

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

IIC1103 – Introducción a la Programación
2019 - 1

Examen

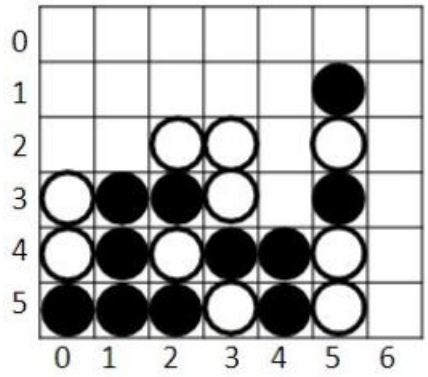
Instrucciones generales:

- La interrogación consta de 3 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**, anotando 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 de esa 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 tres horas y cuarenta minutos para resolver el examen desde que se entregan los enunciados.

Pregunta 1

Conecta Cuatro es un juego cuyo objetivo es alinear cuatro fichas en un tablero de seis filas y siete columnas. Para jugar, se necesitan dos jugadores, que van colocando sus fichas en el tablero de forma alternada. Para poner una ficha en el tablero, los jugadores escogen una de las columnas y la ficha se pone en la posición desocupada más baja de dicha columna. No se pueden realizar jugadas en columnas que estén con todas sus filas ocupadas.

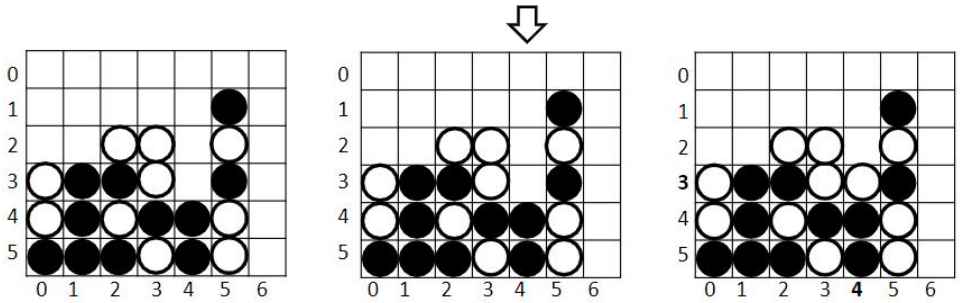
En esta oportunidad, debes implementar cuatro funciones: tres para ver si hay algún ganador luego de la última jugada, y la función que permite encontrar en cuál fila quedará una ficha que se inserta en una de las columnas. Un jugador gana si logra alinear 4 de sus fichas de forma consecutiva, ya sea en sentido horizontal, vertical o diagonal. Para representar el tablero, se utiliza una lista de listas, en donde los espacios vacíos son representados con un cero, y las fichas de los jugadores son representadas con un `1` para el primer jugador, y un `2` para el segundo jugador, como se muestra en la siguiente figura:



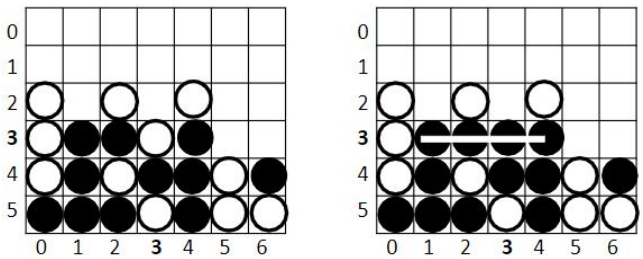

```
tablero = [
    [0,0,0,0,0,0,0],
    [0,0,0,0,0,1,0],
    [0,0,2,2,0,2,0],
    [2,1,1,2,0,1,0],
    [2,1,2,1,1,2,0],
    [1,1,1,2,1,2,0]
    ]
```

Específicamente, debes implementar:

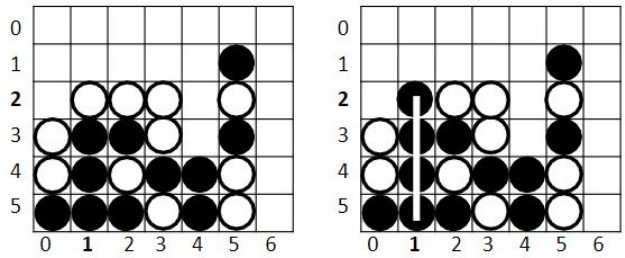
a) (15 pts) `insertar_ficha(c, t)`: Recibe una columna `c` y el tablero `t`. La función debe retornar el número de la fila en la cual quedaría ubicada la ficha si se pone en esa columna. En caso de que no se pueda poner, retorna -1. En el ejemplo de la imagen, el jugador seleccionó la columna 4, por lo que debe retornar 3. Esta función **no debe modificar o insertar la ficha en el tablero**.



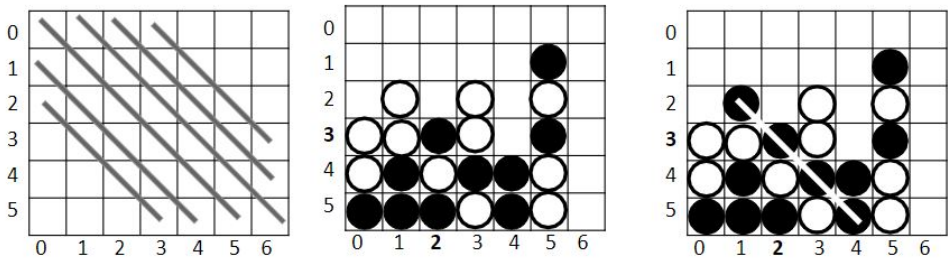
b) (15 pts) `revisar_horizontal(f, c, t)`: Recibe el número de la fila `f` y la columna `c` en donde se hizo la última jugada, y el tablero `t`. Debe retornar `True` si es que hay 4 fichas consecutivas del mismo color en sentido horizontal, y `False` en caso contrario. A continuación se muestran dos tableros distintos. En ambos casos, la función recibe como última jugada la fila 3 y la columna 3 (para la izquierda, el jugador de fichas blancas, y para la izquierda el de fichas negras). Para el caso de la izquierda, la función debe retornar `False`, y para el caso de la derecha la función debe retornar `True` (pues encuentra 4 fichas negras seguidas en sentido horizontal).



c) (15 pts) `revisar_vertical(f, c, t)`: Recibe el número de la fila `f` y la columna `c` en donde se hizo la última jugada, y el tablero `t`. Debe retornar `True` si es que hay 4 fichas consecutivas del mismo color en sentido vertical, y `False` en caso contrario. A continuación se muestran dos tableros distintos. En ambos casos, la función recibe como última jugada la fila 2 y la columna 1 (para la izquierda, el jugador de fichas blancas, y para la izquierda el de fichas negras). Para el caso de la izquierda, la función debe retornar `False`, y para el caso de la derecha la función debe retornar `True` (pues encuentra 4 fichas negras seguidas en sentido vertical).



d) (15 pts) `revisar_diagonal(f, c, t)`: Recibe el número de la fila `f` y la columna `c` en donde se hizo la última jugada, y el tablero `t`. La función debe retornar `True` si es que hay 4 fichas consecutivas del mismo color en la diagonal, y `False` en caso contrario. Debes revisar solo en el sentido diagonal desde arriba a la izquierda hacia abajo a la derecha. Nota que cada tablero tiene 6 posibles diagonales que contengan 4 fichas (como se muestra en la figura de abajo a la izquierda), pero basta con que la función revise sólo la diagonal que incluye la posición de la última jugada. A continuación se muestran dos tableros distintos (tablero ubicado en el centro y el ubicado a la derecha). En ambos casos, la función recibe como última jugada la fila 3 y la columna 2 (para la izquierda, el jugador de fichas blancas, y para la izquierda el de fichas negras). Para el caso de la izquierda, la función debe retornar `False`, y para el caso de la derecha la función debe retornar `True` (pues encuentra 4 fichas negras seguidas en sentido diagonal).



Pregunta 2

El DCColegio te ha contratado para que hagas un programa que se encargue de hacer todas las asignaciones de niños/as a talleres extraprogramáticos de forma automática.

Cada `Taller` tiene un nombre y una cantidad de vacantes disponibles. Cada `Estudiante` tiene un RUT, nombre, y una lista con las preferencias de los talleres que desea inscribir. El `DCColegio` recibirá las inscripciones de los estudiantes en orden de llegada y luego publicará los inscritos de cada taller.

A continuación, se describen las clases y métodos que tendrás que implementar. Puedes definir los atributos que quieras en cada clase o agregar métodos, pero se deben recibir exactamente los parámetros indicados. Nota que al final de este enunciado hay un código principal: tus clases y métodos deben funcionar con este código.

Clase Taller

a) (3 puntos) `__init__(nom, num)`: Crea un taller de nombre `nom` y número de vacantes disponibles `num`.

Clase Estudiante

a) (3 puntos) `__init__(rut, nom, p)`: Crea un estudiante con RUT `rut` y nombre `nom`, y lista de strings con las preferencias del estudiante `p`. La primera preferencia del estudiante (el taller que más quiere) será el nombre del taller que está en la posición `p[0]`, seguido del segundo que más quiere, y así sucesivamente hasta el final de la lista.

Clase DCColegio

a) (3 puntos) `__init__(t)`: Crea el colegio, el que ofrece los talleres de la lista de Talleres `t`.

b) (5 puntos) `agrega_insc(e)`: Recibe la inscripción de un Estudiante `e`. Es importante que los estudiantes se guarden en orden de llegada para poder asignar los talleres correctamente (los talleres se inscriben según el orden de llegada de las inscripciones de los estudiantes).

c) (15 puntos) `nombres_talleres()`: Retorna una lista de strings con los nombres de los talleres y su número de vacantes totales, ordenados del que tiene más vacantes totales al que tiene menos.

d) (31 puntos) `asignar()`: Realiza la asignación de estudiantes a talleres, retornando una **lista de listas** en la que **cada elemento** tiene el nombre de un taller, seguido del nombre de todos los estudiantes inscritos en éste: `[nombre_taller, [nombre_e1, nombre_e2, nombre_e3, ...]]`. Para realizar esto, el método debe satisfacer las preferencias de los estudiantes en el orden en que los estudiantes hicieron su inscripción, y según las preferencias de cada uno. Si ninguno de los talleres de su preferencia tiene cupo, entonces ese estudiante se quedará sin taller.

A continuación, se muestra un código principal que muestra un ejemplo de uso de las clases y métodos anteriores.

```
============= CÓDIGO PRINCIPAL =============
t1 = Taller("Karate", 2)
t2 = Taller("Atletismo", 1)
t3 = Taller("Danza", 3)
talleres = [t1, t2, t3]
e1 = Estudiante("25123123", "Amanda Arriagada", ["Karate", "Danza"])
e2 = Estudiante("27777222", "Bastian Bermudez", ["Karate", "Atletismo"])
e3 = Estudiante("24232323", "Camila Cancino", ["Karate", "Danza"])
e4 = Estudiante("23235342", "Damian Diaz", ["Danza", "Atletismo", "Karate"])
e5 = Estudiante("19933322", "Elena Escobar", ["Danza", "Karate"])
e6 = Estudiante("23442334", "Felipe Figueroa", ["Karate"])
dcc = DCColegio(talleres)
print("Talleres Disponibles:")
print(dcc.nombres_talleres())
dcc.agrega_insc(e1)
dcc.agrega_insc(e2)
dcc.agrega_insc(e3)
dcc.agrega_insc(e4)
dcc.agrega_insc(e5)
dcc.agrega_insc(e6)
r = dcc.asignar()
print("Estudiantes inscritos:")
for t in r:
    print(t)
```

El código anterior imprime lo siguiente:

```
Talleres Disponibles:
['Danza(3)', 'Karate(2)', 'Atletismo(1)']
Estudiantes inscritos:
['Karate', ['Amanda Arriagada', 'Bastian Bermudez']]
['Atletismo', []]
['Danza', ['Camila Cancino', 'Damian Diaz', 'Elena Escobar']]
```

Notemos que Camila, dado el orden de inscripción, se quedó sin cupo para su primera opción, pero con cupo para su segunda opción. Por otro lado, notemos que Felipe se quedó sin taller, pues todas sus opciones estaban sin cupo. No importa en qué orden se imprimen los talleres al imprimir los estudiantes inscritos.

Pregunta 3

Semestre tras semestre, hay gente que copia en las tareas creyendo que con solo cambiar el nombre de las variables no será detectada la copia. Por esto, se te pide hacer un programa que detecte estas copias.

Tu programa recibirá dos archivos: `codigoA.txt` y `codigoB.txt`, en donde cada uno representa un código a revisar. Los códigos son un conjunto ordenado de instrucciones, y cada una de estas instrucciones es de la forma `variable=int+int` (solo habrá ese tipo de instrucciones). Puedes asumir que en cada línea de los archivos, solo habrá una instrucción, que los nombres de las variables no se repiten dentro de un mismo archivo, que todas las operaciones tendrán solo 2 sumandos y que no habrá espacios. A continuación, se muestran dos archivos de ejemplo: el de la izquierda es el `codigoA.txt` y el del lado derecho es el `codigoB.txt`.

```
hola=1+2
mundo=2+3
var=12+13
aux=5+12
aux2=7+4
valor1=95+14
valor2=28+3
viva_chile=12+4
y=1+2
b=3+8
a=14+95
jug1=5+3
```

```
rango=1+2
y=2+1
contador=28+3
auxiliar=11+5
aumento=5+12
total=1+2
suma_total=13+12
indice=7+9
posicion_x=8+3
posicion_y=14+9
valor1=14+95
valor2=2+2
valor3=5+8
jug1=5+3
```

Se sospecha que dos instrucciones son copiadas si la operación es igual pero la variable se llama distinto (ejemplo: `a=10+2` y `b=10+2` son copia), o bien si la variable se llama igual y los operadores de la suma son los mismos pero están en distinto orden (ejemplo: `a=1+2` y `a=2+1` son copia).

Por último, se concluye la sospecha de que el código A fue copiado del código B si el 90 % de las instrucciones de A son copiadas del código B. No importa el orden de éstas. Además de esto, si hay dos instrucciones distintas de A que coinciden con una misma instrucción de B, solo se contabiliza una de ellas.

Debes escribir un programa que lea los dos archivos, y guarde un nuevo archivo `copiaAB.txt` en donde, en cada línea del archivo se debe poner qué fila del archivo `codigoA.txt` es copia de qué fila del archivo `codigoB.txt`, y finalmente si `codigoA.txt` fue copia de `codigoA.txt` o no. Para el ejemplo proporcionado anteriormente, se tiene que:

- La instrucción 1 de A es copia de la instrucción 1 de B, pues tienen la misma operación pero nombre cambiado.
- La instrucción 2 de A no es copia de B.
- La instrucción 3 de A no es copia de B. Notar que, en particular **no** se considera copia de la instrucción 7 de B.
- La instrucción 4 de A es copia de la instrucción 5 de B, pues tienen el mismo nombre de variable pero los operadores cambiados.
- La instrucción 5 de A no es copia de B.
- La instrucción 6 de A es copia de la instrucción 11 de B, pues tienen el mismo nombre de variable pero los operadores cambiados.
- La instrucción 7 de A es copia de la instrucción 3 de B, pues tienen la misma operación pero nombre cambiado.
- La instrucción 8 de A no es copia de B.
- La instrucción 9 de A es copia de la instrucción 6 de B, pues tienen la misma operación pero nombre cambiado.
- La instrucción 10 de A no es copia de B. Notar que, en particular **no** se considera copia de la instrucción 9 de B.
- La instrucción 11 de A es copia de la instrucción 11 de B, pues tienen la misma operación pero nombre cambiado. Sin embargo, como la instrucción 11 de B ya arrojó copia con la instrucción 6, no es contabilizada como copiada y no hay ninguna otra de B que coincida y no haya sido usada ya.

- La instrucción 12 de A es copia de la instrucción 14 de B, pues tienen la misma operación y el mismo nombre.

De un total de 12 instrucciones, solo 6 fueron copia, lo que significa que el 50 % ($\frac{6}{12} \cdot 100$) de las instrucciones eran copia. Luego, se concluye que A no era una copia de B, pues ese porcentaje es menor a 90 %. Finalmente, el archivo `copiaAB.txt` sería el siguiente:

```
linea 1 copia de linea 1
linea 4 copia de linea 5
linea 6 copia de linea 11
linea 7 copia de linea 3
linea 9 copia de linea 6
linea 12 copia de linea 14
codigoA.txt no es copia de codigoB.txt
```