UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRUPO:B2

PRÁCTICA 3. Algoritmos Voraces. Star Wars: Reparto de agua

Trabajo realizado por (grado de participación):

Miguel de las Heras Fuentes → 50%

Isaac González del Pozo → 50%

Asignatura: Metodología de la Programación

Grupo: B2

Titulación: Grado en Ingeniería Informática

Fecha: 03/05/2021

Tabla de contenido

1. Enfoque backtracking utilizado

```
public static void backtracking(int[]numeros, int tamaño, int objetivo, String solucion, ArrayList<String> soluciones,int total) {
		String operacion;
		for (int i = 0; i < tamaño; i++) {
			if (total==objetivo)
				soluciones.add(solucion);

			for (int j = i+1; j < tamaño; j++) {
				for (int k = 0;  k < OPERATIONS.length; k++) {
					int resultado= OPERATIONS[k].operacion(numeros[i], numeros[j]);

					if (resultado!=0) {
						int guardari= numeros[i], guardarj=numeros[j];
						numeros[i]=resultado;
						numeros[j]=numeros[tamaño-1];

						operacion=solucion+Math.max(guardari, guardarj)+""+OPERATIONS[k].signo()+
								""+ Math.min(guardari, guardarj)+ "="+ resultado+" ; ";

						backtracking(numeros, tamaño-1, objetivo, operacion, soluciones, resultado);

						numeros[i]= guardari;
						numeros[j]=guardarj;
					}
				}
			}
		}
	}
```

Como vemos en la imagen, nuestro algoritmo de backtracking consta de un solo método al que se le pasan los siguientes parámetros:

- Números: corresponde al array de números con el que tenemos que hacer las combinaciones en esa etapa
- Tamaño: corresponde a la cantidad de números con los que tenemos que hacer combinaciones en esa etapa y que marcarán el numero de iteraciones de los dos primeros bucles for
- Objetivo: variable int que indica cual es el número que debemos obtener
- Solución: String que guarda la solución parcial obtenida hasta el momento en esa rama del árbol
- Soluciones: ArrayList de las soluciones que hemos obtenido y que vamos actualizando cada vez que se cumple el caso base.

Como podemos observar, nuestro algoritmo se basa en 3 bucles “for” de los cuales, los dos primeros nos permitirán hacer las combinaciones con los números del vector que le hemos pasado como parámetro al método. El tercer bucle for nos permitirá asignarle a cada combinación de números las 4 operaciones posibles(+,-,*,/) llamando al método operación de la interfaz operaciones. Si ese método un resultado igual a 0, significara que esa operación no se puede llevar a cabo y por tanto se descartará esa posible solución. En caso contrario, guardaremos el valor contenido en la posición “i” del array y en la posición “j” obtenidos de los dos primeros bucles. Estas posiciones las cambiaremos por el resultado y la última posición del vector. Este último cambio de debe a que, en la siguiente llamada recursiva, la variable tamaño habrá decrementado en una posición y por tanto si no se hiciera ese cambio, no se leería el último número del vector y no se harían las combinaciones con el mismo.

Una vez realizados estos cambios, le asignaremos a la variable local operación, la solución recogida hasta el momento, más la operación realizada en esta llamada, junto a los operandos y el resultado.

A continuación, realizaremos una llamada recursiva al método backtracking para seguir buscando la solución en esta rama del árbol. Una vez ha finalizado esta llamada, volveremos a dejar el array como estaba al principio con las posiciones que habíamos guardado anteriormente. En caso de que en una llamada se encuentre la solución, se ejecutará el caso base en el que se añadirá la solución obtenida en el ArrayList de soluciones.

Una vez obtenidas todas las soluciones, llamaremos a método ordenarImprimir para ordenar la lista de soluciones de menor a mayor numero de operaciones realizadas para obtener la solución. Finalmente, si imprimirá toda la lista.

Algunos de los elementos de nuestra estrategia backtracking son:

- Solución parcial: corresponde a la variable solución que se va actualizando en cada llamada recursiva
- Test de fracaso: En caso de que la llamada al método operación de la interfaz operación devuelva un resultado=0, significará que por esa rama no se encontrará un solución válida y por tanto se descarta.
- Restricciones explicitas: No se pueden utilizar valores con decimales ni negativos.

2. Complejidad teórica del algoritmo

T(n)= 4 * (n) * (n-1) * T(n-1)

T(n) - 4 * (n) *(n-1) * T(n-1)

T_n - 4 * (n) * (n-1) * T_{n-1}

$T_n = X^n$

X^n – 4 * n * (n-1) * X^{n-1}

X^{n-1} * (x – 4 * n * (n-1)) = 0

X^{n-1} = 0 Trivial

X - $4n^2$ – 4n = 0

$t_n = C_1 * (4n^2 - 4n)^n \quad X_1 = 1, \quad X_2 = 0$

$t_n = C_1 * (n)^n$

∈ O $(\mathbf{n})^{\boldsymbol{n}}$

3. ¿Es el algoritmo óptimo en cuanto al resultado obtenido?

Podemos decir que se trata de una algoritmo óptimo porque se combinan todos los números de todas la formas posibles. Esto permite encontrar todas las soluciones al problema; tanto las más optimas como las que no lo son. Una vez encontradas todas las soluciones podemos elegir la que más nos convenga.