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, Ulysse Schaller

16. November 2020

Algorithms & Data Structures Exercise sheet 9 HS 20

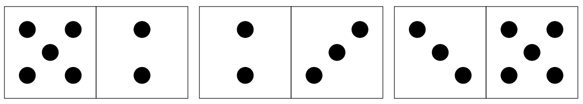
Exercise Class (Room & TA):	
Submitted by:	
Peer Feedback by:	
Points:	

Submission: On Monday, 23 November 2020, 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.

Exercise 9.1 *Domino.*

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 9.2 *Post-Corona Party* **(1 point)**.

It is 2021, we have multiple Corona vaccines and the pandemic is over. Imagine you go to a party with your friends. Assume that people shake hands with each other to introduce themselves.

Prove that at each party with at least two participants there are two people that shook hands with the same number of people.

Hint: *Model the problem as a problem defined on a graph. Define the set of vertices and the set of edges in words.*

Solution: Formally, a party can be seen as an undirected graph $G = (V, E)$, where V is the set of participants and, for $u, v \in V$, the edge $\{u, v\}$ is in E if person u and person v shook hands. The degree of vertex $\deg(v)$ is equal to the number of people that v shook hands with. Thus, we have to prove that for each undirected graph there are two vertices with the same degree.

The degree function takes values between 0 and $n-1$, i.e., for all $v \in V$ we have $\deg(v) \in \{0, 1, \ldots, n-1\}$. Observe that if there is a vertex $v \in V$ with $\deg(v) = 0$, there cannot be a vertex $u \in V$ with $\deg(u) = n - 1$ and vice versa. Thus, for each graph either $\deg(v) \in \{0, \ldots, n-2\}$ or $\deg(v) \in \{1, \ldots, n-1\}$ must hold for all $v \in V$. Because there are more vertices $|V| = n$ than different values of $\deg(v)$ (namely $n - 1$), at least two vertices always must have the same degree.

Exercise 9.3 *Graph connectivity* **(2 points)**.

In this exercise, you will need to prove or find counterexamples to some statements about the connectivity of graphs. We first need to introduce/recall a few definitions.

Definition 1. A *cycle* is a sequence of vertices $v_1, \ldots, v_k, v_{k+1}$ with $k \geq 3$ such that all $v_1, \ldots, v_k$ are distinct, $v_1 = v_{k+1}$ and such that any two consecutive vertices are adjacent.

Definition 2. A graph is *connected* if there is a walk between every pair of vertices. It is called *disconnected* otherwise.

Definition 3. A vertex v in a connected graph is called a *cut vertex* (or *articulation point*) if the subgraph obtained by removing v (and all its incident edges) is disconnected.

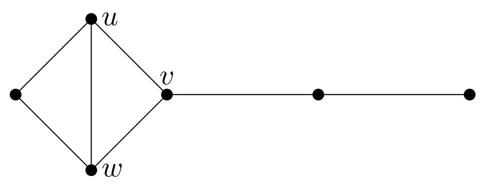
Definition 4. An edge e in a connected graph is called a *cut edge* (or *bridge*) if the subgraph obtained by removing e (but keeping all the vertices) is disconnected.

In the following, we always assume that the original graph is connected. Prove or find a counterexample to the following statements:

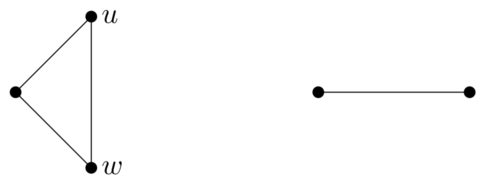
a) If a vertex v is part of a cycle, then it is not a cut vertex.

   Solution:

   The following graph is a counterexample:



Indeed, v is clearly part of a cycle (the triangle uvw for example), but removing v yields the following graph:

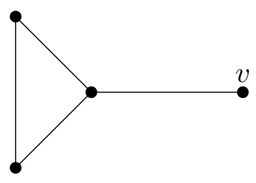


The above graph is disconnected. Hence, v is also a cut vertex.

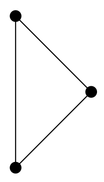
b) If a vertex v is not a cut vertex, then v must be part of a cycle.

Solution:

The following graph is a counterexample:



Indeed, v is not part of a cycle (remember that the vertices forming a cycle must be disjoint). However, removing v yields the following connected graph:



Hence, v is also not a cut vertex.
Remark. The following statement is true: If a vertex of degree at least 2 is not a cut vertex, then it must lie on a cycle. (Proof: Consider two neighbours u_1, u_2 of that vertex v. Since v is not a cut vertex, after removing v there is still a path from u_1 to u_2. Together with the edges $\{u_2, v\}$ and $\{v, u_1\}$, this forms a cycle that contains v.)

c) If an edge e is part of a cycle (i.e. e connects two consecutive vertices in a cycle), then it is not a cut edge.

Solution:

This statement is correct, and we can prove it as follows. Let G be a conncted graph and let $e = \{v_1, v_2\}$ be an edge of G that is part of a cycle $v_1 \dots v_k$ for some $k \geq 3$. To show that e is not a cut edge, we will show that any two vertices u and w of G can be joined by a walk that does not use the edge e. So consider any two vertices u and w. Since G is connected, there is a walk $uu_1 \dots u_n w$ from u to w in G. If the walk doesn't use the edge e, then we are already done. If the walk does use edge e, this means that the vertices v_1 and v_2 must appear consecutively (at least once) in $uu_1 \dots u_n w$. We replace every appearance of $v_1 v_2$ in the walk by the path $v_1 v_k v_{k-1} \dots v_2$, and every apperance of v_2v_1 by the same path in the other direction $v_2v_3\dots v_kv_1$. This yields a walk from u to w that does not use the edge e and concludes the proof.

d) If an edge e is not a cut edge, then e must be part of a cycle.

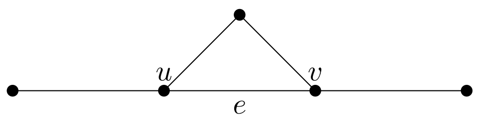
Solution:

This statement is correct, and we can prove it as follows. Let G be a conncted graph and let $e = \{v_1, v_2\}$ be an edge of G that is not a cut edge. Then any pair of vertices in G can be joined by a walk that doesn't use the edge e. In particular, there is a walk from v_1 to v_2 that doesn't use e. Moreover, we can turn this walk into a path. Indeed, for any vertex u that appears more than once in the walk, we just remove the whole walk between its first appearance and its last appearance in the walk. Doing this sequentially along the walk, we obtain a path $v_1, u_1, \dots, u_k, v_2$ from v_1 to v_2 that does not use e. In particular, this path must contain at least 3 vertices (the only way to have a path from v_1 to v_2 using 2 vertices is by using the edge e). We can then close this path with the edge e to form a cycle $v_1, u_1, \dots, u_k, v_2, v_1$, and hence e is part of a cycle.

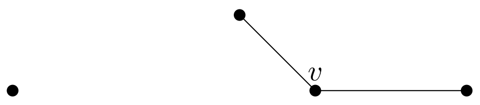
e) If u and v are two adjacent cut vertices, then the edge $e = \{u, v\}$ is a cut edge.

Solution:

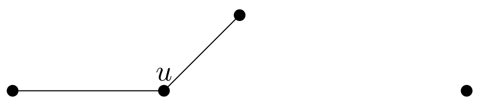
The following graph is a counterexample:



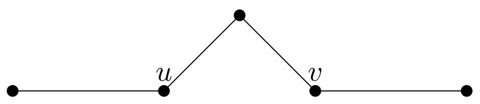
Indeed, removing u yields



while removing v yields



and both of these graphs are disconnected, which means that u and v are cut vertices. However, removing the edge $e = \{u, v\}$ yields the following connected graph:

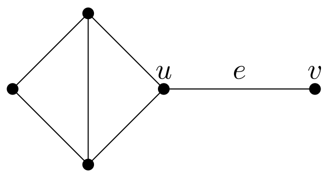


Hence, e is not a cut edge.

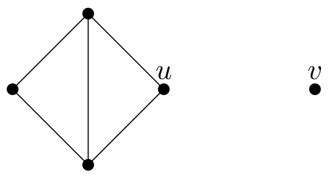
f) If $e = \{u, v\}$ is a cut edge, then u and v are cut vertices. What if we add the condition that u and v have degree at least 2 ?

Solution:

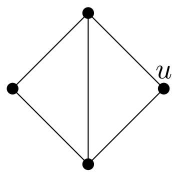
The following graph is a counterexample:



Indeed, removing e yields the following graph:



The above graph is disconnected, so e is a cut edge. However, removing v yields the following connected graph:



Hence, v is not a cut vertex.

If we add the condition that u and v have degree at least 2, then the statement is actually correct. We will only show that u is a cut vertex, since the proof that v is also a cut vertex is exactly the same with the two vertices exchanged.

Since $\deg(u) \geq 2$, u must have a neighbor $u' \neq v$ in the original graph G. We claim that after removing u and all its incident edges, there is no walk from u' to v in the obtained subgraph G', which means that it is disconnected and thus u is indeed a cut vertex.

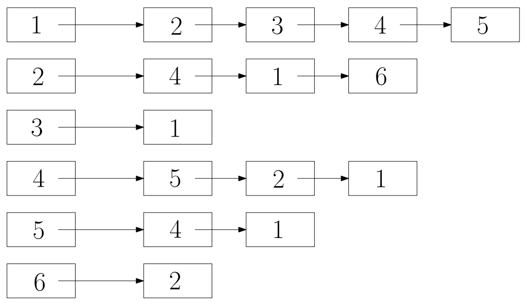
Suppose by contradiction that there is a walk from u' to v in G'. Using the same trick as in part d), we can turn this walk into a path π from u' to v. Since $u' \neq v$, this path must use at least one edge. Since every edge incident to u in G was removed to create G', π does not use any edge incident to u, and in particular it does not use the edge e nor the edge uu'. But then we obtain a cycle in G by concatenating the path π with the edge e and then uu' (note that this cycle indeed contains at least 3 edges, and hence 3 vertices). So e is part of a cycle in G, and by part c) it therefore cannot be a cut edge. This is the desired contradiction.

Exercise 9.4 *Data structures for graphs.*

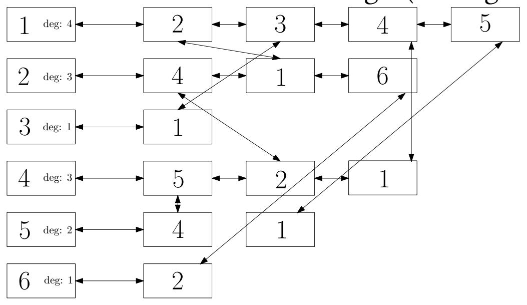
Consider three types of data structures for storing a graph G with n vertices and m edges:

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, $\Theta(1+\deg(v))$ 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 $u, v \in V$. Decide whether u and v are adjacent.

Solution: $\Theta(1)$ in adjacency matrix, $\Theta(1+\min\{\deg(v), \deg(u)\})$ 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: A vertex $u \in V$. Find a neighbor $v \in V$ of u and delete the edge $\{u, v\}$ from the graph.
Solution: $\Theta(n)$ in the adjacency matrix ($\Theta(n)$ for finding a neighbor and $\Theta(1)$ for the edge deletion).

$\Theta(1+\max_{w:\{u,w\}\in E} \deg(w))$ for the adjacency list ($\Theta(1)$ for finding a neighbor and $\Theta(\max_{w:\{u,w\}\in E} \deg(w))$ for the edge deletion).

$\Theta(1)$ for the improved adjacency list ($\Theta(1)$ for finding a neighbor and $\Theta(1)$ for the edge deletion).

(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, $\Theta(1 + \min\{\deg(v), \deg(u)\})$ 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 out 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 $\Theta(n)$).

Exercise 9.5 *Walks in a graph.*

Recall that a *walk of length k* in a graph $G = (V, E)$ is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that for $1 \leq i \leq k$ there is an edge $\{v_{i-1}, v_i\} \in E$.

Assume that you are given a graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$, and a matrix M that contains for every pair $v_i, v_j \in V$ in the entry M_{ij} the number of walks of length k from v_i to v_j. For every pair $v_k, v_\ell \in V$, find a formula for computing the number of walks of length $k+1$ from v_k to v_ℓ from that data.

Solution: The walks of length k can be extended to walks of length $k+1$ by adding an edge. Formally, the number of walks of length $k+1$ from v_i to v_j is given by

$$M_{ij}^{+} = \sum_{v_l \in \mathcal{N}(v_j)} M_{il}, \tag{1}$$

where $\mathcal{N}(v_j) = \{v_l \in V : \{v_l, v_j\} \in E\}$ denotes the neighbors of v_j.