

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.1

2020-2021 Güz Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

(SUBSTRiNG) ARAMA

Substring Arama

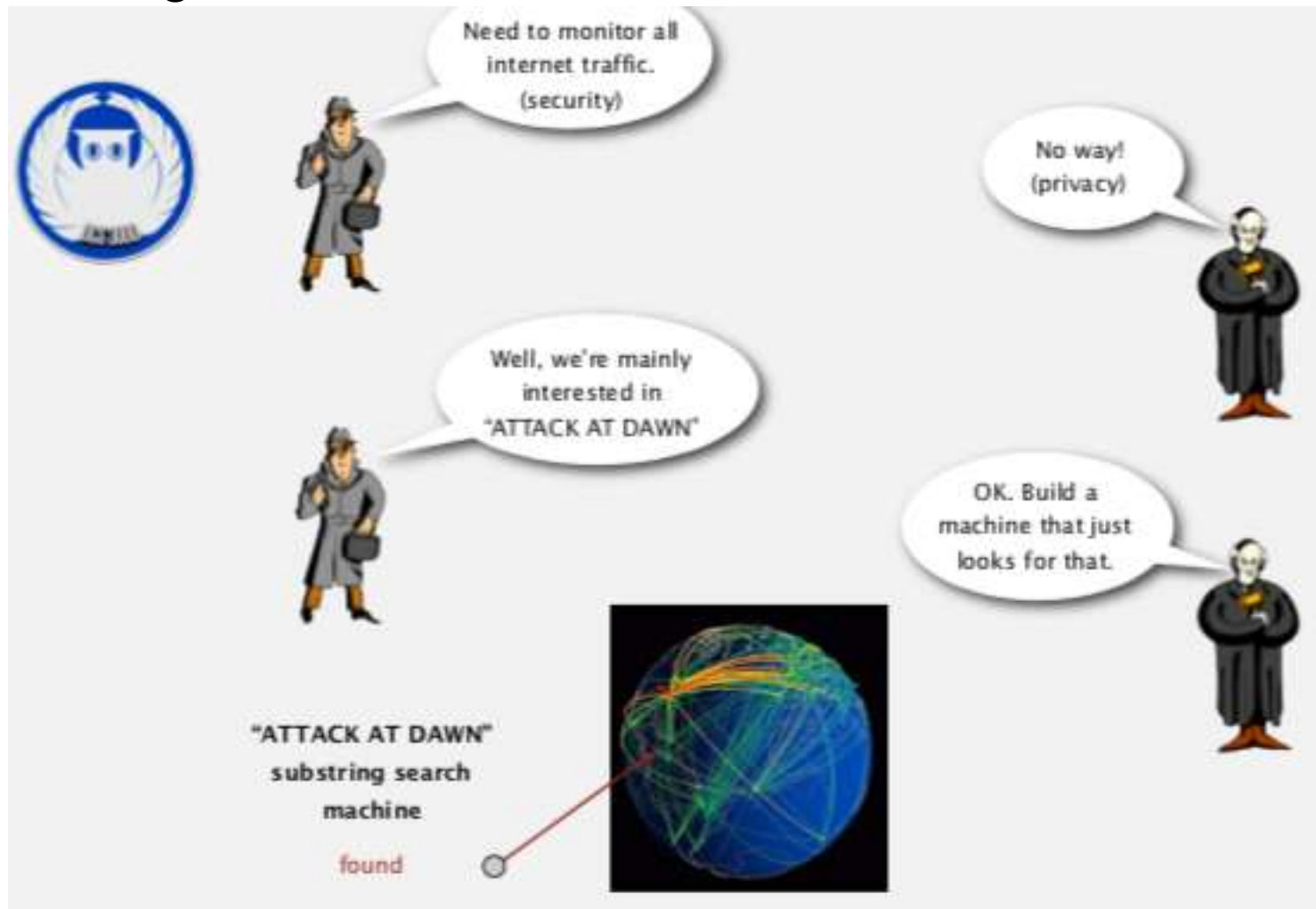
- Amaç, N boyutundaki bir metin içinde M boyutundaki bir metni (deseni) bulmak
- Genellikle, $N \gg M$
- Sonsuz uzunlukta akan veri de olabilir.
- pattern: **NEEDLE**
- text: ...INAHAYSTACK**NEEDLE**INAHAYSTACK...
- Çok büyük miktarda verinin içinde hızı ve verimli bir şekilde örüntüyü bulmamız gerekir.
 - Ör: Binlerce satır kod içinde «public» anahtar kelimesini arıyoruz.

String Arama Uygulamaları

- Web araması
- Veritabanı sorguları
- İntihal tespiti
- Adli Bilişim: Hafıza ya da diskte belirli imzaların aranması.
 - Ör. tüm URL'ler ya da RSA anahtarları...
- E-posta: Spam belirteci olan örüntüleri tanımlama
 - PROFITS
 - LOSE WE1GHT
 - herbal Viagra
 - There is no catch.
 - This is a one-time mailing.
 - This message is sent in compliance with spam regulations.

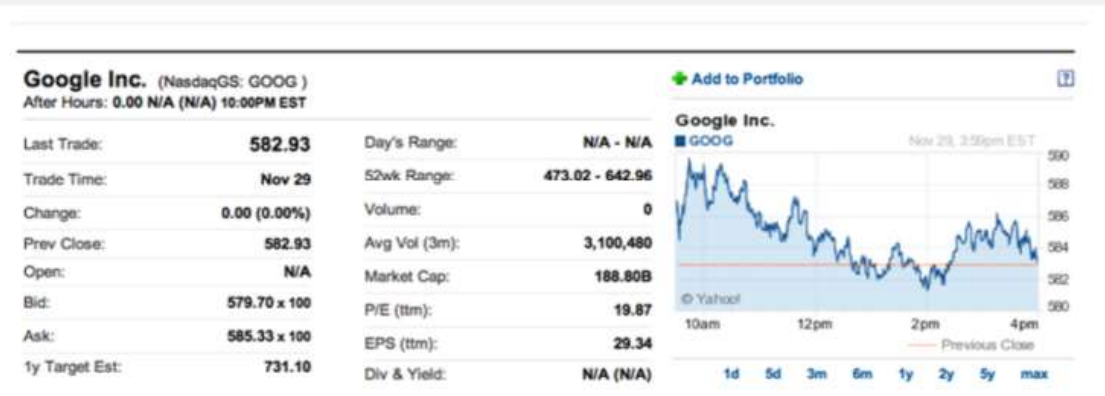
String Arama Uygulamaları

- Elektronik gözetim



String Arama Uygulamaları

- Webden bilgi çıkarımı



<http://finance.yahoo.com/q?s=goog>

```
...
<tr>
<td class= "yfnc_tablehead1"
width= "48%">
Last Trade:
</td>
<td class= "yfnc_tabledata1">
<big><b>452.92</b></big>
</td></tr>
<td class= "yfnc_tablehead1"
width= "48%">
Trade Time:
</td>
<td class= "yfnc_tabledata1">
...

```

SUBSTRING ARAMA YÖNTEMLERİ

Brute-Force Substring Arama

- Veri içinde aranan örüntünün başlangıcını bir kaydırarak ara
- Bulursan, örüntünün 2..m arası karakterlerini de eşleştirmeye çalış.
- Hepsi eşleşirse, bulduk 😊
- Eşleşmezse, bir kaydırıp aramaya devam et.
- Sona geldiysek, bulamadık 😞
- Javada `string.indexOf()` da bunu kullanır.

Brute-Force Substring Arama

NAIVE-STRING-MATCHER.T;P/

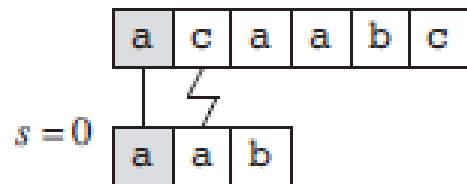
1 $n = T.length$

2 $m = P.length$

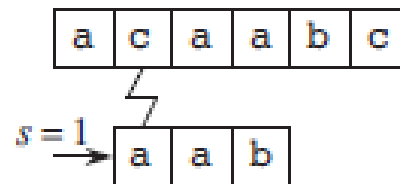
3 **for** $s = 0$ **to** $n - m$

4 **if** $P[1..m] == T[s + 1 .. s + m]$

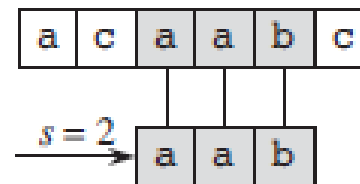
5 print “Pattern occurs with shift” s



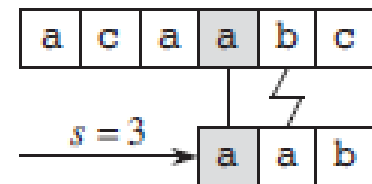
(a)



(b)

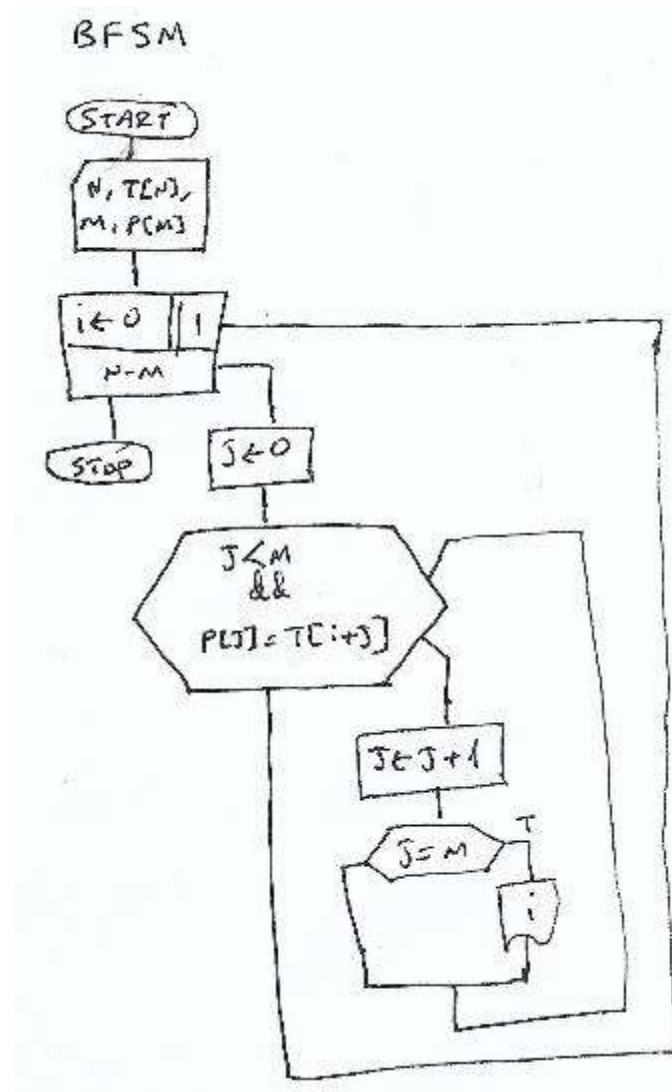


(c)



(d)

Brute-Force Substring Arama



Brute-Force Substring Arama

T	H	I	S		I	S		A		S	I	M	P	L	E		E	X	A	M	P	L	E
---	---	---	---	--	---	---	--	---	--	---	---	---	---	---	---	--	---	---	---	---	---	---	---

S	I	M	P	L	E																		
	S	I	M	P	L	E																	
		S	I	M	P	L	E																
			S	I	M	P	L	E															
				S	I	M	P	L	E														
					S	I	M	P	L	E													
						S	I	M	P	L	E												
							S	I	M	P	L	E											
								S	I	M	P	L	E										
									S	I	M	P	L	E									
										S	I	M	P	L	E								
											S	I	M	P	L	E							

Brute-Force Substring Arama

- En kötü durum: veri ve örüntü tekrarlı ise?
- $O(MN)$

AAAAB ?

i	j	i+j	txt	0	1	2	3	4	5	6	7	8	9
				A	A	A	A	A	A	A	A	A	B
0	4	4		A	A	A	A	B					
1	4	5			A	A	A	A	B				
2	4	6				A	A	A	A	B			
3	4	7					A	A	A	A	B		
4	4	8						A	A	A	A	B	
5	5	10							A	A	A	A	B

Brute-Force Substring Arama

- Gerçek hayattaki pek çok uygulamada, geriye dönüp kaldığımız yerden devam edemeyiz !!! (Akan veri)

A A A A A A A A A A A A **A** A A A A A A A A A A

A A A A A B

< = = = backup (bir sağa kaydır)

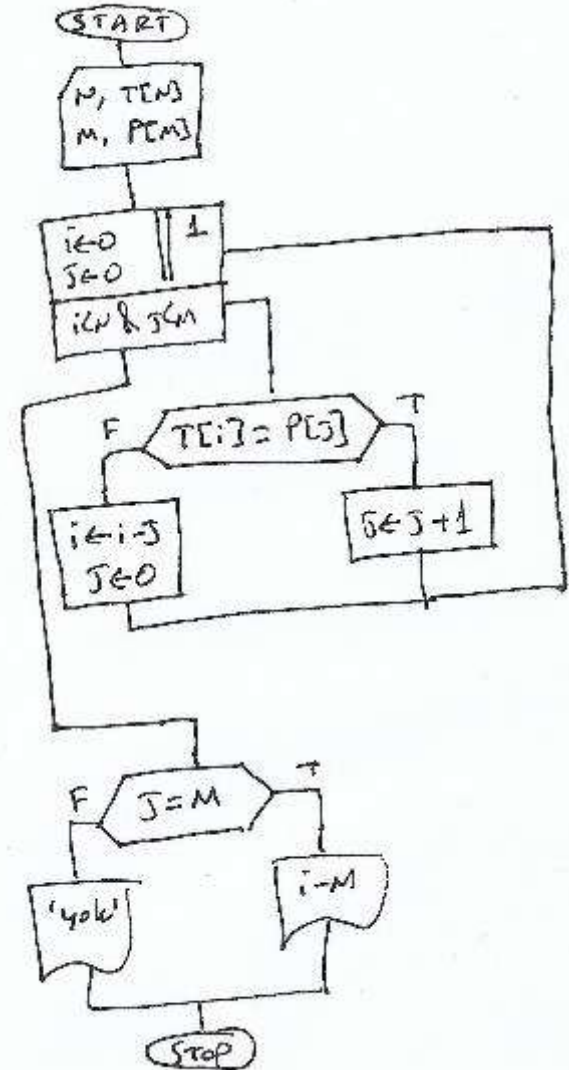
A A A A A B ...

- M karakteri önbellekte tutmamız gerekir.

Brute-Force Substring Arama/Backup

- Pratikte akan veride backup kadar bilgiyi saklayacak
 - Yerimiz
 - Zamanımız
 - İznimiz
- Olmayabilir !

BFSM BACKUP



Boyer-Moore-Horspool Algoritması

- Eşleşmeyen karakter sezgisi (Bad Character Heuristic) olarak da bilinir.
- Örüntüdeki karakterleri sağdan sola! tara
- Örüntüde olmayan bir karakter olduğunda, X adet karakteri geçebiliriz.
 - Brute Force'ta birer birer ilerleyebiliyorduk.
 - Sorun, X kaç olacak?
 - Bir eşleşmeyi kaçırmamayı garanti ederek, olası en büyük ilerlemeyi yapmaya çalışmalıyız.
- “Kötü Eşleşme Tablosu” “Bad Match Table” yaratılır. Buradaki miktar kadar kaydırma yapılır.
- Genellikle lineer zaman.
- En kötü durumda kuadratik.

Boyer-Moore-Horspool Algoritması

- Peki, *Bad Match Table* nasıl yaratılacak?
- **Durum 1:** Text'te karşılaştırılan karakter (c) Pattern'da yoktur.
 - Rahatlıkla Pattern'in boyu kadar kaydırabiliriz.

s_0 ... S ... s_{n-1}
X
B A R B E R
B A R B E R

Boyer-Moore-Horspool Algoritması

- Peki, *Bad Match Table* nasıl yaratılacak?
- **Durum 2:** Text'te karşılaştırılan karakter (c) Pattern'da var ama son karakter değil.
 - (c) karakterini, Pattern'deki **en sağ pozisyonu** ile eşleştirecek kadar kaydırırız (olası bir eşleşmeyi kaçırmayalım diye)

s_0 ... B ... s_{n-1}

X

B A R B E R

B A R B E R

Boyer-Moore-Horspool Algoritması

- Peki, *Bad Match Table* nasıl yaratılacak?
- **Durum 3:** Text'te karşılaştırılan karakter (c) Pattern'daki son karakter ve kalan m-1 karakter içinde başka tekrarı yok.
 - Durum 1'e benzer şekilde, pattern'in boyu kadar kaydırırız.

s_0 ... M E R ... s_{n-1}
 X || ||
 L E A D E R
 L E A D E R

Boyer-Moore-Horspool Algoritması

- Peki, *Bad Match Table* nasıl yaratılacak?
- **Durum 4:** Text'te karşılaştırılan karakter (c) Pattern'daki son karakter ve başka tekrarı da var.
 - Durum 2'ye benzer şekilde kalan $m-1$ karakter içinde c'nin **en sağda geçtiği yere kadar** kaydırırız.

s_0 ... A R ... s_{n-1}
 X ||
 R E O R D E R
 R E O R D E R

Boyer-Moore-Horspool Algoritması

- Brute-Force'taki teker teker kaydırmaya göre kesinlikle daha iyi !
- Ama her seferinde ne kadar kaydıracağımızı hesaplamak yerine, baştan hesaplayıp bir tablo yapabiliriz.
- **Kaydırma Tablosu (Bad Match Table yaratımı)**

ALGORITHM *ShiftTable*($P[0..m-1]$)

//Fills the shift table used by Horspool's and Boyer-Moore algorithms

//Input: Pattern $P[0..m-1]$ and an alphabet of possible characters

//Output: $Table[0..size-1]$ indexed by the alphabet's characters and

// filled with shift sizes.

for $i \leftarrow 0$ **to** $size - 1$ **do** $Table[i] \leftarrow m$ // all remaining characters

for $j \leftarrow 0$ **to** $m - 2$ **do** $Table[P[j]] \leftarrow m - 1 - j$

return $Table$

Boyer-Moore-Horspool Algoritması

- Horspool Eşleştirme Algoritması

ALGORITHM *HorspoolMatching*($P[0..m-1]$, $T[0..n-1]$)
// Implements Horspool's algorithm for string matching
// Input: Pattern $P[0..m-1]$ and text $T[0..n-1]$
// Output: The index of the left end of the first matching substring
// or -1 if there are no matches
ShiftTable($P[0..m-1]$) //generate *Table* of shifts
 $i \leftarrow m - 1$ //position of the pattern's right end
while $i \leq n - 1$ **do**
 $k \leftarrow 0$ //number of matched characters
 while $k \leq m - 1$ **and** $P[m - 1 - k] = T[i - k]$ **do**
 $k \leftarrow k + 1$
 if $k = m$
 return $i - m + 1$
 else $i \leftarrow i + \text{Table}[T[i]]$
return -1

Boyer-Moore-**Horspool** Algoritması

- **Horspool** Eşleştirme Algoritmasını özetlersek:
 1. m uzunluktaki pattern verildiğinde, text+pattern alfabesi için tabloyu oluştur.
 2. Pattern'i text'in başına hizala.
 3. Eşleşme bulana ya da text'in sonuna gidene kadar tekrarla:
 1. Pattern'in son karakterinden başlayarak m eşleşme bulana kadar veya eşleşmeme olana kadar text ile karşılaştır.
 2. Eşleşmeme olduğunda, pattern'i karakterin tablodaki değeri kadar sağa kaydır.

Boyer-Moore-Horspool Arama Örneği

BARBER için BMT: (İngiliz alfabesi harfleri ve boşluk:

character c	A	B	C	D	E	F	...	R	...	Z	_
shift $t(c)$	4	2	6	6	1	6	6	3	6	6	6

J I M _ S A W _ M E _ I N _ A _ B A R B E R S H O P
B A R B E R B A R B E R
B A R B E R B A R B E R
B A R B E R B A R B E R

Boyer-Moore-Horspool Algoritması

- Bad Match Table Nasıl yaratılacak?
- Değer = Uzunluk – 1 – index
 - (for $j \leftarrow 0$ to $m - 2$ do $Table[P[j]] \leftarrow m - 1 - j$)
 - Uzunluk: aranan pattern'ın uzunluğu
 - Index: Pattern içinde karakterin indisi (0'dan başlayarak)
 - Pattern'de olmayan tüm karakterlerin değeri: Uzunluk
- **T O O T H** **Len=5**
- **0 1 2 3 4**

Bad Match Table:

Harf	T	O	H	*
Değer	5-0-1 = 4 5-3-1 = 1	5-1-1 = 3 5-2-1 = 2	5 (0→5) son karakter	5 kalan tüm karakterler

Boyer-Moore-Horspool Algoritması

- Bad Match Table:

Harf	T	O	H	*
Değer	1	2	5	5

- T R U S **T** H A R D T O O T H B R U S H E S
- T O O T **H** $T \neq H \rightarrow T:1$ kaydır

Boyer-Moore-Horspool Algoritması

- Bad Match Table:

Harf	T	O	H	*
Değer	1	2	5	5

- T R U S T H A R D T O O T H B R U S H E S
-
- T O O T H H ok, T ok, S!=O → S:5 kaydır
-

Boyer-Moore-Horspool Algoritması

- Bad Match Table:

Harf	T	O	H	*
Değer	1	2	5	5

- T R U S T H A R D T O O T H B R U S H E S

-

-

- T O O T H O != H → O:2 kaydır

-

Boyer-Moore-Horspool Algoritması

- Bad Match Table:

Harf	T	O	H	*
Değer	1	2	5	5

- T R U S T H A R D T O O T H B R U S H E S

-

-

-

- T O O T H T != H → T:1 kaydır

-

Boyer-Moore-Horspool Algoritması

- Bad Match Table:

Harf	T	O	H	*
Değer	1	2	5	5

• T R U S T H A R D T O O T H B R U S H E S

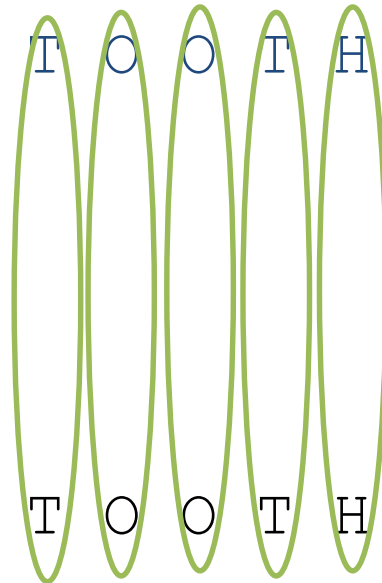
•

•

•

•

•



BULUNDU

Boyer-Moore-Horspool Algoritması

- En kötü durum:
- Input (n) : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- Pattern(m): BAAAAAA
- En kötü durum:
- $O(nm)$
- En iyi m/n (hep m adım ileri)
- Ortalama $m/\text{len}(\text{alfabe})$
- Esasen Horspool, Öncüsü olan Boyer-Moore'un basitleştirilmiş halidir.

Boyer-Moore Algoritması

- Horspool'un çıkış noktasıdır.
 - Ancak daha karmaşıktır.
- Horspool'dan farkı, eşleşmeyen karakter sezgisinin (Bad Match) üzerine bir de iyi sonek sezgisi (Good Suffix Heuristic) kullanmasıdır.
- Her durumda hangisi daha kârlı ise onu kullanır.

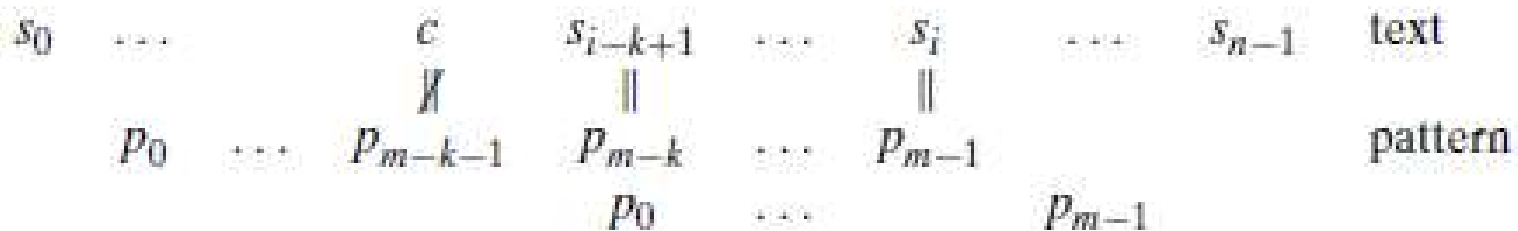
Boyer-Moore Algoritması

- Eğer pattern'in son karakteri text'teki (c) karakteri ile eşleşmezse, algoritma tamamen Horspool ile aynı davranır.
 - Bad Match Table: pattern, tablodaki harfin değeri kadar sağa kaydırılır.
- Sağdan k karakter eşleştiğinde ($0 < k < m$) ise farklı davranır.

s_0	...	c	s_{i-k+1}	...	s_i	...	s_{n-1}	text
		\neq	\parallel		\parallel			
p_0	...	p_{m-k-1}	p_{m-k}	...	p_{m-1}			pattern

Boyer-Moore Algoritması

- Bu durumda, algoritma 2 değere bakarak kaydırma miktarını belirler.
 - Eşleşmeyen (c) karakteri bad match tablomuzda yoksa, pattern'i $c+1$ 'e kaydırır. (**bad symbol shift**)
 - $t_1(c) - k$ ile (tablo değeri – eşleşmeyen karakter sayısı) kaydırır.



Boyer-Moore Algoritması

- Örneğin: S pattern'de yok, $t_1(S) - 2 = 6 - 2 = 4$ kaydır
- (Horspool'da R:3 kaydırmıştık)

```

s0  ...      S E R      ...  s_{n-1}
              X || ||
            B A R B E R
              B A R B E R
    
```

- A pattern'de var, $t_1(A) - 2 = 4 - 2 = 2$ kaydır.

```

s0  ...      A E R      ...  s_{n-1}
              X || ||
            B A R B E R
              B A R B E R
    
```

Boyer-Moore Algoritması

- $t_1(c) - k \leq 0$ olabilir.
- Bu durumda pattern'i geri çekemeyeceğimize göre, brute-force gibi 1 ilerletiriz.
- Kaydırma kararı (bad-match table güncellemesi):
- $d_1 = \max(1, t_1(c) - k)$

Boyer-Moore Algoritması

- 2. kaydırma türü: Good-Suffix-Shift
- Sondan $k > 0$ tane karakter eşleşirse:
 - k boyutunda son ek $\rightarrow \text{suff}(k)$
 - Bad Match Table'da olduğu gibi, c eşleşmeyen karakterine göre pattern'deki $(1, \dots, m-1)$ boyutundaki tüm suffix'ler ile Good-Suffix-Shift tablosunu doldururuz.
 - Önündeki karakteri farklı olan bir $\text{suff}(k)$ 'nin pattern içinde olduğunu varsayalım. (Ör. **ABCBAB**) Bu durumda, patternimizi k uzunluktaki suffix'in tekrarını geçecek kadar (d_2 uzunluğu kadar) kaydırabiliriz.

k	pattern	d_2
1	ABCB <u>AB</u>	2
2	<u>AB</u> CBAB	4

Boyer-Moore Algoritması

- suff(k)'nin, tekrarı yoksa?
 - İlk akla gelen, pattern'in tamamı kadar kaydırmaktır.
 - Ör: DBCBAB, k=3

$$\begin{array}{ccccccccccc}
 s_0 & & & & & c & B & A & B & & & \dots & s_{n-1} \\
 & & & & & X & \parallel & \parallel & \parallel & & & & \\
 & & D & B & C & B & A & B & & & & & \\
 & & & & & & & & D & B & C & B & A & B
 \end{array}$$

- Ancak, bu her zaman doğru olmayabilir.
- Ör. ABCBAB, $k=3 \rightarrow$ ya kaydığımız yer CBAB... ile başlıyorsa?
- Eşleşmeyi kaçırmız.

$$\begin{array}{cccccccccccccccc}
 s_0 & & & & & & c & B & A & B & C & B & A & B & & & \dots & s_{n-1} \\
 & & & & & & // & || & || & || & & & & & & & & \\
 & & & & A & B & C & B & A & B & & & & & & & & \\
 & & & & & & & & & & A & B & C & B & A & B & &
 \end{array}$$

Boyer-Moore Algoritması

- Farkettiyseniz, ABCBAB örneğinde $k=1$ ve $k=2$ için tekrar eden suffix var, bundan sonraki k 'lar için tehlike oluşuyor.
- Bu durumda en uzun prefix l ($l < k$) sonrasındaki tüm suffix'leri, sadece l kadar kaydırmalıyız.
- Ör: ABCBAB:

k	pattern	d_2
1	ABC <u>B</u> AB	2
2	<u>AB</u> CBAB	4
3	<u>ABC</u> BAB	4
4	<u>ABCB</u> AB	4
5	<u>ABCBAB</u>	4

Boyer-Moore Algoritması

- Bad-Match-Table ve Good-Suffix Table var.
- Ne kadar kaydıracağımıza nasıl karar vereceğiz?
- Hangisi daha kârlıysa, o kadar.

$$d = \begin{cases} d_1 & \text{if } k = 0, \\ \max\{d_1, d_2\} & \text{if } k > 0, \end{cases}$$

$$d_1 = \max\{t_1(c) - k, 1\}$$

Boyer-Moore Algoritması-Örnek

- Pattern: BAOBAB
- Text: BESS_KNEW_ABOUT_BAOBABS

- Bad-Match Table:

c	A	B	C	D	...	O	...	Z	_
$t_1(c)$	1	2	6	6	6	3	6	6	6

- Good-Suffix Table:

k	pattern	d_2
1	<u>BAOBAB</u>	2
2	<u>BAOBAB</u>	5
3	<u>BAOBAB</u>	5
4	<u>BAOBAB</u>	5
5	<u>BAOBAB</u>	5

Boyer-Moore Algoritması-Örnek

B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B

$$d_1 = t_1(K) - 0 = 6$$

Boyer-Moore Algoritması-Örnek

B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B

B A O B A B

$$d_1 = t_1(_) - 2 = 4$$

$$d_2 = 5$$

$$\max(4, 5) = 5$$

Boyer-Moore Algoritması-Örnek

B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B
B A O B A B
B A O B A B

$$d_1 = T_1(_) - 1 = 5$$

$$d_2 = 2$$

$$\max(5, 2) = 5$$

Boyer-Moore Algoritması-Örnek

B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B
B A O B A B
B A O B A B

The diagram illustrates the Boyer-Moore algorithm's matching process. It shows four rows of characters. The first row is 'B E S S _ K N E W _ A B O U T _ B A O B A B S'. The second row is 'B A O B A B'. The third row is 'B A O B A B'. The fourth row is 'B A O B A B'. Red ovals highlight mismatches: the 'K' in the first row with the 'B' in the second row, the '_' in the first row with the 'B' in the second row, and the '_' in the first row with the 'A' in the third row. Green ovals highlight matches: the 'A' and 'B' in the first row with the 'A' and 'B' in the second row, and the 'B A O B A B' sequence in the first row with the 'B A O B A B' sequence in the third and fourth rows.

Rabin-Karp Algoritması

- Metni direk aramak yerine, imzasını arama fikri
- İmza? Pattern'e ait bir hash fonksiyonu çıktısı
 - Hash çıktıları çakışabilir!
 - Bu durumda Text içinde hash'i eşleşen parça pattern ile birebir karşılaştırılır.
 - Çoklu aramalarda iyi çalışır.
- Karmaşıklık:
 - Ön işleme: $O(m)$
 - En kötü: $O(mn)$
 - Genellikle $(m+n)$

Rabin-Karp Algoritması

- Her metin pozisyonu için (kaydırmalı olarak) hash değeri hesaplanır ve pattern'in hash'i ile karşılaştırılır.
- Eşleşen hash bulunduğunda pattern ile text karşılaştırılır.
- Hash'in çakışma olasılığını azaltmak için fonksiyonda büyük asal sayılar tercih edilir.

Örnek: 59265 → 31415926535897932384626433

pattern hash: $59265 = 95 \pmod{97}$

text hashes: 31415926535897932384626433

$$31415 = 84 \pmod{97}$$

$$14159 = 94 \pmod{97}$$

$$41592 = 76 \pmod{97}$$

$$15926 = 18 \pmod{97}$$

$$59265 = 95 \pmod{97}$$

Rabin-Karp Algoritması

- Önceki hash değeri kullanılarak bir adım sonraki hesaplanabilir.

$$\begin{aligned} 1415\textcircled{9} &= (31415 - 30000) * 10 + \textcircled{9} \\ 14159 \bmod 97 &= (31415 \bmod 97 - 30000 \bmod 97) * 10 + 9 \bmod 97 \\ &= (\underset{\substack{\downarrow \text{known from} \\ \text{previous position}}}{84} - \underset{\substack{\swarrow \text{precompute } 9 = 10000 \bmod 97}}{3*9}}) * 10 + 9 \bmod 97 \\ &= 579 \bmod 97 = 94 \end{aligned}$$

Key point: all ops involve small numbers
No restriction on N and M

pattern hash: $59265 = 95 \pmod{97}$

text hashes

31415926535897932384626433:

$31415 \bmod 97 = 84$

$14159 \bmod 97 = (84 - 3*9) * 10 + 9 \pmod{97} = 94$

$41592 \bmod 97 = (94 - 1*9) * 10 + 2 \pmod{97} = 76$

$15926 \bmod 97 = (76 - 4*9) * 10 + 6 \pmod{97} = 18$

$59265 \bmod 97 = (18 - 1*9) * 10 + 5 \pmod{97} = 95$

Substring Arama Algoritmalarının Maliyet Karşılaştırması

algorithm	version	operation count		backup in input?	correct?	extra space
		guarantee	typical			
<i>brute force</i>	—	MN	$1.1 N$	<i>yes</i>	<i>yes</i>	1
<i>Knuth-Morris-Pratt</i>	<i>full DFA</i> (Algorithm 5.6)	$2 N$	$1.1 N$	<i>no</i>	<i>yes</i>	MR
	<i>mismatch</i> <i>transitions only</i>	$3 N$	$1.1 N$	<i>no</i>	<i>yes</i>	M
	<i>full algorithm</i>	$3 N$	N / M	<i>yes</i>	<i>yes</i>	R
<i>Boyer-Moore</i>	<i>mismatched char</i> <i>heuristic only</i> (Algorithm 5.7)	MN	N / M	<i>yes</i>	<i>yes</i>	R
<i>Rabin-Karp</i> [†]	<i>Monte Carlo</i> (Algorithm 5.8)	$7 N$	$7 N$	<i>no</i>	<i>yes</i> [†]	1
	<i>Las Vegas</i>	$7 N$ [†]	$7 N$	<i>yes</i>	<i>yes</i>	1

[†] probabilistic guarantee, with uniform and independent hash function

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.1

2020-2021 Güz Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

AÇGÖZLÜ ALGORİTMALAR

Greedy Algorithms

Greedy Algoritmalar

- Probleme, bir dizi seçimden geçerek parça parça çözüm oluştururlar. Bu seçimler:

- yapılabilir (feasible)
- yerel en iyi (locally optimal)
- geri alınamaz (irrevocable)

olmalıdır.

- Bazı problemlerin tüm örnekleri için optimal çözüm sunarlar.
- Çoğu problem için bunu başaramasa da, hızlı yakınsama nedeniyle tercih edilirler.

Greedy Algoritma Örnekleri

- Optimal çözümler:
 - Bozuk para üstü vermek.
 - Minimum spanning tree (göreceğiz)
 - Single-source shortest paths
 - Basit çizelgeleme problemleri
 - Huffman kodlama
- Yaklaşımlar:
 - Gezgin satıcı problemi (Traveling salesman)
 - Sırt çantası problemi (Knapsack)
 - Diğer kombinatorial optimizasyon problemleri

Bozuk Para Üstü Vermek

- Bozuk paralarımız: 1TL, 50Kr, 25Kr, 10 Kr, 5 Kr, 1 Kr ? :))
- 48 kuruş para üstü verelim.
 - 1 tane 25 Kr. (23 Kr kaldı)
 - 1 tane 10 Kr. (13 Kr kaldı)
 - 1 tane 10 Kr. (3 Kr kaldı)
 - 1 tane 1 Kr (2 Kr kaldı)
 - 1 tane 1 Kr (1 Kr kaldı)
 - 1 tane 1 Kr (0 Kr kaldı)
- Her adımda, kalan miktarı en az yapacak olan bozuk parayı ver.

Huffman Kodlama

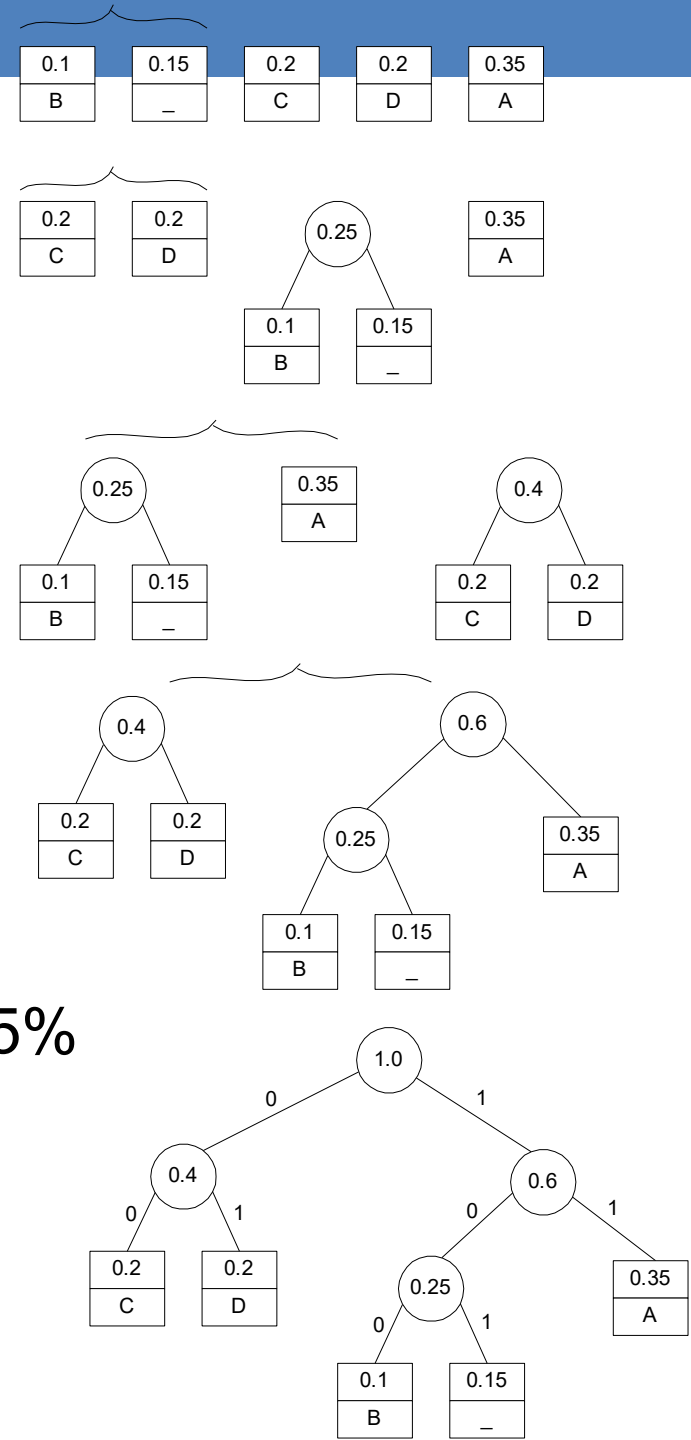
karakter A B C D _
frekans 0.35 0.1 0.2 0.2 0.15

Kod 11 100 00 01 101

Karakter başına ortalama bit: 2.25

Sabit uzunluklu kodlama için: 3

Sıkıştırma oranı: $(3-2.25)/3 \cdot 100\% = 25\%$



BÖL&YÖNET

Divide-and-Conquer

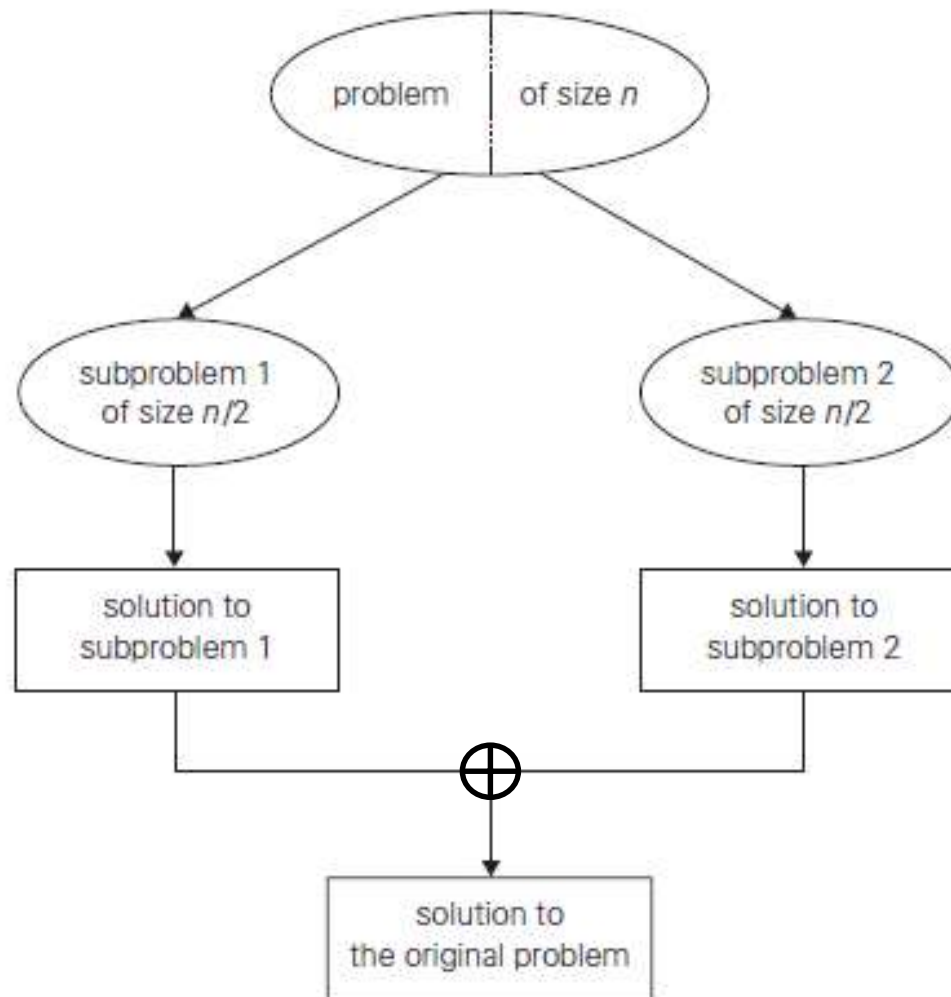
Böl&Yönet

- Şimdiye kadar gördüğümüz algoritmik çözüm yöntemleri:
 - **Brute-Force:** Başlangıçtan dümdüz ilerleyerek çözüme doğru git.
 - Üs Alma $a^n = a * a * \dots * a * a$ (n kere)
 - Selection Sort
 - Bubble Sort
 - Brute-Force Substring Search
 - **Greedy (Açgözlü):** Her adımda, o an için en iyi olan çözümü (local optimum) uygula
 - Huffman Encoding
 - **Decrease-and-Conquer:** Problemi küçülterek çözüme doğru ilerle.
 - Üs Alma $a^n = a^{n-1} * a$
 - Insertion Sort
 - Josephus Problem (Circular Linked List)

Böl&Yönet

- Problem (genellikle eşit boyutlu) aynı tipte küçük problemlere ayrıştırılır.
- Alt problemler çözülür.
 - Genellikle rekürsif olarak daha küçük problemler haline getirilir.
 - Bölünemeyecek kadar küçük problem bir yöntem ile çözülür.
- Gerektiğinde, alt problemlerin çözümleri bir araya getirilerek orijinal probleme ait nihai çözüm oluşturulur.
- Her zaman brute-force yaklaşımdan daha efektif çözüm olacak diye bir şart yoktur.
- Kolaylıkla *paralel hesaplama* şeklinde gerçekleştirilebilir.

Böl&Yönet

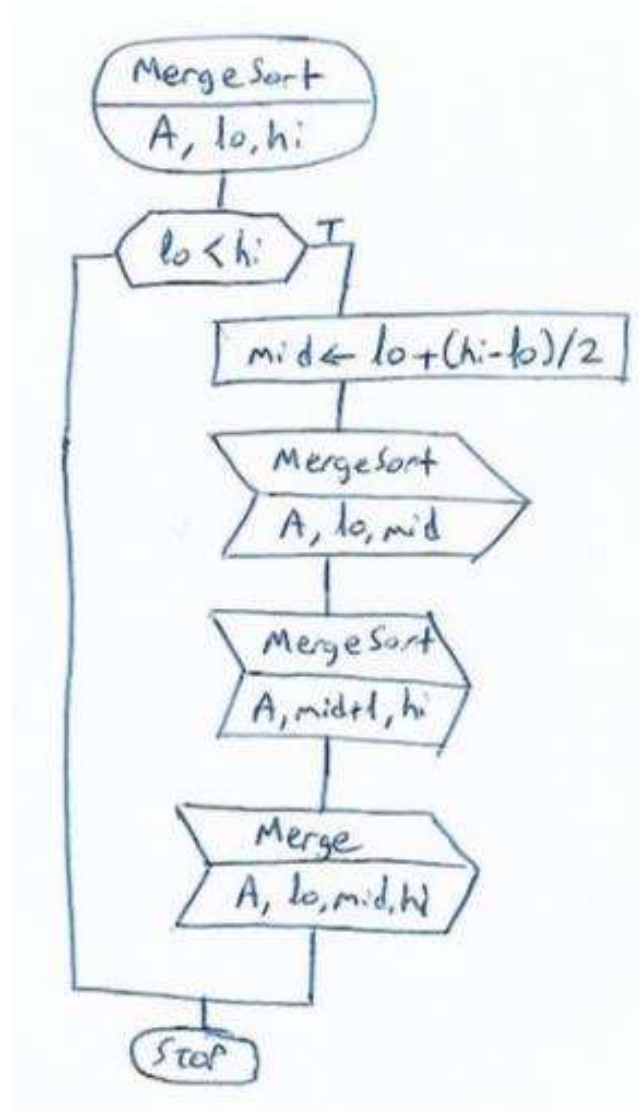


MERGESORT

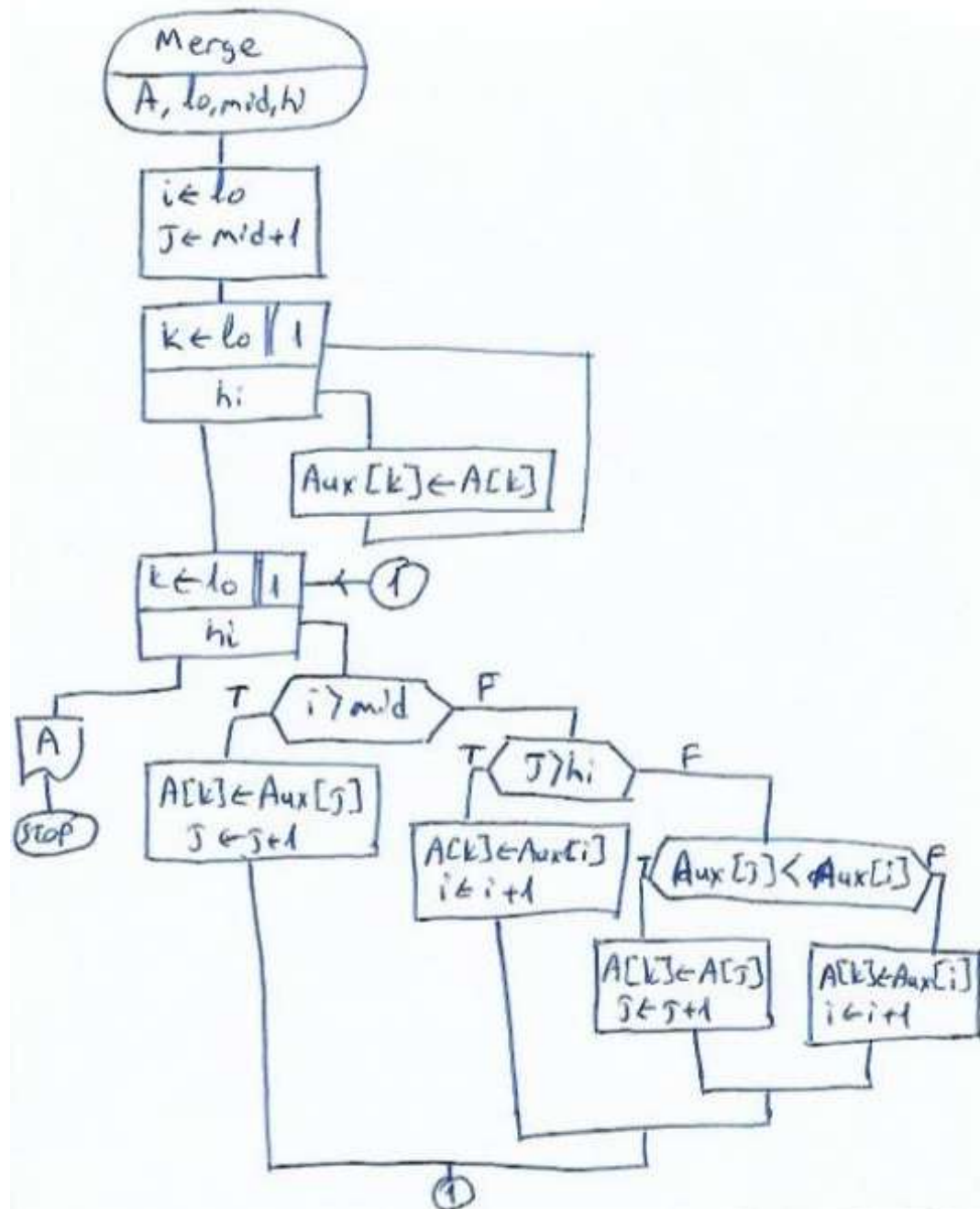
Merge Sort

- Elimizde $A[0..n-1]$ dizisi var.
- Bu diziyi böl&yönet ile sıralayabilir miyiz?
- Diziyi ikiye böl, her parçayı kendi içinde sırala.
 - Rekürsif olarak parçaları kendi içinde ikiye bölmeye devam et.
 - Algorithm MergeSort
- Tek elemanlık parçalar, zaten sıralı demektir.
- Sonra, her sıralı alt-alt parçayı sıralı olarak birleştir.
- Parçalar birleştikçe onları da sıralı olarak birleştir.
 - Algorithm Merge

Merge Sort (Top-Down MergeSort)



Merge



Abstract in-place merge trace

[illegible]

MergeSort

ALGORITHM *Mergesort*($A[0..n - 1]$)

//Sorts array $A[0..n - 1]$ by recursive mergesort

//Input: An array $A[0..n - 1]$ of orderable elements

//Output: Array $A[0..n - 1]$ sorted in nondecreasing order

if $n > 1$

 copy $A[0..n/2 - 1]$ to $B[0..n/2 - 1]$

 copy $A[n/2..n - 1]$ to $C[0..n/2 - 1]$

Mergesort($B[0..n/2 - 1]$)

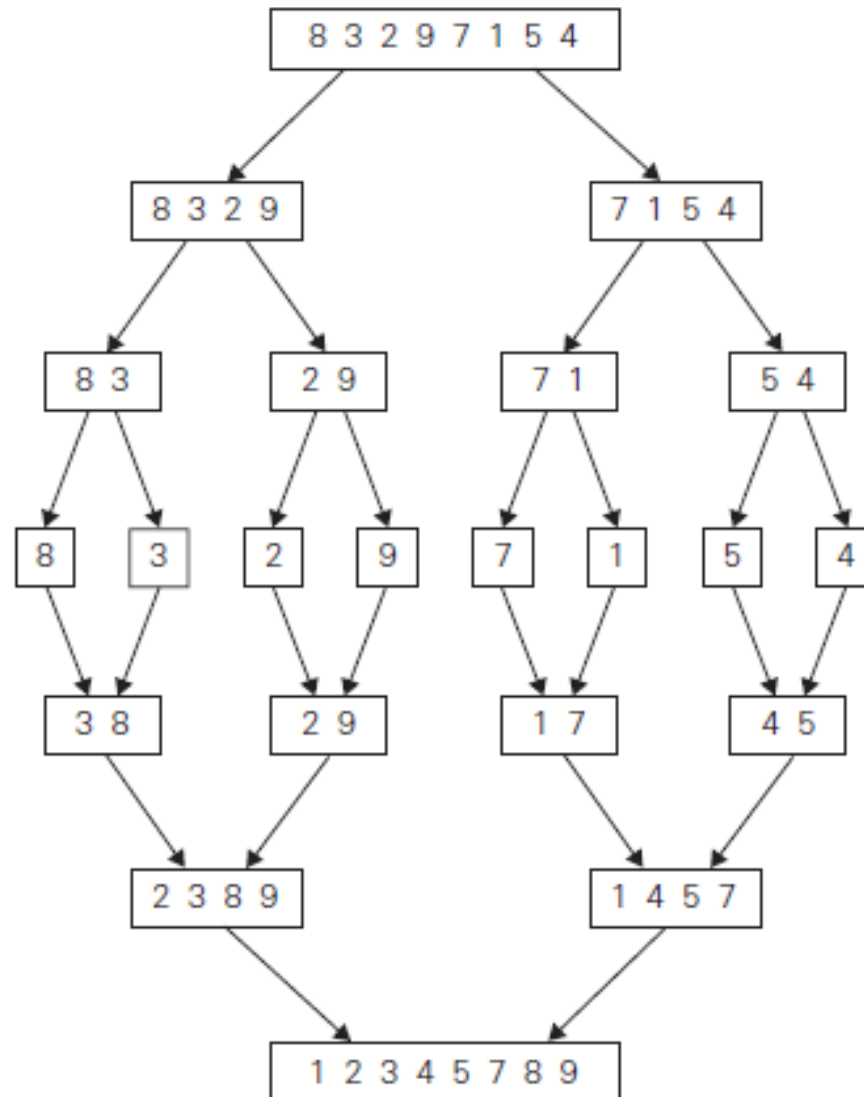
Mergesort($C[0..n/2 - 1]$)

Merge(B, C, A)

Merge

ALGORITHM *Merge*($B[0..p-1]$, $C[0..q-1]$, $A[0..p+q-1]$)
//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
 $i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
 if $B[i] \leq C[j]$
 $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$
 else
 $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$
 $k \leftarrow k + 1$
if $i = p$
 copy $C[j..q-1]$ to $A[k..p+q-1]$
else
 copy $B[i..p-1]$ to $A[k..p+q-1]$

Merge Sort Örnek-1



MergeSort Örnek-2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
1	E	M														
2			G	R												
3	E	G	M	R												
4					E	S										
5							O	R								
6					E	O	R	S								
7	E	E	G	M	O	R	R	S								

MergeSort Örnek-2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	E	G	M	O	R	R	S								
8								E	T						
9										A	X				
10								A	E	T	X				
11												M	P		
12														E	L
13												E	L	M	P
14								A	E	E	L	M	P	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

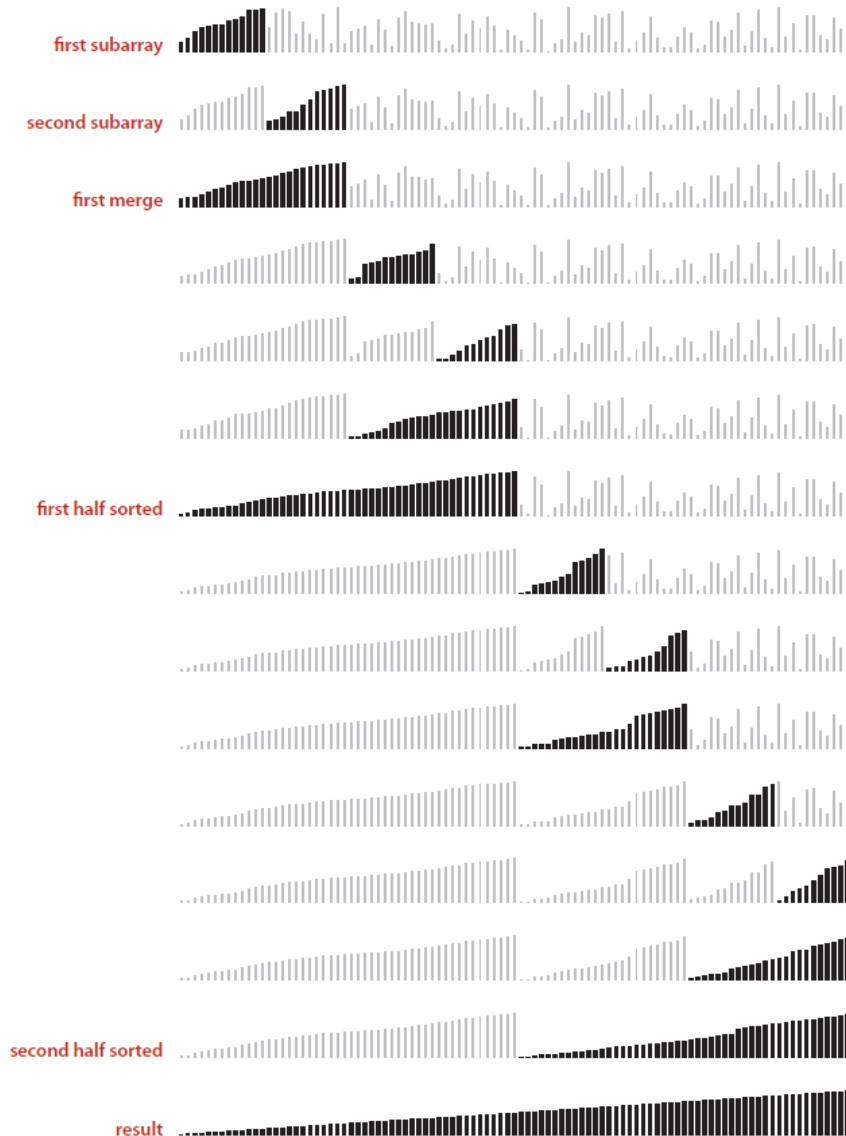
MergeSort Örnek-2 Analiz

- MergeSort(A, 0, 15)
 - MergeSort(A, 0, 7)
 - MergeSort(A, 0, 3)
 - MergeSort(A, 0, 1)
 - Merge(A, 0, 0, 1)
 - MergeSort(A, 2, 3)
 - Merge(A, 2, 2, 3)
 - Merge(A, 0, 1, 3)
 - MergeSort(A, 4, 7)
 - MergeSort(A, 4, 5)
 - Merge(A, 4, 4, 5)
 - MergeSort(A, 6, 7)
 - Merge(A, 6, 6, 7)
 - Merge(A, 4, 5, 7)
 - Merge(A, 0, 3, 7)
 - MergeSort(A, 8, 15)
 - MergeSort(A, 8, 11)
 - MergeSort(A, 8, 9)
 - Merge(A, 8, 8, 9)
 - MergeSort(A, 10, 11)
 - Merge(A, 10, 10, 11)
 - Merge(A, 8, 9, 11)
 - MergeSort(A, 12, 15)
 - MergeSort(A, 12, 13)
 - Merge(A, 12, 12, 13)
 - MergeSort(A, 14, 15)
 - Merge(A, 14, 14, 15)
 - Merge(A, 12, 13, 15)
 - Merge(A, 8, 11, 15)
 - Merge(A, 0, 7, 15)

MergeSort Karmaşıklık Analizi

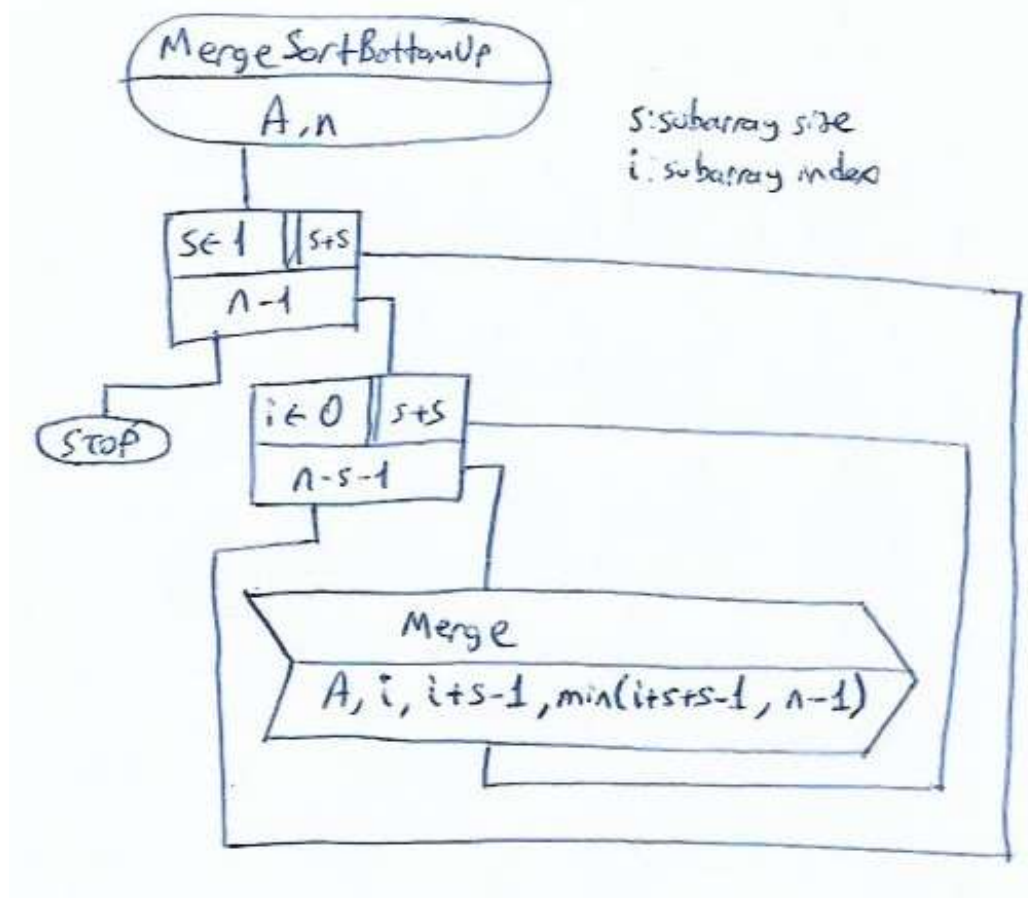
- $O(N \lg N)$
- - $A[0..15]$
 - $A[0..7]$ $A[8..15]$
 - $A[0..3]$ $A[4..7]$ $A[8..11]$ $A[12..15]$
 - $A[0,1]$ $A[2,3]$ $A[14,15]$
- N Seviye, $\lg N$ adım
- $k=0..n-1$
- k.seviyede 2^k alt dizi, uzunlukları 2^{n-k}
- 2^{n-k} karşılaştırma
- $2^k \cdot 2^{n-k} = 2^n$ işlem (her n için)
- $n \cdot 2^n \rightarrow O(N \log N)$
 - ($6N \log N$ dizi erişimi: Her merge'de $2N$ kopya, $2N$ geri taşıma, $2N$ karşılaştırma)

Top-Down MergeSort Görselleştirme



Merge Sort (Bottom-Up MergeSort)

- 1-1, 2-2, 4-4, ... $N/2$ - $N/2$ birleştir
- $1/2 N \lg N$ – $6 N \lg N$ arası karşılaştırma, $6 N \lg N$ dizi erişimi



MergeSort Bottom-Up Örnek

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
1	E	M														
2			G	R												
3					E	S										
4							O	R								
5									E	T						
6											A	X				
7													M	P		
8															E	L

$S=1$

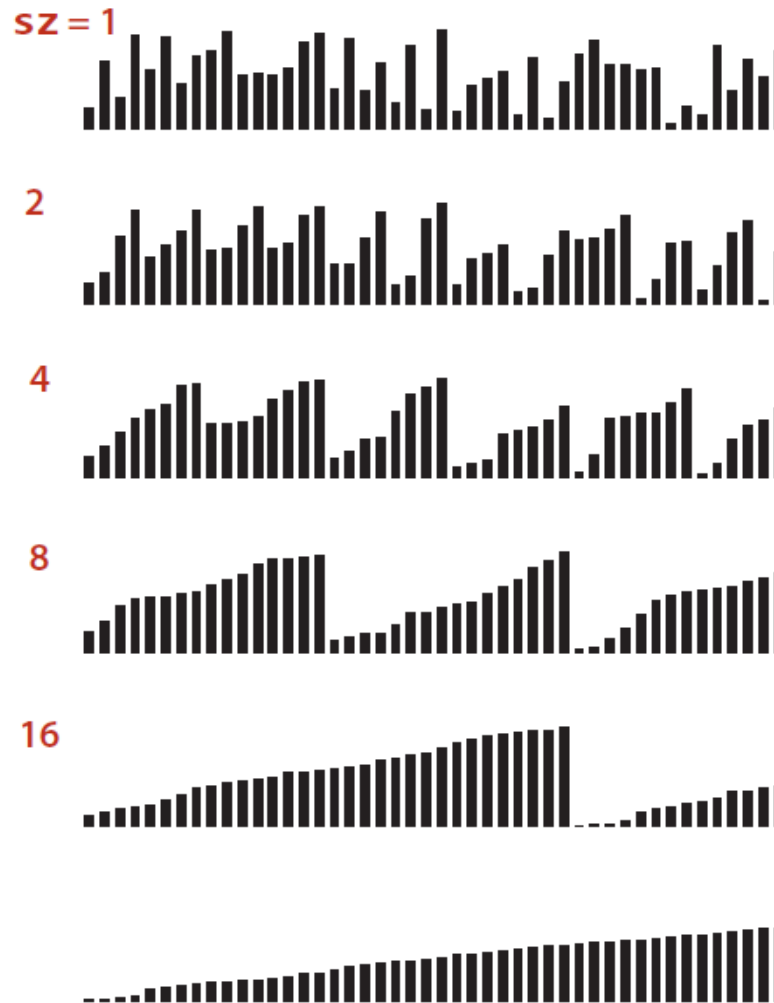
MergeSort Bottom-Up Örnek

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
9	E	G	M	R												
10					E	O	R	S								
11									A	E	T	X				
12													E	L	M	P
13	E	E	G	M	O	R	R	S								
14									A	E	E	L	M	P	T	X
15	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom-Up MergeSort Analiz

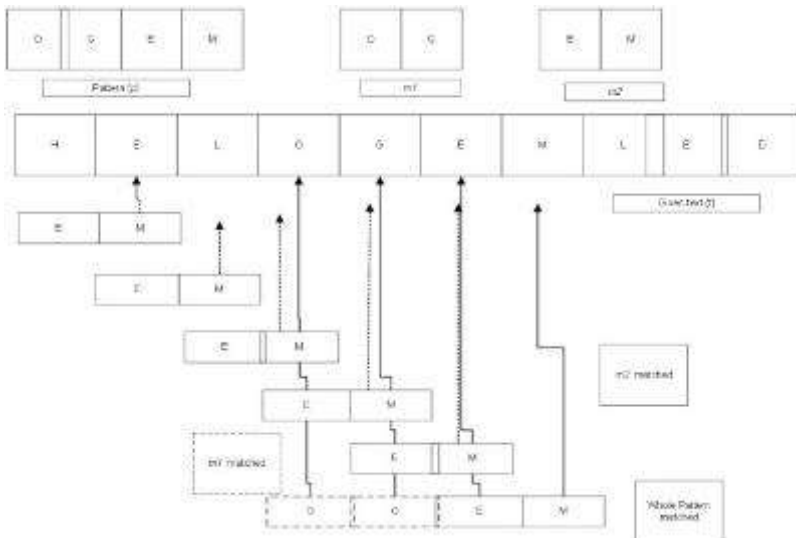
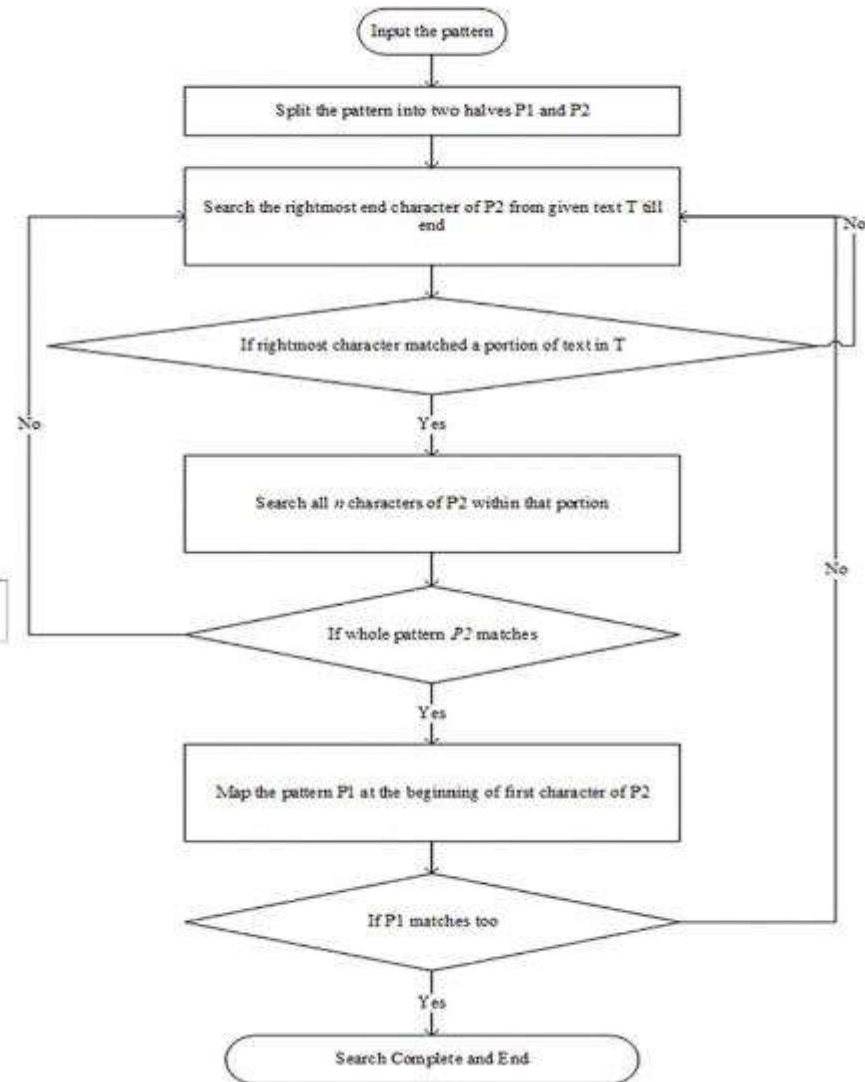
	a[i]															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
S=1																
merge(a, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
S=2																
merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a, 4, 5, 7)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
merge(a, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
S=4																
merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
S=8																
merge(a, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom-Up MergeSort Görselleştirme



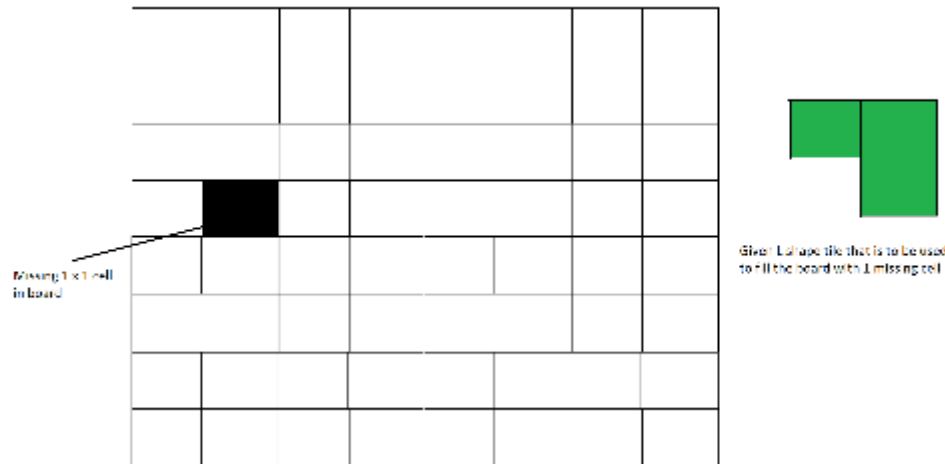
Böl&Yönet Substring Arama?

- Text i bölebilir miyiz?
- Pattern'i?

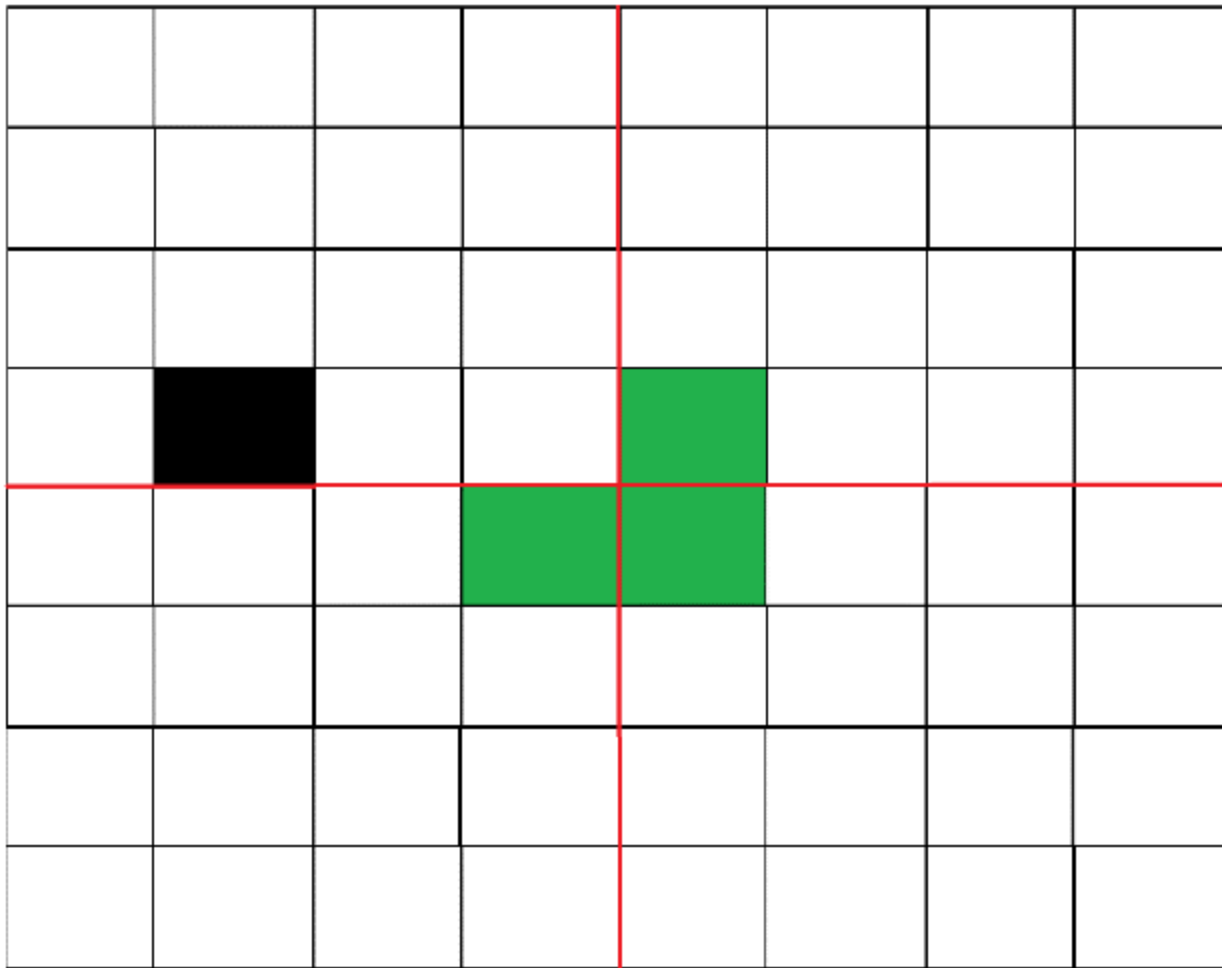


Böl&Yönet – Tromino Puzzle

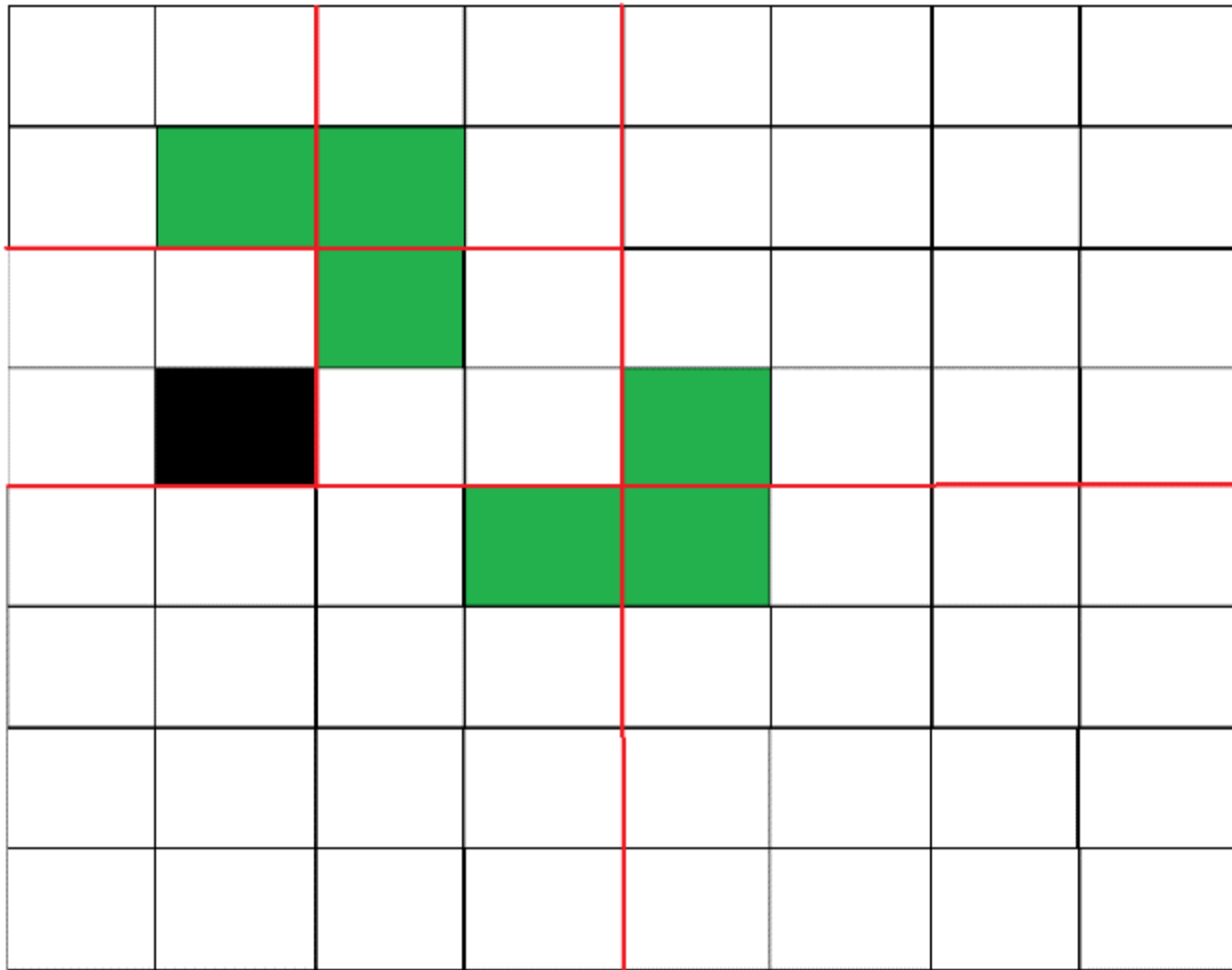
- Elimizde $2^n \times 2^n$ boyutlarında bir kare pano var. Sadece tek bir gözü dolu.
- 3 tane 1×1 kareden oluşan (L şeklinde) karolarımız var.
- Karolar tüm yönler döndürülerek yerleştirilebilir.
- Tüm panoyu karolar ile kaplayarak döşeyin.



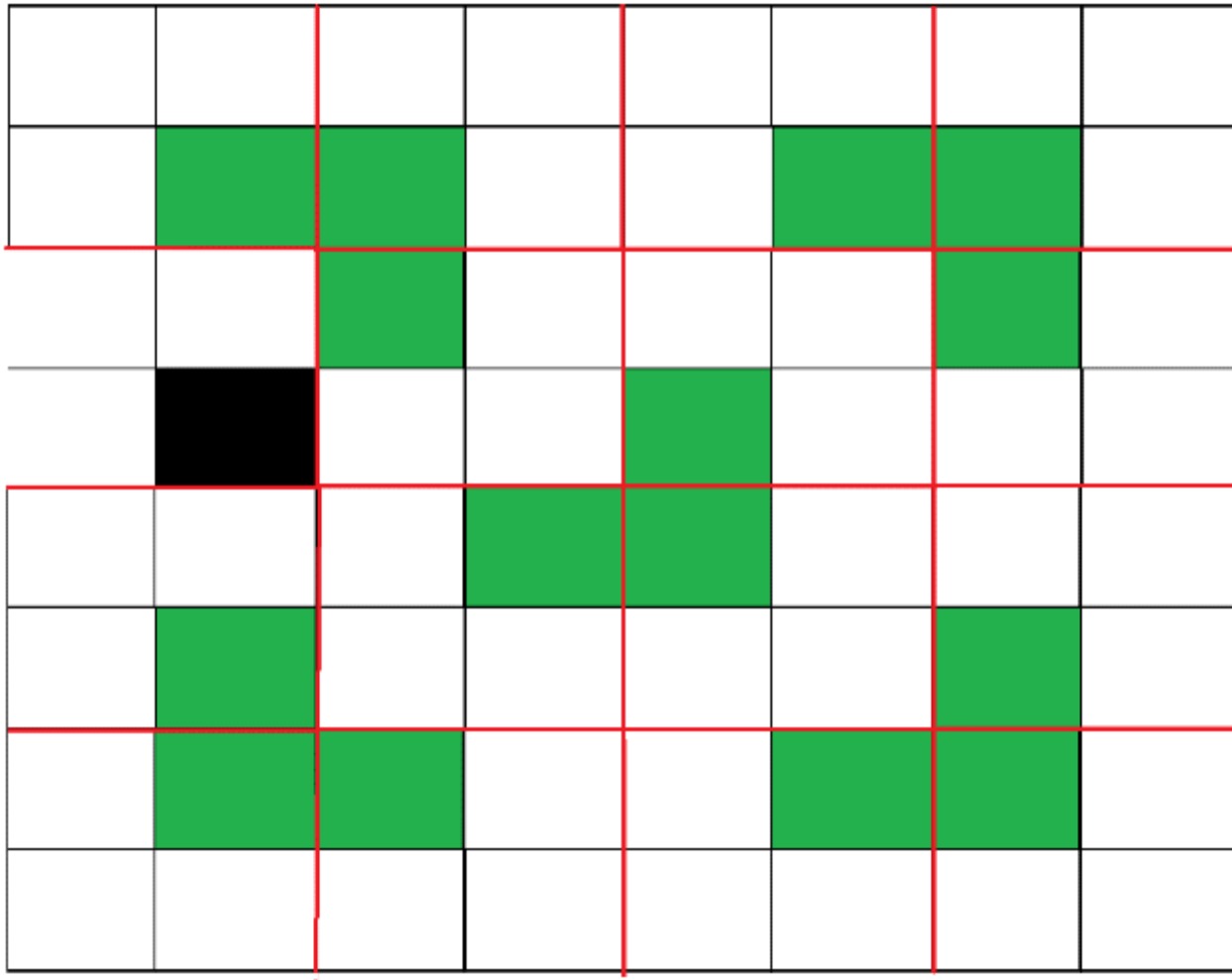
Tromino Puzzle



Tromino Puzzle



Tromino Puzzle



QUICKSORT

QuickSort

- 1959'da Charles Antony Richard Hoare tarafından tasarlanmıştır. 1961'de yayınlanmıştır.
- Merge sort algoritması, elemanları dizideki pozisyonlarına göre bölüyordu.
- QuickSort ise elemanları değerlerine göre bölerek yönetir.
 - Ortadan değil, değere göre bölüm yaratırız.
- Diziden bir elemanı pivot olarak seç.
- Diziyi, pivottan küçükler ve büyükler olarak iki bölüme ayır. (Partitioning)
- Rekürsif olarak bölümleri partitioning işleminden geçir.

QuickSort

- Partition:
- $a[j]$, final pozisyonunda.
- $a[lo, \dots, j-1]$ 'de $a[j]$ 'den daha büyük eleman yok
- $a[j+1, \dots, hi]$ 'da $a[j]$ 'den daha küçük eleman yok

QuickSort

- 4 10 8 7 6 5 3 12 14 2 (pivot:6)
- hepsi daha küçük 4 5 3 2 6 10 8 7 12 14 hepsi daha büyük
- 4 3 2 5 6
- 2 3 4 5 6
- 2 3 4 5 6
- 10 8 7 12 14
- 7 10 8 12 14
- 7 10 8 12 14
- 7 8 10 12 14
- 7 8 10 12 14
- 7 8 10 12 14
- 2 3 4 5 6 7 8 10 12 14

QuickSort

ALGORITHM *Quicksort*($A[l..r]$)

// Sorts a subarray by quicksort

// Input: Subarray of array $A[0..n - 1]$, defined by

// its left and right indices l and r

// Output: Subarray $A[l..r]$ sorted in nondecreasing order

if $l < r$

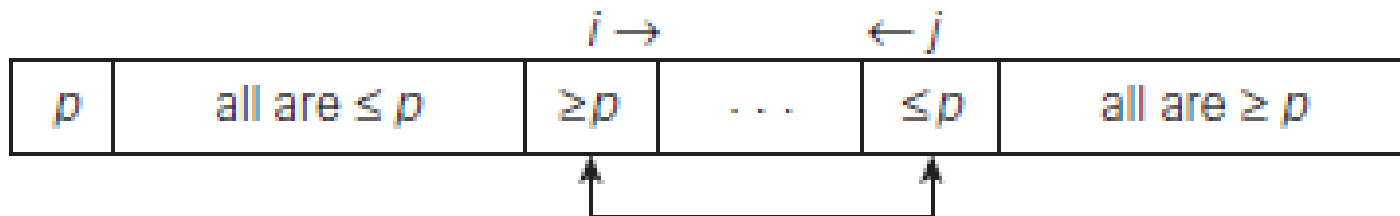
$s \leftarrow \text{Partition}(A[l..r])$ // s is a split position

Quicksort($A[l..s - 1]$)

Quicksort($A[s + 1..r]$)

QuickSort

- Pivot'u nasıl seçeceğiz?
 - Pek çok strateji var.
 - En basiti, ilk eleman pivot olsun.
- Soldan sağa tarama ile (i), pivottan **büyük** eleman bulana kadar ilerleriz.
- Sağdan sola tarama ile (j), pivottan **küçük** eleman bulana kadar ilerleriz.



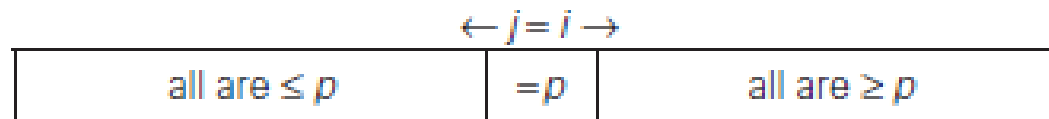
- İkisini de bulunca, yerlerini değiştirip (swap) aramaya devam ederiz.

QuickSort

- Soldan sağa ve sağdan sola taramalar kesiştiğinde ($i > j$ olduğunda) pivotu yerine taşıyoruz, solundakiler küçük, sağındakiler büyük bölümleri oluşturur.



- Pivot noktamızda $s=i=j$ olduğunda bölümler tamamdır. Şimdi onları (rekürsif olarak) partititoning işlemine tabi tutarız.



QuickSort

ALGORITHM *HoarePartition*($A[l..r]$)

// Partitions a subarray by Hoare's algorithm, using the first element as a pivot

// Input: Subarray of array $A[0..n - 1]$, defined by its left and right indices l and r ($l < r$)

// Output: Partition of $A[l..r]$, with the split position returned as this function's value

$p \leftarrow A[l]$

$i \leftarrow l;$

$j \leftarrow r + 1$

repeat

repeat $i \leftarrow i + 1$ **until** $A[i] \geq p$

repeat $j \leftarrow j - 1$ **until** $A[j] \leq p$

 swap($A[i]$, $A[j]$)

until $i \geq j$

swap($A[l]$, $A[j]$) //undo last swap when $i \geq j$

swap($A[l]$, $A[j]$)

return j

QuickSort

0	1	2	3	4	5	6	7		
i					j				
5	3	1	9	8	2	4	7	(repeat .. until i, j)	
i					j				
5	3	1	9	8	2	4	7	(swap)	
5	3	1	4	8	2	9	7	(repeat .. until i, j)	
			i	j					
5	3	1	4	8	2	9	7	(swap)	
5	3	1	4	2	8	9	7		
			j	i					
5	3	1	4	2	8	9	7	(exchange)	
2	3	1	4	5	8	9	7		

QuickSort

0	1	2	3	4	5	6	7
2	3	1	4	5	8	9	7

i **j**
2 3 1 4 (repeat until...)

i **j**
2 3 1 4 (swap)

2 1 3 4
j **i**
2 1 3 4 (Exchange)

1 2 3 4

1 2 3 4

i **j**

1 2 3 4

1 2 3 4

1 2 3 4

QuickSort

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

1	2	3	4	5	8	9	7
---	---	---	---	---	---	---	---

i	j
---	---

8	9	7	(swap)
---	---	---	--------

j	i
---	---

8	7	9	(Exchange)
---	---	---	------------

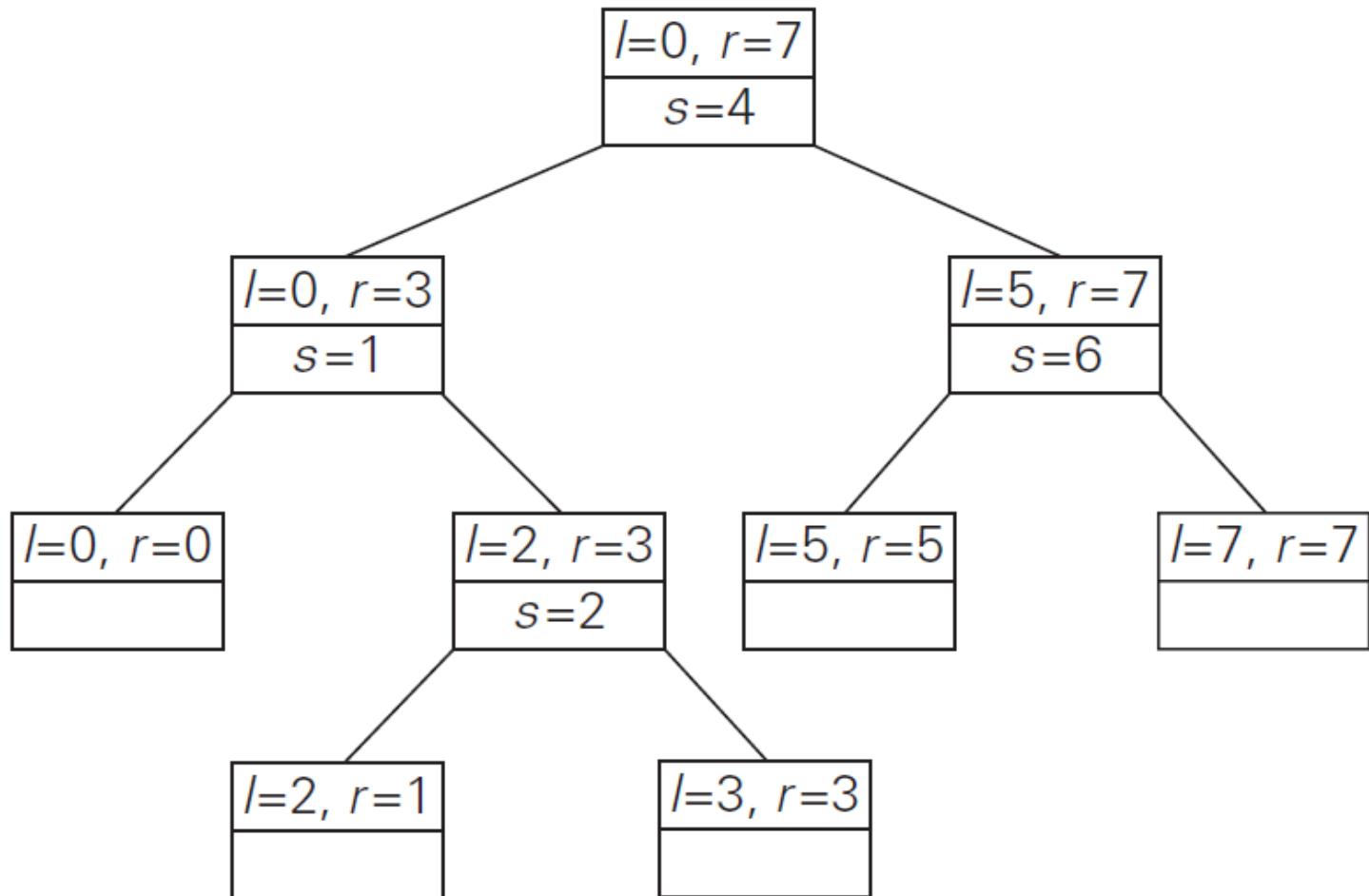
7	8	9
---	---	---

7	8	9
---	---	---

7	8	9
---	---	---

1	2	3	4	5	7	8	9
---	---	---	---	---	---	---	---

QuickSort Rekürsif Çağrı Ağacı



QuickSort

- Partition için pivot'u son eleman olarak da seçebiliriz.

QuickSort(A,p,r)

if ($p < r$)

$q = \text{Partition}(A, p, r)$

 QuickSort(A,p,q-1)

 QuickSort(A,q+1,r)

Partition (A, p, r)

$v = A[r]$

$i = p - 1$

for $j = p$ to $r - 1$

 if $A[j] \leq v$

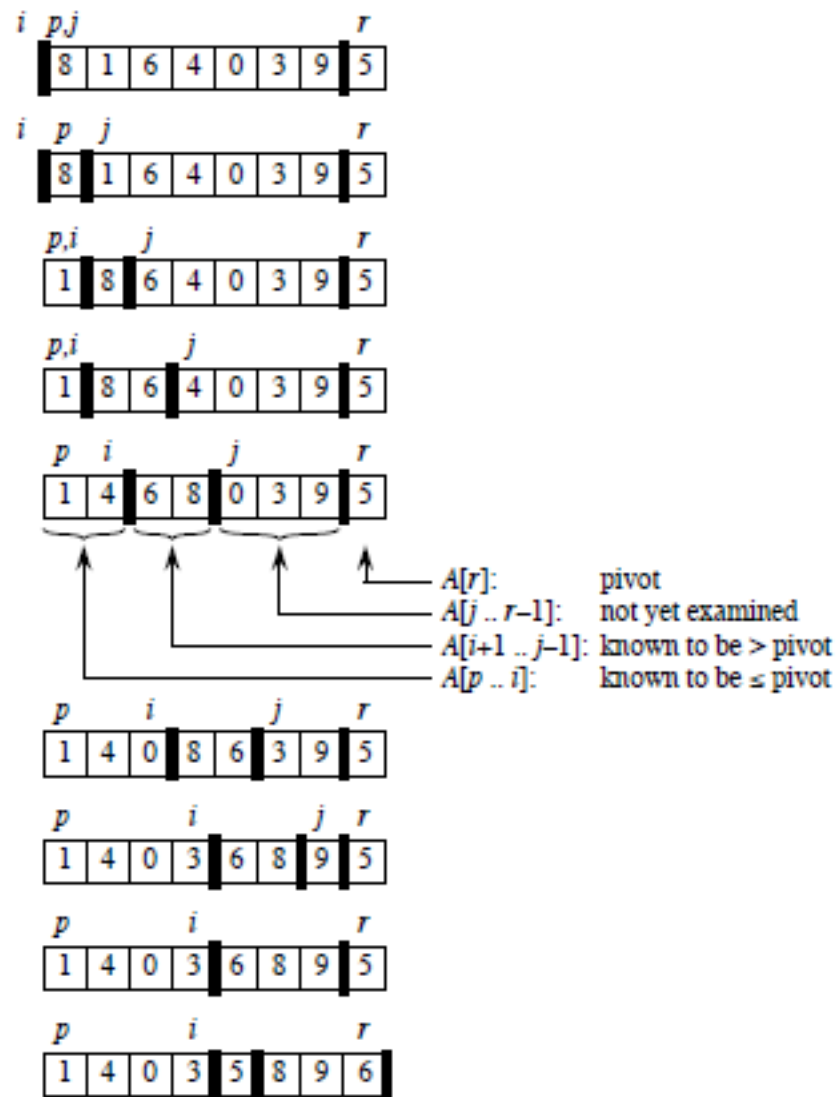
$i = i + 1$

$A[i] \leftrightarrow A[j]$

$A[i + 1] \leftrightarrow A[r]$

return $i + 1$

QuickSort



QuickSort

```
#define exch(A, B) {int t=A; A=B, B=t;}
```

```
int partition(int a[], int l, int r){  
    int i = l-1, j = r, v = a[r];  
    for(;;){  
        while(a[++i]<v);  
        while(v<a[--j]) if(j==l) break;  
        if (i>=j) break;  
        exch(a[i],a[j]);  
    }  
    exch(a[i],a[r]);  
    return i;  
}
```

```
void quicksort(int a[], int l, int r){  
    int i;  
    if (r<l) return;  
    i = partition(a, l, r);  
    quicksort(a, l, i-1);  
    quicksort(a, i+1, r);  
}
```

```
int main(){  
    int a[] = {8,1,6,4,0,3,9,5};  
    int n = sizeof a / sizeof *a;  
    quicksort(a,0,n-1);  
    return 0;  
}
```

8 1 6 4 0 3 9 5
3 1 0 4 5 8 9 6
3 1 0 4 5 8 9 6
0 1 3 4 5 8 9 6
0 1 3 4 5 8 9 6
0 1 3 4 5 8 9 6
0 1 3 4 5 6 9 8
0 1 3 4 5 6 8 9
0 1 3 4 5 6 8 9

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.1

2020-2021 Güz Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

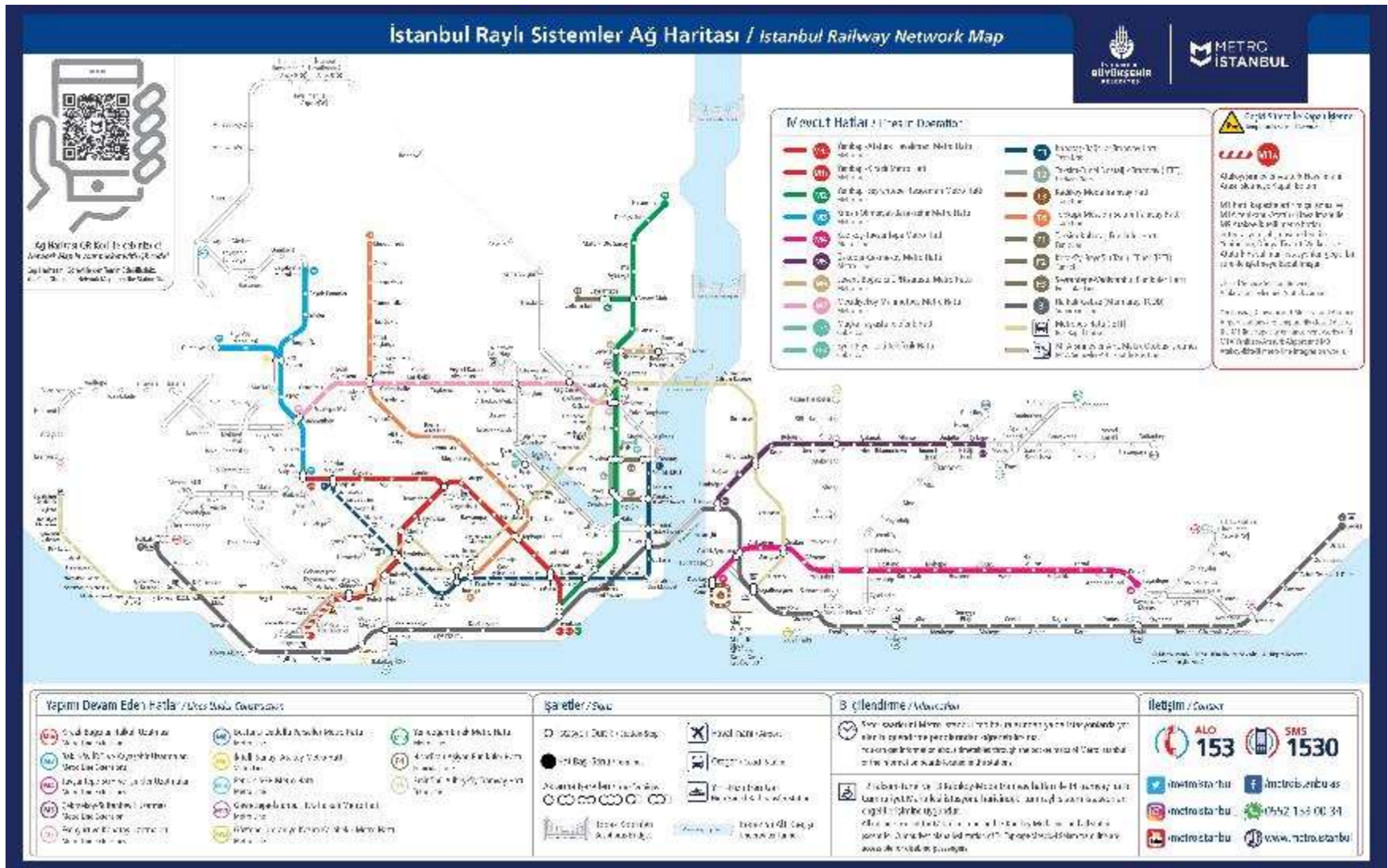
GRAFLAR

Çizgeler

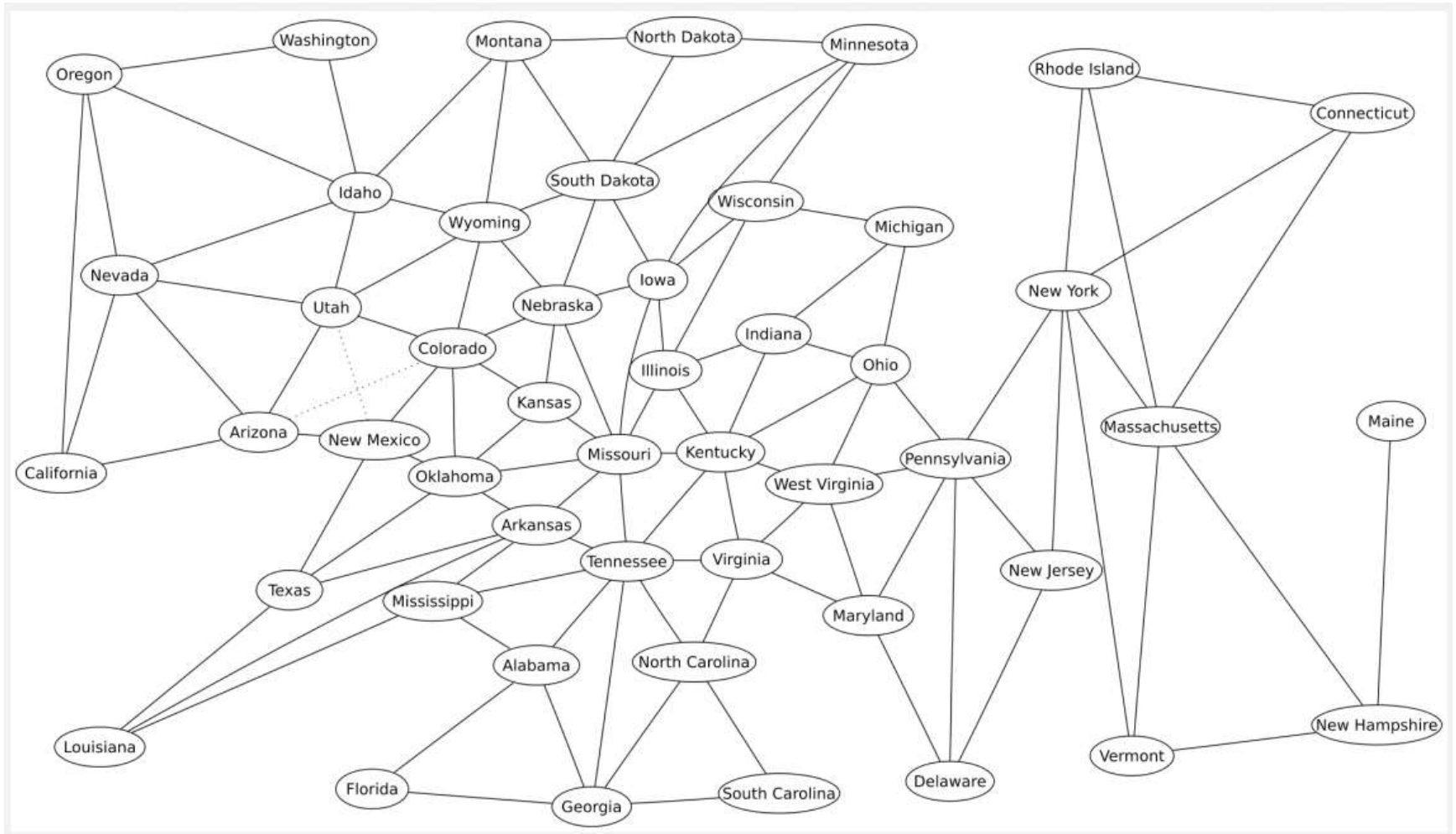
Graflar

- Birbirine bağlı elemanlar kümesine graf adı verilir.
- $G(V,E)$
 - V (vertex) Düğüm, $|V|$ Düğüm sayısı
 - E (edge) Kenar, $|E|$ Kenar sayısı
- Karmaşıklık gösteriminde, $O(VE) \rightarrow O(|V||E|)$

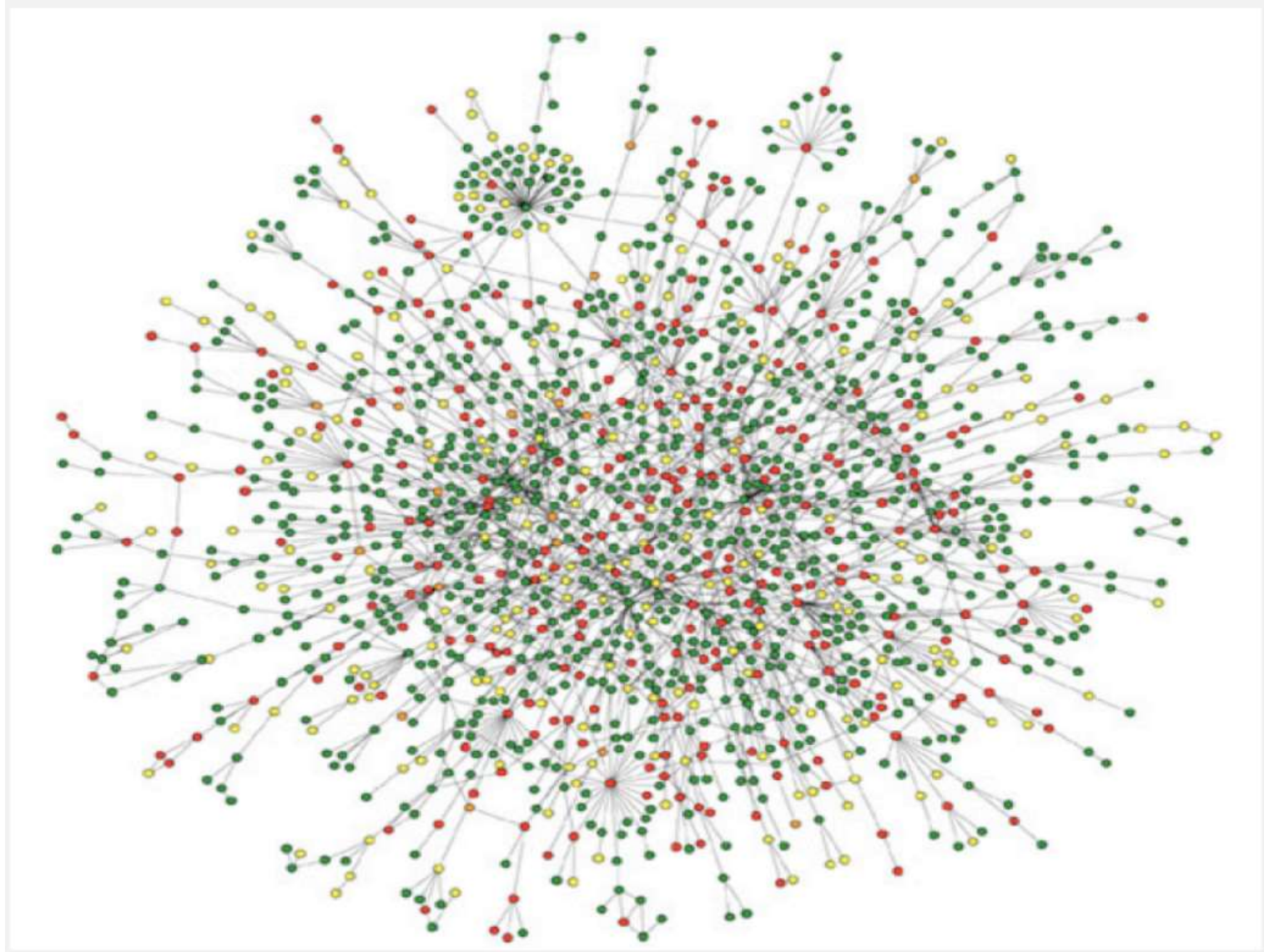
İstanbul Raylı Sistemleri



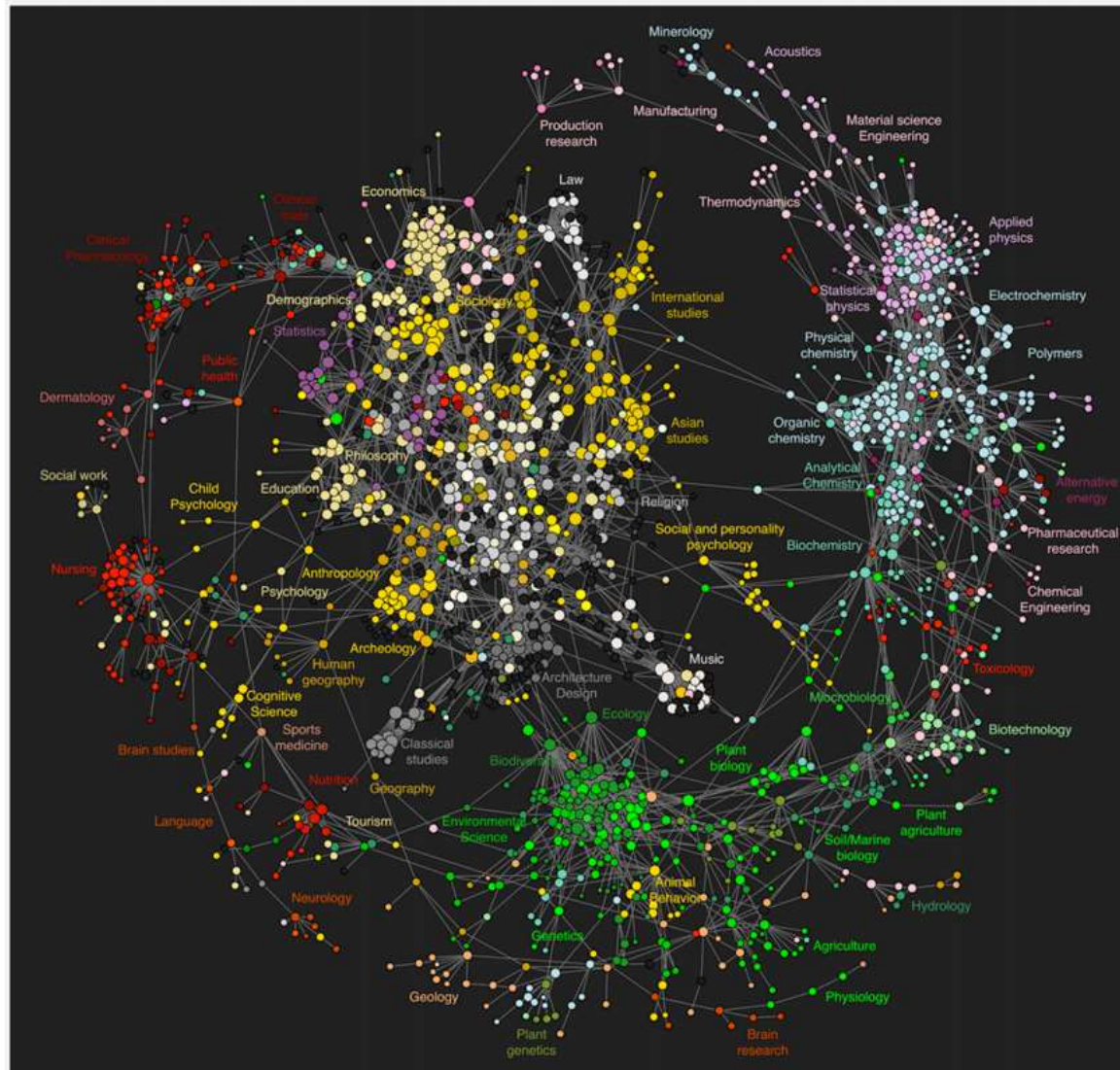
ABD Eyaletleri Sınır Komşulukları



Protein-Protein Etkileşim Ağı



Bilim Konuları Tıklama Bağlantıları



FaceBook Arkadařlık Baęlantıları

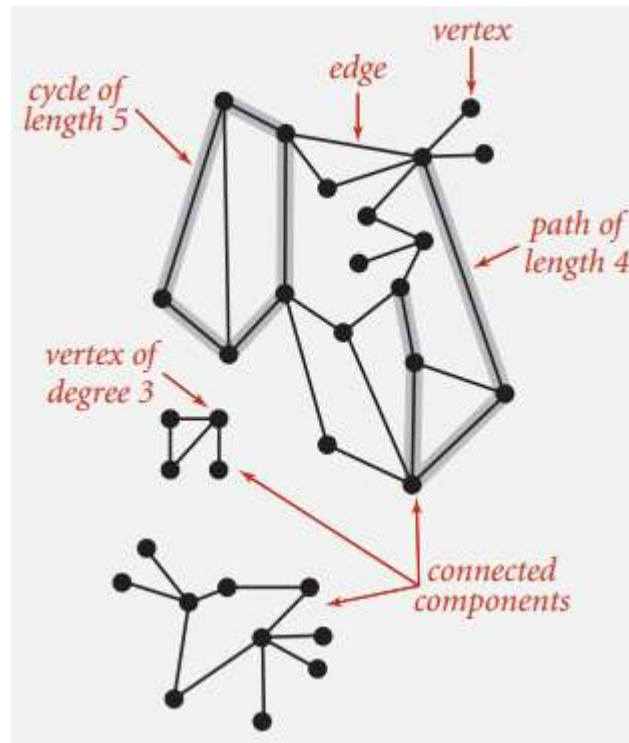


Graf Kullanım Alanları

- **Haritalar:**
 - Beşiktaş-Davutpaşa arası en kısa yol nedir? (Shortest Path)
 - Beşiktaş-Davutpaşa arası en hızlı yol nedir? (Max Flow)
- **Web İçeriği:** Arama Motorları
- **Devreler:** Kısa devre var mı? Bağlantılar çaprazlamadan gerçekleştirilebiliyor mu?
- **Zaman Planlaması/Tarife:** Birbiri ile bağlı işler sırası, kısıtlar altında en kısa sürede nasıl tamamlanır?
- **Ticaret:** Alıcı-Satıcı-Ürün Ağı
- **Eşleştirme:** Öğrenci – Kulüp/Staj eşleştirme
- **Bilgisayar Ağları**
- **Yazılım:** Derleyicilerin modüller arası statik/dinamik çağrıları, kaynakları modellemesi
- **Sosyal ağlar:** Anomali, kanaat önderi, bot ağı...

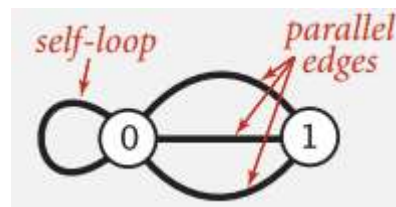
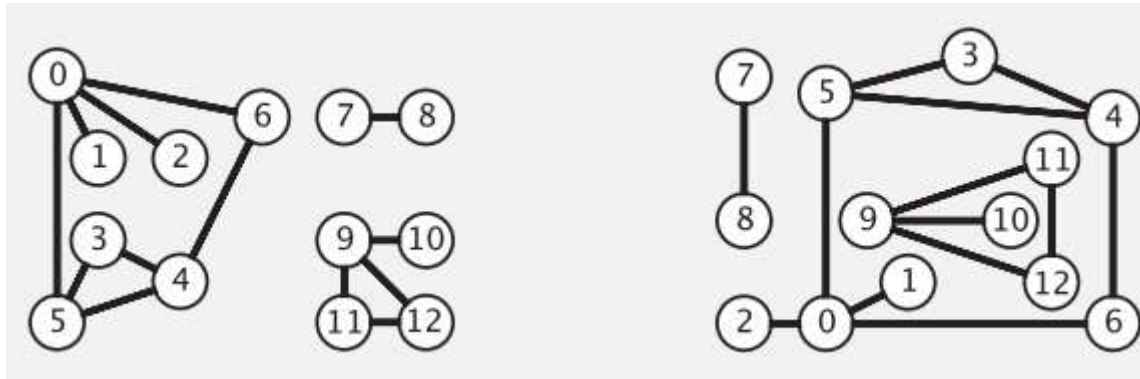
Graf Terminolojisi

- **Path** (Yol): Kenarlarla bağlanmış olan düğümler silsilesi.
- **Cycle** (Çevrim): İlk ve Son düğümü aynı olan yollar.
- İki düğüm arasında bir yol varsa, bu iki düğüm birbirine bağlıdır (**connected**).



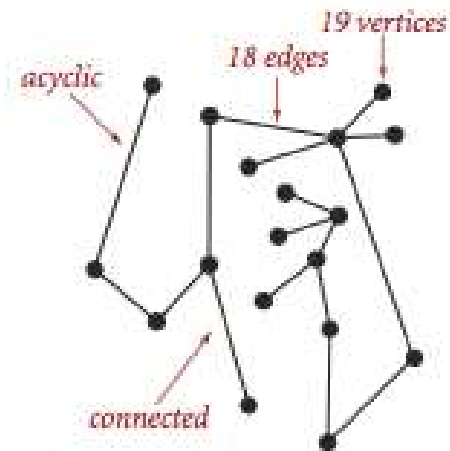
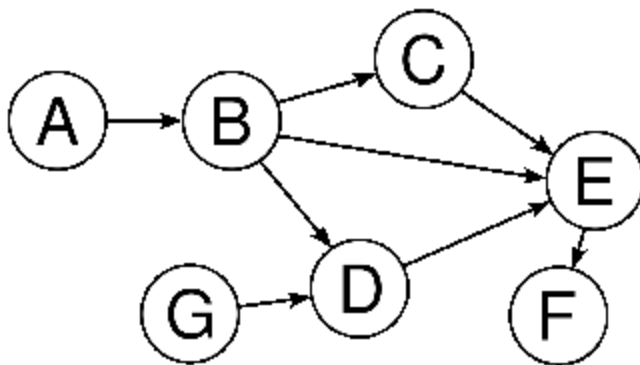
Graf Tipleri

- Yönsüz Graf (Undirected)
- Yönlü Graf (Digraph)
- Kenar Ağırlıklı Yönsüz Graf
- Kenar Ağırlıklı Yönlü Graf



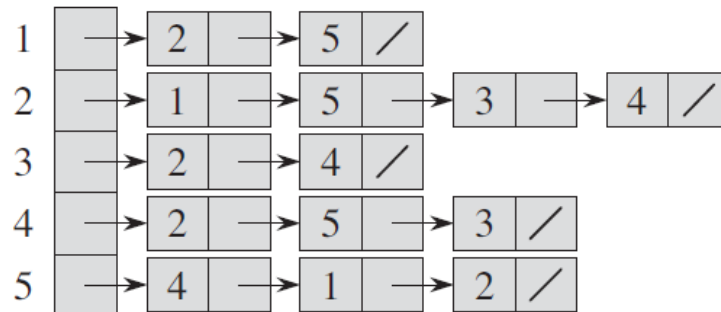
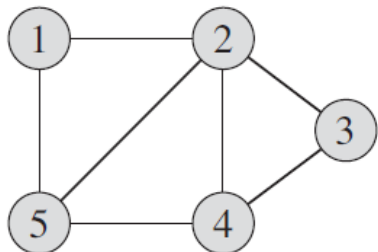
Acyclic (Çevrimsiz) Graf

- Kapalı Döngü içermeyen graflar.
- 5 şart doğru ise ağaçtır:
 - $V-1$ kenar var, döngü yok
 - $V-1$ kenar var, bileşenleri bağlı
 - Herhangi bir kenarı kaldırmak, ağacı keser
 - Çevrimsizdir, bir kenar eklemek cyclic yapar
 - G 'nin her kenar çiftini basit bir yol bağlar.



Graf Gösterimi (Graph Representation)

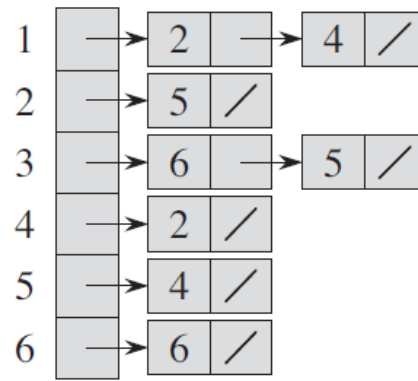
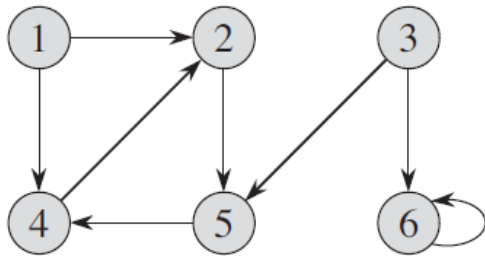
- Komşuluk Listesi (Adjacency List)
 - Seyrek (Sparse) graflar için uygun
- Komşuluk Matrisi (Adjacency Matrix)
 - V^2 yer kaplar
 - $A_{ij} = 1$, if $i, j \in E$; 0 otherwise
 - $A^T = A$
 - 1 bit ile tutabiliriz.



	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

Graf Gösterimi (Graph Representation)

- Yönlü grafa $A^T \neq A$



	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

- Ağırlıklı (weighted) graflarda matriste değerler yer alır.
- Bağlantı var mı?
 - Listede yavaş $O(\text{degree}(V))$
 - Matriste hızlı $O(1)$

GRAF ARAMA YÖNTEMLERİ

BFS, DFS

Önce Genişlemesine Arama (Breadth First Search, BFS)

- En basit arama yöntemlerinden biridir.
- Pekçok algoritma BFS'ye benzer mantık kullanır.
 - Prim MST, Dijkstra SP
- Girdi: $G(V,E)$, yönlü ya da yönsüz, kaynak düğüm $S \in V$
- S 'den ulaşılabilen tüm düğümleri «keşfetmeye» çalışır.
 - «Breadth-First Tree» (S kökünden erişilebilen tüm yollar) yaratır.
 - S 'den erişilebilen v düğümlerine giden en kısa yolları içerir.
- $K+1$ uzaklığa geçmeden önce, K uzaklıktaki tüm yollar bulunur.

Önce Genişlemesine Arama (Breadth First Search, BFS)

- Fikir:
 - S'ten bir dalga gönder.
 - İlk önce S'ten 1 uzaklıktakilere çarpar.
 - Onlardan, 2 uzaklıktakilere çarpar.
 - ...
- Dalganın önünde kim olacak?
- FIFO queue
 - Dalga v'ye çarptı ve daha v'yi terketmediyse, $v \in Q$.

BFS Algoritması

BFS(V, E, s)

for each $u \in V - \{s\}$

do $d[u] \leftarrow \infty$

$d[s] \leftarrow 0$

$Q \leftarrow \emptyset$

ENQUEUE(Q, s)

while $Q \neq \emptyset$

do $u \leftarrow \text{DEQUEUE}(Q)$

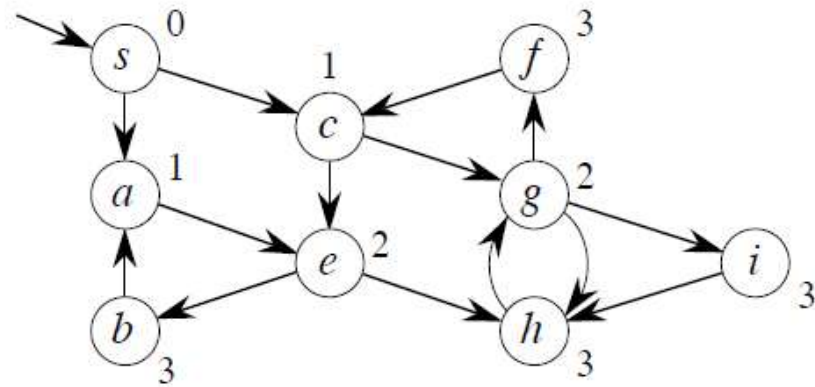
for each $v \in \text{Adj}[u]$

do if $d[v] = \infty$

then $d[v] \leftarrow d[u] + 1$

 ENQUEUE(Q, v)

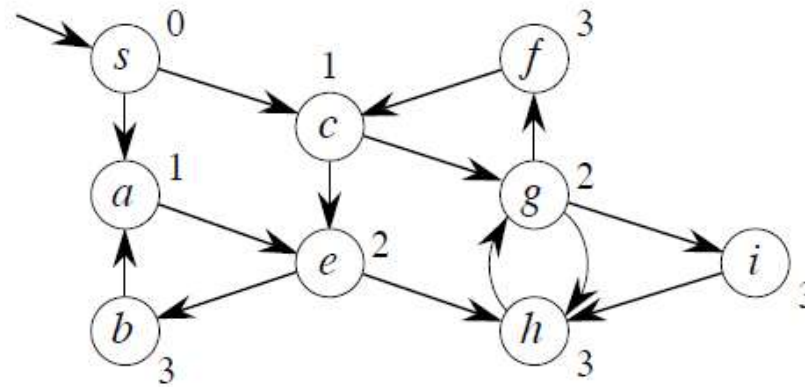
BFS Algoritması



- $Q = NULL$

[illegible]

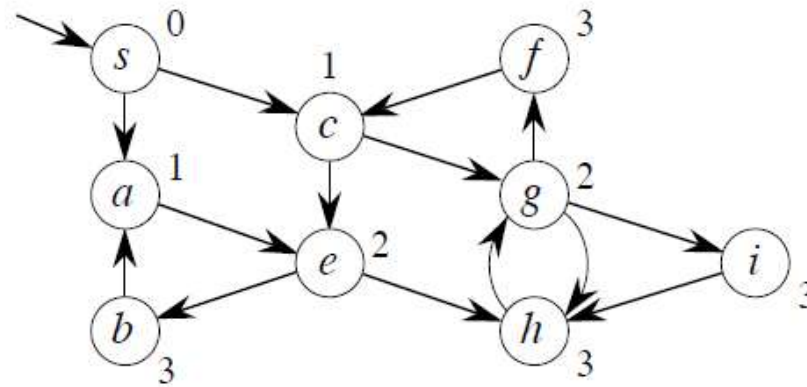
BFS Algoritması



- $Q = \{s\}$
- $u=s$ [a,c] $d[a]=d[s]+1$, $d[c]=d[s]+1$
- $Q=\{a,c\}$

	a	b	c	e	f	g	h	i	s
d	1	∞	1	∞	∞	∞	∞	∞	0

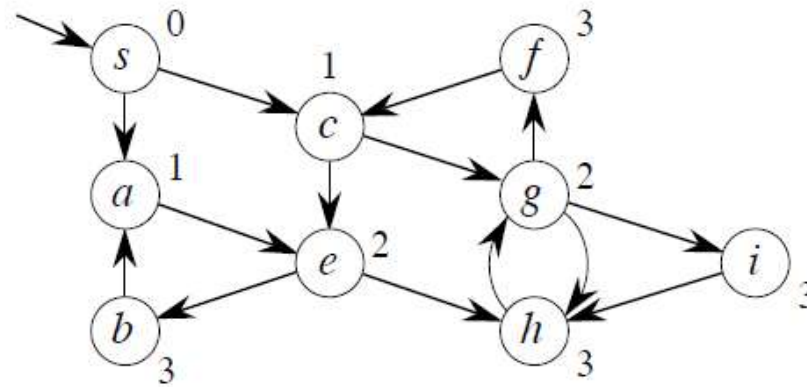
BFS Algoritması



- $Q = \{a, c\}$
- $u=a$ [e] $d[e]=d[a]+1$
- $Q=\{c, e\}$

	a	b	c	e	f	g	h	i	s
d	1	∞	1	2	∞	∞	∞	∞	0

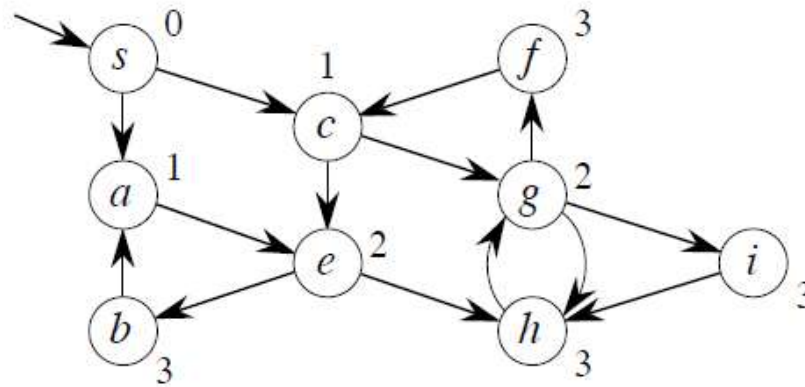
BFS Algoritması



- $Q = \{c, e\}$
- $u=c$ [e,g] $d[e] \neq \infty$, $d[g]=d[c]+1$
- $Q=\{e, g\}$

	a	b	c	e	f	g	h	i	s
d	1	∞	1	2	∞	2	∞	∞	0

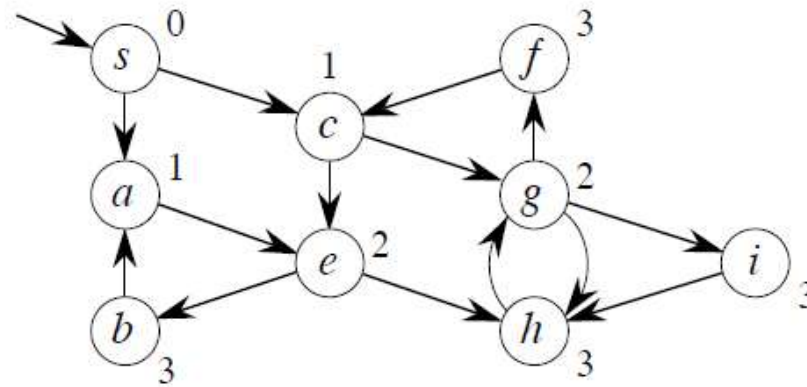
BFS Algoritması



- $Q = \{e, g\}$
- $u=e$ $[b, h]$ $d[b]=d[e]+1$, $d[h]=d[e]+1$
- $Q=\{g, b, h\}$

	a	b	c	e	f	g	h	i	s
d	1	3	1	2	∞	2	3	∞	0

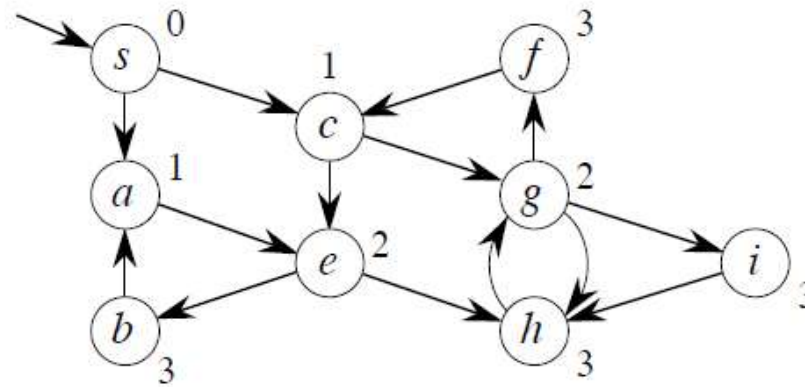
BFS Algoritması



- $Q = \{g, b, h\}$
- $u = g$ $[f, h, i]$ $d[f] = d[g] + 1$, $d[h] \neq \infty$, $d[i] = d[g] + 1$
- $Q = \{b, h, f, i\}$

	a	b	c	e	f	g	h	i	s
d	1	3	1	2	3	2	3	3	0

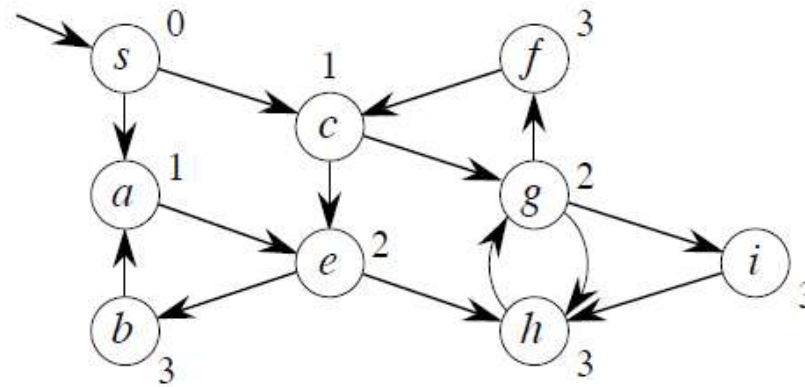
BFS Algoritması



- $Q = \{b, h, f, i\}$
- $u=b$ [a] $d[a] \neq \infty$
- $Q = \{h, f, i\}$

	a	b	c	e	f	g	h	i	s
d	1	3	1	2	3	2	3	3	0

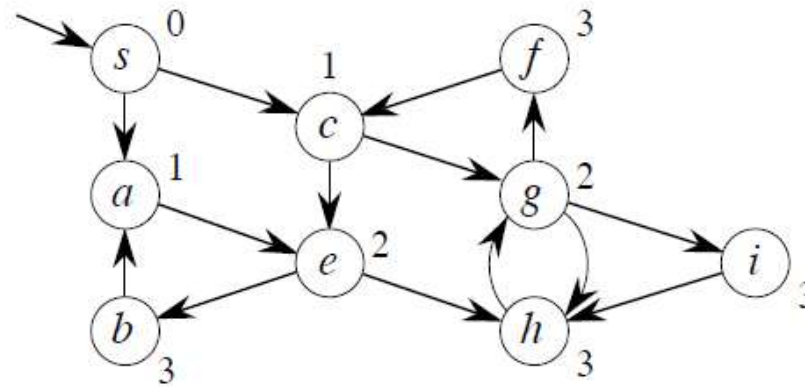
BFS Algoritması



- $Q = \{h, f, i\}$
- $u = h$ [g] $d[g] \neq \infty$
- $Q = \{f, i\}$

	a	b	c	e	f	g	h	i	s
d	1	3	1	2	3	2	3	3	0

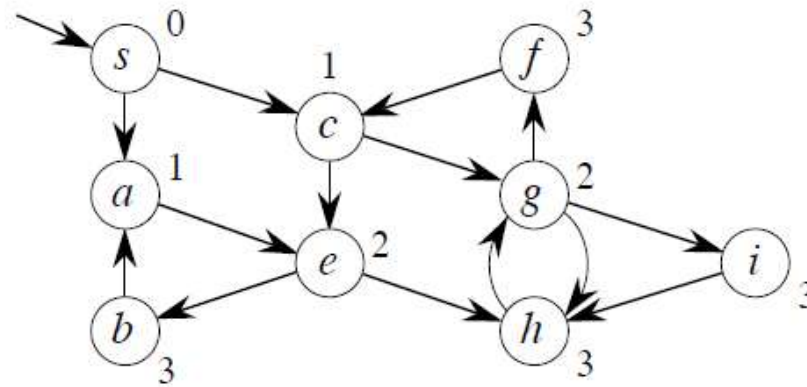
BFS Algoritması



- $Q = \{f, i\}$
- $u=f$ $[c]$ $d[c] \neq \infty$
- $Q = \{i\}$

	a	b	c	e	f	g	h	i	s
d	1	3	1	2	3	2	3	3	0

BFS Algoritması



- $Q = \{i\}$
- $u=i$ [h] $d[h] \neq \infty$
- $Q=\{\}$ \rightarrow *stop*

	a	b	c	e	f	g	h	i	s
d	1	3	1	2	3	2	3	3	0

(Renkli) BFS Algoritması

- Düğümün durumlarını takip etmek isteyebiliriz.
 - Gezilmemiş: Beyaz
 - Gezilmiş Gri/Siyah
 - Siyahların komşuları siyah/gri olabilir. (siyahların tüm komşuları keşfedilmiştir)
 - Grilerin komşuları gri/beyaz olabilir.

(Renkli) BFS Algoritması

BFS (G,s)

for each vertex $u \in G.V - \{s\}$

$u.color = \text{WHITE}$

$u.d = \infty$

$u.p = \text{NIL}$

$s.color = \text{GRAY}$

$s.d = 0$

$s.p = \text{NIL}$

$Q = \{ \} ;$

ENQUEUE(Q,s)

while $Q \neq \{ \}$

$u = \text{DEQUEUE}(Q)$

for each $v \in G.Adj[u]$

if $v.color == \text{WHITE}$

$v.color = \text{GRAY}$

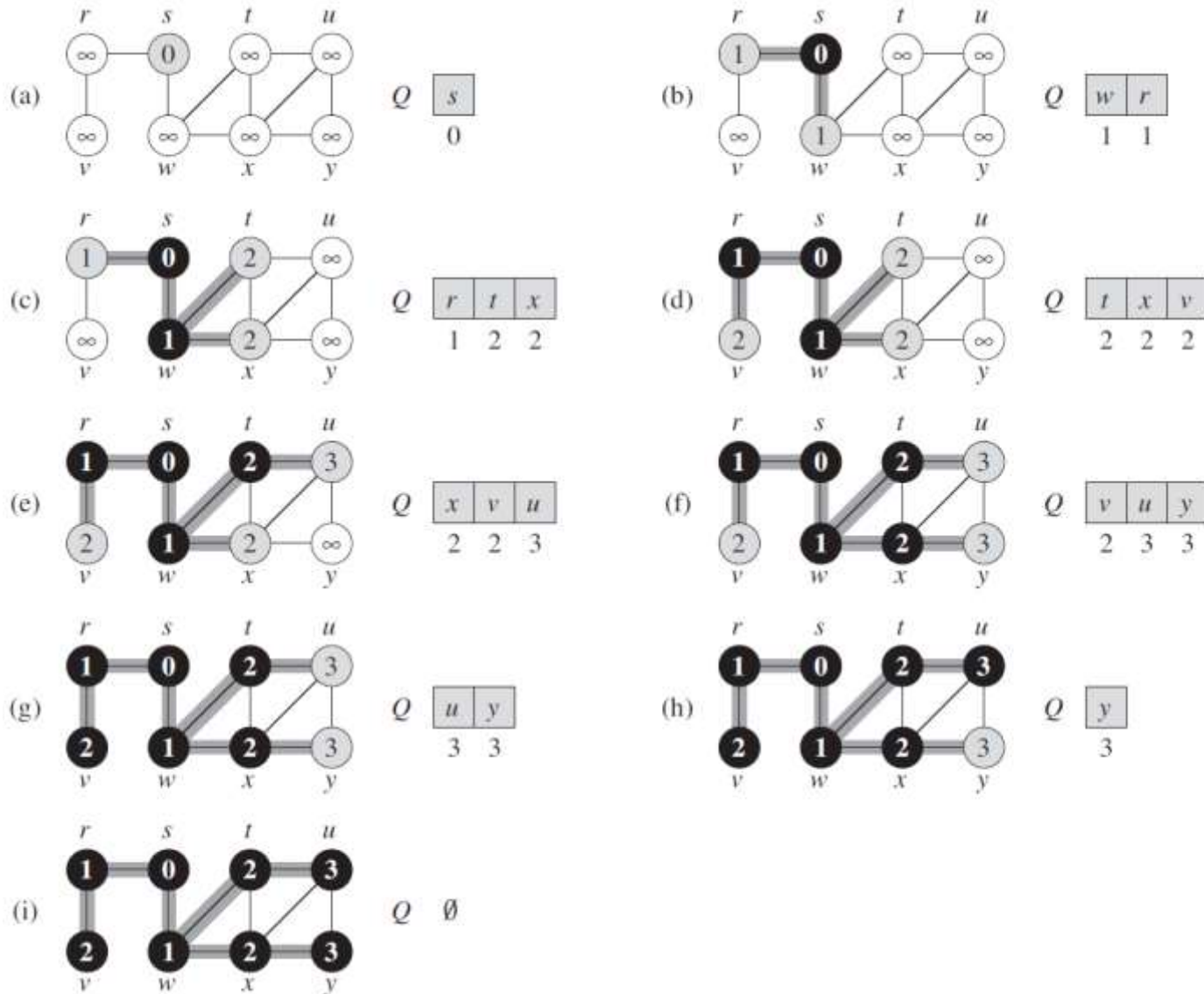
$v.d = u.d + 1$

$v.p = u$

 ENQUEUE(Q,v)

$u.color = \text{BLACK}$

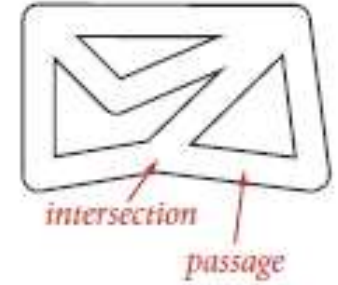
(Renkli) BFS Algoritması



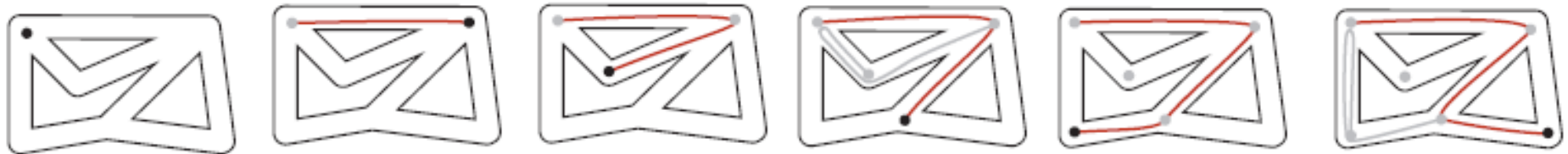
BFS Karmaşıklık Analizi

- Enqueue / Dequeue $O(1)$
- Her düğüm en fazla 1 kere enqueue, toplam $O(V)$
- Her düğüm en fazla 1 kere dequeue ve o zaman (u,v) kontrolü: $O(E)$
- $O(V+E)$ (+ init $O(V)$)
- Komşuluk listesi gösterimi boyutunda lineer zamanda çalışır.

Tremaux Exploration



- Bir labirentin girişindeyiz. Elimizde bir yumak ip var. Kaybolmadan labirentten nasıl çıkarız?
- 1. İşaretsiz bir geçide ip döşe
- 2. İlk kez geçtiğin tüm geçit ve kesişimleri işaretle
- 3. İşaretli bir kesişime gelersen, ip ile geri gel
- 4. Geri gelirken hiçbir işaretsiz kesişim kalmayana kadar adımları geri al.



DFS de bu yöntemle benzer. Hatta daha kolaydır 😊

Önce Derinlemesine Arama (Depth First Search, DFS)

- Düğümleri rekürsif olarak ziyaret et.
- Bir düğümü al, işaretle.
- Tüm (işaretsiz) komşularını rekürsif olarak ziyaret et.

DepthFirstSearch(G, s)

count=0;

for each vertex $u \in G.V$

marked[u]=*FALSE*

 dfs(G, s)

dfs(G, v)

marked[v]=*TRUE*

 count++

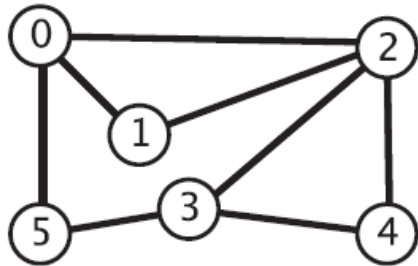
for each $w \in G.adj[v]$

if (! *marked*[w])

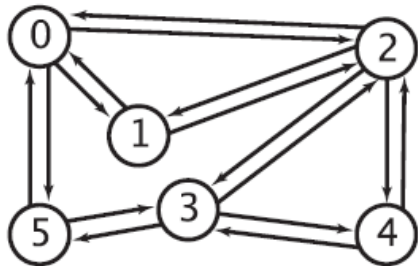
 dfs(G, w)

DFS Algoritması

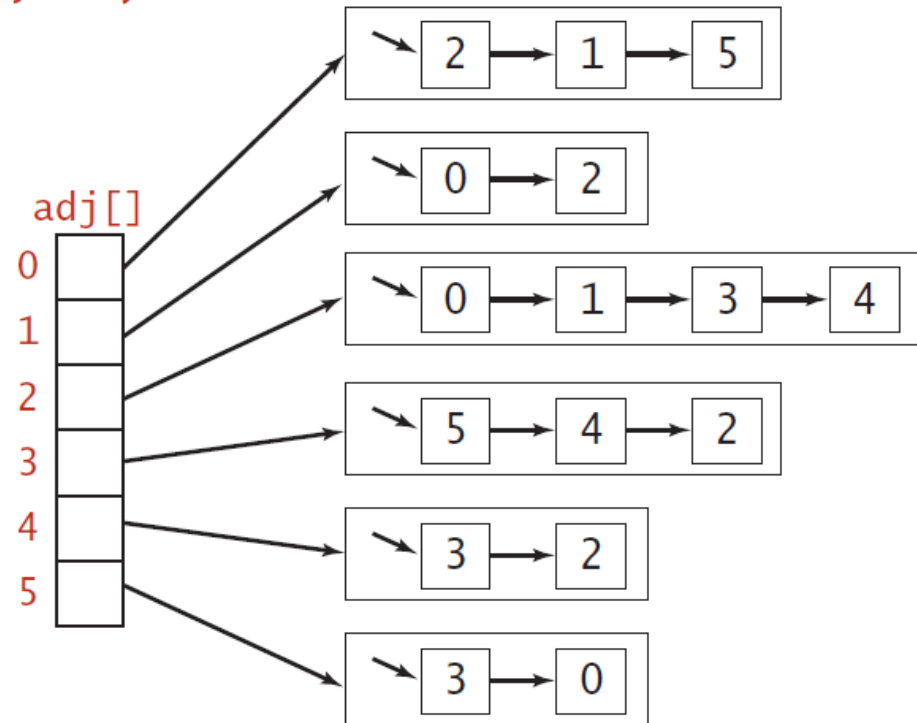
standard drawing



drawing with both edges

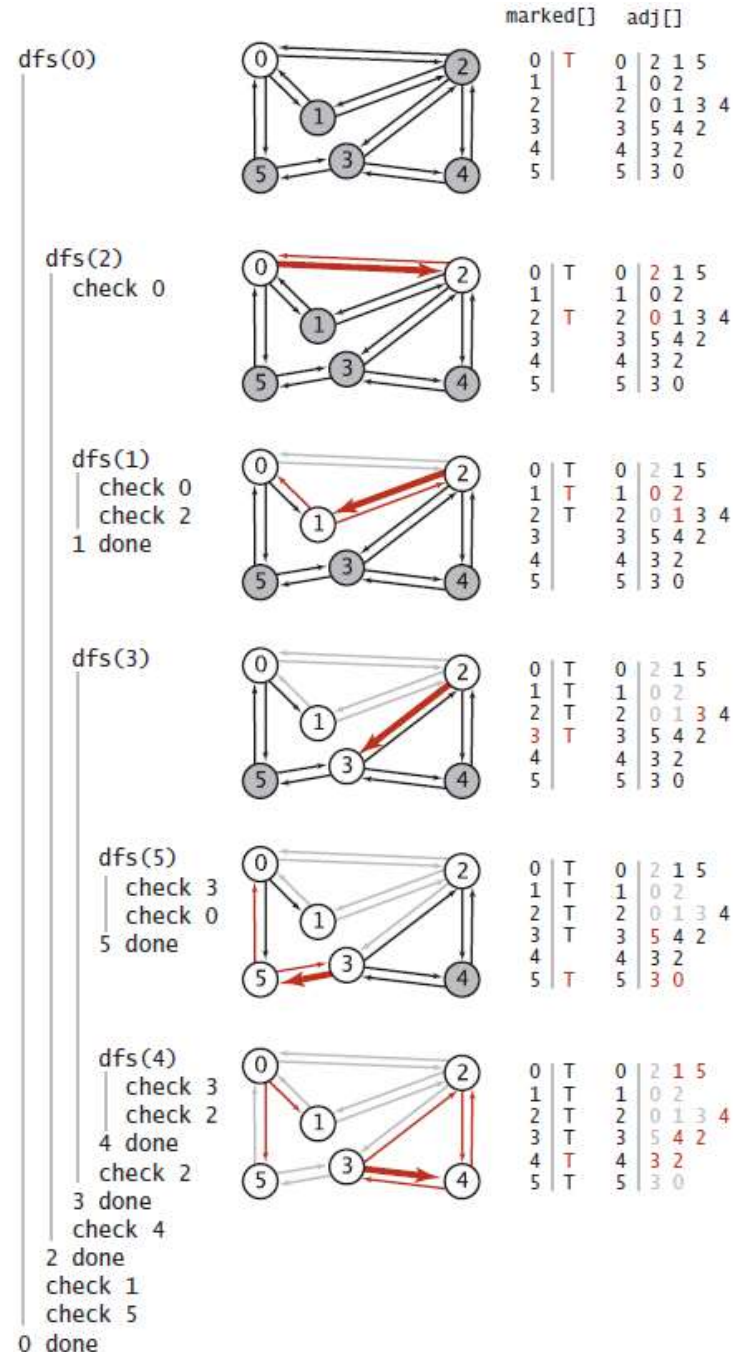


adjacency lists



DFS Algoritması

- 2, 0'ın ilk komşusu ve unmarked. Visit 2.
- 2'nin ilk komşusu 0 ve marked. Geç. Sıradaki komşu 1 ve unmarked. Rekürsif olarak işaretle ve visit 1.
- 1'in tüm komşuları [0,2] marked. Geri dön. 2'nin sıradaki komşusu 3'ü işaretle ve visit (unmarked).
- 3'ün ilk komşusu 5 ve unmarked. İşaretle ve Visit 5.
- 5'in tüm komşuları [3,0] marked. Geri dön. 3'ün sıradaki komşusu 4'ü işaretle ve visit (unmarked).
- 4'ün komşularını kontrol et, geri dön 3'ün kalan komşularını kontrol et, geri dön 2'nin kalan komşularını kontrol et, geri dön 0'ın kalan komşularını kontrol et.
- Tüm gezinti bitti.



DFS ile neyi çözebiliriz?

- Tek kaynaktan nerelere gidebiliriz?
- Verilen 2 düğüm birbirine bağlı mıdır? \Leftrightarrow 2 düğüm arası yol var mıdır?

DepthFirstPaths(G,s)

for each vertex $u \in G.V$

$marked[u]=FALSE$

$edgeTo[u] = 0;$

$dfs(G,s)$

$dfs(G,v)$

$marked[v]=TRUE$

for each $w \in G.adj[v]$

if ($! marked[w]$)

$edgeTo[w]=v$

$dfs(G,w)$

$pathTo(v)$

 // path is a stack

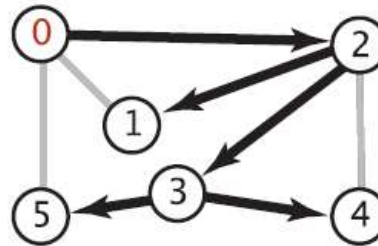
if ($! marked[v]$) **return** NULL

for ($x=v$; $x!=s$; $x=edgeTo[x]$)

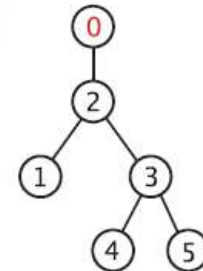
$path.push(x)$

$path.push(s)$

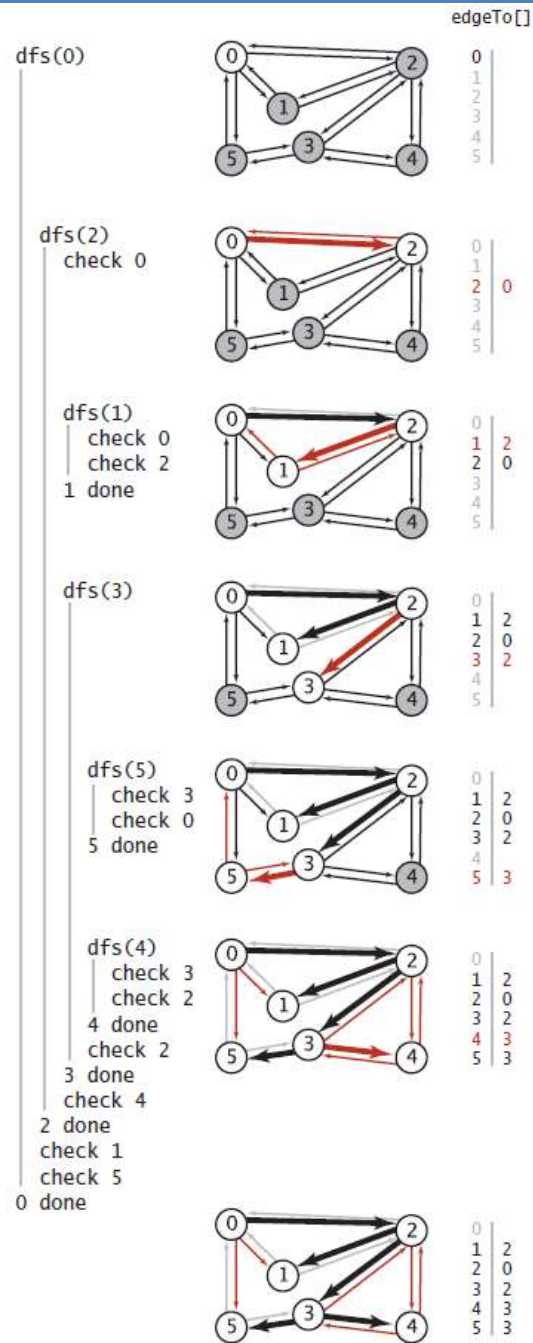
return path



edgeTo[]	
0	
1	2
2	0
3	2
4	3
5	3



x	path
5	5
3	3 5
2	2 3 5
0	0 2 3 5



(Renkli) DFS Algoritması

DFS(V, E)

for each $u \in V$

do $color[u] \leftarrow \text{WHITE}$

$time \leftarrow 0$

for each $u \in V$

do if $color[u] = \text{WHITE}$

then DFS-VISIT(u)

DFS-VISIT(u)

$color[u] \leftarrow \text{GRAY}$ // discover u

$time \leftarrow time + 1$

$d[u] \leftarrow time$

for each $v \in Adj[u]$ // explore (u, v)

do if $color[v] = \text{WHITE}$

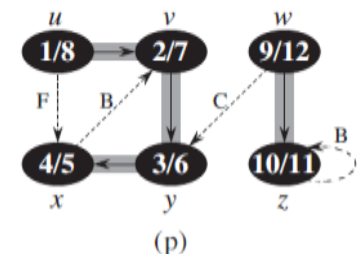
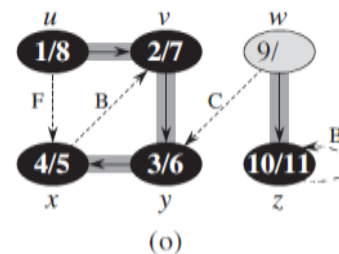
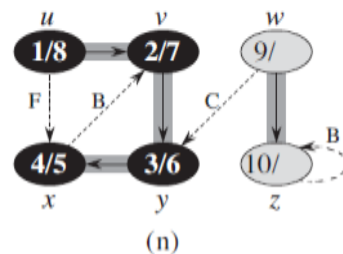
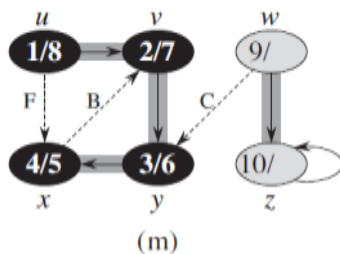
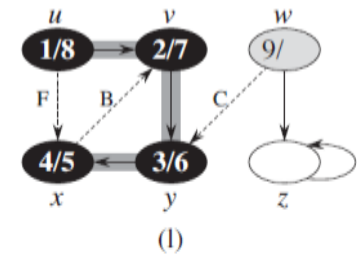
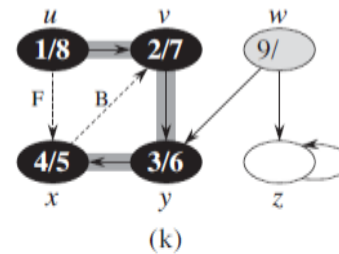
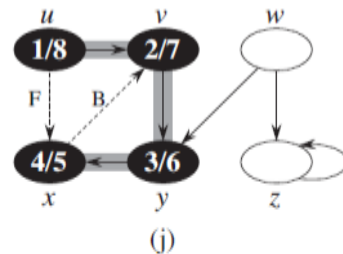
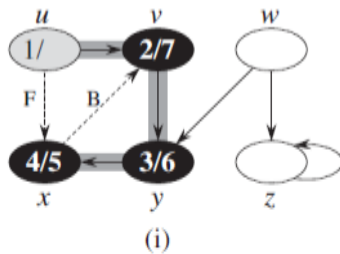
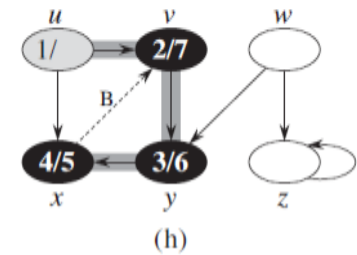
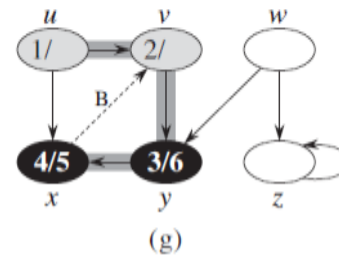
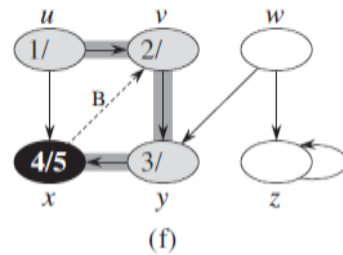
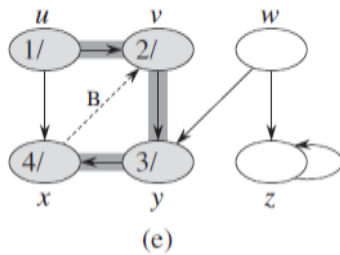
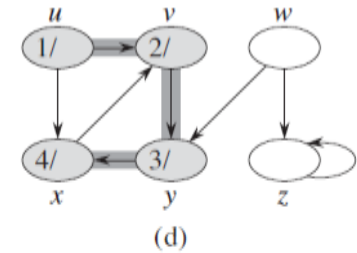
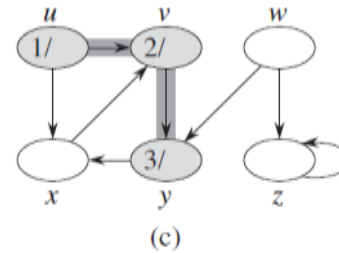
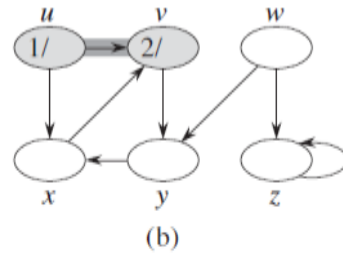
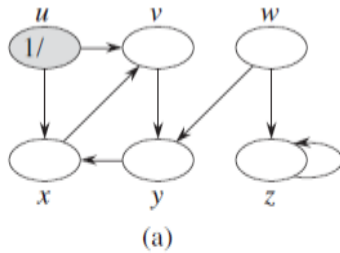
then DFS-VISIT(v)

$color[u] \leftarrow \text{BLACK}$

$time \leftarrow time + 1$

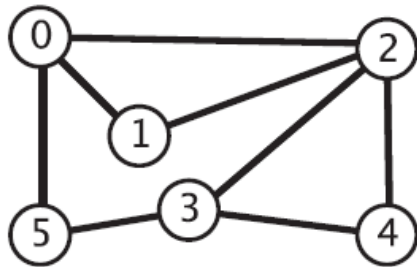
$f[u] \leftarrow time$ // finish u

(Renkli) DFS Algoritması

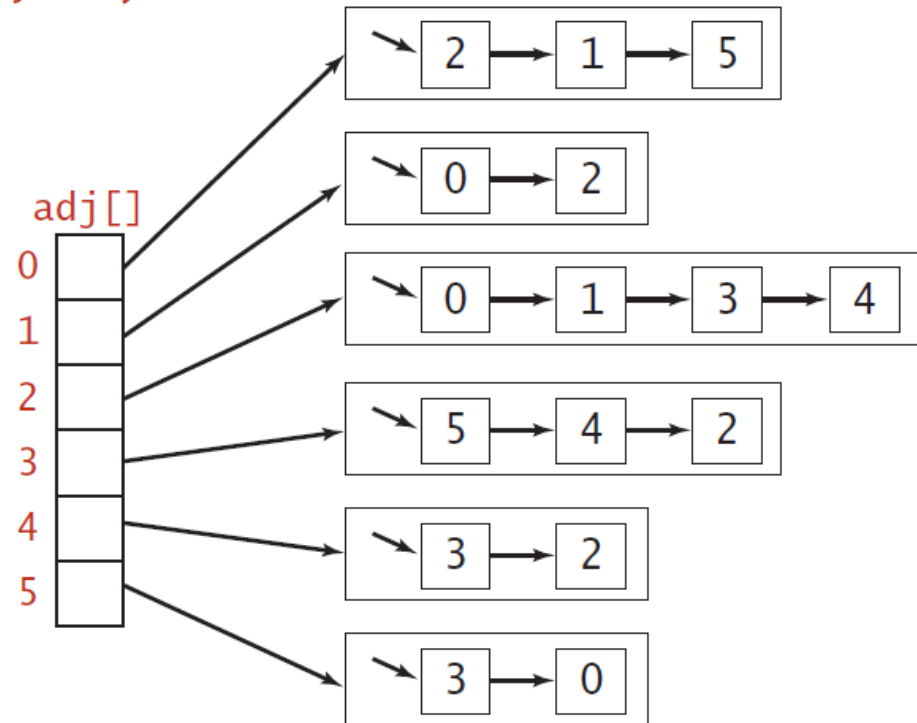


(Renkli) DFS Algoritması

standard drawing



adjacency lists



(Renkli) DFS Algoritması

dfs(0)

2 1 5

dfs(2)

0 1 3 4

dfs(1)

0 2

dfs(3)

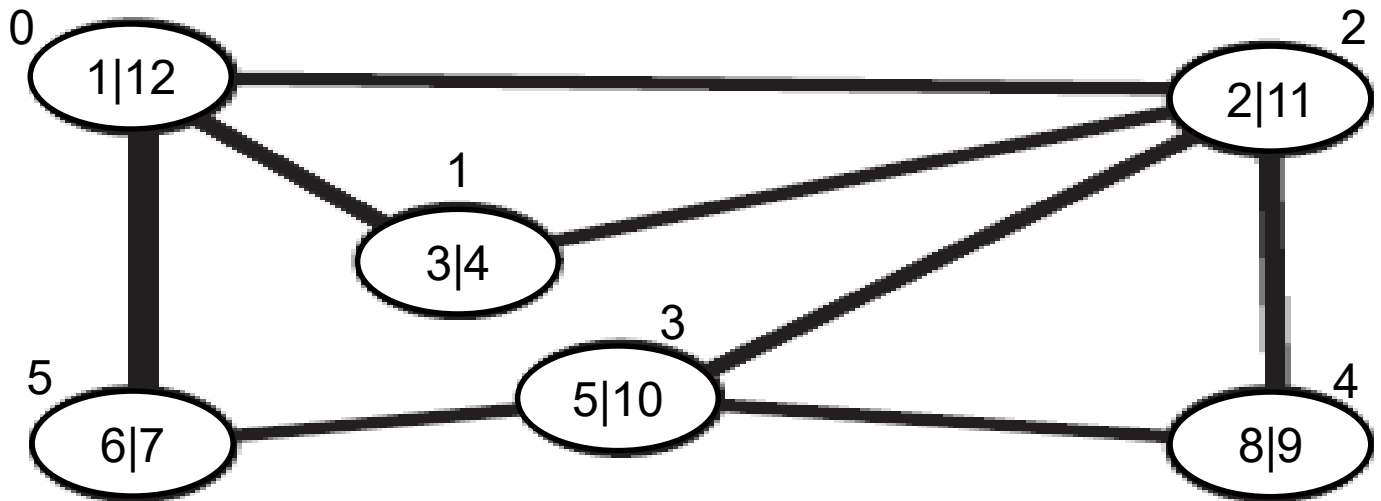
5 4 2

dfs(5)

3 0

dfs(4)

3 2



	d	f	color			Pre
0	1	12	W	G	B	
1	3	4	W	G	B	2
2	2	11	W	G	B	0
3	5	10	W	G	B	2
4	8	9	W	G	B	3
5	6	7	W	G	B	3

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.1

2020-2021 Güz Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

COUNTING SORT

Sorting by Counting

Counting Sort

- Fikir: Sıralanacak dizideki tüm elemanlar için;
- İşlenen elemandan daha küçük olan toplam eleman sayısını sayarız.
- Sonuçları bir tabloya kaydederiz.
- Tablodaki sayılar, sıralı dizide sayıların hangi konumda durması gerektiğini bize gösterir.
 - Örneğin bir elemandan küçük 10 eleman varsa, bu eleman 11.gözde durmalıdır.
- Giriş dizimizi **yeni** bir diziye indis adreslerine göre kopyalarak sıralı çıktımızı oluşturabiliriz.

Comparison Counting Sort

ALGORITHM *ComparisonCountingSort*($A[0..n - 1]$)

//Sorts an array by comparison counting

//Input: An array $A[0..n - 1]$ of orderable elements

//Output: Array $S[0..n - 1]$ of A 's elements sorted in nondecreasing order

for $i \leftarrow 0$ **to** $n - 1$ **do** $Count[i] \leftarrow 0$

for $i \leftarrow 0$ **to** $n - 2$ **do**

for $j \leftarrow i + 1$ **to** $n - 1$ **do**

if $A[i] < A[j]$

$Count[j] \leftarrow Count[j] + 1$

else

$Count[i] \leftarrow Count[i] + 1$

for $i \leftarrow 0$ **to** $n - 1$ **do** $S[Count[i]] \leftarrow A[i]$

return S

Comparison Counting Sort

Array $A[0..5]$

62	31	84	96	19	47
----	----	----	----	----	----

Initially

Count []

0	0	0	0	0	0
---	---	---	---	---	---

After pass $i = 0$

Count []

3	0	1	1	0	0
---	---	---	---	---	---

After pass $i = 1$

Count []

	1	2	2	0	1
--	---	---	---	---	---

After pass $i = 2$

Count []

		4	3	0	1
--	--	---	---	---	---

After pass $i = 3$

Count []

			5	0	1
--	--	--	---	---	---

After pass $i = 4$

Count []

				0	2
--	--	--	--	---	---

Final state

Count []

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

Array $S[0..5]$

19	31	47	62	84	96
----	----	----	----	----	----

Nerede İşe Yarar?

- Minimum sayıda indeks değişikliği ile (en sondaki döngü) sıralı dizide doğru konumlara atama yapar.
- Sayma ile sıralama, sıralanacak öğelerin bilinen küçük bir değer kümesine ait olduğu bir durumda verimli bir şekilde çalışır. Her elemanın tekrar sayıları (frekansları) F dizisinde olsun.
 - Değerleri en küçük I değerine eşit olan elemanlar, sıralı dizinin 0 ile $F[0] - 1$ arasındaki pozisyonlara kopyalanır;
 - Değerleri $I+1$ olanlar $F[0]$ ile $(F[0] + F[1]) - 1$ arasına kopyalanır;
 - ...
 - Bu tür birikmiş frekansların toplamı istatistikte dağılım olarak adlandırıldığından, yöntem **dağıtım sayımı (distribution counting)** olarak bilinir.

Distribution Counting Sort

ALGORITHM *DistributionCountingSort*($A[0..n - 1]$, l , u)

//Sorts an array of integers from a limited range by distribution counting

//Input: An array $A[0..n - 1]$ of integers between l and u ($l \leq u$)

//Output: Array $S[0..n - 1]$ of A 's elements sorted in nondecreasing order

for $j \leftarrow 0$ **to** $u - l$ **do** $D[j] \leftarrow 0$ //initialize frequencies

for $i \leftarrow 0$ **to** $n - 1$ **do** $D[A[i] - l] \leftarrow D[A[i] - l] + 1$ //compute frequencies

for $j \leftarrow 1$ **to** $u - l$ **do** $D[j] \leftarrow D[j - 1] + D[j]$ //reuse for distribution

for $i \leftarrow n - 1$ **downto** 0 **do**

$j \leftarrow A[i] - l$

$S[D[j] - 1] \leftarrow A[i]$

$D[j] \leftarrow D[j] - 1$

return S

$O(N+R)$ (R :range)

Distribution Counting Sort

13	11	12	13	12	12
----	----	----	----	----	----

Array values	11	12	13
Frequencies	1	3	2
Distribution values	1	4	6

$A[5] = 12$
 $A[4] = 12$
 $A[3] = 13$
 $A[2] = 12$
 $A[1] = 11$
 $A[0] = 13$

$D[0..2]$

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

$S[0..5]$

			12		
		12			
					13
	12				
11					
				13	

RADIX SORT

Radix Sort

- Mekanik sıralamadan doğdu.
- IBM delikli kartları sıralamak için kolon kolon çalışan (arada insanı da kullanan) yöntem geliştirmişti.



Radix Sort

- Sayıların (veya kelimelerin) ilk ya da son hanesinden başlayarak (baş | son harf)
 - Her basamakta saymalı sıralama yaparız.
- Son haneden başlarsak LSD,
- İlk haneden başlarsak MSD ile sıralarız.

LSD Radix Sort

- Least Significant Digit Radix Sort

Algorithm LSDRadixSort(R)

// Input: (Multi)set $R = \{S_1, S_2, \dots, S_n\}$ of strings of length m over alphabet $[0..\sigma)$.

// Output: R in ascending lexicographical order.

for $i \leftarrow m - 1$ to 0 do

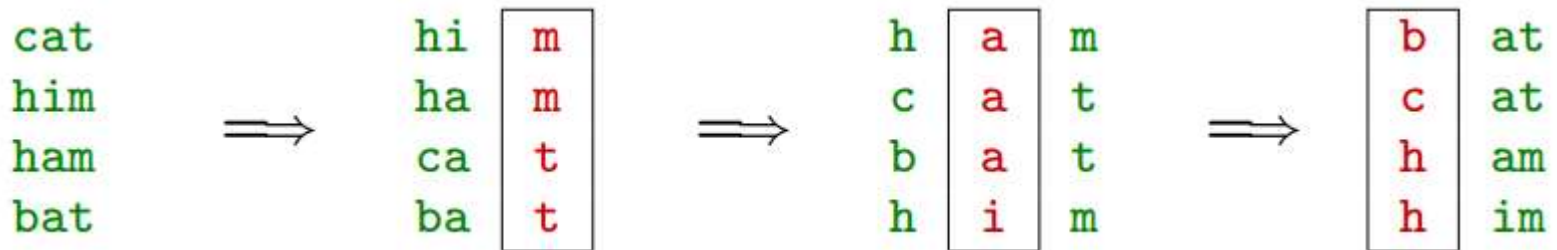
 CountingSort(R, i)

return R

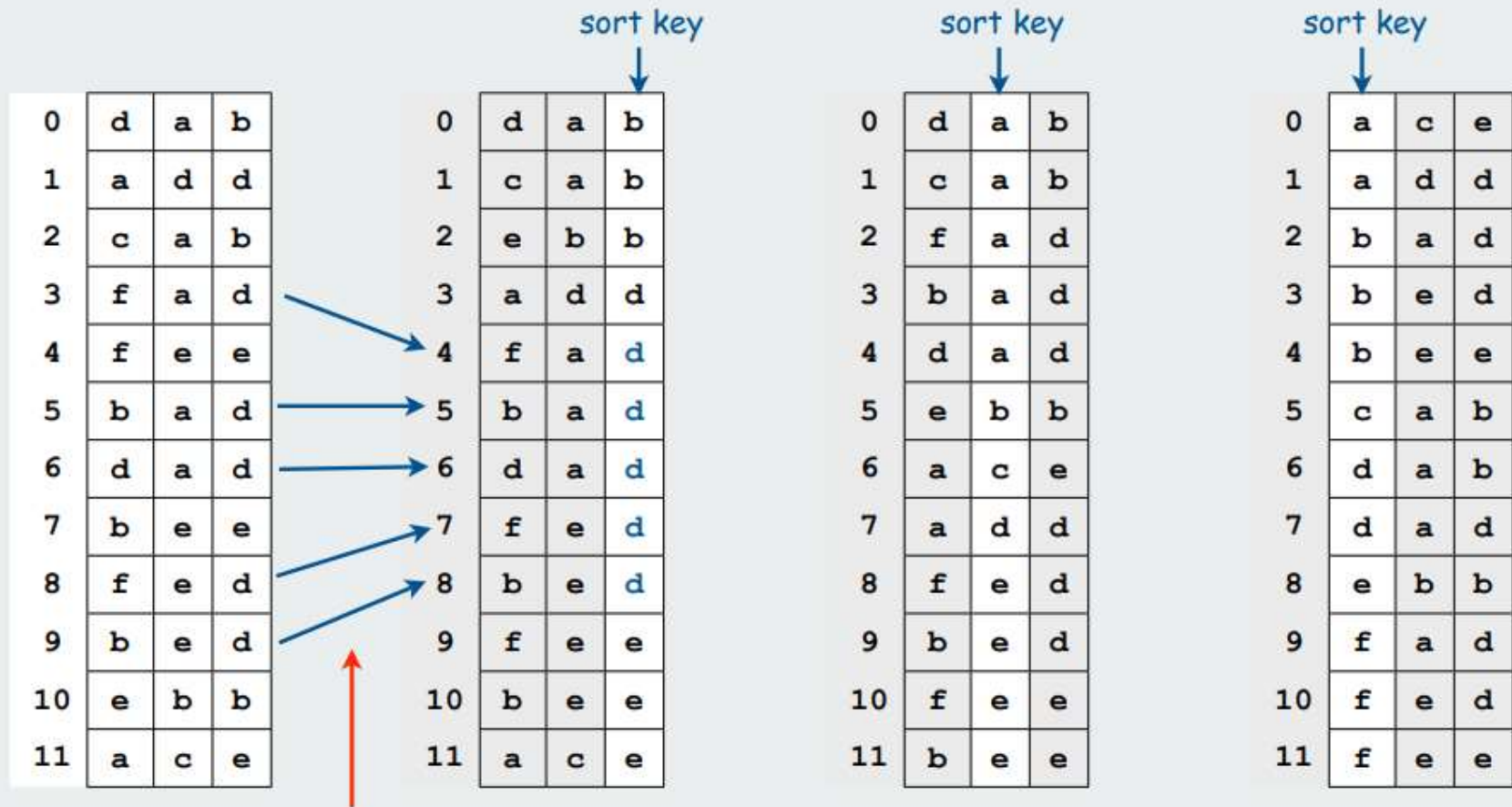
$O(|R| + \sigma)$.

LSD Radix Sort

- Tüm girdilerin aynı uzunlukta olması gerekli.
 - Sayısal girdilerde sol haneleri 0 ile doldururuz.
- $R = \{\text{cat}, \text{him}, \text{ham}, \text{bat}\}$.



LSD Radix Sort



Challenge

- Sabit uzunluklu anahtarlardan oluşan devasa bir ticari veritabanı tablonuz var.
- Ör. Hesap no, kimlik no, GUID, ...
- Neyle sıralasak?
 - Insertion Sort
 - MergeSort
 - QuickSort
 - HeapSort
 - LSD String Sort

	B14-99-8765		
	756-12-AD46		
	CX6-92-0112		
	332-WX-9877		
	375-99-QWAX		
	CV2-59-0221		
	987-SS-0321		
	KJ-01-12388		
	715-YT-013C		
	MJ0-PP-983F		
	908-KK-33TY		
	BBN-63-23RE		
	48G-BM-912D		
	982-ER-9P1B		
	WBL-37-PB81		
	810-F4-J87Q		
	LE9-N8-XX76		
	908-KK-33TY		
	B14-99-8765		
	CX6-92-0112		
	CV2-59-0221		
	332-WX-23SQ		
	332-6A-9877		

Challenge

- Sabit uzunluklu anahtarlardan oluşan devasa bir ticari veritabanı tablonuz var.
- Ör. Hesap no, kimlik no, GUID, ...

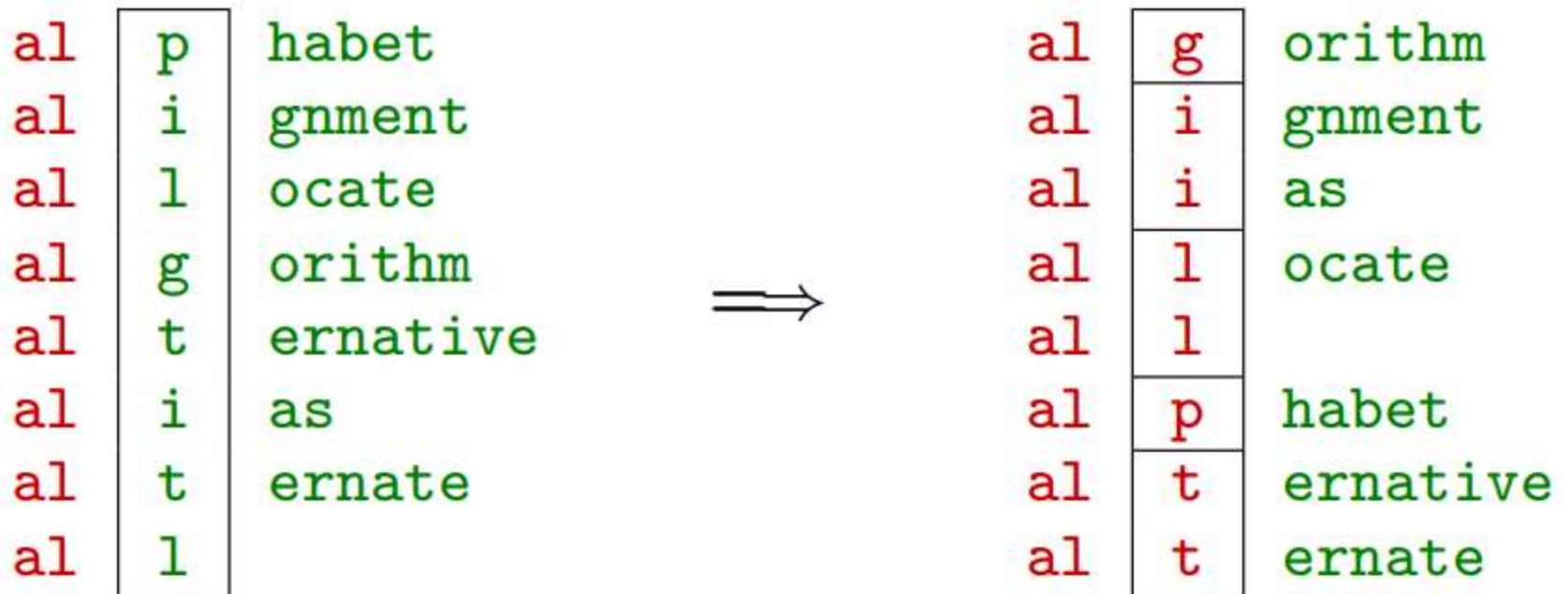
- Neyle sıralasak?

- Insertion Sort
- MergeSort
- QuickSort
- HeapSort
- LSD String Sort
 - 256 (ASCII) ya da 65536 (Unicode) sayaç
 - W adımda sabit uzunluklu stringleri sıralarız.

	B14-99-8765		
	756-12-AD46		
	CX6-92-0112		
	332-WX-9877		
	375-99-QWAX		
	CV2-59-0221		
	987-SS-0321		
	KJ-0-12388		
	715-YT-013C		
	MJ0-PP-983F		
	908-KK-33TY		
	BBN-63-23RE		
	48G-BM-912D		
	982-ER-9P1B		
	WBL-37-PB81		
	810-F4-J87Q		
	LE9-N8-XX76		
	908-KK-33TY		
	B14-99-8765		
	CX6-92-0112		
	CV2-59-0221		
	332-WX-23SQ		
	332-6A-9877		

MSD Radix Sort

- Most Significant Digit Sort
- MSD radix sıralama string quicksort'a benzer, ancak stringleri üç parça yerine σ parçaya böler.
- Her parçayı rekürsif olarak tekrar MSD sort eder.



MSD Radix Sort

Algorithm MSDRadixSort(R, i)

// Input: (Multi)set $R = \{S_1, S_2, \dots, S_n\}$ of strings over the alphabet $[0..\sigma)$ and the length i of their common prefix.

// Output: R in ascending lexicographical order.

if $|R| < \sigma$ then return StringQuicksort(R, i)

$R_{\perp} \leftarrow \{S \in R \mid |S| = i\}; R \leftarrow R \setminus R_{\perp}$

$(R_0, R_1, \dots, R_{\sigma-1}) \leftarrow \text{CountingSort}(R, i)$

for $i \leftarrow 0$ to $\sigma - 1$ do $R_i \leftarrow \text{MSDRadixSort}(R_i, i + 1)$

return $R_{\perp} \cdot R_0 \cdot R_1 \cdot \dots \cdot R_{\sigma-1}$

$O(|R| + \sigma)$

MSD Radix Sort

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

↑
sort key

count[]

a	0
b	2
c	5
d	6
e	8
f	9

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

sort these
independently
(recursive)

MSD Radix Sort

input		d							
she	are	are	are	are	are	are	are	are	are
sells	by	by	by	by	by	by	by	by	by
seashells	she	sells	seashells	sea	sea	sea	seas	sea	sea
by	sells	seashells	sea	seashells	seashells	seashells	seashells	seashells	seashells
the	seashells	sea	seashells	seashells	seashells	seashells	seashells	seashells	seashells
sea	sea	sells	sells	sells	sells	sells	sells	sells	sells
shore	shore	seashells	sells	sells	sells	sells	sells	sells	sells
the	shells	she	she	she	she	she	she	she	she
shells	she	shore	shore	shore	shore	shore	shells	shells	shells
she	sells	shells	shells	shells	shells	shells	shore	shore	shore
sells	surely	she	she	she	she	she	she	she	she
are	seashells	surely	surely	surely	surely	surely	surely	surely	surely
surely	the	the	the	the	the	the	the	the	the
seashells	the	the	the	the	the	the	the	the	the



MSD Radix Sort

are	are	are	are	are	are	are	output
by	by	by	by	by	by	by	are
sea	sea	sea	sea	sea	sea	sea	by
<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	sea
<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	<u>seashells</u>	seashells
<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	seashells
<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	<u>sells</u>	sells
she	she	she	she	she	she	she	sells
shells	shells	shells	shells	shells	shells	shells	sells
she	she	she	she	she	she	she	she
shore	shore	shore	shore	shore	shore	shore	she
surely	surely	surely	surely	surely	surely	surely	shells
the	the	the	the	the	the	the	shore
the	the	the	the	the	the	the	surely
							the
							the

*need to examine
every character
in equal keys*

*end-of-string
goes before any
char value*

Trace of recursive calls for MSD string sort (no cutoff for small subarrays, subarrays of size 0 and 1 omitted)

MSD Radix Sort

- Kötü yanları:
- Küçük parçalarda oldukça yavaş çalışır.
 - Tüm sayaçları sıfırlıyoruz.
 - Bu yüzden algoritmada küçük boyutlar için QuickSort çağırılmıştı.
 - Insertion Sort da olabilir.
- Özyineleme yüzünden çok fazla küçük parça oluşur.
 - Tüm harfler farklıysa, 2 boyutlu $N/2$ ayrı parça

MSD Radix Sort

- İyi yanları:
 - Tüm karakterleri karşılaştırmak zorunda kalmayabiliriz.

0	a	c	e
1	a	d	d
2	b	a	d
3	b	e	d
4	b	e	e
5	c	a	b
6	d	a	b
7	d	a	d

← 19/24 ≈ 80% of the characters examined

- Değişken uzunluklu girdiler ile kolaylıkla çalışabiliriz.

0	a	c	e	t	o	n	e	\0	
1	a	d	d	i	t	i	o	n	\0
2	b	a	d	g	e	\0			
3	b	e	d	a	z	z	l	e	d
4	b	e	e	h	i	v	e	\0	
5	c	a	b	i	n	e	t	r	y
6	d	a	b	b	l	e	\0		
7	d	a	d	\0					

← 19/64 ≈ 30% of the characters examined

SIRALAMA YÖNTEMLERİNİN KARŞILAŞTIRILMASI

Sıralama

- Aynı türden nesnelerden oluşan bir koleksiyonu, birbirini büyüklük-küçüklük açısından mantıksal bir düzene sokmak.
- Girdi: $\{a_1, a_2, \dots, a_n\}$
- Çıktı: $\{a_1', a_2', \dots, a_n'\}$, $a_1' \leq a_2' \leq \dots \leq a_n'$
- Kullanım alanları:
 - Transaction processing, kombinatoryal optimizasyon, astrofizik, moleküler dinamik, dilbilim, genbilim, hava tahmini, ...
- 20.yy en iyi 10 algoritmasından biri: QuickSort !

Sıralama Algoritmalarının Değerlendirilmesi

- Performans:
 - Çalışma zamanı
 - Karşılaştırma ve değiştirme sayısı (değiştirme yoksa dizi erişim sayısı)
- Ekstra Bellek:
 - Yerinde sıralama (**In-place**) (Küçük bir fonksiyon çağrı yığıtı ya da sabit sayıda değişken olabilir): Raftaki kitapları rafta sıralama
 - Harici bellek alanına sıralama (**out of place sort**): Raftaki kitapları yere döküp sıralama
- Tekrar Eden Değerler:
 - İstikrarlı (**Stable**): Girdide tekrar eden öğeler, çıktıda aynı sıra ile yer alırlar. 1. tekrar ilk, 2. tekrar ikinci, N. tekrar sonuncu ...
 - İstikrarsız (**Unstable**): Tekrarların yer garantisi yok.

Temel Yöntemler

- Brute-Force
 - Selection Sort
 - Bubble Sort
 - Shell Sort
- Decrease-and-conquer
 - Insertion Sort
- Transform-and-conquer
 - HeapSort
- Divide-and-conquer
 - MergeSort
 - QuickSort

Merge Sort

- Fikir: İki sıralı diziyi birleştirirsek, bir sıralı büyük dizi elde ederiz.
- Rekürsif olarak
 - ikiye böl,
 - Bölünen parçaları kendi içinde sırala,
 - Sıralı sonuçları birleştir.
- + N elemanı $O(N \lg N)$ 'de sıralar.
- - N ile orantılı ekstra Alana ihtiyaç duyar.
 - Yerinde birleştirme yapabilmeliyiz.

Quick Sort

- Tipik olarak tüm değişken türleri için çalışan, popüler bir algoritmadır.
- Genel olarak $O(N \lg N)$
 - En kötü durumda $O(N^2)$ olabilir.
 - İç döngü çok kısa olduğu için teoride ve pratikte hızlı-efektif çalışır.
- MergeSort'un tümleyenidir.
 - Mergesort'ta diziyi sıralanacak iki diziye böldük ve sıralı diziyi oluşturmak için birleştirdik.
 - Tüm dizide çalışmadan **önce** iki rekrürsif çağrı
 - Ortadan ikiye böldük
 - Quicksort'ta iki alt dizi sıralı olduğunda tüm dizinin sıralı olmasını sağlayacak şekilde dizimizi yeniden organize ederiz.
 - Tüm dizide çalıştıktan **sonra** iki rekürsif çağrı
 - Bölme yerimiz dizinin içeriğine göre değişir.
- MergeSort N 'lik ekstra alan,
- QuickSort $c \log N$ 'lik ekstra alana ihtiyaç duyar

Radix Sort

- Basamak basamak sıralama yapılır.
 - En düşük anlamlıdan en yüksek anlamlıya
 - En yüksek anlamlıdan en düşük anlamlıya

Sıralama Algoritmalarının Karşılaştırması

Algoritma	Ort. Karmaşıklık	İstikrarlı?	Yerinde sıralama?	Ekstra alan
Selection	$O(n^2)$	Hayır	Evet	1
Insertion	$O(n^2)$ ($n^2/2$)	Evet	Evet	1
Shell	$O(n^2)$ ($n^{3/2}$)	Hayır	Evet	1
Merge	$O(n \lg_2 n)$	Evet	Hayır	N
Quick	$O(n \lg_2 n)$ ($1.39n \lg n$)	Hayır	Evet	$C \lg N$
Radix LSD	$O(n)$ ($2nw$)	Evet	Hayır	$N+R$
Radix MSD	$O(n)$ ($2nw$)	Evet	Hayır	$N+DR$ (D fonksiyon stack derinliği)
Heap	$O(n \lg_2 n)$ ($2n \lg n$)	Hayır	Evet	1

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.1

2020-2021 Güz Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

TOPOLOGICAL SORT

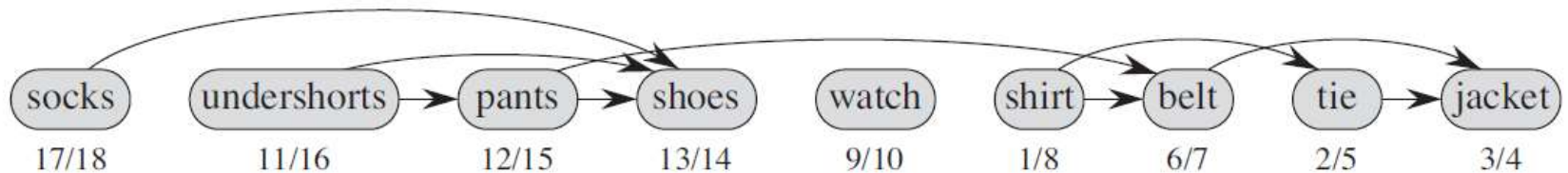
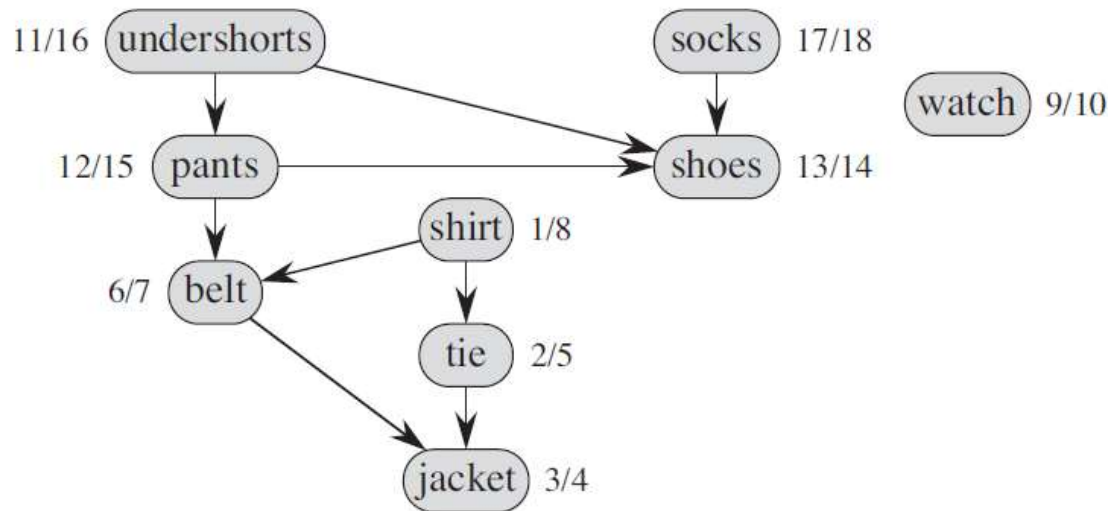
Topological Sort

- $G=(V,E)$ Yönlü Çevrimsiz Çizgesinde (Directed Acyclic Graph) (u,v) kenarı varsa, u 'nun v 'den önce gelmesinin sağlanması.
- Grafin tüm kenarlarını soldan sağa doğru dizmek gibidir.
 - DFS ile gezerek, düğümlerin bitiş zamanlarını bulur ve ona göre sıralarız.
- Olayların oluş sırasını modellemek için kullanılabilir.

TOPOLOGICAL-SORT(G)

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

Topological Sort



Strongly Connected Components

- Bir başka DFS uygulaması
- $G(V,E)$ yönlü grafının güçlü bağlı bileşenleri, $C \subseteq V$ olacak şekilde olası en büyük düğüm kümesidir.
 - Öyle ki, bu kümede yer alan tüm u ve v düğümleri için hem u 'dan v 'ye, hem de v 'den u 'ya yol vardır (u ve v bir diğerinden erişilebilir durumdadır).
 - İki DFS ile bulabiliriz. $DFS(G)$, $DFS(G^T)$
 - $G^T = (V, E^T)$, $E^T = \{(u,v) : (v,u) \in E\}$ (Kenarların yönleri ters)
 - Tanım gereği, her iki DFS gezintisinde de aynı SCC'leri buluruz.

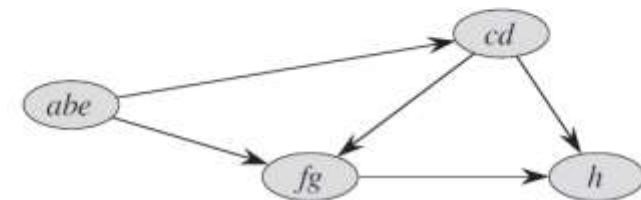
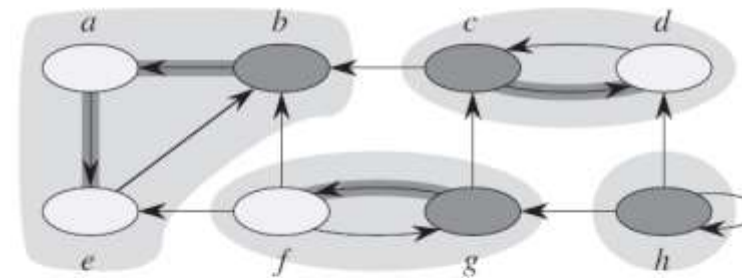
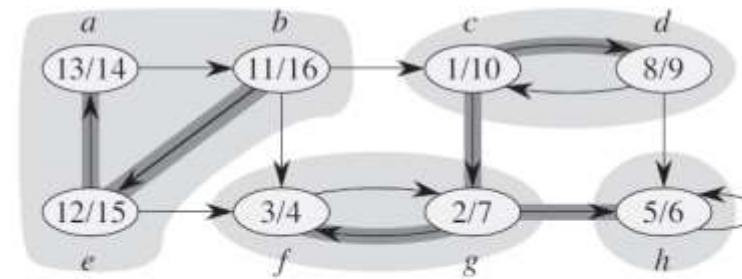
Strongly Connected Components

STRONGLY-CONNECTED-COMPONENTS(G)

- 1 call DFS(G) to compute finishing times $u.f$ for each vertex u
- 2 compute G^T
- 3 call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
- 4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

Strongly Connected Components

1. Her işaretli bölge, keşif ve bitiş zamanlarını içeren düğümlerden oluşan sıkı bağlı bileşenleri gösterir.
2. G^T 'deki her bir sıkı bağlı bileşen, Depth-First ormanındaki bir ağaçtır.
 1. b, c, g, h düğümleri bu DFS ağaçlarının kökleridir.
3. Çevrimsiz bileşen grafi GSCC, bileşenlerin içindeki kenarların tümünü birleştirip her bileşende tek bir düğüm oluşturularak elde edilir.



MINIMUM SPANNING TREE

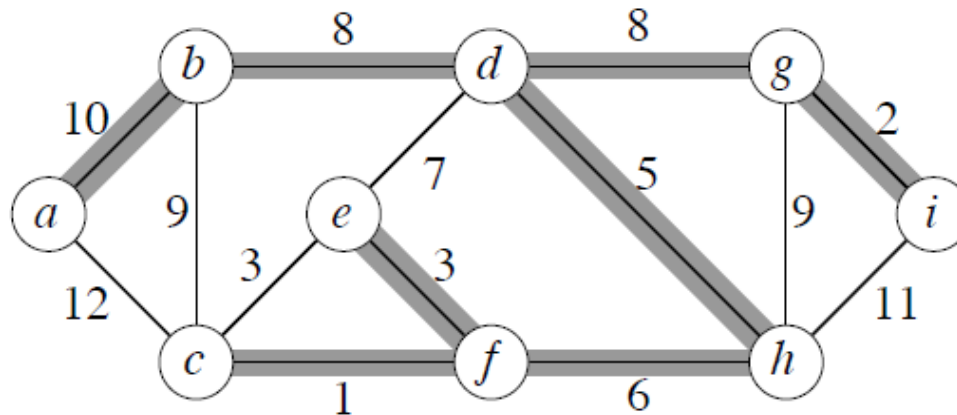
Asgari Tarama/Örtme Ağacı

Minimum Spanning Tree

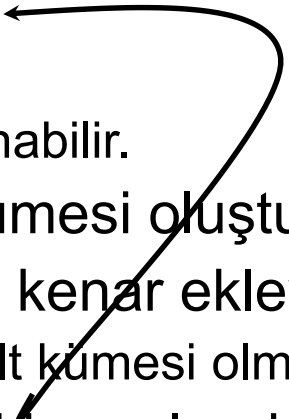
- Bir köyün muhtarisiniz.
 - Sorumluluğunuz gereği, köydeki tüm evleri birbirine bağlamanız gerek.
 - Bir yol parçası w ile 2 ev (u, v) bağlanabilir.
 - Bu yol parçasının yapım/tamir masrafı $w(u,v)$ 'dir.
 - Hedefiniz,
 - Herkesi birbirine bağlı tutacak (bir evden diğer tüm evlere ulaşılabilir)
 - Bunu minimum masraf ile yapacak
- Yol ağını oluşturmaktır.

Minimum Spanning Tree

- Çözüm: Köyünüzü yönsüz graf olarak modelleyin. $G=(V,E)$
- Her $(u,v) \in E$ kenarı için bir **ağırlık** $w(u,v)$ atayın.
- Öyle bir $T \subseteq E$ bulun ki;
 - T tüm düğümleri birbirine bağlasın (tarasın/örtsün)
 - $w(T) = \sum_{(u,v) \in T} w(u,v)$ Değeri asgaride kalsın.
- Tüm örten ağaçlar içerisinde ağırlıklarının toplamı en az olan ağaca asgari tarama ağacı (minimum spanning tree) adı verilir.



Minimum Spanning Tree

- $G(V,E)$ grafımızda tanımlı $w:E \rightarrow \mathbb{R}$ ağırlık fonksiyonumuz olsun.
 - G için tanımlı bir MST (min w) bulmak istiyoruz.
 - MST'nin $|V|-1$ kenarı olmalı.
 - Çevrim (cycle) içermemeli.
 - G için birden fazla MST bulunabilir.
 - Kenarlardan oluşan bir A kümesi oluşturmaliyiz.
 - Boş küme ile başlayıp, A 'ya kenar ekleyerek büyütürüz.
 - Döngü şartı: A , bir MST'nin alt kümesi olmalıdır.
 - Bu durumda ancak **güvenli** kenarları kümeye eklemeliyiz.
 - Her adımda bir stratejiye göre davranıp eylem gerçekleştiriyoruz. ➔ Greedy çözümler MST oluşturmak için uygun.
- 

Generic MST Algoritması

GENERIC-MST(G, w)

1 $A = \emptyset$;

2 **while** A does not form a spanning tree

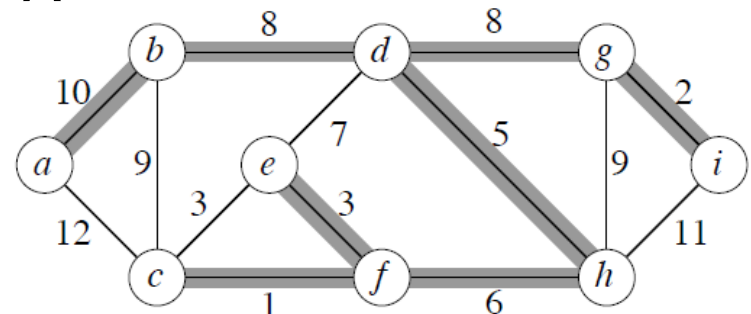
3 find an edge (u,v) that is safe for A

4 $A = A \cup \{(u,v)\}$

5 **return** A

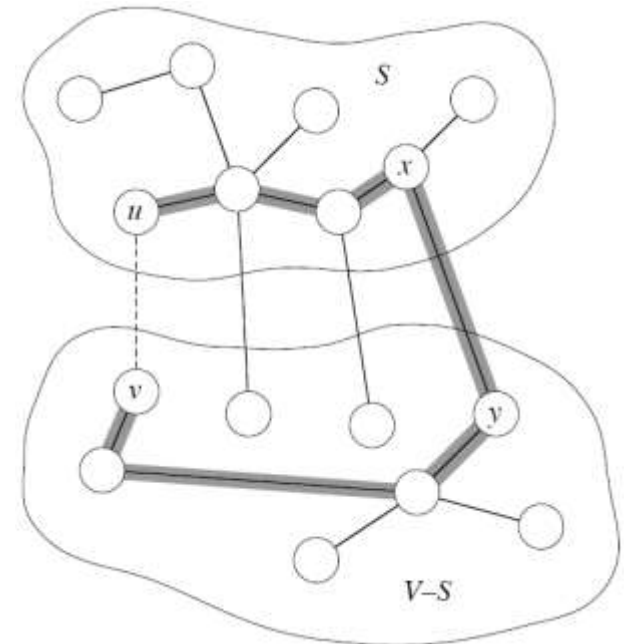
- Güvenli kenar nasıl bulunacak?

- (c,f) kenarı en düşük maliyetli olan. A için güvenli mi?
- O ana kadar oluşan ağaçta $c \in S$ olsun ama f düğümü yok ($f \in V-S$)
- Herhangi bir MST'de en az bir kenar ile f de ağaca bağlanmalıdır. Neden en ucuz olanını seçmeyelim ki? (Greedy seçim stratejisi)



Generic MST Algoritması

- S ve V-S ayırık kısımlarına kesim (cut) diyelim.
- $(S, V-S)$ kesimleri arasında kenarlar $(u,v) \in E$ olabilir.
- Bu kesimler arasında bağlantıyı hafifleten bir kenar bizim için güvenlidir.
- Örnekte, (u,v) kenarının değeri (x,y) kenarının değerinden küçük olsun. (x,y) 'yi kaldırıp (u,v) 'yi ekleriz, Spanning tree yapısını bozmamış oluruz.



Generic MST Algoritması

- Algoritmadaki A kümesi, bağlı bileşenlerden oluşan bir ormandır.
 - Başlangıçta, her bileşen tek bir düğümdür.
- Herhangi bir güvenli kenar, iki bileşeni birleştirerek tek bileşen haline getirir.
 - Her bileşen bir ağaçtır.
- MST'de $|V|-1$ kenar olduğundan, döngü $|V|-1$ kere döner.
 - $|V|-1$ güvenli kenarı eklediğimizde, elimizde tek bir bileşen oluşur.
- Bu noktadan devam edersek, Kruskal'ın MST algoritma çözümüne ulaşırız.

KRUSKAL MST

Kruskal'ın MST Algoritması

- Her düğüm bir bileşen olacak şekilde başlar.
- Tekrarlı olarak, iki bileşeni birbirine bağlayacak «hafif» bir kenar bularak devam eder (kesimler arasındaki hafif kenar).
- Kenarları, ağırlıklarına göre artan şekilde tarar.
- Bir kenarın farklı bileşenlerdeki düğümleri bağlayıp bağlamadığını belirlemek için ayrık küme şeklinde veri yapısı kullanır.

Kruskal'in MST Algoritması

KRUSKAL(V, E, w)

$A \leftarrow \emptyset$

for each vertex $v \in V$

do MAKE-SET(v)

sort E into nondecreasing order by weight w

for each (u, v) taken from the sorted list

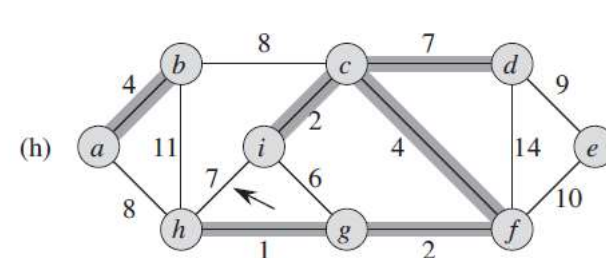
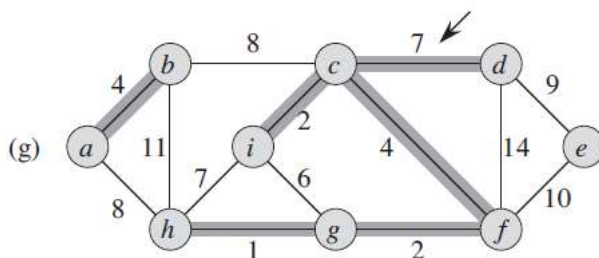
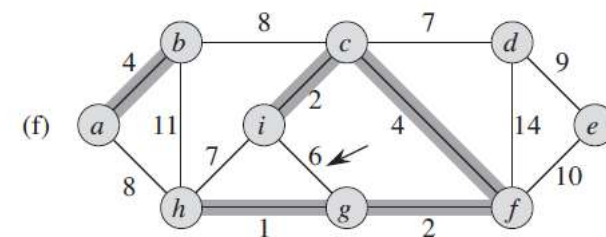
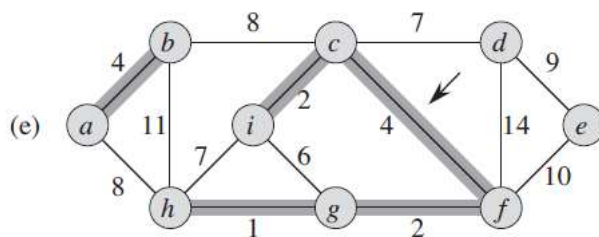
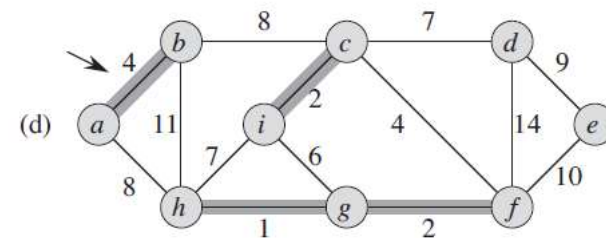
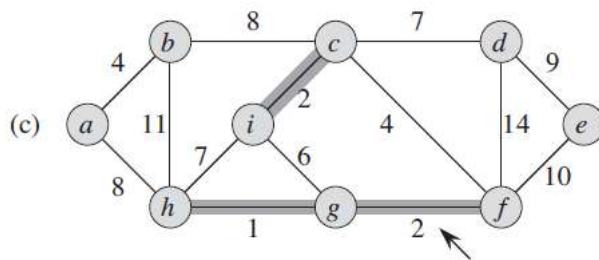
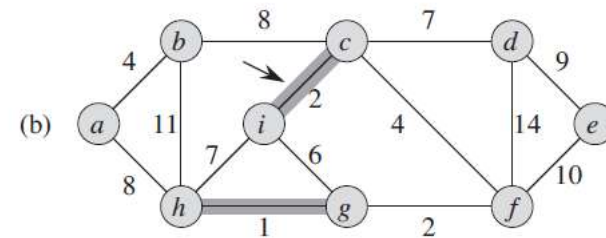
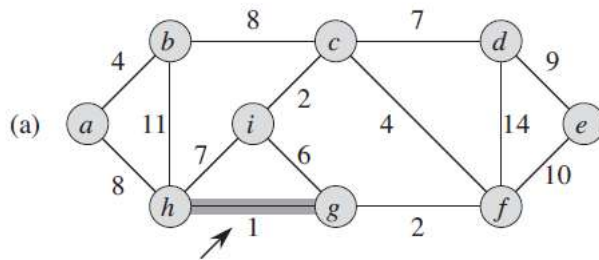
do if FIND-SET(u) = FIND-SET(v)

then $A \leftarrow A \cup \{(u, v)\}$

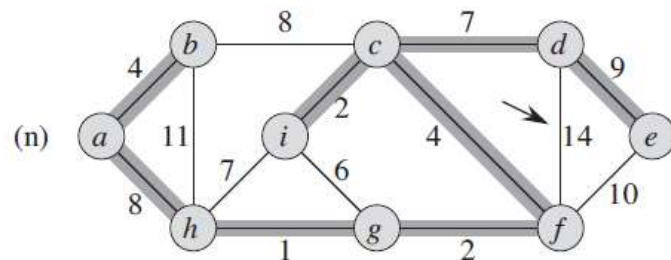
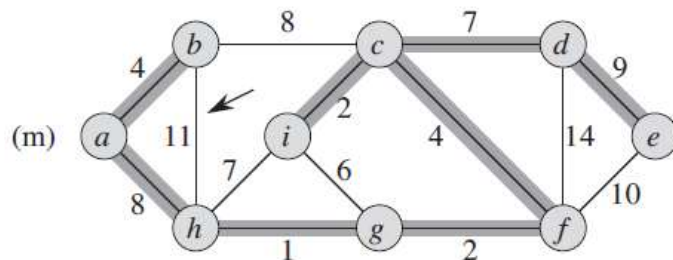
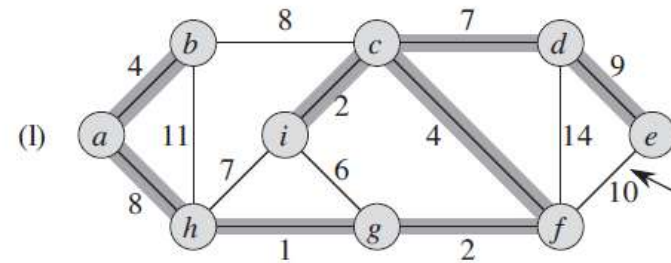
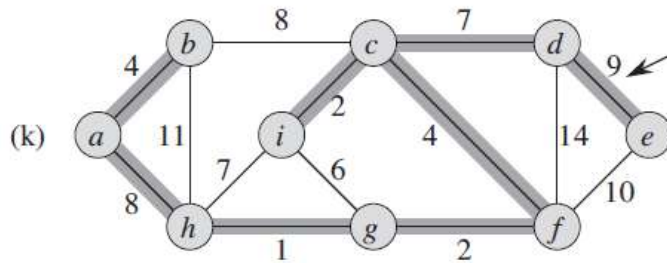
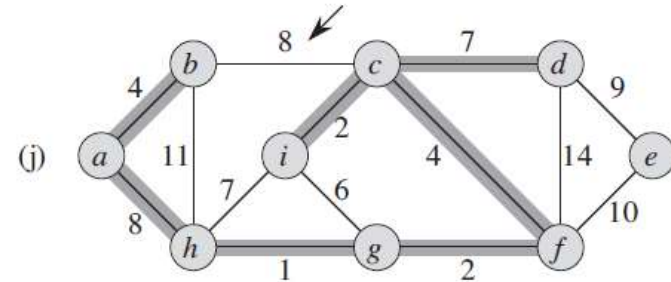
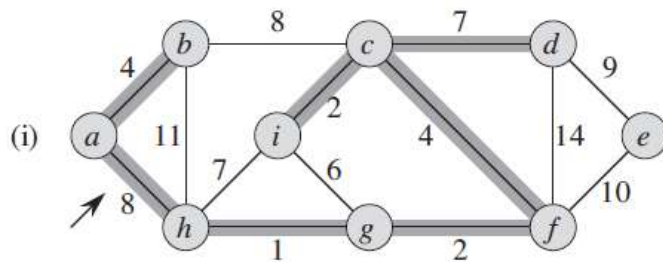
 UNION(u, v)

return A

Kruskal'in MST Algoritması



Kruskal'in MST Algoritması



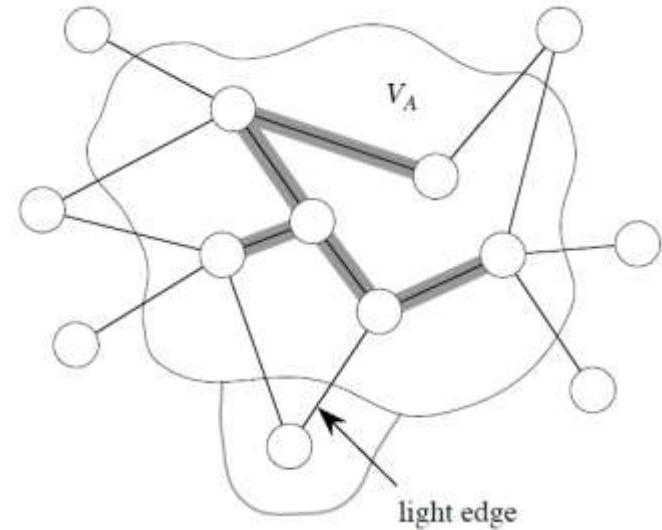
Kruskal MST Karmaşıklık Analizi

- Init A: $O(1)$
- İlk for döngüsü: $|V|$ (MAKE_SET)
- E'ye göre sıralama: $O(E \lg E)$
- İkinci for döngüsü: $O(E)$ FIND_SET ve UNION
- $O(E \lg V)$
- Eğer kenarlar sıralıysa, $O(E \alpha(V))$, neredeyse lineer

PRIM MST

Prim'in MST Algoritması

- Tek bir ağaç oluşturur.
 - A her zaman bir ağaçtır.
- Keyfi bir r kökü ile başlar.
- Her adımda, kesimler arasında bir hafif kenar bularak ağaca ekler. (V_A , $V-V_A$)
- Hafif kenar nasıl bulunabilir? Öncelikli Kuyruk ile.
 - Kuyruktaki elemanlar $V-V_A$ kümesindeki düğümlerdir.
 - Elemanların anahtarları, ağırlık değerleridir.
 - EXTRACT_MIN ile (V_A , $V-V_A$) arasındaki hafif kenar geçişi bulunur.
 - Eğer düğüm V_A 'daki düğümler ile komşu değilse, ağırlığı sonsuzdur.
- A'nın kenarları, r köküne sahip bir ağaç oluşturacaktır.
 - r başlangıcı verilir ama esasen herhangi bir düğüm olabilir.
 - Her düğüm, ağaçtaki ebeveynini ($p[v]$) bilir.
 - Algoritma ilerledikçe V_A büyür, V boş küme olduğunda (tüm düğümlere erişildiğinde) sonlanır.



Prim'in MST Algoritması

$$\text{PRIM}(V, E, w, r)$$
$$Q \leftarrow \emptyset$$

for each $u \in V$

do $key[u] \leftarrow \infty$

$$\pi[u] \leftarrow \text{NIL}$$

INSERT(Q, u)

DECREASE-KEY($Q, r, 0$) $key[r] \leftarrow 0$

while $Q \neq \emptyset$

do $u \leftarrow \text{EXTRACT-MIN}(Q)$

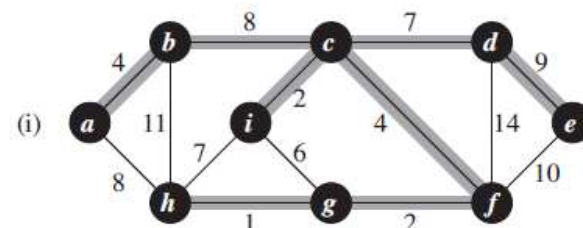
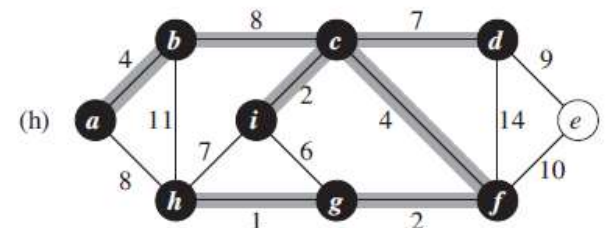
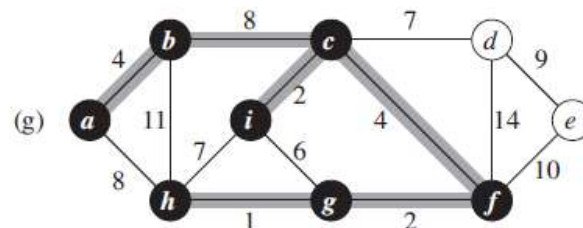
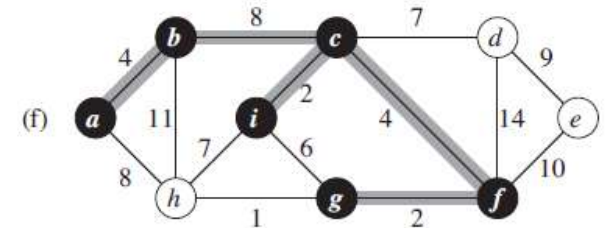
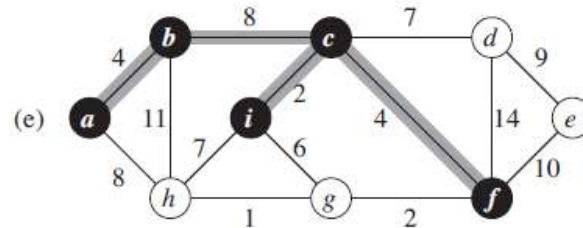
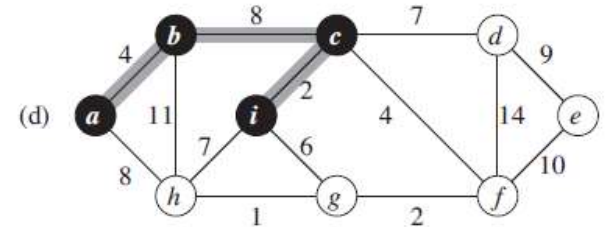
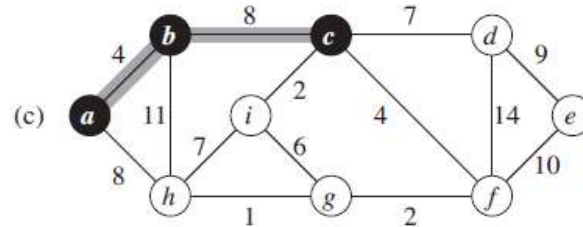
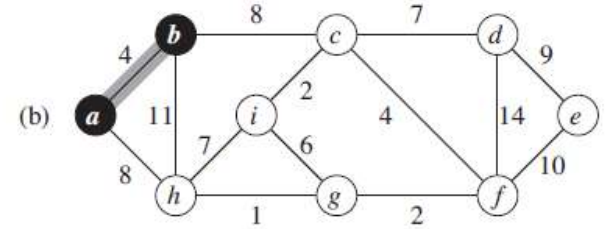
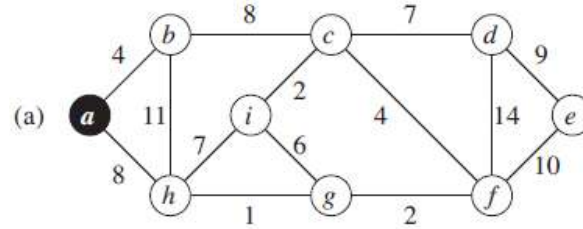
for each $v \in Adj[u]$

do if $v \in Q$ and $w(u, v) < key[v]$

then $\pi[v] \leftarrow u$

DECREASE-KEY($Q, v, w(u, v)$)

Prim'in MST Algoritması



Prim MST Karmaşıklık Analizi

- Q binary heap ile yapılmış olsun.
- Init Q ve ilk for döngüsü: $O(V \lg V)$
- R anahtar değerinin azaltılması: $O(\lg V)$
- While döngüsü
 - $|V|$ Extract_min çağrısı $\rightarrow O(V \lg V)$
 - $\leq |E|$ Decrease_key çağrısı $\rightarrow O(E \lg V)$
- $O(E \lg V)$
- Eğer (fibonacci heap kullanarak) Decrease_key $O(1)$ 'de yapılırsa, $O(V \lg V + E)$

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.1

2020-2021 Güz Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

SHORTEST PATHS

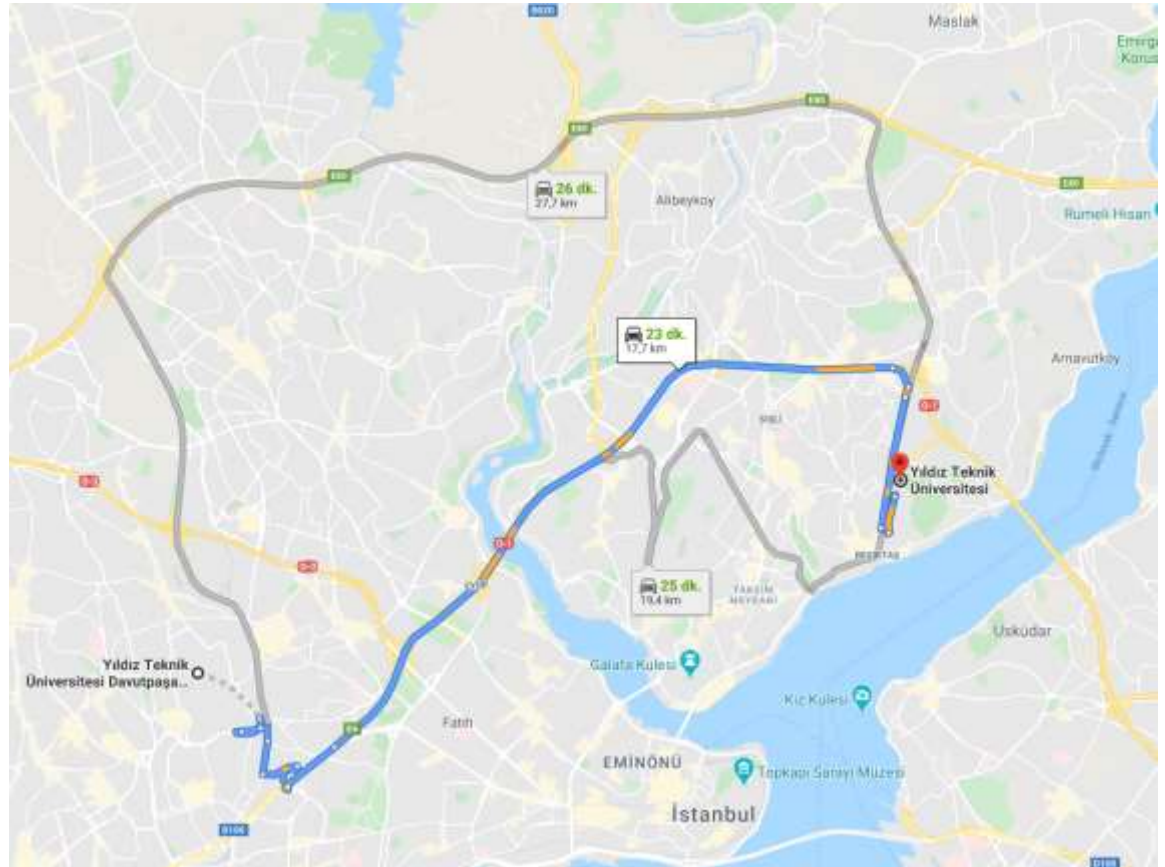
Tek Kaynaktan En Kısa Yollar

Single-Source Shortest Paths

- $G=(V,E)$ Bir yönlü, kenar ağırlıklı graf olsun.
- Amaç: G grafında verilen bir **kaynak** (s) düğümünden yola çıkarak, (erişilebilen-yol olan) diğer tüm düğümlere gidilebilecek en kısa yolları bulmak.
 - S 'den başlayıp, diğer tüm düğümleri gezme problemi ile karıştırılmamalıdır. (Traveling Salesman Problem)
 - Tüm düğümlerden tüm düğümlere : All-pairs Shortest Path
 - Floyd algoritması
- Nereelerde kullanılır?
 - Ulaşım planlama
 - Bilgisayar ağlarında (Internette) packet routing
 - Sosyal ağ analizi
 - Ses tanıma, belge biçimlendirme, robotik, derleyici, havayolu ekip planlama, ...

Kampüsler arası en hızlı yol??

- Brute-Force çözüm:
 - D ğ mler arası olası t m rotaları  ıkar, s relerini hesapla.
 - Sırala
 - En k   k değere sahip olan rotayı se .



Edsger Wybe Dijkstra

- (11 Mayıs 1930 – 6 Ağustos 2002)
- Hollandalı bilgisayar bilimcisi.
- En kısa yol algoritmasını 1950'lerin ortalarında (1956) tasarladı.
- «This was the first graph problem I ever posed myself and solved. The amazing thing was that I didn't publish it. It was not amazing at the time. At the time, algorithms were hardly considered a scientific topic.»
- 1959'da yayınladı.

Dijkstra En Kısa Yol Algoritması

- Yönlü ve yönsüz graflarda kullanılabilir.
 - $G=(V,E)$
- Tüm kenarların pozitif ağırlıkları olmalıdır.
 - $W(u,v) \geq 0$
 - Negatif ağırlıklarla çalışmaz
 - Bellman-Ford algoritmasına bakınız.
- Esasen, BFS arama yönteminin ağırlıklandırılmış bir halidir.
 - FIFO queue yapısı yerine, Priority Queue kullanır.
 - Anahtar değerleri, en kısa yol ağırlıklarıdır ($d[v]$)
- İki düğüm kümesi tutar:
 - S = en kısa yolları hesaplanmış olan düğümler
 - Q = Priority Queue = $V-S$

Dijkstra En Kısa Yol Algoritması

INIT-SINGLE-SOURCE(V, s)

for each $v \in V$
 do $d[v] \leftarrow \infty$
 $\pi[v] \leftarrow \text{NIL}$
 $d[s] \leftarrow 0$

RELAX(u, v, w)

if $d[v] > d[u] + w(u, v)$
 then $d[v] \leftarrow d[u] + w(u, v)$
 $\pi[v] \leftarrow u$

DIJKSTRA(V, E, w, s)

INIT-SINGLE-SOURCE(V, s)

$S \leftarrow \emptyset$

$Q \leftarrow V$ i.e., insert all vertices into Q

while $Q \neq \emptyset$

do $u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

for each vertex $v \in \text{Adj}[u]$

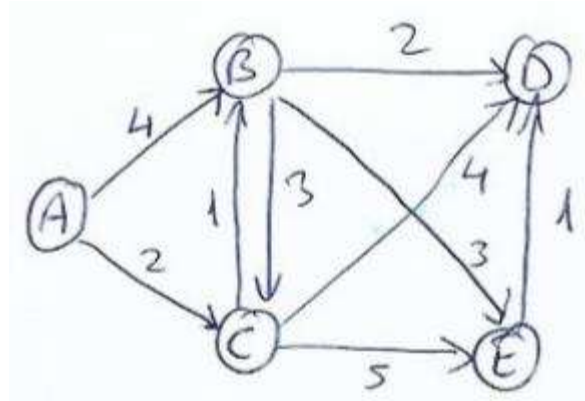
do RELAX(u, v, w)

Prim algoritmasına benzer, farkı $d[v]$ hesabı ve en kısa yol ağırlıklarının anahtar olarak kullanılmasıdır.

Dijkstra En Kısa Yol Algoritması

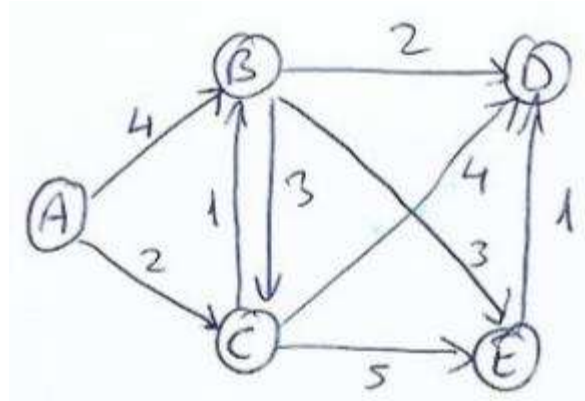
- Tüm düğümler kaynaktan erişilebilir ise,
- Min-priority-queue dizi ile yapılırsa
- $O(V^2)$
- Min-heap ile yapılırsa
- $O(E \log V)$
- Fibonacci heap ile yapılırsa
- $O(E + V \log V)$

Dijkstra En Kısa Yol Algoritması



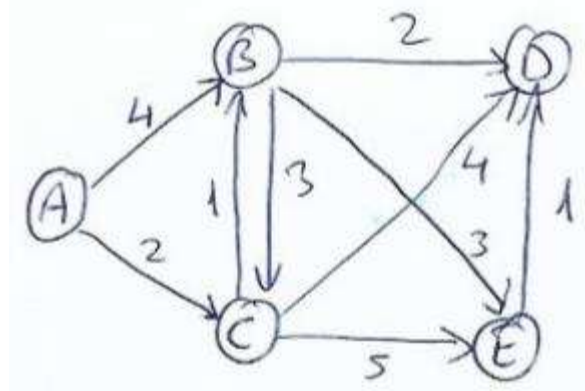
	A	B	C	D	E
1	0	∞	∞	∞	∞

Dijkstra En Kısa Yol Algoritması



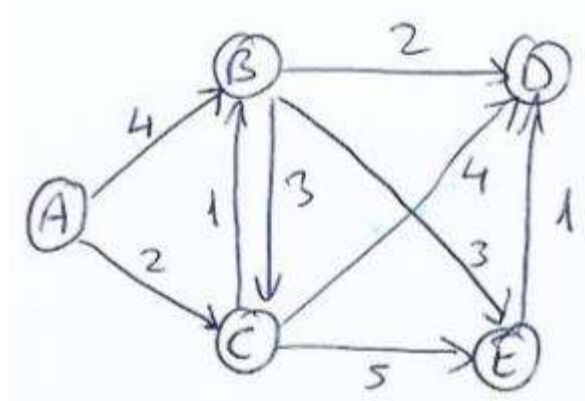
	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4(A)	2(A)	∞	∞

Dijkstra En Kısa Yol Algoritması



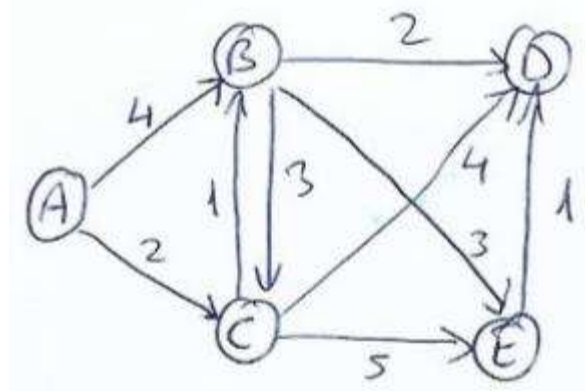
	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4(A)	2(A)	∞	∞
3	0	3 (C)	2(A)	6(C)	7(C)

Dijkstra En Kısa Yol Algoritması



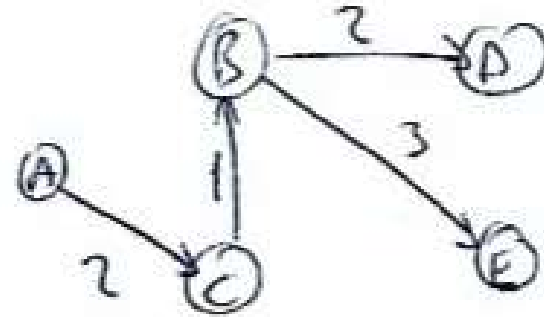
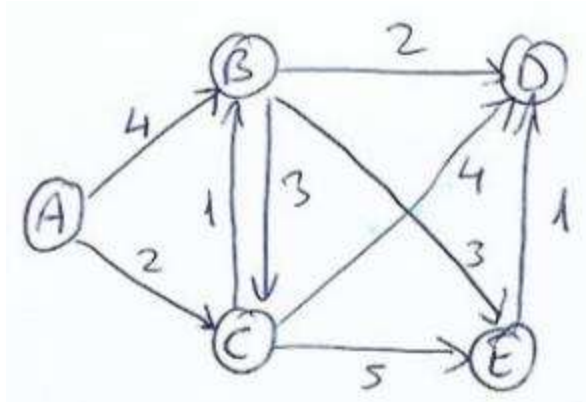
	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4(A)	2(A)	∞	∞
3	0	3(C)	2(A)	6(C)	7(C)
4	0	3(C)	2(A)	5(B)	6(B)

Dijkstra En Kısa Yol Algoritması



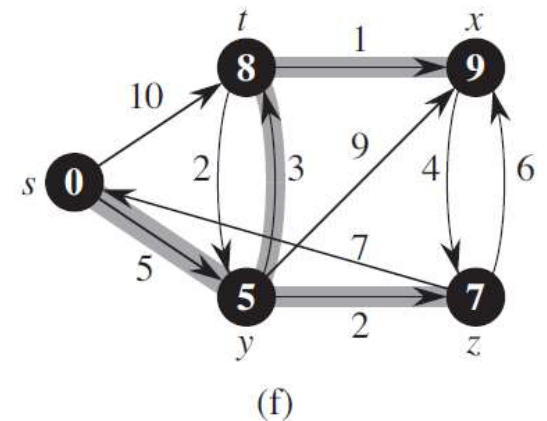
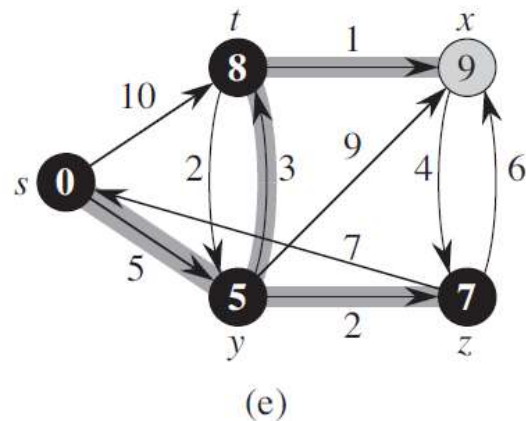
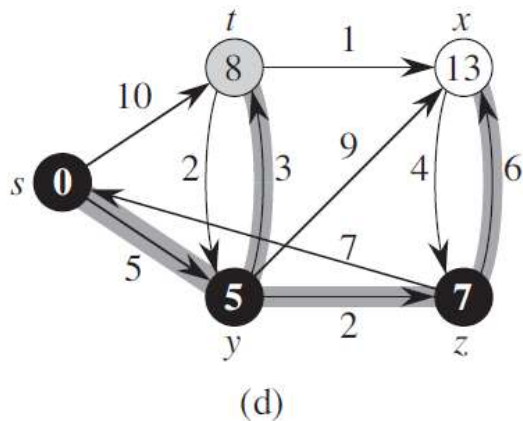
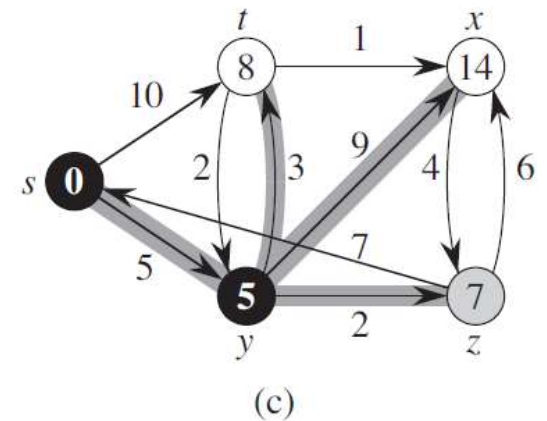
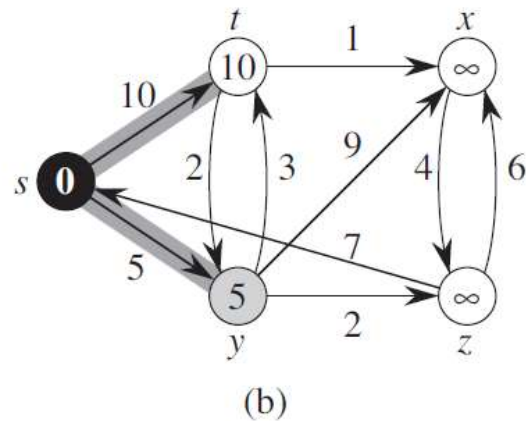
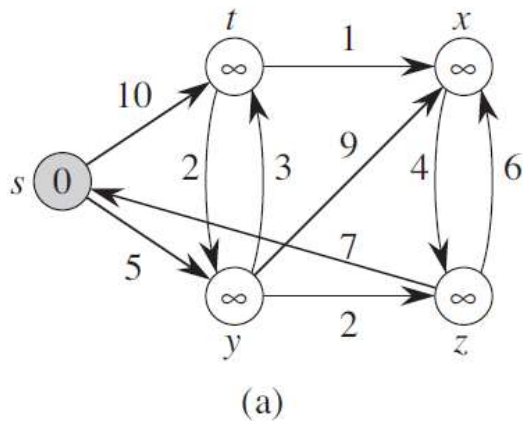
	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4(A)	2(A)	∞	∞
3	0	3(C)	2(A)	6(C)	7(C)
4	0	3(C)	2(A)	5(B)	6(B)
5	0	3(C)	2(A)	5(B)	6(B)

Dijkstra En Kısa Yol Algoritması

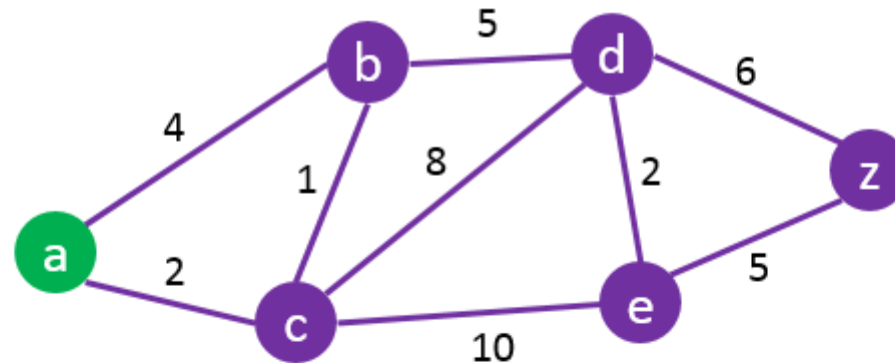


	A	B	C	D	E
1	0	∞	∞	∞	∞
2	0	4(A)	2(A)	∞	∞
3	0	3(C)	2(A)	6(C)	7(C)
4	0	3(C)	2(A)	5(B)	6(B)
5	0	3(C)	2(A)	5(B)	6(B)
6	0	3(C)	2(A)	5(B)	6(B)

Dijkstra En Kısa Yol Algoritması

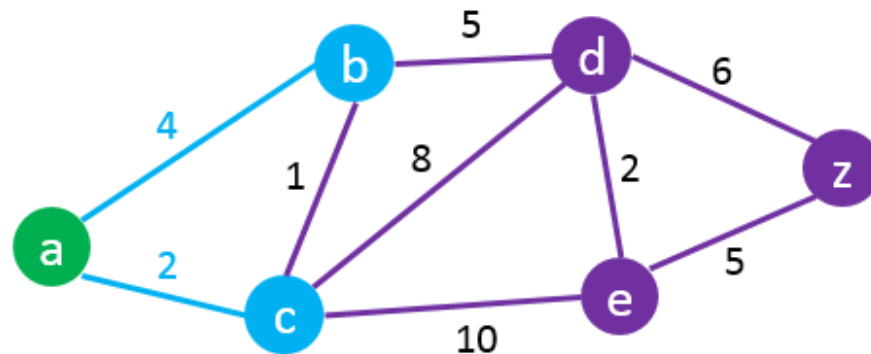


Dijkstra En Kısa Yol Algoritması (a=>z)



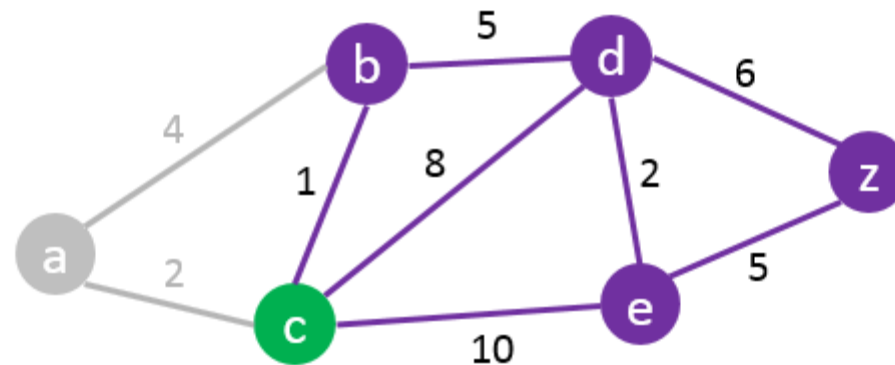
Node	Status	Shortest Distance From A	Previous Node
A	Current Node	0	
B		∞	
C		∞	
D		∞	
E		∞	
Z		∞	

Dijkstra En Kısa Yol Algoritması (a=>z)



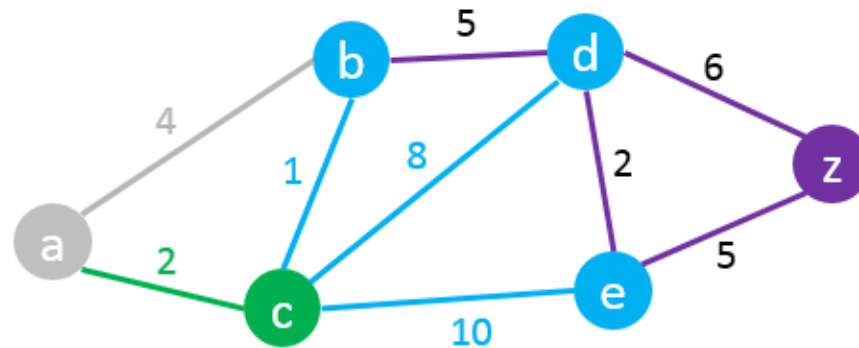
Node	Status	Shortest Distance From A	Previous Node
A	Current Node	0	
B		∞ 4	A
C		∞ 2	A
D		∞	
E		∞	
Z		∞	

Dijkstra En Kısa Yol Algoritması (a=>z)



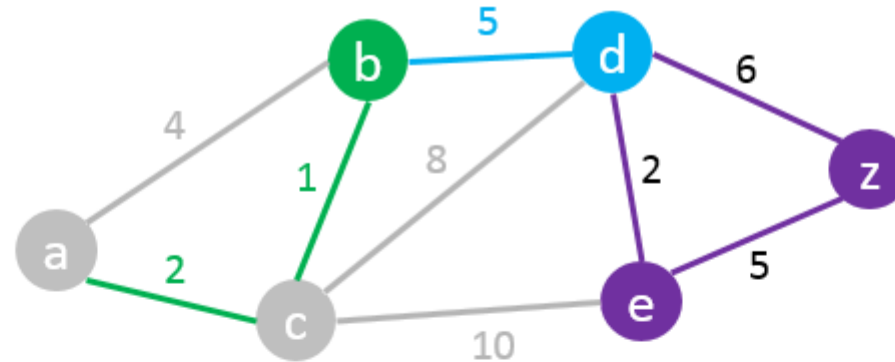
Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B		∞ 4	A
C	Current Node	∞ 2	A
D		∞	
E		∞	
Z		∞	

Dijkstra En Kısa Yol Algoritması (a=>z)



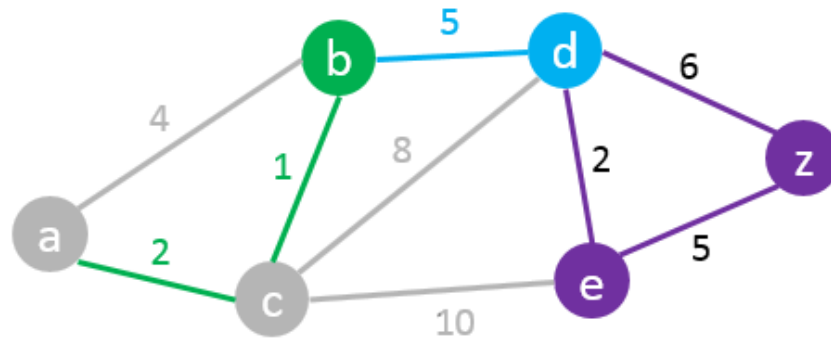
Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B		4 $2+1=3$	C
C	Current Node	2	A
D		∞ $2+8=10$	C
E		∞ $2+10=12$	C
Z		∞	

Dijkstra En Kısa Yol Algoritması (a=>z)



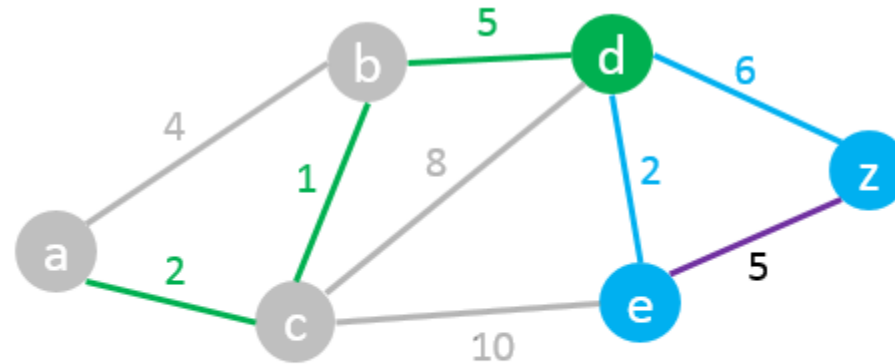
Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Current Node	3	C
C	Visited Node	2	A
D		10	C
E		12	C
Z		∞	

Dijkstra En Kısa Yol Algoritması (a=>z)



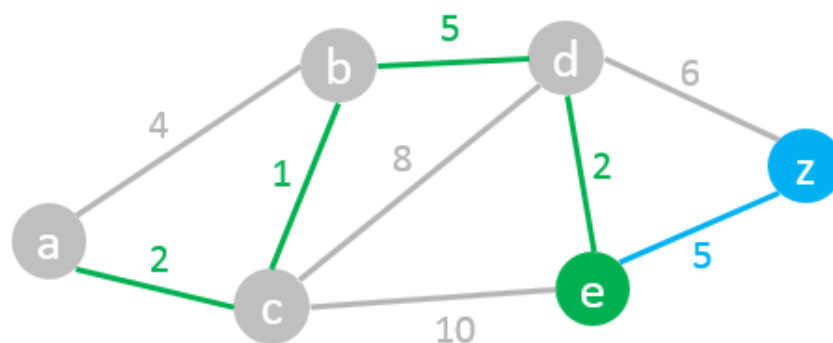
Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Current Node	3	C
C	Visited Node	2	A
D		10 $3+5=8$	B
E		12	C
Z		∞	

Dijkstra En Kısa Yol Algoritması (a=>z)



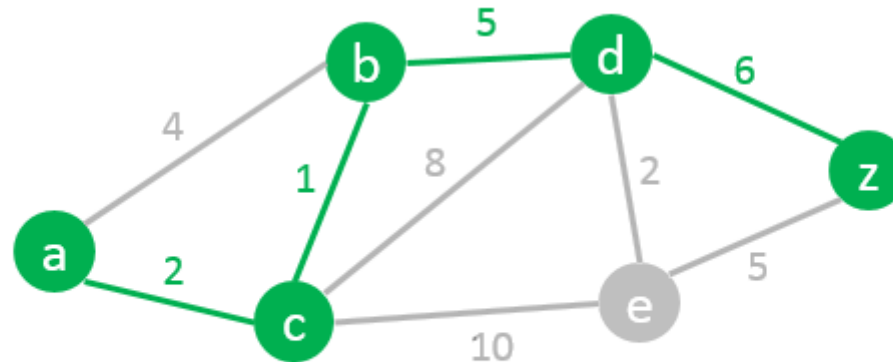
Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Visited Node	3	C
C	Visited Node	2	A
D	Current Node	8	B
E		12 $8 + 2 = 10$	D
Z		∞ $8 + 6 = 14$	D

Dijkstra En Kısa Yol Algoritması (a=>z)



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Visited Node	3	C
C	Visited Node	2	A
D	Visited Node	8	B
E	Current Node	10	D
Z		14 10 + 5 = 15	D

Dijkstra En Kısa Yol Algoritması (a=>z)



Node	Status	Shortest Distance From A	Previous Node
A	Visited Node	0	
B	Visited Node	3	C
C	Visited Node	2	A
D	Visited Node	8	B
E	Visited Node	10	D
Z	Current Node	14	D

THE END
