Examen de Programación 3 y III

12 de diciembre de 2014

- Este examen dura **4 horas** y contiene 4 carillas. El total de puntos es **100** y se requieren **60** para su aprobación.
- En los enunciados llamamos C* a la extensión de C al que se agrega el operador de pasaje por referencia & y las sentencias new, delete y el uso de cout y cin.
- **NO** se puede utilizar ningún tipo de material de consulta.
- **NO** se contestarán dudas durante la última media hora

Se requiere:

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

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

III) Iniciar cada ejercicio en hoja nueva.

IV) Poner en la carátula la cantidad de hojas entregadas y un índice indicando en qué hojas respondió cada problema.

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.*

Ejercicio 1 (10 puntos)

Determine si las siguientes afirmaciones son verdaderas o falsas. Deberá justificar su respuesta **usando solamente** las definiciones de O, Ω y Θ:

a) $sen(n) + 2 \in \Theta(1)$

b) $O(n\ log(n)) \subset O(n^2)$

Si utiliza alguna propiedad vista en el curso deberá demostrarla.

Ejercicio 2 (10 puntos)

a) Escriba el algoritmo *InsertionSort* visto en el teórico del curso.

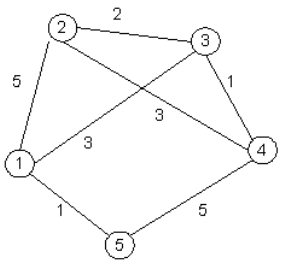
b) Indique cuál es el peor caso de dicho algoritmo.

c) Calcule la cantidad de comparaciones de elementos que se realizan en el peor caso.

NOTA: no se corregirá una parte si no se realizaron las anteriores.

Ejercicio 3 (10 puntos)

Sea G el siguiente grafo no dirigido:



Muestre paso a paso y de forma detallada cómo aplica el algoritmo de *Dijkstra* a partir del vértice 2. Justifique cada paso.

Ejercicio 4 (10 puntos)

Sea $G = (V, E)$ un grafo conexo, no dirigido, sin lazos ni aristas múltiples. Si se realiza una recorrida *BFS* del grafo a partir de cualquier vértice se genera un árbol de cubrimiento de G, $T = (V, E')$. Escriba un algoritmo que dados G y $v \in V$ haga una recorrida *BFS* y devuelva T. Los grafos G y T estarán representados por una matriz de adyacencia.

Problemas

Problema 1 (30 puntos)

Dado $G = (V, E)$ un grafo no dirigido, una coloración de G consiste en asignar a cada vértice un color de $C = \{c_1, \cdots, c_k\}$ colores disponibles de forma que a los vértices adyacentes le sean asignados colores distintos. Todo vértice debe ser coloreado. La cantidad de vértices es $|V| = n$ y los vértices están identificados con un entero en $\{0, \cdots, n-1\}$.
Con el fin de dibujar el grafo, considere que los vértices de G tienen asociado un valor que representa el tamaño con que será dibujado; estos tamaños están dados por la función $D : V \to R^+$.
A su vez se considera un función $A : C \to R^+$ que asigna a cada color un valor indicando qué tan agradable es a la vista.

Se define la agradabilidad de la coloración de un vértice como el producto de su tamaño con la agradabilidad del color asignado.

Se define la agradabilidad de la coloración del grafo como la suma de las agradabilidades de sus vértices.

Se desea colorear el grafo de forma que se maximice su agradabilidad usando la menor cantidad de colores posibles (en particular: dentro de las soluciones que usen la menor cantidad de colores posibles, la que maximice la agradabilidad).

a) Formalice el problema en términos de ***Backtracking*** indicando todos los elementos que correspondan.

b) Implemente el algoritmo de *Backtracking* siguiendo el siguiente pseudocódigo y restricciones.
Las implementaciones que no se ajusten al pseudocódigo serán consideradas incorrectas.

```
void Colorear(tupla t, int indice, tupla &sol) {
    if (PrefijoValido(t,indice)) {
        if (!Poda(t,indice,sol)) {
            if (EsSolucion(t,indice)) {
                Copiar(sol,t); // copia t en sol
            } else {
                for(int c=0; c<k;c++) {
                    AsignarColor(t,indice,c);
                    Colorear(t,indice+1, sol);
                    DesasignarColor(t,indice,c);
                } // for
            } // else
        } // if (!Poda())
    } // if (PrefijoValido())
}
```

Asuma que `D` y `A` son arreglos de tipo `float` de dimensiones `n` y `k` respectivamente.

Implementar:

- Tipo `tupla`
- `bool PrefijoValido(···)`
- `bool Poda(···)`
- `bool EsSolucion(···)`
- `void AsignarColor(···)`
- `void DesasignarColor(···)`

c) Dibuje el espacio de soluciones para el siguiente grafo:

Con $C = \{c_1, c_2\}$; $D(1) = 100$, $D(2) = 2$, $D(3) = 3$ y $A(c_1) = 100$, $A(c_2) = 1$.

Indique claramente las aplicaciones de los distintos elementos de la formalización (poda, mejor solución hasta el momento, mejor solución al problema, etc.).

Problema 2 (30 puntos)

Este problema trata sobre recorridas *DFS* en un grafo. Tenga en cuenta las siguientes consideraciones:

- $G = (V, E)$ es un grafo dirigido, sin lazos ni aristas múltiples. La cantidad de vértices es $|V| = n$ y los vértices están identificados con un entero en $\{1, \cdots, n\}$.
- Para las partes (b) y (c) asuma implementado el invocador/inicializador de la recorrida.
- Cada parte de los requerimientos podrá tener consideraciones específicas adicionales.
- **Toda solución que no respete alguna de las restricciones será considerada incorrecta.**

a) Defina el concepto de arista *Tree*, *Back*, *Cross* y *Forward* de una recorrida *DFS* sobre G.

b) Escriba el pseudocódigo de una recorrida *DFS* sobre un grafo. Indique con un comentario las porciones del código que corresponden al preprocesamiento y postprocesamiento.

Considere la siguiente firma para el pseudocódigo:

```
void DFS(Vertice v, Bool[] visitados); //Asumir visitados inicializado en False
```

c) Basándose en el pseudocódigo anterior, escriba un algoritmo que recolecte las aristas *Tree*, *Back*, *Cross* y *Forward* de la recorrida.

Las aristas deben ser recolectadas en la misma recorrida y para identificar las aristas sólo debe usar información de prenum y posnum.

Considere la siguiente firma:

```
void DFS(Vertice v, bool[] visitados, int[] prenum, int[] posnum,
         ListaArista &tree, ListaArista &back, ListaArista &forward,
         ListaArista &cross);
```

Asuma que:

- Para la actualización de `prenum` y `posnum` cuenta con dos contadores `c1` y `c2` con alcance global, inicializados en 0 en el invocador.
- Los arrays `prenum` y `posnum` están inicializados con -1 en cada posición.
- Las listas de aristas `back`, `cross`, `forward` y `tree` están inicializadas como vacías.