

**Table of contents**

1.INTRODUCTION …………………………………………………………………………………………………………………. 3

2.EXPERIMENTAL DESIGN …………………………………………………………………………………………………….. 3

3. EXPERIMENTAL IMPLEMENTATION …………………………………………………………………………………… 3

4.DATA COLLECTION AND INTERPRETATION OF RESULTS …………………………………………………….. 4

5.CONCLUSION ……………………………………………………………………………………………………………………… 5

6.EPILOGUE …………………………………………………………………………………………………………………………… 6

UNFORSEEN CIRCUMSTANCE............................................................................................. 17

LESSONS LEARNT ............................................................................................................... 17

ONE MORE OPPORTUNITY ................................................................................................. 17

LESSONS FOR FUTURE STUDENTS ....................................................................................... 17

7.APPENDIX …………………………………………………………………………………………………………………………… 7

PROGRAM LISTINGS …………………………………………………………………………………………………… 12

OUTPUT ……………………………………………………………………………………………………………………… 12

1. **INTRODUCTION:**

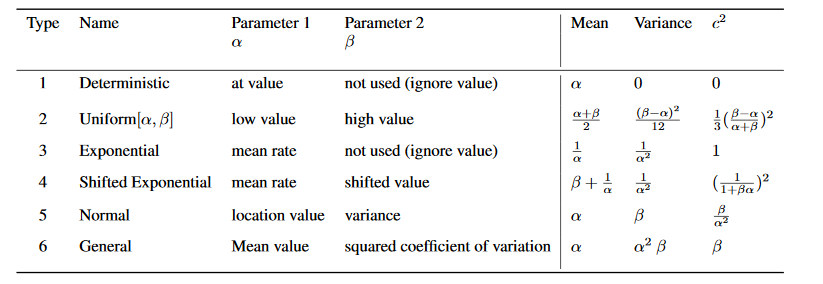
Most of the communication networks are related to concept of graphs these days. In such networks, nodes are the cities, or intersections, or server nodes, and the links connect them. We always have various plans to travel from one node to another node such as by car, by flight or routing internet traffic. Basically, our plan should include source, destination and a planned path in between. Here we built up this project by considering road network between cities as a motivating application. We always look for a shortest route when we plan a trip from source (a) to destination (b). Here cities are termed as nodes. Here, there will be traffic as many cars travel and the traffic is assumed to be random. So, we get delayed due to the traffic. This delay, which is rarely constant along an edge is calculated according to the random variable. So, we define our shortest path in terms of a stochastic setting. We call this path as Uncertain Shortest Path.

1. **EXPERIMENTAL DESIGN:**

We have a number of different input scenario's. Each scenario will have :

1. The size n of the system (the total number of cities), together with the index of two cities, a and b.
2. An adjacency matrix E with edge-weights. The (i,j)th entry E[i,j] represents the main characteristics of the random variable representing the travelling time on the edge between two specific nodes, i and j.
3. Type of distribution 'T' is one of the main characteristics with two more parameters α and β. Edge cost between the nodes is distributed according to different types of distributions.

We have considered the following distributions and calculated Mean, Variance and C^2 based on the type of distribution.

We calculate Mean, Variance and C^2 and generate adjacency matrix E for each criteria.

**We work on the following criteria:**

* **Mean Value** : Pick the path whose total "expected value" is smallest from among all paths. In this case, expected value is interpreted as the edge-cost. Assuming that all link's random variables are independent, total expected value is equal to sum of the expected value of the individual random variables.
* **Optimist**: Pick the path such that it's "expected value - standard deviation" is smallest from among all paths. In this case, "expected value - standard deviation" is interpreted as the edge-cost.

We calculate the expected value and standard deviation for each and every edge and then subtract standard deviation from mean this particular value is considered as edge cost in case of optimist.

* **Pessimist**: Pick the path such that it's "expected value + standard deviation" is smallest from among all paths. In this case "expected value + standard deviation" is interpreted as the edge-cost.

We calculate the expected value and standard deviation for each and every edge and then add standard deviation to mean, this particular value is considered as edge cost in case of pessimist.

* **Double Pessimist**: Pick the path such that it's "expected value + 2 \* standard deviation" is smallest from among all paths. In this case "expected value + 2 \* standard deviation" is interpreted as the edge-cost.

We calculate the expected value and standard deviation for each and every edge and then add square of the standard deviation to the mean, this particular value is considered as edge cost in case of double pessimist.

* **Stable**: Pick the path such that it's "squared coefficient of variation" is smallest from among all paths. In this case, "squared co-efficient of variation" is interpreted as the edge-cost.

**Here we calculate the c^2 value each and every edge based on the type of distribution you consider and the minimum of this value is considered as the edge weight or edge cost in case of stable.**

* **Triple Pessimist:** Pick the path such that it's "expected value + 3 \* standard deviation" is smallest from among all paths. In this case "expected value + 3 \* standard deviation" is interpreted as the edge-cost.

We calculate the expected value and standard deviation for each and every edge and then add cubic of the standard deviation to the mean, this particular value is considered as edge cost in case of double pessimist.

1. **EXPERIMENTAL IMPLEMENTATION:**

**Programming Language Used :**

For this problem we have used JAVA as the programming language because it is an Object oriented Language and will let the developers “write once, run anywhere”. And the members in our team are confident in Java Programming language so we have preferred this programming language to solve this problem. We have Adjacency Matrix in Java that is used to represent the edge weights between two nodes.

Here we are reading the input file and we calculate the mean and variance based on the type of distribution between two nodes. Here we are Using 6 different distributions namely Deterministic, Uniform, Exponential, Shifted Exponential, Normal and General as stated. After calculating the edge weights of corresponding nodes the edge weights calculated based on different criteria. We have calculated edge weights based on different criteria like Mean, Optimist, Pessimist, Double Pessimist, Stable as stated and we have considered a new criteria namely Triple Pessimist as the new criteria.

Data structures Used :

To solve this problem we have used data structures like Priority Queue, Adjacency Matrix, and Array.

As we are calculating the edge weights between start and end node we are representing the edge weight between two nodes using Adjacency Matrix. We have used adjacency matrix because finding out whether or not an edge is completed in O(1) Operations.

We have used Priority Queue to store the node and its corresponding weight from the starting node. We have used priority queue because we can pop out the node that has minimum value from the queue. We have priorityqueue.poll() method that returns the node with minimum value. Once a node is processed and calculated the cost to reach the nodes adjacent to the current node all these nodes and corresponding costs are stored in Priority Queue. Inserting an element and deleting an element from priority queue can be completed in O(logn) operations.

We have used arrays to store the parent nodes of a particular node and also to store the distance from start node to the corresponding node. We have used array because fetching a value at a particular index id done in O(1) time complexity.

**ALGORITHM:**

*Algorithm Dijkstra (graph myGraph, node a ,node b)*

*PriorityQueue <- PriorityQueue.create()*

*Distance[n] <- MAX\_VALUE*

*Parent[n] <- 0*

*inTree[n] <- false*

*finalMean <- 0.0*

*finalStandardDeviation <- 0.0*

*finalCoefficientSquare <- 0.0*

*hopLength <- 0*

*for index in range(0, myGraph.length-1)*

*if myGraph[a][index] >= 0.0*

*if Distance[i] + myGraph[i][index] < Distance[index]*

*oldIndex <- Distance[index]*

*parent[index] <- a*

*Distance[index] <- Distance[i] + myGraph[i][index]*

*PriorityQueue.remove(index,oldIndex)*

*PriorityQueue.add(index,Distance[index])*

*While NOT PriorityQueue.IsEmpty do*

*Vertex <- PQ.poll()*

*for index in range(0, myGraph.length-1)*

*if myGraph[vertex.index][index] >=0.0 && inTree[index] == true then do*

*if Distance[vertex.index] + myGraph[vertex.index][index] < Distance[index] then do*

*oldIndex <- Distance[index]*

*parent[index] <- a*

*Distance[index] <- Distance[vertex.index] + myGraph[i][index]*

*PriorityQueue.remove(index,oldIndex)*

*PriorityQueue.add(index,Distance[index])*

*End do; End-if*

*End do; End-if*

*End-for*

*End while*

*Print "Length of Shortest path from 1 to 12 “ , Distance [j]*

*Print “Path : ”*

*printPath(parent,j,i)*

*calculateFinalMean(parent,Graph,newGraph,j,i);*

*calculateStandardDeviation(parent,Graph,newGraph,j,i);*

*calculatefinalCoefficientSquare(parent,Graph,newGraph,j,i);*

*Print "μ - σ : ", finalMean – finalStandardDeviation*

*Print "μ : ",finalMean*

*Print "μ + σ : ",finalMean + finalStandardDeviation*

*Print "μ + 2σ : ",finalMean + (2 \* finalStandardDeviation);*

*Print "C^2 : " , finalCoefficientSquare;*

*Print "Hop Length : ", hopLength;*

*If Distance[b] == MAX\_VALUE or Distance[b] < 0 then*

*return -1*

*else*

*return Distance[j]*

*end-if*

*end-algorithm*

*// Algorithm for calculating Final Coefficient Square*

*// new Graph contains mean, standard deviation and c^2 values of each and every edge.*

*Algorithm calculatefinalCoefficientSquare (graph myGraph,graph newGraph, node a ,node b)*

*If parent[vertex] == 0 then do*

*Return*

*Else*

*finalCoefficientSquare = finalCoefficientSquare + newGraph[parent[vertex]][vertex].coefficientSquare*

*calculatefinalCoefficientSquare(parent,Graph,newGraph,parent[vertex], startNode)*

*end-if*

*end-algorithm*

*// Algorithm for calculating Final Standard Deviation*

*Algorithm calculateStandardDeviation (graph myGraph,graph newGraph, node a ,node b)*

*If parent[vertex] == 0 then do*

*Return*

*Else*

*finalStandardDeviation = finalStandardDeviation + newGraph[parent[vertex]][vertex].b*

*calculateStandardDeviation(parent,Graph,newGraph,parent[vertex], startNode)*

*end-if*

*end-algorithm*

*// Algorithm for calculating Final Mean*

*Algorithm calculateFinalMean(graph myGraph,graph newGraph, node a ,node b)*

*If parent[vertex] == 0 then do*

*Return*

*Else*

*finalMean = finalMean + newGraph[parent[vertex]][vertex].a*

*calculateFinalMean(parent,Graph,newGraph,parent[vertex],startNode)*

*end-if*

*end-algorithm*

1. **DATA COLLECTION AND INTERPRETATION OF RESULTS:**

|  |  |  |
| --- | --- | --- |
| Test Case | Expected Result | Actual Result |
| Check whether the input file is read and n, start and destination values are obtained | Verified whether the file is read without errors and read the start and destination values from files. | Program successfully reads the input values from the file and n, start and destination nodes are obtained. |
| Verify whether mean, variance and c^2 values are correctly calculated based on the distribution between edges. | Verified whether the mean, variance and c^2 values are correct | Mean, Variance and c^2 values are calculated correctly. |
| Verify that the edge weights are properly updated in the adjacency matrix | Verified whether the edge weights between nodes are correctly updated in the adjacency matrix | All the edge weights are correctly updated in the adjacency matrix. |
| Verify that the program calculates the shortest path between source and destination using 6 different criteria | Verified that the program calculates shortest path between source and destination using 6 different criteria | Program successfully calculates the shortest path between source and destination nodes using 6 different criteria. |
| Verify that the Dijkstra algorithm gives the appropriate shortest path between the nodes | Verified that the Dijkstra algorithm gives the appropriate shortest path between the nodes | Program successfully gives output for the calculation of shortest paths using Dijkstra Algorithm. |
| Verify that the shortest path between source and destination is printed correctly | Verified that the path between source and destination is printed correctly | Program successfully prints the shortest path source and destination nodes. |
| Verify that the path obtained between source and destination is the Shortest Path using Dijkstra | Verified that the path obtained between source and destination is the shortest path calculated using Dijkstra | Program successfully verifies that the path obtained is the shortest path using Dijkstra. |
| Verify that the hop count is printed for given source and destination | Verified that the hop count is printed for given source and destination | Program successfully verifies the hop count and is printed for given source and destination. |

**Input3:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | μ – σ | μ | μ + σ | μ + 2σ |  | hops | Shortest Path |
| Mean Value | 69.91709623134523 | 130.99980100222098 | 192.08250577309673 | 253.16521054397248 | 2.267162638587282 | 6 | 1->3->5->7->8->10->12 |
| Optimist | 69.91709623134523 | 130.99980100222098 | 192.08250577309673 | 253.16521054397248 | 2.267162638587282 | 6 | 1->3->5->7->8->10->12 |
| Pessimist | 140.0 | 141.0 | 142.0 | 143.0 | 0.002066115702479339 | 6 | 1->2->4->7->9->10->12 |
| Doubly Pessimist | 140.0 | 141.0 | 142.0 | 143.0 | 0.002066115702479339 |  | 1->2->4->7->9->10->12 |
| Stable | 140.0 | 141.0 | 142.0 | 143.0 | 0.002066115702479339 | 6 | 1->2->4->7->9->10->12 |
| Triple Pessimist | 140.0 | 141.0 | 142.0 | 143.0 | 0.002066115702479339 | 6 | 1->2->4->7->9->10->12 |

After finding the shortest path between source and destination using the different criteria namely Mean value, Optimist, Pessimist, Double Pessimist, Stable, Triple Pessimist we have calculated the mean and standard deviation for the shortest path obtained using each and every criteria we have calculated the mean, standard deviation, and hops count for the shortest path between source and destination and using the following formulas.

When comparing the above results we find that the mean and optimist criteria have less mean value when compared to pessimist, stable, doubly pessimist and Triple Pessimist. The hop length for all the criteria is the same which is 6.

E [] = E [] + E [] + . . . + E []

V [] = V [] + V [] + . . . + V []

We have calculated mean and variance of the entire shortest path by adding the individual mean and variances of all the edges in the shortest path obtained by implementing Dijkstra Algorithm.

We have calculated using the following formula:

[] = [] + [] + . . . + []

We have calculated by adding the squared coefficient of each and every edge in the final shortest path obtained by implementing Dijkstra Algorithm.

In the below represent the edges obtained from source to destination while calculating the shortest path using Dijkstra Algorithm using different criteria like Mean, Optimist, Pessimist, Double Pessimist, Stable and Triple Pessimist. We have obtained the edge from 10 -> 12 in all the 6 methods and hence 10 -> 12 is the edge that is used maximum number of times.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Mean Value | Optimist | Pessimist | Double Pessimist | Stable | Triple Pessimist | No. of Paths |
| 1 -> 2 | N | N | Y | Y | Y | Y | 4 |
| 1 -> 3 | Y | Y | N | N | N | N | 2 |
| 2 -> 4 | N | N | Y | Y | Y | Y | 4 |
| 3 -> 5 | Y | Y | N | N | N | N | 2 |
| 4 -> 7 | N | N | Y | Y | Y | Y | 4 |
| 5 -> 7 | Y | Y | N | N | N | N | 2 |
| 7 -> 8 | Y | Y | N | N | N | N | 2 |
| 7 -> 9 | N | N | Y | Y | Y | Y | 4 |
| 8 -> 10 | Y | Y | N | N | N | N | 2 |
| 9 -> 10 | N | N | Y | Y | Y | Y | 4 |
| 10 -> 12 | Y | Y | Y | Y | Y | Y | 6 |

**Input 4:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | μ – σ | μ | | μ + σ | | μ + 2σ | |  | | | hops | Shortest Path | |
| Mean Value | | 29.0 | 136.99971900524895 | | 244.9994380104979 | | 352.99915701574685 | | 5.0 | | | 6 | 1->3->5->6->8->11->12 | |
| Optimist | | 29.0 | 136.99971900524895 | | 244.9994380104979 | | 352.99915701574685 | | 5.0 | | | 6 | 1->3->5->6->8->11->12 | |
| Pessimist | | 140.95854811567264 | 152.99998800002402 | | 165.0414278843754 | | 177.08286776872677 | | 0.11412809705921376 | | | 6 | 1->2->4->6->9->11->12 | |
| Doubly Pessimist | | 140.95854811567264 | 152.99998800002402 | | 165.0414278843754 | | 177.08286776872677 | | 0.11412809705921376 | | | 6 | 1->2->4->6->9->11->12 | |
| Stable | | 140.95854811567264 | 152.99998800002402 | | 165.0414278843754 | | 177.08286776872677 | | 0.11412809705921376 | | | 6 | 1->2->4->6->9->11->12 | |
| Triple Pessimist | | 140.95854811567264 | 152.99998800002402 | | 165.0414278843754 | | 177.08286776872677 | | 0.11412809705921376 | | | 6 | 1->2->4->6->9->11->12 | |
|  | Mean Value | | | Optimist | | Pessimist | | Double Pessimist | | Stable | Triple Pessimist | | | No. of Paths | |
| 1 -> 2 | N | | | N | | Y | | Y | | Y | Y | | | 4 | |
| 1 -> 3 | Y | | | Y | | N | | N | | N | N | | | 2 | |
| 2 -> 4 | N | | | N | | Y | | Y | | Y | Y | | | 4 | |
| 3 -> 5 | Y | | | Y | | N | | N | | N | N | | | 2 | |
| 4 -> 6 | N | | | N | | Y | | Y | | Y | Y | | | 4 | |
| 5 -> 6 | Y | | | Y | | N | | N | | N | N | | | 2 | |
| 6 -> 8 | Y | | | Y | | N | | N | | N | N | | | 2 | |
| 6 -> 9 | N | | | N | | Y | | Y | | Y | Y | | | 4 | |
| 8 -> 11 | Y | | | Y | | N | | N | | N | N | | | 2 | |
| 9 -> 11 | N | | | N | | Y | | Y | | Y | Y | | | 4 | |
| 11 -> 12 | Y | | | Y | | Y | | Y | | Y | Y | | | 6 | |

**Calculations:**

**Criteria Considered: Mean**

Adjacency Matrix:

1 2 3 4 5 6 7 8 9 10 11 12

1 -1.0 24.0 23.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

2 -1.0 -1.0 -1.0 25.0 30.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

3 -1.0 -1.0 -1.0 20.000009000009 22.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

4 -1.0 -1.0 -1.0 -1.0 -1.0 25.0 25.0 -1.0 -1.0 -1.0 -1.0 -1.0

5 -1.0 -1.0 -1.0 -1.0 -1.0 28.0 21.000021000021 -1.0 -1.0 -1.0 -1.0 -1.0

6 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 22.0 23.0 -1.0 -1.0 -1.0

7 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 21.999780002199977 22.0 -1.0 -1.0 -1.0

8 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 20.0 28.0 -1.0

9 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 22.0 30.0 -1.0

10 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 23.0

11 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 28.0

12 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

Path: 1 -> 3 -> 5 -> 7 -> 8 -> 10 -> 12

Shortest Path = 23.0 + 22.0 + 21.000021000021 + 21.999780002199977 + 20.0 + 23.0

= 130.99980100222098

Length of Shortest path from 1 to 12 using Mean Method: 130.99980100222098

Mean = 23.0 + 22.0 + 21.000021000021 + 21.999780002199977 + 20.0 + 23.0

= 130.99980100222098

Standard Deviation = 10.0 + 3.4641016151377544 + 21.000021000021 + 21.999780002199977 + 4.618802153517006 + 0.0

= 61.082705

μ – σ (Mean – Standard Deviation) = 130.99980100222098 - 61.082705

= 69.91709623134523

μ (Mean) = 23.0 + 22.0 + 21.000021000021 + 21.999780002199977 + 20.0 + 23.0

= 130.99980100222098

μ + σ (Mean + Standard Deviation) = 130.99980100222098 + 61.082705

= 192.08250577309673

μ + 2σ (Mean + 2 \* Standard Deviation) = 130.99980100222098 + (2 \* 61.082705)

= 253.16521054397248

(Squared Coefficient) = 0.18903591682419663 + 0.024793388429752063 + 1.0 + 1.0 +0.053333333333333344 + 0.0

= 2.267162638587282

Hop Length = 6

**Criteria Considered: Pessimist**

Adjacency Matrix:

1 2 3 4 5 6 7 8 9 10 11 12

1 -1.0 24.0 43.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

2 -1.0 -1.0 -1.0 25.0 30.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

3 -1.0 -1.0 -1.0 38.000027000027 28.928203230275507 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

4 -1.0 -1.0 -1.0 -1.0 -1.0 45.0 25.0 -1.0 -1.0 -1.0 -1.0 -1.0

5 -1.0 -1.0 -1.0 -1.0 -1.0 38.392304845413264 63.000063000063 -1.0 -1.0 -1.0 -1.0 -1.0

6 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 24.0 25.0 -1.0 -1.0 -1.0

7 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 65.99934000659994 24.0 -1.0 -1.0 -1.0

8 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 29.23760430703401 30.0 -1.0

9 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 22.0 30.0 -1.0

10 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 23.0

11 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 30.0

12 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0

Path: 1 -> 2 -> 4 -> 7 -> 9 -> 10 -> 12

Shortest Path = 24.0 + 25.0 + 25.0 + 24.0 + 22.0 + 23.0

= 142.0

Mean = 24.0 + 25.0 + 25.0 + 23.0 + 22.0 + 23.0

= 142.0

Standard Deviation = 0.0 + 0.0 + 0.0 + 1.0 + 0.0 + 0.0

= 1.0

μ – σ (Mean – Standard Deviation) = 141.0 – 1.0

= 140.0

μ (Mean) = 141.0

μ + σ (Mean + Standard Deviation) = 141.0 + 1.0

= 142.0

μ + 2σ (Mean + 2 \* Standard Deviation) = 141.0 + (2 \* 1.0)

= 143.0

(Squared Coefficient) = 0.0 + 0.0 + 0.0 + 0.002066115702479339 + 0.0 +0.0

= 0.002066115702479339

Hop Length = 6

1. **CONCLUSION:**

**Mean:**

**Pros :**

**When considering the mean as criteria to calculate the edge weights between any two nodes, the mean will just predict that there is path between source and destination.**

**Cons:**

**The disadvantage of this criteria is that it doesn’t consider any measurements it will just predict that there exists a path between source and destination without any analysis. Hence mean method is not useful to find the shortest path between source and destination nodes.**

**Optimist:**

**Pros:**

**When trying to find out path between source and destination optimist criteria predicts that there is a straight path between the two nodes. It has more accuracy when compared to other criterion. It also determines the time to reach the destination from source. There are many risks to reach from source to destination.**

**Cons:**

**The disadvantage of this criteria is that it overcomes all the risks to reach destination from the source node. Even though the risks are present it will neglect all the risks.**

**Pessimist:**

**Pros:**

**When trying to reach from source to destination using pessimist approach there will risks between the adjacent path. Pessimist tries to reach the destination node by considering all the risks.**

**Cons:**

**The disadvantage of this criteria is that it has less accuracy and takes much time to reach the destination node from source as it considers all the risks to reach the source node.**

**Double Pessimist:**

**Pros:**

**Double pessimistic criteria will have more risks to reach destination from source when compared to pessimist criteria. It considers all the risks and reaches the destination node.**

**Cons:**

**The disadvantage with this criteria is that it takes much more time than pessimist to overcome all the risks between adjacent nodes and then reaches the destination.**

**Stable:**

**Pros :**

**Stable criteria predicts the location of the edges like how much distance they are away from mean by which we can find out whether we can reach from source to destination that is whether path exists from source to destination.**

**Cons:**

**Stable criteria has a disadvantage that It does not calculate the edge cost fro reaching source to destination it only predicts how much distance the nodes are away from the mean value.**

**Triple Pessimist:**

**Pros:**

**When trying to reach from source to destination using triple pessimist approach there will be much more and more risks between the adjacent path when compared to pessimist and double pessimist. Pessimist tries to reach the destination node by considering all the risks.**

**Cons:**

**Triple pessimist approach has the disadvantage that it takes more time that pessimist and double pessimist to reach from source to destination and has less accuracy when compared to other methods.**

1. **EPOLOGUE:**

* **Unforeseen Circumstance:**

According to the given problem, when planning a car-trip, say from a to b, most people would want to find the shortest route. To find the shortest path, here we used the provided parameters, α and β whose interpretation depends on the type of distributions and the different criterions.

In this project, we haven’t faced any unforeseen situations that could really be defined as a key in implementation, but this is an example where we got slight confusion, while implementing the Dijkstra’s shortest path algorithm on the graph what if the edges of the graph had negative weights. But after that we got to know that to apply Dijkstra’s algorithm to any graph, that must be connected and weights on the edges should be non-negative.

* **Lessons Learnt:**

By doing this group project, we have learned the topics related to this project in deeper levels we equate with understanding. We have started working on the project from the day it was given to us.

We have tried with different data structures to get the better results, and in that process we got sufficient knowledge on those different data structures. So, we learnt to go working on innovative things. It was very helpful in showing how everything works together.

* **One more Opportunity:**

If we have a chance to do this project again with extra necessities, we would probably have done something more successfully to enhance the results in terms of time and space complexities. Other than that, we would go a step ahead and implement more new features that would be helpful. Also we would come up with a better method than this and try to make it more effectively, to make our result improved than preceding outputs.

* **Lessons learnt and for the further students:**

This project helped a lot in thinking in all possibilities for a particular problem, and to think of unthinkable situations while analyzing the problem. My advice to anyone starting this project is to start early and work on it piece by piece. Working on it incrementally will allow anyone to see simple mistakes in finding the shortest path.

**7. APPENDIX:**

* **Program Listings:**

As we previously stated in Experimental Implementation and Data Collection and Interpretation of results parts, we are using Java as programming language and it runs on Java compiler i.e. Java Virtual Machine. We used Windows Operating System, with intel processor with Eclipse Jee Neon as Java IDE.

* **Below are the files of the implemented programs to find the Shortest path as per the given approach:**

**AdjacencyMatrix.java:**

This program contains the functions to calculate the adjacency matrix for all the different criteria namely mean, optimist, pessimist, double pessimist, stable and Triple Pessimist. This program returns the adjacency matrix for the corresponding criteria.



AdjecentyMatrix.java

Dijkstra.java:

This program take adjacency matrix, start and end nodes as inputs and uses apriority queue to store the edge nodes and pops the edge with minimum weight and finds its adjacent nodes. Finally it prints the shortest path from source to destination and its hop length, mean, mean – standard Deviation, mean + standard Deviation, mean + 2\*standard Deviation.



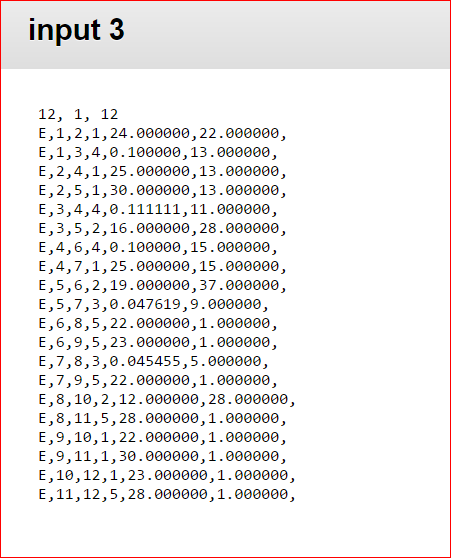
Dijkstra.java

* **Outputs:**

After implementing the program in java, we got the following are the outputs for the given input file with 12 nodes.

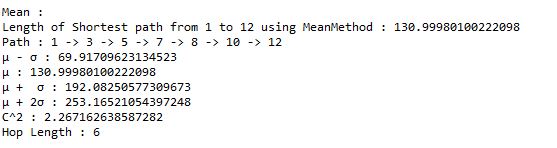
* **Below are the Input and Output screenshots of the implemented programs to find the Shortest path as per the given approach:**

**Input3 Inputs:**

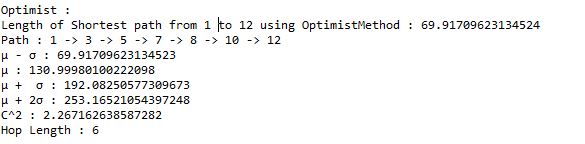


**Input3 Outputs:**

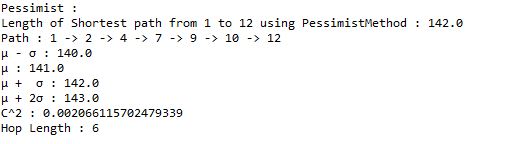
**Mean:**



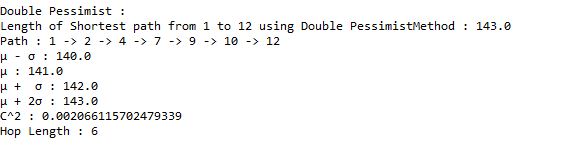
**Optimist:**



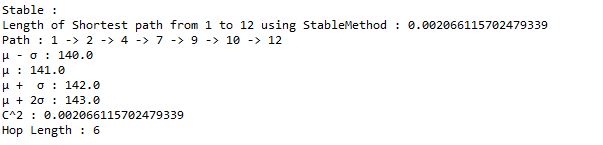
**Pessimist:**



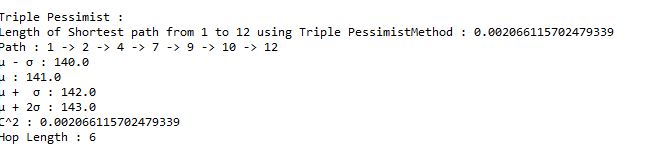
**Double Pessimist:**



**Stable:**

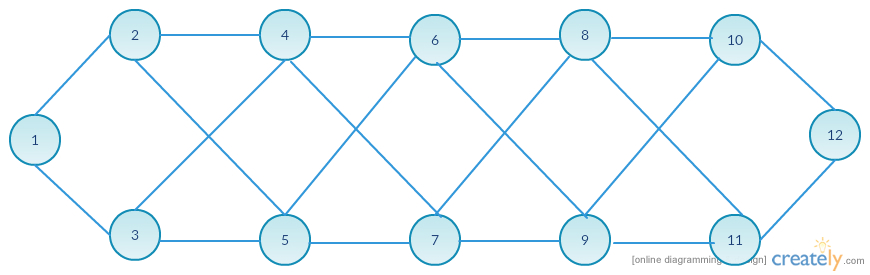


Triple Pessimist:

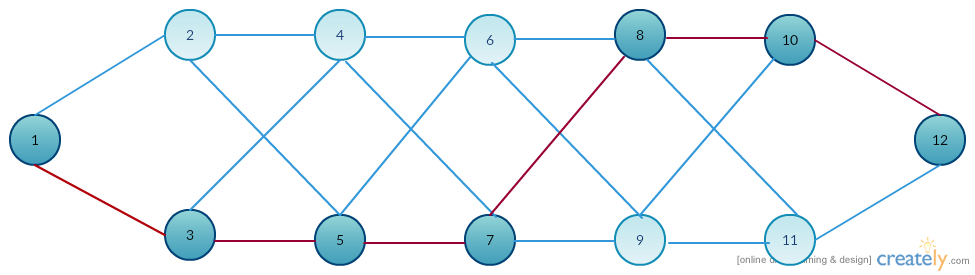


**Shortest path Graph Representation for the different Criterions:**

**Original graph:**

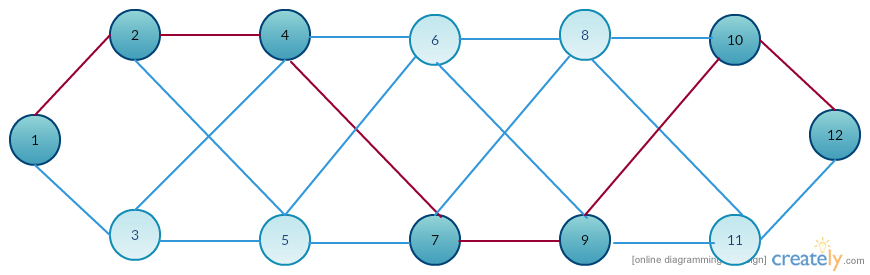


**Shortest Path Graph for Mean and Optimistic criterion:**



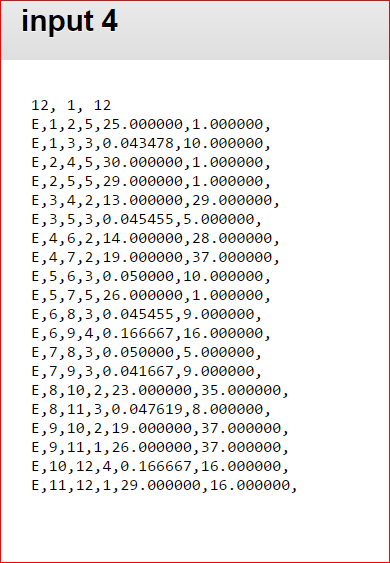
Path: 1 -> 3 -> 5 -> 7-> 8 -> 10 -> 12

**Shortest Path Graph for Pessimist, Double Pessimist, Stable and Triple Pessimist criterion:**



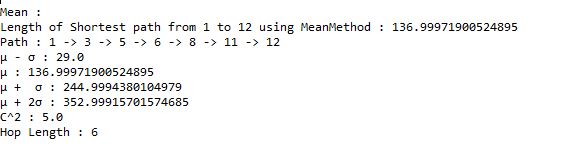
**Path:** 1->2->4->7->9->10->12

**Input4 Inputs:**

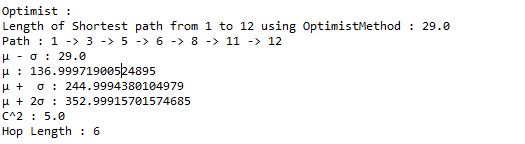


**Input4 Outputs:**

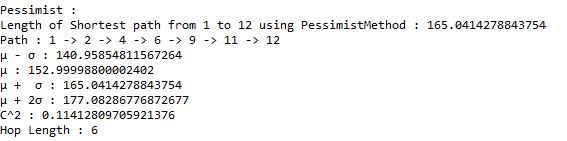
**Mean:**

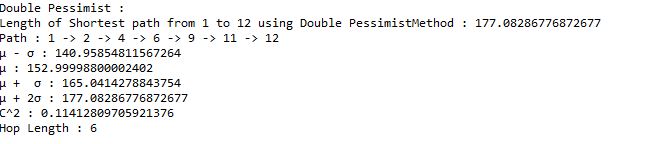


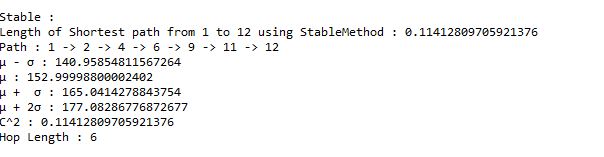
**Optimist:**



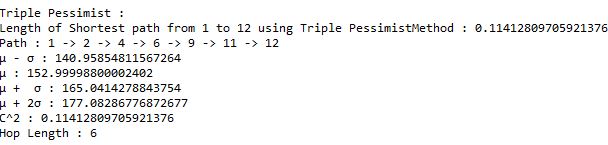
**Pessimist:**



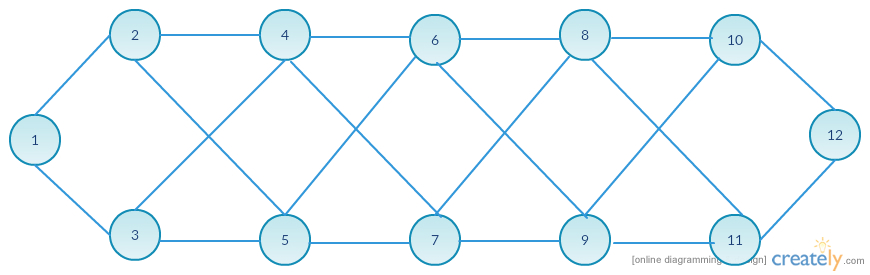
**Double Pessimist:** 

**Stable:**

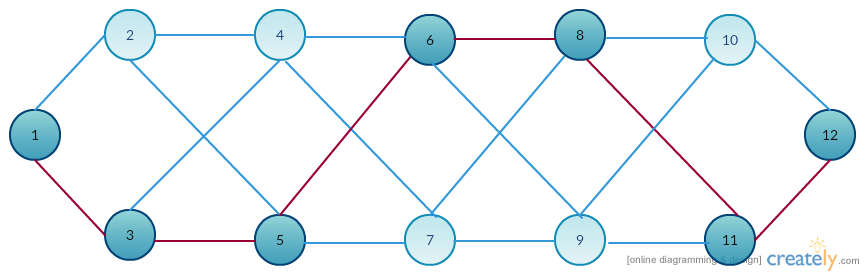
**Triple Pessimist:**



**Input4\_Shortest path Graph Representation for the different Criterions:**

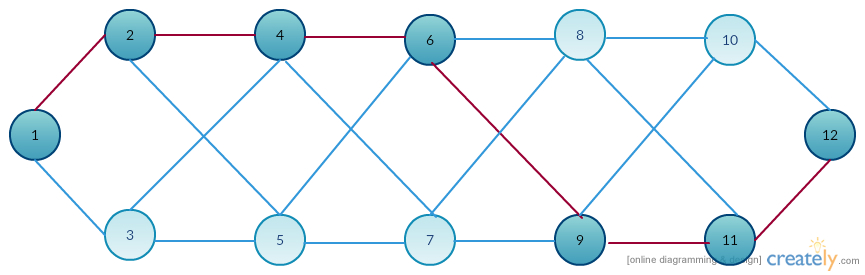
**Original Graph:**

**Shortest Path Graph for Mean and Optimistic criterion:**



Path:1->3->5->6->8->11->12

**Shortest Path Graph for Pessimist, Double Pessimist, Stable and Triple Pessimist criterion:**



Path: 1->2->4->6->9->11->12