Description of adopted solution strategies and data structures

The data structure used to represent the graph contains some standard fields (integers "V", "E", symbol table "st"), the adjacency matrix (the matrix structure, although it may be inconvenient from a memory point of view for sparse graphs, has been chosen, mainly to benefit from direct access in the operations of removing edges from the graph, which can occur frequently in the proposed problems), an array of integers "vertex" for marking vertices (0 -> vertex is not part of the graph, 1 -> vertex is part of the graph) and an integer "nV" (for counting the vertices that are actually part of the graph at a certain time).

To solve the first proposed problem (k-core) a recursive approach was followed: check if any vertex has a degree less than k, if successful proceed with its removal (by appropriately marking the corresponding cell of the “vertex” array) and the removal of the edges that insist on it, and so on until reaching an instance of the recursive call in which no more removals will take place. Having therefore reached this terminal case, the set (potentially empty) of vertices belonging to the k-core has been found, so this solution is printed on screen. The recursive calls are closed by means of a flag and the initial conditions of the graph are restored (vertices and edges previously deleted are added again).

The scheme of powerset with simple combinations was adapted in order to solve the second proposed problem (j-edge-connected): all the edges of the graph are extracted and stored in a “val” array, through an external loop we can iterate with and index “i” variable between 1 and G->E, then simple i-combinations of the |E| edges are computed and in every terminal case we can verify if the computed subset of edges disconnects the graph (by removing the edges and using the standard function GRAPHcc, after which edges are re-added); once a set of i edges that disconnects the graph has been discovered (suitably indicated by the return value of the comb_sempl function), we can check whether this set has cardinality less than, equal to or greater than j and the final result will depend on this (i<j, graph not j-edge-connected, i>=j, graph j-edge-connected).

Notes

I created two input files, grafo2.txt which reproduces the G graph of the exam and is useful for testing request 2 of the exam, grafo3.txt which reproduces the example graph for request 3 of the exam and is useful for testing the j-edge-connected algorithm.