Examen de Programación 3 y III (20/07/2013)

Instituto de Computación, Facultad de Ingeniería, UdelaR

1. Este examen dura 4 horas y contiene **2** carillas. El total de puntos es 100 y se requieren 60 para su aprobación.
2. 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*, el uso de *cout* y *cin y el tipo bool*.
3. NO se puede utilizar ningún tipo de material de consulta.
4. No se contestarán dudas durante la última media hora.

Se requiere:

- Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.
- Utilizar las hojas de un solo lado y escribir con lápiz, iniciando cada ejercicio en hoja nueva.
- Poner en la primera hoja la cantidad de hojas entregadas, y entregar el **índice** indicando en qué hoja se 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)

Para cada afirmación, indique si es verdadera o falsa y justifique detalladamente su respuesta (las respuestas sin justificación se considerarán no respondidas).

1. Sea un grafo dirigido *G*. Si el grado de entrada de uno de sus vértices es 0, entonces es posible realizar una ordenación topológica de los vértices de *G*.
2. Considere un árbol AVL y un ABB que tienen los mismos elementos. Siempre se cumple que la altura del ABB es mayor o igual que la altura del AVL.
3. Considere dos diccionarios de enteros: uno de ellos implementado con un ABB y el otro implementado con una Tabla de Hash utilizando resolución abierta de colisiones (mediante listas enlazadas). Sea *S* una secuencia de inserciones. Si *S* es un peor caso para el ABB, entonces también es un peor caso para la Tabla de Hash.
4. Considere la función de Fibonacci, definida de la siguiente manera:
   $fib(0) = 0$
   $fib(1) = 1$
   $fib(n) = fib(n-1) + fib(n-2)$ para $n \geq 2$
   Es posible implementar un algoritmo que tarde $O(n)$ en calcular $fib(n)$.
5. Los algoritmos de Prim y Kruskal son algoritmos ávidos (o *greedy*).

Ejercicio 2 (10 puntos)

Considere el siguiente algoritmo que ordena una secuencia de ***n*** enteros:

```
void Sort_Sec (int* S, int n){
    int i, j, Primero;
    for (i = 1; i < n; i++){
          Primero = S[i];
          j = i-1;
          while (j >= 0 && Primero <= S[j]){
              S[j+1] = S[j];
              j--;
          }
          S[j+1] = Primero;
    }
}
```

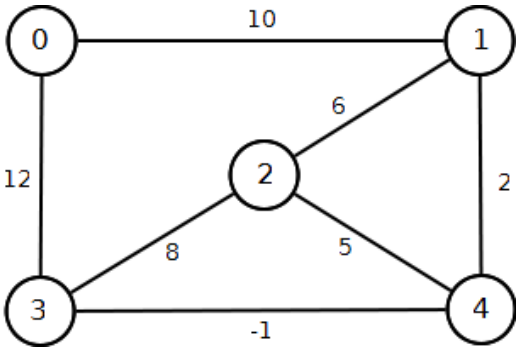
a) Calcule la cantidad exacta de comparaciones de elementos que realiza este algoritmo en su peor caso. Indique cuál es ó cuándo se da el peor caso.
b) Indique si el algoritmo dado es estable. Argumente adecuadamente.

Ejercicio 3 (10 puntos)

a) Plantee la recurrencia que calcula el costo del camino de costo mínimo entre dos vértices de un grafo con costos en sus aristas. Demuestre que su solución cumple con el principio de optimalidad.

Ejercicio 4 (10 puntos)

a) Defina *Árbol de Cubrimiento de Costo Mínimo*.
b) ¿Qué algoritmo visto en el curso utilizaría para encontrar el *Árbol de Cubrimiento de Costo Mínimo* para un grafo dado?
c) Aplique el algoritmo de la parte b) al siguiente grafo, detallando todos pasos del mismo.

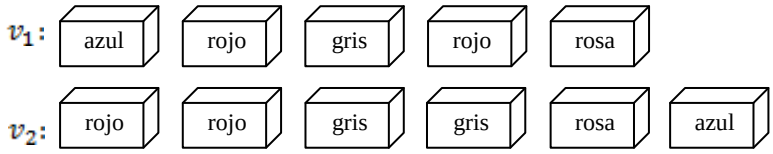


Problemas

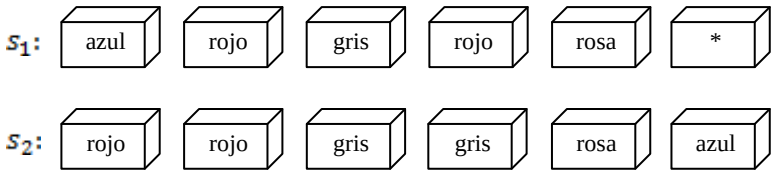
Problema 1 (30 puntos)

Se propone resolver, mediante programación dinámica, el siguiente juego:
Dadas dos secuencias iniciales v_1 y v_2 de cubos de colores, obtener una alineación de costo óptimo de los colores. Una alineación se define como dos secuencias s_1 y s_2 tales que s_i es una secuencia formada con los cubos originales de v_i, conservando su orden original. Es posible utilizar un cubo especial denominado "comodín", que podrá ser intercalado en cualquier posición de las secuencias s_1 y s_2. No hay límite para la cantidad de cubos "comodín" a utilizar. Para una solución ambas secuencias s_1 y s_2 deben tener el mismo largo, aunque las secuencias originales de entrada no la tengan (en dicho caso, la utilización de cubos "comodín" será inevitable).

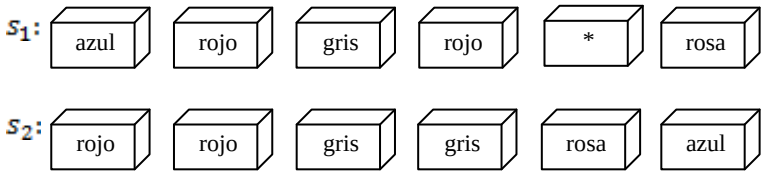
A modo de ejemplo, dadas las secuencias iniciales v_1 y v_2 que se muestran a continuación:



Una alineación factible es, por ejemplo:



Otra posible alineación de las mismas secuencias originales es la siguiente:



Donde el cubo denotado con un asterisco representa al comodín.
Se denomina costo de una alineación a la cantidad de posiciones con cubos de distinto color. Los cubos "comodín" son considerados como cualquier otro cubo, es decir, si en la posición i, sólo una secuencia tiene un comodín, entonces se consideran distintos, mientras que para el caso de que ambas secuencias tengan comodines, entonces se considerarían iguales. En los ejemplos, el costo de la primera alineación es 3, mientras que el costo de la segunda es 4.

a) Se pide resolver, mediante programación dinámica, el problema de determinar el costo de una alineación óptima, dadas dos secuencias de cubos (donde una alineación óptima es aquella de mínimo costo). Llame a la fórmula recursiva $\boldsymbol{f}$. Explicar qué representa el/los paso/s base y cada paso recursivo de la solución, así como también qué representa la función $\boldsymbol{f}$ indicando sus índices, y cómo debe invocarse la misma para resolver el problema.
Suponga que puede acceder a las posiciones de las secuencias originales mediante los vectores v_1 y v_2 (de largos n_1 y n_2 respectivamente) y que puede realizar comparaciones de igualdad entre las posiciones de los mismos (por ejemplo: $v_1[i] = v_2[i]$).

Nota: **NO** se pide determinar las secuencias que conforman la alineación óptima sino el valor de su costo asociado.

b) Utilizando **Programación Dinámica** implemente una función **ITERATIVA** en C* que solucione el problema de la parte a).

Problema 2 (30 puntos)

Se desea conformar un DVD de los grandes éxitos de un determinado artista. Dicho artista tiene un conjunto de ***N*** videos para los cuales se cuenta con la duración (en segundos) y la valoración de cada video. Dicha valoración varia entre 1 y 5 siendo 1 la mejor valoración y 5 la peor.

Por cuestiones de marketing el DVD debe contener 20 videos y la duración máxima del DVD debe ser de 60 minutos.

Se pretende que este DVD se venda de forma masiva por lo que se desea obtener la mejor valoración del mismo (la suma de las valoraciones de los videos que se encuentran en él)

Se pide:

a) Formalizar el problema en términos de **Backtracking**.
Indicar: forma de la solución, restricciones explícitas e implícitas, predicados de poda y función objetivo.

NOTA: No se corregirá la parte b) si no se realizó satisfactoriamente la parte a).

b) Implementar una función en C* que resuelva el problema utilizando **Backtracking**.

Se dispone de las siguientes estructuras:

- duracion[0..N-1], donde duracion[i] indica la duración (en segundos) del i-ésimo video.
- valoracion[0..N-1], donde valoracion[i] indica la valoración del i-ésimo video.