

## Output of the run

```
(hacker@kali)-[~/Downloads]
$ make all
g++ -fprofile-arcs -ftest-coverage dijkstra.cpp -o dijkstra

(hacker@kali)-[~/Downloads]
$ ./dijkstra
Enter number of vertices (or -1 to exit): 2
Enter the adjacency matrix (use 0 for no edge):
[0][0]: 1
[0][1]: 2
[1][0]: 3
[1][1]: 4
Enter source vertex (0 to 1): 1
Vertex    Distance from Source
0          3
1          0
----- Done -----

Enter number of vertices (or -1 to exit): -1
```

**Jcov output when the user enters a “good” input**

```
(hacker@kali)-[~/Downloads]
$ cat dijkstra.cpp.gcov
-:      0:Source:dijkstra.cpp
-:      0:Graph:dijkstra.gcno
-:      0:Data:dijkstra.gcda
-:      0:Runs:1
-:      1:#include <iostream>
-:      2:#include <vector>
-:      3:#include <climits>
-:      4:using namespace std;
-:      5:
1:      6:int minDist(vector<int> &dist, vector<bool> &sptSet) {
1:      7:    int min = INT_MAX, idx = -1;
3:      8:    for (int i = 0; i < dist.size(); i++) {
2:      9:      if (!sptSet[i] && dist[i] < min) {
1:     10:        min = dist[i];
1:     11:        idx = i;
-:     12:      }
-:     13:    }
1:     14:    return idx;
-:     15:}
-:     16:
1:     17:void dijkstra(vector<vector<int>> &graph, int src) {
1:     18:    int V = graph.size();
1:     19:    vector<int> dist(V, INT_MAX);
1:     20:    vector<bool> sptSet(V, false);
-:     21:
1:     22:    dist[src] = 0;
-:     23:
2:     24:    for (int count = 0; count < V - 1; count++) {
1:     25:      int u = minDist(dist, sptSet);
1:     26:      if (u == -1)
#####:     27:        break;
-:     28:
1:     29:      sptSet[u] = true;
-:     30:
3:     31:      for (int v = 0; v < V; v++) {
3:     32:        if (!sptSet[v] && graph[u][v] > 0 &&
4:     33:          dist[u] != INT_MAX &&
1:     34:          dist[u] + graph[u][v] < dist[v]) {
1:     35:            dist[v] = dist[u] + graph[u][v];
-:     36:          }
-:     37:        }
-:     38:      }
```

```

-: 39:
1: 40:     cout << "Vertex \t Distance from Source\n";
3: 41:     for (int i = 0; i < V; i++)
2: 42:         cout << i << " \t\t " << dist[i] << '\n';
1: 43: }
-: 44:
2: 45: int main() {
-: 46:     for (;;) {
-: 47:         int V;
2: 48:         cout << "Enter number of vertices (or -1 to exit): ";
2: 49:         if (!(cin >> V) || V <= 0) break;
-: 50:
2: 51:         vector<vector<int>> graph(V, vector<int>(V));
-: 52:
1: 53:         cout << "Enter the adjacency matrix (use 0 for no edge):\n"
;
3: 54:         for (int i = 0; i < V; i++) {
6: 55:             for (int j = 0; j < V; j++) {
4: 56:                 cout << "[" << i << "]" << j << ": ";
4: 57:                 if (!(cin >> graph[i][j])) {
#####: 58:                     cerr << "Invalid input at [" << i << "]" << j
<< "]\n";
#####: 59:                     return 1;
-: 60:                 }
4: 61:                 if (graph[i][j] < 0) {
#####: 62:                     cerr << "Error: Negative weights are not allowe
d in Dijkstra.\n";
#####: 63:                     return 1;
-: 64:                 }
-: 65:             }
-: 66:         }
-: 67:
-: 68:         int src;
1: 69:         cout << "Enter source vertex (0 to " << V - 1 << "): ";
1*: 70:         if (!(cin >> src) || src < 0 || src >= V) {
#####: 71:             cerr << "Invalid source vertex.\n";
#####: 72:             return 1;
-: 73:         }
-: 74:
1: 75:         dijkstra(graph, src);
1: 76:         cout << "----- Done ----- \n\n";
2: 77:     }
-: 78:
1: 79:     return 0;
-: 80: }

```

## jcov output when the user enters a negative weight

```
(hacker@kali)~/Downloads
$ cat dijkstra.cpp.gcov
-: 0:Source:dijkstra.cpp
-: 0:Graph:dijkstra.gcno
-: 0:Data:dijkstra.gcda
-: 0:Runs:1
-: 1:#include <iostream>
-: 2:#include <vector>
-: 3:#include <limits>
-: 4:using namespace std;
-: 5:
-: 6:int minDist(vector<int> &dist, vector<bool> &sptSet) {
-: 7:     int min = INT_MAX, idx = -1;
-: 8:     for (int i = 0; i < dist.size(); i++) {
-: 9:         if (!sptSet[i] && dist[i] < min) {
-:10:             min = dist[i];
-:11:             idx = i;
-:12:         }
-:13:     }
-:14:     return idx;
-:15: }
-:16:
-:17:void dijkstra(vector<vector<int>> &graph, int src) {
-:18:     int V = graph.size();
-:19:     vector<int> dist(V, INT_MAX);
-:20:     vector<bool> sptSet(V, false);
-:21:
-:22:     dist[src] = 0;
-:23:
-:24:     for (int count = 0; count < V - 1; count++) {
-:25:         int u = minDist(dist, sptSet);
-:26:         if (u == -1)
-:27:             break;
-:28:
-:29:         sptSet[u] = true;
-:30:
-:31:         for (int v = 0; v < V; v++) {
-:32:             if (!sptSet[v] && graph[u][v] > 0 &&
-:33:                 dist[u] != INT_MAX &&
-:34:                 dist[u] + graph[u][v] < dist[v]) {
-:35:                 dist[v] = dist[u] + graph[u][v];
-:36:             }
-:37:         }
-:38:     }
-:39:
-:40:     cout << "Vertex \t Distance from Source\n";
-:41:     for (int i = 0; i < V; i++)
-:42:         cout << i << " \t\t " << dist[i] << "\n";
-:43: }
-:44:
-:45:int main() {
-:46:     for (;;) {
-:47:         int V;
-:48:         cout << "Enter number of vertices (or -1 to exit): ";
-:49:         if (!(cin >> V) || V <= 0) break;
-:50:
-:51:         vector<vector<int>> graph(V, vector<int>(V));
-:52:
-:53:         cout << "Enter the adjacency matrix (use 0 for no edge):\n";
-:54:         for (int i = 0; i < V; i++) {
-:55:             for (int j = 0; j < V; j++) {
-:56:                 cout << "[" << i << "]" << j << ": ";
-:57:                 if (!(cin >> graph[i][j])) {
-:58:                     cerr << "Invalid input at [" << i << "]" << j
-:59:                     << "]\n";
-:60:                     return 1;
-:61:                 }
-:62:                 if (graph[i][j] < 0) {
-:63:                     cerr << "Error: Negative weights are not allowed in Dijkstra.\n";
-:64:                     return 1;
-:65:                 }
-:66:             }
-:67:
-:68:             int src;
-:69:             cout << "Enter source vertex (0 to " << V - 1 << "): ";
-:70:             if (!(cin >> src) || src < 0 || src >= V) {
-:71:                 cerr << "Invalid source vertex.\n";
-:72:                 return 1;
-:73:             }
-:74:
-:75:             dijkstra(graph, src);
-:76:             cout << "----- Done ----- \n\n";
-:77:         }
-:78:
-:79:         return 0;
-:80:     }
}
```