Suraj Iyer
2021300045
SE Comps A
Batch C

DAA Experiment 6A

Aim - How to find Shortest Paths from Source to all Vertices using Dijkstra's Algorithm

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

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree.

Like Prim's MST, generate a SPT (shortest path tree) with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

Follow the steps below to solve the problem:

Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.

Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign the distance value as 0 for the source vertex so that it is picked first.

While sptSet doesn't include all vertices
Pick a vertex u that is not there in sptSet and has a minimum distance value.
Include u to sptSet.
Then update the distance value of all adjacent vertices of u.

To update the distance values, iterate through all adjacent vertices.
For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Note: We use a boolean array sptSet[] to represent the set of vertices included in SPT. If a value sptSet[v] is true, then vertex v is included in SPT, otherwise not. Array dist[] is used to store the shortest distance values of all vertices.

Code -

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

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

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;
}

// A utility function to print the constructed distance
// array
void printSolution(int dist[])
{
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < V; i++)
         printf("%d \t\t\t\t %d\n", i, dist[i]);
```

```
}

void dijkstra(int graph[V][V], int src)
{
   int dist[V];

   bool sptSet[V];

   for (int i = 0; i < V; i++)
         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++) {

         int u = minDistance(dist, sptSet);

         sptSet[u] = true;

         for (int v = 0; v < V; 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];

   }

   printSolution(dist);
}

int main()
{

   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, 14, 10, 0, 2, 0, 0 },
                                    { 0, 0, 0, 0, 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 -

```
students@students-HP-280-G3-MT:~$ gcc surajiyer_daa_6a.c
students@students-HP-280-G3-MT:~$ ./a.out
Vertex           Distance from Source
0                                0
1                                4
2                                12
3                                19
4                                21
5                                11
6                                9
7                                8
8                                14
students@students-HP-280-G3-MT:~$
```

Conclusion -

- The code calculates the shortest distance but doesn't calculate the path information. Create a parent array, update the parent array when distance is updated (like prim's implementation), and use it to show the shortest path from source to different vertices.
- The code is for undirected graphs, the same Dijkstra function can be used for directed graphs also.
- The code finds the shortest distances from the source to all vertices. If we are interested only in the shortest distance from the source to a single target,

break them for a loop when the picked minimum distance vertex is equal to the target.

- The time Complexity of the implementation is $O(V^2)$. If the input graph is represented using adjacency list, it can be reduced to O(E * log V) with the help of a binary heap. Please see Dijkstra's Algorithm for Adjacency List Representation for more details.
- Dijkstra's algorithm doesn't work for graphs with negative weight cycles. It may give correct results for a graph with negative edges but you must allow a vertex can be visited multiple times and that version will lose its fast time complexity. For graphs with negative weight edges and cycles, the Bellman-Ford algorithm can be used, we will soon be discussing it as a separate post.