Eidgenössische
Technische Hochschule
Zürich

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

Department Informatik
Markus Püschel
David Steurer

Karel Kubicek
Johannes Lengler

Programming Exam

Algorithmen und Datenstrukturen

January 27, 2020

DO NOT OPEN!

Last name, first name: ______________________________

Student number: ______________________________

With my signature I confirm that I can participate in the exam under regular conditions. I will act honestly during the exam, and I will not use any forbidden means.

Signature: ______________________________

Good luck!

The **ENROLLMENT PASSWORD** is “`mergesort`”.

1 Relations and Directed Graphs, 24 points

You are given a directed graph $G = (V, E)$ with $V = \{0, \ldots, n-1\}$ represented by an adjacency matrix `diGraph`, and your task is to implement 3 methods on it.
For two of these methods, the edge set E will represent a relation on V. Recall that a relation on R is given by a set $R \subseteq V \times V$, and we set $R := E$. In other words, $(a, b) \in R$ if and only if $(a, b) \in E$.

As a reminder, here are two definitions from Diskrete Mathematik (but note that this exercise is self-contained):

- The relation R is **antisymmetric**, when

$$\forall a, b \in V : \Big((a, b) \in R \wedge (b, a) \in R \implies a = b\Big).$$

- The relation R is **transitive**, when

$$\forall a, b, c \in V : \Big((a, b) \in R \wedge (b, c) \in R \implies (a, c) \in R\Big).$$

You should implement the following three methods (see next page for examples):

a) `isAntisymmetricRelation()` tests if the relation $R = E$ is antisymmetric.

   You can get **4 points** with an algorithm of **runtime $\mathbf{O(|V|^2)}$**.

b) `isTransitiveRelation()` tests if the relation $R = E$ is transitive.

   You can get **4 points** for **runtime $\mathbf{O(|V|^3)}$**

c) `containsStar` tests if the graph contains a *star*. A star is a vertex which has indegree $|V| - 1$ and outdegree 0 and no self-loop.

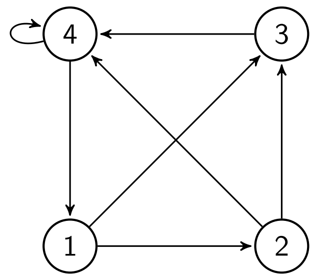
   The input is a sorted array of edges, and as such, it is in a separate class. The sorting is primarily according to the source vertex, secondarily according to the target vertex. E.g., the edge $(1, 5)$ comes before $(2, 3)$, and that comes before $(2, 4)$.

   You can get **2 points** with an algorithm of **runtime $\mathbf{O(|V| \cdot |E|)}$** and **6 points** (in total) for **runtime $\mathbf{O(|E|)}$**.

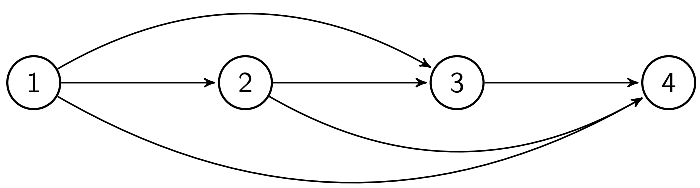
d) `containsCycle()` tests if the graph contains a directed cycle.

   You can achieve up to **10 points** with a correct algorithm of **runtime $\mathbf{O(|V|^2)}$**. You can get partial points for a set of particularly easy instances (**2 points**), or for **runtime $\mathbf{O(|V|^3)}$** (**2 points**). The remaining **6 points** are for the generic case.

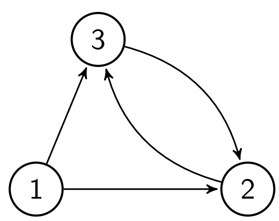
Examples:



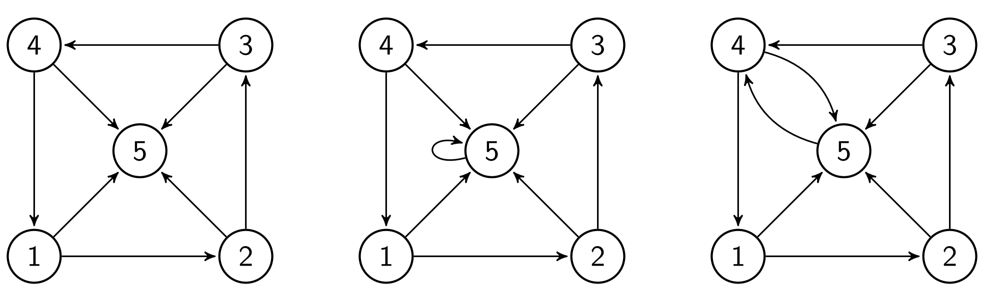
- `isAntisymmetricRelation()` returns `true`.
- `isTransitiveRelation()` returns `false`.
- `containsCycle()` returns `true`.



- `isAntisymmetricRelation()` returns `true`.
- `isTransitiveRelation()` returns `true`.
- `containsCycle()` returns `false`.



- `isAntisymmetricRelation()` returns `false`.
- `isTransitiveRelation()` returns `false`.
- `containsCycle()` returns `true`.



- `containsStar()` returns (from left to right) `true`, `false` (star cannot have self-loop), and `false` (star cannot have outgoing edge).

2 String splitter, 16 points

You are given a string s that contains only characters `a`, `b`, ..., `z` and no spaces. You are also given a dictionary D of words using the same characters. Your task is to find the length of the longest initial sequence s^* of s (i.e., starting from position 0) that can perfectly be split into words from the dictionary. I.e., s^* can be partitioned into consecutive substrings that are all in D.

Words from the dictionary may be used more than once.

Several **Examples:**

- For string $s =$ `aabb` and dictionary { `b`, `c` } the output is 0. (You have to start from the beginning and `a` is not in the dictionary).
- For string $s =$ `aabb` and dictionary { `a`, `c` } the output is 2. (The initial string $s^* =$ `aa` can be split in the form `a|a`, where `|` marks split).
- For string $s =$ `aabb` and dictionary { `a`, `b` } the output is 4. (The initial string $s^* = s =$ `aabb` can be split in the form `a|a|b|b`).
- For string $s =$ `aabb` and dictionary { `a`, `abb` } the output is 4. (The initial string $s^* = s =$ `aabb` can be split in the form `a|abb`).

We use the following notation.

- $|s|$ is the length of the string s.
- $|D|$ is the total number of characters in the dictionary.

The tests contains 4 **test sets**:

- `unique`: the input dictionary is composed of words with unique last character. **2 points**.
- `small`: the general case, but the time limit is generous. **3 points**.
- `large`: the general case, but your solution has to be efficient. A runtime of $\mathcal{O}(|s| \cdot |D|)$ is fast enough, but other efficient implementation with different asymptotic runtimes may also be accepted. **9 points**.
- `extra_large`: a challenge. The input is the same as "large", but an asymptotically even faster runtime is required. **2 points**.

Space for Your Notes:

Space for Your Notes:

Space for Your Notes:

Space for Your Notes:

Space for Your Notes: