Các Giải Thuật Tìm Kiếm

Câu 1: Sinh viên tự thực hiện

Câu 2: Cho một dãy số nguyên có n phần tử ($n \leq 100$) và một số nguyên x. Hãy tìm và xác định xem x có tồn tại trong dãy hay không ? Nếu có xuất ra vị trí xuất hiện đầu tiên của x trong dãy, nếu không xuất ra giá trị -1.

Hướng dẫn:

Trong bài này, chúng ta sẽ thực hiện giải thuật tìm kiếm tuyến tính (tìm kiếm tuần tự) để tìm ra vị trí xuất hiện của x trong dãy.

Gọi foundPosition là biến lưu vị trí xuất hiện đầu tiên của x trong dãy. Ban đầu ta đặt foundPosition = -1. Duyệt các phần tử từ 0 đến n-1 của dãy và so sánh mỗi phần tử như vậy với giá trị x. Nếu phần tử thứ i nào đó trong dãy bằng với x thì ta cập nhật foundPosition = x rồi dừng quá trình tìm kiếm lại. Sau cùng, ta xuất ra foundPosition chính là vị trí xuất hiện đầu tiên của x trong dãy số. Lưu ý: nếu x không tồn tại trong dãy số thì foundPosition sẽ không bao giờ được cập nhật và do đó giá trị của nó vẫn giữ nguyên như thiết lập ban đầu là -1

Chương trình:

```
import java.util.Scanner;

public class Cau2 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nhập vào số phần tử của dãy số:
");
        int n = sc.nextInt();
        System.out.println("Dãy số sinh ngẫu nhiên:");
        int[] a = new int[n];
        for (int i=0; i < n-1; i++) {
            a[i] = (int)(Math.random() * 15.5);
            System.out.println(a[i] + ", ");
        }
        a[n-1] = (int)(Math.random() * 15.5);
```

```
            System.out.println(a[n-1]);
            System.out.print("Nhap giá trị cần tìm: ");
            int x = sc.nextInt();
            sc.close();

            int foundPosition = -1;
            for (int i=0; i<n; i++)
                 if (a[i] == x) {
                      foundPosition = i;
                      break;
                 }
            if (foundPosition != -1) System.out.println("Phần
tử " + x + " xuất hiện tại vị trí thứ " + foundPosition + "
trong dãy!");
            else System.out.println(foundPosition);
       }
}
```

Câu 3: Nhập vào một chuỗi bất kì và một kí tự x. Dùng giải thuật tìm kiếm tuyến tính, hãy xác định xem x có tồn tại trong chuỗi hay không ? Nếu có xuất ra vị trí xuất hiện đầu tiên của x trong chuỗi, nếu không xuất ra giá trị -1

Hướng dẫn: bài này cách làm cũng tương tự như bài 2 nhưng chỉ có điều dữ liệu ở đây của chúng ta là 1 chuỗi và chúng ta phải dùng hàm CharAt() có sẵn của Java để so sánh từng kí tự của chuỗi với kí tự x. Lưu ý, việc so sánh 2 chuỗi hay 2 kí tự với nhau thì phải sử dụng hàm equals() của Java để cho ra kết quả chính xác.

Chương trình:

```
import java.util.Scanner;

public class Cau3 {

     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          System.out.print("Nhập vào một chuỗi kí tự: ");
          String s = sc.nextLine();
          System.out.print("Nhập vào kí tự cần tìm trong
chuỗi: ");
          char x = sc.nextLine().charAt(0);
```

```
            int foundPosition = -1;
            for (int i=0; i<s.length(); i++)
                if (s.charAt(i) == x) {
                    foundPosition = i;
                    break;
                }
            if (foundPosition != -1) System.out.println("Kí tự
" + x + " xuất hiện tại vị trí thứ " + foundPosition + "
trong chuỗi!");
            else System.out.println(foundPosition);
            sc.close();
        }
}
```

Câu 4: Cho một chuỗi A và một chuỗi B (kiểu String). Chuỗi B có độ dài bé hơn hoặc bằng chuỗi A. Hãy viết chương trình để xác định xem chuỗi B có xuất hiện trong chuỗi A hay không ? Nếu có thì hãy xuất ra vị trí xuất hiện đầu tiên của chuỗi B trong chuỗi A. Nếu không thì hãy xuất ra giá trị -1.

Hướng dẫn: để thuận tiện cho việc lập trình, ta nên xây dựng một hàm riêng để kiểm tra xem chuỗi B có nằm trong chuỗi A hay không.

Giả sử chuỗi B nằm trong chuỗi A, thì gọi k là vị trí bắt đầu của B trong A, ta luôn có: k + B.length() ≤A.length() ⇔ k ≤ A.length() – B.length(). Do đó, để kiểm tra xem chuỗi B có nằm trong chuỗi A hay không, ta chỉ cần duyệt qua các kí tự từ 0 đến A.length()-B.length() của chuỗi A và kiểm tra là có thể biết được.

Cách kiểm tra:

Ban đầu, dùng một vòng for i=0 đến A.length()-B.length() để duyệt qua từng kí tự của A. Với mỗi kí tự được xét, ta lồng một vòng for j=0 đến B.length()-1 để duyệt qua từng kí tự của B và kiểm tra nếu tất cả A.charAt(i+j) == B.charAt(j) thì B nằm trong A, và ngược lại. Kích thước của vòng lặp ban đầu từ 0 đến A.length()-B.length() đảm bảo cho việc truy cập kí tự A.charAt(i+j) không bị nằm ngoài kích thước của chuỗi A.

Chương trình:

```
import java.util.Scanner;

public class Cau4 {
```

```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào chuỗi A: ");
        String A = input.next();

        System.out.print("Nhập vào chuỗi B: ");
        String B = input.next();
        input.close();

        System.out.println(Find(B, A));
    }

    // hàm dùng để kiểm tra xem chuỗi B có nằm bên trong
chuỗi A không
    public static int Find(String B, String A) {
        for (int i=0; i<(A.length() - B.length()); i++) {
            boolean check = true; // Biến canh dùng để kiểm
tra
            for (int j=0; j<B.length(); j++)
                if (A.charAt(i+j) != B.charAt(j)) {
                    check = false;
                    break;
                }
            if (check) return i;
        }
        return -1;
    }
}
```

Câu 5: Cho một dãy số nguyên có N phần tử: A_0, A_1, A_2, …, A_{N-1} và một số nguyên K > 0. Giả sử trong dãy trên có một vài số nguyên xuất hiện đúng K lần. Hãy liệt kê ra các số nguyên đó với điều kiện chúng không phải là số có giá trị nhỏ nhất cũng như là lớn nhất trong dãy. Nếu không có số nguyên nào như vậy thì xuất ra màn hình một chuỗi NULL.

Hướng dẫn: để thuận tiện cho việc lập trình, trong câu này ta nên xây dựng ra các phương thức riêng để tìm giá trị lớn nhất nhỏ nhất, tìm số nguyên lặp lại K lần trong dãy và xuất ra kết quả tìm kiếm.

Trong đó, một số phương thức chúng ta cần phải truyền một tham chiếu để cập nhật lại giá trị trong quá trình tính toán và trả về cho hàm gọi. Cách thức truyền một tham chiếu như vậy là

chúng ta tạo ra một static class Parameter trong phần thành phần dữ liệu của lớp chính, và dùng các thuộc tính của lớp Parameter như là những tham chiếu để truyền cho các hàm.

Về quá trình tìm kiếm, ta cần lưu ý nếu một số nguyên lặp lại K lần trong dãy thì ta chỉ cần xét tới số nguyên ở vị trí thứ N-K mà thôi, và số nguyên nào sau khi xét xong thì ta gán cho nó một giá trị bằng min hoặc bằng max để sau này ta không phải đi xét lại số nguyên đó nữa.

Chương trình:

```
import java.util.Scanner;

public class Cau5 {

    private static class Parameter{
        private int max; // Số nguyên lớn nhất trong dãy số
        private int min; // Số nguyên bé nhất trong dãy số
        private int[] result; // Danh sách các số nguyên thỏa
mãn yêu cầu
        private int resultLength; // Kích thước mảng result
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào số phần tử của dãy: ");
        int N = input.nextInt();

        System.out.println("Nhập vào các phần tử của dãy số:
");
        int[] A = new int[N];

        for (int i=0; i<N; i++) {
            A[i] = input.nextInt();
        }

        System.out.print("Nhập vào số nguyên K: ");
        int K = input.nextInt();

        // Tạo ra đối tượng param để chứa các tham số truyền
qua lại giữa các phương thức
        Parameter param = new Parameter();
```

```
        // Tìm giá trị lớn nhất và nhỏ nhất trong dãy số
thông qua tham chiếu program được truyền vào cho phương thức
FindMaxAndMin()
        FindMaxAndMin(A, N, param);

        // Tìm kiếm các số nguyên thỏa mãn yêu cầu đề bài
        FindResult(A, N, K, param);

        // Xuất kết quả tìm được
        PrintResult(param);
    }

    public static void FindMaxAndMin(int[] A, int N,
Parameter param) {
        if (N < 1) return; // Nếu dãy số nhập vào không có
phần tử nào ==> thoát
        param.max = A[0];
        param.min = A[0];
        for (int i=1; i<N; i++) {
            if (A[i] > param.max)
                param.max = A[i];
            else if (A[i] < param.min)
                param.min = A[i];
        }
    }

    public static void FindResult(int[] A, int N, int K,
Parameter param) {
        // Chỉ cần duyệt tới phần tử thứ N - K
        int count;

        // Khởi tạo cho mảng result để chứa các kết quả tìm
được
        param.result = new int[N];
        param.resultLength = 0;
        for (int i=0; i<(N - K); i++) {
            if (A[i] > param.min && A[i] < param.max) {
                count = 1; // Mỗi khi xét tới một giá trị
A[i] mới thì ta phải khởi tạo lại biến đếm
                for (int j = i + 1; j<N; j++) {
                    if (A[j] == A[i]) {
```

```
                        count = count + 1; // Tăng biến đếm
số lần xuất hiện của số nguyên A[i]
                        A[j] = param.min; // Thay đổi giá trị
của số nguyên A[j] để sau này không phải đếm lại
                    }
                }
                if (count == K) {
                    param.result[param.resultLength] = A[i];
                    param.resultLength++;
                }
            }
        }
    }

    public static void PrintResult(Parameter param) {
        // Không tìm được số nguyên nào có số lần lặp là K
        if (param.resultLength == 0) {
            System.out.println("NULL");
        } else {
            System.out.println("Các số nguyên lặp lại K lần
trong dãy là:");
            for (int i=0; i<param.resultLength; i++)
                System.out.print(param.result[i] + " ");
        }
    }
}
```

Câu 6: Cho một số nguyên dương N, tiếp theo là một dãy N số nguyên A_0, A_1, A_2, …, A_{N-1} có thứ tự tăng dần (tức là $A_0 \le A_1 \le A_2 \le \ldots \le A_{N-1}$). Dựa trên giải thuật tìm kiếm nhị phân, hãy tìm một số nguyên $0 \le K \le N-1$ sao cho $A_0 + A_1 + \ldots + A_K = A_{K+1} + A_{K+2} + \ldots + A_{N-1}$. Trong trường hợp không tồn tại số nguyên K như vậy, hãy xuất ra giá trị -1.

Chương trình:

```
import java.util.Scanner;

public class Cau6 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào số phần tử của dãy số: ");
        int N = input.nextInt();
```

```
        System.out.println("Nhập vào các phần tử của dãy số:
");
        int[] A = new int[N];

        for (int i=0; i<N; i++) {
            A[i] = input.nextInt();
        }

        System.out.println("Dãy số đã nhập: ");
        for (int i=0; i<N; i++) {
            System.out.print(A[i] + " ");
        }
        System.out.println();

        int k = findK(A, N);
        if (k == -1) System.out.println("Không tìm thấy số
nguyên K thỏa mãn yêu cầu!");
        else System.out.println("Số nguyên k cần tìm: " + k);
    }

    public static int findK(int[] A, int N) {
        int left = 0, right = N-1, mid;
        int leftSum = 0, rightSum = 0;
        while (left <= right) {
            mid = (left + right) / 2;
            leftSum = Sum(A, 0, mid);
            rightSum = Sum(A, mid+1, N-1);
            if (leftSum > rightSum) right = mid - 1;
            else if (leftSum < rightSum) left = mid + 1;
            else return mid; // Nếu tổng trái bằng tổng phải
thì mid chính là vị trí cần tìm
        }
        return -1;
    }

    // Tính tổng các số từ left tới right
    public static int Sum(int[] A, int left, int right) {
        int sum = 0;
        for (int i=left; i<=right; i++)
            sum = sum + A[i];
        return sum;
```

```
    }
}
```