Estructuras de Datos 2 - ST0247 - 032
Examen Parcial 1 (jueves)

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

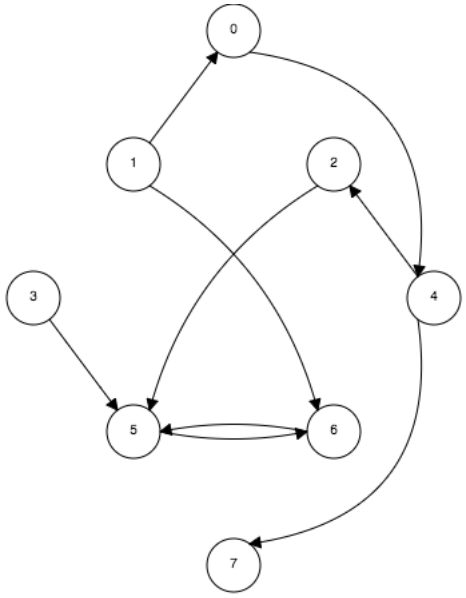
Marzo 23, 2017

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. Implementación grafos 20 %

Considere el siguiente grafo:

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

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

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.

$0 \rightarrow (4, 1)$
$1 \rightarrow$
$2 \rightarrow$
$3 \rightarrow$
$4 \rightarrow$
$5 \rightarrow$
$6 \rightarrow$
$7 \rightarrow$

2. 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.

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

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.

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

3. Fuerza bruta 20 %

La función `ordenar` es un algoritmo de ordenamiento, de menor a mayor, por fuerza bruta. Dicho algoritmo calcula todas las permutaciones posibles de un arreglo `arr` hasta encontrar una permutación donde los elementos están ordenados (es decir, cuando `estaOrdenado` retorna verdadero). Tenga en cuenta que `ordenar` imprime en la pantalla el arreglo `arr` ordenado, pero no necesariamente deja el arreglo `a` ordenado por la forma en que está diseñado el algoritmo. Aunque esto último no es deseable, ese no es el problema de este parcial.

```
static boolean estaOrdenado(int[] a) {
       for (int i = 0; i < a.length - 1; i++)
           if (a[i] > a[i + 1])
               return false;
       return true;
   }
static void cambiar(int[] arr, int i, int k){
       int t = arr[i];
       arr[i] = arr[k];
       arr[k] = t;
   }
static void ordenar(int[] arr, int k){
        for(int i = k; i < arr.length; i++){
            cambiar(arr, i, k);
            if (estaOrdenado(arr))
                System.out.println(
                    Arrays.toString(arr));
            ordenar(______, ______);
            cambiar(arr, k, i);
```

```
    }
}
```

A (10 %) Complete los espacios vacíos en el llamado recursivo del método `ordenar`

_____, _____

B (10 %) Complete la complejidad, en el peor de los casos, del método `ordenar`

O(____)

4. Backtracking 30 %

Wilkenson y Sofronio están aquí de nuevo. En esta vez han traído un juego muy interesante, en el cual Sofronio, en primer lugar, escoge un numero n ($1 \leq n \leq 20$) y, en segundo lugar, escoge tres números a, b y c ($1 \leq a \leq 9, 1 \leq b \leq 9, 1 \leq c \leq 9$). Después, Sofronio le entrega estos números a Wilkenson y Wilkenson le tiene que decir a Sofronio **la cantidad máxima de números, usando a, b y c (se puede tomar un número más de una vez), que al sumarlos den el valor n.**

Como un ejemplo, si Sofronio escoge $n = 14$ y $a = 3, b = 2, c = 7$. ¿Qué posibilidades hay de sumar 14 con a, b y c?

$7 + 7 = 14$	cantidad es 2
$7 + 3 + 2 + 2 = 14$	cantidad es 4
$3 + 3 + 3 + 3 + 2 = 14$	cantidad es 5
...	
$2 + 2 + 2 + 2 + 2 + 2 + 2 = 14$	cantidad es 7

La cantidad máxima de números es 7. Esta sería la respuesta que da Wilkenson a Sofronio. Como Wilkenson es muy astuto, ha diseñado un algoritmo para determinar la cantidad máxima de números y quiere que le ayudes a terminar su código. Asuma que hay al menos una forma de sumar n usando los números a, b y c en diferentes cantidades, incluso si algunos de los números se suman 0 veces como sucede en el ejemplo anterior.

```
1 int solucionar (int n, int a, int b, int c)
2   if (n == 0 )
3     return 0;
4   int res = solucionar(____________) + 1;
5   res = Math.max(_______,__________);
6   res = Math.max(_______,__________);
7   return res;
```

A (10 %) Complete el espacio de la línea 04

_____

B (10 %) Complete los espacios de la línea 05

_____, _____

C (10 %) Complete los espacios de la línea 06

_____, _____

5. Alg. Voraces 10 %

El problema del **agente viajero** consiste en responder la siguiente pregunta: Dada una lista de ciudades y las distancias entre cada par de ciudades, ¿cuál es la ruta posible más corta que visita a cada ciudad exactamente una vez y regresa a la ciudad de origen?

Un algoritmo voraz para solucionar este problem es el **algoritmo del vecino más cercano**. El algoritmo empieza en la primera ciudad y selecciona en cada iteración una nueva ciudad, no visitada, que sea la más cercana a la inmediamente anterior. A continuación una implementación del algoritmo del vecino más cercano que recibe como parámetro un grafo representado como matriz de adyacencia.

La persona que hizo este código es un ingeniero matemático. En Matlab los índices empiezan en 1. Por esta razón, el camino que encuentra el algoritmo empieza con la ciudad numerada con 1 y trabaja con ciclos `while` en lugar de `for`. Adicionalmente, el programador inicia los índices en 1, por ejemplo `i = 1`. Por si fuera poco, la variable `min` no era necesaria inicializarla fuera del bloque `while` y `minFlag` hubiera sido mejor declararla dentro del bloque `while`. Finalmente, el arreglo de visitados sería más eficiente haberlo hecho con tipo `boolean`. No obstante, el programa funciona en Java y tiene una complejidad de $O(n^2)$, que es la esperada para este algoritmo.

```
01 public void tsp(int adjacencyMatrix[][]) {
02  Stack<Integer> stack = new Stack<Integer>();
03  int numberOfNodes = adjacencyMatrix[1].length - 1;
04  int[] visited = new int[numberOfNodes + 1];
05  visited[1] = 1;
06  stack.push(1);
07  int element, dst = 0, i;
08  int min = Integer.MAX_VALUE;
09  boolean minFlag = false;
10  System.out.print(1 + "\t");
11  while (!stack.isEmpty()) {
12   element = stack.peek();
13   i = 1;
14   min = Integer.MAX_VALUE;
15   while (i <= numberOfNodes) {
16    if (adjacencyMatrix[element][i] > 0
17        && visited[i] == 0) {
18         if (_______> ______) {
19          min = adjacencyMatrix[element][i];
20          dst = i;
21          minFlag = true;
22         }
23    }
24    i++;
25   }
26   if (minFlag) {
27    visited[dst] = 1;
28    stack.push(dst);
29    System.out.print(dst + "\t");
30    minFlag = false;
31    continue;
32   }
33   stack.pop();
34  }
}
```

`Continue` es una palabra reservada en Java que permite terminar una iteración de un ciclo abruptamente y pasar a la siguiente iteración del ciclo.

A (10 %) Complete el espacio en la línea 18

------ > ------