

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2013

IIC 1103 ⋆ Introducción a la Programación

Examen

Instrucciones

- Debes marcar tu número de alumno y el número de pregunta correctamente y con un lápiz NOTORIAMENTE OSCURO en cada una de las hojas de respuesta. Si tu número de alumno no se lee bien o está mal marcado en una hoja, tendrás UN PUNTO MENOS en la pregunta correspondiente.
- Los ejercicios deben ser respondidos en HOJAS SEPARADAS. No se aceptarán hojas de respuesta que contengan respuestas a dos ejercicios distintos.
- Puedes utilizar más de una hoja para responder una misma pregunta, pero tu número de alumno y el número de pregunta deben estar marcados correctamente en TODAS LAS HOJAS.
- Marca tu número de alumno por filas de arriba hacia abajo. Si tu número de alumno termina en j, deja la última fila en blanco.
- El tiempo disponible para realizar la evaluación es estrictamente de 3 horas, no se dará tiempo adicional.
- Durante la evaluación NO está permitido utilizar apuntes.
- NO está permitido el uso de aparatos electrónicos, los celulares deben estar apagados y no pueden ser consultados en ningún momento (ni siquiera para ver la hora). El tiempo restante se informará oportunamente.
- El puntaje de las preguntas se distribuye uniformemente, a menos que se indique lo contrario.

1. [10 puntos] La red social Pajaritos se basa en una lista de usuarios de la red, y relaciones entre estos usuarios, especificadas a través de la relación *seguir*. Cuando un usuario a está relacionado con un usuario b, se dice que el usuario a *sigue* al usuario b.

   La relación *seguir* no es necesariamente simétrica, lo que significa que es posible que el usuario a siga al usuario b, pero que el usuario b no siga al usuario a. Además ningún usuario se sigue a sí mismo.

   Estas características permiten identificar algunos perfiles interesantes como el perfil estrella. Éste es un tipo de usuario que tiene muchos seguidores, al menos el 50 % de la red, pero sin embargo él sigue a muy pocas personas (menos del 20 % de la cantidad que lo sigue a él). Estos usuarios provocan una alta carga en la red cada vez que publican un mensaje.

   Además existen categorías definidas por relaciones entre grupos de usuarios. Por ejemplo:

   - Amigos. Dos usuarios son amigos, si se siguen mutuamente.
   - Triángulos. Tres usuarios forman un triángulo cuando el primer usuario sigue al segundo, el segundo sigue a un tercero, y éste a su vez sigue al primero.
   - Comunidad. Tres o más usuarios forman una *comunidad* cuando todos son amigos entre sí. Note que una comunidad de tres amigos también forma un triángulo, pero no todo triángulo forma una comunidad de tres amigos.

   Considera que en la red hay `N` usuarios, y que cada usuario es representado por un número entre `0` y `N-1`. Los desarrolladores de la red social representan la relación de *seguir* utilizando una matriz de 1's y 0's de nombre `sigue` y dimensión `N`×`N`. Un usuario `a` sigue a otro usuario `b` cuando el valor `sigue[a][b]` es `1`. En caso contrario, el valor es `0`.

   Tu tarea será programar algunas funciones que permitan a los mantenedores de la red identificar a todos los usuarios o grupos de usuarios que cumplan alguna condición:

   a) [1 punto] `seguidos(sigue,a)`. Retorna una lista con todos los usuarios que son seguidos por el usuario `a`.

   b) [1 punto] `seguidores(sigue, a)`. Retorna una lista con todos los usuarios que siguen al usuario `a`.

   c) [1 punto] `estrellas(sigue)`. Retorna una lista con todos los usuarios *estrella* de la red social.

   d) [1 punto] `amigos(sigue,a,b)`. Retorna `True` si los usuarios `a` y `b` son amigos, y `False` en caso contrario.

   e) [2 puntos] `triangulo(sigue,a,b,c)`. Retorna `True` si los usuarios `a`, `b` y `c` forma un triangulo. Note que el orden en que los usuarios forman el triángulo puede ser cualquiera, no necesariamente `a`, `b`, `c`.

   f) [2 puntos] `triangulos(sigue)`. Retorna la cantidad de triangulos existentes en la red social.

   g) [2 puntos] `comunidad(sigue, miembros)`. Retorna `True` si los usuarios en la lista `miembros` forma una comunidad, y `False` en caso contrario.

2. [10 puntos]
Considera la siguiente función:

```
def misterio(frase,limite):
    inicio_linea = 0
    ultimo_espacio = -1
    i = 0
    while i < len(frase):
        if i - inicio_linea > limite:
            print(frase[inicio_linea:ultimo_espacio])
            inicio_linea = ultimo_espacio + 1
        if frase[i] == " ":
            ultimo_espacio = i
        i += 1
    print(frase[inicio_linea:])
```

a) [6 puntos] Explica qué hace el siguiente llamado a la función `misterio`:

```
misterio('Estamos en el examen de introduccion a la programacion',15)
```

Luego explica qué hace la función misterio en general.

b) [4 puntos] De acuerdo a tu respuesta en el punto *a*), ¿Es correcto lo que ocurre cuando el string `frase` contiene una palabra de largo mayor a `limite`? ¿Por qué?
Debes proponer e implementar una solución para que la función entregue un resultado razonable en este caso.

3. [12 puntos] En esta pregunta simularemos el tránsito de autos que van desde la Universidad hasta Viña del Mar. Para esto utilizaremos dos clases. La clase `Ruta` modela rutas por donde circulan autos. Tiene tres atributos: los enteros `vel_max` y `longitud`, que representan la velocidad máxima de la ruta en km/h y el largo de la ruta en km, y la lista `autos`, que contiene los autos que están sobre la ruta. La clase `Auto` tiene dos atributos: `velocidad` y `posicion`, que representan respectivamente la velocidad (en km/h) y posición del auto (en km) en la ruta donde éste se encuentra.

a) Implementa los constructores para ambas clases. El constructor de `Ruta` debe recibir como parámetro la velocidad máxima y la longitud, y debe inicializar el atributo `autos` como una lista vacía. El constructor para `Auto` recibe como parámetro la velocidad e inicializa en 0 la posición.

b) Implementa los métodos `agregar_auto` y `eliminar_auto` de la clase `Ruta`. Ambos métodos reciben un auto como parámetro y modifican apropiadamente el atributo `autos` para agregar o eliminar un auto de la ruta.

c) Implementa los métodos `acelerar` y `actualizar_posicion` de la clase `Auto`. `acelerar` recibe un número como parámetro e incrementa la velocidad del auto en dicho número. `actualizar_posicion` recibe un tiempo `t` en minutos y actualiza la posición del auto suponiendo que éste se mueve con velocidad constante (dada por el atributo `velocidad`) durante `t` minutos.

d) Debes escribir una función que reciba una lista de rutas (todas las rutas posibles entre los dos puntos) y un número n. Cada iteración de la simulación debe representar un minuto, y tu función debe simular 5 horas. Al comienzo de cada minuto saldrá de la universidad una cantidad aleatoria entre 0 y n autos. Cada uno tomará aleatoriamente una de las rutas, y su velocidad inicial será un número aleatorio entre la velocidad máxima de la ruta escogida y 30 km/h. En cada iteración deberás simular cuánto avanzó cada auto en dicho

minuto, considerando que su velocidad fue constante. Al final de cada minuto cada auto elige aleatoriamente entre estas 3 opciones: mantener su velocidad, aumentarla en 1 km/h o disminuirla en 1 km/h, sin exceder la velocidad máxima de la ruta ni llegar a menos de 30 km/h. Finalmente, debes eliminar de las rutas a los autos que llegaron al final, es decir aquéllos cuya posición es mayor al largo de la ruta.

Al finalizar, tu simulación deberá imprimir la máxima cantidad de autos que hubo en cada ruta.

4. [10 puntos] Algebraco es una aplicación que sirve para manipular expresiones algebraicas. Internamente, Algebraco representa las expresiones aritméticas como:

   - un número entero, o
   - una lista de tres elementos, donde tanto el primer como el segundo elemento son expresiones, y el tercer elemento es un operador. Los operadores permitidos son los strings `"+"` , `"-"` , `"*"` y `"/"`.

   Así, `4`, `[1,[3,4,"+"],"-"]` y `[[5,5,"/"],[2,1,"-"],"+"]` son expresiones permitidas en Algebraco que corresponden a 4, $(1-(3+4))$, y $((5/5)+(2-1))$ respectivamente.

   a) [7 puntos] Escribe una función `evalua`, que tome como argumento una expresión de Algebraco y retorne el número al cual esta expresión es igual. Por ejemplo,
   `evalua(4)` debe retornar 4,
   `evalua([1,[3,4,'+'],'-'])` debe retornar -6,
   `evalua([[11,7,'-'],[4,3,'+'],'*'])` debe retornar 28

   Ayuda: Para saber si la variable `a` es un `int`, puedes preguntar `if type(a) == int`.
   Más ayuda: Conviene que pienses en construir `evalua` como un procedimiento recursivo y que use el siguiente razonamiento:

   > *Cuando llamamos a* `evalua(expresion)` *pueden pasar dos cosas: que* `expresion` *sea un número o una lista. Si es un número, entonces se debe retornar ese número. En otro caso—es decir, si es una lista (de tres elementos)—es necesario evaluar ambos operandos y luego retornar el resultado de operarlos con la operación que se encuentra en la última posición de la lista.*

   b) [3 puntos] Informalmente, las *subexpresiones* de una expresión `E` son partes de `E` que al mismo tiempo son expresiones.

   Por ejemplo, si `E = [1,[2,3,"-"],"+"]`, las subexpresiones de `E` son `[1,[2,3,"-"],"+"]`, `1`, `[2,3,"-"]`, `2`, y `3`. Por definición, `E` es siempre una subexpresión de `E`.

   Escribe una función `cuenta` que reciba una expresión y un número, tal que `cuenta(exp,n)` retorne el número de subexpresiones de `exp` que, al evaluarse, tienen el valor `n`.

   Ejemplos:

   - `cuenta([1,1,"+"],2)` retorna `1` porque hay una subexpresión de `[1,1,"+"]` que es igual a `2` (la misma expresión).
   - `cuenta([1,1,"+"],1)` retorna `2` porque hay dos subexpresiones de `[1,1,"+"]`, en particular `1` y `1`, que son iguales a `1`.
   - `cuenta([[5,5,"/"],[2,1,"-"],"+"],1)` retorna `3` porque hay tres expresiones de `[[5,5,"/"],[2,1,"-"],"+"]`, en particular `[5,5,"/"]`, `[2,1,"-"]` y `1`, cuyo valor es igual a `1`.