1 Rekursives Layout gewurzelter Bäume

Bei der Visualisierung von Graphen (z.B. Organigrammen, UML Diagrammen bei objekt-orientiertem Design, Präzedenzrelationen bei Scheduling-Problemen usw.) kann es oft mühsam sein, das Layout (sprich Knoten- und Kantenplazierung) „per Hand“ zu erstellen. Dies ist natürlich insbesondere dann der Fall, wenn sich die Graphen häufig ändern und/oder eine große Anzahl von Knoten und Kanten enthalten. In solchen Fällen sind Layout-Algorithmen sehr praktisch. Diese Algorithmen sollen einen Graphen klar und einfach in eine Ebene einbetten, so daß gewisse strukturelle Eigenschaften möglichst leicht erkannt werden können (z.B. Planarität, Hierarchiebenen usw.).

In diesem Skript wird ein einfacher Algorithmus vorgestellt, mit dem ein kompaktes hierarchisches Layout von einem gewurzelten Binärbaum erstellt werden kann. Dabei muß zu jedem Knoten nur die relative X-Position zu seinem Vorgänger ermittelt werden. Die Y-Position ist durch die Tiefe des Knotens im Baum vorgegeben (siehe auch Abbildung 1).

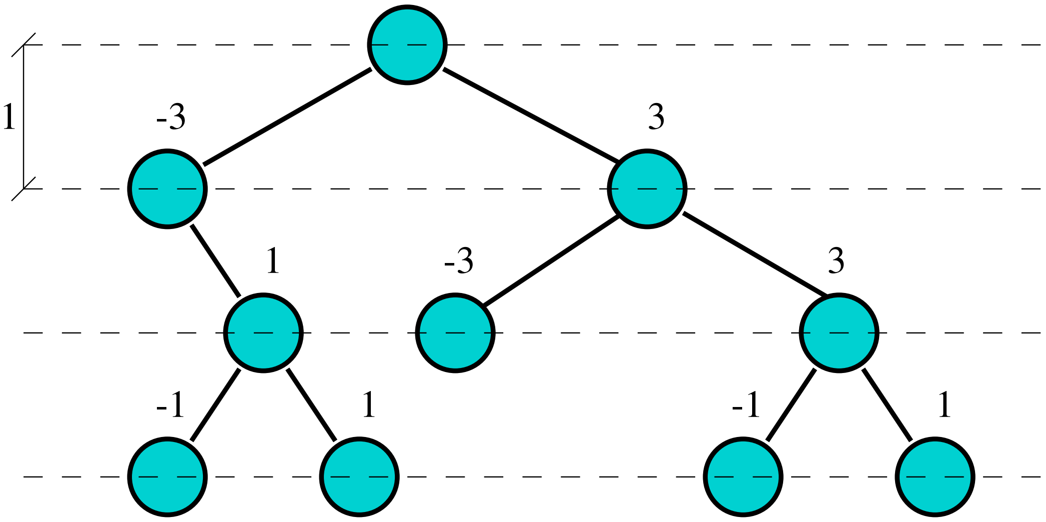

Abbildung 1: Beispiel für ein hierarchisches Layout eines Baumes

Bei dem Algorithmus wird an jedem Knoten (ausgehend von der Wurzel) rekursiv für beide Teilbäume das Layout berechnet, um daraus ein Gesamt-Layout (Knoten und beide Teilbäume) zu ermitteln. Die Funktion $layout(v, T)$ liefert als Ergebnis das Layout eines Baumes T mit Wurzel v folgendermaßen: an jedem Knoten $\neq v$ von T ist die relative Verschiebung zu dessen Vorgänger gespeichert. Zusätzlich werden zwei Listen, eine für die linke und eine für die rechte Kontur, zurückgegeben. Dabei sollten auch wieder jeweils nur die relativen Verschiebungen zur nächsthöheren Ebene gespeichert werden. Im folgenden sei d der erwünschte Mindestabstand zweier Knoten.

Bei der Berechnung von $layout(v, T)$ können folgende drei Fälle auftreten:

- T **besteht nur aus dem Blatt** v**:** es werden zwei leere Listen zurückgeliefert.
- v **hat nur einen Nachfolger** u**:** rekursiv $layout(u, T \setminus \{v\})$ aufrufen. Bei u als relative

X-Position z.B. $-d/2$ oder $d/2$ eintragen (oder 0, wenn v direkt über dem Rest platziert werden soll). Derselbe Wert wird an den Anfang der beiden Konturlisten eingefügt.

- v **hat zwei Nachfolger:** $layout$ rekursiv für beide Teilbäume aufrufen. Sei T' der linke und T'' der rechte Teilbaum. Es ist zu ermitteln wo sich die beiden Teilbäume „am nächsten“ kommen. Die beiden Teilbäume sollen so platziert werden, daß sie sich gerade nicht überlappen. Dazu wird die rechte Konturliste von T' und die linke von T'' simultan von oben durchlaufen, bis einer der beiden Teilbäume erschöpft ist. Bei jeder Ebene werden die aktuellen Auslenkungen $X_{T'}$ und $X_{T''}$, verglichen zur Wurzel des jeweiligen Baumes, aus den relativen X-Positionen in den Konturlisten ermittelt. Die kritische Ebene ist diejenige, bei der $X_{T'} + X_{T''}$ am größten ist.

  Damit zwischen den beiden Teilbäumen ein Abstand von mindestens d vorhanden ist, müssen die beiden Wurzeln die Distanz $l := X_{T'} + X_{T''} + d$ erhalten. Das erreicht man, indem man bei der Wurzel von T' $-l/2$ und bei der Wurzel von T'' $l/2$ einträgt.

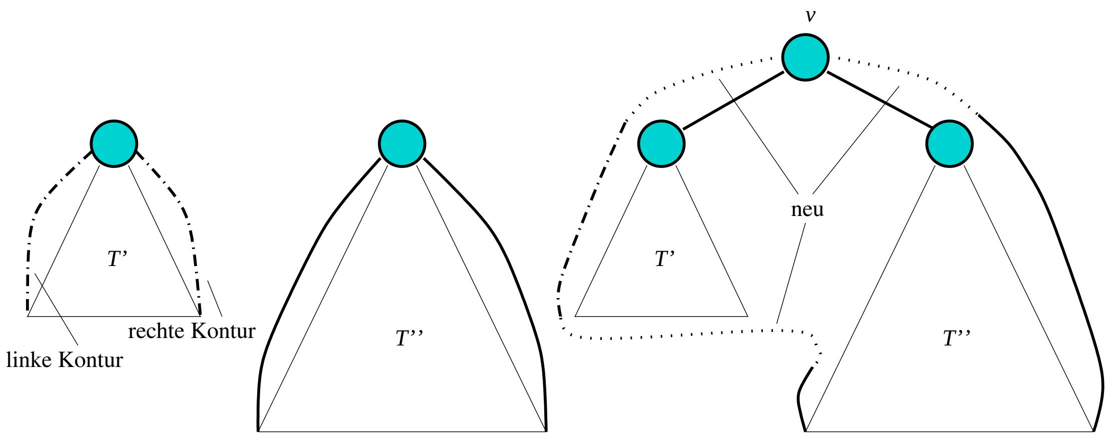

Abbildung 2: Zwei Teilbäume zusammenfügen

Nun sind noch die beiden Konturlisten für den gesamten Baum zu ermitteln (Abbildung 2 stellt dies schematisch dar). Um eine Laufzeit von $O(n)$ zu erhalten, ist es wichtig, daß prinzipiell die Listen des größeren Teilbaums übernommen werden. OBdA. sei T'' der größere Teilbaum. Bei der rechten Konturliste von T'' muß nur $l/2$ am Anfang eingefügt werden, ansonsten kann sie unverändert übernommen werden. Bei der linken Konturliste von T'' ist das Stück, das sich mit T' überlappt, durch T's linke Liste zu ersetzen. Dabei muß lediglich die relative Verschiebung der Ebene neu berechnet werden, bei der T' endet. Nach dieser Ersetzung muß noch $-l/2$ an den Anfang der Liste eingefügt werden.

Bei der Verwaltung der LEDA-Listen ist darauf zu achten, daß man bei der Parameterübergabe mit Referenzen oder Pointern arbeitet, damit diese nicht unnötig kopiert werden. Zum Zusammenfügen der Teillisten im dritten Fall sollte man die LEDA-Funktionen $conc(\ldots)$ und $split(\ldots)$ benutzen.

Die Knoten-Positionen können im Graphwin mit der Funktion $set\_position(\ldots)$ gesetzt werden. Nach dem Layouten sollte noch $place\_into\_win(\ldots)$ aufgerufen werden, damit der Graph schön zentriert dargestellt wird.

Literatur

[1] Battista G., Eades P., Tamassia R., Tollis I.: *Graph Drawing*, Prentice Hall, 1999.