Katedra za računarsku tehniku i informatiku

Algoritmi i strukture podataka 2

Milo V. Tomašević
Marko Mišić
Maja Vukasović

Odsek za softversko inženjerstvo [SI]

Sadržaj

I Linearne strukture podataka

II Nelinearne strukture podataka

III Pretraživanje

IV Sortiranje

Pretraživanje

- Lociranje željenog podatka u cilju pristupa na osnovu neke identifikacije
- Veoma česta aktivnost – utiče na složenost
- Podaci se nalaze u:
  - ✓ tabelama
  - ✓ datotekama
- Zapisi identifikovani ključem
- Ključevi:
  - ✓ unutrašnji i spoljašnji
  - ✓ primarni i sekundarni

Pretraživanje

- Po ishodu operacije pretraživanje:
  - ✓ uspešno ili
  - ✓ neuspešno
- Po mestu pretraživanja:
  - ✓ unutrašnje
  - ✓ spoljašnje
- Skup koji se pretražuje:
  - ✓ statičan ili dinamičan
  - ✓ uređen ili neuređen
- Složenija pretraživanja po pripadnosti intervalu ili složenim uslovima (konjuktivnim ili disjunktivnim)

III.1 Osnovni metodi

Sekvencijalno pretraživanje

- Jednostavnost
- Jedini mogući način kod lista ili neuređenih skupova
- Složenost O(n)

```
SEQ-SEARCH(K, key)
i = 1
while (i ≤ n) do
    if (key = K[i]) then
        return i
    else
        i = i + 1
    end_if
end_while
return 0
```

```
SEQ-SEARCH-SENT(K, key)
K[n + 1] = key
i = 1
while (key ≠ K[i]) do
    i = i + 1
end_while
if (i = n + 1) then
    i = 0
end_if
return i
```

Optimizacije

- Za neunifromne verovatnoće pretraživanja p_i
  $\min \sum (i\, p_i) \Rightarrow p_1 \geq p_2 \geq \ldots \geq p_n$

- Problemi sa verovatnoćama
  - ✓ nisu unapred poznate
  - ✓ menjanju se

- Prebacivanje na početak
  - ✓ nađeni ključ na prvo mesto
  - ✓ oscilacije performansi

- Transpozicija
  - ✓ nađeni ključ se pomera za jedno mesto unapred
  - ✓ stabilnije performanse

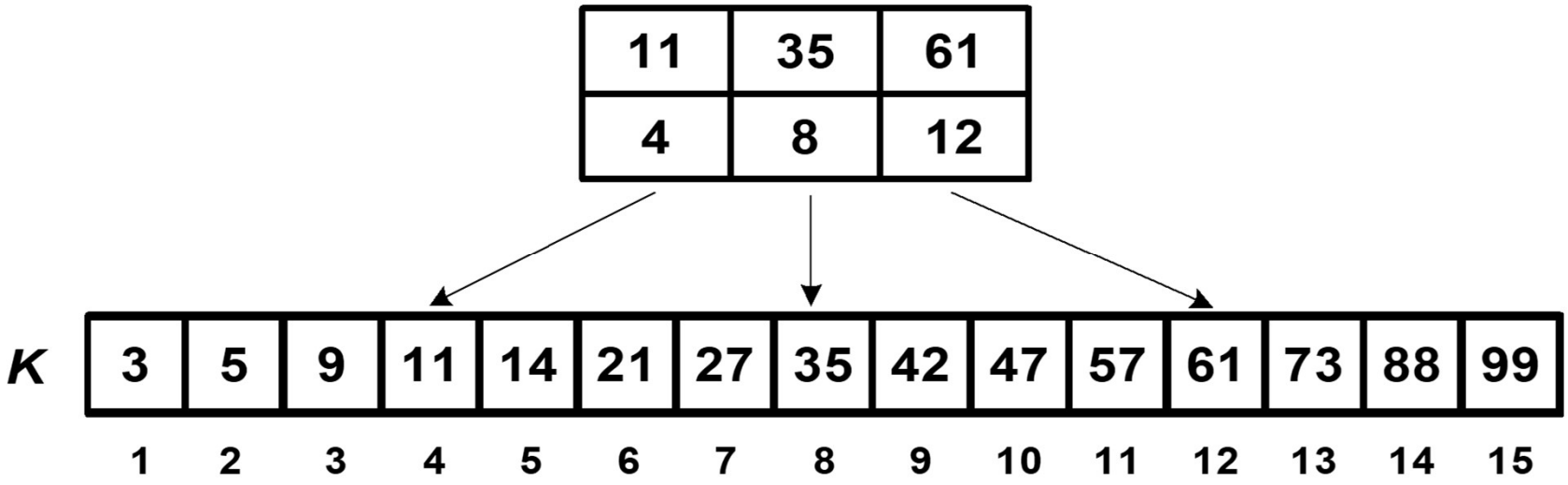
Pretraživanje na više ključeva

```
SEQ-SEARCH-MUL(K, S)
for i = 1 to m do
    P[i] = 0
end_for
i = j = 1
while (i ≤ n) and (j ≤ m) do
    while (i < n) and (S[j] > K[i]) do
        i = i + 1
    end_while
    if (S[j] = K[i]) then
        P[j] = i
    end_if
    j = j + 1
end_while
return
```

- Uređena tabela K
- Uređen niz ključeva S
- Složenost $O(n)$

Pretraživanje sa indeksom

- Pomoćna struktura - indeks
  - ✓ mnogo manji i brže se pretražuje
  - ✓ sužava opseg pretraživanja u tabeli
  - ✓ nekad i više od jednog nivoa
  - ✓ održavanje pri promeni



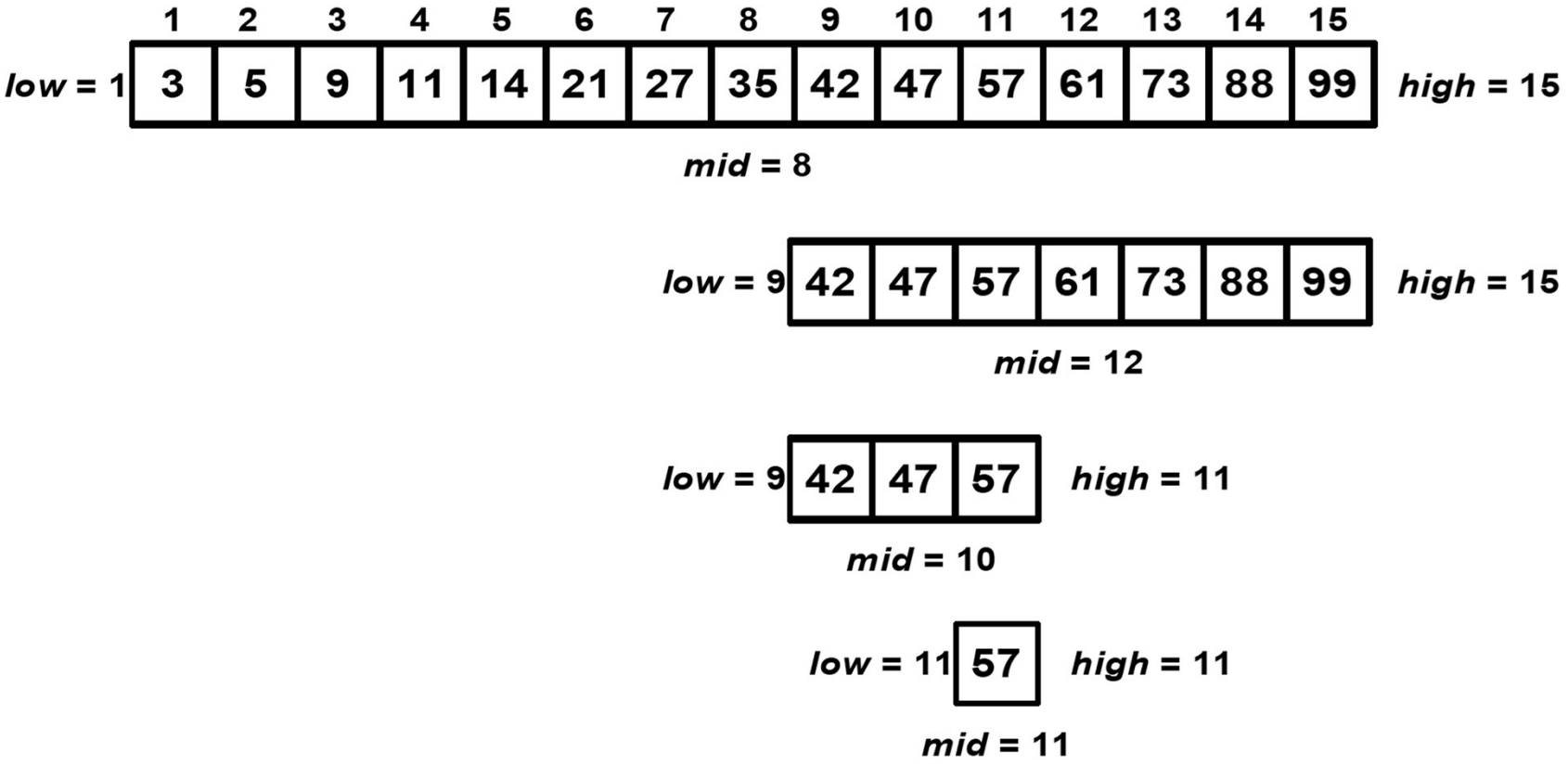
Binarno pretraživanje

- Sekvencijalno pretraživanje sporo konvergira čak i u uređenim nizovima
- Za uređene nizove rekurzivna taktika “podeli-pobedi”
- Binarno odlučivanje polovi interval pretraživanja
- Rekuzivna i iterativna realizacija
- Neprimenljivo za ulančane liste (čak i uređene)
- Vremenska složenost $O(\log n)$

Binarno pretraživanje

```
BIN-SEARCH(K, key)
low =1
high = n
while (low ≤ high) do
      mid = (low + high)/2
      if (key = K[mid]) then
            return mid
      else if (key < K[mid]) then
                  high = mid - 1
            else
                  low = mid + 1
            end_if
      end_if
end_while
return 0
```

Primer


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
low = 1
3 5 9 11 14 21 27 35 42 47 57 61 73 88 99
high = 15
mid = 8
low = 9
42 47 57 61 73 88 99
high = 15
mid = 12
low = 9
42 47 57
high = 11
mid = 10
low = 11
57
high = 11
mid = 11

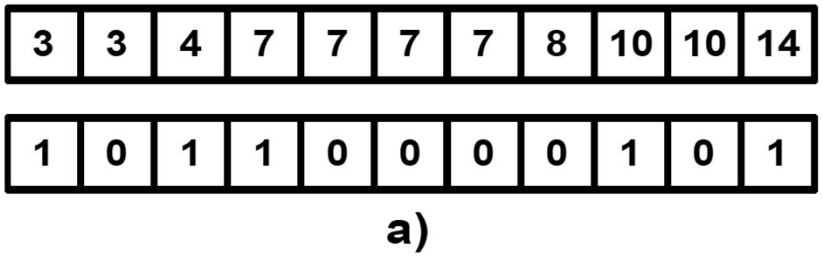
Varijante binarnog pretraživanja

- Binarno pretraživanje u tabeli nepoznate veličine
  - ✓ ne zna se sredina
  - ✓ binarno se odredi interval (i...$2i$) - $O(\log i)$
  - ✓ standardno pretraživanje u intervalu - $O(\log i)$
  - ✓ efikasno za pretraživanje na početku i kraju

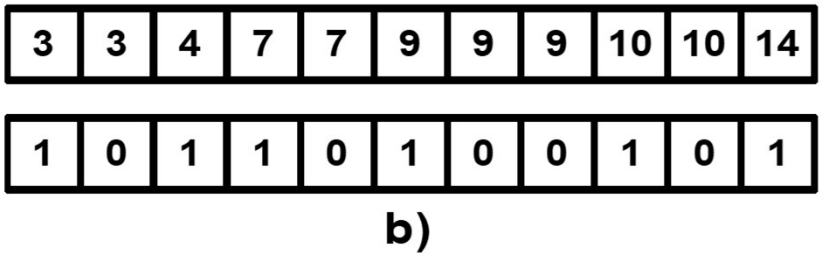
- Binarno pretraživanje u povećanoj tabeli
  - ✓ problem umetanja i brisanja ključeva
  - ✓ maksimizacija veličine tabele
  - ✓ vektor validnosti
  - ✓ "prividni ključevi" na slobodnim pozicijama
    (= ili >od prvog prethodnog ključa,
    a < od prvog narednog validnog ključa)

Varijante binarnog pretraživanja

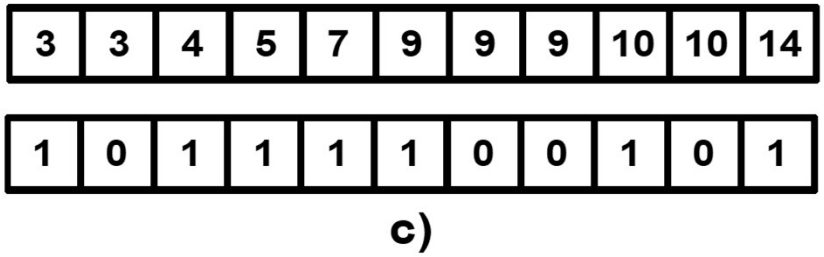
- Operacije
  - ✓ pretraživanje (konsultovanje bita validnosti)
  - ✓ umetanje (moguća replikacija i pomeranja)
  - ✓ brisanje (resetovanje bita validnosti)



a)



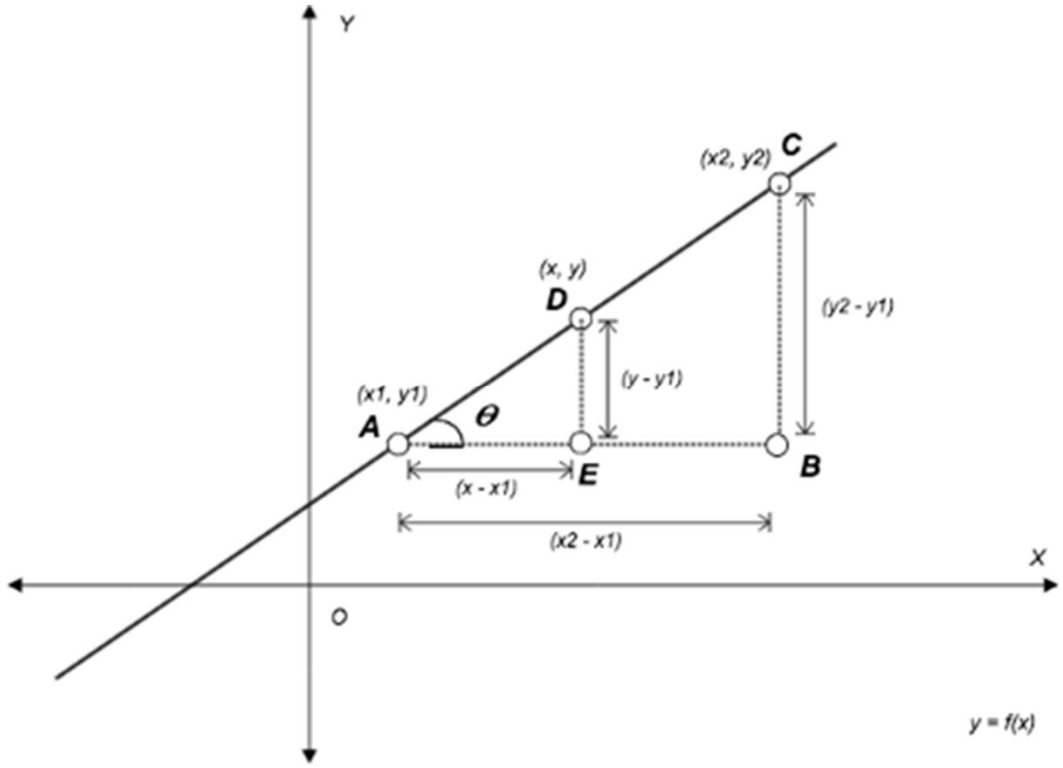
b)



c)

Varijante binarnog pretraživanja

- Interpolaciono pretraživanje - interpretacija
  - ✓ pozicija poređenja
    low + (*high* - *low*)(*key* – *K*[*low*])/(*K*[*high*] - *K*[*low*])
  - ✓ x osa – pozicije ključeva, y osa - vrednost
  - ✓ Detaljno objašnjenje:



Varijante binarnog pretraživanja

- Interpolaciono pretraživanje
  - ✓ binarno pretraživanje neosetljivo na sadržaj
  - ✓ za uniformnu raspodelu ključeva može i bolje
  - ✓ pozicija poređenja
    $low + (high - low)(key - K[low])/(K[high] - K[low])$
  - ✓ performanse i do $O(\log \log n)$
  - ✓ može biti pogodno za spoljašnje pretraživanje
  - ✓ za neuniformne raspodele nije dobro

- Fibonacci-jevo pretraživanje
  - ✓ asimetrična deoba intervala
    u odnosu *Fibonacci*-jevih brojeva
  - ✓ performanse $O(\log n)$, ali nešto lošije

III.2 Stablo binarnog pretraživanja

Stablo binarnog pretraživanja

- BP nepogno za dinamičke tabele

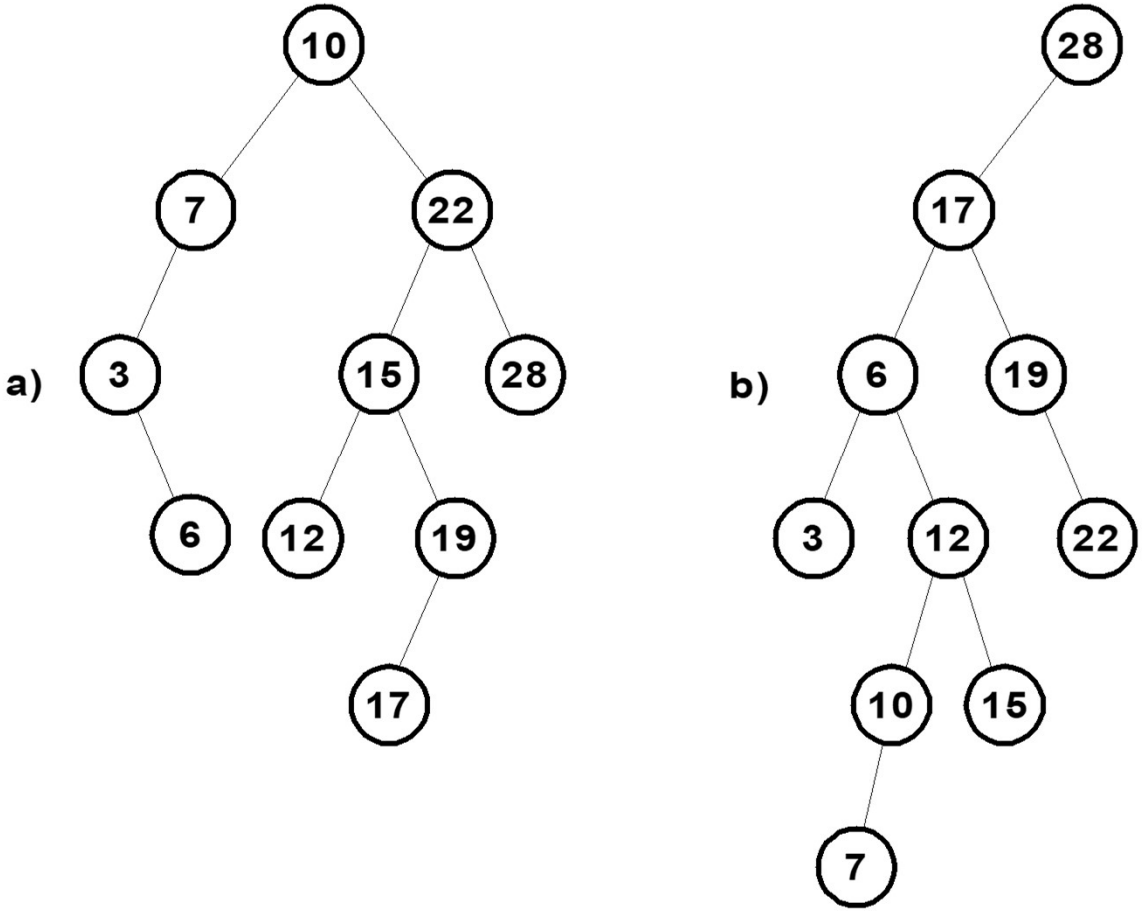
- Rešenje – binarno stablo
  - ✓ dinamička struktura
  - ✓ uređenje po sadržaju

- Stablo binarnog pretraživanja (BST)
  - ✓ za svaki ključ K najviše jedan čvor $key(p) = K$
  - ✓ u levom podstablu važi $key(p_l) < K$
  - ✓ u desnom podstablu važi $key(p_r) > K$

- *Inorder* daje sortirani poredak

- Ključevi se ne ponavljaju

Stablo binarnog pretraživanja



- Isti *inorder* poredak, različita stabla

Pretraživanje BST

```
BST-SEARCH(root, K)
p = root
if (p = nil) or (K = key(p)) then
        return p
else if (K < key(p)) then
                return BST-SEARCH(left(p), K)
        else
                return BST-SEARCH(right(p), K)
        end_if
end_if
```

```
BST-SEARCH-I(root, K)
p = root
while (p ≠ nil) and (K ≠ key(p))
do
        if (K < key(p)) then
                p = left(p)
        else
                p = right(p)
        end_if
end_while
return p
```

- Složenost (maksimalna) - $O(h)$
- Ubrzanje sa eksternim čvorom graničnikom

Ispitivanje BST

```
BST-MIN(root)
p = root
while (left(p) ≠ nil) do
      p = left(p)
end_while
return p
```

- Nalaženje najmanjeg ključa u stablu

```
BST-MAX(root)
p = root
while (right(p) ≠ nil ) do
      p = right(p)
end_while
return p
```

- Nalaženje najvećeg ključa u stablu

Ispitivanje BST

```
BST-SUCC(r)
p = r
if (right(p) ≠ nil) then
    return BST-MIN(right(p))
else
    q = parent(p)
    while (q ≠ nil and p = right(q)) do
        p = q
        q = parent(q)
    end_while
    return q
end_if
```

- Nalaženje sledbenika po *inorderu*
- Pokazivač na oca - *parent*

Umetanje u BST

```
BST-INSERT(new, root)
p = root
if (p = nil) then
    root = new
else
    if (key(new) = key(p)) then
        ERROR(Postoji ključ)
    else if (key(new) < key(p)) then
            BST_INSERT(new, left(p))
        else
            BST_INSERT(new, right(p))
        end_if
    end_if
end_if
```

Umetanje u BST

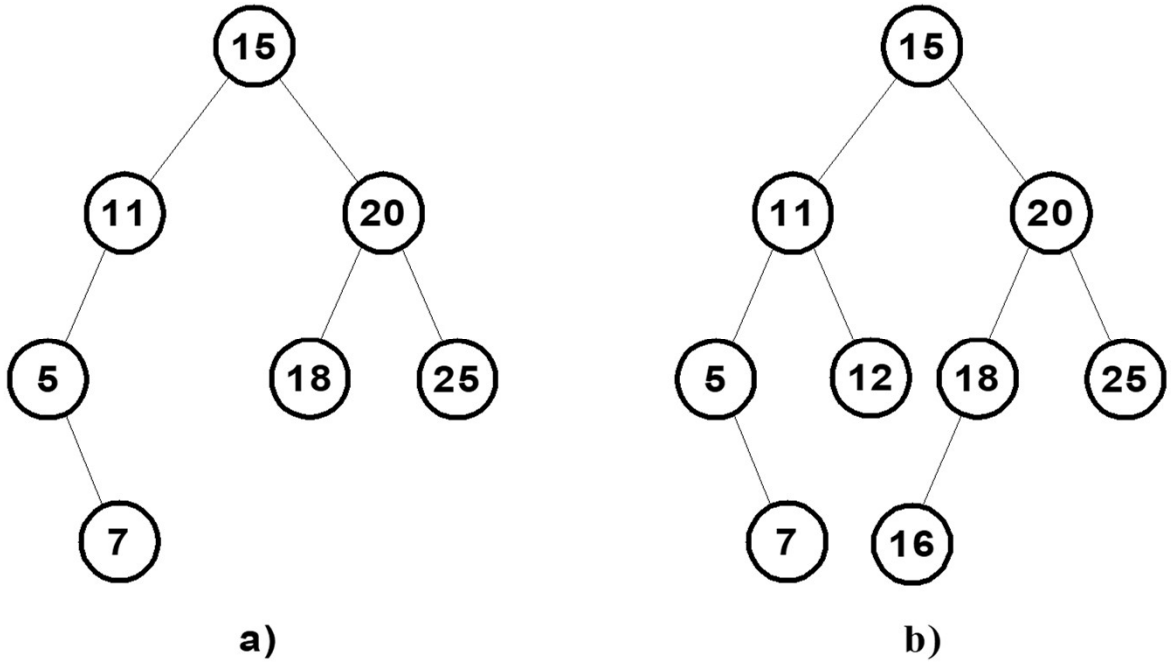
```
BST-INSERT-I(new, root)
p = root
q = nil
while (p ≠ nil) do
    q = p
    if (key(new) = key(p)) then
        ERROR(Postoji ključ)
    else if (key(new) < key(p)) then
        p = left(p)
    else
        p = right(p)
    end_if
    end_if
end_while
```

```
if (q = nil) then
    root = new
else
    if (key(new) < key(q))
then
        left(q) =new
    else
        right(q) = new
    end_if
end_if
```

- Složenost $O(h)$

Umetanje u BST

- Primer – umetanje ključeva 12 i 16

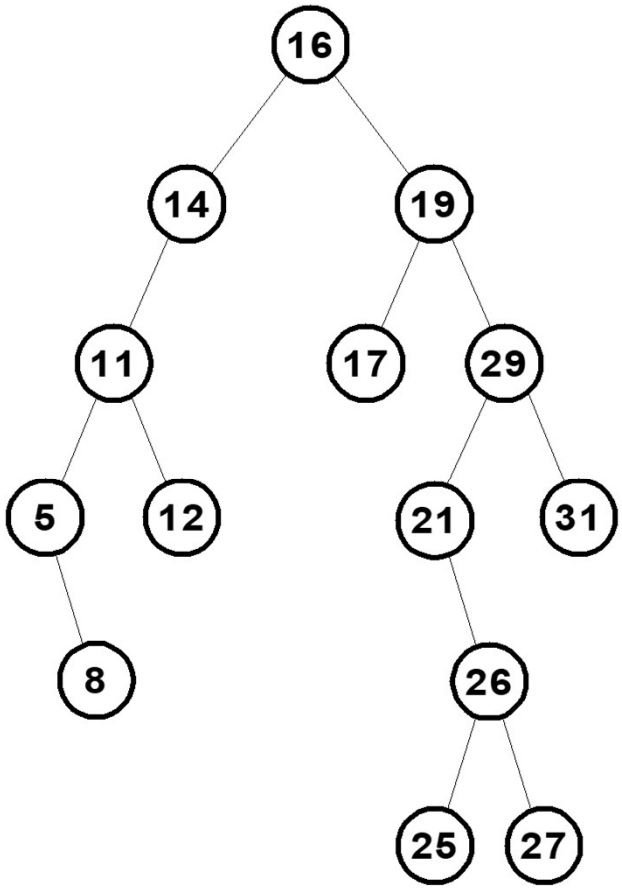


- Sa mogućnošću dodavanja istih ključeva,
  može se koristiti i kao efikasan algoritam sortiranja

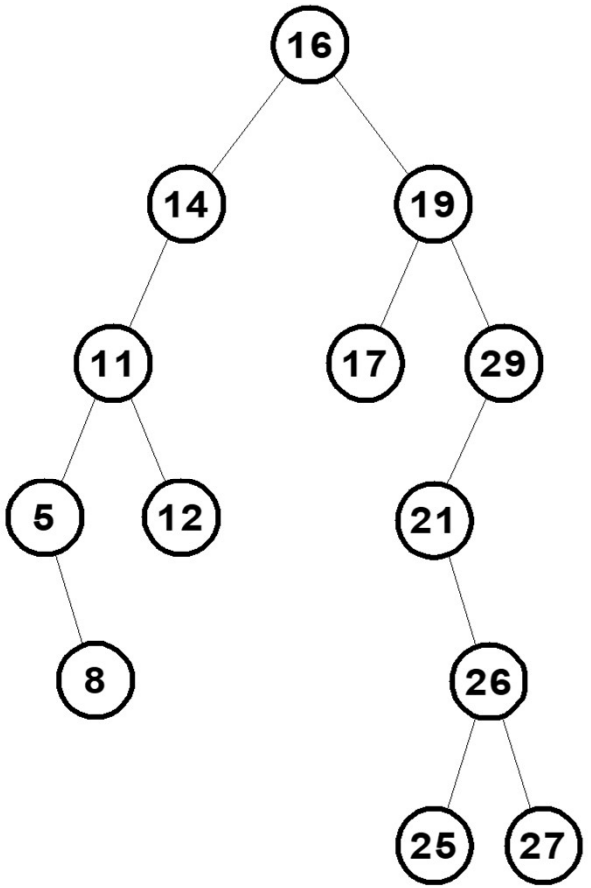
Brisanje iz BST

- Brisanje lista
- Brisanje čvora sa jednim podstablom
  - ✓ preuzima ga otac
- Brisanje čvora sa dva podstabla
  - ✓ zamenjuje ga sledbenik (ili prethodnik)
- Složenost $O(h)$

Brisanje iz BST

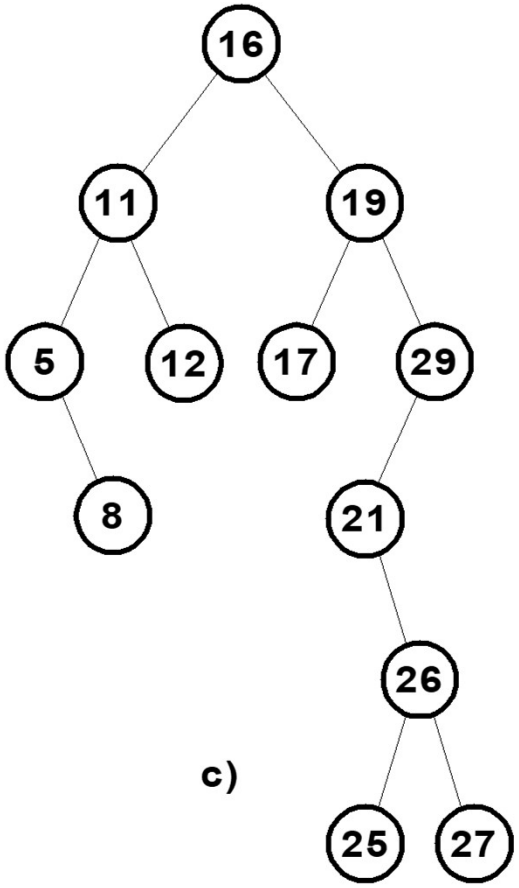


a)



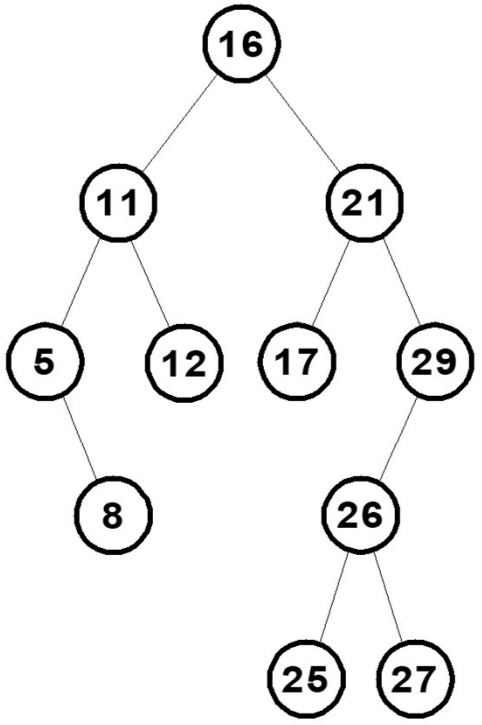
b)

Stabla



c)

- 14



d)

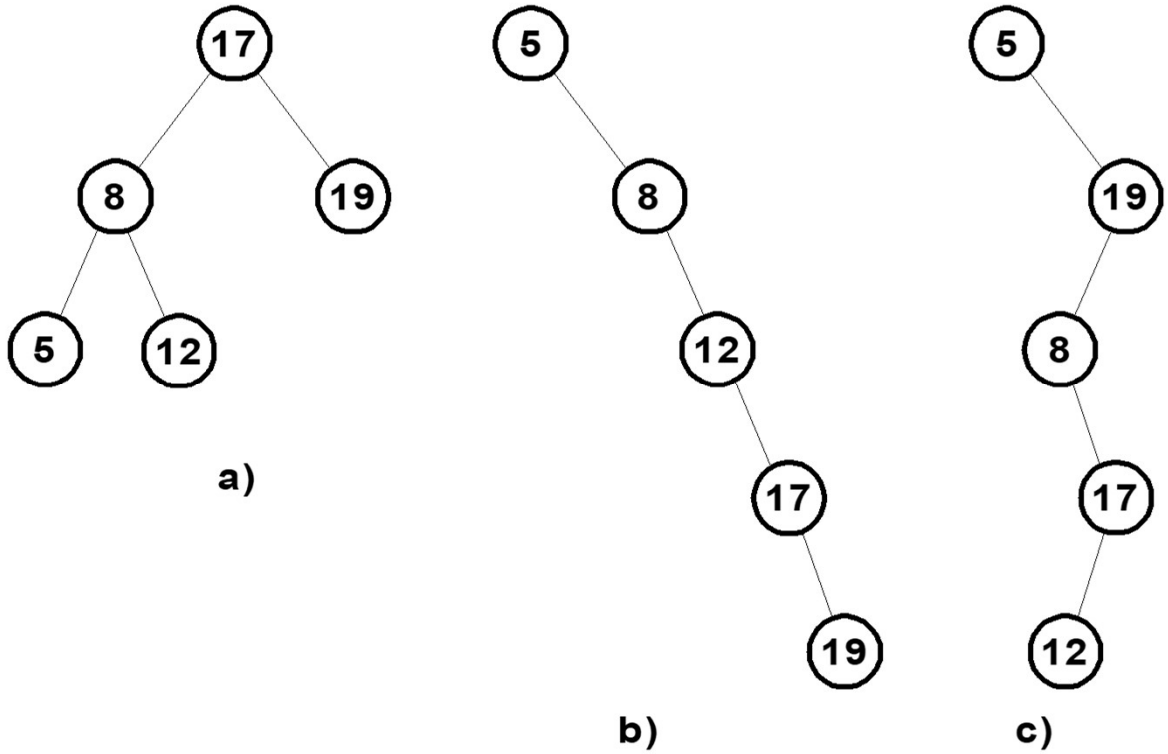
- 19

Brisanje iz BST

```
BST-DELETE(K, root)
p = root
q = nil
while (p ≠ nil and K ≠ key(p)) do
    q = p
    if (K < key(p)) then
        p = left(p)
    else
        p = right(p)
    end_if
end_while
if (p = nil) then
    ERROR(Ključ nije pronađen)
end_if
if (left(p) = nil) then
    rp = right(p)
else if (right(p) = nil) then
        rp = left(p)
    else
        f = p
        rp = right(p)
        s = left(rp)
        while (s ≠ nil) do
            f = rp
            rp= s
            s = left(rp)
        end_while
        if (f ≠ p) then
            left(f) = right(rp)
            right(rp) = right(p)
        end_if
        left(rp) = left(p)
    end_if
end_if
if (q = nil) then
    root = rp
else if (p = left(q)) then
        left(q) = rp
    else
        right(q) = rp
    end_if
end_if
FREENODE(p)
```

Analiza performansi BST

- Efikasnost operacija srazmerna visini
  - ✓ najbolji slučaj $O(\log n)$
  - ✓ najgori slučaj $O(n)$



Analiza performansi BST

$S_n = 1 + (U_0 + U_1 + ... + U_{n-1})/n$

$S_n = (1/n) \sum_{i=1}^{n} (p_i + 1) = (PI + n)/n$

$U_n = PE/(n + 1)$

$PE = PI + 2n$

$S_n = (PE - 2n + n)/n =$
$((n + 1)U_n - n)/n = ((n + 1)/n)U_n - 1$

$2n + U_0 + U_1 + ... + U_{n-1} = (n + 1)U_n$

$2(n - 1) + U_0 + U_1 + ... + U_{n-2} = nU_{n-1}$

$2 + U_{n-1} = (n + 1) U_n - nU_{n-1}$

$U_n = U_{n-1} + 2/(n + 1)$

$U_0 = 0$ i $U_1 = 1$

$H_{n+1} = 1 + 1/2 + ... + 1/(n + 1)$

$U_n = 2 H_{n+1} - 2$

$S_n = ((n + 1)/n) U_n - 1$

$S_n = 2(1 + 1/n) H_n - 3$

$H_n = \gamma + \ln n + 1/(12n^2) + ..$

$\gamma = 0.577$

$S_n = 2(\ln n + \gamma) - 3 = 2\ln n - c$

$2\ln n/\log n = 2\ln 2 \approx 1.386$

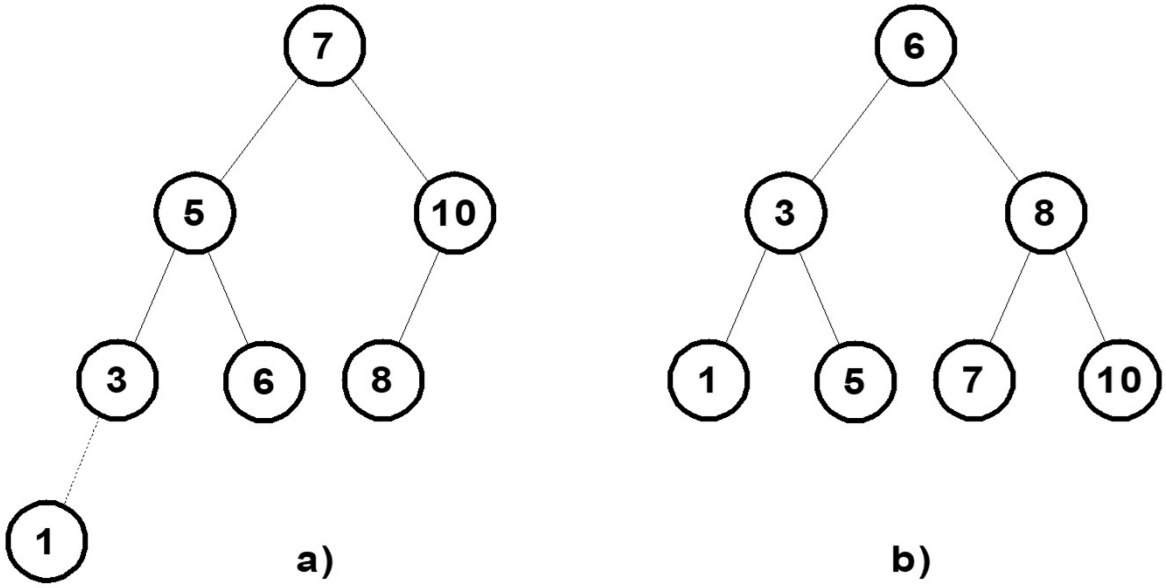
Analiza performansi BST

- Pretpostavke
  - ✓ *n* ključeva sa jednakom verovatnoćom pristupa
  - ✓ "slučajno" BST nastalo umetanjima u proizvoljnom poretku

- Zaključak
  - ✓ prosečna performansa pretraživanja u slučajnom stablu proizvoljne topologije istog reda $O(\log n)$ kao i u optimalnom i lošija samo za konstantan faktor < 40% !

Balansiranje stabla

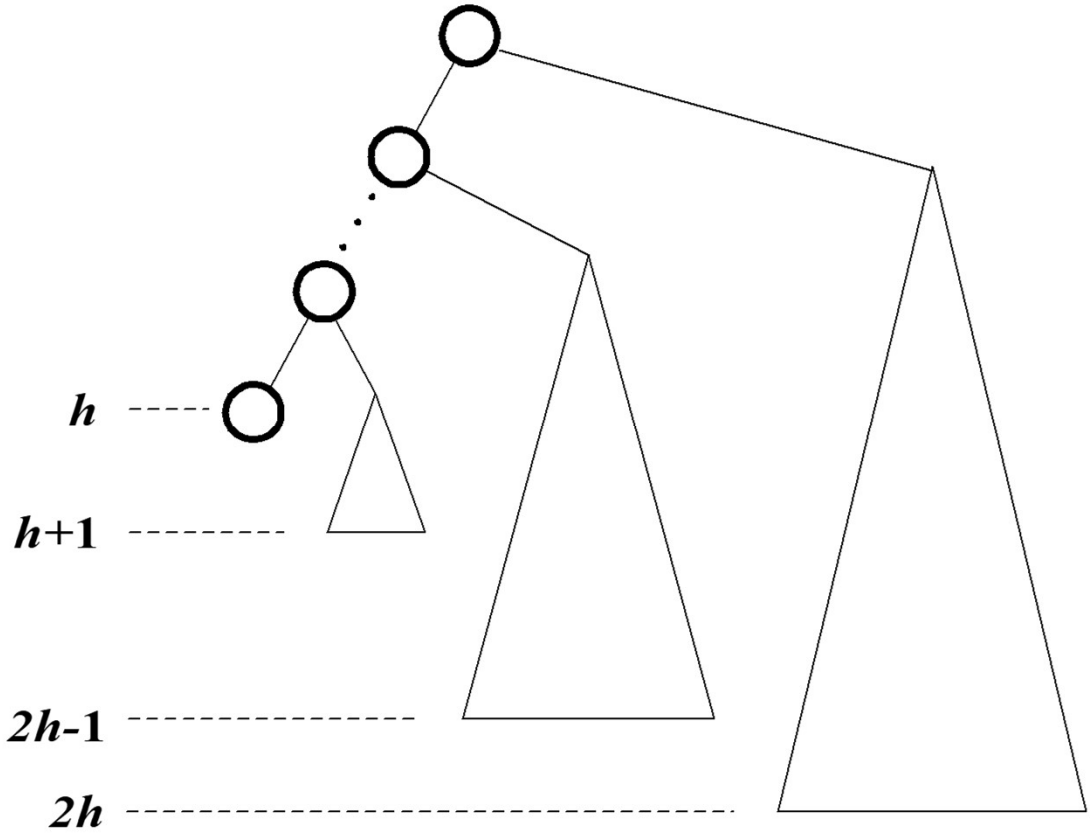
- Balansirano stablo
  - ✓ optimalno za pretraživanje
  - ✓ održavanje balansa može biti skupo
- Suboptimalno rešenje – “skoro balansirana” stabla



AVL stabla

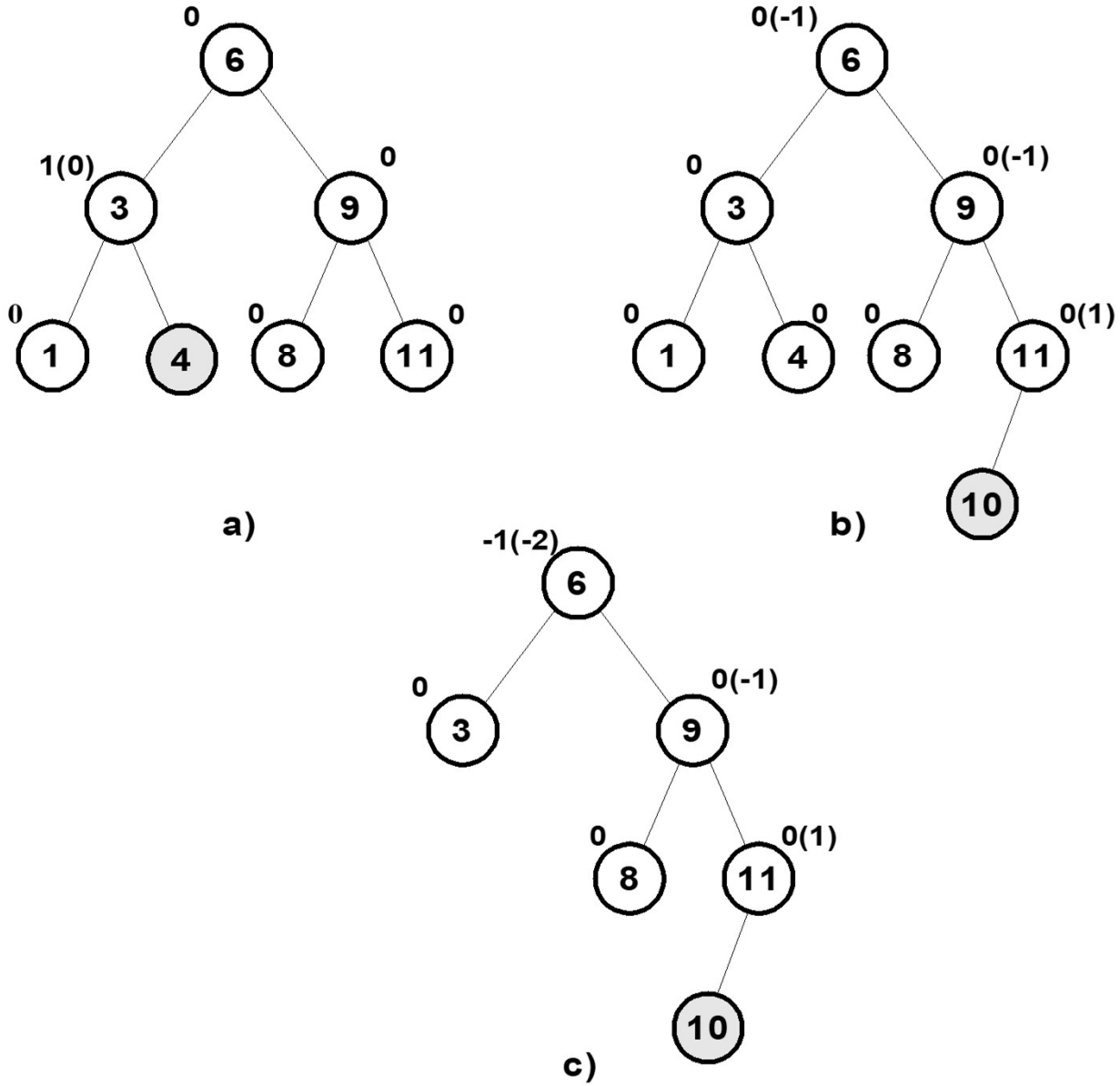
- **A**delson-**V**elski, **L**andis
- Balans čvora – $b = h_l - h_r$
- Visinski balansirano stablo (AVL)
  - ✓ stablo binarnog pretraživanja
  - ✓ za svaki čvor b može biti samo –1, 0 ili 1

AVL stabla



- AVL stablo sa najvećim rasponom nivoa

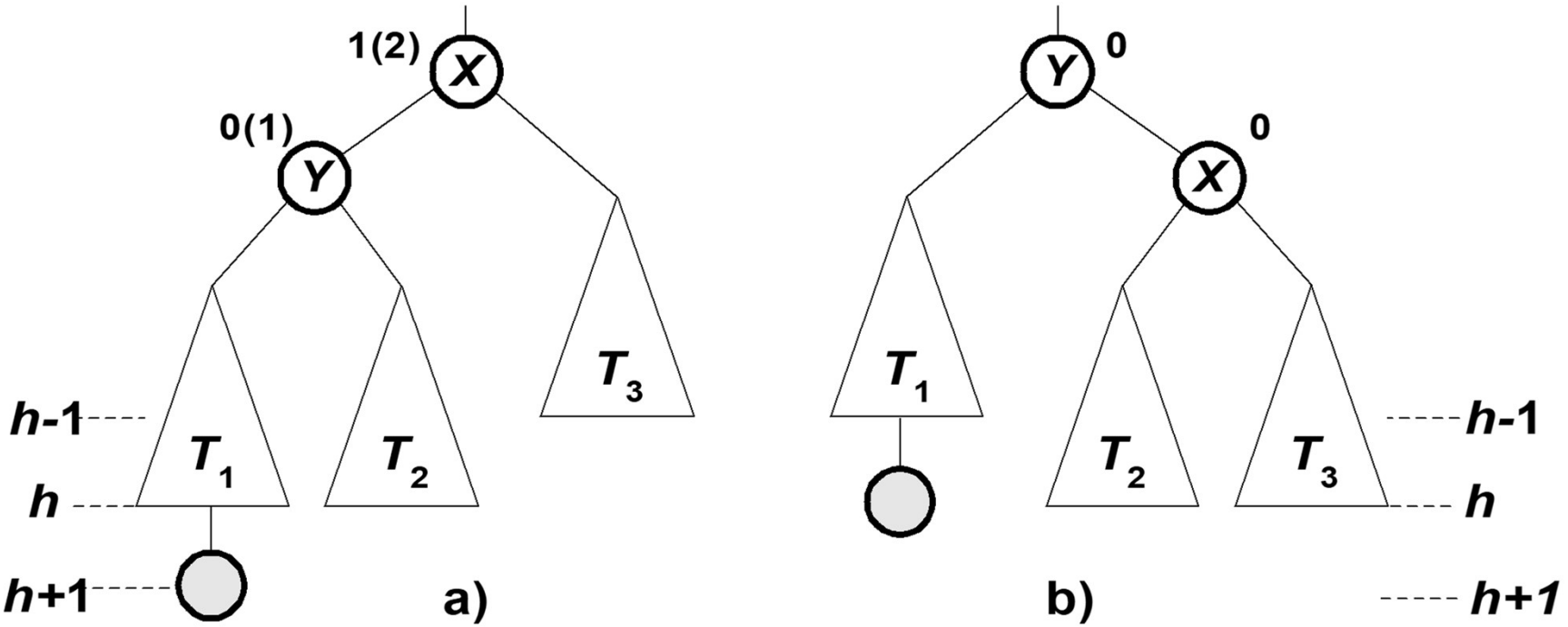
Umetanje u AVL stablo



Umetanje u AVL stablo

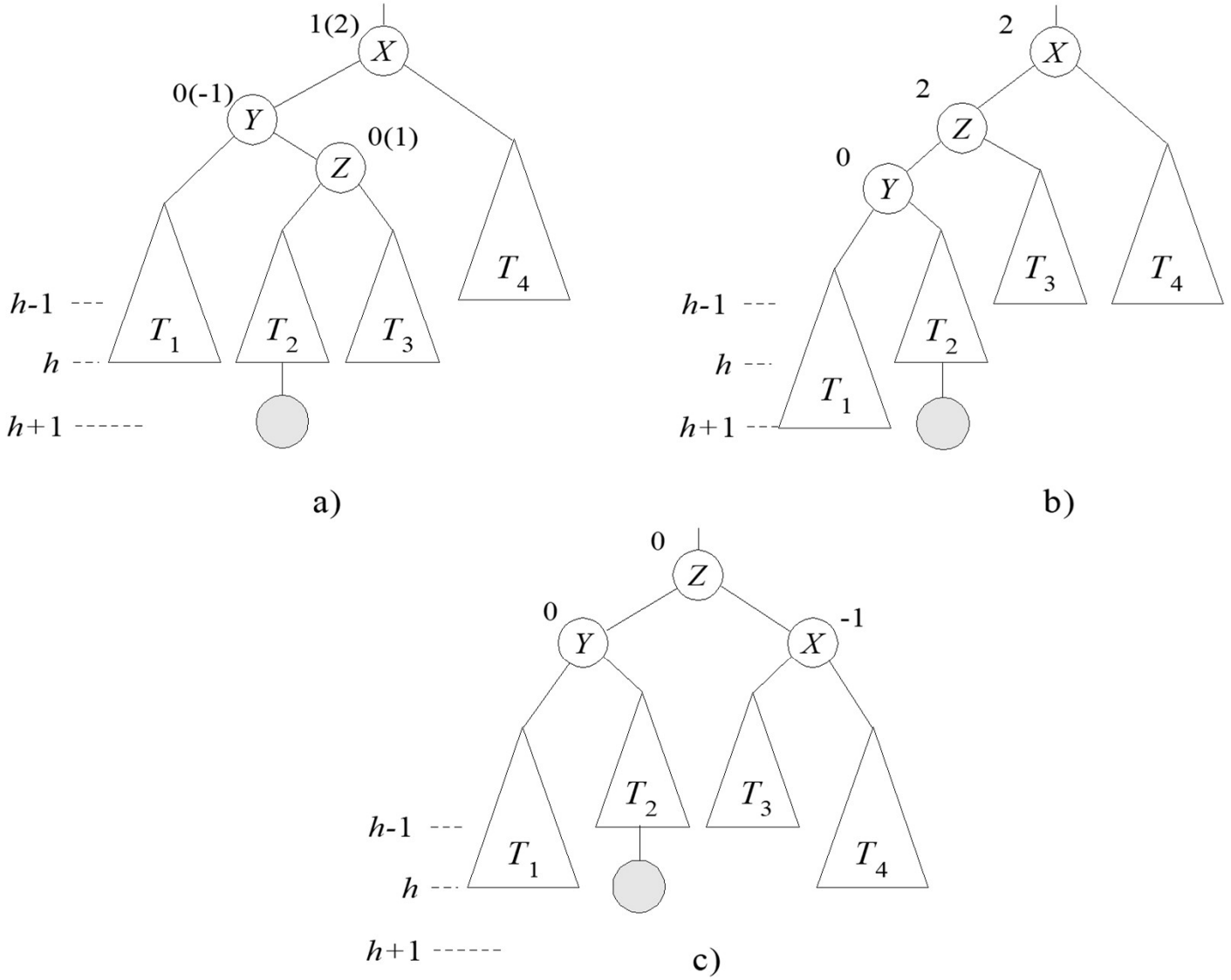
- Umetanje kao u BST
- Kritični čvor
- Dva karakteristična slučajeva
  - ✓ kritični čvor i njegov sin na strani umetanja se naginju na istu stranu
  - ✓ kritični čvor i njegov sin na strani umetanja se naginju na suprotnu stranu
- Rotacije (jednostruke ili dvostruke) oko kritičnog čvora
  - ✓ ispravljaju balans
  - ✓ održavaju *inorder* poredak

Jednostruka rotacija



RIGHT-ROTATION(*x*)
y = *left*(*x*)
temp = *right*(*y*)
right(*y*) = *x*
left(*x*) = *temp*

Dvostruka rotacija



Performanse AVL stabla

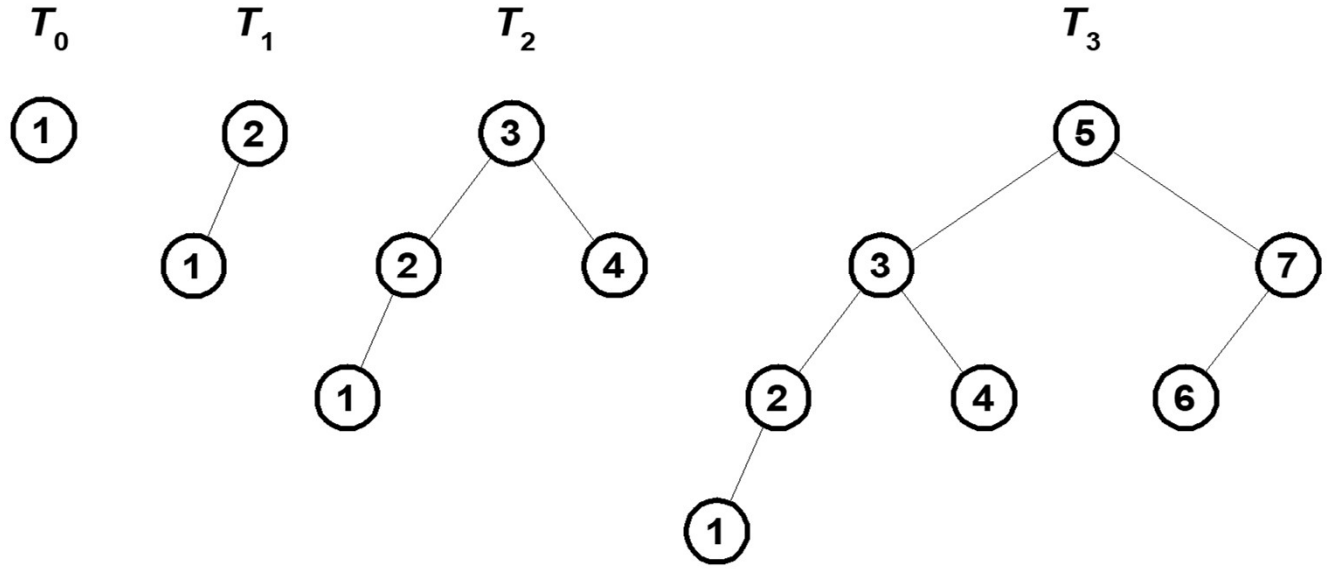
- Održavanje relativno jednostavno
  - ✓ najviše jedna dvostruka rotacija kod umetanja
  - ✓ najviše jedna rotacija po nivou kod brisanja

- Performansa pretraživanja
  $\log (n + 1) < h_{AVL}(n) < 1.4404 \log (n + 2) - 0.328$

- Zaključak
  - ✓ Garantovana performansa pretraživanja malo ispod optimalne
  - ✓ Jeftine i ne mnogo česte operacije balansiranja

Fibonacci-jeva stabla

- Najveća visina za dati broj čvorova *n*
- $n_0 = 1,\ n_1 = 2, \ldots, n_h = n_{h-1} + 1 + n_{h-2}$
- Balans čvorova grananja 1
- Svako brisanje smanjuje visinu



Težinski balansirana stabla

- "Skoro" balansirana stabla
- Kriterijum - težina podstabla (broj eksternih čvorova)
- Za svaki čvor odnos težina podstabla i stabla u opsegu a i $1-a$
- Balansiranje preko rotacija
- Za $a \approx 0.5$ dobar balans, ali zahtevnije održavanje
- Performanse - $O(\log n)$

Samopodešavajuća stabla

- Jedna vrsta BST (Sleator,Tarjan – *splay trees*)
- Cilj – brz pristup skoro pristupanim ključevima
  - ✓ implementacija *cache* i *garbage collection*
- Reorganizacija bez eksplicitnog kriterijuma balansa
- Prilikom svake operacije – umetanja, brisanja i pretraživanja čvor se dovodi u koren stabla
- Podešavanje preko rotacija slično kao u AVL:
  - ✓ duž putanje od ključa do korena
  - ✓ realizuje se operacijom *splay* – tzv. širenje čvora

Samopodešavajuća stabla

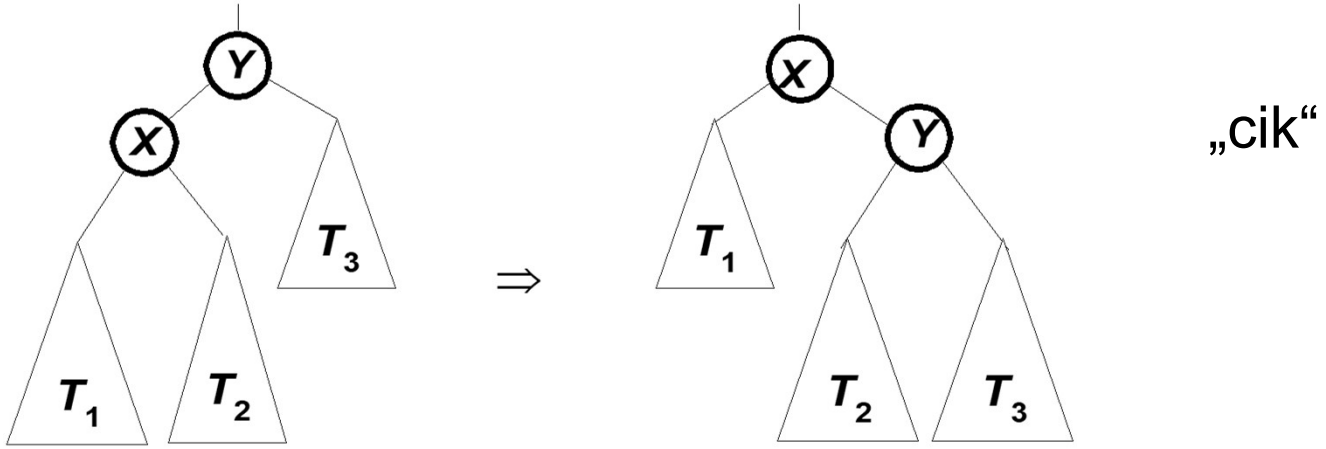
- *Splay* čvora *x* - reorganizacija čvorova tako da čvor *x* postane koren

- Širenje čvora se vrši u koracima
  - ✓ svaki korak se zasniva na rotacijama
  - ✓ leva i desna jednostruka rotacija – „cak“ i „cik“ (eng. „*zag*“ i „*zig*“)
  - ✓ identične operacije kao kod AVL stabla
  - ✓ u zavisnosti od pozicije ključa koji se „širi“, postoje četiri karakteristične situacije

Samopodešavajuća stabla

- Širi se čvor *x*
  - ✓ *y* – otac čvora *x*
  - ✓ *z* – deda čvor *x*

1. *x* je koren – finalno stanje stabla
2. *x* je levi ili desni sin korena *(y* je koren*):*
   - ✓ jedna rotacija u desno („cik“) ili
   - ✓ jedna rotacije u levo („cak“)



„cik“

Samopodešavajuća stabla

3. y nije koren, *y* i *x* su oba ili levi ili desni sinovi

- ✓ dve rotacije - „cik-cik“ ili „cak-cak“ rotacija
- ✓ prvo se rotira oko čvora *z*, pa oko *y*

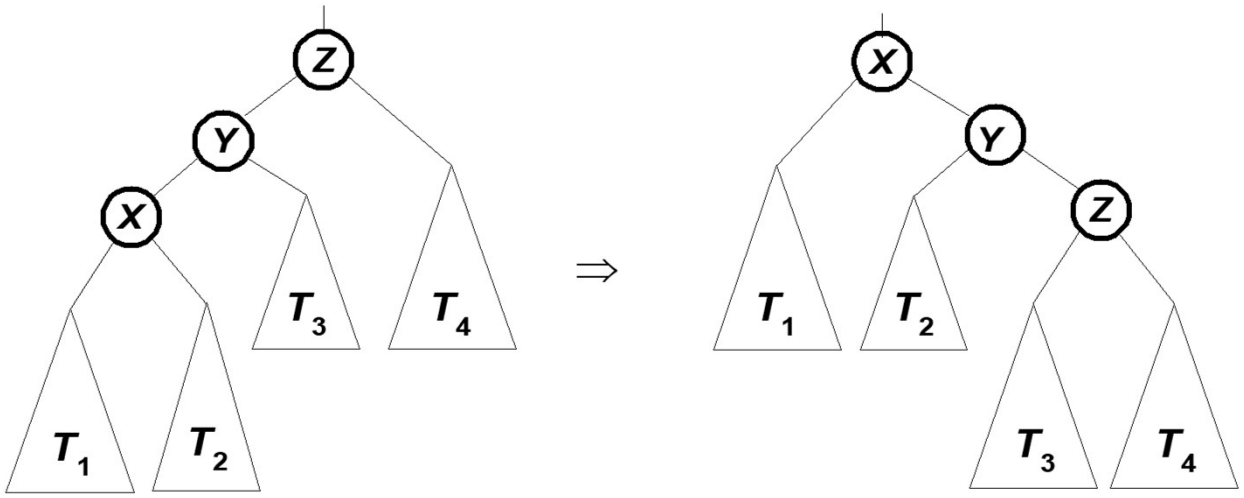
4. y nije koren, y je levi sin, a x je desni, ili obrnuto

- ✓ dve rotacije - „cik-cak“ ili „cak-cik“ rotacija
- ✓ prvo se rotira oko čvora *y*, pa oko *z* – slično AVL

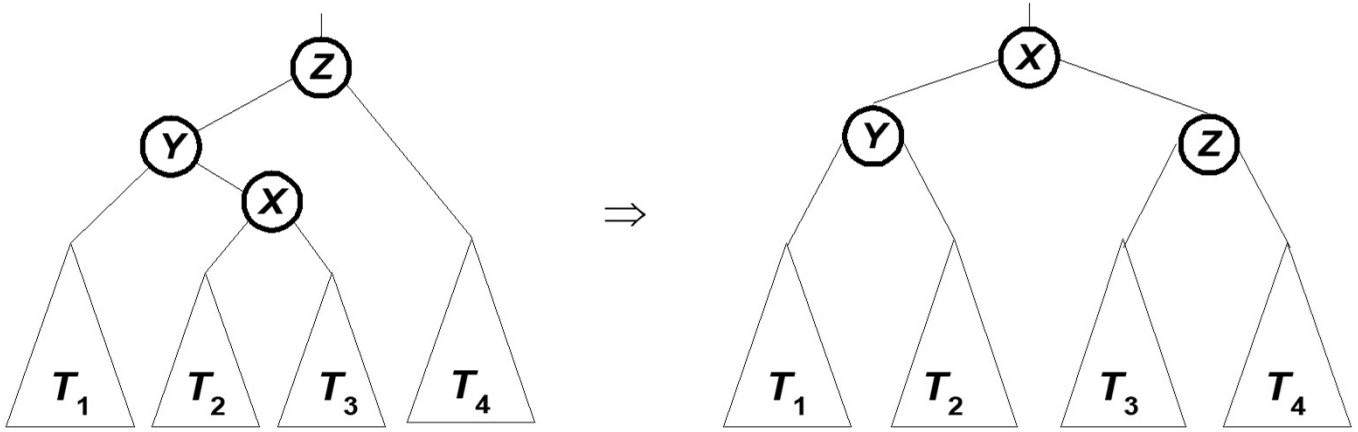
➢ Performanse:

- ✓ pojedinačni pristup može biti vrlo skup zbog trenutne nebalansiranosti stabla
- ✓ prosečna cena pristupa je $O(\log n)$ – što je prosečna visina stabla i amortizovano vreme
- ✓ u najgorem slučaju O(n) - malo verovatno

Samopodešavajuća stabla



"cik-cik"



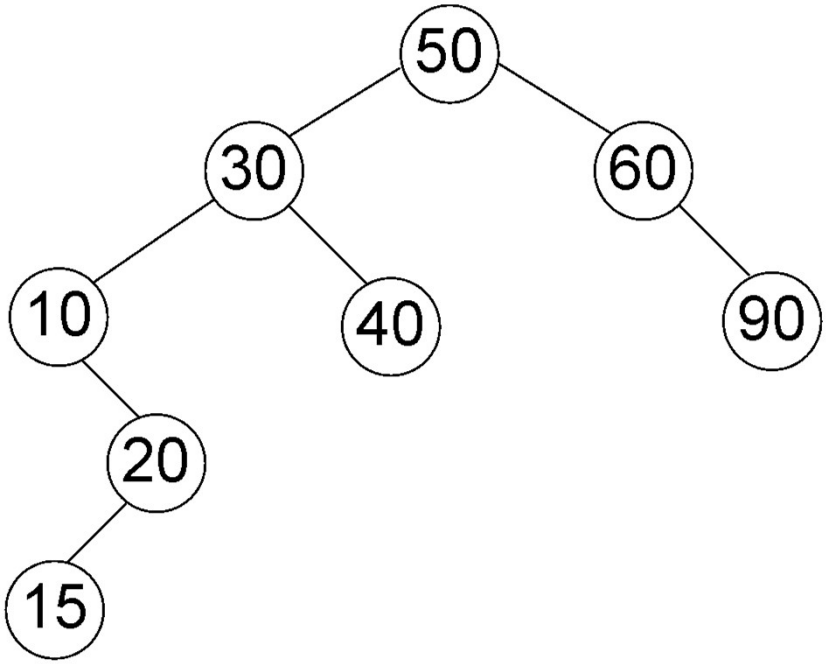
"cak-cik"

Samopodešavajuća stabla

```
SPLAY(root, x)
while parent(x) ≠ nil do { x nije koren – potrebno ažuriranje }
    if parent(parent(x)) = nil then { roditelj x je koren – jedna rotacija }
        if left(parent(x)) = x then RIGHT-ROTATION(parent(x))
        else LEFT-ROTATION(parent(x))
    else if left(parent(x)) = x AND left(parent(parent(x)) = parent(x) then
        RIGHT-ROTATION(parent(parent(x))
        RIGHT-ROTATION(parent(x)) { „cik-cik“ }
    else if right(parent(x)) = x AND right(parent(parent(x)) = parent(x) then
        LEFT-ROTATION(parent(parent(x))
        LEFT-ROTATION(parent(x)) { „cak-cak“ }
    else if left(parent(x)) = x AND right(parent(parent(x)) = parent(x) then
        RIGHT-ROTATION(parent(x))
        LEFT-ROTATION(parent(x)) { „cik-cak“ }
    else
        LEFT-ROTATION(parent(x))
        RIGHT-ROTATION(parent(x)) { „cak-cik“ }
    end_if
end_while
root = x
```

Samopodešavajuća stabla

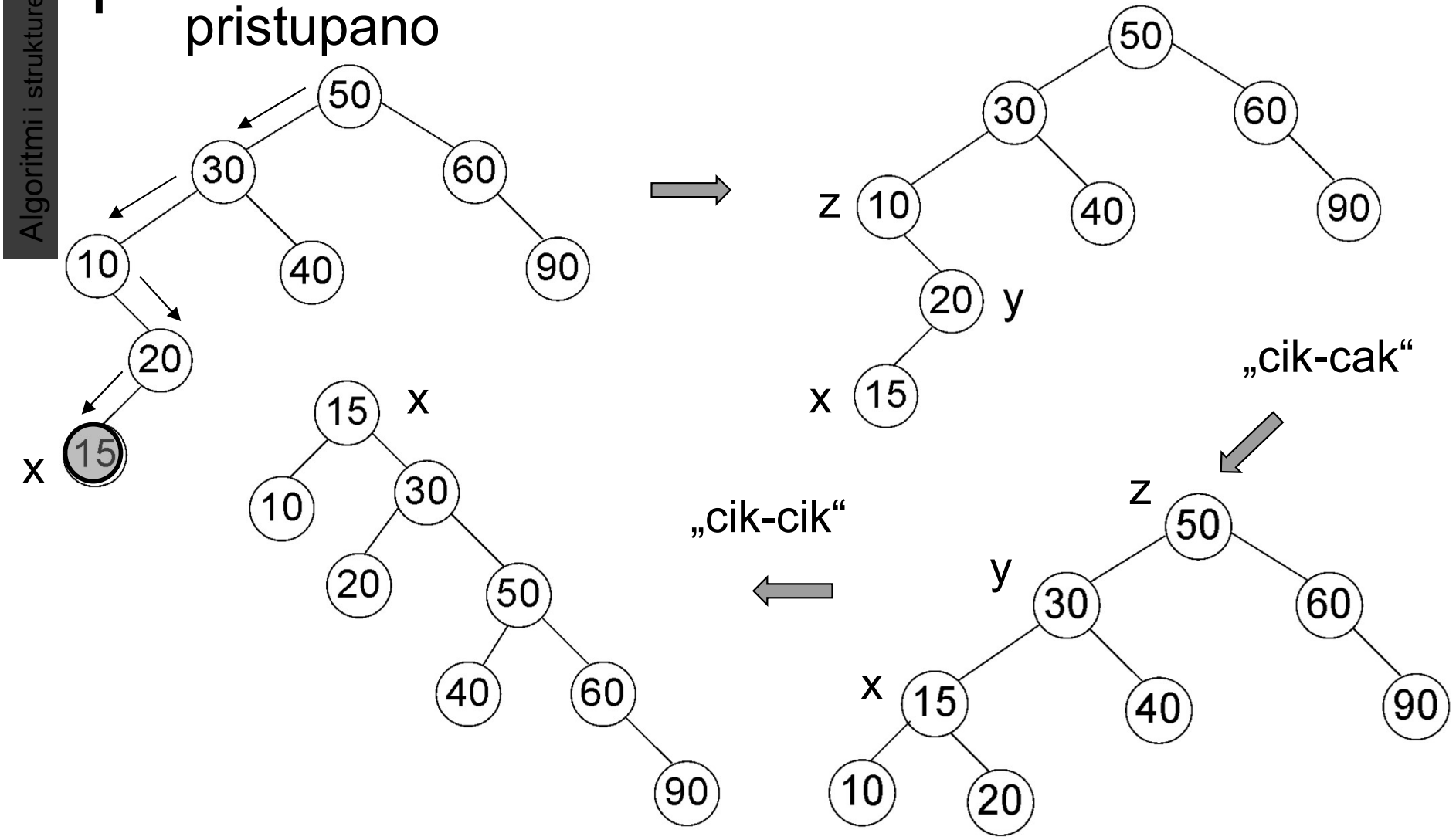
- Pretraga se vrši na isti način kao u BST
  - ✓ nakon pretrage sprovodi se *splay* operacija za čvor kome se poslednje pristupalo
  - ✓ za uspešnu pretragu – čvor koji sadrži ključ
  - ✓ za neuspešnu – čvor kod koga je završena pretraga



search 15

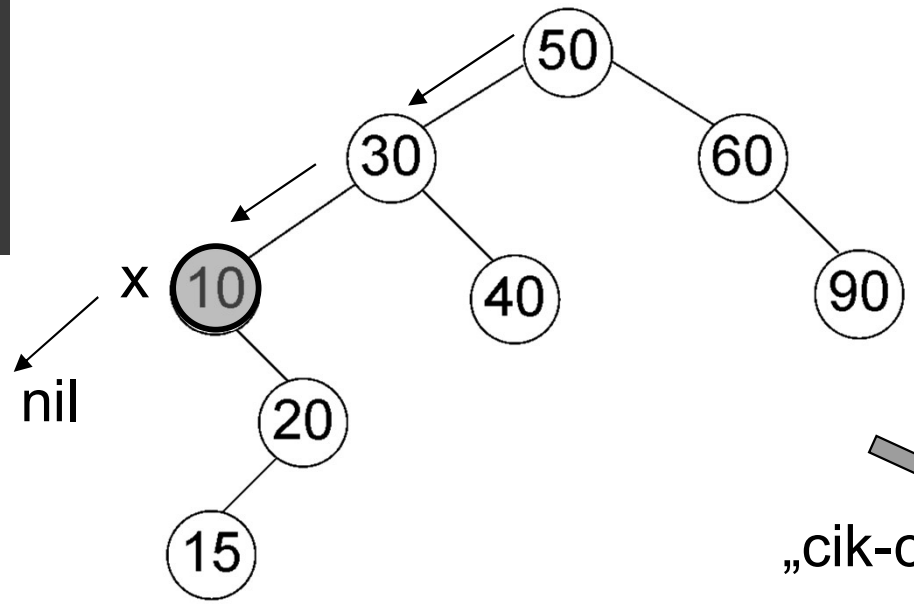
Samopodešavajuća stabla

- Širi se čvor sa ključem 15 – poslednji čvor kome je pristupano



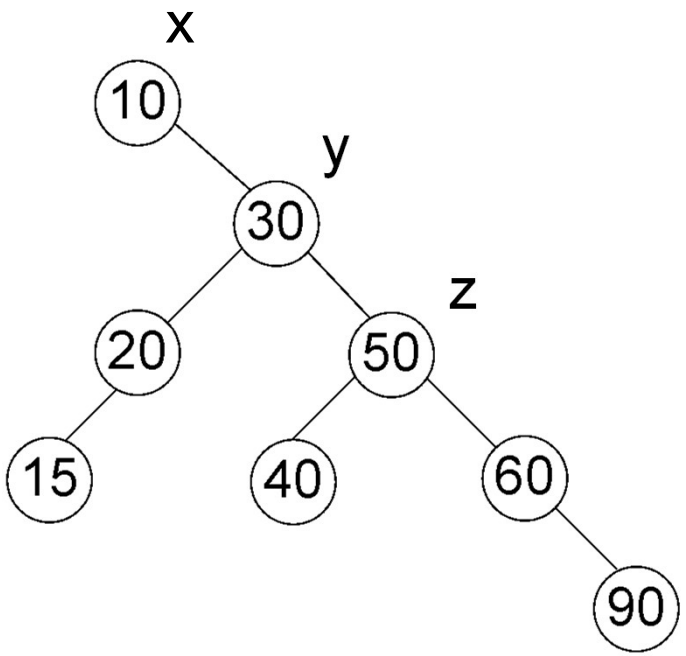
Samopodešavajuća stabla

- Širi se čvor sa ključem 10 – poslednji čvor kome je pristupano



search 5

„cik-cik“



Samopodešavajuća stabla

- Umetanje počinje kao u BST
  - ✓ nakon umetanja novi čvor se širi
  - ✓ ako je ključ postojao, širi se čvor koji ga sadrži

```
INSERT(root, k)
p = nil
next = root
while next ≠ nil do
    p = next
    if key(next) = k then
            break
    else if key(next) < k then
            next = right(next)
    else
            next = left(next)
    end_if
end_while
```

```
if next = nil then
    next = GETNODE
    key(next) = k
    left(next) = right(next) = nil
    if p = nil then              root = next
    else if key(p) < k then      right(p) = next
    else                         left(p) = next
    end_if
end_if
SPLAY(root, next)
```

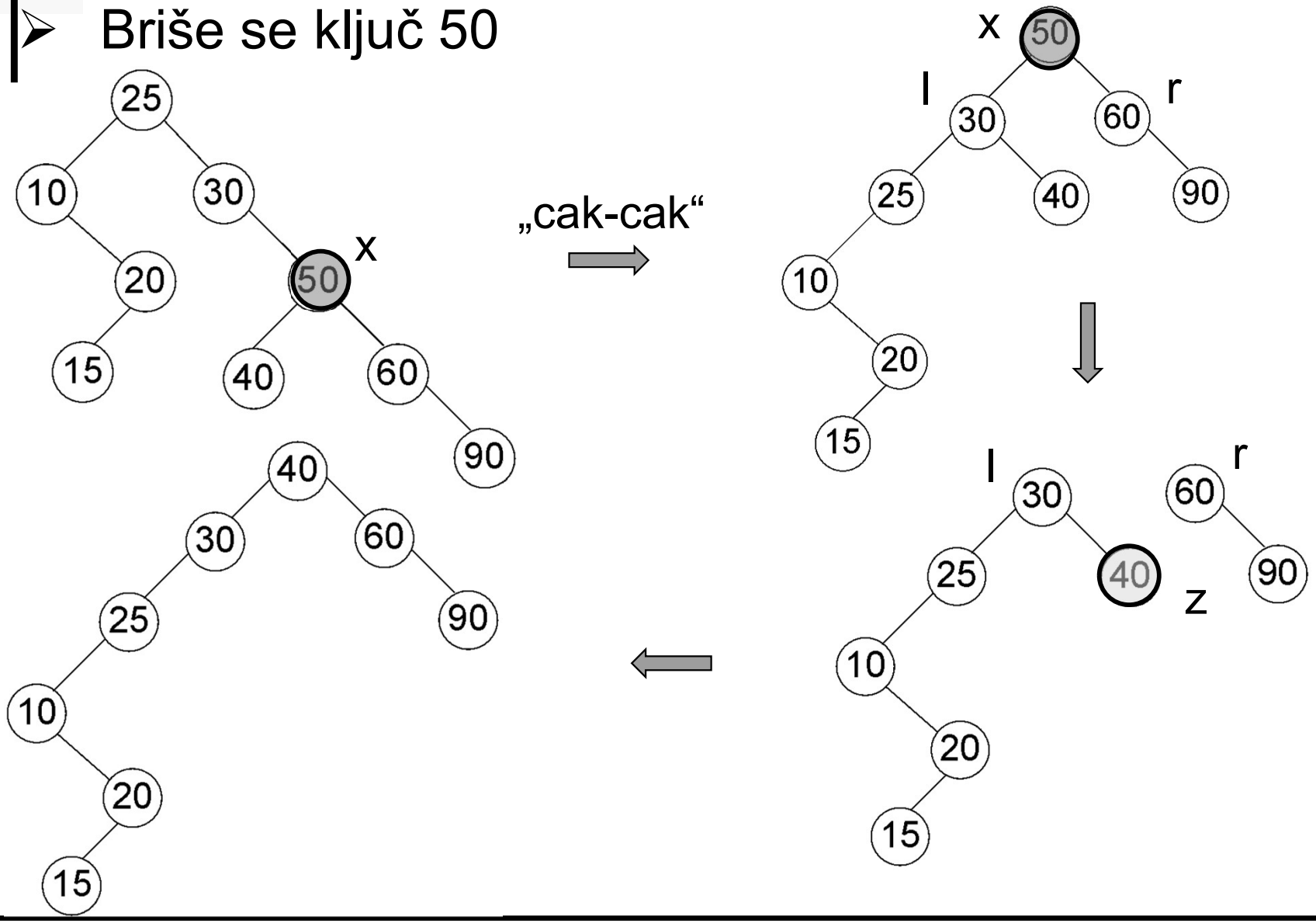
Samopodešavajuća stabla

- Brisanje počinje kao u BST
  - ✓ širi se čvor unutar koga je pronađen ključ, pa se potom sprovodi brisanje
  - ✓ ako ključ nije pronađen, širi se poslednji čvor kojem se pristupalo i ne dolazi do brisanja
- Pronađen čvor *x* se iz korena uklanja i dodatno:

1. *x* nema levog ili desnog sina:
   - ✓ postojeći sin postaje novi koren
2. *x* ima oba sina:
   - ✓ *l* – pokazivač na levo podstablo, *r* – na desno
   - ✓ prethodnik po inorderu *z* – čvor sa najvećim ključem u levom podstablu se širi
   - ✓ *z* – novi koren, *r* – desni potomak čvora *z*

Samopodešavajuća stabla

- Briše se ključ 50



Samopodešavajuća stabla

```
DELETE(root, k)
if root = nil then
    return
end_if
p = nil
next = root
while next ≠ nil do
    p = next
    if key(next) = k then
        break
    else if key(next) < k then
        next = right(next)
    else
        next = left(next)
    end_if
end_while
if next = nil then
    next = p
    { nema ključa k, radimo širenje
    za poslednji ne-nil čvor }
end_if
```

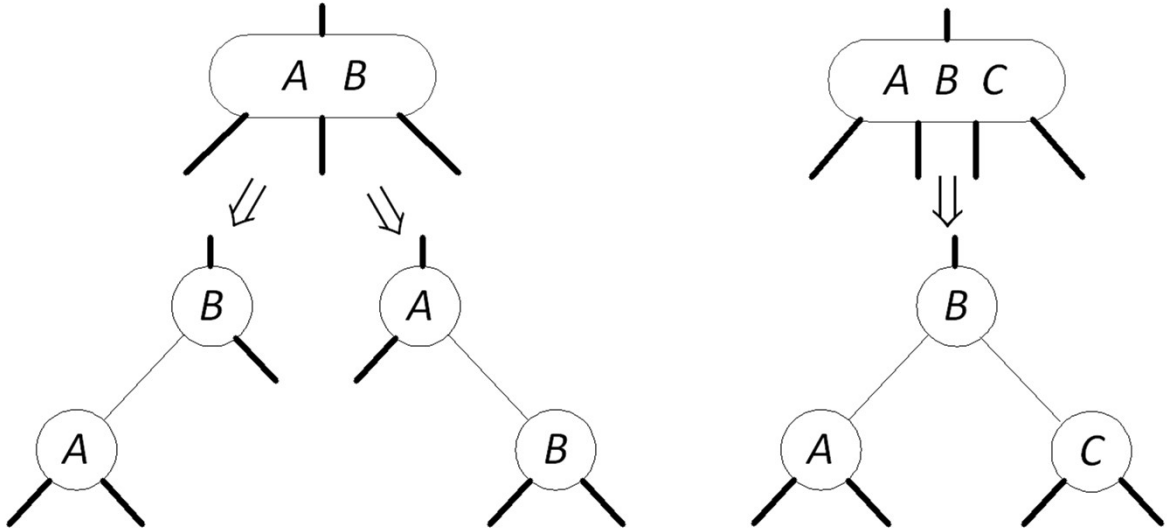
```
SPLAY(root, next)
if key(root) = k then
    { postoji ključ koji želimo da obrišemo }
    tmp = root
    if left(root) = nil then
        root = right(root)
    else if right(root) = nil then
        root = left(root)
    else
        l = left(root)
        r = right(root)
        FREENODE(tmp)
        z = BST-MAX(l)
        SPLAY(l, z)
        right(z) = r
        root = z
    end_if
end_if
```

2-3 stabla

- Hopcroft
- Dve vrste čvorova:
  - ✓ 2-čvor (jedan ključ i dva podstabla)
  - ✓ 3-čvor (dva ključa i tri podstabla)
- Svi listovi na istom nivou – balansiranost
- Umetanje
  - ✓ pretvara 2-čvor u 3-čvor
  - ✓ prelama 3-čvor na dva 2-čvora i šalje jedan ključ u oca (moguća propagacija)
- Balansiranje bez eksplicitnog kriterijuma

Crveno-crna stabla

- Ekstrapolacija – 2-3-4 stabla
- Implementacija preko standardnih binarnih stabala
- Predstavljanje 3-čvora i 4-čvora binarnim stablima sa 2-čvorovima

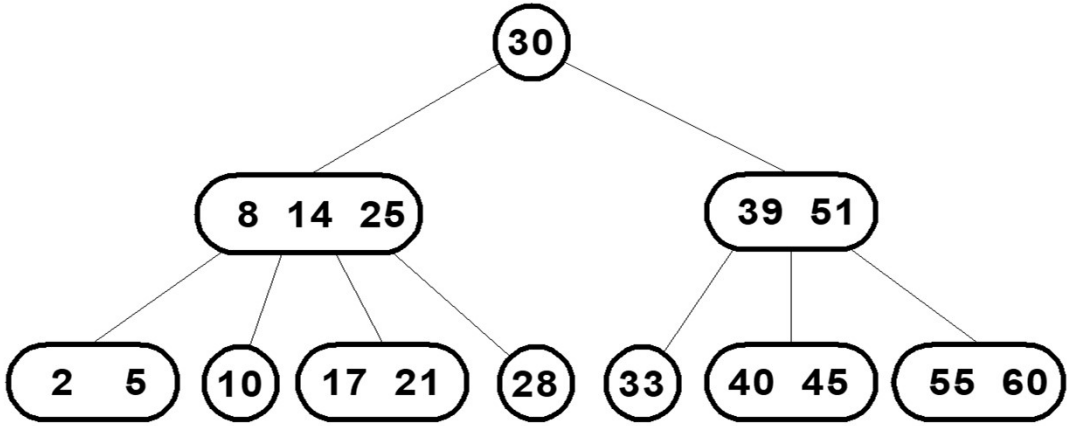


Crveno-crna stabla

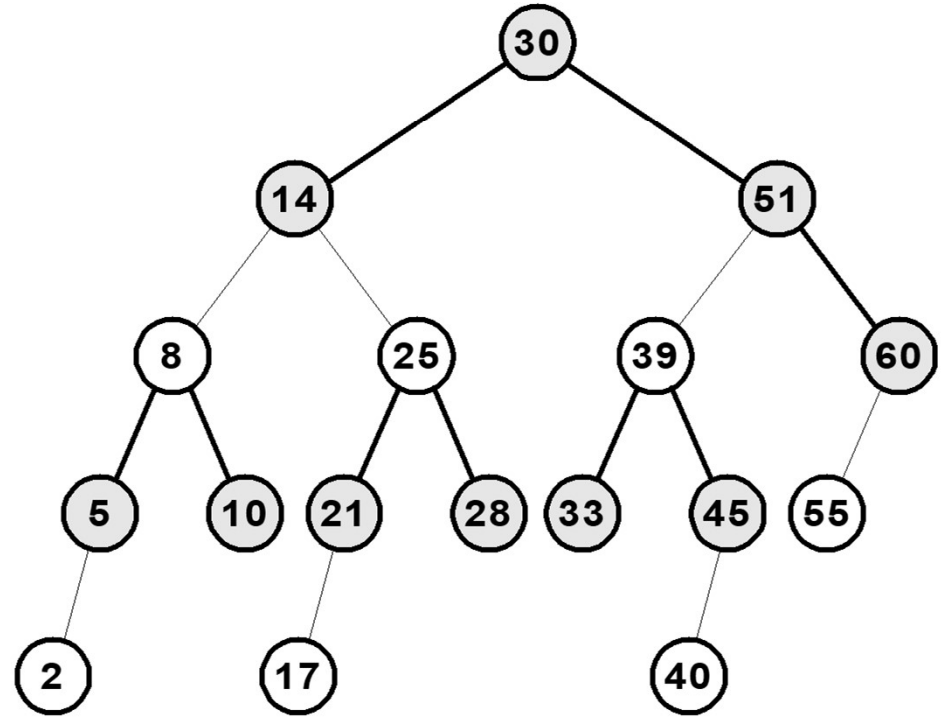
- Grane 2-3-4 stabla crne (vode do crnih čvorova)
- Grane umetnutih binarnih stabla crvene (vode do crvenih čvorova)
- Definicija
  - ✓ svaki čvor stabla je crven ili crn
  - ✓ koren je crn
  - ✓ eksterni čvor je crn
  - ✓ sinovi crvenog čvora su crni
  - ✓ svaki prost put niz stablo od nekog čvora stabla do listova ima jednak broj crnih čvorova

Crveno-crna stabla

a)



b)



Crveno-crna stabla

- Pretraživanje isto kao u BST

- Umetanje i brisanje mogu narušiti svojstva stabla
  - ✓ mehanizam održavanja – rotacije
  - ✓ najviše 2 rotacije kod umetanja, a 3 kod brisanja

- Implementacija:
  - ✓ dodatni bit u čvoru označava boju
  - ✓ eksterni čvor (npr., oznaka – NIL)
  - ✓ svi eksterni čvorovi se mogu predstaviti jednim fizičkim čvorom - graničnikom
  - ✓ pokazivač na oca

Crveno-crna stabla

- "Crna" visina čvora *(bh)* je broj crnih čvorova na putu niz stablo od tog čvora do listova

- Odnos dužine bilo koja dva puta od korena do listova nije veći od 2 !
  - ✓ najkraći – samo B, najduži – naizmenično B i R

- Najveća visina RB stabla je $2 \log(n + 1)$

  Dokaz:
  - ✓ broj čvorova u stablu $n >= 2^{bh} - 1$
  - ✓ $bh >= h/2$
  - ✓ $n >= 2^{h/2} - 1$
  - ✓ $h < 2 \log(n + 1)$

Crveno-crna stabla

- Izomorfizam operacija sa 2-3-4 stablom
- Garantovane performanse $O(\log n)$
- Primene u vremenski kritičnim aplikacijama:
  - ✓ strukture sistemskog SW
  - ✓ funkcionalno programiranje
  - ✓ implementacija skupova u STL C++

Optimalno BST

- Pretpostavke
  - ✓ n ključeva $K_1 < K_2 < \ldots < K_n$
  - ✓ p_i – verovatnoća uspešne pretrage na K_i
  - ✓ q_i – verovatnoća neuspešne pretrage u intervalu između K_i i K_{i+1}

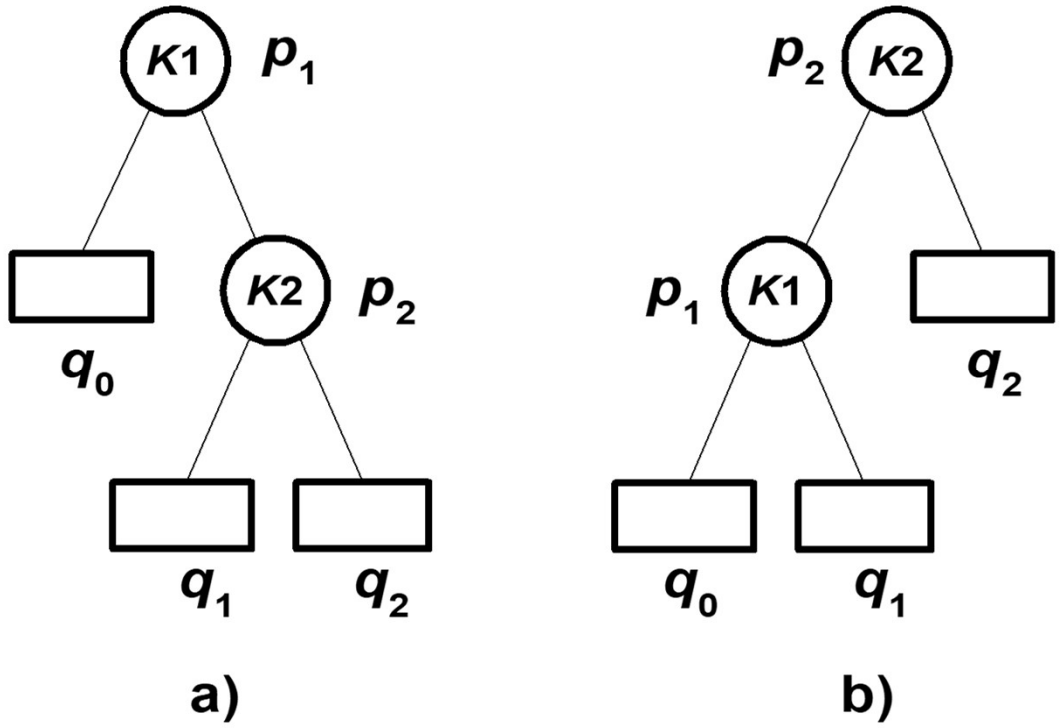
$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

$$C = \sum_{i=1}^{n} p_i(h_i + 1) + \sum_{i=0}^{n} q_i h_i$$

Cena stabla

- Minimalna cena $C \Rightarrow$ optimalno BST

Optimalno BST



- Metod “grube sile” neprihvatljiv
- Sva podstabla optimalnog stabla su optimalna
- Algoritam inkrementalno gradi stablo

Optimalno BST

$w_{i\,j} = q_i + p_{i+1} + q_{i+1} + \ldots + p_j + q_j$ za optimalno podstablo T_{ij}

$$C_{i\,j} = \sum_{m=i+1}^{j} p_m(h_m + 1) + \sum_{m=i}^{j} q_m h_m$$

$$C_{i\,j} = \sum_{m=i+1}^{j} p_m(h_m' + 1) + \sum_{m=i}^{j} q_m h_m' + \sum_{m=i+1}^{j} p_m + \sum_{m=i}^{j} q_m \qquad \text{koren } r_{ij} = k$$

$$C_{i\,j} = \sum_{m=i+1}^{k-1} p_m(h_m' + 1) + p_k * 0 + \sum_{m=k+1}^{j} p_m(h_m' + 1) + \sum_{m=i}^{k-1} q_m h_m' + \sum_{m=k}^{j} q_m h_m' + w_{i\,j}$$

$$C_{i\,j} = C_{i,k-1} + C_{k,j} + w_{i\,j}$$

$$C_{i\,j} = \min(C_{i,k-1} + C_{k,j}) + w_{i\,j} \qquad \text{za } i < k \leq j$$

Optimalno BST

OPT-TREE(p_i, q_i, n)
for i = 1 **to** n - 1 **do**
    $(w_{ii}, r_{ii}, C_{ii}) \leftarrow (q_i, 0, 0)$
    $(w_{i,i+1}, r_{i,i+1}, C_{i,i+1}) \leftarrow (q_i + q_{i+1} + p_{i+1}, i + 1, q_i + q_{i+1} + p_{i+1})$
end_for
$(w_{nn}, r_{nn}, C_{nn}) \leftarrow (q_n, 0, 0)$
for m = 2 **to** n **do**
    for i = 0 **to** n - m **do**
        $j = i + m$
        $w_{ij} = w_{i,j-1} + p_j + q_j$
        find k iz $r_{i,j-1} \leq k \leq r_{i+1,j}$ => min $(C_{i,k-1} + C_{k,j})$
        $C_{ij} = C_{i,k-1} + C_{k,j} + w_{ij}$
        $r_{ij} = k$
    end_for
end_for

Suboptimalna BST

- Strategije manje složenosti za nalaženje stabla sa performansama približnim optimalnim:

A. Umetanje ključeva po opadajućim verovatnoćama:
   - ✓ ne uzima u obzir neuspešno pretraživanje
   - ✓ ne garantuje optimalnost

B. Koren se bira tako da je razlika težina levog i desnog podstabla minimalna:
   - ✓ formira stablo od korena nadole
   - ✓ može izabrati ključ sa malom verovatnoćom kao koren

Suboptimalna BST

C. Kombinovani pristup (Walker, Gotlieb):
- ✓ minimizira razliku težina podstabala
- ✓ u blizini korena traži čvor sa max verovatnoćom pa ažurira stablo
- ✓ na nižim nivoima optimalan algoritam
- ✓ složenost $O(n \log n)$, performanse lošije za 2-3%

D. Optimizacija pod posebnim uslovima (Hu, Tucker):
- ✓ postoje samo neuspešna pretraživanja
- ✓ kombinuju se samo susedni eksterni čvorovi
- ✓ složenost $O(n \log n)$

Primene BST

- Predstavljanje prioritetnog reda
- Vektorska i ulančana reprezentacija neefikasne - $O(n)$
- Realizacija preko stabla binarnog pretraživanja uz omogućavanje postojanja istih ključeva
- Performanse operacija umetanja i brisanja - $O(\log n)$ ako se stablo održava balansiranim
- Fleksibilnije od *heap*-a:
  - ✓ istovremeno i rastući i opadajući red
  - ✓ efikasno brisanje niza ključeva iz zadatog opsega

III.3 Stabla opšteg pretraživanja

Stabla opšteg pretraživanja

- Generalizacija BST - 2-3, 2-3-4, ... stabla
  - ✓ manja visina
  - ✓ veći broj poređenja u čvoru
  - ✓ veći broj praznih pokazivača
- Efekat na performanse
  - ✓ u operativnoj memoriji nije sasvim izvestan
  - ✓ mnogo izraženiji na spoljašnjim memorijama
- Sistem spoljašnjih memorija
  - ✓ veliki kapacitet
  - ✓ permanentnost zapisa

Stabla opšteg pretraživanja

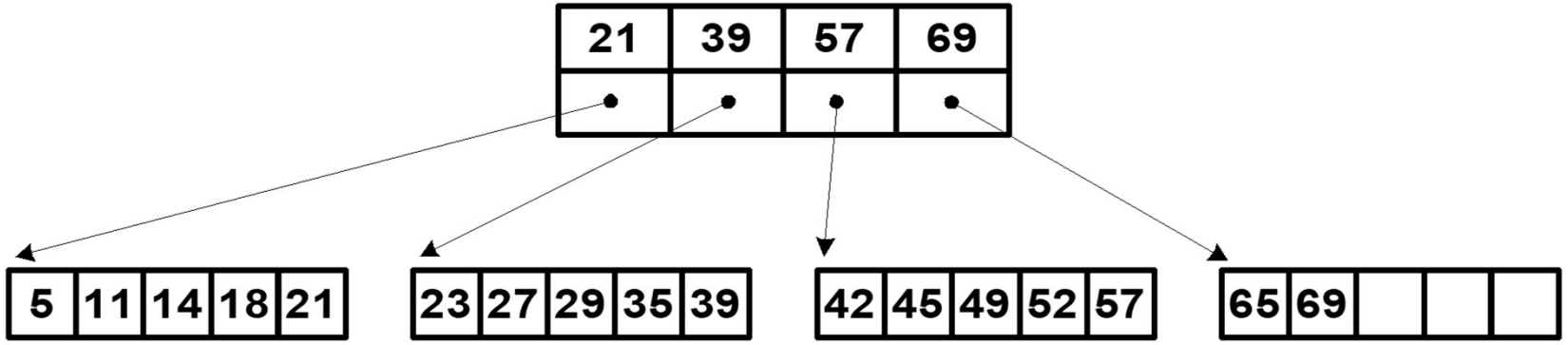
- Fizičke osobine i karakteristike pristupa disku
  - ✓ elektromehanički uređaji
  - ✓ pristup neuniforman i mnogo sporiji od OM
  - ✓ jednim pristupum se dovlači čitav blok
  - ✓ broj pristupa dominantno utiče na operacije

- Datoteka – organizaciona jedinica
  - ✓ zapisi uniformne ili neuniformne dužine
  - ✓ sekvencijalni ili direktni pristup (preko ključa)

- Indeksi – pomoćne strukture za ubrzanje pristupa
  - ✓ parovi “ključ-fizička adresa”
  - ✓ gusti i retki

Stabla opšteg pretraživanja

- Indeksi mogu biti vrlo veliki (na spoljašnjim memorijama)
- Kakva organizacija da bi bili
  - ✓ efikasni za pretraživanje
  - ✓ fleksibilni za održavanje
- Indeks u vidu tabele za sortiranu datoteku
  - ✓ ulaz za svaki blok datoteke
  - ✓ nefleksibilan pri promenama

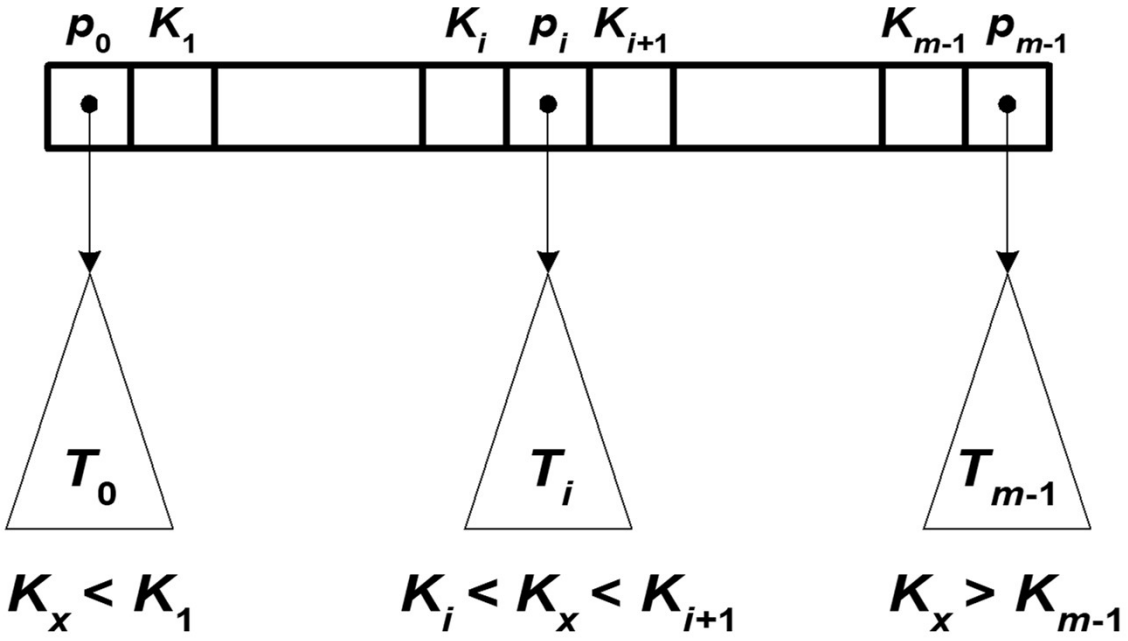


Stabla opšteg pretraživanja

- Struktura stabla daje fleksibilnost
- Stablo binarnog pretraživanja neefikasno zbog velikog broja pristupa
- Rešenje - stablo *m* -arnog pretraživanja
  - ✓ do *m* -1 ključeva u čvoru
  - ✓ manji broj pristupa
  - ✓ više poređenja u operativnoj memoriji
- Više varijanti (*m* vrlo veliko)

Stabla *m*-arnog pretraživanja

- Definicija
  - ✓ najviše m -1 ključeva i m pokazivača u čvoru
  - ✓ svaki ključ razdvaja dva pokazivača $p_{i-1} K_i p_i$
  - ✓ $K_1 < K_2 < \ldots < K_n$
  - ✓ $K_i < K_x < K_{i+1}$ u podstablu T_i
  - ✓ T_i - stablo m -arnog pretraživanja



Operacije

- Pretraživanje
  - ✓ sekvencijalno ili binarno u čvoru
  - ✓ neuspešno se završava u listu

- Inorder obilazak

```
INORDER-M(root)
if (root ≠ nil) then
      p =root
      for i = 0 to m - 2 do
            INORDER-M(p_i(p))
            PRINT(K_{i+1}(p))
      end_for
      INORDER-M(p_{m-1}(p))
end_if
```

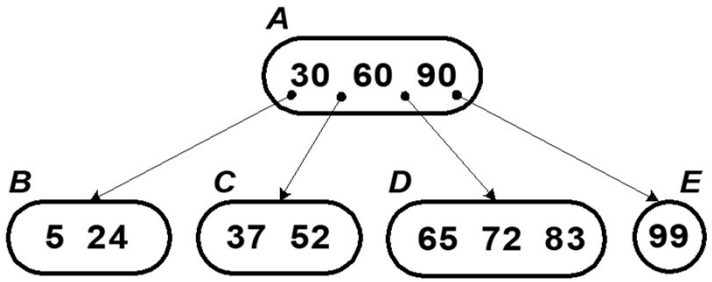
Operacije

- Umetanje ključa
  - ✓ uvek u list
  - ✓ ako nije pun, umetanje iz pomeranje većih
  - ✓ ako je pun, stvara se novi list
  - ✓ popunjavnje odozgo nadole (*top-down*)
- Brisanje ključa iz lista
  - ✓ uklanjanje uz sažimanje
  - ✓ ako je jedini u listu, uklanja se list
- Brisanje ključa iz ostalih čvorova
  - ✓ posle uklanjanja nedostaje jedan ključ
  - ✓ dopunjava se sledbenikom iz podstabla
  - ✓ može propagacija do lista

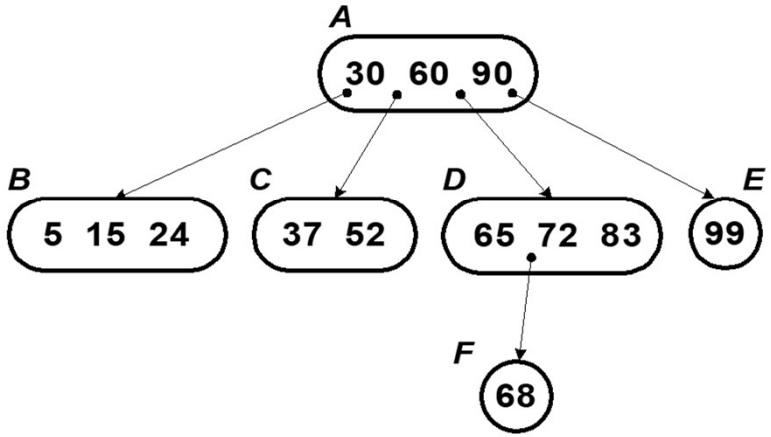
Operacije

+15 +68

a)

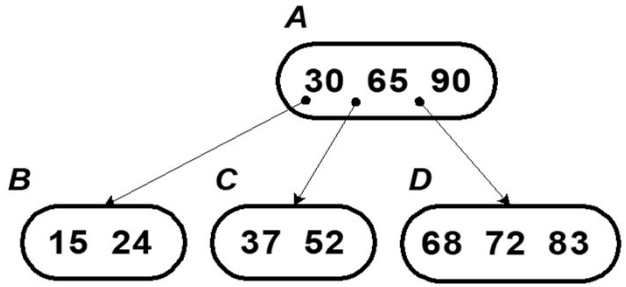


b)



- 5 - 60 – 99

c)



Osobine i performanse

- Osobine
  - ✓ listovi nisu na istom nivou
  - ✓ preci lista puni, ali može i samo jedan ključ u listu

- Peformanse
  - ✓ dominantan faktor - broj obraćanja disku
  - ✓ srazmerne visini stabla
  - ✓ najgori slučaj – $O(n)$
  - ✓ najbolji slučaj – $O(\log_m n)$

Osobine i performanse

$$\sum_{i=0}^{h} m^i = (m^{h+1} - 1)/(m - 1)$$

$$n_{max} = m^{h+1} - 1$$

$$m_{min} = (n + 1)^{1/(h+1)}$$

$$h_{min} = \lceil log_m(n + 1) \rceil - 1$$

- Izbor stepena stabla
  - ✓ za veće *m* - manja visina, ali više poređenja po čvoru
  - ✓ u blizini optimuma izbor nije previše kritičan

Implementacija

- Fiksna veličina čvora
  - ✓ prostija alokacija i način obraćanja
  - ✓ slabije iskorišćenje memorije (nije primarno)

- Ako se zapis smešta zajedno sa ključem
  - ✓ bez dodatnog pristupa podacima
  - ✓ manji stepen stabla ⇒ sporije pretraživanje

- Ako se zapis smešta odvojeno od ključa
  - ✓ fizička adresa zapisa i dodatni pristup podacima
  - ✓ veći stepen stabla ⇒ brže pretraživanje

Implementacija

- Organizacija čvora sa ključevima promenljive dužine
- Kontinualno smeštanje
  - ✓ štedi prostor
  - ✓ onemogućava binarno pretraživanje
  - ✓ pokazivači na ključeve
  - ✓ otežava reorganizaciju čvora pri operacijama
- Rezervisanje maksimalnog prostora za ključ
- Čeona kompresija – izdvajanje prvih istih znakova
  - ✓ prvi isti znakovi više ključeva se jednom pamte
  - ✓ ušteda prostora, a nekad i vremena

B-stabla

- Problemi sa stablom *m* -arnog pretraživanja:
  - ✓ popunjenost
  - ✓ nebalansiranost

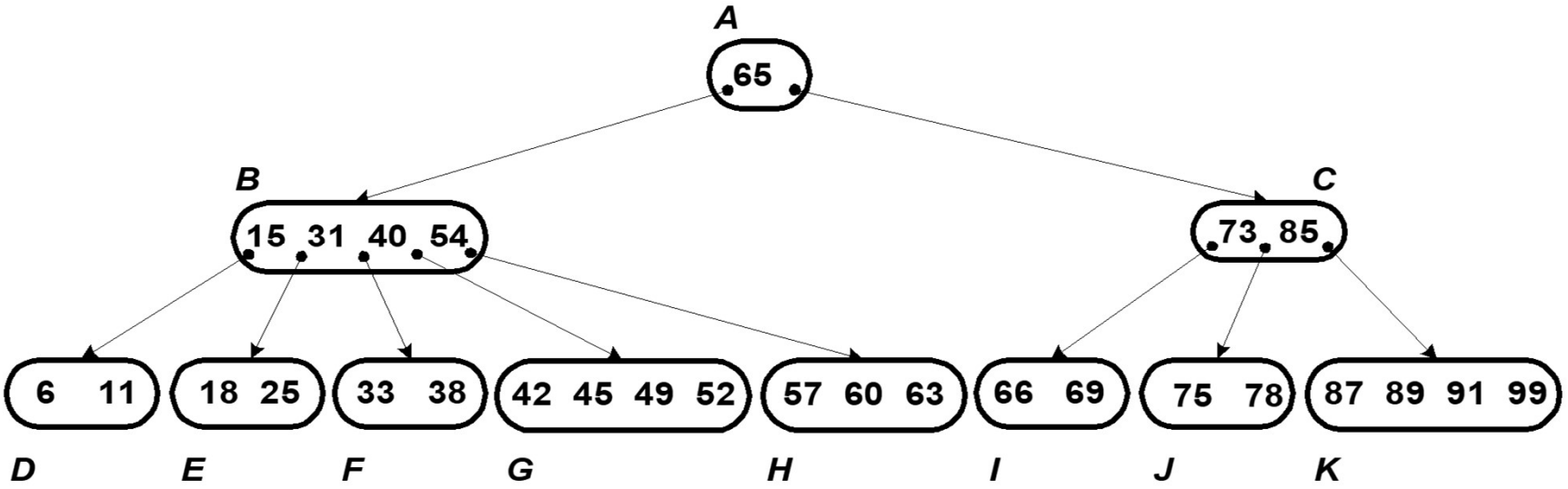
- *B*-stabla (Bayer, McCreight)

- Definicija:
  - ✓ stablo *m* -arnog pretraživanja
  - ✓ koren (ako nije list) ima najmanje dva podstabla
  - ✓ čvorovi grananja imaju najmanje $\lceil m/2 \rceil$ podstabala
  - ✓ listovi imaju najmanje $\lceil m/2 \rceil - 1$ ključeva
  - ✓ svi listovi na istom nivou

B-stabla

- Osobine:
  - ✓ garantovana 50% popunjenost i balansiranost
  - ✓ relativno jeftino održavanje

- Indeksne strukture u datotekama



Operacije

- Pretraživanje kao u stablu *m* -arnog pretraživanja
  - ✓ uspešno može da se završi na bilo kom nivou
  - ✓ neuspešno se završava u listovima

- Garantovane performanse

$$n \geq 1 + (\sum_{i=1}^{h} 2\lceil m/2 \rceil^{i-1})(\lceil m/2 \rceil - 1)$$

$$\sum_{i=1}^{h} \lceil m/2 \rceil^{i-1} = (\lceil m/2 \rceil^{h} - 1)/(\lceil m/2 \rceil - 1)$$

$$n \geq 1 + 2(\lceil m/2 \rceil^{h} - 1)$$

$$n + 1 \geq 2\lceil m/2 \rceil^{h}$$

$$h \leq \log_{\lceil m/2 \rceil}(n+1)/2$$

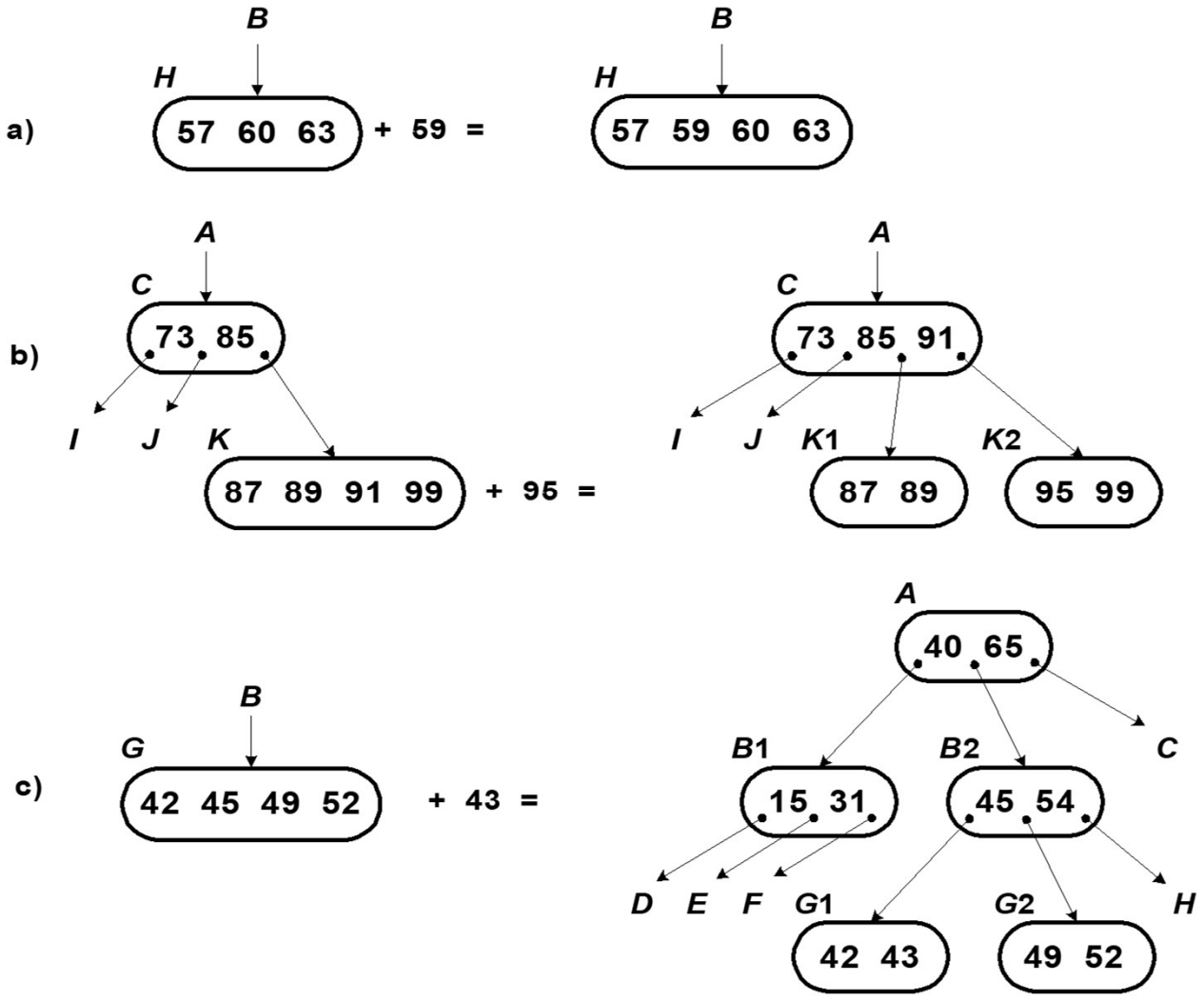
Operacije

- Umetanje
  - ✓ garantovana 50% popunjenost i balansiranost
  - ✓ relativno jeftino održavanje

- Algoritam
  - ✓ pokušava umetanje u list
  - ✓ ako list nije pun, umetanje uz pomeranje većih
  - ✓ ako je pun, prelamanje

- Prelom
  - ✓ manji ključevi idu stari list
  - ✓ veći ključevi idu u novi list
  - ✓ srednji ključ (na poziciji $\lceil m/2 \rceil$) migrira kod oca
  - ✓ eventualna propagacija do korena i njegov prelom

B-stabla


a)
B
H
57 60 63
+ 59 =
B
H
57 59 60 63
b)
A
C
73 85
I J K
87 89 91 99
+ 95 =
A
C
73 85 91
I J K1 K2
87 89
95 99
c)
B
G
42 45 49 52
+ 43 =
A
40 65
B1 B2 C
15 31
45 54
D E F G1 G2 H
42 43
49 52

Performanse

- Razlike od stabla *m* -arnog pretraživanja
  - ✓ raste odozdo nagore
  - ✓ čvorovi na gornjim nivoima obično nisu puni
  - ✓ pretraživanje malo duže (manje ključeva oko korena)
  - ✓ garantovane performanse
  - ✓ najgori slučaj oko 3*h* obraćanja disku

- Maksimalna verovatnoća preloma –
  broj preloma/min broj ključeva

$$(n - 2) / (1 + (n - 1)([m/2] - 1))$$

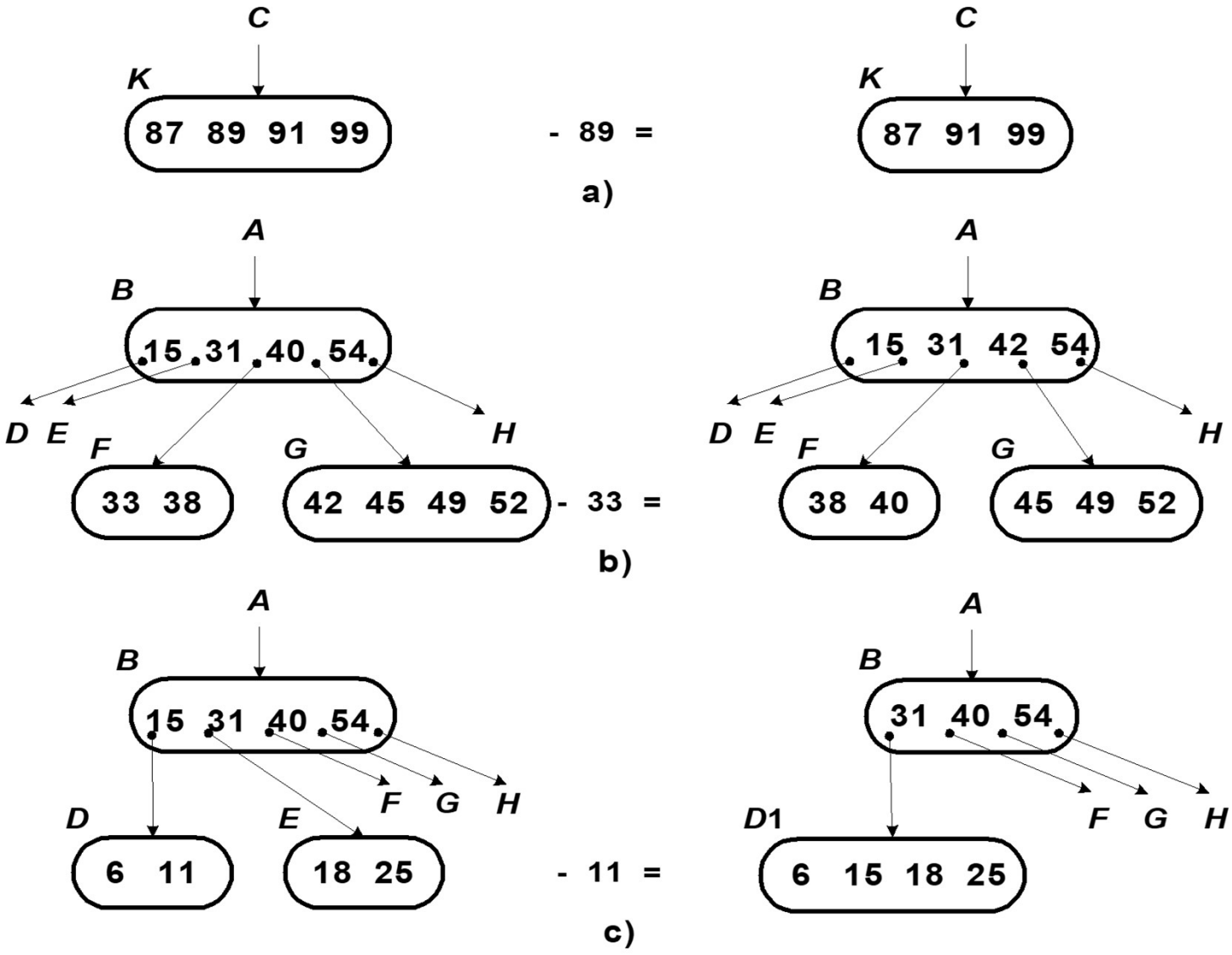
$$1 / ([m/2] - 1)$$

Operacije

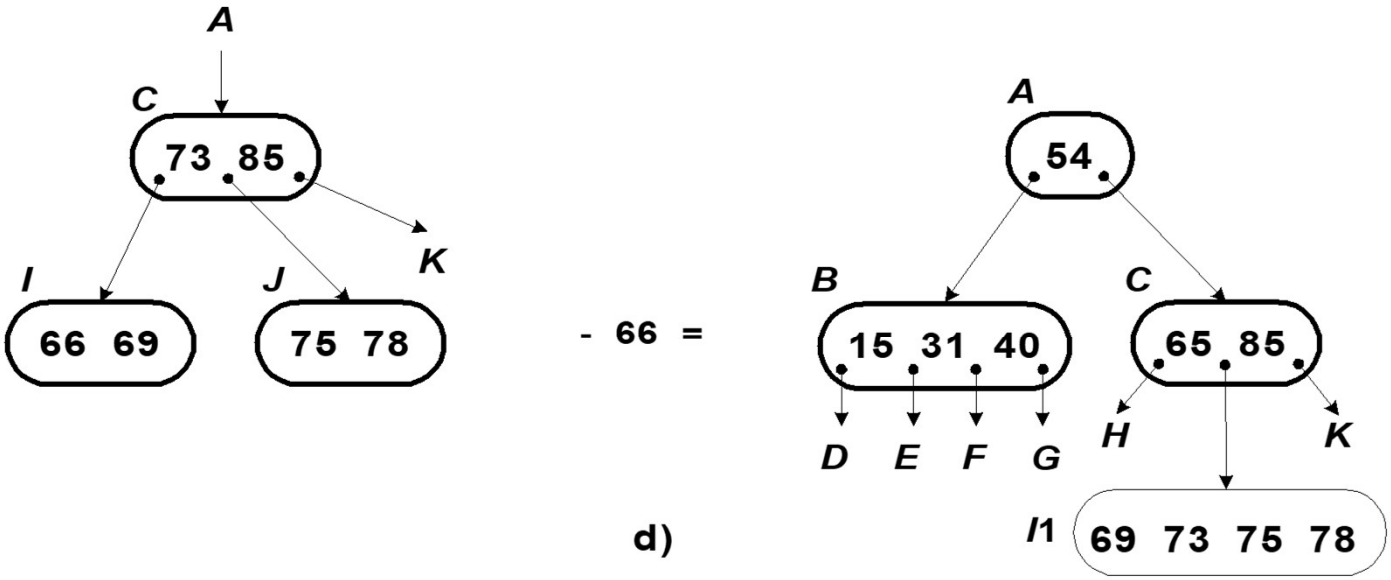
- Brisanje iz lista
  - ✓ ukloni se ključ
  - ✓ ako ostane bar $\lceil m/2 \rceil - 1$ ključeva, samo pomeranje
  - ✓ ako nema dovoljno, proba da preuzme jedan ključ od desnog ili levog brata posredno preko oca
  - ✓ ako ne može da se preuzme – spajanje
  - list + levi (ili desni) brat + razdvojni ključ iz oca
  - $(\lceil m/2 \rceil - 2) + (\lceil m/2 \rceil - 1) + 1 <= m - 1$
  - eventualna propagacija do korena

- Brisanje iz čvora grananja
  - ✓ ključ zameni mesto sa svojim sledbenikom
  - ✓ brisanje iz lista

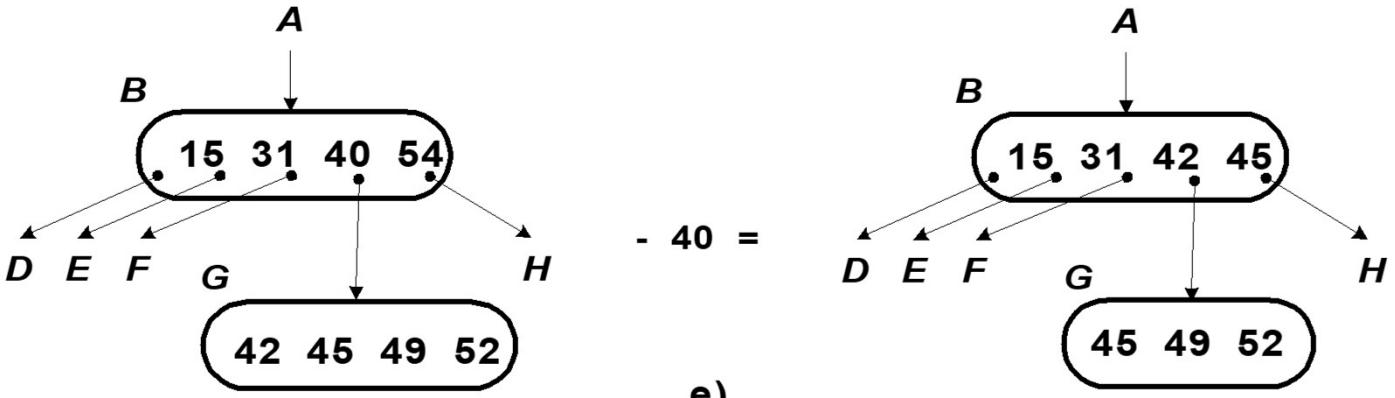
Operacije


C
K
87 89 91 99
- 89 =
C
K
87 91 99
a)
A
B
15 31 40 54
D E
F
G
H
33 38
42 45 49 52
- 33 =
A
B
15 31 42 54
D E
F
G
H
38 40
45 49 52
b)
A
B
15 31 40 54
D
E
F G H
6 11
18 25
- 11 =
A
B
31 40 54
D1
F G H
6 15 18 25
c)

Operacije



d)



e)

Performanse

- Iste kao kod umetanja
- Optimizacije brisanja bez fizičkog uklanjanja
- Iskorišćenje prostora bolje (oko 70%)
- Čvorovi oko korena u baferima

*B**-stabla

- Najskuplji deo umetanja u *B*-stablu - prelom

- Ideja
  - ✓ što više odložiti prelom
  - ✓ redistribucija ključeva među braćom ⇒ *B**-stablo

- Definicija:
  - ✓ stablo *m* -arnog pretraživanja
  - ✓ svaki čvor, osim korena, ima najviše *m* podstabala
  - ✓ svaki čvor, osim korena i listova, ima najmanje $(2m-1)/3$ podstabala
  - ✓ koren ima od 2 do $2\lfloor(2m-2)/3\rfloor+1$ podstabla
  - ✓ svi listovi na istom nivou

Operacije

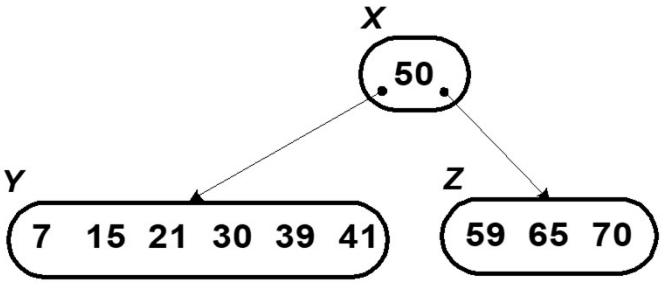
- Minimalna popunjenost oko 2/3
- Pretraživanje kao i u *B*-stablu
- Umetanje
  - ✓ kada je list pun, pokušava se “prelivanje” sa desnim ili levim bratom
  - ✓ list (*m*-1)+brat(*j*)+umetnuti ključ + razdvojni ključ
  - ✓ $\lfloor (m + j + 1)/2 \rfloor + 1$- vi ključ je razdvojni
  - ✓ ako ne može “prelivanje”, onda prelom 2-na-3
  - ✓ $\lfloor (2m - 2)/3 \rfloor$, $\lfloor (2m - 1)/3 \rfloor$, $\lfloor (2m /3 \rfloor$ ključeva, 2 razdvojna ključa
  - ✓ može propagacija do korena

Operacije

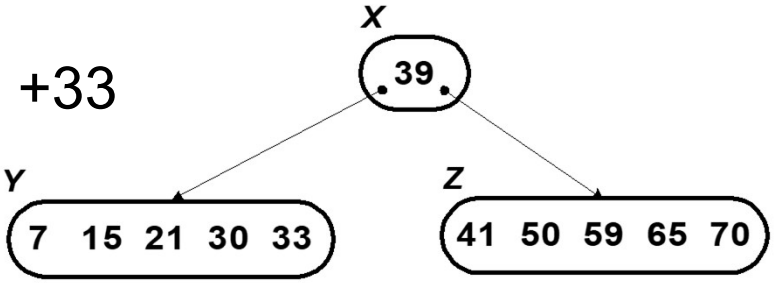
- Pri brisanju spajanje 3-u-2

- U odnosu na *B*-stabla
  - ✓ bolja popunjenost
  - ✓ manja visina, efikasnije operacije

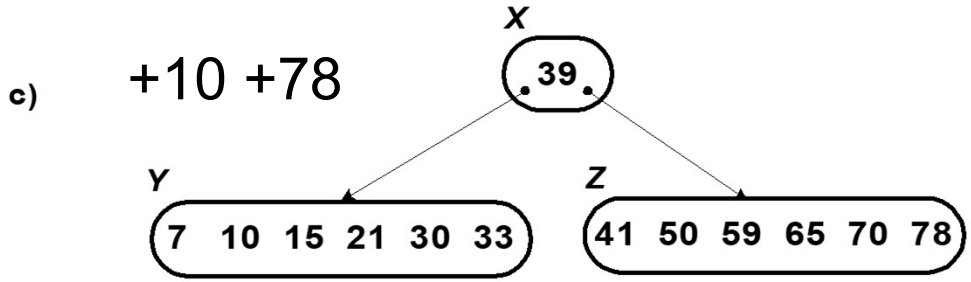
a)



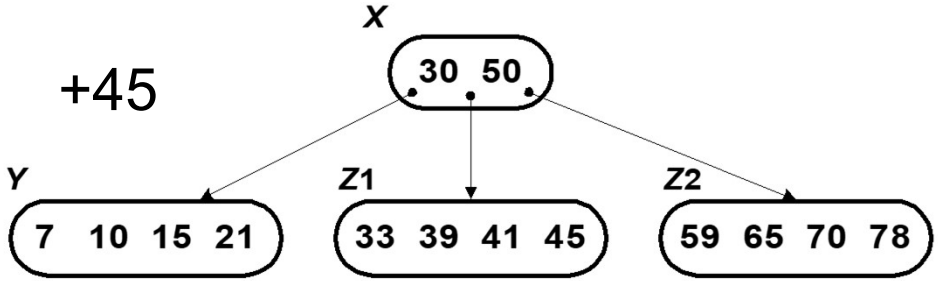
b)



c)



d)



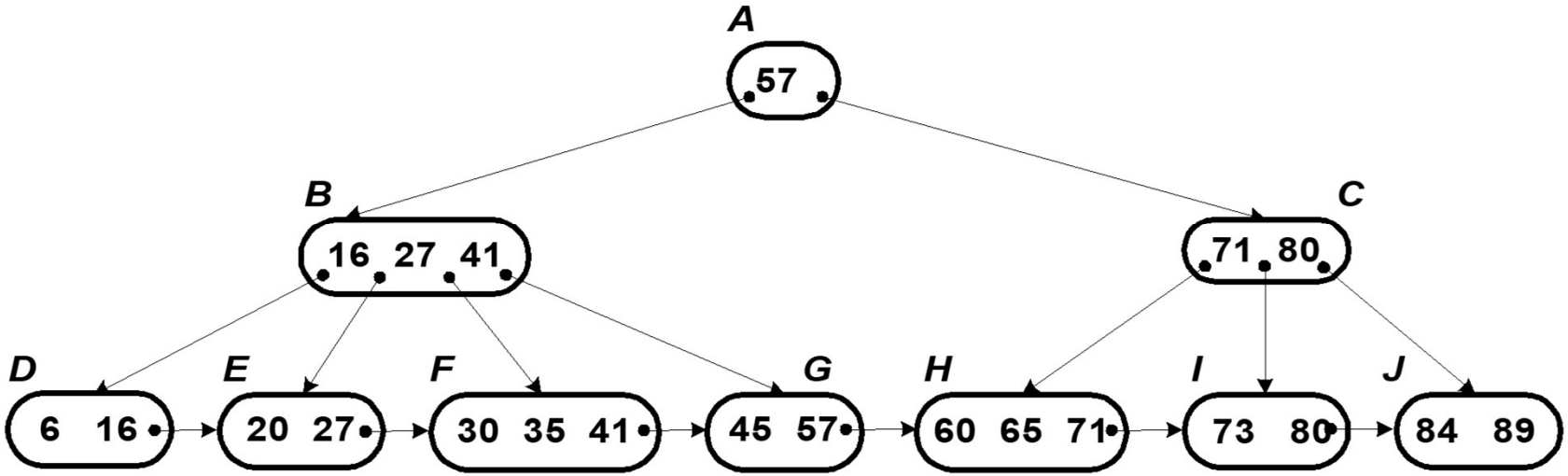
B+-stabla

- Problem – sekvencijalni pristup ⇒ *B*+-stabla

- Čvorovi grananja:
  - ✓ svaki ključ razdvaja dva pokazivača $p_{i-1} K_i p_i$ ali nema fizičkih adresa zapisa
  - ✓ najviše m , a najmanje $\lfloor m/2 \rfloor$ podstabala (koren - 2)
  - ✓ ključevi u čvoru uređeni - $K_i < K_j$ ako je $i < j$
  - ✓ $K_i < K_x \leq K_{i+1}$ u podstablu T_i

- Listovi:
  - ✓ najmanje $\lfloor m/2 \rfloor$ ključeva u uređenom poretku
  - ✓ uz ključ, fizička adresa odgovarajućeg zapisa
  - ✓ svi listovi na istoj dubini
  - ✓ listovi uvezani u ulančanu listu po poretku ključeva

B+-stabla



- Čvorovi grananja – indeksni deo,
  listovi – pristupni deo

- Svaki ključ iz indeksnog dela se replicira u listu
  (kao najveći u levom podstablu)

Operacije

- Sekvencijalni pristup efikasan
  - ✓ ulevo do najmanjeg ključa, pa po ulančanoj listi
  - ✓ nalaženje sledbenika (u istom ili susednom listu)
  - ✓ pristup zapisima sa ključevima u zadatom opsegu

- Ostale operacije slične kao u *B*-stablu

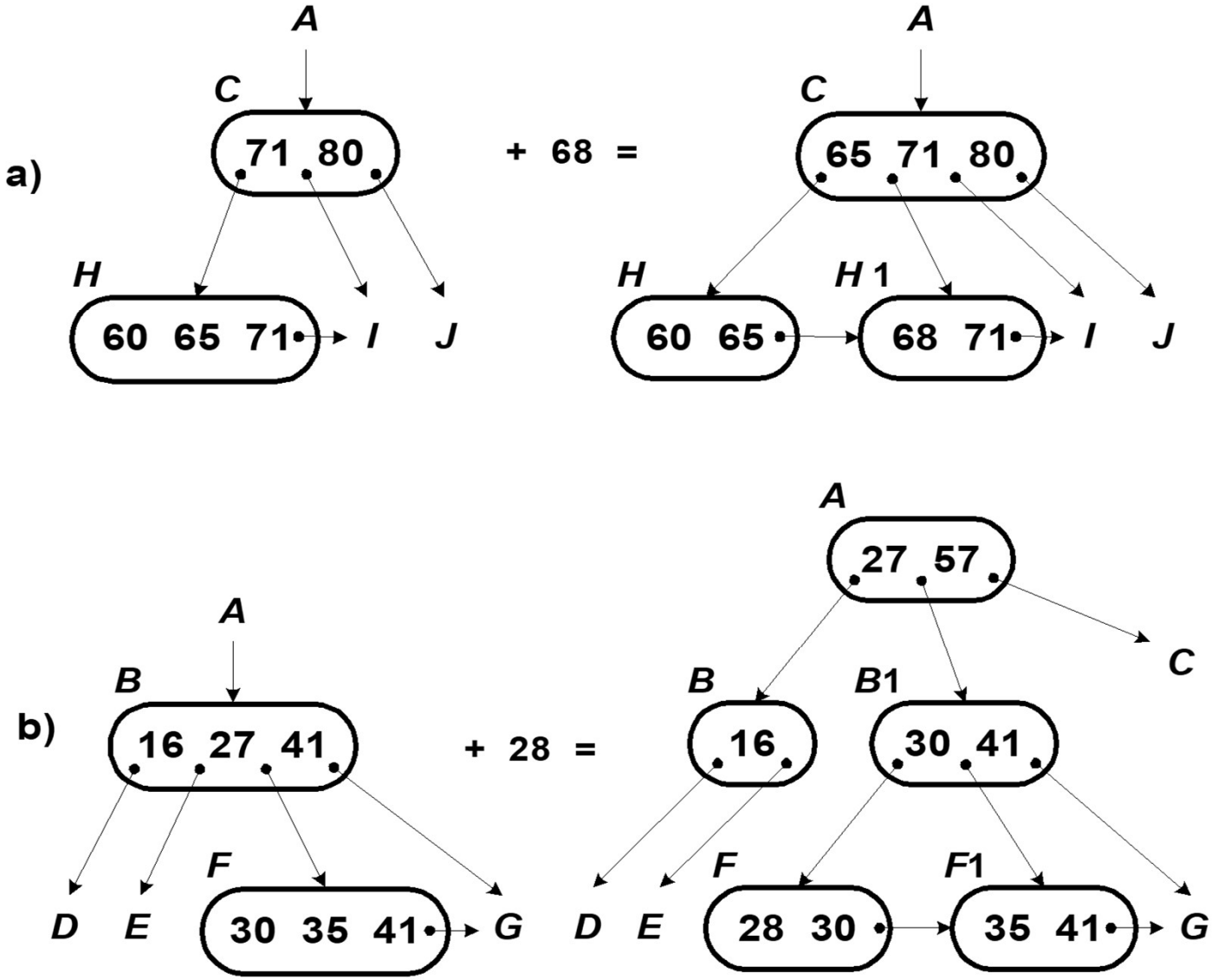
- Pretraživanje
  - ✓ kada se naiđe na isti ključ u indeksnom delu, ide se u levo podstablo
  - ✓ i uspešno i neuspešno se završavaju u listu
  - ✓ fiksne, garantovane performanse

Operacije

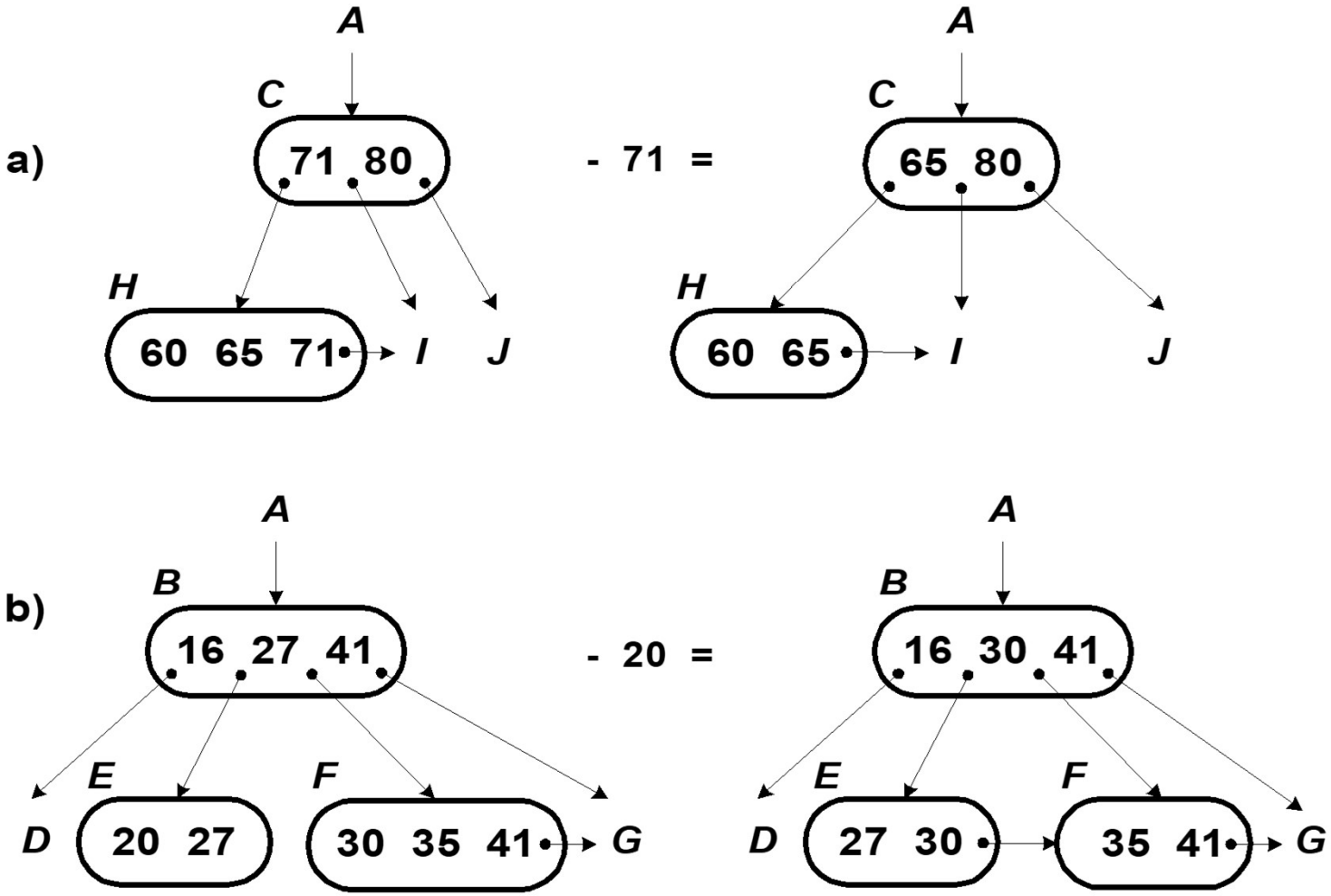
- Umetanje
  - ✓ umetanje u pun list izaziva prelom
  - ✓ manji ključevi i srednji ključ (na poziciji $\lceil m/2 \rceil$) idu u stari list
  - ✓ veći ključevi idu u novi list (ulančava se)
  - ✓ srednji ključ se replicira kod oca
  - ✓ prelom u indeksnom delu - migracija srednjeg ključa

- Brisanje
  - ✓ ključ se ukloni iz lista
  - ✓ ako je najdesniji, prethodnik ga zamenjuje u čvoru grananja
  - ✓ ako padne ispod minimuma, proba preuzimanje
  - ✓ ako ne može preuzimanje, onda spajanje

Operacije


a)
A
C
71 80
H
60 65 71
I
J
+ 68 =
A
C
65 71 80
H
H 1
60 65
68 71
I
J
b)
A
B
16 27 41
D
E
F
30 35 41
G
+ 28 =
A
27 57
B
B1
C
16
30 41
D
E
F
F1
28 30
35 41
G

Operacije


a)
A
C
71 80
H
60 65 71
I
J
- 71 =
A
C
65 80
H
60 65
I
J
b)
A
B
16 27 41
E
F
D
20 27
30 35 41
G
- 20 =
A
B
16 30 41
E
F
D
27 30
35 41
G

Osobine

- Popunjenost čvorova slična kao u *B*-stablu
- Viši stepen grananja za istu veličinu čvora
- Fizički isti format u indeksnom i pristunom delu
- Primena u indeksno-sekvencijalnim datotekama

Stabla digitalnog pretraživanja

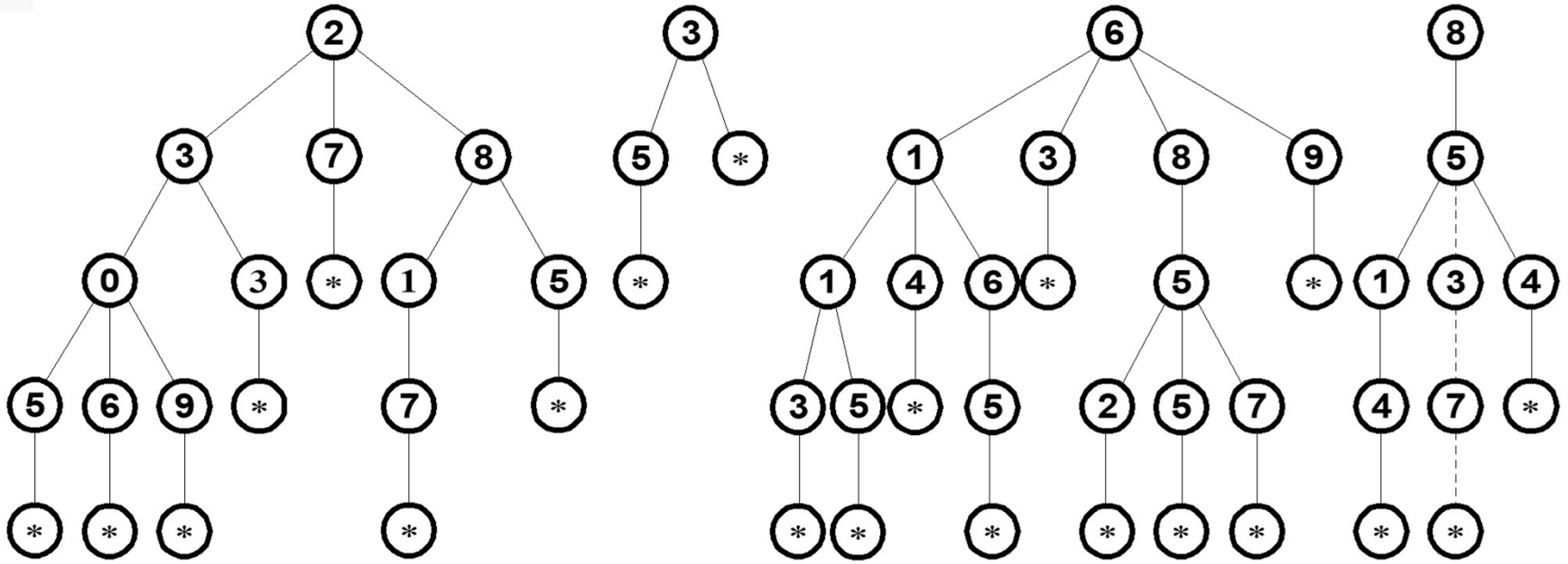
- Stabla opšteg pretraživanja
  zasnovana na poređenju ključeva

- Stabla digitalnog pretraživanja
  zasnovana na interpretaciji ključa kao niza znakova

- Digitalno stablo
  - ✓ šuma opštih stabala sa istim prvim znakom
  - ✓ putanja do lista sadrži redom znakove ključa
  - ✓ kraj ključa – znak EOK u listu (kao i adresa zapisa)

- Ušteda prostora – odgovarajuće binarno stablo
  (braća u uređenoj listi)

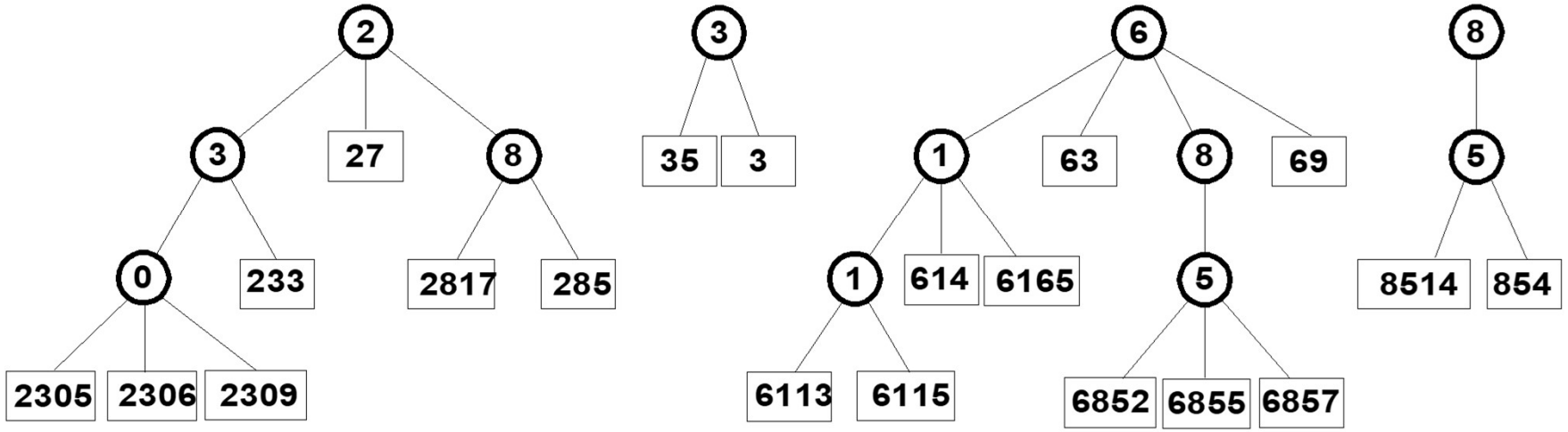
Digitalno stablo



- Pretraživanje
  - ✓ na sledećem nivou, konsultuje sledeći znak ključa
  - ✓ Uspešno pretraživanje se završava u listu (EOK), a neuspešno na bilo kom nivou
- Umetanje i brisanje

Digitalno stablo

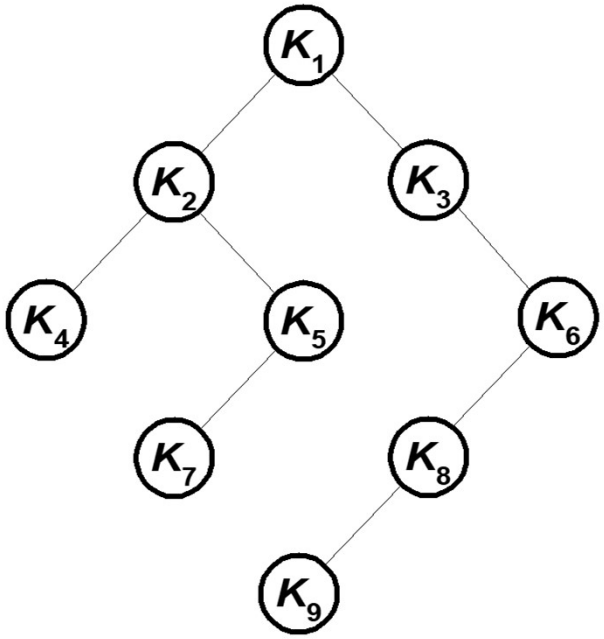
- Modifikovano digitalno stablo
  - ✓ jedinstveni deo putanje do lista – jedan čvor
  - ✓ sadrži čitav ključ i adresu zapisa
  - ✓ efikasnije operacije i ušteda prostora
  - ✓ identifikacija listova



Binarno digitalno stablo

- Svaki čvor sadrži jedan ključ (slično BST)
- Grananje na osnovu narednog bita ključa

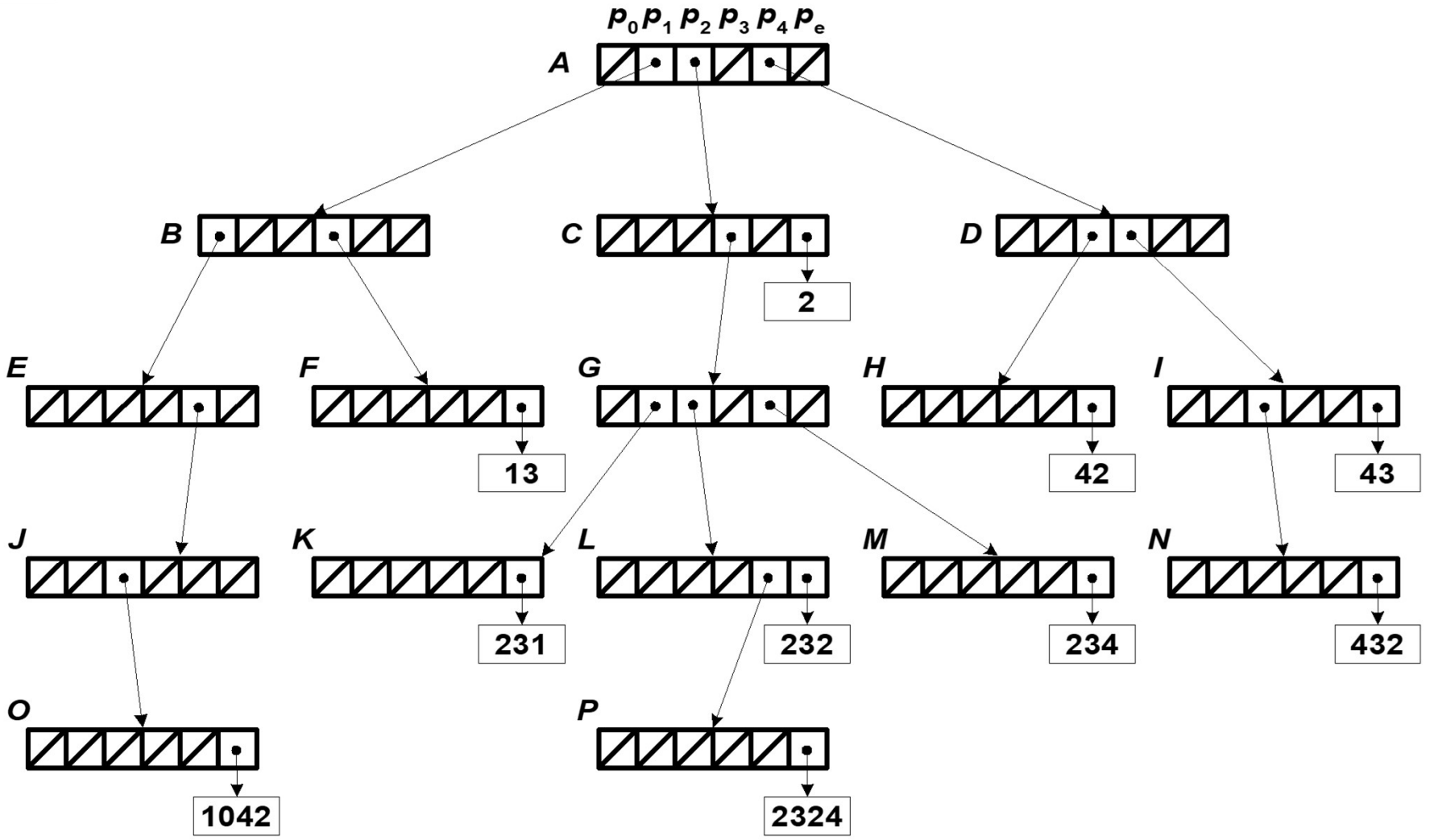
K_1	0110
K_2	0000
K_3	1000
K_4	0001
K_5	0111
K_6	1110
K_7	0100
K_8	1101
K_9	1100



Trie-stablo

- Grananje na osnovu znakova ključa
- Čvor se sastoji samo od pokazivača
  (broj zavisi od radiksa ključa)
- Poziciona korespondencija pokazivača i znakova ključa
- Na svakom nivou konsultuje se jedan znak ključa
- Veličina stabla određena radiksom i dužinom ključa

Trie-stablo


$p_0 p_1 p_2 p_3 p_4 p_e$
A
B
C
D
2
E
F
G
H
I
13
42
43
J
K
L
M
N
231
232
234
432
O
P
1042
2324

Operacije

- Pretraživanje
  - ✓ neuspešno - do prvog praznog pokazivača
  - ✓ uspešno – kad se preko svih znakova ključa dođe do adrese zapisa
- Umetanje
  - ✓ izgraditi lanac pokazivača kroz čvorove
  - ✓ eventualno dodavanje novih čvorova
- Brisanje
  - ✓ adresa nađenog zapisa se ukine
  - ✓ ako nema više aktivnih pokazivača u čvoru, on se ukida

Modifikacije trie-stabla

- Podstablo sa samo jednim ključem se zamenjuje jednim čvorom

- Ograničenje broja nivoa
  - ✓ trie-stablo do određenog nivoa
  - ✓ za duže ključeve male indeksne strukture koje bolje koriste prostor (BST ili liste)
  - ✓ ušteda prostora uz neznatno lošije performanse

- Smanjenje broja pokazivača u ključu uz čuvanje znaka

- Grananje počinje od poslednjeg znaka

Performanse

- Ova stabla pogodna za ključeve neuniformne dužine
- Garantovane performanse
  (ne zavise od broja ključeva, već od dužine ključa)
- Poredak umetanja nema uticaja na strukturu
- Efikasna za neuspešno pretraživanje
- Prostorna efikasnost zavisi od raspodele ključeva
- Problem istih ključeva rešava se ulančavanjem

III.4 Heširanje

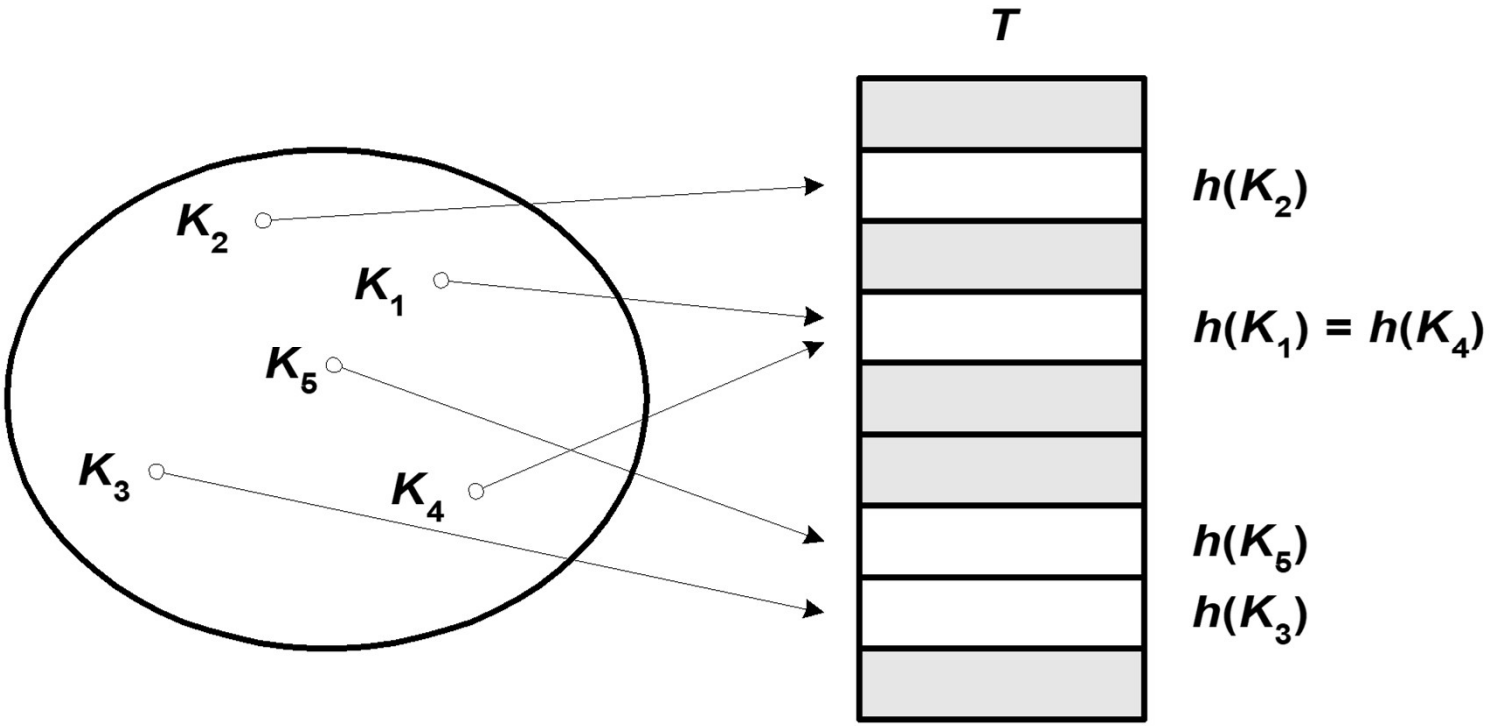
Heširanje

- Obično performanse kod pretraživanja zavise od broja ključeva u skupu
- Idealno pretraživanje – na osnovu ključa direktan pristup bez poređenja sa drugim ključevima
- Moguće sa dovoljno velikom tabelom, ali često nepraktično za prirodne ključeve
- *Heš funkcija* mapira skup ključeva na opseg indeksa u *heš tabeli* (samo ključevi ili zapisi)
- Direktno ili rasuto adresiranje

Heširanje

heš tabela $T[i], 0 \leq i \leq n - 1$

matična adresa $i = h(K)$



Heširanje

- Zadatak heš funkcije – kompresija skupa ključeva na manji opseg indeksa tabele

- Kolizija – $h(K_i) = h(K_j)$, $K_i \neq K_j$
  - ✓ sinonimi $\Rightarrow$ klasa ekvivalencije
  - ✓ degradacija performanse

- Poželjne osobine za efikasnost pristupa
  - ✓ uniformnost – $P(i = h(K)) = 1/n$, $0 \leq i \leq n - 1$
  - ✓ održavanje poretka – $h(K_i) > h(K_j)$ za $K_i > K_j$

- Izbor
  - ✓ efikasne heš funkcije
  - ✓ efikasnog metoda za razrešavanje kolizija

Heš funkcije

- Kriterijumi za izbor heš funkcije
  - ✓ jednostavnost ⇒ brže izračunavanje
  - ✓ uniformnost ⇒ ređe kolizije
- Ključevi – numerički, alfanumerički, alfabetski
- Metod ekstrakcije
- Poželjno da heš funkcija zavisi od svih znakova ključa, kao i njihovih pozicija
- Heš funkcije
  - ✓ nezavisne od raspodele ključeva
  - ✓ zavisne od raspodele ključeva

Nezavisne heš funkcije

- Metod deljenja – $h(K) = K \bmod n$
  - ✓ $n \leq$ veličina heš tabele
  - ✓ jednostavan i često korišćen
  - ✓ ne preporučuje se da n bude:
    - parno,
    - stepen broja 2 ili 10
    - neprosto sa modulom kongruencije
  - ✓ u praksi se preporučuje da bude prost broj ne previše blizu stepena broja 2

- Metod množenja - $h(K) = n(cK \bmod 1)$, $0 < c < 1$
  - ✓ izbor n nije kritičan (može i $n = 2^p$)
  - ✓ $c \approx 0.61803$ (“zlatni presek”)

Nezavisne heš funkcije

Metod sredine kvadrata

$K = 5894$

$5894 * 5894$

23576

53046

47152

29470

34739236

39

Metod sklapanja

$K = 19653014$

19	19
65	56
30	30
14	41
128	146
28	46
a)	b)

Nezavisne heš funkcije

- Metod konverzije osnove
  - ✓ ključ u sistemu sa osnovom p interpretira se u sistemu sa osnovom q ($q > p$)
  - ✓ bira se q uzajamno prosto sa p
  - ✓ npr. 6154 ($p = 10$) $\Rightarrow$ 13420 ($q = 13$)

- Metod algebarskog kodovanja
  - ✓ ključ $(k_{r-1} .. k_0) \Rightarrow K(x) = k_{r-1}x^{r-1} + ... + k_0$
  - ✓ za $n = 2^m$, odabere se $P(x) = x^m + p_{m-1}x^{m-1} + ... + p_0$
  - ✓ $K(x) \bmod P(x) = h_{m-1}x^{m-1} + ... + h_0$.
  - ✓ rezultat $(h_{m-1} .. h_0)$
  - ✓ hardverska implementacija

Nezavisne heš funkcije

- Savršena heš funkcija
  - ✓ nema kolizija
  - ✓ nije lako pronaći, pogotovo za dinamičan skup
  - ✓ lakše za manje popunjene tabele
  - ✓ npr. $H(K) = (K + s)/d$ (Sprugnoli)
  - ✓ minimalna savršena – n ključeva u n ulaza
  - ✓ npr. $H(K) = c/p(K)$ (Chang)

- Univerzalna klasa heš funkcija
  - ✓ slučajan izbor heš funkcije iz konačnog skupa nezavisan od skupa ključeva
  - ✓ dobre prosečne performanse i zaštita od najgoreg slučaja

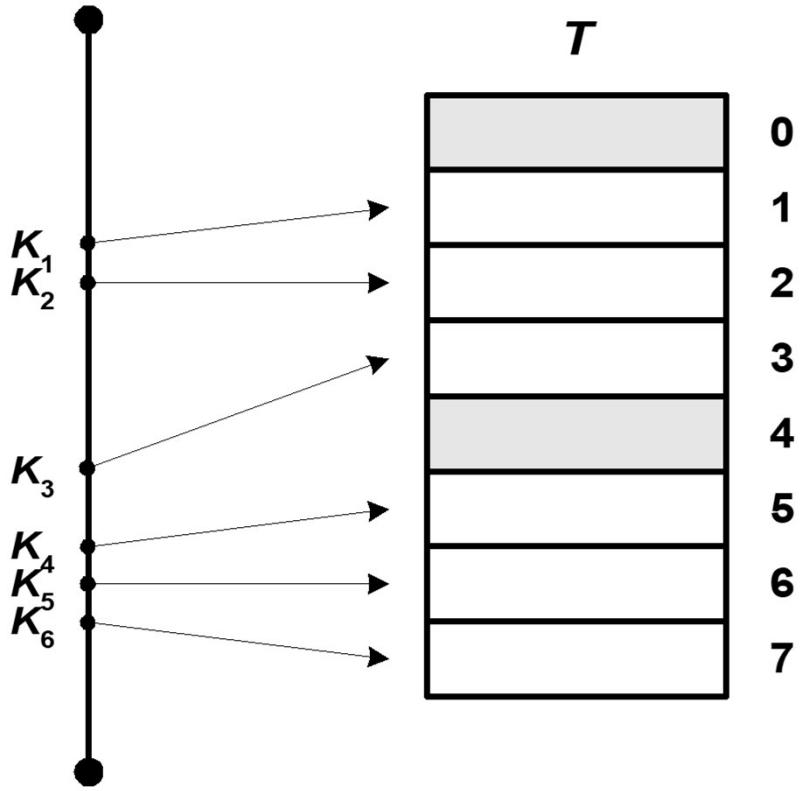
Zavisne heš funkcije

- Moguće za unapred poznatu raspodelu ključeva
- Metod analize cifara
- Diskretna kumulativna funkcija raspodele
  - ✓ $F_z(x) = P(Z \leq x)$
  - ✓ ključevi $K_1 < \ldots < K_m$
  - ✓ diskretna uniformna raspodela
    $P(F_z(Z) \leq i/m) = i/m$, za $0 \leq i \leq m$
  - ✓ $F_z(K_1) = 1/m$, $F_z(K_2) = 2/m$, ... , $F_z(K_m) = 1$
  - ✓ cilj $P(h(K) = i) = 1/n$, za $0 \leq i \leq n - 1$
  - ✓ $H(K) = [n\, F_z(K)] - 1$

Zavisne heš funkcije

Primer:

ključevi 6, 7, 12, 14, 15 i 16 u tabeli sa 8 ulaza



Zavisne heš funkcije

- Aproksimacija – segmentna linearna funkcija
  - ✓ m celobrojnih ključeva u intervalu (a, d)
  - ✓ $l = (d - a)/j$ podintervala
  - ✓ $i = 1 + \lfloor (K - a)/l \rfloor$ redni broj podintervala za ključ K
  - ✓ N_i - broj ključeva u podintervalu i
  - ✓ G_i - ukupan broj ključeva u podintervalima 1,..., i
  - ✓ $P_i(K) = (G_i + ((K - a)/l - i)N_i)/m$
  - ✓ $h_i(K) = [n\, P_i(K)] - 1,\ 1 \le i \le j$

i	N_i	G_i
1	2	2
2	0	2
3	5	7
4	9	16
5	4	20

Primer:

$m = 20$, (0,100), $j = 5$

$K = 51$

$i = 1 + \lfloor (51 - 0)/20 \rfloor = 3$

$h_3(51) = [25\ (7 + ((51 - 0)/20 - 3) * 5)/20] - 1 = 5$

Razrešavanje kolizija

- Kolizija – dva ili više ključeva ima istu matičnu adresu
- Veća tabela ⇒ ređe kolizije,
  ali dodatni prostor i manja popunjenost
- Faktor popunjenosti
- Varijante razrešavanja kolizija
  - ✓ otvoreno adresiranje – u okviru tabele
  - ✓ ulančavanje – lista sinonima

Otvoreno adresiranje

- Ispitni niz
  - ✓ niz adresa za ključ pri pretraživanju ili umetanju
  - ✓ generisanje niza – ponovno heširanje
  - ✓ poželjno – permutacija od (0 .. *n* -1)

- Problem brisanja
  - ✓ “poluslobodne” lokacije
  - ✓ selektivno pomeranje

- Varijante otvorenog adresiranja
  - ✓ linearno pretraživanje
  - ✓ slučajno pretraživanje
  - ✓ kvadratno pretraživanje
  - ✓ dvostruko heširanje

Otvoreno adresiranje

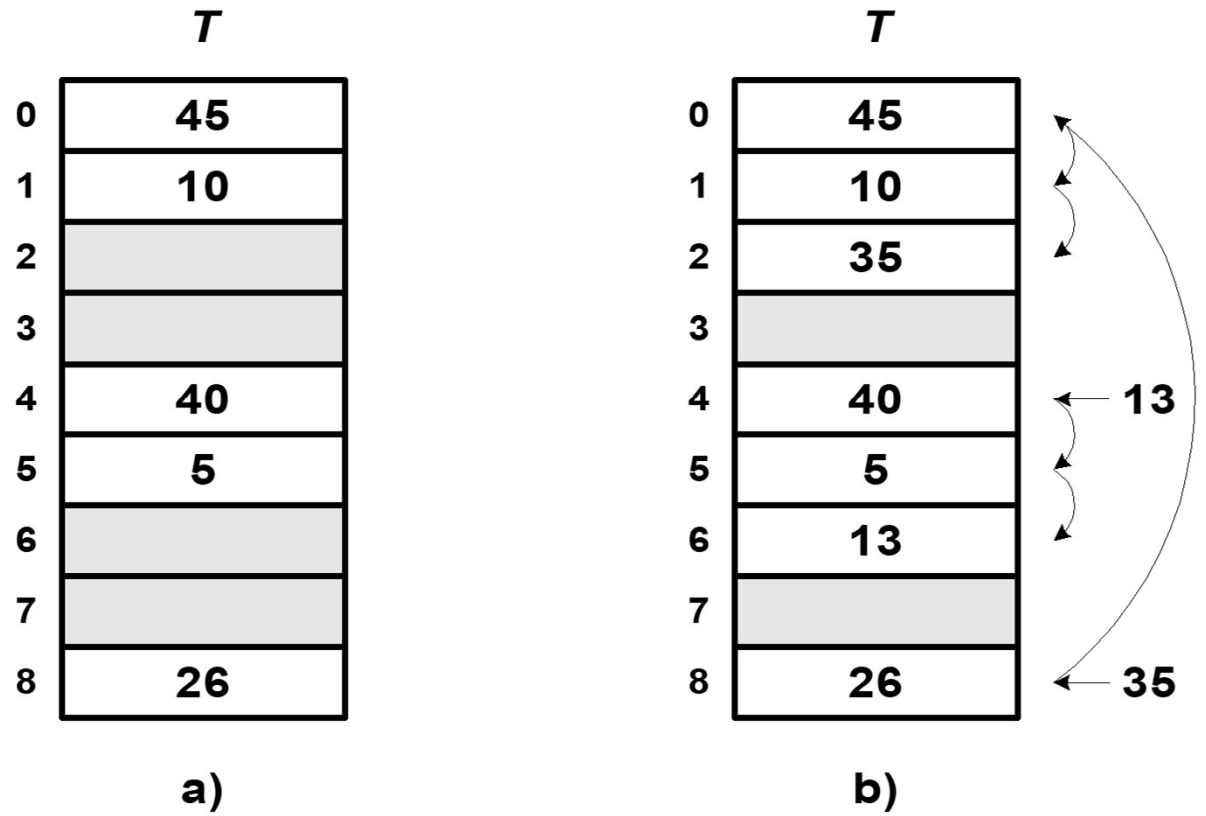
```
HASH-INSERT(T, K)
i = 0
repeat
      j = h_i(K)
      if (T[j] = empty) then
            T[j] = K
            return j
      else
            i = i + 1
      end_if
until i = n
ERROR(Tabela puna)
```

```
HASH-SEARCH(T, K)
i = 0
repeat
      j = h_i(K)
      if (T[j] = K) then
            return j
      else
            i = i + 1
      end_if
until (T[j] = empty) or (i = n)
return empty
```

Linearno pretraživanje

$$h_i(K) = (h_0(K) + i) \bmod n, \quad i = 1, 2, \ldots, n - 1$$

$$h_i(K) = (h_{i-1}(K) + 1) \bmod n$$

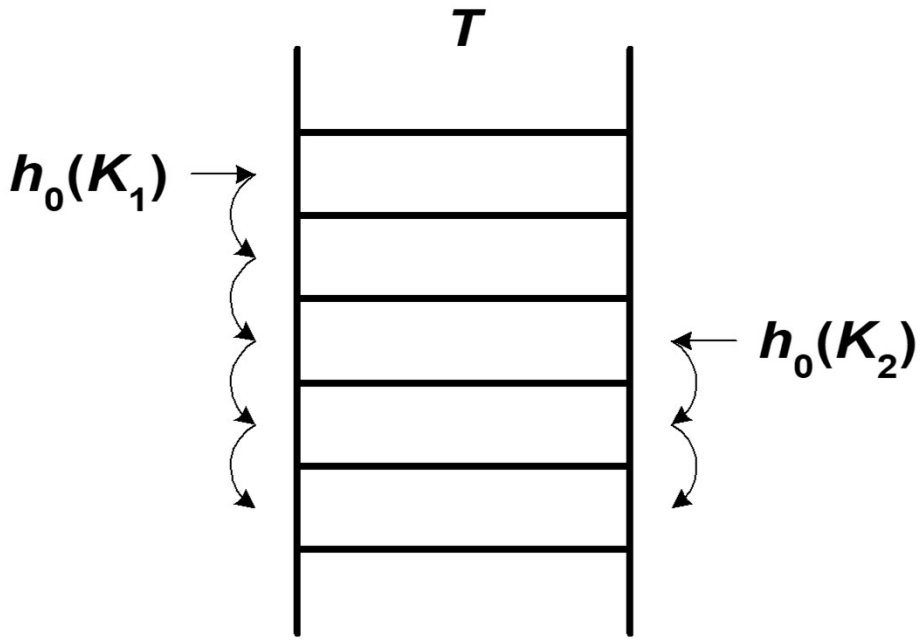


Linearno pretraživanje

- Selektivno pomeranje pri brisanju moguće
  - ✓ brisanje na adresi *i*
  - ✓ prva slobodna adresa *j*
  - ✓ provera počinje od adrese *i* + 1
  - ✓ za *K* sa adrese $l \in (i+1..\ j)$ $r = h(K)$
  - ✓ ako je $r < i$ ili $r > j$,
    K se može premestiti na adresu *i*
  - ✓ ide se dalje prema od *l* ka adresi *j*

Linearno pretraživanje

- Primarno grupisanje
  - ✓ grupe zauzetih lokacija za veću popunjenost
  - ✓ nejednaka verovatnoća popunjavanja slobodnih lokacija
  - ✓ $h_{i+r}(K_1) = h_r(K_2)$ za $r = 0, 1, 2, \ldots$



Linearno pretraživanje

$h_i(K) = (h_0(K) + ic) \bmod n, \quad i = 1, 2, \ldots, n - 1$

$h_i(K) = (h_{i-1}(K) + c) \bmod n$

- Za c uzajamno prosto sa n, perioda ispitnog niza n
- $c > 1$ razbija kontinualne grupe,
  ali primarno grupisanje ostaje
- Naredna adresa ne treba da zavisi samo od prethodne
- Npr. $h_i(K) = (h_{i-1}(K) + i) \bmod n, \quad i = 1, 2, \ldots$
- Linearno pretraživanje sa razdvojenom sekvencom

Slučajno pretraživanje

$$h_i(K) = (h_0(K) + p_i) \bmod n, \quad i = 1, 2, \ldots, n - 1$$

	p_1 =	101	= 5
* 2		1010	
- 8		010	
⊕ 5	p_2 =	111	= 7
* 2		1110	
- 8		110	
⊕ 5	p_3 =	011	= 3
* 2	p_4 =	110	= 6
* 2		1100	
- 8		100	
⊕ 5	p_5 =	001	= 1
* 2	p_6 =	010	= 2
* 2	p_7 =	100	= 4

$n = 8$

$c = 5$

Kvadratno pretraživanje

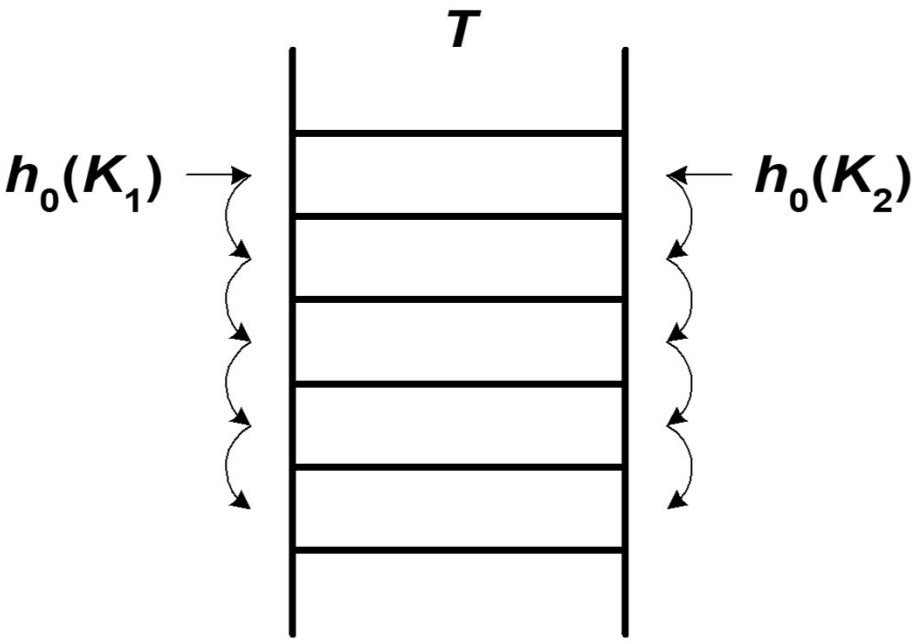
$$h_i(K) = (h_0(K) + i^2) \bmod n, \quad i = 1, 2, \ldots, n - 1$$

$$i^2 = d_i = d_{i-1} + 2i - 1$$

- Problem – potpunost ispitnog niza (perioda n)
- Ako je n prost, ispitni niz sadrži bar $n/2$ adresa
- Ako je n prost oblika $4j + 3$, ispitni niz potpun

Sekundarno grupisanje

- Sekundarno grupisanje
  - ✓ za sinonime
  - ✓ $h_i(K_1) = h_i(K_2)$ za $i = 0, 1, 2, \ldots$



Dvostruko heširanje

$h_i(K) = (h_0(K) + i\, g(K)) \bmod n, \quad i = 1, 2, \ldots, n - 1$

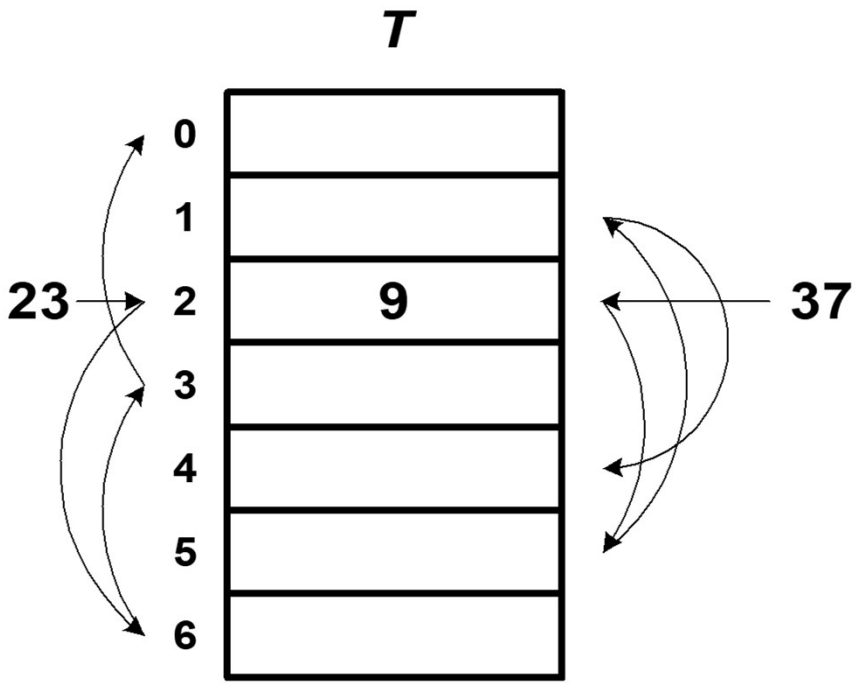
$h_i(K) = (h_{i-1}(K) + g(K)) \bmod n$

- Za nezavisne $h(K)$ i $g(K)$
  verovatnoća sekundarnog grupisanja vrlo mala
- Obično n prost broj, a $g(K) \in (1, n - 1)$
- Knuth predlaže
  - ✓ $h_0(K) = K \bmod n$
  - ✓ $g(K) = 1 + K \bmod (n-2)$
  - ✓ n i $n-2$ prosti brojevi

Dvostruko heširanje

$h_0(K) = K \bmod 7$

$g(K) = 1 + K \bmod 5$



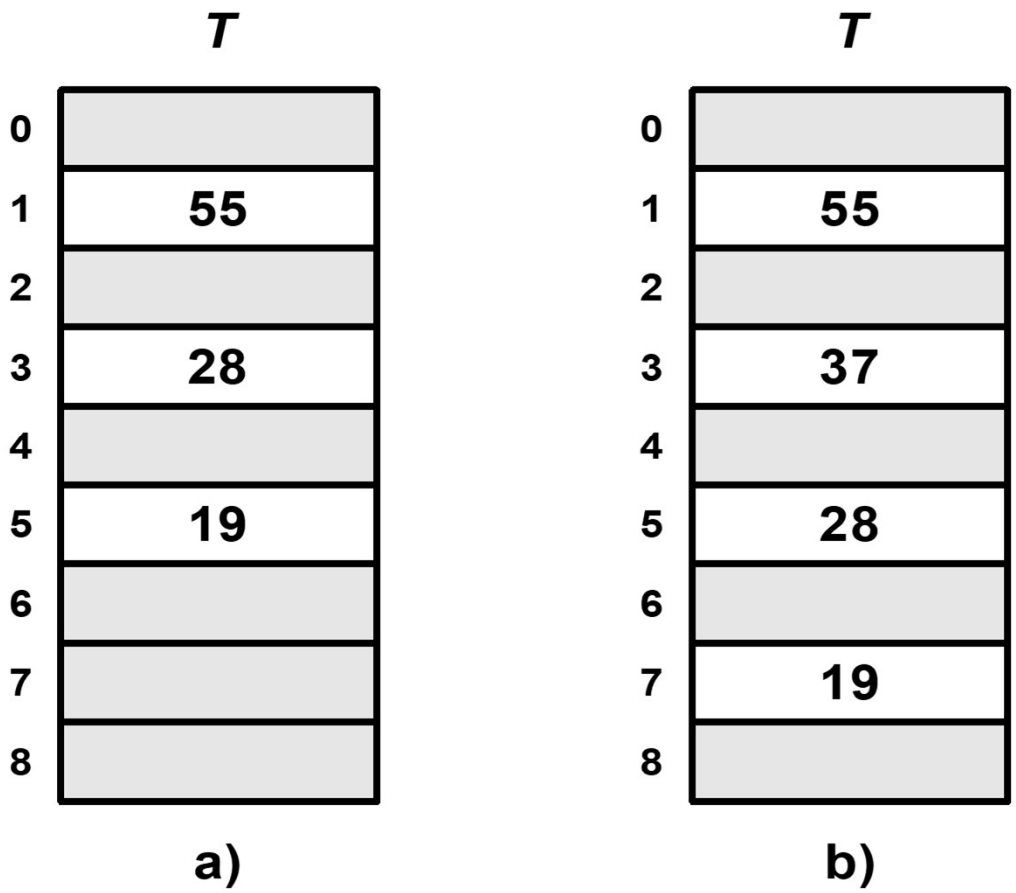
Poboljšani metodi

- Metod uređene tabele
  - ✓ neuspešno pretraživanje efikasnije
  - ✓ sinonimi se drže po opadajućem poretku na adresama u ispitnom nizu
  - ✓ pri umetanju se manji ključevi pomeraju
  - ✓ ista efikasnost uspešnog i neuspešnog pretraživanja

- Brent-ov metod
  - ✓ uspešno pretraživanje efikasnije
  - ✓ preuređenje tabele pri dvostrukom heširanju
  - ✓ premeštanje već umetnutog ključa
  - ✓ cilj – smanjenje prosečnog broja poređenja

Metod uređene tabele

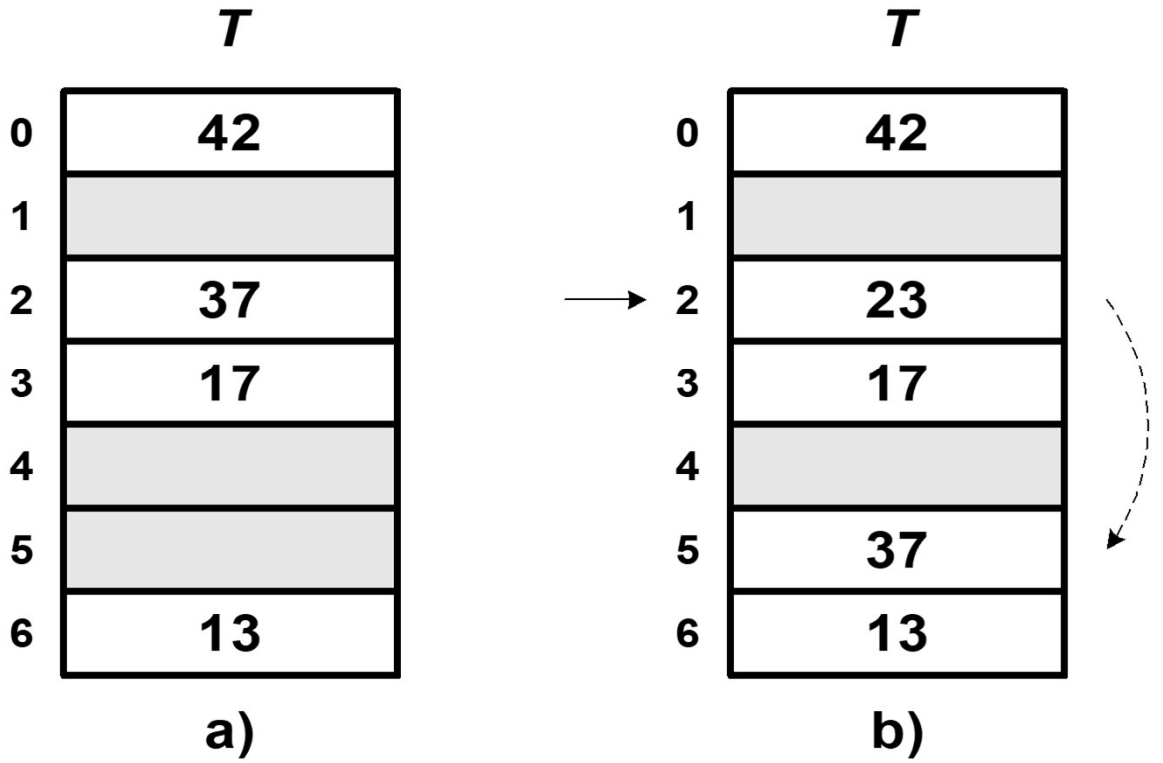
$$h_i(K) = (K + 2i) \bmod n$$



Brent-ov metod

$h_0(K) = K \bmod 7$

$g(K) = 1 + K \bmod 5$

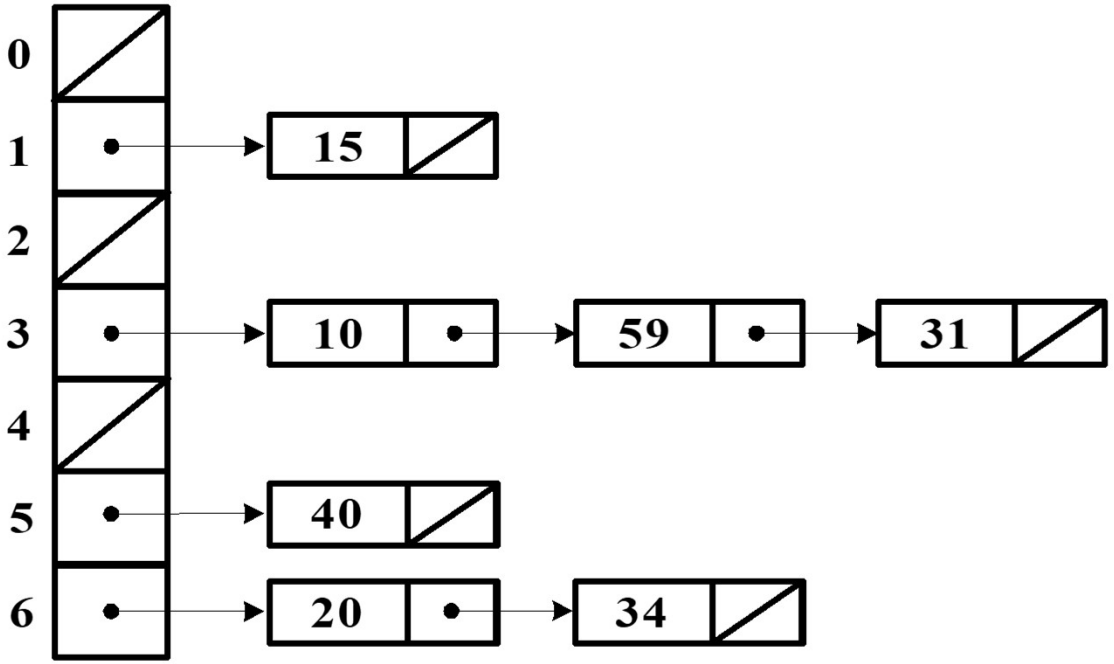


Ulančavanje

- Problemi otvorenog adresiranja
  - ✓ brisanje ključeva
  - ✓ ispitni niz ne mora da sadrži samo sinonime
- Ulančavanje sinonima u liste van oblasti tabele
- Lista – klasa ekvivalencije
- Pretraživanje, umetanje i brisanje efikasnije
- Nema ponovnog heširanja
- Ne ograničava broj ključeva veličinom tabele

Odvojeno ulančavanje

$h(K) = K \bmod 7$

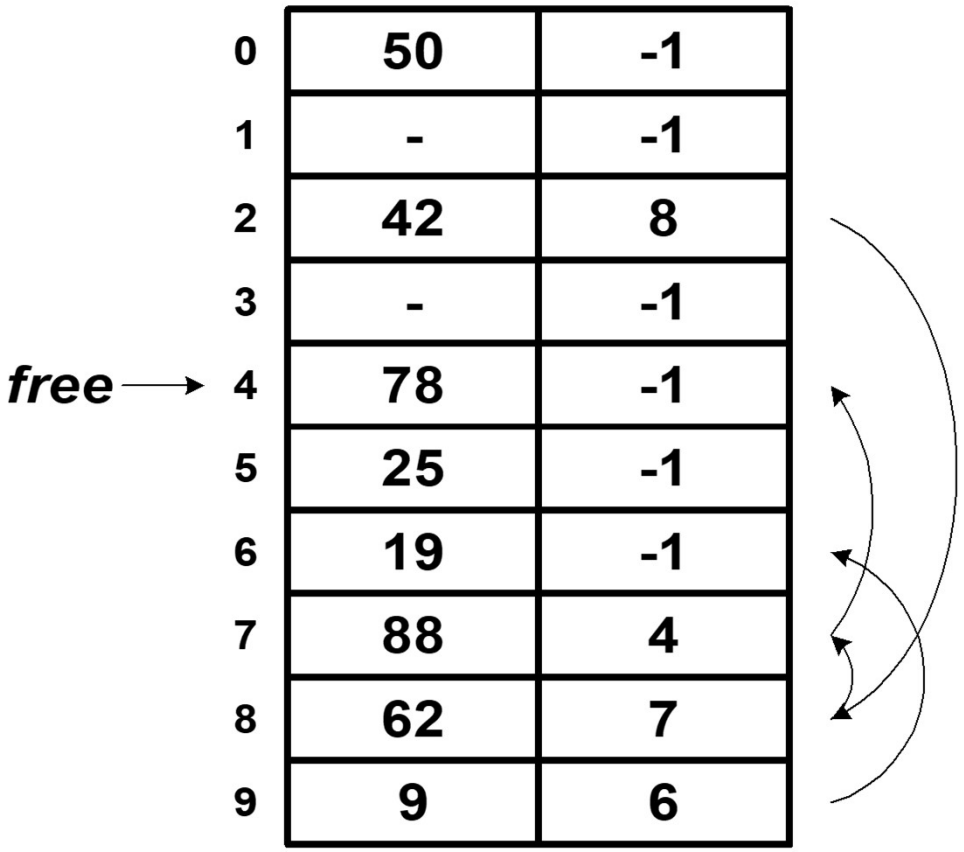


Objedinjeno ulančavanje

```
SEARCH-INSERT-CH(T, K)
i = h(K)                                          T[j].key = K
while (T[i].key) ≠ K) and (T[i].next ≠ -1) do     return j

      i = T[i].next
end_while
if (T[i].key = K) then
      return i
end_if
if (T[i].key = empty) then
      j = i
else
      while (T[free].key ≠ empty) do
            free = free – 1
            if (free < 0) then
                  ERROR(Tabela puna)
            end_if
      end_while
      j = free
      T[i].next = free
end_if
```

Objedinjeno ulančavanje



Performanse

	S			U		
α	**LP**	**DH**	**SC**	**LP**	**DH**	**SC**
0.1	1.056	1.054	1.050	1.118	1.111	1.005
0.2	1.125	1.116	1.100	1.281	1.250	1.019
0.3	1.214	1.189	1.150	1.520	1.429	1.041
0.4	1.333	1.277	1.200	1.889	1.667	1.070
0.5	1.500	1.386	1.250	2.500	2.000	1.107
0.6	1.750	1.527	1.300	3.625	2.500	1.149
0.7	2.167	1.720	1.350	6.060	3.333	1.197
0.8	3.000	2.012	1.400	13.000	5.000	1.249
0.9	5.500	2.558	1.450	50.500	10.000	1.307
0.95	10.500	3.153	1.475	200.50	20.000	1.337

Performanse

- Objedinjeno heširanje nije mnogo lošije od odvojenog heširanja
- Dodatni prostor kod objedinjenog ulančavanja
- Metod deljenja najpraktičniji i najčešći
- Heširanje efikasna tehnika, ali ne ograničava najgori slučaj
- Problem fiksne veličine tabele kod otvorenog adresiranja
- Heširanje po pravilu ne omogućava pristup u poretku

Spoljašnje heširanje

- U datotekama sa direktnim pristupom
- Slične heš funkcije i metodi razrešavanja kolizija
- Mnogo bitnije vreme nego prostor
- Cilj – smanjiti broj kolizija
- Ulazi heš tabele kao baketi
- Baket može sadržati samo ključeve
  ili cele zapise (nema posebne heš tabele)

Standardne tehnike

- Popunjenost $\alpha = m / nb$
- Izbor heš funkcije kritičan – uniformnost!
- Adresiranje baketa – *b* ključeva bez kolizije
- Ključevi unutar baketa mogu biti i uređeni
  pa može i binarno pretraživanje
- Neuspešno pretraživanje kada ključ nije u baketu,
  a on nije pun
- Isti metodi razrešavanja kolizija

Performanse

b	α	LP	DH	SC
1	0.5	1.5	1.386	1.250
	0.8	3.0	2.012	1.4
	0.95	10.5	3.153	1.5
5	0.5	1.031	1.028	1.036
	0.8	1.289	1.184	1.186
	0.95	2.7	1.529	1.3
10	0.5	1.005	1.005	1.007
	0.8	1.110	1.079	1.115
	0.95	1.8	1.292	1.3
50	0.5	1.000	1.000	1.000
	0.8	1.005	1.005	1.005
	0.95	1.1	1.067	1.2

Standardne tehnike

- Za veće *b*, broj kolizija opada
- Odnos performansi metoda za razrešavanje kolizija
- Linearno pretraživanje kompetitivno
  za veće bakete i manje popunjenosti tabele
- Važne i relativne pozicije baketa
- Kod odvojenog ulančavanja oblast prepunjenja
  (obično u fizičkoj blizini)
  - ✓ poseban baket za svaku listu
  - ✓ zajednički baketi za više lista

Standardne tehnike

- Objedinjeno ulančavanje
  - ✓ ulančava kroz bakete u primarnoj oblasti
  - ✓ lista baketa koji nisu puni
  - ✓ u matičnom baketu pokazivač na prvi ključ sa te matične adrese
- Komplikacije sa ključevima neuniformne dužine
- Nemogućnost sekvencijalnog pristupa po poretku

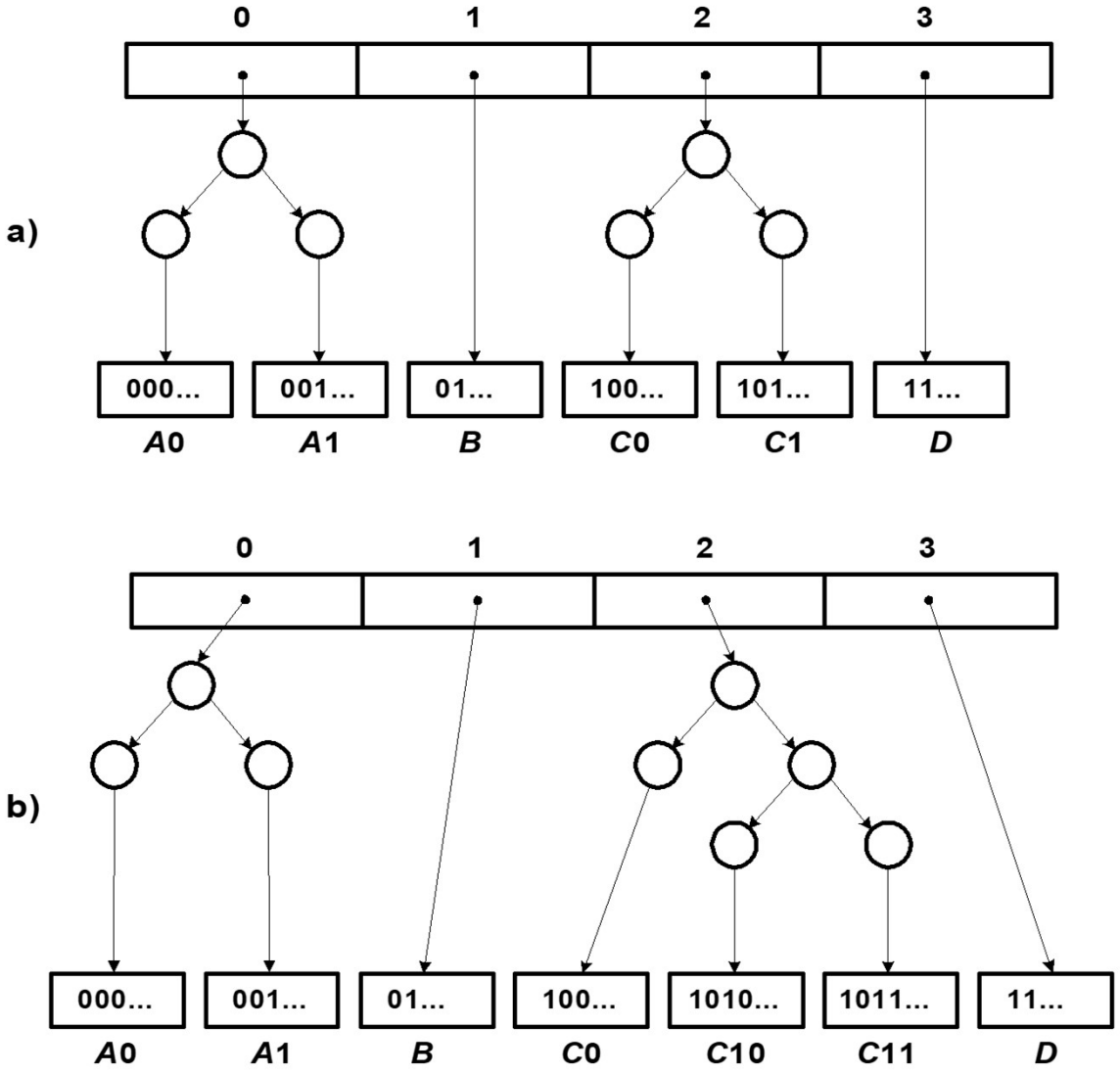
Fleksibilne tehnike

- Datoteke relativno dugovečne strukture
- Veličina može nepredvidljivo da raste
- Standardne tehnike mogu da budu neefikasne
- Fleksibilne tehnike
  - ✓ dinamičko heširanje
  - ✓ proširljivo heširanje

Dinamičko heširanje

- Primarna oblast *m* baketa – *m* ulaza heš tabele
- Heš tabela se adresira sa *b* = log *m* bita
- Umetanje u pun baket izaziva prelom
- Reheširanje i preraspodela po *b* + 1-vom bitu
- Binarna stabla sa korenima u heš tabeli
- Pretraživanje i brisanje

Dinamičko heširanje



Proširljivo heširanje

- Prelom i preraspodela kao kod dinamičkog heširanja
- Dubina baketa i dubina heš tabela (*d*)
- Pri povećanju dubine baketa – razdvajanje ulaza
- Pri povećanju dubine heš tabele – dupliranje
- Heš tabela se adresira sa *d* bita
- Kad se pri brisanju smanji dubina tabele, tabela se prepolovi

Proširljivo heširanje