T-79.4202 **Autumn 2013**
Principles of Algorithmic Techniques
Programming Tutorial 2, November 5–6
Problems

For both of the problem sets 1 and 2 described below, design, implement and test a program for *one* of the alternatives (a) or (b). You may use any programming language of your choice. Your programs should read a graph in DIMACS file format (see below) and print the result as a single line of text to the standard output. Valid input/output pairs for testing your programs are available in Noppa.

The **deadline** for submitting your solutions is Monday November 4, 23:59 p.m. In order to earn credit, you must also attend either one of the tutorial sessions on 5–6 November, and be ready to explain and demonstrate your solutions. For further details and instructions, see `https://noppa.aalto.fi/noppa/kurssi/t-79.4202/harjoitustyot`.

1. **Graph algorithms I:**

   (a) Topological sort of a directed acyclic graph.

   A *topological sort* of a directed graph is an ordering of the vertices such that for every directed edge $u \to v$, the vertex u comes before the vertex v in the order. Given a directed graph in DIMACS format, print the integer labels of the vertices, separated with spaces, in a topologically sorted order. If the graph contains a directed cycle, topological sorting is not possible. In this case, print the word `cyclic`.

   Example. The directed graph G in Fig. 1 is acyclic. One possible topological sort is given below.

   ```
   3 2 4 1 5
   ```

   (b) Bipartiteness of an undirected graph.

   A *bipartition* of a graph is a partition of the vertex set V into two sets V_1 and V_2 (with $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$) such that every edge of the graph has one end in V_1 and the other in V_2. A graph is *bipartite* if it has a bipartition. Given an undirected graph in DIMACS format, construct a bipartition and print the labels of one partite set (say, V_1), separated with spaces, in numerically ascending order. If the graph is not bipartite, only print the text `not bipartite`.

   Hint. Once we fix a vertex $v \in V_1$, all neighbours of v must be in V_2, and vice versa. Apply this idea until all vertices are designated in either V_1 or V_2.

   Example. A possible output given the bipartite graph H in Fig. 1 is as follows.

   ```
   4 5 7
   ```

2. **Graph algorithms II:**

   (a) Longest path in a directed acyclic graph.

   A *path* in a directed graph is a sequence of vertices and edges $v_0 \to v_1 \to \cdots \to v_k$, where $k \geq 0$ and the vertices $v_0, \ldots, v_k$ are distinct. The *length* of the path is number of edges k. Given a directed graph in DIMACS format, check that the graph is acyclic, find a longest path, and print the integer labels of the vertices on the path, separated with spaces, in order from v_0 to v_k. If there are several paths of the maximum length, pick one of them. If the graph contains a directed cycle, print the string `cyclic`.

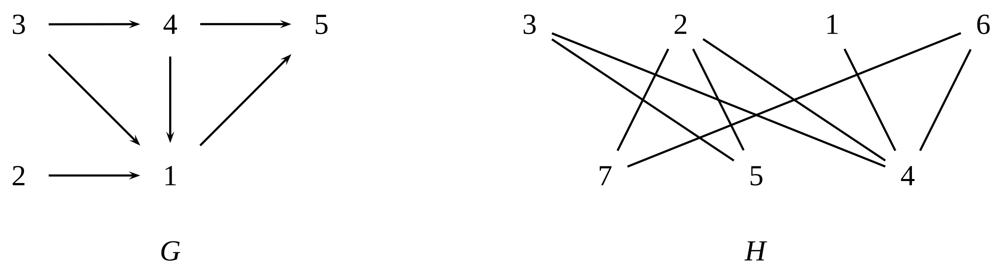

Figure 1: Example graphs.

Hint. Acyclicity makes the problem easier. You can go through the vertices in a reverse topological order, keeping track of the longest path starting from each vertex.

Example. The graph G in Fig. 1 has a unique longest path, given below.

```
3 4 1 5
```

(b) Shortest cycle in an undirected graph.

The *length* of a cycle in a graph is number of edges (or equivalently, the number of vertices) on the cycle. Given an undirected graph in DIMACS format, find a shortest cycle and print the length of the cycle. If the graph contains no cycle, print the string `acyclic`.

Hint. Start a new breadth-first search from every vertex v to find the shortest cycle that contains v.

Example. The graph H in Fig. 1 has several shortest cycles of length 4. One of the cycles consists of the vertices 2, 7, 6, and 4. Below is the correct output.

```
4
```

The DIMACS graph format

We use the textual DIMACS file format for representing graphs. The same format is used for both directed and undirected graphs. The file begins with a line of the form

`p edge` n m

where $n \geq 0$ is the number of vertices and $m \geq 0$ is the number of edges. The vertices are labeled with integers from 1 to n. This is followed by m lines describing the edges. Each such line has the form

`e` u v

where u and v are integers from 1 to n. This represents an edge joining vertices u and v. If the file represents a directed graph, the edge is directed from u to v.
Your programs should read these lines from the standard input, or if you prefer, from a file whose name is given as a command line argument. You can assume that the input conforms to the format described above. It is not necessary to check syntax errors, but correct syntax must result in correct output.