Yao Hong Chun
300232587

Introduction

In project 1, path finding is one of the main problems. There are two types of path finding that is required.

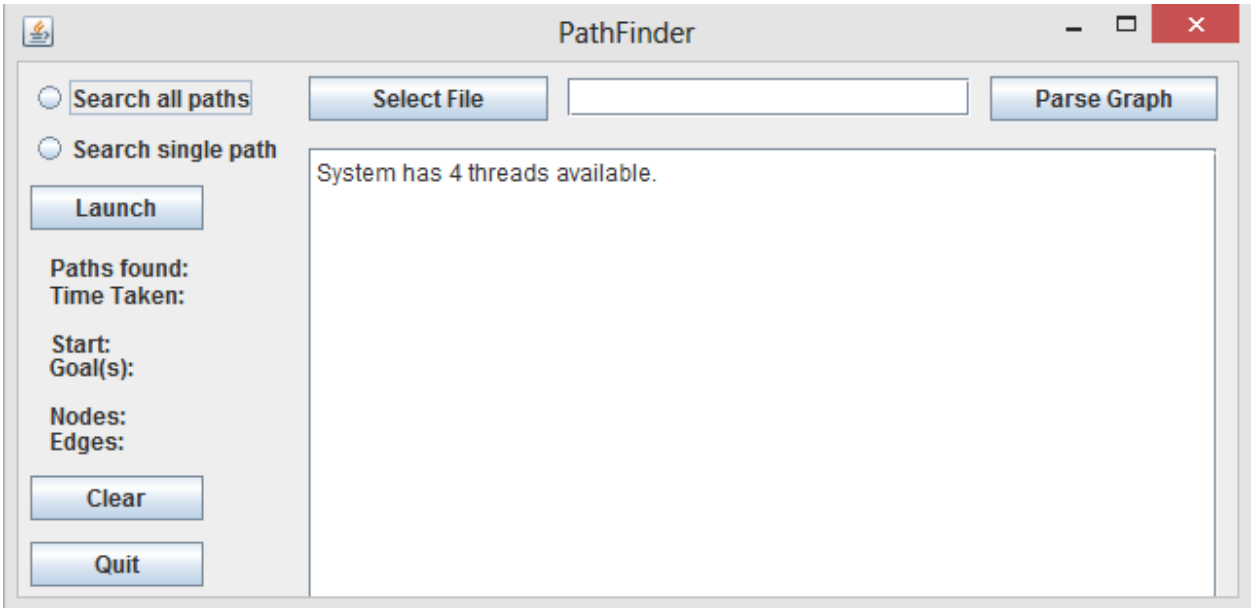
1) Search for all possible paths to goal nodes from a start node.
2) Search for a single path to a goal node from a start node.

Searching for all possible paths will take a very long time if the graph is huge. Searching for a single path to a goal node depends on the algorithm used, although as always, a large graph will take longer than a smaller one. Searching concurrently is another problem and requires proper synchronization techniques for it to be achievable.

Design

The program that is written uses Java's ExecutorService class to manage its threads. It is able to accept tasks and will execute the tasks when it is able to. Doing this means that the program is limited to a set number of threads and will be able to handle as many tasks as possible, although only executing a few of them at a time but is executed efficiently. Tasks can be queued up indefinitely. Additionally, this will maximize search capabilities by fully utilizing the systems available threads. For example if a system has 64 threads, the ExecutorService will use all 64 threads for execution. If there are 1000 tasks submitted, only 64 of the tasks can be executed at a time. This will prevent a single system thread to work on multiple threads/tasks simultaneously.

A simple GUI has been designed and implemented for the path finding program.



Implementation

The program is capable of searching for all possible paths and a single path in a graph. The program works on graphs that are trees, and graphs with infinite cycles. However, it is unable to parse directed graphs.

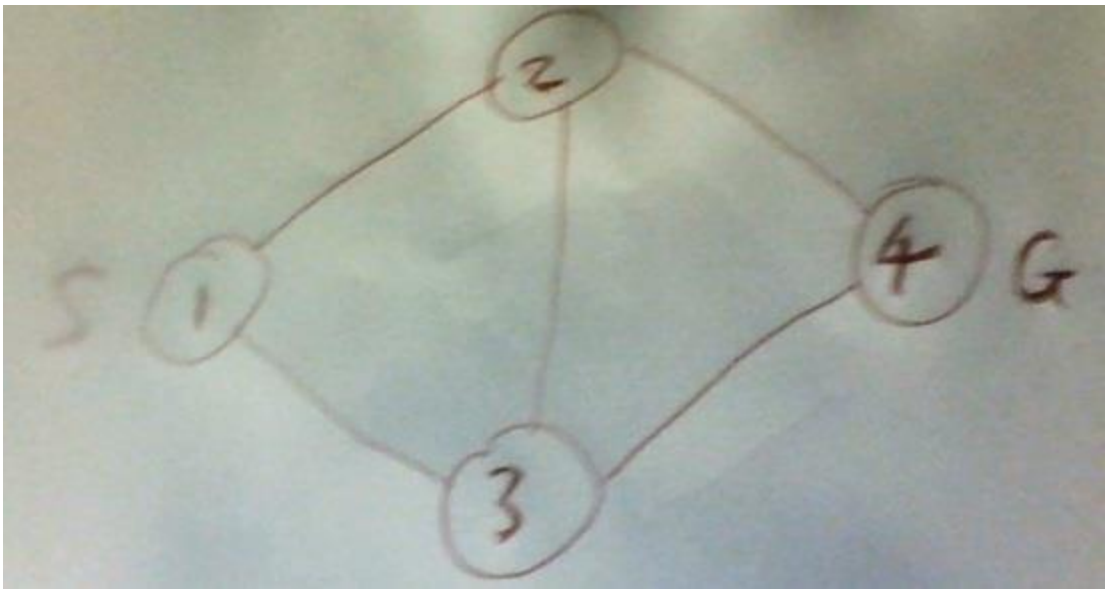
To search for all possible paths, the breadth first search algorithm is used. The algorithm will initialize on the start node, and then create a single task to queue for each of its neighboring nodes which will also do the same thing until the goal is found. This will cause a huge number of tasks to be instantiated if the graph is huge but will not cause out-of-memory problems because the program uses the ExecutorService, creating a large number of threads will no longer be necessary. Only large amounts of tasks will be created which doesn't use much memory therefore the out-of-memory problem is solved.

To search for single paths requires a more complex algorithm. The depth first search algorithm is used to implement the search. Each thread will search for the goal but never going into each other's paths. This is prevented by synchronizing the change of the *visited* Boolean variable in each node so that no more than one thread can change it. When a thread finds a goal, it will shut down the ExecutorService to prevent any more unnecessary searching. The algorithm is achieved with the use of a stack with its first in last out nature. Neighboring nodes are added to the stack, and will be popped to advance to the next node.

Testing

Testing was initially done with a simple graph on a laptop:

Test Machine: 2 cores and 4 threads clocked at 1.8 GHz (My laptop). Graph: SimpleGraph

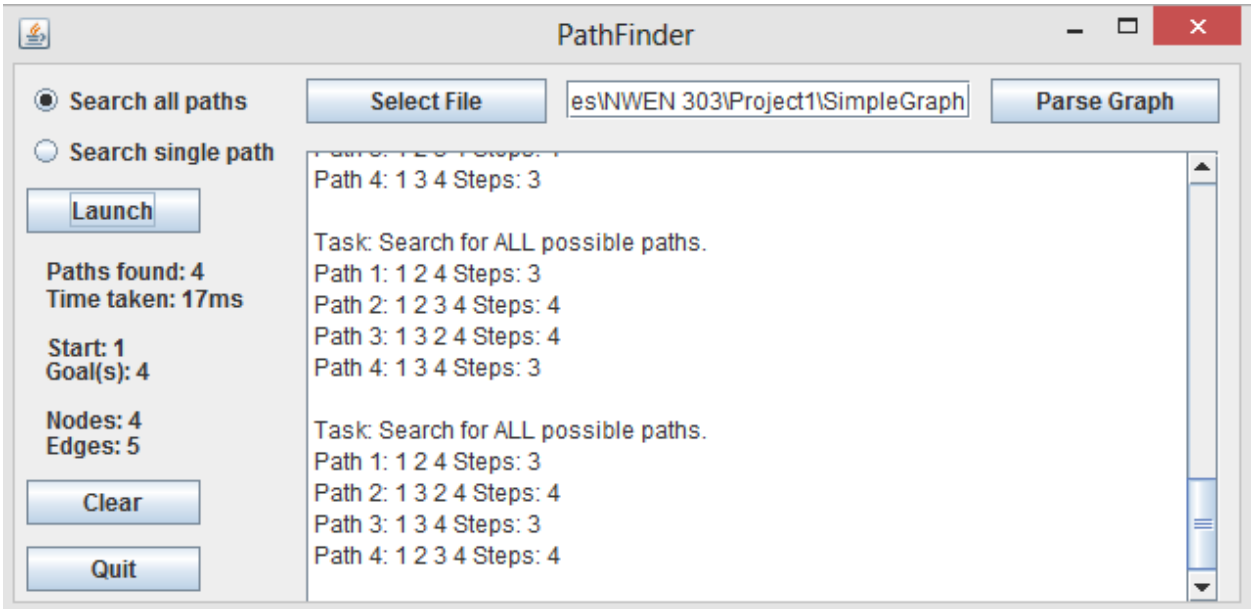


The graph only has four nodes and five edges. Node 1 is set as the start; node 4 is set as the goal.

Yao Hong Chun
300232587

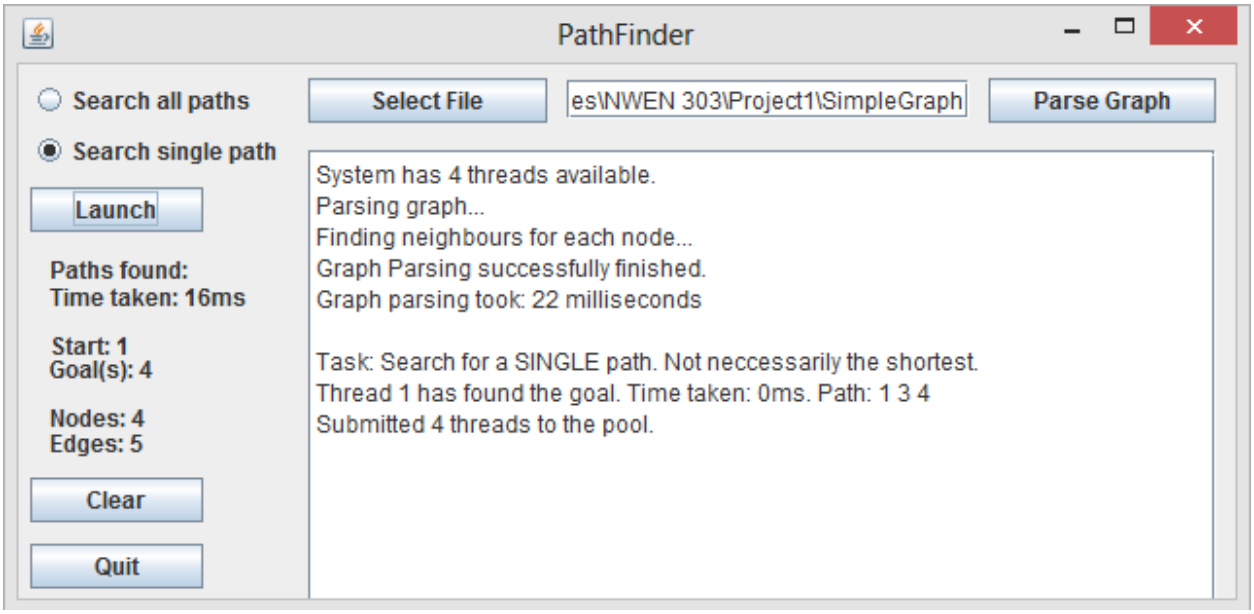
Results:

All paths:



As shown above, there are four paths found. It was easy to verify the result because the graph is so small. Running the search for all paths multiple times gives the same result but in different order.

Single path:



As shown above, thread 1 finds the goal before all the threads was even assigned a task. It only took 16 milliseconds to complete. The path 1->3->4 is indeed correct and is one of the shortest paths.

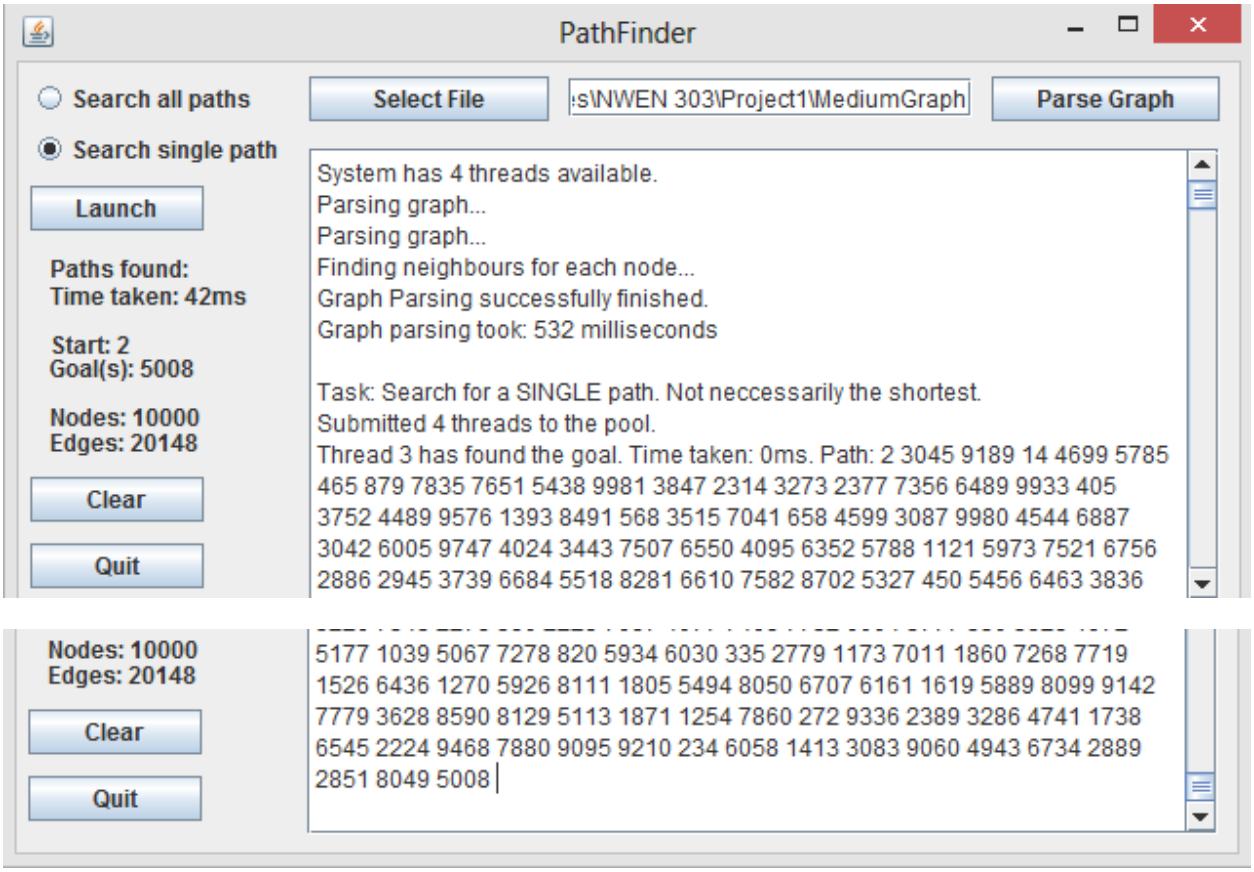
Later on, the program is tested with a larger graph with 10000 nodes and 20148 edges (MediumGraph):

Results:

All paths:

This one was hard to test because there would be thousands of paths. It searched until around 600 paths until it became very slow and had to be terminated manually. The laptop that the program was tested on became noticeably hot and loud with its fans spinning at max speed.

Single path:



As shown above, the algorithm to search for one path is extremely quick even though the graph is large. However, it is difficult to verify its result because the length of the path is huge, over 100 steps. But with prior testing on simpler graphs, i.e. 4 nodes and 5 edges, it is safe to say that the result/path is correct. Another way to partially verify the correctness of the result is to look at both the data and the result. The result shows that it starts at 2 and the next node is 3045, go to the data and look for the Edge with node 2 and 3045. Do this for a few more of the other edges and if they all exist in the data, the correctness of the result would be partially verified.

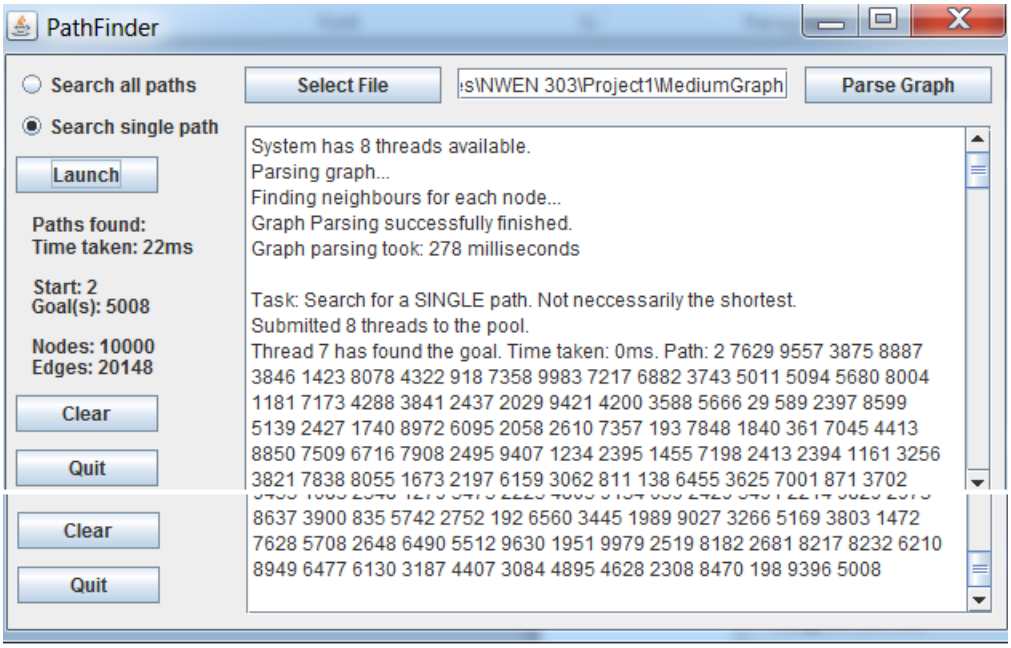
Test Machine: 4 cores and 8 threads clocked at 3.3 GHz (My home PC). Graph: MediumGraph

All paths:

```
Path 863: 2 3045 9817 298 8079 4898 7972 399 6494 6275 7928 8049 5008 Steps: 13
Path 864: 2 3045 9817 298 1976 9213 3678 2442 3202 7662 8380 6438 5008 Steps: 13
Path 865: 2 3045 9817 298 1976 2758 32 1651 5750 2248 7928 8049 5008 Steps: 13
Path 866: 2 5187 2897 8022 7646 8969 9313 1924 9702 7185 3942 9873 5008 Steps: 13
Path 867: 2 5187 2897 8022 7646 8969 9313 3810 1541 9048 4018 2989 5008 Steps: 13
Path 868: 2 5187 2897 8022 743 8411 5032 7282 2096 2392 5369 8049 5008 Steps: 13
Path 869: 2 3045 9817 298 1976 5774 2582 1977 9278 9590 4891 8049 5008 Steps: 13
Exception in thread "AWT-EventQueue-0" java.lang.OutOfMemoryError: GC overhead limit exceeded
        at javax.swing.text.BoxView.updateLayoutArray(Unknown Source)
        at javax.swing.text.BoxView.replace(Unknown Source)
        at javax.swing.text.View.append(Unknown Source)
        at javax.swing.text.FlowView$FlowStrategy.layout(Unknown Source)
        at javax.swing.text.FlowView.layout(Unknown Source)
```

As shown above, the computer managed to find 896 paths before the textbox of the GUI ran out of memory.

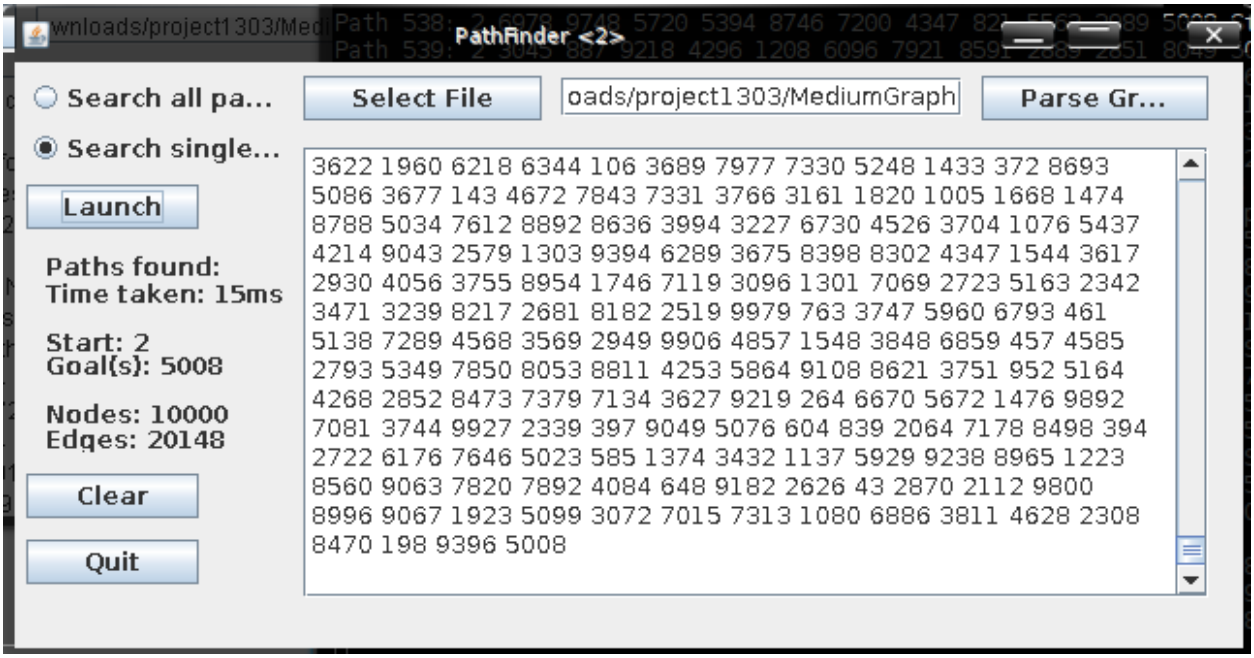
Single path:



It finds a path much quicker at 22 milliseconds down from 42 milliseconds, but the path is different to the previous one. It also parses the graph twice as fast.

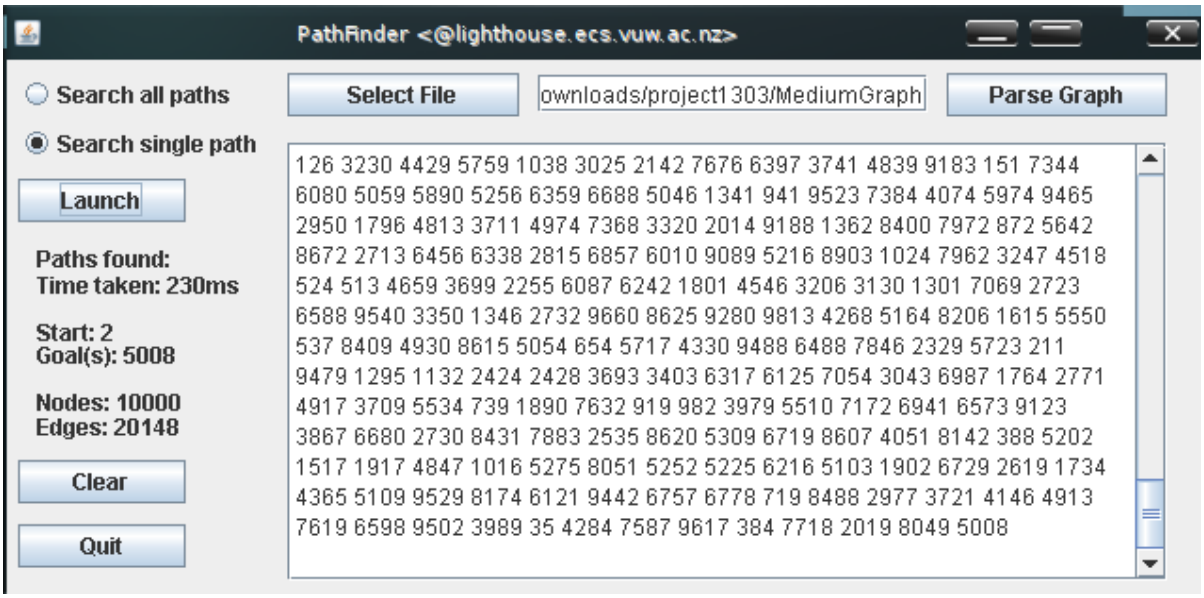
Yao Hong Chun
300232587

Test machine: 2 threads, unknown clock speed (ECS machines). Graph: MediumGraph



When tested on a machine with less system threads, the path to the goal found is significantly longer, more than 2000 steps! This is because the two threads has more unvisited nodes to go through whereas if there are more threads there will be less visited node for each thread and it is likely for two threads to 'collide' and continue in one direction finding a shorter path to the goal. The time taken to search for the path is quicker too, this is because there are fewer threads and there is less synchronization involved.

Test Machine: 16 cores and 64 threads (lighthouse). Graph: MediumGraph



The path found by lighthouse is significantly shorter than the one found by one of the ECS machines and the time taken is around 15 times longer.

Instructions

There are two versions of the program, one with the GUI and one without the GUI.

To run the one without GUI: Open the command line/terminal, browse to the directory and type in:

java –jar project1WithoutGUI.jar <task>

Where <task> is an integer:

0 – search for all paths

1 – search for a single path

You will be prompted with a window to select the graph to be searched.

To run the one with GUI: Just double click on the runnable jar file. The GUI should be easily understood and therefore requires no further explanation. The GUI one can also be run with the command line/terminal, however no arguments are needed.

Java –jar project1WithGUI.jar

Conclusion

Depth first search is a quick way to find a path even on a large graph. However, in the real world, shortest paths are mostly the desired paths therefore depth first search would not be very practical because it doesn't guarantee the shortest path. Although testing has shown that the more threads there are, the likelihood of finding of a shorter path is greater. Testing has also shown that the time taken to search for a path on a machine with more system threads is longer compared to a machine with less system threads due to synchronization overheads. However, when comparing my laptop with my PC, results have shown that machines that are clocked at higher speeds execute faster than machines that are clocked at lower speeds.

Path finding concurrently can prove to be difficult because of its nondeterministic nature. Results are probably in a different order on the next execution if there is more than one. However, if the algorithm is tested on a simple graph with few possible results, it will be easy to verify its correctness. Therefore if it works on a simple graph, it is safe to say that it works on all similar but larger graphs.