Pontificia Universidad JAVERIANA Bogotá

Departamento de Ingeniería de Sistemas
John Corredor, PhD
Estructura de Datos 2023 - 30
Parcial 01

Nombres	David Estevan		
Apellidos	Rodriguez Jurado		
ID:		Fecha:	20/09/2024
Tema	Primer Parcial		

Reglas y recomendaciones

1. El parcial se debe desarrollar en el computador, escribiendo las respuestas en un archivo de texto plano (TXT). Usted sólo debe entregar el archivo TXT con sus respuestas.
2. Guarde regularmente su trabajo (para evitar posibles pérdidas de información), con el nombre de archivo par1_apellido1_apellido2_nombre1_nombre2.txt.
3. El parcial tiene una duración de dos horas, contadas a partir del inicio normal de la clase. La actividad en BS está programada para cerrarse al terminar las dos horas, por lo que cualquier archivo no entregado dentro de ese lapso se considerará como parcial no presentado y tendrá una calificación de 0.0.
4. El parcial es estrictamente individual y se debe desarrollar únicamente en la sala de computadores del día.
5. Se recomienda no utilizar compiladores, intérpretes de lenguajes de programación o entornos de desarrollo de cualquier tipo.
6. Puede utilizar una hoja de apuntes con la información que considere para el parcial. Sin embargo, está absolutamente prohibido comunicarse con cualquier otro ser humano para obtener información sobre el parcial, a través de cualquier medio.
7. Los celulares deben permanecer apagados, y no se debe enviar ni recibir ningún mensaje de texto. La única excepción a lo anterior son los profesores y monitores de la asignatura, quienes sólo responderán consultas respecto a la claridad de las preguntas del parcial y no responderán consultas sobre la materia.
8. Si el estudiante incumple con cualquiera de las reglas, será evaluado con nota 0.0
9. Recuerde que diseñar no implica implementar.

Pontificia Universidad JAVERIANA Bogotá

Departamento de Ingeniería de Sistemas
John Corredor, PhD
Estructura de Datos 2023 - 30
Parcial 01

1.- (15%) Análisis de código

Considere la siguiente función (se garantiza que el argumento cont contiene la cantidad de elementos necesarios):

```
struct C{
     float x[2];
}

C funcion(std::vector< C >& v){
     C r;
     r.x[0] = r.x[1] = 0;
     std::vector< C >::iterator iT;

     for(iT = v.begin(); iT != v.end(); iT++){
          for(int j = 0; j < 2; j++){
               r.x[j] += iT->x[j];
          }
     }

     for(int j = 0; j < 2; j++){
          r.x[j] /= v.size();
     }

     return r;
}
```

1.1.- ¿Qué hace «funcion»? (descríbala brevemente)

la función calcula el promedio de los dos componentes x[0] x[1] de todos los objetos C contenidos en el vector v y devuelve un objeto C con estos promedios.

1.2 ¿Cuál es el orden de complejidad de «funcion»? Justifique brevemente

la complejidad total de la función está dominada por el primer bucle, lo que da como resultado una complejidad temporal de O(n), donde n es el número de elementos en el vector v.

2.- (20%) Selección múltiple con única respuesta

2.1.- Juanito toma la secuencia de números enteros desde 1 hasta n e inserta la primera mitad en una cola y la segunda mitad en una pila. Luego llena una lista extrayendo primero los elementos de la pila, y luego los elementos de la primera cola. ¿Cómo quedan los elementos en la lista?

a. Los primeros n/2 en orden, luego los segundos n/2 en orden inverso.
b. Los segundos n/2 en orden inverso, luego los primeros n/2 en orden.
c. Los segundos n/2 en orden, luego los primeros n/2 en orden inverso.
d. Los primeros n/2 en orden inverso, luego los segundos n/2 en orden.
e. Los segundos n/2 en orden inverso, luego los primeros n/2 en orden inverso.

2.2.- Para realizar un cálculo se cuenta con un algoritmo dividido en dos bloques secuenciales de instrucciones. El primer bloque de instrucciones tiene complejidad $O(n)$, mientras que el segundo bloque de instrucciones tiene complejidad $O(\log_{10} n)$. ¿Cuál es la complejidad del algoritmo completo?

a. $O(n)$
b. $O(n \log_{10} n)$
c. $O(n^2)$
d. $O(\log_{10} n)$
e. $O(n + \log_{10} n)$

3.- (65%) Diseño e Implementación de TADs

El mantenimiento y expansión de la infraestructura vial inter-municipal del país se lleva a cabo a través de un sistema de concesiones, donde tramos específicos son entregados a empresas privadas quienes se encargan de mantenerlos en adecuadas condiciones para el transito de vehículos. Las concesiones suelen financiarse en parte a través del cobro del peaje, una tarifa específica que deben cancelar los vehículos que transitan por determinado tramo de la concesión. Aunque los peajes están regulados por el Ministerio de Transporte, no existe una tarifa fija, sino que ésta es determinada por la concesión en cada tramo dependiendo de sus características y servicios ofrecidos al usuario.

Un punto de cobro de peaje en un tramo de vía está conformado por un conjunto de varias casetas (ubicadas en línea), donde en cada caseta se ubica un cajero que realiza el cobro a cada vehículo individualmente. Cada caseta soporta una cola de vehículos, los cuales van llegando a esperar que se les realice el cobro para poder continuar su camino. Los vehículos suelen identificarse por una placa, combinación única de letras y números que los identifica individualmente. Para el cobro del peaje, los vehículos se clasifican en varios tipos, dependiendo principalmente de su tamaño (o número de ejes); por lo general, a mayor tamaño, mayor es el valor del peaje que debe cancelar. El punto de cobro cuenta con una tabla en la que, para cada tipo de vehículo, tiene especificado el valor del peaje que debe cobrarse.

En cuanto al cobro del peaje, cada caseta suele llevar una cuenta individual de los valores cobrados a lo largo del día. En algún momento, se suele hacer un arqueo de caja, en donde el cajero de la caseta entrega lo recogido hasta el momento y se re-inicia en ceros la cuenta de la caseta. La cuenta individual de la caseta tiene, además del total recogido hasta el momento, los totales parciales de lo recogido de acuerdo a los tipos de vehículos que han pasado por el punto de cobro.

Para un manejo eficiente de los movimientos de vehículos y cobros de peaje en un punto de cobro específico, se está considerando desarrollar un sistema de apoyo para el cobro de peajes. Se ha identificado que este sistema debe proveer, entre otras, las siguientes operaciones:

a. Modelar la llegada de un vehículo al punto de cobro. Se ha identificado que los vehículos que llegan suelen ubicarse en la cola de la caseta que tiene menos vehículos esperando, para así poder continuar su camino lo más rápido posible.
b. Modelar el cobro del peaje para un vehículo en una caseta dada. De acuerdo al tipo del vehículo que está adelante en la cola, se almacena su información de identificación y el valor cobrado como peaje, se actualiza la cuenta individual de la caseta especificada, y se retira el vehículo de la cola, para dejar disponible el siguiente.

c. Realizar el arqueo de caja de las casetas del punto de cobro. Se recopila la información de las cuentas individuales de cada caseta para averiguar el monto total recibido hasta el momento por concepto de peajes, discriminado también de acuerdo a los tipos de vehículo especificados en la tabla del punto de cobro.

Se requiere entonces diseñar e implementar (en C++) los componentes ya descritos del sistema de apoyo al cobro de peajes.

3.1 – (15%) Diseño

Diseñe el sistema y el (los) TAD(s) solicitado(s). Recuerde que diseñar es un proceso previo a la implementación, por lo que no debería contener ninguna referencia a lenguajes de programación (código fuente). Para simplicidad del diseño, no es necesario incluir los métodos obtener y fijar (get / set) del estado de cada TAD. Para el diagrama de relación, anexar en formato PDF como parte de su entrega.

TAD Vehículo

Datos mínimos:

- **placa:** Identificación única del vehículo.
- **tipo**: Tipo del vehículo, que determina el valor del peaje.

TAD Caseta

Datos mínimos:

- **colaVehiculos**: Cola de vehículos esperando para el cobro.
- **totalCobrado**: Monto total acumulado en la caseta.
- **totalPorTipo**: Monto acumulado por cada tipo de vehículo.

Operaciones:

- **agregarVehiculo(Vehiculo v):** Agrega un vehículo a la cola.
- **cobrarPeaje():** Cobra el peaje al vehículo que está al frente de la cola.
- **arqueo():** Devuelve el total cobrado y lo discrimina por tipo de vehículo.

Pontificia Universidad JAVERIANA Bogotá

Departamento de Ingeniería de Sistemas
John Corredor, PhD
Estructura de Datos 2023 - 30
Parcial 01

TAD PuntoDeCobro

Datos minimos:

- **casetas:** Conjunto de casetas en el punto de cobro.
- **tarifas:** Tabla con los valores del peaje para cada tipo de vehículo.

Operaciones:

- **llegadaVehiculo(Vehiculo v):** Asigna un vehículo a la caseta con menos vehículos en la cola.
- **cobrarEnCaseta(int numCaseta):** Realiza el cobro del peaje en la caseta indicada.
- **realizarArqueo():** Devuelve el total recaudado en todas las casetas y lo discrimina por tipo de vehículo.

3.2.- Algoritmos

De acuerdo al diseño realizado en el punto anterior, escriba la implementación de los siguientes algoritmos en C++. Estas implementaciones deberán tener en cuenta:

- La definición apropiada de los prototipos de los métodos/funciones (i.e. recibir/retornar los datos suficientes y necesarios para su correcta ejecución),
- El NO uso de salidas/entradas por pantalla/teclado (i.e. paso/retorno correcto de valores y/o objetos),
- El correcto uso del (de los) TAD(s) diseñado(s) en el punto anterior, y la escritura de todo el código que pueda llegar a necesitar que no esté incluido en la STL (excepto los métodos get / set del estado de cada TAD).

3.2.1.- (12%) Modelar la llegada de un vehículo.

```
void llegadaVehiculo(Vehiculo v)
{
    // Encontrar la caseta con menos vehículos en la cola
    int casetaConMenosVehiculos = 0;
    int minVehiculos = casetas[0]. colaVehiculos.size();

    for (int i = 1; i < casetas.size(); i++)
{
        if (casetas[i]. colaVehiculos.size() < minVehiculos)
    {
            minVehiculos = casetas[i].colaVehiculos.size();
            casetaConMenosVehiculos = i;
        }
    }

    // Agregar el vehículo a la caseta seleccionada
    casetas[casetaConMenosVehiculos].agregarVehiculo(v);
}
```

3.2.2.- (18%) Modelar el cobro del peaje en una caseta dada.

```
Vehiculo obtenerSiguienteVehiculo()
{
    if (!colaVehiculos.empty())
    {
        Vehiculo v = colaVehiculos.front();
        colaVehiculos.pop();
        return v;
    }
    else
    {
        throw std::runtime_error("No hay vehículos en la cola");
    }
}

void cobrarPeaje(const std::map<int, float> &tarifas)
{
    if (colaVehiculos.empty())
    {
        throw std::runtime_error("No hay vehículos en la caseta");
    }

    Vehiculo vehiculo = obtenerSiguienteVehiculo();
    float peaje = tarifas.at(vehiculo.tipo); // Obtener el valor del peaje para el tipo del vehiculo

    // Actualizar el total cobrado
    totalCobrado += peaje;
    totalPorTipo[vehiculo.tipo] += peaje;
}
```

3.2.3.- (20%) Realizar el arqueo de caja del punto de cobro.

```
float obtenerTotalCobrado()
{
    return totalCobrado;
}

std::map<int, float> &obtenerTotalesPorTipo()
{
    return totalPorTipo;
}

void realizarArqueo(std::map<int, float> &totalesPorTipo, float &totalGeneral)
{
    totalGeneral = 0;
    totalesPorTipo.clear();

    for (std::deque<Caseta>::iterator it = casetas.begin(); it != casetas.end(); ++it)
    {
        totalGeneral += it->obtenerTotalCobrado();
        auto &totalesCaseta = it->obtenerTotalesPorTipo();

        for (auto itTipo = totalesCaseta.begin(); itTipo != totalesCaseta.end(); ++itTipo)
        {
            totalesPorTipo[itTipo->first] += itTipo->second;
        }
    }
}
```