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

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

Laboratorio 2 - Estructuras de datos y algoritmos

Objetivos

- Aplicar variadas técnicas de programación 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 **L02**.
- **Entrega: miércoles 16 de septiembre a las 23:59 hrs.**
- **Formato de entrega (ES IMPORTANTE EL NOMBRE DEL ARCHIVO):**
  - Archivo python notebook (**L02.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 (**L02.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 **L02**. 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 **jueves 17 de septiembre 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 este laboratorio deberán solucionar 4 problemas de programación, que pueden ser resueltos de manera eficiente si utilizan estructuras de datos y/o algoritmos adecuados, en vez de un enfoque de fuerza bruta. Para cada problema, se indicará la estructura y/o algoritmo que deberán obligatoriamente utilizar en su solución.

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 fracción del puntaje.

Con el fin de facilitar el desarrollo, los problemas han sido agrupados de acuerdo a su dificultad (baja, media y alta). 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 el lunes 31 de agosto). 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. Dificultad baja (1 punto)

a) **Llaves (listas, stacks o colas):**

Portos es un famoso mosquetero, enamorado de la bella Cleopatra. Sin embargo, está en una encrucijada, ya que Cleopatra ha sido secuestrada por Hades y la única forma de salvarla es por la escalera al inframundo. El problema es que esta escalera esta protegida por diferentes bestias infernales, las cuales solo puede ser vencidas con espadas especiales para cada tipo. Por si esto fuera poco, después de derrotar a cada bestia, la espada usada se desvanece en el ether y no puede ser utilizada nuevamente.

Afortunadamente, Portos tiene un gran amigo, Rubio Hagrid, quien tiene un mapa del orden de las bestias a derrotar y de diferentes espadas tiradas en el camino de otros (intentos de) héroe que sucumbieron ante el poder del inframundo. Si bien este mapa es simple, muestra todo lo necesario. En él, se representa con letras minúsculas el tipo de las espadas tiradas (una sola antes de cada bestia) y con letras mayúsculas el tipo de bestia que sigue. Existen 26 tipos de bestias y de espadas, indicadas a continuación:

Espadas: 'a', 'b', 'c', ..., 'x', 'y', 'z'.

Bestias: 'A', 'B', 'C', ..., 'X', 'Y', 'Z'.

En base a la nomenclatura del mapa, para matar una bestia es necesario utilizar un tipo de espada denotado por la misma letra que la bestia. Por ejemplo, solo una espada de tipo 'j' puede derrotar a una bestia de tipo 'J'.

Dado que es posible que Portos no tenga acceso a todas las espadas necesaria al inicio de su viaje, Rubio Hagrid le indica que es posible comprar espadas de cualquier tipo en la armería Estigia, ubicada cerca de la entrada al inframundo. Si bien Portos es un romántico, tampoco le gusta gastar dinero extra, por lo que te pidió a ti que le ayudes a determinar la cantidad mínima de espadas extra que debe comprar en la armería para poder llegar al inframundo y rescatar a su preciada Cleopatra.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
mapa = 'fGgJjJhEdDiHaGcAbCdG'
min_espadas = numero_espadas(mapa)
print(min_espadas)
```

Salida

```
4
```

II. Dificultad media (3 puntos)

a) **El jinete de las montañas (A*):** En las imponentes montañas del oeste, Jaime, un habilidoso jinete, las recorre diariamente en búsqueda de leña. Al final de cada día, su caballo llega agotado, por lo que ha decidido crear una herramienta que le permita optimizar la energía consumida por su caballo al recorrer las montañas. Jaime cuenta con un mapa del perfil de alturas de las montañas, este se compone de una grilla rectangular de $m \times n$ casillas, donde cada casilla almacena la altura de esta. Dado que Jaime conoce muy bien a su caballo, sabe que este consume energía por desplazarse y subir a una celda de mayor altura. El caballo consume una unidad de energía por moverse a una casilla vecina. Además, a la hora de subir, gasta una unidad extra de energía por cada unidad de altura que sube. Por tanto, si su caballo se mueve de una casilla de altura h_1 a una casilla vecina de altura h_2 este consume $h_2 - h_1 + 1$ si $h_2 > h1$ o 1 si $h_2 \leq h_1$. Al estar en una casilla, Jaime ha determinado que se puede mover a cualquiera de las 8 casillas vecinas.

Tu objetivo es determinar la secuencia de casillas que minimiza la energía consumida por el caballo para moverse de un origen a un destino. Para ello usted recibe como *input* las coordenadas de la casilla de origen y de destino en forma de tuplas $(fila, columna)$ donde $fila \in [0, m-1]$ y $columna \in [0, n-1]$; y un mapa formado por una lista de listas de enteros. Su *output* debe ser la secuencia de casillas de la ruta óptima (lista de tuplas) y la energía total que consume el caballo (entero), ambos valores en una tupla.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

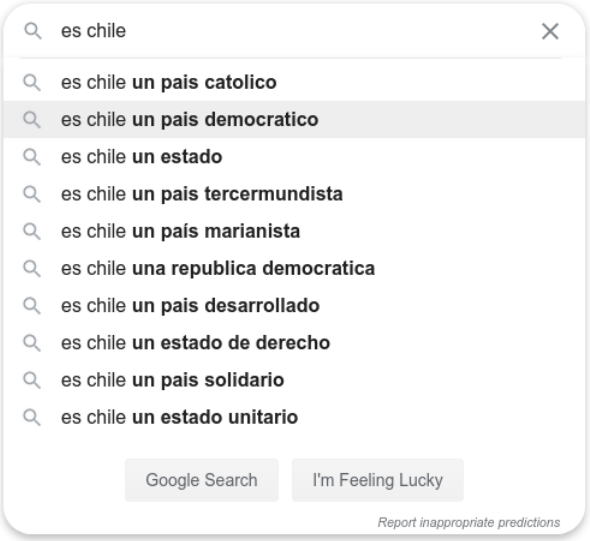
```
origen = (1,2)
destino = (5,6)
mapa = [[31, 27, 32, 24, 26, 34, 32, 24],
        [44, 3, 2, 46, 45, 6, 25, 38],
        [14, 9, 36, 39, 46, 15, 5, 31],
        [44, 18, 32, 34, 9, 3, 2, 1],
        [23, 10, 38, 46, 2, 19, 5, 39],
        [7, 44, 13, 40, 20, 1, 19, 25],
        [32, 9, 25, 41, 23, 14, 50, 50],
        [18, 35, 45, 17, 34, 35, 6, 37]]

ruta_minima = ruta_de_jaime(origen, destino, mapa)
print(ruta_minima)
```

Salida

```
([(1, 2), (2, 1), (3, 1), (4, 2), (5, 3), (5, 4), (4, 5), (5, 6)], 45 )
```

b) **LöbelFinder (Árbol):** Google te ha contratado para crear el nuevo complemento LöbelFinder e integrarlo en su clásico buscador. El objetivo de este complemento es retornar la mejor recomendación en base a las frecuencias de búsqueda anteriores. Por ejemplo, cuando buscamos `'es chile'` en el actual buscador aparecen las siguientes recomendaciones:



Para desarrollar este complemento, te han entregado un registro que almacena un histórico de búsquedas con sus respectivas frecuencias, por ejemplo:

Frecuencia	Búsqueda
5	el pais es grande
8	el gato con botas
4	el gatoman heroe
10	el

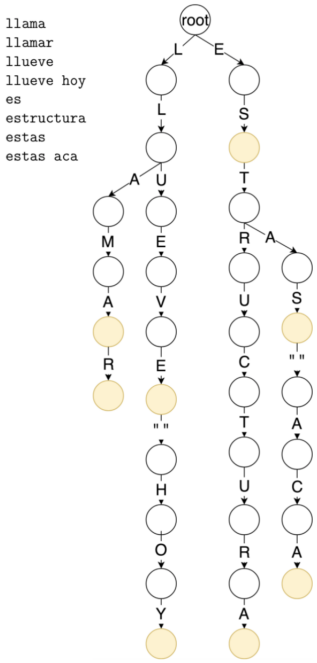
De la tabla se deduce que `el pais es grande` fue buscado un total de 5 veces.

Tu objetivo es responder para cada elemento de una lista de prefijos (es decir, inicios de frases) la frase que tenga la mayor frecuencia de usos históricos y que inicie con el prefijo a evaluar. Por ejemplo, usando los datos históricos de la tabla (para ver el formato de input del registro de históricos, revisar el ejemplo de código), si el input fuese `['el gato', 'el p']` la respuesta debiese ser `['el gato con botas', 'el pais es grande']`, dado que son las que más búsquedas tuvieron y empezaban con el prefijo del input.

Algunas consideraciones:

- No habrá entradas duplicadas en la base de datos.
- En caso de empate de frecuencia de dos o más *strings*, se deberá entregar cualquiera.
- Si el prefijo no está en la base de datos, el programa no debiese autocompletar, por lo que en este caso, deberá entregar el mismo prefijo.
- Si el prefijo es ya la frase de mayor frecuencia, se debe entregar el mismo prefijo.
- Si el prefijo está vacío (tiene largo 0) su programa debiera retornar la frase más frecuente de toda la base de datos.
- Los únicos caracteres válidos serán las letras minúsculas del castellano, sin tilde.

HINT: Para poder implementar una solución eficiente a este problema se recomienda usar una estructura de datos llamada **Trie**. Un **Trie** es una estructura tipo árbol que permite almacenar *strings* y hacer búsquedas eficientes sobre ellos. Para esto se almacenan los caracteres del *string* en las aristas que conectan los nodos. Además, se marcan los nodos donde termina un *string*. En la Figura de abajo se muestra un ejemplo de esta estructura en donde podemos notar, primero, que los nodos donde terminan las palabras están pintados con amarillo y segundo, que el carácter espacio está representado por " ".



Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
prefijos = ['llam', 'est', 'es']
historico = [(5, 'llama'),
             (6, 'llamar'),
             (12, 'llueve'),
             (4, 'llueve hoy'),
             (7, 'es'),
             (5, 'estructura'),
             (2, 'estas'),
             (6, 'estas aca')]
recomendaciones = motor_lobel(prefijos, historico)
print(recomendaciones)
```

Salida

```
['llamar', 'estas aca', 'es']
```

III. **Dificultad alta (2 puntos)**

a) **Traductor extraterrestre (Backtracking):** Desde un planeta muy lejano la sonda 6D2 de la empresa SpaceY ha detectado las primeras señales de lo que podría ser un mensaje de una civilización extraterrestre. Lamentablemente, a simple vista parecen ininteligibles dado que llegan todas las letras pegadas, sin espacios. Por suerte, luego de cada mensaje, la civilización extraterrestre envía la estructura escondida de cada mensaje. Por ejemplo, si el mensaje es '*gatocongato*' y la estructura '*A B A*', esto significa que '*gatocongato*' debe cumplir la estructura entregada. En este caso, una posible asignación es $A =$ '*gato*' y $B =$ '*con*', lo que descifraría el mensaje como '*gato con gato*'.

Su objetivo es determinar el significado de cada letra en el patrón utilizando el mensaje entregado por la civilización extraterrestre. Para ello usted recibe como *input* un *string* con el mensaje compuesto solo por letras y un *string* con la estructura compuesta por mayúsculas separadas por espacio. Su *output* debe ser una tupla de tuplas, donde cada tupla esta formada por la letra de la estructura y su significado.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
mensaje = 'elaguaeselaguaeselpez'
estructura = 'A B B C'
respuesta = traduccion(mensaje, estructura)
print(respuesta)
```

Salida (('A', 'el'), ('B', 'aguaesel'), ('C', 'pez'))

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.