Proyecto Final

Julián Andrés Bermúdez Valderrama, Juan Camilo Lara Navarro
{ja.bermudez10, jc.lara10}@uniandes.edu.co
{201519648, 201424726}
Universidad de los Andes, Bogotá, Colombia

Problema A

1. Algoritmo de solución.

Caso 1:

El problema se abordó mediante el reconocimiento y almacenamiento de las posiciones cuyo valor en el conjunto (arreglo) era igual a cero (0). Esto se hizo con el fin de facilitar la búsqueda del sub-arreglo de mayor tamaño.

Es así como, por ejemplo, para el caso del arreglo $Z = [12, 0, -3, 0, 0, 1, 2, 3, 4, 0, 4, 5, 23, 0, 100]$, las posiciones de los ceros apartadas en otro arreglo serían: $ceros = [1, 3, 4, 9, 13]$.

De este modo, se define el rango de búsqueda, el cual es dado por la entrada c. Sí se sabe que en el sub-arreglo a encontrar debe haber a lo sumo c ceros, se puede llegar a un acercamiento a lo que sería el tamaño del sub-arreglo en Z, mediante el establecimiento de una cota mínima dada por $ceros[i]$ y una cota máxima dada por $ceros[j]$ en donde se cumple qué, $0 \leq i < j < |ceros|$. Inicialmente $i = 0$, por ende, el primer índice en Z será $ceros[0]$. Por otra parte, dado que $ceros$ es un arreglo basado en cero (zero-based, es decir que su primer elemento está en la posición 0), se concluye que el índice del $c -$ $ésimo$ cero en Z, estará en $ceros[c - 1]$.

A continuación, se muestra una gráfica para aclarar el concepto.

Con la entrada: $15\ 3\ 12\ 0\ -3\ 0\ 0\ 1\ 2\ 3\ 4\ 0\ 4\ 5\ 23\ 0\ 100$, se tiene entonces:

$n = 15$, la dimensión del arreglo.
$c = 3$, el máximo número de ceros en un sub-arreglo.

Arreglo (Z)															
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	12	0	-3	0	0	1	2	3	4	0	4	5	23	0	100

Arreglo de entrada (Z)

Arreglo de indices de cero en Z (ceros)					
Indice	0	1	2	3	4
Indice de cero en Z	1	3	4	9	13

Arreglo de índices de cero en Z

Ahora, se establece la cota mínima y la cota máxima:

Se recuperan los índices de Z en $ceros$, así:

$ceroInicio = ceros[0]$
$ceroFin = ceros[c - 1]$

Arreglo de indices de cero en Z (ceros)					
	v		v		
Indice	0	1	2	3	4
Indice de cero en Z	1	3	4	9	13

Índices para establecer búsqueda en Z

Arreglo (Z)															
		v			v										
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	12	0	-3	0	0	1	2	3	4	0	4	5	23	0	100

Rango de búsqueda básico

Si bien en el rango de búsqueda establecido se cumple con la condición del máximo número de ceros posibles c, aun no se cumple con el tamaño máximo a nivel local en este punto de la ejecución.

Se necesita entonces incluir aquellos elementos distintos de cero (0), ubicados hacia los extremos de las cotas. Tomando la cota mínima como referencia (i'), se aumentará el rango del sub-arreglo moviendo a la izquierda siempre y cuando se cumpla qué $i' > 0 \wedge Z[i' - 1] \neq 0$; De igual manera, tomando la cota máxima como referencia (j'), se aumentará el rango del sub-arreglo moviendo a la derecha siempre y cuando se cumpla qué $j' < (|Z| - 1) \wedge Z[j' + 1] \neq 0$. Así:

Arreglo (Z)															
	v	v			v				v						
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	12	0	-3	0	0	1	2	3	4	0	4	5	23	0	100

Máximo sub-arreglo local
Tamaño sub-arreglo: 8

El cálculo del tamaño del arreglo es dado por el último valor de j' menos el último valor tomado por i', más uno (1). Teniendo entonces al mayor tamaño del sub-arreglo local como, $(j' - i') + 1$.

Finalmente, se procede a actualizar las cotas, tanto la mínima como la máxima, avanzando su índice al siguiente cero perteneciente en $ceros$ (siempre y cuando sea posible).

Se repite el proceso de:

- Establecer rango y/o rango de búsqueda con base en cotas.
- Expandir rango y/o rango hacia los extremos de las cotas, para encontrar el sub-arreglo.
- Actualizar el máximo tamaño encontrado hasta el momento con un tamaño mayor.

El siguiente avance completo es:

Arreglo de indices de cero en Z (ceros)					
		v		v	
Indice	0	1	2	3	4
Indice de cero en Z	1	3	4	9	13

Actualización de índices para establecer búsqueda en Z

Arreglo (Z)															
				v						v					
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	12	0	-3	0	0	1	2	3	4	0	4	5	23	0	100

Rango de búsqueda básico (Iteración 2)

Arreglo (Z)															
			v	v						v			v		
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	12	0	-3	0	0	1	2	3	4	0	4	5	23	0	100

Máximo sub-arreglo local (Iteración 2)
Tamaño sub-arreglo: 11

El máximo tamaño del sub-arreglo, es dado después de
- El Avance de las cotas.
- La expansión del rango.
- La actualización del tamaño máximo (hasta que no se pueda actualizar más).

Caso 2:

Cuando el número de ceros en el arreglo de entrada (Z) es menor o igual a c, se asume que el tamaño del sub-arreglo más largo es $|Z|$. Dado que c, es una cota superior, el problema así lo describe: *"**Problema** Encontrar la longitud del sub-arreglo más largo de a que tiene, a lo sumo, c ceros."*

Con la entrada: $15\ 3\ 12\ 0\ -3\ 1\ 8\ 1\ 2\ 3\ 4\ 0\ 4\ 5\ 23\ 14\ 100$, se tiene entonces:

$n = 15$, la dimensión del arreglo.
$c = 3$, el máximo número de ceros en un sub-arreglo.

Arreglo (Z)															
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	12	0	-3	1	8	1	2	3	4	0	4	5	23	14	100

Caso 2
Tamaño sub-arreglo: 15

Caso 3:

Cuando el número de ceros en el arreglo de entrada (Z) es igual a $|Z|$, se asume que el tamaño del sub-arreglo más largo es c. Puesto que si todos los elementos en Z, son cero (0), lo anterior se cumple.

Con la entrada: 15 3 0 0 0 0 0 0 0 0 0 0 0 0 0 , se tiene entonces:

$n = 15$, la dimensión del arreglo.
$c = 3$, el máximo número de ceros en un sub-arreglo.

Arreglo (Z)															
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Valor en arreglo	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Caso 3
Tamaño sub-arreglo: 3

- Métodos:
    - Principales:

m1. ConstruirArreglo

Ctx: $input[0..n), longitud : nat$
Pre: $true$
Post: $construirArreglo(input, longitud)$

```
fun construirArreglo(input array [0, n) of String, longitud : nat) ret arregloDesdeDos
: array [0, n) of String
      arregloDesdeDos array [0, n) of String = new array [0, longitud) of String;
      var i : nat;
      i := 2;

      do (i < longitud) →
            arregloDesdeDos[i - 2] := input[i];
            i := i+1;
      od

      return arregloDesdeDos;
}
```

m2. ObtenerIndicesCeros

Ctx: $arreglo[0..n)$
Pre: $true$
Post: $obtenerIndicesCeros(input, longitud)$

```
fun obtenerIndicesCeros(arreglo array [0, n) of String) ret indicesCeros : array [0,
n) of nat
      var cantCeros : nat;
      cantCeros := contarCeros(arreglo);
      indicesCeros array [0, n) of nat = new array [0, cantCeros) of nat;

      var i : nat;
      var j : nat;
      i, j := 0, 0;

      do ((i < |arreglo|) ∧ (j < |indicesCeros|)) →
            if (arreglo[i] = "0") →
                  indicesCeros[j] := i;
                  j := j+1;
            fi
            i := i+1;
      od

      return indicesCeros;
```

m3. CalcularMayorSubarreglo

Ctx: $arreglo[0..n)$
Pre: $true$
Post: $obtenerIndicesCeros(input, longitud)$

```
fun calcularMayorSubarreglo(arreglo array [0, n) of String, indicesCeros array [0, n) of nat, n : nat, c : nat) ret
maxLongitudSubarreglo : nat

      maxLongitudSubarreglo := 0;

      var indiceCeroInicio : nat;
      var indiceCeroInicio : nat;

      indiceCeroInicio, indiceCesimoCero := 0, c - 1;

      do ((indiceCeroInicio < |indicesCeros|) && (indiceCesimoCero < |indicesCeros|)) →

            ceroInicio := indicesCeros[indiceCeroInicio];
            cEsimoCero := indicesCeros[indiceCesimoCero];

            indiceIzq := ceroInicio;
            indiceDer := cEsimoCero;

            while (indiceIzq > 0 ∧ !(arreglo[indiceIzq - 1] = ("0"))) →
                  indiceIzq := indiceIzq-1;
            elihw

            while (indiceDer < (n - 1) ∧ !(arreglo[indiceDer + 1] = ("0"))) →
                  indiceDer := indiceDer+1;
            elihw


            if (((indiceDer - indiceIzq) + 1) > maxLongitudSubarreglo) →
                  maxLongitudSubarreglo := (indiceDer - indiceIzq) + 1;
            fi

            indiceCeroInicio, indiceCesimoCero := indiceCeroInicio+1, indiceCesimoCero+1
      od

      return maxLongitudSubarreglo;
```

- Auxiliares:

m1. ContarCeros

Este método cuenta la cantidad de ceros (0) presentes en el arreglo de entrada.

Ctx: $arreglo[0..n)$
Pre: $true$
Post: $contarCeros(nums)$

```
fun contarCeros(arreglo array [0, n) of String) ret contadorCeros : nat
      var i : nat;
      var contadorCeros : nat;

      i, contadorCeros := 0, 0;

      do (i < n) →
            if (arreglo[i] = "0") →
                  contadorCeros := contadorCeros + 1;
            fi
            i := i+1;
      od

      return contadorCeros;
}
```

2. Análisis de complejidades espacial y temporal.

- Complejidad temporal
    - CalcularMayorSubarreglo

En el peor caso, se puede realizar k permutaciones.

Operación	Veces
Asignación (:=)	$2n^2 + 5n + 1$
Comparación (>=, <=)	$4n^2 + 3n$

Complejidad: $O(n^2) = n(6n) + 8n + 1$

- Complejidad espacial

La complejidad espacial a diferencia de la temporal es $S(n)$, puesto que se trabaja sobre el arreglo solamente.

3. Comentarios finales.

Utilizando los casos de prueba suministrados en el documento y otros casos planteados por nosotros, nuestra solución demora tiempos aceptables en resolver el problema.

Una de las decisiones importantes fue definir los posibles casos de reconocimiento del sub-arreglo más largo (los cuales se pueden ver en el literal 1. Algoritmo de solución.). Reconocer los 3 casos allí mencionados, permitió facilitar el cálculo de la respuesta, en busca de optimizar la respuesta del programa.