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

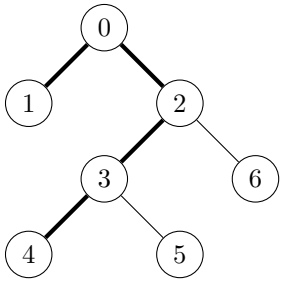
Nombre .....
Departamento de Informática y Sistemas
Universidad EAFIT

21 de Marzo de 2019

En las preguntas de selección múltiple, una respuesta incorrecta tendrá una deducción de 0.2 puntos en la nota final. Si dejas la pregunta sin responder, la nota será de 0.0. Si no conoces la respuesta, no adivines.

1 BFS y Grafos (20%)

Como bien sabes, un *árbol* es un grafo no dirigido con n vértices y $n-1$ artistas donde el peso de cada arista es 1. Considera un árbol con raíz v. Sabemos que d_i, usando algoritmo de *búsqueda primero en amplitud* (en Inglés, BFS), es la distancia más corta del 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 número máximo de aristas del camino más corto entre cualesquiera dos vértices del árbol. Como un ejemplo, en el siguiente árbol, se resalta su diámetro, el cual es de 4.



La siguiente es la implementación del algoritmo de BFS para calcular las distancias más cortas del nodo v a todos los demás nodos de un árbol

```
1    //Retorna di: las distancias mas cortas
2    // del nodo v al nodo i
3    int[] bfs(ArrayList<Integer>[] g, int v) {
4      int[] d = new int[g.length];
5      Arrays.fill(d, Integer.MAX_VALUE);
6      d[v] = 0;
7      Queue<Integer> q;
8      q = new LinkedList<Integer>();
9      q.add(v);
10     while(!q.isEmpty()){
11       int s = q.poll();
12       Iterator<Integer> i=g[s].listIterator();
13       while(i.hasNext()){
14         int n = i.next();
15         if(d[s] + 1 < d[n]){
16           d[n] = d[s] + 1;
17           q.add(n);
18         }
19       }
20     }
21     return d;
22   }
```

Para encontrar el diámetro de un árbol se puede usar el siguiente algoritmo:

- Tomar un vértice v como raíz del árbol g y encontrar d_i, para todo i, usando `bfs(g, v)`.
- Encontrar un vértice u como nodo inicial tal que $d_u \geq d_t$ para todo t. Sea f_i la distancia más corta del nodo u a cualquier nodo i del árbol, el diámetro es $\max_i f_i$

```
1    int diametro(ArrayList<Integer> g[]){
2      int v, u, w;
3      v = u = w = 0;
4      int[] d = bfs(g, v);
5      int n = d.length;
6      for(int i = 0; i < n; ++i)
7        if( ..... ) u = i;
8      int[] f = bfs(g, u);
9      for(int i = 0; i < n; ++i)
10       if( ..... ) w = i;
11     return f[w];
12   }
```

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

i $O(n)$
ii $O(n^2)$
iii $O(n \log n)$
iv $O(n \times \sqrt{n})$

b (10%) Complete las líneas 7, 13 ....., ...

2 Backtracking (20%)

Te entregan una cadena de caracteres s y un entero n, $n \geq |s|$. Necesitamos contar todas las posibles cadenas de caracteres de tamaño a lo sumo n, que contienen **al menos una vez** cada uno de los caracteres de s. Las cadenas generadas sólo pueden contener los caracteres de s. Como algunos ejemplos:

- $s =$ "a", $n = 2$. Respuesta: 2. "a", "aa"
- $s =$ "ab", $n = 2$. Respuesta: 2. "ab", "ba"

- $s =$ "ab", $n = 3$. Respuesta: 8. "aab", "ab", "aba", "abb", "ba", "baa", "bab", "bba".
- $s =$ "abc", $n = 4$. Respuesta: 42.

Nota: `s.toCharArray()` convierte una cadena de caracteres en un arreglo de caracteres, `s.contains(c)` verifica si un caracter está dentro de una cadena de caracteres o no, y `s.length()` retorna la longitud de una cadena.

```
1   //s: Cadena dada
2   //n: Numero dado
3   //gen: Cadena generada
4   int count(String s,int n,String gen){
5     if(..............) return 0;
6     int res = 0;
7     //gen contiene todos los
8     //caracteres que hay en s?
9     boolean all_chars = true;
10    for(char ch: s.toCharArray()){
11      if(.....................){
12        all_chars = false;
13      }
14    }
15    if(all_chars){
16      ..................;
17    }
18    for(char ch: s.toCharArray()){
19      res += count(s, n, gen + ch);
20    }
21    return res;
22  }
```

a (10%) Complete las líneas 5, 11 ....., ...

b (10%) Complete la línea 16 .....

3 Fuerza Bruta (20%)

Dadas dos cadenas de caracteres s_1 y s_2, determine si la cadena s_2 aparece al menos k veces en la cadena s_1.

```
1   boolean valid(String s1, String s2, int k){
2     int cnt = 0;
3     for(int i=0; i < s1.length();++i){
4       String s = ..........;
5       for(int j = i + 1; j < s2.length(); ++j)
            {
6         if(s.equals(s2)){
7           cnt++;
8         }
9         s += s1.charAt(j);
10      }
11    }
12    return cnt >= k;
13  }
```

a (10%) Complete la línea 4 .....

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

i $O(|s1| + |s2|)$

ii $O(|s1| \times |s2|)$

iii $O(|s1|^2 + |s2|)$

iv $O(|s1| + |s2|^2)$

4 Divide y Vencerás (20%)

Dado un arreglo binario ordenado de manera decreciente a (sólo contiene ceros y unos). Determine el número de unos en el arreglo. El tamaño del arreglo es de $n, n \geq 1$ elementos.

```
1   int count(int[] a, int l, int h){
2     if(h >= l){
3       int m = l + (h - l) / 2;
4       if((m==h || a[m+1]==0)&&(a[m]==1)){
5         return m + 1;
6       }
7       if(a[m] ........){
8         return count(a, m + 1, h);
9       }
10      return count(a, l, m - 1);
11    }
12      return 0;
13  }
```

a (10%) Complete la línea 7 .....

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

i $T(n) = T(n-1) + c$, que es $O(n)$

ii $T(n) = T(n/2) + cn$, que es $O(n \log n)$

iii $T(n) = T(n/2) + c$, que es $O(\log n)$

iv $T(n) = c$, que es $O(1)$

5 Voraces (20%)

El problema de colorear los vértices de un grafo es de los más comunes problemas de coloreamiento de grafos. El problema consiste en que, dado m colores, encontrar una forma de colorear los vértices del grafo de tal forma que no haya dos vértices adyacentes que usen el mismo color.

Un algoritmo voraz para este problema funciona de la siguiente manera. Primero, se colorea el primer vértice con el primer color. Segundo, se hace lo siguiente para los $v-1$ vértices restantes: Se intenta colorear el vértice actual con el color enumerado con el menor valor posible que no haya sido utilizado previamente en un vértice adyacente a él. Si todos todos los colores disponibles hasta el momento han sido utilizados en vértices adyacentes al vértice actual, asigna un nuevo color para este vértice.

Nota: Un iterador es un mecanismo que se utiliza en implementación de listas enlazadas para recorrer, en este caso, los vértices adyacentes a un vértice. Las operaciones siguiente (`next()`) y hay un siguiente (`hasNext()`) tienen complejidad $O(1)$.

```
1  // Asigna colores (empezando en 0) a todos
2  // los vertices e imprime la asignacion de colores
3  void greedyColoring() {
4     int result[] = new int[V];
5     // Asigna el primer color al primer vertice
6     result[0]  = 0;
7
8     // Inicializa los otros v-1 vertices con ??
9     for (int u = 1; u < V; u++)
10       ----------------;  // es decir, u no tiene color
11
12    boolean available[] = new boolean[V];
13    for (int cr = 0; cr < V; cr++)
14          available[cr] = false;
15
16    // Asigna colores a los v-1 vertices
17    for (int u = 1; u < V; u++) {
18          Iterator<Integer> it = adj[u].iterator() ;
19          while (it.hasNext())
20          {
21                int i = it.next();
22                if (result[i] != -1)
23                    available[result[i]] = true;
24          }
25
26          // Encuentra el primer color disponible
27          int cr;
28          for (cr = 0; cr < V; cr++)
29              if (available[cr] == false)
30                    break;
31
32          result[u] = cr; // Asigna el color encontrado
33
34          // Pone los valores en falso para la proxima iteracion
35          it = adj[u].iterator() ;
36          while (it.hasNext()) {
37              int i = it.next();
38              if (result[i] != -1)
39                 available[result[i]] = false;
40          }
41    }
42    for (int u = 0; u < V; u++)
43       System.out.println("Vertex " + u + " ---->  Color "
44                          + result[u]);
45 }
```

a (10%) ¿Cuál es la complejidad asintótica, para el peor de los casos, para el algoritmo anterior?

O(----------)

b (10%) Completa el espacio de código que hace falta en la línea 10, para inicializar los colores

----