Estructuras de Datos 2 - ST0247
Segundo Parcial

Nombre .....
Departamento de Informática y Sistemas
Universidad EAFIT

Octubre 25 de 2018

1 Divide y Vencerás 30%

Considere una secuencia de números $a : a_1, a_2, a_3, ..., a_n$. Para esta secuencia siempre se cumple que $a_1 \leq a_2 \leq ... \leq a_n$, es decir, esta secuencia está ordenada de menor a mayor (siendo a_1 el menor elemento de a y a_n el mayor elemento de a). Ahora, considere un entero z. Queremos encontrar, en la secuencia, **el mayor numero que es menor o igual a z**. La solución a este problema simplemente es hacer **búsqueda binaria** en la secuencia, teniendo en cuenta, en la búsqueda, que el elemento actual sea menor o igual al elemento examinado. Ayudanos a terminar el código.

- **Ejemplo**: Sea $a = \{3, 7, 11, 12, 23, 24, 27, 77, 178\}$ y $z = 45$. La respuesta es 27.

```
1   int bus(int[] a, int iz, int de, int z){
2     if(iz>de){
3       return -1;        }
4     if(a[de]>=z){
5       return a[de];     }
6     int mitad=(iz+de)/2;
7     if(a[mitad]==z){
8     return .........; }
9     if(mitad>0){
10      if(a[mitad-1]<=z && z<a[mitad]){
11        return a[mitad - 1];      }    }
12    if(z<a[mitad]){
13      return bus(a,iz,mitad-1,z); }
14    //else
15    return bus(...., ...., ...., ....);
16  }
17  public int bus(int[] a, int z) {
18     return bus(a, 0, a.length - 1, z);
19  }
```

(a) (10%) ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior?

(i) $T(n) = 2.T(n/2) + C$ que es $O(n)$
(ii) $T(n) = 2.T(n/2) + Cn$ que es $O(n \times \log_2 n)$
(iii) $T(n) = T(n/2) + C$ que es $O(\log_2 n)$
(iv) $T(n) = 4.T(n/2) + C$ que es $O(n^2)$

(b) (10%) Complete la línea 8 .....

(c) (10%) Complete la línea 15 ...., ..., ..., ...

2 Voraces 40%

Se está preparando la llegada del Rey a su castillo. Por seguridad se han ubicado N puestos de guardia, cada uno a $10m$ de distancia. Ya se han ubicado algunos guardias en posiciones estratégicas. Se sabe que el guardia en la posición i protegerá al guardia en la posición j sí $|j - i| \leq K$. En la posición 0 y en la posición $N - 1$ siempre hay guardias, pero hay algunas posiciones que aún no se han cubierto.

La guardia real quiere saber cuál es la mínima cantidad de guardias que tiene que contratar, de tal manera que todos los guardias siempre se estén cuidando mutuamente y ¡haya más seguridad! Las posiciones se entregan como un arreglo A donde la posición i del arreglo contiene el número $i + 1$ si la posición i está ocupada por un guardia o un 0 si no hay un guardia en esa posición.

Ejemplos: Para $x_1 = \{1, 2, 0, 0, 5, 0, 0, 8\}$, $K_1 = 2$, la respuesta sería 2. Para $x_2 = \{1, 0, 3, 4\}$, $K_2 = 2$, la respuesta sería 0. Para $x_3 = \{1, 0, 0, 0, 5\}$, $K_3 = 2$, la respuesta sería 1. Para el primer ejemplo, sería optimo ubicar guardias en las posiciones 2 y 6. Así, todos los guardias se protegerán mutuamente.

```
1  int solucion(int[]x, int K){
2    int last = 0; //ultimo guardia
3    int res = 0; //respuesta
4    int n = x.length;
5    for (int i = 0; i < n; ++i) {
6      if (x[i] == .....) last = i;
7      if (i - last == K) {
8        res = ..............;
9        last = ..............;
10     }
11   };
12   return res;
13 }
```

(a) (10%) Complete la línea 6 .....

(b) (10%) Complete la línea 8 .....

(c) (10%) Complete la línea 9 .....

(d) (10%) Determine la salida para $x_t = \{1, 0, 0, 4, 0, 0, 0, 0, 0, 0, 11\}$, $K_t = 3$: .....

3 Programación Dinámica 40%

Dados dos números no negativos n y k, encuentre el numero de formas de que la suma de k enteros no negativos sume exactamente n. Por ejemplo, si $n = 5$ y $k = 2$, hay exactamente $f(5, 2) = 6$ formas de sumar $n = 6$ con $k = 2$ números: $0 + 5 = 5$, $1 + 4 = 5$, $2 + 3 = 5$, $3 + 2 = 5$, $4 + 1 = 5$, $5 + 0 = 5$. La función $f(n, k)$ describe recursivamente ese número de formas de sumar:

$$f(n,k) = \begin{cases} 1, & \text{si } k = 1. \\ \sum_{i=0}^{n} f(n-i, k-1), & \text{si } k > 1. \end{cases} \quad (1)$$

Y a continuación está la versión de esa función, utilizando programación dinámica *bottom-up*:

```
1  int sol(int n, int k, int[][] f){
2    if(k == 1) return 1;
3    if(n < 0 | k < 0) return 0;
4    if(f[n][k] != -1) return f[n][k];
5    int formas = 0;
6    for(int i = 0; i < n + 1; ++i){
7      formas =formas + ..............;
8    }
9    ........... = formas;
10   return formas;
11 }
12 public int sol(int n, int k){
13   int[][] f = new int[n+1][k+1];
14   for(int i = 0; i < n + 1; i++){
15     for(int j = 0; j < k + 1; j++){
16       f[i][j] = -1;
17     }
18   }
19   return ...........;
20 }
```

a) (10%) ¿Cuál es la complejidad asintótica, para el peor de los casos, del algoritmo anterior?

   (i) $O(n + k)$

   (ii) $O(n^2 + k)$

   (iii) $O(n^2 \times k)$

   (iv) $O(n \times k)$

b) (10%) Complete la línea 7 .....

c) (10%) Complete la línea 9 .....

d) (10%) Complete la línea 19 .....