ISIS 1105 Diseño y Análisis de Algoritmos
Semestre 2022-10. Proyecto – PARTE 2
Entrega: viernes, mayo 13 de 2022 6:00PM (por Bloque Neon)

0 OBJETIVOS

- Diseñar soluciones computacionales para problemas.
- Estimar costos de las soluciones planteadas.
- Implementar soluciones.

Se premiarán las mejores soluciones y se castigarán las peores, en cuanto a eficiencia en tiempo y espacio.

1 CONDICIONES GENERALES

El proyecto se divide en tres partes independientes entre sí. Este documento describe la PARTE II. Cada parte contiene un problema a resolver mediante soluciones implementadas en `Java` o `Python`.

Para cada problema se pide:
- Descripción de la solución.
- Análisis temporal y espacial.
- Una implementación en Java o Python

2 DESCRIPCIÓN DEL PROBLEMA

A Grafo permutación

Un grafo no dirigido de n vértices ($v_1, v_2, \ldots, v_n$) es construido por un genio matemático según el valor de una permutación de los n primeros números naturales que le es concebida en sus sueños. La permutación le es suministrada como un arreglo ($[p_1, p_2, \ldots, p_n]$) y determina la creación de las aristas. Una arista es creada para los vértices v_i, v_j ($i < j$) si y solo si $p_i > p_j$.

Problema

Dado un grafo permutación definido mediante una permutación p_n el problema es calcular el número de componentes conectados (a.k.a. conexos) en este grafo.

Ejemplo:

Suponga la permutación $p = [2,1,4,3,5]$, de la cual se deduce que $n = 5$. El grafo correspondiente que genera esta permutación es:

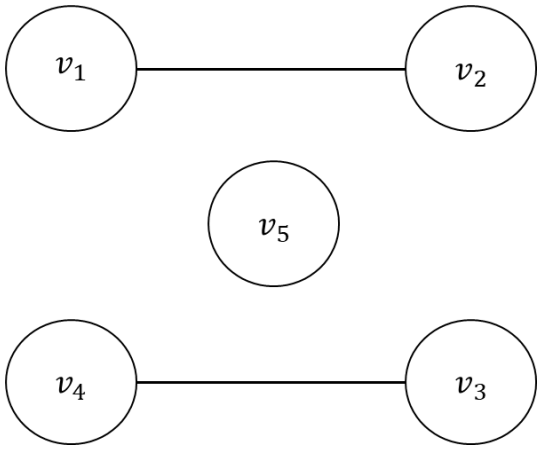

Fig 1. Grafo permutación resultante para $p=[2,1,4,3,5]$

Nótese que:

- Se crea la arista (v_1, v_2) ya que $p_1 > p_2$ $(2>1)$.
- Se crea la arista (v_3, v_4) ya que $p_3 > p_4$ $(4>3)$.

La respuesta a este problema es por lo tanto 3 ya que el grafo resultante lo constituyen 3 componentes conectados.

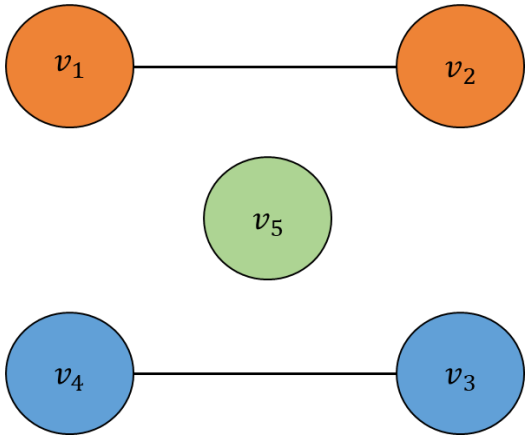

Fig 2. Componentes conectados resultantes. Cada color representa uno de los componentes conectados.

3 ENTRADA Y SALIDA DE DATOS

En todas las soluciones que se presenten, la lectura de los datos de entrada se hace por la entrada estándar; así mismo, la escritura de los resultados se hace por la salida estándar.

Puede suponer que ninguna línea de entrada tiene espacios al principio o al final, y que los datos que se listan en cada línea están separados por exactamente un espacio.

A continuación, se establecen parámetros que definen su tamaño y formato de lectura de los datos, tanto de entrada como de salida.

Descripción de la entrada

La primera línea de entrada especifica el número de casos de prueba que contiene el archivo. El programa debe terminar su ejecución, una vez termine de resolver la cantidad de casos de prueba dados por este número.

Cada caso este compuesto de una línea con *n*números enteros que representan la permutación: $p_1, p_2, \ldots, p_n \, (1 \leq p_i < n) \, \left(1 < n < 10^5\right)$

Descripción de la salida

Para cada caso de prueba, imprimir el número de componentes conectados.

Ejemplo de entrada / salida

Para el ejemplo de la Fig 1.

Entrada	Salida
1 2 1 4 3 5	3

Cinco casos de prueba:

Entrada	Salida
5 1 2 3 2 1 4 3 5 6 1 4 2 5 3 1 3 2 1 6 5 4	3 3 1 1 2

Nota: Se van a diseñar casos de prueba para valores de *n* muchos más grandes y dentro de los valores establecidos en el enunciado. Los casos mostrados en este documento son demostrativos de la estructura de entrada/salida esperada.

4 COMPRENSIÓN DE PROBLEMAS ALGORITMICOS

A continuación, se presentan un conjunto de escenarios hipotéticos que cambian el problema original. Para cada escenario debe contestar[1]: (i) que nuevos retos presupone este nuevo escenario -si aplica-?, y (ii) que cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?

ESCENARIO 1: Se le pide devolver como salida el número mínimo de aristas que se deben añadir para que exista un único componente conectado.

ESCENARIO 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 v_i, v_j $(i < j)$ si y solo si $p_i < p_j$.

Nota: Los escenarios son independientes entre sí.

5 ENTREGABLES

El proyecto puede desarrollarse por grupos de hasta tres estudiantes de la misma sección. La entrega se hace por bloque neon (una sola entrega por grupo de trabajo).

El grupo debe entregar, por bloque neon, un archivo de nombre `proyectoDalgoP2.zip`. Este archivo es una carpeta de nombre `proyectoDalgoP2`, comprimida en formato `.zip`, dentro de la cual hay archivos fuente de soluciones propuestas y archivos que documentan cada una de las soluciones.

5.1 Archivos fuente de soluciones propuestas

Todos los programas implementados en *Java* o en Python

Para el problema:

- Entregar un archivo de código fuente en *Java* (`.java`) o python (`.py`) con su código fuente de la solución que se presenta.
- Incluir como encabezado de cada archivo fuente un comentario que identifique el (los) autor(es) de la solución.
- Denominar `ProblemaP2.java` o `ProblemaP2.py` el archivo de la solución que se presente.

Nótese que, si bien puede utilizarse un *IDE* como *Eclipse* o Spyder durante el desarrollo del proyecto, la entrega requiere incluir solo un archivo por cada solución. El archivo debe poderse compilar y ejecutar independientemente (sin depender de ninguna estructura de directorios, librerías no estándar, etc.).

[1] NO tiene que implementar la solución a estos escenarios, el propósito es meramente analítico.

5.2 Archivos que documentan la solución propuesta

La solución al problema debe acompañarse de un archivo de máximo 3 páginas que la documente, con extensión `.pdf`. El nombre del archivo debe ser el mismo del código correspondiente (`ProblemaP2.pdf`).

Un archivo de documentación debe contener los siguientes elementos:

0 Identificación
Nombre de autor(es)
Identificación de autor(es)

1 Algoritmo de solución
Explicación del algoritmo elegido. Si hubo alternativas de implantación diferentes, explicar por qué se escogió la que se implementó.
Deseable:
Anotación (contexto, pre-, poscondición, …) para cada subrutina o método que se use.

2 Análisis de complejidades espacial y temporal
Cálculo de complejidades y explicación de estas. Debe realizarse un análisis para cada solución entregada.

3 Respuestas a los escenarios de comprensión de problemas algorítmicos.
Respuesta a las preguntas establecidas en cada escenario. NO tiene que implementar la solución a estos escenarios, el propósito es meramente analítico.

Téngase en cuenta que los análisis de 2 tienen sentido en la medida que la explicación de 1 sea clara y correcta. No se está exigiendo formalismo a ultranza, pero sí que, como aplicación de lo estudiado en el curso, se pueda describir un algoritmo de manera correcta y comprensible.

No describa un algoritmo con código GCL a menos que lo considere necesario para explicarlo con claridad. Y, si lo hace, asegúrese de incluir aserciones explicativas, fáciles de leer y de comprender.