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

IIC2115 – Programación como Herramienta para la Ingeniería (I/2020)

Laboratorio 2 - Estructuras de Datos

Objetivos

- Aplicar los contenidos de estructuras de datos para la resolución eficiente de problemas de programación

Entrega

- **Lenguaje a utilizar:** Python 3.6
- **Lugar:** repositorio privado en GitHub. Recuerde incluir todo en una carpeta de nombre **L02**.
- **Entrega: domingo 12 de abril a las 23:59 hrs.**
- **Formato de entrega:**
  - Archivo python notebook (**.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 (**.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. No debe incluir en este archivo ejemplos de ejecución ni la ejecución de dichas funciones.
  - Todos los archivos deben estar ubicados en la carpeta **L02**. No se debe subir ningún otro archivo a la carpeta. Los archivos **.ipynb** y **.py** deben contener la misma solución.
- **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 **lunes 13 de abril 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.

Introducción

En este laboratorio deberán solucionar 5 problemas de programación, que pueden ser resueltos de manera eficiente si se utilizan estructuras de datos, en vez de un enfoque de fuerza bruta. Para cada problema, se indicará la estructura de datos deberán obligatoriamente utilizar en su solución, o se indicará que el problema es de solución libre, y puede utilizar las estructuras que prefiera.

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.

Con el fin de facilitar el desarrollo, los problemas han sido agrupados de acuerdo a su dificultad (baja, media y alta). Cada uno de estos grupos tiene el mismo puntaje (2 ptos.), independiente de la cantidad de ejercicios que tenga. Dentro de cada grupo, cada ejercicio tiene el mismo valor.

Problemas

I. Dificultad baja (2.0 ptos.)

a) **Superficie de nivel (Listas):** considere un mapa de $M \times N$ celdas, donde cada una captura la altura con respecto al nivel del mar de una región geográfica, mediante un número natural $n \in [0, 255]$. Dada una celda ubicada en la posición (i, j), $0 \leq i < M$, $0 \leq j < N$, con valor de altura $n_{i,j}$, encuentre el conjunto de celdas que cubra la mayor superficie posible, sujeto a que contenga la celda (i, j) y a que todas tengan valor $n_{i,j}$. Su solución debe retornar una lista de tuplas, donde cada una contiene las coordenada de una celda. La lista debe estar ordenada en orden de filas, es decir, aparecen primero las celdas de las filas con menor índice, y dentro de una fila, se ordenan de menor a mayor de acuerdo a su columna.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
def superficie_de_nivel(mapa, celda):
    #solucion al problema


mapa = [[1,1,1,1],[1,2,4,4],[1,1,5,7]]
celda = (1,1)
celdas = superficie_de_nivel(mapa, celda)
print(celdas)
```

Salida

```
[(0,0),(0,1),(0,2),(0,3),(1,0),(2,0),(2,1)]
```

Tiempo de ejecución

Pendiente

b) **Anagramas (Diccionarios):** dos strings son anagramas si al reordenar los caracteres de uno, es posible formar el otro. Teniendo esto en consideración, escribe un programa que dados dos strings, responda si estos son o no anagramas.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
def son_anagramas(string1, string2):
    #solucion al problema


string1 = 'solucion'
string2 = 'oclusion'
s = son_anagramas(string1, string2)
print(s)
```

Salida

```
True
```

Tiempo de ejecución

Pendiente

II. Dificultad media (2.0 ptos.)

a) **El muro de los Ojibwe (Listas):** La aldea Ojibwe ha estado en guerra con el pueblo Menominee por muchos años. A raíz de estos constantes enfrentamientos, los Ojibwe construyeron un muro entre los pueblos para cesar las diferencias. El muro fue construido con unos bloques de piedra, uno al lado del otro. Los bloques de piedra son todos del mismo ancho pero de diferentes alturas. Ahora, los Ojibwe desean que el muro sea escalable a lo largo (como si de una escalera se tratase), de este modo los Ojibwe podrán subir al muro y observar a los Menominee. Para hacer esta modificación, los Ojibwe van a destruir bloques de los bordes del muro para que estos queden dispuestos (de izquierda a derecha) en una primera parte estrictamente creciente (escalera que solo sube) y luego estrictamente decreciente (escalera que baja). También es válida una escalera que solo sube o una escalera que solo baja. Además, el muro debe quedar lo más largo posible.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
def nuevo_muro(muro):
    #solucion al problema


muro_actual = [4, 6, 9, 5, 6, 10, 11, 9, 6, 4, 5]
nuevo_muro = nuevo_muro(muro_actual)
print(nuevo_muro)
```

Salida

```
[5, 6, 10, 11, 9, 6, 4]
```

Tiempo de ejecución

Pendiente

b) **El viajero en el tiempo (Solución libre):** considere un viajero en el tiempo, que desea llegar a un punto de la historia ubicado a N unidades de tiempo en el futuro. Usando su máquina del tiempo, el viajero es capaz de moverse a lo más k unidades de tiempo hacia el futuro, antes de detenerse para recargar las baterías de la máquina. Cada unidad de tiempo recorrida hacia el futuro consume una unidad de de energía de las baterías de la máquina.

Como nada es sencillo en la vida, debido a alteraciones en el espacio-tiempo, existen agujeros de gusano unidireccionales que evitan que el viaje al futuro sea lineal. Estas anomalías son conocidas y pueden llevar al viajero a momentos predefinidos del pasado o futuro, al costo de consumir toda la energía restante de su máquina.

En base a lo descrito anteriormente, y asumiendo que la carga de batería es instantánea y puede realizarse al terminar cualquier viaje, escriba un programa que dada una descripción de los agujeros de gusano unidireccionales (inicio y fin), la capacidad k de las baterías de la máquina y un punto N en el futuro, encuentre la cantidad mínima de energía que necesita el viajero para llegar al punto N, sin considerar la energía consumida por los agujeros de gusano.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
def viajero_en_el_tiempo(hoyos_de_gusano, k, N):
    #solucion al problema


agujeros_de_gusano = [(1,7),(5,2),(8,0)]
k = 3
N = 9
energia = viajero_en_el_tiempo(agujeros_de_gusano, k, N)
print(energia)
```

Salida

```
3
```

Tiempo de ejecución

Pendiente

III. Dificultad alta (2.0 ptos.)

La torre misteriosa (Solución libre): Pablo, un joven aventurero ha descubierto una misteriosa torre en medio del bosque, que a su izquierda tiene una pileta de agua. La torre tiene una infinidad pisos que se pierden en el cielo. Pablo ha decidido entrar y se da cuenta que cada piso tiene habitaciones, y que en cada una de ellas hay una serie de puertas en el muro de al fondo. De haber puertas, estas están ordenadas de izquierda a derecha y conducen a un elevador que lleva a una habitación del piso siguiente. Pablo hizo un mapa del lugar y enumeró todas las habitaciones identificando las habitaciones a las que puede llegar (desde la puerta izquierda a la derecha). Por ejemplo (7, 8, 9, 10) significa que la habitación 7 tiene elevadores a las habitaciones 8, 9 y 10.

Al salir, se para en la pileta y mira la torre, En cada piso, ve las ventanas de las habitaciones que dan hacia ese lado. ¿Qué habitaciones observa Pablo desde la pileta basado en su numeración (de abajo a arriba)?

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
def vista_desde_pileta(torre):
    #solucion al problema


mapa_torre = [(1, 2, 3), (2, 4), (3, 5, 6), (5, 7, 8)]
habitaciones = vista_desde_pileta(mapa_torre)
print(habitaciones)
```

Salida

```
[1, 2, 4, 7]
```

Tiempo de ejecución

Pendiente

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 comente correctamente su código para 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 ítem 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 se evaluará su correctitud. Además de esto, el tiempo de ejecución de sus algoritmos no debe sobrepasar el indicando en el enunciado para cada problema. Por lo tanto, para cada problema, si el resultado entregado por el algoritmo no es siempre correcto, o si el tiempo de ejecución supera el máximo indicado, su solución no obtendrá puntaje en este ítem.

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.