Tarea 3

Algoritmos y Complejidad

Containers Contentos

2017-04-04

1. Problema

Un puerto almacena *containers* para ser cargados en los barcos que zarpan del mismo. Estos *containers*, traídos por camiones, deben apilarse en el puerto conforme van llegando.

Los *containers* llegan en la mañana en un orden específico y una vez se posicionan no se pueden mover hasta que se cargan en su barco correspondiente, lo que sucede en la tarde.

Para identificar a qué barco le pertenece cada *container* estos últimos tienen pintado el número de serie del barco al que pertenecen, los barcos zarpan con sus *containers* desde menor a mayor número de serie.

Para poder cargar un *container* en un barco se deben mover todos los que estén encima del mismo, pero esta operación es muy costosa, así que se opta por apilar las cajas en varias pilas estratégicamente para que, cuando un barco llegue, no haya ningún *container* de otro barco sobre los suyos.

El problema consiste en encontrar una forma de apilar los contenedores recibiéndolos en el orden en que llegan (sin poder reubicarlos luego de apilarlos) minimizando el número de pilas necesarias.

En resumen, aunque las pilas pueden ser tan grandes como se quiera, un contenedor que llega antes no puede apilarse sobre uno que llega después, tampoco se puede apilar un contenedor de un barco que llega después sobre otro de un barco que llega antes.

1. Diseñe un algoritmo voráz para resolver este problema de manera óptima.
   Defina bien cuál es el **problema**, la **decisión que se debe tomar** y **como se contruye la solución** del problema más grande, en cada paso.
   Imporante: su definición del problema debe servir para todos los pasos, no sólo para el primero.

   (25 puntos)

2. Demuestre la optimalidad de este algoritmo.

   (35 puntos)

3. Haga un programa en Python, C, C++ o Haskell que reciba una línea con los índices de las cajas en el orden que llegan e imprima las pilas resultantes.
(40 puntos)

   Siga el formato mostrado en los siguientes ejemplos de ejecución:
   Ejemplo 1:

   ```
   INSERTE PILAS:
   10 20 35 5 10 10
   PILAS:
   1> 35 10 10
   2> 20 5
   3> 10
   ```

   Ejemplo 2:

   ```
   INSERTE PILAS:
   1 3 15 10 3 20 3
   PILAS:
   1> 20 3
   2> 15 10 3
   3> 3
   4> 1
   ```

Se evaluará:

- **Ejecución correcta**: que funcionen los casos de prueba y sea posible encontrar casos en que el programa entregue una respuesta equivocada.
- **Complejidad computacional adecuada**: Que el algoritmo implementado tenga una complejidad igual o mejor que la esperada, y que sea ad-hoc a la materia que se está evaluando (e.g. no utilizar programación dinámica si se pide programar un algoritmo voraz).
- **Calidad del programa**: uso adecuado de funciones, uso de estructuras de control, uso de estructuras de datos, código claro y simple.
- **Código ordenado**: nombres adecuados, indentación correcta, comentarios suficientes, ausencia de código comentado.

2. Condiciones de entrega

- La tarea se realizará *individualmente* (esto es grupos de una persona), sin excepciones.
- La entrega debe realizarse vía Moodle en un *tarball* en el área designada al efecto, bajo el formato `tarea-3-`*`rol`*`.tar.gz` (`rol` con dígito verificador y sin guión).
  Dicho *tarball* debe contener dos directorios:

- Un directorio `tarea`, que contenga las fuentes en LaTeX (al menos `tarea.tex`) de la parte escrita de su entrega, además de un archivo `tarea-3.pdf`, correspondiente a la compilación de esas fuentes.
- Un directorio `programa`, que contenga su código y un `README` indicando las instrucciones de compilación y ejecución y, opcionalmente, información adicional que deba saber el evaluador.

- Además de esto, la parte escrita de la tarea debe en hojas de tamaño carta en Secretaría Docente de Informática (Piso 1, edificio F3).

- Tanto el *tarball* como la entrega física deben realizarse el día indicado en Moodle.

  Por cada día de atraso se descontarán 20 puntos y a partir del tercer día de atraso no se reciben más tareas y la nota es automáticamente cero.