Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

Einführung in die Programmierung mit C++

Übungsblatt 11

Klassen und Codestruktur

Sebastian Christodoulou, Alexander Fleming

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

In dieser Übung soll ein *Set* von **int** Werten implementiert werden. Wie wir aus der Vorlesung wissen (Standardbibliothek II), ist ein Set als Binärbaum implementiert.
Dieser Binärbaum besteht aus Knoten ("Nodes"), die maximal zwei Unterbäume haben können. Unterbäume sind widerum selbst Knoten. Alle Knoten besitzen einen Wert. Der Baum ist so strukturiert, dass bezüglich des Wertes jedes Knotens gilt:

- Der Wert von Nachfolgern im linken Unterbaum ist kleiner
- Der Wert von Nachfolgern im rechten Unterbaum ist größer

Ein Wert kann maximal einmal im Baum vorkommen.

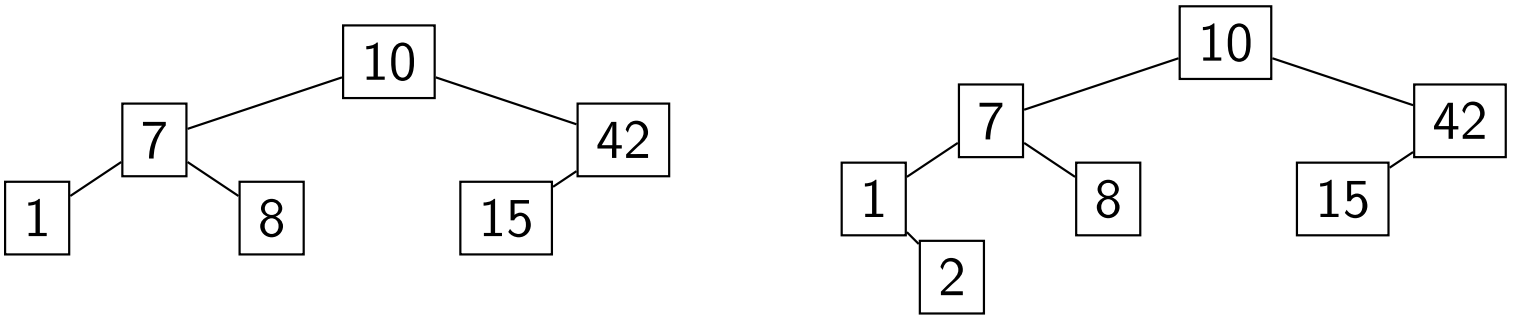

Figure: Baum vor und nach dem Einfügen eines Knotens mit Wert 2

Beim Einfügen von neuen Werten in den Baum, müssen also neue Knoten erstellt werden. Damit die oben genannten Regeln gelten, müssen Knoten an der Richtigen Stelle eingefügt werden. In unserem Fall bedeutet "einfügen", Zeiger zu setzen.

1. Die Klasse Node, (für Knoten) enthält folgende Daten:
   - einen Wert des Typs **int**,
   - einen Node Zeiger auf den Anfangsknoten des linken Unterbaums
   - einen Node Zeiger auf den Anfangsknoten des rechten Unterbaums

   Die Klasse soll folgende Routinen zur Verfügung stellen:
   - Node(**const int**& value): einen Konstruktor, der einen Knoten mit dem Wert value erstellt. Seine Zeiger auf beide Unterbäume sind mit nullptr initialisiert.
   - insert(Node* n): Hinzufügen eines Zeigers auf den Knoten n an der richtigen Stelle im Binärbaum. Um diesen an der richtigen Stelle einzufügen, muss geprüft werden, ob sein value größer oder kleiner ist, als der Wert der Node, die ihn einfügt. Kleinere Werte werden rekusriv im linken Unterbaum, und größere Werte rekursiv im rechten Unterbaum eingefügt. Falls der in Frage kommende Unterbaum leer ist (nullptr), wird dessen Zeiger auf die Node n gesetzt.
   - print(): Ausgeben der Werte des Baums auf der Kommandozeile in geordneter Reihenfolge. Hierfür tut man folgendes
     - rufe auf dem linken Unterbaum print() auf (falls dieser existiert)
     - gib den eigenen Wert auf der Kommandozeile aus

  - rufe auf dem rechten Unterbaum print() auf (falls dieser existiert)
- max_height(): Hat die Node keine Unterbäume, so wird 0 zurückgegeben. Andernfalls wird 1+ das maximum der rekusriven Aufrufe von height auf den Unterbäumen (die existieren) ausgegeben

2. Die Klasse Set besitzt folgende Daten:
   - einen Zeiger vom typ Node, der auf den Anfang eines Baumes Zeigt. Nennen wir ihn *root*.
   - ein statisches Feld values_unordered, das alle Knoten speichert, die jemals eingefügt wurden.
   - einen counter, der mitzählt, wie oft insert() aufgerufen wurde.

   Die Klasse Set soll folgende Routinen zur Verfügung stellen:
   - insert(**int** value): Erstellt eine Node mit Wert value, die in values_unordered an der Position nach der zuvor eingefügten Node abgelegt wird. Falls root zugewiesen ist, soll die erstellte Node mithilfe deren insert eingefügt werden. Andernfalls soll root auf Node zeigen.
   - print(): ruft print() auf root auf, falls root existiert. Sonst wird ausgegeben, dass das Set leer ist.
   - max_height(): Gibt das Ergebnis von max_height() von root aus, sofern sie existiert. Andernfalls wird ausgegeben, dass das Set leer ist.

3. Initialisiere ein Set und füge mittels insert(**const int**& value) ein: 10, 7, 1, 8, 42, 15, 2. Rufe dann die print() Funktion des Sets auf.
   Optional: ein Aufruf von to_dot() erstellt eine .dot-file, die man visualisieren kann: dot −Tpdf graph.dot −o graph.pdf
4. Nachdem alles funktioniert, trenne die Deklaration der Klassen und Funktionen von deren Definitionen. Schreibe dabei Deklarationen der Klassen in `.hpp`-Dateien und deren Definitionen `.cpp`. `.hpp`-Dateien kommen in den Unterordner `inc`, und `.cpp`-Dateien in den unterordner `src`. In `main.cpp` bleibt somit nur `main()`-routine, und nötige includes.
5. Schreibe ein Makefile, welches
   - die Dateien in src zu einer library namens libset.a kompiliert.
   - main.cpp kompiliert und dabei libset.a verlinkt.

   Tipp: Achte stets darauf, dass die header (.hpp) inkludiert sind, wo nötig.

Abgabe: Folgende Verzeichnisstruktur

```
./
  inc/
    Node.hpp
    Set.hpp
  src/
    Node.cpp
    Set.cpp
    Makefile
    Makefile.inc [optional]
  main.cpp
  Makefile
  Makefile.inc [optional]
```

als `nodelib.zip` oder `nodelib.tar.gz`