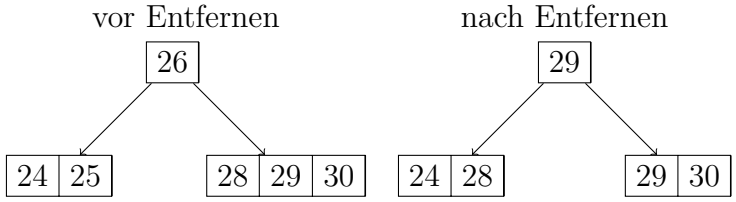
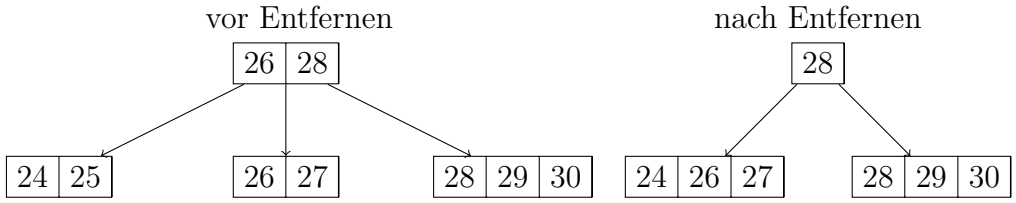
Aufgabe 1

Stellen Sie sich vor, eine Datenbankseite kann höchstens vier Werte vom Typ `INTEGER` *enthalten. Konstruieren Sie B^+-Bäume der Ordnung $d = 2$, so dass sich folgende Szenarien einstellen:*

1. *Einen B^+-Baum, in welchem das Löschen des Schlüssels 25 zu einer Umverteilung (*redistribution*) auf Blattebene führt. Zeigen Sie die Struktur des Baumes vor und nach dem Entfernen des Schlüssels.*



2. *Einen B^+-Baum, in welchem das Löschen des Schlüssels 25 zu einer Verschmelzung (*merge*) von zwei Knoten auf Blattebene führt, aber ohne die Höhe des Baumes zu ändern.*



Aufgabe 2

1. *Betrachten Sie für diese Aufgabe zunächst den k-d Baum in Abbildung 1. Auf welche Schlüsselwerte (Knoten und Blätter) der Baumes muss zugegriffen werden, um die folgenden Queries zu beantworten?*

   (a) `salary = 120`

   $$(140, 85), (100, 50), (60, 40), (110, 70), (120, 50)$$

   (b) `age = 50`

   $$(140, 85), (100, 50), (110, 70), (75, 45), (120, 50), (350, 45), (275, 50), (270, 60)$$

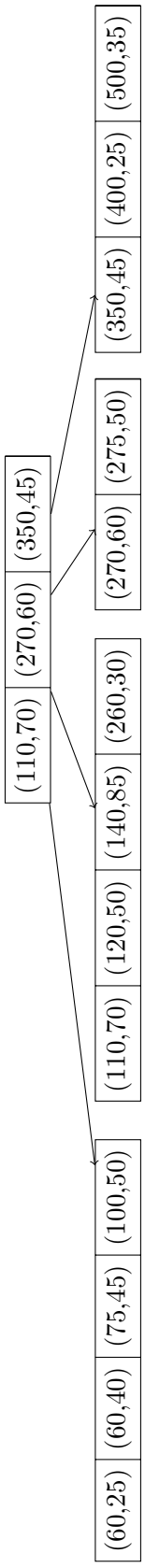
   (c) `salary <= 300 AND age <= 40`

   $$(140, 85), (100, 50), (60, 40), (60, 25), (350, 45), (400, 25), (260, 30)$$

2. *Eine alternative Strategie um mehrere Attribute in einem Schlüssel zusammenzufassen sind B^+-Bäume mit* composite search-keys*. Konstruieren Sie einen B^+-Baum mit einem solchen zusammengesetzten Schlüssel* `<salary,age>` *und $d = 2$ aus den Schlüsselwerten des k-d Baumes in Abbildung 1. Sortieren Sie dazu die Tupel entsprechend und bauen Sie den Baum mit Hilfe der* bulk loading *Methode auf.*
   Der *composite search-key* entspricht der Verknüpfung der 10-Bit langen Binärdarstellungen von `salary` und `age`:

<salary, age>	Verknüpfung		
	binär		dezimal
(60, 25)	0000 1111 00	00 0001 1001	61465
(60, 40)	0000 1111 00	00 0010 1000	61480
(75, 45)	0001 0010 11	00 0010 1101	76845
(100, 50)	0001 1001 00	00 0011 0010	102450
(110, 70)	0001 1011 10	00 0100 0110	112710
(120, 50)	0001 1110 00	00 0011 0010	122930
(140, 85)	0010 0011 00	00 0101 0101	143445
(260, 30)	0100 0001 00	00 0001 1110	266270
(270, 60)	0100 0011 10	00 0011 1100	276540
(275, 50)	0100 0100 11	00 0011 0010	281650
(350, 45)	0101 0111 10	00 0010 1101	358445
(400, 25)	0110 0100 00	00 0001 1001	409625
(500, 35)	0111 1101 00	00 0010 0011	512035

Mit `bulk loading` aufgebauter B^+-Baum:



3. *Auf welche Schlüsselwerte des B^+-Baumes aus Nr. 2 muss nun zugegriffen werden, um die Queries aus Nr.1 mit dessen Hilfe zu beantworten?*

   (a) `salary = 120`
   $(110, 70), (120, 50), (140, 85)^*$

   (b) `age = 50`
   $(110, 70), (60, 25), (60, 40), (75, 45), (100, 50),$
   $(120, 50), (140, 85), (260, 30), (270, 60), (275, 50)$
   $(350, 45), (400, 25), (500, 35)$

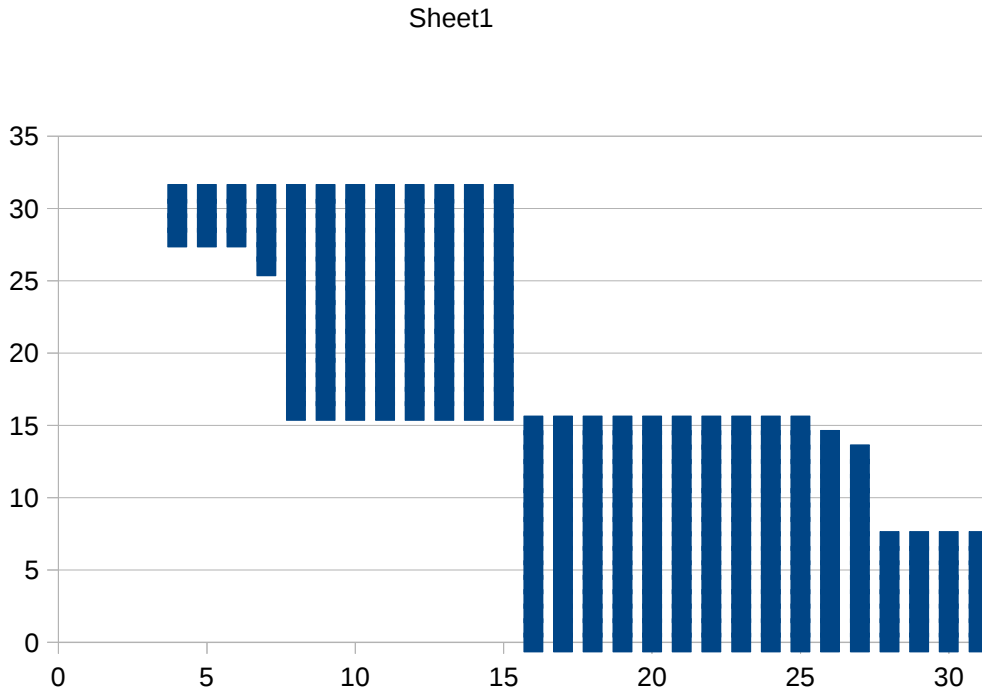
   (c) `salary <= 300 AND age <= 40`
   $(110, 70), (60, 25), (60, 40), (75, 45), (100, 50),$
   $(120, 50), (140, 85), (260, 30), (270, 60), (275, 50)$
   $(350, 45)^*$

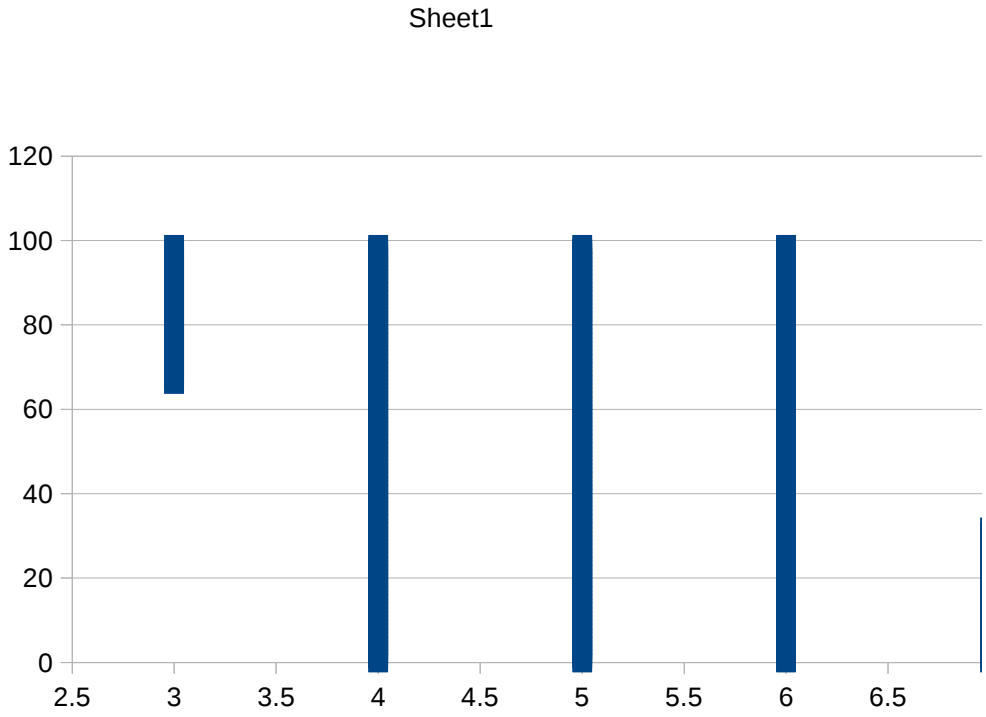
   $*$ Markieren das Ende der Suche, da außerhalb des Prädikats

Aufgabe 3

```
1  -- Repraesentiert Punkte in einem 2-dim Raum.
2  DROP TABLE points;
3  CREATE TABLE IF NOT EXISTS points(
4    x int,
5    y int
6  );
7  -- Belege alle moeglichen Punkte im Bereich x=[0,99], y=[0,99]
8  INSERT INTO points(x, y) (
9    SELECT s1, s2
10   FROM   GENERATE_SERIES(0,99) s1,
11          GENERATE_SERIES(0,99) s2
12 );
13
14 -- Gegebene Funktion: Abbildung 2, Aufgabenblatt
15 CREATE OR REPLACE FUNCTION point_to_z(x int, y int)
16 RETURNS int AS $$
17 DECLARE
18 z bit varying := '';
19 BEGIN
20 FOR i IN 0..6 LOOP
21 z := (x >> i) :: bit(1) || (y >> i) :: bit(1) || z;
22 END LOOP;
23 RETURN z :: bit(14) :: int;
24 END
25 $$ LANGUAGE PLPGSQL IMMUTABLE;
26 --
27
28 -- Erzeuge Z-Code Index
29 CREATE INDEX zindex ON points USING btree (point_to_z(x,y));
30 -- Erzeuge composite search-key Index: points <x,y>
31 CREATE INDEX cindex ON points USING btree (x,y);
32
33 --
34 CREATE EXTENSION IF NOT EXISTS pageinspect;
35
36
37 -- Z-Code Index von Punkten der Page 2
38 COPY (
39   SELECT      ctid, x, y
40   FROM        bt_page_items('zindex',2) i
41   INNER JOIN points p ON (i.ctid = p.ctid)
42   ORDER BY    ctid
```

```
43 ) TO '/tmp/zindex.csv';
44
45 -- Composite search-key Index von Punkten der Page 2
46 COPY (
47   SELECT      ctid, x, y
48   FROM        bt_page_items('cindex',2) i
49   INNER JOIN points p ON (i.ctid = p.ctid)
50   ORDER BY    ctid
51 ) TO '/tmp/cindex.csv';
```


Sheet1
35
30
25
20
15
10
5
0
0
5
10
15
20
25
30


Sheet1
120
100
80
60
40
20
0
2.5
3
3.5
4
4.5
5
5.5
6
6.5

Beim C-Index sind keine Zusammenhaenge ersichtlich.

Der Z-Index hingegen zeigt die Punkte auf einer in der Z Anordnung (verbindet man die Punkte zu einer Linie), da hier nach beiden Koordinaten sortiert wird.