```
 1: package problems;
 2:
 3: public class Adding2NumbersList
 4: {
 5:         static int carry = 0;
 6:
 7:         private static void padZeros(MyLinkedList<Integer> l, int len)
 8:         {
 9:                 for (int i = 0; i < len; i++)
10:                         l.insertAtStart(0);
11:         }
12:
13:         private static MyLinkedList<Integer> addLists(MyLinkedList<Integer> l1, MyLinkedList<Integer> l2)
14:         {
15:                 if (l1 == null && l2 == null)
16:                         return null;
17:
18:                 if (l1 == null || l1.getHead() == null)
19:                         return l2;
20:
21:                 if (l2 == null || l2.getHead() == null)
22:                         return l1;
23:
24:                 int len1 = l1.length();
25:                 int len2 = l2.length();
26:
27:                 if (len1 > len2)
28:                         padZeros(l2, len1 - len2);
29:                 else
30:                         padZeros(l1, len2 - len1);
31:
32:                 l1.printList();
33:                 l2.printList();
34:                 MyLinkedList<Integer> l3 = new MyLinkedList<Integer>();
35:                 l3.setHead(addLists(l1.getHead(), l2.getHead()));
36:
37:                 if (carry > 0)
38:                 {
39:                         l3.insertAtStart(carry);
40:                         carry = 0;
41:                 }
42:
43:                 return l3;
44:         }
45:
46:         private static ListNode<Integer> addLists(ListNode<Integer> t1, ListNode<Integer> t2)
47:         {
48:                 if (t1 == null && t2 == null)
```

```
49:                             return null;
50:
51:                     ListNode<Integer> t = addLists(t1.next, t2.next);
52:                     int sum = t1.data + t2.data + carry;
53:                     ListNode<Integer> t3 = new ListNode<Integer>(sum % 10);
54:                     carry = sum / 10;
55:                     t3.next = t;
56:
57:                     return t3;
58:             }
59:
60:             public static void main(String[] args)
61:             {
62:                     MyLinkedList<Integer> l1 = new MyLinkedList<Integer>();
63:                     l1.insertAtEnd(7);
64:                     l1.insertAtEnd(1);
65:                     l1.insertAtEnd(6);
66:                     l1.insertAtEnd(6);
67:
68:                     // l1.printList();
69:                     // l1.setHead(l1.reverseList(l1.getHead()));
70:                     // l1.printList();
71:
72:                     MyLinkedList<Integer> l2 = new MyLinkedList<Integer>();
73:                     l2.insertAtEnd(5);
74:                     l2.insertAtEnd(9);
75:                     l2.insertAtEnd(5);
76:
77:                     MyLinkedList<Integer> l3 = addLists(l1, l2);
78:                     if (l3 != null)
79:                             l3.printList();
80:             }
81:
82: }
```

```
 1: package problems;
 2:
 3: public class Adding2NumbersListRev
 4: {
 5:
 6:         public static MyLinkedList<Integer> addLists(MyLinkedList<Integer> l1, MyLinkedList<Integer> l2)
 7:         {
 8:                 if (l1 == null && l2 == null)
 9:                         return null;
10:
11:                 ListNode<Integer> t1 = (l1 != null) ? l1.getHead() : null;
12:                 ListNode<Integer> t2 = (l2 != null) ? l2.getHead() : null;
13:
14:                 MyLinkedList<Integer> l3 = new MyLinkedList<Integer>();
15:
16:                 int sum = 0, carry = 0;
17:
18:                 while (t1 != null && t2 != null)
19:                 {
20:                         sum = (t1.data + t2.data + carry);
21:                         carry = sum / 10;
22:                         l3.insertAtEnd(sum % 10);
23:                         t1 = t1.next;
24:                         t2 = t2.next;
25:                 }
26:
27:                 while (t1 != null)
28:                 {
29:                         sum = t1.data + carry;
30:                         l3.insertAtEnd(sum % 10);
31:                         carry = sum / 10;
32:                         t1 = t1.next;
33:                 }
34:
35:                 while (t2 != null)
36:                 {
37:                         sum = t2.data + carry;
38:                         l3.insertAtEnd(sum % 10);
39:                         carry = sum / 10;
40:                         t2 = t2.next;
41:                 }
42:
43:                 if (carry > 0)
44:                         l3.insertAtEnd(carry);
45:
46:                 return l3;
47:
48:         }
```

```
49:
50:         public static void main(String[] args)
51:         {
52:                  MyLinkedList<Integer> l1 = new MyLinkedList<Integer>();
53:                  l1.insertAtEnd(7);
54:                  l1.insertAtEnd(1);
55:                  l1.insertAtEnd(6);
56:                  l1.printList();
57:
58:                 MyLinkedList<Integer> l2 = new MyLinkedList<Integer>();
59:                 l2.insertAtEnd(5);
60:                 l2.insertAtEnd(9);
61:                 l2.insertAtEnd(5);
62:                 l2.printList();
63:
64:                 MyLinkedList<Integer> l3 = addLists(l1, l2);
65:                 if (l3 != null)
66:                         l3.printList();
67:         }
68: }
```

```
 1: package problems;
 2:
 3: public class ArrayHopper
 4: {
 5:         // prints the path from first touch down till end of the canyon
 6:         static void printPath(int[] path, int idx)
 7:         {
 8:                 if (idx == 0)
 9:                         return;
10:                 printPath(path, path[idx]);
11:                 System.out.print(path[idx] + ", ");
12:         }
13:
14:         // finds the last touch down which can lead out of the canyon
15:         static int findLastIndex(int[] canyons)
16:         {
17:                 int lastIndex = -1;
18:                 for (int i = canyons.length - 1; i >= 0; i--)
19:                 {
20:                         if ((canyons[i] + i) >= canyons.length)
21:                                 lastIndex = i;
22:                 }
23:                 return lastIndex;
24:         }
25:
26:         static void findPath(int[] canyons)
27:         {
28:                 int size = canyons.length;
29:                 if (size == 0 || canyons[0] == 0)
30:                 {
31:                         System.out.println("failure");
32:                         return;
33:                 }
34:                 int hops[] = new int[size]; // stores the min touch downs from to reach
35:                                                                         // all points in the array.
36:                 int path[] = new int[size]; // stores the previous touch down for all
37:                                                                         // points in the array.
38:                 hops[0] = 0;
39:
40:                 for (int i = 1; i < size; i++)
41:                 {
42:                         hops[i] = Integer.MAX_VALUE;
43:                         for (int j = 0; j < i; j++)
44:                         {
45:                                 if (i <= j + canyons[j] && hops[j] != Integer.MAX_VALUE)
46:                                 {
47:                                         if (hops[i] > hops[j] + 1)
48:                                         {
```

```
49:                                                     path[i] = j;
50:                                                     hops[i] = hops[j] + 1;
51:                                             }
52:                                             break;
53:                                     }
54:                             }
55:                     }
56:                     int lastIndex = findLastIndex(canyons);
57:                     if (hops[size - 1] == Integer.MAX_VALUE || lastIndex == -1)
58:                     {
59:                             System.out.println("failure");
60:                             return;
61:                     }
62:
63:                     printPath(path, lastIndex);
64:                     System.out.print(lastIndex + ", out");
65:             }
66:
67:             public static void main(String args[]) throws Exception
68:             {
69:                     int[] canyons = { 1, 0, 0, 4, 0, 0, 0, 2, 0 };
70:                     findPath(canyons);
71:             }
72: }
```

```
 1: package problems;
 2:
 3: public class ArrayRearrange
 4: {
 5:         public static void swap(int a[], int head, int tail)
 6:         {
 7:                 int c = a[head];
 8:                 a[head] = a[tail];
 9:                 a[tail] = c;
10:         }
11:
12:         public static void printArray(int a[])
13:         {
14:                 int length = 0;
15:                 while (length < a.length)
16:                 {
17:                         System.out.print(a[length] + " ");
18:                         length++;
19:                 }
20:                 System.out.println();
21:         }
22:
23:         public static void main(String[] args)
24:         {
25:                 int a[] = { -1, 0, -2, 0, 6, -5, 0, -3, -4 };
26:
27:                 int head = 0, pass = 0;
28:                 while (pass < 2)
29:                 {
30:                         int tail = a.length - 1;
31:                         while (head <= tail)
32:                         {
33:                                 if (pass > 0 ? (a[head] == 0) : (a[head] < 0))
34:                                 {
35:                                         head++;
36:                                         continue;
37:                                 }
38:                                 else
39:                                 {
40:                                         if (pass > 0 ? (a[tail] == 0) : (a[tail] < 0))
41:                                         {
42:                                                 swap(a, head, tail);
43:                                                 head++;
44:                                         }
45:                                         tail--;
46:                                 }
47:                         }
48:                         pass++;
```

```
49:                         System.out.println(head + " " + tail);
50:                         printArray(a);
51:                     }
52:             }
53: }
```

```
 1: package problems;
 2:
 3: import java.io.File;
 4: import java.io.FileWriter;
 5: import java.io.IOException;
 6:
 7: class A
 8: {
 9:
10: }
11:
12: public class B extends A
13: {
14:         public void getA()
15:         {
16:                 System.out.println("A");
17:         }
18:
19:         public static void main(String[] args) throws IOException
20:         {
21:                 A a = new A();
22:                 A b = new B();
23:                 ((B) a).getA(); // throws a Runtime exception - ClassCastException
24:                 ((B) b).getA();
25:
26:                 File file = new File("/Users/Anand/Documents/home_value.txt");
27:
28:                 FileWriter writer = new FileWriter(file);
29:
30:                 writer.write("50000 26000\n");
31:                 for (int i = 0; i < 50000; i++)
32:                         writer.write(i + " ");
33:                 writer.flush();
34:                 writer.close();
35:         }
36:
37: }
```

```
 1: package problems;
 2:
 3: import java.util.LinkedList;
 4: import java.util.Queue;
 5:
 6: class TreeNode
 7: {
 8:
 9:         public int data;
10:         public TreeNode left = null;
11:         public TreeNode right = null;
12:
13:         public TreeNode(int d)
14:         {
15:                 data = d;
16:         }
17: }
18:
19: public class BinaryTree
20: {
21:         TreeNode root;
22:
23:         public void deleteTree()
24:         {
25:                 root = null;
26:         }
27:
28:         public int height(TreeNode root)
29:         {
30:                 if (root == null)
31:                         return -1;
32:
33:                 return Math.max(height(root.left), height(root.right)) + 1;
34:         }
35:
36:         public static void printInOrder(TreeNode root)
37:         {
38:                 if (root == null)
39:                         return;
40:                 printInOrder(root.left);
41:                 System.out.print(root.data + " ");
42:                 printInOrder(root.right);
43:         }
44:
45:         static TreeNode last_node = null;
46:
47:         public TreeNode inOrderSuccessor(TreeNode root, TreeNode node)
48:         {
```

```
49:                     if (root == null || node == null)
50:                             return null;
51:                     TreeNode succ = null;
52:
53:                     succ = inOrderSuccessor(root.left, node);
54:                     if (last_node != null && last_node == node)
55:                             return root;
56:                     last_node = root;
57:                     succ = inOrderSuccessor(root.right, node);
58:
59:                     return succ;
60:             }
61:
62:             public TreeNode find(TreeNode root, int d)
63:             {
64:                     if (root == null)
65:                             return null;
66:
67:                     if (root.data == d)
68:                             return root;
69:
70:                     TreeNode temp;
71:
72:                     temp = find(root.left, d);
73:                     if (temp != null)
74:                             return temp;
75:                     temp = find(root.right, d);
76:                     return temp;
77:
78:             }
79:
80:             public TreeNode insert(TreeNode root, int d)
81:             {
82:                     return null;
83:             }
84:
85:             public void printPaths()
86:             {
87:                     int[] path = new int[height(root)];
88:                     printPaths(root, path, 0);
89:             }
90:
91:             private void printPaths(TreeNode node, int[] path, int level)
92:             {
93:                     if (node == null)
94:                             return;
95:
96:                     path[level] = node.data;
```

```
97:                     // System.out.println("level: " + level);
98:
99:                     if (node.left == null && node.right == null)
100:                    {
101:                            printArray(path, level);
102:                    }
103:                    else
104:                    {
105:                            printPaths(node.left, path, level + 1);
106:                            printPaths(node.right, path, level + 1);
107:                    }
108:            }
109:
110:            private void printArray(int[] ints, int level)
111:            {
112:                    for (int i = 0; i <= level; i++)
113:                            System.out.print(ints[i] + " ");
114:                    System.out.println();
115:            }
116:
117:            public boolean BFS(TreeNode root, int d)
118:            {
119:                    if (root == null)
120:                            return false;
121:
122:                    if (root.data == d)
123:                            return true;
124:
125:                    Queue<TreeNode> q = new LinkedList<TreeNode>();
126:                    TreeNode temp;
127:                    q.offer(root);
128:
129:                    while (!q.isEmpty())
130:                    {
131:                            temp = q.poll();
132:                            if (temp != null)
133:                            {
134:                                    if (temp.data == d)
135:                                            return true;
136:                                    if (temp.left != null)
137:                                            q.offer(temp.left);
138:                                    if (temp.right != null)
139:                                            q.offer(temp.right);
140:                            }
141:                    }
142:                    return false;
143:            }
144:
```

```
145:         public boolean DFS(TreeNode root, int d)
146:         {
147:                 if (root == null)
148:                         return false;
149:
150:                 if (root.data == d)
151:                         return true;
152:
153:                 return (DFS(root.left, d) || DFS(root.right, d));
154:         }
155:
156:         public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
157:         {
158:                 if (p == null || q == null)
159:                         return null;
160:
161:                 if (root == null)
162:                         return null;
163:
164:                 if (root == p || root == q)
165:                         return root;
166:
167:                 TreeNode leftLCA = lowestCommonAncestor(root.left, p, q);
168:                 TreeNode rightLCA = lowestCommonAncestor(root.right, p, q);
169:
170:                 if (leftLCA != null && rightLCA != null)
171:                         return root;
172:
173:                 return (leftLCA == null) ? rightLCA : leftLCA;
174:         }
175:
176:         private boolean isBST(TreeNode root, int min, int max)
177:         {
178:                 if (root == null)
179:                         return true;
180:
181:                 if (root.data <= min || root.data > max)
182:                         return false;
183:
184:                 return isBST(root.left, min, root.data) && isBST(root.right, root.data, max);
185:         }
186:
187:         public boolean isBST(TreeNode root)
188:         {
189:                 return isBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
190:         }
191: }
```

```
 1: package problems;
 2:
 3: import java.util.Scanner;
 4:
 5: public class BitFlip
 6: {
 7:         static int findWindow(int[] bin)
 8:         {
 9:                 int start = 0, end = 0;
10:                 int len = bin.length, max = 0, numZeros = 0;
11:                 int globalstart = 0;
12:                 for (int i = 0; i < len; i++)
13:                 {
14:                         if (numZeros == 0 && bin[i] == 1)
15:                         {
16:                                 start = i + 1;
17:                                 continue;
18:                         }
19:                         numZeros = bin[i] == 0 ? numZeros + 1 : numZeros - 1;
20:                         if (max < numZeros)
21:                         {
22:                                 max = numZeros;
23:                                 end = i;
24:                                 globalstart = start;
25:                         }
26:                 }
27:                 if (globalstart == len)
28:                         return len;
29:
30:                 int i = 0, res = 0;
31:                 while (i < globalstart)
32:                 {
33:                         if (bin[i++] == 1)
34:                                 res++;
35:                 }
36:
37:                 while (i <= end)
38:                 {
39:                         if (bin[i++] == 0)
40:                                 res++;
41:                 }
42:
43:                 while (i < len)
44:                 {
45:                         if (bin[i++] == 1)
46:                                 res++;
47:                 }
48:
```

```
49:                     return res;
50:             }
51:
52:             public static void main(String[] args)
53:             {
54:                     Scanner sc = new Scanner(System.in);
55:                     int n = sc.nextInt();
56:                     int bin[] = new int[n];
57:                     for (int i = 0; i < n; i++)
58:                             bin[i] = sc.nextInt();
59:                     System.out.println(findWindow(bin));
60:                     sc.close();
61:             }
62:
63: }
```

```java
package problems;

import java.util.*;

public class BiValuedSlice
{
        public static int solution(int[] A)
        {
                int idx = 0, numCount = 0;
                Set<Integer> visited = new HashSet<Integer>();

                while (idx < A.length)
                {
                        int tmp = 0;
                        for (int i = idx; i < A.length; i++)
                        {
                                visited.add(A[i]);
                                if (visited.size() == 3)
                                {
                                        visited.clear();
                                        if (tmp > numCount)
                                                numCount = tmp;
                                        break;
                                }
                                tmp++;
                        }
                        idx++;
                }
                return numCount;
        }

        public static void main(String[] args)
        {
        }

}
```

```
 1: package problems;
 2:
 3: public class BST extends BinaryTree
 4: {
 5:         public BST()
 6:         {
 7:                 root = null;
 8:         }
 9:
10:         @Override
11:         public TreeNode find(TreeNode root, int d)
12:         {
13:                 if (root == null)
14:                         return null;
15:
16:                 while (root != null)
17:                 {
18:                         if (root.data == d)
19:                                 return root;
20:
21:                         if (d < root.data)
22:                                 root = root.left;
23:                         else
24:                                 root = root.right;
25:                 }
26:
27:                 return null;
28:         }
29:
30:         @Override
31:         public TreeNode insert(TreeNode root, int d)
32:         {
33:                 if (root == null)
34:                         return new TreeNode(d);
35:
36:                 if (d < root.data)
37:                         root.left = insert(root.left, d);
38:
39:                 else
40:                         root.right = insert(root.right, d);
41:
42:                 return root;
43:         }
44:
45:         private int minVal(TreeNode root)
46:         {
47:                 TreeNode node = root;
48:                 while (node.left != null)
```

```
49:                         node = node.left;
50:                 return node.data;
51:         }
52:
53:         public TreeNode deleteNode(TreeNode root, int key)
54:         {
55:                 if (root == null)
56:                         return null;
57:
58:                 if (root.data > key)
59:                         root.left = deleteNode(root.left, key);
60:                 else if (root.data < key)
61:                         root.right = deleteNode(root.right, key);
62:                 else
63:                 {
64:                         if (root.right == null)
65:                                 return root.left;
66:                         else if (root.left == null)
67:                                 return root.right;
68:
69:                         root.data = minVal(root.right);
70:                         root.right = deleteNode(root.right, root.data);
71:                 }
72:                 return root;
73:         }
74:
75:         @Override
76:         public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
77:         {
78:                 if (root == null || p == null || q == null)
79:                         return null;
80:
81:                 if (root.data < p.data && root.data < q.data)
82:                         return lowestCommonAncestor(root.right, p, q);
83:                 if (root.data > p.data && root.data > q.data)
84:                         return lowestCommonAncestor(root.left, p, q);
85:                 return root;
86:         }
87:
88:         public static void main(String[] args)
89:         {
90:                 BinaryTree tree = new BST();
91:                 tree.root = tree.insert(tree.root, 10);
92:                 tree.root = tree.insert(tree.root, 6);
93:                 tree.root = tree.insert(tree.root, 8);
94:                 tree.root = tree.insert(tree.root, 14);
95:                 tree.root = tree.insert(tree.root, 16);
96:                 tree.root = tree.insert(tree.root, 9);
```

```
   97:                 tree.root = tree.insert(tree.root, 5);
   98:
   99:                 TreeNode temp = tree.find(tree.root, 9);
  100:                 System.out.println("temp: " + ((temp == null) ? "null" : temp.data));
  101:
  102:                 System.out.println(tree.inOrderSuccessor(tree.root, temp).data);
  103:
  104:                 System.out.println(tree.BFS(tree.root, 7));
  105:
  106:                 System.out.println(tree.DFS(tree.root, 7));
  107:
  108:                 System.out.println(tree.isBST(tree.root));
  109:
  110:         }
  111: }
```

```
    1: package problems;
    2:
    3: import java.util.Arrays;
    4:
    5: public class CanPartition
    6: {
    7:         static boolean canPartition(int[] nums)
    8:         {
    9:                 if (nums == null || nums.length < 2)
   10:                         return false;
   11:
   12:                 int target = 0;
   13:                 for (int i : nums)
   14:                         target += i;
   15:
   16:                 if ((target & 1) == 1)
   17:                         return false;
   18:
   19:                 target /= 2;
   20:                 boolean[] part = new boolean[target + 1];
   21:                 part[0] = true;
   22:                 for (int i = 0; i < nums.length; i++)
   23:                 {
   24:                         for (int j = target; j >= nums[i]; j--)
   25:                                 part[j] = part[j] || part[j - nums[i]];
   26:                         System.out.println(Arrays.toString(part));
   27:                 }
   28:                 return part[target];
   29:         }
   30:
   31:         public static void main(String[] args)
   32:         {
   33:                 int[] a = { 4, 6, 8 };
   34:                 System.out.println(canPartition(a));
   35:         }
   36:
   37: }
```

```java
package problems;

import java.util.Arrays;

public class ClimbSteps
{
	public static long countWays(int s, long arr[])
	{
		if (s < 0)
			return 0;
		if (s == 0)
			return 1;
		if (arr[s] >= 0)
			return arr[s];
		// since we count no of ways there is no "1 + prev value" logic.
		// it is only for finding min no: of steps
		arr[s] = countWays(s - 1, arr) + countWays(s - 2, arr) + countWays(s - 3, arr);
		return arr[s];
	}

	public static int countWaysSlow(int s)
	{
		if (s < 0)
			return 0;
		if (s == 0)
			return 1;
		return countWaysSlow(s - 1) + countWaysSlow(s - 2) + countWaysSlow(s - 3);
	}

	public static void main(String[] args)
	{
		long[] arr = new long[37];
		Arrays.fill(arr, -1);
		System.out.println(countWays(4, arr));
		// System.out.println(countWaysSlow(36));
	}
}
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class CoinChange
 6: {
 7:         static int countWays(int[] coins, int n)
 8:         {
 9:                 int[] ways = new int[n + 1];
10:                 ways[0] = 1;
11:
12:                 for (int i = 0; i < coins.length; i++)
13:                 {
14:                         // for every new coin that you see update the array with new ways
15:                         // only starting from that new coin since lesser denoms can't be
16:                         // obtained.
17:                         for (int j = coins[i]; j <= n; j++)
18:                         {
19:                                 ways[j] += ways[j - coins[i]];
20:                                 System.out.println(Arrays.toString(ways));
21:                         }
22:                 }
23:
24:                 System.out.println(Arrays.toString(ways));
25:                 return ways[n];
26:         }
27:
28:         public static void main(String[] args)
29:         {
30:                 int[] coins = { 4, 2, 3 };
31:                 System.out.println("No: of ways: " + countWays(coins, 7));
32:         }
33: }
```

```java
package problems;

import java.util.*;

public class CommonMin
{

        public static int solution(int[] A, int[] B)
        {
                int result = Integer.MAX_VALUE;
                Set<Integer> hashSet = new HashSet<Integer>();
                for (int num : A)
                        hashSet.add(num);

                for (int num : B)
                {
                        if (!hashSet.contains(num))
                                continue;
                        if (num < result)
                                result = num;
                }

                return (result == Integer.MAX_VALUE) ? -1 : result;
        }

        public static void main(String[] args)
        {
                int a = Integer.MAX_VALUE, b = Integer.MAX_VALUE;
                int c = Math.addExact(a, b);
                System.out.println(c);
        }

}
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Arrays;
 5: import java.util.List;
 6:
 7: public class ConfusingFibbonaci
 8: {
 9:         public static void main(String[] args)
10:         {
11:                 List<Integer> l = new ArrayList<Integer>();
12:                 l.add(0);
13:                 l.add(1);
14:                 int n = l.size();
15:
16:                 while (n < 8)
17:                 {
18:                         n = l.size();
19:                         l.add(l.get(n - 1) + l.get(n - 2));
20:                         n++;
21:                 }
22:                 System.out.println(Arrays.toString(l.toArray()));
23:                 System.out.println(l.get(l.size() - 1));
24:         }
25: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class CountingSort
 6: {
 7:         static void sort(char[] a)
 8:         {
 9:                 int n = a.length;
10:                 char output[] = new char[n];
11:
12:                 int freq[] = new int[26];
13:                 for (int i = 0; i < n; ++i)
14:                         freq[a[i] - 'a']++;
15:
16:                 for (int i = 1; i < 26; i++)
17:                         freq[i] += freq[i - 1];
18:
19:                 for (int i = n - 1; i >= 0; --i)
20:                 {
21:                         output[freq[a[i] - 'a'] - 1] = a[i];
22:                         freq[a[i] - 'a']--;
23:                 }
24:
25:                 for (int i = 0; i < n; ++i)
26:                         a[i] = output[i];
27:         }
28:
29:         public static void main(String[] args)
30:         {
31:                 char a[] = { 'a', 'n', 'a', 'n', 'd', 'k', 'u', 'm', 'a', 'r' };
32:                 sort(a);
33:                 System.out.print(Arrays.toString(a));
34:         }
35:
36: }
```

```java
package problems;

public class CountIslands
{
	static int[] rowNum = { -1, -1, -1, 0, 0, 1, 1, 1 };
	static int[] colNum = { 0, -1, 1, -1, 1, -1, 1, 0 };

	static void merge(char[][] grid, int i, int j)
	{
		int m = grid.length, n = grid[0].length;

		if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] != '1')
			return;

		grid[i][j] = 'X';

		for (int a = 0; a < rowNum.length; a++)
			merge(grid, i + rowNum[a], j + colNum[a]);
	}

	public static int numIslands(char[][] grid)
	{
		if (grid == null || grid.length == 0 || grid[0].length == 0)
			return 0;

		int m = grid.length, n = grid[0].length, count = 0;
		for (int i = 0; i < m; i++)
		{
			for (int j = 0; j < n; j++)
			{
				if (grid[i][j] == '1')
				{
					count++;
					merge(grid, i, j);
				}
			}
		}
		return count;
	}

	public static void main(String[] args)
	{
		char[][] grid = { { '1', '1', '0', '0', '0' }, { '0', '1', '0', '0', '1' }, { '1', '0', '0', '1', '1' },
				{ '0', '0', '0', '0', '0' }, { '1', '0', '1', '0', '1' } };
		System.out.println(numIslands(grid));
	}

}
```

```java
package problems;

public class CountOccurencesSortedArray
{
	public static int countOccurences(int[] a, int x)
	{
		int i = firstOccurence(a, x);
		System.out.println("first: " + i);
		if (i == -1)
			return 0;
		int j = lastOccurence(a, i, x);
		System.out.println("last: " + j);
		return j - i + 1;
	}

	// go left if a[mid] is equal to search element
	private static int firstOccurence(int a[], int x)
	{
		int start = 0, end = a.length - 1;
		while (start <= end)
		{
			int mid = (start + end) / 2;
			if ((mid == 0 || a[mid - 1] < x) && a[mid] == x)
				return mid;
			else if (a[mid] >= x)
				end = mid - 1;
			else
				start = mid + 1;
		}
		return -1;
	}

	// go right if a[mid] is equal to search element
	private static int lastOccurence(int a[], int i, int x)
	{
		int start = i, end = a.length - 1;
		while (start <= end)
		{
			int mid = (start + end) / 2;
			if ((mid == a.length - 1 || a[mid + 1] > x) && a[mid] == x)
				return mid;
			else if (a[mid] > x)
				end = mid - 1;
			else
				start = mid + 1;
		}
		return -1;
	}
```

```
49:
50:         public static void main(String[] args)
51:         {
52:                 int[] a = { 2, 2, 2, 2, 2, 7, 8, 9 };
53:                 System.out.println(countOccurences(a, 2));
54:         }
55: }
```

```
 1: package problems;
 2:
 3: public class DecimalValue
 4: {
 5:         public static void main(String[] args)
 6:         {
 7:                 int[][] input = { { 0, 1, 0 }, { 1, 1, 0 }, { 0, 0, 1 } };
 8:                 int max = 0;
 9:                 for (int i = 0; i < input.length; i++)
10:                 {
11:                         int tmp = 0;
12:                         for (int j = input[i].length - 1; j >= 0; j--)
13:                         {
14:                                 tmp |= (input[i][j] << (input[i].length - 1 - j));
15:                                 System.out.println(tmp);
16:                         }
17:                         if (tmp > max)
18:                                 max = tmp;
19:                 }
20:                 System.out.println("Max: " + max);
21:         }
22: }
```

```
 1: package problems;
 2:
 3: public class DecipherMsg
 4: {
 5:         public static void main(String args[])
 6:         {
 7:                 // String cipher = "jussDs sfsfs fwfsldfms Atvt hrqgse, Cnikg";
 8:                 String cipher = "Li, ailu jw au facntll";
 9:                 String plain = decipher(cipher);
10:                 System.out.println(plain);
11:         }
12:
13:         private static String decipher(String encrypted_message)
14:         {
15:                 StringBuilder plain = new StringBuilder("");
16:                 String key = "8251220";
17:                 int keycounter = 0;
18:                 for (int i = 0; i < encrypted_message.length(); i++)
19:                 {
20:                         char temp = encrypted_message.charAt(i);
21:                         if (keycounter == key.length())
22:                                 keycounter = 0;
23:                         if (temp >= 65 && temp <= 90)
24:                         {
25:                                 char plainChar = (char) (encrypted_message.charAt(i)
26:                                                 - Character.getNumericValue(key.charAt(keycounter)));
27:                                 if (plainChar < 65)
28:                                 {
29:                                         int diff = 65 - plainChar;
30:                                         plainChar = (char) (91 - diff);
31:                                 }
32:                                 System.out.println(encrypted_message.charAt(i));
33:                                 System.out.println(key.charAt(keycounter));
34:                                 System.out.println(plainChar);
35:                                 System.out.println();
36:                                 plain.append(plainChar);
37:                                 keycounter++;
38:                         }
39:                         else if (temp >= 97 && temp <= 122)
40:                         {
41:                                 char plainChar = (char) (encrypted_message.charAt(i)
42:                                                 - Character.getNumericValue(key.charAt(keycounter)));
43:                                 if (plainChar < 97)
44:                                 {
45:                                         int diff = 97 - plainChar;
46:                                         plainChar = (char) (123 - diff);
47:                                 }
48:                                 System.out.println(encrypted_message.charAt(i));
```

```
49:                                     System.out.println(key.charAt(keycounter));
50:                                     System.out.println(plainChar);
51:                                     System.out.println();
52:                                     plain.append(plainChar);
53:                                     keycounter++;
54:                             }
55:                             else
56:                             {
57:                                     plain.append(temp);
58:                             }
59:                     }
60:                     return plain.toString();
61:             }
62: }
```

```
 1: package problems;
 2:
 3: public class EquilibriumIndex
 4: {
 5:         public static int solution(int[] a)
 6:         {
 7:                 long sumL = 0, sumR = 0;
 8:                 int len = a.length;
 9:                 for (int i = 0; i < len; i++)
10:                         sumR += a[i];
11:                 for (int i = 0; i < len; i++)
12:                 {
13:                         sumR -= a[i];
14:                         if (sumL == sumR)
15:                                 return i;
16:                         sumL += a[i];
17:                 }
18:                 return -1;
19:         }
20:
21:         public static void main(String[] args)
22:         {
23:                 int[] a = { -1, 3, -4, 5, 1, -6, 2, 1 };
24:                 System.out.println(solution(a));
25:
26:         }
27:
28: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.HashSet;
 5: import java.util.List;
 6: import java.util.Scanner;
 7: import java.util.Set;
 8:
 9: class Edge
10: {
11:         int dest;
12:         int weight;
13:
14:         public Edge(int d, int w)
15:         {
16:                 dest = d;
17:                 weight = w;
18:         }
19: }
20:
21: class DG
22: {
23:         int v;
24:         List<Edge>[] adj;
25:         static int freq[];
26:         static boolean vis[];
27:
28:         @SuppressWarnings("unchecked")
29:         public DG(int v)
30:         {
31:                 this.v = v;
32:                 adj = new ArrayList[v + 1];
33:                 for (int i = 1; i <= v; i++)
34:                         adj[i] = new ArrayList<Edge>();
35:                 freq = new int[10];
36:                 vis = new boolean[v + 1];
37:         }
38:
39:         public void addEdge(int a, int b, int w)
40:         {
41:                 Edge e = new Edge(b, w);
42:                 adj[a].add(e);
43:                 e = new Edge(a, 1000 - w);
44:                 adj[b].add(e);
45:         }
46:
47:         public void findAllPaths()
48:         {
```

```
49:                 for (int i = 1; i <= v; i++)
50:                 {
51:                         vis = new boolean[v + 1];
52:                         Set<Integer> proc = new HashSet<Integer>();
53:                         findAllPaths(i, i, 0, proc);
54:                 }
55:                 for (int i = 0; i < 10; i++)
56:                         System.out.println(freq[i]);
57:         }
58:
59:         public void findAllPaths(int src, int curr, int w, Set<Integer> proc)
60:         {
61:                 vis[curr] = true;
62:                 for (Edge e : adj[curr])
63:                 {
64:                         if (e.dest == src || e.dest == curr || proc.contains(e.dest))
65:                                 continue;
66:                         // System.out.println(w + e.weight);
67:                         freq[(w + e.weight) % 10]++;
68:                         proc.add(e.dest);
69:                         if (!vis[e.dest])
70:                                 findAllPaths(src, e.dest, w + e.weight, proc);
71:                         if (curr == src)
72:                         {
73:                                 proc.clear();
74:                                 vis = new boolean[v + 1];
75:                         }
76:                 }
77:         }
78: }
79:
80: public class FindAllPathsDG
81: {
82:         public static void main(String[] args)
83:         {
84:                 Scanner in = new Scanner(System.in);
85:                 int n = in.nextInt();
86:                 DG g = new DG(n);
87:                 int e = in.nextInt();
88:                 for (int a0 = 0; a0 < e; a0++)
89:                 {
90:                         int x = in.nextInt();
91:                         int y = in.nextInt();
92:                         int r = in.nextInt();
93:                         g.addEdge(x, y, r);
94:                 }
95:                 g.findAllPaths();
96:                 in.close();
```

```
97:         }
98: }
```

```java
package problems;

import java.util.ArrayList;
import java.util.List;

public class FindAnagrams
{
	static List<Integer> findAnagrams(String s, String p)
	{
		List<Integer> list = new ArrayList<>();
		if (s == null || s.length() == 0 || p == null || p.length() == 0)
			return list;

		int[] freq = new int[256];

		for (char c : p.toCharArray())
			freq[c]++;

		int left = 0, right = 0, count = p.length();
		while (right < s.length())
		{
			if (freq[s.charAt(right)] >= 1)
				count--;
			freq[s.charAt(right)]--;
			right++;

			if (count == 0)
				list.add(left);

			if (right - left == p.length())
			{
				// if we are throwing away a valid char, increase the count
				// a char is valid if it occurs in p
				if (freq[s.charAt(left)] >= 0)
					count++;
				// restore the count of left most char before throwing it away
				freq[s.charAt(left)]++;
				left++;
			}
		}
		return list;
	}

	public static void main(String[] args)
	{
		// System.out.println(findAnagrams("abab", "ab"));
		System.out.println(findAnagrams("caababaaaa", "aaba"));
		// System.out.println(findAnagrams("bbbbcacaa", "aaba"));
```

```
49:         }
50:
51: }
```

```java
package problems;

public class FindDuplicate
{
	static int findDuplicate(int[] nums)
	{
		int l = 1, r = nums.length - 1;

		while (l < r)
		{
			int m = l + (r - l) / 2;
			int count = 0;

			for (int a : nums)
			{
				if (a <= m)
					count++;
			}

			// System.out.println("mid: " + m + " count: " + count);
			if (count > m)
				r = m;
			else
				l = m + 1;
		}
		return l;
	}

	static int findDuplicateFast(int[] nums)
	{
		/*
		 * since the array contains numbers only between [1..N], the array will
		 * have atleast one cycle. starting at 0 is the key. since the array has
		 * no 0, the cycle we encounter starting at 0 will not be based on the
		 * number at 0th index, i.e. it will definitely contain the duplicate
		 * element, which will be the starting node of cycle.
		 */
		int slow = nums[0], fast = nums[slow];

		while (slow != fast)
		{
			slow = nums[slow];
			fast = nums[nums[fast]];
		}

		slow = 0;
		while (fast != slow)
		{
```

```
49:                         fast = nums[fast];
50:                         slow = nums[slow];
51:                 }
52:                 return slow;
53:         }
54:
55:         public static void main(String args[])
56:         {
57:                 int a[] = { 4, 1, 3, 2, 6, 7, 5, 1 };
58:                 System.out.println(findDuplicateFast(a));
59:         }
60:
61: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.HashSet;
 5: import java.util.List;
 6: import java.util.Set;
 7:
 8: public class FindMutation
 9: {
10:         public static void main(String args[])
11:         {
12:                 String bank[] = { "AAAAAAAA", "AAAAAAAT", "AAAAAATT", "AAAAATTT" };
13:                 String start = "AAAAAAAA";
14:                 String end = "AAAAATTT";
15:                 int dist = minMutation(start, end, bank);
16:                 System.out.println(dist);
17:         }
18:
19:         public static int minMutation(String start, String end, String[] bank)
20:         {
21:                 if (bank == null || start == null || start.length() == 0 || end == null || end.length() == 0)
22:                         return 0;
23:
24:                 Set<String> bankStrings = new HashSet<String>();
25:                 for (String s : bank)
26:                         bankStrings.add(s);
27:
28:                 if (!bankStrings.contains(end))
29:                         return -1;
30:
31:                 int count = minMutation(start, end, bankStrings);
32:                 return count == 0 ? -1 : count;
33:         }
34:
35:         public static int minMutation(String start, String end, Set<String> bankStrings)
36:         {
37:                 List<Integer> indices = new ArrayList<>();
38:                 int length = start.length();
39:                 for (int i = 0; i < length; ++i)
40:                 {
41:                         if (start.charAt(i) != end.charAt(i))
42:                                 indices.add(i);
43:                 }
44:                 int curCount = 0;
45:                 String temp = "";
46:                 for (int i : indices)
47:                 {
48:                         temp = start.substring(0, i) + "" + end.charAt(i) + "" + start.substring(i + 1, length);
```

```
49:                         if (bankStrings.contains(temp))
50:                                 curCount = 1 + minMutation(temp, end, bankStrings);
51:                 }
52:                 return curCount;
53:         }
54: }
```

```java
package problems;

import java.util.Arrays;

public class Frog
{
	public static int minSecondsToCross(int A[], int X, int D)
	{
		if (D >= X)
			return 0;

		int finalMin = -1, localMin = A.length, localStart = 0, finalPos = 0, loopIdx = 0;
		int[] times = new int[X];
		Arrays.fill(times, -1);

		// Storing the seconds in an array indexed by position of the leaf
		for (; loopIdx < A.length; loopIdx++)
		{
			// if multiples leaves fall in same position, consider only the leaf
			// that fell first
			if (times[A[loopIdx]] >= 0)
				continue;

			times[A[loopIdx]] = loopIdx;
		}

		/*
		 * Idea: 1. Split the pond into equal gaps of width 'D' The frog can
		 * jump only if there is at least one leaf in all these gaps. 2. Find
		 * minimum times at which a leaf fall in each of these gaps 3. Maximum
		 * of the set of minimum times found in step 2 would be the required
		 * answer.
		 */
		for (loopIdx = 0; loopIdx < times.length; loopIdx++)
		{
			if ((localStart + D + 1) == loopIdx)
			{
				localStart = loopIdx - 1; // mark the start of each gap

				if (finalMin < localMin)
					finalMin = localMin;

				localMin = A.length; // reset localMin at the start of each gap
			}

			if (times[loopIdx] < 0)
				continue; // continue if there is no leaf at this position

```

```
49:                         if (finalPos == 0 && loopIdx > D) // no leaf to start-off, so return
50:                                                           // -1
51:                                 return -1;
52:
53:                         if (localMin > times[loopIdx])
54:                                 localMin = times[loopIdx];
55:
56:                         if (loopIdx > finalPos && loopIdx - finalPos <= D)
57:                         {
58:                                 finalPos = loopIdx;
59:                                 if (finalPos + D >= X)
60:                                         break;
61:                         }
62:
63:                 }
64:
65:                 if (localMin < A.length && finalMin < localMin)
66:                         finalMin = localMin;
67:
68:                 if (finalPos + D >= X)
69:                         return finalMin;
70:
71:                 return -1;
72:         }
73:
74:         public static void main(String[] args)
75:         {
76:                 int X = 4, D = 2; // X is diameter of the pond. D is the max distance
77:                                   // the frog can caver in a single jump
78:                 int A[] = { 2, 2, 2, 2, 2 }; // A[k] denotes the position at which a
79:                                              // leaf fall in kth second
80:
81:                 // Given the above parameters, find the minimum seconds in which the
82:                 // frog can jump across the pond
83:                 int ans = minSecondsToCross(A, X, D);
84:                 if (ans < 0)
85:                 {
86:                         System.out.println("Frog can't cross the pond");
87:                 }
88:                 else
89:                 {
90:                         System.out.println("Frog can cross the pond in " + ans + " seconds");
91:                 }
92:         }
93: }
```

```java
package problems;

import java.util.LinkedList;
import java.util.List;

@SuppressWarnings("unchecked")
public class Graph
{
	int vCount;
	List<Integer>[] adj;

	public Graph()
	{}

	public Graph(int v)
	{
		vCount = v;
		adj = new LinkedList[v];
		for (int i = 0; i < v; i++)
			adj[i] = new LinkedList<Integer>();
	}

	public void addEdge(int a, int b)
	{
		adj[a].add(b);
		adj[b].add(a);
	}
}

class DirectedGraph extends Graph
{

	public DirectedGraph(int v)
	{
		super(v);
	}

	@Override
	public void addEdge(int a, int b)
	{
		adj[a].add(b);
	}

}
```

```
 1: package problems;
 2:
 3: import java.util.Iterator;
 4: import java.util.LinkedList;
 5: import java.util.Scanner;
 6:
 7: public class GraphBFS
 8: {
 9:         public static void printReach(Graph g, int v, int st)
10:         {
11:                 int reach[] = new int[v + 1];
12:                 boolean visited[] = new boolean[v + 1];
13:                 LinkedList<Integer> q = new LinkedList<Integer>();
14:                 q.offer(st);
15:                 visited[st] = true;
16:                 int level = 1, currNodeCount = 1, newNodeCount = 0;
17:                 while (!q.isEmpty())
18:                 {
19:                         int s = q.poll();
20:                         currNodeCount--;
21:                         Iterator<Integer> it = g.adj[s].iterator();
22:                         while (it.hasNext())
23:                         {
24:                                 int n = it.next();
25:                                 if (!visited[n])
26:                                 {
27:                                         reach[n] += (6 * level);
28:                                         visited[n] = true;
29:                                         q.offer(n);
30:                                         newNodeCount++;
31:                                 }
32:                         }
33:                         if (currNodeCount == 0)
34:                         {
35:                                 level++;
36:                                 currNodeCount = newNodeCount;
37:                                 newNodeCount = 0;
38:                         }
39:                 }
40:                 for (int i = 1; i <= v; i++)
41:                 {
42:                         if ((i != st))
43:                         {
44:                                 if (reach[i] != 0)
45:                                         System.out.print(reach[i] + " ");
46:                                 else if (reach[i] == 0)
47:                                         System.out.print("-1 ");
48:                         }
```

```
49:                 }
50:                 System.out.println();
51:         }
52:
53:         public static void main(String[] args)
54:         {
55:                 Scanner sc = new Scanner(System.in);
56:                 int q = sc.nextInt();
57:                 for (int i = 0; i < q; i++)
58:                 {
59:                         int v = sc.nextInt();
60:                         Graph g = new Graph(v);
61:                         int e = sc.nextInt();
62:                         for (int j = 0; j < e; j++)
63:                                 g.addEdge(sc.nextInt(), sc.nextInt());
64:                         int st = sc.nextInt();
65:                         printReach(g, v, st);
66:                 }
67:                 sc.close();
68:         }
69: }
```

```
 1: package problems;
 2:
 3: public class GraphDFS
 4: {
 5:         static void DFSUtil(DirectedGraph g, int v, boolean[] visited)
 6:         {
 7:                 visited[v] = true;
 8:                 System.out.print(v + " ");
 9:                 for (int neighbor : g.adj[v])
10:                 {
11:                         if (!visited[neighbor])
12:                                 DFSUtil(g, neighbor, visited);
13:                 }
14:         }
15:
16:         static void DFS(DirectedGraph g, int v)
17:         {
18:                 boolean visited[] = new boolean[v];
19:                 for (int i = 0; i < v; i++)
20:                 {
21:                         if (!visited[i])
22:                                 DFSUtil(g, i, visited);
23:                 }
24:         }
25:
26:         static void DFS(DirectedGraph g, int v, int n)
27:         {
28:                 boolean visited[] = new boolean[v];
29:                 if (!visited[n])
30:                         DFSUtil(g, n, visited);
31:         }
32:
33:         public static void main(String[] args)
34:         {
35:                 DirectedGraph g = new DirectedGraph(6);
36:                 g.addEdge(5, 2);
37:                 g.addEdge(5, 0);
38:                 g.addEdge(4, 0);
39:                 g.addEdge(4, 1);
40:                 g.addEdge(2, 3);
41:                 g.addEdge(3, 1);
42:
43:                 System.out.println("Depth First Traversal");
44:                 DFS(g, 6);
45:                 System.out.println();
46:                 System.out.println("Depth First Traversal from 5: ");
47:                 DFS(g, 6, 5);
48:         }
```

```
49:
50: }
```

```
 1: package problems;
 2:
 3: import java.util.Scanner;
 4:
 5: public class HomeValue
 6: {
 7:         /*
 8:          * diff denotes # of increasing subranges - # of decreasing subranges in a
 9:          * window of given size
10:          */
11:         static long diff = 0;
12:
13:         /*
14:          * enum Direction is to denote if the subrange is increasing or decreasing
15:          * or none (i.e. when all elements of subarray are equal)
16:          */
17:         enum Direction
18:         {
19:                 NONE, DECR, INCR;
20:         }
21:
22:         static int updateDiff(int start, int end, Direction dir)
23:         {
24:                 /*
25:                  * Variable subArrays will hold # of inc/dec subranges between start and
26:                  * end index. It's calculated based on the formula that a set of size N
27:                  * has N(N+1)/2 subarrays. But here we don't need subsets of size 1.
28:                  * Thus N(N-1)/2 subsets.
29:                  */
30:                 long size = end - start + 1;
31:                 long subArrays = (size * (size - 1)) / 2;
32:
33:                 /* if the prev subrange is positive then add subArrays to diff */
34:                 if (dir == Direction.INCR)
35:                         diff += subArrays;
36:                 /* if the prev subrange is negative then subtract subArrays from diff */
37:                 else if (dir == Direction.DECR)
38:                         diff -= subArrays;
39:
40:                 /* return the next start position */
41:                 return end;
42:         }
43:
44:         public static void calcWindowDiff(int[] a, int k)
45:         {
46:                 if (a == null || a.length == 0 || k <= 0 || k > a.length)
47:                 {
48:                         System.out.println("Invalid Input!");
```

```
49:                     return;
50:             }
51:
52:             int left = 0, right = k - 1, n = a.length;
53:
54:             /* start denotes start of an inc/dec subrange */
55:             int start = 0;
56:
57:             Direction dir = Direction.NONE;
58:
59:             for (; right < n; right++, left++)
60:             {
61:                     start = left;
62:                     diff = 0;
63:                     for (int i = left; i < right; i++)
64:                     {
65:                             if (a[i] < a[i + 1])
66:                             {
67:                                     /*
68:                                      * if subrange starts increasing then update diff with # of
69:                                      * decreasing subranges seen so far
70:                                      */
71:                                     if (dir == Direction.DECR)
72:                                             start = updateDiff(start, i, dir);
73:                                     dir = Direction.INCR;
74:                             }
75:                             else if (a[i] > a[i + 1])
76:                             {
77:                                     /*
78:                                      * if subrange starts decreasing then update diff with # of
79:                                      * increasing subranges seen so far
80:                                      */
81:                                     if (dir == Direction.INCR)
82:                                             start = updateDiff(start, i, dir);
83:                                     dir = Direction.DECR;
84:                             }
85:                             else
86:                             {
87:                                     /*
88:                                      * if numbers in subrange are equal then update diff with #
89:                                      * of decreasing/increasing subranges seen so far
90:                                      */
91:                                     start = updateDiff(start, i, dir) + 1;
92:                                     dir = Direction.NONE;
93:                             }
94:
95:                             /*
96:                              * if we reach the end of a window then update diff with # of
```

```
 97:                                     * decreasing/increasing subranges seen so far
 98:                                     */
 99:                                    if (i == right - 1)
100:                                    {
101:                                            updateDiff(start, right, dir);
102:                                            dir = Direction.NONE;
103:                                    }
104:                            }
105:                            System.out.println(diff);
106:                    }
107:            }
108:
109:            public static void main(String[] args)
110:            {
111:                    Scanner input = new Scanner(System.in);
112:                    int n = input.nextInt();
113:                    int k = input.nextInt();
114:                    int a[] = new int[n];
115:                    for (int i = 0; i < n; i++)
116:                            a[i] = input.nextInt();
117:                    calcWindowDiff(a, k);
118:                    input.close();
119:            }
120:
121: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4: import java.util.List;
 5: import java.util.stream.Collectors;
 6:
 7: public class JavaStream
 8: {
 9:         public static void main(String args[])
10:         {
11:                 List<Integer> myList = Arrays.asList(1, 2, 3, 4, 5, 6);
12:                 List<Integer> list = myList.parallelStream().map(a -> a + 1).collect(Collectors.toList());
13:                 System.out.println(Arrays.toString(list.toArray()));
14:         }
15: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Comparator;
 5: import java.util.List;
 6: import java.util.PriorityQueue;
 7:
 8: /* class Point implements Comparable<Point>
 9: {
10:         double x, y;
11:
12:         public Point()
13:         {
14:                 x = 0;
15:                 y = 0;
16:         }
17:
18:         public Point(double a, double b)
19:         {
20:                 x = a;
21:                 y = b;
22:         }
23:
24:         public double distFromOrigin()
25:         {
26:                 return Math.hypot(x, y);
27:         }
28:
29:         @Override
30:         public int compareTo(Point o)
31:         {
32:                 if (this.distFromOrigin() < o.distFromOrigin())
33:                         return 1;
34:                 else if (this.distFromOrigin() > o.distFromOrigin())
35:                         return -1;
36:                 return 0;
37:         }
38: } */
39:
40: class Point
41: {
42:         double x, y;
43:
44:         public Point()
45:         {
46:                 x = 0;
47:                 y = 0;
48:         }
```

```
49:
50:         public Point(double a, double b)
51:         {
52:                 x = a;
53:                 y = b;
54:         }
55:
56:         public double distFromOrigin()
57:         {
58:                 return Math.hypot(x, y);
59:         }
60: }
61:
62: public class KNearestPoints
63: {
64:         static void findKNearestPoints(List<Point> points, int k)
65:         {
66:                 PriorityQueue<Point> maxHeap = new PriorityQueue<Point>(k, new Comparator<Point>()
67:                 {
68:                         @Override
69:                         public int compare(Point o1, Point o2)
70:                         {
71:                                 if (o1.distFromOrigin() > o2.distFromOrigin())
72:                                         return -1;
73:                                 if (o1.distFromOrigin() < o2.distFromOrigin())
74:                                         return 1;
75:                                 return 0;
76:                         }
77:                 });
78:                 for (Point p : points)
79:                 {
80:                         if (maxHeap.size() < k)
81:                         {
82:                                 maxHeap.offer(p);
83:                         }
84:                         else
85:                         {
86:                                 if (maxHeap.peek().distFromOrigin() > p.distFromOrigin())
87:                                 {
88:                                         maxHeap.poll();
89:                                         maxHeap.offer(p);
90:                                 }
91:                         }
92:                 }
93:                 while (maxHeap.size() > 0)
94:                 {
95:                         Point p = maxHeap.poll();
96:                         System.out.println(p.x + "," + p.y);
```

```
97:                     }
98:             }
99:
100:            public static void main(String[] args)
101:            {
102:                    List<Point> points = new ArrayList<Point>();
103:                    points.add(new Point(1, 1));
104:                    points.add(new Point(1, 2.5));
105:                    points.add(new Point(-1, 1.4));
106:                    points.add(new Point(-1, 2));
107:                    points.add(new Point(1.5, -1.5));
108:                    points.add(new Point(1.6, -1));
109:                    points.add(new Point(-1, -1.5));
110:                    points.add(new Point(-1, 3));
111:                    points.add(new Point(2, 2));
112:                    findKNearestPoints(points, 5);
113:            }
114: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: class KGraph
 6: {
 7:         class KEdge implements Comparable<KEdge>
 8:         {
 9:                 int src, dst, weight;
10:
11:                 @Override
12:                 public int compareTo(KEdge that)
13:                 {
14:                         return this.weight - that.weight;
15:                 }
16:         }
17:
18:         class KSubset
19:         {
20:                 int parent, rank;
21:         }
22:
23:         int V, E;
24:         KEdge[] edge;
25:
26:         public KGraph(int v, int e)
27:         {
28:                 V = v;
29:                 E = e;
30:                 edge = new KEdge[E];
31:                 for (int i = 0; i < E; i++)
32:                         edge[i] = new KEdge();
33:         }
34:
35:         public int find(KSubset[] subset, int i)
36:         {
37:                 if (subset[i].parent != i)
38:                         subset[i].parent = find(subset, subset[i].parent);
39:                 return subset[i].parent;
40:         }
41:
42:         public void union(KSubset[] subset, int x, int y)
43:         {
44:                 int xParent = find(subset, x);
45:                 int yParent = find(subset, y);
46:
47:                 if (subset[xParent].rank < subset[yParent].rank)
48:                 {
```

```
49:                         subset[xParent].parent = yParent;
50:                 }
51:                 else if (subset[yParent].rank < subset[xParent].rank)
52:                 {
53:                         subset[yParent].parent = xParent;
54:                 }
55:                 else
56:                 {
57:                         subset[xParent].parent = yParent;
58:                         subset[yParent].rank++;
59:                 }
60:         }
61:
62:         public void kruskalMST()
63:         {
64:                 int i = 0;
65:                 int e = 0;
66:                 KEdge mst[] = new KEdge[E];
67:                 for (i = 0; i < E; i++)
68:                         mst[i] = new KEdge();
69:
70:                 Arrays.sort(edge);
71:
72:                 KSubset subset[] = new KSubset[V];
73:                 for (i = 0; i < V; i++)
74:                 {
75:                         subset[i] = new KSubset();
76:                         subset[i].parent = i;
77:                         subset[i].rank = 0;
78:                 }
79:                 i = 0;
80:                 while (e < V - 1)
81:                 {
82:                         KEdge curr_edge = edge[i++];
83:
84:                         int x = find(subset, curr_edge.src);
85:                         int y = find(subset, curr_edge.dst);
86:                         if (x != y)
87:                         {
88:                                 mst[e++] = curr_edge;
89:                                 union(subset, x, y);
90:                         }
91:                 }
92:
93:                 for (i = 0; i < E; i++)
94:                 {
95:                         System.out.println("Edge " + mst[i].src + "-->" + mst[i].dst + " (" + mst[i].weight + ")");
96:                 }
```

```
 97:         }
 98: }
 99:
100: public class KruskalMST
101: {
102:         public static void main(String[] args)
103:         {
104:                 int V = 4; // Number of vertices in graph
105:                 int E = 5; // Number of edges in graph
106:                 KGraph graph = new KGraph(V, E);
107:
108:                 // add edge 0-1
109:                 graph.edge[0].src = 0;
110:                 graph.edge[0].dst = 1;
111:                 graph.edge[0].weight = 10;
112:
113:                 // add edge 0-2
114:                 graph.edge[1].src = 0;
115:                 graph.edge[1].dst = 2;
116:                 graph.edge[1].weight = 6;
117:
118:                 // add edge 0-3
119:                 graph.edge[2].src = 0;
120:                 graph.edge[2].dst = 3;
121:                 graph.edge[2].weight = 5;
122:
123:                 // add edge 1-3
124:                 graph.edge[3].src = 1;
125:                 graph.edge[3].dst = 3;
126:                 graph.edge[3].weight = 15;
127:
128:                 // add edge 2-3
129:                 graph.edge[4].src = 2;
130:                 graph.edge[4].dst = 3;
131:                 graph.edge[4].weight = 4;
132:
133:                 graph.kruskalMST();
134:         }
135: }
```

```
 1: package problems;
 2:
 3: import java.util.LinkedList;
 4: import java.util.Queue;
 5:
 6: public class LargestSubTree
 7: {
 8:         static int getLeftMostChild(TreeNode root)
 9:         {
10:                 TreeNode temp = root;
11:                 while (temp.left != null)
12:                         temp = temp.left;
13:                 return temp.data;
14:         }
15:
16:         static int getRightMostChild(TreeNode root)
17:         {
18:                 TreeNode temp = root;
19:                 while (temp.right != null)
20:                         temp = temp.right;
21:                 return temp.data;
22:         }
23:
24:         static TreeNode getLargeRoot(TreeNode root, int i, int j)
25:         {
26:                 if (root == null)
27:                         return null;
28:                 Queue<TreeNode> q = new LinkedList<TreeNode>();
29:                 q.offer(root);
30:                 while (!q.isEmpty())
31:                 {
32:                         TreeNode curr = q.poll();
33:                         if (curr.data >= i && curr.data <= j)
34:                         {
35:                                 int lMost = getLeftMostChild(curr);
36:                                 int rMost = getRightMostChild(curr);
37:                                 if (lMost >= i && rMost <= j)
38:                                         return curr;
39:                         }
40:                         if (curr.left != null)
41:                                 q.offer(curr.left);
42:                         if (curr.right != null)
43:                                 q.offer(curr.right);
44:                 }
45:                 return null;
46:         }
47:
48:         public static void main(String[] args)
```

```
49:         {
50:                 BinaryTree tree = new BST();
51:                 tree.root = tree.insert(tree.root, 10);
52:                 tree.root = tree.insert(tree.root, 6);
53:                 tree.root = tree.insert(tree.root, 8);
54:                 tree.root = tree.insert(tree.root, 14);
55:                 tree.root = tree.insert(tree.root, 16);
56:                 tree.root = tree.insert(tree.root, 9);
57:                 tree.root = tree.insert(tree.root, 5);
58:                 tree.root = tree.insert(tree.root, 12);
59:                 tree.root = tree.insert(tree.root, 7);
60:
61:                 TreeNode largeRoot = getLargeRoot(tree.root, 5, 6);
62:                 if (largeRoot != null)
63:                         System.out.println(largeRoot.data);
64:                 else
65:                         System.out.println("No such tree is possible...");
66:         }
67: }
```

```
 1: package problems;
 2:
 3: import java.util.HashMap;
 4: import java.util.Map;
 5:
 6: public class LinkedListRandPtr
 7: {
 8:         public RandomListNode copyRandomList(RandomListNode head)
 9:         {
10:                 if (head == null)
11:                         return null;
12:
13:                 Map<RandomListNode, RandomListNode> map = new HashMap<RandomListNode, RandomListNode>();
14:                 RandomListNode temp = head;
15:
16:                 while (temp != null)
17:                 {
18:                         RandomListNode node = new RandomListNode(temp.label);
19:                         map.put(temp, node);
20:                         temp = temp.next;
21:                 }
22:
23:                 RandomListNode cpy_temp = null;
24:                 temp = head;
25:                 while (temp != null)
26:                 {
27:                         cpy_temp = map.get(temp);
28:                         cpy_temp.next = map.get(temp.next);
29:                         cpy_temp.random = map.get(temp.random);
30:                         temp = temp.next;
31:                 }
32:
33:                 return map.get(head);
34:         }
35:
36:         public static void main(String[] args)
37:         {
38:
39:         }
40:
41: }
```

```
 1: package problems;
 2:
 3: public class LinkedListToBST
 4: {
 5:         static TreeNode convertSLLtoBST(MyLinkedList<Integer> list)
 6:         {
 7:                 if (list.head == null)
 8:                         return null;
 9:
10:                 return recurSLLtoBST(list, 0, list.length() - 1);
11:                 // return recurSLLtoBST(list, list.length());
12:
13:         }
14:
15:         static TreeNode recurSLLtoBST(MyLinkedList<Integer> list, int low, int high)
16:         {
17:                 if (low > high)
18:                         return null;
19:
20:                 int mid = (low + high) / 2;
21:                 TreeNode left = recurSLLtoBST(list, low, mid - 1);
22:                 TreeNode root = new TreeNode(list.head.data);
23:                 root.left = left;
24:                 list.head = list.head.next;
25:                 root.right = recurSLLtoBST(list, mid + 1, high);
26:                 return root;
27:         }
28:
29:         public static void main(String[] args)
30:         {
31:                 MyLinkedList<Integer> list = new MyLinkedList<Integer>();
32:                 list.insertAtEnd(1);
33:                 list.insertAtEnd(2);
34:                 list.insertAtEnd(3);
35:                 list.insertAtEnd(4);
36:                 list.insertAtEnd(5);
37:                 list.insertAtEnd(6);
38:                 list.insertAtEnd(7);
39:                 list.insertAtEnd(8);
40:                 list.printList();
41:
42:                 TreeNode root = convertSLLtoBST(list);
43:                 BinaryTree.printInOrder(root);
44:         }
45:
46: }
```

```
    1: package problems;
    2:
    3: import java.io.File;
    4: import java.io.FileNotFoundException;
    5: import java.util.HashMap;
    6: import java.util.Map;
    7: import java.util.Scanner;
    8:
    9: public class LongestDirectoryPath
   10: {
   11:         public static int longestDirPath(String s)
   12:         {
   13:                 Map<Integer, Integer> dirMap = new HashMap<Integer, Integer>();
   14:                 int maxLen = 0;
   15:                 for (String entry : s.split("\n"))
   16:                 {
   17:                         if (entry.isEmpty())
   18:                                 continue;
   19:
   20:                         int currLevel = 0, currLen = 0;
   21:                         String dirEntry = entry.replaceAll("^\\s+", "");
   22:                         int len = dirEntry.length();
   23:                         currLevel = entry.length() - len;
   24:                         if (dirEntry.indexOf('.') == -1)
   25:                         {
   26:                                 currLen = ((currLevel == 0) ? 0 : (dirMap.get(currLevel - 1) + 1)) + len;
   27:                                 dirMap.put(currLevel, currLen);
   28:                         }
   29:                         else
   30:                         {
   31:                                 maxLen = Math.max(maxLen,
   32:                                                 (dirMap.containsKey(currLevel - 1) ? (dirMap.get(currLevel - 1) + 1) + len
: len));
   33:                         }
   34:                 }
   35:                 return maxLen;
   36:         }
   37:
   38:         public static void main(String[] args)
   39:         {
   40:                 Scanner sc = null;
   41:                 try
   42:                 {
   43:                         sc = new Scanner(new File("/Users/Anand/Documents/input.txt"));
   44:                         StringBuilder sb = new StringBuilder();
   45:                         while (sc.hasNext())
   46:                         {
   47:                                 sb.append(sc.nextLine());
```

```
48:                                     sb.append(System.lineSeparator());
49:                             }
50:                             System.out.println(sb.toString());
51:                             System.out.println(longestDirPath(sb.toString()));
52:                     }
53:                     catch (FileNotFoundException e)
54:                     {
55:                             System.out.println("Error reading input: " + e.getMessage());
56:                     }
57:                     finally
58:                     {
59:                             sc.close();
60:                     }
61:             }
62: }
```

```java
package problems;

public class LongestPalindromicChunks
{
	static int longestPalindrome(String s)
	{
		if (s.length() == 0)
			return 0;

		int inpLen = s.length();
		int start = 0, end = inpLen, chunkCount = 0, matchedLen = 0;

		for (int i = 1; i <= inpLen / 2; i++)
		{
			if (s.substring(start, i).equals(s.substring(inpLen - i, end)))
			{
				chunkCount += 2;
				int len = s.substring(start, i).length();
				matchedLen += (2 * len);
				start += len;
				end -= len;
			}
		}
		if (matchedLen < inpLen)
			chunkCount++;
		return chunkCount;
	}

	public static void main(String args[])
	{
		System.out.println(longestPalindrome("antaprezatepzapreanta"));
		System.out.println(longestPalindrome("merchant"));
		System.out.println(longestPalindrome("volvo"));
		System.out.println(longestPalindrome("ghiabcdefhelloadamhelloabcdefghi"));

	}
}
```

```
 1: package problems;
 2:
 3: public class LongestPalindromicSubstring
 4: {
 5:         public static String longestPalindrome(String s)
 6:         {
 7:                 int len = s.length();
 8:                 if (len == 1)
 9:                         return s;
10:                 String longest = "";
11:                 for (int i = 0; i < len; i++)
12:                 {
13:                         String tmp = getPalindrome(s, i, i);
14:                         if (tmp.length() > longest.length())
15:                                 longest = tmp;
16:
17:                         tmp = getPalindrome(s, i, i + 1);
18:                         if (tmp.length() > longest.length())
19:                                 longest = tmp;
20:                 }
21:                 return longest;
22:         }
23:
24:         private static String getPalindrome(String s, int start, int end)
25:         {
26:                 while (start >= 0 && end < s.length() && (s.charAt(start) == s.charAt(end)))
27:                 {
28:                         start--;
29:                         end++;
30:                 }
31:                 return s.substring(start + 1, end);
32:         }
33:
34:         public static void main(String[] args)
35:         {
36:                 String s = "456789zazasxabcdeedcba123";
37:                 System.out.println(longestPalindrome(s));
38:         }
39: }
```

```
 1: package problems;
 2:
 3: import java.util.HashSet;
 4: import java.util.Set;
 5:
 6: public class LongestSubstring2Unique
 7: {
 8:
 9:         static String findLongestSubstring(String input)
10:         {
11:                 if (input == null || input.length() == 0)
12:                         return input;
13:                 int len = input.length();
14:                 int globalStart = 0, localStart = 0;
15:                 int maxLen = Integer.MIN_VALUE, currLen = 0;
16:                 char lastChar = 0, lastCharCount = 0;
17:                 /*
18:                  * HashSet stores 2 unique characters seen so far
19:                  */
20:                 Set<Character> uniqueChars = new HashSet<Character>();
21:                 for (int i = 0; i < len; i++)
22:                 {
23:                         char c = input.charAt(i);
24:                         /*
25:                          * if the HashSet contains the current character, then increase the
26:                          * running length.
27:                          */
28:                         if (uniqueChars.contains(c))
29:                         {
30:                                 currLen++;
31:                         }
32:                         else
33:                         {
34:                                 /*
35:                                  * if the current character is a new character, then add it to
36:                                  * HashSet.
37:                                  */
38:                                 if (uniqueChars.size() < 2)
39:                                 {
40:                                         uniqueChars.add(c);
41:                                         currLen++;
42:                                 }
43:                                 /*
44:                                  * If the new character is 3rd unique character, then update the
45:                                  * HashSet, globalStart and maxLen accordingly.
46:                                  */
47:                                 else
48:                                 {
```

```
49:                                         uniqueChars.clear();
50:                                         uniqueChars.add(lastChar);
51:                                         uniqueChars.add(c);
52:                                         if (maxLen < currLen)
53:                                         {
54:                                                 maxLen = currLen;
55:                                                 globalStart = localStart;
56:                                                 currLen = lastCharCount + 1;
57:                                                 localStart = i - lastCharCount;
58:                                         }
59:                                 }
60:                         }
61:                         if (c != lastChar)
62:                         {
63:                                 lastChar = c;
64:                                 lastCharCount = 0;
65:                         }
66:                         lastCharCount++;
67:                 }
68:                 /*
69:                  * include the character at end of the string to our running length and
70:                  * global length if needed.
71:                  */
72:                 if (maxLen < currLen)
73:                 {
74:                         maxLen = currLen;
75:                         globalStart = localStart;
76:                 }
77:                 return input.substring(globalStart, globalStart + maxLen);
78:         }
79:
80:         public static void main(String[] args)
81:         {
82:                 System.out.println(findLongestSubstring("aabaaaccacccadef"));
83:         }
84:
85: }
```

```
 1: package problems;
 2:
 3: import java.util.HashMap;
 4: import java.util.Map;
 5:
 6: class Node
 7: {
 8:         int key;
 9:         int value;
10:         Node pre;
11:         Node next;
12:
13:         public Node(int key, int value)
14:         {
15:                 this.key = key;
16:                 this.value = value;
17:         }
18: }
19:
20: public class LRUCache
21: {
22:         int capacity;
23:         Map<Integer, Node> map = new HashMap<Integer, Node>();
24:         Node head = null;
25:         Node end = null;
26:
27:         public LRUCache(int capacity)
28:         {
29:                 this.capacity = capacity;
30:         }
31:
32:         public int get(int key)
33:         {
34:                 if (map.containsKey(key))
35:                 {
36:                         Node n = map.get(key);
37:                         remove(n);
38:                         setHead(n);
39:                         return n.value;
40:                 }
41:
42:                 return -1;
43:         }
44:
45:         public void remove(Node n)
46:         {
47:                 if (n.pre != null)
48:                 {
```

```
49:                         n.pre.next = n.next;
50:                 }
51:                 else
52:                 {
53:                         head = n.next;
54:                 }
55:
56:                 if (n.next != null)
57:                 {
58:                         n.next.pre = n.pre;
59:                 }
60:                 else
61:                 {
62:                         end = n.pre;
63:                 }
64:
65:         }
66:
67:         public void setHead(Node n)
68:         {
69:                 n.next = head;
70:                 n.pre = null;
71:
72:                 if (head != null)
73:                         head.pre = n;
74:
75:                 head = n;
76:
77:                 if (end == null)
78:                         end = head;
79:         }
80:
81:         public void set(int key, int value)
82:         {
83:                 if (map.containsKey(key))
84:                 {
85:                         Node old = map.get(key);
86:                         old.value = value;
87:                         remove(old);
88:                         setHead(old);
89:                 }
90:                 else
91:                 {
92:                         Node created = new Node(key, value);
93:                         if (map.size() >= capacity)
94:                         {
95:                                 map.remove(end.key);
96:                                 remove(end);
```

```
   97:                                     setHead(created);
   98:
   99:                             }
  100:                             else
  101:                             {
  102:                                     setHead(created);
  103:                             }
  104:
  105:                             map.put(key, created);
  106:                     }
  107:             }
  108: }
```

```java
package problems;

import java.util.Arrays;

public class MagicIndex
{
	/*
	 * provide different solution for Dup and NoDup variations. NoDup solution
	 * is very fast O(logn). Dup solution will work fast only for arrays with
	 * duplicates. For non-dup arrays, it is equivalent to linear search.
	 */
	// static int i = 1;

	static int magicIndex(int[] a, int low, int high)
	{
		if (low > high)
			return -1;

		// System.out.println(i++ + ": " + low + " " + high);

		int mid = low + (high - low) / 2;
		if (a[mid] == mid)
			return mid;

		int lIndex = magicIndex(a, low, Math.min(a[mid], mid - 1));
		if (lIndex >= 0)
			return lIndex;

		int rIndex = magicIndex(a, Math.max(a[mid], mid + 1), high);
		return rIndex;
	}

	static int magicIndexNoDup(int[] a, int low, int high)
	{
		while (low <= high)
		{
			// System.out.println(i++ + ": " + low + " " + high);

			int mid = low + (high - low) / 2;
			if (a[mid] == mid)
				return mid;
			else if (a[mid] < mid)
				low = mid + 1;
			else
				high = mid - 1;
		}
		return -1;
	}
```

```
49:
50:         public static void main(String[] args)
51:         {
52:                 int a[] = { -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13 };
53:                 System.out.println(Arrays.toString(a));
54:                 System.out.println(magicIndex(a, 0, a.length - 1));
55:                 // i = 0;
56:                 System.out.println(magicIndexNoDup(a, 0, a.length - 1));
57:         }
58: }
```

```
 1: package problems;
 2:
 3: import java.text.DateFormat;
 4: import java.text.ParseException;
 5: import java.text.SimpleDateFormat;
 6: import java.util.Calendar;
 7: import java.util.Date;
 8: import java.util.GregorianCalendar;
 9:
10: public class MagicTime
11: {
12:         public static boolean IsMagicNumber(long num)
13:         {
14:                 boolean[] isPresent = new boolean[10];
15:                 int count = 0;
16:                 while (num > 0)
17:                 {
18:                         int digit = (int) (num % 10);
19:                         if (!isPresent[digit])
20:                         {
21:                                 isPresent[digit] = true;
22:                                 count++;
23:                         }
24:                         num /= 10;
25:                 }
26:                 return count == 2 ? true : false;
27:         }
28:
29:         public static void printMagicTime(String a, String b) throws ParseException
30:         {
31:                 DateFormat df = new SimpleDateFormat("yyyy/MM/dd HH:mm");
32:                 Date d1 = df.parse(a);
33:                 Date d2 = df.parse(b);
34:                 df = new SimpleDateFormat("yyyyMMddHHmm");
35:                 Calendar cal = new GregorianCalendar();
36:                 cal.setTime(d1);
37:                 while (cal.getTime().before(d2))
38:                 {
39:                         long tempDate = Long.parseLong(df.format(cal.getTime()));
40:                         if (IsMagicNumber(tempDate))
41:                                 System.out.println(cal.getTime());
42:                         cal.add(Calendar.MINUTE, 1);
43:                 }
44:         }
45:
46:         public static void main(String[] args)
47:         {
48:                 try
```

```
49:                     {
50:                             printMagicTime("1100/10/05 12:07", "1110/01/20 12:09");
51:                     }
52:                     catch (ParseException e)
53:                     {
54:                             e.printStackTrace();
55:                     }
56:             }
57:
58: }
```

```
 1: package problems;
 2:
 3: import java.util.*;
 4: import java.util.LinkedList;;
 5:
 6: class FNode
 7: {
 8:
 9:         public String data;
10:         public FNode left = null;
11:         public FNode right = null;
12:
13:         public FNode(String d)
14:         {
15:                 data = d;
16:         }
17: }
18:
19: public class Main
20: {
21:         FNode root;
22:
23:         public Main()
24:         {
25:                 root = null;
26:         }
27:
28:         public FNode find(FNode root, String d)
29:         {
30:                 if (root == null)
31:                         return null;
32:
33:                 if (root.data.equals(d))
34:                         return root;
35:
36:                 FNode temp;
37:
38:                 temp = find(root.left,d);
39:                 if(temp != null)
40:                         return temp;
41:                 temp = find(root.right,d);
42:                 return temp;
43:         }
44:
45:         public FNode insert(FNode root, String parent, String d)
46:         {
47:                 FNode newNode = new FNode(d);
48:                 if(root == null)
```

```
49:                 {
50:                         FNode newParent = new FNode(parent);
51:                         root = newParent;
52:                         root.left = newNode;
53:                         return root;
54:                 }
55:
56:                 FNode temp = find(root, parent);
57:
58:                 if(temp.left == null)
59:                         temp.left = newNode;
60:                 else
61:                         temp.right = newNode;
62:
63:                 return root;
64:         }
65:
66:         public void BFS(FNode root, String d)
67:         {
68:                 if(root == null) return;
69:
70:                 Queue<FNode> q = new LinkedList<FNode>();
71:                 Set<String> level = new TreeSet<String>();
72:                 int currLevel = 0, nextLevel = 0;
73:                 FNode temp = root;
74:
75:                 q.offer(temp);
76:                 currLevel++;
77:                 level.add(temp.data);
78:
79:                 if(level.contains(d))
80:                 {
81:                         printSiblings(level);
82:                         return;
83:                 }
84:
85:                 level.clear();
86:                 while(!q.isEmpty())
87:                 {
88:                         temp = q.poll();
89:                         currLevel--;
90:                         if(temp.left != null)
91:                         {
92:                                 q.offer(temp.left);
93:                                 nextLevel++;
94:                                 level.add(temp.left.data);
95:                         }
96:                         if(temp.right != null)
```

```
97:                             {
98:                                     q.offer(temp.right);
99:                                     nextLevel++;
100:                                    level.add(temp.right.data);
101:                            }
102:                            if(currLevel == 0)
103:                            {
104:                                    currLevel = nextLevel;
105:                                    nextLevel = 0;
106:                                    if(level.contains(d))
107:                                    {
108:                                            printSiblings(level);
109:                                            return;
110:                                    }
111:                                    level.clear();
112:                            }
113:                    }
114:            }
115:
116:            public void printSiblings(Set<String> set)
117:            {
118:                    StringBuilder sb = new StringBuilder();
119:                    for(String a : set)
120:                            sb.append(a + ",");
121:                    sb.setLength(sb.length()-1);
122:                    System.out.println(sb.toString());
123:            }
124:
125:            public static void main(String args[])
126:            {
127:                    Scanner sc = new Scanner(System.in);
128:                    Main tree = new Main();
129:                    String s = "";
130:                    while(sc.hasNext())
131:                    {
132:                            s = sc.nextLine();
133:                            System.out.println(s);
134:                            String[] s1 = s.split(",");
135:                            String person = s1[s1.length-1];
136:                            for(int i=0;i<s1.length-1;i++)
137:                            {
138:                                    String parent = s1[i].split("->")[0];
139:                                    String child = s1[i].split("->")[1];
140:                                    tree.root = tree.insert(tree.root, parent, child);
141:                            }
142:                            tree.BFS(tree.root,person);
143:                    }
144:                    sc.close();
```

```
145:         }
146: }
```

```
 1: package problems;
 2:
 3: public class MaxHeap
 4: {
 5:         private int[] Heap;
 6:         private int size;
 7:         private int maxsize;
 8:
 9:         private static final int FRONT = 1;
10:
11:         public MaxHeap(int maxsize)
12:         {
13:                 this.maxsize = maxsize;
14:                 this.size = 0;
15:                 Heap = new int[this.maxsize + 1];
16:                 Heap[0] = Integer.MAX_VALUE;
17:         }
18:
19:         private int parent(int pos)
20:         {
21:                 return pos / 2;
22:         }
23:
24:         private int leftChild(int pos)
25:         {
26:                 return (2 * pos);
27:         }
28:
29:         private boolean isLeaf(int pos)
30:         {
31:                 if (pos > (size / 2) && pos <= size)
32:                 {
33:                         return true;
34:                 }
35:                 return false;
36:         }
37:
38:         private void swap(int fpos, int spos)
39:         {
40:                 int tmp;
41:                 tmp = Heap[fpos];
42:                 Heap[fpos] = Heap[spos];
43:                 Heap[spos] = tmp;
44:         }
45:
46:         private void maxHeapify(int pos)
47:         {
48:                 while (!isLeaf(pos))
```

```
49:                 {
50:                         int newPos = leftChild(pos);
51:                         if ((newPos < size) && Heap[newPos + 1] > Heap[newPos])
52:                                 newPos++; // move to right child
53:                         if (Heap[newPos] < Heap[pos])
54:                                 break;
55:                         swap(pos, newPos);
56:                         pos = newPos;
57:                 }
58:         }
59:
60:         public void insert(int element)
61:         {
62:                 Heap[++size] = element;
63:                 int current = size;
64:                 while (Heap[current] > Heap[parent(current)])
65:                 {
66:                         swap(current, parent(current));
67:                         current = parent(current);
68:                 }
69:         }
70:
71:         public void maxHeap() // this function is just to form a maxheap from
72:                                                         // existing array
73:         {
74:                 for (int pos = (size / 2); pos >= 1; pos--)
75:                         maxHeapify(pos);
76:         }
77:
78:         public int remove()
79:         {
80:                 int popped = Heap[FRONT];
81:                 if (size == FRONT)
82:                 {
83:                         size--;
84:                 }
85:                 else
86:                 {
87:                         Heap[FRONT] = Heap[size--];
88:                         maxHeapify(FRONT);
89:                 }
90:                 return popped;
91:         }
92:
93:         public void removeAll()
94:         {
95:                 while (size > 0)
96:                         System.out.print(remove() + " ");
```

```
   97:                   System.out.println();
   98:           }
   99:
  100:           public static void main(String... arg)
  101:           {
  102:                   System.out.println("The Max Heap is ");
  103:                   MaxHeap maxHeap = new MaxHeap(10);
  104:                   maxHeap.insert(5);
  105:                   maxHeap.insert(3);
  106:                   maxHeap.insert(17);
  107:                   maxHeap.insert(10);
  108:                   maxHeap.insert(84);
  109:                   maxHeap.insert(19);
  110:                   maxHeap.insert(6);
  111:                   maxHeap.insert(22);
  112:                   maxHeap.insert(9);
  113:                   maxHeap.insert(100);
  114:
  115:                   maxHeap.removeAll();
  116:           }
  117: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayDeque;
 4: import java.util.Deque;
 5:
 6: public class MaxInSlidingWindow
 7: {
 8:         public static void printMaxInSlidingWindow(int[] a, int n, int k)
 9:         {
10:                 if (a == null || a.length == 0)
11:                         return;
12:
13:                 Deque<Integer> deq = new ArrayDeque<Integer>();
14:                 int idx = 0;
15:                 for (; idx < k; idx++)
16:                 {
17:                         while (!deq.isEmpty() && a[idx] >= a[deq.peekLast()])
18:                                 deq.pollLast();
19:                         deq.offer(idx);
20:                 }
21:
22:                 for (; idx < n; idx++)
23:                 {
24:                         System.out.print(a[deq.peek()] + " ");
25:
26:                         // Reason for storing indices in deque:
27:                         // since index is used here to delete old elements from window.
28:                         if (!deq.isEmpty() && deq.peek() <= (idx - k))
29:                                 deq.poll();
30:
31:                         while (!deq.isEmpty() && a[idx] >= a[deq.peekLast()])
32:                                 deq.pollLast();
33:                         deq.offer(idx);
34:                 }
35:                 if (!deq.isEmpty())
36:                         System.out.print(a[deq.peek()]);
37:         }
38:
39:         public static void understandDeque(int[] a, int n)
40:         {
41:                 Deque<Integer> deq = new ArrayDeque<Integer>();
42:                 for (int i = 0; i < n; i++)
43:                 {
44:                         deq.offer(a[i]);
45:                         System.out.println(deq);
46:                 }
47:                 System.out.println("peekFirst: " + deq.peekFirst());
48:                 System.out.println("peek: " + deq.peek());
```

```
49:                 System.out.println("peekLast: " + deq.peekLast());
50:                 System.out.println("poll: " + deq.poll());
51:                 System.out.println("pollLast: " + deq.pollLast());
52:                 System.out.println(deq);
53:                 deq.offer(100);
54:                 System.out.println(deq);
55:         }
56:
57:         public static void main(String[] args)
58:         {
59:                 // Scanner sc = new Scanner(System.in);
60:                 // int n = sc.nextInt();
61:                 // int k = sc.nextInt();
62:                 // int[] a = new int[n];
63:                 int[] a = { 8, 5, 10, 7, 9, 4, 15, 12, 90, 13 };
64:                 // for (int i = 0; i < n; i++)
65:                 // a[i] = sc.nextInt();
66:                 printMaxInSlidingWindow(a, 10, 3);
67:                 // understandDeque(a, 10);
68:                 // sc.close();
69:         }
70:
71: }
```

```java
package problems;

import java.util.TreeSet;

public class MaxModSumSubarray
{
	static void maxModSumSubarray(long[] a, int m)
	{
		TreeSet<Long> s = new TreeSet<Long>();
		long sum = 0, ans = -1, n = a.length;
		for (int i = 0; i < n; i++)
		{
			if (i == 0)
			{
				sum = a[i] % m;
				ans = Math.max(ans, sum);
				s.add(sum);
			}
			else
			{
				sum = (sum + (a[i] % m)) % m;
				Long temp = s.higher(sum);
				ans = Math.max(ans, (sum - ((temp == null) ? 0 : temp) + m) % m);
				s.add(sum);
			}
		}
		System.out.println(ans);
	}

	public static void main(String[] args)
	{
		long a[] = { 3, 3, 9, 9, 5 };
		maxModSumSubarray(a, 7);
	}
}
```

```java
package problems;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class MaxSet
{
	public static boolean AddToSet(int[] A, int[] B, int i, int temp)
	{
		if (B[i] > 0)
			return false;
		B[i] = temp++;
		boolean ret = AddToSet(A, B, A[i], temp);
		return ret;
	}

	public static int solution(int A[], int N)
	{
		if (A.length == 0)
			return 0;
		int[] B = new int[N];
		Set<Integer> set = new HashSet<Integer>();
		int totalSize = 0;
		for (int i = 0; i < N; i++)
		{
			if (B[i] > 0)
				continue;

			while (AddToSet(A, B, i, 1));
			System.out.println(set);
			System.out.println(Arrays.toString(B));

			totalSize += set.size();
			System.out.println(totalSize);
			if (totalSize >= N)
				break;
			set.clear();
		}

		int max = -1;
		for (int i = 0; i < B.length; i++)
		{
			if (B[i] > max)
				max = B[i];
		}
		return max;
	}
```

```
49:
50:         public static void main(String[] args)
51:         {
52:                 int A[] = { 5, 4, 1, 0, 3, 6, 2 };
53:                 // int A[] = {5,4,0,3,1,6,2};
54:                 // int A[] = {0,1,2,3,4,5,6};
55:                 System.out.println(solution(A, A.length));
56:         }
57: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class MaxSumSubarray
 6: {
 7:         static int maxSumSubSeq(int[] nums)
 8:         {
 9:                 int globalMax = nums[0];
10:                 int len = nums.length;
11:                 for (int i = 1; i < len; i++)
12:                         globalMax = Math.max(nums[i], Math.max(globalMax, globalMax + nums[i]));
13:                 return globalMax;
14:         }
15:
16:         static int maxSumSubArray(int[] nums)
17:         {
18:                 int localMax = nums[0];
19:                 int globalMax = nums[0];
20:                 int len = nums.length;
21:                 for (int i = 1; i < len; i++)
22:                 {
23:                         localMax = Math.max(nums[i], localMax + nums[i]);
24:                         globalMax = Math.max(globalMax, localMax);
25:                 }
26:                 return globalMax;
27:         }
28:
29:         static void maxSubArray(int[] nums)
30:         {
31:                 int localMax = nums[0];
32:                 int globalMax = nums[0];
33:                 int localStart = 0, globalStart = 0, globalEnd = 0;
34:                 int len = nums.length;
35:                 for (int i = 1; i < len; i++)
36:                 {
37:                         if (localMax + nums[i] > nums[i])
38:                         {
39:                                 localMax += nums[i];
40:                         }
41:                         else
42:                         {
43:                                 localMax = nums[i];
44:                                 localStart = i;
45:                         }
46:
47:                         if (globalMax < localMax)
48:                         {
```

```
49:                                 globalMax = localMax;
50:                                 globalStart = localStart;
51:                                 globalEnd = i;
52:                             }
53:                     }
54:                     System.out.println("start: " + globalStart + " end: " + globalEnd);
55:             }
56:
57:             public static void main(String[] args)
58:             {
59:                     int[] a = { -2, 1, -3, 4, -1, 2, 1, -9, 4 };
60:                     System.out.println(Arrays.toString(a));
61:                     System.out.println("Max sum of subarray: " + maxSumSubArray(a));
62:                     maxSubArray(a);
63:                     System.out.println("Max sum of subsequence: " + maxSumSubSeq(a));
64:             }
65: }
```

```java
package problems;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class MaxUniqueNumsSlidingWindow
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		Deque<Integer> deque = new ArrayDeque<Integer>();
		Map<Integer, Integer> map = new HashMap<Integer, Integer>();
		int n = sc.nextInt();
		int m = sc.nextInt();
		int max = 0;

		for (int i = 0; i < n; i++)
		{
			int num = sc.nextInt();
			deque.offer(num);
			if (map.containsKey(num))
				map.put(num, map.get(num) + 1);
			else
				map.put(num, 1);

			if (deque.size() > m)
			{
				int head = deque.poll();
				if (map.get(head) > 1)
					map.put(head, map.get(num) - 1);
				else
					map.remove(head);
			}
			max = Math.max(max, map.size());
		}
		System.out.println(max);
		sc.close();
	}
}
```

```
 1: package problems;
 2:
 3: public class MaxValueSubTree
 4: {
 5:         static void decorateTree(TreeNode root)
 6:         {
 7:                 if (root == null)
 8:                         return;
 9:
10:                 decorateTree(root.left);
11:                 decorateTree(root.right);
12:
13:                 if (root.left != null)
14:                         root.data = Math.max(root.data, root.left.data);
15:                 if (root.right != null)
16:                         root.data = Math.max(root.data, root.right.data);
17:         }
18:
19:         public static void main(String[] args)
20:         {
21:                 BinaryTree tree = new BST();
22:                 tree.root = tree.insert(tree.root, 10);
23:                 tree.root = tree.insert(tree.root, 6);
24:                 tree.root = tree.insert(tree.root, 8);
25:                 tree.root = tree.insert(tree.root, 14);
26:                 tree.root = tree.insert(tree.root, 12);
27:                 tree.root = tree.insert(tree.root, 16);
28:                 tree.root = tree.insert(tree.root, 9);
29:                 tree.root = tree.insert(tree.root, 5);
30:                 tree.printPaths();
31:                 System.out.println();
32:                 decorateTree(tree.root);
33:                 tree.printPaths();
34:         }
35: }
```

```java
package problems;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class Interval
{
	int start;
	int end;

	Interval()
	{
		start = 0;
		end = 0;
	}

	Interval(int s, int e)
	{
		start = s;
		end = e;
	}
}

public class MergeIntervals
{
	public List<Interval> merge(List<Interval> intervals)
	{
		List<Interval> ans = new ArrayList<Interval>();

		if (intervals == null || intervals.size() == 0)
			return ans;

		Collections.sort(intervals, new Comparator<Interval>()
		{
			@Override
			public int compare(Interval i1, Interval i2)
			{
				if (i1.start != i2.start)
					return i1.start - i2.start;
				else
					return i1.end - i2.end;
			}
		});

		Interval prev = intervals.get(0);
		for (int i = 1; i < intervals.size(); i++)
```

```
49:                 {
50:                     Interval curr = intervals.get(i);
51:                     if (curr.start > prev.end)
52:                     {
53:                         ans.add(prev);
54:                         prev = curr;
55:                     }
56:                     else
57:                     {
58:                         Interval merged = new Interval(prev.start, Math.max(prev.end, curr.end));
59:                         prev = merged;
60:                     }
61:                 }
62:                 ans.add(prev);
63:                 return ans;
64:         }
65: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Comparator;
 5: import java.util.PriorityQueue;
 6:
 7: public class MergeKSortedLists
 8: {
 9:         public static ListNode<Integer> Merge(ArrayList<ListNode<Integer>> lists)
10:         {
11:                 if (lists == null || lists.isEmpty())
12:                         return null;
13:
14:                 PriorityQueue<ListNode<Integer>> minheap = new PriorityQueue<ListNode<Integer>>(lists.size(),
15:                                 new Comparator<ListNode<Integer>>()
16:                                 {
17:                                         @Override
18:                                         public int compare(ListNode<Integer> o1, ListNode<Integer> o2)
19:                                         {
20:                                                 return o1.data - o2.data;
21:                                         }
22:                                 });
23:
24:                 for (ListNode<Integer> list : lists)
25:                         minheap.add(list);
26:
27:                 ListNode<Integer> head = minheap.poll();
28:                 ListNode<Integer> end = head;
29:
30:                 while (!minheap.isEmpty())
31:                 {
32:                         if (end.next != null)
33:                                 minheap.add(end.next);
34:
35:                         end.next = minheap.poll();
36:                         end = end.next;
37:                 }
38:
39:                 return head;
40:         }
41:
42:         public static void main(String args[])
43:         {
44:                 ArrayList<ListNode<Integer>> lists = new ArrayList<ListNode<Integer>>();
45:                 MyLinkedList<Integer> l1 = new MyLinkedList<Integer>();
46:                 l1.insertAtEnd(10);
47:                 l1.insertAtEnd(12);
48:                 l1.insertAtEnd(13);
```

```
49:                 l1.insertAtEnd(15);
50:                 lists.add(l1.head);
51:                 l1.printList();
52:
53:                 MyLinkedList<Integer> l2 = new MyLinkedList<Integer>();
54:                 l2.insertAtEnd(1);
55:                 l2.insertAtEnd(5);
56:                 l2.insertAtEnd(6);
57:                 l2.insertAtEnd(8);
58:                 lists.add(l2.head);
59:                 l2.printList();
60:
61:                 MyLinkedList<Integer> l3 = new MyLinkedList<Integer>();
62:                 l3.insertAtEnd(2);
63:                 l3.insertAtEnd(3);
64:                 l3.insertAtEnd(4);
65:                 l3.insertAtEnd(9);
66:                 lists.add(l3.head);
67:                 l3.printList();
68:
69:                 MyLinkedList<Integer> l4 = new MyLinkedList<Integer>();
70:                 l4.insertAtEnd(7);
71:                 l4.insertAtEnd(11);
72:                 l4.insertAtEnd(14);
73:                 l4.insertAtEnd(16);
74:                 lists.add(l4.head);
75:                 l4.printList();
76:
77:                 MyLinkedList<Integer> l5 = new MyLinkedList<Integer>();
78:                 l5.head = Merge(lists);
79:                 l5.printList();
80:         }
81: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class MergeSort
 6: {
 7:         static void mergeSort(int[] a)
 8:         {
 9:                 mergeSort(a, 0, a.length - 1);
10:         }
11:
12:         static void mergeSort(int[] a, int start, int end)
13:         {
14:                 if (start < end)
15:                 {
16:                         int mid = start + (end - start) / 2;
17:                         mergeSort(a, start, mid);
18:                         mergeSort(a, mid + 1, end);
19:
20:                         merge(a, start, mid, end);
21:                 }
22:         }
23:
24:         static void merge(int[] a, int start, int mid, int end)
25:         {
26:                 int n1 = mid - start + 1;
27:                 int n2 = end - mid;
28:                 int left[] = new int[n1];
29:                 int right[] = new int[n2];
30:                 for (int i = 0; i < n1; i++)
31:                         left[i] = a[start + i];
32:                 for (int i = 0; i < n2; i++)
33:                         right[i] = a[mid + i + 1];
34:
35:                 int i = 0, j = 0, k = start;
36:                 while (i < left.length && j < right.length)
37:                 {
38:                         if (left[i] <= right[j])
39:                                 a[k++] = left[i++];
40:                         else
41:                                 a[k++] = right[j++];
42:                 }
43:
44:                 while (i < left.length)
45:                         a[k++] = left[i++];
46:
47:                 while (j < right.length)
48:                         a[k++] = right[j++];
```

```
49:         }
50:
51:         public static void main(String[] args)
52:         {
53:                 int[] a = { 32, 4, 15, 66, 7, 2, 88, 45, 3, 9, 4, 23 };
54:                 mergeSort(a);
55:                 System.out.println(Arrays.toString(a));
56:         }
57: }
```

```
 1: package problems;
 2:
 3: import java.util.Date;
 4:
 5: /*
 6:    If V == 0, then 0 coins required.
 7:    If V > 0
 8:    minCoin(coins[0..m-1], V) = min {1 + minCoins(V-coin[i])}
 9:                                where i varies from 0 to m-1
10:                                and coin[i] < V
11:  */
12:
13: public class MinCoins
14: {
15:         public static int minCoinsFast(int coins[], int m, int v)
16:         {
17:                 // table[i] will be storing the minimum number of coins
18:                 // required for i value. So table[v] will have result
19:                 int[] table = new int[v + 1];
20:
21:                 // Base case (If given value v is 0)
22:                 table[0] = 0;
23:
24:                 // Initialize all table values as Infinite
25:                 for (int i = 1; i <= v; i++)
26:                         table[i] = Integer.MAX_VALUE;
27:
28:                 // Compute minimum coins required for all
29:                 // values from 1 to v
30:                 for (int i = 1; i <= v; i++)
31:                 {
32:                         // Go through all coins smaller than i
33:                         for (int j = 0; j < m; j++)
34:                         {
35:                                 if (coins[j] < i)
36:                                 {
37:                                         int sub_res = table[i - coins[j]];
38:                                         table[i] = Math.min(sub_res + 1, table[i]);
39:                                 }
40:                                 // comment this else only if exact amount is required
41:                                 // and add check for Integer.MAX_VALUE above before finding min
42:                                 else
43:                                 {
44:                                         table[i] = 1;
45:                                 }
46:                                 // System.out.println(Arrays.toString(table));
47:                         }
48:                 }
```

```
49:
50:                     return table[v];
51:             }
52:
53:             public static int minCoinsSlow(int coins[], int m, int v)
54:             {
55:                     if (v <= 0)
56:                             return 0;
57:
58:                     int res = Integer.MAX_VALUE;
59:
60:                     for (int j = 0; j < m; j++)
61:                     {
62:                             if (coins[j] <= v)
63:                             {
64:                                     int sub_res = minCoinsSlow(coins, m, v - coins[j]);
65:                                     if (sub_res + 1 < res)
66:                                             res = sub_res + 1;
67:                             }
68:                             else
69:                             {
70:                                     return 1;
71:                             }
72:                     }
73:
74:                     return res;
75:             }
76:
77:             public static void main(String[] args)
78:             {
79:                     int coins[] = { 3, 5, 7 };
80:                     System.out.println(new Date());
81:                     System.out.println("Minimum coins required is " + minCoinsFast(coins, coins.length, 100));
82:                     System.out.println(new Date());
83:                     System.out.println("Minimum coins required is " + minCoinsSlow(coins, coins.length, 100));
84:                     System.out.println(new Date());
85:             }
86: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class MinGates
 6: {
 7:         static int findMinGates(int[] arrivals, int[] departures, int flights)
 8:         {
 9:                 if (flights < 1)
10:                         return 0;
11:                 Arrays.sort(arrivals);
12:                 Arrays.sort(departures);
13:
14:                 int gates = 1, minGates = 1, i = 1, j = 0;
15:                 while (i < flights && j < flights)
16:                 {
17:                         if (arrivals[i] <= departures[j])
18:                         {
19:                                 gates++;
20:                                 i++;
21:                                 if (gates > minGates)
22:                                         minGates = gates;
23:                         }
24:                         else
25:                         {
26:                                 gates--;
27:                                 j++;
28:                         }
29:                 }
30:                 return minGates;
31:         }
32:
33:         public static void main(String[] args)
34:         {
35:
36:         }
37:
38: }
```

```
 1: package problems;
 2:
 3: public class MinHeap
 4: {
 5:         private int[] Heap;
 6:         private int size;
 7:         private int maxsize;
 8:
 9:         private static final int FRONT = 1;
10:
11:         public MinHeap(int maxsize)
12:         {
13:                 this.maxsize = maxsize;
14:                 this.size = 0;
15:                 Heap = new int[this.maxsize + 1];
16:                 Heap[0] = Integer.MIN_VALUE;
17:         }
18:
19:         private int parent(int pos)
20:         {
21:                 return pos / 2;
22:         }
23:
24:         private int leftChild(int pos)
25:         {
26:                 return (2 * pos);
27:         }
28:
29:         private boolean isLeaf(int pos)
30:         {
31:                 if (pos > (size / 2) && pos <= size) { return true; }
32:                 return false;
33:         }
34:
35:         private void swap(int fpos, int spos)
36:         {
37:                 int tmp;
38:                 tmp = Heap[fpos];
39:                 Heap[fpos] = Heap[spos];
40:                 Heap[spos] = tmp;
41:         }
42:
43:         private void minHeapify(int pos)
44:         {
45:                 while(!isLeaf(pos))
46:                 {
47:                         int newPos = leftChild(pos);
48:                         if((newPos < size) && Heap[newPos+1] < Heap[newPos])
```

```
49:                                 newPos++; // move to right child
50:                         if(Heap[newPos] > Heap[pos])
51:                                 break;
52:                         swap(pos, newPos);
53:                         pos = newPos;
54:                 }
55:         }
56:
57:         public void insert(int element)
58:         {
59:                 Heap[++size] = element;
60:                 int current = size;
61:                 while (Heap[current] < Heap[parent(current)])
62:                 {
63:                         swap(current, parent(current));
64:                         current = parent(current);
65:                 }
66:         }
67:
68:         public void minHeap() // this function is just to form a minheap from existing array
69:         {
70:                 for (int pos = (size / 2); pos >= 1; pos--)
71:                 {
72:                         minHeapify(pos);
73:                 }
74:         }
75:
76:         public int remove()
77:         {
78:                 int popped = Heap[FRONT];
79:                 if(size == FRONT)
80:                 {
81:                         Heap[size--] = Integer.MAX_VALUE;
82:                 }
83:                 else
84:                 {
85:                         Heap[FRONT] = Heap[size];
86:                         Heap[size--] = Integer.MAX_VALUE;
87:                         minHeapify(FRONT);
88:                 }
89:
90:                 return popped;
91:         }
92:
93:         public void removeAll()
94:         {
95:                 while(size>0)
96:                         System.out.print(remove()+" ");
```

```
   97:                   System.out.println();
   98:           }
   99:
  100:           public static void main(String... arg)
  101:           {
  102:                   System.out.println("The Min Heap is ");
  103:                   MinHeap minHeap = new MinHeap(10);
  104:                   minHeap.insert(5);
  105:                   minHeap.insert(3);
  106:                   minHeap.insert(17);
  107:                   minHeap.insert(10);
  108:                   minHeap.insert(84);
  109:                   minHeap.insert(19);
  110:                   minHeap.insert(6);
  111:                   minHeap.insert(22);
  112:                   minHeap.insert(9);
  113:                   minHeap.insert(100);
  114:
  115:                   minHeap.removeAll();
  116:
  117:           }
  118: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Collections;
 5: import java.util.Comparator;
 6: import java.util.List;
 7: import java.util.Random;
 8:
 9: class Order
10: {
11:         String productId;
12:         double cost;
13:         int quantity;
14:
15:         public Order(String id, double cost, int q)
16:         {
17:                 this.productId = id;
18:                 this.cost = cost;
19:                 this.quantity = q;
20:         }
21:
22:         @Override
23:         public String toString()
24:         {
25:                 return "Q: " + this.quantity + " C: " + this.cost;
26:         }
27: }
28:
29: public class MyComparator
30: {
31:         public static void main(String[] args)
32:         {
33:                 List<Order> list = new ArrayList<Order>();
34:                 Random r = new Random();
35:                 for (int i = 0; i < 100; i++)
36:                 {
37:                         Order o = new Order("Product" + i, r.nextDouble() * 100, r.nextInt(100));
38:                         list.add(o);
39:                 }
40:
41:                 for (int i = 0; i < 100; i++)
42:                         System.out.print(list.get(i).toString() + " ");
43:                 System.out.println();
44:
45:                 Collections.sort(list, new Comparator<Order>()
46:                 {
47:                         @Override
48:                         public int compare(Order o1, Order o2)
```

```
49:                             {
50:                                     if (o1.quantity != o2.quantity)
51:                                             return o1.quantity - o2.quantity;
52:                                     else
53:                                             return (int) (o1.cost - o2.cost);
54:                             }
55:                     });
56:
57:                     for (int i = 0; i < 100; i++)
58:                             System.out.print(list.get(i).toString() + " ");
59:             }
60:
61: }
```

```
 1: package problems;
 2:
 3: class ListNode<T>
 4: {
 5:         public T data;
 6:         public ListNode<T> next;
 7:
 8:         public ListNode()
 9:         {
10:                 next = null;
11:         }
12:
13:         public ListNode(T d)
14:         {
15:                 data = d;
16:                 next = null;
17:         }
18: }
19:
20: class RandomListNode
21: {
22:         int label;
23:         RandomListNode next, random;
24:
25:         RandomListNode(int x)
26:         {
27:                 this.label = x;
28:         }
29: };
30:
31: public class MyLinkedList<T>
32: {
33:         ListNode<T> head;
34:         ListNode<T> tail;
35:
36:         public MyLinkedList()
37:         {
38:                 head = null;
39:                 tail = null;
40:         }
41:
42:         public ListNode<T> getHead()
43:         {
44:                 return head;
45:         }
46:
47:         public void setHead(ListNode<T> h)
48:         {
```

```
49:                     head = h;
50:             }
51:
52:             public ListNode<T> getTail()
53:             {
54:                     return tail;
55:             }
56:
57:             public int length()
58:             {
59:                     ListNode<T> t = head;
60:                     int len = 0;
61:                     while (t != null)
62:                     {
63:                             len++;
64:                             t = t.next;
65:                     }
66:
67:                     return len;
68:             }
69:
70:             void insertAtStart(T d)
71:             {
72:                     ListNode<T> newNode = new ListNode<T>(d);
73:
74:                     if (head != null)
75:                             newNode.next = head;
76:                     else
77:                             tail = newNode;
78:                     head = newNode;
79:             }
80:
81:             void insertAtEnd(T d)
82:             {
83:                     ListNode<T> newNode = new ListNode<T>(d);
84:
85:                     if (head == null)
86:                             head = newNode;
87:                     else
88:                             tail.next = newNode;
89:                     tail = newNode;
90:             }
91:
92:             void printList()
93:             {
94:                     if (head == null)
95:                             return;
96:
```

```
   97:                     ListNode<T> node = head;
   98:                     while (node != null)
   99:                     {
  100:                             System.out.print(node.data);
  101:                             if (node.next != null)
  102:                                     System.out.print("->");
  103:                             node = node.next;
  104:                     }
  105:                     System.out.println();
  106:             }
  107:
  108:             ListNode<T> reverseList(ListNode<T> head)
  109:             {
  110:                     if (head == null || head.next == null)
  111:                             return head;
  112:
  113:                     ListNode<T> node = head;
  114:                     ListNode<T> prev = null;
  115:
  116:                     while (node != null)
  117:                     {
  118:                             ListNode<T> next = node.next;
  119:                             node.next = prev;
  120:                             prev = node;
  121:                             node = next;
  122:                     }
  123:
  124:                     return prev;
  125:             }
  126:
  127:             void reverseList()
  128:             {
  129:                     ListNode<T> curr = head;
  130:                     ListNode<T> prev = null;
  131:                     ListNode<T> next = null;
  132:
  133:                     while (curr != null)
  134:                     {
  135:                             next = curr.next;
  136:                             curr.next = prev;
  137:                             prev = curr;
  138:                             curr = next;
  139:                     }
  140:
  141:                     head = prev;
  142:             }
  143:
  144: }
```

```
 1: package problems;
 2:
 3: import java.util.Comparator;
 4: import java.util.PriorityQueue;
 5:
 6: public class MyPriorityQueue
 7: {
 8:         public static void main(String[] args)
 9:         {
10:                 Comparator<Integer> comp = new Comparator<Integer>()
11:                 {
12:                         @Override
13:                         public int compare(Integer o1, Integer o2)
14:                         {
15:                                 return o2 - o1;
16:                         }
17:                 };
18:                 PriorityQueue<Integer> pq = new PriorityQueue<Integer>(comp);
19:                 pq.add(7);
20:                 pq.add(8);
21:                 pq.add(5);
22:                 pq.add(9);
23:                 while (pq.size() > 0)
24:                         System.out.println(pq.poll());
25:         }
26: }
```

```
 1: package problems;
 2:
 3: import java.util.HashSet;
 4: import java.util.LinkedList;
 5: import java.util.Queue;
 6: import java.util.Set;
 7:
 8: class NaryTreeNode<T>
 9: {
10:         T data;
11:         NaryTreeNode<T> firstChild;
12:         NaryTreeNode<T> nextSibling;
13:
14:         public NaryTreeNode(T d)
15:         {
16:                 data = d;
17:                 firstChild = null;
18:                 nextSibling = null;
19:         }
20: }
21:
22: public class NaryTree<T>
23: {
24:         NaryTreeNode<T> root;
25:
26:         public NaryTree()
27:         {
28:                 root = null;
29:         }
30:
31:         public NaryTreeNode<T> insert(NaryTreeNode<T> root, T parent, T d)
32:         {
33:                 NaryTreeNode<T> newNode = new NaryTreeNode<T>(d);
34:                 if (root == null)
35:                 {
36:                         root = newNode;
37:                         return root;
38:                 }
39:
40:                 NaryTreeNode<T> temp = findNode(root, parent);
41:
42:                 if (temp == null)
43:                 {
44:                         newNode.nextSibling = root.nextSibling;
45:                         root.nextSibling = newNode;
46:                 }
47:                 else if (temp.firstChild != null)
48:                 {
```

```
49:                         newNode.nextSibling = temp.firstChild.nextSibling;
50:                         temp.firstChild.nextSibling = newNode;
51:                 }
52:                 else
53:                 {
54:                         temp.firstChild = newNode;
55:                 }
56:
57:                 return root;
58:         }
59:
60:         public NaryTreeNode<T> findNode(NaryTreeNode<T> root, T d)
61:         {
62:                 if (root == null)
63:                         return null;
64:
65:                 if (root.data == d)
66:                         return root;
67:
68:                 NaryTreeNode<T> temp = findNode(root.firstChild, d);
69:
70:                 if (temp == null)
71:                 {
72:                         temp = root.nextSibling;
73:                         while (temp != null)
74:                         {
75:                                 if (temp.data == d)
76:                                         break;
77:
78:                                 NaryTreeNode<T> t = findNode(temp.firstChild, d);
79:                                 if (t != null)
80:                                 {
81:                                         temp = t;
82:                                         break;
83:                                 }
84:
85:                                 temp = temp.nextSibling;
86:                         }
87:                 }
88:
89:                 return temp;
90:         }
91:
92:         public void printTree(NaryTreeNode<T> root)
93:         {
94:                 if (root == null)
95:                         return;
96:
```

```
 97:                 System.out.print(root.data + " ");
 98:
 99:                 NaryTreeNode<T> temp = root.nextSibling;
100:                 while (temp != null)
101:                 {
102:                         System.out.print(temp.data + " ");
103:                         printTree(temp.firstChild);
104:                         temp = temp.nextSibling;
105:                 }
106:                 System.out.println();
107:                 printTree(root.firstChild);
108:         }
109:
110:         public void printSiblings(T d)
111:         {
112:                 if (root == null)
113:                         return;
114:
115:                 Queue<NaryTreeNode<T>> q = new LinkedList<NaryTreeNode<T>>();
116:                 Set<T> level = new HashSet<T>();
117:                 int currLevel = 0, nextLevel = 0;
118:                 NaryTreeNode<T> temp = root;
119:                 while (temp != null)
120:                 {
121:                         q.offer(temp);
122:                         currLevel++;
123:                         level.add(temp.data);
124:                         temp = temp.nextSibling;
125:                 }
126:
127:                 if (level.contains(d))
128:                 {
129:                         System.out.println(level);
130:                         return;
131:                 }
132:
133:                 level.clear();
134:                 while (!q.isEmpty())
135:                 {
136:                         temp = q.poll();
137:                         currLevel--;
138:                         if (temp.firstChild != null)
139:                         {
140:                                 q.offer(temp.firstChild);
141:                                 nextLevel++;
142:                                 level.add(temp.firstChild.data);
143:                                 temp = temp.firstChild;
144:                                 while (temp.nextSibling != null)
```

```
145:                                    {
146:                                            q.offer(temp.nextSibling);
147:                                            nextLevel++;
148:                                            level.add(temp.nextSibling.data);
149:                                            temp = temp.nextSibling;
150:                                    }
151:                            }
152:                            if (currLevel == 0)
153:                            {
154:                                    currLevel = nextLevel;
155:                                    nextLevel = 0;
156:                                    if (level.contains(d))
157:                                    {
158:                                            System.out.println(level);
159:                                            return;
160:                                    }
161:                                    level.clear();
162:                                    System.out.println();
163:                            }
164:                    }
165:
166:            }
167:
168:            public static void main(String[] args)
169:            {
170:                    NaryTree<String> ftree = new NaryTree<String>();
171:
172:                    ftree.root = ftree.insert(ftree.root, "", "adam");
173:                    ftree.root = ftree.insert(ftree.root, "adam", "sam");
174:                    ftree.root = ftree.insert(ftree.root, "adam", "watson");
175:                    ftree.root = ftree.insert(ftree.root, "sam", "bob");
176:                    ftree.root = ftree.insert(ftree.root, "sam", "jon");
177:                    ftree.root = ftree.insert(ftree.root, "sam", "ruby");
178:                    ftree.root = ftree.insert(ftree.root, "watson", "roger");
179:
180:                    ftree.printTree(ftree.root);
181:                    ftree.printSiblings("bob");
182:            }
183: }
```

```
 1: package problems;
 2:
 3: import java.util.LinkedList;
 4: import java.util.Queue;
 5:
 6: public class NextBiggestString
 7: {
 8:         public static String findNextBiggest(String in)
 9:         {
10:                 int len = in.length();
11:                 int i = len - 1;
12:                 char temp = 0;
13:                 StringBuilder sb = new StringBuilder(in);
14:                 StringBuilder sb1 = new StringBuilder();
15:                 Queue<Character> queue = new LinkedList<Character>();
16:                 for (; i > 0; i--)
17:                 {
18:                         char a = in.charAt(i);
19:                         char b = in.charAt(i - 1);
20:                         if (a <= b)
21:                         {
22:                                 queue.add(a);
23:                                 continue;
24:                         }
25:                         queue.add(a);
26:
27:                         System.out.println("queue " + queue.toString());
28:
29:                         temp = queue.poll();
30:                         while (temp <= b)
31:                         {
32:                                 sb1.append(temp);
33:                                 temp = queue.poll();
34:                         }
35:                         System.out.println("sb1: " + sb1.toString());
36:                         sb.setCharAt(i - 1, temp);
37:                         sb1.append(b);
38:
39:                         while (!queue.isEmpty())
40:                                 sb1.append(queue.poll());
41:                         break;
42:                 }
43:                 System.out.println("sb1: " + sb1.toString());
44:                 sb.setLength(i);
45:                 return sb.append(sb1).toString();
46:         }
47:
48:         public static void main(String args[])
```

```
49:         {
50:                 String s = findNextBiggest("abcdeedcba");
51:                 System.out.println(s);
52:         }
53: }
```

```
 1: package problems;
 2:
 3: import java.io.File;
 4: import java.io.FileNotFoundException;
 5: import java.util.*;
 6:
 7: /*
 8:     24988
 9:         28693
10:         12907
11:         9197
12:         35287
13:  */
14:
15: public class Palindrome
16: {
17:
18:         public static boolean isPalindrome(String s)
19:         {
20:                 for (int i = 0, j = s.length() - 1; i < s.length() / 2; i++, j--)
21:                 {
22:                         if (s.charAt(i) != s.charAt(j))
23:                                 return false;
24:                 }
25:                 return true;
26:         }
27:
28:         public static void PalIndex(String s)
29:         {
30:                 int i = 0, j = s.length() - 1;
31:                 for (; i < s.length() / 2;)
32:                 {
33:                         if (s.charAt(i) == s.charAt(j))
34:                         {
35:                                 i++;
36:                                 j--;
37:                                 continue;
38:                         }
39:                         if (isPalindrome(s.substring(i + 1, j + 1)))
40:                         {
41:                                 System.out.println(i);
42:                                 break;
43:                         }
44:                         System.out.println(j);
45:                         break;
46:                 }
47:         }
48:
```

```
49:         public static void main(String[] args) throws FileNotFoundException
50:         {
51:                 System.out.println(new Date());
52:                 Scanner sc = new Scanner(new File("pal1.txt"));
53:                 int count = sc.nextInt();
54:                 List<String> strList = new ArrayList<String>();
55:                 for (int i = 0; i < count; i++)
56:                         strList.add(sc.next());
57:                 sc.close();
58:                 for (String s : strList)
59:                 {
60:                         if (isPalindrome(s))
61:                         {
62:                                 System.out.println("-1");
63:                                 continue;
64:                         }
65:                         PalIndex(s);
66:                 }
67:                 System.out.println(new Date());
68:         }
69: }
```

```java
package problems;

import java.util.Scanner;

public class Pangram
{
	static void isPangram(String s)
	{
		if (s == null || s.isEmpty())
			return;

		int[] freq = new int[26];
		int len = s.length();
		for (int i = 0; i < len; i++)
		{
			char c = s.charAt(i);
			if (c < 'a' || c > 'z')
				continue;
			freq[c - 'a']++;
		}

		int count = freq[0];
		for (int i = 0; i < 26; i++)
		{
			if (count == 0)
			{
				System.out.println("not pangram");
				return;
			}
			else if (count != freq[i])
			{
				System.out.println("pangram");
				return;
			}
		}
		System.out.println("multiple pangram " + count);
	}

	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		String s = sc.nextLine();
		isPangram(s.toLowerCase());
		sc.close();
	}
}
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Date;
 5: import java.util.HashSet;
 6:
 7: public class Permutation
 8: {
 9:         public static HashSet<String> perm(String s)
10:         {
11:                 if (s == null)
12:                         return null;
13:                 HashSet<String> permList = new HashSet<String>();
14:                 if (s.isEmpty())
15:                 {
16:                         permList.add("");
17:                         return permList;
18:                 }
19:
20:                 char first = s.charAt(0);
21:                 String remaining = s.substring(1);
22:                 HashSet<String> inter = perm(remaining);
23:
24:                 for (String str : inter)
25:                 {
26:                         for (int i = 0; i <= str.length(); i++)
27:                                 permList.add(insertAt(str, first, i));
28:                 }
29:                 return permList;
30:         }
31:
32:         public static String insertAt(String s, char c, int i)
33:         {
34:                 return (s.substring(0, i) + c + s.substring(i));
35:         }
36:
37:         // alternative approach
38:         static ArrayList<String> getPerms(String s)
39:         {
40:                 ArrayList<String> result = new ArrayList<>();
41:                 getPerms("", s, result);
42:                 return result;
43:         }
44:
45:         static void getPerms(String prefix, String rem, ArrayList<String> result)
46:         {
47:                 if (rem.length() == 0)
48:                         result.add(prefix);
```

```
49:
50:                 int len = rem.length();
51:                 for (int i = 0; i < len; i++)
52:                 {
53:                         String before = rem.substring(0, i);
54:                         String after = rem.substring(i + 1);
55:                         char c = rem.charAt(i);
56:                         getPerms(prefix + c, before + after, result);
57:                 }
58:         }
59:
60:         public static void main(String[] args)
61:         {
62:                 String a = "aaaaaaaaaaaa";
63:                 System.out.println(new Date());
64:                 System.out.println(perm(a).size());
65:                 System.out.println(new Date());
66:                 System.out.println(getPerms(a).size());
67:                 System.out.println(new Date());
68:         }
69:
70: }
```

```
 1: package problems;
 2:
 3: public class PermutationPalindrome
 4: {
 5:         public static boolean isPermPalindrome(String s)
 6:         {
 7:                 int bitVector = 0;
 8:                 for (char c : s.toCharArray())
 9:                         bitVector = bitVector ^ (1 << (c - 'a'));
10:                 // System.out.println(Integer.toBinaryString(bitVector) + " " +
11:                 // Integer.toBinaryString(bitVector - 1));
12:                 return (bitVector & (bitVector - 1)) == 0;
13:         }
14:
15:         public static void main(String[] args)
16:         {
17:                 System.out.println(isPermPalindrome("Tactooca".toLowerCase()));
18:         }
19:
20: }
```

```
 1: package problems;
 2:
 3: import java.io.File;
 4: import java.io.FileNotFoundException;
 5: import java.util.*;
 6:
 7: public class PoisonousPlant
 8: {
 9:
10:         public static void main(String[] args) throws FileNotFoundException
11:         {
12:                 Scanner sc = new Scanner(new File("poison.txt"));
13:                 List<Integer> nums = new ArrayList<Integer>();
14:                 int listSize = sc.nextInt();
15:                 for (int i = 0; i < listSize; i++)
16:                 {
17:                         nums.add(sc.nextInt());
18:                 }
19:                 int ans = 0;
20:                 while (true)
21:                 {
22:                         int deadCnt = 0;
23:                         int left = -1;
24:                         for (int i = 1; i < nums.size();)
25:                         {
26:                                 if (nums.get(i) > ((left == -1) ? nums.get(i - 1) : left))
27:                                 {
28:                                         left = nums.remove(i);
29:                                         deadCnt++;
30:                                 }
31:                                 else
32:                                 {
33:                                         left = -1;
34:                                         i++;
35:                                 }
36:                         }
37:
38:                         if (deadCnt == 0)
39:                                 break;
40:                         ans++;
41:                 }
42:                 System.out.print(ans);
43:                 sc.close();
44:         }
45: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.List;
 5:
 6: public class PowerSet
 7: {
 8:         public static void printAllSets(char[] c)
 9:         {
10:                 List<String> list = new ArrayList<String>();
11:                 printSets(c, 0, list);
12:                 System.out.println(list);
13:                 System.out.println(list.size());
14:         }
15:
16:         private static void printSets(char[] c, int index, List<String> list)
17:         {
18:                 if (index == c.length)
19:                         return;
20:                 int len = list.size();
21:
22:                 for (int i = 0; i < len; i++)
23:                         list.add(list.get(i) + c[index]);
24:
25:                 list.add(c[index] + "");
26:                 index++;
27:                 printSets(c, index, list);
28:         }
29:
30:         public static void main(String args[])
31:         {
32:                 char[] c = { 'a', 'b', 'c', 'd', 'e' };
33:                 printAllSets(c);
34:         }
35: }
```

```
 1: package problems;
 2:
 3: import java.util.ArrayList;
 4: import java.util.Arrays;
 5:
 6: public class PowerSetIterative
 7: {
 8:         static ArrayList<String> buildSubsequences(String s)
 9:         {
10:                 ArrayList<String> subsequence = new ArrayList<String>();
11:                 int len = s.length();
12:                 for (int i = 0; i < len; i++)
13:                 {
14:                         if (subsequence.size() > 0)
15:                         {
16:                                 int l = subsequence.size();
17:                                 for (int j = 0; j < l; j++)
18:                                         subsequence.add(subsequence.get(j) + s.charAt(i));
19:                         }
20:                         subsequence.add(s.charAt(i) + "");
21:                 }
22:
23:                 return subsequence;
24:         }
25:
26:         public static void main(String[] args)
27:         {
28:                 ArrayList<String> arr = buildSubsequences("abba");
29:                 System.out.println(Arrays.toString(arr.toArray(new String[0])));
30:         }
31: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4: import java.util.Random;
 5:
 6: public class QuickSort
 7: {
 8:         private static int partition(int[] a, int l, int r)
 9:         {
10:                 int i = l - 1, j = r, temp = 0;
11:                 while (true)
12:                 {
13:                         while (a[++i] < a[r]);
14:
15:                         while (a[--j] > a[r])
16:                                 if (j == i)
17:                                         break;
18:
19:                         if (i >= j)
20:                                 break;
21:
22:                         temp = a[i];
23:                         a[i] = a[j];
24:                         a[j] = temp;
25:                 }
26:                 if (i != r)
27:                 {
28:                         temp = a[i];
29:                         a[i] = a[r];
30:                         a[r] = temp;
31:                 }
32:                 return i;
33:         }
34:
35:         public static void sort(int[] a, int low, int high)
36:         {
37:                 if (low >= high)
38:                         return;
39:                 int p = partition(a, low, high);
40:                 sort(a, low, p - 1);
41:                 sort(a, p + 1, high);
42:         }
43:
44:         public static void main(String[] args)
45:         {
46:                 int a[] = new int[1000];
47:                 Random r = new Random();
48:                 for (int i = 0; i < 1000; i++)
```

```
49:                     a[i] = r.nextInt(1000);
50:                 System.out.println(Arrays.toString(a));
51:                 sort(a, 0, 999);
52:                 System.out.println(Arrays.toString(a));
53:         }
54:
55: }
```

```
 1: package problems;
 2:
 3: import java.util.Stack;
 4:
 5: public class RemoveMatchingPairs
 6: {
 7:
 8:         boolean isUpper(char c)
 9:         {
10:                 return (c >= 'A' && c <= 'Z');
11:         }
12:
13:         boolean isLower(char c)
14:         {
15:                 return (c >= 'a' && c <= 'z');
16:         }
17:
18:         boolean equalsIgnoreCase(char upper, char lower)
19:         {
20:                 return (lower - upper == 32);
21:         }
22:
23:         int findMatchingPair(String input)
24:         {
25:                 Stack<Character> st = new Stack<Character>();
26:                 int len = input.length();
27:                 int retValue = -1;
28:                 for (int i = 0; i < len; i++)
29:                 {
30:                         char c = input.charAt(i);
31:                         if (isUpper(c))
32:                         {
33:                                 st.push(c);
34:                         }
35:                         else
36:                         {
37:                                 if (!st.empty() && equalsIgnoreCase(st.peek(), c))
38:                                 {
39:                                         retValue = i;
40:                                         st.pop();
41:                                 }
42:                                 else
43:                                 {
44:                                         return retValue;
45:                                 }
46:                         }
47:                 }
48:                 return retValue;
```

```
49:         }
50:
51:         public static void main(String[] args)
52:         {
53:
54:         }
55:
56: }
```

```
 1: package problems;
 2:
 3: public class RLE
 4: {
 5:         public static String compressBad(String str)
 6:         {
 7:                 String mystr = "";
 8:                 char last = str.charAt(0);
 9:                 int count = 1;
10:                 for (int i = 1; i < str.length(); i++)
11:                 {
12:                         if (str.charAt(i) == last)
13:                         {
14:                                 count++;
15:                         }
16:                         else
17:                         {
18:                                 mystr += last + "" + count;
19:                                 last = str.charAt(i);
20:                                 count = 1;
21:                         }
22:                 }
23:                 return mystr + last + count;
24:         }
25:
26:         public static void main(String[] args)
27:         {
28:                 String str = "abcd";
29:                 String str2 = compressBad(str);
30:                 System.out.println("New String (len = " + str2.length() + "): " + str2);
31:         }
32:
33: }
```

```
 1: package problems;
 2:
 3: public class Roate90Degree
 4: {
 5:         static int[][] rotateExtraSpace(int[][] a)
 6:         {
 7:                 int m = a.length, n = a[0].length;
 8:                 int[][] b = new int[n][m];
 9:                 for (int i = 0; i < m; i++)
10:                         for (int j = 0; j < n; j++)
11:                                 b[j][m - i - 1] = a[i][j];
12:                 return b;
13:         }
14:
15:         // can be done only for sq matrix. ratate one number at a time
16:         static void rotateInPlace(int[][] a)
17:         {
18:                 if (a.length == 0 || a.length != a[0].length)
19:                         return;
20:                 int n = a.length;
21:                 for (int i = 0; i < n / 2; i++)
22:                 {
23:                         for (int j = i; j < n - 1 - i; j++)
24:                         {
25:                                 int temp = a[i][j];
26:                                 a[i][j] = a[n - 1 - j][i];
27:                                 a[n - 1 - j][i] = a[n - 1 - i][n - 1 - j];
28:                                 a[n - 1 - i][n - 1 - j] = a[j][n - 1 - i];
29:                                 a[j][n - 1 - i] = temp;
30:                         }
31:                 }
32:         }
33:
34:         static void printMatrix(int[][] a)
35:         {
36:                 for (int i = 0; i < a.length; i++)
37:                 {
38:                         for (int j = 0; j < a[0].length; j++)
39:                                 System.out.print(a[i][j] + " ");
40:                         System.out.println();
41:                 }
42:         }
43:
44:         public static void main(String[] args)
45:         {
46:                 int[][] a = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 9, 10, 11, 12 } };
47:                 printMatrix(rotateExtraSpace(a));
48:                 rotateInPlace(a);
```

```
49:                     System.out.println();
50:                     printMatrix(a);
51:             }
52:
53: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class RodCutting
 6: {
 7:         static int cutRod(int price[], int n)
 8:         {
 9:                 int max[] = new int[n + 1];
10:                 max[1] = price[0];
11:                 for (int i = 2; i <= n; i++)
12:                 {
13:                         int max_val = Integer.MIN_VALUE;
14:                         for (int j = 0; j < i; j++)
15:                                 max_val = Math.max(max_val, price[j] + max[i - j - 1]);
16:                         max[i] = max_val;
17:                 }
18:                 System.out.println(Arrays.toString(max));
19:                 return max[n];
20:         }
21:
22:         public static void main(String args[])
23:         {
24:                 int arr[] = new int[] { 3, 7, 8, 9, 10, 17, 17, 20 };
25:                 int size = arr.length;
26:                 System.out.println("Maximum Obtainable Value is " + cutRod(arr, size));
27:
28:         }
29: }
```

```
 1: package problems;
 2:
 3: public class RotatedArray
 4: {
 5:         public static int findMin(int[] nums)
 6:         {
 7:                 if (nums == null || nums.length == 0)
 8:                         return -1;
 9:
10:                 int low = 0, high = nums.length - 1;
11:
12:                 if (nums[low] < nums[high])
13:                         return nums[low];
14:
15:                 while (low < high)
16:                 {
17:                         int mid = low + (high - low) / 2;
18:
19:                         if (mid < high && nums[mid + 1] < nums[mid])
20:                                 return nums[mid + 1];
21:
22:                         if ((mid > low) && (nums[mid] < nums[mid - 1]))
23:                                 return nums[mid];
24:
25:                         if (nums[low] < nums[mid])
26:                                 low = mid + 1;
27:
28:                         else
29:                                 high = mid - 1;
30:                 }
31:
32:                 return nums[low];
33:         }
34:
35:         static int find(int[] nums, int target)
36:         {
37:                 if (nums == null || nums.length == 0)
38:                         return -1;
39:
40:                 int low = 0, high = nums.length - 1;
41:                 while (low <= high)
42:                 {
43:                         int mid = low + (high - low) / 2;
44:                         if (nums[mid] == target)
45:                                 return mid;
46:
47:                         if (nums[low] <= nums[mid])
48:                         {
```

```
49:                                     if (target >= nums[low] && target < nums[mid])
50:                                             high = mid - 1;
51:                                     else
52:                                             low = mid + 1;
53:                             }
54:                             else
55:                             {
56:                                     if (target > nums[mid] && target <= nums[high])
57:                                             low = mid + 1;
58:                                     else
59:                                             high = mid - 1;
60:                             }
61:                     }
62:                     return -1;
63:             }
64:
65:             public static void main(String[] args)
66:             {
67:                     int[] nums = { 1 };
68:                     // int[] nums = { 7, 8, 8, 1, 2, 4, 5, 6, 7 };
69:                     // int[] nums = { 7, 7, 7, 7, 7, 7, 7 };
70:                     System.out.println(findMin(nums));
71:
72:             }
73: }
```

```
    1: package problems;
    2:
    3: import java.util.Collections;
    4: import java.util.PriorityQueue;
    5: import java.util.Scanner;
    6:
    7: public class RunningMedian
    8: {
    9:         public static void main(String args[]) throws Exception
   10:         {
   11:                 Scanner sc = new Scanner(System.in);
   12:                 int n = sc.nextInt(), a = 0;
   13:                 PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>(n / 2 + 1);
   14:                 PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>((n / 2 + 1), Collections.reverseOrder());
   15:                 double median = 0;
   16:                 for (int i = 0; i < n; i++)
   17:                 {
   18:                         a = sc.nextInt();
   19:                         if (a > median)
   20:                         {
   21:                                 if (maxHeap.size() < minHeap.size())
   22:                                         maxHeap.add(minHeap.poll());
   23:                                 minHeap.add(a);
   24:                                 median = minHeap.size() == maxHeap.size() ? (minHeap.peek() + maxHeap.peek()) / 2.0 : minHe
ap.peek();
   25:                         }
   26:                         else
   27:                         {
   28:                                 if (minHeap.size() < maxHeap.size())
   29:                                         minHeap.add(maxHeap.poll());
   30:                                 maxHeap.add(a);
   31:                                 median = minHeap.size() == maxHeap.size() ? (minHeap.peek() + maxHeap.peek()) / 2.0 : maxHe
ap.peek();
   32:                         }
   33:                         System.out.println(median);
   34:                 }
   35:                 sc.close();
   36:         }
   37: }
```

```
 1: package problems;
 2:
 3: import java.util.Scanner;
 4:
 5: /* Class SBBSTNode */
 6: class SBBSTNode
 7: {
 8:         SBBSTNode left, right;
 9:         int data;
10:         int height;
11:
12:         /* Constructor */
13:         public SBBSTNode()
14:         {
15:                 left = null;
16:                 right = null;
17:                 data = 0;
18:                 height = 0;
19:         }
20:
21:         /* Constructor */
22:         public SBBSTNode(int n)
23:         {
24:                 left = null;
25:                 right = null;
26:                 data = n;
27:                 height = 0;
28:         }
29: }
30:
31: public class SelfBalancingBST
32: {
33:
34:         private SBBSTNode root;
35:
36:         public SelfBalancingBST()
37:         {
38:                 root = null;
39:         }
40:
41:         /* Function to check if tree is empty */
42:         public boolean isEmpty()
43:         {
44:                 return root == null;
45:         }
46:
47:         /* Make the tree logically empty */
48:         public void clear()
```

```
49:         {
50:                 root = null;
51:         }
52:
53:         /* Function to insert data */
54:         public void insert(int data)
55:         {
56:                 root = insert(data, root);
57:         }
58:
59:         /* Function to get height of node */
60:         private int height(SBBSTNode t)
61:         {
62:                 return t == null ? -1 : t.height;
63:         }
64:
65:         /* Function to max of left/right node */
66:         private int max(int lhs, int rhs)
67:         {
68:                 return lhs > rhs ? lhs : rhs;
69:         }
70:
71:         /* Function to insert data recursively */
72:         private SBBSTNode insert(int x, SBBSTNode t)
73:         {
74:                 if (t == null)
75:                         t = new SBBSTNode(x);
76:                 else if (x <= t.data)
77:                 {
78:                         t.left = insert(x, t.left);
79:                         if (height(t.left) - height(t.right) == 2)
80:                                 if (x < t.left.data)
81:                                         t = rotateWithLeftChild(t);
82:                                 else
83:                                         t = doubleWithLeftChild(t);
84:                 }
85:                 else
86:                 {
87:                         t.right = insert(x, t.right);
88:                         if (height(t.right) - height(t.left) == 2)
89:                                 if (x > t.right.data)
90:                                         t = rotateWithRightChild(t);
91:                                 else
92:                                         t = doubleWithRightChild(t);
93:                 }
94:                 t.height = max(height(t.left), height(t.right)) + 1;
95:                 return t;
96:         }
```

```
 97:
 98:         /* Rotate binary tree node with left child */
 99:         private SBBSTNode rotateWithLeftChild(SBBSTNode k)
100:         {
101:                 SBBSTNode k1 = k.left;
102:                 k.left = k1.right;
103:                 k1.right = k;
104:                 k.height = max(height(k.left), height(k.right)) + 1;
105:                 k1.height = max(height(k1.left), k.height) + 1;
106:                 return k1;
107:         }
108:
109:         /* Rotate binary tree node with right child */
110:         private SBBSTNode rotateWithRightChild(SBBSTNode k)
111:         {
112:                 SBBSTNode k1 = k.right;
113:                 k.right = k1.left;
114:                 k1.left = k;
115:                 k.height = max(height(k.left), height(k.right)) + 1;
116:                 k1.height = max(height(k1.right), k.height) + 1;
117:                 return k1;
118:         }
119:
120:         private SBBSTNode doubleWithLeftChild(SBBSTNode k)
121:         {
122:                 System.out.println("doubleWithLeftChild");
123:                 k.left = rotateWithRightChild(k.left);
124:                 return rotateWithLeftChild(k);
125:         }
126:
127:         private SBBSTNode doubleWithRightChild(SBBSTNode k)
128:         {
129:                 System.out.println("doubleWithRightChild");
130:                 k.right = rotateWithLeftChild(k.right);
131:                 return rotateWithRightChild(k);
132:         }
133:
134:         /* Functions to count number of nodes */
135:         public int countNodes()
136:         {
137:                 return countNodes(root);
138:         }
139:
140:         private int countNodes(SBBSTNode r)
141:         {
142:                 if (r == null)
143:                         return 0;
144:                 else
```

```
145:                 {
146:                         int l = 1;
147:                         l += countNodes(r.left);
148:                         l += countNodes(r.right);
149:                         return l;
150:                 }
151:         }
152:
153:         /* Functions to search for an element */
154:         public boolean search(int val)
155:         {
156:                 return search(root, val);
157:         }
158:
159:         private boolean search(SBBSTNode r, int val)
160:         {
161:                 boolean found = false;
162:                 while ((r != null) && !found)
163:                 {
164:                         int rval = r.data;
165:                         if (val < rval)
166:                                 r = r.left;
167:                         else if (val > rval)
168:                                 r = r.right;
169:                         else
170:                         {
171:                                 found = true;
172:                                 break;
173:                         }
174:                         found = search(r, val);
175:                 }
176:                 return found;
177:         }
178:
179:         /* Function for inorder traversal */
180:         public void inorder()
181:         {
182:                 inorder(root);
183:         }
184:
185:         private void inorder(SBBSTNode r)
186:         {
187:                 if (r != null)
188:                 {
189:                         inorder(r.left);
190:                         System.out.print(r.data + " ");
191:                         inorder(r.right);
192:                 }
```

```
193:         }
194:
195:         /* Function for preorder traversal */
196:         public void preorder()
197:         {
198:                 preorder(root);
199:         }
200:
201:         private void preorder(SBBSTNode r)
202:         {
203:                 if (r != null)
204:                 {
205:                         System.out.print(r.data + " ");
206:                         preorder(r.left);
207:                         preorder(r.right);
208:                 }
209:         }
210:
211:         /* Function for postorder traversal */
212:         public void postorder()
213:         {
214:                 postorder(root);
215:         }
216:
217:         private void postorder(SBBSTNode r)
218:         {
219:                 if (r != null)
220:                 {
221:                         postorder(r.left);
222:                         postorder(r.right);
223:                         System.out.print(r.data + " ");
224:                 }
225:         }
226:
227:         public static void main(String[] args)
228:         {
229:                 Scanner scan = new Scanner(System.in);
230:                 SelfBalancingBST sbbst = new SelfBalancingBST();
231:                 char ch;
232:
233:                 do
234:                 {
235:                         System.out.println("\nSelfBalancingBST Operations\n");
236:                         System.out.println("1. insert ");
237:                         System.out.println("2. search");
238:                         System.out.println("3. count nodes");
239:                         System.out.println("4. check empty");
240:                         System.out.println("5. clear tree");
```

```
241:
242:                         int choice = scan.nextInt();
243:                         switch (choice)
244:                         {
245:                                 case 1:
246:                                         System.out.println("Enter integer element to insert");
247:                                         sbbst.insert(scan.nextInt());
248:                                         break;
249:                                 case 2:
250:                                         System.out.println("Enter integer element to search");
251:                                         System.out.println("Search result : " + sbbst.search(scan.nextInt()));
252:                                         break;
253:                                 case 3:
254:                                         System.out.println("Nodes = " + sbbst.countNodes());
255:                                         break;
256:                                 case 4:
257:                                         System.out.println("Empty status = " + sbbst.isEmpty());
258:                                         break;
259:                                 case 5:
260:                                         System.out.println("\nTree Cleared");
261:                                         sbbst.clear();
262:                                         break;
263:                                 default:
264:                                         System.out.println("Wrong Entry \n ");
265:                                         break;
266:                         }
267:                         /* Display tree */
268:                         System.out.print("\nPost order : ");
269:                         sbbst.postorder();
270:                         System.out.print("\nPre order : ");
271:                         sbbst.preorder();
272:                         System.out.print("\nIn order : ");
273:                         sbbst.inorder();
274:
275:                         System.out.println("\nDo you want to continue (Type y or n) \n");
276:                         ch = scan.next().charAt(0);
277:                 } while (ch == 'Y' || ch == 'y');
278:                 scan.close();
279:         }
280: }
```

```
 1: package problems;
 2:
 3: import java.util.LinkedList;
 4: import java.util.Queue;
 5:
 6: class MazeNode
 7: {
 8:         int x;
 9:         int y;
10:
11:         public MazeNode(int a, int b)
12:         {
13:                 x = a;
14:                 y = b;
15:         }
16: }
17:
18: class QueueNode
19: {
20:         MazeNode a;
21:         int dist;
22:
23:         public QueueNode(MazeNode a, int dist)
24:         {
25:                 this.a = a;
26:                 this.dist = dist;
27:         }
28: }
29:
30: public class ShortestPathBinaryMaze
31: {
32:         static boolean isValid(int x, int y, int rows, int cols)
33:         {
34:                 if (x >= 0 && y >= 0 && x < rows && y < cols)
35:                         return true;
36:                 return false;
37:         }
38:
39:         static int shortestPath(int[][] maze, MazeNode src, MazeNode dst)
40:         {
41:                 if (maze[src.x][src.y] == 0 || maze[dst.x][dst.y] == 0)
42:                         return -1;
43:
44:                 int rows = maze.length, cols = maze[0].length;
45:                 int adjX[] = {-1, 0, 1, 0}, adjY[] = {0, -1, 0, 1};
46:                 Queue<QueueNode> q = new LinkedList<QueueNode>();
47:                 boolean[][] visited = new boolean[rows][cols];
48:                 q.offer(new QueueNode(src, 0));
```

```
49:                 visited[src.x][src.y] = true;
50:                 while (!q.isEmpty())
51:                 {
52:                         QueueNode temp = q.poll();
53:                         if (temp.a.x == dst.x && temp.a.y == dst.y)
54:                                 return temp.dist;
55:
56:                         for (int i = 0; i < 4; i++)
57:                         {
58:                                 int tempX = temp.a.x + adjX[i];
59:                                 int tempY = temp.a.y + adjY[i];
60:
61:                                 if (isValid(tempX, tempY, rows, cols) && !visited[tempX][tempY] && maze[tempX][tempY] > 0)
62:                                 {
63:                                         q.offer(new QueueNode(new MazeNode(tempX, tempY), temp.dist + 1));
64:                                         visited[tempX][tempY] = true;
65:                                 }
66:                         }
67:                 }
68:                 return -1;
69:         }
70:
71:         public static void main(String[] args)
72:         {
73:                 int maze[][] =
74:                 {
75:                         { 1, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
76:                         { 1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },
77:                         { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
78:                         { 0, 0, 1, 1, 1, 0, 0, 0, 0, 1 },
79:                         { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
80:                         { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
81:                         { 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
82:                         { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
83:                         { 1, 1, 0, 0, 0, 0, 1, 0, 0, 1 }
84:                 };
85:                 System.out.println(shortestPath(maze, new MazeNode(0, 0), new MazeNode(0, 6)));
86:         }
87: }
```

```java
package problems;

import java.util.Arrays;

public class SlidingWindowSum
{
	static int[] slidingWindowSum(int a[], int k)
	{
		if (a == null || a.length == 0)
			return null;
		int win[] = new int[k];
		int sum = 0, i = 0, n = a.length, j = 0;
		int res[] = new int[n - k + 1];
		for (i = 0; i < n; i++)
		{
			sum -= win[i % k];
			win[i % k] = a[i];
			sum += win[i % k];
			if (i >= k - 1)
				res[j++] = sum;
		}
		if (i < k)
			res[j++] = sum;
		return res;
	}

	public static void main(String[] args)
	{
		int[] a = { 1, 3, -1 };
		System.out.println(Arrays.toString(slidingWindowSum(a, 3)));
	}
}
```

```
 1: package problems;
 2:
 3: public class SpiralPrint
 4: {
 5:         static void spiralPrint(int a[][])
 6:         {
 7:                 int i, rowStart = 0, colStart = 0, rowEnd = a.length, colEnd = a[0].length;
 8:
 9:                 while (rowStart < rowEnd && colStart < colEnd)
10:                 {
11:                         for (i = colStart; i < colEnd; ++i)
12:                                 System.out.print(a[rowStart][i] + " ");
13:                         rowStart++;
14:
15:                         for (i = rowStart; i < rowEnd; ++i)
16:                                 System.out.print(a[i][colEnd - 1] + " ");
17:                         colEnd--;
18:
19:                         // to prevent printing first row again in case of single row matrix
20:                         if (rowStart < rowEnd)
21:                         {
22:                                 for (i = colEnd - 1; i >= colStart; --i)
23:                                         System.out.print(a[rowEnd - 1][i] + " ");
24:                                 rowEnd--;
25:                         }
26:                         // to prevent printing first column again in case of single column
27:                         // matrix
28:                         if (colStart < colEnd)
29:                         {
30:                                 for (i = rowEnd - 1; i >= rowStart; --i)
31:                                         System.out.print(a[i][colStart] + " ");
32:                                 colStart++;
33:                         }
34:                 }
35:         }
36:
37:         static void spiralPrintReverse(int a[][])
38:         {
39:                 int sr = a.length / 2, sc = sr, totalCount = (a.length * a[0].length);
40:                 int leftInc = 1, rightInc = 2;
41:                 int count = 0;
42:                 while (count < totalCount)
43:                 {
44:                         for (int i = 0; i < leftInc; i++)
45:                         {
46:                                 System.out.print(a[sr][sc--] + " ");
47:                                 count++;
48:                         }
```

```
49:                         for (int i = 0; i < leftInc && count < totalCount; i++)
50:                         {
51:                                 System.out.print(a[sr--][sc] + " ");
52:                                 count++;
53:                         }
54:
55:                         for (int i = 0; i < rightInc && count < totalCount; i++)
56:                         {
57:                                 System.out.print(a[sr][sc++] + " ");
58:                                 count++;
59:                         }
60:                         for (int i = 0; i < rightInc && count < totalCount; i++)
61:                         {
62:                                 System.out.print(a[sr++][sc] + " ");
63:                                 count++;
64:                         }
65:                         leftInc += 2;
66:                         rightInc += 2;
67:                 }
68:         }
69:
70:         public static void main(String[] args)
71:         {
72:                 int[][] a = { { 1, 2 }, { 1, 2 } };
73:                 spiralPrint(a);
74:                 System.out.println();
75:                 spiralPrintReverse(a);
76:         }
77: }
```

```
    1: package problems;
    2:
    3: import java.util.Scanner;
    4:
    5: public class StrobogrammaticNumber
    6: {
    7:         public static boolean isStrobogrammatic(String n)
    8:         {
    9:                 if (n == null || n.isEmpty())
   10:                         return false;
   11:                 int start = 0, end = n.length() - 1;
   12:                 while (start <= end)
   13:                 {
   14:                         if (n.charAt(start) == n.charAt(end))
   15:                         {
   16:                                 if (isStrobo(n.charAt(start)))
   17:                                 {
   18:                                         start++;
   19:                                         end--;
   20:                                 }
   21:                                 else
   22:                                 {
   23:                                         return false;
   24:                                 }
   25:                         }
   26:                         else
   27:                         {
   28:                                 if (n.charAt(start) == '6' && n.charAt(end) == '9' || n.charAt(start) == '9' && n.charAt(en
d) == '6')
   29:                                 {
   30:                                         start++;
   31:                                         end--;
   32:                                 }
   33:                                 else
   34:                                 {
   35:                                         return false;
   36:                                 }
   37:                         }
   38:                 }
   39:                 return true;
   40:         }
   41:
   42:         static boolean isStrobo(char c)
   43:         {
   44:                 return (c == '8' || c == '0' || c == '1');
   45:         }
   46:
   47:         public static void main(String[] args)
```

```
48:         {
49:                 Scanner sc = new Scanner(System.in);
50:                 String n = sc.next();
51:                 sc.close();
52:                 System.out.println(isStrobogrammatic(n));
53:         }
54: }
```

```java
package problems;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SubPowerSet
{
	static List<List<Integer>> combine(int n, int k)
	{
		List<List<Integer>> result = new ArrayList<List<Integer>>();
		if (k > n || k < 0)
			return result;

		if (k == 0)
		{
			result.add(new ArrayList<Integer>());
			return result;
		}
		result = combine(n - 1, k - 1);
		for (List<Integer> list : result)
			list.add(n);

		result.addAll(combine(n - 1, k));
		return result;
	}

	public static void main(String[] args)
	{
		List<List<Integer>> list = combine(20, 16);
		for (List<Integer> l : list)
			System.out.println(Arrays.toString(l.toArray()));
	}

}
```

```
 1: package problems;
 2:
 3: // this program will compile and run w/o any error regardless of Test1 extending Test (javac Test.java)
 4: // java Test - no output
 5: // java Test1 - hello
 6: // Only constraint - filename should match with public class name.
 7: // A java file can have ony one public class, but it can have multiple classes with one main method each
 8: // On compiling such a class multiple class files are created and they can be run separately
 9:
10: public class Test
11: {
12:
13: }
14:
15: class Test1 extends Test
16: {
17:         public static void main(String[] args)
18:         {
19:                 System.out.println("hello");
20:         }
21: }
```

```
 1: package problems;
 2:
 3: import java.util.Stack;
 4:
 5: public class TopologicalSort
 6: {
 7:         static void topologicalSortUtil(DirectedGraph g, int v, boolean[] visited, Stack<Integer> stack)
 8:         {
 9:                 visited[v] = true;
10:                 for (int n : g.adj[v])
11:                 {
12:                         if (!visited[n])
13:                                 topologicalSortUtil(g, n, visited, stack);
14:                 }
15:                 stack.push(v);
16:         }
17:
18:         static void topologicalSort(DirectedGraph g, int v)
19:         {
20:                 boolean visited[] = new boolean[v];
21:                 Stack<Integer> stack = new Stack<Integer>();
22:
23:                 for (int i = 0; i < v; i++)
24:                 {
25:                         if (!visited[i])
26:                                 topologicalSortUtil(g, i, visited, stack);
27:                 }
28:
29:                 while (!stack.isEmpty())
30:                         System.out.print(stack.pop() + " ");
31:         }
32:
33:         public static void main(String[] args)
34:         {
35:                 DirectedGraph g = new DirectedGraph(6);
36:                 g.addEdge(0, 2);
37:                 g.addEdge(0, 5);
38:                 g.addEdge(1, 5);
39:                 g.addEdge(1, 4);
40:                 g.addEdge(2, 3);
41:                 g.addEdge(3, 4);
42:
43:                 System.out.println("Topological Sort:");
44:                 topologicalSort(g, 6);
45:         }
46: }
```

```
 1: package problems;
 2:
 3: public class Triangle
 4: {
 5:         public static void main(String[] args)
 6:         {
 7:                 int n = 4;
 8:                 for (int i = 0; i < n; i++)
 9:                 {
10:                         for (int j = 1; j < n + i + 1; j++)
11:                         {
12:                                 if (j < n - i)
13:                                         System.out.print(" ");
14:                                 else
15:                                         System.out.print("*");
16:                         }
17:                         System.out.println();
18:                 }
19:         }
20: }
```

```
 1: package problems;
 2:
 3: class TrieNode
 4: {
 5:         TrieNode[] children;
 6:         boolean isLeaf;
 7:
 8:         public TrieNode()
 9:         {
10:                 this.children = new TrieNode[256];
11:         }
12:
13:         public void printChildren()
14:         {
15:                 for (char i = 0; i < 256; i++)
16:                 {
17:                         if (children[i] == null)
18:                                 continue;
19:                         if (children[i].isLeaf)
20:                                 System.out.println(i);
21:                         else
22:                                 System.out.print(i);
23:                         children[i].printChildren();
24:                 }
25:         }
26:
27:         public void printChildren(String prefix)
28:         {
29:                 for (char i = 0; i < 256; i++)
30:                 {
31:                         if (children[i] == null)
32:                                 continue;
33:                         System.out.print(prefix);
34:                         children[i].printChildren();
35:                 }
36:         }
37: }
38:
39: public class Trie
40: {
41:         private TrieNode root;
42:
43:         public Trie()
44:         {
45:                 this.root = new TrieNode();
46:         }
47:
48:         public TrieNode getRoot()
```

```
49:         {
50:                 return this.root;
51:         }
52:
53:         public void printChildren()
54:         {
55:                 TrieNode t = getRoot();
56:                 for (char i = 0; i < 256; i++)
57:                 {
58:                         if (t.children[i] == null)
59:                                 continue;
60:                         System.out.print(i);
61:                         t.children[i].printChildren();
62:                         System.out.println();
63:                 }
64:         }
65:
66:         // Inserts a word into the trie.
67:         public void insert(String word)
68:         {
69:                 TrieNode t = getRoot();
70:                 for (int i = 0; i < word.length(); i++)
71:                 {
72:                         char c = word.charAt(i);
73:                         if (t.children[c] == null)
74:                                 t.children[c] = new TrieNode();
75:                         t = t.children[c];
76:                 }
77:                 t.isLeaf = true;
78:         }
79:
80:         // Returns if the word is in the trie.
81:         public boolean search(String word)
82:         {
83:                 TrieNode t = searchNode(word);
84:
85:                 if (t != null)
86:                         return t.isLeaf;
87:                 else
88:                         return false;
89:         }
90:
91:         // Returns if there is any word in the trie
92:         // that starts with the given prefix.
93:         public boolean startsWith(String prefix)
94:         {
95:                 if (searchNode(prefix) == null)
96:                         return false;
```

```
97:                     else
98:                             return true;
99:             }
100:
101:            public void autoComplete(String s)
102:            {
103:                    TrieNode t = searchNode(s);
104:                    if (t.isLeaf)
105:                            System.out.println(s);
106:                    t.printChildren(s);
107:            }
108:
109:            public TrieNode searchNode(String str)
110:            {
111:                    TrieNode t = getRoot();
112:                    for (int i = 0; i < str.length(); i++)
113:                    {
114:                            char c = str.charAt(i);
115:                            if (t.children[c] != null)
116:                                    t = t.children[c];
117:                            else
118:                                    return null;
119:                    }
120:                    return t;
121:            }
122:
123:            public static void main(String args[])
124:            {
125:                    Trie t = new Trie();
126:                    t.insert("cat");
127:                    t.insert("cater");
128:                    t.insert("base");
129:                    t.insert("basement");
130:                    t.insert("baseline");
131:                    t.printChildren();
132:                    // System.out.println(t.search("cat"));
133:                    // System.out.println(t.search("cate"));
134:                    t.autoComplete("cat");
135:            }
136: }
```

```
 1: package problems;
 2:
 3: import java.util.Arrays;
 4:
 5: public class WiggleSort
 6: {
 7:         static void wiggleSort(int[] nums)
 8:         {
 9:                 for (int i = 1; i < nums.length; i++)
10:                 {
11:                         int a = nums[i - 1];
12:                         if ((i % 2 == 1) == (a > nums[i]))
13:                         {
14:                                 nums[i - 1] = nums[i];
15:                                 nums[i] = a;
16:                         }
17:                 }
18:         }
19:
20:         public static void main(String[] args)
21:         {
22:                 int[] nums = { 1, 2, 3, 4, 5, 6, 7 };
23:                 wiggleSort(nums);
24:                 System.out.println(Arrays.toString(nums));
25:         }
26:
27: }
```