Mandatory exercise 3
INF102

Deadline: 10th of November (23:59).

This assignment is an individual task, however you are allowed to collaborate and discuss solutions *as long as you do not share code* (see our policy on Collaboration and Cheating). Similarly, for tasks that are not answered with code, you may freely discuss your answers with your peers; but clearly, blindly transcribing answers from others is not allowed.

The assignment is graded on a scale from 0 to 80, and accounts for 10% of your final grade.

Organizational Instructions

First download the zipfile *Oblig3INF102(H19).zip* from files/Oblig 3 on mittuib. The zipfile contains the classes you are required to implement as well as other classes needed which contains methods for reading input, generating random numbers, and for timing your program.

The answers to code questions should go into the respective classes for each question (*PerfectBST.java* for Task 2, *MedianPQ.java* for Task 3, and *WordCount.java* for Task 4). Note that classes should not be placed in a package.

The answers to non-code questions should be submitted in a pdf named {yourid}.pdf, where yourid is your UiB SEBRA account id. For example, if your id is abc123 the pdf should be named abc123.pdf. You can answer questions in either Norwegian or English.

Every file (the pdf + sourcefiles) should be placed in a folder named {yourid} (without the {}). Do not use subfolders, all files should be placed on the first level of your folder. It should be possible to run each program from the folder, thus include any dependent files. This directory should then be placed in a zipfile named {yourid}.zip. Finally this file should be handed in on mittuib.

If your submission does not follow these instructions (ie. we need to search for your code), you will be deducted up to 10 points.

Task 1 (20 points)

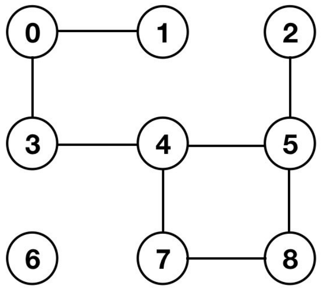
Given a positive integer *k* and an array of *n* unordered distinct integers $A = [a_0, a_1, \ldots, a_{n-1}]$.
In this task we are looking at different ways to compute all pairs $\{a_i, a_j\}$ such that $a_i + a_j = k$ while $i \neq j$. For each scenario, give a short description (max 3 sentences) describing how you would do it in order to achieve the specified running time. (Note that the running time is given using Theta- and not O-notation).

a) $\Theta(n^2)$
b) $\Theta(n \log n)$
c) $\Theta(n)$

Task 2 (20 points)

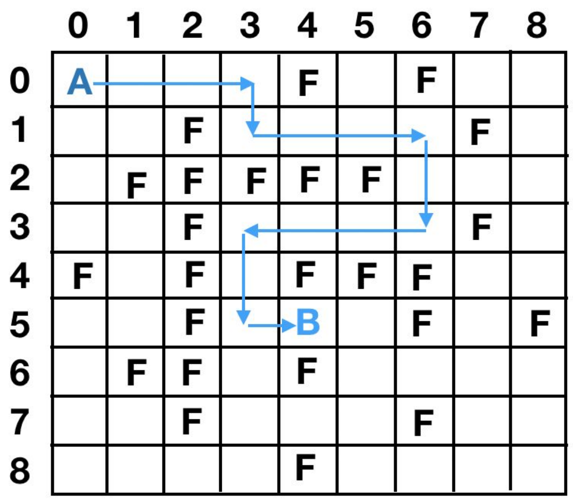
In this assignment you are to write a method detectCycle(edges) in the class `Problem2.java`, that takes an undirected graph G given in an adjacency list representation (see `DetectCycleDriver.java`) as argument and determines if it contains at least one cycle. If there is such a cycle your program should return an ArrayList containing the vertices of a cycle in the order they were discovered and if there is no cycle a NULL value should be returned. Note that the graph might not be connected. You can use the class Detect`CycleDriver.java` to test your program. Your program should run in time O(V + E), where V is the number of vertices and E the number of edges in G.

Example:
If the graph has the following structure, your program could return `[4,5,8,7]`.



Task 3 (20 points)
Write a method findPath(maze) in the class `Problem3.java`, that computes and outputs the shortest path between two points A and B in a 2D maze as an ArrayList<Pos>. The maze is given as a two dimensional array of boolean values where a false value indicates a wall. The path can only contain positions containing true values that are either connected horizontally or vertically (not diagonally). If no such path exists then a NULL value should be returned. You can use the file `MazeDriver.java` to test your program.

Example: With the input as shown below your output should be:
[(0,0),(0,1),(0,2),(0,3),(1,3),(1,4),(1,5),(1,6),(2,6),(3,6),(3,5),(3,4),(3,3),(4,3),(5,3),(5,4)]



Appendix: Style Guide

The following rules applies to all source code that is handed in. These guidelines are loosely based on Google Java Style Guide, though with some minor differences.

Most important

- Make your code easy to read.
    - Comment tricky parts of the code.
    - Use descriptive and logical variable names and function names.
    - Break the code into appropriate functions for readability and code reuse; no function should ideally be more than 30 lines of code (with the exception of extremely monotone code, such as sanity tests).
    - Write javadoc comments for functions that are not self-explanatory
- All files should use UTF-8 character encoding.

Also important

- The only whitespace characters allowed are spaces and newlines (tabs are not allowed).
- Each indentation level increases by 4 spaces.
- No line may exceed 120 characters - lines exceeding this limit must be line-wrapped.
- Some exceptions, e.g. very long URL's that don't fit on one line.
- File name should be the same as the name of the class, written in UpperCamelCase.
- Function names, parameter names and variable names should be written in lowerCamelCase.
- Constants should be written in ALL_CAPS.
- No line breaks before open braces ({).
- Blank lines should be used sparingly, but can be used to separate logic blocks and to increase readability.