

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

IIC1103 – Introducción a la Programación
2018 - 1

Examen

Instrucciones generales:

- La prueba consta de 4 preguntas con igual ponderación.
- Solo se reciben consultas sobre el enunciado en los primeros 30 minutos.
- No está permitido utilizar material de apoyo adicional al recordatorio adjunto.
- Todas las preguntas están pensadas para ser resueltas en forma sencilla con lo visto en clases. No es necesario utilizar otros mecanismos más avanzados.
- Si el enunciado de la pregunta no lo indica explícitamente, no es necesario validar los ingresos del usuario. Puedes suponer que no hay errores por ingreso de datos.
- Responde cada pregunta en una hoja separada, marcando tu Número de alumno y el número de pregunta en todas las hojas. Si usas más de una hoja en una pregunta, enumera las hojas; la numeración es por pregunta.
- Puedes utilizar cualquier cosa que te hayan pedido programar en un ítem anterior dentro de las preguntas (incluso si no respondiste ese ítem).
- Tienes cuatro horas para resolver la prueba.

Pregunta 1

Estás preparando tu siguiente semestre. Para evaluar cuáles cursos tomar, decides asignar a cada curso dos números entre 0 y 100. El primer número representa el beneficio que tiene para ti tomar el curso, que considera factores como ser prerequisito de otro curso, créditos, e interés. El otro número es el riesgo que el curso tiene para ti, que se relaciona con la cantidad de tiempo invertido, dificultad, o probabilidades de reprobarlo. Para realizar el análisis, decides crear un programa usando *programación orientada a objetos (OOP)*, para lo que crearás las clases `Curso`, `Semestre` y `Estudiante`, con los siguientes métodos:

a) `Curso`:

- **(5 puntos)** `__init__`: recibe la sigla del curso, y el beneficio y riesgo que tiene el curso para ti.

b) `Semestre`:

- **(5 puntos)** `__init__`: no recibe parámetros. El semestre debe guardar una lista de cursos, la cual debe comenzar vacía.
- **(5 puntos)** `agregar`: recibe una lista de cursos. Para cada uno, si no se encuentra en la lista de cursos del semestre, lo agrega. No retorna algún valor.
- **(15 puntos)** `calcular`: no recibe parámetros. Retorna una lista con dos elementos: el promedio de los beneficios de los cursos del semestre, y el promedio de los riesgos de los cursos del semestre.
- **(5 puntos)** `__str__`: no recibe parámetros. Retorna un string, que corresponde a las siglas de todos los cursos agregados al semestre, separadas por un espacio.

c) `Estudiante`:

- **(5 puntos)** `__init__`: recibe el nombre del alumno, el beneficio mínimo que considera que debe tener el semestre y su nivel de riesgo máximo tolerado.
- **(5 puntos)** `inscribir_semestre`: recibe un semestre, y se lo asigna al estudiante. No retorna algún valor.
- **(15 puntos)** `evaluar_semestres`: recibe una lista de semestres y usando los métodos de la clase semestre encuentra el mejor semestre para el estudiante, y se lo inscribe. Nota que el mejor semestre es un semestre cuyo riesgo es menor o igual al riesgo tolerado por el estudiante, su beneficio es mayor o igual al beneficio esperado por el estudiante, y que, cumpliendo lo anterior, tiene el máximo valor de **(beneficio - riesgo)**.

El código que escribas debe ser compatible con el siguiente programa:

```
c1 = Curso("IIC1103", 100, 50) # 100 = beneficio, 50 = riesgo
c2 = Curso("ABC3500", 80, 60)
c3 = Curso("ABC1212", 0, 10)
c4 = Curso("ABC2322", 20, 30)

s1 = Semestre()
s1.agregar([c1,c2,c3,c2])
print("s1>",s1.calcular())

s2 = Semestre()
s2.agregar([c1,c3,c4])
print("s2>",s2.calcular())

s3 = Semestre()
s3.agregar([c2,c4])
s3.agregar([c1,c3,c4])
print("s3>",s3.calcular())

e = Estudiante("Yo", 45, 40) # 45 = beneficio mínimo, 40 = riesgo máximo
e.evaluar_semestres([s1,s2,s3])
print(e.semestre)
```

El que imprime:

```
s1> [60.0, 40.0]
s2> [40.0, 30.0]
s3> [50.0, 37.5]
IIC1103 ABC3500 ABC1212
```

Pregunta 2

Te han encomendado analizar el funcionamiento de un taller mecánico en un mes. Para ello, usarás un *log*, que es un registro histórico de cada una de las reparaciones realizadas. Este registro se puede representar como una lista de listas, donde cada lista interna contiene información sobre:

- Auto: patente del auto reparado.
- Reparación: nombre de una reparación realizada al auto.
- Mecánico: nombre de la persona que realiza la reparación.
- Tiempo de inicio: minuto en el cual comienza la reparación.
- Tiempo de término: minuto en el cual termina la reparación.

Por ejemplo, el primer evento en el siguiente *log* registra la ejecución de la reparación `'A'` por el mecánico `'Samuel'` para el auto 'XK1234', que se inicia en el minuto 0 y termina en el minuto 2, y por ende dura dos minutos. Cabe señalar, que el *log* siempre mantendrá un orden por auto, y para cada auto, por orden temporal en que se realizan las reparaciones a cada auto.

```
log = [['XK1234', 'A', 'Sam', 0, 2], ['XK1234', 'B', 'Luis', 4, 10], ..., ['FABC04', 'A', 'Flo', 441, 447]]
```

Para analizar las reparaciones de los autos necesitarás implementar las siguientes funciones para un determinado *log*:

- **(10 puntos)** `cantidad_autos(log)`: Recibe un *log* y retorna la cantidad de autos que existen.
- **(10 puntos)** `reparaciones_realizadas_por(log, nombre)`: Recibe un *log* y el nombre de un mecánico, retornando el nombre de todas las reparaciones distintas realizadas por dicho mecánico.
- **(20 puntos)** `reparacion_mas_larga(log)`: Recibe un *log* y retorna el nombre de la reparación cuya duración promedio toma más tiempo.
- **(20 puntos)** `variantes(log)`: Recibe un *log* y retorna una lista con todas las variantes de reparación. Una variante es una secuencia **distinta** de actividades de reparación que se han realizado a distintos autos, ordenadas por el tiempo de inicio (exactamente en el orden en que vienen en el *log*). Se puede representar como una lista con el nombre de todas las reparaciones que hay en una variante.

Por ejemplo, el siguiente código:

```
log = [
  ['XK1234', 'A', 'Alejandro', 0, 1],
  ['XK1234', 'B', 'Florencia', 2, 10],
  ['XK1234', 'E', 'Samuel', 34, 35],
  ['XK1234', 'F', 'Rodrigo', 34, 35],
  ['DKJL21', 'A', 'Florencia', 447, 448],
  ['DKJL21', 'B', 'Luis', 447, 453],
  ['DKJL21', 'E', 'Samuel', 468, 469],
  ['GHRC32', 'A', 'Alejandro', 697, 698],
  ['GHRC32', 'B', 'Luis', 697, 704],
  ['GHRC32', 'E', 'Alejandro', 740, 741],
  ['FABC04', 'A', 'Samuel', 812, 813],
  ['FABC04', 'B', 'Luis', 812, 820],
  ['FABC04', 'C', 'Hugo', 824, 837] ]
```

```
print("Cantidad de autos:", cantidad_autos(log))
print("Reparaciones realizadas por Samuel: ", reparaciones_realizadas_por(log,"Samuel"))
print("Reparación más larga:", reparacion_mas_larga(log))
print("Variantes:")
for v in variantes(log):
        print(v)
```

Debería imprimir los siguientes valores:

```
Cantidad de autos: 4
Reparaciones realizadas por Samuel:  ['E', 'A']
Reparación más larga: C
Variantes:
['A', 'B', 'E', 'F']
['A', 'B', 'E']
['A', 'B', 'C']
```

Pregunta 3

Un *blanagrama* es una palabra que contiene las mismas letras y cada una en la misma cantidad que otra palabra pero en otro orden, excepto por el reemplazo de **un solo caracter** por otro (nota que un blanagrama siempre tendrá el mismo largo que la otra palabra). A continuación, se ilustra el concepto a través de algunos ejemplos.

- Ejemplo 1: un *blanagrama* de `CABELLO` es `BOTELLA` ya que contienen las mismas letras, excepto la `'C'` que se cambia por la `'T'`.
- Ejemplo 2: `PRODUCTORA` es *blanagrama* de `COMPUTADOR` ya que contienen las mismas letras, excepto que un caracter `'R'` se cambia por la `'M'` que sólo aparece en `COMPUTADOR`.
- Ejemplo 3: `DADO` es blanagrama de `LADO` (se cambia un caracter `'D'` por `'L'`), pero `DADO` **no** es *blanagrama* de `RARO`, dado que se diferencian en dos caracteres (habría que reemplazar dos `'D'`).
- Ejemplo 4: `ARREABA` es *blanagrama* de `BARRERA` ya que contienen las mismas letras, excepto que un caracter `'A'` de la primera se cambia por un caracter `'R'` de la segunda.
- Ejemplo 5: Las palabra `ROMA` y `AMOR` no son *blanagramas*, dado que no tienen ningún caracter diferente.

Escribe un programa que le pida al usuario una palabra, y escriba en un archivo `resultado.txt` todos los *blanagramas* de esa palabra que se encuentren en un diccionario. El diccionario es un archivo, llamado `diccionario.txt`, con una palabra por línea. Puedes asumir que las palabras siempre estarán en mayúsculas y sin tildes, tanto la ingresada por el usuario como las que se encuentran en el diccionario.

El diálogo con el usuario debe seguir el siguiente formato:

```
>> Palabra: COMPUTADOR
>> Hay 3 blanagramas, que han sido grabados en resultado.txt
```

Para el ejemplo anterior, dado el siguiente archivo `diccionario.txt`:

```
AGUA
COMPOSTURA
COMIDA
CONMUTADOR
COMPUTADOR
PRODUCTORA
PRODUCCION
VIAJE
```

El archivo `resultado.txt` contendría:

```
COMPOSTURA
CONMUTADOR
PRODUCTORA
```

Pregunta 4

Habrá un torneo de fútbol entre los participantes de varios *realities*, y el productor de uno te pidió hacer un programa que le permita crear el mejor equipo con los participantes de su *reality*. Sin embargo, sabes que los faranduleros suelen tener roces entre ellos, por lo que al momento de escoger los participantes de un equipo tendrás que maximizar sus habilidades y evitar los conflictos entre jugadores[1].

Cada equipo se compondrá de 5 jugadores. Cada jugador tiene un puntaje asociado que representa su habilidad. El puntaje de un equipo es la suma del puntaje de sus jugadores. Los jugadores pueden tener conflictos entre ellos. Un equipo conflictivo es aquel en el que hay al menos un par de jugadores con conflictos entre ellos.

Tu misión es formar el equipo no-conflictivo con mayor puntaje. Para ello cuentas con las siguientes variables:

- `puntajes`: Lista de enteros. `puntajes[i]` representa el puntaje asociado al i-ésimo jugador.
- `conflictos`: Lista de listas de enteros. Si `j` está en `conflictos[i]`, entonces existe un conflicto entre el i-ésimo y j-ésimo jugador.

Los jugadores se representan por enteros desde el *0* hasta *cantidad_de_jugadores_menos_1*. Un equipo se representa por una lista de enteros, donde cada elemento de la lista es el índice de un jugador. Por ejemplo, la lista `[0,1,4,5,9]` representa el equipo que está formado por los jugadores `0`, `1`, `4`, `5` y `9`.

Debes implementar (usando recursión) una función que encuentre el equipo no-conflictivo de mayor puntaje. Si existe más de un equipo con estas características, puedes entregar cualquiera de ellos. Si no existe el equipo, el resultado será la lista vacía. Por ejemplo, para la siguiente configuración:

```
puntajes = [88,49,66,24,42,47,44,41,26,20]
conflictos = [[2, 7, 3], [6], [0, 6], [0], [8], [], [1, 2], [0, 9], [4], [7]]
```

Una posible solución sería el equipo `[0,1,4,5,9]`, cuyo puntaje es 246 (pues 88+49+42+47+20=246). Una representación gráfica de lo anterior se puede ver en la Figura 1. Cada círculo es un jugador. El número dentro del círculo muestra el índice del jugador y el número bajo el círculo muestra el puntaje asociado a éste. Las líneas punteadas muestran los conflictos. Los jugadores seleccionados para formar parte del equipo están encerrados en el rectángulo. Se puede apreciar que no existe conflicto entre ellos.

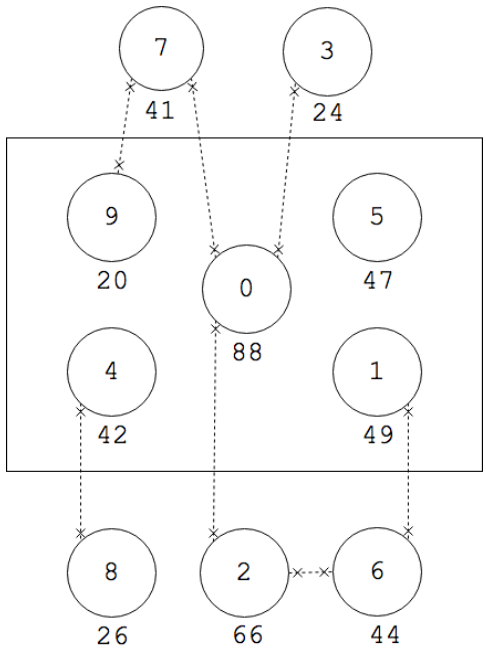


Figura 1: Representación gráfica de la situación

[1]Obvio, no querrás que se pongan a pelear entre ellos en mitad del partido