Algorithm

```
for (int i = 0; i < new_nodes.size(); i++)
    {
        int u = new_nodes[i];
        for (int v = 0; v < shm->count; v++)
        {
            if (shm->graph[u][v] == 1)
            {
                int src, dest;
                if (distance[u] < distance[v] - 1)
                {
                    distance[v] = distance[u] + 1;
                    src = u;
                    par[v]=u;
                    dest = v;
                }
                else if (distance[v] < distance[u] - 1)
                {
                    distance[u] = distance[v] + 1;
                    src = v;
                    par[u]=v;
                    dest = u;
                }
                else
                {
                    continue;
                }
                priority_queue<pair<int, int>> pq;
                pq.push({-distance[dest], dest});
                while (!pq.empty())
                {
                    int top = pq.top().second;
                    pq.pop();
                    for (int next = 0; next < shm->count; next++)
                    {
                        if (shm->graph[top][next] == 1 && distance[next] -1 > distance[
                            top])
                        {
                            distance[next] = distance[top] + 1;
                            par[next]=top;
                            pq.push({-distance[next], next});
                        }
                    }
                }
            }
        }
    }
```

Description

When a new node is added all the edges directed with it are added. So, we need to handle all the edges connected to all the new nodes only.
There will be three cases:

1. when *distance*[new node] > *distance*[neighbouring node] + 1. Then we can reach the new node from its neighbour by distance = *distance*[neighbouring node] + 1 and mark the parent of new node as this neighbouring node.

2. when *distance*[new node] < *distance*[neighbouring node] − 1. Then we can reach the neighbouring node from new node by distance = *distance*[new node] + 1 and mark the parent of neighbouring node as new node.

3. when *distance*[new node] = *distance*[neighbouring node]±1. Then we don't need to do anything as nothing will change either in minimum distance of new node or minimum distance of its neighbour.

Then apply the Breadth First Search Algorithm from the node whose distance is changed. By this way, we are finding the path for new nodes from the path of its neighbouring node and modify the path of its neighbouring node as per the changes of the new node.
So, no calculation of complete Dijktra's Algorithm. Only a small amount of priority queue BFS is running.