Estructuras de Datos 1 - ST0245
Segundo Parcial Grupo 033

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

Mayo 14 de 2020

1 Pilas 20%

Cuando se realiza una obra de construcción, existen varios impactos ambientales que pueden ser medidos con indicadores de sostenibilidad. Un problema consiste en encontrar la combinación de procesos que minimizan estos tres indicadores de sostenibilidad: Las emisiones de CO_2, la energía consumida y los costos económicos. Este problema de optimización es un problema de búsqueda; por eso necesitamos que implementes el algoritmo *búsqueda del primero en profundidad (DFS)* usando pilas. Como una ayuda se te entrega el siguiente código, al cual le faltan algunas líneas; por favor, complételas. ¡Vamos por el medio ambiente!

```
1  void dfs(ArrayList<Integer>[] g, int v,
       boolean[] seen){
2    Stack<Integer> s = new Stack();
3    s.push(v);
4    while(!s.empty()){
5      v = ................;
6      if(seen[v])
7        ................
8      seen[v] = true;
9      for(int i=0; i<g[v].size(); ++i){
10       int u = ..............;
11       if(!seen[u])
12         ..................;
13 } } }
```

A (10%) Completa la línea 5 .....
Completa la línea 7 .....

B (10%) Completa la línea 10 .....
Completa la línea 12 .....

Pista: La palabra reservada `continue` se salta la iteración actual un ciclo y salta a la siguiente; la palabra `break` termina abruptamente un ciclo. En una pila, el método `push` ingresa un elemento a la pila y `pop` lo returna y elimina.

2 Colas 20%

En el curso de Organización de Computadores (obligatoria para Sistemas, electiva para Matemáticas), profundizaremos sobre el uso de números binarios en la definición de las operaciones que realiza la unidad central de procesamiento (CPU). Mientras tanto, resolvamos el siguiente problema: Dado un entero N, genera los binarios entre 1 y N. Por ejemplo, si $N = 7$, entonces la respuesta es $[1, 10, 11, 100, 101, 111]$. El siguiente código, resuelve el problema, pero faltan algunas líneas; por favor, complételas.

```
1  List<String> solve(int N){
2    List<String> res = new ArrayList();
3    Queue<String> q = new ArrayDeque();
4    q.add("1");
5    for(int i = 0; i < N; ++i){
6      q.add(..... + "0");
7      q.add(..... + "1");
8      res.add(q.poll());
9    }
10   return res;
11 }
```

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

B (10%) Completa la línea 6 .....
Completa la línea 7 .....

ArrayQueue es una clase que implementa una cola con arreglos. El método `poll` elimina y retorna un elemento de la cola, `peek` permite verlo sin eliminarlo y `add` ingresa un elemento a la cola.

3 Tablas de Hash 20%

Cuando un sitio web te solicita tu login y password para autenticarte, el password que se almacena en el servidor es el el hash del password. De esa manera, si un atacante accede a los passwords almacenados en el servidor, estos no le serán útiles para autenticarse en el sitio web. Para el siguiente ejercicio, se usa la función de hash

$hash(s) = \sum_{i=0}^{|s|-1} s[i] * p^i \mod m$, donde p es un número primo menor que 40, m es $10^9 + 7$ y *mod* es el módulo (o residuo) de la división entera.

Te entregan n cadenas de caracteres de tamaño no mayor que m. La idea es encontrar cuan-

tas de esas cadenas de caracteres son diferentes. Por ejemplo, si la entrada es s = ["*aba*","*abc*","*abcd*","*aba*","*aba*","*abd*","*abb*","*abcd*"], entonces la respuesta es 5. El siguiente código resuelve el problema, pero faltan algunas líneas; por favor, complételas.

```
1  int solve(String[] s){
2    long[] h = new long[s.length];
3    for(int i = 0; i < s.length; ++i){
4      h[i] = hash(s);
5    }
6    Arrays.sort(h); // Sort es O(n. log n)
7    int cnt = 0;
8    for(int i = 0; i < s.length; ++i){
9      if(..................)
10       cnt ++;
11   }
12   return cnt;
13 }
```

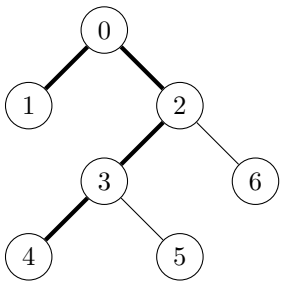
A (10%) En términos de n y de m, ¿cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior? .....

B (10%) Completa la línea 9 .....

Pista: No olvides la complejidad de `hash(s)`

4 Árboles 30%

El árbol de codificación de Huffman es un tipo de árbol binario que se utiliza para compresión de datos sin pérdidas en formatos como mp3 y jpg. Volviendo a los temas del curso, en este problema, como bien sabes, un *árbol* es un grafo no dirigido con n vértices y $n-1$ artistas; también se puede asumir que el peso de cada arista es de 1. Considera un árbol con raíz en el nodo v, sabemos que d_i (usando el algoritmo de *búsqueda primero en amplitud* [BFS]) es la distancia mas corta desde el nodo v al nodo i. En este ejercicio, vamos a encontrar el *diámetro* de un árbol de n nodos. El diámetro de un árbol se define como el largo máximo (es decir, el número de aristas) entre los caminos más cortos que hay entre cualesquiera dos vértices del árbol. En el siguiente árbol se describe su diámetro con una línea repintada, el cual es de 4.



El siguiente código encuentra el diámetro de un árbol, pero faltan algunas líneas; por favor, complétalas.

```
1  int[] bfs(ArrayList<Integer>[] g, int v){
2    int[] d = new int[g.length];
3    Arrays.fill(d, Integer.MAX_VALUE);
4    d[v] = 0;
5    Queue<Integer> q;
6    q = new LinkedList<Integer>();
7    q.add(v);
8    while(!q.isEmpty()){
9      int s = q.poll();
10     Iterator<Integer> i=g[s].listIterator
           ();
11     while(i.hasNext()){
12       int n = i.next();
13       if(d[s] + 1 < d[n]){
14         d[n] = d[s] + 1;
15         q.add(n);
16   } } }
17   return d;
18 }
19 int diametro(ArrayList<Integer> g[]){
20   int v, u, w;
21   v = u = w = 0;
22   int[] d = bfs(g, v);
23   int n = d.length;
24   for(int i = 0; i < n; ++i)
25     if( ..... ) u = i;
26     int[] f = bfs(g, u);
27     for(int i = 0; i < n; ++i)
28       if( ..... ) w = i;
29   return f[w];
30 }
```

El método `Arrays.fill(a,v)` coloca el valor v en todas las posiciones del arreglo a.

A (10%) Completa la línea 25 .....

(a) $d[u] > d[i]$
(b) $d[i] > d[u]$
(c) $f[w] > f[i]$
(d) $f[i] > f[w]$

B (10%) Completa la línea 28 .....

(a) $d[u] > d[i]$
(b) $d[i] > d[u]$
(c) $f[w] > f[i]$
(d) $f[i] > f[w]$

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

5 Grafos 10%

En la vida real, un grafo se utiliza para representar los mapas de *Google Maps*. Para cada una de las siguiente proposiciones, determine si son verdaderas o falsas.

A (10%) Al representar un grafo completo con matrices de adyacencia y con listas de adyacencia, en ambos casos, el consumo de memoria (NO tiempo, sino memoria) es $O(n)$.

(a) Verdadero
(b) Falso

Cuando representamos un grafo con matrices de adyacencia, la complejidad asintótica, en el peor de los casos, de insertar un nuevo vértice es $O(n^2)$, donde n es el número de vértices.

(a) Verdadero

(b) Falso

Un grafo completo es áquel donde existe una arista entre cada par de vértices del grafo.