5 Graphs

1. In this case both.

2. (a) c a b f e d
   (b) c a f e b d
   (c) c a b d f e

3. *Adj. list  Adj. matrix*

```
 v  0  0  0  0
    0  0  0  0
    0  1  0  1
    1  0  0  0

 k
```

The most important operations are InsertEdge\(^1\) and DeleteEdge together with
the traversal operations First and Next. We summarize the complexity results
in the table below. Both IsIndex and Vertex execute in constant time, independ-
ently of the representation chosen. By a neighbour we mean in this case a vertex
reachable by an outgoing edge.

<table>
<thead>
<tr>
<th>Operation</th>
<th>adj. matrix</th>
<th>adj. list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Edge</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Delete Edge</td>
<td>$\Theta(1)$</td>
<td>$\Omega(1), O(k)$</td>
</tr>
<tr>
<td>First</td>
<td>$\Omega(1), O(v)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Next</td>
<td>$\Omega(1), O(k)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Exist Edge</td>
<td>$\Theta(1)$</td>
<td>$\Omega(1), O(k)$</td>
</tr>
</tbody>
</table>

The instruction
```
for each neighbour $u$ of $v$ do
  foo
```

can be implemented as
```
i \leftarrow First(v)
while IsIndex(i) do
  foo
  i \leftarrow Next(i, v)
```

\(^1\)We assume that the operation is only called for insertion of a new edge, not for an already existing
one.
<table>
<thead>
<tr>
<th>Operation</th>
<th>No. of calls</th>
<th>adj. matrix</th>
<th>adj. list</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>1</td>
<td>$\Omega(1), O(v)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>IsIndex</td>
<td>$\Omega(1), O(k)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Vertex</td>
<td>$\Omega(1), O(k)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Next</td>
<td>$\Omega(1), O(k)$</td>
<td>$\Omega(1), O(k)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>

For the adjacency matrix representation we can do a more precise analysis if we look together at all calls of First and Next. This requires only traversal of one row in the matrix, which includes $v$ vertices. We get thus complexity $\Theta(v)$, which is more precise than $\Omega(1), O(k, v)$.

4. (a) Modify depth-first search algorithm (p. 595).

Notice that the DFS algorithm called for $G$ and $v$:

- Visits only the vertices in the connected component of $G$ including $v$;
- marks the edges as discovery edges and back edges; a graph is cyclic iff there exists a back edge.

The modification of DFS algorithm includes thus:

- exploration of the whole graph $G$ (in contrast to DFS exploring only one connected component). This can be achieved by starting DFS for each unexplored vertex of $G$.
- reporting a cycle and stopping instead of marking an edge as a back edge.

(b) We assume that

- accessing a vertex and accessing a neighbour of a vertex takes time $O(1)$; this is true when graph is represented by adjacency matrix,
- Changing the encountered information of a vertex and checking it takes time $O(1)$

At initialization each vertex is visited once, $\Theta(n)$. DFS is called at most once on each vertex (since discovering of a cycle stops the exploration), and every edge is examined at most twice, when checking encountered neighbours for both end vertices of the edge. Hence the complexity is $O(n + e)$. Notice that an acyclic graph has at most $n - 1$ edges which gives $O(n)$ complexity in that case.

(c) A tree is a connected acyclic graph. Thus, if the first call of DFS does not find a cycle, it suffices to check if it explored all vertices of the graph.

5. (a) This is a directed acyclic graph.
(b) We use Topological Sort. The result is shown below: the annotation 1..9 shows a possible ordering of the courses to be taken by Johan.

(c) The DAG describes a partial ordering of prerequisites: $c_1 > c_2$ if there exists a path from $c_1$ to $c_2$. The courses which are not ordered can be taken in parallel. But the courses on the same path must not be taken in the same term. Thus the number of nodes on the longest path determines the minimum number of terms needed to schedule all courses. The scheduling can be done by the breadth-first search.

In our example the maximal number of nodes on a path is four. Thus the schedule may look as follows:

Term 1: DM, P1; Term 2: DS, TP; Term 3: FL, P2, AA; Term 4: CC, OT.

6. The idea is to label the vertices by different natural numbers showing their topological order. To achieve this extend the DFS algorithm with a counter initialized to the number of vertices in the graph. Whenever DFS reaches a backtrack vertex, the vertex is assigned the actual number of the counter, and the content of the counter is decremented. If the graph is not connected the DFS has to be successively called on each of the components. For the example see the solution of the next problem.

7. The following graph shows the restrictions:

This is a directed acyclic graph and the problem can be solved by Topological Sort. It is based on the modification of depth-first search discussed above. We assume here that the neighbours of a vertex are explored in DFS in the order reflecting their geometrical placement on the figure: the neighbours placed higher
are selected first.

With the above assumption on ordering of neighbours the numbers are assigned in the following order:

- Moving in: 12
- Painting interior: 11
- Floor: 10
- Roof cover: 9
- Roof: 8
- Chimney: 7
- Insulation: 6
- Doors: 5
- Interior walls: 4
- Windows: 3
- Exterior walls: 2
- Ground: 1

The schedule is determined by undertaking the task in increasing order of their computed numbers.