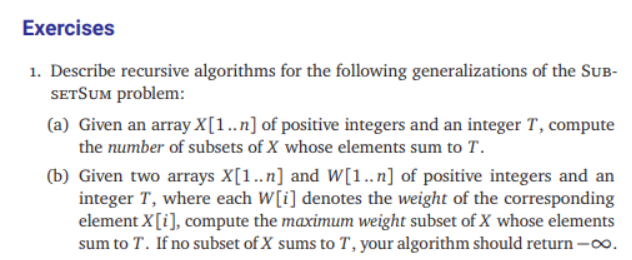
**Parte Teorica - Tarea 4**

**1.**



**A)**

**Especificación**

**Entrada:** Un arreglo X [1..n) de enteros positivos y un entero positivo T >= 1

**Salida:** El número de subconjuntos de X que suman T

**Ejemplo**

Dado el siguiente ejemplo X = [2,4,1,3,5] para un T = 6, la cantidad de subconjuntos que sumados sus elementos suman 6 es de 3 y los subconjuntos respectivos son:

* [2,4]
* [2,1,3]
* [1,5]

**Estrategia**

La estrategia de reintento es tener dos caminos para construir las ramas del árbol de soluciones, las cuales son, recorrer de izquierda a derecha el arreglo X de números, del cual se toma y no se toma el valor en el que se esté en un determinado momento, donde cada solución es la suma de los elementos que se han tomado por uno de los dos caminos, para podar se utiliza una verificación que dirá si la suma solución es mayor que el numero T, esa rama ya no se sigue generando.

**Código**

**Captura de pantalla de computadora

Descripción generada automáticamente**

**B)**

**Especificación**

**Entrada:** Un arreglo X [1..n) y un arreglo W [1..n) de enteros positivos, donde W[i] representa el peso del elemento X[i]. Además de, un entero positivo T >= 1.

**Salida:** El subconjunto de X que tenga los elementos más pesados que suman T,

**Ejemplo**

Dado el siguiente ejemplo X = [2,4,1,3,5] y W = [15,2,10,20,5] para un T = 6, el subconjunto con los elementos más pesados que suman 6 es:

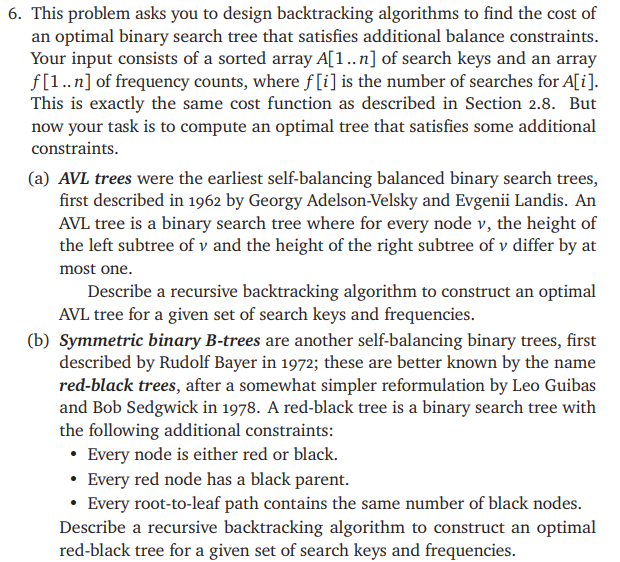
* [2,1,3]

**Estrategia**

La estrategia de reintento es tener dos caminos para construir las ramas del árbol de soluciones, las cuales son, recorrer de izquierda a derecha el arreglo X de números, del cual se toma y no se toma el valor en el que se esté en un determinado momento, donde cada solución es la suma de los elementos que se han tomado por uno de los dos caminos, para podar se utiliza una verificación que dirá si la suma solución es mayor que el numero T, esa rama ya no se sigue generando y para devolver el subconjunto más pesado lo que se hace es que una vez un subconjunto sume T se escoge o reemplaza ese subconjunto verificando en el arregle W cada elemento.

**Código**

****

**2. **

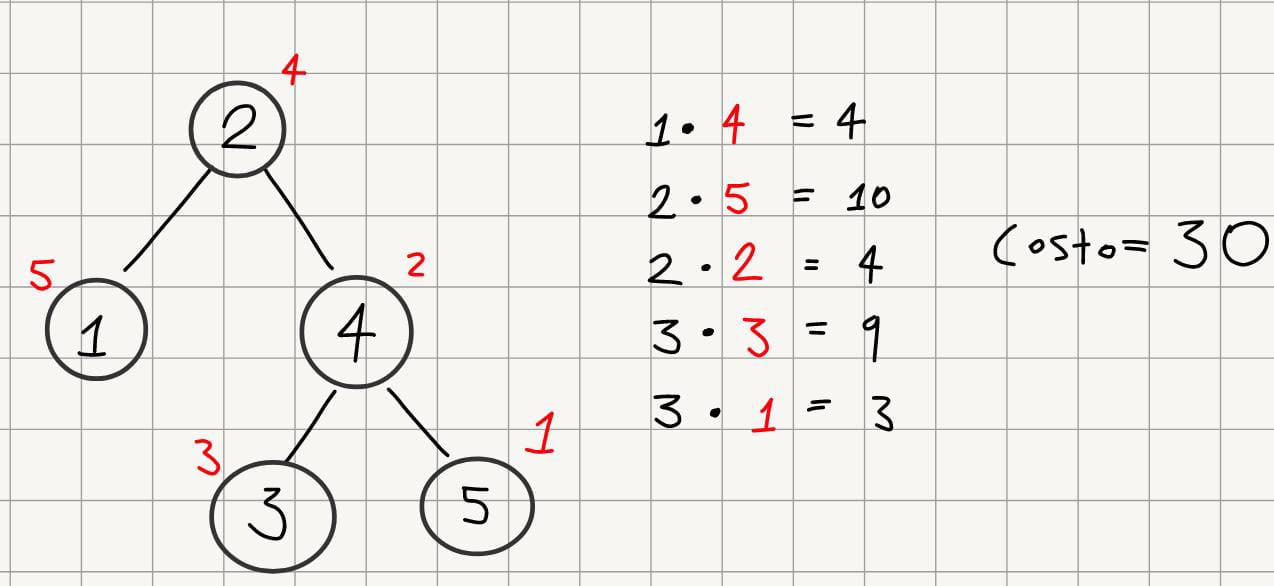
**A)**

**Especificación:**

**Entrada:** Un arreglo ordenado A[1..n) correspondiente al valor de los nodos, así como un arreglo F[1..n) correspondiente a la frecuencia de accesos en A, tal que n >= 1 y donde hay exactamente una frecuencia de accesos F[i] para cada nodo A[i].

**Salida:** Un óptimo árbol T de búsqueda binaria AVL de costo mínimo, así como un entero positivo K que representa el costo de las búsquedas en el árbol.

**Ejemplo:** Dado el siguiente ejemplo A = [1,2,3,4,5] y F = [5,4,3,2,1] el árbol de búsqueda AVL de costo mínimo es el siguiente:



**Estrategia:** La estrategia a seguir para poder hacer el mayor numero de podas posibles es primero que todo utilizar un algoritmo recursivo o una ecuación que dado un arreglo ordenado de menor a mayor (para este problema el arreglo es A el cual tiene el valor de los nodos) decida cuales son los nodos que pueden ser utilizados como nodos raíz para la construcción de un árbol óptimo T de búsqueda binaria AVL sin tener en cuenta los costos, Luego de eso se utiliza otra función recursiva para ahora si realizar la elaboración del árbol óptimo de búsqueda binaria AVL de costo mínimo **,** para lo que haremos uso en cada llamado recursivo a los nodos raíz posibles para la construcción de cada rama del árbol, con el ejemplo anterior entonces se tendría que dado un A = [1,2,3,4,5] y utilizando la función que encuentra los nodos raíz posibles para cada rama, se encontraría que para el primer nodo raíz los valores posibles son [2,3,4] ahora con eso en mente y teniendo constancia de en que nivel nos encontramos del árbol, como apenas se está construyendo entonces se asume que su nivel de profundidad es 1, entonces utilizando la lista de frecuencias para cada valor se tiene que las frecuencias para 2 = 4, para 3 = 3 y para 4 = 2, ahora para elegir el nodo raíz más optimo se deben multiplicar estas frecuencias con el respectivo nivel del árbol, por lo que tendríamos algo tal que así: 1x4 = 4, 1x3 = 3 y 1x2 = 2, luego de tener los cálculos realizados se procede a tomar como nodo raíz al que dio el valor con la multiplicación más alta, en este caso el nodo de valor 2 que tiene frecuencia de 4, posterior a eso se debe hacer el mismo procedimiento para cada rama del árbol resultante, para la izquierda se evaluaría que no nodo es posible entre A = [1] y para la rama derecha evaluaría que nodo es posible para A= [3,4,5], para seguidamente realizar la misma operación y para ir construyendo el árbol en queda decisión

**Seudocódigo:**

arbol T

costo\_minimo = 0

función arbol\_avl (A, F, nivel):

global arbol T, costo\_minimo

si len(A) es igual a 1 entonces:

agregar A[0] al arbol T

si no entonces:

nodos\_raiz = encontrar\_nodos\_raiz(A)

mejor\_nodo\_raiz = None

frecuencia = 0

for i en nodos\_raiz:

si F[i] \* nivel > frecuencia entonces:

mejor\_nodo\_raiz = i

frecuencia = F[i] \* nivel

agregar A[mejor\_nodo\_raiz] al abol T

costo\_minimo += frecuencia

arbol\_avl (A[: mejor\_nodo\_raiz], F, nivel + 1) # rama izquierda

arbol\_avl (A[mejor\_nodo\_raiz +1:], F, nivel + 1) # rama derecha

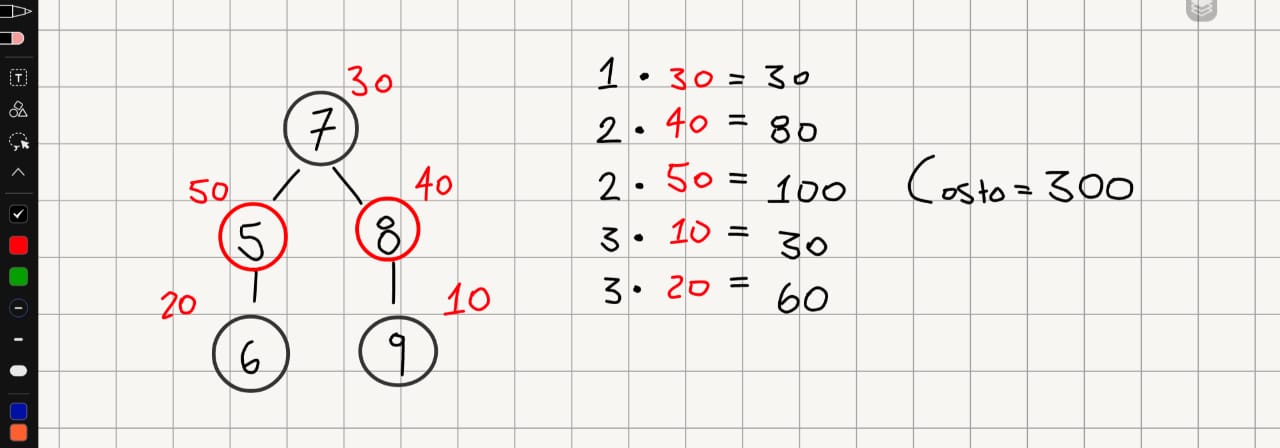
**B)**

**Especificación:**

**Entrada:** Un arreglo ordenado A[1..n) correspondiente al valor de los nodos, así como un arreglo F[1..n) correspondiente a la frecuencia de accesos en A, tal que n >= 1 y donde hay exactamente una frecuencia de accesos F[i] para cada nodo A[i].

**Salida:** Un óptimo árbol T de búsqueda binaria simétrico (árbol rojo - negro) de costo mínimo, así como un entero positivo K que representa el costo de las búsquedas en el árbol si es posible.

**Ejemplo:** Dado el siguiente ejemplo A = [5,6,7,8,9] y F = [50,20,30,40,10] el árbol de búsqueda simétrico de costo mínimo es el siguiente:



**Estrategia:** Similar al punto anterior también se debe tener un algoritmo recursivo o ecuación que permita encontrar los nodos raíz posibles para la construcción de un árbol óptimo de búsqueda binaria simétrico, para este árbol la búsqueda de nodos raíz potenciales es un poco más sencilla de encontrar usualmente están en el centro de los nodos disponibles cuando la cantidad de nodos disponibles es un numero impar mayor o igual a 3, en caso de que hayan únicamente 2 nodos disponibles esos dos son nodos raíz potenciales, luego dentro de la función recursiva que permite la construcción del árbol lo que se hace es básicamente lo mismo que en el punto anterior donde se multiplican las frecuencias de cada nodo raíz posible por el nivel del árbol y se escoge el que de el mayor resultado, con la particularidad de que al momento de elegir el nodo raíz se debe preguntar de que color es el nodo padre de la raíz, de modo que se cumplan las siguientes reglas:

Si no tiene nodo padre, el nodo raíz elegido será coloreado de negro

Si el nodo padre es de color negro, el nodo raíz elegido será coloreado de rojo

Si el nodo padre es de color rojo, el nodo raíz elegido será coloreado de negro.

Y así constantemente hasta terminar la construcción de todas las ramas del árbol.

**Seudocódigo**

arbol T

costo\_minimo = 0

función arbol\_simetrico (A, F, nivel, nodo\_padre):

global arbol T, costo\_minimo

si len(A) es igual a 1 entonces:

agregar A[0] al arbol T

si no entonces:

nodos\_raiz = encontrar\_nodos\_raiz(A)

mejor\_nodo\_raiz = None

frecuencia = 0

for i en nodos\_raiz:

si F[i] \* nivel > frecuencia entonces:

mejor\_nodo\_raiz = i

frecuencia = F[i] \* nivel

nodo\_elegido = A[mejor\_nodo\_raiz]

si el color del nodo\_padre es None entonces:

colorear nodo\_elegido de negro

si el color del nodo\_padre es negro entonces:

colorear nodo\_elegido de rojo

si no entonces:

colorear nodo\_elegido de negro

agregar nodo\_elegido al abol T

costo\_minimo += frecuencia

arbol\_alv (A[: mejor\_nodo\_raiz], F, nivel + 1) # rama izquierda

arbol\_alv (A[mejor\_nodo\_raiz +1:], F, nivel + 1) # rama derecha