Práctica 3: Algoritmos Greedy/Voraces
Problema de los Contenedores en un barco.

Javier Labrat Rodríguez
José Antonio Martínez López
Adrián Portillo Sánchez
Alejandro Durán Castro
José Juan Pérez González

21 de abril de 2015

Índice

1. Contenedores en un barco

Se tiene un buque mercante cuya capacidad de carga es de K toneladas y un conjunto de contenedores $c_1,...,c_n$ cuyos pesos respectivos son $p_1,...,p_n$ (expresados también en toneladas) y para los que se obtiene un beneficio al ser transportados de $b_1,...,b_n$ euros, respectivamente.
Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores:

- Diseñe un algoritmo que maximice el número de contenedores cargados, y demuestre su optimalidad.
- Diseñe un algoritmo que intente maximizar el beneficio obtenido.

1.1. Algoritmo que maximiza el número de contenedores cargados

Solución *greedy/voraz* para el problema MaxContenedoresEnUnBarco(K, C, A):

- *Conjunto de Candidatos*:
  - Conjunto de contenedores C ($c_1,...,c_n$)
- *Conjunto de Seleccionados*:
  - Conjunto solución A
- *Función Solución*:
  - El barco no tiene capacidad restante para un contenedor más.
- *Función de Factibilidad*:
  - La capacidad de carga restante del barco es superior al peso del contenedor candidato.
- *Función Selección*:
  - Determina el mejor candidato seleccionando el contendor de menor peso.
- *Función objetivo*:
  - Cargar el máximo número de contenedores posibles en el barco.

Suponemos el conjunto *C* de Contenedores ordenado por el peso de éstos de la siguiente forma:

$$p_1 \leq p_2 \leq p_3 \leq \dots \leq p_n$$

Para confirmar que nuestro algoritmo greedy alcanza las solución óptima tenemos que demostrar que:

Óptimo Local ⇒ Solución Global Optimal

Esta demostración consta de dos partes:

- Primer paso:
  Demostrar que la primera decisión tomada es correcta.
- Segundo paso:
  Demostrar que existen subestructuras optimales.

Es decir, con esto vamos a conseguir demostrar que, cuando se toma la primera decisión, el problema se reduce a encontrar una solución óptima para el sub-problema que tiene como candidatos los compatibles con la primera decisión tomada.

1.1.1. Demostrar que la primera decisión tomada es correcta:

Teorema: dado un conjunto *C* de Contenedores ordenados por su peso en orden creciente, existe una solución óptima *A* que contiene al primer elemento.

Demostraremos el teorema utilizando reducción al absurdo:

> Si ninguna solución optimal contiene al primer elemento, siempre podremos escoger una solución *B* y reemplazar el primer contenedor en ésta por el primero del total de contenedores, pues este es de menor peso que el reemplazado y la solución seguiría siendo óptima ya que el número de contenedores sigue siendo el mismo.

1.1.2. Demostrar que existen subestructuras optimales:

Teorema: Sea *A* una solución óptima al problema MaxContenedoresEnUnBarco(K, C, A) y c_1 el primer contenedor en *A*, entonces *A* - $\{c_1\}$ es solución óptima para MaxContenedoresEnUnBarco(K-c_1, C*, A) con C* subconjunto de contenedores sin c_1.

Demostraremos el teorema utilizando reducción al absurdo:

> Suponemos que *A* - $\{c_1\}$ NO es solución óptima para MaxContenedoresEnUnBarco(K-c_1, C*, A), entonces, podemos encontrar una solución *B* óptima donde $|B| > |A - \{c_1\}|$ y, por lo tanto, $B \cup c_1$ es solución para MaxContenedoresEnUnBarco(K, C, A), lo que nos lleva que $|B \cup c_1| > |A|$ que es una contradicción pues hemos dicho que *A* es solución óptima del problema MaxContenedoresEnUnBarco(K, C, A).

1.1.3. Posible implementación del algoritmo:

```
Procedimiento MaxContenedoresEnUnBarco(K, C, A)
      // K es la capacidad total de carga del barco.
      // C[1..n] es el conjunto de contenedores ordenados …
      // en orden creciente en función de su peso.
      // A[∅] contendrá el conjunto resultado con los …
      // contenedores elegidos para cargarse al barco.

      kr = K; // Capacidad restante del barco
      i = 1;
      Mientras (C[i].peso <= kr)
            A.push_back(C[i]);
            kr = kr - C[i].peso;
            i = i+1;
```

1.2. Algoritmo que intenta maximizar el beneficio obtenido:

Solución *greedy/voraz* para el problema MaxBeneficioEnUnBarco(K, C, A):

- *Conjunto de Candidatos*:
  - Conjunto de contenedores C (c_1,...,c_n)
- *Conjunto de Seleccionados*:
  - Conjunto solución A
- *Función Solución*:
  - El barco no tiene capacidad restante para un contenedor más.
- *Función de Factibilidad*:
  - La capacidad de carga restante del barco es superior al peso del contenedor candidato.
- *Función Selección*:
  - Determina el mejor candidato seleccionando el contenedor de mayor razón beneficio/peso.
- *Función objetivo*:
  - Cargar el máximo número de contenedores posible en el barco que aporten el mayor beneficio posible al transportarse.

Suponemos el conjunto *C* de Contenedores ordenado por la razón beneficio/peso de éstos de la siguiente forma:

$$b_1/p_1 \geq b_2/p_2 \geq \dots \geq b_{n-1}/p_{n-1} \geq b_n/p_n$$

1.2.1. Posible implementación del algoritmo:

```
Procedimiento MaxBeneficioEnUnBarco(K, C, A)
      // K es la capacidad total de carga del barco.
      // C[1..n] es el conjunto de contenedores ordenados …
      // en orden decreciente en función de su razón …
      // beneficio/peso.
      // A[∅] contendrá el conjunto resultado con los …
      // contenedores elegidos para cargarse al barco.

      kr = K; // Capacidad restante del barco
      i = 1;
      Mientras (C[i].peso <= kr)
            A.push_back(C[i]);
            kr = kr - C[i].peso;
            i = i+1;
```