Demostrando cotas inferiores: Arboles de decisión

De la misma forma que la técnica basada en la mejor estrategia del adversario, vamos a utilizar los árboles de decisión para establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos $\mathcal{C}$ que resuelven un problema específico.

En particular, vamos a utilizar esta técnica para algoritmos cuya operación básica es la comparación.

- Por ejemplo, buscar un elemento en una lista u ordenar una lista

Arboles de decisión para un algoritmo

Sea $\mathcal{A}$ un algoritmo en una clase $\mathcal{C}$ de algoritmos

- Recuerde que medimos la complejidad de $\mathcal{A}$ contando el número de comparaciones realizadas por $\mathcal{A}$

Para cada $n \in \mathbb{N}$ definimos un árbol de decisión $\mathcal{T}_{\mathcal{A},n}$ que describe el funcionamiento de $\mathcal{A}$ con las entradas de largo n

- Usamos el mismo árbol para todas las entradas de largo n
  - El árbol de decisión tiene un registro de las comparaciones hechas por el algoritmo $\mathcal{A}$
- Para distintos valores de n podemos tener distintos árboles de decisión

Las etiquetas en un árbol de decisión

$\mathcal{T}_{\mathcal{A},n}$ es un árbol con etiquetas en los nodos.

La etiqueta de un nodo u de $\mathcal{T}_{\mathcal{A},n}$:

- es una comparación si u es un nodo interno
- es una instrucción de la forma **return** v si u es una hoja, donde v es un valor retornado por $\mathcal{A}$

Los caminos en un árbol de decisión

Si v es un posible valor retornado por $\mathcal{A}$, entonces debe existir al menos una hoja de $\mathcal{T}_{\mathcal{A},n}$ con etiqueta **return** v

- Más de una hoja en $\mathcal{T}_{\mathcal{A},n}$ puede tener etiqueta **return** v

Dado un camino en $\mathcal{T}_{\mathcal{A},n}$ desde la raíz hasta una hoja con etiqueta **return** v:

- ¿Qué representa este camino? Una ejecución de $\mathcal{A}$ que retorna v
- ¿Qué representa el largo (número de arcos) de este camino? El número de comparaciones realizadas por $\mathcal{A}$ en una ejecución que retorna v

La profundidad de un árbol de decisión

¿Qué representa el camino más largo en $\mathcal{T}_{\mathcal{A},n}$ desde la raíz hasta las hojas?

- El peor caso para el algoritmo $\mathcal{A}$ para las entradas de largo n

Notación

profundidad($\mathcal{T}_{\mathcal{A},n}$): largo del camino de mayor longitud entre la raíz y las hojas de $\mathcal{T}_{\mathcal{A},n}$

Tenemos que $\mathcal{A}$ en el peor caso es $\Omega(\text{profundidad}(\mathcal{T}_{\mathcal{A},n}))$

La profundidad de un árbol de decisión

Si demostramos que profundidad($\mathcal{T}_{\mathcal{A},n}$) es $\Omega(f(n))$, entonces $\mathcal{A}$ en el peor caso es $\Omega(f(n))$

- Como $\mathcal{A}$ es un algoritmo arbitrario en $\mathcal{C}$, concluimos que todo algoritmos en $\mathcal{C}$ en el peor caso es $\Omega(f(n))$

¿Cómo podemos deducir una cota inferior para profundidad($\mathcal{T}_{\mathcal{A},n}$)?

- Una manera sencilla es utilizando el número de hojas del árbol

Las hojas de un árbol de decisión

Notación

hojas($\mathcal{T}_{\mathcal{A},n}$): número de hojas de $\mathcal{T}_{\mathcal{A},n}$

Dado que $\mathcal{T}_{\mathcal{A},n}$ es un árbol binario, tenemos la siguiente relación:

Lema

$hojas(\mathcal{T}_{\mathcal{A},n}) \leq 2^{profundidad(\mathcal{T}_{\mathcal{A},n})}$

Ejercicio

Demuestre el lema.

Las hojas de un árbol de decisión

Concluimos que $\log_2(\text{hojas}(\mathcal{T}_{\mathcal{A},n})) \leq \text{profundidad}(\mathcal{T}_{\mathcal{A},n})$

- Tenemos entonces una cota inferior para $\text{profundidad}(\mathcal{T}_{\mathcal{A},n})$

¿Pero cómo obtenemos una cota inferior para $\text{hojas}(\mathcal{T}_{\mathcal{A},n})$?

Sabemos que $\text{hojas}(\mathcal{T}_{\mathcal{A},n})$ debe ser mayor o igual al número de posibles valores retornados por $\mathcal{A}$ para las entradas de largo n

- En general esta cota inferior es suficiente para obtener una buena cota inferior para $\text{profundidad}(\mathcal{T}_{\mathcal{A},n})$

Vamos a ver dos aplicaciones de esta técnica ...

Encontrando un elemento en una lista ordenada

Sea $L[1 \ldots n]$ una lista de números enteros ordenada de menor a mayor, y sea a un número entero.

Queremos estudiar el número de comparaciones que debe realizar un algoritmo para determinar una posición i tal que $a = L[i]$ o retornar no en caso que a no esté en L

- Consideramos algoritmos cuya operación básica es la comparación

Ejercicio

Construya un algoritmo que resuelva el problema realizando $f(n)$ comparaciones, donde $f(n) \in \Theta(\log_2(n))$

Encontrando un elemento en una lista ordenada

¿Existe un algoritmo para encontrar un elemento en una lista ordenada que realice $g(n)$ comparaciones, donde $g(n)$ es de orden menor que $\log_2(n)$?

- ¿Qué significa que una función g sea de orden menor que una función h?

Definición

$$o(h) \quad = \quad \{f : \mathbb{N} \to \mathbb{R}_0^+ \mid (\forall c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)\,(f(n) \leq c \cdot h(n))\}$$

Encontrando un elemento en una lista ordenada

La pregunta reformulada: ¿Existe un algoritmo para encontrar un elemento en una lista ordenada que realice $g(n)$ comparaciones, donde $g(n) \in o(\log_2(n))$?

Vamos a demostrar que no existe tal algoritmo.

- Vamos a utilizar una técnica basada en árboles de decisión

Encontrando una cota inferior

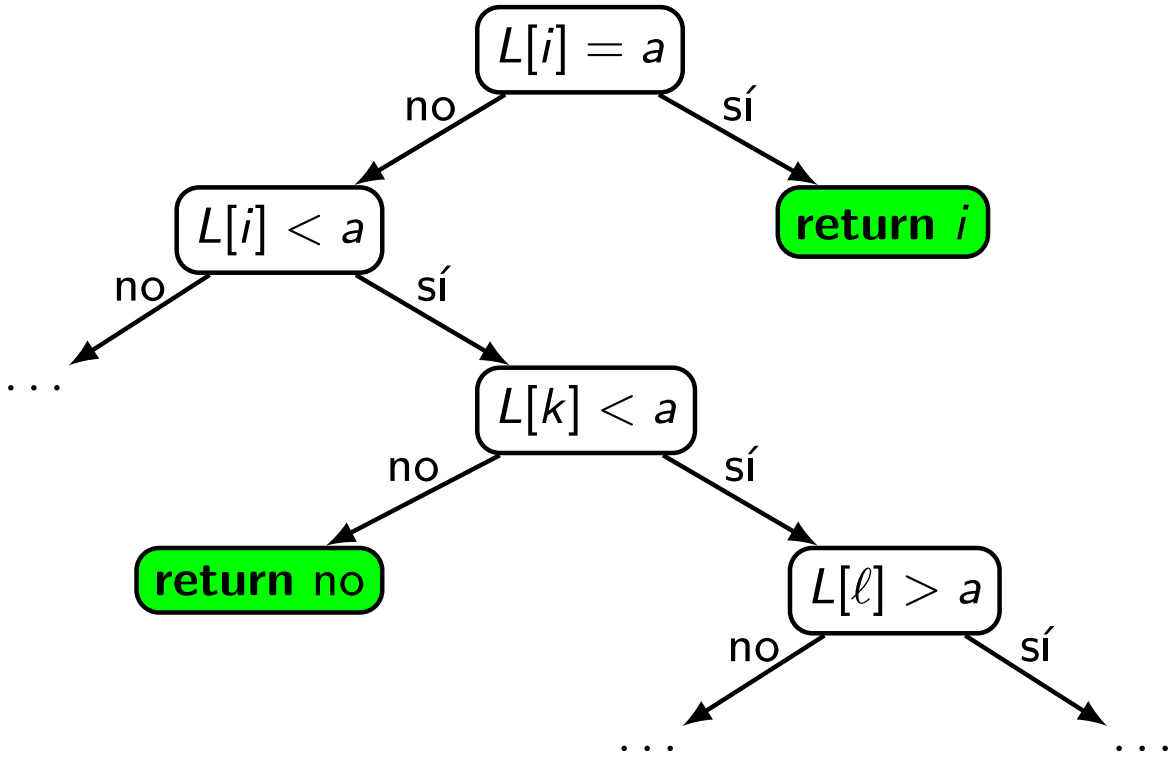
Sea $\mathcal{A}$ un algoritmo que resuelve el problema de encontrar un elemento en una lista ordenada

- Dada una lista ordenada de números enteros $L[1 \ldots n]$ y un número entero a, el algoritmo $\mathcal{A}$ determina una posición i tal que $a = L[i]$ o retorna no en caso que a no esté en L

Utilizamos un árbol de decisión $\mathcal{T}_{\mathcal{A},n}$ para describir el funcionamiento de $\mathcal{A}$ con las entradas (L, a) tales que la lista L tiene n elementos

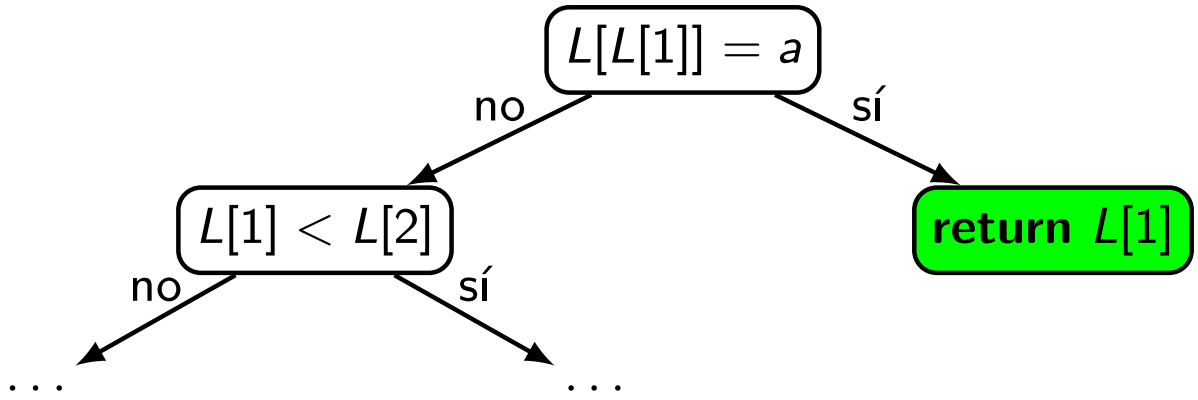
- Recuerde que debemos usar el mismo árbol para todas las entradas donde L tiene n elementos
- Para distintos valores de n podemos tener distintos árboles de decisión

El árbol de decisión $\mathcal{T}_{\mathcal{A},n}$



Algunas características de $\mathcal{T}_{\mathcal{A},n}$

Todas las comparaciones hechas por $\mathcal{A}$ son almacenadas en los nodos internos de $\mathcal{T}_{\mathcal{A},n}$, en las hojas se almacenan los valores retornados:



La etiqueta de un nodo u de $\mathcal{T}_{\mathcal{A},n}$:

- es una comparación si u es un nodo interno
- es una instrucción de la forma **return** no o **return** i, donde $i \in \{1, \ldots, n\}$, si u es una hoja

Una cota inferior para la búsqueda en una lista ordenada

En este caso tenemos que $n + 1 \leq \text{hojas}(\mathcal{T}_{\mathcal{A},n})$

Deducimos que $\log_2(n+1) \leq \text{profundidad}(\mathcal{T}_{\mathcal{A},n})$

Conclusión

Un algoritmo debe realizar al menos $\log_2(n+1)$ comparaciones para determinar, dado un entero a y una lista ordenada de enteros $L[1 \ldots n]$, una posición i tal que $a = L[i]$ o retornar no en caso que a no esté en L

- Todo algoritmo para encontrar un elemento en una lista ordenada y que esté basado en comparaciones en el peor caso es $\Omega(\log_2(n))$

Ordenando una lista

Sea $L[1 \ldots n]$ una lista de números enteros.

Queremos estudiar el número de comparaciones que debe realizar un algoritmo para ordenar L de menor a mayor.

- Al igual que en los casos anteriores consideramos algoritmos cuya operación básica es la comparación

Ejercicio

Construya un algoritmo que resuelva el problema realizando $f(n)$ comparaciones, donde $f(n) \in \Theta(n \cdot \log_2(n))$

Ordenando una lista

¿Existe un algoritmo para ordenar una lista que realice $g(n)$ comparaciones, donde $g(n) \in o(n \cdot \log_2(n))$?

Vamos a demostrar que no existe tal algoritmo.

- Vamos a utilizar una técnica basada en árboles de decisión

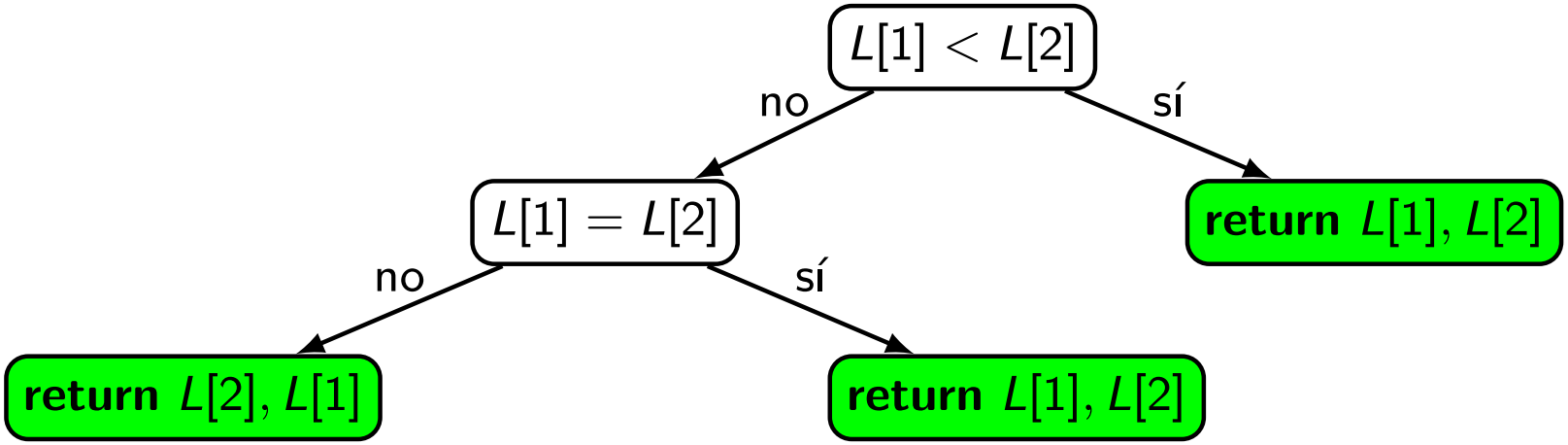
Encontrando una cota inferior

Sea $\mathcal{B}$ un algoritmo que resuelve el problema de ordenar una lista

- Dada una lista de números enteros $L[1 \ldots n]$, el algoritmo $\mathcal{B}$ retorna la lista L ordenada de menor a mayor

Utilizamos un árbol de decisión $\mathcal{T}_{\mathcal{B},n}$ para describir el funcionamiento de $\mathcal{B}$ con las entradas $L[1 \ldots n]$

El árbol de decisión $\mathcal{T}_{\mathcal{B},2}$



Una cota inferior para la ordenación

En este caso tenemos que $n! \leq \text{hojas}(\mathcal{T}_{\mathcal{B},n})$

- ¿Por qué?

Deducimos que $\log_2(n!) \leq \text{profundidad}(\mathcal{T}_{\mathcal{B},n})$

Lema

Para todo $n \in \mathbb{N}$ *se tiene que* $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

Una demostración del lema

El lema se cumple para $n = 0$ y $n = 1$

- Suponemos entonces que $n \geq 2$

Si $n = 2 \cdot k$ tenemos que:

$$
\begin{aligned}
n! &= (2 \cdot k)! \\
&= (2 \cdot k) \cdot (2 \cdot k - 1) \cdot \ldots \cdot (k + 1) \cdot k \cdot (k - 1) \cdot \ldots \cdot 1 \\
&\geq (2 \cdot k) \cdot (2 \cdot k - 1) \cdot \ldots \cdot (k + 1) \\
&> \underbrace{k \cdot k \cdot \ldots \cdot k}_{k \text{ veces}} \\
&= k^k \\
&= \left(\frac{n}{2}\right)^{\frac{n}{2}}
\end{aligned}
$$

Una demostración del lema

Finalmente, si $n = 2 \cdot k + 1$ tenemos que:

$$
\begin{aligned}
n! &= (2 \cdot k + 1)! \\
&= (2 \cdot k + 1) \cdot (2 \cdot k) \cdot (2 \cdot k - 1) \cdot \ldots \cdot (k + 1) \cdot k \cdot (k - 1) \cdot \ldots \cdot 1 \\
&\geq (2 \cdot k + 1) \cdot (2 \cdot k) \cdot (2 \cdot k - 1) \cdot \ldots \cdot (k + 1) \\
&> \underbrace{(k + 1) \cdot (k + 1) \cdot \ldots \cdot (k + 1)}_{(k+1) \text{ veces}} \\
&= (k + 1)^{(k+1)} \\
&> \left(\frac{n}{2}\right)^{\frac{n}{2}}
\end{aligned}
$$

□

Una cota inferior para la ordenación

Del lema deducimos que $\frac{n}{2} \cdot \log_2 \left(\frac{n}{2}\right) \leq \log_2(n!) \leq \text{profundidad}(\mathcal{T}_{\mathcal{B},n})$

Conclusión

Un algoritmo debe realizar al menos $\lceil \frac{n}{2} \cdot \log_2(\frac{n}{2}) \rceil$ comparaciones para ordenar una lista de números enteros.

- Todo algoritmo de ordenación basado en comparaciones en el peor caso es $\Omega(n \cdot \log_2 n)$