Paul Philip Mitchell

TDT4136 Assignment 3 – Using the A* Algorithm

General information

I downloaded the original algorithm from http://scriptogr.am/jdp/post/pathfinding-with-python-graphs-and-a-star. My code has been heavily modified to meet the requirements set by this assignment.

Because the images will be so small if I paste the entire console, I have only included images of the initial grid and the resulting grid. Other information which is also present in the console, such as *start node*, *end node, array of obstacles* and *array of path*, will just be written above the images. Full images can be found in the "images" folder of this delivery. *"a"* at the end of the file name indicates that it is an image of the grid using A*, *"d"* at the end indicates that it is an image of the grid using Dijkstra's.

I have struggled really hard to complete this assignment. When the assignment was first handed out, I tried for a long time to write the entire implementation by myself, but after four days of failure, I gave up. I then tried to translate the pseudo-code on Wikipedia, but then I struggled with representing the graph correctly with paths, open nodes and closed nodes in the console output. Finally, I found the implementation on the website mentioned above, and I sat for 5 days trying to make it work with the sort of input we were handed (the boards), and correctly displaying the output.

As you will see in this report, there are a few things I did not have time to implement;

(1) Displaying the output prettier. This would either be done with Tkinter or the Python Imaging Library. I did not have time to complete this. If I had the time to work on it a few more hours, I would have used the "*colorama"* library for python, which lets me print to the console with both a foreground and background color for each character. This would have been useful for improving readability on the final output. Colorama also fixes issues of printing out ANSI escapes to DOS-based shells.

(2) BFS. Modifying the code to use Dijkstra's was easy, that was just to change the lambda function on line 21 to instead only use the nodes *g* value. However, since I began using the downloaded code, I have used *sets* for storing the list of open and closed nodes. If you "pop" from a set, you will get an arbitrary value, which is why this does not work with BFS, where I would need a FIFO data structure. If I had the time, I would rewrite the algorithm to use simple *lists* instead of *sets* for storing open and closed nodes, since speed performance is not really an issue in this assignment. This would make it possible to run both A*, BFS and Dijkstra's.

The colored console output works in JetBrains' PyCharm environment. It does not work in Python IDLE on Windows, and I am pretty sure it won't work in a DOS-shell, such as Windows' *command prompt.* If the output displays incorrectly in your environment, simply remove the ANSI escapes ("\033 *etc*) on line 156, 159, 161.

Paul Philip Mitchell

SUB PROBLEM A.1: GRIDS WITH OBSTACLES

General information and issues I encountered

If you would like to view the resulting grid containing only the path when you run the program, not nodes in the closed and open set, you will have to comment out line 149 through 153.

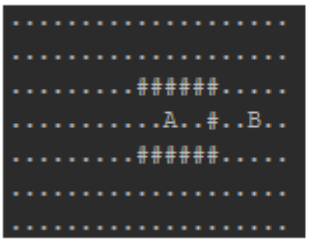
Symbol declaration:

"."	-	clear path
"#"	-	blocked path
"A"	-	start node
"B"	-	goal node
"o"	-	path

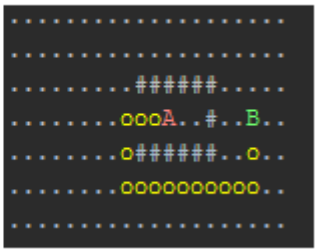
A1.1 BOARD1-1.TXT VISUALIZATION

Start node: (11, 3)
End node: (17, 3)
Obstacles found: [[9, 2], [10, 2], [11, 2], [12, 2], [13, 2], [14, 2], [14, 3], [9, 4], [10, 4], [11, 4], [12, 4], [13, 4], [14, 4]]
Path found: [(11 3), (10 3), (9 3), (8 3), (8 4), (8 5), (9 5), (10 5), (11 5), (12 5), (13 5), (14 5), (15 5), (16 5), (17 5), (17 4), (17 3)]

initial grid



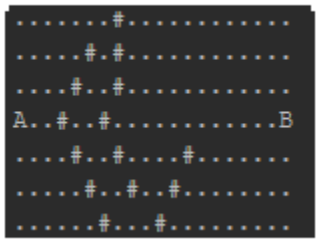
resulting grid



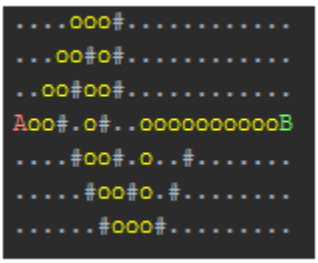
A1.2 BOARD1-2.TXT VISUALIZATION

Start node: (0, 3)
End node: (19, 3)
Obstacles found: [[7, 0], [5, 1], [7, 1], [4, 2], [7, 2], [3, 3], [6, 3], [4, 4], [7, 4], [12, 4], [5, 5], [8, 5], [11, 5], [6, 6], [10, 6]]
Path found: [(0 3), (1 3), (2 3), (2 2), (3 2), (3 1), (4 1), (4 0), (5 0), (6 0), (6 1), (6 2), (5 2), (5 3), (5 4), (6 4), (6 5), (7 5), (7 6), (8 6), (9 6), (9 5), (9 4), (9 3), (10 3), (11 3), (12 3), (13 3), (14 3), (15 3), (16 3), (17 3), (18 3), (19 3)]

initial grid



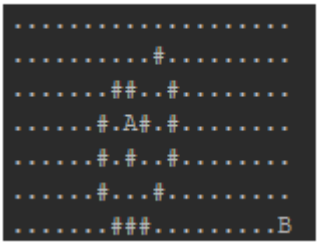
resulting grid



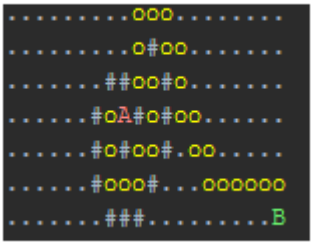
A1.3 BOARD1-3.TXT VISUALIZATION

Start node: (8, 3)
End node: (19, 6)
Obstacles found: [[10, 1], [7, 2], [8, 2], [11, 2], [6, 3], [9, 3], [11, 3], [6, 4], [8, 4], [11, 4], [6, 5], [10, 5], [7, 6], [8, 6], [9, 6]]
Path found: [(8 3), (7 3), (7 4), (7 5), (8 5), (9 5), (9 4), (10 4), (10 3), (10 2), (9 2), (9 1), (9 0), (10 0), (11 0), (11 1), (12 1), (12 2), (12 3), (13 3), (13 4), (14 4), (14 5), (15 5), (16 5), (17 5), (18 5), (19 5), (19 6)]

initial grid



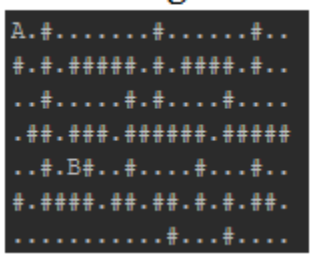
resulting grid



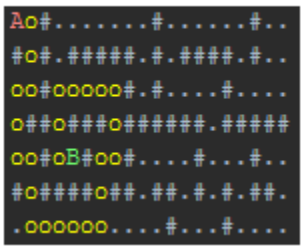
A1.4 BOARD1-4.TXT VISUALIZATION

Start node: (0, 0)
End node: (4, 4)
Obstacles found: [[2, 0], [10, 0], [17, 0], [0, 1], [2, 1], [4, 1], [5, 1], [6, 1], [7, 1], [8, 1], [10, 1], [12, 1], [13, 1], [14, 1], [15, 1], [17, 1], [2, 2], [8, 2], [10, 2], [15, 2], [1, 3], [2, 3], [4, 3], [5, 3], [6, 3], [8, 3], [9, 3], [10, 3], [11, 3], [12, 3], [13, 3], [15, 3], [16, 3], [17, 3], [18, 3], [19, 3], [2, 4], [5, 4], [8, 4], [13, 4], [17, 4], [0, 5], [2, 5], [3, 5], [4, 5], [5, 5], [7, 5], [8, 5], [10, 5], [11, 5], [13, 5], [15, 5], [17, 5], [18, 5], [11, 6], [15, 6]]
Path found: [(0 0), (1 0), (1 1), (1 2), (0 2), (0 3), (0 4), (1 4), (1 5), (1 6), (2 6), (3 6), (4 6), (5 6), (6 6), (6 5), (6 4), (7 4), (7 3), (7 2), (6 2), (5 2), (4 2), (3 2), (3 3), (3 4), (4 4)]

initial grid



resulting grid



Paul Philip Mitchell

SUB PROBLEM A.2: GRIDS WITH DIFFERENT CELL COSTS

General information and issues I encountered

Because I did not have time to create any sort of GUI or image of the output, the current program only outputs text to the console. This makes the final output significantly harder to read than it should be (especially for sub-problem A3). However, I hope that only coloring the actual path the algorithm chooses will be sufficient for this assignment.

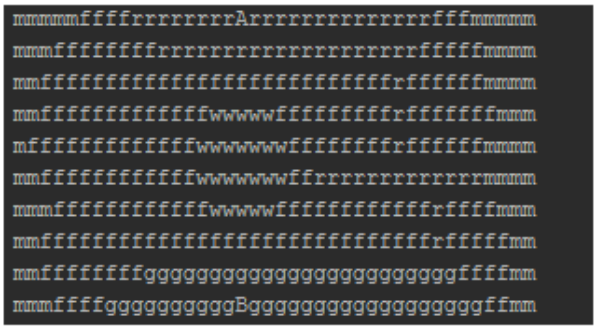
Symbol declaration:

"w"	-	water cell type (cost 100)
"m"	-	mountain cell type (cost 50)
"f"	-	forest cell type (cost 10)
"g"	-	grasslands cell type (cost 5)
"r"	-	road cell type (cost 1)
"A"	-	start node
"B"	-	goal node
"o (o)"	-	path

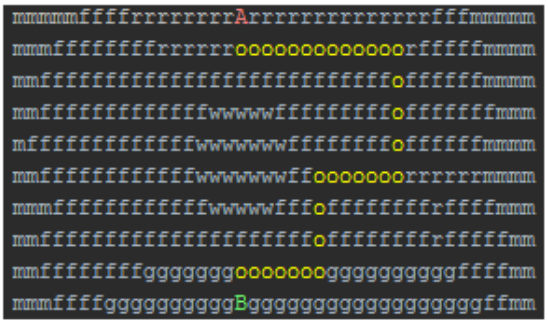
A2.1 BOARD2-1.TXT VISUALIZATION

Start node: (17, 0)
End node: (17, 9)
Path found: [(17 0), (17 1), (18 1), (19 1), (20 1), (21 1), (22 1), (23 1), (24 1), (25 1), (26 1), (27 1), (28 1), (29 1), (29 2), (29 3), (29 4), (29 5), (28 5), (27 5), (26 5), (25 5), (24 5), (23 5), (23 6), (23 7), (23 8), (22 8), (21 8), (20 8), (19 8), (18 8), (17 8), (17 9)]

initial grid



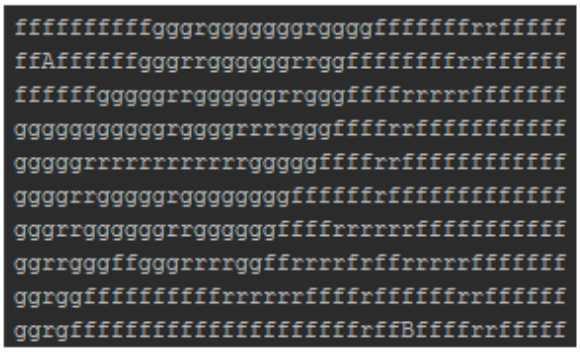
resulting grid



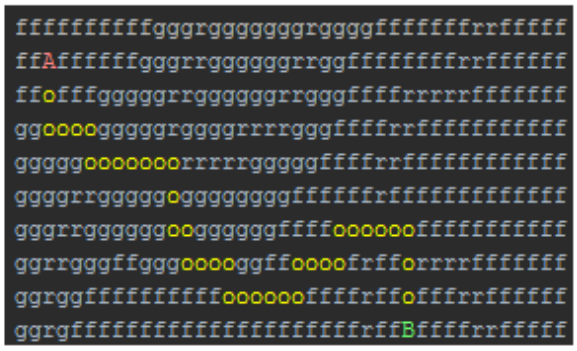
A2.2 BOARD2-2.TXT VISUALIZATION

Start node: (2, 1)
End node: (28, 9)
Path found: [(2 1), (2 2), (2 3), (3 3), (4 3), (5 3), (5 4), (6 4), (7 4), (8 4), (9 4), (10 4), (11 4), (11 5), (11 6), (12 6), (12 7), (13 7), (14 7), (15 7), (15 8), (16 8), (17 8), (18 8), (19 8), (20 8), (20 7), (21 7), (22 7), (23 7), (23 6), (24 6), (25 6), (26 6), (27 6), (28 6), (28 7), (28 8), (28 9)]

initial grid



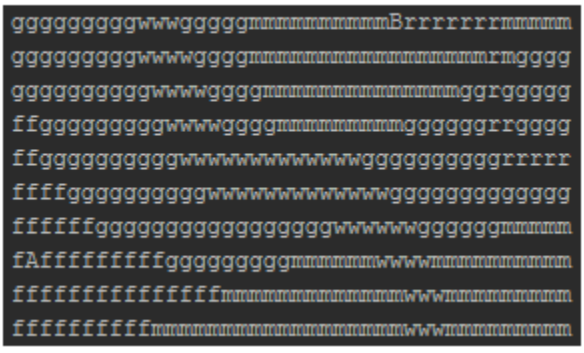
resulting grid



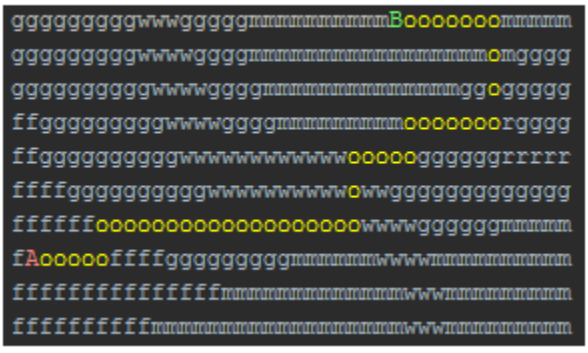
A2.3 BOARD2-3.TXT VISUALIZATION

Start node: (1, 7)
End node: (27, 0)
Path found: [(1 7), (2 7), (3 7), (4 7), (5 7), (6 7), (6 6), (7 6), (8 6), (9 6), (10 6), (11 6), (12 6), (13 6), (14 6), (15 6), (16 6), (17 6), (18 6), (19 6), (20 6), (21 6), (22 6), (23 6), (24 6), (24 5), (24 4), (25 4), (26 4), (27 4), (28 4), (28 3), (29 3), (30 3), (31 3), (32 3), (33 3), (34 3), (34 2), (34 1), (34 0), (33 0), (32 0), (31 0), (30 0), (29 0), (28 0), (27 0)]

initial grid



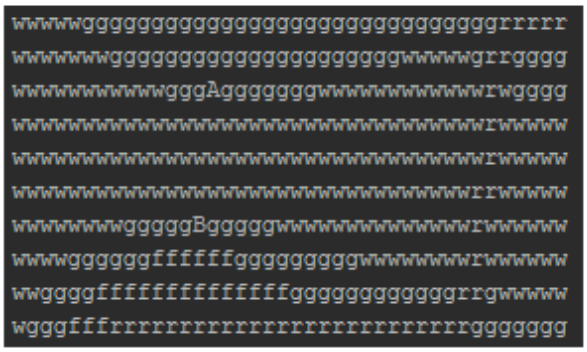
resulting grid



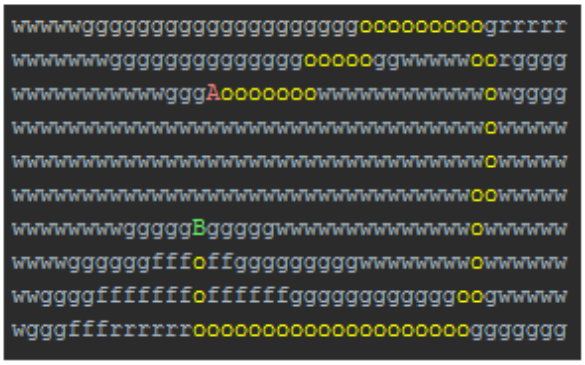
A2.4 BOARD2-4.TXT VISUALIZATION

Start node: (14, 2)
End node: (13, 6)
Path found: [(14 2), (15 2), (16 2), (17 2), (18 2), (19 2), (20 2), (21 2), (21 1), (22 1), (23 1), (24 1), (25 1), (25 0), (26 0), (27 0), (28 0), (29 0), (30 0), (31 0), (32 0), (33 0), (33 1), (34 1), (34 2), (34 3), (34 4), (34 5), (33 5), (33 6), (33 7), (33 8), (32 8), (32 9), (31 9), (30 9), (29 9), (28 9), (27 9), (26 9), (25 9), (24 9), (23 9), (22 9), (21 9), (20 9), (19 9), (18 9), (17 9), (16 9), (15 9), (14 9), (13 9), (13 8), (13 7), (13 6)]

initial grid



resulting grid



Paul Philip Mitchell

SUB PROBLEM A.3: COMPARISON WITH BFS AND DIJKSTRA'S ALGORITHM

General information and issues I encountered

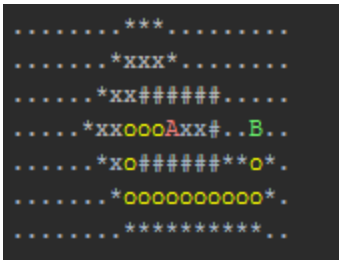
As I've mentioned in the introduction of this delivery, I did not have time to implement BFS, therefore only A* and Dijkstra's are shown. A* outputs are on the left side, Dijkstra's on the right. For sub-problem A3.5 and out, the initial grid is on top, for easier comparison. Because I did not have time to implement any imaging output, the readability is drastically reduced – it is very difficult to see what type of cell the algorithm puts in the open and closed lists. If I had the time to use the *colorama* library, the background of each cell type would be set to its correct color, and that would have increased the readability slightly. However, I hope that this is sufficient for this delivery – the algorithm works as it should, and I would call it a minor implementation detail that it does not output in a very readable way.

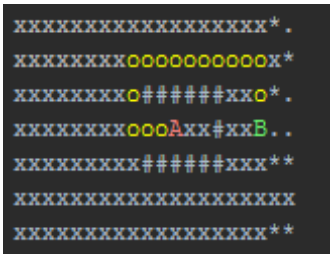
Symbol declaration:

"w"	-	water cell type (cost 100)
"m"	-	mountain cell type (cost 50)
"f"	-	forest cell type (cost 10)
"g"	-	grasslands cell type (cost 5)
"r"	-	road cell type (cost 1)
"*"	-	open nodes
"x"	-	closed nodes
"A"	-	start node
"B"	-	goal node
"o (o)"	-	path

A3.1 BOARD1-1.TXT VISUALIZATION AND COMPARISON

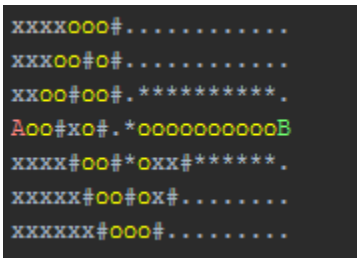
The two algorithms choose different paths, but they both have the same path length. Also, you can see that the Dijkstra output shows that Dijkstra's processes a lot more nodes and put them in the closed nodes list. The A* algorithm processes a minimal amount of nodes, as you can see – and, for that same reason, executes faster than Dijkstra's.

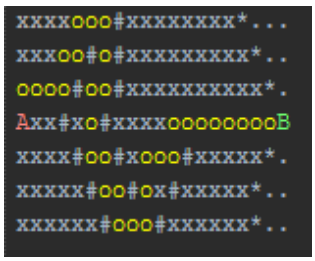




A3.2 BOARD1-2.TXT VISUALIZATION AND COMPARISON

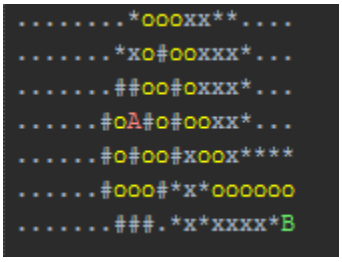
Both A* and Dijkstra's choose nearly the same path, the only difference is that A* rather chooses a straight path near the end, whereas Dijkstra's chooses a stair-like path. Of course, the paths must be nearly the same, because the board itself does not allow major differences. Also, for this board (as well as board 1-1), Dijkstra's processes far more nodes than A*.

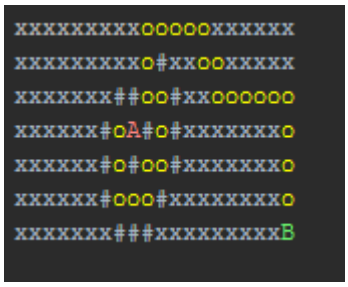




A3.3 BOARD1-3.TXT VISUALIZATION AND COMPARISON

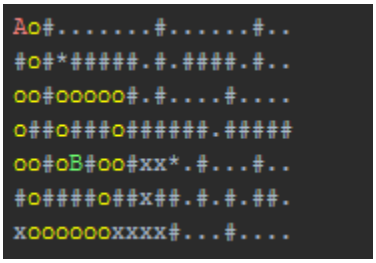
Both algorithms have the same path length, but in this case, A* chooses a more stair-like path, while Dijkstra's chooses a straighter path. The most interesting for this board, however, is how many nodes each algorithm process – A*, with its heuristic, does not spend much time looking left when reaching the top, and therefore saves resources. Dijkstra's however, adds <u>all</u> nodes to its closed list, including all the nodes on the left side, which naturally hogs a lot of resources.

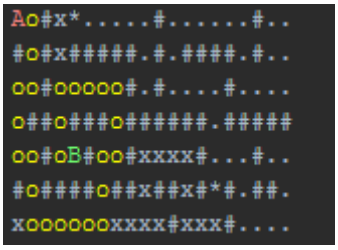




A3.4 BOARD1-4.TXT VISUALIZATION AND COMPARISON

For this board, because the board itself does not allow much difference in paths, both the algorithms choose the exact same path. They process nearly the same amount of nodes, but as you can see, Dijkstra's processes a few more unnecessary nodes near the bottom right corner of the board.

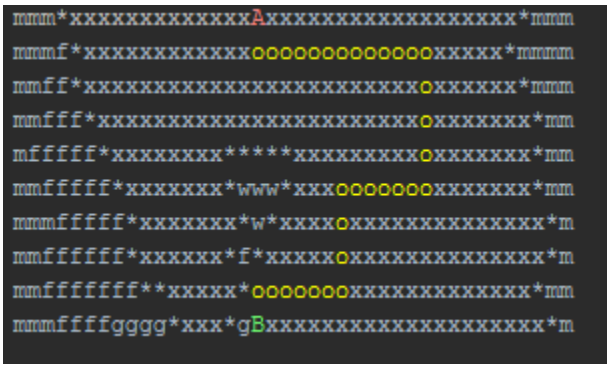


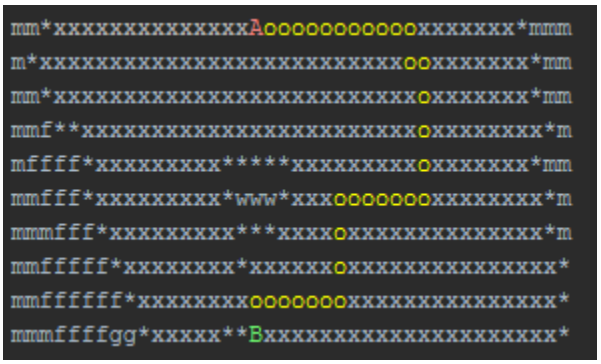


A3.5 BOARD2-1.TXT VISUALIZATION AND COMPARISON

For this sub-problem and the ones proceeding, readability is especially decreased.
The algorithms choose very similar paths. The only differences occur in the last ~9 nodes on the path; they both travel on roads for these last 9 nodes however, so the actual distance covered is the same. The difference in nodes added to open and closed lists is minor.

initial grid

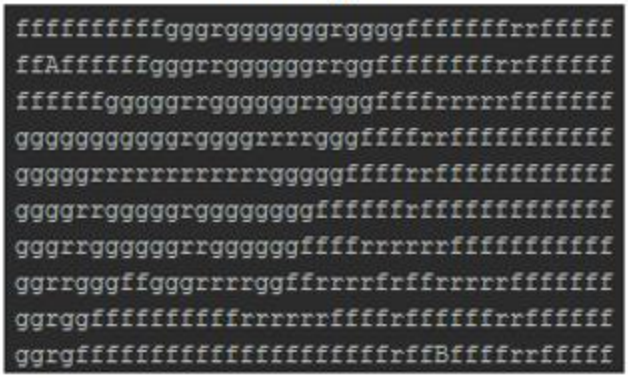


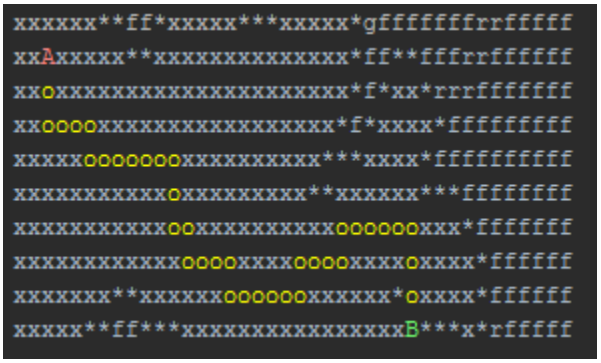


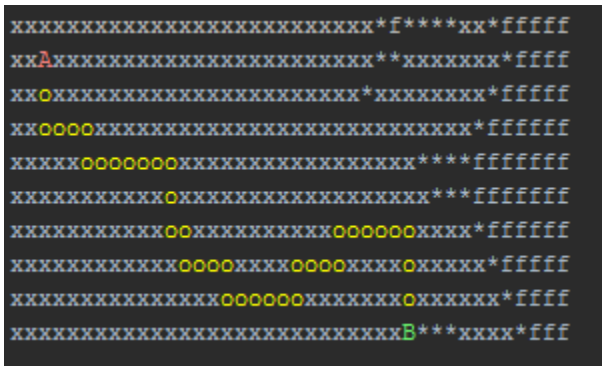
A3.6 BOARD2-2.TXT VISUALIZATION AND COMPARISON

For this board, the algorithms actually produces the exact same path.
The differences occur in the amount of nodes in the open and closed lists – Dijkstra's add all nodes except the far right side of the board to the closed list. This differs from A-Star's execution, where the algorithm is pickier as to which nodes are added to the open and closed lists, respectively. It is clear that A* processes far fewer nodes than Dijkstra's.

initial grid







A3.7 BOARD2-3.TXT VISUALIZATION AND COMPARISON

The path produced by the algorithms only slightly differ – A* chooses a steeper stair-like path, while Dijkstra's is straighter. The distance covered is nevertheless the same, because they both choose to travel on grassland where the differences occur.
Both A* and Dijkstra's include more or less the same list of closed nodes. However, Dijkstra's has processed a few more nodes on the right hand side, comparing to A*. This also applies to the list of open nodes.

initial grid

A3.8 BOARD2-4.TXT VISUALIZATION AND COMPARISON

Both algorithms choose the exact same path.
The open and closed lists are nearly identical for the two algorithms, but as you can see on the line above the goal node “B”, A* chooses to add more of the water cell types to its list of open nodes during execution, while Dijkstra’s adds some of them to the list of closed nodes instead.

initial grid