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.

Solution: Let G be a forest. Consider some longest path $P = v_1, v_2, \dots, v_k$ in G. This path exists since the set of paths is not empty (because G has at least one edge) and lengths of paths are bounded by the number of vertices in G.

Let's prove that the vertices v_1 and v_k have degree 1 in G. Assume without loss of generality that deg $v_1 > 1$ (it cannot be 0 since v_1 has a neighbour v_2). It means that v_1 has at least two different neighbours, so there exists a neighbour u of v_1 different from v_2.

Case 1 If u belongs to P, then $u = v_i$, where $2 < i \leq k$, and $v_i, v_1, v_2, \dots, v_i$ is a cycle in G, which contradicts the fact that T is acyclic.

Case 2 Otherwise, $u, v_1, \dots, v_k$ is a path in G of length $k + 1$, which contradicts the fact that P is a longest path in G. Hence deg $v_1 = 1$.

Using the same argument one can show that deg $v_k = 1$. Therefore, any longest path in a nonempty forest contains at least two leaves.

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.

Solution: We use induction on k. For $k = 0$, the forest must be empty, since every non-empty forest has at least two leaves by part (a). So the statement is true for $k = 0$.

Now assume that $k > 0$. Consider a longest path P in the forest, and consider the graph G' that is obtained from G by removing all edges of P. Then we claim that G' has $k - 2$ vertices of odd degree: The two endpoints of P have degree 1 in G by part a), so they have degree zero in G'. All other vertices decreased their degree by 2 (if they are on P) or by 0 (if they are not on P), so they do not change their parity.

By induction hypothesis, the edge set of G' can be written as the union of $(k - 2)/2 = k/2 - 1$ paths. Together with P, these paths cover exactly the edges of G, as required.

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.

Solution: Suppose there are two different paths P_1, P_2 between u and v. Let x be the first place they diverge. Let y be the next vertex on P_1 that also appears on P_2. Then the vertices between x and y on P_1 do not appear on P_2, and thus the two paths from x to y (on P_1 and P_2) are disjoint except for start- and endpoint. Thus together they form a cycle. This contradicts the acyclic assumption. Thus there is only one path.

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

Solution: There is a path between u and v because a tree is connected by definition. Together with an additional edge from u to v, this will form 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).*

Solution:

- **Base Case:** Let $n = 1$. There is a single node, and there cannot be any edge from it to itself because then there would be a cycle. There are no other nodes to connect, so there must be 0 edges.
- **Induction Hypothesis:** Assume that any tree with n vertices has exactly $n - 1$ edges.
- **Inductive Step:** Consider a tree with $n + 1$ vertices. Remove any vertex of degree 1 from the tree. There must be such a vertex due to Exercise 9.1 a). The resulting graph is still connected because no connecting path can run over a vertex, and it is also still acyclic. Thus the resulting graph is a tree with n vertices. By the induction hypothesis, this tree has $n - 1$ edges. Add the leaf node back to the tree. This adds only one edge to the tree. Thus the full tree has n edges.

  By the principle of mathematical induction, a tree with n vertices has $n - 1$ edges for any positive integer n.

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

Solution: We need to show that every connected graph $G = (V, E)$ with n vertices and $n-1$ edges is acyclic. Assume that it has a cycle $v_1, \ldots, v_s, v_1$. Let $e = \{v_1, v_s\}$. Then we claim that the graph G' with vertex set V and edge set $E' := E \setminus \{e\}$ is still connected. To see this, let $u, w \in V$. We want to show that there is a walk from u to w in G'. We already know that there is a walk W from u to w in G. If the walk does not contain e, then this is also a walk in G', and we are done. Otherwise, we can just replace the edge e by the walk $v_1, \ldots, v_s$ (in this order if we traverse e from v_1 to v_s, and in the reverse order if we traverse e from v_s to v_1). If e appears several times in in the walk then we replace it each time. Hence G' is connected.

If G' contains a cycle, we repeat the procedure described above, i.e. we remove an edge from G' to get a connected graph G''. We repeat this procedure until we get a connected acyclic graph T. T has at most $n-2$ edges (since we removed e and possibly some other edges), but T is a tree and by part c) any tree with n vertices has $n-1$ edges, so we get a contradiction.

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.*

Solution: Let $P = v_1, \ldots, v_k$ be a longest path. Then every edge from v_k must target one of the vertices $v_1, \ldots, v_{k-1}$, since otherwise we could extend the path to the neighbour of v_k. Let v_i be the first vertex among $v_1, \ldots, v_{k-1}$ adjacent to v_k. Then all neighbours of v_k must be among the vertices $v_i, \ldots, v_{k-1}$, so this list contains at least $\deg(v_k) \geq d$ vertices. In particular, if $d > 1$ then $v_i, \ldots v_{k-1}, v_k$ forms a cycle of length at least $d+1$. Note that for $d = 1$ we might have $i = k-1$, and we would not get a cycle since v_{k-1}, v_k, v_{k-1} uses the same edge twice.

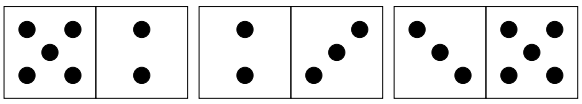
b) For every integer $d > 1$ provide an example of a graph G with minimal vertex degree d that does not contain any cycle of length strictly greater than $d+1$.

Solution: The complete graph K_{d+1} satisfies the condition. Every vertex has degree d, but there is no 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?

Solution: We directly solve the general problem.

First we note that we may neglect tiles of the form $[x|x]$. If we have a line without them, then we can easily insert them to any place with an x. Conversely, if we have a line with them then we can just remove them. Thus the problem with and without these tiles are equivalent.

Consider the following graph G with n vertices, labelled with $\{1, \ldots, n\}$. We represent the domino tile $[x|y]$ by an edge between vertices x and y. Then the resulting graph G is a complete graph K_n, i.e., the graph where every pair of vertices is connected by an edge. A line of domino tiles corresponds to a walk in this graph that uses every edge at most once, and vice versa. A complete line (of *all* tiles) corresponds to an Eulerian walk in G. Thus we need to decide whether $G = K_n$ has an Euler walk or not.

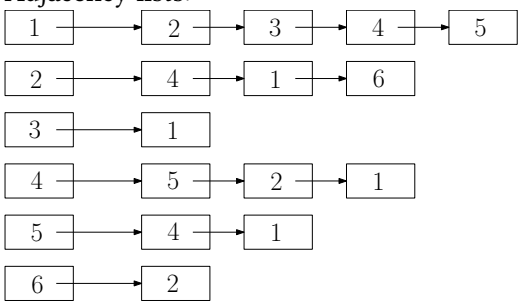
K_n is obviously connected. If n is odd then all vertices have even degree $n - 1$, and thus the graph is Eulerian. On the other hand, if n is even then all vertices have odd degree $n - 1$. If $n \geq 4$ is even, then there are more than 3 vertices of odd degree, and therefore K_n does not have an Euler walk. Finally, for $n = 2$, the graph K_n is just an edge and has an Euler walk. Summarizing, there exists an Euler walk if $n = 2$ or n is odd, and there is no Euler walk in all other cases. Hence, it is possible to line up the domino tiles if $n = 2$ or n is odd, and it is impossible otherwise.

Exercise 9.5 *Data structures for graphs.*

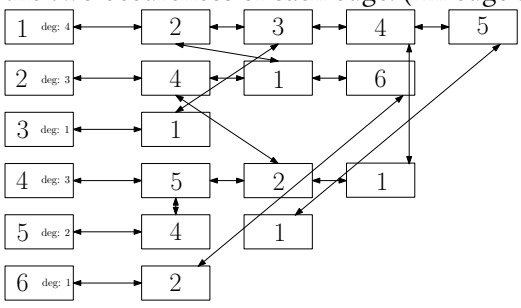
Consider three types of data structures for storing a graph G with n vertices and m edges:

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)?

Solution: $\Theta(n^2)$ for adjacency matrix, $\Theta(n + m)$ for adjacency list and improved adjacency list.

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 $\deg(v)$.

Solution: $\Theta(n)$ in adjacency matrix, $\Theta(\deg(v))$ in adjacency list, $\Theta(1)$ in improved adjacency list.

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

Solution: $\Theta(n)$ in adjacency matrix, $\Theta(1)$ in adjacency list and in improved adjacency list.

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

Solution: $\Theta(1)$ in adjacency matrix, $\Theta(\min\{\deg(v), \deg(u)\})$ in adjacency list and in improved adjacency list.

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

Solution: $\Theta(1)$ in adjacency matrix, $\Theta(\deg(v)+\deg(u))$ in adjacency list and $\Theta(\min\{\deg(v), \deg(u)\})$ in improved adjacency list.

(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.

Solution: We assume that the pointer points to the occurrence of e in the adjacency list of u. Then the runtimes are:
$\Theta(1)$ in adjacency matrix, $\Theta(\deg(v))$ in adjacency list and $\Theta(1)$ in improved adjacency list.

(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.

Solution: $\Theta(1)$ in adjacency matrix, $\Theta(\min\{\deg(v), \deg(u)\})$ in adjacency list and in improved adjacency list.

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

Solution: $\Theta(n^2)$ in adjacency matrix, $\Theta(n+m)$ in adjacency list and $\Theta(n)$ in improved adjacency list.

For the last two queries, describe your algorithm.

Solution: Query (vi): We check whether the edge $\{u, v\}$ does not exist. In adjacency matrix this information is directly stored in the u-v-entry. For adjacency lists we iterate over the neighbours of u and the neighbours of v in parallel and stop either when one of the lists is traversed or when we find v among the neighbours of u or when we find u among the neighbours of v. If we didn't find this edge, we add it: in the adjacency matrix we just fill two entries with ones, in the adjacency lists we add nodes to two lists that correspond to u and v. In the improved adjacency lists, we also need to set pointers between those two nodes, and we need to increase the degree for u and v by one.

Query (vii): In the adjacency matrix we copy the complete matrix, but leave our the row and column that correspond to v. This takes time $\Theta(n^2)$. There is an alternative solution if we are allowed to *rename* vertices: In this case we can just rename the vertex n as v, and copy the n-th row and column into the v-th row and column. Then the $(n-1) \times (n-1)$ submatrix of the first $n-1$ rows and columns will be the new adjacancy matrix. Then the runtime is $\Theta(n)$. Whether it is allowed to rename vertices depends on the context. For example, this is not possible if other programs use the same graph.

In the adjacency lists we remove v from every list of neighbours of every vertex (it takes time $\Theta(n+m)$) and then we remove a list that corresponds to v from the array of lists (it takes time $\Theta(n)$). In the improved adjacency lists we iterate over the neighbours of v and for every neighbour u we remove v from the list of neighbours of u (notice that for each u we can do it in $\Theta(1)$ since we have a pointer between

two occurences of $\{u, v\}$) and decrease $\deg(u)$ by one. Then we remove the list that corresponds to v from the array of lists (it takes time $\Theta(n)$).