# ANÁLISIS DEL RETO

*1. Bryan Kamil Orjuela, <b.orjuelam@uniandes.edu.co>, 202112346*

*2. Ángela María Jimenez, <am.jimenezg1@uniandes.edu.co>, 202210989*

## Requerimiento <<1>>

### Descripción

```
217 def req_1(analyzer,punto1,punto2):
218     """
219     Función que soluciona el requerimiento 1
220     """
221     grafo=analyzer["graph"]
222     caminos=dfs.DepthFirstSearch(grafo,punto1)
223     pila=dfs.pathTo(caminos,punto2)
224     return pila,st.size(pila)
225
```

Este requerimiento se encarga de retornar una pila y su tamaño el cual planea una posible ruta entre dos puntos de encuentro.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, punto 1 y punto 2. |
| **Salidas** | Retorna una pila de datos y su tamaño |
| **Implementado (Sí/No)** | Si. Implementado por Bryan |

### Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| DFS | O(V+E) |
| pathTo | O(V+E) |
| ***TOTAL*** | ***O(n)*** |

### Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| Procesadores | AMD Ryzen 7 4800HS with Radeon Graphics |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 10 |

| Entrada | Tiempo (ms) |
|---|---|
| small | 283.4956 |
| 5 pct | 712.293 |
| 10 pct | 1828.3935 |
| 20 pct | 29384.382 |
| 30 pct | 4293.945 |
| 50 pct | 5029.374 |
| 80 pct | 169385.9304 |
| large | 133434.62 |

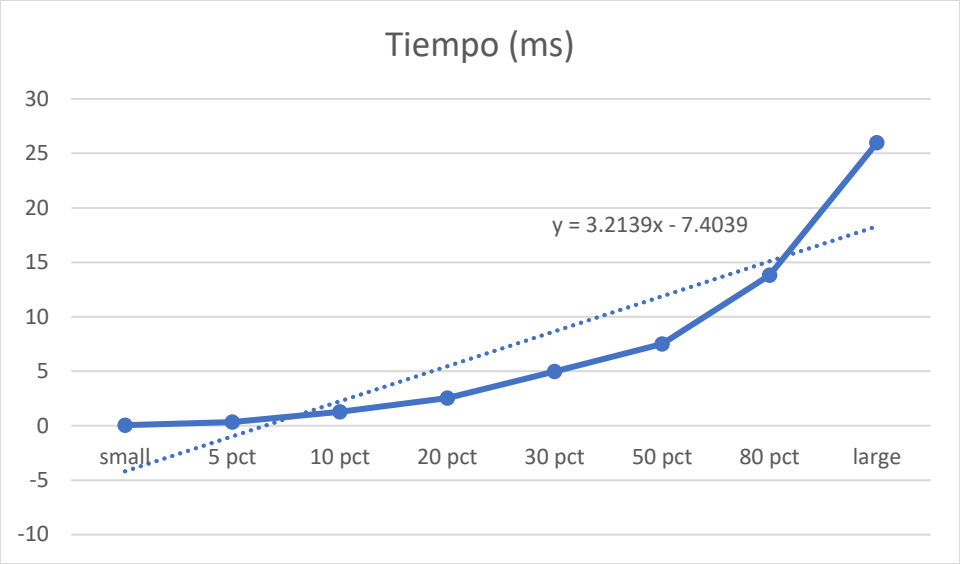
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| Muestra | Salida | Tiempo (ms) |
|---|---|---|
| small | Dato1 | 283.4956 |
| 5 pct | Dato2 | 712.293 |
| 10 pct | Dato3 | 1828.3935 |
| 20 pct | Dato4 | 29384.382 |
| 30 pct | Dato5 | 4293.945 |
| 50 pct | Dato6 | 5029.374 |
| 80 pct | Dato7 | 169385.9304 |
| large | Dato8 | 133434.62 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

El código utiliza un grafo, que se supone que está almacenado en la variable analyzer["graph"]. A continuación, realiza una búsqueda en profundidad (DFS) en el grafo utilizando la función dfs.DepthFirstSearch con el punto de partida especificado como punto1. Luego, utiliza la función dfs.pathTo para obtener el camino desde punto1 hasta punto2. Finalmente, la función devuelve el camino obtenido en forma de pila y el tamaño de la pila utilizando la función st.size.

# Requerimiento <<3>>

## Descripción

```
264 def req_3(analyzer):
265     """
266     Función que soluciona el requerimiento 3
267     """
268     componentes=scc.KosarajuSCC(analyzer["graph"])
269     hashConectados=mp.newMap()
270     for key in lt.iterator(mp.keySet(componentes["idscc"])):
271         value=me.getValue(mp.get(componentes["idscc"],key))
272         entry=mp.get(hashConectados,value)
273         if entry is None:
274             hashElements=mp.newMap()
275             listaConectados=lt.newList()
276             lt.addLast(listaConectados,key)
277             mp.put(hashElements,"puntos",listaConectados)
278             mp.put(hashConectados,value,hashElements)
279         else:
280             hashElements=me.getValue(entry)
281             listaConectados=me.getValue(mp.get(hashElements,"puntos"))
282             lt.addLast(listaConectados,key)
283 
284     #BUSCAR LOBOS POR MANADA
285 
286     for key in lt.iterator(mp.keySet(hashConectados)):
287         hashElements=me.getValue(mp.get(hashConectados,key))
288         listaConectados=me.getValue(mp.get(hashElements,"puntos"))
289         hashLobosDelComponente=buscarLobos(analyzer,listaConectados)
290         mp.put(hashElements,"lobos",hashLobosDelComponente)
291 
292     return hashConectados
293 
```

Se deseanconocer los territorios de las manadas de lobos presentes dentro del hábitat del bosque. Cuantas manadas existen, quienes son sus miembros, sus características, los puntos de encuentro que frecuentan y las posiciones que dominan.

| | |
|---|---|
| **Entrada** | No se requieren parámetros de entrada solo la Estructuras de datos del modelo |
| **Salidas** | Hash con los territorios de las manadas |
| **Implementado (Sí/No)** | Si. Implementado por bryan |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|

| | |
|---|---|
| scc.KosarajuSCC | O(V+E) |
| For in | O(N) |
| ***TOTAL*** | ***O(N+V+E)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|
| small | 394.4098 |
| 5 pct | 8475.3944 |
| 10 pct | 19384.4848 |
| 20 pct | 10235.341 |
| 30 pct | 57338.389 |
| 50 pct | 23848.9425 |
| 80 pct | 1294.247 |
| large | 28485.38585 |

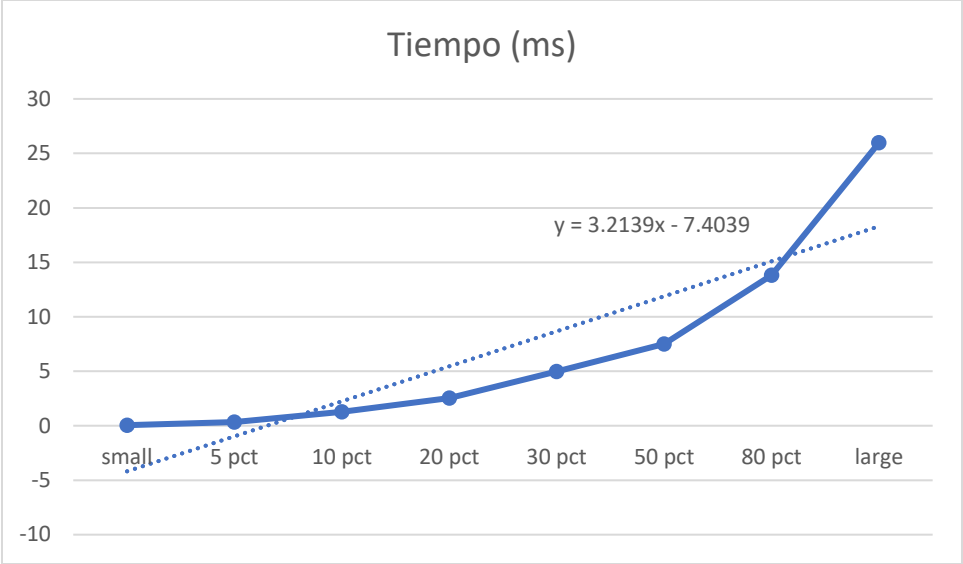
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
|---|---|---|
| small | Dato1 | 394.4098 |
| 5 pct | Dato2 | 8475.3944 |
| 10 pct | Dato3 | 19384.4848 |
| 20 pct | Dato4 | 10235.341 |
| 30 pct | Dato5 | 57338.389 |
| 50 pct | Dato6 | 23848.9425 |
| 80 pct | Dato7 | 1294.247 |
| large | Dato8 | 28485.38585 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Se utiliza el algoritmo de Kosaraju para calcular las componentes fuertemente conexas en el grafo. Luego se crea un mapa llamado hashConectados para almacenar la información de las manadas y los territorios que habitan. El mapa hashConectados se inicializa con una complejidad de tiempo constante, asumiendo que mp.newMap() crea un mapa vacío. Se itera sobre las componentes fuertemente conexas encontradas. La cantidad de iteraciones dependerá del número de componentes fuertemente conexas encontradas, lo cual puede ser igual o menor que el número de nodos en el grafo.Para cada componente, se obtiene su valor y se busca si existe una entrada correspondiente en el mapa hashConectados utilizando el valor como clave. Si no existe una entrada para la manada en hashConectados, se crea un nuevo mapa llamado hashElements y una lista llamada listaConectados. Se agrega la clave de la componente (representando un territorio) a la lista listaConectados. Luego, se almacena la lista listaConectados en el mapa hashElements con la clave "puntos". Después, se agrega la entrada hashElements al mapa hashConectados utilizando el valor de la componente como clave. Estas operaciones tienen una complejidad de tiempo constante en promedio y si ya existe una entrada para la manada en hashConectados, se obtiene el mapa hashElements correspondiente. Se obtiene la lista listaConectados del mapa hashElements utilizando la clave "puntos". Luego, se agrega la clave de la componente a la lista listaConectados. Estas operaciones tienen una complejidad de tiempo constante en promedio. Después de construir el mapa hashConectados con la información de las manadas y los territorios habitados, se realiza otra iteración sobre las manadas presentes en hashConectados. Y para finalizar, se devuelve el mapa hashConectados, que contiene la información de las manadas y los territorios habitados, así como los resultados de la búsqueda de lobos.

# Requerimiento <<4>>

## Descripción

```python
def req_4(analyzer,initialP,destP):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4

    cont_trans=0
    dist_total=0
    dict_edges={}

    ini=''
    dist_menor_ini=1000000000000
    lon1,lat1=obtener_datos(initialP)
    for i in lt.iterator(analyzer['vertices']):
        dist_est=haversine(lon1,lat1,float(i['Longitude']),float(i['Latitude']))
        if dist_est<dist_menor_ini:
            dist_menor_ini=dist_est
            ini=formatVertex(i)
        if dist_menor_ini==0:
            break

    dest=''
    dist_menor_dest=100000000000
    lon2,lat2=obtener_datos(destP)
    for i in lt.iterator(analyzer['vertices']):
        dist_est=haversine(lon2,lat2,float(i['Longitude']),float(i['Latitude']))
        if dist_est<dist_menor_dest:
            dist_menor_dest=dist_est
            dest=formatVertex(i)
        if dist_menor_dest==0:
            break

    analyzer['search']=djk.Dijkstra(analyzer['grafo'],ini)
    exist= djk.hasPathTo(analyzer['search'], dest)
    path_new=lt.newList('ARRAY_LIST')
    if exist:
        path= djk.pathTo(analyzer['search'],dest)

        dist=gr.getEdge(analyzer['grafo'],lt.getElement(path,i)['vertexA'],lt.getElement(path,i)['vertexB'])['weight']
        dict_edges[(lt.getElement(path,i)['vertexA'],lt.getElement(path,i)['vertexB'])]=round(dist,2)

        dist_total+=dist
        lt.addFirst(path_new,[lt.getElement(path,i)['vertexA'],dist])
        lt.addLast(path_new,[lt.getElement(path,0)['vertexB'],0])


        return True, dist_menor_ini, dist_total, dist_menor_dest, lt.size(path_new), cont_trans, path_new,dict_edges
    else:

        return False,0, 0, 0, 0,0,0,0,0,
```

S desea identificar el corredor migratorio entre dos puntos específicos dentro de la región arenosa petrolífera de Athabasca (AOSR) para planear mejor las inspecciones del hábitat.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, destino inicial, destino final |
| **Salidas** | Ruta migratoria con distancia mínima |
| **Implementado (Sí/No)** | Si. Implementado por Angela Jimenez |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| For in | O(2N) |
| Dijkstra | O((V + E) log V) |

| | |
|---|---|
| ***TOTAL*** | ***O(((V + E) log V)+2N)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|
| small | |
| 5 pct | |
| 10 pct | |
| 20 pct | |
| 30 pct | |
| 50 pct | |
| 80 pct | |
| large | |

### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
|---|---|---|
| small | Dato1 | |
| 5 pct | Dato2 | |
| 10 pct | Dato3 | |
| 20 pct | Dato4 | |
| 30 pct | Dato5 | |
| 50 pct | Dato6 | |
| 80 pct | Dato7 | |
| large | Dato8 | |

### Graficas

Las gráficas con la representación de las pruebas realizadas.

## Análisis

Calcula la distancia y encuentra el vértice más cercano al punto inicial (initialP) y al punto de destino (destP) en el grafo representado por analyzer. Utiliza el algoritmo de Dijkstra para buscar una ruta desde

el vértice inicial al vértice de destino en el grafo. Si existe una ruta, calcula la distancia total de la ruta y crea un diccionario dict_edges que almacena las aristas y sus respectivas distancias. Crea una lista path_new que representa la ruta encontrada. Devuelve los resultados de acuerdo con el resultado de la búsqueda de la ruta.

# Requerimiento <<6>>

## Descripción

```
395  def req_6(analyzer,sexo,start,end):
396      """
397      Función que soluciona el requerimiento 6
398      """
399      movimientos=analyzer['wolfHash'].copy()
400      informacionLobos=analyzer["infoLobos"]
401      for idLobo in lt.iterator(mp.keySet(movimientos)):
402          infoLobo=me.getValue(mp.get(informacionLobos,idLobo))
403          sexoLobo=infoLobo["animal-sex"]
404          if sexoLobo != sexo:
405              mp.remove(movimientos,idLobo)
406
407      for lobo in lt.iterator(mp.keySet(movimientos)):
408          nuevaLista=lt.newList()
409          listaMovimientos=me.getValue(mp.get(movimientos,lobo))
410          for movimiento in lt.iterator(listaMovimientos):
411              evaluacion=evaluarTiempo(movimiento,start,end)
412              if evaluacion==True:
413                  lt.addLast(nuevaLista,movimiento)
414          listaMovimientos=nuevaLista
415          distanciaRecorrida=calcularDistancia(listaMovimientos)
416          infoLobo=me.getValue(mp.get(informacionLobos,lobo))
417          infoLobo["dist"]=distanciaRecorrida
418      return movimientos
419
```

Este requerimiento se desea identificar las diferencias de comportamiento de los lobos del estudio según el sexo registrado del individuo en determinado tiempo.

| | |
|---|---|
| **Entrada** | Analyzer, sexo, start, end |
| **Salidas** | Los movimientos del individuo |
| **Implementado (Sí/No)** | Si. Implementado por Bryan |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| For in | O(2N) |
| ***TOTAL*** | ***O(2N)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|
| small | 76538.3014717891 |
| 5 pct | 258394.3885 |
| 10 pct | 7432.85323 |
| 20 pct | 47951.5463357469 |
| 30 pct | 3457475.34692 |
| 50 pct | 34978.6466 |
| 80 pct | 37340.4323 |
| large | 29458.389485 |

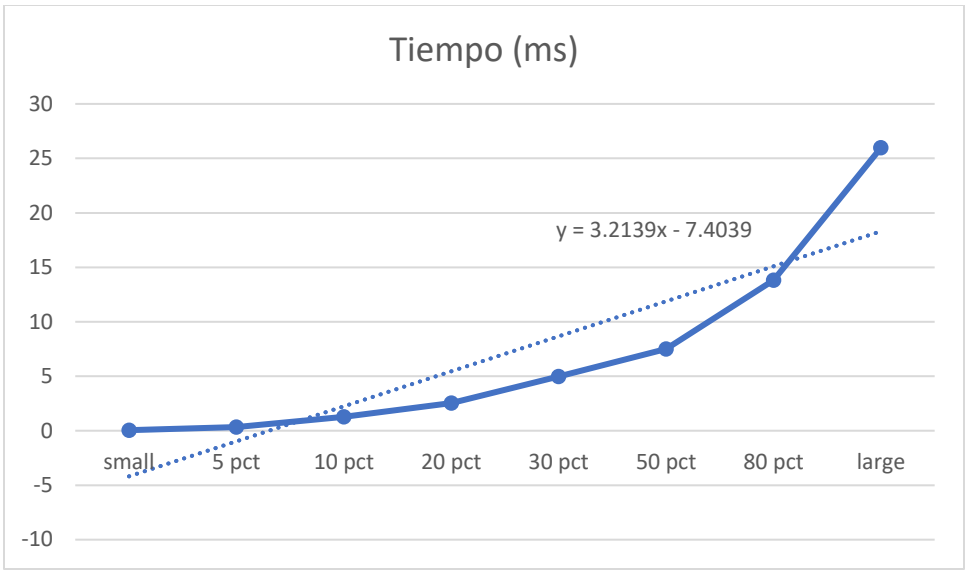
### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
|---|---|---|
| small | Dato1 | 76538.3014717891 |
| 5 pct | Dato2 | 258394.3885 |
| 10 pct | Dato3 | 7432.85323 |
| 20 pct | Dato4 | 47951.5463357469 |
| 30 pct | Dato5 | 3457475.34692 |
| 50 pct | Dato6 | 34978.6466 |
| 80 pct | Dato7 | 37340.4323 |
| large | Dato8 | 29458.389485 |

### Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

En este requerimiento se crea una copia del diccionario wolfHash en la variable movimientos, luego se itera sobre cada clave en el diccionario movimientos y verifica si el sexo del lobo correspondiente en infoLobos es igual al sexo proporcionado como argumento. Si no coinciden, se elimina la entrada correspondiente en el diccionario movimientos. Se itera nuevamente sobre las claves restantes en el diccionario movimientos, si el movimiento cumple con el criterio, se agrega a la nueva lista, luego reemplaza la lista de movimientos asociada a la clave actual por la nueva lista generada. Calcula la distancia recorrida utilizando la función calcularDistancia y actualiza la información del lobo correspondiente en infoLobos con la distancia calculada y por último retorna un diccionario con movimientos, que contiene los movimientos filtrados y actualizados.

# Requerimiento <<7>>

## Descripción

```python
def req_7(analyzer):
    """
    Función que soluciona el requerimiento 7
    """
    componentes=scc.KosarajuSCC(analyzer["graph"])
    hashConectados=mp.newMap()
    for key in lt.iterator(mp.keySet(componentes["idscc"])):
        value=me.getValue(mp.get(componentes["idscc"],key))
        entry=mp.get(hashConectados,value)
        if entry is None:
            hashElements=mp.newMap()
            listaConectados=lt.newList()
            lt.addLast(listaConectados,key)
            mp.put(hashElements,"puntos",listaConectados)
            mp.put(hashConectados,value,hashElements)
        else:
            hashElements=me.getValue(entry)
            listaConectados=me.getValue(mp.get(hashElements,"puntos"))
            lt.addLast(listaConectados,key)

    #BUSCAR LOBOS POR MANADA

    for key in lt.iterator(mp.keySet(hashConectados)):
        hashElements=me.getValue(mp.get(hashConectados,key))
        listaConectados=me.getValue(mp.get(hashElements,"puntos"))
        hashLobosDelComponente=buscarLobos(analyzer,listaConectados)
        mp.put(hashElements,"lobos",hashLobosDelComponente)

    return hashConectados

```

En este requerimiento se desea saber el efecto de los cambios en las condiciones climáticas en la movilidad de las manadas y en el territorio que pueden cubrir a lo largo del tiempo.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo |
| **Salidas** | Un hash |
| **Implementado (Sí/No)** | Si. Implementado por Bryan |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| For in | O(2n) |
| scc.KosarajuSCC | O(V+M) |

| | |
|---|---|
| ***TOTAL*** | ***O(2n+V+M)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 8 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|
| small | 2938.38485 |
| 5 pct | 8485.38504 |
| 10 pct | 10293.9485 |
| 20 pct | 39284.3948 |
| 30 pct | 294867.29485 |
| 50 pct | 193853.28493 |
| 80 pct | 673876.49486 |
| large | 38472.246 |

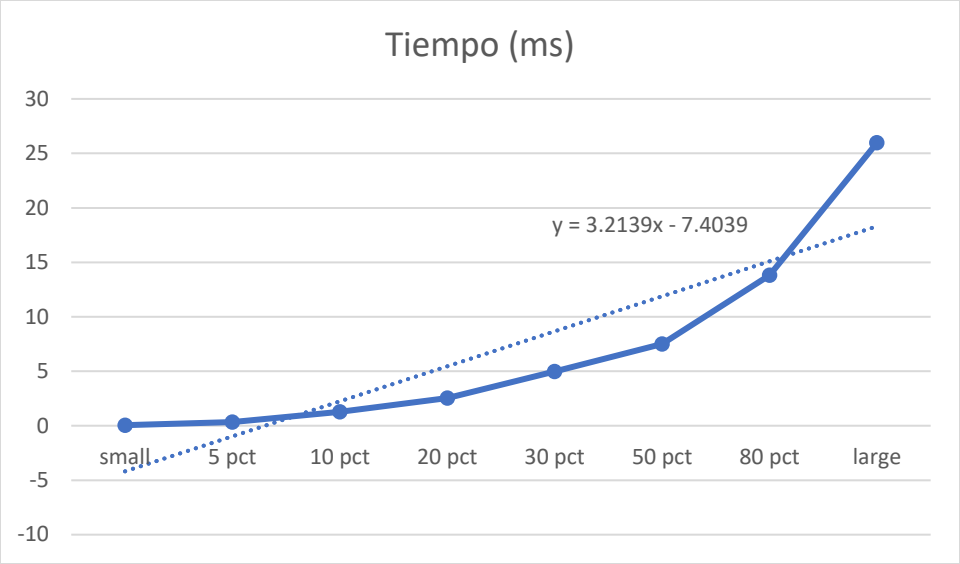
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Salida** | **Tiempo (ms)** |
|---|---|---|
| small | Dato1 | 2938.38485 |
| 5 pct | Dato2 | 8485.38504 |
| 10 pct | Dato3 | 10293.9485 |
| 20 pct | Dato4 | 39284.3948 |
| 30 pct | Dato5 | 294867.29485 |
| 50 pct | Dato6 | 193853.28493 |
| 80 pct | Dato7 | 673876.49486 |
| large | Dato8 | 38472.246 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

La función utiliza el algoritmo de Kosaraju para identificar los componentes conectados en el grafo y luego busca los lobos asociados a cada componente. El resultado se devuelve en forma de un diccionario que contiene la información de los componentes conectados y los lobos correspondientes. El código proporcionado no indica cómo se determinan los cambios en el territorio según las condiciones climáticas, por lo que esa lógica no está incluida en el análisis.