**Programación Dinámica**

1. **Que es la programación dinámica**

La programación dinámica es una técnica esencial de optimización utilizada para abordar problemas combinatorios y de toma de decisiones. Su importancia radica en su capacidad para descomponer problemas complejos en subproblemas más manejables y resolverlos de forma independiente. Esto no solo hace posible resolver problemas que de otro modo serían difíciles o incluso imposibles de abordar, sino que también permite una solución eficiente para problemas complejos como el recorrido del viajante, el problema de la mochila y la búsqueda de la subsecuencia común más larga. entre otros.

1. **Características**

* Almacenamiento de resultados previos: La programación dinámica almacena los resultados de los subproblemas previamente resueltos para evitar cálculos redundantes.
* Divide y vencerás: La técnica se basa en el análisis del problema en subproblemas más pequeños y manejables.
* Soluciones óptimas: La programación dinámica proporciona soluciones óptimas a los problemas, garantizando la mejor solución posible dentro del conjunto de todas las posibles soluciones.
* Problemas solapados: Los subproblemas en la programación dinámica se solapan, lo que significa que los mismos subproblemas se resuelven varias veces. Esto permite una optimización al evitar recalcular soluciones ya conocidas.

1. **Ventajas de la programación dinámica:**

* Eficiencia: La programación dinámica puede ofrecer soluciones eficientes para problemas complejos, reduciendo significativamente el tiempo de ejecución en comparación con otros enfoques.
* Optimalidad: Proporciona soluciones óptimas garantizadas para problemas de optimización, lo que significa que la solución encontrada es la mejor posible dentro del conjunto de todas las soluciones posibles.
* Reutilización de resultados: Almacenando y reutilizando los resultados de subproblemas previamente resueltos, se evita el cálculo redundante, lo que mejora aún más la eficiencia del algoritmo.
* Aplicabilidad: La técnica de programación dinámica es aplicable a una amplia gama de problemas combinatorios y de toma de decisiones, lo que la hace versátil y útil en diversas situaciones.

1. **Desventajas de la programación dinámica:**

Complejidad de implementación: La implementación de algoritmos de programación dinámica puede ser complicada y requerir un entendimiento profundo del problema y de la técnica en sí misma. Esto puede hacer que sea difícil para los programadores novatos.

Requisitos de memoria: Al almacenar los resultados de subproblemas previamente resueltos, la programación dinámica puede requerir una cantidad significativa de memoria, especialmente para problemas con muchos subproblemas.

Dificultad para identificar subproblemas: Identificar correctamente los subproblemas y definir la recurrencia puede ser desafiante en algunos casos, lo que puede dificultar la aplicación efectiva de la programación dinámica.

No siempre es la mejor opción: Aunque la programación dinámica puede proporcionar soluciones eficientes y óptimas para muchos problemas, no es la mejor opción para todos los problemas. En algunos casos, otros enfoques pueden ser más adecuados o más fáciles de implementar.**Principio del formulario**

1. **Problema de la mochila**

El problema de la mochila es un desafío clásico en optimización combinatoria que implica seleccionar la combinación óptima de objetos para maximizar el valor total, sin exceder la capacidad de una mochila dada. La técnica de programación dinámica es una forma eficaz de resolver este problema al descomponerlo en subproblemas más pequeños y reutilizar las soluciones parciales. La relación de recurrencia, que define cómo calcular el valor máximo posible para diferentes capacidades de mochila y diferentes conjuntos de objetos, se utiliza para construir una tabla de programación dinámica. Esta tabla almacena las soluciones parciales y se actualiza gradualmente durante la resolución del problema, lo que permite encontrar la solución óptima de manera eficiente.

* 1. **Código**

***Mochila.cpp***

#include <iostream>

#include <vector>

#include "validaciones.h"

**using** **namespace** std;

// Estructura de un objeto

**struct** Objeto {

**int** peso;

**int** valor;

};

// Función para calcular el máximo entre dos números

**int** max(**int** a, **int** b) {

**return** (a > b) ? a : b;

}

// Función que resuelve el problema de la mochila utilizando programación dinámica

**int** problemaMochila(**int** capacidad, vector<**int**>& pesos, vector<**int**>& valores, **int** n) {

vector<vector<**int**>> V(n + 1, vector<**int**>(capacidad + 1, 0));

**for** (**int** i = 1; i <= n; i++) {

**for** (**int** j = 1; j <= capacidad; j++) {

**if** (pesos[i - 1] <= j) {

V[i][j] = max(V[i - 1][j], valores[i - 1] + V[i - 1][j - pesos[i - 1]]);

} **else** {

V[i][j] = V[i - 1][j];

}

}

}

**return** V[n][capacidad];

}

**int** main() {

system("cls");

validaciones v;

**int** n;

// Validación para el número de objetos

**do** {

n = v.ingresar\_enteros("\nIngrese el numero de objetos (menos de 25): ");

**if** (n <= 0 || n >25) {

cout << "\nPor favor, ingrese un numero valido de objetos." << endl;

}

} **while** (n <= 0 || n > 25);

vector<**int**> pesos(n);

vector<**int**> valores(n);

// Ingreso de los pesos y valores de los objetos

**for** (**int** i = 0; i < n; ++i) {

cout << "\nObjeto " << i + 1 << endl;

// Validación para el peso del objeto

**do** {

pesos[i] = v.ingresar\_enteros("\nIngrese el peso del objeto (mayor que 0 y menor que 100): ");

**if** (pesos[i] <= 0 || pesos[i] >= 100) {

cout << "\nPor favor, ingrese un peso mayor que 0 y menor que 100" << endl;

}

} **while** (pesos[i] <= 0);

// Validación para el valor del objeto

**do** {

valores[i] = v.ingresar\_enteros("\nIngrese el valor del objeto (mayor que 0): ");

**if** (valores[i] <= 0) {

cout << "\nPor favor, ingrese un valor mayor que 0." << endl;

}

} **while** (valores[i] <= 0);

}

**int** capacidadMaxima;

capacidadMaxima = v.ingresar\_enteros("\nIngrese la capacidad maxima de la mochila: ");

// Resolución del problema de la mochila

**int** maxValor = problemaMochila(capacidadMaxima, pesos, valores, n);

cout << "\nEl valor maximo que se puede llevar en la mochila es: " << maxValor << endl;

system("pause");

**return** 0;

}

* 1. **Ejecución del código:**

A screenshot of a computer screen

Description automatically generated

*Figura 1. Ejecucicón de código de Mochila*

1. **Bibliografía**

(2010) *Programación Dinámica*. Available at: https://ocw.ehu.eus/pluginfile.php/40172/mod\_resource/content/1/diseno\_alg/contenidos/programacion-dinamica.pdf (Accessed: 06 March 2024).

*TOMi.digital - Programación dinámica*. (s. f.). TOMi.Digital. https://tomi.digital/es/27819/programacion-dinamica?utm\_source=google&utm\_medium=seo

inDrive.Tech. (2023, 4 septiembre). *Comprender la programación dinámica para poder utilizarla de forma eficaz*. HackerNoon. https://hackernoon.com/es/comprender-la-programacion-dinamica-para-que-puedas-usarla-eficazmente

‌