Estructura de datos y de la información
Boletín de problemas - Tema 7

1. Un concesionario de coches tiene un número limitado de `M` modelos, todos en un número limitado de `C` colores distintos. Cuando un cliente quiere comprar un coche, pide un coche de un modelo y color determinados. Si el coche de ese modelo y color no está disponible en el concesionario, se toman los datos del cliente (nombre y dirección), que verá atendida su petición cuando el coche esté disponible. Si hay más de una petición de un coche de las mismas características, se atienden las peticiones en orden cronológico. Se pide:

   a) Definir la estructura de datos más adecuada capaz de contener las peticiones de un modelo y color de coche.

   b) Definir la estructura de datos más adecuada, capaz de contener las peticiones para todos los modelos y colores de coche del concesionario.

   c) Implementar una función que, dado un cliente (nombre y dirección) que desea comprar un coche de un modelo y color determinado, coloque sus datos como última petición de ese modelo y color.

   d) Implementar una función que, dado un modelo del que se han recibido `k` coches de un determinado color, elimine los `k` primeros clientes de la lista de peticiones de ese coche y los devuelva en un vector, sabiendo que $k \leq 20$.

2. Supongamos que sólo podemos definir vectores hasta un tamaño dado por la constante `MAXIMO`. Sin embargo queremos poder manejar colas de mayor longitud.

   a) Definir la estructura de datos necesaria para poder utilizar colas con una capacidad de `2*MAXIMO` elementos. Utilizar para ello dos vectores de tamaño MAXIMO.

   b) Implementar las operaciones del TAD Cola utilizando la estructura definida en el apartado anterior.

3. Se desea tener una estructura de datos cola que tenga disponible en todo momento la cantidad de elementos que contiene. Se pide:

   a) Modificar la estructura de datos cola para poder almacenar en todo momento su longitud, sabiendo que se trata de una estructura enlazada.

   b) Implementar las operaciones del TAD Cola, añadiendo una que devuelva la longitud de la misma.

4. Una librería registra las peticiones de cualquier libro que no tiene en ese momento. La información de cada libro consiste en el titulo, el precio, el número de libros en *stock* y las peticiones del libro en estricto orden de llegada. Cada petición consiste en el nombre de una persona y su dirección.

   a) Define los tipos de datos necesarios para contener toda esta información para un máximo de 2000 libros.

   b) Implementa una operación que dado un cliente que pide un libro, vea si hay en stock, y si quedan, actualice el *stock* con la venta de ese libro, y si no, guarde los datos del cliente como última petición de ese libro.

   c) Implementa otra operación que dado un libro del que se reciben k ejemplares, vea si hay peticiones y si las hay, escriba por pantalla los k primeros clientes y elimine sus peticiones. Si hay más de k peticiones, el *stock* seguirá a 0. Si hay menos de k peticiones, habrá que actualizar el *stock* con los libros que queden.

5. Una agencia de viajes ofrece `N` destinos; para cada destino se puede optar por 5 clases de viaje: super, luxe, normal, turista y estudiante, y además se ofrecen 3 tipos de alojamiento: AD (alojamiento y desayuno), MP (media pensión) y PC (pensión completa). Cada programa de viaje se caracteriza por la información (destino, clase, alojamiento). Por cada programa de viaje se quiere saber el número de plazas disponibles, de manera que cuando un cliente contrata un determinado programa, el número de plazas disponibles se decrementa. Cuando un programa no dispone de plazas, la información del cliente (nombre, dirección y NIF) se almacena en orden cronológico. Así, cuando se disponga de nuevas plazas en ese programa, se atenderán las peticiones en orden.

   a) Definir las estructuras de datos más adecuadas para almacenar toda la información descrita.

   b) Implementar un algoritmo que, dado un cliente y un determinado programa de viajes, compruebe si es posible o no que el cliente lo contrate. En cualquier caso, habrá que realizar las acciones oportunas para que las estructuras se actualicen de forma adecuada.

   c) Implementar un algoritmo que indique el número total de plazas disponibles por cada destino. La solución debe incluir la definición de la estructura de datos más adecuada para devolver la información. El algoritmo debe devolver además el destino con más plazas disponibles.

6. En un supermercado hay 20 cajas registradoras, en cada una de las cuales se colocan los clientes con sus carros de la compra en orden de llegada. Por cada caja registradora queremos guardar el nombre de la cajera, la recaudación acumulada y los carros en espera.
   Por otro lado, en cada carro se amontonan los distintos productos, de modo que tan sólo puede extraerse el situado en la parte superior. Por cada producto guardamos su nombre y precio.

   a) Definir las estructuras de datos más adecuadas para guardar la siguiente información, utilizando en cada apartado las de los anteriores:

      1) Un carro de la compra.
      2) Una caja registradora.
      3) Todas las cajas del supermercado.

   b) Implementar un algoritmo `atender_cliente` que dado un carro de la compra, pase los productos que contiene por caja y calcule el precio total.

   c) Implementar un algoritmo que calcule la recaudación de todas las cajas después de pasar los primeros `x` carros por cada una de ellas. El valor `x` será un argumento de entrada que puede ser mayor que el número de carros en espera en algunas de las cajas. Actualizar la recaudación acumulada en cada una de las cajas de modo adecuado.

7. Con el fin de mejorar el servicio de autobuses universitario, cierto ayuntamiento y cierta universidad firman un convenio para simular dicho servicio. En la simulación se trabaja con dos estructuras básicas: la estructura `Tparada`, que representa la personas que esperan un autobús, y la estructura, `Tautobus`, que representa las personas que están dentro del autobús. Supongamos ya definido el tipo básico `Tpersona` que guarda la información de cada pasajero.
   En la simulación hay dos premisas que cumplir: asumiendo un comportamiento civilizado por ambas partes, nadie se cuela en la parada y nadie viaja de pie en un autobús (sólo se ocupan los asientos). Se supone que en el autobús hay F filas y C columnas de asientos. Cuando el autobús llegue a la parada, los asientos estarán ocupados o libres de forma aleatoria. Se pide:

   a) La definición de las estructuras `Tparada` y `Tautobus`.

   b) La definición de la operación `coger_autobus`, cuyo objetivo es recoger personas de la parada y acomodarlas en el autobús.

8. Se pretende simular el comportamiento de la CPU de un ordenador ante los distintos requerimientos de distintos tipos de procesos. La idea básica es la siguiente:
   Existen P tipos de proceso distintos (por ejemplo: de usuario, de Entrada/Salida, del sistema operativo, etc.). Cada tipo de procesos tiene una prioridad. La CPU atenderá primero al primer proceso de entre el grupo con mayor prioridad en un momento dado. La asignación de prioridades para un grupo de procesos se otorga según el algoritmo LRU (Last Recently Used): el que lleve más tiempo sin ser atendido, es el grupo de procesos con mayor prioridad. Así pues, la CPU indaga qué grupo de procesos tiene la prioridad más alta; una vez encontrado, incrementa en una unidad la prioridad de los demás grupos y pone a cero la prioridad del grupo elegido; coge el primer proceso del grupo y lo atiende. Al finalizar con este proceso, pasa a atender el siguiente aplicando el mismo criterio.
   Por cada proceso individual se guarda un identificativo (0 a 100) y un valor entero indicando el tiempo que le queda para finalizar.
   Atender un proceso supone restar uno al tiempo que le queda; si el tiempo es cero, eliminarlo del grupo y si es mayor, colocarlo como último de su grupo.

   a) Definir las estructuras de datos más adecuadas para guardar los procesos individuales y cada uno de los grupos con su prioridad asociada.

   b) Implementar la operación `Escoger_proceso` que devuelve el proceso del grupo con mayor prioridad y modifica las prioridades.

   c) Implementar la operación `Atender_proceso` que, dado un proceso, realiza los pasos descritos anteriormente.

   d) Implementar un algoritmo `cpu` al que se le dan como entrada los distintos grupos de procesos inicializados e itera según la descripción anterior hasta que no quedan procesos. Utilizar las operaciones y tipos definidos en los otros apartados.

9. Supongamos que queremos gestionar los trabajos de impresión en una impresora. Los trabajos pueden tener cuatro prioridades distintas (de 0 a 3, siendo 0 la mayor prioridad), en función de las cuales se atienden. Además, los trabajos con igual prioridad se atienden en orden de llegada. Por cada trabajo guardamos el nombre del usuario que lo imprime, el tamaño del documento y la prioridad.
   Para gestionar los trabajos vamos a implementar una cola con prioridad. La implementación de la cola se realizará con un vector de cuatro colas normales, cada una de las cuales guardará los trabajos de una de las prioridades. Implementar las distintas operaciones de una cola con prioridad utilizando el tipo de datos anterior.

10. Dada la especificación de la clase *deque* vista en teoría, realizar una implementación enlazada de la misma.

11. Se quiere modelar un servicio medico de urgencias. Cuando un paciente llega a recepción, se le asigna una prioridad de acuerdo con los síntomas que presenta. Para cada paciente se guarda su nombre, número de la seguridad social, prioridad y hora de llegada (de 0 a 24). El médico de urgencias atiende a los pacientes por prioridad (a igual prioridad se atienden por orden de llegada).

    a) Define la estructura de datos más adecuada para guardar cada paciente, teniendo en cuenta que se van a almacenar posteriormente en una cola de prioridad ordenada en base al campo prioridad. Denomina a la estructura `Tpaciente`.

    b) Define un tipo `ColaPacientes` que sea una cola de prioridad de pacientes.

    c) Implementar una operación `atiende_paciente`. Esta operación recibirá como entradas la cola de pacientes y la hora actual (0 a 24). La operación atenderá al paciente eliminándolo de la cola y devolviendo el tiempo que llevaba esperando en la misma.

    d) Implementar una operación `simula_urgencias`. Esta operación debe atender a todos los pacientes de la cola y devolver el tiempo medio de espera. Para determinar la hora en la que es atendido cada paciente se utiliza una función auxiliar `int Hora()` que devuelve la hora que es al ser llamada.