TDA384 - Lab 3 - Documentation

--

Robin Lilius-Lundmark (CID: lurobin)
Andreas Carlsson (CID: andrc)

We came up with our solution by first introducing parallelism to the sequential solution, not worrying about private data structures but only to make it work. We then gradually removed global data structures until we felt that our solution was optimal.

Since we were supposed to still implement a depth-first-search, we more or less copied the sequential solution and started there. The sequentialDepthFirstStep()-method is just that.

We decided that only the data structure "visited" would need to be shared among all the forks, since otherwise a lot of unnecessary work would be needed to avoid checking nodes several times. We create new data structures for "predecessors" and "frontier" in new forks.

We also made use of an AtomicBoolean, distributed among all ForkJoinSolver objects to keep track of when a solution is found, making it possible to cancel all other processes immediately. After all: once a goal is found, the search is completed.

Every time we fork, we create as many new ForkJoinSolver as there are unvisited nodes left in the current ForkJoinSolver. To keep the current thread from just waiting, we also keep one of these ForkJoinSolver objects working in the original thread. To visualize this, we decided to re-use the player ID that the previous object made use of, in the new local one. This is the reason for the two similar constructors.

Lastly, when the goal is found, a list of the nodes, the path to the goal, is returned (as in the sequential solution). The problem is that we only get the path to the goal from the starting position of the specific ForkJoinSolver that found it. Therefore, making use of the original pathFromTo()-method, we have extended this in the addPath()-method so that the partial solutions can be built together until we have a complete one, that leads from the very beginning to the definite goal.