Algoritmos y estructuras de datos

Ordenamiento interno

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

24 de julio de 2024

Citas

Benjamin Franklin

Orden. Haz que todas tus cosas tengan su lugar.

Thomas Mann

El orden y la simplificación son los primeros pasos para dominar un tema. El verdadero enemigo es lo desconocido.

Donald Knuth

Cualquier imprecisión en este índice se puede explicar por el hecho de que ha sido ordenado con la ayuda de una computadora.

Thomas Kuhn

De pronto los fragmentos en mi cabeza se ordenaron de una nueva forma y cayeron en su lugar.

Ordenamiento interno

Definición

Problema abstracto

Dado un arreglo A con n elementos, se desea reacomodar sus elementos de modo que $A_0 \leq A_1 \leq \cdots \leq A_{n-1}$.

Problema concreto

Dado un arreglo `int a[MAX]` con `int n` elementos en las posiciones `a[0], ..., a[n-1]` se desea reacomodar sus elementos de modo que `a[0] <= a[1] <= ... <= a[n-1]`.

Ordenamiento interno

Un problema, mil soluciones

El problema de ordenamiento interno es uno de los más estudiados en la computación y para el que se conocen muchos algoritmos. Algunos son:

- Ordenamiento por cuenta (*Counting sort*).
- Ordenamiento de burbuja (*Bubble sort*).
- Ordenamiento por inserción (*Insertion sort*).
- Ordenamiento por mezcla (*Merge sort*).
- Ordenamiento por selección (*Selection sort*).
- Ordenamiento por partición (*Quicksort*).
- Ordenamiento por árbol (*Tree sort*).
- Ordenamiento por montículo (*Heapsort*).

¿Cómo ordenar?

Dado que la idea de orden está fuertemente ligada a la idea de comparación, uno puede imaginar que todos los algoritmos de ordenamiento hacen comparaciones entre datos.

¿Cómo ordenar?

Dado que la idea de orden está fuertemente ligada a la idea de comparación, uno puede imaginar que todos los algoritmos de ordenamiento hacen comparaciones entre datos.

Ordenamiento sin comparaciones

Sin embargo, hay casos especiales (incluso muy importantes) en los que se puede ordenar sin llevar a cabo comparaciones de datos:

1. Si hay pocos valores distintos, en particular, si hay n valores enteros del 0 al $k-1$, con k pequeña.
2. Si los datos son cadenas de n caracteres, donde hay k caracteres distintos (para **char** se tendría $k = 256$).

Ordenamiento por cuenta

Idea: Cuenta cuántos datos de cada valor hay

Ejemplo

Si tenemos un arreglo con $n = 16$ datos del 0 al 9 ($k = 10$):

$$3 \quad 1 \quad 4 \quad 1 \quad 5 \quad 9 \quad 2 \quad 6 \quad 5 \quad 3 \quad 5 \quad 8 \quad 9 \quad 7 \quad 9 \quad 3$$

Ordenamiento por cuenta

Idea: Cuenta cuántos datos de cada valor hay

Ejemplo

Si tenemos un arreglo con $n = 16$ datos del 0 al 9 ($k = 10$):

$$3 \quad 1 \quad 4 \quad 1 \quad 5 \quad 9 \quad 2 \quad 6 \quad 5 \quad 3 \quad 5 \quad 8 \quad 9 \quad 7 \quad 9 \quad 3$$

entonces tenemos cero 0, dos 1, un 2, ..., tres 9 que podemos representar en otro arreglo:

0	1	2	3	4	5	6	7	8	9
0	2	1	3	1	3	1	1	1	3

Ordenamiento por cuenta

Idea: Cuenta cuántos datos de cada valor hay

Ejemplo

Si tenemos un arreglo con $n = 16$ datos del 0 al 9 ($k = 10$):

$$3 \quad 1 \quad 4 \quad 1 \quad 5 \quad 9 \quad 2 \quad 6 \quad 5 \quad 3 \quad 5 \quad 8 \quad 9 \quad 7 \quad 9 \quad 3$$

entonces tenemos cero 0, dos 1, un 2, ..., tres 9 que podemos representar en otro arreglo:

0	1	2	3	4	5	6	7	8	9
0	2	1	3	1	3	1	1	1	3

de donde podemos recuperar el arreglo ordenado:

$$1 \quad 1 \quad 2 \quad 3 \quad 3 \quad 3 \quad 4 \quad 5 \quad 5 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 9 \quad 9$$

Ordenamiento por cuenta

Implementación

Los n datos a[0],...,a[n-1] tienen valores en 0,...,k-1 con k constante.

```
void cuenta(int n, int a[]) {
  int i, j, r;
  int c[k] = {0};                // inicializar en cero
  for (i = 0; i < n; i++)        // contar cada aparicion
    c[a[i]]++;
  for (i = j = 0; j < k; j++) // para cada 0 <= j < k
    for (r = 0; r < c[j]; r++, i++)
      a[i] = j;                  // poner c[j] copias de j
}
```

Ordenamiento por cuenta

Número de pasos y cantidad de memoria

Número de pasos

Se inicializa el arreglo c (k pasos) y se visita dos veces el arreglo a, una vez para leerlo y otra vez para escribirlo ($2n$ pasos). En total son $2n + k$ pasos, muy rápido si k es pequeña.

Memoria usada

Se usa un arreglo adicional c con k entradas. Esto es muy poco si k es pequeña.

¿Cuándo usarlo?

Sólo si k is pequeña. Por el contrario, si k es muy grande, el arreglo c podría ser prohibitivamente grande o su inicialización podría tomar demasiado tiempo.

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9
1	1	2	3	3	4	5	5	5	6	3	**7**	8	9	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9
1	1	2	3	3	4	5	5	5	6	3	**7**	8	9	9	9
1	1	2	3	3	4	5	5	5	3	**6**	7	8	9	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9
1	1	2	3	3	4	5	5	5	6	3	**7**	8	9	9	9
1	1	2	3	3	4	5	5	5	3	**6**	7	8	9	9	9
1	1	2	3	3	4	5	5	3	**5**	6	7	8	9	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9
1	1	2	3	3	4	5	5	5	6	3	**7**	8	9	9	9
1	1	2	3	3	4	5	5	5	3	**6**	7	8	9	9	9
1	1	2	3	3	4	5	5	3	**5**	6	7	8	9	9	9
1	1	2	3	3	4	5	3	**5**	5	6	7	8	9	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9
1	1	2	3	3	4	5	5	5	6	3	**7**	8	9	9	9
1	1	2	3	3	4	5	5	5	3	**6**	7	8	9	9	9
1	1	2	3	3	4	5	5	3	**5**	6	7	8	9	9	9
1	1	2	3	3	4	5	3	**5**	5	6	7	8	9	9	9
1	1	2	3	3	4	3	**5**	5	5	6	7	8	9	9	9

Ordenamiento de burbuja

Idea: Ordena cada pareja de elementos consecutivos

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
1	3	1	4	5	2	6	5	3	5	8	9	7	9	3	**9**
1	1	3	4	2	5	5	3	5	6	8	7	9	3	**9**	9
1	1	3	2	4	5	3	5	5	6	7	8	3	**9**	9	9
1	1	2	3	4	3	5	5	5	6	7	3	**8**	9	9	9
1	1	2	3	3	4	5	5	5	6	3	**7**	8	9	9	9
1	1	2	3	3	4	5	5	5	3	**6**	7	8	9	9	9
1	1	2	3	3	4	5	5	3	**5**	6	7	8	9	9	9
1	1	2	3	3	4	5	3	**5**	5	6	7	8	9	9	9
1	1	2	3	3	4	3	**5**	5	5	6	7	8	9	9	9
1	1	2	3	3	3	**4**	5	5	5	6	7	8	9	9	9

Ordenamiento de burbuja

Implementación del intercambio

En general, el intercambio de dos valores a y b requiere de una variable temporal t:

```
int a, b, t;
t = a;
a = b;
b = t;
```

Usaremos intercambios en arreglos con tanta frecuencia que declararemos una función:

```
void intercambio(int a[], int i, int j) {
  int t = a[i];
  a[i] = a[j];
  a[j] = t;
}
```

Ordenamiento de burbuja

Implementación de una fase de comparaciones

En un arreglo a[0], ..., a[j] queremos comparar las parejas de elementos consecutivos e intercambiarlos si están en desorden. Regresa falso en ese caso.

```
int fase(int j, int a[]) {
  int bien = 1;                // bien al principio
  for (int i = 0; i < j; i++)
    if (a[i] > a[i+1]) {       // si estan en desorden
      intercambio(a, i, i+1)   // intercambialos
      bien = 0;                // no estaban bien
    }
  return bien;
}
```

Ordenamiento de burbuja

Implementación iterativa

Hacemos las fases que terminan en a[n-1], . . . ,a[1] en ese orden.

```
void burbuja(int n, int a[]) {
  for (int j = n-1; j > 0; j--)
    if (fase(j, a) == 1) // no hubo intercambios
      return;              // podemos terminar
}
```

Ordenamiento de burbuja

Implementación iterativa

Hacemos las fases que terminan en `a[n-1]`, . . . , `a[1]` en ese orden.

```
void burbuja(int n, int a[]) {
  for (int j = n-1; j > 0; j--)
    if (fase(j, a) == 1) // no hubo intercambios
      return;              // podemos terminar
}
```

Cantidad de comparaciones

La llamada a `fase(j, a)` hace a lo mucho `j` comparaciones. En total se hacen a lo mucho $1 + 2 + \cdots + (n-1) = \frac{1}{2}n(n-1) \approx \frac{1}{2}n^2$ comparaciones.

Ordenamiento de burbuja

Implementación recursiva

Se hace primero la fase que termina en `a[n-1]` y luego, de ser necesario, se ordenan recursivamente con ordenamiento de burbuja los primeros $n - 1$ elementos.

```
void burbuja(int n, int a[]) {
  // si hay elementos y hubo intercambios
  if (n > 1 && fase(n-1, a) == 0)
    burbuja(n-1, a); // continuar recursivamente
}
```

Ordenamiento de burbuja

Implementación recursiva

Se hace primero la fase que termina en `a[n-1]` y luego, de ser necesario, se ordenan recursivamente con ordenamiento de burbuja los primeros $n-1$ elementos.

```
void burbuja(int n, int a[]) {
  // si hay elementos y hubo intercambios
  if (n > 1 && fase(n-1, a) == 0)
    burbuja(n-1, a); // continuar recursivamente
}
```

Cantidad de comparaciones

Si `burbuja(n, a)` hace $T(n)$ comparaciones, entonces $T(n) \leq T(n-1) + (n-1)$ si $n > 1$ y $T(1) = 0$. En total son $\leq (n-1) + \cdots + 1 = \frac{1}{2}n(n-1) \approx \frac{1}{2}n^2$ comparaciones.

Propiedades de los algoritmos de ordenamiento

Relacionadas a los tipos de datos

Nuestros algoritmos de ordenamiento ordenarán arreglos `int a[]` de forma creciente.

1. Si queremos ordenar de forma decreciente bastaría invertir las comparaciones.
2. Si queremos ordenar arreglos de otro tipo (para el que esté definido <) bastaría cambiar el tipo de las variables correspondientes (por ejemplo a `float` o a `double`).
3. Se pueden implementar algoritmos de ordenamiento genéricos para cualquier tipo de datos para el que podamos definir un orden total. La biblioteca estándar de C contiene un algoritmo de ordenamiento genérico que requiere saber, entre otras cosas, cómo se lleva a cabo la comparación de dos datos del tipo de interés.

Propiedades de los algoritmos de ordenamiento

Estabilidad

Si un arreglo tiene datos repetidos, un algoritmo de ordenamiento los puede dejar en el orden que ya estaban. Un algoritmo de ordenamiento con esta propiedad se dice estable.

Ordenamiento por cuenta

No es estable, pues pierde toda la información del orden original.

Ordenamiento de burbuja

Sí es estable, pues cuando compara dos elementos consecutivos que son iguales no los intercambia de lugar, preservando su orden original.

Importancia

Si cada dato consiste de dos partes (x, y), entonces ordenar por x y luego ordenar establemente por y conserva el orden de x entre los datos con la misma y.

Ordenamiento de burbuja

Comparación con ordenamiento por cuenta

Uso de memoria y cantidad de operaciones

Propiedad	Burbuja	Cuenta
Memoria	n	$n + k$
Operaciones	$\frac{1}{2}n^2$	$2n + k$

Tiempo de ejecución (1 ns por comparación)

Considerando una k muy pequeña.

n	Burbuja	Cuenta
10^3	0.5 ms	2 μs
10^4	50 ms	20 μs
10^5	5 s	0.2 ms
10^6	500 s	2 ms

Ordenamiento de burbuja

Ejercicios

1. Implementa `burbuja` de modo que compare las parejas de elementos consecutivos de atrás para adelante.
2. ¿Es esta versión de `burbuja` estable?
3. Implementa `burbuja` de modo que haga fases alternadamente de adelante para atrás y de atrás para adelante (*Cocktail sort*).
4. Sea $c > 1$ y considera una variante de ordenamiento de burbuja donde en la fase i se comparan elementos a distancia $g_i = \lceil \frac{n}{c^i} \rceil$. A partir de que $g_i = 1$ se continúa como en el ordenamiento de burbuja. A esta variante se le conoce como *Comb sort*.