Inf ProgTech Übung: Balancierte Bäume

Klaus Kusche

Die Aufgabe ist dieselbe wie im Wortindex-Beispiel der Übung 4 und 5,
aber der binäre Baum soll nun *balanciert* werden, und zwar als *AVL-Baum*.

Du kannst wahlweise deine eigene vorige Lösung oder meine Musterlösung
als Ausgangsbasis nehmen.

Folgende Dinge wirst du ändern müssen, sonst solltest du möglichst nichts modifizieren:

- Im Typ der Wortknoten kommt ein *Balance-Feld* dazu
  (welchen Wert bekommt es initial in **`new_word`** ?).
  Da ich das Einfügen rekursiv gelöst habe,
  bin ich auch weiterhin *ohne* Vater-Pointer ausgekommen.
- **`save_word`** ist komplett neu zu schreiben:

  Ich habe dafür eine *rekursive Hilfsfunktion* implementiert,
  die mit einem *Pointer* **`ptr`** *auf den Pointer auf den Teilbaum* aufgerufen wird
  (d.h. in **`save_word`** mit **`&root`** und rekursiv
  mit **`&((*ptr)->left)`** oder **`&((*ptr)->right)`**)
  und als *Returnwert 0 oder 1* liefert, je nachdem,
  ob der Baum in der *Höhe* durch das Einfügen *gewachsen* oder *gleichgeblieben* ist.

  Diese Funktion besteht im Wesentlichen aus vielen **`if`**:

  - Zeigt dieser Pointer **`ptr`** auf **`NULL`**,
    so ist das Wort an dieser Stelle als *neues Blatt* anzuhängen.

    Dabei wächst der Baum logischerweise.
  - Sonst ist das Wort mit dem Wort im Knoten **`*ptr`** zu *vergleichen*.

    Bei *Gleichheit* wird ein *Positionseintrag* im Knoten **`*ptr`** angehängt,
    der Baum ändert sich dadurch nicht.
  - *Sonst* wird die Funktion *rekursiv für den linken oder rechten Sohn* aufgerufen.

    Auf dem *Rückweg aus der Rekursion* wird dann *balanciert*
    (und dabei möglicherweise **`*ptr`** auf einen anderen Knoten umgeändert).
    Dazu sind in jedem der beiden Fälle (links bzw. rechts eingefügt)
    wieder zwei Fälle zu prüfen:

    - Wenn der Returnwert 0 war, ist *nichts* zu tun.
    - Bei Returnwert 1 ist das eigene *Balance-Feld* zu prüfen (-1, 0, 1).

      In einem der drei Fälle ist eine *Balancierung* notwendig,
      in den beiden anderen Fällen nur eine *Änderung des Balance-Wertes*.

      Weiters ist für jeden der drei Fälle der entsprechende *Returnwert zu berechnen*.

- Damit diese rekursive Funktion nicht zu lang wird,
  habe ich die *Links-Rotationen und die Rechts-Rotationen*
  jeweils in eine *eigene Funktion* mit **`ptr`** als Parameter ausgelagert
  (d.h. die beiden Funktionen sind symmetrisch zueinander).

  Jede dieser Funktionen muss zuerst einmal prüfen,
  ob eine Einfach- oder eine Doppel-Rotation vorliegt.
  Dann muss sie für jeden der beiden Fälle

  - einerseits die entsprechenden *Pointer-Änderungen* vornehmen
  - und andererseits die *Balance-Felder* der verschobenen Knoten aktualisieren.

  Achtung: Bei den Balance-Werten nach einer Doppel-Rotation gibt es
  einen Fall mehr, als beim ersten Blick auf die Grafik zu vermuten ist!

Um den Effekt der Balancierung (flacherer Baum) zu sehen,
solltest du die *Statistik-Ausgaben* aus der vorigen Übung implementiert haben!

Diese Übung ist sehr komplex, die Gefahr von Fehlern ist groß.
Ich empfehle daher, eine (rekursive) Funktion zu schreiben, die im Hauptprogramm
nach dem Einlesen aller Files und vor dem Abfragen der Worte aufgerufen wird
und die *Korrektheit des entstandenen Baumes prüft*:

- Stimmt die *Sortierung*? (Linker Sohn ‹ Vater ‹ rechter Sohn für alle Knoten)
- Stimmt die *Balance*?
  (Die Funktion sollte als Returnwert die *Höhe des jeweiligen Sohn-Baumes* liefern.
  Im Vater werden dann beide Sohn-Höhen miteinander vergleichen,
  ihre Differenz mit dem Balance-Feld abgeglichen,
  und aus der größeren der beiden die neue Höhe berechnet.)