**Optimizing approach for maze solver**

To begin with, the sample code from professor contains dfs, bfs and recursion version of solve functions. I was first trying to figure out a faster way of searching the function, like dynamic programming. But I noticed that we are looking for a single way out rather than the shortest way out. It would be slower if we try to figure out the shortest way out rather than return when found the exit. Finally, I decided to use dfs as the algorithm rather than other algorithms.

Sample code is a sequential search which is in great single thread. My idea is using multithreads to do the search task. But it would be slower if we just initiate multiple threads and run dfs since it also needs resource to initiate and destroy threads. In addition, since there are multiple choices for each step, in another word, there may have possibility for every step to be invoked concurrently. A great way for the solution is to use ExecutorService to help us manage threads rather than we do it by ourselves. The next question is about the thread pool we initiate which is also related to the solution time.

From the textbook we know that the best size for the thread pool should follow the formula:

Nthreads = Ncpu \* Ucpu \* (1 + )

Ncpu: Number of CPUs

Ucpu: target CPU utilization, 0 <= Ucpu <= 1

： ratio of wait time to compute time

I imagine there is no wait time and the utilization of the CPU is 100 percent for which the size of the thread pool should equals to the number of CPUs.

After having the thread pool, there are mainly two solutions to figure out the exit of the maze, one is initiate the tasks at the beginning, the other is initiate tasks during every single choice. I choose the first one since the second one could be extremely hard to solve. The number of tasks in my solution is based on the start point. Finally, let the executor invoke all tasks using worker threads in the thread pool. Check more details from the comment in the solver class.