Estructuras de Datos y Algoritmos – IIC2133

Outline

- Propiedades de Insertion Sort
- El problema de ordenar

Outline

- Propiedades de Insertion Sort
- El problema de ordenar

Propiedades de Insertion Sort

Insertion Sort es un algoritmo correcto

¿Qué propiedades debe tener para ser correcto?

Correctitud de Insertion Sort

Para que un algoritmo sea correcto, debe:

- Terminar su ejecución
- Cumplir su objetivo

En este caso el objetivo es ordenar

Propiedades de Insertion Sort

¿Cómo demostramos que el algoritmo termina?

¿Cómo demostramos que ordena?

El algoritmo *Insertion Sort*

Para la secuencia inicial de datos, A:

1. Definir una secuencia ordenada, B, inicialmente vacía
2. Tomar el primer dato x de A y sacarlo de A
3. Insertar x en B de manera que B quede ordenado
4. Si quedan elementos en A, volver a 2.

Finitud

En cada paso se saca un elemento de A y se inserta en B

Cuando no quedan elementos en A, el algoritmo termina

La inserción requiere como máximo recorrer todo B

Como A y B son finitos, el algoritmo termina en tiempo finito

Corrección

PD: Al terminar la n-ésima iteración, B se encuentra ordenada

Podemos hacerlo por **inducción** sobre i

Caso Base: Después de la primera iteración, B tiene un solo dato

$\rightarrow B$ está ordenada

Hipótesis Inductiva: Después de la i-ésima iteración**,** B está ordenada

Demostraremos que después de la iteración $i+1$, B está ordenada

Extraemos el primer elemento de A, y lo insertamos ordenadamente en B.
Termina el paso $i+1$ y B tiene $i\ +\ 1$ elementos ordenados

En particular, al terminar el algoritmo después del paso n, B está ordenada.

Complejidad de Insertion Sort

Vimos la clase pasada que Insertion Sort es $O(n^2)$

Pero, ¿Qué tiempo toma si los datos vienen ordenados?

$$\mathit{insertionSort}(A, n):$$

$$\quad \mathit{for}\ i = 1\,..\,n-1:$$

$$\quad\quad j = i$$

$$\quad\quad \mathit{while}\ (j > 0) \wedge (A[j] < A[j-1]):$$

$$\quad\quad\quad \text{Intercambiar } A[j] \text{ con } A[j-1]$$

$$\quad\quad\quad j = j - 1$$

Complejidad de Insertion Sort

Parecería que la complejidad de insertion sort depende de que tan ordenados vienen los datos.

¿Cómo podemos medir esto?

Inversiones

Sea A un arreglo con n números distintos de 1 a n

Si $i < j$ pero $\mathrm{A[i]} > \mathrm{A[j]}$, se dice que (i, j) son un **par invertido**

El número de pares invertidos es una métrica de **desorden**

Complejidad de Insertion Sort

Tenemos un arreglo A de largo n que tiene k **inversiones**

¿Cuanto tiempo toma Insertion Sort en ordenar A?

¿Cuántas inversiones se arreglan con un intercambio?

$\mathit{insertionSort}(A, n)$:

$\mathit{for}\ i = 1 \ldots n - 1$:

$j = i$

$\mathit{while}\ (j > 0) \wedge (A[j] < A[j-1])$:

Intercambiar $A[j]$ con $A[j-1]$

$j = j - 1$

Antes de cada intercambio se hace una comparación, y esos datos se intercambian solo si están invertidos

Por lo tanto, cada intercambio de elementos adyacentes en el arreglo deshace exactamente una inversión

Además, cada elemento se compara al menos una vez

Complejidad de Insertion Sort

La complejidad es entonces $O(n + k)$

¿Qué valor tiene k en el mejor caso? ¿Y el en peor?

¿Qué hay del caso promedio?

Complejidad de Insertion Sort

Sea $K(A)$ el número de pares invertidos en A

Sea $H(A)$ el número de pares no invertidos en A

¿Cuánto es $K(A) + H(A)$ si A tiene n elementos?

La cantidad de pares que existen en un arreglo de n elementos es

$$\frac{n^2 - n}{2}$$

Cada par puede estar ó no estar invertido, por lo tanto:

$$K(A) + H(A) = \frac{n^2 - n}{2}$$

Dado n, definimos $\mathcal{A}$ como la secuencia ordenada de todos los números del 1 al n.

Sea $p(\mathcal{A})$ una permutación de $\mathcal{A}$. Definimos $P(\mathcal{A})$ como el conjunto de todas las posibles permutaciones de $\mathcal{A}$.

$P(\mathcal{A})$ contiene todos los posibles inputs de largo n que puede recibir un algoritmo de ordenación.

Complejidad de Insertion Sort

Sea $P(\mathcal{A})$ el conjunto de permutaciones de $\mathcal{A}$

Queremos encontrar el número de inversiones promedio:

$$\mathbb{E}\big(K(\mathcal{A})\big) = \frac{1}{|P(\mathcal{A})|} \sum_{a \in P(\mathcal{A})} K(a)$$

Notar que

$$\mathbb{E}\big(K(\mathcal{A})\big) = \mathbb{E}\big(H(\mathcal{A})\big)$$

Al sumarlos, queda

$$\mathbb{E}\big(K(\mathcal{A})\big) + \mathbb{E}\big(H(\mathcal{A})\big) = \frac{1}{|P(\mathcal{A})|}\left(\sum_{a\in P(\mathcal{A})} K(a) + \sum_{a\in P(\mathcal{A})} H(a)\right)$$

$$2 \cdot \mathbb{E}\big(K(\mathcal{A})\big) = \frac{1}{|P(\mathcal{A})|}\left(\sum_{a\in P(\mathcal{A})} K(a) + H(a)\right)$$

$$2 \cdot \mathbb{E}\big(K(\mathcal{A})\big) = \frac{1}{|P(\mathcal{A})|}\left(\sum_{a \in P(\mathcal{A})} K(a) + H(a)\right)$$

$$2 \cdot \mathbb{E}\big(K(\mathcal{A})\big) = \frac{1}{|P(\mathcal{A})|}\left(\sum_{a \in P(\mathcal{A})} \frac{n^2 - n}{2}\right)$$

$$2 \cdot \mathbb{E}\big(K(\mathcal{A})\big) = \frac{1}{|P(\mathcal{A})|} \cdot |P(\mathcal{A})| \cdot \frac{n^2 - n}{2}$$

$$\mathbb{E}\big(K(\mathcal{A})\big) = \frac{n^2 - n}{4} \in \Theta(n^2)$$

Complejidad de *Insertion Sort*

La cantidad de inversiones promedio es entonces $O(n^2)$

Eso significa que Insertion Sort es $O(n^2)$ en el caso promedio

Si un algoritmo sólo resuelve una inversión por intercambio, no puede ordenar más rápido que $O(n^2)$ en promedio y por lo tanto en el peor caso

Outline

- Propiedades de Insertion Sort
- El problema de ordenar

El problema de ordenar

No podemos ordenar más rápido que $O(n^2)$ resolviendo solo una inversión a la vez

¿Cómo podemos resolver más de una inversión por paso?

El problema de ordenar

Debemos intercambiar elementos no adyacentes

¿Qué tan rápido puede ordenar un algoritmo haciendo esto?

El problema de ordenar

Definimos **problema** como una relación entre input y output

Una **instancia** de un problema es un input específico

¿Cómo podríamos definir formalmente el **problema de ordenar**?

El problema de ordenar

De la manera más general posible:

Input: Una secuencia $A = p(\mathcal{A})$

Output: Una permutación p^{-1} tal que $p^{-1}(A) = \mathcal{A}$

Es decir buscamos la permutación p^{-1} que “deshace” p

El problema de ordenar

¿Cuántas posibles soluciones tiene una instancia?

¿Qué pasa si hay elementos repetidos?

Una instancia sin elementos repetidos tiene una única solución.

Si hay elementos repetidos, significa que hay más de una posible solución. Esto hace que el problema sea **más fácil**.

Nos importa analizar que tan difícil es el problema, por lo que analizaremos el caso más difícil: sin elementos repetidos.

El problema de ordenar

Para una instancia, debemos encontrar la única permutación que la resuelve.

¿Cuántas permutaciones hay?

¿Cuántas posibles instancias hay?

El conjunto de instancias se define como:

$$P(\mathcal{A}) = \{\, p(\mathcal{A}) \quad | \quad p \text{ es una permutación} \,\}$$

Una solución para una instancia $p(\mathcal{A})$ es una permutación p^{-1} tal que $p^{-1}\big(p(\mathcal{A})\big) = \mathcal{A}$. Todas estas permutaciones forman el conjunto de soluciones:

$$S = \{\, p \quad | \quad p \text{ es una permutación} \,\}$$

El problema de ordenar

Hay $n!$ instancias y $n!$ permutaciones, y cada instancia tiene como solución una permutación en particular.

¿Es cada permutación solución de una única instancia?

La respuesta es si, pero debemos definir formalmente las permutaciones para ver por que. Una forma de hacerlo es usando matrices de permutación:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

La inversa de una matriz de permutación es su traspuesta

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Definiendo las permutaciones con matrices es fácil ver que

$$p^{-1}\big(p(\mathcal{A})\big) = p\big(p^{-1}(\mathcal{A})\big)$$

Ya que $p^{-1} \cdot p = I$ (la matriz identidad)

Como la inversa de una matriz es única, cada permutación es solución de una sola instancia.

El problema de ordenar

Tenemos entonces una correspondencia 1:1 entre el conjunto de instancias y el de soluciones.

Volviendo a la pregunta original,

¿En cuantos pasos podemos encontrar la solución haciendo intercambios?

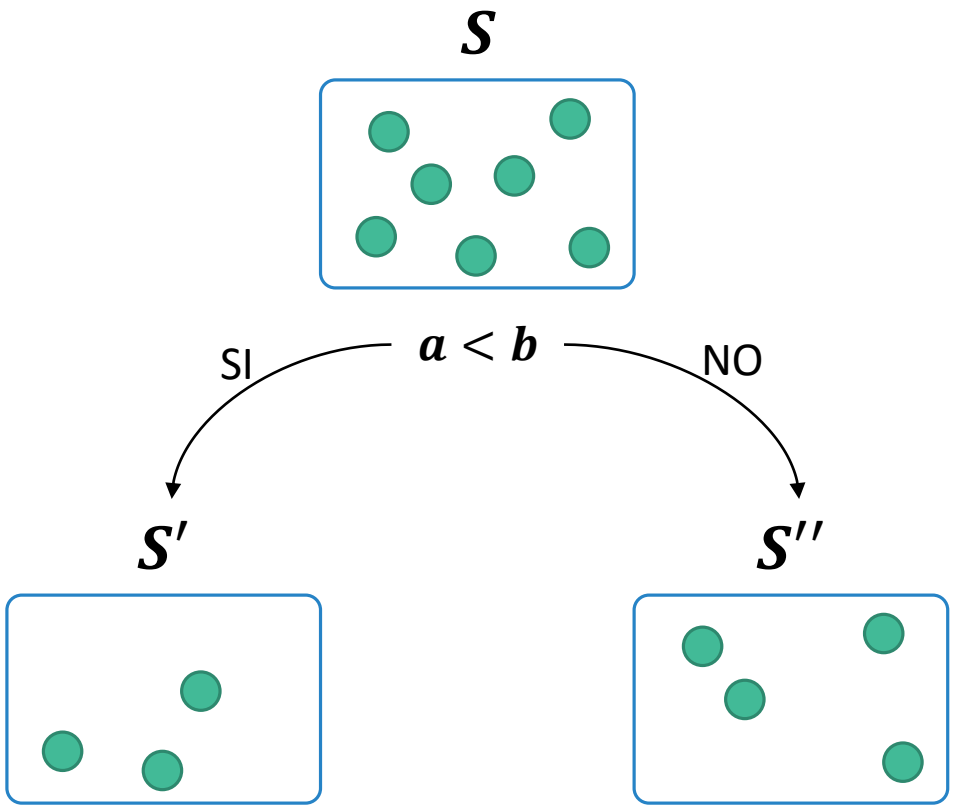
Lo que busca cualquier algoritmo de ordenación es encontrar la permutación solución para esa instancia.

Para esto debe ser capaz de descartar todas las otras permutaciones que no son solución.

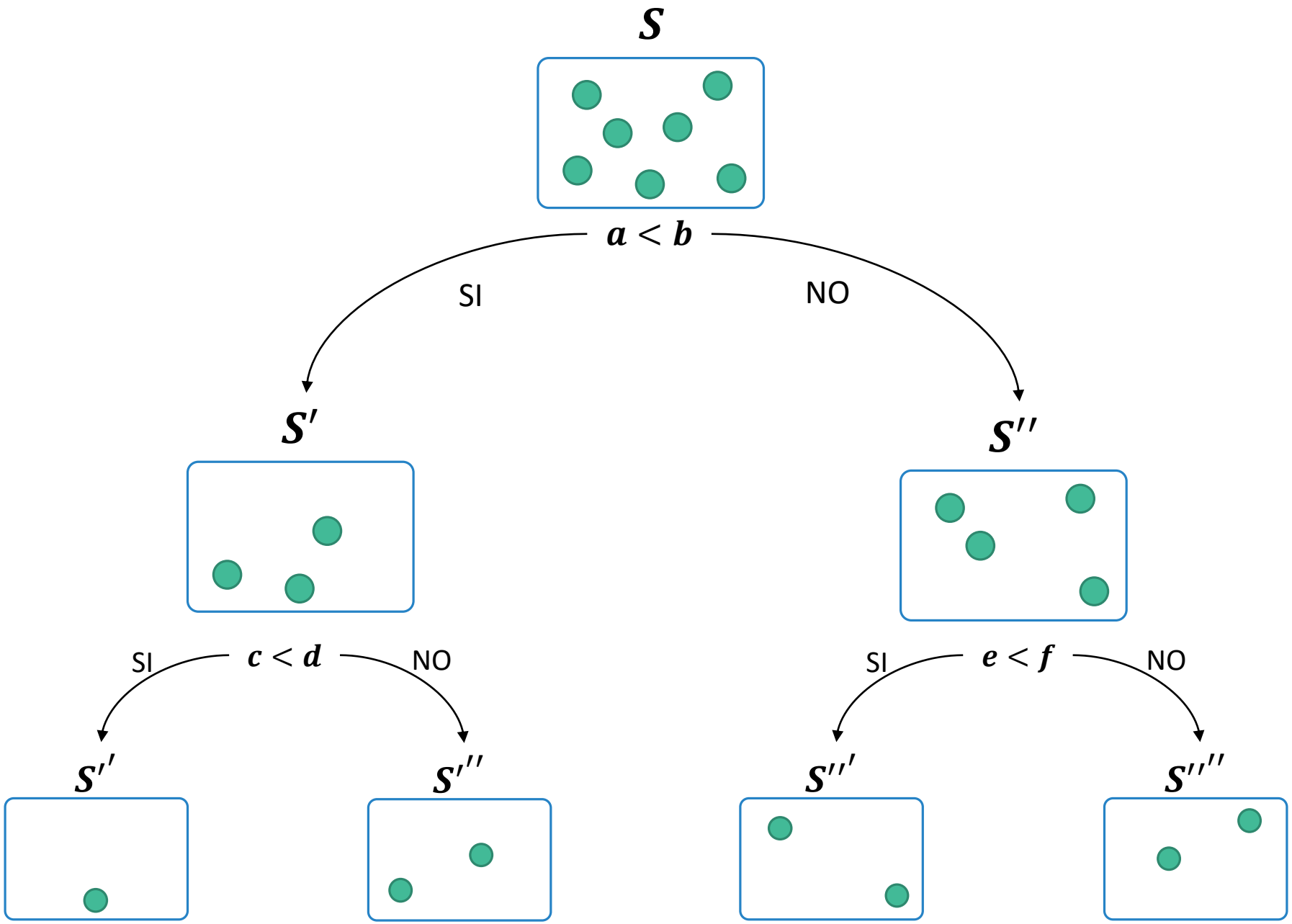
Los algoritmos que funcionan en base a comparación e intercambio cuentan con esta única herramienta para decidir que elementos de S son o no posibles soluciones.

Qué tan rápido se puede ordenar

¿Qué hace el algoritmo al hacer una comparación?



En cada comparación, podemos separar el conjunto de permutaciones en 2: las que sabemos que no son solución, y las que podrían serlo. En el mejor caso esta separación se hace de manera equitativa, pero no tiene por que serlo.


S
a < b
SI
NO
S′
S″
c < d
SI
NO
e < f
SI
NO
S′′′
S′′′′
S′′′′′

Qué tan rápido se puede ordenar

Pero necesitamos que sea posible llegar a cada permutación para cada posible instancia.

Para esto el algoritmo debe realizar suficientes comparaciones de manera que cada hoja del árbol tenga una única permutación.

Si dividimos equitativamente en cada paso y ejecutamos h comparaciones, tenemos un árbol con 2^h hojas.

Para que cada permutación pueda tener su propia hoja, necesitamos tener al menos $n!$ hojas en el árbol. Para esto se debe cumplir que:

$$2^h > n!$$

$$h > \log_2 n!$$

Podemos acotar $n!$

$$n! = 1 \cdot 2 \cdot 3 \dots \left(\frac{n}{2} - 1\right) \cdot \left(\frac{n}{2}\right) \cdot \left(\frac{n}{2} + 1\right) \dots (n-1) \cdot n$$

$$n! < n \cdot n \cdot n \dots \; n \cdot n \cdot n \dots \; n \cdot n = n^n$$

Por lo tanto

$$\log_2 n! < n \cdot \log_2 n$$

$$n! = 1 \cdot 2 \cdot 3 \dots \left(\frac{n}{2} - 1\right) \cdot \left(\frac{n}{2}\right) \cdot \left(\frac{n}{2} + 1\right) \dots (n-1) \cdot n$$

$$n! > 1 \cdot 1 \cdot 1 \dots 1 \cdot \frac{n}{2} \cdot \frac{n}{2} \dots \frac{n}{2} \cdot \frac{n}{2} = \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Por lo tanto

$$\log_2 n! > \frac{n}{2} \cdot \log_2 \frac{n}{2}$$

Cota inferior de ordenación

$$\log_2 n! \in \Theta(n \cdot \log n)$$

Entonces

$$h \in \Omega(n \cdot \log n)$$

Esto significa que:

Cualquier algoritmo de ordenación por comparación debe ejecutar como mínimo $n \cdot \log n$ comparaciones en su peor caso.