Info2 Uebung vom 21.5.02

Gruppe F, Thomas Briner, http://n.ethz.ch/student/brinerth/info2

20. Mai 2002

1 Nachbesprechung Uebung 4

1.1 Heap mit k-närer Baumstruktur, beginnend mit Index 1 für die Wurzel

f(i):= Index des ersten Kindes (am meisten links) des Knotens mit Index i

- erstes Kind der Wurzel hat immer Index 2: f(1)=2
- wenn f(i)=j, dann ist f(i+1)=j+k, weil im Ganzen k Kinder am Knoten i dranhängen ⇒Teleskopieren

Anmerkung: beginnend mit Index 0 für die Wurzel:

1.2 mögliche Implementation des Heaps als ADT (nicht generisch!)

```
MODULE heap;        (* fm 02  *)

CONST heaplen = 65; (* 64 + 1 *)

TYPE Heap* = ARRAY heaplen OF INTEGER;
(* das Element 0 ist der Counter -> da steht drin, wieviele Elemente es im Heap hat *)

PROCEDURE sift(VAR H: Heap; i: INTEGER);
        VAR j,t: INTEGER;
BEGIN
        WHILE 2*i <= H[0] DO
                j:=2*i;
                IF j<H[0] THEN
                        IF H[j] > H[j+1] THEN
                                j := j+1;
                        END; (* j zeigt auf kleineren Bruder *)
                END;
                IF H[i]>H[j] THEN (* falls Parent groesser -> vertauschen *)
                        t := H[i];
                        H[i] := H[j];
                        H[j] := t;
                        i := j;
                ELSE
                        i := H[0];
                END;
        END;
END sift;

PROCEDURE init*(VAR H: Heap);
BEGIN
        H[0] := 0;
END init;

PROCEDURE insert*(VAR H: Heap; x: INTEGER):BOOLEAN;
        VAR t,i: INTEGER;done:BOOLEAN;
BEGIN
        IF H[0] < heaplen-1 THEN
                INC(H[0]);
                H[H[0]] := x;
                i := H[0];
                WHILE (i DIV 2>0)&(H[i DIV 2]>H[i]) DO
                        t := H[i];
                        H[i] := H[i DIV 2];
                        H[i DIV 2] := t;
                        i := i DIV 2;
                END;
                done:=TRUE;
        ELSE
                done:=FALSE;
        END;
        RETURN done;
END insert;
```

```
PROCEDURE min*(VAR H: Heap;VAR t:INTEGER):BOOLEAN;
        VAR done:BOOLEAN;
BEGIN
        IF H[0]>0 THEN
                t := H[1];
                H[1] := H[H[0]];
                DEC(H[0]);
                sift(H,1);
                done:=TRUE;
        ELSE
                done:=FALSE;
        END;
        RETURN done;
END min;

PROCEDURE count*(H: Heap):INTEGER;
BEGIN
        RETURN H[0];
END count;

END heap.
```

1.3 i-t kleinstes Element finden

1.3.1 mit Heapsort

Mittels der Prozedur min i Mal das Minimum herausholen ⇒i * siftDown aufrufen ⇒$O(n*log(n))$

1.3.2 mit modifiziertem Quicksort

Grundidee: nach jedem Durchgang drei Möglichkeiten

- i ist gerade der Ort, wo die beiden Scanlinien stehen bleiben ⇒Algorithmus abbrechen, i-tes Element gefunden
- i ist kleiner ⇒nur die linke Hälfte weiter bearbeiten
- i ist grösser ⇒nur die rechte Hälfte weiter bearbeiten

Analyse:

- best case: Die beiden Scanlinien bleiben beim ersten Durchgang genau am richtigen Ort stehen ⇒~n Vergleiche
- worst case: Das Array ist sortiert und wir erwischen als Pivotelement immer solche, die nur ein Element abschneiden ⇒ $O(n^2)$
- average case: Das Array wird bei jedem Schritt ungefähr in der Mitte geteilt, bis nur noch ein Element übrig ist und das ist dann das gesuchte.

$$T(1) = 1;$$

$$T(n) = T(n/2) + n$$

Teleskopieren:

2 Vorbesprechung Uebung 6

2.1 Fibonacci-Zahlen

$$Fib(i) := Fib(i-1) + Fib(i-2)$$

0	1	2	3	4	5	6	7	8	9
0	1								

Die Fibonacci-Zahlen findest du übrigens nicht nur in deinem Informatikstudium sondern zum Beispiel auch in der Struktur eines Tannenzapfens, auf der Ananas, im Hauptbahnhof und eben auch im Zusammenhang mit AVL-Bäumen!

zu 6.1b) Konstruiere den AVL-Baum mit möglichst wenigen Knoten rekursiv, d.h. überlege dir, wie du aus Bäumen mit minimaler Knotenzahl der Höhe i zu einem Baum der Höhe i+1 kommst und wieso der nun auch die minimale Knotenzahl haben muss.

2.2 Implementation AVL

2.3 Liniensegmentproblem

(Algorithmen und Datenstrukturen, S. 428ff)

Das ist die verallgemeinerte Version des Liniensegmentproblems nur mit horizontalen und senkrechten Linien.

Wir lösen hier nur das Schnittpunkttest-problem, d.h. wir wollen wissen:

Gibt es mindestens einen Schnittpunkt zwischen zwei Linien - ja oder nein?

Sobald wir also einen Schnittpunkt gefunden haben, bricht der Algorithmus ab und gibt TRUE zurück.

Der Algorithmus geht folgendermassen vor:

- Zuerst werden die Liniensegmentanfangs- und -endpunkte in x Richtung sortiert.
- Dann wird von links begonnen
- Es wird immer eine Liste geführt, in der alle Linien, die in diesem x-Bereich liegen, in der richtigen Reihenfolge gemäss y-Richtung enthalten sind.
- Es gibt zwei mögliche Ereignisse:
  - Eine Linie beginnt: nun wird sie am richtigen Platz in der Liste eingetragen. Dann wird die neu eingetragene Linie und ihr Nachbar oben und unten auf ihrer gesamten Länge auf einen Schnittpunkt überprüft.
  - Eine Linie endet: Sie wird aus der Liste entfernt. Am Platz, wo so sie vorher stand, sind nun zwei neue Nachbarn entstanden. Diese beiden Linien werden auf ihrer gesamten Länge auf einen Schnittpunkt überprüft.

Beispiel: