Prueba de Oposición

AY2 - Algoritmos

Teodoro Freund

6 de Noviembre del 2018
15:20

Contexto

- Materia: Algoritmos y Estructuras de Datos II

Contexto

- Materia: Algoritmos y Estructuras de Datos II
- Práctica 6: Ordenamiento

Contexto

- Materia: Algoritmos y Estructuras de Datos II
- Práctica 6: Ordenamiento
- Primera Parte
  - Especificación con Tipos Abstractos de Datos (TADs)
  - Demostración de Propiedades (Inducción Estructural)
  - Diseño: REP y ABS
  - Nociones de Complejidad
- Segunda Parte
  - Diseño: elección de estructuras de datos

Contexto

- Materia: Algoritmos y Estructuras de Datos II
- Práctica 6: Ordenamiento
- Primera Parte
  - Especificación con Tipos Abstractos de Datos (TADs)
  - Demostración de Propiedades (Inducción Estructural)
  - Diseño: REP y ABS
  - Nociones de Complejidad
- Segunda Parte
  - Diseño: elección de estructuras de datos
  - **Ordenamiento**

Contexto

- Materia: Algoritmos y Estructuras de Datos II
- Práctica 6: Ordenamiento
- Primera Parte
  - Especificación con Tipos Abstractos de Datos (TADs)
  - Demostración de Propiedades (Inducción Estructural)
  - Diseño: REP y ABS
  - Nociones de Complejidad
- Segunda Parte
  - Diseño: elección de estructuras de datos
  - **Ordenamiento**
  - Divide and Conquer

El Ejercicio 15

Dado un conjunto de naturales, diremos que un agujero es un natural x tal que el conjunto no contiene a x y sí contiene algún elemento menor que x y algún elemento mayor que x.

El Ejercicio 15

Dado un conjunto de naturales, diremos que un agujero es un natural x tal que el conjunto no contiene a x y sí contiene algún elemento menor que x y algún elemento mayor que x.

Diseñar un algoritmo que, dado un arreglo de n naturales, diga si existe algún agujero en el conjunto de los naturales que aparecen en el arreglo. Notar que el arreglo de entrada podría contener elementos repetidos, pero en la vista de conjunto, no es relevante la cantidad de repeticiones.

El Ejercicio 15

Dado un conjunto de naturales, diremos que un agujero es un natural x tal que el conjunto no contiene a x y sí contiene algún elemento menor que x y algún elemento mayor que x.

Diseñar un algoritmo que, dado un arreglo de n naturales, diga si existe algún agujero en el conjunto de los naturales que aparecen en el arreglo. Notar que el arreglo de entrada podría contener elementos repetidos, pero en la vista de conjunto, no es relevante la cantidad de repeticiones.

1. Implementar la función

   $$\text{tieneAgujero?(in A: arreglo(nat))} \rightarrow \text{bool}$$

   que resuelve el problema planteado. La función debe ser de tiempo lineal en la cantidad de elementos de la entrada, es decir, O(n), dónde n = tam(A).

El Ejercicio 15

Dado un conjunto de naturales, diremos que un agujero es un natural x tal que el conjunto no contiene a x y sí contiene algún elemento menor que x y algún elemento mayor que x.

Diseñar un algoritmo que, dado un arreglo de n naturales, diga si existe algún agujero en el conjunto de los naturales que aparecen en el arreglo. Notar que el arreglo de entrada podría contener elementos repetidos, pero en la vista de conjunto, no es relevante la cantidad de repeticiones.

1. Implementar la función

   $$\text{tieneAgujero?(in A: arreglo(nat))} \rightarrow \text{bool}$$

   que resuelve el problema planteado. La función debe ser de tiempo lineal en la cantidad de elementos de la entrada, es decir, O(n), dónde n = tam(A).
2. Calcular y justificar la complejidad del algoritmo propuesto.

¿Por qué?

- Es un problema sencillo si se lo encara por la intuición

¿Por qué?

- Es un problema sencillo si se lo encara por la intuición, a veces es mejor empezar a hacer un ejercicio y ver a donde se llega

¿Por qué?

- Es un problema sencillo si se lo encara por la intuición, a veces es mejor empezar a hacer un ejercicio y ver a donde se llega
- No usa ningún algoritmo complicado de sorting, es un buen ejercicio inicial

¿Por qué?

- Es un problema sencillo si se lo encara por la intuición, a veces es mejor empezar a hacer un ejercicio y ver a donde se llega
- No usa ningún algoritmo complicado de sorting, es un buen ejercicio inicial
- Es uno de los pocos ejercicios que no pide "...un algoritmo de ordenamiento..."

El Ejercicio 15, de nuevo

Dado un conjunto de naturales, diremos que un agujero es un natural x tal que el conjunto no contiene a x y sí contiene algún elemento menor que x y algún elemento mayor que x.

Diseñar un algoritmo que, dado un arreglo de n naturales, diga si existe algún agujero en el conjunto de los naturales que aparecen en el arreglo. Notar que el arreglo de entrada podría contener elementos repetidos, pero en la vista de conjunto, no es relevante la cantidad de repeticiones.

1. Implementar la función

   $$\text{tieneAgujero?(in A: arreglo(nat))} \rightarrow \text{bool}$$

   que resuelve el problema planteado. La función debe ser de tiempo lineal en la cantidad de elementos de la entrada, es decir, O(n), dónde n = tam(A).
2. Calcular y justificar la complejidad del algoritmo propuesto.

tieneAgujero?, primera version

Algoritmo 1 tieneAgujero?

procedure $\textsc{tieneAgujero?}(in\ A : Arreglo(nat)) \rightarrow res : bool$
    $B \leftarrow \textsc{Copiar}(A)$
    $n \leftarrow \textsc{tam}(A)$
    $\textsc{cSortD}(B)$
    $res \leftarrow false$
    for $i \leftarrow 1$ *to* n **do**
        $res \leftarrow res \vee B[i] - B[i-1] > 1$

- Los ciclos-for se mueven sin llegar al último indice, en este caso i se mueve en $[1..n)$

tieneAgujero?, primera version

Algoritmo 2 tieneAgujero?

procedure $\textsc{tieneAgujero?}(in\ A : Arreglo(nat)) \rightarrow res : bool$

  $B \leftarrow \textsc{Copiar}(A)$ $\triangleright\ O(n)$

  $n \leftarrow \textsc{tam}(A)$

  $\textsc{cSortD}(B)$ $\triangleright\ O(n+k)$

  $res \leftarrow false$ $\triangleright\ O(1)$

  for $i \leftarrow 1$ *to* n **do** $\triangleright\ O(n)$

    $res \leftarrow res \vee B[i] - B[i-1] > 1$ $\triangleright\ O(1)$

- Los ciclos-for se mueven sin llegar al último indice, en este caso i se mueve en $[1..n)$
- n es el tamaño de A
- k es el rango de A, $k = max(A) - min(A)$

Counting Sort

Algoritmo 3 Counting Sort Desplazado

```
procedure cSortD(in/out A : Arreglo(nat))
    n ← tam(A)
    if n ≠ 0 then
        m ← min(A)
        for i ← 0 to n do
            A[i] ← A[i] − m
        countingSort(A)
        for i ← 0 to n do
            A[i] ← A[i] + m
```

Counting Sort

Algoritmo 4 Counting Sort Desplazado

procedure $\text{cSortD}(in/out\ A : Arreglo(nat))$
    $n \leftarrow \text{TAM}(A)$ $\triangleright\ O(1)$
    if $n \neq 0$ **then**
        $m \leftarrow \text{MIN}(A)$ $\triangleright\ O(n)$
        for $i \leftarrow 0$ *to* n **do** $\triangleright\ O(n)$
            $A[i] \leftarrow A[i] - m$ $\triangleright\ O(1)$
        $\text{COUNTINGSORT}(A)$ $\triangleright\ O(n+k)$
        for $i \leftarrow 0$ *to* n **do** $\triangleright\ O(n)$
            $A[i] \leftarrow A[i] + m$ $\triangleright\ O(1)$

Observación

Si A' es el arreglo una vez que le reste min(A), max(A')=max(A)-min(A), y entonces countingSort sobre A' tiene complejidad $O(n + max(A) - min(A))$, y la complejidad de cSortD queda $O(n + k)$, donde k es max(A)-min(A).

tieneAgujero?, versión buena

Algoritmo 5 tieneAgujero?

```
procedure TIENEAGUJERO?(in A : Arreglo(nat))→ res : bool
    n ← TAM(A)
    if n ≠ 0 ∧ (MAX(A) − MIN(A)) ≥ n then
        res ← true
    else
        B ← COPIAR(A)
        CSORTD(B)
        res ← false
        for i ← 1 to n do
            res ← res ∨ B[i] − B[i − 1] > 1
```

tieneAgujero?, versión buena

Algoritmo 6 tieneAgujero?

procedure $\textsc{tieneAgujero?}(\mathit{in}\ A : \mathit{Arreglo}(\mathit{nat})) \rightarrow \mathit{res} : \mathit{bool}$

  $n \leftarrow \textsc{tam}(A)$ $\triangleright\ O(1)$

  if $n \neq 0 \wedge (\textsc{max}(A) - \textsc{min}(A)) \geq n$ **then** $\triangleright\ O(n)$

    $\mathit{res} \leftarrow \mathit{true}$ $\triangleright\ O(1)$

  else

    $B \leftarrow \textsc{Copiar}(A)$ $\triangleright\ O(n)$

    $\textsc{cSortD}(B)$ $\triangleright\ O(n+k) = O(n), n > k$

    $\mathit{res} \leftarrow \mathit{false}$ $\triangleright\ O(1)$

    for $i \leftarrow 1$ *to* n **do** $\triangleright\ O(n)$

      $\mathit{res} \leftarrow \mathit{res} \vee B[i] - B[i-1] > 1$ $\triangleright\ O(1)$

Operaciones auxiliares

Algoritmo 7 max

procedure MAX(*in* A : *Arreglo*(*nat*)) $\rightarrow$ *res* : *nat*
- $res \leftarrow A[0]$ $\quad \triangleright\ O(1)$
- $n \leftarrow \text{TAM}(A)$ $\quad \triangleright\ O(1)$
- **for** $i \leftarrow 0$ *to* n **do** $\quad \triangleright\ O(n)$
  - $res \leftarrow \text{MAX}(res, A[i])$ $\quad \triangleright\ O(1)$

Algoritmo 8 min

procedure MIN(*in* A : *Arreglo*(*nat*)) $\rightarrow$ *res* : *nat*
- $res \leftarrow A[0]$ $\quad \triangleright\ O(1)$
- $n \leftarrow \text{TAM}(A)$ $\quad \triangleright\ O(1)$
- **for** $i \leftarrow 0$ *to* n **do** $\quad \triangleright\ O(n)$
  - $res \leftarrow \text{MIN}(res, A[i])$ $\quad \triangleright\ O(1)$