



Kelime (Text) İşleme Algoritmaları

Prof.Dr.Banu Diri



➤Trie Ağacı

➤Sonek Ağacı (Suffix Tree)

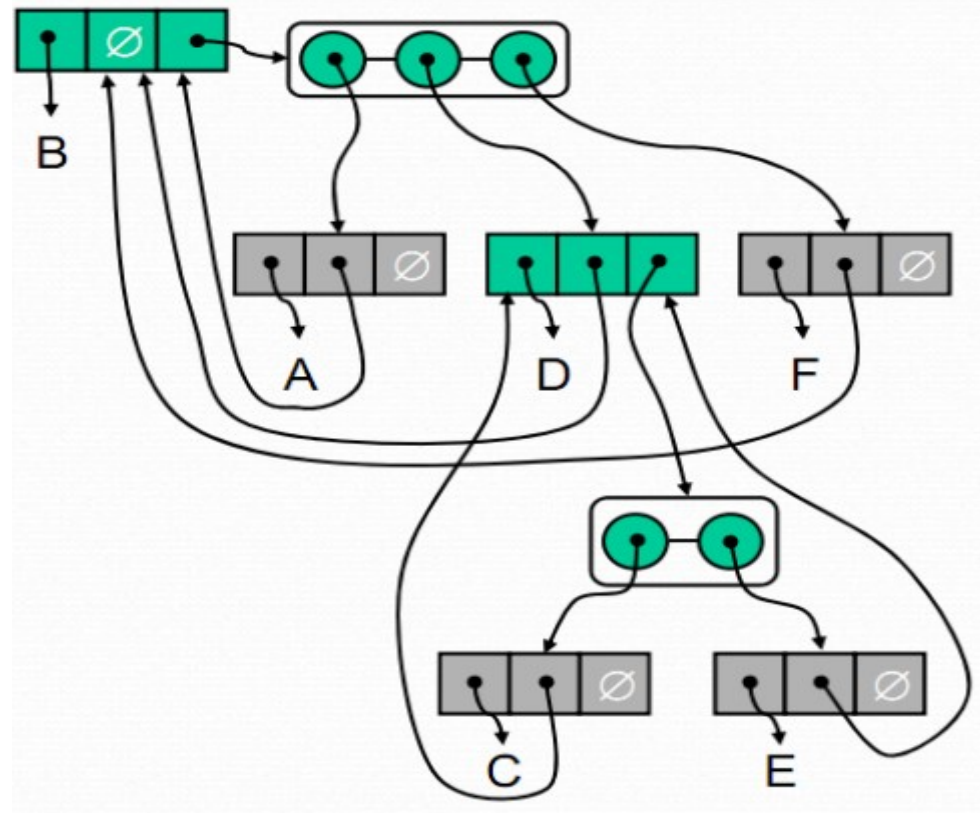
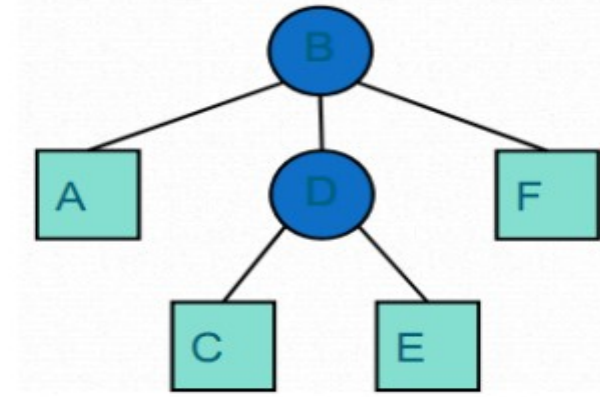
➤Longest Common String (LCS)

➤Minimum Edit Distance

Ağaçların Bağlı Yapısı

- ❖ Düğüm (node), çeşitli bilgiler ile ifade edilen bir nesnedir.
- ❖ Her bir bağlantı (edge) için, birer bağlantı bilgisi tutulur.

- Nesne/Değer (Element)
- Ana düğüm (Parent node)
- Çocuk düğümlerin listesi



Metin ağaçları (TRIE)

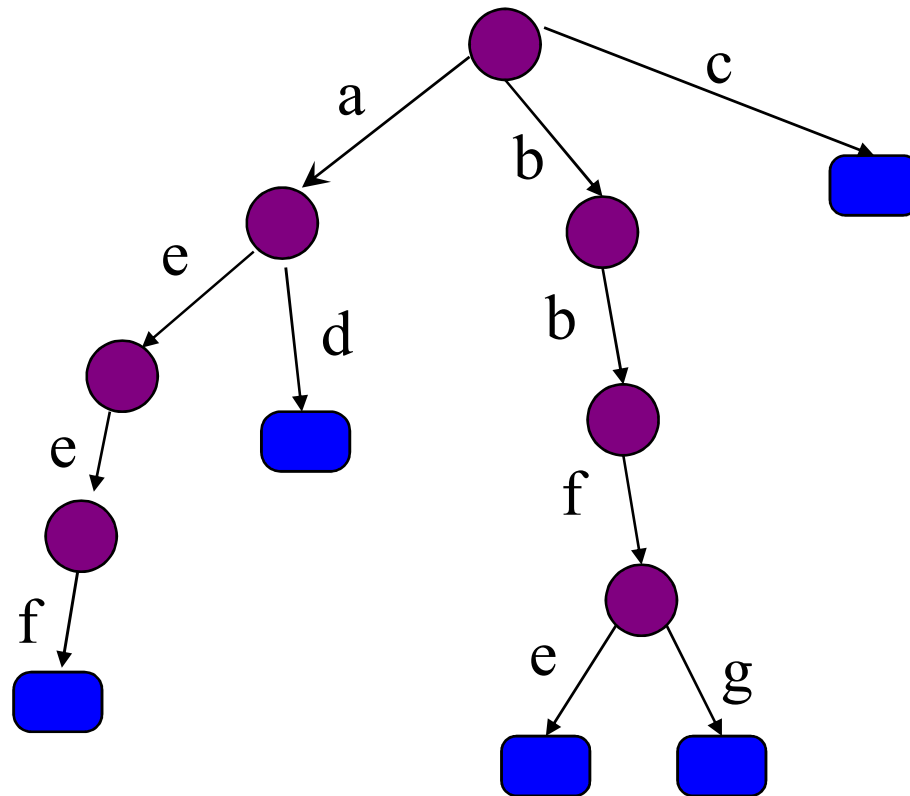
Trie ağacının ismi retrieval kelimesininin [3..6] arasındaki harflerinden oluşmaktadır.

Bir ağacın üzerinde bir metin (string, sözlük, ...) kodlanmak isteniyorsa TRIE ağaçları tercih edilir.

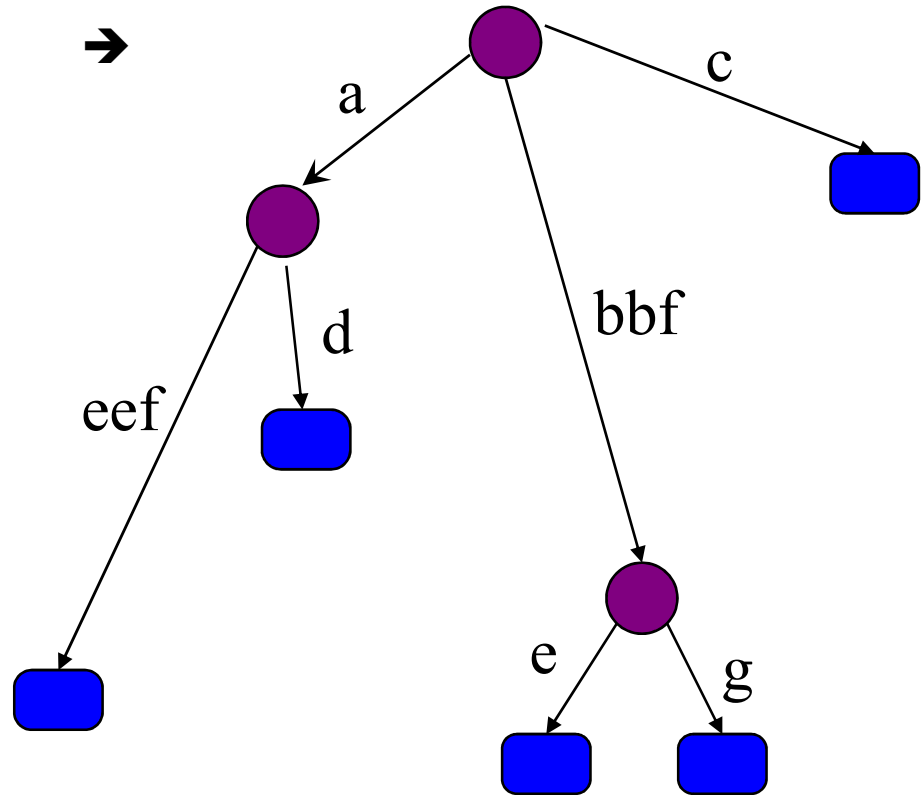
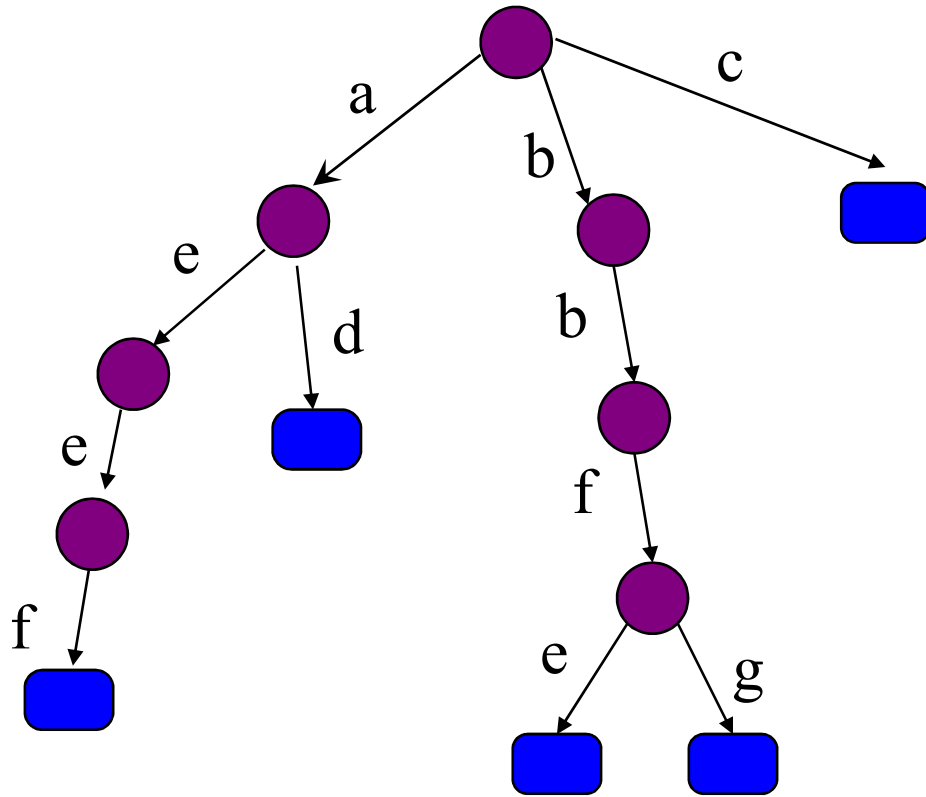
- İgili metni veren ağacın üzerinde izlenebilir tek bir yol vardır.
- Kök düğüm her zaman boş bir metni (string) ifade eder.
- Her düğüm kendisinden sonra gelen harfi işaret eder.
- Boş metin hangi harf ile devam ederse, o harfe ait dal takip edilir ve gelinen düğüm o ana kadar geçilmiş olan dallardaki harflerin birleştirilmiş halidir.
- Bir düğümden bir harf taşıyan sadece bir dal çıkabilir.
- Metin ağaçlarının en önemli avantajı, bir metni ararken metnin boyutu kadar işlem gerektirmesidir .
- Ağaçta ne kadar bilgi bulunduğu önemi yoktur.
- Hafızayı verimli kullanırlar. Trie ağacının en derin noktası, ağaç üzerindeki en uzun metin kadardır.

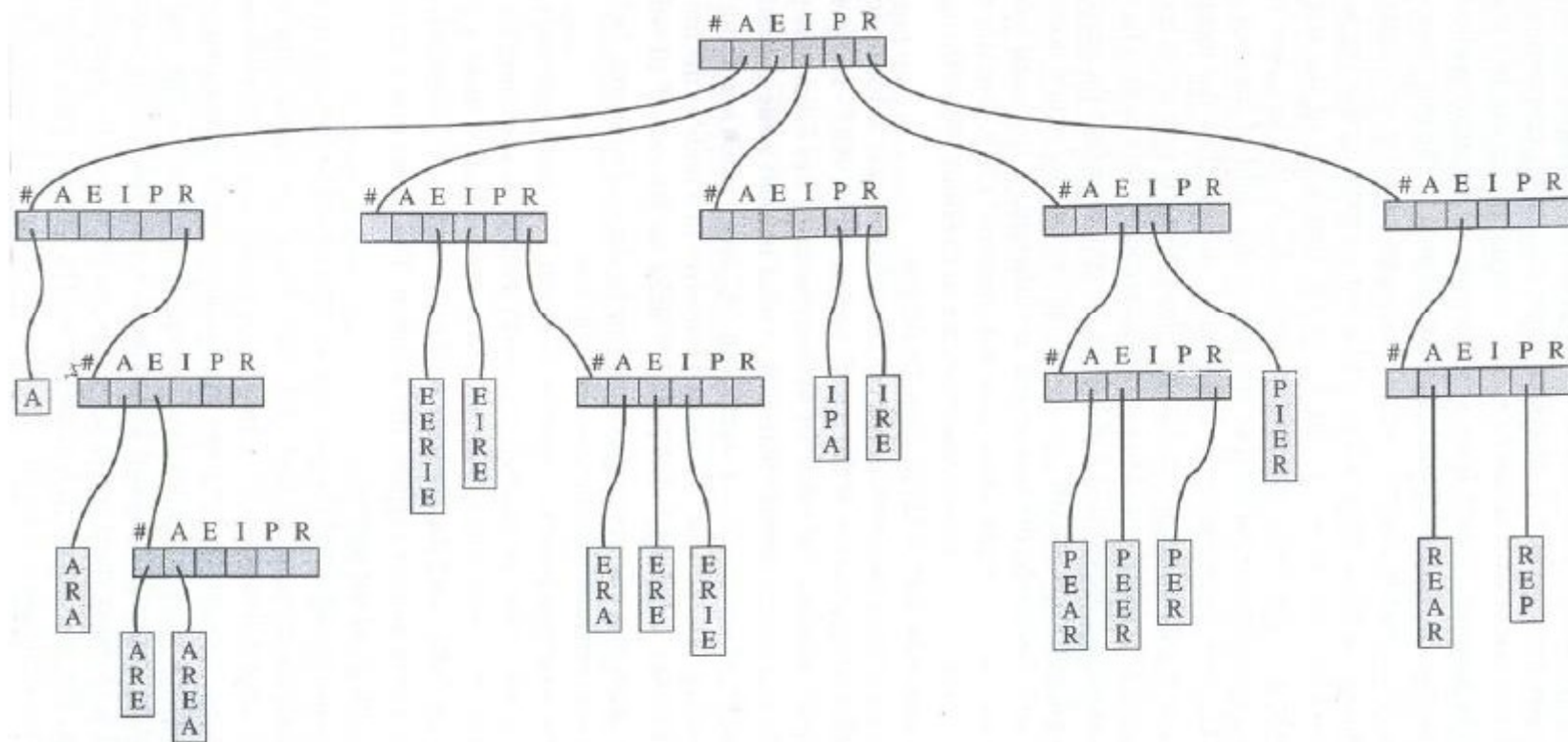
String kümesinin TRIE üzerinde gösterilimi

{
aeef
ad
bbfe
bbfg
c
}

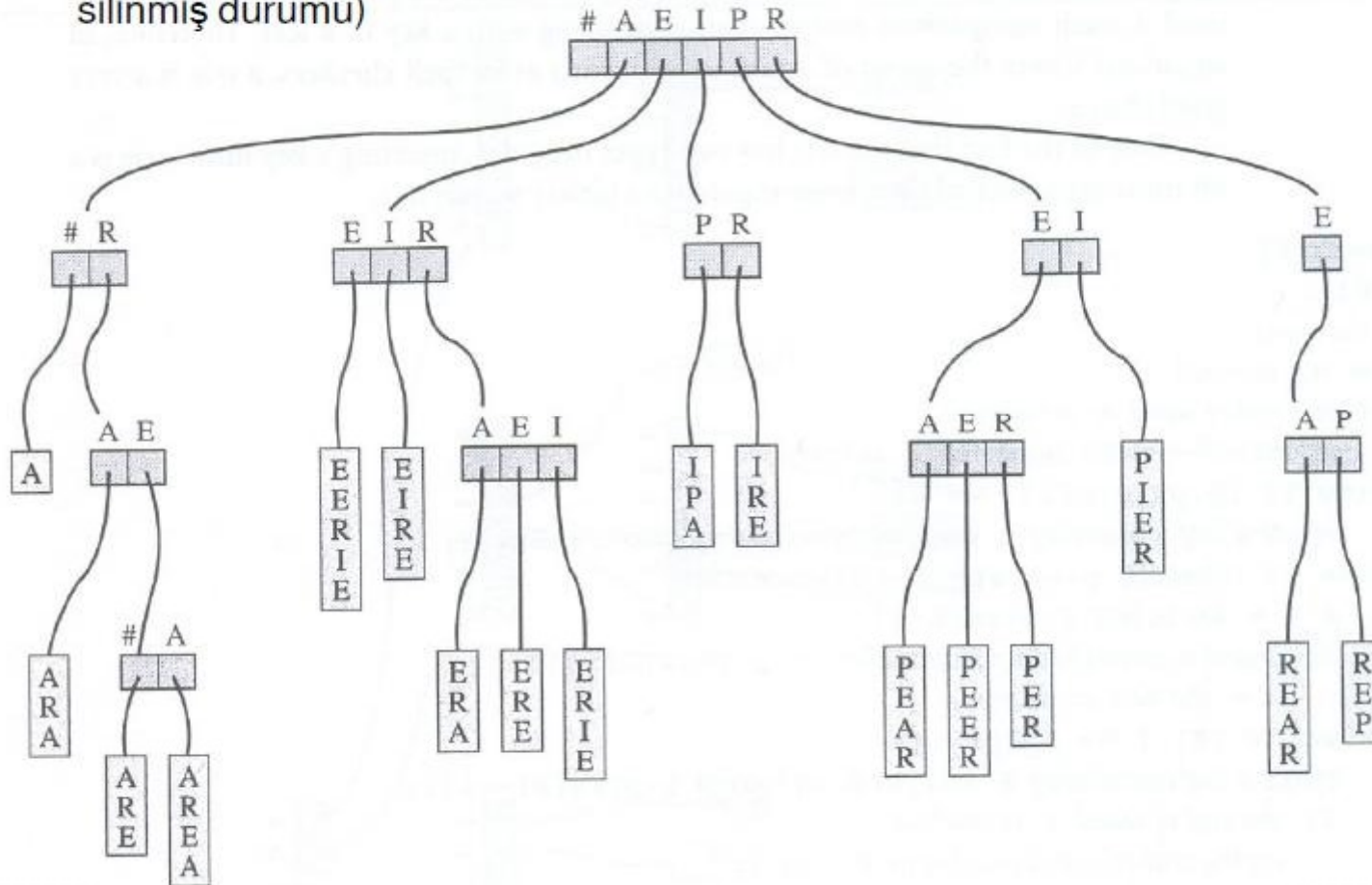


Sıkıştırılmış TRIE

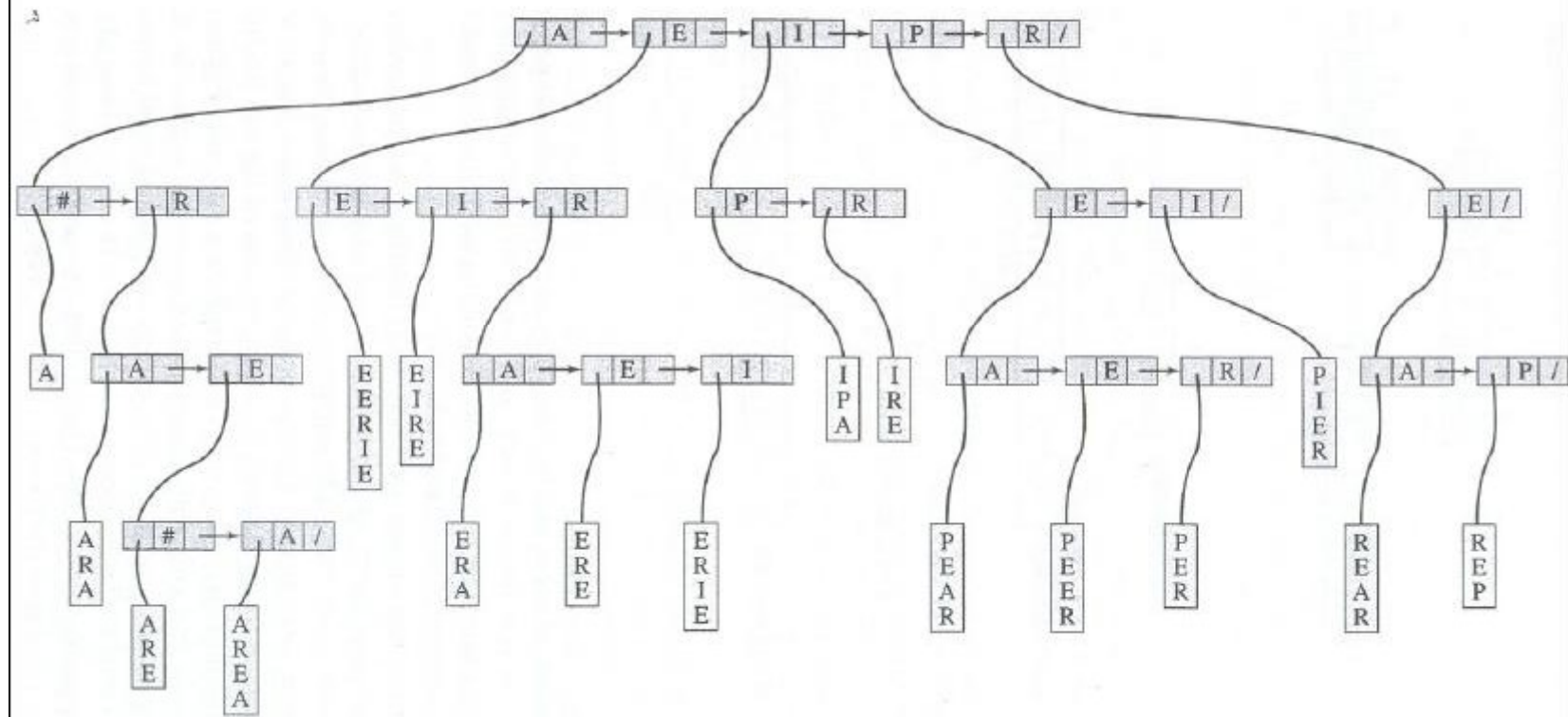




Örnek: A, E, I, P, R harflerinden oluşan bir trie (kullanılmayan alanların silinmiş durumu)



Örnek: A, E, I, P, R harflerinden oluşan bir trie'in binary tree olarak gösterimi



Suffix Tree

- Suffix Tree (Sonek Ağacı) kelime işleme algoritmalarındandır
- DNA dosyaları gigabyte seviyesinde yer kapladıklarından DNA analizinin elle yapılması mümkün değildir. Hatta, DNA dosyalarının bilgisayar yardımıyla işlenmesi de çok uzun sürmektedir.
- Biyolojik veriler, arama motorları, derleyici tasarımı, işletim sistemi, veri tabanı, vs... kullanılır.

Suffix Trees

Substring bulma problemidir...

- Verilen text m uzunluğunda bir string (S)
- S için harcanan zaman $O(m)$
- Bulunması istenen string Q olup, n uzunluğunda olsun
- Q 'nun S içerisinde aranması için harcanan zaman $O(n)$

Suffix Tree ler kullanılarak bu problemi çözebiliriz.

Suffix Tree'nin Tanımı

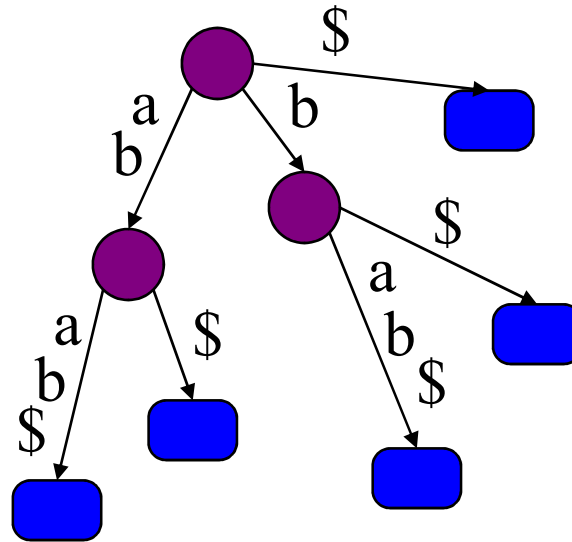
m uzunluğundaki bir S string için T suffix tree aşağıdaki özelliklere sahiptir:

- Köklü bir ağaçtır ve yönlüdür
- 1 ile m arasında etiketlenmiş m yaprağı vardır
- Ağaçtaki her bir dal S string nin bir alt stringini oluşturur
- Kökten, i . yaprağa kadar etiketlenmiş bir yol üzerindeki kenarlar birleştirilebilir
- Kök olmayan her ara düğümün en az 2 yaprağı vardır
- Bir düğümden çıkan kenarlar farklı karakterler ile başlar

S=abab

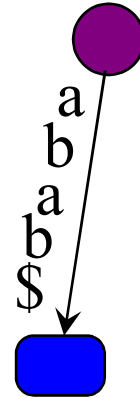
S string'inin suffix tree'si, **S**'nin bütün suffix'lerini sıkıştırılmış bir trie de tutsun. **\$** sembolü ilgili suffix'in sonunu göstereyin.

{
\$
b\$
ab\$
bab\$
abab\$
}

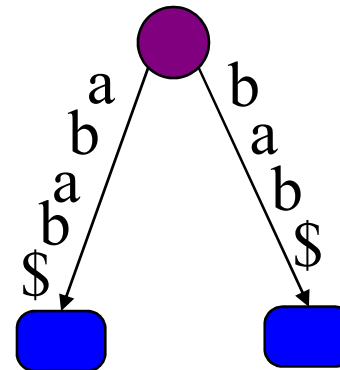


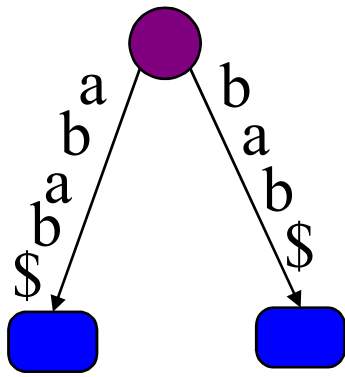
Suffix Tree'nin oluşturulması

En geniş suffix

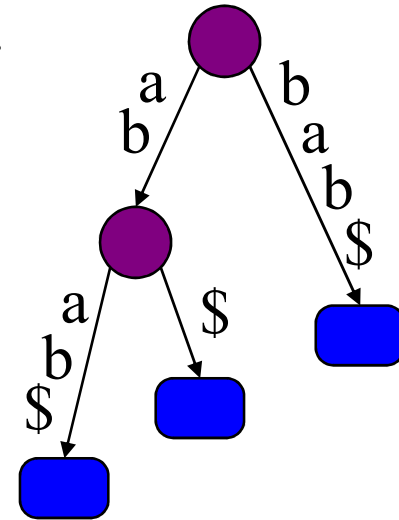


bab\$ suffix'inin eklenmesi

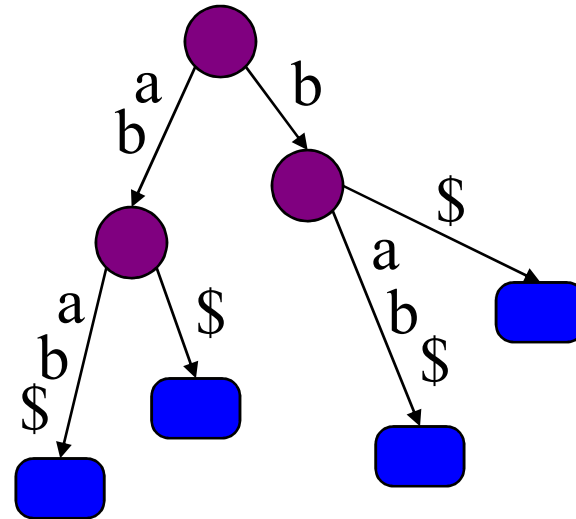


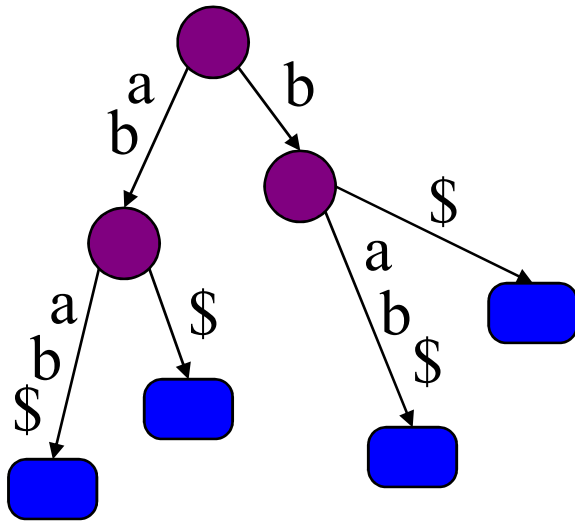


ab\$ suffix'inin eklenmesi

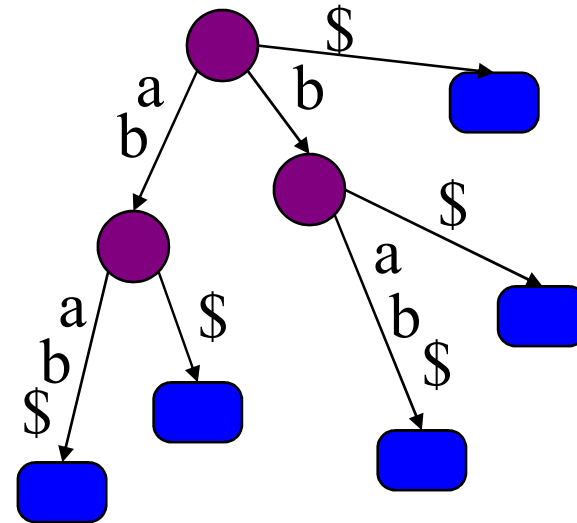


b\$ suffix'inin eklenmesi

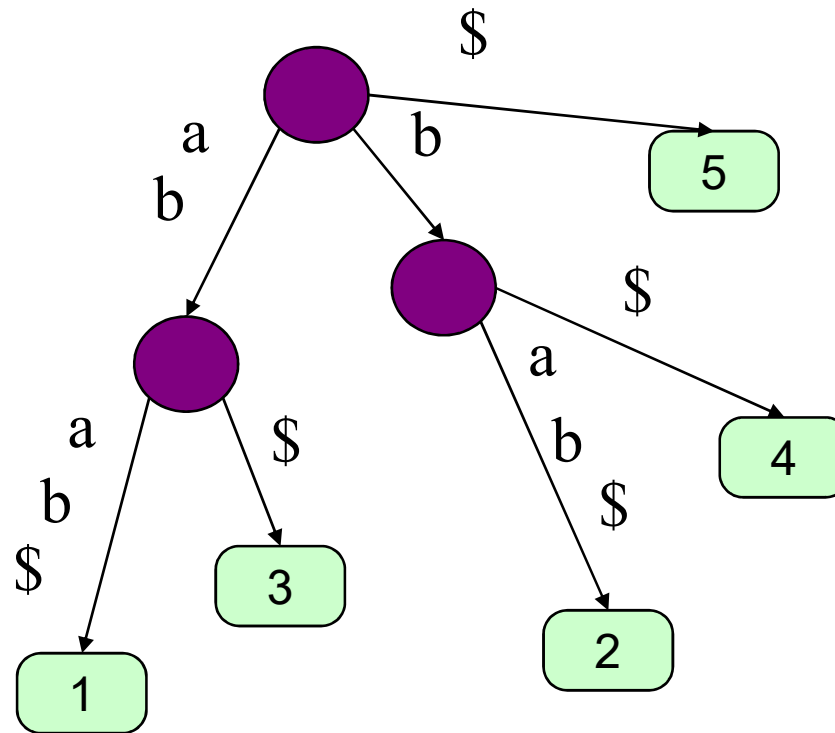




\$ suffix'in eklenmesi



Herbir yaprağı etiketleyerek nerden erişeceğimizi biliriz.



Longest Common Subsequence

A subsequence of a string S , is a set of characters that appear in left-to-right order, but not necessarily consecutively.

Example

ACTTGCG

- ACT, ATTC, T, ACTTGC are all subsequences.
- TTA is not a subsequence

A common subsequence of two strings is a subsequence that appears in both strings. A longest common subsequence is a common subsequence of maximal length.

Example

$S_1 = \text{AAACCGTGAGTTATTCGTTCTAGAA}$

$S_2 = \text{CACCCCTAAGGTACCTTTGGTTC}$

$S_1 = \text{AAACCGTGAGTTATTCGTTCTAGAA}$

$S_2 = \text{CACCCCTAAGGTACCTTTGGTTC}$

LCS is **ACCTAGTACTTTG**

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 , \\ c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\ \max(c[i, j - 1], c[i - 1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . \end{cases}$$

LCS – Length(*X*, *Y*)

```

1  m ← length[X]
2  n ← length[Y]
3  for i ← 1 to m
4      do c[i, 0] ← 0
5  for j ← 0 to n
6      do c[0, j] ← 0
7  for i ← 1 to m
8      do for j ← 1 to n
9          do if xi = yj
10             then c[i, j] ← c[i − 1, j − 1] + 1
11                 b[i, j] ← “↖”
12             else if c[i − 1, j] ≥ c[i, j − 1]
13                 then c[i, j] ← c[i − 1, j]
14                     b[i, j] ← “↑”
15                 else c[i, j] ← c[i, j − 1]
16                     b[i, j] ← “←”
17  return c and b
```

$X_{m=7} \rightarrow \text{ACTTGCG}$
 $Y_{n=6} \rightarrow \text{ATTCGG}$

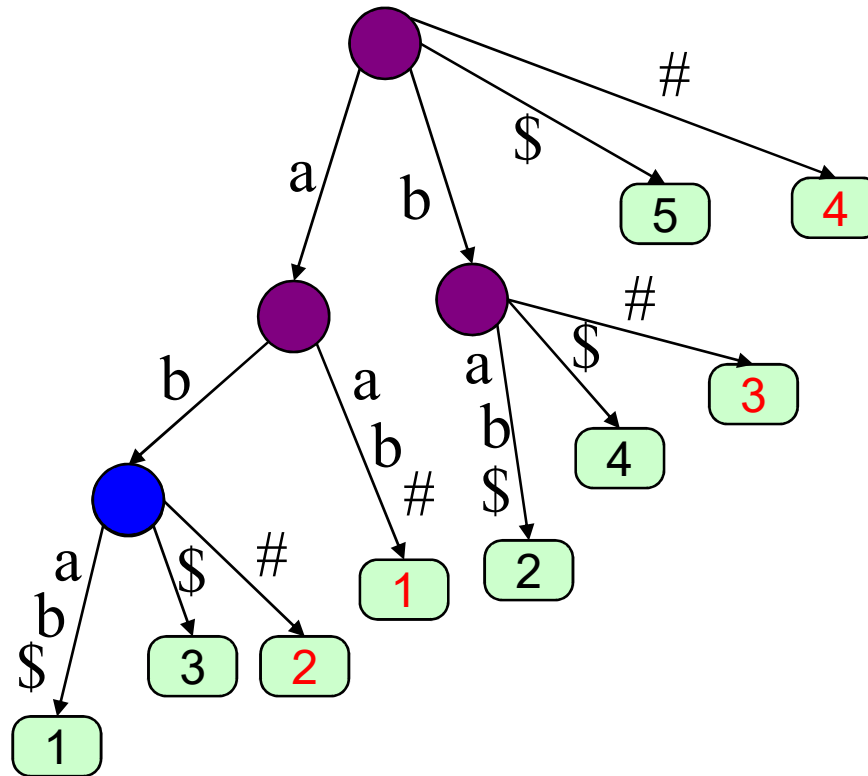
$\text{LCS} \rightarrow \text{ATTGG}$

			A	T	T	C	G	G
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	1↖	1←	1←	1←	1←	1←
C	2	0	1↑	1↑	1↑	2↖	2←	2←
T	3	0	1↑	2↖	2↖	2↑	2↑	2↑
T	4	0	1↑	2↖	3↖	3←	3←	3←
G	5	0	1↑	2↑	3↑	3↑	4↖	4↖
C	6	0	1↑	2↑	3↑	4↖	4↑	4↑
G	7	0	1↑	2↑	3↑	4↑	5↖	5↖

Longest Common Substring (of two strings)

S₁ aab#

S₂ abab\$



Longest Common Suffix

$$LCSuff(S_{1..p}, T_{1..q}) = \begin{cases} LCSuff(S_{1..p-1}, T_{1..q-1}) + 1 & \text{if } S[p] = T[q] \\ 0 & \text{otherwise} \end{cases}$$

Örnek : "ABAB" ve "BABA"

		A	B	A	B
	0	0	0	0	0
B	0	0	1	0	1
A	0	1	0	2	0
B	0	0	2	0	3
A	0	1	0	3	0

function LCSubstr(S[1..m], T[1..n])

L := **array**(1..m, 1..n)

z := 0

ret := {}

for i := 1..m

for j := 1..n

if S[i] = T[j]

if i = 1 or j = 1

 L[i,j] := 1

else

 L[i,j] := L[i-1,j-1] + 1

if L[i,j] > z

 z := L[i,j]

 ret := {}

if L[i,j] = z

 ret := ret ∪ {S[i-z+1..z]}

return ret

Dinamik Programlama kodu

Minimum Edit Distance

- Is the minimum number of editing operations needed to transform one into the other
 - Insertion
 - Deletion
 - Substitution
- Many applications in string comparison/alignment, e.g., spell checking, machine translation, bioinformatics, etc.

Minimum Edit Distance

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- If each operation has cost of 1
 - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
 - Distance between them is 8

Minimum Edit Distance

One possible path

i n t e n t i o n	← <i>delete i</i>
n t e n t i o n	← <i>substitute n by e</i>
e t e n t i o n	← <i>substitute t by x</i>
e x e n t i o n	← <i>insert u</i>
e x e n u t i o n	← <i>substitute n by c</i>
e x e c u t i o n	

Defining Min Edit Distance

- For two strings S_1 of len n , S_2 of len m
 - distance(i, j) or $D(i, j)$
 - means the edit distance of $S_1[1..i]$ and $S_2[1..j]$
 - i.e., the minimum number of edit operations need to transform the first i characters of S_1 into the first j characters of S_2
 - The edit distance of S_1, S_2 is $D(n, m)$
- We compute $D(n, m)$ by computing $D(i, j)$ for all i ($0 \leq i \leq n$) and j ($0 \leq j \leq m$)
- Note the index associated with the source/target string: first is source and second is the target

Defining Min Edit Distance

- Base conditions:

- $D(i, 0) = i$ */* deletion cost*/*

- $D(0, j) = j$ */* insertion cost*/*

- Recurrence Relation:

- $$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{/* cost for deletion*/} \\ D(i, j-1) + 1 & \text{/* cost for insertion*/} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} & \text{/* cost for substitution */} \end{cases}$$

Dynamic Programming

- A tabular computation of $D(n,m)$
- Bottom-up
 - Compute $D(i,j)$ for smaller i,j
 - Increase i, j to compute $D(i,j)$ using previously computed values based on smaller indexes.

The Edit Distance Table

source	N	9									
	O	8									
	I	7									
	T	6									
	N	5									
	E	4									
	T	3									
	N	2									
	I	1									
	#	0	1	2	3	4	5	6	7	8	9
		#	E	X	E	C	U	T	I	O	N

target

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$



Adding Backtrace to MinEdit

- Base conditions:

- $D(i,0) = i$
- $D(0,j) = j$

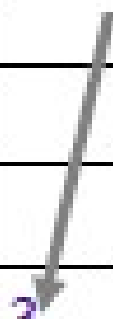
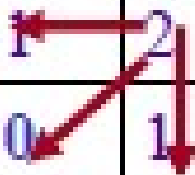
- Recurrence Relation:

$$- D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{DOWN} \\ D(i,j-1) + 1 & \text{LEFT} \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} & \text{DIAGONAL} \end{cases}$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
insertion										
I	1	2								
#	0	1	deletion		4	5	6	7	8	9
substitution	E	X	E	C	U	T	I	O	N	

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3							
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$



N	9											
O	8											
I	7											
T	6											
N	5											
E	4											
T	3											
N	2											
I	1	←	→	←	→							
#	0	↙	↘	↖	↗	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N		

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

MinEdit with Backtrace

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↘ 8	
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↘ 8	← 9	
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↘ 8	← 9	← 10	
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↘ 8	← 9	← 10	← 11	
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10	
e	4	↘ 3	← 4	↘ 5	← 6	← 7	← 8	↙←↓ 9	↙←↓ 10	↓ 9	
t	3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↘ 7	← 8	↙←↓ 9	↓ 8	
n	2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↘ 7	
i	1	↙←↓ 2	↙←↓ 3	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↘ 6	← 7	← 8	
#	0	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

MinEdit with Backtrace

n	9	↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙←↓ 12	↓ 11	↓ 10	↓ 9	↘ 8	
o	8	↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↓ 10	↓ 9	↘ 8	← 9	
i	7	↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	↘ 8	← 9	← 10	
t	6	↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↘ 8	← 9	← 10	←↓ 11	
n	5	↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↙←↓ 9	↙←↓ 10	↙←↓ 11	↙↓ 10	
e	4	↘ 3	← 4	↙← 5	← 6	← 7	←↓ 8	↙←↓ 9	↙←↓ 10	↓ 9	
t	3	↙←↓ 4	↙←↓ 5	↙←↓ 6	ins	↙←↓ 8	↘ 7	←↓ 8	↙←↓ 9	↓ 8	
n	2	↙←↓ 3	sub	↙←↓ 5	↙←↓ 6	↙←↓ 7	↙←↓ 8	↓ 7	↙←↓ 8	↘ 7	
i	1	sub	↙←↓ 4	↙←↓ 5	↙←↓ 6	↙←↓ 7	↘ 6	← 7	← 8		
# del	0		2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

Performance

- Time: $O(nm)$
- Space: $O(nm)$
- Backtrace: $O(n+m)$