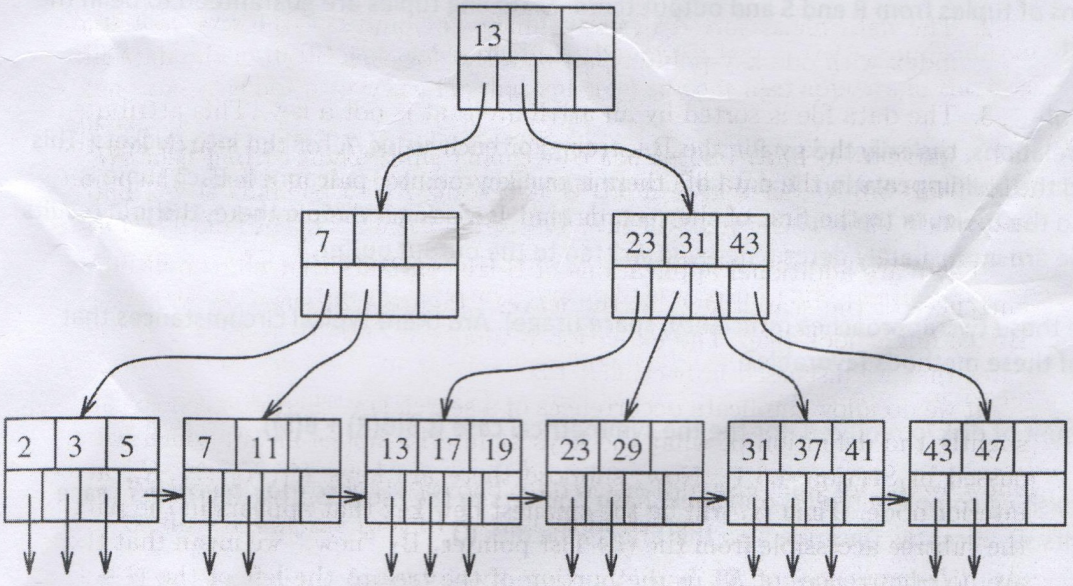
# Examination Queries & Retrieval

December 12, 2013
9:00 – 10:45 BBL 075

- You may use a cheat sheet (A4)
- Mention your name and student number on every answer sheet
- Write and formulate clearly
- Always explain your answers
- If you are running out of time, do not panic, but try to give as many answers as you can
- Good luck

## [1] B-trees

Below, you see a B+-tree from the course book.




1. Describe how the tree looks after insertion of node 18, i.e. make a draw of the part of the tree that has changed.
2. Suppose someone has the following claim. *B-trees should be fat, but according to the current rules, nodes may be only half filled. You could increase performance and decrease space usage by requiring a higher minimal node filling rate, e.g. 2/3 or 3/4.* Discuss this claim. Is it feasible? If so, sketch the main technical issues. Does it make sense?

**[2] Map Reduce**

Our input is a large file containing pairs of integer values `<g,val>`. The `g` is an integer used to identify groups. The value `val` represents an positive integer. We are interested in groups having a summation of the values below a (very low) threshold `T`. So the output should consist of pairs of the group id's with the summation of the corresponding values.

In SQL, our query would be:

```
SELECT g, SUM(val)    FROM InputFile
GROUP BY g    HAVING SUM (val) <= T
```

Give the (pseudo)code for a M/R program solving this problem. Performance is critical.

**[3] hash-join**

We distinguish two variants of the hash-join method to process natural joins.
Suppose we have tables R and S with A as join attribute.

*Symmetrical*
We have buckets that may contain tuples both from R and S. Based on the value of A, all tuples of both R and S are hashed into the right bucket. Finally, we determine for each bucket the valid join-combinations of tuples from R and S and output them. Matching tuples are guaranteed to be in the same bucket.

*Asymmetrical*
One of the relations, typically the smallest, is, according to the value of A, hashed into buckets. This one is called the building relation. The other one (the probing relation) is scanned. Each tuple is compared to the tuples in the appropriate bucket. Instead of storing the tuple there, the join results for this tuple are immediately determined and written to the output buffer.

[i] Compare these two approaches (efficiency, space usage). Are there typical circumstances that make one of these methods favorable?

[ii] The amount of disk io required for the the symmetrical case is $3(B(R) + B(S))$.

Instead of hashing complete tuples, one could hash pointers to the records. This decreases space requirements significantly. Describe how this affects the disk IO.

[iii] The approach described in [ii] is not considered in classical query processing. Explain why. Could you think of circumstances that might overcome the problems with this approach?