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

IIC2115 – Programación como Herramienta para la Ingeniería (II/2019)

Laboratorio 2 - Estructuras de datos

Objetivos

- Aplicar los contenidos de estructuras de datos, a través de la resolución de problemas específicos.

Entrega

- **Lenguaje a utilizar:** Python 3.6
- **Lugar:** repositorio privado en GitHub. Recuerde incluir todo en una carpeta de nombre **L02**.
- **Entrega: domingo 15 de septiembre a las 23:59 hrs.**
- **Formato de entrega:** archivo python notebook con nombre **solucionL02** (**solucionL02.ipynb**) y archivo python con el mismo nombre (**solucionL02.py**) con la solución de este enunciado. Los archivos deben estar ubicados en la carpeta **L02**. No se debe subir ningún otro archivo a la carpeta. Utilice múltiples celdas de texto y código para facilitar la revisión de su tarea. Los archivos **.ipynb** y **.py** deben contener la misma solución.
- **Descuentos:** se descontará 0.5 puntos por cada hora de atraso y fracción en la entrega final. Tareas que no cumplan el formato de entrega tendrán un descuento de 0.5 ptos.
- **Tareas con errores de sintaxis y/o que generen excepciones serán calificadas con nota 1.0.**
- Si su laboratorio es entregado fuera de plazo, tiene un hasta el **lunes 16 de septiembre a las 23:59 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.

Resolución de problemas

En este laboratorio deberán resolver una serie de problemas cortos sobre estructuras de datos. Cada problema es independiente de los demás y se espera que sean resueltos utilizando los conocimientos del capítulo 2 del curso. Para obtener el máximo de puntaje por problema, estos deben ser solucionados usando los tópicos indicados en cada uno de ellos. De lo contrario, sólo obtendrán una parte pequeña del puntaje.

Para entregar sus soluciones, debe definir una función llamada `problema_x()` (con minúsculas), donde `x` corresponde al número del problema. Además, cada función debe recibir como input y retornar su respuesta de la forma indicada en cada problema. Por ejemplo, Si el problema 1 requiere, un entero, un texto y una lista (en ese orden) y se le pide retornar un entero, debe crear la siguiente función:

```
def problema_1(entero, texto, lista):
    #Lógica y resolución
    return numero
```

Problemas

1 **Energía combinada (listas) - dificultad básica** (0.5 pts)

Un antiguo castillo ha decidido crear grupos competitivos con su ejercito en función de la energía de cada guerrero. El objetivo del castillo es construir grupos de 4 caballeros que sumen la misma energía. La energía de cada caballero se ve representada por un número entero no negativo. Para resolver el problema debe encontrar todos los grupos de 4 guerreros que sumen la misma energía solicitada. Un guerrero puede estar en más de un grupo. Usted va a recibir una lista con las energías de todos los guerreros y el valor de energía solicitada por equipo. Debe retornar la cantidad de grupos y su combinación.

Entrada:

```
Lista de enteros: [2,7,4,0,9,5,1,3]
entero: 20
```

Salida:

```
tupla de entero y lista de listas: (3,[[0,4,7,9],[1,3,7,9],[2,4,5,9]])
```

2 **Movimientos de juego (diccionarios) - dificultad básica** (0.5 pts)

Se desea construir un sistema para registrar las jugadas de un juego a base de piezas. El sistema debe recibir una lista de tuplas (jugador, pieza) y responder i) `JX: Ha realizado su primera jugada`, en caso que el jugador X no haya realizado una jugada previa, o ii) el nombre de la pieza, concatenado con un número entero que indica la cantidad de veces que ese jugador ha usado esa pieza. Para resolución del problema no se puede asumir que se conoce completamente la lista jugadas, es decir, estos son ingresos ocurren temporalmente. De este modo. Un ejemplo de ejecución del sistema es el siguiente:

Entrada:

```
Lista de tuplas de texto: [('J1','alfil'),('J2','perro'),('J1','gato'),
('J1','alfil'),('J3','alfil'),('J2','alfil'),('J1','alfil')]
```

Salida:

```
lista de texto: ['J1: Ha realizado su primera jugada',
'J2: Ha realizado su primera jugada', 'gato', 'alfil2',
'J3: Ha realizado su primera jugada', 'alfil1', 'alfil3']
```

3 **Camino máximo (grafos) - dificultad media** (1.0 pt)

Se tiene un conjunto con N listas de números hexadecimales (más información sobre qué es un número hexadecimal aquí), donde cada lista está ordenada de menor a mayor. Se debe encontrar el camino que maximiza la suma recorriendo elementos de las N listas, solo es posible cambiarse de una lista a otra lista utilizando un número común entre ambas. Un camino comienza desde el menor número de alguna de las listas y va avanzando hacia los mayores, solo se puede cambiar de lista al existir coincidencia de los número. Finalmente, se debe retornar una tupla con la lista de números hexadecimales que corresponde a la suma máxima y el valor numérico (en base 10) de la suma.

Nota: tanto las listas de la entrada como la lista de la salida contienen números hexadecimales ordenados de menor a mayor.

Entrada:

```
Lista de listas (N):
[['0', '1', '3', '5', '7', '8', 'a'],
```

```
 ['0', '1', '9', 'a', 'b', 'c', 'e', 'f'],
 ['2', '3', '4', '6', '7', '9', 'c']]
```

Salida:

```
tupla de lista y entero:
(['2', '3', '4', '6', '7', '7', '8', 'a', 'a', 'b','c', 'e', 'f'], 109)
```

El mismo ejemplo con números en base 10 sería:

Entrada:

```
[[0, 1, 3, 5, 7, 8, 10],
[0, 1, 9, 10, 11, 12, 14, 15],
[2, 3, 4, 6, 7, 9, 12]]
```

Salida:

```
([2, 3, 4, 6, 7, 7, 8, 10, 10, 11, 12, 14, 15], 109)
```

4 **Sopa de letras (libre, con criterio) - dificultad media** (1.0 pt)

Dada una lista de palabras, diseñe un método que busque estas palabras en una matriz de $\mathbf{M} \times \mathbf{N}$ letras. Encuentre todas las palabras que pueden ser formadas por una secuencia adyacente de letras. Note que es posible conectar letras con cualquiera de las 8 adyacentes. No se puede utilizar una letra más de una vez en una misma palabra.

Entrada:

```
Lista de textos: ['ARBOL', 'PAR', 'CASA','EL']
Lista de listas de textos (MxN):
[['A','S','A'],
['A','R','O'],
['C','L','B']]
```

Salida:

```
Listas de texto: ['ARBOL', 'CASA']
```

5 **La extraña enfermedad (árboles) - dificultad alta** (3.0 pts)

Un amigo ha decidido encontrar a todos sus parientes sanguíneos para estudiar una extraña enfermedad. Como buen programador, le ofreces tu ayuda. Lo primero que te das cuenta es que en su familia todos, sin excepción, han tenido 0, 1 o 2 hijos. De cada persona conoces el nombre, la edad, si es o no portador de la enfermedad y el nombre del padre [1]. Por último, como esta enfermedad solo afecta a hombres, no incluirás a las mujeres en tu programa. Por tanto, cada persona tiene un único progenitor. Con esto en mente, tu amigo te pide tres cosas:

5.1 **Construir árbol** (0.7 pts). Crea un árbol en donde almacene la información de los hombres de esta familia. Se debe cumplir que el más longevo es el nodo raíz.

Entrada:

```
Lista de listas, donde cada elemento es
[Nombre, edad, Portador, Padre]:


[[Francisco, 99999, False, None],
[Felipe, 22, True, Francisco],
[Pablo, 34, True, Felipe],
[Cristobal, 25, False, Felipe]]
```

Salida:

```
Retornar objeto Nodo raíz.
```

5.2 **Visualizar árbol** (0.3 pts). Cree una representación visual en donde se muestre el árbol hecho en la parte anterior. Para ello utilice la librería *networkx* (utilice el comando `pip3 install networkx`).

Entrada:

[1] None para el más longevo de la familia

```
Lista de listas (Entrada de parte 5.1)
```

Salida:

```
'Bonita' representación de networkx en formato png.
En clases explicaremos lo que es una representación 'Bonita'.
```

5.3 **Nivel con más portadores** (0.8 pts). Se define un nivel de la familia como todas las personas que están a la misma distancia del más longevo de la familia (el nodo raíz). Por ejemplo, si Pablo es hijo del más longevo estará en el nivel 1. Felipe, su nieto, estará en nivel 2 y así sucesivamente. Se te pide reportar el nombre de cada persona del nivel con más portadores de la enfermedad, así como el número de portadores de ese nivel [2].

Entrada:

```
Lista de listas (Entrada de parte 5.1)
```

Salida:

```
Tupla de entero y lista:  (2, [Felipe, Pablo])
```

5.4 **Riesgo de nueva enfermedad** (1.2 pts) ¡Una nueva enfermedad puede afectar a esta familia mientras más dispar sea el nivel de los más jóvenes! Se definen los más jóvenes como todos aquellos que no tienen ningún hijo. Tu amigo descubre que si la diferencia de nivel entre dos de los más jóvenes es mayor a 1 están en riesgo de contagiarse de esta nueva enfermedad. Por algún misterio de la ciencia, si todos los más jóvenes de la familia están en el mismo nivel o a lo más a 1 de distancia, la familia se declara segura. Se te pide hacer una función que retorne True si la familia está libre de riesgo.

Entrada:

```
Lista de listas (Entrada de parte 5.1)
```

Salida:

[2] Considere al más longevo como el nivel 0

`bool: True o False, según corresponda (en el ejemplo True)`

Corrección

Para la corrección de este laboratorio, se revisarán dos aspectos, 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 comente correctamente su código y que sea más sencilla la corrección. Recuerde que debe utilizar las estructuras de datos adecuadas a cada problema. Además, se evaluará el orden de su trabajo.

El segundo aspecto será la correctitud de los resultados y el tiempo de ejecución de las soluciones entregadas. Por cada problema se probarán distintos valores de entrada, y para cada uno de estos, el tiempo de respuesta no deberá ser superior a 1 segundo. Por lo tanto, para cada caso probado, si el resultado entregado por el algoritmo no es correcto, o si el tiempo de resolución supera 1 segundo, la respuesta será considerada como incorrecta.

Durante la ayudantía del 10 de septiembre se entregarán referencias para el tamaño del problema y el tiempo de ejecución esperado.

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.