Diego Viñals Lage y Carlos Vega

Complejidad computacional  3B

Práctica 2: Algoritmos voraces, dinámicos y vuelta atrás

Tabla de contenido

[Introducción 3](#_Toc136973697)

[Algoritmo “Voraz” 4](#_Toc136973698)

[Código 4](#_Toc136973699)

[Complejidad en tiempo y espacio 4](#_Toc136973700)

[Solución 4](#_Toc136973701)

[Características 5](#_Toc136973702)

[¿Divisible? 5](#_Toc136973703)

[Algoritmo “vuelta atrás” 6](#_Toc136973704)

[Código 6](#_Toc136973705)

[Complejidad computacional en tiempo y espacio 6](#_Toc136973706)

[Salida 7](#_Toc136973707)

[Características 7](#_Toc136973708)

[¿Divisible? 7](#_Toc136973709)

[Algoritmo “Dinámico” 8](#_Toc136973710)

[Código 8](#_Toc136973711)

[Complejidad computacional en tiempo y espacio 8](#_Toc136973712)

[Salida 9](#_Toc136973713)

[Características 9](#_Toc136973714)

[¿Divisible? 9](#_Toc136973715)

# Introducción

El objetivo de esta práctica es afianzar y aprender los algoritmos voraces, dinámicos y de vuelta atrás. Para lograrlo, se nos presenta el problema de la mochila, el cual debemos resolver utilizando estos tres tipos de algoritmos. Una empresa de café se ha comunicado con nosotros para que los ayudemos a decidir qué mercancías deben transportar en un vuelo internacional, teniendo en cuenta los costos de transporte y seguros. La empresa tiene un límite de peso que puede llevar y desea transportar el mayor valor posible.

Para esto, aplicaremos los diferentes paradigmas que hemos visto en clase: el paradigma voraz, el paradigma dinámico y el paradigma de vuelta atrás.

La capacidad límite que el avión puede transportar es de 600.000 kg, y nosotros disponemos de estos cafés.

|  |  |  |
| --- | --- | --- |
| Producto | Peso (kg) | Precio total ($) |
| Pales cafés de Tanzania | 260.000 | 26.000.000 |
| Pales cafés de Hawái | 150.000 | 18.000.000 |
| Pales cafés de Nicaragua | 240.000 | 19.200.000 |
| Pales cafés de Jamaica | 190.000 | 34.200.000 |
| Pales cafés de Colombia | 130.000 | 19.500.000 |
| Pales cafés de Kenia | 140.000 | 9.800.000 |

# Algoritmo “Voraz”

## Código

Texto

Descripción generada automáticamente

## Complejidad en tiempo y espacio

La complejidad computacional de este algoritmo es de O(n log n), donde n es el número de productos. Usa OrderByDescending, que tiene esa complejidad, después isa foreach que tiene complejidad de N, pero al final la complejidad viene dada por la ordenación de los elementos.

La complejidad espacial es de O(n), ya que usa una lista llamada productos, que contiene todos los productos, es decir, n

## Solución

La solución que devuelve este algoritmo es:

Texto

Descripción generada automáticamente

## Características

## ¿Divisible?

# Algoritmo “vuelta atrás”

## Código

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Complejidad computacional en tiempo y espacio

Este algoritmo tiene una complejidad temporal en el peor de los casos de O(2^n), ya que la función BuscarMejorSeleccion se ejecuta dos veces anidadas, lo que aumenta exponencialmente la complejidad.

La complejidad espacial de este algoritmo es de O(n), ya que utiliza una lista que almacena todos los productos, y el espacio requerido es proporcional al número de productos, que en este caso se representa por "n".

## Salida

La salida de este algoritmo es la siguiente:

Texto

Descripción generada automáticamente

## Características

## ¿Divisible?

# Algoritmo “Dinámico”

## Código

Texto

Descripción generada automáticamente

## Complejidad computacional en tiempo y espacio

La complejidad temporal de este algoritmo es de O(n), donde "n" es el número de productos. El algoritmo utiliza dos bucles, pero no están anidados, lo que significa que se ejecutan secuencialmente uno después del otro. Por lo tanto, la complejidad del algoritmo viene dada por el bucle que itera sobre los productos, y en este caso, ese bucle tiene una complejidad lineal O(n).

La complejidad espacial de este algoritmo es de O(n), donde "n" es el número de productos. El algoritmo utiliza una lista para almacenar todos los productos disponibles. Dado que la lista contiene "n" productos, el espacio requerido para almacenarlos es proporcional al número de productos.

## Salida

La salida de este algoritmo es la siguiente:

Texto

Descripción generada automáticamente

## Características

## ¿Divisible?