Segundo Parcial de Programación 3 (1/12/2009)

Instituto de Computación, Facultad de Ingeniería

- Este parcial dura 4 horas y contiene 4 carillas. El total de puntos es 60.
- 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. Salvo que se indique lo contrario puede usarse todo lo visto en el teórico, práctico y laboratorio sin demostrarlo, indicando claramente lo que se está usando.

Se requiere:

i. Numerar todas las hojas e incluir en cada una el nombre y la cédula de identidad.

ii. Utilizar las hojas de un sólo 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 respondieron cada problema.

Se valorará muy especialmente la justificación de las respuestas para todos los problemas, especialmente donde esto se pide de forma explícita.

Ejercicio 1 (20 puntos)

Un DVD Club cuenta con un buzón para la devolución de DVDs. Todas las mañanas, antes de abrir sus puertas, hay una persona encargada de la tarea de llevar cada DVD del buzón a la sección a la cual pertenece. Cada DVD pertenece a una única sección.
Se sabe que el buzón contiene una cantidad *B* de DVDs donde b_h es la cantidad de DVDs que corresponden a la sección *h*, de esta manera se cumple $\sum_{h=0}^{cantSecciones-1} b_h = B$. También se dispone de los costos mínimos de ir de una sección a cualquier otra sección del DVD Club. Se desea encontrar un **tour** de mínimo costo para esta tarea. Un **tour** es un conjunto de rutas y cada **ruta** es un circuito de secciones que comienza y termina en el buzón. El costo del tour es la suma de los costos de las rutas que lo componen. La persona encargada de la tarea puede transportar hasta *C* DVDs por ruta.

Se pide:

1. **(10 puntos)** Formalizar el problema en términos de Backtracking. Indicar: forma de la solución, restricciones explícitas e implícitas, función objetivo y predicados de poda.

2. **(10 puntos) No se corregirá** esta parte si no se realizó **satisfactoriamente** la parte 1. Implementar la función *bestTour* en C*:

```
bool bestTour (int cantSecciones, int** costoEntreSecciones,int* cantDVDBuzonSeccion,
               int cantDVDSoportado, Tour* &tour);
```

que busca el tour de costo mínimo y retorna *true* si encuentra algún tour y *false* en caso contrario. Los parámetros de la función son los siguientes:

- *cantSecciones* es la cantidad de secciones del DVD Club. Las secciones se identifican por un entero en el rango 0..*cantSecciones*-1.
- *costoEntreSecciones* es una matriz de enteros donde:
    - *costoEntreSecciones[i][j]* indica el costo, en segundos, de ir de la sección ***i*** a la sección ***j*** del DVD Club.
    - *costoEntreSecciones[i][j] = costoEntreSecciones[j][i]* con $0 \leq i, j < cantSecciones$.
- *cantDVDBuzonSeccion* es un arreglo de enteros (de largo *cantSecciones*) donde se indica la cantidad de DVD a llevar a cada sección.
- *cantDVDSoportados* es un entero que indica la cantidad máxima de DVDs que se pueden transportar.
- *tour* es una estructura que contiene la secuencia de rutas para llevar los DVDs a las secciones:

```
struct tour{
  int  cantRutas;
  Ruta* tour;
};
Tour* crearTour();
void destruirTour(Tour* &t );
Tour* copiarTour(Tour* t);
```

```
struct Ruta{
  int costo;//costo de la ruta
  int largoRuta;//cantidad de elementos en ruta
  int* ruta;//arreglo que contiene la secuencia de secciones de la ruta
};
Ruta* crearRuta();
Void destruirRuta(Ruta* &r);
```

Importante:

- En las estructuras *costoEntreSecciones* y *cantDVDBuzonSeccion* el buzón siempre corresponde al índice *0*. Se cumple que: *cantDVDBuzonSeccion [0] = 0*.
- No se pueden llevar más de *C* DVDs en cada ruta.
- En caso de que no existan DVDs en el buzón (el arreglo *cantDVDBuzonSección* sólo contiene ceros), la función deberá retornar *true* (que existe un tour) y en el parámetro *tour*, el tour resultado de invocar la función *crearTour()*.
- Si en la ida a una sección se dejan *n* DVDs, debe aparecer esa sección *n* veces consecutivas en la ruta. Es decir, que si en una ida a la sección *i* se dejan 3 DVDs, en la secuencia de la ruta debe aparecer: ... → *i* → *i* → *i* → ...
- No se puede pasar por una sección sin dejar DVDs. Toda sección (distinta del buzón) que aparece en una ruta corresponde a dejar uno y sólo un DVD.
- El buzón no puede aparecer consecutivamente en una ruta (es decir que en una ruta no puede aparecer más de un cero consecutivo).

Ejercicio 2 (20 puntos)

Un programador tiene interés de desarrollar un programa para hacer traducciones de textos entre distintos idiomas.
Suponga que se cuenta con D diccionarios, donde cada diccionario permite la traducción bidireccional entre dos idiomas. Dados I idiomas, no se tiene porque tener diccionarios para cada par de idiomas, por lo tanto, puede ser necesario realizar varias traducciones para traducir un texto de un idioma a otro. Los idiomas se identifican por un entero en el rango 0..I-1.
Se quiere determinar si es posible realizar la traducción entre dos idiomas dados y, en caso de ser posible, determinar la cantidad mínima traducciones.

Se pide:

1. **(14 puntos)** Dar una fórmula recursiva para solucionar el problema. Llame a la fórmula recursiva ***f***. Explicar qué representa el/los pasos base y cada paso recursivo de la solución, así como también qué representa la función ***f*** indicando sus índices. Se debe indicar la invocación de la función para resolver el problema. Considere disponible la función *ExisteDiccionario* que dados dos idiomas retorna *verdadero* si existe un diccionario que traduce entre ambos idiomas.

   ```
   bool ExisteDiccionario( int a, int b);
   ```

2. **(6 puntos)** Describa qué cambios tendría que hacer para no hallar solamente la cantidad mínima de traducciones para traducir un texto de un idioma otro, sino también para reconstruir la secuencia de traducciones mínima. No es necesario reformular la recurrencia.

Ejercicio 3 (20 puntos)

1. **(6 puntos)** Dados un grafo $G = (V, A)$ dirigido, con costos en sus aristas y una secuencia de vértices $v_1, \ldots, v_k$ que pertenecen a V, se desea encontrar un camino de v_1 a v_k que tenga costo mínimo y visite todos los vértices de la secuencia **respetando el orden de la misma**.

   a) Describa una solución basada en técnicas/algoritmos vistos en el teórico **sin** utilizar Backtracking, especificando, si es necesario, las condiciones mínimas que debe cumplir el grafo para que la solución propuesta funcione correctamente y explique por qué estas condiciones deben cumplirse.

   b) Suponiendo que se cumplen las condiciones especificadas, justifique porqué la solución es correcta.

Nota

- El camino de costo mínimo puede repetir vértices.
- Con respecto a las técnicas/algoritmos y sus propiedades vistas en el teórico deberá explicar cómo se utilizan pero no es necesario probarlas.

2. *Complejidad del problema de sorting*

Sean D_n el conjunto de secuencias de n elementos (con elementos repetidos eventualmente) y el subconjunto D'_n de secuencias de n elementos sin repetidos.

En las partes a) y b) se puede utilizar: $\sum_{i=1}^{n} \log i \geq n \log n - 1.5n$

y para simplificar se puede considerar: $\sum_{i=1}^{n} \log i \cong n \log n$

a) **(6 puntos)** *Complejidad del problema de sorting en el peor caso – cota inferior en el peor caso*

Sea $F_w(n)$ la mejor cota inferior del costo en el peor caso para un algoritmo *A* genérico que ordena una secuencia de $\boldsymbol{D_n}$ por comparaciones de elementos 2 a 2.

Se sabe que:

- cualquier algoritmo *A* tiene asociado su árbol de decisión (AD).
- el costo del algoritmo *A* en su peor caso se denota *Tw(n)*
- *prof(AD) = k*. De lo anterior se concluye que $T_w(n) = k$
- si ***m*** es la cantidad de **nodos externos**, entonces por ***el lema 1*** dado en el teórico: $\boldsymbol{m \leq 2^k}$

i. Calcule $F_w(n)$; justificando detalladamente los pasos de su razonamiento.
ii. Demuestre que la complejidad del problema de sorting es $\Theta(F_w(n))$.

b) **(8 puntos)** *Complejidad del problema de sorting en el caso medio – cota inferior en caso medio*

Sea $F_A(n)$ la mejor cota inferior del costo en el caso medio para un algoritmo A genérico que ordena una secuencia de $\boldsymbol{D'_n}$ por comparaciones de elementos 2 a 2.

Se sabe que:

- cualquier algoritmo *A* tiene asociado su árbol de decisión (*AD*).
- las secuencias de D'_n son equiprobables (como entrada para el algoritmo *A*).
- el costo del algoritmo *A* en su caso medio se denota $T_A(n)$.
- $\boldsymbol{k_i}$ es el largo del camino de la raíz al nodo externo ***i*** .
- $\boldsymbol{p_i}$ es la probabilidad de alcanzar el nodo externo ***i*** .
- Si ***m*** es la cantidad de **nodos externos**, por el ***lema 2*** visto en el teórico la sumatoria de profundidades de los nodos externos de *AD* es mayor o igual a ***m log m*** .

i. Calcule $F_A(n)$; justificando detalladamente los pasos de su razonamiento.
ii. Demuestre que la complejidad del problema de sorting es $\Theta(F_A(n))$.