CS5011 A1

150006108

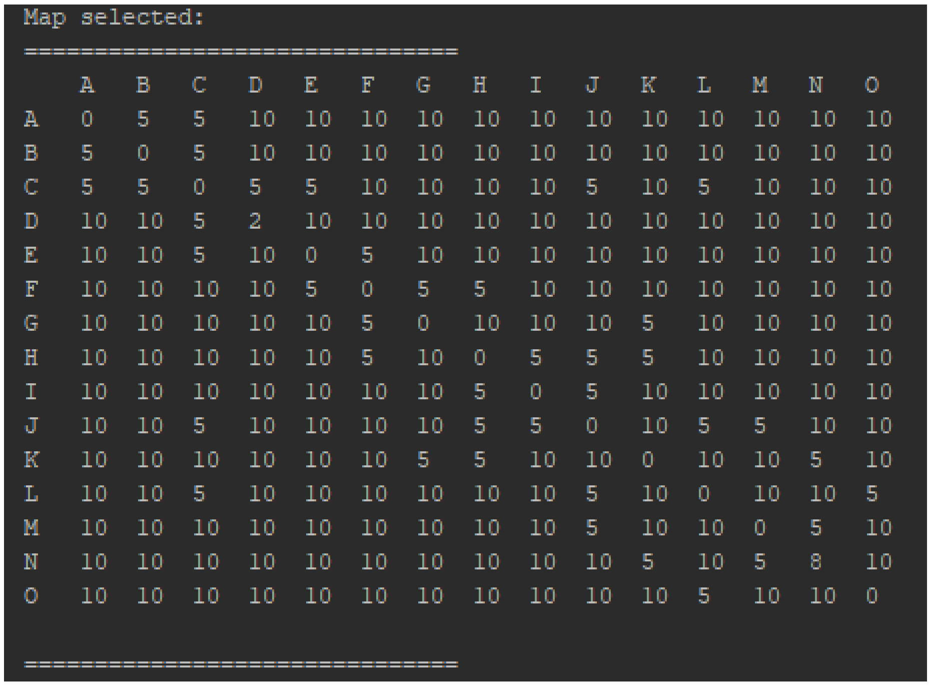
October 2018

1 Search 1

1.1 The Maps

The maps were stored in a two dimensional array, acting as a matrix storing which nodes of the graph were connected to which. The initial state is denoted by a 2, the goal state is denoted by an 8, and a connection between two nodes is denoted by a 5.

When printed, the selected map looks like this:



```
Map selected:
================================
    A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
A   0   5   5   10  10  10  10  10  10  10  10  10  10  10  10
B   5   0   5   10  10  10  10  10  10  10  10  10  10  10  10
C   5   5   0   5   5   10  10  10  10  5   10  5   10  10  10
D   10  10  5   2   10  10  10  10  10  10  10  10  10  10  10
E   10  10  5   10  0   5   10  10  10  10  10  10  10  10  10
F   10  10  10  10  5   0   5   5   10  10  10  10  10  10  10
G   10  10  10  10  10  5   0   10  10  10  5   10  10  10  10
H   10  10  10  10  10  5   10  0   5   5   5   10  10  10  10
I   10  10  10  10  10  10  10  5   0   5   10  10  10  10  10
J   10  10  5   10  10  10  10  5   5   0   10  5   5   10  10
K   10  10  10  10  10  10  5   5   10  10  0   10  10  5   10
L   10  10  5   10  10  10  10  10  10  5   10  0   10  10  5
M   10  10  10  10  10  10  10  10  10  5   10  10  0   5   10
N   10  10  10  10  10  10  10  10  10  10  5   10  5   8   10
O   10  10  10  10  10  10  10  10  10  10  10  5   10  10  0

================================
```

Figure 1: Map1 printed to console after being selected.

Looking at the table above, the initial state can be found to be 'D', as a 2 can found where 'D' is "connected" to itself. The same goes for the goal state.

The maps are stored in a class *WorldMap*, where a switch statement can be found in the class constructor. The constructor takes an integer parameter, which stores the desired map to select. If a matching map cannot be found, the system exits.

The class contains methods to retrieve the start and goal nodes, as well as a *toString* override, providing "pretty print" functionality for the table. A *getConnected* method is also implemented, which takes a node value as a parameter, and returns a list of all connected nodes to it. This simply iterates throughout nodes column in the map and stores all indexes where a 5 is located, representing a connection.

1.2 Breadth First Search

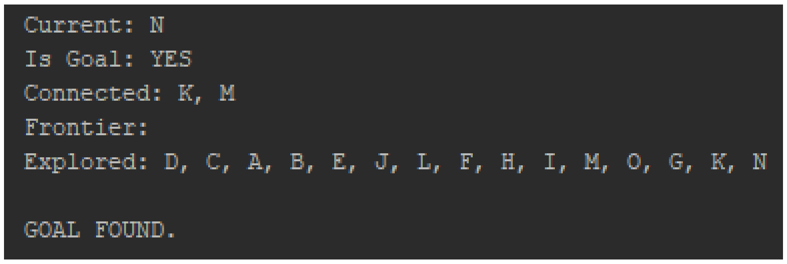
Breadth first inspects all nodes on the same depth level, before moving on to inspect the children of all the previously inspected nodes. In essence, my algorithm starts at the initial state, adds all it's children to the frontier in order that it finds them, and moves on to the the first node in the frontier. It checks whether this is the goal state. If not, any children of this node that have not already been visited or contained within the frontier are added to the end of the frontier, and we then move on to the next node in the frontier, moving the current node from the frontier into the explored list. This cycle continues until either the goal is found, our all available nodes have been explored.

1.3 Depth First Search

Depth first is very similar to breadth first, except it inspects children first rather than every node on a certain depth. The algorithm is much the same as breadth first, but when choosing the next node to visit, instead of taking the *first* node in the frontier, we take the *last*. Breadth first is a FIFO (first in, first out) algorithm, where as depth first is a LIFO (last in, first out) algorithm. This means the most recent node added to the frontier will be the next inspected.

1.4 Testing and Evaluation

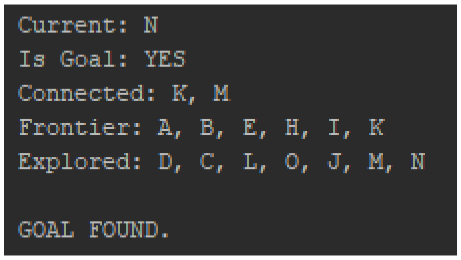
The algorithms were tested using each of the maps contained within *maps1.txt*. Below are the final outputs for both breadth first and depth first when searching for the goal in Map 1.



```
Current: N
Is Goal: YES
Connected: K, M
Frontier:
Explored: D, C, A, B, E, J, L, F, H, I, M, O, G, K, N

GOAL FOUND.
```

Figure 2: Final inspection order of breadth first on Map 1.

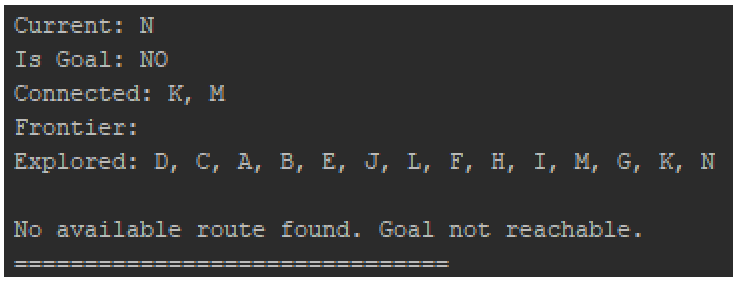


```
Current: N
Is Goal: YES
Connected: K, M
Frontier: A, B, E, H, I, K
Explored: D, C, L, O, J, M, N

GOAL FOUND.
```

Figure 3: Final inspection order of depth first on Map 1.

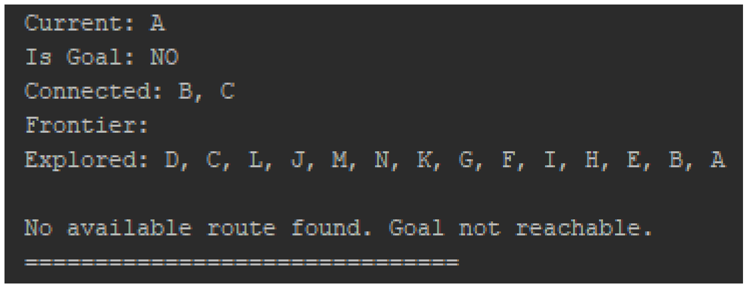
It is evident that depth first inspected many fewer nodes before finding the goal state. However, this is largely because of the design of the map. Since my implementation removes the possibility of loops occurring, both are complete, meaning they will always find the goal state, provided the goal is reachable. Both are optimal too, meaning that they will return the path of the least cost.



```
Current: N
Is Goal: NO
Connected: K, M
Frontier:
Explored: D, C, A, B, E, J, L, F, H, I, M, G, K, N

No available route found. Goal not reachable.
================================
```

Figure 4: Breadth first discovering goal is not reachable in Map 3.



```
Current: A
Is Goal: NO
Connected: B, C
Frontier:
Explored: D, C, L, J, M, N, K, G, F, I, H, E, B, A

No available route found. Goal not reachable.
================================
```

Figure 5: Depth first discovering goal is not reachable in Map 3.

As can be seen above when inspecting Map 3, both algorithms took different routes, yet both eventually exhausted all possible connections without finding the goal. This is because the map was designed so the goal was unreachable. This map is a great way to illustrate how the different algorithms take different routes.

2 Search 2

Implementing the informed searches meant completely changing my approach from the first part. Where the uninformed searches only cared about the nodes themselves, the informed searches care much more on the connections between the nodes.

2.1 WorldMap 2.0

To incorporate these changes, I needed to make additions to the class containing the map. It wasn't just containing the connections anymore, it was now containing the coordinates each node was located at on the grid overlay for the map. This grid provides the foundation for our heuristic.

2.2 Best-First Search

The idea of best first is to take all available connections and order them from shortest to longest, then explore the connection that is the shortest. Originally, we assumed all connections had a length of 1. But now, with the introduction of the grid, we can calculate the distance between any two points. This can be used to represent the length of a connection, and as such, can be used as our heuristic for our best-first algorithm.

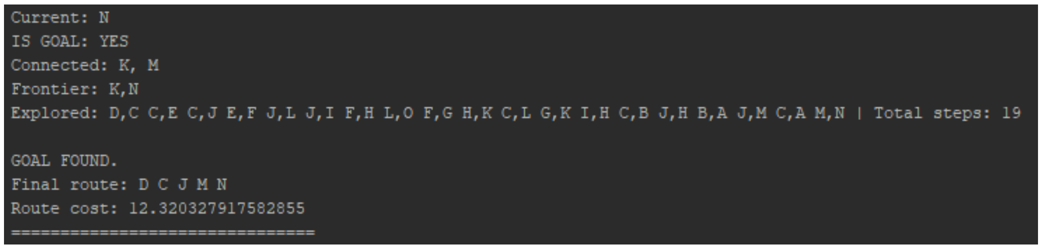
Before, we would add a nodes children to the frontier and choose either the first or last in the list. But now, we must change. Instead, we add the *vertices* leading away from the parent node to the frontier, and order them all in ascending order of size. We then choose the connection with the lowest path cost to travel down, and inspect the node at the end.

Rather than constantly calculating connection lengths, when the map is selected, the system automatically builds a matrix much like the original map which stores the lengths of all node connections. This provides a quick look-up functionality for any connection within a map.

2.3 A* Search

A* search is very similar to best-first, except it's more directional. Instead of just using a connection length as it's heuristic, it also uses the position of a node relative to the goal state. This is represented by using the grid to calculate the distance between the node at the end of a connection and the goal state. This is then added to the connection length, and then all connections are sorted in ascending order of this new heuristic. This ensures that longer connections closer to the goal have a higher priority over short connections further away from the goal. This means a route to the goal can be found in a much shorter time.

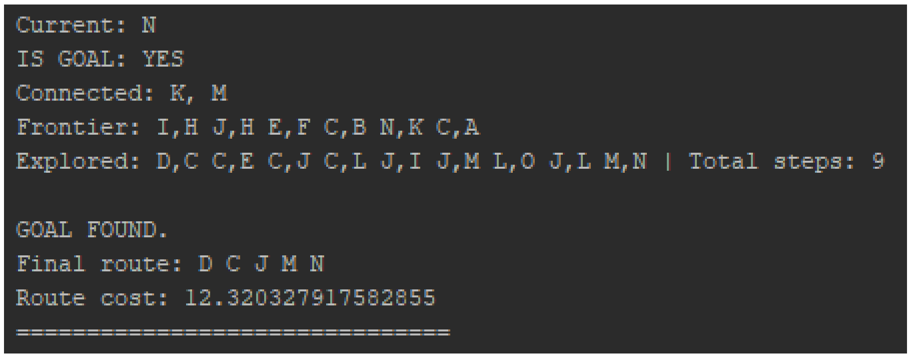
2.4 Testing and Evaluation



```
Current: N
IS GOAL: YES
Connected: K, M
Frontier: K,N
Explored: D,C C,E C,J E,F J,L J,I F,H L,O F,G H,K C,L G,K I,H C,B J,H B,A J,M C,A M,N | Total steps: 19

GOAL FOUND.
Final route: D C J M N
Route cost: 12.320327917582855
==============================
```

Figure 6: Best-first route for Map 1.

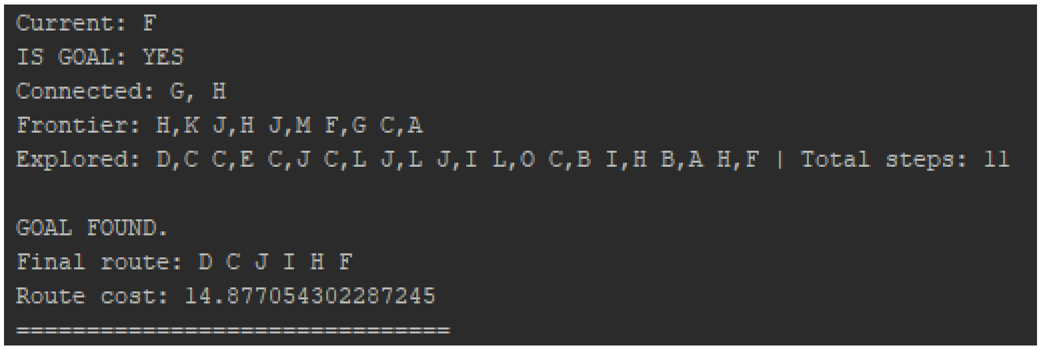


```
Current: N
IS GOAL: YES
Connected: K, M
Frontier: I,H J,H E,F C,B N,K C,A
Explored: D,C C,E C,J C,L J,I J,M L,O J,L M,N | Total steps: 9

GOAL FOUND.
Final route: D C J M N
Route cost: 12.320327917582855
==============================
```

Figure 7: A* route for Map 1.

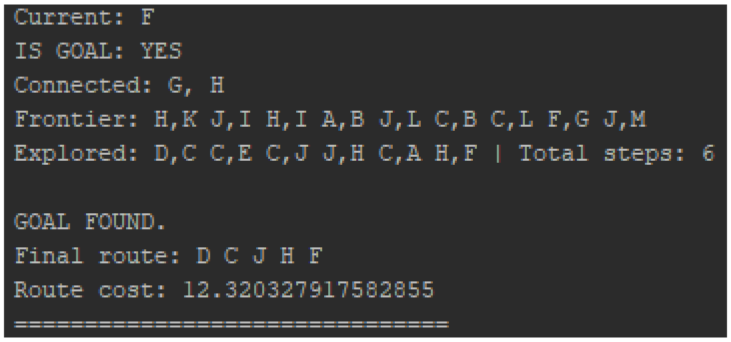
Above are the results of both algorithms attempting to find a route for Map 1. The same route is found by both, but A* finds it in fewer steps. This is an effect of the new directional heuristic.



```
Current: F
IS GOAL: YES
Connected: G, H
Frontier: H,K J,H J,M F,G C,A
Explored: D,C C,E C,J C,L J,L J,I L,O C,B I,H B,A H,F | Total steps: 11

GOAL FOUND.
Final route: D C J I H F
Route cost: 14.877054302287245
================================
```

Figure 8: Best-first route for Map 2.

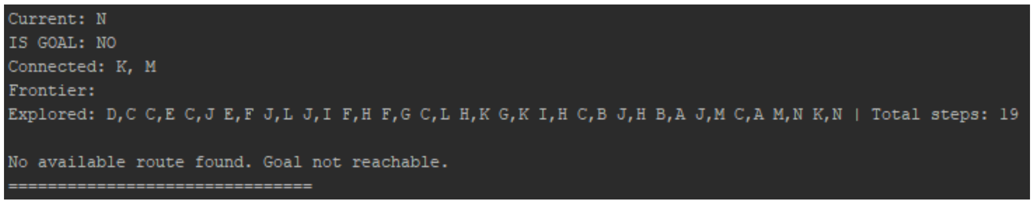


```
Current: F
IS GOAL: YES
Connected: G, H
Frontier: H,K J,I H,I A,B J,L C,B C,L F,G J,M
Explored: D,C C,E C,J J,H C,A H,F | Total steps: 6

GOAL FOUND.
Final route: D C J H F
Route cost: 12.320327917582855
================================
```

Figure 9: A* route for Map 2.

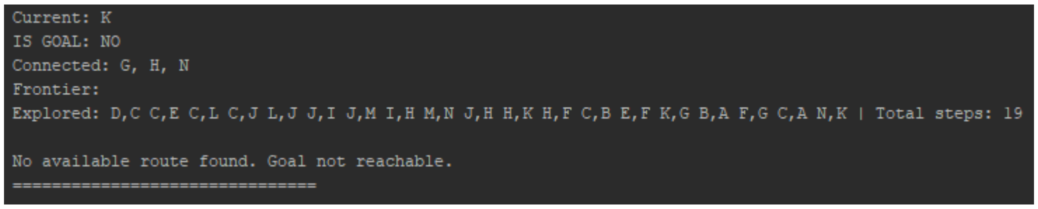
Above are the results for both algorithms attempting to find a route for Map 2. You will notice that best-first decided to use a different route to A*, and as such, A* has a shorter total path cost. A* also discovered it's route in almost half as many steps as best-first, meaning it performed substantially better on Map 2 than best-first.



```
Current: N
IS GOAL: NO
Connected: K, M
Frontier:
Explored: D,C C,E C,J E,F J,L J,I F,H F,G C,L H,K G,K I,H C,B J,H B,A J,M C,A M,N K,N | Total steps: 19

No available route found. Goal not reachable.
===============================
```

Figure 10: Best-first attempting to find route for Map 3.



```
Current: K
IS GOAL: NO
Connected: G, H, N
Frontier:
Explored: D,C C,E C,L C,J L,J J,I J,M I,H M,N J,H H,K H,F C,B E,F K,G B,A F,G C,A N,K | Total steps: 19

No available route found. Goal not reachable.
===============================
```

Figure 11: A* attempting to find route for Map 3.

Above are the results for both algorithms attempting to find a route for Map 3. As we already know, the goal in Map 3 is unreachable, so this test allows us to observe how a difference in heuristic effects the search order of each algorithm.

3 Running

To run part 1, use the command **java -jar Search1.jar** [**map number**], where the argument is the map number you wish to run breadth and depth search on.

To run part 2, use the command **java -jar Search2.jar** [**map number**], where again the argument is the map number you wish to run best-first and A* search on.

4 Evaluation and Conclusion

In conclusion, I feel I have successfully implemented all four search algorithms, providing functionality for them to smoothly and efficiently traverse the given maps to find a route from the start to the goal.

Regarding part 1, both algorithms are essentially equal, with their efficiency dependant on the graph structure more than anything else. If the goal is right next to the start, breadth first will win, whereas if the goal is underneath the start, depth first shall win.

Regarding part 2, A* is clearly the better of the two algorithms. Introducing the directional part to the heuristic allows it to outperform the best-first algorithm.

Looking back at part 1, I could take the functionality of storing vertices and use that to build the route found by both breadth and depth first.

Word Count: 1294
Estimated Time Taken: 20 hours