Estructuras de Datos

Grado en Ingeniería en Informática

Examen de Junio de 2012

Nombre: Grupo:

Pregunta 1:

Contestar V/F las siguientes cuestiones, teniendo en cuenta que una respuesta incorrecta invalida una correcta.

[ ] Una matriz definida como (int **a) se almacena en una zona contigua de memoria

[ ] No hay ningún método que permita accesos eficientes por clave en un contenedor lineal

[ ] La operación de inserción de un nodo al final de una lista simplemente enlazada implementada con cola y cabecera necesita un tiempo O(1)

[ ] Esta sentencia es correcta usando STL y produce los resultados esperados: vector<int> v; v.insert(v.begin() + 5, 100);

[ ] La altura del árbol es la altura del nodo raíz

[ ] Un heap es un árbol binario equilibrado en altura

[ ] Un grid no es una estructura espacial adecuada si la mayoría de los puntos se concentran en una parte concreta de la escena

[ ] Un dato almacenado en una lista dinámica que no sufre ningún tipo de modificación puede que cambie su posición de memoria

[ ] La inserción de un dato en una matriz dispersa puede implicar añadir dos nodos a la estructura de datos

[ ] La utilidad de las cubetas en dispersión es minimizar el número de elementos reasignados a otras posiciones

Pregunta 2:

Indicar de que manera evolucionaría un árbol binario de búsqueda de enteros durante la inserción/borrado de los siguientes datos: I(6), I(2), I(4), I(10), I(1), I(15), I(14), I(3), I(5), I(8), B(15), B(6). Mostrar el estado del árbol cada 3 operaciones.

Pregunta 3:

Dada la siguiente definición de una tabla de dispersión cerrada de enteros con función de dispersión por división y resolución de colisiones mediante exploración lineal, implementar la operación *existe()*, que devuelve true si el dato existe en la tabla.

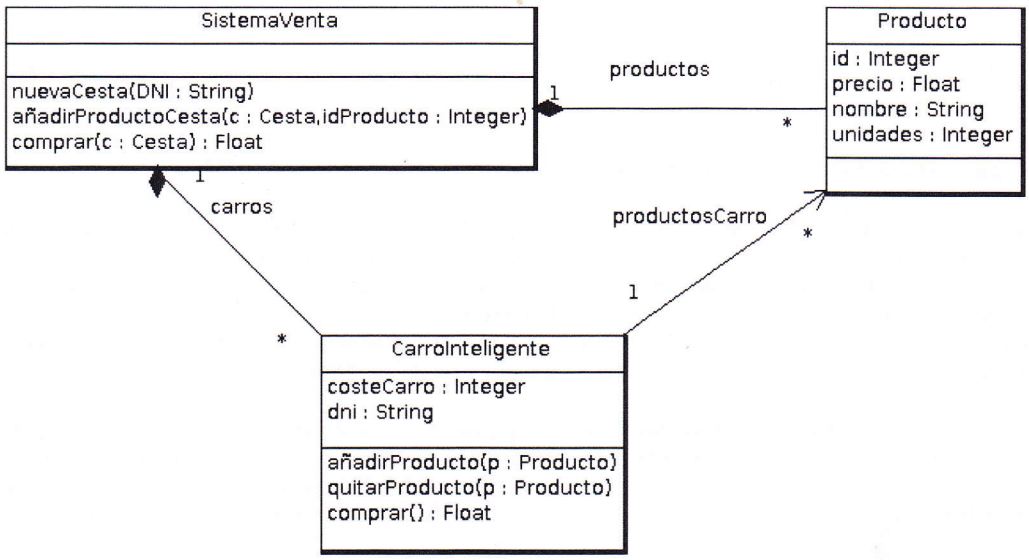
```
#define TAM_TABLA 1003
struct PosicionEnt {
  int dato;
  char estado; // Puede ser "D" (disponible), "B" (borrado), "S" (borrado especial)
};
class DispCerradaEnt {
  PosicionEnt tabla[TAM_TABLA];

  int hash(int dato) { return pos % TAM_TABLA; }
public:
  bool existe(int dato);
  ...
};
```

Pregunta 4:

Una empresa ya está diseñando el carro de supermercado del futuro. Este carro inteligente permite hacer la cuenta automáticamente a medida que se echa un producto en el carro mediante lectura automática del código de barras. Tiene un visor que permite al cliente visualizar en todo momento el coste de su carro, y paga sólo con pasar por el punto de salida, sin necesidad de cajeros con dependientes. A su vez, otra serie de sensores detectan cuando un producto sale de un estante para echarse en el carro, lo que permite controlar posibles hurtos, desabastecimiento de algún producto o controlar en tiempo real cómo de efectiva es una oferta.

Implementad parte de la funcionalidad de este sistema según el diseño de clases que aparece abajo.

Producto: se identifica de forma única mediante un identificador numérico (id), tiene un nombre (nombre), un precio (precio) y un número de unidades en las estantería (unidades).



CarroInteligente: cada carro que está funcionando en el super está asociado a un cliente con dni (DNI), y tiene un visor con el importe del carro en todo momento. Cuando un cliente echa un nuevo producto en el carro, un sistema de sensores del carro junto con el de los estantes se encargan conjuntamente de la operación *aniadirProducto()*. Entonces se descuenta el número de unidades de ese producto en los estantes y en el visor del carro aparece el precio, el nombre del producto y el subtotal de la compra hasta el momento (usar cout<< para visualizar). Si el cliente saca el artículo del carro para no comprarlo, se sigue el proceso contrario mediante la función *quitarProducto()*. Otra parte del sistema se encargaría de detectar el lugar donde se deposita el artículo para hacerle el seguimiento. Si de un producto se compran varias unidades se añadirían varias entradas del mismo producto en la relación productosCarro. *Coprar()* lo único que hace es devolver el total de la compra, otra parte del sistema se encargía del cobro.

SistemaVenta: Controla todas las ventas del super, cuando llega un nuevo cliente y coge un carro, éste debe pasarle su DNI electrónico con la operación *nuevoCarro()*. La función *aniadirProductoCesto()* internamente debe localizar el producto utilizando la relación productos y llamar a *CarroInteligente::añadirProducto()* que se encarga de decrementar el número de unidades como se ha dicho anteriormente. Finalmente la función *compra()* se llama cuando el usuario sale por la puerta, liberando el carro. Otra parte del sistema se encarga del cobrar el importe automáticamente en la cuenta del cliente.