Estructuras de Datos 2 - ST0247
Examen Parcial 1

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

Septiembre 14 de 2017

Criterios de calificación

- Selección múltiple con única respuesta
  - Respuesta correcta: 100 %
  - Respuesta incorrecta: 0 %
- Completar código
  - Respuesta correcta 100 %
  - Respuesta incorrecta o vacía 0 %

NOTAS IMPORTANTES:
- Responda en la hoja de PREGUNTAS
- Marque la hoja de PREGUNTAS

1. Fuerza bruta 30 %

Un problema frecuente es buscar si una una cadena llamada **patrón** (`pat`) se encuentra dentro de otra cadena llamado **texto** (`txt`). Esto es lo que sucede en muchos programas cuando le damos *Edición, Buscar*. El objetivo es encontrar la posición en la que aparece por primera vez `pat` en `txt`.

Como un ejemplo, el patrón "atr" aparece dentro del texto "patrón" en la posición 1. Como otro ejemplo, el patrón "mat" no aparece en el texto "patrón". Cuando no aparece, el algoritmo retorna la longitud del texto, es decir, en este caso, 5.

Una forma de resolverlo es por fuerza bruta, probando todas las posibles posiciones en las que `pat` puede aparecer dentro de `txt` como se muestra a continuación:

```
01 int indexOf(String pat, String txt) {
02  int m = pat.length();
03  int n = txt.length();
04  int i, j;
05  for (i = 0, j = 0; i < n && j < m; i++) {
06   if (txt.charAt(i) == pat.charAt(j)) j++;
07   else {
08     i = i - j;
09     j = 0;
10   }
11  }
12  if (j == m) return ______  // encontrado
13  else        return ______; // no encontrado
14 }
```

(10 %) Complete la línea 12

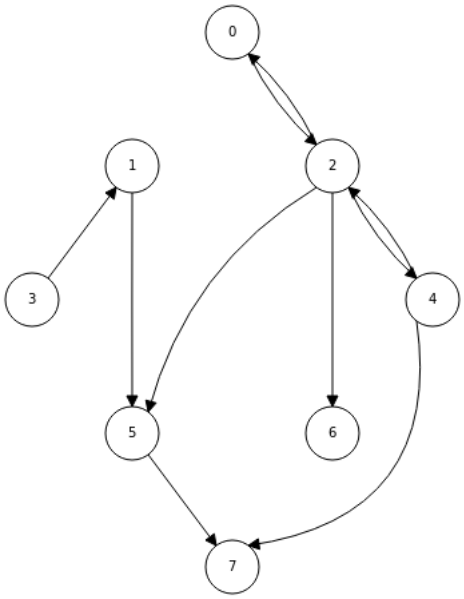
(10 %) Complete la línea 13

(10 %) ¿Cuál es la complejidad asintótica, para el peor de los casos, del algoritmo? (En términos de n y m)

O(____)
Pista: NO es $O(n!)$.

2. Implementación grafos 20 %

Considere el siguiente grafo:



A (10 %) Complete la representación de **matrices de adyacencia**. Si no hay arco, deje el espacio en blanco. NO coloque ceros.

	0	1	2	3	4	5	6	7
0			1					
1						1		
2								
3								
4								
5								
6								
7								

Pista: NO coloque ceros.

B (10 %) Complete la representación de **listas de adyacencia**. Como el grafo no tiene pesos, sólo se colocan los sucesores en la lista de adyacencia. Los sucesores o vecinos son los vértices a los que se puede llegar a partir de un arco directamente, NO transitivamente.

$0 \rightarrow [2]$
$1 \rightarrow [5]$
$2 \rightarrow$
$3 \rightarrow$
$4 \rightarrow$
$5 \rightarrow$
$6 \rightarrow$
$7 \rightarrow$

3. Recorridos de grafos 20 %

Para el grafo anterior, complete la salida que darían los siguientes algoritmos:

A (10 %) Complete el orden en que se recorren los nodos usando **búsqueda en profundidad** (en Inglés DFS) a partir de cada nodo. Si hay varias opciones de recorrer el grafo con DFS, elija siempre el vértice más pequeño.

$0 \rightarrow$
$1 \rightarrow 5 \rightarrow 7$
$2 \rightarrow$
$3 \rightarrow$
$4 \rightarrow$
$5 \rightarrow$
$6 \rightarrow$
$7 \rightarrow$

B (10 %) Complete el orden en que se recorren los nodos usando **búsqueda en amplitud** (en Inglés BFS) a partir de cada nodo. Si hay varias opciones de recorrer el grafo con BFS, elija siempre el vértice más pequeño.

$0 \rightarrow$
$1 \rightarrow 5 \rightarrow 7$
$2 \rightarrow$
$3 \rightarrow$
$4 \rightarrow$
$5 \rightarrow$
$6 \rightarrow$
$7 \rightarrow$

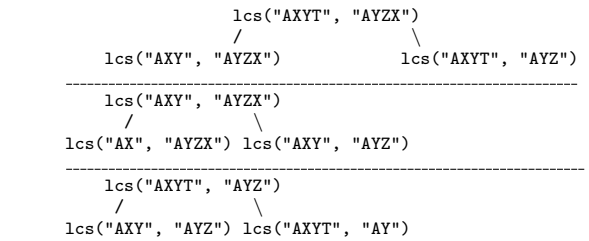
4. Backtracking 30 %

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.
- Para “AGGTAB” y “GXTXAYB” es “GTAB” y su longitud es 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:



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

```
01 private int lcs(int i, int j, String s1, String s2){
02     if(i == 0 || j == 0){
03         return 0;
04     }
05     boolean prev = i < s1.length() && j < s2.length();
06     if(prev && s1.charAt(i) == s2.charAt(j)){
07         return ______+ lcs(i - 1, j - 1, s1, s2);
08     }
09     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 }
```

(10 %) Linea 7 _____

(10 %) Linea 11 _____, _____

Y complete la complejidad por favor

(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.