Algoritam DSW (**Day–Stout–Warren**) za jednokratno uravnotežavanje binarnog stabla

Pitanje 1:Zbog čega nam je značajna osobina balansiranosti za binarna stabla pretrage, BST?
Primer uz pitanje 1:
Neka imamo perfektno balansirano binarno stablo pretrage sa n= 10000 čvorova

Visina stabla je $h=\lceil \log(10{,}001) \rceil = \lceil 13.289 \rceil = 14$.

Dakle, u stablu sa n= 10000 elemenata najviše 14 čvorova će biti testirano dok ne dobijemo odgovor da li se neka vrednost nalazi među elementima.

Uporedite ovaj primer sa jednostruko povezanom listom sa n čvorova. Dakle, vredi tokom rada modifikovati BST stabla tako da budu balansirana ili ih nakon rada balansirati.

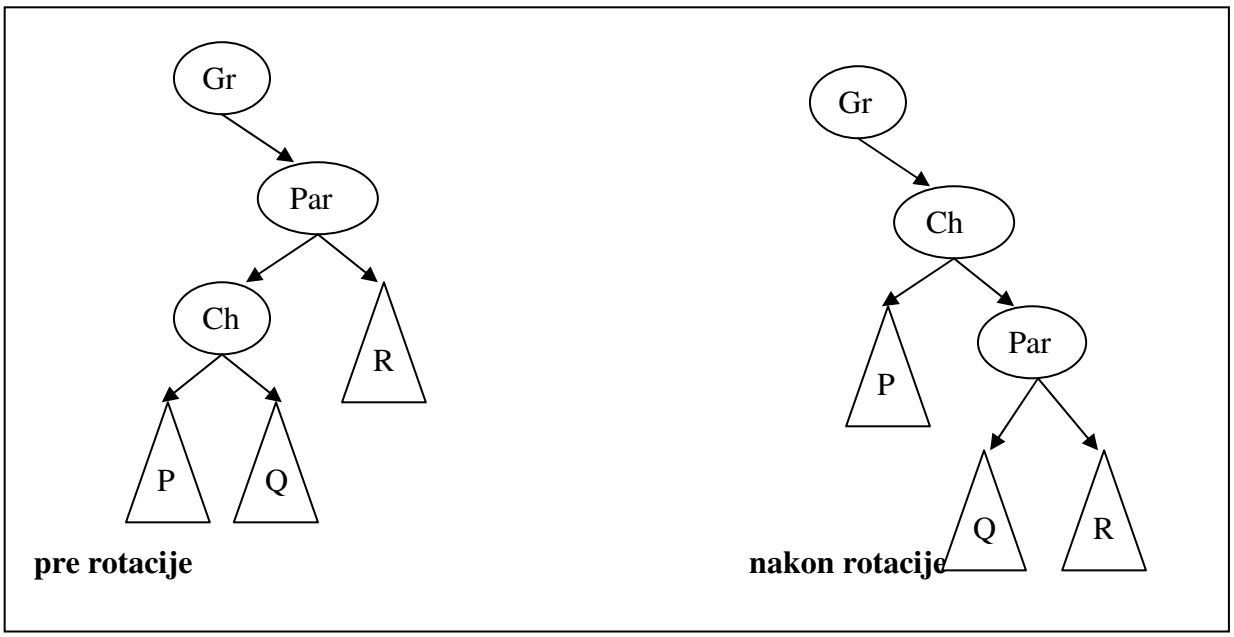
Day-Stout-Warren algoritam služi za balansiranje binarnih stabala pretrage.
1. Ovaj algoritam ne radi inkrementalno nakon svakog unošenja elementa u stablo, već se može pozivati periodično.
2. Ima linearnu vremensku složenost O(n) i konstantnu memorijsku složenost (radi u mestu). Posebno je pogodan za balansiranje stabla kada se zna da se u njega više neće umetati elementi.
3. Algoritam se sastoji iz dve faze. **U prvoj fazi se stablo ispravlja u listu, tako da svaki čvor ima samo desnog sina, a zatim se ova lista prepravlja u balansirano binarno stablo.** Jedan od načina da se ovo provede jest da dokle god koren stabla ima levih sinova, vrši se rotacija tako da levi sin korena postane koren, a koren postane desni sin novog korena. Kada koren više nema levih sinova, prelazi se na njegov desni koren, i ponavlja se procedura dok se celo stablo ne ispravi u listu.
U drugoj fazi sledi transformacija liste u savršeno balansirano stablo. I ovde se koriste rotacije. Svaki drugi čvor se rotira oko svog roditelja. Proces se ponavlja niz desnu granu sve dok se ne dobije balansirano stablo.

4. DSW algoritam je razvip Colin Day 1976, a kasnije su ga poboljšali Quentin Stout i Bette Warren.

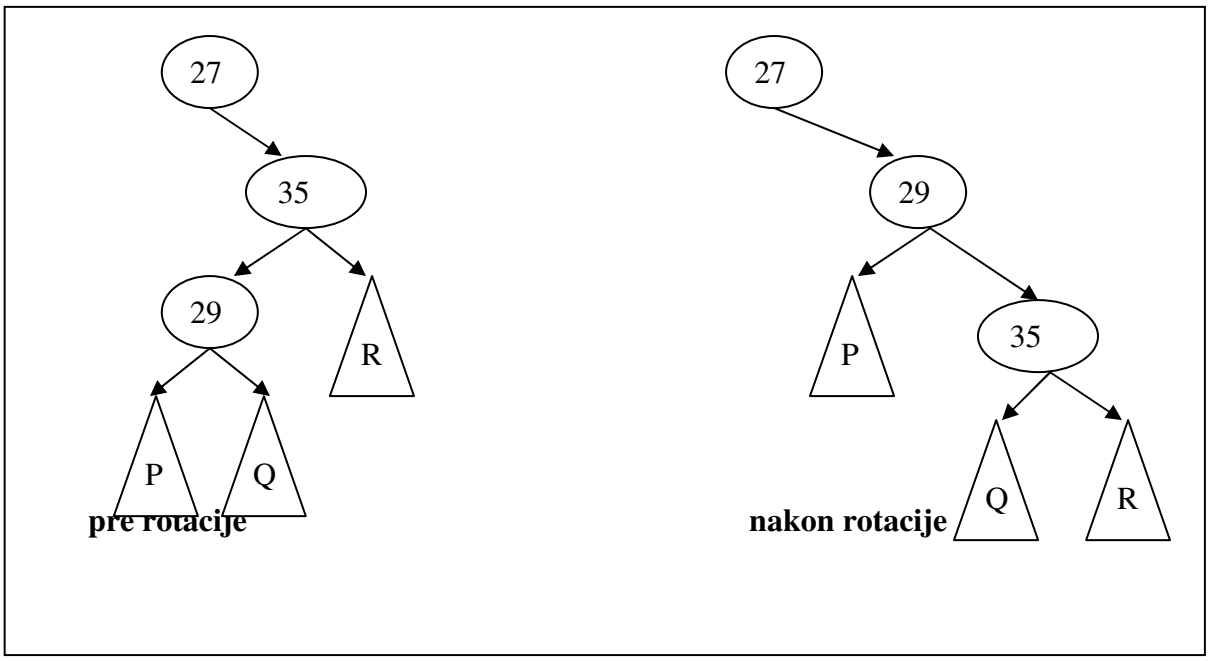
5. Osnovni gradivni blokovi algoritma su rotacije: leva i desna. Obe su simetrične u svom izvođenju.
Desna rotacija čvora *Ch* (= child) oko njegovog roditelja *Par(*=parent*)*:

```
Algortam Desna rotacija (Gr, Par, Ch) //Gr=deda
  if Par nije koren stabla  //npr, if Gr != NULL
    Gr od  Ch postaje roditelj  od Ch zamenom  Ch sa Par;
  Desno poddrvo Q od Ch postaje levo poddrvo od  Ch-ovog
roditelja Par;
  cvor Ch dobija Par kao svoje desno dete;
```

Graficki prikaz desne rotacije



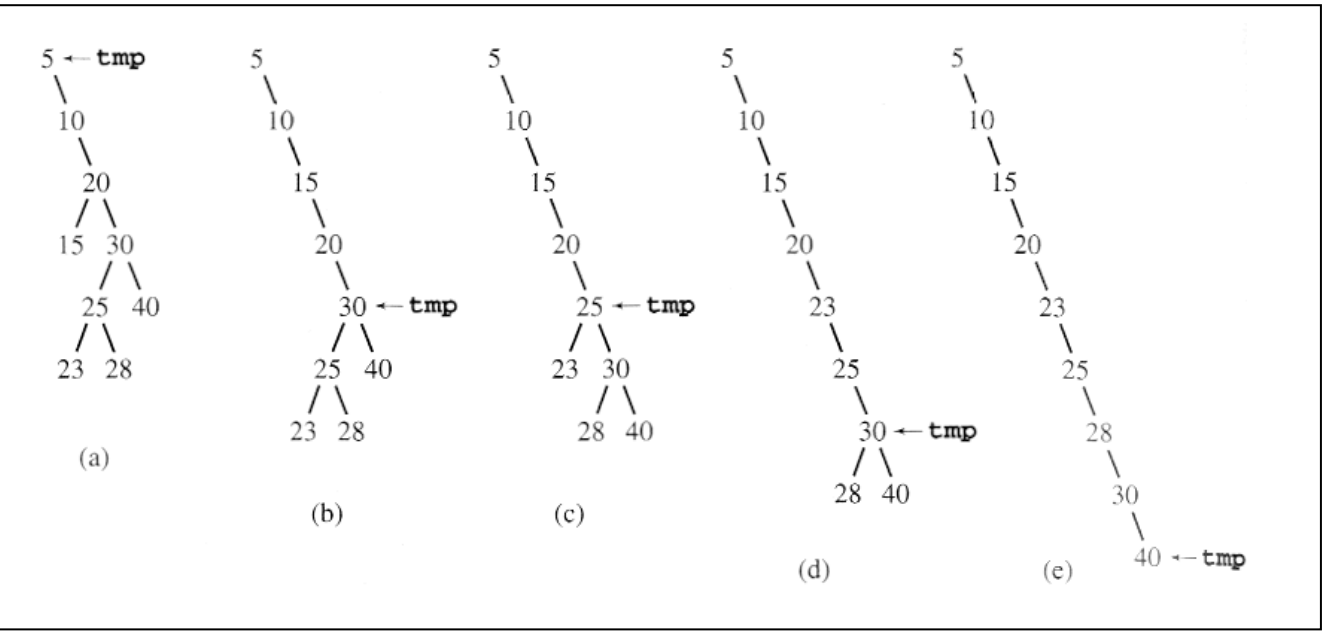
Uočite da 1. i 2. korak u desnoj rotaciji osiguravaju opstanak pretraživačkog svojstva u stablu. Na primer,



Važna pretprocesna radnja DSW algoritma je kreiranje okosnice ili kičme ili gradivne loze stabla(engl. *backbone*, *vine*). U toj fazi algoritma vrši se transformacija datog BST u jednostruko povezanu listu. Dobija se izduženo stablo koje se u nizu prolaza pretvara u perfektno balansirano stablo uzastopnim rotiranjem svakog drugog čvora oko svojih roditelja. U prvoj fazi okosnica se stvara prema sledećem algoritmu:

```
createBackbone (root, n)
  tmp = root;
  while (tmp != null)
    if tmp ima levo dete
       rotiraj to dete oko  tmp;  //levo dete postaje otac od tmp
      postavi  tmp da bude dete od novostvorenog roditelja;
    else postavi  tmp na mesto njegovog desnog deteta;
```

Pogledajmo primer stvaranja okosnice



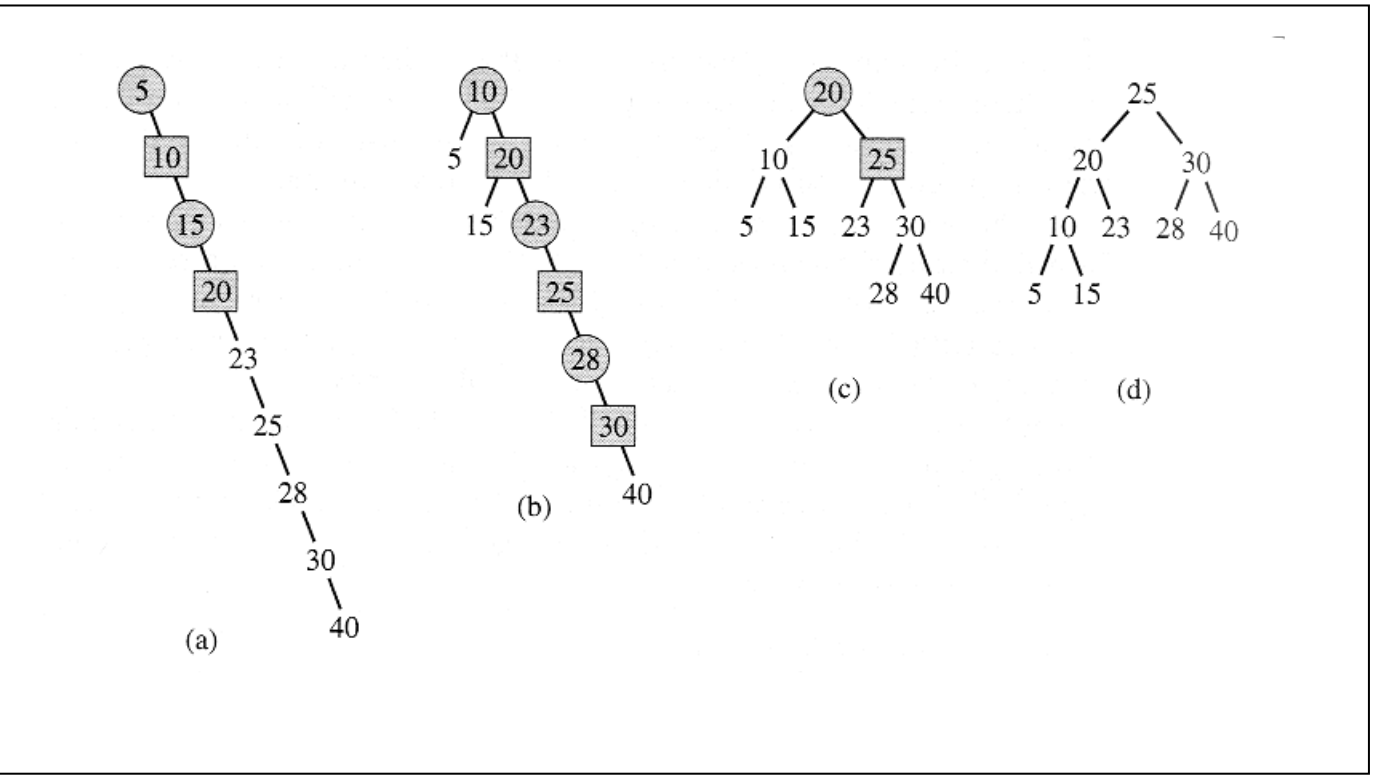
U najboljem slučaju BST je već degenerisano u oblik okosnice i while ciklus se ponavlja *n* puta i ne obavljaju se rotacije.

U najgorem slučaju, kada koren nema desno dete, while ciklus se izvršava *2n-1* puta i obavlja se *n-1* rotacija (*n* = broj čvorova stabla). Dakle, vremenska složenost prve faze DSW algoritma je O(n).

U drugoj fazi, okosnica se pretvara u drvo, ali ovaj put savršeno uravnoteženo što ostavlja listove samo na dva susedna nivoa. U svakom prolazu niz okosnicu, svaki drugi čvor se okreće oko svojih roditelja. Jedan takav prolaz upolovljava veličinu okosnice. Samo prvi prolaz može da ne dođe do kraja okosnice. To se koristi za evidentiranje razlika između broja čvorova n u tekućem stablu i broja $2^{\lfloor \lg(n+1) \rfloor}-1$ čvorova u najbližem kopletnom binarnom stablu.
Algoritam faze 2:

```
createPerfectTree(n)
  m = 2^⌊lg(n+1)⌋ -1;
  obavi n-m rotacija počevši od vrha okosnice;
  while (m > 1)
    m = m/2;
    obavi m rotacija počevši od vrha okosnice;
```

Primer faze 2 u DSW algoritmu



Primer startuje s okosnicom (a) generisanom u fazi 1. Prvi prolaz kroz okosnicu kreira okosnicu (b). Potom se izvršavaju još dva prolaza: U svakoj okosnici prikazani su u obliku kvadrata čvorovi koji se promovišu za jedan nivo putem levih rotacija, a u obliku kružiča prikazani su njihovi roditelji oko kojih se vrše rotacije.

Da bi izračunali složenost faze 2, tj. izgradnju drveta, uočite da je broj iteracija u while ciklusu:

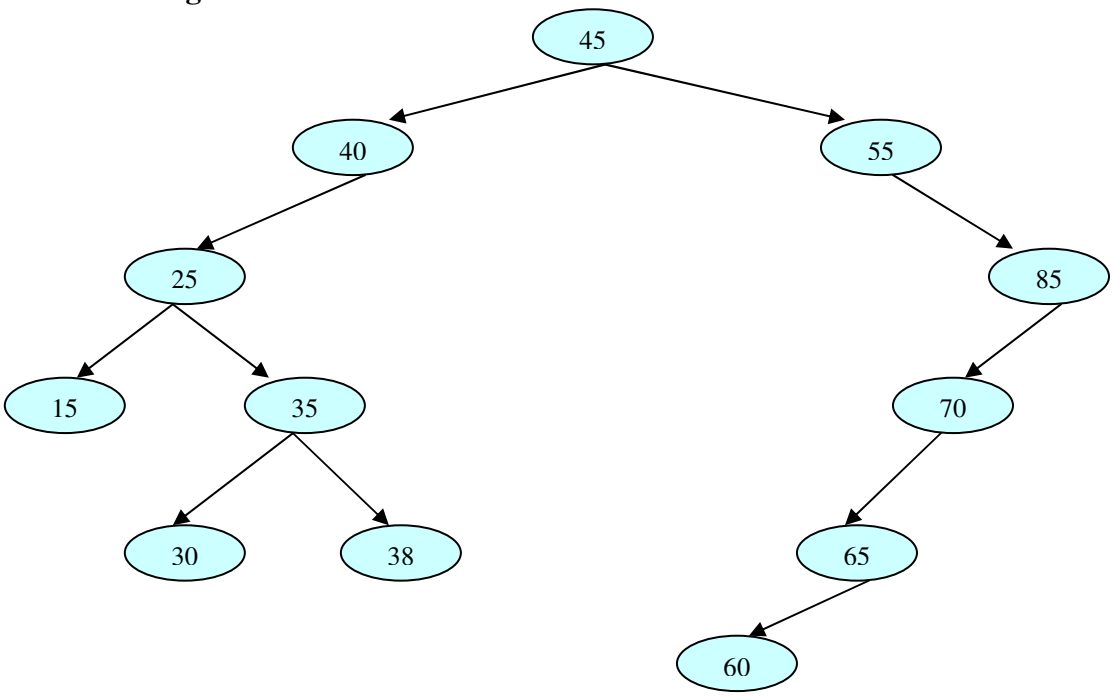
$$\left(2^{\lg(m+1)-1}-1\right)+\cdots+15+7+3+1\Big)=\sum_{i=1}^{\lg(m+1)-1}(2^i-1)=m-\lg(m+1)$$

Dakle, broj rotacija je $n-m+(m-\lg(m+1))=n-\lg(m+1)=n-\lfloor\lg(n+1)\rfloor$

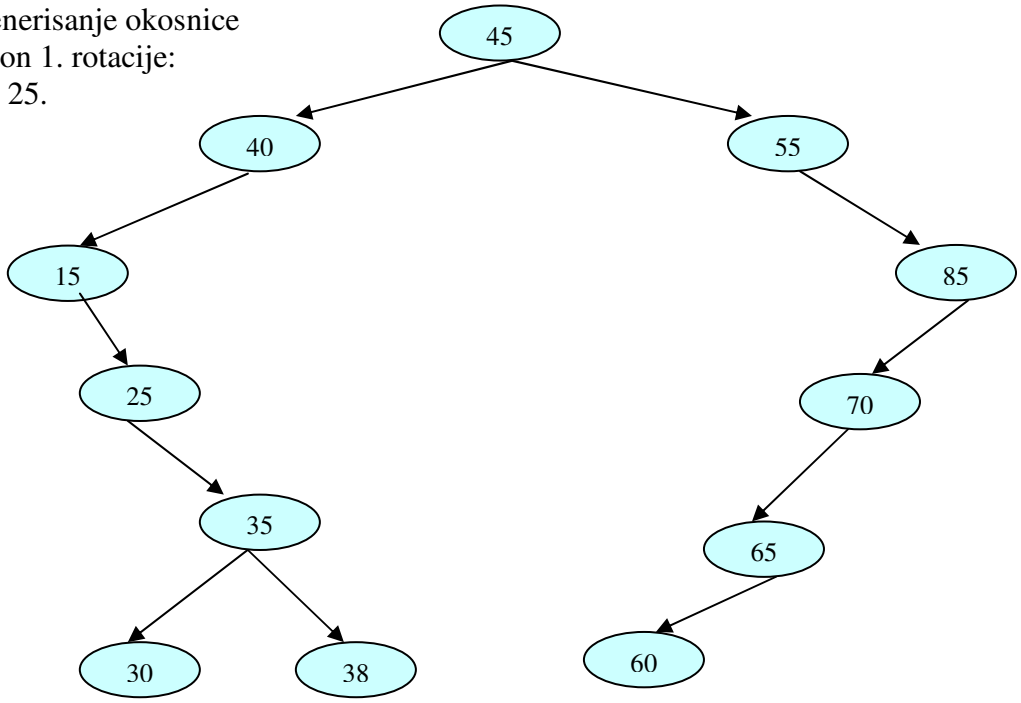
Dakle, broj rotacije je O(n).

Kako je za stvaranje okosnice takođe potrebno najviše O (n) rotacije, trošak globalnog rebalansa u DSW algoritmu je optimalan u smislu vremena, jer raste linearno sa n i zahteva vrlo malo fiksiranog memorijskog prostora za čuvanje međurezultata.

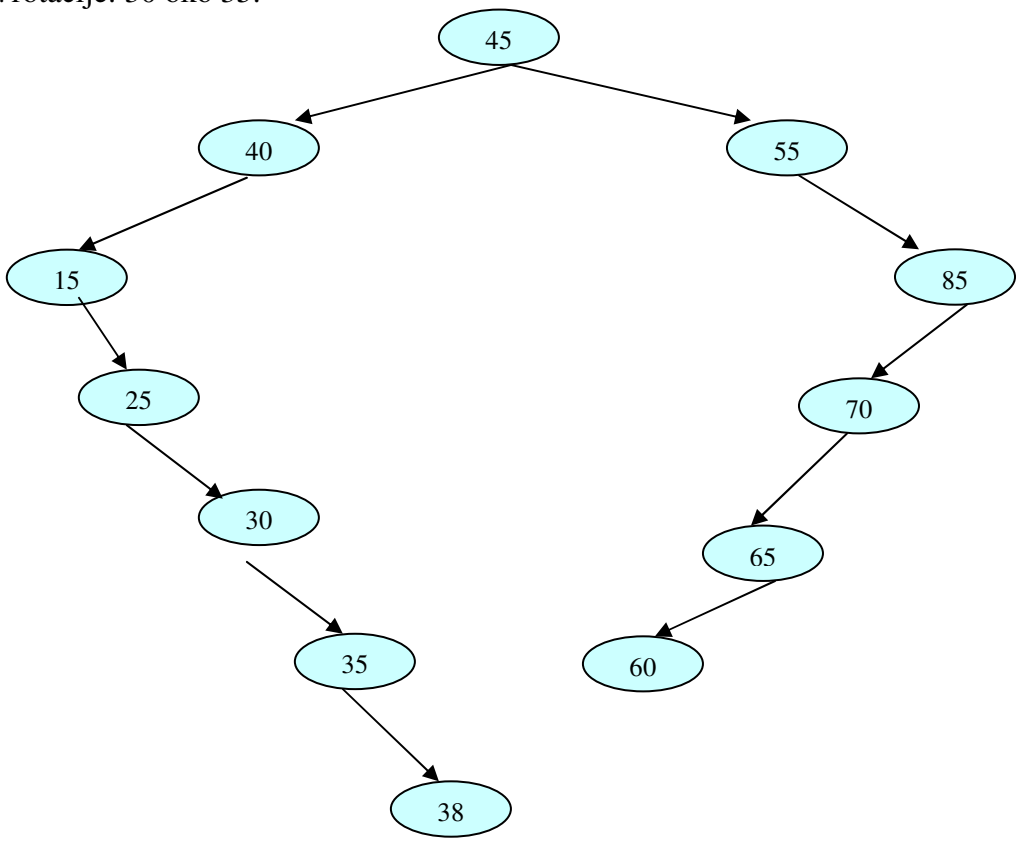
1. Dato je binarno stablo pretrage koje nije balansirano. Upotrebom DSW algoritma kreirajte perfektno balansirani oblik datog stabla.



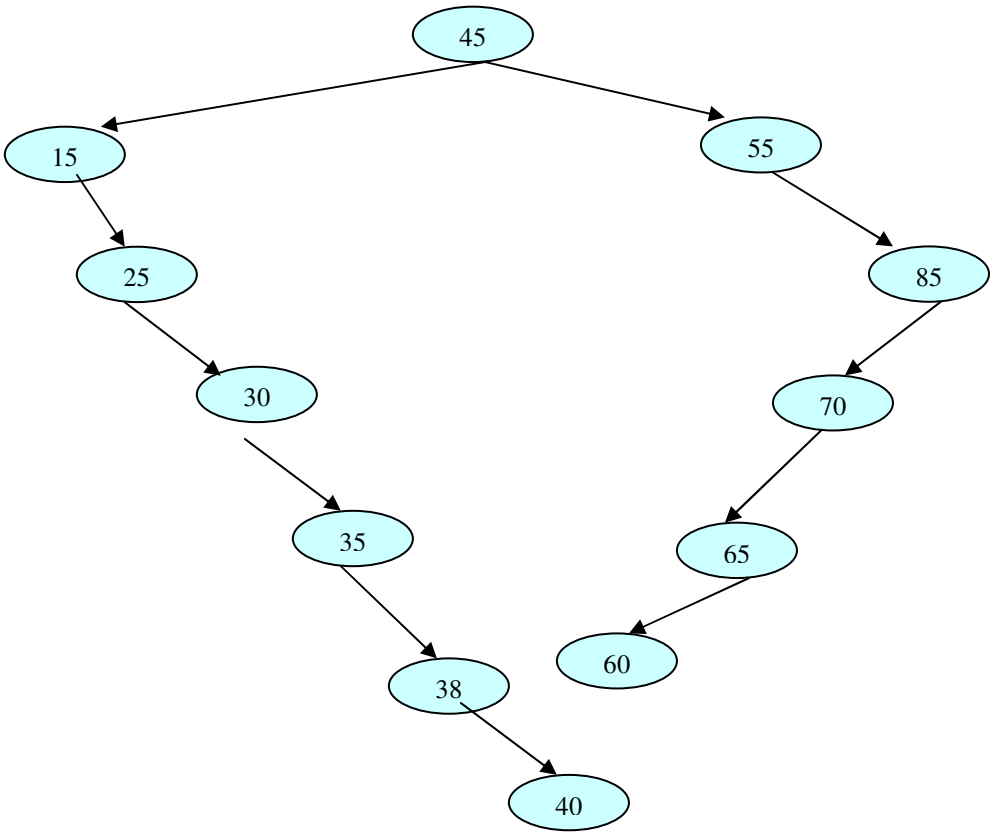
Rešenje:
Faza 0: Generisanje okosnice
Stablo nakon 1. rotacije:
15 oko 25.



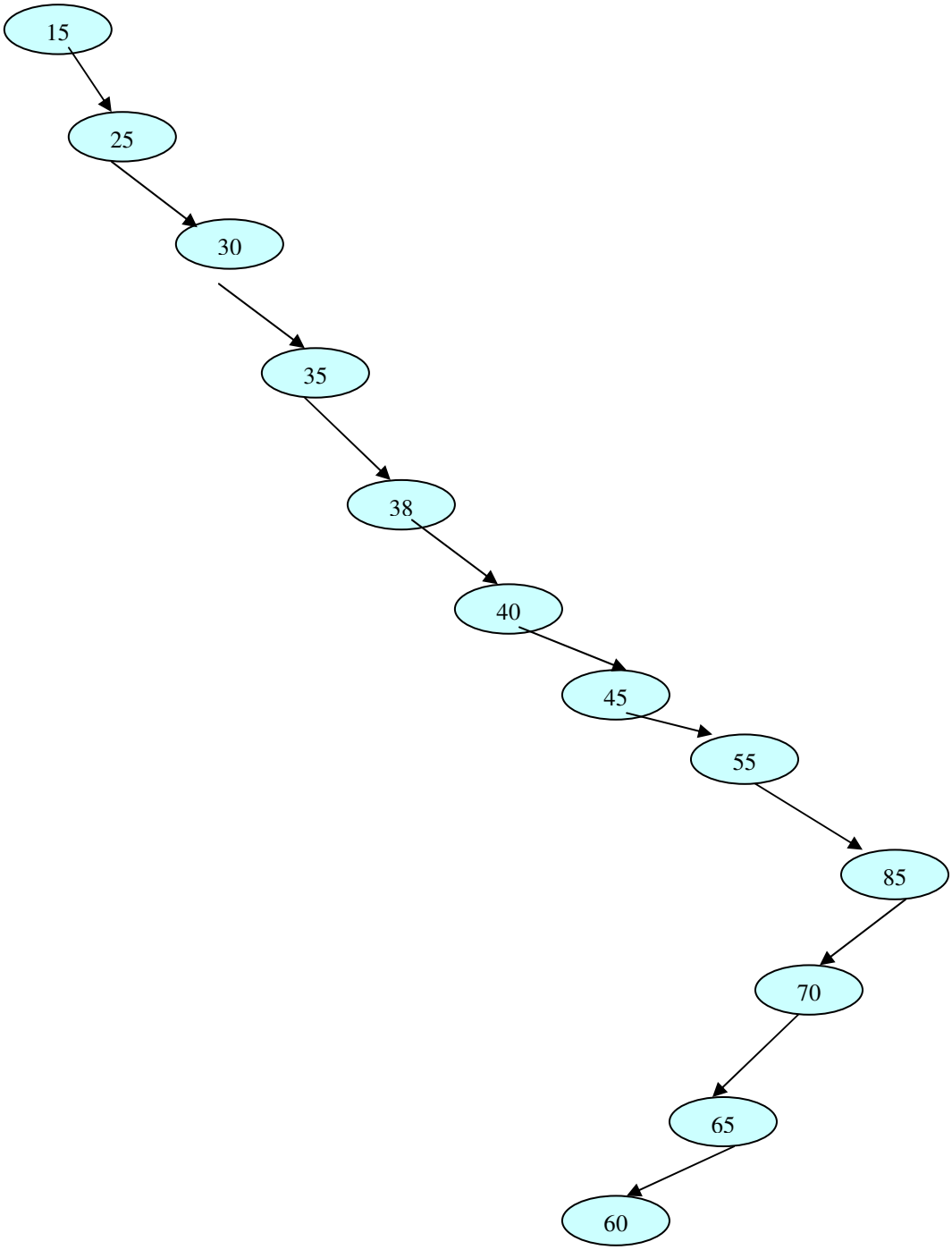
Stablo nakon 2. rotacije: 30 oko 35.



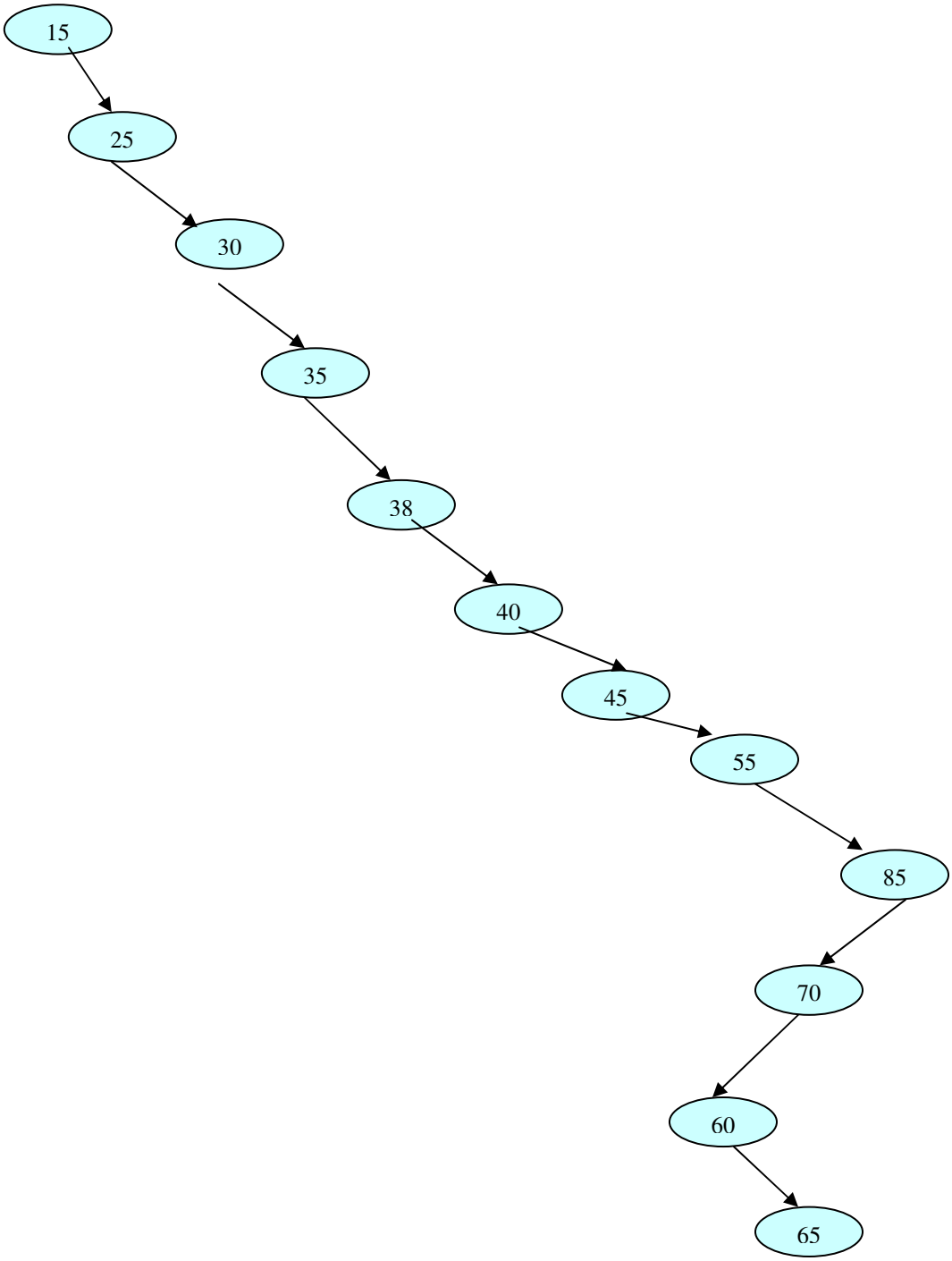
Stablo nakon 3. rotacije: 15 oko 40.



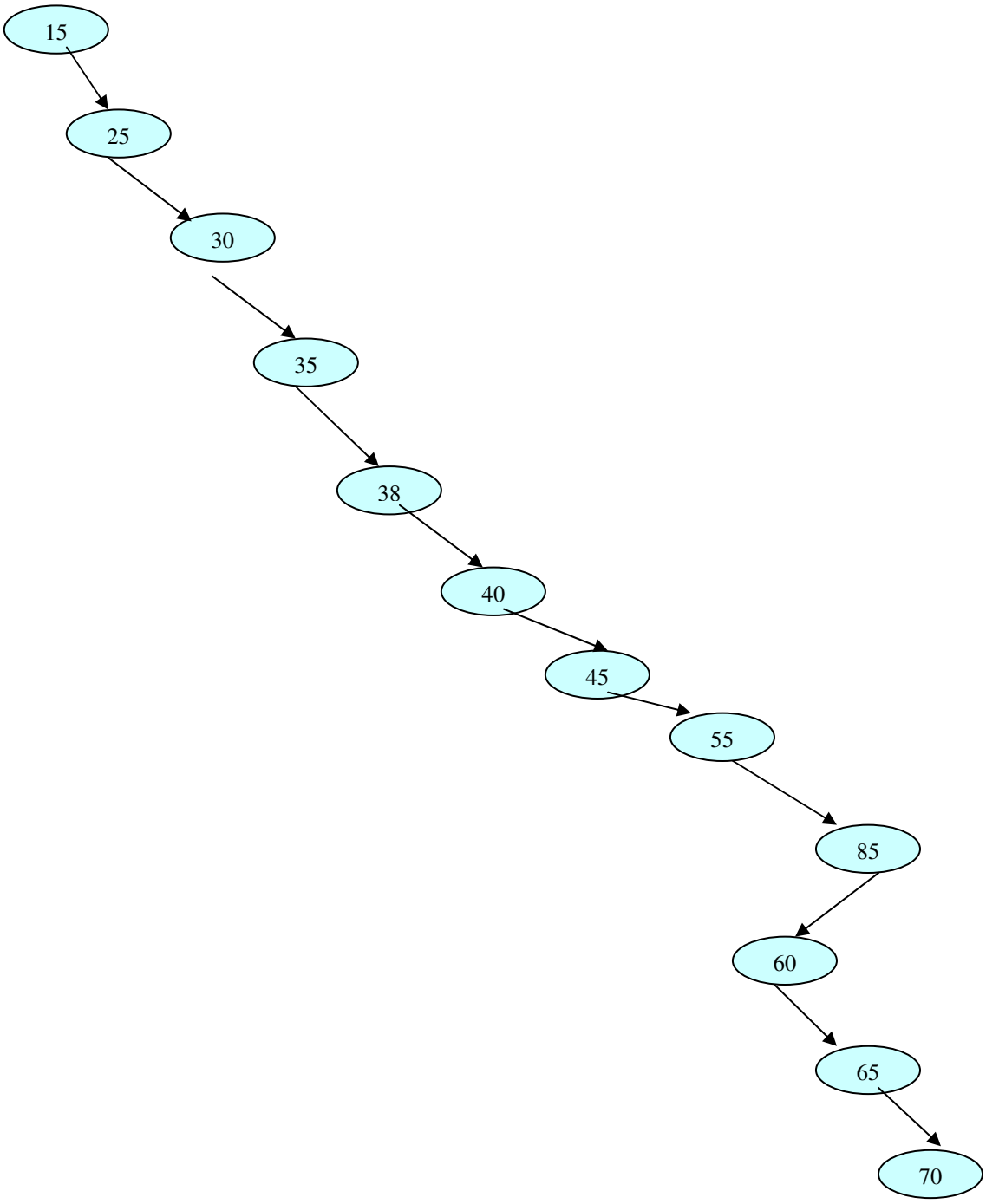
Stablo nakon rotacije: 15 oko 45.



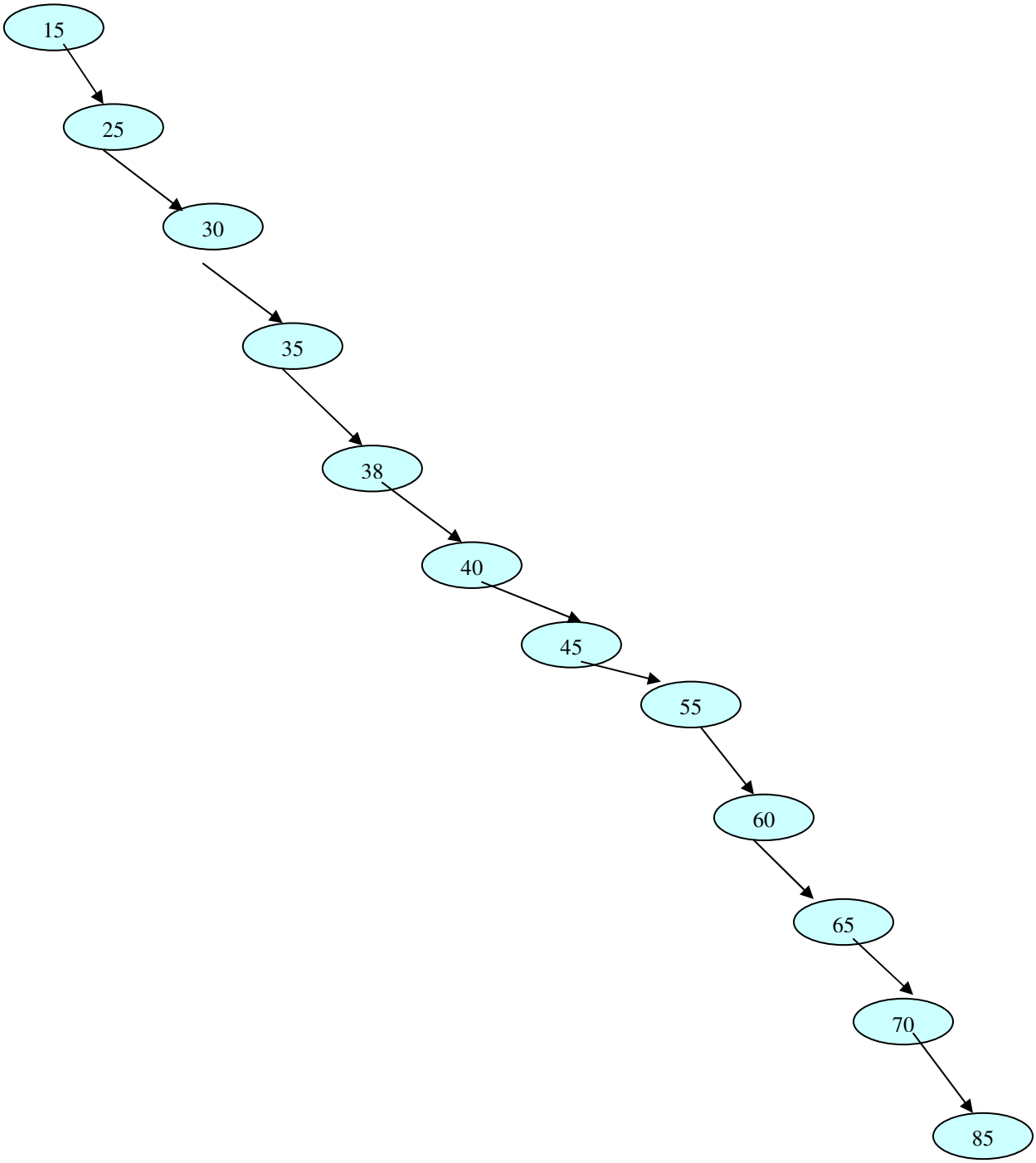
Stablo nakon rotacije: 60 oko 65.



Stablo nakon rotacije: 60 oko 70

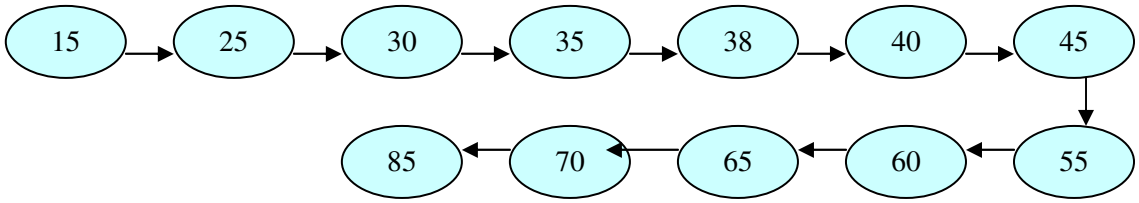


Stablo nakon rotacije: 60 oko 85



Rezultat: okosnica, tj. sortirana lista

Faza 1: Uspostavljanje okosnice

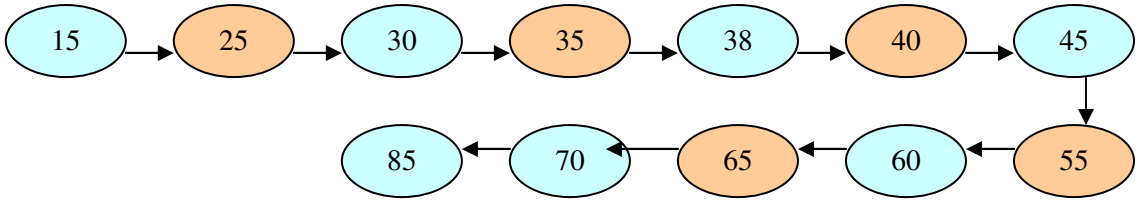


Faza 2: Izvođenje potrebnog broja početnih rotacija.

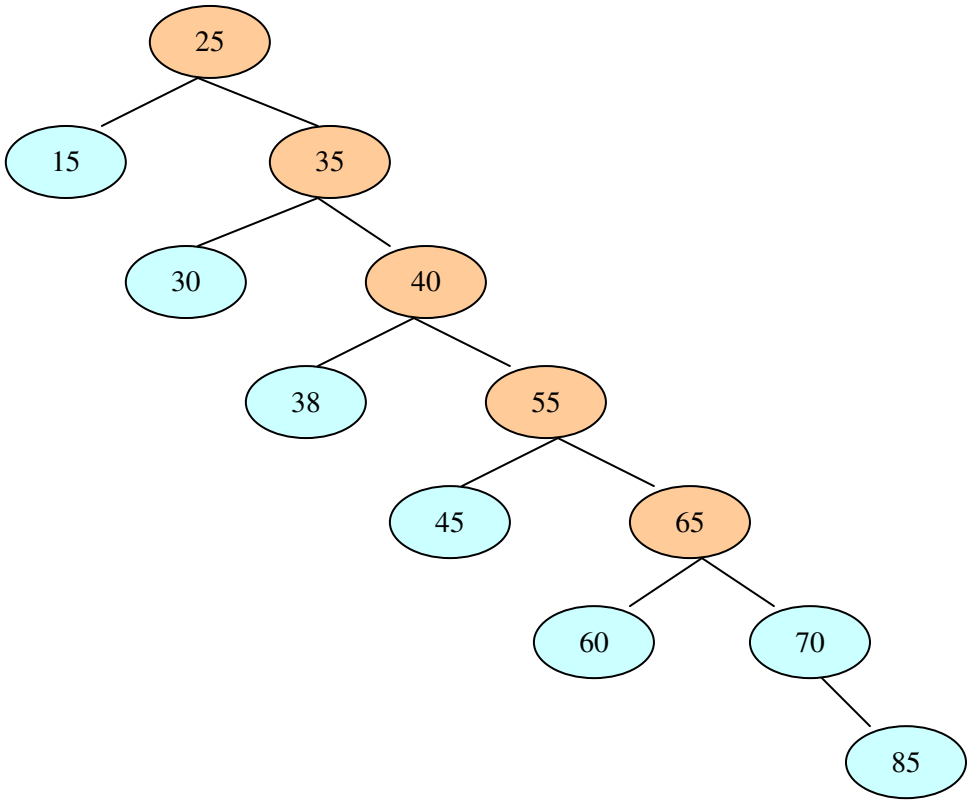
Postoji 12 čvorova u ovom stablu, te je $n = 12$.

$m = 2^{\lfloor \log_2(n+1) \rfloor} - 1 = 2^{\lfloor \log_2(12+1) \rfloor} - 1 = 2^3 - 1 = 7$

Izvođenje ukupno $n - m = 12 - 7 = 5$ početnih rotacija na okosnici.



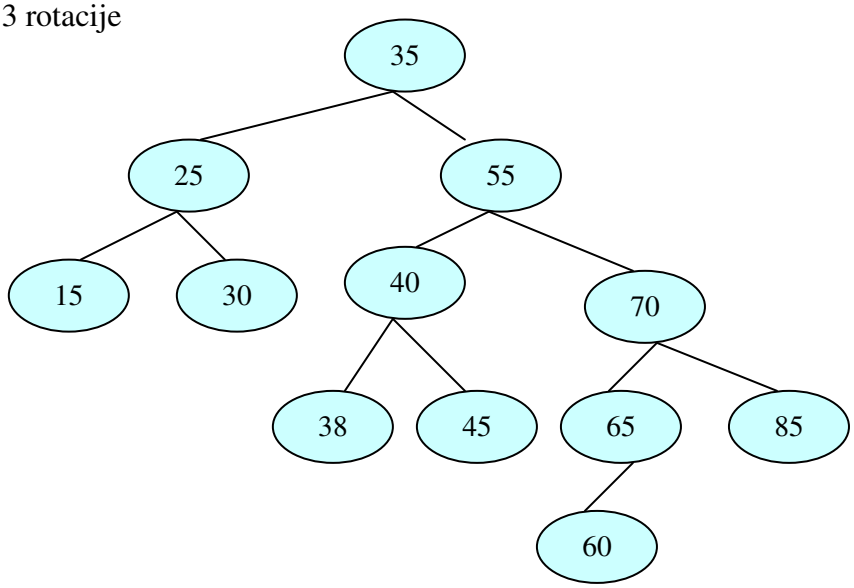
Rotiranje oko tih čvorova (deca rotacijske tačke označeni su gore u okosnici) proizvodi sledeću okosnicu.



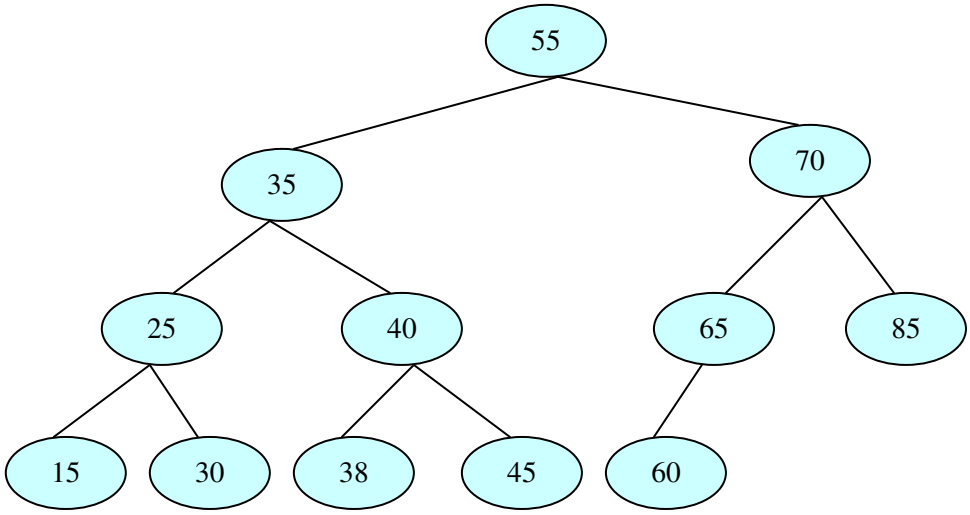
Okosnica nakon početnih rotacija

Faza 3: Izvođenje potrebnog broja rotacija da bi se dobilo perfektno balansirano stablo. Budući da m = 7 i njegova vrednost je prepolovljena nakon svake rotacije i kako rotacija prestaje kada m = 1, mi ćemo izvesti ukupno četiri rotacije (kada m = 7 uraditi sledeće 3 rotacije: 35 oko 25, 55 oko 40, 70 oko 65,
kada m = 3 uraditi sledeću 1 rotaciju: 55 oko 35).

Prve 3 rotacije



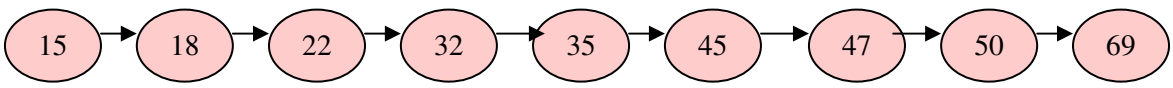
Poslednja rotacija 55 oko35.



Rezultujuće perfektno balansirano stablo.

2. Dato je binarno stablo pretrage koje je nakon niza operacija umetanja i brisanja postalo nebalansirano. Upotrebom DSW algoritma rebalansirajte dato stablo.

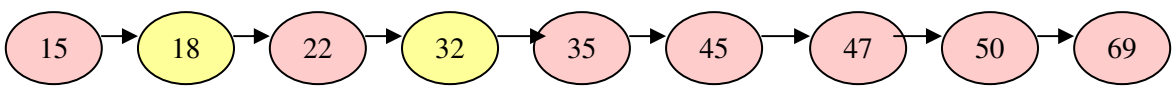
Faza1: Okosnica je:

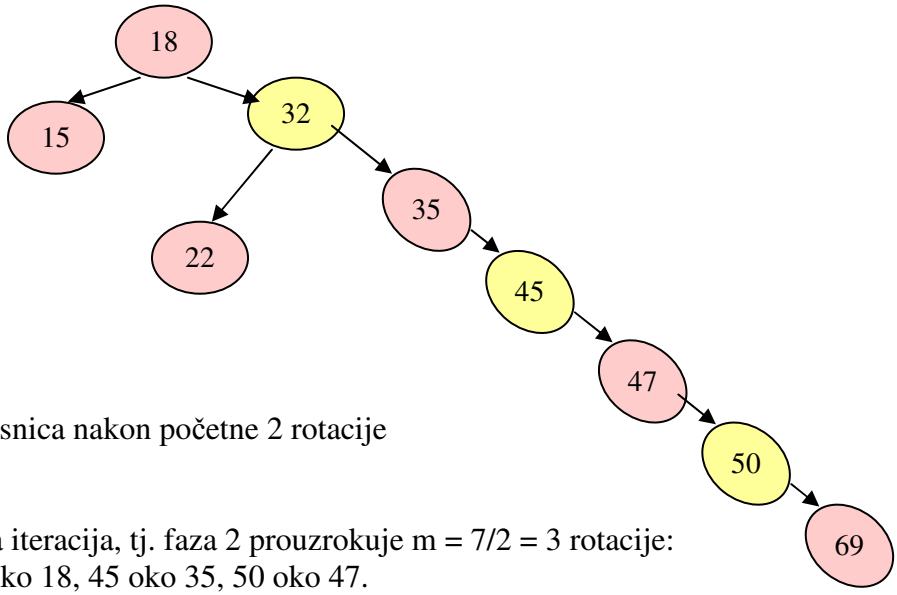


Za n = 9, m = $2^{\lfloor \log_2(n+1) \rfloor} - 1 = 2^3\text{-}1 = 7$

Broj početnih rotacije je n – m = 2

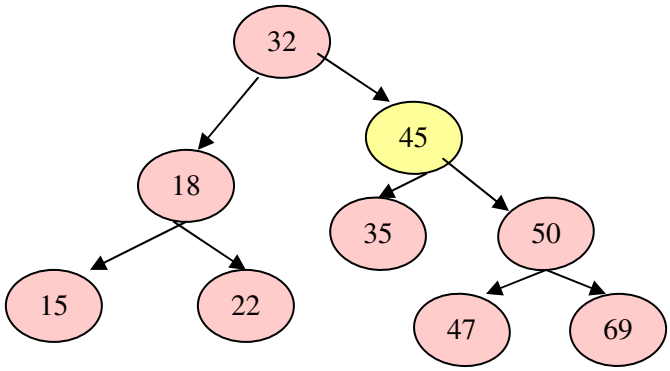
Faza 2: Rotacije: 18 oko 15, 32 oko 22



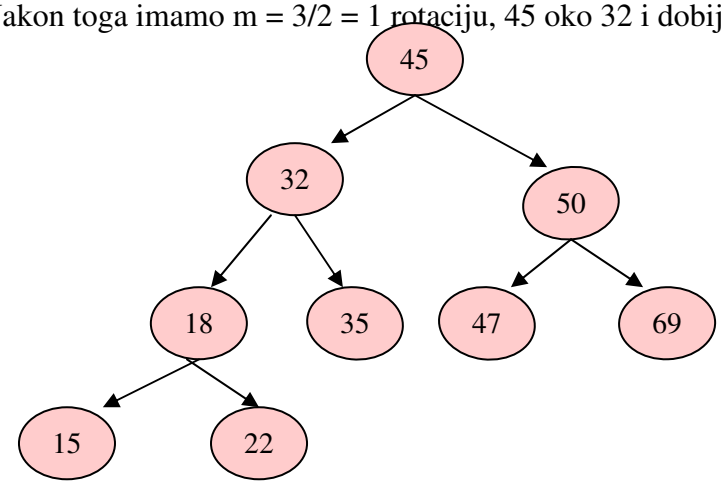


Okosnica nakon početne 2 rotacije

Prva iteracija, tj. faza 2 prouzrokuje m = 7/2 = 3 rotacije:
32 oko 18, 45 oko 35, 50 oko 47.



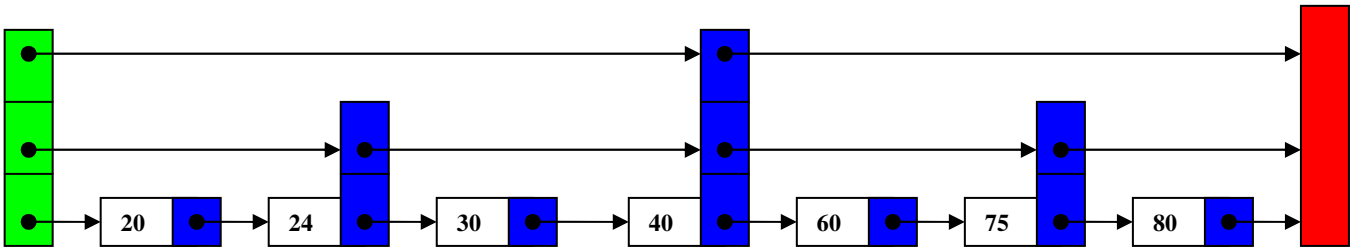
Nakon toga imamo m = 3/2 = 1 rotaciju, 45 oko 32 i dobijamo konačno perfektno balansirano stablo.



Skip liste

Skip liste su probabilistička struktura podataka, bazirana na paralelnim povezanim listama. Predstavljaju alternativu balansiranim stablima. Iako skip liste imaju loše performanse za odabir najgoreg slučaja, ni jedan ulazni niz ne proizvodi konzistentno najlošije performanse. Mala je verovatnoća da skip lista bude značajnije nebalansirana. Balansiranje strukture podataka probailistički jednostavnije je od eksplicitnog održavanja ravnoteže.
Ako je lista spremljena u rastućem / opadajućem redosledu i svaki drugi čvor liste ima pokazivač na čvor udaljen za dva mesta unapred, maksimalni broj koraka u potrazi je n/2 +1 čvorova. Ako dodate svaki četvrti, onda broj koraka u potrazi je n/4 +1...
Ako svaki 2^iti ima pokazivače na 2^i ispred njega, broj čvorova koje treba ispitati je log n, a broj pokazivača je samo dupliran.
Čvor sa k pokazivača nazivamo čvor sa k nivoa. Ako svaki 2^i-ti čvor ima pokazivač na čvor 2^i mesta unapred, tada su nivoi čvorova raspodeljeni na sledeći način:
50% nivo 1,
25% nivo 2,
12,5% nivo 3 ...
Kada bi nivoi čvorova bili proizvoljno birani ali u istom odnosu, čvorov i-ti pokazivač pokazuje na sledeći čvor nivoa *i* ili većeg. Umetanja i brisanja bi zahtevala samo lokalne izmene i nivo čvora se ne bi menjao.

Primer skip liste sa pokazivačima na svakom drugom čvoru:



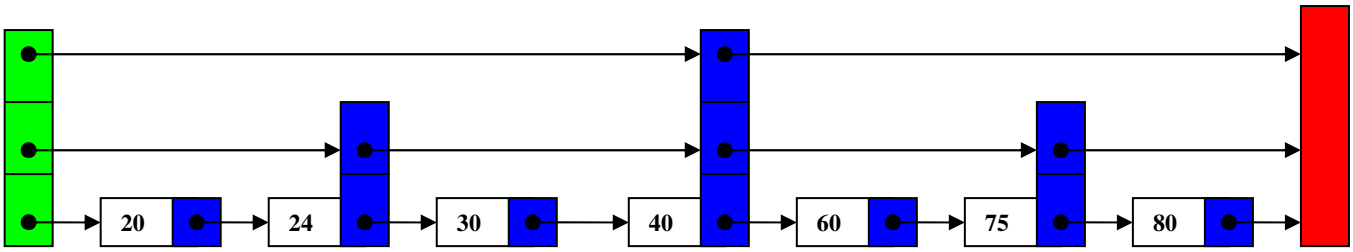
Unutar gornjeg primera, postoje tri odvojene povezane liste unutar skip liste. Lista na nivou 0 je uređena povezana lista koja sadrži pokazivače između svaka dva uzastopna čvora u listi. Lista na nivou 1 sadrži pokazivače između svakog drugog čvora u listi. Lista na nivou 2 sadrži pokazivače između svakog četvrtog čvora u listi.

Uopšteno važi da element skip liste je element nivoa ***i akko*** se nalazi u listi od nivoa 0 do *i* i nije na nivou *i+1* (ako taj nivo postoji).

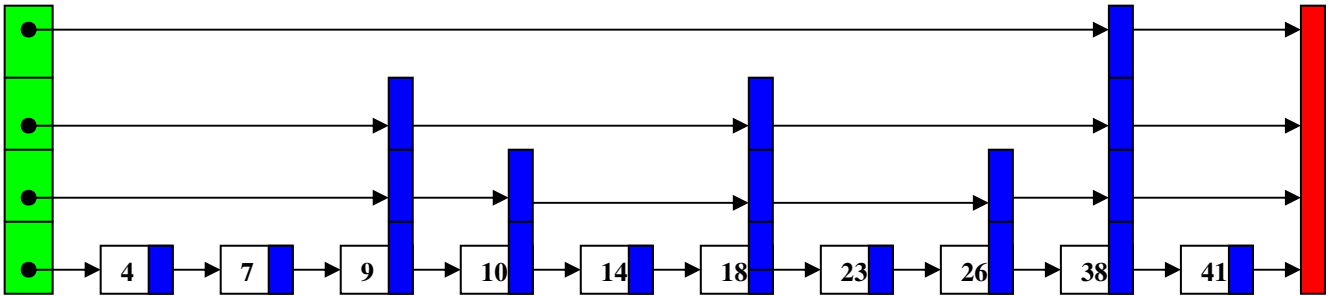
U gornjem primeru, element 40 je jedini element nivoa 2, jer se pojavljuje redom na nivoima 0, 1, 2 (ne postoji nivo 3 u ovom primeru).
Elementi 24 i 75 su elementi nivoa 1 (pojavljuju se na nivoima 0 i 1, ali ne i na nivou 2).
Element 20, 30, 60, 80 su elementi nivoa 0,
Uopšteno , nivo *i* liste uključuje svaki 2^i ti element koji se pojavljuje u listi na nivou 0.

Dakle, **skip list** je struktura koja sadrži hijerarhiju uređenih jednostruko povezanih listi.

Postoje dva pojavna oblika skip liste: **regularna i iregularna.**
Regularna skip lista je ona u kojoj odabrani podskupovi generišu parno razdvojene preskoke, skip-ove.
Iregularna skip lista je ona u kojoj odabrani podskupovi ne generišu parno razdvojene preskoke, skip-ove.



Regularna Skip Lista



Iregularna Skip Lista

Ako uporedimo regularne i iregularne skip liste, mnogo je teže analizirati performanse iregularnih skip listi pri slučajnoj pretrazi ili nizu slučajnih pretraga. Često su skipovi kod iregularnih skip listi generisani na slučajan način.

Regularne skip liste

Ako regularna skip lista ima *n* čvorova, onda važi:

Za svako *k*, *i* takvo da $1 \le k \le \lfloor \lg n \rfloor$, $1 \le i \le \lfloor n/2^{k-1} \rfloor - 1$, čvor na poziciji $2^{k-1} \times i$ pokazuje na čvor na poziciji $2^{k-1} \times (i+1)$

Dakle, to znači da svaki drugi čvor pokazuje na naredni čvor udaljen za dve pozicije (sused njegovog suseda), svaki četvrti čvor pokazuje na naredni čvor udaljen za 4 pozicije,...
Dalje, to znači da
polovina čvorova ima samo jedno pokazivačko polje,
četvrtina čvorova ima dva pokazivačko polje,
osmina čvorova imaju tri pokazivačka polja,...

Broj pokazivačkih polja u datom čvoru predstavlja nivo tog čvora, a broj nivoa u regularnoj skip listi je:

$$\text{maximum\_level} = \lfloor \lg n \rfloor + 1.$$

Pretraga elementa u regularnoj skip listi

1. Pogledajte regularnu skip listu u gornjem primeru i ispišete sve posećene čvorove prilikom pretrage sledećih vrednosti u skip listi: **65, 75, 80, 40**

Rešenje
(a): glava, 40, rep, 40, 75, 40, 60, 75 – kraj, neuspeh

(b): glava, 40, rep, 40, 75 – kraj, uspeh

(c): glava, 40, rep, 40, 75, rep, 75, 80 – kraj, uspeh

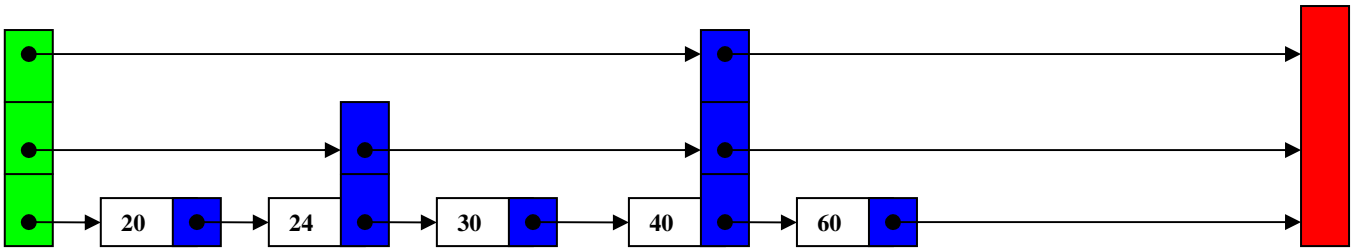
(d): glava, 40 – kraj, uspeh

Algoritam pretrage u regularnoj skip listi:

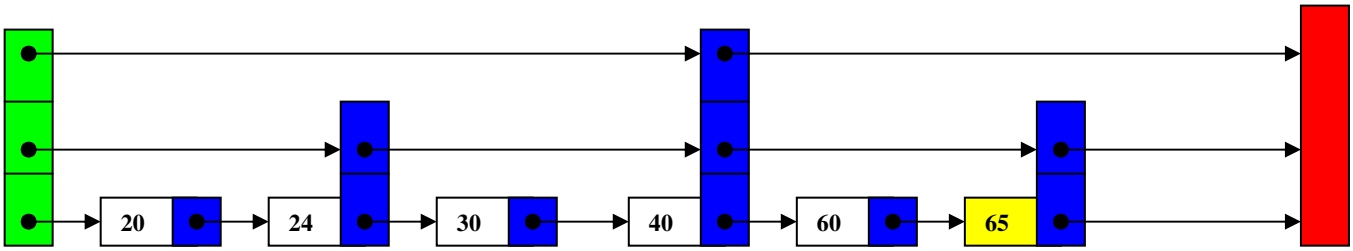
```
search_regular_skip_list (x)
   p = non null list na najvišem nivou i;
     while (x notfound and i > 0)
    {   if p.key < x
            p = podlista koja počinje sa prethodnikom od p na nivou  --i;
        else  if p.key > x
                if p je poslednji čvor na nivou  i
                    p = non null  podlista koja počinje sa p na najvišem nivou  < i;
                    i = broj novog nivoa;
                else p = p.next;
```

Vremenska složenost pretrage unutar regularne skip liste je $O(\log_2 n)$. Dakle, slično binarnoj pretrazi.

Umetanje u regularnu skip listi bez povećnja nivoa
1. U datu skip listu sa 5 članova umetni broj 65.



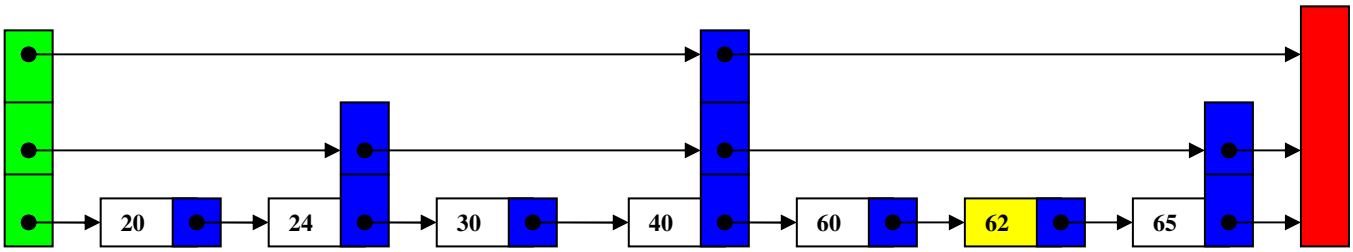
Rešenje: Pošto ova lista ima m=2 nivoa i n=5 članova onda lista **sadrži manje od maksimalnog broja čvorova skip listu nivoa 2. Otuda, postoje mesta da se umetne čvor bez povećanja broja nivoa. Zapravo sve dok se ne desi da $3 < \lfloor \log n \rfloor + 1$, broj nivoa se ne mora povećavati.**
Sada pretpostavimo da novi čvor koji sadrži 65 će biti umetnut u listu. Kako bi se utvrdio nivo čvora koji će sadržati 65, generiše se slučajan broj između 1 i 7. Pretpostavimo da taj slučajni broj je pet. Prema tablici umetanja, to znači da će 65 biti smešteno u čvoru nivoa 1 kao na slici:



rand_num	Level of node to be inserted (-1)	possible occurrences
7	3	1
5-6	2	2
1-4	1	4

Primer 2: U skip listu iz primera 1 ubacite vrednost 62.
Rešenje:
Uočimo da skip list iz primera 1 je i nakon umetanja ostala regularna skip lista.

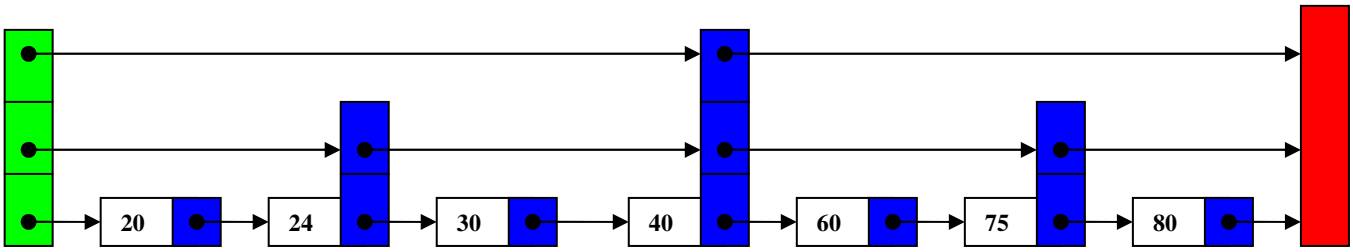
Ali, kako već postoji čvor 40 na nivou 2, to znači da ne može postojati drugi čvor na nivou 2 u ovoj strukturi.
Jer za regularnu listu sa 6 čvorova, očekujemo 6/4 čvora na 2. nivou, 6/2 čvora na nivou 1. Kako, pak postoji već tri čvora na nivou 1, sledi da 62 mora se smestiti u čvor na nuvou 0 kao što je prikazano na slici:



Uočite da nakon umetanja čvora 62 skip lista je postala iregularna, što je nekad jeftinije (u smislu vremena i potrebne memorije) nego restrukturiranje rezultujuće skip liste tako da ona ostane regularna.

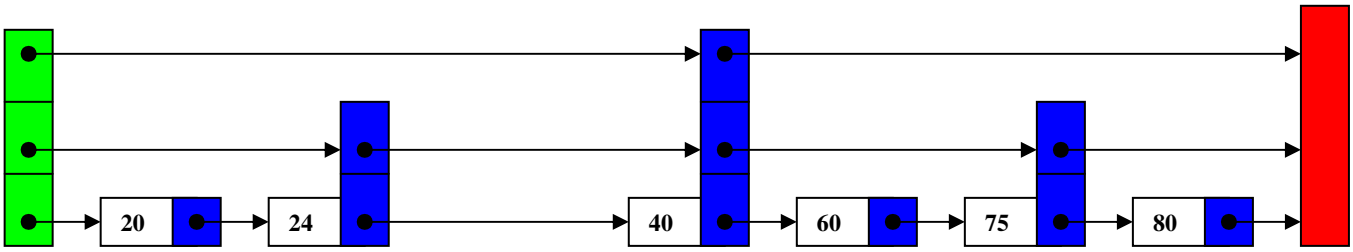
Brisanje iz regularne skip liste

1. Iz date skip listu ukloni redom članove 30, 24, 20, 65, 70, 80

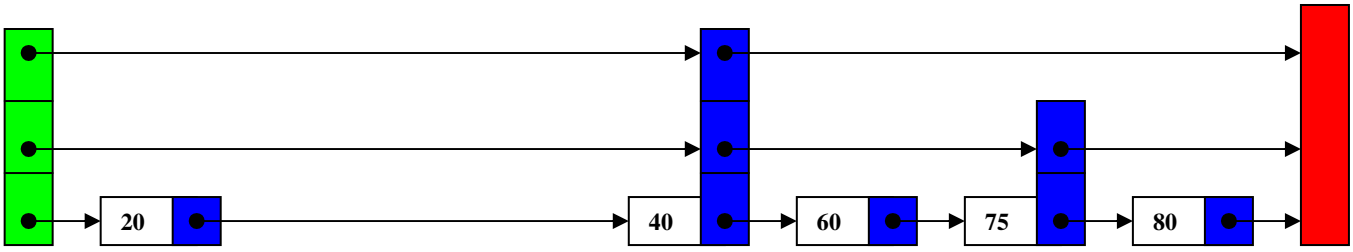


RešenjeČ

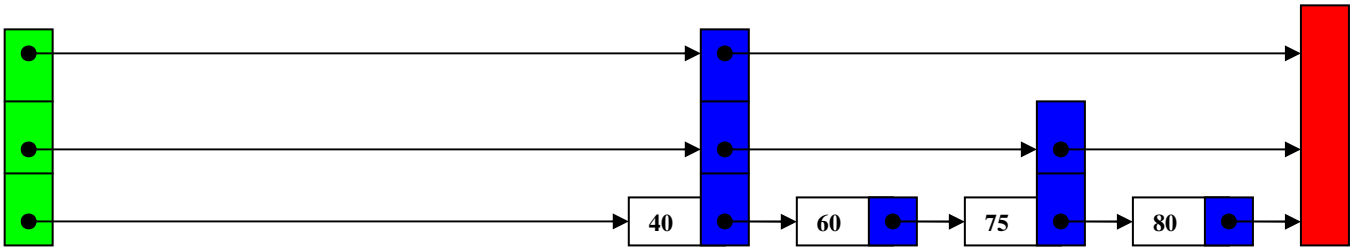
Uklonimo 30



Lista postaje iregularna. Uklonimo 24

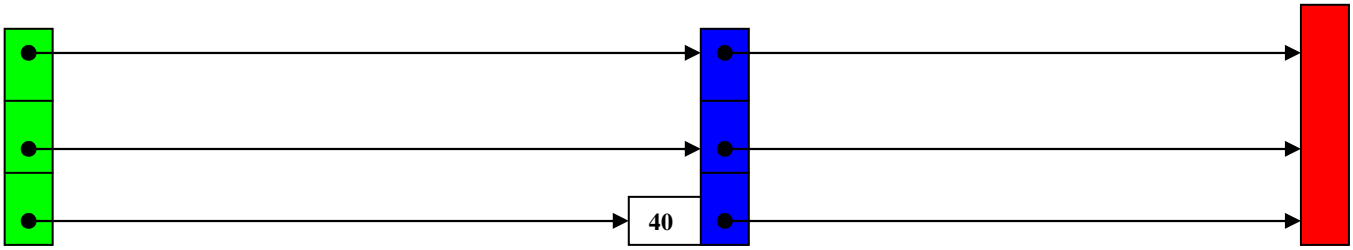


Lista postaje iregularna. Uklonimo 20



Uočite da u ovakvoj listi vremenska složenost svake pretrage čvora x<40 će biti ista.

Uklonimo 60, 75, 80 i dobijamo jednočlanu skip listu:



1. Osnovne operacije sa skip-listom:

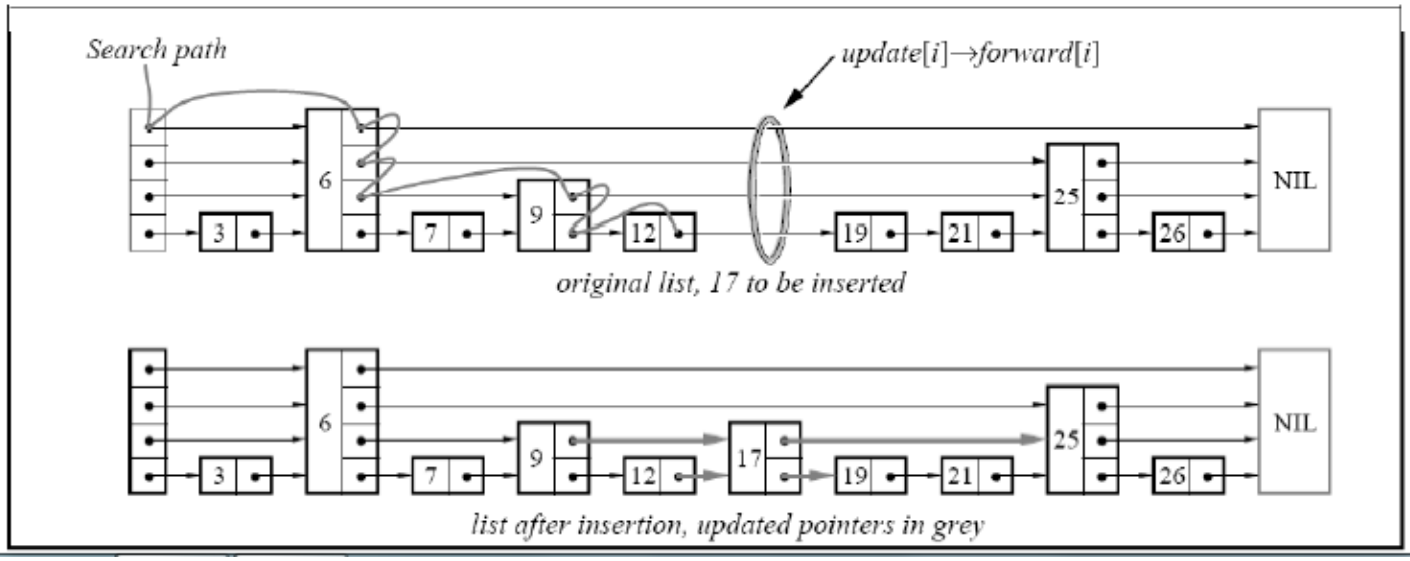
Algoritam Search(S, k)
Ulaz S `(data skip lista),` k `(ključ elementa koji tražimo)`
Izlaz `vrednost elementa sa ključem` k

```
begin
    x := S → header;
    for i := S → level downto 1
        while (x → forward[i] → key < k) do
            x := x → forward[i];
    x := x → forward[1];
    if (x → key == k) then return x → value;
        else return failure;
end
```

Algoritam randomLevel()

Izlaz `nivo novog čvora generisan na slučajan način tako da važi da procenat` p `čvorova nivoa` i `ima` $(i+1)$`-vi pokazivač`

```
begin
    |v| := 1;
    while (random() < p and |v| < maxLevel) do
        |v| := |v| + 1;
    return |v|;
end
```



Algoritam Insert(S, k, v)

Ulaz S (data skip lista), k (ključ elementa koji dodajemo), v (vrednost elementa koji dodajemo)

Izlaz skip lista sa dodatim elementom

```
begin
    x := S → header;
    for i := S → level downto 1
        while (x → forward[i] → key < k) do
            x := x → forward[i];
        update[i] := x; /* pamtimo prethodni element nivoa i */
    x := x → forward[1];
    if (x → key == k) then x → value := v;
    else
        |v| :=randomLevel();
        if |v| > S → level then
            for i := S → level + 1 to |v| do
                update[i] := S → header;
            S → level := |v|;
        x :=makeNode(|v|, k, v);
```

```
        for i := 1 to level do
            x → forward[i] := update[i] → forward[i];
            update[i] → forward[i] := x;
end
```

Algoritam Delete(S, k)
Ulaz S (data skip lista), k (ključ elementa koji brišemo)
Izlaz skip lista sa obrisanim elementom

```
begin
    x := S → header;
    for i := S → level downto 1
        while (x → forward[i] → key < k) do
            x := x → forward[i];
        update[i] := x;
    x := x → forward[1];

    if (x → key == k) then
        for i := 1 to S → level do
            /* ako je i preskočilo nivo datog čvora,
            izlazimo iz petlje */
            if update[i] → forward[i] ≠ x then break;
            update[i] → forward[i] := x → forward[i];
        free x;
        /* ako je to bio čvor najvišeg nivoa
        dekrementiramo nivo liste */
        while (S → level > 1) and
            (S → header → forward[S → level] == NIL) do
                S → level := S → level − 1;
end
```

2.

Napisati algoritam Select(S, k) za odredjivanje k-tog najvećeg elementa skip liste S koja ima n elemenata. Dozvoljeno je dodati polja svakom od elemenata liste S. Složenost algoritma treba da bude $O(\log n)$.

Algoritam Select(S, k)

Ulaz S (data skip lista), k (rang elementa koji tražimo)

Izlaz k-ti najveći element

```
begin
    p := S → header;
    pos := 0;
    for i := S → level downto 1
        while (pos + d(p,i) ≤ k) /* d(p,i) je broj elemenata u
listi od cvora p do p → forward[i] */
            pos := pos + d(p,i);
            p := p → forward[i];
        if (pos == k)
            return p;
end
```

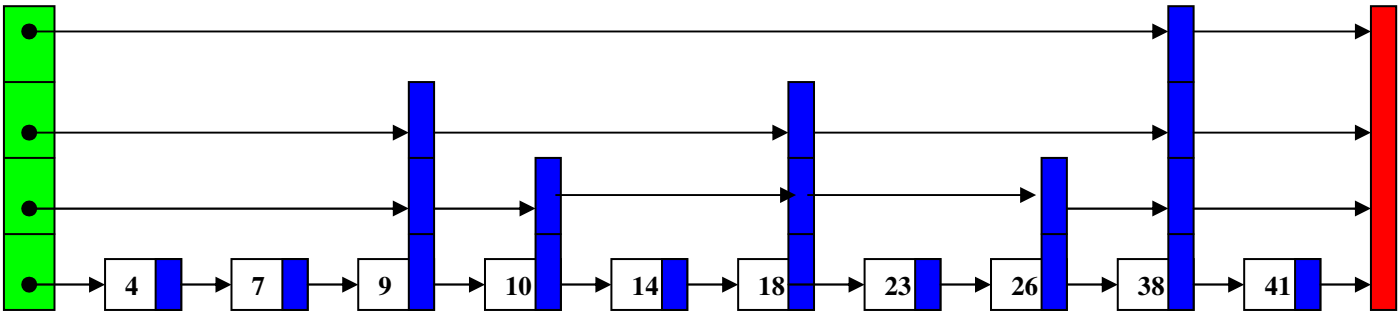
3. Napisati algoritam koji konstruiše skip listu S na osnovu zadatog binarnog stabla pretrage T koje ima n elemenata, tako da vremenska složenost svake od operacija za S u najgorem slučaju bude O(log n). Stablo T može biti nebalansirano. Vremenska složenost algoritma treba da bude O(n).

```
Algoritam Build(T, S)
Ulaz T (dato stablo)
Izlaz S (skip lista)
begin
    S_1 := INORDER(T);
    i := 1;
    while (i < log n) /* konstruisemo listu S_{i+1} */
        for j := 1 to |S_i|
            if (mod(j, 2) == 0)
                S_{i+1}.add(S_i[j]); /* dodajemo svaki drugi element
prethodne liste */
                S_{i+1}[|S_{i+1}|-1].setChildPtr(S_i[j]); /* od prethodnog
elementa te liste dodajemo pokazivac ka trenutnom */
        i++;
end
```

4. Data je iregularna 4-nivo skip lista.

(a) Ako tražimo vrednost 23, izlistajte tačan redosled u kom su čvorovi liste posećeni tokom pretrage.

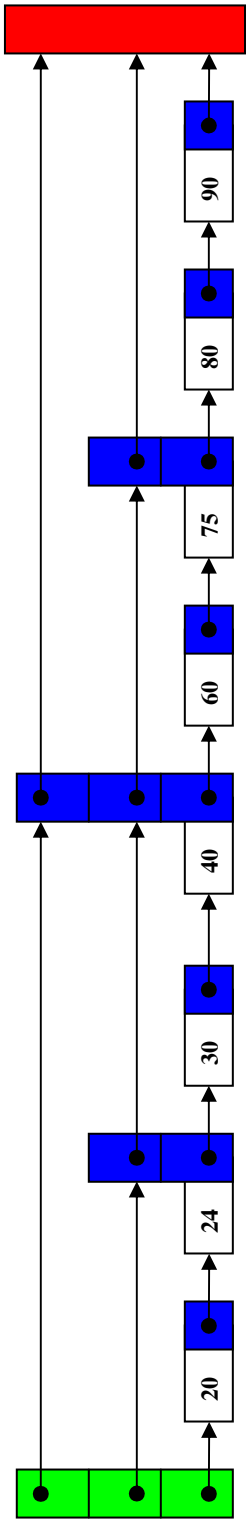
(b) Ako tražimo vrednost 49, izlistajte tačan redosled u kom su čvorovi liste posećeni tokom pretrage.



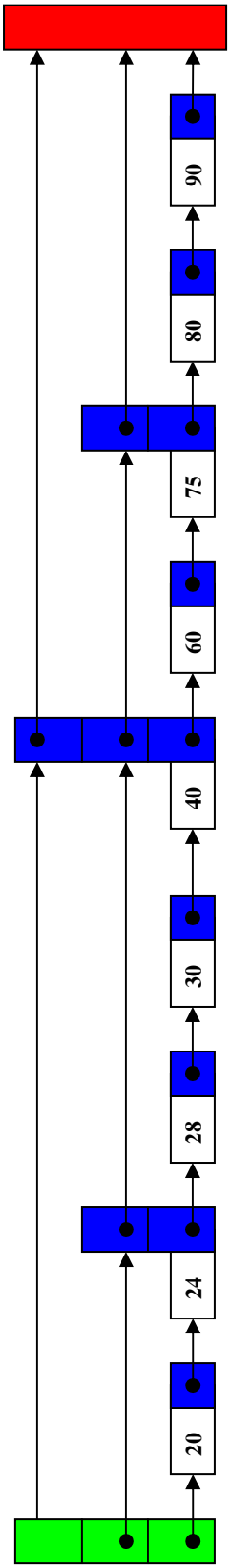
(a) glava, 38, glava, 9, 18, 38, 18, 26, 18, 23 (uspeh)

(b) glava, 38, rep, 38, rep, 38, rep, 38, 41, rep (neuspeh)

5. Data je 3-nivo regularna skip lista. Ponovo iscrtajte skip listu i demonstrirajte promene koje će nastati u strukturi kada se listi doda novi čvor sa vrednošću 28. Rezultat iscrtajte što preglednije.



Rešenje:



Prikazano rešenje je jednostavan slučaj kad je dozvoljeno da lista postane neregularna. Ako lista mora ostati regularna, onda se mora povećati visina strukture za jedan nivo, jer je trenutno popunjena (pre umetanja). Nakon toga sledi tekuća operacija umetanja.