1.Create a network topology with at least 20 nodes. Write the following modules for:

-Node initialization

-Creating distance matrix

-Detecting neighbours

-Selecting working nodes

-Building link cost packets

-Updating routing tables

-Tracing the shortest path from source to destination.

Using the modules mentioned above, implement the shortest path routing algorithm. Find the end-to-end delay for this algorithm considering the same pair of source and destination.

PROGRAM CODE:

```
#include<bits/stdc++.h>

using namespace std;

int dist_mat[100][100];

map<int,int>working;
```

```
struct Node{
int par_node, dist;
};
int minDist(Node Arr[], bool set[], int N) {
int mn = INT_MAX, min_idx;
for(int i = 1; i <=N; i++) {
  if(working[i])
  {
if(set[i] == false && Arr[i].dist <= mn)
mn = Arr[i].dist, min_idx = i;
  }
}
return min_idx;
}
Node* nodeInitialisation(int& N) {
cout << "Enter the no of nodes for the network: ";
cin >> N;
Node* Arr;
Arr = (Node*)malloc((N+1)*sizeof(Node));
return Arr;
}
void createDistanceMatrix(int N){
```

```
for(int i=1;i<=N;i++)
{
for(int j=1;j<=i;j++)
{ if(i==j)
{
dist_mat[i][j]=0;
continue;
}


int wt=rand()%100;
if(wt==0)
{
dist_mat[i][j]=-1;
dist_mat[j][i]=-1;
continue;
}
dist_mat[i][j]=wt;
dist_mat[j][i]=wt;
}
}
for(int i=1;i<=N;i++)
{
 for(int j=1;j<=N;j++)
 {
 cout<<dist_mat[i][j]<<" ";
 }
 cout<<endl;
 }
 cout<<endl;
}
void select_work_node(int N)
{
 cout<<"Probability is 50%\n";
 cout<<"No. of working node is "<<N/2<<endl;

 int x=1;
 for(int i=1;i<=N/2;i++)
 {
 working[x]=1;;
 x+=2;
 }
 cout<<"Working node distance matrix is \n";
 for(int i=0;i<=N;i++)
 { if(i==0)
 {
```

```
 for(int j=0;j<=N;j++)
 { if(j==0)
 cout<<j<<" ";
 if(working[j])
 cout<<j<<" ";
 }
 cout<<endl;
 continue;
 }

 if(working[i])
 { cout<<i<<" ";
 for(int j=1;j<=N;j++)
 {
 if(!working[j])
 continue;
 cout<<dist_mat[i][j]<<" ";
 }
 cout<<endl;
 }
 }
 cout<<endl;
}
```

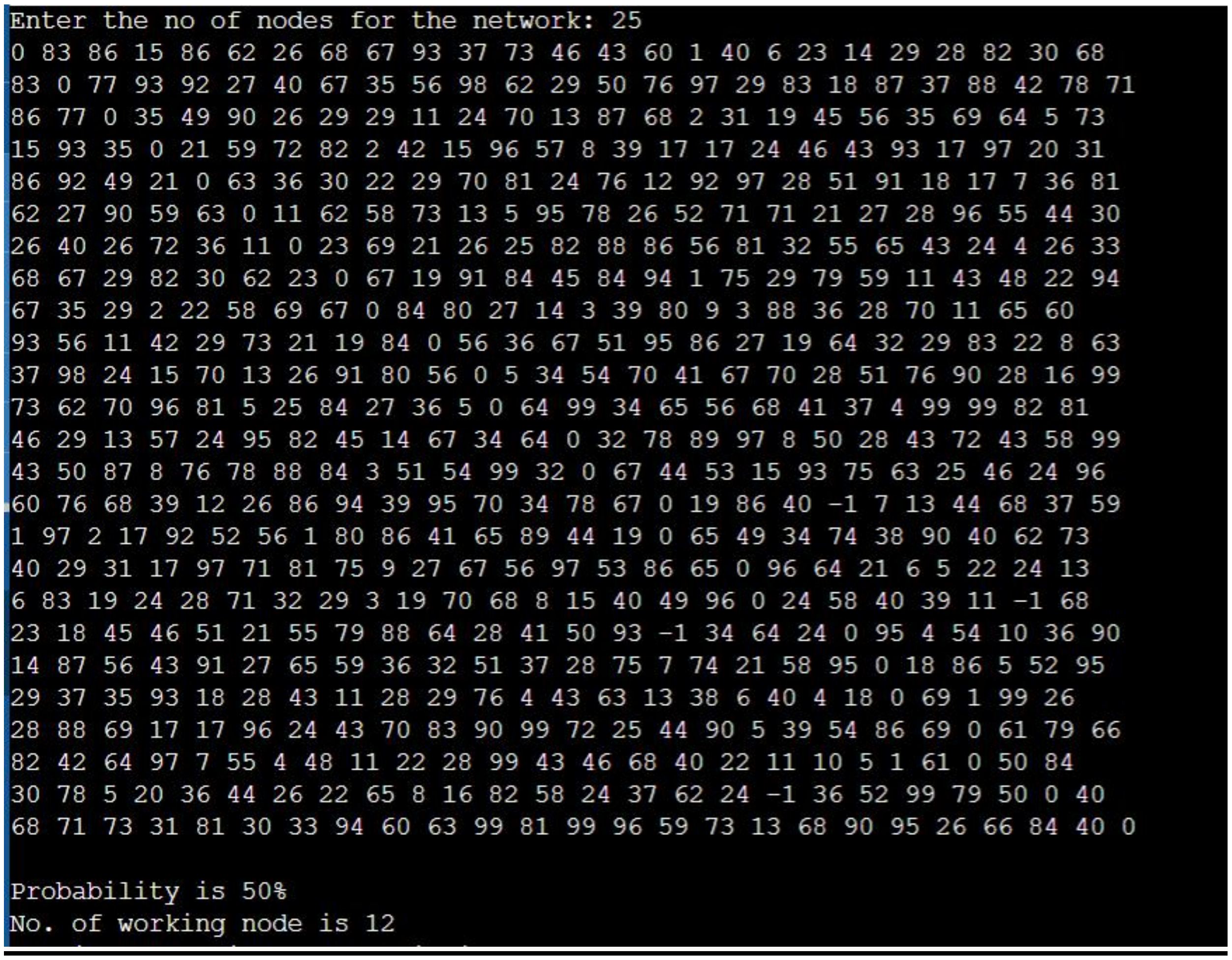
```
}

void Routing(Node Arr[], int dist_mat[100][100], int N, int
src) {
bool set[N+1];
for(int i = 1; i <=N; i++) {
Arr[i].dist = INT_MAX;
Arr[i].par_node = -1;
set[i] = false;
}
Arr[src].dist = 0;
for(int ct = 1; ct < N; ct++) {
 if(working[ct])
 {
int u = minDist(Arr, set, N);
set[u] = true;
for(int v = 1; v <=N; v++) {
 if(working[v])
 {
if(!set[v] && dist_mat[u][v]!=-1||!dist_mat[u]
[v]!=0 && Arr[u].dist != INT_MAX && (Arr[u].dist +
dist_mat[u][v] < Arr[v].dist)) {
Arr[v].dist = Arr[u].dist + dist_mat[u][v];
Arr[v].par_node = u;
}
```

```
 }
}
 }
}
}
void shortestPathFromSrcToDest(Node Arr[], int Dest) {
vector<int> path;
int i = Dest;
while(i != -1) {
path.push_back(i);
i = Arr[i].par_node;
}
path.push_back(1);
reverse(path.begin(), path.end());
for(i = 1; i < path.size() - 1; i++) {
cout << path[i] << " --> ";
}
cout << path[i] << endl;
}
int main() {
memset(dist_mat, 0, 100*100*sizeof(int));
Node* Arr;
int size;
```

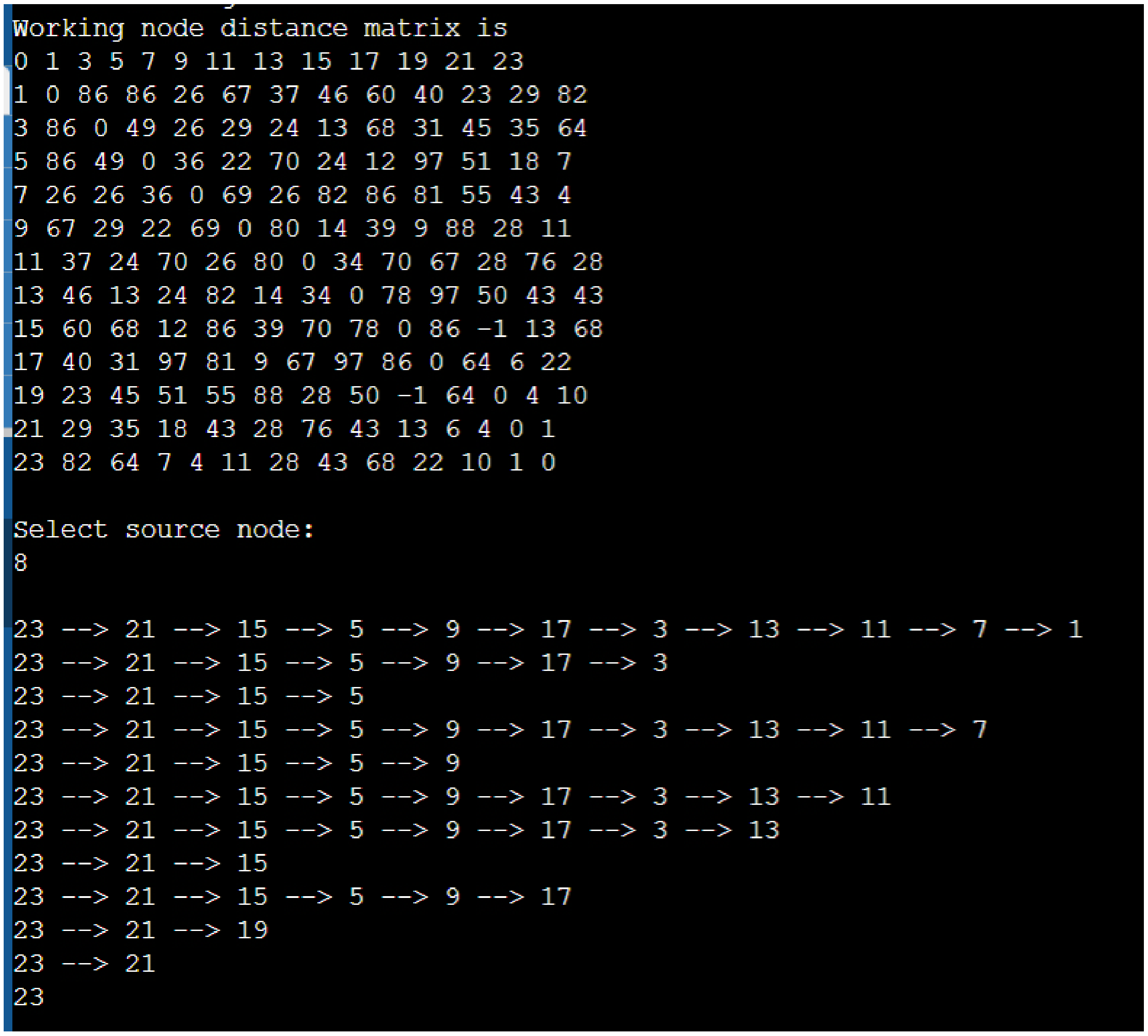
```
Arr = nodeInitialisation(size);
createDistanceMatrix(size);
 select_work_node( size);
 cout<<"Select source node: \n";
 int src;
 cin>>src;
 
Routing(Arr, dist_mat, size, src);
 cout<<endl;
for(int i = 1; i <=size; i++)
 { if(working[i])
shortestPathFromSrcToDest(Arr, i);
 }
return 0;
}
```

OUTPUT:



```
Enter the no of nodes for the network: 25
0 83 86 15 86 62 26 68 67 93 37 73 46 43 60 1 40 6 23 14 29 28 82 30 68
83 0 77 93 92 27 40 67 35 56 98 62 29 50 76 97 29 83 18 87 37 88 42 78 71
86 77 0 35 49 90 26 29 29 11 24 70 13 87 68 2 31 19 45 56 35 69 64 5 73
15 93 35 0 21 59 72 82 2 42 15 96 57 8 39 17 17 24 46 43 93 17 97 20 31
86 92 49 21 0 63 36 30 22 29 70 81 24 76 12 92 97 28 51 91 18 17 7 36 81
62 27 90 59 63 0 11 62 58 73 13 5 95 78 26 52 71 71 21 27 28 96 55 44 30
26 40 26 72 36 11 0 23 69 21 26 25 82 88 86 56 81 32 55 65 43 24 4 26 33
68 67 29 82 30 62 23 0 67 19 91 84 45 84 94 1 75 29 79 59 11 43 48 22 94
67 35 29 2 22 58 69 67 0 84 80 27 14 3 39 80 9 3 88 36 28 70 11 65 60
93 56 11 42 29 73 21 19 84 0 56 36 67 51 95 86 27 19 64 32 29 83 22 8 63
37 98 24 15 70 13 26 91 80 56 0 5 34 54 70 41 67 70 28 51 76 90 28 16 99
73 62 70 96 81 5 25 84 27 36 5 0 64 99 34 65 56 68 41 37 4 99 99 82 81
46 29 13 57 24 95 82 45 14 67 34 64 0 32 78 89 97 8 50 28 43 72 43 58 99
43 50 87 8 76 78 88 84 3 51 54 99 32 0 67 44 53 15 93 75 63 25 46 24 96
60 76 68 39 12 26 86 94 39 95 70 34 78 67 0 19 86 40 -1 7 13 44 68 37 59
1 97 2 17 92 52 56 1 80 86 41 65 89 44 19 0 65 49 34 74 38 90 40 62 73
40 29 31 17 97 71 81 75 9 27 67 56 97 53 86 65 0 96 64 21 6 5 22 24 13
6 83 19 24 28 71 32 29 3 19 70 68 8 15 40 49 96 0 24 58 40 39 11 -1 68
23 18 45 46 51 21 55 79 88 64 28 41 50 93 -1 34 64 24 0 95 4 54 10 36 90
14 87 56 43 91 27 65 59 36 32 51 37 28 75 7 74 21 58 95 0 18 86 5 52 95
29 37 35 93 18 28 43 11 28 29 76 4 43 63 13 38 6 40 4 18 0 69 1 99 26
28 88 69 17 17 96 24 43 70 83 90 99 72 25 44 90 5 39 54 86 69 0 61 79 66
82 42 64 97 7 55 4 48 11 22 28 99 43 46 68 40 22 11 10 5 1 61 0 50 84
30 78 5 20 36 44 26 22 65 8 16 82 58 24 37 62 24 -1 36 52 99 79 50 0 40
68 71 73 31 81 30 33 94 60 63 99 81 99 96 59 73 13 68 90 95 26 66 84 40 0

Probability is 50%
No. of working node is 12
```



```
Working node distance matrix is
0 1 3 5 7 9 11 13 15 17 19 21 23
1 0 86 86 26 67 37 46 60 40 23 29 82
3 86 0 49 26 29 24 13 68 31 45 35 64
5 86 49 0 36 22 70 24 12 97 51 18 7
7 26 26 36 0 69 26 82 86 81 55 43 4
9 67 29 22 69 0 80 14 39 9 88 28 11
11 37 24 70 26 80 0 34 70 67 28 76 28
13 46 13 24 82 14 34 0 78 97 50 43 43
15 60 68 12 86 39 70 78 0 86 -1 13 68
17 40 31 97 81 9 67 97 86 0 64 6 22
19 23 45 51 55 88 28 50 -1 64 0 4 10
21 29 35 18 43 28 76 43 13 6 4 0 1
23 82 64 7 4 11 28 43 68 22 10 1 0

Select source node:
8

23 --> 21 --> 15 --> 5 --> 9 --> 17 --> 3 --> 13 --> 11 --> 7 --> 1
23 --> 21 --> 15 --> 5 --> 9 --> 17 --> 3
23 --> 21 --> 15 --> 5
23 --> 21 --> 15 --> 5 --> 9 --> 17 --> 3 --> 13 --> 11 --> 7
23 --> 21 --> 15 --> 5 --> 9
23 --> 21 --> 15 --> 5 --> 9 --> 17 --> 3 --> 13 --> 11
23 --> 21 --> 15 --> 5 --> 9 --> 17 --> 3 --> 13
23 --> 21 --> 15
23 --> 21 --> 15 --> 5 --> 9 --> 17
23 --> 21 --> 19
23 --> 21
23
```