UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
Escuela de Ingeniería de Sistemas y Computación
Curso: Análisis de Algoritmos II
Docentes: Juan Francisco Díaz Frias, Jesús Alexander Aranda
Integrantes: Johan Sebastían Tombe Campo & César Alejandro Grijalba Zúñiga

---

Proyecto Uno - Subasta pública de acciones

2.3. ¿Entendimos el problema?

Teniendo en cuenta la siguiente entrada:

A=1000, B=100, n=2
[(500, 600, 100), (450, 800, 400), (100, 1000, 0)]

En la siguiente tabla se pueden observar las posibles asignaciones:

# Asignación	Asignación	Valor recibido
1	[600,400,0]	480.000
2	[500,400,100]	440.000
3	[100,800,100]	420.000
4	[100,400,500]	280.000
5	[200,600,200]	390.000

- La asignación de acciones donde vr(X) es máximo es la asignación número 1, la cuál ofrece como beneficio 480.000

3. El proyecto

3.1. Usando fuerza bruta

3.1.1. Entendimos el algoritmo

Teniendo en cuenta la siguiente entrada:

A=1000, B=100, n=4
[(500, 600, 400), (450, 400, 100), (400, 400, 100),(200, 200, 50), (100, 1000, 0)]

En la siguiente tabla se pueden observar las posibles asignaciones:

# Asignación	Asignación	Valor recibido
1	[600,400,0,0,0]	480000
2	[600,0,400,0,0]	460000
3	[0,400,400,200,0]	380000
4	[0,0,0,0,1000]	100000

3.1.2. Complejidad

La complejidad del algoritmo fuerza bruta, en el peor de los casos corresponde a O(n log n) complejidad lineal rítmica, ya que en un punto se usa la función sorted() de python con un parámetro key, este parámetro no afecta la complejidad porque se especifica una función lambda que toma un elemento de la lista oferentes y devuelve el mismo, debido a que la función lambda tiene una complejidad constante la complejidad no se verá afectada.

En el mejor de los casos y caso promedio la complejidad será de O(n) y esto se debe a que la lista de oferentes puede venir totalmente ordenada o casi ordenada.

3.1.3. Corrección

Teniendo en cuenta que el algoritmo de fuerza bruta organiza las ofertas de manera lexicográfica, dará prioridad a la oferta que tenga mayor precio y no le importará la cantidad que esa oferta pretenda comprar, por eso es posible que en alguna oferta donde puede que el precio esté por debajo del máximo por muy poquito, pero que tenga capacidad de comprar más acciones que el máximo, entonces se perdería una muy buen opción, por ejemplo, tenemos la siguiente entrada:

A=150, B=100, n=2

[(500, 100, 50), (450, 150, 50), (100, 150, 0)]

Nuestro algoritmo de fuerza bruta según sus parámetros, nos deberá entregar la siguiente asignación: [100,0,50] con un beneficio de 55.000, pero si observamos bien el segundo mejor oferente ofrece un precio considerable con muchas más acciones, es decir, que si se hubiera generado la asignación de la manera [0,150,0] se hubiera obtenido un beneficio de 67.500, pero al este haber ofrecido un precio un poco más bajo se descartó por parte de nuestro algoritmo.

En ese orden de ideas, el algoritmo no nos dará siempre la mejor respuesta al problema.

3.2. Usando un algoritmo voraz

3.2.1. Describiendo el algoritmo

Para el algoritmo de programación voraz se inicia organizando las entradas de manera lexicográfica, luego se examina cada oferente (p,c,r) con el fin de comprobar que sus condiciones de compra concuerden con las condiciones y disponibilidades de venta, es decir, que por cada oferente se verifica si su ***r*** es igual o menor a nuestras acciones disponibles y luego se verifica si le podemos vender todas las acciones que el oferente nos determina en ***c*** o si nuestras acciones disponibles se encuentran en el intervalo [c, r]. Una vez finalizada la comprobación por cada oferente, verificamos si contamos con acciones disponibles aún y dado el caso en que sí, estas se deberán vender al gobierno.

3.2.2. Entendimos el algoritmo

Tenemos la entrada:

A=1000, B=100, n=4
[(500, 600, 400), (450, 400, 100), (400, 400, 100),(200, 200, 50), (100, 1000, 0)]

El valor recibido de parte de nuestro algoritmo a la anterior entrada es de: [600,400,0,0,0] y un beneficio de 480.000.
En esta entrada nuestro algoritmo sí nos entrega la solución más óptima.

En la siguiente tabla se podrán observar 4 nuevas entradas que se probaron con nuestro algoritmo de programación voraz:

N°	ENTRADA	SOLUCIÓN ALGORITMO	SOLUCIÓN ÓPTIMA
1	A=1000, B=100, n=2 [(600, 550, 400), (450, 600, 500), (100, 1000, 0)]	375.000	510.000
2	A=1000, B=100, n=3	281.000	281.000

	[(300, 100, 0), (280, 800, 100), (270, 1000, 100), (100, 1000, 0)]		
3	A=1000, B=100, n=4 [(600, 400, 200), (300, 300, 100), (290, 1000, 400), (200, 300, 100), (100, 1000, 0)]	390.000	414.000
4	A=1000, B=100, n=5 [(600, 550, 400), (450, 300, 100), (400, 300, 200), (400, 300, 100), (200, 400, 100), (100, 1000, 0)]	525.000	525.000

3.2.3. Complejidad

La complejidad del algoritmo con programación voraz, corresponde a una complejidad O(n log n) en el peor de los casos, ya que se usa la función sorted() de python con un parámetro key, este parámetro no afecta la complejidad porque se especifica una función lambda que toma un elemento de la lista oferentes y devuelve el mismo, debido a que la función lambda tiene una complejidad constante la complejidad no se verá afectada.

En el mejor de los casos y caso promedio la complejidad será de O(n) y esto se debe a que la lista de oferentes puede venir totalmente ordenada o casi ordenada.

3.2.4. Corrección

Nuestro algoritmo no siempre nos da la respuesta más óptima pero si es posible una muy buena respuesta. Los casos en los que sí logra darnos la respuesta más acertada, es decir, con mayor beneficio, es en los casos donde nuestras entradas están muy bien organizadas y las ofertas cuentan con un mínimo de acciones por comprar muy bajo, puesto que cuando el mínimo de acciones que pretende comprar el oferente es bajo, nuestro algoritmo logra que las acciones se distribuyan muy bien entre las mejores ofertas.

En el mejor de los casos para obtener la respuesta más óptima con nuestro algoritmo voraz, sería si en nuestra entrada todos los oferentes tuvieran como 0 la cantidad mínima de compra, de esta manera el algoritmo venderá las acciones en orden de mejores precios ofertados.

3.3. Usando programación dinámica

No se logró hacer el algoritmo adecuado.