9.Hafta
Veri sıkıştırma ve Aç gözlü algoritmalar

Veri Sıkıştırma (Compression)

Kayıplı-Kayıpsız Veri Sıkıştırma

Sabit ve Değişken Genişlikli Kodlama

Huffman Algortiması (Greedy Algoithms)

Veri Sıkıştırma

- Veri sıkıştırma;
- **1- Alan gereksinimini azaltmak için**
  - Hard disklerin boyutu büyümekle birlikte, yeni programların ve data dosyalarının boyutu da büyümektedir ve alan ihtiyacı artmaktadır.
- **2-Zaman ve bant genişliği gereksinimini azaltmak için**
  - Birçok program ve dosya internetten download edilmektedir.
  - Birçok kişi düşük hızlı bağlantıyı kullanmaktadır.
  - Sıkıştırılmış dosyalar transfer süresini kısaltmakta ve bir çok kişinin aynı server'ı kullanımına izin vermektedir.

Kayıplı ve Kayıpsız Sıkıştırma

- **Kayıplı Sıkıştırma**
  - Bilginin bir kısmı geri elde edilemez (MP3, JPEG, MPEG)
  - Genellikle ses ve görüntü uygulamalarında kullanılır. Çok büyük ses ve görüntü dosyalarında çok iyi sıkıştırma yapar ve kullanıcı kalitedeki azalmayı fark etmez.
- **Kayıpsız Sıkıştırma**
  - Orijinal dosyalar tekrar ve tam olarak elde edilir (D(C(X)))=X, burada C sıkıştırılan dosyayı, D ise açılan dosyayı ifade eder.
  - .txt, .exe, .doc, .xls, .java, .cpp vs. gibi dosyalarda kullanılır.
  - Ses ve görüntü dosyalarında da kullanılabilir ancak kayıplı sıkıştırma kadar yüksek sıkıştırma oranına sahip olmaz.

Kayıpsız Sıkıştırma

- Kayıpsız sıkıştırma temel olarak üç algoritma altında toplanır
  - **Huffman –** her karakter için değişken genişlikli bir kelime kodu (codeword) kullanır .
  - **LZ277 –** "sliding window (kayan pencere)" kullanır . Bir grup karakteri bir anda sıkıştırır.
  - **LZ278/LZW** – daha önce karşılaşılan işaretleri saklayan bir sözlük kullanır ve bir grup karakteri aynı anda sıkıştırır.
- Kayıpsız sıkıştırma uygulamaları
  - Unix compress, gif : LZW
  - pkzip, zip, winzip, gzip: Huffman+LZ277
- ASCII, 8 bit
- 29 farklı karakter 5 bit ile kodlanabilir.
- 5/8 = %62,5 sıkıştırma oranı, kazanç %37,5 tir

Huffman Kodlama

- Renksiz ve özelliksiz karakterlerden oluşan dosyaların sıkıştırılması için kullanılan kodlardır. Dosyadaki bir karakterin tekrar sayısına o karakterin frekansı denir ve karakterlerin tekrar frekansından faydalanarak ikili kodlar üretilir.
- İki tip kodlama vardır: Sabit uzunluklu ve değişken uzunluklu.
- Toplam 1.000.000.000 karakterden oluşan bilginin sadece 6 harften {a, b, c, d, e, f} oluştuğunu varsayalım. Karakterlerin kullanım sıklıkları:

a	*b*	*c*	*d*	*e*	*f*	
.45	.13	.12	.16	.09	.05	Frekans
000	001	010	011	100	101	Fixed length
0	101	100	111	1101	1100	Variable length

Fixed length $3 \bullet 1.000.000.000 = 3.000.000.000$ bits $\approx$ **3GB**

Variable length $(.45 \bullet 1 + .13 \bullet 3 + .12 \bullet 3 + .16 \bullet 3 + .09 \bullet 3 + .05 \bullet 3) \bullet 1.000.000.000$
$= 2.240.000.000$ bits $\approx$ **2.24GB**

Huffman Kodlama

- Değişken uzunluklu kodlamada, frekansı en yüksek olan karaktere en kısa kod ve diğer karakterlere benzer mantıkla kod verilirse, sabit uzunluklu kodlamaya göre daha başarılı bir sonuç elde edilir.
- Kodlama ve kodlamayı çözmek için ön ek (prefix) kodlamadan faydalanılır.
- Kodlama her zaman için daha kolaydır. Kodu çözmek, sabit uzunluklu kodlamada oldukça kolaydır; değişken uzunluklu kodlamada ise, biraz daha zahmetlidir.
- Sıkıştırılmış bir dosyanın kod ağacı tam dolu ikili ağaç ise, bu sıkıştırma işlemi optimaldir; aksi halde optimal değildir.
- Sabit uzunluklu kodlama optimal değildir, çünkü ağacı tam dolu ikili ağaç değildir.

Karakterlerin Kodunu Çözme

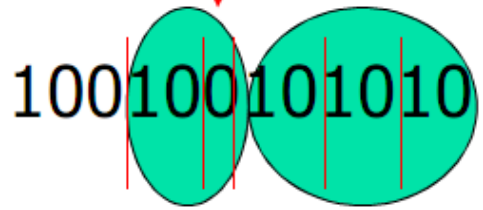
E	0
T	11
N	100
I	1010
S	1011

110100100101011011

T E N N I S

Prefix code

E	0
T	10
N	100
I	0111
S	1010



Belirsiz

Karakterlerin Kodunu Çözme

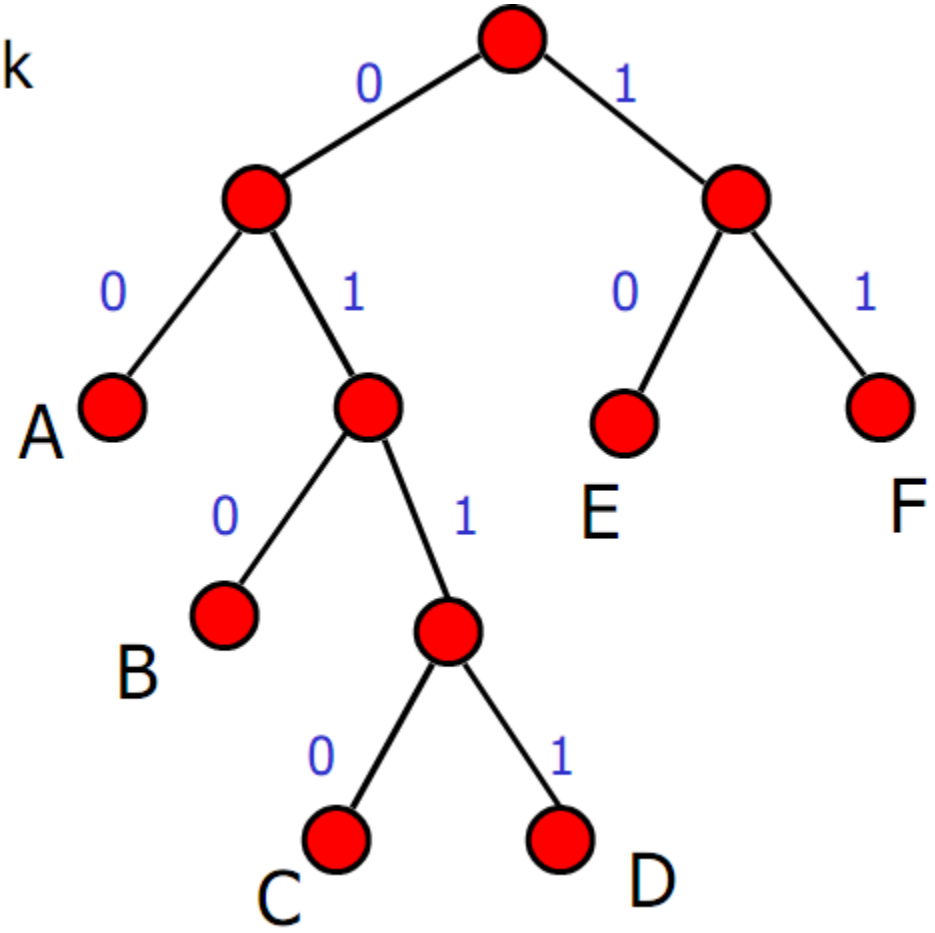
- Bir codeword başka bir codeword'ün prefix'i olamaz
- Kod çözülmesinde teklik olmalı

A	00	1	00
B	010	01	10
C	011	001	11
D	100	0001	0001
E	11	00001	11000
F	101	000001	101

Ön ek Kodları ve İkili ağaçta gösterimi

Prefix code'ların ikilik ağaçla gösterimi

A	00
B	010
C	0110
D	0111
E	10
F	11



Kodlama için gereken boyut

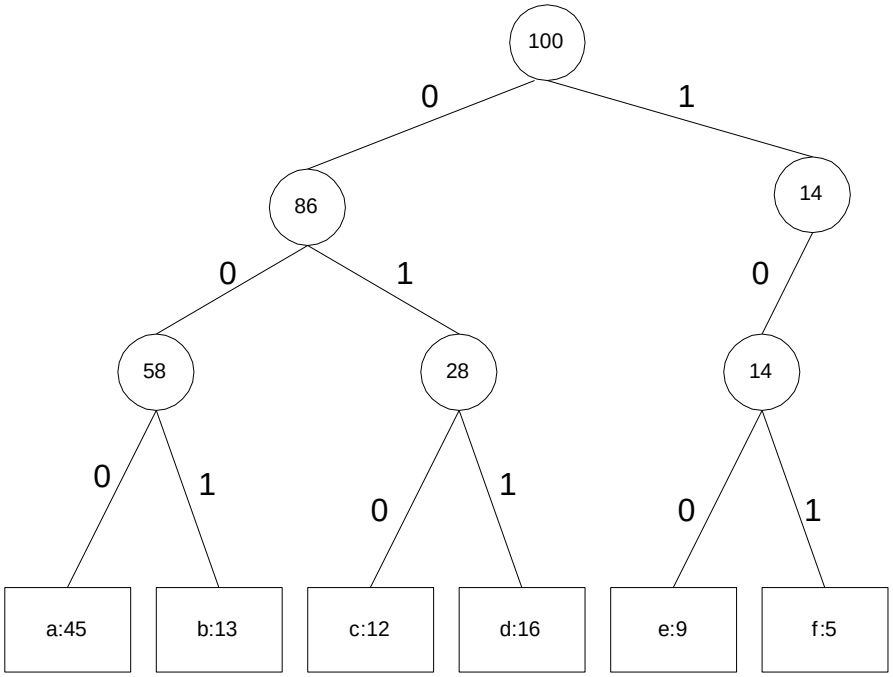
- Eğer ağaç **C** alfabesinde üretilmiş ise, bu durumda yaprak sayısının **|C|** olması gerekir ve ara düğümlerde **|C|-1** olmalıdır.
- **c∈C** olsun ve **f(c)** bu karakterin frekansını göstersin. **T** ağacı da ön ek(prefix) kod ağacı olsun. **$d_T(c)$** ise **c** karakterinin **T** ağacındaki derinliği olsun ve aynı zamanda **c** karakterinin kod uzunluğunu gösterir. Dosyanın kodlandıktan sonraki bit olarak uzunluğu B(T) aşağıdaki gibi ifade edilir

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

- olur.

Kodlama ağacının oluşturulması

- Sabit uzunlukta kodlama



Kodlama ağacının oluşturulması

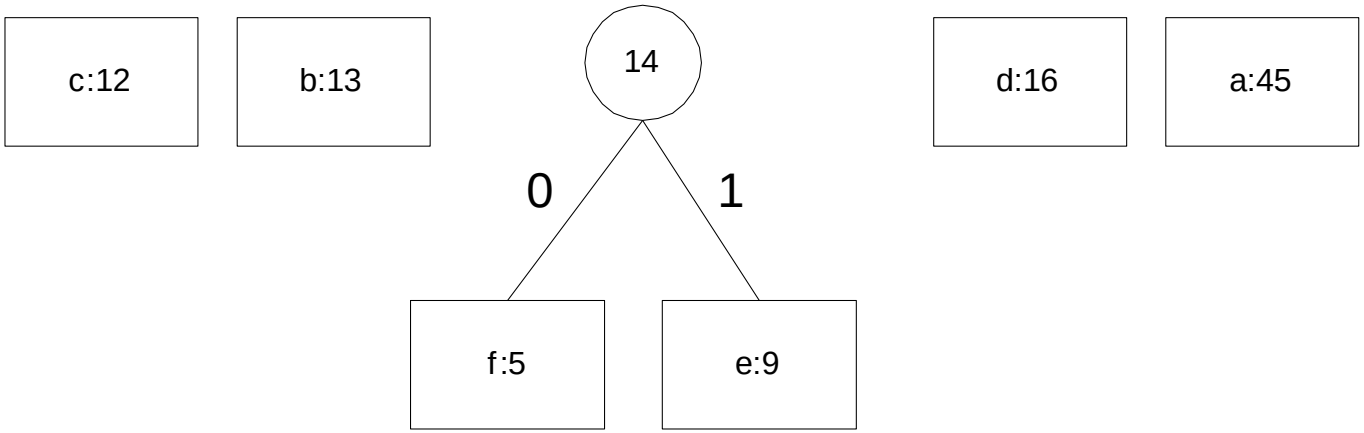
- Değişken uzunlukta kodlama: Frekansa göre sıralanır ve her biri bir yaprağı temsil eder. En küçük iki değer bir atada toplanır ve geriye kalan yapraklar ile ata tekrar sıralanır.

f:5	e:9	c:12	b:13	d:16	a:45

- *Değişken uzunluklu kodlama için frekansların sıralanması.*

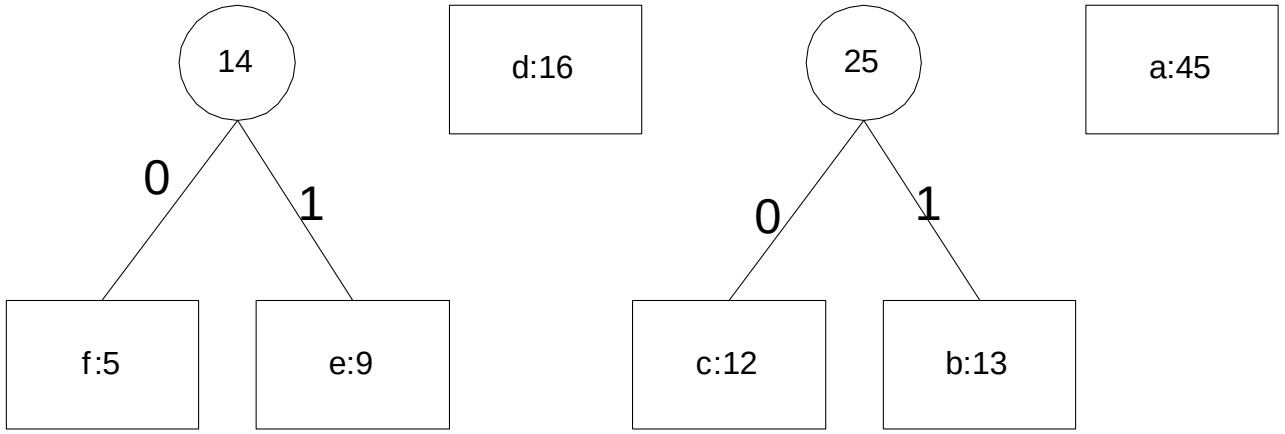
Kodlama ağacının oluşturulması

- *Değişken uzunluklu kodlama*



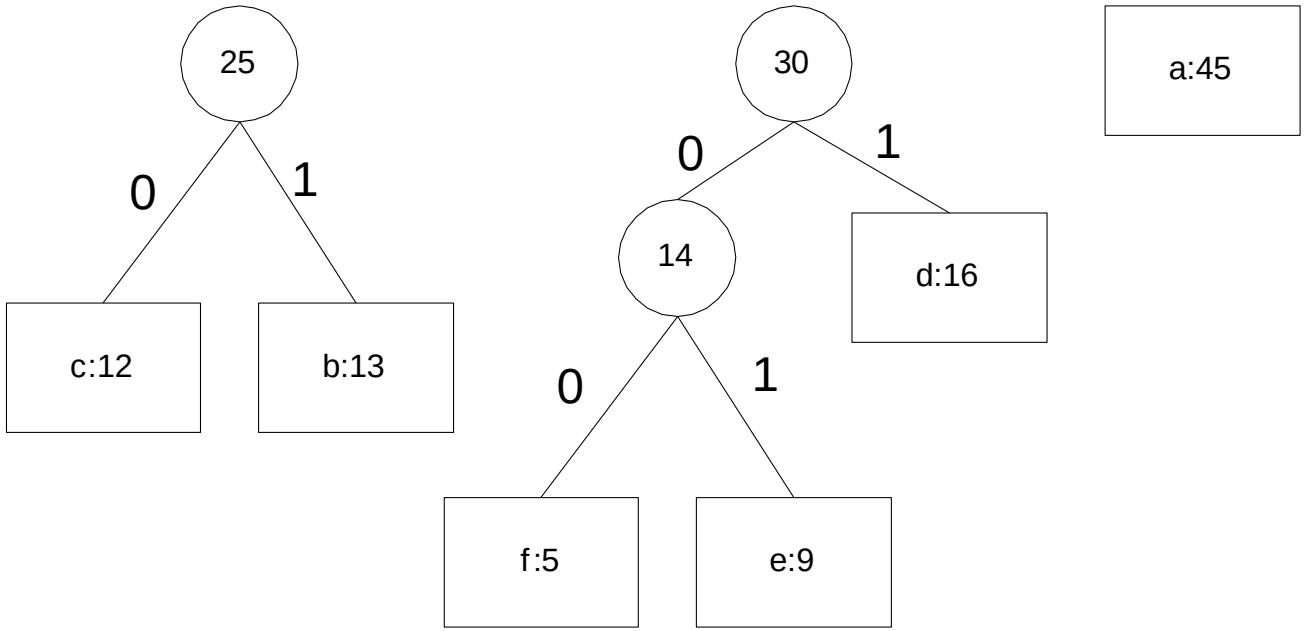
Kodlama ağacının oluşturulması

- *Değişken uzunluklu kodlama*



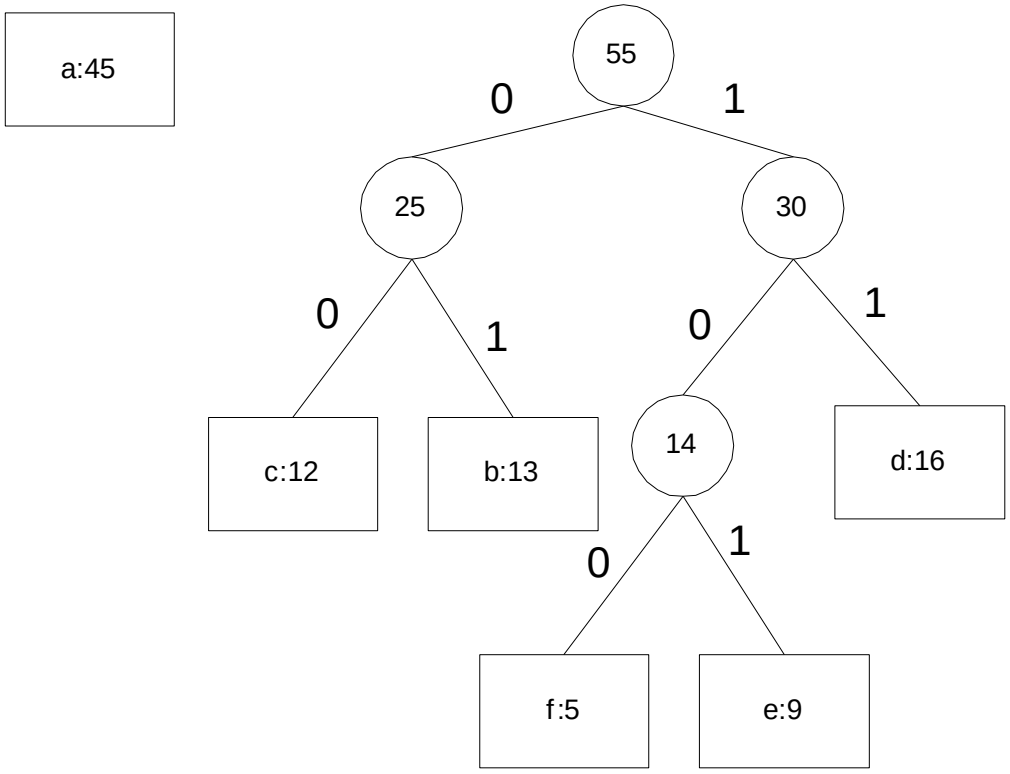
Kodlama ağacının oluşturulması

- *Değişken uzunluklu kodlama*



Kodlama ağacının oluşturulması

- *Değişken uzunluklu kodlama*



Huffman Algoritması

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      z.left = x = EXTRACT-MIN(Q)
6      z.right = y = EXTRACT-MIN(Q)
7      z.freq = x.freq + y.freq
8      INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Farklı karakter sayısı

Min-Priority Queue

En düşük frekanslı iki düğüm

İki düğümün toplamı yeni düğüm

Yeni düğümü kuyruktaki yerine ekleme

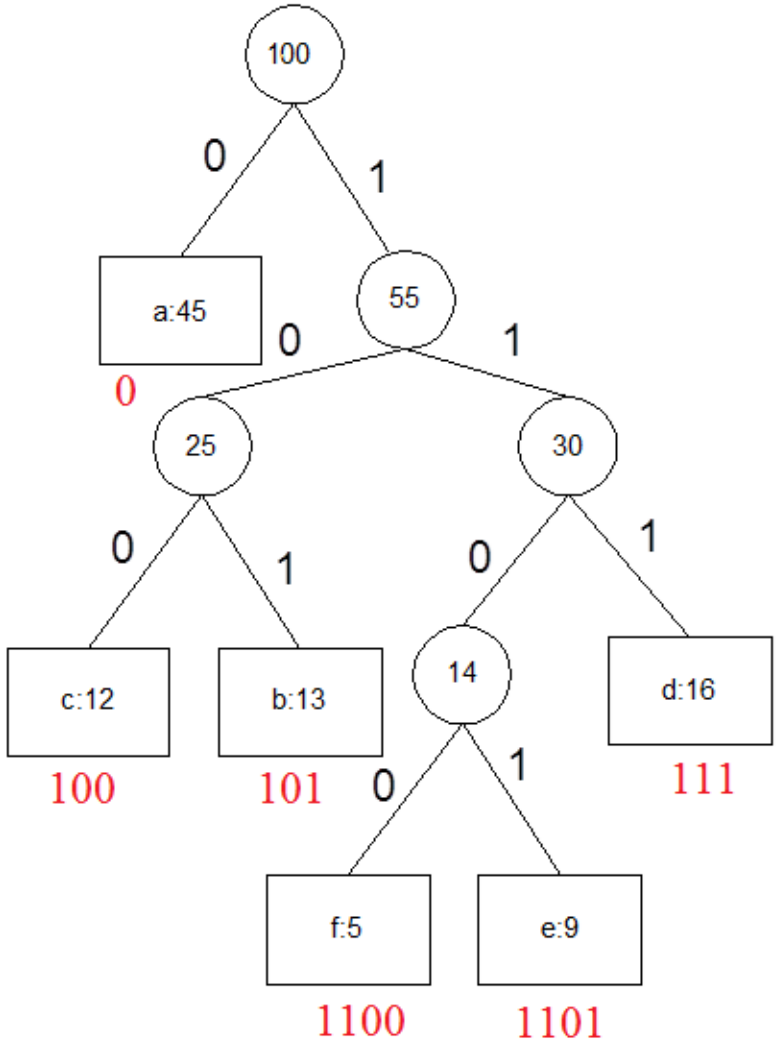
Kodlama ağacının oluşturulması ve Huffman Algoritması

- *Değişken uzunluklu kodlama*

Algoritma -Huffman(C)

▷C alfabe olmak üzere bu algoritma Huffman kodu üretir.

1. **n←|C|**
2. **Q←C**
3. **i←1, …, n-1**
4. **z ← Düğüm_Oluştur()**
5. **x ← Sol (z)←Minimum_Al(Q)**
6. **y ← Sağ(z)←Minimum_Al(Q)**
7. **f(z) ← f(x) + f(y)**
8. **Ekle(Q,Z)**
9. **Sonuç←Minimum_Al(Q)**



Huffman Algoritması Analizi

- Uygun veri yapısı olarak min-heap kullanılabilir. (Her seferinde frekansı düşük olan düğüm çıkarılır ve toplanarak ağaca tekrar eklenir)
- (http://rosettacode.org/wiki/Huffman_coding)
- Heap oluşturma O(log n) ve for döngüsünde n-1 adet işlem yapılır
- Toplam Çalışma süresi **O(nlogn)** olur.
- Huffman algoritması az sayıda karakter çeşidine sahip ve büyük boyutlardaki verilerde çok kullanışlı olabilir. Fakat oluşturulan ağacın sıkıştırılmış veriye eklenmesi zorunludur. Bu da sıkıştırma verimini düşürür. Adaptive Huffman gibi teknikler bu sorunu halletmek için geliştirilmişlerdir.

Örnek:

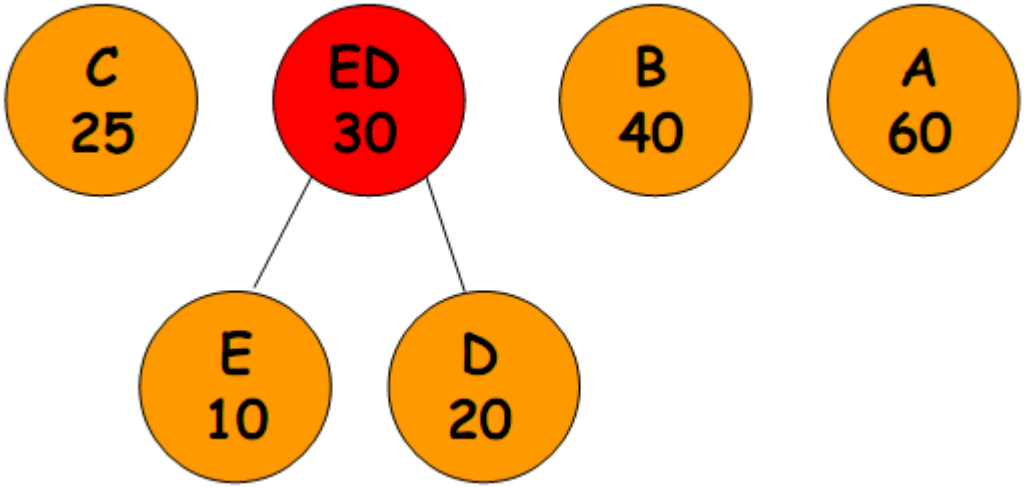
- Veri setine ait frekans tablosu aşağıdaki gibi olsun.

Sembol	Frekans
A	60
B	40
C	25
D	20
E	10

- **Adım 1:** Huffman ağacındaki en son düğümleri oluşturacak olan bütün semboller frekanslarına göre küçükten büyüğe doğru sıralanırlar.

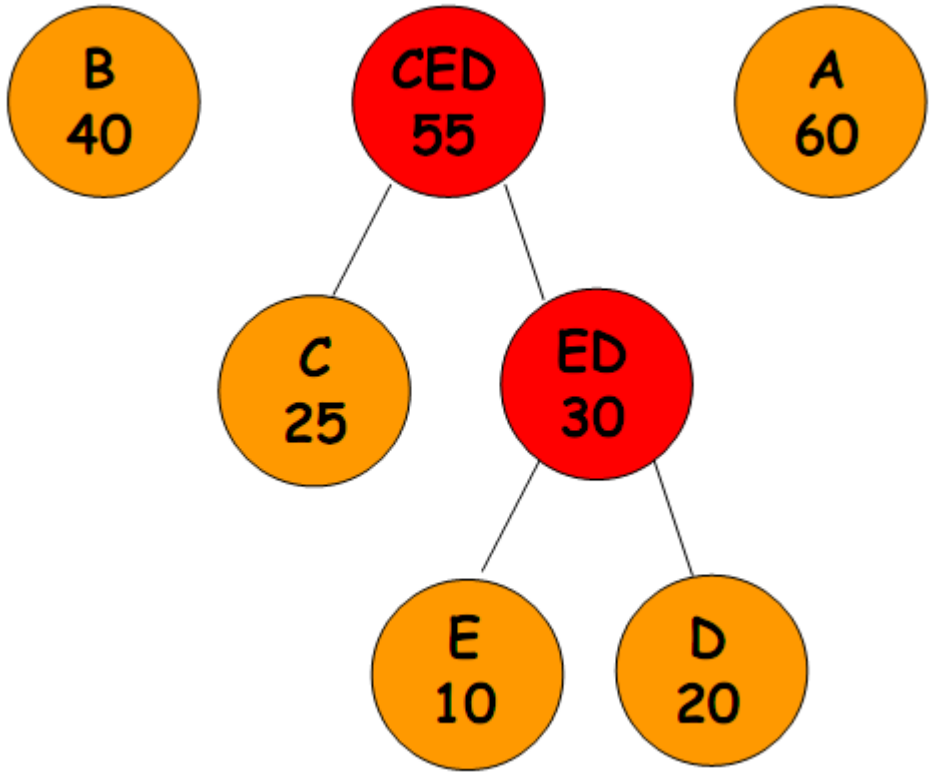
Örnek: Adım 2

- En küçük frekansa sahip olan iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm var olan düğümler arasında uygun yere yerleştirilir. Bu yerleştirme frekans bakımından küçüklük veya büyüklüğe göredir.

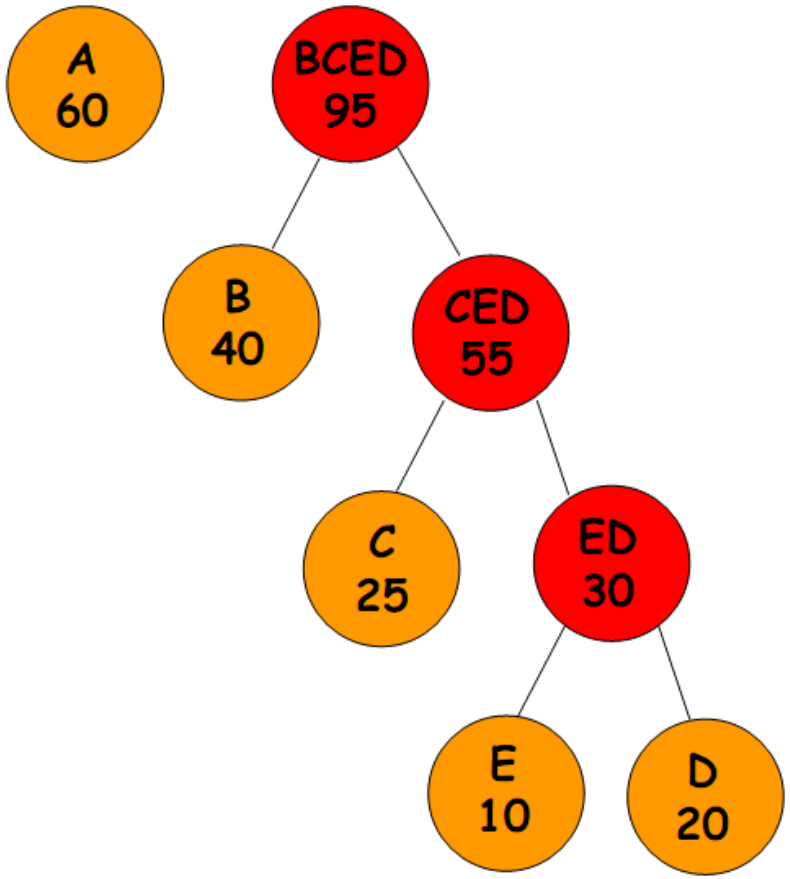


Örnek: Adım 3

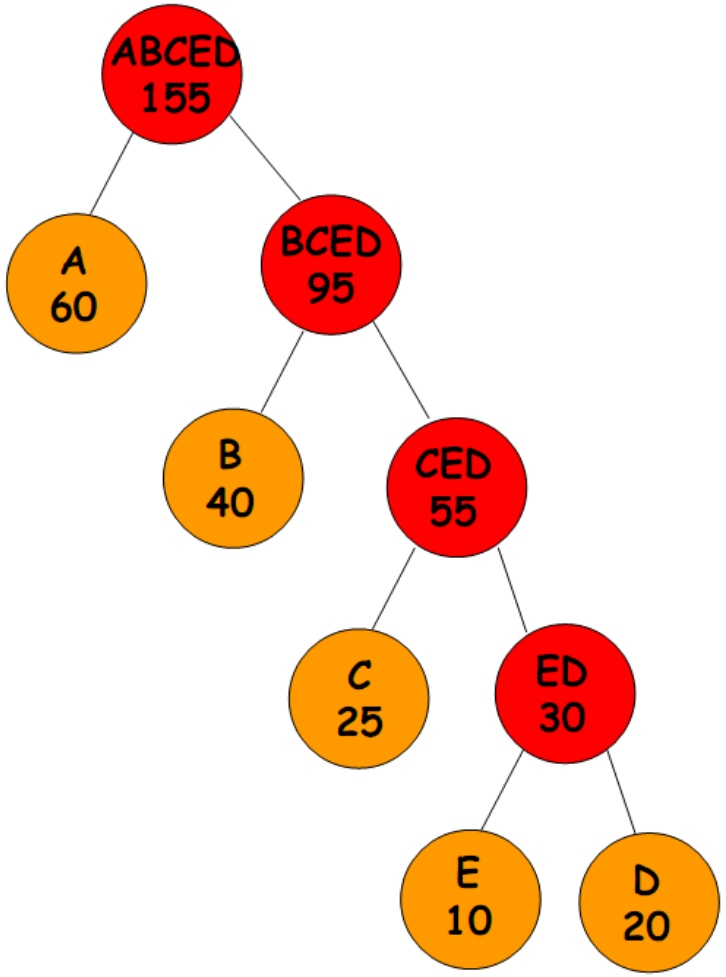
- Bir önceki adımdaki işlem terkarlanılarak en küçük frekanslı 2 düğüm tekrar toplanır ve yeni bir düğüm oluşturulur.



Örnek: Adım 4



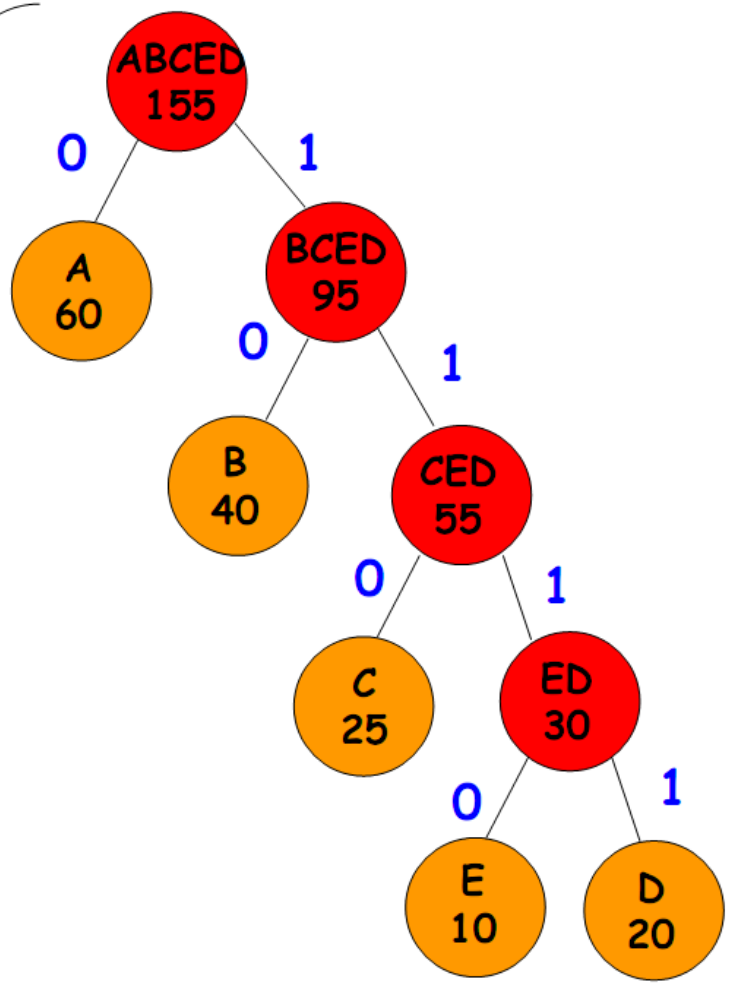
Adım 5



Örnek: Adım 6

Huffman ağacının kodu oluşturulurken ağacın en tepesindeki kök düğümden başlanır.

Kök düğümün soluna ve sağına giden dallara sırasıyla 0 ve 1 yazılır. Sırası seçimliktir.



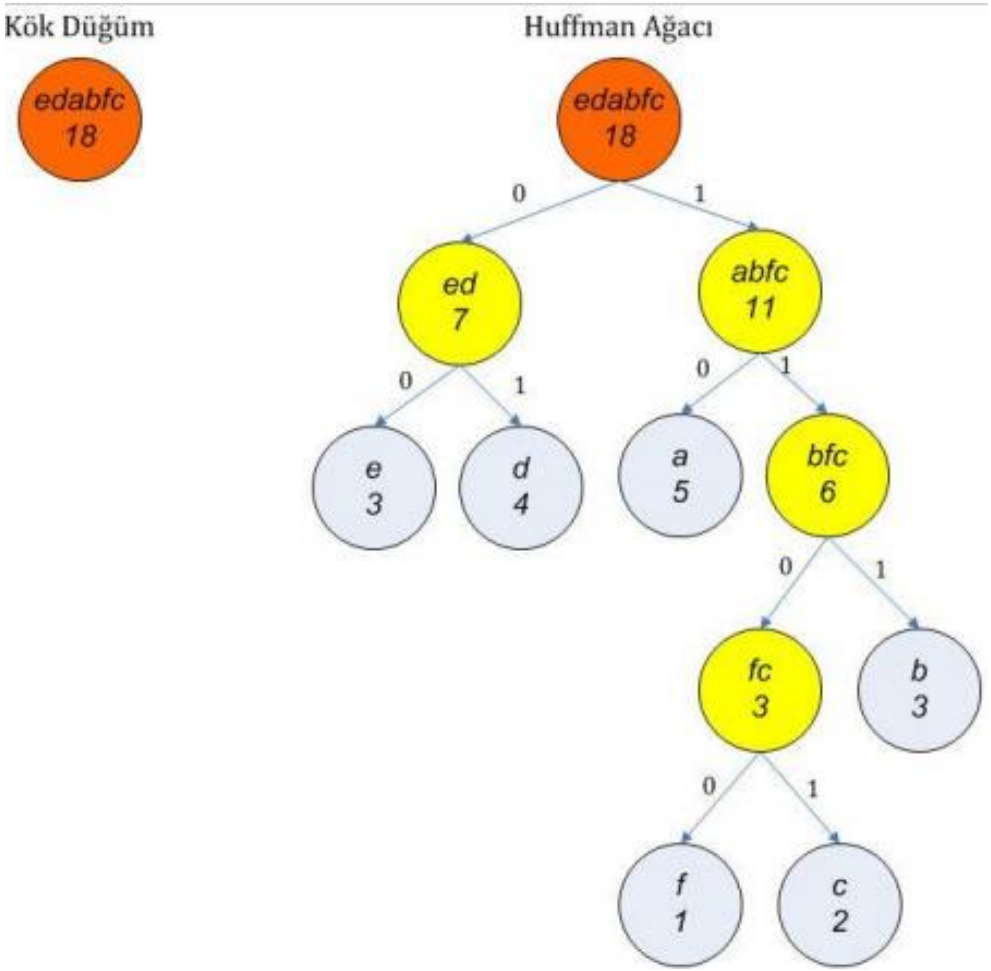
Örnek: Veri setinin kodlanmış hali

Sembol	Huffman Kodu	Bit Sayısı	Frekans
A	0	1	60
B	10	2	40
C	110	3	25
D	1111	4	20
E	1110	4	10

Örnek: Karşılaştırma

- Veri setini ASCII kodu ile kodlanırsa, her karakter için 1 byte(8 bit)'lık alan gerektiğinden toplamda 155 byte(1240 bit)'a ihtiyaç olacaktı.
- Huffman kodlaması ile bu sayı :
- 60x1 + 40x2 + 25x3 + 20x4 +10x4= 335 bittir.
- Görüldüğü gibi Huffman kodlaması ile 335/1240= %27'lik bir sıkıştırma oranı sağlanmaktadır. Bu oran, veri setindeki sembollerin frekansları arttıkça daha da artmaktadır.

Örnek:



- e: 00
- d: 01
- a: 10
- b: 111
- f: 1100
- c: 1101

Örnek

- Elimizde sıkıştırılması istenen alttaki katarın olduğunu varsayalım;
  - aaaabbbccddddeeefa
- Bu katardaki karakter sıklıkları hesaplandığında;
  - f = 1, c = 2, b = 3, e = 3, d = 4, a = 5
- ASCII olarak tutulduğunda bu katar toplam 18 * 8 = 144 bit yer kaplamaktadır.

Örnek:

- Sık olan karakterler üst dallarda, seyrek olanlar alt dallarda.
- Sıkıştırmadan önce
  - 144 bit (18 * 8 bit)
- Huffman kodu
  - 45 bit (sıklık * kod sözcüğü uzunluğu) yer
- Sıkıştırma oranı
  - 45 / 144 = %31

- a: 10 sıklık:5
- d: 01 sıklık:4
- e: 00 sıklık:3
- b: 111 sıklık:3
- c: 1101 sıklık:2
- f: 1100 sıklık:1

Kodları elde ettikten sonra veri içerisindeki tüm veriler baştan okunur. Her karaktere karşılık gelen kod yerine yerleştirilir.

a a a a b b b c c d d d d e e e f a

10 10 10 10 111 111 111 1101 1101 01 01 01 01 00 00 00 1100 10

verimizin boyutu küçülmüş oldu

Huffman Kodunun Çözümü

- Frekans tablosu ve sıkıştırılmış veri mevcutsa bahsedilen işlemlerin tersini yaparak kod çözülür. Diğer bir değişle:
- Sıkıştırılmış verinin ilk biti alınır. Eğer alınan bit bir kod sözcüğüne karşılık geliyorsa, ilgili kod sözcüğüne denk düşen karakter yerine konulur.
- Eğer alınan bit bir kod sözcüğü değil ise sonraki bit ile birlikte alınır ve yeni dizinin bir kod sözcüğü olup olmadığına bakılır. Bu işlem dizinin sonuna kadar yapılır. Böylece huffman kodu çözülerek karakter dizisi elde edilmiş olur.

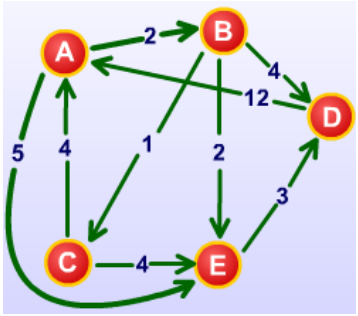
Açgözlü Algoritmalar (ve Grafikler)
Greedy Algorithms (and Graphs)

- Grafik gösterim
- Minimum kapsayan ağaç
- Optimal altyapı
- Açgözlü seçim
- Prim'in açgözlü MST algoritması

Graflar (Graphs)
Konular

- Graflar-Tanım
- Graf Gösterimleri
- Graflarda Dolaşma
  - Breadth- First Search
  - Depth Search
- Hamiltonian cycle
- Euler cycle

Graflar (Graphs) Tanım



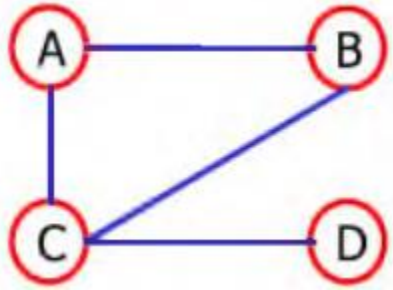
- Graf, matematiksel anlamda, düğümlerden ve bu düğümler arasındaki ilişkiyi gösteren kenarlardan oluşan bir kümedir. Mantıksal ilişki, düğüm ile düğüm veya düğüm ile kenar arasında kurulur.
  - Bağlantılı listeler ve ağaçlar grafların özel örneklerindendir.
- Fizik, Kimya gibi temel bilimlerde ve mühendislik uygulamalarında ve tıp biliminde pek çok problemin çözümü ve modellenmesi graflara dayandırılarak yapılmaktadır.
  - Elektronik devreler, networkler
  - Ulaşım ve iletişim network'leri
  - Herhangi bir türdeki ilişkilerin modellenmesi (parçalar, insanlar, süreçler, fikirler vs.)

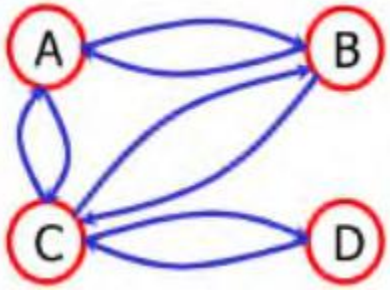
GRAFLAR

- Bir **G** grafı, V ile gösterilen düğümlerden (node veya vertex) ve E ile gösterilen kenarlardan (Edge) oluşur.
  - Her kenar iki düğümü birleştirir.
- Her kenar, iki bilgi (düğüm) arasındaki ilişkiyi gösterir ve (u,v) şeklinde ifade edilir.
  - e=(u, v) iki düğümü göstermektedir.
  - Bir graf yönlü değil ise u ve v düğümleri arasındaki kenar $(u,v) \in E$ ve $(v,u) \in E$ olarak gösterilebilir.
- Bir graf üzerinde n tane düğüm ve m tane kenar varsa, matematiksel gösterilimi, düğümler ve kenarlar kümesinden elamanların ilişkilendirilmesiyle yapılır:
  - $V=\{v_0, v_1, v_2 \dots v_{n-1}, v_n\}$ Düğümler kümesi
  - $E=\{e_0, e_1, e_2 \dots e_{m-1}, e_m\}$ Kenarlar kümesi
  - $G=(V, E)$ → Graf

Graflar

- G = (V, E) grafları aşağıda verilmiştir.

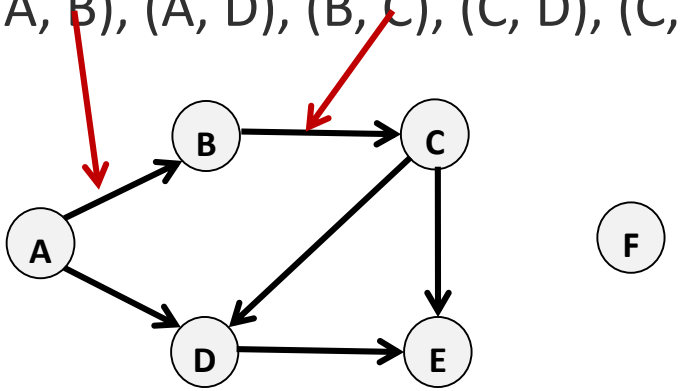


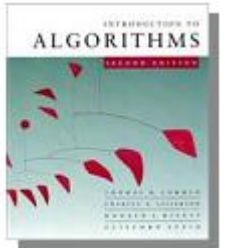


V = {A, B, C, D}

E = {(A,B), (B,A), (A,C), (C,A), (C,D), (D,C), (B,C), (C,B)}

- V = {A, B, C, D, E, F}
- E = {(A, B), (A, D), (B, C), (C, D), (C, E), (D, E)}





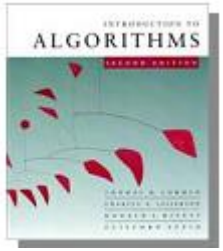
Graflar

Tanım. ***Yönlendirilmiş grafik (digraf)***
$G = (V, E)$,

- V köşeler kümesi ve
- $E \subseteq V \times V$ kenarlar kümelerinden oluşur.

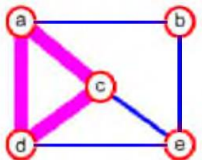
Yönlendirilmemiş grafik $G = (V, E)$'de, E kenar kümesi, sıralı olmayan köşe çiftlerinden oluşur.

Her durumda, elimizde $|E| = O(V^2)$ vardır.
Ayrıca, eğer G bağlantılıysa, $|E| \geq |V| - 1$, ki bu da $\lg|E| = \Theta(\lg V)$ anlamına gelir.

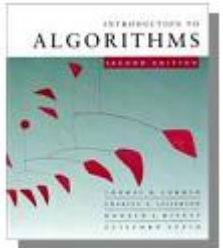


Graflar-Tanımlar

- **Komşu(adjacent):** Eğer (u,v) $\in$ E ise, u ve düğümleri komşudur.
- **Derece(degree):**Bir düğümün derecesi, komşu düğüm sayısına eşittir.
- **Yol (path):** v_0, v_1, v_2 ... v_n, düğümlerinin sırasıdır, v_{i+1} düğümü v_i düğümünün komşusudur. (i= 0..n-1)
- **Basit Yol (simple path):** düğüm tekrarı olmayan yoldur.
- **Döngü(cycle):** basit yoldur, sadece ilk ve son düğüm aynıdır.

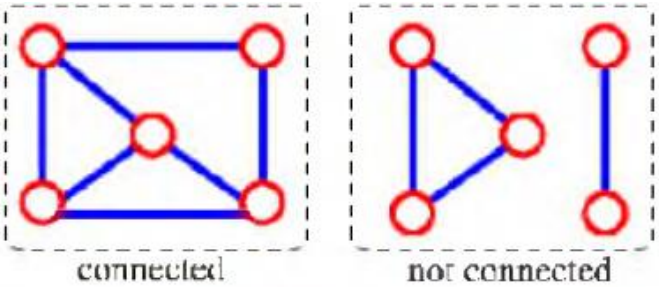


acda

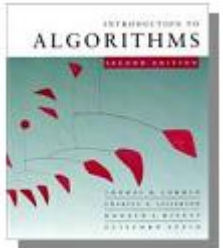


Graflar-Tanımlar

- **Bağlı Graf(connected graph):** Bütün düğümler bir birine herhangi bir yol ile bağlıdır.

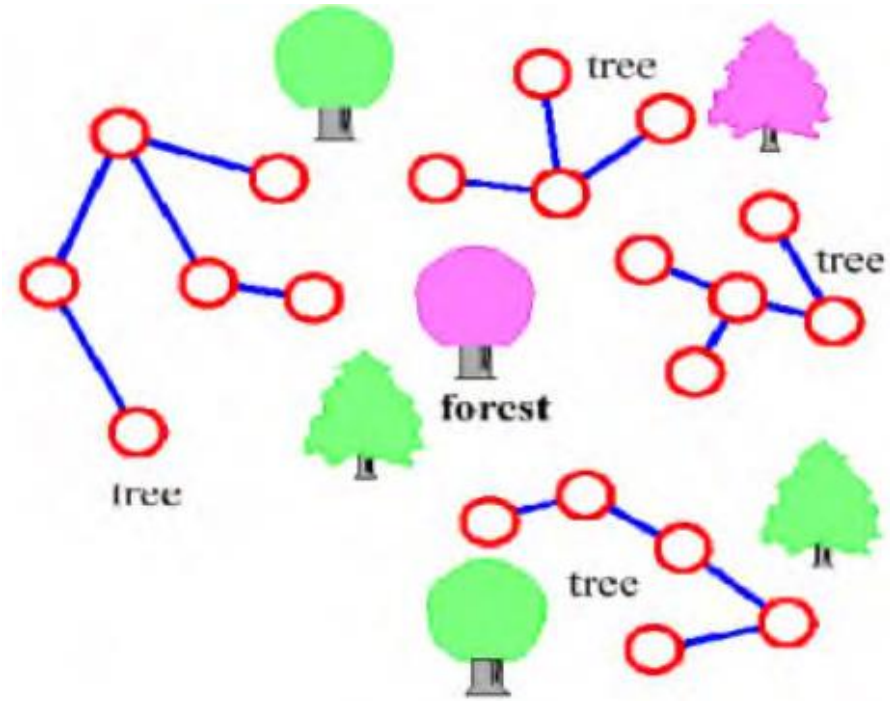


- **Alt graf(Subgraph):**Bir grafı oluşturuan alt düğümler ve kenarlar kümesidir.
- **Bağlı elaman (Connected component):** bağlı alt graflar. Örnek: 3 tane bağlı elamana sahip graflar

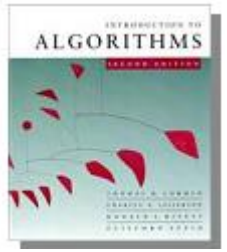


Graflar-Tanımlar

- **Ağaç (tree):** Döngü olmayan bağlı graflardır.
- **Orman(forest):** Ağaçların toplamıdır.



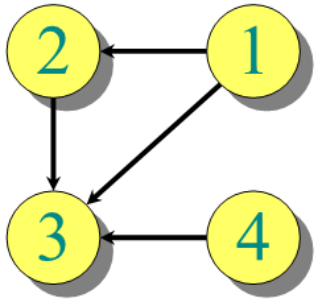
Graflar için Veri yapıları
Komşuluk matrisi gösterimi



Bir $G = (V, E)$ grafiğinin *komşuluk matrisi* $V = \{1, 2, \ldots, n\}$ iken,

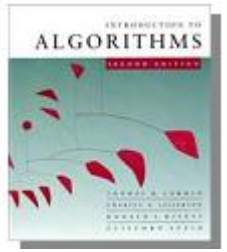
$$A[i,j] = \begin{cases} 1 \text{ eğer } (i,j) \in E \text{ ise,} \\ 0 \text{ eğer } (i,j) \notin E \text{ ise,} \end{cases}$$

$A[1 \ldots n, 1 \ldots n]$ matrisidir.



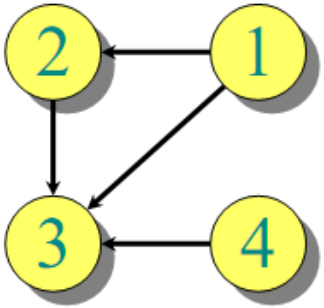
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ depolama:
$\Rightarrow$***yoğun*** gösterimi.



Komşuluk listesi gösterimi

Bir $v \in V$ köşesinin *komşuluk listesi,* v′ ye komşu olan köşelerin $Adj[v]$ listesidir.



$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$

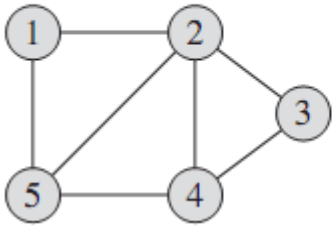
Yönlendirilmemiş grafikler için, $|Adj[v]| = degree$ (derece)(v).
Yönlendirilmiş grafikler için, $|Adj[v]| = out\text{-}degree$ *(dış-derece)*(v).

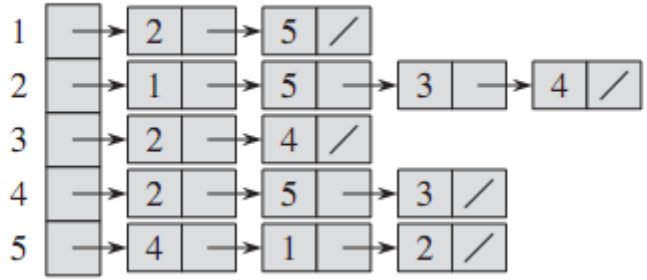
Handshaking Lemma

Tokalasma önkuramı: Yönlendirilmemiş grafikler için $\sum_{v \in V} = 2|E| \Rightarrow$ komşuluk listeleri, $\Theta(V + E)$ depolama alanını kullanır. Bu seyrek gösterimdir. (her tür grafik için.)

Komşuluk listesi gösterimi

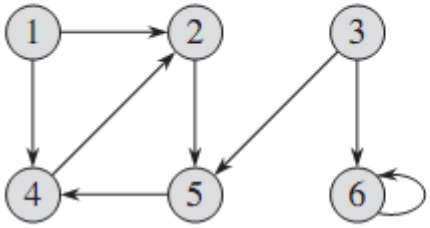
- Yönsüz graf komşuluk listesi

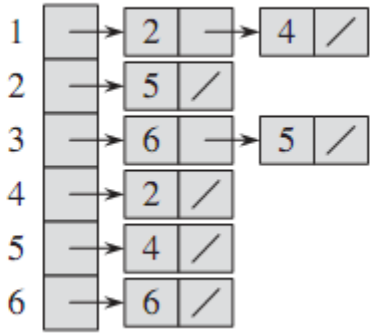




	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

- Yönlü graf komşuk listesi

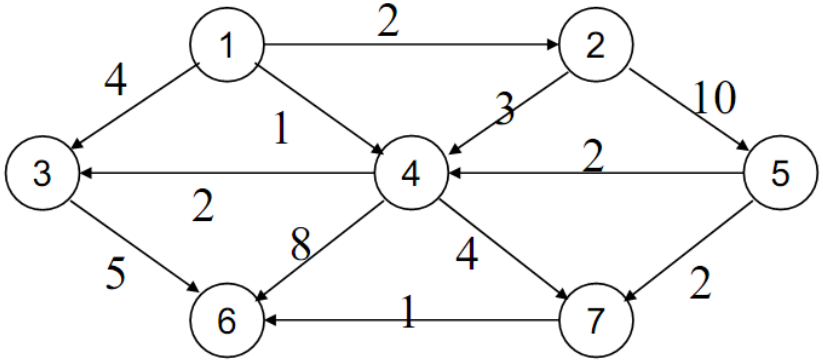


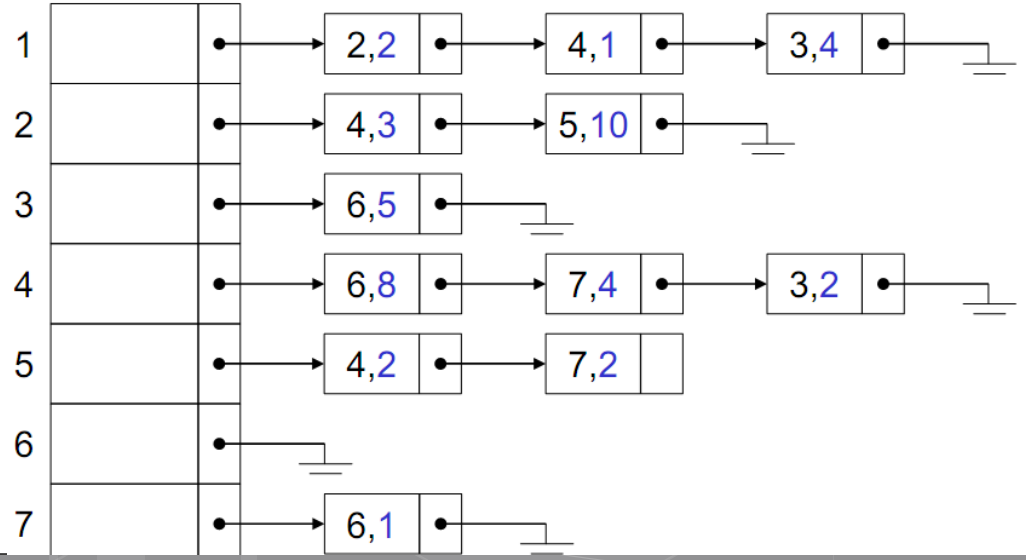


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Komşuluk listesi gösterimi

- Her kenarı ağırlıklandırılmış yönlü graf komşuluk listesi





Graf Arama Algoritmaları

- Graf içindeki her kenar ve düğüm için sistematik arama.
- G=(V,E) grafı yönlü olabilir veya olamayabilir.
- Uygulamalar
  - Derleyiciler(Compilers)
  - Grafikler(Graphics)
  - Haritalama(Mapping)
  - Ağlar(Networks):routing, searchind, clustering, vs.

Enine Arama
Breatdh First Search (BFS)

- BFS, bir grafın bağlı olan parçalarını dolaşır ve bir kapsayan ağaç (spaning tree) oluşturur.
- BFS, arama ağaçlarındaki level order aramaya benzer.
- 1-Seçilen düğümün tüm komşuları sırayla seçilir ve ziyaret edilir.
- 2- Her komşu kuyruk içerisine atılır.
- 3- Komşu kalmadığında kuyruk içerisindeki ilk düğüm alınır ve 2. adıma gidilir.

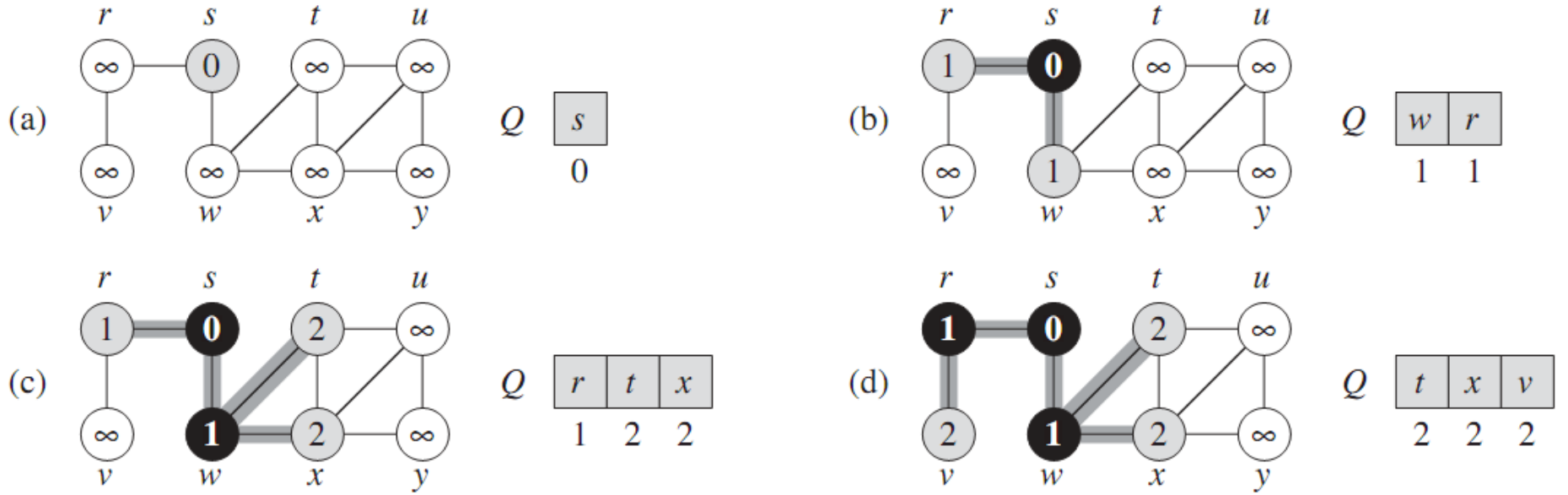
Breatdh First Arama-düğüm renklendirme

- Ulaşılmayan bir düğüm beyaz renklidir.
- Bir düğüm, eğer kendine ulaşılmış ancak tüm kenarlarına bakılmamışsa gri renklidir.
- Bir düğüm, eğer kendisinin tüm komşu düğümlerine ulaşıldıysa (komşuluk listesi tamamen incelenir) siyah renklidir.

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

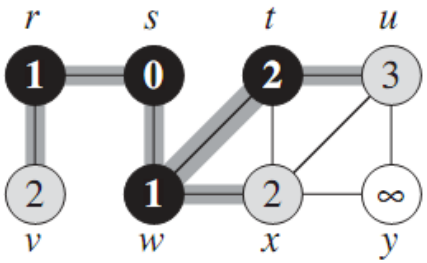
Breatdh First Arama Örnek

- S düğümü ile başla



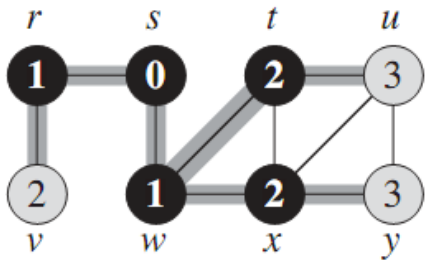
Breatdh First Arama Örnek

(e)



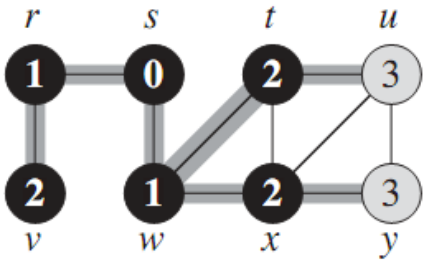
Q | x | v | u |
2 2 3

(f)



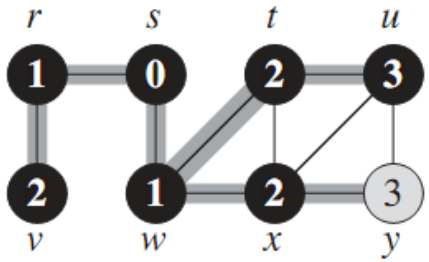
Q | v | u | y |
2 3 3

(g)



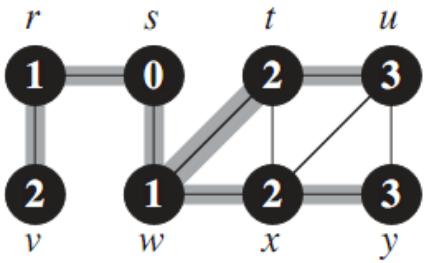
Q | u | y |
3 3

(h)



Q | y |
3

(i)

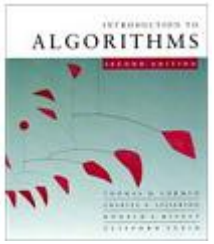


Q $\emptyset$

BFS algoritmasının Analizi

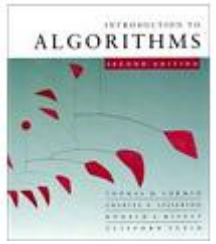
- G= (V,E) grafı için
  - Beyaz renkli tüm düğümler sıraya konur. Kuyruğa ekleme ve çıkarma işlemleri O(1) süresinde yapılırsa toplam çalışma süresi düğüm sayısı kadardır, O(V).
  - Bir düğüm kuyruktan çıkarıldığında komşuluk listesi taranır. Toplamda komşuluk listesini taramanın çalışma süresi O(E) olur.
  - BFS nin toplam çalışma zamanı: O(V+E) olur.

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

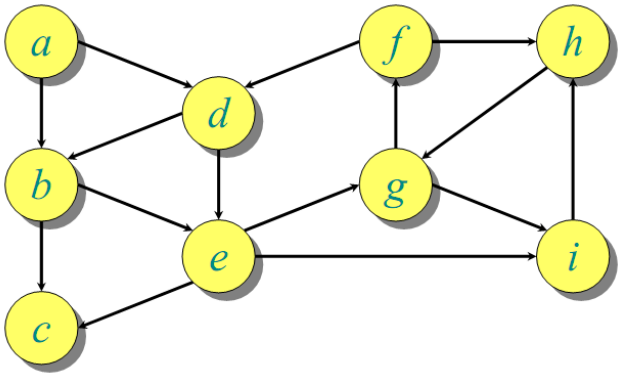


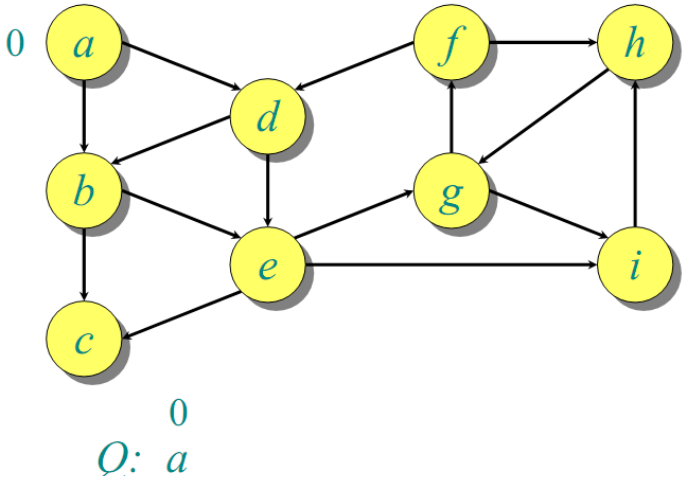
BFS Özellikleri

- G=(V,E) grafı için BFS, s düğümünden ulaşılabilecek tüm düğümleri dolaşır.
- Ulaşılabilecek düğümler için en kısa yolu hesaplayabilir.
- Ulaşılabilecek tüm düğümler için BF ağacını oluşturur.
- s düğümünden ulaşılan bir v düğümü için BF ağacı içindeki yol, G grafındaki en kısa yolu (shortest path) ifade eder.



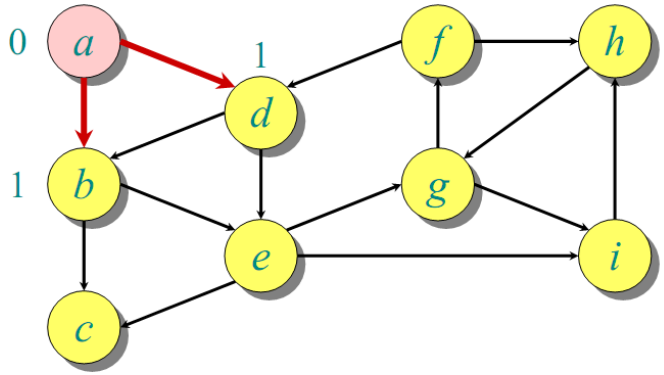
Yönlü Graf(Digraph) için Enine arama Örnek



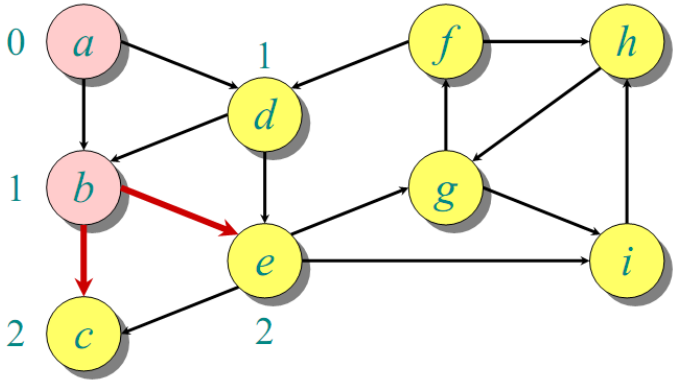


0
Q: a

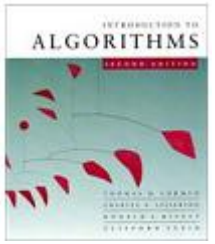
Q:



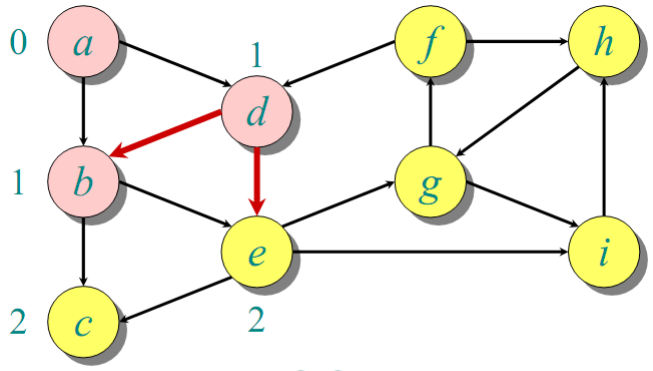
1 1
Q: a b d



1 2 2
Q: a b d c e

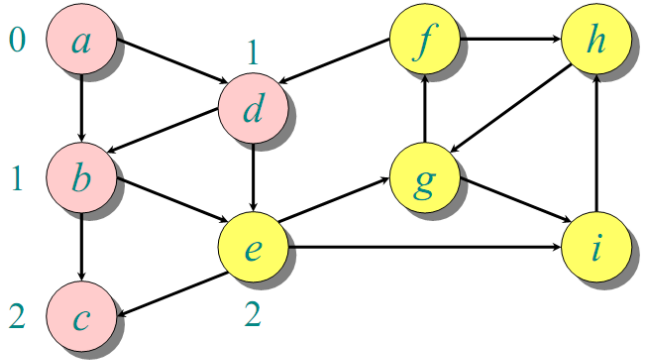


Enine arama için örnek



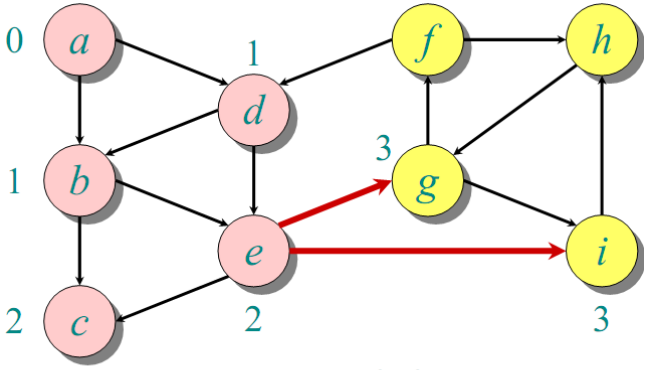
2 2

Q: a b d c e



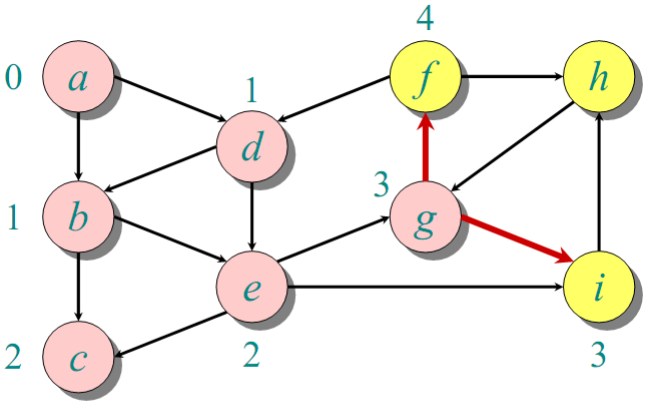
2

Q: a b d c e



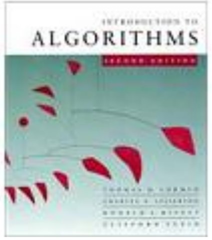
3 3

Q: a b d c e g i

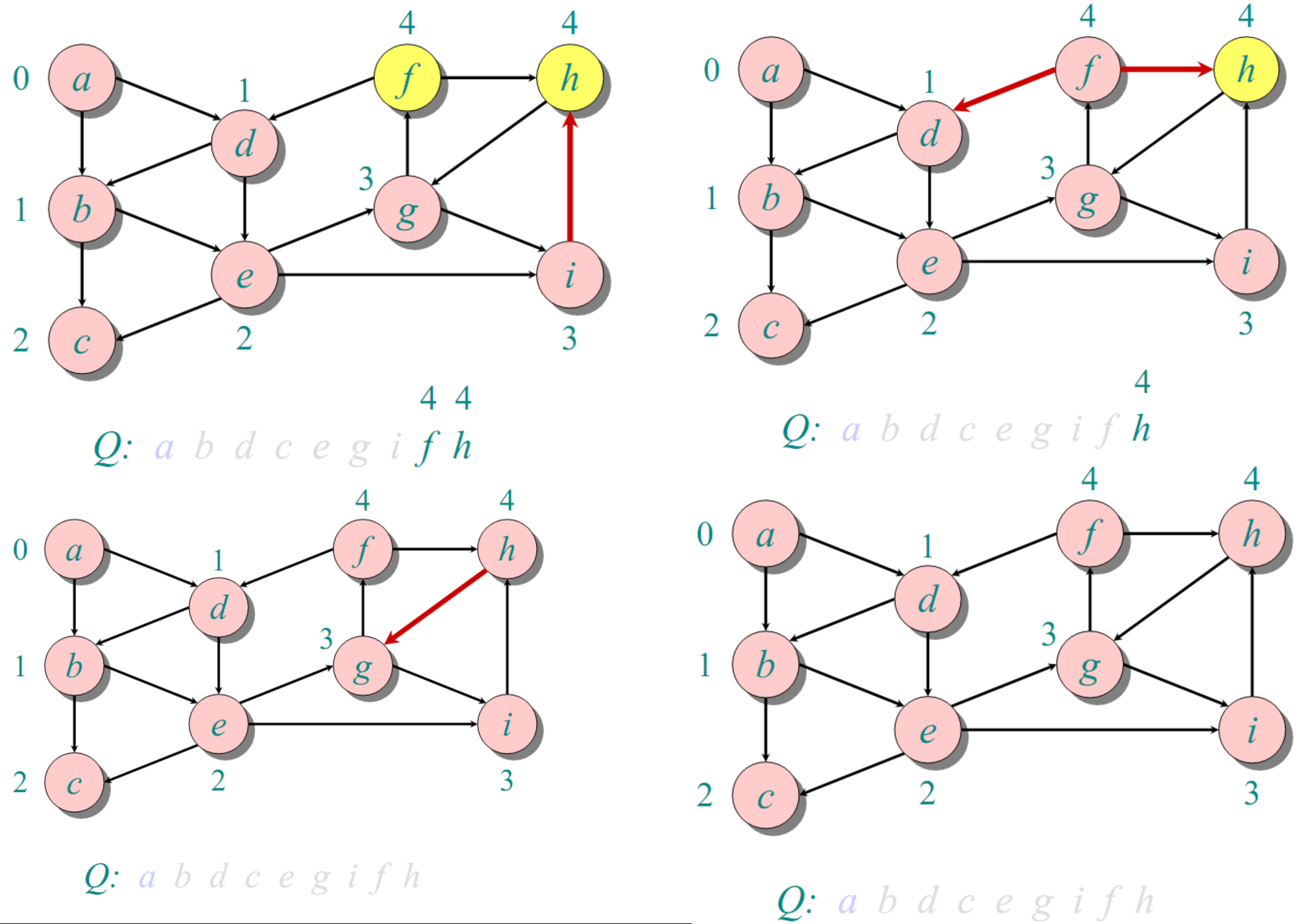


3 4

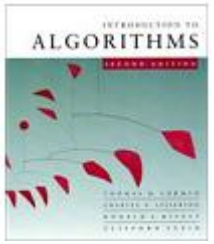
Q: a b d c e g i f



Enine arama için örnek

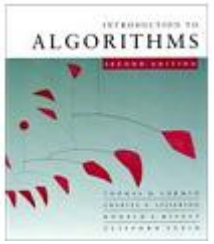


Derinlemesine Arama
Depth First Search(DFS)

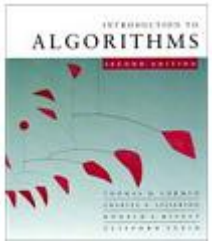


- **1-** DFS ile önce bir başlangıç düğümü seçilir ve ziyaret edilir.
- **2-** Seçilen düğümün bir komşusu seçilir ve ziyaret edilir.
- **3-** Seçilen komşu düğümün bir komşusu seçilir ve ziyaret edilir.
- **4-** 3. adım koşu düğüm kalmayıncaya kadar devam eder.
- **5-** Komşu kalmadığında geri dönülür (backtracking) ve her düğüm için yeniden 3.adıma gidilir.

Derinlemesine Arama
Depth First Search(DFS)

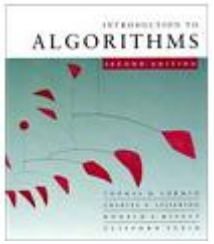



```
DFS(G,s)
    for each vertex u ∈ V[G]
        color[u] ← WHITE
        Π[u] ← NIL
    time ← 0
    for each vertex u ∈ V[G]
        if color[u] == WHITE
                then DFS-VISIT(u)
DFS-VISIT(u)
    color[u] ← GRAY          // white vertex u has just been discovered
    d[u] ← time ← time + 1
    for each vertex v ∈ Adj[u]          // explore edge (u,v)
        do if color[v] == WHITE
                then Π[v] ← u
                        DFS-VISIT(v)
    color[u] ← BLACK                    // Blaken u; it is finished
    f[u] ← time ← time + 1
```



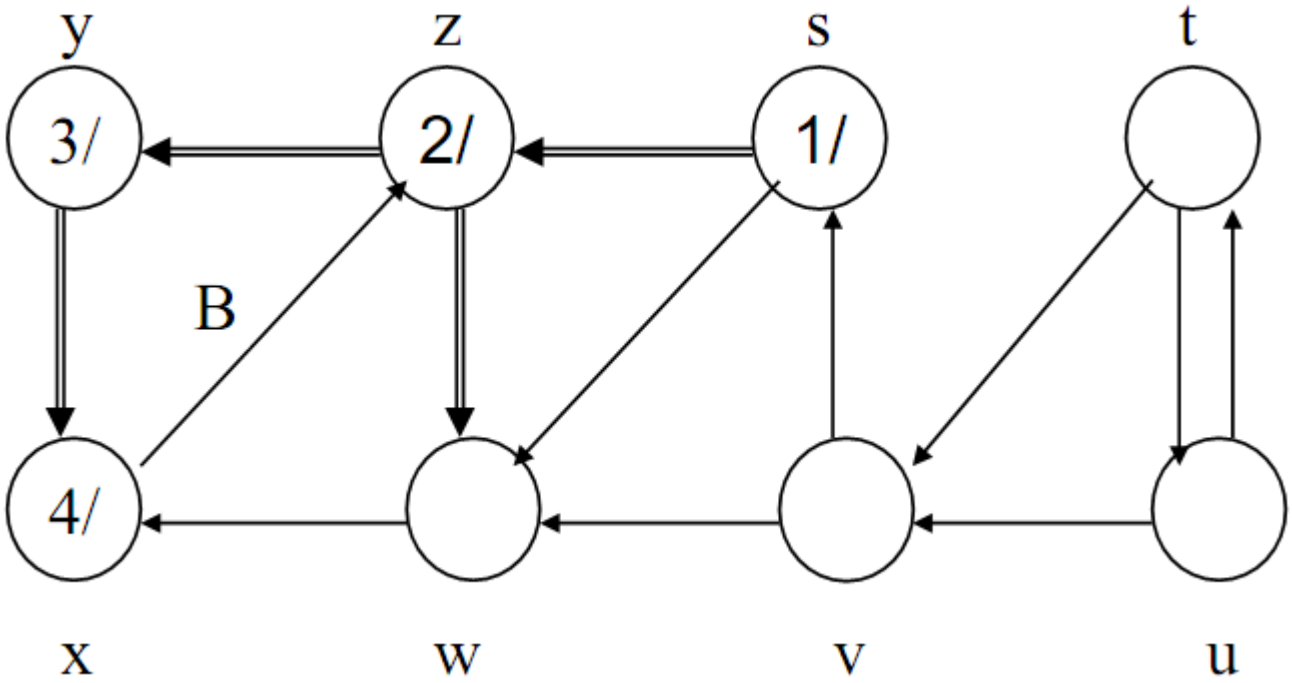
Derinlemesine Arama-(DFS)

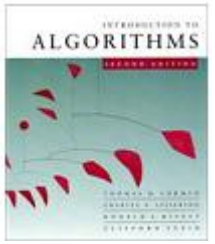
- Başlatma(initialize)- tüm düğümler beyaz yapılır.
- Tüm beyaz düğümler DFS-Visit kullanılarak dolaşılır.
- DFS-Visit(u) çağrıldığında u düğümü yeni ağacın kökü yapılır.
- DFS çalışmasını bitirince, her u düğümü için erişilme zamanı (discovery time) **d[u]**, ve bitirme zamanı (finishing time) **f[u]** değerleri atanır.
- **Back Edge(B):** DF ağacında u düğümünün v atasına (ancestor) bağlı olması.
- **Forward Edge (F):**DF ağacında u düğümünün, v toruna (descendant) bağlı olması.
- **Cross Edge(C):** Geri kalan tüm kenarlar. Bu kenarlar ile aynı yada farklı DF ağacında, atası olmadığı diğer düğümler arasında dolaşabilir.



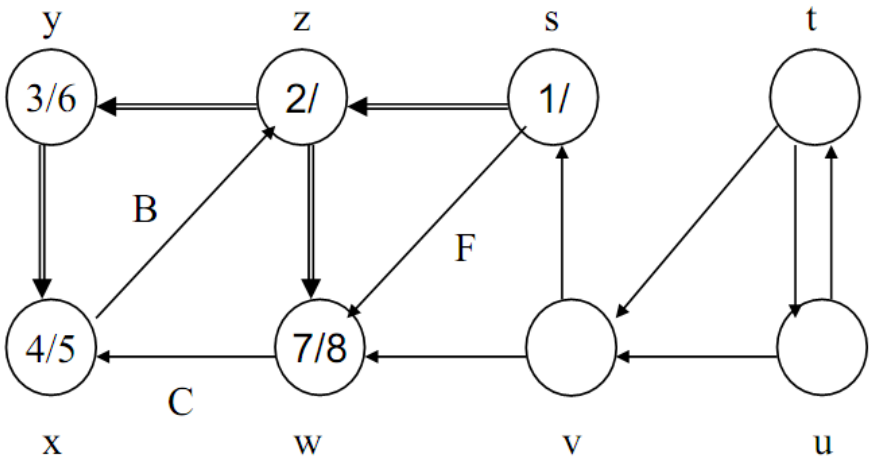
DFS-Örnek

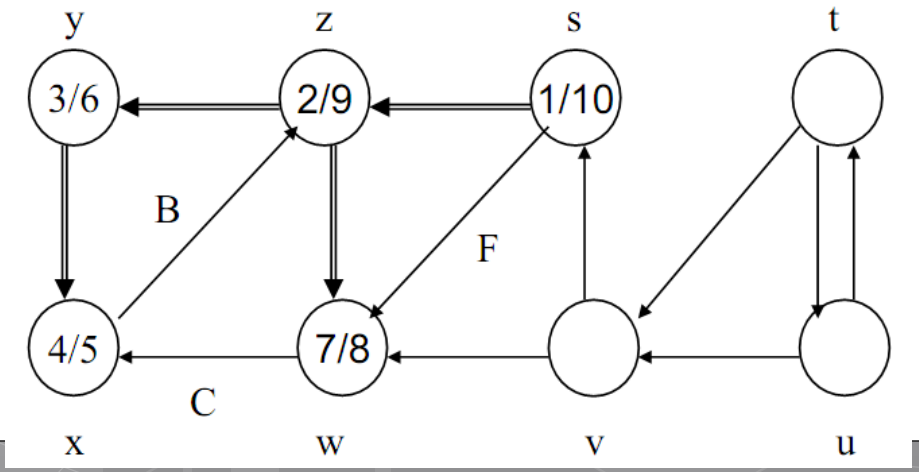
- S düğümü ile başla,
- d/f→ d: erişilme zamanı, f: bitirme zamanı

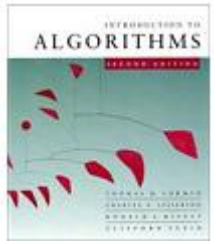



INTRODUCTION TO
ALGORITHMS

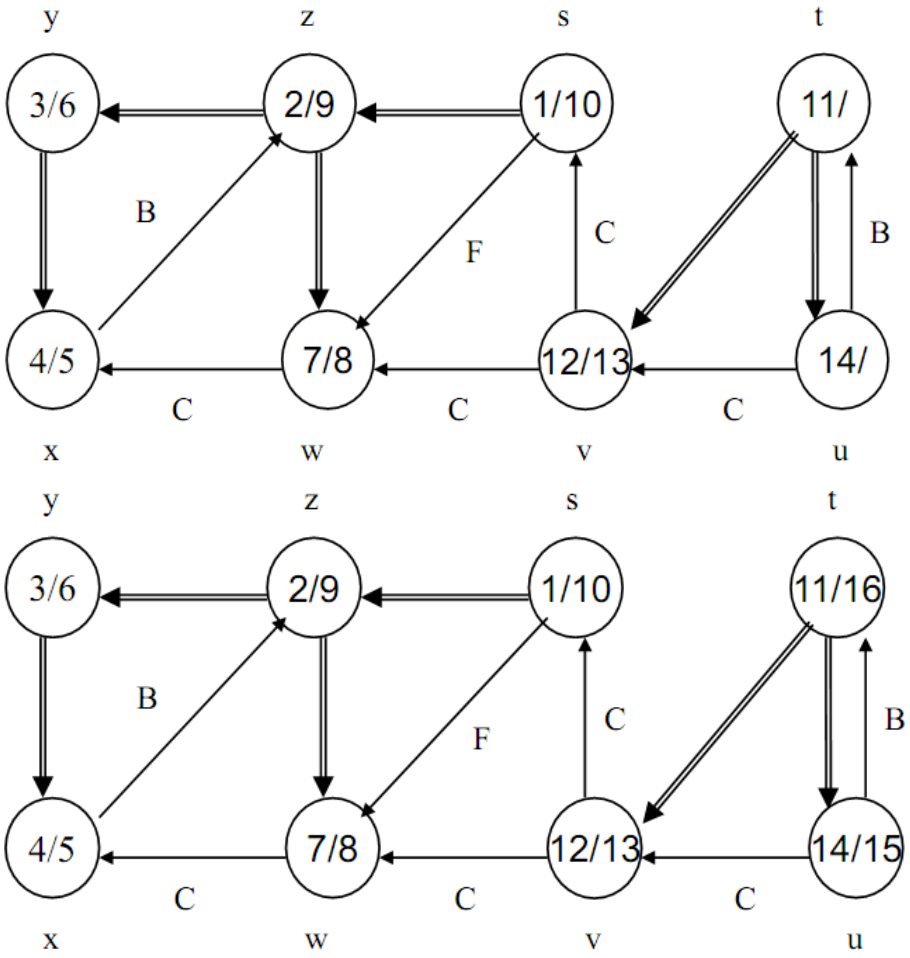
DFS-Örnek

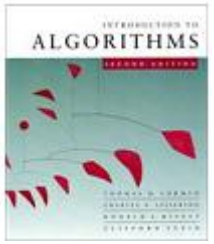

y
z
s
t
3/6
2/
1/
B
F
4/5
7/8
C
x
w
v
u


y
z
s
t
3/6
2/9
1/10
B
F
4/5
7/8
C
x
w
v
u

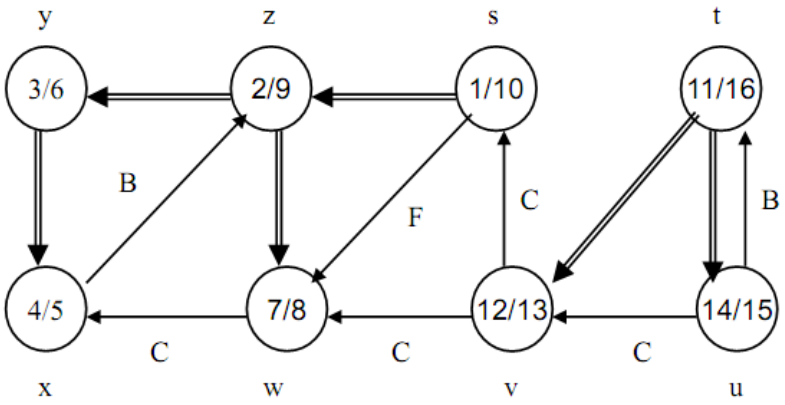

INTRODUCTION TO
ALGORITHMS

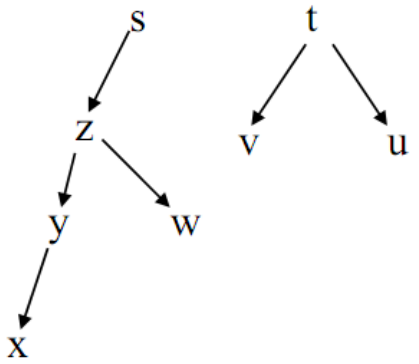
DFS-Örnek


y
z
s
t
3/6
2/9
1/10
11/
B
F
C
B
4/5
7/8
12/13
14/
C
C
C
x
w
v
u
y
z
s
t
3/6
2/9
1/10
11/16
B
F
C
B
4/5
7/8
12/13
14/15
C
C
C
x
w
v
u

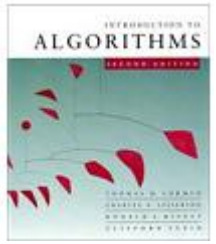


DFS-Örnek

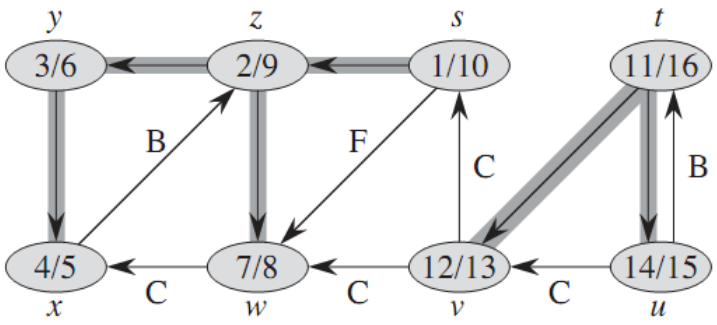


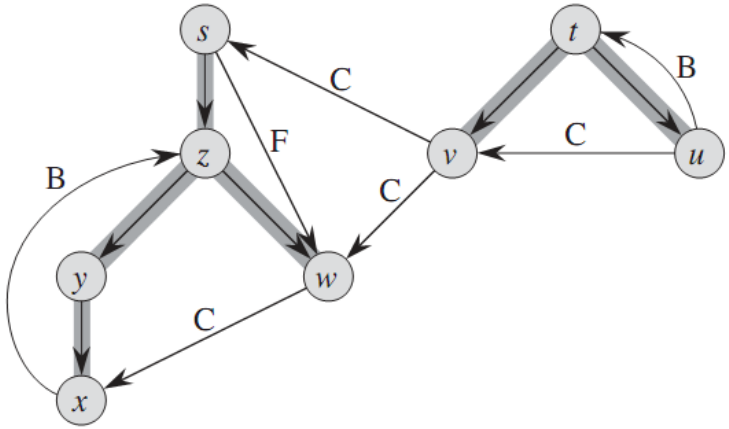


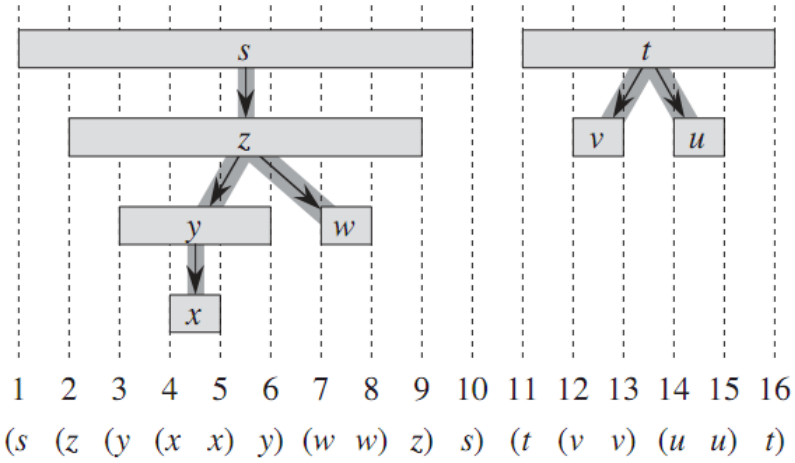
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
(s	(z	(y	(x	x)	y)	(w	w)	z)	s)	(t	(v	v)	(u	u)	t)

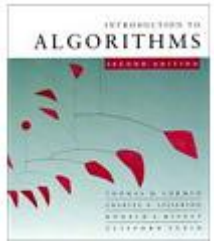

INTRODUCTION TO
ALGORITHMS

DFS-Örnek

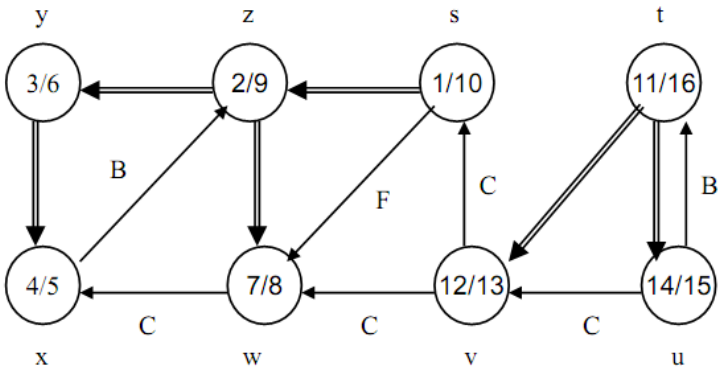

y
z
s
t
3/6
2/9
1/10
11/16
B
F
C
B
4/5
7/8
12/13
14/15
C
C
C
x
w
v
u

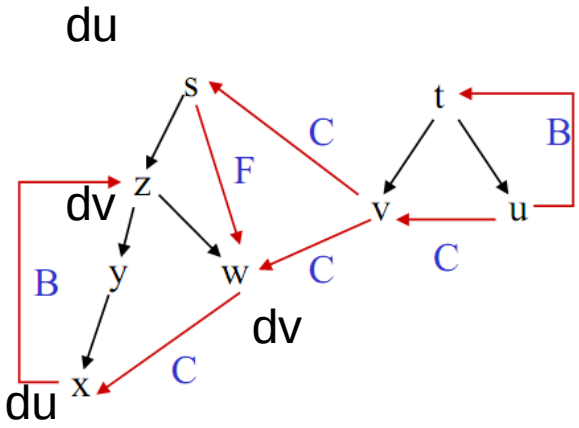

s
t
C
B
z
F
v
C
B
C
y
w
C
x


s
t
z
v
u
y
w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)



DFS-Örnek





Edge (u,v) is:

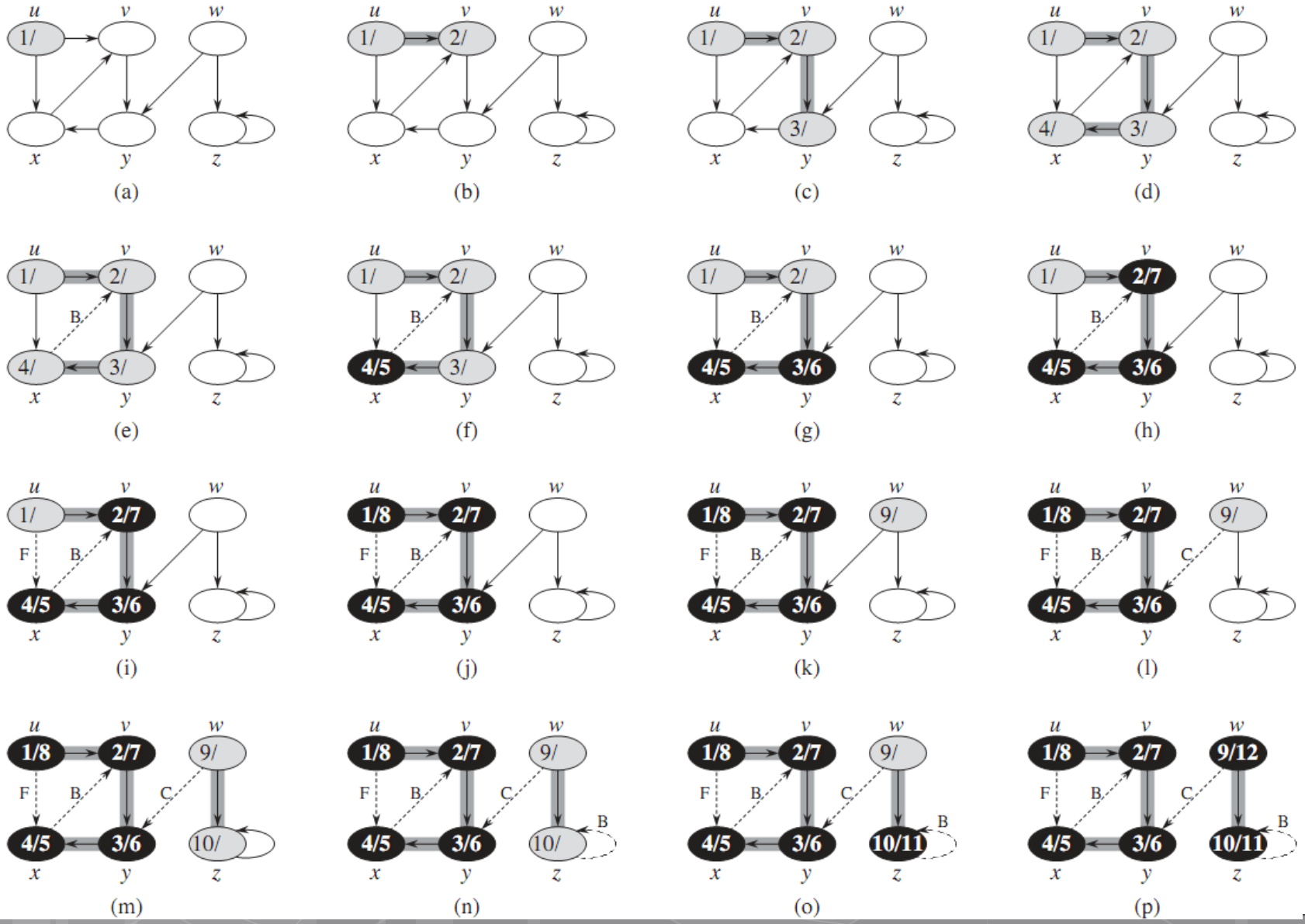
A tree edge if $d[u] < d[v] < f[v] < f[u]$

A forward edge if $d[u] < d[v] < f[v] < f[u]$ and (u,v) is not a tree edge

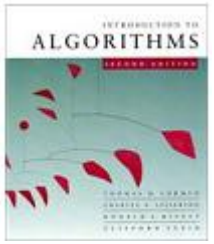
A back edge iff $d[v] < d[u] < f[u] < f[v]$

A cross edge otherwise.

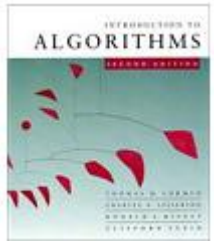
DFS-Örnek


u v w
1/ x y z
(a)
u v w
1/ 2/ x y z
(b)
u v w
1/ 2/ 3/ x y z
(c)
u v w
1/ 2/ 4/ 3/ x y z
(d)
u v w
1/ 2/ B 4/ 3/ x y z
(e)
u v w
1/ 2/ B 4/5 3/ x y z
(f)
u v w
1/ 2/ B 4/5 3/6 x y z
(g)
u v w
1/ 2/7 B 4/5 3/6 x y z
(h)
u v w
1/ 2/7 F B 4/5 3/6 x y z
(i)
u v w
1/8 2/7 F B 4/5 3/6 x y z
(j)
u v w
1/8 2/7 9/ F B 4/5 3/6 x y z
(k)
u v w
1/8 2/7 9/ F B C 4/5 3/6 x y z
(l)
u v w
1/8 2/7 9/ F B C 4/5 3/6 10/ x y z
(m)
u v w
1/8 2/7 9/ F B C 4/5 3/6 10/ B x y z
(n)
u v w
1/8 2/7 9/ F B C 4/5 3/6 10/11 B x y z
(o)
u v w
1/8 2/7 9/12 F B C 4/5 3/6 10/11 B x y z
(p)

DFS-Çalışma Zamanı ve düğüm renklendirme

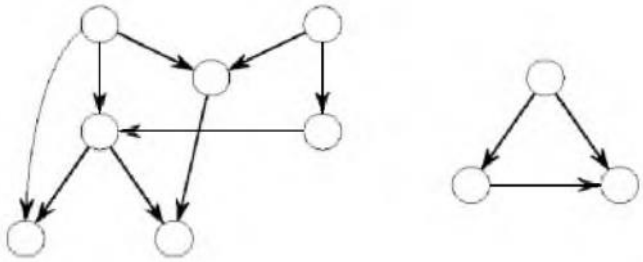


- Çalışma zamanı:
  - DFS içindeki döngü $\Theta(V)$ süresi alır (DFS-Visit çalışma süresi hariç).
  - DFS-Visit her düğüm için bir kez çağrılır.
    - Sadece beyaz renkteki düğümler için çağrılır
    - Düğüm hemen gri renk yapılır.
  - DFS-Visit her çağrılışında |adj[v]| kez çalışır $\sum_{v \in V} |Adj[v]| = \Theta(E)$
  - DFS –Visit için toplam çalışma zamanı→ $\Theta(E)$
  - DFS için toplam çalışma zamanı → $\Theta(V+E)$
- Düğüm u,
- d[u] zamanından önce beyazdır.
- d[u] ve f[u] zamanda gridir, bundan sonra ise siyahtır.

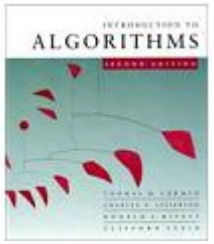


Döngü olmayan yönlü graflar (DAG)
DAG-Directed Acylic Graphs

- Bir DAG döngü olmayan yönlü bir graftır.

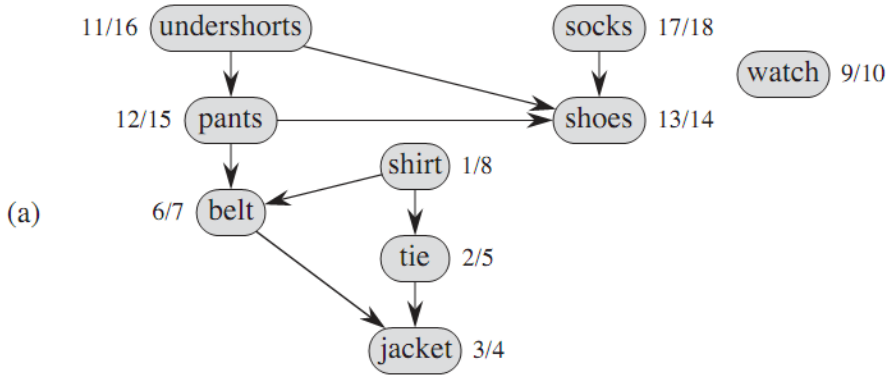


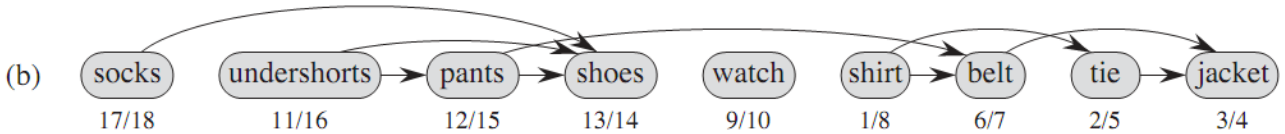
- Sıklıkla olaylar arasındaki öncelik sırasını göstermek için kullanılır. Örnek a olayı b olayından önce olmak zorundadır (paralel kod çalıştırma).
- Tüm sıra topolojik sıralamayla (topological sort) verilebilir.

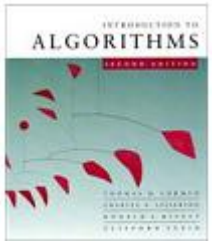


Topolojik Sıralama

- **Öncelik İlişkileri (Precedence relations):** x' ten y' ye bir kenar, x olayı y den önce olur anlamını içerir. Bir iş, kendisinin tüm alt işleri planladıktan sonra planlanabilir.







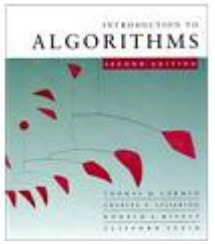
Topolojik Sıralama

- Bir DAG'nın topolojik sıralaması tüm düğümlerin doğrusal sırasını ifade eder. **Grafta döngü yoktur!**
- Topolojik sıralamada herhangi bir (u,v) kenarında, u, sıralamada v'den öncedir.
- Aşağıdaki algoritma bir DAG için topolojik sıralamayı yapar

TOPOLOGICAL-SORT(G)

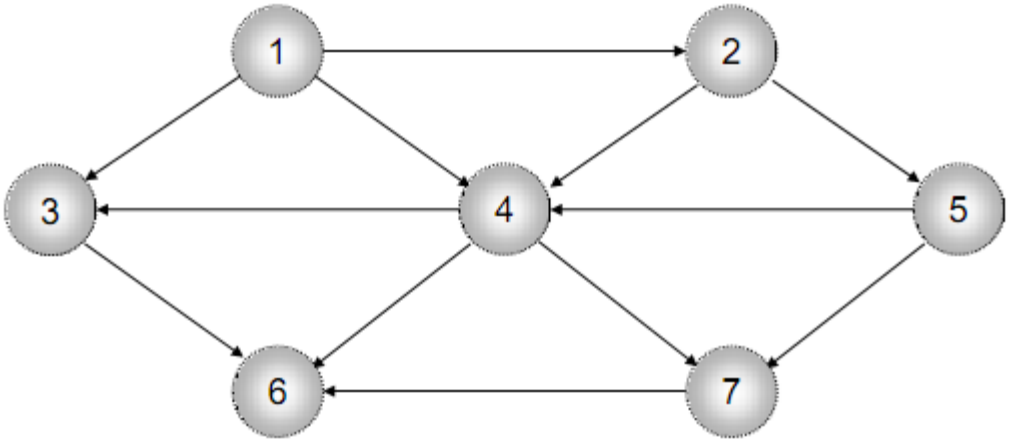
1 call DFS(G) to compute finishing times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

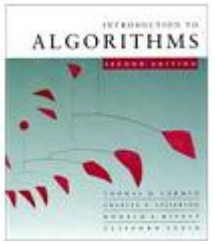
- Toplam sıralama bağlı dizilerle ile oluşturulur.



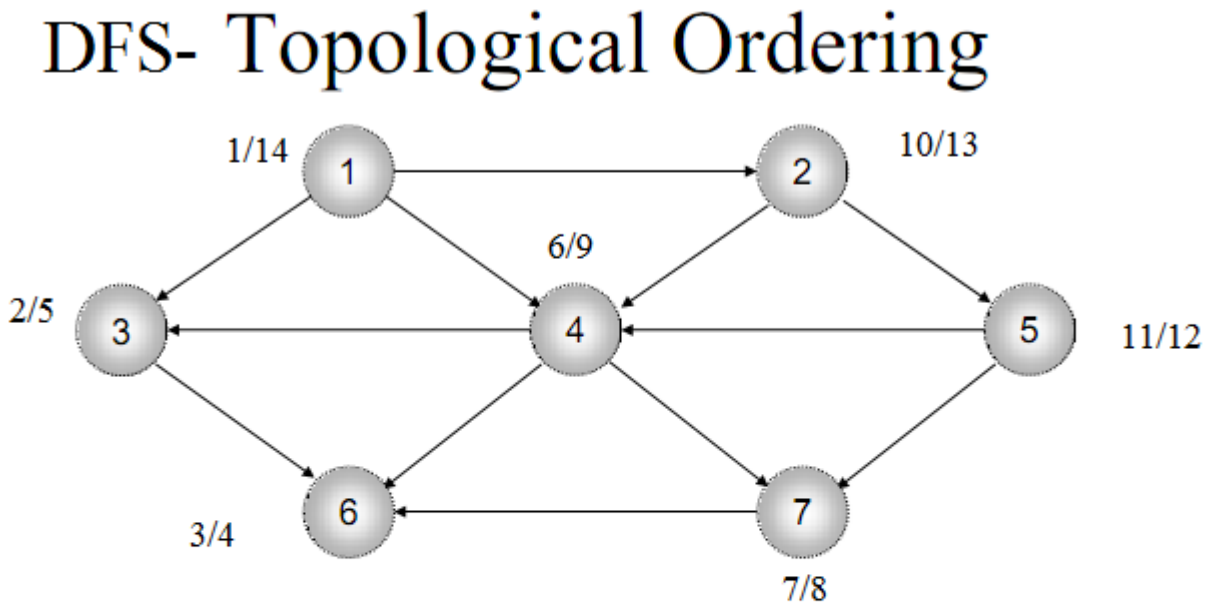
Topolojik Sıralama

DFS- Topological Ordering



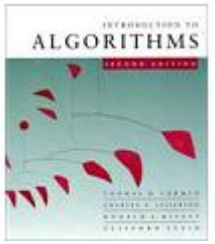


Topolojik Sıralama



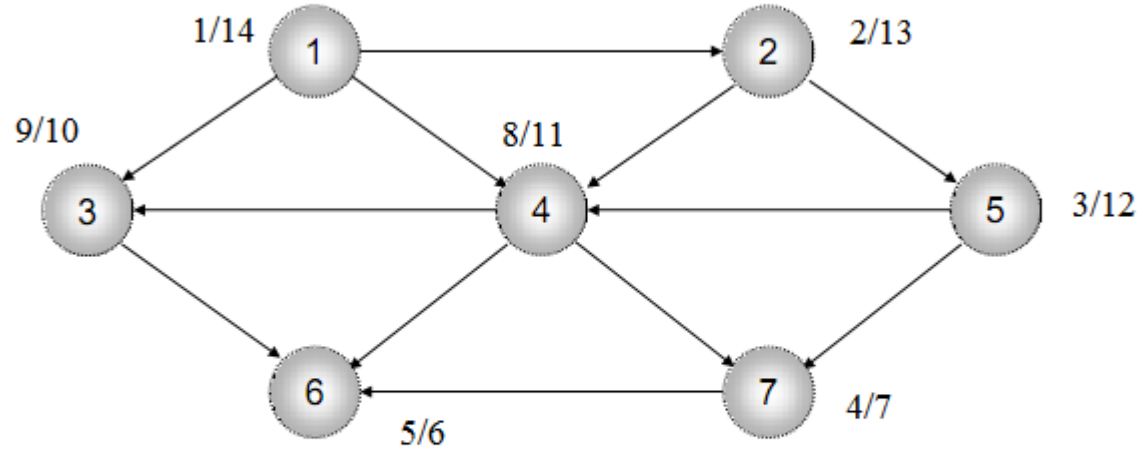
1	2	5	4	7	3	6
1/14	10/13	11/12	6/9	7/8	2/5	3/4

Topolojik sıralama için Sol düğüme göre DFS dolaşma



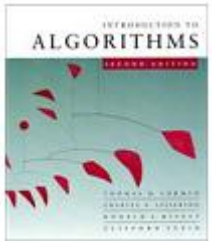
Topolojik Sıralama

DFS- Topological Ordering



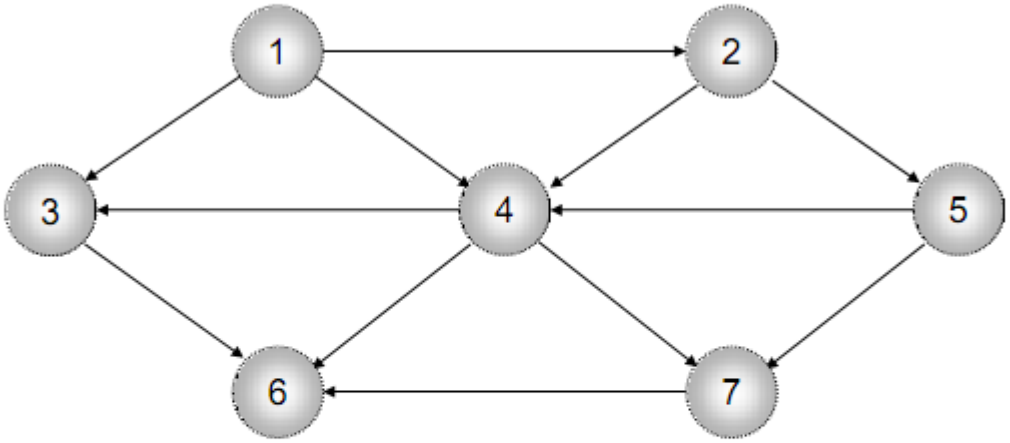
1	2	5	4	3	7	6
1/14	2/13	3/12	8/11	9/10	4/7	5/6

Topolojik sıralama için Sağ düğüme göre DFS dolaşma



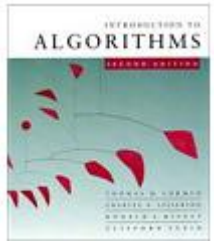
Topolojik Sıralama Çalışma zamanı

Topological Ordering



A topological ordering 1 2 5 4 3 7 6

Another topological ordering 1 2 5 4 7 3 6

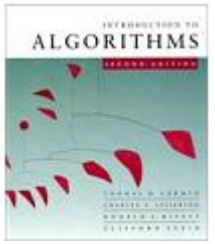


Topolojik Sıralama Çalışma zamanı

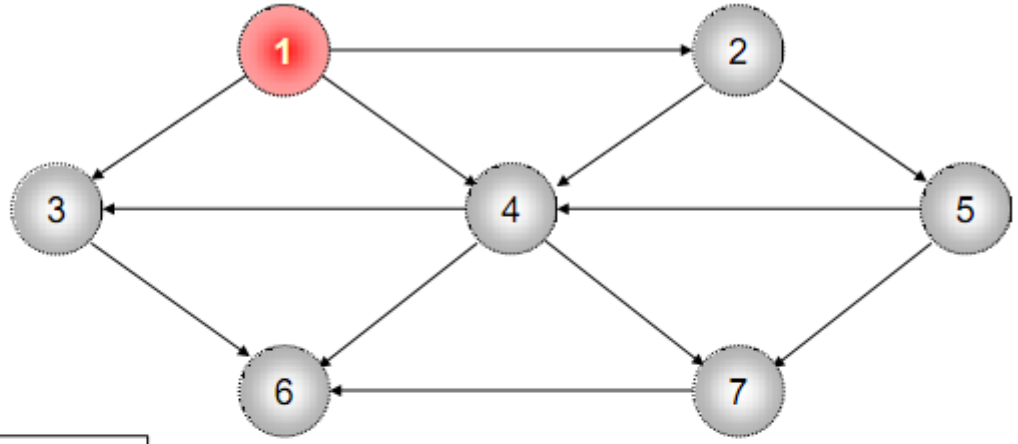
- Çalışma Zamanı:
- DFS: O(V+E)
- Her bir |V| düğümünün bağlı dizi başına eklenmesi O(1) süresini alır (her ekleme için)
- Toplam çalışma zamanı: O(|V^2|) veya O(|V|+|E|) olur.

```
void Graph::topsort()
{
        Queue q(NUM_VERTICES);
        int counter = 0; //topological order of a vertex:1,2,3,…,NUM_VERTICES
        Vertex v, w;

        q.makeEmpty();
        for each vertex v
                if (v.indegree == 0)
                        q.enqueue(v):
        while (! q.isEmpty())
        {
                v = q.dequeue();
                counter++;

                for each w adjacent to v
                        if (--w.indegree == 0)
                                q.enqueue(w);
        }
        if (counter != NUM_VERTICES)
                throw CycleFound();
}
```

Pseudocode of a Topological Sort Algorithm

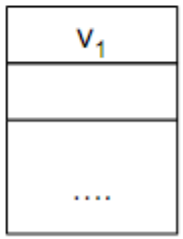


Topolojik Sıralama

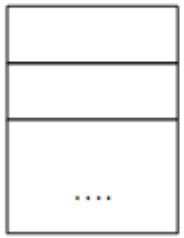


Indegrees	
v_1	0
v_2	1
v_3	2
v_4	3
v_5	1
v_6	3
v_7	2

After Enqueue

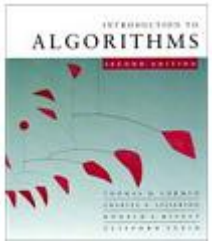


After Dequeue

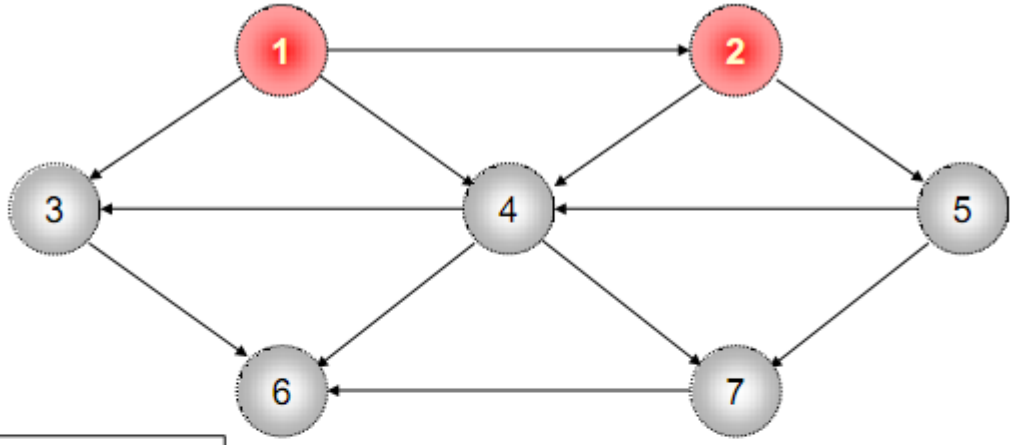


Print

v_1

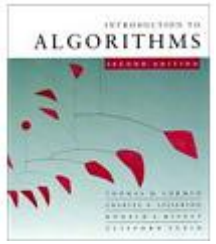


Topolojik Sıralama

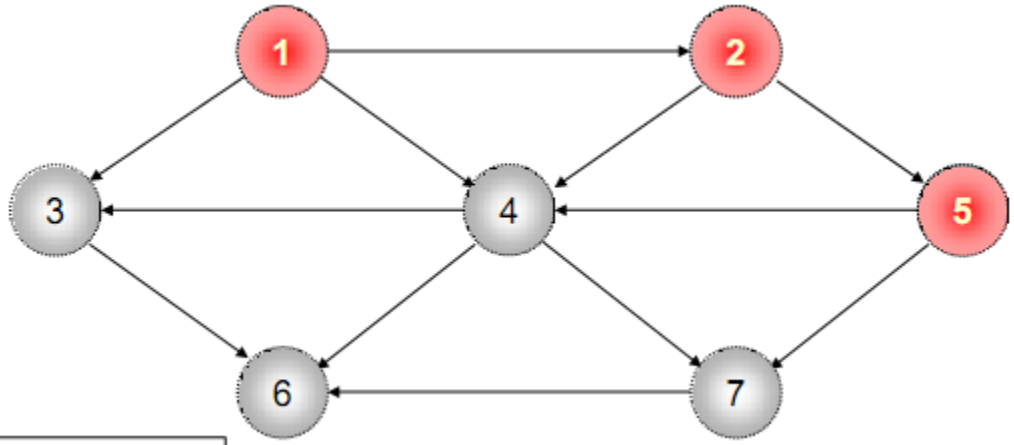


Updated Indegrees	
v_1	0
v_2	0
v_3	1
v_4	2
v_5	1
v_6	3
v_7	2

After Enqueue	*After Dequeue*	*Print*
v_2		v_2
....	....	

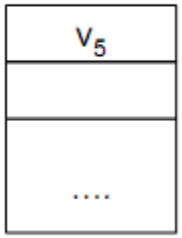


Topolojik Sıralama



Updated Indegrees	
v_1	0
v_2	0
v_3	1
v_4	1
v_5	0
v_6	3
v_7	2

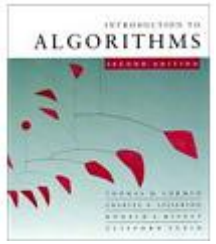
After Enqueue



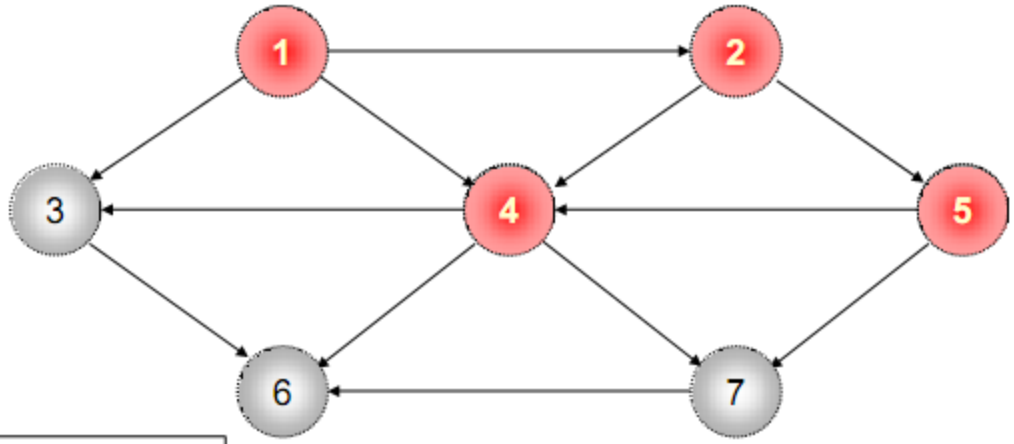
After Dequeue

Print

v_5

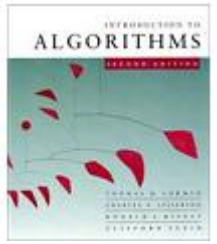


Topolojik Sıralama

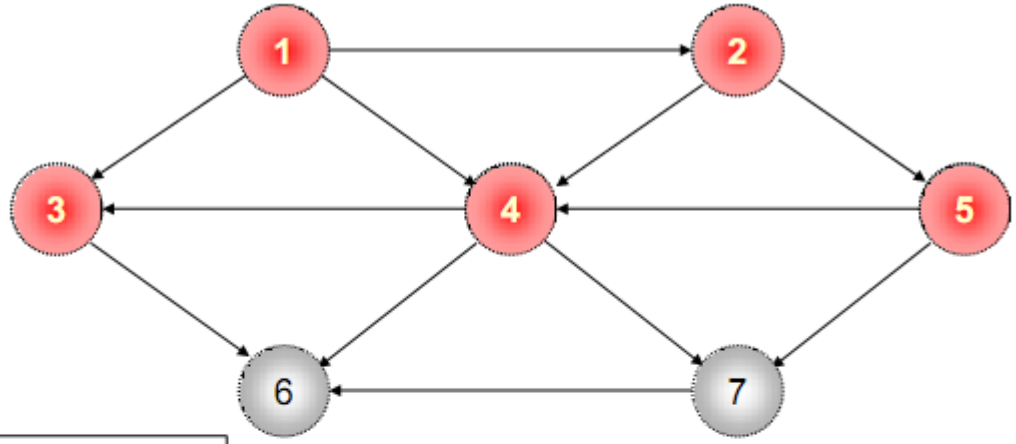


Updated Indegrees	
v_1	0
v_2	0
v_3	1
v_4	0
v_5	0
v_6	3
v_7	1

After Enqueue	*After Dequeue*	*Print*
v_4		v_4
....	....	

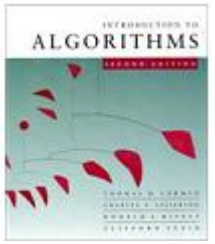


Topolojik Sıralama

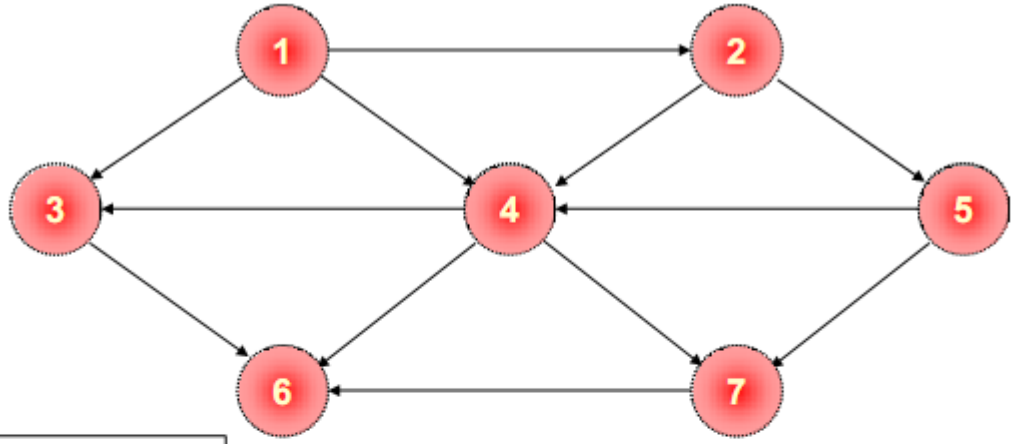


Updated Indegrees	
v_1	0
v_2	0
v_3	0
v_4	0
v_5	0
v_6	2
v_7	0

After Enqueue	*After Dequeue*	*Print*
v_3	v_7	v_3
v_7		
....	....	



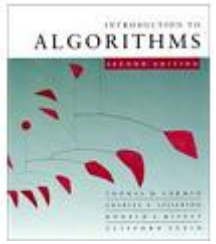
Topolojik Sıralama



Updated Indegrees	
v_1	0
v_2	0
v_3	0
v_4	0
v_5	0
v_6	0
v_7	0

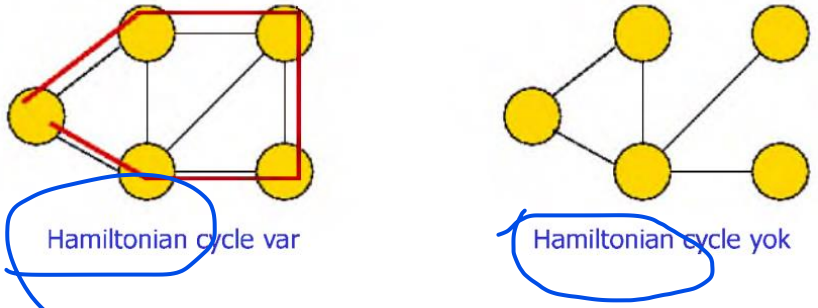
After Enqueue	*After Dequeue*	*Print*
v_6		v_6
....	....	

Finished! Result: 1,2,5,4,3,7,6

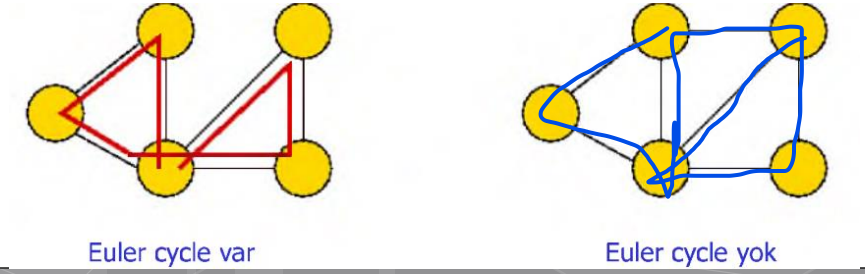


Hamiltonian ve Euler Cycle

- **Hamiltonian döngüsü**: Bir G grafında, başlangıç ve bitiş düğümleri hariç her düğüm bir kez bulunduğu yoldur.



- **Euler döngüsü**: Bir G grafında, bir düğümünden başlayıp yine kendisinde bitecek şekilde oluşturulan bir yoldur. Bu yolda bütün düğümler en az bir kez bulunur ancak her kenar mutlaka bir kez bulunur.



10.Hafta
Minimum kapsayan ağaçlar
Minimum spanning trees (MST)