IN2010 Obligatorisk oppgave 2

Kristin Eng, Anine Lome & Didrik Sten Ingebrigtsen

13. oktober 2021

Instruction for running the code

Run the code with this command:

```
python oblig2.py
```

File system:

```
obligl2.py
actor.py
movie.py
travers.py
components.py
```

Problem 1

We read the data with the function **read_data**. We store the data in respectively **all_movies** and **all_actors**.

We built the graph as an abstract data structure. The idea is that we don't fill in the information about which actors an actor is connected with through a movie, until it is needed.

We implemented one class for movie objects, **Movie**, and one class for actor, **Actor**, objects. The actor class has a **neighbors** property that defines an adjacency list containing neighbor actors as keys and common movies as values. The adjacency list is made by a call to the method **_build_adjacency**, which again is only called for the actors we use in our examples, and only if the actor is not already in the adjacency list.

Relevant files: oblig.py, actor.py, movie.py

Problem 2

To implement Six Degrees of IMBD, we find the shortest path using **breadth first search** (BFS). We ended up using a double-ended BFS, after timing both normal BFS and double-ended, which resulted in the double-ended being almost

10 times as fast. The double-ended BFS works by making one finding all the neighbors of the actor we want to go from, then around the actor we want to go to, and then finding the first actors neighbors neighbors, and so on until we find an actor that can link these ever growing sets of connected actors.

Relevant files: oblig.py, actor.py, movie.py, traverse.py

Problem 3

Here, we find the shortest path (/"chillest path") with weights from one actor to another, using **Dijkstra's algorithm**. We use a heap as data structure, and go from neighbor to neighbor until the heap empty.

Relevant files: oblig.py, actor.py, movie.py, traverse.py

Problem 4

To find all components and their size, we use two functions: **find_components** and **find_all_components**. The first function finds one component and measure its size, using a queue that keeps track of the nodes, running until the queue is empty. The second function calls the first for all nodes in the graph, however not for nodes that are already visited in previous searches (on another nodes path). **find_all_components** returns a dictionary where keys are the component sizes and the values are the number of components with the key-size.

Relevant files: oblig.py, actor.py, movie.py, components.py

Timing

Problem	**Time (s)**
Problem 1	1.56902
Problem 2	0.02130
Problem 3	18.24883
Problem 4	21.79660
Sum	41.64

We know that we saved some time in problem 2, by using double-ended BFS. However, we acknowledge that problem 3 and 4 probably can be done in a faster way. Could we have stopped searching earlier in problem 3?

We mention that one of the reasons that the time distribution is skewed, probably is because the graph is not built completely before problem 3 and 4.