Assignment 3: Floyd-Warshall algorithm

Author: Nuno Nelas <fc51691@alunos.fc.ul.pt>

A description of the necessary modifications for GPU-execution

To make this algorithm work for GPU-execution we have to first allocate memory for the copy of the graph that will run on the device. To do that, CUDA runtime API has a method called `cudaMalloc` that works similar to C `malloc`.

After that, we have to actually copy the graph created on the main coroutine to the array created in the previous step. In practice, we have to use `cudaMemcpy` that depending on `cudaMemcpyKind`(eg: `cudaMemcpyHostToDevice` and `cudaMemcpyDeviceToHost`), it copies from the device to host or vice-versa.

Now we're ready to launch the algorithm itself. `floyd_gpu_compute` will be the kernel that contains the Floyd-Warshall algorithm. The information between the triple chevrons is the execution configuration, which dictates how many device threads execute the kernel in parallel.

Inside `floyd_gpu_compute`, I've kept the same algorithm as for the CPU version, however I've decided to parallelize the `i` for-loop. If we want each thread to process an element of the resultant array, then we need a means of distinguishing and identifying each thread. CUDA defines the variables `blockDim`, `blockIdx`, and `threadIdx`. The expression `i = blockIdx.x * blockDim.x + threadIdx.x;` generates a global index that is used to access elements of the arrays.

After running the algorithm, we have to copy the resulting array back to the host using `cudaMemcpy` and finally free the data, by just passing the pointer to `cudaFree`.

The rationale of the choice of parameters for GPU-execution (blocks, threads) and memory usage or transfers.

CUDA GPUs run kernels using blocks of threads that are a multiple of 32. I've started with a number of threads per block (TPB) of 128, increasing it until 1024.

For defining the number of blocks I've simply divided the number of `GRAPH_SIZE` by the `THREADS_PER_BLOCK` size.

Table below shows the execution time in seconds for the algorithm running in CPU and GPU on a Google Colab environment:

	CPU	GPU
Exec time (sec) for 128 TPB	27.347742	2.236863
Exec time (sec) for 256 TPB	27.146551	1.356490
Exec time (sec) for 512 TPB	27.225489	1.240561
Exec time (sec) for 1024 TPB	26.999142	4.510954

As we can see, we can achieve better results with a TPB=256 or TPB=512, while using Google Colab environment.