**CODE**

| **#include<studio>** |  |
| --- | --- |
|  | **#include<conio.h>** |
|  | **#define INFINITY 9999** |
|  | **#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("\nEnter the adjacency matrix:\n");** |
|  |  |
|  | **for(i=0;i<n;i++)** |
|  | **for(j=0;j<n;j++)** |
|  | **scanf("%d",&G[i][j]);** |
|  |  |
|  | **printf("\nEnter 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;** |
|  |  |
|  | **//pred[] stores the predecessor of each node** |
|  | **//count gives the number of nodes seen so far** |
|  | **//create the cost matrix** |
|  | **for(i=0;i<n;i++)** |
|  | **for(j=0;j<n;j++)** |
|  | **if(G[i][j]==0)** |
|  | **cost[i][j]=INFINITY;** |
|  | **else** |
|  | **cost[i][j]=G[i][j];** |
|  |  |
|  | **//initialize pred[],distance[] and visited[]** |
|  | **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;** |
|  |  |
|  | **//nextnode gives the node at minimum distance** |
|  | **for(i=0;i<n;i++)** |
|  | **if(distance[i]<mindistance&&!visited[i])** |
|  | **{** |
|  | **mindistance=distance[i];** |
|  | **nextnode=i;** |
|  | **}** |
|  |  |
|  | **//check if a better path exists through nextnode** |
|  | **visited[nextnode]=1;** |
|  | **for(i=0;i<n;i++)** |
|  | **if(!visited[i])** |
|  | **if(mindistance+cost[nextnode][i]<distance[i])** |
|  | **{** |
|  | **distance[i]=mindistance+cost[nextnode][i];** |
|  | **pred[i]=nextnode;** |
|  | **}** |
|  | **count++;** |
|  | **}** |
|  |  |
|  | **//print the path and distance of each node** |
|  | **for(i=0;i<n;i++)** |
|  | **if(i!=startnode)** |
|  | **{** |
|  | **printf("\nDistance of node%d=%d",i,distance[i]);** |
|  | **printf("\nPath=%d",i);** |
|  |  |
|  | **j=i;** |
|  | **do** |
|  | **{** |
|  | **j=pred[j];** |
|  | **printf("<-%d",j);** |
|  | **}while(j!=startnode);** |
|  | **}** |
|  | **}**  This C program implements Dijkstra's algorithm to find the shortest path from a starting node to all other nodes in a given graph represented by an adjacency matrix. Here's an analysis of the code structure and documentation of how each function works: **Code Structure Analysis**  1. **Includes and Definitions:**    * #include <stdio.h>: Includes the standard input/output library.    * #include <conio.h>: Includes console input/output functions (not used in this program).    * #define INFINITY 9999: Defines a large value to represent infinity.    * #define MAX 10: Defines the maximum number of vertices in the graph. 2. **Function Declarations:**    * void dijkstra(int G[MAX][MAX], int n, int startnode);: Declaration of the Dijkstra function. 3. **Main Function:**    * Reads the number of vertices (n) and the adjacency matrix (G).    * Reads the starting node (u).    * Calls the dijkstra function to compute shortest paths from the starting node. 4. **Dijkstra Function:**    * Computes the shortest paths from the starting node to all other nodes using Dijkstra's algorithm.    * Prints the shortest path and distance to each node from the starting node.  **Function Documentation****Main Function** c  Copy code  int main() {  int G[MAX][MAX], i, j, n, u;  printf("Enter no. of vertices:");  scanf("%d", &n);  printf("\nEnter the adjacency matrix:\n");    for (i = 0; i < n; i++)  for (j = 0; j < n; j++)  scanf("%d", &G[i][j]);    printf("\nEnter the starting node:");  scanf("%d", &u);  dijkstra(G, n, u);    return 0;  }   * **Purpose:** Reads input from the user and calls the dijkstra function to compute shortest paths. * **Input:**   + Number of vertices (n).   + Adjacency matrix (G).   + Starting node (u). * **Output:** Calls dijkstra to print shortest paths and distances.  **Dijkstra Function** c  Copy code  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;    // Create the cost matrix  for (i = 0; i < n; i++)  for (j = 0; j < n; j++)  if (G[i][j] == 0)  cost[i][j] = INFINITY;  else  cost[i][j] = G[i][j];    // Initialize pred[], distance[] and visited[]  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;    // nextnode gives the node at minimum distance  for (i = 0; i < n; i++)  if (distance[i] < mindistance && !visited[i]) {  mindistance = distance[i];  nextnode = i;  }    // Check if a better path exists through nextnode  visited[nextnode] = 1;  for (i = 0; i < n; i++)  if (!visited[i])  if (mindistance + cost[nextnode][i] < distance[i]) {  distance[i] = mindistance + cost[nextnode][i];  pred[i] = nextnode;  }  count++;  }  // Print the path and distance of each node  for (i = 0; i < n; i++)  if (i != startnode) {  printf("\nDistance of node%d=%d", i, distance[i]);  printf("\nPath=%d", i);    j = i;  do {  j = pred[j];  printf("<-%d", j);  } while (j != startnode);  }  }   * **Purpose:** Implements Dijkstra's algorithm to compute the shortest path from the starting node to all other nodes. * **Input:**   + Adjacency matrix (G).   + Number of vertices (n).   + Starting node (startnode). * **Output:** Prints the shortest path and distance to each node from the starting node. * **Process:**   + **Cost Matrix Creation:** Converts the adjacency matrix to a cost matrix, where INFINITY represents no direct path.   + **Initialization:** Initializes distance, predecessor, and visited arrays.   + **Algorithm Execution:** Repeatedly selects the nearest unvisited node and updates distances.   + **Output Results:** Prints the shortest path and distance to each node. |