***1. Implementacija gomile pomocu niza***

Bilo koje binarno stablo se može implementirati pomoću niza na opisani način, ako ostavimo praznine za one elemente za koje ne postoje odgovarajući čvorovi u stablu. Ovakav način implementacije binarnog stabla je neefikasan, jer će u nizu ostati određeni broj praznih mjesta neiskorišten. Međutim, postoji vrsta binarnog stabla koja ima upravo takva svojstva, koja ga čine izuzetno pogodnim za implementaciju pomoću nizova. Takva vrsta binarnog stabla se naziva gomila (eng. heap). Gomila je binarno stablo koje ima sljedeća svojstva:

1. Svi listovi su na dva susjedna nivoa

2. Svi listovi na najnižem nivou su u lijevom dijelu stabla.

3. Svi listovi iznad najnižeg nivoa su popunjeni

4. Oboje djece bilo kojeg čvora su ponovno gomile.

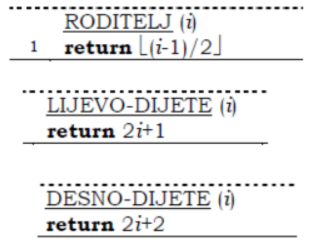
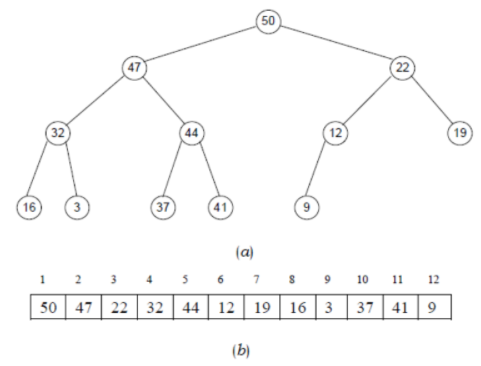
5. Vrijednost pohranjena u bilo kojem čvoru je barem toliko velika kao i vrijednosti pohranjene u čvorovima koji predstavljaju djecu dotičnog čvora.

Stablo na slici a ima najveći element (50) smješten u korijenu, a da su elementi smješteni u svakom od čvorova veći od elemenata smještenih u djeci dotičnog čvora. Listovi gomile se nalaze na susjednim nivoima (treći i četvrti). Čvorovi na najnižem nivou popunjavaju taj nivo s lijeve strane. Ako bismo uklonili npr. čvor 9, stablo bi opet zadržalo svojstva gomile. S druge strane, ako bismo uklonili bilo koji drugi čvor sa zadnjeg nivoa, bilo bi narušeno svojstvo gomile označeno kao 2

Gomila prikazana na slici a se može implementirati pomoću niza A, koji je prikazan na slici b. Primijetite da redoslijed elemenata u nizu ima određeni stepen uređenosti po opadajućem poretku. Niz A koji predstavlja gomilu je objekat koji ima dva atributa:

• duzina[A] predstavlja broj elemenata u nizu;

• velicina[A] predstavlja broj elemenata u nizu koji čine gomilu



Mada niz A[0..duzina[A]-1] fizički može sadržavati određene vrijednosti, elementi koji se nalaze iza A[velicina[A]] u logičkom smislu ne pripadaju gomili. Na primjer ako atributi gomile A imaju sljedeće vrijednosti: duzina[A]=12 i velicina[A]=5 – to znači da gomila A sadrži sljedeće elemente: A[0..velicina[A]-1] = {50, 47, 22, 32, 44}, bez obzira što su u nizu prisutni i drugi elementi na pozicijama većim ili jednakim od velicina[A]

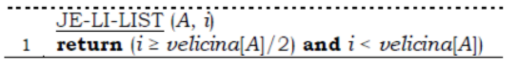
Budući da će opis tipičnih operacija u radu sa gomilom zahtijevati često pristup roditelju nekog čvora, kao i njegovoj djeci uvest ćemo tri sljedeće funkcije:

• RODITELJ (i) vraća indeks roditelja čvora i;

• LIJEVO-DIJETE (i) vraća indeks lijevog djeteta čvora i;

• DESNO-DIJETE (i) vraća indeks desnog djeteta čvora i

Nadalje, za utvrđivanje da li je neki čvor list, koristit ćemo također vrlo jednostavnu funkciju JE-LI-LIST. Funkcija vraća true ako je čvor s indeksom i list, odnosno vrijednost false ako dotični čvor nije list.



Dakako, na sličan način kako je implementirana gomila sa maksimalnim elementom u korijenu, može se na analogan način realizirati i gomila sa minimalnim elementom u korijenu. Prema tome, postoje dvije vrste gomile:

• gomile s maksimalnim elementom u korijenu;

• gomile s minimalnim elementom u korijenu.

Kod obje vrste vrijednosti u čvorovima zadovoljavaju tzv. svojstvo poretka, koje je zapravo već prethodno navedeno kao peto, pri nabrajanju i ostalih svojstava gomile. To svojstvo kod gomile s maksimalnim elementom u korijenu je:

• A[RODITELJ(i)] => A[i].

Dakle, svojstvo poretka definira da su vrijednosti u roditeljskim čvorovima veće ili jednake od vrijednosti u čvorovima koji su njihova djeca. Prema tome, najveća vrijednost je pohranjena u korijenu.

Gomila s minimalnim elementom u korijenu je organizirana na suprotan način. Svojstvo poretka takve gomile je:

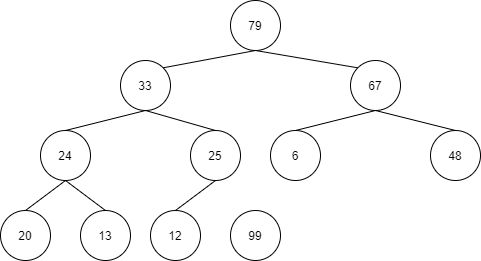
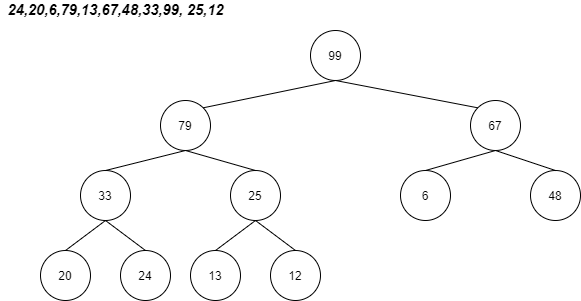
• A[RODITELJ(i)] <= A[i]

U daljem izlaganju gomilu ćemo podrazumijevati kao niz max. Elemenata bez gubitka općenitosti

***2. Sortirajte sljedeci niz 24,20,6,79,13,67,48,33,99,33,25,12 koristeći Quick Sort i Heap sort***

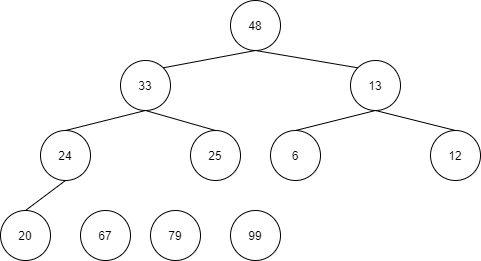
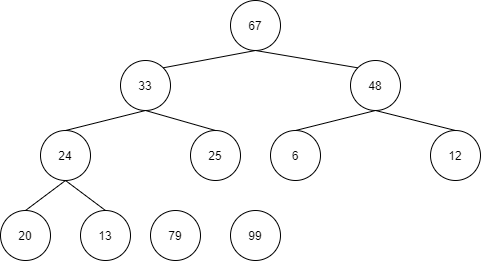
Quick je već objašnjen na jednom primjeru.

Za heap je prvo potrebno pozvati funkciju napravi gomilu i proslijediti naš niz. Nakon toga imamo gomilu u obliku: (izbacila sam dupli 33 iz niza)

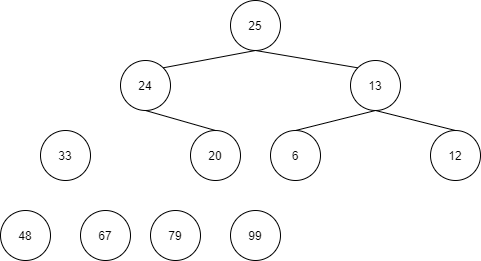
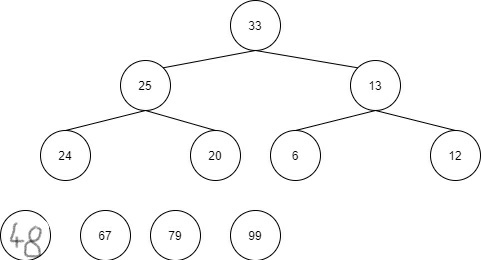


Kod sortiranja metodom heap, izbacujemo element u korijenu i zatim popravljamo gomilu funkcijom popravi-dolje

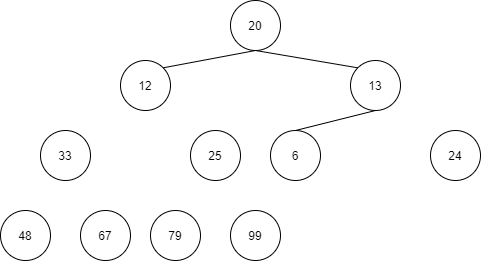
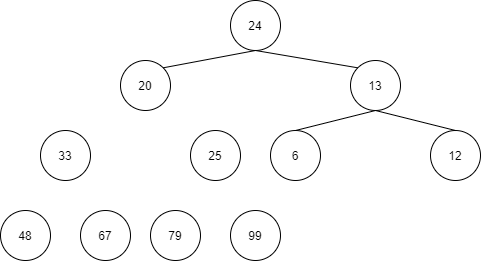
Dakle sada imamo 99 na zadnjoj poziciji niza i novu gomilu, tako ćemo nastavljati dok ne izbacimo sve elemente iz gomile(slike ispod su novnonastale gomile a niz ispod je trenutni sortirani niz koji imamo):



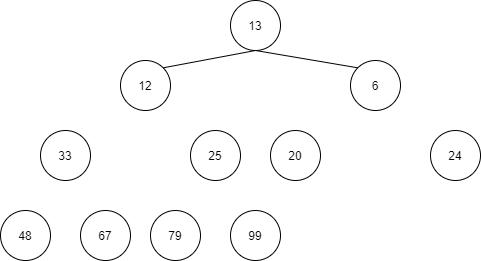
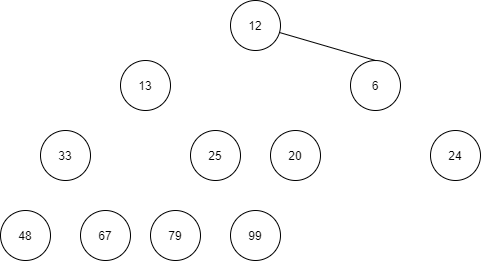
79, 99 67,79, 99



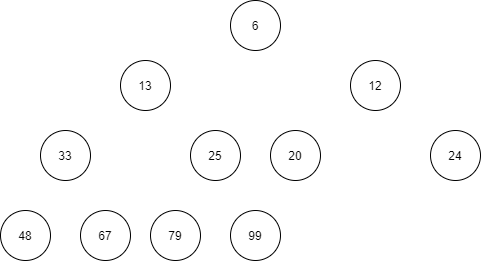
48, 67, 79, 99 33, 48, 67, 79, 99



25, 33, 48, 67, 79, 99 24, 25, 33, 48, 67, 79, 99

20, 24, 25, 33, 48, 67, 79, 99 13, 20, 24, 25, 33, 48, 67, 79, 99



12, 13, 20, 24, 25, 33, 48, 67, 79, 99 i na kraju zadnji element koji je ostao je 6 pa imamo finalno 6, 12, 13, 20, 24, 25, 33, 48, 67, 79, 99

***3. Objasniti operaciju umetanja u AVL stablo (navesti primjer)***

Počinjemo sa umetanjem čvora 20 u korijen (slika a). Budući da ovaj čvor nema djecu, njegov balans je 0, te stablo koje trenutno ima samo jedan čvor zadovoljava uvjete AVL stabla. Sada dodajemo sljedeći čvor 30, što je prikazano na slici b. Čvor 30 ima balans 0, a čvor 20 sada ima balans -1.

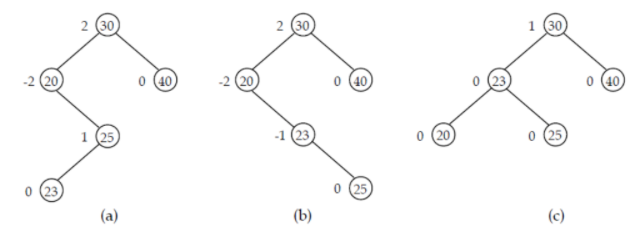
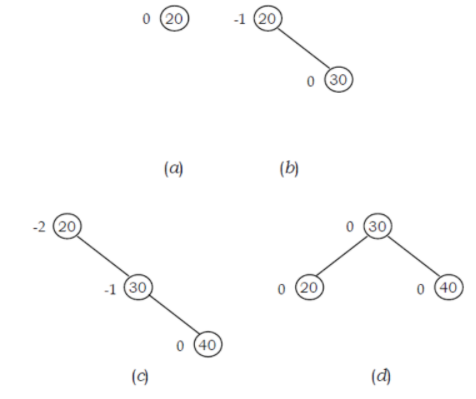
Sljedeći čvor kojeg dodajemo je čvor 40 (slika c). Počevši od tog čvora, izračunavamo balans čvorova na putu prema korijenu. Pošto je čvor 40 terminalni čvor, njegov balans je 0. Čvor 30 ima balans -1, što ne narušava uvjete AVL stabla. Međutim, čvor 20, koji ima balans -2, je čvor koji narušava uvjete AVL stabla, pa je potrebno preurediti stablo na način da se obnovi balansiranost stabla.

Operacija kojom se obnavlja balansiranost stabla se naziva rotacija. Proces dodavanja novih čvorova nastavljamo sa dodavanjem čvorova 25 i 23. Nakon dodavanja čvora 25, nije došlo do narušavanja uvjeta AVL stabla. Međutim, dodavanjem čvora 23 dolazi do narušavanja balansiranosti. Izgled stabla nakon dodavanja čvorova 25 i 23 je prikazan na slici a.

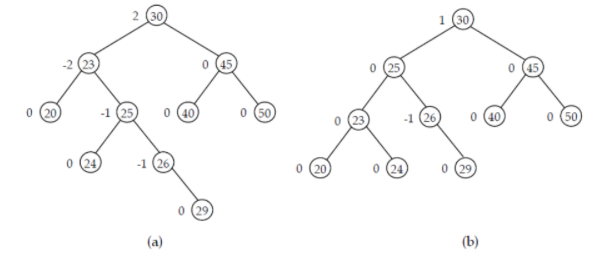
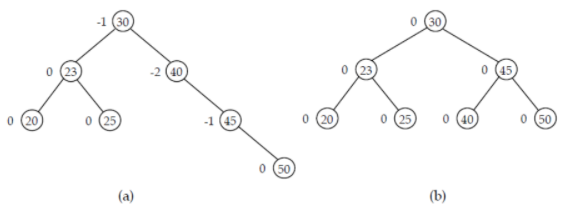
Utvrđujemo na koju stranu naginje kritični čvor (čvor 20) i njegovo desno dijete. Možemo vidjeti da čvor 20 naginje na desnu stranu, dok čvor 25 naginje na lijevu stranu. Prema tome, potrebno je primijeniti dvostruku rotaciju da bi se obnovila balansiranost stabla. Dvostruka rotacija se sastoji od dvije jednostruke rotacije u suprotnim smjerovima. Prva rotacija se obavlja na dva nivoa ispod kritičnog čvora. U našem primjeru ta rotacija uključuje čvorove 25 i 23. Čvor 23 se rotira tako da zamjenjuje čvor 25, a čvor 25 postaje desno dijete čvora 23. Nakon ove prve rotacije stablo ima izgled kao na slici b.

Na toj slici možemo vidjeti da prva rotacija nije obnovila balansiranost stabla, pa je potrebno primijeniti još jednu rotaciju, koja uključuje čvorove: 25, 23 i 20. Čvor 23 se rotira tako da zamjenuje čvor 20, dok se čvor 20 rotira na način da postaje lijevo dijete čvora 23. Rezultirajuće stablo, nakon ove druge rotacije, je prikazano na slici c. Možemo vidjeti da je kod rezultirajućeg stabla obnovljena balansiranost, jer nijedan od čvorova u stablu nema apsolutnu vrijednost balansa veću od 1

Proces dodavanja novih čvorova u stablo nastavljamo sa dodavanjem čvorova 45 i 50. Dodavanje čvora 45 neće narušiti balansiranost stabla, za razliku od dodavanja čvora 50. Rezultirajuće stablo nakon dodavanja ova dva čvora je prikazano na slici a. Prvi kritični čvor koji narušava balansiranost na putu umetanja novog čvora je čvor 40.



Ponovno utvrđujemo na koju stranu naginju kritični čvor i njegovo desno dijete, te zaključujemo da oba čvora naginju na desnu stranu. Prema tome, za obnavljanje balansiranosti stabla je potrebno napraviti jednostruku rotaciju. Rezultirajuće balansirano stablo nakon rotacije je prikazano na slici b



Dodajmo čvorove 26, 24 i 29. Dodavanje čvorova 26 i 24 neće narušiti balansiranost stabla, za razliku od dodavanja čvora 29. Rezultirajuće stablo nakon dodavanja ovih čvorova je prikazano na slici a. Počevši od novog čvora 29, vraćamo se nazad putem umetanja tog čvora i nailazimo na čvor 23, kao prvi kritični čvor koji narušava balansiranost stabla. Opet razmatramo na koju stranu naginju kritični čvor i njegovo desno dijete

Počevši od novog čvora 29, vraćamo se nazad putem umetanja tog čvora i nailazimo na čvor 23, kao prvi kritični čvor koji narušava balansiranost stabla. Opet razmatramo na koju stranu naginju kritični čvor i njegovo desno dijete. Vidimo da oba čvora naginju na desnu stranu, te zaključujemo da je za obnavljanje balansiranosti stabla potrebno napraviti jednostruku lijevu rotaciju. Čvor 25 se rotira tako da zamjenjuje čvor 23, dok se čvor 23 rotira tako da postaje lijevo dijete čvora 25. Nadalje, bitno je uočiti kako se pozicionira i čvor 24, koji je prethodno bio lijevo dijete čvora 25. Sada čvor 24 postaje novo desno dijete čvora 23. Primjenom prethodno opisanih izmjena, dobija se balansirano stablo kao na slici b

***4. Opisati Algoritam SELECTION-SORT***

Kod ovog algoritma se pomjeranje podataka pri sortiranju obavlja direktno na njegovu konačnu poziciju u sortiranom nizu.

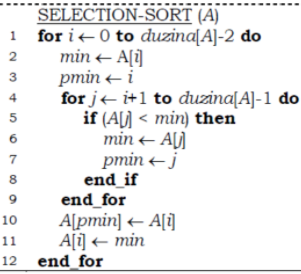
Slično kao i kod algoritma Insertion sort, i kod algoritma Selection sort se u procesu sortiranja razlikuju sortirani i nesortirani dio niza.

Na početku svi elementi niza A[0..n-1]={a0, a1, ..., ai, ai+1,...an-1} pripadaju nesortiranom dijelu, a zatim se sekvencijalnim pretraživanjem pronalazi najmanji element, koji se zatim premješta na prvu poziciju, dok se prethodni element s prve pozicije premješta na poziciju pronađenog minimalnog elementa.

Na taj način je u prvom koraku oblikovan sortirani dio niza koji čini element a1, dok ostali elementi niza (a2, a3, ..., an-1) čine nesortirani dio niza. Poslije nekog (i-1)-og koraka u sortiranom dijelu niza su elementi: a0, a1, ..., ai-2; dok su u nesortiranom dijelu niza elementi: ai-1, ai,..., an-1.

U sljedećem i-tom koraku se u nesortiranom dijelu niza pronalazi najmanji element, te se taj element pozicionira na prvo mjesto u neuređenom dijelu niza, a to je pozicija i-1.

U tom trenutku sortirani dio niza je: a0, a1, ..., ai-1; dok je nesortirani dio niza: ai, ai+1, ..., an-1. Postupak sortiranja se završava kada u nesortiranom dijelu na poziciji n-1 preostane samo jedan element, koji zapravo ostaje na toj poziciji.



***5. Pretpostavimo da koristimo heš tabelu sa 13 mjesta pri cemu su mjesta indeksirana od 0 do 12. Koristeci dvostruko hesiranje sa otvorenim adresiranjem za rjesavanje kolizija unesite sljedece kljuceve: 1, 6, 30, 19, 18, 52 i 26***

***Hes funkcije su: h1(k) = k mod 13 h2(k) = obrni(k+1) mod 11***

U tabelu ispod ćemo upisivati vrijednosti, a procedura je:

h(k) = (h1(k) + i\*h2(k)) mod 13

element 1: h1(1) = 1, nema kolizije upisujemo na lokaciju 1

element 6: h1(6) = 6, nema kolizije upisujemo na lokaciju 6

element 30: h1(30) = 4, nema kolizije upisujemo na lokaciju 4

element 19: h1(19) = 6, kolizija postoji pa racunamo

h(19) = (h1(19) + 1\*h2(19)) mod 13 -> h(19) = 6 + 1\*2 = 8 mod 13 pa upisujemo na lokaciju 8

element 18: h1(18) = 5, nema kolizije upisujemo na lokaciju 5

element 52: h1(52) = 0, nema kolizije upisujemo na lokaciju 0

element 26: h1(26) = 0, kolizija postoji pa racunamo

h(26) = (h1(26) + 1\*h2(26)) mod 13 -> h(26) = 0 + 1\*6 = 6 mod 13, kako ponovo postoji kolizija racunamo dalje: h(26) = 0 + 2\*6 = 12 mod 13 pa upisujemo na lokaciju 12

|  |  |
| --- | --- |
| 0 | 52 |
| 1 | 1 |
| 2 |  |
| 3 |  |
| 4 | 30 |
| 5 | 18 |
| 6 | 6 |
| 7 |  |
| 8 | 19 |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 | 26 |

***6. Za Quicksort metodu napravite sljedece:***

***a) Opisati algoritam detaljno***

***b) Opisati svaki korak za niz brojeva 10, 7, 4, 15, 13, 6, 28, 20, 5 I 11***

Algoritam **Quick sort** također koristi strategiju “podijeli, pa vladaj”.

Kod njega se ulazni niz dijeli na dva dijela, tako što se taj niz reorganizira na način da su svi elementi u prvom donjem dijelu manji od svih elemenata u drugom gornjem dijelu niza.

Ta dva dijela nakon reorganizacije su nesortirana i razdvojena su elementom koji se naziva pivot i koji se nalazi na nekoj poziciji j.

Donji dio čine elementi koji imaju vrijednost manju ili jednaku vrijednosti pivota (A[i] ≤ pivot, i=0, 1,..., j-1), dok gornji dio čine elementi koji imaju vrijednost veću ili jednaku vrijednosti pivota (A[i] ≥ pivot, i=j+1, j+2,..., n-1). **Mogući su različiti načini izbora pivota**.

Izbor pivota možemo napraviti tako da uzmemo prvi element niza koji se dijeli

**Idealno**, pivot element bi trebao dijeliti niz na dva podniza sa jednakim brojem elemenata.

Zbog načina na koji je ulazni niz reorganiziran, možemo zaključiti da je pivot element pozicioniran na svoju konačnu poziciju j, te se u daljnjem postupku više neće pomjerati.

Možemo zaključiti da se sortiranje može nastaviti tako da isti algoritam rekurzivno poziva

Rekurzivni postupak se nastavlja, sve dok rezultirajući dijelovi ne budu imali samo jedan element. Završetkom rekurzivnog postupka se dobiva sortirani niz Funkcija **PARTICIJA** vraća indeks pozicije na kojoj je smješten pivot.

U primjeru na slici za pivot se uzima element 42 (element je označen tankom donjom crtom).

Funkcijom PARTICIJA je potrebno reorganizirati niz, tako da se pronađe odgovarajuća pozicija za pivota, te da svi elementi manji od pivota budu na pozicijama ispred njega, a svi veći elementi da budu na pozicijama iza pivota.

Na slici oznaka p označava poziciju elemenata koji se trenutno uspoređuju s pivotom, do trenutka dok se ne pronađe prvi element koji je veći od pivota.

Kada se pronađe prvi element veći od pivota, za označavanje pozicija elemenata koji se trenutno uspoređuju sa pivotom koristi se index i.

Te pozicije su na slici istaknute donjom podebljanom crtom.

b) ***10, 7, 4, 15, 13, 6, 28, 20, 5 I 11*** // *drugi brojevi ali ideja ista*

na pocetku imamo ***10, 7, 4, 15, 13, 6, 28, 20, 5, 11***

Pivot je broj **10**, provjeravamo da li su elementi manji od njega, za **7, 4,**  to vrijedi

nailazimo na **15** za koji to nije slučaj i njeg označavamo pa sada imamo

***10, 7, 4, 15, 13, 6, 28, 20, 5, 11***

**13** je ponovo veći, pa prelazimo na **6** koji je manji i vršimo rotaciju a oznaku pomjeramo na **13**

***10, 7, 4, 6, 13, 15, 28, 20, 5, 11,***  *naš trenutni p=4*

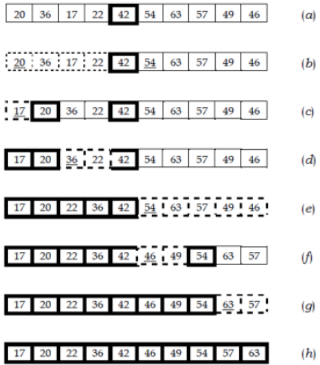
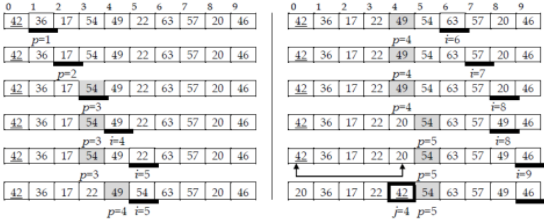
***28*** je veći od **10** pa idemo dalje sve do broja **5,** te sada mijenjamo mjesta i imamo

***10, 7, 4, 6, 5, 15, 28, 20, 13, 11,***  *naš trenutni p=5*

sada mijenajmo mjesta **6 i 10** i konačno imamo

***5, 7, 4, 6, 10, 15, 28, 20, 13, 11***

sada imamo prvu particiju i dalje nastavljamo raditi isti postupak na preostalim elementima

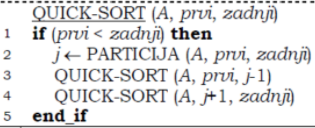
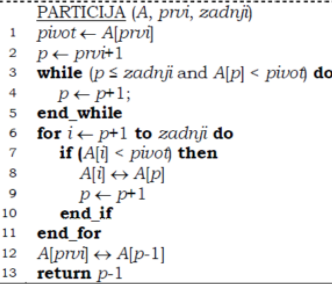


Sada lijevi dio <10 posmatramo kao particiju pa radimo isti algoritam gdje je sada pivot broj **5** pa imamo(siva pozadina je finlna pozicija(analogno punoj liniji na slici iznad, a plava je analogna isprekidanoj)

***5, 7, 4, 6, 10, 15, 28, 20, 13, 11 -> 4, 5, 7, 6, 10, 15, 28, 20, 13, 11 -> 4, 5, 7, 6, 10, 15, 28, 20, 13, 11***

***-> 4, 5, 7, 6, 10, 15, 28, 20, 13, 11 -> sada je pivot 15 pa imamo***

***4, 5, 7, 6, 10, 13, 11, 15, 28, 20 -> 4, 5, 7, 6, 1 0, 11, 13, 15, 28, 20 - > 4, 5, 7, 6, 10, 11, 13, 15, 20, 28***



***7. Objasnite detaljno Kruskalov algoritam***

Kod Kruskalovog algoritma, skup A je šuma. Sigurna grana koja se dodaje u A je uvijek grana sa najmanjom težinom u grafu, koja povezuje dvije odvojene komponente.

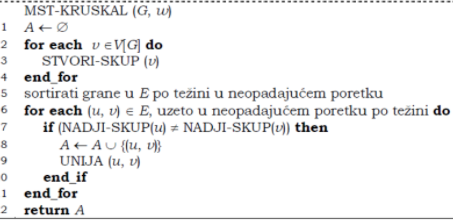
Kod **Primovog** algoritma skup A formira jedno rastuće stablo. Sigurna grana, koja se dodaje u A, je uvijek grana sa najmanjom težinom u grafu, koja povezuje rastuće stablo sa čvorom koji u datom trenutku još nije u stablu.

**Kruskalov** algoritam pronalazi sigurnu granu za dodavanje u rastuću šumu, tako što među svim granama koje povezuju bilo koja dva stabla u šumi, pronalazi granu (u, v) sa najmanjom težinom.

Ovaj algoritam polazi od šume (T1,...Tn), koja ne sadrži niti jednu granu, a sadrži n nepovezanih čvorova grafa.

Algoritmom se dodaje jedna po jedna grana, sve dok se šuma ne pretvori u minimalno obuhvatno stablo grafa G.

Kruskalov algoritam je opisan funkcijom **MST-KRUSKAL**.



***8. Koji je osnovni problem binarnog pretrazivanja I na koji nacin se rjesava ovaj problem.***

Sa aspekta efikasnosti izvođenja pojedinih operacija različite topologije stabala imaju i različitu efikasnost.

Poželjno je da za dati broj čvorova stablo ima što manju visinu sa što više grananja. Međutim, u situacijama kada stablo ima veliku visinu i malo grananje, raspored čvorova u binarnom stablu je nepovoljan.Npr, ako bi se u stablo dodavali čvorovi sa ključevima u opadajućem poretku, tada bi svi čvorovi imali samo lijevu granu, kao što je prikazano stablom na slici a), koje nastaje kao rezultat umetanja ključeva sljedećim redoslijedom: 39, 34, 29, ..., 7, 3. To se rješava balansiranjem stabla npr po visini AVL.

***a) Detaljno opistai rjesenje na bazi LL,RR,LR, I RL rotacije tri cvora***

Postoje **dva tipa rotacije** kod AVL stabala: **jednostruka i dvostruka rotacija**.

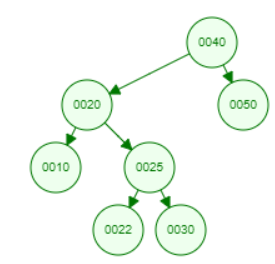
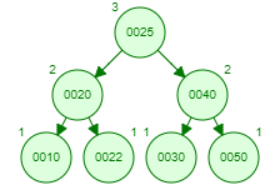
Da bismo odlučili koju rotaciju treba primijeniti koriste se sljedeća pravila:

Krećemo od novog dodanog čvora u stablo, slijedeći put umetanja prema korijenu I provjeravamo balans čvorova na tom putu. Kada pronađemo kritični čvor kod kojeg je narušen balans, potrebno je utvrditi da li kritični čvor i njegovo dijete na putu umetanja naginju na istu stranu.

Ako kritični čvor i njegovo dijete na putu umetanja naginju na istu stranu, za obnavljanje balansiranosti stabla je potrebno primijeniti **jednostruku** rotaciju. **Lijeva rotacija** se primjenjuje ako čvorovi naginju na desnu stranu , dok se **desna rotacija** primjenjuje ako čvorovi naginju na lijevu stranu.

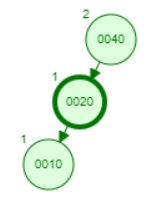
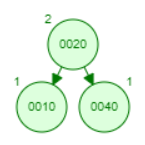
Ako kritični čvor i njegovo dijete na putu umetanja naginju na različite strane, za obnavljanje balansiranosti je potrebno primijeniti **dvostruku rotaciju**

***b) Zatim na primjeru niza 40, 20, 10, 25, 30, 22, 50 demonstrirati na koji nacin “radi” binarno stablo pretrazivanja, a na koji nacin “radi” AVL stablo za dati niz.***

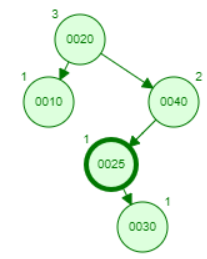
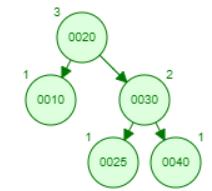
 

binarno vs AVL

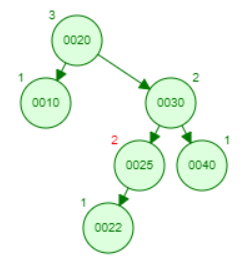
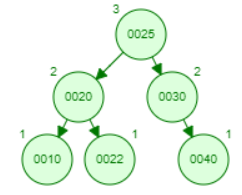
U slučaju binarnog stabla se elementi dodaju kako dolaze i zbog toga imamo “nebalansiranost”, koristeci avl stablo, prilikom unosenja elemenata odmah popravljamo balans. unosimo 40, 20 sve je balansirano. Nakon unosenja 10 stablo nije balansirano i ima izgled:

 nakon balansiranja -> 

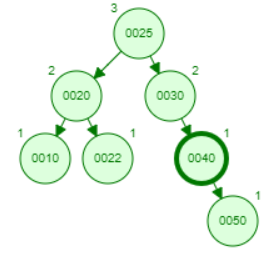
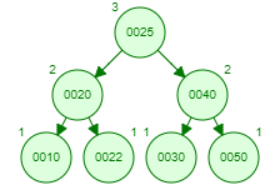
25 se unosi bez problema, nakon 30 izgleda ovako

 nakon balansiranja -> 

zatim unosimo 22

a nakoon avl -> 

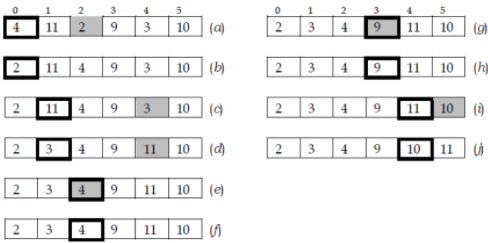
50 ponovo narusava balans

 zadnji oblik -> 

***9. S obzirom na binarno stablo definisano nizom 24, 18, 17, 12, 9, 15, 24, 3 opisati korake umetanja cvora 20 u binarno stablo na nacin da isto nakon proceduire umetanja u gomilu izgleda kao gomila***

***10. Sortiranje niz 7, 2, 1 , 6, 8, 5, 3, 4 koristeci Selection Sort I Merge Sort***

Selection sort:(slika je primjer s drugim brojevima, ispod je rjesenja za konkretni primjer iz zadatka 10)



Na početku imamo ***7, 2, 1 , 6, 8, 5, 3, 4;***

U prvom koraku uzimamo element **7** i trazimo manji od njeg ***7, 2, 1 , 6, 8, 5, 3, 4;*** Kada naiđemo na 2 utvrdimo da je on manji pa je sada nas minimum 2 ali daljnjom pretragom nailazimo na . Do kraja niza ne nailazimo na broj manji od 1 pa njeg stavljamo na 0.poziciju i imamo: ***1, 2, 7 , 6, 8, 5, 3, 4***

Ponavljanjem istog postupka imamo ***1, 2, 7 , 6, 8, 5, 3, 4*** utvrdimo da je 2 na pravoj poziciji

Dalje trazimo element za poziciju broj 2 pa imamo

***1, 2, 7, 6, 8, 5, 3, 4 ->*** *sljedeći manji su 6, pa od njeg manji 5, i na kraju 3 od kojeg nema manjeg pa imamo*  ***1, 2, 3, 6, 8, 5, 7, 4***

Sljedeća je pozicija 3

***1, 2, 3, 6, 8, 5, 7, 4 ->****sljedeći manji su 5 i na kraju 4 od kojeg nema manjeg pa imamo*  ***-> 1, 2, 3, 4, 8, 5, 7, 6***

Dalje uzimamo broj 8 na poziciji 4 kao min pa imamo:

***1, 2, 3, 4, 8, 5, 7, 6 ->*** *prvi manji je 5 i dalje se ne nalazi manji pa mijenjamo mjesta 8 i 5* ***-> 1, 2, 3, 4, 5, 8, 7, 6***

Sljedeci će ponovo biti broj 8

***1, 2, 3, 4, 5, 8, 7, 6 ->*** *prvi manji je 7 i dalje se nalazi 6, mijanjamo mjesta 8 i 6* ***-> 1, 2, 3, 4, 5, 6, 7, 8***

Kod odabira broja 6 ne nailazimo na manji, isto se desava i za slucaj broja 7. Zadnji broj (u nasem slucaju 8) se svakako ne provjerava jer on svakako mora biti na svojoj poziciji

Merge:

Zadati niz dijelimo dok ne dobijemo po jedan element:

***7, 2, 1 , 6, 8, 5, 3, 4***

***7, 2, 1 , 6 8, 5, 3, 4***

***7, 2 1 , 6 8, 5 3, 4***

***7 2 1 6 8 5 3 4***

Sada kada imamo podijeljen niz idemo po jedan koraj unazad i sastavljamo nizove tako da su uvijek sortirani

U 1. koraku imamo (svjetliji nizovi su oni koje u tom koraku spajamo)

***7, 2, 1 , 6, 8, 5, 3, 4***

***7, 2, 1 , 6 8, 5, 3, 4***

***2, 7 1 , 6 5, 8 3, 4***

***7 2 1 6 8 5 3 4***

U 2. koraku imamo

***7, 2, 1 , 6, 8, 5, 3, 4***

***1, 2, 6, 7 3, 4, 5, 8***

***2, 7 1 , 6 5, 8 3, 4***

U 3. koraku imamo

***1, 2, 3, 4, 5, 6, 7, 8***

***1, 2, 6, 7 3, 4, 5, 8***

Kako dolazi do spajanja dva niza ćemo objasniti na 3. koraku.

Prvo posmatramo elemente na pozicijama 0 u oba podniza i uzimamo manji tj 1 pa imamo

***1***

***/, 2, 6, 7 3, 4, 5, 8***

Sada posmatramo element na poziciji 1 u prvom podnizu(jer smo uzeli 0. element) i 0.element u drugom(desnom) podnizu, vidimo da je 2 < 3

***1, 2***

***/, /, 6, 7 3, 4, 5, 8***

Sada posmatramo element na poziciji 2 u prvom podnizu(jer smo uzeli 1. element) i 0.element u drugom(desnom) podnizu, vidimo da je 3 < 6

***1, 2, 3***

***/, /, 6, 7 /, 4, 5, 8***

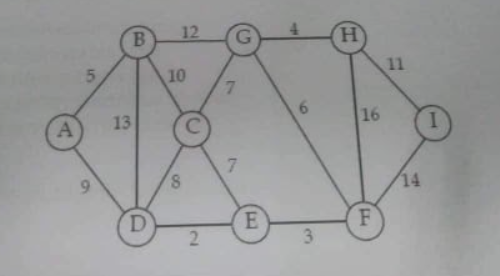
Sada posmatramo element na poziciji 2 u prvom podnizu i 1.element u drugom podnizu, vidimo da je 4 < 6

***1, 2, 3, 4***

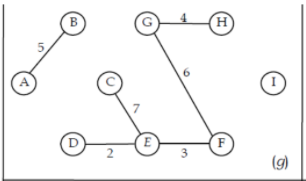
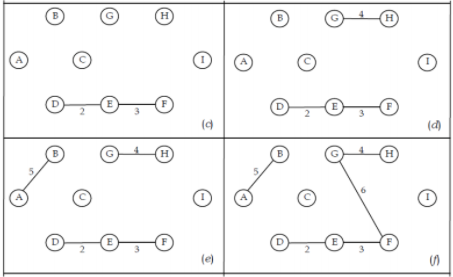
***/, /, 6, 7 /, /, 5, 8***

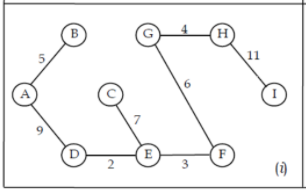
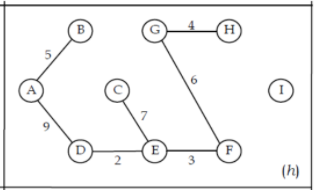
itd analogno

***11. Navedite redoslijed kojim ce se u minimalno obuhvatno stablo dodavati grane grafa izvodjenem Primovog algoritma***

******

Najmanja grana je D,E=2. Zatim proces rasta obuhvatnog stabla nastavljamo dodavanjem grana (E, F), (G, H), (A, B) i (F, G) - a ti koraci su ilustrirani na slikama c-f.





U sljedećem koraku utvrđujemo da postoje dvije grane u grafu G sa najmanjom težinskom vrijednošću 7, a to su grane (C, G) i (C, E).

Dodavanjem bilo koje od navedene dvije grane neće biti formiran ciklus, jer se kod obe grane krajnji čvorovi nalaze u različitim skupovima. Prema tome, možemo izabrati bilo koju od te dvije grane i dodati je u skup A.

U ovom primjeru je izabrana grana (C,E), pa je trenutno stanje u procesu određivanja minimalnog obuhvatnog stabla prikazano na slici g.

Sljedeća grana koju je potrebno razmotriti je grana (C, D), jer ima najmanju težinsku vrijednost.

Utvrđujemo da su čvorovi C i D u istom skupu, što znači da bi se dodavanjem grane (C, D) kreirao ciklus, pa tu granu izostavljamo i dodajemo granu sa narednim najmanjim težinskim koeficijentom, a to je grana (A, D).

Trenutno stanje procesa formiranja obuhvatnog stabla je prikazano na slici h.

Sljedeća grana sa najmanjim težinskim koeficijentom je grana (B,C). Ponovno utvrđujemo da su čvorovi B i C u istom skupu, što znači da bi se dodavanjem i grane (B, C) kreirao ciklus, pa i ovu granu izostavljamo i dodajemo granu sa narednim najmanjim težinskim koeficijentom, a to je grana (H, I).

U ovom trenutku imamo samo jedan skup povezanih čvorova {A, B, C, D, E, F, G, H, I}, pa zaključujemo da su svi čvorovi povezani i da je proces formiranja obuhvatnog stabla završen.

***12. Opisati rekurzivnu procedure za odredivanje permutacija skupa od n elemenata (mozete uzeti primjer skupa A[0..3] = {1,2,3,4})***

***13. Objasniti procedure popravka gomile (pseudokod POPRAVI-DOLJE)***

Za proces izgradnje gomile jedna od ključnih procedura predstavlja postupak spuštanja nekog izabranog čvora u gomili, s ciljem pronalaženja njegove prihvatljive pozicije da bi se održalo potrebno svojstvo poretka. Taj postupak je opisan procedurom **POPRAVI-DOLJE**

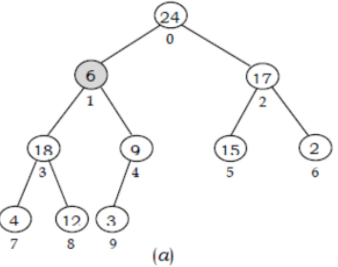
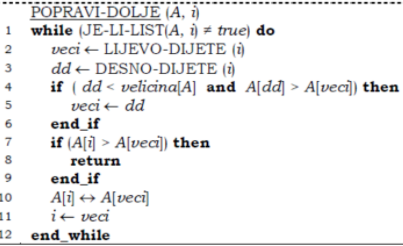
Ulaz u proceduru su niz A i indeks i. Prije poziva procedure pretpostavljamo da su lijevo i desno podstablo čvora i gomile, a nakon izvođenja procedure podstablo koje ima korijen smješten na poziciji s indeksom i također postaje gomila.

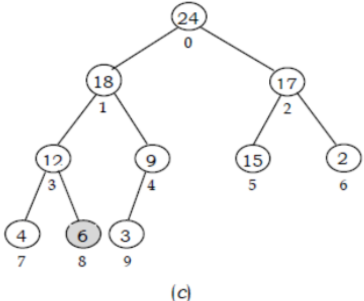
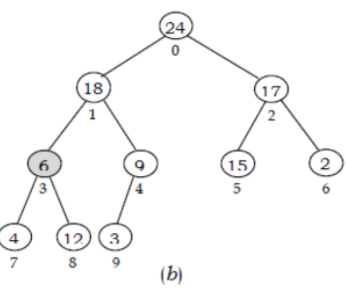
Postupak spuštanja čvora se izvršava u petlji while, tako da u pojedinim iteracijama varijabla veci poprima vrijednost indeksa većeg djeteta čvora i.

Ako je element A[i] veći od svoje djece, što se utvrđuje njegovim poređenjem sa većim djetetom A[veci], tada se procedura prekida, jer ostatak stabla sigurno ima svojstvo poretka (linije 7-9).

U suprotnom, elementi niza A sa indeksima i i veci izmjenjuju pozicije, te indeks i poprima vrijednost većeg djeteta veci (linije 10-11).

Opisani postupak se nastavlja dalje prema listovima stabla





Na slici je ilustriran način rada procedure POPRAVI-DOLJE, za primjer gomile prikazane pomoću niza A veličine 10.

Na slici a je prikazana početna konfiguracija gomile A, pri čemu element A[1]=6 narušava svojstvo poretka, jer taj element nije veći od svoje djece

Pozivom procedure POPRAVI-DOLJE (A, 1) se popravlja gomila, tako da u prvom koraku dolazi do izmjene pozicija elemenata A[1] i A[3] , jer je veće dijete čvora s indeksom 1 pronađeno na poziciji 3 (slika b)

Varijabla i poprima vrijednost i=3, pa se postupak nastavlja sa podstablom koje ima korijen u čvoru A[3].

Prethodnom izmjenom je narušeno svojstvo poretka za podstablo sa korijenom A[3], pa u sljedećoj iteraciji dolazi do izmjene pozicija elemenata A[3] A[8], jer je veće dijete čvora A[3] pronađeno na poziciji 8 (slika c).

Varijabla i poprima vrijednost i=8, pa u sljedećem koraku poziv procedure JE-LI-LIST(A, 8) daje vrijednost true, te se izvođenje petlje while prekida.

***14. Sortirajte sljedeci niz 24, 20, 6, 79, 13, 67, 48, 33, 99, 25, 12 koristeci Selection Sort I Bubble Sort***

Primjer već ima za selection pa ćemo objasniti bubble samo

U bubble sortu poredimo dva susdjedna elementa, ako je lijevi veci onda im mijenjamo mjesta, tako da na kraju prve provjere imamo na zadnjoj poziciji najveci broj. Kada stavimo taj broj na zadnju poziciju u sljedecem ponavljanju cemo ici do predzadnjeg elementa s poredjenjima(u primjeru ispod pozadina siva oznacava da poredimo dva elementa, pozadina plava je fiksna pozicija elementa)

***24, 20, 6, 79, 13, 67, 48, 33, 99, 25, 12 -> 20, 24, 6, 79, 13, 67, 48, 33, 99, 25, 12 ->***

***20, 6, 24, 79, 13, 67, 48, 33, 99, 25, 12 -> 20, 6, 24, 79, 13, 67, 48, 33, 99, 25, 12 ->***

***20, 6, 24, 13, 79, 67, 48, 33, 99, 25, 12 -> 20, 6, 24, 13, 67, 79, 48, 33, 99, 25, 12 ->***

***20, 6, 24, 13, 67, 48, 79, 33, 99, 25, 12 -> 20, 6, 24, 13, 67, 48, 33, 79, 99, 25, 12 ->***

***20, 6, 24, 13, 67, 48, 33, 79, 99, 25, 12 -> 20, 6, 24, 13, 67, 48, 33, 79, 25, 99, 12 ->***

***20, 6, 24, 13, 67, 48, 33, 79, 25, 12, 99***

Za ostale korake ćemo samo prikazati krajnji izgled niza:

***6, 20,13, 24, 48, 33, 67, 25, 12, 79, 99***

***6, 13, 20, 24, 33, 48, 25, 12, 67, 79, 99***

***6, 13, 20, 24, 33, 25, 12, 48, 67, 79, 99***

***6, 13, 20, 24, 25,12, 33, 48, 67, 79, 99***

***6, 13, 20, 24, 12, 25, 33, 48, 67, 79, 99***

***6, 13, 20, 12, 24, 25, 33, 48, 67, 79, 99***

***6, 13, 12, 20, 24, 25, 33, 48, 67, 79, 99***

***6, 12, 13, 20, 24, 25, 33, 48, 67, 79, 99***

***6, 12, 13, 20, 24, 25, 33, 48, 67, 79, 99***

***6, 12, 13, 20, 24, 25, 33, 48, 67, 79, 99***

***15. Neka je dat sljedeci niz brojeva: 156, 88, 488, 980, 787, 869, 567, 562, 976, 63. Gornje brojeve treba smjestiti u hes tabelu sa 10 pozicija, koristeci otvoreno adresiranje. Pretpostaviti da je hes funkcija h: h(broj) = broj mod 10.***

***Za rjesavanje kolizija treba koristiti linerano ispitivanje:***

***a) Navedite sadrzaj hes tabele nakon umetanja gore navedenih brojeva***

***b) Kolika je vjerojatnost popunjavanja svih ostalih lokacija, nakon unosa prvog broja 156?***

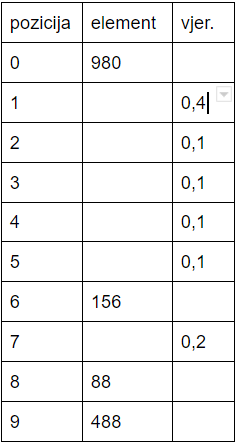
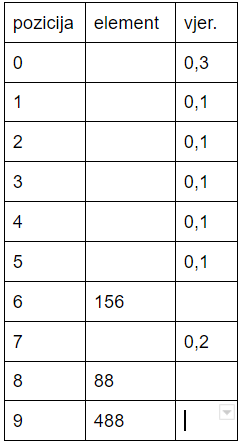
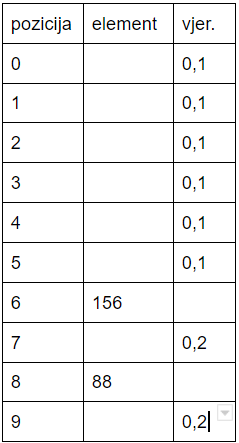
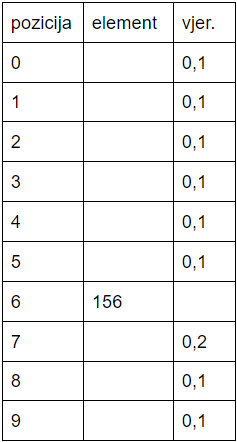
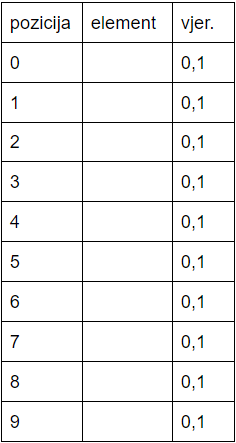
***c) Kolika je vjerojatnost popunjavanja svih preostalih slobodnih lokacija, nakon unosa prvih 5 brojeva?***

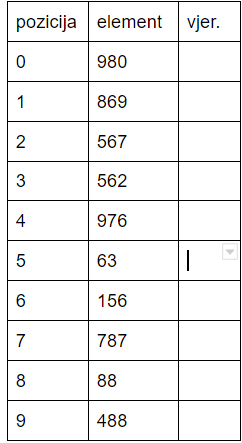
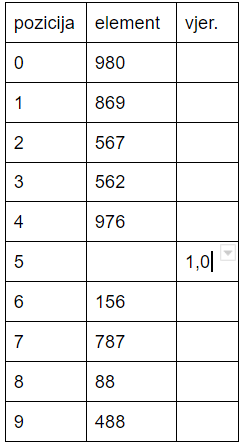
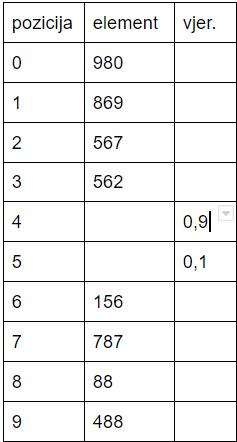
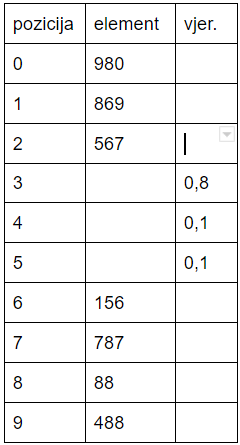
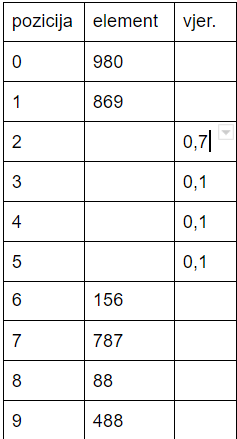
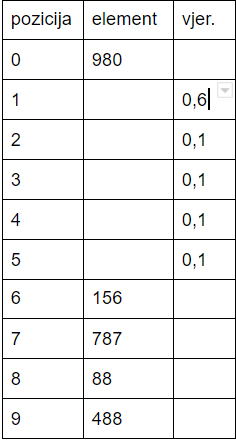
na samom pocetku tabela ima oblik prikazan na prvoj tabeli ispod. dakle vjerovatnost za sve pozicije je 0.1 jer ih ima 10.

a) Slika broj 11

b) Prvi element 156 će zauzeti poziciju 6, te će se vjerovatnost pozicije 7 povećati na 0,2.

c) Vjerovatnosti nakon unosa 5 elemenata su opisane u slici broj 6





***16. Sta je binarno stablo I koje su osobine binarnog stabla?***

**Binarno stablo** je stablo kod kojeg svaki roditelj ima najviše dvoje djece. Svako dijete u binarnom stablu je označeno ili kao **lijevo** dijete ili kao **desno** dijete,

**Puno binarno stablo** je binarno stablo kod kojeg svaki roditelj ima tačno dvoje djece.

Puno binarno stablo kod kojeg su svi listovi na istom nivou se naziva **kompletno binarno stablo.**

**Binarno stablo pretraživanja za svaki čvor n u kojem je pohranjena vrijednost ključa k ima sljedeće osobine:**

sve vrijednosti smještene u njegovom **lijevom** podstablu **su manje** od neke vrijednosti k;

sve vrijednosti pohranjene u **desnom** podstablu **su veće** od k.

***17. Sta je heap (gomila) i na koji nacin funkcionira funkcija POPRAVI-GOMILU?***

Gomila objasnjena u pitanju 1. Postoje funkcije POPRAVI-GORE I POPRAVI-DOLJE. U pitanju 13 je objasnjeno POPRAVI-DOLJE. U nastavku objačnjeno **POPRAVI-GORE**

Nakon povećanja vrijednosti nekog ključa A[i], postupak popravljanja gomile se izvodi podizanjem čvora prema korijenu i opisan je procedurom POPRAVI-GORE

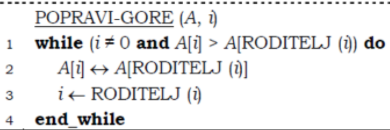
Prije poziva procedure pretpostavljamo da niz A[0..velicina[A]-1] zadovoljava svojstvo poretka, te da je došlo do povećanja ključa A[i] koje može narušiti to svojstvo.

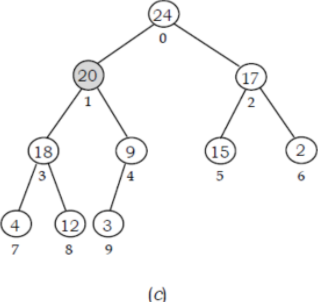
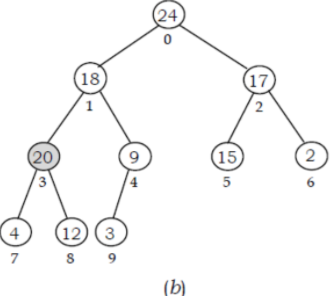
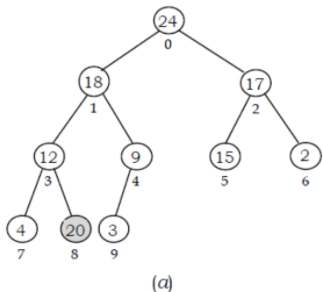
Popravak se izvodi počevši od čvora u kojem je došlo do povećanja vrijednosti ključa, do korijena gomile da bi se pronašlo ispravno mjesto za novu povećanu vrijednost.

Kretanjem kroz gomilu prema korijenu uspoređuju se elementi sa svojim roditeljima, pri čemu dolazi do izmjene njihovih pozicija, ako je vrijednost ključa nekog elementa veća od vrijednosti ključa njegovog roditelja (linija 2).

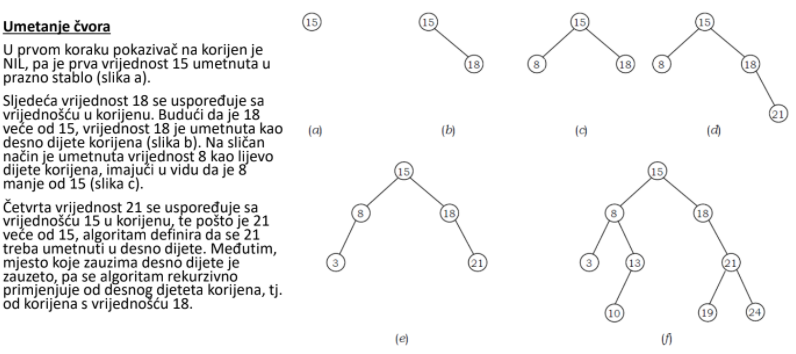
**POPRAVI-GORE O(logn)**

Na slici je ilustriran način rada procedure POPRAVI-GORE, za primjer gomile prikazane nizom A veličine 10.





***18. Objasniti procedure umetanja cvorova u binarno stablo pretrazivanja (rekurzivni pseudokod)***



Ova procedura umeće novi čvor sa adresom z u stablo sa korijenom na kojeg pokazuje pokazivač korijen

