

TDDI16: Datastrukturer och algoritmer

Lab 1: AVL-träd

Tommy Färnqvist, Rita Kovordanyi, Filip Strömbäck

1 Uppgift

I den givna koden finns ett AVL-träd implementerat. Din uppgift är att komplettera AVL-trädet med operationen `remove`. I den givna koden är `remove` deklarerad i klassen `AVL_Tree`, men den kastar bara ett undantag när den anropas. En motsvarande deklaration i klassen `AVL_Tree_Node` finns inte, men det ska du göra.

Operationen `remove` ska ta bort det angivna elementet (om det finns). Utgå från den givna funktionen som finns på filen `simple_remove.cpp` och komplettera den med balansjustering. Studera hur balanskontroll och justering görs i `insert`, och gör i princip samma sak i `remove`. Tänk dock noga igenom var en obalans ligger i förhållande till noden som har obalans, och hur man kan avgöra om en enkelrotation eller en dubbelrotation ska göras! På den punkten skiljer sig borttagning påtagligt från vad som gäller vid insättning, även om koden ska bli snarlik. Tänk också på att borttagningen resulterar i att vissa pekare kan bli `null` i `remove` där vi i `insert` vet att de alltid är giltiga.

2 Givna filer

- Filen `avl_tree.h` innehåller definitionen av AVL-trädet och några tillhörande funktioner som ej är medlemmar (`swap` och `operator <<`).
- Filen `avl_tree.cpp` innehåller definitionerna för medlemsfunktionerna i klassen `AVL_Tree` samt definitioner för den nodtyp som AVL-trädet implementeras med, `AVL_Tree_Node`.
- Filen `avl_tree-test.cpp` innehåller ett testprogram för AVL-trädet. Det sätter in några värden i ett AVL-träd och senan kan en del av operationerna på AVL-trädet testas interaktivt.
- En funktion för borttagning i ett enkelt binärt sökträd finns på filen `simple_remove.cpp`.
- För att kompilera programmet finns en make-fil, `Makefile`. Kommandot `make` kör den.

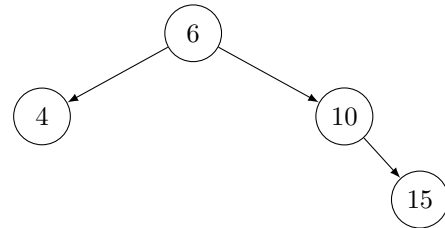
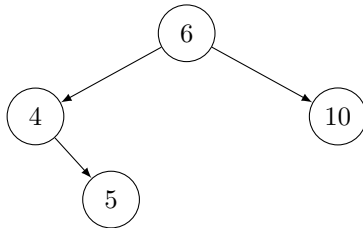
3 Fundera på

Vad får din `remove`-funktion för tidskomplexitet och minneskomplexitet? Vad borde den ha för tid- och minneskomplexitet?

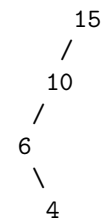
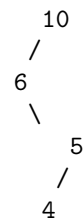
4 Testfall

Nedan finns tre testfall din kod måste klara av. Observera att din kod naturligtvis måste fungera även i det generella fallet och inte bara på de tre testfallen, så tänk noga igenom vilka situationer som kan uppkomma och hur de ska hanteras!

Testfallen nedan antar att borttagning av en nod med två barn görs genom att ersätta den med efterföljande nod i en inorder-iteration. Det finns andra sätt att lösa detta på, men då kommer träden efter borttagning i vissa fall inte stämma med de exempel som ges nedan.



Testfall 1 är borttagning av nod 10 i trädet ovan till vänster och testfall 2 är borttagning av nod 4 i trädet ovan till höger. Notera att träden skrivs ut med rotnoden till vänster av testprogrammet. Träden ovan ser alltså ut som nedan när de skrivs ut i terminalen:



Testfall 3 är borttagning av nod 9 i trädet nedan till vänster. Trädet kan byggas genom att köra `avl_tree-test` med följande indata: 1 9 1 4 1 11 1 2 1 6 1 10 1 12 1 1 1 3 1 5 1 7 1 13 1 8. Resultatet av borttagningen ska bli som i trädet nedan till höger.

