alınan her (u, v) kenarı için relaxation işlemini yapar.[5]

Dijkstra Algortiması'nın çalışma adımları:

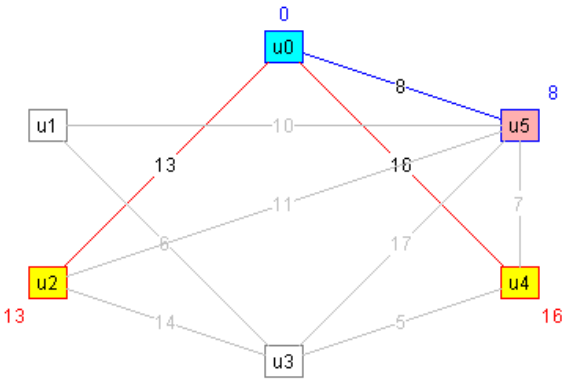


(a) 1.adım (b) 2.adım

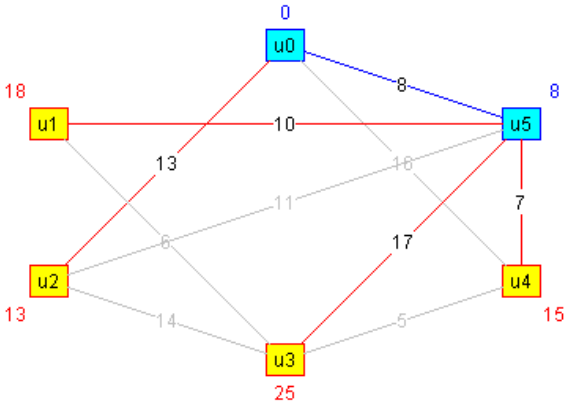
5 http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm



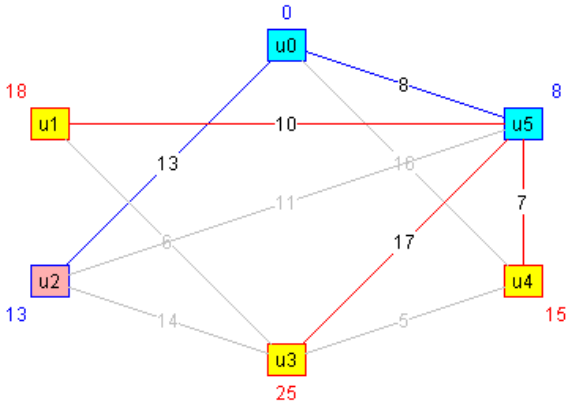
(c) 3.adım



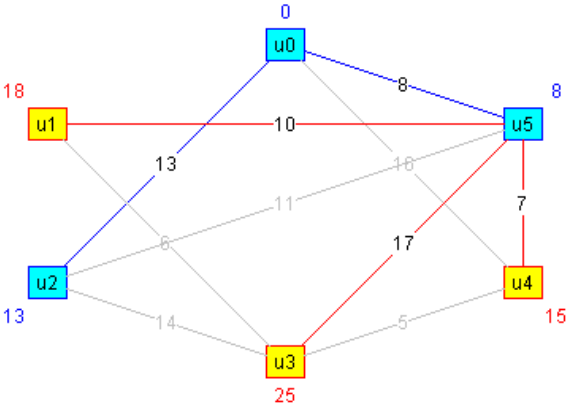
(d) 4.adım



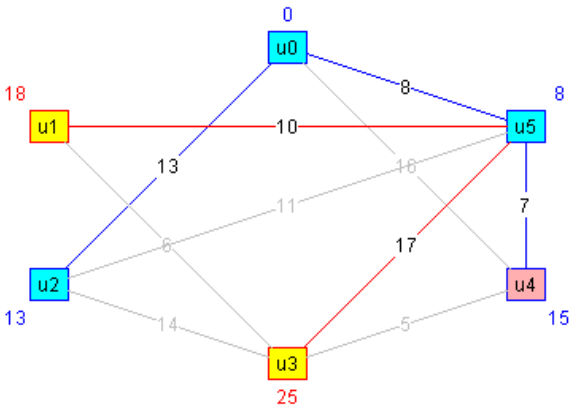
(e) 5.adım



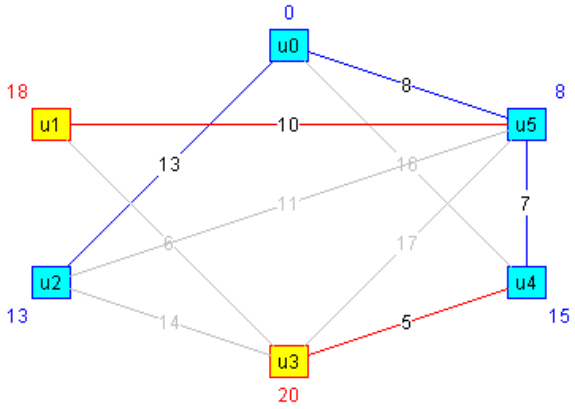
(f) 6.adım



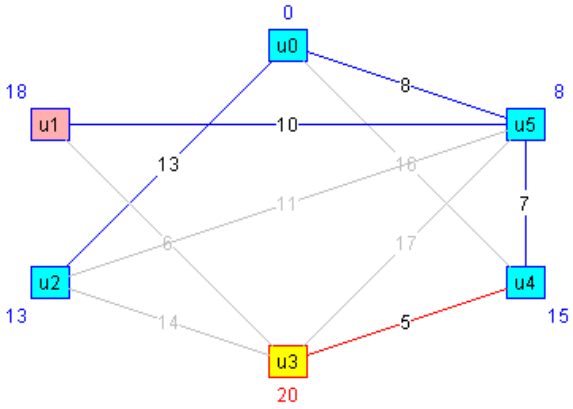
(g) 7.adım



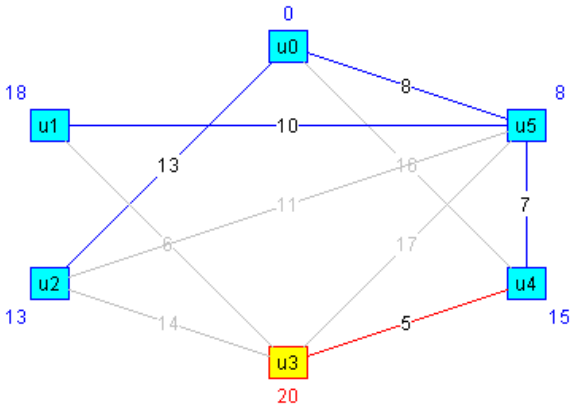
(h) 8.adım



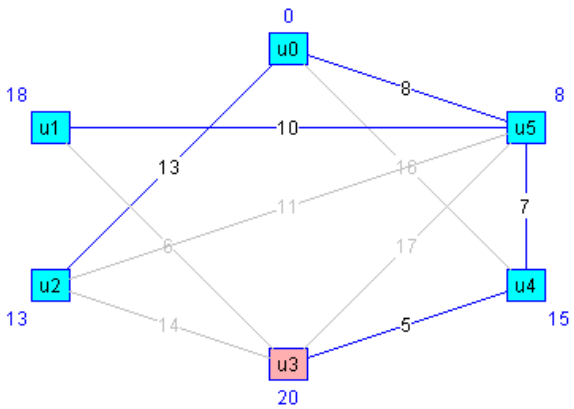
(i) 9.adım



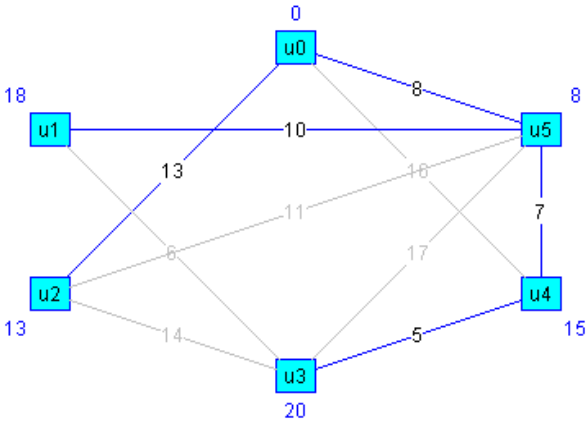
(j) 10.adım



(k) 11.adım



(l) 12.adım



(m) 13.adım

Şekil 4.2 (a)-(m) Dijkstra algoritmasının çalışma şekli[6]

[6] www-b2.is.tokushima-u.ac.jp/ ~ikeda/suuri/dijkstra/DijkstraApp.shtml/Demo1

4.3.2. Bellman-Ford Algoritması

Bellman-Ford algoritması maliyetli (negatif maliyetli de olabilir) yönlü graflarda tek-kaynaklı en kısa yolları (single-source shortest path) bulmak için kullanılır.

Dijkstra algoritmasında aynı problem çok daha düşük çalışma zamanıyla gerçekleştirilmesine karşın maliyetlerin pozitif olma koşulu aranır. Bundan dolayı Bellman-Ford algoritma sadece maliyetler negatif olduğunda kullanılmaktadır.

Algoritmanın zaman karmaşıklığı $O(V - E)$ dir. Burada V düğüm, E de kenarları temsil etmektedir.[7]

Algoritmanın Kaba Kodu:[8]

```
function BellmanFord(list vertices, list edges, vertex source)
//Adım 1: graf hazırlanıyor
for each Düğüm v in DüğümKümesi:
       if v is başlangıç then v.uzaklık := 0
       else v.uzaklık := sonsuz
       v.predecessor := null

// Adım 2: Tekrarlı olarak kenarları relax et.
for i from 1 to size(DüğümKümesi):
       for each kenar uv in KenarlarKümesi:
           u := uv.baslangıç
           v := uv.hedef                    // uv u' dan v' ye olan kenar
           if v.uzaklık > u.uzaklık+ uv.ağırlık:
               v.uzaklık := u.uzaklık+ uv.ağırlık
               v.predecessor := u
// Adım 3: negatif maliyetli çevrimler (cycles) için kontrol et
for each kenar uv in KenarlarKümesi:
       u := uv.başlangıç
       v := uv.hedef
       if v.uzaklık > u.uzaklık+ uv.ağırlık:
           error "Graf  negatif ağırlıklı çevrim içermekte"
```

[7] http://en.wikipedia.org/wiki/Bellman-Ford_algorithm
[8] http://en.wikipedia.org/wiki/Bellman-Ford_algorithm

4.3.3. A* Arama Algoritması

A* (A star) algoritması tek-kaynaklı en kısa yol problemini sezgisel (heuristics) öngörme yöntemi ile çözmeye çalışır. Algoritma ilk olarak 1968 yılında Peter Hart, Nils Nilsson, ve Bertram Raphael tarafından tanımlanmıştır.

Algoritmanın sezgisel değerlendirme için kullandığı fonksiyon:

$$f(x) = g(x) + h(x)$$

şeklindedir. Bu fonksiyonda $g(x)$ verilen başlangıç düğümünden x düğümüne kadar olan *toplam yolu*, $h(x)$ ise x den hedefe kadar olan *tahmini en uygun yol maliyetini* gösterir. Fonksiyondan beklenen, hedefe yaklaştığında değerinin küçülmesidir. $h(x) = 0$ durumunda algoritma derinine arama (best-first) algoritması gibi çalışır.

Genel olarak, derinine arama (Depth-first search) ve enine arama (breadth-first search) algoritmaları A*'ın özel bir halidir. Ayrıca Dijkstra algoritması bir enine arama algoritması olarak her x düğümü ve $h(x) = 0$ için A*'ın özel bir halidir. Derinine arama algoritması için, C gibi çok büyük değere sahip bir sayaç varmış gibi varsayılır. Düğümleri her işlediğimizde, bu düğüme komşu yeni bulunan düğümlere C'yi atarız. Her işlemden sonra C, bir azaltılır. Böylece büyük $h(x)$ değerlerine sahip olan düğümler önce bulunurlar.[9]

Algoritmanın Kaba Kodu:[10]

```
function A*(başlangıçdüğümü,hedef)
    var kapalı := BoşKüme
    var q := kuyruk_olustur(yol(başlangıçdüğümü))
    while q boş değilse
        var p := ilkini_sil(q)
        var x := p nin son düğümü
        if x in kapalıküme
            devam et
        if x == hedef
           return p
        add x to kapalı
        for each y in ardıl(p)
            kuyruk_ekle(q, y)
```

[9] http://en.wikipedia.org/wiki/A*_search_algorithm
[10] http://en.wikipedia.org/wiki/A*_search_algorithm

```
return failure
```

Algoritmada, kuyruğun f değerine göre otomatik sıralandığı varsayılır. *Kapalı küme*, p'nin işlenen tüm düğümlerini içerir, böylece tekrardan ve çevrimden kaçınılmış olunur. Kuyruk bazen *açık küme* olarak da adlandırılır. Eğer çözümün varlığı garanti edilmiş ya da ardıl fonksiyon kapalı çevrimleri kabul etmeyecek gibi ayarlanmışsa, kapalı küme ihmal edilebilir.

4.3.4. Floyd-Warshall Algoritması

Floyd–Warshall algoritması, Bernard Roy (1959) algoritmayı tanımladığından bazen *Roy–Floyd algoritması* olarak adlandırılır. Yönlü ve maliyetli graflarda en kısa yolları bulmak için kullanılan algoritmalardandır. Algoritma tüm düğüm çiftleri arasındaki en kısa yolları bulur. Bunu V^3 zamanıyla yapar, burada V graf üzerindeki düğüm sayısıdır.

Floyd-Warshall algoritması, graf üzerindeki tüm olası düğüm çiftleri arasındaki yolları karşılaştırır. Yoğun graflarda kullanılması daha iyi bir seçim olabilir denebilir. Algoritmanın verdiği sonuç Dijkstra algoritmasının graf daki düğüm sayısı olan $|V|$ kez çalıştırılması ile elde edilir. Eğer graf seyrek ise tüm düğümler arasında en kısa yolları belirlemek için Dijkstra algoritmasının $|V|$ kez çalıştırılması, graf yoğun ise Floyd–Warshall algoritması daha iyi sonuç verir. [11]

Algoritmanın Kaba Kodu:[12]

// edgeCost(i,j) fonksiyonun i ile j arasındaki maliyeti verildiği varsayılsın (aralarında bir ilişki yoksa fonksiyon sonsuz döner).
// n düğüm sayısı ve edgeCost(i,i)=0 varsayılsın

```
int path[][]; // A 2-boyutlu matris. Algoritmanın her adımında path[i][j] en kısa yoldur.

procedure FloydWarshall()
  for k:=1 to n
    begin
       for each (i,j) in (1..n)
       begin
          path[i][j] = min( path[i][j], path[i][k]+path[k][j] );
```

[11] http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm
[12] http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm

```
        end
    end
endproc
```

4.3.5. Johnson's Algoritması

Algoritma yönlü, maliyetli ve seyrek graflar üzerinde tüm düğüm çifti için en kısa yolun (all-pairs shortest path) bulunması, problemlerin çözümünde kullanılan bir başka yöntemdir. Seyrek(sparse) graflar üzerinde Floyd-Warshall algoritmasından daha hızlı sonuç verebilir.

İlk olarak bir düğüm sıfır maliyetli bir kenar ile beraber diğer düğümlere eklenir. Bellman-Ford algoritması negatif maliyetli bir çevrimin olup olmadığının kontrolü ve yeni eklenen düğümden v düğümüne en az maliyet $h(v)$'nin bulunması için çalıştırılır. Sonra $h(v)$ kullanılarak maliyetler yeniden hesaplanır. Sonuçta her bir düğüm için, Dijkstra algoritması çalıştırılır ve diğer düğümlere olan en az maliyetli hesaplanıp saklanır. $h(v)$ kullanılarak maliyetler tekrar hesaplanır. Algoritmanın zaman karmaşıklığı $O(V^2 log\ V + VE)$ dir.[13]

[13] http://en.wikipedia.org/wiki/Johnson%27s_algorithm

V. BÖLÜM: EN KISA YOL PROBLEMİ UYGULAMASI

5.1. Uygulama Adımları

Türkiye haritası üzerindeki iki şehir arasındaki "en kısa yol" problemi için aşağıdaki adımlar takip edilmiştir:

- ❖ En kısa yol problemi için verilerin toplanması.
- ❖ Veri yapılarının tasarımı.
- ❖ Algoritmanın kodlanması için platform ve dilin belirlenmesi.
- ❖ En kısa yol için kullanılacak algoritmanın seçilmesi.
- ❖ Belirlenen rotanın harita üzerinde gösteriminin sağlanması.

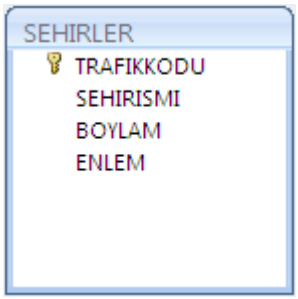
5.2. Verilerin Toplanması

Problem için gerekli şehirler arasındaki yol uzunlukları Beykent Üniversite'sinde bir bitirme tezinden alınmıştır.[1] Söz konusu tez çalışmadaki veriler Türkiye Karayolları Müdürlüğünden alınmış olup şehirler arası mesafeler için çift yönlü verilmiştir. En kısa yol uygulamasında ise şehirler arası yönsüz olarak düşünülmüş, böylece bu verilerden faydalanılarak üst üçgen matrisi oluşturulmuştur. Şehirlerin harita üzerinde gösterilmesi için gerekli olan enlem ve boylam bilgileri Google Map API de yer alan GclientGeocoder sınıfın getlatLng() metodu yardımıyla elde edilmiş, elde edilen veriler, http://www.mapsofworld.com/lat_long/turkey-lat-long.html sitesinde yer alan bilgilerle kontrol edilmiştir.

5.3. Kullanılan Veri Tabloları ve Veri Yapılarının Tasarlanması

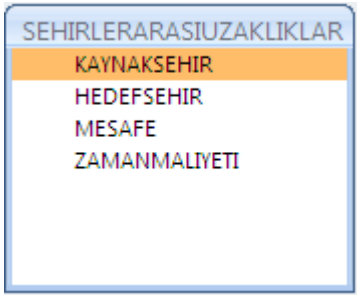
Gerekli olan şehir adları, enlem ve boylam bilgilerinin tutulması için Ms Access de **Sehirler** adlı tablo kullanılmıştır.

[1] Asuman OCAK, Design and Implementation Traveling Salesman Problem to Turkey Roadway Graph, Beykent University, 2003 s. 30-33



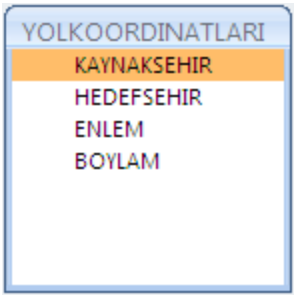
Şekil.5.1 Enlem boylam bilgilerini içeren tablo

Şehirler arası bağlantı ve yol mesafeleri, **SehirlerArasiUzakliklar** adlı tabloda tutulmuştur.



Şekil.5.2 Şehirler arası mesafeleri içeren tablo

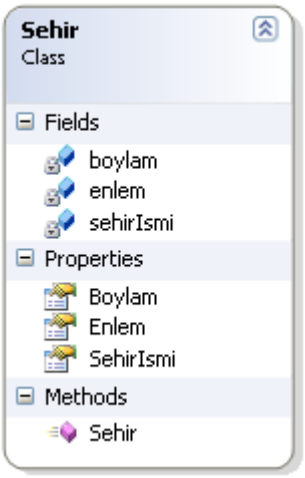
Harita üzerinde gösterilecek olan yolların gerçeğe uygun olabilmesi için yol koordinatlarının tutulabilmesi için **YolKoordinatlari** adli tablo hazırlanmıştır.



Şekil.5.3 Yol koordinatları içeren tablo

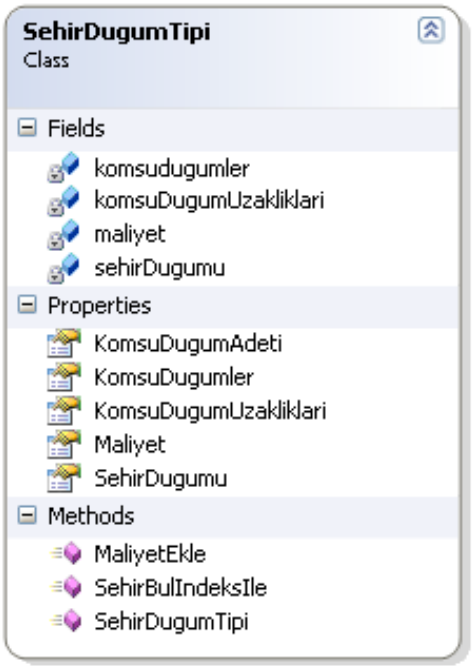
Problemin çözümü için gerekli veri yapıları Şekil.5.4, Şekil.5.5 ve Şekil.5.6' da gösterilen üç sınıf yardımıyla oluşturulmuştur;Sehir,SehirDugumTipi ve SehirGrafi.

Sehir Sınıfı: Grafın en küçük yapıtaşıdır. İçinde şehre ait isim, enlem, boylam gibi bilgiler bulunmaktadır.



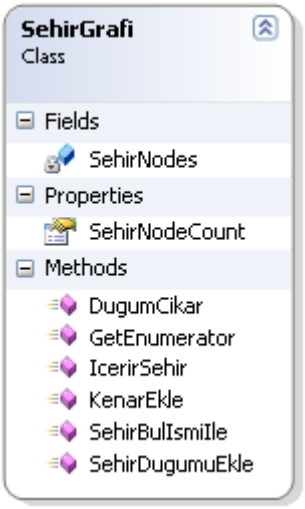
Şekil.5.4 Sehir Sınıfı

SehirDugumTipi: Sehir nesnelerinden oluşmaktadır. Nesne, harita üzerideki şehirleri temsil etmekte, şehirlerin komşuları ve bu komşu şehirlere ait uzaklık değerlerini içermektedir. Bu sınıfta hesaplamalarda kullanılan nesneler **ArrayList** sınıfı yardımlıyla tutulmaktadır.



Şekil.5.5 SehirDugumTipi Sınıfı

SehirGrafi: Şehirlerin graf üzerinde yerleştirilmesi, çıkarılması, kenarların eklenmesi gibi işlevleri yerine getirir. Şehir bilgisini için olarak **Sehir** sınıfını kullanır. Komşu düğüm adet düğümün derecesini gösterir. Grafa maliyetlerin eklenmesi veya değiştirilmesi işlevini yerine getirir.



Şekil.5.6 SehirGrafi Sınıfı

5.3.1. Veri Yapılarını Oluşturan Sınıflarda Kullanılan Yöntemler (Method)

Sehir sınıfı, verileri yapıcısı (constructor) yardımıyla almaktadır.

```
public Sehir(string Cname, string Lng, string Lat)
        {
            SehirIsmi = Cname;
            Enlem = Lat;
            Boylam = Lng;
        }
```

MaliyetEkle( ) Yöntemi : İstenilen şehirden komşu şehre ağrılıkların girilmesini sağlamaktadır. Eğer girilen maliyetler daha önce girilmişse, girilmiş olan yeni değeri eski değerle değiştirir.

```
public void MaliyetEkle(SehirDugumTipi mSehir, int mCost)
```

```
        {
        int mSehirIndex = SehirBulIndeksIle(mSehir.SehirDugumu.SehirIsmi);
            if (mSehirIndex >= 0)
                KomsuDugumUzakliklari[mSehirIndex] = mCost;
            else
            {
                KomsuDugumler.Add(mSehir.SehirDugumu);
                KomsuDugumUzakliklari.Add(mCost);
            }

        }
```

SehirBulIndeksIle ( ) Yöntemi: Bir şehrin, ismi verilen bir komşu şehrinin varsa bulunması ve indeksinin belirlenmesi görevini görür.

```
  public int SehirBulIndeksIle(string mSehir)
        {

            for (int i = 0; i < KomsuDugumler.Count; i++)
            {
                if (((Sehir)(KomsuDugumler[i])).SehirIsmi == mSehir)
                    return i;
            }
            return -1; // þehir bulunamadı
        }
```

IcerirSehir( ) Yöntemi: Bu yöntem aracılığıyla, aranan şehir düğümünün varlığı sorgulanıp, grafın oluşturulması aşamasında, düğüm ekleme veya çıkarma işleminde kullanılır. Söz konusu şehrin eklenip eklenmemesi veya tersi olarak çıkarılıp çıkarlımayacağına karar verilir.

```
 public bool IcerirSehir(SehirDugumTipi mSehir)
        {
            for (int i = 0; i < SehirDugumleri.Count; i++)
                if (mSehir.SehirDugumu.SehirIsmi ==
((SehirDugumTipi)(SehirDugumleri[i])).SehirDugumu.SehirIsmi)
                    return true;
            return false;
        }
```

SehirDugumuEkle ( ) Yöntemi: Şehrin graf üzerine eklenmesi sağlanır.

```
  public void SehirDugumuEkle(SehirDugumTipi mSehir)
        {
            if (!IcerirSehir(mSehir))
                SehirDugumleri.Add(mSehir);
        }
```

SehirDugumuCikar( ) Yöntemi: Şehrin grafdan çıkarılmasını sağlanır.

```
public void SehirDugumCikar(SehirDugumTipi mSehir)
     {
         if (IcerirSehir(mSehir))
             SehirDugumleri.Remove(mSehir);
     }
```

5.4. Algoritmanın Kodlanması İçin Platform Ve Programlama Dilinin Belirlenmesi

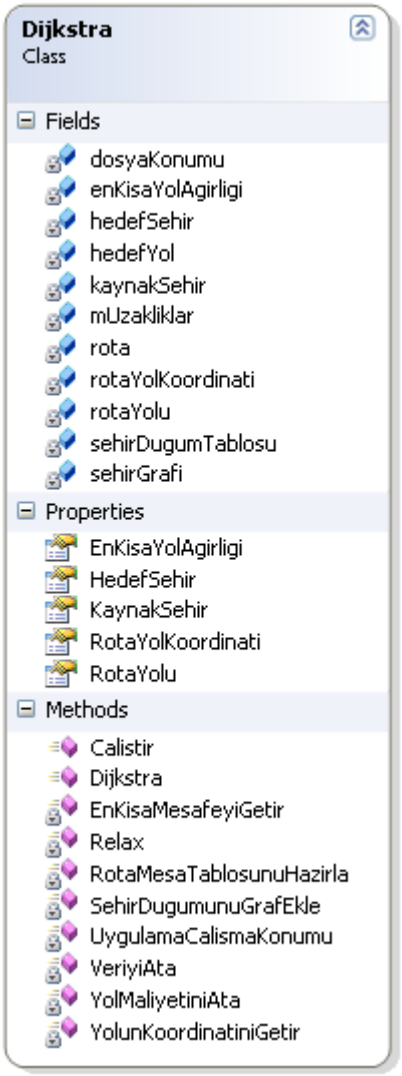
Uygulamanın erişilebilir olması, herhangi bir kurulum gerektirmemesi, makine ve platform bağımsız olması için Web tabanlı bir uygulama olmasına karar verilmiştir.

Uygulamanın gerçekleştirilebileceği teknolojiler ASP.Net, Java, PHP gibi teknolojiler sıralanabilir. Yazılımı çok daha kolay olması ve gücünden dolayı, C# dili kullanılarak bir ASP.NET uygulaması geliştirilmiştir.

Uygulama için Visual Studio 2005 aracı kullanılarak .Net 2.0 üzerinde kodlanmış, ayrıca şehirlerin ve rotanın harita üzerinde gösterilimi için Google Maps V2 API kullanılmıştır.

5.5. En Kısa Yol İçin Kullanılacak Algoritmanın Seçilmesi

Problemin çözümü için pek çok yöntem mevcuttur. Şehirler arasında negatif yol maliyetinin olmaması, kodlamanın kolay ve etkili yapılabilmesi, nesneye yönelimli metodoloji rahatlıkla kullanılabilmesi nedeniyle Dijkstra Algoritmasının kullanılmasına karar verilmiştir. Dijkstra algoritmasının kullanımı için Şekil 5.7'deki Dijkstra Class'ı oluşturulmuştur. Class içinde algoritmanın çalışması için gerekli olan yöntemleri: Verinin Database'den alınması , Grafın oluşturulması, Algoritmanın en kısa/en hızlı yol için çalıştırılması,Rotanın harita üzerinde gösterilmesi şeklindedir.



Şekil.5.7 Dijkstra Class

5.4.1. Verinin Okunması ve Grafın Oluşturulması

Dijkstra nesnesi, aralarındaki en kısa yolun belirlenmesi için verilen iki şehir parametresi ve en kısa veya en hızlı yolun belirlenmesi için toplam üç parametre alır. Nesnenin yapıcısı **VeriyiAta()** yöntemiyle şehir bilgilerini databaseden alarak SehirDugumu nesnesine yerleştirir.

```
public Dijkstra(string kaynakSehir, string hedefSehir,bool shortestpath)
        {
            hedefYol = shortestpath;
            this.HedefSehir = hedefSehir;
            this.KaynakSehir = kaynakSehir;
            EnKisaYolAgirligi = "";
```

```
            VeriyiAta();
            RotaMesaTablosunuHazirla(kaynakSehir);
        }
```

Veri tabanı üzerinden okunan şehir bilgilerini graf üzerine yerleştirilmesi **SehirDugumunuGrafEkle()** yöntemi ile yapılır.

```
  private void SehirDugumunuGrafEkle()
        {
            try
            {
   string source = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
UygulamaCalismaKonumu() + "\\App_Data\\ShortestPath.mdb";
                OleDbConnection conn = new OleDbConnection(source);
                conn.Open();
                string cmdTextSehir = "SELECT SEHIRISMI,ENLEM,BOYLAM " +
                              " FROM SEHIRLER  ORDER BY SEHIRISMI";
                OleDbCommand cmd = new OleDbCommand(cmdTextSehir);
                cmd.Connection = conn;
                OleDbDataAdapter da = new OleDbDataAdapter(new
OleDbCommand(cmdTextSehir, conn));
                DataTable SehirInfoDT = new DataTable();
                da.Fill(SehirInfoDT);
                SehirDugumTipi m;
                for (int i = 0; i < SehirInfoDT.Rows.Count; i++)
                {
      m = new SehirDugumTipi(new
Sehir(SehirInfoDT.Rows[i]["SEHIRISMI"].ToString(),
SehirInfoDT.Rows[i]["BOYLAM"].ToString(),
SehirInfoDT.Rows[i]["ENLEM"].ToString()));
                    sehirGrafi.SehirDugumuEkle(m);
                }
            }
            catch(Exception ex)
            {throw ex; }        }
```

Şehirler arasındaki maliyetlerin graf üzerine yerleştirilmesi **YolMaliyetiniAta( )** yöntemi yardımı ile yapılmaktadır.

```
private void YolMaliyetiniAta()
        {
            string kaynak = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
+ UygulamaCalismaKonumu() + "\\App_Data\\ShortestPath.mdb";

            OleDbConnection conn = new OleDbConnection(kaynak);
            conn.Open();
            string cmdText = "SELECT SEHIRLER.SEHIRISMI as startSehir,
Sehir_1.SEHIRISMI as endSehir, SEHIRLERARASIUZAKLIKLAR.MESAFE as
Cost,SEHIRLERARASIUZAKLIKLAR.ZAMANMALIYETI as timeCost" +
                           " FROM (SEHIRLER INNER JOIN
SEHIRLERARASIUZAKLIKLAR ON SEHIRLER.TRAFIKKODU=
```

```
SEHIRLERARASIUZAKLIKLAR.KAYNAKSEHIR) INNER JOIN SEHIRLER AS Sehir_1 ON
SEHIRLERARASIUZAKLIKLAR.HEDEFSEHIR = Sehir_1.TRAFIKKODU" +
                            " ORDER BY SEHIRLER.SEHIRISMI,
Sehir_1.SEHIRISMI";
            OleDbCommand cmd = new OleDbCommand(cmdText);
            cmd.Connection = conn;
            OleDbDataAdapter da = new OleDbDataAdapter(cmd);
            DataTable mSehirDT = new DataTable();
            da.Fill(mSehirDT);

            string m_startSehir, m_endSehir;
            int m_cost;

            for (int i = 0; i < mSehirDT.Rows.Count; i++)
            {
                m_startSehir = mSehirDT.Rows[i]["startSehir"].ToString();
                m_endSehir = mSehirDT.Rows[i]["endSehir"].ToString();
                if(hedefYol)
                    m_cost =
Convert.ToInt32(mSehirDT.Rows[i]["cost"].ToString());
                else
                    m_cost =
Convert.ToInt32(mSehirDT.Rows[i]["timeCost"].ToString());
                sehirGrafi.KenarEkle(m_startSehir, m_endSehir,
Convert.ToInt32(m_cost));
                }

            conn.Close();

        }
```

Şehir mailiyetlerine mümkün olan en büyük sayının yerleştirilmesi işlemi için **RotaMesaTablosunuHazirla ( )** yöntemi kullanılmaktadır.

```
  private void RotaMesaTablosunuHazirla(string start)
        {

            IEnumerator ma = sehirGrafi.GetEnumerator();
            string mSehirName = "";

            while (ma.MoveNext())
            {
                mSehirName =
((SehirDugumTipi)ma.Current).SehirDugumu.SehirIsmi;
                mUzakliklar.Add(mSehirName, Int32.MaxValue);
                rota.Add(mSehirName, null);
                sehirDugumTablosu.Add(mSehirName, ma.Current);
            };

            mUzakliklar[start] = 0;
        }
```

5.4.2. Algoritmanın Çalıştırılması

Arama sırasında en düşük maliyetli şehir düğümünün bulunması **EnKisaMesafeyiGetir ( )** yöntemi ile yapılır.

```
private SehirDugumTipi EnKisaMesafeyiGetir(Hashtable nodes)
        {

            // nodes kümesi içindeki en düşük maliyetli nodun bulunmasý
            int minDist = Int32.MaxValue;
            SehirDugumTipi minNode = null;
            foreach (SehirDugumTipi n in nodes.Values)
            {


                if (n != null &&
((int)mUzakliklar[n.SehirDugumu.SehirIsmi]) <= minDist)
                {
                    minDist = (int)mUzakliklar[n.SehirDugumu.SehirIsmi];
                    minNode = n;
                }
            }

            return minNode;

        }
```

Bölüm 4.3.1 de anlatılan relaxation işlemi **Relax( )** yötemi yardımı ile gerçekleştirilir.

```
     private void Relax(string uSehir, string vSehir, int cost)
        {
            int distTouSehir = (int)mUzakliklar[uSehir];
            int distTovSehir = (int)mUzakliklar[vSehir];

            if (distTovSehir > distTouSehir + cost)
            {
                // rota ve uzaklýklar güncelleniyor
                mUzakliklar[vSehir] = distTouSehir + cost;
                rota[vSehir] = uSehir;
            }
        }
```

Dijkstra için kullanılan algoritma kod bloğu aşağıdaki gibidir. Burada graf üzerindeki SehirDugumu teker teker ele alınarak buna en yakın şehir bulunur. Daha sonra bu şehire gelen tüm yollar yeni maliyete göre Relax( ) yöntemi yardımıyla gözden geçirilir ve değerleri güncellenir. Bu işlem incelenecek olan tüm şehirler bitene kadar yani Q kümesi boşalana kadar devam eder.

```
            while (sehirDugumTablosu.Values.Count > 0)
            {
                SehirDugumTipi u = EnKisaMesafeyiGetir(sehirDugumTablosu);

                sehirDugumTablosu.Remove(u.SehirDugumu.SehirIsmi);

                if (u != null && u.KomsuDugumler != null)
                    for (int i = 0; i < u.KomsuDugumAdeti; i++)
                        Relax(u.SehirDugumu.SehirIsmi,
((Sehir)u.KomsuDugumler[i]).SehirIsmi,
Convert.ToInt32(u.KomsuDugumUzakliklari[i].ToString()));          // her
kenar için relax işlemi
            }     // Q kümesi boş olana kadar devam et
```

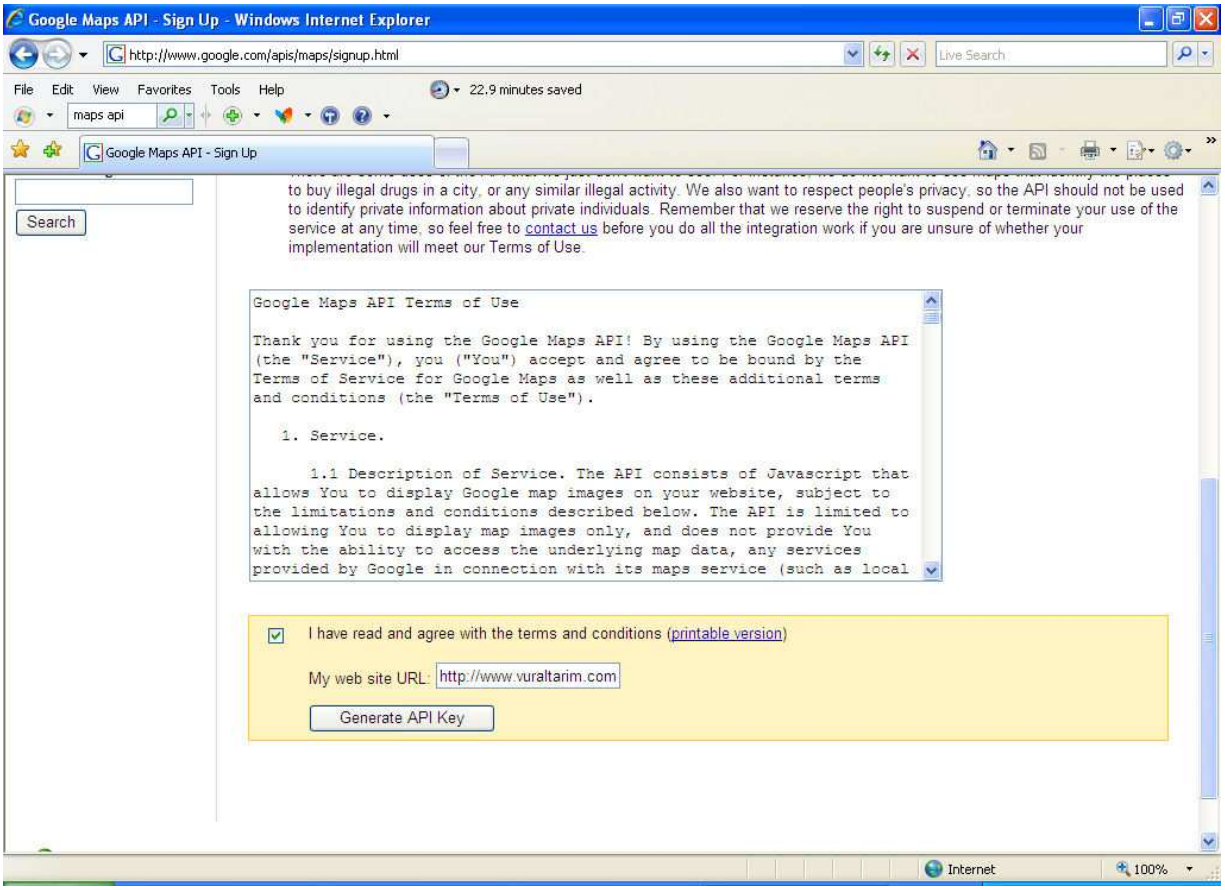
En kısa yol bulunduktan sonra elde edilen rota geriye doğru gidilerek yazdırılmak üzere önyüze döndürülür.

```
  public  void Calistir()
        {
            /****  DIJKSTRA ****/
            while (sehirDugumTablosu.Values.Count > 0)
            {
                SehirDugumTipi u = EnKisaMesafeyiGetir(sehirDugumTablosu);

                sehirDugumTablosu.Remove(u.SehirDugumu.SehirIsmi);

                if (u != null && u.KomsuDugumler != null)
                    for (int i = 0; i < u.KomsuDugumAdeti; i++)
                        Relax(u.SehirDugumu.SehirIsmi,
((Sehir)u.KomsuDugumler[i]).SehirIsmi,
Convert.ToInt32(u.KomsuDugumUzakliklari[i].ToString()));          // her
kenar için relax işlemi
            }     // Q kümesi boş olana kadar devam et
```

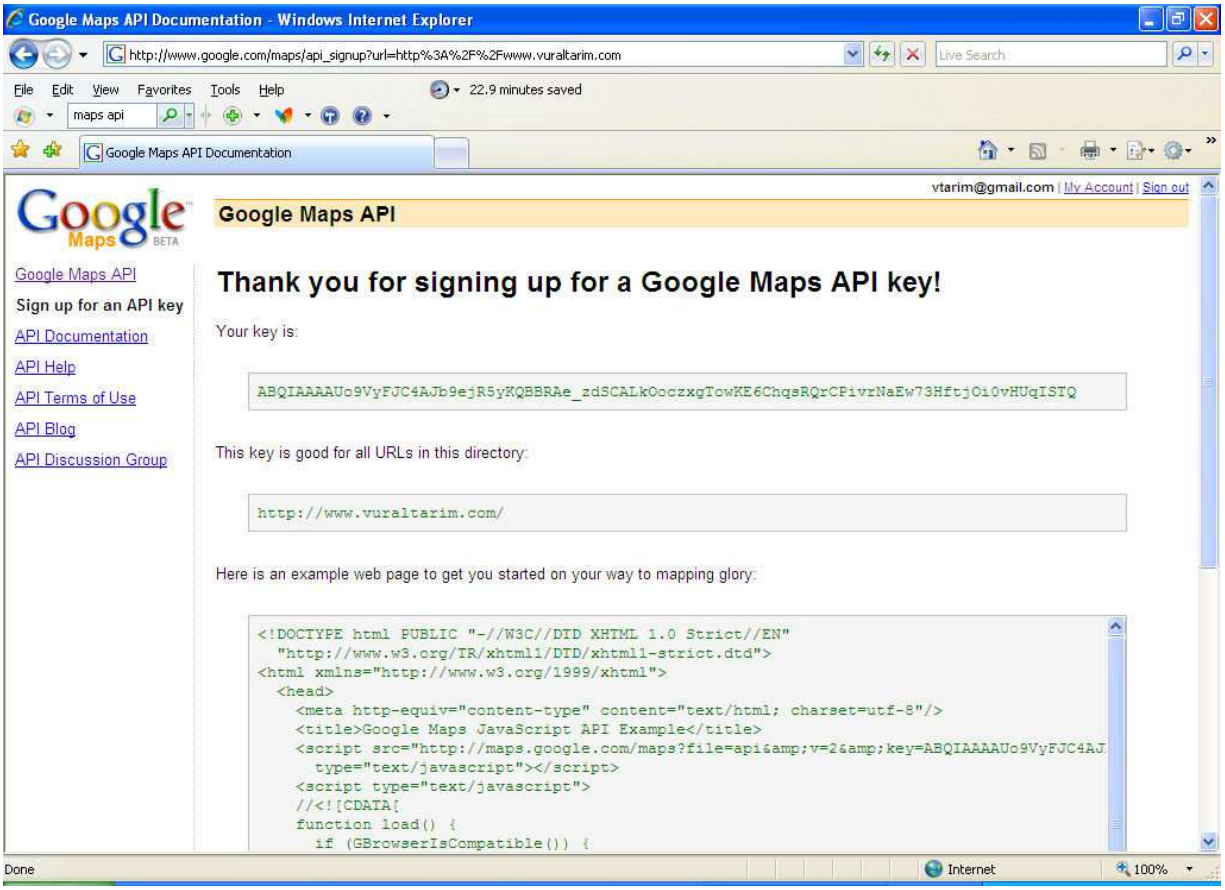
```
            /**** DIJKSTRA ****/

            Stack traceBackSteps = new Stack();
            string temp = " ";
            string current = HedefSehir, prev = null;
            do
            {
                prev = current;
                current = (string)rota[current];
                //temp = " - " + prev;
                temp = prev;
                traceBackSteps.Push(temp);
            } while (current != KaynakSehir);


            EnKisaYolAgirligi = mUzakliklar[HedefSehir].ToString();
            temp = KaynakSehir;
            traceBackSteps.Push(temp);
            RotaYolu           = new StringBuilder(30 *
traceBackSteps.Count);
            RotaYolKoordinati = new StringBuilder(30 *
traceBackSteps.Count);


            string mSehirName = "";
            string pSehirName = "";
            string mSehirNameCoord = "";
            Sehir  mSehirCoord ;
            int mCount = 0;

            #region Path Geo
            if (System.IO.File.Exists(dosyaKonumu))
                System.IO.File.Delete(dosyaKonumu);
            System.IO.File.Create(dosyaKonumu);
            #endregion Path Geo

            while (traceBackSteps.Count > 0)
            {
                mSehirName = (string)traceBackSteps.Pop();
                mSehirNameCoord = mSehirName;
                if (mCount > 0)
                    mSehirName = "->" + mSehirName;
                RotaYolu.Append(mSehirName);
                mSehirCoord =
sehirGrafi.SehirBulIsmiIle(mSehirNameCoord).SehirDugumu;
                mSehirName = mSehirNameCoord + '&' + mSehirCoord.Boylam +
'&' + mSehirCoord.Enlem + '$';
                RotaYolKoordinati.Append(mSehirName);
                if (pSehirName != "" && pSehirName != mSehirName)
                {
                    YolunKoordinatiniGetir(pSehirName, mSehirName);
                }
                pSehirName = mSehirName;
                mCount++;
            }
        }
```

5.4.3. Harita Üzerinde Rotanın Gösterilmesi

Bulunan rota bilgisi, enlem ve boylam bilgileri Google Maps API kullanılarak Türkiye Haritası üzerinde gösterilmektedir. Api nin kullanılması için öncelikle http://www.google.com/apis/maps/signup.html adresinden gerekli olan anahtarın (key) alınmış, bu anahtar uygulamada gerekli olan sayfalara konmuştur.



Şekil.5.8 Google Maps API anahtar alma sayfası



Şekil.5.9 Oluşturulan anahtar

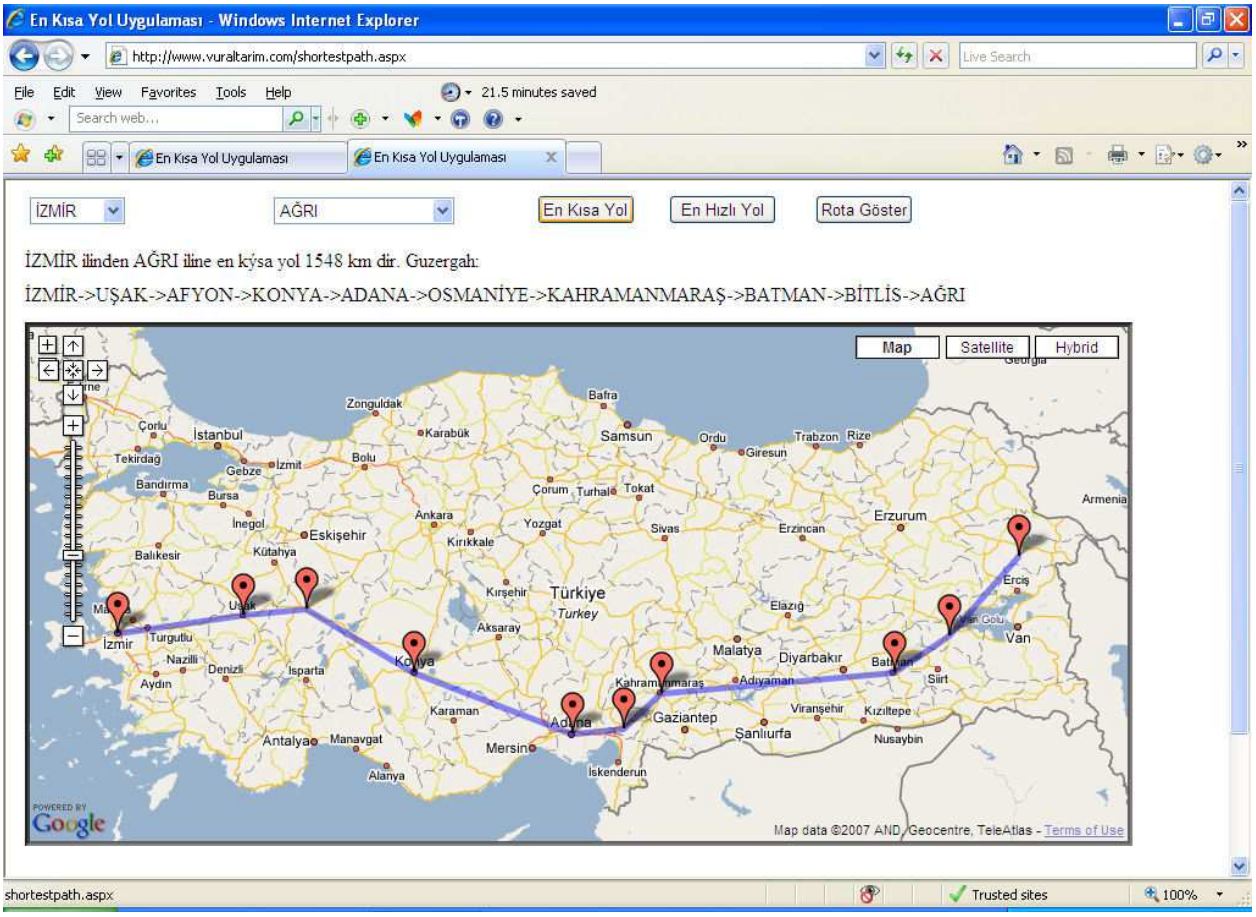
```
<head runat="server">
     <meta http-equiv="content-type" content="text/html";  />
    <title>En Kısa Yol Uygulaması</title>
     <style type="text/css">
        v\:* {behavior:url(#default#VML);}
      </style>
    <script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=ABQIAAAAUo9VyFJC4
AJb9ejR5yKQBBQK8q4CNLz-8IIGWaeUxD-5j-Ie2xTACxDEtVqzpH2o9tdwjWJfgbNSDw"

      type="text/javascript"></script>
    <script type="text/javascript">
```

Şekil.5.10 Oluşturulan anahtarın kullanıması

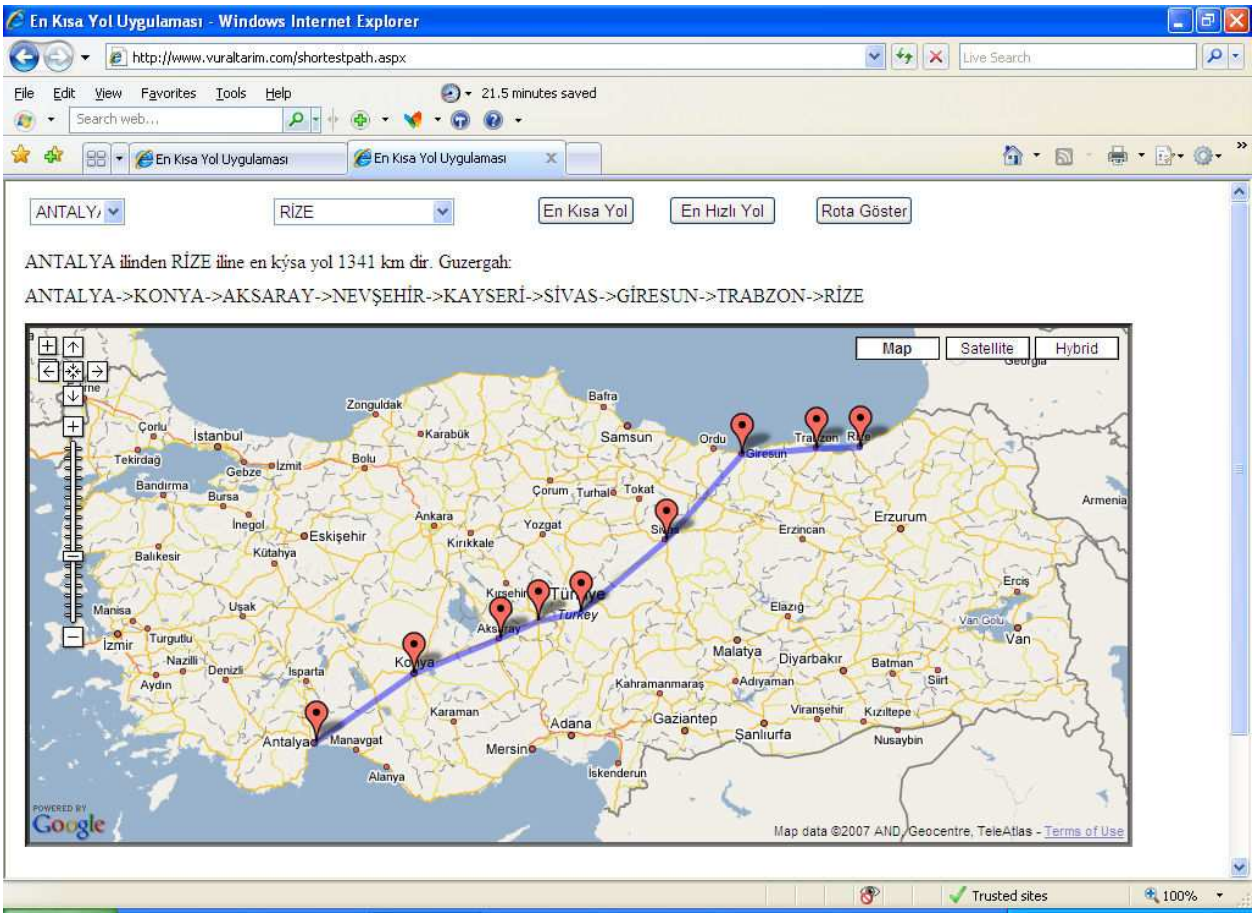
Google Maps Api yardımıyla Dijkstra sınıfında bulunan yolun ön yüzde gösterilmesi için hazırlanmış olan JavaScript kod bloğu aşağıdadır.

```
function RotaGoster()
      {
      for(j=0;j<i;j++)
      {
          map.removeOverlay(gmarker[j]);
          points[j]=0;
      }
      i=0;
      var   str = document.getElementById("txtPaths").value;
        if(str==null)
            return;
        var routeArry=str.split("$");
        var pts = [];
        var lng;
        var lat;
        var cname;
        if(routeArry.length>0)
        {
        for (var i=0;i<routeArry.length;i++)
              {
              pts = routeArry[i].split("&");
              cname = pts[0];
              lng = pts[1];
              lat = pts[2];
              adresGoster("'" + cname + "'",lat,lng);
              }

          map.addOverlay(new GPolyline(points));
       }
      }


  function adresGoster(address,lat,lng)
  {
      if(address != null && address!=' ' && lng>0 && lat>0)
      {

        point = new GLatLng(lat,lng);
        marker= new GMarker(point)
        map.addOverlay(marker);
        gmarker[i]=marker;
        points[i]=point;
        i++;
      }
  }
```

Uygulamanın oluşturduğu örnek ekran çıktıları aşağıdaki gibidir.



Şekil.5.10 Örnek 1



Şekil.5.11 Örnek 2

SONUÇLAR

Çalışmada verilen iki şehir arasındaki en kısa yolun belirlenmesi problemi incelenmeye ve konuyla ilgili bir uygulama geliştirmeye çalışılmıştır. Çalışmada, öncelikle graf teorisinin temel kavramları ele alınmış, problemin graf teorisiyle modellenmesi ve problemin algoritmik çözümü ile ilgili çalışmalar incelenmiştir. En kısa yol algoritmaları hakkında yapılmış çalışmalar ve bunların dayandığı temeller hakkında bilgiler verilmeye çalışılmıştır. İncelenmiş olan algoritmalar içerisinden, Dijkstra algoritması kullanılarak, Türkiye karayolları için ASP.NET teknolojisi aracılığıyla bir Web uygulaması geliştirilmiştir. Şehirlerin ve bulunan rotanın gösterimi için Google Maps API si kullanılarak, rotanın görselleştirmesi sağlanmıştır. Uygulamaya en kısa yanında en hızlı yolun bulunması için gerekli fonksiyonlar eklenmiş ve çalışması sağlanmıştır.

Uygulamanın daha efektif kullanılabilmesi için anlık yol durumunun alınarak verilerin güncellenmesi sağlanabilir. Fakat Türkiye Karayolları Müdürlüğü tarafından sağlanan bu türlü bulunamamıştır. Müdürlük tarafından sadece, statik bir şekilde yol yapım durumu bilgisi verilmektedir. Yol durumu ile ilgili bilgiler Web Servisleri aracılığıyla verildiğinde uygulama o anki duruma göre en az maliyetli ve en hızlı yol durumunu verebilecek şekilde geliştirilebilir. Ayrıca uygulamanın çok daha karmaşık graf modellerinde daha etkili çalışabilmesi için "graf parçalama yöntemi" kullanılabilir. Böylece gerekli olan iş yükü işlemciler arası bölünerek çalışma zamanından tasarruf edilebilir.

EK-1 : ŞEHİR ENLEM VE BOYLAM BİLGİLERİ

TRAFIKKODU	SEHIRISMI	ENLEM	BOYLAM
1	ADANA	36.99998	35.32436
2	ADIYAMAN	37.76475	38.27856
3	AFYON	38.76376	30.54034
4	AĞRI	39.49948	43.36688
5	AMASYA	40.64991	35.83532
6	ANKARA	39.93	32.85
7	ANTALYA	36.89272	30.70948
8	ARTVİN	41.18277	41.81829
9	AYDIN	37.85604	27.84163
10	BALIKESİR	39.64837	27.88261
11	BİLECİK	40.15013	29.98306
12	BİNGÖL	38.88535	40.49829
13	BİTLİS	38.3938	42.12318
14	BOLU	40.73948	31.61156
15	BURDUR	37.72691	30.28888
16	BURSA	40.19619	29.07496
17	ÇANAKKALE	40.15531	26.41416
18	ÇANKIRI	40.60715	33.62114
19	ÇORUM	40.55437	34.96372
20	DENİZLİ	37.77652	29.08639
21	DİYARBAKIR	37.91441	40.23063
22	EDİRNE	41.68181	26.56227
23	ELAZIĞ	38.68097	39.2264
24	ERZİNCAN	39.73993	39.5045
25	ERZURUM	39.90445	41.29182
26	ESKİŞEHİR	39.7843	30.51922
27	GAZİANTEP	37.06877	37.38958
28	GİRESUN	40.91281	38.38953
29	GÜMÜŞHANE	40.46438	39.484
30	HAKKARİ	37.56842	43.75143
31	HATAY	36.41827	36.24744
32	ISPARTA	37.76477	30.55656
33	MERSİN	36.8042	34.63857
34	İSTANBUL	41.1	29
35	İZMİR	38.43343	27.14542
36	KARS	40.59267	43.07783
37	KASTAMONU	41.38871	33.78273

TRAFIKKODU	SEHIRISMI	ENLEM	BOYLAM
38	KAYSERİ	38.73122	35.47873
39	KIRKLARELİ	41.74291	27.22609
40	KIRŞEHİR	39.14249	34.17091
41	KOCAELİ	40.77869	29.90947
42	KONYA	37.87759	32.48177
43	KÜTAHYA	39.41927	29.97987
44	MALATYA	38.35519	38.30946
45	MANİSA	38.6191	27.42892
46	KAHRAMANMARAŞ	37.58583	36.93715
47	MARDİN	37.33	40.78
48	MUĞLA	37.22019	28.3672
49	MUŞ	38.73032	41.50216
50	NEVŞEHİR	38.62442	34.72397
51	NİĞDE	37.97646	34.69376
52	ORDU	40.98388	37.87641
53	RİZE	41.02005	40.52345
54	SAKARYA	40.77378	30.40535
55	SAMSUN	41.29278	36.33128
56	SİİRT	37.94429	41.93288
57	SİNOP	42.03	35.15
58	SİVAS	39.74766	37.01788
59	TEKİRDAĞ	40.98544	27.51663
60	TOKAT	40.30627	36.56332
61	TRABZON	41.00145	39.7178
62	TUNCELİ	39.1058	39.55848
63	ŞANLIURFA	37.15915	38.79691
64	UŞAK	38.6823	29.40819
65	VAN	38.49	43.4
66	YOZGAT	39.81808	34.81469
67	ZONGULDAK	41.45641	31.79873
68	AKSARAY	38.36869	34.03698
69	BAYBURT	40.25517	40.22488
70	KARAMAN	37.17593	33.22875
71	KIRIKKALE	39.84682	33.51525
72	BATMAN	37.88117	41.13509
73	ŞIRNAK	37.51873	42.46976
74	BARTIN	41.64152	32.34558
75	ARDAHAN	41.11048	42.70217

TRAFIKKODU	SEHIRISMI	ENLEM	BOYLAM
76	IĞDIR	39.9193	44.0655
77	YALOVA	40.65917	29.27454
78	KARABÜK	41.2061	32.62035
79	KİLİS	36.7184	37.12122
80	OSMANİYE	37.06805	36.26159
81	DÜZCE	40.84385	31.15654

EK-2: ŞEHİRLER ARASI YOL MESAFE BİLGİLERİ

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
1	31	192
1	33	69
1	42	356
1	51	205
1	68	265
2	21	205
2	27	150
2	31	320
2	44	185
2	63	109
2	80	244
3	6	257
3	15	171
3	20	226
3	26	145
3	32	170
3	42	223
3	43	101
3	64	117
4	13	234
4	25	183
4	36	215
4	49	245
4	65	232
4	76	143
5	14	409
5	19	92
5	24	365
5	51	441
5	55	131
5	57	263
5	60	114
5	66	196
5	69	460
6	14	191
6	18	131
6	19	244
6	26	233

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
6	40	186
6	42	258
6	67	268
6	68	225
6	71	77
6	78	215
7	15	122
7	20	221
7	32	128
7	33	487
7	42	323
7	48	313
7	70	377
8	25	203
8	53	159
8	69	319
8	75	114
9	20	126
9	35	130
9	45	156
9	48	99
10	11	245
10	16	151
10	17	210
10	35	173
10	43	227
10	45	137
10	64	224
11	14	216
11	16	94
11	26	80
11	43	110
11	54	102
11	64	251
11	77	126
12	21	144
12	23	144
12	24	275

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
12	25	180
12	49	114
12	62	145
12	63	325
12	69	304
13	21	209
13	49	83
13	56	97
13	65	168
13	72	135
14	18	235
14	26	296
14	37	245
14	54	114
14	55	475
14	67	159
14	78	134
14	81	45
15	20	150
15	32	51
15	48	242
15	64	172
16	17	271
16	26	149
16	41	132
16	43	179
16	54	158
16	64	310
16	77	69
17	22	217
17	35	319
17	45	330
17	59	188
18	19	156
18	37	114
18	71	106
18	78	195
19	37	196

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
19	55	175
19	57	312
19	60	178
19	66	104
19	71	167
19	78	307
20	45	206
20	48	146
20	64	152
21	23	151
21	44	252
21	47	95
21	49	259
21	56	186
21	63	181
21	72	99
22	34	228
22	39	62
22	59	137
23	44	191
23	62	133
23	63	332
24	25	188
24	28	298
24	29	145
24	44	364
24	58	246
24	60	305
24	62	130
24	69	154
25	36	203
25	49	246
25	53	354
25	69	124
25	75	243
25	76	294
26	42	338
26	43	78

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
27	31	197
27	46	180
27	63	137
27	79	68
27	80	122
28	29	162
28	52	44
28	58	385
28	61	137
29	58	371
29	61	100
29	69	78
30	65	203
30	73	191
31	44	380
31	46	176
31	79	148
31	80	128
32	33	587
32	42	264
32	64	171
32	70	371
33	42	349
33	51	198
33	68	258
33	70	236
34	39	209
34	41	111
34	59	132
35	45	36
35	64	211
36	69	325
36	75	91
36	76	138
37	55	312
37	57	194
37	74	181
37	78	111

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
38	40	134
38	44	341
38	46	273
38	50	104
38	51	128
38	58	194
38	66	175
39	59	118
40	50	91
40	66	112
40	68	109
40	71	112
41	54	37
41	77	65
42	51	255
42	68	148
42	70	119
42	71	300
43	45	316
43	64	141
44	46	223
44	58	247
44	62	234
45	64	193
46	58	345
46	80	101
47	56	225
47	63	188
47	72	148
47	73	199
49	72	218
50	51	80
50	66	198
50	68	75
51	66	267
51	68	121
51	70	189
52	55	165

KAYNAKSEHIR	HEDEFSEHIR	MESAFE
52	58	341
52	60	233
53	61	75
53	69	353
54	77	102
54	81	69
55	57	168
55	60	242
56	65	265
56	72	87
56	73	97
58	60	108
58	66	224
60	66	207
60	69	400
61	69	178
62	69	260
63	73	366
65	73	362
65	76	225
66	71	142
67	74	89
67	78	173
67	81	114
68	70	211
69	75	359
74	78	84

KAYNAKCA

Price, W.L. Graphs and Networks, London Butterworth, First Edition,1971.

Baase, Sara & Van Gelder, Allen Computer Algorithms, Addison-Wesley, Thrid Edition 2000.

Çölkesen, Rıfat Veri Yapıları ve Algoritmalar, Papatya Yayıncılık, 1. Basım 2002.

Ocak, Asuman Design and Implementation Traveling Salesman Problem to Turkey Roadway Graph, Beykent University, 2003 Undergraduate Thesis.

Grimaldi, Ralph Discrete and Combinatorial Mathematics, Pearson, Fifth Edition 2004.

Internet Kaynakları

http://w3.gazi.edu.tr/~akcayol/files/AAL12Greedy2.pdf (15.05.2007)

http://www.graphtheory.com (15.05.2007)

http://en.wikipedia.org/wiki/Search_algorithm (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Bellman-Ford_algorithm (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Greedy_algorithm (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Johnson%27s_algorithm (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Shortest_path_problem (01.02.2007-05.06.2007)

http://en.wikipedia.org/wiki/Sparse_graph (01.02.2007-05.06.2007)

http://www.cs.berkeley.edu/~adamc (20.04.2007)

http://www.cs.binghamton.edu/~zhongfei/cs333_s07/dijkstra.ppt (18.03.2007)

http://www.cs.sunysb.edu/~skiena/combinatorica/animations/dijkstra.html (18.03.2007)

http://www.cs.ucsb.edu/~teo/publications/GRAPH.html (22.04.2007)

http://www-cs-students.stanford.edu/~amitp/Articles/AStar1.html (26.05.2007)

http://www.mapsofworld.com/lat_long/turkey-lat-long.html (22.05.2007)

ÖZGEÇMİŞ

Sivas ili Hafik ilçesi doğumluyum. İlkokulu Seyrantepe İlkokulunda, ortaokul ve liseyi ise Yeni Levent Lisesi'nde tamamladıktan sonra, Lisans öğrenimimi Yıldız Teknik Üniversitesi, Fen-Edebiyat Fakültesi, Matematik Bölümü'nde 1993 yılında tamamladım. Daha sonra İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü Mühendislik Bilimleri-Sistem Analizi Anabilim Dalı'nda yüksek lisans öğrenimine başladım. Buradan da 2001 yılında mezun oldum. 2006 yılında da, Beykent Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans eğitimine başladım. Yabancı dilim İngilizce olup, evli ve bir çocuk babasıyım.

Aday: Vural TARIM