6th Practical Class – Graphs: Using the graph visualization API

Instructions

- Download file **cal_fp06_CLion.zip** from the Moodle area of the course unit (it contains folder **lib,** folder **resources** with files of a map, and files **CMakeLists** and **FichaJUNG.cpp** which implements the main() function. IMPORTANT: this exercise does not use unit tests!
- In CLion, open a ***project***, selecting the folder containing the files as described above.
- Do "*Load CMake Project*" over the file *CMakeLists.txt*
- Run the project (**Run**)
- ***You should implement the exercises following the order suggested***.
- Implement your solutions in the respective function stubs, in file FichaJUNG.cpp.
- To access the data for the example map use relative paths, since the folder **resources** is included in file CMakeLists.txt already:

  Vertices data: `"../resources/mapa1/nos.txt"`

  Edges data: `"../resources/mapa1/arestas.txt"`
- The parts to be coded in file **FichaJUNG.cpp** are marked with **TODO** and, in some cases, there are some explanations and hints on how to implement them.
- This exercise uses *sockets*. If you are implementing it under Windows, you should include the Wsock32 library, by uncommenting the following line, if file **CMakeLists.txt**, in the case it is commented:
  `link_libraries(ws2_32 wsock32)`

Exercises

1. Base structure of a graph

a) Configure your development environment

i. Create a 600*600 window

Note: to create a window you should use the following code
```
GraphViewer *gv = new GraphViewer(600, 600, true);
gv->createWindow(600, 600);
```

ii. Configure the vertex color to *blue*

Note: to configure the vertices' color you should use the following code:
```
gv->defineVertexColor("blue");
```

iii. Configure the edge color to *black*

Note: to configure the edges' color you should use the following code:
```
gv->defineEdgeColor("black");
```

b) Create a vertex

i. Create a vertex with the following attributes:

Id: 0

Note: to create a vertex you should use the following code
gv->addNode(idVertex);
Note: to update the graph with your changes you must run the following method:
gv->rearrange();

ii. Create a vertex with the following attributes:

Id: 1

iii. Create an edge between the previously created vertices.

Note: to create edges, use the following code:
// for bidirectional edges
gv->addEdge(idEdge,idSourceVertex,idDestinationVertex,EdgeType::UNDIRECTED);

// for directed edges
gv->addEdge(idEdge,idSourceVertex,idDestinationVertex, EdgeType::DIRECTED);

iv. Remove vertex 1

Note: to remove a vertex, run the following method:
gv->removeNode(idVertex);

v. Add a new vertex with the following attributes:

Id: 2

vi. Add an edge between the two previously created vertices

vii. Add a label to vertex 2 with a text of your choosing

Note: to add a label, use the following code:
gv->setVertexLabel(idVertex, "This is a vertex");

viii. Add a label to an edge with a text of your choosing

Note: to add a label to an edge use the following code:
gv->setEdgeLabel(idEdge, "This is an edge");

ix. Make vertex 2 green

Note: tu configure a vertex's color, use the following code:
gv->setVertexColor(idVertex, "green");

x. Make the edges yellow

Note: to configure an edge's color use the following code:
gv->setEdgeColor(idEdge, "yellow");

xi. Make the "background.jpg" image the background

Note: to configure the background image use the following code
//must be used before gv->createWindow(600, 600);
gv->setBackground("background.jpg");

2. Graph animations simulation.

a) Add vertices with the following attributes:

id: 0, x: 300, y: 50
id: 1, x: 318, y: 58
id: 2, x: 325, y: 75
id: 3, x: 318, y: 93
id: 4, x: 300, y: 100
id: 5, x: 282, y: 93
id: 6, x: 275, y: 75
id: 7, x: 282, y: 58
id: 8, x: 150, y: 200
id: 9, x: 300, y: 200
id: 10, x: 450, y: 200
id: 11, x: 300, y: 400
id: 12, x: 200, y: 550
id: 13, x: 400, y: 550

Note: in order to manually set the positioning of vertices, initialize GraphViewer with the last argument set to *false*:
GraphViewer *gv = new GraphViewer(600, 600, false);

b) Add edges with the following attributes

id: 0, idSourceVertex: 0, idDestinationVertex: 1
id: 1, idSourceVertex: 1, idDestinationVertex: 2
id: 2, idSourceVertex: 2, idDestinationVertex: 3
id: 3, idSourceVertex: 3, idDestinationVertex: 4
id: 4, idSourceVertex: 4, idDestinationVertex: 5
id: 5, idSourceVertex: 5, idDestinationVertex: 6
id: 6, idSourceVertex: 6, idDestinationVertex: 7
id: 7, idSourceVertex: 7, idDestinationVertex: 0
id: 8, idSourceVertex: 4, idDestinationVertex: 9
id: 9, idSourceVertex: 9, idDestinationVertex: 8
id: 10, idSourceVertex: 9, idDestinationVertex: 10
id: 11, idSourceVertex: 9, idDestinationVertex: 11
id: 12, idSourceVertex: 11, idDestinationVertex: 12
id: 13, idSourceVertex: 11, idDestinationVertex: 13

c) Animation

Note: in order to simulate graph animation, you should alter the graph as you wish, and then order a re-draw. So that the animation is perceptible, you can pause the execution between re-draws (sleep(numSeconds) in Linux and Sleep(numMiliSeconds) in Windows).

i. Remove nodes 12 and 13

ii. Add nodes with the following attributes

id: 14, x: 250, y: 550
id: 15, x: 350, y: 550

d) Based on the concepts you have learned so far, make the animation progressive and cyclic

Note: Due to the way the API currently works, it is not possible to change a node's position. You should remove the node and add a new one with a different identifier and the intended position.

3. Load a graph from a file.

a) Read the nodes.txt and edges.txt files to load the graph represented in them

The files use the following format:
Each line of the nodes.txt file represents a vertex

- idNode;X;Y

Each line of the edges.txt file represens an edge

- idEdge;idSourceVertex;idDestinationVertex