Analisis y diseño de algoritmos. Algoritmos voraces (Greedy)

Juan Gutiérrez

September 2019

Son algoritmos que construyen una solución escogiendo de manera local la mejor opción. Son fáciles de diseñar, pero lo dificil es demostrar que el algoritmo devuelve la solución óptima en el largo plazo.

1 Intervalos disjuntos (sin pesos)

Sean $[s_1, f_1]$, $[s_2, f_2]$, ..., $[s_n, f_n]$ una secuencia de intervalos cerrados en la recta. Dos intervalos son *compatibles* si no se traslapan.

Problema Max-Intervalos-Disjuntos. Dada una secuencia de intervalos cerrados en la recta, encontrar un subconjunto de intervalos compatibles dos a dos de tamaño máximo.

Algunos posibles enfoques voraces para resolver el problema:

- Seleccionar un intervalo compatible que empieza antes (menor s_i)
  No funciona si el intervalo es muy grande:

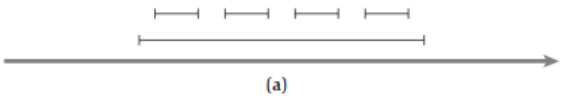

Figure 1: Tomada del libro Kleinberg, Algorithm Design

- Seleccionar un intervalo compatible con menor tamaño (menor $t_i - s_i$)
  No funciona en algunos casos:

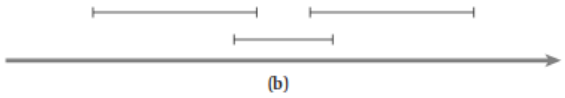

Figure 2: Tomada del libro Kleinberg, Algorithm Design

- Seleccionar un intervalo compatible con menor cantidad de intersecciones
  Es más dificil encontrar un contraejemplo, pero tampoco funciona siempre:

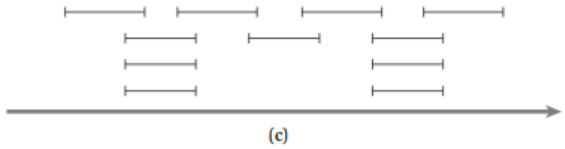

Figure 3: Tomada del libro Kleinberg, Algorithm Design

Una idea que **sí** funciona: tomar el intervalo compatible **con menor valor de su punta final**.

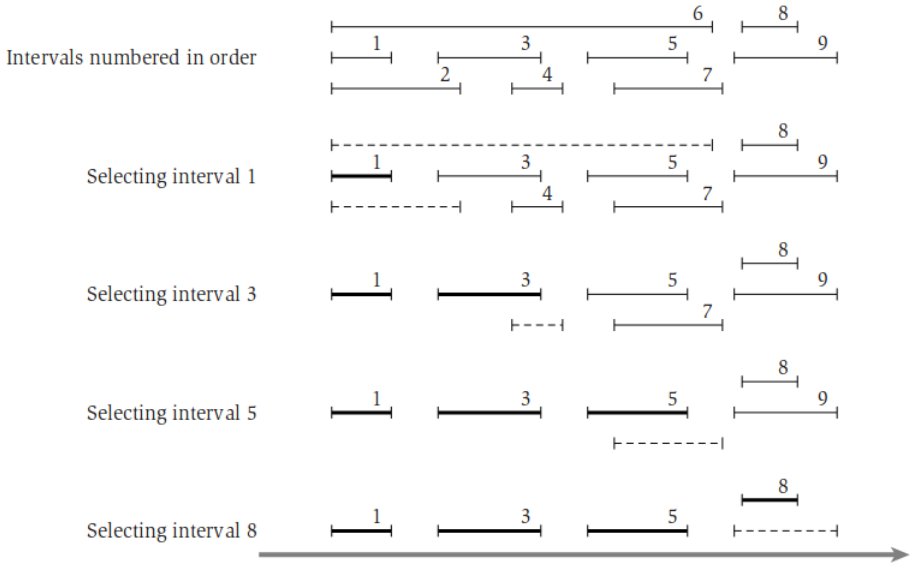

Figure 4: Tomada del libro Kleinberg, Algorithm Design

2 Una técnica general para demostraciones

Si queremos demostrar que nuestro algoritmo está correcto basta demostrar dos cosas.

1. *Elección voraz:* debemos demostrar que siempre existe una solución óptima que contiene a la eleccción voraz
2. *Subestructura óptima*: debemos demostrar que la subsolución dejada es óptima para el subproblema dejado por la elección voraz

Si se demuestran esos dos puntos, demuestro que mi voraz está correcto. Volvamos al problema de intervalos disjuntos para demostrar usando esta técnica.

2.1 Demostración para intervalos disjuntos

Recordemos el problema de intervalos disjuntos.

Sean $[s_1, f_1]$, $[s_2, f_2]$, $\cdots$, $[s_n, f_n]$ una secuencia de intervalos cerrados en la recta. Dos intervalos son *compatibles* si no se traslapan.

Problema Max-Intervalos-Disjuntos. Dada una secuencia de intervalos cerrados en la recta, encontrar un subconjunto de intervalos compatibles dos a dos.

Elección voraz: elegir el intervalo con menor f_i.

Planteamos el algoritmo recursivo:

Recibe: un conjunto $\mathcal{I} = \{[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]\}$ de intervalos, ordenados de manera creciente por punta final

Devuelve: un subconjunto de intervalos compatibles dos a dos

MAX-INTERVALOS-DISJ-REC($\mathcal{I}$)

1: **if** $\mathcal{I} = \emptyset$
2: return $\emptyset$
3: $\mathcal{I}' = \mathcal{I} \setminus \{[s_i, f_i] : s_i \leq f_1\}$
4: **return** $\{[s_1, f_1]\} \cup$ MAX-INTERVALOS-DISJ-REC($\mathcal{I}'$)

Lema 2.1 (Elección voraz)**.** *Existe una solución óptima para el problema que contiene el intervalo* $[s_1, f_1]$.

Proof. Sea X una solución óptima para el problema. Si X contiene a $[s_1, f_1]$ entonces no tenemos nada que probar.

Suponga entonces que $[s_1, f_1] \notin X$. Sea $[s_j, f_j]$ el intervalo en X con menor valor de f_j. Sea $X' = (X \setminus \{[s_j, f_j]\}) \cup \{[s_1, f_1]\}$. Mostraremos que X' es una solución para el problema. Para esto basta mostrar que $[s_1, f_1]$ es compatible con cualquier intervalo en $X \setminus \{[s_j, f_j]\}$.

Sea $[s_k, f_k]$ un intervalo cualquiera en $X \setminus \{[s_j, f_j]\}$. Note que

$$f_1 \leq f_j < s_k,$$

portanto $[s_1, f_1]$ es compatible con $[s_k, f_k]$.

Como X' es una solución para el problema y $|X| = |X'|$, concluimos que X' es una solución óptima al problema que contiene $[s_1, f_1]$. □

Lema 2.2 (Subestructura óptima)**.** *Si X es una solución óptima al problema que contiene a $[s_1, f_1]$ entonces $X \setminus \{[s_1, f_1]\}$ es una solución óptima al subproblema dejado por la elección voraz.*

Proof. Sea $\mathcal{I}$ la colección de intervalos del problema original. Sea $\mathcal{I}'$ la colección de intervalos luego de aplicar la elección voraz, es decir

$$\mathcal{I}' = \{[s_k, f_k] : f_1 < s_k\}.$$

Suponga por contradicción que $X' = X \setminus \{[s_1, t_1]\}$ no es una solución óptima para $\mathcal{I}'$. Entonces existe una solución Y' para $\mathcal{I}'$ con $|Y'| > |X'|$. Pero en ese caso $Y = Y' \cup \{[s_1, f_1]\}$ es una solución para $\mathcal{I}$ con tamaño $|Y| = |Y'| + 1 > |X'| + 1 = |X|$, contradicción. □

Con estos dos lemas a la mano, podemos probar la correctitud del algoritmo.

Teorema 2.1. *El algoritmo* Max-Intervalos-Disj-Rec *hace lo pedido*

Proof. Por inducción en $|\mathcal{I}|$. Si $|\mathcal{I}| = \emptyset$, entonces $\mathcal{I} = \emptyset$ y el algoritmo retorna $\emptyset$, lo cual es correcto. Suponga entonces que $|\mathcal{I}| > 0$ Como $|\mathcal{I}'| \leq |\mathcal{I}|$, por hipótesis de inducción, en la línea 4, el algoritmo retorna una solución óptima X' para $\mathcal{I}'$. Debemos demostrar que $X = X' \cup \{[s_1, f_1]\}$ es una solución óptima para $\mathcal{I}$.

Por el Lema 2.1 (Elección Voraz), existe una solución óptima que contiene a $[s_1, f_1]$. Sea $Y = \{[s_1, f_1\} \cup Y'$ esta solución. Por el Lema 2.2 (Subestructura óptima), se tiene que Y' es óptima para el subproblema. Luego $|Y'| = |X'|$ y por lo tanto

$$|Y| = |Y'| + 1 = |X'| + 1 = |X|.$$

Luego, como X tiene el mismo tamaño que Y, X es una solución óptima para el problema. □

Si bien es cierto, el Teorema anterior prueba correctamente la correctitud del algoritmo voraz propuesto. De acá en adelante, para los problemas que veamos, podemos asumir que basta demostrar solamente los dos lemas: elección voraz y subestructura óptima.

A continuación, veamos otro ejemplo.

Ejercicio 2.1. *Describa un algoritmo eficiente que, dado un conjunto $\{a_1, a_2, \ldots, a_n\}$ de puntos en la recta, tal que $a_1 \leq a_2 \leq \ldots a_n$ determine un conjunto mínimo de intervalos de tamaño 1 que contiene a todos los puntos. Justifique que su algoritmo es correcto usando la propiedades de elección voraz y subestructura óptima.*

Solución: dividiremos nuestra solución en 4 pasos.

1. Elección voraz: Seleccionar $[a_1, a_1 + 1]$.
2. Algoritmo recursivo.
   Recibe: Un conjunto $A = \{a_1, \ldots, a_n\}$ de puntos en la recta real tal que $a_1 \leq \cdots \leq a_n$

Devuelve: Un conjunto de intervalos unitarios de tamaño mínimo que cubre A

VORAZ-SEGMENTOS(A)
1: **if** $A = \emptyset$
2: **return** $\emptyset$
3: $A' = \{a_i \in A : a_i > a_1 + 1\}$
4: **return** $\{[a_1, a_1 + 1]\} \cup$ VORAZ-SEGMENTOS(A')

3. Prueba de la elección voraz.

   Lema 2.3 (Elección voraz)**.** *Existe una solución óptima que contiene a* $[a_1, a_1 + 1]$.

   Proof. Sea X una solución óptima. Si $[a_1, a_1 + 1] \in X$ entonces no hay más que mostrar. Suponga entonces que $[a_1, a_1 + 1] \notin X$. Como X es una solucion, existe un intervalo $[p, p + 1] \in X$ que contiene a a_1, osea $p \leq a_1 \leq p + 1$. Mostraremos que $X' = X \setminus \{[p, p + 1]\} \cup \{[a_1, a_1 + 1]\}$ es también una solución al problema.

   Para esto, debemos probar que todo $a_i \in A$ está en algún intervalo de X'. Sea $a_i \in A$. Si $a_i \notin [p, p + 1]$ entonces está en algún intervalo de $X \setminus \{[p, p + 1]\}$. Dicho intervalo también está en X' y por lo tanto a_i está cubierto por algún intervalo de X'. Si $a_i \in [p, p + 1]$ entonces

   $$a_1 \leq a_i \leq p + 1 \leq a_1 + 1,$$

   y portanto $a_i \in [a_1, a_1 + 1]$. Luego X' es una solución y como $|X| = |X'|$, entonces X' es una solución óptima. □

4. Prueba de subestructura óptima

   Lema 2.4 (Subestructura óptima)**.** *Si X es una solución óptima para A que contiene a $\{[a_1, a_1 + 1]\}$, entonces $X \setminus \{[a_1, a_1 + 1]\}$ es una solución óptima para A'.*

   Proof. Suponga por contradicción que $X' = X \setminus \{[a_1, a_1 + 1]\}$ no es óptima en A'. Entonces existe una solución Y' para A' tal que $|Y'| < |X'|$. Entonces $Y = Y' \cup \{[a_1, a_1 + 1]\}$ es una solución para A. Pero $|Y| = |Y'| + 1 < |X'| + 1 = |X|$, una contradicción. □

3 Mochila fraccionaria

Recordemos el problema de la mochila

Problema Mochila-entera. Dado un conjunto $\{1, 2, \ldots, n\}$ de items cada uno con un peso natural w_i, un valor natural v_i y un número natural W, encontrar un subconjunto de items cuya suma de valores es la mayor posible, pero menor o igual a W.

El problema fraccionario permite elegir "fracciones de items"' en cada elección.

Problema Mochila-fraccionaria. Dado un conjunto $\{1, 2, \ldots, n\}$ de items cada uno con un peso natural w_i, un valor natural v_i y un número natural W, encontrar un vector de racionales entre 0 y 1 $(x_1, x_2, \ldots, x_n)$ que maximize $\sum_{i=1}^{n} x_i v_i$ sobre la restricción $\sum_{i=1}^{n} x_i w_i \leq W$

Ejemplo: suponga que $W = 50, n = 5$, $w = [40, 30, 20, 10, 20]$, $v = [840, 600, 400, 100, 300]$. Entonces $x = [1, 1/3, 0, 0, 0]$ es una solución viable para el problema, ya que $1 \cdot 40 + 1/3 \cdot 30 + 0 \cdot 20 + 0 \cdot 10 + 0 \cdot 20 = 50 \leq 50$.

Elección voraz: escoger siempre los items con mayor ratio valor/peso. Podemos suponer que $v_1/w_1 \leq v_2/w_2 \leq \cdots \leq v_n/w_n$. Tenemos el siguiente algoritmo.

Recibe: Una instancia v, w, W del problema MOCHILA-FRACCIONARIA
Devuelve: Una solución óptima para dicha instancia
MOCHILAFRACCIONARIA-GREEDY(v, w, W)

```
1: for j = n to 1
2:    if w[j] ≤ W
3:       x_j = 1
4:       W = W − w[j]
5:    else
6:       x_j = W/w[j]
7:       W = 0
8: return x
```

Note que dicho algoritmo no resuelve el caso de mochila entera, en ese caso se debe aplicar programación dinámica.

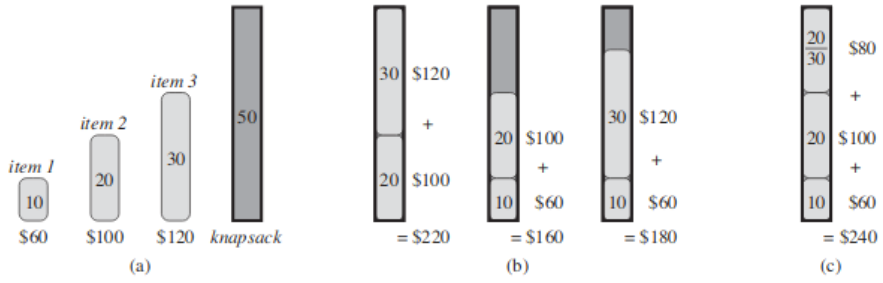

Figure 5: Tomada del libro Cormen, Introduction to Algorithms

A continuación demostraremos que nuestro algoritmo está correcto.

Lema 3.1 (Elección voraz)**.** *Existe una solución óptima* $(x_1, x_2, \ldots, x_n)$ *al problema tal que* $x_n = \min\{1, W/w_n\}$

Proof. Sea $\alpha = \min\{1, W/w_n\}$. Sea $(y_1, y_2, \ldots, y_n)$ una solución óptima al problema **tal que $\mathbf{y_n}$ es máximo**.

Si $y_n = \alpha$ entonces no hay nada que probar. Suponga entonces que $y_n \neq \alpha$.

Como y es una solución óptima, se cumple que $y \cdot w \leq W$. Portanto existe un $i \in \{1, \ldots, n-1\}$ tal que $y_i > 0$.

Sea

$$\delta = \min\{y[i], (\alpha - y[n])\frac{w[n]}{w[i]}\}$$

y

$$\beta = \delta \frac{w[i]}{w[n]}.$$

Note que $\delta, \beta > 0$.

Defina x según

$$x[j] = \begin{cases} y[j] & \text{si } j \notin \{i, n\} \\ y[i] - \delta & \text{si } j = i \\ y[n] + \beta & \text{si } j = n \end{cases}$$

Note que

$$x \cdot w = y \cdot w - \delta w[i] + \beta w[n] = y \cdot w - \delta w[i] + \delta w[i] = y \cdot w.$$

También,

$$\begin{aligned} x \cdot v &= y \cdot v - \delta v[i] + \beta v[n] \\ &= y \cdot v - \delta v[i] + \delta \frac{w[i]}{w[n]} v[n] \\ &= y \cdot v + \delta(\frac{w[i]}{w[n]} v[n] - v[i]) \\ &= y \cdot v + \delta w[i](\frac{v[n]}{w[n]} - \frac{v[i]}{w[i]}) \\ &\geq y \cdot v \end{aligned}$$

Concluimos que $xv \geq yv$, y como y es óptima, x también es óptima. Pero $x_n > y_n$, una contradicción a la elección de y.

Lema 3.2 (Subestructura óptima)**.** *Si $(x_1, x_2, \ldots, x_n)$ es una solución óptima al problema con $x_n = \min\{1, W/w_i\}$, entonces $(x_1, x_2, \ldots, x_{n-1})$ es una solución óptima al subproblema dejado con $W = W - x_n w_n$.*

Proof. Suponga por contradicción que no es el caso. Sea $(y_1, y_2, \ldots, y_{n-1})$ una solución óptima al subproblema con peso máximo $W - x_n w_n$ que escoge los $n-1$ primeros items. Entonces $(y_1, y_2, \ldots, y_{n-1}, x_n)$ es una solución viable al problema original con valor mayor que $x \cdot v$, contradicción. □

□