Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2115 – Programación como Herramienta para la Ingeniería (I/2021)

Laboratorio 1 Parte a: estructuras de datos y técnicas de programación

Objetivos

- Aplicar variadas técnicas de programación y estructuras de datos para la resolución eficiente de problemas de programación.

Entrega

- **Lenguaje a utilizar:** Python 3.6 o superior
- **Lugar:** repositorio privado en GitHub. Recuerde incluir todo en una carpeta de nombre **L1a**.
- **Entrega: jueves 27 de mayo a las 23:59 hrs.**
- **Formato de entrega (ES IMPORTANTE EL NOMBRE DEL ARCHIVO):**
  - Archivo python notebook (**L1a.ipynb**) con la solución de los problemas y ejemplos de ejecución. Utilice múltiples celdas de texto y código para facilitar la revisión de su laboratorio.
  - Archivo python (**L1a.py**) con la solución de los problemas. Este archivo sólo debe incluir la definición de las funciones, utilizando exactamente los mismos nombres y parámetros que se indican en el enunciado, y la importación de los módulos necesarios. Puede crear y utilizar nuevas funciones si lo cree necesario. No debe incluir en este archivo ejemplos de ejecución ni la ejecución de dichas funciones. **No deje instancias del método `print()` en el archivo py**.
  - Todos los archivos deben estar ubicados en la carpeta **L1a**. No se debe subir ningún otro archivo a la carpeta.
- **Descuentos:** El descuento por atraso se realizará de acuerdo a lo definido en el programa del curso. Además de esto, tareas que no cumplan el formato de entrega tendrán un descuento de 0,5 ptos.

- **Laboratorios con errores de sintaxis y/o que generen excepciones serán calificadas con nota 1.0.**
- Si su laboratorio es entregado fuera de plazo, tiene hasta el **viernes 28 de mayo a las 11:59AM hrs** para responder el formulario de **entregas fuera de plazo** disponible en el Syllabus.
- Las discusiones en las *issues* del Syllabus en GitHub son parte de este enunciado.
- El uso de librerías externas que sean estructurales en la solución del los problemas no podrán ser utilizadas. Solo se podrán utilizar las que han sido aprobadas en las *issues* de GitHub.

Introducción

En esta parte del laboratorio deberán solucionar 4 problemas de programación, que pueden ser resueltos de manera eficiente utilizando estructuras de datos y/o algoritmos adecuados, en vez de un enfoque de fuerza bruta. Cada problema dentro de un grupo tiene el mismo puntaje.

En cada problema se indica con un ejemplo, cómo se debe llamar la función que resuelve el problema. Además, se indica como se deben recibir los *inputs*, los *outputs* y el formato de estos. Si no se siguen estas instrucciones, el revisor automático no revisará sus problemas (ver detalles a continuación).

Corrección

Para la corrección de este laboratorio, se revisarán dos ítems por problema, cada uno con igual valor en la nota (50%). El primero serán los contenidos y mecanismos utilizados para resolver cada uno de los problemas propuestos. De este modo, es importante que comenten correctamente su código para que sea más sencilla la corrección. Recuerde que debe utilizar las estructuras de datos y algoritmos adecuados a cada problema. Además, se evaluará el orden de su trabajo.

El segundo ítem será la correctitud de los resultados y el tiempo de ejecución de las soluciones entregadas, que serán verificadas mediante un revisor automático. Por cada problema se probarán distintos tamaños y valores de entrada, y para cada uno de estos se evaluará su correctitud. Además de esto, el tiempo de ejecución de sus algoritmos no debe sobrepasar cierto límite (serán entregados la semana del 3 de mayo). Por lo tanto, por cada tamaño de input, si el resultado entregado por el algoritmo no es correcto, o si el tiempo de ejecución supera el máximo indicado, su solución no obtendrá puntaje.

Problemas

I. Los Cristaloides y su red del metro (0,8 pts)

Los cristaloides son criaturas alienígenas fascinantes que tienen un nivel de inteligencia similar al de los humanos, pero presentan una visión particular: pueden **distinguir pero no identificar** colores. Por ejemplo, si le muestras dos cuadrados de colores distintos, separados, no sabrían decir si se trata del mismo color o no, pero si le pones uno junto (o parcialmente superpuesto) al otro, sabrían con certeza si es o no el mismo color. Esto les plantea inconvenientes a la hora de leer los planos del metro de su planeta natal, ya que como todas las líneas son del mismo color, se pierden con frecuencia en las combinaciones.

Para ayudar a estos seres, su misión es programar un algoritmo que, dada una red de metro, le indique a los cristaloides el color del cual deben pintar cada una de las lineas para que puedan ubicarse sin problemas, con la restricción de que la cantidad de colores utilizados debe ser mínima. La entrada a su algoritmo serán: i) una lista de listas, donde cada sublista indica las estaciones de una línea de metro (cada estación de la red tiene un identificador único dado por un número natural), y ii) un `set` de tuplas, donde cada tupla indica una combinación entre dos líneas de la red. La salida de su algoritmo debe ser una lista de enteros, donde cada posición de la lista indica el color (número natural) asignado a línea asociada a ese índice, es decir, si en la posición 3 de la lista hay un 5, significa que a la línea 3 se le asignó el color 5.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
lineas = [[0, 1, 2], [3, 4, 5, 6], [7, 8, 9, 10, 11, 12]]
combinaciones = {(0, 10), (1, 6)}
colores = asignar_colores(lineas,combinaciones)
print(colores)
```

Salida

```
[0, 1, 1]
```

II. Sopa de letras (0,8 pts)

Un restaurant especializado en sopas con servicio de delivery promete a sus clientes sopas de letras personalizadas, donde es posible leer un texto libre elegido por el cliente. Lamentablemente, debido al exceso de demanda, el servicio de delivery ha tenido problemas para mantener la "integridad" del texto, por lo que cuando las sopas llegan a los clientes, estas vienen con el texto desordenado. Con el fin de evitar la fuga de clientes, el restaurant ofrece a todos aquellos que hayan tenido problemas, una sopa gratis, si son capaces de comprobar que no es posible encontrar el texto personalizado.

Para lograr esto, su misión es programar un algoritmo que, dada una matriz de letras mayúsculas (lista de strings, todos del mismo largo, primer string indica la primera fila de la matriz) que representa una vista superior de la sopa de letras, encuentre todas las ocurrencias del texto personalizado considerando las 8 direcciones. En caso de encontrar ocurrencias, su algoritmo debe retornarlas como una lista de listas de tuplas, donde cada lista de tuplas contiene las posiciones de los caracteres del texto buscado, es decir, cada tupla de la lista indica la posición de un carácter, en formato (fila, columna), donde la coordenada (0,0) corresponde al carácter ubicado en la posición superior izquierda de la matriz.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
sopa = ["LAMXB", "AOEYF", "FCHTB", "GFKAR", "POSFD"]
texto = "HOLA"
posiciones = encontrar_ocurrencias(sopa, texto)
print(*posiciones, sep="\n")
```

Salida

```
[(2, 2), (1, 1), (0, 0), (0, 1)]
[(2, 2), (1, 1), (0, 0), (1, 0)]
```

III. Qué evaluaciones conviene hacer (0,8 pts)

Nadie duda que la etapa universitaria es una de las más exigentes y cansadoras en la vida, en particular en esta época de pandemia. Uno de los elementos que más problemas causa es la organización del tiempo de estudio, en particular aquel dedicado al desarrollo de evaluaciones. Con el fin de facilitar la vida de los futuros estudiantes, ud. deberá desarrollar un sistema de asignación de tiempo, que dada una lista de evaluaciones, su fecha de entrega y su ponderación en la nota final, entregue una programación para su desarrollo que maximice el promedio de notas. Para esto, puede asumir lo siguiente:

- Cada evaluación puede desarrollarse durante 1 día.
- No pueden desarrollarse 2 o más evaluaciones de manera simultánea.
- Cada evaluación puede entregarse en cualquier día menor o igual al límite.
- La ponderación total de las notas puede sumar un número natural arbitrario.
- Las evaluaciones entregadas reciben nota 1 y las no entregadas nota 0.

Para construir el sistema, programe un algoritmo que reciba una lista de tuplas, donde cada tupla almacena el nombre de la tarea, los días que restan para su entrega, y la ponderación que tiene esta en la nota final. La salida del algoritmo debe consistir en una lista que presenta el orden en que deben realizarse las evaluaciones, y en un número que represente la nota ponderada final que se obtendrá. Esta se calcula como la suma de la ponderación de cada evaluación multiplicado por su nota, dividido por la suma de las ponderaciones, el valor obtenido se debe aproximar a 2 cifras después de la coma.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
evaluaciones = [('Tarea4',9,15),('Control1',2,2),('I1',5,18),('Control3',7,1),
                ('Control2',4,25),('Taller1',2,20),('Tarea2',5,8),('Tarea3',7,10),
                ('Taller2',4,12),('Tarea1',3,5)]
orden, nota = programar_evaluaciones(evaluaciones)
print(orden)
print(nota)
```

Salida

```
['Tarea2','Taller1','Taller2','Control2','I1','Control3','Tarea3','Tarea4']
```

```
0.94
```

IV. Cómo ordenar el proceso de vacunación (0,8 pts)

Con el fin de privilegiar la inmunidad de los sectores más susceptibles a sufrir síntomas graves de COVID19, se le pide organizar el orden de vacunación de una comunidad de nativos indígenas del Amazonas. La información disponible de la población es sumamente rudimentaria y se reduce a una serie de anotaciones donde se indican comparaciones de edad entre los nativos. Específicamente, se tiene para cada nativo al menos una comparación con otro nativo, donde se indica cuál de los dos es mayor. Agregado a esto, se tiene conocimiento de que algunos registros son incorrectos, lo que puede llevar a situaciones donde un nativo es simultáneamente mayor y menor que otro. En base a esto, ud. debe construir un sistema que entregue todas las maneras en que es posible ordenar a los nativos de mayor a menos prioridad de vacunación, en base a sus edades. Además, si detecta anomalías en la información, como la descrita anteriormente, debe indicar aquellos nativos que presentan problemas en sus registros.

Para construir el sistema, programe un algoritmo que reciba una lista de tuplas, donde cada tupla codifica la relación de edad entre dos nativos (cada nativo es identificado con un número natural único). Específicamente, cada tupla codifica la relación $A > B$, donde A de es el primer elemento de la tupla y B el segundo. En el caso que sea posible generar ordenes de vacunación consistentes, la salida del algoritmo debe consistir en una lista de listas de enteros, donde cada una presenta un posible orden en que deben realizarse las vacunaciones y cada elemento de la lista representa a un nativo. En caso de no existir ordenes posibles, su algoritmo debe retornar una lista de enteros, que contenga los identificadores de los nativos que presentan inconsistencias en sus registros.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
relaciones = [(0,6),(1,2),(1,4),(1,6),(3,0),(3,4),(5,1),(7,0),(7,1)]
resultado = ordenes_vacunacion(relaciones)
print(resultado)
print()
relaciones = [(0,6),(1,2),(1,4),(1,6),(3,0),(3,4),(5,1),(6,3),(7,0),(7,1)]
resultado = ordenes_vacunacion(relaciones)
print(resultado)
```

Salida

```
[3, 5, 7, 0, 1, 2, 4, 6]
[3, 5, 7, 0, 1, 2, 6, 4]
[3, 5, 7, 0, 1, 4, 2, 6]
[3, 5, 7, 0, 1, 4, 6, 2]
[3, 5, 7, 0, 1, 6, 2, 4]
[3, 5, 7, 0, 1, 6, 4, 2]
[3, 5, 7, 1, 0, 2, 4, 6]
[3, 5, 7, 1, 0, 2, 6, 4]
[3, 5, 7, 1, 0, 4, 2, 6]
[3, 5, 7, 1, 0, 4, 6, 2]
...

[0, 3, 6]
```

Política de Integridad Académica

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Ejemplos de actos deshonestos son la copia, el uso de material o equipos no permitidos en las evaluaciones, el plagio, o la falsificación de identidad, entre otros. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica en relación a copia y plagio: Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Si un alumno (grupo) copia un trabajo, se le calificará con nota 1.0 en dicha evaluación y dependiendo de la gravedad de sus acciones podrá tener un 1.0 en todo ese ítem de

evaluaciones o un 1.1 en el curso. Además, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir un procedimiento sumario. Por "copia" o "plagio" se entiende incluir en el trabajo presentado como propio, partes desarrolladas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.