- *All codes must be written in one programming language, amongst C++, Python, Java.*
- *Source files must be sent at guillaume.ducoffe@fmi.unibuc.ro.*
- *Students must solve one subject. Partial solutions of multiple subjects will not be taken into account.*
- *All classroom material is allowed. Students may dispose of their own code, but they are not allowed to use others' codes. If two students share, even partially, the same code (and it is not the one provided in the correction), then both students automatically fail.*
- *Codes are tested using files with the following syntax:*
  - the number n of nodes and the number m of edges are given on the first line
  - every other line represents one edge. The two end-vertices of the edge are given and, if the graph is weighted, the line ends with the integer weight of the edge.
  Students may use in their code any implementation for their Graph class. However, they must provide a main function that reads on the terminal a file name, opens the corresponding graph file, transforms the latter in a Graph object, which is used as input for the students' program.
- *The notation is as follows:*
  - compilation 1p
  - execution 1p (for Python programmers, 2p)
  - correct output for the example graph file 2p
  - correctness of the algorithm 2p
  - time complexity (serial) or correct use of parallelism 2p
  - presence of justifications/explanations as comments in the code 2p

---

1) Consider an undirected unweighted graph. In a Maximum-Cardinality Search (MCS), vertices are visited one at a time, and, at every step, the next vertex to be visited must be one that maximizes its number of already visited neighbours.

   Implement MCS.
   Complexity: O(m+n)

2) Consider an undirected weighted graph. In Weighted-A*, we are given a fixed weight w > 1 (you may fix w = 2). As usual with A*, for open and closed vertices x we have estimates g(x) of their distance from the source node. The next open vertex to visit is chosen according to some function f(x), that depends on g(x) and the heuristic function h(x). Then, the only (but important) difference with A* is the way that we define f:

   * if g(x) < (2*w-1)*h(x), then f(x) = g(x)/(2*w-1) + h(x)
   * otherwise, f(x) = [g(x) + h(x)]/w

   Implement Weighted-A*.
   Complexity: at most O(log(n)) to find and process the next open vertex.

3) In Algorithm AA for computing connected components, one starts replacing every edge uv, where v < u, by the arc u → v. Then, the following steps are repeated until there is no more arc v → w such that w ≠ v.p:

   - we consider all arcs v → w in parallel. We set v.p = min{v.p,w}. Conflicts are resolved

according to the min rule.
- we again consider all arcs v → w in parallel. We delete arc v → w. Furthermore, if w ≠ v.p, then we add a new arc w → v.p.
- finally, we consider all vertices v in parallel. If v ≠ v.p, then we add an arc v → v.p.

Implement Algorithm AA. Modifications of adjacency list must occur in critical sections. The min-rule must be simulated by using a reducer.

4) Recall the array implementation of a queue, using two variables in order to retain the front and back elements. Modify the bitmap PBFS algorithm in order to use this array implementation for the queue. In doing so, there should be no synchronization mechanism for dequeue operations.

5) Implement UFSCC, the parallel version of Purdom-Monroe algorithm seen in class in order to compute strong connected components.