|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

*Profesor:* Ing.Castillo Hernandez Gabriel

*Asignatura: Algoritmos y Estructura de Datos I*

*Grupo: 11*

*No de Práctica(s): 11*

*Integrante(s): Calzada Martinez Jonathan Omar*

*Semestre: 2019-1*

*Fecha de entrega: 14 de Nov del 2019*

*Observaciones:*

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

Practica #11

Estrategias para la construcción de algoritmos

Parte I

## Objetivos

Conocer las diferentes técnicas para implementar algoritmos

## Actividades

1. Aplicar las bases para desarrollar algoritmos eficientemente
2. Probar los códigos ejemplo para ver cómo se aplican las diferentes técnicas de implementación

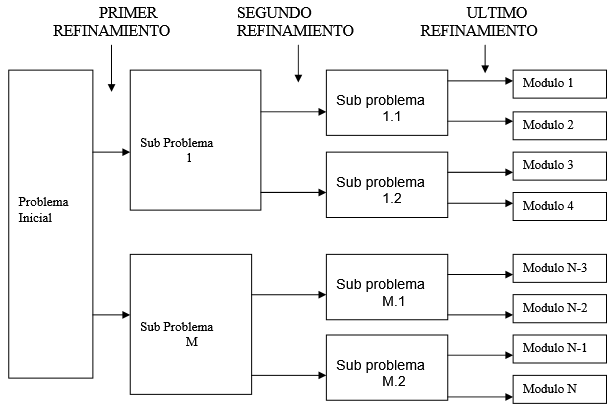
## Introducción

1. **Top-down**

El diseño top-down es una herramienta que presenta en primer lugar una solución a un problema general utilizando tres o cuatro pasos solamente. Cada uno de esos pasos en la primera solución se dividen en otros subpasos. Este proceso se repite varias veces, en cada iteración se produce una solución más detallada al problema original. Cuando los pasos ya no se pueden subdividir, el algoritmo ha terminado. El diseño top-down también se conoce como **descomposición funcional o refinamiento de pasos.**

El diseño top-down automáticamente produce módulos en un algoritmo. Recordemos que los módulos son grupos de instrucciones que desarrollan funciones específicas.

Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.



La utilización de la técnica de diseño Top-Down tiene los siguientes objetivos básicos:

* Simplificación del problema y de los subprogramas de cada descomposición.
* Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
* El programa final queda estructurado en forma de bloque o módulos lo que hace más sencilla su lectura y mantenimiento.

El enfoque *top-down* enfatiza la planificación y conocimiento completo del sistema. Se entiende que la codificación no puede comenzar hasta que no se haya alcanzado un nivel de detalle suficiente, al menos en alguna parte del sistema. Esto retrasa las pruebas de las unidades funcionales del sistema hasta que gran parte del diseño se ha completado.

A diferencia de bottom-up, aquí se empiezan a hacer los cálculos de n hacia abajo. Además, se aplica una técnica llamada memorización la cual consiste en guardar los resultados previamente calculados, de tal manera que no se tengan que repetir operaciones.

1. **Bottom-Up**

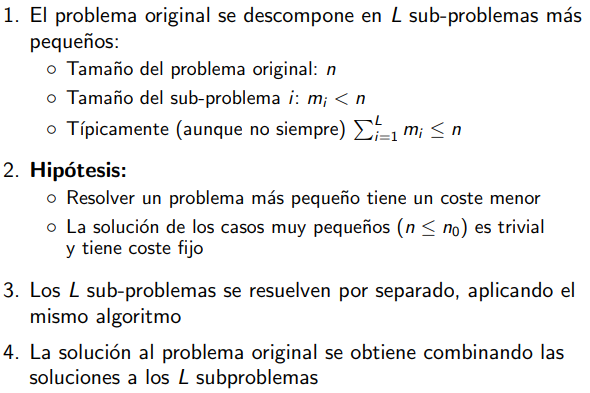
Hace énfasis en la programación y pruebas tempranas, que pueden comenzar tan pronto se ha especificado el primer módulo. Este enfoque tiene el riesgo de programar cosas sin saber como se van a conectar al resto del sistema, y esta conexión puede no ser tan fácil como se creyó al comienzo. La reutilización del código es uno de los mayores beneficios del enfoque *bottom-up*

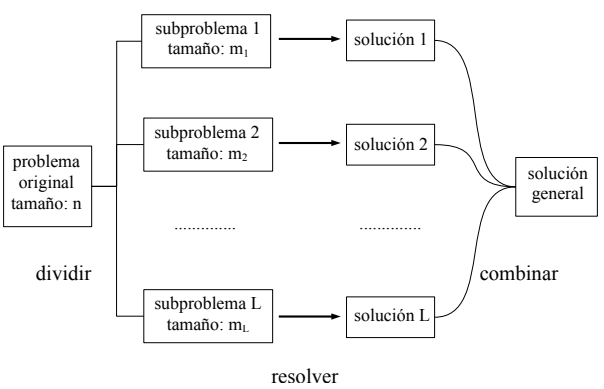
El objetivo de esta estrategia es resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de una o más soluciones que se guardan en una tabla, ésta previene que se vuelvan a calcular las soluciones.

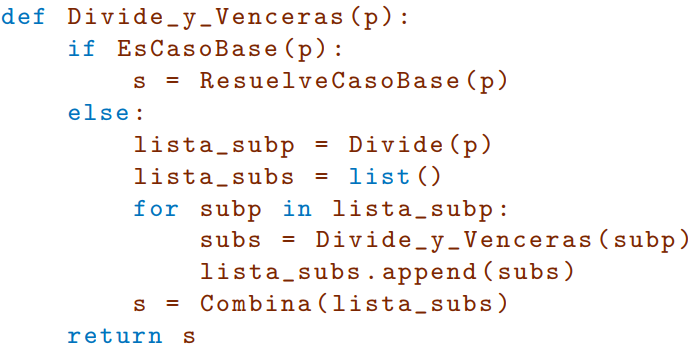
1. **Divide y vencerás**

* Consiste en descomponer un problema en subproblemas, resolver independientemente los subproblemas para luego combinar sus soluciones y obtener una solución del problema original.
* La técnica de diseño de algoritmos Divide y Vencerás trata de resolver un problema de forma recursiva, a partir de la solución de subproblemas del mismo tipo pero de menor tamaño.
* A su vez, estos subproblemas se resolverán de la misma forma y así sucesivamente (recuérdese el caso de las torres de Hanoi), hasta alcanzar subproblemas lo bastante pequeños como para que se puedan resolver directamente.

**Esquema:**







El número de subproblemas (L) en que se divide el problema original debe ser pequeño, independiente de la entrada, y a ser posible balanceado (es preferible dividir n en (n/2, n/2) que en (n − 1, 1)).

Cuando el problema original genera un único subproblema, se habla de algoritmos de simplificación (por ejemplo, el algoritmo que calcula el factorial).

**Ventajas e inconvenientes que se derivan de un diseño recursivo:**

Ventajas:

* simple, robusto y elegante
* buena legibilidad
* fácil depuración y mantenimiento del código

Inconvenientes:

* mayor costo espacial por el uso intensivo de la pila del sistema
* En ocasiones no se reduce, sino que incluso aumenta la complejidad temporal con respecto a soluciones iterativas

**Aplicaciones:**

* + Esta técnica puede ser utilizada con éxito en problemas como multiplicación de matrices, ordenación de vectores, búsquedas en arreglos, algoritmos de naturaleza recursiva, etc.

1. **Fuerza bruta**

El objetivo de resolver problemas por medio de fuerza es bruta es hacer una búsqueda exhaustivamente de todas las posibilidades que lleven a la solución. Por ejemplo, para encontrar una contraseña se debe realizar una combinación exhaustiva de caracteres alfanuméricos, generando cadenas de longitud variable. La desventaja de resolver problemas por medio de esta estrategia es el tiempo.

1. **Algoritmos ávidos**

Esta estrategia se diferencia de fuerza bruta porque va tomando una serie de decisiones en un orden específico, una vez que se ha ejecutado esa decisión ya no se vuelve a considerar. En comparación con fuerza bruta, un algoritmo ávido puede ser más rápido, aunque una desventaja es que la solución que se obtiene no siempre es la más óptima.

Este método trata de producir algún tipo de mejor resultado a partir de un conjunto de opciones candidatas. Par ello se va procediendo paso a paso realizándose la mejor elección (utilizando una función objetivo que respeta un conjunto de restricciones) de entre las posibles.

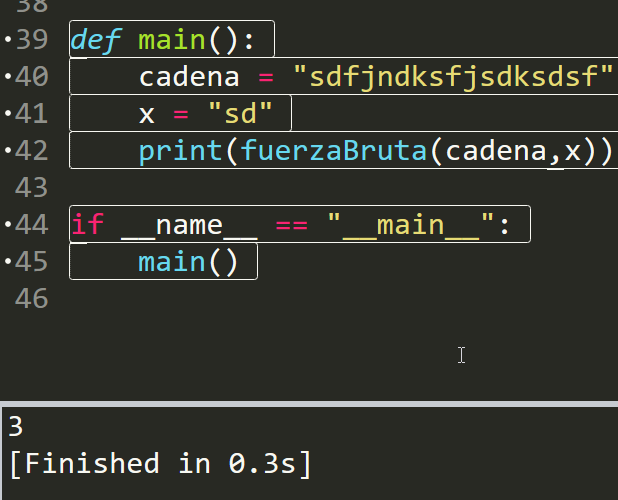
Ejemplos de su aplicación: problemas de optimización, como el conocido de la mochila, en la búsqueda de caminos mínimos sobre grafos, la planificación en la ejecución de programas en un computador, etc.

## Desarrollo de la práctica

**Ejercicio 1:**

Hacer un programa que encuentre el número de ocurrencias de una secuencia de caracteres s en un texto t por ejemplo: s =texto abbd t=jdfabbdjabbdhjabbdlaabbddabb nos da como resultado 3 mientras que s =bb t = jfbbbfbbbhh da como resultado 4





**Conclusiones:**

Solo se logro concretar el nivel básico por falta de tiempo, pero se logro entender el concepto y la aplicación de fuerza bruta.

La practica me sirvió para afirmar el conocimiento de los diseños de solución de algoritmos que vimos de tarea como previo a esta práctica y así mismo, pude aclarar dudas de como ir elaborándolos en casos concretos.