Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Centro Académico de Alajuela
Investigación de Operaciones
I Semestre, 2022
Profesor: Carlos Gamboa Venegas

# Proyecto 2: Desempeño de Programación Dinámica

## Administrativos

- Entrega domingo 15 de mayo. Subiendo en el tec-digital el archivo .zip con el código del repositorio. Último commit antes de las 11:59 del día de la entrega.
- El proyecto deberá tener un repositorio en git, si no existe el repositorio cada persona tendrá una nota de 0.
- Se deben trabajar en grupos de 3 personas. Grupos que serán designados por el profesor.
- Nombre del repositorio: **io_grupo(número del grupo)_proyecto2**

## Especificación

### Implementación de algoritmos

- Se deben implementar dos versiones de cada problema, el primero es fuerza bruta, también llamada búsqueda exhaustiva, pero no se debe confundir con backtracking. El segundo es programación dinámica.
- En todos los algoritmos implementados se debe indicar por consola en tiempo de ejecución en segundos y milésimas de segundo.
- Se deberán programar en archivos .py separados por problema. Al final se tendrán tres archivos como mínimos.
- Los archivos .py deben llamarse como se indica en cada problema, no deben cambiar el nombre por ningún motivo.

### Problema 1: 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.

Se deben programar dos versiones del algoritmos, el primero es fuerza bruta, también llamada búsqueda exhaustiva, pero no se debe confundir con backtracking. El segundo es programación dinámica (implementación bottom-up visto en clase).

El algoritmo a correr debe ser indicado como parámetros en la linea de entrada del programa. Con dos parámetros, el tipo de algoritmo a correr y el archivo con los datos del problema.

Ejemplo de ejecución:
`./contenedor.py algoritmo archivo.txt`

- `algoritmo` requerido, valores 1=fuerza bruta, 2=programación dinámica
- `archivo.txt` archivo con los datos del problema

Estructura de un archivo de entrada:

*linea 1: Peso máximo de la mochila*
*linea 2: elemento i (peso, beneficio) separados por comas*
*linea n: elemento n (peso, beneficio)*

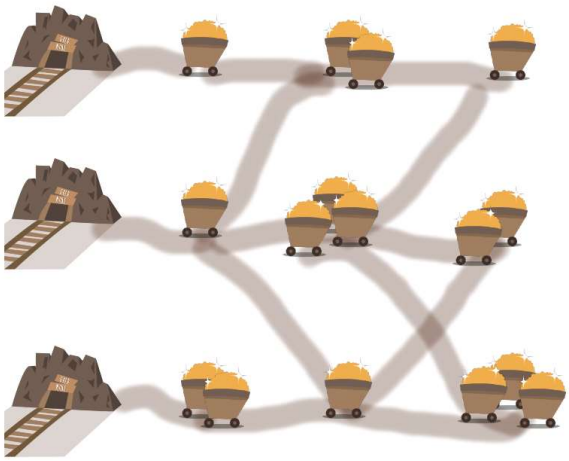
*Ejemplo:* Con un contenedor de peso W = 30. 5 artículos con pesos: 5, 15, 10, 10, 8, beneficios: 20, 50, 60, 62 y 40. Tiene como solución un beneficio máximo de 162 agregando los artículos 3, 4, y 5.

```
./contenedor.py 1 p1_mochila.txt
Input: (problema1.txt)
30
5,20
15,50
10,60
10,62
8,40

Output:
Beneficio máximo: 162
Incluidos: 3,4,5
Tiempo de ejecución: 10.5498 segundos
```

**Problema 2: 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 (camino) seleccionadas.



También debe ser programada con fuerza bruta y programación dinámica, para ser comparados. El método a correr debe ser indicado en los parámetros.

`./mina.py algoritmo archivo.txt`

- `algoritmo` parámetro requerido, valores 1=fuerza bruta, 2=programación dinámica
- `archivo.txt` archivo con datos del problema

Input: *mina1.txt*

```
1, 3, 3
2, 1, 4
0, 6, 4
Output : 12
```

Ejemplo de selección de casillas: `{(1,0)→(2,1)→(2,2)}`

```
Tiempo de ejecución: 5.268678221 segundos
```

Input: mina2.txt

```
1, 3, 1, 5
2, 2, 4, 1
5, 0, 2, 3
0, 6, 1, 2
Output : 16
```

Ejemplo de selección de casillas:

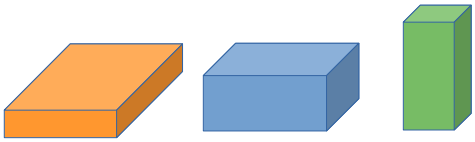
```
(2,0) -> (1,1) -> (1,2) -> (0,3) OR
(2,0) -> (3,1) -> (2,2) -> (2,3)
Tiempo de ejecución: 0,2255 segundos
```

Input : mina3.txt

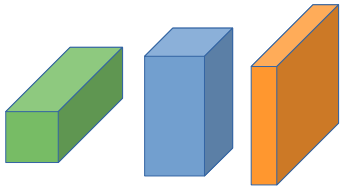
```
10, 33, 13, 15
22, 21, 04, 1
5, 0, 2, 3
0, 6, 14, 2
Output : 83
Ruta:
Tiempo de ejecución: 0,095123 segundos
```

**Problema 2: Torre de bloques**

Se tienen una cantidad bloques de madera tridimensionales, cada uno con medida de Largo (L), Ancho (W) y Altura (H). Se deben poner uno encima de otro de manera que se logre hacer la torre más alta posible. Sin embargo un bloque puede ir encima del otro si su base es estrictamente menor que superficie superior del bloque de abajo. Esto quiere decir que las dimensiones de Largo y Ancho (base) son **estrictamente menores** al bloque inferior.



Los bloques se pueden rotar en sus dimensiones, y se pueden usar más de un bloque según las rotaciones del mismo tipo para formar la torre. Cada bloque tiene 3 posibles rotaciones.



Debe implementar la solución con fuerza bruta y programación dinámica.

**Input**
En cada linea del archivo se indicara las dimensiones de un bloque, separados por coma en el siguiente orden: Largo (L), Ancho (W) y Altura (H)

Ejemplo
Input: *bloques1.txt*

```
2, 3, 3
2, 4, 4
1, 1, 4
4, 4, 2
Output: altura máxima 11
Bloques (2,3,3), (1,1,4), (4,2,4)
```

```
./bloques.py algoritmo bloques_p1.txt
```

- `algoritmo` parámetro requerido, valores 1=fuerza bruta, 2=programación dinámica
- `archivo.txt` archivo con datos del problema

## Evaluación

| | |
|---|---|
| Cada problema, resultados correctos y tiempo de ejecución | 30% |
| Escritura de código propio, documentación | 10% |

Aclaración: si se detecta y confirma la no participación de una persona en el proyecto, esta persona tendrá un 0 en la nota. Esta participación será verificada en los aportes al repositorio git, y con los compañeros de ser necesario. Además con el formulario de evaluación cruzada que se aplicará al final del proyecto.