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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | | | | | | | Shortest Path Service User Guide | | | | | | |  | | |  |  | | | | Status: | Draft | | | | Version: | 0.1 | | | | Date: | **16 Nov 2018** | | | | Project ID |  | | | | | |
|  |  |  | |

Description:

Given a list of cities and the distance between each pair of cities. Problem is to find the shortest possible route that visits each city and return to the origin city.

Design:

Below algorithm can be used for this problem:

* Brute-Force
* Dijkstra
* Held karp
* Branch and Bound
* greedy algorithm

Proposed algorithm:

Brute force algorithm with dynamic programming:

Brute force:

Brute force algorithm says to compute the cost of every possible tour. This means to try all the combination of nodes in which it is traversing all the nodes. The path or route which traverse each and every city is known as Hamiltonian cycle.

Time taken: O(n!)

Dynamic programming:

Dynamic programming came into picture for improve the time complexity

Time complexity changes from O(n!) to O(n^2)(2^n)

Note: Dynamic programming is not optimal or feasible for graph which has less no. of nodes as for small value

Of n, O(n!) < O(n^2) (2^n). But is feasible for large number of nodes.

This algorithm is using the mask concept to check whether the current node is already visited or not

Ex: if node A (first node) is visited then its leftmost, ith bit will be shifted from 0000 to 0001

So when we are going to next node say ‘B’ then we will change the 0000 to 0010 and next node will become (0001 OR 0010) i.e 0011. 0011 implies that node A and B visited.

**Unit Test Cases:**

Input:

Enter the number of nodes in the graph

5

Enter the adjacency matrix

0 12 10 19 8

12 0 3 7 2

10 3 0 6 20

19 7 6 0 4

8 2 20 4 0

Enter the origin city

2

Output:

Tour: [2, 0, 4, 3, 1, 2]

Tour cost: 32.0

Input:

Enter the number of nodes in the graph

5

Enter the adjacency matrix

0 12 10 19 8

12 0 3 7 2

10 3 0 6 20

19 7 6 0 4

8 2 20 4 0

Enter the origin city

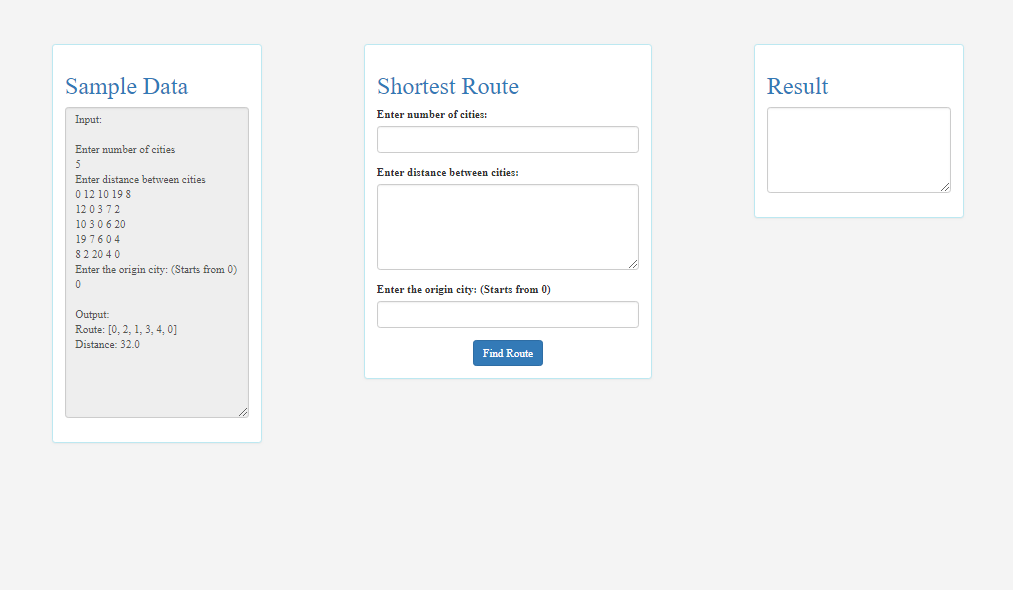
0

Output:

Tour: [0, 2, 1, 3, 4, 0]

Tour cost: 32.0

**User Interface:**



* Go to localhost:3000/
* Give input as shown in 
* Click on button
* You can see your output in 