# Reporte Escrito - Proyecto 2
# Desempeño de Programación Dinámica
# Investigación de Operaciones IC-6400

Guerrero. P*, Guzmán.J†,

***Abstract*—El presente reporte presenta los resultados del segundo proyecto del curso de Investigación de Operaciones de la Escuela de Ingeniería en Computación del Tecnológico de Costa Rica. El cual tiene como principal objetivo el brindarle al estudiante la posibilidad de desarrollar conocimientos en el área de programación, para lo cual se le presentan dos problemas, el primero se denomina contenedor o mochila, en el cual se tendrá que implementar tres versiones diferentes con la finalidad de comprender más a fondo la naturaleza de este. El segundo se denomina mina de oro, de este únicamente se implementarán dos métodos. A continuación, veremos los resultados y el análisis de distintos experimentos realizados con la finalidad de demostrar el comportamiento de los algoritmos desarrollados por las estudiante con los cuales se puede concluir que la búsqueda de la forma óptima de realizar las cosas siempre resulta ser valiosa dado que se puede observar una gran ventaja entre la comparación de la programación dinámica a con el de fuerza bruta pero no sin dejar de lado la complejo que resulta llegar a pensar el cómo las dichos problema**

## I. Introducción

El presente documento pretende demostrar los resultados del segundo proyecto de Investigación de operaciones, donde se tratarán diferentes etapas desde el diseño, desarrollo y experimentación con los algoritmos que se asignaron.

Con este proceso se pretende mejorar los conocimientos en esta área de la carrera, y a su vez lograr un mayor entendimiento de los problemas y las diferentes maneras que existen para resolverlo, esto consecuentemente llevándonos a la analizar la importancia del diseño.

## II. Problemas

### A. Problema del contenedor

Se tienen n elementos distintos, y un contenedor que soporta una cantidad especifica de peso W. Cada elemento i tiene un peso $w_i$, un valor o beneficio asociado dado por $b_i$. El problema consiste en agregar elementos al contenedor de forma que se maximice el beneficio total sin superar el peso máximo que soporta el contenedor. La solución debe ser dada con la lista de los elementos que se ingresaron y el valor del beneficio máximo obtenido.

Para este problema se presentarán tres soluciones la primera es fuerza bruta, también llamada búsqueda exhaustiva. La segunda es programación dinámica bottom-up, y la tercera es programación dinámica top-down utilizando memoization

### B. Problema de la Mina de Oro

Una mina de oro de dimensiones n*m. Cada campo de la mina contienen un número entero positivo el cual es la cantidad de oro en toneladas. Inicialmente un minero puede estar en la primera columna pero en cualquier fila y puede mover a la derecha, diagonal arriba a la derecha o diagonal abajo a la derecha de una celda dada. El programa debe retornar la máxima cantidad de oro que el minero puede recolectar llegando hasta el límite derecho de la mina, y las casillas seleccionadas.

Al igual que el problema anterior, este tiene más de una posible solución, por lo cual serán implementados dos algoritmos de diferente naturaleza para obtener los resultados esperados, el primero será fuerza bruta y el segundo será programación dinámica, con el fin de hacer una análisis de su comportamiento

## III. Diseño de algoritmos

### A. Contenedor

La diferencia entre la programación dinámica y la fuerza bruta radica en que la dinámica "recuerda los estados que ya ha calculado" y subdivide al problema en esos estados. Mientras que la fuerza bruta calcula todas las posibles soluciones. Se utilizo programación dinámica con top down y con bottom-up para implementar el algoritmo, utilizando memoization, recursividad y divide y vencerás, partiendo desde un estado general del problemas hasta los subproblemas específicos. Cuando el algoritmo se encuentra en un estado visitado previamente simplemente retorna el valor guardado en el memoize, de esta manera no se repiten pasos y la complejidad del problema disminuye.

Fuerza bruta

```
DEFINE FUNCTION knapsack_brute_force(items, max_weight):
    SET knapsack TO []
    SET best_weight TO 0
    SET best_value TO 0
    ids =[]
    FOR item_set  IN powerset(items):
        SET set_weight TO sum([e[1] FOR e IN item_set])
        SET set_value TO sum([e[2] FOR e IN item_set])
        IF set_value > best_value and set_weight <= max_weight:
            SET best_value TO set_value
```

```
            SET best_weight TO set_weight

            SET knapsack TO item_set

        FOR i IN knapsack:

            SET ids TO [i[0]] +ids

        RETURN ids, best_weight, best_value
```

Programación dinámica: bottom-up

```
DEFINE FUNCTION dynamicKnapSack(W, items, n):

    SET V TO []


    FOR i IN range(n + 1):

        V.append([0] * (W + 1))


    FOR i IN range(1, n + 1):

        FOR w IN range(W + 1):

            IF (items[i - 1][1] > w):

                SET V[i][w] TO V[i - 1][w]

            ELSEIF (items[i - 1][2] + V[i - 1][w - items[i - 1][1]] > V[i - 1][w]):

                SET V[i][w] TO items[i - 1][2] + V[i - 1][w - items[i - 1][1]]

            ELSE:

                SET V[i][w] TO V[i - 1][w]

    RETURN V
```

Programación dinámica: top-down

```
DEFINE FUNCTION topDownKnapsack(item, capacity, memo):

    if(capacity < 0):

        RETURN -(1<<60)

    ELSEIF capacity EQUALS 0 or item EQUALS 0:

        RETURN 0

    ELSEIF memo[item][capacity]:

        RETURN memo[item][capacity]

    ELSE:

        SET memo[item][capacity] TO max(val[item] +
        topDownKnapsack(item-1, capacity - wt[item],memo),
        topDownKnapsack(item-1,capacity ,memo))

    RETURN memo[item][capacity]
```

### *B. Mina de oro*

La diferencia entre la programación dinámica y la fuerza bruta radica en que la dinámica "recuerda los estados que ya ha calculado" y subdivide al problema en esos estados. Mientras que la fuerza bruta calcula todas las posibles soluciones. Se utilizo programación dinámica top down para implementar el algoritmo, utilizando memoization, recursividad y divide y vencerás, partiendo desde un estado general del problemas hasta los subproblemas específicos. Cuando el algoritmo se encuentra en un estado visitado previamente simplemente retorna el valor guardado en el memoize, de esta manera no se repiten pasos y la complejidad del problema disminuye.

Implementación dinámica

```
DEFINE FUNCTION dp_max(mine,dp,i,j,N,M,route):

    SET indexes TO [1,0,-1]

    IF(i < 0 or i EQUALS N):

        RETURN 0

    ELSE IF (j EQUALS 0):

        SET dp[i][j] TO mine[i][j]

    ELSE IF (dp[i][j]!=-1000000):

        RETURN dp[i][j]

    ELSE:
        SET left_down TO i+1

        SET left_up TO i-1

        SET aux TO [dp_max(mine, dp, left_down, j-1, N, M,route),
                    dp_max(mine, dp, i, j-1, N, M,route),
                    dp_max(mine, dp, left_up, j-1, N, M,route)]

        SET aux_index,max_value TO max_aux(aux,3)

        SET route_aux TO i+indexes[aux_index]

        SET route[i][j] TO route[route_aux][j-1]+[route_aux]

        SET dp[i][j] TO mine[i][j] + max_value

    RETURN dp[i][j]
```

Implementación de fuerza bruta

```
DEFINE FUNCTION bruteForce(mine,i,j,N,M,route):

    IF (i < 0 or i EQUALS N):

        RETURN 0

    ELSE IF(j==0):

        RETURN mine[i][j]

    ELSE:

        SET left_down TO i+1

        SET left_up TO i-1

    RETURN mine[i][j] +
            max(bruteForce(mine, left_down, j-1, N, M),
            bruteForce(mine,i, j-1, N, M),
            bruteForce(mine, left_up, j-1, N, M))
```

## IV. EXPERIMENTOS

### *A. Contenedor*

#### *1) Experimento 1:*

- Nombre del archivo: experimentoC1.txt
- Capacidad de la Mochila: 20
- iteraciones: 10
- Pesos: : 8, 7, 4, 8, 7
- Beneficios: 13, 13, 14, 11, 10

```
./contenedor.py 1 -p 20 5 4-8 10-14 10
./contenedor.py 2 -p 20 5 4-8 10-14 10
./contenedor.py 3 -p 20 5 4-8 10-14 10
./contenedor.py 1 -a experimentoC1.txt 10
./contenedor.py 2 -a experimentoC1.txt 10
./contenedor.py 3 -a experimentoC1.txt 10
```

#### *2) Experimento 2:*

- Nombre del archivo: experimentoC2.txt
- Capacidad de la Mochila: 40
- iteraciones: 10
- Pesos: 7, 1, 11, 3, 5, 10, 8, 4, 10, 1, 8, 5, 5, 14, 10
- Beneficios: 15, 5, 9, 12, 7, 9, 23, 11, 12, 4, 11, 18, 19, 19, 9

```
./contenedor.py 1 -p 40 15 1-15 1-30 10
./contenedor.py 2 -p 40 15 1-15 1-30 10
./contenedor.py 3 -p 40 15 1-15 1-30 10
./contenedor.py 1 -a experimentoC2.txt 10
./contenedor.py 2 -a experimentoC2.txt 10
./contenedor.py 3 -a experimentoC2.txt 10
```

*3) Experimento 3:* Este experimento se realizó con gran cantidad de items con el objetivo de probar los algoritmos en condiciones más demandantes que en casps anteriores, además se realizó con pesos de un rango mayor dificultando la tarea de encontrar el beneficio máximo.

- Nombre del archivo: experimentoC3.txt
- Capacidad de la Mochila: 200
- iteraciones: 10
- Pesos:
  18, 27, 38, 48, 6, 24, 50, 17, 36, 14, 48, 7, 14, 12, 32, 1, 49, 10, 4, 45, 10, 34, 32, 49, 45, 48, 49, 25, 29, 29, 8, 35, 11, 7, 13, 47, 15, 42, 13, 22, 13, 50, 16, 20, 27, 47, 48, 5, 2, 38, 39, 4, 4, 17, 50, 19, 27, 28, 34, 50, 50, 22, 14, 2, 25, 17, 15, 14, 38, 9, 47, 19, 36, 10, 29, 10, 43, 36, 46, 31, 47, 3, 50, 17, 13, 25, 5, 11, 41, 14, 34, 49, 44, 34, 46, 48, 38, 12, 40, 13

- Beneficios:
  40, 33, 44, 31, 17, 39, 10, 21, 26, 17, 31, 19, 14, 53, 26, 48, 19, 3, 1, 39, 5, 16, 16, 17, 53, 22, 53, 47, 35, 30, 38, 16, 4, 47, 4, 8, 5, 54, 43, 47, 54, 39, 48, 14, 48, 21, 44, 34, 42, 37, 23, 13, 23, 25, 21, 24, 14, 36, 59, 11, 22, 38, 44, 4, 14, 37, 2, 46, 31, 42, 16, 18, 50, 56, 19, 14, 59, 26, 15, 26, 23, 37, 34, 48, 28, 22, 35, 53, 22, 44, 13, 6, 17, 6, 23, 21, 38, 50, 53, 11

```
./contenedor.py 1 -p 200 100 1-50 1-60 10
./contenedor.py 2 -p 200 100 1-50 1-60 10
./contenedor.py 3 -p 200 100 1-50 1-60 10
./contenedor.py 1 -a experimentoC3.txt 10
./contenedor.py 2 -a experimentoC3.txt 10
./contenedor.py 3 -a experimentoC3.txt 10
```

*4) Experimento 4:*

- Nombre del archivo: experimentoC4.txt
- Capacidad de la Mochila: 200
- iteraciones: 10
- Pesos: 46, 23, 30, 28, 34, 14, 18, 23, 2, 20
- Beneficios: 42, 17, 38, 48, 28, 34, 61, 27, 68, 68

```
./contenedor.py 1 -p 200 10 1-50 1-80 10
./contenedor.py 2 -p 200 10 1-50 1-80 10
./contenedor.py 3 -p 200 10 1-50 1-80 10
./contenedor.py 1 -a experimentoC5.txt 10
./contenedor.py 2 -a experimentoC5.txt 10
./contenedor.py 3 -a experimentoC5.txt 10
```

*5) Experimento 5:*

- Nombre del archivo: experimentoC5.txt
- Capacidad de la Mochila: 200
- iteraciones: 10
- Pesos: 65, 57, 76, 81, 56
- Beneficios: 7, 24, 12, 4, 71

```
./contenedor.py 1 -p 200 5 1-90 1-80 10
./contenedor.py 2 -p 200 5 1-90 1-80 10
./contenedor.py 3 -p 200 5 1-90 1-80 10
./contenedor.py 1 -a experimentoC5.txt 10
./contenedor.py 2 -a experimentoC5.txt 10
./contenedor.py 3 -a experimentoC5.txt 10
```

*6) Experimento 6:* Este experimento se realizó con gran cantidad de items con el objetivo de probar los algoritmos en condiciones más demandantes que en casps anteriores,a diferencia del experimento tres se tiene un menor rango de pesos haciendo la tarea de los algoritmos más sencillos, y por ende menos exigente en ese aspecto por ende se añadieron mayor cantidad de elementos.

- Nombre del archivo: experimentoC6.txt
- Capacidad de la Mochila: 200
- iteraciones: 10
- Pesos:
  2, 9, 9, 5, 1, 1, 1, 4, 9, 6, 7, 1, 4, 10, 4, 2, 9, 8, 10, 7, 2, 7, 10, 1, 8, 6, 3, 8, 8, 7, 7, 1, 1, 9, 1, 2, 7, 7, 2, 4, 1, 2, 9, 6, 9, 10, 2, 5, 6, 10, 5, 7, 2, 4, 10, 4, 3, 4, 10, 1, 1, 2, 7, 9, 3, 7, 1, 9, 10, 6, 4, 3, 2, 7, 2, 9, 8, 4, 6, 6, 6, 1, 1, 8, 7, 5, 4, 9, 10, 8, 5, 5, 3, 7, 5, 4, 8, 6, 1, 2, 1, 8, 10, 10, 4, 3, 1, 1, 3, 5, 1, 7, 9, 10, 6, 3, 7, 2, 4, 4

- Beneficios:
  15, 12, 3, 4, 2, 14, 8, 7, 12, 11, 4, 7, 4, 6, 10, 11, 9, 13, 12, 7, 1, 12, 14, 12, 10, 14, 11, 14, 3, 5, 1, 3, 2, 8, 5, 6, 12, 14, 15, 13, 8, 11, 8, 9, 1, 6, 2, 9, 3, 12, 15, 4, 2, 13, 4, 1, 15, 7, 10, 1, 8, 4, 10, 11, 4, 2, 11, 4, 1, 9, 14, 7, 9, 13, 7, 14, 3, 8, 15, 4, 8, 15, 2, 1, 5, 6, 13, 15, 10, 12, 1, 11, 11, 14, 6, 8, 13, 1, 14, 12, 6, 14, 6, 3, 6, 12, 1, 15, 10, 8, 14, 1, 7, 6, 10, 9, 13, 7, 9, 8

```
./contenedor.py 1 -p 400 120 1-10 1-15 1
./contenedor.py 2 -p 400 120 1-10 1-15 1
./contenedor.py 3 -p 400 120 1-10 1-15 1
./contenedor.py 1 -a experimentoC6.txt 10
./contenedor.py 2 -a experimentoC6.txt 10
./contenedor.py 3 -a experimentoC6.txt 10
```

### *B. Mina de Oro*

*1) Experimento 1:*

- Nombre del archivo: mineExperiment1.txt
- N: 4
- M: 6
- min: 5
- max: 35
- iteraciones: 10

```
./goldmine.py 1 -p 4 6 5 35 10
./goldmine.py 2 -p 4 6 5 35 10
./goldmine.py 1 -a mineExperiment1.txt 10
./goldmine.py 2 -a mineExperiment1.txt 10
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 12 | 7 | 14 | 20 | 4 | 3 | 14 |
| 7 | 14 | 6 | 6 | 14 | 17 | 14 |
| 16 | 2 | 8 | 6 | 18 | 11 | 18 |
| 4 | 14 | 5 | 19 | 19 | 2 | 18 |

*2) Experimento 2:*

- Nombre del archivo: mineExperiment2.txt
- N: 6
- M: 4

- min: 1
- max: 50
- iteraciones: 10

```
./goldmine.py 1 -p 6 4 1 50 10
./goldmine.py 2 -p 6 4 1 50 10
./goldmine.py 1 -a mineExperiment2.txt 10
./goldmine.py 2 -a mineExperiment2.txt 10
```

| | | | |
|---|---|---|---|
| 41 | 34 | 42 | 18 |
| 29 | 41 | 49 | 23 |
| 33 | 47 | 36 | 21 |
| 49 | 50 | 36 | 43 |
| 2 | 5 | 1 | 19 |
| 28 | 13 | 42 | 40 |

*3) Experimento 3:*

- Nombre del archivo: mineExperiment3.txt
- N: 6
- M: 6
- min: 1
- max: 50
- iteraciones: 10

```
./goldmine.py 1 -p 6 6 1 50 10
./goldmine.py 1 -p 6 6 1 50 10
./goldmine.py 1 -a mineExperiment3.txt 10
./goldmine.py 2 -a mineExperiment3.txt 10
```

| | | | | | |
|---|---|---|---|---|---|
| 15 | 18 | 17 | 15 | 16 | 41 |
| 3 | 42 | 45 | 20 | 44 | 40 |
| 7 | 7 | 4 | 8 | 12 | 48 |
| 5 | 23 | 31 | 9 | 34 | 4 |
| 49 | 12 | 21 | 33 | 27 | 43 |
| 10 | 50 | 31 | 2 | 7 | 26 |

*4) Experimento 4:*

- Nombre del archivo: mineExperiment4.txt
- N: 4
- M: 4
- min: 1
- max: 100
- iteraciones: 10

```
./goldmine.py 1 -p 4 4 1 100 10
./goldmine.py 1 -p 4 4 1 100 10
./goldmine.py 1 -a mineExperiment4.txt 10
./goldmine.py 2 -a mineExperiment4.txt 10
```

| | | | |
|---|---|---|---|
| 87 | 89 | 67 | 81 |
| 96 | 43 | 49 | 35 |
| 15 | 28 | 82 | 90 |
| 15 | 5 | 10 | 23 |

*5) Experimento 5:*

- Nombre del archivo: mineExperiment5.txt
- N: 7
- M: 7
- min: 20
- max: 200
- iteraciones: 10

```
./goldmine.py 1 -p 7 7 20 200 10
./goldmine.py 2 -p 7 7 20 200 10
./goldmine.py 1 -a mineExperiment5.txt 10
./goldmine.py 2 -a mineExperiment5.txt 10
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 165 | 184 | 54 | 69 | 116 | 53 | 91 |
| 70 | 123 | 62 | 129 | 129 | 44 | 50 |
| 67 | 121 | 88 | 35 | 130 | 142 | 123 |
| 95 | 163 | 50 | 148 | 41 | 50 | 101 |
| 134 | 61 | 130 | 30 | 116 | 34 | 171 |
| 158 | 26 | 99 | 135 | 114 | 43 | 85 |
| 92 | 145 | 84 | 55 | 172 | 161 | 185 |

*6) Experimento 6:*

- Nombre del archivo: mineExperiment6.txt
- N: 9
- M: 4
- min: 1
- max: 235
- iteraciones: 10

```
./goldmine.py 1 -p 9 4 1 235 10
./goldmine.py 1 -p 9 4 1 235 10
./goldmine.py 1 -a mineExperiment6.txt 10
./goldmine.py 2 -a mineExperiment6.txt 10
```

| | | | |
|---|---|---|---|
| 202 | 205 | 2 | 19 |
| 209 | 107 | 113 | 152 |
| 178 | 116 | 207 | 102 |
| 79 | 210 | 151 | 86 |
| 40 | 120 | 155 | 234 |
| 187 | 34 | 122 | 38 |
| 13 | 64 | 38 | 37 |
| 179 | 6 | 126 | 49 |
| 23 | 223 | 133 | 77 |

## V. Resultados

A continuación se muestran los resultados de los experimentos para cada problema, contenedor y mina de oro. Se utilizan gráficas para comparar los tiempos de ejecución entre las iteraciones de cada experimento y el tiempo de ejecución promedio de los experimentos tratados con programación dinámica y fuerza bruta.

### *A. Contenedor*

*1) Experimento 1:*

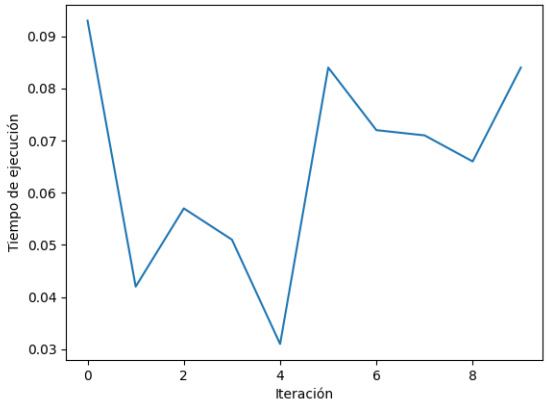


Figure 1. Gráfico de tiempo de iteraciones del experimento 1 con fuerza bruta

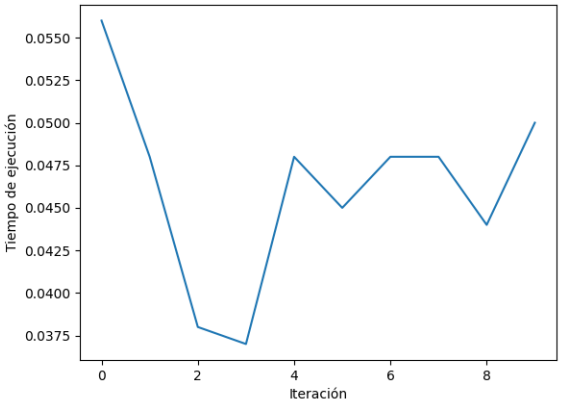


Figure 2. Gráfico de tiempo de iteraciones del experimento 1 con s programación dinámica bottom-up

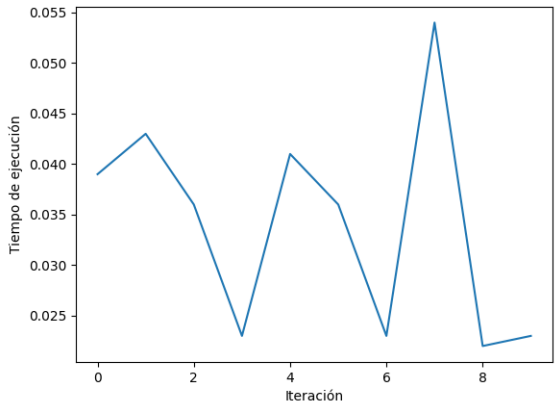


Figure 3. Gráfico de tiempo de iteraciones del experimento 1 con dinámica top-down

```
Beneficio maximo: 40
Incluidos: 3, 2, 1
Average Execution Time fuerza bruta:
0.06509999999999999 ms
Average Execution Time bottom up:
0.0462 ms
Average Execution Time top-down:
03400000000000001 ms
```

*2) Experimento 2:*

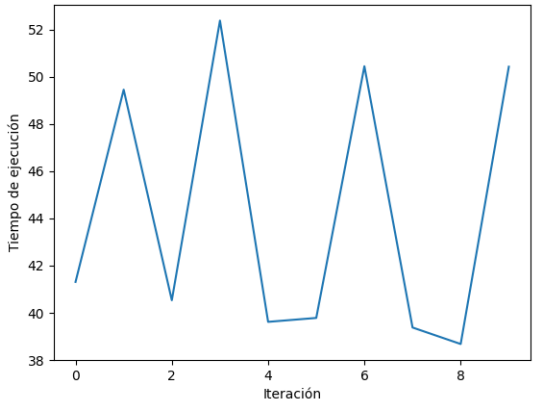


Figure 4. Gráfico de tiempo de iteraciones del experimento 1 con fuerza bruta

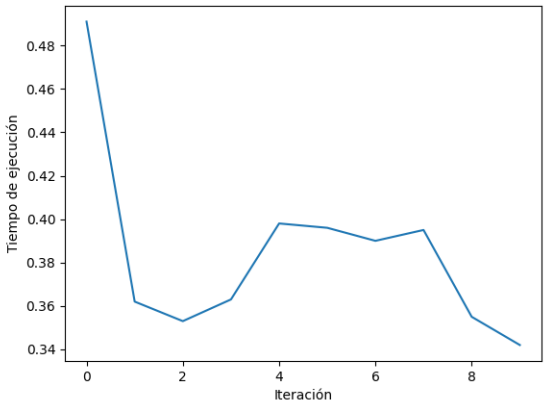


Figure 5. Gráfico de tiempo de iteraciones del experimento 1 con s programación dinámica bottom-up

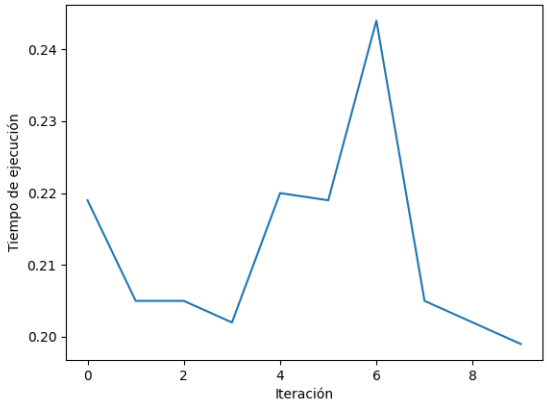


Figure 6. Gráfico de tiempo de iteraciones del experimento 1 con dinámica top-down

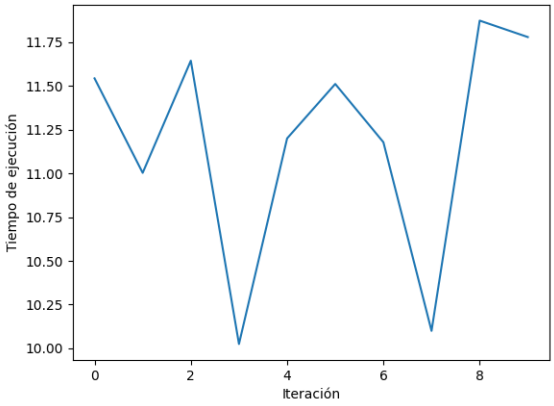


Figure 8. Gráfico de tiempo de iteraciones del experimento 1 con dinámica top-down

```
Beneficio maximo: 114
Incluidos: 13, 12, 10, 8, 7, 5, 4, 2, 1
Average Execution Time fuerza bruta:
44.20219999999999 ms
Average Execution Time bottom up:
0.3845 ms
Average Execution Time top-down:
0.21200000000000002 ms
```

```
Beneficio m ximo: 915
Incluidos: 98, 90, 88, 87, 84, 82,
74, 70, 68, 63,53, 52, 49, 48, 43,
41, 39, 34, 31, 16, 14, 5
Average Execution Time fuerza bruta:
Lack of memory
Average Execution Time bottom up:
7.8783 ms
Average Execution Time top-down:
11.185599999999997 ms
```

*3) Experimento 3:*

*4) Experimento 4:*

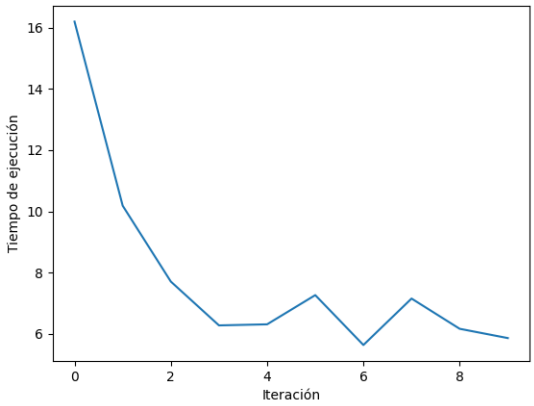


Figure 7. Gráfico de tiempo de iteraciones del experimento 1 con s programación dinámica bottom-up

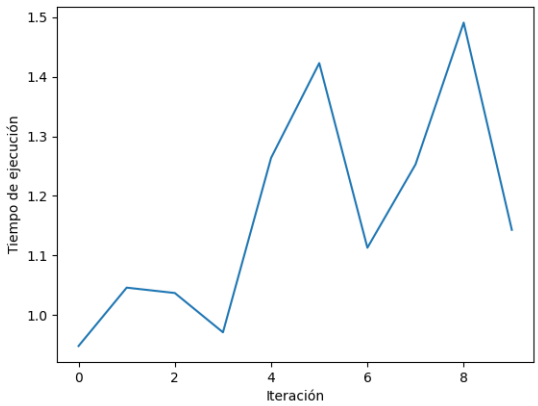


Figure 9. Gráfico de tiempo de iteraciones del experimento 1 con fuerza bruta

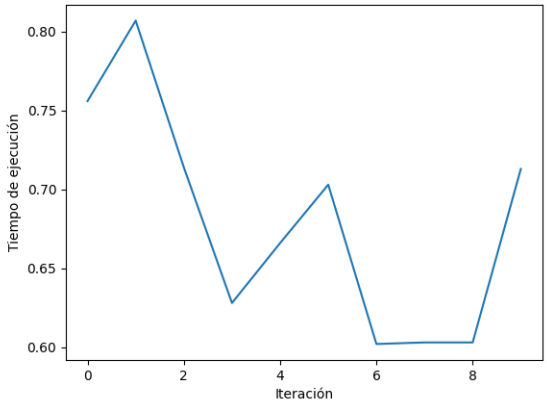


Figure 10. Gráfico de tiempo de iteraciones del experimento 1 con dinámica bottom-up

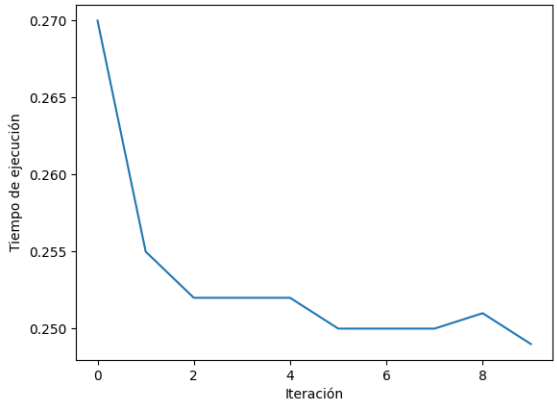


Figure 11. Gráfico de tiempo de iteraciones del experimento 1 con dinámica top-down

```
Beneficio maximo: 389
Incluidos: 10, 9, 8, 7, 6, 5, 4, 3, 2
Average Execution Time fuerza bruta:
1.1689 ms
Average Execution Time bottom up:
0.6795 ms
Average Execution Time top-down:
0.2531 ms
```

*5) Experimento 5:*

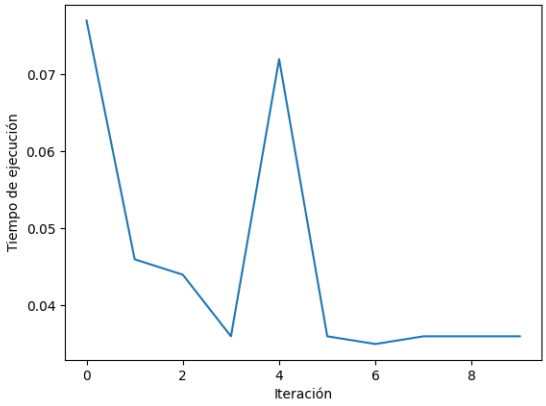


Figure 12. Gráfico de tiempo de iteraciones del experimento 1 con fuerza bruta

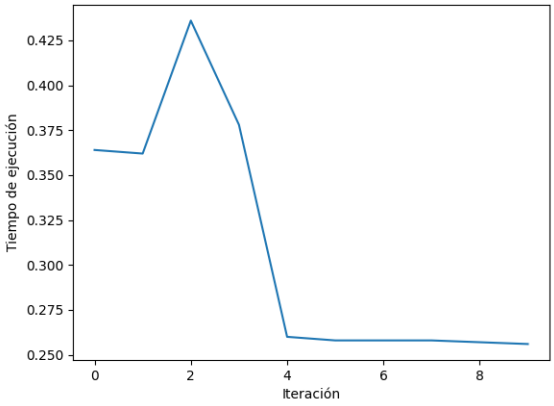


Figure 13. Gráfico de tiempo de iteraciones del experimento 1 con s programación dinámica bottom-up

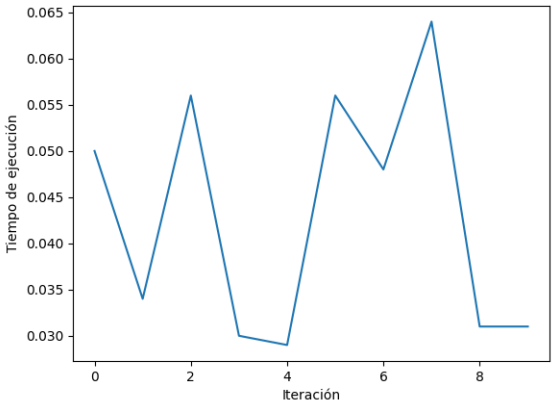


Figure 14. Gráfico de tiempo de iteraciones del experimento 1 con dinámica top-down

```
Beneficio m ximo: 107
Incluidos: 5, 3, 2
Average Execution Time fuerza bruta:
0.04540000000000001 ms
Average Execution Time bottom up:
0.3087 ms
Average Execution Timetop-down:
0.04290000000000001 ms
```

*6) Experimento 6:*

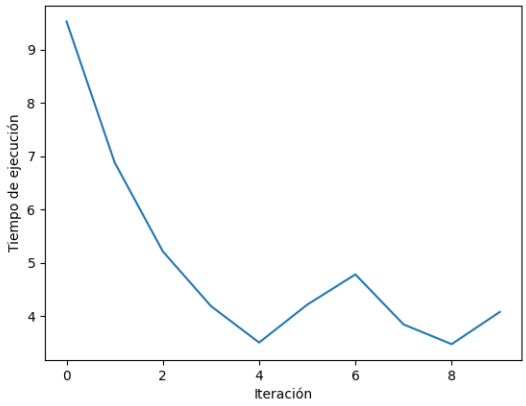


Figure 15. Gráfico de tiempo de iteraciones del experimento 1 con s programación dinámica bottom-up

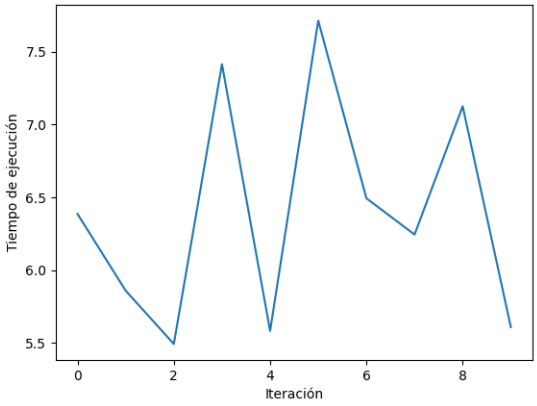


Figure 16. Gráfico de tiempo de iteraciones del experimento 1 con dinámica top-down

```
Beneficio m ximo: 437
Incluidos: 119, 118, 116, 111, 109, 108,
106, 101, 100, 99, 93, 92, 87, 82, 79, 75,
73, 72, 71,67, 61, 57, 54, 51, 42, 41, 40,
39, 36,35, 32, 27, 26, 24, 16, 15, 12, 7,
6, 5, 1
Average Execution Time fuerza bruta:
Lack of Memory
Average Execution Time bottom up:
4.9716000000000005 ms
Average Execution Time top-down:
6.392499999999999 ms
```

*B. Mina de oro*

*1) Experimento 1:*

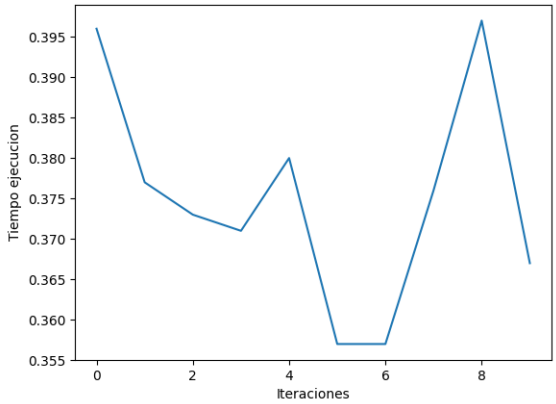


Figure 17. Gráfico de tiempo de iteraciones del experimento 1 con fuerza bruta

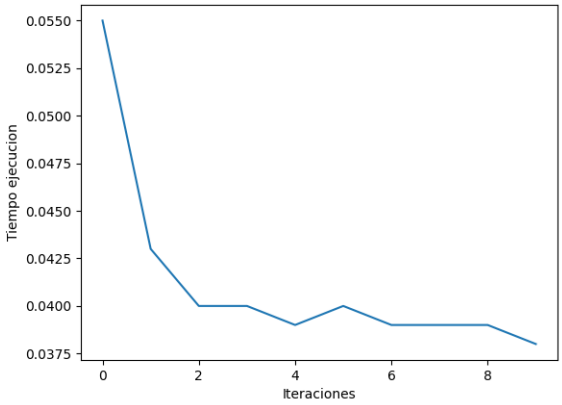


Figure 18. Gráfico de tiempo de iteraciones del experimento 1 con programación dinámica

```
Output: 168

(1,0)->(0,1)->(1,2)->(1,3)->(1,4)->(1,5)

Average Execution Time BF: 0.37509999999993 ms
Average Execution Time DP: 0.04119999999994 ms
```

*2) Experimento 2:*

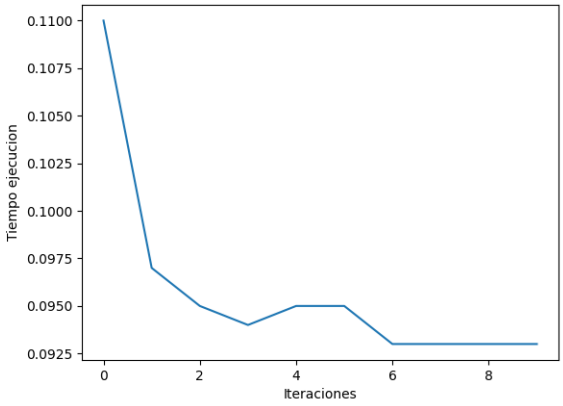


Figure 19. Gráfico de tiempo de iteraciones del experimento 2 con fuerza bruta

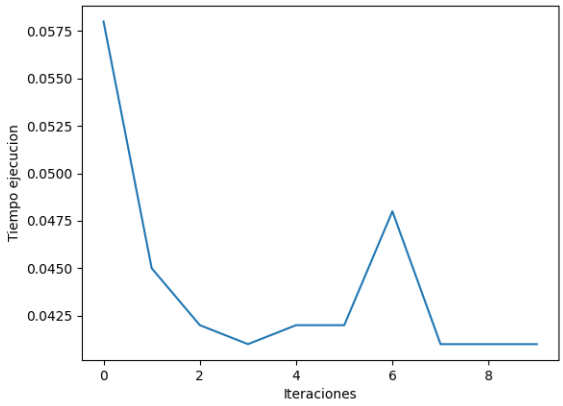


Figure 20. Gráfico de tiempo de iteraciones del experimento 2 con programación dinámica

Output: 178

(3,0)->(3,1)->(3,2)->(3,3)

Average Execution Time BF: 0.09579999999998 ms
Average Execution Time DP: 0.04409999999999 ms

*3) Experimento 3:*

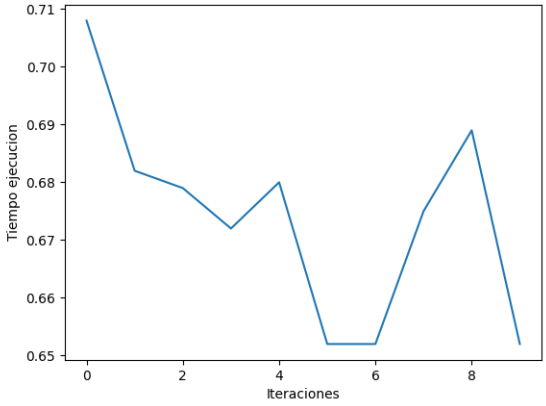


Figure 21. Gráfico de tiempo de iteraciones del experimento 3 con fuerza bruta

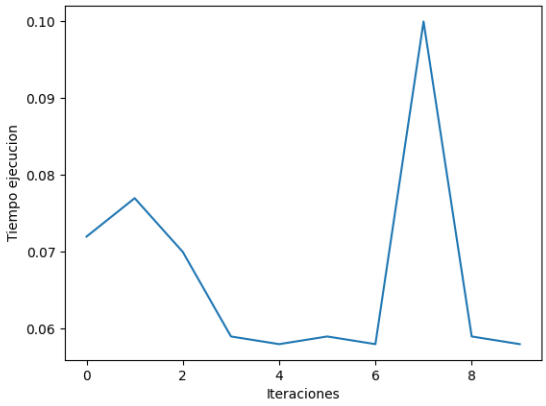


Figure 22. Gráfico de tiempo de iteraciones del experimento 3 con programación dinámica

Output: 245

(4,0)->(5,1)->(5,2)->(4,3)->(3,4)->(2,5)

Average Execution Time BF: 0.6740999999999 ms
Average Execution Time DP: 0.0670000000002 ms

*4) Experimento 4:*

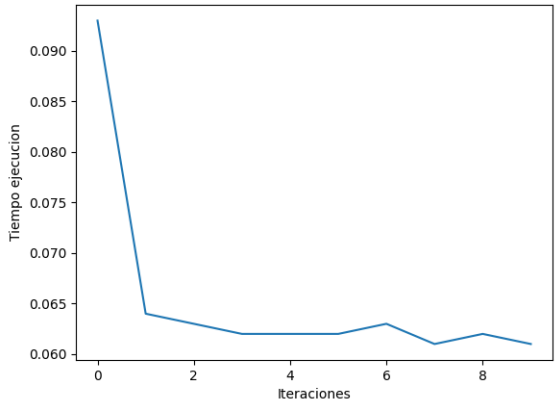


Figure 23. Gráfico de tiempo de iteraciones del experimento 4 con fuerza bruta

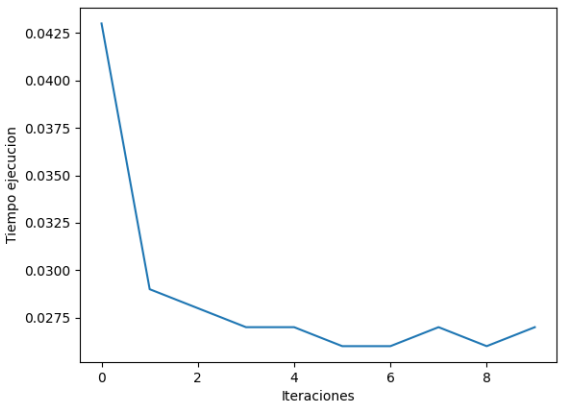


Figure 24. Gráfico de tiempo de iteraciones del experimento 4 con programación dinámica

Output: 333

(1,0)->(0,1)->(0,2)->(0,3)

Average Execution Time BF: 0.0653 ms
Average Execution Time DP: 0.0286000000004 ms

*5) Experimento 5:*

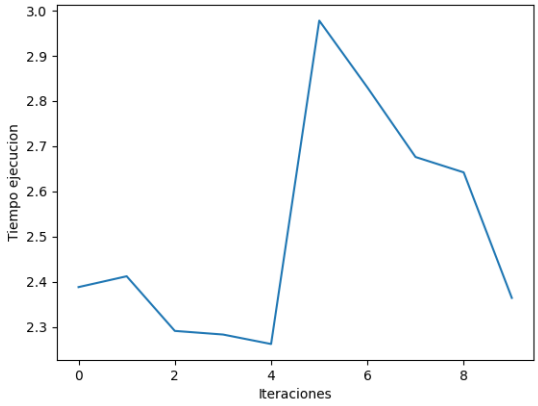


Figure 25. Gráfico de tiempo de iteraciones del experimento 5 con fuerza bruta

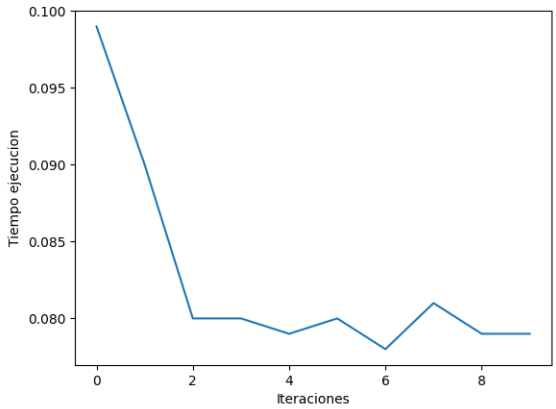


Figure 26. Gráfico de tiempo de iteraciones del experimento 5 con programación dinámica

Output: 1080

(4,0)->(3,1)->(4,2)->(5,3)->(6,4)->(6,5)->(6,6)

Average Execution Time BF: 2.5126 ms
Average Execution Time DP: 0.0824999999999999 ms

*6) Experimento 6:*

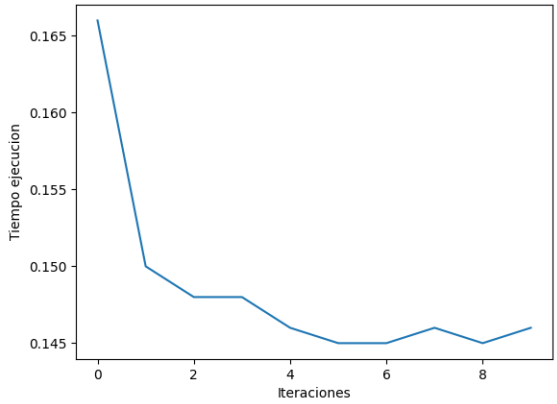


Figure 27. Gráfico de tiempo de iteraciones del experimento 6 con fuerza bruta

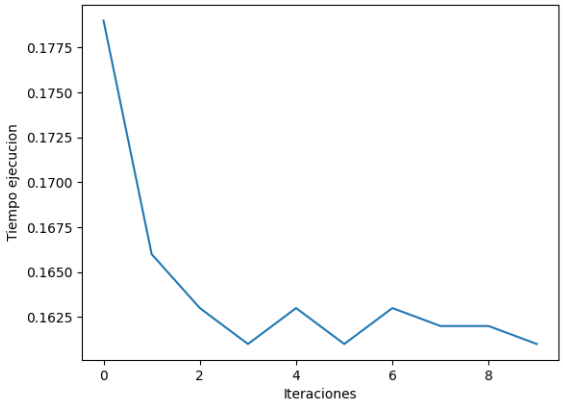


Figure 28. Gráfico de tiempo de iteraciones del experimento 6 con programación dinámica

```
Output: 777

(2,0)->(3,1)->(4,2)->(4,3)

Average Execution Time BF: 0.1485 ms
Average Execution Time DP: 0.1641 ms
```

## VI. Análisis de Resultados

### A. Contenedor

De acuerdo a los resultados y los tiempos promedios de cada método se desarrollaron las siguientes gráficas que harán más sencilla la visualización de los resultados, y la comparación de métodos.

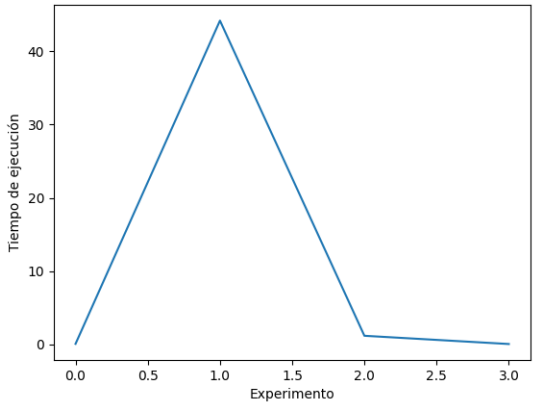


Figure 29. Gráfico de tiempo promedio con fuerza bruta

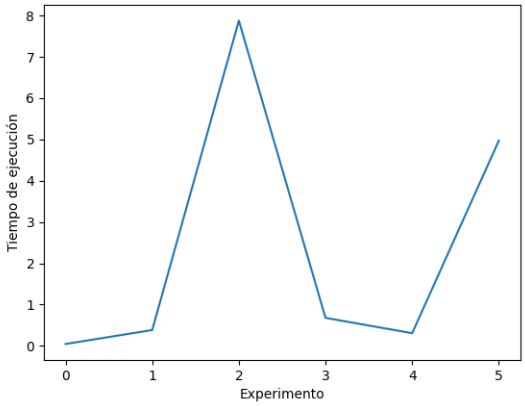


Figure 30. Gráfico de tiempo promedio con programación dinámica bottom-up

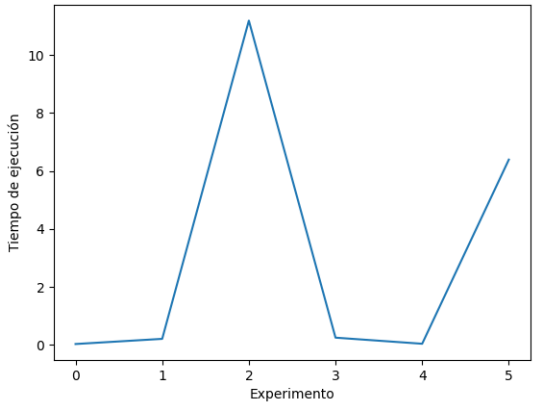


Figure 31. Gráfico de tiempo promedio con programación dinámica top-down

Primero hablemos de la complejidad de cada método, comenzando por fuerza bruta el cual para encontrar la

solución calcula cada posible solución, lo que ha su vez calcula los mismo subproblemas una y otra vez, resultando en una complejidad exponencial $O(2^n)$, lo cual da como resultado que en caso grande gaste recursos de manera exagerada, y esto lo podemos ver en los experimentos tres y seis, los cuales fueron creados con una cantidad considerable de elementos y que este método no logró resolver debido a que acabo con los recursos de la computadora.
Por otro lado, tenemos el método de programación dinámica bottom-up,el cual a diferencia del anterior no calcula cada subproblema una y otra vez, para evitar la perdida de recursos en recalcular datos de manera innecesaria se crea una matriz, la cual ayuda a encontrar soluciones sin consumir tantos recursos, la complejidad de esta es de $(O(N * W))$, donde 'N' es el número de elemento de peso y 'W' es la capacidad, gracias a estas diferencias podemos ver como los tiempos en comparación del anterior mejoran, y como el experimento tres y seis si pudieron ser resueltos.

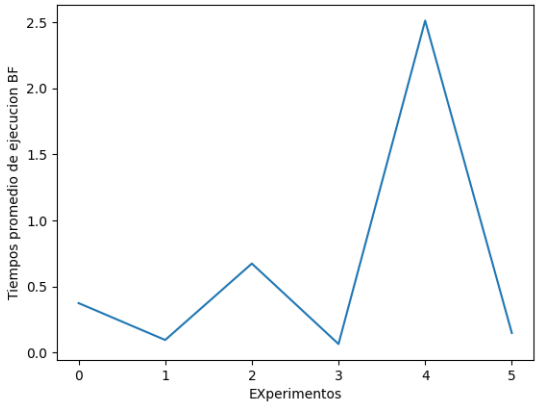


Figure 32. Gráfico de tiempo promedio de ejecución de los experimentos con fuerza bruta

Por ultimo, el método de programación dinámica top-down, este método usa la técnica de memorización, la cual consiste en crear una matriz 2D, donde se puede almacenar estados particulares del proceso, donde si más adelante se vuelve a repetir el estado no existe la necesidad de recalculando de nuevos, ya que se puede retornar directamente reduciendo la complejidad exponencial $(O(N))$, ya que únicamente necesita pasar una vez por cada estructura

Sin embargo, al observar las graficas de promedios podemos ver como los valores de bottom-up son ligeramente menores a los de top-down, esto dado principalmente debido a que en el experimento tres hubieron iteraciones donde los tiempo de ejecución se elevaron y esto sesgó el promedio de iteraciones y por ende la gráfica de promedios.

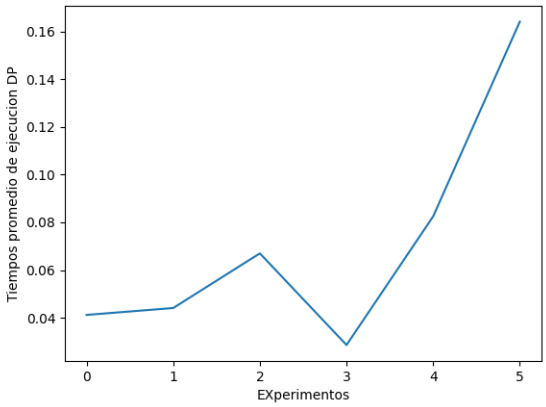


Figure 33. Gráfico de tiempo promedio de ejecución de los experimentos con programación dinámica

*B. Mina de Oro*

De acuerdo a las gráficas podemos observar que los tiempos promedio de los experimentos con programación dinámica se efectúan de manera mas óptima, con una complejidad de O(N*M), siendo N las filas y M las columnas. Mientras que el algoritmo de fuerza bruta tiene una complejidad mayor $O(3^n)$. El algoritmo de programación dinámica logra resolver minas de 200x200 en un promedio de 139.8156 ms, 10 iteraciones, con fuerza bruta excedió 5 minutos, se abortó el experimento.

## VII. Conclusión

Gracias a la ejecución de estos experimentos fue posible ver de una manera más clara ver la importancia de la optimización de códigos con el fin de ahorrar recursos y que estos puedan ser utilizados en otras tareas del programa. Así mismo, la labor tan vital de la programación dinámica en el desarrollo de programas más eficientes y que necesitan menor cantidad de recursos.

La programación dinámica es una gran herramienta para optimizar los problemas de gran complejidad temporal y espacial, es similar a la fuerza bruta porque recuerda los estados previos, la programación dinámica top down tiene ventajas sobre bottom up, en algunas ocasiones, ya que parte de un estado o problema más grande hasta llegar a los más específicos o pequeños.

## VIII. Referencias


Cormen, T.H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009) *Introduction to Algorithms (3rd ed.).* London, England: MIT Press.

Halim, S., Halim, F., Effendy, S. (2020). *Competitive programming 4: the lower bound of programming contests in the 2020s (4th ed.).* Morrisville.