**Optimizing real road networks by applying time efficient shortest path algorithms**

*A*

***Project Report***

*submitted in partial fulfillment of the*

*requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING**

Specialization

**Business Analytics and Optimization**

By

|  |  |
| --- | --- |
| **Student Name** | **Roll Number** |
| **Divyam Verma** | **R103218043** |
| **Devansh Messon** | **R103218037** |
| **Mayank Rastogi** | **R103218081** |

*under the guidance of*

upes-new-logo**Mr. Amit Singh**

DEPARTMENT OF INFORMATICS

SCHOOL OF COMPUTER SCIENCE

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES,

BIDHOLI, DEHRADUN, UTTRAKHAND, INDIA

upes-new-logo

**DECLARATION**

We hereby certify that the project work entitled **“Optimizing real road networks by applying time efficient shortest path algorithms”** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING specialization in Business Analytics and Optimization and submitted to the Department of Informatics, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **Aug 2020** to **Dec 2020** under the supervision of **Mr.Amit Singh.**

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Divyam Verma** | **Devansh Messon** | **Mayank Rastogi** |
|  | **R103218043** | **R103218037** | **R103218081** |

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

**Date:** \_\_\_\_\_\_\_\_\_\_\_\_\_ **Mr.Amit Singh**

Project Mentor

**Dr. Thipendra Pal Singh**

Head**,** Department of Informatics

School of Computer Science

University of Petroleum & Energy Studies

Dehradun – 248 001 (Uttarakhand)

upes-new-logo

**ACKNOWLEDGEMENT**

We wish to express our deep gratitude to our mentor(s) **Mr.Amit Singh**, for all advice, encouragement and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

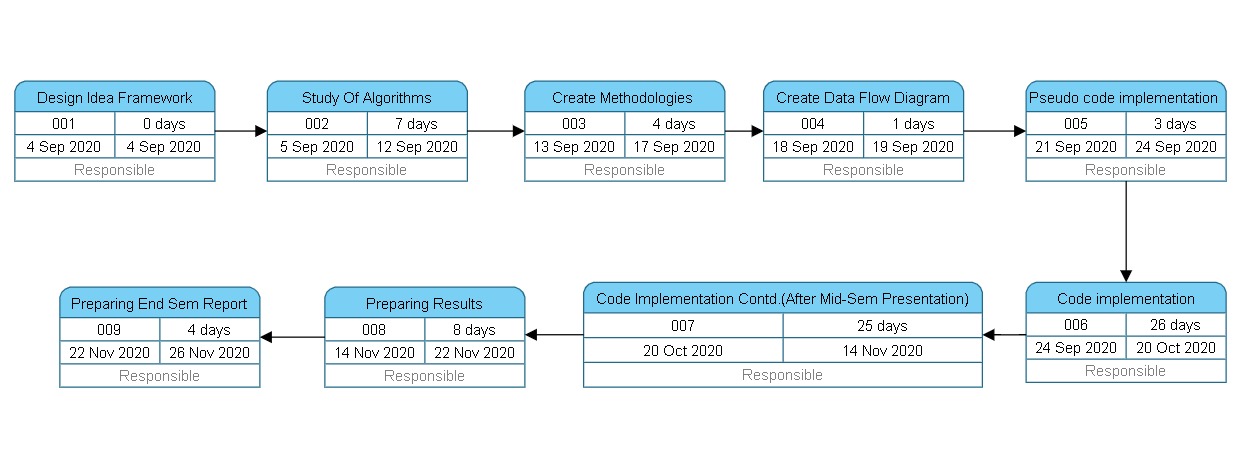
We sincerely thank to our respected Head of Department, Informatics, **Dr. Thipendra Pal Singh,** for his guidance and support as and when required.

We are also grateful to **Dr. Manish Prateek, Dean SCS**, for providing the necessary facilities to carry out our project work successfully.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our **parents** who have shown us this world and for everything they have given to us.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Divyam Verma** | **Devansh Messon** | **Mayank Rastogi** |
|  | **R103218043** | **R103218037** | **R103218081** |

**Pert Chart**

****

**Abstract**

Project’s problem description is to solve single source shortest path problem and Travelling Salesman Problem. The shortest path problem is a classical algorithm in graph theory with the intention is to locate the shortest path from a specific source to one or multiple destinations. To solve the shortest path problem 4 solving optimizations are being proposed. To solve Travelling salesman problem 3 solving optimizations are being proposed. The project’s goal is to optimize the naive algorithms in terms of space and time which will thereby improve efficiency. This project successfully achieves the goal.In Dijkstra algorithm,the proposed algorithm named as Bidirectional Dijkstra algorithm takes just **3** **milliseconds** to process graph of 1 lakh nodes and 10 lakh edges while bruteforce based solution takes **37172 milliseconds** to process the same graph.In Travelling Salesman Problem, genetic algorithm takes nearly **6 seconds** to process a graph of 13 nodes while bruteforce based solution takes nearly 1 minute to process the same graph.

**Introduction**

**Dijkstra algorithm**

Dijkstra algorithm is used to Compute the shortest route possible from a particular source location to a particular destination in least possible time by optimizing and applying time efficient shortest path algorithms. According to Dijkstra algorithm, distance cannot be negative because negative edge weights are not present in the graph.So, Dijkstra algorithm is preferred over bellman ford algorithm.Dijkstra algorithm applies a greedy method to optimally solve single source shortest path problem. The algorithm chooses the vertex with the minimum distance from source and this picked vertex relaxes its adjacent vertices and these two steps runs V times, where V is the number of vertices in the graph. The algorithm makes the optimal choice at each step to find the overall optimal way to compute the shortest route. time complexity and space complexity of naive approach of Dijkstra algorithm is O(**V2**) and O(**V2**) respectively, where V is the number of vertices in the graph. Finding the vertex with minimum distance from source is computed in O(V) time which can be optimized to achieve a better running time.

**Travelling Salesman Problem algorithm**

Travelling Salesman Problem (TSP) algorithm is a NP-hard problem which states that on a given set of cities and distance between every pair of cities compute a shortest route that visits every city exactly once and returns to the starting point.

Naive algorithm of TSP generates all (n-1)! permutations (in such way that if any vertex is already visited then that would never be visited again) of the vertices.After every permutation, the algorithm calculates the cost of the permutation and it keeps on updating the value of minimum cost. After all permutations are generated, the algorithm returns the permutation with optimal cost along with its shortest route

The time complexity of Travelling salesman problem algorithm is O(n factorial).Nature inspired technique can be used to improve the running time of the algorithm .Since this algorithm is NP-hard problem so, there is no polynomial time known solution. Naive approach cannot process graphs with large nodes.

**Related Work**

**Related work in Dijkstra algorithm**

* The shortest path problem is a classical algorithm in graph theory with the purpose is to find the shortest path from a particular source to multiple destinations.[1]
* Dijkstra algorithm applies a greedy method to optimally solve single source shortest path problem. The algorithm chooses the vertex with the minimum distance from source and this picked vertex relaxes its adjacent vertices and these two steps runs V times, where V is the number of vertices in the graph.[2]
* Bruteforce based solution of Dijkstra algorithm takes O(**V2**) time and O(**V2**) space, where V is the number of vertices in the graph.[3]
* Advanced data structures like minimum binary heap and singly linked list are used to optimize Dijkstra algorithm. Binary heap is the implementation of modern priority queue which efficiently finds and deletes the minimum element in O(log n) iterations ,where n is the number of elements in heap. Decrease key operation also takes O(log n) time.[4]
* Fibonacci heap is also used to implement priority queue which finds and deletes the minimum element in O(log n) iterations(amortized) ,where n is the number of elements in heap. Decrease key operation is done in O(1) [amortized] Although Fibonacci heaps have a reputation for being slow in practice.[5]
* Application of a smart and mathematical technique speeds up the optimized Dijkstra by 2 times. This technique runs the Dijkstra algorithm from the source vertex and destination vertex simultaneously and the algorithm terminates when a vertex is processed by forward minimum binary heap and backward minimum binary heap. This algorithm is known as Bi-directional Dijkstra algorithm.[6]
* Artificial intelligence is also used to optimize Dijkstra algorithm. A\* algorithm is an AI algorithm which optimizes Dijkstra algorithm by generating heuristics and based upon the heuristics, shortest route is decided.[7]
* Contraction hierarchies is another smart technique to optimize Dijkstra algorithm. Contraction hierarchies is a speed-up method optimized to exploit properties of graphs representing road networks. The speed-up is achieved by creating shortcuts in a preprocessing phase which are then used during a shortest-path query to skip over "unimportant" vertices.[8]

**Related work in Travelling Salesman Problem**

* Travelling Salesman Problem states that given a list of cities and the distances between each pair of cities,”Find the shortest possible route that visits each city exactly once and returns to the origin city.” It is an [NP-hard](https://en.wikipedia.org/wiki/NP-hardness) problem.[9]
* Naive approach to solve travelling salesman problem is to generate (n-1)! Permutations and calculate each permutation’s cost. Now, find the minimum cost among all costs.[10]
* Naive approach is optimized by a method called recursion with bitmasking. This method generates 2n subsets of the total number of nodes and stores it in a bitmask and then it eventually finds the shortest route by finding out the minimum cost among all costs.[10]
* Recursion with bitmasking approach is optimized by applying Genetic algorithm. Genetic algorithm works by generating random numbers in a range and thereby making a route. Many such routes are being generated and the route with the minimum cost is chosen.[11]
* Travelling Salesman Problemcan be further optimized by a nature inspired technique named Ant Colony Method.In this method, artificial ants travel from origin city and then take different orders to visit all cities and returns back to the origin city.While travelling through the cities,they release a fluid called pheromone. The path which has largest amount of pheromone is the shortest path.[12]

**Problem Statement**

In this pandemic, many people encountered various lethal health problems, therefore it becomes an emergency case and such situation expects an ambulance to reach the patient’s place as soon as possible. Obviously, ambulance must take the shortest possible path from the hospital to patient’s place in the real road network given in the form of a graph to reach the destination faster to save a life.

**Objective**

1. Suggest the ambulance a shortest possible path as soon as possible in the real road network by applying time efficient shortest path algorithms.
2. For single destination: determine the shortest possible path from source to destination.(Dijkstra algorithm)
3. For multiple destinations: determine a route such that the vehicle visits all the destinations and comes back to the source location in the least possible time.(Travelling Salesman Problem)

**Design**

**1.Methodology:**

**For Dijkstra algorithm:**

4 solving optimizations are being proposed:

* Brute force implementation
* Binary heap implementation
* Fibonacci heap implementation
* Bi-directional Dijkstra algorithm

**For solving Travelling Salesman Problem:**

3 solving optimization is being proposed:

* Backtracking/brute force implementation
* Recursion with bitmasking
* Genetic algorithm

**2.Defining the algorithms:**

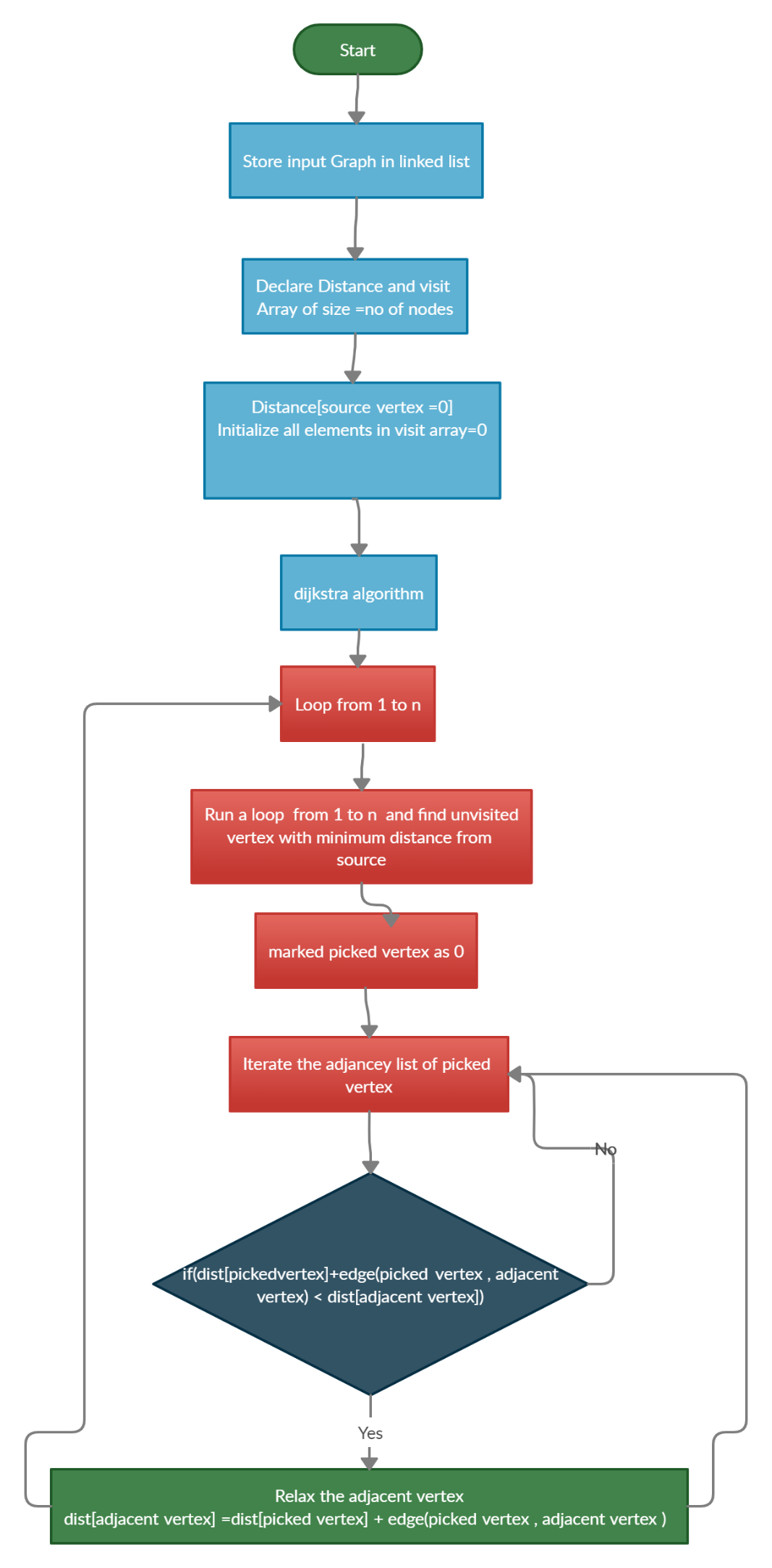
**Pseudocode and flowchart of the proposed algorithms in Dijkstra algorithm**

**1. Bruteforce-based solution**

**Pseudocode:**

1. Distance of nodes(except source node) from source node is infinity.
2. Find the unvisited node with minimum distance by running a O(n) loop.
3. Mark the picked node as visited.
4. Relax the nodes adjacent to picked node.
5. Adjacent nodes which are marked as visited cannot be relaxed further.
6. Relaxing condition:
7. if( dist[picked vertex] + edge(picked vertex , adjacent vertex ) < dist[adjacent vertex])
8. Repeat the above steps n times or break if there is no vertex found with minimum distance.

**Flowchart:**



**2.Binary Heap Data-Structure Implementation**

**Pseudocode:**

**Sub-algorithm** **Building Minimum Heap:**

1. Distance of nodes from source node is infinity except source node.
2. Put distance[source node]=0 as root of the heap.
3. Traverse the heap array from 2nd index and put distance of all other nodes.

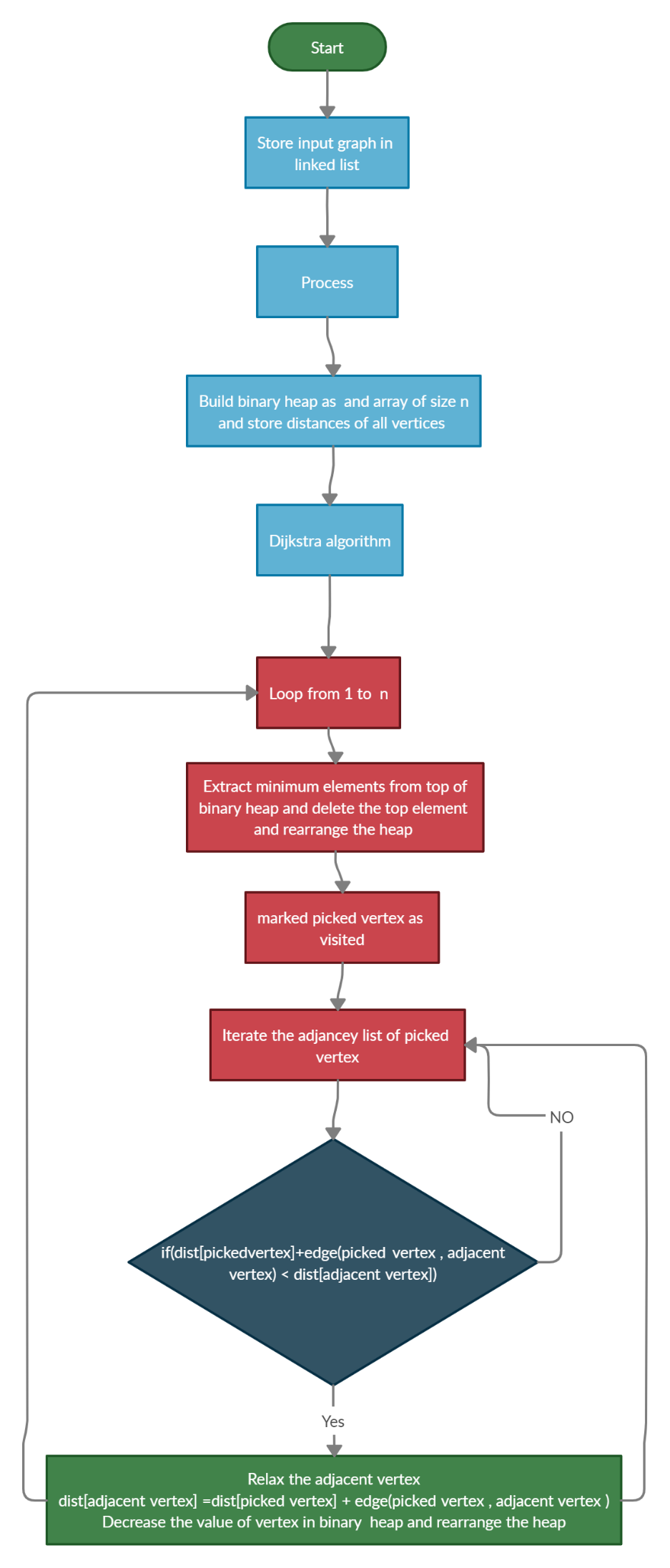
**Sub-algorithm Deleting Minimum Node from binary heap**

1. Move the last element of heap to 1st index of heap array.
2. Rearrange the heap such that min. element Is on top.
3. Start from top element:
4. Child1=2\*(parent’s index)+1
5. Child2=2\*(parent’s index)+2
6. If distances[children] < distance[parent]:
7. Take min. of distances of Child1 , Child2 and swap the minimum one with the parent node.
8. Make the minimum element as parent and Repeat the above steps.
9. Else: Do nothing

**Sub-algorithm Decrease a value of ith element in heap:**

1. Map array is used to get the position of a particular element in heap.
2. Assign value of ith element=Decreased value
3. Heap might get disturbed so rearrange it.
4. Start from the ith element and go upwards In heap.Compare it’s value with parent.
5. Parent’s index in heap=i / 2;
6. while Parent’s value>ith element’s value:
7. -swap(parent,ith element)
8. -update the map array
9. -ith element=parent(going upwards).

**Flowchart:**



**3.Fibonacci Heap Data Structure Implementation-**

**Pseudocode:**

**Sub-algorithm Deleting Minimum element from Fibonacci heap:**

1. Unlink the minimum node from the root list and add all its children to the root list.
2. Degree of node=number of children.
3. All Subtrees should have unique degree.
4. Max. Degree in the Root list= Log2 (N)
5. Declare array pointer which is pointing to node which has degree=array’s index
6. Traverse the root list of Fibonacci heap.
7. If array[current node’s degree]==NULL:
8. then array[current node’s degree] will point to current node.
9. If array[current node’s degree]!=NULL:
10. If array[current node’s degree] value > current node value
11. then make current node as parent of array[current node’s degree]
12. **//**array[current node’s degree] will point to current node
13. else vice versa
14. After every subtree has unique degree in the Fibonacci heap.
15. Traverse the array pointer and find the minimum element of the Fibonacci heap

**Sub-algorithm Building fibonacci heap:**

1. Allocate a new node.
2. Make its parent,child,left,right as NULL.
3. degree,mark=0
4. Distance of nodes from source node is infinity except source node.
5. If heapsize==0:
6. Minimum element=input
7. if heapsize>0:
8. Minimum element=min(Minimum element,input

**Sub-Algorithm CUT Function():**

1. If value of parent of ith node > value of ith node:
2. then : Move ith node to the root list and mark it 0.
   * 1. Degree of parent is reduced by 1

**Sub-Algorithm CASCADE Function():**

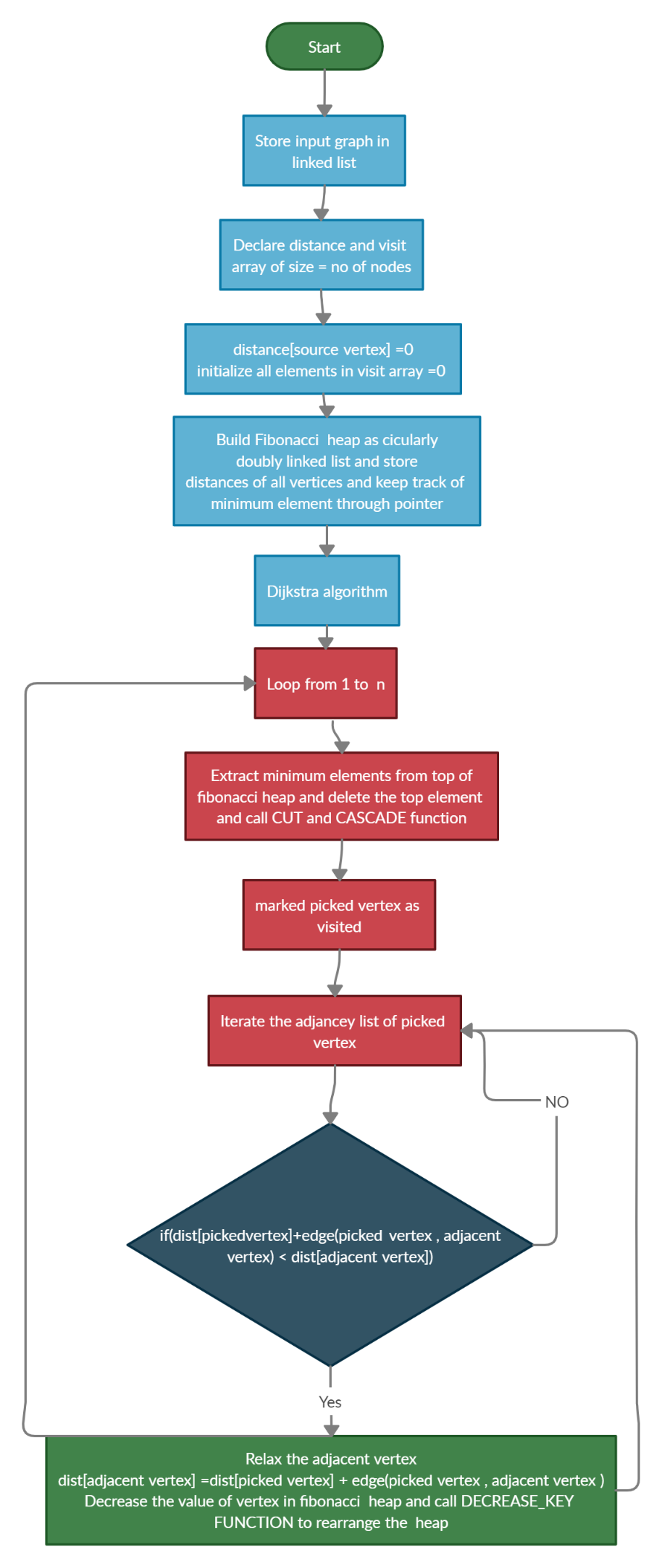
1. If parent of ith node is marked as 0:
2. then parent of ith node is marked as 1
3. else if parent of ith node is already marked as 1.
4. then: CUT()

CASCADE\_CUT()

**Algorithm Decreasing key of Fibonacci heap:**

1. Map array is used to get the position of a particular element in heap.
2. Assign: value of ith element=Decreased value.

**Flowchart:**



**4.Bi-directional Dijkstra algorithm**

**Pseudocode:**

**Optimized\_Dijkstra()**

1. input(number\_of\_nodes)
2. input(number\_of\_edges)
3. FOR LOOP till number\_of\_edges:
4. input(node\_1)
5. input(node\_2)
6. input(edge\_weight)
7. Call Build\_Graph(node\_1,node\_2,edge\_weight)
8. Call Build\_Graph(node\_2,node\_1,edge\_weight)
9. declare Forward\_distance[] and Backward\_distance[] array
10. Forward\_distance[source]=0
11. Backward\_distance[destination]=0
12. Forward\_distance[all\_nodes\_except\_source]=INFINITY
13. Backward\_distance[all\_nodes\_except\_destination]= INFINITY
14. Call Build\_Binary\_Heap(Forward,source)
15. Call Build\_Binary\_Heap(Backward,destination)
16. Call Bidirectional\_Dijkstra()

**SubAlgorithm Build\_Graph(node\_1,node\_2,edge\_weight)**

1. put node\_2 into linked\_list of node\_1
2. put node\_1 into linked\_list of node\_2

**SubAlgorithm Build\_Binary\_Heap(Binary\_Heap\_type,top)**

1. **//Binary\_Heap\_Type means Forward\_heap OR Backward\_heap**
2. declare Binary\_Heap\_type[] array
3. Top\_of\_binary\_heap=distance[top]
4. Put distance[Other\_nodes\_except\_top] below the top node element

**Algorithm Bidirectional\_Dijkstra()**

1. processed[all\_nodes]=0
2. **//processed[i] signifies that node i's distance from source node is minimizied.**
3. WHILE Forward\_Heap\_size>0 AND Backward\_Heap\_size>0 :
4. **//Forward Dijkstra Starts**
5. Dijkstra(Forward\_Dijkstra)
6. **//Check if final path is going through 'pick' node and if it is minimizing**
7. **//the final distance from source to destination**
8. if Final\_distance > Forward\_distance[pick] + Backward\_distance[pick]:
9. Final\_distance = Forward\_distance[pick] + Backward\_distance[pick]
10. **//Above line is minimizing the final distance from source to destination**
11. if processed[pick]==1:break
12. **//if pick'distance is already minimized then Bidirectional algorithm terminates.**
13. else processed[pick]=1
14. **//pick's distance from source node is minimizied.**
15. **//Backward Dijkstra Starts**
16. Dijkstra(Backward\_Dijkstra)
17. **//Check if final path is going through 'pick' node and if it is minimizing**
18. **//the final distance from source to destination**
19. if Final\_distance > Forward\_distance[pick]+ Backward\_distance[pick]:
20. Final\_distance = Forward\_distance[pick] + Backward\_distance[pick]
21. **//Above line is minimizing the final distance from source to destination**
22. if processed[pick]==1: break
23. **//if pick's distance is already minimized then Bidirectional algorithm terminates.**
24. else processed[pick]=1
25. **//pick's distance from source node is minimized.**

**SubAlgorithm Relax\_Value\_Of\_Node(adjacent\_node,Binary\_Heap\_type)**

1. child=adjacent\_node
2. parent=child/2
3. **//Min.binary heap property of the tree below the relaxed node is satisfied.**
4. **//Min.binary heap property of the tree above the relaxed node is violated.**
5. **//Go upwards the heap by swapping**
6. WHILE parent > child:
7. SWAP parent and child
8. child=parent
9. parent=child/2
10. **//Go upwards the binary heap for satisfying the min.binary heap property**

**SubAlgorithm Dijkstra(Dijkstra\_type)**

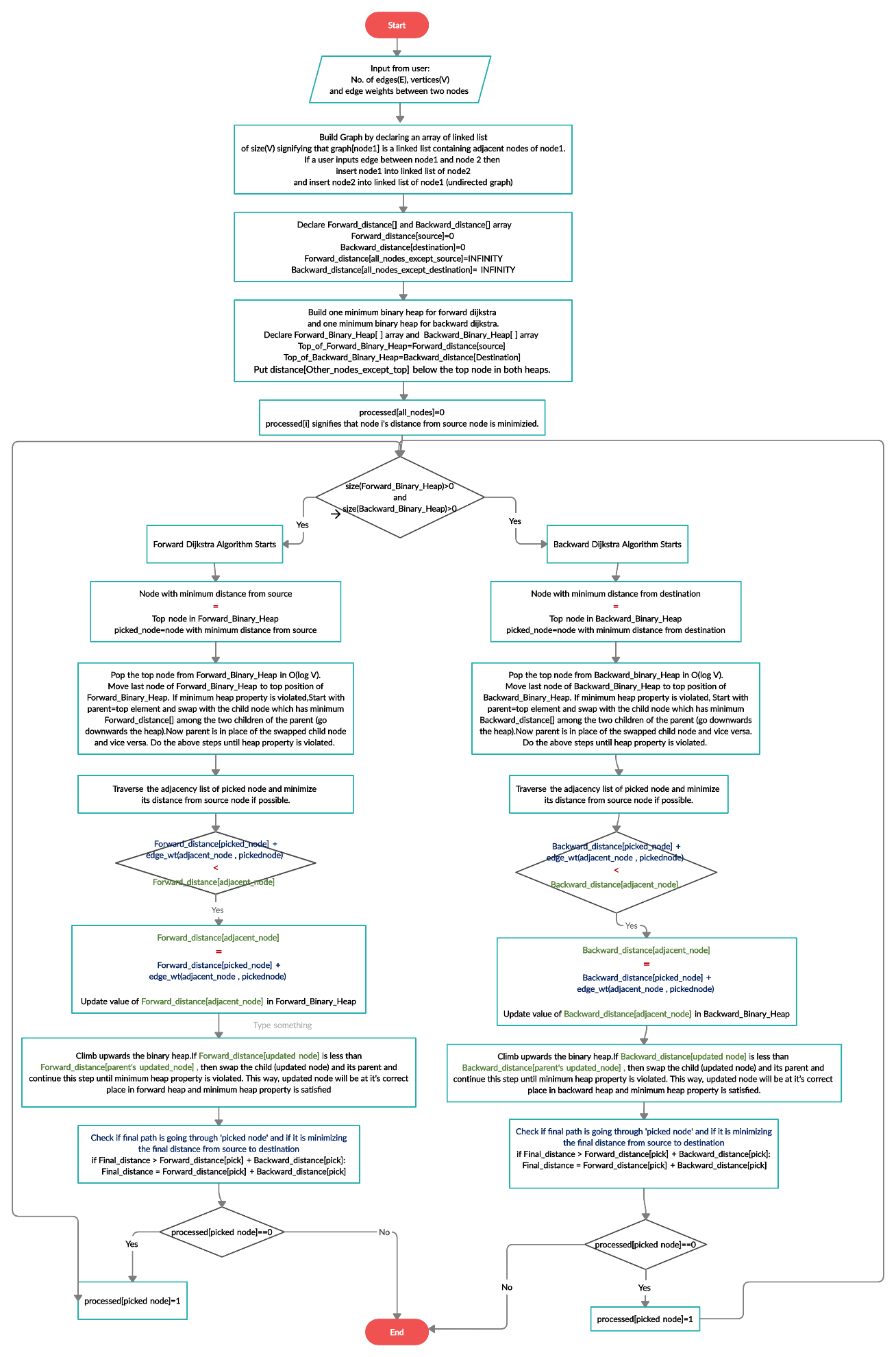
1. **//Binary\_heap\_type means Forward\_Dijkstra OR Backward\_Dijkstra**
2. int pick=top\_element\_of\_heap
3. **//pick signifies the node with minimum distance from source node.**
4. **//Delete Top element from Forward\_Binary\_Heap to access next node in next iteration**
5. Delete\_Top\_Element(Binary\_Heap\_Type) **//Binary\_Heap\_Type means Forward\_heap OR Backward\_heap**
6. **//Traverse the adacency\_list of pick to relax the adjacnent nodes of pick**
7. FOR LOOP till size\_of\_adjacencylist\_of\_pick:
8. **//Relax the adjacent node if possible**
9. if distance\_type[pick]+edge\_weight(pick,adjacent\_node) < distance\_type[adjacent]:
10. **//distance\_type means Forward\_distance OR backward\_distance**
11. **//Relax the adjacent node as condition is satisfied**
12. distance\_type[adjacent\_node]=distance\_type[pick]+edge\_weight(pick,adjacent\_node)
13. Relax\_Value\_Of\_Node(adjacent\_node,Binary\_Heap\_Type)
14. **//Reflect the changes in minimum heap**

**SubAlgorithm Delete\_Top\_Element(Binary\_Heap\_Type)**

1. **//Binary\_Heap\_Type means Forward\_heap OR Backward\_heap**
2. MOVE last\_node\_in\_heap TO Top\_position\_in\_heap
3. **//Top element deleted**
4. **//Rearrange the heap**
5. WHILE True:
6. LEFT\_CHILD=2\*(Index\_of\_parent\_in\_heap) **//left child node**
7. RIGHT\_CHILD=2\*(Index\_of\_parent\_in\_heap)+1 **//right child node**
8. **//Go downwards in heap if minimum heap property is violated**
9. if Parent > LEFT\_CHILD OR RIGHT\_CHILD:
10. SWAP Parent and Minimum(LEFT\_CHILD,RIGHT\_CHILD)
11. Parent=Minimum(LEFT\_CHILD,RIGHT\_CHILD)

12. else break

**Forward Dijkst**



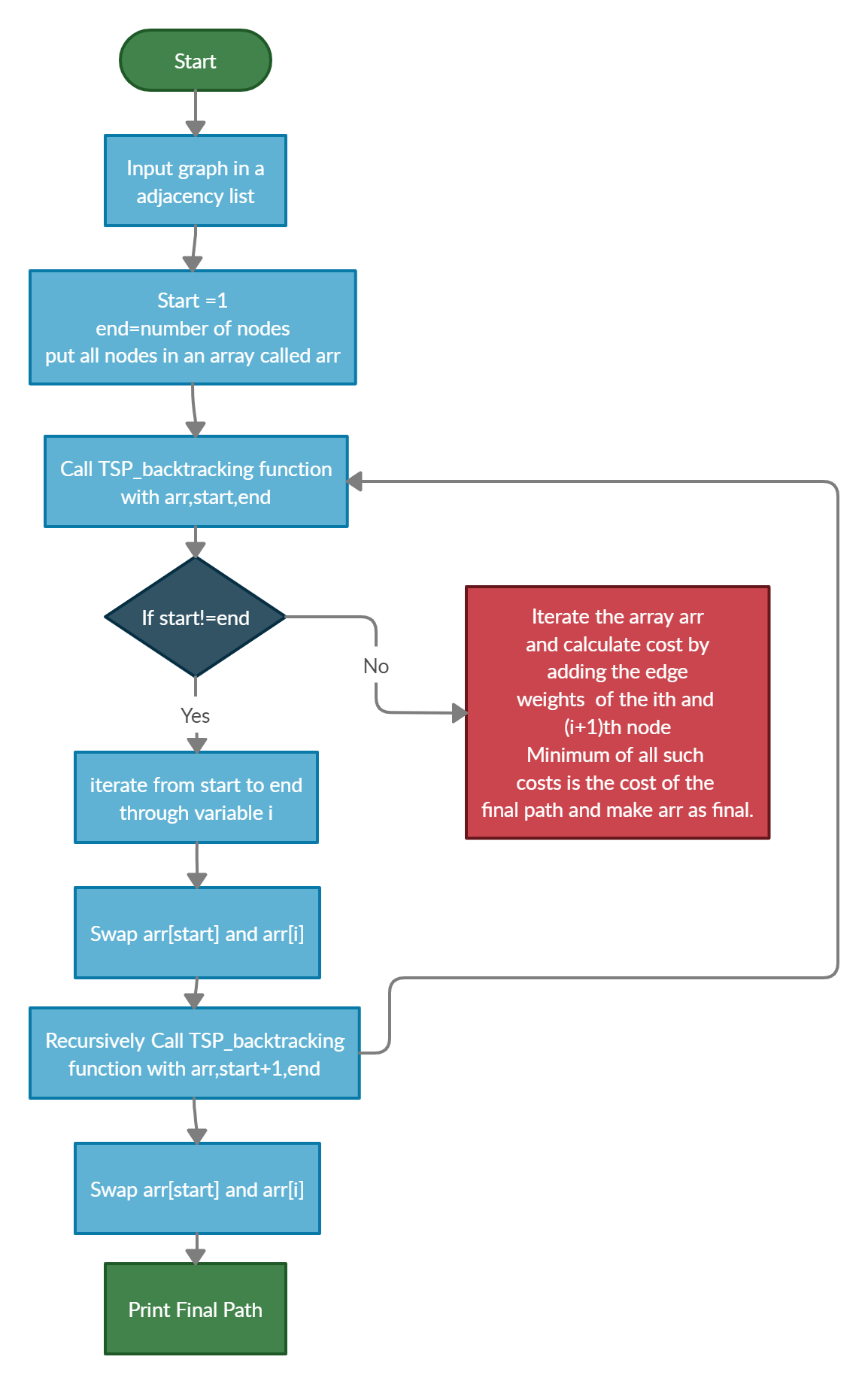
**Pseudocode and flowchart of the proposed algorithms in Travelling Salesman Problem**

**1.Backtracking/Brute Force solution**

**Pseudocode**

1. Root node or source node(1) is the starting point and ending point of the final path.
2. Generate all (n-1)! permutations of the nodes as:
3. -Traverse the array of nodes by variable i.
4. -Swap starting node and ith node.
5. -Recursively call the function with starting node=starting node+1.
6. -Swap starting node and ith node.
7. when starting node==ending node, Calculate cost of the path.
8. Return the permutation having the minimum cost.

**Flowchart:**



**2.Recursion with bitmasking method**

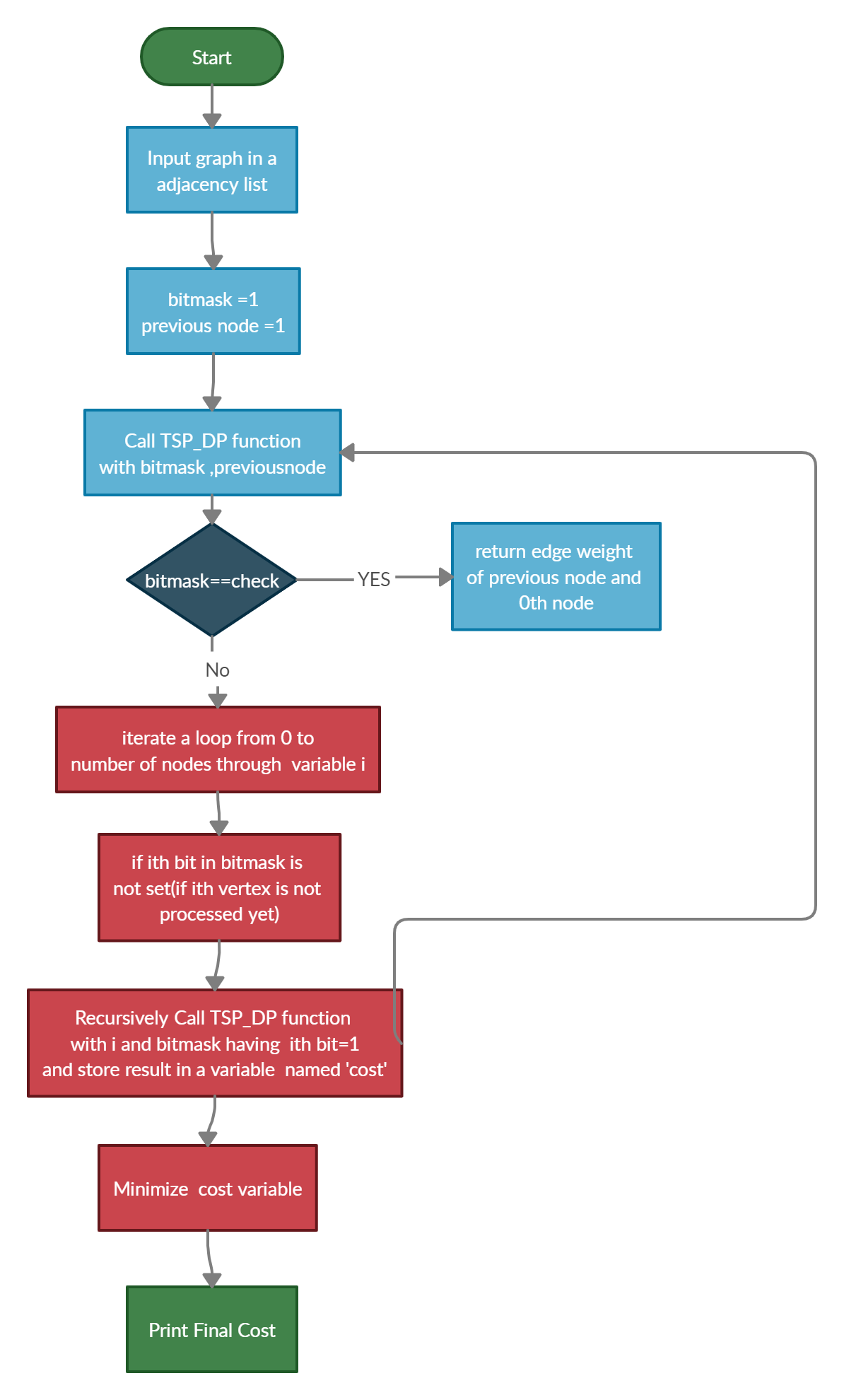
**Pseudocode:**

1. Considering Starting vertex and ending vertex as 1.
2. Find i**th** vertex which must be second last vertex in the path.
3. For point 2,Generate all 2n subsets of the nodes through bit manipulation.
4. Bit manipulation works as visited arr,bits of the bitmask will represent nodes.
5. 1 means visited and 0 means not visited.
6. Recurrence relation for point 2:

cost(S,i) = dist[i][j]= + cost( {S-i-1} , j )

where S represents set of all vertices and j belongs to S

**Flowchart:**



**3.Genetic algorithm**

**Important Terms used in Genetic Algorithm**

* A **Chromosome** represents a TSP path containing cities.
* **Fitness value** of chromosome is the cost of the whole path.
* A **Population** contains a finite number of chromosomes. This finite number is the population size.
* **Mutation** of a chromosome is a process to generate a new child chromosome by swapping two cities in the parent chromosome.
* When a mutation is performed, a new **generation** is defined.
* For example, child chromosome is 2nd generation and parent chromosome is 1st generation.
* Number of generations = Number of populations, which is done by performing mutation on chromosomes of a population.

**Pseudocode:**

**Algorithm Genetic()**

1. input number\_of\_cities, population\_size, number\_of\_generations.
2. FOR LOOP till population\_size:
3. Parent\_Chromosome = Generate\_chromosome()
4. Parent\_Fitness = Calculate\_fitness\_of\_chromosome()
5. If Parent\_Fitness < Optimal\_cost:
6. Optimal\_cost=Parent\_Fitness
7. Optimal\_path=Parent\_Chromosome
8. Append Chromosome,fitness in Initial\_population
9. WHILE LOOP till number\_of\_generations:
10. FOR LOOP till population\_size(iterator i):
11. Parent\_chromosome=Initial\_chromosome[i]
12. While true:
13. Child\_chromosome = mutated\_chromosome(Parent\_chromosome)
14. Child\_Fitness = Calculate\_fitness\_of\_chromosome()
15. If Child\_Fitness < Optimal\_cost:
16. Optimal\_cost=Child\_Fitness
17. Optimal\_path=Child\_chromosome
18. If Child\_Fitness < Parent\_Fitness:
19. Append Child\_chromosome in New\_population
20. Append Child\_Fitness in New\_population
21. Break
22. Else:
23. Percentage\_change\_fitness=(Child\_Fitness-Parent\_Fitness)/100
24. If Percentage\_change\_fitness > 0.50:
25. Append Child\_chromosome in New\_population
26. Append Child\_Fitness in New\_population
27. Break
28. Copy contents of New\_population to Initial\_population

3..Generating a chromosome is done by generating z-1 different random numbers in a range [2,z+1].1st and last city in the chromosome must be 1(origin city).Size of a chromosome=z+1.

4.Optimal cost of the path is found by performing mutation on the chromosomes of the initial population and thereby making new child chromosomes.

5.Make a new population by appending x valid chromosomes (along with their fitness value) which are generated due to mutation.

6.New population is created by performing mutation of chromosomes of previous population.

7.Mutation is performed by swapping two random cities.

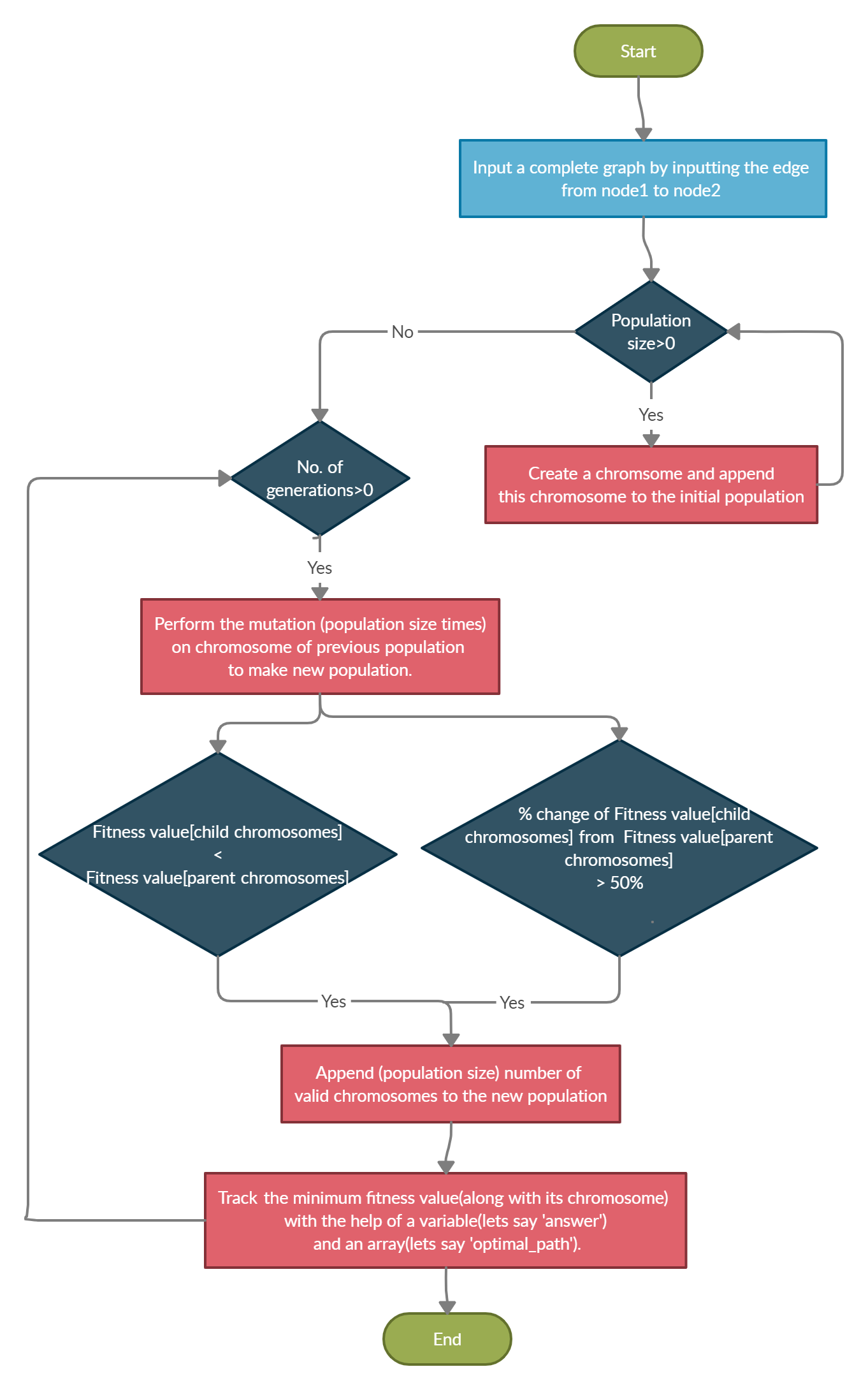
8.A valid chromosome after mutation is a chromosome whose fitness value is less than its parent chromosome's fitness value OR a valid chromosome after mutation is a chromosome whose percentage change of its fitness value from its parent's fitness value is more than 50%.

9.Track the minimum fitness value(along with its chromosome) with the help of a variable(lets say 'answer') and an array (lets say 'optimal\_path').

10.Repeat steps [5,9] 'y' times.

11.'answer' variable contains the optimal cost of the shortest route and 'optimal\_path' array contains the shortest route of the cities.

**Flowchart:**

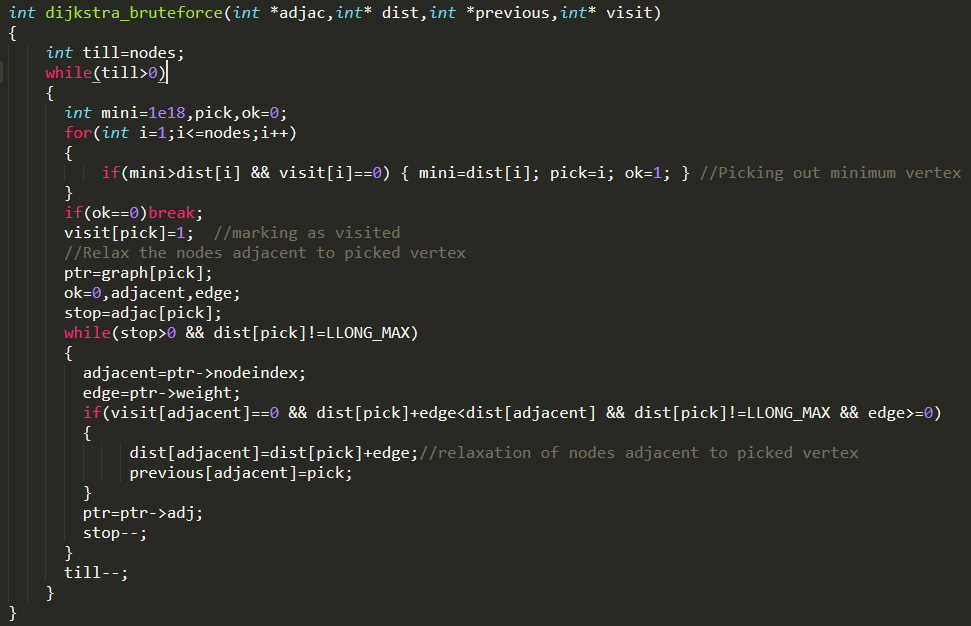


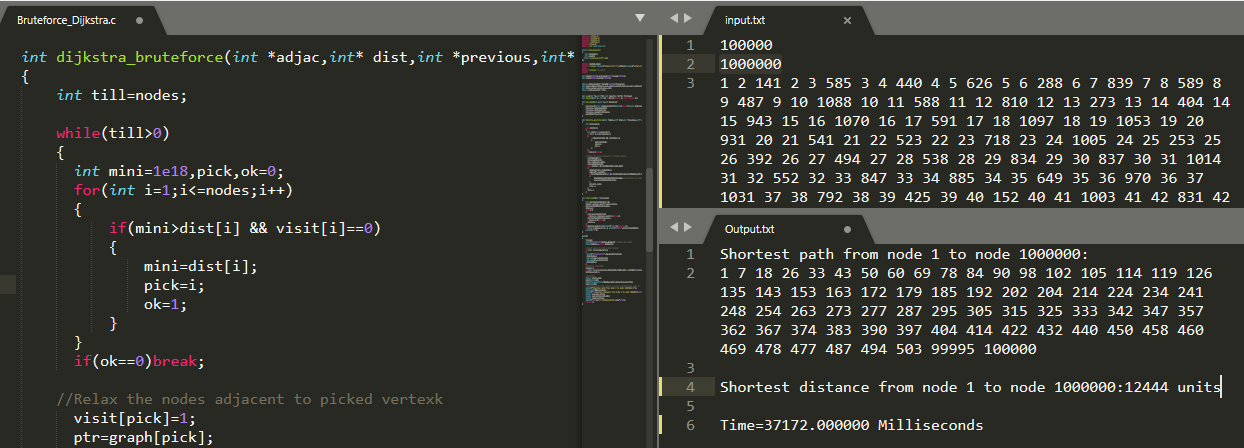
**Implementation**

**Implementation, Output screen and Result analysis of the 4 proposed algorithms in**

**Dijkstra algorithm**

**1. Bruteforce-based solution**

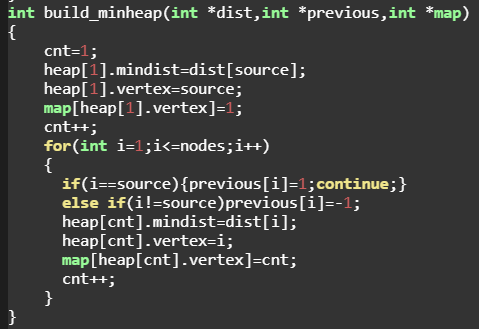
 **Implementation:**

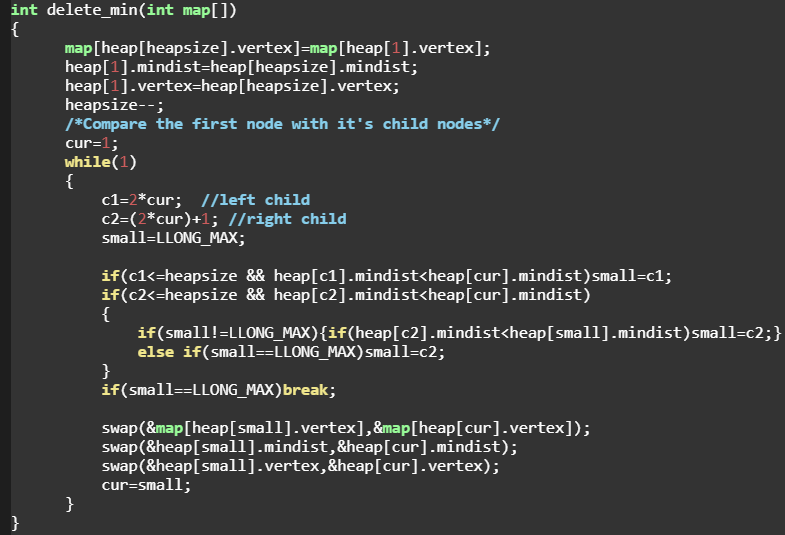
**Output Screen:**

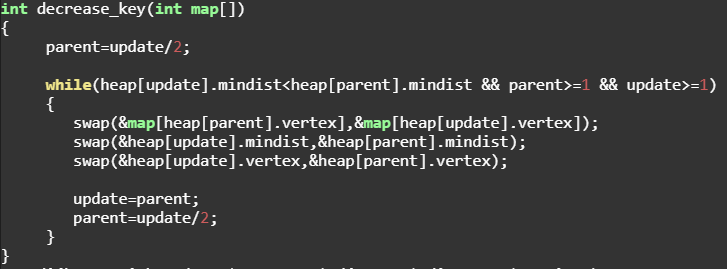
**Note:**

**Result analysis will be there after implementation and output screen of all the proposed algorithms in Dijkstra algorithm have been displayed.**

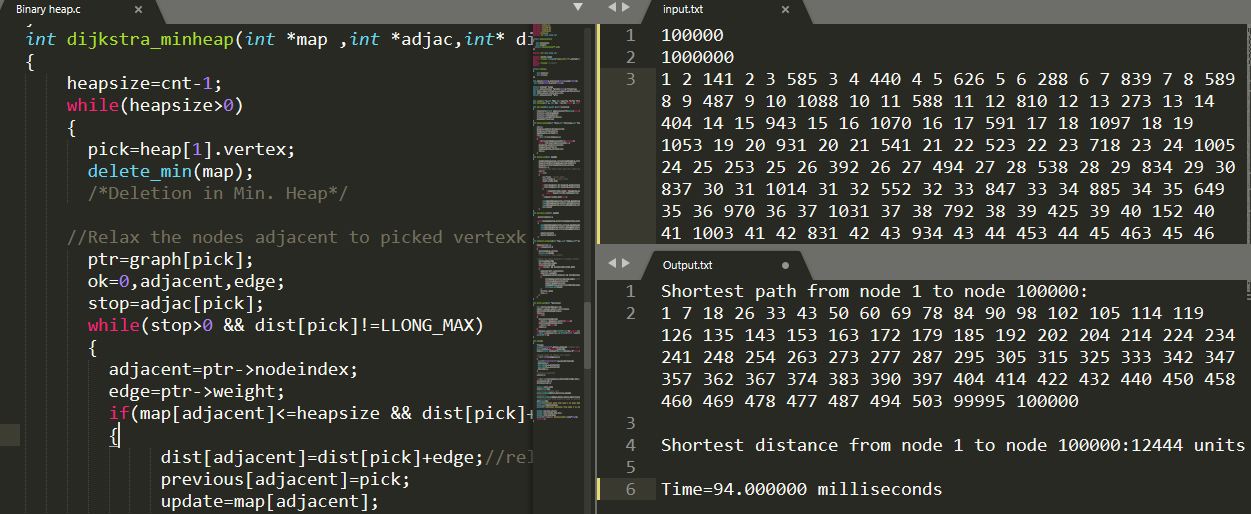
**2.Binary Heap Data-Structure Implementation:**



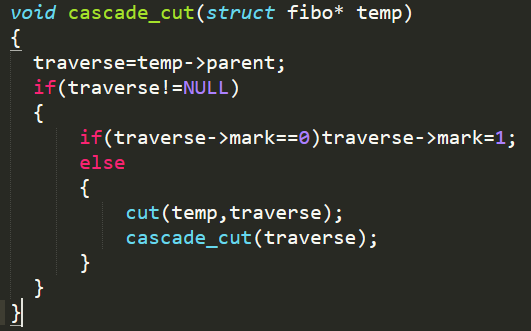
******

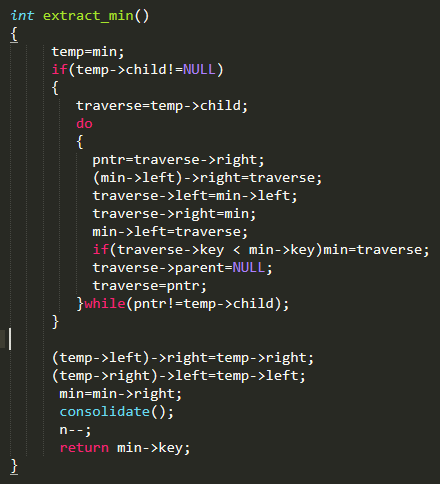


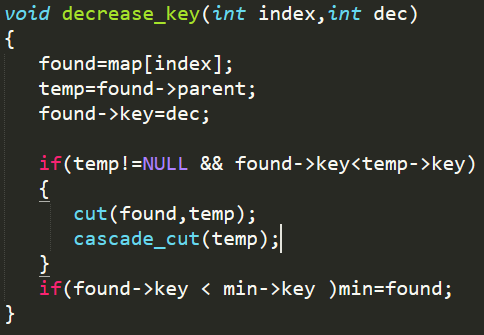
**Output screen:**

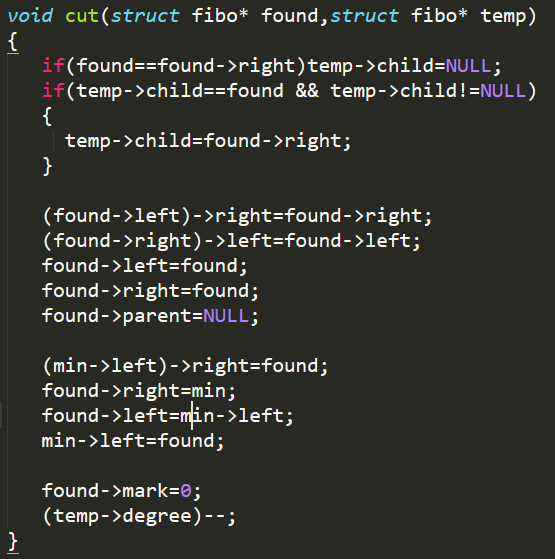
**Input:1 lakh nodes 10 lakh edges**

1. ****Fibonacci Heap Data Structure Implementation-**

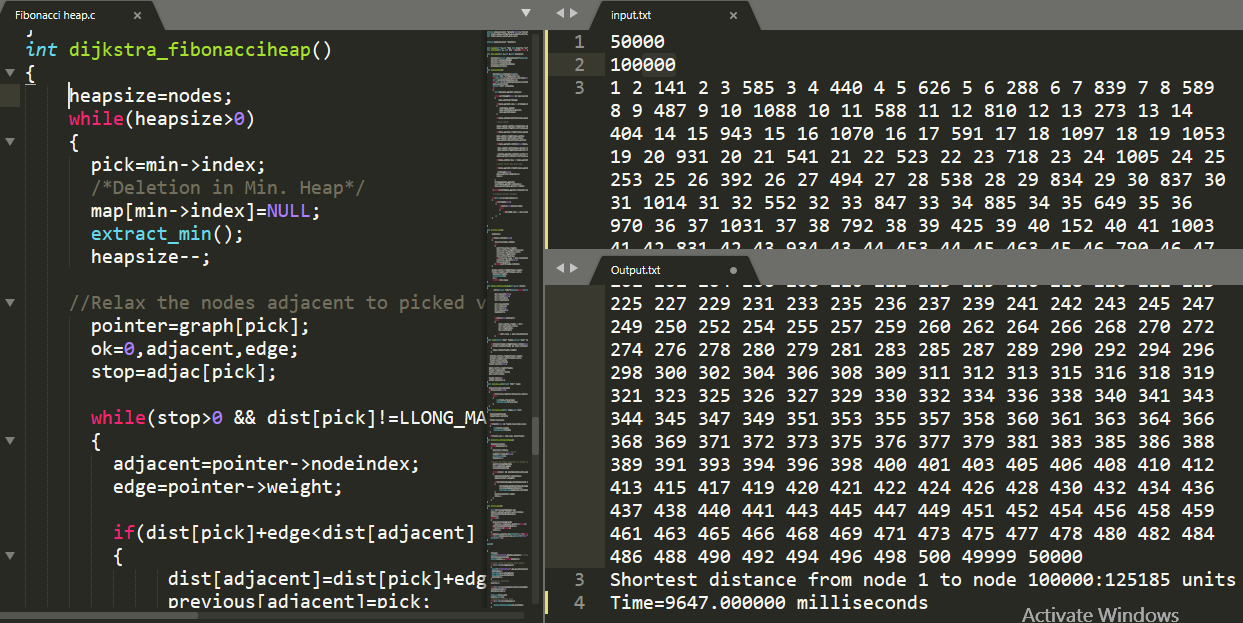


******

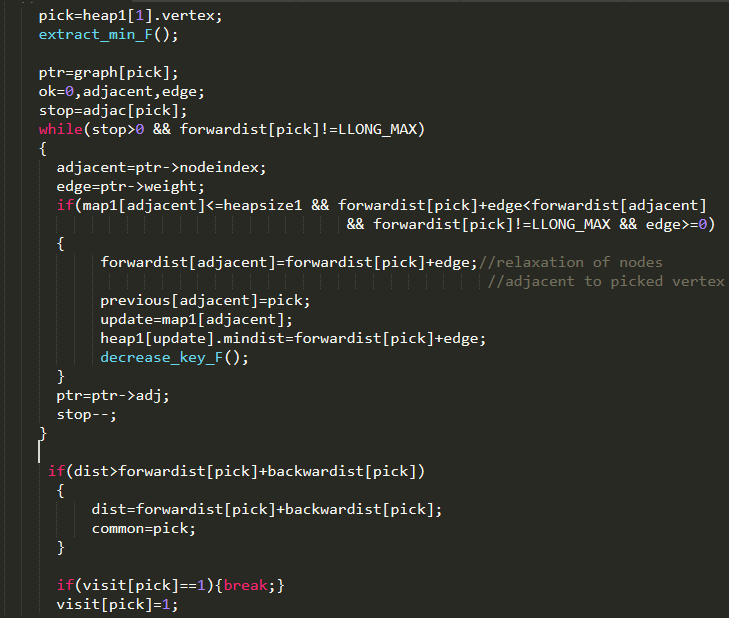




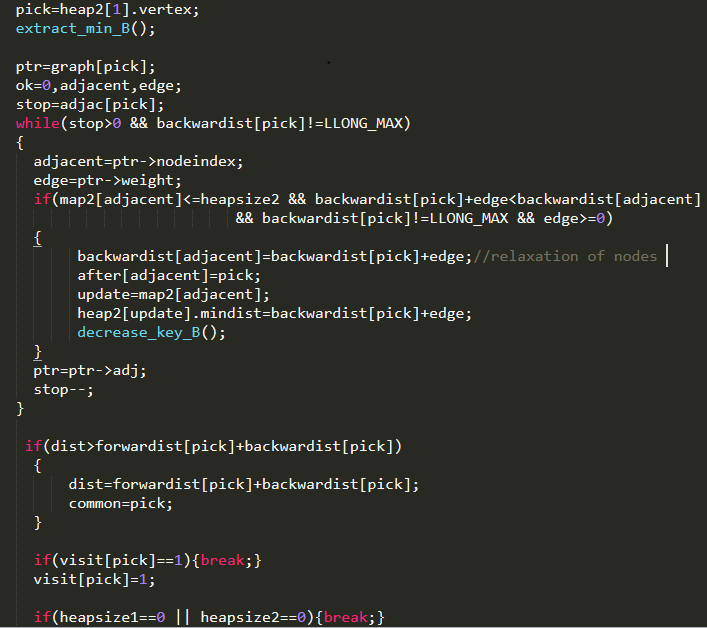
**Output Screen:**

**Input: 50 thousand nodes and 1 lakh edges**

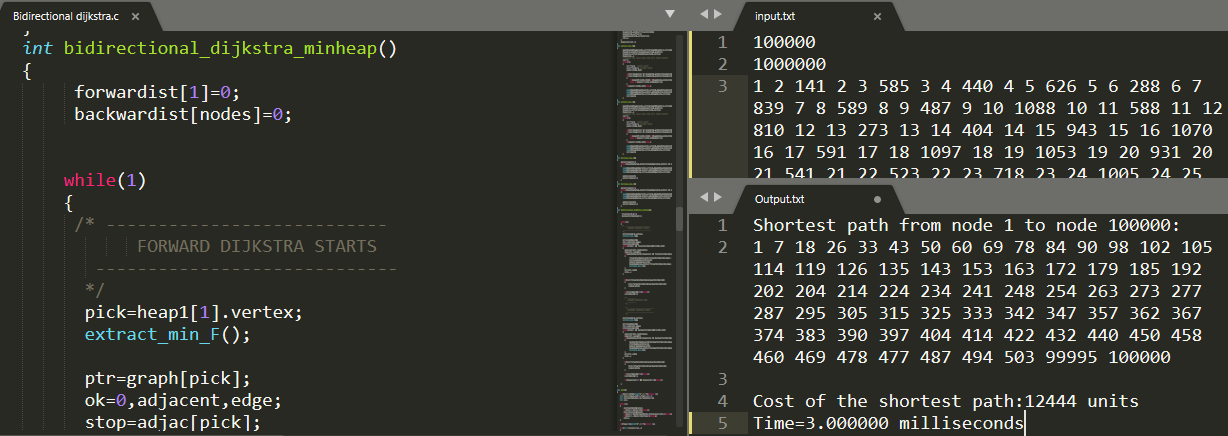
**3.Bidirectional Dijkstra algorithm**

**Forward Dijkstra code snippet:**

**Backward Dijkstra code snippet:**

**

**Output Screen:**

**Input:1 lakh nodes and 10 lakh edges**

**Result Analysis**

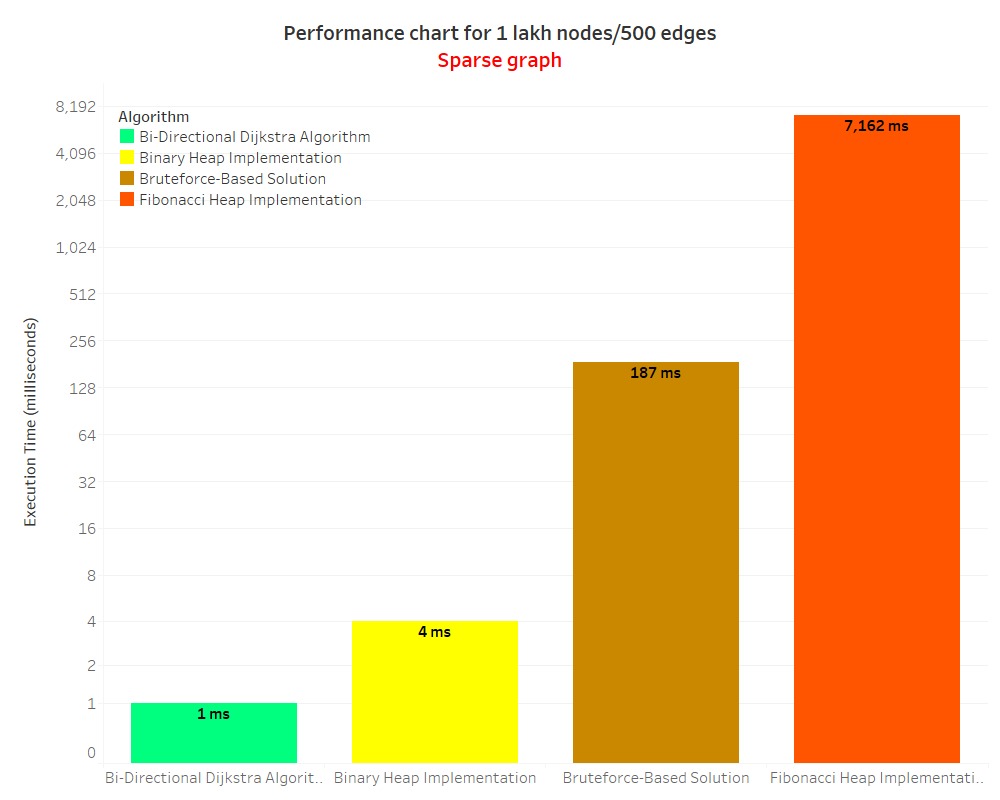
**Results of the 4 proposed algorithms of Dijkstra algorithm**

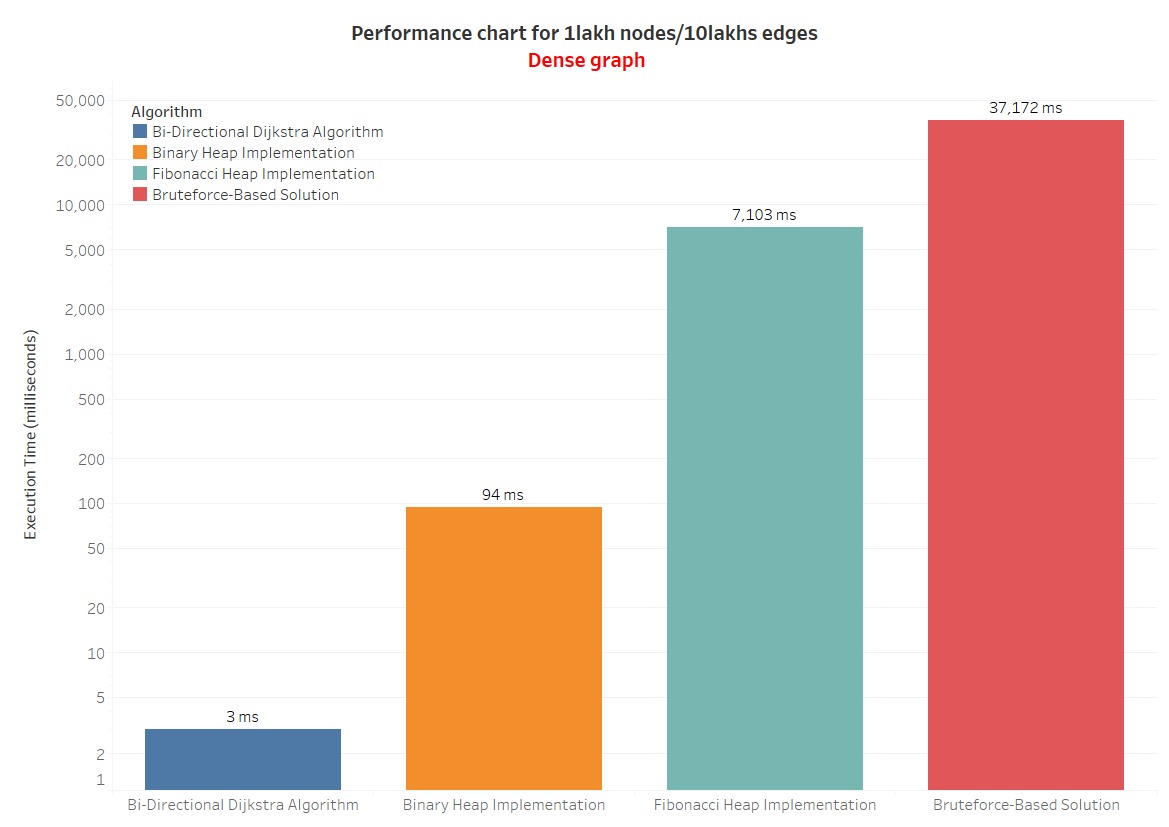
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bruteforce based solution** | **Number of nodes** | **Dense Graphs** | | | **Sparse Graphs** | | |
|  | **105 Edges** | **5\*105 Edges** | **106 Edges** | **500 Edges** | **2000 Edges** | **4000 Edges** |
| 10**4** Nodes | 392 ms | 485 ms | 526 ms | 16 ms | 79 ms | 156 ms |
| 5\*10**4** Nodes | 9417ms | 10442ms | 10828ms | 125ms | 437ms | 806ms |
| 105Nodes | 38559 ms | 39672 ms | 37172 ms | 187 ms | 798 ms | 1515 ms |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Binary heap implementation** | **Number of nodes** | **Dense Graphs** | | | **Sparse Graphs** | | |
|  | **105 Edges** | **5\*105 Edges** | **106 Edges** | **500 Edges** | **2000 Edges** | **4000 Edges** |
| 10**4** Nodes | 8ms | 32ms | 80ms | 1ms | 1ms | 2ms |
| 5\*10**4** Nodes | 22ms | 44ms | 66ms | 2ms | 3ms | 4ms |
| 105Nodes | 34ms | 68ms | 94ms | 4ms | 7ms | 8ms |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Bidirectional Dijkstra Algorithm** | **Number of nodes** | **Dense Graphs** | | | **Sparse Graphs** | | |
|  | **105 Edges** | **5\*105 Edges** | **106 Edges** | **500 Edges** | **2000 Edges** | **4000 Edges** |
| 10**4** Nodes | 1ms | 2ms | 3ms | 1 ms | 1 ms | 2 ms |
| 5\*10**4** Nodes | 1 ms | 4ms | 4ms | 3ms | 4ms | 3ms |
| 105Nodes | 2ms | 2ms | 3ms | 1ms | 3ms | 3ms |

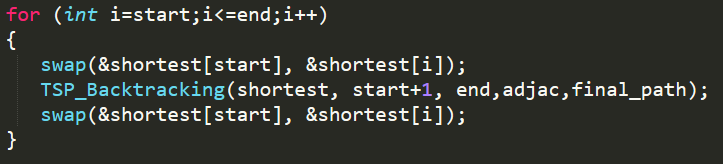
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Fibonacci heap implementation** | **Number of nodes** | **Dense Graphs** | | | **Sparse Graphs** | | |
|  | **105 Edges** | **5\*105 Edges** | **106 Edges** | **500 Edges** | **2000 Edges** | **4000 Edges** |
| 10**4** Nodes | 282ms | 291ms | 303ms | 292ms | 294ms | 296ms |
| 5\*10**4** Nodes | 9647ms | 9664ms | 9723ms | 9682ms | 9643ms | 9682ms |
| 105Nodes | 7023ms | 7054ms | 7103ms | 7162ms | 7191ms | 7215ms |

**Tableau visualization for 4 proposed algorithms for Dijkstra algorithm in terms of execution time**

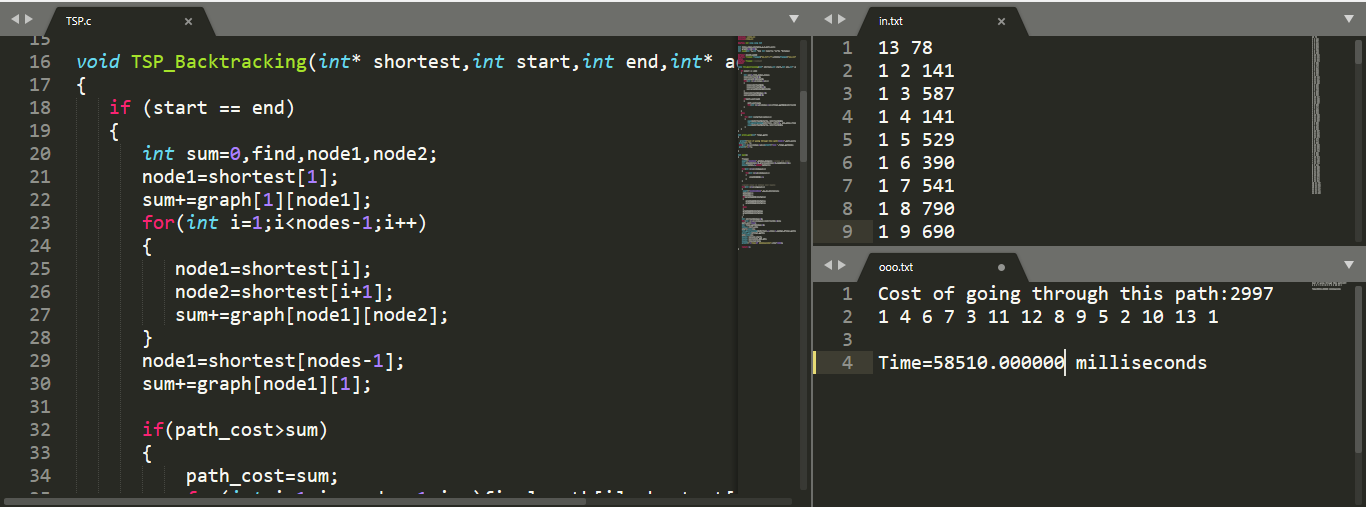


**1.Backtracking/Brute Force solution**

**Implementation, Output screen and Result analysis of the proposed algorithms in Travelling Salesman Problem**

**Implementation-**

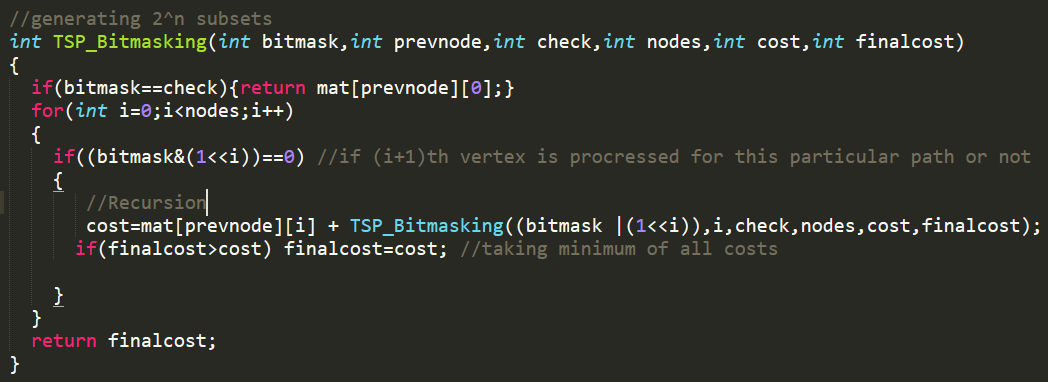
**Output Screen**

Input:13 nodes 78 edges(fully connected)

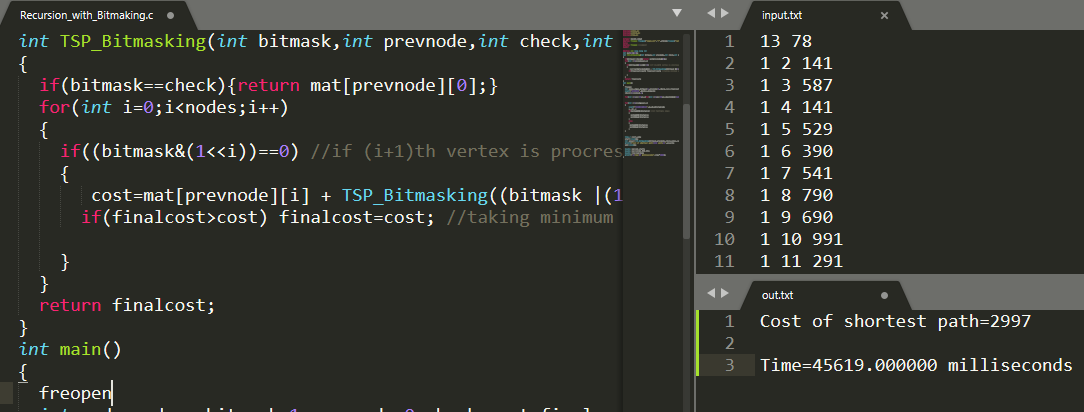
**Note:**

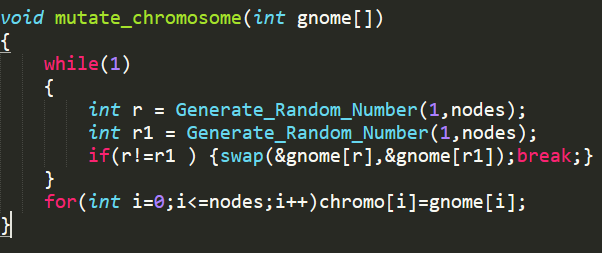
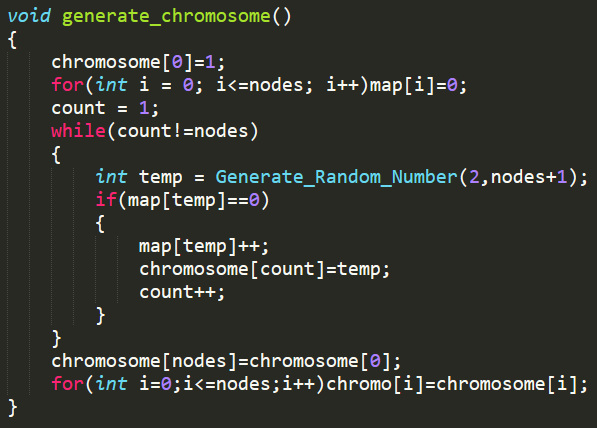
**Result analysis will be there after implementation and output screen of all the proposed algorithms in Travelling Salesman Problem have been displayed.**

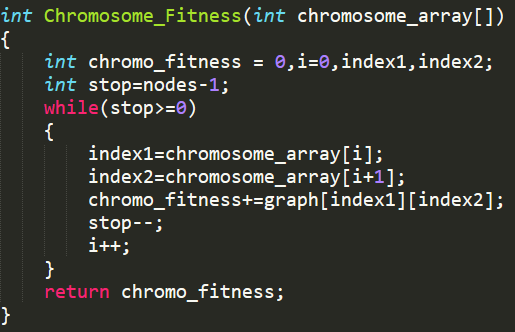
**2.Recursion with bitmasking method**

**Implementation-**

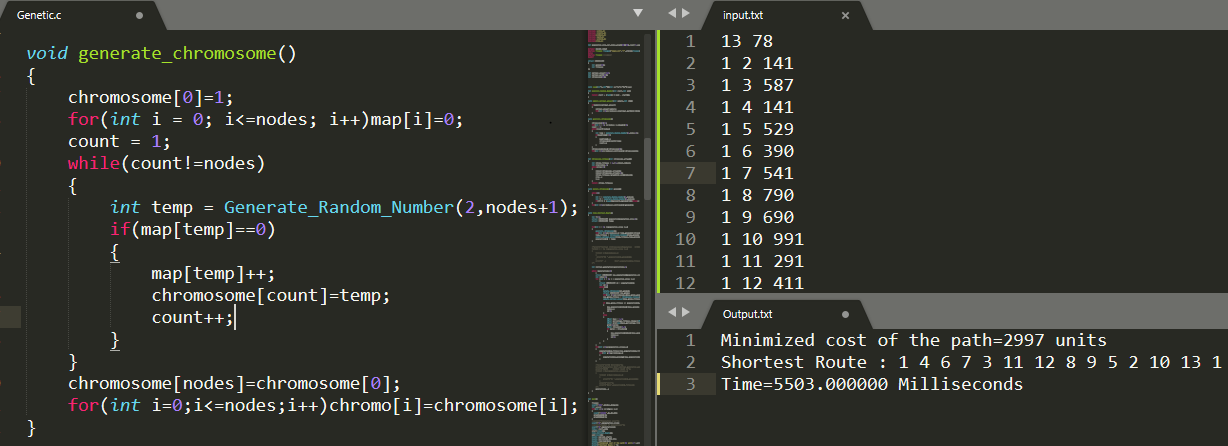
**Output Screen**

Input:13 nodes 78 edges(fully connected)

**3. Genetic Algorithm**

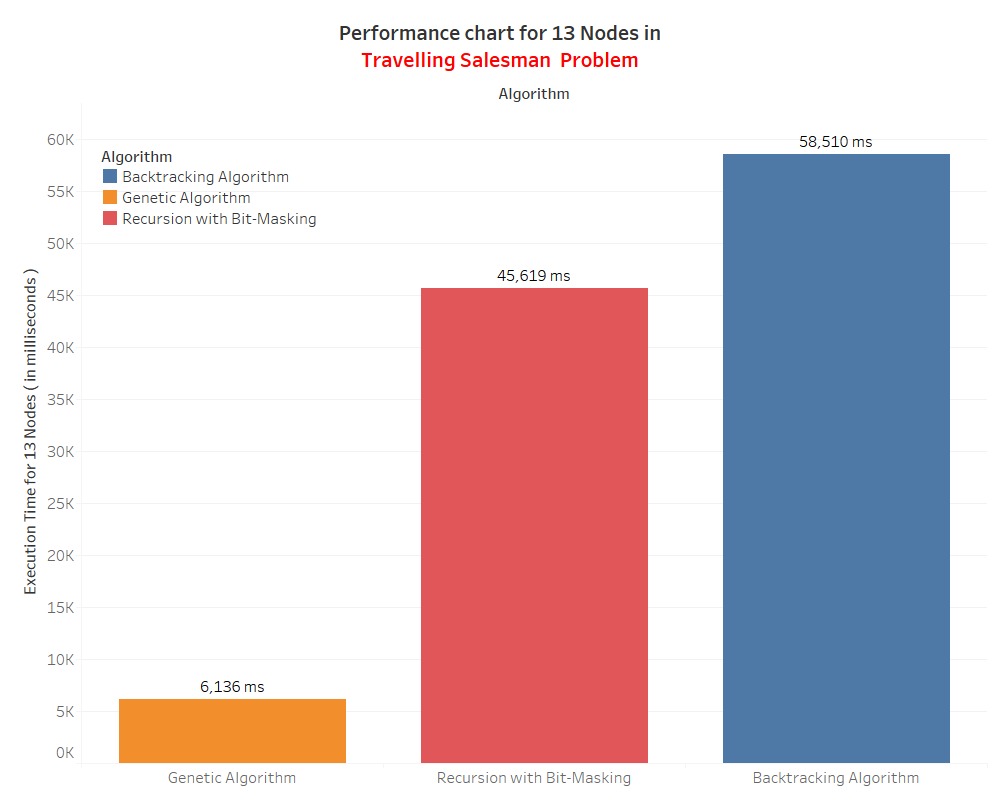


Output Screen:

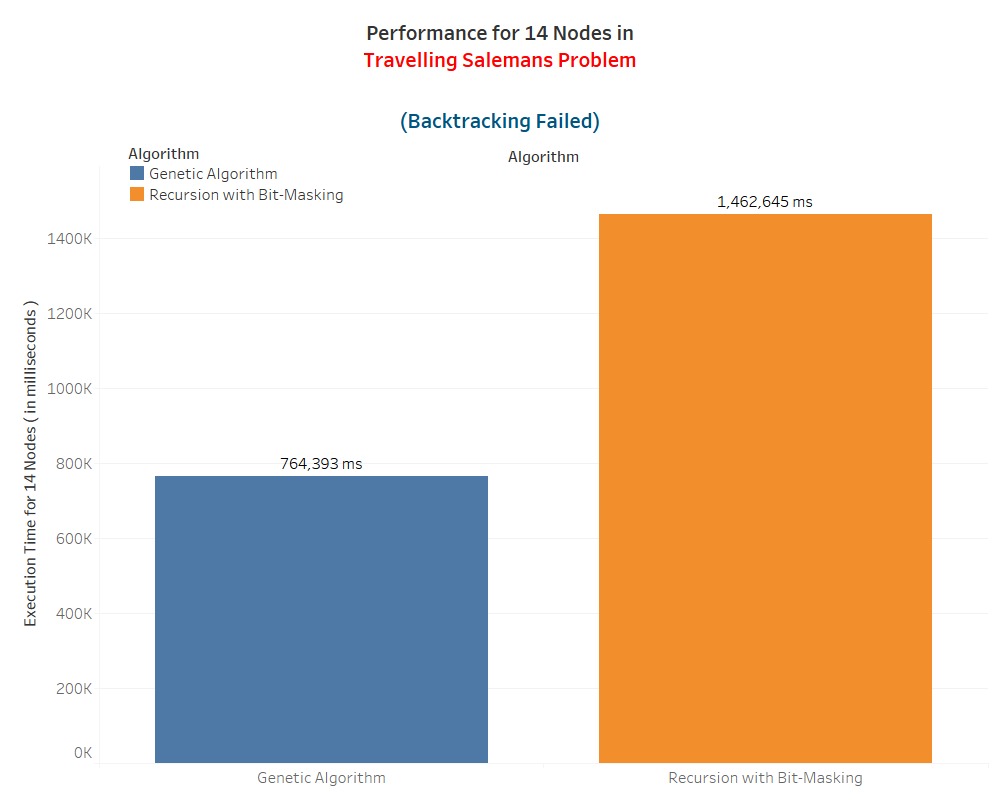
Input:13 nodes 78 edges (fully connected)

**Results of the 3 proposed algorithms for Travelling Salesman Problem**

|  |  |  |  |
| --- | --- | --- | --- |
| Number of Nodes | Backtracking Algorithm | Dynamic Programming Method | Genetic Algorithm |
| 10 | 48 ms | 30 ms | 1 ms |
| 11 | 328 ms | 311 ms | 16 ms |
| 12 | 4050 ms | 3613 ms | 156 ms |
| 13 | 58510 ms | 45619 ms | 6136 ms |
| 14 | Fail | 1462645 ms | 764393 ms |

****

**Tableau visualization for 3 proposed algorithms for Travelling Salesman Problem in terms of execution time**



**Conclusion and Future Scope**

In an emergency case where a person faces lethal health problems. Ambulance must take the shortest route to reach as fast as possible to the patient’s home.

**When an ambulance is needed to reach only one place from the hospital**.

Taking a real-world example:

In a map showing the hospital and patient’s home, map is having **1 lakh** places(nodes) and **10 lakh** connections between all 1 lakh places(edges), the proposed Bidirectional Dijkstra algorithm will suggest the shortest route from hospital to patient’s home in just **3 milliseconds**.

**When an ambulance is needed to reach multiple places from the hospital**,

Two variations in this case:

**1st Variation**

If hospital has already chosen the order in which they will visit the places according to the level of emergency:

**Solution:**

Then Bidirectional Dijkstra algorithm must be used to compute the shortest route. Suppose the order in which 3 places to be visited is: Hospital, place1, place2, place3, Hospital. Then first apply Bidirectional Dijkstra algorithm between hospital and place1, then apply the algorithm from place1 to place2, then apply the algorithm from place2 to place3, then apply the algorithm from place3 to hospital.

**2nd Variation**

If order of visiting places is not known, then it’s similar to Travelling Salesman Problem. Genetic algorithm will be used to provide the shortest route containing the order in which the places should be visited to save time.

**Solution:**

Taking a real-world example:

If there are 13 places to be visited, Genetic algorithm will provide the shortest route in just nearly **6 seconds**.

**Note:**

**In most cases, Hospital knows the order in which the ambulance must visit the patient places. So 1st variation is preferred.**

**Future scope**

* Artificial intelligence can be used to optimize Dijkstra algorithm. A\* algorithm is an AI algorithm which optimizes Dijkstra algorithm by generating heuristics. These heuristics are the estimates which helps in choosing the optimal path from source to destination.
* Contraction hierarchies is another smart technique to optimize Dijkstra algorithm. Contraction hierarchies is a speed-up method optimized to exploit properties of graphs representing road networks. The speed-up is achieved by creating shortcuts in a preprocessing phase which are then used during a shortest-path query to skip over "unimportant" vertices. This is based on the observation that road networks are highly hierarchical. Some intersections, for example highway junctions, are "more important" and higher up in the hierarchy than for example a junction leading into a dead end.
* Travelling Salesman Problemcan be further optimized by a nature inspired technique named Ant Colony Method. In this method, artificial ants travel from origin city and then take different orders to visit all cities and returns back to the origin city. While travelling through the cities, they release a fluid called pheromone. The path which has largest amount of pheromone is the shortest path.

**References**

1. https://en.wikipedia.org/wiki/Shortest\_path\_problem
2. <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>
3. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
4. <https://www.geeksforgeeks.org/binary-heap/>
5. <https://en.wikipedia.org/wiki/Fibonacci_heap>
6. Luyao Chen\*, Ji Zhou , Jing Li & Yunhao Chen, Bidirectional dijkstra algorithm for best-routing of Urban traffic network Beijing : doi: 10.1117/12.764679
7. [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)
8. https://en.wikipedia.org/wiki/Contraction\_hierarchies
9. <https://en.wikipedia.org/wiki/Travelling_salesman_problem>
10. <https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/>
11. <https://en.wikipedia.org/wiki/Genetic_algorithm>
12. https://en.wikipedia.org/wiki/Ant\_colony\_optimization\_algorithms

**Approval by Respected mentor**

