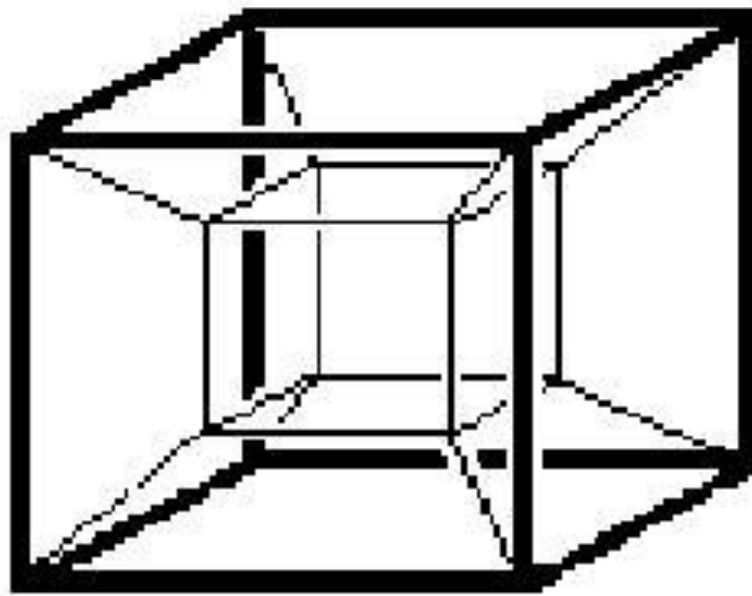


Graf Teorisi (Graph Theory)

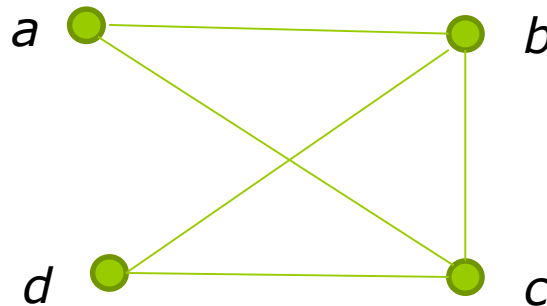


Graf – Çizge

Tanım: Bir $G = (V, E)$ grafi, boş olmayan bir dizi V köşesinden (veya düğümlerden) ve bir dizi E kenarından oluşur. Her kenarın uç noktaları adı verilen, kendisiyle ilişkili bir veya iki köşesi vardır. Bir kenar, graftaki bu köşeleri birbirine bağlayan ve iki köşe arasında bir ilişkiyi gösteren elemandır.

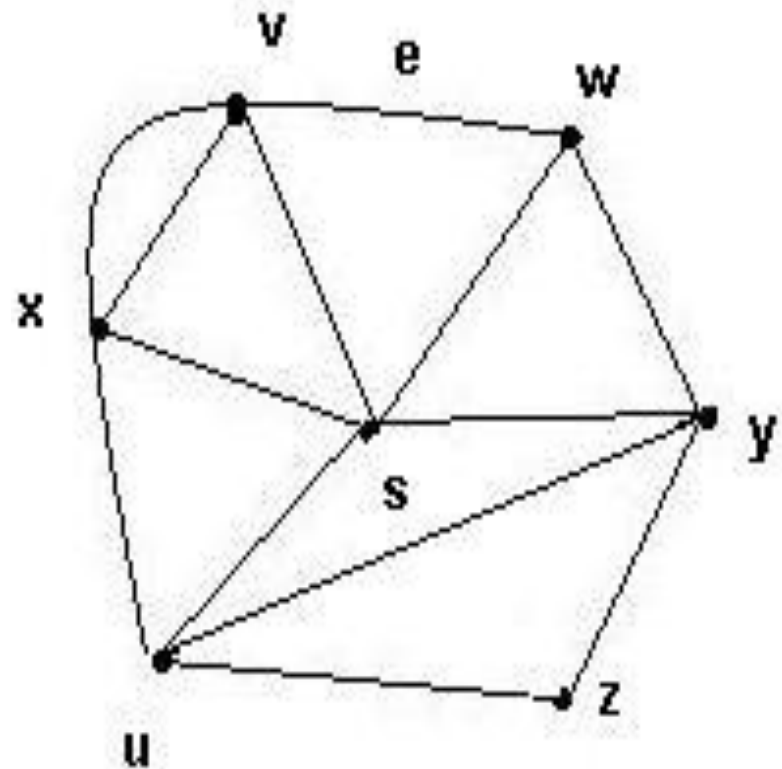
Örnek:

Dört köşeli ve
beş kenarlı bir
graf



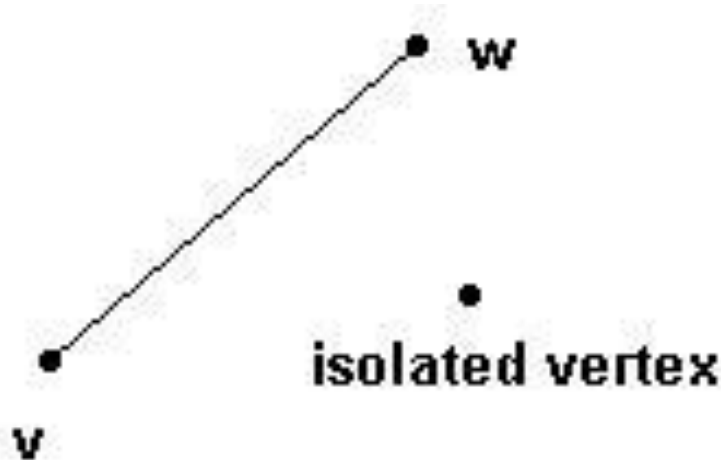
Giriş

- G grafi nedir ?
- $G = (V, E)$
 - $V = V(G) =$ düğümler kümesi
 - $E = E(G) =$ kenarlar kümesi
- Örnek:
 - $V = \{s, u, v, w, x, y, z\}$
 - $E = \{(x,s), (x,v)_1, (x,v)_2, (x,u), (v,w), (s,v), (s,u), (s,w), (s,y), (w,y), (u,y), (u,z), (y,z)\}$



Kenarlar (Edges)

- Kenar bir çift düğüm ile etiketlenmiş olup $e = (v,w)$ şeklinde gösterilir.
- Ayırık düğüm (Isolated vertex) = a kenar bağlantısı olmayan düğümdür.



Özel Kenarlar

▣ Paralel kenarlar(Parallel edges)

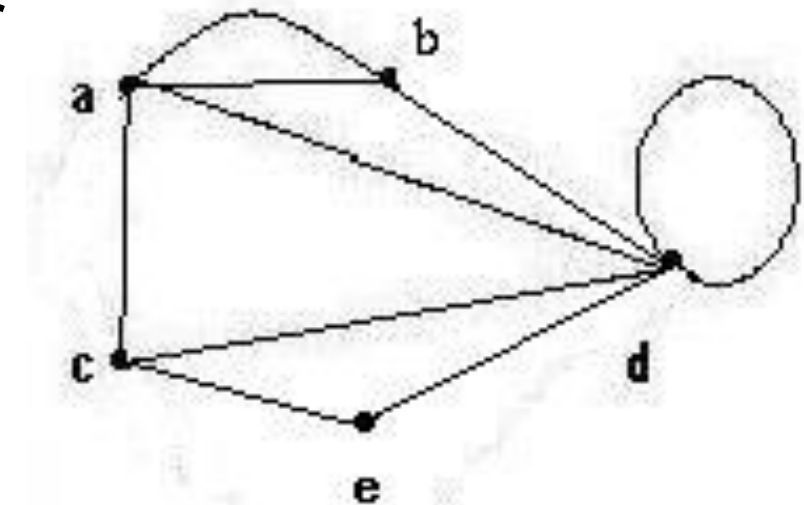
- İki veya daha fazla kenar bir düğüm çifti ile bağlanmıştır.

- ▣ a ve b iki paralel kenar ile birleşmiştir

▣ Döngüler (Loops)

- Kenarın başlangıç ve bitiş noktası aynı düğümdür.

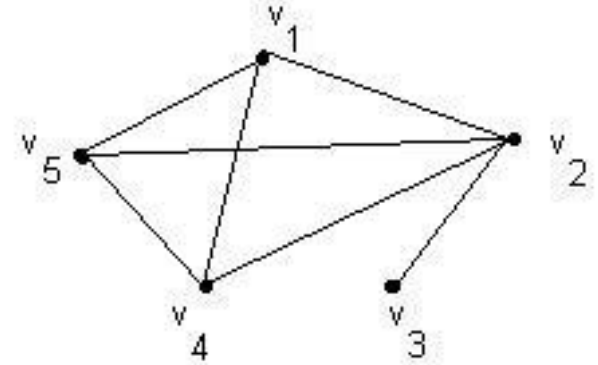
- ▣ d düğümü gibi.



Özel Graflar

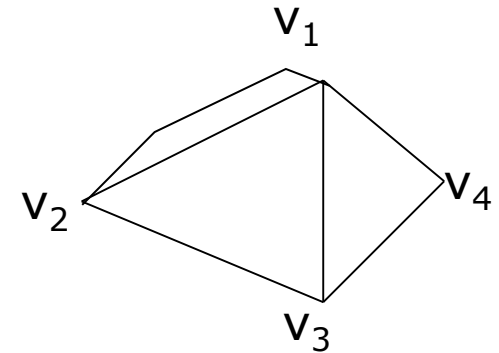
□ Basit (Simple) Graf

- Yönsüz, paralel kenar olmayan ve döngü içermeyen graflardır.



□ Çoklu (Multi) Graf

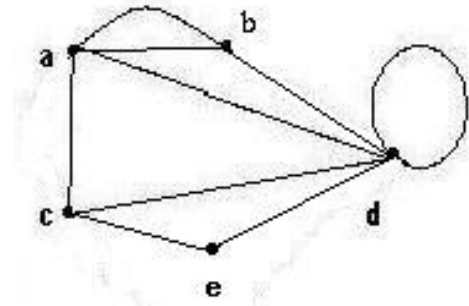
- Basit grafların yeterli olmadığı durumlarda kullanılır.
- Yönsüz, paralel kenarı olan ve döngü içermeyen graflardır.



Basit graflar, çoklu graftır fakat çoklu graflar basit garf değildir.

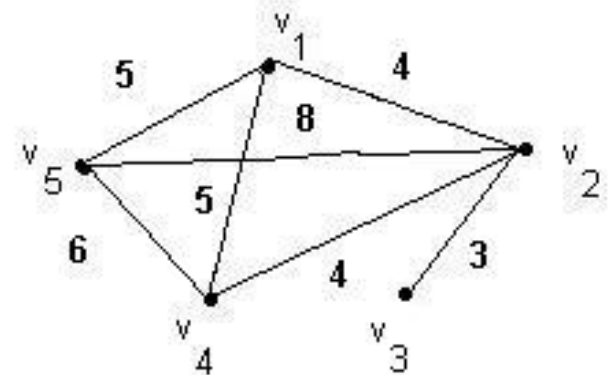
□ Pseudo Graflar

- Çoklu grafların yeterli olmadığı durumlarda kullanılır.
- Yönsüz, Paralel kenarı olan ve döngü içeren graflardır.
- Yönsüz grafların en temel halidir.



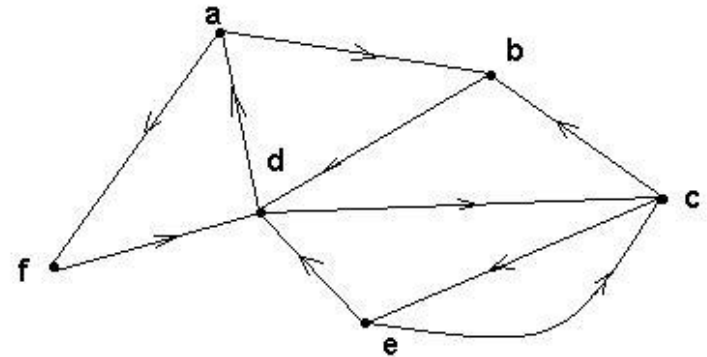
□ Ağırlıklı (Weighted) Graf

Her bir kenarına nümerik bir değer, ağırlık verilmiş bir graftır.



Yönlü (Directed) Graflar (digraphs)

G, yönlü bir graf
(*directed*) veya *digraph*
ise her bir kenarı sıralı
bir düğüm çifti ile
ilişkilendirilmiş ve her
kenarı yönlüdür.



Tip	Kenar	Çoklu Kenara İzin ?	Döngüye İzin ?
Basit Graf	Yönsüz	Hayır	Hayır
Çoklu Graf	Yönsüz	Evet	Hayır
Pseudo Graf	Yönsüz	Evet	Evet
Yönlü Graf	Yönlü	Hayır	Evet
Yönlü Çoklu Graf	Yönlü	Evet	Evet

Graflarda Benzerlik (similarity) (1)

Problem: Nesnelerin değişik özellikleri referans alınarak nesneleri sınıflandırabiliriz.

Örnek:

- Bilgisayar programlarında üç ayrı özelliğin olduğunu kabul edelim. $k = 1, 2, 3$ gibi:
- 1. Programın satır sayısı
- 2. Kullanılan “return” sayısı
- 3. Çağrılan fonksiyon sayısı

Graflarda benzerlik (2)

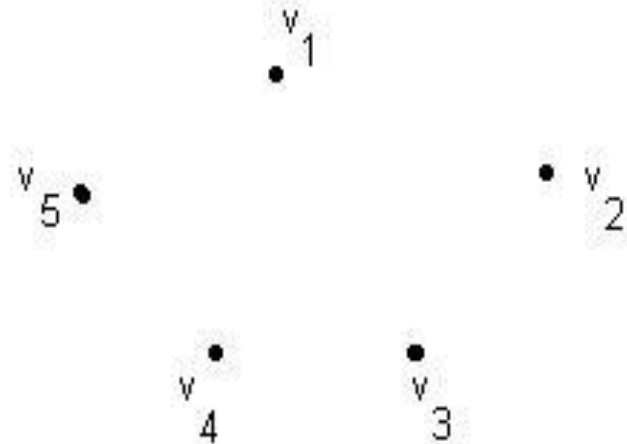
Aşağıdaki tabloda 5 programın birbirleriyle karşılaştırıldığını farzedelim.

Program	# of lines	# of “return”	# of function calls
1	66	20	1
2	41	10	2
3	68	5	8
4	90	34	5
5	75	12	14

Graflarda benzerlik (3)

□ G grafını aşağıdaki gibi oluşturun:

- $V(G)$ programlardan oluşan bir küme $\{v_1, v_2, v_3, v_4, v_5\}$.
- Her düğüm, v_i bir üçlü ile gösterilir (p_1, p_2, p_3) ,
- burada p_k özellik değerleridir $k = 1, 2$, veya 3
- $v_1 = (66, 20, 1)$
- $v_2 = (41, 10, 2)$
- $v_3 = (68, 5, 8)$
- $v_4 = (90, 34, 5)$
- $v_5 = (75, 12, 14)$



Benzer olmayan fonksiyonlar (1)

- ❑ Benzer olmayan (*dissimilarity function*) bir fonksiyon aşağıdaki gibi tanımlanır.
- ❑ Her bir düğüm çifti $v = (p_1, p_2, p_3)$ ve $w = (q_1, q_2, q_3)$ ile gösterilsin.

$$s(v,w) = \sum_{k=1}^3 |p_k - q_k|$$

- ❑ v ve w gibi iki programın *dissimilarity* $s(v,w)$ ile ölçülür.
- ❑ N seçilen sabit bir sayı olsun. Eğer $s(v,w) < N$ ise v ve w arasındaki kenar eklenir. Sonra:
- ❑ Eğer $v = w$ veya v ve w arasında bir yol varsa v ve w nun aynı sınıfta olduğunu söyleyebiliriz.

Benzer olmayan fonksiyonlar(2)

■ $N = 25$ (denemeler ile belirleniyor)

$$s(v_1, v_2) = 36$$

$$s(v_2, v_3) = 38$$

$$s(v_3, v_4) = 54$$

$$s(v_1, v_3) = 24$$

$$s(v_2, v_4) = 76$$

$$s(v_3, v_5) = 20$$

$$s(v_1, v_4) = 42$$

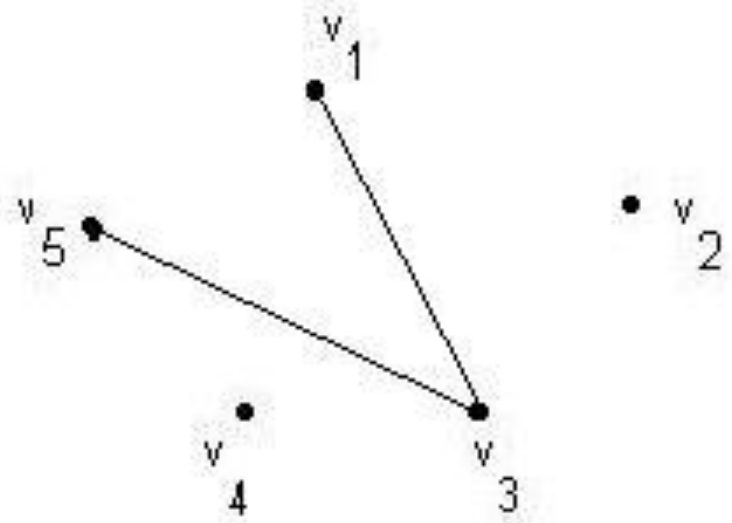
$$s(v_2, v_5) = 48$$

$$s(v_4, v_5) = 46$$

$$s(v_1, v_5) = 30$$

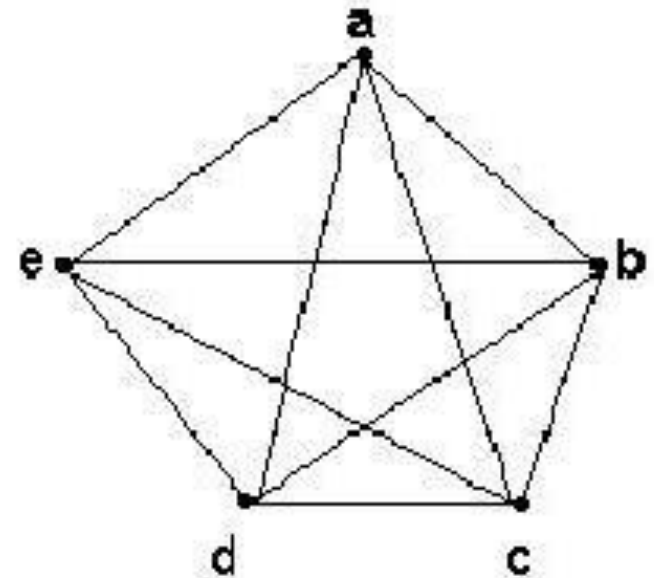
Benzer olmayan fonksiyonlar(2)

- $N = 25$.
- $s(v_1, v_3) = 24$, $s(v_3, v_5) = 20$
ve diğerleri $s(v_i, v_j) > 25$
- Üç sınıf vardır:
- $\{v_1, v_3, v_5\}$, $\{v_2\}$ and $\{v_4\}$
- **similarity graph** şekildeki gibidir.



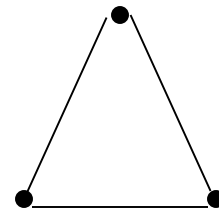
Tam (Complete) Graf K_n

- $n \geq 3$
- *complete graph* K_n : n adet düğüm içeren basit graf yapısındadır. Her düğüm, diğer düğümlere bir kenar ile bağlantılıdır.
- Şekilde K_5 grafi gösterilmiştir.
- Soru: K_3 , K_4 , K_6 graflarını çiziniz.



Cycles (Çember) Graf C_n

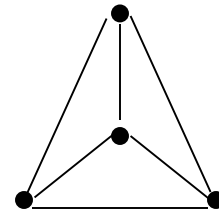
- $n \geq 3$
- *cycles graph* C_n : n adet düğüm ve $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$, düğüm çiftlerinden oluşan kenarlardan meydana gelir.
- Şekilde C_3 grafi gösterilmiştir.
- Soru: C_4, C_5, C_6 graflarını çiziniz.



C_3

Wheel (Tekerlek) Graf W_n

- *wheel graph* W_n : Cycle C_n grafına ek bir düğüm eklenerek oluşturulur. Eklenen yeni düğüm, diğer bütün düğümlere bağlıdır.
- Şekilde W_3 grafi gösterilmiştir.
- Soru: W_4 , W_5 , W_6 graflarını çiziniz.



W_3

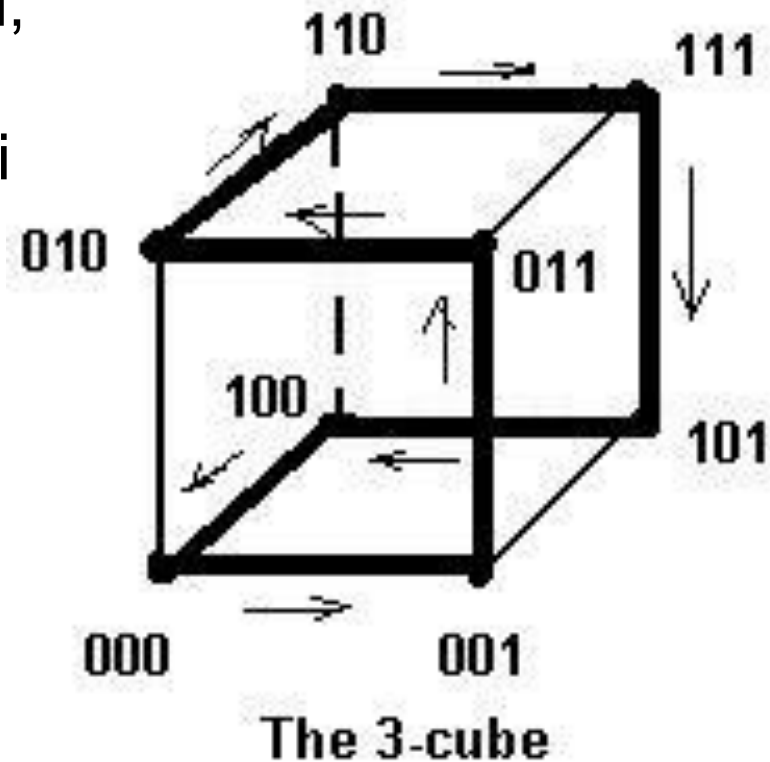
N-Cube (Küp) Graf Q_n

- *N-cube* Q_n : Grafın düğüm noktaları n uzunluğunda 2^n bit stringi ile gösterilir. Düğümlerin string değeri, bir düğümden diğerine geçerken aynı anda sadece bir bitin değerini değiştirmektedir.

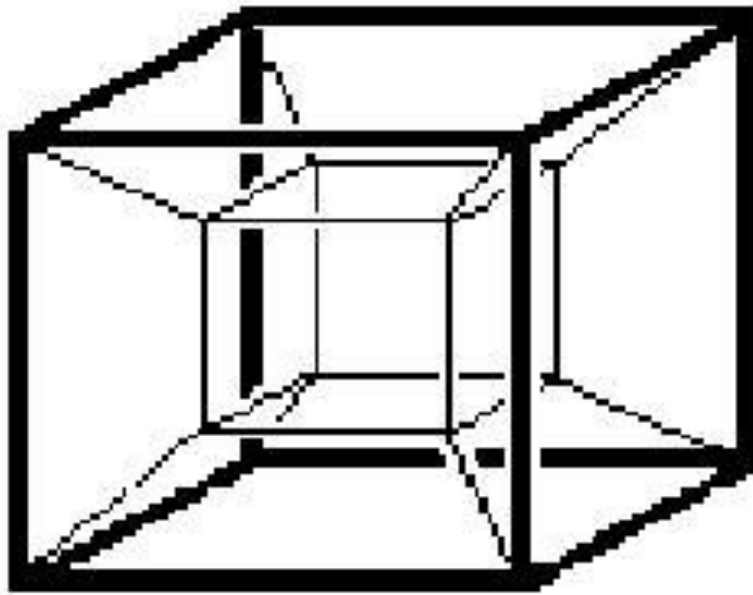
(000, 001, 011, 010, 110, 111, 101, 100, 000)

- Şekilde Q_3 grafi gösterilmiştir.

Soru: Q_1 , Q_2 graflarını çiziniz.



hypercube veya 4-cube



16 düğüm, 32 kenar ve 20 yüzey

□ Düğüm etiketleri:

0000 0001 0010 0011

0100 0101 0110 0111

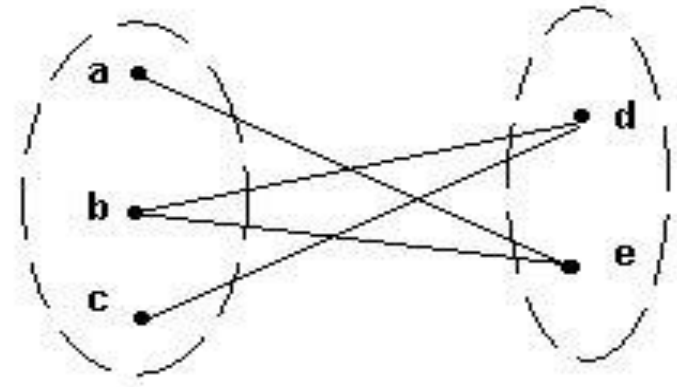
1000 1001 1010 1011

1100 1101 1110 1111

İki Parçalı (Bipartite) Graflar

□ G , bipartite graf ise:

- $V(G) = V(G_1) \cup V(G_2)$
- $|V(G_1)| = m, |V(G_2)| = n$
- $V(G_1) \cap V(G_2) = \emptyset$



- Bir grafi oluşturan düğümleri iki ayrı kümeye bölerek grafi ikiye ayırabiliriz. Bu ayırma işleminde izlenecek yol; bir kenar ile birbirine bağlanabilecek durumda olan düğümleri aynı küme içerisine yerleştirmemektir.
- Mevcut küme içerisindeki düğümler birbirlerine herhangi bir kenar ile bağlanmamalıdır.

-
- K_3 Bipartite graf mıdır ?

Hayır

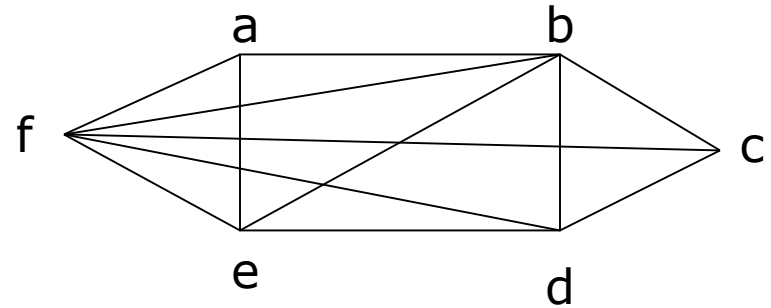
- C_6 Bipartite graf mıdır?

Evet

$\{1,3,5\}$ ve $\{2,4,6\}$

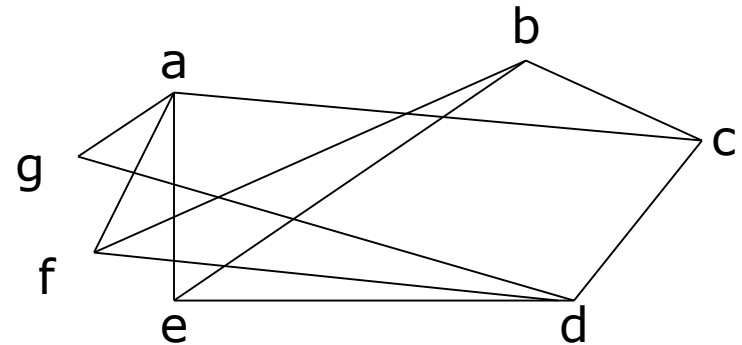
Yandaki graf Bipartite graf mıdır?

Hayır

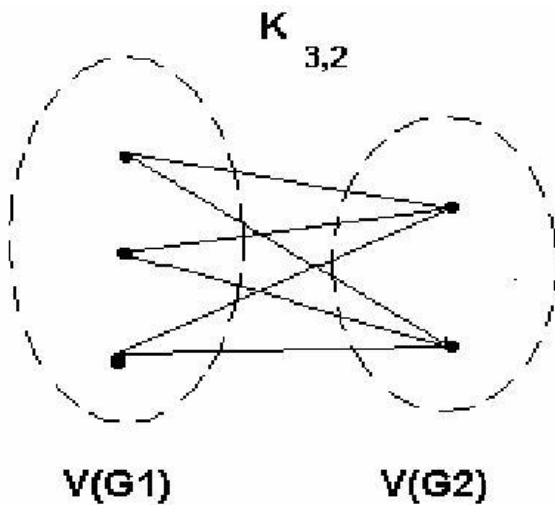


Yandaki graf Bipartite graf mıdır?

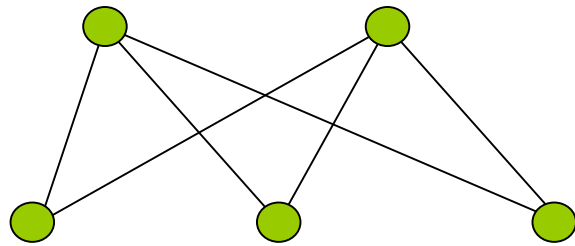
Evet. $\{a,b,d\}$ ve $\{c,e,f,g\}$



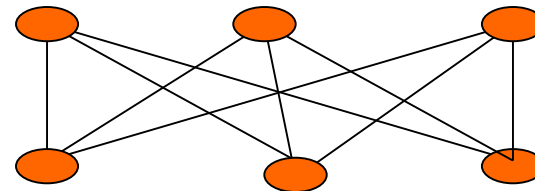
Tam (complete) bipartite graph $K_{m,n}$



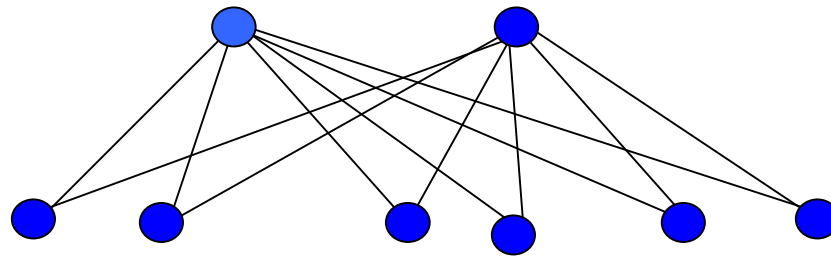
- ❑ *complete* bipartite graf $K_{m,n}$ şeklinde gösterilir. İlgili grafın düğümlerinin kümesi m ve n elemanlı iki alt kümeye ayrılır.
- ❑ Bir kenarı birbirine bağlayan iki düğümünde farklı alt kümelerin elemanı olmak zorundadırlar.
- ❑ $|V(G_1)| = m$
- ❑ $|V(G_2)| = n$



$K_{2,3}$



$K_{3,3}$



$K_{2,6}$

$K_n, C_n, W_n, K_{m,n}, Q_n$ graflarının kenar ve düğüm sayılarını formüle edecek olursak:

K_n n düğüm $n(n-1)/2$ kenar

C_n n düğüm n kenar

W_n $n+1$ düğüm $2n$ kenar

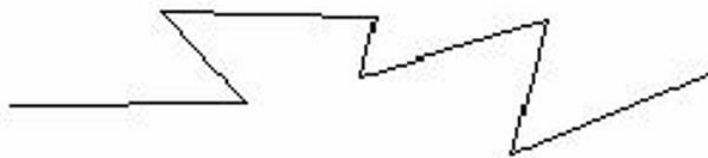
$K_{m,n}$ $m+n$ düğüm $m*n$ kenar

Q_n 2^n düğüm $n2^{n-1}$ kenar

Yollar (Paths) ve Döngüler(Cycles)

Path (Yol) Bir grafta bir düğümden diğer bir düğüme gidilirken izlenecek düğümlerin tamamına yol (path) denir.

- Eğer basit bir graf söz konusu ise yolun uzunluğu üzerinden geçilen kenar sayısına eşittir.
- Eğer ağırlıklı bir graf söz konusu ise yolun uzunluğu her bir kenarın aldığı değerlerin toplamına eşittir.

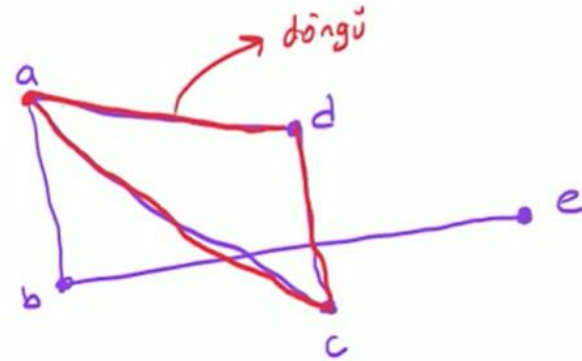
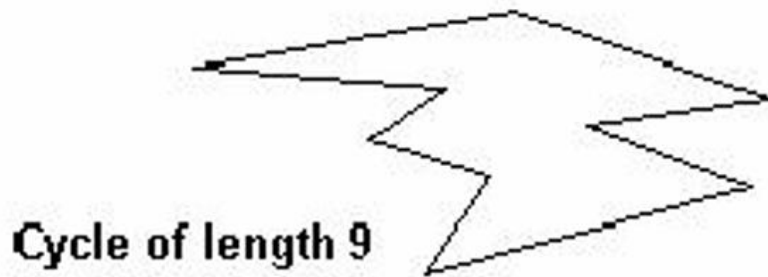


Path of length 7

- ▣ **n** uzunluğundaki bir yol'un (path) **$n+1$** adet düğümü ve n adet de ardışık kenarı vardır

Döngü (Cycle) Başladığı düğüme geri dönen ve aynı düğümden iki kez geçmeyen yola döngü (cycle) denir.

- Bir grafta ki kenar sayısı $\geq$ düğüm sayısı ise o grafta en az bir döngü vardır
- Uzunluğu n olan döngüde, n düğüm vardır.



Euler Döngüsü (Euler cycles)

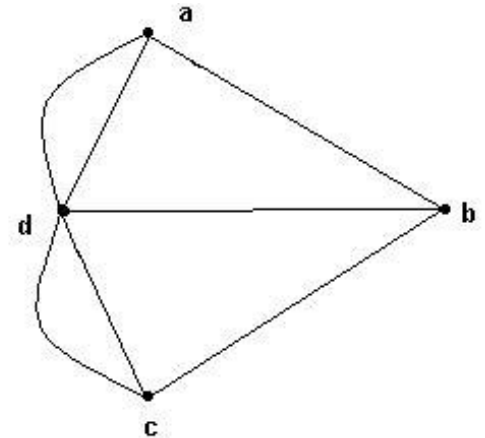
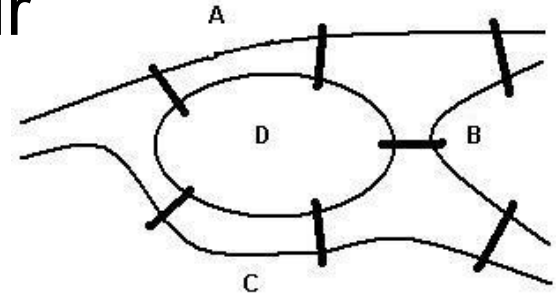
□ G grafi içerisindeki *Euler döngüsü* basit bir çevrim olup, G grafi içerisindeki her kenardan sadece bir kez geçilmesine izin verir.

□ Königsberg köprü problemi:

□ Yedi köprüden sadece bir kez geçerek başlangıç noktasına dönmek mümkün müdür?

□ Bu problemi grafa indirgeyelim.

□ Kenarlar köprüleri ve düğüm noktaları da bölgeleri gösterebilir.

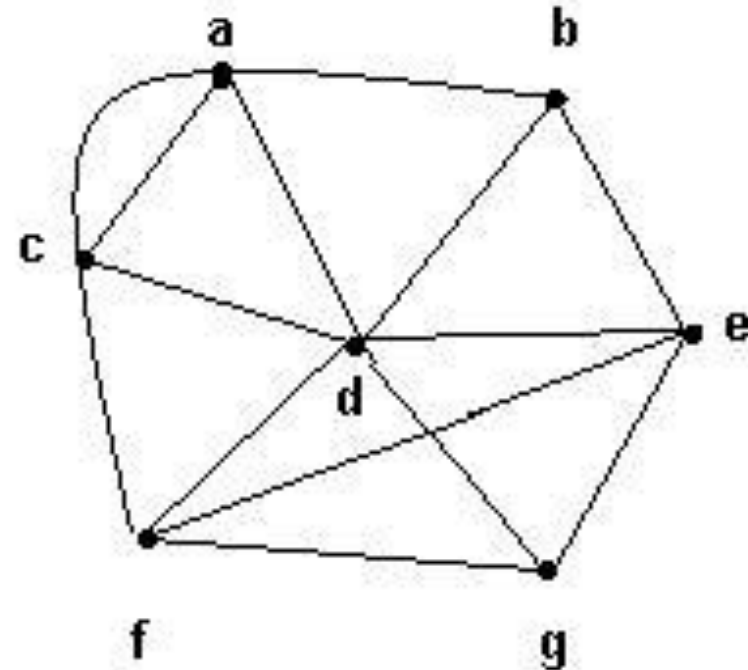


Bir düğümün derecesi

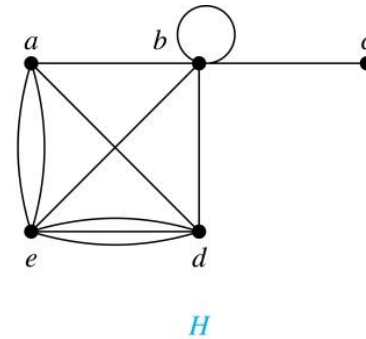
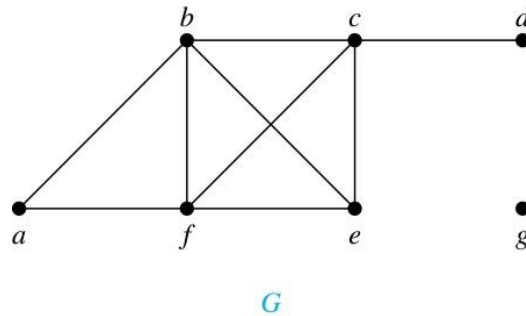
- v düğümünün derecesi $\delta(v)$ ile gösterilir ve bu da yönsüz bir grafta düğüme gelen kenarlar toplamıdır. Düğüm noktalarındaki döngü, düğüm derecesine 2 kez katılır.

■ Örnek:

- $\delta(a) = 4$, $\delta(b) = 3$,
- $\delta(c) = 4$, $\delta(d) = 6$,
- $\delta(e) = 4$, $\delta(f) = 4$,
- $\delta(g) = 3$.



Example: Verilen G ve H graflarındaki düğümlerin dereceleri ve komşuları nedir?



G: $\deg(a) = 2, \deg(b) = \deg(c) = \deg(f) = 4, \deg(d) = 1,$
 $\deg(e) = 3, \deg(g) = 0.$

$N(a) = \{b, f\}, N(b) = \{a, c, e, f\}, N(c) = \{b, d, e, f\}, N(d) = \{c\},$
 $N(e) = \{b, c, f\}, N(f) = \{a, b, c, e\}, N(g) = \emptyset.$

H: $\deg(a) = 4, \deg(b) = \deg(e) = 6, \deg(c) = 1, \deg(d) = 5.$

$N(a) = \{b, d, e\}, N(b) = \{a, b, c, d, e\}, N(c) = \{b\},$
 $N(d) = \{a, b, e\}, N(e) = \{a, b, d\}.$

Euler Yolu (path) ve Euler Döngüsü (cycle)

Euler Yolu (Euler Path) : Yönsüz bir graftaki tüm yollardan bir kez geçmek şartıyla (aynı düğümden birden fazla geçilebilir) tüm düğümleri dolaşan bir yol bulunabiliyorsa bu yola **Euler Yolu** denir.

❖ Euler Yolu bulunan grafa yarı Euler graf (semi Euler) denir.

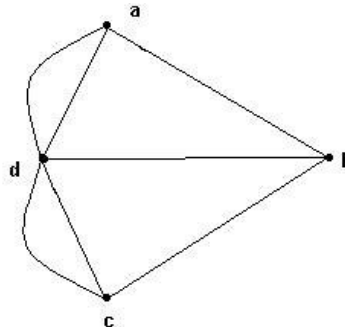
Euler Döngüsü (Euler Cycle) : Euler yolunun başlangıç ve bitiş düğümleri aynı ise bu yola **Euler Döngüsü** denir.

❖ Euler döngüsü içeren grafa Euler grafi denir.

- Yönlü graflarda bu tanımlara yön ifadesi eklenerek yönlü Euler yolu ve yönlü Euler döngüsü adı verilir. Graf üzerinde gezinirken yöne dikkat edilmelidir.
- Euler döngüsü varsa Euler yolu da vardır. Euler yolu varsa Euler döngüsü olmayabilir.

devam...

- Yönsüz bağlı bir grafın bütün **düğümünün dereceleri çift** ise bu graf Euler grafıdır. Hem Euler yolu hem de Euler döngüsü vardır.
- Bir yönsüz grafta Euler yolu bulunabilmesi için **iki veya sıfır sayıda tek düğüm derecesi** bulunmalıdır.
- Yönlü bir grafta ancak ve ancak bütün düğümlerin giren ve çıkan derecelerinin toplamı eşit ise Euler grafı olabilir.
- Königsberg bridge problemi bir Euler grafı değildir.
- Königsberg bridge probleminin çözümü yoktur.



Grafın düğüm derecelerinin toplamı

- Sıfır dereceli bir düğüm ***isolated*** olarak adlandırılır. Isolated olan bir düğümden, başka bir düğüme yol yoktur.
- Düğüm derecesi bir olan düğüme ***pendant*** denir.

■ Teorem: *Handshaking*

e adet kenarlı ve ***n*** adet düğümlü bir grafın $G(V,E)$ düğümlerinin dereceleri toplamı kenar sayısının iki katıdır.

$$\sum_{i=1}^n \delta(v_i) = 2e$$

Örnek: Her birinin derecesi 6 olan 10 düğümlü bir grafın kaç tane kenarı vardır.

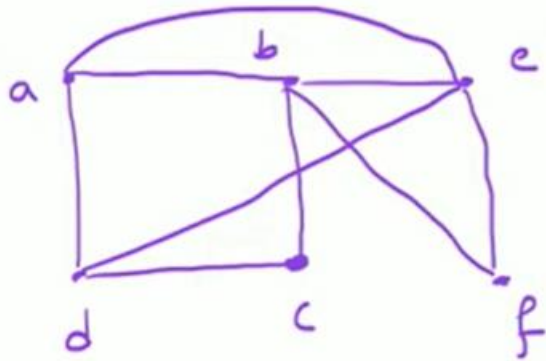
$$e=30$$

□ G grafında (v,w) yönlü bir kenar olsun ve yön v 'den w 'ya verilsin. v ***initial vertex***, w 'da ***terminal*** veya ***end vertex*** olarak adlandırılır. Bir düğüm noktasında döngü söz konusu ise bu düğümün *initial vertex*'i ve *end vertex*'i birbirinin aynıdır.

□ Yönlü bir grafta, herhangi bir düğümün ***in_degree***'si $\delta^-(v)$, ***out_degree***'si $\delta^+(v)$ olarak gösterilir.

□ Yönlü bir grafın *in_degree* ve *out_degree*'lerinin toplamı birbirinin aynıdır.

$$\sum_{v \in V} \delta^-(v) = \sum_{w \in V} \delta^+(w)$$

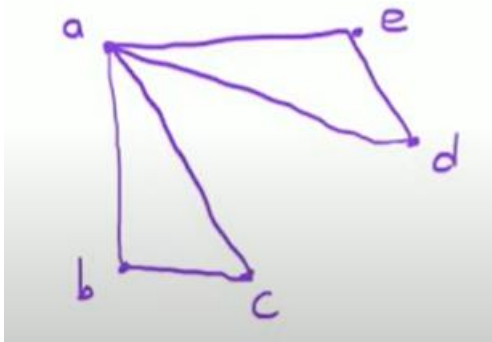


Örnek: Verilen graf Euler yolu ve döngüsüne sahip midir?

$$a \rightarrow 3 \quad b \rightarrow 4 \quad c \rightarrow 2 \quad d \rightarrow 3 \quad e \rightarrow 4 \quad f \rightarrow 2$$

- Tüm dereceler çift olmadığı için **Euler döngüsü yoktur.**
- İki tane düğümün derecesi tek diğer düğümlerin dereceleri çift olduğu için **Euler yolu vardır.**

Euler Döngülerinin Sayısı Nasıl Bulunur?



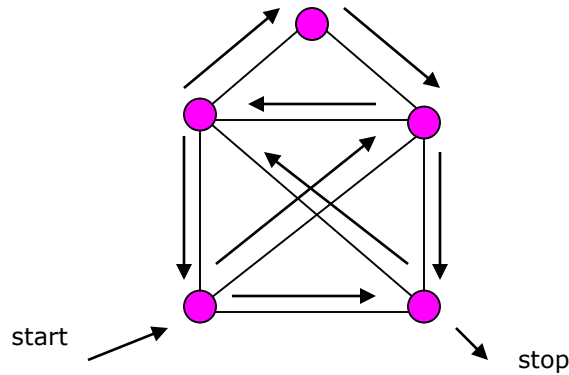
$$a \rightarrow 4 \quad b \rightarrow 2 \quad c \rightarrow 2 \quad d \rightarrow 2 \quad e \rightarrow 2$$

- ✓ Euler yolu var
- ✓ Euler döngüsü var

Bütün düğüm derecelerinin bir eksikliğinin faktöreyellerinin çarpımı Euler döngü sayısını verir.

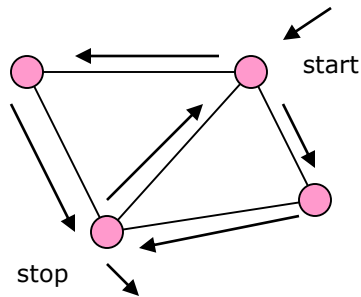
$$3! * 1! * 1! * 1! * 1! = 6 \quad \text{farklı Euler döngüsü vardır}$$

Örnek: Aşağıda verilmiş olan graflardan hangilerinde her kenardan en az bir kez geçirilerek graf gezilmiştir, hangileri Euler grafıdır, eğer değilse sebebi nedir ?



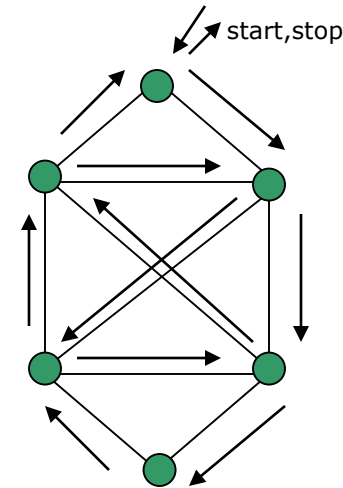
Euler yolu var, Euler grafı değil

Düğüm dereceleri çift değil

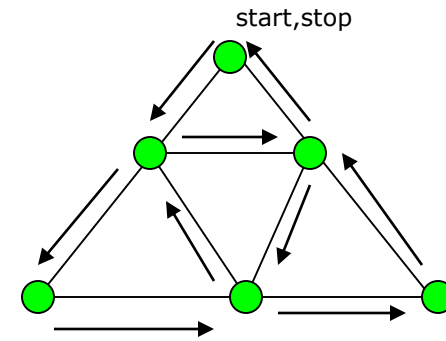


Euler yolu var, Euler grafı değil

Düğüm dereceleri çift değil



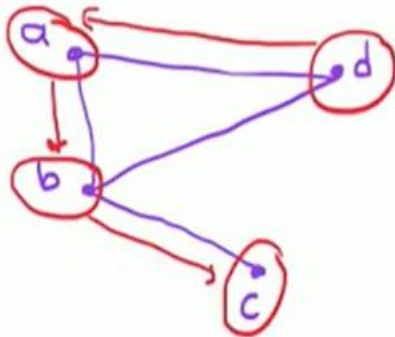
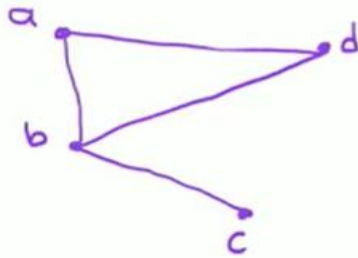
Euler grafı



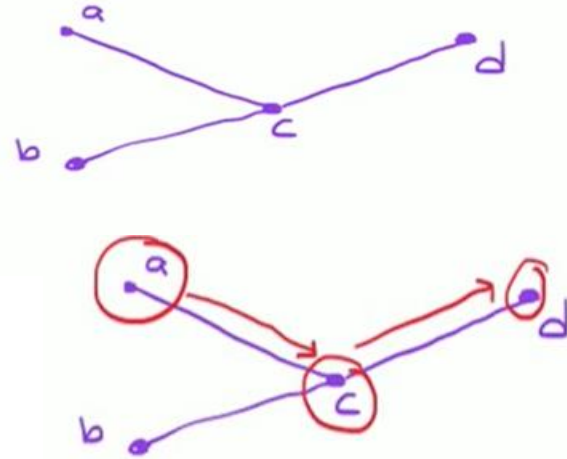
Euler grafı

Hamilton Yolu, Döngüsü, Grafi

Hamilton Yolu : Eğer bağlı bir grafta bütün düğümlerden geçen ve bir yoldan sadece bir kez geçmek kaydıyla bir yol varsa buna **Hamilton yolu** denir. Her yoldan geçmek zorunluluğu yoktur.



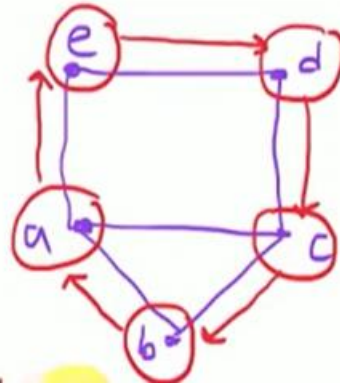
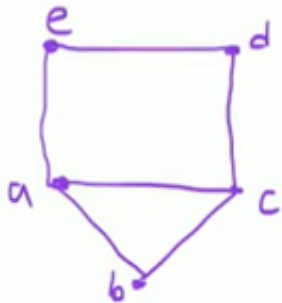
dabc şeklinde Hamilton yolu vardır



Hamilton yolu yoktur.

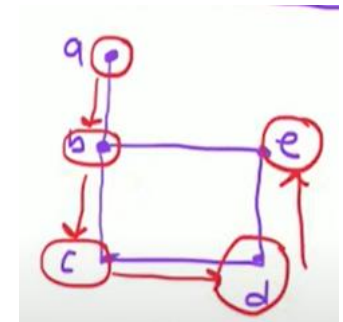
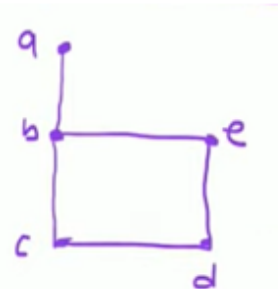
devam...

Hamilton Döngüsü : Eğer bağlı bir grafta bütün düğümlerden bir kez geçerek (başlangıç düğümü hariç) ve bir yoldan sadece bir kez geçmek kaydıyla başladığımız düğüme dönen bir yol varsa buna **Hamilton döngüsü** denir. Her yoldan geçmek zorunluluğu yoktur.



aedcba Hamilton döngüsü var

Hamilton döngüsü yoktur.
Hamilton yolu vardır. abcde

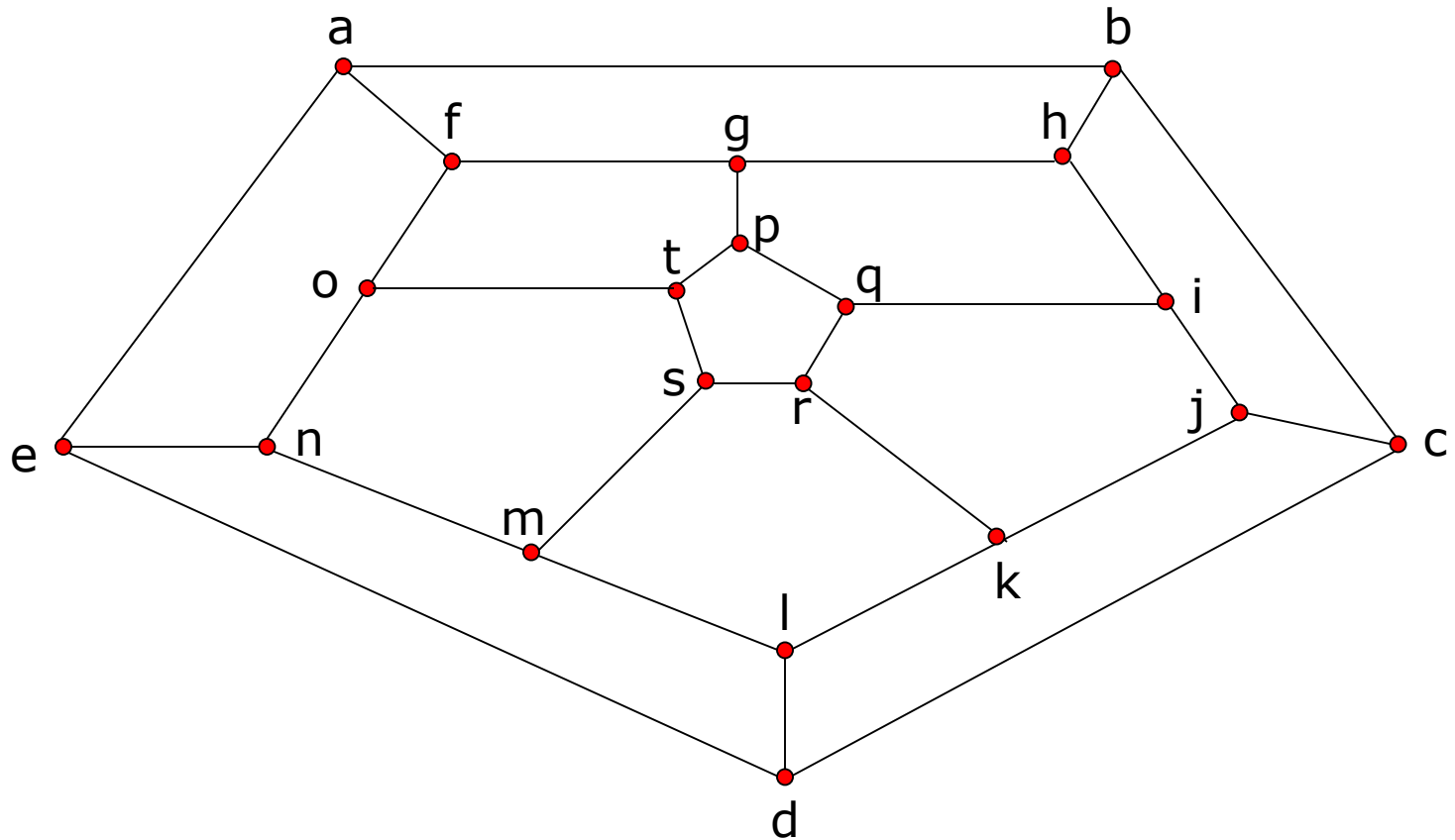


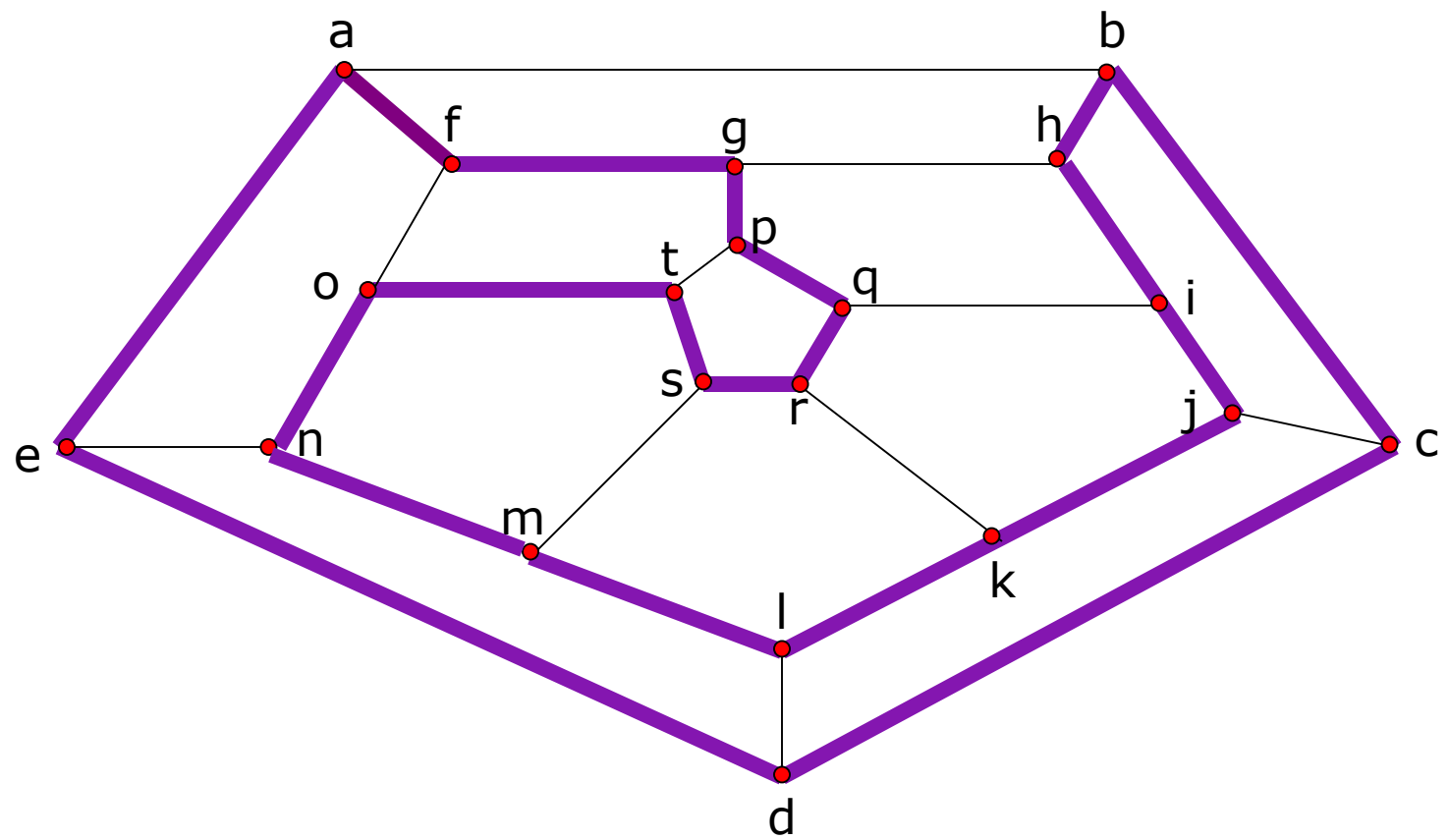
devam...

Hamilton Graf : Hamilton döngüsü içeren grafa Hamilton Grafı denir.

- Hamilton döngüsüne sahip bir graftan bir kenar çıkarılarak Hamilton yoluna çevrilebilir.
- Hamilton döngüsü içeren her grafa Hamilton yolu içerir.
- Birden fazla Hamilton yolu ve döngüsü aynı grafa bulunabilir.

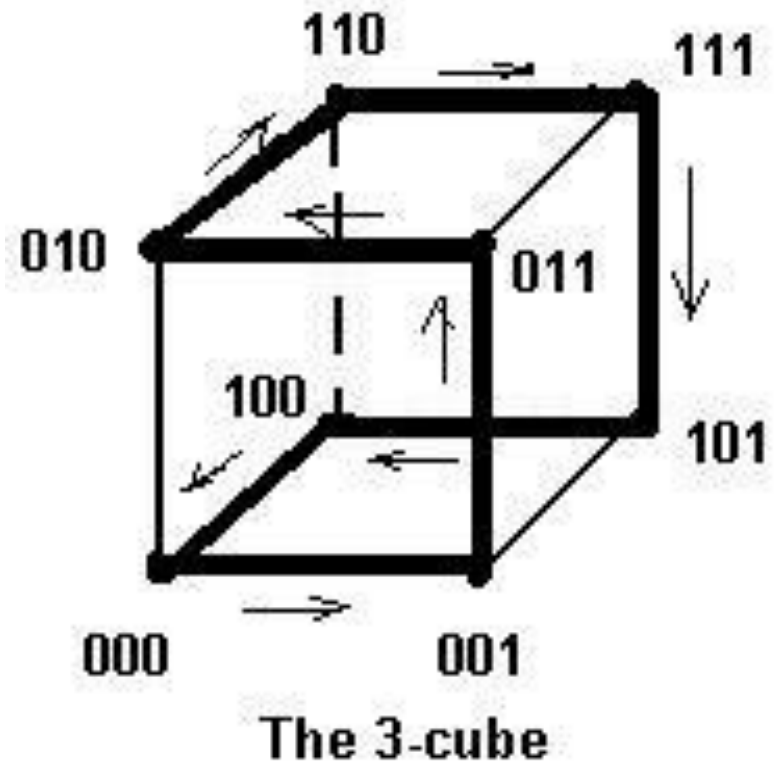
Örnek : Hamilton graf mıdır?





3-cube

Hamiltonian cycle
(000, 001, 011, 010,
110, 111, 101, 100,
000) örnek bir graf
3-cube olarak
verilebilir.



Grafların Temsil Edilmesi

- ❑ Komşuluk Listesi (Adjacency Lists)
- ❑ Komşuluk Matrisi (Adjacency Matrices)
- ❑ İlişki Matrisi (Incidence Matrices)

Komşuluk Listesi (Adjacency Lists)

Tanım: Yönsüz bir grafta her bir düğüme komşu olan düğümlerin veya yönlü olan bir grafta her düğümden çıkan kenarların bağlandığı düğümlerin listesini çıkarmak için kullanılır.

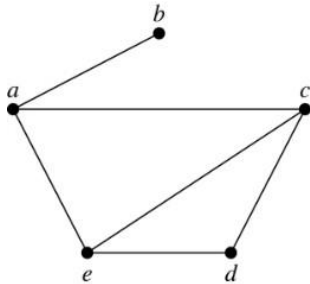
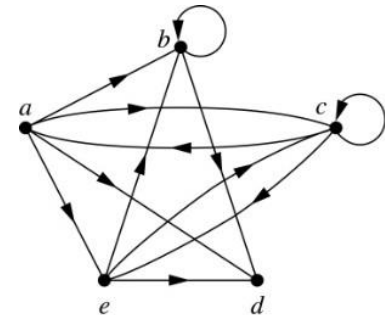


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	Adjacent Vertices
<i>a</i>	<i>b, c, e</i>
<i>b</i>	<i>a</i>
<i>c</i>	<i>a, d, e</i>
<i>d</i>	<i>c, e</i>
<i>e</i>	<i>a, c, d</i>

TABLE 2 An Adjacency List for a Directed Graph.

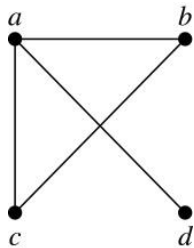
Initial Vertex	Terminal Vertices
<i>a</i>	<i>b, c, d, e</i>
<i>b</i>	<i>b, d</i>
<i>c</i>	<i>a, c, e</i>
<i>d</i>	<i>b, c, d</i>
<i>e</i>	<i>b, c, d</i>



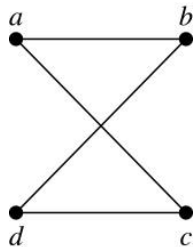
Komşuluk Matrisi (Adjacency Matrice)

Tanım: Bir grafın komşuluk matrisi $\mathbf{A}_G = [a_{ij}]$,

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$



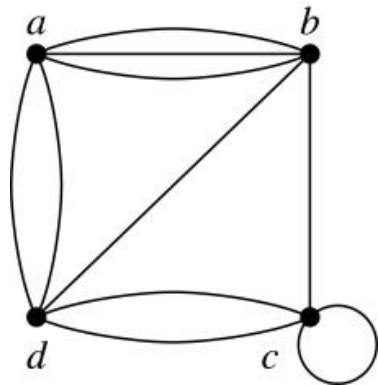
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Not: Bir graf seyrek olduğunda, yani sahip olduğu kenar sayısı toplam olası kenar sayısına göre çok az ise, bu grafı komşuluk listesi kullanarak temsil etmek komşuluk matrisinden çok daha verimlidir.

(devam...)



$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

İlişki Matrisi (Incidence Matrice)

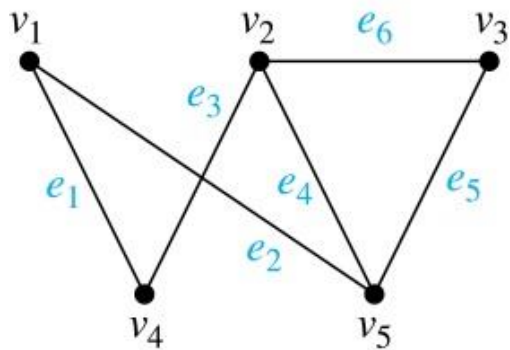
Tanım: $G = (V, E)$, düğümleri $v_1, v_2, \dots, v_n$ ve kenarları $e_1, e_2, \dots, e_m$ olan yönsüz bir graf olsun. v ve e 'nin sıralamasına göre ilişki matrisi $n \times m$ boyutlu olup, $M = [m_{ij}]$

$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

olsun.

(devam...)

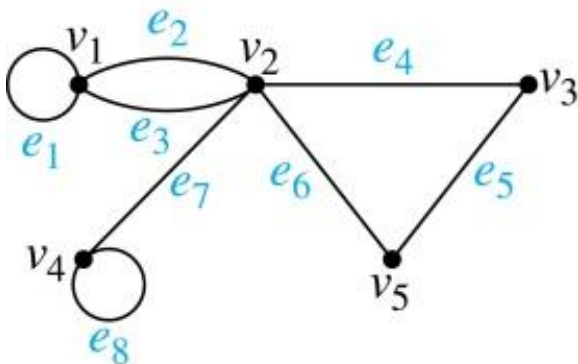
Basit graf ve İlişki matrisi



$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Satırlar düğümleri, sütunlar da kenarları temsil etsin.

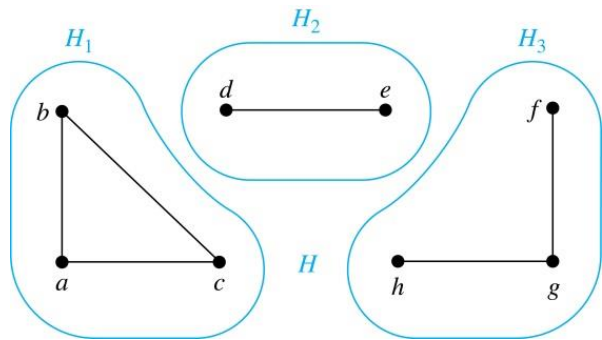
Pseudograph ve İlişki matrisi



$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Graflarda Bağlılıklar

Tanım: Bir grafta herhangi bir düğümden, herhangi bir başka düğüme her zaman için bir yol varsa bu grafa **Bağlı Graf (connected graph)** denir. Bir düğüm için bile bu şart sağlanmaz ise **Bağlı Olmayan Graf (disconnected graph)** adı verilir.



- ❖ H_1 grafi Bağlantılı Graftır
- ❖ H_1 ve H_2 den oluşan H grafi ise Bağlantılı Olmayan Graftır

Bağlı Bileşenler (Connected Components)

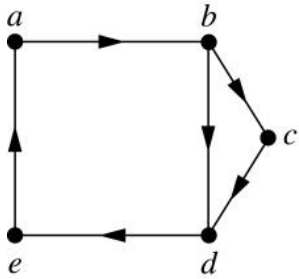
H_1 , H_2 ve H_3 den oluşan H grafının Bağlı Bileşenleri ise: H_1 , H_2 ve H_3

Yönlü Graflarda Bağlılıklar

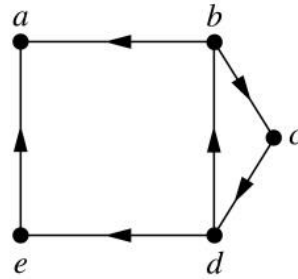
Tanım: Yönlü bir grafta seçilen herhangi iki düğüm arasında gidiş ve dönüş için bir yol bulunuyorsa bu grafa **Güçlü Şekilde Bağlı Graf (Strongly Connected Graph)** denir.

Tanım: A directed graph is *weakly connected* if there is a path between every two vertices in the underlying undirected graph, which is the undirected graph obtained by ignoring the directions of the edges of the directed graph.

devam...



G



H

G : strongly connected graph
 H : not strongly connected graph

EN KISA YOL (SHORTEST PATH) ALGORİTMASI

Dijkstra's Algorithm

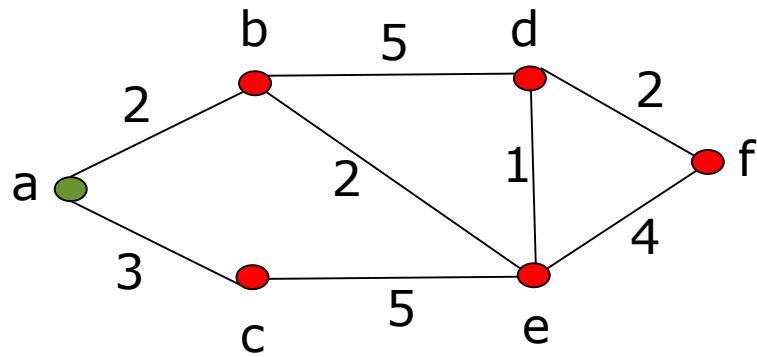
Dijkstra's Algorithm

Dijkstra's algorithm is known to be a good algorithm to find a shortest path.

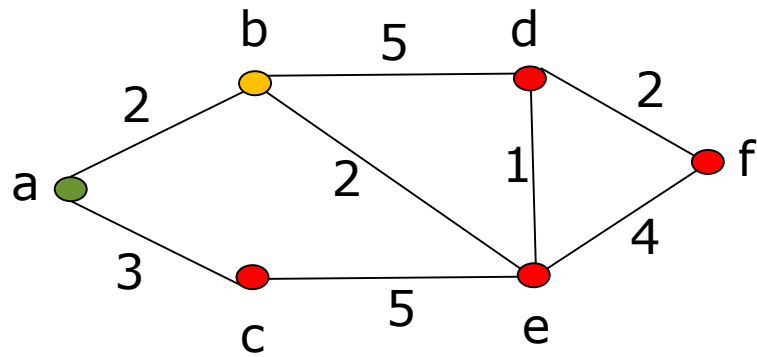
1. Set $i=0$, $S_0 = \{u_0=s\}$, $L(u_0)=0$, and $L(v)=\text{infinity}$ for $v \neq u_0$.
If $|V| = 1$ then stop, otherwise go to step 2.
2. For each v in $V \setminus S_i$, replace $L(v)$ by $\min\{L(v), L(u_i)+d_v^{u_i}\}$.
If $L(v)$ is replaced, put a label $(L(v), u_i)$ on v .
3. Find a vertex v which minimizes $\{L(v) : v \in V \setminus S_i\}$, say u_{i+1} .
4. Let $S_{i+1} = S_i \cup \{u_{i+1}\}$.
5. Replace i by $i+1$. If $i=|V|-1$ then stop, otherwise go to step 2.

The time required by Dijkstra's algorithm is $O(|V|^2)$.

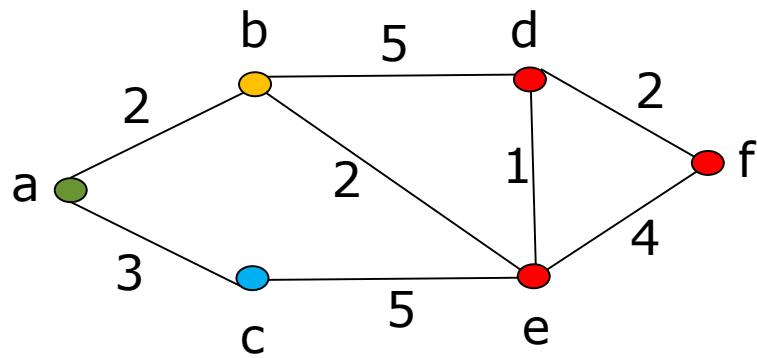
It will be reduced to $O(|E|\log|V|)$ if heap is used to keep $\{v \in V \setminus S_i : L(v) < \text{infinity}\}$.



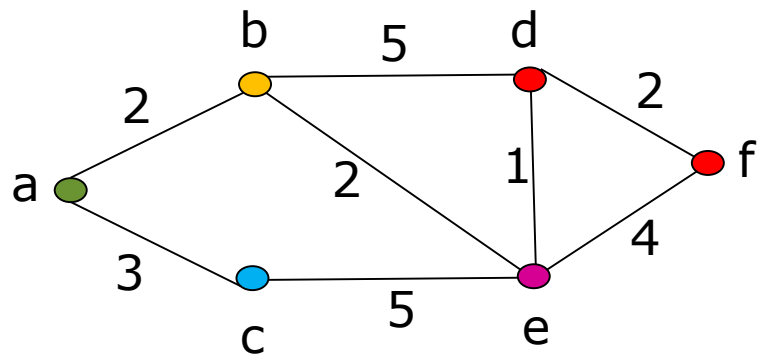
a	0					
b	M					
c	M					
d	M					
e	M					
f	M					



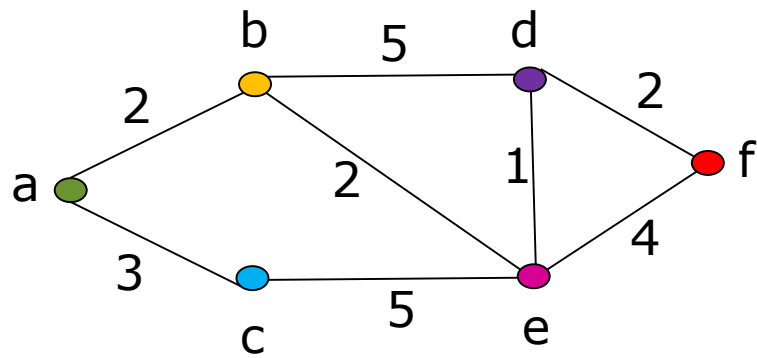
a	0	-				
b	M	2a				
c	M	3a				
d	M	M				
e	M	M				
f	M	M				



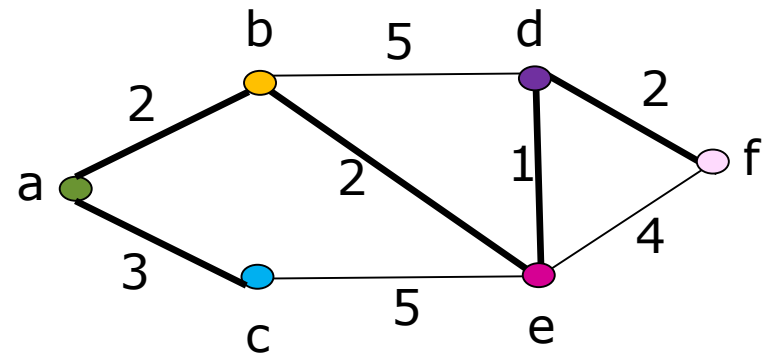
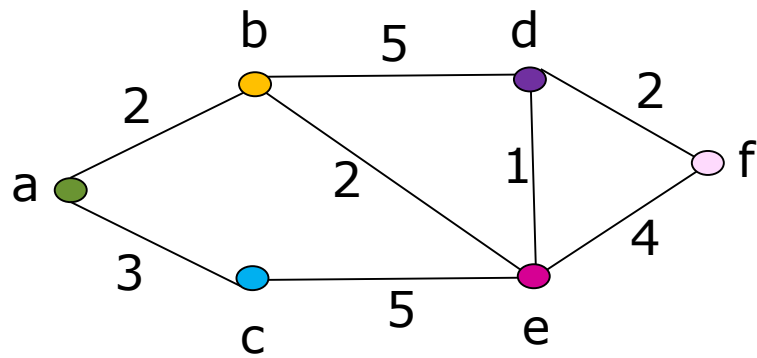
a	0	-				
b	M	2a	-			
c	M	3a	3a			
d	M	M	7ab			
e	M	M	4ab			
f	M	M	M			



a	0	-				
b	M	2a	-			
c	M	3a	3a	-		
d	M	M	7ab	7ab		
e	M	M	4ab	8ac ✕		
f	M	M	M	M		



a	0	-				
b	M	2a	-			
c	M	3a	3a	-		
d	M	M	7ab	7ab	5abe	
e	M	M	4ab	8ac ✕	-	
f	M	M	M	M	8abe	



a	0	-				
b	M	2a	-			
c	M	3a	3a	-		
d	M	M	7ab	7ab	5abe	
e	M	M	4ab	8ac ✕	-	
f	M	M	M	M	8abe	7abed

Mathematical Programming

[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)

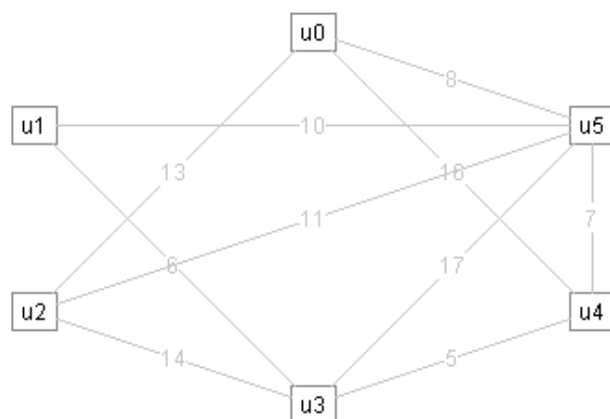
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

CLICK on the java applet below for several times to find a shortest path from u_0 .

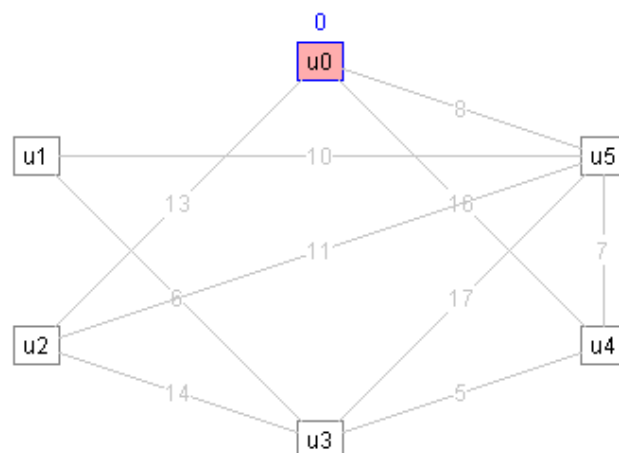


A (6,10) graph

Mathematical ProgrammingSimplexTwophaseDijkstraPrimKruskalFord-FulkersonDijkstra
Java applet demos:

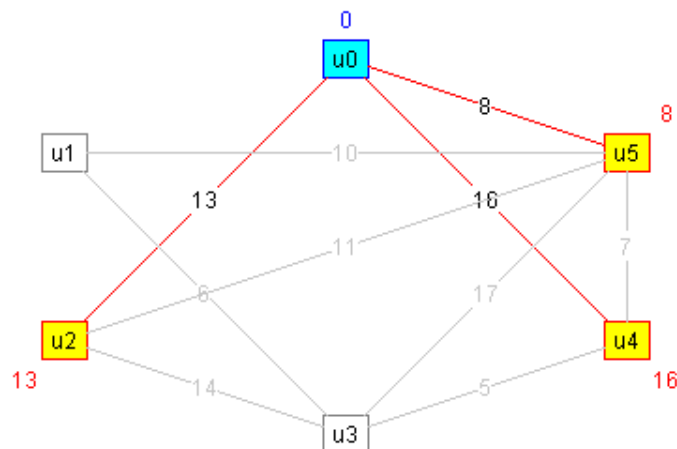
- demo1
- demo2
- demo3
- demo4
- demo5
- demo6
- demo7
- demo8
- demo9
- demo10

FAQ

Java Applet Demos of Dijkstra's AlgorithmCLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

**Mathematical
Programming**[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)Dijkstra
Java applet demos:

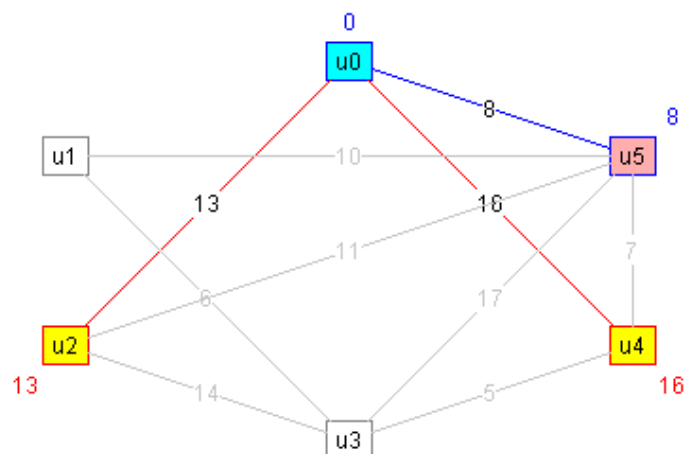
- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)**Java Applet Demos of Dijkstra's Algorithm**CLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

Mathematical ProgrammingSimplexTwophaseDijkstraPrimKruskalFord-FulkersonDijkstra
Java applet demos:

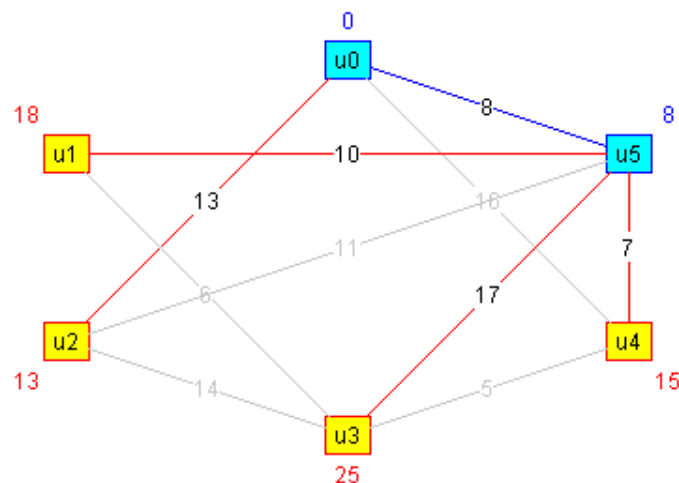
- demo1
- demo2
- demo3
- demo4
- demo5
- demo6
- demo7
- demo8
- demo9
- demo10

FAQ

Java Applet Demos of Dijkstra's AlgorithmCLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

**Mathematical
Programming**[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)Dijkstra
Java applet demos:

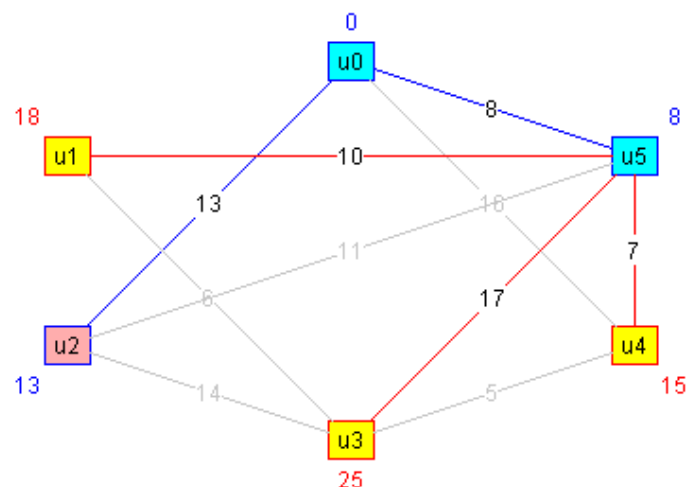
- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)**Java Applet Demos of Dijkstra's Algorithm**CLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

Mathematical ProgrammingSimplexTwophaseDijkstraPrimKruskalFord-FulkersonDijkstra
Java applet demos:

- demo1
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

FAQ

Java Applet Demos of Dijkstra's AlgorithmCLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

Mathematical Programming

[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)

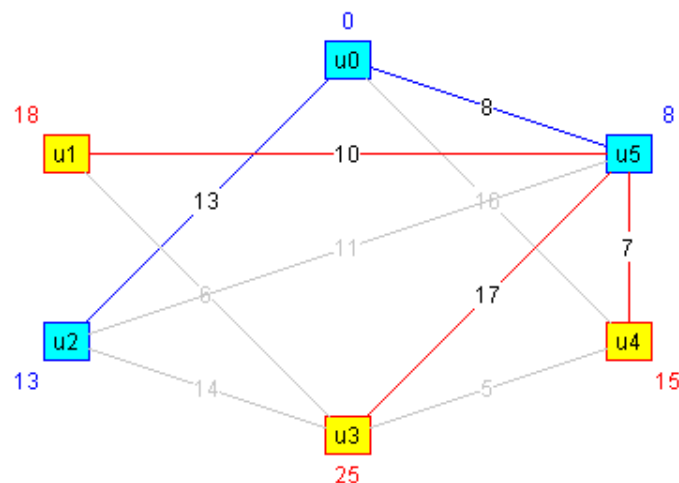
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

CLICK on the java applet below for several times to find a shortest path from u_0 .

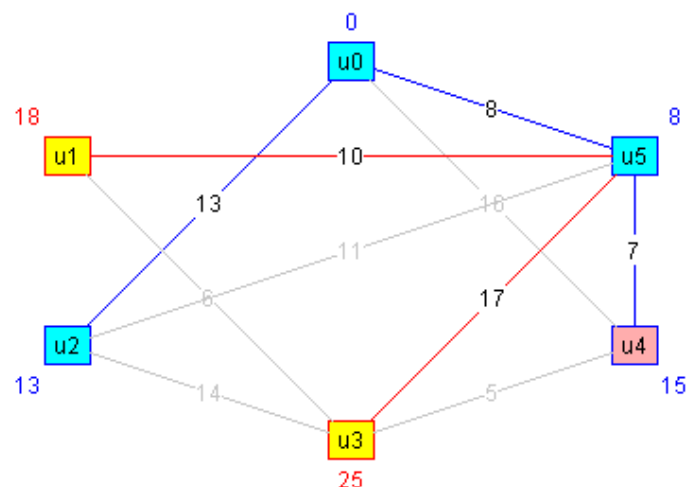


A (6,10) graph

Mathematical ProgrammingSimplexTwophaseDijkstraPrimKruskalFord-FulkersonDijkstra
Java applet demos:

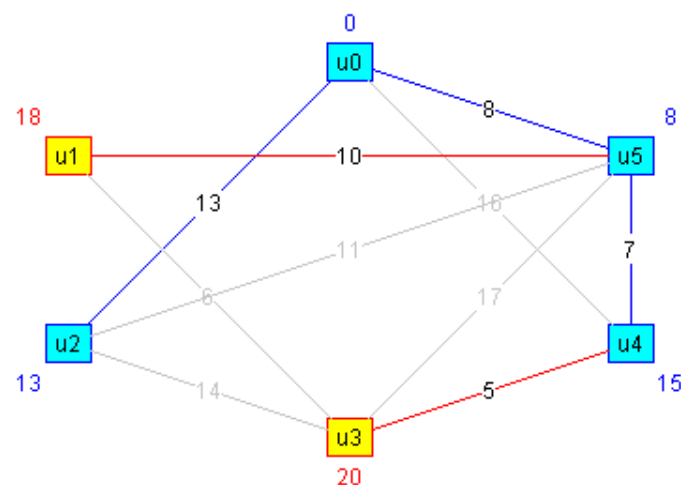
- demo1
- demo2
- demo3
- demo4
- demo5
- demo6
- demo7
- demo8
- demo9
- demo10

FAQ

Java Applet Demos of Dijkstra's AlgorithmCLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

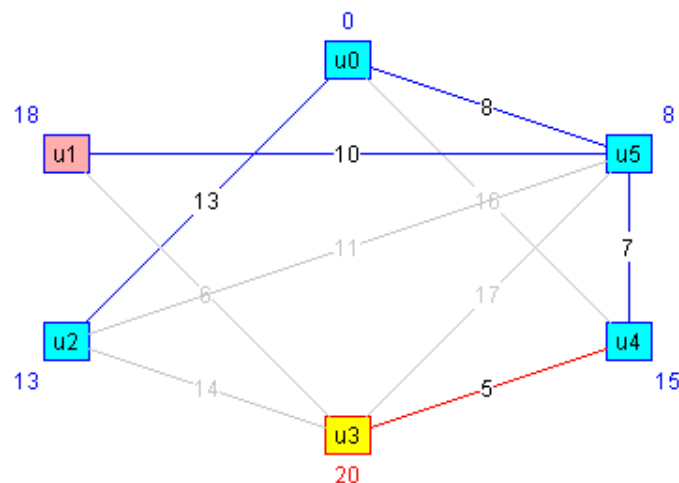
**Mathematical
Programming**[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)**Java Applet Demos of Dijkstra's Algorithm**CLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

**Mathematical
Programming**[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)**Java Applet Demos of Dijkstra's Algorithm**CLICK on the java applet below for several times to find a shortest path from u_0 .A (6,10) graph

Mathematical Programming

[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)

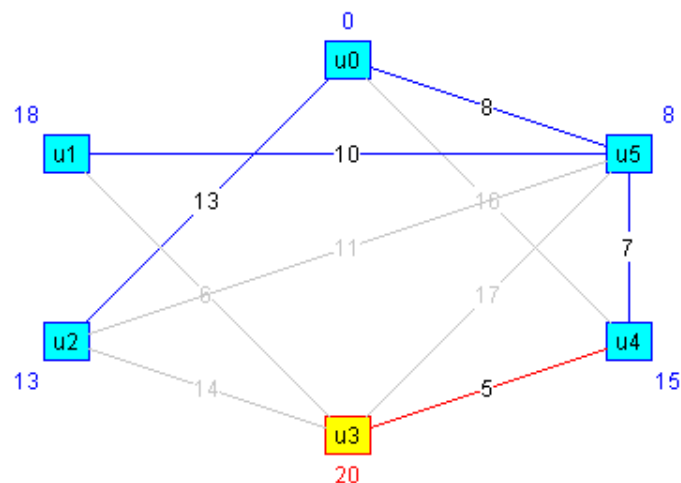
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

CLICK on the java applet below for several times to find a shortest path from u_0 .



A (6,10) graph

Mathematical Programming

[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)

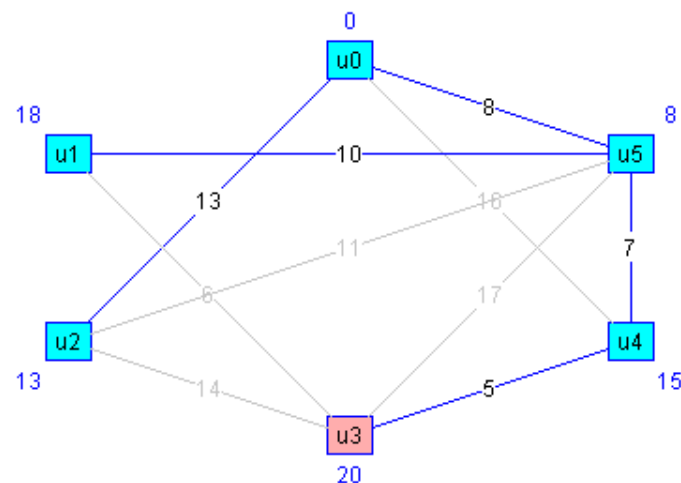
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

CLICK on the java applet below for several times to find a shortest path from u_0 .



A (6,10) graph

Mathematical Programming

[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)

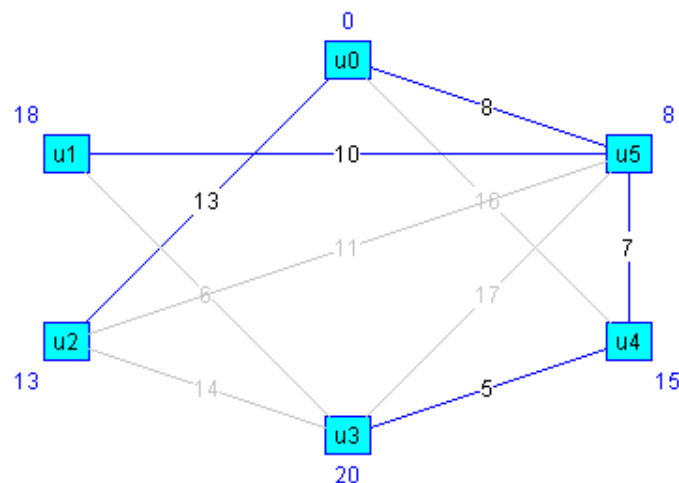
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

CLICK on the java applet below for several times to find a shortest path from u_0 .



A (6,10) graph

Mathematical Programming

[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)

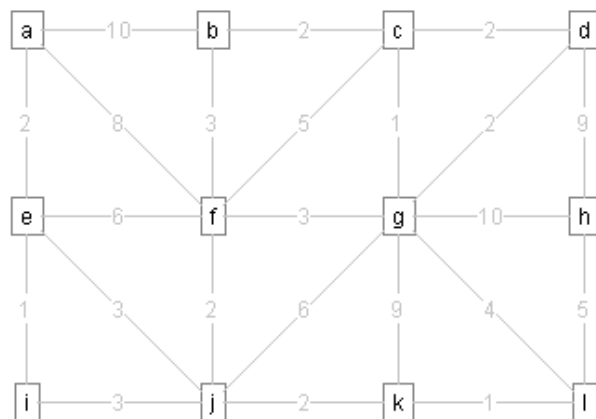
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

Click on the applet below for several times to find a shortest path from node a.



[A \(12,23\) graph](#)

Mathematical Programming

[Simplex](#)
[Twophase](#)
[Dijkstra](#)
[Prim](#)
[Kruskal](#)
[Ford-Fulkerson](#)

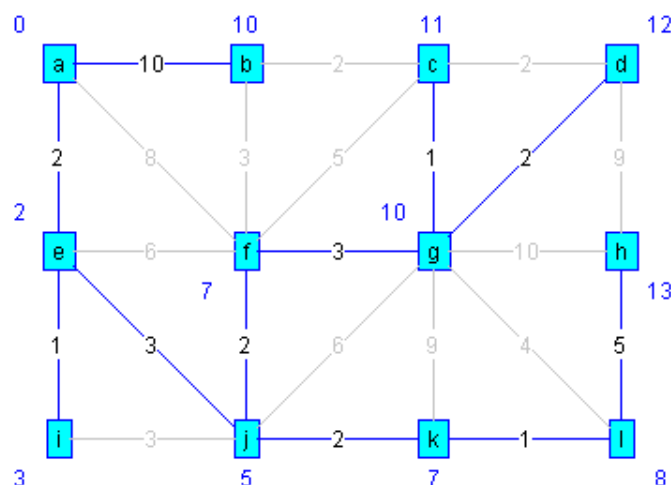
Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

[FAQ](#)

Java Applet Demos of Dijkstra's Algorithm

Click on the applet below for several times to find a shortest path from node a.



A (12,23) graph

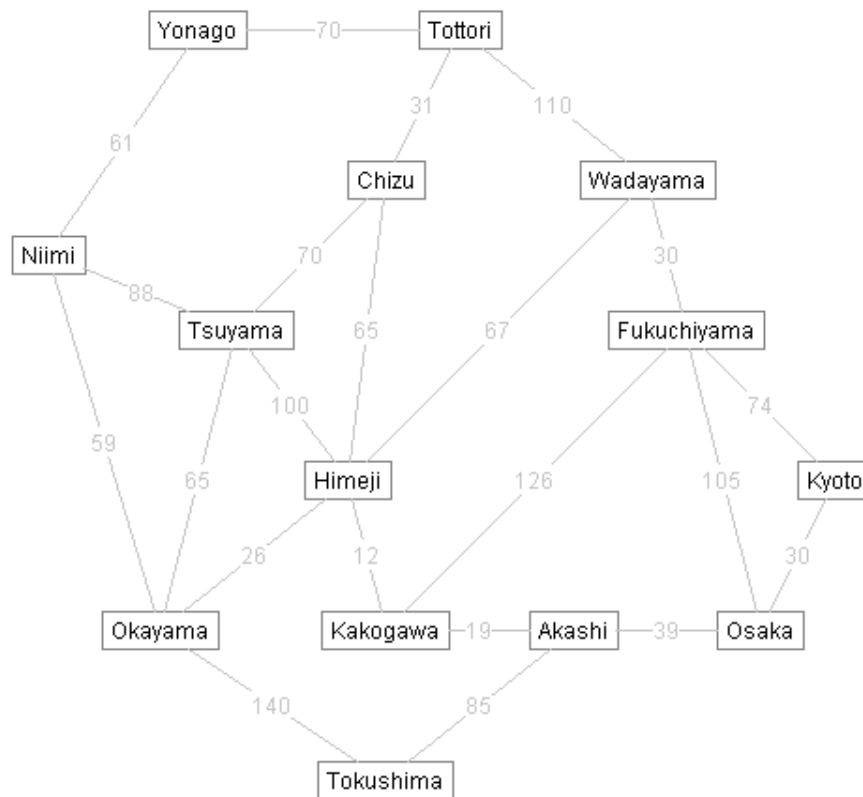
**Mathematical
Programming**[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

FAQ

Java Applet Demos of Dijkstra's Algorithm

Click on the applet below for several times to find a shortest path from Tokushima.

[A \(14,22\) graph](#)

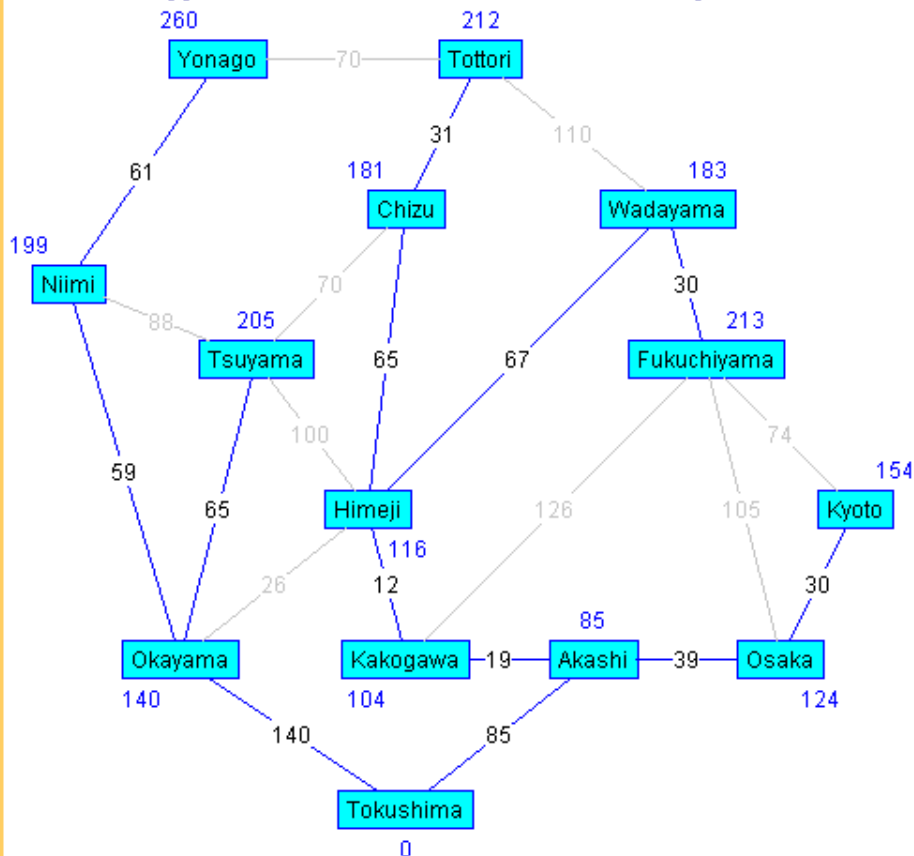
**Mathematical
Programming**[Simplex](#)[Twophase](#)[Dijkstra](#)[Prim](#)[Kruskal](#)[Ford-Fulkerson](#)Dijkstra
Java applet demos:

- [demo1](#)
- [demo2](#)
- [demo3](#)
- [demo4](#)
- [demo5](#)
- [demo6](#)
- [demo7](#)
- [demo8](#)
- [demo9](#)
- [demo10](#)

FAO

Java Applet Demos of Dijkstra's Algorithm

Click on the applet below for several times to find a shortest path from Tokushima.

[A \(14,22\) graph](#)

Graf Modelleri

Farklı alanlarda farklı graf modelleri kullanılır.

Niche Overlap Graf : Eko sistem içerisindeki farklı grupları modellemede kullanılır.

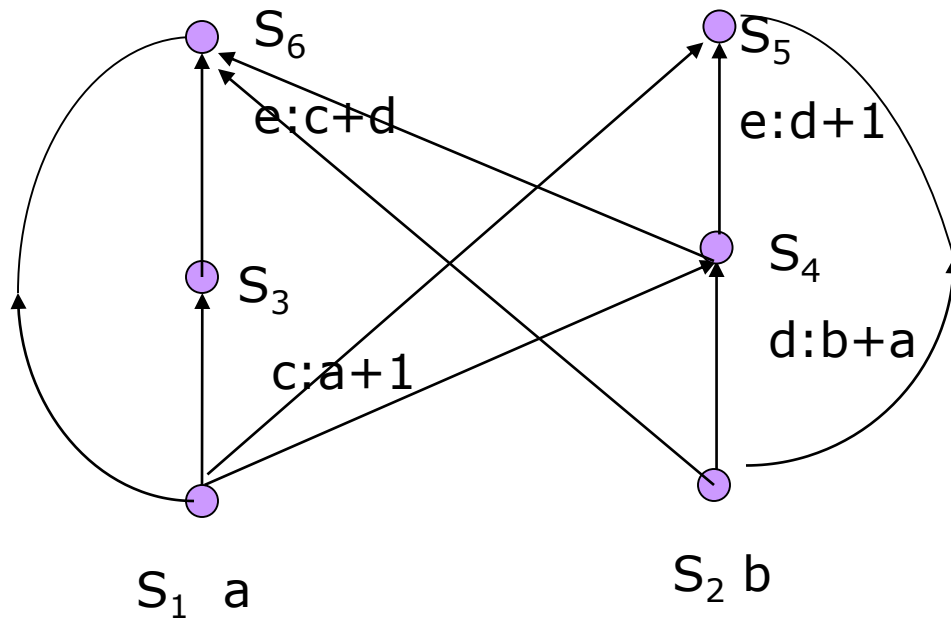
Influence Graf: Grup çalışmalarında, grup içerisindeki kişilerin birbirlerini etkilemesini modellemede kullanılır.

Round-Robin Tournament Graf: Turnuvada yer alan her takımın, hangi takımla karşılaştığını ve oyunu kimin kazandığını göstermede kullanılır.

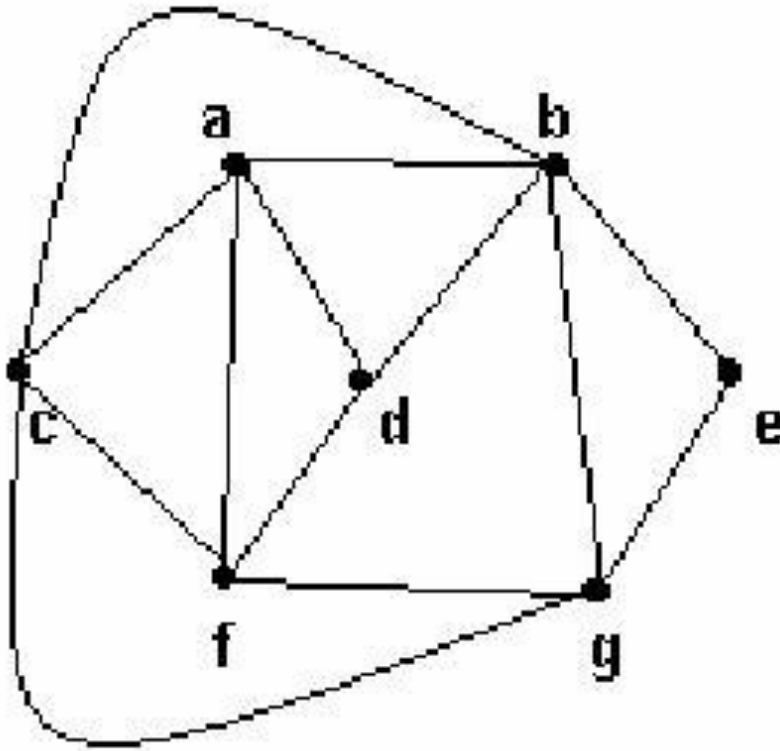
Precedence Graf: Bir işlemin sonucu, kendisinden önce gelen işlemin sonucuna bağlı olarak değişen sistemleri modellemede kullanılır.

Precedence grafa örnek....

S_1 **a:0** S_2 **b:1** S_3 **c:a+1** S_4 **d:b+a** S_5 **e:d+1** S_6 **e:c+d**



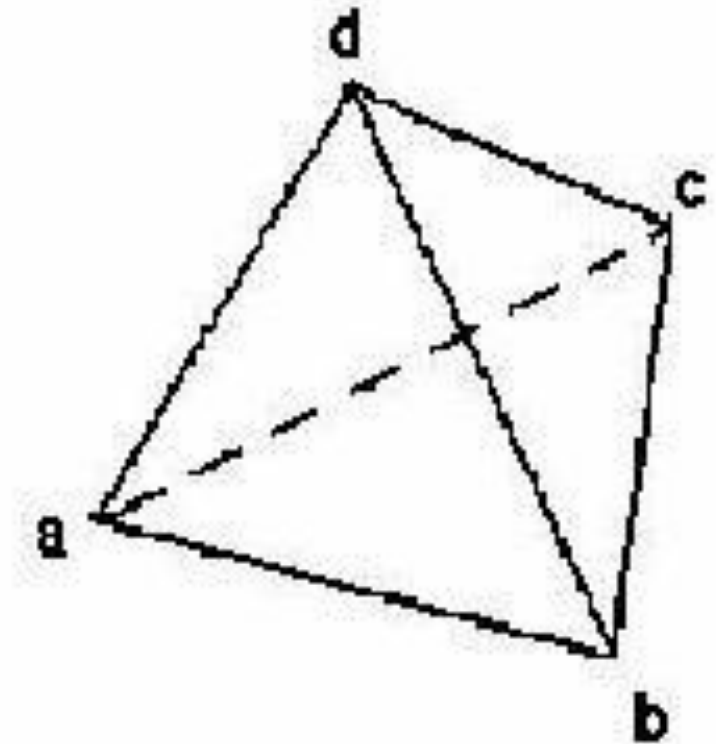
Planar Graflar



Bir G grafinın kenarları birbirlerini kesmeyecek şekilde çizilebiliyorsa ***Planar*** graf olarak adlandırılır.

Euler'in formülü

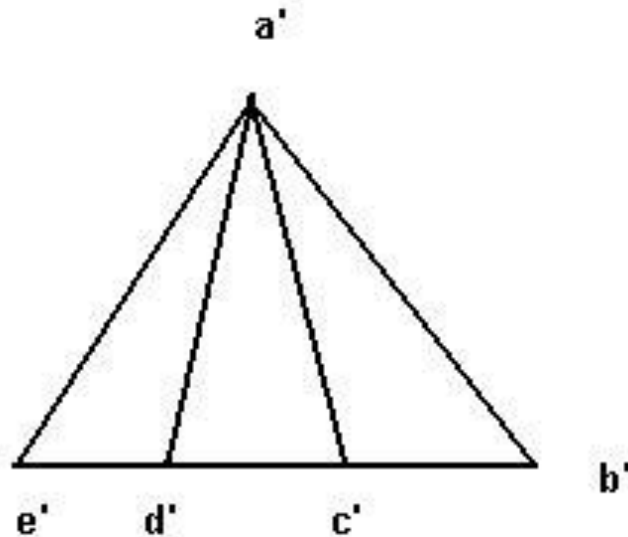
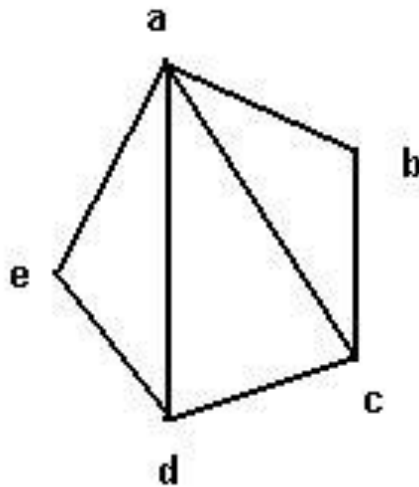
- ❑ Eğer G bir *planar* graph is
 - ❑ v = düğüm sayısı
 - ❑ e = kenar sayısı
 - ❑ f = yüzey sayısı
- ❑ Öyleyse $v - e + f = 2$



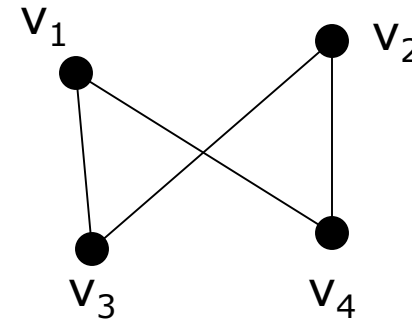
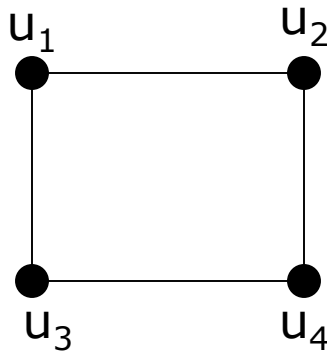
İzomorfik (Isomorphic) Graflar

İki grafin izomorfik olup olmadığı nasıl kontrol edilir ?

- ❑ Kenar sayıları aynı olmalıdır.
- ❑ Düğüm sayıları aynı olmalıdır.
- ❑ Düğüm dereceleri aynı olmalıdır.
- ❑ Dğümler arasındaki ilişkiyi gösteren matrisler aynı olmalıdır. Bu matrislerdeki benzerlik satır ve sütunlardaki yer değişikliği ile de sağlanabilir.



Örnek



Bu iki graf izomorfik midir?

EYET

Her iki grafında 4 düğümü, 4 kenarı ve her düğümünün de derecesi 2 dir

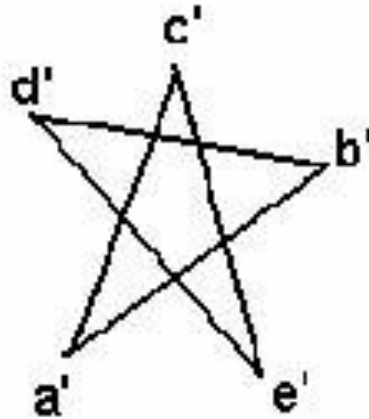
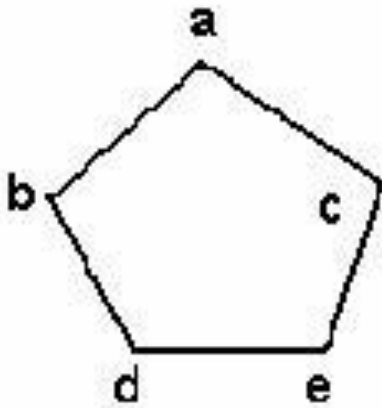
	u1	u2	u3	u4		v1	v2	v3	v4
u1	0	1	1	0	v1	0	0	1	1
u2	1	0	0	1	v2	0	0	1	1
u3	1	0	0	1	v3	1	1	0	0
u4	0	1	1	0	v4	1	1	0	0

u_2 ve u_4 satır ve sütunlar yerdeğiřtiriyor

Örnek

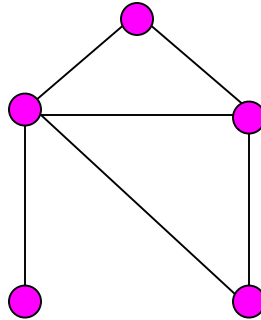
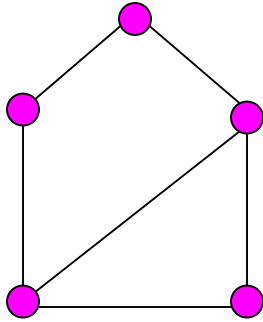
- Aşağıda verilmiş olan iki graf izomorfik midir?

EYET



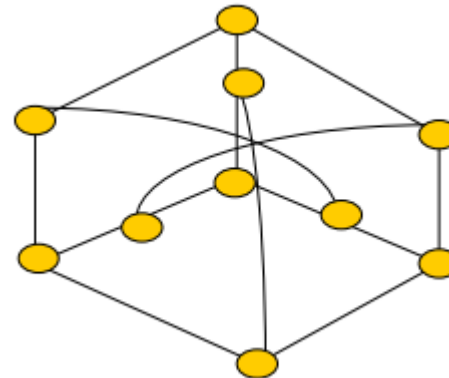
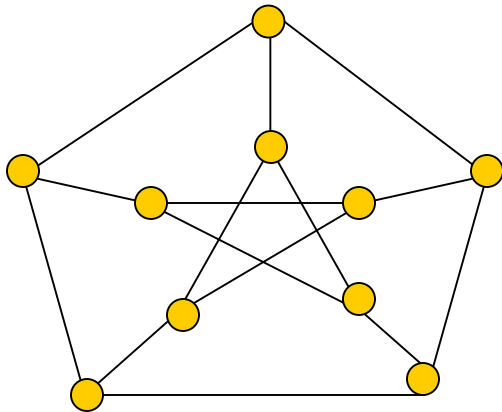
	a	b	c	d	e
a	0	1	1	0	0
b	1	0	0	1	0
c	1	0	0	0	1
d	0	1	0	0	1
e	0	0	1	1	0

Örnek



Bu iki graf izomorfik midir ?

HAYIR



Bu iki graf izomorfik midir ?

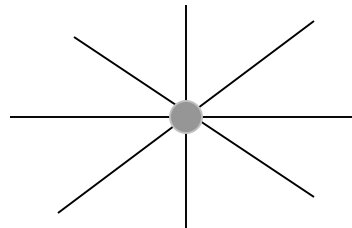
EVET

Özel Tip Graflar

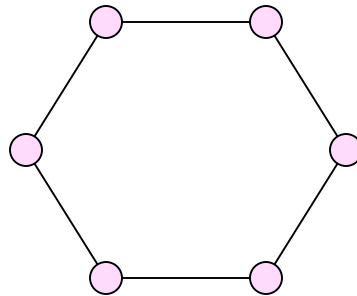
❑ Özel tip graflar genellikle veri iletişimi ve paralel veri işleme uygulamalarında kullanılır.

Local Area Network : Bir bina içerisindeki midi ve pc gibi farklı bilgisayarları ve çevrebirimlerini birbirine bağlamak için kullanılır. Bu ağların farklı topolojileri mevcuttur.

« **Star Topology :** Bütün cihazlar, merkezdeki cihaz üzerinden birbirlerine bağlanırlar. $K_{1,n}$ complete Bipartite Graf kullanılır.



« **Ring Topology** : Bu modelde her cihaz diğer iki farklı cihaz ile birbirine bağlıdır. n -cycles C_n modelidir.



« **Hybrid Topology** : Star ve Ring topology'sini birlikte kullanır. Bu tekrarlılık network'ün daha güvenli olmasını sağlar. Wheel, W_n graf modeline karşılık gelir.

