F3 – Berechenbarkeit und Komplexität

Aufgabenzettel 8 : Algorithmentechniken

Besprechung in der Zeit vom 15.12. bis zum 18.12.2003.

Präsenzaufgabe 8 :

Diskutieren Sie Gemeinsamkeiten und Unterschiede der Begriffe *Tiefensuche*, *Backtracking* und *Branch-and-Bound*.

Übungsaufgabe 8.1 :

von
14

Betrachten wir die Suche in Binärbäumen in drei Varianten der Programmierung: Tiefensuche mit back-tracking, Breitensuche und iterierte Tiefensuche. Für alle drei Algorithmenentwurfstechniken geben wir hier die wesentlichen (abstrakten) Programmfragmente an, die Sie in einer Komplexitätsanalyse vergleichen sollen.
Problem *Binärbaumsuche*:

Gegeben: Ein beliebiger binärer Baum B mit $|B|$ vielen Knoten, die paarweise verschiedene Bewertungen der Art value(node) $\in \mathbb{N}$ besitzen, sowie eine Zahl $k \in \mathbb{N}$.

Frage: Kommt der Wert k bei einem Knoten von B als Bewertung vor?

Die verwendete Datenstruktur erlaubt folgende Zugriffe und Anweisungen:
`:=` für die Zuweisungen, `depth(B)` ist die Tiefe (auch als Höhe bezeichnet) und `root(B)` die Wurzel von B von B. Die Variable für Knoten ist `node` mit den Nachfolgern `successor` und dem Vorgänger `predecessor`. `level(i)` ergibt eine lineare, ungeordnete Liste der Knoten der Tiefe i in B mit der Komplexität $O(l)$, wenn diese Liste l Elemente enthält. Der Befehl `subtree(i)` erzeugt mit der Komplexität $O(t)$ den Teilbaum von B, der alle Knoten bis zur Tiefe i enthält, wobei t die Anzahl der Knoten dieses Teilbaums ist.

(a) *Tiefensuche mit back-tracking* geht zunächst in die Tiefe und wendet back-tracking an, sobald ein Teilbaum vollständig abgesucht wurde (pre-order traversal).

```
depth-first-search(tree: B)
begin
  node := root(B);
  LOOP: if value(node) = k
        then print('gefunden'); stop;
        else
          if node hat nicht besuchten successor
          then node := v.l.n.r. erster nicht besuchter successor
          else node := predecessor
          end-if
        end-if
    goto LOOP
 end
```

(b) *Breitensuche* arbeitet alle Knoten gleicher Tiefe ab, bevor weiter in die Tiefe gegangen wird (level-order traversal).

```
breadth-first-search(tree: B)
begin
  node := root(B);
  i := 0
  while not i > depth(B) do
    if Es gibt Knoten node aus level(i) mit value(node) = k
    then print('gefunden'); stop;
    else
      i := i+1
    end-if
  end-while
end
```

(c) *Iterierte Tiefensuche* geht in jedem Schritt nur bis zu einer gegebenen Tiefe vor. Die Suche beginnt mit der Tiefe Null (Wurzel!). Wenn sich das gesuchte Element nicht im Teilbaum der aktuellen Tiefenbeschränkung befindet, wird die Suchtiefe inkrementiert und erneut gesucht. Man beachte, dass die iterierte Tiefensuche bereits besuchte Knoten in der nächsten Iteration erneut bearbeitet, d.h. das Verfahren führt Überprüfungen überflüssiger Weise mehrfach durch.

```
iterate-depth-first-search(tree: B)
begin
  for i := 0 step 1 to depth(B) do
    if Es gibt Knoten node aus subtree(i) mit value(node) = k
    then print('gefunden'); stop;
    end-if
  end-for
end
```

Intuitiv ist klar, dass Breitensuche mehr Platz und iterierte Tiefensuche mehr Zeit braucht als Tiefensuche. Sei n die Tiefe, in der sich das gesuchte Element befindet. Es soll entschieden werden, ob der Mehrverbrauch für große n auch erheblich ist.

1. Geben Sie den Zeitbedarf $t(n)$ der drei Algorithmen an (gemessen in der Anzahl der bearbeiteten Knoten)! Je Algorithmus (2 Pkt.)
2. Geben Sie den Platzbedarf $s(n)$ der drei Algorithmen an (gemessen in der Anzahl aller benutzten, erzeugten und besuchten Knoten)! Je Algorithmus (2 Pkt.)

Es ist hilfreich, zunächst die Abhängigkeiten zwischen Tiefe, Anzahl aller Knoten (Blätter) in Binärbäumen (auch den nicht vollständigen) festzustellen und zu notieren.

Für den Beweis, dass die Zahl der Blätter eines Binärbaums immer um 1 größer ist als die Zahl seiner anderen, sogenannten inneren, Knoten erhalten Sie zusätzlich (2 Pkt.)

Bisher erreichbare Punktzahl: $\boxed{100}$