

Solucionando el problema de la mochila en el Modelo Sticker

Mario J. Pérez–Jiménez & Fernando Sancho–Caparrini

Ernesto Mancebo

Universidad de Sevilla

Febrero 2020

Contenido

- 1 Modelo Sticker
 - Concepto de Cadena de Memoria
 - Concepto de Tubo
 - Operaciones sobre las moléculas
 - Concepto de Librería
 - Sub-Rutina Ordenado por Cardinalidad
 - Subrutina Llenado en Paralelo
- 2 Problema de Suma de Subconjuntos
- 3 Problema de la mochila acotado
- 4 Problema de la mochila no acotado
- 5 Bibliografía

Modelo Sticker

- Modelo propuesto por Sam Roweis.
- Inspirado en Cadenas de ADN.
- Basado en operaciones de filtrado.
- Distinguido por utilizar Memoria de Acceso Aleatorio y la manera de representar la data.

Concepto de Cadena de Memoria

Las cadenas de memoria (también llamadas moléculas σ) son cadenas de forma (n, k, m) , tal que $n \geq k.m$, siendo n la longitud de la cadena, k la cantidad de sub-cadenas, y m la longitud de cada sub-cadena p .

Cada región m tiene un estado "*apagado*" / "*encendido*" o "*0*" / "*1*"

Concepto de Tubo

Un Tubo (T) consiste en un multiconjunto de complejos de memoria o de moléculas σ del mismo tipo.

Se puede ilustrar de la manera:

$$\begin{bmatrix} \sigma_0 \\ \sigma_i \\ \sigma_{i+1} \end{bmatrix}$$

Operaciones sobre las moléculas

- $mezclar(T_1, T_2)$
- $separar(T, i)$
- $encender(T, i)$
- $apagar(T, i)$
- $leer(T)$

Concepto de Librería

- Complejo de memoria de forma (k, l) tal que $(1 \leq k \leq l)$.
- Se inicializan las primeras $k - l$ regiones
"encendidas"/"apagadas" en todas sus posibles combinaciones.

Sub-Rutina Ordenado por Cardinalidad

Partiendo de los siguientes elementos:

- $A = \{1, \dots, p\}$
- $B = \{b_1, \dots, b_s\} \subseteq A$
- $F = \{D_1, \dots, D_t\} \subseteq P(A)$

Se busca ordenar F con respecto a su cardinalidad en B , o en otras palabras, la cantidad de elementos que coincidan de manera $B \cap D_i$.

Codificando los subconjuntos F

Sea σ una molécula para el tubo T_0 , la misma se codifica de forma $T_0 = \{\{\sigma : |\sigma| = p \wedge \exists j(\chi_{D_j} = \sigma)\}\}$; tal que χ_{D_j} es la función característica en A , siendo $(\chi_{D_j}(i) = 1$ si $i \in D_j$ de lo contrario $\chi_{D_j} = 0$ si $i \in A - D_j)$

Resultado esperado

- Una vez concluido el paso i , se tendrán $i + 1$ tubos, empezando en T_0 .
- Se tendrá $\forall \sigma (\sigma \in T_j \rightarrow |\sigma \in \{b_1, \dots, b_i\}| = j)$.

Algoritmo

```
1: procedure Cardinal_Sort( $T_0, B$ )
2:   for  $i = 1$  to  $s$  do
3:      $(T_0, T'_0) = \text{separar}(T_0, b_i)$ 
4:     for  $j = 0$  to  $i - 1$  do
5:        $(T'_{j+1}, T''_j) = \text{separar}(T_j, b_i)$ 
6:        $T_j = \text{mezclar}(T'_j, T''_j)$ 
7:     end for
8:      $T_i = T'_i$ 
9:   end for
10:  Return  $[T_0, \dots, T_s]$ 
11: end procedure
```

A fin de ilustrar el comportamiento:

- $A : \{0, 1, 2, 3, 4, 5, 6\}$
- $B : \{1, 2, 4\}$
- $F : \{[2, 6], [3], [4], [2, 4]\}$

Los elementos que cumplen $B \cap D_j$ en F son: $\{[2, 6], [3], [4], [2, 4]\}$.

F codificado y procesado

Codificando F para llevarlo a un tubo tendremos:

$$\begin{bmatrix} [0, 0, 1, 0, 0, 0, 1] \\ [0, 0, 0, 1, 0, 0, 0] \\ [0, 0, 0, 0, 1, 0, 0] \\ [0, 0, 1, 0, 1, 0, 0] \end{bmatrix}$$

Tras la ejecución de *Cardinal_Sort* tendremos:

$$\begin{bmatrix} T_0 : [[0, 0, 0, 1, 0, 0, 0]] \\ T_1 : [[0, 0, 0, 0, 1, 0, 0], [0, 0, 1, 0, 0, 0, 1]] \\ T_2 : [[0, 0, 1, 0, 1, 0, 0]] \\ T_3 : [] \\ T_4 : [] \\ T_5 : [] \\ T_6 : [] \end{bmatrix}$$

Otras Notaciones

- $Cardinal_Sort(T_0)$ cuando $B = A$.
- $Cardinal_Sort(T_0, l, k)$ cuando $B = \{l, l + 1, \dots, k\}$.

Subrutina Llenado en Paralelo

El propósito de la subrutina de llenado es manipular en el tubo T_0 las moléculas en $(\sigma(i))$ y codifiquen su valor con respecto a f en el sub-conjunto A , esto se define como:

- $A = \{1, \dots, p\}$
- $r \in \mathbb{N}$
- $f : A \rightarrow \mathbb{N}$

Delimitación de Regiones

Podemos considerar la idea de segmentar cada molécula σ en T_0 en posibles regiones como:

- $(A\sigma) = \sigma(1) \cdots \sigma(p)$
- $(L\sigma) = \sigma(p+1) \cdots \sigma(p+r)$
- $(F\sigma) = \sigma(p+r+1) \cdots \sigma(p+r+q_f)$
- $(R\sigma) = \sigma(p+r+q_f+1) \cdots$

Notaciones

Denotamos que si $B \subseteq A$, decimos que $f(B) = \sum_{i \in B} f(i)$. Tal suma nos indica el espacio a reservar en el complejo de memoria T_0 para codificar $f(B)$.

Asimismo, definimos $q_f = f(A)$, tal que $A_i = \{0, \dots, i\}$ siendo $(1 \leq i \leq p)$.

Por último, tenemos que r corresponde a la región comprendida entre p y $f(B)$.

Todos estos ingredientes nos dan soporte para definir el tubo T_0 de moléculas σ cuyo $k \geq p + r + q_f$.

Algoritmo

```
1: procedure Parallel_Fill( $T_0, f, p, r$ )
2:   for  $i = 1$  to  $p$  do
3:      $(T_{i,0}^+, T_{i,0}^-) = \text{separar}(T_{i-1}, i)$ 
4:     for  $j = 1$  to  $f(i)$  do
5:        $T_{i,j}^+ = \text{encender}(T_{i,j}^+, p + r + f(A_{i-1}) + j)$ 
6:     end for
7:      $T_i = \text{mezclar}(T_{i,f(i)}^+, T_{i,0}^-)$ 
8:   end for
9:   Return  $T_0$ 
10: end procedure
```

A fin de ilustrar el comportamiento, estudiemos lo siguiente:

- $A : \{1, 2, 3, 4\}$
- $B : \{2, 3, 4\}$
- $T_0 : \{[2], [4], [3]\}$

Para el tubo T_0 lo codificaríamos de la siguiente forma:

$$\begin{bmatrix} 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \end{bmatrix}$$

T_0 procesado

Una vez el tubo T_0 sea procesado por *Parallel_Fill*, tendremos:

$$\begin{bmatrix} [0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0] \\ [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] \\ [0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0] \end{bmatrix}$$

Problema de Suma de Subconjuntos

Se busca determinar si existe en B un subconjunto cuyo valor equivalga a k .

En ese sentido, definimos $A = \{1, \dots, p\}$, $k \in \mathbb{N}$, $w : A \rightarrow \mathbb{N}$, siendo w la función peso tal que $k \leq w(A) = q_w$.

Algoritmo

```
1: procedure Subset_Sum( $p, w, k$ )  
2:    $q_w = \sum_{i=1}^p w(i)$   
3:    $T_0 = \text{Libería}(p + q_w, p)$   
4:    $T_1 = \text{Parallel\_Fill}(T_0, w, p, 0)$   
5:    $T_k = \text{Cardinal\_Sort}(T_1, p + 1, p + q_w)[k]$   
6:   leer( $T_k$ )  
7: end procedure
```

Traza

Teniendo como ejemplo el T_0 anterior:

$$\begin{bmatrix} [0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0] \\ [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] \\ [0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0] \end{bmatrix}$$

Para un $k = 10$, una vez el T_0 sea procesado por *Subset_Sum* tendríamos como salida: $[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$

Problema de la mochila acotado

Es un problema considerado NP-completo, en el que se busca recolectar una serie de objetos cuyo peso sea menor que k y valor mayor o igual al k' . En ese sentido, consideramos los valores: $A = \{1, \dots, p\}$ un conjunto no vacío, $w : A \rightarrow \mathbb{N}$ una función que codifica el valor para un A , $\rho : A \rightarrow \mathbb{N}$ una función que codifica el valor para un A , asimismo, $k, k' \in \mathbb{N}$, tal que $k \leq w(A) = q_w$ y $k' \leq \rho(A) = q_\rho$.

Algoritmo (I)

```
1: procedure Bounded_Knapsack( $p, w, \rho, k, k'$ )  
2:    $q_w = \sum_{i=1}^p w(i); q_\rho = \sum_{i=1}^p \rho(i);$   
3:    $T_0 = \text{Libería}(p + q_w + q_\rho, p)$   
4:    $T_0 = \text{Parallel\_Fill}(T_0, w, p, 0)$   
5:    $T_0 = \text{Cardinal\_Sort}(T_0, p + 1, p + q_w)$   
6:    $T_1 = \emptyset$   
7:    $\dots$   
8: end procedure
```

Algoritmo (II)

```
1: procedure Bounded_Knapsack( $p, w, \rho, k, k'$ )
2:   ...
3:   for  $i = k'$  to  $q_\rho$  do
4:
5:      $T_1 = \text{merge}(T_1, \text{Cardinal\_Sort}(T_0, p + q_w + 1, p + q_w + q_\rho)[i])$ 
6:   end for
7:   leer( $T_1$ )
8: end procedure
```

Problema de la mochila no acotado

Este se distingue de la anterior implementación en el resultado a mostrar, pues ordena los tubos de salida y elije aquel tubo cuyo valor (definido por la función $\rho(B)$) sea el máximo.

Matemáticamente sería: $\rho(B) = \max\{\rho(C) : C \subseteq A \wedge w(C) \leq k\}$.

Algoritmo (I)

```
1: procedure Unbounded_Knapsack( $p, w, \rho, k, k'$ )
2:    $q_w = \sum_{i=1}^p w(i);$ 
3:    $q_\rho = \sum_{i=1}^p \rho(i);$ 
4:    $T_0 = \text{Libería}(p + q_w + q_\rho, p)$ 
5:    $T_0 = \text{Parallel\_Fill}(T_0, w, p, 0)$ 
6:    $T_0 = \text{Cardinal\_Sort}(T_0, p + 1, p + q_w)$ 
7:    $T_1 = \emptyset$ 
8:   for  $i = 1$  to  $k$  do
9:      $T_1 = \text{mezclar}(T_1, \text{Cardinal\_Sort}(T_0, p + 1, p + q_w)[i])$ 
10:  end for
11:  ...
12: end procedure
```

Algoritmo (II)

```
1: procedure Unbounded_Knapsack( $p, w, \rho, k, k'$ )
2:   ...
3:    $T_0 = \text{Parallel\_Fill}(T_1, \rho, p, q_w)$ 
4:    $i = q_\rho; t = 0$ 
5:   while  $i \geq 1 \wedge t \neq 0$  do
6:      $T' = \text{Cardinal\_Sort}(T_0, p + q_w + 1, p + q_w + q_\rho)[i]$ 
7:     if  $T' \neq \emptyset$  then
8:        $\text{leer}(T')$ 
9:        $t = 1$ 
10:    else
11:       $i = i - 1$ 
12:    end if
13:  end while
14: end procedure
```

Bibliografía