

Bài 1.9: Thuật toán sinh hoán vị chính tắc

- ✓ Mô tả bài toán
- ✓ Thuật toán tổng quát
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Mô tả thuật toán

- Thuật toán sinh hoán vị n phần tử áp dụng thuật toán sinh cấu hình kế tiếp để giải quyết vấn đề.
- Bài toán: cho số nguyên dương n . Viết chương trình sinh toàn bộ các cấu hình của các hoán vị từ 1 đến n theo thứ tự từ điển.
- Cấu hình $H = \{h_1, h_2, \dots, h_n\}$ với $h_i = \{1, 2, \dots, n\}$ và $i = \{1, n\}$ là xâu nhị phân độ dài n .
- Mỗi cấu hình của một hoán vị là bộ các giá trị gồm n phần tử có thứ tự khác nhau.
- Cấu hình X , gọi là cấu hình liền trước Y tồn tại x_i và y_i sao cho $x_i < y_i$ và mọi x_k, y_k khác thỏa $x_k == y_k$. Ví dụ 123 đứng trước 132.
- Với một số nguyên dương n cho trước, ta có $n!$ cấu hình thỏa mãn.

Thuật toán tổng quát

➤ Sau đây là mã giả thuật toán sinh hoán vị kế tiếp:

```
// thuật toán sinh hoán vị kế tiếp
bool nextPermutation(arr[]) { // arr: mảng đầu vào
    i = arr.length - 2; // xuất phát từ phần tử trước phần tử cuối
    while(i >= 0 && arr[i] > arr[i+1]) { // tìm i sao cho arr[i] < arr[i+1]
        i--;
    }
    if(i >= 0) { // nếu i chưa vượt quá phần tử trái cùng
        k = arr.length - 1; // xuất phát từ phần tử phải cùng
        while(arr[i] > arr[k]) { // tìm k sao cho arr[k] > arr[i]
            k--;
        }
        // đổi chỗ hai phần tử tại vị trí i và k
        tmp = arr[i];
        arr[i] = arr[k];
        arr[k] = tmp;
        r = i + 1; // gán r = i + 1
        s = arr.length - 1; // gán s = vị trí cuối mảng
        while(r <= s) { // lật ngược đoạn từ j + 1 đến n
            t = arr[r];
            arr[r] = arr[s];
            arr[s] = t;
            r++;
            s--;
        }
        return false; // trả về thông báo cấu hình chưa phải cuối cùng
    } else {
        return true; // trả về thông báo cấu hình cuối cùng
    }
}
```

Thuật toán tổng quát

➤ Sau đây là mã giả thuật toán sinh hoán vị chính tắc:

```
// thuật toán sinh hoán vị chính tắc
void generatePermutation(arr) { // arr: mảng chứa các phần tử đầu vào
    isFinalConfig = false; // khởi tạo biến đánh dấu cấu hình cuối cùng
    while(!isFinalConfig) { // hiển thị tất cả các cấu hình hoán vị
        ouput(arr); // gọi hiển thị
        isFinalConfig = nextPermutation(arr); // sinh cấu hình kế tiếp
    }
}
```

Thuật toán tổng quát

➤ Sau đây là mã thật thuật toán sinh hoán vị kế tiếp:

```
// thuật toán sinh hoán vị kế tiếp
public static boolean nextPermutation(int[] arr) {
    int i = arr.length - 2; // xuất phát từ phần tử trước phần tử cuối
    while (i >= 0 && arr[i] > arr[i + 1]) { // tìm i sao cho arr[i] < arr[i+1]
        i--;
    }
    if (i >= 0) { // nếu i chưa vượt quá phần tử trái cùng
        int k = arr.length - 1; // xuất phát từ phần tử phải cùng
        while (arr[i] > arr[k]) { // tìm k sao cho arr[k] > arr[i]
            k--;
        }
        // đổi chỗ hai phần tử tại vị trí i và k
        int tmp = arr[i];
        arr[i] = arr[k];
        arr[k] = tmp;
        int r = i + 1; // gán r = i + 1
        int s = arr.length - 1; // gán s = vị trí cuối mảng
        while (r < s) { // lật ngược đoạn từ j + 1 đến n
            int t = arr[r];
            arr[r] = arr[s];
            arr[s] = t;
            r++;
            s--;
        }
        return false; // trả về thông báo cấu hình chưa phải cuối cùng
    } else {
        return true; // trả về thông báo cấu hình cuối cùng
    }
}
```

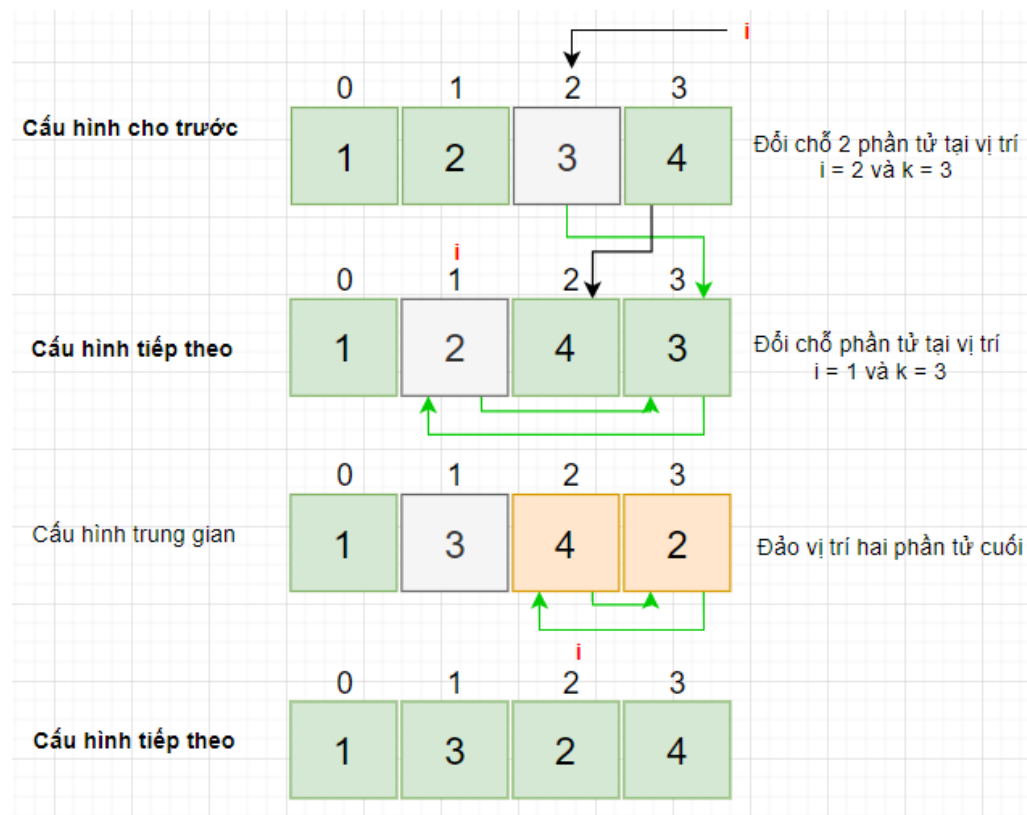
Thuật toán tổng quát

➤ Sau đây là mã thật thuật toán sinh hoán chính tắc:

```
// thuật toán sinh hoán vị chính tắc
public static void generate(int[] arr) {
    boolean isFinalConfig = false; // khởi tạo biến đánh dấu cấu hình cuối cùng
    while (!isFinalConfig) { // hiển thị tất cả các cấu hình hoán vị
        output(arr); // gọi hiển thị
        isFinalConfig = nextPermutation(arr); // sinh cấu hình kế tiếp
    }
}
```


Ví dụ minh họa

➤ Ví dụ cho cấu hình 1234, hãy tìm 2 cấu hình kế tiếp:



Nội dung tiếp theo

Thuật toán sinh tổ hợp chập k của n