Thomas Kruczyk
Gruppe DO4

ALP III – 4.Übung
Musterlösung
Tutor: Till Zoppke

Aufgabe 28:

Mit Endrekursion:

```
void zugroß(int i) // a[i] ist möglicherweise größer als seine Nachfolger.
{   int kleinsterNachfolger;
        if (2*i+1 <= n) {
           if (a[2*i]<a[2*i+1]) kleinsterNachfolger = 2*i;
           else                 kleinsterNachfolger = 2*i+1;
        }
        else if (2*i <= n) kleinsterNachfolger = 2*i;
        else return;
        if (a[i] > a[kleinsterNachfolger]) {
           vertausche(i, kleinsterNachfolger);
           zugroß(kleinsterNachfolger);              //Endrekursion
        }

}
```

Ohne Endrekursion:

1.Schritt:
„setzt ein goto“

```
void zugroß(int i) // a[i] ist möglicherweise größer als seine Nachfolger.
{   int kleinsterNachfolger;
        if (2*i+1 <= n) {
           if (a[2*i]<a[2*i+1]) kleinsterNachfolger = 2*i;
           else                 kleinsterNachfolger = 2*i+1;
        }
        else if (2*i <= n) kleinsterNachfolger = 2*i;
        else return;
        if (a[i] > a[kleinsterNachfolger]) {
           vertausche(i, kleinsterNachfolger);
           i := kleinsterNachfolger;
           goto Anfang;                              // goto gesetzt
        }

}
```

2.Schritt:

„setze eine while-Schleife, um die Endrekursion zu eliminieren“

```
void zugroß(int i)                    // a[i] ist möglicherweise größer als seine Nachfolger.
{   int kleinsterNachfolger;

        while (i<n)                   //while-Schleife um Endrekursion zu eliminieren
        {
           if (2*i+1 <= n) {
              if (a[2*i]<a[2*i+1]) kleinsterNachfolger = 2*i;
              else                 kleinsterNachfolger = 2*i+1;
           }
           else if (2*i <= n) kleinsterNachfolger = 2*i;
           else return;
           if (a[i] > a[kleinsterNachfolger]) {
              vertausche(i, kleinsterNachfolger);
              i = kleinsterNachfolger;          //Endrekursion gekillt
           }else break;
        }
}
```

➔ Veränderter Programmcode ist mit ***rot*** markiert!!!!

Backtracking → Prinzip

Was ist eigentlich Backtracking?

Backtracking arbeitet nach dem Verfahren, dass man einen Algorithmus so lange ausführt, bis man an eine Grenze stößt, wo es nicht mehr weitergeht. Wenn jenes der Fall ist, kehrt man zum letzten Schritt zurück und testet einen anderen Folgeschritt.

Kurzgefasst arbeitet Backtracking folgende Schritte ab:

1. Versuche, eine gültige Teillösung auf dem Weg zum Ergebnis zu finden
2. Baue den restlichen Weg zum Ziel auf den Teillösungen auf
3. Ist dies nicht möglich, gehe einen Schritt zurück und versuche, eine andere Teillösung zu finden.

Das berühmte 8-Damen-Problem:

Bei 8-Damen-Problem geht es darum, genau 8 Damen auf einem Schachbrett so anzuordnen, dass sie sich nicht gegenseitig schlagen können. Um solch eine Lösung zu finden bietet es sich geradezu an Backtracking zu verwenden.
Man geht also wieder wie folgt vor: Setze die erste Damen. Versuche die nächsten Damen so zu setzen, dass sie in keinen Konflikt mit einer anderen Dame gerät. Falls dies nicht möglich ist, gehe ein Schritt zurück und versuch die zuletzt gesetzte Dame auf eine andere Position zu setzen, wo sie natürlich auch nicht in einen Konflikt mit einer anderen Dame geriet. ……usw.