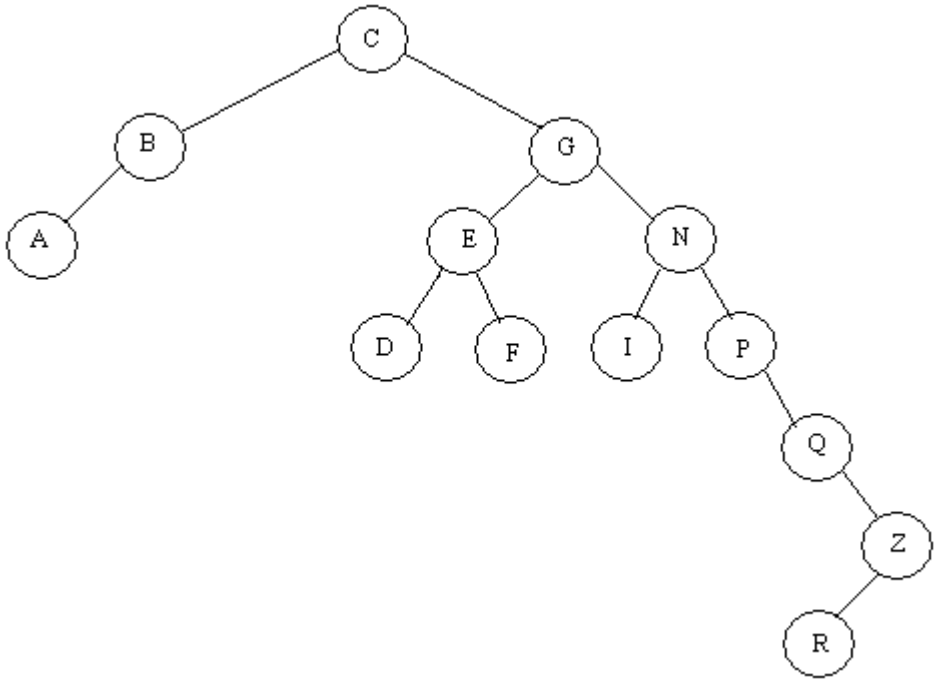
ALGORITMI I STRUKTURE PODATAKA – GRUPA 9
DODACI UZ DVANAESTI I TRINAESTI TJEDAN NASTAVE
Robert Manger, 3. lipnja 2007.

SORTIRANO BINARNO STABLO

Sortirano binarno stablo također se naziva i uređeno binarno stablo ili binarno stablo za traženje (binary search tree). Sastoji se od čvorova (s različitim vrijednostima) koji se mogu uspoređivati nekom uređajnom relacijom <=, tj za svaka dva čvora može se odrediti koji je veći a koji je manji. Za bilo koji odabrani čvor vrijedi: čvorovi u pripadnom lijevom pod-stablu su manji od odabranog, čvorovi u desnom pod-stablu su veći od odabranog.

Iduća slika prikazuje primjer sortiranog binarnog stabla, gdje se čvorovi uspoređuju po abacedi, tj. A < B < C ... itd. Vidimo da vrijedi svojstvo o rasporedu manjih čvorova u lijeva pod-stabla te većih čvorova u desna pod-stabla.



Sortirano binarno stablo predstavlja efikasan način za prikaz skupa podataka – čvorova. Naime, efikasno se mogu obavljati sljedeće (skupovne) operacije: ubacivanje čvora, izvacivanje čvora, traženje čvora.

Traženje čvora

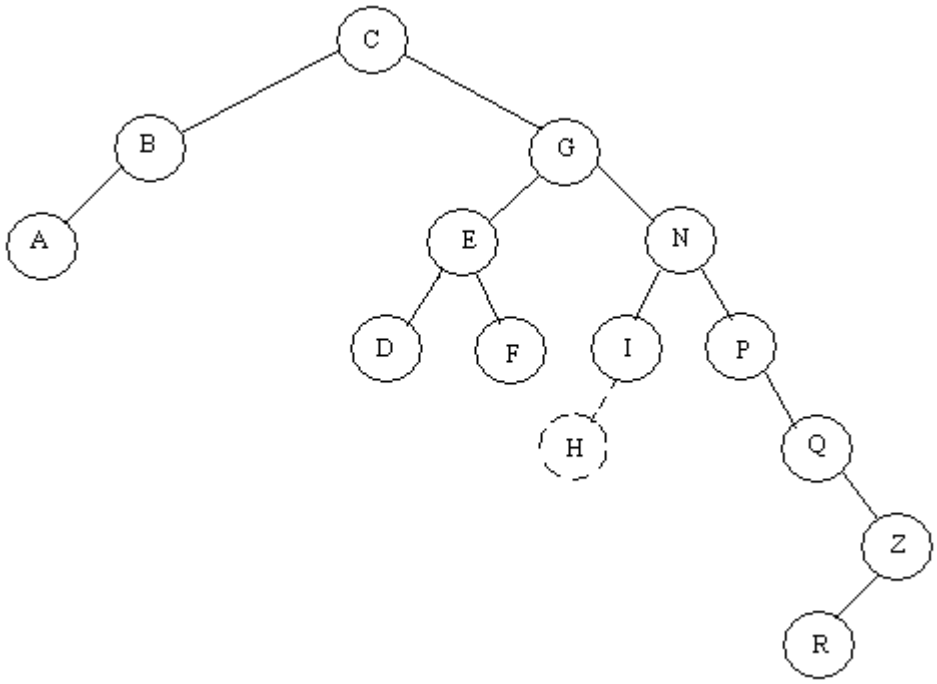
Traženje čvora u sortiranom binarnom stablu obavlja se tako da usporedimo traženu vriijednost s vrijednošću u korijenu. Ako su te dvije vrijednosti jednake, traženje je završeno jer je korijen traženi čvor. Inače, ako je tražena vrijednost manja od korijena, rekurzivno ponavljamo isti postupak u lijevom pod-stablu, a ako je tražena vrijednost veća od korijena, rekurzivno ponavljamo isti postupak u desnom pod-stablu. Ako je odgovarajuće pod-stablo prazno, zaključujemo da traženi čvor ne postoji.

Na primjer, ako u prethodnom stablu tražimo F, tada krećemo od korijena; budući da je F>C, skrećemo udesno; budući da je F<G, dalje skrećemo ulijevo; budući da je F>E dalje skrećemo udesno te pronalazimo čvor F. Ako u istom stablu tražimo J, tada opet krećemo od korijena, te zatim dvaput skrećemo udesno i jednom ulijevo; iz čvora I trebali bi dalje skrenuti udesno no odgovarajuće pod-stablo je prazno pa zaključujemo da čvor J ne postoji.

Ubacivanje čvora

Ubacivanje čvora u sortirano binarno stablo obavlja se tako da ga najprije pokušamo pronaći u već postojećem stablu. Ako ga nađemo, tada ga zapravo ne treba ubacivati. Ako ga ne nađemo, tada je postupak traženja odredio mjesto gdje ga treba ubaciti (to je mjesto gdje smo naišli na prazno pod-stablo).

Na primjer, ubacivanjem čvora H u sortirano binarno stablo s prethodne slike dobivamo sljedeće sortirano binarno stablo. Novoubačeni čvor označen je isprekidanim crtama.



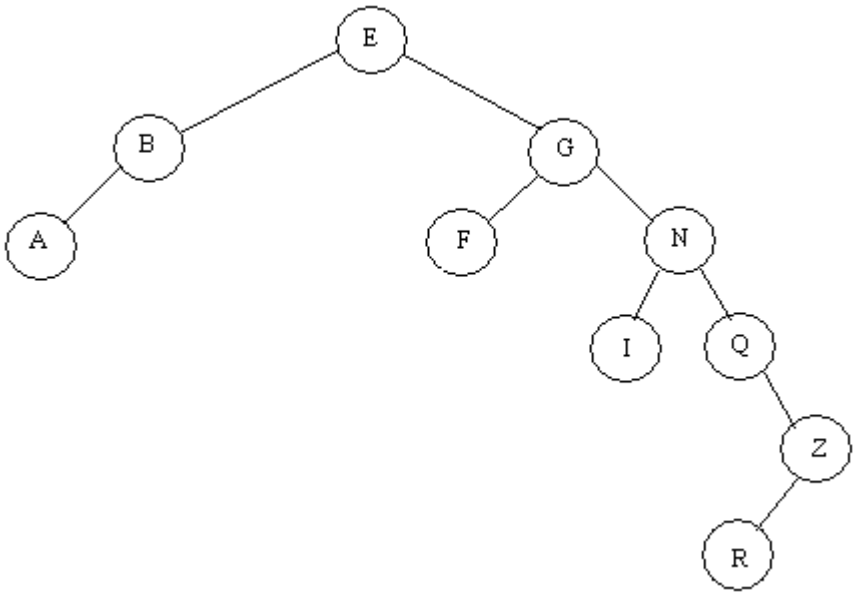
Izbacivanje čvora

Izbacivanje čvora sa zadanom vrijednošću iz sortiranog binarnog stabla nešto je kompliciranije od ubacivanja. Najprije postupkom traženja odredimo da li takav čvor postoji ili ne, te ako postoji koliko ima djece. Dalje razlikujemo sljedeće slučajeve.

- Ako čvor ne postoji, ne treba ga ni izbacivati.
- Ako čvor postoji i nema djece, jednostavno ga izbacimo iz stabla.
- Ako čvor postoji i ima jedno dijete, tada ga izbacimo, a na njegovo mjesto stavimo njegovo (jedino) dijete zajedno sa svim podređenim čvorovima tog djeteta.

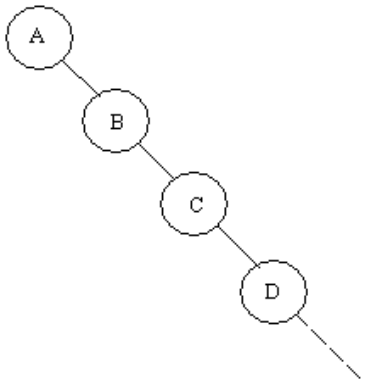
- Ako čvor postoji i ima dvoje djece, tada najprije izbacimo najlijeviji čvor iz njegovog desnog pod-stabla (to se svodi na jedan od prethodna dva slučaja). Zatim vrijednost čvora kojeg smo htjeli izbaciti promijenimo tako da postane jednaka vrijednosti spomenutog najlijevijeg čvora iz desnog pod-stabla kojeg smo upravo izbacili.

Na primjer, iz sortiranog binarnog stabla s polazne slike izbacujemo redom čvorove D, P, C. Dobivamo stablo koje je prikazano na idućoj slici.



Vremenska složenost operacija sa čvorovima

Sortirano binarno stablo koje sadrži unaprijed zadani skup podataka-čvorova dobiva se postepenim ubacivanjem jednog po jednog čvora, s time da se krene od praznog stabla. Konačni izgled stabla ovisi o redoslijedu ubacivanja čvorova. Na primjer, stablo s polazne slike odgovara redoslijedu C, B, G, A, E, F, D, N, P, Q, Z, I, R. Ako bi čvorove ubacivali po abecedi, dobili bi sljedeće «koso» stablo.



Primijetimo da svaka od operacija ubacivanja, izbacivanja ili traženja u sortiranom binarnom stablu prolazi jednim putom od korijena do (najdalje) lista. Zato je vrijeme

izvršavanja jedne operacije ograničeno odozgo s dubinom stabla, dakle s brojem razina u njemu. Sa stanovišta efikasnog obavljanja operacija, od različitih sortiranih binarnih stabala koja prikazuju jedan te isti skup podataka pogodnija su nam ona koja su što više razgranata, dakle ona koja imaju dubinu reda O(log *n*) gdje je *n* broj podataka odnosno čvorova. Koso stablo je najmanje pogodno budući da mu je dubina proporcionalna s *n*.

OBILAZAK BINARNOG STABLA

Obilazak je postupak kojim «posjećujemo» čvorove binarnog stabla, tako da svaki čvor posjetimo točno jednom. To je potrebno ako želimo obaviti neku obradu nad svim čvorovima, na primjer ispis, prebrajanje, računanje prosjeka i slično.

Najpoznatiji obilasci su preorder, inorder i postorder. Ta tri postupka zadaju se rekurzivno na sljedeći način.
- *Preorder:* posjeti korijen, obiđi lijevo pod-stablo, obiđi desno podstablo.
- *Inorder:* obiđi lijevo pod-stablo, posjeti korijen, obiđi desno pod-stablo.
- *Postorder:* obiđi lijevo pod-stablo, obiđi desno pod-stablo, posjeti korijen.

Za binarno stablo sa polazne slike, tri obilaska posjećuju čvorove u sljedećem redoslijedu.

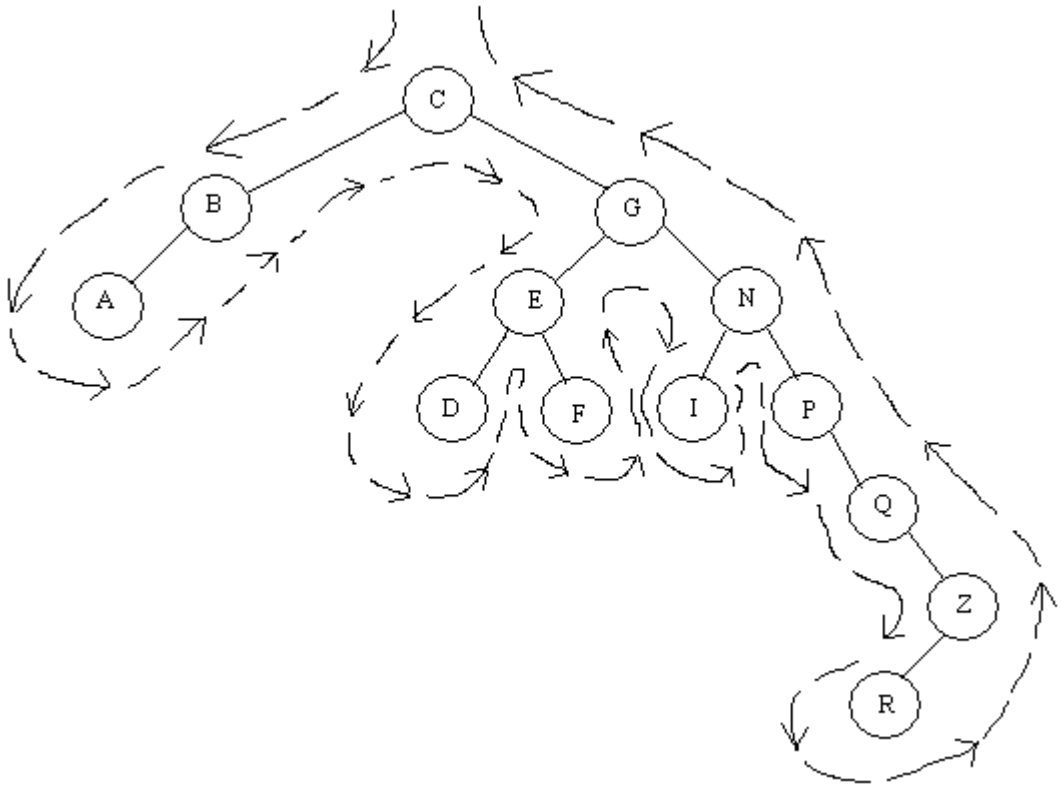
- Preorder: C,B, A, G, E, D, F, N, I, P, Q, Z, R.

- Inorder: A, B, C, D, E, F, G, I, N, P, Q, R, Z.

- Postorder: A, B, D, F, E, I, R, Z, Q, P, N, G, C.

Zorna interpretacija za tri obilaska

Opisana tri obilaska binarnog stabla možemo zorno interpretirati kao «plovidbu» oko «obale» koju čine čvorovi i lukovi. Zamišljeni brodić plovi uz obalu i ulazi u svaki «zaljev». Na taj način, brodić se svakom čvoru približava tri puta: najprije slijeva, zatim odozdo, te na kraju sdesna. Kod preorder-obilaska, čvor se posjećuje onda kad mu se brodić prvi put približio (slijeva). Kod inorder-obilaska, čvor se posjećuje onda kad mu se brodić drugi put približio (odozdo). Kod postorder-obilaska, čvor se posjećuje onda kad mu se brodić treći put približio (sdesna). Sve je to ilustrirano idućom slikom.



Sortiranje obilaskom binarnog stabla

Iz definicije sortiranog binarnog stabla i inorder-obilaska očigledno je da inorder-obilazak posjećuje čvorove sortiranog binarnog stabla u sortiranom redoslijedu. Zaista, inorder-obilazak najprije posjećuje sve čvorove koji su manji od korijena, zatim korijen, te na kraju čvorove koji su veći od korijena – dakle korijen se u nizu posjećenih čvorova nalazi na svom «pravom mjestu» u smislu uzlazno-sortiranog poretka. Isto svojstvo vrijedi zbog rekurzije i za sve ostale čvorove, pa je cijeli niz posjećenih čvorova ispravno sortiran. Kao primjer, pogledajmo opet sortirano binarno stablo s polazne slike i njegov inorder-obilazak.

Uočeno svojstvo sortiranog binarnog stabla i inorder-obilaska može se iskoristiti za oblikovanje još jednog algoritma za sortiranje – to je tzv. *tree-sort*. Algoritam kreće od praznog sortiranog binarnog stabla. Podaci koje treba sortirati redom se ubacuju u stablo. Nakon što su svi podaci ubačeni, obilazimo stablo postupkom indorder i ispisujemo posjećene podatke u sortiranom poretku.

Ako su podaci koje sortiramo algoritmom tree-sort dobro izmiješani, tada će dobiveno sortirano binarno stablo biti dobro razgranato, tj njegova dubina bit će reda $\log_2 n$, gdje je n broj podataka koje sortiramo. Budući da svaka operacija ubacivanja zahtijeva vrijeme proporcionalno dubini stabla, te budući da imamo n ubacivanja, gradnja stabla zahtijeva vrijeme $O(n \log n)$. Inorder-obilazak očito ima linearnu složenost $O(n)$. Zato cijeli algoritam sortiranja ima vrijeme $O(n \log n)$. Ipak, ta ocjena vrijedi samo za vrijeme u prosječnom slučaju. Najgori slučaj nastupa na primjer onda kad je niz podataka već sortiran: binarno stablo se tada pretvara u koso stablo s dubinom n, pa cijeli algoritam sortiranja zahtijeva $O(n^2)$ operacija.

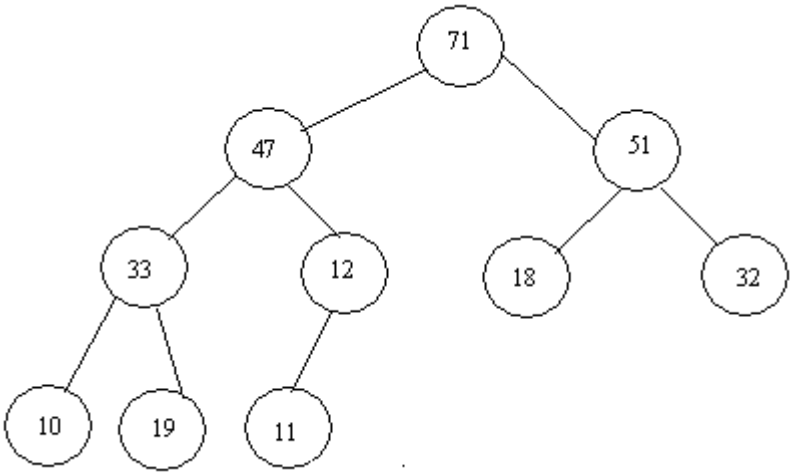
PRIORITETNI RED I GOMILA

Prioritetni red je struktura podataka koja donekle podsijeća na obični red. Sličnost s običnim redom je u tome što se podaci mogu dodavati u prioritetni red te skidati iz njega. Razlika je u tome što se ne skida onaj podatak koji je prvi bio dodan, već onaj koji ima najveću vrijednost.

Prioritetni red može se prikazati na razne načine.
- Prikaz sortiranim poljem ili sortiranom vezanom listom. Dodavanje novog podatka tada se obavlja umetanjem na «pravo mjesto» u smislu sortiranog poretka. Skidanje se obavlja na jednom od krajeva polja odnosno liste (tako gdje je najveći podatak).
- Prikaz sortiranim binarnim stablom. Dodavanje novog podatka obavlja se ubacivanjem odgovarajućeg čvora. Skidanje se obavlja izbacivanjem «najdesnijeg» čvora budući da je on najveći.

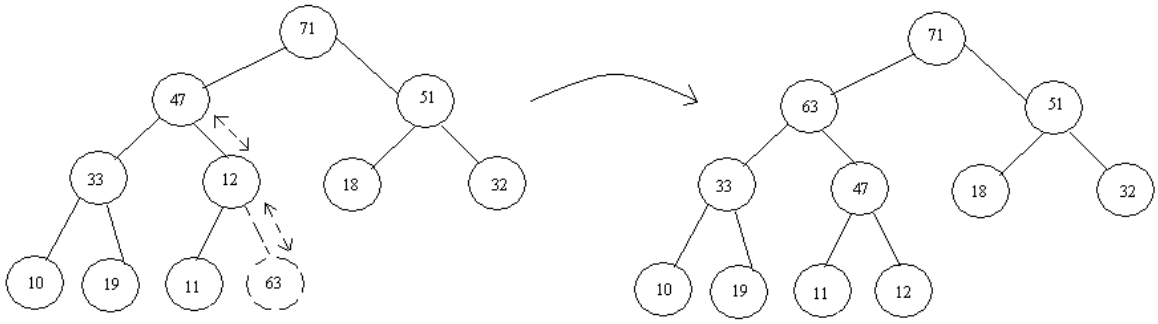
Ipak, najefikasniji način prikaza prioritetnog reda je pomoću gomile (hrpe, heap). *Gomila* je potpuno binarno stablo gdje se čvorovi mogu uspoređivati nekom uređajnom relacijom <=, i gdje je bilo koji čvor u smislu te relacije veći ili jednak od svoje djece.

Na primjer, prioritetni red s podacima 10, 19, 51, 71, 12, 18, 32, 47, 33, 11 mogao bi se prikazati gomilom koja se vidi na sljedećoj slici. Očigledna posljedica definicije gomile je da se najveći podatak (onaj koji je prvi na redu za skidanje iz prioritetnog reda) mora nalaziti u korijenu gomile.



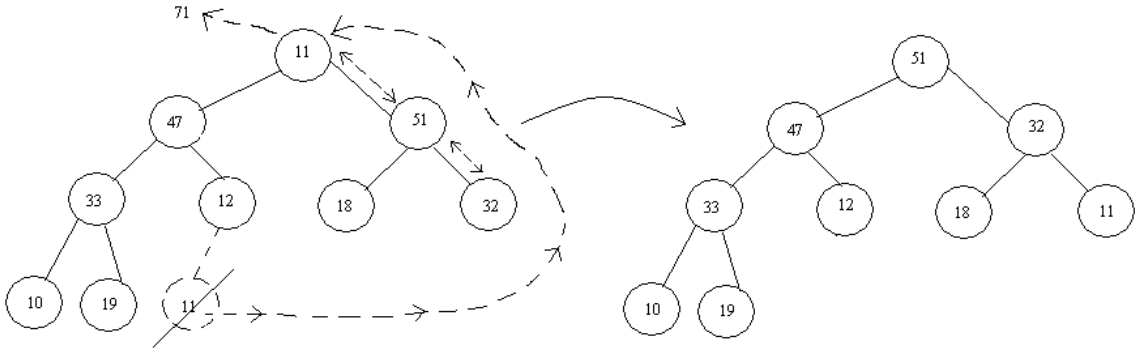
Dodavanje u gomilu

Dodavanje novog podatka u prioritetni red prikazan gomilom provodi se tako da se na zadnju nepopunjenu razinu gomile na prvo slobodno mjesto dodaje novi list s tim novim podatkom. Time se možda narušilo svojstvo gomile jer je novi list možda veći od svog roditelja. Popravak se obavlja tako da se novi list zamijeni sa svojim roditeljem, zatim s roditeljem od roditelja, ... itd. dok god je potrebno. Iduća slika prikazuje ubacivanje podatka 63 u gomilu s prethodne slike.



Skidanje s gomile

Skidanje najvećeg podatka iz prioritetnog reda prikazanog gomilom provodi se tako da se «vrati» vrijednost korijena, vrijednost zadnjeg lista sa zadnje razine se prepiše u korijen, te se zatim izbaci taj zadnji list. Time se možda narušilo svojstvo gomile jer je novi korijen možda manji od svog djeteta. Popravak se obavlja tako da se korijen zamijeni sa svojim većim djetetom, zatim s većim djetetom od djeteta, ... itd. dok god je potrebno. Iduća slika prikazuje skidanje najvećeg podatka 71 iz gomile s početne slike.



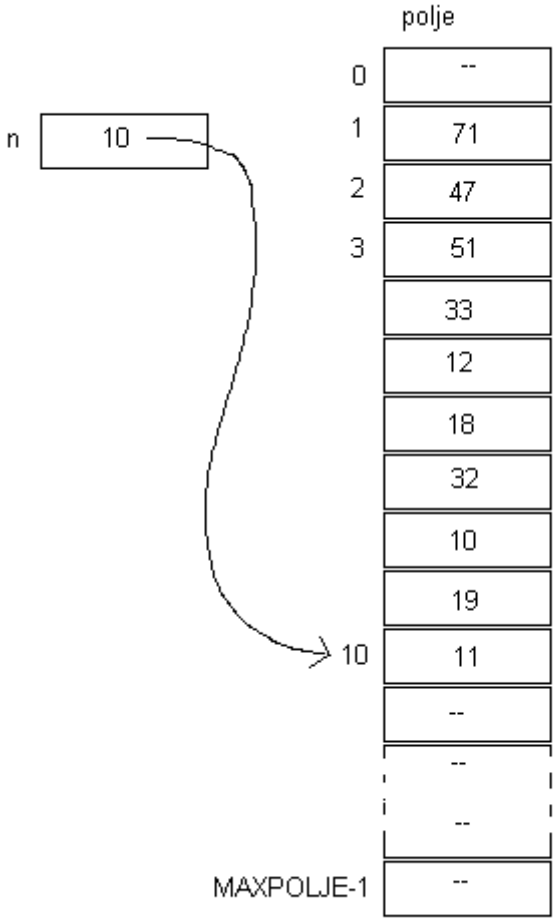
Vremenska složenost dodavanja i skidanja

Vidimo da se operacija dodavanja (odnosno skidanja) u prioritetnom redu od n podataka svodi na pronalaženje puta u gomili od n čvorova, i to od lista pa najdalje do korijena (odnosno od korijena pa najdalje do lista). Zato je vrijeme izvršavanja takve operacije ograničeno dubinom gomile (brojem razina). Budući da potpuno binarno stablo od n čvorova ima dubinu $\sim\log_2 n$, slijedi da je apriorna ocjena vremena izvršavanja u najgorem slučaju za obje operacije $O(\log n)$.

Ukoliko je na početku zadan skup od n podataka koji čine prioritetni red, tada se odgovarajuća gomila može stvoriti iz prazne gomile postepenim dodavanjem jednog po jednog podatka. Budući da jedno dodavanje ima računsku složenost u najgorem slučaju $O(\log n)$, cijeli postupak gradnje gomile zahtijeva vrijeme $O(n \log n)$. Na službenim folijama opisan je poboljšani postupak gradnje, koji odjednom uzima u obzir sve podatke (umjesto da ih gleda jednog po jednog) i koji radi u vremenu $O(n)$.

Prikaz gomile pomoću polja

Budući da je gomila jedno potpuno binarno stablo, ona se u memoriji računala vrlo efikasno može prikazati pomoću polja, tako da se podatak iz *i*-tog čvora pohrani u *i*-tom elementu polja. Osim polja potrebna nam je i kazaljka na zadnji element gomile. Na primjer, gomila s početne slike prikazuje se poljem i kazaljkom koje se vide na sljedećoj slici. Manevriranje po gomili u smjeru roditelj -> dijete ili obratno lagano se ostvaruje množenjem ili dijeljenjem indeksa elementa polja s 2.

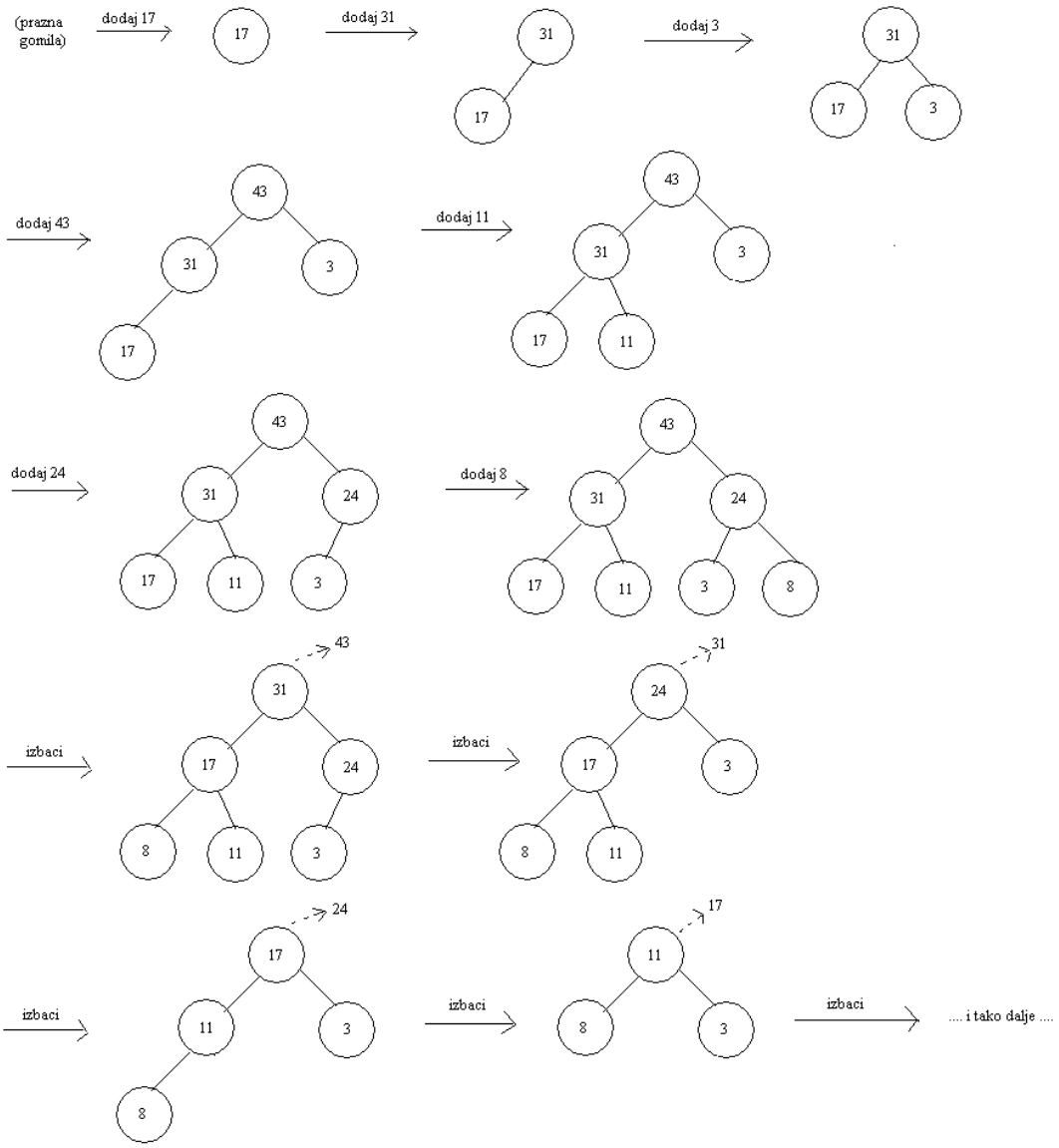


Sortiranje pomoću gomile

Slično kao sortirano binarno stablo, i gomila se može primijeniti za sortiranje. Podaci koje želimo sortirati najprije se redom dodaju u gomilu, a zatim se redom skidaju sa gomile. Zahvaljujući svojstvima gomile, podaci koje skidamo izlaze iz gomile u silazno-sortiranom poretku, dakle od najvećeg prema najmanjem. Zamjenom uređajne relacije <= s relacijom >= mogli bi izravno dobiti i uzlazno-sortirani poredak.

Rezultirajući algoritam za sortiranje naziva se *heap-so*rt. Algoritam se obično implementira tako da se gomila pohrani u polju, te se u konačnici svodi na manipuliranje s podacima u polju (slično kao i ostali algoritmi za sortiranje).

Na idućim slikama vidi se sortiranje niza podataka 17, 31, 3, 43, 11, 24, 8 algoritmom heap-sort. Koristimo gomilu s nepromijenjenom uređajnom relacijom <=, tako da algoritam izbacuje podatke u silazno-sortiranom redoslijedu.



Budući da se heap-sort svodi na *n*-struku primjenu dodavanja i skidanja u gomili, te budući da dodavanje i skidanje imaju složenost O(log *n*), slijedi da je vrijeme u najgorem slučaju cijelog heap-sort-a O(*n* log *n*), gdje je *n* broj podataka koje treba sortirati. Eksperimenti pokazuju da heap-sort spada među najbrže poznate algoritme za sortiranje i da uspješno sortira vrlo velika polja.