Izvještaj

Zadatak ovog projekta je bio implementirati klasu „Stablo“ koja pruža podršku za rad sa binarnim stablima pretrage. Implementirano stablo mora biti dvostruko povezano, tj. da svaki čvor u stablu ima informaciju o svojoj djecu, ali i roditelju. Osim toga, potrebno je bilo dodati klasu „Iterator“ koja pruža podršku za rad sa iteratorima. Za iteratore je bilo potrebno implementirati operatore ++ i -- (i postfiksnu i prefiksnu verziju), operatore += i -=, operator *. Operator ++ treba da pomjeri iterator na naredni element po veličini unutar stabla. Slično, operator -- treba da pomjeri iterator na prethodni element po veličini unutar stabla. Ukoliko se iterator već nalazi na kraju stabla, tj. pokazuje iza najvećeg elementa, tada operator ++ ne treba da radi ništa. Slično pravilo važi i za operator --. Operator += treba da pomjeri iterator naprijed za n mjesta, gledano u sortiranom poretku. Ukoliko nema dovoljno čvorova za pomjeranje, odgovarajuća operatorska funkcija treba da baci izuzetak. Slično vrijedi i za operator -=. Operator * treba da omogući pristupanje sadržaju čvora za koji je iterator vezan. Također, klasi „Stablo“ se treba dodati podrška za rad sa iteratorima, tj. odgovarajuće metode koje rade sa iteratorima. Tako je trebalo dodati metodu „begin“ koja vraća iterator vezan za najmanji element u stablu, te metodu „end“ koja vraća iterator koji „pokazuje“ tačno iza najvećeg elementa. Ovakva implementacija binarnog stabla pretrage, ne samo da čuva elemente povezane u stablo, nego su ti elementi povezani u ulančanu listu, koja je pri tome sortirana, te dvostruko povezana. Prvo, primjetimo da je struktura „Cvor“ koja služi za kreiranje čvorova unutar stabla trebala imati sljedeće podatke: ključ/vrijednost koju će čuvati u sebi, pokazivač na roditeljski čvor, pokazivače na lijevo i desno dijete, te osim toga pokazivače na prethodni i sljedeći element stabla gledano u sortiranom poretku. Ideja je da pri ubacivanju, odnosno brisanju elementa u stablo, ažuriramo te podatke i tako čuvamo povezanost elemenata kako u stablo, tako i u ulančanu listu. Stablo se zapravo sastoji od tri atributa: korijen, taj atribut predstavlja samo stablo, tj. njegov korijen; najveci i najmanji, to su atributi koji opisuju listu, tj. početak i kraj ulančane sortirane liste.

Za stablo smo implementirali samo jedan trivijalni konstruktor bez parametara koji atribute postavlja na nulu, tj. pravi prazno stablo, stablo bez elemenata. Pošto stablo može biti jako glomazna struktura, međusobno dodjeljivanje, konstruktor kopije, te prenos stabala kao

parametre u funkcije (kao i vraćanje stabala kao rezultat funkcija) smo zabranili jer navedene radnje bi jako dugo trajale za velika stabla i ne vidi se njihova svrha. Te radnje su zabranjene stavljanjem konstruktora kopije i preklopljenog operatora dodjele u privatni dio klase i njihovo neimplementacijom. Također, destruktor je implementiran uz pomoć dodatne metode „unisti“ koja prvo izbrise cvor, zatim njegovu djecu. Ta metoda pozvana nad korijenom će izbrisati cijelo stablo, tj. uništiti ga, što je zapravo i posao destruktora, pa destruktor zapravo i jeste pozivanje navedene metode nad korijenom.

Jedna od najvažnijih metoda je metoda „umetni“, koja ima zadatak da umetne novi član u stablu a da pri tome očuva njegovu povezanost u vidu ulančane sortirane liste. Za tu potrebu, dovoljno je umetnuti element na klasični način u stablo (prateći put do mjesta na koje treba umetnuti novi element), ali pri tome pamtiti određene vrijednosti. Pri umetanju elementa u stablo, moramo odrediti koji element (čvor) je njegov prethodnik, odnosno sljedbenik u sortiranom poretku. Primjetimo da je prethodnik čvora zadnje desno „skretanje“, tj. čvor kod kojeg smo zadnji put skrenuli desno u pronalasku mjesta za umetanje čvora u stablo. Slično, sljedbenik čvora je zadnje lijevo „skretanje“. Uz te podatke imamo sve potrebne informacije za umetanje čvora u stablo. Isto tako, da bi očuvali povezanost elemenata u povezanu listu, potrebno je dati element umetnuti između njegovo prethodnika i sljedbenika, što je trivijalno, poznajući algoritme za umetanje u listu. Na taj način smo očuvali dobru povezanost elemenata u stablu. Primjetimo da svaki čvor ima informaciju o svom prethodniku i sljedbeniku. To će nam dobro doći kasnije, jer na taj način implementacija operatora ++, --, +=, -= postaje trivijalna za klasu „Iterator“.

Još jedna vrlo važna metoda je metoda koja briše čvor iz stabla. Moramo voditi računa da trebamo ažurirate podatke u čvoru koji su vezani za stablo, ali i one koji se tiču povezanosti čvora u listu. Brisanje iz liste je trivijalno, samo se moraju ažurirati odgovarajući pokazivači prethodnika i sljedbenika elementa koji se briše. Brisanje iz stabla je nešto komplikovanije. Razlikovali smo tri slučaja, kada čvor koji se briše nema djece, kada ima samo jedno dijete (ovaj slučaj ima dva podslučaja, kada je to dijete lijevo odnosno desno), te kada čvor ima obadva djeteta. Prvi slučaj je veoma jednostavan, potrebno je samo ažurirati informaciju o djeci roditeljskog čvora od onoga koji brišemo. Nakon toga je potrebno bukvalno izbrisati iz memorije dati čvor. Ukoliko čvor koji brišemo ima samo jedno dijete, potrebno je samo

preurediti stablo tako da to dijete bude novo dijete roditeljskog čvora od onog koji brišemo. Nakon toga ponovo obrišemo čvor. Ukoliko čvor koji brišemo ima obadva djeteta, situacija se komplikuje. Potrebno je pronaći najveći čvor u lijevom podstablu čvora koji brišemo i staviti ga na mjesto čvora koji brišemo. Pri tome voditi računa o njegovim podstablima i svim ostalim pokazivačima. Nakon izmjene odgovarajućih pokazivača, potrebno je fizički ukolniti čvor iz memorije. Time smo ažurirali povezanost i u stablo i u listu, te nam je stablo i nakon brisanja čvora ponovo dobro povezano. Naravno, pri svim ovim postupcima moramo voditi računa o nekim posebnim slučajevima, npr. šta se dešava ukoliko je čvor koji brišemo zapravo korijen stabla, ili najmanji ili najveći element i sl. Ova metoda za brisanje čiji je parametar čvor, je privatna i koriste je samo neke druge javne metode stabla. Te metode služe za brisanje elementa prema njegovom ključu, odnosno brisanje elementa za koji je vezan neki iterator. Metodu za brisanje čiji je parametar iterator jednostavno obriše čvor na koji je vezan iterator, te ažurira iterator tako da pokazuje na naredni element u listi. Ta metoda je javna i mogu je pozivati korisnisi klase. Isto tako, metoda čiji je parametar broj prvo pronađe taj broj u stablu (u koliko postoji). Pronađe ga pomoću metodu „find“ o kojoj će biti govora nešto kasnije. Metoda find vrati kao rezultat iterator koji vezan na čvor sa datim brojem, te izbriše čvor na koji je vezan taj iterator.

Implementirane su još i metode koje služe za obilazak čvorova stabla u PostOrder, InOrder te PreOrder redoslijedu. Za svaki od navedenih redoslijeda obilaska, implementirane su po dvije metode, jedna koja se poziva nad čvorom i druga koja se poziva nad sadržajem čvora. Također, implemenitrana je i metoda „ispisiCvor“ koja ispisuje sadrzaj čvora koji je proslijeđen kao parametar. Ona služi za ispisivanje elemenata u određenomo poretku, tako što se ta metoda, tj. statička funkcija članica proslijedi kao parametar PostOrder, InOrder i PreOrder metodama.

Metode klase „Iterator“ su prilično trivijalne zbog načina na koji smo implementirali strukturu „Cvor“ i metode za umetanje odnosno brisanje u stablu. Iterator se interno sastoji od dva pokazivača, pokazivača na trenutni čvor, tj. na čvor za koji je vezan iterator, te na njegov prethodnik. To nam je potrebno u jednom slučaju, kada dođemo do kraja liste. U tom slučaju pokazivač „trenutni“ treba da pokazuje „ni na šta“, tj. da bude nul-pokazivač, ali ipak nekako moramo sačuvati informaciju da smo na kraju liste, što ćemo uraditi u drugom

pokazivaču koji se nalazi u iteratoru. Dakle, u svakom trenutku prvi pokazivač iteratora pokazuje na trenutni element, dok drugi pokazuje na njegovog prethodnika, osim u slučaju kad se nalazimo na kraju liste, tada prvi pokazivač pokazuje ni na šta, dok drugi pokazivač pokazuje na najveći element liste. Analogno razmatranje vrijedi za početak liste. Sada je lagano implementirati operatore ++, --, += i -=. Operator ++ prosto treba da pomjeri obadva pokazivača na njihove sljedbenike, vodeći računa o slučaju kada smo došli na kraj liste ili eventalno o slučaju kada smo bili na početku liste. Slična stvar vrijedi za operator --. Implementirane su i postfixna i prefixna verzija oba operatora. Operator += je sličan operatoru ++, samo u ovom slučaju se pomjeramo n puta naprijed. Ukoliko smo došli do kraja liste, a nismo se pomjerili dovoljan broj puta, to znači da nam nedostaje čvorova, i tada je potrebno baciti izuzetak iz odgovarajuće operatorske funckije. Analogno razmatranje vrijedi i za operator -=. Operator * prosto treba da vrati vrijednost u čvoru za koji je iterator vezan i on je trivijalan.

Također, u klasi „Stablo“ su implementirane tri metode koje služe za podršku za rad sa iteratorom. Metoda begin vraća iterator vezan za najmanji element u stablu, end vraća iterator koji „pokazuje“ iza najvećeg elementa u stablu, dok metoda find vraća kao rezultat iterator vezan za čvor koji u sebi sadrži traženi broj ili iterator iza kraj ukoliko taj broj nije u stablu.

Implementirana je i trivijalna metoda koja testira da li je stablo prazno. Ona prosto testira vrijednost pokazivača na korijen stabla, te ukoliko je to nul-pokazivač vraća informaciju da je stablo prazno, u suprotnom vraća false, tj. informaciju da stablo nije prazno.