- Describa cuál es el criterio de selección.
- ¿En qué estructuras de datos representará la información del problema?
- Explique el algoritmo, es decir, los pasos a seguir para obtener el resultado. No se pide que "lea" el algoritmo ("se define una variable x", "se declara un for"), si no que lo explique ("se recorre la lista/el arreglo/" o "se elije de tal conjunto el que satisface...").
- Escriba el algoritmo en el lenguaje de programación de la materia.

3. Se desea realizar un viaje en un automóvil con autonomía A (en kilómetros), desde la localidad l_0 hasta la localidad l_n pasando por las localidades $l_1, \ldots, l_{n-1}$ en ese orden. Se conoce cada distancia $d_i \leq A$ entre la localidad l_{i-1} y la localidad l_i (para $1 \leq i \leq n$), y se sabe que existe una estación de combustible en cada una de las localidades.

   Escribir un algoritmo que compute el menor número de veces que es necesario cargar combustible para realizar el viaje, y las localidades donde se realizaría la carga.

   Suponer que inicialmente el tanque de combustible se encuentra vacío y que todas las estaciones de servicio cuentan con suficiente combustible.

   - **Criterio de selección**
     Elige la primer ciudad en la que tengo que cargar combustible, o dicho de otra manera, la primera localidad cuya distancia acumulada sobrepase la autonomía constante *A*.

   - **Estructuras de datos**
     Planteo a las localidades como una tupla de dos elementos compuesta por un identificador con su nombre y un natural que expresa la distancia en kilómetros entre ellas.

```
type Localidades = tuple
                     id: string
                     distancia: nat
                   end tuple

type Paradas = tuple
                 localidad: List of Localidades
                 numero_de_paradas: nat
               end tuple

fun viaje(A: nat, L: List of Localidades) ret res: Paradas
```

     Las paradas a realizar también son definidas como una tupla conformada por una lista con las localidades en donde se carga combustible y la cantidad total de veces que se realizó.

   - **Descripción de cómo se soluciona el problema**
     Cargo siempre en la primera localidad (por la suposición del tanque vacío inicial), recorro hasta donde más se pueda, cargo combustible de vuelta y así repetitivamente. Se agrega la ciudad donde se para a la lista de ciudades junto al incremento de la cantidad de veces que se detuvo.

   - **Definición del algoritmo**

```
type Localidades = tuple
                     id: string
                     distancia: nat
                   end tuple

type Paradas = tuple
                 localidad: List of Localidades
                 numero_de_paradas: nat
               end tuple
```

```
fun viaje(A: nat, L: List of Localidades) ret res: Paradas
    var L_aux: List of Localidades
    var l: Localidades
    var nafta: nat

    L_aux := copy_list(L)
    res.localidad := empty_list()
    res.numero_de_paradas := 0
    nafta := 0

    while not is_empty_list(L_aux) do
        l := localidad_donde_recargar(L_aux,nafta)
        nafta := A
        addr(res.localidad,l)
        res.numero_de_paradas := numero_de_paradas + 1
        eliminar_localidades(L_aux,l)
    do

    destroy_list(L_aux)
end fun

fun localidad_donde_recargar(L: List of Localidades, n: nat) ret res: Localidades
    var L_aux: List of Localidades
    var l: Localidades
    var nafta: nat
    var loc_encontrada: bool

    L_aux := copy_list()
    nafta := n
    loc_encontrada := false

    while not is_empty_list(L_aux) ∧ not loc_encontrada do
        l := head(L_aux)
        if l.distancia ≤ nafta then
            res := l
            nafta := nafta - l.distancia
        else
            loc_encontrada := true
        fi
        tail(L_aux)
    od

    destroy_list(L_aux)
end fun

proc eliminar_localidades(in/out L: List of Localidades, in l: nat)
    var l_aux: Localidades
    var loc_encontrada: bool

    loc_encontrada := false

    while not loc_encontrada do
        l_aux := head(L)
        if l_aux = l then
            loc_encontrada := true
        fi
        tail(L)
    od
end proc
```