**Approach**

Explaining for the DepthFirstSolution(DFS) approach as it takes the least time. The maze traversal is divided in tasks and those tasks are assignment in thread pool as follows:

1. Get number for processors available

numProcessors = Runtime.getRuntime().availableProcessors();

1. Create an Executor thread pool

pool = Executors.newFixedThreadPool(processors);

1. Get the start of the maze

Choice startPts = firstChoice(puzzle.getStart());

1. for all the choices available from the start position get a next available choice

Choice currentChoice = follow(startPts.at, startPts.choices.peek());

1. add a new DFS task for the currentChoice
2. Once all the tasks are created, invoke the tasks

possibleSolution = pool.invokeAll(tasks);

1. for each futureTask in futureList

if (!futureTask==null) add it to outputList

1. outputList will contain the choices which can lead to final position return outputList

**Pre/Post Conditions, Invariants & Exceptions**

1. **StudentMultiPuzzleSolver.java**

**Invariants:**

* Puzzle will contain at most one solution.
* Puzzle will not contain cycles and/or loops.
* Puzzle will be no larger than 20,000x20,000 cells.
* Puzzle will have four compass directions (North, South, East, West) to advance to an adjacent position.
* The start and end point of the puzzle should not be the same.

**solve()**

**PreConditions:**

* The thread pool size should not exceed the no. of available processors to avoid thread starvation
* At any given point in the execution, the list of first choices yields the current path.
* A stack of choices is maintained. (Should not be empty and should not reach the dead-end on the first attempt)

**PostCondition:**

* An executor service has been created for the threads.
* Tasks are initialized for each thread in the pool.
* if there is no choice available to move to from current position, then it’s a dead-end. Backtrack by popping elements from stack which stores the progress of movement through maze
* return list of Directions till the end point else return "no solution found"

**Exception:**

* If deadlock appears then throw interrupted exception
* Execution Exception occurs when callable is not completed

**Pseudo-Code**

public class StudentMultiPuzzleSolver extends SkippingPuzzleSolver

{

public StudentMultiPuzzleSolver(Puzzle puzzle)

{ super(puzzle);}

public List<Direction> solve() {

// TODO: Implement your code here

//get number of processor

Initialize numProcessor as availableProcessors during runtime

//create an Executer thread pool

Initialize pool as fixed threadpool of size processor

Initialize tasks as LinkedList as <Callable<List<Direction>>>

try {

Initialize Choice startPts to get start of the maze

WHILE (start of maze isNotEmpty) {

add a new DFS task for the currentChoice

}

} catch (Exception e) { }

Initialize List<Direction> possibleSolutions

try {

FOR (loop for size of tasks) {

//invoke the tasks

possibleSolution is equal to pool.invokeAll(tasks);

IF PossibleSolutions is empty Break

}

} catch (InterruptedException e) {

// TODO Auto-generated catch block

} catch (ExecutionException e) {

// TODO Auto-generated catch block

}

Shutdown threadPool

return possibleSolutions

}

private class DFS implements Callable<List<Direction>> {

//implement DFS

return best solutionPath

}