Nicole Kaiser fI Tutorium: Do. 12-14 Uhr fI Tutor: Martin Klossek

20.11.2001

Theoretische Informatik 1, WS 2001/2002

Übungsblatt 5

Keine Garantie für die Korrektheit der hier vorgestellten Lösungen.

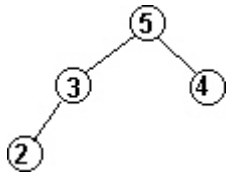
Aufgabe 5.1

(a.i)

Sei die p die kleinste Priorität.
Angenommen p sei kein Blatt, dann ist p ein Knoten. Ein Knoten hat mindestens ein Kind. Das Kind eines Knotens hat in einem Baum mit Heap-Ordnung immer eine kleinere Priorität als der Knoten selbst. Damit gäbe es also ein Element, dass kleiner als p. Das ist aber ein Widerspruch zur Behauptung, dass p die kleinste Priorität besitzt.

(a.ii)

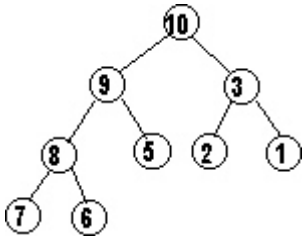
Gegenbeispiel:



Die zweitkleinste Priorität ist hier 3.
Die Tiefe dieses Baumes ist t = 2.
Die Bedingungen für Heap-Struktur sind erfüllt (vgl. Skript S. 55):
(a)√, (b)√, (c)√
Der Baum besitzt auch Heap-Ordnung.

(a.iii)

Gegenbeispiel:



Die drittgrößte Priorität ist 8 und kein Kind der Wurzel.
Die Bedingungen für Heap-Struktur und Heap-Ordnung sind auch erfüllt.

(b)

Wenn für einen Baum B die Anzahl der Knoten mindestens $2^{t+1}-2$ ist, dann ist laut Vorlesung die Tiefe des Baumes mindestens t.
Sei $p(B_l)$ die Position des linkesten Blattes in B und $p(B_r)$ die Position des rechtesten Blattes in B.
Laut Vorlesung liegen die Blätter an den Positionen $\left\lfloor \frac{n}{2} \right\rfloor + 1$ bis n . Also gilt :

$$p(B_l) = 2^t \quad \text{und} \quad p(B_r) = 2^{t+1} - 2$$

Behauptung:
Die t-größte Priorität wird in der Tiefe $\leq$ t-1 gespeichert.
Beweis:
Angenommen die t-größte Priorität wird in Tiefe > t -1 gespeichert. Dann wird die größte Priorität (Wurzel) in der Tiefe 1 gespeichert. Das ist aber ein Widerspruch, da nach Definition 2.3c (Skript) die Wurzel die Tiefe 0 hat.
Damit ist die Annahme falsch und die Behauptung bewiesen.

Bei mindestens $2^{t+1}-2$ Knoten gibt es höchstens einen Knoten der nur ein Kind hat. Das ist der rechteste Knoten in der Tiefe t-1. Da er aber mindestens ein Kind hat, ist er Vater und damit kein Blatt. Also liegen alle Blätter in der Tiefe t.
Der rechteste Knoten in Tiefe t-1 hat dann die Position $2^t - 1$. Weil aber die t-größte Priorität in einer Tiefe

$\leq$ t-1 gespeichert wird, kann diese Priorität nicht von einem Blatt gespeichert werden.

(c)

Idee:

Beginnend bei dem ersten Element des Heaps (die Wurzel in der Binärbaumdarstellung), wird geprüft, ob die im Heap gespeicherten Prioritäten größer als das übergebene Argument x sind. Zuerst wird Testen() für den Vater aufgerufen. Wenn es ein linkes Kind gibt, wird Testen() rekursiv für dieses aufgerufen. Analog mit dem rechten Kind, falls es existiert.

Für eine Position j im Array wird Testen() nicht mehr aufgerufen, wenn

- $j \geq n$ ist, also nicht mehr im Array liegt oder
- $j < x$ ist.

Das Programm terminiert, wenn für jeden rekursiven Aufruf, einer der o.g. Fälle aufgetreten ist.

Wenn alle Rekursionen abgeschlossen sind, wurden alle Prioritäten $\geq$ x ausgeben.

Algorithmus:

```
Prioritaet_groesser_x (int x) {
        i := 1;
        n := Anzahl Prioritäten im Heap;

        Testen (i ) {
                if (H[i] ≥ x) {
                        ausgabe H[i];
                        if (2i ≤ n)
                                Testen(2i);                    // linkes Kind
                        if (2i + 1 ≤ n)
                                Testen(2i + 1);                // rechtes Kind
                }
        }
}
```

Laufzeit:

Sei K die Anzahl der Prioritäten, die ausgegeben werden.

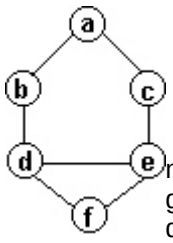
Für jede Priorität die ausgegeben wird, wird Testen() einmal aufgerufen → O(K)
Innerhalb von Testen() wird für jede ausgegebene Priorität geprüft, ob deren Kinder noch im Heap-Array sind. (Wenn die Kinder nicht im Heap sind, dann hat dieser Knoten keine Kinder.) Zugriff auf Elemente eines Arrays → O(1)
Bei weniger als diese K Prioritäten werden alle deren Kinder (maximal 2) mit Testen() überprüft, aber nicht ausgegeben. →O(2K)

Insgesamt ist die Laufzeit: O(K).

Aufgabe 5.2

(a)

Abarbeitung des Beispiels mittels des gegebenen Algorithmus:



W = $\emptyset$, V = {a,b,c,d,e,f}
w := a, entferne a,b,c aus V: V := {d,e,f}, W = {a}
w := e, entferne e,f,d aus V: V := $\emptyset$, W = {a,e}

Der Algorithmus liefert eine unabhängige Menge mit 2 Elementen.
Wenn man sich diesen Graphen genau ansieht, erkennt man dass die unabhängige Menge größter Mächtigkeit W‘ = {b,c,f} ist. Also liefert die Heuristik des minimalen Graphens in diesem Fall nicht die Menge mit der größten Mächtigkeit.