DIVIDE Y VENCERÁS

CARLOS BRIONES
JULIO VALLADARES
SANTIAGO MENA

QUE ES ?

El paradigma de "Divide y Vencerás" es una técnica fundamental en la informática para resolver problemas complejos

Consiste en dividir un problema en un conjunto de subproblemas del mismo tipo, pero de menor tamaño, resolverlos de forma independiente cada uno y una vez que se obtienen las soluciones, combinar todas las soluciones para conseguir la solución del problema original.

CONCEPTOS CLAVE

Etapa de División: El problema original se divide en subproblemas más pequeños.

Etapa de Resolución: Los subproblemas se resuelven de manera independiente.

Etapa de Combinación: Las soluciones de los subproblemas se combinan para obtener la solución del problema original.

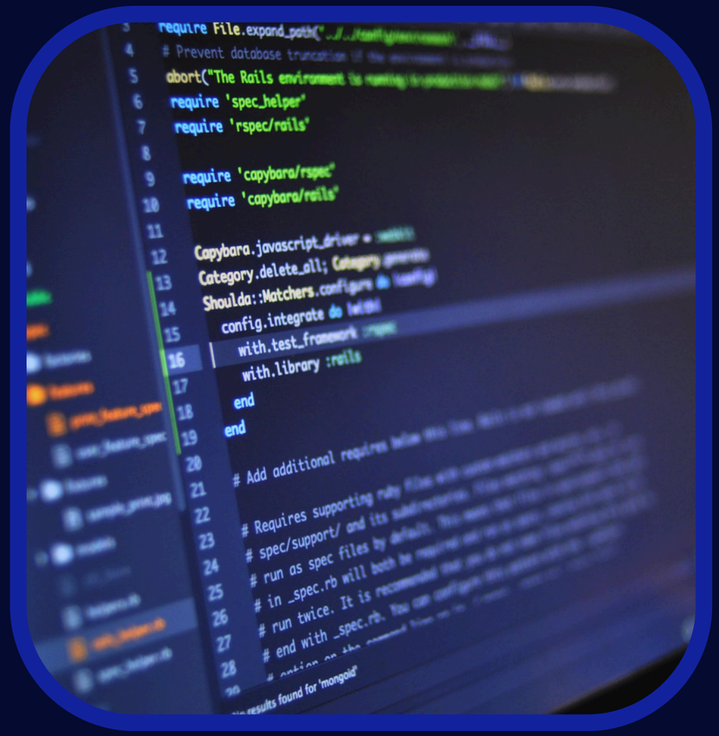
EJEMPLOS DE APLICACIÓN

Merge Sort: Un algoritmo de clasificación que divide la lista en mitades, las ordena recursivamente y luego fusiona las soluciones parciales.

División: Se divide la lista original en dos mitades hasta que las sublistas tengan un solo elemento.

Resolución: Se ordenan las sublistas pequeñas.

Combinación: Se combinan las sublistas ordenadas comparando los elementos de ambas sublistas y combinandolas en orden ascendente o descendente.



EJEMPLOS DE APLICACIÓN

Algoritmo de Búsqueda Binaria: Se utiliza para encontrar un elemento en una lista ordenada.

División: Se compara el elemento medio del arreglo con el valor que se desea buscar.

Resolución: Dependiendo si el valor es mayor o menor que el elemento medio, se descarta una de las mitades del arreglo, y se sigue buscando en la otra mitad.

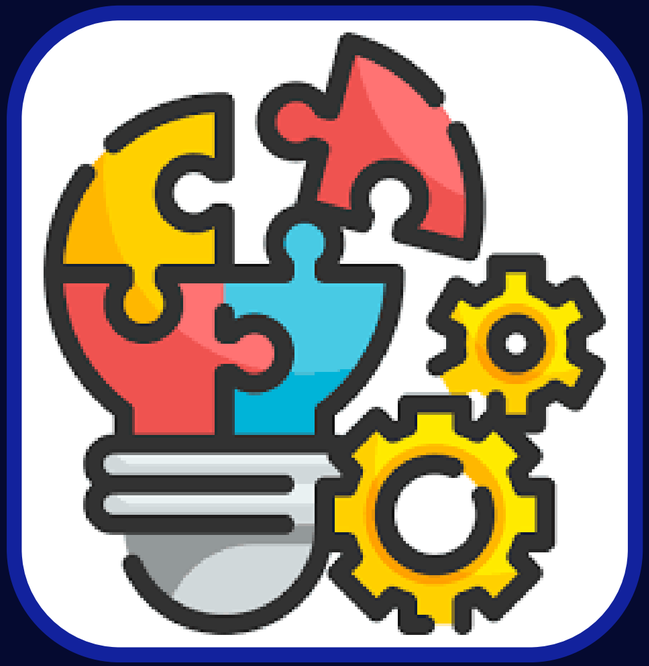
Combinación: En este caso, esta última fase no es necesaria, debido a que el subproblema anterior ya ha resuelto el problema.

Eficiencia y Aplicabilidad

Este enfoque es útil cuando los subproblemas se pueden resolver de manera eficiente y la combinación de las soluciones no es excesivamente difícil. Sin embargo, no es adecuado utilizar este enfoque en todos los problemas, debido a que en algunos casos la fase de combinación es demasiado compleja.

Impacto en la Resolución de Problemas

El paradigma “Divide y Vencerás” tiene un gran impacto en la resolución de problemas, sobre todo en aquellos donde las soluciones recursivas son naturales o cuando los problemas tienen una estructura jerárquica que permite su descomposición.



EJEMPLO 1

```
namespace Ejemplo1
{
    internal class Program
    {
        //  Funcíon para verificar que el numero ingresado contiene tres digitos
        static int Verificar(int n)
        {
            if (n >= 100 && n <= 999)
            {
                return n;
            }
            return 0;
        }

        //  Funcion para obtener el primer digito del numero ingresado
        static int PriDig(int n) { return (n / 100) % 10; }

        //  Funcion para obtener el segundo digito del numero ingresado
        static int SegDig(int n) { return (n / 10) % 10; }

        //  Funcion para obtener el tercer digito del numero ingresado
        static int TerDig(int n) { return n % 10; }

        //  Funcion para calcular la suma de los tres numeros
        static int Suma(int n) { return PriDig(n) + SegDig(n) + TerDig(n); }

        //  Funcion para solicitar el numero al usuario y mostrar el resultado de la suma
        static void Resultados()
        {
            Console.WriteLine("Digite un numero de tres digitos.");
            Console.Write("Numero: ");
            int n = Convert.ToInt32(Console.ReadLine());

            if (Verificar(n) == n)
            {
                Console.WriteLine($"La suma de los tres digitos es: {Suma(n)}");
            }
            else
            {
                Console.WriteLine("ERROR: El numero digitado no es de tres digitos.");
            }
        }

        static void Main(string[] args)
        {
            Resultados();
            Console.ReadKey();
        }
    }
}
```

EJEMPLO 2

```
namespace Ejemplo2
{
    internal class Program
    {
        //  Funcion para encontrar el elemento realizando busqueda binaria
        static int BusqBin(int[] arreglo, int izq, int der, int dato)
        {
            if (der >= izq)
            {
                int med = izq + (der - izq) / 2;

                //  Si el elemento esta en el medio
                if (arreglo[med] == dato) return med;
                //  Si el elemento es mas pequeño, deberá estar en la mitad de la izquierda
                if (arreglo[med] > dato) return BusqBin(arreglo, izq, med - 1, dato);
                //  Si el elemento es mas grande, deberá estar en la mitad de la derecha
                return BusqBin(arreglo, med + 1, der, dato);
            }
            //  El elemento no se encuentra en el arreglo
            return -1;
        }
```

```
        static void Main(string[] args)
        {
            int[] arreglo = { 2, 5, 7, 10, 12, 13, 15, 16, 18, 20};
            int result, dato;

            Console.WriteLine("Arreglo:");
            for (int i = 0; i < arreglo.Length; i++) { Console.Write(arreglo[i] + " "); }
            Console.Write($"\n\nElemento que se desea buscar: ");
            dato = Convert.ToInt32(Console.ReadLine());

            result = BusqBin(arreglo, 0, arreglo.Length - 1, dato);

            if (result != -1)
            {
                Console.WriteLine($"Posicion del elemento en el arreglo: {result}");
            } else Console.WriteLine("Elemento no encontrado.");

            Console.ReadKey();
        }
    }
}
```

EJEMPLO 3

```
namespace Ejemplo3
{
    internal class Program
    {
        //  Funcion para imprimir los elementos del arreglo
        static void ImprimirArr(int[] arr)
        {
            foreach (var dato in arr)
            {
                Console.Write(dato + " ");
            }
        }

        //  Funcion para combinar los dos sub-arreglos ordenados en un solo
        static void merge(int[] arr, int izq, int mitad, int der)
        {
            int n1 = mitad - izq + 1;
            int n2 = der - mitad;

            int[] izqArr = new int[n1];
            int[] derArr = new int[n2];

            Array.Copy(arr, izq, izqArr, 0, n1);
            Array.Copy(arr, mitad + 1, derArr, 0, n2);

            int i = 0, j = 0;
            int k = izq;

            while (i < n1 && j < n2)
            {
                if (izqArr[i] <= derArr[j])
                {
                    arr[k] = izqArr[i];
                    i++;
                }
                else
                {
                    arr[k] = derArr[j];
                    j++;
                }
                k++;
            }

            while (i < n1)
            {
                arr[k] = izqArr[i];
                i++; k++;
            }

            while (j < n2)
            {
                arr[k] = derArr[j];
                j++; k++;
            }
        }

        static void mergeSort(int[] arr, int izq, int der)
        {
            if (izq < der)
            {
                int mitad = izq + (der - izq) / 2;

                mergeSort(arr, izq, mitad);
                mergeSort(arr, mitad + 1, der);
                merge(arr, izq, mitad, der);
            }
        }

        //  Funcion para ordenar el arreglo completo
        static void MergeSort(int[] arr)
        {
            mergeSort(arr, 0, arr.Length - 1);
        }

        static void Main()
        {
            int[] arr = { 8, 1, 7, 4, 2, 3, 10, 6, 5, 9 };
            Console.WriteLine("Numeros Desordenados:");
            ImprimirArr(arr);

            MergeSort(arr);

            Console.WriteLine("\n\nNumeros Ordenados");
            ImprimirArr(arr);

            Console.ReadKey();
        }
```

EJEMPLO 4

```
namespace Ejemplo4
{
    internal class Program
    {
        //  Funcion recursiva para encontrar el maximo y el minimo en un arreglo
        static (int, int) MaxMin(int[] arr, int izq, int der)
        {
            //  Si hay un solo elemento, ese es tanto el máximo como el mínimo.
            if (izq == der)
                return (arr[izq], arr[izq]);

            //  Si hay dos elementos, se comparan y se retornan el mayor y el menor.
            if (der == izq + 1)
            {
                if (arr[izq] > arr[der]) return (arr[izq], arr[der]);
                else return (arr[der], arr[izq]);
            }

            //  Encuentra el punto medio para dividir el arreglo en dos mitades.
            int mid = (izq + der) / 2;

            //  Llama recursivamente para encontrar el máximo y mínimo en las dos mitades.
            var (max1, min1) = MaxMin(arr, izq, mid);
            var (max2, min2) = MaxMin(arr, mid + 1, der);

            //  Devuelve el máximo y mínimo combinando los resultados de las dos mitades.
            return (Math.Max(max1, max2), Math.Min(min1, min2));
        }
```

```
        static void Main(string[] args)
        {
            Console.WriteLine("Buscar Máximo y Mínimo usando Divide y Vencerás");

            Console.Write("Ingrese el tamaño del arreglo: ");
            int tam = int.Parse(Console.ReadLine());

            int[] arr = new int[tam];
            Console.WriteLine("Ingrese los elementos del arreglo:");
            for (int i = 0; i < tam; i++)
            {
                Console.Write($"Elemento {i + 1}: ");
                arr[i] = int.Parse(Console.ReadLine());
            }

            var (max, min) = MaxMin(arr, 0, arr.Length - 1);

            Console.WriteLine("------------------------------");
            Console.WriteLine($"Máximo: {max}");
            Console.WriteLine($"Mínimo: {min}");

            Console.ReadKey();
        }
    }
}
```

CONCLUSIÓN

CONCLUSIÓN

“Divide y Vencerás” es un enfoque poderoso y versátil para el diseño de algoritmos. Su capacidad para simplificar problemas complejos y mejorar la eficiencia de los algoritmos lo convierte en una técnica esencial para cualquier desarrollador de sistemas. Su aplicabilidad ha desencadenado la creación de algoritmos robustos y eficientes en áreas como el ordenamiento y búsqueda de datos.

GRACIAS