Homework #1

(5 points) Consider the special case of finding the shortest path in a graph with positive integer distances in the range $[1, \ldots, C]$. Implement Dijkstra's algorithm with radix heap (with `decrease-key` operation) and compare it with an implementation using a classic binary heap.

You can find an explanation of radix heap here:

`https://kubokovac.eu/ds/mat/radix.pdf` (sk)

`https://kubokovac.eu/ds/mat/radix-heap-en.pdf` (en)

Find out, which algorithm is better, depending on C and depending on the graph density (number of edges per node $\delta = M/N$). (I find the most interesting the ranges of say $C \in [1 \ldots 1\,000\,000]$ and δ around 10, say $\delta \in [3 \ldots 100]$.)

If you want, you can start with this code for comparison:

`https://kubokovac.eu/ds/src/dijkstra.cpp`

where Dijkstra's algorithm with a binary heap is already implemented (traditional implementation with `decrease_key` and lazy implementation adding copies to the heap) and also with an ad hoc data structure where we maintain for each distance a bucket of vertices at that distance from the origin. Submit the source code and the measured data plotted in a graph. Tips and notes:

- Call the submitted files `hw1.cpp` and `hw1.pdf` (text/graphs). After you're done with development, go over your program again, clean it up and simplify it. Use an automatic code formatter.
- Don't leave the homework for the last moment – you'll regret it; if you need help, message me.
- Test your program thoroughly – correctness $\gg$ speed.
- Make enough measurements for each choice of parameters (not just one!) – preferably, calculate average and standard deviation or display the data as a box plot. Don't forget to label the axes and include units.
- Write the code in a low-level programming language – ideally C/C++/Rust, in the worst case Java/C#/D.
- Don't forget to turn on optimization (e.g. for `gcc`, you want the `-O3` flag) – this task is about efficient code, we care about constants.
- I recommend turning on all warnings (e.g. `gcc -Wall -Werror -pedantic` and while testing, also `-fsanitize=address`), so that possible bugs are already caught by the compiler.