Universidad del Valle de Guatemala Christopher Sandoval 13660
Análisis y Diseño de Algoritmos - Sección 10 13/05/2020

Tarea 4
Análisis probabilístico y algoritmos aleatorizados

1. **Describa una implementación de un generador de números enteros aleatorios dentro del intervalo [*a,b*], llamada random(a,b). Su generador debe estar basado en llamadas a otro generador de bits aleatorios (*p*(0) = *p*(1) = $\frac{1}{2}$, donde *p* es la función de probabilidad) llamado random(0,1). Luego, calcule el tiempo de ejecución esperado de su algoritmo.**

   Comenzamos la implementación por establecer la cantidad de veces que queremos generar números enteros aleatorios; esta cantidad de veces viene dada por (b-a). Ahora, deseamos que la suma de estas probabilidades sea igual (o menor) a 1, pero no sabemos cuántas probabilidades hay que sumar, así que lo haremos k cantidad de veces. Agregando a esto, hay que tomar en cuenta la probabilidad que sea 0 o 1 del generador de bits aleatorios. En conclusión, se tiene la ecuación $(b-a)*(\frac{1}{2})^k = 1$. Al despejar para la cantidad k de veces que queremos ejecutar este algoritmo, obtenemos que $k = log_2(b-a)$. Es decir, esta cantidad k es el número de bits con el que podemos escribir (b-a) números.

   El tiempo de ejecución es de $\Theta(log_2(b-a))$. Este algoritmo es constante en cada bit, pero le toma k ejecutarse cada vez.

2. **Dado un generador de bits llamado biased-random() que produce 1 con probabilidad *p* y 0 con probabilidad 1 − *p* (donde 0 < *p* < 1), diseñe un algoritmo que use biased-random() para generar 1 o 0, ambos con probabilidad $\frac{1}{2}$. Calcule el tiempo de ejecución esperado.**

   Para este nuevo algoritmo, únicamente nos interesa cuando el generador biased-random() dé como resultado dos bits diferentes si se ejecuta dos veces. En este contexto, debemos conocer la probabilidad de que salga 0 y 1 o de que salga 1 y 0 en estas ejecuciones. Cada ejecución no depende del valor anterior, por lo que son independientes y las probabilidades solamente se multiplicarán. La probabilidad de que los resultados sean 1 y 0 es $p(1-p)$; la probabilidad de que los resultados sean 0 y 1 es $(1-p)p$. Al nuevo algoritmo solo le interesa saber cuando las dos ejecuciones den como resultado 1,0 o 0,1; por lo que dará 1 y 0 en cada caso respectivamente. Si los resultados dan 0,0 o 1,1 no se tomarán en cuenta y deberá seguir hasta que sean diferentes.

   Este nuevo algoritmo continuará ejecutándose hasta que los resultados consecutivos sean diferentes. Al considerar esto, se deben sumar las probabilidades de estos dos casos que se obtuvieron anteriormente. Esto da como resultado una probabilidad de $2p(1-p)$. Entonces, el tiempo de ejecución únicamente de nuestro algoritmo sería de

$\frac{1}{2p(1-p)}$. Obtendremos un tiempo de ejecución total de $O(\frac{tiempo\ biased-random}{2p(1-p)})$, si multiplicamos el tiempo de ejecución del primer algoritmo en el cual nos apoyamos.

3. **¿Cuáles son las probabilidades de obtener el best-case scenario (contratar sólo una vez) y el worst-case scenario (contratar *n* veces) si la distribución del input del hiring problem es aleatoria uniforme?**

   Usaremos el concepto de permutaciones para este análisis. En el best-case, el solo contratar una vez implica que el primer candidato de la lista es el mejor de todos. Al encontrarnos con $(n-1)!$ permutaciones (la cantidad de candidatos en total), la probabilidad de que solo contratemos una vez es de $\frac{1}{n}$.

   Por otro lado en el worst-case, consideramos contratar n veces porque los candidatos están en orden ascendente y el mejor es el último. Como solo puede suceder de una forma, la probabilidad de contratar a todos los candidatos, es decir n veces, es de $\frac{1}{n!}$.

4. **Use variables aleatorias para calcular la suma esperada de *n* dados lanzados a la vez.**

   Para calcular la suma de los puntos resultantes en cada dado lanzado, se usará la siguiente ecuación:

   $$S = \Sigma_{i=1}^{n} c_i$$

   donde c_i es el resultado de la cara del dado i.

   Entonces, la suma esperada viene dada por

   $$E[S] = E[\Sigma_{i=1}^{n} c_i] = \Sigma_{i=1}^{n} E[c_i]$$

   Ahora, debemos encontrar el valor esperado de el resultado de la cara del dado i. Como sabemos que el dado sigue una distribución uniforme, es decir que no da preferencia a un lado de la cara y todos tienen probabilidad de caer de ⅙, se tiene que:

   $$E[c_i] = \Sigma_{k=1}^{6} k * P$$
   $$E[c_i] = \Sigma_{k=1}^{6} k * \frac{1}{6} = \frac{21}{6}$$

   Sabiendo ya el valor esperado de cada dado, sustituímos en la ecuación de sumatoria anterior y obtenemos el valor esperado de la sumatoria total:

   $$E[S] = \Sigma_{i=1}^{n} E[c_i] = \Sigma_{i=1}^{n} \frac{21}{6} = \frac{21n}{6}$$

5. **Suponga una lista *A* con una permutación aleatoria de los números [1..*n*]. Una inversión es una pareja (*i,j*) donde *i* < *j* pero *A*[*i*] > *A*[*j*]. Use variables aleatorias indicadoras para calcular el número esperado de inversiones en *A*.**

   La variable aleatoria indicadora que usaremos tomará cada pareja (i, j) y nos regresará 1 si existe una inversión y 0 si no. Como es una lista A, el número esperado de inversiones viene dado al sumar el valor esperado de cada variable aleatoria indicadora de cada pareja (i, j). Con lo anterior se tiene lo siguiente:

   $$E[S] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} E[S_{i,j}] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} P(A[i] > A[j])$$

   donde P es la probabilidad de que suceda A[i] > A[j]

Para poder saber el valor de la probabilidad, debemos averiguar primero las permutaciones de los elementos restantes y cuántas parejas distintas podemos obtener de estos elementos. La ecuación de permutaciones que cumple la condición de A[i] < A[j] será:

$$\left(\frac{n}{2}\right)(n-2)! = \left(\frac{n!}{2!(n-2)!}\right)(n-2)! = \frac{n!}{2}$$

Conociendo ya la permutación del evento, procedemos a calcular la probabilidad del mismo evento, dividiendo la permutación entre las posibles opciones. Esto es:

$$\left(\frac{\frac{n!}{2}}{n!}\right) = \frac{n!}{2n!} = \frac{1}{2}$$

Finalmente reemplazamos el valor obtenido en la primera ecuación para obtener el número esperado de inversiones en A:

$$E[S] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} = \frac{1}{2}\left(\frac{n}{2}\right) = \frac{n(n-1)}{4}$$

6. **Responda las siguientes preguntas:**
   a. **Explique la fórmula usada para calcular $P[I_k]$ en la prueba del teorema 2.**
      Sabemos que para calcular la probabilidad, debemos encontrar el número de permutaciones del evento y dividirlo dentro del total de posibles ordenamientos de todo el arreglo. Primero, las combinaciones de que k números estén ordenados de 1 hasta n al inicio del arreglo son (n 2). A esto debemos agregarle las combinaciones de que el resto de números estén en cualquier orden, dándonos como resultado (n - k)!. El resultado final teniendo ya la permutación es de 1/k!, siendo k! El total de posibles ordenamientos de todo el arreglo.
   b. **En la prueba del teorema 2, ¿por qué $E[C] = \Sigma_{k>0} P[I_k]$?**
      El teorema 2 considera los siguientes aspectos. Primero, toma a C como una variable aleatoria que indica el número de comparaciones que determinan si el arreglo está ordenado o no. Luego, la variable I_k devuelve 1 si los primeros k números del arreglo sí están ordenados. Una variable depende de la otra, ya que para determinar si están ordenados, debemos comparar cada uno de estos primeros elementos. Esto quiere decir que es equivalente decir Ik = 1 y C ≥ k. Ingresando estas suposiciones al calcular E[C], llegamos a la igualdad indicada.
   c. **En la sección 2.3, ¿por qué la variable aleatoria I que cuenta el número de iteraciones del algoritmo tiene distribución geométrica?**
      Porque tiene las siguientes características la variable I: cada iteración es independiente de la anterior porque la permutación es al azar y la variable contará cuántas veces falló el algoritmos antes de indicar el resultado correcto.

7. **Responda las siguientes preguntas:**

a. **El algoritmo guess-sort presenta una mejora con respecto a bozo-sort$^{+}_{opt}$ (sección 3 del paper del inciso anterior). ¿Cuál es la diferencia que permite esta mejora?**

   El algoritmo guess-sort, a diferencia del bozo-sort, se ahorra n-1 comparaciones en cada iteración al no verificar que el arreglo esté ordenado después de intercambiar las posiciones. Solo verifica si está arreglado hasta la n-ésima repetición.

b. **En las conclusiones, una pregunta sugiere que el tiempo de ejecución de Fun-sort en el average-case ha de ser más o menos rápido gracias al teorema 6. Si las condiciones del teorema 6 se cumplen, ¿cuál sería el tiempo de ejecución esperado?**

   Sabemos que el tiempo de ejecución promedio de este algoritmo es de $O(\log_2 n)$. Según el teorema 6, el resultado esperado es la posición que le corresponde al elemento buscado. Para cambiarlo de posición a la correcta, la búsqueda binaria debe ser exitosa. Entonces, para cada elemento n del arreglo, debemos hacer una búsqueda binaria. Esto tomaría el tiempo de ejecución esperado de $O(n*\log_2 n)$.