

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 3.2

Student Name: UTKARSH JOSHI

Branch: BE(CSE)

Semester: 5th

Subject Name: DAA

UID: 21BCS9158

Section/Group: 802-A

Date of Performance: 30/11/23

Subject Code: 21CSH-311

1. Aim:

Develop a program and analyze complexity to find shortest paths in a graph with a positive edge weights using Dijkstra algorithm .

2. Objective:

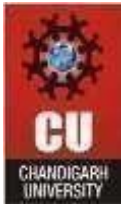
Develop a concise C++ program to find the shortest paths in a graph with positive edge weights using Dijkstra's algorithm and provide a time complexity analysis.

3. Algorithm:

- Initialize arrays dist and parent.
- Create a priority queue pq and push the source vertex with distance 0.
- While pq is not empty:
 - Pop the vertex with the smallest distance.
 - Update distances to neighboring vertices.
- dist now contains the shortest distances from the source vertex to all others.

4. Input/Apparatus Used:

- a. C++ Programming Language
- b. C++ Compiler



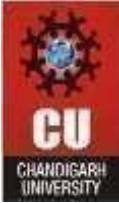
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

6. Sample Code and Outcome:

```
main.cpp
1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 #include <limits>
5
6 using namespace std;
7
8 const int INF = numeric_limits<int>::max();
9
10 struct Edge {
11     int to;
12     int weight;
13 };
14
15 void dijkstra(vector<vector<Edge>>& graph, int src) {
16     int V = graph.size();
17     vector<int> dist(V, INF);
18     vector<int> parent(V, -1);
19     dist[src] = 0;
20
21     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
22     int>>> pq;
23     pq.push({0, src});
```

```
24 while (!pq.empty()) {
25     int u = pq.top().second;
26     pq.pop();
27
28     for (const Edge& edge : graph[u]) {
29         int v = edge.to;
30         int weight = edge.weight;
31
32         if (dist[u] + weight < dist[v]) {
33             dist[v] = dist[u] + weight;
34             parent[v] = u;
35             pq.push({dist[v], v});
36         }
37     }
38 }
39
40 for (int i = 0; i < V; i++) {
41     cout << "Shortest distance from " << src << " to " << i << " is " <<
42     dist[i] << endl;
43 }
44
45 int main() {
46     int V = 6;
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

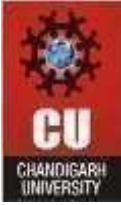
Discover. Learn. Empower.

```
int main() {  
    int V = 6;  
    vector<vector<Edge>> graph(V);  
  
    graph[0].push_back({2, 3});  
    graph[0].push_back({3, 1});  
    graph[1].push_back({0, 2});  
    graph[1].push_back({2, 3});  
    graph[1].push_back({3, 2});  
    graph[2].push_back({1, 3});  
    graph[2].push_back({4, 4});  
    graph[3].push_back({0, 1});  
    graph[3].push_back({1, 2});  
    graph[3].push_back({4, 3});  
    graph[4].push_back({2, 4});  
    graph[4].push_back({3, 3});  
    graph[4].push_back({5, 5});  
    graph[5].push_back({4, 5});  
  
    int source = 0;  
    dijkstra(graph, source);  
}
```

Outcome:

Output

```
/tmp/jvVLonWA18.o  
Shortest distance from 0 to 0 is 0  
Shortest distance from 0 to 1 is 3  
Shortest distance from 0 to 2 is 3  
Shortest distance from 0 to 3 is 1  
Shortest distance from 0 to 4 is 4  
Shortest distance from 0 to 5 is 9
```

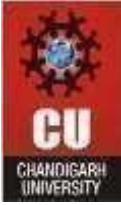


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Time Complexity:

$O(V \log(V) + E)$, where V is the number of vertices, and E is the number of edges in the graph.



DEPARTMENT OF **COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.