Studienarbeit Nr. 1580 – Nachtrag: Heapsort-Varianten in Theorie und Praxis

Stefan Lewandowski

14. Januar 1997

3.3 Optimal-Heapsort nach Gu & Zhu

Der Absatz über das *best-case*-Verhalten ist durch folgenden Text zu ersetzen:

Der triviale *best-case* von $\frac{1}{2}n\lg n + O(n)$ Vergleichen kann analog zu den *worst-case*-Überlegungen bei Bottom-Up-Heapsort praktisch sicher nicht erreicht werden. Möglich erscheinen hingegen Heaps, die $\frac{3}{4}n\lg n + o(n\lg n)$ Vergleiche benötigen. Dazu müßten die *leichten* Elemente jeweils in der oberen Hälfte des Heaps bleiben und die *schweren* weitmöglichst oben. Das Bottom-Up-Einfügen darf dabei jeweils nur konstant viel Vergleiche benötigen. Dies ist möglich, da durch spätere `reheap`-Aufrufe die Elemente weiter nach oben rücken können und so wieder „günstige" Einsinkpositionen frei werden.

3.4 Bottom-Up-Heapsort nach Diekert

3.4.1 Worst-Case

Die Abschätzung für die *leichten* Elemente war leicht fehlerhaft – die Auswirkungen sind jedoch gering, so daß sie sich in den Abbildungen nicht bemerkbar machen und die qualitativen Aussagen zur Praxis korrekt bleiben.

Somit beträgt die Anzahl der Vergleiche in der ersten Iteration

$$\begin{aligned}
\ldots &= \frac{3}{2}n + \sum_{j=0}^{\lg h - 3} \frac{n}{16} \cdot \frac{1}{2^{2^j}}(j+4) \\
&\overset{(6),(8)}{<} \frac{3}{2}n + \frac{n}{16} \cdot \left[\underbrace{\frac{1}{2} \cdot \frac{0.9143 + 0.6329}{2}}_{\sum „j"} + \underbrace{4 \cdot \left(0.5 + \frac{0.6329}{2}\right)}_{\sum „4"} \right] \\
&< 1.7283n
\end{aligned}$$

Iteriert man die Abschätzung für alle Ebenen, ergibt sich für die Anzahl der Vergleiche

$$C(n) \quad \leq \quad \frac{1}{4}n\,(\lg\lg n)^2 + \frac{3}{4}n\lg\lg n + 3.9566n$$

Zusammen mit den $2.2033n$ Vergleichen für den Heap-Aufbau und $n\lg n - 2n$ Vergleichen für die Bestimmung der *potentiellen Einsinkpfade* folgt

Satz 3.9 *Der* worst-case *bei Bottom-Up-Heapsort nach Diekert ist durch*

$$n\left(\lg n + \frac{1}{4}(\lg\lg n)^2 + \frac{3}{4}\lg\lg n + 4.1599\right)$$

Vergleiche beschränkt.

B Der Code

Die Implementierung der Klasse `path_first` fehlte:

path1st.cc

```
// Datei: path1st.cc
// Autor: SL
// Stand: 05/12/96
//
// Basisklasse aller Bottom-Up-Varianten

#include "ooheap.h"

//------------------------------
void path_first::reheap(int i, int n)
//------------------------------
// laesst Element i im Heap der Groesse n einsinken
// vertauscht nicht mit Einzelkindern!
//------------------------------
{
  cyclic_swap(i,
              elem_search(i,
                          leaf_search(i,n)));
} // END path_first::reheap

//------------------------------
int path_first::leaf_search(int i, int n)
//------------------------------
// sucht den Pfad der leichteren Soehne
// liefert Blatt dieses Pfades
//------------------------------
```

```
{
  int path=i;
  while (path<=((n-1)>>1)) // Knoten hat noch 2 Kinder
    {
      path=(counting,heap[(path<<1)]>heap[(path<<1)+1]) ? (path<<1) : (path<<1)+1;
    }
  return path;
}

//------------------------------
void path_first::cyclic_swap(int i, int j)
//------------------------------
// vertauscht Elemente auf Pfad (i,...,j) zyklisch
//------------------------------
{
  int temp=heap[i], temp2;
  while (j>i)
    {
      temp2=heap[j];
      heap[j]=temp;
      temp=temp2;
      j>>=1;
    }
  heap[i]=temp;
}

//EOF
```