Segundo Parcial de Programación 3
28 de noviembre de 2014

- Este parcial dura **4 horas** y contiene 2 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 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, 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.

Ejercicio 1 (22 puntos)

Un artesano quiere planificar qué artesanías construir para una gran feria que comenzará en unos días. Cada artesanía de tipo r, con $1 \leq r \leq R$, requiere materiales por un costo c_r pesos, lleva un tiempo de preparación de t_r horas y genera al momento de la venta v_r pesos. El artesano cuenta con C pesos para comprar materiales y H horas de trabajo antes del comienzo de la feria. Por un tema de exclusividad, no pueden prepararse más de E artesanías del mismo tipo.

Se desea planificar la cantidad de artesanías de cada tipo a fabricar antes del comienzo de la feria, de manera de maximizar la ganancia total. La ganancia de una artesanía r es v_r–c_r . Asumir: $c_r < v_r$ para toda artesanía r y que toda artesanía fabricada será vendida.

Se pide:

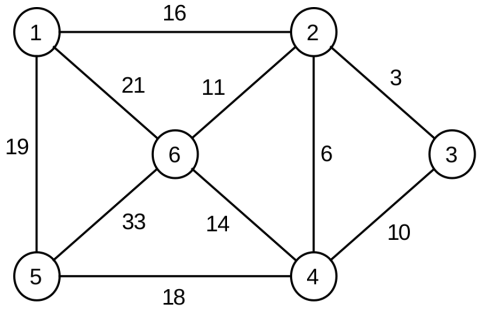
1. Formalizar el problema para la estrategia de **Backtracking** indicando: forma de la tupla, restricciones implícitas y explícitas, y función objetivo si corresponde (además explique todos los ítems en lenguaje natural). No se pide que especifique los predicados de poda.
2. Argumente de forma breve si las siguientes condiciones podan el espacio de soluciones y además si son o no predicados de poda (considere r como la artesanía actual, es decir la siguiente a planificar):
   a) La ganancia acumulada hasta la planificación de la artesanía r inclusive, es menor que la ganancia de la mejor solución encontrada hasta el momento.
   b) La ganancia acumulada hasta la planificación de la artesanía r inclusive, sumada al valor más optimista para el resto de la planificación, es menor o igual que la ganancia de la mejor solución encontrada hasta el momento. ¿Cuál es el valor más optimista?
   c) No considerar como opción a preparar una determinada cantidad de artesanías del tipo r si, al incluirlas, el costo acumulado de la planificación es mayor que C.
3. Dibuje el árbol de soluciones para el caso de 4 artesanías. Determine convenientemente los parámetros del problema de forma que en el árbol se muestre el efecto de los diferentes ítems que surjan de la formalización (incluyendo predicados de poda de la parte 2, si existen).

Ejercicio 2 (23 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.

1. a) Defina el concepto de árbol de cubrimiento de G.
   b) Defina el concepto de árbol de cubrimiento de costo mínimo de G.
2. Demuestre la propiedad **MST** (***M**inimum **S**panning **T**ree*) vista en el teórico:

   Sea U un subconjunto de los vértices V, si (u, v) es una arista de costo mínimo tal que $u \in U$ y $v \in V - U$ entonces existe un árbol de cubrimiento de costo mínimo que incluye a (u, v) como una de sus aristas.
3. a) Explique la utilidad del algoritmo de Prim.
   b) Escriba el algoritmo de Prim.
   c) Indique la técnica de diseño de algoritmos que utiliza y demuestre por qué es aplicable la misma.
4. Dibujar la ejecución del algoritmo de Prim, paso a paso, para el siguiente grafo:



5. Explique la utilidad del algoritmo de Dijkstra y escríbalo.

Ejercicio 3 (15 puntos)

1. Escriba el algoritmo **MergeSort** visto en el curso. Puede asumir implementada la siguiente función `Merge`:

   ```
   void Merge (int* v, int ai, int af, int bi, int bf);
   ```

   Precondición:

   I) $0 \leq$ `ai` $\leq$ `af` $<$ `bi` $\leq$ `bf` $<$ `n`, siendo `n` la cantidad de elementos de `v` (`v` es un array).
   II) Las subsecuencias `v[ai], ..., v[af]` y `v[bi], ..., v[bf]` están ordenadas.

   Postcondición:

   I) La secuencia `v[ai], ..., v[af], v[bi], ..., v[bf]` está ordenada.
2. Deduzca el tiempo de ejecución para el peor caso.
3. Deduzca el tiempo de ejecución para el caso medio.

Puede suponer que no hay elementos repetidos en la secuencia y que `n` es potencia de 2.
El costo de Merge es: `af-ai+bf-bi+1` (la suma de los largos de ambas secuencias menos 1).