Universidad del Valle de Guatemala
Análisis y Diseño de Algoritmos - Sección 10
María Fernanda Estrada 14198
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.**
   Se necesita un generador de 0's y 1's en un intervalo [a, b], por lo que podemos decir que queremos generar (b - a) cantidad de números. Sin embargo, sabiendo que la probabilidad que dé 0 o 1 es de ½, debemos multiplicar ese factor. También, como la suma de estas probabilidades debe ser igual a 1 y no sabemos la cantidad de veces que se dará esta suma, elevamos a la x esta multiplicación e igualamos a 1. Al despejar, nos queda la siguiente ecuación:
   $$(b-a) * \frac{1}{2^x} = 1$$
   $$(b-a) = 2^x$$
   $$log_2(b-a) = x$$
   Entonces, $log_2(b-a)$ es la cantidad de veces que se generarán (b - a) números aleatorios deseados. Deberá repetirse esta fórmula hasta encontrar el número entre el rango de [a, b].
   Ya que el algoritmo deberá ejecutarse $log_2(b-a)$ una cierta cantidad de veces, y que generar cada bit será constante, el tiempo de ejecución sería $\Theta(log_2(b-a))$.

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.**
   Contamos con que las ejecuciones de este algoritmo sean independientes, por lo que las probabilidades se multiplican. Entonces, la probabilidad que dé 1 en ambas ocasiones es p^2 y la probabilidad que dé 0 en ambas ocasiones es $(1-p)^2$. Por otro lado, la probabilidad que en ambas ocasiones sea diferente es de $p(1-p)$, sin importar el orden.
   Tomando esto en cuenta, el nuevo algoritmo deberá verificar si en ambas ejecuciones el resultado es diferente o igual. Si los bits son iguales en ambas ocasiones, se repite el proceso. Si los bits son diferentes, se devolverá 0 si el orden es 0 y 1; se devolverá 1 si el orden es 1 y 0.
   Ahora el tiempo de ejecución viene dado si los resultados son diferentes o no en ambas ocasiones. Para ello se suman las probabilidades de que sí se detenga el

algoritmo, es decir $p(1-p)+p(1-p)=2p(1-p)$. Si dividimos esta probabilidad dentro del tiempo de ejecución del algoritmo biased-random(), obtendremos el tiempo de ejecución de nuestro algoritmo, siendo este de $O(\frac{tiempo\ biased-random}{2p(1-p)})$.

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?**
El best-case scenario es cuando el primer candidato es el mejor de todos y solo lo debemos contratar a él. Si consideramos las permutaciones totales de igual manera y decidimos que solo una vez se ejecutará el best-case, se tiene que la probabilidad de contratar solo una vez es de
$$\frac{(n-1)!}{n!} = \frac{1}{n}$$
Ahora, en el worst-case, cuando se deben contratar a todos los candidatos porque el mejor está hasta de último, sí se deben considerar todas las permutaciones. Entonces la probabilidad de contratar n veces es de
$$\frac{(n-1)!}{n!} = \frac{1}{n!}$$

4. **Use variables aleatorias para calcular la suma esperada de *n* dados lanzados a la vez.**
$$Y = \Sigma_{i=1}^{n} d_i \qquad \text{(}d_i\text{ es el resultado del dado i)}$$
$$E[Y] = E[\Sigma_{i=1}^{n} d_i] = \Sigma_{i=1}^{n} E[d_i]$$
$$\Sigma_{i=1}^{n} E[d_i] = \Sigma_{i=1}^{n} \Sigma_{j=1}^{6} j * Prob$$
$$\Sigma_{i=1}^{n} E[d_i] = \Sigma_{i=1}^{n} \Sigma_{j=1}^{6} j * \frac{1}{6} = \Sigma_{i=1}^{n} (\frac{1}{6} + \frac{2}{6} + \frac{3}{6} + \frac{4}{6} + \frac{5}{6} + \frac{6}{6}) \qquad \text{dist. uniforme}$$
$$E[Y] = \Sigma_{i=1}^{n} E[d_i] = \Sigma_{i=1}^{n} \frac{21}{6}$$
$$E[Y] = \frac{21n}{6}$$

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*.**
Se deberán sumar los valores esperados de las variables aleatorias indicadoras correspondientes a las parejas (i, j). Entonces se tiene que:
$$E[Y] = \Sigma_{i=0}^{n-1} \Sigma_{j=i+1}^{n} E[Y_{i,j}] = \Sigma_{i=0}^{n-1} \Sigma_{j=i+1}^{n} P(A[i] > A[j])$$
Ahora calculamos el número de permutaciones $Perm(A[i] > A[j])$:
$$Perm(A[i] > A[j]) = \binom{n}{2} (n-2)!$$
$$Perm(A[i] > A[j]) = \left(\frac{n!}{2!(n-2)!}\right) (n-2)! = \frac{n!}{2}$$
Ya con esta permutación, calculamos la probabilidad $P(A[i] > A[j])$:
$$P(A[i] > A[j]) = \left(\frac{n!/2}{n!}\right) = \frac{1}{2}$$
Reemplazamos en la ecuación original y obtenemos el número esperado de inversiones en A:
$$E[Y] = \Sigma_{i=0}^{n-1} \Sigma_{j=i+1}^{n} \frac{1}{2} = \frac{1}{2} \Sigma_{i=0}^{n-1} \Sigma_{j=i+1}^{n} 1$$
$$E[Y] = \frac{1}{2} * \binom{n}{2} = \frac{n(n-1)}{4}$$

6. **Descargue el paper "Sorting the Slow Way: An Analysis of Perversely Awful Randomized Sorting Algorithms". Luego, responda las siguientes preguntas:**
    a. **Explique la fórmula usada para calcular $P[I_k]$ en la prueba del teorema 2.**
    Como $P[I_k]$ calcula la probabilidad de que al menos los primeros k elementos estén ordenados en un arreglo, debemos conocer las permutaciones de estos números que podrían estar ordenados al inicio.La combinaciones de k números enre 1 y n son $\binom{n}{k}$. Ahora, como no todos estarán ordenados, tendremos $(n-k)$ números que no estarán ordenados en el resto del arreglo. Juntando estos dos conceptos, obtenemos que las permutaciones son $\binom{n}{k} * (n-k)!$. Por último, para obtener la probabilidad que nos piden, dividimos dentro del total de posibles ordenamientos, es decir n!. El resultado es la fórmula planteada en el teorema 2.
    b. **En la prueba del teorema 2, ¿por qué $E[C] = \Sigma_{k>0}P[I_k]$?**
    La variable aleatoria indicadora I_k da 1 si los primeros k números están ordenados, pero se debe comparar cada uno de los primeros k números. El teorema toma a C como la variable aleatoria que indica el número de comparaciones que se realizan para saber si el arreglo está ordenado. Como se tienen que hacer al menos k comparaciones, se tiene que $I_k = 1 \Leftrightarrow C \geq k$. Sabiendo esto, la probabilidad $P[C \geq k]$ es igual es la suma acumulada de las probabilidades cuando C = k, k+1,k+2,... Entonces se tiene la expresión:
    $$E[C] = \Sigma_{i>0} i * P[C = i] = \Sigma_{i>0} \Sigma_{j \geq i} P[C = j] = \Sigma_{i>0} P[C \geq i] = \Sigma_{k>0} P[I_k]$$
    c. **En la sección 2.3, ¿por qué la variable aleatoria I que cuenta el número de iteraciones del algoritmo tiene distribución geométrica?**
    Tiene distribución geométrica porque cumple con su definición. I cuenta cuántas veces falló bogo-sort antes de indicar el resultado. Además, cada ejecución es independiente al anterior porque la permutación es completamente al azar.

7. **Descargue el paper "Fun-sort – or the chaos of unordered binary search". Luego, 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?**
    La diferencia es que guess-sort después de cada intercambio no verifica que el arreglo está ordenado, ahorrandonos n - 1 comparaciones.
    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?**
    El tiempo de ejecución del fun-sort en average-case es de $O(log_2 n)$. Si las condiciones del teorema 6 se cumplen, este tiempo de ejecución se dará por cada uno de los n elementos, teniendo así un tiempo de ejecución esperado de $O(n * log_2 n)$.