Eidgenössische Technische Hochschule Zürich

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

Departement of Computer Science
Markus Püschel, David Steurer
Johannes Lengler, Gleb Novikov, Chris Wendler

18. November 2019

Algorithms & Data Structures — Exercise sheet 9 — HS 19

Exercise Class (Room & TA):	
Submitted by:	
Peer Feedback by:	
Points:	

Submission: On Monday, 25 November 2019, hand in your solution to your TA *before* the exercise class starts. Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

Remark. Some material for this sheet will be covered in the lecture on Tuesday, 19.11. In the following we already provide the definitions that are necessary for this sheet.

Definition 1. A graph is called *empty* if it does not contain any edge. A graph is called *complete* if every pair of vertices is adjacent to each other.

Definition 2. A *cycle* is a sequence $v_1, \dots, v_k, v_{k+1}$ with $k \geq 3$ such that all $v_1, \dots, v_k$ are distinct, $v_1 = v_{k+1}$ and such that any two consecutive vertices are adjacent. A graph is called *acyclic* (or a *forest*) if does not contain cycles.

Definition 3. A graph is *connected* if there is a path between every pair of vertices.

Definition 4. A graph which is acyclic and connected is called a *tree.*

Definition 5. A *leaf* in a graph is a vertex of degree one.

Exercise 9.1 *Forests* **(1 point)**.

a) Show that any longest path in a nonempty forest connects two different leaves of this forest.

b) Assume that a forest $G = (V, E)$ has k vertices of odd degree, where $k \geq 0$ (notice that k has to be even). Show that E is the union of (the edge sets of) $k/2$ edge-disjoint paths. Note that the paths do not need to be vertex-disjoint.

Exercise 9.2 *Trees.*

a) Show that in any tree there is a unique path between any pair of two different vertices u and v.

b) Show that adding an edge between any pair of non-adjacent vertices of a tree creates a cycle.

c) Prove by mathematical induction that the number of edges in a tree with n vertices is $n - 1$.

Hint: *For the inductive step, start with a tree of $n+1$ vertices, and find a way to construct a tree of n vertices. For this construction, it is helpful to use Exercise 9.1 a).*

d)* Prove that every connected graph G with n vertices and $n-1$ edges is a tree.

Exercise 9.3 *Cycles and minimal vertex degree* **(1 point)**.

a) Assume that minimal vertex degree of a graph G is $d > 1$. Show that G contains a cycle of length at least $d+1$.

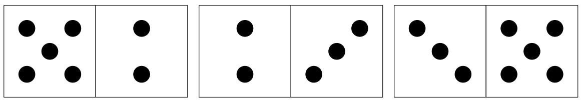
Hint: *Consider some longest path in G.*

b) For every integer $d > 1$ provide an example of a graph G with minimal vertex degree d that does not contain any cycle of length strictly greater than $d+1$.

Exercise 9.4 *Domino* **(1 point)**.

a) A domino set consists of all possible $\binom{6}{2} + 6 = 21$ different tiles of the form $[x|y]$, where x and y are numbers from $\{1, 2, 3, 4, 5, 6\}$. The tiles are symmetric, so $[x|y]$ and $[y|x]$ is the same tile and appears only once.

Show that it is impossible to form a line of all 21 tiles such that the adjacent numbers of any consecutive tiles coincide.



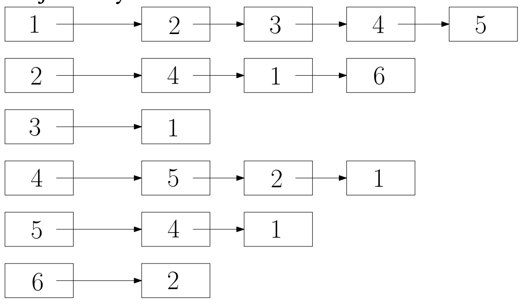
b) What happens if we replace 6 by an arbitrary $n \geq 2$? For which n is it possible to line up all $\binom{n}{2} + n$ different tiles along a line?

Exercise 9.5 *Data structures for graphs.*

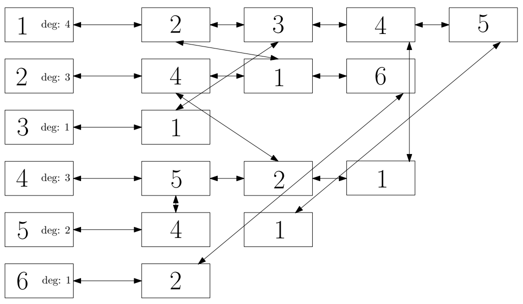
Consider three types of data structures for storing a graph G with n vertices and m edges:

a) Adjacency matrix.

b) Adjacency lists:



c) Adjacency lists, and additionally we store the degree of each node, and there are pointers between the two occurences of each edge. (An edge appears in the adjacency list of each endpoint).



For each of the above data structures, what is the required memory (in Θ-Notation)?

Which runtime (worst case, in Θ-Notation) do we have for the following queries? Give your answer depending on n, m, and/or $\deg(u)$ and $\deg(v)$ (if applicable).

(i) Input: A vertex $v \in V$. Find $\deg(v)$.

(ii) Input: A vertex $v \in V$. Find a neighbour of v (if a neighbour exists).

(iii) Input: Two vertices $u, v \in V$. Are u and v adjacent?

(iv) Input: Two adjacent vertices $u, v \in V$. Delete the edge $e = \{u, v\}$ from the graph.

(v) Input: An edge $e = \{u, v\} \in E$, i.e., a pointer to the location of that edge in your data structure. Delete e from the graph.

(vi) Input: Two vertices $u, v \in V$ with $u \neq v$. Insert an edge $\{u, v\}$ into the graph if it does not exist yet. Otherwise do nothing.

(vii) Input: A vertex $v \in V$. Delete v and all incident edges from the graph.

For the last two queries, describe your algorithm.