HÁZI FELADAT

Programozás alapjai 2.

Végleges

Borbola Martin
AC380P

2016. május 15.

Tartalom

1. Feladat

```
Programozás alapjai II. házi feladat
Borbola Martin (AC380P) részére:

Készítsen GENERIKUS bináris fát! A kulcsok közötti rendezettséget a
szokásos relációs operátorokkal vizsgálja, amit szükség esetén specializál!

Valósítsa meg az összes értelmes műveletet operátor átdefiniálással
(overload), de nem kell ragaszkodni az összes operátor átdefiniálásához!
Amennyiben lehetséges használjon iterátort!

Demonstrálja a működést külön modulként fordított tesztprogrammal! A
megoldáshoz NE használjon STL tárolót vagy algoritmust!

A tesztprogramot úgy specifikálja, hogy az parancssoros batch
alkalmazásként (is) működjön, azaz a szabványos bemenetről olvasson, és a
szabványos kimenetre, és/vagy a hibakimenetre írjon!
Amennyiben a feladat teszteléséhez fájlból, vagy fájlokból kell input
adatot olvasnia, úgy a fájl neve *.dat alakú legyen!
```

2. Pontosított feladatspecifikáció

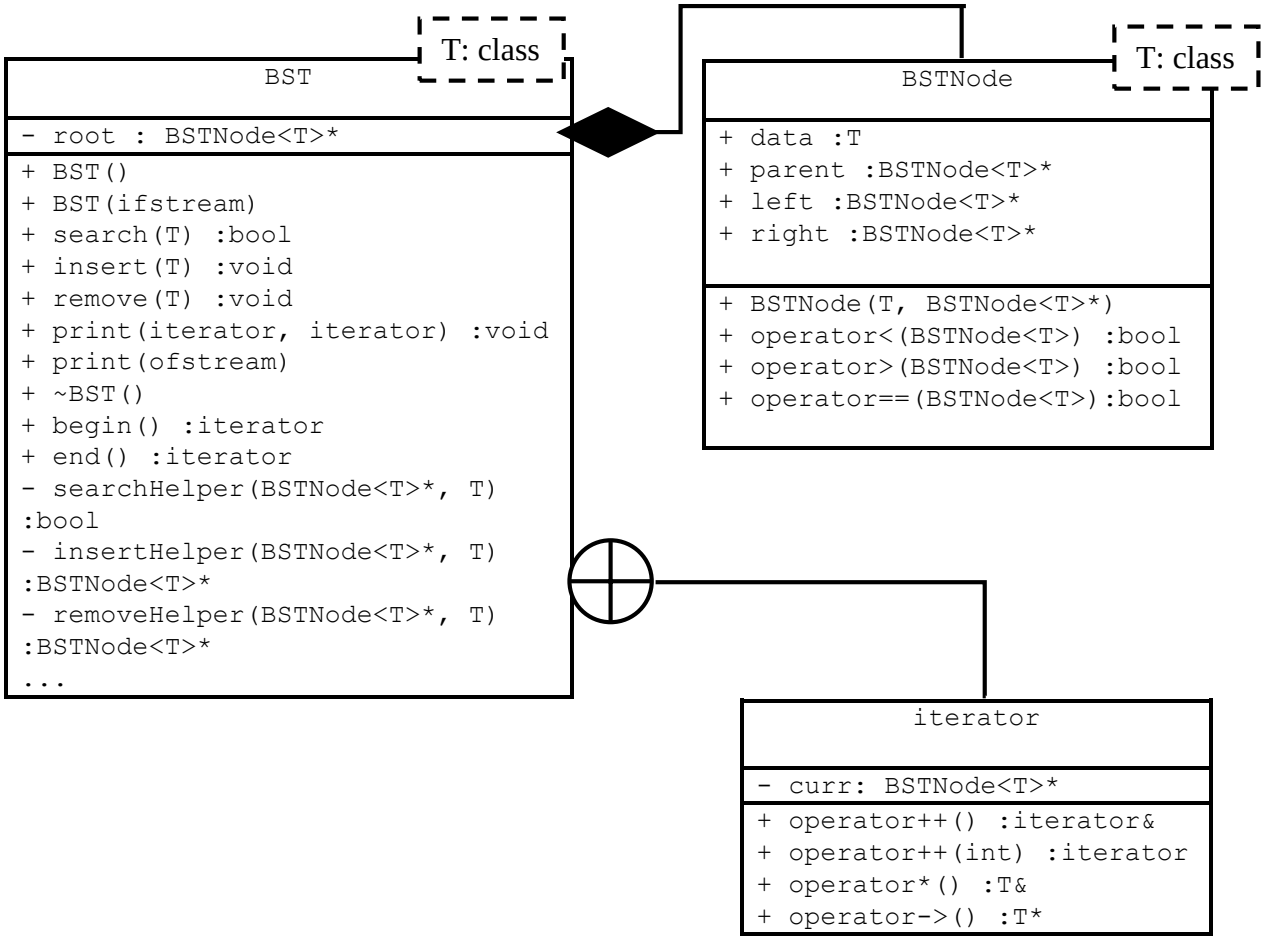
A feladat egy generikus bináris fa elkészítése. A fa típusát sablon paraméterként lehet megadni. Az automatikusan létrejövő tagfüggvények mellett (másolás, értékadás, létrehozás, megszüntetés) keresést, beillesztést és a törlés műveletet valósítom meg, lehetőleg operátor átdefiniálással. Ezenkívül lehet fájlba menteni és onnan betölteni.
A teszteléséhez egy olyan programot készítek, ami különböző adattípusokkal létrehozott fákkal a standard inputról beolvasott parancsok alapján teszteket végez.

3. Terv

A feladat egy objektum és a tesztprogram megtervezését igényli.

3.1 Objektum terv

A generikus bináris fát két sablonnal fogom megvalósítani. Egy struktúraként a fa elemeit és egy osztályként az magát a fát. A sablon sablonparaméterként veszi át a fa elemeinek típusát.
Az iterator belső osztály, csak print(interator first, iterator last) függvénynél használom.



3.2 Algoritmusok

3.2.1 Az adatszerkezet algoritmusai

Az inorder kiíráson és az iterátorok next() függvényén kívül minden algoritmust rekurzívan valósítottam meg.

Keresés:

1. a gyökér elemtől indulunk
2. ha az aktuális nem létezik, akkor nincs a fában a keresett
3. összehasonlítjuk a keresett elemmel
   - ha pont az, akkor végeztünk
   - ha nagyobb, akkor balra megyünk tovább
   - ha kisebb, akkor jobbra megyünk tovább
4. folytatjuk a 2. ponttól

Beszúrás:

1. a gyökér elemtől indulunk
2. ha nem létezik, akkor hozzávesszük
   - bal részfára, ha kisebb
   - jobb részfára, ha nagyobb
3. ha nem létezik, akkor hozzávesszük

Törlés:

1. megkeressük a törlendő elemet
2. ha nincs gyereke, egyszerűen töröljük a pointert és a foglalt területet
3. ha egy gyereke van, átirányítjuk a pointert és töröljük a köztes elemet
4. ha két gyereke van, a törlendőt egyenlővé tesszük a jobb részfa minimumával, és töröljük a jobb részfából a minimumot

Inorder bejárás:

1. bal részfa
2. akármilyen művelet
3. jobb részfa

Preorder bejárás (fájlba írás):

1. kiírás
2. bal részfa
3. jobb részfa

Postorder bejárás (felszabadítás):

1. bal részfa
2. jobb részfa
3. törlés

Következő elem:

1. ha van jobb oldali részfája, annak a legbaloldali levele
2. ha nincs, akkor addig megyünk felfele amíg bal részfából jövünk

3.2.2 Tesztprogram algoritmusai

A tesztprogram a standard inputról egy teszteset sorszámát olvassa be. Ez dönti el, hogy melyik teszteset fut. Minden teszteset mást tesztel.

4. Megvalósítás

A feladat megoldásához két osztályt hoztam létre. A BST használja a BSTNode-ot. Az osztályok használják az std:iostream, std:fstream osztályokat. A végleges interfészen csak nevek változtak.
A sablonok forrása a BST.hpp, míg a tesztprogram a BST.cpp fájlba került. A továbbiakban bemutatom a fontosabb interfészeket és algoritmusokat.

4.1 Az elkészített osztálysablon bemutatása

A sablon a paraméterként átvett típusú dinamikus pointerekben tárolja a generikus adatokat. A másoló konstruktort és az értékadó operátort priváttá tettem, így tilos egy fa másolása.
A generikus adatokról feltételezzük, hogy van és helyesen működik az:

- operator< függvénye
- operator> függvénye
- operator== függvénye

A rekurzív megvalósítás miatt a search, remove, insert, print, bejáró függvényeknek és a destruktornak vannak Helper függvényeik. Ezekben történnek a tényleges műveletek.

5. Tesztelés

A teszteléshez létrehoztam egy SajatOsztaly nevű osztályt, ami dinamikus adattaggal is rendelkezik.
A tesztesetekben ezzel is kipróbáltam az adatszerkezetet.
A lefedettségi tesztnél csak azok a részek nem futottak le, amik nem voltak kötelezők.