Algoritmos y Estructura de Datos

# **Práctica 3:**
# Exploración Exhaustiva:
# PROBLEMA DE LA MOCHILA.

Grupo: **2CM2**

*Profra. Cecilia Alborante Morato*

Integrantes:

- Guevara Badillo Areli Alejandra
- Hernández Simón Rebeca

04 de Abril, 2023

# INTRODUCCIÓN

***Exploración Exhaustiva:***
Los algoritmos de búsqueda exhaustiva, también conocidos como fuerza bruta, son un tipo de algoritmo que busca encontrar la solución a un problema probando todas las posibles combinaciones de entrada en un conjunto finito de datos. Esto significa que el algoritmo examina cada una de las posibles soluciones hasta encontrar la solución correcta.

Son una técnica simple y general para resolver problemas, ya que pueden aplicarse a una amplia variedad de situaciones. Sin embargo, su principal limitación es su eficiencia, ya que la cantidad de soluciones posibles a un problema puede crecer exponencialmente con el tamaño de los datos de entrada. Esto significa que, en muchos casos, los algoritmos de búsqueda exhaustiva pueden ser demasiado lentos para resolver problemas prácticos en un tiempo razonable.

Este tipo de algoritmos sigue la siguiente estructura:

1. Se define el conjunto de datos y se establece el problema a resolver.
2. Se establece un conjunto de posibles soluciones para el problema.
3. Se prueba cada una de las posibles soluciones una por una.
4. Se evalúa cada solución probada para determinar si es correcta o no.
5. Se repite este proceso hasta que se encuentra la solución correcta o se han probado todas las posibles soluciones.

En general, los algoritmos de búsqueda exhaustiva pueden ser muy útiles en situaciones donde no hay una solución alternativa más eficiente disponible o cuando se necesita encontrar una solución exacta al problema en cuestión. Sin embargo, en situaciones donde el tamaño de los datos de entrada es grande, estos algoritmos pueden ser demasiado lentos o requerir demasiados recursos computacionales para ser practicos.

***Problema de la Mochila:***
En algoritmia, el problema de la mochila, es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema.

El problema de la mochila (también conocido como problema de la mochila 0/1) es un problema clásico de optimización combinatoria que consiste en encontrar la mejor manera de llenar una mochila con un conjunto limitado de objetos, de tal manera que se maximice el valor total de los objetos incluidos y se respete la capacidad máxima de la mochila. Cada objeto tiene un valor y un peso, y la mochila tiene una capacidad máxima de peso que no puede ser sobrepasada.

El problema se denomina 0/1 porque se plantea la cuestión de si cada objeto debe ser incluido o no, es decir, si debe ser seleccionado o no. Este problema es considerado NP-completo, lo que significa que no existe un algoritmo eficiente conocido para encontrar la solución óptima en todos los casos, y su resolución puede requerir mucho tiempo de cómputo en casos de gran tamaño.

## PROBLEMA DE LA MOCHILA

*Planteamiento del problema:*
*Se tiene una mochila la cual soporta cierta cantidad de peso y se tienen n cantidad de objetos con un valor y un peso asociado, halla todas las combinaciones posibles y la combinación más óptima.*

- Empezamos declarando tres variables globales y las funciones que necesitaremos

```
int n, w, nc;

int entrada();//pide los datos al usuario
void llenado(int obj[3][n]);//le asigna los valores y pesos a los objetos
void combinaciones (int comb[nc][n]);//hace la matriz de combinaciones
void resultados (int obj[3][n],int bin[nc][n]);//calcula los resultados
void mejorres (int*,int*,int bin[nc][n]);//busca el resultado mas optimo
```

- En la función *main()* solamente tendremos las llamadas a las funciones y declararemos el número de combinaciones posibles.

```
void main(){
    printf("Practica 3:\n");
    printf("El Problema de la Mochila\n\n");

    entrada();

    nc = pow(2,n);//calcula el numero de combinaciones posibles
    int obj[3][n];//arreglo de datos
    int comb[nc][n];

    printf("\n");
    llenado(obj);
    combinaciones(comb);
    resultados(obj,comb);
}
```

- Se llama a la función *entrada()* la cual pedirá al usuario introducir el número de objetos que podremos ingresar en la mochila y el peso máximo que esta puede soportar.

```
int entrada(){
    printf("Cuantos objetos quiere meter en la mochila? ");
    scanf("%d", &n);//recibe el numero de objetos

    printf("Cual es el peso maximo que soporta la mochila? ");
    scanf("%d", &w);//peso maximo de la mochila
}
```

- La siguiente función en llamarse es la función *llenado()*, la cual asignará un índice, un valor y un peso a cada uno de los objetos de forma aleatoria.

```
void llenado(int obj[3][n]){
    srand(time(NULL));

    //da el indice de los objetos
    printf("Obj.\t");
    for(int i = 0; i < n; i++){
        obj[0][i] = i+1;
        printf("%d\t",obj[0][i]);
    }

    //asigna un peso aleatorio a los objetos igual o menor al peso maximo.
    printf("\nPeso:\t");
    for(int j = 0; j < n; j++){
        obj[1][j] = (rand()%(w-1)+1);;
        printf("%d\t",obj[1][j]);
    }

    //asigna un valor a los objetos de 1 a 20
    printf("\nValor:\t");
    for(int k = 0; k < n; k++){
        obj[2][k] = (rand()%19+1);
        printf("%d\t",obj[2][k]);
    }
    printf("\n\n");
}
```

- Ahora se llama a la función *combinaciones()*, la cual creará una matriz de tamaño nxm, donde n será el número de objetos y m el número de combinaciones posibles.
- El algoritmo utilizado para generar las combinaciones es conocido como "método de la mitad" o "algoritmo de Gray". Este método genera una secuencia ordenada de combinaciones binarias de forma que cada par de combinaciones sucesivas difieren en sólo un bit.

```
void combinaciones(int comb[nc][n]){
    int razon, fil, cont;
    int i = 0, div = 2;

    for(int j = 0; j < n; j++){//recorre las columnas de la matriz de forma descendiente
        i = 0;//indica indicial de cada bloque
        cont = 1;//indica cuando se debe de poner un uno o un cero
        razon = nc / div;//numero de filas de cada bloque
        do{
            for(fil = i ; fil < i + razon ; fil++ ){//recorre las filas
                if(cont%2!=0){//si el contador es impar se asigna 0 a la casilla
                    comb[fil][j] = 0;
                }
                else{//si es par se asina 1
                    comb[fil][j] = 1;
                }
            }
            cont++;
            i += razon;
        }while(i < nc);//mientras i sea menor al numero de filas
        div *= 2;
    }
}
```

- Por último en la main se llama a la función *resultados()*, en la primera parte de la función se harán los cálculos del peso y los valores totales y se colocaran en su respectivo arreglo.

```
void resultados(int obj[3][n],int comb[nc][n]){
    int pesot[nc];//en este vector se guardaran los pesos totales de las combinaciones
    int valort[nc];//en este vector se guardaran los valores totales de las combinaciones
    int f = 0, c = 0;
    int peso, valor;//variables para guardar los pesos parciales

    do{
        peso = 0;//inicializa el peso en 0
        pesot[c] = 0;//limpia el indice del arreglo en el que se va a guardar el peso
        do{
            peso = obj[1][f] * comb[c][f];//multiplica el peso del objeto por
                                          //la cantidad de veces que estara en la mochila
            pesot[c] = pesot[c] + peso;//va guardando la suma de los pesos
            f++;//aumenta una fila
        }while(f<n);
        f = 0;//reinicia el indice de la fila
        c++;//aumenta una columna
    }while(c<nc);

    c = 0;//reincia el indice de la columna
    f = 0;//reinicia el indice de la fila
    do{
        valor = 0;//inicializa el valor a 0
        valort[c] = 0;//limpia el indice del arreglo en el que se va a guardar el valor
        do{
            valor = obj[2][f] * comb[c][f];//multiplica el valor del objeto por
                                           //la cantidad de veces que estara en la mochila
            valort[c] = valort[c] + valor;//va guardando la suma de los valores
            f++;//aumenta una fila
        }while(f<n);
        f = 0;//reinicia el indice de la fila
        c++;//aumenta una columna
    }while(c<nc);
```

- Y la segunda parte imprimirá la tabla de combinaciones con sus respectivos resultados.

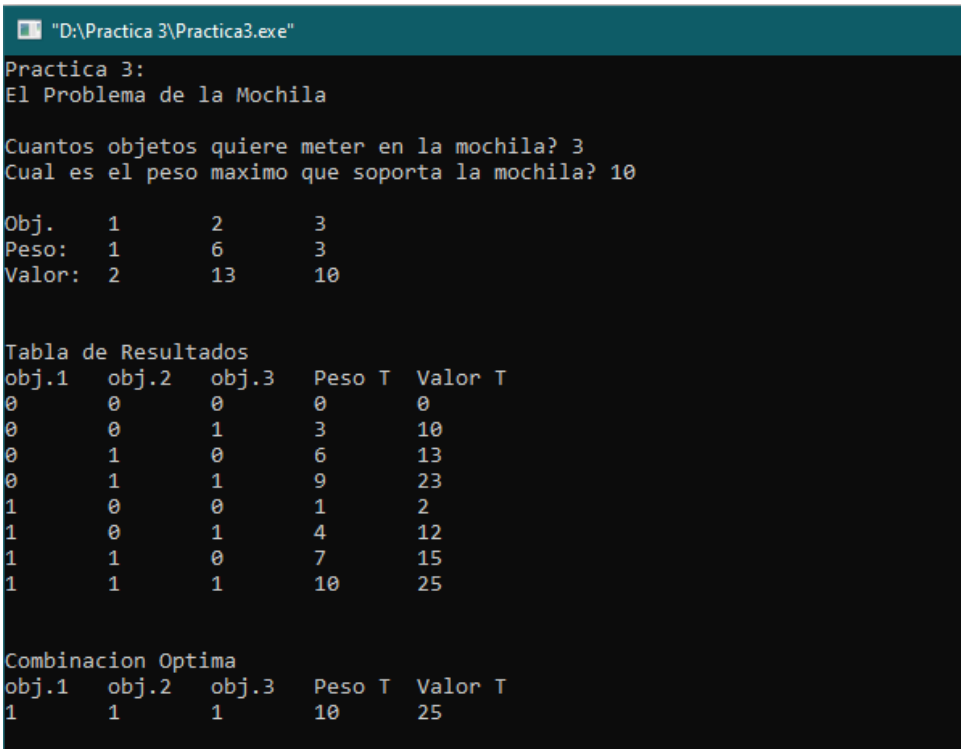
```
    //imprime la tabla de combinaciones y resultados
    printf("\nTabla de Resultados\n");

    //imprime los encabezados de las columnas de la tabla
    for(int i = 0; i < n; i++){
            printf("obj.%d\t",i+1);
    }
    printf("Peso T\tValor T\n");

    //imprime las combinaciones y los pesos y valores totales para cada combinacion
    for(int i = 0; i < nc; i++){
        for(int j = 0; j < n; j++){
            printf("%d\t",comb[i][j]);
        }
        printf("%d\t",pesot[i]);
        printf("%d\n",valort[i]);
    }

    mejorres (pesot,valort,comb);

}
```

- En esta función también llama a la función de *mejorres()* , el cual buscará el índice de la combinación más óptima y después lo imprimirá.

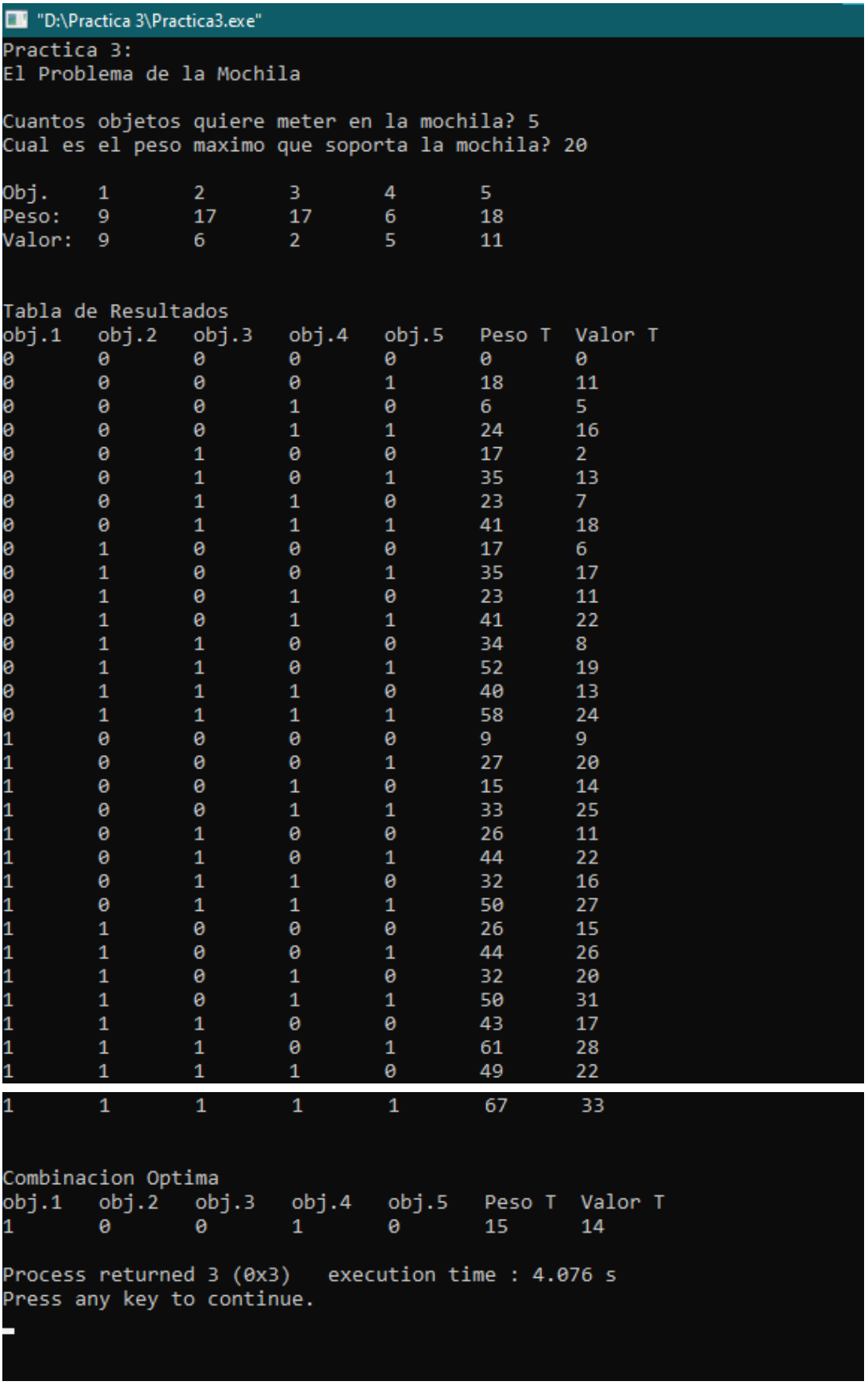
```
void mejorres (int* pesos,int* valores, int comb[nc][n]){
    int mi = 0;//supone que la mejor combinacion se encuentra en el indice 0

    for(int j = 1; j < nc; j++){
        if(valores[j] > valores[mi] && pesos[j]<=w)
            //si el valor de j es mayor que el de mi y el peso de j es menor o igual a w
            mi = j;//la mejor combinacion sera en j
    }

    //imprime la mejor combinacion
    printf("\n\nCombinacion Optima\n");
    if(pesos[mi]<= w){
        for(int i = 0; i < n; i++){
            printf("obj.%d\t",i+1);
        }
        printf("Peso T\tValor T\n");


        for(int j = 0; j < n; j++){
            printf("%d\t",comb[mi][j]);
        }
        printf("%d\t",pesos[mi]);
        printf("%d\n",valores[mi]);
    }
    else printf("NO HAY COMBINACION OPTIMA!!!\n Todas se  pasan del peso permitido :(");

}
```

## RESULTADOS

- Para 3 objetos con un peso máximo de 10 unidades.



- Para 5 objetos con un peso máximo de 20 unidades:



```
"D:\Practica 3\Practica3.exe"
Practica 3:
El Problema de la Mochila

Cuantos objetos quiere meter en la mochila? 5
Cual es el peso maximo que soporta la mochila? 20

Obj.    1       2       3       4       5
Peso:   9       17      17      6       18
Valor:  9       6       2       5       11


Tabla de Resultados
obj.1   obj.2   obj.3   obj.4   obj.5   Peso T  Valor T
0       0       0       0       0       0       0
0       0       0       0       1       18      11
0       0       0       1       0       6       5
0       0       0       1       1       24      16
0       0       1       0       0       17      2
0       0       1       0       1       35      13
0       0       1       1       0       23      7
0       0       1       1       1       41      18
0       1       0       0       0       17      6
0       1       0       0       1       35      17
0       1       0       1       0       23      11
0       1       0       1       1       41      22
0       1       1       0       0       34      8
0       1       1       0       1       52      19
0       1       1       1       0       40      13
0       1       1       1       1       58      24
1       0       0       0       0       9       9
1       0       0       0       1       27      20
1       0       0       1       0       15      14
1       0       0       1       1       33      25
1       0       1       0       0       26      11
1       0       1       0       1       44      22
1       0       1       1       0       32      16
1       0       1       1       1       50      27
1       1       0       0       0       26      15
1       1       0       0       1       44      26
1       1       0       1       0       32      20
1       1       0       1       1       50      31
1       1       1       0       0       43      17
1       1       1       0       1       61      28
1       1       1       1       0       49      22
1       1       1       1       1       67      33


Combinacion Optima
obj.1   obj.2   obj.3   obj.4   obj.5   Peso T  Valor T
1       0       0       1       0       15      14

Process returned 3 (0x3)   execution time : 4.076 s
Press any key to continue.
```