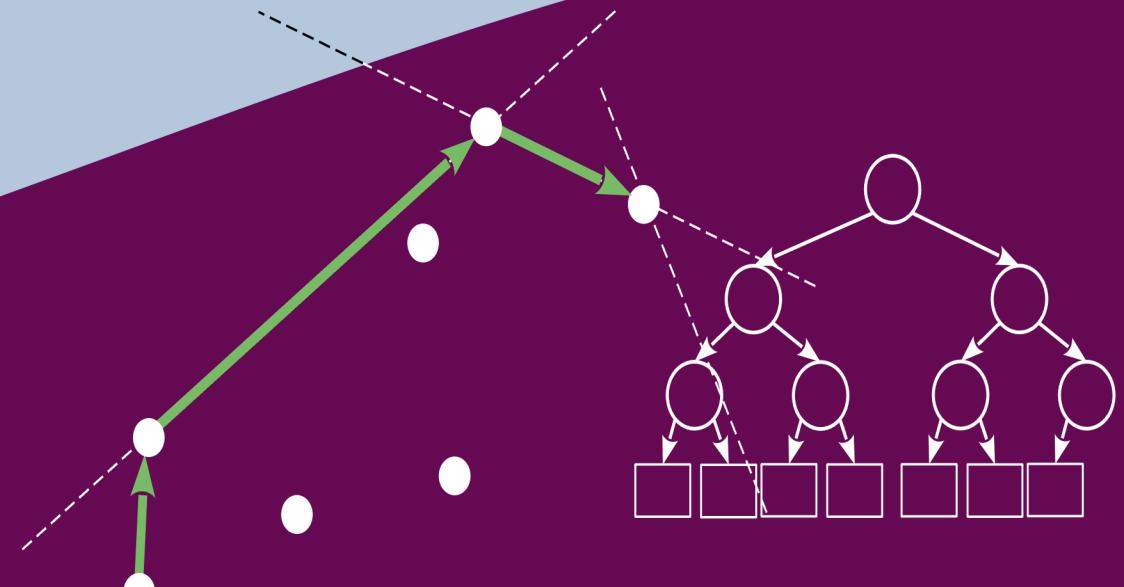
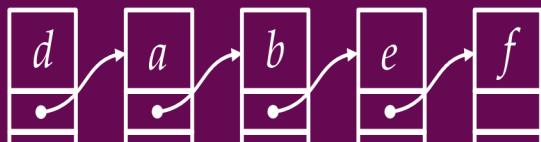


Napredni algoritmi i strukture podataka

Tjedan 2: B-stabla i Crveno-crna stabla (RB)



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 daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

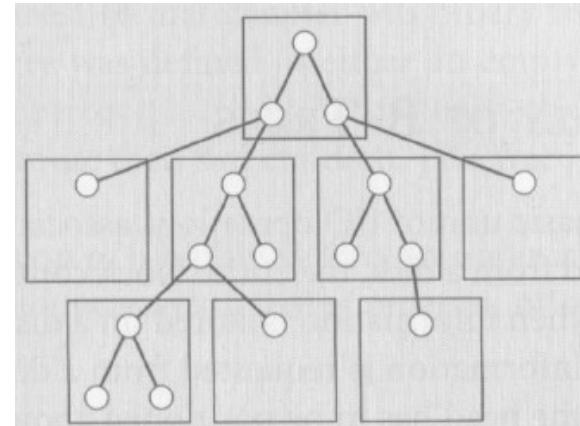
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

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

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

Motivacija

- Vanjska memorija
 - Sekvencijalno čitanje po blokovima
 - Susjedni čvorovi u stablu mogu biti razasuti u udaljenim blokovima



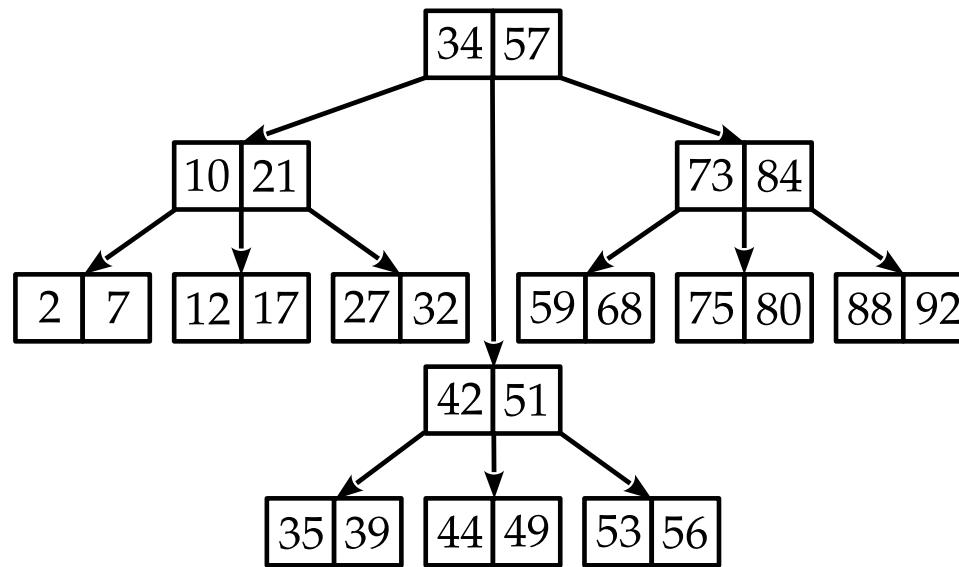
- B-stabla ublažavaju efekte ograničenja sekvencijalnog blokovskog čitanja
 - Veličina čvora se prilagođava veličini bloka

Karakteristike

- Potpuna balansiranost
- Sortiranje podataka po vrijednosti ključa
- Čuvanje određenog broja elemenata u jednom čvoru

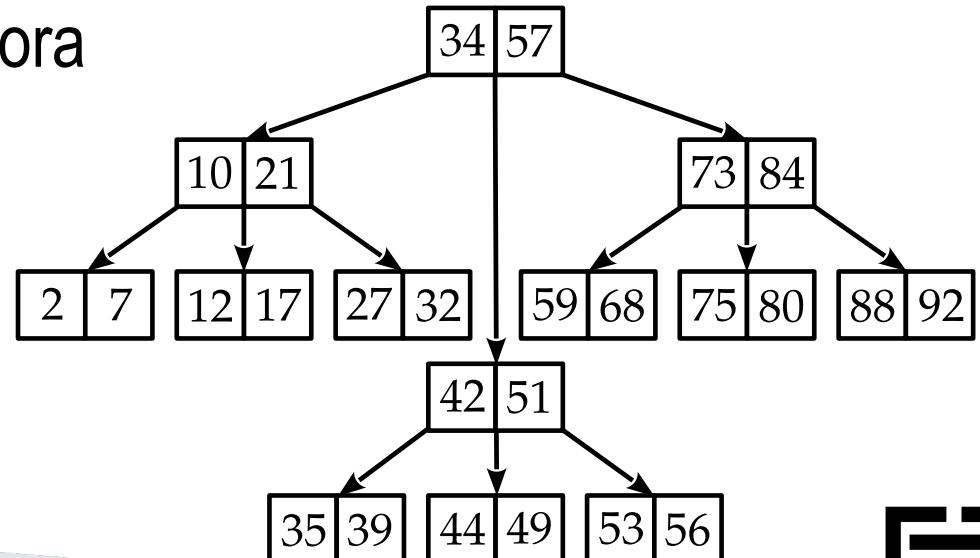
M stabla

- M stabla (*multiway tree*): stabla u kojima čvorovi mogu imati proizvoljan broj djece
- M stablo m -tog reda: M stablo u kojem čvorovi mogu imati najviše m djece.



M stabla

- Svojstva M stabla m -tog reda:
 1. Svaki čvor ima najviše m djece i $m-1$ podataka (ključeva)
 2. Ključevi u čvorovima su sortirani
 3. Ključevi u prvih i djece nekog čvora su manji od $\hat{\imath}$ -tog ključa promatranog čvora
 4. Ključevi u zadnjih $m-i$ djece nekog čvora su veći od $\hat{\imath}$ -tog ključa promatranog čvora

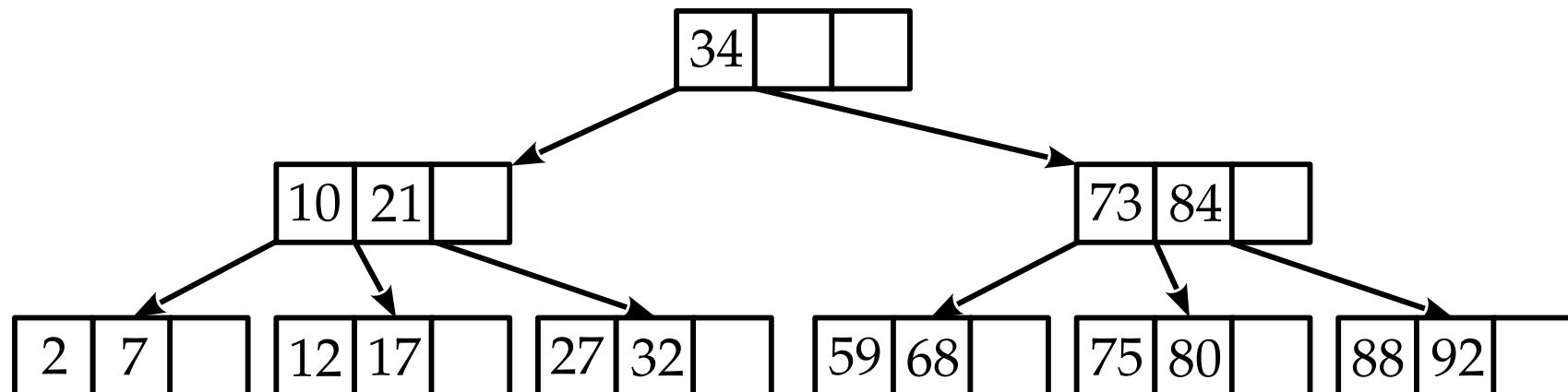


B-stablo

- B stablo m -tog reda je M-stablo sa dodatnim svojstvima:
 1. Korijen ima najmanje dvoje djece, osim ako je ujedno i list (jedini čvor u stablu)
 2. Svaki čvor, osim korijena i listova, sadrži **barem** $k-1$ ključeva i k pokazivača na podstabla (ima k djece), pri čemu je
$$\lceil m/2 \rceil \leq k \leq m$$
 3. Svi listovi sadrže **barem** $k-1$ ključeva, pri čemu je
$$\lceil m/2 \rceil \leq k \leq m$$
 4. Svi listovi su na istoj razini
- } Savršena uravnoteženost

B-stablo

- Osobitosti
 - Popunjeno barem 50%
 - Savršeno uravnoteženo



B-stablo

- Implementacija #1
 - Struktura (klasa) s poljem od $m-1$ ključeva i poljem od m pokazivača – u Pythonu može biti i jedno jedinstveno polje gdje se izmjenjuju pokazivači i ključevi
 - Moguće dodati podatke radi lakšeg održavanja (npr. broj upisanih podataka u čvoru)
- Implementacija #2
 - Svaki čvor je dvostruko povezana lista
 - Svaki ključ ima pokazivače na djecu – samo posljednji ključ koristi oba pokazivača

Algoritam pretraživanja B-stabla

1. Ući u čvor i redom pregledavati ključeve sve dok je trenutni manji od traženog, a još ima neprovjerenih
 - Prvi čvor u koji se ulazi je korijen
2. Ako je 1. korak završio zbog nailaska na ključ veći od traženog ili zbog dolaska do kraja čvora, spusti se na razinu niže i ponovi prvi korak
 - Ako nema niže razine, nema traženog ključa

Pretraživanje B-stabla - implementacija

```
function BTREESEARCH( $n, v_s$ )
     $n_v \leftarrow$  starting value of the node  $n$ 
    while  $value(n_v) < v_s$  and  $next(n_v)$  is not nil do
         $n_v \leftarrow next(n_v)$ 
        if  $value(n_v) = v_s$  then
            return  $n$ 
        else if  $next(n_v)$  is nil and  $value(n_v) < v_s$  then
            if  $rightChild(n_v)$  is not nil then
                return BTREESEARCH( $rightChild(n_v), v_s$ )
            else
                return no searched key
        else
            if  $leftChild(n_v)$  is not nil then
                return BTREESEARCH( $leftChild(n_v), v_s$ )
            else
                return no searched key
```

- Napomena
 - $value(n_v)$ – vrijednost ključa n_v – npr. cijeli broj
 - $next(n_v)$ – sljedeći ključ nakon n_v u listi ključeva čvora – ovo ovisi o implementaciji čvora
 - $leftChild(n_v)$ i $rightChild(n_v)$ – lijevo i desno dijete ključa n_v

Dodavanje podataka u B-stablo

- Jednostavnije je graditi B-stablo **odozdo prema gore**
- Algoritam:
 1. pronaći list u koji bi trebalo smjestiti novi podatak
 2. ako ima mjesta, upisati novi podatak
 3. ako je taj list pun, “rascijepiti” (*split*) ga (napraviti novi list, ravnomjerno raspodijeliti elemente između dva čvora, a središnji element upisati u roditelja)
 4. ako je i roditelj pun, “rascijepiti” i roditelja (ponavljati proceduru iz koraka 3)
 5. ako je i korijen pun, “rascijepiti” ga i napraviti novi korijen

Dodavanje podataka u B-stablo

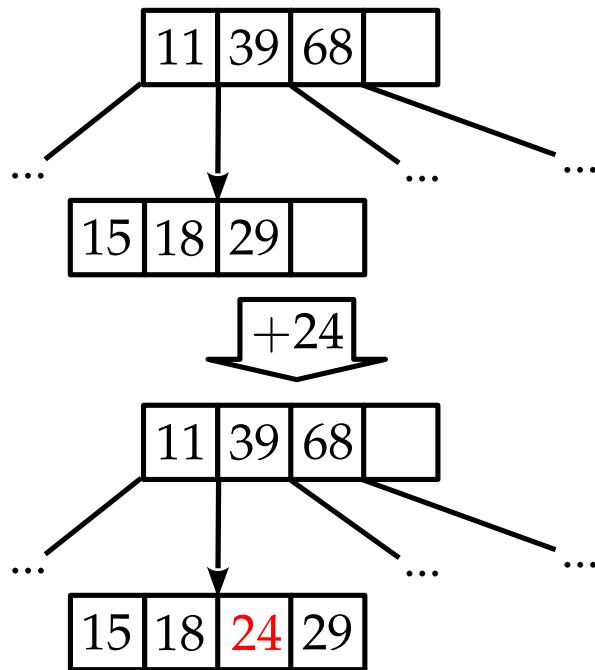
- Prilikom ubacivanja novog podatka moguće su 3 situacije:
 1. list u koji treba ići novi element nije pun
 - ubaciti novi element u taj list na odgovarajuće mjesto, pomicući po potrebi prethodni sadržaj
 2. list u koji treba ići novi element je pun, ali korijen stabla nije
 - list se dijeli (stvara se novi čvor) i svi elementi se ravnomjerno raspoređuju, s tim da se središnji element upisuje u roditelja
 3. list u koji treba ići novi element je pun, a isto tako i korijen stabla
 - kad se razdijeli korijen nastaju dva B-stabla koja treba sjediniti

Dodavanje podataka u B-stablo

- Sjedinjenje u trećem slučaju se postiže stvaranjem još jednog čvora koji će biti novi korijen i upisivanjem središnjeg elementa u njega
 - To je jedini slučaj koji završava povisivanjem stabla
 - **B-stablo je uvijek savršeno uravnoteženo**

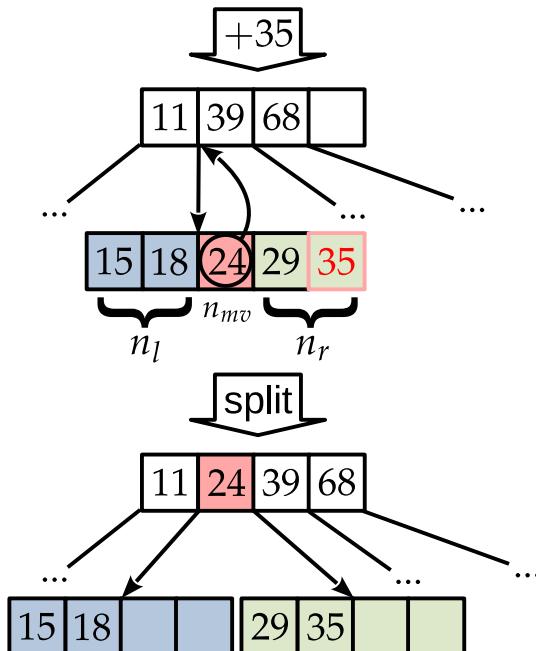
Primjer dodavanja podataka u B-stablo

- **Primjer 1:** dodajemo 24 u list u kojem ima manje od $m - 1$ ključeva. Nema potrebe za restrukturiranjem B-stabla.



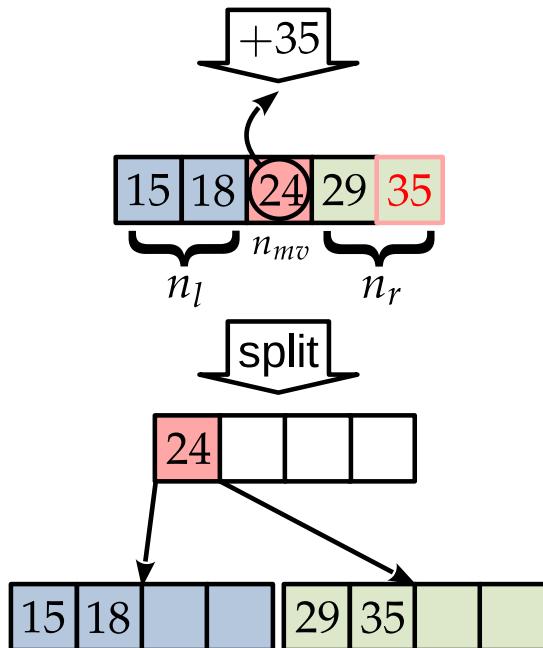
Primjer dodavanja podataka u B-stablo

- **Primjer 2:** dodajemo 35 u list u kojem ima točno $m - 1$ ključeva i koji ima roditeljski čvor.
 - Dešava se preljev u listu.
 - List se dijeli na središnji ključ i dva dijela.
 - Središnji ključ ubacujemo u roditelja, a list razdvajamo na dva čvora.



Primjer dodavanja podataka u B-stablo

- **Primjer 3:** dodajemo 35 u čvor u kojem ima točno $m - 1$ ključeva i koji **nema** roditeljski čvor (očito je korijenski čvor).
 - Dešava se preljev u čvoru.
 - List se dijeli na središnji ključ i dva dijela.
 - Središnji ključ koristimo za stvaranje novog korijenskog čvora, a list razdvajamo na dva čvora.



Primjer dodavanja podataka u B-stablo

- B-stablo reda 4 – 4 kazaljke, 3 ključa
- Dodajemo redom ključeve:
**12,75,34,62,19,25,66,30,33,71,47,21,15,
23,27**

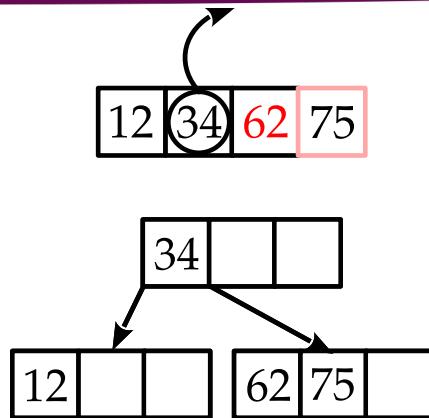


- **Korak 1:** Formiramo korijenski čvor s prvim ključem 12

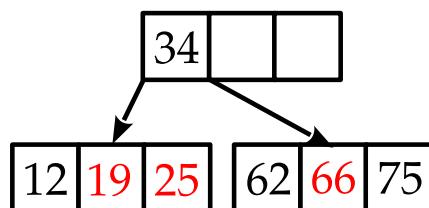


- **Korak 2:** U čvor dodajemo ključeve do dok možemo – dodajemo 75 i 34

Primjer dodavanja podataka u B-stablo

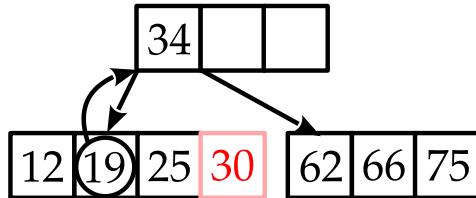


- **Korak 3:** Dodavanjem ključa **62**, dolazi do preljeva u korijenskom čvoru, kojeg razdvajamo uz stvaranje novog korijenskog čvora.

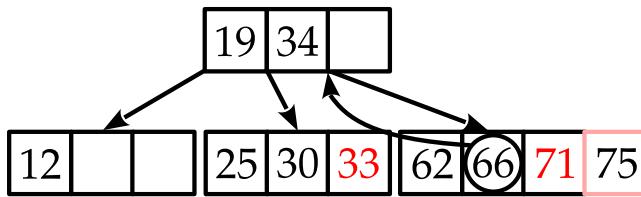
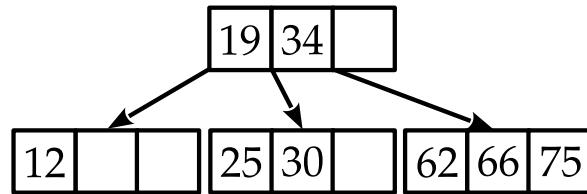


- **Korak 4:** Dodajemo ključeve **19, 25 i 66** direktno u listove.

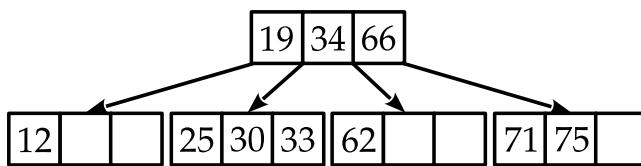
Primjer dodavanja podataka u B-stablo



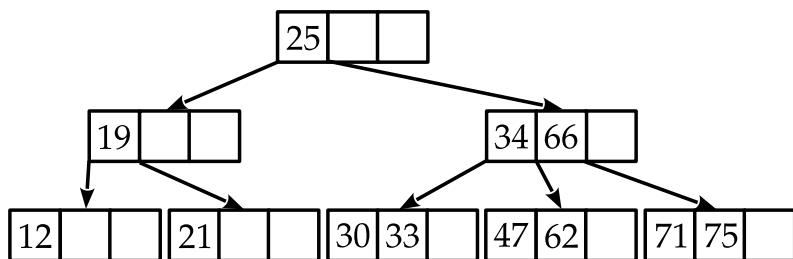
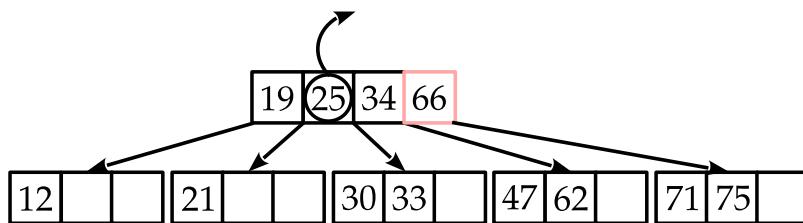
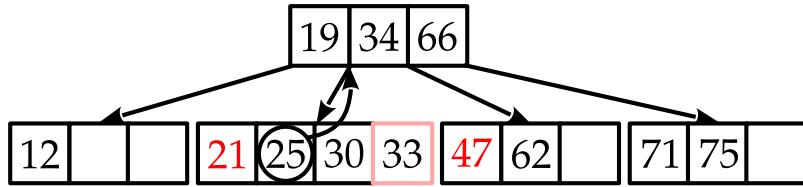
- **Korak 5:** Dodavanjem ključa 30, dolazi do preljeva u lijevom listu, kojeg razdvajamo uz ubacivanje srednjeg ključa u korijenski čvor.



- **Korak 6:** Dodajemo ključ 33, a zatim 71. Dodavanjem 71 izazivamo preljev u desnom listu, kojeg razdvajamo uz ubacivanje srednjeg ključa u korijenski čvor.



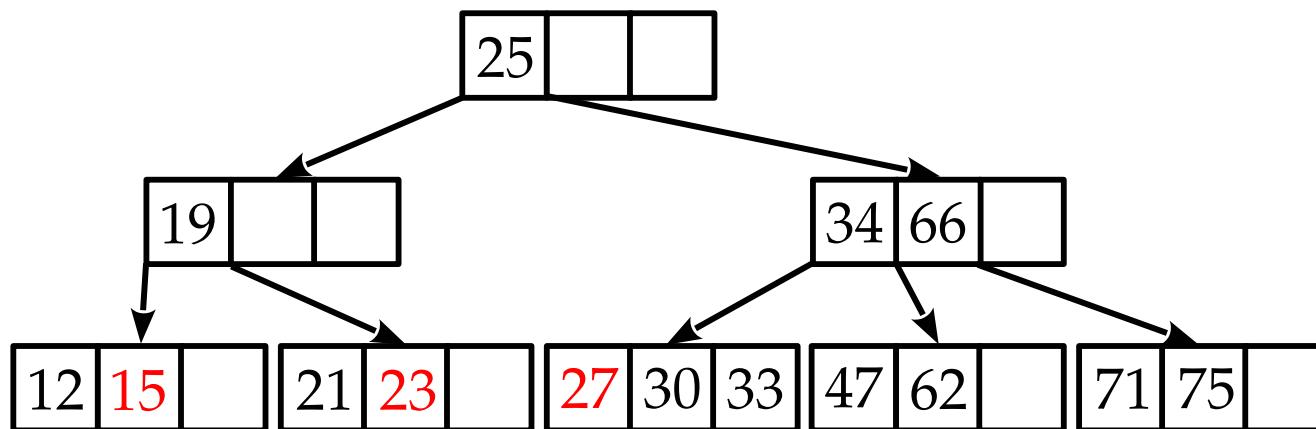
Primjer dodavanja podataka u B-stablo



- **Korak 7:** Dodajemo ključ 47, a zatim ključ 21.
 - Dodavanjem 21 izazivamo preljev u listu, kojeg razdvajamo uz ubacivanje srednjeg ključa u korijenski čvor.
 - Ubacivanjem 25 u korijenski čvor izazivamo preljev korijenskog čvora, kojeg razdvajamo uz stvaranje novog korijenskog čvora.

Primjer dodavanja podataka u B-stablo

- **Korak 8:** Dodajemo ključeve **15, 23 i 27** direktno u listove B-stabla bez restrukturiranja.

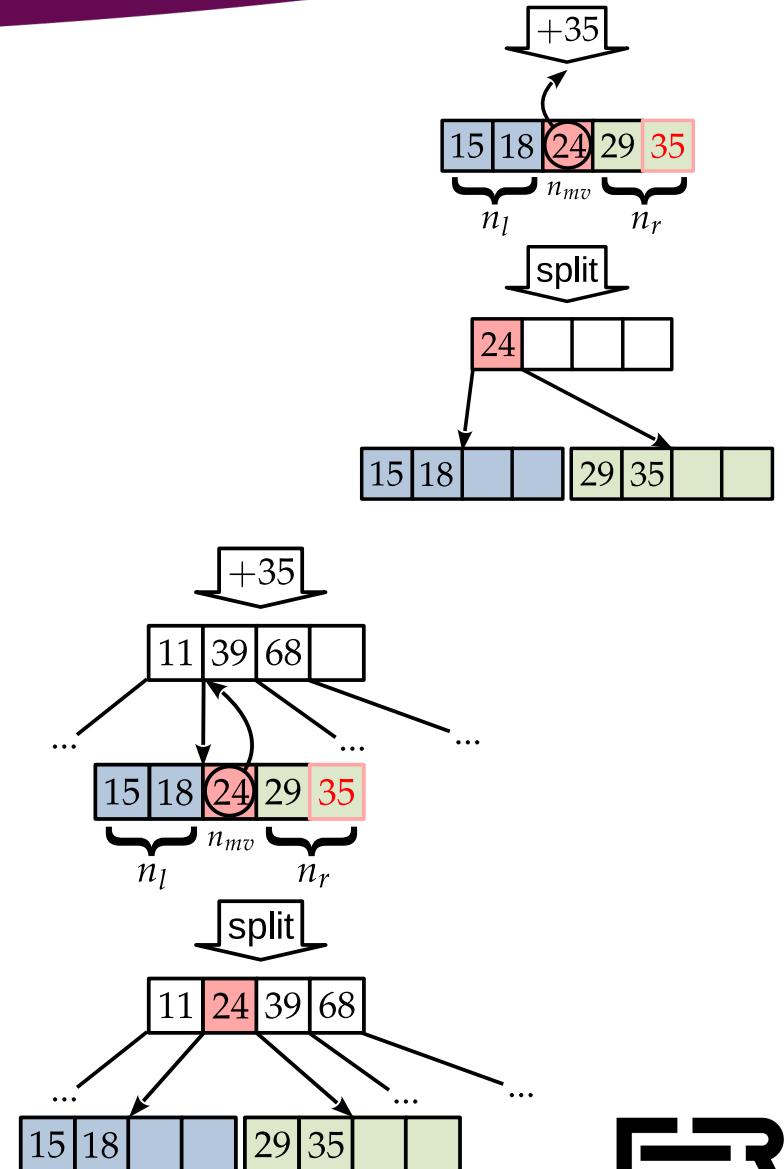


Implementacija dodavanja u B-stablo

```

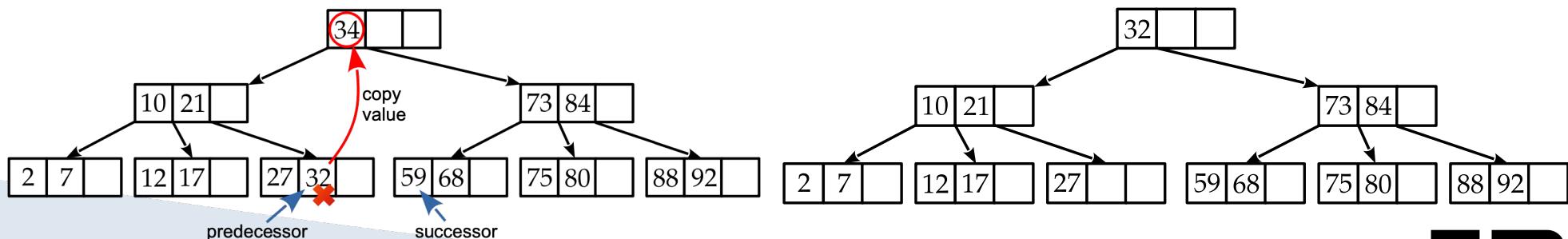
procedure BTREESPLIT(btree, n)
    if  $|n| == m$  then
         $n_l \leftarrow$  new node having first  $\lceil m/2 \rceil - 1$  values in the node n
         $n_{mv} \leftarrow$  the next value in the node n
         $n_r \leftarrow$  new node having the rest of values from the node n
         $n_{last} \leftarrow$  the last value in nl
        rightChild(nlast)  $\leftarrow$  leftChild(nm)
        if parent(n) is nil then
             $n_{root} \leftarrow$  new node
             $n_{mv}' \leftarrow$  insert value(nmv) into the node nroot
            root of btree  $\leftarrow$  nroot
            leftChild(nmv')  $\leftarrow$  nl
            rightChild(nmv')  $\leftarrow$  nr
        else
             $n_{mv}' \leftarrow$  insert value(nmv) into the parent node of n
            leftChild(nmv')  $\leftarrow$  nl
            rightChild(nmv')  $\leftarrow$  nr
            BTREESPLIT(btree, parent of n)
    procedure BTREEINSERT(btree, vn)
        (n, nv)  $\leftarrow$  BTREESearch(root of btree, vn)
        insert vn into the node n
        BTREESPLIT(btree, n)

```



Brisanje podataka u B-stablu

- 2 slučaja:
 1. brisanje elementa u listu stabla
 2. brisanje elementa u čvoru
 - svodi se na brisanje elementa iz lista
 - na mjesto elementa koji treba izbrisati upisuje se njegov neposredni prethodnik (koji može biti samo u listu), potom se u listu briše prepisani element standardnim postupkom za brisanje lista

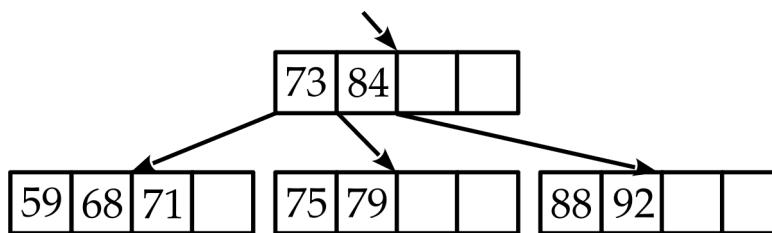
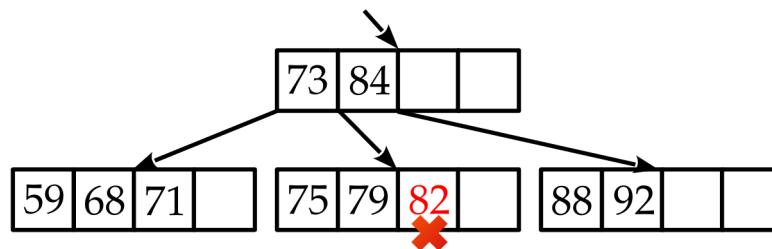


Brisanje podataka u B-stablu

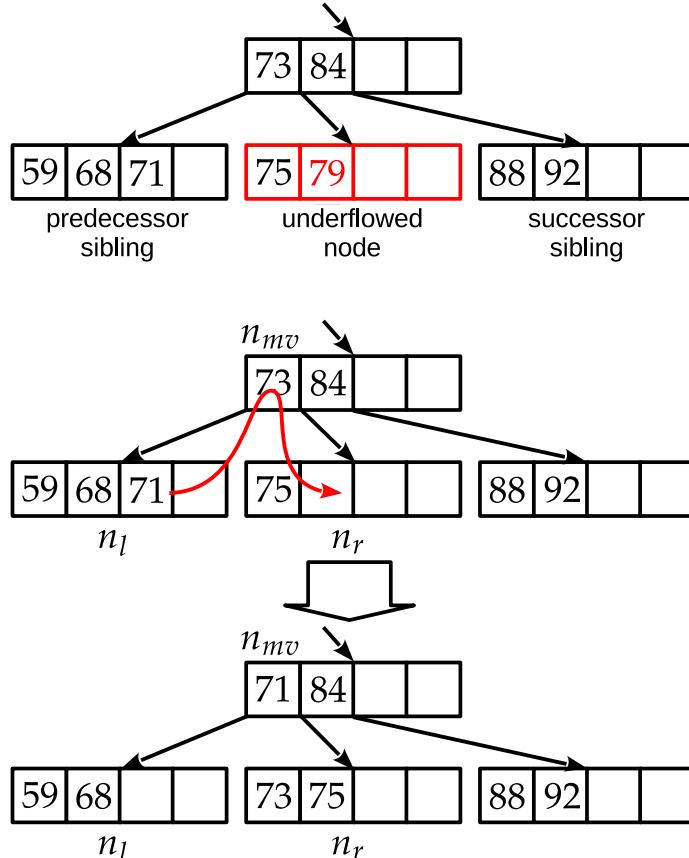
1. list i nakon brisanja elementa ima $\geq \lceil m/2 \rceil - 1$ ključeva; **KRAJ**
2. broj preostalih elemenata(ključeva) $< \lceil m/2 \rceil - 1$
 1. ako lijevo ili desno postoji susjed s $> \lceil m/2 \rceil - 1$ ključeva
 - elemente lista, elemente susjeda i središnji element iz roditelja ravnomjerno rasporediti u list i susjeda, a kao novi središnji element u roditelja upisati središnji element ujedinjenog skupa (unije) elemenata; **KRAJ**
 2. list i susjed se sjedajuju (svi elementi lista i susjeda + središnji element iz roditelja se upisuju u list, a susjed se briše); **NASTAVITI s roditeljem**
 3. postupkom 2.2 dolazimo do korijena:
 - ako korijen ima više od 1 elementa: sjediniti trenutni čvor i susjeda kao u 2.2; **KRAJ**
 - inače: sve elemente lista, susjeda i korijena upisati u 1 čvor koji postaje novi korijen, a 2 čvora se brišu iz stabla; **KRAJ**

Primjer brisanja podataka iz B-stabla

- **Primjer 1:** brišemo ključ **82** iz lista. List je nakon brisanja još uvijek popunjen $\geq 50\%$ zato jer ima $\geq \lceil m/2 \rceil - 1$ ključeva. Nema potrebe za restrukturiranjem B-stabla.

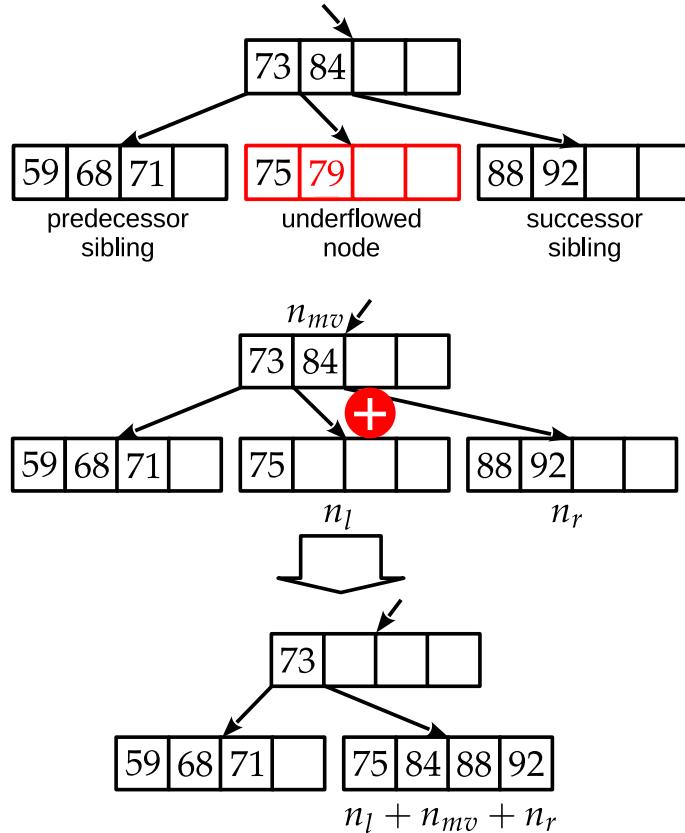


Primjer brisanja podataka iz B-stabla



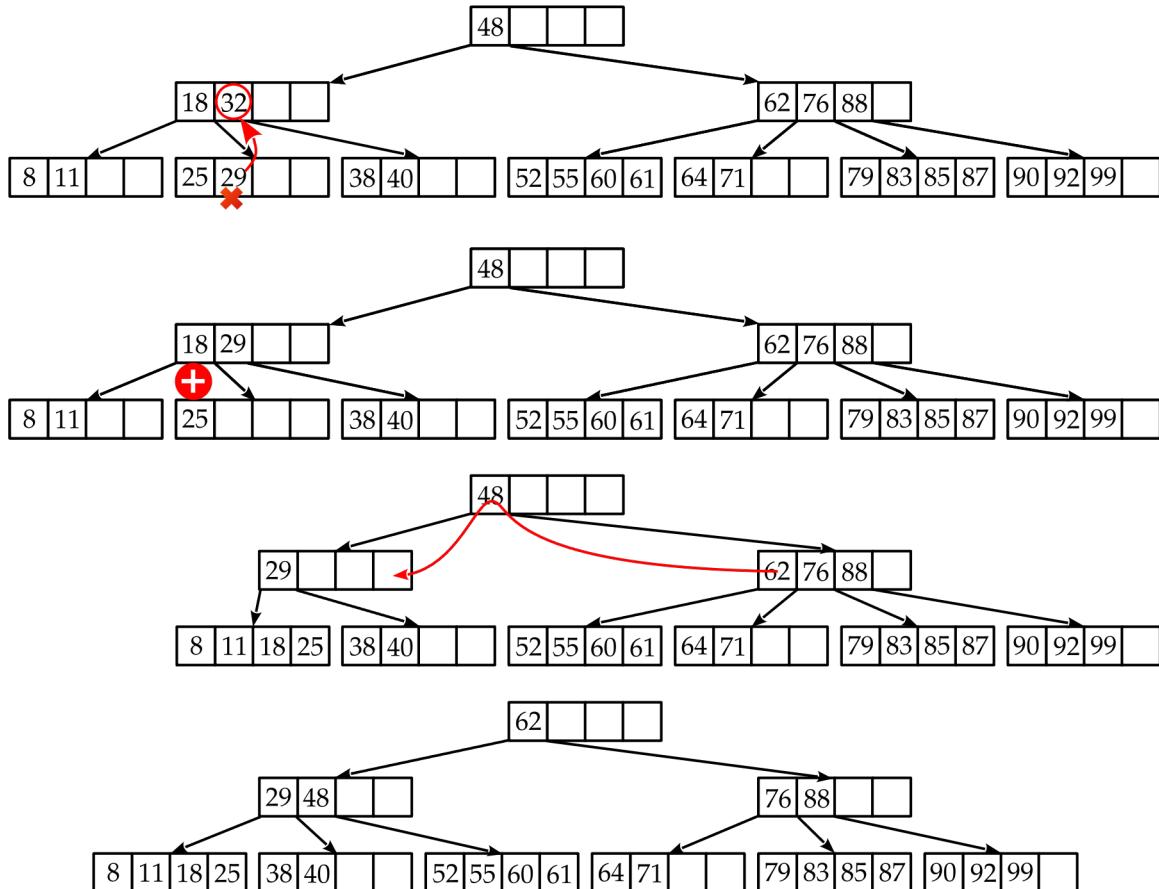
- **Primjer 2:** brišemo ključ 79 iz lista. List nakon brisanja nije više popunjen $\geq 50\%$ zato jer ima $< \lceil m/2 \rceil - 1$ ključeva.
- Gledamo li lijevog susjeda (blizanca), vidimo da je on popunjen $> \lceil m/2 \rceil - 1$
 - Radimo restrukturiranje tako da prebacujemo ključeve iz lijevog susjeda u čvor koji je u podljevu
 - Pri tome pazimo na zajednički ključ u čvoru roditelja

Primjer brisanja podataka iz B-stabla



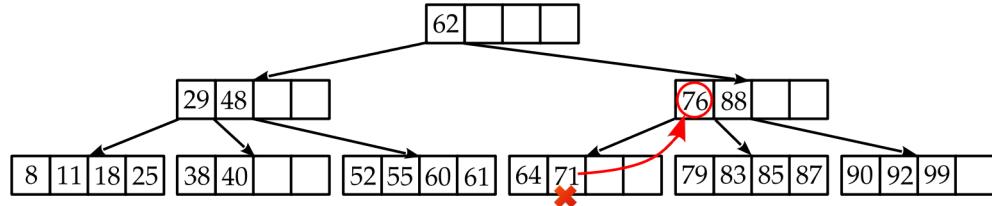
- **Primjer 3:** brišemo ključ **79** iz lista. List nakon brisanja nije više popunjen $\geq 50\%$ zato jer ima $< \lceil m/2 \rceil - 1$ ključeva.
- Gledamo li desnog susjeda (blizanca), vidimo da je on popunjen = $\lceil m/2 \rceil - 1$
 - Radimo spajanje desnog susjeda i čvora koji je u podljevu (*underflow*)
- Primijetimo da je sada roditeljski čvor u podljevu, što moramo riješiti rekurzivno

Primjer brisanja podataka iz B-stabla

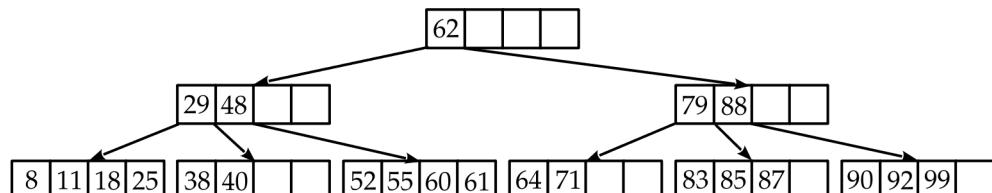
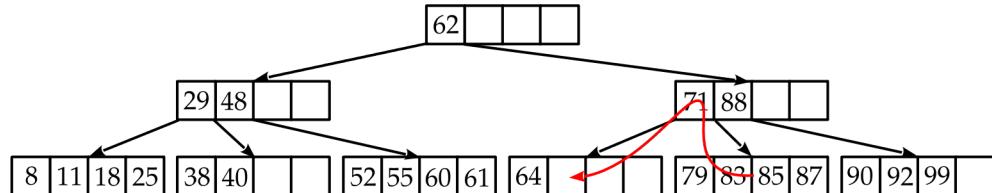


- Iz početnog B-stabla brišemo ključeve 32, 76, 48, 25 i 11
- Korak 1:** brišemo ključ 32 postupkom kopiranja
 - List u podljevu spajamo s lijevim susjedom
 - To uzrokuje podljev unutarnjeg čvora
 - Restrukturiramo desnog susjeda i unutarnji čvor u podljevu

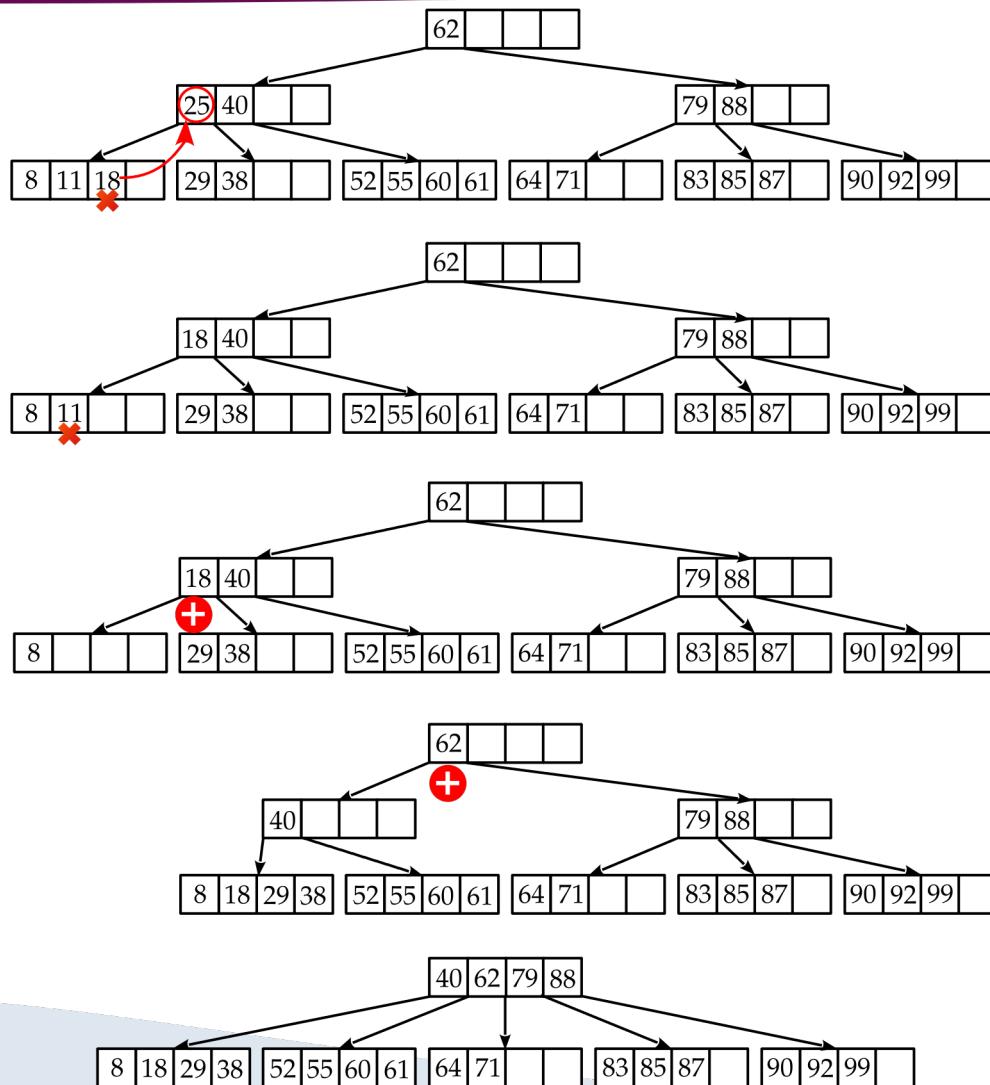
Primjer brisanja podataka iz B-stabla



- **Korak 2:** brišemo ključ **76** postupkom kopiranja. List u kojem smo obrisali zamjenski ključ je u podljevu.
 - Desni susjed do čvora u podljevu ima 100% popunjenošć
 - Restrukturiramo desnog susjeda i čvor u podljevu



Primjer brisanja podataka iz B-stabla



- **Korak 3:** brišemo ključ **25** postupkom kopiranja, a zatim brišemo ključ **11**. List u kojem smo obrisali 18 i 11 je sada u podljevu.
 - Desni susjed do čvora u podljevu ima točno 50% popunjenošć, te čvor u podljevu spajamo s desnim susjedom
 - Sada je unutarnji čvor u podljevu, a njegov desni susjed ima točno 50% popunjenošć, te čvor u podljevu spajamo s desnim susjedom
 - Prethodnim spajanjem nestaje stari korijenski čvor, a novi spojeni čvor postaje korijenski. Dubina stabla smanjuje se za 1.

Implementacija brisanja elementa u B-stablu

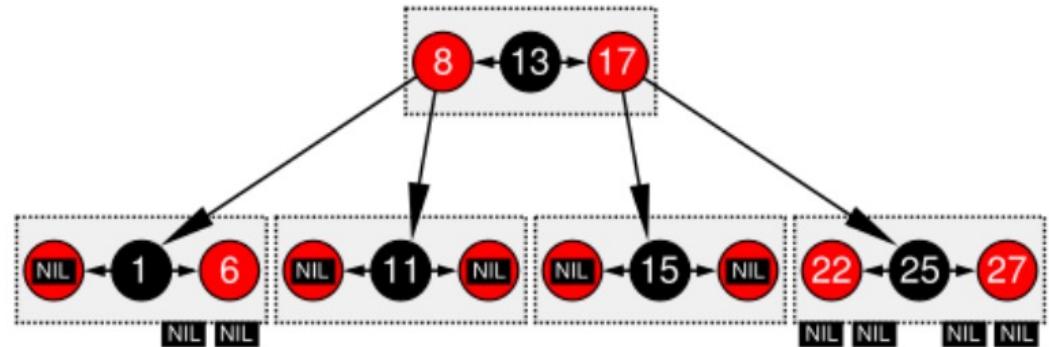
```
procedure BTREEREMOVAL(btree, val)
    ( $n_{rem}, n_v$ )  $\leftarrow$  BTREESEARCH(root of btree, val)
    if  $n_v$  is not nil then
        remove value val from the node  $n_{rem}$ 
        BTREEREMOVALCONSOLIDATION(btree, nrem)
```

```
procedure BTREEREMOVALCONSOLIDATION(btree, n)
    if  $|n| < \lceil \text{degree of } btree / 2 \rceil - 1$  then
        ps  $\leftarrow$  the predecessor sibling
         $n_{root} \leftarrow \text{nil}$ 
        if ps is not nil then
             $n_{mv} \leftarrow$  the shared parent value between ps and n
            if  $|ps| > \lceil \text{degree of } btree / 2 \rceil - 1$  then
                BTREEREDISTRIBUTE(btree, ps, nmv, n)
            else
                 $n_{root} \leftarrow$  BTREEMERGE(btree, ps, nmv, n)
        else
            ss  $\leftarrow$  the successor sibling
             $n_{mv} \leftarrow$  the shared parent value between ss and n
            if  $|ss| > \lceil \text{degree of } btree / 2 \rceil - 1$  then
                BTREEREDISTRIBUTE(btree, n, nmv, ss)
            else
                 $n_{root} \leftarrow$  BTREEMERGE(btree, n, nmv, ss)
        if  $n_{root}$  is nil and parent of n exists then
            BTREEREMOVALCONSOLIDATION(btree, parent of n)
```

Crveno-crna stabla

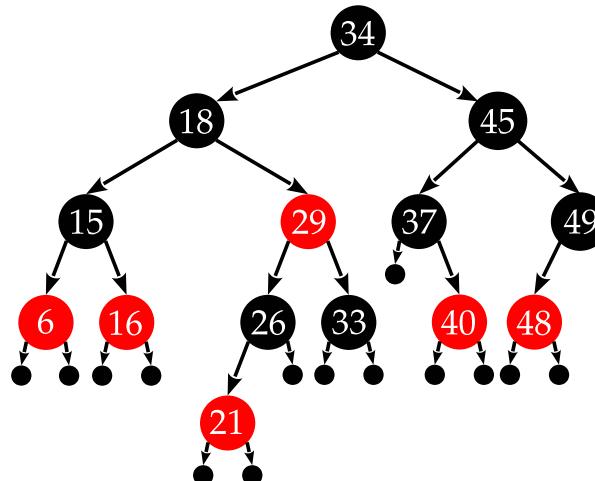
Crveno-crno stablo (*Red-black tree*)

- Binarno stablo koje **idejno** proizlazi iz B-stabla 4. reda ako mu se elementi čvorova smatraju obojanima prema strogim pravilima
- Usporedba s B-stablim:
 - manji utrošak memorije
 - zadržava uravnoteženost
 - složenost ista



Definicijska pravila

1. svaki čvor je **crven** ili **crn**
2. korijen je **crn** (neobavezno, ali uobičajeno)
3. svaki list* je **crn**
4. oba potomka **crvenog** čvora su **crna**
5. svaka staza od nekog čvora do (bilo kojeg) lista koji je njegov potomak prolazi istim brojem crnih čvorova



- *Listovi u crveno-crnom (RB) stablu ne sadrže informacije pa ne moraju ni postojati, nego roditelji mogu imati NULL pokazivače ili svi pokazivati isti poseban čvor, sentinel

Crvena i crna visina stabla

- Razlikujemo **crvenu** i **crnu** visinu stabla:
 - $rh(x)$, $bh(x)$
 - broj čvorova određene boje na putu od čvora x do lista koji mu je potomak (x se nebroji).
- Ključno svojstvo za uravnoteženost RB stabla:
 - najduži put od korijena do nekog lista najviše je dvostruko duži od najkraćeg puta od korijena do nekog (drugog) lista
 - tj. najduži put je najviše dvostruko duži od najkraćeg.

Teorem

- Visina RB-stabla s n unutarnjih čvorova je

$$h \leq 2\log_2(n + 1)$$

Dokaz:

Binarno stablo visine h ima najviše $n = 2^h - 1$ čvorova
Zbog 4. pravila, barem polovica visine je crna visina pa je
 $hb \geq h/2$. Budući da je n veći ili jednak broju crnih
čvorova na putu od korijena do najnižeg lista, slijedi:

$$n \geq 2^{hb} - 1 \geq 2^{\frac{h}{2}} - 1, \text{ a iz toga izravno}$$

$$h \leq 2\log_2(n + 1)$$

- Pretraživanje binarnog stabla je složenosti $O(h)$ pa je složenost pretraživanja RB-stabla $O(\log_2 n)$

Dodavanje čvora u RB-stablo

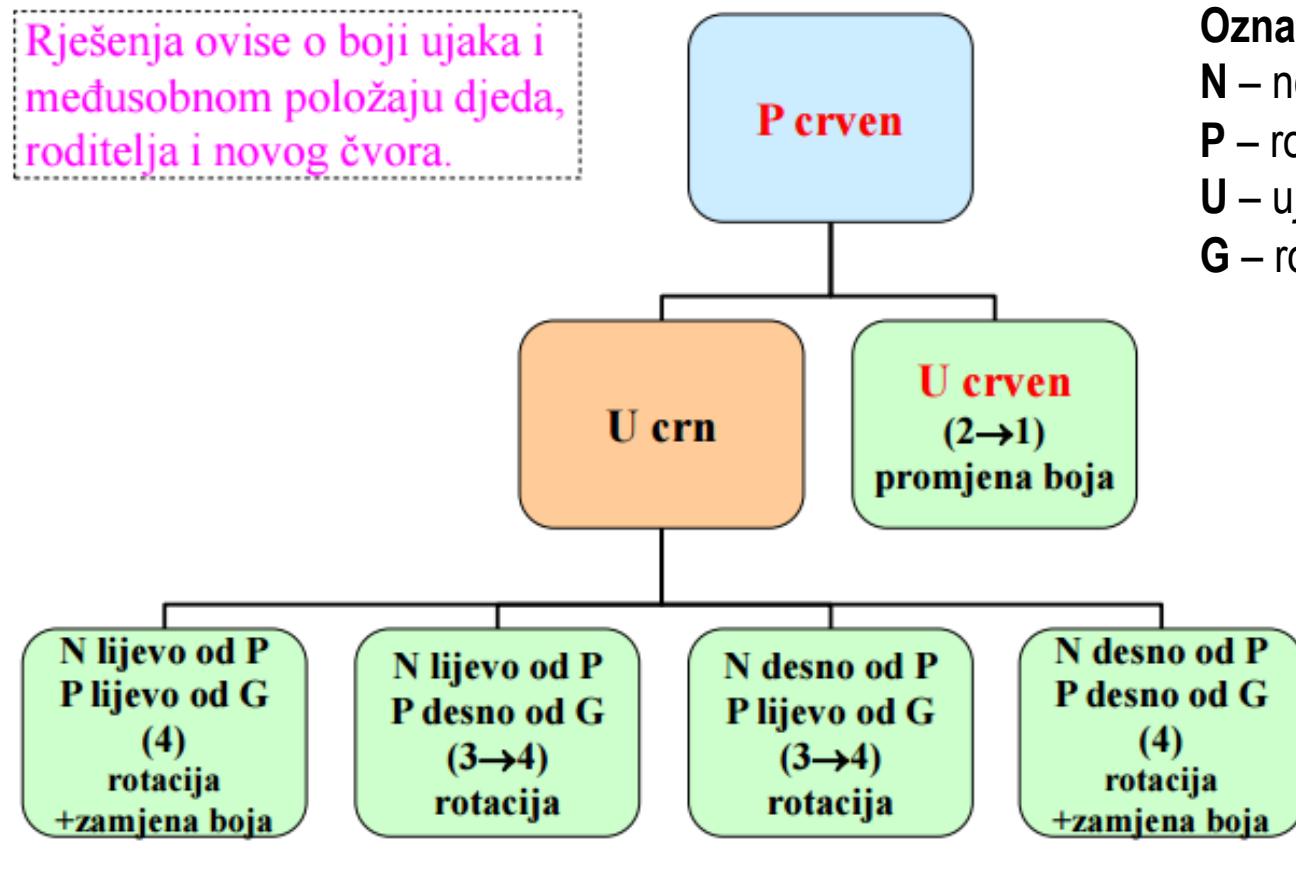
- Radi lakše analize, uvode se pojmovi
 - čvor-ujak (uncle) koji se označava s **U**, a znači blizanac roditelja promatranog čvora (roditeljev brat/sestra)
 - čvor-djed koji se označava s **G** (grandparent), a znači roditelj roditelja
1. ubaciti novi čvor kao u svako drugo binarno search stablo i pridijeliti mu **crvenu** boju
 2. restrukturirati stablo (primjenom rotacija i bojanjem čvorova) da bi zadovoljilo definicijska pravila

Dodavanje čvora u RB-stablo

- Definicijska pravila 1, 3 i 5 su uvijek zadovoljena kod dodavanja novog čvora, a 2 i 4 mogu biti ugrožena (ne istodobno) na sljedeće načine:
 - pravilo 2 ako je novi čvor korijen
 - pravilo 4 ako je roditelj novog čvora **crven**
 - U oba slučaja potrebno je restrukturiranje
- Restrukturiranje:
 1. novi čvor je korijen:
 - prebojati ga u **crno** (5. pravilo ostaje zadovoljeno jer je to dodatni crni čvor u svim putevima u stablu)

Dodavanje čvora u RB-stablo

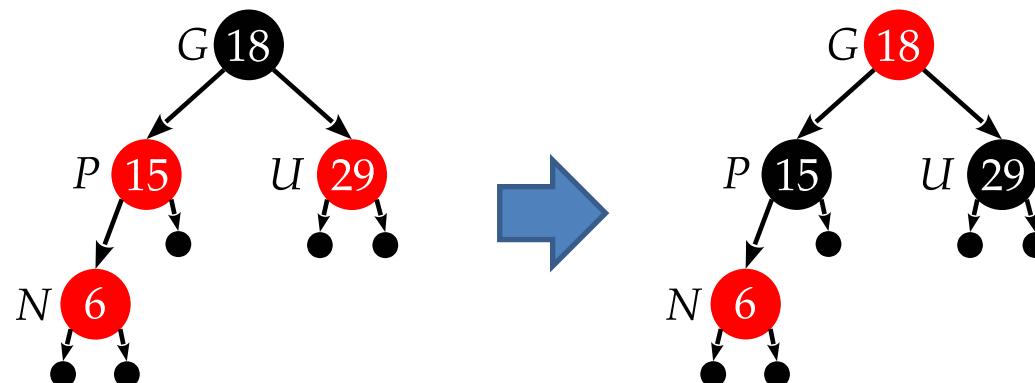
- Restrukturiranje:
 2. roditelj novog čvora je **crven** (korak 1):



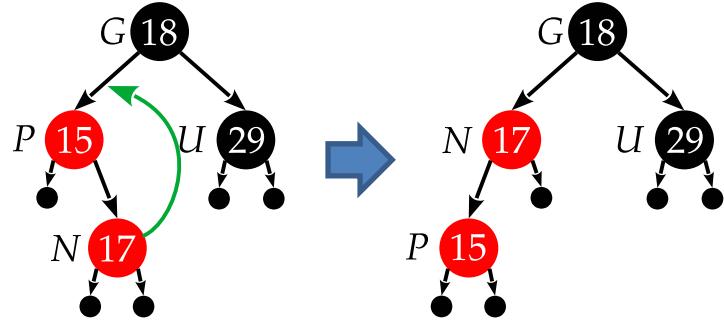
Dodavanje čvora u RB-stablo

2. Roditelj i ujak su crveni

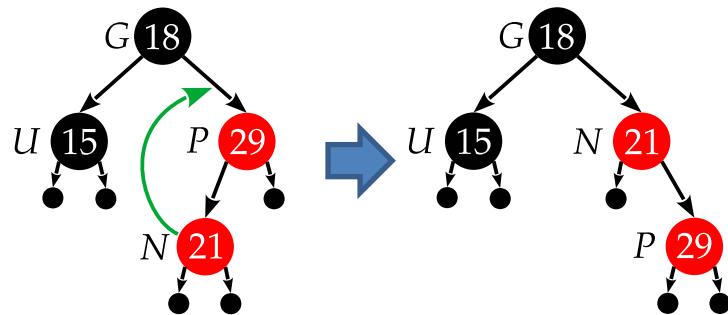
- Narušeno 4. pravilo (nanizana dva crvena; P i N)
 - prebojati P i U u crno (rješava 4. pravilo), a G u crveno (očuvanje 5. pravila) - sada G može narušavati 4. pravilo ako ima crvenog roditelja ili 2. pravilo ako je korijen
 - **Nastaviti** s provjerom promatrajući G kao novi čvor (N)



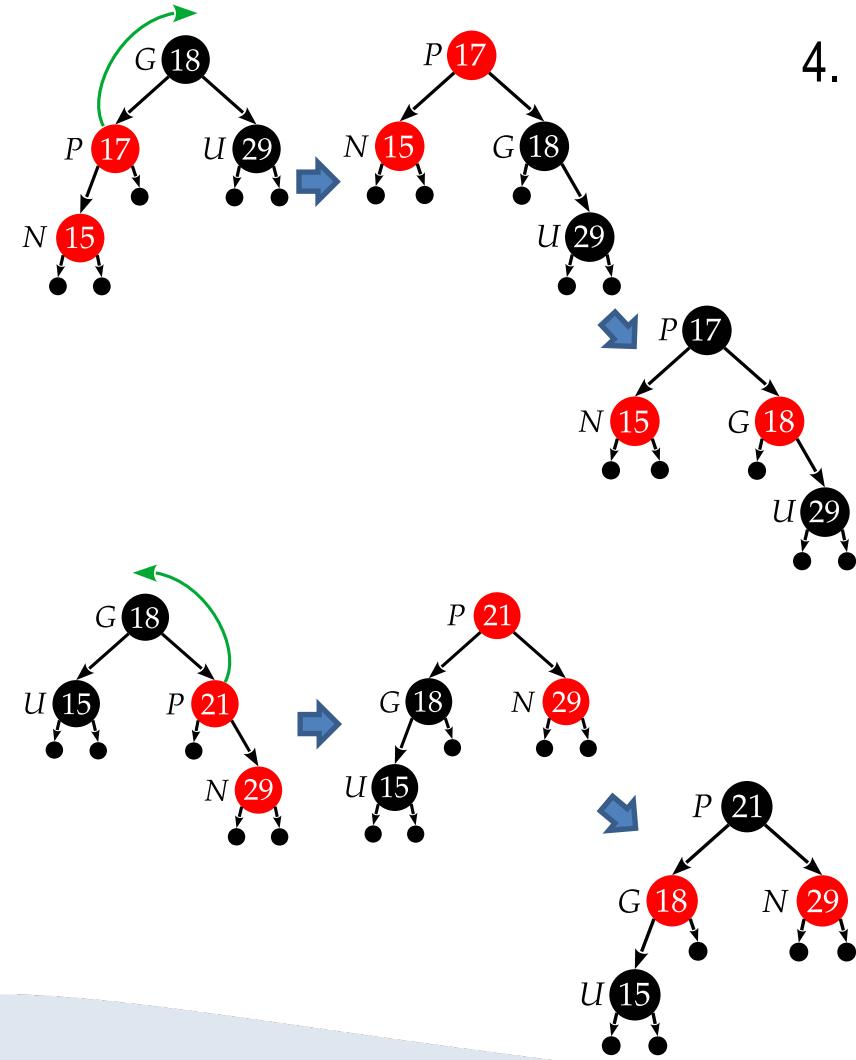
Dodavanje čvora u RB-stablo



3. Roditelj **crven** i ujak crn (“izlomljeni” poredak N, P i G)
- Dva simetrična slučaja:
 - N desno dijete od P i P lijevo dijete od G
 - N lijevo dijete od P i P desno dijete od G
 - Rješenje:
 - rotacija N oko P, čime se stanje prevodi u “izravnati poredak” N, P i G koji se rješava u 4. provjeri
 - **Nastaviti** s provjerom (4), pridajući P-u ulogu N-a



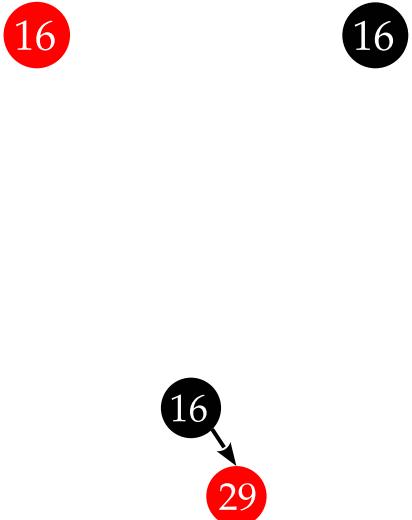
Dodavanje čvora u RB-stablo



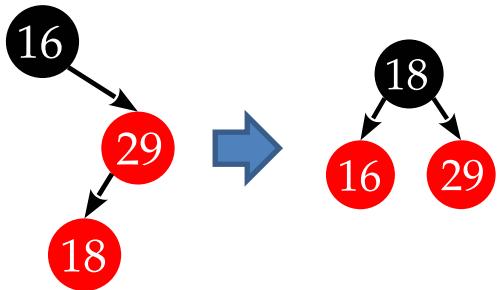
4. Roditelj **crven** i ujak crn (“linijski” poredak N, P i G)
- Dva simetrična slučaja:
 - N lijevo dijete od P i P lijevo dijete od G
 - N desno dijete od P i P desno dijete od G
 - Rješenje:
 - *rotacija* P oko G
 - *zamjena boja* P i G (znamo da je G crn jer u protivnom P ne bi mogao biti **crven**); **KRAJ**

Primjer dodavanja podataka u RB-stablo

- Dodajemo redom ključeve: **16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40**
- **Korak 1:** Formiramo korijenski čvor s prvim ključem **16**. Nakon dodavanja, čvor je crveni, pa ga pretvorimo u crni (pravilo 2).
- **Korak 2:** U stablo dodajemo ključ **29**. Nema restrukturiranja RB-stabla.



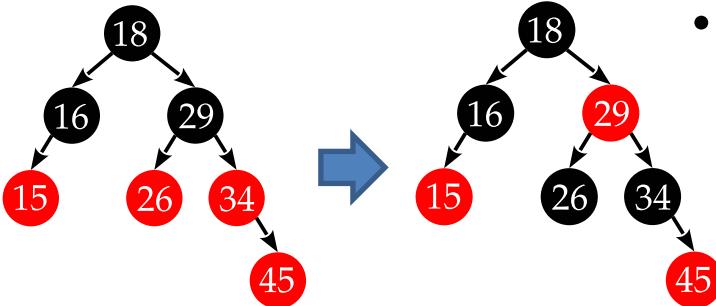
Primjer dodavanja podataka u RB-stablo



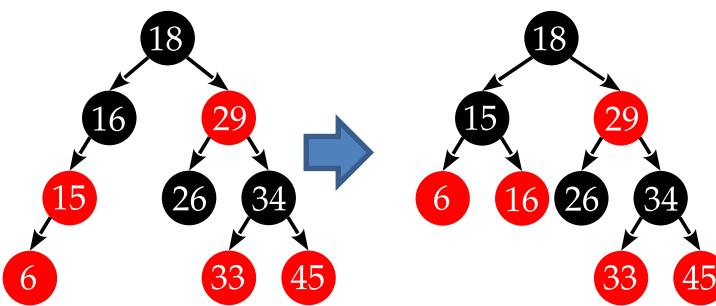
- **Korak 3:** U stablo dodajemo ključ 18.
 - **Slučaj 3:** Desna rotacija 18 oko 29
 - **Slučaj 4:** Lijeva rotacija 18 oko 16 + zamjena boja.

- **Korak 4:** U stablo dodajemo ključ 34.
 - **Slučaj 2:** Postavi 18 u crveno, a 16 i 29 u crno
 - **Slučaj 1:** Postavi korijen 18 u crno

Primjer dodavanja podataka u RB-stablo

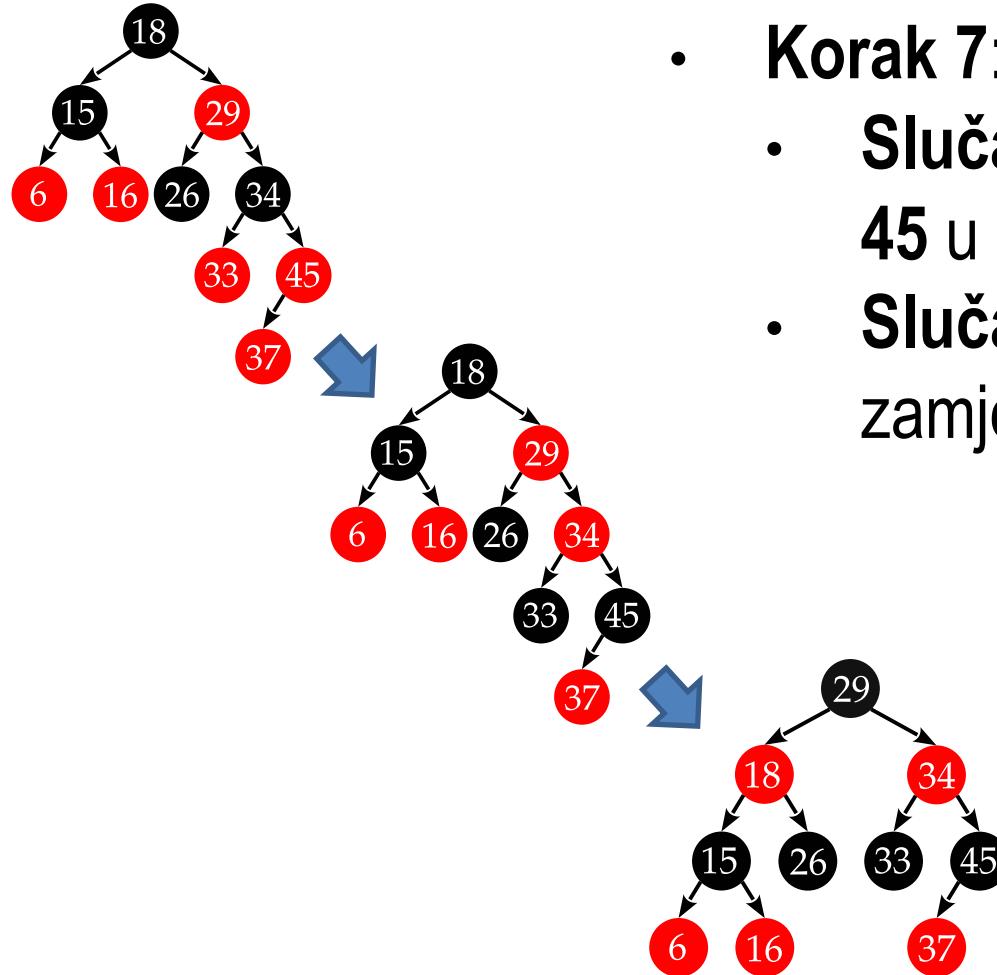


- **Korak 5:** U stablo dodajemo ključeve **26**, **15** i **45**.
 - Nakon dodavanja **45** imamo **slučaj 2**: Postavi **29** u crveno, a **26** i **34** u crno.



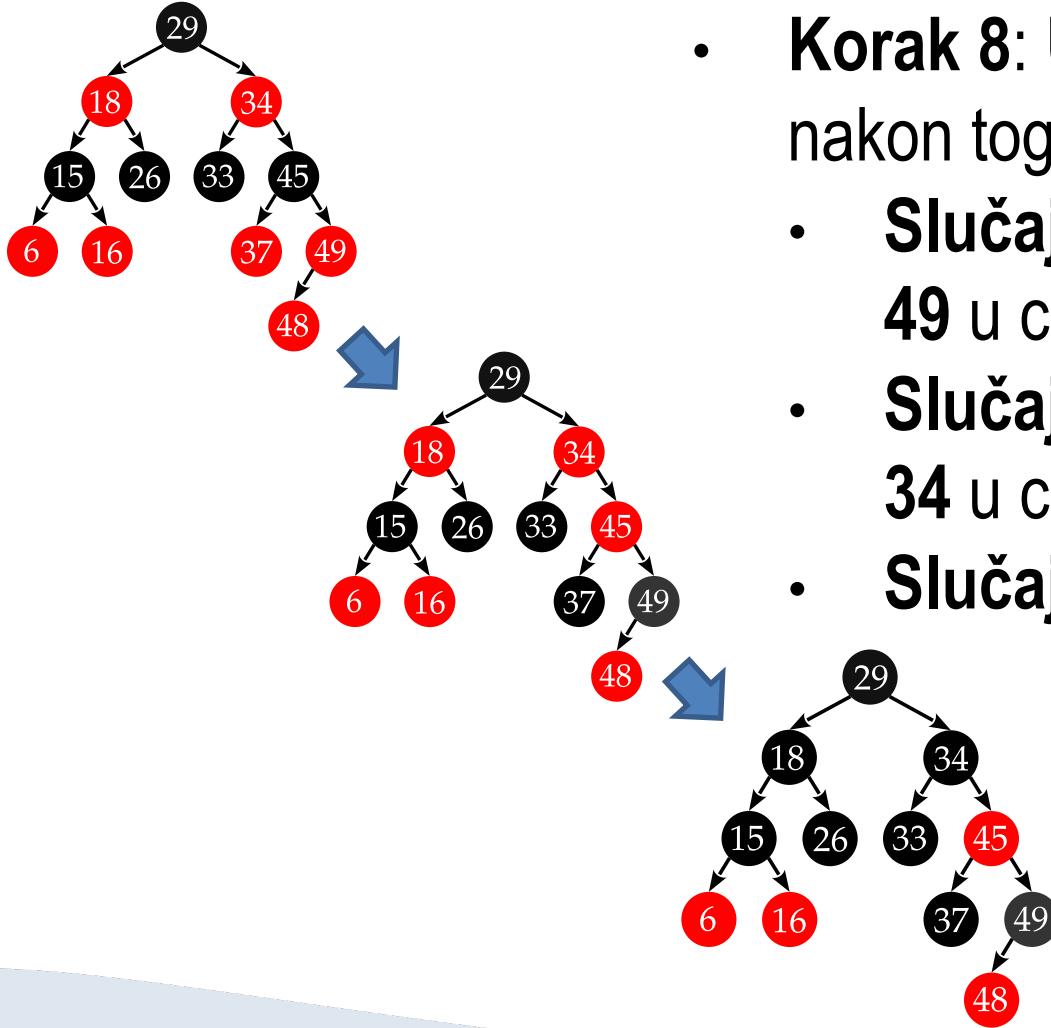
- **Korak 6:** U stablo dodajemo ključeve **33** i **6**.
 - Nakon dodavanja **6** imamo **slučaj 4**: desna rotacija **15** oko **16** i zamjena boja između **15** i **16**

Primjer dodavanja podataka u RB-stablo



- **Korak 7:** U stablo dodajemo ključ **37**.
 - **Slučaj 2:** Postavi **34** u crveno, a **33** i **45** u crno.
 - **Slučaj 4:** Lijeva rotacija **29** oko **18** i zamjena boja **18** i **29**.

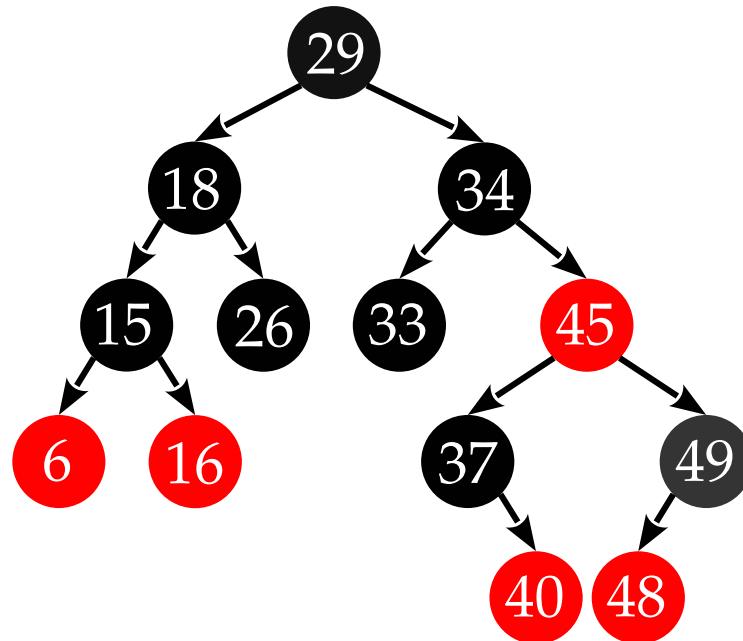
Primjer dodavanja podataka u RB-stablo



- **Korak 8:** U stablo dodajemo ključ **49**, te nakon toga ključ **48**.
 - **Slučaj 2:** Postavi **45** u crveno, a **37** i **49** u crno.
 - **Slučaj 2:** Postavi **29** u crveno, a **18** i **34** u crno.
 - **Slučaj 1:** Postavi korijen **29** u crno.

Primjer dodavanja podataka u RB-stablo

- **Korak 9:** U stablo dodajemo ključ 40



16, 29, 18, 34, 26, 15, 45, 33, 6, 37, 49, 48, 40

Implementacija dodavanja čvora u RB-stablo

```
procedure RBTREEINSERT(rbtree, N)
    P  $\leftarrow$  parent(N)
    G  $\leftarrow$  parent(P)
    while P is  $\bullet$  do
        if P is the left child then
            case  $\leftarrow$  LL
            if N is the right child then
                case  $\leftarrow$  LR
                U  $\leftarrow$  the right child of G
            else
                case  $\leftarrow$  RR
                if N is the left child then
                    case  $\leftarrow$  RL
                    U  $\leftarrow$  the left child of G
                if U and P are  $\bullet$  then
                    P  $\leftarrow$  U  $\leftarrow$   $\bullet$ 
                    G  $\leftarrow$   $\bullet$ 
                    N  $\leftarrow$  G
                else if P is  $\bullet$  and U is  $\bullet$  then
                    if case  $\in \{LL, RR\}$  then            $\triangleright$  straight cases
                        if case is LL then
                            right rotate P around G
                        else
                            left rotate P around G
                            switch P and G colors
                            break
                    else                                $\triangleright$  broken cases
                        if case is LR then
                            right rotate N around P
                        else
                            left rotate N around P
                            N  $\leftarrow$  P
                    P  $\leftarrow$  parent(N)
                    G  $\leftarrow$  parent(P)
                    root(rbtree)  $\leftarrow$   $\bullet$ 
                
```

▷ Rule 2

Brisanje čvora u RB-stablu

- Algoritam:
 1. Brisanje kopiranjem (zamjenski čvor; u nastavku oznaka X)
 2. Ukloniti zamjenski čvor; on može imati najviše jedno dijete pa je problem pojednostavljen
- Ako je zamjenski čvor:
 - **crven**: svojstva RB-stabla nisu narušena, postupak je gotov
 - **crn**: složeniji postupak

Uklanjanje crnog čvora

- 3 su moguća problema nakon uklanjanja crnog čvora:
 1. ako je uklonjen korijen, mogao je imati samo jedno dijete (N) koje postaje novi korijen, a ono može biti i **crveno**
 - povreda 2. pravila (korijen je crn)
 2. nakon uklanjanja X, njegovo dijete N i roditelj P su u odnosu dijete-roditelj i ako su oboje **crveni**
 - povreda 4. pravila (djeca **crvenog** su crna)
 3. uklanjanje crnog X znači smanjenje crne visine svih njegovih prethodnika (predaka)
 - povreda 5. pravila
- Za prvi slučaj dovoljno je prebojati N u crno i sve je riješeno jer se mijenjanjem boje korijena jednako mijenja crna visina svim čvorovima stabla. Ostala dva slučaja ovise o boji čvora N.

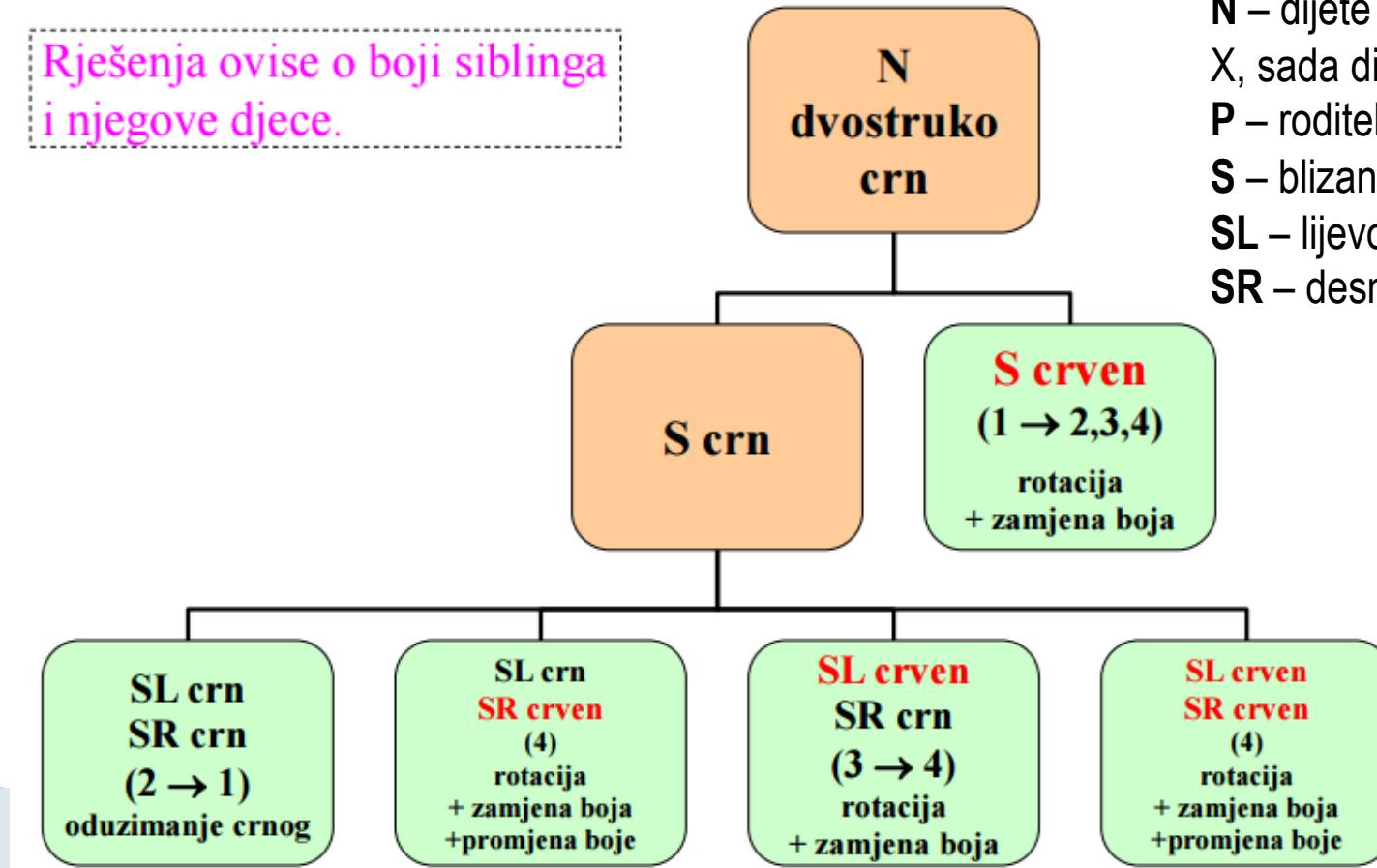
Uklanjanje crnog čvora

- Zamislimo da možemo nekako prenijeti crninu X-a na N. Tada ju uklanjanjem X ne bismo izgubili i RB pravila ne bi bila prekršena:
 - Ako je N prethodno bio **crven**, postat će crveno-crn i crnoj visini doprinositi 1.
 - Ako je N prethodno bio crn, postat će dvostruko crn i crnoj visini doprinositi 2.
- Rješenje:
 - Ako je crveno-crn, dovoljno je prebojati ga u čisto crno.
 - Ako je dvostruko crn, ideja je proslijediti višak crnog prethodniku i tako taj višak podizati sve dok ne dođe na mjesto gdje ga možemo trajno ugraditi u stablo ili dok ne dođe u korijen gdje ga možemo zanemariti.

Uklanjanje crnog čvora (dvostruko crn)

- 4 (+4 simetrična) su moguća slučaja, a ovise o boji čvora blizanca (dijete istog roditelja kao i N) i njegove djece

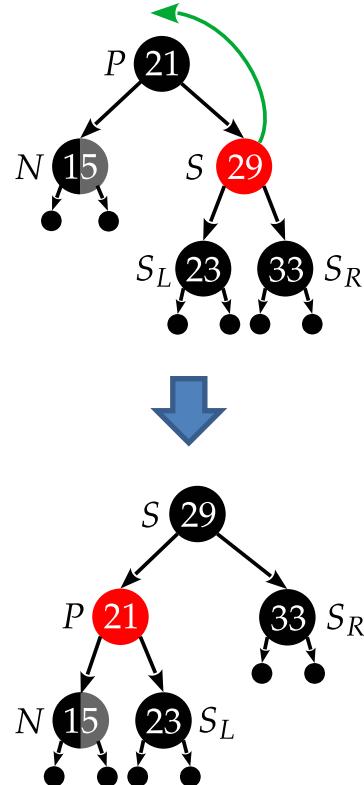
Rješenja ovise o boji siblinga
i njegove djece.



Oznake:

N – dijete od uklonjenog
X, sada dijete od P
P – roditelj od N
S – blizanac od N
SL – lijevo dijete od S
SR – desno dijete od S

Uklanjanje crnog čvora (dvostruko crn)



1. Blizanac S je **crven**

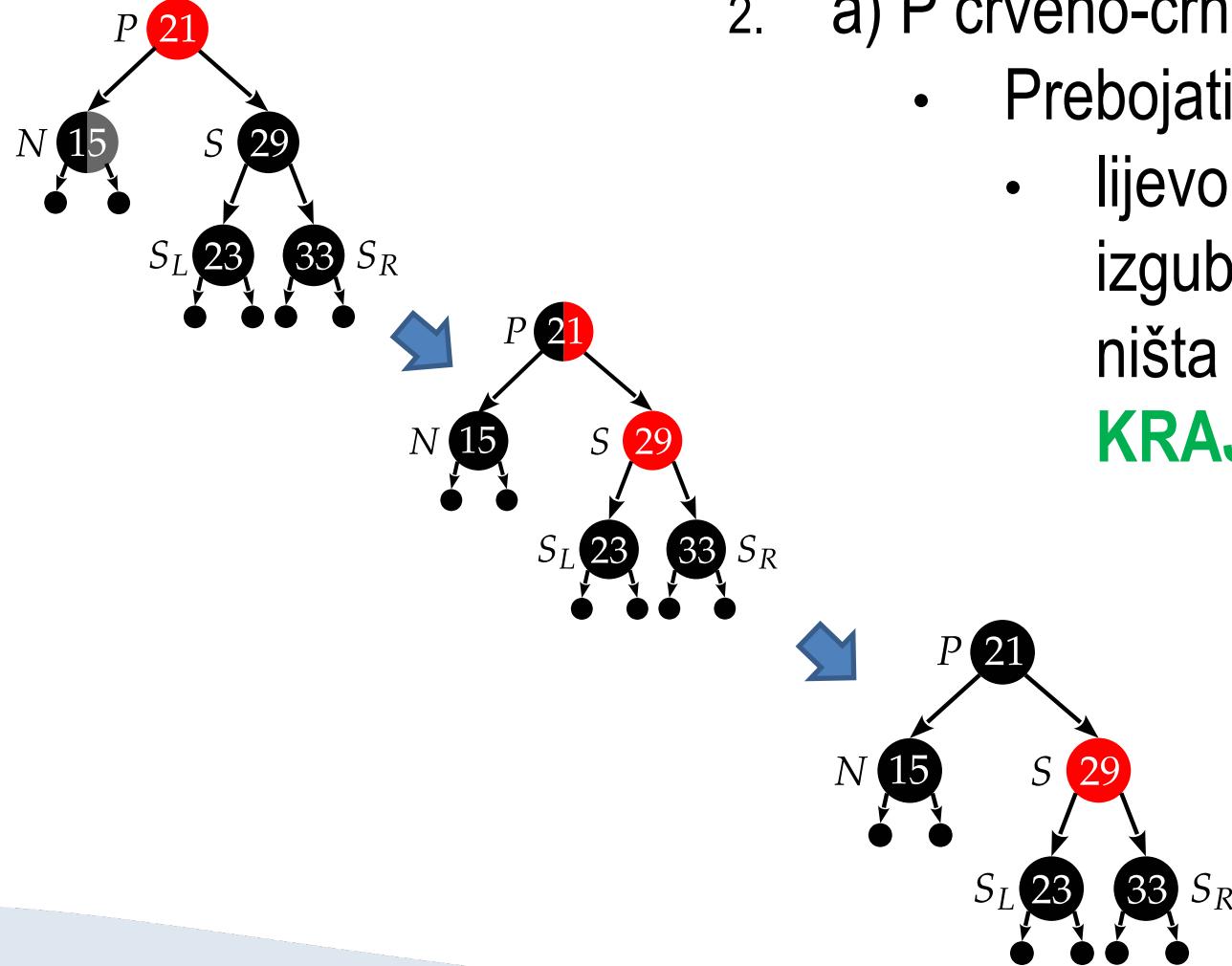
- P je sigurno crn jer ima **crveno** dijete
- Nakon brisanja X crna visina lijevog podstabla od P za jedan je manja od crne visine desnog podstabla (tj. N dvostruko crn)
- Rješenje: rotirati S oko P (simetrija) pa zamijeniti boje P i S
- **NASTAVAK** uravnotežavanja iz N

Uklanjanje crnog čvora (dvostruko crn)

2. S crn, djeca od S crna

- Oduzeti jedno crno N-u i S-u; N ostaje jednostruko crn, a S postaje **crven**
- Taj višak crnoga proslijediti višoj razini (konvergencija!), tj. P-u koji time postaje ili crveno-crn ili dvostruko crn
- O P-u ovisi postupanje nakon intervencije (slučajevi 2a i 2b):

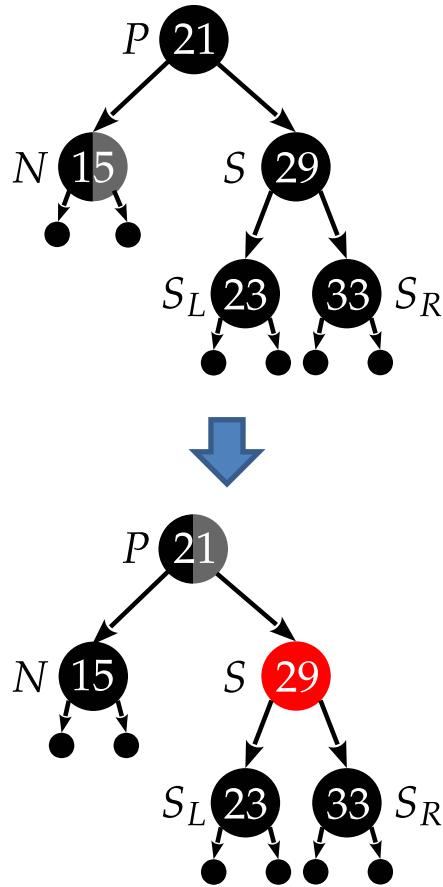
Uklanjanje crnog čvora (dvostruko crn)



2. a) P crveno-crn

- Prebojati P u crno
- lijevo podstablo time dobiva izgubljeno crno, a desnom se ništa ne mijenja jer je S **crven**;
KRAJ

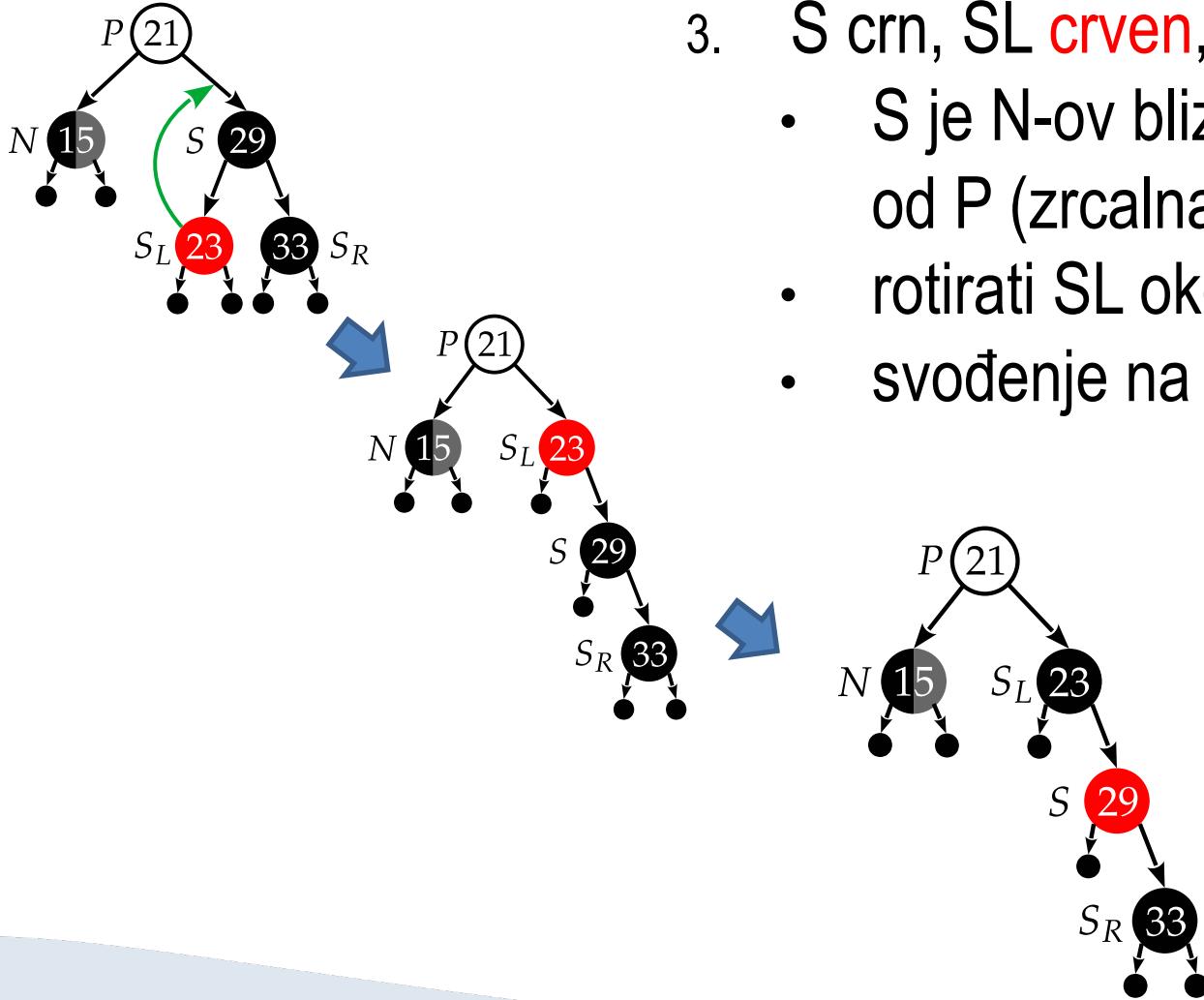
Uklanjanje crnog čvora (dvostruko crn)



2. b) P dvostruko crn

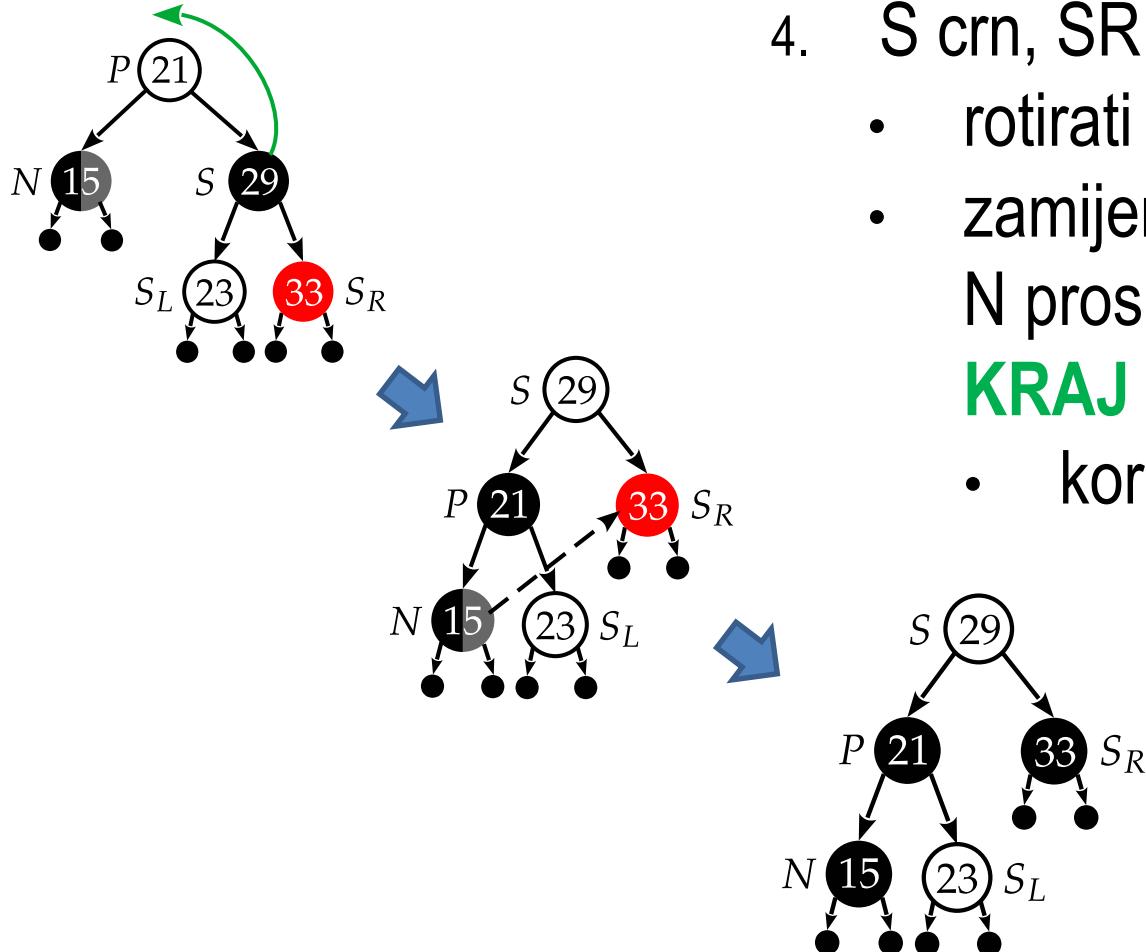
- P je korijen
 - višak crnog se odbacuje; **KRAJ**
 - P nije korijen
 - natrag na slučaj 1 promatrajući P kao N; **NASTAVAK**
 - problem je razinu više (konvergencija!)

Uklanjanje crnog čvora (dvostruko crn)



3. S crn, SL crven, SR crn, P nevažan
- S je N-ov blizanac, a N je lijevo dijete od P (zrcalna simetrija!)
 - rotirati SL oko S i zamijeniti im boje
 - svodjenje na slučaj 4; **NASTAVAK**

Uklanjanje crnog čvora (dvostruko crn)



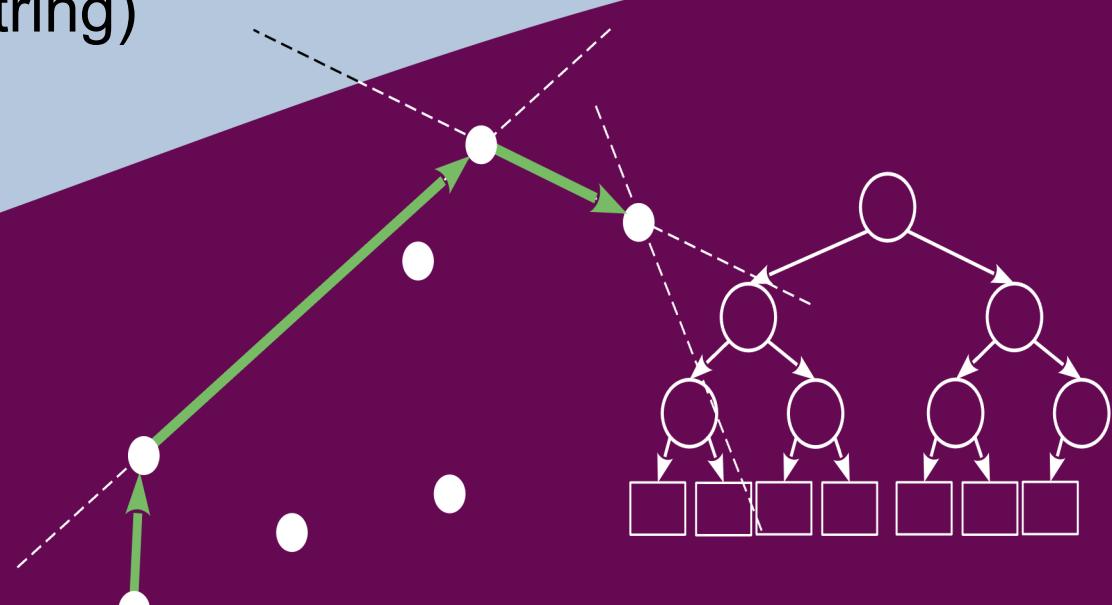
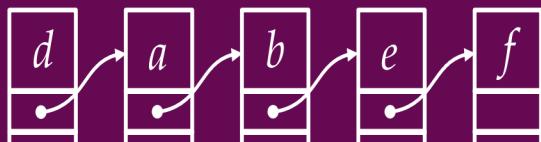
4. S crn, SR **crven**, P i SL nevažni
- rotirati S oko P (zrcalna simetrija!)
 - zamijeniti boje S i P, a višak crnog iz N proslijediti u SR (prebojati u crno);
- KRAJ**
- korijen podstabla ostaje iste boje

Implementacija brisanja čvora u RB-stablu

```
procedure RBTREEREMOVE(rbtree, N)
    while  $N$  is not  $\text{root}(rbtree)$  and  $N$  is  $\bullet$  do
         $P \leftarrow$  the parent of  $N$ 
        if  $N$  is the left child of  $P$  then
             $S \leftarrow$  the right child of  $P$ 
             $S_L, S_R \leftarrow$  children of  $S$ 
            if  $S$  is  $\bullet$  then
                 $S \leftarrow \bullet$ 
                 $P \leftarrow \bullet$ 
                left rotate  $S$  around  $P$ 
            if  $S_L$  is  $\bullet$  and  $S_R$  is  $\bullet$  then
                 $S \leftarrow \bullet$ 
                 $N \leftarrow$  the parent of  $N$ 
            else
                if  $S_R$  is  $\bullet$  then
                     $S_L \leftarrow \bullet$ 
                     $S \leftarrow \bullet$ 
                    right rotate  $S_L$  around  $S$ 
                    color of  $S \leftarrow$  color of  $P$ 
                     $P \leftarrow S_R \leftarrow \bullet$ 
                    left rotate  $S$  around  $P$ 
                     $N \leftarrow \text{root}(rbtree)$ 
                else                                 $\triangleright$  implement the symmetrical cases
                     $N \leftarrow \bullet$ 
```

Napredni algoritmi i strukture podataka

Tjedan 3: Strukture podataka za znakovne nizove (string)



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 daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

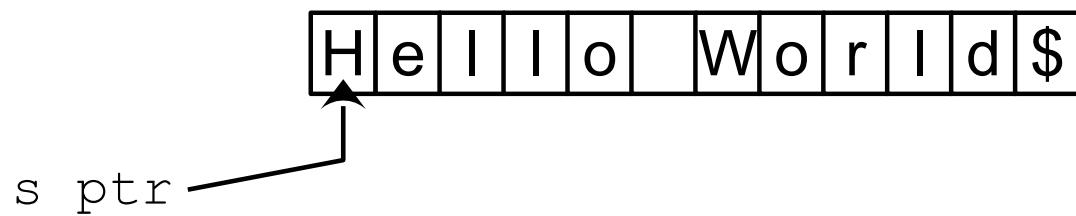
Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

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

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

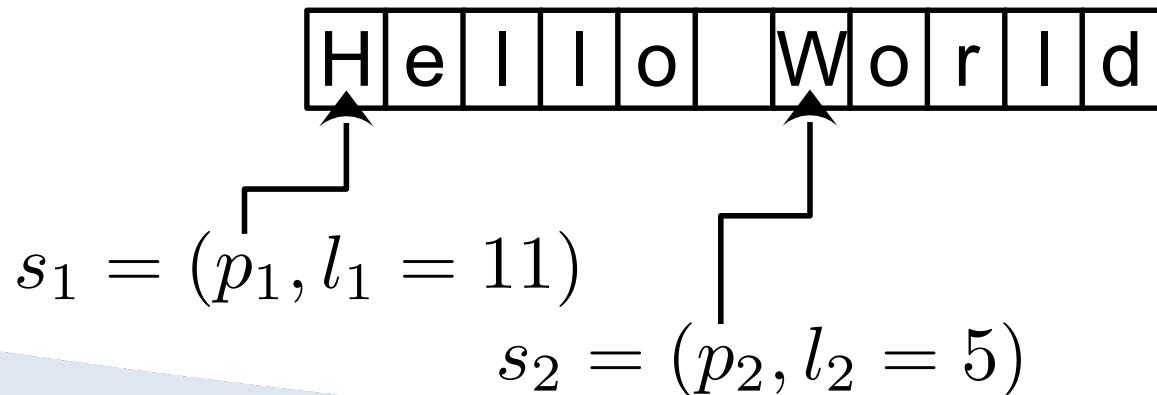
Reprezentacija znakovnog niza (1)

- 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 ovaku reprezentaciju nije nužno korištenje NULL terminatorsa



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 (rijeci, 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

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

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
 - Korijenski 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

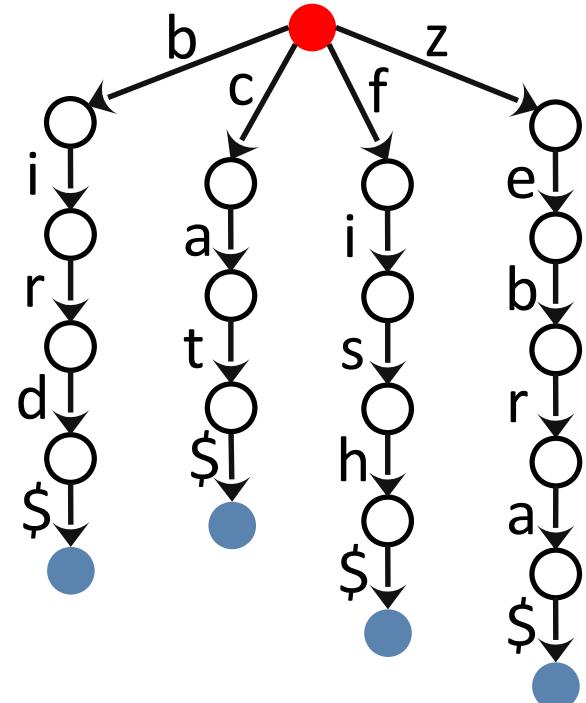
Prefiksno stablo (Trie) (3)

- Primjer

$$S_1 = \{"bird\$", "cat$", "fish$", "zebra$\"}$$
$$\Sigma = \{\$, a, b, c, d, e, f, h, i, r, s, t, z\}$$

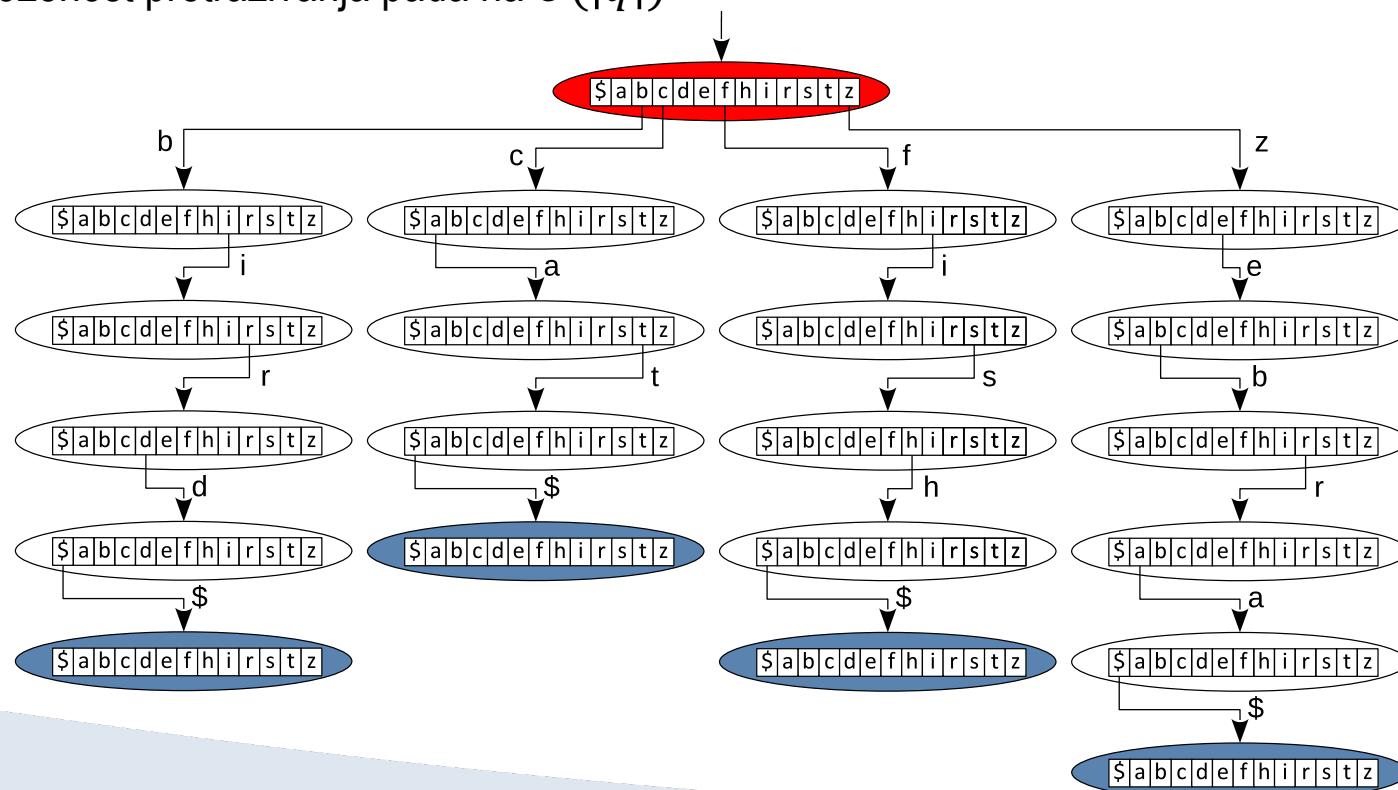
- Problem u ovakovom stablu je pretraživanje izlaznih bridova

- 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 $O(|q|)$



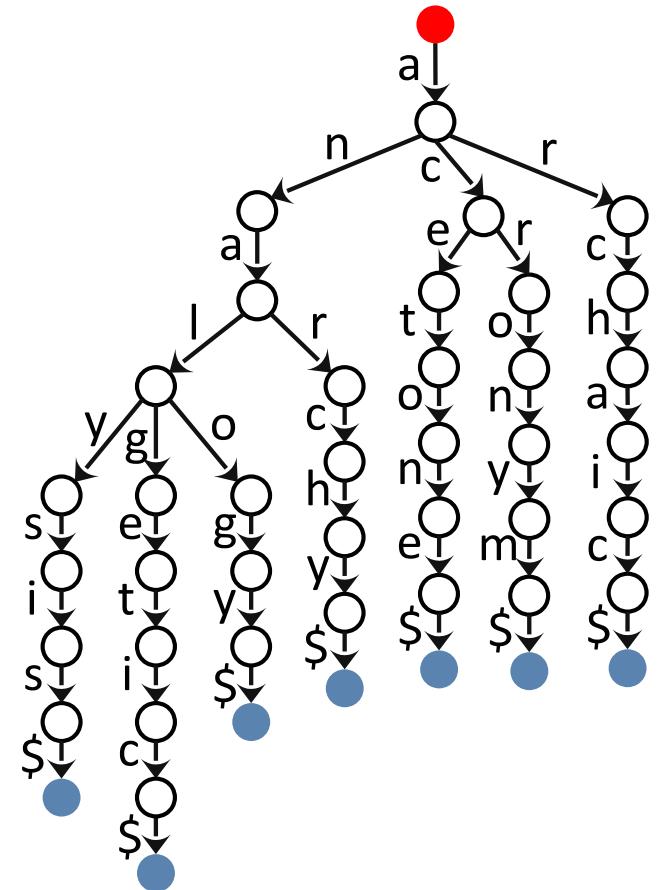
Prefiksno stablo (Trie) (5)

- Mnogo znakovnih nizova u S može dijeliti zajednički prefiks

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

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 prefksi znakovnih nizova dijele strukturu stabla



Prefiksno stablo (Trie) (6)

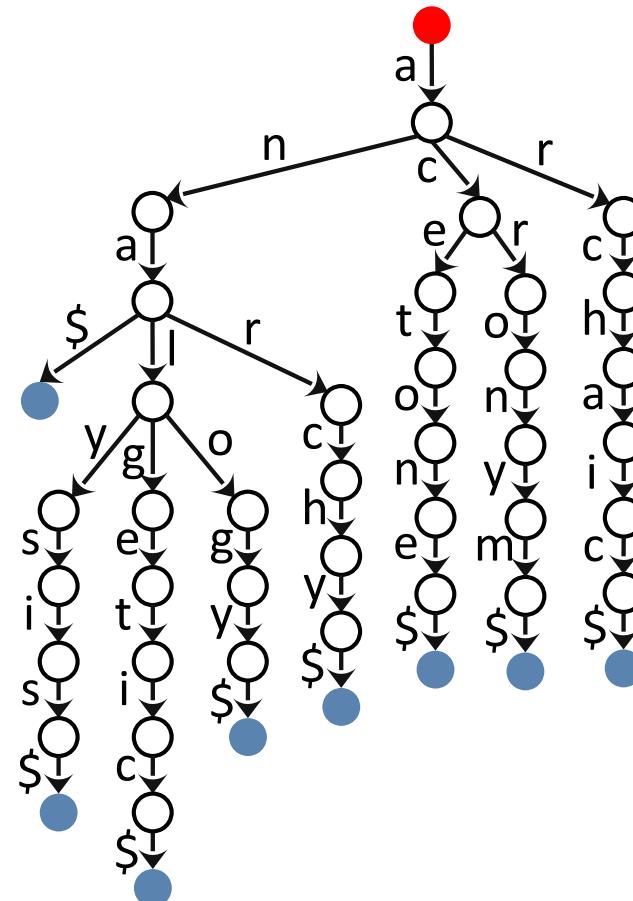
- Pretraživanje

- 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 $q = "ana\$"$ u prethodnom Trie-u ?

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

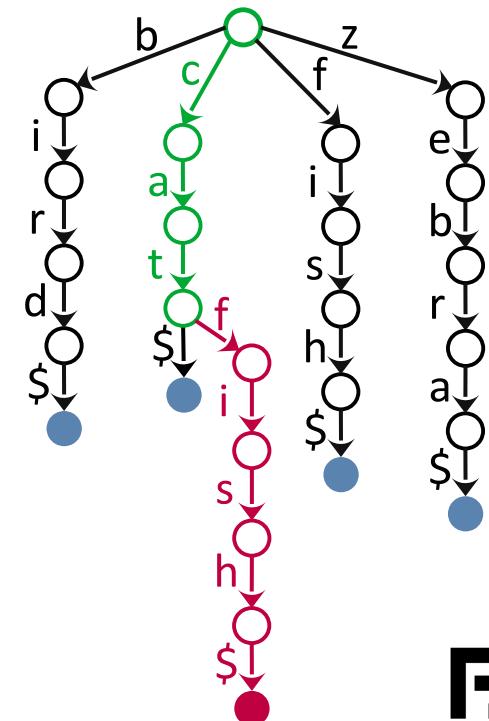
- Primjer $S_3 = S_2 \cup \{"ana\$"\}$



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 $s = \text{„catfish\$”}$

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



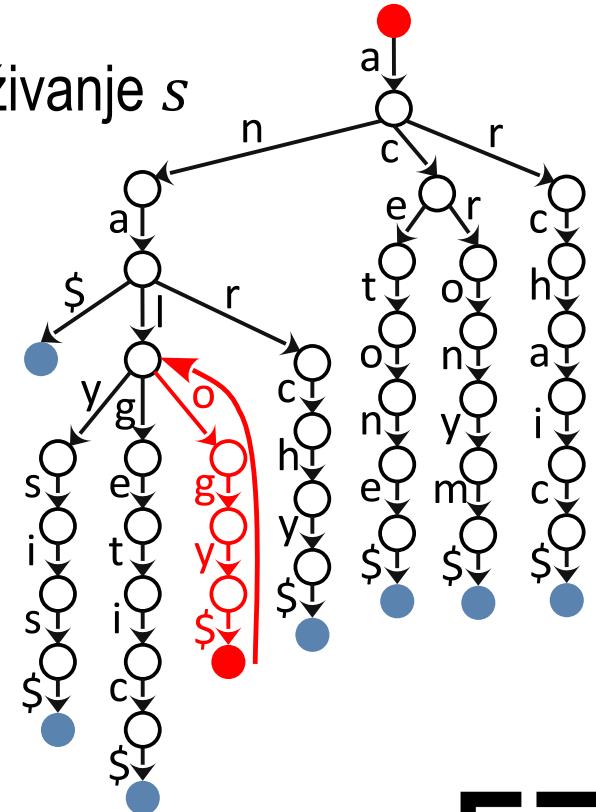
Prefiksno stablo (Trie) (9)

- 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 $s = \text{„analogy$”}$

```

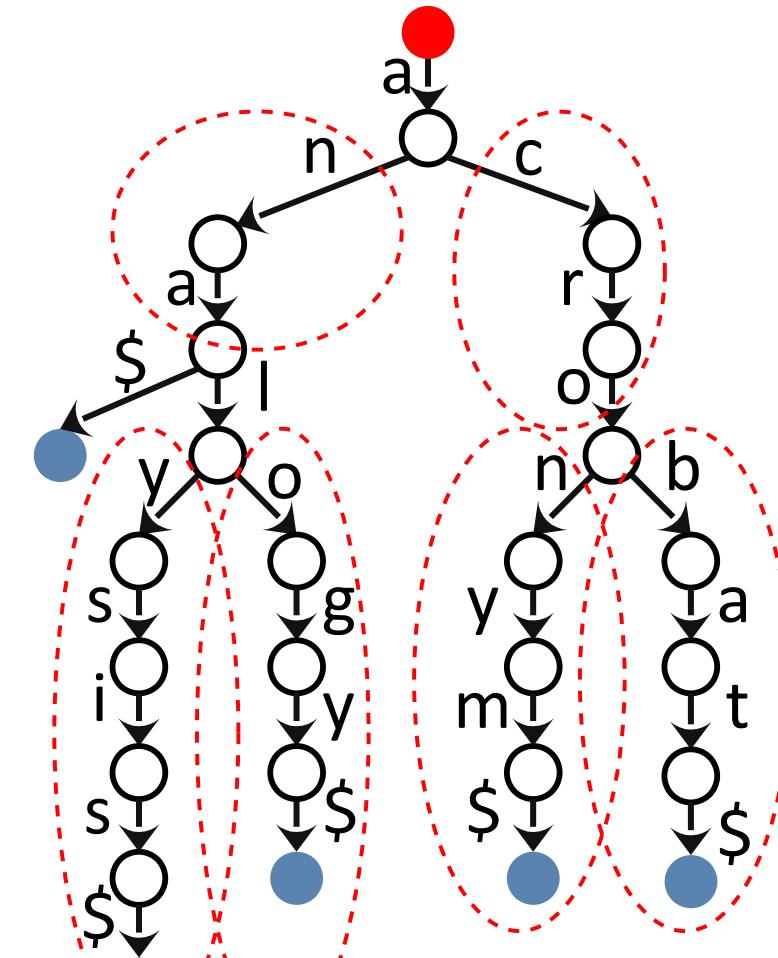
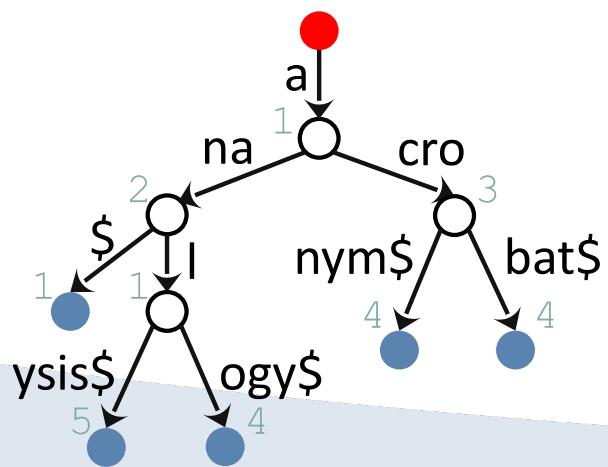
procedure REMOVETRIE( $T, s$ )
  ( $succ, cn$ )  $\leftarrow$  SEARCHTRIE( $T, s$ )
  if not succ then
    return  $\triangleright s$  not found in Trie
   $s \leftarrow \text{reversed}(s)$ 
   $cn \leftarrow \text{parent}(cn)$ 
  for  $ch \in s$  do
    remove transition from  $cn$  for character  $ch$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

```



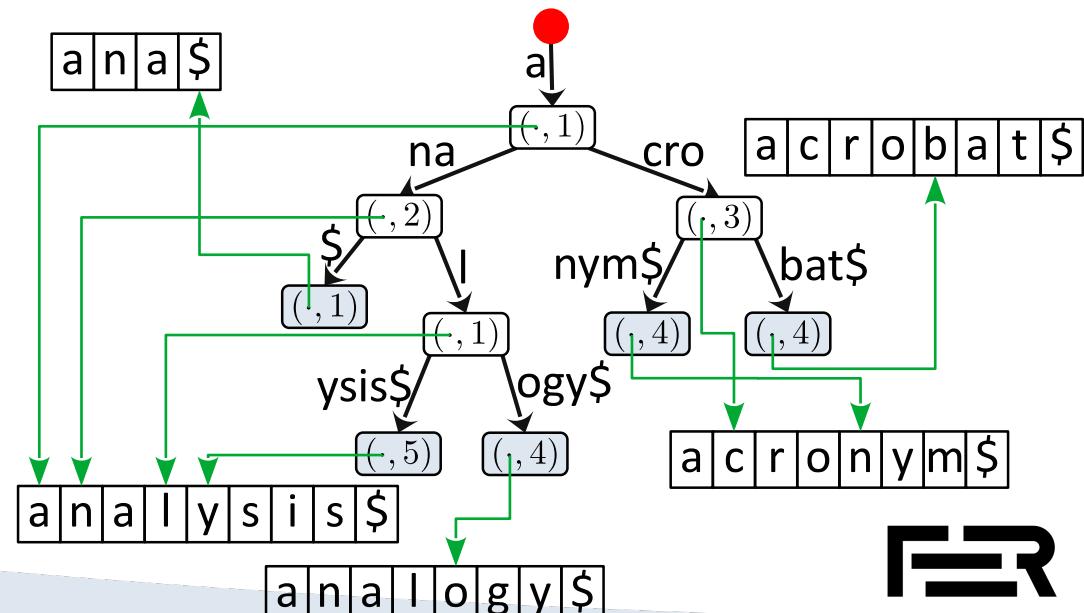
Patricia stablo (1)

- 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)



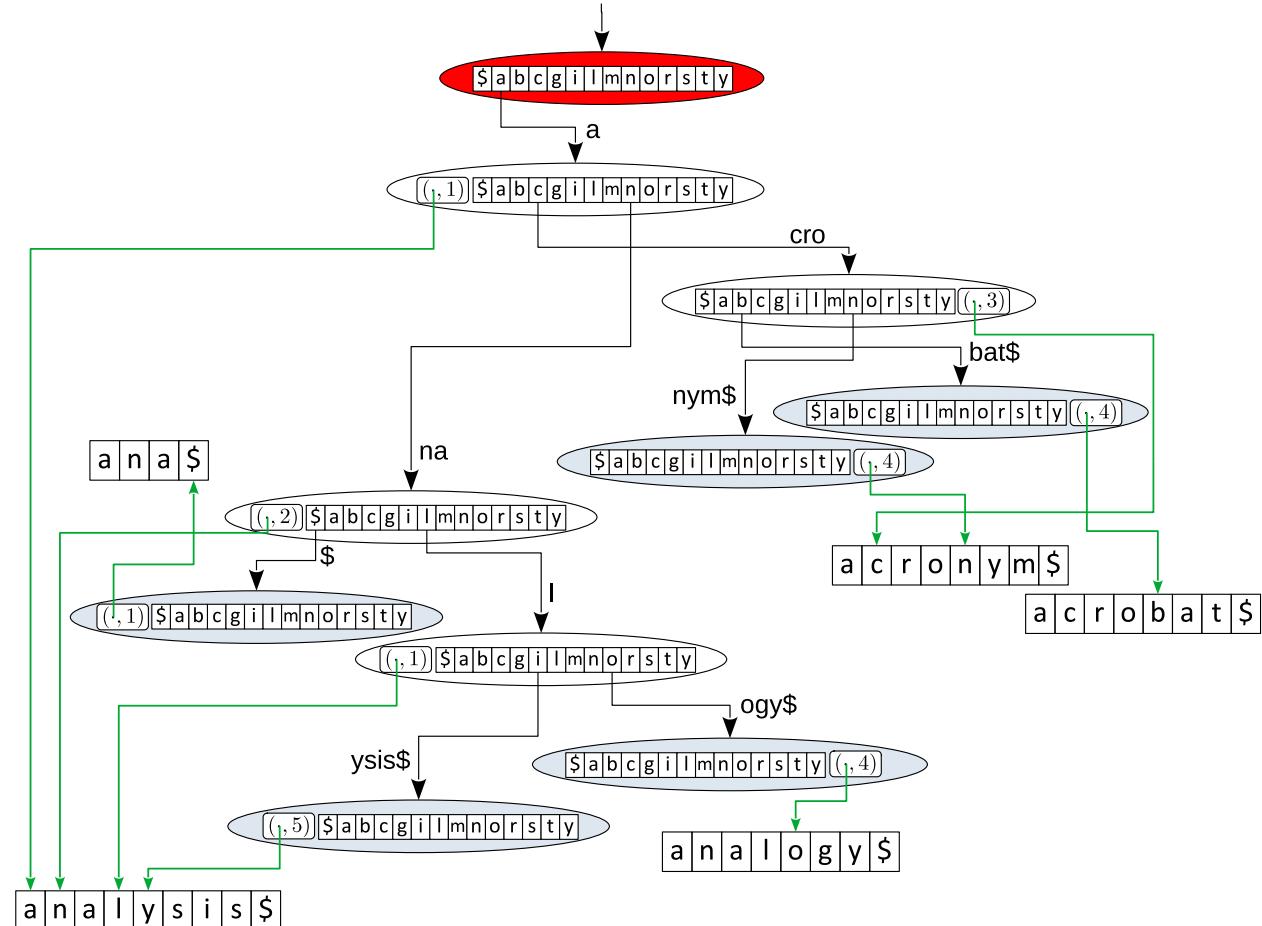
Patricia stablo (2)

- 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 skupu 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 funkcijoniraju 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



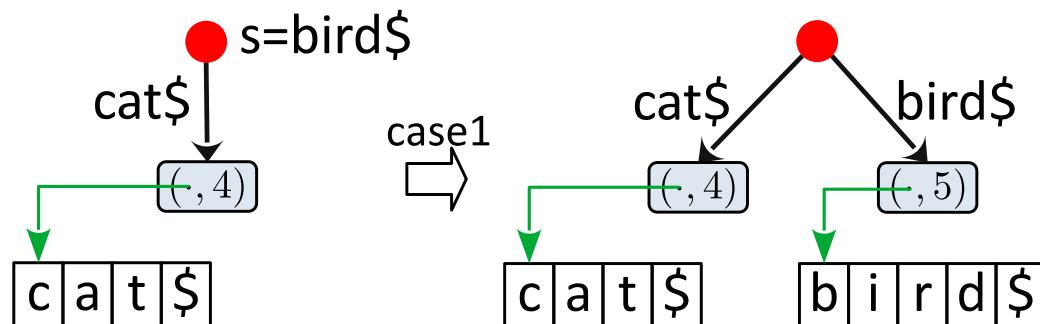
Patricia stablo (4)

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

```
function SEARCHPATRICIA( $P, q = (q_p, q_l)$ )
     $sc \leftarrow 0$ 
     $cn \leftarrow root(P)$ 
    while  $cn$  not leaf do
        if there is transition from  $cn$  for character  $q_p[sc]$  to  $cn_c$  then
             $\triangleright array[q_p[sc]] \neq NULL$  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  $sc = q_l$ 
```

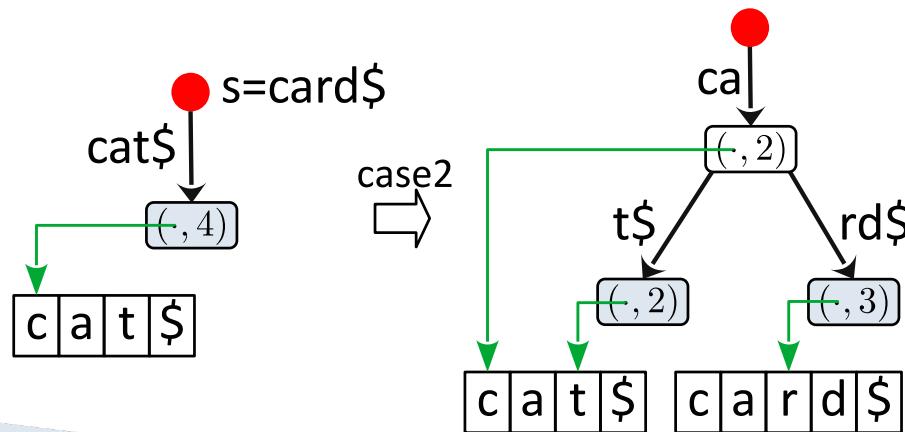
Patricia stablo (5)

- 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



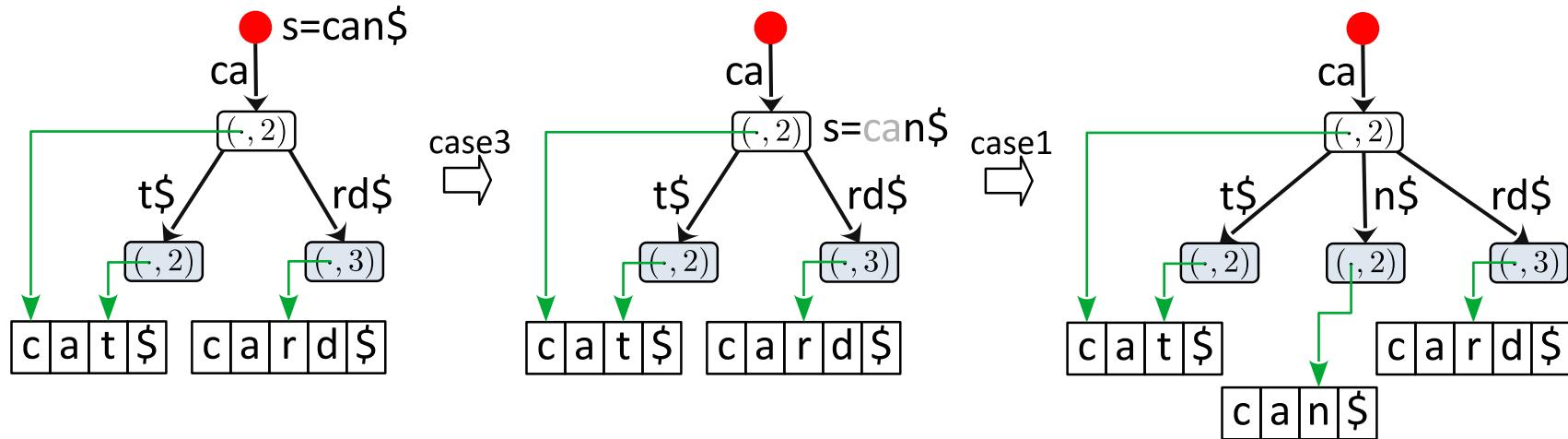
Patricia stablo (6)

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 $O(|s| + |\Sigma|)$

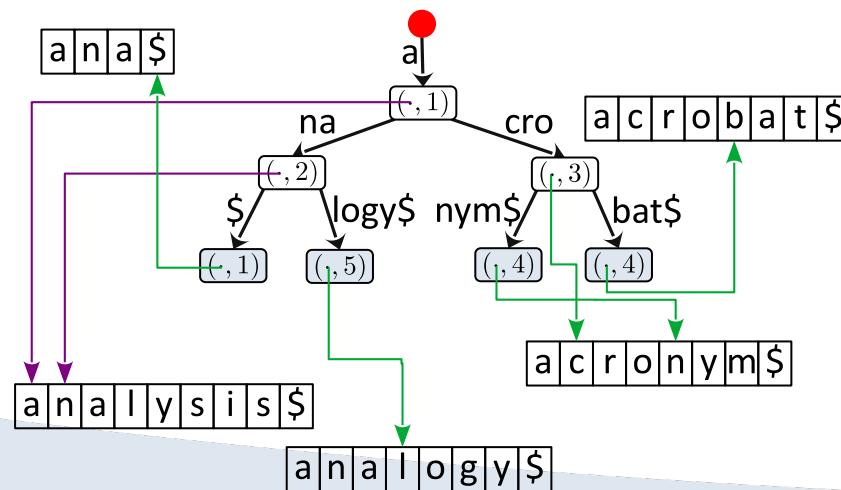
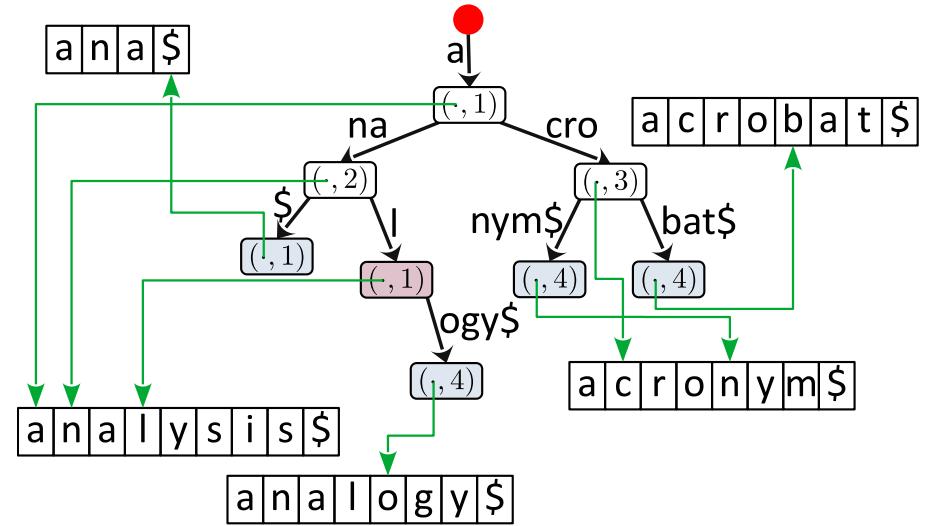
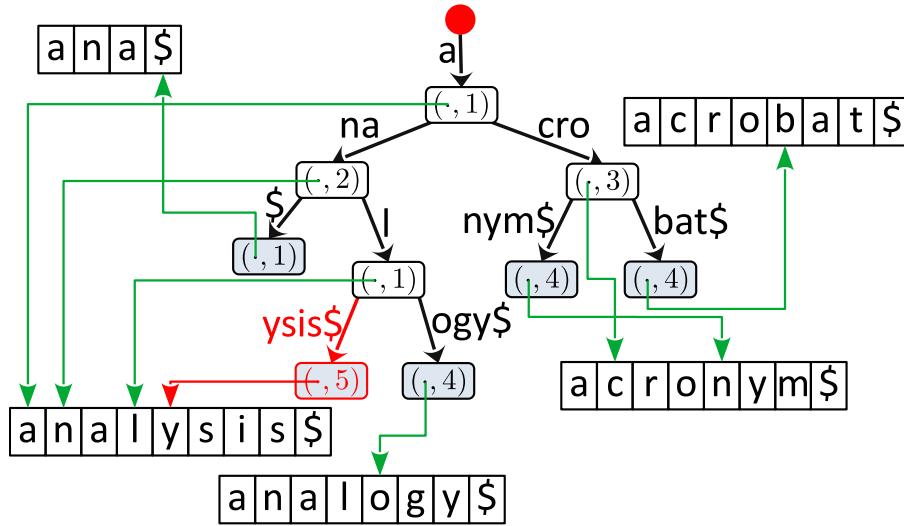
```
procedure INSERTPATRICIA( $P, s = (s_p, s_l)$ )
     $sc \leftarrow 0$ 
     $cn \leftarrow root(P)$ 
    while  $cn$  is root or not leaf do                                 $\triangleright O(|s|)$ 
        if there is transition from  $cn$  for character  $s_p[sc]$  to  $cn_c$  then
             $\triangleright array[s_p[sc]] \neq NULL$  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
 - Uklanjamo 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

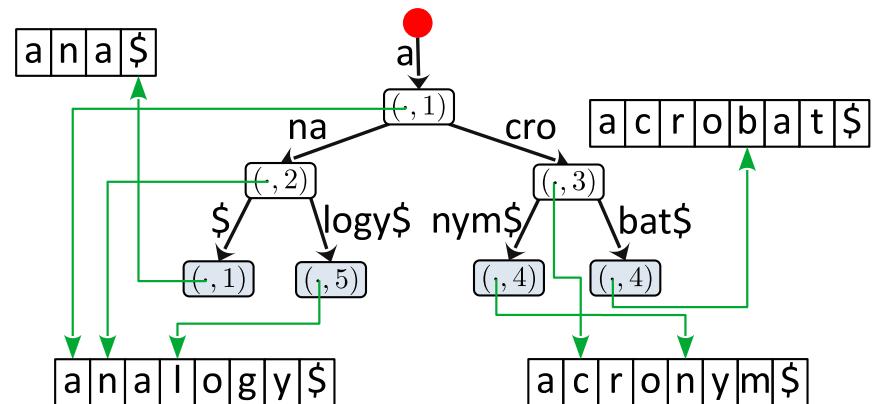
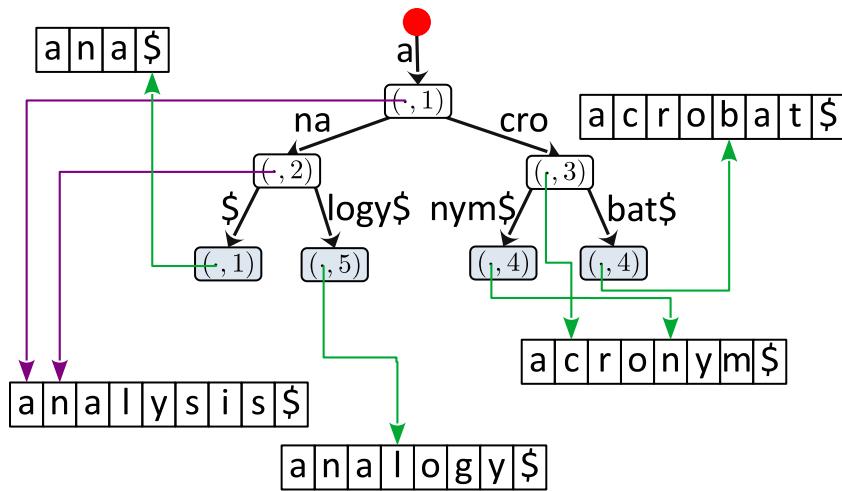
Patricia stablo (10)

- Primjer: uklanjamo $s = \text{„analysis\$”}$



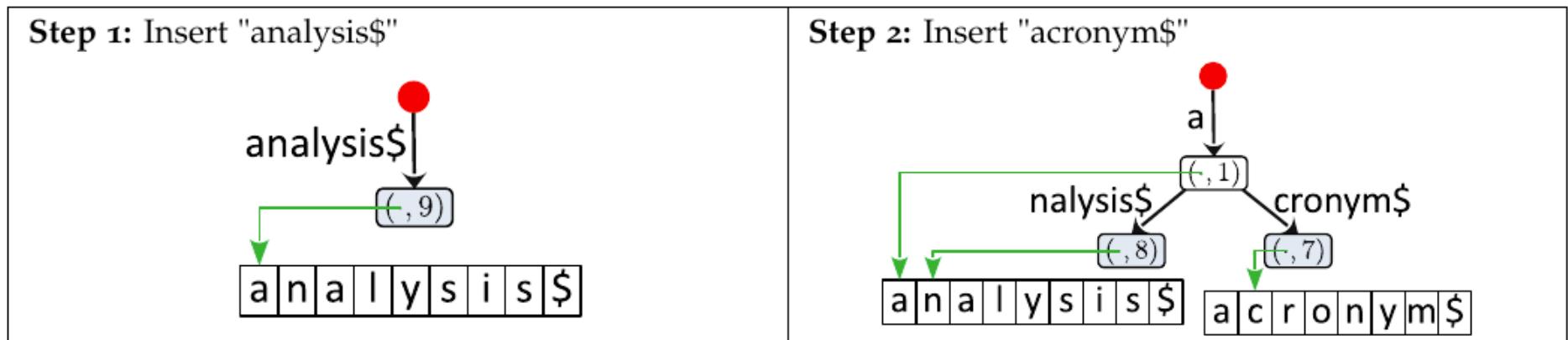
Patricia stablo (11)

- 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

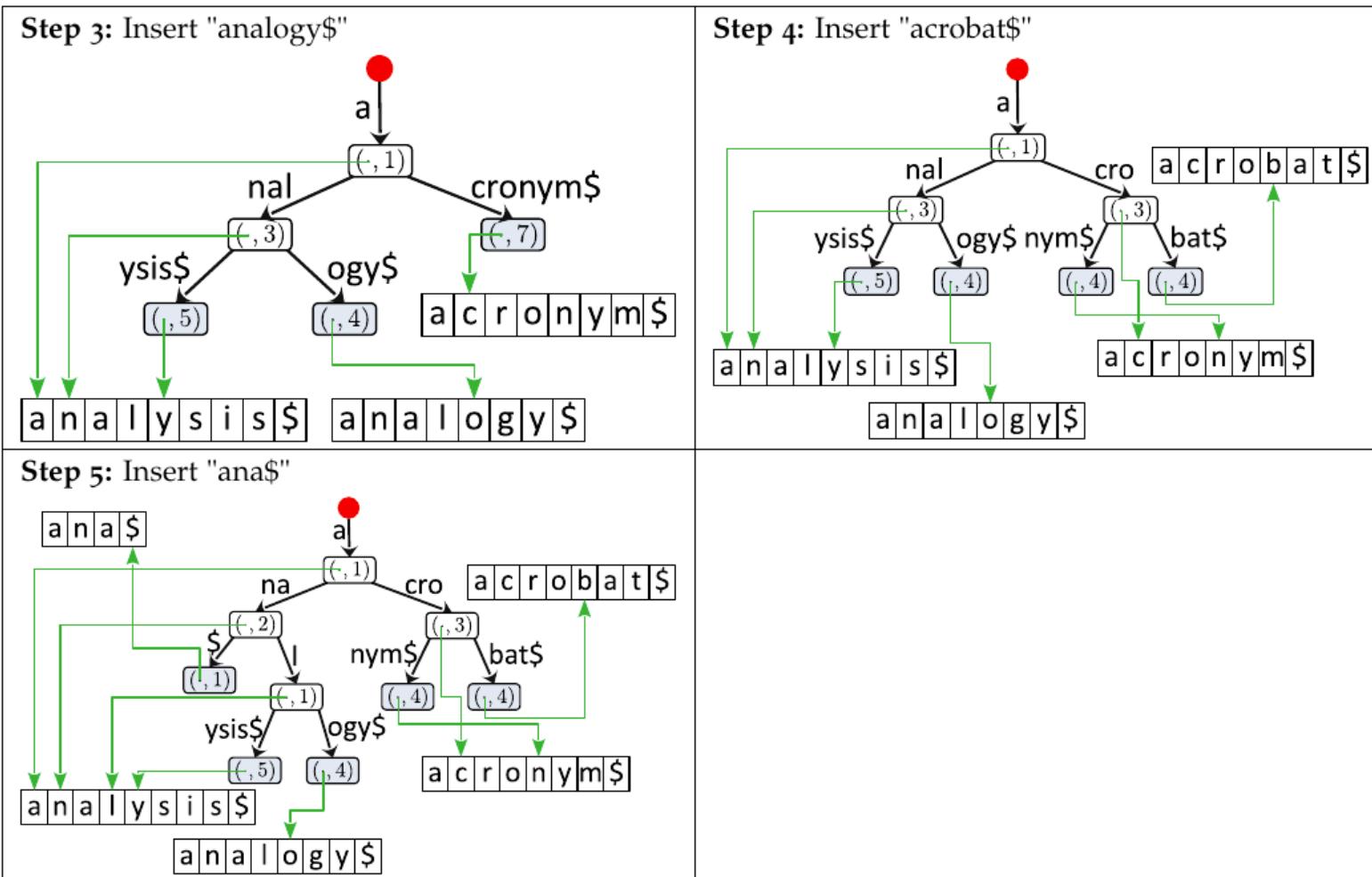


Patricia stablo (12)

- Primjer
 - Nacrtajte evoluciju Patricia stabla za upis sljedećeg slijeda znakovnih nizova
⟨"analysis\$", "acronym\$", "analogy\$", "acrobat\$", "ana\$"⟩



Patricia stable (13)

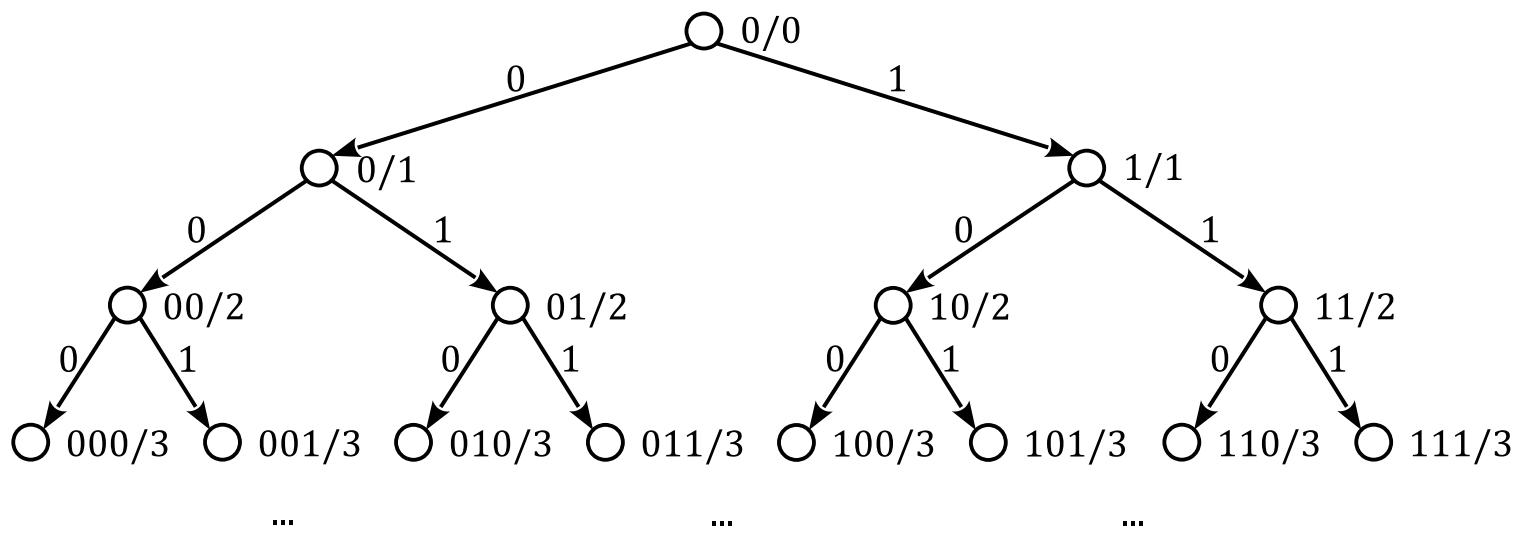


Patricia stablo (14)

- 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 . * . *

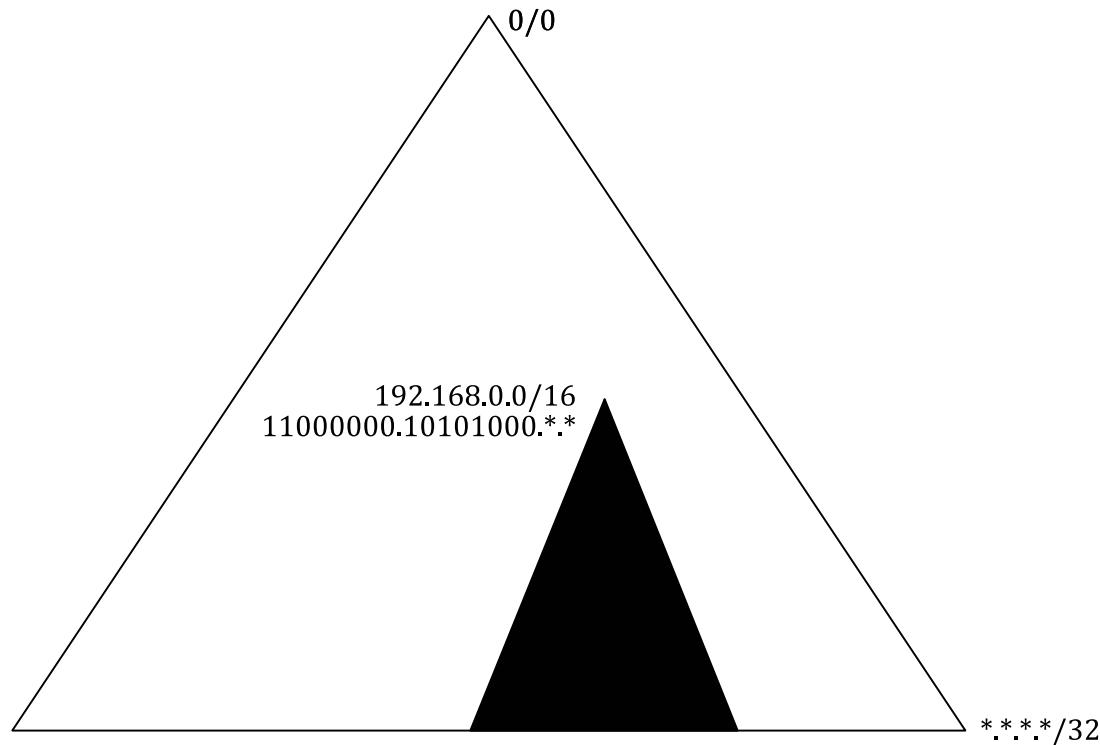
Radix stabla (1)

- 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

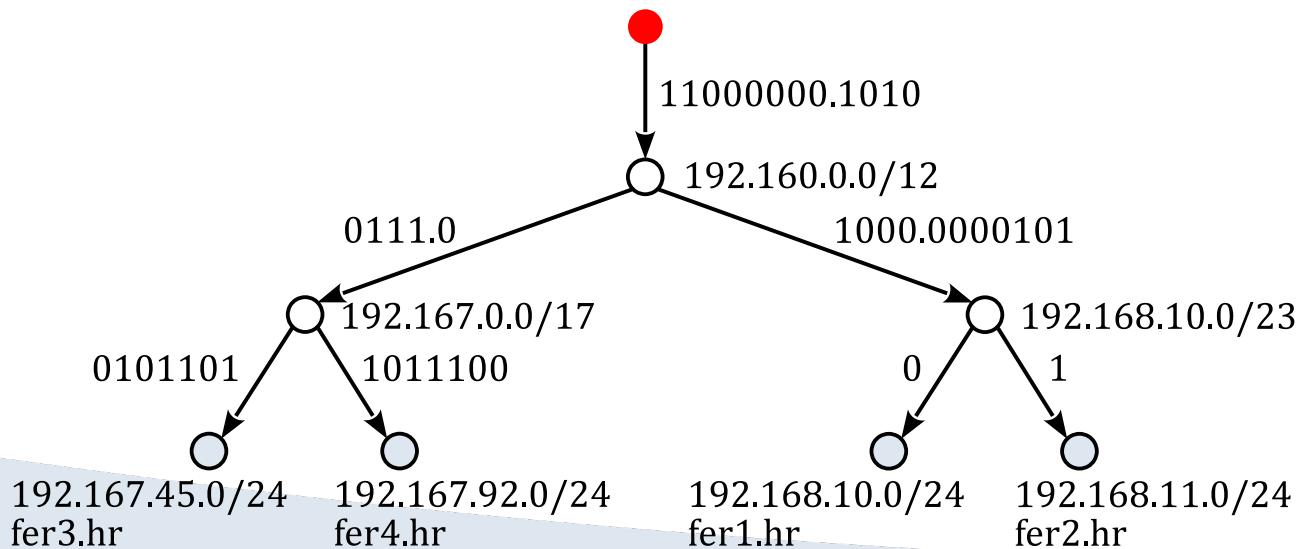


IP routing

- 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 $t = \text{„program$”}$
 - 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
$$(n - 1) + (n - 2) + \cdots + 1 = \frac{n(n + 1)}{2} - n$$
- 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)

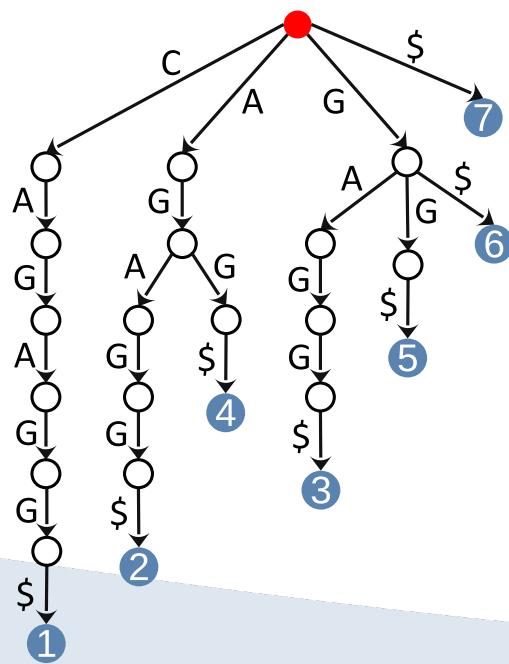
- Kompleksnost stvaranja sufiksnog polja je dosta visoka
- Koristimo algoritam za sortiranje što je $O(n \log n)$, pa kad tome dodamo usporedbu znakovnih nizova, to se može aproksimirati kao $O(n^2 \log n)$
- Ideja sufiksnog polja je provjeriti da li se znakovni niz q nalazi u znakovnom nizu t
 - Naivno 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

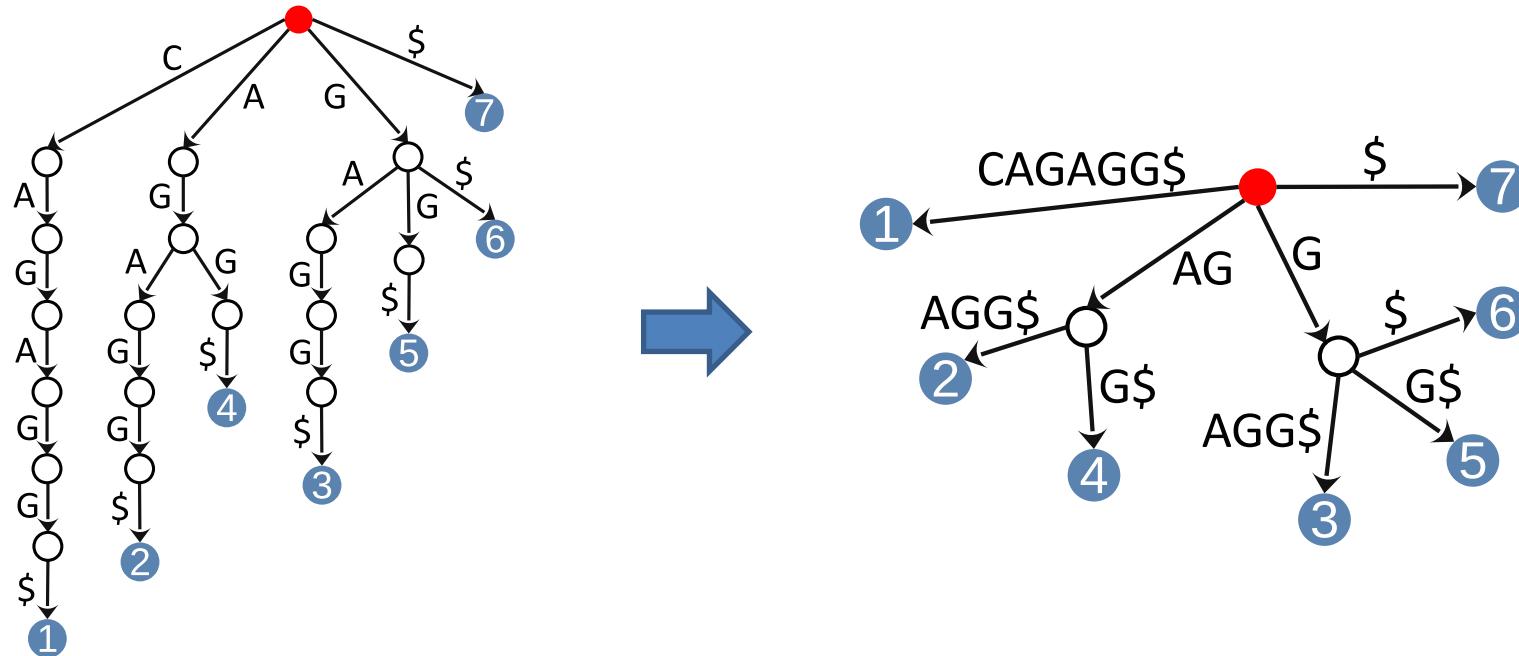


C	A	G	A	G	G	\$
A	G	A	G	G	\$	
G	A	G	G	\$		
A	G	G	\$			
G	G	\$				
G	\$					
\$						

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

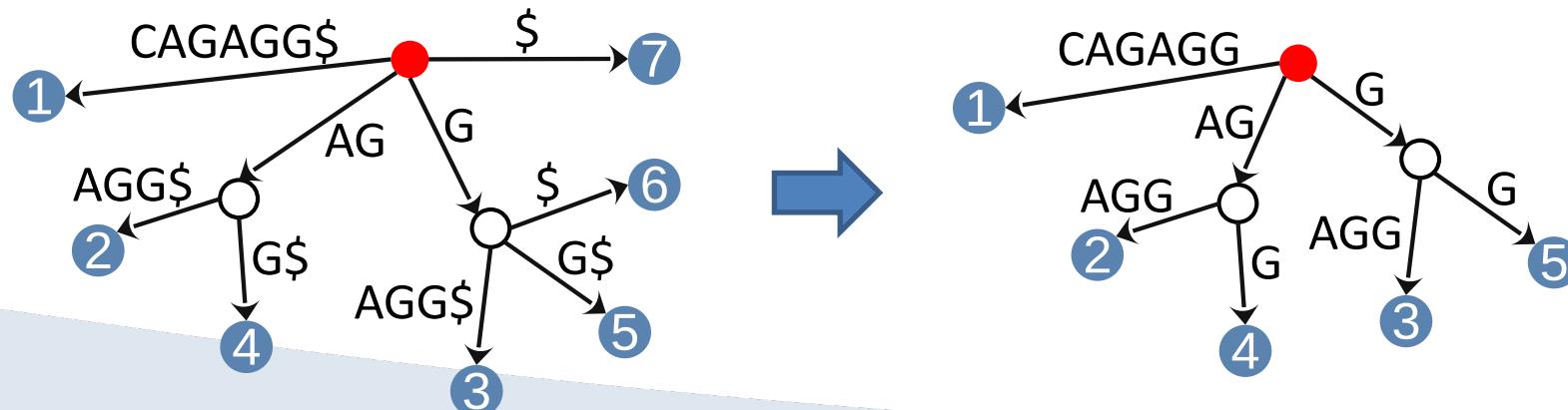
Sufiksno stablo (1)

- 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 eksplisitnog sljedećom transformacijom:
 1. Uklonimo sve NULL terminatore (\$) za bridova eksplisitnog sufiksнog 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 uklanjuju, a njihove tranzicije konkateniraju



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, uvezvi 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
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)

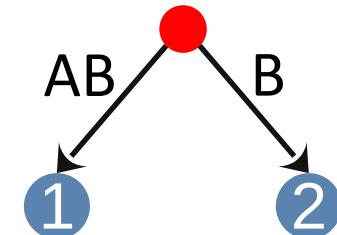
- **Sufiksne veze** dodatak su implicitnom sufiksnom stablu koji omogućavaju brže prolazeње stabla za $t[j, i]$
 - Dodaju se samo između unutarnjih čvorova
 - Imamo unutarnji čvor n_1 u koji smo stigli prolaskom $t[j, i]$
 - Ako imamo neki drugi unutarnji čvor n_2 do kojeg se stigne prolaskom $at[j, i]$, 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 prijeđemo $t[j_1, i]$, tada se za novi $t[i + 1]$ stvara čvor n_2
- U nekom (j_2) . koraku, gdje je $j_2 > j_1$, prijeđemo $t[j_2, i]$ 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

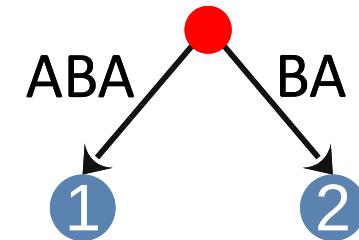
Primjer Ukkonen (1)

- 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.



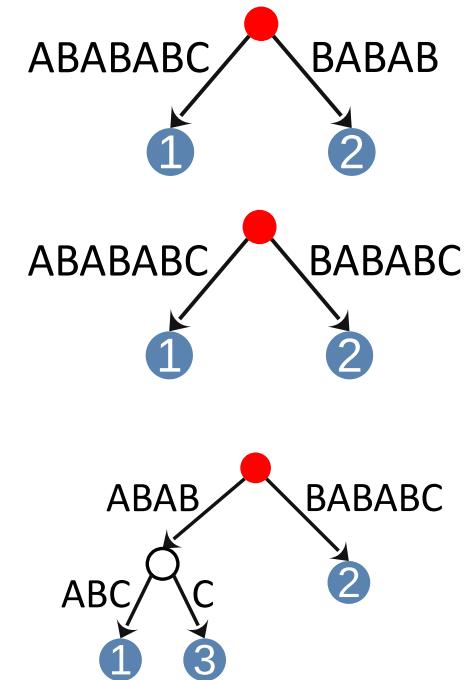
Primjer Ukkonen (2)

- **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.



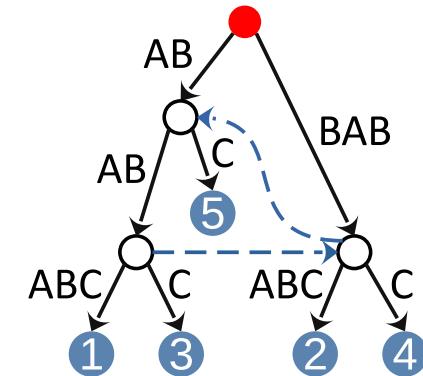
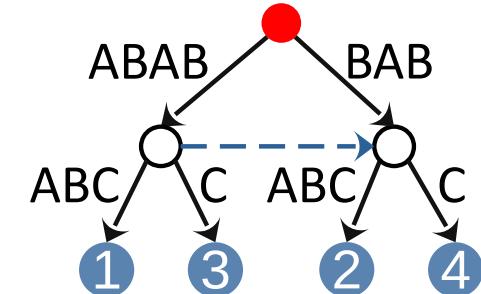
Primjer Ukkonen (3)

- 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)

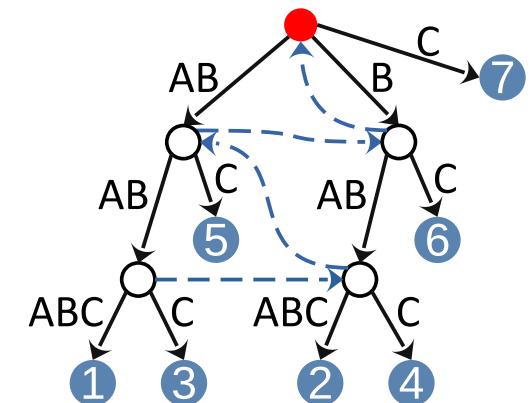
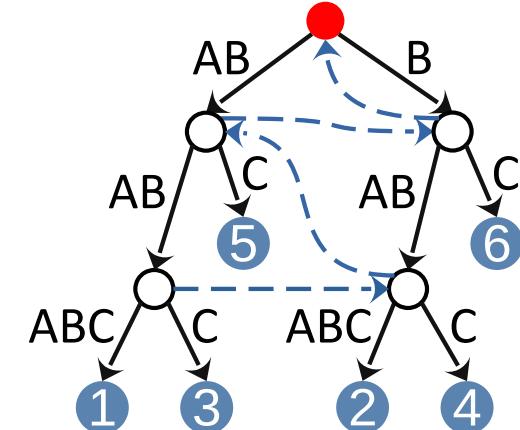
- Korak 4 : $j = 4, t[j, i] = BAB$. Prelazak nas vodi na sredinu tranzicije, a sljedeći znak nije C, što je slučaj 2. Razdvajamo puteve za $BABABC$ i $BABC$. U ovom koraku imamo dva unutarnja čvora $n_1 = BAB$ i $n_2 = ABAB$. Spajamo sufiksnom vezom.
- Korak 5 : $j = 5, t[j, i] = AB$. Prelazak nas vodi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

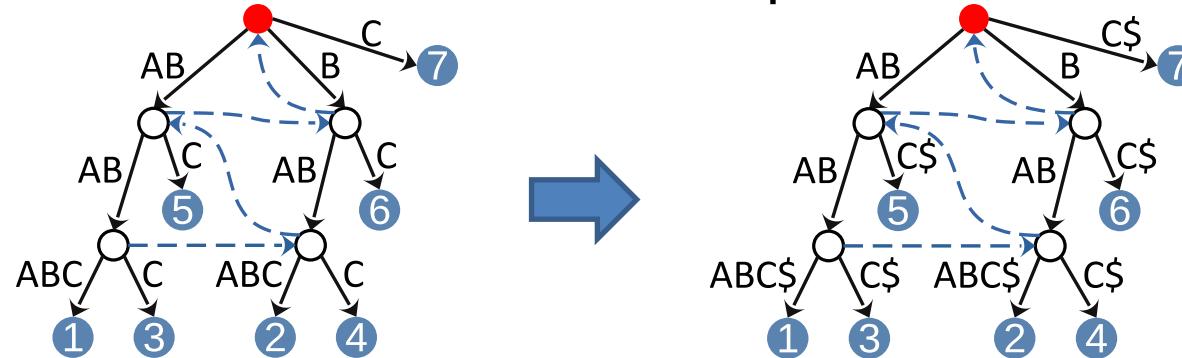
- Korak 6 : $j = 6, t[j, i] = B$. Prelazak nas vodi na sredinu tranzicije, a sljedeći znak nije C, što je slučaj 2. Razdvajamo puteve za $\{BABABC, BABC\}$ i BC . U ovom koraku imamo dva unutarnja čvora $n_1 = B$ i $n_2 = AB$. Spajamo sufiksnom vezom.
- Korak 7 : $t[j, i] = \epsilon$. S obzirom da niti 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 (6)

- Sada možemo transformirati implicitno sufiksno stablo u eksplisitno

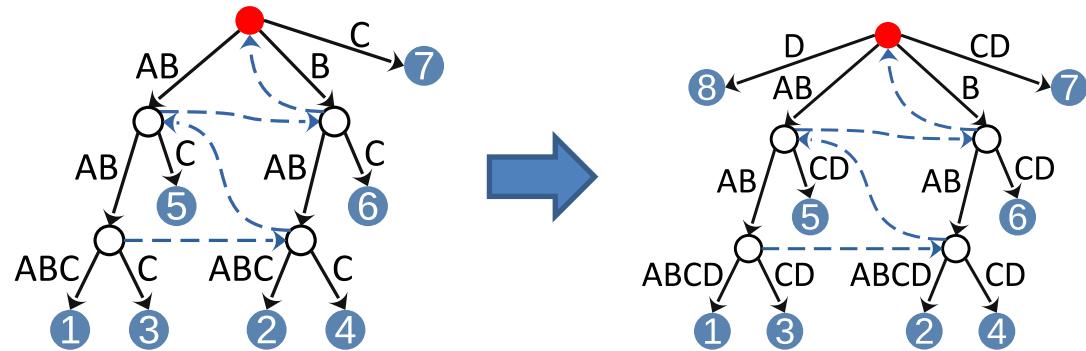


- Recimo da na ovo sufiksno stablo sada dodajemo osmi znak D ($i + 1 = 8$)
- U trenutku kada prolazimo prefiks $t[1,7] = ABABABC$, nailazimo na sufiksne veze
 - Kod $ABAB$ imamo sufiksne veze na 4 dodatna čvora
 - Kod prelaska imamo 5 čvorova od kojih krećemo sa tranzicijama
 - Znamo da možemo preskočiti na $t[6,7]$ jer smo sufikse BAB, AB, B i ϵ imali u prolasku $ABAB$

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 sufiksnim vezama
 - To nas dovodi do provjere tranzicija prema listovima
 - Korištenjem reprezentacije s uređenim parom, dodavanje znaka se svodi na ažuriranje duljine

Pitanja ?