

Creative Commons



- slobodno smijete:

- dijeliti — umnožavati, distribuirati i javnosti priopćavati djelo
- prerađivati djelo



- pod sljedećim uvjetima:

- imenovanje: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- nekomercijalno: ovo djelo ne smijete koristiti u komercijalne svrhe.
- dijeli pod istim uvjetima: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

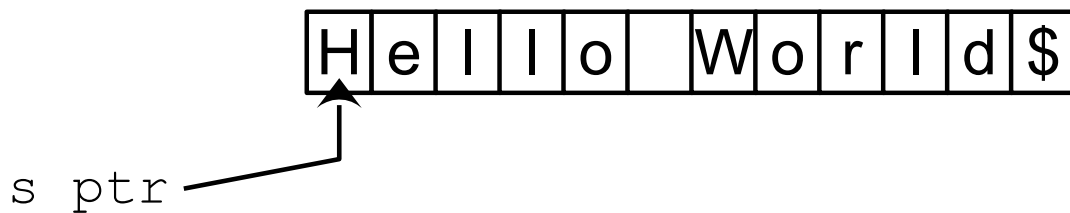
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>

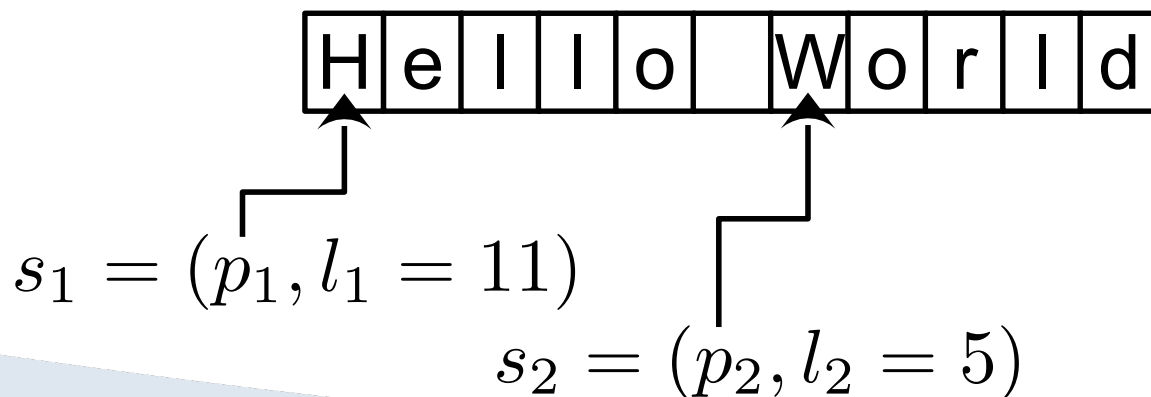
Reprezentacija znakovnog niza (1)

- Znakovni niz je okarakteriziran svojim alfabetom Σ – definira sve znakove koji se u nizu mogu naći
 - Imamo mapiranje $\mu: \Sigma \rightarrow \mathbb{N}$, kojim se znakovima alfabeta pridodaje cijeli broj
 - Tako definiramo znakovne tablice, poput ASCII (1 bajt – 8 bitova) ili Unicode (2 bajta – 16 bitova)
- Osnovna reprezentacija – korištenjem ASCII tablice znakovni niz definiramo kao slijed bajtova koji je završen s NULL terminatorom (vrijednost 0) – a koji u predavanju označavamo kao \$



Reprezentacija znakovnog niza (2)

- Reprezentacija uređenim parom (p, l) , gdje je
 - p – pokazivač na prvi znak znakovnog niza
 - l – duljina niza
- Ovakva reprezentacija je zanimljiva kada na jednostavan način trebamo izolirati znakovni podniz bez stvaranja nove instance
 - Dobro rješenje kada imamo veći tekst u memoriji, pa u njemu treba mapirati pojedine dijelove
 - Za ovakvu reprezentaciju nije nužno korištenje NULL terminatora



Prefiksno stablo (Trie) (1)

- Zamislimo veći tekst u memoriji računala, te skup ključnih riječi (ili izraza) od interesa u tom tekstu
- Skup ključnih znakovnih nizova (riječi, izrazi, dijelovi teksta) definiramo kao

$$S = \{s_i : 0 < i \leq N\}$$

- gdje je N broj znakovnih nizova
- Želimo znati da li je znakovni niz q sadržan u tekstu
 - To možemo provjeriti tako da testiramo hipotezu $q \in S$
 - Naivna implementacija bi prolazila kroz sve znakove znakovnih nizova u skupu S , što rezultira s

$$O\left(\sum_{s_i \in S} |s_i|\right)$$

Prefiksno stablo (Trie) (2)

- Trie je uređena trojka $T = (N, E, \mu)$ koja predstavlja m-stablo
 - Funkcija mapiranja $\mu: E \rightarrow \Sigma$ mapira bridove stabla na alfabet
 - Koriijenski čvor predstavlja prazni znakovni niz
 - Svaki čvor Trie-a može imati najviše $|\Sigma|$ djece
 - Svaki izlazni brid čvora mora se mapirati na drugi znak alfabeta – dva različita izlazna brida jednog čvora ne mogu biti mapirani na isti znak alfabeta
 - Listovi su terminirajući čvorovi čiji su ulazni bridovi mapirani na \$ (NULL terminator)
 - Trie je zapravo konačni deterministički automat koji omogućava parsiranje znakovnog niza

Prefiksno stablo (Trie) (3)

- Primjer

$$S_1 = \{"bird\$", "cat\$", "fish\$", "zebra\$"\}$$

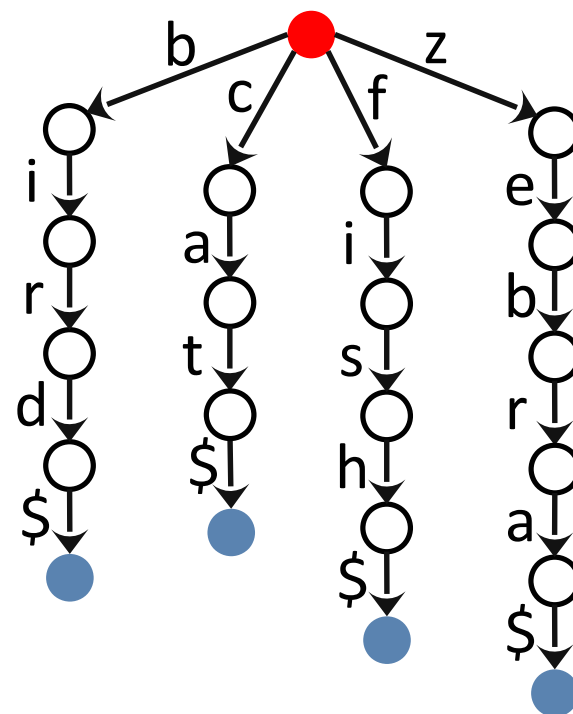
$$\Sigma = \{ \$, a, b, c, d, e, f, h, i, r, s, t, z \}$$

- Problem u ovakvom stablu je pretraživanje izlaznih bridova

- Kako ustanoviti da li imamo izlazni brid koji je istovjetan našem sljedećem znaku u nizu q
- Naivna implementacija bi podrazumijevala iteraciju po svim izlaznim bridovima

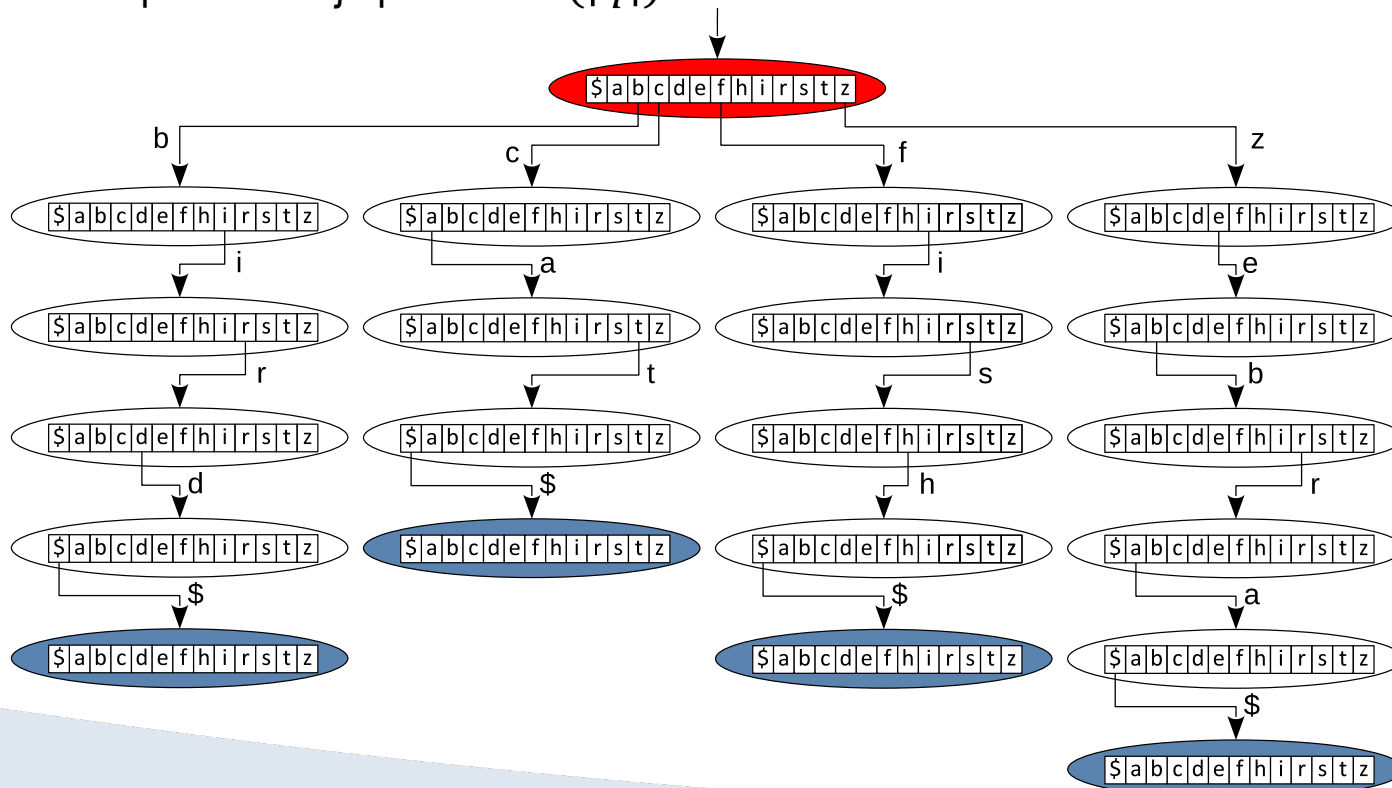
$$O(|q| * |\Sigma|)$$

- Postoji li bolji način od toga?



Prefiksno stablo (Trie) (4)

- U svaki čvor stavimo mapirajuće polje alfabeta s pokazivačima na djecu
 - Iz mapiranja $\mu: \Sigma \rightarrow \mathbb{N}$ znamo koji element polja gledamo
 - Ako u tom elementu ima pokazivač, tada ta tranzicija postoji
 - Prostorna kompleksnost se povećava na $O(\sum_{s_i \in S} |s_i| * |\Sigma|)$,
 - No složenost pretraživanja pada na $O(|q|)$



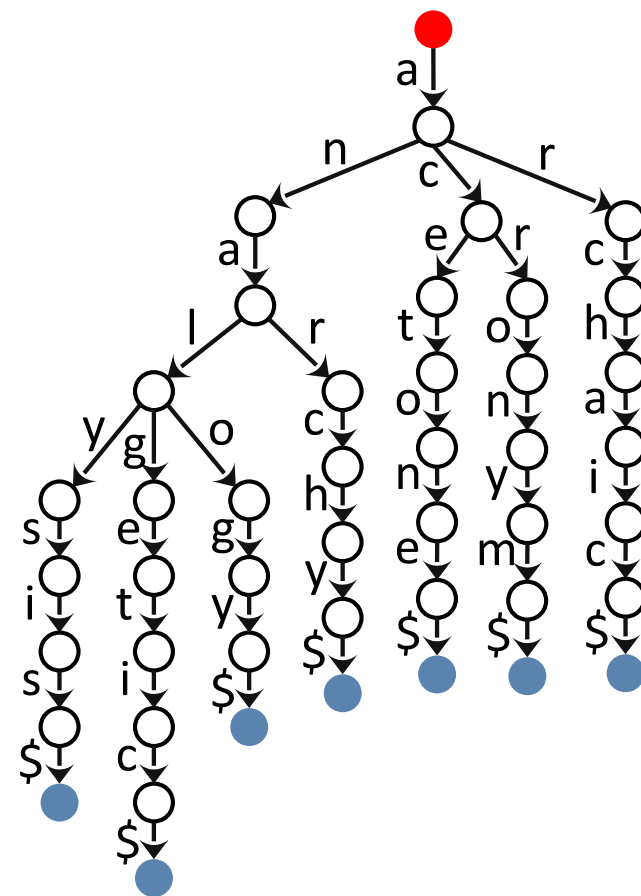
Prefiksno stablo (Trie) (5)

- Mnogo znakovnih nizova u S može dijeliti zajednički prefiks

$S_2 = \{"analysis\$","analgetic\$","analogy\$","anarchy\$","acetone\$","acronym\$","archaic\$"\}$

a	n	a	l	y	s	i	s	
a	n	a	l	g	e	t	i	c
a	n	a	l	o	g	y		
a	n	a	r	c	h	y		
a	c	e	t	o	n	e		
a	c	r	o	n	y	m		
a	r	c	h	a	i	c		

- Sama definicija Trie-a, koja ograničava da više se izlaznih bridova jednog čvora mapira na isti znak alfabeta, nas prisiljava da zajednički prefiksi znakovnih nizova dijele strukturu stabla



Prefiksno stablo (Trie) (6)

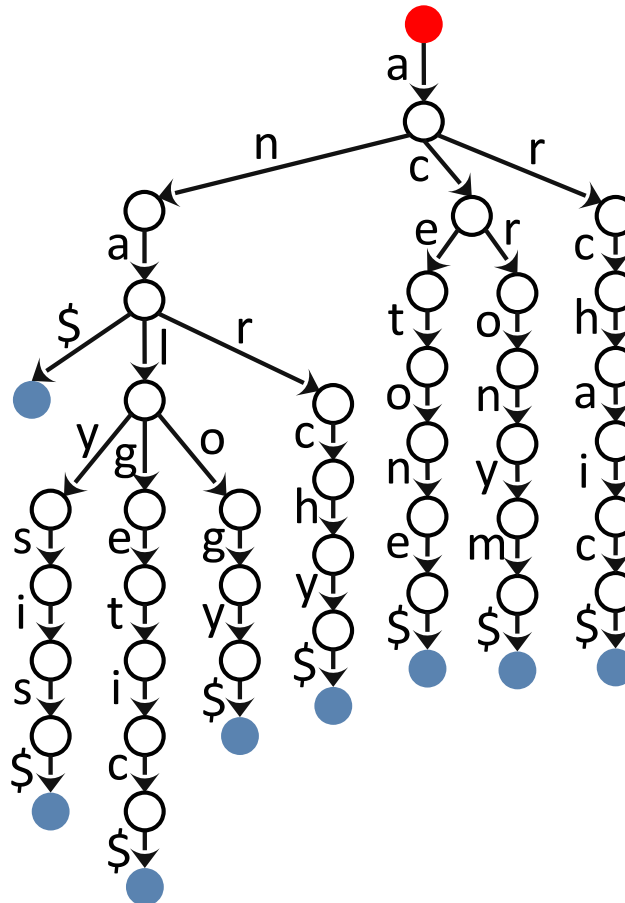
- Pretraživanje

- Uzimamo znak po znak iz znakovnog niza q i slijedimo tranzicije u Trie-u T
- Ako smo nakon \$ u q došli do lista, tada S (i Trie T) sadrži q
 - Iskoristili smo sve znakove u q , što nas je dovelo do lista Trie-a T
- U svakom drugom slučaju q nije pronađen u S
 - Što se desi kada tražimo $q = \text{"ana\$"} u prethodnom Trie-u ?$

```
function SEARCHTRIE( $T, q$ )  
   $cn \leftarrow \text{root}(T)$   
  for  $ch \in q$  do  
    if there is transition from  $cn$  for character  $ch$  to  $cn_{child}$  then  
       $\triangleright \text{array}[ch] \neq \text{NULL in } cn$   
       $cn \leftarrow cn_{child}$   
    else  
      return ( $false, cn$ )  
  if  $cn$  is a leaf then  
    return ( $true, cn$ )  
  return ( $false, cn$ )
```

Prefiksno stablo (Trie) (7)

- Primjer $S_3 = S_2 \cup \{ "ana$" \}$



Prefiksno stablo (Trie) (8)

- Upis novog znakovnog niza s (*insert*)
 - Krenemo s pretraživanjem $q = s$
 - Ako pronađemo cijeli q u Trie-u T , tada je $s \in S$ i upis nije potreban
 - Inače stanemo na prvom znaku za kojeg nemamo tranziciju u Trie-u – to može biti i \$
 - Stvorimo slijednu strukturu Trie-a za ostatak znakovnog niza s
- Primjer: u S_1 dodamo $s = „catfish$”$

```
procedure INSERTTRIE( $T, s$ )
```

```
   $cn \leftarrow \text{root}(T)$ 
```

```
  for  $ch \in s$  do
```

```
    if there is transition from  $cn$  for character  $ch$  to  $cn_{child}$  then  
       $\triangleright \text{array}[ch] \neq \text{NULL}$  in  $cn$ 
```

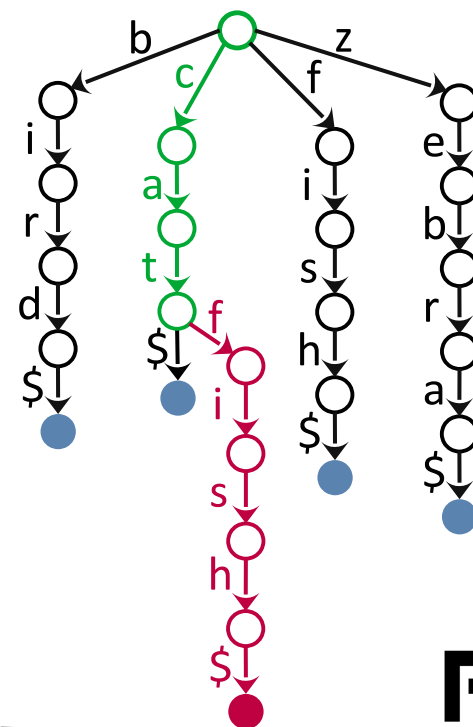
```
       $cn \leftarrow cn_{child}$ 
```

```
    else
```

```
      create new node  $cn_{new}$ 
```

```
      add transition for character  $ch$  from  $cn$  to  $cn_{new}$ 
```

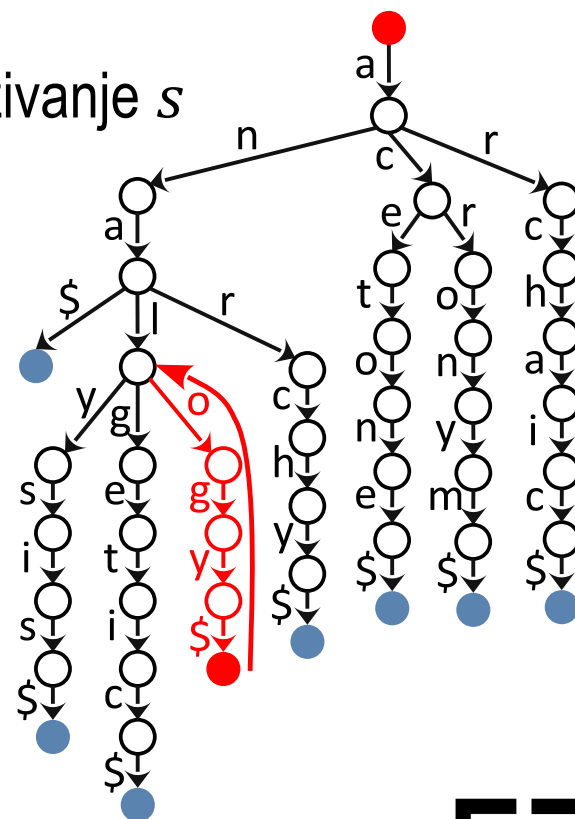
```
       $cn \leftarrow cn_{new}$ 
```



Prefiksno stablo (Trie) (9)

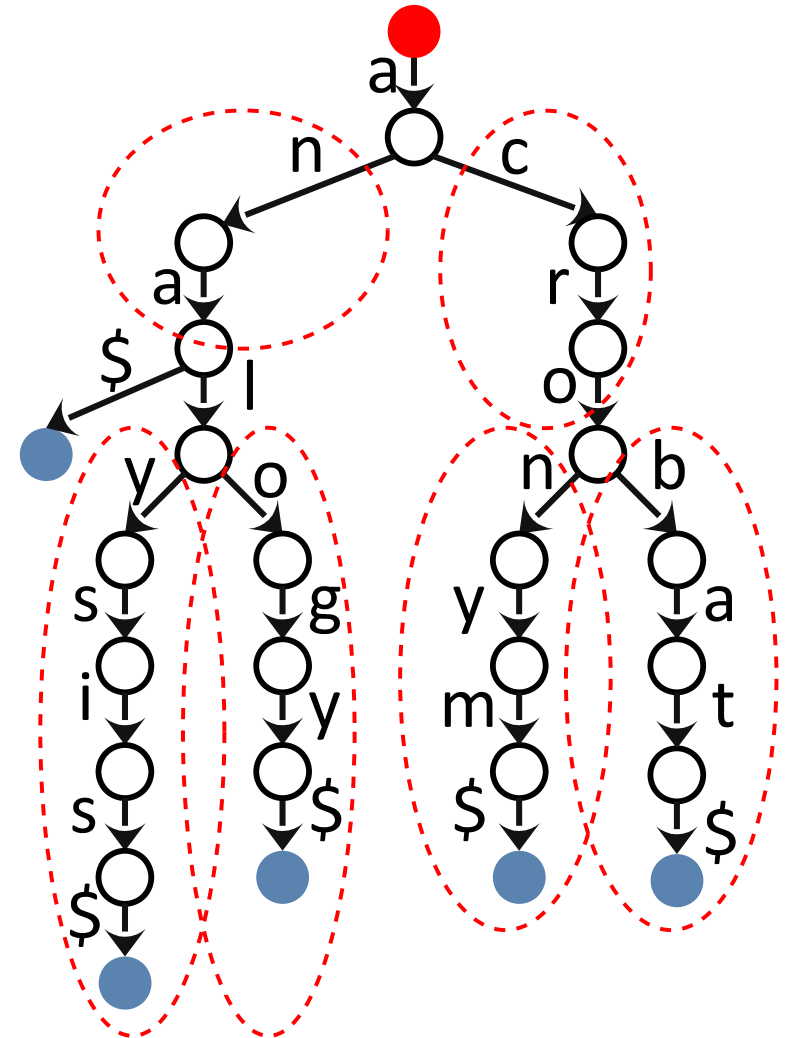
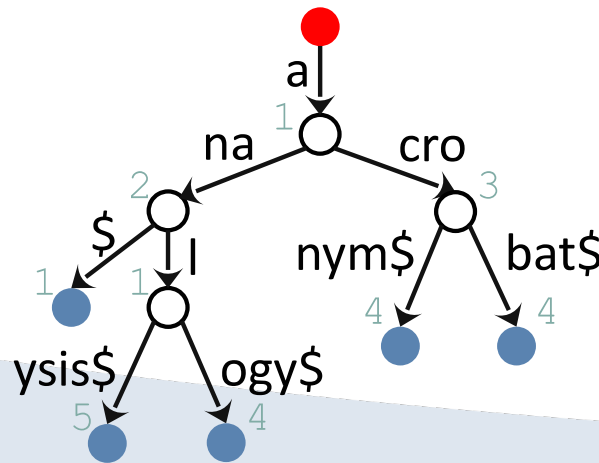
- Brisanje znakovnog niza s (*remove*)
 - Krenemo s pretraživanjem $q = s$
 - Ako pronađemo cijeli q u Trie-u T , tada je $s \in S$ i brisanje je moguće
 - Inače stanemo s brisanjem
 - Vraćamo se od lista do kojeg nas je dovelo pretraživanje s
 - Tako dugo do dok roditeljski čvor ima samo jedno dijete
- Primjer: u S_3 brišemo $s = „analogy$“$

```
procedure REMOVETRIE( $T, s$ )  
  ( $succ, cn$ )  $\leftarrow$  SEARCHTRIE( $T, s$ )  
  if not succ then  
    return ▷  $s$  not found in Trie  
   $s \leftarrow$  'reversed( $s$ )'  
   $cn \leftarrow$  parent( $cn$ )  
  for  $ch \in s$  do  
    remove transition from  $cn$  for character  $ch$   
    if  $cn$  has no children and there is a parent of  $cn$  then  
       $cn \leftarrow$  parent( $cn$ )  
  else  
    return
```



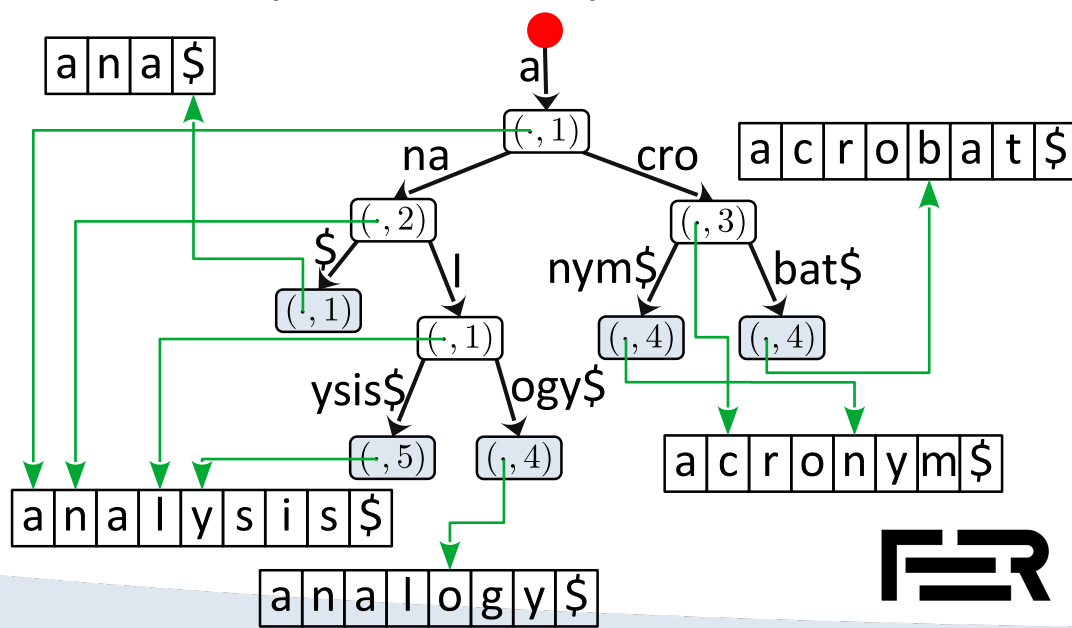
Patricia stablo (1)

- Struktura Trie-a nije kompaktna
 - Sjetite se svih polja u čvorovima u kojima imamo samo jedan jedini pokazivač
 - Prostorna kompleksnost Trie-a se može smanjiti tako da slijed čvorova koji imaju samo jednu tranziciju pretvorimo u jednu tranziciju
- Dobivamo kompresirani Trie ili Patricia stablo (Practical Algorithm To Retrieve Information Coded In Alphanumeric)



Patricia stablo (2)

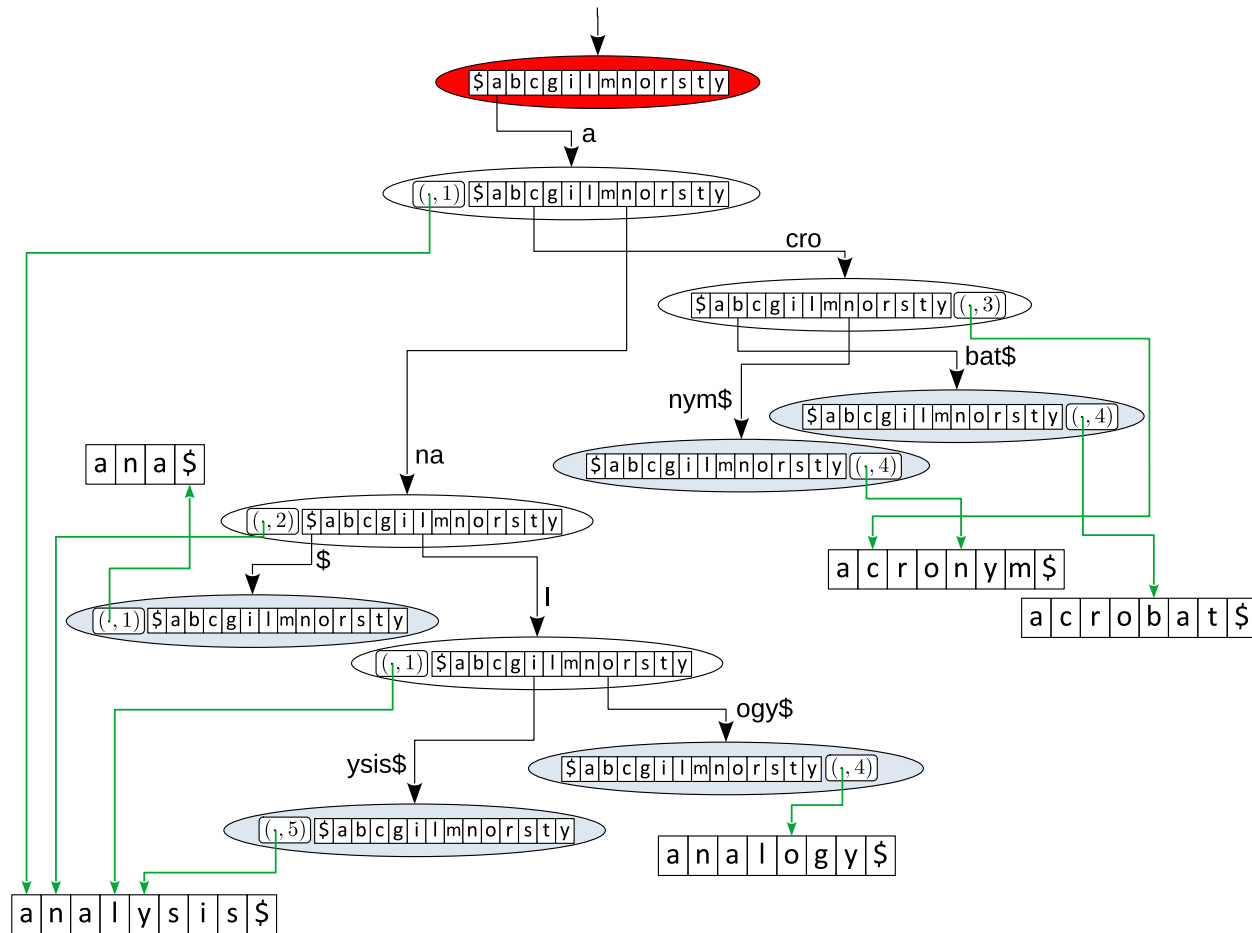
- Neke bitne stavke Patricia stabla
 - Svi unutarnji čvorovi Patricia stabla imaju barem dva djeteta – inače se vraćamo na Trie strukturu
 - Reprezentacija znakovnog niza temeljena na uređenom paru (p, l) pokazuje se boljom u Patricia stablima – manja potrošnja memorije
 - U memoriji imamo skup znakovnih nizova S
 - Čvorovi pokazuju samo na određene znakovne podnizove u skup S
- Sjetimo se da u svaki čvor Patricia stabla imamo samo jedan ulazni brid
 - U čvoru možemo čuvati uređeni par za tu ulaznu tranziciju



Patricia stablo (3)

- Pokazivači u čvorovima i dalje funkcioniraju kao i kod Trie-a uz malu nadopunu

- Tranzicije u Patricia stablu predstavljaju znakovni podniz
- Pokazivač u čvoru odgovara prvom znaku znakovnog podniza tranzicije
- Nije moguće da dvije tranzicije, dva izlazna brida iz čvora, imaju znakovni podniz koji počinje s istim znakom
 - Ne mogu dijeliti isti prefiks jer bi to značilo da je taj prefiks dio prethodne tranzicije



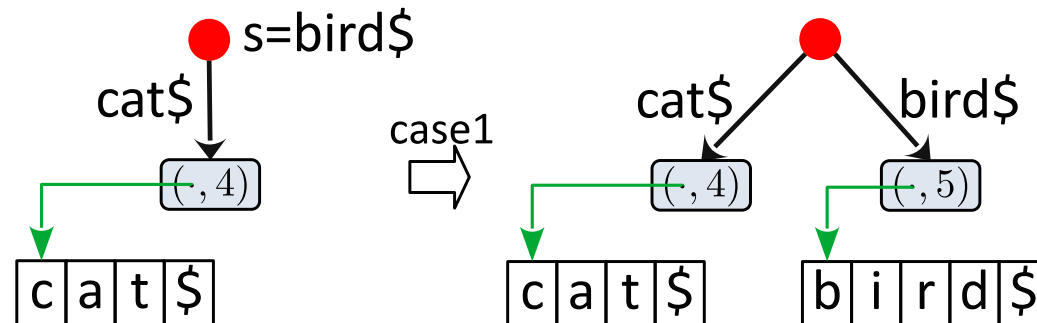
Patricia stablo (4)

- Pretraživanje zahtijeva usporedbu znakovnog podniza sadržanog u tranziciji sa adekvatnim znakovnim podnizom od q
 - Tijekom pretraživanja trebamo pamtit na kojem smo mjestu u znakovnom nizu q (varijabla sc)

```
function SEARCHPATRICIA( $P, q = (q_p, q_l)$ )  
   $sc \leftarrow 0$   
   $cn \leftarrow \text{root}(P)$   
  while  $cn$  not leaf do  
    if there is transition from  $cn$  for character  $q_p[sc]$  to  $cn_c$  then  
       $\triangleright \text{array}[q_p[sc]] \neq \text{NULL}$  in  $cn$   
       $(t_p, t_l) \leftarrow$  string representation in  $cn_c$   
      if  $q_p[sc : sc + t_l] = t_p[0 : t_l]$  then  
         $\triangleright$  substring matching (Python notation)  
         $sc \leftarrow sc + t_l$   
         $cn \leftarrow cn_c$   
      else  
        return false  
    else  
      return false  
  return  $sc = q_l$ 
```

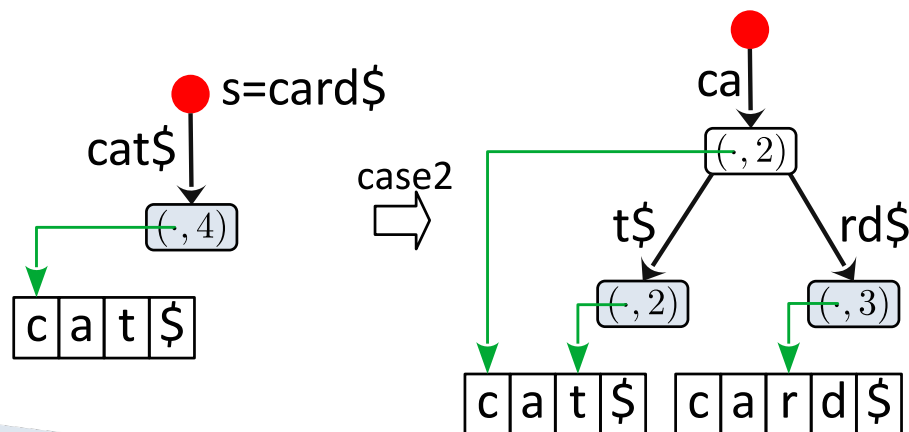
Patricia stablo (5)

- Upis novog znakovnog niza s je nešto kompliciraniji nego u Trie-u
 - Započinjemo s pretraživanjem
 - Gledamo s aspekta trenutnog čvora – počinjemo iz korijenskog čvora
- 1. Nije moguće naći izlaznu tranziciju iz čvora koja bi barem djelomično podudarala ostatku znakovnog niza s
 - Dodamo novi list u strukturu i tranziciju koja je identična ostatku znakovnog niza s



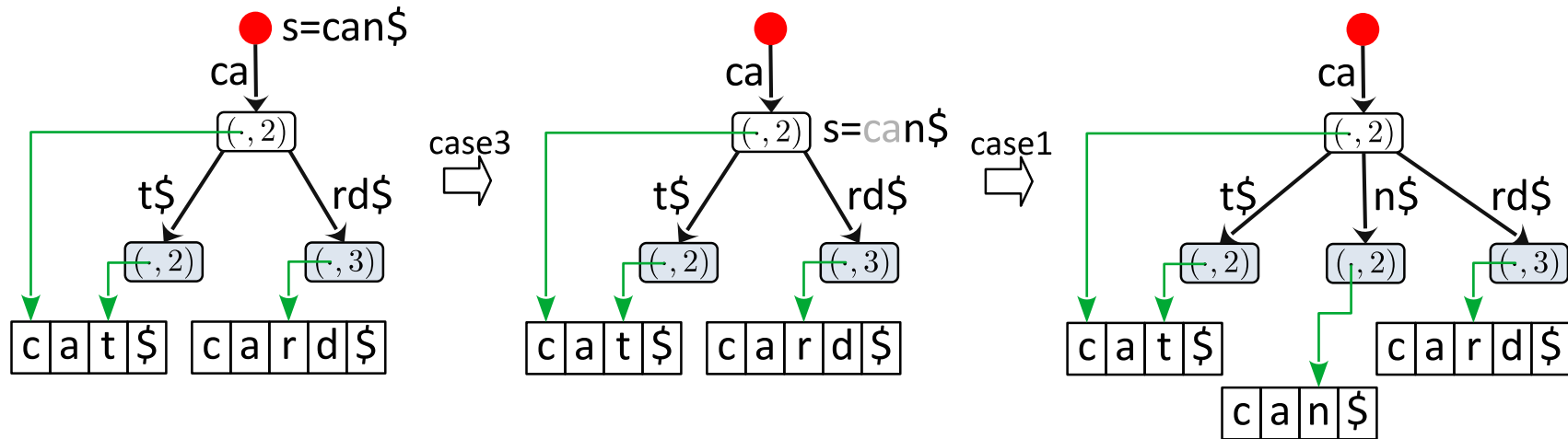
Patricia stablo (6)

2. Moguće je naći izlaznu tranziciju iz čvora koja djelomično podudara dijelu ostatka znakovnog niza s
- Dodamo novi čvor cn_{ins} kojim razdvajamo staru tranziciju na dva dijela, prvi dio koji se podudara s ostatkom znakovnog niza s i drugog dijela koji se ne podudara
 - Dodamo novi list cn_{leaf} i tranziciju između cn_{ins} i cn_{leaf} koja odgovara ostatku znakovnog niza s , a koji se nije podudarao s prvim dijelom razdvojene tranzicije u prethodnom koraku



Patricia stablo (7)

3. Moguće je naći izlaznu tranziciju iz čvora koja potpuno podudara dijelu ostatka znakovnog niza s
- Prolazimo tranziciju i pozicioniramo se u novom čvoru
 - Pazimo na ostatak znakovnog niza s



- Evolucija Patricia stabla – slijed upisa i brisanja znakovnih nizova u S : svaki korak se može vizualizirati

Patricia stablo (8)

- Kompleksnost dodavanja znakovnog niza je $O(|s| + |\Sigma|)$

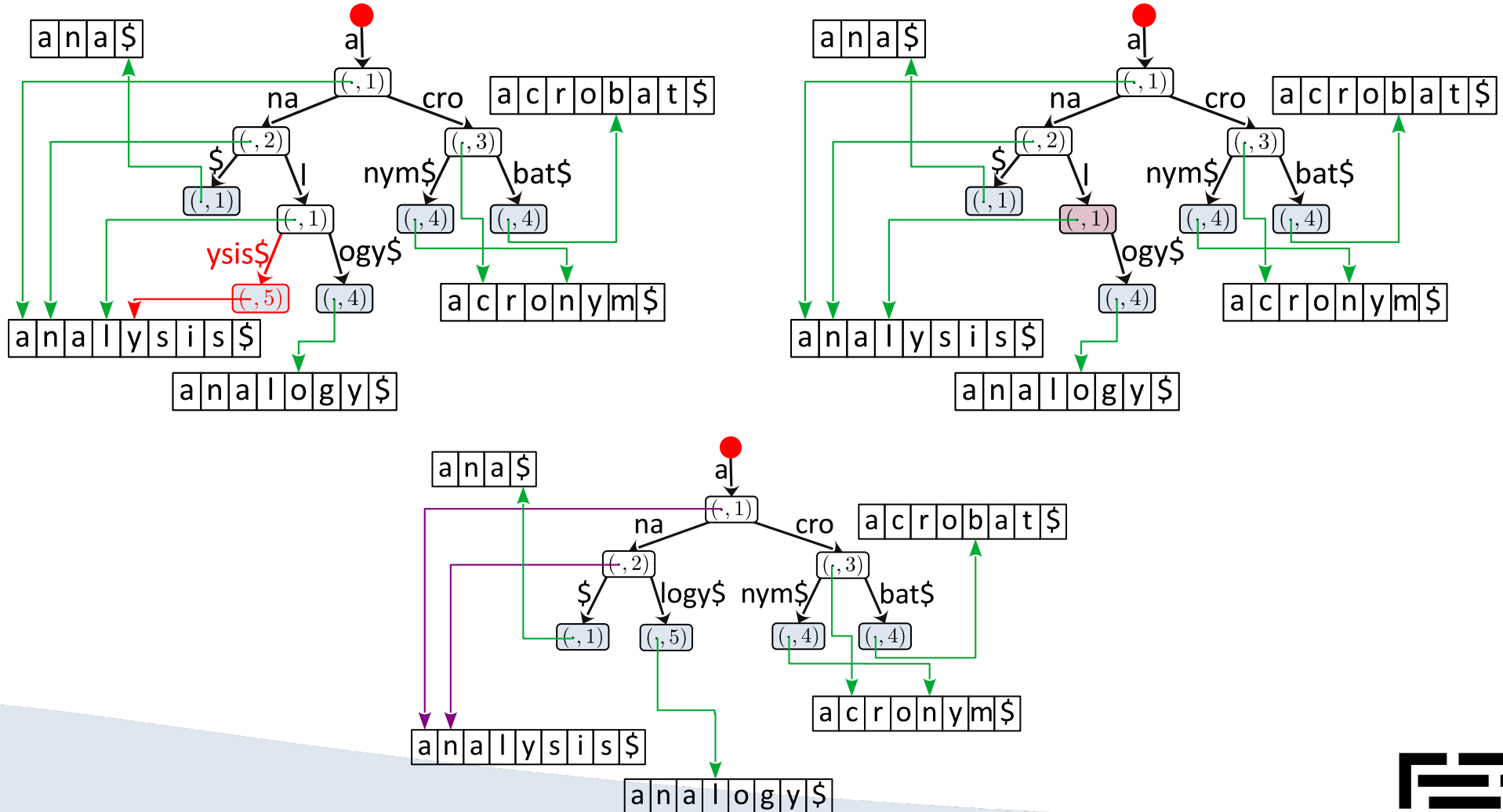
```
procedure INSERTPATRICIA( $P, s = (s_p, s_l)$ )  
   $sc \leftarrow 0$   
   $cn \leftarrow \text{root}(P)$   
  while  $cn$  is root or not leaf do  $\triangleright O(|s|)$   
    if there is transition from  $cn$  for character  $s_p[sc]$  to  $cn_c$  then  
       $\triangleright \text{array}[s_p[sc]] \neq \text{NULL}$  in  $cn$   
       $(t_p, t_l) \leftarrow$  string representation in  $cn_c$   
      if  $s_p[sc : sc + t_l] \subset t_p[0 : t_l]$  then  $\triangleright$  case 2  
         $\triangleright$  we know for sure that the first character is matched  
         $i \leftarrow$  first unmatched character from  $t_p$   $\triangleright i > 0$   
        remove transition between  $cn$  and  $cn_c$   
        add node  $cn_{ins}$  with  $(t_p, i)$   $\triangleright O(|\Sigma|)$   
        update node  $cn_c$  with  $(t_p + i, t_l - i)$   
        add transition between  $cn$  and  $cn_{ins}$   
        add transition between  $cn_{ins}$  and  $cn_c$   
        add leaf node  $cn_{leaf}$  with  $(s_p + (sc + i), s_l - (sc + i))$   
         $\triangleright O(|\Sigma|)$   
        add transition between  $cn_{ins}$  and  $cn_{leaf}$   
        return  
      else if  $s_p[sc : sc + t_l] = t_p[0 : t_l]$  then  $\triangleright$  case 3  
         $sc \leftarrow sc + t_l$   
         $cn \leftarrow cn_c$   
      else  $\triangleright$  case 1  
        add leaf node  $cn_{leaf}$  with  $(s_p + sc, s_l - sc)$   $\triangleright O(|\Sigma|)$   
        add transition between  $cn$  and  $cn_{leaf}$   
        return
```

Patricia stablo (9)

- Brisanje znakovnog niza s iz skupa S
 - Započinje pretraživanjem – s mora postojati u Patricia stablu
 - Krećemo od lista koji predstavlja s , kao i u Trie-u
 - Uklanjammo list i njegovu ulaznu tranziciju
 - Ako roditelj uklonjenog lista ostane s jednim djetetom, tada se taj roditelj treba ukloniti i njegove tranzicije spojiti – nekompresirani slučaj
 - Spajanje tranzicija za znakovne nizove (p, l) je jednostavno
 - Pokazivač p na početak prebacimo za nekoliko znakova prema početku znakovnog niza
 - Duljinu l povećamo za taj broj znakova

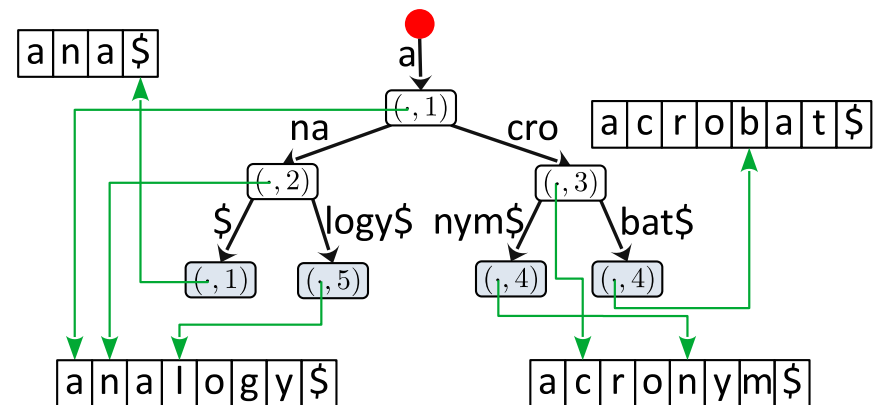
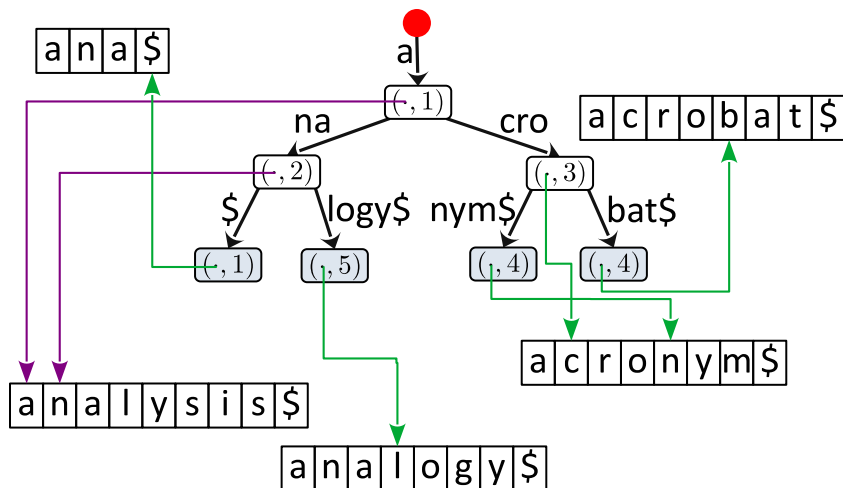
Patricia stablo (10)

- Primjer: uklanjamo $s = „analysis$“$



Patricia stablo (11)

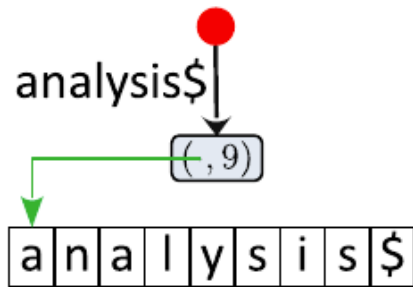
- Znakovni niz "*analysis*\$" trebamo ukloniti iz memorije (destruktor u C-u)
 - No, imamo još dva pokazivača na tu instancu
 - Pokazivače prebacimo na znakovni niz jedno od preostale djece



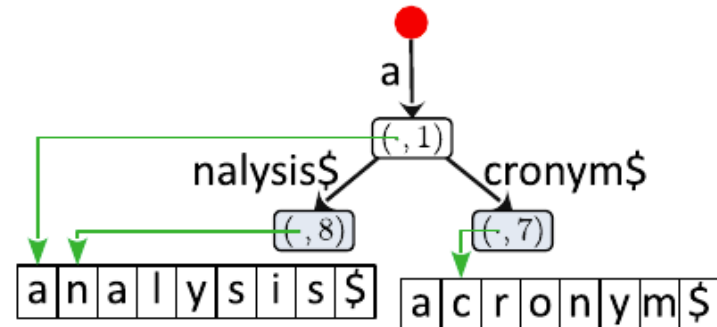
Patricia stablo (12)

- Primjer
 - Nacrtajte evoluciju Patricia stabla za upis sljedećeg slijeda znakovnih nizova
 $\langle \text{"analysis\$"}, \text{"acronym\$"}, \text{"analogy\$"}, \text{"acrobat\$"}, \text{"ana\$"} \rangle$

Step 1: Insert "analysis\$"

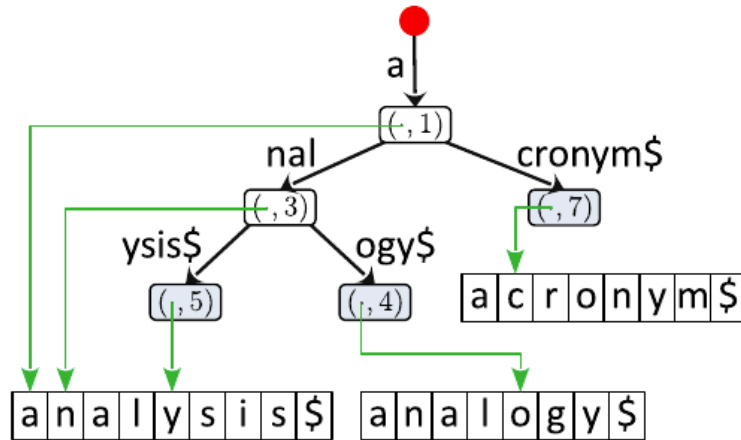


Step 2: Insert "acronym\$"

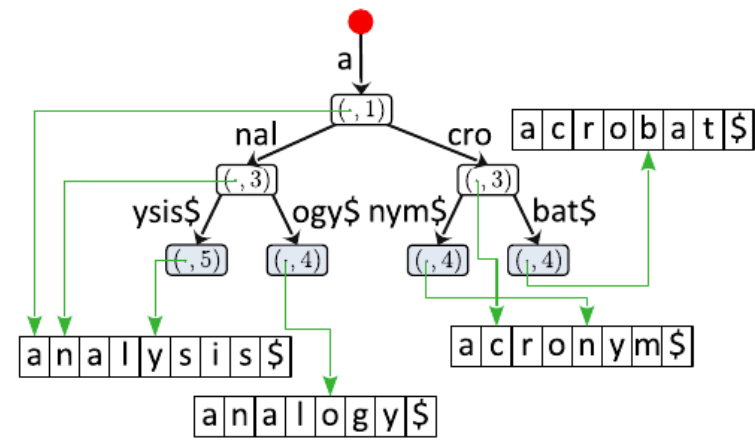


Patricia stablo (13)

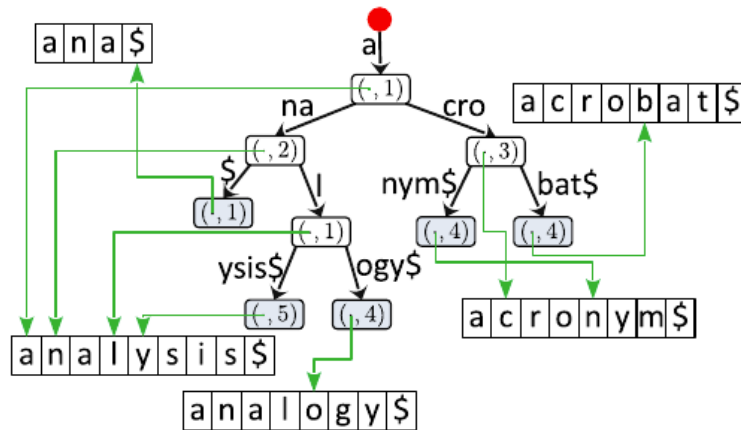
Step 3: Insert "analogy\$"



Step 4: Insert "acrobat\$"



Step 5: Insert "ana\$"



Patricia stablo (14)

- Neke značajne primjene: IP routing (u routerima), IP filtering, firewall, IP lookup, ...
- Ako za alfabet uzmemo $\Sigma = \{0,1\}$
- IPv4 adrese možemo transformirati u znakovni niz kao

192.168.11.45/32

1100 0000 . 1010 1000 . 0000 1011 . 0010 1101

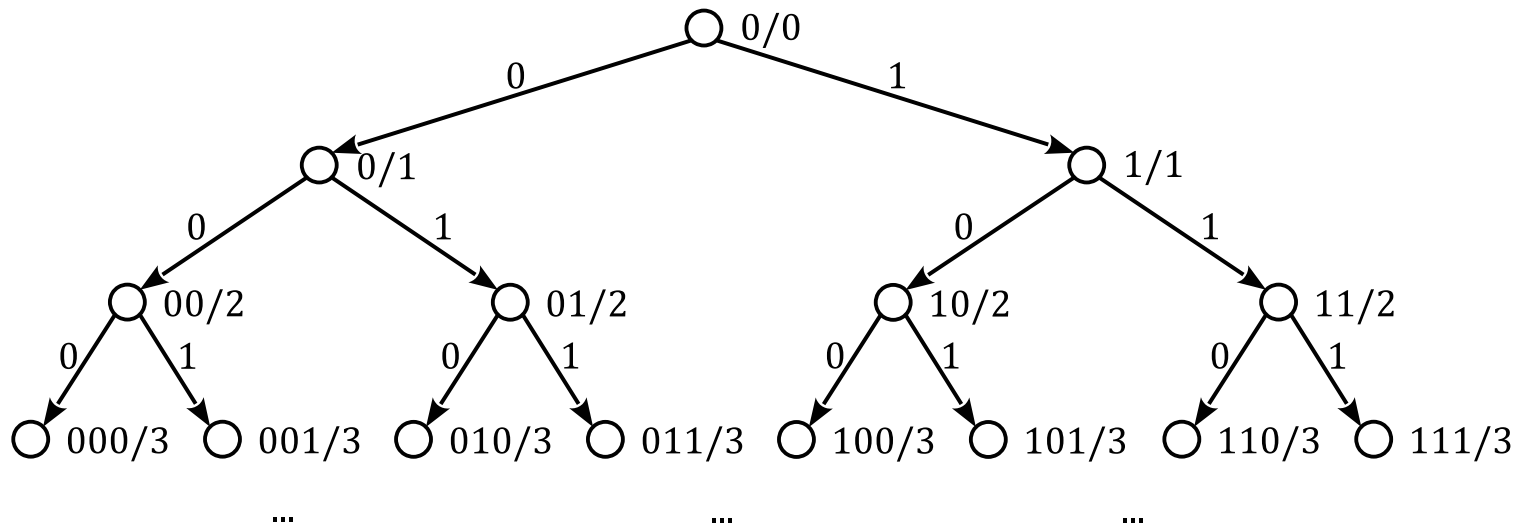
- Ili masku kao

192.168.0.0/16

1100 0000 . 1010 1000 . * . *

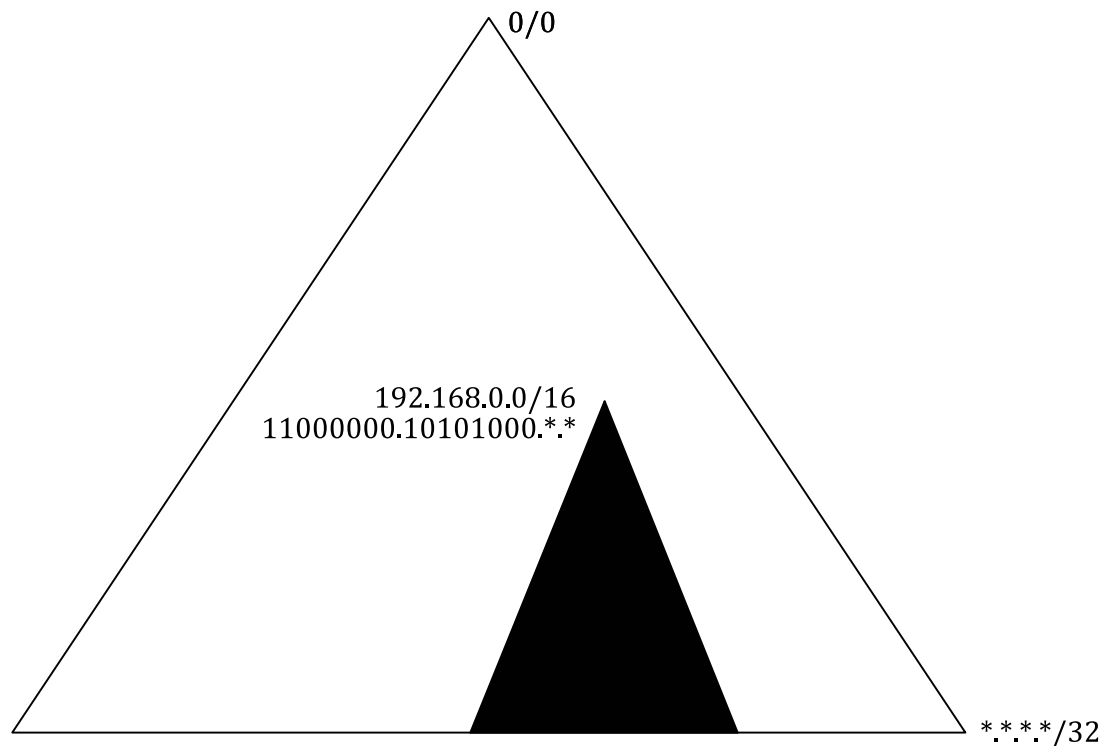
Radix stabla (1)

- Radix stablo za IP routing gradimo na sljedeći način – binarno stablo



Radix stabla (2)

- U takvom Radix stablu, podmreža se detektira kroz svoju masku
- Svi čvorovi podstabla predstavljaju članove podmreže 192.168.0.0/16

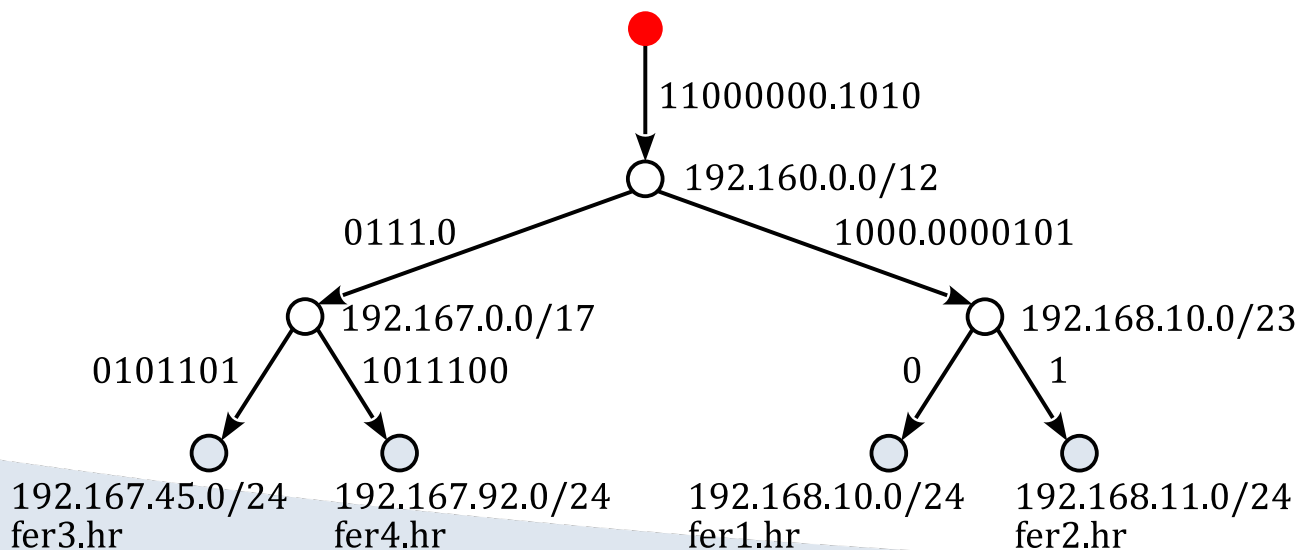


IP routing

- Kompresiramo Radix stablo – dobivamo binarno Patricia stablo
- Sljedeća IP routing tablica

192.168.10.0/24	11000000.10101000.00001010	fer1.hr
192.168.11.0/24	11000000.10101000.00001011	fer2.hr
192.167.45.0/24	11000000.10100111.00101101	fer3.hr
192.167.92.0/24	11000000.10100111.01011100	fer4.hr

- pretvara se u Patricia stablo
- listovi mogu imati pokazivače na definiciju sučelja (ili na nešto drugo)



Sufiksna polja

Sufiksna stabla

Sufiks znakovnog niza

- Zamislimo znakovni niz $t = „program\$”$
 - gdje je $t = t[1], t[2], \dots, t[j], \dots, t[n]$, a $t[j]$ je j -ti znak znakovnog niza
- Iz znakovnog niza možemo izvesti skup svih njegovih sufiksa, gdje je i -ti sufiks znakovnog niza t , definiran kao

$$t_i = t[i], t[i + 1], \dots, t[n]$$

1	2	3	4	5	6	7	8
p	r	o	g	r	a	m	\$

1	p	r	o	g	r	a	m	\$
2	r	o	g	r	a	m	\$	
3	o	g	r	a	m	\$		
4	g	r	a	m	\$			
5	r	a	m	\$				
6	a	m	\$					
7	m	\$						
8	\$							

Svi sufiksi znakovnog niza

- Reprezentacija znakovnog niza s uređenim parom daje nam prednost kod stvaranja liste svih sufiksa znakovnog niza
- Naivno: kada bismo htjeli stvoriti novu instancu znakovnog niza za i -ti sufiks $t[i]$, trebali bismo kopirati znakove od i do n u tu novu instancu.

- Za sve sufikse znakovnog niza to znači sljedeći broj kopiranja

$$(n - 1) + (n - 2) + \dots + 1 = \frac{n(n + 1)}{2} - n$$

- Kada koristimo uređeni par (p, l) kao reprezentaciju znakovnog niza, u n iteracija radimo
 - Pomičemo pokazivač p za jedno mjesto unaprijed
 - Smanjimo l za jedan

Sufiksno polje (1)

- Prethodnu listu sufiksa sortiramo alfabetski (abecedno) i dobivamo sljedeću sortiranu listu sufiksa (sjetimo se $\$=0$)

i	A_i	t_{A_i}							
1	8	\$							
2	6	a	m	\$					
3	4	g	r	a	m	\$			
4	7	m	\$						
5	3	o	g	r	a	m	\$		
6	1	p	r	o	g	r	a	m	\$
7	5	r	a	m	\$				
8	2	r	o	g	r	a	m	\$	

- gdje je $A = \langle A_1, A_2, \dots, A_i, \dots, A_n \rangle$ sortirani niz sufiksa koji definira sufiksno polje

$SA = \langle \$, am$, gram$, m$, ogram$, program$, ram$, rogram$ \rangle$

Sufiksno polje (2)

- Kompleksnost stvaranja sufiksnog polja je dosta visoka
- Koristimo algoritam za sortiranje što je $O(n \log n)$, pa kad tome dodamo usporedbu znakovnih nizova, to se može aproksimirati kao $O(n^2 \log n)$
- Ideja sufiksnog polja je provjeriti da li se znakovni niz q nalazi u znakovnom nizu t
 - Naivno pretraživanje bi bilo $O(|q| * n)$
 - Pretraživanje sufiksnog polja sa binarnim algoritmom za pretraživanje je $O(|q| \log n)$
 - Binarni algoritam za pretraživanje je sličan pretraživanju binarnog stabla
 - Pronađemo sredinu, pa provjerimo da li je q manji ili veći od te sredine
 - S obzirom na rezultat, pretraživanje svodimo na pola sufiksnog polja
 - Nastavljamo rekurzivno do konačnog rezultata: *false* ili *true*

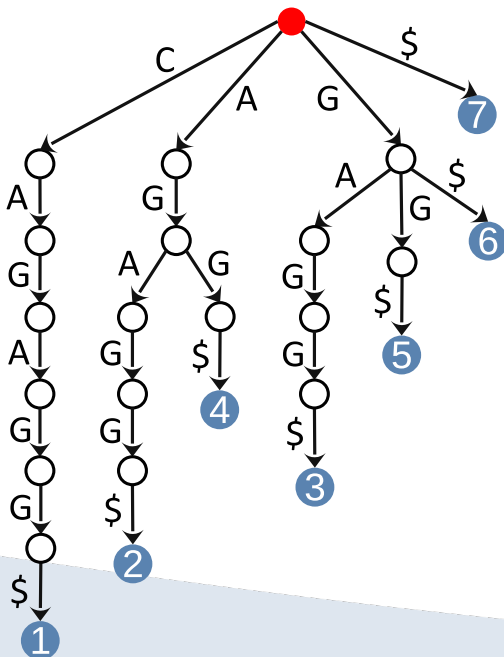
Sufiksno polje (3) (info)

- Stvaranje sufiksnog polja može biti linearno korištenjem posebnih algoritama, kao *skew* algoritam
- Korištenjem koncepata LCP-a i ℓ -matrice, pretraživanje se može ubrzati na $O(|q| + \log n)$
- Ti su algoritmi i načini stvaranja kompleksni. Studenti koji žele znati više o ovome mogu pronaći te teme u skripti NASP-a.

Sufiksni Trie (1)

- Listu svih sufiksa određenog znakovnog niza možemo transformirati u Trie
- Za alfabet $\Sigma = \{C, A, G, T, \$\}$ (DNA baze) i $t = CAGAGG\$$ imamo sljedeću listu svih sufiksa
- Pretvoreno u sufiksni Trie

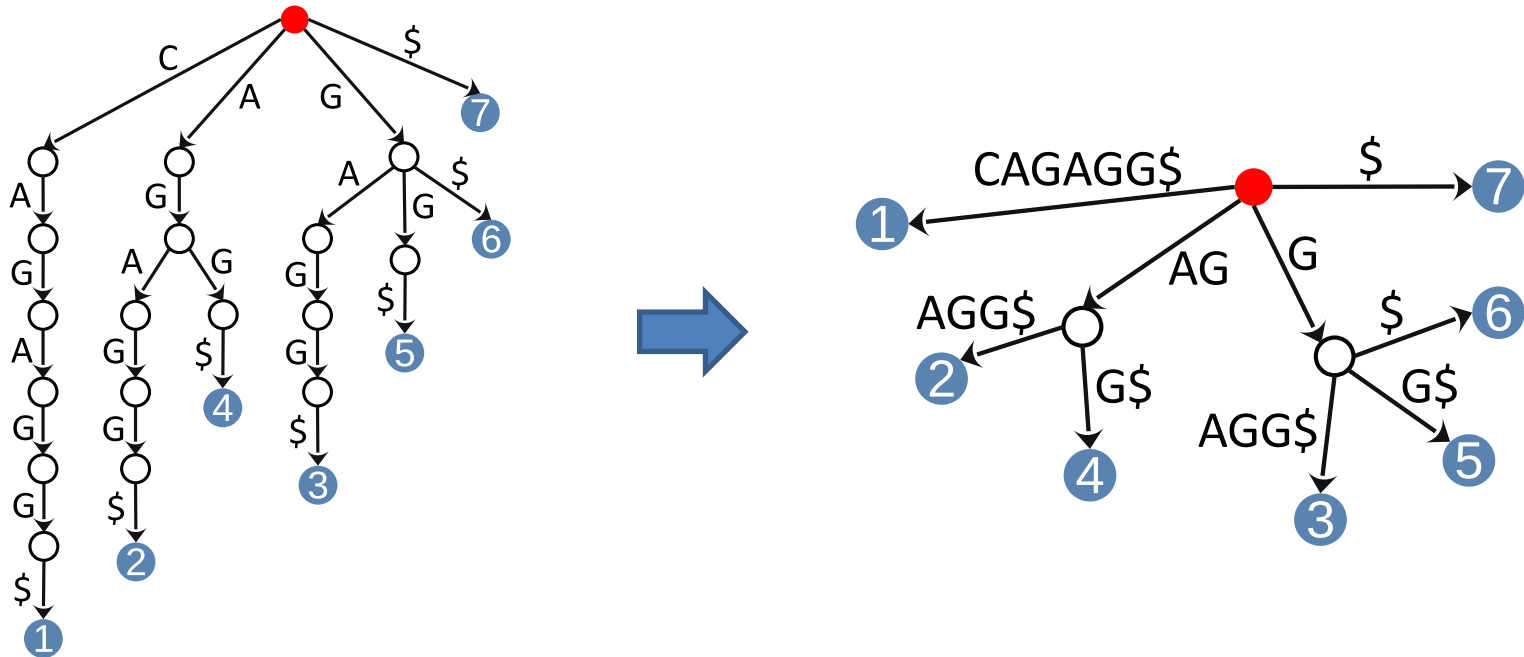
C	A	G	A	G	G	\$
A	G	A	G	G	\$	
G	A	G	G	\$		
A	G	G	\$			
G	G	\$				
G	\$					
\$						



- Svaki list sufiksnog Trie-a sadrži redni broj tog sufiksa ili A_i

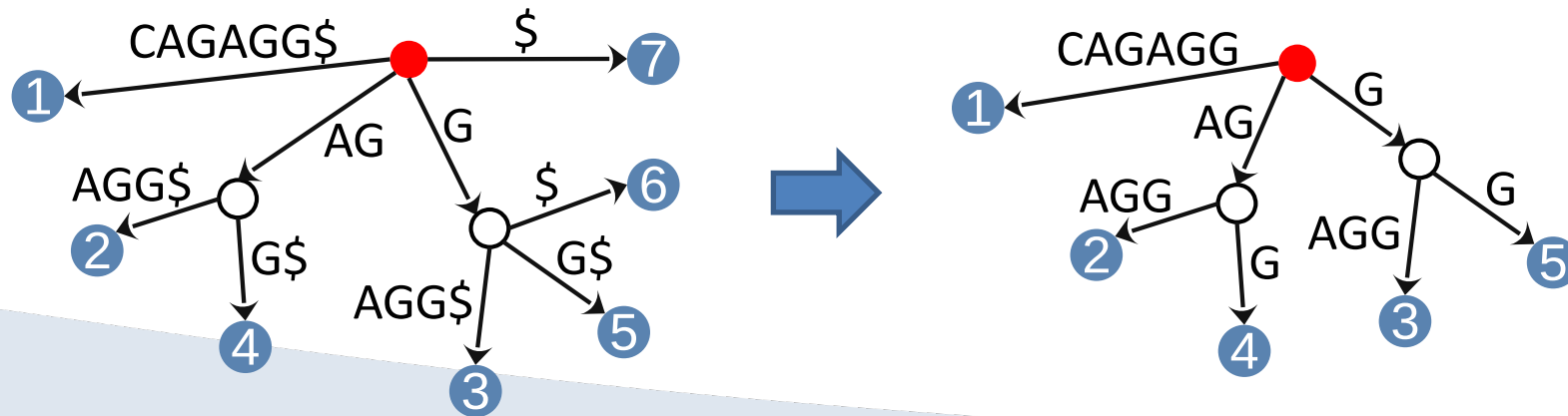
Sufiksno stablo (1)

- Sufiksno stablo je kompresirani sufiksni Trie ili Patricia stablo svih sufiksa određenog znakovnog niza
- Ovo se još i naziva **eksplicitno sufiksno stablo**



Sufiksno stablo (2)

- **Implicitno sufiksno stablo** dobiva se iz eksplicitnog sljedećom transformacijom:
 1. Uklonimo sve NULL terminatore (\$) za bridova eksplicitnog sufiksnog stabla
 2. Uklonimo sve bridove koji su ostali s praznim znakovnim nizovima ϵ , uključujući i njihova podstabla
 3. Ponovno kompresiramo sufiksno stablo: svi čvorovi osim korijenskog, koji imaju manje od dvoje djece se uklanjaju, a njihove tranzicije konkateneraju



Ukkonenov algoritam (1)

- Naivni pokušaj stvaranja sufiksnog stabla je $O(n^2)$
 - Ovo je nekoliko puta bilo poboljšano raznim algoritmima: McCreigh i Weiner
 - Ukkonen gradi svoj algoritam na Weinerovom algoritmu
 - Ukkonenov algoritam je *online* algoritam koji gradi sufiksno stablo znak po znak iz t - $O(n \log n)$ za najgori slučaj i $O(n)$ za konačni alfabet
- Imamo implicitno sufiksno stablo nakon i koraka koje označimo S_i
 - Uzimamo znak $t[i + 1]$ i izvršavamo $(i + 1)$. fazu nadogradnje stabla
 - $(i + 1)$. faza se izvodi u $i + 1$ koraka
 - U svakom koraku dodajemo $t[i + 1]$ na sve sufikse prefiksa $t[1, i]$
 - $t[a, b]$, gdje je $a \leq b$, predstavlja podniz znakovnog niza t od a -tog do b -tog znaka, uključivo, uzevši da znakovni niz počne od 1
 - U $(i + 1)$. koraku dodajemo $t[i + 1]$ na korijenski čvor stabla
 - Rezultat je implicitno sufiksno stablo S_{i+1}

Ukkonenov algoritam (2)

- Za $t = ABABABC\$$ imamo sljedeće faze i korake
 - Varijablom j definiramo sufiks prefiksa $t[1, i]$, ili $t[j, i]$
- Prolazimo stablo za svaki $t[j, i]$ i dodajemo $t[i + 1]$, a imamo tri slučaja:
 1. Kod prolaska $t[j, i]$ dolazimo do lista. Cijela vertikalna putanja predstavlja $t[j, i]$. U zadnjoj tranziciji prije lista samo dodajemo $t[i + 1]$ na kraj tranzicije.

Phase 1, $i + 1 = 1, t[i + 1] = A$							
Step 1	A						
Phase 2, $i + 1 = 2, t[i + 1] = B$							
Step 1, $j = 1$	A	B					
Step 2	B						
Phase 3, $i + 1 = 3, t[i + 1] = A$							
Step 1, $j = 1$	A	B	A				
Step 2, $j = 2$	B	A					
Step 3	A						
Phase 4, $i + 1 = 4, t[i + 1] = B$							
Step 1, $j = 1$	A	B	A	B			
Step 2, $j = 2$	B	A	B				
Step 3, $j = 3$	A	B					
Step 4,	B						
...							
Phase 7, $i + 1 = 7, t[i + 1] = C$							
Step 1, $j = 1$	A	B	A	B	A	B	C
Step 2, $j = 2$	B	A	B	A	B	C	
...							
Step 6, $j = 6$	B	C					
Step 7	C						

Ukkonenov algoritam (3)

2. Prošli smo sve znakove $t[j, i]$ i stali na sredini tranzicije. Sljedeći znak u tranziciji **nije** $t[i + 1]$. Dodajemo unutarnji čvor na stablo i razdvajamo tranziciju. Dodajemo list i tranziciju $t[i + 1]$ od novo dodanog unutarnjeg čvora do novo dodanog lista. Ista operacija kao i kod Patricia stabla.
3. Prošli smo sve znakove $t[j, i]$ i stali na sredini tranzicije. Sljedeći znak u tranziciji **je** $t[i + 1]$. Ne radimo ništa jer $t[j, i]t[i + 1]$ je već sadržan u stablu.

Nakon što smo završili gradnju implicitnog sufiksnog stabla, pretvaramo ga u eksplicitno dodavanjem \$ u sve tranzicije koje vode u listove stabla.

Ukkonenov algoritam (4)

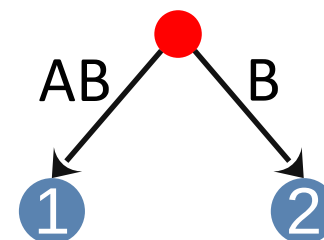
- **Sufiksne veze** dodatak su implicitnom sufiksnom stablu koji omogućavaju brže prolaženje stabla za $t[j, i]$
 - Dodaju se samo između unutarnjih čvorova
 - Imamo unutarnji čvor n_1 u koji smo stigli prolaskom $t[j, i]$
 - Ako imamo neki drugi unutarnji čvor n_2 do kojeg se stigne prolaskom $at[j, i]$, tada je n_2 spojen na n_1 sufiksnom vezom
 - Ovo samo znači da za sve faze $\geq i$ trebamo početi od oba čvora n_1 i n_2 , čime zapravo preskačemo sve korake koji uključuju podniz $t[j, i]$
 - To znači da ne moramo startati s $j=1$, već odmah nakon podniza $t[j, i]$
 - n_2 može biti i korijenski čvor kada je $a = \epsilon$ i $j = i$
 - Ako imamo dva unutarnja čvora, n_2 do kojeg se stigne prolaskom $at[j, i]$ i n_1 do kojeg se stigne prolaskom $t[j, i]$, a $t[j, i]$ je očiti sufiks od $at[j, i]$, tada nam prolaz $t[j, i]$ slijedi nakon $at[j, i]$, te već unaprijed znamo u kojem ćemo se čvoru naći

Ukkonenov algoritam (5)

- Sjetimo se da se unutarnji čvorovi dodaju samo ako se $t[i + 1]$ dodaje usred tranzicije
- Ako u (j_1) . koraku faze i prijedemo $t[j_1, i]$, tada se za novi $t[i + 1]$ stvara čvor n_2
- U nekom (j_2) . koraku, gdje je $j_2 > j_1$, prijedemo $t[j_2, i]$ za koji vrijedi $t[j_1, i] = at[j_2, i]$ i stvorimo n_1 za novi $t[i + 1]$
- Dovoljan i nužan uvjet je da je $j_2 = j_1 + 1$, pa je tada a sačinjen od jednog znaka

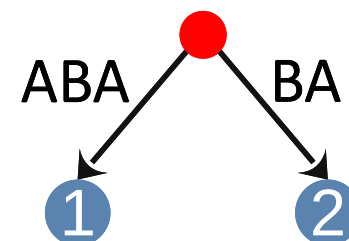
Primjer Ukkonen (1)

- Stvaramo implicitno sufiksno stablo korištenjem Ukkonenovog algoritma za $t = ABABABC\$$
- **Faza 1** : $i + 1 = 1, t[i + 1] = A$
 - Korak 1 : Imamo samo korijenski čvor. Dodajemo novu tranziciju i list.
- **Faza 2** : $i + 1 = 2, t[i + 1] = B$
 - Korak 1 : $j = 1, t[j, i] = A$. Prolazak nas dovodi do lista 1, što je slučaj 1. Dodajemo B na posljednju tranziciju.
 - Korak 2 : $t[j, i] = \epsilon$. Dodajemo novu tranziciju na korijenski čvor i pripadni list.



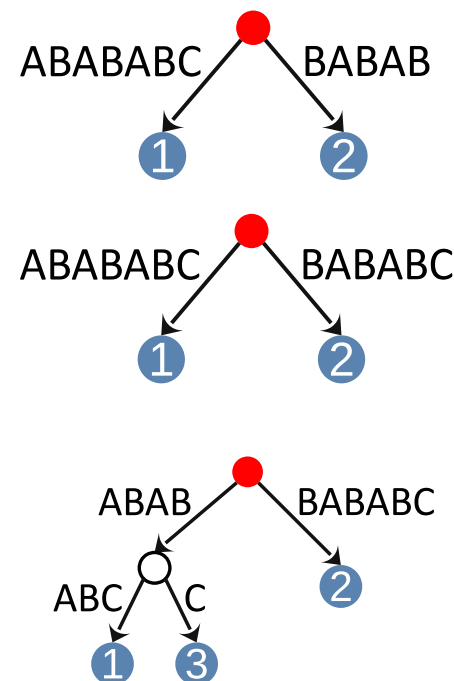
Primjer Ukkonen (2)

- **Faza 3** : $i + 1 = 3, t[i + 1] = A$
 - Korak 1 : $j = 1, t[j, i] = AB$. Prolazak nas dovodi do lista 1, što je slučaj 1. Dodajemo A na posljednju tranziciju.
 - Korak 2 : $j = 2, t[j, i] = B$. Prolazak nas dovodi do lista 2, što je slučaj 1. Dodajemo A na posljednju tranziciju.
 - Korak 3: $t[j, i] = \epsilon$. S obzirom na ϵ , ostajemo u korijenskom čvoru, no prolazak $t[i + 1]$ nas vodi u tranziciju, što je slučaj 3. Ne radimo ništa.



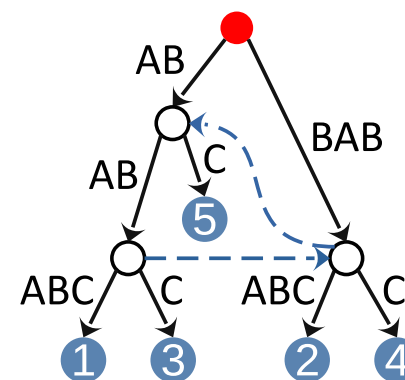
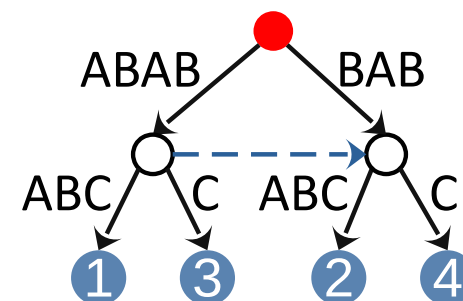
Primjer Ukkonen (3)

- Preskačemo niz istih faza
- **Faza 7** : $i + 1 = 7, t[i + 1] = C$
 - Korak 1 : $j = 1, t[j, i] = ABABAB$. Prolazak nas dovodi do lista 1, što je slučaj 1. Dodajemo C na posljednju tranziciju.
 - Korak 2 : $j = 2, t[j, i] = BABAB$. Prolazak nas dovodi do lista 2, što je slučaj 1. Dodajemo C na posljednju tranziciju.
 - Korak 3 : $j = 3, t[j, i] = ABAB$. Prelazak nas vodi na sredinu tranzicije, a sljedeći znak nije C, što je slučaj 2. Razdvajamo puteve za $ABABABC$ i $ABABC$.



Primjer Ukkonen (4)

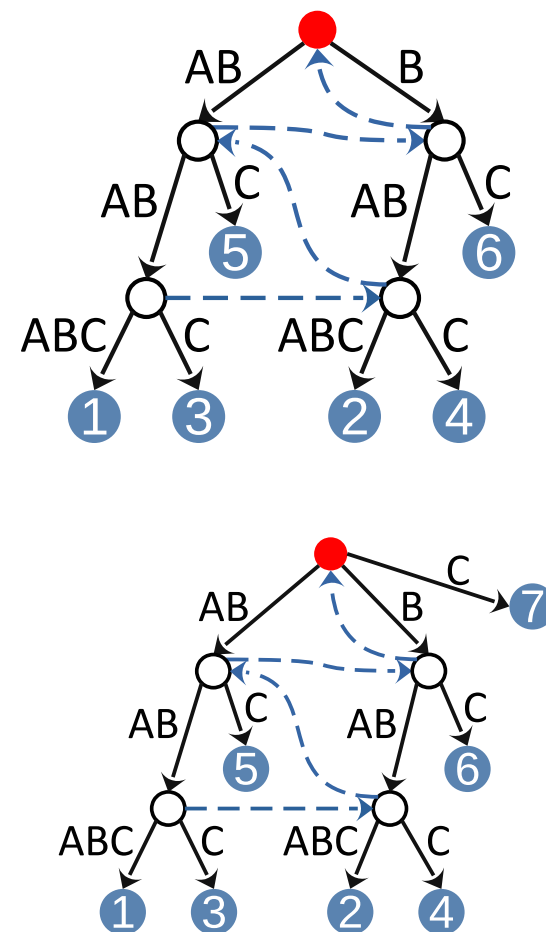
- Korak 4 : $j = 4, t[j, i] = BAB$. Prelazak nas vodi na sredinu tranzicije, a sljedeći znak nije C, što je slučaj 2. Razdvajamo puteve za $BABABC$ i $BABC$. U ovom koraku imamo dva unutarnja čvora $n_1 = BAB$ i $n_2 = ABAB$. Spajamo sufiksnom vezom.
- Korak 5 : $j = 5, t[j, i] = AB$. Prelazak nas vodi na sredinu tranzicije, a sljedeći znak nije C, što je slučaj 2. Razdvajamo puteve za $\{ABABABC, ABABC\}$ i ABC . U ovom koraku imamo dva unutarnja čvora $n_1 = AB$ i $n_2 = BAB$. Spajamo sufiksnom vezom.



Primjer Ukkonen (5)

- Korak 6 : $j = 6, t[j, i] = B$. Prelazak nas vodi na sredinu tranzicije, a sljedeći znak nije C, što je slučaj 2. Razdvajamo puteve za $\{BABABC, BABC\}$ i BC . U ovom koraku imamo dva unutarnja čvora $n_1 = B$ i $n_2 = AB$. Spajamo sufiksnom vezom.
- Korak 7 : $t[j, i] = \epsilon$. S obzirom da niti jedna tranzicija ne počinje sa C, dodajemo novi list i novu tranziciju. Dodajemo i sufiksnu vezu između čvora $n_2 = B$ i korijenskog čvora.

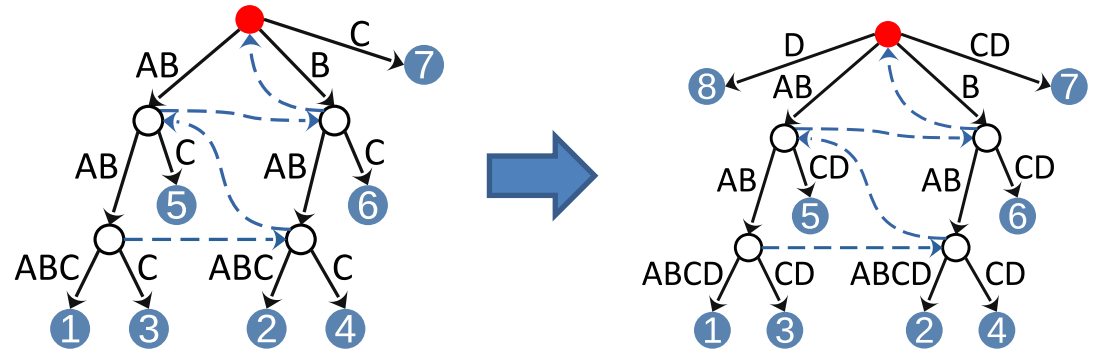
$t[1,7]$	A	B	A	B	A	B	C
$t[2,7]$		B	A	B	A	B	C
$t[3,7]$			A	B	A	B	C
$t[4,7]$				B	A	B	C
$t[5,7]$					A	B	C
$t[6,7]$						B	C
$t[7,7]$							C



Ukkonenov algoritam (7)

- Na $t[6,7]$ krećemo ispočetka jer u našem skupu čvorova nemamo korijenski čvor

$t[1,7]$	A	B	A	B	A	B	C
$t[2,7]$		B	A	B	A	B	C
$t[3,7]$			A	B	A	B	C
$t[4,7]$				B	A	B	C
$t[5,7]$					A	B	C
$t[6,7]$						B	C
$t[7,7]$							C



- Time smanjujemo i broj koraka, a i broj prijeđenih znakova u koraku
 - Pravi dobitak je u smanjenju broja prijeđenih znakova u određenom koraku
 - Primijetimo da se unutarnji čvorovi ulančavaju sufiksним vezama
 - To nas dovodi do provjere tranzicija prema listovima
 - Korištenjem reprezentacije s uređenim parom, dodavanje znaka se svodi na ažuriranje duljine

Pitanja ?