Autores e identificación

William Mendez – 202012662

Juliana Galeano – 202012128

Daniel Aguilera – 202010592

Boris N. Reyes R. – 202014743

PROYECTO, PARTE 2

Introducción

El propósito de este informe es mostrar la documentación del proyecto de curso en su segunda parte y brindar respuesta a los escenarios planteados en su enunciado. Se presentan tres componentes respectivos al algoritmo solución, análisis de complejidad tanto temporal como espacial, y la respuesta ante los escenarios de comprensión de problemas algorítmicos.

1. Algoritmo de solución

Primero se inicializa un diccionario en el cual se almacenarán todos los números que ya han sido conectados, a su vez se inicializa una variable en 0 que será el número de componentes. Luego el algoritmo utiliza la lista de strings que recibe por parámetro haciendo un for en el que se analizará cada una de las posiciones de la lista (se convierten a int), dentro del for se verifica que este vértice aún no esté en el diccionario de encontrados, si está se continua con otro, pero si no está inicializamos una variable booleana que se utilizará para saber si los vértices asociados al vértice actual hacen parte de un nuevo componente, también se guarda el vértice actual en el diccionario de encontrados, luego se hace un nuevo for que inicia desde la posición siguiente al vértice actual, cada nuevo vértice se compara con el vértice actual para saber si es menor y a la vez se verifica si ya está en el diccionario de encontrados, y cuando un vértice ya está en el diccionario la variable booleana se convierte en True, lo cual indica que todos los vértices asociados al vértice actual ya no hacen parte de un nuevo componente porque están conectados con un componente ya existente. Al final del ciclo interno se analiza la variable booleana y si es False le sumamos uno a la cantidad de componentes. Por último, cuando el ciclo más externo haya terminado imprimimos el número de componentes obtenidos.

Otras soluciones fueron planteadas, se usaron diccionarios y listas para llevar track de los componentes. Sin embargo, la solución descrita probó ser la más eficiente.

2. Análisis de complejidades espacial y temporal

Tiempos:

- Asignación (c_1)
- Comparaciones (c_2)
- +,*,-,/ (c_3)

Método lectura( )

```
def lectura():
    start = timer()
    nCasos = int(stdin.readline())
    # print(nCasos)
    while nCasos != 0:

        linea = stdin.readline().replace('\n', '')
        lista = linea.split(" ")
        # print("lista", lista)
        procesar(lista)

        nCasos -= 1

    elapsed_time = timer() - start
    print("Time: %.10f" % elapsed_time)
```

La complejidad está dada por la ecuación: $2C_1 + nCasos(3C_1 + C2 + O(nLista^2) + C_3)$ Debido al orden natural, se descartan constantes dando como resultado $O(nCasos\ nLista^2)$

Método procesar( )

```python
def procesar(lista):
    encontrados = {}
    componentes = 0
    for n in range(0, len(lista)):
        n1 = int(lista[n])
        if n1 not in encontrados:
            repetido = False
            encontrados[n1] = 0
            for i in range(n+1, len(lista)):
                n2 = int(lista[i])
                if repetido and (n1 > n2) and n2 not in encontrados:
                    encontrados[n2] = 0
                if (not(repetido)) and (n1 > n2):
                    if n2 in encontrados:
                        repetido = True
                    if n2 not in encontrados:
                        encontrados[n2] = 0
            if repetido == False:
                componentes += 1
    print(componentes)
```

$$2C_1 + nLista\left(4C_1 + 2C_2 + C_3 + iLista(4C_1 + 7C_2 + C_1)\right)$$

Por orden natural se descartan constantes, por otro lado, al ser $iLista = nLista - 1$ se aproximan, pero no llegan a ser iguales, ambos pueden ser vistos de forma que n = nlista, y m = iLista. En el peor de los casos el algoritmo llega a ser o(n*m)

Complejidad Temporal Ponderada:

$$O(nCasos\ (\text{n*m}))$$

Complejidad del algoritmo:

$$O(\text{n*m})$$

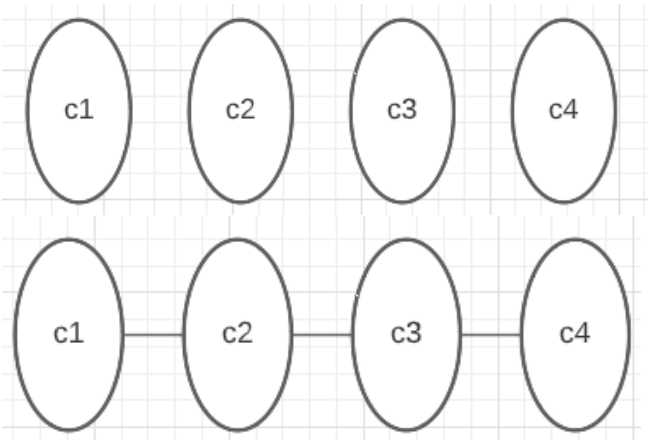
Complejidad espacial:

A lo largo de la solución se usaron las siguientes estructuras: Diccionarios y Listas.
Teniendo esto en cuenta es posible calcular la complejidad espacial, ya que sabemos la complejidad espacial de cada estructura. En el orden dado tenemos, O(n), O(n). En conclusión, es posible afirmar que la complejidad espacial es O(n) debido a las razones dadas.

3. Respuestas a los escenarios de comprensión de problemas algorítmicos.

3.1. Se le pide devolver como salida el número mínimo de aristas que se deben añadir para que exista un único componente conectado.

Para hacer no se debe realizar cambios al algoritmo, pues como ya calculamos la cantidad de componentes del grafo y sabemos que para unir los componentes de un grafo solo debemos añadir una arista para n-1 componentes y formar un solo componente.



3.2. Se mantiene el problema de obtener los componentes conectados, pero se cambia la instrucción de construcción de aristas así:

Una arista es creada para los vértices vi,vj (i<j) si y solo si pi<pj

Realizar esto no implica ningún reto nuevo para la solución planteada, ya que como podemos observar usamos un condicional a la hora de realizar las comparaciones con los demás elementos, teniendo en cuenta que i debe ser menor que j simplemente cambiamos este operador a < y de esta forma obtenemos el resultado esperado, el número de componentes conectados teniendo en cuenta que las aristas se crean si el elemento siguiente es menor al evaluado.

```
if repetido and (n1 > n2) and n2 not in encontrados:
    encontrados[n2] = 0
```