Técnicas para demostrar cotas inferiores

Queremos establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos que resuelven un problema específico.

Técnicas para demostrar cotas inferiores

Queremos establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos que resuelven un problema específico.

- Por ejemplo, una cota inferior para los algoritmos que ordenan una lista de números enteros y cuya operación básica es la comparación

Técnicas para demostrar cotas inferiores

Queremos establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos que resuelven un problema específico.

- Por ejemplo, una cota inferior para los algoritmos que ordenan una lista de números enteros y cuya operación básica es la comparación

Vamos a estudiar tres técnicas para demostrar cotas inferiores:

- Mejor estrategia del adversario
- Arboles de decisión
- Reducciones

Calculando el máximo de una lista

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

Calculando el máximo de una lista

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

Queremos estudiar el número de comparaciones que debe realizar un algoritmo para encontrar el máximo elemento de L

- Consideramos algoritmos cuya operación básica es la comparación

Calculando el máximo de una lista

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

Queremos estudiar el número de comparaciones que debe realizar un algoritmo para encontrar el máximo elemento de L

- Consideramos algoritmos cuya operación básica es la comparación

Ejercicio

Construya un algoritmo que encuentre el máximo elemento de $L[1 \ldots n]$ realizando $n - 1$ comparaciones de elementos de la lista.

Calculando el máximo de una lista

¿Es posible calcular el máximo elemento de L con menos de $n - 1$ comparaciones de elementos de L?

Calculando el máximo de una lista

¿Es posible calcular el máximo elemento de L con menos de $n - 1$ comparaciones de elementos de L?

Vamos a demostrar que no es posible.

- Vamos a utilizar una técnica basada en buscar la mejor estrategia del adversario

Una técnica basada en la mejor estrategia del adversario

Recuerde que vamos a utilizar esta técnica para establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos $\mathcal{C}$ que resuelven un problema específico.

Una técnica basada en la mejor estrategia del adversario

Recuerde que vamos a utilizar esta técnica para establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos $\mathcal{C}$ que resuelven un problema específico.

Lo primero que necesitamos entonces es una caracterización general de los algoritmos $\mathcal{A}$ que pertenecen a $\mathcal{C}$

Una técnica basada en la mejor estrategia del adversario

Recuerde que vamos a utilizar esta técnica para establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos $\mathcal{C}$ que resuelven un problema específico.

Lo primero que necesitamos entonces es una caracterización general de los algoritmos $\mathcal{A}$ que pertenecen a $\mathcal{C}$

Esta caracterización debe servir para identificar qué puede decidir $\mathcal{A}$ y qué depende del medio externo.

Una técnica basada en la mejor estrategia del adversario

Recuerde que vamos a utilizar esta técnica para establecer una cota inferior para el número de operaciones realizadas por una clase de algoritmos $\mathcal{C}$ que resuelven un problema específico.

Lo primero que necesitamos entonces es una caracterización general de los algoritmos $\mathcal{A}$ que pertenecen a $\mathcal{C}$

Esta caracterización debe servir para identificar qué puede decidir $\mathcal{A}$ y qué depende del medio externo.

- El medio externo es el adversario

Una técnica basada en la mejor estrategia del adversario

En la medida que $\mathcal{A}$ procesa una entrada:

- El adversario toma decisiones que ponen $\mathcal{A}$ en el peor caso
- $\mathcal{A}$ responde de la mejor manera posible

Una técnica basada en la mejor estrategia del adversario

En la medida que $\mathcal{A}$ procesa una entrada:

- El adversario toma decisiones que ponen $\mathcal{A}$ en el peor caso
- $\mathcal{A}$ responde de la mejor manera posible

A partir de esta interacción establecemos una cota inferior para el número de operaciones realizadas por $\mathcal{A}$

- Esta es una cota inferior para el número de operaciones realizadas por los algoritmos en $\mathcal{C}$, ya que $\mathcal{A}$ es un elemento arbitrario de $\mathcal{C}$

Calculando el máximo: mejor estrategia del adversario

Sea $\mathcal{A}$ un algoritmo para calcular el máximo elemento de una lista de números enteros sin repeticiones y cuya operación básica es la comparación.

- Suponemos que $\mathcal{A}$ no compara un elemento consigo mismo. ¿Es razonable suponer esto? ¿Cómo se puede implementar esto?

Calculando el máximo: mejor estrategia del adversario

Sea $\mathcal{A}$ un algoritmo para calcular el máximo elemento de una lista de números enteros sin repeticiones y cuya operación básica es la comparación.

- Suponemos que $\mathcal{A}$ no compara un elemento consigo mismo. ¿Es razonable suponer esto? ¿Cómo se puede implementar esto?

Caracterizamos $\mathcal{A}$ como un torneo donde el máximo es el ganador.

- Si b es comparado con c por $\mathcal{A}$, decimos que b gana la comparación si $b > c$

Calculando el máximo: mejor estrategia del adversario

Suponga que $L[1 \ldots n]$ es una lista de números enteros que es dada como entrada a $\mathcal{A}$

- Recuerde L no tiene elementos repetidos

Calculando el máximo: mejor estrategia del adversario

Suponga que $L[1 \ldots n]$ es una lista de números enteros que es dada como entrada a $\mathcal{A}$

- Recuerde L no tiene elementos repetidos

Para describir el funcionamiento de $\mathcal{A}$ utilizamos los siguientes conjuntos que muestran el estado del algoritmo en cada instance de tiempo:

U : elementos de L que todavía no han sido comparados

G : elementos de L que han ganado todas sus comparaciones (y al menos han participado en una comparación)

P : elementos de L que perdieron al menos una comparación

Calculando el máximo: mejor estrategia del adversario

Sean $u = |U|$, $g = |G|$ y $p = |P|$

En cada instante de tiempo el vector (u, g, p) describe el estado de $\mathcal{A}$

Calculando el máximo: mejor estrategia del adversario

Sean $u = |U|$, $g = |G|$ y $p = |P|$

En cada instante de tiempo el vector (u, g, p) describe el estado de $\mathcal{A}$

- En cada instante se tiene que $u + g + p = n$

Calculando el máximo: mejor estrategia del adversario

Sean $u = |U|$, $g = |G|$ y $p = |P|$

En cada instante de tiempo el vector (u, g, p) describe el estado de $\mathcal{A}$

- En cada instante se tiene que $u + g + p = n$
- $\mathcal{A}$ encontró el máximo elemento de L si $(u, g, p) = (0, 1, n - 1)$

Calculando el máximo: mejor estrategia del adversario

La siguiente tabla describe el cambio del vector (u, g, p) dependiendo de la comparación $b > c$:

	$c \in U$	$c \in G$	$c \in P$
$b \in U$	$(u-2, g+1, p+1)$	$(u-1, g, p+1)$	$b > c$: $(u-1, g+1, p)$ $b < c$: $(u-1, g, p+1)$
$b \in G$		$(u, g-1, p+1)$	$b > c$: (u, g, p) $b < c$: $(u, g-1, p+1)$
$b \in P$			(u, g, p)

Calculando el máximo: mejor estrategia del adversario

La siguiente tabla describe el cambio del vector (u, g, p) dependiendo de la comparación $b > c$:

	$c \in U$	$c \in G$	$c \in P$
$b \in U$	$(u-2, g+1, p+1)$	$(u-1, g, p+1)$	$b > c$: $(u-1, g+1, p)$ $b < c$: $(u-1, g, p+1)$
$b \in G$		$(u, g-1, p+1)$	$b > c$: (u, g, p) $b < c$: $(u, g-1, p+1)$
$b \in P$			(u, g, p)

Las preguntas fundamentales: ¿Qué elige el algoritmo? ¿Qué elige el adversario?

Las decisiones del algoritmo y el adversario

$\mathcal{A}$ escoge los elementos b y c a comparar

- En particular, elige los conjuntos desde los cuales sacar estos elementos

Las decisiones del algoritmo y el adversario

$\mathcal{A}$ escoge los elementos b y c a comparar

- En particular, elige los conjuntos desde los cuales sacar estos elementos

El adversario elige el peor caso según la decisión tomada por $\mathcal{A}$:

Las decisiones del algoritmo y el adversario

$\mathcal{A}$ escoge los elementos b y c a comparar

- En particular, elige los conjuntos desde los cuales sacar estos elementos

El adversario elige el peor caso según la decisión tomada por $\mathcal{A}$:

	$c \in U$	$c \in G$	$c \in P$
$b \in U$	$(u-2, g+1, p+1)$	$(u-1, g, p+1)$	$b > c$: $(u-1, g+1, p)$ $b < c$: $(u-1, g, p+1)$
$b \in G$		$(u, g-1, p+1)$	$b > c$: (u, g, p) $b < c$: $(u, g-1, p+1)$
$b \in P$			(u, g, p)

Una cota inferior para el cálculo del máximo

Dado un vector (u, g, p), una comparación de $\mathcal{A}$ da como resultado un vector (u', g', p') tal que:

$$p' = p \quad \text{o} \quad p' = p + 1$$

Una cota inferior para el cálculo del máximo

Dado un vector (u, g, p), una comparación de $\mathcal{A}$ da como resultado un vector (u', g', p') tal que:

$$p' = p \quad \text{o} \quad p' = p + 1$$

Entonces para que $\mathcal{A}$ llegue al vector $(0, 1, n-1)$ debe realizar al menos $n - 1$ comparaciones

Una cota inferior para el cálculo del máximo

Dado un vector (u, g, p), una comparación de $\mathcal{A}$ da como resultado un vector (u', g', p') tal que:

$$p' = p \quad \text{o} \quad p' = p + 1$$

Entonces para que $\mathcal{A}$ llegue al vector $(0, 1, n-1)$ debe realizar al menos $n-1$ comparaciones

Conclusión

Un algoritmo debe realizar al menos $n-1$ comparaciones para encontrar el máximo elemento de una lista con n elementos no repetidos, suponiendo que el algoritmo no compara un elemento consigo mismo.

Calculando el máximo: eliminando una restricción

¿Qué pasa si permitimos al algoritmo $\mathcal{A}$ de las transparencias anteriores comparar un elemento consigo mismo?

Calculando el máximo: eliminando una restricción

¿Qué pasa si permitimos al algoritmo $\mathcal{A}$ de las transparencias anteriores comparar un elemento consigo mismo?

Para describir el funcionamiento de $\mathcal{A}$ modificamos los conjuntos U, G y P de la siguiente forma:

- U : elementos de L que todavía no han sido comparados o sólo han sido comparados consigo mismo
- G : elementos de L que han ganado todas sus comparaciones con elementos distintos (y al menos han participado en una comparación con un elemento distinto)
- P : elementos de L que perdieron al menos una comparación con un elemento distinto

Calculando el máximo: eliminando una restricción

Tenemos entonces que usar la siguiente tabla para describir los cambios del vector (u, g, p):

	$c \in U$	$c \in G$	$c \in P$
$b \in U$	$b \neq c$: $(u-2, g+1, p+1)$ $b = c$: (u, g, p)	$(u-1, g, p+1)$	$b > c$: $(u-1, g+1, p)$ $b < c$: $(u-1, g, p+1)$
$b \in G$		$b \neq c$: $(u, g-1, p+1)$ $b = c$: (u, g, p)	$b > c$: (u, g, p) $b < c$: $(u, g-1, p+1)$
$b \in P$			(u, g, p)

Calculando el máximo: eliminando una restricción

Es claro que al algoritmo $\mathcal{A}$ no le conviene comparar un elemento consigo mismo.

- El valor de p no aumenta en este caso

Calculando el máximo: eliminando una restricción

Es claro que al algoritmo $\mathcal{A}$ no le conviene comparar un elemento consigo mismo.

- El valor de p no aumenta en este caso

Al igual que para el caso anterior concluimos que:

Conclusión

Un algoritmo debe realizar al menos $n - 1$ comparaciones para encontrar el máximo elemento de una lista con n elementos no repetidos.

Calculando el máximo y el mínimo de una lista

Sea $L[1 \ldots n]$ una lista de número enteros sin elementos repetidos.

Calculando el máximo y el mínimo de una lista

Sea $L[1 \ldots n]$ una lista de número enteros sin elementos repetidos.

Queremos estudiar el número de comparaciones que debe realizar un algoritmo para encontrar tanto el máximo como el mínimo elemento de L

- Nuevamente consideramos algoritmos cuya operación básica es la comparación

Calculando el máximo y el mínimo de una lista

Sea $L[1 \ldots n]$ una lista de número enteros sin elementos repetidos.

Queremos estudiar el número de comparaciones que debe realizar un algoritmo para encontrar tanto el máximo como el mínimo elemento de L

- Nuevamente consideramos algoritmos cuya operación básica es la comparación

Ejercicio

Construya un algoritmo que encuentre el máximo y el mínimo elemento de $L[1 \ldots n]$ realizando $2 \cdot n - 3$ comparaciones de elementos de la lista.

Calculando el máximo y el mínimo de una lista

¿Es posible calcular el máximo y el mínimo elemento de L con menos de $2 \cdot n - 3$ comparaciones de elementos de L?

Calculando el máximo y el mínimo de una lista

¿Es posible calcular el máximo y el mínimo elemento de L con menos de $2 \cdot n - 3$ comparaciones de elementos de L?

Vamos a utilizar la técnica basada en la mejor estrategia del adversario para encontrar una cota inferior para este problema.

- Esto nos va a dar una idea de cómo construir un algoritmo más eficiente

Máximo y mínimo: mejor estrategia del adversario

Sea $\mathcal{A}$ un algoritmo para calcular el máximo y el mínimo elemento de una lista de números enteros sin repeticiones y cuya operación básica es la comparación.

- Suponemos que $\mathcal{A}$ no compara un elemento consigo mismo, la misma conclusión se puede obtener si eliminamos esta restricción

Máximo y mínimo: mejor estrategia del adversario

Sea $\mathcal{A}$ un algoritmo para calcular el máximo y el mínimo elemento de una lista de números enteros sin repeticiones y cuya operación básica es la comparación.

- Suponemos que $\mathcal{A}$ no compara un elemento consigo mismo, la misma conclusión se puede obtener si eliminamos esta restricción

Al igual que para el caso del cálculo del máximo, caracterizamos $\mathcal{A}$ como un torneo donde el máximo y el mínimo son los ganadores.

Máximo y mínimo: mejor estrategia del adversario

Suponga que $L[1 \ldots n]$ es una lista de números enteros sin repeticiones que es dada como entrada a $\mathcal{A}$, donde $n \geq 2$

Para describir el funcionamiento de $\mathcal{A}$ utilizamos los siguientes conjuntos que muestran el estado del algoritmo en cada instance de tiempo:

- U : elementos de L que todavía no han sido comparados
- G : elementos de L que han ganado todas sus comparaciones (y al menos han participado en una comparación)
- P : elementos de L que han perdido todas sus comparaciones
- E : elementos de L que ganaron al menos una comparación y perdieron al menos una comparación

Máximo y mínimo: mejor estrategia del adversario

Suponga que $L[1 \ldots n]$ es una lista de números enteros sin repeticiones que es dada como entrada a $\mathcal{A}$, donde $n \geq 2$

Para describir el funcionamiento de $\mathcal{A}$ utilizamos los siguientes conjuntos que muestran el estado del algoritmo en cada instance de tiempo:

- U : elementos de L que todavía no han sido comparados
- G : elementos de L que han ganado todas sus comparaciones (y al menos han participado en una comparación)
- P : elementos de L que han perdido todas sus comparaciones
- E : elementos de L que ganaron al menos una comparación y perdieron al menos una comparación

¿Por qué en este caso utilizamos los conjuntos P y E?

Máximo y mínimo: mejor estrategia del adversario

Sean $u = |U|$, $g = |G|$, $p = |P|$ y $e = |E|$

En cada instante de tiempo el vector (u, g, p, e) describe el estado de $\mathcal{A}$

Máximo y mínimo: mejor estrategia del adversario

Sean $u = |U|$, $g = |G|$, $p = |P|$ y $e = |E|$

En cada instante de tiempo el vector (u, g, p, e) describe el estado de $\mathcal{A}$

- En cada instante se tiene que $u + g + p + e = n$

Máximo y mínimo: mejor estrategia del adversario

Sean $u = |U|$, $g = |G|$, $p = |P|$ y $e = |E|$

En cada instante de tiempo el vector (u, g, p, e) describe el estado de $\mathcal{A}$

- En cada instante se tiene que $u + g + p + e = n$
- $\mathcal{A}$ encontró el máximo y el mínimo elemento de L si $(u, g, p, e) = (0, 1, 1, n - 2)$

Máximo y mínimo: mejor estrategia del adversario

Al igual que para el cálculo del máximo, la siguiente tabla describe el cambio del vector (u, g, p, e) dependiendo de la comparación $b > c$:

	$c \in U$	$c \in G$	$c \in P$	$c \in E$
$b \in U$	$(u-2, g+1, p+1, e)$	$b > c$: $(u-1, g, p, e+1)$ $b < c$: $(u-1, g, p+1, e)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p, e+1)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p+1, e)$
$b \in G$		$(u, g-1, p, e+1)$	$b > c$: (u, g, p, e) $b < c$: $(u, g-1, p-1, e+2)$	$b > c$: (u, g, p, e) $b < c$: $(u, g-1, p, e+1)$
$b \in P$			$(u, g, p-1, e+1)$	$b > c$: $(u, g, p-1, e+1)$ $b < c$: (u, g, p, e)
$b \in E$				(u, g, p, e)

Máximo y mínimo: las decisiones del adversario

Usando la tabla podemos ver cuáles son las decisiones que va a tomar el adversario, las cuales corresponden a los peores casos para $\mathcal{A}$:

	$c \in U$	$c \in G$	$c \in P$	$c \in E$
$b \in U$	$(u-2, g+1, p+1, e)$	$b > c$: $(u-1, g, p, e+1)$ $b < c$: $(u-1, g, p+1, e)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p, e+1)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p+1, e)$
$b \in G$		$(u, g-1, p, e+1)$	$b > c$: (u, g, p, e) $b < c$: $(u, g-1, p-1, e+2)$	$b > c$: (u, g, p, e) $b < c$: $(u, g-1, p, e+1)$
$b \in P$			$(u, g, p-1, e+1)$	$b > c$: $(u, g, p-1, e+1)$ $b < c$: (u, g, p, e)
$b \in E$				(u, g, p, e)

Máximo y mínimo: las decisiones del algoritmo

Hay posible decisiones de $\mathcal{A}$ que no cambian el vector (u, g, p, e) dada la estrategia del adversario.

Máximo y mínimo: las decisiones del algoritmo

Hay posible decisiones de $\mathcal{A}$ que no cambian el vector (u, g, p, e) dada la estrategia del adversario.

Estas posibilidades no deben ser consideradas por $\mathcal{A}$:

	$c \in U$	$c \in G$	$c \in P$	$c \in E$
$b \in U$	$(u-2, g+1, p+1, e)$	$b > c$: $(u-1, g, p, e+1)$ $b < c$: $(u-1, g, p+1, e)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p, e+1)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p+1, e)$
$b \in G$		$(u, g-1, p, e+1)$		
$b \in P$			$(u, g, p-1, e+1)$	

Una cota inferior para el cálculo del máximo y el mínimo

Cada vez que un elemento es colocado en E puede dejar de ser comparado.

Una cota inferior para el cálculo del máximo y el mínimo

Cada vez que un elemento es colocado en E puede dejar de ser comparado.

El adversario entonces tiene que evitar que un elemento sea colocado en E, lo cual es representado por las elecciones en rojo:

	$c \in U$	$c \in G$	$c \in P$
$b \in U$	$(u-2, g+1, p+1, e)$	$b > c$: $(u-1, g, p, e+1)$ $b < c$: $(u-1, g, p+1, e)$	$b > c$: $(u-1, g+1, p, e)$ $b < c$: $(u-1, g, p, e+1)$
$b \in G$		$(u, g-1, p, e+1)$	
$b \in P$			$(u, g, p-1, e+1)$

Una cota inferior para el cálculo del máximo y el mínimo

Los elementos que son agregados a E vienen entonces de G o P

Una cota inferior para el cálculo del máximo y el mínimo

Los elementos que son agregados a E vienen entonces de G o P

El algoritmo $\mathcal{A}$ debe entonces tratar de que los conjuntos G y P crezcan lo más rápido posible.

- Puede realizar $\lfloor \frac{n}{2} \rfloor$ comparaciones entre elementos de U, después de lo cual se va a tener que:

$$g = p = \lfloor \frac{n}{2} \rfloor$$

Una cota inferior para el cálculo del máximo y el mínimo

Los elementos que son agregados a E vienen entonces de G o P

El algoritmo $\mathcal{A}$ debe entonces tratar de que los conjuntos G y P crezcan lo más rápido posible.

- Puede realizar $\lfloor \frac{n}{2} \rfloor$ comparaciones entre elementos de U, después de lo cual se va a tener que:

$$g = p = \lfloor \frac{n}{2} \rfloor$$

Si n es impar se debe realizar una comparación adicional para que todos los elementos en algún momento estén en G o P

- Si n es par esta propiedad se tiene después de realizar las comparaciones entre elementos de U

Una cota inferior para el cálculo del máximo y el mínimo

Se debe realizar comparaciones entre elementos de G y entre elementos de P para lograr que $e = n - 2$

- Cada una de estas comparaciones incrementa el valor de e en 1
- Recuerde que $\mathcal{A}$ encontró el máximo y el mínimo cuando llega al vector $(0, 1, 1, n - 2)$

Una cota inferior para el cálculo del máximo y el mínimo

Se debe realizar comparaciones entre elementos de G y entre elementos de P para lograr que $e = n - 2$

- Cada una de estas comparaciones incrementa el valor de e en 1
- Recuerde que $\mathcal{A}$ encontró el máximo y el mínimo cuando llega al vector $(0, 1, 1, n-2)$

Concluimos entonces que $\mathcal{A}$ llega al vector $(0, 1, 1, n-2)$ después de realizar el siguiente número de comparaciones:

n par: $\frac{n}{2} + (n-2) = \frac{3}{2} \cdot n - 2$

n impar: $\lfloor \frac{n}{2} \rfloor + 1 + (n-2) = \lceil \frac{n}{2} \rceil + (n-2) = \lceil \frac{3}{2} \cdot n \rceil - 2$

Una cota inferior para el cálculo del máximo y el mínimo

Conclusión

Un algoritmo debe realizar al menos $\lceil \frac{3}{2} \cdot n \rceil - 2$ comparaciones para encontrar el máximo y el mínimo elemento de una lista con n elementos no repetidos.

Una cota inferior para el cálculo del máximo y el mínimo

Conclusión

Un algoritmo debe realizar al menos $\lceil \frac{3}{2} \cdot n \rceil - 2$ comparaciones para encontrar el máximo y el mínimo elemento de una lista con n elementos no repetidos.

Ejercicio

Encuentre un algoritmo que logre la cota inferior.