Estructuras de Datos 2 - ST0247
Segundo Parcial

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

Mayo 10 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 siguiente 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 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 $O(n \times \log n)$
(iii) $T(n) = T(n/2) + C$ que es $O(\log n)$
(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 Programación Dinámica 40%

Dado un conjunto de enteros $a = a_1, a_2, ..., a_n$, queremos encontrar su **subsecuencia creciente máxima**. La subsecuencia creciente máxima de una secuencia, es una subsecuencia de esta secuencia cuyos elementos están ordenados y esta subsecuencia es tan larga como sea posible. Los elementos de esta subsecuencia no necesariamente tienen que ser contiguos en la secuencia original. Este problema tiene aplicaciones en bioinformática para analisis de nicho ecológico y para filogenética. **Nota:** En la otra página hay algunos ejemplos.

En este problema estamos interesados en encontrar el tamaño de la subsecuencia creciente máxima. La ecuación recursiva está definida de esta forma:

$$scm(i) = \begin{cases} 1 + max(scm(j)) & \text{donde } 0 < j < i \text{ y } a_j < a_i \\ 1 & \text{si no existe tal valor } j \end{cases}$$

El siguiente código hace el trabajo, utilizando programación dinámica, pero faltan algunas líneas, complétalas por favor.

```
1  int scm(int arr[]) {
2    int n = arr.length;
3    int scm[] = new int[n];
4    int i,j,max = 0;
5    /* Inicializar la tabla scm */
6    for ( i = 0; i < n; i++ ) {
7         ...........; }
8    /* Calcular usando la tabla scm*/
9    for ( i = 1; i < n; i++ ) {
10        for ( j = 0; j < i; j++ ) {
11          if ( arr[i] > arr[j] && scm[i]
                < scm[j] + 1) {
12            ...........; } } }
13   /* El maximo en la tabla scm */
14   for ( i = 0; i < n; i++ ) {
15        if ( max < scm[i] ) {
16            ...........;   } }
17   return max;
18 }
```

Veamos los siguientes **ejemplos** para que lo entendamos mejor.

- **Ejemplo 1:** Sea $a = \{7, 4, 8, 1, 3, 9, 2, 6, 10\}$. La subsecuencia creciente máxima es **{1, 2, 6, 10}**. Nota que **{1, 3, 6, 10}** es otra subsecuencia creciente máxima. En todo caso, la longitud de la secuencia creciente máxima es 4.

- **Ejemplo 2:** Sea $a = \{4, 2, 1, 9, 3, 4, 10, 2, 1\}$. La subsecuencia creciente máxima es **{1, 3, 4, 10}**. La longitud de la secuencia creciente máxima es 4.

(a) (10%) Complete la línea 7 .....

(b) (10%) Complete la línea 12 .....

(c) (10%) Complete la línea 16 .....

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

(i) $O(1)$

(ii) $O(n)$

(iii) $O(n^2)$

(iv) $O(n \log n)$

3 Programación Dinámica 30%

El **algoritmo de Floyd-Warshall** encuentra la distancia mínima entre cada par de vértices i, j de un grafo con pesos. Este algoritmo muchas veces es usado para determinar la clausura transitiva de un grafo, que es útil en Lenguajes Formales y Compiladores. Dada la representación usando **matrices de adyacencia** g de un grafo, donde la posición i, j de la matriz es ∞ si el vértice i no está conectado con el vértice j o la posición i, j es un entero $a_{i,j}$ si existe una conexión entre el vértice i y j, queremos encontrar la distancia mínima entre el vértice u y el vértice v. Ayúdanos a completar el siguiente código:

```
1  int calcular(int[][]g,int u,int v){
2    int n = g.length;
3    int[][] d = new int[n][n];
4    for(int i = 0; i < n; ++i){
5    for(int j = 0; j < n; ++j){
6      d[i][j] = g[i][j];
7    }
8    }
9    for(int k = 0; k < n; ++k){
10   for(int i = 0; i < n; ++i){
11     for(int j = 0; j < n; ++j){
12       int ni = .......;
13       int nj = .......;
14       int nk = .......;
15       int res = Math.min(ni, nj + nk);
16       d[i][j] = res;
17     }
18   }
19   }
20   return d[u][v];
21 }
```

(a) (10%) Completa la línea 12 .....

(b) (10%) Completa la línea 13 .....

(c) (10%) Completa la línea 14 .....

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