PROYECTO 2: DESEMPEÑO DE LA PROGRAMACIÓN DINÁMICA

Alejandro Salas Rojas
Ingeniería en Computación
Alajuela, Costa Rica
Carnet: 2014010742

Jose Daniel Rodríguez Huertas
Ingeniería en Computación
Alajuela, Costa Rica
Carnet: 2015108864

Erick Alfaro
Ingeniería en Computación
Alajuela, Costa Rica
Carnet: 2015097870

I. INTRODUCCIÓN

En el presente trabajo se deben resolver dos problemas con una versión en fuerza bruta y programación dinámica para cada problema. También se necesita programar un generador para crear los experimentos, donde se pueda incluir los parámetros y genere de manera fácil los datos para hacer las corridas. Debe ejecutar cada experimento una cantidad razonable de iteraciones para comparar el tiempo de ejecución de fuerza bruta con programación dinámica.
El problema 1 es La Mina de Oro, la cual tiene 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 (camino) seleccionadas.
El problema 2 es Problema del contenedor, se tienen n elementos distintos, y un contenedor que soporta una cantidad específica de peso W. Cada elemento i tiene un peso wi, un valor o beneficio asociado dado por bi, y un 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 qué soporta. La solución debe ser dada con la lista de los elementos que se ingresaron y el valor del beneficio máximo obtenido.
Se van a comparar los resultados para ver qué tanto varía el tiempo de resolución para cada problema.

II. MINA DE ORO

II-A. Especificación del problema

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 (camino) seleccionadas.
El programa debe tener dos modos de ejecución, uno donde reciba un archivo con los datos del experimento, y otro en donde se pueda generar un experimento con parámetros.

II-B. Parametros - Datos relevantes

Para la ejecución de este problema se cuenta con mina.py, el cual es el encargado de dar solución a dicho problema.
Para resolver un problema con Programacion Dinamica:

```
$ python3 mina.py -pd archivo.txt
```

Para resolver un problema con Fuerza Bruta:

```
$ python3 mina.py -fb mina.txt
```

Para generar un nuevo experimento:

```
$ python3 mina.py -g gen -n # -m # -om # -i #
```

Donde:

- -n y -m representan la dimensión de la matriz (nxm).
- -i número de iteraciones.
- -om oro máximo por casillas.
- -g palabra para identificar que el usuario quiere generar un experimento.
- -pd para resolver un ejemplo con Programación Dinámica.
- -fb para resolver un ejemplo con Fuerza Bruta.

II-C. Diseño del experimento

El generador de experimentos para el problema de la mina de oro consiste en lo siguiente:

- El usuario ingresa la cantidad de filas y columnas con las cuales se desea crear la matriz.
- El usuario ingresa la cantidad de veces que quiere repetir ese experimento (número de iteraciones)
- El generador se encarga de generar una matriz con números random de 1 al 100 de acuerdo al tamaño de dicha matriz
- Soluciona el problema n cantidad de veces utilizando fuerza bruta y devuelve el tiempo promedio y resultado.
- Soluciona el problema n cantidad de veces utilizando programación dinámica y devuelve el tiempo promedio y resultado.
- Escribe en un archivo llamado experimentoMinaOro.txt los parámetros y resultados obtenidos.

Para encontrar la solución usando programación dinámica se utiliza una matriz de cantidad o pesos que va ser del mismo

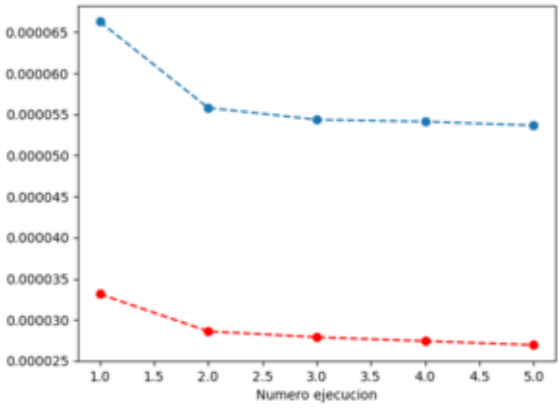
tamaño que la matriz original originada. Esta matriz auxiliar o memoize nos va ayudar para ir guardando los caminos y los pesos máximos acumulados hasta dicho camino. Esto nos ayudara en gran medida, ya que no tendremos que calcular todas las rutas o posibilidades, debido a que esta matriz auxiliar nos ayudara a verificar si un camino en específico ya se calculó o recorrió, de serlo así, se devuelve el resultado que se guardo para ese camino y de lo contrario se procede a calcular los 3 caminos posibles (derecha, derecha arriba y derecha abajo), se saca el máximo y se guarda en la matriz. La solución por fuerza bruta es básicamente igual a la solución dinámica, con la única diferencia que se quita la matriz auxiliar de pesos/cantidades (memoize), por lo que obligatoriamente se le obliga al programa a calcular todos los caminos posibles las veces que sea necesario, lo que hace que este método sea considerablemente menos eficiente.

II-D. Resultados de los experimentos y gráficos

Todos los datos estan tomados en segundos, Azul representa Fuerza Bruta y Rojo Programacón Dinámica.
Ejercicios de prueba (Especificación del proyecto):

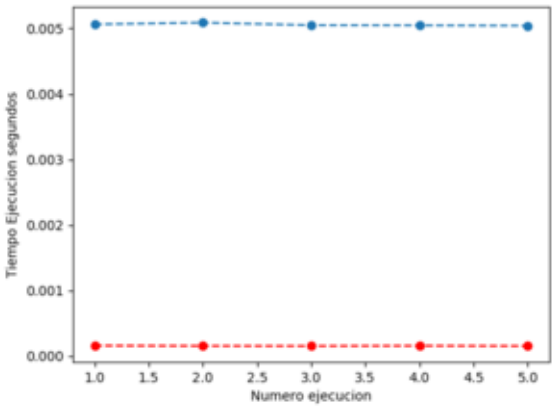
Ejercicios	PD	FB
mina.txt	6,20E-05	8,32E-05
mina1.txt	8,58E-05	2,68E-04
mina2.txt	8,73E-05	2,61E-04

Casos de prueba generando experimentos:
1) $ python3 mina.py -g gen -n 3 -m 3 -om 20 -i 5



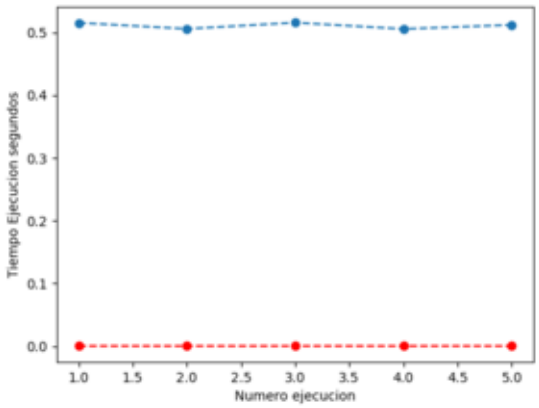
Iteración	PD	FB
1	3,31E-05	6,63E-05
2	2,86E-05	5,58E-05
3	2,79E-05	5,44E-05
4	2,74E-05	5,41E-05
5	2,69E-05	5,36E-05

2) $ python3 mina.py -g gen -n 8 -m 6 -om 12 -i 5



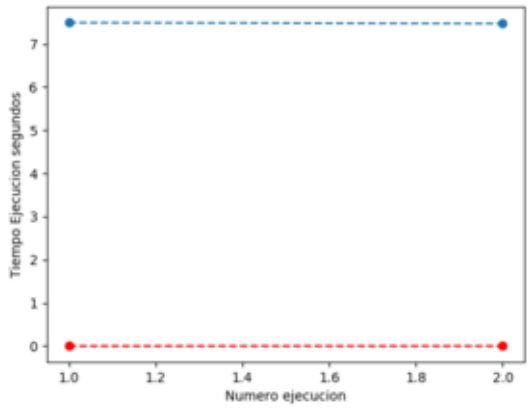
Iteración	PD	FB
1	0,0001690387726	0,005101203918
2	0,0001654624939	0,005086421967
3	0,0001645088196	0,005082130432
4	0,0001640319824	0,005071878433
5	0,0001645088196	0,005076885223

3) $ python3 mina.py -g gen -n 10 -m 10 -om 20 -i 5



Iteración	PD	FB
1	0,0003476142883	0,5156466961
2	0,0003447532654	0,5059769154
3	0,0003387928009	0,5158975124
4	0,0003387928009	0,5057740211
5	0,0003390312195	0,5122916698

4) $ python3 mina.py -g gen -n 15 -m 12 -om 20 -i 2



Iteración	PD	FB
1	0,0006151199341	7,496441126
2	0,0006079673767	7,474016428

III. Problema del contenedor

III-A. Especificación del problema

Se tienen n elementos distintos, y un contenedor que soporta una cantidad específica de peso W. Cada elemento i tiene un peso wi, un valor o beneficio asociado dado por bi, y un 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.
Ejemplo: Con un contenedor de peso W = 50. 3 artículos con pesos: 5, 15, 10, beneficios: 20, 50, 60, y unidades 4, 3 y 3 respectivamente. Tiene como solución un beneficio máximo de 260 agregando los artículos 3 (3 unidades) y 1 (4 unidades).

III-B. Parametros - Datos relevantes

Para la ejecución de este problema se cuenta con mochila.py, el cual es el encargado de dar solución a dicho problema.
Para resolver un problema con Programacion Dinamica:

```
$ python3 mochila.py -pd mochila.txt
```

Para resolver un problema con Fuerza Bruta:

```
$ python3 mochila.py -fb mochila.txt
```

Para generar un nuevo experimento:

```
$ python3 mochila.py -g gen -w # -n # -i # -p # -b #
```

Donde:

- -w es el tamaño del contenedor
- -i número de iteraciones.
- -b beneficio máximo para un artículo
- -p peso máximo para un articulo
- -g palabra para identificar que el usuario quiere generar un experimento.
- -pd para resolver un ejemplo con Programación Dinámica.
- -fb para resolver un ejemplo con Fuerza Bruta.

III-C. Diseño del experimento

Primeramente, se lee el archivo de texto línea por línea, el paso siguiente fue realizar un Split (“,”) para separar cada uno de los elementos de tipo int. Al tener los elementos en una lista se procede a convertirlos a entero, pues al leerlos se muestran como char.
Una vez realizados lo pasos anteriores, se crean dos listas, una que incluye los pesos y otra los beneficios, tomando en cuenta la cantidad de cada ítem, es decir si se ingresa un elemento con 5,20,3; en la lista de pesos deberá de existir 3 números 5 y en la lista de beneficios 3 números 20.
Al tratarse de la función de programación dinámica, lo recibido son ambas listas y el peso máximo (línea 0 del txt); la base del algoritmo es incluir un memoize para evitar demorar mayor tiempo y una complejidad de alto grado, como lo sería el log(n).
Si fuese fuerza bruta, es requerido convertir las listas en una matriz, la cual debe seguir la siguiente estructura:

```
((n ítem,Peso,Valor),(n ítem,Peso,Valor))
```

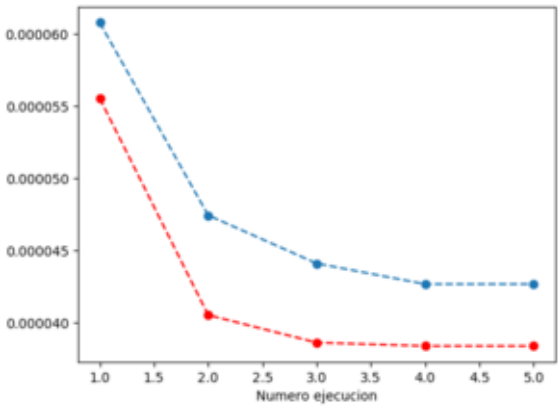
Fuerza bruta, lo que conlleva es la búsqueda de todas las posibles combinaciones, para luego encontrar la óptima, por esta razón tiende a ser menos eficiente.

III-D. Resultados de los experimentos y gráficos

Todos los datos estan tomados en segundos, Azul representa Fuerza Bruta y Rojo Programacón Dinámica.
Ejercicios de prueba (Especificación del proyecto):

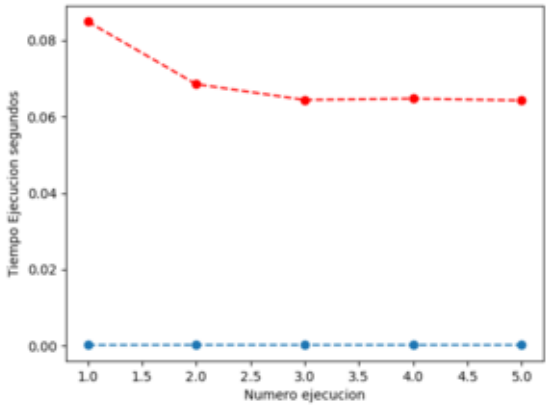
Ejercicios	PD	FB
mochila.txt	4,81E-04	5,20E-01

Casos de prueba generando experimentos:
1) $ python3 mochila.py -g gen -w 10 -n 3 -i 5 -p 8 -b 10



Iteración	PD	FB
1	7,49E-05	5,53E-05
2	4,79E-05	4,03E-05
3	4,43E-05	3,79E-05
4	4,32E-05	3,74E-05
5	4,29E-05	3,72E-05

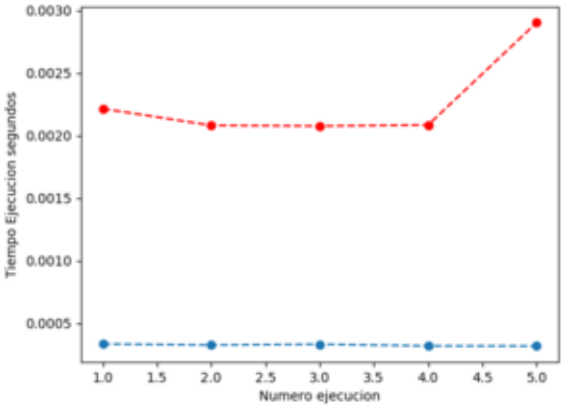
2) $ python3 mochila.py -g gen -w 30 -n 7 -i 5 -p 10 -b 10



Iteración	PD	FB
1	2,39E-04	8,49E-02
2	2,62E-04	6,85E-02
3	2,56E-04	6,44E-02
4	2,54E-04	6,47E-02
5	2,51E-04	6,42E-02

3) $ python3 mochila.py -g gen -w 50 -n 7 -i 5 -p 40 -b 70

Iteración	PD	FB
1	3,35E-04	2,21E-03
2	3,26E-04	2,08E-03
3	3,33E-04	2,08E-03
4	3,19E-04	2,09E-03
5	3,19E-04	2,90E-03



IV. Analisis de los resultados

- La técnica de programación dinámica permite convertir un problema grande que a veces es difícil de resolver, en problemas pequeñas y fáciles de solucionar.
- La programación dinámica es útil para problemas de asignación de recursos como el de la mochila y asignación de pesos máximos por caminos o recorridos como es el de la mina de oro
- Utilizar programación dinámica posibilita al programador generar un O(n) optimo, ahorrando tiempo de ejecución y empleando buenas prácticas de programación.
- Se dispone de una relación recursiva que identifica la política óptima para la etapa n, dada la política óptima para la etapa n+1 en la programación dinámica.
- Fuerza bruta al tener que buscar cada una de las rutas posibles, emplea mayor tiempo, ese tiempo depende de cómo se encuentren ubicados los números y de la complejidad de la entrada.
- La diferencia se ve más marcada en el tamaño del problema o de la matriz (mina de oro), cuando las entradas son grandes la diferencia se vuelve mayor entre Fuerza Bruta y Programación Dinámica.