Algoritmos CC40A- 2005
Cotas Inferiores

Profesor: Gonzalo Navarro

Auxiliar: Mauricio Cerda

1 Cotas Inferiores

1.1 ¿Qué es una cota inferior?

Se ha trabajado hasta ahora construyendo diferentes clases de algoritmos, y luego obteniendo su complejidad tanto temporal como espacial. Si escribimos $T_A(X)$ como el tiempo de computo para el algoritmo A con la entrada X. El peor caso de dicho algoritmo para cualquier entrada se define como:

$$T_A(n) = max_{|X|=n}(T_A(X)) \tag{1}$$

Ahora bien, la complejidad del peor caso del PROBLEMA Π, es el peor caso del algoritmo más rápido para resolverlo.

$$T_\Pi(n) = min_{A resuelve \Pi}(max_{|X|=n}(T_A(X))) \tag{2}$$

Ahora suponiendo se tiene un algoritmo A de $O(f(n))$, entonces, en forma inmediata se tiene la cota superior del problem $O(f(n))$. Para la cota inferior, en cambio, se debe mostrar que $T_\Pi(n) = \Omega(f(n))$, es decir que TODO algoritmo tiene al menos ese peor caso, ya no basta con un algoritmo en particular para encontrar la solución.

1.2 Técnicas para encontrar cotas

El mayor problema es que no hay una manera formal de señalar todos los algoritmos, por lo que existen técnicas para ayudar en la construcción de estas cotas.

1.2.1 Teoría de la información

Un modelo de computo muy usado son los árboles de decisión, los cuales corresponden a una pregunta por nodo, y a cada hoja del árbol a una posible respuesta para una cierta entrada, con esto, el tiempo de solución de un problema es el número de preguntas con que se puede llegar a una hoja del árbol, es decir, la altura del árbol. Una observación importante, es que un algoritmo correcto debe al menos entregar después de todas las preguntas en el árbol de decisión, todas las respuestas posibles.

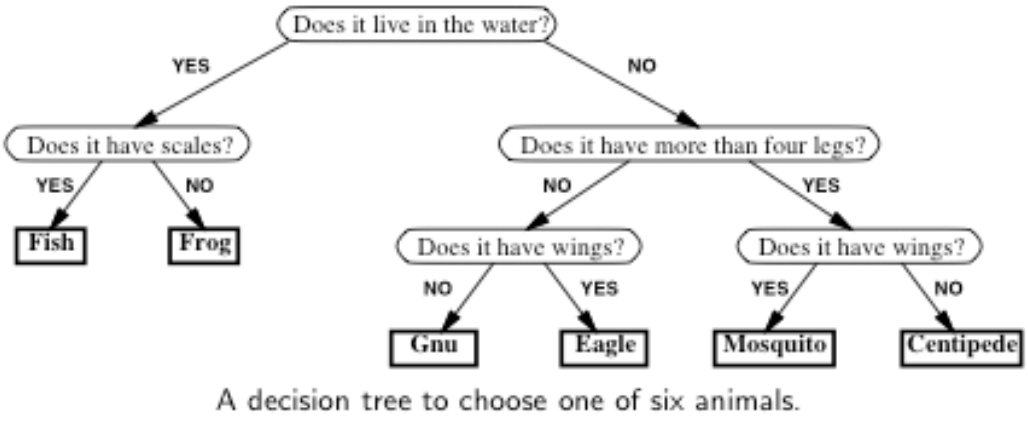

Figure 1:

Es decir, conociendo la cantidad de posibles soluciones diferentes para un problema (N), ya podemos encontrar una cota inferior (altura del árbol), notar que es en base k.

$$T_{\Pi}(n) = \Omega(log_k N) \tag{3}$$

Ejemplo 1, Ordenamiento Busquemos la cantidad de posibles soluciones, suponiendo un arreglo de N objetos, para el primer elemento hay N alternativas, para el segundo $N-1$, tercero $N-2$, etc. Es decir, hay $N!$ posibles resultados, utilizando la ecuación anterior tenemos que el problema Π de ordenar es

$$T_{\Pi}(n) = \Omega(log_k N!) \tag{4}$$

Utilizando la aproximación de Stirling (Reif, termodinámica) se tiene que $N! > (\frac{N}{e})^N$, con lo cual

$$T_{\Pi}(n) = \Omega(N log_k N) \tag{5}$$

La cota anterior es la que alcanzan algoritmos como quicksort, mergesort, heapsort, etc.

Ejemplo 2, Búsqueda del máximo Otro problema donde ya no es tan directo encontrar la cota inferior es el máximo en un conjunto de N elementos desordenados. Hay N posibles resultados, con lo cual la cota sería $\Omega(log_k N)$ pero, esta no es la mejor cota podemos encontrar, pues se sabe que si hacemos menos de N-1 comparaciones, hay posibilidades de fallar, esto conduce a una nueva técnica, descrita a continuación.

1.2.2 Argumento del Adversario

La idea del adversario se refiere a suponer un contrincante malicioso que pretende elegir una entrada, pero cada vez que el algoritmo hace una pregunta, el adversario responde de tal manera que hace hacer el máximo de trabajo al adversario. Si el algoritmo no pregunta lo suficiente, el adversario podrá contruir varias respuestas para la misma entrada, haciendo fallar al algoritmo. Es importante notar que para que esta técnica sea válida, el adversario no puede hacer supuestos hacerca del orden de las preguntas. Una manera útil de verlo es construir un algoritmo PARA el adversario (notar que no es lo mismo que construir uno para resolver el problema).

Ejemplo 1, El juego de las Cartas En el centro de la ciudad, aún se pueden encontrar personajes que dan la posibilidad de apostar en un juego de tres cartas, donde una está premiada (no lo intente). Supongamos queremos jugar algo diferente, conocer si existe alguna carta premiada. El adversario, en un primer momento da vuelta todas las cartas, y nosotros podemos preguntar, supongamos que elegimos una carta, el adversario nos responde diciendo que no era la carta, y estamo forzados a seguir preguntando, pues si hiciera una sóla pregunta, (o dos), el nos diría que cualquiera de las otras era la premiada, y que nos equivocamos.

Ejemplo 2, Propiedades evasivas de grafos Un patrón se puede llamar evasivo si para buscarlo en un string se debe recorrer todo el string (como se demostro en clase auxiliar, para el patrón 01 en un strin binario). En un grafo existen propiedades evasivas, las cuales obligan a cubrir las $\frac{n!}{2!(n-2)!}$ combinaciones de arcos. Una de estas propiedades es si un grafo es o no vacío. En este caso el algoritmo del adversario sería ir respondiendo que cada arco, que esta vacío, pero dejando el resto como incognita, esto obliga a preguntar por cada uno de los arcos. Propuesto, demostrar (construir algoritmo para el adversario) la evasividad de mostrar si un grafo tiene o no ciclos.

Veamos ahora un problema un poco más delicado en su análisis y que utiliza ambas técnicas para encontrar la cota inferior, el problema de la mediana.

Dada una secuencia x_1, ..., x_n de n elementos diferentes, encuentre el índice m, tal que x_m es la mediana del conjunto.

Primero busquemos una forma más clara de escribir el problema. De encontrar la mediana, debemos de forma obligatoria, identificar a los elementos mayores y menores a ella. Esto se demuestra utilizando un argumento de adversario, pues de no asociar a ún elemento (x_i) con alguno de esos grupos, siempre se puede encontrar una entrada en que x_i y x_m son adyacentes, y aún sabiendo a cual grupo pertenecen todos los otros elementos, el adversario puede variar la entrada de forma que la mediana pudiera ser x_i o x_m, haciendo nuestro resultado siempre erróneo. Con lo anterior podemos reescribir el problema como

Dada una secuencia x_1, ..., x_n de n elementos diferentes, encuentre los elementos tal que uno es mayor a $\frac{n}{2}-1$ elementos y menor al resto, y el otro, que sea mayor a $\frac{n}{2}$ y menor a $\frac{n}{2}-1$

Para facilitar el análisis, consideremos n un número par. Supongamos que conocemos el grupo de elementos R que son mayores que la mediana, el algoritmo deberá identificar ahora el mayor elemento de un grupo de $\frac{n}{2}$ elementos, como ya se ha visto en clase auxiliar, otro argumento de adversario, muestra que se deben hacer al menos $\frac{n}{2}-1$ comparaciones, es decir, esa es la altura de nuestro arbol de decisión. es decir que dicho árbol tiene al menos $2^{\frac{n}{2}-1}$ hojas. Ahora nos queda saber cuantos de estos posible grupos R es posible encontrar, esto son $\binom{n}{\frac{n}{2}-1}$ alternativas, con lo cual la cantidad de hojas de nuestro árbol en total será de:

$$\binom{n}{\frac{n}{2}-1} 2^{\frac{n}{2}-1} \tag{6}$$

Como ya se ha visto, puedo saber a partir de esta información cuantas comparaciones son las mínimas necesarias.

$$\lceil \frac{n}{2} - 1 + \log \binom{n}{\frac{n}{2}-1} \rceil = \frac{3n}{2} - O(log(n)) \tag{7}$$

La pregunta es si se puede hacer mejor, en textos hay demostraciones para cotas $2n - o(n)$ con argumentos similares, se eligió esta manera por ser mas clara.