Projekat je rađen u sklopu predmeta Strukture podataka i algoritmi na Odsjeku za matematičke i kompjuterske nauke Prirodno-matematičkog fakulteta Univerziteta u Sarajevu. Ova dokumentacija opisuje funkcionalnosti implementiranih funkcija.

ZADATAK

Implementirati stablo koje sadrži:

1) Iterator i reverse iterator za kretanje kroz stablo. Petlja za prolaženje kroz stablo treba da radi kao što radi i u strukturu set. Prolazak kroz čitavo stablo je vremenske kompleksnosti *O(n)*. Obavezno dodati funkcije *Begin()* i *End()* koristeći dva fiktivna čvora. Neophodno je omogućiti da radi petlja:

   `for(auto it = s.Begin(); it != s.End(); it++)`

2) Prijateljsku funkciju koja provjerava da li je jedno stablo podskup drugog. Funkcija treba da radi u vremenu *O(m+n)*, gdje su m i n brojevi elemenata ta dva stabla.

3) Indeksiranje, tako da `stablo[k]` vraća k-ti po veličini element stabla (indeksiranje kreće od nule). Funkcija treba da radi u vremenu *O(h)*, gdje je h visina stabla.

Student: Ema Djedović

Profesor: Prof. dr. Esmir Pilav

Asistent: Mr. Admir Beširević

01/2024, PMF, Sarajevo

Imamo pet fajlova: *stablo.h, stablo.cpp, main.cpp* te dodatno *podstablo.h* i *podstablo.cpp*. Najvažniji fajl nam je *stablo.cpp* u kojem je implementirana većina funkcija. Ovdje se nalaze samo one koje je bilo potrebno dodatno objasniti, dok se većina korisnih natuknica može naći i u samom kodu kroz C++ komentare.

STABLO.H

Sadrži **klasu Stablo** unutar koje se nalaze **struktura Cvor** te **klase inOrderIterator** i **inOrderReverseIterator**.

STABLO.CPP

Implementirane metode deklarisane u fajlu *stablo.h*.

`Cvor* nadjiCvor(int vrijednost)`

- ➢ Krećemo od korijena i u ovisnosti od toga je li vrijednost manja ili veća od one u korijenu nastavljamo pretragu rekurzivno u (respektivno) lijevom ili desnom podstablu.

`void dodajCvor(int vrijednostInput)`

- ➢ Ako je korijen nullptr onda nemamo stablo pa nova vrijednost postaje korijen.
- ➢ Pratimo hoće li naša nova vrijednost postati najmanja ili najveća vrijednost kako bismo znali po potrebi ažurirati krajnji lijevi ili krajnji desni čvor.
- ➢ Spuštamo se niz stablo analogno kao u nadjiCvor dok ne nađemo ispravnu poziciju za novi čvor.
- ➢ Kada smo našli lokaciju onda kreiramo novi čvor, svim roditeljima povećavamo broj nasljednika za jedan, ažuriramo vrijednosti sljedeciInOrder i prethodniInOrder…

`void inOrder(Cvor* trenutniKorijen)`

- ➢ Rekurzivno prolazi kroz lijevo podstablo, posjećuje trenutni čvor, a zatim rekurzivno prolazi kroz desno podstablo.
- ➢ Tijekom prolaza, ispisuje vrijednosti čvorova.
- ➢ Posljedica je sortiran niz naših čvorova u stablu.

`void inOrderReverse(Cvor* trenutniKorijen)`

- ➢ Rekurzivno prolazi kroz desno podstablo, posjećuje trenutni čvor, a zatim rekurzivno prolazi kroz lijevo podstablo.
- ➢ Analogno kao inOrder ispisuje vrijednosti čvorova i sortira čvorove u obrnutom poretku po vrijednosti.

`void levelOrder(Cvor* trenutniKorijen)`

- ➢ Koristi red za obradu čvorova tako što počevši od korijena izlistava nivo po nivo.
- ➢ Red se popunjava lijevim i desnim djecom svakog obrađenog čvora.
- ➢ Funkcija je korisna za debagiranje i stvaranje slike o strukturi stabla.

`void inOrderVektorNapuni(Cvor* trenutniKorijen, vector<int>& linearnaStruktura) const`

- ➢ Vrši in-order prolaz počevši od datog čvora (trenutniKorijen) i puni vektor vrijednostima čvorova.
- ➢ Na taj način od nelinearne dobivamo linearnu strukturu s kojom je lakše raditi.

`void pripremiZaKMP(const vector<int>& stabloVektor, vector<int>& lps) const`

- ➢ Priprema LPS niz *(the longest prefix which is also a suffix)* za dati in-order vektor stabla koristeći KMP (Knuth-Morris-Pratt) algoritam.
- ➢ Inicijalizira varijable za praćenje dužine prefiksa (duzina) i indeksa (i,j) u skladu s KMP algoritmom.
- ➢ Prolazi kroz vektor i ažurira niz LPS vrijednostima.

`bool traziSaKMP(const vector<int>& veceStabloVektor, const vector<int>& manjeStabloVektor) const`

- ➢ Koristi KMP algoritam kako bi našao vektor manjeg stabla unutar vektora većeg stabla. Koristi funkciju pripremiZaKMP (jer algoritam sam po sebi nije pogodan za int vrijednosti).
- ➢ Iterira kroz oba vektora i upoređuje elemente. Ako pronađe podniz vraća true, inače vraća false.

`bool bPodstabloOdA(const Stablo& A, const Stablo& B)`

- ➢ Formira in-order vektore oba stabla pa koristi KMP algoritam za traženje podniza (podstabla) kako bi se osigurala linearna vremenska kompleksnost.

`bool bPodskupOdA(const Stablo& A, const Stablo& B)`

- ➢ Ukoliko je B prazno vraća true.
- ➢ Ukoliko su i A i B prazni vraća true.
- ➢ Formira in-order vektore oba stabla i ukoliko je vektor od B veći od A vraća false.
- ➢ While petljom ide kroz oba vektora u isto vrijeme dok ne dođe do kraja barem jednog i provjerava određene uslove, čime se osigurana efikasnost O(n).
- ➢ Ukoliko smo prvo izašli iz B onda vraća true, u suprotnom false (izašli iz A a nismo našli sve elemente iz B).

`int ktiNajveci(int k, Cvor* trenutniKorijen)`

- ➢ Koristi informaciju o broju nasljednika kako bi se efikasno dobio rezultat, odnosno postižemo vremensku efikasnost O(visina_stabla).
- ➢ Vraćamo -1 (nepostojeći indeks) ukoliko je prazno stablo ili je k izašlo iz opsega.
- ➢ Ukoliko postoji desno dijete onda pogledamo koliko nasljednika ima desno dijete. Ukoliko ima k-1 nasljednika znači da je k čvorova većih od korijena (uključujemo i prvo desno dijete), stoga je stablo[k] vrijednost u korijenu. Kako indeksiranje počinje od 0 to je (k+1). najveći element.
- ➢ Ukoliko je desno više od k elemenata onda rekurzivno nastavljamo tražiti ktiNajveci u desnom podstablu.
- ➢ Ukoliko je desno manje od k elemenata i korijen ima lijevo dijete onda isti proces radimo na lijevo podstablo.
- ➢ Ukoliko desnog stabla uopšte nema onda ako je k=0 vraćamo korijen, a inače ukoliko postoji lijevo dijete rekurizvno tražimo (k-1). najveći u lijevom podstablu (jer kako smo izbacili korijen onda imamo jedan čvor manje).
- ➢ U *stablo.h* je implementiran operator [] koji poziva ovu funkciju.

PODSTABLO.CPP

Sadrži testnu funkciju za kod koji provjerava je li jedno stablo podstablo drugog, odnosno za `bool bPodstabloOdA(const Stablo& A, const Stablo& B)`. Unosi se početno stablo (s1), jedno podstablo (s2) i jedno koje nije podstablo (s3). Ta testna funkcija se po potrebi poziva u *main.cpp*.

MAIN.CPP

Sve tražene funkcionalnosti se testiraju kroz tri glavne funkcije:

```
void PrviZadatak(Stablo s);
void DrugiZadatak();
void TreciZadatak(Stablo s);
```

s tim da je redoslijed pozivanja druge i treće zamijenjen jer se testiraju na istom Stablu “s” pa je zgodno da su rezultati blizu.

Dodatno se može pozvati i testna funkcija iz *podstablo.cpp*.