Bài 8.9: Thuật toán radix sort

- ✓ Tổng quan về thuật toán
- ✓ Thuật toán sắp xếp radix sort
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Tổng quan

- Thuật toán radix sort là thuật toán sắp xếp không dựa trên việc so sánh các phần tử mảng.
- Nó tránh việc so sánh bằng cách tạo và phân phối các phần tử vào các nhóm theo cơ số của chúng.
- Thuật toán radix sort dựa trên thuật toán counting sort.
- Radix sort có thể được triển khai theo hướng tiếp cận MSD(chữ số quan trọng nhất) hoặc LSD(chữ số ít quan trọng nhất).
- Hướng tiếp cận LSD thường tuân thủ quy tắc: các khóa ngắn đứng trước khóa dài. Các khóa có cùng độ dài sẽ được sắp xếp theo thứ tự từ điển.

Tổng quan

- Có thể áp dụng cho các tập dữ liệu có thể sắp xếp theo thứ tự từ điển, các số nguyên, các từ hoặc email…
- Độ phức tạp của thuật toán này là O(nw) với n là lượng các khóa trong mảng và w là số chữ số cấu thành nên khóa.

Thuật toán sắp xếp đếm

➢ Mã giả của thuật toán counting sort:

```
// thuật toán counting sort
void countingSort(input[], int exp) { // input: mảng input
    n = input.length; // lấy số phần tử mảng
    k = 9; // số chữ số tối đa của giá trị kiểu int(~2.1 tỉ)
    count[] = new [k + 1]; // tạo mảng count có k + 1 phần tử
    output = new [n]; // tạo mảng output có n phần tử

    for(i = 0; i < n; i++) { // đếm số lần xuất hiện của
        j = (input[i] / exp) % 10; // phần tử tại vị trí i trong mảng input
        count[j] += 1; // tăng biến đếm tại vị trí j lên 1
    }

    for(i = 1; i <= k; i++) { // tính tổng tiền tố cho p.tử vị trí i
        count[i] += count[i-1];
    }
    // đưa các phần tử vào đúng vị trí của nó
    for(i = n - 1; i >= 0; i--) {
        j = (input[i] / exp) % 10; // lấy phần tử tại vị trí i của mảng input
        count[j] -= 1; // giảm biến đếm của nó đi 1
        output[count[j]] = input[i]; // gán phần tử vào vị trí
    }
    // sao chép các phần tử trong mảng vào mảng gốc
    for (i = 0; i < n; i++)
    {
        input[i] = output[i];
    }
}
```

Thuật toán sắp xếp đếm

➢ Mã giả của thuật toán counting sort:

```
// thuật toán counting sort
void countingSort(input[], int exp) { // input: mảng input
    n = input.length; // lấy số phần tử mảng
    k = 9; // số chữ số tối đa của giá trị kiểu int(~2.1 tỉ)
    count[] = new [k + 1]; // tạo mảng count có k + 1 phần tử
    output = new [n]; // tạo mảng output có n phần tử

    for(i = 0; i < n; i++) { // đếm số lần xuất hiện của
        j = (input[i] / exp) % 10; // phần tử tại vị trí i trong mảng input
        count[j] += 1; // tăng biến đếm tại vị trí j lên 1
    }

    for(i = 1; i <= k; i++) { // tính tổng tiền tố cho p.tử vị trí i
        count[i] += count[i-1];
    }
    // đưa các phần tử vào đúng vị trí của nó
    for(i = n - 1; i >= 0; i--) {
        j = (input[i] / exp) % 10; // lấy phần tử tại vị trí i của mảng input
        count[j] -= 1; // giảm biến đếm của nó đi 1
        output[count[j]] = input[i]; // gán phần tử vào vị trí
    }
    // sao chép các phần tử trong mảng vào mảng gốc
    for (i = 0; i < n; i++)
    {
        input[i] = output[i];
    }
}
```

Thuật toán sắp xếp đếm

- Mã giả của thuật toán radix sort:

```
// thuật toán radix sort
void radixSort(arr[]) {
    int max = findMax(arr);
    for (i = 1; max / i > 0; i *= 10)
    {
        countingSort(arr, i);
    }
}
// tìm giá trị lớn nhất trong mảng
int findMax(arr[]) {
    max = arr[0];
    for (i = 1; i < arr.length; i++)
    {
        arr[i] > max;
    }
    return max;
}
```

Thuật toán sắp xếp đếm

- Mã thật của thuật toán radix sort:

```
private static int findMax(int[] arr) {
    int maxValue = arr[0];
    for (int e : arr) {
        if (e > maxValue) {
            maxValue = e;
        }
    }
    return maxValue;
}

public static void radixSort(int[] arr) {
    int max = findMax(arr);
    for (int i = 1; max / i > 0; i *= 10) {
        countingSort(arr, i);
    }
}
```

Nội dung tiếp theo

Sắp xếp mảng với lớp Arrays