CC4102 - Examen

Prof. Gonzalo Navarro

5 de Enero de 2021

P1 (1.5 pt)

El Profesor Locovich está empeñado en determinar cuál es el último piso de un edificio desde el que puede tirarse un huevo de avestruz sin romperse. El edificio tiene n pisos, pero dado el alto costo de los huevos de avestruz, el Profesor sólo dispone de k huevos. Debe usted diseñar un algoritmo para resolver el problema. El costo de su algoritmo es la cantidad de veces que necesita tirar el huevo para responder.

1. (0.5pt) Demuestre que si $k = 1$, la complejidad del problema es n.
2. (0.5pt) Encuentre un algoritmo de costo $O(\sqrt{n})$ para $k = 2$.
3. (0.5pt) Demuestre que, con $k = 2$, la complejidad del problema es $\Omega(\sqrt{n})$.

Solución:

Para el punto 1, deben probarse los pisos $1, 2, \ldots, n$ o el adversario romperá el huevo luego del primer piso no probado. Para 2, se tira el primer huevo a intervalos de $\sqrt{n}$ y luego se refina con el segundo huevo. Para 3, se puede modelar el problema con el intervalo de pisos donde está la solución. El algoritmo tira el huevo en forma creciente (no tiene sentido hacerlo decrecientemente para el mismo huevo), y el adversario elegirá el mayor de los dos subintervalos. Luego queda un huevo, que es como en el punto 1. Así, si el algoritmo hace s tiradas y el adversario se queda con el mayor intervalo, el costo será al menos $s+n/(s+1)$, pues el mayor intervalo es de largo $n/(s+1)$. Esto se optimiza con $s = \sqrt{n}$ e intervalos todos iguales.

P2 (2.5 pt)

Un determinado algoritmo de compresión sobre un texto $T[1..n]$ funciona como sigue:

1. Recolecta todos los pares distintos $ab = T[i]T[i+1]$ que aparecen en el texto, con la cantidad de veces que aparece cada uno.
2. Selecciona el par más frecuente, digamos ab.
3. Reemplaza todas las ocurrencias de ab en T por un nuevo símbolo A (note que si $a = b$, el par aa podría reemplazarse sólo una vez cada dos en $aaaaaa$, pero puede ignorar esto).
4. Anota la regla $A \to ab$.
5. Actualiza los nuevos pares para considerar el reemplazo de cada ab por A, es decir, si donde decía $cabd$ ahora dice cAd, entonces ha desaparecido una ocurrencia de los pares ca y bd y ha aparecido una de cA y Ad.
6. Si queda en T algún par que se repite, vuelve al punto 2.
7. El output es la secuencia final más las reglas que se produjeron.

Se pide:

1. (0.5pt) Defina las estructuras de datos necesarias para poder realizar el paso 2, y los pasos 3 y 5 por cada ocurrencia de ab, en tiempo $O(\log n)$. Indique cuánto espacio ocupan sus estructuras y si su tiempo es de peor caso o esperado.

2. (0.5pt) Dado el punto anterior, demuestre que el algoritmo completo funciona en tiempo $O(n \log n)$.

3. (0.5pt) Demuestre que los pares cuya frecuencia se modifica, y los que se generan, nunca superan la frecuencia del que se está reemplazando.

4. (0.5pt) Usando el hecho anterior, modifique sus estructuras de datos para reducir la complejidad total a $O(n)$.

5. (0.5pt) Entendiendo que el alfabeto original de T es $\{1, 2, \ldots, \sigma\}$ y que usted puede numerar los símbolos nuevos A como desee, diseñe y analice un algoritmo de peor caso $O(n)$ para reconstruir T a partir de la secuencia final y las reglas.

Solución:

Es RePair. Para el punto 1 podemos tener una lista doblemente enlazada con las posiciones de T, y otra con los pares distintos (cada ab apunta al previo y al siguiente ab). Se puede guardar un hash (espacio $O(n)$, tiempo esperado) con los pares, o una tabla (espacio $O(n^2)$, tiempo de peor caso), y además tenerlos en un max-heap por frecuencia, donde el heap también apunte a una ocurrencia del par en T. Así, dado el par a reemplazar, encontramos todas sus ocurrencias en la lista de pares, y tenemos los links para modificar T, modificar las listas de pares ca, bd, cA y Ad en $O(1)$, más cambiar sus frecuencias en el heap en tiempo $O(\log n)$. Para el punto 2, es un análisis amortizado, mostrando que el texto decrece en 1 letra cada vez que se hace un reemplazo que cuesta $O(\log n)$. También deben mostrar cómo hacer el paso 1, que no es difícil. Para el punto 3, basta ver que los otros pares existentes ya tenían frecuencias no mayores, y decrecen, y que los pares nuevos no pueden ser más que los que se están reemplazando. Para el punto 4, se puede reemplazar el heap por un arreglo que para cada frecuencia en $[1..n]$ tenga la lista de pares con esa frecuencia, y ver que todas las operaciones se hacen en tiempo constante. Encontrar el máximo, dada la propiedad anterior, se reduce a recorrer el arreglo a lo largo de todo el algoritmo, $O(n)$ en total amortizadamente. Para el punto 5, se asignan números $\sigma+1, \sigma+2, \ldots$ a los nuevos símbolos y se guardan los pares en un arreglo. Se recorre la secuencia, y cada símbolo $> \sigma$ se reemplaza recursivamente por su par. Amortizadamente se ve que generar r símbolos cuesta $O(r)$, por ejemplo son las hojas del árbol binario de las invocaciones, donde éstas son los nodos internos.

P3 (1.5 pt)

Sea S un conjunto de n elementos. Sean $S_1, \ldots, S_k$ subconjuntos de S, cada uno de ellos de tamaño exactamente r. Suponga que se cumple $k \cdot 2^{-r} \leq 1/4$ (es decir no pueden ser demasiados conjuntos muy chicos). Se desea colorear los elementos de S rojo o azul, de modo que todo S_i contenga elementos de ambos colores.

1. (0.5pt) Proponga un algoritmo tipo Monte Carlo que encuentre un coloreo válido con probabilidad al menos $1/2$, y analice su costo de peor caso.

2. (0.5pt) Reduzca la probabilidad de error del algoritmo anterior a $1/2^t$ para cualquier $t > 0$ dado, y analice su costo de peor caso.

3. (0.5pt) Proponga un algoritmo tipo Las Vegas y analice su costo promedio.

Solución:

Para 1, uso un coloreo al azar (costo $O(n)$, o $O(rk)$, lo que sea menor, cualquiera de las dos cotas vale). La prob de que cada S_i sea todo rojo es 2^{-r}, o todo azul es 2^{-r}. La prob de que alguno sea todo rojo o todo azul es $\leq k \cdot 2 \cdot 2^{-r} \leq 1/2$. Para 2, lo repetimos t veces, comprobando luego de cada vez, luego nos rendimos. Ojo que ahora tenemos que verificar el coloreo, lo que cuesta $O(rk)$, totalizando $O(trk)$. Para 3, repetir lo anterior hasta dar con un coloreo válido. En promedio necesito 2 corridas para acertar, así que el costo es $O(rk)$.

P4 (2.0 pt)

En clase se vio una 2-aproximación sencilla para el recubrimiento de vértices sin pesos. Considere ahora la siguiente variante aleatorizada del método: cada vez que considera una arista (u, v), elige u con probabilidad $1/2$ y elige v con probabilidad $1/2$, independientemente. En cada caso, si elige un nodo, elimina todas las otras aristas que inciden en él.

Sea S_i el conjunto de nodos elegidos hasta el paso i y sea OPT un conjunto óptimo.

1. (0.5pt) Demuestre que, en todo momento, por cada arista que no ha sido aún eliminada, el algoritmo deberá elegir al menos uno de sus extremos.

2. (1pt) Demuestre que lo anterior implica que para todo i vale $E(|S_i \cap OPT|) \geq E(|S_i \setminus OPT|)$.

3. (0.5pt) Demuestre que lo anterior implica que el tamaño esperado del conjunto de nodos que entrega el algoritmo es una 2-aproximación.

Solución:

Notar que el algoritmo podría no insertar ni u ni v, pero en ese caso tampoco eliminaría la arista que consideró, por lo que esas iteraciones se pueden ignorar. Para el punto 1, la arista que no ha sido eliminada no tiene ningún extremo en S_i, por lo cual el algoritmo deberá cubrirla más tarde. Más aún, Para el punto 2, por el punto anterior, la esperanza de $|S_i|$ se incrementa en $1/2$ por u y $1/2$ por v, por lo que $|S_i \cap OPT|$ se incrementa al menos en $1/2$, y $|S_i \setminus OPT|$ se incrementa a lo sumo en $1/2$ (si uno solo de u, v está en OPT). Para el 3, resulta que $E(|ALG \cap OPT|) \geq E(|ALG \setminus OPT|)$, por lo que en promedio más de la mitad de los elementos elegidos están en el óptimo.