Subarrays with k different integers

```
import java.util.*;
class Main{
    public static int k_dist_subarrays(int a[], int k){
        int r=0;
        Map<Integer, Integer> m=new HashMap<>();
        int i=0, j=0;
        while(j<a.length){
            m.put(a[j], m.getOrDefault(a[j], 0)+1);
            while(m.size()==k+1){
                m.put(a[i], m.get(a[i])-1);
                if(m.get(a[i])==0){
                    m.remove(a[i]);
                }
                i++;
            }
            r+=j-i+1;
            j++;
        }
        return r;
    }
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int a[]=new int[n];
        for(int i=0; i<n; i++){
            a[i]=sc.nextInt();
        }
        int k=sc.nextInt();
        System.out.println(k_dist_subarrays(a, k)-k_dist_subarrays(a, k-1));
    }
}
```

1)
```
5
1 2 1 2 3
2
=>7
```

2)
```
5
1 2 1 3 4
3
=>3
```

Shortest subarray with sum at least k

```
import java.util.*;
class Main{
    public static int shortest_subarray(int a[], int k){
        int r=Integer.MAX_VALUE;
        long ps[]=new long[a.length+1];
        Deque<Integer> dq=new LinkedList<>();
        dq.offerLast(0);
        for(int i=0; i<a.length; i++){
            ps[i+1]=ps[i]+a[i];
            while(!dq.isEmpty() && ps[dq.peekLast()]>=ps[i+1]){
                dq.pollLast();
            }
            dq.offerLast(i+1);
            while(!dq.isEmpty() && ps[dq.peekFirst()]+k<=ps[i+1]){
                r=Math.min(r, i-dq.peekFirst()+1);
                dq.pollFirst();
            }
        }
        return (r==Integer.MAX_VALUE)?-1:r;
    }
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int a[]=new int[n];
        for(int i=0; i<n; i++){
            a[i]=sc.nextInt();
        }
        int k=sc.nextInt();
        System.out.println(shortest_subarray(a, k));
    }
}
```

1) 1
   1
   1
   =>1
2) 2
   1 2
   4
   =>-1
3) 3
   2 -1 2

```
3
=>3
```

Implement Fenwick Tree

```
import java.util.*;
class Fenwick{
    public int nums[], bit[];
    public Fenwick(int n[]){
        nums=n;
        bit=new int[nums.length+1];
        buildtree();
    }
    public void buildtree(){
        for(int i=0; i<nums.length; i++){
            updatetree(i, nums[i]);
        }
    }
    public void updatetree(int ind, int val){
        ind=ind+1;
        while(ind<bit.length){
            bit[ind]+=val;
            ind+=ind&(-ind);
        }
    }
    void update(int ind, int val){
        int d=val-nums[ind];
        updatetree(ind, d);
        nums[ind]=val;
    }
    public int sum(int ind){
        int s=0;
        ind=ind+1;
        while(ind>0){
            s+=bit[ind];
            ind-=ind&(-ind);
        }
        return s;
    }
    public int sumRange(int i, int j){
        return sum(j)-sum(i-1);
    }
}
```

```
class Main{
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int a[]=new int[n];
        for(int i=0; i<n; i++){
            a[i]=sc.nextInt();
        }
        Fenwick tree=new Fenwick(a);
        int s1=sc.nextInt();
        int s2=sc.nextInt();
        System.out.println(tree.sumRange(s1, s2));
    }
}
```

```
1)  8
    1 2 13 4 25 16 17 8
    2 6
    =>75
```

Segment tree

```
import java.util.*;
class Solution{
    class SegmentTreeNode{
        int start, end;
        SegmentTreeNode left, right;
        int val;
        public SegmentTreeNode(int start, int end){
            this.start=start;
            this.end=end;
            left=right=null;
            val=0;
        }
    }
    SegmentTreeNode root;
    public int maxEvents(int[][] events){
        if(events==null || events.length==0){
            return 0;
        }
        Arrays.sort(events, (a, b)->{
```

```
        if(a[1]==b[1]){
            return a[0]-b[0];
        }
        return a[1]-b[1];
    });
    int lastDay=events[events.length-1][1];
    int firstDay=Integer.MAX_VALUE;
    for(int i=0; i<events.length; i++){
        firstDay=Math.min(firstDay, events[i][0]);
    }
    root=buildSegmentTree(firstDay, lastDay);
    int count=0;
    for(int event[]: events){
        int earliestDay=query(root, event[0], event[1]);
        if(earliestDay!=Integer.MAX_VALUE){
            count++;
            update(root, earliestDay);
        }
    }
    return count;
}
private SegmentTreeNode buildSegmentTree(int start, int end){
    if(start>end){
        return null;
    }
    SegmentTreeNode node=new SegmentTreeNode(start, end);
    node.val=start;
    if(start!=end){
        int mid=start+(end-start)/2;
        node.left=buildSegmentTree(start, mid);
        node.right=buildSegmentTree(mid+1, end);
    }
    return node;
}
private void update(SegmentTreeNode curr, int lastDay){
    if(curr.start==curr.end){
        curr.val=Integer.MAX_VALUE;
    }
    else{
        int mid=curr.start+(curr.end-curr.start)/2;
```

```
            if(mid>=lastDay){
                update(curr.left, lastDay);
            }
            else{
                update(curr.right, lastDay);
            }
            curr.val=Math.min(curr.left.val, curr.right.val);
        }
    }
    private int query(SegmentTreeNode curr, int left, int right){
        if(curr.start==left && curr.end==right){
            return curr.val;
        }
        int mid=curr.start+(curr.end-curr.start)/2;
        if(mid>=right){
            return query(curr.left, left, right);
        }
        else if(mid<left){
            return query(curr.right, left, right);
        }
        else{
            return Math.min(query(curr.left, left, mid), query(curr.right, mid+1, right));
        }
    }
}
class Main{
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        sc.nextLine();
        String str[]=sc.nextLine().split(",");
        int nums[][]=new int[n][2];
        for(int i=0; i<n; i++){
            String val[]=str[i].split(" ");
            nums[i][0]=Integer.parseInt(val[0]);
            nums[i][1]=Integer.parseInt(val[1]);
        }
        Solution sol=new Solution();
        System.out.println(sol.maxEvents(nums));
    }
```

```
}
```

1) 4
   1 2,2 4,2 3,2 2
   =>4

Treap

```
import java.util.*;
class Main{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        System.out.println(findNoOfRevPairs(arr));
        sc.close();
    }

    static int findNoOfRevPairs(int[] arr) {
        Treap treap = new Treap();
        int count = 0;
        for (int i = 0; i < arr.length; i++) {
            count += treap.query(arr[i] * 2 + 1, Integer.MAX_VALUE);
            treap.insert(arr[i]);
        }
        return count;
    }
}


class Node {
    int key, priority;
    Node left, right;

    Node(int key) {
        this.key = key;
        priority = new Random().nextInt();
    }
}


class Treap {
    Node root = null;
```

```
void insert(int key) {
    root = insert(root, key);
}

Node insert(Node root, int key) {
    if (root == null)
        return new Node(key);
    if (key < root.key) {
        root.left = insert(root.left, key);
        if (root.left.priority > root.priority)
            root = rightRotate(root);
    } else {
        root.right = insert(root.right, key);
        if (root.right.priority > root.priority)
            root = leftRotate(root);
    }
    return root;
}

private Node rightRotate(Node root2) {
    Node left = root2.left;
    Node right = left.right;
    left.right = root2;
    root2.left = right;
    return left;
}

Node leftRotate(Node root2) {
    Node right = root2.right;
    Node left = right.left;
    right.left = root2;
    root2.right = left;
    return right;
}

int query(int start, int end) {
    return query(root, start, end);
}

int query(Node root, int start, int end) {
    if (root == null)
        return 0;
    if (root.key >= start && root.key <= end)
```

```
            return 1 + query(root.left, start, end) + query(root.right, start, end);
        else if (root.key < start)
            return query(root.right, start, end);
        else
            return query(root.left, start, end);
    }
}
```

```
   1)  5
       1 3 2 3 1
       => 2
```

Topological sort

```
import java.util.*;
class Solution{
    private static void dfs(int x, List<List<Integer>> adj, boolean visited[], Stack<Integer> stack){
        visited[x]=true;
        for(int i: adj.get(x)){
            if(!visited[i]){
                dfs(i, adj, visited, stack);
            }
        }
        stack.push(x);
    }
    public static int[] topologicalSort(int V, List<List<Integer>> adj){
        Stack<Integer> stack=new Stack<>();
        boolean visited[]=new boolean[V];
        for(int i=0; i<V; i++){
            if(!visited[i]){
                dfs(i, adj, visited, stack);
            }
        }
        int arr[]=new int[V];
        int i=0;
        while(!stack.isEmpty()){
            arr[i++]=stack.pop();
        }
        return arr;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        int vertices=sc.nextInt();
        int no_of_edges=sc.nextInt();
        List<List<Integer>> l=new ArrayList<>();
        for(int i=0; i<vertices; i++){
            l.add(new ArrayList<>());
        }
        for(int i=0; i<no_of_edges; i++){
            int source=sc.nextInt(), destination=sc.nextInt();
            l.get(source).add(destination);
        }
        int r[]=topologicalSort(vertices, l);
        System.out.println(Arrays.toString(r));
    }
}
```

```
1)  6 6
    5 2
    5 0
    4 0
    4 1
    2 3
    3 1
    => [5,4,2,3,1,0]
```

Finding articulation point in a graph

```
import java.util.*;
class Solution{
    public static int time=0;
    public static void dfs(List<List<Integer>> adj, int disc[], int low[], int parent[], boolean is_articulation[], int u){
        disc[u]=low[u]=time;
        time++;
        int children=0;
        for(int v: adj.get(u)){
            if(disc[v]==-1){
                children++;
                parent[v]=u;
                dfs(adj, disc, low, parent, is_articulation, v);
                low[u]=Math.min(low[u], low[v]);
                if(parent[u]==-1 && children>1){
                    is_articulation[u]=true;
                }
                if(parent[u]!=-1 && low[v]>=disc[u]){
                    is_articulation[u]=true;
                }
            }
            else if(v!=parent[u]){
                low[u]=Math.min(low[u], disc[v]);
            }
        }
    }
    public static List<Integer> find_articulation_points(int n, int edges[][]){
        int disc[]=new int[n];
        Arrays.fill(disc, -1);
        int low[]=new int[n];
        Arrays.fill(low, -1);
        int parent[]=new int[n];
        Arrays.fill(parent, -1);
        boolean is_articulation[]=new boolean[n];
        List<List<Integer>> adj=new ArrayList<>();
        for(int i=0; i<n; i++){
          adj.add(new ArrayList<>());
        }
        for(int a[]: edges){
            adj.get(a[0]).add(a[1]);
            adj.get(a[1]).add(a[0]);
        }
        for(int i=0; i<n; i++){
            if(disc[i]==-1){
                dfs(adj, disc, low, parent, is_articulation, i);
            }
        }
        List<Integer> l=new ArrayList<>();
        for(int i=0; i<n; i++){
            if(is_articulation[i]){
                l.add(i);
            }
        }
        return l;
    }
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        int vertices=sc.nextInt();
        int no_of_edges=sc.nextInt();
        int edges[][]=new int[no_of_edges][2];
        for(int i=0; i<no_of_edges; i++){
            int source=sc.nextInt(), destination=sc.nextInt();
            edges[i][0]=source;
            edges[i][1]=destination;
        }
        System.out.println(find_articulation_points(vertices, edges));
    }
}
```

```
   1)  7 8
       0 1
       1 2
       2 0
       1 3
       1 4
       1 6
       3 4
       1 5
       => 1
```

Is string permutation palindrome

```
import java.util.*;
class Test{
    public static boolean is_permuation_palindrome(String s){
        int k=0;
        for(char c: s.toCharArray()){
            k^=(1<<(c-'a'));
        }
        int c=0;
        while(k>0){
            k&=(k-1);
            c++;
        }
        return c==1 || c==0;
    }
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        String s=sc.next();
        System.out.println(is_permuation_palindrome(s));
    }
}
```

1) racecar
   => true

Find index pairs

```
import java.util.*;
class Solution {
    public static List<int[]> indexPairs(String text, String[] words){
        List<int[]> l=new ArrayList<int[]>();
        for(String word: words) {
            int wl=word.length();
            int i=0;
            while(i>=0) {
                i=text.indexOf(word, i);
                if(i>=0){
                    l.add(new int[]{i, i+wl-1});
                    i++;
                }
            }
        }
        Collections.sort(l, (a, b)->{
            if(a[0]==b[0]){
                return a[1]-b[1];
            }
            return a[0]-b[0];
        });
        return l;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        String text=sc.nextLine();
        String words[]=sc.nextLine().split(" ");
        for(int a[]: indexPairs(text, words)){
            System.out.println(Arrays.toString(a));
        }
    }
}
```

1) thestoryofleetcodeandme
   story fleet leetcode
   ```
   => [3, 7]
         [9, 13]
         [10, 17]
   ```

Lowest Common Ancestor

```
import java.util.*;
class Solution{
    class Node{
        int data;
        Node left, right;
        public Node(int d){
            data=d;
            left=right=null;
        }
    }
    Node root=null;
    void insert(int d){
        if(root==null){
            root=new Node(d);
            return;
        }
        Queue<Node> q=new LinkedList<Node>();
        q.add(root);
        while(!q.isEmpty()){
            Node temp=q.poll();
            if(temp.left==null) {
                temp.left=new Node(d);
                break;
            }
            else{
                q.add(temp.left);
            }
            if(temp.right==null){
                temp.right=new Node(d);
                break;
            }
            else{
                q.add(temp.right);
            }
        }
    }
    Node lowestCommonAncestor(Node root, int p, int q){
        if(root==null || root.data==p || root.data==q){
            return root;
        }
        Node left=lowestCommonAncestor(root.left, p, q);
        Node right=lowestCommonAncestor(root.right, p, q);
        if(left==null){
            return right;
        }
        else if(right==null){
            return left;
        }
        else{
            return root;
        }
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        Solution bt=new Solution();
        int n=sc.nextInt();
        for(int i=0; i<n; i++){
            bt.insert(sc.nextInt());
        }
        int p=sc.nextInt(), q=sc.nextInt();
        Node r=bt.lowestCommonAncestor(bt.root, p, q);
        System.out.println((r==null)?-1:r.data);
    }
}
```

```
  1)  7
      1 2 3 4 5 6 7
      4 5
      => 2
```

Longest Increasing path in a matrix

```
import java.util.*;
class Solution{
    public static int dx[]=new int[]{0, 0, -1, 1};
    public static int dy[]=new int[]{1, -1, 0, 0};
    public static int longestIncreasingPath(int[][] matrix) {
       if(matrix==null || matrix.length==0){
            return 0;
       }
       int longest=0, m=matrix.length, n=matrix[0].length;
       int[][] dp = new int[m][n];
       for (int i=0; i<m; i++){
           for(int j=0; j<n; j++){
              longest=Math.max(longest, dfs(i, j, matrix, dp));
            }
        }
        return longest;
    }
    public static int dfs(int row, int col, int[][] matrix, int[][] dp) {
        if(dp[row][col]>0){
            return dp[row][col];
        }
        int m=matrix.length, n=matrix[0].length;
        int currentLongest=0;
        for(int c=0; c<4; c++){
            int i=row+dx[c];
            int j=col+dy[c];
            if(i>=0 && i<m && j>=0 && j<n && matrix[row][col]<matrix[i][j]){
                currentLongest = Math.max(currentLongest, dfs(i, j, matrix, dp));
            }
        }
        return dp[row][col]=1+currentLongest;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        int m=sc.nextInt(), n=sc.nextInt();
        int matrix[][]=new int[m][n];
        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
                matrix[i][j]=sc.nextInt();
            }
        }
        System.out.println(longestIncreasingPath(matrix));
    }
}
```

```
   1)  3 3
       9 9 4
       6 6 8
       2 1 1
       => 4
```

Lexicographically Smallest Equivalent String

```
import java.util.*;
class Solution {
    public static String smallestEquivalentString(String s1, String s2, String baseStr) {
        int[] graph=new int[26];
        for(int i=0; i<26; i++){
            graph[i]=i;
        }
        for(int i=0; i<s1.length(); i++){
            int first=s1.charAt(i)-'a', second=s2.charAt(i)-'a';
            int end1=find(graph, first);
            int end2=find(graph, second);
            if(end1<end2){
                graph[end2]=end1;
            }
            else{
                graph[end1]=end2;
            }
        }
        StringBuilder sb=new StringBuilder();
        for(int i=0; i<baseStr.length(); i++){
            char c=baseStr.charAt(i);
            sb.append((char)('a'+find(graph, c-'a')));
        }
        return sb.toString();
    }
    public static int find(int[] graph, int index){
        while(graph[index]!=index){
            index=graph[index];
        }
        return index;
    }
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        String a=sc.nextLine(), b=sc.nextLine();
        String base=sc.nextLine();
        System.out.println(smallestEquivalentString(a, b, base));
    }
}
```

1) parker
   morris
   parser
   => makkek