Algoritmos y Estructuras de Datos III

Trabajo Práctico N°1

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Integrante	LU	Correo electrónico
Izcovich, Sabrina	550/11	sizcovich@gmail.com
Garcia Marset, Matias	356/11	matiasgarciamarset@gmail.com
Orellana, Ignacio	229/11	nacho@foxdev.com.ar
Vita, Sebastián	149/11	sebastian_vita@yahoo.com.ar

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción

Este trabajo práctico consiste en la resolución de ciertos problemas algorítmicos delimitados por algunas restricciones impuestas por la cátedra, como por ejemplo el orden de complejidad máximo de los mismos entre otras. Para justificar las implementaciones de los problemas en cuestión, fue necesaria la utilización de herramientas lógico-matemáticas que serán mencionadas a lo largo del desarrollo de cada ejercicio.
Para comprobar que nuestras soluciones resolvieran correctamente los problemas propuestos, debimos dividir el análisis de los mismos en secciones a fin de estudiar minuciosamente cada característica de éstos. Estas secciones se dividen de la siguiente forma:

- **Problema a resolver:** En esta sección, nos encargamos de describir detalladamente el problema a resolver dando ejemplos del mismo y sus soluciones.
- **Resolución coloquial:** En esta parte, nos dedicamos a explicar de forma clara, sencilla, estructurada y concisa las ideas desarrolladas para la resolución del problema en cuestión. Para ello, decidimos utilizar pseudocódigo y lenguaje coloquial combinando ambas herramientas de manera adecuada.
- **Demostración de correctitud:** Utilizamos este apartado para justificar que el punto anterior resuelve efectivamente el problema y demostramos formalmente la correctitud del mismo.
- **Complejidad del algoritmo:** En esta sección, nos ocupamos de deducir una cota de complejidad temporal del algoritmo propuesto en función de los parámetros considerados como correctos. Por otro lado, justificamos por qué el algoritmo desarrollado para la resolución del problema cumple con la cota dada.
- **Código fuente:** En esta parte, presentamos las funciones relevantes del código fuente que implementa la solución propuesta. Para ello, decidimos utilizar el lenguaje `C++` dado que éste cuenta con la librería *stl* que proporciona las estructuras necesarias para la realización de dicha tarea.
- **Instancias posibles:** Este sector presenta un conjunto de instancias que permiten verificar la correctitud del programa implementado cubriendo todos los casos posibles y justificando la elección de los mismos. Dichas instancias fueron evaluadas por el algoritmo realizado y los resultados obtenidos fueron comprobados.
- **Testing:** Por último, los tests consistieron en experimentaciones computacionales utilizadas para medir la performance del programa implementado. Para ello, debimos preparar un conjunto de casos de test que permitieran observar los tiempos de ejecución en función de los parámetros de entrada que fueran relevantes. De este modo, nos encargamos de generar instancias aleatorias como también con instancias particulares. Para que los resultados fueran visibles y claros, utilizamos una comparación gráfica entre los tiempos medidos y la complejidad teórica calculada.

2. Ejercicio 1

2.1. Problema a resolver

El siguiente ejercicio consiste en hallar una manera de implementar un sistema proporcionado respetando una cota determinada de orden de complejidad. El problema se sitúa en un centro de distribución de correo que recibe paquetes todos los días cuyo destino final es la sede central de la empresa. Para el transporte de los mismos, éstos son cargados a camiones de igual capacidad. El encargado de logística, Pascual, tiene un sistema que utiliza desde hace años para agilizar la carga de los camiones asegurando el uso de una baja cantidad de los mismos para el envío de paquetes al final del día. Dicho sistema consiste en agarrar los paquetes y ubicarlos en algún camión que ya tenga paquetes dentro, eligiendo entre éstos el que menos peso esté cargando hasta ese momento. Si el peso del paquete permite que éste sea cargado en ese camión, se lo ubica allí, sino, se lo incluye en un nuevo camión.
El problema a resolver se basa en escribir un algoritmo que tome los pesos de los paquetes que hay que acomodar e indique cuántos camiones se van a utilizar y cuánto peso se cargará en cada uno de ellos al final del día considerando el sistema de Pascual. Para esto, se respeta el orden de llegada de los paquetes a medida que ingresan.
Las consideraciones a tener en cuenta son que los camiones tienen la misma capacidad de carga, la cantidad disponible de los mismos alcanza para transportar todos los paquetes y que el peso de un paquete no supera la capacidad de carga de un camión. Del mismo modo, el tamaño de los paquetes no es tenido en cuenta. Además, es importante tener en cuenta que las cargas son valores enteros positivos.

Formatos de entrada y salida:

La entrada contiene varias instancias del problema. Cada instancia consta de una línea con el siguiente formato:

$$L\ n\ p_1\ p_2\ ...\ p_n$$

donde L es el límite de carga de los camiones, n es la cantidad de paquetes a acomodar y p_1**, ...,** p_n son los pesos de cada paquete en el orden en el que deben ser almacenados.

La salida debe contener una línea por cada instancia de entrada, con el siguiente formato:

$$k\ c_1\ c_2\ ...\ c_n$$

donde k es la cantidad de camiones utilizados y c_1**, ...,** c_k es el peso que se cargó en cada uno de los k camiones al final del día.

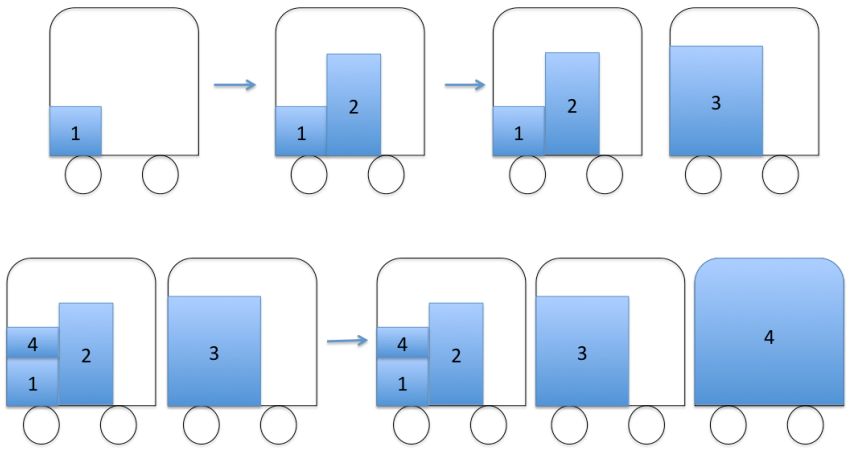
En lo que sigue, presentaremos dos ejemplos sobre el sistema impulsado por Pascual:

- **Ejemplo 1:**

  En este ejemplo, decidimos conveniente develar un caso en el que fuera agregado un paquete a un camión ya cargado. Por otro lado, quisimos contrarestarlo insertando un paquete con una carga que superaba la capacidad del camión creado anteriormente. Por último, nos pareció importante mostrar un caso en el que al agregar un nuevo paquete, si bien un nuevo camión había sido creado, éste era colocado en el camión cuya carga fuera la menor.

  Formato de entrada:

  100 5 20 40 80 15 100



Formato de salida:

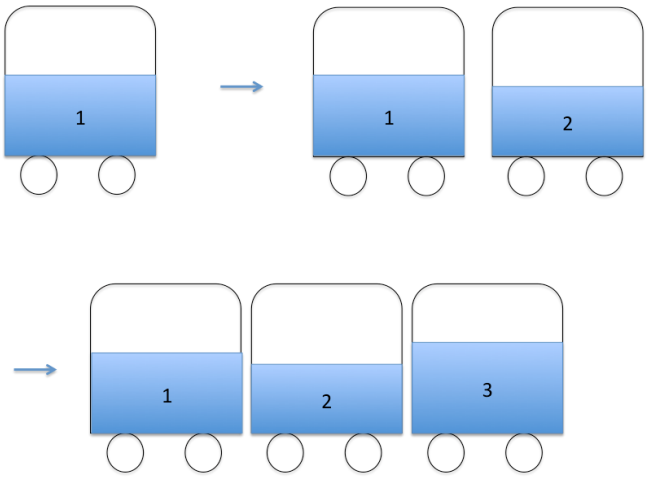
3 75 80 100

- **Ejemplo 2:**

En este ejemplo, quisimos mostrar lo que ocurría en caso en el que cada paquete superara el 50 % de la carga disponible en un camión.

Formato de entrada:

150 3 80 75 82



Formato de salida:

3 80 75 82

2.2. Resolución coloquial

Al analizar el problema a resolver, nos percatamos de que lo más conveniente era implementar la solución en base a un *algoritmo goloso*. Dicho algoritmo consiste en la construcción de una solución seleccionando, en cada paso, la mejor alternativa sin considerar las implicancias de ésta.
Por otro lado, dicha técnica de diseño algorítmico resulta fácil de implementar, suele ser eficiente y

permite construir soluciones razonables.

En este caso, la resolución basada en un algoritmo goloso nos permitió tomar cada instancia de la entrada y ubicarla en el lugar más conveniente en ese momento. Esto significa que dada la $i-ésima$ caja ingresada en un mismo día, ésta era ubicada en el camión que menor cargado se encontraba en el momento en el que se la agregaba. Para lograr esto, decidimos utilizar como estructura un heap.

Un heap consiste en una estructura de datos del tipo árbol binario completo con información perteneciente a un tipo de datos cumpliendo con un cierto orden. Un heap máximo tiene la característica de que cada nodo padre tiene un valor mayor que el de cualquiera de sus nodos hijos.
Para adaptar el problema de Pascual al heap, decidimos que cada nodo debía representar la capacidad de carga disponible correspondiente a un camión determinado en un momento dado.

El pseudocódigo ideado para resolver el problema es el siguiente:

Algorithm 1: Algoritmo de Pascual

```
Input: capacidadCamion, cantidadDePaquetes, pesosDePaquetes
Output: cantidadDeCamiones, listaDeCamiones
cantidadDeCamiones = 0
actual = 0
camionesCreados = vacio
while pesosDePaquetes ≥ vacio do
    if camionesCreados ≠ vacio ∧ camionMenosCargado(camionesCreados)[a] ≥
    pesosDePaquetes_actual then
        camionActual = camionMenosCargado(camionesCreados)
        camionActual = camionActual - pesosDePaquetes_actual
        camionesCreados ← camionActual
    else
        camionActual = capacidadCamion - pesosDePaquetes_actual
        camionesCreados ← camionActual
        cantidadDeCamiones = cantidadDeCamiones + 1
    end
    actual = actual + 1
end
```

[a] Esta funcion devuelve el camion menos cargado

Algorithm 2: Algoritmo de Pascual (version Nacho)

```
Input: Paquetes ps
Output: cantidadDeCamiones, listaDePesos
Camiones ca ← {nuevoCamion}
cantidadDeCamiones := 1
if ps = ∅ then devolver 0, ∅
for Paquete p ∈ ps do
    Camion c := camionMenosCargado(ca)
    if peso(p) ≤ capacidad(c) then
        capacidad(c) − = peso(p)
    else
        ca ← Camion cNuevo
        capacidad(cNuevo) − = peso(p) cantidadDeCamiones + 1;
    end
end
devolver cantidadDeCamiones, ca
```

2.3. Demostración de correctitud

Esta estructura resultó la más adecuada para nuestra implementación dado que

2.4. Complejidad del algoritmo

Tal como requerido, la complejidad temporal del algoritmo debe ser estrictamente menor a $\mathcal{O}(n^2)$.

2.5. Código fuente

2.6. Instancias posibles

2.7. Testing

3. Ejercicio 2

3.1. Problema a resolver

3.2. Resolución coloquial

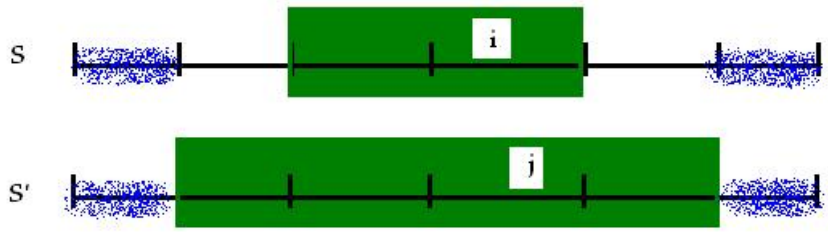
3.3. Demostración de correctitud

Primero veamos el invariante de nuesta solucion:

- Sea S una solucion, cumple que el i-esimo curso tiene la fecha de finalizacion mas proxima posible[1] al anterior, es decir, al (i-1)-esimo curso. Estando siempre el curso que primero finaliza como solucion.

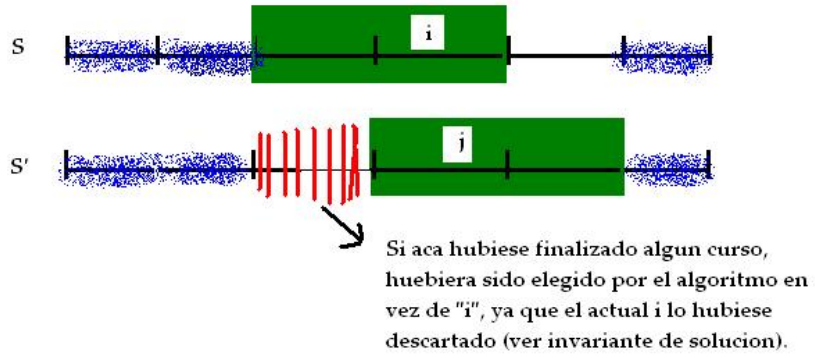
Sea C un conjunto de cursos tal que:

- Ningun curso se solapa, la solucion es C. Esto es trivial, ya que mas cursos de los que hay no pueden haber.
- Existen dos cursos que se solapan. Sea i y j dichos cursos. Sin perder generalidad supongamos que la fecha de terminacion de j es mayor o igual que la de i. Quiero ver que, si S es la solucion que contiene a i y S' es la solucion que contiene a $j \Rightarrow |S'| \leq |S|$. Esto se cumple porque S' contiene a j que termina luego que i, por lo que por lo menos todos los cursos de S' desde el j estan contenidos en S y, ademas, si j comienza antes que i se que la cantidad de cursos de S' antes de j estan antes de i en S



ò si j empieza luego de i yo se que no termina ningun otro curso entre el comienzo de i y j, porque de terminar ubiese elejido ese en S (ya que elijo siempre el que termina antes),

[1] Con "posible"nos referimos a que no se solapa con ningun curso que tenga menor fecha de finalizacion.



por lo tanto, |S'| $\leq$ |S|.

3.4. Complejidad del algoritmo

3.5. Código fuente

3.6. Instancias posibles

3.7. Testing

4. Ejercicio 3

4.1. Problema a resolver

4.2. Resolución coloquial

4.3. Demostración de correctitud

4.4. Complejidad del algoritmo

4.5. Código fuente

4.6. Instancias posibles

4.7. Testing

5. Referencias