Linköping university
Department of Science and Technology/ITN
Aida Nordman

TND004: Data Structures
Lab 4

Goals

To implement several well-known graph algorithms.

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

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. Like previous labs, you can then use CMake to create a solution with 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 with the `main` function, for each of the projects.
- Implement the graph algorithms requested in **Part A**.
- Read **Part B** and understand the code given for this part.

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:` …".

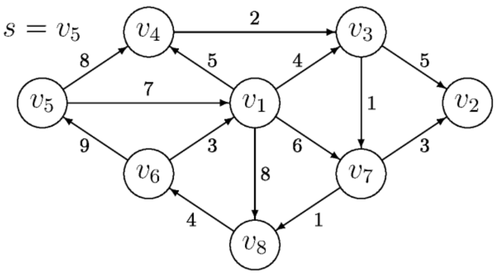
Part A: shortest path algorithms

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`). To this end, follow the steps below.

- 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.
- Study carefully the code given for this exercise: `edge.h`, `digraph.h`, and `digraph.cpp`.
- **Implement** functions `Digraph::uwsssp`, `Digraph::pwsssp`, and `Digraph::printPath`.

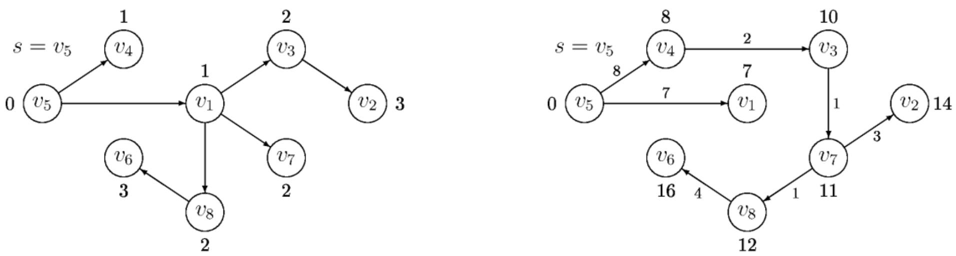
- *Unweighted **s**ingle-**s**ource **s**hortest **p**aths* (`UWSSSP`): given a (unweighted) directed graph $G = (V, E)$ and a start vertex $s \in V$, find the shortest unweighted path from s to every other vertex in V. The container [`std::queue`]() is useful for the implementation of this function.
- ***P**ositive **w**eighted **s**ingle-**s**ource **s**hortest **p**aths* (`PWSSSP`[1]): given a weighted directed graph $G = (V, E)$ and a start vertex $s \in V$, find the shortest weighted path from s to every other vertex in V.

An example of the input graph for the exercises in this part is shown below.



Figur 1: A (weighted) directed graph G and a start vertex s.

The corresponding output, a shortest path-tree, is exemplified in figure 2.



dist	1	3	2	1	0	3	2	2
v	1	2	3	4	5	6	7	8
path	5	3	1	5	0	8	1	1

dist	7	14	10	8	0	16	11	12
v	1	2	3	4	5	6	7	8
path	5	7	4	5	0	8	3	7

Figur 2: Unweighted (*left*) and weighted (*right*) shortest path-tree for s.

In this lab, graphs are represented with adjacency lists. This graph representation was introduced in [lecture 12](). Note that the loop below (in pseudo-code) to iterate through the adjacency list of a vertex v (used in several of the graph algorithms you have seen during the lectures of the course)

```
for all (v,u) ∈ E do {
    ...;
}
```

can be implemented as (`table` is a [`std::vector`]() with all vertices of the graph and [`std::list`]() is used to store the adjacency lists)

```
for (auto& e : table[v]) {
     int u = e.to;     // e = (v,u)
      ...;
 }
```

[1] This is also known as the Dijkstra's algorithm.

The files `digraph1_test_run.txt` and `digraph2_test_run.txt` contain an example of the program execution, for the graphs provided in the files `digrph1.txt` and `digraph2.txt`, respectively.

(If you have a Mac then you may need to manually change the input file path in function `readGraph`, file `main.cpp`).

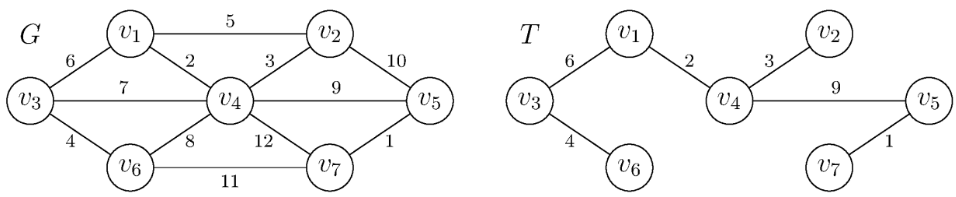
Part B: minimum spanning tree algorithms

In this second part, you are requested to implement the Prim's and Kruskal's algorithms. To this end, follow the steps below.

- Review lecture 14.
- Read sections [8.1-8.5] and 9.5 of the course book.
- Study carefully the code given for this exercise: `dsets.h`, `dsets.cpp`, `edge.h`, `graph.h`, and `graph.cpp`.
- Implement functions `Graph::mstPrim` and `Graph::mstKruskal` and test that they work as expected. Both functions build a minimum spanning tree[2] (MST) from a given connected weighted undirected graph $G = (V, E)$. They should display the edges of a minimum spanning tree and the total weight of the MST built. The implementation of `Graph::mstKruskal` requires a min-heap of vertices and STL functions for heap operations can be used (see also `std::make_heap`).
- Re-implement function `Dsets::join` so that **union by size** is performed. Re-implement also function `Dsets::find` so that find with path compression is performed. The code distributed with this lab uses simple union and simple find for these functions.

Like as for Part A, graphs are represented with adjacency lists.

Input and ouput for the Prim's and Kruskal's algorithms are exemplified in the figure below. In particular, the ouput (i.e. a MST) is essentially just a list of $|V| - 1$ graph edges.



Figur 1: A connected weighted undirected graph G and a mimimum spanning tree T for this graph (in this case, the tree is unique).

The files `graph1_test_run.txt` and `graph2_test_run.txt` contain an example of the program execution, for the graphs provided in the files `graph1.txt` and `graph2.txt`, respectively.

(If you have a Mac then you may need to manually change the input file path in function `readGraph`, file `main.cpp`).

[2] i.e. a spanning tree for graph G of minimum total weight.

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.

As usual, we expect that good programming practices are followed. Your code must be readable and well-indented.

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**. In the extra RE lab session, priority is given to presentation of lab 4, then lab 3, and finally lab 2.