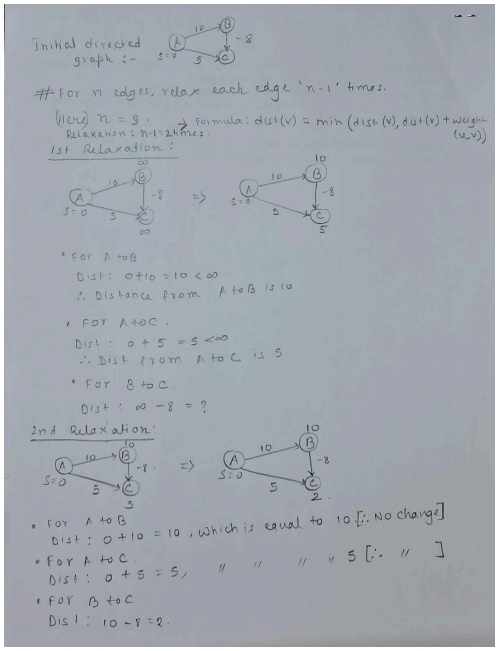
7.Bellman ford

Formula: $dist[v]=\min(dist[v],dist[u]+weight(u,v))$



(expl)

Algorithm:

1. Input:
- Number of vertices (V)
- Number of edges (E)
- List of edges with their source vertex (u), destination vertex (v), and weight (w)
- Source vertex (src)

2. Initialize Distances:
- Create an array `dist` of size (v)
- Set `dist[src]` to 0 (the distance from the source to itself is 0)
- Set all other entries in `dist` to infinity (9999)

3. Relax All Edges:
- Repeat the following (V-1) times:
  - For each edge ((u, v, w)\) in the list of edges:
    - If `dist[u]` is not infinity and `dist[u] + w < dist[v]`:
      - Update `dist[v]` to `dist[u] + w`

4. Check for Negative-Weight Cycles:
- For each edge ((u, v, w) in the list of edges:
  - If `dist[u]` is not infinity and `dist[u] + w < dist[v]`:

- Print "Graph contains negative cycle" and exit the algorithm

5. Output Distances:
- Print "Vertex \t tDistance from Source"
- For each vertex (i) from 0 to (V-1):
  - Print vertex (i) and `dist[i]`

Code:

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

#define MAX_VERTICES 10
#define MAX_EDGES 20
#define INF 9999

typedef struct {
    int source, destination, weight;
} Edge;

void findShortestPaths(int numVertices, int numEdges, int sourceVertex, Edge edges[]) {
    int distances[MAX_VERTICES];

    // Initialize distances
    for (int i = 0; i < numVertices; i++)
        distances[i] = INF;
    distances[sourceVertex] = 0;

    // Relax all edges numVertices - 1 times
    for (int i = 0; i < numVertices - 1; i++)
            {
        for (int j = 0; j < numEdges; j++)
                    {
            int u = edges[j].source;
            int v = edges[j].destination;
            int weight = edges[j].weight;
            if (distances[u] != INF && distances[u] + weight < distances[v])
                distances[v] = distances[u] + weight;
        }
    }

    // Check for negative cycles
    for (int i = 0; i < numEdges; i++)
            {
```

```
            int u = edges[i].source;
            int v = edges[i].destination;
            int weight = edges[i].weight;
            if (distances[u] != INF && distances[u] + weight < distances[v])
                        {
                printf("Graph contains negative cycle\n");
                return;
            }
        }

        // Print distances
        printf("Vertex\tDistance from Source\n");
        for (int i = 0; i < numVertices; i++)
            printf("%d\t%d\n", i, distances[i]);
}

int main()
{
        int numVertices, numEdges;

        printf("Enter the number of vertices and edges: ");
        scanf("%d %d", &numVertices, &numEdges);

        Edge edges[MAX_EDGES];

        printf("Enter source, destination, and weight for each edge:\n");
        for (int i = 0; i < numEdges; i++)
            scanf("%d %d %d", &edges[i].source, &edges[i].destination, &edges[i].weight);

        int sourceVertex;

        printf("Enter the source vertex: ");
        scanf("%d", &sourceVertex);

        findShortestPaths(numVertices, numEdges, sourceVertex, edges);

        return 0;
}
```

input/output: