Se requiere desarrollar un programa en Haskell para procesar la información acerca de los resultados de un rally.
Se elige representar la información de cada corredor como una lista de ternas (origen,destino,tiempo de sprints medido en minutos), cada terna representa un tramo del rally. P.ej.

```
cocho = [("bsas","rosario",[45,41,38]),
         ("rosario","cordoba",[72,76,73]),
         ("cordoba","tucuman",[107,104,112])]

lucho = [("bsas","rosario",[36,35,40]),
         ("rosario","cordoba",[79,77,83]),
         ("cordoba","tucuman",[92,102,99]),
         ("tucuman","santiago",[65,68,63])]

pedro = [("bsas","rosario",[43,47,44]),
         ("rosario","cordoba",[74,68,71]),
         ("santiago","salta",[70,82,95,51])]

vueltaArgentina = [("cocho",cocho),("lucho",lucho),("pedro",pedro)]
```

Se cuenta con las funciones fst3, snd3 y trd3, que devuelven el primer, segundo y tercer elementos de una terna respectivamente.
También conviene tener en cuenta las funciones any (dadas una condición y una lista, indica si algún elemento de la lista cumple con la condición), last (devuelve el último de una lista) y maxLista/minLista (devuelven el máximo y el mínimo de una lista respectivamente), aparte de las más usuales (head, map, filter,any, all).
Para algunos ítems viene bien definir alguna función auxiliar, vale hacerlo.
También vale usar las funciones definidas en ítems anteriores.

Para entrar en calor: indicar el tipo de

- la expresión `vueltaArgentina`
- la función g definida así: `g l p u = filter (u<=) [ f p | f <- l ]`

Definir las siguientes funciones

1. **esTramoEntre** : dados un par de ciudades y un tramo (terna) indica si el origen y destino del tramo son las ciudades del par, respetando el orden. P.ej. entre estas tres
   ```
   esTramoEntre ("bsas","rosario") ("bsas","rosario",[36,35,40])
   esTramoEntre ("bsas","cordoba") ("bsas","rosario",[36,35,40])
   esTramoEntre ("rosario","bsas") ("bsas","rosario",[36,35,40])
   ```
   la primera devuelve True, mientras que las otras dos devuelven False

2. **tiempos** : dados un par (origen,destino) y un corredor, devuelve la lista de tiempos de los sprints del tramo entre las ciudades del par para ese corredor. P.ej.
   ```
   tiempos ("bsas","rosario") lucho
   ```
   devuelve `[36,35,40]`.

3. **pasaPorCiudad** : dados una ciudad y un corredor indica si el corredor pasa por la ciudad. P.ej. en
   ```
   pasaPorCiudad "santiago" cocho
   pasaPorCiudad "santiago" lucho
   ```
   la primera devuelve False y la segunda devuelve True.

4. **incluyeTramoEntre** : que dados un par (origen,destino) y un corredor, indica si el corredor hizo el tramo entre el par de ciudades. P.ej. en
   ```
   incluyeTramoEntre ("tucuman","santiago") cocho
   ```

```
incluyeTramoEntre ("bsas","rosario") cocho
```
la primera devuelve False y la segunda devuelve True.
En alguno de estos dos ejemplos interviene la evaluación diferida, ¿en cuál y por qué?

5. **cuantosLesFaltaTramoEntre** : dados un par (origen,destino) y una lista de corredores, devuelve la cantidad de sublista de aquellos que no hicieron el tramo indicado. P.ej.
```
cuantosLesFaltaTramoEntre ("cordoba","tucuman") [cocho,lucho,pedro]
```
devuelve 1, porque ese tramo le falta a Pedro y a nadie más.

6. **daLoMismo** : dadas dos funciones y un valor, indica si la evaluación de las dos funciones sobre el valor dan el mismo resultado. P.ej. en
```
daLoMismo (*2) (+2) 2
daLoMismo (*2) (+2) 3
```
la primera devuelve True y la segunda devuelve False.

7. **primeroMenor** y **ultimoMenor** : indican en una lista de números si el primer elemento/último elemento son los menores de la lista. P.ej. en
```
primeroMenor [45,41,38]
primeroMenor [92,102,99]
ultimoMenor [45,41,38]
ultimoMenor [92,102,99]
```
la primera y la cuarta devuelven False; segunda y tercera devuelven True.
Usar daLoMismo y minLista.

8. **enQueTramos** : dados un criterio y un corredor devuelve la sublista de pares (origen, destino) cuya cuya lista de sprints cumple con el criterio. P.ej.
```
enQueTramos primeroMenor cocho
```
devuelve
```
[("rosario","cordoba"),("cordoba","tucuman")]
```
Escribir las consultas que indiquen para un corredor
   - en qué tramos hay algún sprint hecho en menos de una hora.
   - en qué tramos todos los sprints se hicieron en menos de una hora.
   - en qué tramos el tiempo total (sumando todos los sprints) es menor a cuatro horas.
   - en qué tramos se hicieron más de 3 sprints.

usando la función enQueTramos.

9. **tramoPopular**: indica si un par (origen,destino) es popular en una competencia, o sea, si todos los corredores de la competencia hicieron ese tramo . P.ej. esta consulta devuelve True:
```
tramoPopular ("bsas","rosario") vueltaArgentina
```

10. **esParejo**: indica para un corredor si para todos los tramos que corrió, la dispersión de los sprints para ese tramo es <= 10 minutos. La dispersión es la difefencia entre el máximo y el mínimo. P.ej.
```
esParejo cocho
```
devuelve True porque sus dispersiones son 7, 4 y 8 respectivamente, mientras que
```
esParejo pedro
```
devuelve False porque su dispersiçon para el tramo Santiago-Salta es 44 > 10.
No usar recursividad explícita, o sea, se pueden usar funciones definidas recursivamente (p.ej. all), pero la función esParejo no puede ella ser recursiva.
Sí vale definir una función auxiliar, pero que tampoco es recursiva explícitamente.

11. **competencia**: recibe una función, un par (origen,destino) y una competencia (p.ej. `vueltaArgentina`) y devuelve una lista de pares
    (nombre del corredor, valor de la función sobre los tiempos para el tramo indicado)
    los corredores que no hicieron el tramo no deben estar incluidos en el resultado.
    P.ej.
    ```
    competencia sum ("cordoba","tucuman") vueltaArgentina
    ```
    devuelve `[("cocho",323),("lucho",293)]`; pedro no está incluido porque no hizo ese tramo.

12. **ganador**: recibe una función, un par (origen,destino) y una competencia (p.ej. `vueltaArgentina`) y devuelve el mismo par que competencia para ganador del tramo según este criterio, o sea, de aquel cuyo valor para la función sea mínimo.
    P.ej.
    ```
    ganador sum ("cordoba","tucuman") vueltaArgentina
    ```
    devuelve `("lucho",293)`.

13. **pasaCorte**: recibe una función, un valor de corte, un corredor, y un par (origen,destino).
    Devuelve True si la función, aplicada a los tiempos de los sprints para el tramo entre las ciudades indicadas, no supera el valor de corte.
    Si el corredor no hizo ese tramo, devuelve False.
    P.ej.
    ```
    pasaCorte sum 300 "cocho" ("cordoba","tucuman")
    ```
    devuelve False porque la suma de los sprints para ese tramo para Cocho es 323, entonces no pasa el corte.
    No usar competencia.

14. **clasificados**: recibe una función, un valor de corte, una lista de ciudades, y una competencia.
    Devuelve la lista de nombres de los clasificados, o sea los que pasan el corte (según lo definido en el ítem anterior) en todos los tramos entre las ciudades indicadas, tomados de a pares consecutivos.
    P.ej.
    ```
    clasificados maxLista 80 ["bsas","rosario", "cordoba"] vueltaArgentina
    ```
    devuelve `["cocho","pedro"]` porque son los que pasan el corte de 80 para el valor máximo en los tramos ("bsas","rosario") y ("rosario","cordoba").
    Lucho no pasa el corte en el tramo ("rosario","cordoba"), porque su valor máximo es de 83, y no entra en 80, por lo tanto no pasa el corte en un tramo, por lo tanto no está clasificado.
    Ayudas: usar pasaCorte, observar que el par de ciudades está justo atrás de todo, definir una función que arma la lista de tramos a evaluar.