

**ՃԱՐՏԱՐԱՊԵՏՈՒԹՅԱՆ ԵՒ ՇԻՆԱՐԱՐՈՒԹՅԱՆ ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՀԱՄԱԼՍԱՐԱՆ**

**Ինֆորմատիկայի, հաշվողական տեխնիկայի և կառավարման համակարգերի ամբիոն**

**Կառավարման և տեխնոլոգիայի ֆակուլտետ**

**ԿՈՒՐՍԱՅԻՆ**

**ՀԿ-32 խմբի ուսանող**

**Գրիգորյան Տիգրան**

**Ծրագրավորում**

**(C++)**

**(Թեմա)**

**Ղեկավար՝ Գրիգորյան Ա.**

**(Ազգանուն Ա., Հ.) (Ստորագրություն)**

**Լամպեր** *(Ժամանակը` 2 վրկ. Հիշողություն՝ 16 ՄԲ Բարդություն՝ 94%)*

Կա N լամպերից բաղկացած շարք, որոնք համարակալված են 1-ից N։ Սկզբնական պահին ոչ մի լամպ չի վառվում։ Այնուհետև տեղի են ունենում այս շարքի K հերթական գծային ինվերսիաներ։ Գծային ինվերսիայի ընթացքում տեղի է ունենում յուրաքանչյուր P-րդ լամպի ինվերսիա շարքում։ Օրինակ, եթե P=3, ապա ինվերսիա կլինի 3-րդ, 6-րդ, 9-րդ և այդպես շարունակ լամպերի համար։

Անհրաժեշտ է որոշել՝ քանի վառվող լամպ կմնա բոլոր նշված գծային ինվերսիաների իրականացումից հետո։

**Մուտքային տվյալներ**

Մուտքային ֆայլի (INPUT.TXT) առաջին տողում տրված են N և K թվերը՝ լամպերի քանակը և գծային ինվերսիաների քանակը։ Երկրորդ տողում տրված են K ամբողջ թվեր՝ *Pi*, որոնք նշում են տվյալ ինվերսիաների պարբերականությունը։ (1 ≤ N ≤ 109, 1 ≤ K ≤ 100, 1 ≤ *Pi* ≤ 50)

**Ելքային տվյալներ**

Ելքային ֆայլում (OUTPUT.TXT) անհրաժեշտ է տպել խնդրի պատասխանը։

**Լուծում**

### **Տվյալների մուտքագրում (Input)**

Ծրագիրը կարդում է մուտքային տվյալները INPUT.TXT ֆայլից.

### **Հիմնական բաղադրիչներ**

#### **ԳՀԲ (GCD) և ՓՀԲ (LCM)**

1. **gcd(a, b) ֆունկցիան**՝ հաշվում է երկու թվերի մեծագույն ընդհանուր բաժանարարը (ԳՀԲ), ինչը օգնում է փոքրագույն ընդհանուր բազմապատիկը (ՓՀԲ) արդյունավետորեն հաշվելու համար։
2. **lcm(a, b, number\_of\_bulbs) ֆունկցիան**՝ հաշվում է երկու թվերի ՓՀԲ-ն, բայց միայն այն դեպքում է վերադարձնում այն, եթե ՓՀԲ-ն փոքր է կամ հավասար լամպերի ընդհանուր քանակին։ Սա երաշխավորում է, որ արդյունքը խնդիրին համատեքստային է։

#### **Ինվերսիաների հետևում**

* Ծրագիրը պահում է ինվերսիաների վիճակը ցուցակի (effective\_inversions) միջոցով, որը բաղկացած է 50 տրամաբանական արժեքից (բուլյան)՝ յուրաքանչյուր ինվերսիայի տեսակին համապատասխան։
* Եթե ինվերսիայի որոշակի տեսակ հանդիպում է, նրա բուլյան արժեքը փոխվում է՝ False → True (կամ հակառակը)։ Սա ցույց է տալիս, որ տվյալ ինվերսիան ակտիվ է։

#### **Ինվերսիաների համակցում**

* **merge(divisioner, coef) ֆունկցիան** թարմացնում է տվյալների բառարանը՝ map\_data։  
  Այս բառարանը հետևում է, թե լամպերի որ դիրքերն են ազդում յուրաքանչյուր ինվերսիայից (կամ ինվերսիաների համակցությունից) և պահում է արդյունքը որպես գործակից (coef)։
* Եթե գործակիցը զրո է դառնում, ինվերսիայի գրառումը հեռացվում է։

### **Գլխավոր ցիկլ**

1. **Ակտիվ ինվերսիաների մշակում**

Ծրագիրը ստուգում է effective\_inversions ցուցակի յուրաքանչյուր ինվերսիա։ Եթե այն ակտիվ է, ապա.

* Անցնում է map\_data-ի պահված ինվերսիաների վրա և դրանց հետ հաշվարկում ՓՀԲ՝ օգտագործելով lcm() ֆունկցիան։
* Սա օգնում է որոշել, թե որ լամպերն են ազդում մի քանի ինվերսիաների համակցությամբ։

1. **Արդյունքների թարմացում**

* **merge()** ֆունկցիան օգտագործվում է թարմացնելու համար, թե որ լամպերն են ազդվում նոր ինվերսիայի կամ նրա համակցությունների արդյունքում։

### **Վերջնական հաշվարկ**

1. Վերջում ծրագիրը հաշվարկում է, թե քանի լամպ կմնան միացված՝ օգտագործելով map\_data բառարանը։
   * Բանալին (key) ներկայացնում է լամպերի դիրքերը։
   * Արժեքը (value) ցույց է տալիս, թե քանի այդպիսի լամպ են դեռ միացված։
2. Վերջնական արդյունքը գրվում է OUTPUT.TXT ֆայլի մեջ։

Այսպիսով, ծրագիրը արդյունավետորեն հաշվում է լամպերի վիճակը՝ հաշվի առնելով բոլոր ինվերսիաների ազդեցությունները։

from pathlib import Path

def merge(divisioner: int, coef: int) -> None:

already\_present = map\_data.get(divisioner)

if already\_present:

coef = already\_present + coef

if coef == 0:

del map\_data[divisioner]

else:

map\_data[divisioner] = coef

else:

map\_data[divisioner] = coef

def gcd(a: int, b: int) -> int:

while b != 0:

a, b = b, a % b

return a

def lcm(a: int, b: int, number\_of\_bulbs: int) -> int:

res = (a // gcd(a, b)) \* b

if res <= number\_of\_bulbs:

return res

return None

map\_data = {}

def main() -> None:

with Path("INPUT.TXT").open("r") as f:

number\_of\_bulbs, \_ = map(int, f.readline().split())

inversions\_list = list(map(int, f.readline().split()))

effective\_inversions = [False] \* 50

for inversion in inversions\_list:

effective\_inversions[inversion - 1] = not effective\_inversions[inversion - 1]

for idx, inversion in enumerate(effective\_inversions):

if not inversion:

continue

nat\_inversion = idx + 1

size = len(map\_data)

keys = list(map\_data.keys())

values = list(map\_data.values())

for i in range(size):

lcm\_num = lcm(keys[i], nat\_inversion, number\_of\_bulbs)

if lcm\_num and lcm\_num <= number\_of\_bulbs:

ss = -2 \* values[i]

merge(lcm\_num, ss)

merge(nat\_inversion, 1)

on\_bulbs = 0

for key, value in map\_data.items():

if value == 0:

continue

on\_bulbs += (number\_of\_bulbs // key) \* value

Path("OUTPUT.TXT").write\_text(str(on\_bulbs))

if \_\_name\_\_ == "\_\_main\_\_":

main()