Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

8th December 2016

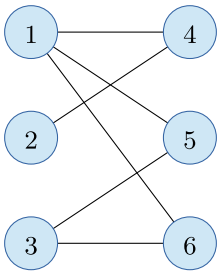
Algorithms & Data Structures — Solutions to Sheet 11 — AS 16

Solution 11.1 *Properties of example graphs.*

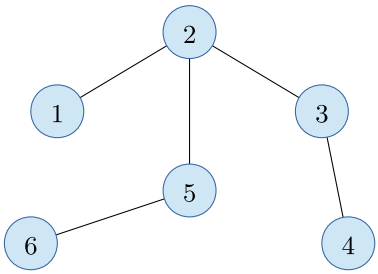
a) (i)
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

(ii)
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

b) (iii)



(iv)



c) (i) directed, not bipartite, not acyclic, loop at vertex 2,

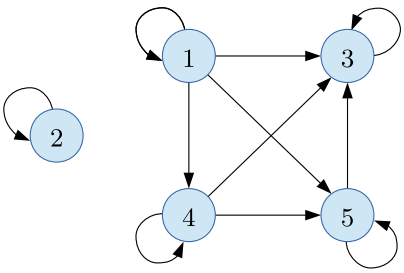
(ii) undirected, not bipartite, not acyclic, no loops,

(iii) undirected, bipartite, not acyclic, no loops,

(iv) undirected, bipartite, acyclic, no loops.

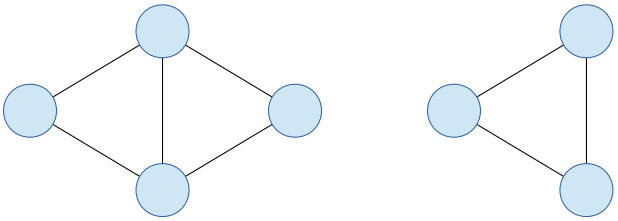
d) Only the graphs (ii), (iii) and (iv) are undirected. They are all connected, as there is a path from v to w for each pair of two vertices v and w. The graph (ii) has an Eulerian cycle, e.g. $\langle 1, 4, 3, 2, 5, 3, 1 \rangle$. Therefore, it also has an Eulerian path (where the starting vertex and the end vertex are not equal). The graph (iii) has no Eulerian cycle, as there are vertices with odd degree. However, as there are only two vertices with odd degree, there exists an Eulerian path, e.g. $\langle 1, 5, 3, 6, 1, 4, 2 \rangle$. The graph (iv) has neither an Eulerian cycle, nor an Eulerian path, as there are four vertices with odd degree.

e)



$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

f) There exist multiple solutions. For example,



There are even more solutions, e.g. graphs with some vertices with loops.

There exists no graph with 7 vertices, such that exactly one of them has degree 3 and all others have degree 2, because the sum of all vertex degrees is always even.

Solution 11.2 *Properties of graphs.*

a) Every undirected graph without loops has at most $\binom{n}{2}$ many edges, as there are exactly as many subsets of size two (edges) of a set with n elements (vertices). If loops are allowed, there are n additional edges. That is, an undirected graph with loops has at most

$$\binom{n}{2} + n = \frac{n(n-1)}{2} + \frac{2n}{2} = \frac{n(n+1)}{2} = \binom{n+1}{2} \tag{1}$$

edges.

A directed graph with loops has at most n^2 edges (every vertex has one edge to every other vertex). In a directed graph without loops, every vertex is connected to most $n-1$ other vertices. Therefore, such a graph has at most $n(n-1)$ edges.

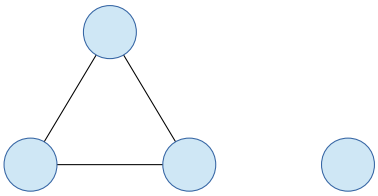
b) *Base case* ($n = 1$): Every graph with exactly one vertex is connected. It is acyclic if and only if it has no loop and, hence, no edges. Therefore, every tree with $n = 1$ has $n - 1 = 0$ edges.

Induction hypothesis: Every tree with n vertices has exactly $n - 1$ edges.

Inductive step ($n \to n + 1$): Consider an arbitrary tree $T = (V, E)$ with $n + 1$ vertices. First, we show that every tree has at least one vertex with degree 1. Suppose that there is a tree containing only vertices with degree 2 or higher. Then, we could choose an arbitrary vertex, visit one of its neighbors using an edge that has not been used before and proceed from this vertex in the same way. After at most n steps we would visit a vertex that we already visited earlier. That is, the graph would not be acyclic and, therefore, not a tree.

Let v be an arbitrary vertex in T with degree 1 and let $e = \{v, w\}$ be the corresponding edge incident to v. As v has degree 1, there is exactly one such edge e. If we remove v and e from T, we get a graph $T' = (V', E')$ with $V' = V \backslash \{v\}$ and $E' = E \backslash \{\{v, w\}\}$. This graph has to be connected and acyclic (otherwise T would not be connected or not be acyclic). Therefore, T' is also a tree. As this graph has exactly one vertex less than T, it has – by induction hypothesis – exactly $n - 1$ edges. As T has exactly one edge more than T' (namely $\{v, w\}$), we know that T has $n - 1 + 1 = (n + 1) - 1$ edges. ∎

c) The statement is wrong, as the following, not acyclic graph with $n = 4$ vertices and $n - 1 = 3$ edges shows:



Solution 11.3 *Test for acyclicity.*

a) A cycle with length 1 is a loop. A graph has a loop at vertex v_i if and only the entry $a_{ii} = 1$ in the diagonal of the adjacency matrix. To check, whether the graph has a loop, it suffices to consider all diagonal entries a_{ii} for $i = 1, \ldots, n$ and to check if one of them equals 1.

b) Here, it is sufficient to compute the transitive closure of the graph and then to check, if there exists a diagonal entry with value one. In contrast to the lecture, the closure should only be transitive and not necessarily be reflexive. To compute it we can use the algorithm for computing the reflexive and transitive closure presented in the lecture with a small modification: First, we remove the instruction $a_{k,k} \leftarrow 1$ in step 2. Second, we later check if there is a diagonal entry with value 1.

HAS-CYCLE($A_G = (a_{ij})_{1 \le i,j \le n}$)

```
1 for k ← 1, ..., n do
2     for i ← 1, ..., n do
3         for j ← 1, ..., n do
4             if a_ik = 1 and a_kj = 1 then a_ij ← 1
5 for k ← 1, ..., n do
6     if a_kk = 1 then return "found a cycle"
7 return "acyclic"
```

One can immediately see that the running time of the algorithm is still in $\Theta(n^3)$.

c) For this exercise we need both the given adjacency matrix A_G and the transitive closure (denoted by B_G for the rest of the exercise) computed by the algorithm in b). An entry b_{ij} equals 1 if and only if there is a directed path from i to j.

We observe: If a diagonal entry b_{kk} in B_G equals 1, then there is a cycle (and especially also a simple cycle), that contains v_k. Hence, v_k has at least one neighbor v_j, such that v_k can be reached starting from v_j. Therefore, there is also a j with $a_{kj} = 1$ and $b_{jk} = 1$. The vertex v_j has again a neighbor, starting from which v_k can be reached. To find an arbitrary cycle, it is sufficient to start from v_k and then follow an *arbitrary* neighbor, starting from which v_k can be reached, in each step. Such a neighbor always exists, as the currently considered vertex is part of a cycle that contains v_k.

It can happen that we reach a vertex that we have visited earlier. In this case, we have found another cycle that we can return. We only have to store the order, in which we have visited the vertices, i.e. we store the traversed path W. We can extract the corresponding cycle from this path. For a path W and a vertex v_i, we denote in the following by $W \oplus \langle v_i \rangle$ the path that is obtained by adding v_i to the end of W.

EXTRACT-CYCLE($A_G = (a_{ij})_{1 \leq i,j \leq n}, B_G = (a_{ij})_{1 \leq i,j \leq n}$)

```
1  for j ← 1, ..., n do VISITED[j] ← 0
2  i ← k                                  ▷ currently considered vertex
3  W ← ⟨v_k⟩                              ▷ so far considered path
4  repeat
5      VISITED[i] ← 1                     ▷ mark current vertex as visited
6      j ← 1                              ▷ search for a neighbor v_j of v_i,
7      while a_ij = 0 or b_jk = 0 do      ▷ such that v_k can be reached from v_j
8          j ← j + 1
9      i ← j                              ▷ Continue with v_j
10     W ← W ⊕ ⟨v_i⟩                       ▷ Add the current vertex to the path
11 until VISITED[i] = 1
12 Iteratively remove the first vertex of W until it is exactly v_i.
13 return W
```

As we mark a new vertices as visited in every step, the loop in lines 4-11 is executed at most n times. The steps 1, 7, and 8 (together) as well as step 12 need both $\mathcal{O}(n)$, all other steps can be executed in constant time. Therefore, the total running time of this algorithm is in $\mathcal{O}(n^2)$.