¡No más empates!

En el primer concurso hubo un problema: se clasificaban 7 concursantes ¡y hubo un empate a puntuación de 14 participantes en la séptima posición! A los organizadores no nos gusta tener que desempatar por el tiempo acumulado, pero al ser tantos los concursantes empatados no nos quedó otra opción.

Ahora queremos saber si esto ocurrirá en el segundo clasificatorio. En este clasificatorio van a participar n concursantes y se clasificarán los k primeros, con $1 \leq k < n$. Tenemos las n puntuaciones de los participantes y queremos saber si habrá empates que impidan separar los k primeros participantes de los siguientes sin tener que recurrir al tiempo.

Estrategia de la solución

Este problema era el más fácil. Una solución era ordenar los puntos de mayor a menor y mirar si el k-ésimo entero y el (k+1)-ésimo eran iguales.

Solución en Python

```python
t = int(input())
for test in range(t):
        n, k = map(int, input().split())
        p = list(map(int, input().split()))
        p.sort(reverse=True)
        print("BIEN" if p[k-1] != p[k] else "EMPATE")
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;

int main() {
  int t;
  cin >> t;
  while (t--) {
    int n, k;
    cin >> n >> k;
    vector<int> V(n);
    for (int i = 0; i < n; ++i) cin >> V[i];
    sort(V.rbegin(), V.rend());
    if (V[k] == V[k-1]) cout << "EMPATE" << endl;
    else cout << "BIEN" << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Dividiendo

Dados *n* enteros, ¿hay alguno de ellos que divida a todos los demás?

Estrategia de la solución

Si x divide a y, entonces es fácil ver que si x, y >= 1 tiene que cumplirse que x <= y. Por lo tanto el único candidato a dividir todos los enteros es el menor de la lista. Encontramos este entero y comprobamos si divide al resto de los enteros.

Solución en Python

```
t = int(input())
for test in range(t):
	n = int(input())
	a = list(map(int, input().split()))
	mini = a[0]
	for i in range(1, n):
	mini = min(mini, a[i])
	ans = True
	for i in range(n):
	if a[i] % mini != 0:
		ans = False
		break
	print("SI" if ans else "NO")
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> vi;

int main() {
  int t;
  cin >> t;
  while (t--) {
    int n;
    cin >> n;
    vi V(n);
    int mini = 1e9 + 10;
    for (int i = 0; i < n; ++i) {
      cin >> V[i];
      mini = min(mini, V[i]);
    }
    bool res = true;
    for (int i = 0; i < n; ++i) {
```

```
      if (V[i]%mini) {
        res = false;
        break;
      }
    }
   if (res) cout << "SI" << endl;
    else cout << "NO" << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Juegos con Queta

En 2005, con el objetivo de promover y fomentar el uso del catalán en la vida cotidiana de los catalanes, la Generalitat de Catalunya creó una campaña que se llamaba: "Dóna corda al Català". Esta campaña tenía una mascota, Queta, que era una boca con patas que cuando le dabas cuerda avanzaba abriendo y cerrando la boca.

15 años más tarde, la Generalitat ya no tiene espacio para guardar más material de campañas pasadas, así que han decidido vender todas las existencias de Quetas que tenían a una familia inglesa que se dedica a las apuestas ilegales.

Estos han creado un juego con el que ganar dinero con las apuestas. El juego consiste en lo siguiente:

Se dibuja una línea en el suelo, en esta línea se hacen dos marcas. Se colocan *n* Quetas en la línea entre las dos marcas, todas en posiciones diferentes, cada Queta mira hacia una de las dos marcas. Se les da cuerda y empiezan a correr, a la misma velocidad. Cuando dos Quetas chocan de frente se destruyen ambas. El juego acaba en el momento en que no quedan Quetas entre las dos marcas. El objetivo del juego es adivinar cuántas Quetas quedarán al final del juego.

Dadas las posiciones de las marcas y las Quetas al inicio del juego, decid cuántas Quetas quedarán al final.

Estrategia de la solución

Ordenamos las Quetas por posición. Iteramos las Quetas de izquierda a derecha, guardaremos en un entero der el número de Quetas “vivas” que hemos visto que van hacia la derecha. Cuando encontramos una Queta puede haber tres opciones:

- Si la Queta va hacia la derecha de momento está viva, entonces sumamos 1 a der
- Si la Queta va hacia la izquierda y der = 0, la Queta va a seguir tirando hasta el final y por lo tanto sumamos uno a la solución

- Si la Queta va hacia la izquierda y der > 1, la Queta chocará con una de las Quetas vivas que va hacia la derecha y por lo tanto restamos 1 a der

Solución en Python

```python
t = int(input())
for test in range(t):
	n, m1, m2 = map(int, input().split())
	q = list(map(int, input().split()))
	d = list(map(int, input().split()))
	v = [None for i in range(n)]
	for i in range(n):
	v[i] = (q[i], d[i])
	v.sort()
	res = n
	cont = 0
	for i in range(n):
	if v[i][1] == 1 and cont:
		cont -= 1
		res -= 2
	elif v[i][1] == 2:
		cont += 1
	print(res)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

typedef pair<int, int> pi;
typedef vector<pi> vpi;

int main() {
  int t;
  cin >> t;
  while (t--) {
    int n, m1, m2;
    cin >> n >> m1 >> m2;
    vpi V(n);
    for (int i = 0; i < n; ++i) cin >> V[i].first;
    for (int i = 0; i < n; ++i) cin >> V[i].second;
    sort(V.begin(), V.end());
    int res = n;
    int cont = 0;
    for (int i = 0; i < n; ++i) {
      if (V[i].second == 1 and cont) {
        --cont;
        res -= 2;
```

```
      }
      else if (V[i].second == 2) ++cont;
    }
    cout << res << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Monedas 3

Pedro tiene *n* monedas, cada una con un valor m_i. Pedro quiere pagar exactamente *k* euros al dependiente de una tienda, para ello sigue el siguiente algoritmo:

- Ordena las monedas de mayor a menor valor.
- Va sacando las monedas en el orden marcado.
- Cada vez que saca una moneda hace lo siguiente: si el valor es menor o igual a lo que le falta por pagar le da la moneda al dependiente, si es mayor se la guarda en el bolsillo.

Este algoritmo no siempre funciona, y a veces Pedro no consigue pagar *k* euros. Él cree que es porque le faltan monedas, por eso te pide dinero.

Sabiendo qué monedas tiene Pedro, la cantidad que quiere pagar y el algoritmo que usa para pagar, ¿Cuál es la mínima cantidad de euros que le tienes que dar para que pueda pagar? Puedes dárselo en más de una moneda.

Estrategia de la solución

Si simulamos el algoritmo descrito, podemos ver que el mínimo número de dinero que tenemos que darle es exactamente lo que le falta por pagar con sus monedas originales.

Solución en Python

```python
t = int(input())
for test in range(t):
	n, k = map(int, input().split())
	m = list(map(int, input().split()))
	m.sort(reverse=True)
	for i in range(n):
	if k >= m[i]:
		k -= m[i]
	print(k)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

typedef long long ll;
typedef vector<ll> vi;

int main() {
```

```
  int t;
  cin >> t;
  while (t--) {
    int n;
    ll k;
    cin >> n >> k;
    vi V(n);
    for (int i = 0; i < n; ++i) cin >> V[i];
    sort(V.rbegin(), V.rend());
    for (int i = 0; i < n; ++i) {
      if (k >= V[i]) k -= V[i];
    }
    cout << k << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Celdas separadas

La prisión de Nlogonia ha sido remodelada reciente. Tiene *n* celdas, cada una con capacidad para un único preso. Algunas celdas están conectadas entre si de tal manera que es posible ir de cualquier celda a cualquier otra de una única manera (sin repetir celdas). Dicho de otra manera, el grafo de las celdas y sus conexiones es un árbol. Tres presos muy peligrosos han llegado a la prisión y tienen que ser asignados a distintas celdas con una única restricción: no puede haber 2 de ellos en celdas conectadas, ya que podrían urdir un plan para escapar. Contad de cuántas maneras es posible seleccionar sus 3 celdas cumpliendo la restricción.

Nota: Las permutaciones de celdas cuentan una única vez (1, 2, 3 y 2, 3, 1 son lo mismo). Dos maneras se consideran distintas si alguna de las celdas es distinta.

Estrategia de la solución

El número de grupos distintos de tres nodos son n*(n-1)*(n-2)/6. De estos grupos tenemos que restar los que tienen dos nodos conectados con una arista. Entonces restamos los grupos de una arista y otro nodo distinto que són (n-1)*(n-2). Podemos ver que si los tres nodos formaban un camino los hemos restado dos veces, entonces tenemos que sumar estas combinaciones, por cada nodo x, si g_x es el grado de este vértice (número de aristas adyacentes al vértice), sumamos g_x*(g_x-1)/2.

Solución en Python

```
t = int(input())
for test in range(t):
        n = int(input())
        deg = [0 for i in range(n)]
        dsum = 0
        for i in range(n-1):
        u, v = map(int, input().split())
        deg[u] += 1
        deg[v] += 1
        dsum += 2
        d2sum = 0
        for i in range(n):
        d2sum += deg[i]*(deg[i]-1)//2
        print( n*(n-1)*(n-2)//6 - dsum*(n-2)//2 + dsum )
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```
#include <vector>
#include <iostream>

using namespace std;
```

```cpp
typedef long long ll;
typedef vector<ll> vi;

int main() {
    int t;
    cin >> t;
    while (t--) {
        ll n;
        cin >> n;
        vi V(n,0);
        for (int i = 1; i < n; ++i) {
            int x, y;
            cin >> x >> y;
            ++V[x];
            ++V[y];
        }
        ll res = n*(n-1)*(n-2)/6 - (n-1)*(n-2);
        for (int i = 0; i < n; ++i) res += V[i]*(V[i]-1)/2;
        cout << res << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Pintando cuadrados

Aniol tiene una cuadrícula de cuadrados blancos de dimensiones *n x m*. Puede pintar algunos de estos cuadrados de color negro, pero solo puede hacer dos tipos de operaciones para pintar los cuadrados: pintar toda una fila de negro o pintar toda una columna de negro.

Sus tres primos pequeños quieren jugar a un juego. Cada uno de ellos, incluido Aniol, piensa un entero a_i, entre *2* y 2^{15}. Entonces calculan $k = a^1 \cdot a^2 \cdot a^3 \cdot a^4$, Aniol gana si puede conseguir que queden exactamente *k* cuadrados blancos haciendo solamente las operaciones anteriores.

Estrategia de la solución

Podemos ver que la respuesta es si, si y solo si k = a*b con a <= n y b <= m. Sabemos que todos los factores primos p de k cumplen que p <= 2^15. Podemos hacer una criba de Eratóstenes para pre-calcular estos primos para poder factorizar k más rápido. Una vez tenemos los factores primos podemos generar todos los divisores a de k y comprobar si alguno cumple que a <= n y k/a <= m. Esta solución tiene coste O(D), D es el número de divisores de k. La solución oficial tiene un coste O(sqrt(D)) pero si se programaba de forma eficiente se podía conseguir 100 puntos con la solución O(D).
La solución oficial utiliza una estrategia llamada meet in the middle, podéis leer más sobre esta técnica aquí: https://www.geeksforgeeks.org/meet-in-the-middle/

Solución en C++

```cpp
#include <vector>
#include <set>
#include <algorithm>
#include <iostream>

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef vector<ll> vi;
typedef pair<ll, ll> pi;
typedef vector<pi> vpi;
typedef set<pi> si;

void f(pi x, vpi& V, int i, ll n, ll m, vpi& R) {
  if (i == (int)V.size()) {
    R.push_back(x);
    return;
  }
  ll valtot = 1;
  for (int j = 0; j < V[i].second; ++j) valtot *= V[i].first;
  ll val1 = 1;
```

```
  for (int j = 0; j <= V[i].second; ++j) {
    ll val2 = valtot/val1;
    pi y = pi(x.first*val1, x.second*val2);
    if (y.first <= n and y.second <= m) f(y, V, i+1, n, m, R);
    val1 *= V[i].first;
  }
}

int main() {
  int N = (1<<15);
  vi V(N, 0);
  vi P;
  for (ll i = 2; i < N; ++i) {
    if (!V[i]) {
      P.push_back(i);
      for (ll j = i*i; j < N; j += i) V[j] = 1;
    }
  }
  int t;
  cin >> t;
  while (t--) {
    ll n,m,k;
    cin >> n >> m >> k;
    if (n > m) swap(n, m);
    ll val = k;
    vpi F;
    ll tot = 1;
    for (int i = 0; i < (int)P.size(); ++i) {
      if (val%P[i] == 0) {
        ll ac = 1;
        F.push_back(pi(P[i], 0));
        while (val%P[i] == 0) {
          val /= P[i];
          F.back().second++;
          ++ac;
        }
        tot *= ac;
      }
      if (val == 1) break;
    }
    vpi V1, V2;
    int ind = 0;
    ll valf = 1;
    ll sum = 1;
    while (ind < (int)F.size() and (valf*sum)*(valf*sum) < tot) {
      if (V1.size() == 0 or V1.back().first != F[ind].first) {
        valf *= sum;
        sum = 1;
        V1.push_back(pi(F[ind].first, 0));
      }
      ++sum;
      ++V1.back().second;
      --F[ind].second;
      if (F[ind].second == 0) ++ind;
```

```
    }
    while (ind < (int)F.size()) {
      if (V2.size() == 0 or V2.back().first != F[ind].first) V2.push_back(pi(F[ind].first,
0));
      ++V2.back().second;
      --F[ind].second;
      if (F[ind].second == 0) ++ind;
    }
    vpi R1, R2;
    f(pi(1,1), V1, 0, n, m, R1);
    f(pi(1,1), V2, 0, n, m, R2);
    sort(R1.begin(), R1.end());
    sort(R2.rbegin(), R2.rend());
    bool pots = false;
    for (int i = 0, j = 0; i < (int)R1.size(); ++i) {
      while (j < (int)R2.size() and R1[i].first*R2[j].first > n) ++j;
      if (j == (int)R2.size()) break;
      if (R1[i].second*R2[j].second <= m) {
        pots = true;
        break;
      }
    }
    if (pots) cout << "SI" << endl;
    else cout << "NO" << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)