Linköping University
Department of Science and Technology/ITN
Aida Nordman

TND004: Data Structures
Lab 4
(complementary instructions)

Goals

To implement several well-known graph algorithms.

- Unweighted single-source shortest path algorithm (`UWSSSP`) based on breadth-first search.
- Dijktra's algorithm.
- Prim's algorithm.
- Kruskal's algorithm.

Correction

This lab consists of two parts, **A** and **B**. The pdf files[1] describing the exercises contain some minor imprecisions, which are clarified here.

- It is lab 4, not lab 5 as stated in the pdfs.
- **Part A** requires the concepts presented in lectures 12 and 13.
- **Part B** requires the concepts presented in lecture 14.
- No Code::Blocks project files (`.cbp`) are provided. Instead, a CMake file is given, as in the previous labs.
- As usually, all files required for the exercises in this lab can be downloaded from the course website (and not from `S:\TN\D\004\Lab` as indicated in the files `Lab4a.pdf` and `Lab4b.pdf`).

Preparation

You must perform the tasks listed below before the start of the lab session *Lab4 HA*.

- Download the files for this exercise from the course website. Similar to previous labs, you can then use CMake to create two projects for this lab, one for the exercise in part A and another for the exercise in part B.
- For each project, it's possible to compile, link, and execute the program. There is a `main.cpp` file, for each of the projects.
- Implement the graph algorithms requested in **Part A**.
- Read **Part B** and understand the code given for this part.

[1] Unfortunately, these pdf files were generated from old latex files used in another course and the source latex files are no longer available.

If you have any specific question about the exercises, then send us an e-mail. Be short and concrete, otherwise you won't get a quick answer. You can write your e-mail in Swedish. Add the course code to the e-mail's subject, i.e. "`TND004: ...`".

Part A

In this first part, you are requested to implement the `UWSSSP` and the Dijktra's algorithm (in the lectures, we also named the Dijktra's algorithm as `PWSSSP`[2]).

- Review lecture 12 and lecture 13.
- Read sections 9.1 and 9.3.1, 9.3.2, and 9.3.5 of the course book.
- Read the file `Lab4a.pdf`.
- Study carefully the code given for this exercise[3]: `list.h`, `list.cpp`, `queue.h`, `digraph.h`, and `digraph.cpp`. Except where explicitly indicated (with a commented line saying "`TODO`") the given code cannot be modified.
- **Implement** functions `Digraph::printPath`, `Digraph::uwsssp`, and `Digraph::pwsssp`.

The files `digraph1_test_run.txt` and `digraph2_test_run.txt` contain a test run of the program provided in `main.cpp`, for each of the graphs provided in the files `digraph1.txt` and `digraph2.txt`, respectively.

In this lab, graphs are represented with adjacency lists, as discussed in lecture 12 and a class `List` is provided. Note that the loop below (in pseudo-code), occurring in many of the graph algorithms you have seen during the lectures of the course,

```
for all (v,u) ∈ E do
{ ...; }
```

can be implemented as (`array` is a table with all vertices of the graph)

```
Node *p = array[v].getFirst();
//u ≡ p->vertex

while (p != nullptr) {
    ...;
    p = array[v].getNext();
}
```

Part B

In this second part, you are requested to implement the Prim's and Kruskal's algorithms.

- Review lecture 14.
- Read sections [8.1-8.5] and 9.5 of the course book.
- Read the file `Lab4b.pdf`.

[2] `PWSSSP` stands for Positive Weighted Graph Single-Source Shortest Path.
[3] The files for part A exercise belong to the project in your IDE named `Lab4a`.

- Study carefully the code given this exercise[4]: `dsets.h`, `dsets.cpp`, `edge.h`, `edge.cpp`, `heap.h`, `graph.h`, and `graph.cpp`. Except where explicitly indicated (with a commented line saying "`TODO`") the given code cannot be modified. Files `list.h`, `list.cpp` are provided again and they have the same code as in part A. In the file dsets.cpp, you should implement union by size and find with path compression.

Presenting lab and deadlines

The exercises in this lab are compulsory and you should demonstrate your solutions during the lab session *Lab4 RE*. Read the instructions given in the labs webpage and consult the course schedule.

Necessary requirements for approving your lab are given below.

- Use of global variables is not allowed, but global constants are accepted.
- Readable and well-indented code. Note that complicated functions and over-repeated code make programs quite unreadable and prone to bugs.
- There are no memory leaks neither other memory related bugs. On the labs webpage you can find a list of tools that can help to check for memory related problems in the code.
- The code generates no compilation warnings.

If your solution for lab 4 has not been approved in the scheduled lab session *Lab4 RE* then it is considered a late lab. All groups have the possibility to present one late lab on the extra RE lab session scheduled in the end of the course.

Note that **we can only guarantee that each group can present one late lab**. Priority is given to presentation of lab 4, then lab 3, and finally lab 2.

Lycka till!

[4] The files for part B exercise belong to the project in your IDE named `Lab4b`.