# Final Lab: Unlock Your Mind!

Goals:

* Students will solve a basic problem in Artificial Intelligence
* Students will utilizes graph search techniques
* Students will discover the differences between the performance of several search techniques.
* Students will write a professional document to report their findings.
* Students will implement their solution on their own.
* Students will write a design document on their own.

In this lab, you will evaluate the suitability of three search algorithms to open TheLock. TheLock can be opened by using a sequence of operations (consisting of twisting, pulling, poking and shaking). However, we don't know how many operations are needed to open or for that matter, the sequence of operations.

Your task is to implement all three algorithms, test their suitability in finding the solution to opening TheLock, and report on it. Except for the design phase, which you will do in teams, you will work on this lab by yourself. **This lab and the accompanying Design Document are both due by 4:00 PM on Friday, December 6,** the last regular day of class (before finals week). **This is a hard deadline:** **Late work will not be accepted or graded**.

This lab will be worth 5% of your final grade. One third of the points are for the implementation, one third for your design writeup, and one third for your experimental results and analysis. Unlike other labs, your code may require a very long time to run, and it can only be run on the local machine, not Moodle. Please also keep in mind that if you do not implement the code, you can’t write it up or analyze the data. It is worth doing this well, as it could also be assessed by other faculty members for placement purposes.

## TheLock

You have been given TheLock. You only have access to the class file (compiled Java bytecode that will run on the Java Virtual Machine) and not the source code itself.

The baseline constructor will create a lock that can be opened with an unknown sequence of actions:

TheLock(String s)

The sequence of actions is generated based on the string that you pass to the constructor. The constructor creates what is known as a cryptographic hash of your string. Note that if you create two objects using the same string, this guarantees that the constructor will output a Lock object with the same solution sequence. This is useful for experimentation and reproducibility.

It is likely that the constructor will generate an very long sequence of actions. In fact, you have no way of knowing in advance just how long it is! However, there is a second constructor that will allow us to dictate the length of the sequence:

TheLock(String s, int length)

You can perform the following 4 actions on TheLock to attempt to unlock it.

twistIt()

pokeIt()

pullIt()

shakeIt()

The method isUnlocked() will tell you if the sequence of actions you performed has unlocked the lock. (Note that if you do any additional action, the lock will “relock.”)

Finally, the method resetLock() will allow you to start a brand new sequence in order to unlock the lock.

## Recording Passing time in Java

To test your code, you will need to know how to record the amount of time that it takes.

long start = System.nanoTime();

// do something

long end = System.nanoTime();

long diff = end-start;

System.out.println(TimeUnit.NANOSECONDS.toMillis(diff));

## Phase 0: Design

On this step ONLY you may work with a group to design solutions to this problem. Take as long as you need, but once you start writing code, you are on your own for this final lab.

## Phase 1: Get Started with the Code

Get comfortable with TheLock. You should write a test file to create a lock using your name with a solutions length of 1. Use brute force to figure out the answer. Keep in mind that if you guess the wrong method, you must **reset** the lock to try a new combination.

Once you found the combination to a length one lock, try a length 2 lock. Notice that it could be a completely different sequence, even if you give it the same string with a limit of 1.

Note: You do not need to submit your code for this step. For now you are exploring the problem.

## Phase 2: Behold the Infinite!

We would like to get the computer to solve this problem for us. One problem is that we don't know how long the sequence of actions will be in order to open a lock. To do that, we'll create an infinite tree to represent all of the possible combinations of what we can do. This is a common approach in classical Artificial Intelligence (AI) known as *uninformed search*.

But professor, why are we not building a graph? Those things are awesome!

Young coder, this same technique will work with a graph, but you don't need one in this case, since none of the cases will create a loop. A lack of loops means that we don't need to worry about "marking" nodes.

O, professor! I just read about these permutation algorithms. I can pop that lock in no time!

Young algorithmist, you are completely right. A direct permutation attack would work and be more efficient. However, you will be assessing the performance of different search techniques. So outside of possibly an extra comparison point for your writeup and the warm feeling of learning something new, there is no reason to explore that technique.

Think about the UML for your tree. Each node should represent an action that you can do to your lock. Explore the following phases to determine if there is anything you should add to your UML.

## Phase 3: Checking an Answer

When you visit a node, you should determine if the path from the root to that node represents a solution to the lock. If you found a solution, you will also want to print out information about that solution, i.e., the length of the solution and the sequence of actions (things you need to do to unlock the lock). The solution sequence can then be used to unlock the lock.

## Phase 4: Generating Children

Determine how to generate NEW children after you visit a node.

## Phase 5: Finding a solution, Breadth First Search

Use the Breadth First Search (BFS) algorithm we discussed for graphs to find a solution to the lock. Try something relatively small to start (like a length-2 lock). Then try successively longer locks. What happens? What is the longest sequence that you can solve? Why?

## Phase 6: Depth Limited Search

Next, we will try a variant of the Depth First Search (DFS) algorithm. Depth Limited Search is a Depth First Search algorithm that will stop expanding nodes at some predetermined depth. This can work very well when you know what the maximum depth for a solution will be. Create a Depth Limited Search to solve your problem. Be sure that the depth is a parameter to the problem you are solving.

## Phase 7: Iterative Deepening Search

There are times when we don't know how deep we must travel in order to find a solution. In these cases, we can use the Iterative Deepening Search (IDS) algorithm. We start by using the Depth Limited Search on size 1. If that fails to find a solution, we iteratively call the Depth Limited Search for depth 2, 3, etc. Each time we recall the Depth Limited Search, the algorithm will revisit previously seen nodes. However, it will not infinitely chase a path that will not lead to a solution. (And surprisingly, the time complexity turns out to be the same for the problem as a whole. Remind me to post a paper by Dr. Korf on this if you are interested.)

## Phase 8: Measuring Solutions

If you only measure this once, you may have some anomalies. You should test the following on 5 different locks and report the averages for all of the data. You can quickly create new locks by appending a different number to each string.

for (int i = 0; i < **5**; i++)

{

TheLock l = new TheLock("yourName" + i, length);

//run your tests on the lock

}

### A) Report Trends

For each solution, BFS, DLS, and IDS, report the following for a set of different length locks up to length 14.

* Average running time (CPU or wall time) to find the solution.
* Average number of nodes visited before you find the solution. Keep in mind that the IDA algorithm will visit nodes over and over.
* Average length of the stack or queue used in each algorithm, *when you find the solution.*

If you find that the system runs out of memory, simply mark that it cannot complete that process. (For example, with an asterisk \* and a note below the table or graph explaining what happened.)

Be sure to estimate what these values should be by hand. If you find that the numbers are drastically off, there could be an error in your program that needs to be addressed. (For example, if you say it visits 10,000 nodes to solve a length 2 problem, you have an error in your code!)

### B) Report your running times for the unknown sized lock.

For the two complete solutions, BFS and IDS, report the data on how long it takes to find a solution for the lock where we don't know the length **(You only need to report one value, not an average.)** Keep in mind this may take a while and not all algorithms might work. You should go get some coffee while this is running, maybe read a book, sleep, do homework, or study for finals.

## Phase 9: Write Up Your Results

In your document, you should present:

* The problem
* Your design to record this solution
* Outline the 3 experimental algorithms
* Report on your results
* Draw conclusions based on your experiments

Be sure to make good use of **tables, charts and graphs** to present your data.

You must both submit your document online and provide a written document for me to grade.