Exercise Sheet 8

You are allowed to work in groups of two. Only one submission per pair is necessary, but please indicate who you are working with. Submit your results as a zip-file to jpreusse@uni-koblenz.de until 02.07.2012, 11:59 pm.

1 Ford–Fulkerson Maximal Flow Algorithm (6 points)

The Ford–Fulkerson algorithm computes a maximal flow $f$ along edges $E$ of the network $N = (V, E, C)$ between the source node $q \in V$ and the sink node $z \in V$, with edge capacities $C \in \mathbb{N}^{E}$ defined for all edges $E$. Further, the following condition holds for all $e \in E$: $f(e) \leq c(e)$, i.e. the maximal flow on a given edge cannot exceed the edge’s capacity.

The core idea of this algorithm is as follows. As long as there exists a path $P(q, z) \subseteq E$, i.e. a directed connection from $q$ to $z$ with a positive flow $f$ defined by

$$f \leftarrow \min_{(i,j) \in P(q,z)} c((i,j)),$$

augment the maximal flow with this newly found flow $f$. In Octave, this can be implemented using a slightly modified BFS\[1] from $q$ to $z$. Whenever an augmenting flow $f$ was found, the edges on that path $(i, j) \in P(q, z)$ from node $i$ to node $j$ have to be adjusted the following way:

$$c_{new}(i,j) = c_{old}(i,j) - f$$
$$c_{new}(j,i) = c_{old}(j,i) + f,$$

i.e. the current flow is subtracted from a traversed edge and is added to the backwards edges.

Now, you should implement the Ford-Fulkerson algorithm. The signature of your function should look like this:

\begin{verbatim}
function [maxFlow, flow] = fordFulkerson(matrix, q, z)
\end{verbatim}

where $q$ is the source node and $z$ is the sink node ID and a row in matrix has the form $[i, j, c(i,j)]$ for all edges $(i, j) \in E$ and according capacities $c \in \mathbb{N}^{E}$. Your function should return the size of the maximal flow, maxFlow, and the actual maximal flow, flow, which should be a $|V| \times |V|$ matrix with flow$(i,j)$ defined as the size of the flow along this edge.

![Abbildung 1: Visualization of network 'flowNet.txt']

Thus, your method should be called in the command line, e.g. with

\[1\] But you can also implement a DFS, it is up to you.
[maxFlow, flow] = fordFulkerson(load('flowNet.txt'),1,7)

where 'flowNet.txt' is attached as a dataset and visualized in Figure 1 and return something equivalent to

maxFlow = 8

flow =
(1,2) -> 2
(1,4) -> 3
(1,6) -> 2
(2,3) -> 2
(2,5) -> 1
(3,7) -> 2
(4,5) -> 2
(4,6) -> 1
(5,7) -> 3
(6,7) -> 3.

What is the complexity of the Ford-Fulkerson algorithm and how can it be improved in theory, so without implementing it (2 points)?

2 Transformation from Matching to Maximal Flow (4 points)

2.1 Maximal and Perfect Matchings (2 points)

Recall that a matching $M$ on a graph $G = (V,E)$ is a subset of edges $M \subseteq E$ such that every node $v \in V$ is adjacent to at most one matching edge. If a perfect matching on $G$ exists, every node is adjacent to exactly one matching edge. For this exercise we only consider matchings on bipartite graphs.

How can the problem of finding a maximal matching in a bipartite graph be reduced to the problem of finding a maximal flow in a network? I.e. given a bipartite graph $G = (I \uplus J, E)$ with $E \subseteq I \times J$, how can this be converted into a maximal flow problem, where the size of the maximal flow equals the size of the maximal matching? (2 points)

2.2 Transformation (2 points)

You should now implement the graph transformation from Task 2.1. The signature of your function should look like this:

function [capacityMatrix] = transform2FlowProblem(edges)

where the capacityMatrix is exactly the input for the Ford-Fulkerson algorithm and edges $E$ are the edges of a given bipartite graph $G = (I \uplus J, E)$.

Thus, your method should be called in the command line, e.g. with the set of edges from 'matching.txt', shown in Figure 2

[maxFlow, flow] = fordFulkerson(transform2FlowProblem(load('matching.txt'))),q,z)

and return
Abbildung 2: Visualization of network ‘matching.txt’

maxFlow = 5

flow = ???