Departamento de
Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Algoritmos y Complejidad
Proyecto de Programación Dinámica

Para minimizar el tiempo promedio de una búsqueda, conviene que los árboles binarios de búsqueda se mantengan balanceados. Esto es cierto siempre que sea verdad que todos los elementos del árbol se acceden con la misma frecuencia. Si se conoce que algunos de los nodos son accedidos con mayor frecuencia que otros, la disposición óptima de los nodos puede resultar en un árbol no balanceado.

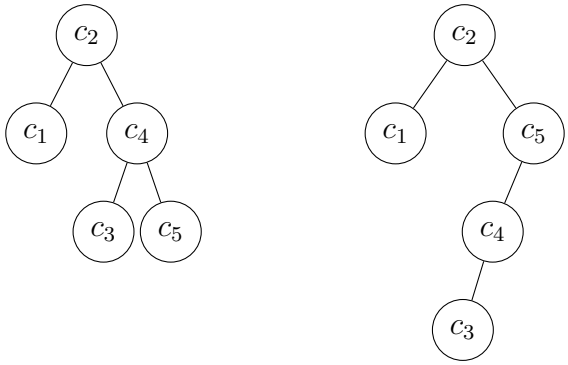
Para formalizar, suponga que se tiene un conjunto ordenado de n claves $c_1 < c_2 < \ldots < c_n$, y la secuencia de accesos es tal que la clave c_i es accedida k_i veces. Definimos la profundidad d_i del i-ésimo nodo como 0 si es la raíz, y 1 más que la profundidad del padre para el resto. Entonces, la cantidad total de comparaciones para la secuencia de accesos dada es:

$$C = \sum_{i=1}^{n} k_i(d_i + 1)$$

Por ejemplo, dadas 5 claves $c_1 < c_2 < c_3 < c_4 < c_5$ con la siguiente distribución en el número de accesos:

Clave	c_1	c_2	c_3	c_4	c_5
# Accesos	5	4	1	4	6

Dos posibles árboles binarios de búsqueda son:



El número de comparaciones realizadas será:

- Primer árbol: $\sum_{i=1}^{5} k_i(d_i + 1) = 5 \cdot 2 + 4 \cdot 1 + 1 \cdot 3 + 4 \cdot 2 + 6 \cdot 3 = 43$
- Segundo árbol: $\sum_{i=1}^{5} k_i(d_i + 1) = 5 \cdot 2 + 4 \cdot 1 + 1 \cdot 4 + 4 \cdot 3 + 6 \cdot 2 = 42$

El problema del *árbol binario de búsqueda optimal* consiste en encontrar el árbol binario de búsqueda que minimice el número total de comparaciones dada la descripción de la secuencia de accesos como una lista $[k_1, k_2, \ldots, k_n]$.

1. Diseñe un algoritmo siguiendo la técnica de programación dinámica que resuelva el problema de *árbol binario de búsqueda optimal*.

   (Ayuda: para cada subárbol que contiene las claves $c_i, c_{i+1}, \ldots, c_j$ calcular $C(i, j)$: el número mínimo de comparaciones para acceder a esos nodos.

   a) Muestre la recurrencia que modela la estrategia planteada.

   b) Escriba el pseudocódigo del algoritmo.

2. Demuestre que el problema presenta *principio de optimalidad*. En particular para este caso, es necesario probar que un árbol optimal T tiene como hijos 2 subárboles T_1 y T_2 que son optimales para las claves que contienen.

3. Muestre con un ejemplo simple que la solución propuesta presenta *solapamiento de subinstancias*.

4. Analice el orden del tiempo de ejecución del algoritmo dado y el orden del espacio requerido para su ejecución.

5. Implemente el algoritmo propuesto en lenguaje *Java*.

   El programa deberá leer la entrada desde *standard input* ó desde un archivo especificado como parámetro por línea de comando y deberá devolver la solución por *standard output*.

   En la página de la materia se proveerá código *Java* que realiza la entrada y salida, y un esqueleto de los métodos a implementar para resolver el problema. También podrán encontrar archivos de entrada y salida de ejemplo para que controlen la correctitud de sus soluciones.

   La entrada consiste de 2 líneas. En la primera línea hay un entero n. En la segunda línea hay n enteros separados por espacios $k_1\ k_2\ \ldots\ k_n$.

   La salida consiste de 2 líneas. La primera línea debe contener un entero C, el número mínimo de comparaciones que se logran con un árbol binario de búsqueda optimal. La segunda línea debe contener una representación del árbol binario optimal hallado cuyo formato se detalla a continuación.

   - Un árbol binario vacío se representa con la cadena vacía.
   - Un árbol binario con raíz c_R, subárbol izquierdo $T1$ y subárbol derecho $T2$ se representa como `(<representación T1>)R(<representación T2>)`

Entrada de ejemplo	**Salida para la entrada de ejemplo**
5 5 4 1 4 6	42 ()1())2((()3())4())5())