INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PRÁCTICA 3. Algoritmos Voraces(Greedy)

ALUMNOS:

Colín Ramiro Joel

Ramírez Jiménez Itzel Guadalupe

GRUPO: 3CM3

PROFESORA: Sánchez García Luz María

MATERIA: Análisis y Diseño de Algoritmos

I. Planteamiento del Problema

Se solicita elaborar, analizar y codificar 2 voraces. Los algoritmos con los que se trabajara son los siguientes:

- **Mochila fraccionaria.-** Tenemos: – n objetos, cada uno con un peso (pi) y un beneficio (bi) – Una mochila en la que podemos meter objetos, con una capacidad de peso máximo M.
  - Objetivo: llenar la mochila, maximizando el beneficio de los objetos transportados, y respetando la limitación de capacidad máxima M. • Los objetos se pueden partir en trozos.
- **Cambio de monedas. -** Construir un algoritmo que dada una cantidad P devuelva esa cantidad usando el menor número posible de monedas.

Estas pruebas experimentales, se realizaron en el IDE Dev C++

II. Actividades

Mochila Fraccionaria

```
#include<iostream>

using namespace std;
struct mochila{
	float valor;
	float peso;
	float indice;
};

int main(){

	int n,i,max,j,cont;
	float wmax,valmax=0;
	cout<<"Ingresar el numero de elementos en la mochila: ";
	cin>>n;
	mochila elemento[n],aux;

	cout<<"\n Ingresar elemento en la mochila : \n";

	for(i=0;i<n;i++){
			cout<<"Beneficio "<<i+1<<" :";
			cin>>elemento[i].valor;
			cout<<"Peso "<<i+1<<" :";
			cin>>elemento[i].peso;
```

```
                elemento[i].indice=elemento[i].valor/elemento[i].peso;
        }
//ordenamos los objetos de mayor a menor segun los indices

        for(i=0;i<n;i++){
                max=i;
                        for(j=i+1;j<n;j++){
                                if(elemento[j].indice>elemento[max].indice){
                                        max=j;
                                }
                        }

                                aux=elemento[i];
                                elemento[i]=elemento[max];
                                elemento[max]=aux;
        }

        //verificamos que los objetos se han ordenado

        for(i=0;i<n;i++){
                cout<<i+1<<":";
                cout<<elemento[i].indice<<"%,";
                cout<<elemento[i].peso<<"kg,";
                cout<<elemento[i].valor<<endl;
        }

        cout<<endl;

        cout<<"Peso maximo de la mochila: \n";
        cin>>wmax;

        //Ingresamos los objetos a la mochila
        i=0;

        while(wmax>0){
                if(wmax>elemento[i].peso){
                        wmax=wmax-elemento[i].peso;
                        valmax=valmax+elemento[i].valor;
                }
                else{
                        valmax=valmax+(elemento[i].valor*(wmax/elemento[i].peso));
                        wmax=0;
                }
                i++;
        }
        cout<<"\n El valor maximo de la mochila es: "<<valmax;

}
```

Otras formas de resolver el algoritmo:

Si, por la estrategia de backtracking

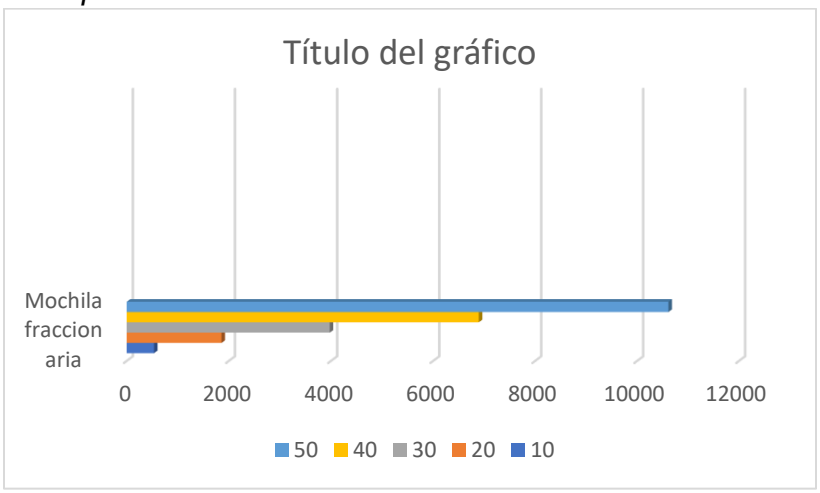
Análisis temporal

Ecuación de recurrencia

t(n) = (4n^2+12n+15)

O(n^2)

Gráfica temporal



Valores de entrada y resultados

Beneficio 1	45
Peso 1	10
Beneficio 2	25
Peso 2	40
Beneficio 3	80

Peso 3	15
Beneficio 4	20
Peso 4	19
Peso Máximo	100
Valor Máximo	170

Cambio de monedas

```
#include <iostream>

using namespace std;

int main()
{
 int moneda[] = {50,25,5,1};
 int veces[] = { 0, 0, 0, 0};
 int n = 4;
 int monto,i;

 cout<<"Monto: ";
 cin>>monto;

 for(i=0;i<n;i++)
 {
  veces[i] = monto/moneda[i];
  monto = monto%moneda[i];
 }
 for(i=0;i<n;i++)
 {
  cout<<"Monedas "<<moneda[i]<<" cantidad "<<veces[i]<<endl;
 }
}
```

Otras formas de resolver el algoritmo:

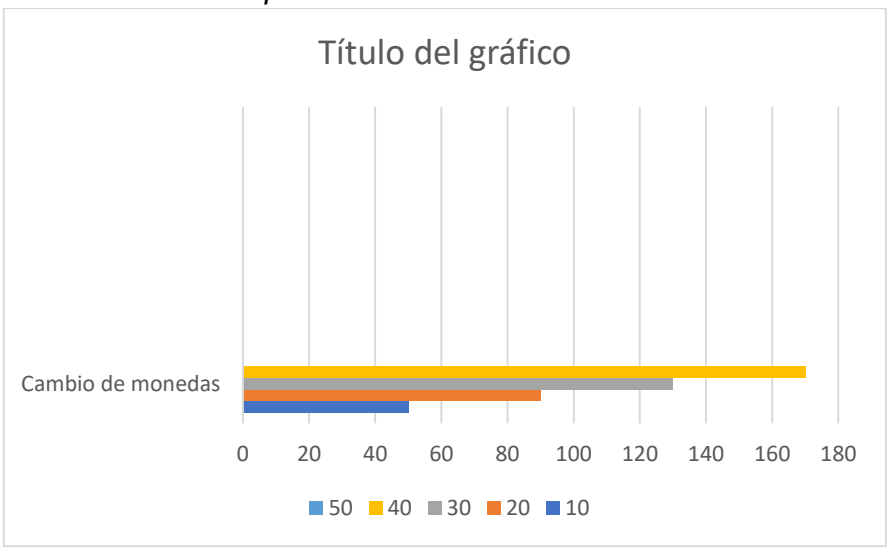
Si, por el método de condicionales y programación dinámica

Análisis temporal

Ecuación de recurrencia

t(n) = 4n+10
O(n)

Gráfica temporal



Valores de entradas y resultados

Monedas	$110	$565	$380
50	**2**	**11**	**7**
25	**0**	**0**	**1**
5	**2**	**3**	**1**
1	**0**	**0**	**0**

A. Para cada algoritmo ¿Existe alguna solución que no sea voraz y que sea óptima? ¿Cuál es?

Si para Mochila fraccionaria por la estrategia de backtracking y para Cambio de monedas por el método de condicionales y programación dinámica.

B. ¿Cuál de los 2 algoritmos es más fácil de implementar?

El más sencillo de implementar fue el de “Cambio de monedas”

C. ¿Cuál de los 2 algoritmos es el más difícil de implementar?

El más difícil de implementar fue el de “Mochila fraccionaria”

D. ¿El comportamiento experimental de los algoritmos era el esperado?¿Por qué?

Si, gracias a que no son difíciles de comprender en teoría fue más fácil implementarlos.

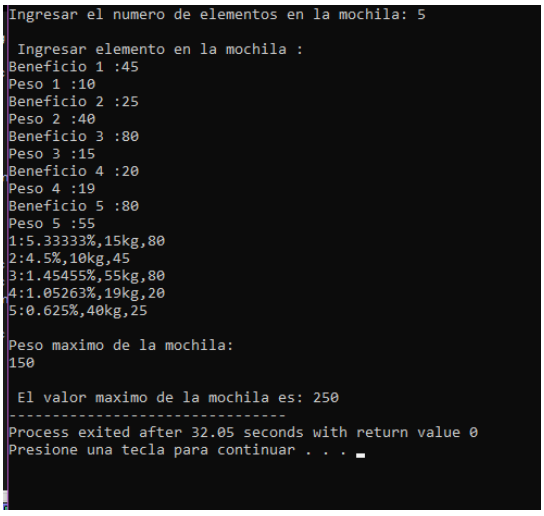
E. ¿Qué otros algoritmos voraces recomendaría para realizar esta práctica?

Códigos de Huffman y camino mínimo en grafos

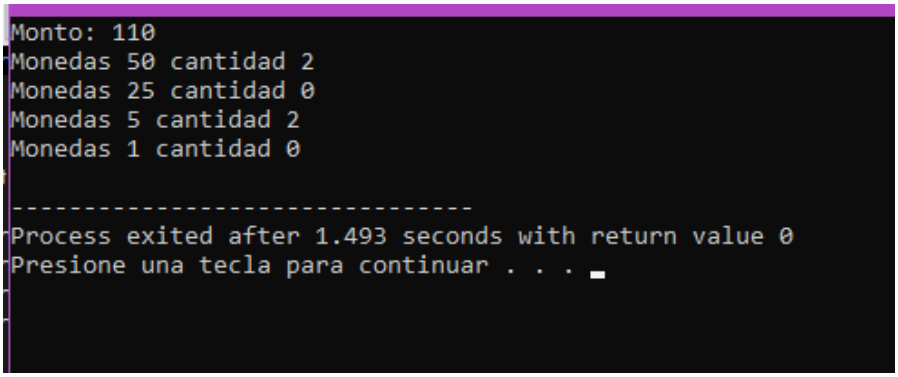
III. Pruebas

A continuación, se presentarán los pantallazos de los resultados de los códigos implementados.

- ***Mochila fraccionaria***



- ***Cambio de monedas***



IV. Conclusiones

Es fundamental la comprensión de estas técnicas, así como de muchas otras para la toma de decisiones a la hora de querer resolver un problema por medio de la computación más específicamente de la programación.

Estos algoritmos consideramos que son más sencillos de implementar que algunos otros con otras técnicas varias. Y es por eso por lo que al finalizar de codificar y analizar su comportamiento frente a otras técnicas podemos concluir que pueden llegar a ser más eficientes en cuanto a costo de tiempo y de recursos.

V. Bibliografía

1. http://verso.mat.uam.es/~pablo.fernandez/cap6-recurrencias.pdf
2. https://www.wextensible.com/temas/voraces/devolver-cambio.html
3. https://www.xatakaciencia.com/matematicas/algoritmos-voraces-problema-del-cambio-de-monedas
4. https://prezi.com/cir9dz-mc93d/mochila-fraccionaria/
5. https://elvex.ugr.es/decsai/algorithms/slides/4%20Greedy.pdf