Estructuras de Datos 2 - ST0247
Segundo Parcial - Jueves (032)

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

Mayo 09 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 Búsqueda Local (20%)

Los algoritmos de ordenamiento son ampliamente usados en diferentes áreas de la Ingeniería de Sistemas como lo es *Big Data*. Debido a esto, es necesario diseñar e implementar algoritmos que resuelvan este problema eficientemente. En este ejercicio, se plantea un algoritmo, que para algunos casos, puede resolver el problema de manera eficiente:

Datos: Secuencia a_i de n números enteros.
Resultado: Secuencia a'_i, tal qué:
$$a'_1 \leq a'_2 \leq \cdots \leq a'_n.$$
curCost = $cost(a)$;
mientras *curCost* > 0 **hacer**
  para $i = 1$ **a** $n - 1$ **hacer**
    $swap(a_i, a_{i+1})$;
    newCost = $cost(a)$;
    si *newCost* $>$ *curCost* **entonces**
      curCost = *newCost*;
    en otro caso
      $swap(a_i, a_{i+1})$;
    fin
  fin
fin

Algoritmo 1: Ordenamiento mediante *Búsqueda Local*.

Sabemos que la función $swap(a_i, a_{i+1})$ intercambia los elementos en las posiciones i e $i+1$ del arreglo a. Estamos interesados en encontrar la función de costo $cost(a)$ para la búsqueda. La función de costo simplemente cuenta todos los pares $(a_i, a_j) \mid i < j \wedge a_j < a_i$. Por favor, completa el siguiente código:

```
1   int cost(int[] a){
2     int respuesta = 0;
3     int n = a.length;
4     for(int i=0; i<n; ++i){
5       for(int j=i+1; j<n; ++j){
6         if(____________){
7           ____________;
8         }
9       }
10    }
11    return respuesta;
12  }
```

a (10%) Completa la línea 6 .....

b (10%) Completa la línea 7 .....

2 Programación Dinámica 30%

Hoy, Liko y Kefa encontraron un arreglo de n monedas. En la posición i del arreglo hay –exactamente– a_i monedas. Ellos quieren saber si pueden repartir las monedas en 2 partes que sumen igual. Ayúdanos, por favor, a resolver este problema. Como un ejemplo, para el arreglo $100, 200, 50, 50$ la respuesta es sí porque se puede dar a una persona $100, 50, 50$ y a la otra persona 200.

```
1   boolean puede(int[] a){
2     int n = a.length;
3     int sum = 0;
4     for(int e: a){
5       sum += e;
6     }
7     if(sum % 2 != 0){
8       ..................;
9     }
10    boolean[][] can = new boolean[sum /
          2 + 1][n + 1];
11    for(int i = 0; i <= n; ++i){
12      can[0][i] = true;
13    }
14    for(int i = 1; i <= sum / 2; ++i){
15      can[i][0] = false;
16    }
17    for(int i = 1; i <= sum / 2; ++i){
18      for(int j = 1; j <= n; ++j){
19        can[i][j] = can[i][j - 1];
20        if(i - a[j - 1]>= 0){
21        can[i][j] |= ..............;
22        }
23      }
24    }
25    return can[sum / 2][n];
```

```
26      }
```

a) (10%) Completa la línea 8 .....

b) (10%) Completa la línea 21 .....

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

i) $O(n)$

ii) $O(n^2)$

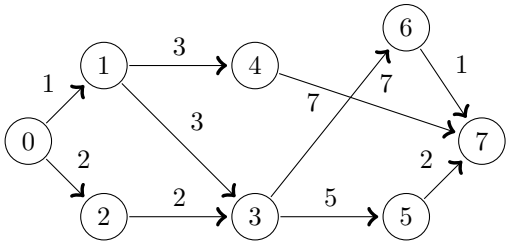
iii) $O(n \times sum)$

iv) $O(n^3)$

3 Dijkstra (30%)

Dado un grafo ponderado y dirigido $G = (V, E, \omega(E))$ y un par de vértices $s, v \in V$, se requiere contar el número de caminos más cortos de s a v. Como un ejemplo, para el siguiente grafo, el número de caminos más cortos entre 0 y 7 es 3 ($0-1-4-7$, $0-2-3-5-7$, $0-1-3-5-7$).

- La palabra reservada **break** termina la ejecución de un ciclo.
- La palabra reservada **continue** interrumpe la ejecución de la iteración actual de un ciclo y empieza una nueva.




```
1     int dijkstra(int[][] g,int s, int v){
2       int n = g.length;
3       int inf = Integer.MAX_VALUE/(2*n);
4       int[] d = new int[n];
5       int[] paths = new int[n];
6       boolean[] w = new boolean[n];
7       Arrays.fill(d, inf);
8       d[s] = 0;
9       paths[s] = ..............;
10      for(int i=0; i<n; i++){
11        int next=-1;
12        for(int j=0; j<n; j++){
13          if(w[j]) continue;
14          if(next==-1 || d[j]<d[next]){
15            next = j;
16          }
17        }
18        if(d[next] == inf) break;
19        w[next] = true;
20        for(int j=0; j<n; j++){
21          int to = j;
22          if(to==next) continue;
23          int weight = g[i][to];
24          int sz = d[next]+weight;
25          if(sz < d[to]){
26            d[to] = sz;
27            paths[to] = ----------;
28          }else if(sz == d[to]){
29            paths[to] = ----------;
30          }
31        }
32      }
33      return paths[v];
34    }
```

a (10%) Completa las líneas 8, 27 ....., ...

b (10%) Completa la línea 29 .....

c (10%) Determina la complejidad asintótica, en el peor de los casos, del algoritmo anterior

i $O(n \log n)$

ii $O(n)$

iii $O(n^2)$

iv $O((n \log n)^2)$

4 Divide y Vencerás 20%

El siguiente algoritmo cálcula a^b. Sin embargo, falta algo. Por favor, complétalo.

```
1     int power(int a, int b){
2       if(b == 0) return 1;
3       if(b == 1) return a;
4       int sig = ----------------;
5       if(b % 2 != 0) return sig * sig * a
            ;
6       else return sig * sig;
7     }
```

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

i) $O(a)$

ii) $O(b)$

iii) $O(\log(a))$

iv) $O(\log(b))$

b) (10%) Completa la línea 4 .....