INSTITUTO POLITÉCNICO NACIONAL

"ESCOM" (ESCUELA SUPERIOR DE CÓMPUTO)

DOCENTE: CECILIA ALBORTANTE MORATO

INTEGRANTES:

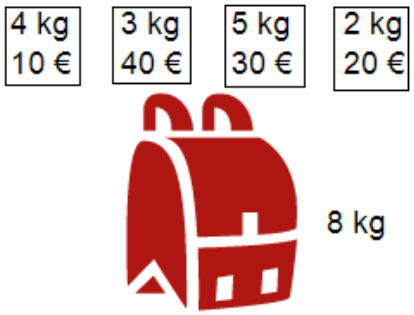
MONTEALEGRE ROSALES DAVID URIEL

VÁZQUEZ BLANCAS CÉSAR SAID

U.A.: ANÁLISIS Y DISEÑO DE ALGORITMOS

GRUPO: 3CM6

PRÁCTICA 7. Mochila entera con programación dinámica.



Mochila entera con programación dinámica.

Objetivo de la práctica.

Implementar en el lenguaje de su preferencia la solución al problema de la mochila entera mediante programación dinámica.

Indicaciones solicitadas.

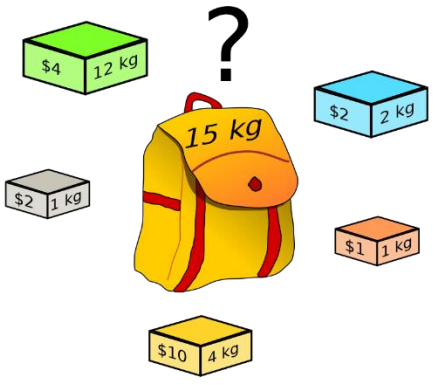
- Se debe definir el número de objetos, el peso y valor de cada objeto y el peso máximo de la mochila.
- Construir la solución óptima e imprimirla en pantalla indicando que objetos se deben ingresar a la mochila, el peso y valor de esa carga óptima.
- También pueden construir los problemas a probar de forma aleatoria.

Introducción.

El problema de la mochila entera es un desafío clásico en la teoría de la optimización y la investigación de operaciones. Este problema es común en campos como la informática. El objetivo es determinar la combinación óptima de objetos que puedes colocar en la mochila para maximizar el valor total, sin exceder la capacidad de carga. No se pueden dividir los objetos para colocar fracciones de ellos en la mochila. Cada objeto se toma completamente o se deja fuera.

Este problema pertenece a la categoría de problemas NP-duros, lo que significa que no existe un algoritmo eficiente conocido para encontrar la solución óptima en todos los casos en un tiempo razonable. Sin embargo, existen enfoques algorítmicos, como la programación dinámica, que pueden proporcionar soluciones aproximadas o exactas según la complejidad del problema.

El problema de la mochila entera tiene aplicaciones prácticas en la toma de decisiones en situaciones donde se deben seleccionar elementos limitados con recursos finitos, como asignación de recursos en proyectos, planificación de inventario y diseño de redes de distribución. Su resolución es fundamental para abordar problemas de optimización en diversos campos.



Desarrollo de la práctica.

Para la resolución de la práctica solicitada, fue necesario implementar un algoritmo el cuál resuelva el problema conocido de mochila entera, con la diferencia de que ahora se utilizará programación dinámica para su resolución. El código utilizado es el siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void imprimir(int* a, int n) {//funcion que imprime los valores de los arreglos enteros
   for (int i = 0; i < n; i++) {
      printf("%d\t", a[i]);
   }
   printf("\n");
}

void imprimir2(float* a, int n) {//funcion que imprime el valor del arreglo flotante
   for (int i = 0; i < n; i++) {
      printf("%.2f\t", a[i]);
   }
   printf("\n");
}

void swap(int* xp, int* yp) {//funcion para hacer un cambio en un arreglo
   int temp = *xp;
   *xp = *yp;
   *yp = temp;
}

void swap2(float* xp, float* yp) {//funcion para hacer un cambio en un arreglo flotante
o el de v/w
   float temp = *xp;
   *xp = *yp;
   *yp = temp;
}

void selection(float* a, int* b, int* c, int n) {//funcion para ordenar de mayor a menor
   int i, j, m;
   for (i = 0; i < n - 1; i++) {
      m = i;
      for (j = i + 1; j < n; j++)//recorre el arreglo de v/w para ordenar en base a el
         if (a[j] > a[m])
```

```
            m = j;
        swap2(&a[m], &a[i]);//hacemos el cambio del v/w para tener todo ordenado en el
arreglo
        swap(&b[m], &b[i]);//hacemos el cambio en los pesos
        swap(&c[m], &c[i]);//hacemos el cambio en los valores
    }
}

float* valorxpeso(int* a, int* b, int n) {
    float* vw = (float*)malloc(n * sizeof(float));
    for (int i = 0; i < n; i++) {//recorre el arreglo para llenarlo
        vw[i] = (float)a[i] / (float)b[i];//asigna los valores dandolos por v/w
    }
    return vw;
}

void datos(int* w, int* v, int n) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        w[i]= rand()%30+1;//asigna los valores random de 1 a 30 en los pesos
        v[i]= rand()%50+1;//asigna los valores random de 1 a 50 en los valores
    }
}

int max(int a, int b) {
    return (a > b) ? a : b;//retorna el maximo entre 2 variables
}

int** mochila(int* w, int* v, int p, int n) {
    int** a = (int**)malloc((n + 1) * sizeof(int*));//reservamos memoria para la tabla
    for (int i = 0; i < (n + 1); i++) {
        a[i] = (int*)malloc((p + 1) * sizeof(int));//reservamos el valor de la tabla por columna
    }

    for (int i = 0; i <= n; i++) {//primer que va a ir para llenar la tabla, recorre a las
columnas
        for (int j = 0; j <= p; j++) {//segundo for para llenar la tabla, recorre las filas
            if (i == 0 || j == 0) {// si estas en la primera fila y columa
                a[i][j] = 0;//asigna ceros
            } else if (w[i - 1] <= j) {//si no y el valor de w en i menos 1 es menor o igual a j
                a[i][j] = max(v[i - 1] + a[i - 1][j - w[i - 1]], a[i - 1][j]);//asigna el valor de la posicion
de la tabla entre el valor de arriba de la posicion bien dada y el de la formula dad que
da un valor de una posicion y le asigna el mayor de estos
```

```
            } else {//si no pasa ninguna
                a[i][j] = a[i - 1][j];//asigna el valor de arriba de la posicion en la que estamos
            }
        }
    }

    return a;//retorna la tabla
}

void imptabla(int** a, int n, int p) {//funcion que imprime la tabla
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= p; j++) {
            printf("%d\t ", a[i][j]);
        }
        printf("\n");
    }
}
void solucion(int** a, int* w, int* v, int p, int n) {//funcionnque te da la solucion con los
pesos de los objetos
    int i = n, j = p;//iniciamos el recorrido desde el final

    while (i > 0 && j > 0) {//mientras i sea mayor a 0 y j lo mismo, esto para recorrer la
tabla
        if (a[i][j] == a[i - 1][j]) {//si la posion en la que estamos es igual al de arriba
            i--;//decrementamos i
        } else {//si no
            printf("Objeto %d - Peso: %d - Valor: %d\n", i, w[i - 1], v[i - 1]);//imprimimos los
valores de los objetos
            j = j - w[i - 1];// j sera el valor del mismo menos el peso de i menos 1
            i--;//decrementamos i
        }
    }
}
int  sumatotal(int* w,int n){
    int p=0;//variable para sumar y retornar la suma total
    for(int i=0;i<n;i++){//recorremos el arreglo de pesos
        p+=w[i];//a p se le suma el valor que hay en la posicion del peso
    }
    return p;//retornamos p
}
int main() {
    int n, p;
    printf("Dame el numero de objetos que quieres en la mochila: ");
```

```
    scanf("%d", &n);//guarda el numero de objetos

    int* w = (int*)malloc(n * sizeof(int));//reserva memoria para el arreglo de pesos
    int* v = (int*)malloc(n * sizeof(int));//reserva memoria para el arreglo de valores

    datos(w, v, n);//llena los arreglos

    float* vw = valorxpeso(v, w, n);//crea y le da valores al arreglo de v/w
    selection(vw, v, w, n);//ordena por el v/w

    printf("Pesos:\t");
    imprimir(w, n);//imprime pesos
    printf("Valores:\t");
    imprimir(v, n);//imprime valores
    printf("Valores por Peso:\t");
    imprimir2(vw, n);//imprime v/w

    printf("Ingresa la capacidad de la mochila: ");
    scanf("%d", &p);//guarda la capacidad de la mochila
    if(sumatotal(w,n)<p){//si la suma de pesos es menor a la capacidad
        printf("La capacidad de la mochila arrebasa el peso de los objetos");//mensaje de
error
    }
    else{//si no
    int** a = mochila(w, v, p, n);//guarda la tabal en a

    printf("\nTabla de Programacion Dinamica:\n");
    imptabla(a, n, p);//imprime la tabla

    printf("\nEl valor máximo que se puede obtener en la mochila es: %d\n",
a[n][p]);//da el resultado de lo maximo de valor en la mochila que es la ultima posicion
de la tabla
    solucion(a,w,v,p,n);//te da los objetos
    free(w);//libera a pesos
    free(v);//libera a valores
    free(vw);//libera a v/w

    for (int i = 0; i <= n; i++) {
        free(a[i]);
    }
    free(a);//libera a la tabla

    return 0;
}
}
```

El código utiliza las siguientes funciones:

Imprimir/imprimir2: se utilizan para imprimir los arreglos de pesos, valores y valores de v/w.

Swap/swap2: Estas funciones realizan el intercambio de valores en los arreglos, que es necesario para ordenar los valores de v/w y mantener la coherencia con los arreglos de pesos y valores.

Selection: Esta función implementa el algoritmo de ordenación por selección para ordenar los valores de v/w en orden descendente, manteniendo la coherencia con los arreglos de pesos y valores.

Valorxpeso: Esta función calcula los valores de v/w dividiendo los valores (a) entre los pesos (b) y devuelve un nuevo arreglo.

Datos: Esta función llena los arreglos de pesos (w) y valores (v) con datos aleatorios.

Max: Esta función simplemente devuelve el máximo entre dos números.

Mochila: Esta función utiliza programación dinámica para llenar una tabla (a) que representa la solución al problema de la mochila. Devuelve la tabla resultante.

Imptabla: Esta función imprime la tabla generada por la función de la mochila.

Solucion: Esta función utiliza la tabla generada por la función de la mochila para imprimir los objetos que deben incluirse en la mochila para obtener el valor máximo.

Sumatotal: Esta función calcula y devuelve la suma total de un arreglo de pesos.

Main: La función principal solicita al usuario la cantidad de objetos, genera datos aleatorios, calcula y ordena los valores de v/w, solicita la capacidad de la mochila, verifica las restricciones, utiliza programación dinámica para resolver el problema de la mochila, e imprime la solución y la tabla generada.

Resultados.

Aquí se puede observar el correcto funcionamiento del código diseñado:

```
C:\Users\alexl\Downloads>Mochila
Dame el numero de objetos que quieres en la mochila: 6
Pesos:  19      9       20      2       7       29
Valores:        47      12      26      2       4       13
Valores por Peso:       2.47    1.33    1.30    1.00    0.57    0.45
Ingresa la capacidad de la mochila: 20

Tabla de Programacion Dinamica:
0       0       0       0       0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0       0
        0       0       0       0       0       0       47      47
0       0       0       0       0       0       0       0       0       12      12      12      12
        12      12      12      12      12      12      47      47
0       0       0       0       0       0       0       0       0       12      12      12      12
        12      12      12      12      12      12      47      47
0       0       2       2       2       2       2       2       2       12      12      14      14
        14      14      14      14      14      14      47      47
0       0       2       2       2       2       2       4       4       12      12      14      14
        14      14      14      16      16      18      47      47
0       0       2       2       2       2       2       4       4       12      12      14      14
        14      14      14      16      16      18      47      47

El valor maximo que se puede obtener en la mochila es: 47
Objeto 1 - Peso: 19 - Valor: 47
```

```
C:\Users\alexl\Downloads>Mochila
Dame el numero de objetos que quieres en la mochila: 3
Pesos:  5       4       28
Valores:        38      5       9
Valores por Peso:       7.60    1.25    0.32
Ingresa la capacidad de la mochila: 10

Tabla de Programacion Dinamica:
0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       38      38      38      38      38      38
0       0       0       0       5       38      38      38      38      43      43
0       0       0       0       5       38      38      38      38      43      43

El valor maximo que se puede obtener en la mochila es: 43
Objeto 2 - Peso: 4 - Valor: 5
Objeto 1 - Peso: 5 - Valor: 38

C:\Users\alexl\Downloads>
```

Conclusión.

En conclusión, la práctica implementa una solución al clásico problema de la mochila entera utilizando programación dinámica. El código proporciona una solución eficiente para determinar la combinación óptima de objetos que maximiza el valor total, respetando la capacidad de carga de la mochila.

También, logramos observar que con la ayuda de la programación dinámica podemos hacer de mayor eficiencia un algoritmo que previamente nos arrojaba una respuesta, pero con mayor tiempo de espera. Finalmente podemos concluir que el uso de programación dinámica hace que resolvamos problemas con mayor eficiencia y de forma correcta.