

Der Algorithmus von Walker und seine Verbesserung

basierend auf einer Arbeit von
Christoph Buchheim, Michael Jünger und Sebastian Leipert [1]

Olaf Weinmann

Der im Jahre 1990 vorgestellte Algorithmus von Walker wird in vielen Bereichen genutzt, um Bäume von unbeschränktem Grad zu zeichnen. Bis zu diesem Zeitpunkt waren nur Algorithmen bekannt, die geordnete Binärbäume in Linearzeit zeichnen. Zu nennen ist dabei der Algorithmus von Wetherell und Shannon, der 1979 präsentiert wurde und bereits in der Lage war, die Ordnung und Hierarchie der Knoten zu visualisieren. Einen deutlich besseren Linearzeitalgorithmus präsentierten Reingold und Tilford 1981. Dieser Algorithmus basiert auf dem Algorithmus von Wetherell and Shannon, verbessert diesen jedoch in der Weise, dass strukturgleiche Unterbäume bis auf Verschiebung gleich gezeichnet werden und das Vorgehen des Algorithmus symmetrisch ist. Jedoch beschränkt sich auch die Funktionalität dieses Algorithmus auf Binärbäume. Erst neun Jahre später stellte Walker einen Algorithmus vor, der Bäume von unbeschränktem Grad zeichnen konnte. In seiner Arbeit „A node-positioning algorithm for general trees“ [2] behauptet Walker, dass dieser Algorithmus linear in der Anzahl der Knoten des zu zeichnenden Baumes arbeite. Es stellte sich jedoch heraus, dass der Algorithmus nicht in Linearzeit arbeitet. Die fehlerbehafteten Stellen werden im folgenden aufgezeigt und Korrekturvorschläge gegeben. Diese Ausarbeitung basiert auf der Arbeit „Improving Walker’s Algorithm to Run in Linear Time“ [1], welche 2002 von Christoph Buchheim, Michael Jünger und Sebastian Leipert vorgestellt wurde.

Inhaltsverzeichnis

1	Vorbemerkungen	3
2	Der Algorithmus von Reingold und Tilford	4
2.1	Beschreibung des Algorithmus	4
2.2	Besonderheiten	5
2.3	Beispiel	6
3	Der Algorithmus von Walker	7
3.1	Beschreibung des Algorithmus	8
3.2	Laufzeitprobleme	8
3.2.1	Erstes Problem: Das Durchlaufen der Konturen	8
3.2.2	Zweites Problem: Finden der Vorgänger und Aufsummieren der Modifier	9
3.2.3	Drittes Problem: Das Zählen und Verschieben der kleineren Unterbäume	9
4	Die Verbesserung von Walker's Algorithmus	10
4.1	Das Durchlaufen der Konturen und die Summation der Modifier	10
4.2	Das Finden der größten getrennten Vorgänger	11
4.3	Das Zählen der kleineren Unterbäume	11
4.4	Die Verschiebung der kleineren Unterbäume	12
5	Zusammenfassung	12

1 Vorbemerkungen

Wir wollen zunächst an einige Begriffe erinnern, die wir im folgenden immer wieder benötigen werden.

- (1) Unter einem *Baum* verstehen wir einen azyklischen ungerichteten Graphen mit einem ausgezeichneten Knoten, genannt Wurzel, so dass von diesem Knoten zu jedem weiteren Knoten genau ein Pfad existiert (Abbildung 1).
- (2) Ist v ein Knoten in einem Baum, so ist das *Level* von v die Länge des (eindeutigen) Pfades von der Wurzel zu v . Jeder von v verschiedene Knoten w auf diesem Pfad heißt *Vorgänger* von v . In diesem Zusammenhang heißt v *Nachfolger* von w .
- (3) Ist (v, w) eine Kante in einem Baum, so nennen wir v den *Vater(knoten)* von w und w das *Kind* von v . Sind w_1 und w_2 zwei verschiedene Kinder von v , so bezeichnen wir sie als *Geschwister*.
- (4) Ein Baum, dessen Knoten jeweils höchstens zwei Kinder besitzen, heißt *Binärbaum*.
- (5) Ein *geordneter* Baum ist ein Baum, welcher die Eigenschaft besitzt, dass die Kinder eines jeden Knoten v eine festgelegte Reihenfolge haben.
- (6) Es sei l eine natürliche Zahl und v ein Knoten in einem geordneten Baum. Weiter seien v_1 und v_2 Kinder von v so, dass v_1 entsprechend der Ordnung links von v_2 liegt. Ist dann w_1 der rechteste Nachfolger von v_1 im Level l und w_2 der linkeste Nachfolger von v_2 im Level l , dann bezeichnet man w_1 und w_2 als Nachbarn, falls alle durch die zwischen v_1 und v_2 liegenden Kinder induzierten Unterbäume von v keinen Knoten im Level l haben (Abbildung 1).
- (7) Ein Knoten im Baum, der keine Kinder besitzt, heißt *Blatt*.
- (8) *Spiegelung* eines Baumes bedeutet, dass für jeden Knoten v im Baum die Reihenfolge der Kinder invertiert wird.
- (9) Es seien v_- und v_+ zwei Knoten im Baum, so dass weder v_- Nachfolger von v_+ noch v_+ Nachfolger von v_- ist. Die *größten getrennten Vorgänger* w_+ und w_- sind dann durch folgende Eigenschaften charakterisiert
 - (i) w_+ ist Vorgänger von v_+ und w_- ist Vorgänger von v_- ,
 - (ii) w_+ und w_- sind Geschwister.

Wir bemerken, dass w_- und w_+ durch diese Eigenschaften eindeutig bestimmt sind. Siehe auch Abbildung 1.

- (10) Einen Baum in einer Ebene zu *zeichnen*, bedeutet

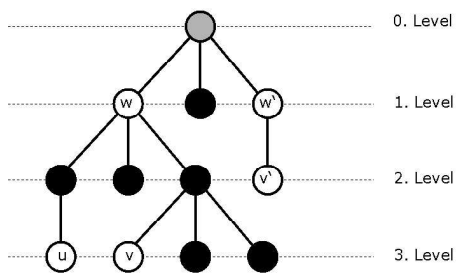


Abbildung 1: In dem links stehenden Baum sind u und v Nachbarn. Die Knoten w und w' sind die größten getrennten Vorgänger von v und v' . Der grau gefärbte Knoten ist die Wurzel des Baumes.

- (i) Jedem Knoten v wird ein Koordinatenpaar (x_v, y_v) zugewiesen.
- (ii) Eine Kante zwischen zwei Knoten v und w wird durch eine Strecke repräsentiert, die die Punkte (x_v, y_v) und (x_w, y_w) in der Ebene verbindet.

Es ist nun zu erwarten, dass, wenn ein Baum in eine Ebene gezeichnet wird, gewisse ästhetische Forderungen an das Layout gestellt werden. Man denke etwa daran, dass es ansonsten möglich wäre, alle Knoten an der selben Stelle (x, y) durch Punkte darzustellen. Der gezeichnete Baum wäre dann ein Punkt. Wir fordern folgende fünf Eigenschaften

- (A1) Die y -Koordinate eines Knotens entspricht dem Level des Knotens.
- (A2) Die Kanten kreuzen sich nicht und Knoten auf dem selben Level haben einen vorgegebenen horizontalen Minimalabstand.
- (A3) Unterbäume, die strukturgleich sind, werden bis auf Verschiebung gleich gezeichnet.
- (A4) Die Reihenfolge der Kinder eines Knotens wird in der Zeichnung dargestellt.
- (A5) Der Zeichenalgorithmus arbeitet symmetrisch, das heißt, dass die Zeichnung der Spiegelung des Baumes gleich der Spiegelung des gezeichneten Baumes ist.

2 Der Algorithmus von Reingold und Tilford

Der erste Linearzeitalgorithmus, welcher geordnete Binärbäume unter Berücksichtigung von (A1) – (A5) zeichnete, wurde 1981 von Reingold und Tilford vorgestellt. Wir wollen nun zunächst die rekursive Vorgehensweise des Algorithmus beschreiben, darauf auf die Besonderheiten in Bezug auf Laufzeit hinweisen und uns schließlich die Vorgehensweise des Algorithmus anhand eines Beispiels veranschaulichen.

2.1 Beschreibung des Algorithmus

Ein Blatt wird in der Weise gezeichnet, dass ihm eine vorläufige x -Koordinate und eine dem Level entsprechende y -Koordinate zugewiesen wird. Nachdem die Unterbäume, die durch die Kinder eines Knotens v induziert werden, gezeichnet wurden, wird der rechte

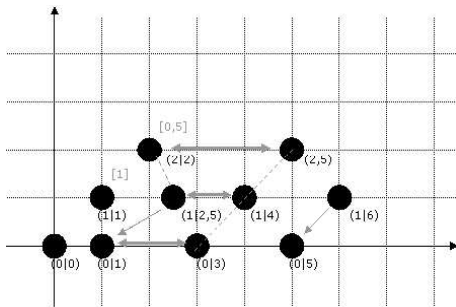


Abbildung 2: Beispiel für den Vergleich der Konturen zweier Bäume. Die Konturen sind durch gestrichelte Linien oder durch Pfeile gekennzeichnet. Die Pfeile innerhalb der Konturen stellen die Konturzeiger dar.

Unterbaum so verschoben, dass er so nahe wie möglich auf der rechten Seite des linken Unterbaumes platziert ist. Daraufhin wird v über seinen Kindern zentriert. Offensichtlich erfüllt dieser Algorithmus (A1) – (A5). Weniger offensichtlich ist aber, dass dieses Vorgehen tatsächlich in Linearzeit möglich ist. Wir wollen nun die in Bezug auf Laufzeit kritischen Stellen im Algorithmus eingehen, da wir die von Reingold und Tilford entwickelten Vorgehensweisen später zur Laufzeitverbesserung von Walker's Algorithmus benötigen.

2.2 Besonderheiten

Die in Bezug auf Laufzeit kritischen Stellen im Algorithmus sind offenbar:

- (i) Das Berechnen der neuen Position des rechten Unterbaumes der aktuellen Wurzel.
- (ii) Das Verschieben des rechten Unterbaumes der aktuellen Wurzel.

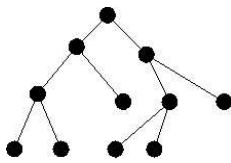
Zur Lösung des letzteren Problems führten Reingold und Tilford einerseits für jeden Knoten v die vorläufige horizontale Koordinate $prelim(v)$ und andererseits den Modifier $mod(v)$ ein. Soll nun der rechte Unterbaum von v verschoben werden, so speichern Reingold und Tilford zwei Größen $prelim(v)$ und $mod(v)$ die eine vorläufige Koordinate von v und den Wert dieser Verschiebung repräsentieren. Die aktuelle horizontale Position eines Knotens v ist demnach die vorläufige Koordinate $prelim(v)$ plus $modsum(v)$, wobei $modsum(v)$ die Summe aller Modifier auf dem Weg des Vaters von v bis zur aktuellen Wurzel ist. Damit ist es aber möglich in einem abschließenden Durchlauf des Baumes sämtliche Verschiebungen vorzunehmen.

Damit bleibt (i) zu diskutieren. Wir bemerken zunächst, dass wir, wenn wir die Position eines rechten Unterbaumes bestimmen wollen, nur die linke Kontur des rechten Unterbaumes mit der rechten Kontur des linken Unterbaumes vergleichen müssen (Abbildung 2). Damit stellt sich also bei (i) nur die Frage, wie die Konturen möglichst effizient durchlaufen werden können. Reingold und Tilford führen dazu so genannte „Threads“ ein. Für jedes Blatt im aktuellen Unterbaum, welches einen Nachfolger in der gleichen Kontur hat, ist der Thread ein Zeiger auf jenen Nachfolger. Ist nun ein beliebiger Knoten etwa auf der linken Kontur gegeben, so ist der Nachfolgeknoten in dieser Kontur, sofern existent, entweder durch sein linkstes Kind oder durch einen Thread gegeben und kann somit

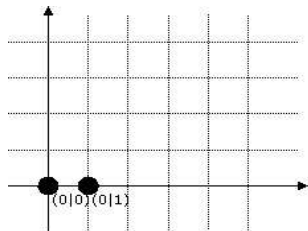
in konstanter Laufzeit zugegriffen werden. Es muss aber nun sichergestellt werden, dass man während des Durchlaufens der Konturen auch die derzeit aktuellen Positionen der Konturknoten bestimmen kann. Offenbar muss hier gesichert werden, dass man die Modifiersummen durch Aufaddieren der Modifier von Konturknoten berechnen kann. Hierzu werden Modifier von Blättern eingeführt. Ist also v ein Blatt mit einem Thread auf einen Knoten w , so wird in $mod(v)$ der Wert $modsum(w) - modsum(v)$ abgelegt. Damit ist nur noch zu überlegen, wie man die Threads und die Modifier von Blättern aktuell halten kann. Ein Thread muss nur gesetzt werden, wenn zwei Bäume unterschiedlicher Höhe zusammengefügt werden. Dieser kann aber bereits beim Vergleich der beiden Bäume in konstanter Laufzeit gesetzt werden. Ebenso können hier die Modifiersummen berechnet und damit die Modifier der Blätter aktualisiert werden.

2.3 Beispiel

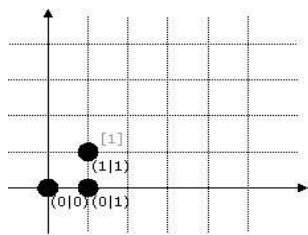
Wir wollen die Vorgehensweise des Algorithmus von Reingold und Tilford nocheinmal anhand eines Beispiels erläutern. Hierbei gehen wir stets davon aus, dass der rechte Unterbaum der aktuellen Wurzel bereits vollständig gezeichnet ist.



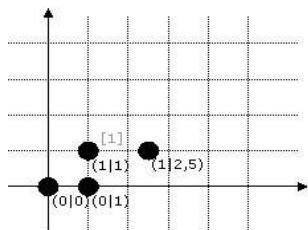
Wir wollen den links stehenden Graphen zeichnen. Dabei soll der horizontale Mindestabstand zweier benachbarter Knoten 2 sein.



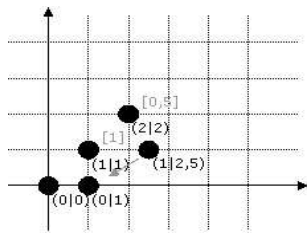
Es werden nun im ersten Schritt dem linken Blatt des Baumes und daraufhin dem rechten Unterbaum der aktuellen Wurzel vorläufige Koordinaten zugewiesen.



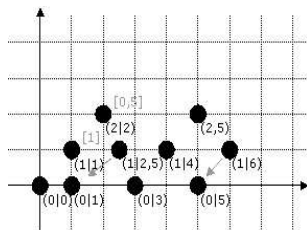
Darauf wird der rechte Unterbaum der aktuellen Wurzel nach rechts verschoben, also der Modifier der aktuellen Wurzel auf den Wert 1 gesetzt. Schließlich wird die aktuelle Wurzel über den aktuellen Positionen der Kinder zentriert. Die Werte in den Eckigen klammern stellen die Modifierwerte dar.



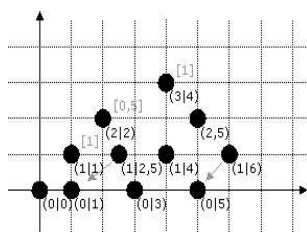
Es werden wieder dem rechten Unterbaum der aktuellen Wurzel vorläufige Koordinaten zugewiesen.



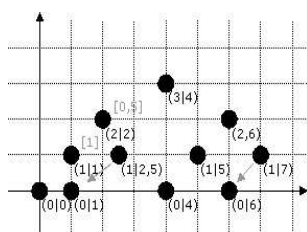
Der rechte Unterbaum der aktuellen Wurzel wird entsprechend verschoben, ein Konturzeiger gesetzt und die aktuelle Wurzel platziert.



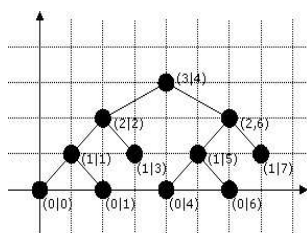
Es werden wieder dem rechten Unterbaum der aktuellen Wurzel vorläufige Koordinaten zugewiesen. Hier wurde davon ausgegangen, dass dieser rechte Unterbaum bereits vollständig gezeichnet wurde.



Der rechte Unterbaum der aktuellen Wurzel wird wieder entsprechend verschoben und die aktuelle Wurzel zentriert.



Es werden nun in dem abschließenden Durchlauf die Verschiebungen durch Berechnen der Modifiersummen vorgenommen. Links wurde bereits der rechte Unterbaum der Wurzel verschoben.



Nachdem alle Verschiebungen vorgenommen wurden, terminiert der Algorithmus und das Ergebnis sieht wie der links stehende Baum aus.

3 Der Algorithmus von Walker

Zunächst wollen wir darauf hinweisen, dass der Algorithmus von Reingold und Tilford ohne größere Schwierigkeiten so verändert werden kann, dass auch Bäume von unbeschränktem Grad unter Berücksichtigung von (A1) – (A5) gezeichnet werden können. Wir wollen die nötigen Modifikationen kurz beschreiben:

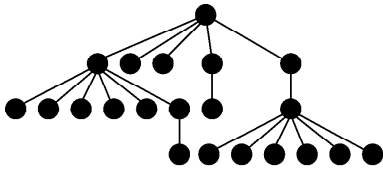


Abbildung 3: Die kleineren Unterbäume der Wurzel sind zu weit links angeordnet.

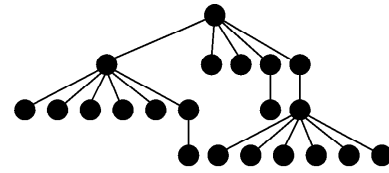


Abbildung 4: Die kleineren Unterbäume der Wurzel sind zu weit rechts angeordnet.

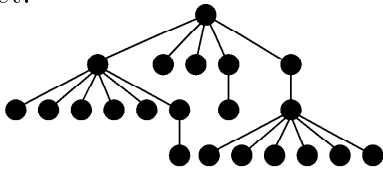


Abbildung 5: Die kleineren Unterbäume der Wurzel sind im mittleren Bereich konzentriert.

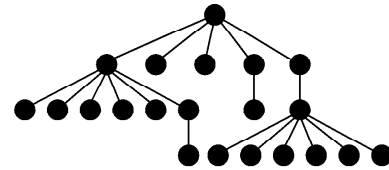


Abbildung 6: Walker's Algorithmus zieht die kleineren Unterbäume auseinander.

Zunächst bietet sich die Vorgehensweise an, dass die Kinder eines Knotens v von links nach rechts durchlaufen werden und die induzierten Unterbäume nacheinander platziert werden. Dabei wird aber die Forderung (A5), also die Symmetrie des Algorithmus verletzt, da kleinere Unterbäume möglicherweise zu weit links positioniert werden (Abbildung 3). Zur Behebung dieses Problems wird der Baum, ähnlich wie oben, ein zweites Mal durchlaufen, nur werden in diesem Durchlauf die Unterbäume eines Knotens von rechts nach links durchlaufen und platziert (Abbildung 4). Berechnet man nun Durchschnittswerte der horizontalen Koordinaten, welche bei den beiden Durchläufen errechnet wurden, so wird (A5) erfüllt.

Damit sind nun zwar die bisher geforderten ästhetischen Eigenschaften gegeben, jedoch werden bei dieser Vorgehensweise die kleineren Unterbäume zusammengestaucht (Abbildung 5). Der Algorithmus von Walker behebt nun auch dieses Problem (Abbildung 6).

3.1 Beschreibung des Algorithmus

Wie der Algorithmus von Reingold und Tilford geht auch Walker's Algorithmus rekursiv vor. Die Unterbäume der aktuellen Wurzel werden nacheinander von links nach rechts abgearbeitet. Wird ein Unterbaum an den linken Teilwald angefügt, so wird die Wurzel dieses Unterbaumes zunächst so nahe wie möglich rechts neben seinem linken Geschwisterknoten platziert. Dann wird wie beim Algorithmus von Reingold und Tilford die linke Kontur des anzufügenden Baumes durchlaufen und die Positionen der einzelnen Knoten mit den Positionen der linken Nachbarknoten verglichen. Werden nun zwei in Konflikt stehende Knoten v_- und v_+ gefunden, welche dazu führen, dass der anzufügende Unterbaum um s verschoben werden muss, so werden alle kleineren Unterbäume deren Wurzel zwischen den

größten getrennten Vorgängern von v_- und v_+ liegen, ebenfalls verschoben. Genauer: Es seien w_- und w_+ die größten getrennten Vorgänger von v_- und v_+ und w_1, \dots, w_n Kinder der aktuellen Wurzel zwischen w_- und w_+ . Dann wird der Knoten w_k um $\frac{k \cdot s}{n+1}$ nach rechts verschoben.

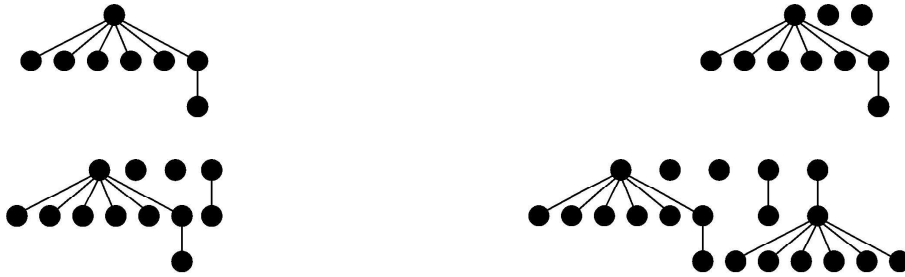


Abbildung 7: Die Vorgehensweise von Walkers Algorithmus.

3.2 Laufzeitprobleme

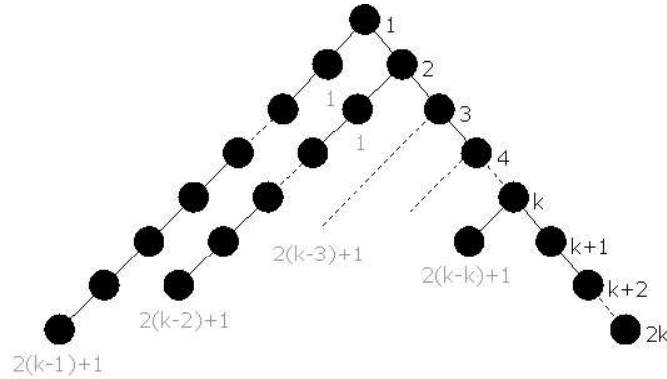
Der von Walker präsentierte Algorithmus arbeitet zwar korrekt, jedoch arbeiten verschiedene Teile des Algorithmus nicht in Linearzeit. Einige der Probleme können mit den Ideen von Reingold und Tilford ohne Schwierigkeiten behoben werden, andere Probleme erfordern komplett neue Ideen. Wir werden im folgenden die kritischen Stellen des Algorithmus beschreiben und die Laufzeit jener Stellen analysieren. Im nächsten Abschnitt werden wir Verbesserungsvorschläge geben, die dann lineare Laufzeit sichern werden.

3.2.1 Erstes Problem: Das Durchlaufen der Konturen

Walker verwendet obgleich die Strategie von Reingold und Tilford zum Durchlaufen der Konturen schon bekannt war eine rekursive Funktion „GETLEFTMOST“, welche in einem gegebenen Level l den linken Nachfolger eines Knotens v bestimmt. GETLEFTMOST geht dabei folgendermaßen vor: Ist das Level von v bereits l , so wird v zurückgegeben. Andernfalls wird GETLEFTMOST auf alle Kinder von v von links nach rechts angewandt. Die Gesamtlaufzeit von GETLEFTMOST ist nun im allgemeinen nicht linear. Um dies zu zeigen geben wir eine Folge von Bäumen T_k an, die folgende Eigenschaften erfüllen soll

- (i) Die Knotenzahl n_k des Baumes T_k liegt in $\Theta(k^2)$;
- (ii) Die Anzahl der GETLEFTMOST-Aufrufe in T_k liegt in $\Theta(k^3)$.

Wir definieren T_k wie folgt: Die Wurzel sei der erste Knoten einer Kette von $2k$ Knoten. Für $i \in \{1, \dots, k\}$ hat der i -te Knoten dieser Kette einen linken Nachfolgeknoten. Dieses Kind ist der erste Knoten einer Kette mit $2(k-i)+1$ Knoten (Abbildung 8). Die Anzahl

Abbildung 8: Konstruktion des Baumes T_k für beliebiges k .

der Knoten in T_k ist damit offensichtlich

$$2k + \sum_{i=1}^k (2(k-i) + 1) = 2k + k(k-1) + k \in \Theta(k^2)$$

Es müssen nun für $i = 0 \dots k-1$ beim Besuchen des Knotens auf der rechten Kontur von T_k im Level i zwei Unterbäume zusammengefügt werden. Nach Konstruktion des Baumes T_k muss aber GETLEFTMOST auf alle Knoten im rechten Unterbaum angewandt werden. Die Anzahl dieser Knoten ist

$$\begin{aligned} \#(T_{k-i-1}) + 1 &= 3(k-i-1) + (k-i-1)(k-i-2) + 1 \\ &= (k-i-1)(3+k-i-2) + 1 \\ &= (k-i-1)(k-i) + (k-i-1) + 1 = (k-i)^2 \end{aligned}$$

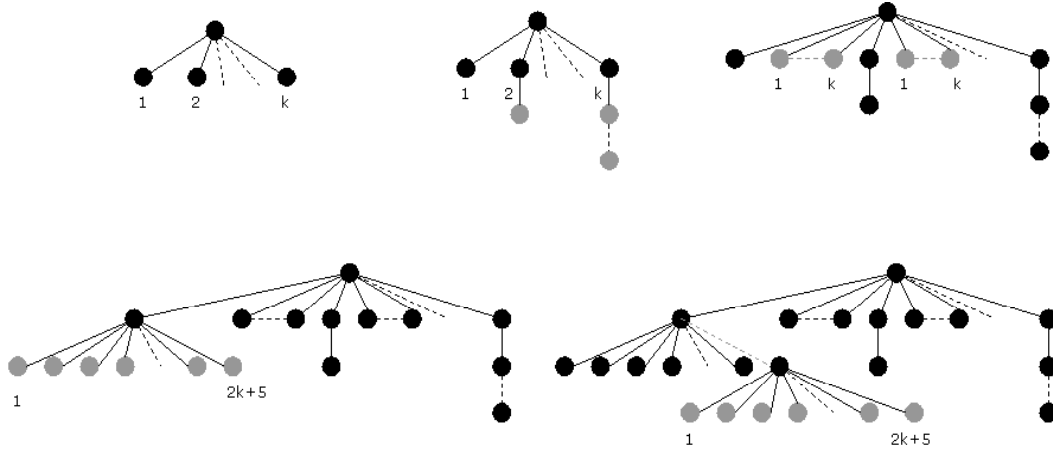
dabei sei für einen Baum T die Anzahl seiner Knoten durch $\#(T)$ beschrieben. Damit ist die Gesamtanzahl an GETLEFTMOST-Aufrufen größer als

$$\sum_{i=0}^{k-1} (k-i)^2 \in \Theta(k^3)$$

Wenn wir nun $k \in \Theta(n^{1/2})$ wählen, so ist die Anzahl der Knoten von T_k in $\Theta(n)$ aber die Anzahl der Aufrufe von GETLEFTMOST ist in $\Omega(n^{3/2})$.

3.2.2 Zweites Problem: Finden der Vorgänger und Aufsummieren der Modifier

Wird ein Teilbaum der aktuellen Wurzel an den linken Teilwald angepasst, so werden die größten getrennten Vorgänger für alle möglicherweise in Konflikt stehenden Nachbarn berechnet. Dies wird durch Durchlaufen des Weges von dem aktuellen Knoten bis zur aktuellen Wurzel bewerkstelligt. Zur gleichen Zeit werden die Modifiersummen berechnet. Dieses Vorgehen ist offensichtlich nicht linear.

Abbildung 9: Konstruktion des Baumes T^k für beliebiges k .

3.2.3 Drittes Problem: Das Zählen und Verschieben der kleineren Unterbäume

Wird ein Unterbaum der aktuellen Wurzel wegen eines Konflikts zum linken Teilwald nach rechts verschoben, so verschiebt eine Prozedur APPORTION alle (kleineren) Unterbäume, die zwischen den in Konflikt stehenden Unterbäumen liegen. Die Anzahl dieser kleineren Unterbäume wird dabei durch Zählen der einzelnen bestimmt. Beide Aktionen haben eine Gesamtlaufzeit in $\Omega(n^{3/2})$. Wir geben eine Folge von Bäumen T^k an, die dies beweist. Sei v die Wurzel des Baumes T^k . An v werden k Kinder angehängt. Für $i = 1 \dots k$ wird an das i -te Kind eine Kette von i Knoten angehängt. Zwischen jedes Paar der Kinder der Wurzel werden nun wieder k Kinder eingefügt. Das linkeste Kind der Wurzel hat $2k + 5$ Kinder. Bis einschließlich dem Level $k - 1$ hat jedes rechteste Kind der $2k + 5$ Kinder wieder $2k + 5$ Kinder (Abbildung 9). Die Anzahl der Knoten in T^k ist

$$1 + \sum_{i=1}^k i + (k-1)k + (k-1)(2k+5) \in \Theta(k^2)$$

nach Konstruktion des linken Unterbaumes ergibt sich für $i = 2 \dots k$ beim Anfügen der i -ten „Unterbaumkette“ ein Konflikt mit dem linken Unterbaum der Wurzel im Level i . Damit werden alle $(i-1)(k+1) - 1$ kleineren Unterbäume, die zwischen den in Konflikt stehenden Unterbäumen liegen, gezählt und verschoben. Die Anzahl dieser Arbeitsschritte ist

$$\sum_{i=2}^k ((i-1)(k+1) - 1) = \frac{(k+1)k(k-1)}{2} - k + 1 \in \Theta(k^3)$$

Und wir erkennen wie im letzten Beispiel, dass auch hier keine lineare Laufzeit vorliegt.

4 Die Verbesserung von Walker's Algorithmus

Wir wollen nun erklären, wie man den Algorithmus von Walker so verbessern kann, dass er tatsächlich in Linearzeit läuft.

4.1 Das Durchlaufen der Konturen und die Summation der Modifier

Hier können wir genau wie beim Algorithmus von Reingold und Tilford vorgehen. Zu beachten ist nur, dass meist ein linker Teilwald und dessen rechte Kontur betrachtet werden muss. In dieser Situation können aber genau wie in der „Binärbaumsituation“ Threads gesetzt werden.

4.2 Das Finden der größten getrennten Vorgänger

Dieses Problem kann mit dem Algorithmus von Schieber und Vishkin gelöst werden. Im Einzelnen können die größten getrennten Vorgänger w_- und w_+ eines jeden Knotenpaares v_- und v_+ in konstanter Laufzeit gefunden werden, nachdem eine Linearzeitprozedur ausgeführt wurde. Für unsere Anwendung gibt es aber einen wesentlich einfacheren Algorithmus. Wir bemerken, dass wir w_+ bereits kennen - es ist die Wurzel des aktuellen Teilbaumes, der angefügt werden soll. w_- hängt nun nur von v_- ab und wir bezeichnen ihn deshalb als „Vorgänger“ von v_- . Wir benutzen Knotenzeiger $\text{ancestor}(x)$ für jeden Knoten x um seinen Vorgänger zu speichern. Zu Anfang wird für jeden Knoten x der Vorgänger $\text{ancestor}(x)$ mit x initialisiert. defaultAncestor sei ein weiterer Knotenzeiger. Unser Ziel ist es nun, dass nach jedem Zusammensetzungsschritt die folgende Aussage richtig ist.

Für jeden Knoten x auf der rechten Kontur des linken Teilwaldes gilt: Falls $\text{ancestor}(x)$ up to date ist, also auf ein Kind der aktuellen Wurzel zeigt, so zeigt er auf w_- ; Ansonsten zeigt defaultAncestor auf w_- .

Damit stellt sich die Frage, wie wir die Richtigkeit dieser Aussage nach jedem Arbeitsschritt sichern können. Es sei v unsere aktuelle Wurzel. Es werden bekanntlich die Unterbäume links beginnend platziert. Sei w die Wurzel des linkesten Unterbaumes. Nachdem dieser Baum angefügt wurde, wird defaultAncestor auf w gesetzt. Wird nun ein weiterer Unterbaum von v etwa mit Wurzel w' angefügt, so sind zwei Fälle zu unterscheiden. Ist der Baum mit Wurzel w' kleiner als der linke Teilwald, so wird für alle Knoten x auf der rechten Kontur dieses Baumes $\text{ancestor}(x) := w'$ gesetzt. Ansonsten wird defaultAncestor aktualisiert (Abbildung 10).

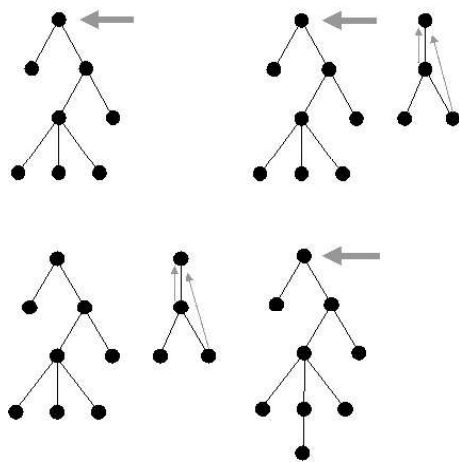


Abbildung 10: Verhalten der Pointer ancestor und defaultAncestor beim Anfügen von Unterbäumen verschiedener Größen. Ein fett gezeichneter grauer Pfeil stellt jeweils den defaultAncestor dar. Die verbleibenden Pfeile stellen jeweils einen ancestor dar.

4.3 Das Zählen der kleineren Unterbäume

Die Behebung dieses Problems ist einfach. Wir durchlaufen den gesamten Baum zu Anfang und nummerieren sämtliche Kinder eines Knotens von links nach rechts. Damit kann später bei bekannten größten getrennten Vorgängern die Anzahl der dazwischenliegenden Knoten sofort berechnet werden.

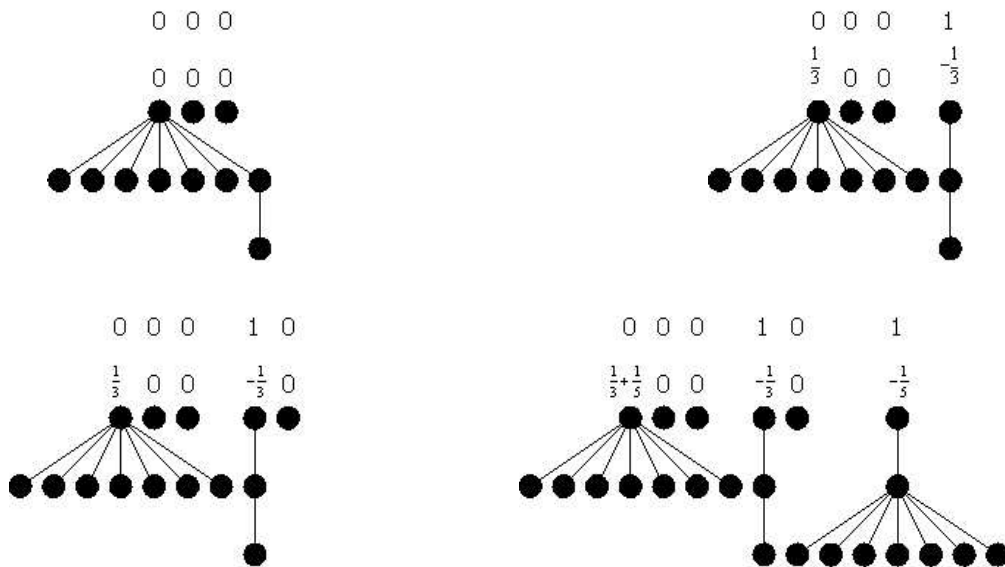
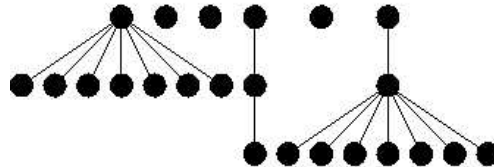


Abbildung 11: Zusammenfassung der Verschiebungen.

4.4 Die Verschiebung der kleineren Unterbäume

Um hier lineare Laufzeit zu erzielen, wird der Algorithmus so verändert, dass jeder beliebige Teilbaum höchstens zweimal verschoben wird. Wir führen für jeden Knoten x die Werte $shift(x)$ und $change(x)$ ein und initialisieren sie mit 0. Es sei der Baum, mit Wur-

0	0	0	1	0	1
$\frac{1}{3} + \frac{1}{5}$	0	0	$-\frac{1}{3}$	0	$-\frac{1}{5}$
0	$\frac{1}{3} + \frac{1}{5}$	$\frac{2}{3} + \frac{2}{5}$	$\frac{3}{5}$	$\frac{4}{5}$	0



zel w_+ der aktuell anzufügende Unterbaum und w_- die Wurzel desjenigen Unterbaumes des linken Teilwaldes, mit dem ein Konflikt entsteht. Damit muss der Unterbaum mit Wurzel w_+ um einen Wert s nach rechts verschoben werden. Es sei $subtrees$ die Zahl der Unterbäume zwischen w_- und w_+ plus eins. Nach Walker's Idee muss der i -te dieser Unterbäume um den Wert $i \cdot s / subtrees$ nach rechts verschoben werden. Wir speichern dies, indem wir $shift(w_+) := shift(w_+) + s$ und $change(w_+) := change(w_+) - s / subtrees$ setzen. Weiter wird $change(w_-) := change(w_-) + s / subtrees$ gesetzt. Damit können nun sämtliche Verschiebungen in einem abschließenden Durchlauf der Kinder der aktuellen Wurzel von rechts nach links ausgeführt werden. Dazu werden zwei Hilfsvariablen $shift$ und $change$ zunächst mit 0 initialisiert. Wird hierbei etwa das Kind v besucht, so wird v um $shift$ nach rechts verschoben. Daraufhin wird $change := change + change(v)$ und $shift := shift(v) + change$ gesetzt und mit dem linken Geschwisterknoten gleich verfahren.

5 Zusammenfassung

Einige fehlerbehaftete Teile des Algorithmus von Walker können mit Hilfe der Ideen, die von Reingold und Tilford entwickelt wurden so verändert werden, dass sie tatsächlich in Linearzeit arbeiten. Eine neue Vorgehensweise wird beim Finden der größten getrennten Vorgänger entwickelt. Hierbei wird die Tatsache verwendet, dass zu jedem Zeitpunkt ein größter getrennter Vorgänger des betrachteten Knotens bereits bekannt ist. Deshalb ist es hier nicht notwendig, auf den wesentlich komplizierteren Algorithmus von Schieber und Vishkin zurückzugreifen. Schließlich wird die Verschiebung der kleineren Unterbäume so vorgenommen, dass jeder beliebige Teilbaum höchstens zweimal verschoben wird.

Literatur

- [1] Buchheim, Jünger, Leipert: *Improving Walker's Algorithm to Run in Linear Time*, Proc. GD'02, Springer LNCS 2528, pp. 344-353
- [2] J. Walker II. *A node-positioning algorithm for general trees*. Software - Practice and Experience, 20(7):685-705, 1990