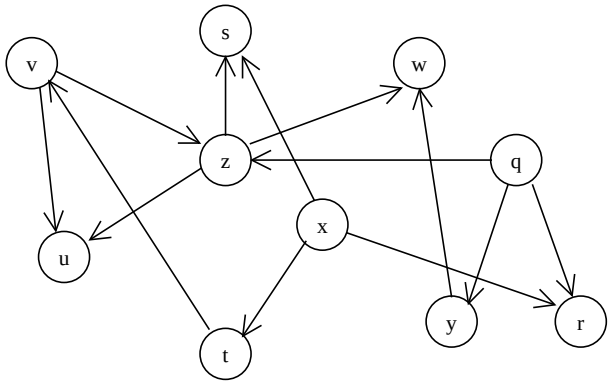
Data Structures, exercises week 11, 19-23.4.

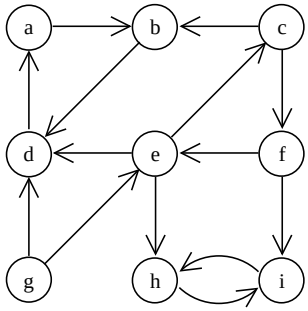
1. In Cormen breath first search is implemented iteratively (without recursion) and deapth first search is implemented with recursion. Show how breath first search could be done with recursion and deapth first search without recursion.

2. Do a topological ordering to the graph below using the algorithm given in Cormen. Assume that nodes are in aphabetical order in the adjacency lists.



3. Define the strongly connected components of the graph below using the algorithm in Cormen. Assume that nodes are in aphabetical order in the adjacency lists.

   What is the smallest amount of edges needed to make the graph strongly connected?



4. Adjacency list representation of a graph consumes memory only $O(|V| + |E|)$, but it is slow to check if a given edge (v, w) belongs to the graph.

   In adjacency matrix representation testing if a given edge (v, w) belongs to the graph takes only constant amount of time. The bad news is that adjacency matrix representation cosumes memory $O(|V|^2)$ which is for sparse graphs much more that adjacency list representation.

   Design yet another way to represent graphs, such that it does not take too much memory but in which it is quick to check the existence of a given edge (v, w). The memory consumption should be $O(|E| + |V|)$, and checking the existence of an edge should take time $O(\log |V|)$. Are there any weaknesses in your solution?

5. Give an algorithm that checks if a graph is *bipartite*.

   A graph is biparite if its vertices can be divided into two disjoint sets U and V such that every edge connects a node in U to one in V.

6. Make a program, which counts how many rooms there are in a given house. As input the algorithm gets a two-dimentional character table representing the house. `#` means the wall and `.` means the floor.

   The input could be for example:

```
#############
#..#..#....#
#..#..#....#
#####.#.#..#
#...#...#..#
###.########
#.....#....#
#############
```

   There are 4 rooms in the example above.

   To make the mark it is also enough to do a thoroughly thought exact pseudo code.

7. **A bonus exercise which substitutes 2 other questions from either this or some earlier excercises.**

   The program gets a set of dependencies as input. The input could be for example the dependencies between course passing order:

```
ohpe ohja
ohja tira
tira lama
tira tiralabra
ohja javalabra
javalabra tiralabra
ohpe ohma
tiralabra ohtuprojekti
javalabra ohtuprojekti
ohma ohtu
ohtu ohtuprojekti
lama ohtuprojekti
ohtuprojekti ohja
```

   Each row declares one dependency, for example `ohja tira` definies that `ohja` must be passed before taking `tira`.

   The program checks first, that the dependencies don't form a cycle. If we added this line to the list above, `ohtuprojekti tira`

   then a dependency cycle would be introduced:
   `ohtuprojekti tira tiralabra ohtuprojekti`

   It is not necessary to print the dependency cycle. It is sufficient, if the program declares that a cycle is found. Printig out the cycle of course has style.

   If there is no dependency cycle, the program prints *one* possible order, that doesn't violate any dependency.

   You can use any programming language.