Tarea 4
Grafos para python

Entrega: viernes 24 de junio a las 21:00 hrs.

1. Motivación

Los lenguajes de alto nivel existen, entre otras cosas, para facilitar el trabajo del programador y aumentar su productividad. Los lenguajes de alto nivel más comunes y desarrollados ofrecen una enorme variedad de librerías que permiten hacer, en pocas líneas de código, trabajos altamente complejos que finalmente significan muchas líneas de código compilado.

Para algunas de esas aplicaciones, la velocidad es fundamental. Eso implica que cualquier optimización en el desempeño puede ser un aporte: desde mejoras a los algoritmos a paralelización. Si el lenguaje además es interpretado —como `python`—, existe la posibilidad de optimizar escribiendo las librerías en un lenguaje distinto: los lenguajes interpetados pueden ser varias veces más lentos que el código de máquina para hacer lo mismo. Existen formas de escribir librerías en C o C++, compilarlas, y usarlas desde otro lenguaje (como `python`).

Diversas razones pueden hacer necesaria la escritura de librerías nuevas: p.ej. que se trate de algo muy específico, o que por la aplicación concreta se haga conveniente alguna optimización específica que no está presente en las librerías existentes.

En esta tarea, se le pide desarrollar una librería de grafos en C o C++ que pueda ser utilizada desde `python`. El trabajo se puede llevar a cabo en parejas.

1.1. Grafos

Un grafo es una estructura matemática compuesta de un conjunto de *nodos* y un conjunto de *arcos* que los conectan. Si el hecho de que haya un arco que conecta desde el nodo A al nodo B implica que también existe conexión desde el B al A, se dice que dicho arco es adireccional o (más habitualmente) *no-dirigido*. Cada nodo representa una entidad abstracta. Adicionalmente, se puede asociar costos o capacidades máximas a los arcos.

Los grafos se usan para la modelación de una gran gama de problemas en ingeniería, p.ej. los nodos pueden representar intersecciones de calles y los arcos los segmentos de vías que los conectan. Google Maps y Waze modelan la ciudad como un gran grafo para construir las rutas que luego ofrecerán a sus usuarios.

Se dice que existe un camino entre dos nodos A y B si A está conectado a B por un arco, o existe un tercer nodo C tal que existe un arco entre A y C y existe un camino desde C a B.

Existen varias formas de representar computacionalmente el conjunto de arcos de un grafo, con distintos resultados de desempeño dependiendo de la aplicación (y de la cantidad relativa de arcos respecto a la cantidad de nodos). Algunas de ellas se llaman: matriz de adyacencia, lista de adyacencia, matriz de incidencia, etc.

Concretamente, se le pide programar una clase `grafo` que represente grafos no-dirigidos y provea los siguientes métodos:

- un constructor sin parámetros, que cree un grafo vacío
- un constructor que reciba como parámetro un número entero que represente el número de nodos que tendrá inicialmente el grafo.
- un método `void add_arco(int d, int h)` que cree un arco entre los nodos `d` y `h`.
- un método `int add_nodo()` que agregue un nodo y retorne el número que lo identifica. Los nodos serán identificados por un número secuencial que comenzará en cero.
- un método `bool hay_arco(int d, int h)` que retorne `true` si existe un arco entre `d` y `h` o `false` en caso contrario.
- un método `bool hay_camino(int d, int h)` que retorne `true` si existe un *camino* entre `d` y `h` o `false` en caso contrario.

1.2. Librería para python

Una vez programada la clase, se sugiere generar la librería con una herramienta de *software* para generar interfaces como SWIG.

Como resultado, deberá crear un módulo precompilado que pueda ser importado y utilizado desde `python`. A modo de ejemplo, el programa 1 debiera ejecutar correctamente:

Programa 1: Ejemplo que debiera funcionar en python con la librería

```
from grafo import *

#Funcion para construir frases dependiendo de una
#variable booleana.
#Transforma un false en "no ". Si es true, retorna
   un
#string vacio.
def castellano(booleano):
    if not booleano:
        return "no "
    return ""

g=grafo(3)
g.add_arco(1,2)
g.add_nodo()
g.add_arco(2,3)

print castellano(g.hay_arco(2,1)) + "hay arco entre
   los nodos 2 y 1"
print castellano(g.hay_camino(1,3)) + "hay camino
   entre los nodos 1 y 3"
print castellano(g.hay_camino(0,3)) + "hay camino
   entre los nodos 0 y 3"
```

Y mostrar en la consola de `python` lo siguiente:

```
hay arco entre los nodos 2 y 1
hay camino entre los nodos 1 y 3
no hay camino entre los nodos 0 y 3
```

2. Evaluación y Entrega

El plazo para la entrega de la tarea vence impostergablemente el viernes 24 de junio a las 21:00 hrs.

Formato de entrega: Subir un solo archivo comprimido con todo el código fuente de su programa al buzón correspondiente en Canvas, con el nombre de archivo "APELLIDO1-APELLIDO2-Tarea3.tar.gz", que deberá contener los archivos de código fuente en C++ (archivos .cpp y .h, de ser necesario), reemplazando "APELLIDO1" y "APELLIDO2" según corresponda (ej. VON_NEUMANN-TURING-Tarea3.tar.gz). Los archivos compilados no serán tomados en cuenta, si se llega a subir solo un archivo compilado, este será ignorado, y será evaluado con nota 1.

Como parte de su tarea, deberá escribir un `makefile` que genere la librería automáticamente y que incluya los siguientes *flags*:

```
-std=c++11 -Wall -Wextra -Wundef -Werror -Wuninitialized -Winit-self
```

(puede asumir que SWIG estará instalado en el computador del ayudante corrector).

Al compilar deberá generar una librería de `python` llamada "grafo", con todos los archivos intermedios requeridos. Si este paso falla y se debe realizar la compilación de forma manual, o la librería no se llama como se indica se procederá a descontar 1 punto de la nota obtenida.

Aquellas tareas que no compilen de la forma normal serán evaluadas con nota 1. Las que compilen, pero se caigan durante la ejecución, serán evaluadas con nota máxima 3.

Su programa será evaluado con múltiples casos de prueba y deberá ser capaz de ejecutarlos todos de manera correcta. De fallar en algún caso de prueba serán descontados los puntos correspondientes a dicho caso.

3. Consideraciones de Trabajo

El trabajo en las tareas debe ser hecho solo por su grupo, por lo que cuide su tarea para que no sea copiada parcial o íntegramente por otros. Todas las tareas entregadas serán comparadas por un sistema automático de detección de plagios. Cualquier copia será penalizada, recibiendo el mismo castigo tanto quien copia como quien permite que le copien. También es considerada copia cualquier ayuda externa recibida directamente en la tarea,

sin importar si proviene de un alumno del curso, de la universidad, o de otro lugar. El castigo será establecido por el Consejo de la Facultad, siendo como mínimo un 1,0 de promedio en el curso.