Graf Nedir

Graf G düğümler olarak adlandırılan V = { v_1,v_2,v_3, ...} ve kenarlar olarak adlandırılan

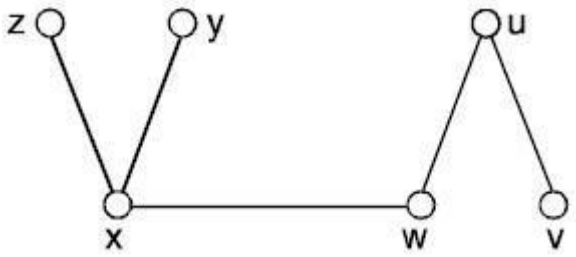
E= { e_1,e_2,e_3, ...} nesnelerin oluşumudur.

V(G) düğümler kümesi olarak isimlendirilirken E(G) kenarlara kümesi olarak isimlendirilir. Genellikle Graf G=(V,E) olarak adlandırılır.

Bir Grafdaki G ise şu kümelerle tanımlanır :

V(G) = {u, v, w, x, y, z} ve E(G) = {uv, uw, wx, xy, xz} olmak üzere,

Şimdi ise bu hesaplama sonucu aşağıdaki Graf ı elde ederiz.

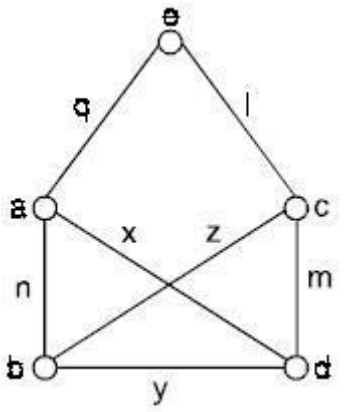


Şekil 1

Her bir Graf da bir diyagram ilişkili bulunmaktadır.Tepe noktası p,kenarları q olan bir Graf a(p,q) şeklinde tanımlanır.

Aşağıdaki şekilde tepe noktaları a ve b ler bitişik iken a ve c noktaları bitişik değil. Bunun yanında x ve y noktaları bitişik iken x ve z bitişik değil.

x ve z köşe noktalarının diyagramda kesişmesine rağmen onların kesişmeleri Graf da tepe noktasında değillerdir.



Şekil 2

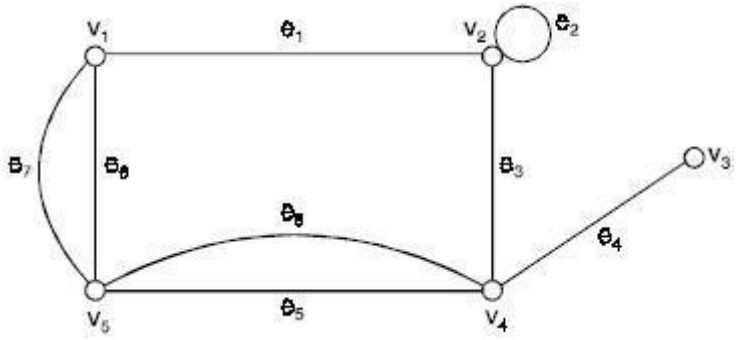
Örnekler:

(1) V = {1, 2, 3, 4} ve E = {{1, 2}, {1, 3}, {3, 2}. {4, 4}}

O zaman G(V,E) bir Graf tır.

(2) V = {1, 2, 3, 4} ve E = {{1, 5}, {2, 3}}

O zaman G(V, E) 5 in V de olmamasından dolayı bir Graf değildir.



Şekil 3

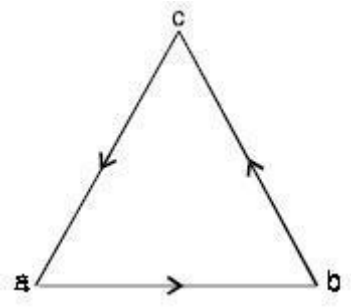
5-tepe noktası ve 8-köşelere sahip bir Graf (5,8) Grafdır.

Yönlü ve Yönsüz Graflar

Yönlü Graflar: Eğer Graf da her bir kenar noktasının bir yönü var ise o zaman bu Graflar yönlü Graf olarak adlandırılır.

Yönlü Graf diyagramlarında her bir kenar e=(u,v) bir ok ile temsil edilmektedir.

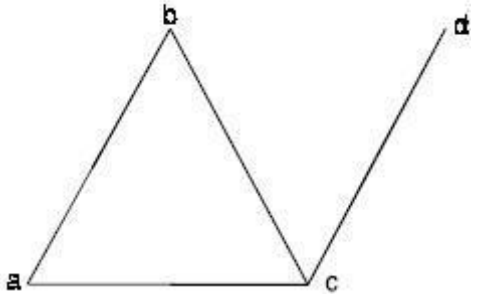
Şekil 1(a) bir yönlü Grafa örnek olarak gösterilebilir.



Şekil 4

Yönsüz Graflar : Eğer Graf G üzerinde her bir kenar noktasının yönü yok ise o zaman bu tür Graflar yönsüz Graf olarak adlandırılır.

Şekil 1(b) yönsüz Graf lara örnek olarak gösterilebilir.



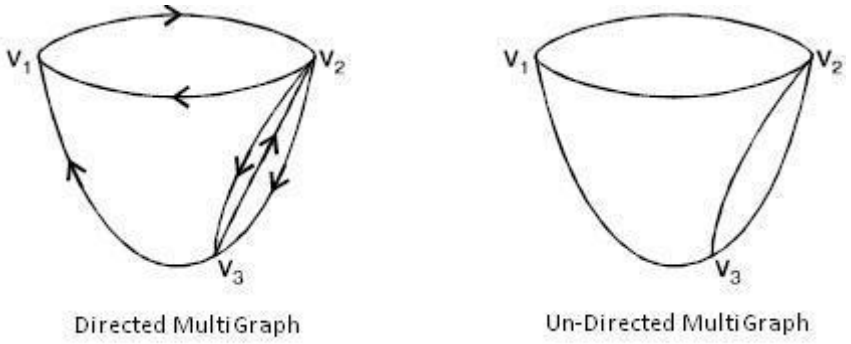
Şekil 5

Temel Terminoloji

Loop : Kenar noktası düğüme tekrar katılan Graf lara loop (döngü) veya self loop denir.

MultiGraf : Çoklu Graf larda döngüler kullanılamaz fakat birden fazla kenar 2 tepe noktasına katılabilir ve bu kenalar Multiple Edges (çoklu kenar) veya paralel kenar olarak adlandırılır ve Graf ise Multigraf (çoklu Graf) olarak adlandırılır.

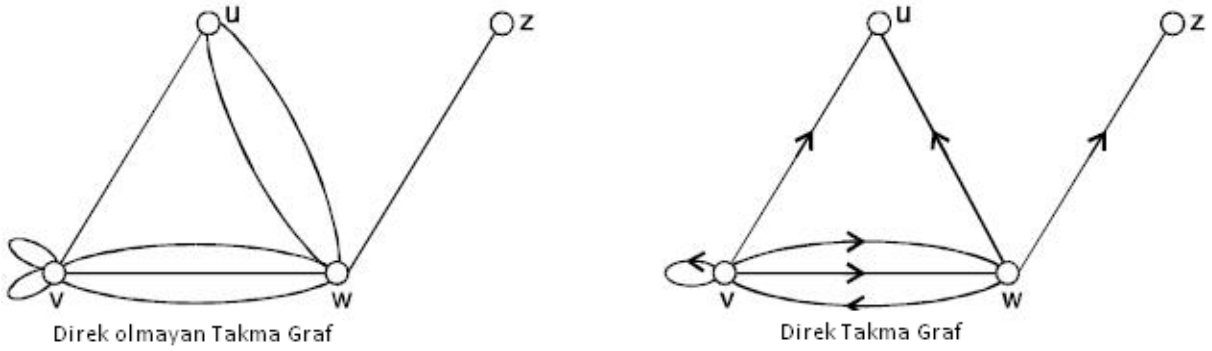
İki Kenar (v_i, v_j)ve (*vf , vr*)paralel kenarlardır eğer *vi = vr* ve *vj, vf*.



Şekil 6

Geçici (Pseudo) Graf :

Çoklu kenarların ve döngülerin kullanılabileceği Graflar geçici Graf olarak adlandırılır.



Şekil 7

Basit (Simple) Graf :

Döngü veya çoklu köşesi olmayan Graflar basit (Simple) Graf olarak adlandırılır.

El Sıkışma Teoremi (Hand-Shaking Theory)

Eğer G= (v,E) e köşeleriyle bir yönsüz Graf ise o zaman;

$$\sum_{v \in V} \deg_G (v) = 2e$$

Sonuç : Yönsüz olmayan bir Graf da, tek sayılı tepe noktası derecelerinin toplamı çift sayıdır.

Kanıt : G = (V,E) bir yönsüz Graf olsun. U çift sayıların düğüm nokta derecesini göstersin ve W ise tek sayılı düğüm noktaların derecesini göstersin.

$$\text{O zaman} \sum_{v_i \in V} \deg_G (v_i) = \sum_{v_i \in U} \deg_G (v_i) + \sum_{v_i \in W} \deg_G (v_i)$$

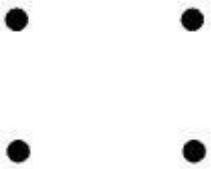
$$\Rightarrow \quad 2e - \sum_{v_i \in U} \deg_G (v_1) = \sum_{v_i \in W} \deg_G (v_1)$$

$$\text{Şimdi} \sum_{v_i \in W} \deg_G (v_i) \text{ çifttir}$$

Graf Türleri

Bazı önemli Graf türleri aşağıda listelenmiştir.

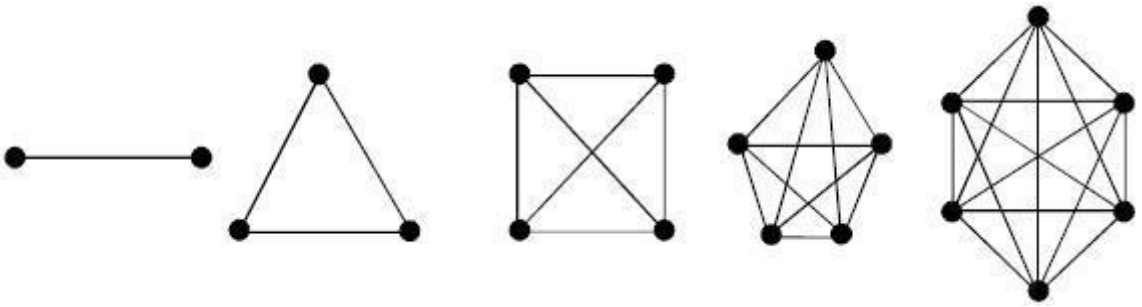
Null Graf : Bütün noktaları birbirinden izole edilmiş Graf türüne denir.



Şekil 8

Tam (Complete) Graf : Her bir köşe noktasının diğer her bir nokta ile birleşmesinden oluşan Graf türüdür.Tam graflar genellikle **K*n*** ile gösterilir.

n=1,2,3,4,5,6 Graf ları aşağıdaki şekilde gösterilmiştir.

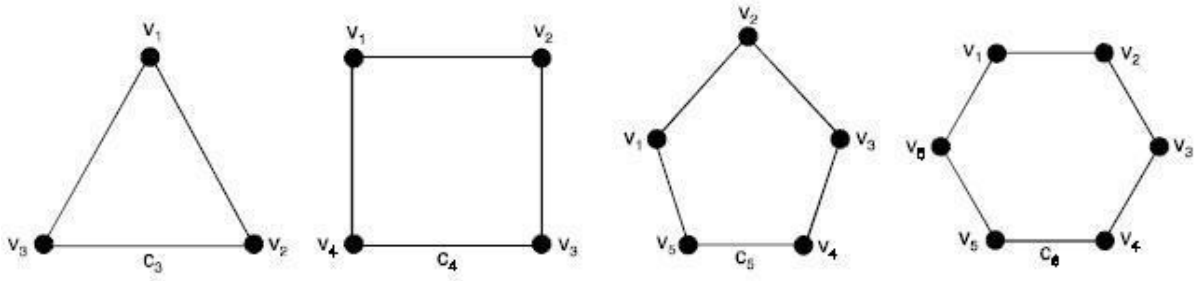


Şekil 9

Düzenli (Regular) Graf : Bütün noktaların eşit dereceli olduğu Graf türüne düzenli Graf denir.

Eğer her bir nokta r ise o zaman derecesi r olan düzenli Graf olarak adlandırılır.

*c*3, *c*4, *c*5 ve *c*6 çemberleri aşağıdaki şekilde gösterilmiştir.

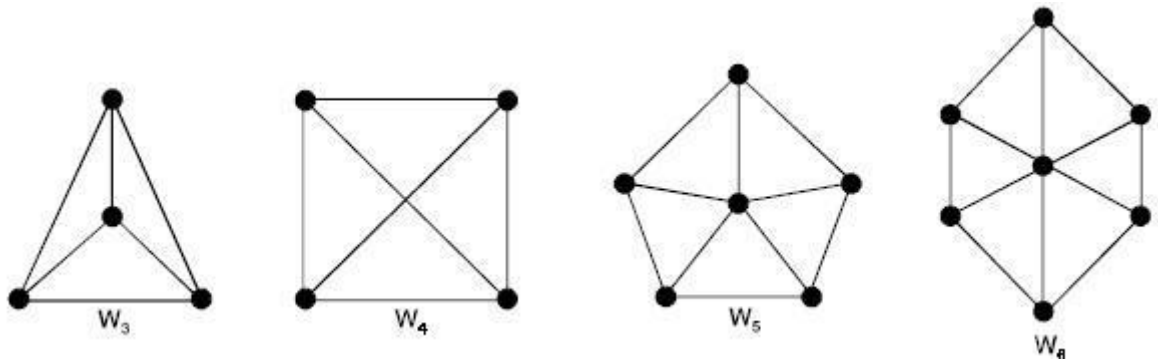


Şekil 10

Dönme (Wheels)

Cycle Cn Grafına ek bir dügüm eklenerek oluşturulur.Eklenen yeni düğüm, diğer bütün düğümlere bağlıdır.

W3, W4, W5 ve W6 tekerlekleri aşağıdaki şekilde gösterilmiştir.

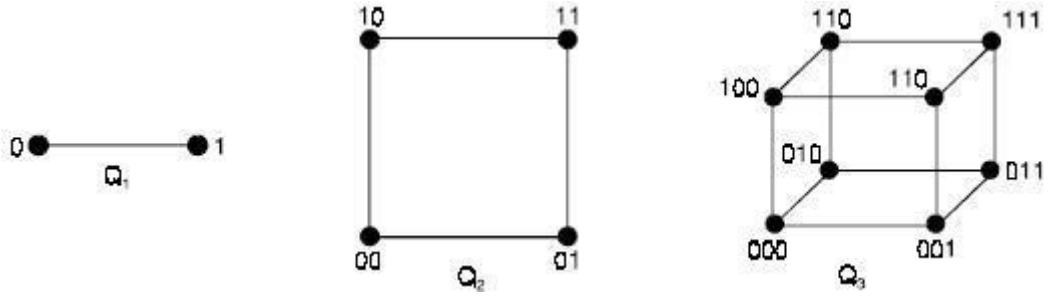


Şekil 11

N-cube (Küp) Graf

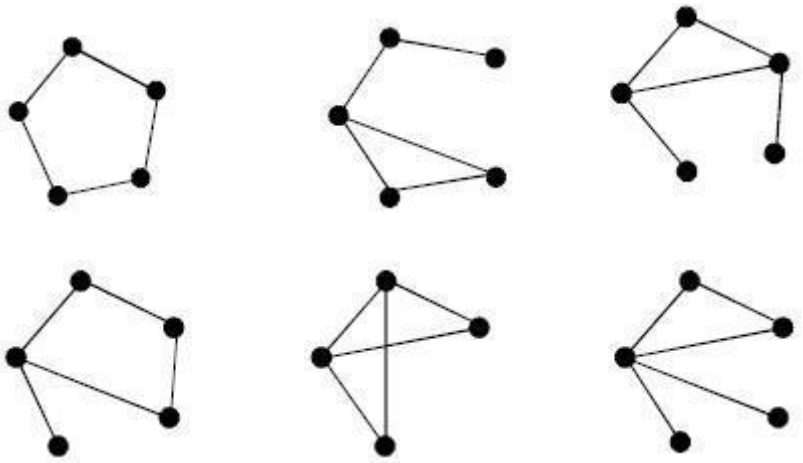
N-cube Qn : Graf ın düğüm noktaları n uzunluğunda 2n bit dizi ile gösterilir. Düğümlerin dizi değeri, bir düğümden diğerine geçerken ayni anda sadece bir bitin değerini değiştirmektedir.

Q1, Q2, Q3 Graf ları aşağıdaki şekilde gösterilmiştir.



Şekil 12

6 adet beş kenar ve aynı sayıda düğüm noktası olan Graf.



Şekil 13

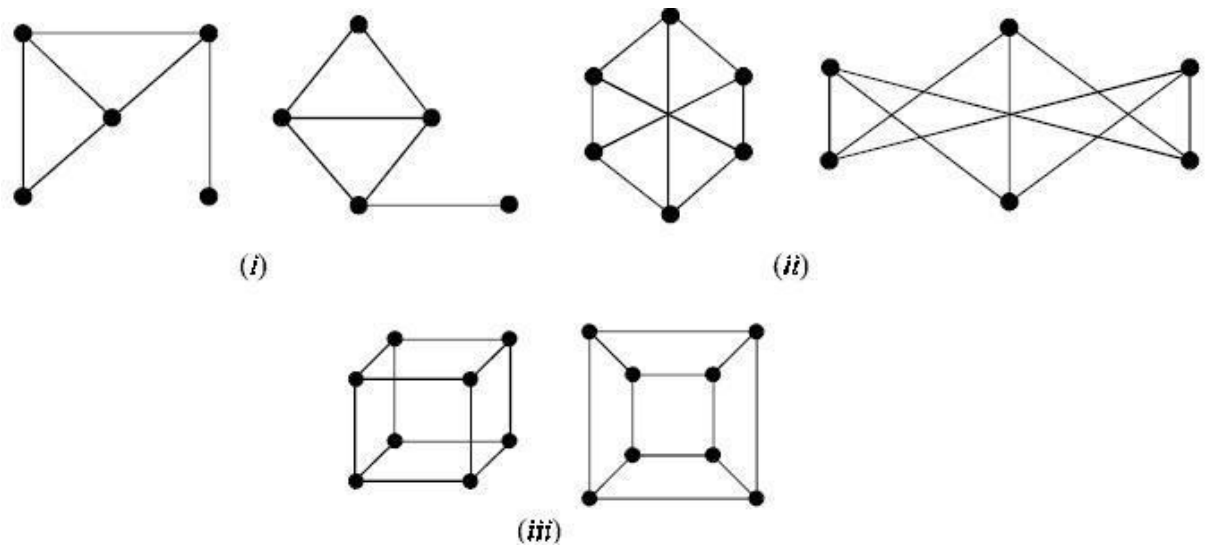
İzomorfik (Isomorphic) Graflar

İki Graf ın izomorfik olup olmadığı nasıl kontrol edilir ?

V,E boş olmayan graf olsun.Eğer V,E Graf için,

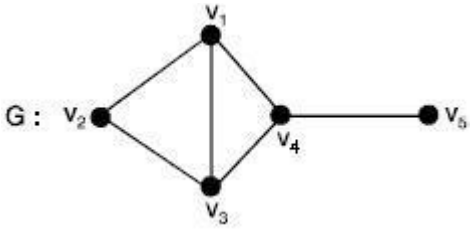
- Kenar sayıları eşitse,
- Düğüm sayıları eşitse,
- Düğüm dereceleri eşitse,
- Düğümler arasındaki ilişkiyi gösteren matrisler eşitse,

Bu matrislerdeki benzerlik satir ve sütunlardaki yer değişikliği ile de sağlanabilir.



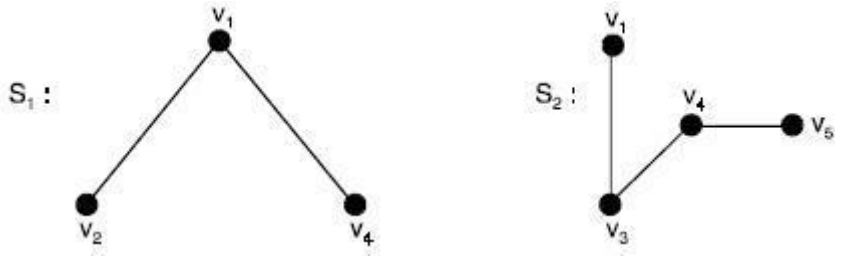
Şekil 14

Problem : Aşağıdaki Graf dan 2 ayrılmış kenar noktası ve 2 ayrılmış düğüm noktası oluşturun



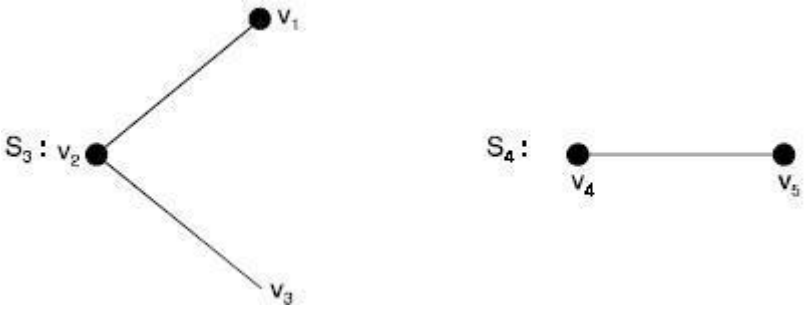
Şekil 15

Çözüm :



Şekil 16

S1 ve S2 Graf ları kenar-parçalanmış G nin Alt Graflarıdır.



Şekil 17

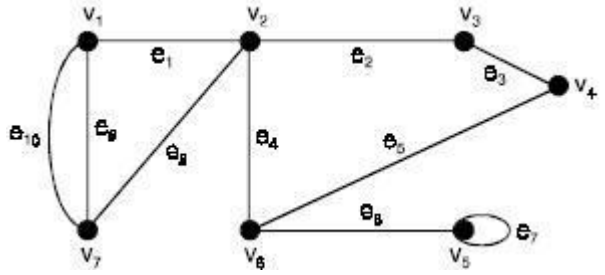
S3 and S4 G den düğüm notları ayrılmış ve aynı zamanda alt Graf da kenarları ayrılmıştır.

WALK, PATHS

Walk (Yürüyüş): Notka ve köşelerden oluşan sınır alternatif dizilere denir.

viej, *vi* + 1 *ej* + 1, *vi* + 2, ..., *ekvm*

Örneğin, x şeklinde Graf da gösterildiği gibi *v*2*e*4*v*6*e*5*v*4*e*3*v*3 ve *v*1*e*8*v*2*e*4*v*6*e*6*v*5*e*7*v*5 yürüyüşlerdir.



Şekil 18

Yol(Path) : Yürüyüşte tepe noktaları birden faazla görüşebilir. Açık bir walk (yürüyüşte) birden fazla tepe notkası görünmeyen şekillere basit yol (simple path)ve ya yol diyoruz.

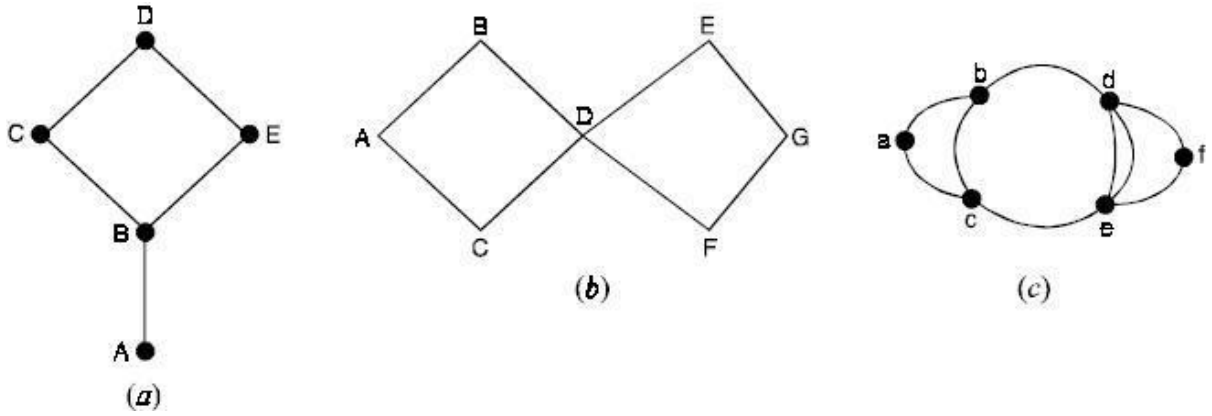
Örnek olarak yukarıdaki şekilde görünen Graf ta;

*v*6*e*5*v*4*e*3*v*3*e*2*v*2 bir yol iken *v5e7v5e6v6* bir açık yürüyüştür bir yol değildir.

Euler Graf

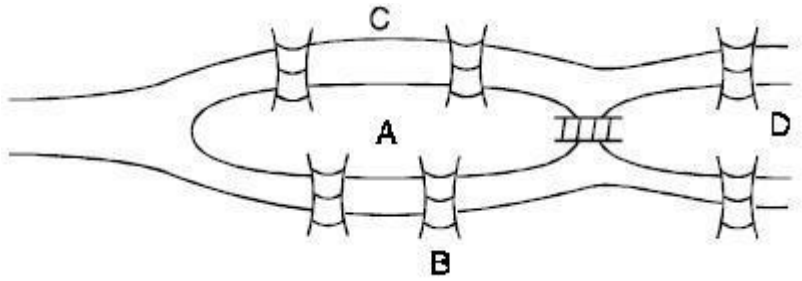
Euler yolu : Eğer her bir kenarı tam olarak bir kere kapsıyorsa bu yol Euler yolu olarak adlandırılır.

Euler devre : Euler yolu halka şeklinde olanlara Euler devre denir.



Şekil 19

KÖNIGSBERG'NİN KÖPRÜ PROBLEMİ



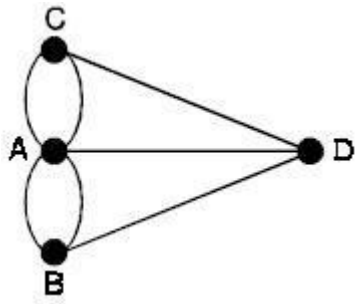
Şekil 20

Problemin Kökeni : Königsberg kentinde Eski ve Yeni Pregel nehirleri birleşerek Pregel)nehrini oluşturmaktadır. Bu nehirler şehri 4 bölüme ayırmaktadır ve nehir üzerinde bu bölgeleri birleştiren yedi köprü bulunmaktadır. *Bütün köprülerden bir ve yalnız bir kez geçmek koşulu ile bir yürüyüş yapılabilir mi?*

1736'da Euler'in incelemeleri böyle bir gezintinin mümkün olmadığını kanıtlamış ve bu tür dolaşmayı mümkün kılacak çizgelerin şu özelliklere sahip olmaları gerektiğini göstermiştir: Birleşik bir çizgenin bütün elemanlarını bir ve yalnız bir kez kullanarak dolaşmak için o çizgenin tek dereceli düğümlerinin sayısı, eğer varsa, iki olmalıdır. Tek dereceli düğümler dolaşmanın başlangıç ve bitiş düğümleridir. Çizgede böyle düğümler yoksa dolaşmaya herhangi bir düğümden başlanabilir.

Çözümün temelinde yatan düşünce şudur: Bir düğüm, başlangıç ya da bitiş düğümü değilse o düğüme gelen kişinin turu tamamlayabilmek için oradan ayrılması gerekecektir. Dolayısıyla bu tip düğümler çift dereceleri olmalıdır. Oysa tek dereceli bir düğüme, örneğin *D* düğümüne ikinci kez gelen bir kişi çıkış yolu bulamayacaktır. Dolayısıyla bu düğüm ya gezintinin bitiş düğümü olmalıdır ya da başlangıç düğümü olarak seçilmelidir ki ikinci gelişte çıkış yolu bulunabilsin. Buna göre tek dereceli düğüm sayısı ikiden fazlaysa gezinti tamamlanamayacaktır.

Yürüyüşün sonunda başlangıç noktasına dönülebilmesi içinse bütün düğümler çift dereceli olmalıdır. Böylece, başlangıç ve bitiş düğümü aynı olan ve her bir elemanı sedece ve en az bir kez içeren turlara "Euler turu" ve Euler turu içeren çizgelere de "Euler çizgesi" denmiştir.



Şekil 21

Düşlemsel Graflar

G Genel bir Graf da G = (V, E, Ψ)olarak tanımlanabilir ve V yi a,b,c,d ve e olarak adlandırılan 5 nesneden oluşturduğumuzu varsayarken ki bu

V = {*a*, *b*, *c*, *d*, *e*}

anlamına gelir ve E yi V den farklı 7 nesneden oluşturalım

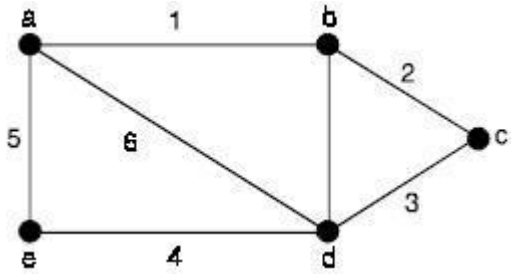
E={1,2,3,4,5,6,7}

Ve bu iki küme arasındaki ilişki Ψ 'i map leyerek çıkıyor.

$$\Psi = \begin{bmatrix} 1 \longrightarrow (a, c) \\ 2 \longrightarrow (c, d) \\ 3 \longrightarrow (a, d) \\ 4 \longrightarrow (a, b) \\ 5 \longrightarrow (b, d) \\ 6 \longrightarrow (d, e) \\ 7 \longrightarrow (b, e) \end{bmatrix}$$

-> Graf ın tümleşik bileşimi

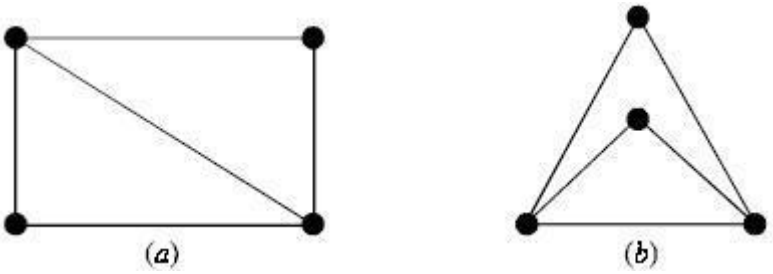
Aşağıdaki şekilde görüldüğü gibi bir geometrik şekle dönüştürülebilir



Şekil 22

PLANAR GRAFLAR

Graf G de eğer bir veya birden fazla geometrik şekil var ise bu Graf türlerine Planar Graf denilir. Kesişme noktalarına ise kesişim denir.



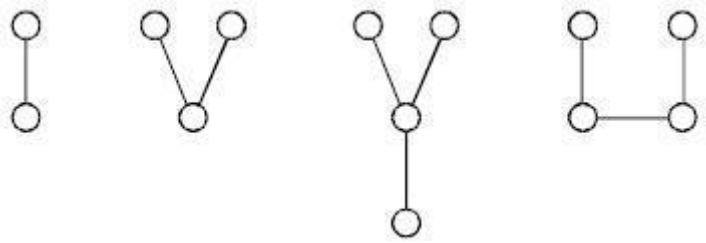
Şekil 23

TREE (AĞAÇ)

Çevrimsiz Graf : Dönümleri olmayan Graflar çevrimsiz Graf olarak adlandırılır.

Ağag(Tree) : Bağlı çevrimsiz Graflar

Orman(Forest) : Herhangi çevrimi olmayan Graf orman (forest) olarak adlandırılır



Şekil 24

Notlar :

- Her bir ağacın kenar ı bir köprüdür
- G nin her bir bloğu çevrimsizdir.
- Graf G ye bağlı her bir kenar bir köprüdür.
- Herhangi bir döngünün kendisinin bir çevrim olduğu unutulmamalıdır.

DALLANAN AĞAÇLAR

Graf teorisinde ağaçların öneminin bir nedeni, her bağlantılı grafın özel bir ağaç içermesidir. Bu tip ağaçlar geren ağaç (spanning tree) olarak adlandırılır.

T, düğüm kümesi V ve ayrıt kümesi E olan bir ağaç olsun. Bu durumda

- Her u ve w gibi iki ayrık düğüm çifti için T içerisinde bir birlerini bağlayan tek bir yol vardır.
- T'deki herhangi bir ayrıtın silinmesi her birisi ağaç olan iki bileşenli bir graf oluşturur.
- |E| = |V| - 1

İkili(Binary) , Köklü (Rooted)Ağaçlar

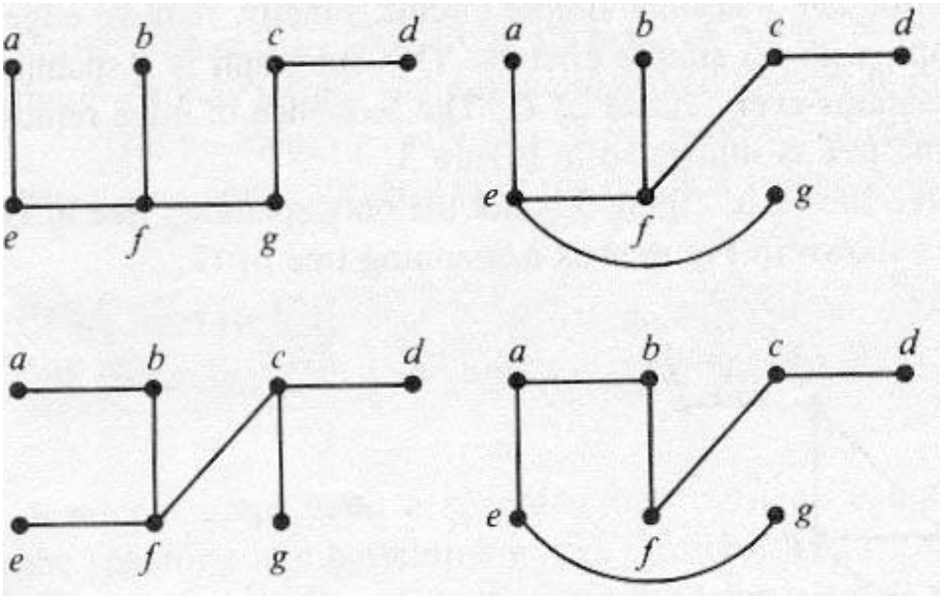
- Bir ağaçta tam bir tane düğümün derecesi 2 ve diğerlerinin dereceleri 1 veya 3 ise tam ikili ağaç (full binary tree) olarak adlandırılır.
- Derecesi 2 olan düğüm kök (root) olarak,
- Derecesi 3 olan düğümleri karar düğümleri (decesion vertices),
- Derecesi 1 olan düğümlerde yaprak düğümler (leaf vertices) olarak adlandırılır.

Kullanım Alanı

İkili ağaçlar, genellikle bilgisayar bilimlerinde sıralama ve arama algoritmalarının tasarlanmasında kullanılmaktadır

DALLANAN AGACLAR KURMAK İÇİN GEREKLİ ALGORİTMALAR

Teorem 1'in kanıtı, basit dolanımlardan kenarlar çıkarmak suretiyle dallanan ağaçlar bulmak için bir algoritma verir. Bu algoritma, basit dolanımları teşhis etmek gerektiğinden yetersizdir. Kenarları kaldırarak dallanan ağaçlar kurmak yerine, kenarlar ekleyerek dallanan ağaçlar elde edebiliriz. Burada bu prensibe dayanan iki algoritma tanıtılacaktır.



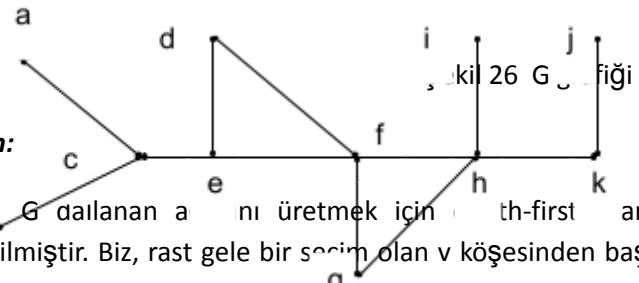
Şekil 25

Bağlantılı basit bir grafik için depth-first search kullanarak bir dallanan ağaç inşa edebiliriz. Köklü bir ağaç kuracağız ve dallanan ağaç, bu köklü ağacın aşağıya uzanan direkt olmayan grafiği olacak. Grafiğin bir isteğe göre seçilmiş bir köşesini kök olarak seçin. Bu köşeden başlayan birbirini izleyen kenarlar ekleyen bir yol inşa edin. Her bir yeni kenar, yolda bulunan son köşele henüz yolda bulunmayan köşe arasında olmalıdır. Bu yola mümkün olduğu sürece kenar ekleyin. Eğer yol grafiğin bütün köşelerinden geçiyorsa, bu yolu içeren ağaç, dallanan ağaçtır. Bununla birlikte, eğer yol, tüm köşelerden geçmiyorsa, daha fazla kenar eklenmelidir. Yol üzerindeki son köşeden geri gidin ve eğer mümkünse, bu köşeden başlayan ve ziyaret edilmemiş köşelerden geçen yeni bir yol çizin. Eğer bu yapılamıyorsa, yol üzerindeki başka bir köşeye geri gidin, bu da yoldaki iki köşe geri olan köşedir. Yeniden deneyin. Bu yöntemi, ziyaret edilen son köşeden başlayarak, yoldan her seferinde bir geri köşeye giderek, artık daha fazla kenar eklenemeyecek hale gelene kadar olabildiğince uzun bir yol inşa edene kadar devam edin. Grafik sınırlı sayıda kenara sahip olduğundan ve bağlantılı olduğundan bu işlem bir dallanan ağacın elde edilmesiyle son bulacaktır. Algoritmanın bir aşamasında bir yolu bitiren her bir köşe, köklü ağacın yaprakları olacaktır. Ve kurulan yolun başlangıcı olan her bir köşe , bir dahili köşe olacaktır. Bu yöntemin tekrarlamalı özelliğine dikkat edin. Aynı zamanda, eğer grafikteki köşeler sıralı ise, yöntemin her aşamadaki kenar seçimlerinin hepsinin belirli olduğuna dikkat edin. Yine de bir grafiğin köşelerini her zaman belirli bir şekilde sıralamayacağız.

Depth-first search aynı zamanda (**backtracking) aynı yere geri dönme** olarak da adlandırılır. Çünkü algoritma, yol eklemek için önceden ziyaret edilmiş köşelere döner. Aşağıdaki örnek aynı yere geri dönmeye örnek teşkil edecektir.

ÖRNEK

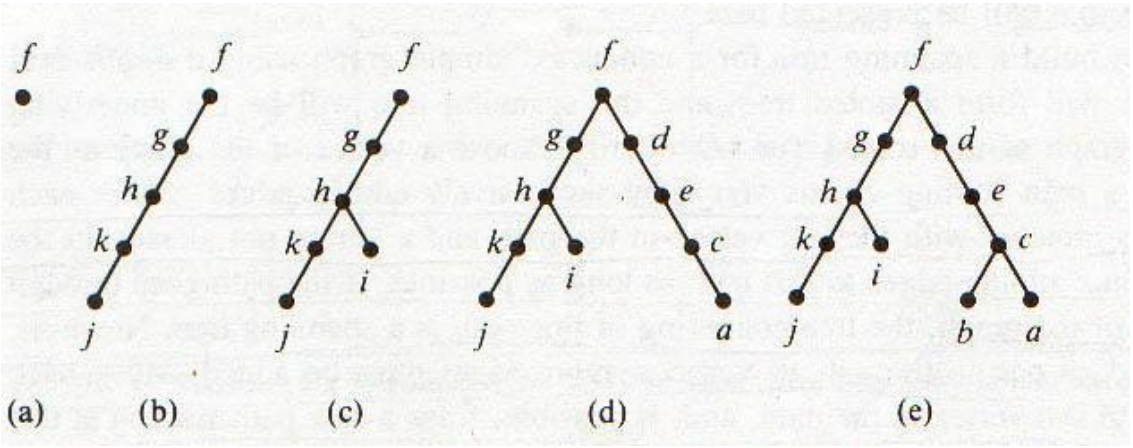
Şekil-5'te gösterilen G grafiği için; Depth-first search algoritmasını bir dallanan ağaç bulmak için kullanın.



Şekil 26 G grafiği

Çözüm:

G dallanan ağacını üretmek için depth-first search içinde kullanılan adımlar şekil-6'da gösterilmiştir. Biz, rast gele bir seçim olan v köşesinden başladık. Bir yol, ardışık olarak yol üzerinde henüz bulunmayan köşelere gidilerek mümkün oldukça yeni kenarlar ekleyerek kurulur. Bu, f,g,h,k,j yolunu üretir. (Başka yolların da kurulabileceğine dikkat ediniz.) daha sonra, k'ya geri dönüş yapın. k'dan başlayan ve ziyaret edilmemiş köşe içeren bir yol bulunmamaktadır. O halde, h'ye geri dönüş yapın. h,i yolunu kurun ve önce h'ye sonra f'ye geri dönüş yapın. f'den; f,d,e,c,a yolunu kurun. Sonra, c'ye dönüş yaparak c,b yolunu kurun. Bu, dallanan ağacı üretecektir.



Şekil 27

Bir basit grafiğin dallanan ağacını breadth-first searchi kullanarak da üretebiliriz. Yine, köklü bir ağaç kurun, bu köklü ağacın aşağıya uzanan direkt olmayan grafiği dallanan ağacı oluşturur. Grafiğin köşelerinden rast gele bir kök seçin. Sonra tüm kenarları bu köşeye bağlayın. Bu aşamada eklenen yeni köşeler, dallanan ağacın seviye 1'deki köşeleri olurlar. Bunları rast gele sıralayın. Daha sonra, seviye 1'deki sıralı olarak ziyaret edilen her bir köşe için, bu köşeye bağlanan kenarlar ekleyin. Ekleme işini, basit dolanım üretmeden yapmalısınız. Seviye 1'deki her bir köşesi rast gele sıralayın. Bu, ağaçta, seviye 2'de bulunan köşeler üretir. Aynı işlemi, ağaçtaki her köşe eklenmiş olana kadar

tekrarlayın. Grafiğin her bir köşesini içeren bir ağaç ürettiğimizde, bu yöntem sonucunda dallanan ağaç üretilmiş olur.

AYNI YERE GERİ DÖNME (BACKTRACKING)

Sadece tüm olası çözümlerin ayrıntılı araştırmasıyla çözümlenebilecek problemler vardır. Bir çözüme sistematik olarak ulaşmak için olan yollardan biri, bir karar ağacı kullanmaktır. Bu karar ağacındaki her bir dahili köşe bir kararı, ve her bir yaprak da olası bir sonucu gösterir. geri dönmeden geçen bir çözüm bulmak için, öncelikle olabildiğince uzun bir çözüme ulaşabilmeyi denemek için ardışık kararlar vermek gerekmektedir. Ardışık kararlar, ağaçta, bir yol ile gösterilebilir. Hiçbir çözüm verilen ardışık kararlardan başka bir kaynaktan sonuçlanamaz. Bunu bir kez bildikten sonra, eğer mümkünse, şu andaki köşesin ebeveynine geri dönüş yapıp, sonuca başka karar dizilerinden ulaşmak gerekir. Yöntem, ya bir sonuç bulunana kadar devam eder, ya da problemin herhangi bir sonucu olmadığına karar verilinir. Aşağıdaki örnek, geri dönme olayının kullanılışlığına örnek temsil etmektedir.

ÖRNEK

Geri dönme, bir grafiğin n adet renk kullanarak renklendirilip renklendirilemeyeceğine karar verirken nasıl kullanılabilir?

Çözüm:

Bu problemi, geri dönmeyi aşağıdaki yoldaki gibi kullanarak çözebiliriz. Öncelikle, bir a köşesi seçin ve, onu renk 1 olarak atayın. Sonra, ikinci bir b köşesi seçin. Eğer b, a'ya komşu değilse renk 1 olarak atayın; komşuysa b'yi renk 2 olarak atayın. Üçüncü c köşesinden devam edin. eğer mümkünse, c için renk 1'i kullanın, değilse renk 2'yi. Eğer ne renk 1, ne de renk 2 kullanılamıyorsa, c'ye renk 3'ü atayın. Bu yönteme, her bir ek köşe için n adet renklerden birini atayarak ve her zaman listenin ilk uygun, olası rengini kullanarak mümkün olduğunca devam edin. Eğer verilen n renkten birini kullanamayacak bir köşeye erişilirse, mümkünse, bir önceki atamaya geri dönüş yapıp son renklendirilmiş köşesin rengini bir sonraki olası rengi kullanarak değiştirin. Eğer bu mümkün değilse, bu renklendirmeyi

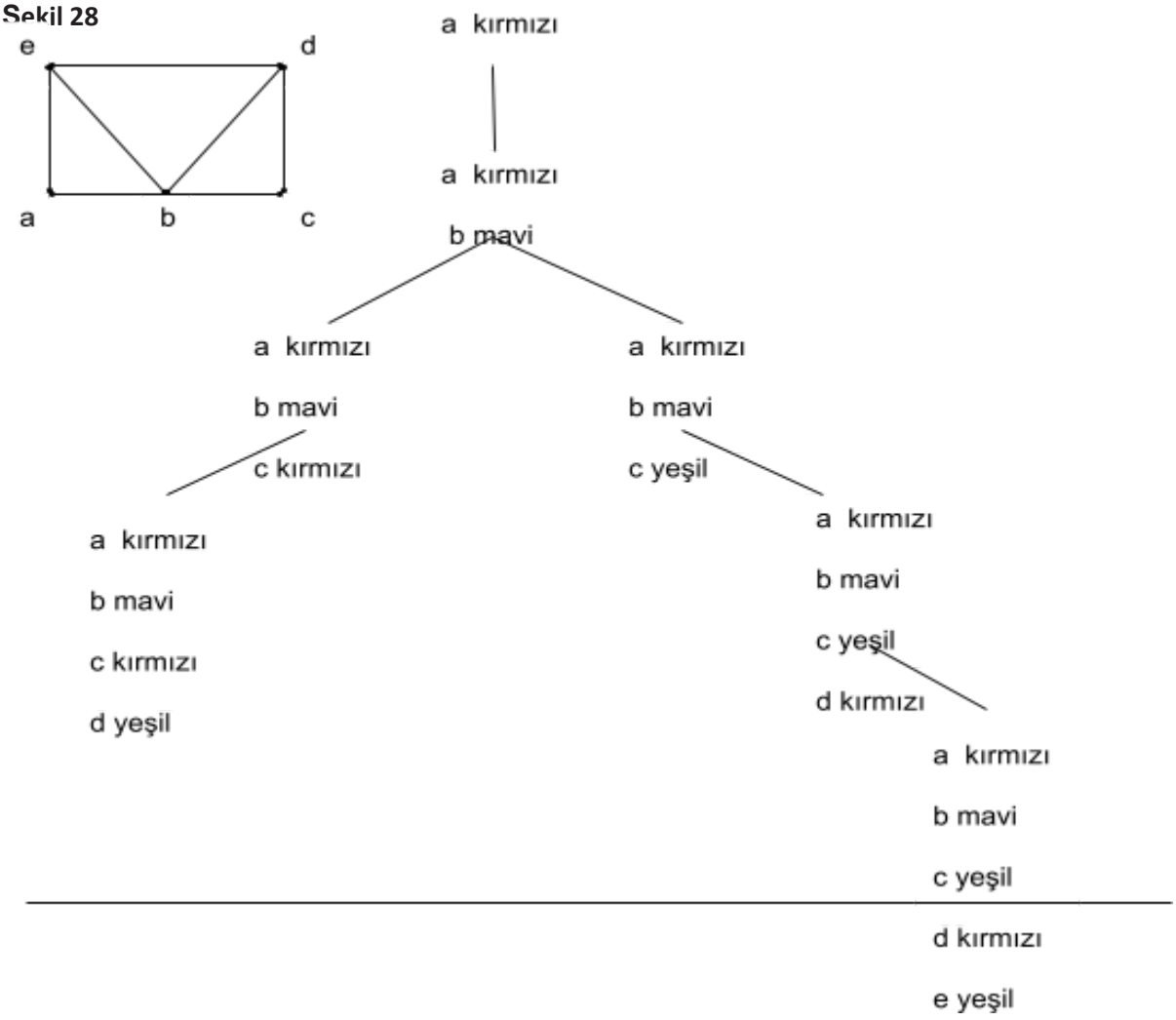
değiştirip, daha önceki atamalara geri dönüş yapın. bir köşesin rengini değiştirmek mümkün olana kadar her seferinde bir adım geriye gidin. Sonra, köşelerin renk atamalarına mümkün olduğunca devam edin. eğer n adet rengi kullanarak renklendirme mümkünse, geri dönme bunu üretecektir. (Ancak ne yazık ki, bu yöntem son derece etkisizdir.)

Şekil-9'da gösterilen grafiğin üç adet renkle renklendirilmesi problemini düşünün. şekil-9'da gösterilen ağaç, 3'lü renklendirmeyi kurarken geri dönmeyi nasıl kullanacağımızı göstermektedir. Bu yöntemde, önce kırmızı, sonra mavi ve son olarak da yeşil kullanılmıştır. Bu basit örnek açıkça görülebileceği gibi geri dönme kullanmadan da yapılabilir, ancak teknik olarak iyi bir örnektir.

Bu ağaçta, kırmızının a'ya atanmasını gösteren ve kökten başlayan ilk yol, a'nın kırmızıyla, b'nin maviyle c'nin kırmızıyla ve d'nin yeşille renklendirilmesine yol açar. a,b,c, ve d bu yolla renklendirildiğinde, e'yi bu üç renkten biriyle renklendirmek mümkün değildir. d için başka bir renk

kullanılamayacağı için, bir fazla adıma geri dönünüz. Daha sonra, c'nin rengini yeşille değiştiriniz. Biz, e'ye yeşil ve d'ye kırmızı vererek grafiği renklendirebildik.

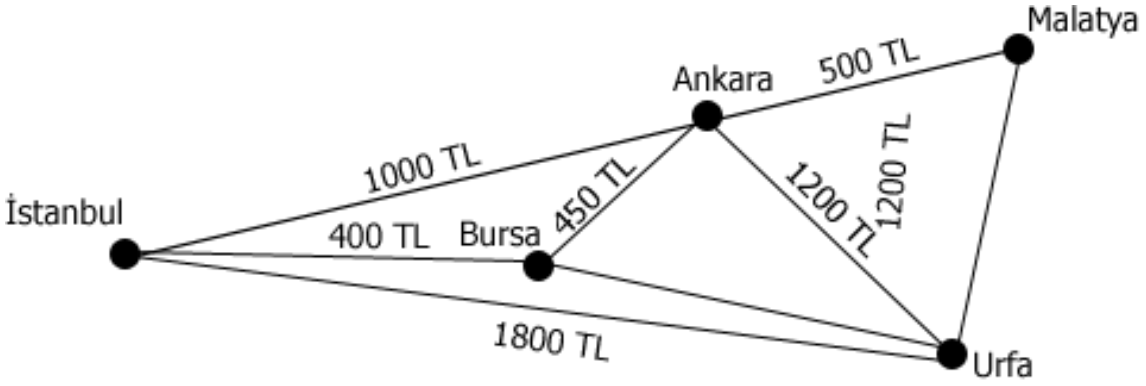
Şekil 28



MINIMAL DALLANAN AĞACLAR

Bir merkez ve 4 farklı lokasyonda bulunan şubeler birbirleri arasında bir network oluşturmaya karar vermiştir. .Bu merkezlerden herhangi ikisi kiralanmış bir telefon hattıyla birbirlerine bağlanabilmektedirler.Hangi bağlantılar yapılmalıdır ki herhangi iki şube arasında bir yolun oluşturulması garantilensin ve böylece toplam maliyet en az olsun?

Bu problemi Şekil 2 de gösterilen ağırlık grafiği kullanarak modelleyebiliriz.



Şekil 29 Bir network deki lease line (kiralık hat) bedellerini gösteren graf

MİNİMAL DALLANAN AĞAÇLARI İÇİN ALGORİTMALAR

Prim Algoritması : 1957'de Robert Prim tarafından verilmiştir. Prim'in algoritmasını devam ettirmek için, en küçük ağırlığa sahip kenarı seçip, dallanan ağacına koyarak işe başlayın. Devamında, ağaçtaki mevcut köşeye bağlı olan minimum ağırlıktaki kenarlarla, ağaçtaki mevcut kenarlarla basit dolanım oluşturmayacak şekilde ekleyin. n-1 sayıda kenar ekleninceye kadar devam edin.

Prosedür Prim *(G: ağırlık bağlantılı, n köşeli direkt olmayan grafik)*

T: = minimum ağırlıklı kenar
aralık i:=1'den n-2'ye kadar

başlangıç

e:= T'de bir köşeye bağlı ve T'ye eklendiğinde T'de bir basit dolanım oluşturmayan minimum ağırlıklı bir kenar

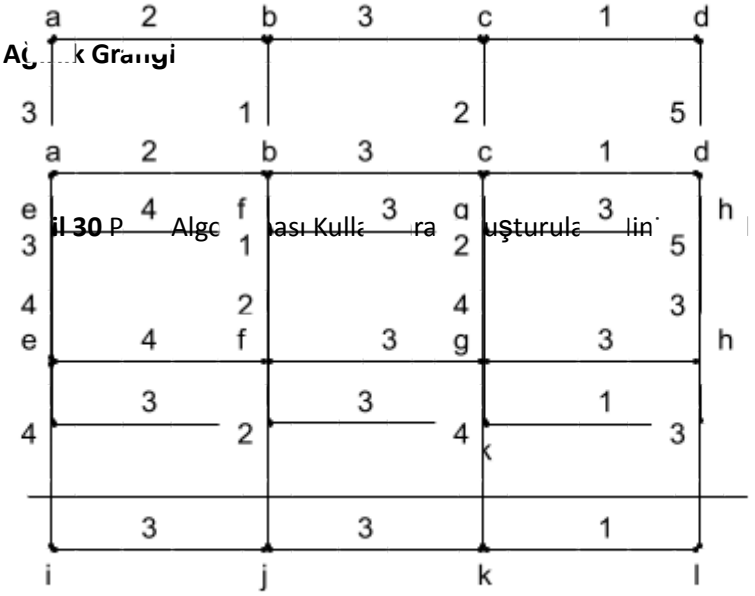
T:= e eklenmiş T
Bitiş
{T, G'nin minimal dallanan ağacıdır}

Prim Algoritması Örnek:

Prim algoritmasını kullanarak Şekil 3'teki gösterilen grafik için minimal dallanan ağacını oluşturunuz.

Çözüm : Prim algoritması kullanılarak oluşturulan minimal dallanan ağacı Şekil 4'te gösterilmiştir. Sırayla seçilen kenarlar gösterilmiştir.

İnceleyeceğimiz ikinci algoritma Joseph Kruskal tarafından 1956'da keşfedilmiştir. Kruskal'ın algoritmasını devam ettirmek için grafikteki minimum ağırlığa sahip kenarı seçin



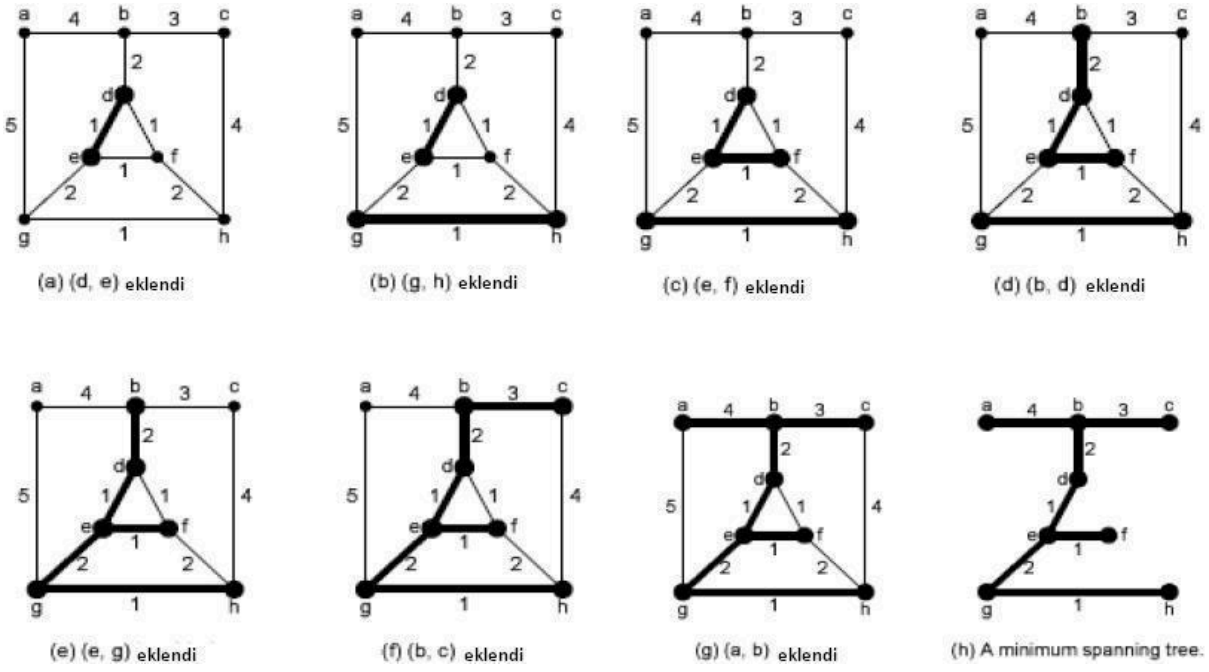
Ağırlık Grafiği

Şekil 30 Prim Algoritması Kullanılarak Oluşturulan

Seçenek	Kenar	Ağırlık
{b,f}	1	
{a,b}	2	
{f,j}	2	
{a,e}	3	
{i,j}	3	
{f,g}	3	
{c,g}	2	
{c,d}	1	
{g,h}	3	
{h,l}	3	
{k,l}	(b)	
Total:24		

Kruskal Algoritması

- Grafın her bir düğümünün başlangıçta ayrık ağaç olduğu bir orman oluşturulur.
- Daha sonra ayrıtlar ağırlıklarına göre sıralanılır.
- Sıralanmış her bir (u,v) kenarları için; eğer u ve v iki farklı ağacın düğümleri ise iki ağaç tek bir ağaca birleştirilerek (u,v) ormana dahil edilir.
- Bu işlem tüm ayrıtlar işleninceye kadar devam eder.



Şekil 31

1. ALGORİTMA Kuskal'ın Algoritması
Prosedür *Kruskal* (*G*: ağırlık bağlantılı, n köşeli direkt olmayan grafik)

T: = boş grafik

aralık *i*:=1'den n-1'ye kadar

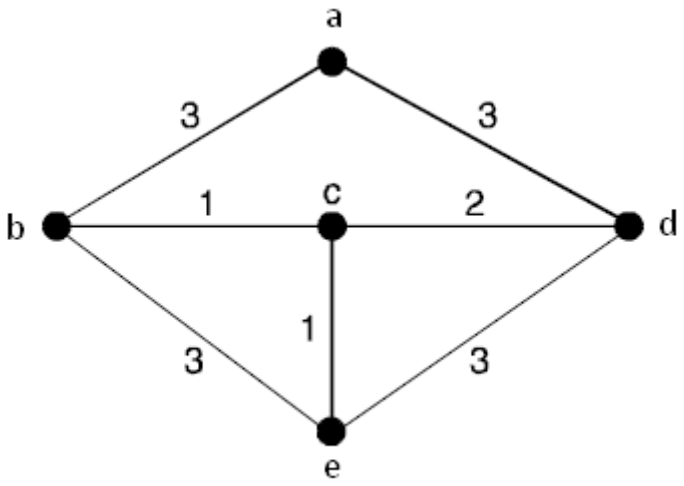
başlangıç

e:= G'de, *T*'ye eklendiğinde bir basit dolanım oluşturmayan minimum ağırlıklı herhangi bir kenar

T:= *e* eklenmiş *T*

bitiş {*T*, *G*'nin minimal dallanan ağacıdır}

Örnek : Şekil 4.1 de Kruskal algoritmasının nasıl Minimal dallanan ağacı bulduğunu gösterin



Şekil 32

Çözüm : Ağırlıkları ile birlikte köşelerini bir tablo içerisine topluyoruz.

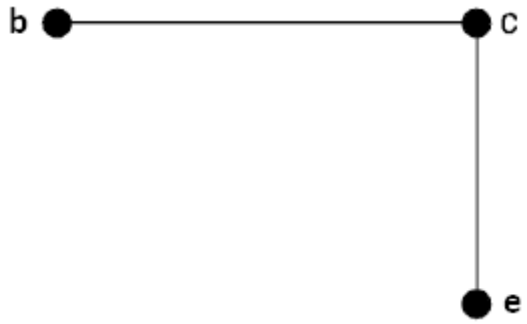
Kenar	Ağırlık
(b, c)	1
(c, e)	1
(c, d)	2
(a, b)	3
(e, d)	3
(a, d)	4
(b, e)	4

Bu aşamadan sonra izlememiz gereken adımlar aşağıdaki gibidir :

1. En düşük ağırlığa sahip kenar noktasını seçin (b,c)

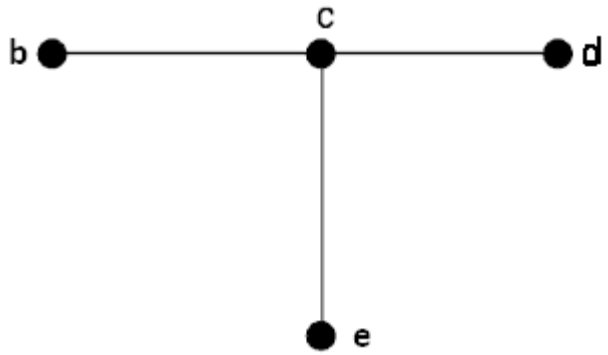
Şekil 33

2. Bir sonraki kenar noktasını ekle**(c,e)**



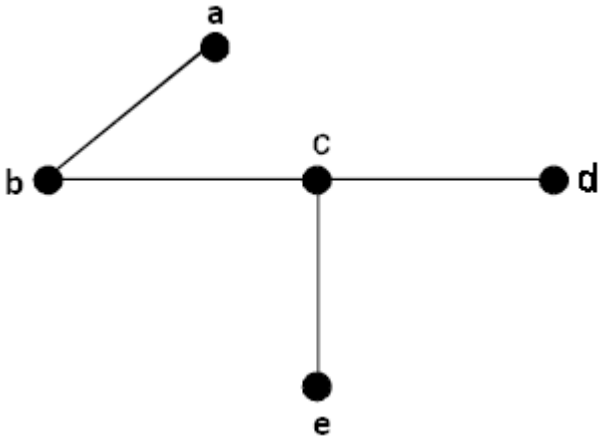
Şekil 34

3. Kenar noktasını ekle**(c,d)**



Şekil 35

4. Kenar noktasını ekle **(b,a)**



Şekil 36

5 düğüm noktası ve 4 kenar oluştuğunda algoritmayı durduruyoruz ve minimal dallanan ağaçlar üretilmiş oluyor.

Routing (Yönlendirme)

Routing Protokol: Amacı iyi bir yol belirliyerek (router serileri) kaynaktan gelen datayı hedef router a ulaştırmaktır.

Routing algoritmaları için graf soyutlandırılması:

- Graf düğümleri routerlardır
- Graf köşe noktaları fiziksel bağlantılardır
    - Link Ücreti: geçikme, fiyat,tıkanıklık seviyesi

Routing (Yönlendirme)Algoritmaları

Routing algoritmalarının vazgeçilmez temel özellikleri vardır. Bunlar:

- Doğruluk
- Basitlik
- Sağlamlık
- İstikrar
- Mükemmellik

Routing Algoritmalarının Sınıflandırılması

Global ve Merkezi olmak üzere Routing Algoritmaları iki sınıfa ayrılır.

Küresel: Tüm yönlendiriciler tam topolojiye sahip, bağlantı ücretleri vardır.

Merkezi: Yönlendiriciler fiziksel olarak bağlı olduğu komşuları ve bağlantı maliyetlerini bilir. Mesafe vektor algoritması.

Bu Algoritmalar statik ve dinamik olmak üzere 2 ye ayrılır.

Statik Algoritmalar : Statik olarak update edilen bu algoritmaların converge süresi oldukça yavaştır.

Dinamik Algoritmalar: Tabloların daha hızlı update olması için tasarlanan bu algoritma türleri statik algoritmalara göre çok daha fazla tercih edilmektedir.

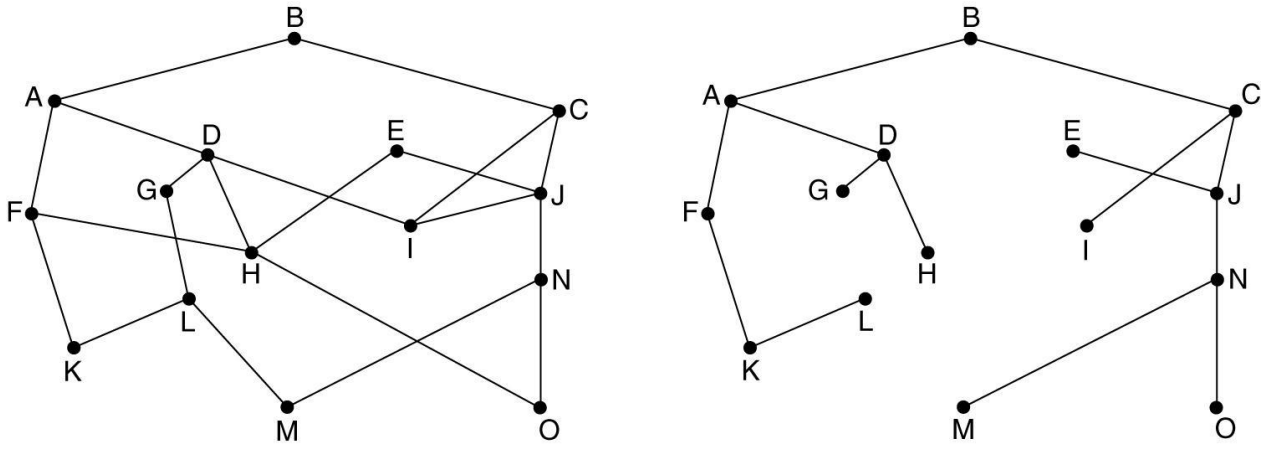
Algoritmalar:

- Optimallik Prensibi (The Optimality Principle)
- En Kısa Yol Yönlendirilmesi (Shortest Path)
- Taşma(Flooding)
- Uzaklık vektör yönlendirme (Distance Vector Routing)
- Bağlantı Durum Yönlendirme (Link State Routing)
- Hiyerarşik yönlendirme (Hierarchical Routing)
- Yayın Yönlendirme (Broadcast Routing)
- Çoklu Yönlendirme (Multicast Routing)

Optimallik prensibi

F->A->B en iyi yol =>A->B *en iyi yol*

Optimal yönlendirme tüm kaynaklarını bir ağaç kökü gibi en uygun yola yönlendirir.



(a) Subnet **(b)** B Routerı için dallanan ağaç

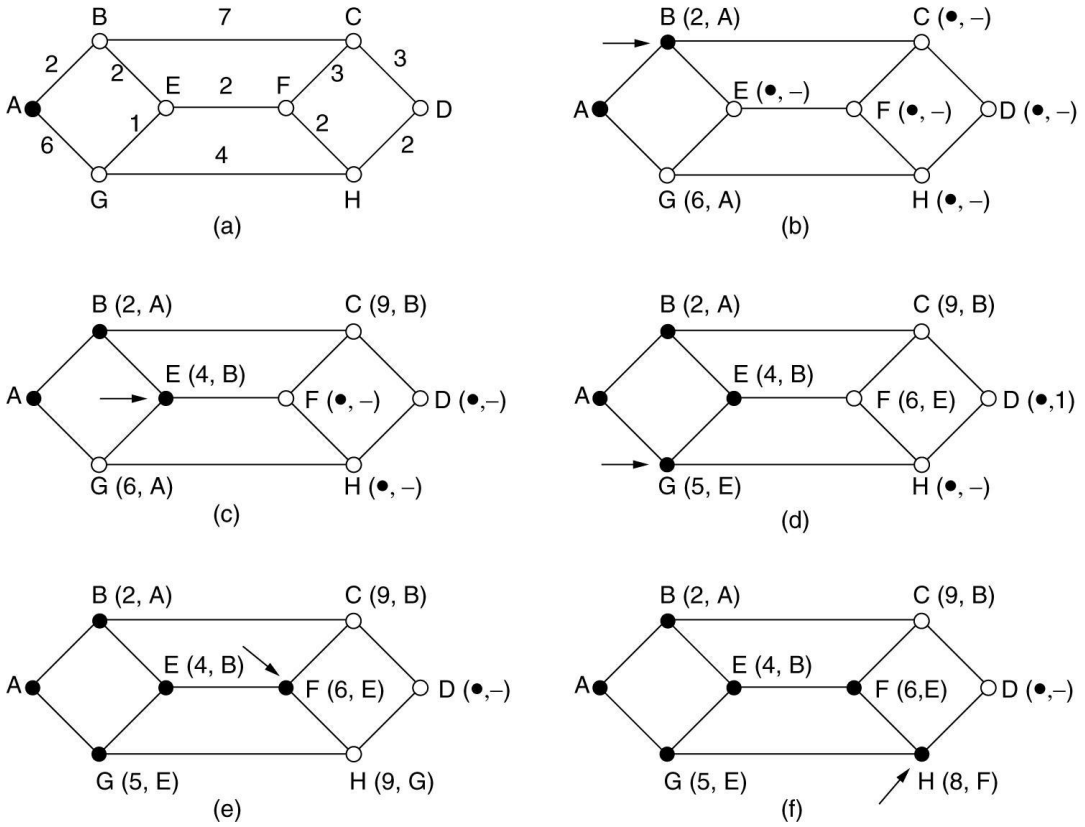
Şekil 37

En kısa yol yönlendirme

- Mesafe ölçümü
- Hops sayısı
- Coğrafi mesafe
- Kuyruk ve ileti gecikme ortalaması
- Maliyet
- Bant genişliği

En kısa yol yönlendirmesi (Dijkstra)

A dan D ye 5 adımda bilgisayar en kısa yolu kullanabilir. Oklar çalışma düğümünü gösterir.



Şekil 38

Taşma (static)

- Kendisine gelen yol dışında gelen paketi mevcut bütün yollardan gönder
- Her bir zıplamada hop sayacı bir azalacak
    - Hop sayacı sıfır olduğunda paketi yoksay

Taşmanın (flooding) dezavantajları

Network uygulamalarında bu yöntemi kullanmamın birden çok dezavantajı vardır.En temel olarak bandwitdh kullanımda ciddi bir şişme oluşturur.Mesajın sadece tek bir hedefi olabilirken bu mesaj kendisine gelen düğüm noktası dışında bağlı olduğu bütün bağlantılara bu veriyi gönderir.

Taşmanın(flooding) Avantajları

Mesajın yoluna devam edebilmesi için her bir bağlantıyı gönderdiğinden mesajın hedefine gitme olasılığı çok yüksektir.

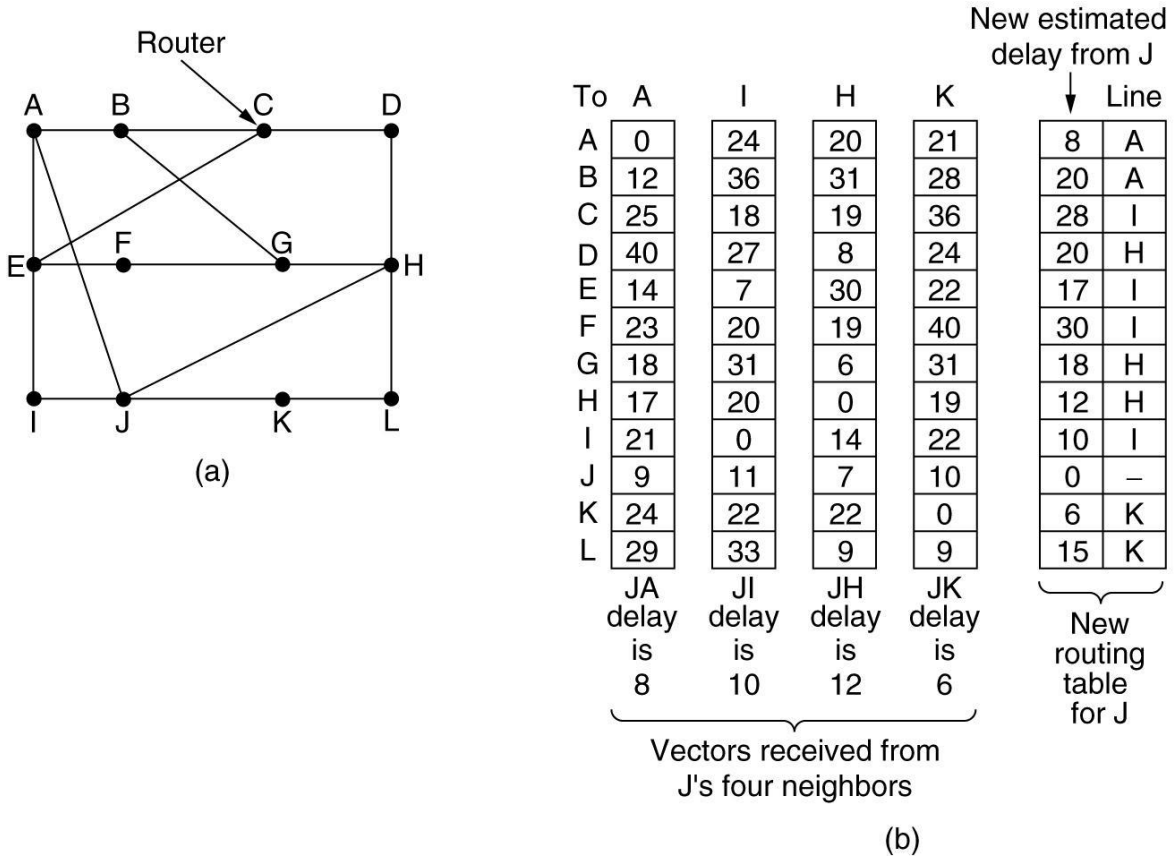
Uzaklık vektör yönlendirme (Distance Vector Routing)

Başlama

$$D(1) = 0$$
$$(n(v) = ?, D(v) = \infty) \text{ for } v = 2,..., N$$

Genel Adım

$$n(v) = \arg\min_{w}(D(w) + l(w, v))$$
$$D(v) = \min_{w}(D(w) + l(w, v))$$



Şekil 39

(a) subnet. **(b)** giriş A, I, H, K, ve yeni J yönlendirme tablosu

Link State yönlendirme

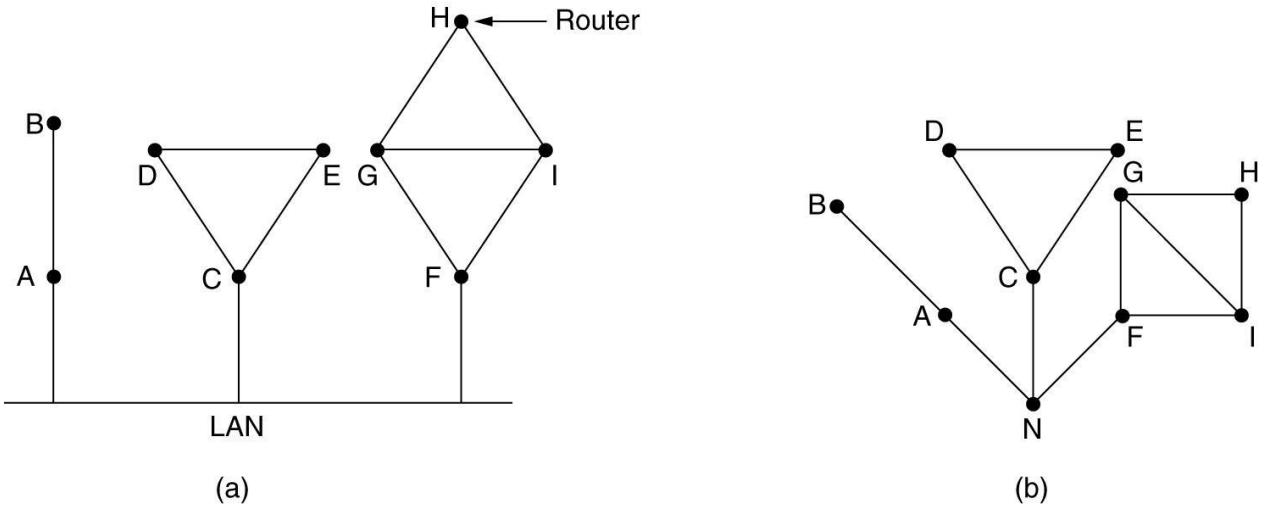
Dijkstra's algoritmasını uygulamak için bir yöntem(dağıtılmış)

Her yönlendirici için aşağıdakileri yapmak gerekir:

1. Komşularını anlamak, kendi ağ adresini öğrenmek
2. Gecikme tedbiri veya her bir komşunun maliyeti
3. Bir paket kurmak(düzenlemek) tümünü yeniden öğrendiğini anlatır.
4. Tüm diğer yönlendiriciler bu paketi gönderir.
5. Her yönlendirici en kısa yolu hesaplar.

Komşuları hakkında öğrenme

Açılıştan sonra kendisi üzerinden dışarıya giden her bir bağlantı üzerinden bir *HELLO* paketi gönderir ve bu paketlere karşılık alarak çevresindeki cihazlardan haberi olur.



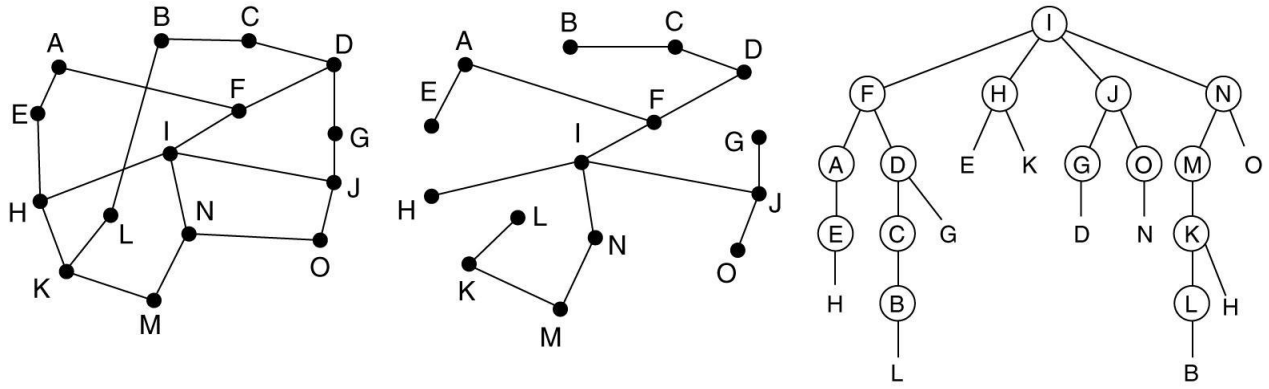
(a) Dokuz yönlendirici ve bir LAN **(b)** Graf modeli

Şekil 40

Broadcast Yönlendirme

- Broadcast: bütün yollara aynı anda bir paket gönder
- Her hedef için ayrı bir paket gönder
- Sel : her düğüm dışarıya giden tüm hatlara paketleri kopyalar
- Multidestination yönlendirme: kopya olarak giden en az bir hedef için en iyi yol hangisiyse , her paket dahil bu hattı kullanır (hiyerarşik olarak bölümlenmiş hedef koymak)
- Sink tree/spanning tree: gelen bir paket dışındaki tüm satırları kapsayan ağaca kopyalayın.
- Ters yol yönlendirme: eğer gelen hat kaynağa en iyi hedefse tüm giden çizgilere kopyala (bu nedenle paket ilk kopya).

  Son üç yöntem Bandwitdh üzerinde etkilidir.

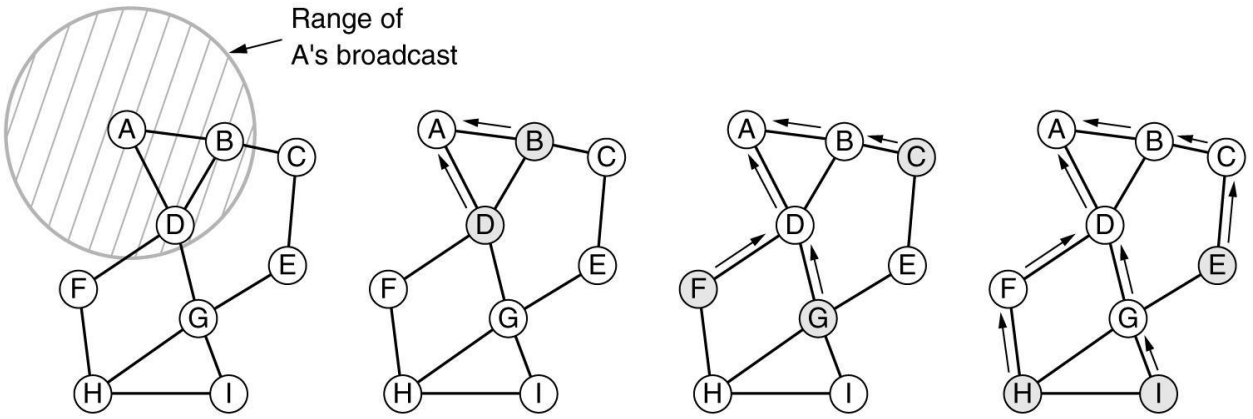


(a) Subnet **(b)** Sink tree. **(c)**Ağaç ters yol

Şekil 41

On Demand Route Discovery (AODV)

Graf düğümler i(router+host).



Şekil 42

(a) A'nın yayın çeşitliliği

(b) A'nın yayınını sonra B ve D almış

(c) A'nın yayınını sonra C,F ve G almış. (B ve D diğer yayınları reddeder)

(d) A'nın yayınından sonra E,H ve I almış.

Değişen düğümler yeni alıcıdır. Oklar olası ters yolları gösterir.

Source address	Request ID	Destination address	Source sequence #	Dest. sequence #	Hop count

ROUTE REQUEST packet processing (broadcast):

- *Eğer* (*Kaynak Adres,İstek ID*) yeni ise, eşlemeyi kaydet
    - *Değilse* paketi yoksay ve dur
- *Eğer* yeni bir yol (Hedef id sine göre) un gidiş yolu bilinirse route yanıtı gönder
    - *Değilse* hop sayacını arttır, route istediğini broadcast olarak gönder.

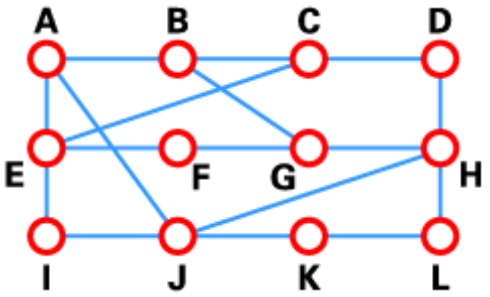
Source address	Destination address	Destination sequence #	Hop count	Lifetime

Yaşam Süresi (Lifetime): Route ne kadar süre boyunca geçerli

Hop Sayacı: Hedef ne kadar uzaklıkta. ROUTE REPLY paket işleme geri dönüş yolundaki her ara düğüme ne kadar uzaklıkta(unicast)

DV Algoritmaları

DV Algoritmaları ayrıca Bellam-Ford ve Ford-Fulkerson routing(yönlendirme) algoritmaları olarakta bilinir.Bu algoritmalarda, her bir router kendisine herhangi bir yönlendirme için kendisine ait bir routing tablosu vardır.



Şekil 43

Ağırlık	Hedef	Çizgi
A	8	A
B	20	A
C	28	I
D	20	H
E	17	I
F	30	I
G	18	H
H	12	H
I	10	I
J	0	---
K	6	K
L	15	K

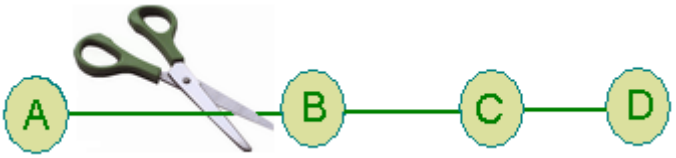
Tabloda gözüktüğü üzere, eğer router J router D ye paket göndermek isteseydi , paketi router H ye göndermesi gerekirdi. Paket Router H ye ulaştığı zaman, kendi routing tablosunu kontrol eder ve bu paketi D ye nasıl göndericeğine karar verir.

DV Algoritmasında, her bir router bu adımları izlemesi gerekmektedir:

1. Direk bağlı olduğu linklerin ağırlıklarını hesaplayıo ve bu bilgileri kendi tablosunda saklar.
2. Belirli bir zaman aralıklarında, kendisine ait olan routing tablosunu komşu router lara gönderir ve diğer router lardan onlara ait routing bilgilerini alır.
3. Komşu router ın routing tablosundaki bilgilere göre kendisininkini günceller

DV Algoritmaların içinde en önemlilerinden biri olan problem : “sonsuzluğa saymak” dır. Bu problemi bir örnek ile inceleyelim.

Aşağıda gösterilen bir network ü düşünelim.Graf ta görebileceğiniz gibi, A ile diğer networkler arası sadece tek bir link var.



Şekil 44

Aşağıda bu düğümlerin graf ve routing tablolarını görebilirsiniz:

	A	B	C	D
A	0,-	1,A	2,B	3,C
B	1,B	0,-	2,C	3,D
C	2,B	1,C	0,-	1,C
D	3,B	2,C	1,D	0,-

Network Graf ve Routing Tabloları

Şimdi düşüninki A ile B arasında link kapandı. Bu durumda, B routing i kendi routing tablosunu düzeltir yani update eder. Belli bir süre zarfı sonrasında routerlar routing tablolarını günceller, ve B router ı C routing table ının bilgilerini alır. C routerı A ile B routerları arasındaki linkde bir sorun olup olmadığını bilmediği için, kendi router tablosunda; A router ına 2 ağırlık ile bir linki mevcuttur.(1 Cden B ye, ve 1 B den A ya—B den A ya bir link olmadığını bilmiyor.)

Bu işlem döngüsü bütün düğümlerin A linkine olan ağırlığı sonsuzluk olduğu anlaşılana kadar devam eder. Bu durum aşağıdaki tabloda gösterilmiştir. Bu durumda, uzmanlar DV algoritmasının **yavaş bir convergence(değişim) süresinin** olduğunu söylüyorlar.

	B	C	D
Sum of weight to A after link cut	∞,A	2,B	3,C
Sum of weight to B after 1st updating	3,C	2,B	3,C
Sum of weight to A after 2nd updating	3,C	4,B	3,C
Sum of weight to A after 3rd updating	5,C	4,B	5,C
Sum of weight to A after 4th updating	5,C	6,B	5,C
Sum of weight to A after 5th updating	7,C	6,B	7,C
Sum of weight to A after nth updating	...	...	...
∞	∞	∞	∞

sonsuzluğa sayma problemi