مینی پروژه سوم

درخت AVL (AVL Tree)

بخش پیاده‌سازی

برای پیاده‌سازی این پروژه تنها مجاز به استفاده از کتابخانه‌های زیر می‌باشید:

Functions and Classes	Header File
همه توابع و کلاس‌ها	#include <iostream>
assert(predicate);	#include <cassert>

std::min(T, T);
std::max(T, T);
std::swap(T&, T&);
	#include <algorithm>

همه توابع	#include <cmath>
همه ثابت‌ها	#include <cfloat>
همه ثابت‌ها	#include <climits>

برای هر تابع کامنت‌گذاری کرده و موارد زیر را توضیح دهید:

- عملکرد تابع
- کاربرد هر پارامتر
- فرضیات تابع
- خطاهایی که ممکن است رخ دهد و نحوه مدیریت‌شان

نام کلاس شما باید دقیقا SearchTree بوده و در فایل search_tree.h پیاده‌سازی شود. یک درخت جستجوی دودویی (Binary Search Tree) برای شما پیاده‌سازی شده است. شما باید تغییرات موردنیاز برای تبدیل این درخت به یک درخت AVL را اعمال کنید. مورد دیگری که باید به درخت جستجوی دودویی جاری اضافه کنید، یک پیمایشگر (Iterator) است. ساختار و امضاهای موردنیاز برای پیمایشگر ارائه شده است؛ تنها کافی است آن‌ها را پیاده‌سازی کنید. پس از پیاده‌سازی، پیمایشگر به صورت زیر مورد استفاده قرار می‌گیرد.

```
SearchTree myTree;
// Add some nodes to my tree

// This gets an iterator that currently points to the
// node containing the smallest entry (the 'front') of the search tree.
SearchTree::Iterator itr = myTree.begin();

for (SearchTree::Iterator itr = myTree.begin(); itr != myTree.end(); ++itr ) {
    std::cout << *itr << " ";
}
std::cout << std::endl;
```

کد بالا تمامی مقادیر موجود در درخت را چاپ می‌کند. برای حرکت به گره بعدی و قبلی از دو دستور itr++ و itr– استفاده می‌شود. دقت کنید که از itr++ و itr-- استفاده نکنید. شکل پیاده‌سازی این دو متفاوت است.

فایل tester.cpp برای آزمون پیاده‌سازی شما نوشته شده است. این کلاس، دستوراتی را به شکل خلاصه شده از ورودی دریافت کرده و توابع متناظر از لیست پیوندی پیاده‌سازی شده توسط شما را فراخوانی می‌کند. صحت پیاده‌سازی شما براساس مقادیری که این فایل در خروجی استاندارد چاپ می‌کند، بررسی خواهد شد. شما نیز می‌توانید با استفاده از همین فایل به تست پیاده‌سازی خود بپردازید.

دستور	توضیحات
new	یک درخت جدید ایجاد می‌کند
end	آخرین درخت ایجاد شده را حذف می‌کند
exit	تست را خاتمه می‌دهد
size	اندازه آخرین درخت ایجاد شده را چاپ می‌کند (تعداد گره‌ها)
height	ارتفاع آخرین درخت ایجاد شده را چاپ می‌کند
print	مقدار تمامی خانه‌های آخرین درخت ایجاد شده را با استفاده از پیمایشگر چاپ می‌کند.
printDFS	پیمایش اول عمق درخت را همراه با ارتفاع هر گره چاپ می‌کند.
empty	تابع empty را برای آخرین درخت ایجاد شده فراخوانی کرده و نتیجه را چاپ می‌کند.
clear	تابع clear را برای آخرین درخت ایجاد شده فراخوانی می‌کند.
front	تابع front را برای آخرین درخت ایجاد شده فراخوانی کرده و نتیجه را چاپ می‌کند.
back	تابع back را برای آخرین درخت ایجاد شده فراخوانی کرده و نتیجه را چاپ می‌کند.
insert x	تابع insert را برای آخرین درخت ایجاد شده فراخوانی کرده و مقدار x را برای آن ارسال می‌کند
erase x	تابع erase را برای آخرین درخت ایجاد شده فراخوانی کرده و مقدار x را برای آن ارسال می‌کند
find x	تابع find را برای آخرین درخت ایجاد شده فراخوانی کرده و نتیجه را چاپ می‌کند.

نکته: پیچیدگی هیچ عملگری نباید بیشتر از $O(lgn)$ باشد. تنها پیمایش همه گره‌های درخت است که پیچیدگی آن برابر با $O(n)$ می‌باشد.

نکته: دو تابع front و back در صورتی که درخت خالی باشد، خطایی از نوع Underflow پرتاب می‌کنند.

یک نمونه ورودی

```
new
 insert 1
 insert 5
 insert 3

 printDFS
 print
 size
 height
 front
 back
end
```

خروجی نمونه بالا

```
true
true
true
START->[3, 1]->[1, 0]->[5, 0]->END
1 3 5
3
1
1
5
```