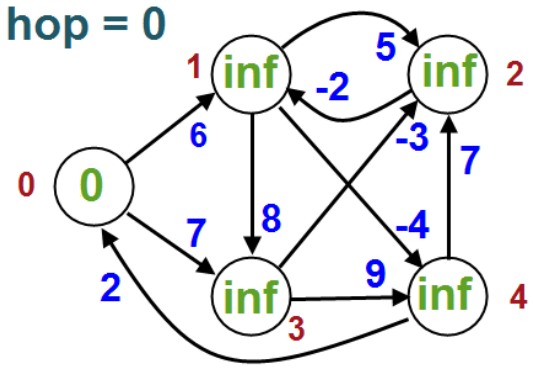
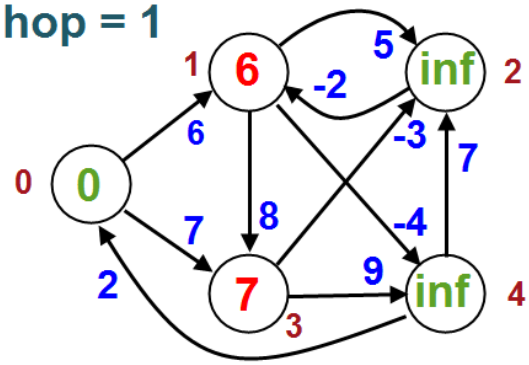
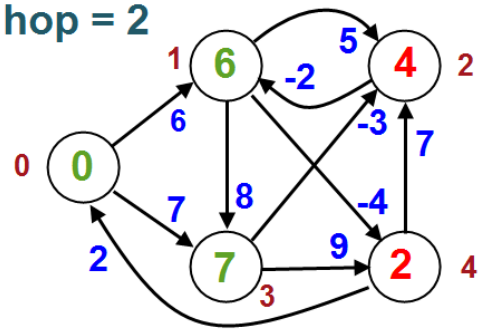
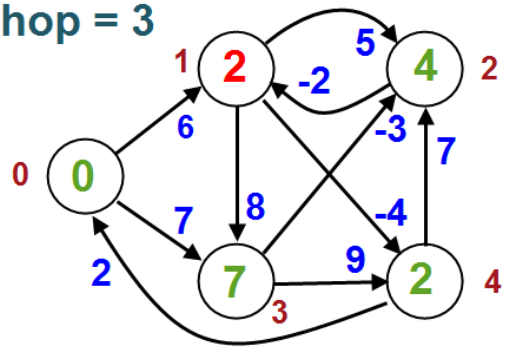
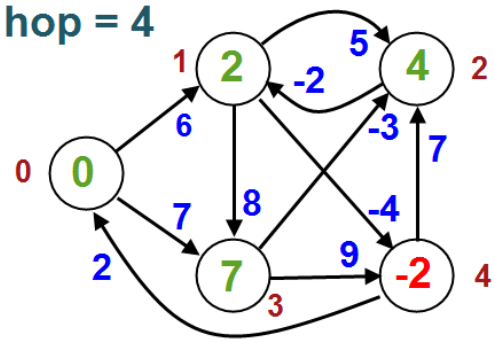
```
function BellmanFord(list vertices, list edges, vertex source)
    ::distance[],predecessor[]
     // This implementation takes in a graph, represented as lists of vertices and edges, and
    // fills two arrays (distance and predecessor) with shortest-path
    // (less cost/distance/metric) information

    // 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 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[]
```












```
typedef struct {
    int u, v, w;
} Edge;

#define INFINITY INT_MAX
```

```
const int NODES = 5 ;       /* the number of nodes */

int EDGES;                  /* the number of edges */

Edge edges[32];             /* large enough for 2^NODES */

int dist[32];               /* dist[i] is the minimum distance from source node s to node i */



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

      for (i = 0; i < NODES; ++i)
            dist[i] = INFINITY;
      dist[src] = 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;
                  }
            }
      }

      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]) {
                        printf("Graph contains a negative-weight cycle!! \n");
                        exit(1);
                  }
            }
      }
}
```