Mục đích

Luyện tập tính độ phức tạp của thuật toán.

Mô tả bài toán

Viết một chương trình cho phép xác định ký tự xuất hiện nhiều nhất trong một chuỗi.

Chương trình cho phép nhập vào một chuỗi và hiển thị ký tự có số lần xuất hiện nhiều nhất trong chuỗi đó.

Tính độ phức tạp của thuật toán vừa triển khai.

Hướng dẫn thực hiện

**Bước 1:** Tạo lớp AlgorithmComplexityTest, viết hàm main cho phép nhập vào một chuỗi ký tự

**public class** AlgorithmComplexityTest {  
  
 **public static void** main(String[] args) {  
 Scanner scanner = **new** Scanner(System.***in***);  
 System.***out***.println(**"Enter a string:"**);  
 String inputString = scanner.nextLine();

*//codes below here*

}

**Bước 2:**Tính tần suất xuất hiện của các ký tự trong chuỗi vừa nhập

Ta biết rằng mỗi ký tự tương ứng với 1 số nguyên trong bảng mã ASCII gồm 255 ký tự. Vì thế, ta sẽ tạo ra một mảng số nguyên có 255 phần tử, mỗi vị trí trong mảng đại diện cho một ký tự trong bảng ASCII, giá trị tại đó biểu diễn tần suất xuất hiện của ký tự. Bạn đầu đặt tất cả các phần tử bằng không. Sử dụng vòng lặp duyệt qua các kí tự của chuỗi vừa nhập và tính tần suất xuất hiện của các ký tự

**int**[] frequentChar = **new int**[255]; {1}  
**for** (**int** i = 0; i < inputString.length(); i++) { {2}

// Tìm ví trí tương ứng của ký tự trong bảng ASCII  
 **int** ascii = (**int**) inputString.charAt(i); {3}

// Tăng giá trị tần suất  
 frequentChar[ascii] += 1; {4}  
}

**Bước 3:** Tìm ký tự có tần suất xuất hiện nhiều nhất

**int** max = 0; {5}  
**char** character = (**char**) 255; *//empty character* {6}**for** (**int** j = 0; j < 255; j++) { {7}  
 **if** (frequentChar[j] > max) { {8}  
 max = frequentChar[j]; {9}  
 character = (**char**) j; {10}  
 }  
}  
System.***out***.println(**"The most appearing letter is '"** + character + **"' with a frequency of "** + max + **" times"**);

**Bước 4:** Chạy chương trình. Quan sát kết quả trả về.

**Bước 5:** Tính độ phức tạp của thuật toán vừa triển khai

Ta có:

* Thời gian thực hiện lệnh {1}là O(1)  
  Hai lệnh gán {3} và {4} đều tốn O(1) thời gian. Vòng lặp {2} thực hiện (n) lần, mỗi lần O(1) do đó vòng lặp {2} tốn O((n).1) = O(n) với n là độ dài chuỗi ký tự.
* Thời gian thực hiện lệnh {5} và {6} đều là O(1)
* Hai lệnh gán {9} và {10} đều tốn O(1) thời gian, việc so sánh frequentChar[j] > max cũng tốn O(1) thời gian, do đó lệnh {8} tốn O(1) thời gian. Vòng lặp {7} thực hiện (255) lần, mỗi lần O(1) do đó vòng lặp {7} tốn O((255).1) = O(255).

*Chú ý: Trong trường hợp vòng lặp không xác định được số lần lặp thì chúng ta phải lấy số lần lặp trong trường hợp xấu nhất.*

Độ phức tạp của thuật toán cũng chính là thời gian thực hiện chuỗi lệnh {1}, {2}, {5}, {6}, {7}, và được tính bằng thời gian thi hành lệnh lâu nhất trong chuỗi lệnh.

Ta sẽ có 2 trường hợp  
TH1: n <= 255

Khi đó độ phức tạp của thuật toán sẽ là thời gian thực hiện lệnh {7} và bằng O(255)

TH2: n> 255

Khi đó độ phức tạp của thuật toán sẽ là thời gian thực hiện lệnh {2} và bằng O(n)