Printing Paths in Dijkstra's Shortest Path Algorithm

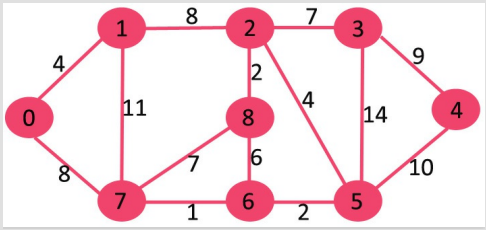
Given a graph and a source vertex in graph, find shortest paths from source to all vertices in the given graph.

We have discussed Dijkstra's Shortest Path algorithm in below posts.

- Dijkstra's shortest path for adjacency matrix representation
- Dijkstra's shortest path for adjacency list representation

The implementations discussed above only find shortest distances, but do not print paths. In this post printing of paths is discussed.

```
For example, consider below graph and source as 0,
```




```
Output should be
Vertex        Distance         Path
0 -> 1    4  0 1
0 -> 2    12  0 1 2
0 -> 3    19  0 1 2 3
0 -> 4    21  0 7 6 5 4
0 -> 5    11  0 7 6 5
0 -> 6    9  0 7 6
0 -> 7    8  0 7
0 -> 8    14  0 1 2 8
```

The idea is to create a separate array parent[]. Value of parent[v] for a vertex v stores parent vertex of v in shortest path tree. Parent of root (or source vertex) is -1. Whenever we find shorter path through a vertex u, we make u as parent of current vertex.

Once we have parent array constructed, we can print path using below recursive function.

```
void printPath(int parent[], int j)
{
    // Base Case : If j is source
    if (parent[j]==-1)
        return;

    printPath(parent, parent[j]);

    printf("%d ", j);
}
```

Below is complete running C++ program.

```
// A C / C++ program for Dijkstra's single source shortest
// path algorithm. The program is for adjacency matrix
// representation of the graph.
#include <stdio.h>
#include <limits.h>

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum distance
// value, from the set of vertices not yet included in shortest
// path tree
```

```
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// Function to print shortest path from source to j
// using parent array
void printPath(int parent[], int j)
{
    // Base Case : If j is source
    if (parent[j]==-1)
        return;

    printPath(parent, parent[j]);

    printf("%d ", j);
}

// A utility function to print the constructed distance
// array
int printSolution(int dist[], int n, int parent[])
{
    int src = 0;
    printf("Vertex\t  Distance\tPath");
    for (int i = 1; i < V; i++)
    {
        printf("\n%d -> %d \t\t %d\t\t%d ", src, i, dist[i], src);
        printPath(parent, i);
    }
}

// Funtion that implements Dijkstra's single source shortest path
// algorithm for a graph represented using adjacency matrix
// representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V];  // The output array. dist[i] will hold
                  // the shortest distance from src to i

    // sptSet[i] will true if vertex i is included / in shortest
    // path tree or shortest distance from src to i is finalized
    bool sptSet[V];

    // Parent array to store shortest path tree
    int parent[V];

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
    {
        parent[0] = -1;
        dist[i] = INT_MAX;
        sptSet[i] = false;
    }

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V-1; count++)
    {
        // Pick the minimum distance vertex from the set of
        // vertices not yet processed. u is always equal to src
        // in first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
```

```
        // Mark the picked vertex as processed
        sptSet[u] = true;

        // 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] + graph[u][v] < dist[v])
            {
                parent[v]  = u;
                dist[v] = dist[u] + graph[u][v];
            }
    }

    // print the constructed distance array
    printSolution(dist, V, parent);
}

// driver program to test above function
int main()
{
    /* Let us create the example graph discussed above */
    int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                       {4, 0, 8, 0, 0, 0, 0, 11, 0},
                       {0, 8, 0, 7, 0, 4, 0, 0, 2},
                       {0, 0, 7, 0, 9, 14, 0, 0, 0},
                       {0, 0, 0, 9, 0, 10, 0, 0, 0},
                       {0, 0, 4, 0, 10, 0, 2, 0, 0},
                       {0, 0, 0, 14, 0, 2, 0, 1, 6},
                       {8, 11, 0, 0, 0, 0, 1, 0, 7},
                       {0, 0, 2, 0, 0, 0, 6, 7, 0}
                      };

    dijkstra(graph, 0);

    return 0;
}
```

Output:

```
Vertex        Distance         Path
0 -> 1    4  0 1
0 -> 2    12  0 1 2
0 -> 3    19  0 1 2 3
0 -> 4    21  0 7 6 5 4
0 -> 5    11  0 7 6 5
0 -> 6    9  0 7 6
0 -> 7    8  0 7
0 -> 8    14  0 1 2 8
```