Tomás Maturana
19.186.973-7

Examen

P1:

1) Para k=1, bastaría con lanzar el huevo en el piso 1, luego si sobrevive, lanzarlo desde el 2… hasta llegar al piso en el que se rompa o hasta el piso n, en caso de que no se rompa. Dado que existen n pisos, se podría llegar a lanzar el huevo n veces, subiendo al piso x, lanzando el huevo y bajando a buscar el huevo (o bajando a limpiar el desastre). Además, dada la información del problema, no se puede estimar que el huevo resista al menos hasta cierto piso, por lo que el problema es de complejidad n.

2) Teniendo dos huevos, se puede subir el edificio con ambos huevos. Sea $P=n^{1/2}$, luego, en el primer paso, se sube hasta el piso 1 (considerando el piso 0 como en Argentina), se lanza un huevo, luego se sigue subiendo hasta el piso P y se lanza el segundo huevo. Si ni un huevo se rompió, entonces se baja a buscar los huevos, se sube hasta el piso P+1 y se lanza un huevo, luego se sube hasta el piso 2*P y se lanza el segundo. Va a llegar un punto en el que el primer o el segundo huevo lanzado se rompa. Luego, si el primer huevo se rompe, tenemos la respuesta; si es el segundo huevo el que se rompe, quedará recorrer ese intervalo de pisos (de largo $n^{1/2}$-2, ya que ya se eliminaron dos casos posibles con los huevos lanzados), hasta que se rompa el huevo. Este algoritmo en el peor caso toma $n^{1/2}$+ $n^{1/2}$-2 subidas y bajadas al edificio, por lo tanto es $O(n^{1/2})$.

3) Para el mismo caso de dos huevos, la cota inferior del problema es $\Omega(n^{1/2})$, ya que si quisiéramos dividir los pisos en intervalos más grandes que de $n^{1/2}$, por ejemplo a intervalos de $n^{1/2}$+1, entonces el costo seguiría siendo de $\Omega(n^{1/2})$. Si seguimos haciéndolos más grandes, por ejemplo a $2*n^{1/2}$, tampoco mejoraría el costo, ya que una vez que se encuentre el intervalo clave para buscar, tendremos que hacer una búsqueda de costo $\Omega(n^{1/2})$ igualmente. Es decir, no sirve aumentar el tamaño del intervalo para reducir el costo del algoritmo.
Si quisiéramos reducir los pasos tenemos un problema parecido. De hecho, si el intervalo se hace lo suficientemente grande, podríamos romper de inmediato el segundo huevo, haciendo del problema $\Omega(n)$. De igual manera si lo hacemos suficientemente pequeño, se podría tener costos $\Omega(n)$

P2:

1) Se necesitaría un arreglo para guardar todos los pares y su número de ocurrencias (esto sería de la parte 1), con esto se puede ordenar el arreglo según su número de ocurrencias, para poder obtener el par más frecuente. Además, un arreglo para guardar los índices de las ocurrencias encontradas más la misma concurrencia. Estos arreglos ocupan O(n).
   Luego, para los pasos 3 y 5, se tiene una lista enlazada a la que se le agrega la concurrencia elegida y su regla.
   El tiempo para cada reemplazo será de O(log n), pero más específicamente, se podrá hacer en tiempo constante, ya que tenemos las posiciones donde hay que reemplazar.

2) Ya que una reemplazo tiene costo O(log n) y el texto original tiene n elementos, supongamos que todas las concurrencias que aparecen en el texto ocurren máximo 2 veces, y que luego de cierto punto ya no quedan caracteres iniciales (se han reemplazado todos) y que además vuelven a haber concurrencias que sólo aparecen 2 veces como máximo. En el peor caso, esto ocurre múltiples veces hasta tener un texto de largo 2. Un ejemplo de esto sería en el caso abcdabcd -> 1cd1cd -> 2d2d -> 33. Más específicamente, el peor caso demora tiene una cota O(1/2n*log(n)), por lo que podemos decir que el algoritmo es de tiempo O(n log n).

3) Ya que se parte reemplazando por los pares con mayor frecuencia, luego de reemplazar dichos pares, supongamos x pares, aparece un nuevo elemento Y, x veces en el texto, el cual puede formar a lo más x pares nuevos. Por contradicción, si luego de los reemplazos se generan x+1 pares Yz, implicaría que antes del reemplazo existía previamente un elemento Y en el texto formando el par Yz. Independientemente de cuál sea ese z, no podía existir un elemento Y en el texto antes de los reemplazos.

4) Las estructuras que tenía previamente definidas, ya cumplen con el tiempo O(n), ya que luego de elegir qué concurrencia reemplazar, basta con verificar las posiciones de las concurrencias y reemplazarlas en el texto en tiempo constante. Esto se hará como máximo n veces. Luego, tenemos que el algoritmo es O(n).

5) Se tendría que mapear la posición de los elementos presentes en el texto comprimido y luego reemplazar los elementos según las reglas, agregando al mapeo los nuevos pares creados. Como máximo habrán n reglas (de hecho menos), por lo que se harán n iteraciones de reemplazos. En efecto, el algoritmo queda de costo O(n).

P3:

1) Se puede iterar el conjunto S, lanzando una moneda para decidir si pintar de rojo o de azul cada uno de sus elementos, de esta manera, aproximadamente la mitad de los elementos de S estarán pintados de rojo y la otra azul. Aún así, podría pasar que los subconjuntos S_i tengan todos sus elementos pintados de el mismo color, pero esto sucederá con probabilidad ½, independientemente de cómo se elijan los subconjuntos S_i (obviamente asumiendo que no se elijen sabiendo de antemano el color de cada elemento). Ya que es un algoritmo Montecarlo, la respuesta puede estar errónea (con probabilidad ½ como vimos), pero su costo se ve reducido al mínimo, garantizando la probabilidad de ½. Este algoritmo tiene costo O(n).

2) Se desea reducir la probabilidad de error,

3) Luego de usar el algoritmo anterior para pintar los elementos de S correctamente con probabilidad $1-1/(2^t)$, podemos recorrer los elementos de cada S_i, viendo si tienen uno rojo y uno azul. Al final quedarán pocos S_k con todos sus elementos de un color, entonces habría que ver si alguno de sus elementos se puede cambiar de color sin dejar otro S_i con el mismo problema. Para eso, es mejor volver a correr el algoritmo y tener suerte la próxima vez. Su costo promedio será O(n), ya que será el mismo costo del algoritmo previo, pero lanzado 1/p veces, con p la probabilidad de estar correcto.

P4:

1) El algoritmo óptimo entregará un subconjunto de nodos mínimo, con tal de que toda arista del grafo incida en alguno de los nodos del subconjunto. Es por esto, que para la variante aleatorizada mencionada en el enunciado, en todo momento que queden aristas que aún no han sido "trabajadas", es decir, que aún no han sido consideradas para elegir uno de sus nodos; cada una de esas aristas tendrá dos nodos, y al menos uno de ellos habrá sido elegido por el algoritmo óptimo, ya que en caso contrario, el algoritmo óptimo estaría entregando un subconjunto de nodos, pero que en ni uno de sus nodos incide la arista previamente comentada, por lo tanto no sería un *vortex cover* válido.

2)

- Por inducción tenemos:
  Sea X1= |S_i inter OPT|, e Y1= |S_i \ OPT|. Digamos que en el paso 1, se consideró la arista e1=(u_1,v_1).
  Supongamos además que u_1 y v_1 están en OPT.
  Podemos notar que X1 puede tomar valores 0, 1, 2, con P(X1=0)= ¼; P(X1=1)= ½; P(X1=2)= ¼. Luego E(X1)=1
  Y1 puede tomar sólo el valor 0, pues u_1 y v_1 están en OPT. Luego E(y1)=0.
  En este caso, se tiene que E(|S_i inter OPT|)>= E(|S_i \ OPT|).
  Supongamos que sólo uno de los nodos u_1 y v_1 está en OPT. Particularmente, supongamos que u_1 está en OPT y v_1 no lo está.
  X1 toma valores entre 0 y 1, luego P(X1=0)= ¼ + ¼= ½; P(x1=1)= ½.
  Luego, E(X1)=1/2.
  Y1 toma valores 0 o 1.
  Entonces, P(Y1=0)=P("en el paso 1 se eligió sólo u1 o no se eligió ni uno")= ½; y P(Y1=1)= ½; entonces E(Y1)=1/2, por lo que la propiedad se mantiene.
- Paso inductivo:
  Supongamos que en un paso i, tenemos que E(|S_i inter OPT|)>= E(|S_i \ OPT|).
  Sea e_i+1=(u_i+1, v_i+1), la arista considerada en el paso i+1. Sea X_i+1=|(S_i+1 \ S_i) inter OPT|), es decir, el número de nodos que elige el paso i+1 que están en OPT. Y sea Y_i+1=|(S_i+1 \ S_i) \ OPT|) el número de nodos considerados en el paso i+1 que no están en OPT.
  Supongamos u_i+1, v_i+1 está en OPT
  Luego X_i+1 toma valores 0, 1 o 2 (siguiendo el mismo razonamiento del caso base), entonces P(X_i+1=0)=P("no elegir ni u_i ni v_i")= ½*1/2= ¼; P(X_i+1=1)=P("elegir sólo u_i o v_i")= ¼+ ¼=1/2; y P(X_i+1=2) = ¼.
  Luego E(Xi+1)=1.
  Yi+1 toma sólo el valor 0, entonces su esperanza es cero. Luego tenemos que E(X_i+1) >= E(y_i+1).
- Por otro lado, si sólo uno de los u_i+1 o v_i+1 está en OPT, digamos que u_i+1 sí está en OPT:
  X_i+1 toma valores 0 o 1, entonces P(X_i+1=0)=1/2; P(X_i+1=1)=1/2.
  Además, Y_i+1 toma valores 0 o 1, entonces P(Y_i+1=0)=1/2; P(Y_i+1=1)=1/2, entonces si esperanza es de ½.

- Finalmente, tenemos que en todo caso se cumple la desigualdad E(X_i+1)>= E(Y_i+1)
- Notar que:
  E(|S_i+1 inter OPT|)=
  E(|S_i inter OPT| + X_i+1)=
  E(|S_i inter OPT| + E(X_i+1))
  ****H.I.******
  >= E(|S_i \ OPT|) + E(Y_i+1))
  = E(|S_i \ OPT| + Y_i+1)
  >= E(|S_i+1 \ OPT|)

3) Se puede ver que el conjunto resultante del algoritmo es un recubrimiento de vértices, ya que toda arista que se consideró, estaba cubierta por algún nodo del subconjunto final. Esto queda claro ya que en caso de no elegir ni un nodo de la arista considerada, entonces la arista no es eliminada de las aristas a considerar en un futuro; por lo tanto podría volver a ser considerada o ser eliminada luego de elegir el nodo en común que tenga con una arista contigua. Además, por cada par de nodos que elige el algoritmo, el algoritmo óptimo tuvo que haber elegido al menos uno, y ya que se eliminan las aristas que inciden en los nodos elegidos, todos los nodos elegidos comparten a lo más una arista, por lo tanto el algoritmo óptimo debe meter al menos uno de esos dos nodos, pero no menos.
   Ya que la desigualdad vista en la parte anterior se cumple para todo paso i, en particular se cumple para el último paso. En el último paso, el algoritmo habrá elegido más nodos que el algoritmo óptimo (y en el mejor caso, los mismos). Ésta cantidad extra de nodos, no puede ser menor que la cantidad de nodos elegidos por el algoritmo y que no fueron elegidos por el algoritmo óptimo, ya que como vimos, el algoritmo óptimo tuvo que elegir al menos uno de dos nodos que comparten una arista. Por contradicción, esto pasaría si el algoritmo elige nodos, tal que existe un trío de nodos que comparten sus aristas, lo que es imposible, ya que como máximo el algoritmo permite elegir dos de ellos y luego elimina las aristas incidentes. Se concluye que el algoritmo es una 2-aproximación.