AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
i-CREDIT HOURS ENGINEERING PROGRAMS
Computer Engineering and Software Systems Program

Spring 2021 **Course Code:** ***CSE 472***

Artificial Intelligence

Course Project

Submitted by

	Name	**ID**
1	Abdel-Rahman Ibrahim Megahed	18P7423
2	Maryam Ahmed Galal Nouh	18P2824
3	Seif Ahmed El-Sayed Elewa	18P5662
4	Youssef Mohamed Mostafa Mansi	18P5848

Add node functionality:

The software allows the user to add a node manually by left clicking anywhere in the right window of the graph area which also prompts the user to enter the node heuristic.

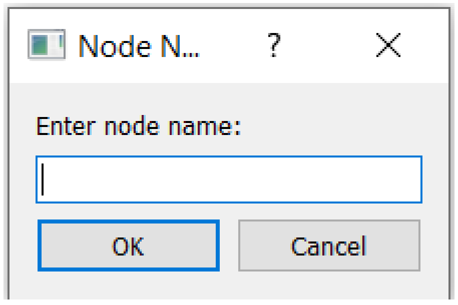


Figure 1 Node Name Insertion

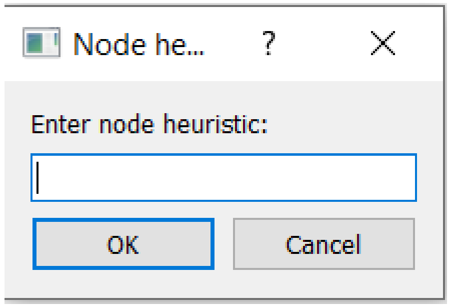


Figure 2 Heuristic of Node Insertion

Add edge functionality:

The software allows the user to add an edge by right clicking on the two nodes wanted to create an edge between and entering the weight required. Through the control panel, the user can add the node by entering the two node names and the weight of the edge.

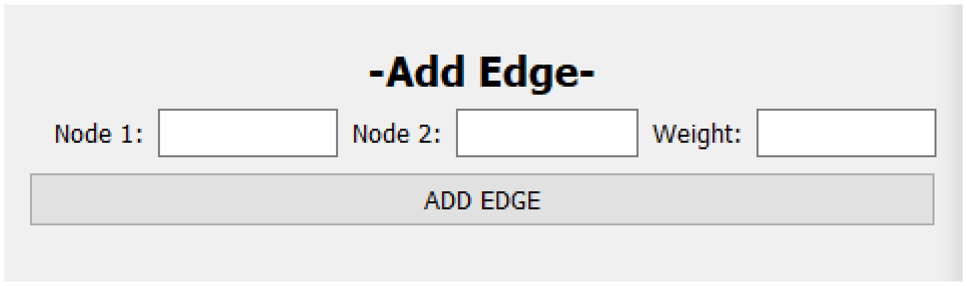


Figure 3 Add Edge (Control Panel)

Delete node functionality:

The software allows the user to remove a node from the graph area and any connected edges by selecting the node(s) by right clicking on them and hitting backspace. From the control panel, it can remove a node by entering the name which removes the node and the connected edges by pressing REMOVE NODE button.

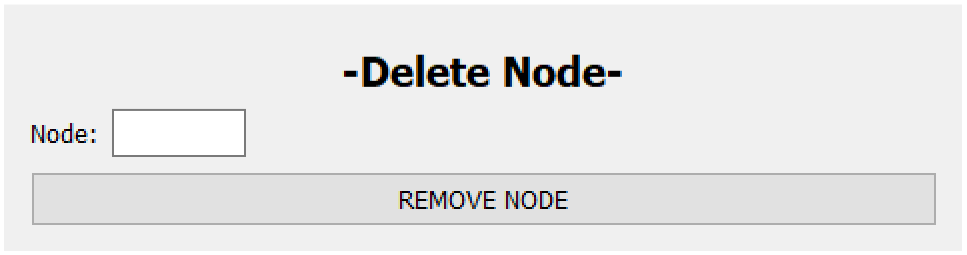


Figure 4 Delete Node (Control Panel)

Delete edge functionality:

The software allows the user to remove the edge from the graph by selecting the two connected nodes and hitting the delete button on the keyboard. From the control panel, it allows the user to enter the edge name (two connected nodes) and pressing DELETEGRAPH button.

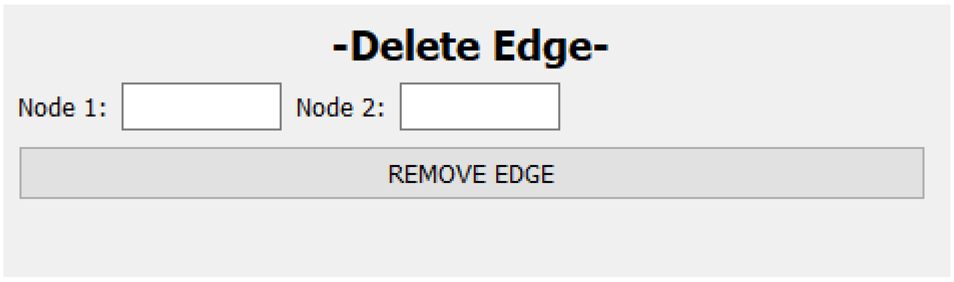


Figure 5 Delete Edge (Control Panel)

Display/Clear Path Functionality:

The software can show path by simply selecting two nodes and hitting the UP-ARROW key in the keyboard.
In the control panel, it allows the user to enter the name of the nodes to show the path between them after selecting which algorithm to use. Multiple goals can be set through the control panel by typing node names with a space between them; space works as a delimiter.
Clear path is achievable by pressing CLEAR PATH button or hitting the DOWN ARROW key in the keyboard.

Editing Graph:

To edit nodes or edges of the graph the user must clear path first.

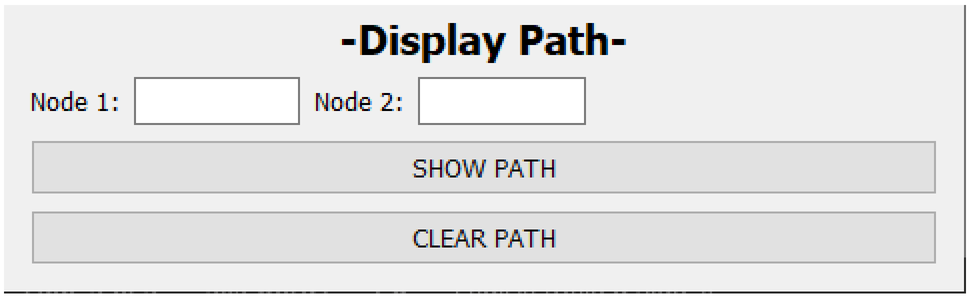


Figure 7 Display Path (Control Panel)

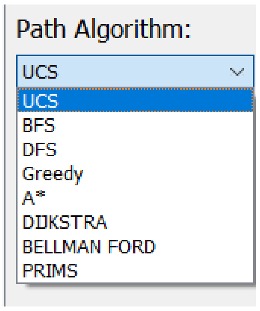
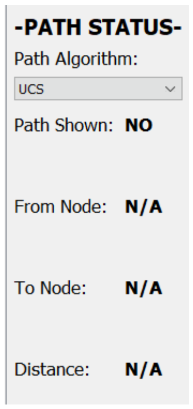


Figure 6 Algorithm Selection

Show path status functionality:

The software displays the path status through the GUI. The GUI specifies if the path is currently shown or not. It also provides the start node and a list of the goal nodes with the total distance from start node to the first goal node found.



-PATH STATUS-

Path Algorithm: UCS

Path Shown: NO

From Node: N/A

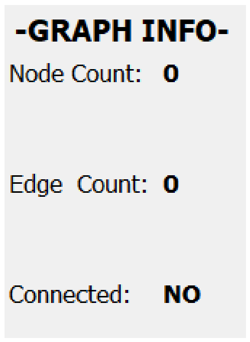
To Node: N/A

Distance: N/A

Figure 8 Path Status

Show graph info functionality:

The software shows the graph info through the GUI. It displays the node and edge count of the graph entered, while also specifying if the graph is connected or not.



-GRAPH INFO-

Node Count: 0

Edge Count: 0

Connected: NO

Figure 9 Graph Info

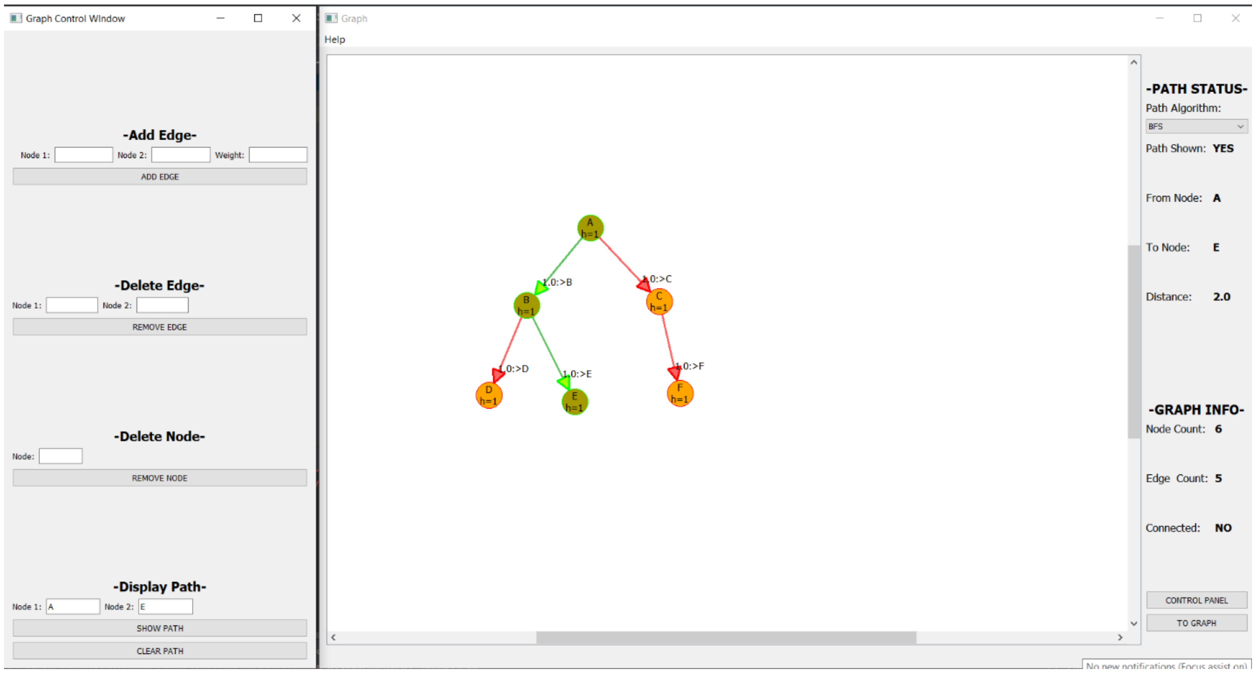
Change graph type functionality:

The software allows the swap between directed graph and normal graph through the GUI by pressing TO GRAPH/DIGRAPH button in the right bottom corner of the graph window.

Uninformed Search Algorithms:

Breadth-First Search:

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key', and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.



Node 1: A Node 2: E

Path Algorithm: BFS

Path Shown: YES

From Node: A

To Node: E

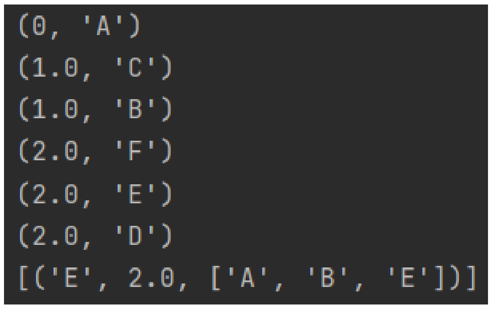
Distance: 2.0

Node Count: 6

Edge Count: 5

Connected: NO

Figure 10 BFS Result

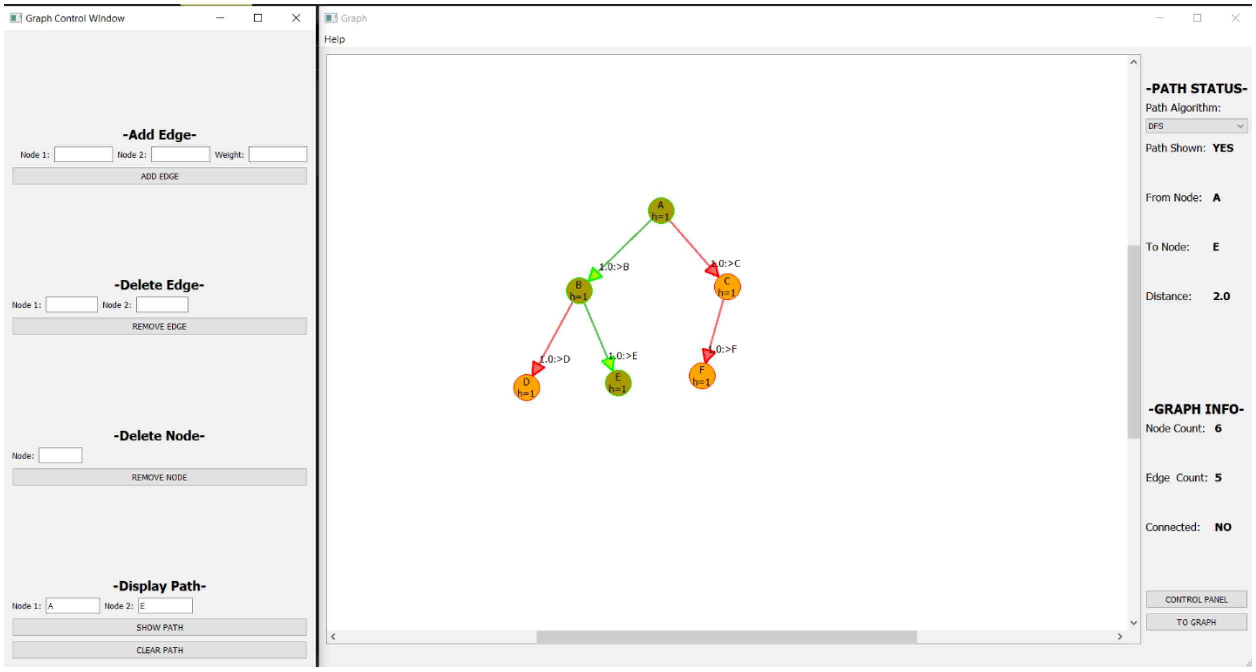


```
(0, 'A')
(1.0, 'C')
(1.0, 'B')
(2.0, 'F')
(2.0, 'E')
(2.0, 'D')
[('E', 2.0, ['A', 'B', 'E'])]
```

Figure 11 Console Representation for Visited nodes

Depth-First Search:

Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So, the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally print the nodes in the path.

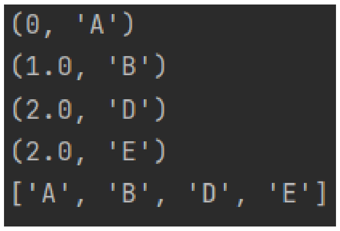


Node 1: A
Node 2: E

Path Algorithm: DFS
Path Shown: YES
From Node: A
To Node: E
Distance: 2.0

Node Count: 6
Edge Count: 5
Connected: NO

Figure 12 DFS Result

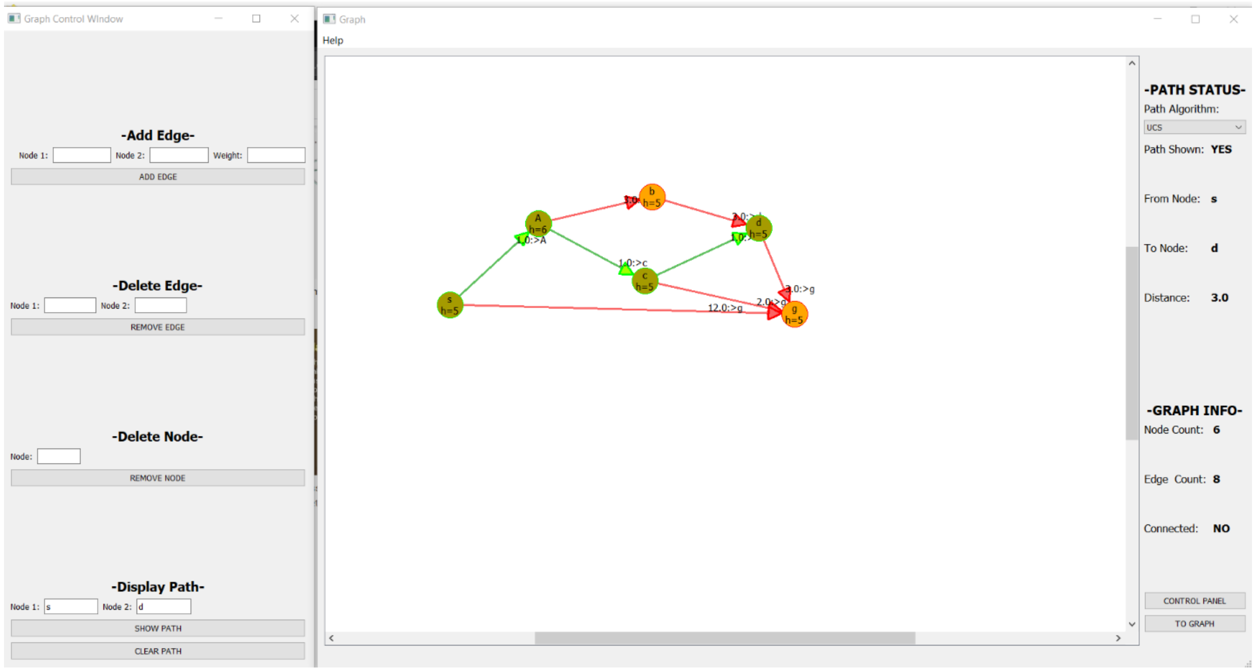


```
(0, 'A')
(1.0, 'B')
(2.0, 'D')
(2.0, 'E')
['A', 'B', 'D', 'E']
```

Figure 13 Console Representation for Visited Node

Uniform Cost Search:

Uniform-Cost Search is similar to Dijikstra’s algorithm. In this algorithm from the starting state we will visit the adjacent states and will choose the least costly state then we will choose the next least costly state from the all un-visited and adjacent states of the visited states, in this way we will try to reach the goal state (note we won’t continue the path through a goal state), even if we reach the goal state we will continue searching for other possible paths(if there are multiple goals) . We will keep a priority queue which will give the least costly next state from all the adjacent states of visited states.



Node 1: s Node 2: d

Path Algorithm: UCS

Path Shown: YES

From Node: s

To Node: d

Distance: 3.0

Node Count: 6

Edge Count: 8

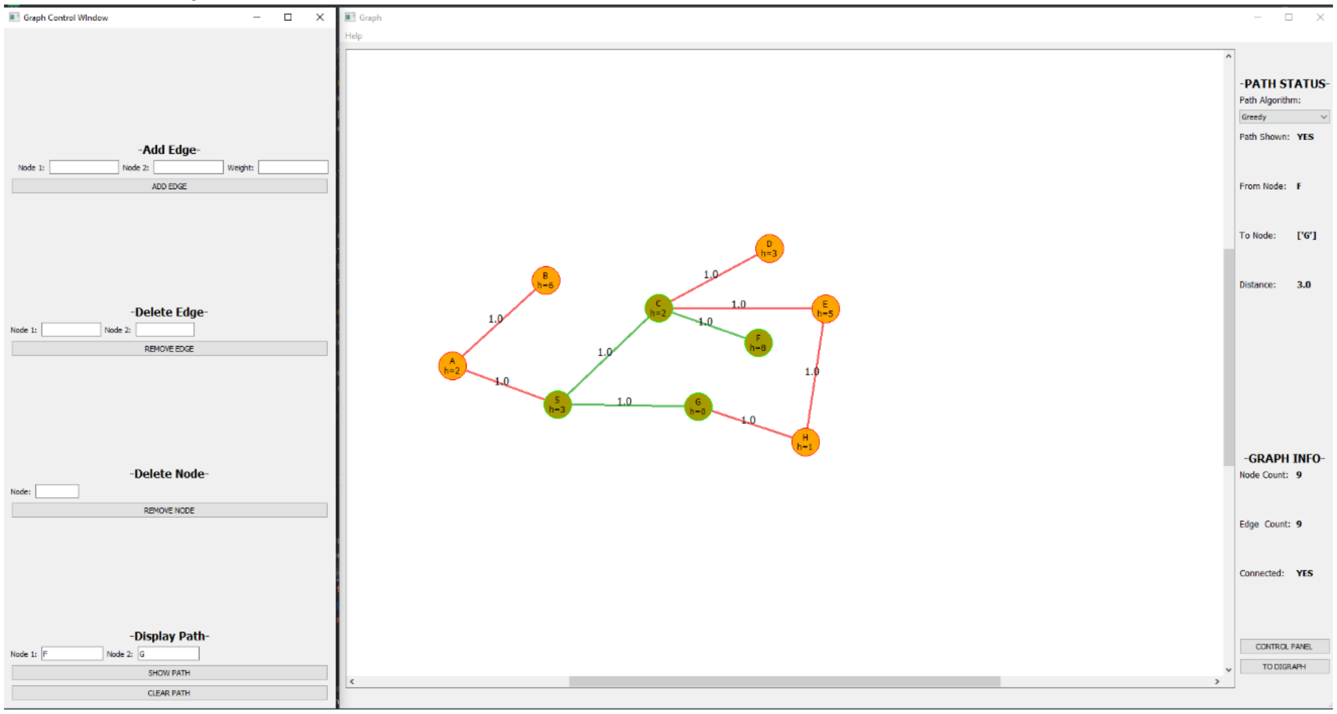
Connected: NO

Figure 14 UCS Result

Informed Search Algorithm:

Greedy Best-first Search:

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function.



Node 1: F Node 2: G

Path Algorithm: Greedy

Path Shown: YES

From Node: F

To Node: ['G']

Distance: 3.0

Node Count: 9

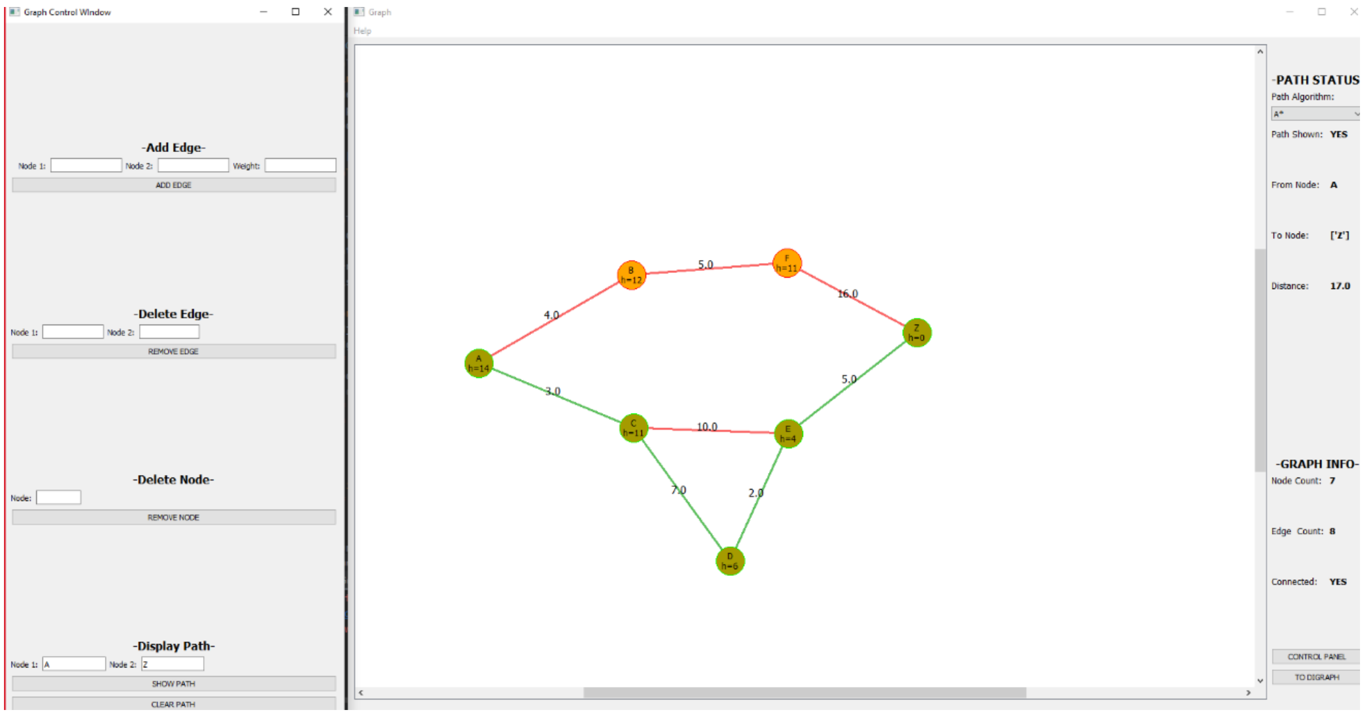
Edge Count: 9

Connected: YES

Figure 15 Greedy Result

A* Search:

A* search is the most known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is like UCS except that it uses g(n)+h(n) instead of g(n). In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence, we can combine both costs as following, and this sum is called as a **fitness number.**



-PATH STATUS-
Path Algorithm: A*
Path Shown: YES
From Node: A
To Node: ['Z']
Distance: 17.0

-GRAPH INFO-
Node Count: 7
Edge Count: 8
Connected: YES

-Display Path-
Node 1: A Node 2: Z

Figure 16 A* Result

Extra Algorithms:

Bellman Ford Algorithm:

Like Dijkstra, Bellman Ford's algorithm is used to find the single source shortest path to other nodes in the graph, however this algorithm can support negative edges in directed graphs. If a digraph contains negative edges Dijkstra's algorithm does not function properly, however Bellman Ford will continue to work as long as there are no negative weight cycles in the digraph (cycles in a graph for which the sum of the edge weights add to a negative value). To find the shortest path between two nodes using Bellman Fords, first ensure BELLMAN FORD is selected in the combo box. Select two nodes by right clicking nodes or typing node values into the text boxes below the -Display Path- label. When the UP ARROW is hit on the keyboard or the SHOW PATH button is pressed on the control panel a path with the lowest possible weight value will appear connecting the two selected nodes.

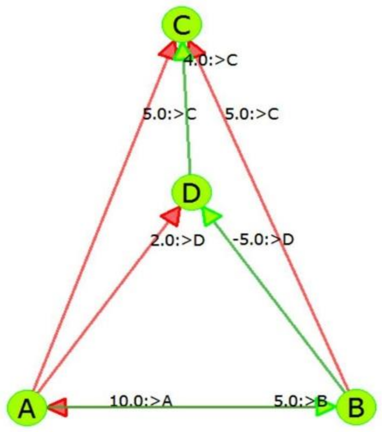


Figure 17 Bellman Ford Result

Prim's Search Algorithm:

Prim's algorithm finds the Minimum Spanning Tree of a graph or the path through the graph that touches all the nodes by traversing the edges with the lowest possible weights. To find the MST of the graph using Prim's, first ensure PRIMS is selected in the algorithm combo box and then either hit the UP ARROW on the keyboard or the SHOW PATH button on the control panel. All the nodes in the graph will then be highlighted and the least costly edges connecting those nodes will also be highlighted.

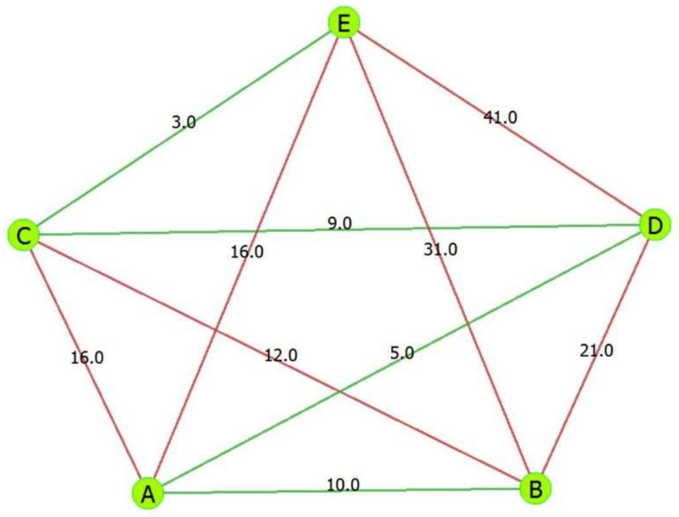


Figure 18 Prim's Result

Dijkstra’s Algorithm:

Dijkstra’s algorithm can be used to find the single source shortest path from a single node to all other nodes in the graph that can be reached from the selected node. Dijkstra is implemented in this program to only find the shortest path between two selected nodes. To find the shortest path between two nodes using Dijkstra, first ensure DIJKSTRA is selected in the combo box. Select two nodes by right clicking nodes or typing node values into the text boxes below the -Display Path- label. When the UP ARROW is hit on the keyboard or the SHOW PATH button is pressed on the control panel a path with the lowest possible weight value will appear connecting the two selected nodes.

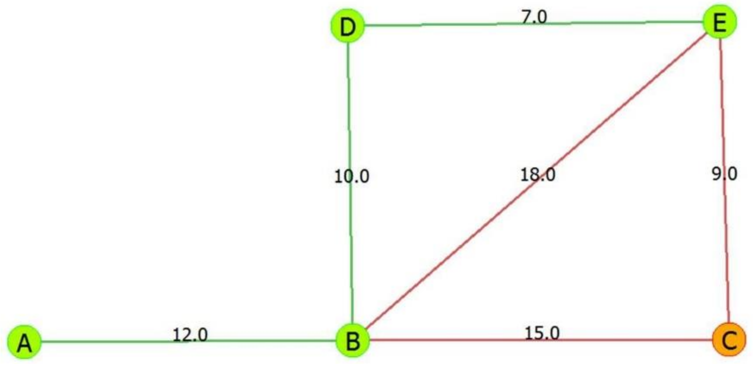


Figure 19 Dijkstra's Result