Proyecto 2: Desempeño de la Programación Dinámica

E. Mena[1]

Abstract

El siguiente reporte se encuentra realizado en base al segundo proyecto del curso de Investigación de Operaciones del Tecnológico de Costa Rica, el cual consiste en el desarrollo de los algoritmos de alineamiento de secuencias y contenedores/mochila tanto su programación de manera dinámica como su realización con fuerza bruta. A continuación, veremos los resultados y el análisis de distintos experimentos realizados para comprobar la ejecución de los algoritmos desarrollados por el 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 el constaste de programación dinámica con el de fuerza bruta pero no sin dejar de lado la complejidad que resulta llegar a pensar el cómo resolver las dichos problemas.

I. Problema del contenedor

Especificación

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

Diseño de los experimentos

Para el diseño de los siguientes experimentos se utilizó el generador de experimentos realizado como parte del proyecto, de los cuales se crearon 6 para es este específico caso, estos están en el repositorio del proyecto, a continuación tenemos los experimentos:

[1]E. Mena estudiante de Ingeniería en Computación Instituto Tecnológico de Costa Rica, Costa Rica

1. Parámetros del experimento:

- nombre del archivo = experimentoC1
- W = 50
- N = 3
- minPeso = 5
- maxPeso = 15
- minBeneficio = 20
- maxBeneficio = 60
- minCantidad = 1
- maxCantidad = 5

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 1 experimentoC1 50 3 5 15 20 60 1 5
```

[11pt,draft,peerreview]

2. Parámetros del experimento:

- nombre del archivo = experimentoC2
- W = 25
- N = 10
- minPeso = 1
- maxPeso = 10
- minBeneficio = 5
- maxBeneficio = 20
- minCantidad = 1
- maxCantidad = 5

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 1 experimentoC2 25 10 1 10 5 20 1 5
```

3. Parámetros del experimento:

- nombre del archivo = experimentoC3
- W = 60
- N = 3
- minPeso = 10

- maxPeso = 55
- minBeneficio = 55
- maxBeneficio = 70
- minCantidad = 7
- maxCantidad = 15

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 1 experimentoC3 60 3 10 55 55 70 7 15
```

4. Parámetros del experimento:

- nombre del archivo = experimentoC4
- W = 96
- N = 20
- minPeso = 34
- maxPeso = 80
- minBeneficio = 31
- maxBeneficio = 67
- minCantidad = 3
- maxCantidad = 8

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 1 experimentoC4 96 20 34 80 31 67 3 8
```

5. Parámetros del experimento:

- nombre del archivo = experimentoC5
- W = 195
- N = 17
- minPeso = 15
- maxPeso = 52
- minBeneficio = 34
- maxBeneficio = 89
- minCantidad = 1
- maxCantidad = 14

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 1 experimentoC5 195 17 15 52 34 89 1 14
```

6. Parámetros del experimento:

- nombre del archivo = experimentoC6
- W = 250
- N = 25
- minPeso = 37
- maxPeso = 75
- minBeneficio = 67
- maxBeneficio = 211
- minCantidad = 10
- maxCantidad = 30

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 1 experimentoC6 250 25 37 75 67 211 10 30
```

Resultados de los experimentos

Todos los experimentos realizados están en orden en base a la sección anterior, además, todos los tiempos de ejecución son en segundos y milisegundos.

1. Resultado:

```
KnapSack with dynamic programing
186
1, 3
3, 1
Time execution: 0.0

KnapSack with backtracking
186
1, 3
3, 1
Time execution: 0.0
```

2. Resultado:

```
KnapSack with dynamic programing
93
2, 3
3, 1
7, 3
9, 1
10, 2
Time execution: 0.00099945068359375

KnapSack with backtracking
93
2, 3
3, 1
7, 3
9, 1
10, 2
Time execution: 9.158337831497192
```

3. Resultado:

```
KnapSack with dynamic programing
290
3, 5
Time execution: 0.0010006427764892578

KnapSack with backtracking
290
3, 5
Time execution: 0.10309314727783203
```

4. Resultado:

```
KnapSack with dynamic programing
130
12, 2
```

```
Time execution: 0.004002809524536133

KnapSack with backtracking
130
12, 2
Time execution: 0.07606911659240723
```

5. Resultado:

```
KnapSack with dynamic programing
777
3, 3
7, 9
Time execution: 0.01500391960144043

KnapSack with backtracking
777
3, 3
7, 9
Time execution: 3450.19587170802
```

6. Resultado:

```
KnapSack with dynamic programing
1203
5, 5
23, 1
Time execution: 0.06505894660949707

El algoritmo por FB no fue capaz de terminar.
```

Gráficos

En base a los experimentos se pueden ver los siguientes gráficos, donde el valor ’9999’ representa infinito ya que el algoritmo en cuestión fue capaz de terminar dicho caso.

Fig. 1. Gráfico de tiempos de los experimentos para el problema de contenedores con PD

Fig. 2. Gráfico de tiempos de los experimentos para el problema de contenedores con FB

II. Problema del alineamiento de secuencias

Especificación

Para este problema tenemos que se debe desarrollar el problema de alineamiento global de secuencias de Saul Needleman y Christian Wunsch, el cual sirve para alinear dos o más cadenas de secuencias que por lo general en bioinformática es de relevancia la hora de comparar proteínas o de ácidos nucleicos. Para la valorización entre dos cadenas se utilizará la siguiente escala: en caso de que haya un "match" (+1), en caso de que haya un "mismatch" entre dos letras (-1), en caso de que haya un GAP en cualquiera de las dos hileras (-2).

Diseño de los experimentos

Para el diseño de los siguientes experimentos se utilizó el generador de experimentos realizado como parte del proyecto, de los cuales se crearon 6 para es este específico caso, estos están en el repositorio del proyecto, a continuación tenemos los experimentos:

1. Parámetros del experimento:

- nombre del archivo = experimentoS1
- largo hilera 1 = 5
- largo hilera 2 = 2

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 2 experimentoS1 5 2
```

2. Parámetros del experimento:

- nombre del archivo = experimentoS2
- largo hilera 1 = 4
- largo hilera 2 = 4

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 2 experimentoS2 4 4
```

3. Parámetros del experimento:

- nombre del archivo = experimento3
- largo hilera 1 = 3
- largo hilera 2 = 6

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 2 experimentoS3 3 6
```

4. Parámetros del experimento:

- nombre del archivo = experimentoS4
- largo hilera 1 = 10
- largo hilera 2 = 8

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 2 experimentoS4 10 8
```

5. Parámetros del experimento:

- nombre del archivo = experimentoS5
- largo hilera 1 = 7
- largo hilera 2 = 4

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 2 experimentoS5 7 4
```

6. Parámetros del experimento:

- nombre del archivo = experimentoS6
- largo hilera 1 = 8
- largo hilera 2 = 9

Dando como resultado el siguiente comando por consola:

```
python .\generador.py 2 experimentoS6 8 9
```

Resultados de los experimentos

Todos los experimentos realizados están en orden en base a la sección anterior, además, todos los tiempos de ejecución son en segundos y milisegundos.

1. Resultado:

```
Sequence alignment with dynamic programing
Final score: -6
Hilera1: AATTG
Hilera2: ___CG
Time execution: 0.0
```

```
Sequence alignment with backtracking
Final score: -6
Hilera 1: AATTG
Hilera 2: ___CG
Time execution: 6.53303861618042
```

2. Resultado:

```
Sequence alignment with dynamic programing
Final score: 2
Hilera1: CAGT
Hilera2: CATT
Time execution: 0.0

Sequence alignment with backtracking
Final score: 2
Hilera 1: CAGT
Hilera 2: CATT
Time execution: 1.0596039295196533
```

3. Resultado:

```
Sequence alignment with dynamic programing
Final score: -7
Hilera1: ___AGC
Hilera2: TTTACA
Time execution: 0.0

Sequence alignment with backtracking
Final score: -7
Hilera 1: ___AGC
Hilera 2: TTTACA
Time execution: 1812.0948112010956
```

4. Resultado:

```
Sequence alignment with dynamic programing
Final score: 0
Hilera1: TGACCTCAGT
Hilera2: T_AGTTC_GT
Time execution: 0.001001596450805664

El algoritmo por FB no fue capaz de terminar.
```

5. Resultado:

```
Sequence alignment with dynamic programing
Final score: -6
Hilera1: CGGTAAC
Hilera2: __AT_TC
Time execution: 0.0

Sequence alignment with backtracking
Final score: -6
Hilera1: CGGTAAC
Hilera2: __AT_TC
Time execution: 20520.02176
```

6. Resultado:

```
Sequence alignment with dynamic programing
Hilera1: _TCGGAGGA
Hilera2: ACCCTTTCA
Time execution: 0.0

El algoritmo por FB no fue capaz de terminar.
```

Gráficos

En base a los experimentos se pueden ver los siguientes gráficos, donde el valor '9999' representa infinito ya que el algoritmo en cuestión fue capaz de terminar dicho caso.

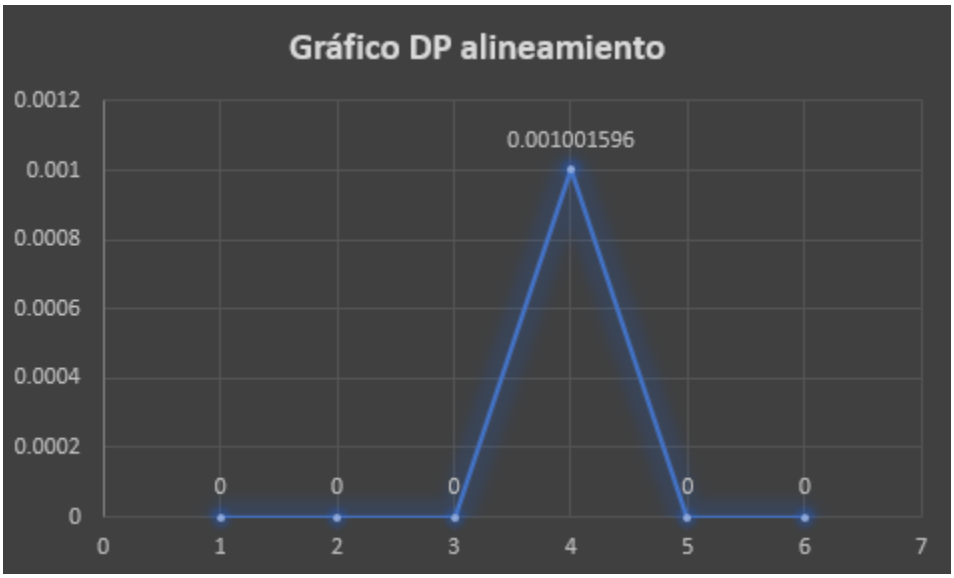

Fig. 3. Gráfico de tiempos de los experimentos para el problema de alineamiento con PD

Fig. 4. Gráfico de tiempos de los experimentos para el problema de alineamiento con FB

III. Análisis

En esta sección se analizarán los tiempos de los experimentos las y complejidades de los problemas realizados.

Contenedores

1. Programación dinámica:

Dada la forma en que se resolvió este problema la complejidad es de O(N*W), dado a que la función que realiza esta parte posee dos ciclos anidados donde N es la cantidad de elementos a evaluar y W el peso máximo permitido por el contenedor.

Para 5 de los 6 experimentos realizados se puede observar que el tiempo de resolución no fue de cero segundos y cero milésimas de segundos sino por lo que las operaciones para este tipo de problema son más complejas de realizar.

2. Fuerza bruta:

Dada la forma en que se resolvió este problema la complejidad es de O(2^N), dado a que la función que realiza esta parte si hay N cantidad de artículos por escoger hay 2^N posibles combinaciones entre ellos en el contenedor, esto se puede ver claro porque se puede generar un árbol binario con cada ejecución, dando como resultado final un árbol de N niveles.

Si bien al ser fuerza bruta se realizan todos y cada uno de los casos podemos deducir que los tiempos serán mayores y en base a su respectivo gráfico lo demuestra, donde podemos observar que ya no toma milésimas de segundos sino tanto segundos como minutos y casi que alcanzado la hora.
Aunque los tiempos estén en segundos podemos observar que uno tomó 57 minutos para dar el resultado y uno ni siquiera fue capaz de terminar pese a pasar más de 3 horas corriendo.

Alineamiento de secuencias

1. Programación dinámica:

Dada la forma en que se resolvió este problema la complejidad es de O(N + M + N*M), pudiéndose simplificar a O(N*M) ya que que las sumas son insignificantes, dado a que la función que realiza esta parte posee dos ciclos anidados donde N es el largo de la hilera 1 y M el largo de la hilera 2. Esto sin contar el costo que representa el determinar las secuencias el cual es de O(N).

Para 1 de los 6 experimentos realizados se puede observar que el tiempo de resolución no fue de cero segundos y cero milésimas de segundos sino que dada la complejidad del experimento 3 resultó ser mayor que el resto para resolverse de esta manera.

2. Fuerza bruta:

Dada la forma en que se resolvió este problema la complejidad es de O(N! * M!), dado a que la función que realiza esta parte se realizan N permutaciones donde N es la hilera 1 y M permutaciones donde M es la hilera 2, para al final comparar todo con todo con ciclos anidados.

Esta solución es peor que la realizada por fuerza bruta de contenedores en tiempos de complejidad y complejidad espacial, en los experimentos donde hubiera una hilera con 6 o más letras y lo podemos observar en el experimento 3 al ver que el tiempo fue de 30 minutos y al aumentar una letra en el experimento 5 llegó a tardar 5.7 horas. Por medio de fuerza bruta no se pudo resolver el experimento 4 y 6 que poseían 10 y 9 letras respectivamente, y aunque el problema de la especificación de 10 letras para ambas hileras con DP no era ni un segundo resolverlo con FB no hubiera sido capaz de terminar en tal vez días, semanas o incluso meses.

IV. Conclusiones

Dado que los problemas son muy populares para la programación dinámica no es la primera que se trabaja con ellos porque en cursos como Análisis de algoritmos o Lenguajes de programación se llegan a trabajar por lo que su implementación para este proyecto no resultó ser complicada, para la versión óptima de los mismos, dando como resultado un repaso de los algoritmos más que un aprendizaje. Si bien la versión óptima fue sencilla de implementar no terminó siendo el caso para las versiones de fuerza bruta debido tanto al par de modificaciones que realizó el profesor al problema de contenedores donde podían haber N cantidad de unidades de un mismo artículo y también que el problema de alineamiento no hay mucha información de cómo se podría resolver con fuerza bruta de una manera óptima sin tener problemas con las permutaciones de las cadenas que incrementaban sin lugar a a duda los tiempos de ejecución.

En base al análisis, y al sentido común, podemos decir que la programación dinámica es la mejor manera de resolver un problema en donde a primera vista parezca que tenemos que probar todos y cada uno de los casos que se nos presenten en el problema, donde llegamos a obtener menores tiempos de ejecución con problemas más complejos.