In the lab-classes this week we experiment with *elementary graph algorithms*: BFS and DFS.

**Update the environment:**  Update the repository via
git pull
or, if you didn’t create it before, create a (new) clone by
git clone git://github.com/OKullmann/CS-242-Algorithms.git

Change to the subdirectory for week 4:
cd ~/CS-242-Algorithms/201112/Week04/

**Basic setup:**  We

1. compile the programs
2. run the code on small examples, to understand the algorithms.

Compilation is done by
Week04> make

The executables produced are

BreadthFirstSearch
BreadthFirstSearch_forest
DepthFirstSearch_forest

**Comments on the programs**  The three programs are as follows:

1. **BreadthFirstSearch** $N$ $u$ runs BFS on a directed graph with $N$ vertices and start vertex $u$.
2. **BreadthFirstSearch_forest** $N$ performs BFS on an undirected graph with $N$ vertices, for all vertices, in the given order.
3. **DepthFirstSearch_forest** $N$ performs DFS for all vertices, in the given order, by default for directed graphs, while a second input switches it to undirected graphs.

Vertices are represented by natural numbers from 0 to $N - 1$ (so examples from the script need renaming). All three programs take as first argument the number $N$ of vertices.

**Understanding BFS**  Undirected graphs are handled via **BreadthFirstSearch_forest**, which runs in an outer loop through all vertices (in the natural order), as with DFS. The example from the script is as follows (entering edges as pairs of vertices; recall that the end of the input is announced to a program reading from standard input by Ctrl-D).
Week04> ./BreadthFirstSearch_forest 7
0 1
1 2 1 3
2 3 2 4
3 4 3 5 3 6
4 6
9 edges have been read.
Vertex 0: Distance: 0; Parent: nil
Vertex 1: Distance: 1; Parent: 0
Vertex 2: Distance: 2; Parent: 1
Vertex 3: Distance: 2; Parent: 1
Vertex 4: Distance: 3; Parent: 2
Vertex 5: Distance: 3; Parent: 3
Vertex 6: Distance: 3; Parent: 3

Directed graphs are handled as follows, now entering the start vertex as second argument:

Week04> ./BreadthFirstSearch 3 0
1 0 1 2
2 edges have been read.
Vertex 0: Distance: 0; Parent: nil
Vertex 1: Distance: infinity
Vertex 2: Distance: infinity

Week04> ./BreadthFirstSearch 3 1
1 0 1 2
2 edges have been read.
Vertex 0: Distance: 1; Parent: 1
Vertex 1: Distance: 0; Parent: nil
Vertex 2: Distance: 1; Parent: 1

> ./BreadthFirstSearch 3 2
1 0 1 2
2 edges have been read.
Vertex 0: Distance: infinity
Vertex 1: Distance: infinity
Vertex 2: Distance: 0; Parent: nil

Tasks (answers to be shown to the postgrads):

1. Draw the above examples (of course, on paper), perform the algorithms, and check/understand the above output.

2. Create an undirected graph with three connected components and 12 vertices altogether, predict the output of the forest-version, and check.

3. Create the digraph which is the circuit with 5 edges, predict the output of the start-vertex-version for all five possible choices of the start-vertex, and check.

4. Draw the computed spanning trees / spanning forests for all (di)graphs you considered.
Understanding DFS

> ./DepthFirstSearch_forest N

prepares a directed graph with N vertices. The program then reads edges as pairs of vertices from standard input, and outputs the nodes of the BFS-forest. If the graph shall be undirected, then use some second parameter (the value doesn’t matter). The example from the script is replicated as follows (where the second parameter could be anything — without it the graph is directed, with it undirected):

> ./DepthFirstSearch_forest 7 u

0 1
1 2 1 3
2 3 2 4
3 4 3 5 3 6
4 6
9 edges have been read.
Vertex 0: Discovery time: 1, Finishing time: 14; Parent: nil
Vertex 1: Discovery time: 2, Finishing time: 13; Parent: 0
Vertex 2: Discovery time: 3, Finishing time: 12; Parent: 1
Vertex 3: Discovery time: 4, Finishing time: 11; Parent: 2
Vertex 4: Discovery time: 5, Finishing time: 8; Parent: 3
Vertex 5: Discovery time: 9, Finishing time: 10; Parent: 3
Vertex 6: Discovery time: 6, Finishing time: 7; Parent: 4

Note that the order of edges is important, if we wish to exactly reproduce the example from the script — with different orders the neighbours are enumerated in different orders, and then parents as well as discovery and finishing times will be different.

Tasks (answers to be shown to the postgrads):

1. Draw the above examples, perform the algorithm, and check/understand the above output.

2. Enter the example for “topological sorting” from the script, given on the next page, and understand the computation. Note that you have to figure out first how to number the vertices.

3. In the section on BFS, you entered a graph with 3 components and 12 vertices: Apply also DFS to this graph, and understand the output.

4. How many edges can a digraph have, so that the spanning forest computed by DFS will just consist of trivial, one-node trees? (Of course, this will be the case if there are no (directed) edges at all, however how many can there be maximally?) Check your answer experimentally.

5. On the other hand, what is the minimum number of (directed) edges so that we just get a single tree?

6. Draw the computed spanning trees / spanning forests for all (di)graphs you considered.
(labelling: d[u]/f[u])