

```
1 #Arrays: Left Rotation
2 def array_left_rotation(a, n, k):
3     r = k % n
4     return a[r:] + a[:r]
5
6 n, k = map(int, raw_input().strip().split(' '))
7 a = map(int, raw_input().strip().split(' '))
8 answer = array_left_rotation(a, n, k);
9 print ' '.join(map(str,answer))
10
11 #Strings: Making Anagrams
12 def number_needed(a, b):
13     cTable = [0]*26
14     for i in a:
15         cTable[ord(i)-97] += 1
16     for i in b:
17         cTable[ord(i)-97] -= 1
18
19     return sum([abs(i) for i in cTable])
20
21 a = raw_input().strip()
22 b = raw_input().strip()
23 print number_needed(a, b)
24
25 #Hash Tables: Ransom Note
26 def ransom_note(magazine, ransom):
27     d = {}
28
29     for i in magazine:
30         if (i not in d):
31             d[i] = 1
32         else:
33             d[i] += 1
34
35     for i in ransom:
36         if (i not in d or d[i] == 0):
37             return False
38         if (i in d):
39             d[i] -= 1
40
41     return True
42
43 m, n = map(int, raw_input().strip().split(' '))
44 magazine = raw_input().strip().split(' ')
45 ransom = raw_input().strip().split(' ')
46 answer = ransom_note(magazine, ransom)
47 if(answer):
48     print "Yes"
49 else:
50     print "No"
51
52 #Linked Lists: Detect a Cycle
53 def has_cycle(head):
54     a = head
55     b = head
56
57     while (True):
58         if (a.next == None): return 0
59
60         a = a.next
61         for i in xrange(2):
62             if (b.next != None): b = b.next
63             else: return 0
64
65         if (a.data == b.data): return 1
```

```
66
67 #Stacks: Balanced Brackets
68 def is_matched(expression):
69     stack = []
70
71     for i in expression:
72         if i == '[':
73             stack.append('[')
74         elif i == '(':
75             stack.append('(')
76         elif i == '{':
77             stack.append('{')
78         elif i in [']', ')', '}']:
79             if len(stack) == 0 or stack.pop() != i: return False
80     return len(stack) == 0
81
82
83
84 t = int(raw_input().strip())
85 for a0 in xrange(t):
86     expression = raw_input().strip()
87     if is_matched(expression) == True:
88         print "YES"
89     else:
90         print "NO"
91
92
93 #Queues: A Tale of Two Stacks
94 class MyQueue(object):
95     def __init__(self):
96         self.labels = ['first', 'second']
97         self.cur = 0
98         self.first = []
99         self.second = []
100
101     def peek(self):
102         if (not len(self.first) and not len(self.second)): return
103
104         if (not len(self.second)):
105             while (len(self.first) > 0):
106                 self.second.append(self.first.pop())
107
108         res = self.second[-1]
109
110         return res
111
112     def pop(self):
113         if (not len(self.first) and not len(self.second)): return
114
115         if (not len(self.second)):
116             while (len(self.first) > 0):
117                 self.second.append(self.first.pop())
118
119         res = self.second.pop()
120
121         return res
122
123     def put(self, value):
124         self.first.append(value)
125
126
127
128
129
130
```

```
131
132 queue = MyQueue()
133 t = int(raw_input())
134 for line in xrange(t):
135     values = map(int, raw_input().split())
136
137     if values[0] == 1:
138         queue.put(values[1])
139     elif values[0] == 2:
140         queue.pop()
141     else:
142         print queue.peek()
143
144
145 #Trees: Is this a binary search tree?
146 def check_binary_search_tree_(root):
147     def traverse(node, low, high):
148         if ((low == None or node.data > low) and (high == None or node.data < high)):
149             left = True if node.left == None else traverse(node.left, low, node.data)
150             right = True if node.right == None else traverse(node.right, node.data, high)
151
152             return left and right
153         else: return False
154     return traverse(root, None, None)
155
156
157 def bfs(graph, start):
158     visited, queue = set(), [start]
159     while queue:
160         vertex = queue.pop(0)
161         if vertex not in visited:
162             visited.add(vertex)
163             queue.extend(graph[vertex] - visited)
164     return visited
165
166
167
168
```