Ejercicio 7

Rojo Mata Daniel

- **Algoritmo:** Realizar Operación.
- **Entrada:** Un operador aritmético, `operador`, y dos operandos, `operando1` y `operando2`.
- **Salida:** El resultado de la operación (`operando1 operando operando2`).

```java
public int RealizarOperacion(char operador, int operando1, int operando2) {
    int resultado = 0;

    switch (operador) {
        case '+':
            resultado = operando1 + operando2;
            break;
        case '-':
            resultado = operando1 - operando2;
            break;
        case '*':
            resultado = operando1 * operando2;
            break;
        case '/':
            if (operando2 != 0) {
                resultado = operando1 / operando2;
            } else {
                System.out.println("Error: División por cero.");
            }
```

```
20                  break;
21              default:
22                  System.out.println("Error: Operador no definido.");
23                  break;
24          }
25
26          return resultado;
27      }
```

Descripción del algoritmo

El algoritmo anterior acepta tres parámetros: un carácter que indica la operación a realizar y dos números enteros sobre los cuales se aplicará dicha operación. Utiliza una estructura de control tipo `switch` para determinar qué operación se llevará a cabo. Las operaciones permitidas son suma, resta, multiplicación y división.
Para el caso de la división se revisa el caso en que no haya una división entre cero, en tal caso se arroja un mensaje de error.

La salida del programa es finita pues solo se están utilizando operaciones elementales, además, se regresa lo esperado; el resultado de la operación correspondiente, haciendo que el algoritmo sea correcto.

Obtención de $T(n)$

En cada bloque `case` se realizan 3 operaciones elementales. Tómese el caso del primer bloque, líneas $5-7$. Se realiza una operación, a saber `operando1 + operando2`, posterior a esto, se realiza una asignación, `resultado = operando1 + operando2`.

Hasta este punto hay dos operaciones elementales, sin embargo, switch hace el comparativo del argumento `operador` con los operadores `+, -, *, /`.
Esto añade una operación más por comparación, así, en general hay 3 operaciones elementales en los bloques en que se compara con `+, -, *`.

En el bloque de división, línea $15-20$, hay un comparativo adicional al hacer `operando2 !=0`.

Mientras que en `default` se hace otro comparativo. Finalmente se tiene una operación elemental de asignación en la línea 2.

Al contar lo anterior se llega al tiempo de ejecución $T(n) = 3 * 3 + 4 + 1 + 1$. Esto es:

$$T_{operacion}(n) = 16 \tag{1}$$

No se utiliza espacio extra en memoria, por lo que:

$$M_{operacion}(n) = 0 \tag{2}$$

De esta manera; $T_{operacion}(n), M_{operacion}(n) \in O(1)$.

- **Algoritmo:** contieneElemento
- **Entrada:** Un arreglo de caracteres denominado `arreglo` y un caracter denominado `elemento`.
- **Salida:** `true` si `elemento` se encuentra en `arreglo`, `false` en caso contrario.

```
1     public static boolean contieneElemento(char[] arreglo, char elemento) {
2         for (char valor : arreglo) {
3             if (valor == elemento) {
4                 return true;
5             }
6         }
7         return false;
8     }
```

Descripción del algoritmo

El algoritmo anterior recibe como entrada un arreglo de tipo `char` y un elemento a buscar en el arreglo. Se compara cada valor del arreglo con el elemento introducido y en caso de que dicho elemento coincida con un valor del arreglo se devuelve `true`, de lo contrario retorna `false`.

Obtención de $T(n)$

Se realiza un ciclo `for` para poder comparar cada elemento del `arreglo` con `elemento`. Nótese que dentro del bloque `if` se hace solo una operación elemental, es por ello, que a lo más se hacen n comparaciones, siendo n la longitud del `arreglo`. Es por ello que:

$$T_{contiene}(n) = n + b \tag{3}$$

Siendo b una constante que cuenta lo dentro del *for-each*.

No se utiliza memoria extra, por ello:

$$M_{contiene}(n) = 0 \tag{4}$$

- **Algoritmo:** EvaluarExpresionAritmetica.
- **Entrada:** Un arreglo `E` que contiene la expresión aritmética a evaluar en notación `postfija`.
- **Salida:** El resultado de la evaluación o `Exception` si la expresión no es válida.

```
1     public int EvaluarExpresionAritmetica(char[] E) throws Exception {
2
3         int n = E.length;
4         Pila<Integer> PilaOperandos = new Pila<>();
5         char[] Digitos = { '1', '2', '3', '4', '5', '6', '7', '8', '9' , '0'};
6
7         for (int i = 0; i < n; i++) {
8             if (contieneElemento(Digitos, E[i])) {
9                 int valorNumerico = Character.getNumericValue(E[i]);
10                PilaOperandos.apilar(valorNumerico);
11            } else {
12                int operando2 = PilaOperandos.darElementoCima();
13                PilaOperandos.desapilar();
```

```
14                  int operando1 = PilaOperandos.darElementoCima();
15                  PilaOperandos.desapilar();
16                  int resultado = RealizarOperacion(E[i], operando1, operando2);
17                  PilaOperandos.apilar(resultado);
18              }
19          }
20
21          return PilaOperandos.darElementoCima();
22      }
```

Descripción del algoritmo

Se define `n` como la longitud de `E`. Como primer paso se crea una pila vacía llamada `PilaOperandos` en donde sus elementos son de tipo `Integer` (enteros).
Se crea un arreglo de caracteres llamado `Digitos` en donde sus elementos son los caracteres que representan a los números desde el 0 hasta el 9.

Se realiza un ciclo `for` teniendo un contador `i` desde 0 hasta $n-1$ en donde se consideran los siguientes caso en un bloque `if-else`.

- Bloque `if` $8-10$. El elemento `E[i]` pertenece al arreglo `Digitos`.

  Esta comparación se realiza al utilizar el algoritmo `contieneElementos`.

  En este caso se convierte en un valor numérico al elemento `E[i]` (línea 9) y se agrega a la cima de la pila `PilaOperandos`.

- Bloque `else` $11-18$. El elemento `E[i]` no pertenece al arreglo `Digitos`.

  La condición anterior garantiza que `E[i]` es un operando de la forma `+, -, *, /`. En esta situación, comienza a operarse con los valores que hay dentro de la pila. Se obtiene el valor de la cima y se asigna a la variable `operando2` para después quitarlo de la pila aplicando el método `desapilar()`.

  Se realiza lo propio pero ahora con el siguiente valor que se encuentra en la cima, esto es; a la variable `operando1` se asigna la cima de la pila y después se remueve ésta con el método `desapilar()`.

Se llama al método `RealizarOperacion` cuyos argumentos son `operando2`, `operando1` y `E[i]`, en donde éste último hace el papel de `operador`. El resultado de la operación se almacena, con el método `apilar`, en la cima de la pila para posteriormente obtnerlo adecuadamente con el método `darElementoCima()`.

Corrección del algoritmo

Se expresa un invariante de ciclo en forma de *Lema* para poder demostrar la corrección del algoritmo.

LEMA: Después de cada iteración del bucle, la pila `PilaOperandos` *contiene el resultado de las operaciones evaluadas hasta el índice i de la expresión aritmética* `E`.

- **Inicialización:** Para este caso se tiene `i = 0` lo que quiere decir que no se ha entrado al ciclo, esto quiere decir que `PilaOperandos` contiene el resultado hasta el momento, es decir, no contiene nada pues no se ha efectuado alguna operación. Es así que el invariante se mantiene.

- **Mantenimiento:** Se demuestra por inducción matemática, teniendo como caso base la **Inicialización**.

  Demostración. Hipótesis de inducción: El invariante de ciclo es cierto para la iteración `k`. Siguiendo el invariante, la suposición anterior implica que después de la iteración `k` la pila `PilaOperandos` contiene el resultado de las operaciones realizadas hasta este índice de la expresión `E`.

  Se pretende demostrar que después de la iteración `k`, es decir, que en la iteración `k+1` el invariante es cierto.

  Si se tiene la iteración `k+1` hay dos posibles situaciones que pueden ocurrir:

  1. `E[k+1]` $\in$ `Digitos`. En esta situación `E[k+1]` es casteado a su valor numérico correspondiente y añadido a la cima de la pila. Por hipótesis de inducción, la pila `PilaOperandos` contiene los resultado de las operaciones realizadas hasta `E[k]`, pero el agregar `E[k+1]` a la cima no afecta lo realizado anteriormente, pues solo se está almacenando un valor entero a la pila. Esto implica que el invariante se mantiene.

2. `E[k+1]` $\notin$ `Digitos`. En este caso `E[k+1]` es alguno de los siguientes operandos `+, -, *, /`.

   Se desapilan los dos operandos superiores de la pila y se realiza la operación correspondiente. Luego, el resultado se apila de nuevo en `PilaOperandos`. Por la hipótesis de inducción, la pila contiene el resultado de las operaciones evaluadas hasta `i = k` y al realizar una operación adicional, evaluada correctamente, el invariante de ciclo se mantiene, pues se ha añadido el resultado adecuado a la cima de la pila. Esto implica que el invariante se mantiene.

Así, al considerar los casos anteriores se concluye que el invariante se cumple en esta etapa, pues la pila guarda el resultado de la operación hasta el índice `i=k`. □

- **Finalización:** Cuando el bucle termina (es decir, cuando `i = n - 1`, donde `n` es la longitud de la expresión aritmética `E`), el invariante de ciclo implica que la pila `PilaOperandos` contiene el resultado de todas las operaciones en la expresión y al salir del bucle se devuelve la cima de la pila que es justamente el resultado que se busca.

Al cumplirse el invariante se garantiza que el algoritmo es correcto.

Obtención de $T(n)$

Se ejecutan métodos de la clase `Pila` para la ejecución del algoritmo, sin embargo, el tiempo de ejecución de éstos es en su mayoría constante. Se enlistan los tiempos de ejecución de estos métodos, y de otros, de manera general[1].

1. `apilar`: `C1`
2. `desapilar`: `C2`
3. `darElementoCima`: `C3`
4. `Crear una pila (línea 4):` D_0
5. `Crear un arreglo (línea 4):` D_1

[1]Se está suponiendo que el crear objetos y castear toma tiempo de ejecución, si esto no ocurriese las constantes que se definen para estos métodos pueden ser sustituidas por 0 en el resultado que se muestra al final

6. `Castear (línea 9):` D_2

Con los tiempos anteriores se procede a calcular lo requerido.

En el bloque `if` se tiene, por orden, $T_{contiene} + 1 + D_2 + C_1$.

En el bloque `else`: $C_3 + C_2 + C_3 + C_2 + T_{operacion} + 1 + C_1$ [2]

Al interior del ciclo `for` se realizan `n` iteraciones, es por ello que dentro del ciclo se tiene:

$$n\Big(T_{contiene} + 1 + D_2 + C_1 + C_3 + C_2 + C_3 + C_2 + T_{operacion} + 1 + C_1\Big)$$

Agrupando las constantes C_i y los 1 en una constante C se sigue:

$$n\Big(T_{contiene} + T_{operacion} + D_2 + C\Big)$$

Ahora, se suma lo que contribuyen las líneas $3, 4, 5, 7$ y 21, obteniendo $T(n)$.

$$T(n) = n\Big(T_{contiene} + T_{operacion} + D_2 + C\Big) + D_0 + D_1 + 4 + C_3$$

Reescribiendo las constantes restantes en una constante C' se obtiene:

$$T(n) = n\Big(T_{contiene} + T_{operacion} + D_2 + C\Big) + D_0 + D_1 + C'$$

Sustituyendo 3 y 1 se llega a lo siguiente.

$$T(n) = n + \Big(n + b + 16 + D_2 + C\Big) + D_0 + D_1 + C'$$

Reagrupando las constantes agregadas en C se obtiene el tiempo de ejecución del algoritmo `EvaluarExpresionAritmetica`.

$$T(n) = n\Big(n + D_2 + C\Big) + D_0 + D_1 + C'$$

Luego, la complejidad en espacio es:

$$M(n) = M_{contiene} + M_{operacion} + M_{CrearPila} + M_{CrearArreglo}$$

[2] Al llamar al método `RealizarOperacion` se toma como argumento `E[i]`, es por eso que contribuye con $T_{operacion} + 1$, análogo en el bloque `if` con el método `contieneElemento`

Usando lo obtenido en 4 y 2

$$M(n) = M_{CrearPila} + M_{CrearArreglo}$$

Si se hace el supuesto de que se crea una pila de al menos n elementos (que es la longitud de `E`) entonces se tiene:

$$M(n) = n + M_{CrearArreglo}$$

Siendo $M_{CrearArreglo}$ la memoria que ocupa crear el arreglo `Digitos` (que por lo general está relacionado con la longitud de dicho arreglo).

Resumiendo lo obtenido:

$$T(n) = n\Big(n + D_2 + C\Big) + D_0 + D_1 + C'$$

$$M(n) = n + M_{CrearArreglo}$$

De esta manera se puede apreciar que $T(n) \in O\big(n^2\big)$, mientras que $M(n) \in O\big(n^m\big)$, con $1 \leq m \leq 2$. [3]

[3]Se pone una cota superior como $m = 2$ en el caso de crear el arreglo, dependerá de lo que ocupe en memoria el índice que se escoja, lo que es un hecho seguro es de que al menos, $M(n)$ es lineal.