1. Provide an implementation of the B-Tree data structure. Note that you are required to choose how to implement the Node structure. Recall that different choices may have different performances both in terms of computational complexity and in terms of I/O complexity. Motivate your choice.

   Provide, moreover, a script that verifies all implemented functionalities.

2. Consider a CPU that has to run a bunch of jobs. You are asked to develop a scheduler for the CPU. Your program should run in a loop, each iteration of which corresponds to a time slice for the CPU. Each job is assigned a priority, which is an integer between −20 (highest priority) and 19 (lowest priority), inclusive. From among all jobs waiting to be processed in a time slice, the CPU must work on a job with highest priority. In this simulation, each job will also come with a length value, which is an integer between 1 and 100, inclusive, indicating the number of time slices that are needed to process this job. For simplicity, you may assume jobs cannot be interrupted—once it is scheduled on the CPU, a job runs for a number of time slices equal to its length. Moreover, each time a job has spent x time slices in the waiting line its priority is increased by 1. Notice that after the priority of a job is increased its waiting time restarts from 0.
   Your simulator must output the name of the job running on the CPU in each time slice and must process a sequence of commands, one per time slice, each of which is of the form "add job name with length n and priority p" or "no new job this slice".

3. During the course, we have seen a recursive implementation for the DFS graph visit. There is also a simple equivalent iterative implementation that uses a stack as an auxiliary data structure (it runs just like the BFS algorithm, except that we use the stack in place of the queue). However, there is an algorithm that iteratively implements the DFS visit and does not use any auxiliary data structure.
   Design this algorithm and implement it.

Warning: you are allowed to modify the Graph Node Class.

4. In order to block the diffusion of misinformation on its social network, BaceFook has decided to install on the computers of some users a monitoring software that is able to recognize and block the fake news. Specifically, BaceFook wants to install the software on the minimum number of users so that for every pair of friends, at least one of them has the software.

   Suppose that the BaceFook network is a tree. Design and implement a Dynamic Programming algorithm that solve the problem in time O(n).

5. Suppose now that the BaceFook network can be any graph. Design and implement a Greedy algorithm for the problem.

   Moreover, generate 100 different graphs of 100 nodes. To do this, you may use the random Python library: in particular, the function bool(random.getrandbits(1)) can be used to return a random Boolean, that in turn can be used to decide whether to add or not a given edge.
   Finally, evaluate how the greedy algorithm performs on these graphs. Does the algorithm return an optimal solution? If not, can you bound the approximation ratio of the algorithm?

   After the deadline, all proposed algorithms will be tested on 5 secret graphs of 100 nodes. The group whose algorithm is the faster among the ten that achieve the best performances on these runs will receive a bonus point. Similarly, a bonus point will be assigned to the group whose algorithm is the one that achieves the best performances on these runs among the ten that are faster.