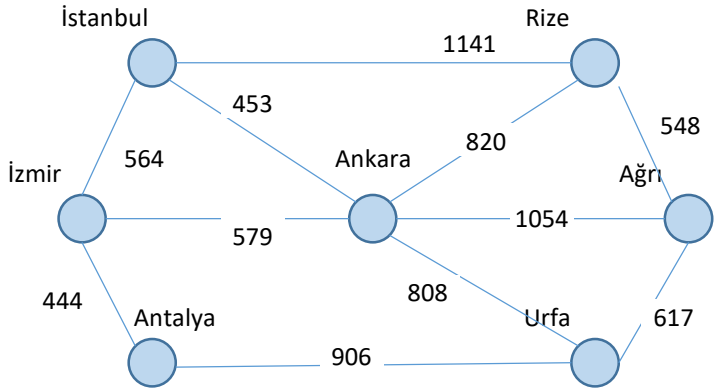
10 Graflar

10.1 Giriş

Çizge kuramının başlangıcı Königsberg'in 7 köprüsü (Kaliningrad/Rusya) problemine dayanır. 1736'da Leonhard Euler'ın söz konusu probleme ilişkin kullandığı sembol ve çizimler graf kuramına ilişkin başlangıç olarak kabul edilir. Problem, "Pregel nehri üzerinde iki ada birbirine bir köprü ile adalardan biri her iki kıyıya ikişer köprü, diğeri de her iki kıyıya birer köprü ile bağlıdır. Her hangi bir kara parçasında başlayan ve her köprüden bir kez geçerek başlangıç noktasına dönülen bir yürüyüş (walking) var mıdır?" sorusunu içeriyordu. Euler kara parçalarını düğüm, köprüleri de kenar kabul eden bir çizge elde etti. Söz konusu yürüyüşün ancak her düğümün derecesinin çift olması durumunda yapılabileceğini gösterdi. Graf kuramının bu ve buna benzer problemlerle başladığı düşünülür. Günlük yaşamdaki birçok problemi, graf kuramı uzayına taşıyarak çözebiliyoruz.

Graf (çizge), bilgisayar dünyasında ve gerçek hayatta çeşitli sebeplerle karşılaşılan bir olay veya ifadenin düğüm ve çizgiler kullanılarak gösterilmesi şeklidir. Fizik, kimya gibi temel bilimlerde, mühendislik uygulamalarında ve tıp biliminde pek çok problemin çözümü ve modellenmesi graflara dayandırılarak yapılmaktadır. Örneğin elektronik devreleri (baskı devre kartları), entegre devreleri, ulaşım ağlarını, otoyol ağını, havayolu ağını, bilgisayar ağlarını, lokal alan ağlarını, interneti, veri tabanlarını, bir haritayı veya bir karar ağacını graflar kullanarak temsil etmek mümkündür. Ayrıca planlama projelerinde, sosyal alanlarda, kimyasal bileşiklerin moleküler yapılarının araştırılmasında ve diğer pek çok alanda kullanılmaktadır. Örneğin Türkiye'de bazı illerin karayolu bağlantıları ile yolun uzunluğunu gösteren aşağıdaki yapı, yönsüz bir graftır.



Şekil 10-1. Yönsüz graf örneği.

Uygulamada karayolları ve havayolları rotaları farklıdır. Bu yüzden havayollarında yönlü graf kullanılması daha mantıklıdır. Eğer kullanımda bir simetri varsa (gidiş ve geliş yolu aynı gibi) yönsüz graf kullanmak daha iyidir.

Şekil 10-1'de verilen yapıyı grafın teknik deyimleri ile belirttiğimizde iki il arasındaki yollar **ayrıt** (**edge**), iki ayrıtın birleştiği yer **tepe** ya da **düğüm** (**vertex**, **node**) olarak anılır. Biz bu derste herhangi bir tepeyi (düğüm) küçük harf ve o tepenin indisi ile örneğin v_4 biçiminde göstereceğiz. Grafı oluşturan tepelerin sırası önemli olmaksızın yalnız bir kez yazıldığı kümeyi de (set) büyük harf *V* ile belirteceğiz. Örneğin yukarıdaki grafta *V*'nin elemanı olan tepeler (vertices);

v_1: İstanbul
v_2: İzmir
v_3: Ankara
v_4: Antalya
v_5: Urfa
v_6: Rize
v_7: Ağrı

olarak belirlenirse,

```
V(G) = v1, v2, v3, v4, v5, v6, v7
```

Olur. Tek bir ayrıtı küçük harf e_i ile, ayrıtların oluşturduğu kümeyi (set) ise büyük harf *E* ile belirteceğiz. Her tepe bir bilgi parçasını gösterir ve her ayrıt iki bilgi arasındaki ilişkiyi gösterir. Eğer ayrıtlar üzerinde bir yön ilişkisi belirtilmemişse bu tür graflar **yönsüz graf** (**undirected graph**) adını alırlar. Yönsüz grafın ayrıtları bir parantez çifti içine o ayrıtın birleştiği tepeler yazılarak belirtilir. Örneğin (v_3 , v_6) ayrıtı, ANKARA ile RİZE arasındaki yolu belirtmektedir. Yönsüz grafta ayrıtı belirten tepelerin

sırasının değiştirilmesi bir değişiklik oluşturmaz. Öteki deyişle (v_4 , v_6) ile (v_6 , v_4) aynı ayrıttır. Ayrıca veri yapısının kolay kuruluşunu sağlamak ve veri yapısını kurarken gözden kaçan bir ayrıt olmadığını daha kolay denetlemek için indisi daha küçük olan tepe önce yazılır. Bu yazım biçimine göre yukarıdaki örnekte ayrıt listesi şöyledir;

E = { v_1 , v_2 , 564 , v_1 , v_3 , 453 , v_1 , v_6 , 1141 , v_2 , v_3 , 579 , v_2 , v_4 , 444 , v_3 , v_5 , 808 , v_3 , v_6 , 820 , v_3 , v_7 , 1054 , v_4 , v_5 , 906 , v_5 , v_7 , 617 , v_6 , v_7 , 548 }

Buradaki 564, 453, … gibi sayılar, iki şehir arasındaki uzaklık, maliyet ya da iki router arasındaki trafik, band genişliğini belirten bir ağırlıktır.

Aynen ağaçlar gibi graflar da doğrusal olmayan veri yapıları grubuna girerler. Bağlı listeler ve ağaçlar grafların özel örneklerindendir. Bir grafta düğümler dairelerle, ayrıtlar da çizgilerle gösterilir.

V = { v_0 , v_1 , v_2 , v_3 , v_4 , … v_{n-1}, v_n }	Tepeler (vertices)
E = { e_0 , e_1 , e_2 , e_3 , e_4 , … e_{m-1}, e_m }	Ayrıtlar (edges)
G = {V , E}	Graf (graph)

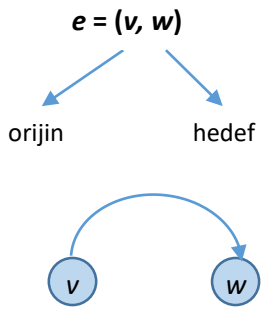
Bu anlatılanlardan sonra bilimsel olarak grafı şu şekilde tanımlayacağız; Bir G grafı, tepeler olarak adlandırılan boş olmayan bir $V(G)$ sonlu nesneler kümesi ile G'nin farklı tepe çiftlerinin düzensiz sıralanışı olan ve V'nin tepelerini birleştiren bir $E(G)$ (boş olabilir) ayrıtlar kümesinden oluşur ve $G=(V,E)$ şeklinde gösterilir. Buradaki ayrıtlar G'nin ayrıtları diye adlandırılır. G grafının tepeler kümesinin (V G= v_1, v_2, … , v_n) eleman sayısına 'nin sıralanışı (order) denir ve | V G| = n olarak gösterilir. Diğer taraftan ayrıtlar kümesinin (E G = e_0, e_1, … , e_m) eleman sayısına boyut (size) denir ve |E G | = m olarak gösterilir.

Grafları ayrıtların yönlü olup olmamasına göre **yönlü graflar** ve **yönsüz graflar** olarak ikiye ayırmak mümkündür. Ayrıca ayrıtların ağırlık almasına göre **ağırlıklı graflar** veya **ağırlıksız graflar** isimleri verilebilir.

10.2 Terminoloji, Temel Tanımlar ve Kavramlar

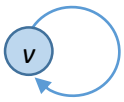
Yönsüz Ayrıt (Undirected Edge): Çizgi şeklinde yönü belirtilmeyen ayrıtlar yönsüz ayrıtlardır. Sırasız node çiftleriyle ifade edilir. **(*v*, *w*)** ile **(*w*, *v*)** olması arasında fark yoktur.

Yönlü Ayrıt (Directed Edge / digraph): Ok şeklinde gösterilen ayrıtlar yönlü ayrıtlardır. Sıralı node çiftleriyle ifade edilir. Birinci node **orijin**, ikinci node ise **hedef** olarak adlandırılır. Örneğin Şekil 10-2'de **(*v*, *w*)** ile **(*w*, *v*)** aynı değildir.



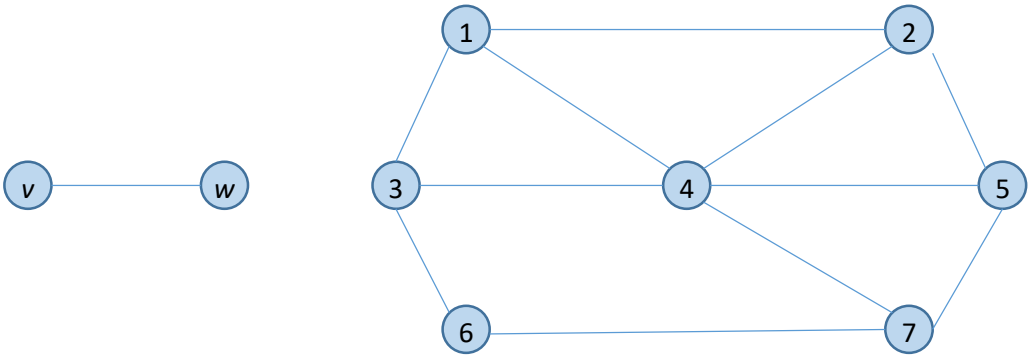
Şekil 10-2. Sıralı tepe çifti.

Self Ayrıt (Döngü –Loop): (*v*, *v*) şeklinde gösterilen ve bir tepeyi kendine bağlayan ayrıttır.



Şekil 10-3. Self ayrıt.

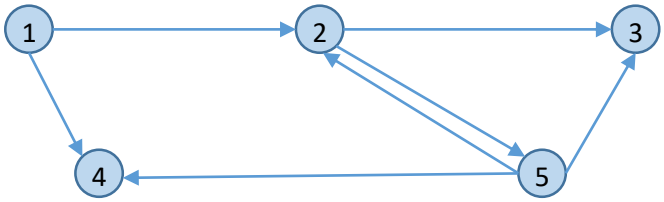
Yönsüz Graf (Undirected Graph): Tüm ayrıtları yönsüz olan grafa yönsüz graf denilir. Yönsüz grafta bir tepe çifti arasında en fazla bir ayrıt olabilir. ***e*** sırasız bir çifttir. Bir ayrıtın çoklu kopyalarına izin verilmez. Ağaçlar bir graftır fakat her graf bir ağaç değildir. ***e* = *v*, *w* = (*w*, *v*)**



Şekil 10-4. Sırasız çift ve yönsüz graf örneği.

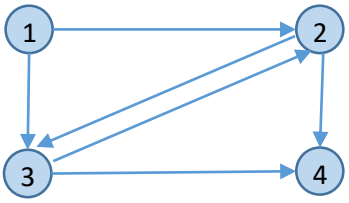
Yönlü Graf (Directed Graph, Digraph): Tüm ayrıtları yönlü olan grafa yönlü graf adı verilir. Yönlü grafta bir tepe çifti arasında ters yönlerde olmak üzere en fazla iki ayrıt olabilir. ***e*** sıralı bir çifttir. Her ***e*** ayrıtı bazı ***v*** tepesinden diğer bazı ***w*** tepesine yönlendirilmiştir. Yönsüz graflar, yönlü graf olabilir ama yönlü graf yönsüz graf olamaz.

Yönlü graf tek yönde haraketli olduğu halde yönsüz graf her iki yönde de hareket eder.



Şekil 10-5. Yönlü graf örneği.

Komşu Tepeler (Adjacent): Aralarında doğrudan bağlantı (ayrıt) bulunan ***v*** ve ***w*** tepeleri komşudur. Diğer tepe çiftleri komşu değildir. ***e*** ayrıtı hem ***v*** hem de ***w*** tepeleriyle bitişiktir (incident) denilir.



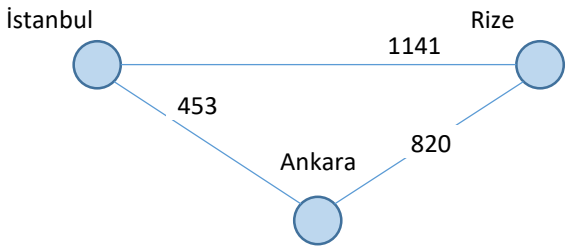
Tepe	Komşuları
1	{ 2, 3 }
2	{ 3, 4 }
3	{ 2, 4 }
4	{ }

Şekil 10-6. Komşuluk tablosu (adjacency table).

Komşuluk ve Bitişiklik: Bir *G* grafı komşuluk ilişkisiyle gösteriliyorsa G_{vv}= v_i, v_j ... bitişiklik ilişkisiyle gösteriliyorsa G_{ve}= v_i, e_j ... şeklinde yazılır. (G_{vv}: Gtepe_tepe, G_{ve}: Gtepe_ayrıt)

Ağırlıklı Graf (Weighted Graph): Her ayrıta bir **ağırlık** (**weight**) veya **maliyet** (**cost**) değerinin atandığı graftır. Örneğin Şekil 10-7'deki graf ağırlıklı graftır.

Yol (Path): *G*(*V*, *E*) grafında i_1 ve i_k tepeleri arasında $P = i_1, i_2, \ldots, i_k$ şeklinde belirtilen tepelerin bir dizisidir (*E*'de, $1 <= j < k$ olmak üzere, her *j* için (i_j , i_{j+1}) şeklinde gösterilen bir ayrıt varsa). Eğer graf yönlü ise yolun yönü ayrıtlar ile hizalanmalıdır. Şekil 10-7'deki grafta İstanbul'dan Ankara'ya doğrudan veya Rize'den geçerek gidilebilir.



Şekil 10-7. Ağırlıklı bir graf.

Basit Yol (Simple Path): Tüm düğümlerin farklı olduğu yoldur.

Çevre: Her bir iç tepesinin derecesi iki olan ***n*** ayrıtlı kapalı ayrıt katarına çevre denir ve bu da C_n ile gösterilir.

Uzunluk: Bir yol üzerindeki ayrıtların sayısı o yolun uzunluğudur.

Bir Tepenin Derecesi: Yönsüz bir grafın bir ***v*** tepesine bağlı olan ayrıtlarının sayısına ***v***'nin derecesi denir ve bu `degree(v)` ile gösterilir. Self-ayrıtlar `1` olarak sayılır.

Yönlü graflarda iki çeşit derece vardır. Bunlar;

Giriş Dercesi (Indegree): Yönlü grafta, bir tepeye gelen ayrıtların sayısına indegree denir.

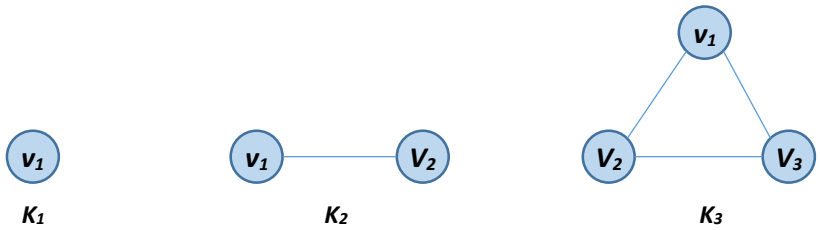
Çıkış Derecesi (Outdegree): Yönlü grafta, bir tepeden çıkan ayrıtların sayısına outdegree denilir. Örneğin Şekil 10-6'teki yönlü grafta 4 nolu tepenin giriş derecesi 2, çıkış derecesi ise 0'dır.

Bağlı Graf (Connected Graph): Her tepe çifti arasında en az bir yol olan graftır.

Alt Graf (Subgraph): ***H*** grafının tepe ve ayrıtları ***G*** grafının tepe ve ayrıtlarının alt kümesi ise ***H*** grafı ***G*** grafının alt grafıdır.

Ağaç (Tree): Çevre içermeyen yönsüz bağlı graftır.

Tam Graf: Birbirinden farklı her tepe çifti arasında bir ayrıt bulunan graflardır. ***n*** tepeli bir tam graf K_n ile gösterilir ve `n*(n-1)/2` tane ayrıtı vardır (kendilerine bağlantı yok). Şekil 10-8'de K_1, K_2 ve K_3 tam grafları gösterilmiştir.



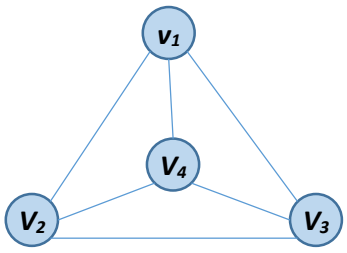
Şekil 10-8. K_1 , K_2 ve K_3 tam grafları.

Tam Yönlü Graf (Complete Digraph): ***n*** tepe sayısı olmak üzere `n.(n-1)` ayrıtı olan graftır.

Seyrek Graf: Ayrıt sayısı mümkün olandan çok çok az olan graftır.

Katlı Ayrıt: Herhangi iki tepe çifti arasında birden fazla ayrıt varsa, böyle ayrıtlara katlı ayrıt denir.

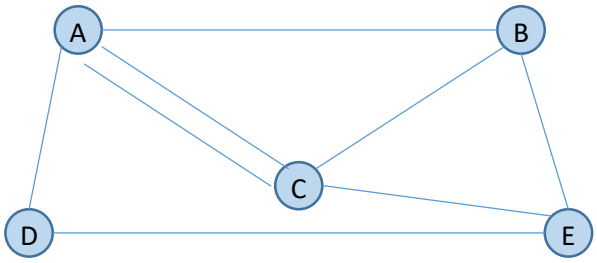
Tekerlek (Wheel) Graf: Bir cycle grafına ek bir tepe eklenerek oluşturulan ve eklenen yeni tepenin, diğer bütün tepelere bağlı olduğu graftır. W_n ile gösterilir.



Şekil 10-9. Tekerlek Graf.

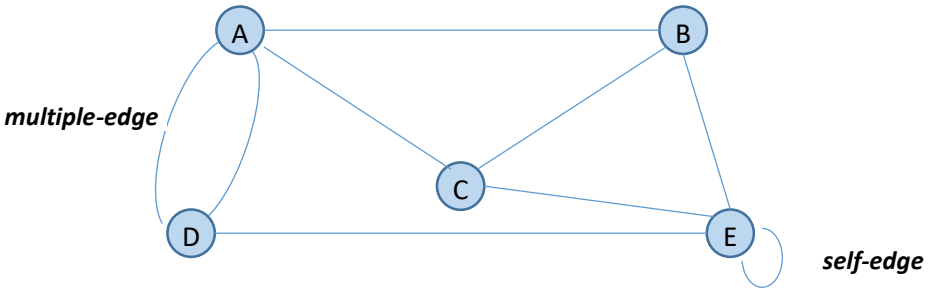
Daire veya Devir (Cycle): Başlangıç ve bitiş tepeleri aynı olan basit yoldur. Şekil 10-7'deki İstanbul-Rize-Ankara-İstanbul örneği.

Paralel Ayrıtlar (Parallel Edges): İki veya daha fazla ayrıt bir tepe çifti ile bağlanmıştır. Şekil 10-10'da **A** ve **C** iki paralel ayrıt ile birleşmiştir.



Şekil 10-10. Paralel ayrıtlı bir graf.

Çoklu (Multi) Graf: Multigraf iki tepe arasında birden fazla ayrıta sahip olan veya bir tepenin kendi kendisini gösteren ayrıta sahip olan graftır.



Şekil 10-11. Çoklu (multi) graf.

Spanning Tree: ***G***'nin tüm tepelerini içeren bir ağaç şeklindeki alt graflardan her birine spanning tree denir.

Forest: Bağlı olmayan ağaçlar topluluğudur.

10.3 Grafların Bellek Üzerinde Tutulması

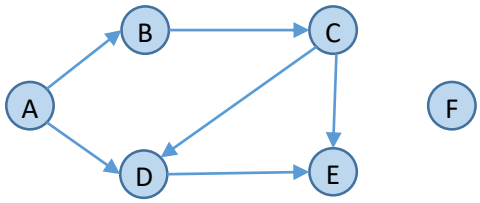
Grafların bellekte tutulması veya gösterilmesi ile ilgili birçok yöntem vardır. Bunlardan birisi, belki de en yalın olanı, doğrudan matris şeklinde tutmaktır; grafın komşuluk matrisi elde edilmişse, bu matris doğrudan programlama dilinin matris bildirim özellikleri uyarınca bildirilip kullanılabilir. Ancak grafın durumu ve uygulamanın gereksinimine göre diğer yöntemler daha iyi sonuç verebilir. Bir ***G***= (***V***, ***E***) grafının bilgisayara aktarılmasındaki en yaygın iki standart uygulama şunlardır:

- Komşuluk matrisi ile ardışık gösterim (adjacency matrix representation),
- Bağlı listeler ile bağlı gösterim veya komşuluk yapısı ile gösterim (adjacency list representation).

Bu uygulamalar hem yönlü graflar hem de yönsüz grafların her ikisinde de geçerlidir. Genelde, yoğun graflar (|***E***|'nin |$\boldsymbol{V^2}$|'ye yakın olduğu graflar) için ya da iki tepeyi birbirine bağlayan bir ayrıtın varlığını hızlı bir şekilde belirlemek için matrisler kullanılır. Seyrek graflar (|***E***|'nin |$\boldsymbol{V^2}$|'den çok daha az olduğu graflar) için ise bağlı listeler kullanılır.

10.3.1 Komşuluk Matrisi (Adjacency Matrix)

Tepelerden tepelere olan bağlantıyı `boolean` değerlerle ifade eden bir kare matristir. İki boyutlu bir dizi ile implemente edilir. ***G***(***V***, ***E***), |***v***| düğümlü bir graf ise |***v***| x |***v***| boyutunda bir matris ile bir grafı bellek ortamında gerçekleştirilebilir. Eğer (***A***, ***B***) ∈ ***E*** ise `1`, diğer durumlarda ise `0` olarak işaretlenir. Yönsüz bir grafta matris simetrik olacağından dolayı bellek tasarrufu amacıyla alt yarı üçgen ya da üst yarı üçgen kullanmak yerinde olur. Her iki durumda da gerçekleştirimin bellek karmaşıklığı ***O***(|$\boldsymbol{v^2}$|) olur.

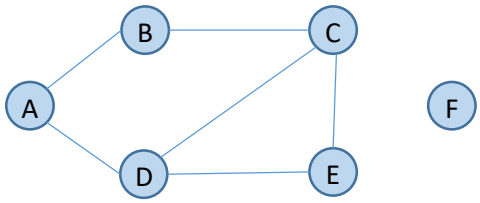


hedef →

orijin ↓	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	0	0	1	1	0
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Şekil 10-12. Yönlü bir grafın komşuluk matrisi ile gösterimi.

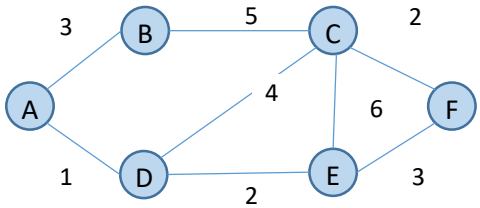
Yönlü grafların matrisi genelde simetrik olmaz fakat yönsüz graflarda bir simetriklik söz konusudur. Şimdi Şekil 10-13'te yönsüz bir grafın komşuluk matrisiyle gösterimi yer almaktadır.



	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	1	1	0
D	1	0	1	0	1	0
E	0	0	1	1	0	0
F	0	0	0	0	0	0

Şekil 10-13. Yönsüz bir grafın komşuluk matrisi ile gösterimi.

Aynı ayrıt bilgilerinin, örneğin (***A***, ***B***) ve (***B***, ***A***) ayrıtlarının her ikisinin birden depolanması biraz gereksiz gibi görünebilir fakat ne yazık ki bunun kolay bir yolu yoktur. Çünkü ayrıtlar yönsüzdür ve **parent** ve **child** kavramları da yokturŞekil 10-14'te ağırlıklı ancak yönlendirilmemiş bir grafın komşuluk matrisi gösterilmiştir.



	A	B	C	D	E	F
A	0	3	0	1	0	0
B	3	0	5	0	0	0
C	0	5	0	4	6	2
D	1	0	4	0	2	0
E	0	0	6	2	0	3
F	0	0	2	0	3	0

Şekil 10-14. Yönlendirilmemiş ağırlıklı bir grafın komşuluk matrisi ile gösterimi.

Yukarıdaki şekillerde komşuluk matrisleri ile gösterilen grafların bilgisayardaki uygulamasını `int a[6][6];` şeklinde iki boyutlu bir dizi ile gerçekleştirebiliriz.

Örnek 26: Komşuluk matrisi ile temsil edilen yönlü bir grafta bir düğümün giriş ve çıkış derecelerini bulan fonksiyonu yazalım. Grafta `6` adet tepe olduğu varsayılsın. Bir düğümün derecesini bulmak için ilgili satır ya da sütundaki `1`'ler sayılmalıdır (***O***(***n***)). Satırdaki `1`'ler `outdegree`, sütundakiler ise `indegree` değerini verir. Bu ikisinin toplamı ise o düğümün derecesidir (***O***(2***n***)).

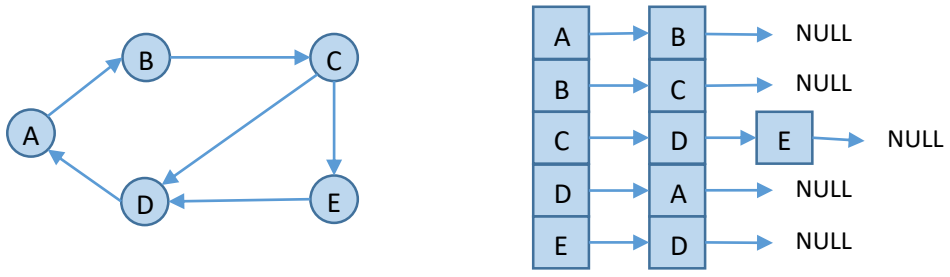
```
#define tepe_sayisi 6 // tepe sayısının 6 olduğu varsayılıyor
/* Yönlü bir grafta giriş derecesini veren fonksiyon */
int indegree(int a[][tepe_sayisi], int v) {
     int i, degree = 0;
     for(i = 0; i < tepe_sayisi; i++)
          degree += a[i][v]; // v sütunundaki 1'ler toplanıyor
     return degree; // giriş derecesi (indegree) geri döndürülüyor
}
```

Çıkış derecesini bulan fonksiyon için yalnızca a[i][v] yerine a[v][i] yazmak yeterlidir.

```
/* Yönlü bir grafta çıkış derecesini veren fonksiyon */
int outdegree(int a[][tepe_sayisi], int v) {
     int i, degree = 0;
     for(i = 0; i < tepe_sayisi; i++)
          degree += a[v][i]; // v satırındaki 1'ler toplanıyor
     return degree; // çıkış derecesi (outdegree) geri döndürülüyor
}
```

10.3.2 Komşuluk Listesi (Adjacency List)

Her ***v*** tepesi kendinden çıkan ayrıtları gösteren bir bağlı listeye sahiptir ve her bir tepe için komşu tepelerin listesi tutulur. Bellek gereksinimi ***O*** $|V| + |E|$ → dizi boyutu + ayrıt sayısı şeklindedir. Komşuluk listesi seyrek graflar için hem zaman hem de bellek açısından daha verimlidir fakat tam bir graf veya yoğun graflar için verim daha azdır.



Şekil 10-15. Yönlü bir grafın komşuluk listesi ile gösterimi.

Komşuluk listesi için veri yapısı aşağıdaki gibi tanımlanabilir;

```
#define tepe_sayisi 6 // tepe sayısının 6 olduğu varsayılıyor
```

```
struct vertex {
     int node;
     struct vertex *nextVertex;
};
struct vertex *head[tepe_sayisi]; // 6 boyutlu, bir bağlı liste başı
```

10.3.3 Komşuluk Matrisleri ve Komşuluk Listelerinin Avantajları-Dezavantajları

Komşuluk matrisi

- Çok fazla alana ihtiyaç duyar. Daha az hafıza gereksinimi için seyrek matris teknikleri kullanılmalıdır.
- Herhangi iki düğümün komşu olup olmadığına çok kısa sürede karar verilebilir.

Komşuluk listesi

- Bir düğümün tüm komşularına hızlı bir şekilde ulaşılır.
- Daha az alana ihtiyaç duyar.
- Oluşturulması matrise göre daha zor olabilir.