Các Giải Thuật Sắp Xếp Cơ Bản

Mục tiêu

Nội dung các bài tập dưới đây nhằm mục đích củng cố kiến thức cho sinh viên về các giải thuật sắp xếp cơ bản. Thực hành các bài tập này sẽ giúp sinh viên biết cách xử lý và áp dụng giải thuật sắp xếp vào các vấn đề thực tế trong cuộc sống.

Nội dung

Câu 1, 2, 3, 4: Sinh viên dựa vào bài giảng trên lớp để làm

Câu 5: Cho một dãy số A gồm n số tự nhiên ($n \leq 1000$). Hãy xuất ra số tự nhiên nhỏ nhất chưa xuất hiện trong dãy

Hướng dẫn

Ý tưởng giải quyết bài toán này rất đơn giản: ban đầu, chúng ta sẽ sắp xếp lại các phần tử trong dãy số A theo thứ tự tăng dần từ nhỏ đến lớn. Sau đó, ta duyệt qua các phần tử trong mảng bắt đầu từ phần tử thứ 2 đến phần tử cuối cùng. Với mỗi phần tử được duyệt, ta so sánh giá trị của nó với giá trị của phần tử trước đó, nếu giá trị phần tử đang xét lớn hơn giá trị của phần tử trước đó cộng 1, thì số tự nhiên nhỏ nhất chưa xuất hiện trong dãy chính là giá trị của phần tử trước đó cộng 1, dừng quá trình tìm kiếm và xuất ra kết quả. Ngược lại, tiếp tục quá trình tìm kiếm như vậy. Trường hợp nếu đã xét hết các phần tử trong dãy mà vẫn chưa tìm được số tự nhiên thỏa mãn yêu cầu đề bài, thì số tự nhiên đó được sinh ra bằng cách lấy giá trị của phần tử cuối cùng trong mảng (mảng đã sắp xếp) cộng 1. Xuất ra kết quả.

Chương trình

```
package Week3;
import java.util.*;

public class LabEx_Cau1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào số phần tử trong dãy: ");
        int n = input.nextInt();

        System.out.println("Nhập vào các phần tử trong dãy: ");
        int[] A = new int[n];
        for (int i=0; i<n; i++) {
            System.out.print("A[" + i + "] = ");
            A[i] = input.nextInt();
        }
```

```
        Sort(A, n); // Sắp xếp lại dãy số theo thứ tự tăng dần

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

        System.out.println("Số tự nhiên nhỏ nhất chưa xuất hiện trong dãy: " +
Find(A, n));
    }

    // Giải thuật sắp xếp Shaker Sort
    public static void Sort(int[] A, int n) {
        int left = 0, right = n - 1, k = 0;
        int trungGian;

        while (left < right) {
            for (int i=left; i<right; i++) {
                if (A[i] > A[i+1]) {
                    // Hoán vị
                    trungGian = A[i];
                    A[i] = A[i+1];
                    A[i+1] = trungGian;
                    k = i;
                }
            }
            right = k;

            for (int i=right; i>left; i--) {
                if (A[i] < A[i-1]) {
                    // Hoán vị
                    trungGian = A[i];
                    A[i] = A[i-1];
                    A[i-1] = trungGian;
                    k = i;
                }
            }
            left = k;
        }
    }

    public static int Find(int[] A, int n) { // hàm dùng để kiểm tra xem chuỗi B có
nằm bên trong chuỗi A không
        for (int i=1; i<n; i++) {
            if (A[i] > (A[i-1] + 1))
                return (A[i-1] + 1);
        }

        return (A[n-1]+ 1);
    }
}
```

Câu 6: Cho một dãy số A gồm n số nguyên dương. Hãy in ra các số chính phương có trong dãy theo thứ tự tăng dần, nếu không tồn tại số chính phương nào trong dãy thì in ra NULL.

Số chính phương là số có thể biểu diễn được dưới dạng bình phương của một số khác.

Hướng dẫn

Ý tưởng giải quyết bài toán này cũng khá đơn giản: ban đầu, chúng ta sẽ sắp xếp lại các phần tử trong dãy số A theo thứ tự tăng dần từ nhỏ đến lớn. Sau đó, ta duyệt qua tất cả các phần tử trong mảng A, với mỗi phần tử được duyệt, kiểm tra xem nếu nó là số chính phương thì ta xuất ra màn hình, ngược lại thì bỏ qua. Kết quả xuất ra màn hình tự động sẽ theo thứ tự tăng dần. Tuy nhiên, cách làm này có một khuyết điểm đó là nó sẽ xuất ra các số chính phương trùng nhau có trong dãy nhiều lần, và kết quả này không phù hợp với mô tả của đề bài. Do đó, để khắc phục nhược điểm này, chúng ta dùng thêm một biến để lưu số chính phương vừa tìm được có tên là lastSquaredNumber, ban đầu lastSquaredNumber = -1. Sau đó, mỗi khi duyệt qua một phần tử của mảng, thì nếu phần tử được duyệt có giá trị lớn hơn lastSquaredNumber thì ta mới xét tiếp, và nếu phần tử được xét là một số chính phương thì ta dùng lastSquaredNumber để lưu lại số này.

Chương trình

```
package Week3;
import java.util.*;
import java.lang.Math;

public class LabEx_Cau2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào số phần tử trong dãy: ");
        int n = input.nextInt();

        System.out.println("Nhập vào các phần tử trong dãy: ");
        int[] A = new int[n];
        for (int i=0; i<n; i++) {
            System.out.print("A[" + i + "] = ");
            A[i] = input.nextInt();
        }

        Sort(A, n); // Sắp xếp lại dãy số theo thứ tự tăng dần

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

        System.out.println("Các số chính phương có trong dãy: ");
        int lastSquaredNumber = -1;
        for (int i=0; i<n; i++) {
            if (A[i] > lastSquaredNumber && IsSquaredNumber(A[i])) {
                lastSquaredNumber = A[i];
                System.out.print(A[i] + " ");
```

```
            }
        }
    }

    // Giải thuật sắp xếp Shell Sort
    public static void Sort(int[] A, int n) {
        int h = 1;
        while (h < n / 3) h = h * 3 + 1;

        int x, j;
        while (h > 0) {
            for (int i=h; i<n; i++) {
                x = A[i];
                j = i - h; // A[j] là phần tử đứng kề trước A[i] trong cùng dãy con

                // Sắp xếp dãy con có chứa x bằng phương pháp chèn trực tiếp
                while (j >= 0 && A[j] > x) {
                    A[j + h] = A[j];
                    j = j - h;
                }
                A[j + h] = x;
            }
            h = (h - 1) / 3;
        }
    }

    // Hàm dùng để kiểm tra xem một số có phải là số chính phương hay không
    public static boolean IsSquaredNumber(int number) {
        double root = Math.sqrt(number);
        if (root * root == number) return true;
        else return false;
    }
}
```

Câu 7: Cho một dãy số A gồm N số nguyên và một số nguyên dương M ($0 < M \leq N$). Hãy tính hiệu số giữa tổng của N-M số lớn nhất và tổng của N-M số bé nhất.
Giới hạn: $0 < N, M < 1000$

Hướng dẫn

Ý tưởng thực hiện: sắp xếp các phần tử trong dãy số theo thứ tự tăng dần. Sau đó, tính tổng của N-M phần tử đầu tiên trong dãy là tổng bé nhất, và tổng của N-M phần tử cuối cùng trong dãy là tổng lớn nhất. Tính hiệu số giữa 2 tổng trên ta sẽ được kết quả

Chương trình

```
package Week3;
import java.util.*;
public class LabEx_Cau3 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào số phần tử trong dãy: ");
        int N = input.nextInt();

        int  M;
```

```
        do {
            System.out.print("Nhập vào số nguyên dương M: ");
            M = input.nextInt();
        } while (M < 0 || M > N);

        System.out.println("Nhập vào các phần tử trong dãy: ");
        int[] A = new int[N];
        for (int i=0; i<N; i++) {
            System.out.print("A[" + i + "] = ");
            A[i] = input.nextInt();
        }

        Sort(A, N); // Sắp xếp lại dãy số theo thứ tự tăng dần

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

        // Tính tổng của N-M số đầu tiên
        int minSum = 0;
        for (int i=0; i<N-M; i++)
            minSum = minSum + A[i];

        // Tính tổng của N-M số cuối cùng
        int maxSum = 0;
        for (int i=0; i<N-M; i++)
            maxSum = maxSum + A[N-1-i];

        System.out.print("Hiệu số giữa hai tổng min và max của dãy số: " + (maxSum -
minSum));
    }


    public static void Heapify(int A[], int n, int i) {
        int max = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if (left < n && A[left] > A[max]) max = left;
        if (right < n && A[right] > A[max]) max = right;
        if (max != i)
        {
            int temp = A[i];
            A[i]= A[max];
            A[max] = temp;
            Heapify(A, n, max);
        }
    }

    // Giải thuật sắp xếp Heap Sort
    public static void Sort(int A[], int n) {// Sắp xếp
        for (int i=n/2-1; i>=0; i--) {
            Heapify(A, n, i);
        }
        for (int i=n-1; i>=0; i--)
        {
            int temp = A[0];
            A[0]= A[i];
```

```
            A[i] = temp;
            Heapify(A, i, 0);
        }
    }

}
```

Câu 8: Một loại virus nguy hiểm chết người mới đang lây lan trên khắp hành tinh. Các nhà khoa học đã chạy đua với thời gian để tìm ra một loại vaccine mới có khả năng chống lại loại virus này. Vaccine được tạo ra trên cơ sở biến đổi các tế bào của virus thành những tác nhân sinh học chống lại chúng. Tuy nhiên, vaccine chỉ có thể chữa khỏi cho người bệnh nếu lượng tác nhân sinh học bên trong vaccine nhiều hơn 1/3 số tế bào virus bên trong cơ thể người. Cho một lô hàng có N lọ vaccine dùng để chữa trị cho N bệnh nhân đang bị nhiễm virus. Cho biết thông tin về lượng tác nhân có trong mỗi lọ vaccine và số lượng tế bào virus trong cơ thể mỗi người bệnh. Hãy xác định xem lô vaccine nói trên có thể chữa khỏi cho tất cả những bệnh nhân này hay không. Kết quả xuất ra màn hình là YES hoặc NO.

Hướng dẫn

Ý tưởng thực hiện: sắp xếp các lọ vaccine theo thứ tự từ nhỏ đến lớn về liều lượng tác nhân sinh học và sắp xếp danh sách các bệnh nhân theo thứ tự từ nhẹ tới nặng về tải lượng tế bào virus có trong cơ thể. Sau đó, so sánh tuần tự lọ vaccine thứ i với tải lượng tế bào virus của bệnh nhân thứ i. Nếu có một lọ vaccine nào đó có lượng tác nhân sinh học không đủ để kháng lại lượng tế bào virus có trong cơ thể bệnh nhân thì ta xuất ra kết quả là NO, còn ngược lại sau khi xét hết tất cả mà vẫn đáp ứng thì kết quả là YES.

Chương trình

```
package Week3;
import java.util.*;
public class LabEx_Cau4 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Nhập vào số lượng bệnh nhân: ");
        int N = input.nextInt();

        System.out.println("Nhập vào liều lượng sinh học của " + N + " lọ vaccine:
");
        int[] vaccine = new int[N];
        for (int i=0; i<N; i++) {
            System.out.print("vaccine[" + i + "] = ");
            vaccine[i] = input.nextInt();
        }

        System.out.println("Nhập vào tải lượng tế bào virus trong cơ thể của " + N +
" bệnh nhân: ");
        int[] virus = new int[N];
        for (int i=0; i<N; i++) {
```

```
            System.out.print("virus[" + i + "] = ");
            virus[i] = input.nextInt();
        }


        Sort(vaccine, N); // Sắp xếp lại các lọ vaccine theo thứ tự tăng dần
        Sort(virus, N); // Sắp xếp lại các bệnh nhân theo thứ tự tăng dần

        System.out.println("Danh sách vaccine/virus đã sắp xếp: ");
        for (int i=0; i<N; i++) {
            System.out.print(vaccine[i] + " ");
        }
        System.out.println();

        for (int i=0; i<N; i++) {
            System.out.print(virus[i] + " ");
        }
        System.out.println();

        System.out.println(Check(vaccine, virus, N));
    }


    // Giải thuật sắp xếp Insertion Sort
    public static void Sort(int A[], int n) {
        int x, j;
        // Phần tử A[0] của mảng xem như đã có thứ tự
        for (int i=1; i<n; i++) {
            x = A[i];
            j = i - 1;
            while (j >= 0 && A[j] > x) {
                A[j+1] = A[j];
                j = j - 1;
            }
            A[j+1] = x; // Chèn x vào dãy
        }
    }

    public static String Check(int[] vaccine, int[] virus, int N) {
        for (int i=0; i<N; i++) {
            if (vaccine[i] <= (virus[i] / 3))
                return "NO";
        }
        return "YES";
    }
}
```