Bài 10
Giải quyết vấn đề và thuật toán

Module: BOOTCAMP PREPARATION

Kiểm tra bài trước

Hỏi và trao đổi về các khó khăn gặp phải trong bài “Chuỗi”

Tóm tắt lại các phần đã học từ bài “Chuỗi”

Mục tiêu

- Mô tả bài toán tìm kiếm
- Trình bày được ý tưởng và các bước thực hiện của thuật toán tìm kiếm tuyến tính
- Triển khai được thuật toán tìm kiếm tuyến tính
- Debug được ứng dụng JavaScript
- Thực hiện được các thao tác với mảng

Thảo luận

Thuật toán tìm kiếm

Các bài toán tìm kiếm trong mảng

- Tìm chỉ số của một phần tử trong mảng
- Kiểm tra sự tồn tại của một giá trị trong mảng
- Đếm số lần xuất hiện của một giá trị trong mảng
- Tìm giá trị lớn nhất của mảng
- Tìm giá trị nhỏ nhất của mảng

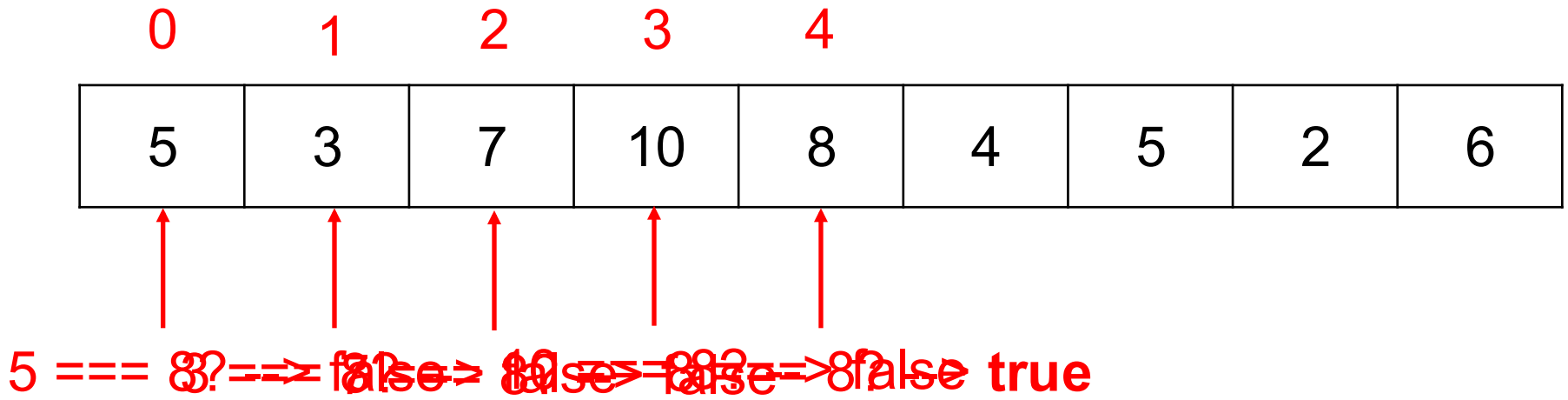
Các giải thuật tìm kiếm

Tìm kiếm tuyến tính

Tìm kiếm nhị phân

Tìm kiếm tuyến tính (LinearSearch)

- Giải thuật:
  - Bắt đầu từ phần tử đầu tiên, so sánh với giá trị muốn tìm, nếu bằng giá trị muốn tìm thì trả về vị trí hiện tại còn nếu không bằng thì chuyển sang phần tử kế tiếp
  - Nếu đến phần tử cuối cùng mà không tìm thấy giá trị nào bằng thì nghĩa là không tìm thấy
- Ví dụ: Tìm phần tử có giá trị là 8 ở trong mảng



Tìm kiếm tuyến tính: Mã giả

Đầu vào: mảng a có N phần tử và giá trị x

Đầu ra: Trả về vị trí nếu tìm thấy, ngược lại trả về -1

- Bước 1: i = 0 //bắt đầu từ phần tử đầu tiên của dãy
- Bước 2: So sánh a[i] với x, có 2 khả năng
  - a[i] = x: Tìm thấy. Trả về i và dừng lại
  - a[i] != x: Sang bước 3
- Bước 3:
  - i = i + 1 //xét tiếp phần tử kế trong mảng
  - Nếu i >= N: Hết mảng, trả về -1. Ngược lại: Lặp lại Bước 2.

Cài đặt hàm tìm kiếm tuyến tính

```
function linearSearch(arr, value){
    for(var i = 0; i < arr.length; i++){
        if(arr[i] === value) {
            return i;
        }
    }
    return - 1;
}
```

```
var numbers = [5, 3, 7, 10, 8, 4, 5, 2, 6];
var result = linearSearch(numbers, 8);
```

Demo

Triển khai thuật toán tìm kiếm

Thảo luận

Debug ứng dụng JavaScript

Debug

- Debug (dò lỗi) là quá trình tìm kiếm và sửa chữa các lỗi trong một chương trình
- Các loại lỗi thường gặp:
  - Lỗi cú pháp (syntax error) dễ phát hiện bởi vì compiler đưa ra thông báo
  - Lỗi thực thi (runtime error) cũng dễ phát hiện bởi vì interpreter cũng hiển thị lỗi trên màn hình
  - Lỗi logic thường khó phát hiện hơn
- Các lỗi logic còn được gọi là bug
- Thông thường, cần kết hợp nhiều cách để tìm được bug

Một số phương pháp debug

- Đọc mã nguồn (hand-trace)
- Chèn các lệnh in ra các giá trị trong từng đoạn của chương trình để kiểm tra các giá trị và việc thực thi các câu lệnh
- Sử dụng debugger: một chương trình cho phép quan sát quá trình thực thi của một ứng dụng
- Các IDE thông thường cũng tích hợp sẵn debugger
- Các công cụ debug của trình duyệt

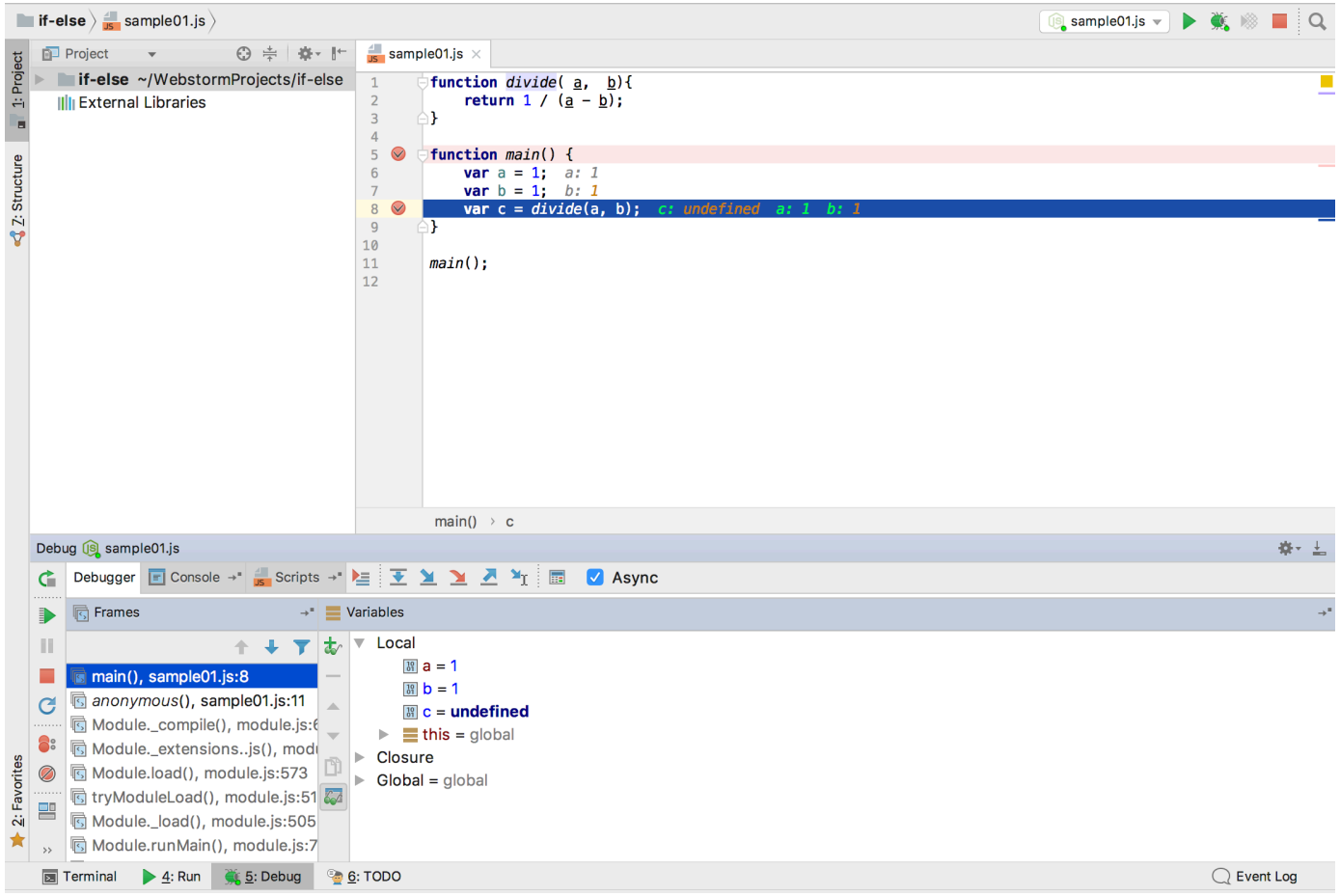
Các tính năng của debugger

- Thực thi riêng lẻ từng câu lệnh tại một thời điểm
- Theo dõi hoặc bỏ qua luồng thực thi vào sâu bên trong một phương thức
- Đặt các breakpoint: các điểm mà chương trình sẽ dừng lại để lập trình viên quan sát trạng thái hiện tại của chương trình
- Hiển thị giá trị của các biến tại từng thời điểm
- Hiển thị ngăn xếp các phương thức được gọi
- Thay đổi giá trị của biến tại thời điểm thực thi (chỉ một số debugger hỗ trợ tính năng này)

Debug với WebStorm

- Khởi chạy chế độ debug bằng phím (^D) hoặc nút 
- Đặt breakpoint
- Xem thông tin chi tiết tại điểm breakpoint
- Step into (quan sát sâu vào bên trong một phương thức được gọi)
- Step over (không quan sát bên trong một phương thức được gọi)

Debug với WebStorm



Demo

Debug ứng dụng

Thảo luận

Các lỗi thường gặp

Các lỗi thường gặp #1: Chỉ số bắt đầu từ 1

- Việc nhầm lẫn chỉ số của mảng bắt đầu từ 1 sẽ dẫn đến xử lý sai dữ liệu
- Chẳng hạn, đoạn mã tính tổng các số trong mảng không hoạt động đúng:

```
var total = 0;
for (var i = 1; i < myList.length; i++){
    total += myList[i];
}

console.log("Total is: " + total);
```

Các lỗi thường gặp #2: Nhầm chỉ số cuối cùng

- Đôi khi xảy ra tình huống nhầm lẫn chỉ số của phần tử cuối cùng là n thay vì n – 1
- Việc này dẫn đến truy xuất không đúng dữ liệu, hoặc thậm chí gây lỗi
- Ví dụ, đoạn mã sau sẽ tung ra lỗi, vì truy xuất đến phần tử nằm ở ngoài phạm vi của mảng:

```
var sumOfLastTwoElements = myList[myList.length - 1] + myList[myList.length];
```

Thảo luận

Sử dụng vòng lặp for để duyệt mảng

Thực hiện tác thao tác thông dụng với mảng

Sử dụng vòng lặp for-each để duyệt mảng

#1. Sử dụng vòng lặp for

- Sử dụng vòng lặp for, chạy từ *0* đến *length - 1* để duyệt qua các phần tử của mảng

```
for (var i = 0; i < myList.length; i++){
    //Thao tác với phần tử myList[i]
}
```

#2. Khởi tạo giá trị ngẫu nhiên cho mảng

- Đoạn mã sau sẽ gán các giá trị double ngẫu nhiên nằm trong khoảng từ 0 đến 100 cho các phần tử của mảng:

```
var myList = new Array(10);

for (var i = 0; i < myList.length; i++) {
    myList[i] = Math.floor((Math.random() * 100) + 1);
}
```

Lưu ý: Phương thức Math.random() trả về một giá trị ngẫu nhiên nằm trong khoảng [0 – 1]

#3. Hiển thị các phần tử của mảng

```
for (var i = 0; i < myList.length; i++) {
    console.log(”myList[" + i + "] =" + myList[i]);
}
```

```
myList[0] =61
myList[1] =40
myList[2] =26
myList[3] =60
myList[4] =85
myList[5] =25
myList[6] =95
myList[7] =61
myList[8] =59
myList[9] =72
```

#4. Tính tổng các phần tử kiểu số

```
var total = 0;

for (var i = 0; i < myList.length; i++) {
    total += myList[i];
}

console.log("Total is: " + total)
```

#6. Tìm phần tử lớn nhất

```
var max = myList[0];
for (var i = 1; i < myList.length; i++) {
    if (myList[i] > max)
        max = myList[i];
}
```

#7. Tìm vị trí đầu tiên của phần tử lớn nhất

```
var max = myList[0];
var indexOfMax = 0;
for (var i = 1; i < myList.length; i++) {
    if (myList[i] > max) {
        max = myList[i];
        indexOfMax = i;
    }
}
```

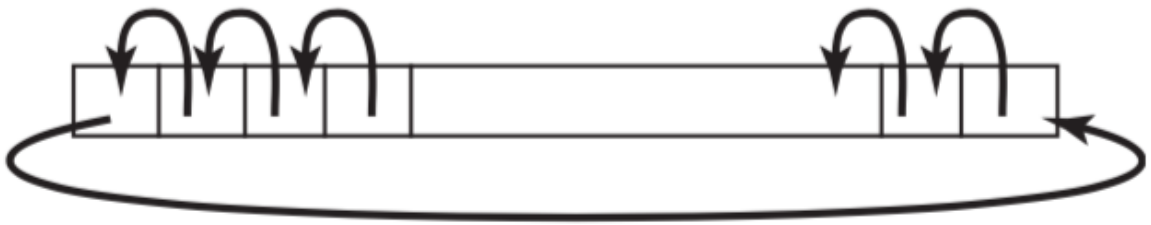
#8. Dịch chuyển các phần tử

- Có thể dịch chuyển một số phần tử sang trái hoặc sang phải
- Ví dụ sau đây dịch chuyển các phần tử sang trái 1 vị trí và đưa phần tử đầu tiên vào thay thế cho phần tử cuối cùng

```
var firstElement = myList[0];

for (var i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}

myList[myList.length - 1] = firstElement;
```



#9. Đơn giản hoá mã nguồn (1)

- Trong một số trường hợp, có thể sử dụng mảng để đơn giản hoá mã nguồn
- Chẳng hạn, trong trường hợp muốn hiển thị tên của các tháng trong năm, tương ứng với giá trị số của tháng đó
- Nếu sử dụng câu lệnh điều kiện bình thường (*if-else*, hoặc *switch-case*) thì mã nguồn có thể là như sau:

```
if (monthNumber == 1)
    console.log ("The month is January");
else if (monthNumber == 2)
    console.log("The month is February");
...
else
    console.log("The month is December");
```

#9. Đơn giản hoá mã nguồn (2)

- Sử dụng mảng trong trường hợp này, chúng ta có thể đơn giản hoá mã nguồn như sau:

```
var months = ["January", "February",..., "December"];

var monthNumber = parseInt(prompt("Enter a month number (1 to 12):"));
alert("The month is " + months[monthNumber - 1]);
```

Tóm tắt bài học

- Thuật toán tìm kiếm
- Các thao tác với mảng
- Debug ứng dụng JavaScript

Hướng dẫn

Hướng dẫn làm bài thực hành và bài tập

Chuẩn bị bài tiếp theo: *Case Study*