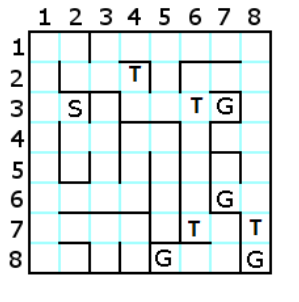
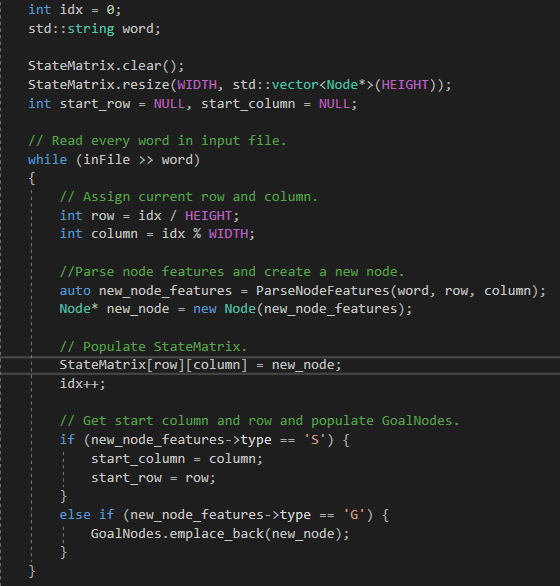
**CSE\_4082 Project 1 Design Document**

**Alperen Bayraktar 150116501**

**Onur Can Yücedağ 150116825**

****

**Creation of the Map:**

In main(), the .txt file we’ve given as input gets transformed into a 8x8 matrix called “*StateMatrix”* .

inFile is the maze.txt that represents the maze cells in words.

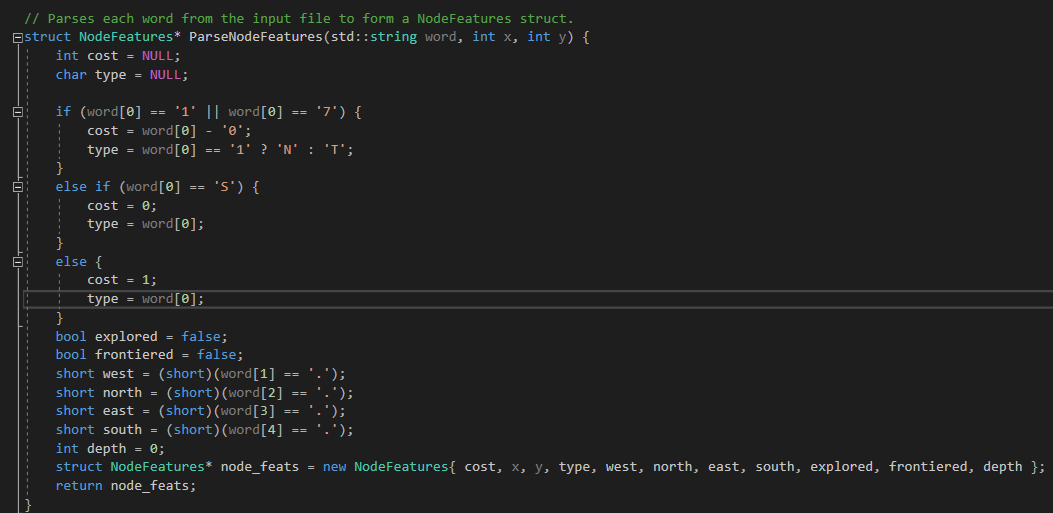
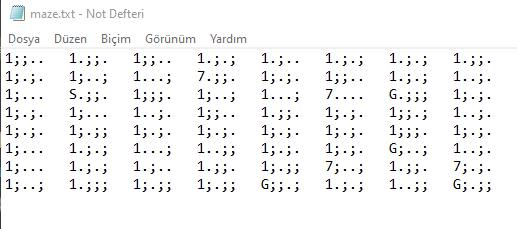
To do the transformation, *“ParseNodeFeatures”* method is used. A node is getting created with the features(struct) as a return value.

Then this new node is added to the StateMatrix.

Index gets incremented and then the type control is made.

If the type of a cell is “S”, then that index of the StateMatrix is the start point.

If the type of a cell is “G”, then that index of the StateMatrix is a goal point.



First letter represents the type, thus the cost.

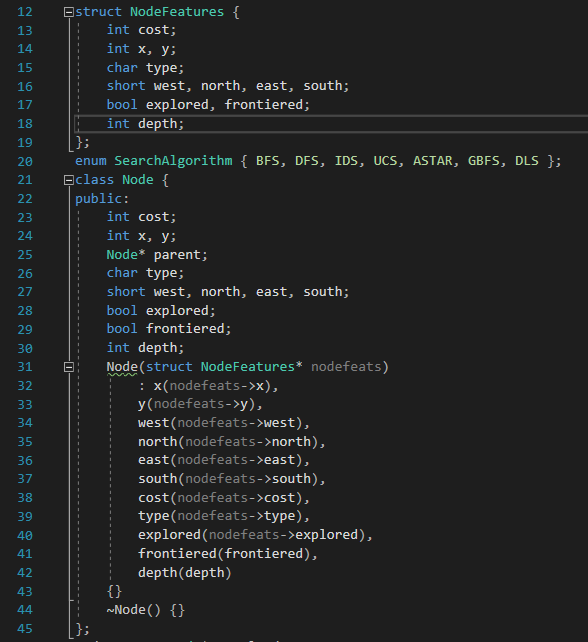
The rest 4 letters are the direction status in respect to “west, north, east, south” . “;” if it’s blocked and “.” If it’s not blocked.

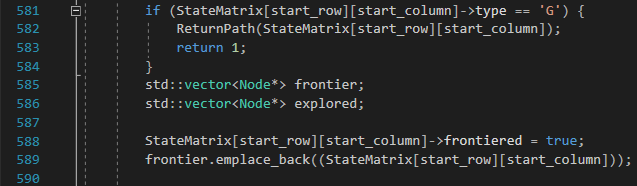
Costs and types are set according to the first letter than directions are converted to 1’s and 0’s.

The explored and frontiered blooleans are set to false for initialization.

Finally, the struct gets created and fed to the return.

Node Struct and class in the header file:

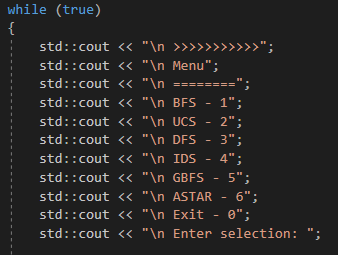




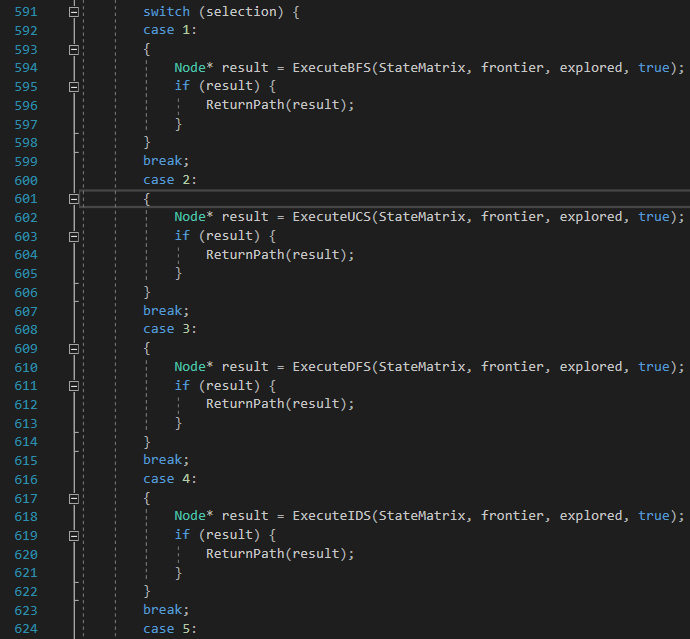
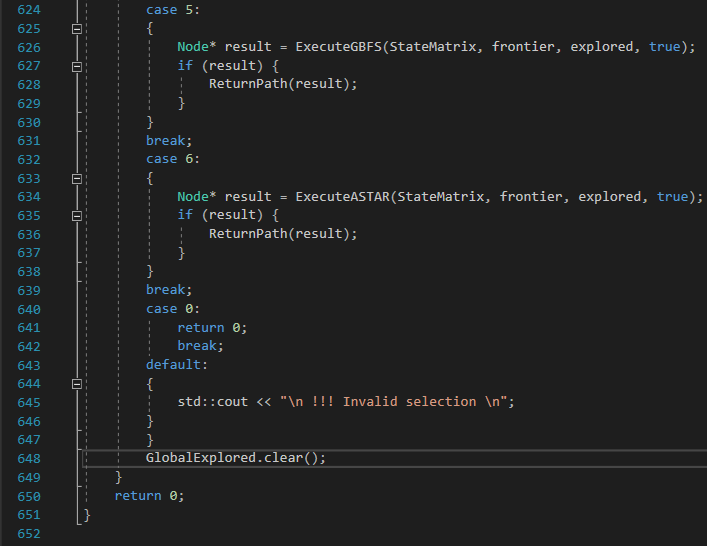
In main(), as a start, if the start node is also Goal, the program exits with success.

Frontier and expanded lists are created.

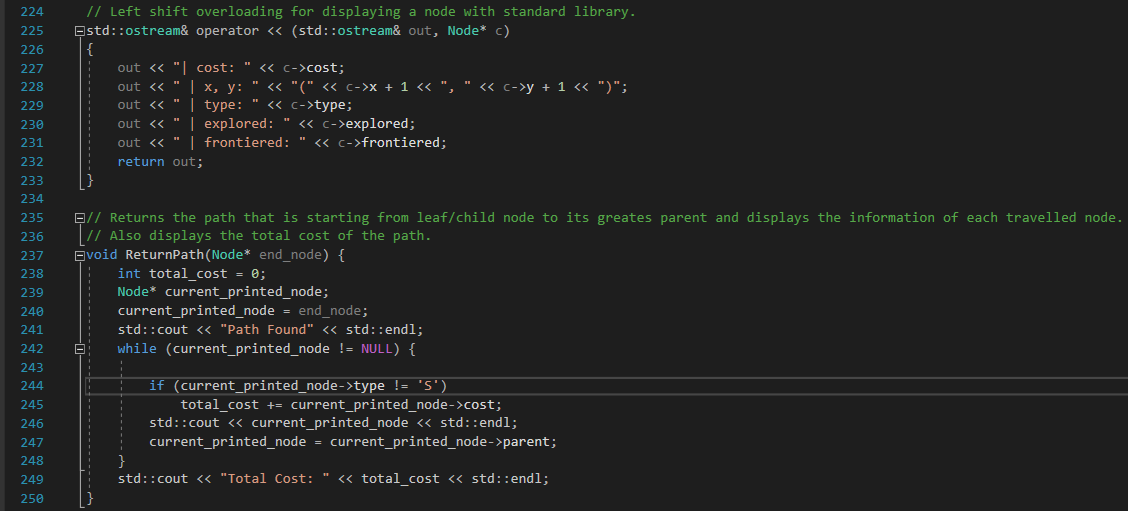
Start node gets added to frontier list.

Then, according to the user input, a method is implemented on the maze:

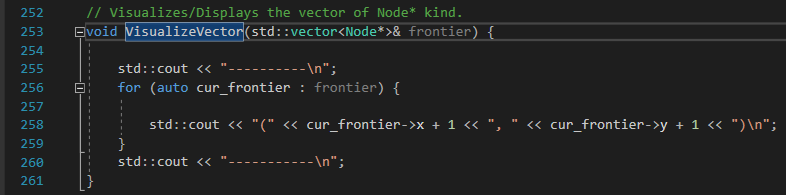
This is a menu-driven code.



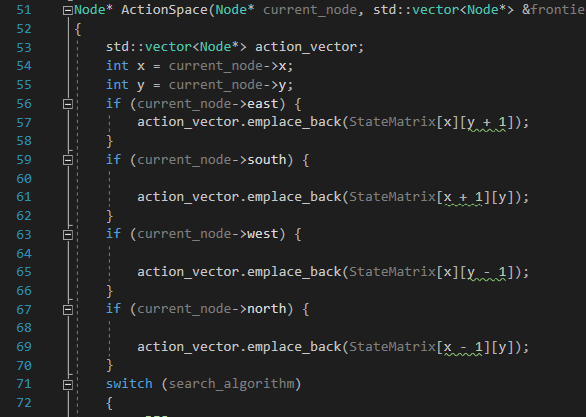
Line 648 is for IDS.



ReturnPath() function is for printing both the path to goal state and the cost. We’ve overloaded the print of a node and used it in the method. It simply goes to parent until it is the start cell.



This method is for printing the vectors like expanded or frontier.



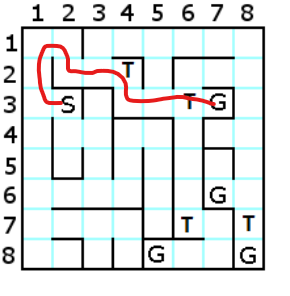
ActionSpace() method is for creating the frontier list for each algorithm, it has modifications for each algorithm.

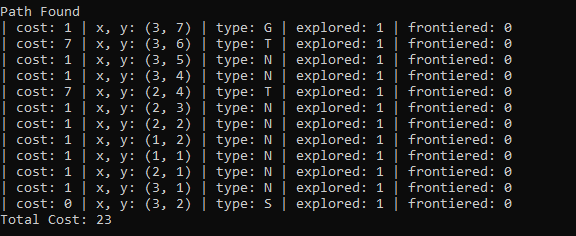
Since the given priority is east->south->west->north, it adds nodes to the action vector respect to this.

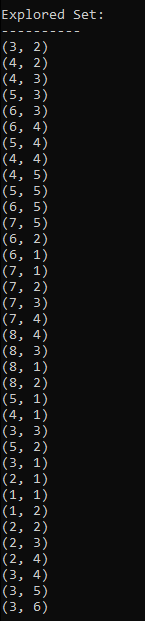
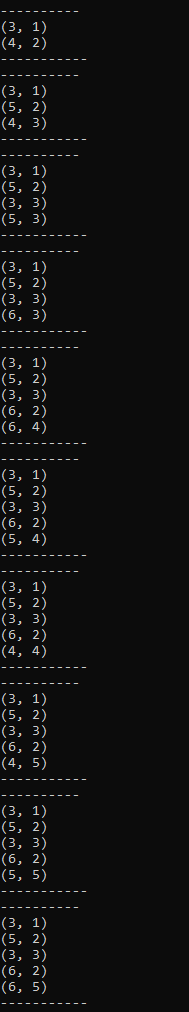
The cases will be explained in each algorithms own section…

* Frontier list for each iteration is printed.
* Expanded list is printed.
* Total cost is printed.

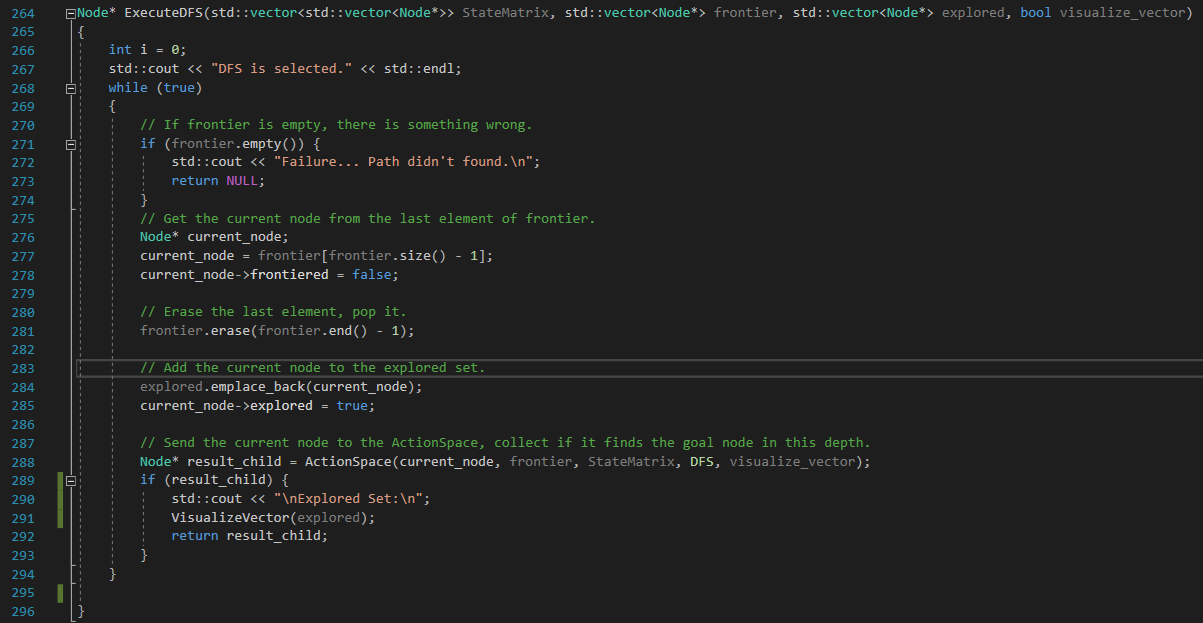
1. **Depth First Search**



And the path it followed: 



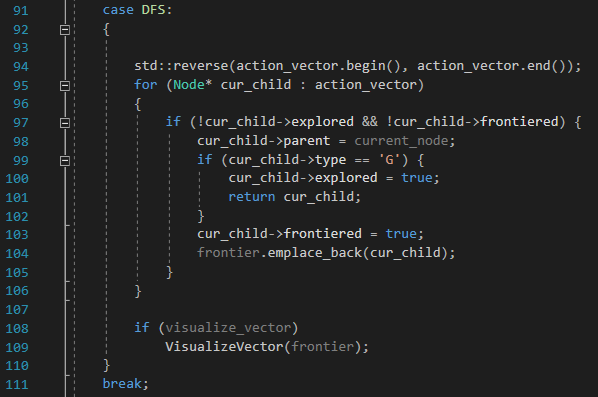
(Expanded Nodes) (Frontier list per iteration)



The last node gets extracted from the frontier.

Gets added to expanded list.

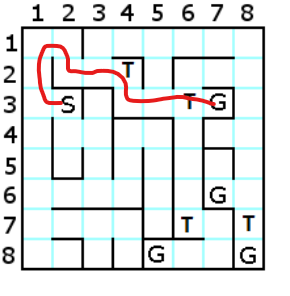
Then gets fed to ActionSpace() method to get its children to frontier.

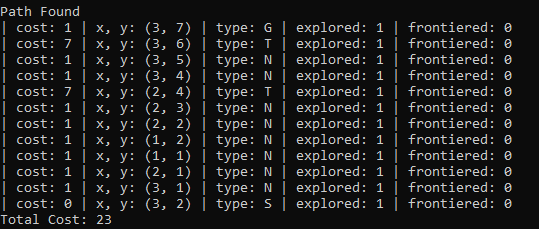


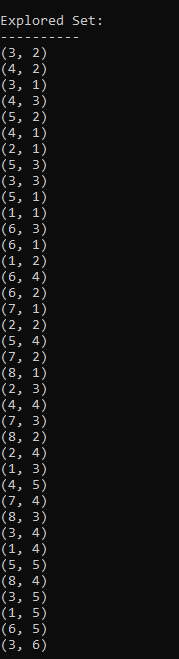
Action Vector has the current nodes children, so it needs to get reversed because DFS uses a stack data structure.

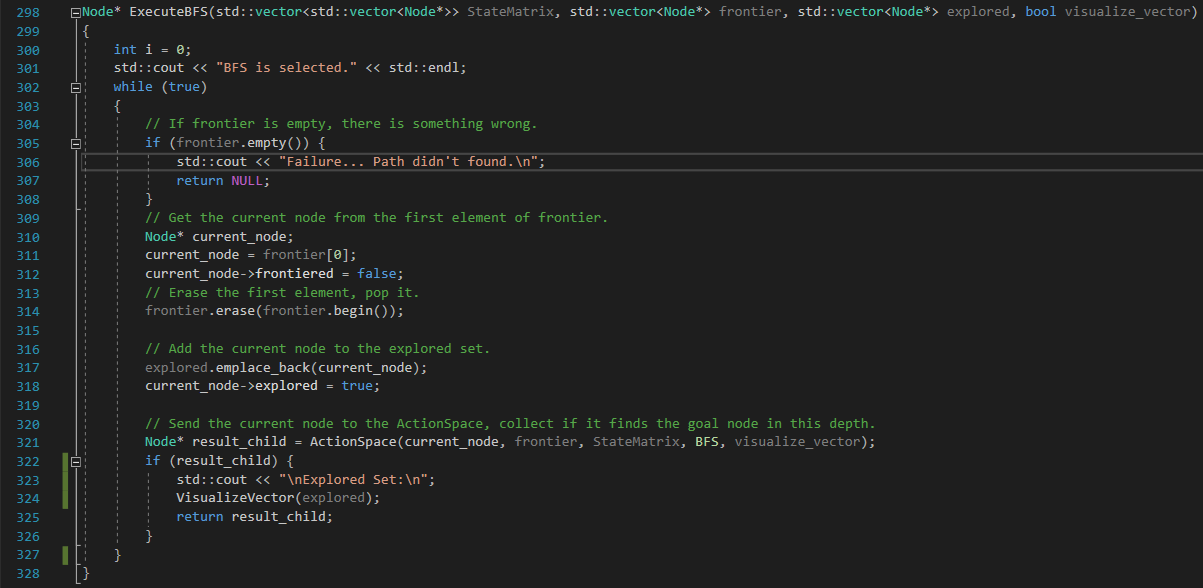
If the child is not in frontier or expanded list, it gets added to frontier list if it is not a Goal state.

Then frontier list gets printed by VisualizeVector().

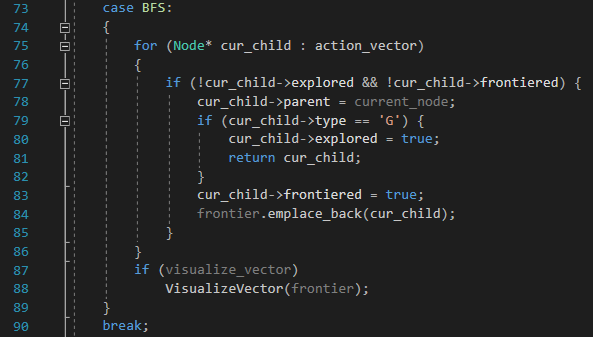
1. **Breadth First Search**



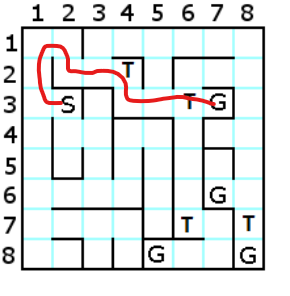


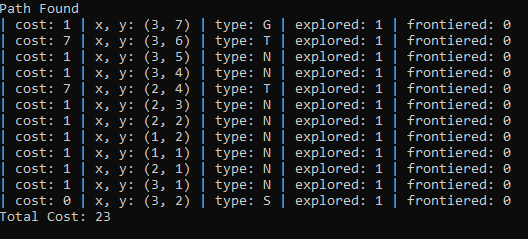


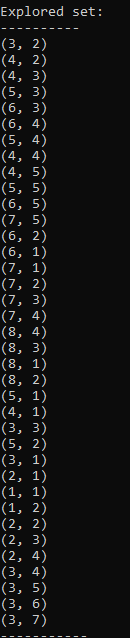
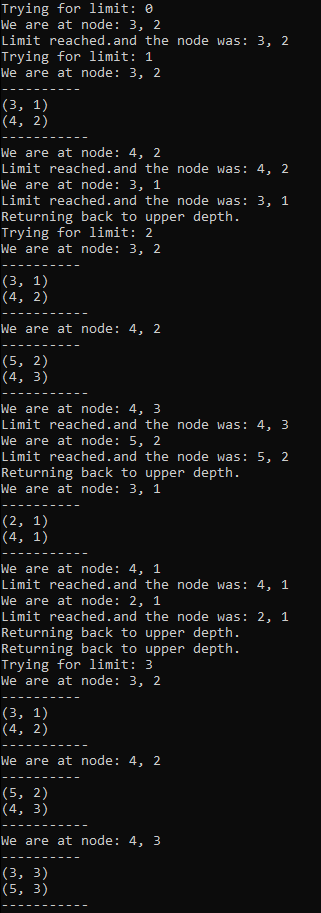
Every step is same with DFS except that the last node getting extracted. This time the first node gets extracted every time and ActionSpace() adds the nodes in their normal order.



1. **Iterative Deepening**

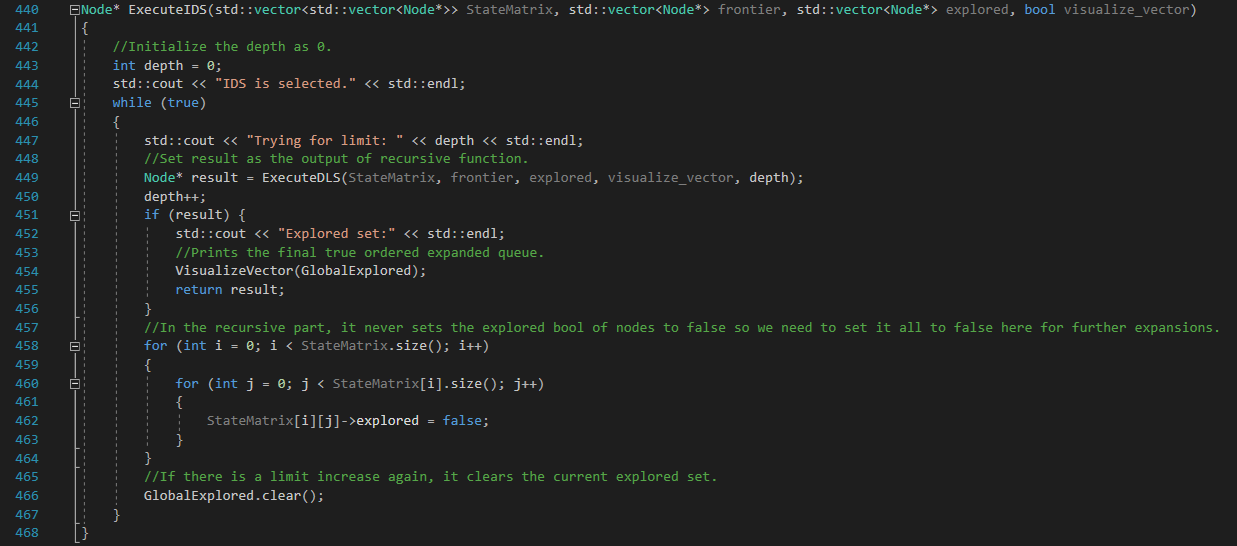






To trace recursiveness, we’ve added some print outs.

The implementation was made similar to the course slides.

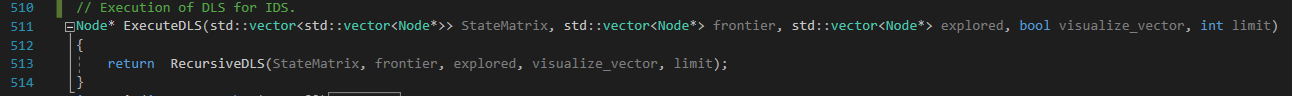


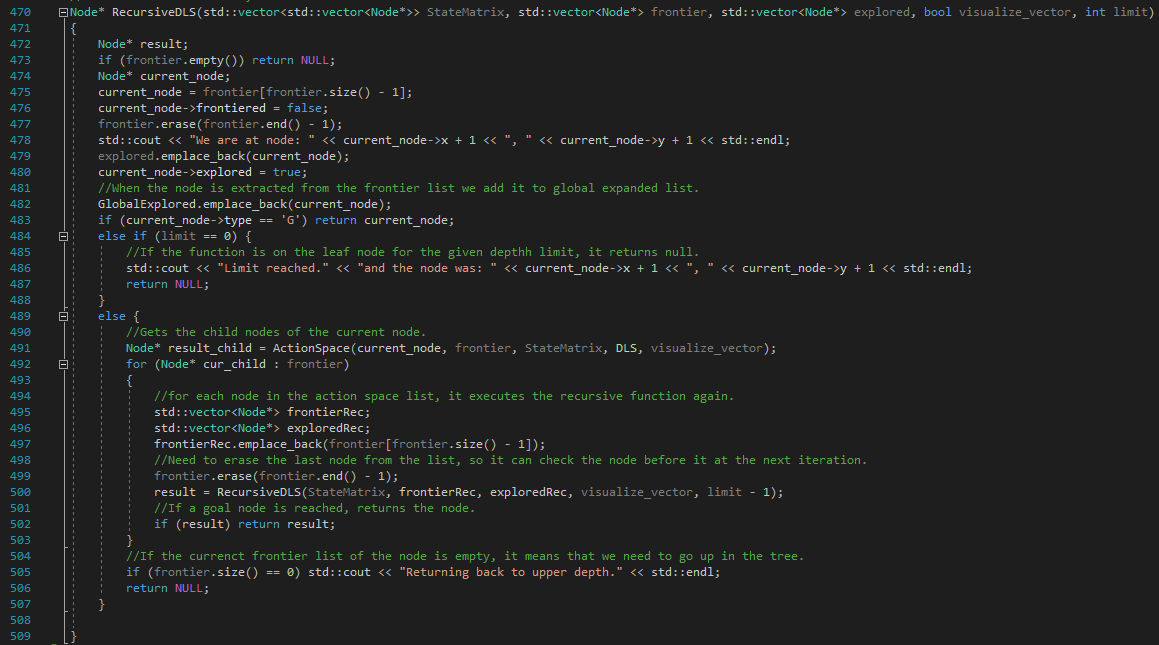
In ExecuteIDS(), depth gets instantiated as 0.

Then in infinite loop, limit gets increased and recursive function gets executed.

If the goal state is reached, explored list gets printed out and function returns success.

If the goal state is not reached, then every nodes explored Boolean is set false, so program can work correctly. Also explored list is cleared.





If frontier is empty, it means either failure or program needs to go to upper depths.

Node gets extracted from frontier list and gets added to explored.

If the nodes limit is 0, the program can’t go to lower depths and cuts off.

If the node still has limit, it gets fed to ActionSpace() function and gets its frontier list. This time, goal state check is made in this function only.

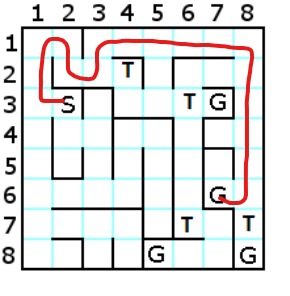
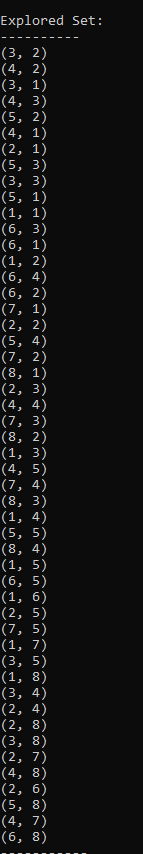
New expanded and frontier lists are created for the current node.

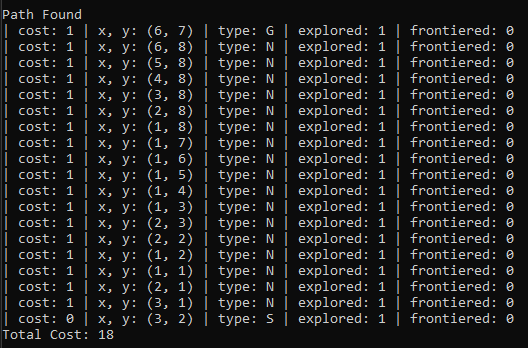
Then for each child, node gets added to new frontier list (acts like a start node itself)

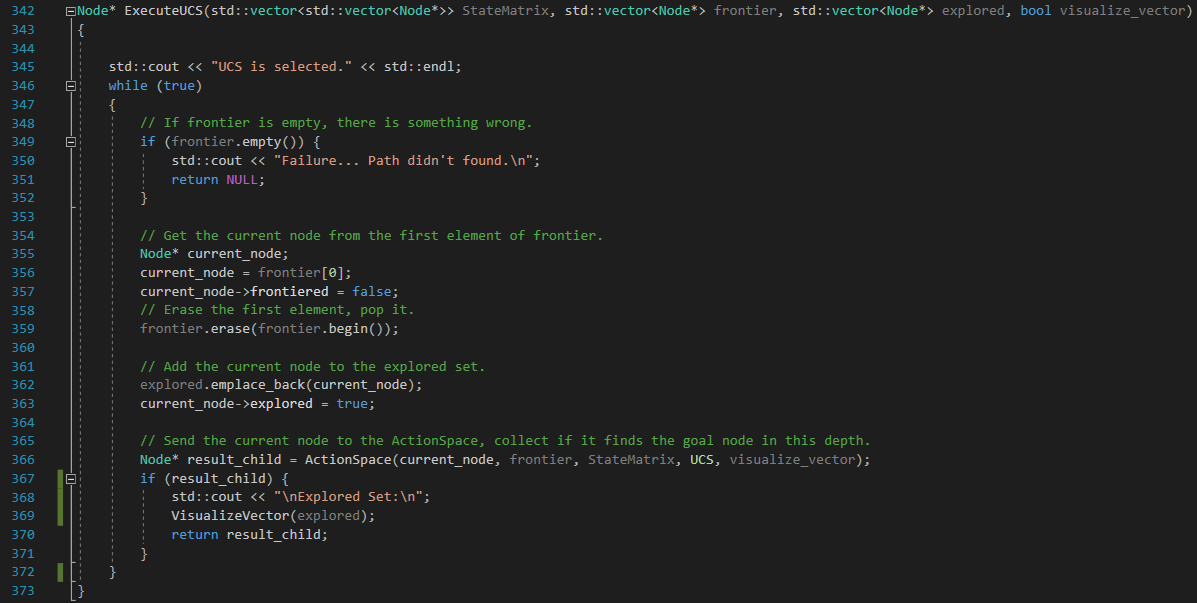
When the original frontier list is empty this loop ends, if a goal state is reached, it returns the node.

If frontier size is 0, then program goes back to the upper depth.

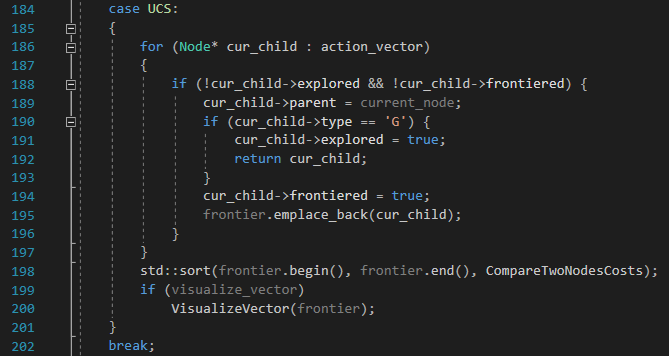
1. **Uniform Cost Search**

****

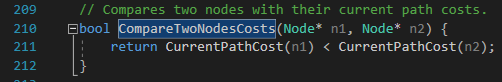
****

****

UCS works like BFS except the frontier gets sorted each iteration in ActionSpace().

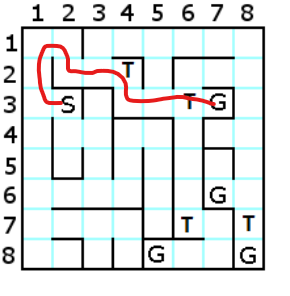
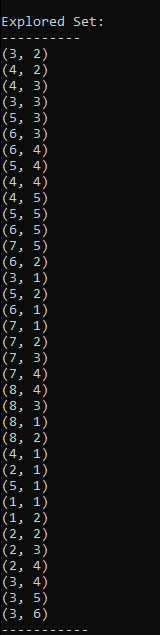
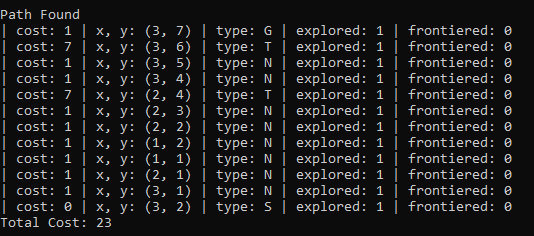
****

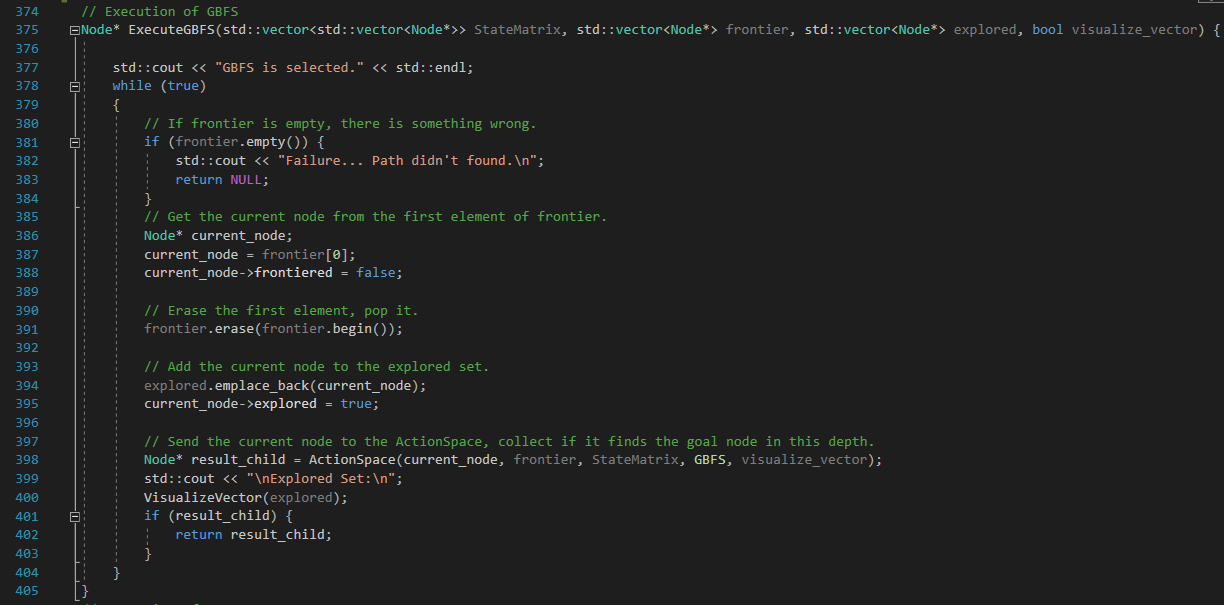
After, addition of new nodes, list gets sorted.

****

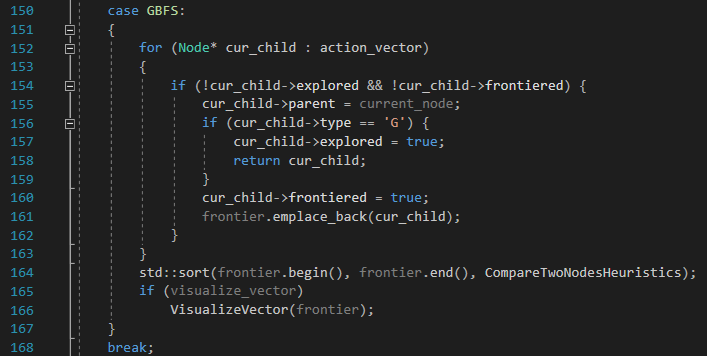
This comparison method is simply binary sort.

1. **Greedy Best First Search**

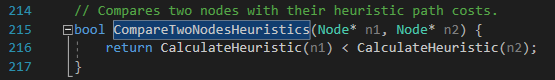




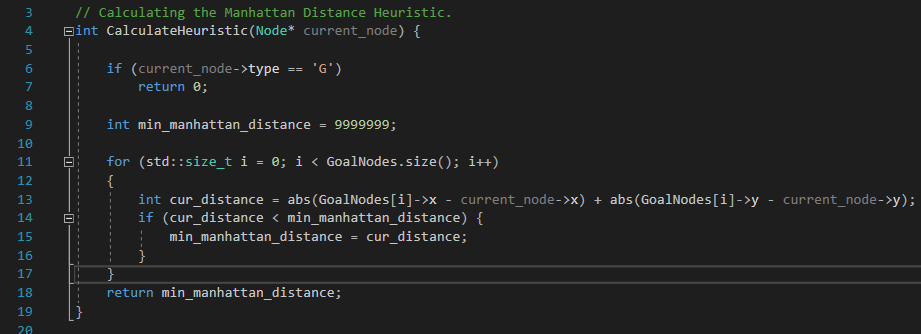
ExecuteGBFS() does the same thing with BFS in its main function.



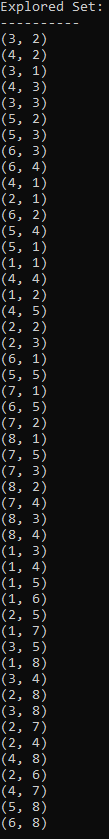
In ActionSpace(), it sorts the list according to their heuristic functions result.

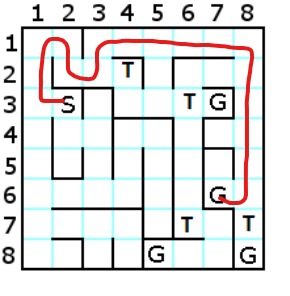


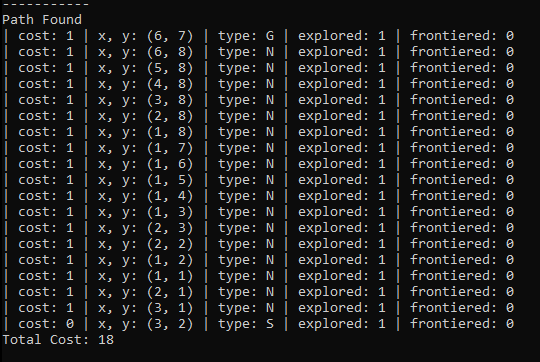
This function simply returns the result of Manhattan distance function comparison.

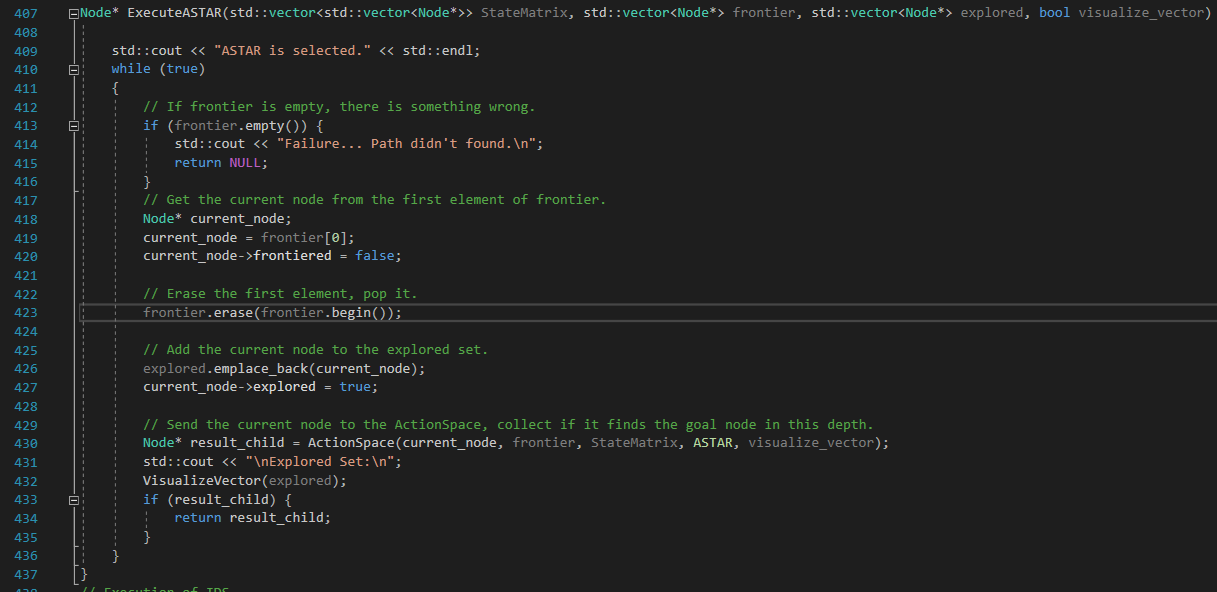


This function looks for the lowest Manhattan distance to the current cell the node’s in.

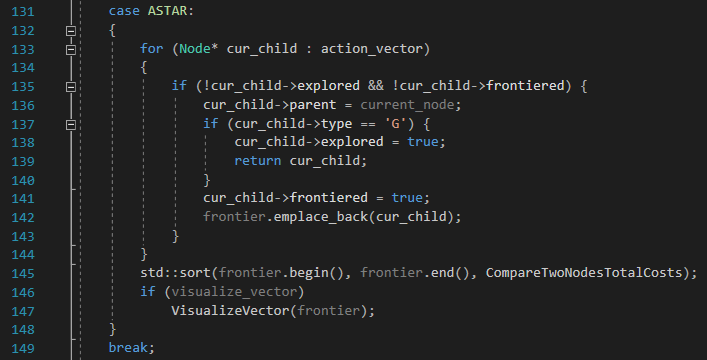
1. **A\* Heuristic Search**

****

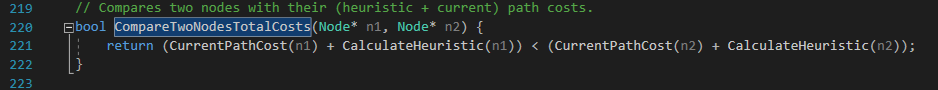




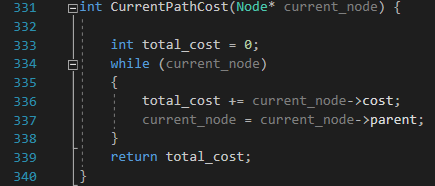
Same with GBFS.



In ActionSpace(), after the new children are added to the frontier, the total cost is compared this time.



This time, current cost is added to the heuristics.

Current cost is the cost of the path taken so far.