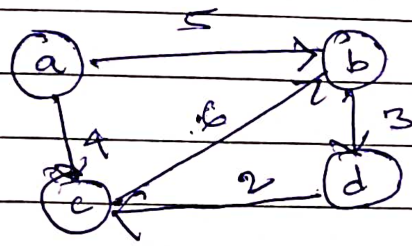

Bellman

```
#include<stdio.h >
 # include <stdlib.h>
  # int  Bellman-ford ( int  G[20][20], int v , int E,
                       int  edge[20][2] )
  {
  int  i, u,v, k, distance [20] ,parent [20], S, flag = 1;
   for (i=0 ; i<V; i++ )
                distance[i] = 1000 ; parent [i] = -1 ;
                 printf (" Enter  source: ") ;
                  scanf ( "%d" , &S) ;
                  distance [S-1] =0;
         for (i=0; i<V-1 ,i++)
          {
                for (k=0; k < E; k++)
                 {
                      u =  edge [k] [0] , v=edge [k][1];
                      if (distance[u] + G [u][V] < distance[v])
                            distance[V] = distance[u] + G [u] [V] ;
                            parent[v]=u ;
                 }
                }
      for (k=0 ; k<E; k++)
          {
                u=edge [k][0], v=edge[k][1];
                if (distance[u]+ G [u][V] < distance[u] )
                      flag =0;
          }
if (flag)
    for (i=0; i<v; i++)
            printf("vertex %d -> cost = %d parent = %d\n",
                        i+1, distance[i], parent[i]+1);

    return flag;

 }
int main()
 {
    int v, edge[20][2], er[20][20], i, j, k=0;
    printf("BELLMAN FORD\n");
    printf("Enter no. of vertices:");
    scanf("%d", &v);
    printf("Enter graph in matrix form:\n");
    for (i=0; i<v; i++)
          for (j=0; j<v; j++)
        {
              scanf("%d", &er[i][j]);
              if(er[i][j]!=0)
                   edge[k][0]=i, edge[k++][1]=j;
        }
     if (Bellman_ford(er,v,k,edge))
          print("\n No negative weight cycle \n");
     else printf("\n Negative weight cycle exists\n");
     return 0;

  }
```

OUTPUT

```
      BELLMAN FORD
      Enter no. of vertices : 4
      Enter graph in matrix form:
       0   5   4   999
       5   0   6    3
       999 3   1   6
       2   0   1   4
   Enter source : 1
    Vertex 1 -> cost = 0   parent = 0
    Vertex 2 -> cost = 5   parent = 1
    vertex 3 ->  cost = 4  parent = 1
     Vertex 4 -> cost = 8  parent = 2
```

Dijkstra

```
#include <stdio.h>
#include <conio.h>
# define  INFINITY 999;
# define  MAX 10
void dijkstra (int G[MAX][MAX], int n, int startnode);
int main () {
        int G[MAX][MAX], i, j, n, u;
        printf(" Enter no. of vertices:"):
        scanf ("%d", &n); printf("\n Enter the adjacency)
        for (i=0 ; i<n; i++)
              for (j=0; j<n; j++)
                    scanf ("%d", &G[i][j]);
        printf("\n Enter the starting node:");
         scanf ("%d" , &u);
          dijkstra (G,n,u);
          return 0;
   }
void dijkstra (int G[MAX][MAX], int n, int startnode)
  {
      int cost[MAX][MAX], distance[MAX], pred[MAX];
      int visited [MAX], count, mindistance, nextnode, i, j;
      for (i=0; i<n; j++)
          for (j=0 ; j<n ; j++)
                if (G[i][j]=0)
                   cost [i][j] = INFINITY;
                else
                   cost[i][j] = G[i][j];
      for (i=0; i<n; i++) {
          distance[i] = cost[startnode][i];
           pred [i] = startnode;  visited [i]=0;
      }
      distance[startnode]=0;
       visited [startnode]=1;
        count=1;
```

```
while (count<n-1) {
        mindistance =INFINITY;
        for (i=0; i<n; i++)
            if (distance[i] <mindistance && !visited[i
                    mindistance = distance[i]; nextnode
        }
        visited[nextnode] = 1;
          for (i=0; i<n; i++)
              if (!visited[i])
                  if (mindistance + cost[nextnode][i]
                              < distance[i])
                  {
                      distance[i]=mindistance cost[nextnod
                          pred[i]=nextnode;
                  }
              count++;
      }
      for (i=0; i<n; i++)
          if (i!=startnode) {
              printf ("\n Distance of node %d =%d",i,distance[i
             printf ("\n Path =%d", i) ; j=i ;
            do {
                j = pred[j];
                printf ("<- %d", j);
                } while (j!=startnode);
            }
          }
       }
```

OUTPUT

```
Enter no. of vertices : 4
Enter the adjacency matrix:
    0   5   4  999
    5   0   6   3
  999   3   1   6
    2   0   1   4

Enter the starting node: 1
Distance of node0 = 5
    path = 0 <- 1
Distance of node2 = 4
    path = 2 <- 3 <- 1
Distance of node3 = 3
    Path = 3 <- 1
```