Examen de Programación 3

12 de diciembre de 2016

- Este parcial dura **3** horas y consta de 2 carillas. El total de puntos es **100**.
- **NO** se puede utilizar ningún tipo de material de consulta. Salvo que se indique lo contrario podrá usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.
- **Justifique todas sus respuestas**.

Se requiere:

I) Numerar todas las hojas e incluir en cada una el **nombre, cédula de identidad, número de página y cantidad de hojas entregadas**.

II) Utilizar las hojas de **un solo lado** y escribir con lápiz.

III) Iniciar cada ejercicio en hoja nueva.

Parte obligatoria

Esta parte es eliminatoria. Para la aprobación del examen debe obtenerse un mínimo del **50 % de esta parte (20 puntos)**. En caso de no llegar a dicho mínimo, **NO** se corregirán los problemas.

Pregunta 1 (12 puntos)

Asuma que $\forall n \geq n_0, f(n) \geq g(n)$ y que $f(n), g(n) \in \Theta(h(n))$.

(a) Utilizando la regla del limíte indique el/los órden(es) posibles de: $f(n) - g(n)$.

Pregunta 2 (14 puntos)

Considere el algoritmo mostrado a la derecha.

(a) ¿Qué problema resuelve?

(b) Si se toma como operación básica la comparación con elementos del arreglo A, ¿cuándo se da el peor caso y por qué?

(c) Explique si el algoritmo es estable o no. En caso de no ser estable modifíquelo para que sea estable.

```
1 Procedimiento(A[1..n])
2     for i = 2 to n
3         dato ← A[i]
4         j ← 1
5         while dato > A[j]
6             j++
7         for h = i downto j + 1
8             A[h] ← A[h-1]
9         A[j] ← dato
```

Pregunta 3 (14 puntos)

Dada una recorrida (DFS o BFS) en un grafo no dirigido.

(a) Defina arista *forward*.

(b) ¿Es posible tener más aristas *back* que *forward* si la recorrida es DFS? Justifique.

(c) ¿Es posible tener más aristas *cross* que *back* si la recorrida es BFS? Justifique.

(d) Escriba una condición para una artista (v, w) en función de los prenums y posnums de una recorrida DFS que sea verdadera si y solo si la artista es *back*.

Problemas

Problema A (30 puntos)

Kailor se propone determinar los recorridos con los que más rápidamente puede llegar a cada una de las paradas de ómnibus de su barrio. Para ello identifica cada una de las esquinas de la ciudad (algunas de las cuales son las paradas de ómnibus de su barrio) con un número entre 1 y n y mantiene en el arreglo bidimensional C el tiempo que le lleva recorrer cada una de las cuadras. Asigna $C[i, i] = 0$ y, para $i \neq j$, $C[i, j] = \infty$ si no hay una cuadra desde i hasta j. Se denota con P el conjunto que contiene los identificadores de las esquinas en que se encuentran cada una de las paradas y con K la esquina de la casa de Kailor. Se asume que desde K se puede llegar a cada una de las esquinas en P.

Se quiere construir una estructura E mediante la cual dada una esquina cualquiera $p \in P$ se pueda obtener en tiempo $O(r)$ el recorrido más rápido desde K hasta p, siendo r la cantidad de cuadras del recorrido.

(I) Escriba el seudocódigo de `Estructura(C, n, K, P)` que devuelve la estructura E. El tiempo de ejecución debe ser polinómico en n y el algoritmo debe terminar tan pronto como en E se disponga de la información necesaria para obtener el recorrido desde K hasta cada esquina en P.

(II) Escriba el seducódigo de `Recorrido(E, p)` que dadas la estructura E y una esquina $p \in P$ imprime las esquinas del recorrido más rápido desde K hasta p, en el orden en que deben visitarse. El tiempo de ejecución debe ser $O(r)$, siendo r la cantidad de cuadras del rcorrido.

Problema B (30 puntos)

Sea $G = (V, E)$ un grafo dirigido, se denomina *clique* a todo subgrafo de G que sea completo. Encontrar un *clique maximal* en un grafo consiste en encontrar un clique con la mayor cantidad de vértices posibles. Sea pretende encontrar un *clique maximal* en un grafo mediante *backtracking* utilizando una formulación con tuplas de tamaño fijo.

(I) Defina la forma de la tupla.

(II) Defina restricciones explícitas, implícitas y función objetivo.

(III) Defina un predicado de poda.

(IV) Considere el siguiente algoritmo:

```
clique_maximal(G):
  conj_vertices = conjunto_de_partes(G.vertices) // subconjutos de V
  clique = {}

  For conj in conj_vertices:
    If es_clique(conj, G) and (tamanio(conj) > tamanio(clique)):
      clique = conj

  return clique
```

¿El algoritmo planteado es más eficiente que su propuesta de backtracking? Justifique