T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

ISE 402 BİTİRME ÇALIŞMASI

ŞEBEKE (AĞ) ANALİZİ YÖNTEMİ

B161210309 - OĞUZHAN ÖZDEMİR
G161210305- TALHA ÇERÇİ
G151210106 – İLAYDA ÖZDEMİR

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Tez Danışmanı : Doç.Dr. Nilüfer YURTAY

2019-2020 Bahar Dönemi

T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

ŞEBEKE (AĞ) ANALİZİ YÖNTEMİ

ISE 402 - BİTİRME ÇALIŞMASI

Adı SOYADI

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Bu tez .. / .. / … tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

…….	…….	…….
Jüri Başkanı	**Üye**	**Üye**

ÖNSÖZ

Şebeke (ağ) model sistemlerinde varolan kaynakların (işgücü, zaman, kapital, süreçler, hammaddeler, kapasite, ekipman gibi) en verimli şekilde kullanılarak belirli amaçlara (maliyet en azaltılması, kâr en çoklanması, kapasite kullanımının en yükseltilmesi ve verimliliğin en çoklanması gibi) ulaşmayı sağlayan bir teknoloji olarak tanımlanmaktadır.

Şebeke (ağ) modelleme ve çözümleme iki önemli bileşen olarak nitelendirilmektedir.
Modelleme gerçek yaşamda karşılaşılan problemin matematiksel olarak ifade edilmesi;
çözümleme ise bu modeli sağlayan en iyi çözümün elde edilmesini kapsamaktadır.
Şebeke (ağ) model teknolojisinin gelişiminde araştırmacılar öncelikli olarak modellemeyle
ilgilenmişlerdir.

Şebeke (ağ) modellerinin özellikle ekonomi sistemlerde kullanılması ve üretim / dağıtım sistemlerinde karşılaşılan problemlerin birçoğunun şebeke (ağ) problemi olarak modellenmesine rağmen şebeke (ağ) modellerinin teorik özelliklerinin araştırması ve genel çözüm algoritmalarının geliştirmesi halen devam etmektedir.

Özet:

Bu çalışmada farklı mühendislik disiplinlerinde yöneylem araştırması tekniklerinin kullanılabileceğine dikkat çekilmesi amaçlanmıştır. Bu amaç doğrultusunda profesyonel iş yaşamından ve akademik alandan örneklere yer verilmiştir. Çalışmada da ifade edildiği üzere, yöneylem araştırmaları matematiksel modellere dayanır, ekonomik etkinliği değerlendirme kriteri olarak ele alır ve yöneticilerin karar vermelerini kolaylaştırmayı amaçlar. Yöneylem teknik ve algoritmaları pek çok alanda uygulama imkânına sahip olup, uygulayıcısına etkin çözümler üretebilmektedir. Söz konusu bu teknikler aracılığıyla, projeleri yönetenler tüm kaynakları en etkin ve verimli bir şekilde kullanarak, olabilecek en kısa sürede ve en ekonomik şekilde çalışmalarını hayata geçirebilmektedir. Her mühendislik probleminin de farklı bir ele alınış biçimi ve yapısı olduğu bilinmektedir. Bu durumda her tekniğin her mühendislik problemine uygulanamayacağı söylenebilir. Bu sebeple bir mühendislik probleminin çözümü için kullanılacak olan uygun yöneylem tekniğinin seçimi büyük önem taşımakta ve farklı uzmanlık alanlarına sahip bir ekiple problemlerin çözümüne yaklaşılması, diğer bir ifadeyle disiplinlerarası bir çalışmanın yürütülmesi gerekmektedir.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

N1X	: Başlangıç Noktasının X Ekseni
N1Y	: Başlangıç Noktasının Y Ekseni
N2X	: Bitiş Noktasının X Ekseni
N2Y	: Bitiş Noktasının Y Ekseni
X_eksen	: X Eksenindeki Koordinata Ekleme
Y_eksen	: Y Eksenindeki Koordinata Ekleme
Btn	: Buton
EKM	: En Küçük Maliyetli
Txt_Nereden	: Rotanın Nereden Başlayacağını Belirler
Txt_Nereye	: Rotanın Nerede Biteceğini Belirler
Min	: Minimum
Max	: Maksimum
Min_İndex	: Minimum İndeks
Spt	: Sepet
EBM	: Kritik Yol Analizi
CPM	: Kritik Yol Yöntemi
TE	: En Erken Olay Zamanı
TL	: Çalışma Boyunca En Geç Olay Zamanı
ES	: Faaliyetlerin En Erken Başlama Süresi
EF	: En Erken Bitme Süresi
LF	: En Geç Bitme Süresi
LS	: En Geç Başlama Süresi
EB	: Erken Başlangıç Zamanı
GB	: En Geç Başlangıç Zamanı
ES	: En Erken Sonlanma Zamanı
GS	: En Geç Sonlanma Zamanı

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

BÖLÜM 1. GİRİŞ

Herbert Alexander Simon, ekonomi dalında Nobel ödülü almış bir bilim adamı, ***The Sciences of The Artificial*** **(Yapayın Bilimleri)** adlı kitabında, Yapay tanımlamasıyla insan yapısı sistemleri ve bunların önemli bir bölümü olan mühendislik yapılarını söz konusu ediyor. Ve bu incelemesinde geleneksel mühendislikte bu yapıların doğayla 'interface' (arakesiti)'nde bilimsel olduğunu, fakat insanla arakesiti'nde bilimsel olmadığını söylüyor. Malum bu yapay sistemler insan içeriyor ve insana hizmet için yapılıyor. Onun için insanla arakesiti doğayla olduğu kadar, hatta belki de daha önemli. Ancak arkadan ekliyor, 20. asırda bu alanda da bilimsel gelişmeler var. Karar ve Değer Teorileri, Oyunlar Teorisi, Organizasyon Teorileri, 'Management Science' (Yönetim Bilimi), tasarım bilimi gibi gelişmeler var. Bunlar arasında önemli biri de ***'Operational Research'*** **(Yöneylem Araştırması).** İkinci Dünya Savaşı yıllarında , İngilizler denizin altından sessizce yaklaşan Alman denizaltlarını tespit edemiyor, tespit ettiklerinde ise uçaklardan atılan bombaların denizin altına ulaşamıyor olması nedeniyle onları vuramıyorlardı. İngiliz mühendisler konuyu tekrardan ele aldı ve Alman denizaltlarının torpidolarını ateşleme süresini tekrar hesapladılar. Uçaklara yüklenen bombalar yeni hesaplara göre üretildi. Daha derine inip daha geç patlayan bombalar sayesinde İngilizler amaçlarına ulaştılar. İşte ***'Operational Research'*** adı verilen bilim dalı bu şekilde ortaya çıktı. Yani konunun yeniden, farklı düşünceler ve farklı hesaplamalarla ele alınmasıyla. ***Yöneylem Araştırması***, belirli kısıtların olduğu bir durumda, belirli bir amaca yönelik en uygun çözümün bulunması için geliştirilmiş bir yöntem olmak ile birlikte bir organizasyon içinde operasyonların koordinasyonu ve yürütmesi ile ilgili dünyanın gerçek karmaşık sorunları için fikir üretmede matematiksel modelleme, istatistik ve algoritma gibi bilimsel yöntemleri kullanan disiplinlerarası bir bilimdir. Bu yeni yaklaşımda önemli olan şey, çözüm için doğru soruyu sormak ve ona doğru bir bakışla yaklaşmaktı.

Bugün Türkiye dahil dünyadaki mühendislik fakültelerine ders olarak okutulan bu disiplini Türkiye'ye ilk getiren isim ***Prof.Dr Halim Doğrusöz***'dür.

1.1-Amaç

Yöneylem Araştırması, insan, makina, para ve malzemeden oluşan endüstriyel, ticari, resmi ve askeri sistemlerin yönetiminde karşılaşılan sorunlara, bilim aracılığıyla çözüm sağlanmasıdır. Soruna bilimsel olarak en uygun çözümü sağlamak için bu bilimi kullandıktan sonraki hedef organizasyonun performansını iyileştirmek ve optimize etmektir. Bu çözüm yapılırken, sistemin şans ve risk ölçüsünü de içeren; ve seçenekli karar, strateji ve kontrollerin sonuçlarını tahmin ve karşılaştırmaya yarayan, bilimsel bir model geliştirilir. Böylece, ***Yöneylem Araştırması***'nda sorun çözümlenirken, yarar ilkesine bağlı kalınarak bilim kullanılmış olur.

1.2-Gereklilikler

Yöneylem Araştırması, çalışmalarının 3 temel özelliği vardır:

- Sistem yaklaşımı özelliği (Bir olayı tümüyle ele almak)
- Disiplinler arası yaklaşım özelliği (Farklı disiplinlerden oluşan bir ekip kurmak)
- Bilimsel yöntem kullanma özelliği
- Bu üç temel özellik dışında yöneylem araştırmasının özellikle günümüzde kazandığı en önemli özelliği,
- Yüksek hızlı bilgisayar kullanımıdır.

1.3-Yöntem

Yöneylem araştırmacıları tarafından kullanılan öncelikli araçlar istatistik, optimizasyon, rassallık, oyun kuramı, kuyruk kuramı, çizge kuramı, karar analizi ve simülasyondur.Bu alanların sayısal niteliğinden dolayı yöneylem araştırması bilgisayar bilimleriyle de ilgilidir.Dolayısıyla yöneylem araştırmacıları özel olarak yazılmış ya da hazır yazılımları kullanırlar.***Yöneylem araştırması,*** spesifik elemanlara yönelmektense sistem genelini ele alarak bütünüyle geliştirme yeteneğiyle diğerlerinden ayrılır.Yeni bir problemle karşılaşan yöneylem araştırmacısının hangi tekniklerin sistemin doğasına, geliştirme hedeflerine, zaman ve hesap gücü kısıtlarına en yatkın olduğuna karar vermesi beklenir.Bu ve diğer nedenlerden ötürü insan etkeni yöneylem araştırması için yaşamsaldır. Öteki herhangi araçlar gibi yöneylem araştırması teknikleri problemleri kendi başlarına çözemezler. Herhangi bir sorunu yöneylem araştırması yöntemiyle çözümleyebilmek için, farklı disiplinlerden bir araştırma ekibinin oluşturulması gerekir.Yöneylem araştırmasındaki disiplinlerarası yaklaşım, çalışmaları konu alan problemlerin çok yönlü ve karmaşık yapıdaki sistemlerle ilgili olmasından kaynaklanmaktadır. Bu nedenle problemin çözümünde belli bir disiplinin katkısı yetersiz kalabilmektedir. Yöneylem araştırması disiplinlerarası yaklaşımı benimsemekle, mümkün olduğu

kadar çok etkileşimi saptamaya çalışmaktadır. Karar verme sürecinde karar vericilerin kullanabileceği yöntemler 2 ana gruba ayrılır:

- Geleneksel veya klasik yöntemler (doğaüstü güçler, sağduyu, sezgi, mantıksal yöntemler vb.)
- Bilimsel yöntem

Bilimsel yöntem şu aşamalardan oluşur:

- Problemin tanımı
- Hipotezin geliştirilmesi
- Veri ve bilgilerin toplanması
- Deneyler yoluyla hipotezin test edilmesi
- Hipotez hakkında sonuçlara varılması

Yöneylem Araştırmasının Karar Süreci:

- Problemi tanımak ve formüle etmek
- İncelenen sistemi temsil etmek üzere matematik bir modelini kurmak
- Modelden hareket ederek bir çözüm bulmak
- Modeli ve modelden elde edilen çözümü kontrol etmek
- Çözümü uygulamaktır

1.4-Kapsam

- Yöneylem araştırmasının kullanıldığı birkaç uygulama örneği:
- Verimli bir madde akışı için fabrikanın yerleşim planını tasarlarken
- Yol trafiği yönetiminde
- Bilgisayar çipinin üretim zamanını ve maliyetini azaltmak için tasarımını değiştirirken
- Tedarik zincirinde bitmiş ürünler için belirsiz talep söz konusu olduğunda hammadde, ürün akışını yönetirken
- İnsanla yönetilen süreçlerin robotlaştırılması ve otomasyonunda
- Ucuz madde, işgücü, yer ve diğer üretkenlik girdilerinde avantaj sağlamak için operasyon süreçlerini küreselleştirirken
- Nakliyat ve taşıma yönetiminde
- Ağ veri trafiğinde: bunlar kuyruk modeli olarak da bilinir.
- Petrol rafinerilerinde hammaddeleri karıştırırken

Mühendislik Alanı	Uygulama Niteliği	Kullanılan Teknik
Haberleşme Sistemleri Mühendisliği	Filtre tasarımı	Oyun Teorisi
	Bilgisayar iletişim ağlarının tasarımı ve analizi	Simülasyon
	Bilgisayar iletişim ağlarının tasarımı ve analizi	Kuyruk Teorisi ve Rassal Süreçler
	Çoklu erişim protokol sorunu	Dinamik Programlama
İnşaat Mühendisliği	Betonarme yapıların optimal tasarımı	Matematiksel programlama
	Çelik donatı miktarının optimizasyonu	Doğrusal programlama
	Sürekli titreşen yapılar için optimal kiriş tasarımı	Doğrusal programlama
Kimya Mühendisliği	Tek bileşenli soğutucu tasarımı	Karma tamsayılı doğrusal olmayan programlama
Uzay Mühendisliği	Uzay yapılarının tasarımı	Doğrusal olmayan programlama
Elektrik- Elektronik Mühendisliği	Elektrik devrelerinde filtre tasarımı	Doğrusal olmayan programlama
	Termik santrallerde yakıt maliyetlerinin minimizasyonu	Doğrusal olmayan programlama
Bilgisayar Mühendisliği	Hata Toleranslı Bilgisayar Sistemleri Tasarımı	Markov zincirleri

Kaynak: Levary (2001)'den uyarlanmıştır.

Tablo1.1:Yöneylem araştırması tekniklerinin çeşitli mühendislik disiplinlerindeki uygulamaları

Model Tipi	Yöntem	Tanım ve/veya Amaç
DETERMİNİSTİK	Doğrusal Programlama	Doğrusal bir amaç fonksiyonunu belirli kısıtlayıcı fonksiyonlar altında eniyilemeye çalışan yöntem
	Tamsayılı Programlama	Bütün değişkenleri tamsayı olan doğrusal bir amaç fonksiyonunu belirli kısıtlayıcı fonksiyonlar altında eniyilemeye çalışan yöntem
	Hedef Programlama	Birden fazla hedefi aynı anda ele alarak, hedeflerde meydana gelebilecek istenmeyen sapma düzeylerini en küçüklemeye çalışan yöntem
	Analitik Hiyerarşi Süreci	Birden fazla alternatifin yine birden fazla kritere göre değerlendirilerek en iyi alternatifin seçilmesini sağlayan yöntem
	Ulaştırma ve Atama Modelleri	Belli sayıda arz noktasından, belli sayıdaki talep noktasına ürün ulaştırma sonucu ortaya çıkan nakliye maliyetlerini minimize eden dağıtım planını belirlemeye yarayan yöntem
	Doğrusal Olmayan Programlama	Doğrusal olmayan bir amaç fonksiyonunu belli kısıtlayıcı fonksiyonlar(doğrusal veya doğrusal olmayan) altında eniyilemeye çalışan yöntem
	Deterministik Dinamik Programlama	Birbirleriyle ilişkili bir dizi karar alınmasını gerektiren problemlerde problemlerin küçük alt problemlere bölünmesi sonra her bir problem için çözüm aranması ve tüm çözümlerin birleştirilerek bütün problemin en iyi çözümünün bulunmasını amaçlayan yöntem
	Deterministik Stok Modelleri	Talebin bilindiği piyasa koşullarında yıllık toplam stok maliyetlerini minimize edecek üretim veya sipariş miktarını bulmaya çalışan yöntem
	Şebeke (Ağ) Analizi	Eniyileme problemlerinin çözümünde grafik veya ağların kullanıldığı yöntemler (En kısa yol problemleri, En yüksek akış problemleri, En küçük yayılmalı ağaç problemleri, PERT/CPM)
OLASILIKLI	Markov Zincirleri	Değişkenlerin sergiledikleri davranışları çözümleyerek, aynı değişkenlerin gelecekteki davranışlarını tahmin etmeye çalışan yöntem
	Kuyruk Teorisi (Bekleme hattı modelleri)	Her türlü fiziksel akışın olduğu mekânlarda hem hizmet verme maliyetini hem de hizmetlerinin anında karşılanmasını bekleyen müşterileri dikkate alarak servis olanaklarının saptanmasını sağlayan ve en iyi giriş-çıkış sürelerini belirleyen yöntem
	Karar Analizi	Belirlilik, belirsizlik veya risk durumunda karar vericiye karar alma ve karar sürecini geliştirme konularında yardımcı olan yaklaşım
	Oyun Teorisi	Birbirine rakip olan ve çıkarları çatışan tarafların (firma, ülke veya oyuncular vs) davranışlarının nasıl olması gerektiğini inceleyen ve çözüm arayan yöntem
	Simülasyon	Gerçek durumu bir modelle ifade edip bu modeli kurarak gerçek durum hakkında sonuçlar çıkaran yöntem
	Tahmin Modelleri	Geçmiş zamansal verilerden hareketle geleceğe ilişkin öngörüde bulunmaya çalışan yöntemler
	Olasılıklı Stok Modelleri	Talebin önceden bilinmediği veya olasılıklara göre belirlendiği piyasa koşullarında yıllık toplam stok maliyetlerini minimize edecek üretim veya sipariş miktarını bulmayı amaçlayan yöntem
	Olasılıklı Dinamik Programlama	Birbirleriyle ilişkili bir dizi karar alınmasını gerektiren problemlerde problemlerin küçük alt problemlere bölünmesi sonra her bir problem için çözüm aranması ve tüm çözümlerin birleştirilerek bütün problemin en iyi çözümünün bulunmasını amaçlayan yöntem (parametreler bilinmiyor)

Kaynak: Taha, 2007; Esin ve Şahin, 2012; Öztürk, 2014'ten aktaran: Demirer ve Alkan 2015

Tablo1.2:Yöneylem Teknikleri ve Tanımlamalar

Hazırladığımız bu çalışmada ***Yöneylem Yöntemlerinden Şebeke(Ağ) Analizi Yöntemini*** inceleyeceğiz.

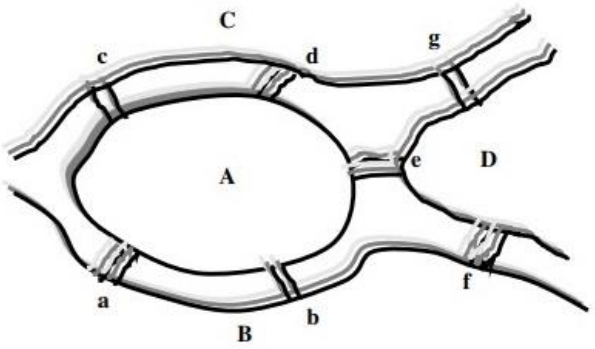
BÖLÜM 2. ŞEBEKE(AĞ) ANALİZİ YÖNTEMİ

2.1- Şebeke (Ağ) Analizi Yöntemi

Şebeke yapısı karmaşık sistemlerin temsili için kullanılan etkili bir modelleme tekniğidir. Şebeke analizinde kullanılan metodoloji, temelleri 1700'lü yıllarda İsveçli matematikçi ***Euler*** tarafından ortaya atılan Çizge Kuramından hareketle oluşturulmuştur.

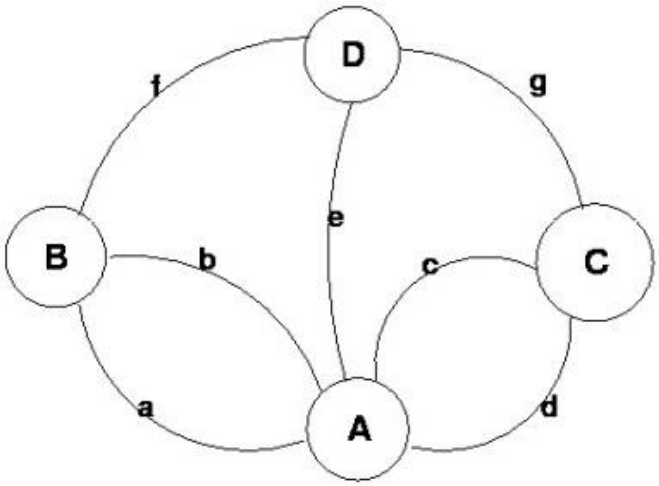
Şekil 1.1:Königsburg (Biggs, 1976)

Şekil 1.1'de betimlenen Königsburg şehri içinden geçen Pregel Nehri ve üzerindeki 7 köprü ile ***Çizge kuramının*** ilk uygulama alanıdır. Şehirde herhangi bir noktadan başlayıp her köprüden bir ve yalnızca bir kere geçerek başlanılan noktaya geri dönmeyi esas alan bu problem şehir halkı için yöresel bir spor halini almıştı. Uzun yıllar boyunca üzerinde çalışılmış olsa da kimse sonucu bulamamıştı. Ta ki 1736 yılında ***Leonhard Euler Könisburg Köprüsü Problemi*** isimli makalesinde bu problemin çözümünün bulunmadığını kanıtlayana kadar. Bu makale şebeke akışı problemini ilk olarak ortaya koyan yayındır. Şehrin ve şehri bağlayan köprülerin *Şekil 1.2*'de görülen açık yapısı bu makalede çizge yapısı ile açıklanmıştır.



Şekil 1.2:Königsburg Şehri ve Köprüler

Bu yapıdaki her bir nokta(A,B,C,D) bir düğüm ve her bir köprü (a,b,c,d,e,f,g) ise bir bağ olarak tanımlanmaktadır. ***Euler*** bu problemin çözümünün ancak her bölgeye çift sayıda köprü bağlantısıyla mümkün olabileceğini kanıtlamıştır.



Şekil 1.3:Königsburg Köprülerinin Şebeke Gösterimi

Euler'in çalışmaları , 1.Dünya Savaşı sonrasında ***Çizgi Kuramının*** isimli yeni bir matematik dalının oluşumuna temel oluşturmuştur. Optimizasyon problemlerinin Çizgi Kuramının oluşturduğu tekniklerden yola çıkılarak gösterilme ve çözülme çalışmaları ise Şebeke Analizini yarattı.

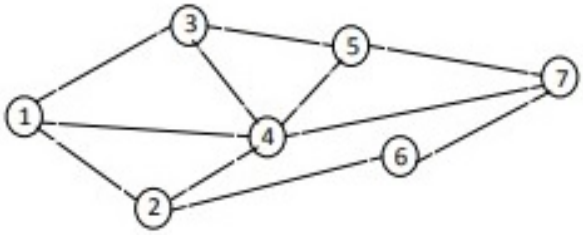
2.2- Şebeke (Ağ) Analizi Yöntemi Nedir?

Şebeke Analizi, bir planlama yöntemi olup, genellikle büyük ölçekli projelerin planlanması, bir noktadan diğerine en kısa yolun bulunması, inşaat planlaması, yeni ürünlerin pazarlamasının programlanması, belirli sistemlerdeki maksimum akışın (trafik akışı, sıvı akışı) bulunması gibi pek çok alanda kullanılabilir.

2.3- Şebeke (Ağ) Analizi Yöntemi Genel Tanımlar

- **Şebeke:** Program amacına ulaşabilmek için gereken faaliyetler ve olaylardan meydana gelen, faaliyet ve olayların birbirleri ile olan planlama gereği bağlantı ve ilişkilerini gösteren şemaya denir.
- **Faaliyet:** Bir işin tamamlanması için zaman ve kaynak harcaması gerektiren hareketi ifade eder. Verilen herhangi bir faaliyetin başlayabilmesi için önceki faaliyetlerin tamamlanmış olması gerekir.

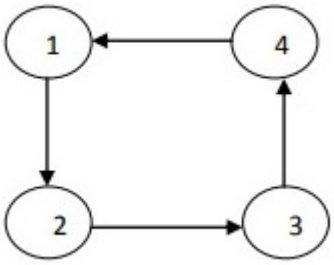
- **Olay:** Bir olay zamanda bir anı ifade eder ki bu anda bir faaliyet bitmiş ve diğer faaliyetler başlamaya hazırdır. Şebeke analizinde tüm faaliyetler oklar ile ve tüm olaylar da daire ve noktalar ile gösterilir.
- **Graf:** İki veya daha fazla nokta veya noktalardan bazı çiftlerin bir veya birkaç çizgi ile birleştirildiğinde ortaya çıkan seriye denir. Nesne çiftlerinin bir anlamda "ilişkili" olduğu bir dizi nesne kümesini belirleyen bir yapıdır.
- **Düğüm:** Grafik üzerindeki birleşim noktalarına denir.
- **Ayrıt:** İki düğümü birleştiren çizgiye denir.
- **Akım / Akış:** Ayrıtlar üzerinde tanımlanan bilgilere (fiyat, uzunluk) denir.



Şekil 1.4 Graf Örneği

Şekil 1.4' de gösterilen şemada düğümler ve ayrıtlar belirtilmiştir. Burada, düğümler olayları, ayrıtlar faaliyetleri tanımlar.

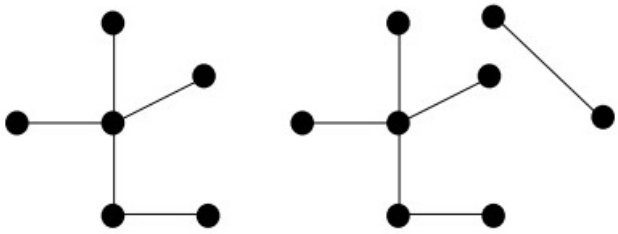
- **Zincir:** (i, j) gibi iki düğümü birleştiren dallar dizisine i ve j düğümleri arasındaki zincir denir.
- **Yol:** Üzerinde gidiş yönü belirlenen zincire yol denir.
- **Döngü:** Bir düğümü kendisine bağlayan zincire döngü denir.



Şekil 1.5 Graf Örneği 2

Şekil 2.5'de , (1-2) (2-3) (4-3) zincirdir. (1-2) (2-3) (3-4) yoldur. Bir zincirin yol olabilmesi için bağlantı noktalarının birbirini takip etmesi gerekir. Yol, aynı zamanda bir zincirdir. (1-2) (2-3) (3-4) (4-1) döngüdür.

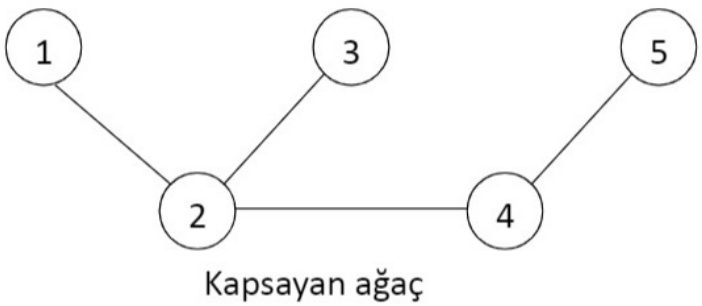
- **Bağlantılı Grafik:** Bir grafikte her düğüm noktasını birleştiren bir zincir varsa bu grafiğe bağlantılı grafik denir.
- **Ağaç:** Döngüsü olmayan bağlantılı grafiğe ağaç denir. Birden fazla ağacın olmasıyla orman oluşur.



Şekil 1.6:Ağaç ve Orman

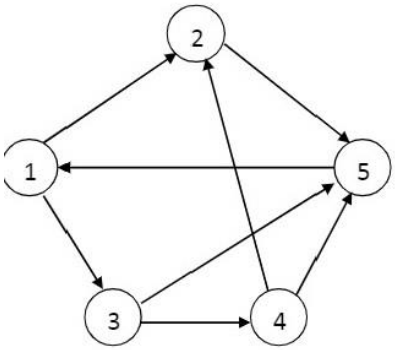
NOT: Bir şebeke bağlantılar (dallar) ve birbirine bağlanmış bir dizi düğümden oluşur. Şebeke, (N, A) notasyonu ile ifade edilir. Burada, N, düğümler kümesi ve A, bağlantılar kümesidir.

- **Kapsayan ağaç:** şebekenin tüm düğümlerini hiçbir döngüye izin vermeden birbirine bağlar. Burada kapsayan ağaç şebekenin tüm düğümlerini bağlayan ağaçtır.



Şekil 1.7 Kapsayan Ağaç

Örnek 1: Aşağıdaki şebeke için **(a)** bir yol, **(b)** bir döngü, **(c)** bir yönlendirilmiş döngü veya devre, **(d)** bir ağaç ve **(e)** bir kapsayan ağaç belirleyin.

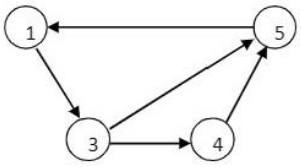


(a) Yol: 1-3-4-2

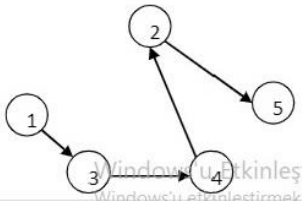
(b) Döngü: 1-5-4-3-1

(c) Yönlendirilmiş Döngü: 1-3-4-5-1

(d) Ağaç:



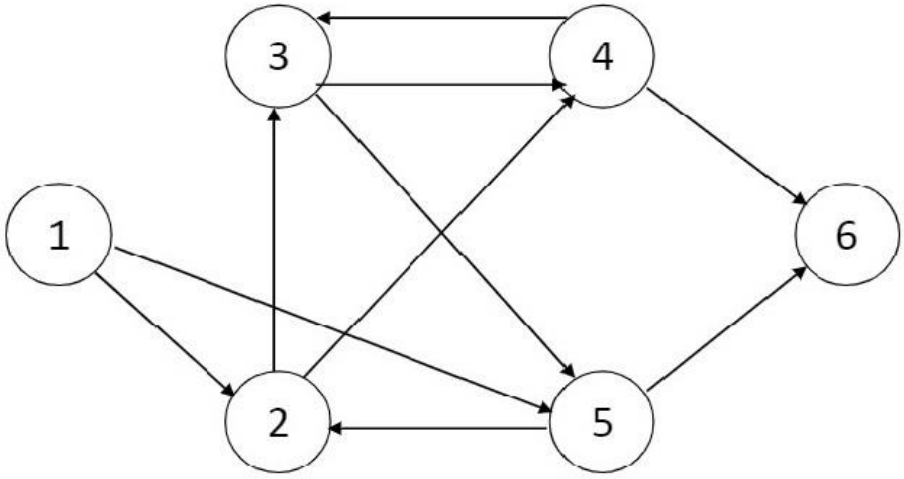
(e) Kapsayan Ağaç:



Örnek 2:

N = {1, 2, 3, 4, 5, 6}

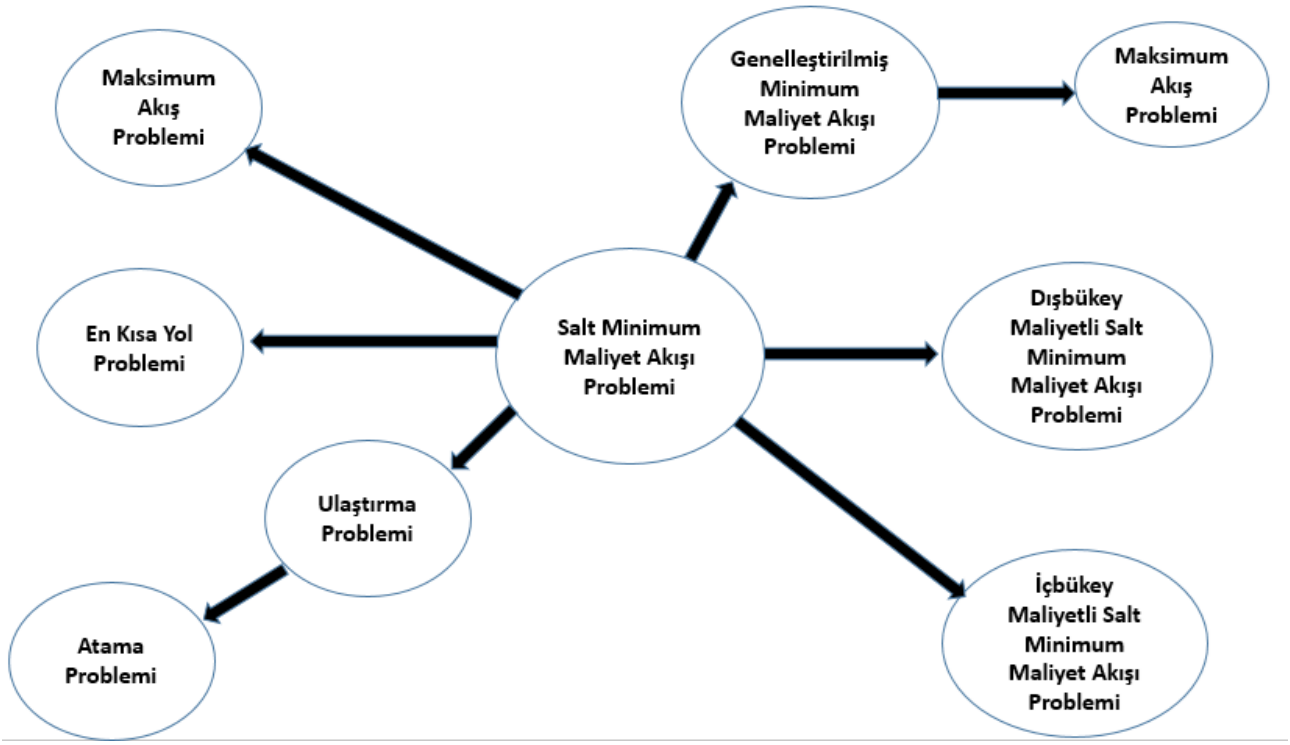
A = {(1,2), (1,5), (2,3), (2,4), (3,4), (3,5), (4,3), (4,6), (5,2), (5,6)}



BÖLÜM 3. ŞEBEKE(AĞ) MODELLERİ

3.1- Optimizasyon Modelleri Nedir?

Modeller, temel bilimlerde ve mühendislikte yoğun olarak kullanılan, büyük kapsamlı bir sistemin tüm özelliklerini yansıtacak daha küçük boyutlardaki yapılardır. Modeller, genelde sistemin temel özelliklerini yansıtacak ve modelin kullanım amaçlarını gerçekci olarak içerecek detaylar bulunur. Optimizasyon modelleri ise sistemin işleyişini ve özelliklerini yansıtan, sistemin içindeki ve çevresindeki diğer sistemlerle olan etkileşimleri kapsayan matematiksel ifadelerden oluşmaktadır.



Şekil 2.1:Akış Modelleri ve İlişkileri

3.2- Şebeke(Ağ) Uygulamalarının Kapsamı

Yöneylem araştırmasında, şebeke(dallarla birbirlerine bağlanan düğümler) olarak uygun bir biçimde modellenip çözülebilen çok sayıda durum vardır. Bunları anlayabilmek için aşağıdaki durumları inceleyelim.

- Meksika körfezindeki kuyuları iç kesimlerdeki teslim noktalarına bağlayan kıyıdaki doğalgaz boru hattı projesinin tasarımı. Modelin amacı boru hattının inşaat maliyetlerinin minimum kılınmasıdır.
- Varolan yol şebekesinde iki şehir arasındaki en kısa rotanın belirlenmesi.

- Kömür madenini enerji santraline bağlayan, kömürü su içerisinde taşıyarak nakladen bor hattı ağının maksimum kapasitesinin belirlenmesi.
- Petrol yataklarından rafinerilere boru hattıyla bağlanmış şebekenin minimum maliyet akış çizelgesinin belirlenmesi
- İnşaat projesinin faaliyetleri için zaman çizelgesinin (başlangıç ve bitiş tarihlerinin) belirlenmesi.

Bu durumların çözümü çeşitli ***Şebeke(Ağ) optimizasyonu*** algoritmalarıyla gerçekleştirilir. Bu çalışmada aşağıdaki beş algoritma tanıtılacaktır.

1. **Minimum Kapsayan Ağaç Algoritması*(1.durum)***
2. **En Kısa Yol Algoritması*(2.durum)***
3. **Maksimum Akış Algoritması*(3.durum)***
4. **Minimum Maliyet Kapasiteli Şebeke(Ağ) Algoritması*(4.durum)***
5. **Kritik Yol(CPM) Algoritması*(5.durum)***

3.2- Şebeke(Ağ) Uygulamaları Temel Prensipler

Şebeke yapısının kullanımında en verimli sonucun alabilmesi için dikkat edilmesi gereken hususlar aşağıda verilmiştir :

- Problem verilerinden bağımsız bir model yapısı kurulmalıdır. Girdi verilerinden biri değiştirildiğinde temel model yapısı değişmemelidir.
- Veriler birleştirilmemelidir. Mesela üretim ve ulaştırma maliyetleri, miktarları modelde ayrı ayrı gösterilmelidir.
- Mevcut verilere göre optimal olmadığı tahmin edilen alternatifler dışarıda bırakılmamalıdır. Girdiler değişirse daha önce optimal olmayan alternatifler optimal hale gelebilirler.
- Problemi özel bir model türü için belirlenen yapıya uydurmaya çalışmaktansa genel bir model yapısı kullanılması esneklik kazandıracaktır.

BÖLÜM 4. MİNİMUM KAPSAYAN(Yayılan) AĞAÇ ALGORİTMASI

4.1-Minimum Kapsayan(Yayılan) Ağaç Problemi (Minimum Spanning Tree Problem-MST) Nedir?

Minimum kapsayan(yayılan) ağaç problemi (Minimum Spanning Tree Problem-MST) en iyi bilinen şebeke optimizasyon problemlerinden biridir. Bu problemlere, birkaç coğrafik bölgeye dağılmış şehirlerarasında kurulacak iletişim şebekesinin topoloji tasarımı veya doğalgaz boru hattı, su şebekesi örnekleri verilebilir. Literatürde farklı algoritmaları bulunmaktadır. Bu algoritmalardan en çok kullanılanları ***Boruvka, Prim, Kruskal*** ve ***Ters Çevir-Sil*** algoritmalarıdır. Bu algoritmalar ***"Açgözlü (Greedy) Algoritmalar"*** olarak adlandırılmaktadırlar. Yani en uygun sonuca ulaşmak için her adımda uygulanacak belirli bir stratejiyi izlerler. Açgözlü denmesinin sebebi algoritmanın her adımında, sonraki adımları düşünmeden sadece bulunduğu durumdaki en iyi seçeneğe yönelmesindendir. ***Minimum kapsayan(yayılan) ağaç algoritması,*** doğrudan veya dolaylı olarak dalların en kısa bağlantısını kullanarak şebekenin dallarının birbiriyle ilişkilendirilmesini ele alır. Bu çalışmada ***Prim algoritmasını*** inceleyeceğiz.

4.2-Prim Algoritması Nedir?

Bu algoritma 1930 yılında matematikçi bilim adamı Vojtěch Jarník tarafından ortaya atılmıştır (Jarník, 1930). Daha sonra bilgisayar bilimcileri olan Robert C. Prim tarafından (Prim, 1957) ve Edsger W. Dijkstra tarafından (Dijkstra, 1959) tekrar keşfedilmiştir. Basitçe, ilk başta sadece belirlenen bir düğümü içeren ***Minimum kapsayan(yayılan) ağaç algoritması,*** kendisine en yakın düğümü seçerek ve kendisine dâhil ederek her adımda bir hat ve bir düğüm daha genişlemektedir.

4.3-Prim Algoritması Aşamaları

- Giriş: V düğümleri ve E ayrıtları ile bir bağlı ve ağırlıklı G.
- İlklendirme: Vyeni = {x}, burada x, V den bir keyfi düğümdür(başlangıç noktası), Eyeni={}
- Vyeni = V oluncaya kadar tekrarla:
- v nin Vyeni içinde w nin ise içinde olmadığı enküçük ağırlıklı bir (v,w) ayrıtını E den (eğer aynı ağırlıklı birden fazla ayrıt varsa çevrim oluşturmayacak şekilde keyfi olarak) seç.
- v yi Vyeni ye, (u, v) yi Eyeni ye ekle
- Çıkış: Vyeni ve Eyeni bir en küçük kapsama ağacını temsil eder.

4.4-Prim Algoritması Kodlarımız ve Açıklamaları

```
static int numara = 0;
static List<int> X_eksen = new List<int>();
static List<int> Y_eksen = new List<int>();
static int[,] mesafeler;

Graphics cizgi;
Pen kalem;
Point BaslangicNokta;
Point BitisNokta;
2 başvuru
void cizimYap(int n1x, int n1y, int n2x, int n2y)
{
    BaslangicNokta = new Point(n1x, n1y);
    BitisNokta = new Point(n2x, n2y);
    cizgi = panel1.CreateGraphics();
    cizgi.DrawLine(kalem, BaslangicNokta, BitisNokta);
    kalem.Dispose();
    cizgi.Dispose();
```

Şekil 3.1 Çizim Kodu

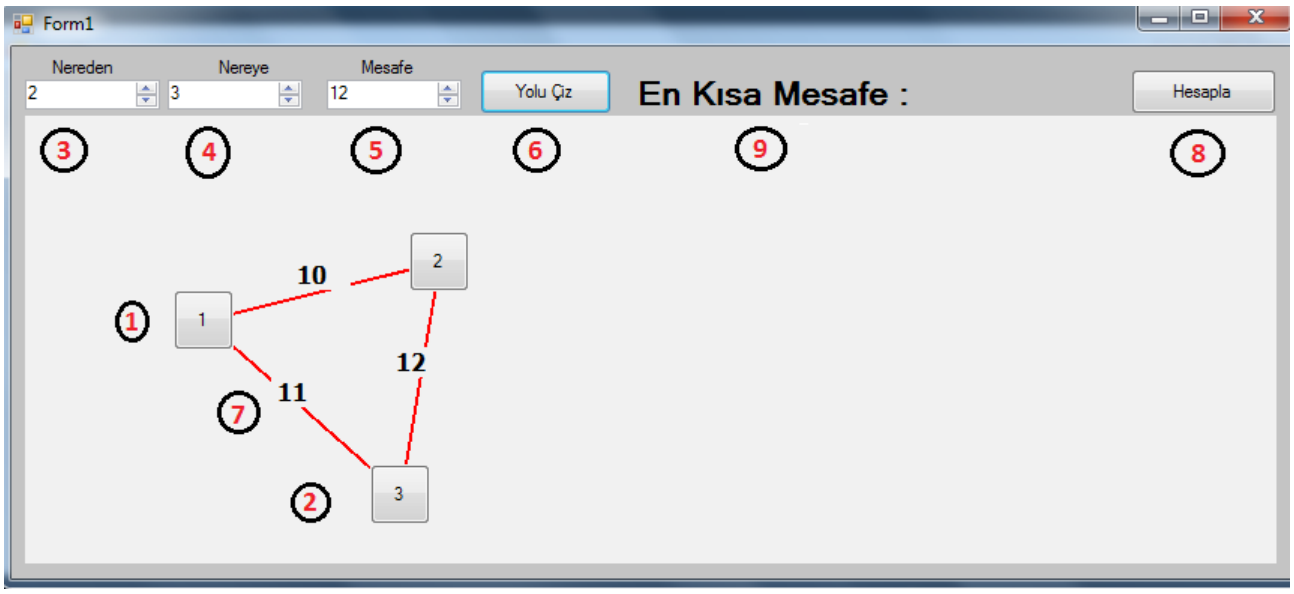
- **Graphics** fonksiyonu çizim yapmamıza yaramaktadır.
- **Pen** ile kalem fonksiyonu kullanılmaktadır.
- **Point** ile başlangıç ve bitiş noktalarımızı belirlenmektedir.

X eksenine ve Y eksenlerini birer liste haline getirdikten sonra çizimler yapılmaktadır.
Çizim yap fonksiyonu dört adet parametre almaktadır.

1. n1x = BaşlangıçNoktasının X ekseni
2. n1y = BaşlangıçNoktasının Y ekseni
3. n2x = BitisNoktasının X ekseni
4. n2y = BitisNoktasının Y eksenleridir.

- **cizgi** = panel1.CreateGraphics fonksiyonu panel1 üzerinde yeni bir çizgi çizileceğini belirler.
- **Drawline** fonksiyonu ile kalem çağırılır ve bu kalem ile belirlenen başlangıç noktasının x ve y koordinatlarından başlanır ve bitiş noktasının x ve y koordinatlarının aralarına bir çizgi çekilmeye yaramaktadır.

- **kalem.Dispose()** = Dispose komutu ile kalemimizi devre dışı bırakıyoruz.
- **cizgi.Dispose()** = Dispose komutu ile cizgi değişkeninin içerisinde bulunan CreateGraphics özelliğini devre dışı bırakıyoruz.



Şekil 3.2 En Kısa Mesafe

Şekil 3.2'deki numaraların işlevleri aşağıda değinilmiştir:

1. Birinci butonun konulduğu yerdir.
2. Üçüncü butonun konulduğu yerdir.
3. Çizginin çizilmeye nereden başlayacağının belirlenmesi için doldurulması gerekli olan alandır.
4. Çizgi bitişinin nerede olacağının belirlenmesi için doldurulması gerekli olan alandır.
5. Çizgi başlangıç ve bitişinin arasındaki mesafenin ne kadar olduğunun yazıldığı alandır.
6. Çizginin başlangıcı, bitişinin ve mesafesi yazıldıktan sonra yolunu çizmek için tıklanılan alandır.
7. Çizgi başlangıç ve bitişinin arasındaki mesafesinin yazıldığı yerdir.
8. Çizimler bitirildikten sonra ***Hesaplama*** butona tıklanır ve hesaplama yapılır.

9. Hesapla butonuna tıklandıktan sonra ***En Kısa Mesafe*** ekrana yazdırıldığı yerdir.

Şekil 3.2'deki çizimde ve altındaki açıklamalarda bulunan 1. Ve 3. Kısımın kodu *Şekil 3.3'de* görülmektedir.

```
private void panel1_MouseClick_1(object sender, MouseEventArgs e)
{
    numara++;
    X_eksen.Add(e.X);
    Y_eksen.Add(e.Y);
    Button btn = new Button();
    btn.Name = numara.ToString();
    btn.Text = numara.ToString();
    btn.Size = new Size(40, 40);
    btn.Location = new Point(e.X, e.Y);
    panel1.Controls.Add(btn);
    mesafeler = new int[numara, numara];
}
```

Şekil 3.3 *Ekrana Mouse İle Graf Ekleme*

- **numara++** =Numara değişkenin arttırılma sebebi kaç adet butonun konulduğunu bulabilmemiz, butonun ismi ve buton yazısı içindir.
- **X_eksen.Add** = X eksenindeki koordinatını *Şekil 3.1'de* bulunan listenin içerisine atmış bulunmaktayız.
- **Y_eksen.Add** = Y eksenindeki koordinatını *Şekil 3.1'de* bulunan listenin içerisine atmış bulunmaktayız.

Koordinata yeni bir buton oluşturuyoruz.

- **btn.Name** = Buton numarısı isimlendirmeyi sağlamaktadır.
- **btn.Text** = Butonun yazı yazılma kısmına butonun numarasını yazıyoruz.
- **btn.Size** = Butonun boyutunu belirlemek için yazılmıştır.
- **btn.Location** = X ekseni ve Y eksenindeki noktalarına butonu yerleştiriyoruz.
- **panel1.controls.Add(btn)** = Oluşturduğumuz butonu panelimize ekliyoruz.
- **mesafeler** = mesafeler Listemize butonun kaçıncı buton olduğunun bilgisini ekliyoruz.

```
private void panel1_MouseClick_1(object sender, MouseEventArgs e)
{
    numara++;
    X_eksen.Add(e.X);
    Y_eksen.Add(e.Y);
    Button btn = new Button();
    btn.Name = numara.ToString();
    btn.Text = numara.ToString();
    btn.Size = new Size(40, 40);
    btn.Location = new Point(e.X, e.Y);
    panel1.Controls.Add(btn);
    mesafeler = new int[numara, numara];
}
```

Şekil 3.4 Yeni Graf Ekleme

Şekil 3.2'de görünen *6. Adımdaki* kodu *Şekil 3.4 'de* görülmektedir.

- **Label lbl** = Oluşturmamızın sebebi butonların arasındaki mesafenin yazılabilmesi içindir.
- **baslangic** = *Şekil 3.2* 'de görülen 3. Adımdaki nereden başlayacağının belirlenmesi için txt_nereden textini baslangic değişkeninin içerisine atılmaktadır.
- **bitis** = *Şekil 3.2* 'de görülen 4. Adımdaki nerede biteceğinin belirlenmesi için txt_nereye textini bitis değişkeninin içerisine atılmaktadır.
- **kalem** = new Pen(Color.Red,2) çizginin renginin ve boyutunun ayarlanması için eklenmiştir.
- **cizimYap(X_eksen[baslangic - 1] + 20, Y_eksen[baslangic - 1] + 20, X_eksen[bitis - 1] + 20, Y_eksen[bitis - 1] + 20)** = cizimYap fonksiyonu çağırılarak X_ekseninin başlangıcının 1 azaltmamız gerekiyor. Çünkü diziler 0. İndis bulunmaktadır bu yüzden her defasında 1 azaltma işlemi yapılmaktadır. +20 dememizin sebebi butonun orta kısmından başlamasını istememizdir. Yani kısaca özetlemek istersek X ekseninin başlangıcıyla Y eksenin başlangıcı, X ekseninin bitişi ve Y eksenin bitişinin arasına çizgi çizilmeye yaramaktadır.
- **mesafe** ise şekil 3.2 de bulunan 5. Adımda yazılan mesafeyi txt_mesafe.text ten alış mesafe değişkeninin içerisine attık.

Daha sonra mesafeler dizinin içerisinde [baslangic -1 , bitiş -1] , [bitiş -1, baslangic -1] e eşitledik çünkü yollarımız 3'ten 5'e giderken aynı zamanda 5'ten de 3'e gidebilmelidir. Yeni bir label açıp içerisine başlangıç ve bitişin tam ortasına girilecek mesafeyi yazdırıyoruz. Ve panele ekliyoruz. *Şekil 3.2'deki* ve 8. Adımdaki Hesapla butonunun kodu *Şekil 3.5'de* verilmiştir.

```
List<int> baglananlar = new List<int>();
List<int> hedef = new List<int>();
baglananlar.Add(0);
int toplamMesafe = 0;
for (int i = 1; i < numara; i++)
{
    hedef.Add(i);
}
int cizgiSay = 0;
int enKucukSatir = 0, enKucukSutun = 0;
while (cizgiSay != numara - 1)
{
    int enKucuk = 1000;
    for (int i = 0; i < baglananlar.Count; i++)
    {
        for (int j = 0; j < hedef.Count; j++)
        {
            if (mesafeler[baglananlar[i], hedef[j]] != 0)
            {
                if (mesafeler[baglananlar[i], hedef[j]] < enKucuk)
                {
                    enKucuk = mesafeler[baglananlar[i], hedef[j]];
                    enKucukSatir = baglananlar[i];
                    enKucukSutun = hedef[j];
                }
            }
        }
    }
    cizgiSay++;
    toplamMesafe += enKucuk;
    hedef.Remove(enKucukSutun);
    baglananlar.Add(enKucukSutun);
    kalem = new Pen(Color.Green, 2);
    cizimYap(X_eksen[enKucukSatir] + 20, Y_eksen[enKucukSatir] + 20,
        X_eksen[enKucukSutun] + 20, Y_eksen[enKucukSutun] + 20);
}
lblEnKisaMesafe.Text += toplamMesafe.ToString();
```

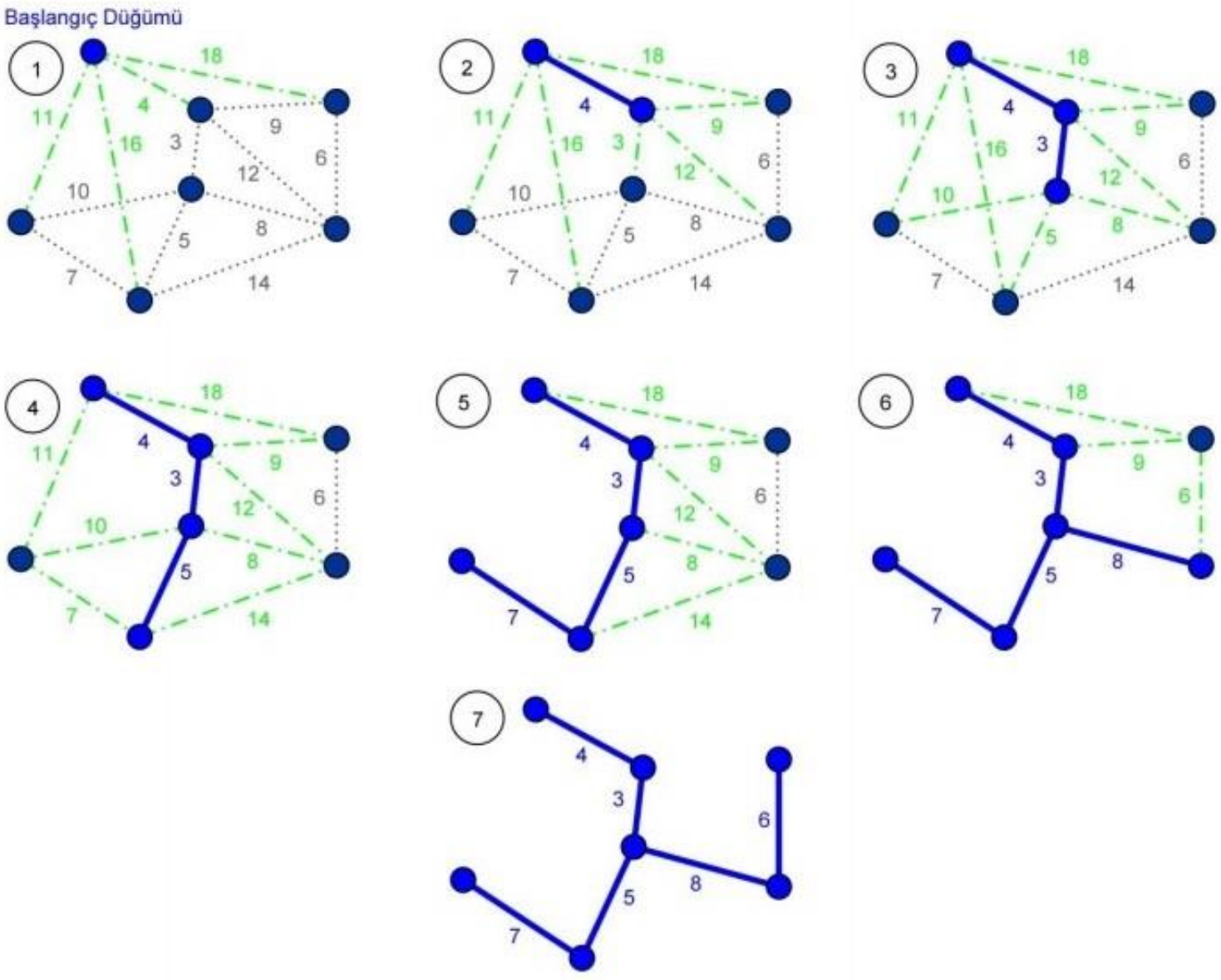
Şekil 3.5 *Prim Algoritmasının C# Kodu*

- Başlangıçta biri bağlananlar diğeri ise hedef olmak üzere iki adet liste oluşturuyoruz.
- Bağlananların listesine sıfırıncı ilk elemanımızı ekliyoruz. Toplam mesafeyi ilk başta 0 olarak atıyoruz.
- Kaç adet numara varsa bunu hedef listemize ekliyoruz. Çizgi sayı sıfıra eşitliyoruz. Ardından en küçük satır ve en küçük sütunu sıfıra eşitliyoruz. While koşul döngümüzün şartını belirliyoruz. Bu şart çizgi say eşit değilse numara -1 e koşul hep çalışacaktır. Başlangıçtaki en küçük düğümü bulabilmemiz için en küçük = 1000 olarak belirliyoruz.
- For döngüsünün bağlananlar sayısı kadar dönecek şekilde ayarlıyoruz. Başka bir for açıp hedef adeti kadar dönmesini sağlıyoruz. Daha sonra iki adet if bloğu açıyoruz. Birinci if bloğunda mesafeler listesinin içerisindeki bağlananlar i ile hedef j sıfıra eşit mi değil mi diye kontrol ediyoruz. Eşitse koşuldan çıkıp bir önceki döngüde j yi bir arttırıp kontrole devam edecek. J'nin For döngüsü bitince İ'nin for döngüsü bir arttırılarak devam edecek. Koşullar

bitince çizgi say bir arttırılacak. Hedef listesinin içerisinden en küçük sütun silinecek. Bağlananlar listesinin içerisine en küçük sütun aktarılacaktır. Kalem ile çizgi rengini yeşil yapıp boyutunu 2 olarak ayarlıyoruz. Çizim yap fonksiyonunu çağırıp çizgimizi çizdiriyoruz. En son while koşul döngüsünde çizgi say numara -1 e eşit olacak ve döngüden çıkacatır.

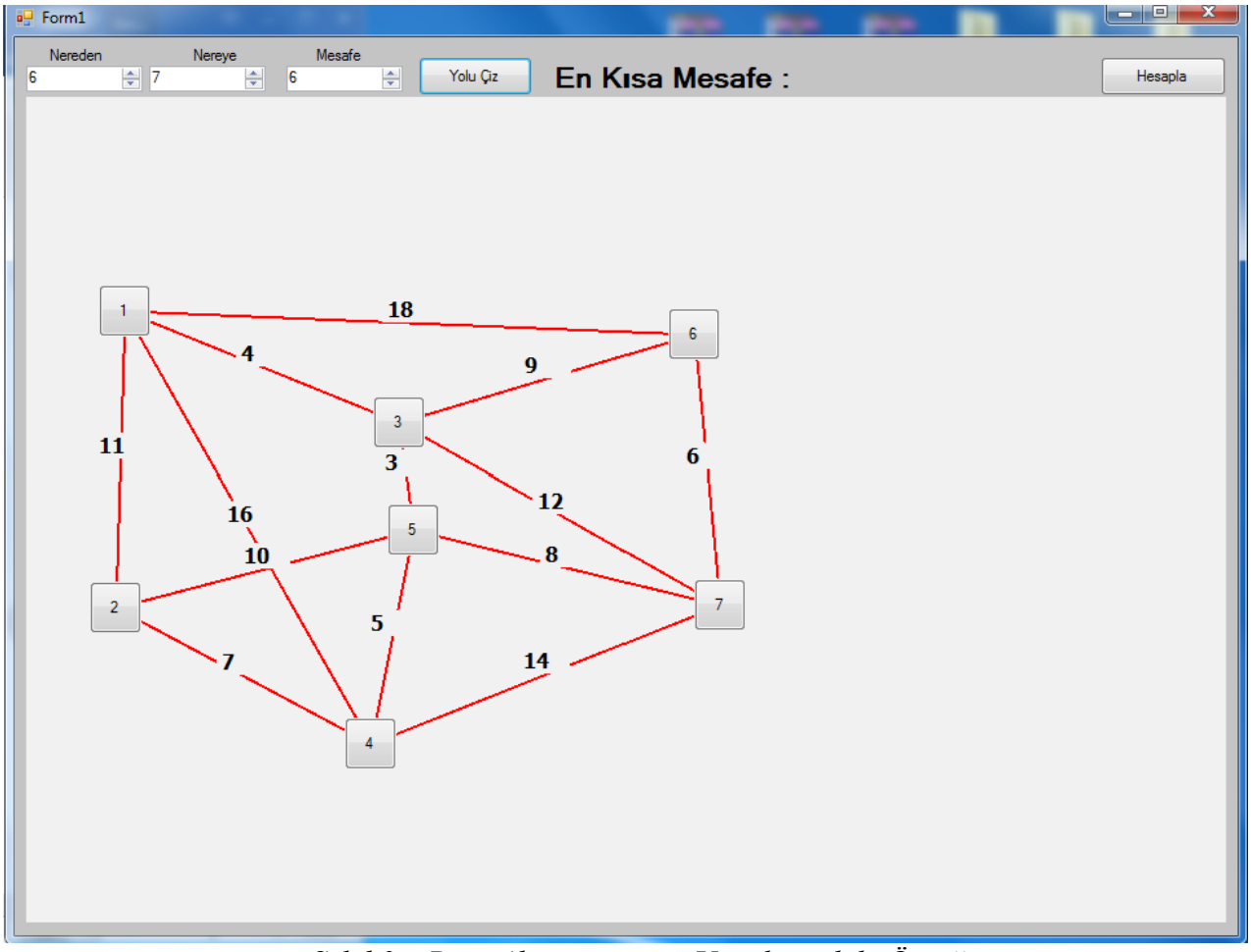
- ***LblEnKisaMesafe.Text***'ye ise toplam mesafeyi yazdırıyoruz.
- Eğer eşit değilse ikinci if e geçecek ve mesafeler listesine bağlananlar **i** ile hedef **j** en küçükten küçük ise en küçük olarak ayarlanacak. Bağlananlar **i '**yi en küçük satırın içerisine yazılacak. Hedef **j** ise en küçük sütunun içerisine atılacak. **J** bir arttırılıp kontroller devam edecek**.** Daha sonra **i**'nin içerisinde bulunan yani birinci for döngüsüne geçilip i bir arttırılarak devam edecek. Döngü bu şekilde devam edip en küçük değer bulunup toplam mesafenin içerisine atılacak. Çizgi say bir arttırılacak. Hedef listesinin içerisinden en küçük sütun silinecek. Bağlananlar listesinin içerisine en küçük sütun aktarılacaktır. Kalem ile çizgi rengini yeşil yapıp boyutunu 2 olarak ayarlıyoruz. Ve çizim yap fonksiyonunu çağırıp çizimimizi yaptırıyoruz. En son while koşul döngüsünde çizgi say numara -1 e eşit olacak ve döngüden çıkacaktır.
- ***LblEnKisaMesafe.Text***'ye ise toplam mesafeyi yazdırıyoruz.

4.5-Prim Algoritması Örneği ve Programdaki Çıktısı



Şekil 3.6:Prim algoritması ile bir grafın Minimum kapsayan(yayılan) ağaç yapısının çıkarılması örneği

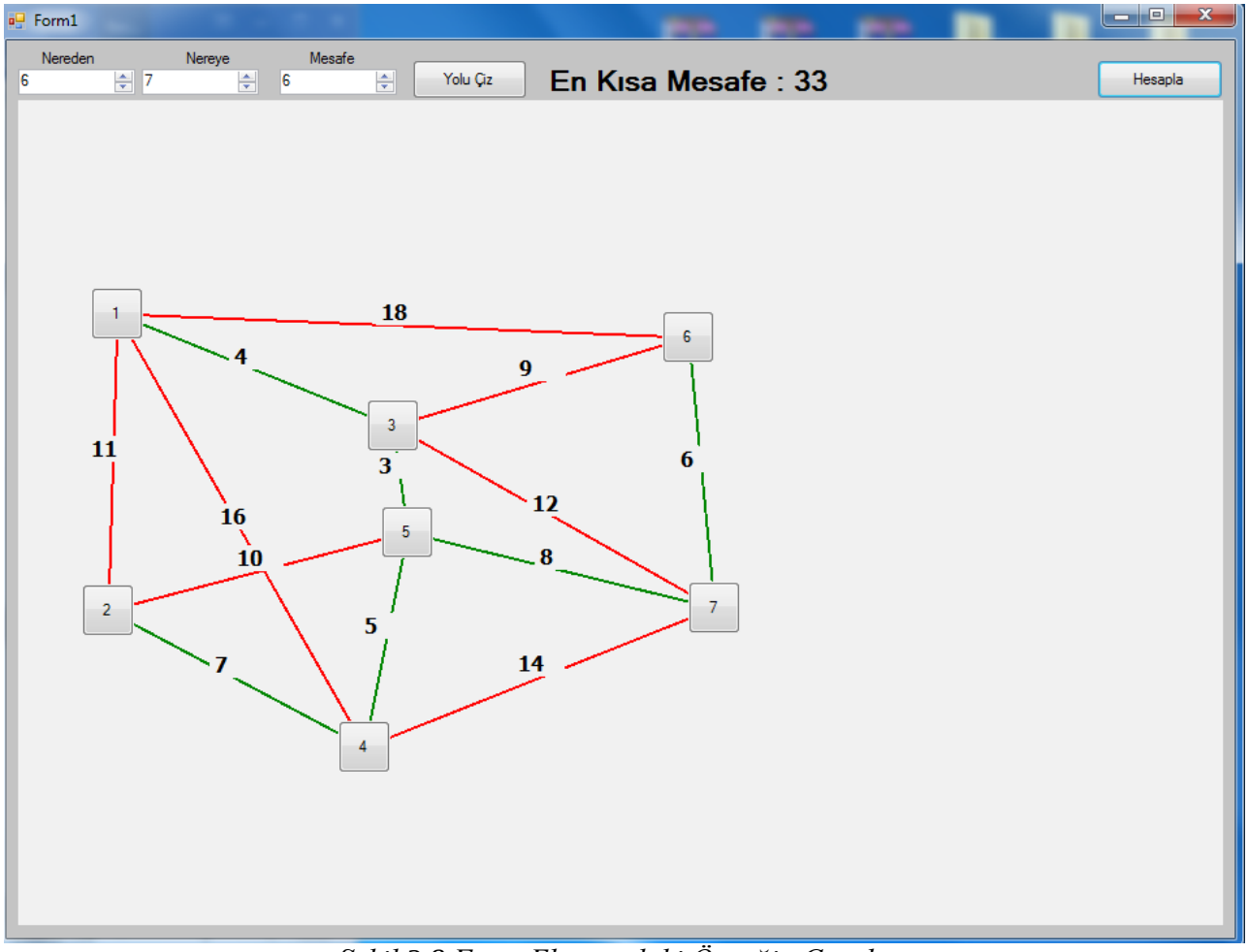
Şekil 3.6'deki örnekte bütün noktalara uğrayıp optimum çözümü 33 olarak bulmaktadır. Şimdi aynı örneği bizim projemizde deneyelim.



Şekil 3.7 Prim Algoritmasının Uygulamadaki Örneği

Nereden, Nereye ve ***Mesafe*** değerlerini girdikten sonra ***Hesapla*** butonuna tıklıyoruz.

Şekil 3.8'deki sonucu elde ediyoruz.



Şekil 3.8 Form Ekranındaki Örneğin Cevabı

Verilen örnek ile bizim yazdığımız form uygulamasında da görüldüğü gibi adımlar ve sonuçlar aynı çıkmıştır.

BÖLÜM 5. EN KISA YOL ALGORİTMASI

5.1-En Kısa Yol Algoritması (Shortest Path Algorithm) Nedir?

En Kısa Yol Algoritması basit bir mantıkla oluşturulmuş ve günümüzde oldukça kullanılan ünlü algoritmalardan bir tanesidir. ***En Kısa Yol Algoritması'nın*** çıkış noktası farklı düğümlerden en kısa yoldan hedefe ulaşmayı amaçlamaktadır. Bu amaçla günümüzde de internet trafiğinin yönlendirilmesinde, oyun programlamada sıkça kullanılmaktadır ve en çok karşılaştığımız navigasyon ve gps sistemlerinde kullanılmaktadır. ***En Kısa Yol Algoritması iki hedef düğüm arasında*** en az uğraş ile gidilebilecek yolun belirlenmesidir. Literatürde farklı algoritmaları bulunmaktadır. Bu algoritmalardan en çok kullanılanları ***Dijkstra*** ve ***Floyd-Warshall*** algoritmalarıdır. Bu çalışmada ***Dijkstra algoritmasını*** inceleyeceğiz.

5.2- Dijkstra Algoritması Nedir?

Dijkstra'nın Algoritması, bir graf üzerinde istenilen bir düğümden, graftaki diğer tüm düğümlere giden ***En Küçük Maliyetli(E.K.M)*** yolları (shortest path) bulmak için kullanılan bir algoritmadır. Algoritma şu şekilde açıklanabilir:

- G=(V,E) yönlü grafında, V={1,2,….,n} ve (i,j) yaylarının negatif olmayan ağırlıkları a(i,j) olsun.Eğer i den j ye bir yay yok ise a(i,j) sonsuz olarak alınır. Böylece G için A= a(i,j) de diyagonalı sıfır olan nxn boyutlu bir ağırlık matrisi vardır. Problem, düğüm 1 den diğer bütün düğümlere en kısa yol ve en kısa mesafeyi bulmaktır.
- Her bir i düğümüne kalıcı veya geçici bir etiket verilir. Kalı cı etiket L(i) 1. düğümde i. Düğüme olan en kısa mesafedir, oysa geçici etiket L'(i) 1. den i. ye olan en kısa mesafenin üst sınırıdır. Algoritmanı n her bir fazında, P kalıcı etiketli düğümlerin kümesi ve T onun eşleniğidir. Başlangıçta her bir i için, P={1}, L(1)=0 ve L'(i) = a(1,i) dir. P=V olduğu zaman algoritma durur.

5.3- Dijkstra Algoritması Aşamaları

- **Adım 1:** (Kalıcı etiketin belirlenmesi) : T de en küçük L'(k) olan bir k düğümünü belirle. Eğer böyle bir k yok ise dur, çünkü T deki düğüm 1 den herhangi bir düğüme yol yoktur. k yı P kümesine ekle. Eğer P=V ise dur.
- **Adım 2:** (Geçici etiketin gözden geçirilmesi) : Eğer j, T de bir düğüm ise, L'(j) yi daha küçük L'(j) ve L(k)+ a(k,j) ile değiştir. Adım 1 e git. min(L'(j), (L(k)+ a(k,j))Düğüm etiketleri geçici ve kalı cı olarak iş aretlenirler. Geçici etiket, aynı düğüme daha k ısa bir yol bulunursa başka bir etiketle değiştirilir. Daha iyi bir yol bulunamayacaksa etiket kalıcı olarak işaretlenir.

5.4-DijkstraAlgoritması Kodlarımız ve Açıklamaları

```
static int numara = 0;
static List<int> X_eksen = new List<int>();
static List<int> Y_eksen = new List<int>();
static int[,] mesafeler;

Graphics cizgi;
Pen kalem;
Point BaslangicNokta;
Point BitisNokta;
2 başvuru
void cizimYap(int n1x, int n1y, int n2x, int n2y)
{
    BaslangicNokta = new Point(n1x, n1y);
    BitisNokta = new Point(n2x, n2y);
    cizgi = panel1.CreateGraphics();
    cizgi.DrawLine(kalem, BaslangicNokta, BitisNokta);
    kalem.Dispose();
    cizgi.Dispose();
```

Şekil 4.1 Çizim Yapma Kodları

- **Graphics** fonksiyonu çizim yapmamıza yaramaktadır.
- **Pen** ile kalem fonksiyonu kullanılmaktadır.
- **Point** ile başlangıç ve bitiş noktalarımızı belirlenmektedir.

X eksenine ve Y eksenlerini birer liste haline getirdikten sonra çizimler yapılmaktadır.

- **Çizim yap** fonksiyonu dört adet parametre almaktadır.

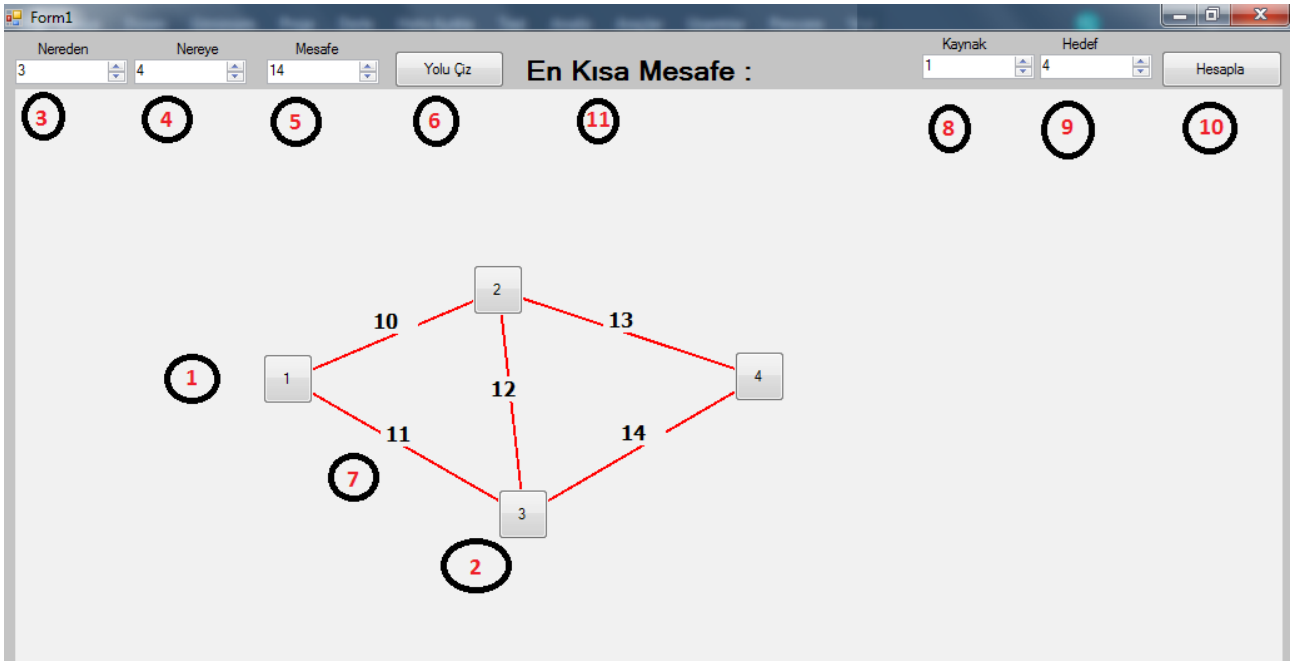
1. n1x = BaşlangıçNoktasının X ekseni
2. n1y = BaşlangıçNoktasının Y ekseni
3. n2x = BitisNoktasının X ekseni
4. n2y = BitisNoktasının Y eksenleridir.

- **cizgi** = panel1.CreateGraphics fonksiyonu panel1 üzerinde yeni bir çizgi çizileceğini belirler.

- **Drawline fonksiyonu** ile kalem çağırılır ve bu kalem ile belirlenen başlangıç noktasının x ve y koordinatlarından başlanır ve bitiş noktasının x ve y koordinatlarının aralarına bir çizgi çekilmeye yaramaktadır.

- **kalem.Dispose()** = Dispose komutu ile kalemimizi devre dışı bırakıyoruz.

- **cizgi.Dispose()** = Dispose komutu ile cizgi değişkeninin içerisinde bulunan CreateGraphics özelliğini devre dışı bırakıyoruz.



Şekil 4.2 Form Ekranı

Şekil 4.2'deki numaraların işlevleri aşağıda değinilmiştir.

1. Birinci butonun konulduğu yerdir.
2. Üçüncü butonun konulduğu yerdir.
3. Çizginin çizilmeye nereden başlayacağının belirlenmesi için doldurulması gerekli olan alandır.
4. Çizgi bitişinin nerede olacağının belirlenmesi için doldurulması gerekli olan alandır.

5. Çizgi başlangıç ve bitişinin arasındaki mesafenin ne kadar olduğunun yazıldığı alandır.
6. Çizginin başlangıcı, bitişinin ve mesafesi yazıldıktan sonra yolunu çizmek için tıklanılan alandır.
7. Çizgi başlangıç ve bitişinin arasındaki mesafesinin yazıldığı yerdir.
8. Algoritmanın başlangıcının belirlendiği yerdir.
9. Algoritmanın bitiş yeridir.
10. Çizimler bitirildikten sonra Hesaplama butona tıklanır ve hesaplama yapılır.
11. Hesapla butonuna tıklandıktan sonra En Kısa Mesafe ekrana yazdırıldığı yerdir.

```
private void panel1_MouseClick_1(object sender, MouseEventArgs e)
{
    numara++;
    X_eksen.Add(e.X);
    Y_eksen.Add(e.Y);
    Button btn = new Button();
    btn.Name = numara.ToString();
    btn.Text = numara.ToString();
    btn.Size = new Size(40, 40);
    btn.Location = new Point(e.X, e.Y);
    panel1.Controls.Add(btn);
    mesafeler = new int[numara, numara];
}
```

Şekil 4.3 *Ekrana Mouse İle Graf Ekleme*

- **numara++** =Numara değişkenin arttırılma sebebi kaç adet butonun konulduğunu bulabilmemiz, butonun ismi ve butonun yazısı içindir.
- **X_eksen.Add** = X eksenindeki koordinatını *Şekil 5.1'*de bulunan listenin içerisine atmış bulunmaktayız.
- **Y_eksen.Add** = Y eksenindeki koordinatını *Şekil 5.1'*de bulunan listenin içerisine atmış bulunmaktayız.

Koordinata yeni bir buton oluşturuyoruz.

- **btn.Name** = Butona numarasının ismini vermemize yaramaktadır.
- **btn.Text** = Butonun yazı yazılma kısmına butonun numarasını yazıyoruz.
- **btn.Size** = Butonun boyutunu belirlemek için yazılmıştır.
- **btn.Location** = X ekseni ve Y eksenindeki noktalarına butonu yerleştiriyoruz.
- **panel1.controls.Add(btn)** = Oluşturduğumuz butonu panelimize ekliyoruz.
- **mesafeler** = mesafeler Listemize butonun kaçıncı buton olduğunun bilgisini ekliyoruz.

```
Label lbl;
1 başvuru
private void button1_Click(object sender, EventArgs e)
{
    int baslangic = Convert.ToInt32(txt_nereden.Text);
    int bitis = Convert.ToInt32(txt_nereye.Text);
    kalem = new Pen(Color.Red, 2);
    cizimYap(X_eksen[baslangic - 1] + 20, Y_eksen[baslangic - 1] + 20, X_eksen[bitis - 1] + 20, Y_eksen[bitis - 1] + 20);
    int mesafe = Convert.ToInt32(txt_mesafe.Text);
    mesafeler[baslangic - 1, bitis - 1] = mesafe;
    mesafeler[bitis - 1, baslangic - 1] = mesafe;
    lbl = new Label();
    lbl.Name = "lbl" + baslangic.ToString() + bitis.ToString();
    lbl.Text = mesafe.ToString();
    lbl.Size = new Size(40, 20);
    lbl.Font = new Font("Tahoma", 12, FontStyle.Bold);
    lbl.Location = new Point((X_eksen[baslangic - 1] + X_eksen[bitis - 1]) / 2, (Y_eksen[baslangic - 1] + Y_eksen[bitis - 1]) / 2);
    panel1.Controls.Add(lbl);
}
```

Şekil 4.4 *Form Ekranında Çizgiyi Çizip Çizginin Üzerine Mesafe Bilgisi Yazan Kod*

- **Label lbl** = Oluşturmamızın sebebi butonların arasındaki mesafenin yazılabilmesi içindir.
- **baslangic** = *Şekil 5.4'de* görülen 3. Adımdaki nereden başlayacağının belirlenmesi için txt_nereden textini baslangic değişkeninin içerisine atılmaktadır.
- **bitis** =*Şekil 5.4'de* görülen 4. Adımdaki nerede biteceğinin belirlenmesi için txt_nereye textini bitis değişkeninin içerisine atılmaktadır.
- **kalem** = new Pen(Color.Red,2) çizginin renginin ve boyutunun ayarlanması için eklenmiştir.
- **cizimYap(X_eksen[baslangic - 1] + 20, Y_eksen[baslangic - 1] + 20, X_eksen[bitis - 1] + 20, Y_eksen[bitis - 1] + 20)** = cizimYap fonksiyonu çağırılarak X_ekseninin başlangıcının 1 azaltmamız gerekiyor. Çünkü diziler 0. İndis bulunmaktadır bu yüzden her defasında 1 azaltma işlemi yapılmaktadır. +20 dememizin sebebi butonun orta kısmından başlamasını istememizdir. Yani kısaca özetlemek istersek X ekseninin başlangıcıyla Y eksenin başlangıcı, X ekseninin bitişi ve Y eksenin bitişinin arasına çizgi çizilmeye yaramaktadır.

- **mesafe** ise *Şekil 4.4'de* bulunan 5. Adımda yazılan mesafeyi txt_mesafe.text ten alış mesafe değişkeninin içerisine attık.

Daha sonra mesafeler dizinin içerisinde [baslangic -1 , bitiş -1] , [bitiş -1, baslangic -1] e eşitledik çünkü yollarımız 3'ten 5'e giderken aynı zamanda 5'ten de 3'e gidebilmelidir.
Yeni bir label açıp içerisine başlangıç ve bitişin tam ortasına girilecek mesafeyi yazdırıyoruz ve panele ekliyoruz.

```
1 başvuru
void printSolution(int[] dist, int n)
{
    Console.WriteLine("Düğümlerin ana kaynağa uzaklığı");
    for (int i = 0; i < numara; i++)
    {
        Console.WriteLine(i + "    " + dist[i]);
    }
}
```

Şekil 4.5 Sonucun Ekrana Yazdırılmasının Kodu

PrintSolution fonksiyonu Dijkstra fonksiyonundan gelen dist ve n parametrelerini işlemektedir. Console.WriteLine ("Düğümlerin ana kaynağa uzaklığı") ekrana yazdırmak için kullanılmıştır. Daha sonrasında for döngümüzü açıyoruz ve numaradan küçük oluncaya kadar dist[i] yi toplayıp ekrana yazdırıyoruz.

```
1 başvuru
int minDistance(int[] dist, bool[] sptSet)
{
    int min = int.MaxValue;
    int min_index = -1;
    for (int i = 0; i < numara; i++)
    {
        if (sptSet[i] == false && dist[i] <= min)
        {
            min = dist[i];
            min_index = i;
        }
    }
    return min_index;
}
```

Şekil 4.6 Minimumum Bulunup Diziye Aktarılmasının Kodu

minDistance adında iki parametresi bulunan bir fonksiyon yazdık. İki parametrenin birisi dist adında int tipindeki listemizdir. İkincisi ise bool tipi olan sptSet adında bir dizimiz bulunmaktadır. Bu fonksiyonun içerisinde;

- Min = Alabileceği maksimum sayıyı veriyoruz ki minimum sayıyı kolay bir şekilde bulabilmek için.
- Min_index = dizilerin sıfırıncı elemanı olduğu için -1 olarak atamaktayız.

For döngümüz form ekranındaki eleman sayısı kadar dönmesi gerekmektedir.
For döngümüzün içerisine bir adet if kontrol koşulumuzu açıp ;

- sptSet[i] == false bir sonraki Disktra adında yazdığımız fonksiyonun içerisinde bulunan bool değeri döndürmektedir
- dist[i] <= min küçükse minimumdan dist[i]'ninci elamanı minimuma eşitliyoruz.

- Min_index = i ; minimum indeksimizi i ye eşitliyoruz.

For döngümüzü bitirip return min index ile programı geri döndürüyoruz.

```
1 başvuru
int dijkstra(int[,] graph, int src, int dest)
{
    int[] dist = new int[numara];
    bool[] sptSet = new bool[numara];
    for (int i = 0; i < numara; i++)
    {
        dist[i] = int.MaxValue;
        sptSet[i] = false;
    }
    dist[src] = 0;
    for (int i = 0; i < numara - 1; i++)
    {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int j = 0; j < numara; j++)
        {
            if (!sptSet[j] && graph[u, j] != int.MaxValue && dist[u] != int.MaxValue && dist[u] + graph[u, j] < dist[j])
            {
                dist[j] = dist[u] + graph[u, j];
            }
        }
    }
    printSolution(dist, numara);
    int sonuc = dist[dest];
    return sonuc;
}
```

Şekil 4.7 Dijkstra Fonksiyonun Kodu

Dijkstra fonksiyonumuz üç adet parametre almaktadır. Birincisi int[,] graph bu parametremiz dizinin iki elemanın çizimini yapmak için kullanılmaktadır. İkinci parametremiz int tipinde tanımlanmış src parametresidir. Başlangıcı noktamızı tutmaktadır. Üçüncüsü ise int tipinde tanımlanmış dest parametresidir. Bitiş noktamızı tutmaktadır.

- İnt[] dist = new int [numara]; = numara değişkenimizin yani eklenen elemanımızı dist dizisinin içerisine atılmaktadır.

- Bool[] sptSet = new bool [numara] = İlk etapta null dönmekte ve daha sonra en küçükten küçük ise true değeri döndürmektedir.

For döngümüz eleman sayısı kadar dönecek ve içerisinde

- Dist[i] = int.MaxValue; dizimizdeki ilk elamana maksimum değeri olarak belirliyoruz.
- Daha sonra sptSet[i] = false değeri atıyoruz.

For döngüsü bittikten sonra

- Dist[src] = 0 ; Yani kaynak noktamızı belirliyoruz.

Daha sonra for döngümüzü açık numara -1 adet kadar döndürüyoruz.

Bu for döngümüzün içerisinde;

- İnt u= minDistance (dist,sptSet) Bir öncekisi resimde yazdığımız fonksiyonu çağırıp sonucunu u nun içerisine atıyoruz. Daha sonra sptSet[u]= true yapıp işlemlerimizi devam ediyoruz.
- minDistance fonksiyonu elamanların arasındaki bağlantıları kontrol etmemizi sağlamaktadır.

Daha sonra bir adet daha for açıp içerisinde bulunan if koşul ifadesiyle kontrollerimizi yapıyoruz.

- !sptSet[j] && graph[u, j] != int.MaxValue && dist[u] != int.MaxValue && dist[u] + graph[u, j] < dist[j]

Bu kontrol şuandaki bulunduğumuz elemanın gidiş yollarının küçüklüğünü kontrol etmek için yapılmıştır.

- Eğer koşul doğru ise dist[j] listemizin içerisine dist[u] ile graph[u,j] toplanıp yazılacaktır.

For döngüsünün bitiminden sonra printSolution fonksiyonu çalışıp (dist, numara) elemanlarını parametresine gönderiyoruz.

- Sonuç değişkenimize dist[dest] e eşitliyoruz ve return sonuç deyip koşumuzu kaldığımız yerden devam ediyoruz.

```
1 başvuru
private void button2_Click(object sender, EventArgs e)
{
    lblEnKisaMesafe.Text = "En Kısa Mesafe : ";
    for (int i = 0; i < numara; i++)
    {
        for (int j = 0; j < numara; j++)
        {
            if (mesafeler[i, j] == 0 && i != j)
                mesafeler[i, j] = int.MaxValue;
        }
    }
    int sonuc = dijkstra(mesafeler, Convert.ToInt32(txt_kaynak.Text) - 1, Convert.ToInt32(txt_hedef.Text) - 1);
    lblEnKisaMesafe.Text += sonuc.ToString();
}
```

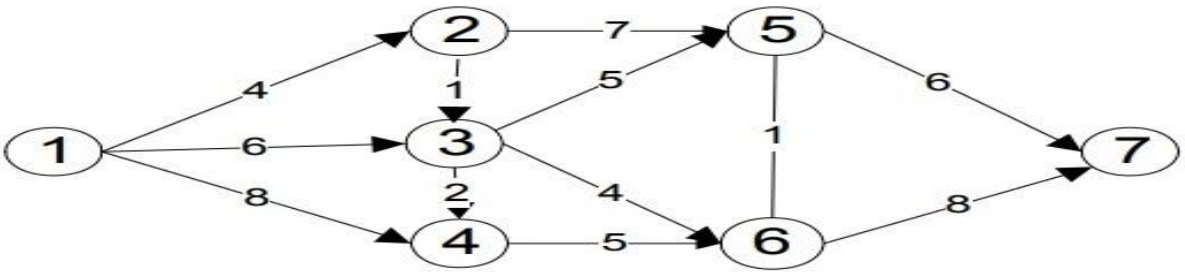
Şekil 4.8 Sonucu Hesaplayan Kod

Şekil 4.8'de Hesapla butonuna ait kodun bulunduğu kısımdır. Burada iki adet iç içe bulunan for döngüleri görülmektedir. For döngüleri numara yani eleman sayımız kadar dönecek olup ardından if kontrol bloğu ile kontrol edilecektir.

İf kontrol bloğu mesafeler [i,j] == 0 ve && i!=j ye mesafeler[i,j]= maksimum sayı değerini alacaktır. Yani bu bir nevi eleman mesafenin 0 girilmesi ve inin ve j nin birbirine eşit olmayacağı anlamına gelmektedir. Biz biliyoruz ki işlediğimiz konu gereğince C noktasındayken olduğumuz yerde tekrar C noktasına gelmemiz gereksiz. Ancak ilk önce bir yere gidip daha sonra tekrar C noktasına gelebilmemiz mümkündür.

Bunu önlemek için bu if ifadesi konulmuştur. Eğer koşullar uyuyorsa mesafe[i,j]'nin içerisine maksimum sayı değerimizi atayıp döngümüzün kontrollerine devam ediyoruz. Döngülerin ardından sonuç ifadesini dijkstra fonksiyonuna mesafeleri, kaynak değerimizi ve hedef değerimizi bir azaltıp içerisine gönderiyoruz. En son dijkstra fonksiyonu bittikten sonra sonuçları toplayıp ekrana yazdırıyoruz.

5.5-Dijkstra Algoritması Örneği ve Programdaki Çıktısı



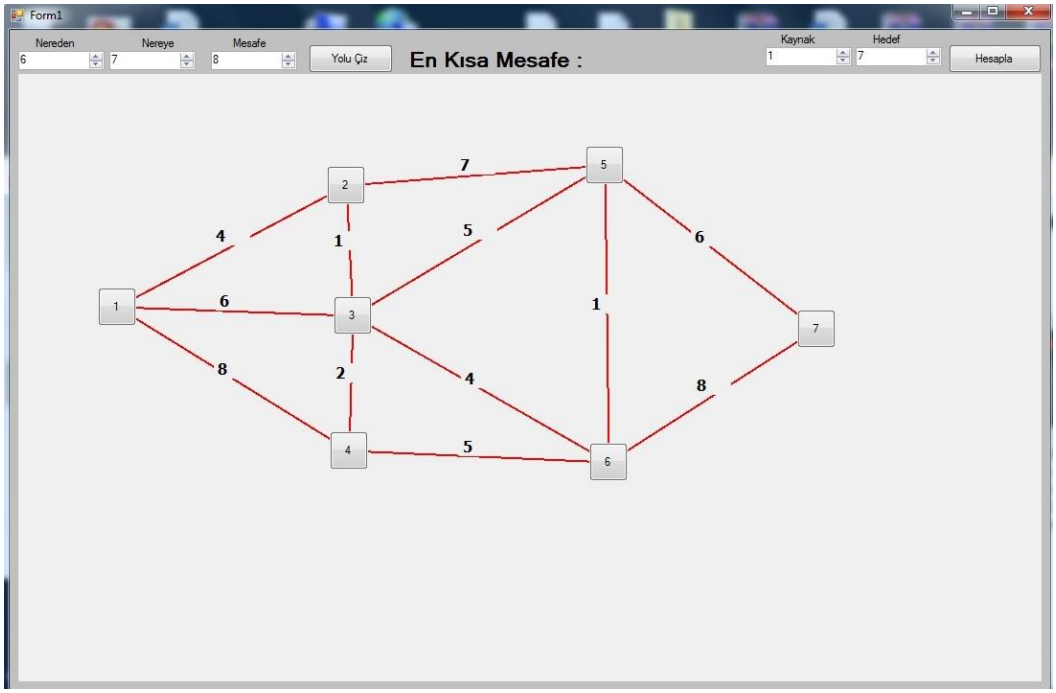
Şekil 4.9 Dijkstra Algoritması ile Bir Grafın En Kısa Yolun Örneği

Şekil 4.9'daki graf için ***Dijkstra Algoritması'nın*** adımları:

P={1} L(1)= 0 **İter 1**	T={2,3,4,5,6,7} L'(2)= 4 L'(3)= 6 L'(4)= 8 L'(5)= - L'(6)= - L'(7)= -	P={1,2} L(1)= 0 L(2)= 4	T= T-{2}={3,4,5,6,7} L'(3)= min{6, L'(2)+ a(2,3)} L'(4)= min{8, L'(2)+ - } L'(5)= min{- , L'(2)+ a(2,5)} L'(6)= min{-, L'(2)+ -} L'(7)= min{-, L'(2)+ -} *gecici*
P={1,2} L(1)= 0 L(2)= 4 **İter 2**	T={3,4,5,6,7} L'(3)= 5 L'(4)= 8 L'(5)= 11 L'(6)= - L'(7)= -	P={1,2,3} L(1)= 0 L(2)= 4 L(3)=5	T= T-{3}={4,5,6,7} L'(4)= min{8, L'(3)+ a(3,4)} L'(5)= min{11, L'(3)+ a(3,5)} L'(6)= min{-, L'(3)+ a(3,6)} L'(7)= min{-, L'(3)+ -}
P={1,2,3} L(1)= 0 L(2)= 4 L(3)=5 **İter 3**	T={4,5,6,7} L'(4)= 7 L'(5)= 10 L'(6)= 9 L'(7)= -	P={1,2,3,4} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7	T= T-{4}={5,6,7} L'(5)= min{10, L'(4)+ -} L'(6)= min{9, L'(4)+ a(4,6)} L'(7)= min{-, L'(4)+ -}
P={1,2,3,4} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7 **İter 4**	T={5,6,7} L'(5)= 10 L'(6)= 9 L'(7)= -	P={1,2,3,4,6} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7 L(6)=9	T= T-{6}={5,7} L'(5)= min{10, L'(6)+ a(6,5)} L'(7)= min{-, L'(6)+ a(6,5)}
P={1,2,3,4,6} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7 L(6)=9 **İter 5**	T={5,7} L'(5)= 10 L'(7)= 17	P={1,2,3,4,6,5} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7 L(6)=9 L(5)=10	T= T-{5}={7} L'(7)= min{17, L'(5)+ a(5,7)}
P={1,2,3,4,5,6} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7 L(6)=9 L(5)=10)1-5) **İter 6**	T={7} L'(7)= 16 P=V ise dur.	P={1,2,3,4,5,6,7} L(1)= 0 L(2)= 4 L(3)=5 L(4)=7 L(5)=10 L(6)=9 L(7)= 16	T= Boş Küme

Şekil 4.10 Algoritma Aşamaları

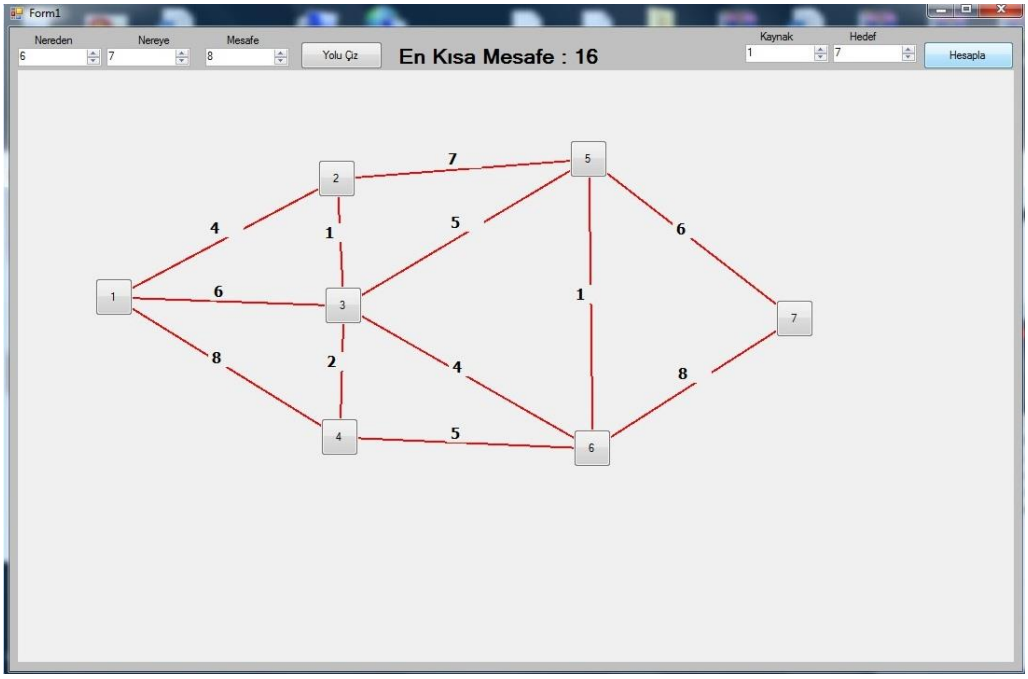
Şekil 4.9'daki örnekte optimum çözümü **16** olarak bulmaktadır. Şimdi aynı örneği bizim projemizde deneyelim.



Şekil 4.11 Form Ekranından Örnek

Nereden, Nereye ve **Mesafe** değerlerini girdikten sonra **Hesapla** butonuna tıklıyoruz.

Şekil 4.12'deki sonucu elde ediyoruz.



Şekil 4.12 *Sonucun Ekrana Yazdırılması*

Verilen örnek ile bizim yazdığımız form uygulamasında da görüldüğü gibi adımlar ve sonuçlar aynı çıkmıştır.

BÖLÜM 6. MAKSİMUM AKIŞ ALGORİTMASI

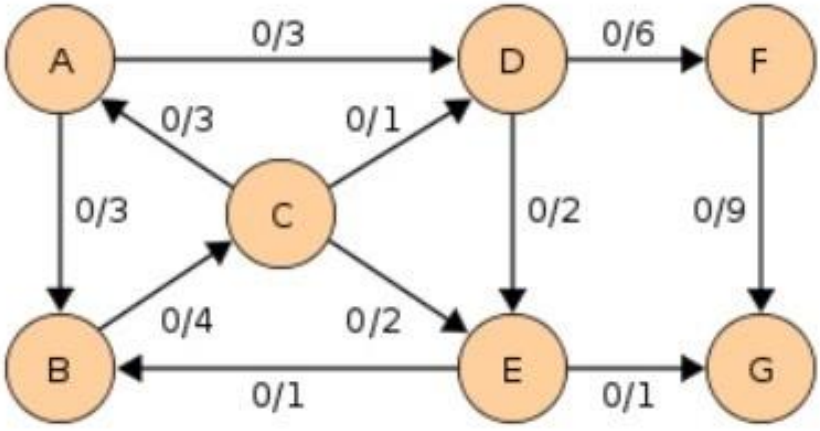
6.1-Maksimum Akış Algoritması(Maximum Flow Problem) Nedir?

Maksimum akış sorunu ilk olarak 1954 yılında **TE Harris** ve **FS Ross** tarafından basitleştirilmiş bir Sovyet demiryolu trafik akış modeli olarak formüle edilmiştir. Maksimum akış algoritmasının düşüncesi, kaynak ve sonuç düğümlerini bağlayan net pozitif akışlı çıkış yolunu bulmaktır. Bir işletmenin imalat birimlerinde ve depolarda darboğaz noktalarının, şehir planlama çalışmalarında trafik yoğunluklarının belirlenmesinde, benzeri şekilde telekomünikasyon ağlarının tasarımında etkili bir yöntem olarak kullanılabilir. Genel tanımla, verilen ağırlıklı yönlü G çizgesinde kaynak köşeden hedef köşeye maksimmum akışı bulmaktır. Ağırlıklar kapasiteleri göstermektedir. Literatürde farklı algoritmaları bulunmaktadır.Bu algoritmalardan en çok kullanılanları ***Edmonds Karp Algoritması*** ve ***Ford Fulkerson Algoritması*** algoritmalarıdır.

6.2- Edmonds Karp Algoritması Nedir?

Bu algoritmanın amacı, literatürde azami akış (maximum flow) olarak geçen ve düğümler (nodes) arasında akış kapasiteleri belirli bir şekildeki (graph) bir başlangıçtan bir hedefe en fazla akışın sağlandığı problemleri çözmektir. Azami akış (maximum flow) problemini örneğin şehirler arasında bağlı boru hattına veya tedarik zincirine benzetebiliriz.

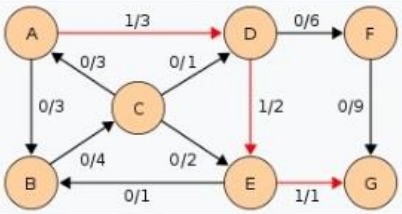
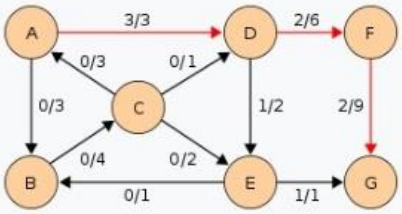
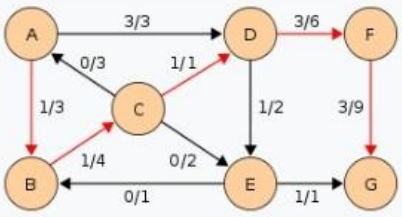
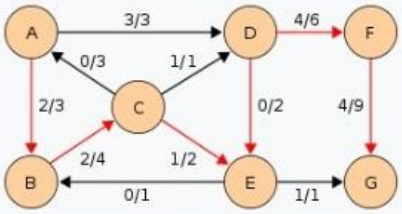
Şekil 5.1 örneği üzerinden algoritmayı açıklamaya çalışacağız.



Şekil 5.1 Edmonds Karp Algoritma Örneği

7 düğümü bulunan ve kaynak A,G ve lavoba kapasiteleri bir ağ üzerinde verilmiştir. Oklar akımı rakamlar ise kapasiteleri temsil etmektedir.

F / Cfcuvcf (u, v) = C (u, v) -f (u, v)uv

Kapasite	yol	Ortaya ağ
$\min(c_f(A,D), c_f(D,E), c_f(E,G))$ $= \min(3-0, 2-0, 1-0) =$ $= \min(3,2,1) = 1$	A, D, E, G	
$\min(c_f(A,D), c_f(D,F), c_f(F,G))$ $= \min(3-1, 6-0, 9-0)$ $= \min(2,6,9) = 2$	A, D, F, G	
$\min(c_f(A,B), c_f(B,C), c_f(C,D), c_f(D,F), c_f(F,G))$ $= \min(3-0, 4-0, 1-0, 6-2, 9-2)$ $= \min(3,4,1,4,7) = 1$	A, B, C, D, F, G	
$\min(c_f(A,B), c_f(B,C), c_f(C,E), c_f(E,D), c_f(D,F), c_f(F,G))$ $= \min(3-1, 4-1, 2-0, 0-(-1), 6-3, 9-3)$ $= \min(2,3,2,1,3,6) = 1$	A, B, C, E, D, F, G	

Şekil 5.2:Örnegin şekil üzerindeki çözümü

c (A, D) + c (C, D) + c (E, G) = 3 + 1 + 1 = 5

6.3- Ford Fulkerson Algoritması Nedir?

Bu algoritmanın amacı, literatürde azami akış (maximum flow) olarak geçen ve düğümler (nodes) arasında akış kapasiteleri belirli bir şekildeki (graph) bir başlangıçtan bir hedefe en fazla akışın sağlandığı problemleri çözmektir.

Azami akış (maximum flow) problemini örneğin şehirler arasında bağlı boru hattına veya tedarik zincirine benzetebiliriz. 2 önemli kavram içermektedir.

- Kalan Ağ
- Büyüyen Yol

Algoritması:

Girişler , bir Ağ Verilen akış kapasitesi ile , c , bir kaynak düğüm s ve bir lavabo düğümü $tG = (V, E)$
Çıkış hesaplayın bir akış f den s için t maksimum değerin

1. $f(u, v) \leftarrow 0$ Tüm kenarlar için (u, v)
2. Bir yol olsa da p gelen s için t olarak , bu şekilde , tüm kenarları için : $G_f c_f(u, v) > 0 (u, v) \in p$
    1. bulmak $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
    2. Her bir kenar için $(u, v) \in p$
        1. $f(u, v) \leftarrow f(u, v) + c_f(p)$(*Yol boyunca akışını Gönder*)
        2. $f(v, u) \leftarrow f(v, u) - c_f(p)$(*Akış daha sonra "iade" olabilir*)

Şekil 5.3 *Ford Fulkerson Algoritma Adımları*

BÖLÜM 7. MİNİMUM MALİYET KAPASİTELİ AKIŞ ALGORİTMASI

7.1-Minimum Maliyet Kapasiteli (Minimum-Cost Flow Problem) Akış Algoritması Nedir?

Minimum maliyetli akış problemine, düğümler için talep ve arz değerleri ile bağlantılarla ilgili maliyetler ve üst / alt sınırlar dahil edilebilir. Bu sorunun tipik uygulaması yol ağı ilişkilendirilen bazı kapasitesi ve maliyeti olan bir depoya bir fabrikadan iyi dağıtım yolu bulmayı içerir.

Minimum Maliyet Kapasiteli Akış problemi, dört açıdan Bölüm.6' da bahsedilen Maksimum Akış modelinin genellemesidir.

1. Tüm bağlantılar tek yönlüdür.
2. Bir (negatif olamayan) birim akış maliyeti her bir bağlantıyla ilişkilendirilir.
3. Bağlantılar pozitif alt kapasite sınırlarına sahip olabilir.
4. Şebekedeki herhangi bir düğüm başlangıç veya bitiş olarak hareket eder.

Yeni model, bağlantılardaki akış sınırlamalarını ve düğümlerdeki üretim ve talepleri karşılayacak farklı bağlantılardaki akışları belirler.

7.2- Minimum Maliyet Kapasiteli Akış Algoritması Tanımı ve Aşamaları

Tanım: En küçük maliyetli akış problemi, verilen şartlarda kullanılabilir arzın ağ boyunca taşınarak talepleri en küçük maliyetle taşınması olarak tanımlanır.

m düğüm ve n bağlantıdan oluşan bir G(N,S) ağını göz önüne alalım. N kümesindeki her i düğümü için bir bi tanımlanır.

bi;

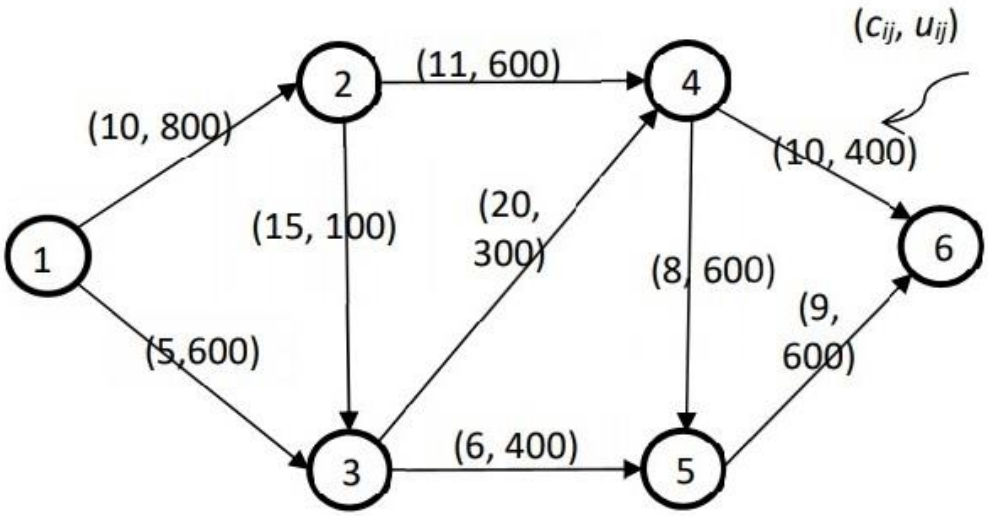
- eğer bi > 0 ise arz miktarını
- eğer bi < 0 ise talep miktarını ifade eder.

i düğümleri;

- eğer bi > 0 ise arz noktası
- eğer bi < 0 ise talep noktası olarak sınıflandırılabilir.
- Eğer bi = 0 ise i düğümü arz veya talep noktası değildir, geçici konaklama veya ara nokta olarak isimlendirilebilir. Her (i,j) S bağlantısının elemanı ve S bağlantısı için bağlantı üzerindeki akış xij gösterilsin. Ayrıca her bağlantı için cij maliyeti ve bağlantı üzerindeki akışın en büyük ve en küçük miktarları uij ve □ij verilsin.

7.3- Minimum Maliyet Kapasiteli Akış Algoritması Örneği

Şekil 7.1'de verilen yol ağına her saat 1 noktasından 900 araç girmektedir. Bu araçlardan 300 tanesi 4 noktasından, 500 tanesi 6 noktasından ve 100 tanesi de 5 noktasından çıkacaktır. Şekil üzerinde her bağlantı üzerinde, araçların bağlantıyı geçme süresi (dakika olarak) ve bağlantıdan bir saatte geçebilecek azami araç sayısı verilmiştir. tüm araçların 1 noktasından 4, 5 ve 6 noktalarına en kısa sürede varması için problemi en küçük maliyetli akış problemi olarak modelleyiniz.



Şekil 7.1 Düğümler Arası İlişki

BÖLÜM 8. KRİTİK YOL ALGORİTMASI

8.1-Kritik Yol Algoritması Nedir?

Projelerin planlamasında birçok yöntem kullanılmaktadır. İşin ne zaman başlayacağı, ne zaman biteceği, ne kadar süreceği ve nelerin olacağına dair birçok verinin yer aldığı yöntemler ve diyagramlar ile projeler desteklenir.Bu algoritmalardan en çok kullanılanları ***Kritik Yol Yöntemi (CPM)*** ya da ***Kritik Yol Analizi (EBM)*** algoritmalarıdır.Bu çalışmada ***Kritik Yol Yöntemi (CPM)*** inceleyeceğiz.

8.2-Kritik Yol Yöntemi (Critical Path Method-CPM) Nedir?

Kritik Yol Yöntemi (CPM) ***Morgan R. Walker*** tarafından 1950'lerin sonlarında geliştirilen bir proje modelleme tekniğidir. Kritik Yol Yöntemi programların yapımı, araştırma faaliyetlerinin planlanması problemleri ile eşgüdümü gerektiren bir plan için oldukça yarar sağlar. Ağ modelleri birçok faaliyet içeren büyük ve karmaşık projeleri çizelgelemek için kullanılabilir. Eğer tüm faaliyetlerin süreleri kesin olarak biliniyorsa projenin tümünün bitirilmesi için gerekli süre ***Kritik Yol Yöntemi (CPM -Critical Path Method)*** ile belirlenebilir. CPM ile faaliyetlerin proje toplam süresini uzatmadan ne kadar ertelenebileceğini bulmak için de kullanılabilir. Bir çok projede ele alınacak faaliyetlerin sayısı çok olduğu gibi, faaliyetlerin sıralı ilişkisi de karmaşıktır. Bu durumda, yapılacak işi şekillendirerek (şema/şebeke çizerek) açık bir görünüm ortaya koymak gerekir. Şemanın çizilmesi olanaklı faaliyetlerin unutulmamasını sağlar.

Kritik Yol Yöntemi (CPM) ile planlanan bir projenin tamamlanabilmesi içinse bazı adımların bilinmesi gereklidir. Bunlar;

- Projenin iş akışı yapısı, böylece projenin her aşaması belirlenir,
- İşlem süreleriyle ilgili tahminler,
- Faaliyetler arasındaki mantıksal ilişkiler.

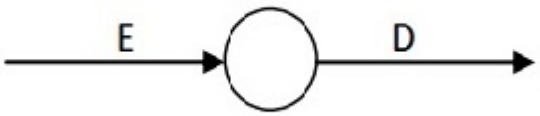
8.3- Kritik Yol Yöntemi Şebeke Gösterimi

Kritik Yol Algoritmasının temeli şebekedir. Şebekenin dalları ortaya konulacak faaliyetleri gösterir. Dalların yönü her zaman başlangıçtan bitişe doğru olduğundan, oklar ile gösterilir. Noktalar veya

daireler, daha önce de ifade edildiği gibi hem olayları gösterir hem de faaliyetlerin tamamlanmasını veya başlamasını belirtirler.

Faaliyet listesi ve faaliyet öncelikleri verilen bir projenin şebeke/ok diyagramını oluşturmak için aşağıdaki kurallar izlenmelidir:

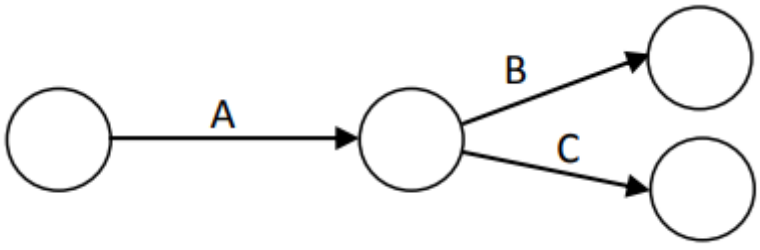
- Olay 1 (nokta) projenin başlangıcını temsil eder. Olay 1' de başlayan faaliyetten önce gelen faaliyet yoktur.
- Projenin tamamlandığını temsil eden bitim noktası veya olay şebekeye dahil edilmemelidir.
- Şebekedeki olayların sayısı başlangıçtan bitime doğru artar. Bir anlamda birden fazla olay sayılandırması vardır.
- Şebekedeki her faaliyet yalnızca bir ok ile gösterilir.



Şekil 7.2 Düğümler Arası İlişki 2

D faaliyeti başlamadan önce E faaliyetinin bitmiş olması gerekir.

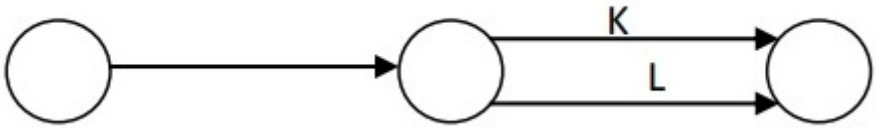
- İki olay en fazla bir ok ile birleştirilir. Her faaliyet iki ayrı düğüm ile tanımlanır.



Şekil 7.3 Düğümler Arası İlişki

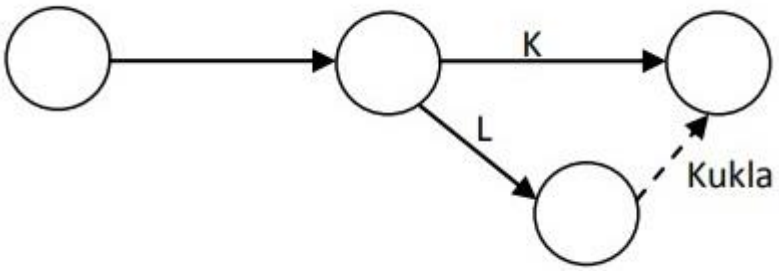
B ve C faaliyetlerinin başlaması için A faaliyetinin bitmiş olması gerekir.

- K ve L iki faaliyet olsun. Bu iki faaliyetten hemen önce ve hemen sonra gelen faaliyetler aynı olsun. Bu durumda şekil;



Şekil 7.4 Düğümler Arası İlişki

biçiminde olur. Fakat, bu durum belirtilen kurala uymaz. İki düğüme iki faaliyet bağlanamaz. Bu sorunu çözmek için kukla faaliyetler oluşturulur. Kukla faaliyetler, faaliyet zamanı gerektirmeyen faaliyetlerin sırasını gösterir. Zaman ve kaynak harcamayan faaliyetlerdir. Kukla faaliyetler, süresi kısa olan faaliyetlerden sonra gelir.



- Doğru öncelik ilişkileri kurabilmek için şebekeye her yeni faaliyet eklenirken aşağıdaki sorular sorulmalıdır:

1. Şebekeye eklenecek faaliyetten hemen önce hangi faaliyetlerin gelmesi zorunludur?
2. Şebekeye eklenecek faaliyeti hangi faaliyetin izlemesi zorunludur?
3. Şebekeye eklenecek faaliyetler ile eş zamanlı olarak hangi faaliyetler bulunmalıdır?

Bu sorulara cevap verebilmek için faaliyetler arasında uygun öncelik ilişkisini sağlayacak kukla faaliyetlerin kullanımı gerekebilir.

8.4- Kritik Yol Yöntemi ile Proje Ağının Çizilmesi

1. Düğüm 1, projenin başlangıcını ifade eder. 1'den çıkan bağlantılar önceliği olmayan faaliyetlerdir.
2. Projenin bitişini ifade etmek üzere bir bitiş düğümü ilave edilmelidir.

3. Ağdaki düğümler öyle numaralandırılmalıdır ki; herhangi bir faaliyetin bittiğini gösteren düğüm her zaman faaliyetin başladığını gösteren düğümden daha büyük numara ile ifade edilmelidir.

4. Bir faaliyet ağda birden fazla bağlantı ile gösterilemez.

5. İki düğüm sadece bir bağlantı ile birleştirilebilir.

6. 4. ve 5. kurallara uymak için ağa sıfır süreli bir yapay faaliyetler eklenebilir.

8.5- Kritik Yol Yöntemi Aşamaları

- **Düğüm Noktalarının En Erken Olay Zamanlarının Tespiti:** En erken olay zamanı (TE), olayın başlangıç teşkil ettiği bütün faaliyetlerin mümkün olan en erken başlama zamanlarına tekabül eder ve projenin son faaliyetlerinin bitiş noktasını teşkil eden olayın (TE) si projenin bitirilmesinin için gereken zamanı verir. Bir olay, birden fazla faaliyetin bitiş noktasını teşkil ediyor ise, bu faaliyetlerin tümü tamamlanmadan ilgili olay gerçekleşmez. Dolayısıyla olayın TE değeri aynı zamanda bu olayda son bulan tüm faaliyetlerin sonuncusunun tamamlanabileceği en erken süreye tekabül eder. Bu çalışmada TE değerlerinin gösterimi, her okun başında $(TE)_i$ ve sonunda $(TE)_j$ olarak gerçekleştirilecektir. En erken olay zamanının tespiti için gerekli işlem şebekenin başından sonuna doğru yani şebekenin solundan sağına doğru gerçekleştirilmelidir.Her olay için en erken olay zamanı, bitimi o olayda olan faaliyetin tamamlanma süresinin yine aynı faaliyetin başlangıcındaki en erken olay zamanına eklenmesi ile elde edilir. Eğer bir olay bir kaç faaliyetin bitim noktasını teşkil ediyorsa, faaliyetlerden sadece tamamlanma süresi en büyük olan işleme alınır.En erken olay zamanı şu şekilde formüle edilebilir :

$$(TE)_j = (TE)_i + Max.\ (Dij)$$

Dikkat edilmesi gereken önemli bir husus şebekenin başlangıç noktasının TE değerinin “0 ” sıfır olduğudur.

- **Düğüm Noktalarının En Geç Olay Zamanlarının Tespiti:**

En erken olay zamanı projenin programlanabilmesi için gerekli olan faaliyet zamanlarının belirlenmesinde tek başına yeterli değildir. Çünkü her faaliyetlerin en erken olay zamanında

başlamak zorunluluğu yoktur. Sonraki başlıklarda da ifade edileceği üzere, özellikle kritik yol üzerinde bulunmayan faaliyetlerin proje tamamlanma süresini aksatmadan belli bir süre geciktirilmeleri mümkündür. Bu eksikliği gidermek amacıyla "en geç olay zamanı" olarak adlandırılan ikinci bir zaman parametresi tarif edilmiştir. En geç olay zamanı, olayda son bulan bütün faaliyetlerin projenin tamamlanma süresini geciktirmeden tamamlanabileceği, mümkün olan en geç bitiş tarihini gösterir. Çalışma boyunca en geç olay zamanı (TL) ile gösterilecektir. Her okun başında $(TL)_i$ ve sonunda $(TL)_j$ en geç olay zamanı olacaktır. En geç olay zamanının tespiti için gerekli işlem şebekenin sonundan başına doğru yani şebekenin sağından soluna doğru gerçekleştirilmektedir. Her olay için en geç olay zamanı, başlangıcı o olayda olan faaliyetin tamamlanma süresinin yine aynı faaliyetin sonucundaki en geç olay zamanından çıkarılması ile elde edilir. Eğer bir olayda bir kaç faaliyetin başlangıcı var ise faaliyetlerden sadece tamamlanma süresi en küçük olan işleme alınır. Bu işlem şu şekilde formüle edilebilir:

$$(TL)_i = (TL)_j - Min\ (D_{ij})$$ 'dir.

Bu işlem yapılırken dikkat edilmesi gereken en önemli husus, şebekeyi oluşturan olayların sonuncusunun TE değerinin TL değerine eşit olduğudur. En erken ve en geç olay zamanları tespit edildikten sonra, projenin hedeflenen tamamlanma süresiyle alakalı analizler yapılmaya başlanabilir.

Faaliyetlerin En Erken Başlama Süresi (Early Starting Time:ES): Bir faaliyetin en erken başlama süresi, faaliyetin başlayabileceği mümkün olan en erken zamanı ifade eder ve faaliyete başlangıç teşkil eden olayın en erken olay zamanına tekabül eder. En erken başlama zamanın formülasyonu şu şekilde ifade edilebilir:

$$\mathbf{ES = (TE)_i}$$

En Erken Bitme Süresi (Early Finishing Time :EF) : Bu süre en erken başlama süresine faaliyet süresinin eklenmesiyle bulunmaktadır ve bir faaliyetin proje programını aksatmadan bitebileceği en erken zamanı göstermektedir. Bir faaliyetin en erken bitme süresi şu şekilde formüle edilebilir:

$$\mathbf{EF = ES + D_{ij} = (TE)_i + D_{ij}}$$

En Geç Bitme Süresi (Late Finishing Time:LF): Bir faaliyetin en geç bitme süresi, o faaliyetin bitim noktasını teşkil eden olayın en geç olay zamanına tekabül eder. En geç bitme süresi bir faaliyetin tüm projeyi geciktirmeden bitebileceği en geç süredir. En geç bitme süresi şu şekilde formüle edilebilir:

$$LF = (TL)_j$$

En Geç Başlama Süresi (Late Starting Time LS): Bu süre bir faaliyetin, tüm projeyi geciktirmeden başlayabileceği en geç suredir. En geç başlama süresi, her faaliyetin en geç bitme süresinden bu faaliyetin tamamlanma süresinin çıkarılmasıyla bulunmaktadır. En geç başlama süresi şu şekilde formüle edilebilir:

$$LS = LF - D_{ij} = (TL)_j - D_{ij}$$

8.6- Kritik Yolun Belirlenmesi

Kritik yol; şebekenin başlangıç ve bitiş olaylarını birleştiren, tamamlanma zamanı açısından en büyük değerlere sahip ve toplam boşluk değeri sıfır olan faaliyetlerin teşkil ettiği faaliyetler dizisi olarak ifade edilebilir. Kritik yol, projenin tamamlanma süresini belirler ve kritik yolu teşkil eden faaliyetlerden her hangi birinin programlanan zamandan geç tamamlanması tüm projenin aksamasına, projenin programlanan tamamlanma süresinin uzamasına neden olur.Bir yolun kritik olabilmesi için gerçekleşmesi gereken şartlar şu şekilde ifade edilebilir:

- **1.** $(TE)_i=(TL)_i$
- **2.** $(TE)_j = (TL)_j$
- **3.** $(TE)_j - (TE)_i = (TL)_j - (TE)_i =$ **Faaliyet süresi** (D_{ij})

Bir yolun kritik olabilmesi için bu üç şartın mutlaka kritik yoldaki her olay için aynı anda gerçekleşmiş olması lâzımdır ve bu şartları sağlayan olayları bağlayan faaliyetlerde kritik faaliyetler olarak tanımlanır.

8.7- Kritik Yol Yönteminin Cevaplayabileceği Sorular

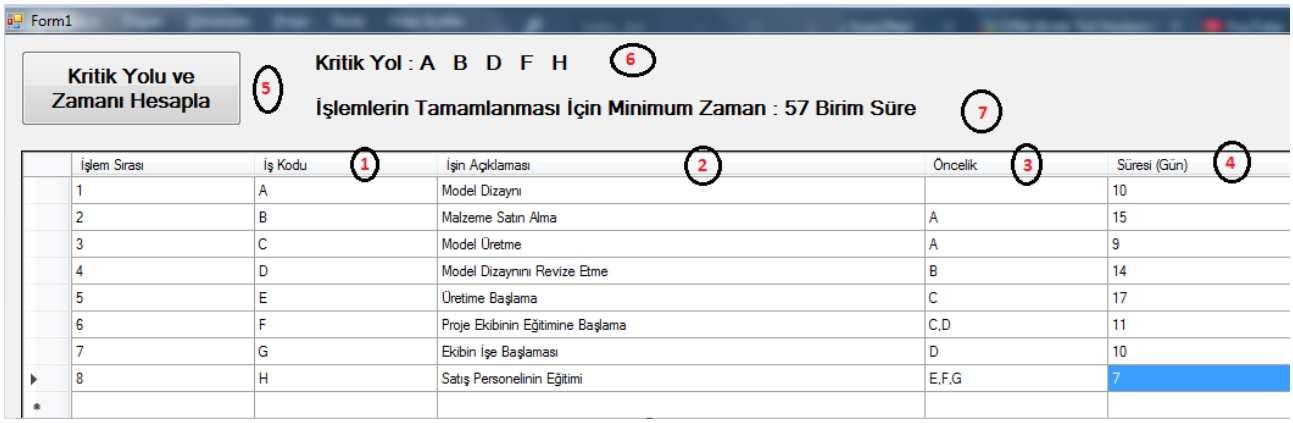
- Proje ne zaman bitecek?
- Projedeki kritik faaliyetler ve işler neler?
- Kritik olmayan faaliyetler hangileri?
- Belirli bir zamanda projenin bitme olasılığı ne?
- Proje plana göre yürüyor mu? Planın önünde mi? Planın gerisinde mi?

- Proje bütçenin üzerinde mi? Altında mı?
- Projeyi zamanında bitirebilmek için yeterli kaynak var mı?
- Eğer proje planlanandan önce bitirilmek isteniyorsa bu en az maliyet ile nasıl yapılabilir?

8.8- Kritik Yol Yöntemi Kullanıldığı Alanlar

- Yüzme havuzları, ofis binaları veya otoyollar gibi inşaat projelerinin çizelgelenmesinde
- 400 yataklı bir hastanenin Portland' dan şehrin dışına taşınması faaliyetinin çizelgelenmesinde
- Uzay uçuşlarında fırlatma faaliyeti için geri sayım ve benzeri prosedürlerin geliştirilmesi
- Yeni bir bilgisayar sisteminin kurulması
- Yeni bir ürünün geliştirilmesi ve pazarlaması
- Bir şirket birleşmesinin tamamlanması
- Bir geminin inşa edilmesi

8.9- Kritik Yol Yöntemi Kodlarımız ve Açıklamaları



Şekil 8.1 Kritik Yol Hesaplama Formu

Şekil 8.1'deki numaraların işlevleri aşağıda değinilmiştir.

1) İş kodunun yazıldığı kısımdır.
2) İşin açıklamasının yazıldığı kısımdır.

3) Önceliklerin yazıldığı kısımdır.

4) İş sürelerinin yazıldığı kısımdır.

5) Kritik yolun ve zamanın hesaplanması için koyulan butondur.

6) Kritik yolun adımlarıdır.

7) İşlemlerinin tamamlanması için minimum zaman gereksinimin yazıldığı kısımdır.

```
private void dataGridView1_RowsAdded(object sender, DataGridViewRowsAddedEventArgs e)
{
    sira++;
    dataGridView1.Rows[sira - 1].Cells[0].Value = sira;
}
```

Şekil 8.2 Otomatik Yeni Satır Ekleme Kodu Olayları

Şekil 8.2'deki kod otomatik bir şekilde yeni bir satır açmamıza yaramaktadır.

```
private bool varm(string kelime, string aranan)
{
    bool varmi = false;
    for (int i = 0; i < kelime.Length; i++)
    {
        if (kelime[i] == Convert.ToChar(aranan))
            varmi = true;
    }
    return varmi;
}
```

Şekil 8.3 Girilen Kelimeyi varm Fonksiyonunda Arama Kodu

Şekil 8.3'deki kod görselinde Bool tipinde varm fonksiyonu oluşturduk. İçerisine iki adet parametre yer almaktadır. Bunlardan ilki string tipindeki kelime değişkenimizdir. İkincisi ise yine string tipinde aranan değişkenidir. Bu fonksiyonun içerisinde varmi adında bool tipinde true yada false döndüren bir değişken tanımladık. Bu değişkeni ilk etapta false değeri atadık. Daha sonra for döngüsü açık i'yi sıfırdan başlatıp i kelime uzunluğundan büyük olunca döngüden çıkarılıncaya kadar döngü devam edecektir. Bu döngünün içerisinde de bir adet if koşul bloğu bulunmaktadır. Bu koşul bloğu kelime[i] indisinin aranana eşit mi diye kontrolü var yapılmaktadır. Eğer koşul doğruysa varmi değişkeni true olacaktır. Return varmi diyerek geriye varmi değişkeninin true yada false olduğunu döndürüyoruz.

```
int[] EB;
int[] GB;
int[] ES;
int[] GS;
1 başvuru
private void button1_Click(object sender, EventArgs e)
{

    //List<int> liste = new List<int>();
    EB = new int[dataGridView1.Rows.Count - 1];
    GB = new int[dataGridView1.Rows.Count - 1];
    ES = new int[dataGridView1.Rows.Count - 1];
    GS = new int[dataGridView1.Rows.Count - 1];
    string oncelik = "";
    string[] oncelikDizi;
    int buyuk = 0;
```

Şekil 8.4 Dizilerin Oluşturulması

EB = Erken Başlangıç Zamanı GB = En Geç Başlangıç Zamanı

ES = En Erken Sonlanma Zamanı GS = En Geç Sonlanma Zamanı

İnt türünde EB, GB,ES ve GS olmak üzere 4 adet dizi oluşturduk. Daha sonra bu oluşturduğumuz dizilere datagridimizin satır sayısından bir eksiği kadar boyut verdik. String türünde öncelik adında bir değişken oluşturduk. String türünde önceliklerimizi tutacak öncelik dizimizi oluşturuyoruz. İnt türünde büyük adında bir değişken oluşturup içerisine sıfır atıyoruz.

```
for (int i = 0; i < dataGridView1.Rows.Count - 1; i++)
{
    if (i == 0)
    {
        EB[i] = 0;
        ES[i] = Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value.ToString());
    }
    else
    {
        if (dataGridView1.Rows[i].Cells[3].Value != null)
        {
            oncelik = dataGridView1.Rows[i].Cells[3].Value.ToString();
        }
        else
        {
            MessageBox.Show("Öncelik Kısmı Boş Geçilemez!!!");
        }
        oncelikDizi = oncelik.Split(',');
        if (oncelikDizi.Length == 1)
        {
            for (int j = 0; j < i; j++)
            {
                if (oncelikDizi[0] == dataGridView1.Rows[j].Cells[1].Value.ToString())
                {
                    EB[i] = ES[j];
                    ES[i] = EB[i] + Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value.ToString());
                    break;
                }
```

Şekil 8.5 *Girilen Satırın Öncelik Sırasının Kontrol Edilmesinin Kodu*

Şekil 8.5'deki kod görselinde sıfırdan başlayarak data gridimizin içerisindeki satır sayısının bir azaltılmış haline sayısı kadar döndürüyoruz. Bir azaltmamızın sebebi data gridimizin en alt satırında hep oluşan bir boş satırın bulunmasından kaynaklanmaktadır. İlk if koşul ifadesinde birinci satırda bulunan öncelik sütunumuzun boş olabilirliğini ayarlıyoruz. Eğer boş değilse else kısma geçiyoruz. Bu seferde Öncelik sütunundaki değerlerinin boş olup olmadığının kontrolünü yapıyoruz. Eğer doluysa öncelik değişkenimize atıyoruz. Eğer boşsa hata mesajı verdiriyoruz. Split ifadesinin bulunduğu kısım öncelik satırımızın içerisinde virgül ile ayrılmış kısımları varsa yani A,B,C gibiyse onu parçalamamıza yaramaktadır. Ve öncelik dizimizin içerisine atılmaktadır. İkinci İf koşumuzda Öncelik dizimiz tek elemanlıysa data gridimizin tüm satırlarının ikinci sütunlarını öncelik dizimizin sıfırıncı satırına atıyoruz. Hangi j indisindeyse erken sonlanma zamanını erken başlangıç zamanın içerisindeki i'ninci indisine atıyoruz. Hangi i'ninci indisteysek erken başlangıç zamanını data grid'in içerisinde bulunan satırının 5. Sütunun içerisindeki gün süresiyle topluyoruz. Ve sonucu en erken sonlanma zamanının içerisinde i'ninci indise atıyoruz. Break ile döngüden çıkıp bir sonraki adıma devam ediyoruz.

```
else if (oncelikDizi.Length > 1)
{
    for (int j = 0; j < i; j++)
    {
        for (int k = 0; k < oncelikDizi.Length; k++)
        {
            if (dataGridView1.Rows[j].Cells[1].Value.ToString() == oncelikDizi[k])
            {
                if (ES[j] > buyuk)
                {
                    buyuk = ES[j];
                }
            }
        }
    }
    EB[i] = buyuk;
    ES[i] = EB[i] + Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value.ToString());
    buyuk = 0;
}
else
{
    MessageBox.Show("Öncelik Kısmı Boş Geçilemez!!!");
}


label2.Text = "İşlemlerin Tamamlanması İçin Minimum Zaman : " + ES[dataGridView1.Rows.Count - 2].ToString() + " Birim Süre ";
```

Şekil 8.6 Kontrollerin Yapılıp Ekrana Yazdırılması

Şekil 8.6'daki kod görselinde Else if koşumuzda öncelik dizimizin uzunluğunun bir den büyük olduğu bölüm için çalışmaktadır. İf bloğumuz data gridin içerisindeki tüm satırların ikinci sütunlarında bulunan iş adlarıyla öncelik dizimizin içerisinde bulunan değerlerle kontrol edilecek eğer birbirlerine eşitse ve en erken sonlanma zamanının j'ninci indisinden büyük mü olduğu kontrol

edilecek. Eğer büyükse en erken sonlanmanın içindeki j'ninci indisteki değer büyüğün içerisine atanacak.

For döngülerinden çıkıldıktan sonra ise büyük değişkeni erken başlangıç zamanının i'ninci indisine eşit olacak. Erken başlangıç zamanının i'ininci değeri ile Data gridin i'ninci satırındaki gün sayısı ile toplanacaktır. Ve bulunan değeri en erken sonlanma zamanının içerisine atıyoruz. Son olarak büyük değişkenini sıfıra eşitleyip else if'ten çıkıyoruz. Label2. Text minimum bulunan sayımızı yazdırıyoruz. Bu sayıyı yazdırırken data grid'in içerisindeki bir adet boş satırdan dolayı bir azaltma yapıyoruz. Bir de dizimizin sıfırıncı indisinden olayı bir azaltma yapıyoruz. Yapılan iki azaltma sonucunda en erken sonlanma zamanını elde ediyoruz.

```
int sayac = 0, syc1mi = 0, kucuk = 10000;
bool varmi = false;
for (int i = dataGridView1.Rows.Count - 2; i >= 0; i--)
{
    if (i == dataGridView1.Rows.Count - 2)
    {
        GS[i] = ES[i];
        GB[i] = GS[i] - Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value.ToString());
    }
    else
    {
        for (int j = dataGridView1.Rows.Count - 2; j > i; j--)
        {
            oncelik = dataGridView1.Rows[j].Cells[3].Value.ToString();
            varmi = varm(oncelik, dataGridView1.Rows[i].Cells[1].Value.ToString());
            if (varmi == true)
            {
                syc1mi = j;
                sayac++;
            }
            varmi = false;
        }
        if (sayac == 1)
        {
            GS[i] = GB[syc1mi];
            GB[i] = GS[i] - Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value.ToString());
        }
```

Şekil 8.7 Kritik Yolun Hesaplanması

Şekil 8.7'deki kod görselinde, kritik yolda asıl işlemlerin yapıldığı kısma geçiyoruz. İnt tipinde sayaç ve syc1mi değişkenlerini oluşturup bunları sıfıra eşitliyoruz. İnt tipinde küçük adından bir değişken oluşturup 10000 değerine eşitliyoruz. Varmi adında bir adet bool değişkeni oluşturup başlangıç olarak false değerini atıyoruz. Buradaki ilk forda satır sayımızın iki çıkarılmış halinden başlayarak i sıfır oluncaya kadar azaltıyoruz. İki azaltmaktaki nedeni daha önce de değinmiştir tekrar değinmek istedik.

Data grid'in içerisindeki bir adet boş satırdan dolayı bir azaltma yapıyoruz. Daha sonra bir adette dizimizin sıfırıncı indisinden olayı bir azaltma yapıyoruz. Bu yüzden iki adet azaltma yapıyoruz. İlk ifteki kontrolde i sonuncu satırdaysa en erken sonlanma zamanının i'ninci değeri en geç başlangıç

zamanının i'ninci indisine atıyoruz. Ardından en geç sonlanma zamanın i'ninci değerinde bulunduğumuz gün sayısını çıkartıyoruz. Çıkarma sonucunu en geç başlangıç zamanının i'ninci indisine atıyoruz. Else kısmı eğer i sonuncu satırda değilse bu kısım çalışacak anlamına gelmektedir. Buradaki forda j sonuncu satırdan başlayacak j i'den büyük ise j azaltılarak devam edecektir. Burada öncelik kontrolü yapılmaktadır. Varm fonksiyonu çalıştırılıp önceliği varsa varmi değişkeni true yada false dönecektir. Daha sonraki if koşul ifadesi true ise j syc1mi değişkenin içerisine atılıp sayaç bir arttırılacaktır.

Varmi true değilse false olarak işleme alınıp devam edilecek. Bir sonraki if koşul ifadesinde sayaç bir ise bu koşul çalıştırılacak. En geç başlangıç zamanının içerisinde bulunan dizi elemanından syc1mi deki elemanı en geç sonlandırma zamanının i'ninci indisine eşitliyoruz. Daha sonra en geç sonlanma zamanının bulunduğu i'ninci indisinden data grid'in i'ninci satırında bulunan 5. Sütunundaki gün sayısını çıkartıyoruz. Ve son olarak sonucu en geç başlangıç zamanının içerisine yazıyoruz.

```
        if (sayac > 1)
        {
            for (int j = dataGridView1.Rows.Count - 2; j > i; j--)
            {
                oncelik = dataGridView1.Rows[j].Cells[3].Value.ToString();
                varmi = varm(oncelik, dataGridView1.Rows[i].Cells[1].Value.ToString());
                if (varmi == true)
                {
                    if (kucuk > GB[j])
                    {
                        kucuk = GB[j];
                    }
                }
                varmi = false;
            }
            GS[i] = kucuk;
            GB[i] = GS[i] - Convert.ToInt32(dataGridView1.Rows[i].Cells[4].Value.ToString());
        }
        sayac = 0;
        kucuk = 10000;
    }
}
for (int i = 0; i < dataGridView1.Rows.Count - 1; i++)
{
    if (ES[i] == GS[i] && EB[i] == GB[i])
    {
        label1.Text += dataGridView1.Rows[i].Cells[1].Value.ToString() + "    ";
    }
```

Şekil 8.8 Kritik Yolun Hesaplanması 2

Şekil 8.8'deki kod görselinde, Diğer resimde sayacın bire eşit olduğu durumu inceledik. Bu resimde ise sayacın bir den büyük olduğu durumu inceleceğiz. İlk if koşulu eğer sayaç bir den büyükse burası çalışacak. For döngüsünde ise öncelik kontrolü yapılmaktadır. Varm fonksiyonu çalıştırılıp

önceliği varsa varmi değişkeni true yada false dönecektir. Sonraki if koşul ifadesi varmi değişkeni true ise bu kısım çalışacak anlamına gelmektedir. Sonraki if koşulunda ise küçük değişkenimiz büyük ise en geç başlangıç zamanındaki j'ninci indisten en geç başlangıç zamanının içerisindeki [j]'ninci elamanı küçüğün içerisine atıyoruz.

İf bittikten sonra Varmı değişkeni true değilse false olacağından false eşitleyip for döngüsü devam ettiriyoruz. For döngümüz bittikten sonra küçük değişkenimizin içerisinde bulunan sayısı en erken sonlama zamanının i'ninci indisine atıyoruz. Ardından En geç sonlanma zamanın i'ninci değerinden data gridimizin i'ninci satırında yer alan 5. Sütunundaki gün değerini çıkartıyoruz. Son olarak en geç sonlanma zamanının i'ninci indisinin içerisine atıyoruz. İf koşul ifadesinden çıkmış oluyoruz. Sayacı sıfır yapıyoruz. Küçük değişkenimizi de 10000 yapıp else den çıkmış oluyoruz.

For döngüsü koşulu tamamladıktan sonra döngüden çıkıyoruz. En sondaki for döngüsünü data grid'inimiz satır sayısı kadar döndürüyoruz. En erken sonlanma zamanının i'ninci değeri en geç sonlanmanın i'ninci değerine eşit mi diğer kontrol ediyoruz. Ve erken başlangıç zamanının i'ninci değeri ile en geç başlangıç zamanındaki i'ninci değer birbirine eşitse data gridin satırlarının okuduğumuz 2. Sütunlarını label'ımıza yazdırıyoruz. Örn: Kritik Yol: "A, C, D, F " dir.

BÖLÜM 9. ALGORİTMALARIN DEĞERLENDİRMELERİ

9.1-Prim Algoritmasının Analizi

• Çalışma Zamanı:

– Herbir düğümü S de değil onun en küçük ağırlıklı komşularının olduğu S de bulundurarak algoritmayı daha verimli hale getirebiliriz.

• Bu komşunun maliyeti cost[v] de ve komşuları other[v] de saklanır.

– Aynı küme işlemlerini Dijkstra'nın algoritmasındaki gibi yaparız(yapıyı ilklendirme, değerleri m kerede azalt, en küçüğü n-1 kerede seç).

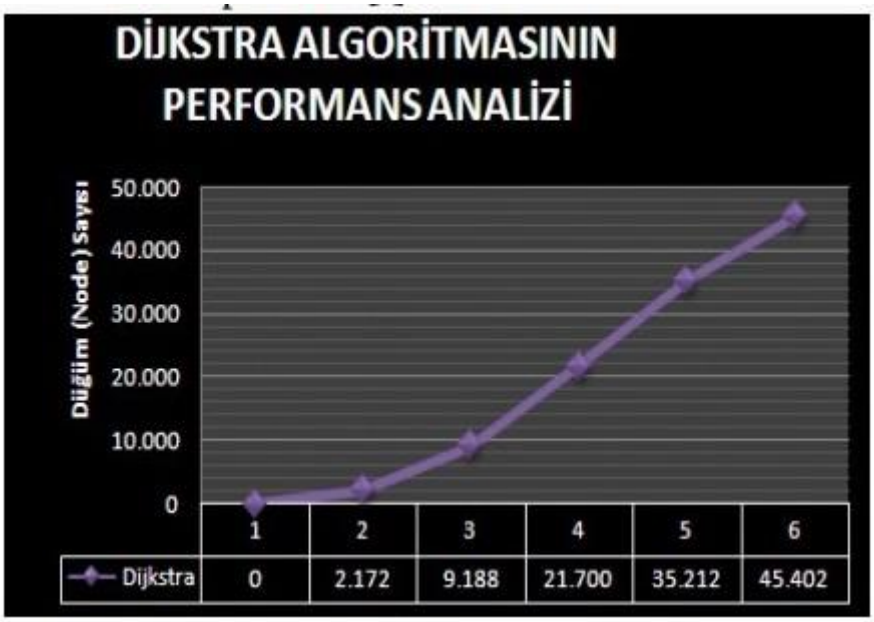
Bu yüzden bir dizi ile gerçeklediğinde $O(n^2)$ zaman bulunur, bir heap ile gerçeklediğinde, $O((n + m)$ log $n)$ zaman bulunur.(n düğüm sayısı, m ayrıt sayısı)

Prim'in algoritmasının icra zamanı sadece düğümlerin sayısına bağlıdır, fakat aynı sayıda düğümlü bir graf için ayrıtların sayısı artarken Kruskal'ın algoritması artar. Bununla birlikte genelde hangisinin daha verimli olduğunu söylemek mümkün değildir. Verimlilik, ağı n yapısına ve ağırlığın dağılımına bağlıdır. Çoğu durumda oluşturulan veri yapısı verimliliğe doğrudan etkilidir.

9.2-Dijkstra Algoritmasının Analizi

```
1:  function Dijkstra(Graph, source):
2:  for each vertex v in Graph:
3:  dist[v] := infinity
4:  previous[v] := undefined
5:  dist[source] := 0
6:  Q := the set of all nodes in Graph
7:  while Q is not empty:
8:  u := node in Q with smallest dist[ ]
9:  remove u from Q
10: for each neighbor v of u:
11: alt := dist[u] + dist_between(u,v)
12: if alt < dist[v]
13: dist[v] := alt
14: previous[v] := u
15: return previous[ ]
```

Şekil 9.1:Dijkstra Algoritmasının Pseudo Kodu



Şekil 9.2: Dijkstra Algoritmasının Performans Analizi Grafi

Şekil 9.2'de Dijkstra kodunun uygulanması sonucu çıkan matematiksel istatik grafiği verilmiştir. Kablosuz Algılayıcı Ağlarda düğüm (node) sayısına göre Dijkstra algoritmasının çalışma zamanları (running time) belirtilmiştir. Algılayıcı (node) sayısı arttıkça; harcanan zamanda artmıştır. Örneğin : 10000 noda sahip olan bir ağın çalışma hızı başlangıçta 2.172 saniye iken, 50000'e ulaştığında çalışma hızı 45.402 saniyedir.

Kaynakça:

- Doğrusöz, H. (1975), "Türkiye'de Yöneylem Araştırması," Yöneylem Araştırması: bildiriler 75 (düz. M. Oral ve Ü. Çınar), Marmara Bilimsel ve Endüstriyel Araştırma Enstitüsü, Gebze.(1980),
- "Türk Toplumunda Yöneylem Araştırmasının Yeri, Rolü ve Gelişme Yönü", Y.A.Derg., Vol. I, 1.
- OPTİMİZASYON MODELLERİ VE ÇÖZÜM METODLARI Doç.Dr. Metin Türkay
- Kablosuz Algılayıcı Ağlarda, Yönlendirme Algoritmalarının Performans Analizi

  Yard.Doc.Dr Coşkun Atay, Sinem Seçgin
- Algorithms and Complexity,-Bozzano G Lusia
- Yöneylem Araştırması- Hamdy Abdelaziz Taha