Análisis de Algoritmos (I)

Profesor: Carlos Zerón Martínez
Ayudante: Edwin Antonio Galván Gamez

Tarea 4: Diseño de Algoritmos
Fecha de entrega: Martes 2 de Abril del 2019.

1. **Ejercicio teórico - práctico**. Considera el siguiente algoritmo *greedy* para el problema de calendarización de todas las tareas de un conjunto dado con el mínimo número de recursos, de modo que cualesquiera dos tareas asignadas al mismo recurso no se traslapen entre sí.

Listado 1 scheduleAll(REQ)

```
1:  array of lists SCH[1,...,n];
2:  min = 1; // número de recursos
3:  WAIT = REQ;
4:  while WAIT ≠ ∅ do
5:    req_i = deleteMinStarting(WAIT);
6:    asignado = false;
7:    for j = 1 to min do
8:      if ∀req_k ∈ SCH[j] : areCompatible(req_k, req_i) then
9:        SCH[j].add(req_i);
10:       asignado = true;
11:       break;
12:     end if
13:   end for
14:   if asignado = false then
15:     min := min + 1;
16:     S[min].add(t_i);
17:     asignado = true;
18:   end if
19: end while
20: return SCH;
```

Las precondiciones y postcondiciones del algoritmo a partir del problema son las siguientes:

Precondiciones: $REQ = \{req_1, req_2, \ldots, req_n\}$ conjunto de $n \geq 1$ tareas, donde cada tarea req_i tiene una descripción con un identificador y un intervalo constituido por su tiempo de inicio $start(i)$ y su tiempo final $end(i)$, con $start(i) < end(i)$, y REQ está ordenado en forma no decreciente por tiempo de inicio.

Postcondiciones: Un arreglo SCH de n listas que representa la calendarización de REQ tal que

a) Para toda tarea req_i donde $i = 1, \ldots, n$ existe un $j \in \{1, \ldots, min\}$ tal que la lista $SCH[j]$ contiene a req_i.

b) Cualesquiera dos tareas req_i y req_k en una lista $SCH[j]$ con $1 \leq j \leq min$ son compatibles, es decir, $end(k) \leq start(i)$ o $end(i) \leq start(k)$.

c) min es el número mínimo de listas no vacías para las cuales se puede garantizar el cumplimiento de las dos condiciones anteriores.

La operación add invocada por una lista inserta un elemento al final de la misma, el procedimiento o subrutina deleteMinStarting elimina una tarea con mínimo tiempo inicial del conjunto de tareas que le pasan y la subrutina areCompatible verifica la compatibilidad entre dos tareas. Implementa en Java el algoritmo completo, suponiendo que el conjunto de tareas satisface las precondiciones, y analiza su complejidad. El progama deberá generar $n = 10k$ tareas ordenadas en forma no decreciente por tiempo inicial, manejando tiempo inicial y final como números enteros, donde $k = 1, \ldots, 100$ y reportar la calendarización obtenida. El archivo con las instrucciones para ejecutar el programa deberá incluir el análisis de complejidad teórico.

2. Explica con detalle la Sección 2.9 del libro de Udi Manber, Introduction to Algorithms. A creative approach que versa sobre el problema de la construcción de *Códigos Gray*. Como mínimo, la explicación debe incluir: la definición de un Código Gray, la construcción inductiva de la solución usando códigos con un número logarítmico de bits sobre el número de objetos a representar y la ilustración de la construcción para 22 y 23 objetos.

3. Considera el algoritmo para la variante del problema de la mochila (*Knapsack problem*) que viene descrito en la Sección 5.10 del libro de Udi Manber, Introduction to Algorithms. A creative approach. Diseña un algoritmo que recupere la solución del problema usando la bandera belong del resultado obtenido de dicho algoritmo y justifica su integridad.

4. Una persona considera abrir una serie de restaurantes a lo largo de una carretera recta. Hay n posibles ubicaciones sobre ella y las distancias de las ubicaciones posibles al punto inicial de la carretera están dadas en kilómetros y en orden creciente $dist_1, dist_2, \ldots, dist_n$. Se tienen las siguientes restricciones:

   - En cada ubicación i se puede abrir a lo más un restaurante y la ganancia esperada de hacerlo se mide en $p_i > 0$ pesos, para $i = 1, \ldots, n$.
   - Cualesquiera dos restaurantes deben estar separados al menos por una distancia de k kilómetros, con k entero positivo.

   El objetivo consiste es determinar en qué posiciones deben abrirse los restaurantes para maximizar la ganancia esperada total. Usando inducción matemática, diseña un algoritmo que resuelva el problema y determina su complejidad. Puedes usar programación dinámica para la descripción definitiva del algoritmo.

Suerte!