CI2612: Algoritmos y Estructuras de Datos II

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Cotas inferiores para ordenamiento basado en comparaciones

Objetivos

- Presentar cotas para el mínimo número de comparaciones necesarias para ordenamiento basado en comparaciones en el peor caso
- Introducir el concepto de árbol de decisión para el análisis
- Derivar cotas inferiores para el número de comparaciones esperadas en el caso de algoritmos randomizados
- Derivar cotas inferiores para el número de comparaciones necesarias para mezclar dos listas con n elementos cada una

Ordenamiento basado en comparaciones

Un algoritmo de **ordenamiento basado en comparaciones** sólo obtiene **información** de los elementos al comparalos entre si, utilizando las relaciones: $\leq$, $\geq$, $<$, $>$, y $=$

Un algoritmo de ordenamiento basado en comparaciones **nunca inspecciona** de forma directa un elemento de la secuencia

Ordenamiento por inserción, mergesort, heapsort y quicksort son algoritmos de ordenamiento basado en comparaciones

Mostraremos que todo algoritmo de ordenamiento basado en comparaciones **necesita hacer** $\Omega(n \log n)$ **comparaciones**

Ordenamiento basado en comparaciones

Sin perder generalidad, suponemos que los elementos $\langle a_1, a_2, \ldots, a_n \rangle$ a ordenar son todos **distintos**

Por lo tanto, podemos suponer que el algoritmo sólo realiza comparaciones de tipo '$\leq$'

Nos **enfocamos sólo en las comparaciones** que hace el algoritmo y no en sus estructuras de control, movimiento de elementos, etc

Las comparaciones que efectua el algoritmo sobre **todas las posibles entradas** de tamaño n (fijo) se representan de forma **abstracta** en un **árbol de decisión**

Árboles de decisión

Un árbol de decisión es un **árbol binario completo** (i.e. cada nodo interno tiene dos hijos) que representa las **comparaciones realizadas** en **todas las ejecuciones posibles** sobre entradas de tamaño n

Cada nodo interno se anota de forma '$i : j$' con $1 \leq i < j \leq n$ y cada hoja con una **permutación** $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ sobre $\{1, 2, \ldots, n\}$

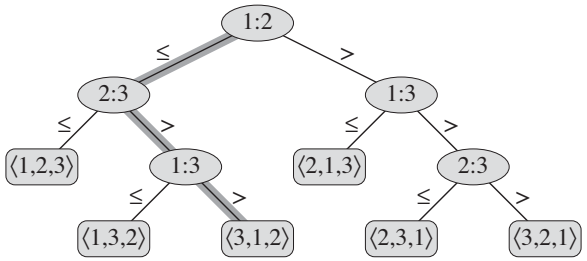

Imagen de Cormen et al. Intro. to Algorithms. MIT Press

Ejecuciones sobre árboles de decisión

La ejecución del algoritmo de ordenamiento sobre una entrada dada corresponde a un **camino en el árbol desde la raíz hasta una hoja**

Dicho camino denota las comparaciones realizadas y su orden de realización para la entrada

Un nodo $i : j$ en el camino denota la comparación $a_i \leq a_j$. Si el camino continua sobre el **hijo izquierdo**, la comparación es **cierta**, y si continua sobre el hijo **derecho** la comparación es **falsa**

Si el camino alcanza la hoja $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$, la permutación de entrada **ordenada** es $\langle a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)} \rangle$

Ejecuciones sobre árboles de decisión

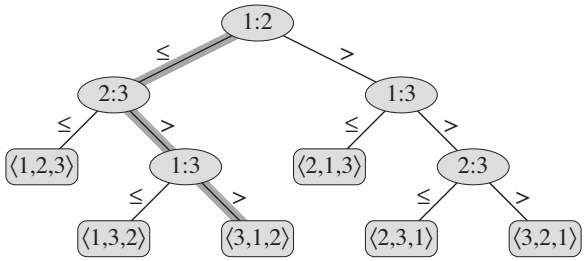

Imagen de Cormen et al. Intro. to Algorithms. MIT Press

La ejecución mostrada corresponde a la entrada $\langle a_1, a_2, a_3 \rangle = \langle 6, 8, 5 \rangle$

- Lo primero que se hace es la comparación $a_1 \leq a_2$ que es cierta
- Luego se compara $a_2 \leq a_3$ que resulta falsa
- Finalmente, se compara $a_1 \leq a_3$ que también resulta falsa
- La entrada ordenada es $\langle a_3, a_1, a_2 \rangle = \langle 5, 6, 8 \rangle$

Tamaño de los árboles de decisión

Un algoritmo correcto de ordenamiento tiene que poder ordenar de forma correcta cualquier entrada

Por lo tanto, su árbol de decisión para entradas de tamaño n debe tener al menos $n!$ **hojas alcanzables** (visitadas por ejecuciones válidas) que corresponden a las $n!$ permutaciones sobre $\{1, 2, \ldots, n\}$

El árbol puede contener más de $n!$ hojas. En dicho caso, existen caminos que no se corresponden a ejecuciones válidas

Cota inferior

Todo algoritmo de ordenamiento basado en comparaciones **requiere** hacer $\Omega(n \log n)$ comparaciones en el **peor caso**

Prueba:

Para n elementos, el algoritmo se corresponde con un árbol de decisión de altura h y ℓ **hojas alcanzables**

Como el algoritmo ordena las $n!$ distintas permutaciones, el árbol de decisión contiene al menos una hoja por cada una de las $n!$ permutaciones. Tenemos

$$n! \leq \ell \leq 2^h$$

ya que un **árbol binario de altura h tiene a lo sumo 2^h hojas**

Tomando logaritmos, $h \geq \log(n!) = \Omega(n \log n)$

Es decir, existe un hoja alcanzable, asociada a una entrada de tamaño n, para la cual el algoritmo realiza $\Omega(n \log n)$ comparaciones □

Optimalidad asintótica de mergesort y heapsort

Como **mergesort** y **heapsort** toman tiempo $O(n \log n)$ en el peor caso, y ambos son algoritmos correctos y basados en comparaciones, entonces ambos algoritmos son **asintóticamente óptimos**

Cota para tiempo promedio/esperado

Ahora mostramos una cota de $\Omega(n \log n)$ comparaciones para ordenar n elementos en promedio

La cota aplica para algoritmos determinísticos y randomizados (comparaciones esperadas)

En el caso de algoritmos randomizados, observe que estamos hablando de dos tipos de promedios:

- promedio del número de comparaciones sobre las $n!$ posibles permutaciones en la entrada
- esperanza sobre todos los eventos aleatorios realizados por el algoritmo (coin flips o llamadas a `Random(p,r)`)

(Problema 8-1)

Cota para tiempo promedio

Comenzamos analizando un algoritmo determinístico cualquiera A con árbol de decisión T_A

Para cada posible entrada, existe un camino en T_A desde la raíz hasta una hoja. Etiquetemos cada hoja con la probabilidad de la entrada que la "alcanza", en nuestro caso $1/n!$ ya que asumimos que todas las $n!$ posibles entradas son **equiprobables**

Las otras hojas en T_A no son alcanzables por ninguna entrada y las etiquetamos con probabilidad 0

El número promedio de comparaciones hechas por A es

$$\mathbb{E}[T_A] \;=\; \sum_{n\in \text{hojas}(T_A)} depth(n) \times \mathbb{P}(n)$$

Número promedio de comparaciones

Para un árbol de decisión T, defina

$$D(T) \;=\; \sum_{n\in \text{hojas}(T)} depth(n)$$

Para un algoritmo A, defina T_A^* como T_A en donde las hojas inalcanzables se han eliminado, y todo nodo interno tiene exactamente 2 hijos

$$\mathbb{E}[T_A] \;=\; \sum_{n\in \text{hojas}(T_A)} depth(n) \times \mathbb{P}(n) \;\geq\; D(T_A^*)/n!$$

Calcularemos una cota inferior sobre $D(T_A^*)/n!$

Cota para tiempo promedio

Si T^* tiene $k > 1$ hojas, entonces $D(T^*) = D(T_L^*) + D(T_R^*) + k$ donde T_L^* y T_R^* son los subárboles izquierdo y derecho asociados a T^*

Defina $d(k) = \min_{T^*} D(T^*)$ donde el mínimo es sobre todos los árboles T^* con $k > 1$ hojas, y $d(1) = 0$

Mostraremos la recurrencia:

$$d(k) \;=\; \begin{cases} 0 & \text{si } k = 1 \\ \min_{1\leq i<k} d(i) + d(k-i) + k & \text{si } k > 1 \end{cases}$$

en donde la igualdad la estableceremos mostrando dos desigualdades

Cota para tiempo promedio

Si T^* tiene $k > 1$ hojas, entonces $D(T^*) = D(T_L^*) + D(T_R^*) + k$ donde T_L^* y T_R^* son los subárboles izquierdo y derecho asociados a T^*

Defina $d(k) = \min_{T^*} D(T^*)$ donde el mínimo es sobre todos los árboles T^* con $k > 1$ hojas, y $d(1) = 0$

- Sea $1 \leq i < k$ y T_L^* y T_R^* dos árboles con i y $k-i$ hojas tales que $d(i) = D(T_L^*)$ y $d(k-i) = D(T_R^*)$. Ambos árboles los podemos unir en un árbol T^*. Entonces,

$$d(k) \;\leq\; D(T) \;=\; d(i) + d(k-i) + k$$

Entonces $d(k) \leq \min_{1\leq i<k} d(i) + d(k-i) + k$

Cota para tiempo promedio

Si T^* tiene $k > 1$ hojas, entonces $D(T^*) = D(T_L^*) + D(T_R^*) + k$ donde T_L^* y T_R^* son los subárboles izquierdo y derecho asociados a T^*

Defina $d(k) = \min_{T^*} D(T^*)$ donde el mínimo es sobre todos los árboles T^* con $k > 1$ hojas, y $d(1) = 0$

- Sea T^* un árbol con $k > 1$ hojas tal que $d(k) = D(T^*)$, e i y $k - i$ el número de hojas en T_L^* y T_R^*:

$$d(k) \;=\; D(T^*) \;=\; D(T_L^*) + D(T_R^*) + k \;\geq\; d(i) + d(k-i) + k$$

Entonces $d(k) \geq \min_{1 \leq i < k} d(i) + d(k-i) + k$

Cota para tiempo promedio

Si T^* tiene $k > 1$ hojas, entonces $D(T^*) = D(T_L^*) + D(T_R^*) + k$ donde T_L^* y T_R^* son los subárboles izquierdo y derecho asociados a T^*

Defina $d(k) = \min_{T^*} D(T^*)$ donde el mínimo es sobre todos los árboles T^* con $k > 1$ hojas, y $d(1) = 0$

Entonces, $d(k) = \min_{1 \leq i < k} d(i) + d(k-i) + k$

Ahora mostraremos $d(k) \geq k \log_2 k$ para $k \geq 1$ por inducción

Cota para tiempo promedio

Caso base: $d(1) = 0 = 1 \log_2 1$

Tesis:

$$\begin{aligned}
d(k) &= \min_{1 \leq i < k} d(i) + d(k-i) + k \\
&= k + \min_{1 \leq i < k} d(i) + d(k-i) \\
&\geq k + \min_{1 \leq i < k} i \log_2 i + (k-i) \log_2 (k-i) \\
&\geq k + \min_{x \in [1,k)} x \log_2 x + (k-x) \log_2 (k-x) \\
&= k + \tfrac{k}{2} \log_2 \tfrac{k}{2} + \tfrac{k}{2} \log_2 \tfrac{k}{2} \\
&= k \log_2 k
\end{aligned}$$

[usando $\operatorname{argmin}_{x \in [1,k)} f(x) = \frac{k}{2}$ para $f(x) = x \log_2 x + (k-x) \log_2 (k-x)$]

Cota para tiempo promedio

Tenemos $d(k) = \Omega(k \log_2 k) = \Omega(k \log k)$

Para un algoritmo A, T_A^* tiene $n!$ hojas alcanzables. Entonces,

$$D(T_A^*) \;\geq\; d(n!) \;=\; \Omega(n! \log n!)$$

Por lo tanto,

$$\mathbb{E}[T_A^*] \;\geq\; D(T_A^*)/n! \;\geq\; d(n!)/n! \;=\; \Omega(\log n!) \;=\; \Omega(n \log n)$$

Cota para tiempo esperado

Ahora extendemos el análisis para **algoritmos randomizados** B

Consideramos árboles de decisión con dos tipo de nodos:

- **nodos de comparación** de tipo $i : j$ con dos hijos
- **nodos aleatorios** de tipo `Random(p,r)` con $r - p + 1$ hijos donde cada hijo denota una elección aleatoria en $\{p, \ldots, r\}$ con igual probabilidad

Cota para tiempo esperado

Mostraremos que para todo **algoritmo randomizado** B, existe un **algoritmo determinístico** A tal que el tiempo promedio de A no es mayor al tiempo esperado de B

Por lo tanto, la cota $\Omega(n \log n)$ para el número de comparaciones promedio también aplica para algoritmos randomizados

De-randomización de algoritmo randomizado

Dado un algoritmo randomizado B, lo convertimos en un algoritmo determinístico A haciendo:

1. Sea T igual al árbol T_B para B
2. Seleccionar un **nodo aleatorio n de menor altura** en T
3. Para cada hijo n_i de n, calcular el número N_i de comparaciones promedio realizadas por B "debajo" de n_i
4. Reemplazar en T el nodo aleatorio n por el nodo de comparación n_{i^*} donde n_{i^*} es un hijo con menor valor N_{i^*}
5. Repertir 2–5 mientras existan nodos aleatorios en T

Cotas para tiempo esperado

Al finalizar obtenemos un algoritmo determinístico A que por construcción:

- es un algoritmo de ordenamiento correcto (ordena todas las entradas)
- el número de **comparaciones promedio** de A es menor o igual al número de **comparaciones esperadas** de B

Cota para la mezcla de dos listas ordenadas

Queremos obtener una cota para el número de comparaciones necesarias para mezclar dos listas ordenadas de n elementos cada una

(Problema 8-6)

Como hicimos anteriormente, dado un algoritmo A representamos todas las comparaciones realizadas por A con un árbol de decisión cuyos nodos internos representan comparaciones y las hojas representan las posibles entradas (las dos listas de n elementos c/u)

Dado un árbol de decisión, su altura representa el número de comparaciones del algoritmo en el peor caso

Cota para la mezcla de dos listas ordenadas

Contemos las posibles entradas. Tenemos $2n$ objetos distintos en total donde cada lista contiene n objetos ordenados

Como los elementos están ordenados, lo único que diferencia dos entradas distintas es el conjunto de elementos en cada lista

El número total de entradas distintas sobre los $2n$ objetos distintos es $\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}(1 + O(\frac{1}{n}))$

Como todo algoritmo correcto debe tener al menos $\binom{2n}{n}$ hojas alcanzables, su altura es al menos $\log_2 \binom{2n}{n}$

Cota para la mezcla de dos listas ordenadas

Utilizamos la aproximación de Stirling $\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi n}}$:

$$\begin{aligned}\log_2 \binom{2n}{n} &\sim \log_2 \frac{4^n}{\sqrt{\pi n}} \\ &= 2n - \log_2 \sqrt{\pi n} \\ &= 2n - \alpha \log_2 n \\ &= 2n - o(n)\end{aligned}$$

Se necesitan $2n - o(n)$ comparaciones para mezclar las dos listas

Cota para la mezcla de dos listas ordenadas

Mejoremos el análisis

Si dos elementos son consecutivos en el orden total y ambos aparecen en listas distintas, entonces **todo algoritmo** debe comparar dichos elementos (¿por qué?)

Considere una instancia con elementos $\{1, 2, \ldots, 2n\}$ donde una lista contienen los pares y la otra los impares

La instancia contiene $2n - 1$ pares $(k, k + 1)$ donde k y $k + 1$ son consecutivos en el orden total y aparecen en listas distintas. Entonces, todo algoritmo debe realizar $2n - 1$ comparaciones en este caso

Todo algoritmo realiza al menos $\max\{2n - o(n), 2n - 1\} = 2n - 1$ comparaciones. `Merge(A,p,q,r)` realiza $2n$ comparaciones

Resumen

- Vimos cotas téoricas sobre el mínimo número de comparaciones que cualquier algoritmo debe realizar para ordenar n elementos
- Todo algoritmo de ordenamiento, determinístico o randomizado, basado en comparaciones debe realizar $\Omega(n \log n)$ comparaciones para ordenar n elementos, tanto en el peor caso como el caso promedio
- Todo algoritmo basado en comparaciones que mezcle dos listas ordenadas de n elementos cada una debe realizar $2n - 1$ comparaciones en el peor caso

Ejercicios (1 de 2)

1. (8.1-1) ¿Cuál puede ser la menor profundidad de una hoja en un árbol de decisión para un algoritmo de ordenamiento para n elementos?
2. (8.1-3) Muestre que no existe un algoritmo de ordenamiento basado en comparaciones que corra en tiempo lineal para al menos: la mitad de las entradas, una fracción $1/n$ de las entradas, y una fracción $1/2^n$ de las entradas
3. Mostrar que la función $f(x) = x \log_2 x + (k - x) \log_2(k - x)$ obtiene su mínimo sobre $x \in [1, k)$ para $x = k/2$
4. Muestre que todo algoritmo que mezcle dos listas de elementos ordenados usando comparaciones debe comparar dos elementos x y y que aparezcan en listas diferentes y que sean consecutivos en el orden total sobre todos los elementos

Ejercicios (2 de 2)

5. (8.1-4) Considere el siguiente problema. Se le dan n/k secuencias de elementos, cada una con k elementos, tal que los elementos de la i-ésima secuencia son todos mayor o igual a los elementos de la $(i - 1)$-ésima secuencia, y todos son menor o igual a los elementos de la $(i + 1)$-ésima secuencia. Se le pide ordenar las n elementos en las n/k secuencias. Claramente, un posible algoritmo es ordenar cada secuencia de forma independiente.

   Muestre una cota inferior de $\Omega(n \log k)$ comparaciones para resolver el problema y la optimalidad asintótica del algoritmo sugerido

   Nota: no es suficiente calcular la complejidad del algoritmo sugerido; se debe estudiar la complejidad del problema