1. Equivalencia de herramientas. Implementar las primitivas de los **semáforos** usando **regiones críticas condicionales**. **(2.5pt)**

2. El juego de la vida concurrente se juega en un tablero rectangular de n filas y m columnas. En cada casilla puede haber una célula o no. Cada célula mide su tiempo de vida en ciclos vitales. El objetivo de cada célula es permanecer con vida. Para decidir si una célula permanece con vida o no tras cada ciclo vital, es necesario contar cuántas células adyacentes tiene. Cada casilla tiene como máximo ocho posiciones adyacentes (incluyendo las diagonales). Si una célula tiene más de tres vecinas o menos de dos está en situación peligrosa. Una célula en situación peligrosa tiene la posibilidad de desplazarse una casilla en cualquier dirección. El desplazamiento sólo es posible a una casilla vacía. Una célula puede sobrevivir en situación peligrosa tres ciclos vitales. Si transcurridos los tres ciclos vitales sigue en situación peligrosa, entonces muere. Una célula examina su entorno de forma instantánea (en nuestra simulación). Por este motivo, las casillas adyacentes no pueden cambiar mientras dura el proceso de evaluación de la situación, pero otras células pueden examinar el contenido de esas mismas casillas. Evidentemente, dos células distintas no pueden moverse al mismo tiempo hacia la misma casilla.

Se pide simular el juego de la vida concurrente mediante un programa concurrente escrito en pseudocódigo utilizando **semáforos**. Utilizar un único tipo de proceso: célula. Para la simulación, cualquier estrategia de movimientos, incluido el movimiento aleatorio, está permitida. Para seguir la simulación, las células deben emitir unos mensajes por pantalla. Cada mensaje debe incluir el número de célula, la posición que ocupa y el número de ciclos vitales que ha vivido. Cada célula debe emitir un mensaje indicando en qué posición del tablero empieza. Esta posición inicial se debe elegir aleatoriamente entre las casillas vacías o bien al iniciarse el proceso, o bien en el programa principal. También deben emitirse mensajes cada vez que una célula decida moverse, especificando de dónde a dónde se produce el movimiento. Cuando una célula se encuentre en situación peligrosa debe emitir un mensaje explicando, además de los datos comunes a todos los mensajes, cuantos ciclos vitales lleva en situación peligrosa. Las células también deben emitir un mensaje antes de morir.

La simulación debe contar con unas estructuras de datos que permitan resolver razonablemente el problema. Además, se prestará atención para evitar las situaciones de interbloqueo, inanición y espera activa. Se debe procurar el máximo nivel de concurrencia entre los procesos. Si se identifica un problema-tipo (productores/consumidores o lectores/escritores) debe mencionarse explícitamente (incluyendo, en su caso, la prioridad más adecuada) antes de proceder con el pseudocódigo. Es muy importante que el comportamiento de las células se ajuste al enunciado, tanto en los aspectos específicamente concurrentes como en los demás. **(7.5pt)**

1. Equivalencia de herramientas: Implementar las primitivas de los **semáforos** usando **invocación remota** (o cita de Ada o rendez-vous) **(2.5pt)**

2. En la recepción de un hotel trabajan tres recepcionistas. Los clientes que quieren abandonar el hotel forman una cola. Cuando algún recepcionista puede atenderles, le dicen el número de habitación y se sientan en una silla a esperar la factura. Los recepcionistas calculan el importe mirando una tabla que indica cuánto se debe en cada habitación y mandan la factura a la cola de impresión. Mientras sale la factura, los recepcionistas siguen trabajando. La impresora simplemente recibe las peticiones de la cola y las imprime cuando puede. Los recepcionistas tienen también que recoger las facturas impresas y avisar a los clientes que están sentados cuando están listas. Cuando un cliente sentado recibe el aviso, deja la silla, coge la factura y se va. Sólo hay cinco sillas. Cuando todas las sillas están ocupadas, los recepcionistas no atienden a nadie más de la cola.

Se pide simular el comportamiento de los clientes, los recepcionistas y la impresora mediante un programa concurrente en pseudocódigo utilizando la herramienta **RCC**. Se debe intentar alcanzar el mayor grado de concurrencia posible, empleando varios recursos si se considera necesario. Se utilizarán únicamente tres tipos de proceso: cliente, recepcionista e impresora. Se prestará especial atención para evitar situaciones de interbloqueo e inanición, así como espera activa.

Entre los criterios de corrección se contarán los siguientes:

- ¿Son apropiadas las estructuras de datos para resolver el problema?¿Es posible resolver el problema utilizando las estructuras de datos propuestas?
- ¿Se alcanza un grado suficientemente alto de concurrencia?
- ¿Se evitan con éxito las situaciones de interbloqueo, inanición y espera activa?
- ¿Se utilizan herramientas de concurrencia no permitidas en el enunciado?¿Se mezclan diversos tipos de primitivas?
- ¿Se identifican e implementan correctamente subproblemas-tipo (productores y consumidores o lectores y escritores) con la correspondiente prioridad en su caso?

- ¿Se ajusta el programa al enunciado?
- ¿Hay problemas de sincronización, de concurrencia o de programación?

(7.5pt)

El juego del laberinto concurrente se juega en un tablero rectangular de **n** filas y **m** columnas. En cada casilla puede haber una pared, un espacio vacío o un coco. El objetivo de cada coco es salir del laberinto. Cada casilla tiene como máximo ocho posiciones adyacentes *(*incluyendo las diagonales*)*. Un coco se va moviendo dentro del tablero, siempre que tenga alguna casilla adyacente libre, hasta que consigue llegar a una de las casillas del rectángulo más externo, que se consideran la salida del laberinto. El desplazamiento sólo es posible a una casilla vacía. Cada coco mantiene una lista de las casillas por las que ha pasado. La siguiente casilla para un coco se calcula de la siguiente manera: Se analizan las direcciones de desplazamiento en el orden de las agujas del reloj, empezando por la dirección arriba y a la izquierda. Si la posición actual es *(*x,y*)*, entonces la primera dirección es la que lleva a *(*x-1, y-1*)*, después, la de *(*x-1,y*)*, a continuación la de *(*x+1,y*)* y así sucesivamente. La primera dirección que proporcione una casilla libre y que no haya sido visitada se utiliza para moverse. Si todas las casillas vecinas han sido visitadas, entonces se levanta esta restricción y el coco se mueve en la primera dirección, según el orden explicado, en la que haya una casilla libre. Si no se encuentra ninguna casilla libre adyacente, el proceso de evaluación de casillas vuelve a empezar. Las casillas adyacentes no pueden cambiar mientras dura el proceso de evaluación de la situación, pero otros cocos pueden examinar el contenido de esas mismas casillas. Evidentemente, dos cocos distintos no pueden moverse al mismo tiempo hacia la misma casilla.

Se pide simular el juego del laberinto concurrente mediante un programa concurrente escrito en pseudocódigo utilizando **semáforos**. Utilizar un único tipo de proceso: coco. Para seguir la simulación, los cocos deben emitir unos mensajes por pantalla. Cada mensaje debe incluir el número de coco, la posición que ocupa y las posiciones de las casillas que ha visitado. Cada coco debe emitir un mensaje indicando en qué posición del tablero empieza. Esta posición inicial se debe elegir aleatoriamente entre las casillas vacías, o bien al iniciarse el proceso, o bien en el programa principal. También deben emitirse mensajes cada vez que un coco decida moverse, especificando de dónde a dónde se produce el movimiento. Los cocos también deben emitir un mensaje antes de salir del laberinto.

La simulación debe contar con unas estructuras de datos que permitan resolver razonablemente el problema. Además, se prestará atención para evitar las situaciones de interbloqueo, inanición y espera activa. Se debe procurar el máximo nivel de concurrencia entre los procesos. Si se identifica un **problema-tipo** *(*productores/consumidores o lectores/escritores*)* debe mencionarse explícitamente *(*incluyendo, en su caso, la prioridad más adecuada*)* antes de proceder con el pseudocódigo. Es muy importante que el comportamiento de los cocos se ajuste al enunciado, tanto en los aspectos específicamente concurrentes como en los demás.

1. Equivalencia de herramientas. Implementar las primitivas de los **buzones** usando **semáforos**. *(2.5pt)*

2. Cuando un enfermo quiere a ir al médico, hace lo siguiente; primero necesita a que uno de los operadores esté libre. Hay cinco operadores. El operador le comunica la hora a la que debe presentarse. Para la misma hora se puede citar a varios enfermos. Los enfermos tienen tiempo libre y siempre van a la consulta antes de la hora, por ese motivo, a veces la sala de espera está llena. En la sala de espera caben cuatro personas. Si la sala de espera está llena, el paciente tiene que esperar en el pasillo. Si la consulta del médico está libre, entonces un enfermo de la sala de espera puede pasar. Para decidir qué enfermo pasa, primero se mira a ver quién estaba citado más pronto. Si hay empate, se mira entonces quién llegó primero a la sala de espera. El enfermo pasa a la consulta y al rato se va. Entonces la consulta queda libre de nuevo.

   Se pide un programa en pseudocódigo que simule la situación descrita usando **monitores**. Utilizar dos tipos de proceso: enfermo y operador. Para que la simulación sea inteligible, los enfermos y operadores deben emitir mensajes informativos por pantalla cada vez que su situación se modifique. De esta forma, mirando la salida deberemos poder tener acceso a toda la información. Esto quiere decir que cuando un enfermo reciba una hora de cita debe indicarlo por pantalla. Del mismo modo, cuando la sala esté llena también debe indicarlo. Cuando entre en la sala debe indicar cuánta gente hay ya en la sala. Cuando salga de la consulta también debe indicarlo... de forma que pueda verificarse que el comportamiento descrito es coherente con el enunciado.

   La simulación debe contar con unas estructuras de datos que permitan resolver razonablemente el problema. Además, se prestará atención para evitar las situaciones de interbloqueo, inanición y espera activa. Se debe procurar un elevado grado de concurrencia entre los procesos. Si se identifica un **problema-tipo** *(productores/consumidores o lectores/escritores)* debe mencionarse explícitamente *(incluyendo, en su caso, la prioridad más adecuada)* antes de proceder con el pseudocódigo. Es muy importante que el comportamiento de los enfermos y operadores se ajuste al enunciado, tanto en los aspectos específicamente concurrentes como en los demás.*(7.5pt)*