UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I

Segundo parcial– 24/06/2019

Nro. de orden:
LU:
Apell[redacted]
Nombres:
Nro. de hojas que adjunta:

JACK 95 (25, 40, 30)

Aclaraciones: Se permite tener UNA hoja A4 con anotaciones durante el parcial. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos.
Entregar cada ejercicio en una hoja separada, numerada y que incluya el nro. de orden.

Ejercicio 1. [30 puntos]
Sea *mat* una matriz cuadrada de N filas y N columnas y la siguiente función:

```
void f(vector<vector<int>> &mat){
  int N = v.size();
  for(int i = N - 1; i >= 0, i--) {
    for(int j = N - 1, j >= 0; j--) {
      mat[i][j] = aux(mat, i, j, N);
    }
  }
}
```

```
int aux(vector<vector<int>> &mat, int I, int J, int N) {
  int res = 0;
  int lim = 0;
  for (int i = 0; i <= I; i++) {
    if (i == I) {
      lim = J;
    } else {
      lim = N - 1;
    }
    for (int j = 0; j <= lim; j++) {
      res += mat[i][j];
    }
  }
  return res;
```

a) Describir con sus palabras qué hace la función `f`.
b) Calcular su tiempo de ejecución de peor caso (notación O grande) en función de M, la **cantidad de elementos** de la matriz.
c) Proponer un nuevo algoritmo que resuelva el mismo problema con tiempo de ejecución de peor caso perteneciente a $O(M)$.

Ejercicio 2. [40 puntos]
Sea v y w dos palabras (secuencias de caractéres). Escribir un programa en C++ que determine si una es una permutación de la otra.

a) Con tiempo de ejecución de peor caso perteneciente a $O(|w|^2)$.
b) Suponiendo que todas las letras se encuentran entre la `'a'` y la `'z'`, con tiempo de ejecución de peor caso pertenciente a $O(min(|v|, |w|))$ y con la restricción de recorrer a lo sumo **una vez** cada palabra.

Por ejemplo, si `v = {'a', 'l', 'l', 'a'}` y `w = {'l', 'a', 'l', 'a'}` el resultado deberá ser verdadero pero cuando `w = {'l', 'a', 'a', 'a'}`, `w = {'l', 'c', 'a', 'a'}` o `w = {'l', 'a', 'l', 'a', 'a'}` el resultado deberá ser falso.

Ejercicio 3. [30 puntos]
Dada la siguiente especificación del problema mesetaMasLarga y una implementación en C++:

```
proc mesetaMasLarga (in s : seq⟨Z⟩, out res : Z) {

    Pre {true}

    Post {(∃m : seq(Z))esSubSec(m, s) ∧ esMeseta(m)
          ∧ |m| = res ∧ ¬(∃m' : seq(Z))esSubSec(m', s)
          ∧ esMeseta(m') ∧ |m'| > |m|}

    pred esSubSec (s1 : seq(Z), s2 : seq(Z)) {

        (∃i, j : Z)0 ≤ i ≤ j ≤ |s2| ∧ subsec(s2, i, j) = s1

    }

    pred esMeseta (s : seq(Z)) {

        |s| = 0 ∨L (∀i : Z)0 ≤ i < |s| −→L s[i] = s[0]

    }
```

```
int mesetaMasLarga(vector<int> &v) {
  int i = 0;
  int j = -1;
  int meseta = -1;
  int maxMeseta = 0;
  while (i <= v.size() - 1) {
    j = i + 1;
    while (j < v.size() - 1 && v[i] == v[j]) {
      j++;
    }
    meseta = i - j;
    i = j;
    if (meseta > maxMeseta) {
      maxMeseta = meseta;
    }
  }
  return maxMeseta;
}
```

a) Realizar el diagrama de control de flujo (control-flow graph) del programa.
b) El código no cumple la especificación ya que tiene al menos 2 errores. Escribir 2 casos de test que sean capaces de detectarlos.
c) Estos casos de tests, ¿Cubren todas las líneas del programa? ¿Cubren todos las ramas del programa (branches)?

Orden: [redacted]

1) a) • En primer lugar, la función 'f' recorre cada uno de los elementos de la matriz 'mat', comenzando por la última columna de la última fila, empleando 'aux', siguiendo con la columna anterior hasta llegar a la primera, luego de la cual pasa a la fila anterior.

(*) más de esto abajo.

• En otras palabras, recorre la matriz de fin a principio, por columnas, luego, filas.

• Ej para una matriz de 3x3:

9°	8°	7°
6°	5°	4°
3°	2°	1°

← órden de recorrido.

aux():

Cada elemento recorrido de la matriz, es reemplazado por lo que devuelve 'aux()':

Esta función auxiliar se encargará de sumar TODOS los elementos ANTERIORES al elemento mat[i][j] (incluído), siguiendo un órden inverso al recorrer las columnas (de principio a fin) teniendo en cuenta que a la fila correspondiente el elemento con el cual es llamado 'aux', solo se recorren (suman) las columnas desde 0 (cero) a j, y en las filas anteriores, todas las columnas de principio a fin.

aux devuelve el resultado de la suma anteriormente descripta, y en 'f', reemplaza ↑ el elemento correspondiente a mat[i][j]. in place!

En otras palabras, el algoritmo modifica la matriz pasada por referencia, escribiendo en cada casilla, la suma de los elementos anteriores (siguiendo el órden antes descripto).

- Ej matriz 3x3

i \ j	0	1	2
0	1	1	1
1	1	2	1
2	1	1	1

$\Rightarrow$ f(mat)

mat luego de ser llenada f

1	2	3
4	6	7
8	9	10

b) M: Cant. de elementos de mat.

Por cada elemento de mat, se recorren todos los elementos anteriores (en aux) por lo que el tiempo de ejecución ~~de 'f' será~~ de la 1° iteración de 'f' será $O(M)$, en la 2° iteración de 'f' $O(M-1)$, hasta llegar a la última iteración, que será $O(1)$

O sea, el tiempo de ejecución de peor caso será:

$$O(M) + O(M-1) + O(M-2) + \dots + O(2) + O(1)$$

$$:= \sum_{i=1}^{M} O(i) := O\left(\frac{M.(M+1)}{2}\right) := O\left(\frac{M^2+M}{2}\right)$$

↑ suma de Gauss

$\therefore$ el tiempo de ejec. de peor caso es $O(M^2)$

√ Perfecto ☺ (Perdón)

ejecución

e 'f' será $O(M)$, en la 2°

legar a la última iteración, f

O sea, el tiempo de ejecuci

$$O(M) + O(M-1) + O(M-2)$$

$$:= \sum_{i=1}^{M} O(i) := O\left(\frac{M.(M+1)}{2}\right)$$

↑ suma de Gauss

∴ el tiempo de ejec. de peor

NOTA

√ Perfecto :) (Perdón)

Ejer 1) c)

Propongo recorrer la matriz desde [0][0] a [N-1][N-1], sumando en cada iteración SOLO el elemento anterior (ya que tendré la suma de todos los anteriores):

```
void f(vector<vector<int>> &mat) {
    int N = mat.size();
    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            mat[i][j] = aux(mat, i, j, N);
        }
    }
}
```

OK. Muy rebuscado! (preguntar)

```
int aux(vector<vector<int>> &mat, int I, int J, int N) {
    int res = 0;
    if (J == 0) {
        # 1° columna
        if (I == 0) {
            # mat[0][0]
            res = mat[I][J];
        } else {
            # 1° columna de alguna otra fila
            res = mat[I][J] + mat[I][J-1];
                                # ∧ ult. elem de fila anterior.
        }
    } else {
        # otra columna: suma elem anterior de misma fila
        res = mat[I][J] + mat[I][J-1];
    }
    return res; } # Fin
```

(*) I-1, N-1

Orden: ●

También podría haber operado directamente sobre mat aprovechando que es llamada por referencia en svd, pero me pareció que quedaría poco claro.

(en este caso no jeje)

Ej 2) a) Estrategia: 0: Chequeo que $|v| == |w|$

1. Ordeno v ($O(|v|^2)$)
2. Ordeno w ($O(|w|^2)$) → mismo código
3. Comparo uno a uno: ($O(|v|)$)

$::= O(|v|^2 + |w|^2 + |v|)$
$|v| == |w|$
$::= O(2|v|^2 + |v|)$
$::= O(|v|^2)$
$::= O(|w|^2)$

o Uso Bubble Sort: (in place)

```
void ordenar (vector<char>& secue) {
    for (int i=0; i < secue.size(); i++) {
        for (int j=0; j < secue.size()-1; j++) {
            if (ord(secue[j]) > ord(secue[j+1])) {
                swap(secue[j], secue[j+1]);
            }
        }
    }
}
```

B

```
bool esPermutacion (vector<char> v, vector<char> w) {
    if (v.size() != w.size()) {
        return false;
    }
    ordenar(v); // # paso por referencia
    ordenar(w); // # ordena en el lugar.
    bool iguales = true;
    int i=0;
    while (iguales && i < v.size()) {
        if (v[i] != w[i]) {
            iguales = false;
        } i++; } // # me quedo sin espacio D:!
    } return iguales; } // # fin.
```

b) Estrategia: 1°. Chequeo |v| == |w|

2°. Cuento letras de cada tipo de v en un vector contador sumando 1

3°: Lo mismo para w, pero restando 1.

4°: Si termino con un vector de ceros, es permutación, sino no.

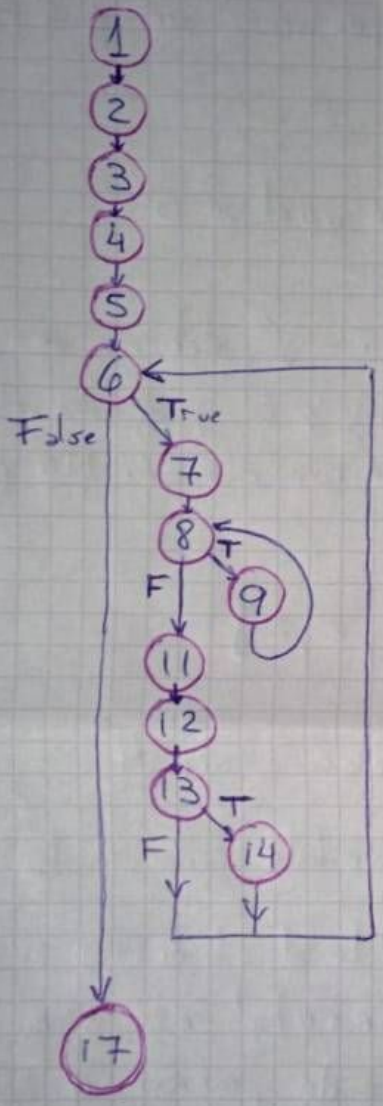
```
bool esPermutacion(vector<char> v, vector<char> w){
    if(v.size() != w.size()){
        return false;
    }
    # supongo ABCdario de 27 letras, con ord('n') < ord('ñ') < ord('o')
    vector<int> contadores(27,0); # tamaño 27, lleno de ceros.
    int idx;
    for(int i=0; i<v.size(); i++){
        idx = ord(v[i]) - ord('a'); # 'normalizo' ord() para que
                                    # comience en cero.
        contadores[idx] += 1;
        idx = ord(w[i]) - ord('a');
        contadores[idx] -= 1;
    }
    # solo resta ver si son todos ceros.
    bool res = true;
    [illegible]
    while(res == true && i < contadores.size()){
        if(contadores[i] != 0){
            res = false;
        }
    }
    return res;
}
```

B

Orden: [redacted]

[redacted]

3) a) Comienza por llamada a 'MeseteMásLarge()' (①):



Aclaraciones

6: corresponde al 1° while

8: 2° while

13: único if.

17: return nombreSete y fin del programa.

b) Errores encontrados:

1. el código no cuenta el último elemento de una secuencia. (error en línea 8, debería ser $\leq$ v.size()-1)

Test: v = [5] ✓

Debería devolver 1, devuelve 0.

2. en la línea (11), "mereta = i - j", debería ser, por ejemplo, "mereta = j - i", ya que j ≥ i, y 'mereta' cuenta el largo de una subsecuencia ∴ No puede ser menor a cero. ✓

Test: v = [1, 1, 1, 1, 2, 2]
idx: 0 1 2 3 4 5

Devuelve cero, cuando debería ser 4.

c) No y No, ya que no es posible cubrir la línea 14 (el 'branch' correspondiente al if de (13) si se cumple la condición) pues 'mereta' es siempre menor o igual a cero, y por ende, menor o igual a 'maxMereta', o sea la línea (14) es inalcanzable. ✓