UNIVERSIDAD DE CHILE
ESCUELA DE INGENIERIA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACION

Diseño y Análisis de Algoritmos – CC4102
Control 1 - Semestre Primavera 2014

1. Considere un algoritmo que verifica si un grafo no dirigido sin loops es acíclico utilizando solo preguntas de la forma: ¿existe un arco entre los vértices u y v? Demuestre que este algoritmo debe realizar $\binom{n}{2}$ preguntas en el peor caso.

   Hint: Utilice una estrategia del adversario basada en mantener las componentes conexas conocidas en cada momento, y responder sí o no a las preguntas del algoritmo dependiendo de si los vértices u, v pertenecen a la misma componente conexa. En caso que el algoritmo responda que el grafo es acíclico, construya una peor entrada cíclica para el algoritmo dependiendo del tamaño de la componente conexa C más grande según los siguientes casos: (1) C contiene un solo vértice, (2) C contiene todos los vértices, o (3) C contiene al menos dos vértices pero no los contiene todos.

   Solución: Asuma por contradicción que para algún $n \geq 3$ se tiene que el algoritmo realiza $x < \binom{n}{2}$ preguntas sobre entradas de tamaño n. El adversario utilizará la siguiente estrategia: Mantiene en memoria todas las componentes conexas conocidas, y responde que u está conectado con v si y solo si u y v pertenecen a componentes conexas distintas. Esto quiere decir que en cada momento el grafo G conocido por el algoritmo es acíclico. Demostraremos que después de las x preguntas realizadas por el algoritmo es posible encontrar grafos G_1 y G_2 consistentes con todas las respuestas entregadas hasta el momento por el adversario, tal que G_1 es acíclico y G_2 no lo es. Dependiendo de la respuesta dada por el algoritmo tendremos que G_1 o G_2 es una entrada de peor caso para el algoritmo.

   Dado que G es acíclico, existe grafo G_1 acíclico que es consistente con todas las respuestas dadas por el adversario (ya que podemos tomar $G = G_1$). Construiremos ahora G_2 dependiendo de la cardinalidad de la componente conexa C más grande que pertenece a G: (1) Si C tiene un solo vértice quiere decir que $x = 0$. Podemos entonces definir a G_2 como el clique de n elementos. (2) Si C contiene a todos los vértices, entonces existe un par u, v de vértices en C tal que el algoritmo aún no pregunta por si existe un arco entre ellos. En este caso G_2 puede definirse como el grafo obtenido desde G al agregar el arco (u, v). (3) C contiene al menos dos vértices, pero no los contiene a todos. Tome dos nodos arbitrarios u, v en C y un nodo w afuera de C. El algoritmo aún no ha preguntado si existe arco entre u y w, ni tampoco si existe arco entre v y w (porque debido a la estrategia del adversario en caso contrario tendríamos que w pertenece a C). Luego, podemos definir G_2 como el grafo obtenido desde G agregando los arcos (u, w) y (v, w). Claramente G_2 contiene un ciclo.

2. Considere un problema de ordenamiento en que los números no son conocidos a priori. En cambio, sí conocemos para cada número un intervalo real al que pertenece. Esto es, la entrada del problema corresponde a n intervalos de la forma $[a_i, b_i]$, donde a_i, b_i son reales tal que $a_i \leq b_i$. El objetivo del problema es encontrar una permutación π sobre $\{1, \ldots, n\}$ que satisface lo siguiente: existen reales $c_1 \leq c_2 \leq \cdots \leq c_n$ tal que $c_i \in [a_{\pi^{-1}(i)}, b_{\pi^{-1}(i)}]$ para cada $1 \leq i \leq n$.

Diseñe un algoritmo para solucionar este problema que tenga la misma estructura general que *Quicksort*, pero que trate de aprovecharse de cuando hay intervalos que se intersectan para mejorar el tiempo de ejecución. En particular, cuando existe un elemento que pertenece a todos los intervalos entonces su algoritmo debiera trabajar en tiempo $O(n)$. (Note que el algoritmo no debe chequear por este caso por separado, sino que su tiempo de ejecución debe mejorar naturalmente en la medida que la cantidad de intersecciones entre intervalos aumenta).

Solución: Una posible solución es la siguiente. Elija un pivote p de la misma forma que en Quicksort. Primero separamos a aquellos intervalos que no se intersectan con p, y los dividimos según corresponda a la izquierda o derecha de este. Sea I el conjunto de intervalos que no se intersectan con p y que deben ser menores a p, y D aquellos que deben ser mayores.

Luego empleamos una estrategia greedy: Elegimos un intervalo al azar de entre los que quedan y computamos su intersección $[a, b]$ con p. Luego tomamos otro intervalo entre los restantes y vemos si se entersecta con $[a, b]$. Si es así computamos su intersección $[a', b']$ con $[a, b]$. En caso contrario lo agregamos al conjunto I o D según corresponda (e.g., si el límte derecho del intervalo es menor que a entonces el intervalo se agrega a I). Continuamos este proceso hasta que hemos prcesado todos los intervalos que se intersectan con p.

Una vez que terminamos tenemos un intervalo $[a^*, b^*]$ en $[a, b]$ y un conjunto de intervalos todos los cuales contienen a $[a^*, b^*]$. Para estos intervalos simplemente imponemos un orden cualquiera (ya que cualquier orden es posible). Luego recursivamente implementamos el algoritmo en I y en D.

Es fácil ver que cuando todos los intervalos comparten un elemento entonces el algoritmo trabaja en tiempo $O(n)$.

3. Considere la siguiente estrategia para buscar un elemento x en un arreglo A no ordenado: Elija un índice i al azar. Si $A[i] = x$, entonces terminamos. De otra forma, seguimos la búsqueda eligiendo un nuevo índice j al azar hasta que encontramos x. Note que en cada paso elegimos sobre el conjunto completo de índices, por lo que podríamos examinar un elemento más de una vez.

   a) (4 pts) Asuma que existe exactamente un índice i tal que $A[i] = x$. ¿Cuál es el número esperado de pasos que el algoritmo realizará hasta encontrar x?

   b) (2 pts) ¿Qué sucede si hay $k \geq 1$ índices i tal que $A[i] = x$?

   Hint: Recuerde que $\sum_{k=1}^{\infty} kx^k = \frac{x}{(1-x)^2}$ para $|x| < 1$.

   Solución: Consideremos (a) primero. Sea X el número de comparaciones realizadas por el algoritmo. Estamos entonces interesados en conocer $E(X)$. Por definición, $E(X) = \sum_{i \geq 1}^{\infty} i \cdot Pr(X = i)$. Además, se puede observar que $Pr(X = i) = (n - 1/n)^{i-1} \cdot 1/n$. Por tanto,

$$E(X) = \frac{\sum_{i \geq 1}^{\infty} i(n - 1/n)^{i-1}}{n} = \frac{\sum_{i \geq 1}^{\infty} i(n - 1/n)^{i}}{n - 1}.$$

   Claramente, $n - 1/n < n$, y por tanto, $E(X) = n$.

   Para la parte (b) utilizamos el mismo razonamiento, solo que en este caso $E(X) = \frac{k}{n} \cdot \sum_{i \geq 1}^{\infty} i(n - k/n)^{i-1}$. Por tanto, $E(X) = n/k$.