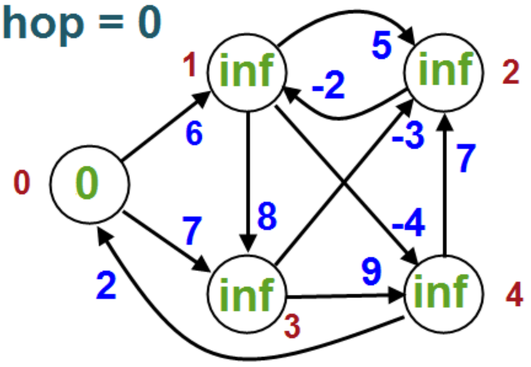
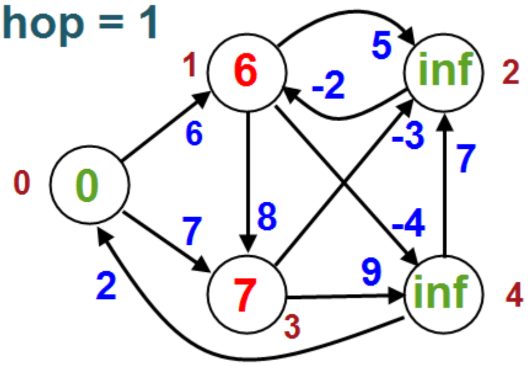
```
function BellmanFord(list vertices, list edges, vertex source)
    ::distance[],predecessor[]
     // This implementation takes in a graph, represented as lists of vertices and edges, and
    // fills two arrays (distance and predecessor) with shortest-path (less cost/distance/metric)

    // Step 1: initialize graph
    for each vertex v in vertices:
        distance[v] := inf                // At the beginning , all vertices have a weight of infinity
        predecessor[v] := null            // And a null predecessor

    distance[source] := 0                 // Except for the Source, where the Weight is zero

    // Step 2: relax edges repeatedly
    for i from 1 to size(vertices)-1:
        for each edge (u, v) with weight w in edges:
            if distance[u] + w < distance[v]:
                distance[v] := distance[u] + w
                predecessor[v] := u

// Step 2': relax edges repeatedly by queue (Shortest Path Faster Algorithm)
   push source into Q
        while Q is not empty
           u := deque Q
           for each edge (u, v) in E(G)
              if distance[u] + w < distance[v]:
                 distance[v] := distance[u] + w
                 predecessor[v] := u
                  if v is not in Q:
                     push v into Q

    // Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
            error "Graph contains a negative-weight cycle"
    return distance[], predecessor[]
```

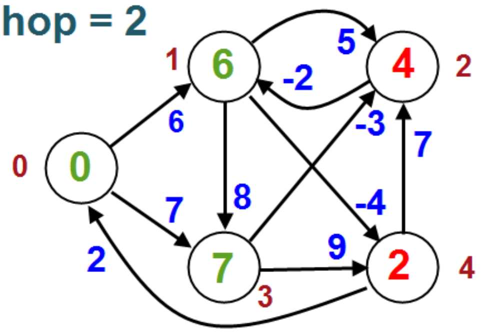
hop = 0



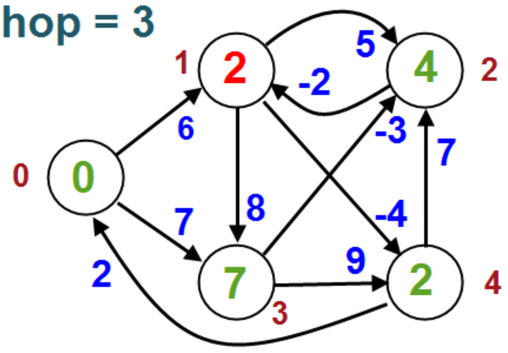
hop = 1



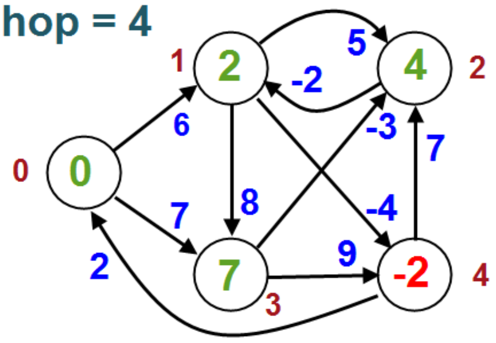
hop = 2



hop = 3



hop = 4



(Example in C)

```
typedef struct {

      int u, v, w;

} Edge;

int NODES, EDGES;     /* the number of nodes and edges */

Edge edges[];    /* large enough for NODES^2 */

int dist[];    /* dist[i] is the minimum distance from source s to node i */

int prev[];    /* prev[i] is the index of the parent of node i in the shortest path from s to node i */


void BellmanFord(int src) {
      int i, j;


      for (i = 0; i < NODES; ++i){
            dist[i] = INFINITY;
            prev[i]=NULL;
      dist[0] = 0;

      for (i = 0; i < NODES - 1; ++i)
            for (j = 0; j < EDGES; ++j){
                  if (dist[edges[j].u] + edges[j].w < dist[edges[j].v]) {
                        dist[edges[j].v] = dist[edges[j].u] + edges[j].w;
                        prev[edges[j].v] = edges[j].u;
                  }

 // ONLY WHEN you want to check for negative cycles //
      for (j = 0; j < EDGES; ++j)
            if (dist[edges[j].u] + edges[j].w < dist[edges[j].v]) {
                  printf("Graph contains a negative-weight cycle!! \n");
                  exit(1);
                  }
```