# 復習済み

A-F

# A Lucky Direction

正解

# B Seek Grid

正解

# C Pigeonhole Query

正解

## メモ

改善の余地あり

最初の正解コードはiの巣に何羽がいるかの数字だけを管理していればいいのに、具体的な鳥の番号も保存しているから管理が複雑になっている。

よって改善したコードも添付する。

実際1000msくらいが500msくらいになった。

## 正解コード

def solve():

N, Q = map(int, input().split())

nest = {}

tori = {}

for i in range(N):

nest[i] = [i]

tori[i] = i

two = set()

for i in range(Q):

q = input().split()

if q[0] == "1":

P = int(q[1])-1

H = int(q[2])-1

if len(nest[tori[P]]) > 2:

nest[tori[P]].remove(P)

nest[H].append(P)

tori[P] = H

if len(nest[H]) == 2:

two.add(H)

elif len(nest[tori[P]]) == 2:

nest[tori[P]].remove(P)

two.remove(tori[P])

nest[H].append(P)

tori[P] = H

if len(nest[H]) == 2:

two.add(H)

else:

nest[tori[P]].remove(P)

nest[H].append(P)

tori[P] = H

if len(nest[H]) == 2:

two.add(H)

else:

print(len(two))

solve()

## 改善コード

n, Q = map(int, input().split())

nest = [1]\*(n+1)

tori = [i for i in range(n+1)]

more\_two\_nest = 0

for i in range(Q):

q = input().split()

if q[0] == "1":

p = int(q[1])

h = int(q[2])

if nest[tori[p]] == 2:

more\_two\_nest -= 1

nest[tori[p]] -= 1

if nest[h] == 1:

more\_two\_nest += 1

nest[h] += 1

tori[p] = h

else:

print(more\_two\_nest)

# D Gravity

不正解

## メモ

解法としては1秒ずつシミュレーションするのではなく、各ブロックがいつ消えるのかを検査するのがポイント。そうすればあるTの時にブロックがあるのかわかる。

## 正解コード

N, W = map(int, input().split())

column = [[] for \_ in range(W)]

deadtime = [None]\*N

T = 0

for i in range(N):

x, y = map(int, input().split())

x -= 1

y -= 1

column[x].append([y, i])

for w in range(W):

column[w].sort()

column\_count = [len(column[w]) for w in range(W)]

column\_id = [0]\*W

while True:

flag = False

max\_d = 0

for w in range(W):

if column\_id[w] < column\_count[w]:

max\_d = max(max\_d, column[w][column\_id[w]][0]-T)

else:

flag = True

break

if flag:

break

T += max\_d+1

for w in range(W):

deadtime[ column[w][column\_id[w]][1] ] = T

column\_id[w] += 1

Q = int(input())

for \_ in range(Q):

t, a = map(int, input().split())

a -= 1

if deadtime[a] == None:

print("Yes")

continue

if deadtime[a] <= t:

print("No")

else:

print("Yes")

# E Hierarchical Majority Vote

正解

自分が正解したコードでも十分速いが、欠点はメモリを多く使用していること。再帰で返していくようにすればそこまでかからない。

## 正解コード

def solve(S, N, i):

if N == 1:

if S[0][i\*3:(i+1)\*3].count(S[1][i]) == 3:

return 2

else:

return 1

else:

zero\_one = S[N][i]

tmp = []

for j in range(3):

if S[N-1][i\*3+j] == zero\_one:

tmp.append(solve(S, N-1, i\*3+j))

return sum(tmp)-max(tmp)

N = int(input())

A = input()

S = []

S.append(list(A))

for i in range(N):

tmp = []

buf = S[i]

for j in range(0, len(buf), 3):

zero = 0

for k in range(3):

if buf[j+k] == "0":

zero += 1

if zero >= 2:

tmp.append("0")

else:

tmp.append("1")

S.append(tmp)

print(solve(S, N, 0))

## 改善コード

def solve(l, r):

if l + 1 == r:

return A[l], 1

d = (r-l)//3

v1, c1 = solve(l, l+d)

v2, c2 = solve(l+d, l+2\*d)

v3, c3 = solve(l+2\*d, r)

if v1 == v2 == v3:

return v1, sum([c1, c2, c3])-max(c1, c2, c3)

elif v1 == v2:

return v1, min(c1, c2)

elif v2 == v3:

return v2, min(c2, c3)

elif v3 == v1:

return v3, min(c3, c1)

else:

print("error")

exit()

N = int(input())

A = input()

print(solve(0, 3\*\*N)[1])

# F K-th Largest Triplet

不正解

## メモ

(正解コード1 (2802ms / 3000ms) )

自分で解いて解説と同じアルゴリズムを考えられたが個のアルゴリズム自体がギリギリだった。

(正解コード2 1889ms)

i, j, kから3つの組み合わせをヒープに追加できるがそうすると重複する。そのため、ある組み合わせに到達するルートを一つにすることで重複することなくK回取り出す操作で答えに辿り着く。

(正解コード３ 575ms)

考え方はある値(mid)以上になる組み合わせがK個以上存在するかどうかで判定をして二分探索をする。最終的にokの値以上はK個以上存在し、ngの値以上はK個ないので、okの値が答えになる。

この判定がポイント。A, B, Cを3重ループで回す。この時、これ以上後ろで組み合わせても、mid以上の組み合わせがないのであれば、breakして次を探索する形で数え上げを高速化している。

## 正解コード1(ヒープ工夫なし) 2803ms

from heapq import heapify, heappop, heappush

def cal(i, j, k):

return -(A[i]\*B[j]+B[j]\*C[k]+C[k]\*A[i])

N, K = map(int, input().split())

A = list(map(int, input().split()))

B = list(map(int, input().split()))

C = list(map(int, input().split()))

A.sort()

B.sort()

C.sort()

cor\_set = set()

cor\_set.add((N-1, N-1, N-1))

q = [(cal(N-1, N-1, N-1), N-1, N-1, N-1)]

heapify(q)

for \_ in range(K-1):

\_, i, j, k = heappop(q)

if i != 0:

n\_cor = (i-1, j, k)

if n\_cor not in cor\_set:

cor\_set.add(n\_cor)

heappush(q, (cal(i-1, j, k), i-1, j, k))

if j != 0:

n\_cor = (i, j-1, k)

if n\_cor not in cor\_set:

cor\_set.add(n\_cor)

heappush(q, (cal(i, j-1, k), i, j-1, k))

if k != 0:

n\_cor = (i, j, k-1)

if n\_cor not in cor\_set:

cor\_set.add(n\_cor)

heappush(q, (cal(i, j, k-1), i, j, k-1))

ans, \_, \_, \_ = heappop(q)

print(-ans)

## 正解コード2(ヒープ工夫あり)

from heapq import heappop, heappush

N, K = map(int, input().split())

A = sorted(list(map(int, input().split())), reverse=True)

B = sorted(list(map(int, input().split())), reverse=True)

C = sorted(list(map(int, input().split())), reverse=True)

hq = []

def add(i, j, k):

if i < N and j < N and k < N:

heappush(hq, (-(A[i] \* B[j] + B[j] \* C[k] + C[k] \* A[i]), i, j, k))

add(0, 0, 0)

for t in range(K):

val, i, j, k = heappop(hq)

if t == K - 1:

print(-val)

if j==0 and k==0: add(i + 1, j, k)

if k==0: add(i, j + 1, k)

add(i, j, k + 1)

## 正解コード3(二分探索1)

def f(a, b, c):

return a\*b+b\*c+c\*a

def check(mid):

cnt = 0

for i in range(N):

if f(A[i], B[0], C[0]) < mid:

break

for j in range(N):

if f(A[i], B[j], C[0]) < mid:

break

cnt += 1

for k in range(1, N):

if f(A[i], B[j], C[k]) >= mid:

cnt += 1

else:

break

if cnt >= K:

return True

return False

N, K = map(int, input().split())

A = sorted(list(map(int, input().split())), reverse = True)

B = sorted(list(map(int, input().split())), reverse = True)

C = sorted(list(map(int, input().split())), reverse = True)

ok = 3

ng = 3\*10\*\*18+1

while ok + 1 != ng:

mid = (ok+ng)//2

if check(mid): #mid以上がK個以上存在する

ok = mid

else:

ng = mid

print(ok)

# G Many LCS

不正解

## メモ

## 正解コード