```
class Program
{
   static void Main()
   {
      string inputDirectory = "DataSet";
      string outputDirectory = "Output";

      // Crear el directorio de salida si no existe
      Directory.CreateDirectory(outputDirectory);

      // Procesar todos los archivos en el directorio de entrada
      foreach (var inputFile in Directory.GetFiles(inputDirectory, "*.txt"))
      {
         var graph = ReadGraph(inputFile);
         string outputFileName = Path.Combine(outputDirectory,
$"Output_{Path.GetFileName(inputFile)}");

         GenerateReport(graph, outputFileName);
      }

      Console.WriteLine("Procesamiento completado. Archivos generados en la
carpeta Output.");
   }

   static Dictionary<string, List<(string destination, int cost)>> ReadGraph(string
filePath)
   {
      var graph = new Dictionary<string, List<(string, int)>>();
```

```
    foreach (var line in File.ReadLines(filePath))
    {
        var parts = line.Split(';');
        string source = parts[0];
        string destination = parts[1];
        int cost = int.Parse(parts[2]);

        if (!graph.ContainsKey(source))
            graph[source] = new List<(string, int)>();
        if (!graph.ContainsKey(destination))
            graph[destination] = new List<(string, int)>();

        // Añadir conexiones bidireccionales
        graph[source].Add((destination, cost));
        graph[destination].Add((source, cost));
    }

    return graph;
  }

  static void GenerateReport(Dictionary<string, List<(string destination, int cost)>>
graph, string outputFileName)
  {
    using (var writer = new StreamWriter(outputFileName))
    {
        foreach (var source in graph.Keys)
        {
            writer.WriteLine($"Costos mínimos desde la estación {source}:");
```

```
            var (distances, pathsWithWeights) = Dijkstra(graph, source);


            foreach (var destination in graph.Keys)
            {
                if (source == destination)
                {
                    writer.WriteLine($"  No hay camino desde la estación {source} hasta
{destination}.");
                }
                else if (distances[destination] == int.MaxValue)
                {
                    writer.WriteLine($"  No hay camino desde la estación {source} hasta
{destination}.");
                }
                else
                {
                    writer.WriteLine($"  Hasta la estación {destination}:
{distances[destination]}.");
                    writer.WriteLine($"Ruta: {string.Join(" -> ",
pathsWithWeights[destination])} ->");
                }
            }


            writer.WriteLine();
        }
    }
}
```

```
  static (Dictionary<string, int> distances, Dictionary<string, List<string>>
pathsWithWeights) Dijkstra(
    Dictionary<string, List<(string destination, int cost)>> graph, string start)
  {
    var distances = new Dictionary<string, int>();
    var paths = new Dictionary<string, List<string>>();
    var priorityQueue = new SortedSet<(int cost, string node)>();

    // Inicialización
    foreach (var node in graph.Keys)
    {
      distances[node] = int.MaxValue;
      paths[node] = new List<string>();
    }

    distances[start] = 0;
    priorityQueue.Add((0, start));
    paths[start].Add($"{start}");

    while (priorityQueue.Count > 0)
    {
      var (currentCost, currentNode) = priorityQueue.Min;
      priorityQueue.Remove(priorityQueue.Min);

      foreach (var (neighbor, cost) in graph[currentNode])
      {
        int newCost = currentCost + cost;
```

```
            if (newCost < distances[neighbor])
            {
                priorityQueue.Remove((distances[neighbor], neighbor));
                distances[neighbor] = newCost;
                priorityQueue.Add((newCost, neighbor));

                // Generar ruta con pesos incluidos
                paths[neighbor] = new List<string>(paths[currentNode])
                {
                    $"{currentNode} -> {neighbor} ({cost})"
                };
            }
        }
    }

    return (distances, paths);
  }
}
```