Estructuras de Datos 2 - ST0247
Examen Parcial 1

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

Septiembre 13 de 2018

1 Voraces 20%

Durante la clase de Estructuras de Datos 2, el profesor propuso el siguiente problema para que se resolviera en parejas. Te daban n números $\{x_1, x_2, x_3, ..., x_n\}$ y un entero $k, k \leq n$. La tarea era encontrar un subconjunto de exactamente k números cuya suma fuera mínima. Sin embargo, para que el problema fuera más entretenido, el profesor prometió un bonus a la pareja que escribiera un algoritmo con una complejidad no superior a $O(n \log n)$. Para esto, las parejas 1 y 2 propusieron lo siguiente, respectivamente:

1. Es imposible encontrar un algoritmo que funcione en complejidad $O(n \log n)$ o inferior porque es necesario intentar cada subconjunto de k números para determinar cual de estos tiene suma mínima.

2. Es posible encontrar un algoritmo que funcione con complejidad $O(n \log n)$ o inferior porque, si ordenamos los elementos de una manera especifica, no es necesario intentar todos los posibles subconjuntos de k elementos.

a) (10%) Escoja la respuesta correcta:

   i) La pareja 1 tiene la razón, pero su justificación es falsa.

   ii) La pareja 1 tiene la razón y su justificación es verdadera.

   iii) La pareja 2 tiene la razón, pero su justificación es falsa.

   iv) La pareja 2 tiene la razon y su justificación es verdadera.

b) (10%) Describa en pocas líneas el algoritmo que usted plantea para el problema anterior. Explique la complejidad de su algoritmo.

2 Fuerza Bruta 20%

Considere el siguiente problema. Tenemos un arreglo de n números no negativos $\{x_1, x_2, x_3, ..., x_n\}$. Queremos determinar si es posible encontrar un $i, 0 \leq i < n$ de tal manera que $\sum_{0}^{j=i} x_j = \sum_{j=i+1}^{n} x_j$. Ayúdanos a resolver este problema. Como un ejemplo, para $x = \{3, 4, 7, 1, 8, 1, 1, 2, 2, 1\}$, la respuesta es verdadero y el $i = 3$.

```
1  boolean sol(int[] x){
2    boolean can = false;
3    int left, right;
4    right=left=0;
5    int n=x.length;
6    for(int i=0; i<n; i++){
7      left = left+x[i];
8      for(int j=.....; j<n; j++){
9        right = right+x[j];
10     }
11     can = can || (.................);
12     right = 0;
13   }
14   return can;
15 }
```

a) (10%) Complete la línea 7 .....

b) (10%) Complete la línea 10 .....

3 Backtracking 30%

El problema de la **subsecuencia común más larga** es el siguiente. Dadas dos secuencias, encontrar la longitud de la secuencia más larga presente en ambas. Una subsecuencia es una secuencia que aparece en el mismo orden relativo, pero –no necesariamente– de forma contigua. Como un ejemplo, "abc", "abg", "bdf", "aeg" y "acefg" son subsecuencias de "abcdefg". Entonces, para una cadena de longitud n, existen 2^n posibles subsecuencias. Este problema es utilizado en la implementación del comando `diff`, para comparación de archivos, disponible en sistemas Unix. También tiene muchas aplicaciones en bioinformática.

Considere los siguientes ejemplos para el problema:

- Para “ABCDGH” y “AEDFHR” es “ADH” y su longitud es 3. Se retornaría 3.
- Para “AGGTAB” y “GXTXAYB” es “GTAB” y su longitud es 4. Se retornaría 4.

Una forma de resolver este problema es usando *backtracking*, como un ejemplo, para las cadenas “AXYT” y “AYZX”, dada una función recursiva `lcs` que resuelve el problema, se obtendría el siguiente árbol (parcial) de recursión:

```
                lcs("AXYT", "AYZX")
                /                 \
    lcs("AXY", "AYZX")            lcs("AXYT", "AYZ")
-------------------------------------------------------------------
    lcs("AXY", "AYZX")
    /                \
lcs("AX", "AYZX") lcs("AXY", "AYZ")
-------------------------------------------------------------------
    lcs("AXYT", "AYZ")
    /                \
lcs("AXY", "AYZ") lcs("AXYT", "AY")
```

Al siguiente código le faltan algunas lineas, complétalas por favor.

```
1  int lcs(int i, int j, String s1, String s2){
2   if(i == 0 || j == 0){
3     return 0;
4   }
5   boolean prev = i < s1.length() && j < s2.
        length();
6   if(prev && s1.charAt(i) == s2.charAt(j)){
7       return 1 + lcs(______, _______, s1, s2);
8   }
9   int ni = lcs(i - 1, j, s1, s2);
10  int nj = lcs(i, j - 1, s1, s2);
11  return Math.max(_____, _____);
12 }
13 public int lcs(String s1, String s2){
14  return lcs(s1.length(), s2.length(), s1, s2);
15 }
```

a) (10%) Linea 7 _____, _____

b) (10%) Linea 11 _____, _____

c) (10%) Suponga que n es la suma de la longitud de las dos cadenas. El algoritmo `lcs` ejecuta, en el peor de los casos, $T(n) =$ _____ instrucciones.

Nota: En la complejidad pueden poner la ecuación de recurrencia o la solución a la ecuación de recurrencia aplicando la notación O.

4 Backtracking 30%

El problema de las N reinas consiste en tomar un tablero de ajedrez de $N \times N$ y ubicar N reinas de tal manera que ninguna reina quede amenazada. Una posible solución para $N = 8$ sería:

$$\begin{bmatrix} 0 & 0 & 0 & * & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & * & 0 \\ 0 & 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & * \\ 0 & * & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & * & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & * & 0 & 0 \end{bmatrix}$$

$$solucion = \{4, 7, 3, 8, 2, 5, 1, 6\}\ (columnas)$$

Ayúdanos a resolver el problema anterior. La solución consiste en entregar un arreglo a donde a_i es un entero que indica la columna de la fila i donde hay una reina.

```
1  void sol(int[] a, int r) {
2    int N = a.length;
3    if (...........) {
4      print(Arrays.toString(a));
5      return;
6    }
7    for (int i = 0; i < N; ++i) {
8      a[r]= ........;
9      if(place(a,r)) sol(a, .......);
10   }
11 }
12 boolean place(int[]a,int r){
13   for(int i=0;i<r;++i){
14     if(a[i]==a[r]) return false;
15     if((a[i]-a[r]) == (r-i)) return
           false;
16     if((a[r]-a[i]) == (r-i)) return
           false;
17   }
18   return true;
19 }
20 void print(int[]a){
21   int n=a.length;
22   System.out.print("[");
23   for(int i=0;i<n-1;i++){
24     System.out.print(a[i]+",");
25   }
26   System.out.println(a[n-1] +"]");
27 }
```

a) (10%) Complete la línea 3 .....

b) (10%) Complete la línea 8 .....

c) (10%) Complete la línea 9 .....