ETH 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

August 7, 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 “`formula`”.

1 Undirected graph, 24 points

You are given an undirected graph G represented by an adjacency matrix `graph`, which is symmetric, as seen in tasks during the semester. We will use the following graph terminology:

- A graph is **k-regular** if and only if each vertex has exactly k neighbors, i.e. every vertex has degree k.
- Given two vertices u and v of G, their **graph distance** $d(u, v)$ is the minimal number of edges that a path connecting u and v has. If there is no such path, the distance is infinity, but for programming purposes, we will define it as -1.
- The graph **diameter** is the maximal distance $d(u, v)$ between any two vertices u, v in G. If the graph contains at least 2 disconnected components, the diameter is infinity (i.e., -1 in the code).

Your task is to implement the following 4 methods.

- `int isKRegular()` returns k if the graph is k-regular for some $k \in \mathbb{N}_0$. Otherwise, it returns -1.

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

- `boolean hasTriangle()` tests, if the graph contains a triangle, i.e., a cycle of length 3. If the graph contains a triangle, it returns `true`, otherwise it returns `false`.

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

- `int getGraphDiameter()` returns the diameter of the graph.

  You can get up to **10 points** for a correct algorithm of **runtime** $\mathcal{O}(|\mathbf{V}|^{\mathbf{3}})$. You can get partial points for a set of particularly easy instances (**2 points**), and/or for runtime $\mathcal{O}(|\mathbf{V}|^{\mathbf{4}})$ (**2 points**). The remaining 6 points are for the generic case.

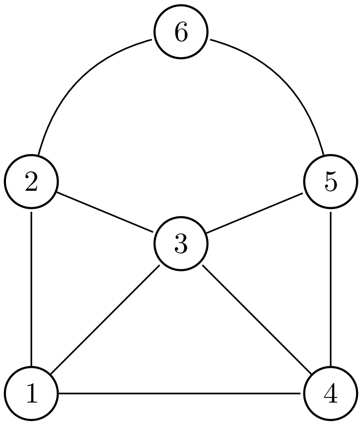
- `boolean dominoSequence(Tile[] tiles)`: In this task, the input is not a graph, as in the previous tasks! Instead, you are given an array of domino tiles in the form $\{a, b\}$, where $a, b \in \mathbb{N}$. The tiles are unique (no duplicates) and their order is random. Your task is to decide if these tiles can form a single sequence such that two adjacent tiles are connected by the same number. If the tiles can form the described sequence, return `true`, else return `false`. You can use the provided function `boolean containsEulerianWalk(Graph G)` that for the given undirected graph G, given as before by an adjacency matrix, determines if it contains an Eulerian walk.

  Note that the tiles are symmetric, so the tile $\{a, b\}$ is identical to the tile $\{b, a\}$.

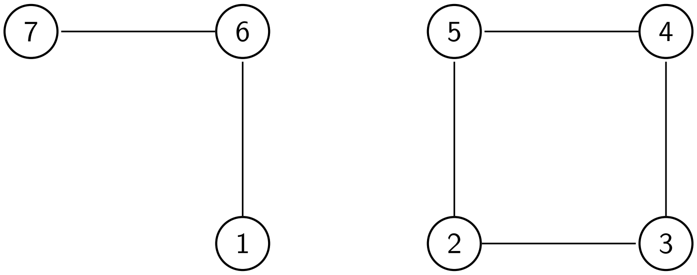
  Example: for `tiles` $= \{\{1, 2\}, \{1, 3\}, \{1, 1\}, \{3, 6\}, \{5, 2\}\}$ the answer is `true`, as the tiles can form a sequence: $(\{5, 2\}, \{2, 1\}, \{1, 1\}, \{1, 3\}, \{3, 6\})$.

  You can get **6 points** for an algorithm of **runtime** $\mathcal{O}(|\mathbf{V}|^{\mathbf{2}})$. The runtime of `containsEulerianWalk` is $\mathcal{O}(|\mathbf{V}|^{\mathbf{2}})$.

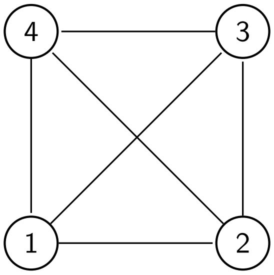
Examples:



- `isKRegular()` returns -1.
- `hasTriangle()` returns `true`.
- `getGraphDiameter()` returns 2.



- `isKRegular()` returns -1.
- `hasTriangle()` returns `false`.
- `getGraphDiameter()` returns -1.



- `isKRegular()` returns 3.
- `hasTriangle()` returns `true`.
- `getGraphDiameter()` returns 1.

2 DNA sequence alignment, 16 points

A DNA sequence is a string of characters from a four-character alphabet $\{A, T, G, C\}$. For a pair of two strings x and y, an *alignment* is given by inserting gaps into both x and y at arbitrary places until they have the same length. Each insertion of a gap costs 2. Afterwards, for each position, we have an additional cost of 1 for each position in which the extended strings do not match. Thus the aligning operations and their costs are:

Inserting a gap costs 2.

Aligning two mismatched characters costs 1.

Aligning two matched characters costs 0.

Your task is to compute the minimal cost c of aligning two DNA sequences x and y.

```
x = - T A G C A G T T A C C
y = C T A G A G G T C A - -
c = 2+0+0+0+1+1+0+0+1+0+2+2 = 9
```

```
x = - T A G C A G T T A C C
y = C T A G - A G G T - C A
c = 2+0+0+0+2+0+0+1+0+2+0+1 = 8
```

The second alignment in this example is actually the one with minimal cost possible. So your algorithm should output "8" in this case.

We prepared two **test sets**:

- `small`: a generic test case with generous time limit. **5 points**.
- `large`: a generic test case, but your solution has to be efficient. A runtime of $\mathcal{O}(\text{length}(x) \cdot \text{length}(y))$ is fast enough, but other efficient implementation with different asymptotic runtimes may also be accepted. **11 points**.