Pontificia Universidad Católica de Chile
Facultad de Ingenieria
Departamento de Ciencias de la Computación
IIC2283 - Diseño y Análisis de Algoritmos

Ayudantía 6

28 de septiembre, 2018

Ayudante: Andrés Ríos

Problema 1

En una lista, se define un *elemento mayoritario* como un valor que aparece repetido al menos en la mitad de las posiciones.

1. Dada una lista de enteros de largo N, desarrolle y analice un algoritmo determinístico que decida si existe un elemento mayoritario, y en ese caso retorne ese valor.

   Solución:

   Existen varias soluciones que funcionan en tiempo $O(n \log(n))$. Una de ellas es:

   ```
   function Encontrar_Mayoritario(A[1...n])
       d = árbol de búsqueda rojo-negro
       for i = 1...n do
           if A[i] not in d then
               d.insert(A[i], 0)
           d[A[i]] = d[A[i]] + 1
       c = llave con el valor asociado más alto en d
       if d[c] ≥ n/2 then
           return c
       else
           return None
   ```

   Básicamente, se mantiene un contador de la frecuencia absoluta de cada elemento dentro del árbol de búsqueda d. Recorrer el arreglo para computar la frecuencia absoluta de cada elemento toma $O(n \log(n))$ (pues buscar e insertar en un árbol rojo-negro es $O(\log(n))$. Una vez hecho esto, determinar el valor de c y luego ver si es mayoritario toma $O(n)$. Por lo tanto, todo el algoritmo corre en $O(n \log(n))$.

2. Desarrolle y analice un algoritmo aleatorizado que resuelva el mismo problema en tiempo lineal.

   Solución:

   La idea es simple: samplear un elemento aleatorio de A y luego contar cuantas veces aparece en el arreglo. Si aparece más de $n/2$ veces, lo retorno. En otro caso, retorno $None$. Este algoritmo toma tiempo lineal: samplear toma tiempo a lo más logarítmico y recorrer el arreglo una vez contando cuantas veces aparece el elemento toma tiempo lineal.

   Sobre la probabilidad de error, notemos que cuando no existe un elemento mayoritario entonces el algoritmo siempre retorna $None$ y no se equivoca. Por el otro lado, cuando sí existe un elemento mayoritario, entonces la probabilidad de samplearlo es mayor o igual a $1/2$. Por lo tanto, la probabilidad de equivocarse cuando sí existe un elemento mayoritario es menor o igual a $1/2$. En total esto significa que la probabilidad de error es menor o igual a $1/2$.

Problema 2

Dadas matrices $A, B, C \in \mathbb{Q}^{n \times n}$, se desea determinar si $A \cdot B = C$

1. Proponga un algoritmo deterministico para resolver el problema y acote su tiempo de ejecución

   Idea de Solución:

   La solución más simple es calcular el resultado de $A \cdot B$ y luego compararlo con C. Si son iguales retornar $true$, en otro caso retornar $false$.

   La multiplicación de matrices toma tiempo $w(n^2)$. Por lo tanto, esta cota aplica también para nuestro algoritmo determinista.

2. Proponga y analice un algoritmo aleatorizado para resolver el problema que tome tiempo $O(n^2)$

   Solución:

   El algoritmo es:

   a) Generar un vector aleatorio $r \in \{0,1\}^n$

   b) Calcular $P = A \cdot B \cdot r - C \cdot r$

   c) Si $P = (0, 0, ..., 0)$ retornar $true$, en otro caso retornar $false$

   Acotemos la probabilidad de error:

   - Si se tiene que $A \cdot B = C$, entonces el algoritmo nunca se equivoca por lo que la probabilidad de error es 0
   - Si $A \cdot B \neq C$, sea $D = A \cdot B - C$, donde $d_{i,j}$ es el elemento en la fila i y la columna j de D. Sea $P = D \cdot r = (p_1, ..., p_n)$. La probabilidad de que el algoritmo se equivoque es igual a la probabilidad de que $P = (0, ..., 0)$.

     Dado que $A \cdot B \neq C$, sabemos que existe algun elemento de D distinto de 0. Supongamos que $d_{i,j} \neq 0$. Tenemos que

     $$p_i = \sum_{k=1}^{n} d_{i,k} r_k = d_{i,1} r_1 + ... + d_{i,j} r_j + ... + d_{i,n} r_n = d_{i,j} r_j + y$$

     para alguna constante y. Por el teorema de probabilidades totales tenemos que

     $$Pr[p_i = 0] = Pr[p_i = 0 | y = 0] \cdot Pr[y = 0] + Pr[p_i = 0 | y \neq 0] \cdot Pr[y \neq 0]$$

     Además,

     $$Pr[p_i = 0 | y = 0] = Pr[r_j = 0] = \frac{1}{2}$$

     $$Pr[p_i = 0 | y \neq 0] = Pr[r_j = 1 \wedge d_{i,j} = -y] \leq Pr[r_j = 1] = \frac{1}{2}$$

     Reemplazando se obtiene que

     $$\begin{aligned} Pr[p_i = 0] &\leq \frac{1}{2} Pr[y = 0] + \frac{1}{2} Pr[y \neq 0] \\ &= \frac{1}{2}(Pr[y = 0] + Pr[y \neq 0]) \\ &= \frac{1}{2} \end{aligned}$$

     Por lo tanto,

     $$Pr[P = (0, ..., 0)] = Pr[p_1 = 0 \wedge ... \wedge p_i = 0 \wedge ... \wedge p_n = 0] \leq Pr[p_i = 0] \leq \frac{1}{2}$$

     Con esto se concluye que la probabilidad de que el algoritmo se equivoque es menor o igual a $\frac{1}{2}$.

Problema 3

Dados dos polinomios $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ en $\mathbb{Q}$, decimos que $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ son equivalentes si para toda secuencia $a_1, ..., a_n$ de números en $\mathbb{Q}$ se tiene que $p(a_1, ..., a_n) = q(a_1, ..., a_n)$.

Dados dos polinomios $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ en $\mathbb{Q}$, decimos que $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ son linealmente-equivalentes si existe un polinomio $r(x_1, ..., x_n)$ tal que: (i) $r(x_1, ..., x_n)$ es nulo o es un polinomio de grado menor o igual a 1, y (ii) $p(x_1, ..., x_n)$ es equivalente a $q(x_1, ..., x_n) + r(x_1, ..., x_n)$. Por ejemplo, los polinomios $p(x, y) = x(x-1) + 2y$ y $q(x, y) = (x-1)(x-2) + y + 1$ son linealmente-equivalentes puesto que

$$p(x, y) = q(x, y) + r(x, y)$$

donde $r(x, y) = 2x + y - 3$.

De un algoritmo aleatorizado que resuelva el problema de determinar si dos polinomios $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ son linealmente-equivalentes. Su algoritmo debe funcionar en tiempo polinomial en el peor caso y su probabilidad de error debe ser menor o igual a $\frac{1}{2^{100}}$. Además, usted debe justificar claramente por qué su algoritmo es correcto, funciona en tiempo polinomial y tiene probabilidad de error acotada por $\frac{1}{2^{100}}$.

Idea de Solución:

La idea es la siguiente: para que p sea linealmente-equivalente con q debe darse que $p(x_1, ..., x_n) - q(x_1, ..., x_n)$ sea equivalente a un polinomio de grado a lo más 1. Por lo tanto, lo que podemos hacer es asumir que $p - q$ es equivalente a un polinomio de grado 1, determinar los coeficientes de este polinomio, y luego ver si el polinomio obtenido es realmente equivalente a $p - q$. Si el polinomio obtenido es equivalente entonces podemos retornar que p y q son linealmente-equivalentes. En caso contrario podemos retornar que no.

El algoritmo en pseudo-código es:

function LINEALMENTE_EQUIVALENTES(p, q)
  $r = p - q$
  $coefs[0...n] = [0, ..., 0]$
  $coefs[0] = r(0, ..., 0)$
  for $i = 1..n$ **do**
    $v = (0, ..., 0)$
    $v[i] = 1$
    $coefs[i] = r(v) - coefs[0]$
  $rp(x_1, ..., x_n) = coefs[0] + coefs[1] \cdot x_1 + ... + coefs[n] \cdot x_n$
  return EquivPolAleatorizado(r, rp)

donde EquivPolAleatorizado es el algoritmo visto en clases para determinar la equivalencia entre polinomios.

Si p y q son linealmente-equivalentes, entonces $r = p - q$ es equivalente a un polinomio de la forma

$$rp(x_1, ..., x_n) = c_0 + c_1 x_1 + ... + c_n x_n$$

Lo que hace el algoritmo es evaluar $r = p - q$ de manera inteligente para determinar los coeficientes $c_0, c_1, ..., c_n$ de rp. Los coeficientes se guardan en la variable $coefs$, donde $coefs[i] = c_i$. Luego si r es equivalente a rp, entonces p y q definitivamente son linealmente-equivalentes. Es fácil demostrar que la otra dirección también es cierta: si p y q son linealmente-equivalentes, entonces r es equivalente a rp. En resumen,

$$p \text{ y } q \text{ son linealmente equivalentes } \iff r = p - q \text{ es equivalente a } rp$$

Por lo tanto, la probabilidad de error del algoritmo Linealmente_Equivalentes es igual a la probabilidad de error de EquivPolAleatorizado: menor o igual a 1/100. Luego para llegar a la cota pedida basta repetir el algoritmo 16 veces. Además, el algoritmo es polinomial pues evaluar un polinomio toma tiempo polinomial y el algoritmo EquivPolAleatorizado también es polinomial.

Problema 4

Dados dos polinomios $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ en $\mathbb{Q}$, decimos que $p(x_1, ..., x_n)$ y $q(x_1, ..., x_n)$ son equivalentes si para toda secuencia $a_1, ..., a_n$ de números en $\mathbb{Q}$ se tiene que $p(a_1, ..., a_n) = q(a_1, ..., a_n)$.

Decimos que un polinomio $r(x_1, ..., x_n)$ es una forma cuadrática si

$$r(x_1, ..., x_n) = \sum_{i=1}^{n} \sum_{j=i}^{n} a_{i,j} x_i x_j$$

donde cada $a_{i,j} \in \mathbb{Q} (1 \leq i, j \leq n)$

De un algoritmo aleatorizado que resuelva el problema de determinar si un polinomio $p(x_1, ..., x_n)$ es equivalente a una forma cuadrática. Su algoritmo debe funcionar en tiempo polinomial en el peor caso y su probabilidad de error debe ser menor a $\frac{1}{2^{100}}$. Además, usted debe justificar claramente por qué su algoritmo es correcto, funciona en tiempo polinomial y tiene probabilidad de error acotada por $\frac{1}{2^{100}}$.

Idea de Solución:

La idea es exactamente la misma que el problema anterior: construir un polinomio rp tal que la equivalencia de rp con r determine la propiedad que estamos buscando. El algoritmo es:

function FORMA_CUADRÁTICA(r)
  $coefs[1...n, 1...n] = [0, ..., 0]$
  for $i = 1...n$ **do**
    $v = (0, ..., 0)$
    $v[i] = 1$
    $coefs[i, i] = r(v)$
  for $i = 1...n$ **do**
    for $j = i + 1...n$ **do**
      $v = (0, ..., 0)$
      $v[i] = 1$
      $v[j] = 1$
      $coefs[i, j] = r(v) - coefs[i, i] - coefs[j, j]$
  $rp =$ Construir polinomio cuadrático con coeficientes $coefs$
  return EquivPolAleatorizado(r, rp)

Al igual que el algoritmo anterior, la idea es asumir que r es equivalente a una forma cuadrática y determinar sus coeficientes. Los coeficientes se guardan en la matriz $coefs$, donde $coefs[i, j] = a_{i,j}$. Luego utilizando los coeficientes se construye rp y se chequea la equivalencia.

Al igual que antes, para alcanzar la cota pedida hay que repetir el algoritmo 16 veces.