Institut für Theoretische Informatik
Leibniz Universität Hannover

Solution to exercise sheet 2

09.04.2013

Exercise 1: Let $G = (V, E)$ be a graph with

$$\Delta(G) + \delta(G) + 1 \geq ||V||.$$

Prove that G is connected.

Solution: We prove this by confutation of the opposite: there is no unconnected graph G such that $\Delta(G) + \delta(G) + 1 = ||V||$. Hence there cannot be any unconnected graph G with $\Delta(G) + \delta(G) + 1 \geq ||V||$, because one would have to add further edges. Thus for all graphs G with $\Delta(G) + \delta(G) + 1 \geq ||V||$ that they are connected.
Let G be an unconnected graph. Then G can be partitioned into the connected components $K_1, K_2, \ldots, K_n$. Let denote with K_{max} the connected component with the maximum number of vertices. Only in this component a vertex degree of $||V_{K_{max}}|| - 1$ can be achieved. Similarly for the component K_{min} consisting of the fewest vertices that the minimum vertex degree can be at most $||V_{K_{min}}|| - 1$. Now it holds that

$$\begin{aligned}\Delta(G) &= \max_{v \in V} \deg_G v \leq ||V_{K_{max}}|| - 1,\\ \delta(G) &= \min_{v \in V} \deg_G v \leq ||V_{K_{min}}|| - 1,\end{aligned}$$

whence we get for the initial equation:

$$\begin{aligned}\Delta(G) + \delta(G) + 1 = \max_{v \in V} \deg_G v + \min_{v \in V} \deg_G v + 1 &\leq ||V_{K_{max}}|| - 1 + ||V_{K_{min}}|| - 1 + 1\\ &= ||V_{K_{max}}|| + ||V_{K_{min}}|| - 1\\ &< ||V||.\end{aligned}$$

For every unconnected graph this equation does not hold. Hence the opposite is true. □

Exercise 2: State an algorithm to determine the connected components of a given undirected graph using BFS.

Solution: We could solve this exercise by the computation of strongly connected components as in the following exercise after adding for each undirected edge both directed ones. However there is an algorithm that does not make use of the SCCs which is fairly easy to find: □

Algorithm 1: Connected-Components(G).

```
Input  : Undirected graph G = (V, E)
1 i ← 0;
2 while there is a vertex v ∈ V without a marking do
3     call BFS(G, v) and mark all vertices u s.t. mark[u] = true with i;
4     i ← i + 1;
```

Exercise 3: State an algorithm to determine the strongly and weakly connected components of a given digraph.

Solution: We will start by outputting the strongly connected components (SCCs): Naturally we construct a depth-first-algorithm DFS from our BFS algorithm by using a stack instead of the queue. However we need to extend the algorithm a little bit by introducing discovery and finishing times. This is required to built an algorithm computing SCCs. So at first the DFS algorithm:

Algorithm 2: Depth-first search DFS(G, s)

```
Input  : Graph G = (V, E).
1  Init route[v] with − for every v ∈ V;
2  stack S ← ∅;
3  time ← 0;
4  while there is an unmarked vertex s ∈ V do
5      S.push(s);
6      while S is not empty do
7          w ← S.pop();
8          time ← time+1;
9          if w has a discovery time then f(w) ←time; // finishing time
10         else
11             d(w) ←time; // discovery time
12             S.push(w);
13             forall the edges e = (w, v) ∈ E do
14                 if not mark[v] then
15                     mark[v] ←true;
16                     S.push(v);
17                     route[v] ←route[w]+1;
```

Further we need a last ingredient for the final algorithm: *transposes* of graphs. Let $G = (V, E)$ be a directed graph. Then $G^T = (V, E^T$ is the *transpose* of G, where $E^T := \{(u, v) \mid (v, u) \in E\}$, i.e., we have reversed the directions. By adjacency-list representation it requires time $O(||V|| + ||E||)$ to create G^T.
Now the algorithm that computes the SCCs of a digraph is the following:

Claim 1. *The algorithm Strongly-Connected-Components(G) is correct.*

Algorithm 3: Strongly-Connected-Components(G)

Input : A digraph $G = (V, E)$

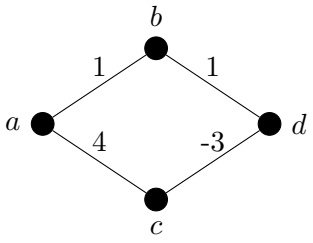
1 call DFS(G) to compute finishing times $f(u)$ for each vertex u;
2 compute G^T;
3 call DFS(G^T), but now consider the edges in the **forall**-loop w.r.t. decreasing order of the finishing times of the vertices;
4 output the vertices of each tree in the depth-first forest formed in the previous line as a separate SCC.

Proof of Claim 1. *We prove the result by induction on the number of depth-first trees found in line 3 of the algorithm that the vertices of each tree form a SCC. The induction hypothesis is that the first k trees produced are SCCs. The basis for $k = 0$ is trivial. Induction step $k \to k+1$: Let u be the root of the $(k+1)$st tree and u be in the SCC C. Through the choice of roots it holds $f(u) = f(C) > f(C')$ for any SCC C' other than C that has yet to be visited, where $f(X)$ is $\max f(x)$ for $x \in X$. Now it is easy to show that all vertices of C are descendants of u in its depth-first tree. Further any edges in G^T that leave C must be to SCCs that have already been visited. Hence no vertex in any SCC other than C will be a descendant of u during the dfs of G^T. Hence, the vertices of the depth-first tree in G^T rooted at u form exactly one SCC.*

We say that the *component graph* G' for a given graph G is defined as follows. Let $S_1, \ldots, S_k$ be the SCCs of G. Then replace each SCC S_i by a fresh vertex s_i and transfer the edge connections outside of the SCC S_i to s_i.
Now turn towards the weakly connected components (WCCs). Here we make use of the following observation. A graph G is weakly connected iff its component graph G' is a path. By this one can easily use the previous algorithm for SCCs to compute the WCCs. □

Exercise 4: Explain why the DJP algorithm does not work properly for negative weights. State an example where it fails.

Solution: The problem is that the algorithm would not decide for a worse edge to get a much better edge in the next step. This is some kind of local minimum phenomenon.



So the shortest path from a to d is over c but DJP would choose the path over b. □