Segundo Parcial de Programación 3 (29/11/2013)

Instituto de Computación, Facultad de Ingeniería

- Este parcial dura **4** horas y contiene 3 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* y el tipo `bool` predefinido en `C++`.
- 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.

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 respondió cada problema.

Ejercicio 1. (20 puntos)

La organización Ptres.org cuenta con voluntarios e instituciones que tienen diferentes necesidades. Los voluntarios al inscribirse en la organización deben indicar sus áreas de especialización, para luego ser asignados a necesidades que puedan realizar. Las necesidades tienen una fecha de inicio, una fecha de fin, un área de especialización y la cantidad de voluntarios requeridos. Los voluntarios asignados a cada necesidad deben tener el área de especialización de la misma. La organización asigna la cantidad de voluntarios que la necesidad solicita en caso de contar con los mismos, de lo contrario no asigna ninguno. Los voluntarios sólo pueden atender una necesidad a la vez.
La organización se propone atender a la mayor cantidad posible de necesidades.
Luego de analizar el problema se decidió resolverlo con **Backtracking**. Considerando que hay una cantidad ***V*** de voluntarios y una cantidad ***N*** de necesidades se plantean 3 posibles formas de solución:

1. Tupla de largo **N** fijo $T =< t_1, ..., t_i, ..., t_N >$, donde **N** es la cantidad de necesidades. Cada componente de la tupla, es un vector de **A** elementos $t_i =< x_1, ..., x_j, ... x_A >$, donde **A** es la cantidad de voluntarios asignados a la necesidad **i** y cada elemento x_j indica el identificador de un voluntario que fue asignado a la necesidad.
2. Tupla de largo **N** fijo $T =< t_1, ..., t_i, ..., t_N >$, donde **N** es la cantidad de necesidades. Cada componente de la tupla es un vector de **V** elementos $t_i =< x_1, ..., x_j, ..., x_V >$, donde **V** es la cantidad total de voluntarios y cada elemento x_j indica si el voluntario con identificador **j** fue asignado a la necesidad **i**.
3. Tupla de largo **V** fijo $T =< t_1, ..., t_j, ..., t_V >$, donde **V** es la cantidad total de voluntarios. Cada componente de la tupla es un vector de **N** elementos $t_j =< x_1, .., x_i, ..., x_N >$, donde **N** es la cantidad total de necesidades y cada elemento x_i indica si el voluntario con identificador **j** fue asignado a la necesidad **i**.

a) **(8 puntos)** Para cada forma de solución escriba formalmente las restricciones explícitas e implícitas.
b) **(3 puntos)** Seleccione una de las formas de solución presentadas y escriba para la misma la función objetivo del problema.
c) **(3 puntos)** La organización constató que con el algoritmo construido en base a lo anterior se asignaban muchas veces a los mismos voluntarios a varias necesidades. Como alternativa se propone, dada la misma realidad, maximizar la cantidad de voluntarios asignados en forma global a las necesidades.

   Seleccione una de las formas de solución presentadas y escriba la función objetivo de este nuevo problema.

d) **(6 puntos)** La organización ahora dispone de recursos que pretende asignar a las necesidades y cuenta con una cierta cantidad de cada recurso (por ejemplo: H hormigoneras, X combis, C computadores, etc.). Si un recurso es asignado a una necesidad, el mismo no podrá ser asignado a otra necesidad que inicie antes de finalizar la anterior. Cada institución puede solicitar sólo uno de los recursos de la organización para cada necesidad.

Adapte la forma de solución (2.) y escriba las restricciones explícitas e implícitas para solucionar el problema de atender a la mayor cantidad posible de necesidades asignadas, considerando la disponibilidad de los recursos.

Importante:

Para todas las partes del problema utilizar los siguientes criterios:

- Las necesidades se identifican con un entero en el rango $[1..N]$.
- Los voluntarios se identifican con un entero en el rango $[1..V]$.
- Las áreas de especialización se identifican con un entero en el rango $[1..E]$.
- $necesidades[i].inicio$ y $necesidades[i].fin$ indican el inicio y fin de la tarea i respectivamente.
- $necesidades[i].cantidad$ indica la cantidad de voluntarios solicitados por la institución para realizar las actividades vinculadas a la necesidad.
- $necesidades[i].areaEsp$ indica el identificador del área de especialización de la necesidad.
- $voluntarios[i].areas[j]$ es 1 si el voluntario i cuenta con el área de especialización j, 0 en caso contrario.
- Los recursos se identifican con un entero en el rango $[1..R]$.
- $recursos[i]$ indica la cantidad de recursos i con los que cuenta la organización.
- $necesidades[i].recurso$ indica el identificador del recurso requerido por la necesidad o 0 en caso de no necesitar recursos.

Ejercicio 2. (20 puntos)

Sea *G=(V,E)* un grafo no dirigido, conexo, sin lazos ni aristas múltiples, con costos positivos asociados a sus aristas. La cantidad de vértices es *N* (constante) y están identificados de 1 a *N*.

e) **(5 puntos)** Sean *u* y *v* dos vértices cualesquiera de *G*. Se desea encontrar el camino de menor costo entre ambos vértices.
   1. Es posible encontrarlo aplicando el algoritmo de Kruskal?
   2. Es posible encontrarlo aplicando el algoritmo de Prim?

   En ambos casos demuestre su respuesta o de un contraejemplo.

f) **(5 puntos)** Sean *u* y *v* dos vértices cualesquiera de G. Demuestre que la solución óptima al problema de encontrar el camino de menor costo entre *u* y *v* cumple el Principio de Optimalidad.

g) **(5 puntos)** Se tiene además una matriz *C* donde están los costos de las aristas del grafo, ésta contiene *MAXINT* si no hay arista entre dos vértices y *0* en las posiciones de la diagonal. Considerando que no hay ciclos de costo negativo, plantee una recurrencia usando Programación Dinámica que, dados dos vértices, retorne el costo del camino de menor costo entre ambos vértices.

h) **(5 puntos)**¿Es posible encontrar un vértice *v* del grafo talque la aplicación del algoritmo de Dijkstra, tomando *v* como origen, retorne como resultado un árbol de cubrimiento de costo mínimo del grafo dado?
   Demuestre su respuesta o de un contraejemplo.

Ejercicio 3. (20 puntos)

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 lo que sigue considere que trabaja con D'_n .

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

c) **(6 puntos)** *QuickSort*

Dado el siguiente algoritmo PARTITION el cual elige un elemento llamado *pivote* y reordena el arreglo entre las posiciones *ini* y *fin* redistribuyendo los elementos adecuadamente; el parámetro *pospiv* devuelve la posición final del pìvote.

```
void PARTITION(int *a, int n, int ini, int fin, int &pospiv)
```

i. Escriba el algoritmo de ordenación QuickSort utilizando el PARTITION dado e indique qué necesita que se haga en PARTITION para que su algoritmo funcione.
ii. Indique qué técnica de diseño de algoritmos se utiliza, identificando los distintos ítems de la misma.
iii. Su algoritmo ¿es estable?. Justifique su respuesta.