Inf ProgTech Übung: Wortindex mit unbalanciertem Baum

Klaus Kusche

Die Aufgabe ist dieselbe wie in der vorigen Übung (Index über alle Wörter angegebener Dateien), aber diesmal soll ein *binärer Baum* statt einer Hashtable für die Speicherung der Wörter verwendet werden (die Positionsliste an jedem Wortknoten bleibt unverändert).

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 ändern sich die *Verkettungs-Pointer* (2 Söhne; einen Vater-Pointer brauchen wir vorläufig nicht).
- Statt dem Hashtable-Array wird die *Wurzel* des Baumes global gespeichert und auf einen leeren Baum initialisiert; die Hashfunktion fällt weg.
- Bei **`new_word`** fällt ein *Parameter weg* (wenn ein Knoten neu in den Baum eingefügt wird: Auf welche Werte müssen seine Sohn-Pointer in jedem Fall initialisiert werden?).
- **`save_word`** und **`process_word`** sind *komplett neu* zu schreiben. Eine rekursive Lösung ist zwar elegant, aber eine iterative ist deutlich effizienter und auch nicht schwieriger.

  Für die ganz Mutigen: Bei **`save_word`** spart derselbe *Pointer-auf-Pointer-Trick* wie bei Listen eine Fallunterscheidung und ein paar Zeilen Code… (der Pointer zeigt auf den Pointer, in den der neue Knoten einzutragen ist. Das kann die Wurzel, der Links-Pointer des Vaterknotens oder der Rechts-Pointer des Vaterknotens sein).

Vorläufig genügt uns ein *unbalancierter* Baum.

Zusatzaufgaben:

- Um einen Überblick zu bekommen, wie sehr der Baum von einem optimalen Baum abweicht, könntest du als Zusatzaufgabe *Statistiken* ausgeben (*Anzahl* der Worte *auf jeder Ebene* des Baumes, *Gesamtzahl* der Worte, *durchschnittliche und maximale Tiefe*).

  Du ermittelst diese Statistiken am besten *rekursiv* (mit der Tiefe als Parameter)! Kannst du zum Vergleich auch die (berechnete) durchschnittliche Tiefe ausgeben, die ein *optimaler* Baum mit dieser Anzahl von Wörtern hätte?
- Weiters könntest du noch eine *komplette Liste aller Wörter in Sortier-Reihenfolge* ausgeben. Auch das geht am Besten mit einer *rekursiven Funktion*.