الغلاف الخارجي للبحث

A close up of a logo

Description automatically generated

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | |
|  |  | | |  | | |  | | |
|  |  | | | | | | | | |
|  |  | | | | | | | | |
|  |  | | | | | | | | |
|  | | | | | | | | | |
|  |  | | | | | | | | |
|  |  | | | | |  | | | |
|  |  | | | | | | | | |
|  |  |  | | | |  | | |  |
|  |  | | | |  | | |  |
|  |  | | | |  | | |  |
|  |  | | | |  | | |  |
|  |  | | | |  | | |  |
|  |  | | | |  | | |  |
|  |  | | | | | | | | |
|  | | | | | | | | | |
|  | | |  | |  | | | | |
|  |  | | | | | | |  | |
|  |  | | | | | |  | |
|  |  | | | | | |  | |
|  |  | | | | | |  | |

|  |  |
| --- | --- |
| **فى حالة عدم قبول البحث يرجى ذكر الأسباب** | * **..............................................................................................................................................** * **..............................................................................................................................................** * **..............................................................................................................................................** * **.............................................................................................................................................** |

**Table of contents**

1. **Part 1: Problem statement** 3
   1. My project 3
   2. Introduction & the algorithm 3
   3. Problem definition 4
2. **Part 2: Design 4**
   1. The low-level design 5
   2. Pseudocode 4
3. **Part 3: Analysis 7**
   1. Time complexity 7
4. **Part 4: System implementation and results 8**
   1. The high-level design (Block Diagram& flow chart) 8
   2. System results and its performance 10
   3. System Snapshots from Outputs 11
5. **Part 5: Appendix 9**
   1. Task code 13
   2. Video or audio 15

**Wormhole (Number 9)**

1. **Part 1: Problem statement**
   1. **My Project:**

Wormhole (Number 9)

* 1. **Introduction & Algorithm**

**Introduction**

Our problem is:

We have a group of planets, among the planets there are worm holes, These wormholes represent the path (edge) between the planets, and this means that in the presence of a wormhole between two planets, this means that there is a path between them, and in the absence of a wormhole, this means that there is no path between them.

**What is needed is the calculation of the shortest path between two planets that the user will define.**

And we have to assume that the user ship has a malfunction as it will be a one-time mistake and will take a wrong way and **we must fix this error.**

**The used Algorithm**

* We will use **Dijkstra Algorithm**, and we will add some modification to it.**In my problem, the purpose of Dijkstra's algorithm is to determine the shortest paths between two given nodes in a weighted undirected graph.**
  + Dijkstra algorithm is also called single source shortest path algorithm. It is based on greedy technique. The algorithm maintains a list visited[ ] of vertices, whose shortest distance from the source is already known.

* + If visited[i], equals 1, then the shortest distance of vertex i is already known.
  + shortest distance of each vertex is stored in an array distance[ ].
* We will create a function called **Malfunction**; we will use it **to check if the vehicle makes mistake.**
  1. **Problem definition**

**What is needed is the calculation of the shortest path between two planets that the user will define.**

And we have to assume that the user ship has a malfunction as it will be a one-time mistake and will take a wrong way and **we must fix this error.**

1. **Part 2: Design**
   1. **Pseudo code**
2. **Function ShortestPath (weightedGraph, num\_of\_nodes, start\_node, end\_node)**
3. **Input:** weightedGraph, num\_of\_nodes, start\_node and end\_node
4. **Output:** distance from start node and end node and shortest path
5. D[ ] //the distance from start node and end node
6. Pred[ ] //array of previous nodes in shortest path
7. Cost[ ][ ] //stores the nonnegative edge weights
8. Visited[ ] //array of visited planets
9. Mini\_D[ ] //minimum distance
10. Next\_node //gives the node of minimum distance
11. **//Initialization**
12. Initialize D[start\_node] 🡨 0
13. For i🡨0 to num\_of\_nodes do
14. For j🡨0 to num\_of\_nodes do
15. If weightedGraph[i][j] 🡨0 then cost[i][j] 🡨infinity
16. else cost[i][j] 🡨 weightedGraph[i][j]
17. End if
18. End for
19. End for
20. For each node n in weightedGraph do
21. D[n]🡨 weightedGraph[start\_node][n]
22. Pred[n]🡨start\_node
23. Visited[n]🡨0
24. End for
25. D[start\_node]🡨0
26. Visited[start\_node]🡨1
27. While (not all planets visited)
28. Mini\_D 🡨infinty
29. For each node n in weightedGraph do
30. If D[n] < Mini\_D then Mini\_D=D[n]
31. Next\_node=n
32. End if
33. End for
34. Visited[next\_node]🡨1
35. **//Relaxation**
36. For each n in neighbors of next\_node do
37. If D[n]> Mini\_D+cost[start\_node][n] then
38. D[n]🡨 Mini\_D+cost[start\_node][n]
39. Pred[n]= next\_node
40. End if
41. End for
42. End while
43. **//Display the outputs**
44. Print🡨 D[end\_node]
45. j🡨end node
46. Do
47. j🡨 Pred[j]
48. Print 🡨j
49. While j != start\_node
50. **End function**
51. Ask user for any malfunction, in case of any malfunction :
52. Read the current\_location from user and Call ShortestPath function with current\_location instead of Start\_node

**2.2 The low level design**

**This algorithm:**

* Named ShortestPath
* weightedGraph, num\_of\_nodes, start\_node & end\_node are procedure parameters and inputs from user.
* D[ ], Pred[ ], Cost[], Visited[ ], Mini\_D[ ] & Next-node are local variables.

**Initialization**

* We initialize cost[ ][ ] matrix from weightedGraph[ ][ ] matrix (values in cost[ ][ ] matrix will be
  + Infinity in case of the values zero in weightedGraph[][] matrix
  + The same values in weightedGraph[ ][ ] otherwise.
* We initialize Distance array D[ ] by store the weight between start\_node and other nodes (from the weightedGraph[ ][ ]) and we modify the distance from start\_node to itself to be 0
* We initialize the values in the Pred[ ] array as the value of start\_node.
* We initialize the values in visited[ ] array as zero until the star\_node completely visited then we marked it as 1.

The while has 2 For loops.

**The purpose of first for loop:**

* Search for the minimum distance in D[ ] and store it in Mini\_D.
* Store the value of the node that has the minimum distance in Next\_node.

**The purpose of second for loop:**

* To do **Relaxation**, by updating the cost of all nodes connected to a start\_node.
* Relaxing an edge (start\_node, end\_node) means testing whether we can improve the shortest path to end\_node found so far by going through Start\_node, then store the previous nodes in shortest path in Pred[].

**Display the outputs**

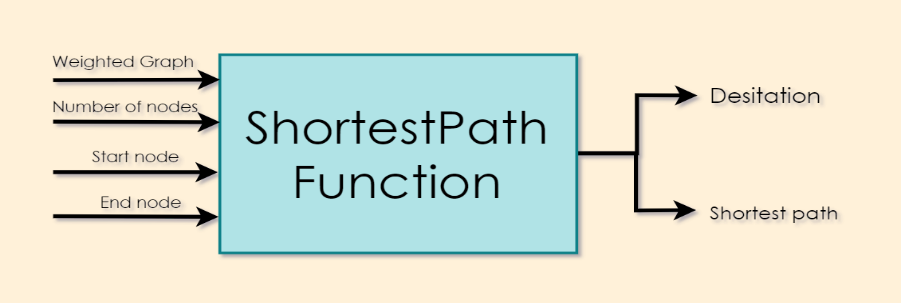
* + After storing the previous nodes in the shortest path in Pred[], we will use Pred[] to print the shortest path step by step like this 3🡨2🡨1.
  + Then we will print the destination between start\_node and end\_node.

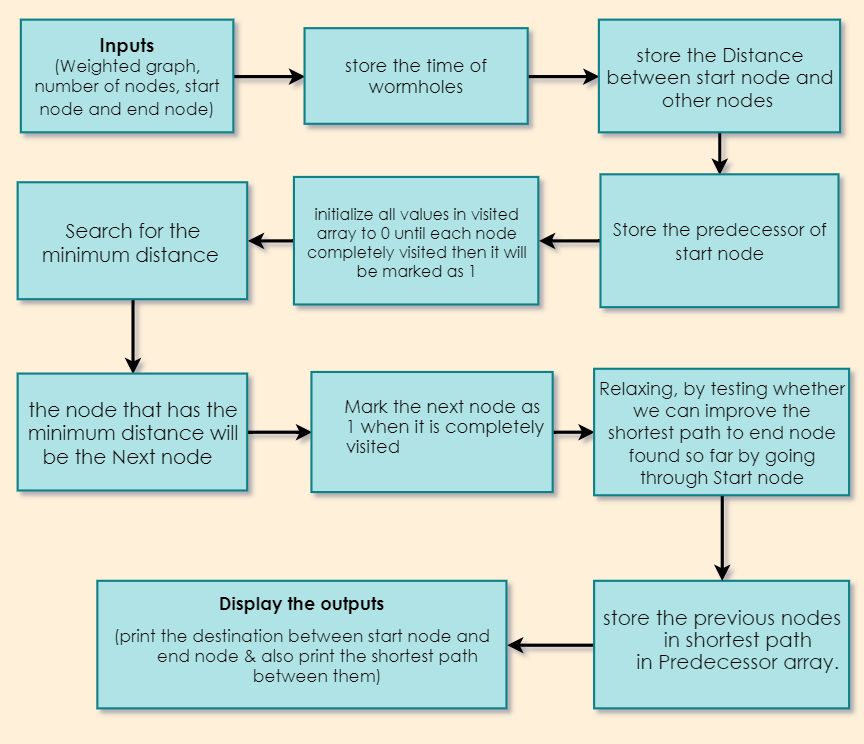
In the end, we assume that **the user’s vehicle will make a mistake and go to wrong way and doesn’t reach the end\_node,** so we will ask user about that.

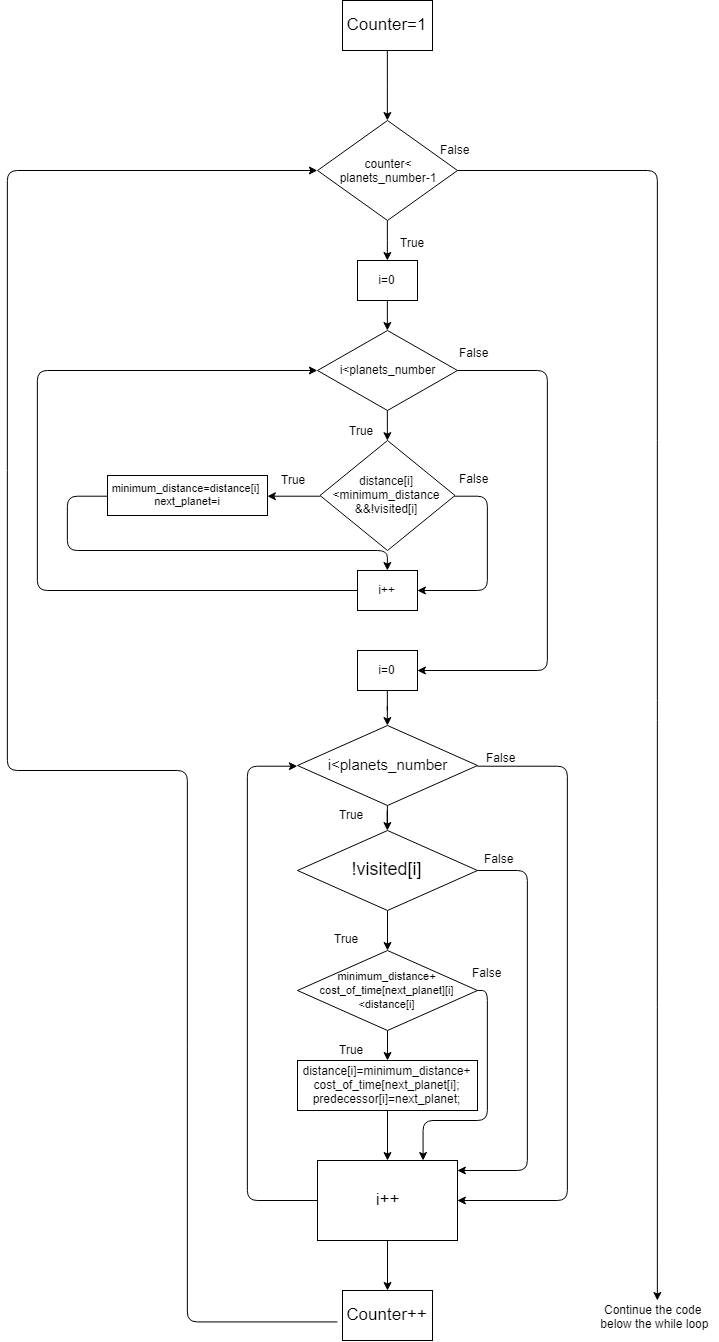
* + In case of any mistake, we will ask user about his current location then we will repeat the **shortestPath** function with current\_location instead of start\_node. (Current\_location is an input from user)
  + Otherwise, the program will be closed.

1. **Part 3: Analysis** 
   1. **Time complexity**
   * The total complexity=1+n2+n+1+1+2n2 +1+n=4+2n+3n2.
   * The Max term is 3n2
   * So, the final complexity is O(n2).
2. **Part 4: System implementation and results**
   1. **The high-level design (block diagram)**

**Simple block diagram (Shortest path function):**



**Detailed block diagram (the operations that happens in shortest path function):**

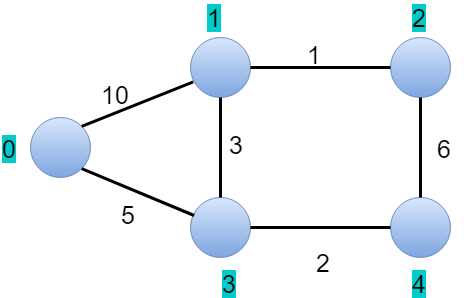
**Flowchart for the basic operation**

* 1. **System results and its performance**

|  |  |
| --- | --- |
| Performance of main function | Performance of Shortestpath function |
| The total complexity= 11+3n2.  The Max term is 3n2  So, the final complexity is O(n2). | The total complexity=  1+n2+n+1+1+2n2 +1+n=4+2n+3n2.  The Max term is 3n2  So, the final complexity is O(n2). |

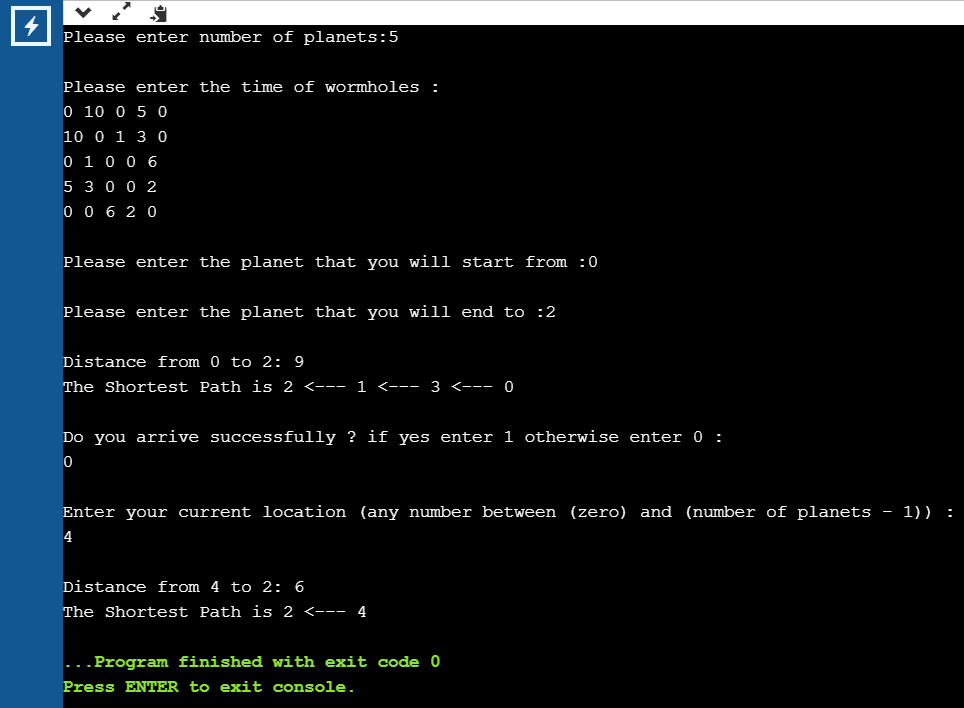
**Main function and shortestPath function are the same complexity, so the predominant is one of them.**

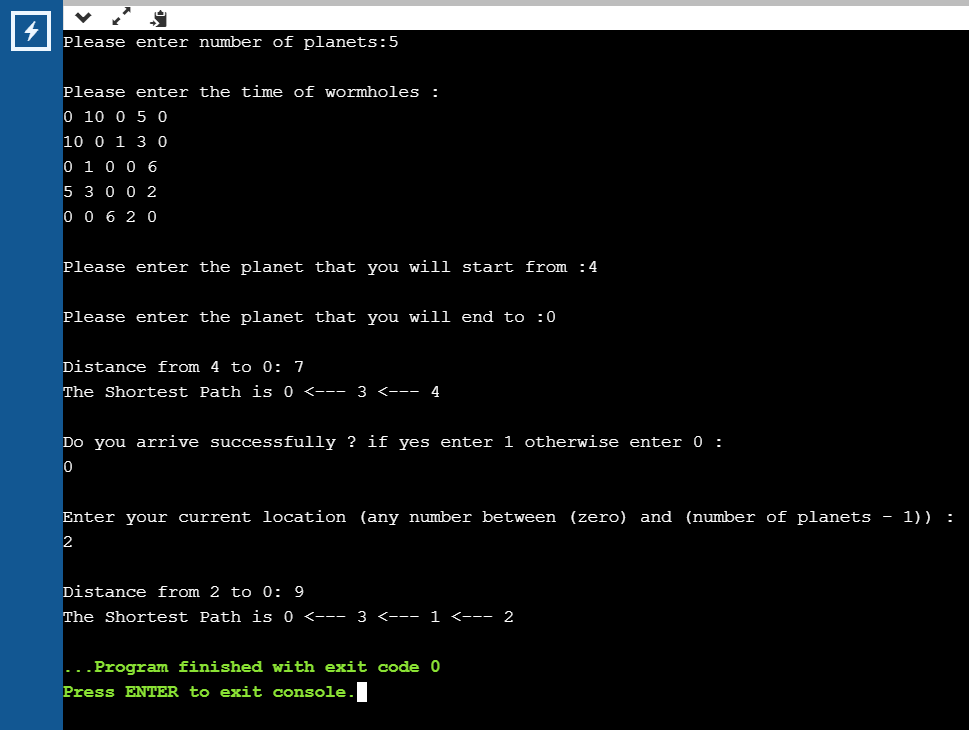
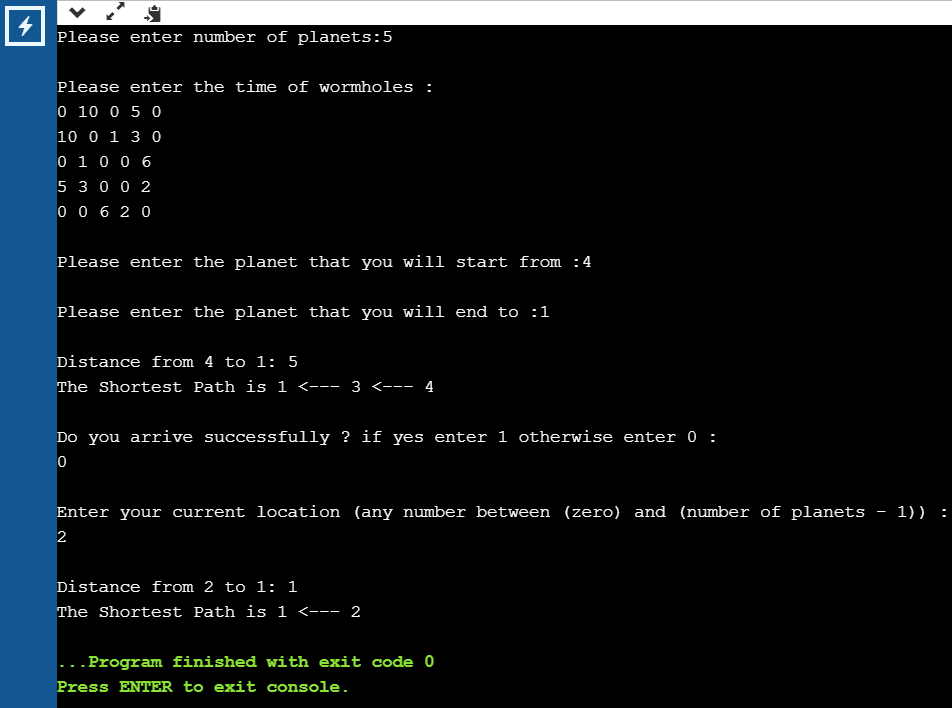
* 1. **System snapshots from outputs**

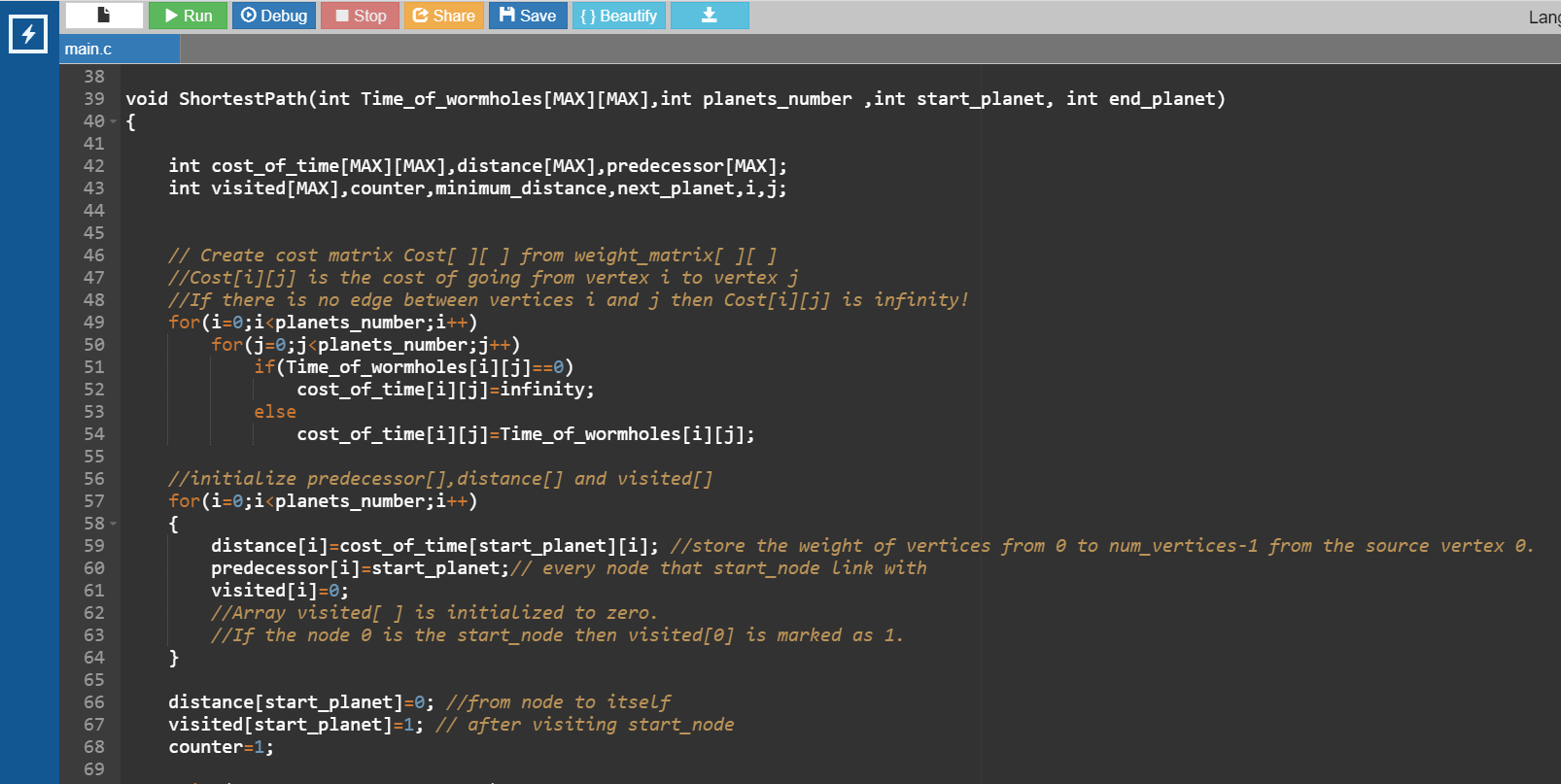
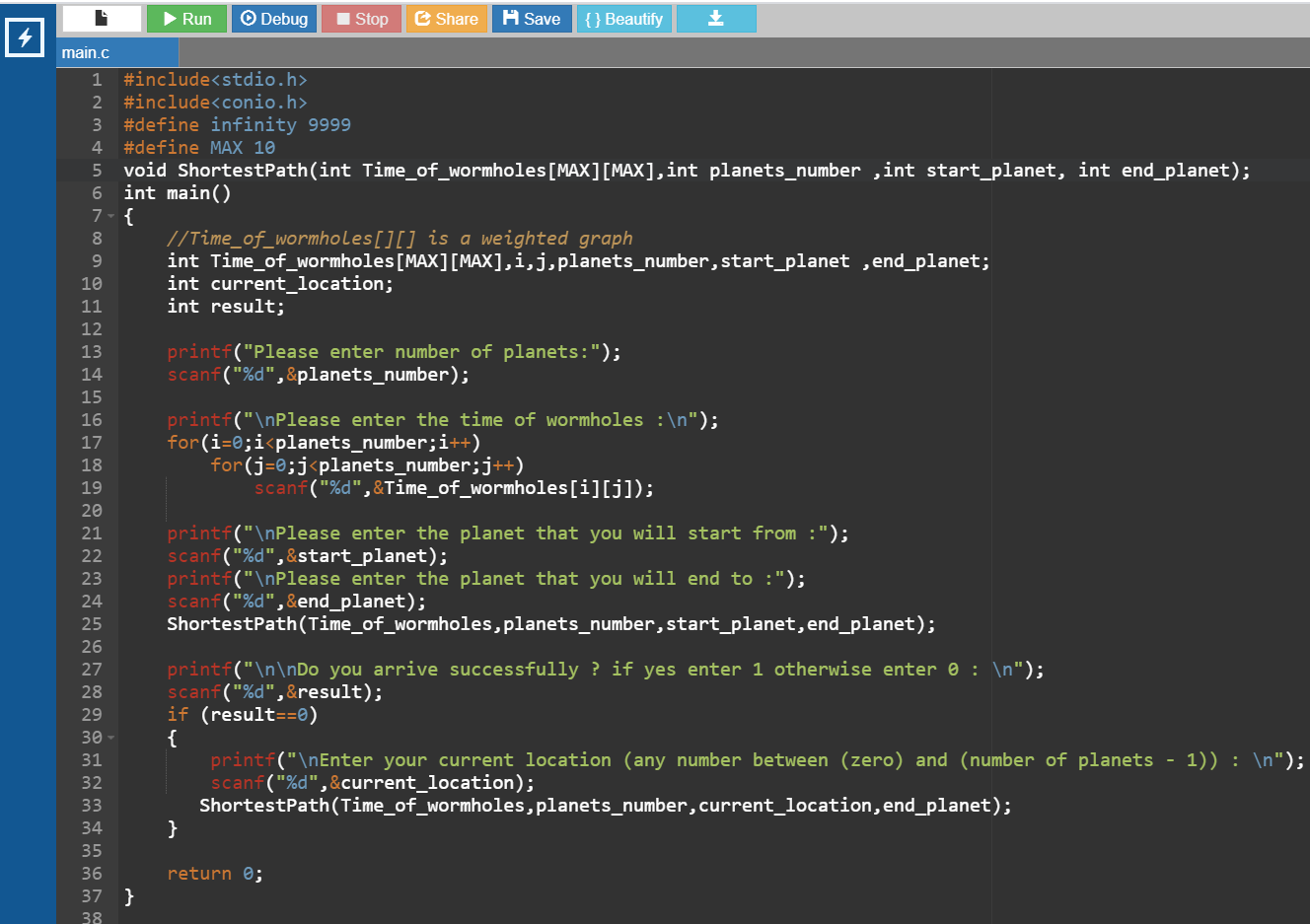


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |  |
| 0 | **0** | **10** | **0** | **5** | **0** |
| 1 | **10** | **0** | **1** | **3** | **0** |
| 2 | **0** | **1** | **0** | **0** | **6** |
| 3 | **5** | **3** | **0** | **0** | **2** |
| 4 | **0** | **0** | **6** | **2** | **0** |

**assume that is the graph and that is the adjacency matrix:**

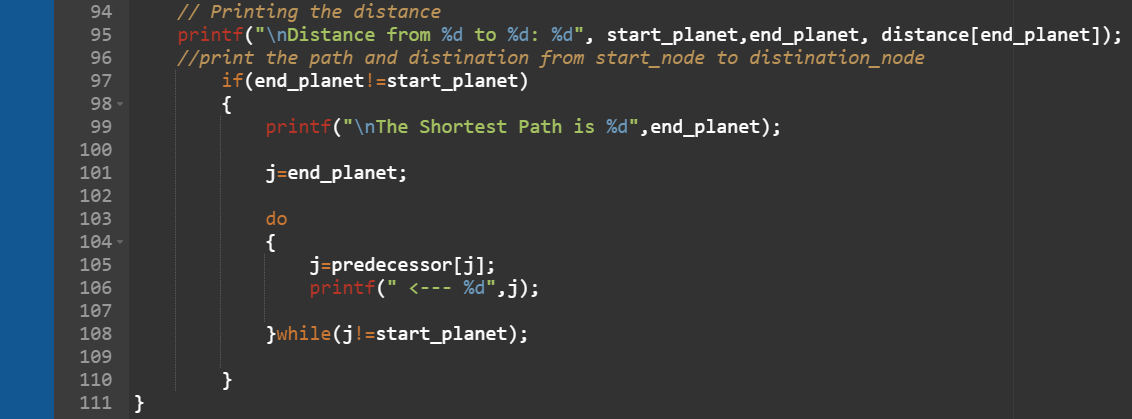
**1. so, if we start from node 0 and end to node 2 the output will be**



1. **Part 5: Appendix**
   1. **Task code**

<https://drive.google.com/file/d/1xrftm1V-ua_lDr08XKrJIwtQJjlxRrOx/view>



**5.2. The Power point link & audio link**

Power point link

<https://drive.google.com/file/d/1pi7AG8dRwqvPo5vxjFRW5uLgNH1ZJQkf/view>

audio link

<https://drive.google.com/file/d/1UiOOHxR_xbmg4ZRN3S27ioDn78QcX9HH/view>

source code

<https://drive.google.com/file/d/1xrftm1V-ua_lDr08XKrJIwtQJjlxRrOx/view>

drive link

<https://drive.google.com/drive/u/0/folders/1hQRsqNTnLCNnn-C_NaSmL6FmhoBdyVAb>