5)

```
public class Mapa{
  private Grafo<String> mapaCiudades;
  public Mapa(Grafo<String> mapa){ this.mapaCiudades = mapa}

  public ListaGenerica<String> devolverCamino(String ciudad1, String ciudad2){
    ListaGenerica<String> c = new ListaEnlazadaGenerica<String>();
    if(!mapaCiudades.esVacio()){
      Vertice<String> V = buscar(ciudad1);
      if(V!= null){
        boolean[] marcas = new boolean[this.mapaCiudades.listaDeVertices.
        tamanio()];
        dfs(V.posicion, c, ciudad2, this.mapaCiudades, marcas);
      }
    }
    return c;
  }
  Vertice<String>
  private buscar(String ciudad){
    ListaGenerica vertices = mapaCiudades.listaDeVertices();
    Vertice<String> V = null;
    vertices.comenzar();
    while(!vertices.fin() && V == null){
      Vertice<String> act = vertices.proximo();
      if(act.dato().equals(ciudad)){ V = act;}
    }
    return V;
  }
```

```
boolean
private dfs (int pos, ListaGenerica<string> c, String destino,
Grafo<string> g, boolean[] marcas);
 boolean encontre = False; marcas[pos] := True;
  Vertice<string> V= g.Vertice(pos);
  c.agregarFinal(V.dato);
  encontre = V.dato().equals(destino);
  if(!encontre){
     ListaGenerica<Arista<string>> adyacentes = g.listaAdyacentes(V);
     adyacentes.comenzar();
     While(!encontre && !adyacentes.fin()){
        Aristas<String> a =adyacenteS.proximo();
        int posSig = a.verticeDestino().posicion();
        if(!marcas[posSig]{encontre =dfs(posSig, c, destino, g,
        marcas}
  }  }
  if(!encontre){c.eliminarEn(c.tamanio());}
  return encontre;
}
```

```
public ListaGenerica<string> devolverCaminoExceptuado(String ciudad
String ciudad2, ListaGenerica<string> ciudades){
  ListaGenerica<string> c= new ListaEnlazadaGenerica<String>();
  if(!this.mapaCiudades.esVacio()){
    Vertice<string> V= buscar(ciudad1);
    if(V!=null){
      boolean[] marcas = new boolean[this.mapaCiudades.listaDeVerti
      ces().tamanio()+1];
```

```
        marcar(ciudades, marcas);
        dfs(V.posicion, c, ciudad2, this.mapaCiudades, marcas);
      }
    }
  }
  return c;
}

private void marcar(ListaGenerica<String> ciudades, boolean[]
marcas){
  ciudades.comenzar();
  [illegible]
  while(!ciudades.fin()){
    marcas[buscar(ciudades.proximo().posicion())]=True;
  }
}

public ListaGenerica<String> caminoMasCorto(String ciudad1, String
ciudad2){
  ListaGenerica<String> c= new ListaEnlazadaGenerica<String>();
  if(!mapaCiudades.esVacio()){
  int pos = buscarPos([illegible] ciudad1);
  if(pos != -1) [illegible]{
    boolean[]marcas = new boolean[this.mapaCiudades.listaDeVertices().
    tamanio()+1];
    dfsMasCorto(pos, c, new ListaEnlazadaGenerica, ciudad2, this.mapa-
    Ciudades, marcas);
  }
  }
  return c;
}
```

```
private int buscarPos(String city){
    ListaGenerica<String> verts = this.mapaCiudades.listaDe
    Vertices();
    int pos = -1;
    while(!verts.fin() && pos==-1){
        Vertice<String> V = verts.proximo();
        if(V.dato.equals(city){ pos = V.posicion}
    }
    return pos;
}
```

```
private Void dfsMasCorto(int pos, ListaGenerica<String> final, ListaGenerica
<String> actual, String destino, Grafo g, boolean[] marcas){
    marcas[pos]=True;
    Vertice<String> V= g.Vertice(pos);
    actual.agregarAlFinal(V);
    if(V.dato.equals(destino){
        if(final.tamanio()>actual.tamanio){clonar(final,
        actual);
    }else{
        ListaGenerica<Arista<String>> ady = V.listaDeAdyacentes();
        ady.comenzar();
        while(!ady.fin()){
            Arista<String> a= adys.proximo();
            int Sigpos= a.VerticeDestino().posicion(); if(!marcas[sigPos]){
            dfsMasCorto(sigpos, final, actual, destino, g, marcas);}
        }
    actual.eliminarEn(actual.tamanio); marcas[pos]= False;
}
```

final.esVacia

```
private Void clonar(ListaGenerica c; ListaGenerica a){
    c.comenzar();
    while(!c.fin()){ c.eliminar(c.proximo)}
    a.comenzar();
    while(!a.fin()){ c.agregarAlFinal(a.proximo);}
}
```

```
public ListaGenerica<String> caminoSinCargarCombustible(String ciudad1, String Ciudad2, int tanqueAuto){
    ListaGenerica<String> c= new ListaEnlazadaGenerica<String>();
    if(!mapaCiudades.esVacio()){
        boolean[] marcas = new boolean[mapaCiudades.listaDeAdyacentes.
        tamanio +1];
        dfsSinCargar(marcas, c, V.posicion, mapaCiudades, ciudad2, tan-
        queAuto);
    }return c;
}
```

```
private boolean dfsSinCargar(boolean[] marcas, ListaGenerica<String>
c, int pos, Grafo<String> g, String d, int tanque){
    marcas[pos] =True;
    Vertice<String> V= g.Vertice(pos);
    boolean encontre = V.dato.equals(d);
    if(!encontre){
        ListaGenerica<Arista<String>> adys= g.listaDeAdyacentes(V);
        adys.comenzar();
```

```
while(!adys.fin() && !encontre){
    [illegible]
    Arista<String> a= adys.proximo(); int SigPos = a.VerticeDestino().
    [illegible]posicion();
    if(!marcas[SigPos] && a.peso()< tanque){
        encontre= dfsSinCargar(marcas, c, sigPos, g, d, tanque-
        a.peso());
    }
}
if(!encontre){c.eliminarEn(c.tamanio());}
return encontre;
}
```

```
public ListaGenerica<String> caminoConMenorCargaDeCombustible(
String ciudad1, String ciudad2, int TanqueAuto){
  ListaGenerica<string> c= new ListaEnlazadaGenerica<string>();
  if(!mapaCiudades.esVacio()){
     int pos = buscarPos(ciudad1);
     if(pos <>-1){
        boolean[] marcas = boolean[mapaCiudades.listaDeVertices().
        tamanio()+1];
        dfsFinal( c, marcas, new ListaEnlazadaGenerica(), ciudad2,
        TanqueAuto, 0, , 0, pos, mapaCiudades);
                    tanqueAuto
     return c;
}
private class CargasyTanque{
private int carga; private int tanque; private int total;
private int dfsFinal(ListaGenerica c, boolean[]
marcas, ListaGenerica<String> act, String destino, int TanqueTotal,
int cargasAct, int tanqueActual, int cargas, int pos Grafo g){
  marcas[pos] = True;
  Vertice<String> V= g.Vertice(pos);
act.agregarAlFinal(V);
  if(V.dato.equals(destino) && cargas>cargasAct
  { clonar(c, act); cargas = cargasAct;}
  else{
  | if (tanqueActual.equals(0)){ tanqueActual+= TanqueTotal;
     cargasAct++;}
```

```
ListaGenerica<Arista<String>> adya = g.listaDeAdyacentes();
adya.comenzar();

while(!ady.fin()){
    Aristas<String> a = adya.proximo();
    posSig = adya.verticeDestino().posicion();
    if(!marca[posSig] && (tanqueActual - a.peso()) >= 0){
cargas = dfsFinal(c, marcas, act, destino, TanqueTotal, cargasAct,
    tanqueAct - a.peso(), cargas, pos, g);
    }
act.eliminarEn(act.tamanio());
marcas[pos] = false;
return cargas;
}
```