|  |  |
| --- | --- |
| **NAME:** | Hardik Garg |
| **UID:** | 2021300036 |
| **SUBJECT** | Design and Analysis of Algorithms |
| **EXPERIMENT NO:** | 06 |
| **AIM:** | To implement Dijkstra’s Algorithm- single source shortest path Algorithm |
| **Algorithm:** | **Djikstra Algorithm**   1. function Dijkstra(Graph, source): 2. for each vertex v in Graph.Vertices: 3. dist[v] ← INFINITY 4. prev[v] ← UNDEFINED 5. add v to Q 6. dist[source] ← 0 7. while Q is not empty: 8. u ← vertex in Q with min dist[u] 9. remove u from Q 10. for each neighbor v of u still in Q: 11. alt ← dist[u] + Graph.Edges(u, v) 12. if alt < dist[v]: 13. dist[v] ← alt 14. prev[v] ← u 15. return dist[], prev[] |
| **CODE:** | **A weighted, directed graph G=(V; E) for the case in which all edge weights are nonnegative with source s (Dijkstra’s Algorithm)**  #include<bits/stdc++.h>  using namespace std;  int V;  int minDistance(int distance[],bool sptSet[]){  int minDist=INT\_MAX;  int minVertex=0;  for(int i=0;i<V;i++){  if(sptSet[i]==false && distance[i]<=minDist){  minDist=distance[i];  minVertex=i;  }  }  return minVertex;  }  void printSolution(int dist[])  {  cout << "\nVertex \t Distance from Source" << endl;  for (int i = 0; i < V; i++)  cout << i << " \t\t\t\t" << dist[i] << endl;  }  void dijkstra(int \*\*graph, int src)  {  int dist[V];  bool sptSet[V];  for (int i = 0; i < V; i++){  dist[i] = INT\_MAX;  sptSet[i] = false; // All s=distance initialised to INF  }  dist[src] = 0;  for (int count = 0; count < V - 1; count++) {  int u = minDistance(dist, sptSet); //u is vertex with min distance  sptSet[u]=true; // u included  // Update dist value of the adjacent vertices of the picked vertex.  for (int v = 0; v < V; v++)  // Update dist[v] only if is not in sptSet, there is an edge from u to v,  // and total weight of path from src to v through u is smaller than current value of dist[v]  if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v])  dist[v] = dist[u] + graph[u][v];  }  // print the constructed distance array  printSolution(dist);  }  int main(){  cout<<"Enter the number of vertices :";  cin>>V;  int \*\*graph=new int\*[V];  for(int i=0;i<V;i++)  {  graph[i]=new int[V];  }  for(int i=0;i<V;i++){  for(int j=0;j<V;j++){  graph[i][j]=0;  }  }  cout<<"Enter the number of edges :";  int e; cin >> e;  for(int i=0;i<e;i++){  cout<<"\nEnter the Vertices of the edge "<<i<<" :";  int a,b,w;  cin>>a>>b;  cout<<"Enter the Weight of the edge "<<i<<" :";  cin>>w;  graph[a][b]=w;  graph[b][a]=w;  }  dijkstra(graph,0);  return 0;  } |
| **Output:** | Output for the given graph: |
| **Conclusion:** | In conclusion, the experiment of Dijkstra's algorithm demonstrated its effectiveness in finding the shortest path in a graph. By iteratively visiting the nodes with the lowest distance and updating their distances, the algorithm guarantees that the final path is indeed the shortest. The results of the experiment showed that Dijkstra's algorithm can handle graphs with various topologies and edge weights efficiently, with a time complexity of O((E+V)logV), where E is the number of edges and V is the number of vertices. Overall, the experiment confirmed the usefulness and versatility of Dijkstra's algorithm in solving shortest path problems in graph theory. |