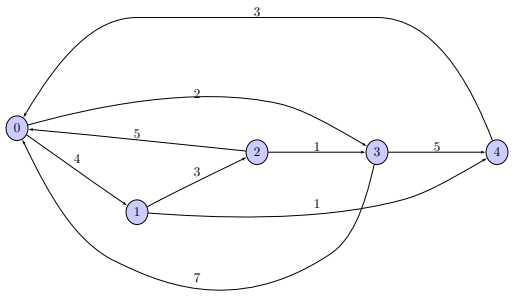
Laboratory Work No. 2
Multithreading
OpenMP

The Floyd–Warshall algorithm, whose pseudo-code is given below, finds the shortest path between all vertices of a weighted directed graph like the one presented below.



A C++ implementation of the algorithm is available on *madoc* in the file `floyd-sequential.cpp`.

1. Using OpenMP, parallelize both the computation of the `shortestDistance` and `nextInPath` matrices, and the recovery of the shortest path between two vertices;
2. Compare the sequential and the parallel versions on large random graphs and plot the results. Modify your parallel program to take into account your findings and only use OpenMP when the overhead does not lead to poorer performances.

```
# V: number of vertices in the graph
# Weights: matrix VxV of weights
# shortestDistance: matrix VxV of distances (sum of weights)
# nextInPath: matrix VxV of next vertex in shortest path
def Floyd_Warshall(Weights,shortestDistance, nextInPath):
  # Initializing shortestDistance to Weights
  shortestDistance = Weights
  for k in range(0,V):
    for i in range(0,V):
      for j in range(0,V):
        if shortestDistance[i][k] + shortestDistance[k][j] < shortestDistance[i][j]:
           shortestDistance[i][j] = shortestDistance[i][k] + shortestDistance[k][j]
           nextInPath[i][j] = k

def recover_path(shortestDistance, nextInPath, i, j):
  if shortestDistance[i][j] == infinity: # No path between i and j
     return empty path
  intermediate = nextInPath[i][j];
  if intermediate == -1: # Edge i--j is shortest path
     add i and j to path
     return path
  else:
     path1 = recover_path(shortestDistance, nextInPath, i, intermediate)
     path2 = recover_path(shortestDistance, nextInPath, intermediate, j)
     if either path is empty:
        return empty path
     return path1 + path2
```

Table 1: Pseudo code for the Floyd–Warshall algorithm