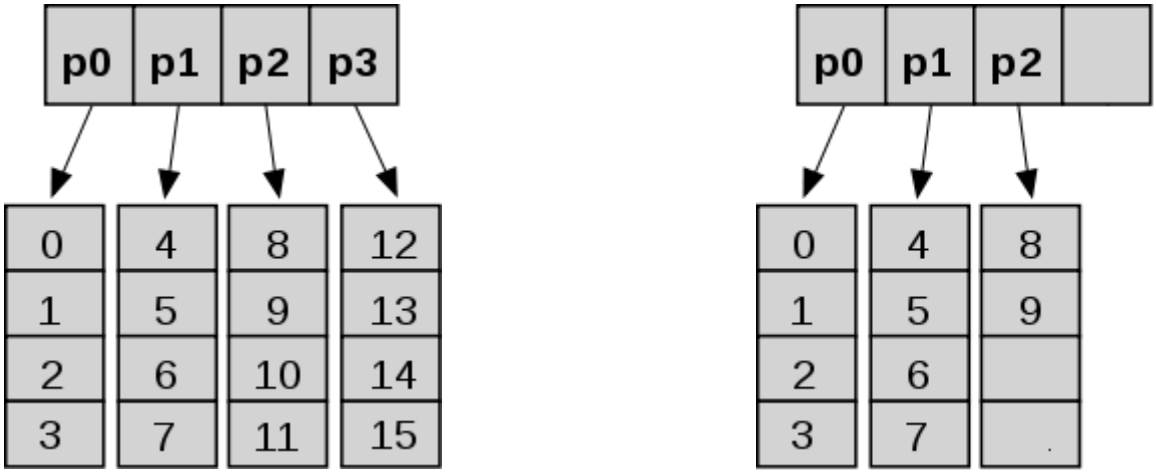
OBJEKTNO ORIJENTISANO PROGRAMIRANJE

- domaći zadatak broj 2 -

Funkcionalna specifikacija

Na programskom jeziku C++ implementirati biblioteku struktura podataka, u kojoj treba da postoje klase *HashedArrayTree* (fragmentirani dinamicki niz) i *HATIterator* (iterator). U nastavku teksta opisane su koncepti HashedArrayTree, HATIterator i BigInt.



Slika 1 Primer punog HAT niza sa 16 elemenata (levo) i delimično popunjenog HAT niza sa 16 elemenata. Prazni pokazivači su označeni nemaju sadržaj. Prazne kućice označavaju nedostajuće ključeve.

Specifikacija klase HashedArrayTree

Fragmentiran niz elemenata (*HashedArrayTree*) predstavlja vrstu dinamičke nizovne strukture podataka, koja ima bolje iskorišćenje memorije kada je polupopunjena u odnosu na klasicne kontinualno alocirane dinamicke nizove. Sastoji se od glavnog niza koji se zove direktorijum fragmenata, i u njemu se nalazi M pokazivača na fragmenate. Podaci se smeštaju u nizove koji se zovu fragmenti. Broj M mora biti stepen dvojke. Fragmenti niza su iste dužine kao i direktorijum. Pun niz sadrži M^2 podataka. Kako se radi o stepenu dvojke, moguće je za indeksiranje koristiti brže bitske operacije, umesto kompleksnijeg deljenja i računanja ostatka. Za isti broj elemenata, ova struktura daje bolje iskorišćenje memorije za razliku od kontinualne strukture, jer se fragmenti alociraju samo kad treba dodatni prostor za dodavanje novih elemenata (sve dok se struktura ne popuni, kao na slici 1 levo). Iako je ova struktura suštinski niz, njen naziv impliciran na nešto sasvim drugo. Naziv je skovan na osnovu toga sto reprezentacija niza podseća na stablo i tu prestaja svaka druga sličnost. U nastavku teksta za *HashedArrayTree* će biti korišćen termin niz. Fragmenti niza sadrže pokazivače na objekte klase BigInt, implementirane u prvom domaćem zadatku.

Ekspanzija i redukcija prostora.

Kada je niz prazan, postoji samo direktorijum, koji ne sadrži nijedan fragment. Dodavanjem prvog elementa u niz, alocira se prvi fragment. Narednih M-1 podataka takođe biva smešteno u prvi fragment. Dodavanje M+1 podatka, uzrokuje alokaciju sledećeg fragmenta i tako redom, sve dok se ne umetne M^2 elemenata, čime niz postaje pun. Dodavanje narednog elementa u pun niz, zahteva restrukturiranje niza. Restrukturiranje niza podrazumeva promenu unutrašnjih struktura, direktorijuma, kao i fragmenata, tako što se red veličine direktorijuma menja za jedan nivo više ili manje, u zavisnosti od toga da li se dešava ekspanzija ili redukcija prostora.

Ekspanzijom prostora, veličina direktorijuma se uvećava na sledeći stepen dvojke. Na primer, ako je se niz na slici popuni sa svih 16 elemenata, ubacivanje 17. elementa zahteva promenu veličine direktorijuma sa 4 na 8 (2^3), pri čemu se i veličina fragmenata povećava sa 4 na 8. Postojeće elemente treba preseliti iz trenutno popunjene strukture na odgovarajuće pozicije u novoj strukturi. Kada se svi elementi iz trenutno pune strukture presele u novu, nova struktura ima samo 1/4

popunjenih mesta (16 od maks. 64 = 8x8). Za ubacivanje 17. elementa potrebno je alocirati samo još jedan fragment.

Redukcijom prostora, veličina direktorijuma, kao i fragmenata, se smanjuje za red veličine i sistemu sa osnovom 2. Ako se iz prethodnog primera izbaci 17. element iz niza, stiču se svi uslovi da se interna struktura niza redukuje za jedan red veličine. Redukcija podrazumeva premeštanje postojećih podataka u novu (redukovanu) strukturu.

Nakon svakog dodavanja ili izbacivanja podatka iz niza, potrebno je ispitati da li je strukturi podataka potrebno restrukturiranje, i ako jeste, izvršiti ga. Za restrukturiranje predvideti pomoćne metode i samo ih pozivati na odgovarajućim mestima.

Niz se stvara prazan sa zadatim parametrom K.

Za potrebe stvaranja niza, implementirati konstruktor klase `HashedArrayTree(int k);` sa parametrom K, podrazumevano 2, gde K predstavlja red veličine unutrašnjih struktura, tako da je $M=2^K$.

Niz se može stvoriti kao kopija već postojećeg niza.

Implementirati obe varijante konstruktora kopije za potrebe kopiranja stabala. Kopiranje niza podrazumeva kopiranje strukture, kao i kopiranje pojedinačnih podataka.

Nizu se može dodeliti stanje drugog niza (operator=).

Implementirati dodelu sadržaja jednog niza drugom (pomoću kopiranja i/ili premeštanjem). Preklopiti operator za dodelu vrednosti (operator=) u obe varijante.

Dohvatanje broja elemenata u nizu (size).

Implementirati metodu int size() const; koja vraća broj elemenata u nizu.

Dodavanje elementa u niz (operator+=).

`void add(const BigInt& b);` je metoda koja dodaje dati broj b na kraj niza. Ukoliko je niz već pun, potrebno je izvršiti restrukturiranje. Metoda pravi dinamičku kopiju zadatog broja i umeće je u niz.

Preklopiti operator+= kao alternativnu podršku za umetanje elemenata u niz koji treba implementirati korišćenjem date metode.

Umetanje elementa na zadatu poziciju (insert).

`void insert(const BigInt& num, int idx) throw (IllegalIndexException);`

Implementirati metodu koja umeće dinamičku kopiju zadatog broja na zadatu poziciju uz odgovarajuće pomeranje ostatka niza. Ukoliko index prevazilazi trenutnu veličinu niza ili ima pogrešnu vrednost, prijaviti grešku podizanjem izuzetka tipa *IllegalIndexException*.

Dohvatanje elementa na zadatoj poziciji (get).

`const BigInt& get(int idx) const throw (IllegalIndexException);`

Implementirati datu metodu tako da vraća referencu na podatak na zadatoj poziciji. Ukoliko index prevazilazi veličinu niza ili ima pogrešnu vrednost, prijaviti grešku podizanjem izuzetka tipa *IllegalIndexException*.

Indeksiranje (operator[], read only).

Preklopiti operator za indeksiranje koji se koristi samo za čitanje podatka na zadatom indeksu. Implementacija operatorske funkcije mora da se svodi na poziv metode get.

Izbacivanje podatka iz niza (operator-=).

Implementirati metodu `int remove(BigInt& num)` koja izbacuje iz niza sve pojave podatka num, uz potrebno pomeranje ostalih elemenata niza i eventualno restrukturiranje. Metoda vraća broj izbačenih elemenata. Preklopiti operator-= kao alternativnu podršku za izbacivanje zadatog podatka iz niza, koji treba da se svede na pomenutu metodu. Operatorska funkcija vraća referencu na objekat niza za koji je pozvana.

Pravljenje iteratora za obilazak elemenata (HATIterator).

Implementirati metodu `HATIterator iterator() const;`.

Vraća se objekat iteratora, koji se pozicionira pre početka niza. Iterator sme da se napravi samo u

objektu niza i nikako drugačije. Ne treba razmatrati izmene stanja niza, dok je itereriranje u toku. Ti aspekti neće biti razmatrani u testu. Detaljnije o iteratoru i sledećoj sekciji.

Omogućiti propisno uništavanje niza.

Implementirati destruktor koji treba da oslobodi celopnu memoriju zauzetu i za strukturu i za podatke.

Specifikacija klase HATIterator

HATIterator omogućava linearni prolazak kroz niz i dohvatanje jednog po jednog podatka.

Pravljenje iteratora.

Iterator sme da napravi samo objekat klase HashedArrayTree. Implementirati privatni konstruktor koji sme da se pozove samo u klasi niza. Kada se napravi, iterator se logički namešta "ispred" prvog elementa niza. Studenti samostalno treba da osmisle implementaciju iteratora.

Može se proveriti da li postoji naredni element

Implementirati metodu `bool hasNext() const;` za ispitivanje da li postoji sledeći element u nizu. Ukoliko je iterator došao na kraj niza, ili je niz prazan, vraća false.

Prelazak na sledeći element (operator++).

Implementirati metodu `void move() throw(InvalidIteratorStateException);` koja pomera iterator na sledeći element. Ukoliko se iterator pomera na sledeći element u trenutku kada hasNext vraća vrednost false, potrebno je prijaviti grešku izuzetkom tipa

Preklopiti operator ++ kao alternativni način pomeranje iteratora na sledeći element. Operatorska funkcija ne vraća vrednost, i treba da se svede na datu metodu. Preklopiti i prefikasni i postfiksni operator.

Dohvatanje trenutnog podatka (operator*).

```
const BigInt& value()const throw(IllegalIteratorStateException);
```

Metodu koja vraća referencu na koji iterator trenutno "ukazuje". Ukoliko iterator ne ukazuje na podatak (pre pocetka ili na kraju), prijaviti grešku podizanjem izuzetka *InvalidIteratorStateException*. Preklopiti `operator*` kao alternativni način za dohvatanje vrednosti iteratora, koja treba da se svodi na pomenutu metodu. Operatorska funkcija ima istu povratnu vrednosti kao i metoda value.

Ograničenja.

Objekte klase HATIterator smeju da prave sključivo objekti niza. Ne sme se dozvoliti kopiranje iteratora, kao ni dodela vrednosti.

Specifikacija dopuna funkcionalnosti klase BigInt.

Preklopiti potrebne operatore u klasi BigInt, sa ciljem da se pojednostavi sintaksa zapisivanja izraza za velikim celim brojevima i učini istom kao za ugrađene tipove podataka. Sve operatorske funkcije funkcionalno i manirski moraju da budu implementirane tako da imaju isti interfejs i ponašanje kao i operatorske funkcije za ugrađene tipove podataka.

Sve operatorske funkcije moraju biti implementirane tako da se svode na poziv odgovarajuće metode koja odradi suštinski posao. Nije dozvoljeno kompletnu implementaciju operacije smestiti direktno u operatorsku funkciju, osim ako nije trvijalna toliko da se svodi na jedan izraz. Ukoliko za neke operatorske funkcije ne postoje odgovarajuće pozadinske (background) metode, dodati ih klasi BigInt.

Aritmetički operatori (+, -, *, /).

Preklopiti redom binarne operator +,-,* i /. Operatori imaju objekte kao argumente i vraćaju privremeni rezultat kao objekat iste klase. Ne smeju menjati operande.

Relacioni operatori (<, >, <=, >=, ==, !=).

Preklopiti sve navedene relacione operatore. Svi operatori rade sa objektima kao operandima. Semantički, svaki operator treba da ima isto ponašanje kao i za ugrađene tipove.

Operator !.

Preklopiti operator! koji proverava da li je veliki broj jedna nuli.

Test funkcija

U projektu glavnog programa treba da postoji funkcija `void test();` koja testira rad sa velikim brojevima. Studenti treba da implementiraju datu funkciju i uslovno je prevode ako je definisan makro STUDENT_TEST. Predvideti i postojanje makroa PROF_TEST. Ako su oba makroa definisana, PROF_TEST ima prioritet i tada se prevodi tajni test primer koji će biti dat na odbrani. Funkcija main treba samo da pozove test funkciju i na kraju ispiše njeno trajanje u milisekundama, korišćenjem tipova i operacija iz zaglavlja <ctime>.

Tehnički zahtevi i smernice za izradu rešenja

Sve klase i metode moraju biti imenovane prema zahtevima iz domaćeg zadatka.

Sve klase i metode moraju biti imenovane prema zahtevima iz domaćeg zadatka. Poljima klase, zaštićenim od direktnog pristupa, se pristupa isključivo pomoću odgovarajućih metoda za čitanje i pisanje vrednosti polja. Za smešanje tekstualnih vrednosti upotrebiti tačno onoliko mesta u memoriji koliko je neophodno. Svaka klasa koja koristi dinamičku memoriju mora imati korektno napisan destruktor i konstruktor kopije. Izuzetno u ovom zadatku, klase sa destruktorom koji nije automatski generisan smeju imati automatski generisan operator dodele vrednosti, ali ga ne treba koristiti!

Glavni program treba da poziva metode/operacije koje obavljaju opisane radnje. Sve metode smestiti u odgovarajuće klase. Programski kod klasa rasporediti u odgovarajuće **.h** i **.cpp** fajlove. Nije dozvoljeno korišćenje globalnih promenljivih za razmenu podataka. Sva razmena podataka između funkcija mora ići preko povratne vrednosti i/ili liste argumenata. U slučaju bilo kakve greške (poziv programa sa neodgovarajućim brojem argumenata komandne linije, neuspešna dodela dinamičke memorije, greška pri radu sa datotekom ili bilo koja druga greška koja se može pojaviti u toku izvršavanja programa), prijaviti grešku podizanjem izuzetka. Greške se obrađuju u try-catch bloku.

Ukoliko u zadatku nešto nije dovoljno jasno definisano, treba usvojiti razumnu pretpostavku i na temeljima te pretpostavke nastaviti izgrađivanje svog rešenja.

VAŽNE NAPOMENE

Za uspešno odbranjen domaći zadatak potrebno je na odbrani pokazati kod podeljen na odgovarajuće projekte, .h i .cpp fajlove.

- Klase za velike brojeve smestiti u u poseban projekat rešenja koji se prevodi kao statička biblioteka (**`bigint.lib`**).
- Klase za nizovnu strukturu podataka smestiti u poseban projekat rešenja koji se prevodi kao statička biblioteka (**`hat.lib`**).
- Glavni program napisati u posebnom projektu koji se prevodi kao Win32 Console Application (**`testdz2.exe`**) fajl i koji treba povezati sa statičkim bibliotekama. Glavni program treba implementirati tako da pozove globalnu funkciju `void test();`.
- Studenti treba da implementiraju svoju verziju ove funkcije tako da demonstriraju operacije sa velikim brojevima.
- Na kraju programa potrebno je ispisati na standardnom izlazu vreme trajanja funkcije `test` u milisekundama. Može se iskoristiti kod za merenje vremena na jeziku C++ koji je dat u zadatku 2.8 u materijalima za vežbe.
- NIJE DOZVOLJENO SMESTITI CEO KOD U JEDAN PROJEKAT ILI CPP fajl!

21.11.2015. godine sa predmeta