PQ-Bäume

Seminar über *Graphenalgorithmen*
Sommersemester 2002, FU Berlin
Vortrag von Günter Rote am 30. April 2002

Literatur: Kellogg S. Booth, George S. Lueker: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences* **13** (1976), 335–379.

Problem: Gegeben ist eine Grundmenge $U = \{u_1, \ldots, u_n\}$ und eine Familie von Teilmengen $S_1, \ldots, S_m$ von U. Gibt es eine Permutation von U, sodass für jede Menge S_j die Elemente von S_j nebeneinander liegen? (*consecutive ones property.*) Bestimme alle derartigen Permutationen.

Satz. Mit PQ-Bäumen kann dieses Problem in $O(n + \sum_{j=1}^{m} |S_j|)$ Zeit und $O(n)$ Speicher gelöst werden.

Anwendungen. Archäologie (Datierung von Gräbern), Bioinformatik (Sequenzierung von DNA)

PQ-Bäume. Die Elemente von U sind in den *Blättern* gespeichert. Die inneren Knoten sind entweder als P-Knoten oder Q-Knoten gekennzeichnet.

Ein PQ-Baum repräsentiert eine *Menge von Permutationen* von U: Bei einem P-Knoten kann man die Kinder beliebig vertauschen. Bei einem Q-Knoten kann man die Reihenfolge der Kinder umkehren. Jede Reihenfolge der Blätter bei einem rekursiven Durchlaufen des Baumes ist eine Permutation, die durch den PQ-Baum repräsentiert wird.

Bei einem Knoten vom Grad 2 ist kein Unterschied zwischen P- und Q-Knoten, und Knoten vom Grad 1 sind überflüssig.

Inkrementelle Lösung des Problems. Gegeben ist ein PQ-Baum T und eine Menge S. Das Ergebnis soll ein PQ-Baum T' sein, der genau die Permutationen repräsentiert, bei denen zusätzlich auch alle Elemente von S nebeneinander liegen, oder die Ausgabe, dass die Eingabe widersprüchlich ist und dass es keine solche Permutation gibt.

Algorithmus.

1. Markieren der Knoten als *voll*, *leer* oder *halbvoll*, je nachdem, welcher Anteil der Blätter im Unterbaum zu S gehört.

2. Umbauen des Baumes, von den Blättern zur Wurzel (bottom-up). Dabei wird jeweils ein Knoten mit allen seinen Kindern betrachtet. Die Kinder sind bereits umgebaut. Der Knoten wird nun nach gewissen Regeln umgebaut, je nachdem, ob es ein P- oder Q-Knoten ist, welche seiner Kinder voll, leer, oder halbvoll

sind und in welcher Reihenfolge diese Kinder (im Fall eines Q-Knotens) auftreten, und ob es Elemente von S auch außerhalb des gerade betrachteten Teilbaums gibt oder nicht.

Ein Knoten mit mehr als zwei halbvollen Kindern führt zum Beispiel direkt zu einem Widerspruch. Es gibt etwa ein Dutzend verschiedene Regeln.

Implementierung. Das rekursive Durchlaufen des Baumes und das Anwenden der Transformationsregeln lässt sich leicht in $O(n)$ Zeit (Größe des Baumes) durchführen. Die Herausforderung besteht darin, es in $O(|S|)$ Zeit (amortisiert) zu schaffen. Man muss vermeiden, zu viele leere Knoten zu besuchen (zum Beispiel alle leeren Kinder eines halbvollen Knotens). Dies gelingt durch sorgfältige Planung der Datenstruktur. Insbesondere in folgenden Punkten weicht die Implementierung von der naheliegen Implementierung ab:

- Der Algorithmus läuft in zwei Phasen ab. In der ersten Phase werden nur die betroffenen Knoten gefunden; erst in der zweiten Phase findet der Umbau statt.
- Die Kinder eines Knotens sind als doppelt-verkettete Liste organisiert, aber bei einem Q-Knoten haben nur die beiden äußersten Kinder Elternzeiger.
- Damit man nicht zu viele Elternzeiger von leeren Knoten ändern muss, werden in gewissen Fällen solche Elternknoten vertauscht.

Weitere Anwendungen

- Erkennen von Intervallgraphen.

  Ein Intervallgraph ist der Durchschnittsgraph einer Menge von Intervallen. Jeder Knoten i ist durch ein Intervall $[a_i, b_i]$ repräsentiert, und zwei Knoten sind benachbart, wenn die entsprechenden Intervalle einen nichtleeren Durchschnitt haben.

- Planaritätstest (effiziente Implementierung eines Algorithmus, der auf Lempel, Even und Cederbaum (1966) zurückgeht).

  Der PQ-Baum verwaltet dabei eine Menge von Kanten des Graphen. Im Algorithmus werden auch Teilmengen von Kanten im Baum gegen andere Kantenmengen ausgetauscht.