INSTITUTO TECNOLÓGICO DE COSTA RICA
INGENIERÍA EN COMPUTACIÓN
INVESTIGACIÓN DE OPERACIONES

Informe Proyecto II

Joel Vega Godínez

Profesor:

Carlos Gamboa Venegas

Sede Inter-Universitaria Alajuela
14 de Diciembre 2020

I. Problema Mochila (contenedor)

A. *Definición:*

El problema de la mochila es un problema de optimización combinatoria, que dice que dado un conjunto de artículos $\{1...N\}$, cada articulo i con un peso w_i y un valor o beneficio b_i, se debe determinar la cantidad de cada artículo para incluir en la mochila de modo que el peso total sea menor o igual a W y el valor total sea lo más grande posible.

B. *Experimentos Realizados:*

Entrada:

$W = 50$

Peso	Beneficio	Cantidad
6	2	2
10	3	1
13	9	2
11	5	5
14	6	3
13	3	4
6	10	5
12	9	4
9	1	4
13	1	3

Resultados Obtenidos:

```
Mochila Fuerza Bruta                    Mochila Programación Dinámica
61                                      61
1,1 # articulo 1 1 unidades             1,1 # articulo 1 1 unidades
3,1 # articulo 3 1 unidades             7,5 # articulo 7 5 unidades
7,5 # articulo 7 5 unidades             8,1 # articulo 8 1 unidades
-- 0.4560418128967285 segundos -- -- 0.005997657775878906 segundos --
```

Entrada:

$W = 100$

Peso	Beneficio	Cantidad
13	9	1
10	7	2
12	8	2
10	10	1
15	3	1
13	9	1
15	10	2
14	6	1
14	6	2
14	8	2
11	9	1
12	7	1
12	8	1
10	6	2
10	5	1
11	8	1
13	10	1
12	10	2
13	6	2
14	1	2

Resultados Obtenidos:

```
Mochila Fuerza Bruta                    Mochila Programación Dinámica
77                                      77
7,2 # articulo 7 2 unidades             2,2 # articulo 2 2 unidades
11,1 # articulo 11 1 unidades           3,2 # articulo 3 2 unidades
13,1 # articulo 13 1 unidades           4,1 # articulo 4 1 unidades
17,1 # articulo 17 1 unidades           11,1 # articulo 11 1 unidades
18,2 # articulo 18 2 unidades           16,1 # articulo 16 1 unidades
                                        18,2 # articulo 18 2 unidades
-- 13.013201475143433 segundos --    -- 0.008000850677490234 segundos --
```

Entrada:

$W = 10$

Peso	Beneficio	Cantidad
4	2	1
4	2	2
3	1	2
3	2	1
3	1	2
2	1	1
3	1	1
2	2	1
3	1	1
2	1	1
1	2	1
4	2	2
4	1	1
4	2	2
3	1	1
3	1	2
1	1	1
3	1	2
1	2	2
3	2	1

Resultados Obtenidos:

```
Mochila Fuerza Bruta                    Mochila Programación Dinámica
11                                      11
1,1 # articulo 1 1 unidades             6,2 # articulo 6 2 unidades
8,1 # articulo 8 1 unidades             8,1 # articulo 8 1 unidades
11,1 # articulo 11 1 unidades           11,1 # articulo 11 1 unidades
17,1 # articulo 17 1 unidades           17,1 # articulo 17 1 unidades
19,2 # articulo 19 2 unidades           19,2 # articulo 19 2 unidades
-- 0.0350039005279541 segundos -- -- 0.003000497817993164 segundos --
```

Entrada:

$W = 500$

Peso	Beneficio	Cantidad
113	7	2
107	5	1
184	5	1
112	8	1
102	6	1

Resultados Obtenidos:

```
Mochila Fuerza Bruta                    Mochila Programación Dinámica
28                                      28
1,2 # articulo 1 2 unidades             1,2 # articulo 1 2 unidades
4,1 # articulo 4 1 unidades             4,1 # articulo 4 1 unidades
5,1 # articulo 5 1 unidades             5,1 # articulo 5 1 unidades
-- 0.002001523971557617 segundos -- -- 0.0069997310638427734 segundos --
```

II. Problema Alineamiento de Secuencias

A. *Definición:*

El alineamiento de una pareja de secuencias permite cuantificar el grado de similitud que hay entre ellas y determinar si existe algún tipo de relación entre ambas. Para alinear dos secuencias, de longitud n y m, respectivamente, se compara una con la otra de modo que el número de letras que coincidan en una misma posición sea máximo. Si es necesario, se pueden introducir gaps ("_") en una o ambas de las secuencias para que aumente el número de coincidencias. Cuando las letras coinciden se suma uno al valor de la similitud, si las letras son diferentes, se resta uno a el valor y si se obtiene un gap en cualquiera de las dos secuencias se resta dos al valor.

B. *Experimentos Realizados:*

Entrada:

Scoring	1,-1,-2
Hilera 1	AGT
Hilera 2	AAGC

Resultados Obtenidos:

```
Alineamiento Programación Dinámica
Scoring Final: -1
Hilera 1: _AGT
Hilera 2: AAGC
Tabla de resultados:
   0  -2  -4  -6
  -2   1  -1  -3
  -4  -1   0  -2
  -6  -3   0  -1
  -8  -5  -2  -1

-- 0.002000570297241211 segundos --
```

```
Alineamiento Fuerza Bruta
Scoring Final: -1
Hilera 1: A_GT
Hilera 2: AAGC

-- 0.03700423240661621 segundos --
```

Entrada:

Scoring	1,-1,-2
Hilera 1	TCTG
Hilera 2	AT

Resultados Obtenidos:

```
Alineamiento Programación Dinámica
Scoring Final: -4
Hilera 1: TCTG
Hilera 2: A_T_
Tabla de resultados:
   0  -2  -4
  -2  -1  -1
  -4  -3  -2
  -6  -5  -2
  -8  -7  -4

-- 0.001999378204345703 segundos --
```

```
Alineamiento Fuerza Bruta
Scoring Final: -4
Hilera 1: TCTG
Hilera 2: A_T_

-- 0.035002946853637695 segundos --
```

Entrada:

Scoring	1,-1,-2
Hilera 1	GAGCC
Hilera 2	TCG

Resultados Obtenidos:

```
Alineamiento Programación Dinámica
Scoring Final: -5
Hilera 1: GAGCC
Hilera 2: TCG__
Tabla de resultados:
   0  -2  -4  -6
  -2  -1  -3  -3
  -4  -3  -2  -4
  -6  -5  -4  -1
  -8  -7  -4  -3
 -10  -9  -6  -5

-- 0.002000570297241211 segundos --
```

```
Alineamiento Fuerza Bruta
Scoring Final: -5
Hilera 1: GAGCC
Hilera 2: TCG__

-- 7.770571708679199 segundos --
```

Entrada:

Scoring	1,-1,-2
Hilera 1	GAGCC
Hilera 2	TCG

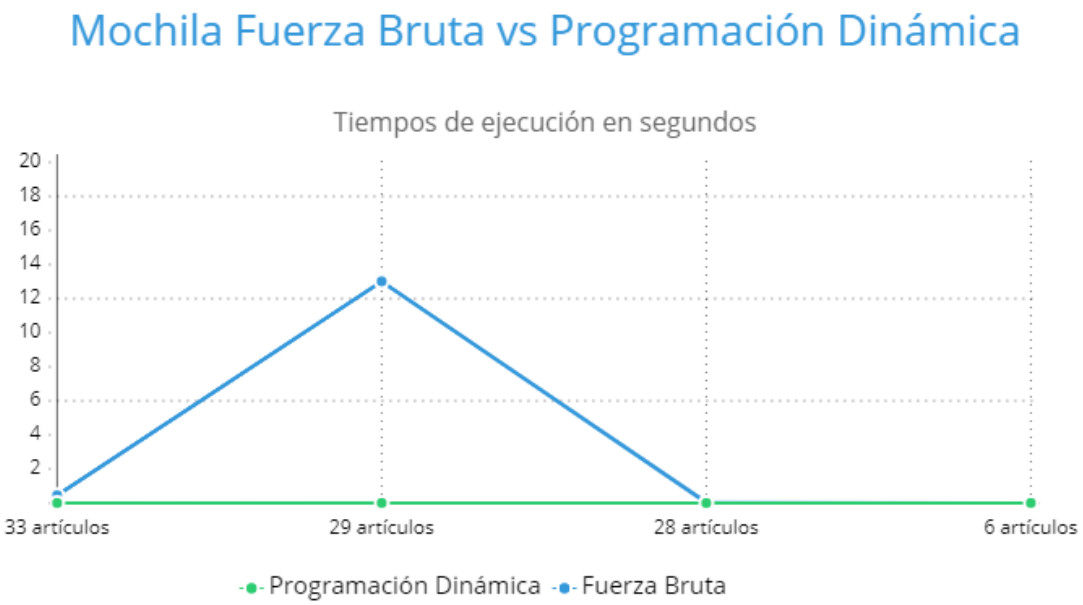
Resultados Obtenidos:

```
Alineamiento Programación Dinámica
Scoring Final: -3
Hilera 1: TCAGC
Hilera 2: AAGGG
Tabla de resultados:
   0  -2  -4  -6  -8 -10
  -2  -1  -3  -5  -7  -9
  -4  -3  -2  -4  -6  -8
  -6  -3  -2  -3  -5  -7
  -8  -5  -4  -1  -2  -4
 -10  -7  -6  -3  -2  -3

-- 0.002000093460083008 segundos --
```

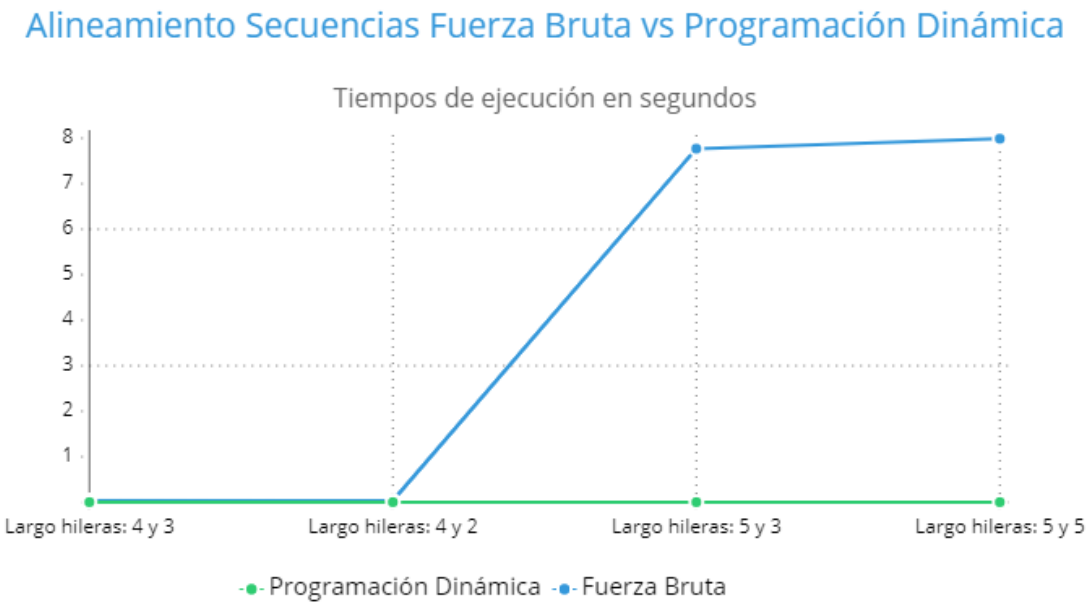
```
Alineamiento Fuerza Bruta
Scoring Final: -3
Hilera 1: TCAGC
Hilera 2: AAGGG

-- 7.989760637283325 segundos --
```

III. Gráficos de los experimentos realizados:

A. *Problema Mochila:*



B. *Problema Alineamiento de Secuencias:*



IV. Complejidad Temporal

A. Mochila Fuerza Bruta

Al existir n elementos, existen 2^n combinaciones de elementos, las cuales deben ser probadas una a una por la fuerza bruta para llegar a la mejor solución. Por lo que la complejidad de este método es:

$$O(2^n)$$

B. Mochila Programación Dinámica

La programación dinámica a diferencia de la fuerza bruta, resuelve cada de los subproblemas más pequeños solo una vez y registra los resultados en una tabla en lugar de resolverlos subproblemas repetidos una y otra vez. Luego, la tabla se usa para obtener una solución del problema original. Por lo que la complejidad de este método esta dada por:

$$O(n * W)$$

Siendo n la cantidad de elementos y W la capacidad de la mochila.

C. Alineamiento Secuencias Fuerza Bruta

El alineamiento de secuencias utilizando fuerza bruta, debe probar todas las permutaciones de la hilera uno, con todas las permutaciones de la hilera dos. Por lo que la complejidad de este método esta dada por:

$$O(n^2)$$

D. Alineamiento Secuencias Programación Dinámica

El alineamiento de secuencias utilizando programación dinámica, al tener una matriz que nos permite verificar entre todas las permutaciones cual es el mayor par de permutaciones que nos dan el máximo valor, entonces la complejidad de este algoritmo estaría dada por:

$$O(n * m)$$

Siendo n y m el largo de las dos hileras respectivamente.

V. Conclusiones

En general, fuerza bruta y programación dinámica son métodos de solución a problemas de optimización, su mayor diferencia es la manera en la cual se aproximan a la solución, debido a que la programación dinámica hace una aproximación dividiendo el problema en subproblemas y obteniendo el resultado óptimo de eso subproblemas, por lo tanto, podemos pensar aproximadamente en la programación dinámica como un método inteligente de fuerza bruta que nos permite analizar todas las soluciones posibles para elegir la mejor . Si el alcance del problema es tal que analizar todas las soluciones posibles es posible y lo suficientemente rápido, la programación dinámica garantiza encontrar la solución óptima, por otro lado, fuerza bruta busca todas las posibles combinaciones una a una que pueda tener la solución, y cuando finaliza muestra la mejor solución.

Otro factor en la que se diferencian estos métodos, es en el tiempo que tardan para encontrar la solución de un mismo problema, como se puede apreciar en los gráficos incluidos anteriormente, los tiempos de ejecución de el problema de alineamiento de secuencias utilizando programación dinámica, esta linea de tiempo se mantiene casi constantemente, a diferencia de cuando se resuelve con fuerza bruta, a medida que el largo de las hileras va aumentando, así lo hace también el tiempo que tarda en encontrar una solución.

Por otro lado, en el gráfico que compara los tiempos de mochila se aprecia que aunque hayan una cantidad de artículos alta en ambos, existen casos en donde la fuerza bruta se acerca al tiempo de la programación dinámica, en este caso lo que afecta en el tiempo de ejecución de este método, es la cantidad de copias por articulo que se posean y el peso que estos pueden aportar, ya que con mayor cantidad de copias de un articulo, existen muchas combinaciones que se deben probar para llegar a la mejor solución.

En conclusión, una de las principales ventajas de usar programación dinámica es que acelera el procesamiento, ya que se usa un algoritmo que ya esta creado y verificado. Como es una técnica de programación recursiva, reduce las líneas de código del programa. La fuerza bruta se puede utilizar para problemas donde el rango de elementos que se ocupan probar sea pequeño, ya que se debe probar uno a uno.

Por otra parte, al hacer uso de la programación dinámica se necesita mucha memoria para almacenar el resultado calculado de cada subproblema, sin poder garantizar que el valor almacenado se utilizará o no. Mientras que el uso de memoria que se hace en la fuerza bruta es mucho menor. Pero al no tener una estructura de datos en donde se verifica cuales combinaciones se han probado, la fuerza bruta puede probar combinaciones que ya se hayan probado.