Algorithm:

Step 1: Start.

Step 2: Input the length of both the arrays.

Step 3: Input the array elements from user.

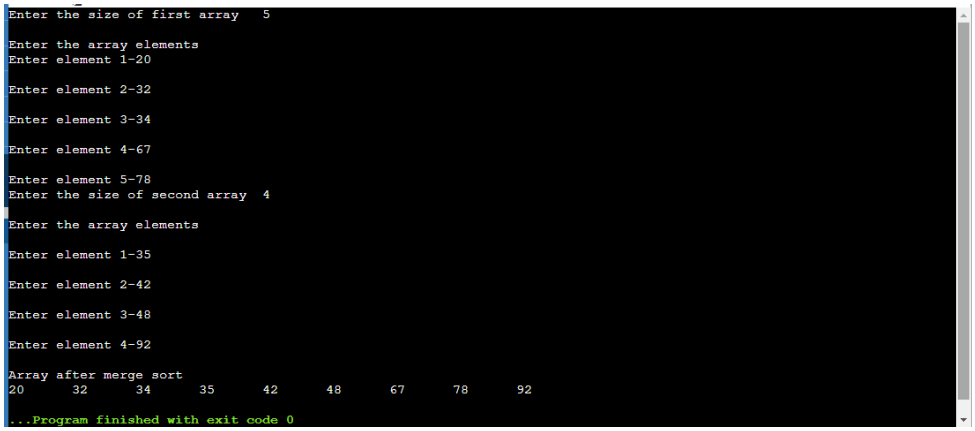
Step 4: If the first array has the smaller element (i.e. array1[i] < array2[j]), fill the third array with the first array's element (i.e. array3[k++] = array1[i++]), otherwise with the second array's element (i.e. array3[k++] = array2[j++]).

Step 5: Find the size of the provided arrays and put them together, then declare a third array of the same size.

Step 6: Print the third array.

Step 7: Stop.

Output:


Enter the size of first array 5
Enter the array elements
Enter element 1-20
Enter element 2-32
Enter element 3-34
Enter element 4-67
Enter element 5-78
Enter the size of second array 4
Enter the array elements
Enter element 1-35
Enter element 2-42
Enter element 3-48
Enter element 4-92
Array after merge sort
20 32 34 35 42 48 67 78 92
...Program finished with exit code 0

Algorithm:

Step 1: Start.

Step 2: Initialize variable to set front and rear as -1

Step 3: Write a function for enqueue operation(insertion).

i)If (rear+1)%max = front

Write " overflow "

Goto step 6

[end of if]

ii)If front = -1 and rear = -1

Set front = rear = 0

Else if rear = max - 1 and front ! = 0

Set rear = 0

Else

Set rear = (rear + 1) % max

[end of if]

iii)Set queue[rear] = val

Step 4: Write a function for dequeue operation(deletion).

i) if front = -1

Write " underflow "

Goto step 6.

[end of if]

ii) set val = queue[front]

iii) if front = rear

Set front = rear = -1

Else

If front = max -1

Set front = 0

Else

Set front = front + 1

[end of if]

[end of if]

Step 5: Write a function to display elements in queue.

i)**if**(front==-1 && rear==-1)

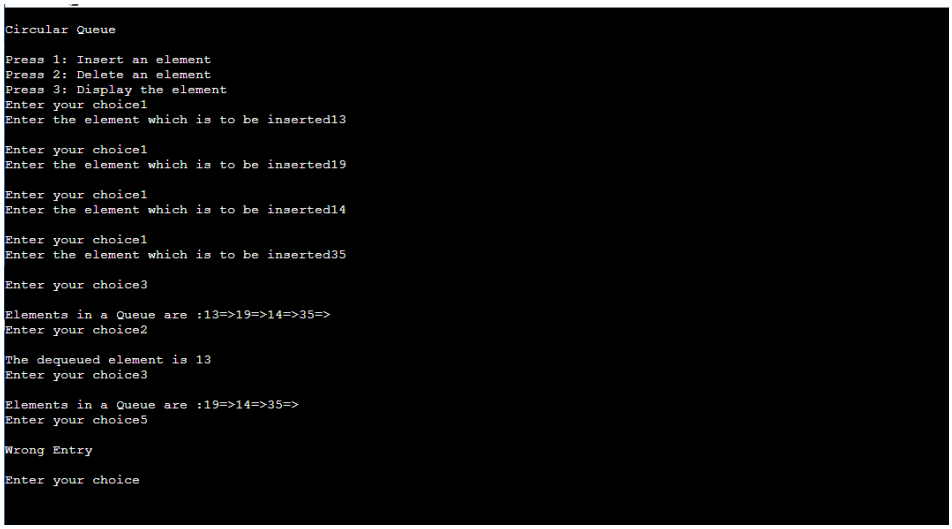
"Queue is empty"

ii)Print elements till i<=rear and update i=(i+1)%max.

Step 6: Call the functions according to the choice.

Step 7: Stop.

Output:


Circular Queue
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice1
Enter the element which is to be inserted13
Enter your choice1
Enter the element which is to be inserted19
Enter your choice1
Enter the element which is to be inserted14
Enter your choice1
Enter the element which is to be inserted35
Enter your choice3
Elements in a Queue are :13=>19=>14=>35=>
Enter your choice2
The dequeued element is 13
Enter your choice3
Elements in a Queue are :19=>14=>35=>
Enter your choice5
Wrong Entry
Enter your choice

Algorithm:

Step 1: Start.

Step 2: Write function for insertion at the front.

i) if head = null

write overflow

go to step

[end of if]

ii) Create a new node

iii) new->data=x

iv) new->link=head

v) head=new

Step 3: Write function for insertion at the end.

i) if head = null

write overflow

go to step 1

[end of if]

ii) Create a new node

iii) new->data=x

iv) new ->link=NULL

v) if head=NULL then

head=new

vi) Else

ptr=head

while(ptr->link!=NULL) do

ptr=ptr->link

ptr->link=new

Step 4: Write function for insertion after a specified node.

i)if head =NULL then

Print “Overflow”

ii) Else

ptr=head

while(ptr->data!=key and ptr->link!=NULL) do

ptr=ptr->link

If ptr->data!=key then

Print “Search failed”

Else

Create a new node

new->data=x

new->link=ptr->link

ptr->link=new

Step 5: Write function for deletion from front.

i)if head=NULL then

Print “List is empty”

ii) Else

temp=head

head=head->link

free(temp)

Step 6: Write function for deletion from end.

i)if head=NULL then

Print “List is empty”

ii) Else if head->link=NULL then

temp=head

head=NULL

free(temp)

iii)Else

prev=head

curr=head->link

while curr->link!=NULL do

prev=curr

curr= curr->link

prev->link=NULL

free(curr)

Step 7: Write function for delete specified node.

i)if head=NULL then

Print “Lis is empty”

ii) Else if head->data= key then

temp=head

head=head->link

free(temp)

iii) Else

prev=head

curr=head

while curr->data!=key and curr->link!=NULL do

prev=curr

curr=curr->link

if curr->data!=key then

Print “Search key not found”

Else

prev->link=curr->link

free(curr)

Step 8: Write function to search a node.

i)if head=NULL

Print “List id empty”

ii)Else

ptr=head

while ptr->data!=key and ptr->link!=NULL then

ptr=ptr->link

If ptr->data=key then

Print “Search data found”

Else

Print “Search data not found”

Step 9: Write function to traverse or display nodes.

i)if head=NULL

Print “List id empty”

ii)Else

ptr=head

while ptr!=NULL do

Print ptr->data

Ptr=ptr->link

Step 10: Call all the function according to the choice.

Step 11: Stop.

Output:

```
Singly Linked List Operations

1.Insert Front
2.Insert  end
3.Insert after a node
4.Delete from front
5.Delete from end
6.Deletion  a node
7.Display
8.Search

Enter the choice1

Enter the item to be inserted25

25 Inserted into the list
Enter the choice2

Enter the item to be inserted28

28 Inserted into the list
Enter the choice3

Enter the value after which the new node to be inserted25

Enter the item to be inserted27

27 Inserted after25
Enter the choice2

Enter the item to be inserted46

46 Inserted into the list
Enter the choice2
```

```
Enter the item to be inserted28

28 Inserted into the list
Enter the choice3

Enter the value after which the new node to be inserted25

Enter the item to be inserted27

27 Inserted after25
Enter the choice2

Enter the item to be inserted46

46 Inserted into the list
Enter the choice2

Enter the item to be inserted49

49 Inserted into the list
Enter the choice7

Elements in list are:    25      27      28      46      49
Enter the choice8
Enter the element to be searched 28

28The value found at position 3
Enter the choice4
25 Deleted element is
Enter the choice5
49 Deleted element is
Enter the choice6
Enter the value tobe deleted 28
The deleted node is 28
```

Algorithm:

Step 1: Start.

Step 2: push() operation

i) Create a new node

ii) new->data=item

iii) new->link=top

iv) top=new

Step 3: pop() operation

i)If top=NULL then

Print “Stack is empty”

ii)Else

temp=top

Print “Popped item is ” top->data

top=top->link

free(temp)

Step 4: Function to display stack items.

i)if top=NULL then

Print “Stack is empty”

ii)Else

ptr=top

while ptr!=NULL do

Print ptr->data

ptr=ptr->link

Step 5: Call all the function according to the choice.

Step 6: Stop.

Output:

```
-----Stack Using Linked List----

Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       1
Enter the to be inserted 27

Insertion successful 27
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       1
Enter the to be inserted 28

Insertion successful 28
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       1
Enter the to be inserted 26

Insertion successful 26
Menu
```

```
Insertion successful 26
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       1
Enter the to be inserted 14

Insertion successful 14
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       2

Deleted element 14
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       2

Deleted element 26
Menu

1.Push
2.Pop
3.Display
```

```
3.Display
4.Exit
Enter your choice       2

Deleted element 14
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       2

Deleted element 26
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       3

Elements in stack are
28----> 27---->NULL
Menu

1.Push
2.Pop
3.Display
4.Exit
Enter your choice       4


...Program finished with exit code 0
Press ENTER to exit console.
```

Algorithm:

Step 1: Start.

Step 2: Write a function to display nodes.

i)ptr=head

ii)while ptr!=NULL do

Print ptr->data

ptr=ptr->link

Step 3: Write a function for insertion at the front.

i)Create a ne node.

ii)new->data=x

iii)new->Llink=new->Rlink=NULL

iv)if head=NULL, then

head = new

v)Else

new->Rlink=head

new->Llink=new

head=new

Step 4: Write a function for insertion at the end.

i)Create a new node

ii)new->data=x

iii)new->Rlink=new->Llink=NULL

iv)if head=NULL then

head= new

v)Else

ptr= head

while(ptr->Rlink!=NULL) do

ptr=ptr->Rlink

ptr->Rlink=new

new->Llink=ptr

Step 5: Write a function for insertion after specified node.

i)if head=NULL, then

Print “Overflow”

ii)Else

ptr=head

while ptr->data!=key and ptr->Rlink!NULL do

ptr=ptr->Rlink

if ptr->data!=key then

Print “Search data not found “

Step 6: Write a function to delete from front.

i)if head=NULL then

Print “List is empty”

ii) Else if head->Rlink=NULL, then

temp=head

head=NULL

free(temp)

iii)Else

head=head->Rlink

free(head->Llink)

head->Llink=NULL

Step 7: Write a function to delete from end.

i)if head=NULL then

Print “List is empty”

ii) Else if head->Rlink=NULL then

temp=head

head=NULL

free(temp)

iii)Else

ptr=head

while ptr->Rlink!=NULL do

ptr=ptr->Rlink

ptr->Rlink->Llink=NULL

free(ptr)

Step 8: Write a function to delete from end.

i)if head==NULL then

Print “List is empty”

ii)Else if head->Rlink->NULL then

if head->data=key then

temp=head

head=NULL

free(temp)

Else

Print “Search data not found”

iii)Else if head->data=key then

head=head->Rlink

free(head->Llink)

head->Llink=NULL

iv)Else

ptr=head

while ptr->data!=key and ptr->Rlink!=NULL do

ptr=ptr->Rlink

if ptr->data!=key then

Print “Search data not found”

Else

ptr->Llink->Rlink=ptr->Rlink

if ptr->Rlink!=NULL then

ptr->Rlink->Llink=ptr->Llink

free(ptr)

Step 9: Write a function to search node.

i)if head==NULL then

Print “List is empty”

ii)Else

ptr=head

while ptr->data!=key and ptr->Rlink!=NULL do

ptr=ptr->Rlink

if ptr->data=key then

Print “Search data found”

Else

Print “Search data not found”

Step 10: Call all the functions according to the choice.

Step 11: Stop.

Output:

```
Doubly Linked List

1.Insert in begining
2.Insert at last
3.Insert at any random location
4.Delete from Beginning
5.Delete from last
6.Delete the node after the given data
7.Search
8.Show
9.Exit

*********Main Menu*********

Enter your choice?
1

Enter Item value11

Node inserted

Enter your choice?
1

Enter Item value12

Node inserted

Enter your choice?
2

Enter value15

node inserted
```

```
Enter your choice?
2

Enter value15

node inserted

Enter your choice?
3
Enter the location2
Enter value17

node inserted

Enter your choice?
2

Enter value19

node inserted

Enter your choice?
8

 printing values...
12
11
15
17
19

Enter your choice?
7
```

```
 printing values...
12
11
15
17
19

Enter your choice?
4

node deleted

Enter your choice?
5

node deleted

Enter your choice?
6

 Enter the data after which the node is to be deleted : 11

Can't delete

Enter your choice?
6

 Enter the data after which the node is to be deleted : 15
```

Algorithm:

Step 1: Start.

Step 2: Choose any one node randomly, to start traversing.

Step 3: Visit its adjacent unvisited node.

Step 4: Mark it as visited in the array and display it.

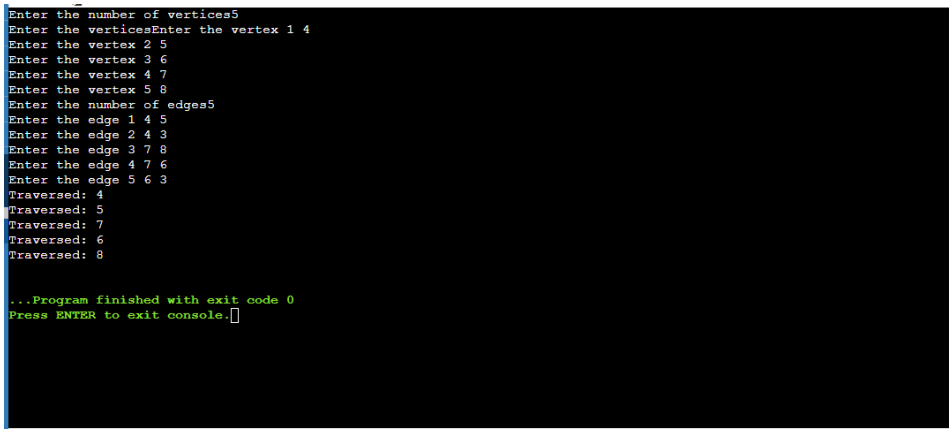
Step 5: Insert the visited node into the queue.

Step 6: If there is no adjacent node, remove the first node from the queue.

Step 7: Repeat the above steps until the queue is empty.

Step 8: Stop

Output:


Enter the number of vertices5
Enter the verticesEnter the vertex 1 4
Enter the vertex 2 5
Enter the vertex 3 6
Enter the vertex 4 7
Enter the vertex 5 8
Enter the number of edges5
Enter the edge 1 4 5
Enter the edge 2 4 3
Enter the edge 3 7 8
Enter the edge 4 7 6
Enter the edge 5 6 3
Traversed: 4
Traversed: 5
Traversed: 7
Traversed: 6
Traversed: 8
...Program finished with exit code 0
Press ENTER to exit console.

Algorithm:

Step 1: Start.

Step 2: Start by putting any one of the graph's vertices on top of a stack.

Step 3: Take the top item of the stack and add it to the visited list.

Step 4: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.

Step 5: Keep repeating steps 2 and 3 until the stack is empty.

Step 6: Stop.

Output:


Enter the number of vertices5
Enter the vertices
Enter the vertex 1 4
Enter the vertex 2 5
Enter the vertex 3 6
Enter the vertex 4 7
Enter the vertex 5 8
Enter the number of edges5
Enter the edge 1 4 6
Enter the edge 2 4 5
Enter the edge 3 5 7
Enter the edge 4 6 8
Enter the edge 5 5 6
Traversed: 4
Traversed: 6
Traversed: 8
Traversed: 5
Traversed: 7
...Program finished with exit code 0
Press ENTER to exit console.

Algorithm:

Step 1: Start.

Step 2: Store each vertex's In-Degree in an array D

Step 3: Initialize queue with all "in-degree=0" vertices

Step 4: While there are vertices remaining in the queue:

i) Dequeue and output a vertex

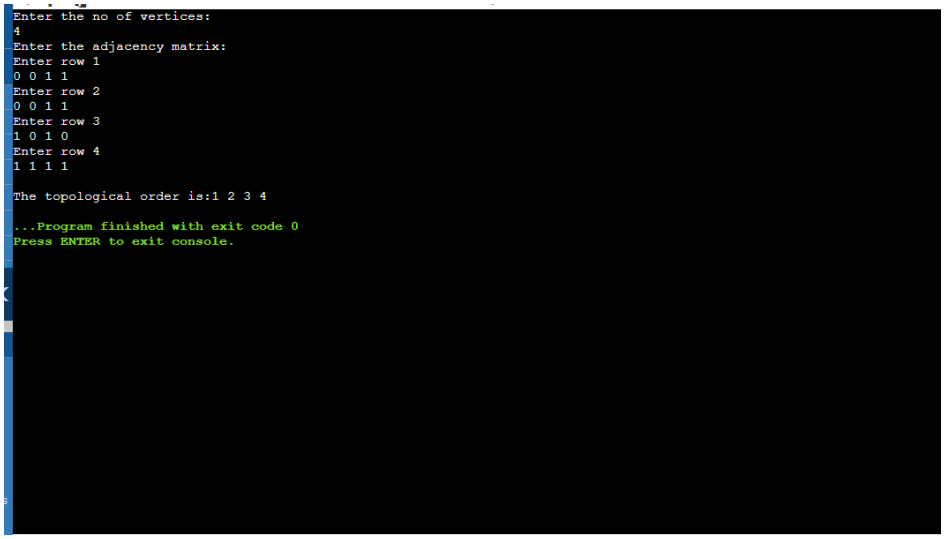
ii) Reduce In-Degree of all vertices adjacent to it by 1

iii) Enqueue any of these vertices whose In-Degree became zero

Step 5: If all vertices are output then success, otherwise there is a cycle.

Step 6: Stop

Output:


Enter the no of vertices:
4
Enter the adjacency matrix:
Enter row 1
0 0 1 1
Enter row 2
0 0 1 1
Enter row 3
1 0 1 0
Enter row 4
1 1 1 1
The topological order is:1 2 3 4
...Program finished with exit code 0
Press ENTER to exit console.

Algorithm:

Step 1: Start.

Step 2: Begin

Step 3: Create edge list of given graphs, with their weights.

Step 4: Draw all nodes to create skeleton for spanning tree.

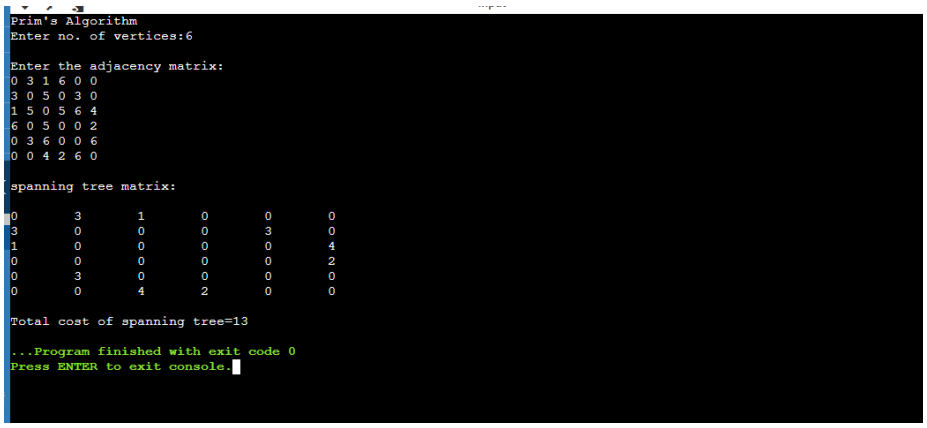
Step 5: Select an edge with lowest weight and add it to skeleton and delete edge from edge list.

Step 6: Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.

Step 7: Repeat step 5 until n-1 edges are added.

Step 8: Stop.

Output


Prim's Algorithm
Enter no. of vertices:6
Enter the adjacency matrix:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0
spanning tree matrix:
0 3 1 0 0 0
3 0 0 0 3 0
1 0 0 0 0 4
0 0 0 0 0 2
0 3 0 0 0 0
0 0 4 2 0 0
Total cost of spanning tree=13
...Program finished with exit code 0
Press ENTER to exit console.

Algorithm:

Step 1: Start.

Step 2: Begin

Step 3: Create the edge list of given graphs, with their weights.

Step 4: Sort the edge list according to their weights in ascending order.

Step 5: Draw all the nodes to create skeleton for spanning tree.

Step 6: Pick up the edge at the top of the edge list (i.e. edge with minimum weight).

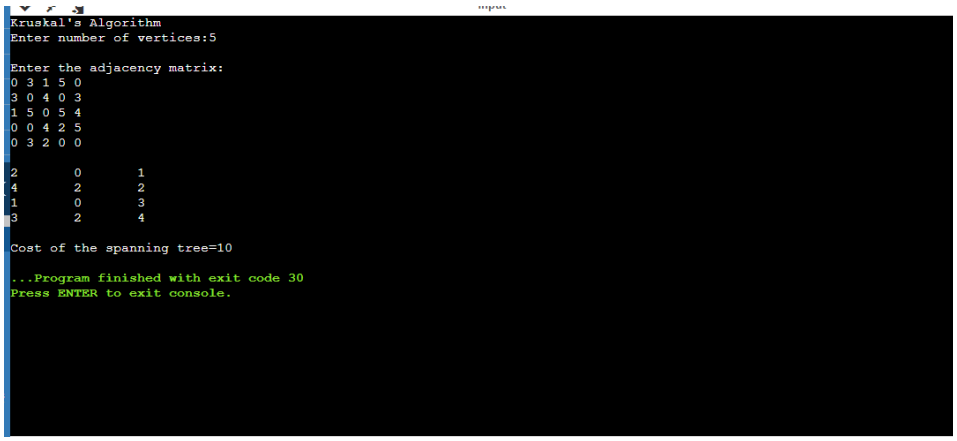
Step 7: Remove this edge from the edge list.

Step 8: Connect the vertices in the skeleton with given edge. If by connecting the vertices, a cycle is created in the skeleton, then discard this edge.

Step 9: Repeat steps 5 to 7, until n-1 edges are added or list of edges is over.

Step 10: Stop.

Output


Kruskal's Algorithm
Enter number of vertices:5
Enter the adjacency matrix:
0 3 1 5 0
3 0 4 0 3
1 5 0 5 4
0 0 4 2 5
0 3 2 0 0
2 0 1
4 2 2
1 0 3
3 2 4
Cost of the spanning tree=10
...Program finished with exit code 30
Press ENTER to exit console.

Algorithm:

Step 1: Start.

Step2: Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.

Step 3: Array visited[ ] is initialized to zero.

```
for(i=0;i<n;i++)
        visited[i]=0;
```

Step 4: If the vertex 0 is the source vertex then visited[0] is marked as 1.

Step 5: Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.

```
for(i=1;i<n;i++)
        distance[i]=cost[0][i];
```

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

Step 6: for(i=1;i<n;i++)

i) Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

ii) Recalculate the shortest distance of remaining vertices from the source.

iii)Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

```
if(visited[v]==0)
        distance[v]=min(distance[v],
        distance[w]+cost[w][v])
```

Step 7: Stop

Output:

```
Enter no. of vertices:5

Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0

...Program finished with exit code 0
Press ENTER to exit console.
```