Contents

1 Data Structure

1.1 SegTree Class

```
import sys
import collections

class segTree:
    def __init__(self, x):
        self.n = 1
        while self.n < x:
            self.n = self.n * 2
        self.data = [0 for i in range(self.n*2+10)]

    def query(self,l,r,node,nodeL,nodeR):
        if r < nodeL or l > nodeR:
            return 0
        if l <= nodeL and nodeR <= r:
            return self.data[node]
        val1 = self.query(l,r,node*2,nodeL,(nodeL+nodeR)//2)
        val2 = self.query(l,r,node*2+1,(nodeL+nodeR)//2+1,nodeR)
        return val1 + val2

    def update(self,p,x):
        p = p + self.n - 1
        self.data[p] = x
        p = p//2
        while p > 0:
            self.data[p] = self.data[p*2] + self.data[p*2+1]
            p = p // 2


N,M,K = [int(x) for x in sys.stdin.readline().split() ]
tree = segTree(N)
for i in range(1,N+1,1):
    x = int(sys.stdin.readline())
    tree.update(i,x)

for i in range(M+K):
    a,b,c = [int(x) for x in sys.stdin.readline().split() ]
    if a == 1:
        tree.update(b,c)
    else:
        if b > c :
            swap(b,c)
        print(tree.query(b,c,1,1,tree.n))
```

1.2 BIT

```
import sys
import collections


N,M,K = [int(x) for x in sys.stdin.readline().split() ]
data = [0] * (N+20)

def update(p,x):
    original = query(p,p)
    diff = x - original
    while p <= N:
        data[p] = data[p] + diff
        p = p + (p&-p)

def query(l,r):
    res = 0
    l = l - 1
    while r > 0:
        res = res + data[r]
        r = r - (r&-r)
    while l > 0 :
        res = res - data[l]
        l = l - (l&-l)

    return res

for i in range(1,N+1,1):
    x = int(sys.stdin.readline())
    update(i,x)


for i in range(M+K):
    a,b,c = [int(x) for x in sys.stdin.readline().split() ]
    if a == 1:
        update(b,c)
    else:
        if b > c :
            swap(b,c)
        print(query(b,c))
```

2 String

2.1 KMP

```
import sys

def preprocess(p):
    n = len(p)
    fail = [0] * n
    j = 0
    for i in range(1,n,1):
        while j > 0 and p[i] != p[j] :
            j = fail[j-1]
        if p[i] == p[j]:
            j = j + 1
            fail[i] = j
        else:
            fail[i] = 0

    return fail

def KMP(s,p):
    n = len(s)
    m = len(p)
    j = 0
    ans = []
    fail = preprocess(p)
    for i in range(n):
        while j > 0 and s[i] != p[j]:
            j = fail[j-1]
        if s[i] == p[j]:
            if j == m-1:
                ans.append(i-m+1)
                j = fail[j]
            else:
                j = j + 1

    return ans

s = str(sys.stdin.readline().strip('\n'))
p = str(sys.stdin.readline().strip('\n'))

ans = KMP(s,p)

print(len(ans))
for x in ans:
    print(x+1,end='')
    print(" ",end='')
```

3 Graph

3.1 Bipartite Matching

```
import sys
import collections


N,M = [int(x) for x in sys.stdin.readline().split()]
vis = [0 for i in range(N+10) ]
xy = [0 for i in range(N+10) ]
yx = [0 for i in range(M+10) ]
E = [ [] for i in range(N+2) ]

def dfs(x):
    vis[x] = True
    for e in E[x]:
        if yx[e] == 0 or ( vis[yx[e]] == False and dfs(yx[e]) == 1 ):
            yx[e] = x
            xy[x] = e
            return 1
    return 0


for i in range(1,N+1,1):
    arr = [int(x) for x in sys.stdin.readline().split()]
    n = len(arr)
    for j in range(1,n,1):
        E[i].append(arr[j])

ans = 0
for i in range(1,N+1,1):
    for j in range(N+10):
        vis[j] = 0
    ans = ans + dfs(i)

print(ans)
```