

In the lab-classes this week we experiment with *elementary graph algorithms*.

Update the environment: In your existing directory CS-242-Algorithms:

```
git checkout master
git pull git://github.com/OKullmann/CS-242-Algorithms.git master
```

Or if this directory (repository) doesn't exist (anymore), resp. you want to create a new repository, create a new clone by

```
git clone git://github.com/OKullmann/CS-242-Algorithms.git
```

In case you created a new clone, by (just) `git pull` you will pull from OKullmann's Github repository in the future. Now change to the subdirectory for week 6:

```
cd CS-242-Algorithms/200910/Week06/
```

Basic setup: We

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

Compilation is done by

```
Week06> make
```

The executables produced are

```
BreadthFirstSearch
BreadthFirstSearch_forest
DepthFirstSearch_forest
```

Comments on the code The files are as follows:

1. `AdjacencyList.hpp` provides class `AdjacencyList` for representing directed and undirected graphs via adjacency lists.
2. `BreadthFirstSearch.hpp` contains the "functor" class `BreadthFirstSearch` (a function wrapper) for performing breadth-first search for a directed graph.
The standard form, for a given start-vertex s , is provided, as well as a global form, which computes a spanning forest. The latter form is somewhat unusual for BFS (while it is the standard for DFS), but makes sense for undirected graphs, where it computes the connected components (the spanning trees for the components are BFS-trees with the smallest vertex in the component as root; recall that vertices are presented by natural numbers).
3. `BreadthFirstSearch.cpp` is a simple application performing BFS on directed graphs with given start vertex.
4. `BreadthFirstSearch_forest.cpp` is another simple application performing BFS on undirected graphs for all vertices, in the given order.

5. `DepthFirstSearch.hpp` contains the functor class `DepthFirstSearch` for performing depth-first search for a directed graph.
6. `DepthFirstSearch_forest.cpp` is a simple application performing DFS for all vertices, in the given order.

Vertices are represented by natural numbers from 0 to $N - 1$ (so examples from the script need renaming).

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`).

```
Week06> ./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
```

For directed graphs only the form with a single start vertex is provided:

```
Week06> ./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
```

```
Week06> ./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 by 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. Consider again the directed circuit with 5 edges, predict the output for the three vertex orders “following the flow”, “against the flow”, and “hopping around”, and check.
3. Enter the example for “topological sorting” from the script (see next page), and understand the computation.
4. 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.
5. 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.

6. On the other hand, what is the minimum number of (directed) edges so that we just get a single tree?
7. Draw the computed spanning trees / spanning forests for all (di)graphs you considered.

