COMP.CS.300 Data structures and algorithms 1, fall 2023

Programming assignment 2: ScholarNet

Last modified on 27. marrask. 2023

Change log

- Changes that have been made will be listed here.
- Clarified the order of ids in connection structs for the compulsory functions. The order of ids in connections for the optional functions is the same as in get_any_path 16. marrask. 2023
- Added pictures of example paths A-D 20. marrask. 2023
- PathWithDist in get_shortest_path, 23. marrask. 2023
- Added more explanation to friction 27. marrask. 2023

The Assignment 2 focuses on graphs

Assignment 2 expands upon Assignment 1 by creating the **ScholarNet** affiliation graph. This graph connects affiliations based on shared publications. The graph enables the implementation of new algorithms, such as the shortest path between affiliations..

Some functions are compulsory. Assignment 2 fails unless all compulsory functions are implemented. While additional functions are optional, implementing them can positively impact your grade. Each efficiently implemented function contributes to a higher grade.

This document specifically covers the new features introduced in Assignment 2. All the content from Assignment 1 is also available in Assignment 2. However, the work done in Assignment 1 does not influence your grade for Assignment 2; assignments are evaluated independently.

You can start by copying your implementations from Assignment 1 as a foundation. Alternatively, you have the flexibility to begin from scratch if you prefer.

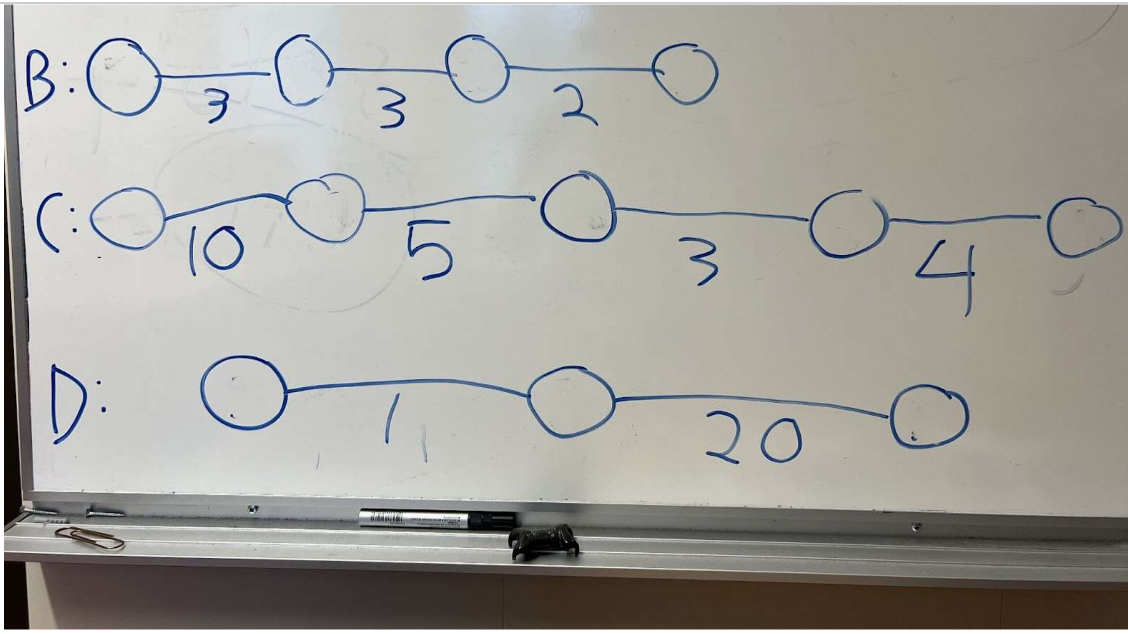
Terminology

New terms:

- **Connection.** A link between two affiliations. The length of a connection is calculated based on the distance between (x, y) coordinates of both affiliations. In addition to geolocation length, the connection has a weight.
- **Weight.** Each common publication between two affiliations increases the weight of the connection between those affiliations by one.

If the friction (given by the equation above) of two paths is the same, the path with least connections has lower friction.

- Examples:
  - A path **A** with two connections with weights 2 and 3 has a minimum connection weight of 2 and the total distance is 2 connections
  - A path **B** with three connections with weights 3, 3 and 2 has the same minimum connection weight (2) but since it has 3 connections, the friction is higher than A
  - A path **C** with four connections with weights 10, 5, 3 and 4 has a minimum connection weight of 3 and the path is 4 connections long, since the minimum connection weight is higher than **A**s or **B**s, the friction of path C is smaller than **A** or **B**
  - A path **D** with two connections with weights 1 and 20 has a minimum connection weight of 1 and a length of 2 connections, since the connection weight is less than any other path in this example, the friction is the highest
- A comparison of friction for paths **A-D:**
- **C<A<B<D**



Here is a graphical representation of the paths A-D, the circles here are Affiliations and the numbers represent a weight of a connection between affiliations. The affiliations have not been named to the picture to simplify it.

- **Path.** A path is a sequence of connections. The length of a path is the sum of connection lengths of the path.
- **Cycle.** A path has a cycle, if it arrives again at the same affiliation where it already visited. It must not go backwards on the same route connection.

In assignment 2, you might not have as much control over the efficiency of new operations; correct behavior is more important than just focusing on how fast the operations are.

- **The implementation will fail if it does not pass all compulsory auto-graders**
    - For example, if the implementation timeouts or crashes before the completion of a test, the assignment will not be graded.
- **The minimum acceptable grade for the compulsory part is 150/1000.**
- **Efficient algorithms may increase the points to 400/1000.**
- **The rest of the points come from the optional part, where each optional function is worth 150p: 4 x 150 p = 600p**

Some parts of the program code will be provided by the prg2 code skeleton, other parts are implemented by students.

Parts provided by the course

The following files are provided ready-made: mainprogram.hh, mainprogram.cc, mainwindow.hh, mainwindow.cc, mainwindow.ui.

NO CHANGES TO THESE FILES ARE ALLOWED

The changes are implemented in the following two files.

File datastructures.hh

- class Datastructures: the public interface of the class is provided by the course.
    - **Changing the public interface is NOT ALLOWED**: no changes to names, return types or parameters of the given public member functions, etc.
    - A student can still add new methods and data members to the private side
- Constants `NO_CONNECTION` and `NO_DISTANCE` have been added.

File datastructures.cc

- Here you write the implementations for your operations.

On using the graphical user interface

When you compile the program using QtCreator, it generates a graphical user interface (GUI) that

- `get_all_connections()`: It provides a list of all connections.

Additionally, the GUI relies on two functions for querying and retrieving specific details:

- `get_affiliation_coord()`: This function allows you to retrieve affiliation coordinates.

- `get_affiliation_name()`: It assists in obtaining affiliation names.

Please note that the graphical representation is based on the output of the student's code and may not always represent the correct result. It reflects the information provided by the student's code.

Running the program

To run the program, use the commands listed in the table below. Commands that require a member function are linked to the corresponding function in the Datastructures class, which you need to implement (certain commands have already been fully implemented by the course staff).

If you provide a file as a command line parameter to the program, it will execute commands from that file and then terminate. You can also initiate a program compiled with QtCreator from the command prompt using the `--console` command line argument for a purely text-based interaction.

The public interface of the Datastructures

The operations below are listed in the order in which it is recommended to implement them:

Compulsory functions

but not all, are required for the prg2 to function	the same as in assignment 1

`get_connected_affiliations AffiliationID`
 	
`std::vector<Connection> get_connected_affiliations(AffiliationID id)`
	This function returns the connections from the given source affiliation to target affiliations. In case of no connections, an empty vector is returned. If an affiliation with the given ID does not exist, an empty vector is returned. The connections can be returned in arbitrary order but for a single connection the following must hold true. The first affiliation in connection (aff1) must be the given ID, and the second affiliation (aff2) must be an affiliation connected to the given one.

`get_all_connections`
 	
`std::vector<Connection> get_all_connections()`
	Returns all connections as a vector. The connection must contain the smaller affiliation_id first (in Connection aff1 < aff2).
	
	There must be no duplicate connections, and the connections are returned in an arbitrary order.
	
	If there are no connections, an empty vector is returned.

have been implemented.)	
get_any_path AffiliationID1 AffiliationID2 std::vector<Connection> get_any_path(AffiliationID source, AffiliationID target)	Returns an arbitrary path between two given affiliations. The returned vector contains connections from source to target. For a single Connection, the order of IDs must be so that the path can be followed along the connections from aff1 to aff2 at all times. If source, target or path does not exist, an empty vector is returned.

Optional functions

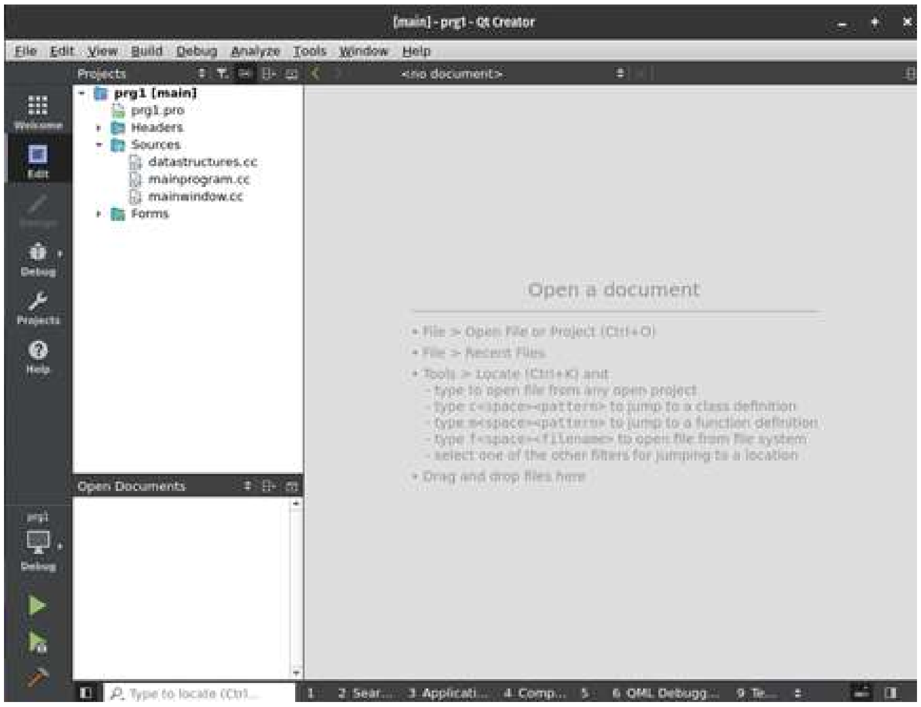
Implementing these commands is not compulsory for passing the project, but implementing them will affect the grade.

Command **Public member function** **Optional part** (Optional parameters are in []-brackets and alternatives separated by |)	**Explanation of public member function**
get_path_with_least_affiliations AffiliationID1 AffiliationID2 std::vector<Connection> get_path_with_least_affiliations(AffiliationID source, AffiliationID target)	**Optional.** Returns a path between the given affiliations with as few affiliations as possible. If several paths with the same number of affiliations exist, any of those can be returned. The returned vector contains connections from source to target. If source, target or path does not exist, an empty vector is returned.
get_path_of_least_friction AffiliationID1 AffiliationID2 std::vector<Connection> get_path_of_least_friction (AffiliationID source, AffiliationID target)	**Optional.** Returns a path between the given affiliations so that the friction is as small as possible. If several paths with equal friction exist, any of those can be returned. The returned vector contains connections from source to target. If source, target or path does not exist, an empty vector is returned.

`get_shortest_path AffiliationID1 AffiliationID2`
 	
`PathWithDist get_shortest_path(AffiliationID source, AffiliationID target)`
	Returns the shortest path between affiliations as a PathWithDist vector. If there are multiple equally valid options, any of them can be returned. The path includes affiliations starting from the initial point and ending at the destination point.
	
	PathWithDist has the format: `std::vector<std::pair<Connection, Distance>>`, where, in addition to the connection, the distance between the affiliations connected by that connection is specified.
	
	If the initial point, destination point, or the connecting path between them does not exist, an empty vector is returned.

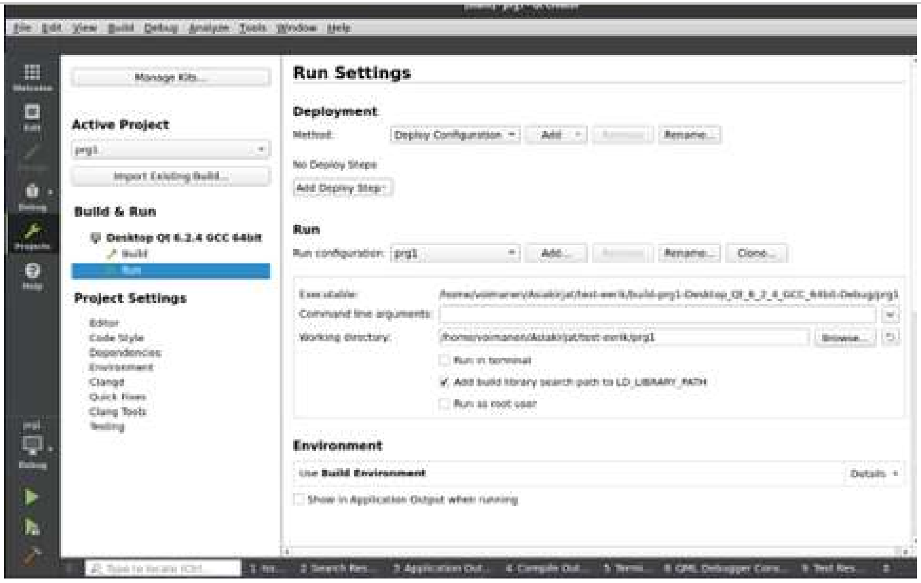
Changing the working directory of a program from QtCreator

Before running tests, change the working directory. This ensures that each test that will read other files will find those files. The directory can be changed as follows:



Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type c<space><pattern> to jump to a class definition
  - type m<space><pattern> to jump to a function definition
  - type f<space><filename> to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here



Method: Deploy Configuration
Run configuration: prg1
Executable: /home/voimanen/Asiakirjat/test-eerik/build-prg1-Desktop_Qt_6_2_4_GCC_64bit-Debug/prg1
Command line arguments:
Working directory: /home/voimanen/Asiakirjat/test-eerik/prg1
☐ Run in terminal
✓ Add build library search path to LD_LIBRARY_PATH
☐ Run as root user
Use **Build Environment**
☐ Show in Application Output when running

Next, find the setting for “Working directory” from the list. Set the working directory to the same directory where the current ‘.pro’ file is. For example, for prg2 the directory should end with `repository_name/prg2` where the `repository_name` is the directory of the Datastructures and algorithms repository.

Data files

The most convenient method for testing the program is by creating 'data files' that add a collection of affiliations and publications. These files can be imported using the 'read' command, allowing you to evaluate other commands without the need to manually input data each time. Below, you will find sample data files located in the 'example-data' folder.

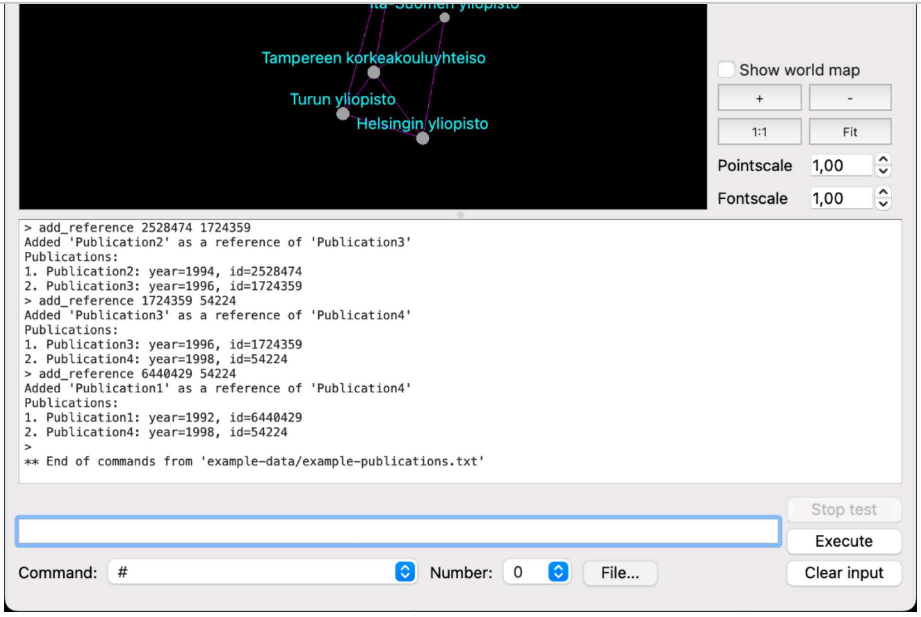
example-affiliations.txt

```
add_affiliation ISY "Ita-Suomen
yliopisto" (945,767)
add_affiliation TUNI "Tampereen
korkeakouluyhteiso" (542,455)
add_affiliation HY "Helsingin
yliopisto" (820,80)
add_affiliation TY "Turun yliopisto"
(366,219)
add_affiliation LY "Lapin yliopisto"
(740,1569)
```

```
"Publication2" 1994
add_publication 1724359
"Publication3" 1996
add_publication 54224 "Publication4"
1998
add_affiliation_to_publication TUNI
6440429
add_affiliation_to_publication HY
6440429
add_affiliation_to_publication ISY
6440429
add_affiliation_to_publication TY
2528474
add_affiliation_to_publication LY
2528474
add_affiliation_to_publication TUNI
2528474
add_affiliation_to_publication HY
1724359
add_affiliation_to_publication TY
1724359
add_reference 2528474 1724359
add_reference 1724359 54224
add_reference 6440429 54224
```

Screenshot of the GUI

Below is a screenshot of the GUI after reading example-affiliations.txt and example-publications.txt.



```
> add_reference 2528474 1724359
Added 'Publication2' as a reference of 'Publication3'
Publications:
1. Publication2: year=1994, id=2528474
2. Publication3: year=1996, id=1724359
> add_reference 1724359 54224
Added 'Publication3' as a reference of 'Publication4'
Publications:
1. Publication3: year=1996, id=1724359
2. Publication4: year=1998, id=54224
> add_reference 6440429 54224
Added 'Publication1' as a reference of 'Publication4'
Publications:
1. Publication1: year=1992, id=6440429
2. Publication4: year=1998, id=54224
>
** End of commands from 'example-data/example-publications.txt'
```

Example run

To get output from the program, after setting up the working directory correctly and starting the program, you can enter commands such as:

```
testread "integration-compulsory/test-00-compulsory-in.txt" "integration-compulsory/test-00-compulsory-out.txt"
```