Elección de mejores inversiónes haciendo uso de

fuerza bruta.

Gustavo Hurtado A.

e-mail: Gustavo.hurtado@usach.cl

[Github](https://github.com/ghurtadoarevalo/lab1_avanzados)

*<Selecting best invertions using brute force>.*

**RESUMEN:** *En el siguiente documento se muestra el cómo se resolvió uno de los problemas algorítmicos más reconocido a nivel mundial, el llamado “Knapsack 0/1 problem” haciendo uso de una estrategia de resolución de problemas, que, si bien es ineficiente, nos garantiza el que siempre tendremos la mejor solución, es decir, fuerza bruta.*

**PALABRAS CLAVE**: “fuerza bruta”, “mochila”, “knapsack”, “algoritmo”, ”resolución”, ”problema”.

# INTRODUCCIÓN

Como antes se mencionó, lo que será expuesto en este documento, es el famoso problema de la mochila y una manera de resolverlo, pero aplicado con un enfoque diferente. En vez de considerar objetos que guardar en una mochila que contiene un peso máximo de objetos, se realizarán inversiones que tengan cierto costo y beneficio, eligiendo las inversiones que entreguen un mayor beneficio sin superar un presupuesto dado.

La importancia de este problema radica en su versatilidad y la gran cantidad de usos que se le puede dar, siendo las inversiones, un muy buen ejemplo de esto.

Lo que se desea lograr, es desarrollador un programa en lenguaje C, capaz de entregar la mejor solución para una cantidad de inversiones entregadas por el usuario, haciendo uso de el método para resolución de problemas llamado fuerza bruta.

# DESCRIPCIÓN DEL PROBLEMA

Una empresa inversora requiere determinar qué inversiones son las que producirán un menor costo, y a la vez, generarán un mayor beneficio para ellos, considerando el hecho de que estas inversiones son vitales para su crecimiento.

Es de suma importancia mencionar que la empresa inversora tiene un presupuesto limitado con el cual puede invertir, el que, por razones obvias, no puede superar.

Un ejemplo del problema:

Presupuesto: 50000

Inversión 1 = Costo: 2000, Beneficio: 200

Inversión 2 = Costo: 10000, Beneficio: 700

Inversión 3 = Costo: 5000, Beneficio: 450

Inversión 4 = Costo: 40000, Beneficio: 1000

A simple vista y con una cantidad de inversiones pequeñas, es posible encontrar la mejor solución, pero este problema puede llegar a escalar rápidamente al tener 10 o más inversiones entre las cuales elegir, haciendo prácticamente imposible el proceso de selección manual.

# MARCO TEÓRICO

Para dar a entender el cómo se resolverá este problema, es importante recalcar que no es algo nuevo, y que se viene estudiando hace bastantes años. El problema de la mochila o knapbag problem ha sido resuelto de múltiples maneras, pero en este documento, se realizará con el método de fuerza bruta.

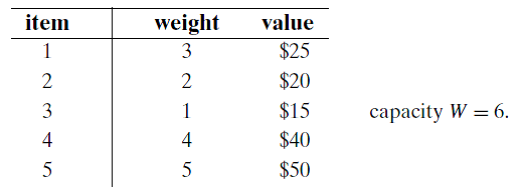


Ilustración 1. Ejemplo de problema de la mochila.

Fuerza bruta es la manera más simple de pensar e implementar una solución, asegurando el que siempre se tendrá la mejor opción según el criterio que sea necesario. Esto se realiza generando absolutamente todas las opciones posibles, considerando incluso aquellas que superan las restricciones del problema inicial, para posteriormente, ir descartando las opciones que son innecesarias o que no cumplen con las restricciones entregadas por el problema en cuestión. Luego simplemente se elige de entre todos los candidatos restantes, la opción o las opciones que entreguen mayor beneficio, que puede ser tanto el menor costo, menor peso, mayor ganancia, etc.

Esta forma de resolución de problemas es bastante simple de implementar, pero su punto débil es su eficiencia. Al generar todas las posibilidades posibles, la cantidad de opciones pueden ser factorial, o en el peor de los casos exponencial.

# DESCRIPCIÓN DE LA SOLUCIÓN

Haciendo uso de lenguaje C, la solución se realizó considerando un archivo de entrada “Entrada.in” que contiene el presupuesto, la cantidad de inversiones, y las respectivas inversiones con sus costos y beneficios.

Se realizó una función que genera todas las combinaciones posibles y a la vez, calcula sus gastos y beneficios, para posteriormente hacer elección de la que otorgue mayor beneficio, sin considerar las que superen el presupuesto establecido.

Teniendo la mejor combinación de inversiones, esta se almacena en el archivo “Salida.out”, mostrando su costo total, beneficio total y las inversiones.

## IDEA

Generar todas las combinaciones posibles de las inversiones disponibles que puede realizar la empresa y por cada combinación, ir calculador de inmediato el costo total y el beneficio total, para así tener un registro de ello. Posteriormente, descartando las opciones que sobrepasan el presupuesto del que la empresa dispone, se entrega la combinación de inversiones que otorguen un mayor beneficio.

*Mayor explicación apartado 8.*

## REPRESENTACIÓN Y ESTRUCTURA

Para hacer más fácil el cómo se maneja la información, se realizaron 2 estructuras:

1. Investment (Inversión): Esta estructura modela una inversión, con los atributos costo y beneficio.

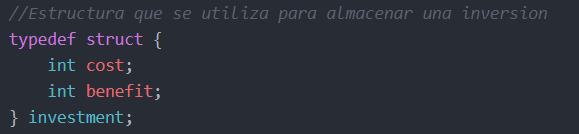


Ilustración 2. Estructura Inversión.

1. Investments (Inversiones): Esta estructura modela una combinación de inversiones, la cantidad de inversiones en la combinación, el costo total de esas inversiones y el beneficio total que otorgan.

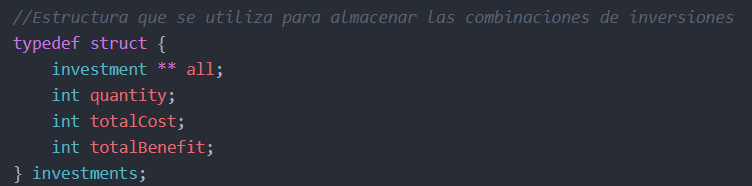


Ilustración 3. Estructura Combinación Inversiones.

Posteriormente se hizo uso de una gran lista de “investments”, para mantener todas las combinaciones.

## PSEUDOCÓDIGO

*Pseudocódigo en apartado 8.*

## TRAZA

*Traza en apartado 8.*

## ORDEN DE COMPLEJIDAD

Haciendo un análisis por función se tiene:

* *readData*: , siendo m la cantidad de caracteres del documento Entrada.in
* *powerSet*: , siendo n la cantidad de inversiones que hay en el documento Entrada.in.
* *bruteForce*: , siendo n la cantidad de inversiones que hay en el documento Entrada.in.
* *getOutput*: , siendo p la cantidad de inversiones en la combinación que genera mayor beneficio.

Considerando esas 4 funciones como las de mayor orden de complejidad, se puede llegar al siguiente orden general:

Aunque, asumiendo que p y m serán siempre menores que , se puede decir que el orden de complejidad de la solución es:

# ANÁLISIS DE LA SOLUCIÓN

El programa siempre entrega la solución correcta, es decir, la combinación de inversiones que tienen un costo total menor al presupuesto y que, además, tengan el mayor beneficio. Aún así, para realizar esto, es en extremo ineficiente, ya que realiza incluso combinaciones que jamás son usadas y no sólo eso, si no, que la cantidad de combinaciones de inversiones es de orden exponencial , lo que es una gran carga para el equipo.

Por ejemplo, con una cantidad de 24 inversiones, se tiene unidades de tiempo.

## ANÁLISIS DE IMPLEMENTACIÓN

El programa se detiene al escribir la mejor combinación de inversiones en el archivo Salida.out y de liberar la memoria.

El programa siempre obtiene la solución, y no sólo eso, si no, siempre obtiene la mejor solución posible, que es elegida de las combinaciones de inversiones.

El programa en absoluto es eficiente, ya que con un orden de ejecución , se aleja demasiado de lo que sería un orden polinomial. Es por esto que fuerza bruta es considerado una manera simple de encontrar la solución a problemas, pero es una de las más ineficientes.

El programa se podría mejorar en eficiencia, al no generar las combinaciones que superen el presupuesto y además, se podrían agregar más validaciones de datos que provienen del archivo Entrada.in, para así, evitar caídas no controladas del programa. Por otro lado, en vez de hacer la búsqueda de la mejor combinación de inversiones luego de crearlas todas, se podría ir guardando en una variable la combinación de inversiones que generan un mayor beneficio, mientras estas combinaciones se van creando, de esta manera, se ahorraría un al buscar la mejor combinación.

Otra idea para resolver esta problemática es usar la metodología de resolución de problemas “Goloso”, en la cual se irían seleccionando las inversiones que generen un mayor beneficio, hasta que se agote el presupuesto. Si bien, no se garantiza la mejor solución, su eficiencia sería bastante mayor a la de fuerza bruta.

## EJECUCIÓN

Para compilar el programa, se debe ingresar al directorio en que se encuentran todos los archivos mediante el uso de la consola, para luego ingresar el siguiente comando:

*gcc FB2lab1.c -lm -o FB2lab1*

El que genera un ejecutable de nombre FB2lab1. Para hacer uso del programa, se debe escribir en la misma consola:

*./FB2lab1*

Con esto el programa se ejecuta leyendo los datos en el archivo Entrada.in y entrega el resultado en Salida.out.

En caso de que se desee entrar en modo DEBUG, el compilado se hace con el siguiente comando:

*gcc FB2lab1.c -lm -D DEBUG -o FB2lab1*

Para luego ejecutar de igual manera que en el modo anterior, con la única diferencia, es que mientras se ejecuta, se deberá ir presionando la tecla enter, para que se vayan generando y mostrando por consola las combinaciones de inversiones posibles.

El programa fue compilado y ejecutado en un entorno Linux, pero en un entorno Windows, no debería haber problema, ya que no se hizo uso de librerías que no existan en ambos sistemas operativos.

# CONCLUSIONES

Se logró el objetivo que llevaba este laboratorio, que era el aprender y utilizar el método de resolución de problemas Fuerza Bruta, en un problema aplicado, entregando una solución que lo satisface.

Si bien la solución entregada es correcta y siempre será la mejor, es bastante cuestionable la ineficiencia de este método, considerando que existen otros capaces de realizar este mismo proceso, en un tiempo mucho menor, tal como lo haría backtraking con acotamiento o goloso, por ejemplo.

Considerando lo anterior, fuerza bruta es un método que facilita el cómo llegar a una solución, por lo que en primera instancia podría ser un buen primer intento, pero considerando su ineficiencia, es recomendable usarlo sólo en casos en que los datos que deben ser combinados, sean bastante acotados, ya que, al ser de orden exponencial, al tener grandes cantidades de datos, podría no terminar jamás de realizar todas las combinaciones.

Por último, se debe mencionar que el problema de la mochila o knapbag, puede ser una muy buena guía para resolver problemas que le son similares, como es el caso de las inversiones. Su versatilidad permite aplicar su misma o muy similar solución, a otros problemas de características similares.

# REFERENCIAS

* “0-1 Knapsack Problem | DP-10”, Desconocido, 2018, disponible en: <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

# APARTADO

**Para el problema:**

La empresa de inversiones internacional ​clover ​inversions es reconocida a nivel mundial por sus grandes ganancias en este rubro.

Últimamente han nacido muchas nuevas empresas y con ello la lista de inversiones ha crecido de forma exponencial. Esto ha preocupado a los inversionistas ya que ante tantas posibilidades no pueden maximizar sus beneficios.

Por este motivo el equipo de inversionistas le ha pedido a usted que diseñe un algoritmo que dado un capital inicial, pueda comprar inversiones con tal de garantizar el máximo de utilidad. Debe ser desarrollado por fuerza bruta.

Asuma que cada inversión tiene una utilidad específica.

## 

## IDEA

La idea principal de esta solución es generar todas las posibles combinaciones de inversiones, incluso considerando aquellas que superen el presupuesto de la empresa inversora. En la *ilustración 1* se puede apreciar un ejemplo de cómo sería el problema de la mochila (knapsack 0/1), por lo que, en este caso, se mostrará con el de las inversiones en la *tabla 1*.

Tabla 1. Ejemplo inversiones.

|  |  |  |  |
| --- | --- | --- | --- |
| Inversión | Costo | Beneficio |  |
| 1 | 250000 | 1000 |  |
| 2 | 600000 | 6000 |  |
| 3 | 450000 | 4000 | Presupuesto = 1400000 |
| 4 | 300000 | 2000 |  |
| 5 | 100000 | 4500 |  |

En la *ilustración 4* se puede apreciar cómo se generan las combinaciones posibles en el problema de la mochila según sus pesos y beneficios que traen los objetos a guardar. En el caso de las inversiones, simplemente es remplazan los objetos por inversiones, el peso por el costo que tiene la inversión y se mantiene el beneficio como elemento

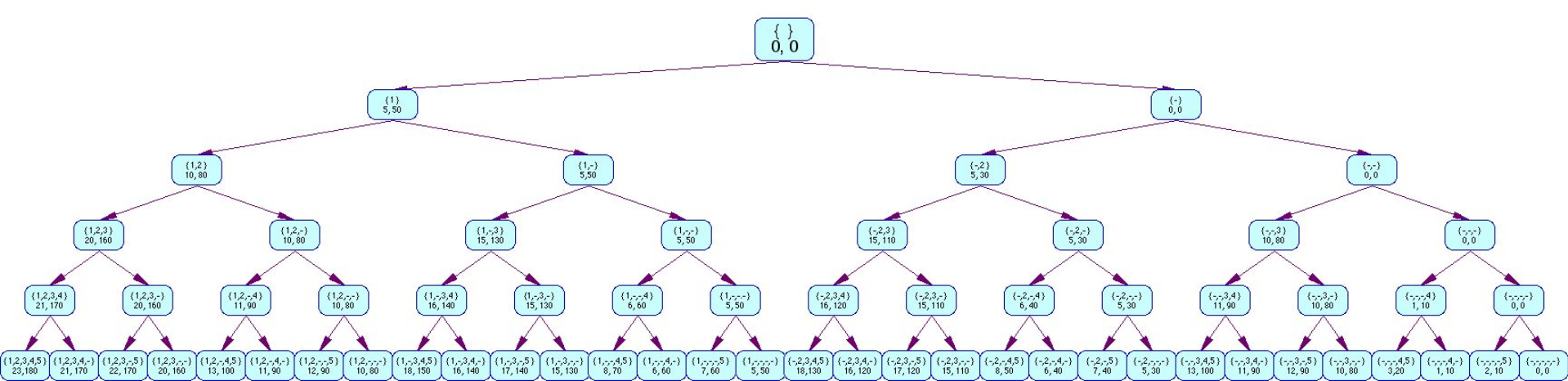


Ilustración 4. Ejemplo combinaciones.

Luego de tener todas las combinaciones posibles con sus costos totales y beneficios, se descartan aquellas que superan el presupuesto, o más bien, no se consideran mientras se realiza la búsqueda entre todas las combinaciones, por la que tenga un mayor beneficio total para la empresa inversora.

## PSEUDOCÓDIGO

powerSet (inversiones, cantidadInversiones):

Combinación = ∅

Contador = 0, Contador2 = 0, Contador3 = 0,

Mientras contador < 2\*cantidadInversiones:

j = 0, Costo\_temp = 0, Beneficio\_temp = 0, Cantidad\_temp = 0

Mientras j < cantidadInversiones:

Si contador & (1<<j):

Combinación->inversión[contador3] = inversiones[j]

Costo\_temp+= inversiones[j]->costo

Beneficio\_temp+= inversiones[j]->costo

Cantidad\_temp += 1

Contador3++

Fin si

Combinación->costoTotal = Costo\_temp

Combinación->beneficioTotal = Beneficio\_temp

Combinación->cantidad = Cantidad\_temp

Combinaciones[contador2] = Combinación

Contador2++

Contador++

Fin mientras

Devolver combinaciones

bruteForce (combinaciones, cantidad\_inversiones, presupuesto):

Mientras i < 2\*\*cantidad\_inversiones:

Si combinaciones[i]->costoTotal < presupuesto

&& combinaciones[i]->beneficioTotal > máximo\_beneficio:

máximo \_beneficio = combinaciones[i]->beneficioTotal

mejor\_combinación = combinaciones[i]

Fin si

Fin mientras

Devolver mejor\_combinacion

## TRAZA

1. Entrada:

* *Presupuesto: 1400000*
* *Cantidad inversiones: 5*
* *Inversiones con su costo y beneficio:*

*250000 1000*

*600000 6000*

*450000 4000*

*300000 2000*

*100000 4500*

1. Creación combinaciones de inversiones:

*[]*

*Costo total: 0 , Beneficio Total; 0*

*[250000 , 1000]*

*Costo total: 250000 , Beneficio Total; 1000*

*.*

*.*

*.*

*[250000 , 1000] [600000 , 6000]*

*Costo total: 850000 , Beneficio Total; 7000*

*.*

*.*

*.*

*[250000 , 1000] [600000 , 6000] [450000 , 4000] [300000 , 2000] [100000 , 4500]*

*Costo total: 1700000 , Beneficio Total; 17500*

1. Mejor combinación de inversiones:

Costo: 1150000 Beneficio: 14500

Costos Inversiones: 600000, 450000, 100000