Routing

Submission deadline:	**2019-05-05 23:59:59**
Late submission with malus:	**2019-06-30 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**5.0000**
Max. assessment:	**5.0000** (Without bonus points)
Submissions:	1 / 20 Free retries + 20 Penalized retries (-2 % penalty each retry)
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to develop a generic class `CRoute` that can find a path connecting two given places.

We assume a map with nodes (e.g. cities, computers, ...) and edges that connect node pairs (e.g. roads, railroads, network links, ...). Class `CRoute` stores the map. Since the map is general, the data types describing nodes and edges will be implemented as generic parameters.

The class will provide a method to insert edges and a method to search a path. The exact interface is:

default constructor
: prepares an empty `CRoute` instance.

method Add (u1, u2, e)
: this method adds a new edge connecting nodes `u1` and `u2`. The third parameter `e` is the description of the edge, e.g. the speed of the line, type of road, ... The edge is assume two-way edge.

method Find (u1, u2, [,f])
: this method will search the map to find a path from `u1` to `u2`. The result is a list (`std::list`) with the nodes along the path from `u1` to `u2`. The first node in the list shall be `u1`, the last `u2`.

: It may be the case that there is no path connecting `u1` to `u2`. In that case the method throws `NoRouteException`. Moreover, there may exist many different ways to get from `u1` to `u2`. In that case, the method returns the shortest path (the path with the lowest number of intermediate nodes). Finally, if there are several options of the path with the lowest number of intermediate nodes, the result may be any of them.

: The method shall be provided in two variants: with and without the third optional parameter `f`. If present, the third parameter acts as a filter that enables/disables individual edges. The filter is provided in the form of a function/functor/lambda expression. It shall be applied to the edges. If the filter returns `false`, the edge must not be included in the resulting path. Thus, the filter may be used to remove certain edges from the search, i.e. we may consider only edges with some properties (speed, quality, ...).

Generic parameter `_T` is the data type that represents nodes. Your implementation may use the following operations with _T:

- copy constructor,
- operator =,
- relational operators (==, !=, <, <=, > >=),
- destructor,
- output operator <<.
- Further operations may exist, however, they are not guaranteed.

Generic parameter `_T` is the data type that represents edges. Your implementation may use the following operations with _E:

- copy constructor,
- operator =,
- destructor.
- Further operations may exist, however, they are not guaranteed.

Submit a source file with the implementation of the `CRoute` class template and `NoRouteException` exception class. Use the attached source code as a basis for your implementation. Please preserve the conditional compile directives - if present, the source may be locally tested and submitted to Progtest without manual modifications.

The assessment is divided into mandatory and optional tests. The mandatory tests use only small maps with small number of nodes. The optional test tests maps with high number of nodes, thus the implementation requires some efficient data structures to finish in the time limit. A significant penalty will apply if the optional test is not passed.

Use BFS (breadth first search) algorithm to search the map. Further inspiration: PA1, proseminar #11, labyrinth problem.

The example source code may be found in the attached archive.

The solution of this homework cannot be used for code review.

Sample data:	**Download**

☐ **Reference**

1	**2019-04-29 15:18:31**	**Download**
Submission status:	Evaluated	
Evaluation:	5.0000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.001 s (limit: 5.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.139 s (limit: 4.999 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Speed test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 3.524 s (limit: 8.000 s)
    - Optional test success, evaluation: 100.00 %
  - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00)
- Total percent: 100.00 %
- Total points: 1.00 * 5.00 = 5.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**13**	--	--	--
Lines of code:	**155**	**11.92 ± 16.34**	**56**	`main`
Cyclomatic complexity:	**29**	**2.23 ± 2.36**	**10**	`Find`