

Práctica 1: Complejidad Computacional

Erick Jesús Ríos González

September 19, 2024

1 Problema del Producto de Subconjuntos

1.1 Pseudocódigo

Algorithm 1 SubsetProductProblem

```
1: function GENERATE_SUBSET(A: List of Integers) → List of Integers
2:   Initialize empty list subset
3:   for each element elem in A do
4:     Generate randomly 0 or 1
5:     if random value is 1 then
6:       Add elem to subset
7:     end if
8:   end for
9:   return subset
10: end function
11:
12: function VERIFY_SUBSET(subset: List of Integers, t: Integer) → Boolean
13:   Initialize product = 1
14:   for each element elem in subset do
15:     product ← product × elem
16:     if product > t then
17:       return No
18:     end if
19:   end for
20:   return Yes
21: end function
22:
23: function SUBSET_PRODUCT_DECISION(A: List of Integers, t: Integer) →
   (Boolean, List of Integers)
24:   subset ← GENERATE_SUBSET(A)
25:   is_valid ← VERIFY_SUBSET(subset, t)
26:   return (is_valid, subset)
27: end function
```

1.2 Correctitud

La función `generate_subset` recorre todos los elementos del conjunto A y, para cada elemento, decide aleatoriamente (con probabilidad 50%) si incluirlo en el subconjunto. Como cada elemento es independiente y el proceso es no determinista, cualquier subconjunto posible de A puede ser generado. Esto garantiza que el subconjunto generado es un subconjunto válido de A , ya que solo puede contener elementos de A y no repite elementos. La función `verify_subset` recibe un subconjunto y calcula el producto de sus elementos. A medida que se va calculando el producto, se verifica si este excede el valor t . Si en algún momento el producto excede t , se retorna "No" (False). Si el producto es menor o igual a t para todos los elementos, se retorna "Sí" (True). Esto garantiza que la verificación se realiza correctamente, ya que la multiplicación de elementos y la comparación con t se hace en cada paso, deteniéndose en el momento en que el producto excede t .

1.3 Análisis de Complejidad

La complejidad del algoritmo es polinomial porque la función generar el subconjunto de forma aleatoria es una primitiva, es $O(n)$. Pues no estamos generando todos los subconjuntos posibles, sino que estamos generando un subconjunto arbitrario. La función de verificar el producto de los elementos del subconjunto es $O(n)$ también, puesto que recorremos el subconjunto y multiplicamos los elementos, si el producto excede el valor t , entonces retornamos No, en caso contrario, retornamos Sí. Finalmente la implementación es correcta porque el algoritmo genera un subconjunto de forma aleatoria y verifica si el producto de los elementos del subconjunto es menor o igual a t en tiempo polinomial.

1.4 Ejecución del Algoritmo

```
PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL PORTS OUTPUTS COMMENTS
Element size: 113
Element size: 188
Element size: 137
Element size: 143
Element size: 157
Element size: 153
Element size: 172
Element size: 82
Element size: 48
Element size: 69
Element size: 33
Element size: 85
Subconjunto generado: { 136 173 16 137 107 172 69 33 85 }
¿El producto del subconjunto es <= 1800000? No
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Subset_product_problems
```

(a) Subfigura 1

```
PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL PORTS OUTPUTS COMMENTS
Element size: 23
Element size: 158
Element size: 161
Element size: 188
Element size: 188
Element size: 75
Element size: 54
Element size: 167
Element size: 159
Element size: 170
Element size: 127
Element size: 177
Subconjunto generado: { 107 138 54 159 170 157 }
¿El producto del subconjunto es <= 1800000? No
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Subset_product_problems
```

(b) Subfigura 2

```
PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL PORTS OUTPUTS COMMENTS
Element size: 104
Element size: 67
Element size: 146
Element size: 200
Element size: 139
Element size: 122
Element size: 199
Element size: 176
Element size: 35
Element size: 75
Element size: 83
Element size: 157
Subconjunto generado: { 170 122 184 67 146 200 139 199 35 157 }
¿El producto del subconjunto es <= 1800000? No
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Subset_product_problems
```

(c) Subfigura 3

```
PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL PORTS OUTPUTS COMMENTS
Element size: 91
Element size: 80
Element size: 169
Element size: 167
Element size: 163
Element size: 198
Element size: 53
Element size: 116
Element size: 26
Element size: 159
Element size: 86
Element size: 47
Subconjunto generado: { 117 68 91 80 47 }
¿El producto del subconjunto es <= 1800000? No
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Subset_product_problems
```

(d) Subfigura 4

```
PROBLEMS 92 OUTPUT DEBUG CONSOLE TERMINAL PORTS OUTPUTS COMMENTS
Element size: 182
Element size: 183
Element size: 27
Element size: 188
Element size: 27
Element size: 166
Element size: 175
Element size: 32
Element size: 67
Element size: 192
Element size: 87
Element size: 178
Subconjunto generado: { 11 27 175 192 178 }
¿El producto del subconjunto es <= 1800000? No
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Subset_product_problems
```

(e) Subfigura 5

Figure 1: Cinco ejecuciones del algoritmo desde terminal

2 Problema de la Mochila

3 Pseudocódigo

Algorithm 2 Knapsack Problem Solver

```
1: function GENERATE_SUBSET(Item: List of [item.weight, value]) → List of
   Integers
2:   Initialize empty list subset
3:   for each element elem in Item do
4:     Generate randomly 0 or 1
5:     if random value is 1 then
6:       Add elem to subset
7:     end if
8:   end for
9:   return subset
10: end function
11: function VERIFY_SUBSET(subset: List of Integers, max_capacity: Integer)
   → Boolean, Integer
12:   Initialize total_value ← 0
13:   for each element elem in subset do
14:     total_weight ← total_weight + elem.weight
15:     total_value ← total_value + elem.value
16:     if total_weight ≥ max_capacity then
17:       return (Yes, total_value)
18:     else
19:       return (No,0)
20:     end if-else
21:   end for
22: end function
23: function KNAPSACK_SOLVER(Item: List of [item.weight, value],
   max_capacity: Integer) → (Boolean, List of Integers)
24:   subset ← GENERATE_SUBSET(Item)
25:   (is_valid, total_value) ← VERIFY_SUBSET(subset, max_capacity)
26:   return (is_valid, total_value)
27: end function
```

3.1 Correctitud

La función `generate_subset` recorre todos los elementos del conjunto *Item* y, para cada elemento, decide aleatoriamente (con probabilidad 50%) si incluirlo en el subconjunto. Como cada elemento es independiente y el proceso es no determinista, cualquier subconjunto posible de *Item* puede ser generado. Esto garantiza que el subconjunto generado es un subconjunto válido de *Item*, ya

que solo puede contener elementos de *Item* y no repite elementos. La función `verify_subset` recibe un subconjunto y calcula el peso total de sus elementos, así como la ganancia total. A medida que se va calculando el peso total, se verifica si este excede el valor *max_capacity*. Si en algún momento el producto excede *max_capacity*, se retorna "No" (False). Si el producto es menor o igual a *max_capacity* para todos los elementos, se retorna "Sí" (True). Esto garantiza que la verificación se realiza correctamente, ya que la suma de los pesos y la comparación con *max_capacity* se hace en cada paso, deteniéndose en el momento en que el *total_weight* excede *max_capacity*.

Nota: Para la implementación el programa nos pide encontrar el subconjunto que maximiza la ganancia, por lo que creo un ciclo for con un número de iteraciones definidas desde el inicio, en este caso 1000 iteraciones. Esto sigue siendo correcto, pues durante un número finito de iteraciones, se generan subconjuntos aleatorios y se verifica si el peso total excede la capacidad máxima. De esta forma obtenemos un máximo local que maximiza la ganancia.

3.2 Análisis de Complejidad

La complejidad del algoritmo es polinomial porque la función generar el subconjunto de forma aleatoria es una primitiva, es $O(n)$. Pues no estamos generando todos los subconjuntos posibles, sino que estamos generando un subconjunto arbitrario. La función de verificar el producto de los elementos del subconjunto es $O(n)$ también, puesto que recorremos el subconjunto y sumamos los pesos de los elementos, si el peso total excede el valor *max_capacity*, entonces retornamos No, en caso contrario, retornamos Sí. Finalmente la implementación es correcta porque el algoritmo genera un subconjunto de forma aleatoria y verifica si el producto de los elementos del subconjunto es menor o igual a *max_capacity* en tiempo polinomial.

Nota: La implementación utilizando un número finito de iteraciones sigue siendo polinomial, pero el algoritmo no garantiza que el subconjunto generado sea el óptimo, sino que es un subconjunto que maximiza la ganancia en un número finito de iteraciones.

3.3 Ejecución del Algoritmo

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS GTITLEN COMMENTS
No.
No.
No.
No.
No.
No.
No.
No.
No.
No.
Best subset found:
Item with weight 13 and value 83
Item with weight 52 and value 85
Item with weight 42 and value 53
Item with weight 82 and value 74
Maximum value: 295
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Backpack_problems
```

(a) Subfigura 1

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS GTITLEN COMMENTS
No.
No.
No.
No.
No.
No.
No.
No.
No.
No.
Best subset found:
Item with weight 18 and value 43
Item with weight 26 and value 42
Item with weight 96 and value 100
Item with weight 6 and value 48
Item with weight 5 and value 66
Item with weight 21 and value 85
Maximum value: 384
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Backpack_problems
```

(b) Subfigura 2

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS GTITLEN COMMENTS
No.
No.
No.
No.
No.
No.
No.
No.
No.
No.
Best subset found:
Item with weight 37 and value 35
Item with weight 93 and value 85
Item with weight 69 and value 90
Maximum value: 210
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Backpack_problems
```

(c) Subfigura 3

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS GTITLEN COMMENTS
No.
No.
No.
No.
No.
No.
No.
No.
No.
No.
Best subset found:
Item with weight 21 and value 83
Item with weight 96 and value 92
Item with weight 43 and value 27
Item with weight 26 and value 15
Item with weight 1 and value 97
Item with weight 1 and value 82
Maximum value: 396
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Backpack_problems
```

(d) Subfigura 4

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS GTITLEN COMMENTS
No.
No.
No.
No.
No.
No.
No.
No.
No.
No.
Best subset found:
Item with weight 69 and value 95
Item with weight 26 and value 89
Item with weight 12 and value 76
Item with weight 79 and value 72
Maximum value: 332
(base) soundskydriver@soundskydriver-System-Product-Name:~/Documents/ComputationalComplexity/practica_uno/src/Backpack_problems
```

(e) Subfigura 5

Figure 2: Cinco ejecuciones del algoritmo desde terminal