Linköping Institute of Technology
Department of Science and Technology/ITN
Aida Nordman

Data Structures
TND004
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 Fö 11 and 12.
- **Part B** requires the concepts presented in Fö 13.
- No Code::Blocks project files (**`.cbp`**) are provided, since these files are not portable (and not everybody is using Code::Blocks, anyway).
- 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

The **HA** lab session on week 20 focus on lab 4 exercises, the last set of programming exercises in the course. As said above the lab consists of two parts, **part A** and **part B**.

Part A

In this first part, you are requested to implement the **`UWSSSP`** and the Dijktra's algorithm (in the lectures, we also named this algorithm as **`PWSSSP`**[2]). You can find below a list of tasks that you must do before you start doing the exercises in **part A**.

- Review Fö 11 and 12.
- Read sections 9.1 and 9.3.1, 9.3.2, and 9.3.5 of the course book

[1] Unfortunately, these pdf files were generated from old latex files used in another course and the source files are no longer available.
[2] **`PWSSSP`** stands for positive weighted graph single-source shortest path.

- Read the file **Lab4a.pdf**.
- Downloaded the code for **part A** of this lab the from the course website.
- Study carefully the code given in the files **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** before the **HA** lab session on week 20. The lab assistant can give feedback on your solution in the beginning of the lab, before you proceed with the exercises in **part B**.

For **part A**, the files **digraph1_test_run.txt** and **digraph2_test_run.txt** contain a test run of the test 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 11 (see slide 12) and a classes **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. You can find below a list of tasks that you must do before you start doing the exercises in **part B**.

- Review Fö 13.
- Read sections [8.1-8.5] and 9.5 of the course book.
- Read the file **Lab4b.pdf**.
- Downloaded the code for **part B** of this lab the from the course website.
- Study carefully the code given in the files **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.

Presenting solutions

You should demonstrate your solution orally during your **RE** lab session on **week 21**

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. Recall that you can use one of the tools suggest in lab 1.
- The code does not generate compiler warnings[3].

Note that in the **end of week 21** there is an **extra RE** lab session for presenting late labs. In this extra lab session, **we can only guarantee that each group can present one late lab**.

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:** ...".

Lycka till!

[3] **g++** is the reference compiler.