Análisis de Algoritmos
Tarea 4: Diseño de Algoritmos usando Inducción Matemática.

Profesora: María de Luz Gasca Soto
Ayudantes: Rodrigo Fernando Velázquez Cruz
Teresa Becerril Torres
Nombre: Alvaro Ramirez Lopez
N°. de Cuenta.: 316276355
Correo: alvaro@ciencias.unam.mx

1. Considera los siguientes Problemas:

Π: Partición. Dada una lista L de n enteros positivos y distintos, particionar (dividir) la lista en dos sublistas L_1 y L_2, cada una de tamaño $\frac{n}{2}$ tal que: $L = L_1 \cup L_2; L_1 \cap L_2 = \emptyset$; se satisface, ademas, que la diferencia entre las sumas de los enteros en las dos listas sea minima. Puedes suponer que n es múltiplo de dos.

β: Cambio de Base. Dado un numero en base 8 convertirlo a binario. La entrada es un arreglo de dígitos en base 8 y la salida es un arreglo de bits.

Δ: Distancias. Sea $T = (V, A)$ un árbol binario con n vertices. El árbol T esta representado por su lista de adyacencias. Construir una matriz M de $n \times n$ tal que el elemento $M[i, j]$ sea igual a la distancia entre los vertices v_i y v_j.

1.1. Para el Problema seleccionado...

a) Diseñar un algoritmo eficiente, usando Inducción Matemática, que solucione el problema y que use el menor numero de comparaciones.
b) Determinar la complejidad del algoritmo obtenido.

Solución:

a) Diseñar un algoritmo eficiente, usando Inducción Matemática, que solucione el problema y que use el menor numero de comparaciones.

Problema seleccionado: Problema β

Dado un numero en base 8 convertirlo a binario. La entrada es un arreglo de dígitos en base 8 y la salida es un arreglo de bits.

Precondiciones: El arreglo de entrada `A` contiene los dígitos a cambiar de base, dichos dígitos están distribuidos dígito a dígito en el arreglo `A`, `A` es de tamaño n, donde $n > 1$ y contiene los dígitos de un numero en base 8, `T` es el diccionario que contiene la representación de los dígitos en base 8 (identificador) y su representación en base 2 (valor), `num_binario` es un arreglo de tamaño 0.

Vista del diccionario T:

Base 8	Base 2
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Diseño de algoritmo: Se recibe un arreglo `A` de dígitos en base 8, se recibe un diccionario `T`

Vamos a analizar los 2 casos que podemos tener en el arreglo `A`:

a) El caso base es cuando el arreglo `A` tiene un solo dígito, en este caso se busca en `T` el identificador del dígito y se agrega a un arreglo `num_binario` el valor que corresponde al identificador en `T`.

b) Para el caso cuando el arreglo `A` tiene mas de un dígito, se realiza un ciclo que recorre el arreglo `A` y se busca en `T` su identificador y se agrega a un arreglo `num_binario` el valor que corresponde al identificador en `T`.

Al final se regresa el arreglo `num_binario` que contiene el cambio de base de los dígitos del arreglo `A` a base 2.

Postcondiciones: La cantidad de elementos del arreglo `num_binario` es igual a la cantidad de elementos del arreglo `A`, los elementos del arreglo `num_binario` son los dígitos del arreglo `A` en base 2.

b) Determinar la complejidad del algoritmo obtenido. Para el caso base, el algoritmo realiza una búsqueda en `T` y agrega un elemento al arreglo `num_binario`, por lo que la complejidad es $O(1)$.

Para el caso donde el arreglo `A` tiene mas de un dígito, el algoritmo realiza un ciclo que recorre el arreglo `A` y realiza una búsqueda en `T` y agrega un elemento al arreglo `num_binario`, por lo que la complejidad es $O(n)$.

2. Caja Negra.

Se tiene acceso a un algoritmo, denominado Caja Negra, del cual solo se conocen sus resultados, contesta: **sí** o **no**. Si se le da una secuencia de n números enteros y un entero k, el algoritmo responde **si** o **no**, dependiendo si existe un subconjunto de esos números cuya suma sea exactamente k.

Mostrar como usar esta Caja Negra $O(n)$ veces en un proceso que encuentre el subconjunto en cuestión, si es que existe, donde n es el tamaño de la secuencia.

Solución:

Tenemos una secuencia $S = [x_1, x_2, ..., x_n]$ de números enteros y un entero k, Podemos considerar a k como una descomposición aditiva, donde cada sumando de esta descomposición es un elemento de S, entonces basándonos en esa lógica podemos hacer el siguiente algoritmo.

Algoritmo:

1. **Declaramos** una variable $c = k$.
2. **Si** c es menor o igual a 0 entonces terminamos el algoritmo, **en otro caso** le pasamos al algoritmo *CajaNegra(c, S)*

3. **Si** el algoritmo *CajaNegra(c, S)* regresa **si** entonces terminamos el algoritmo, **en otro caso** hacemos un bucle que recorra la secuencia S y le pasamos al algoritmo *CajaNegra(c-x_i, S-x_i)* donde x es el elemento de la secuencia S que estamos sacando del final, la condición de termino sera hasta que c sea menor o igual a 0.
4. Dentro del bucle, si *CajaNegra(c, S)* nos regresa true entonces terminamos el algoritmo regresando S, **en otro caso** continuamos con el bucle.

Complejidad: La complejidad del algoritmo es $O(n)$ ya que se hace una llamada a la caja negra por cada elemento de la secuencia S, también cada llamada recursiva se va eliminando elementos de S.

3. Opcional:

Los Dos menores elementos de un Conjunto.

Problema μ: Dada una secuencia $S = [x_1, x_2, ..., x_n]$ de números enteros, encontrar a los dos menores elementos de S, usando la menor cantidad posible de comparaciones.

a) Diseñar, usando Inducción Matemática, un algoritmo que resuelva el problema μ.
b) Determinar el numero de comparaciones que realiza el algoritmo propuesto.

Solución:

a) Diseñar, usando Inducción Matemática, un algoritmo que resuelva el problema μ.

Primero vamos a ver cual seria el o los casos bases:

1. Caso cuando la secuencia S tiene solo 2 elementos: en este caso los elementos x_1, x_2 están en S, ademas $x_1 < x_2$ y se regresan como los dos menores elementos de S.
2. Caso cuando la secuencia S tiene solo 2 elementos pero son iguales: igual que el caso anterior, podemos mostrar los elementos de S como $S = [x_1, x_2]$, pero en este caso $x_1 = x_2$ y se regresan como los dos menores elementos de S aunque sean el mismo elemento.

Ahora vamos a ver el caso donde la secuencia S tiene mas de 2 elementos:

1. En este caso vamos a dividir la secuencia S en dos subsecuencias S_1, S_2 de tamaño $\frac{n}{2}$ cada una, donde S_1 contiene los primeros $\frac{n}{2}$ elementos de S y S_2 contiene los últimos $\frac{n}{2}$ elementos de S.
2. Hacemos recursion sobre esas 2 subsecuencias hasta que sean de un tamaño de 2 elementos o menos.
3. Ahora nos quedamos el elemento menor en esa subsecuencia de 2 elemento comparando uno con otro, el menor lo agregamos a una secuencia auxiliar S'.
4. Hacemos lo mismo con la otra subsecuencia de 2 elementos y el elemento menor lo agregamos a la secuencia auxiliar S' y asi con los otros casos recursivos.
5. Al final nos quedamos con una secuencia S' que contiene los elementos menores de cada subsecuencia de 2 elementos, S' es de tamaño $\frac{n}{2}$, entonces volvemos a hacer el procedimiento de dividir S' en dos subsecuencias hasta que nos queden una subsecuencia de 2 elementos, esos serán los 2 elementos menores de nuestra secuencia S original.

b) Determinar el numero de comparaciones que realiza el algoritmo propuesto.

Para el caso base, el algoritmo realiza una comparación y regresa los 2 elementos de la secuencia S, por lo que la complejidad es $O(1)$.

Para el caso cuando S tiene n elementos, el algoritmo realiza una comparación y divide la secuencia S en 2 subsecuencias de tamaño $\frac{n}{2}$, entonces hace $\frac{n}{2}$ comparaciones y asi sucesivamente hasta que la secuencia S tenga 2 elementos, por lo que la complejidad esta acotada en $O(\log n)$ ya que se hacen $\log n$ comparaciones y hay $\log n$ llamadas sobre la secuencia S'.