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

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

Laboratorio 1

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 **L1**.
- **Entrega: jueves 28 de octubre a las 23:59 hrs.**
- **Formato de entrega (ES IMPORTANTE EL NOMBRE DEL ARCHIVO):**
  - Archivo python notebook (**L1.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 (**L1.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 **L1**. 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 29 de octubre 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 utilizando estructuras de datos y/o algoritmos adecuados, en vez de un enfoque de fuerza bruta. Cada problema tiene su puntaje indicado de manera individual.
En cada problema se indica con un ejemplo, cómo se debe llamar la función que resuelve el problema. Además, se indica el formato y cantidad de parámetros de entrada y valores de retorno. 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 escriban código ordenado y que lo comenten explícitamente para que sea corregido de forma adecuada.
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 4 de octubre). Por lo tanto, por cada tamaño de entrada, 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. Entrenamiento (1,0 pto.)

Un deportista amateur decide mejorar su nivel y diseña una esquema para lograr su objetivo: durante el primer día de su entrenamiento debe realizar exactamente 1 ejercicio, durante el segundo día 2 ejercicios, durante el tercer día 3 ejercicios, y así sucesivamente. De esta forma, durante el k-ésimo día debe realizar k ejercicios.

Como recurso adicional para facilitar el entrenamiento, el deportista amateur tiene una lista de n rutinas, donde la i-ésima consiste en a_i ejercicios. Durante cada día k, el atleta tiene entonces que elegir exactamente una de las rutinas que aún no ha realizado y realizar exactamente k ejercicios de esta, descartando aquellos ejercicios que no haya realizado. Este entrenamiento continua hasta que el atleta no tenga más rutinas con al menos k ejercicios a realizar para el día k.

Escriba un programa que, dada una lista que almacena la cantidad de ejercicios disponibles para cada rutina, calcule la cantidad máxima de días que puede entrenar el atleta si elige las rutinas de forma óptima.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
rutinas = [3, 1, 4, 1]
max_dias = entrenamiento_optimo(rutinas)
print(max_dias)
```

Salida

```
3
```

II. Ninjas (1,0 pts)

Como parte de su entrenamiento diario, un letal ninja ejercita su agilidad en un letal cañón. Este consta de dos paredes verticales paralelas, con una altura de n metros, que han sido divididas en segmentos de 1 metro de largo, enumerados con números enteros positivos de 1 a n de abajo hacia arriba. Algunos segmentos son seguros y el ninja puede escalarlos, mientras que otros son peligrosos, ya que están llenas de elementos afilados que no permitan que el ninja pueda estar allí.

Inicialmente, el ninja está en el segmento inferior de la pared izquierda y solo tiene permitido realizar una de las siguientes acciones por cada segundo de tiempo:

- Subir al segmento inmediatamente superior.
- Descender al segmento inmediatamente inferior.
- Saltar a la pared opuesta. Esto lleva al ninja al segmento que está exactamente k metros más alto que el segmento desde el que saltó. Formalmente, si antes del salto el ninja se encuentra en el segmento x de una pared, luego del salto se ubica en el segmento $x + k$ de la otra pared.

Si en algún momento el ninja intenta llegar a un segmento con un número mayor que n, entonces podemos asumir que el ninja salió del cañón y completó su entrenamiento.

Por si lo anterior fuera poco, el entrenamiento considera que el cañón se inunda y cada segundo el nivel del agua sube un metro. Inicialmente, el nivel del agua está en el borde inferior del primer segmento. Lógicamente, el entrenamiento prohibe que el ninja toque el agua.

Escriba un programa que, dada la descripción de los segmentos de cada pared del cañon y la altura a la que el ninja puede saltar, indique si es posible para el ninja terminar el entrenamiento.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

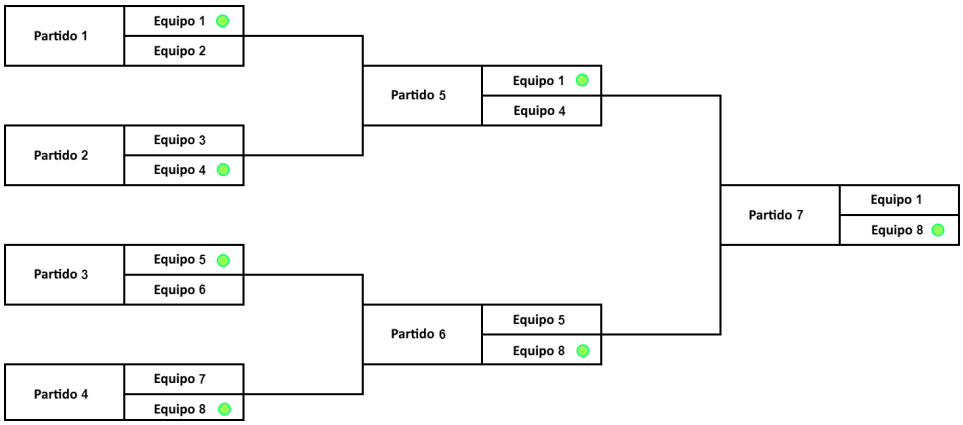
```
pared_izq = "---X--X"
pared_der = "-X--XX-"
altura_salto = 3
vive = entrenamiento_ninja(pared_izq, pared_der, altura_salto)
print(vive)
```

Salida

```
True
```

III. Los torneos incompletos (2,0 ptos.)

En un torneo tipo *playoff* participan 2^k equipos a través de k rondas de eliminación directa, como se muestra en la siguiente figura:



Con el fin de almacenar la información histórica de los *playoff* de la *Chilean Premier League*, se creó una codificación lineal binaria que indica para cada uno de los $2^k - 1$ partidos del *playoff*, qué equipo lo ganó: si fue el de la parte superior se almacena un 0 y si es el de la inferior, un 1. De esta manera, el campeonato de la figura anterior quedaría codificado de la siguiente manera: 0101011.

Si bien en un principio este esquema fue un éxito, el paso del tiempo hizo estragos sobre los documentos donde se almacenaba la información, lo que generó que algunos resultados de los partidos disputados se perdieran. De esta forma, al digitalizar estos registros, los partidos con información faltante fueron denotados con el carácter X. Dado que esta situación ha causado que no pueda saberse con seguridad quién ganó cada torneo, se decidió prorratear cada uno entre los posibles ganadores.

Escriba un programa que, dada una secuencia de caracteres dañada que define un torneo, calcule la cantidad de posibles equipos ganadores. Un equipo es posible ganador si se pueden reemplazar las X por valores de 0 o 1 tales que dicho equipo gane el torneo.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
torneo = "0110XX1"
num_ganadores = ganadores_torneo(torneo)
print(num_ganadores)
```

Salida

```
2
```

IV. El mejor pintor (2,0 ptos.)

Una fábrica de artículos de pintura decide organizar un concurso con el fin de mejorar sus ventas. En él, los participantes deben intentar pintar una cerca formada por n tablones verticales. Los tablones tienen un ancho de 1 metro y están numerados de izquierda a derecha, donde el i-ésimo tiene una altura de a_i metros. Teniendo esto en consideración, cada participante recibe una brocha de 1 metro de ancho, con la que pueden realizar trazos verticales y horizontales. El concursante que logra pintar la cerca con menos brochazos gana. Considere que no existe separación entre los tablones y que al hacer un trazo la brocha debe estar en contacto completamente con la cerca en todo momento.

Escriba un programa que, dada una lista con la altura de cada tablón, calcule la cantidad mínima de trazos necesaria para pintar la cerca completamente. Considere que es posible pintar dos veces la misma parte de la cerca.

Un ejemplo de ejecución del algoritmo es el siguiente:

Código

```
tablones = [2, 2, 1, 2, 1]
min_trazos = pintar(tablones)
print(min_trazos)
```

Salida

```
2
```

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.