The templates required for this exercise can be downloaded here:
https://www.dfki.de/~steffen/advanced-java/graphs2.zip

1 Tarjan's SCC Algorithm (7.5 points)

Implement Tarjan's algorithm for finding strongly connected components as a DFS visitor by completing the provided skeleton class `TarjanVisitor`. The procedure is similar to the `CollectTimesVisitor` from last exercise. After DFS the visitor should contain as result a set of sets of vertices (Integers) that represent the SCCs.
Hints:

- The `else-if` case in the pseudo code from the lecture means that a non-tree edge is used.
- Make sure to remove the `low` map entry for a vertex when you pop it from the stack! This is required to avoid any interference with later DFS runs.

Write a unit test using the following graph. This yields the SCCs from the lecture example:

```
s --> z w
z --> y w
y --> x
x --> z
w --> q x
q --> x
t --> v u
u --> v t
v --> w s
```

2 Dijkstra's SSSP Algorithmus (7.5 points)

Implement Dijkstra's single source shortest path algorithm for a pair of vertices (source and target) by completing the provided skeleton class `DijkstraShortestPath`. Use `Integer` as edge info `<E>`. Abort when the target node is reached and return the best path as a list of **edges** (not as list of vertices as described in the lecture). Implement Q with a `PriorityQueue`, and accept the disadvantage of `lower_key` when adjusting the distance.

Write a unit test to find the shortest path between `s` and `u` using the following graph:

```
s --> w(2) z(4)
z --> w(5) y(9)
v --> w(1) s(6)
w --> x(6) q(3)
t --> u(1) v(9) s(8)
u --> t(5) v(7)
x --> z(8) u(7)
q --> x(2)
y --> x(5)
r --> s(1)
```