Info2 Uebung vom 10.6.02

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

17. Juni 2002

1 Nachbesprechung U8

1.1 Dynamisch:

1.1.1 Pathfinder:

- Anzahl Wege: R oder U, $2 * (n-1)$ Entscheidungen, 14 tief 7
- Alle Wege besser als bestimmte Schranke: Mischung aus dyn Prog und rekursiv

1.1.2 Summe:

Effizienz für 4 7 9 2 5 1 9 = 1000

- rekursiv: $2^6 = 64$
- dynamisch: nur für Initialisieren: 7000

2 Vorbesprechung U10

Benötigte Operationen: lookup, insert, delete

2.1 Move-to-root - Bäume

analog zu den selbstanordnenden Listen, nach jeder lookup-Operation ist Knoten in Wurzel

- lookup: # Rotationen O(n)
- insert: O(n)
- delete: O(n)

2.2 Splay-Trees

kleine Veränderung gegenüber Move-to-root - grosse Wirkung im Bezug auf Laufzeit

anstatt normale Rotationen:

```
WHILE (Tiefe des Knotens >1) DO
        IF (Weg gerade) THEN
                zuerst obere Achse, dann untere Achse drehen
                (* alle 3 Knoten zusammen drehen *)
        ELSE
                zuerst untere Achse, dann obere Achse drehen
```



```
                END;
        END;
        IF (Tiefe des Knotens =1) THEN
                Einfachrotation mit Wurzel
        END;
```

lookup(t,x):

- falls x in t: x ist nachher in Wurzel
- falls x nicht in t: letzter Elterknoten auf der Suche nach x ist in Wurzel (eigentlich symm Vorgänger/Nachfolger)

insert(t,x):

```
lookup(t,x.key);
        IF (t.root.key#x.key) THEN(* falls schon vorhanden: nix tun *)
                t1:=root.left; t2:=root.right;
                IF (t.root.key<x.key) THEN
                        x.left:=t.root;
                        x.right:=t2;
                        t.root.right:=NIL;
                        t.root:=x;
                ELSE
                        umgekehrt
                END;
        END;
```

delete(t,x):

```
lookup(t,x.key);
        IF (t.root.key=x.key) THEN
                t1:=root.left; t2:=root.right;
                lookup(t1,MAX(INTEGER));
                t1.right:=t2;
                t.root:=t1;
        END;
```

Weil lookup amortisiert in O(log n):

- lookup O(log n)
- insert O(log n)
- delete O(log n)

2.3 B-Baum

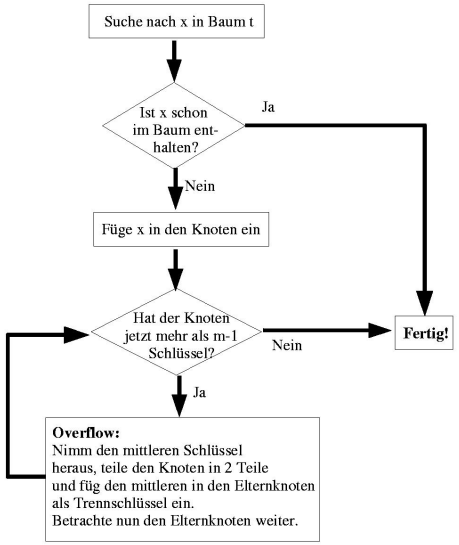
Mehrere Schlüssel pro Knoten -> mehr als 2 Kinder möglich
Alle Blätter auf gleicher Höhe, einfügen neuer Knoten immer nur nach oben
Ordnung m = maximale Anzahl Kinder, Minimum = $\lceil m/2 \rceil$

lookup(t,x): genau wie in normalem Binärbaum, keine Rotationen

insert(t,x):

```
füge in t x im richtigen Blatt ein, falls noch nicht vorhanden
falls Blatt jetzt zu gross (mehr als m-1 Schlüssel):
        nimm alle Schlüssel dieses Blattes und such den mittleren.
        Teile den Knoten und füg den mittleren in den Elterknoten ein.
        Falls dort ein overflow entsteht muss dieser gleich behandelt werden,
        bis man bei der Wurzel ankommt.
        Falls dort ein overflow entsteht, wächst der Baum um 1 nach oben.
```

Insert(x,t):



delete(t,x):

```
such x im Baum
        Falls x kein Blatt ist, dann such den symmetrischen Vorgänger und
        vertausche ihn  mit x

        (* precond: x ist im Blatt *)
        lösche x.
```



```
Falls Knoten nun zuwenig Schlüssel hat(weniger als [m/2])
        Falls ein Nachbarknoten mehr als [m/2]-1 Schlüssel hat
                Nimm die beiden Knoten und den Trennschlüssel im
                Elternknoten zusammen, such den mittleren Schlüssel,
                teil die beiden Knoten auf und füge den mittleren Schlüssel
                dem Elternknoten hinzu.
        Sonst (Falls beide Nachbarknoten auch nur [m/2]-1 Schlüssel haben)
                Nimm die beiden Knoten und den Trennschlüssel im
                Elternknoten zu einem neuen Knoten zusammen.
                Der Elternknoten hat nun einen Schlüssel weniger.
                Falls es deshalb einen underflow gibt, muss dieser
                gleich behandelt werden.
                Falls der underflow bei der Wurzel ist,
                        schrumpft der Baum um 1 in der Höhe
```

Delete(x,t):