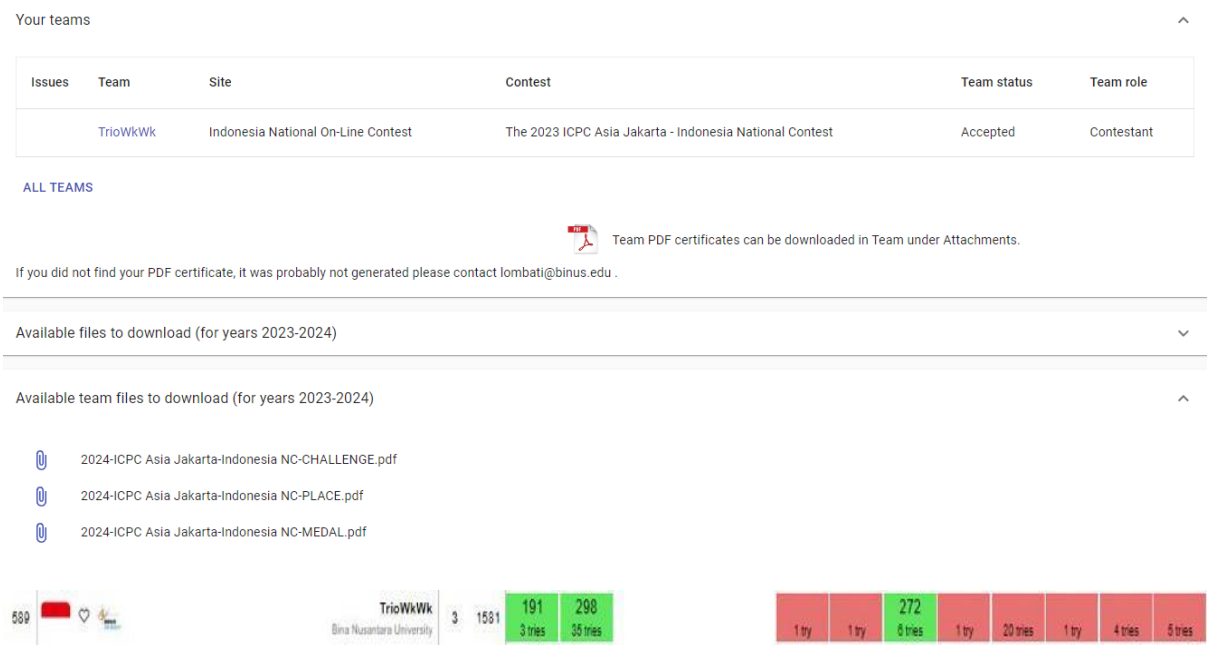
Kelompok 15 ICPC

Anggota Kelompok :

1. BENEDICT ZEVANNO CHRISTABEL - 2602056851
2. EVANDER OCTAVIANUS LAYARDI - 2602075983
3. JEFFLINE KRISTIAN DJAYA - 2602077156



Problem A analysis (Solved)

We will maintain a data structure which contains all of invited institutions. Initialize it with the top M teams. Then, iterate the remaining teams based on their rank. For each team, if the data structure doesn't contain their institution and we haven't added K new institutions, we add the institution into the data structure and the team name into the answer. Depending on the data structure used, the time complexity could be either O(N2) or O(N log N). Both are sufficient to solve this problem.

The key points from the problem description:

- There are N teams participating in INC 2023, each ranked from 1 to N.
- Teams ranked in the top M are eligible to participate in The ICPC Asia Jakarta 2023.

- Moreover, at least K gold tickets will be awarded to the top-ranked teams from institutions not included in the top M institutions.
- Only one gold ticket is given to each institution, and it is awarded to the highest-ranked team in INC 2023 from each qualifying institution.
- To solve the problem, input consisting of N, M, and K needs to be read, followed by information about the rankings and institutions for each team.
- Identify the qualifying teams for gold tickets and display the output, including the number of teams that receive gold tickets along with their names. The output should be sorted based on team rankings.

Problem B analysis (Solved)

Let's fix a day i. Observe that the optimal strategy that minimizes the milk we drink is to replace the largest min(i, K) milk with biscuit. Thus, checking whether it is possible to maintain the diet until the i-th day can be done in O(N log N): Sort the milks from day 1 to i, remove the largest min(i, K) milk, and check whether the remaining milks' sum is not larger than M. As we need to check for n + 1 days (from 0 to n), the time complexity will be O(N2 log N). Note that it is possible to optimize above solution into O(N log N), e.g. by using priority queue. It is not necessary to solve this problem.

Problem H analysis (Solved)

Sort the treasures in ascending order based on their weight. Similarly, sort the carts in ascending order based on the maximum weight they could carry. Iterate the carts in ascending order. We will maintain a max priority queue which stores the treasure that can be carried by the current cart, ordered by their value. Observe that the optimal strategy is to carry the most valuable treasure we have right now in the current cart. After that, we remove the most valuable treasure from the priority queue and continue on to the next cart. Do this until we have processed all carts. The time complexity will be O(N log N + M log M).

Problem E analysis (Unsolved)

This problem can be solved greedily by repeatedly finding the shortest prefix p of the string S such that p is also the suffix of the string T. The shortest prefix will then become part of the split string. We will show why this greedy algorithm works. Suppose there is a solution using a longer prefix q (i.e. $|q| > |p|$). We can see that p is a prefix of q and is also a suffix of q. It must be that $|q| \geq 2|p|$, otherwise there is a shorter prefix that is also a suffix of p. Since $|q| \geq 2|p|$, we can write q = p+r+p for some (possibly empty) string r, where + is concatenation. This implies that we can further split q into p, r, and p, whose reverse order is also p, r, and p. The shortest prefix that is also a suffix of a string can be found using hashing. Hence, this problem can be solved in O(N) time complexity.

Problem M analysis (Unsolved)

We say a bar is split by row i if an (n × m) chocolate bar is split into (i × m) and ((n − i) × m). Split by column is defined similarly.

The solution to this problem always requires three or fewer operations.

If K is exactly N x M, then the answer is 0.

If K is divisible by either N or M, then the answer is 1. More precisely, if K is divisible by N, we can split by column

Next, we check whether 2 operations is possible. We will try all possibilities of splitting by row and column. Note that two splits by the same axes (either row or column) are redundant so we do not have to consider those cases.

• Try to do the first split on every possible row i. We will eat the (i × M) bar and further split the other bar. Next, if K - i x M is positive, check if (K – i × M) is divisible by (N − i). If yes, we can split by column and the answer is 2.

• Try to do the first split on every possible column j. We will eat the (N × j) bar and further split the other bar. Next, if K - N x j is positive, check if (K – N × j) is divisible by (M – j). If yes, we can split by row and the answer is 2.

If none of the above is true, then we show that 3 operations is always possible. We can write K = x x M + y (for some non-negative integer x and y), then we can always split the chocolate bar by row x and x+1, then the single-row chocolate bar is split by column y.

Explanation and Source Code

Problem A

```
N, M, K = map(int, input().split())
Teams = {}


for i in range (N):
        Team, institution = input().split()
        rank = i + 1
        if institution in Teams:
                if rank < Teams [institution] [0]:
                        Teams [institution] = (rank,
                Team)
        else:
                Teams [institution] = (rank, Team)
GoldenTicket = []
for institution, (rank, Team) in Teams.items():
        if rank > M and len (GoldenTicket) <K:
                GoldenTicket.append((rank, Team))
GoldenTicket.sort()
print (len (GoldenTicket))
for rank, Team in GoldenTicket:
print (Team)
```

Problem A

1. Input Reading :

- N, M, K = map(int, input().split()): Reads three integers (N, M, K) from input, representing the number of teams, a threshold rank, and a limit of teams to be selected.

2. Dictionary Initialization :

- Teams = {}: Initializes an empty dictionary to store team information.

3. Loop to Input Team Information :

- for i in range(N):: Iterates N times to collect team and institution information.
- Team, institution = input().split(): Reads the team name and institution from input.
- rank = i + 1: Calculates the rank of the team based on the loop index i.
- Checks if the institution already exists in the Teams dictionary:

  - if institution in Teams:: Checks if the institution is already present in the dictionary.

  - If it exists:

  - if rank < Teams[institution]:: Compares the current rank with the stored rank for the institution.

  - Teams[institution] = (rank, Team): Updates the team and its rank if the current rank is lower (better) than the previously stored rank.

  - If it doesnt exist:

  - Teams[institution] = (rank, Team): Adds the institution with its rank and team to the dictionary.

4. Selection of Teams for GoldenTicket :

- GoldenTicket = []: Initializes an empty list to store teams that qualify for the GoldenTicket.
- Iterates through the items in the Teams dictionary :

  - for institution, (rank, Team) in Teams.items():: Checks if the rank of the team is greater than the threshold rank M and the number of teams selected for the GoldenTicket is less than K.

  - if rank > M and len(GoldenTicket) < K:

  - GoldenTicket.append((rank, Team)): Appends the team to the 'GoldenTicket' list if it meets the criteria.
- Sorts the GoldenTicket list based on ranks :

  - GoldenTicket.sort()
- Prints the count of teams that received the GoldenTicket :

  - print(len(GoldenTicket))
- Prints the teams that received the GoldenTicket :

for rank, Team in GoldenTicket : print(Team)

Visualisasi Step code :

- Ambil input dari pengguna untuk N = 3, M = 1, dan K = 2, yaitu jumlah tim, batas peringkat institusi yang memenuhi syarat untuk ICPC, dan jumlah tiket emas yang tersedia.

- Kemudian melakukan Inisialisasi Struktur Data Teams yaitu Tim pertama, ARUA UOGX (peringkat 1), dimasukkan ke dalam struktur data Teams, Tim kedua, NOISH UHOLO (peringkat 2), juga dimasukkan ke dalam struktur data Teams, Tim ketiga, IKUBUF UHOLO (peringkat 3), juga dimasukkan ke dalam struktur data Teams.

- Kemudian dilakukan pengulangan / Loop untuk Tiket Emas dimana iterasi melalui struktur data Teams untuk mencari tim yang memenuhi syarat untuk tiket emas dimana syaratnya mengecek apakah institusi tidak termasuk dalam M institusi teratas dan belum mendapatkan tiket emas sebelumnya

- dan hanya satu tiket emas yang dapat diberikan kepada institusi UHOLO. Karena itu, tim dengan peringkat tertinggi yang memenuhi syarat, yaitu NOISH UHOLO, dimasukkan ke dalam daftar tiket emas.

  Kemduian dilakukan Sort dan Print Hasil yaitu u rutkan daftar tiket emas berdasarkan peringkat tim dan cetak jumlah tim yang memperoleh tiket emas 1 dan nama tim tersebut NOISH.

Problem B

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

# Read the list of items.
P = list(map(int, input().split()))
for _ in range(K):
        totalMilk = 0
        highestMilk = 0
        highestMilkIndex = -1
        for i in range (N):
                if P[i]== 0: continue
                if P[i] >= highestMilk:
                        highestMilk = P[i]
                        highestMilkIndex = i
                if totalMilk + P[i]> M: break
                totalMilk += P[i]
        P[highestMilkIndex] = 0

counter = 0
totalMilkFinal = 0
for i in range (N):
        if totalMilkFinal + P[i] > M: break
        totalMilkFinal += P[i]
        counter += 1

print(counter)
```

Problem B

1. The code reads three integers (N, M, K) from input, representing the number of items, maximum milk capacity, and a parameter K.
2. It reads a list of integers representing the milk quantity associated with each item.
3. The code executes K iterations of the following :
   - It initializes a variable to keep track of the total milk distributed in each iteration.
   - It initializes variables to keep track of the item index and quantity with the highest milk content.
   - It loops over the items and finds the index of the item with the highest milk content.
   - It checks if distributing the current item's milk would exceed the maximum milk capacity. If so, it breaks the loop.
   - It adds the milk quantity of the current item to the total distributed milk and marks the item with the highest milk content as already distributed by setting its quantity to 0.
   - It initializes counters for the final distribution calculation.
   - It loops over the items and counts the number of items distributed within the maximum capacity.
   - It outputs the count of items distributed within the maximum capacity in each iteration.

Simulasi Hitungan nya :

Dengan input = 7 (N), 70 (M) Dan 1 (K) DAN Kebutuhan susu 70, 30, 40, 20, 50, 10, 60

1. Iterasi Pertama (K) :
   - Pada Hari ke-1 (i=1) :
   - Total kebutuhan susu (totalMilk): 70.
   - Kondisi totalMilk > M terpenuhi, dan karena masih ada 1 biskuit, maka biskuit dikonsumsi. Jumlah hari yang dapat dijalani (counter) bertambah menjadi 1.

2. Iterasi Kedua (K) :
   - Pada Hari ke-2 (i=2) :
   - Total kebutuhan susu (totalMilk) bertambah menjadi 100 (30 + 40).
   - Kondisi totalMilk > M terpenuhi, dan karena masih ada 1 biskuit, maka biskuit dikonsumsi. Jumlah hari yang dapat dijalani (counter) bertambah menjadi 2.

3. Iterasi Ketiga (K) :

- Pada Hari ke-3 (i=3) :
- Total kebutuhan susu (totalMilk) bertambah menjadi 70 (40 + 30).
- Kondisi `totalMilk > M terpenuhi, dan karena masih ada 1 biskuit, maka biskuit dikonsumsi. Jumlah hari yang dapat dijalani (counter) bertambah menjadi 3.

4. Iterasi Keempat (K) :

- Pada Hari ke-4 (i=4) :
- Total kebutuhan susu (totalMilk) bertambah menjadi 90 (20 + 70).
- Kondisi totalMilk > M terpenuhi, tetapi karena biskuit sudah habis (K=0), iterasi berakhir pada Hari ke-4. Jumlah hari dengan konsumsi susu (counter) adalah 3 (sesuai dengan simulasi diet).

Problem E

```
n = int(input())
str_input = input()
q = int(input())

for _ in range(q):
        s = input()
        t1, t2 = " ", " "
        found False

        for i, j in zip (range(n), range(len(str_input) - 1, -1, -1)):
                t1 += s[i]
                t2 = str_input[j] + t2

                if t1 == t2:
                        found = True
                        t2 = " "
                        t1 = " "
                else:
                        found = False

print("YES" if found else "NO")
```

Problem E

This problem can be solved greedily by repeatedly finding the shortest prefix p of the string S such that p is also the suffix of the string T. The shortest prefix will then become part of the split string.

We will show why this greedy algorithm works. Suppose there is a solution using a longer prefix q (i.e. $|q| > |p|$). We can see that p is a prefix of q and is also a suffix of q. It must be

that |q| ≥ 2|p|, otherwise there is a shorter prefix that is also a suffix of p. Since |q| ≥ 2|p|, we can write q = p+r+p for some (possibly empty) string r, where + is concatenation. This implies that we can further split q into p, r, and p, whose reverse order is also p, r, and p.

The shortest prefix that is also a suffix of a string can be found using hashing. Hence, this problem can be solved in O(N) time complexity.

Penjelasan visualisasi code :

- q 1 -> A, Dimana dalam algoritma code yaitu t1 (A) dihasilkan dari string s yaitu input , karena sama maka YES
- q 2 - > B, yaitu t2 (B) tidak dihasilkan dari string s yaitu A, maka masuk ke else dengan menghaislkan output NO. karena panjang string berbeda

Problem H

```
import heapq
N, M = map(int, input().split())
A = [tuple(map(int, input().split())) for _ in range(N)]


# Sort the treasures based on their weight
A.sort()
# Sort the carts based on their maximum weight capacity
S = list(map(int, input().split()))
S.sort()
pq = []  # Priority queue to store treasures that can be carried by the current cart
j = 0    # Index to iterate through the sorted treasures
res = 0  # Variable to store the total value of treasures carried
# Iterate through each cart
for i in range(M):
    # Add treasures that can be carried by the current cart to the priority queue
    while j < N and A[j][0] <= S[i]:
        heapq.heappush(pq, -A[j][1])  # Use negative values for max heap behavior
        j += 1
    # Check if there are treasures in the priority queue
    if pq:
        res += -heapq.heappop(pq)  # Use negative values for max heap behavior
# Print the total value of treasures carried by all carts
print(res)
```

Problem H

1. Input Parsing :

- N, M = map(int, input().split()): Takes two integers N and M as input, where N is the number of treasures and M is the number of carts.
- A = [tuple(map(int, input().split())) for _ in range(N)]: Takes input for the treasures, where each treasure is represented by a tuple of two integers - weight and value.

2. Sorting :

- A.sort(): Sorts the list of treasures based on their weights in ascending order.
- S = list(map(int, input().split())): Takes input for the carts' maximum weight capacity.
- S.sort(): Sorts the list of carts based on their maximum weight capacity in ascending order.

3. Priority Queue and Main Loop :

- pq = []: Initializes an empty priority queue to store treasures that can be carried by the current cart.
- j = 0: Initializes an index to iterate through the sorted treasures.
- res = 0: Initializes a variable to store the total value of treasures carried.

4. Main Loop :

- Iterates through each cart (for i in range(M)).
- Adds treasures that can be carried by the current cart to the priority queue (while j < N and A[j][0] <= S[i]).
- Checks if the priority queue is not empty and pops the most valuable treasure from the queue, updating the total value (if pq: res += -heapq.heappop(pq)).

5. Output :

- Prints the total value of treasures carried by all carts (print(res)).

Visualisasi Code :

N (Jumlah Harta Karun): 2

M (jumlah kereta kuda): 5

- Bobot dan Nilai Harta Karun:

- Harta Karun 1: Bobot = 9, Nilai = 100
- Harta Karun 2: Bobot = 4, Nilai = 100

- Kapasitas Kereta:
  - Kereta 1: Kapasitas = 1
  - Kereta 2: Kapasitas = 2
  - Kereta 3: Kapasitas = 3
  - Kereta 4: Kapasitas = 1
  - Kereta 5: Kapasitas = 3

1. Harta karun diurutkan berdasarkan bobotnya:
   - Harta Karun 2: Bobot = 4, Nilai = 100
   - Harta Karun 1: Bobot = 9, Nilai = 100

2. Kereta diurutkan berdasarkan kapasitas maksimumnya:
   - Kereta 1: Kapasitas = 1
   - Kereta 2: Kapasitas = 2
   - Kereta 3: Kapasitas = 3
   - Kereta 4: Kapasitas = 1
   - Kereta 5: Kapasitas = 3

3. Iterasi dilakukan untuk setiap kereta:
   - Kereta 1 (Kapasitas = 1):
    - bobot harta karun 2 = 4 bandingkan dengan 1 dan harta karun 1 dengan bobot 9 bandingkan dengan 1 -> 4 > 1 dan 9 > 1 artinya harta karun tak bisa dimasukan

   - Kereta 2 (Kapasitas = 2):
     - bobot harta karun 2 = 4 bandingkan dengan 2 dan harta karun 1 dengan bobot 9 bandingkan dengan 2 -> 4 > 2 dan 9 > 2 artinya harta karun tak bisa dimasukan

   - Kereta 3 (Kapasitas = 3):
     - bobot harta karun 2 = 4 bandingkan dengan 3 dan harta karun 1 dengan bobot 9 bandingkan dengan 3 -> 4 > 3 dan 9 > 3 artinya harta karun tak bisa dimasukan

- Kereta 4 (Kapasitas = 1):

- bobot harta karun 2 = 4 bandingkan dengan 1 dan harta karun 1 dengan bobot 9 bandingkan dengan 1 -> 4 > 1 dan 9 > 1 artinya harta karun tak bisa dimasukan

- Kereta 5 (Kapasitas = 3):

- bobot harta karun 2 = 4 bandingkan dengan 3 dan harta karun 1 dengan bobot 9 bandingkan dengan 3 -> 4 > 3 dan 9 > 3 artinya harta karun tak bisa dimasukan

Jadi dengan tak ada bobot yang cocok untuk dimasukkan jadi print hasilnya 0.

Problem M

```
def lachocolatte (n, r, g):
        if n * r == g:
                return 0
        if g % n == 0 or g% r == 0:
                return 1
        for i in range(1, n):
                operation = n * r - - g
                if g % i == 0 and g <= i * r:
                        return 2
                if operation % i == 0 and operation <= i * r:
                        return 2
        return 3
n, r, g = map(int, input().split())
print(lachocolatte (n, r, g))
```

Problem M

1. Input Reading :

- Reads three integers n, r, and g from the input

2. Equality and Divisibility Checks :

- Checks if n * r equals g or if g is divisible by n or r. If any of these conditions are met, it returns specific values (0 or 1) accordingly

3. Iteration and Further Checks :

- Iterates through a range from 1 to n - 1 and performs additional calculations and checks:
- Calculates operation
- Checks if certain conditions involving g and operation are met within the loop. If so, it returns the value 2

4. Default Return :

- If none of the previous conditions are satisfied, it returns the value 3

5. Usage of the Function :

- The code reads three integers from input to assign values to n, r, and g
- It then calls the lachocolatte function with these values and prints the returned value

Visualisasi perhitungan :

Input n = 4 sebagai lebar, r = 4 sebagai luas dan g = 10 luas coklat diinginkan

- jika 4 * 4 == 10 tidak memenuhi karean tidak sama maka abaikan return 0
- jika 10 % 4 == 0 atau 10 % 4 == 0 tidak ada yang memenuhi karena hasilnya sisa bagi / modulusnya 2 dan 2 tidak sama dengan 0 sehingga abaikan return 1
- operation = 4 * 4 –10 yaitu 6 dan lakukan (iterasi ke 1 / i = 1) yaitu jika 10 % 1 dan 10 <= 1 * 4 (tidak terpenuhi) , maka kita lanjut cek ke sub kondisi yaitu 6 % 1 == 0 dan 6 <= 1 * 4 (tidak terpenuhi)
- maka lakukan i ke 2 yaitu jika 10 % 2 dan 10 <= 2 * 4 (tidak terpenuhi) , maka kita lanjut cek ke sub kondisi yaitu 6 % 2 == 0 dan 6 <= 2 * 4 **(terpenuhi**)
- lalu cetak haislnya yaitu retrun 2 sehingga output = 2