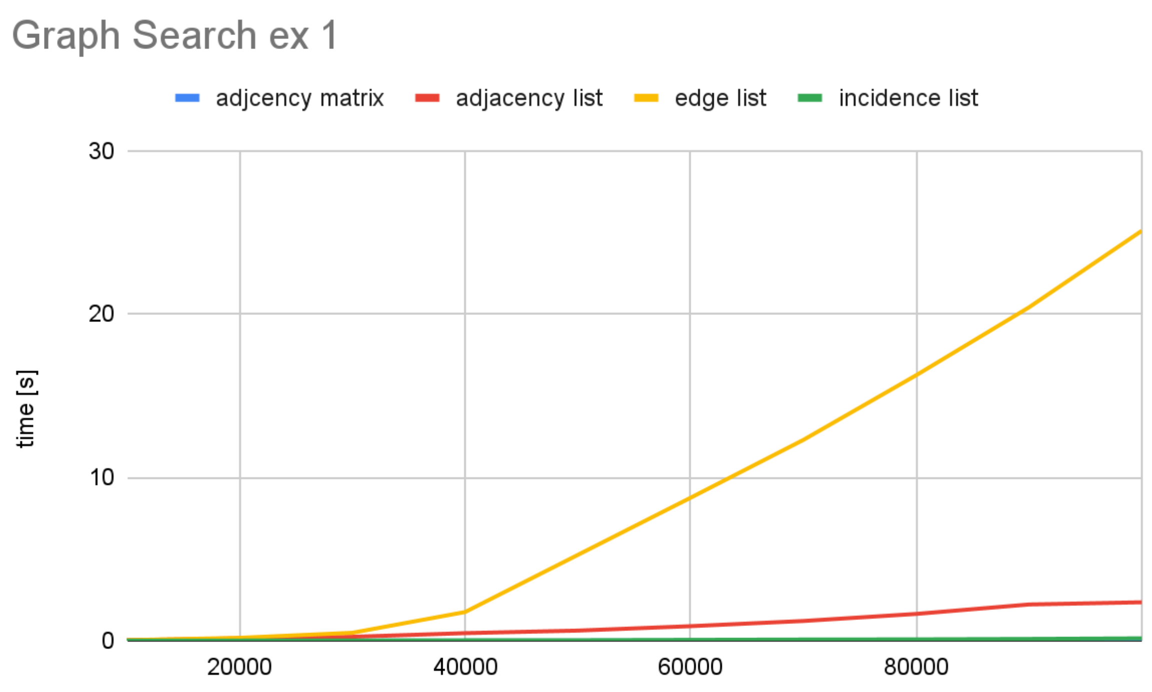
Algorithms and data structures
Graph Algorithms
In this exercise the impact of different graph representation method on the performance of searching operation was tested. Four graph representation methods were implemented:

- vertex matrix
- incident matrix
- list of incidents
- edge list

Edge list was the simplest graph representation to implement, since it is a list of lists that consists of pairs of nodes, between which an edge exists. Despite the intuitiveness of this method, checking whether an edge exists between two nodes was the least time efficient, out of all methods, and the time complexity of such operation was O(n^2) (iterating through whole graph and checking if the nodes in array were the same as the input data). Edge list space complexity is O(V), where V is the number of vertices that exist in said graph.

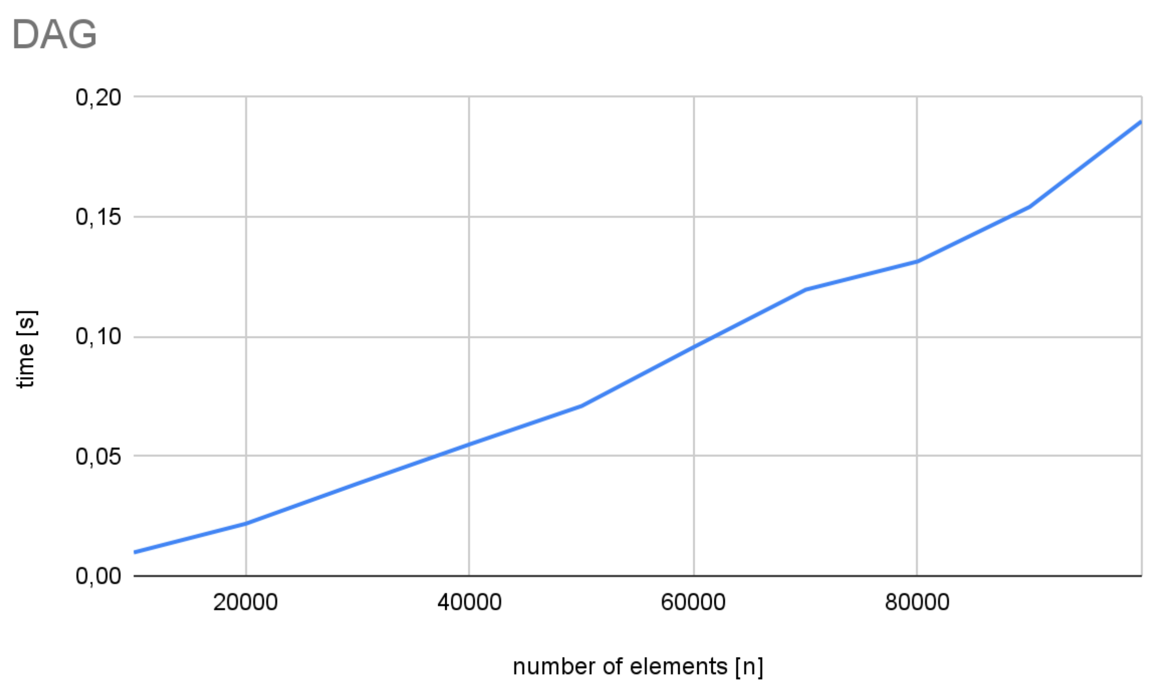
Both incident list and incident matrix check the connectivity of the graph by representing the connection in some way, rather than writing existing pairs of nodes. For incidence list, for each node in a graph there are other nodes written in a table, that are connected to it, while incidence matrix has two coordinates, edges and vertices. Edge connects two nodes if the value in table is equal to boolean true. Space complexity of list of incidents is O(V+E), while for incident matrix it is O(V*E), while time complexity for those representations is O(V) and O(V^2).

Vertex matrix is a matrix where both coordinates are vertices. If an edge exists between two vertices, a value equal to boolean true is in the matrix. Space complexity for this representation is O(V^2), while time complexity is just O(1).



Ex 2

For topological sorting procedure of DAG graph incident list graph representation was chosen, since it has lower memory usage and smaller time complexity than the adjacency matrix [O(|V|+|E|) for incident list and O(|V|*|E|) for incident matrix].



Implementation of DAG graph in incident list and then using a topological sorting algorithm was easier than in other methods because of how dfs works. Adjacency list not only uses memory more efficiently than other graph representations, but also it allows us to store pointers to nodes the node is connected with. Traversing it with dfs means following that linked list and finding the node that has no nodes going outwards.