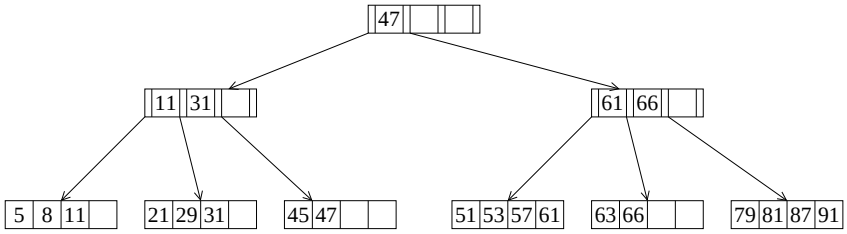
Data structures, exercise 4, 15.-19.3.

This week there are several alternative questions. You can pick freely what you do. The maximum amount of crosses is 6.

1. Read the B+-tree material from address
   `www.cs.helsinki.fi/u/mluukkai/tirak2010/b-tree.pdf`

2. Show how keys 33, 25, 19, 43 and 49 are added to below B^+-tree.



3. In an initially empty B^+-tree the following keys are added: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12 (in the listed order). Show how operations proceed. The maximum size of a node is $2t = 4$.

4. Show what happens when keys 5, 8 and 47 are deleted from the B^+-tree of question 2.

5. Delete 45 from the three that results in question 4.

6. In the presentation given of B^+-trees, the maximum number of children to internal nodes is the same as the maximum number of keys in the leaves, but this could be different. Alternatively, we could make the size of each node (internal nodes and leaves) as large as we can fit into one block of disk memory. Further assume that

   - one block is 4 kilobytes
   - one key takes 8 bytes
   - each child pointer takes 4 bytes
   - the data associated with each key takes 64 bytes.

   If you create a B^+-tree of 100 million elements, what is the largest possible height of the tree? If one disk read takes 10 milliseconds on the average, then what is the average time to access a key, assuming the root node is stored in main memory and any processing of the data in the main memory takes negligible time?

   What would be the largest height and the average search time for an AVL tree with 100 million elements? (The AVL tree cannot take advantage of the block structure of disk memory.)

7. **Rotation-game:** 3 crosses pseudo code, 6 crosses implementation. You may limit your implementation to cover e.g. only cases 3×3 and 4×4

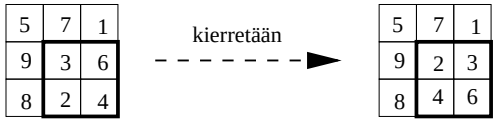
   - game is played on $n \times n$-table
   - each square has a number from interval $1 \ldots n \times n$
   - and each number is in the table only once
   - at start the numbers are on the table at random order, e.g.

5	7	1
9	3	6
8	2	4

   - the numbers should be put in increasing order:

1	2	3
4	5	6
7	8	9

   - the game is played by making *rotations*, e.g.



   - a rotation moves 4 adjacent numbers one step clock wise
   - a rotation can be made to any 4 adjacent squares
   - a rotation is only way to move the numbers

   Define an algorithm that finds a solution to a rotation game

   - as input a $n \times n$-table with numbers in random order
   - if a solution is found, algorithm prints the rotations that lead to the solution
   - if solution not found, the user is informed

   Analyze the time and space complexity of your algorithm

   Does your algorithm find the solution with least amount of rotates? If not, how should it be changed if the shortest solution is needed?

   Note: When using Java to prevent `java.lang.StackOverflowError` you may need to set the stack size with parameter `-Xss`, eg. `java -Xss100m Rotation` sets the stack size to 100 megabytes.

   Do not run your program on machines melkki, melkinkari, melkinpaasi

8. **Magic square:** 3 crosses pseudo code, 6 crosses implementation.

   Magic square is a $n \times n$ -array that contains values $1...n^2$. The requirement is that the sum of the values is same at each row, column and diagonal.

   In the following a 4×4 -magic square where the sum is 34. Eg. the second column 2+7+10+15 = 34, fourth row 1+15+4+14 = 34 and the diagonal form upper left to lower right 8+7+5+14 = 34.

```
8  2  13 11
9  7  12  6
16 10  5  3
1  15  4 14
```

   Do an algorithm or implement a program that counts how many different $n \times n$ magic squares exists. How fast you can your program to run when n=4?

   Do not run your program on machines melkki, melkinkari, melkinpaasi