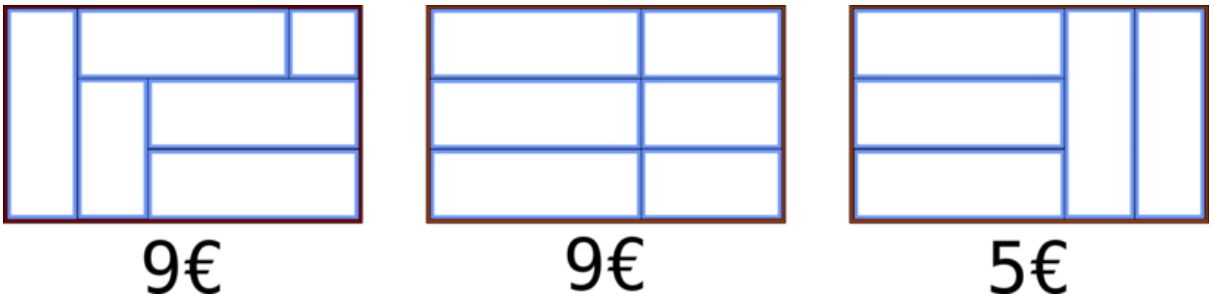
Piezas

Rogelio tiene un cuadro de dimensiones *n* x *m*, con n ≤ 3. El cuadro es muy feo y quiere taparlo. Para taparlo tiene tres tipos de piezas:

- Las piezas con dimensiones 1 x 1, cada una de ellas cuesta 3 euros.
- Las piezas con dimensiones 1 x 2, cada una de ellas cuesta 2 euros.
- Las piezas con dimensiones 1 x 3, cada una de ellas cuesta 1 euro.

Las piezas se pueden girar y rotar en cualquier dirección. Rogelio quiere tapar el cuadro con las piezas pero con las condiciones siguientes: que el cuadro quede completamente tapado, que no haya más de una pieza tapando un mismo punto del cuadro, que todas las piezas estén completamente dentro del cuadro y que cueste el mínimo dinero posible.

Por ejemplo, aquí hay tres formas diferentes de tapar un cuadro 3 x 5. La de la derecha tiene el mínimo coste posible.



Estrategia de la solución

En este problema teníamos que separar en casos, los diferentes escenarios eran los valores de n (1, 2 o 3) y el resto al dividir entre 3 de m (0, 1 o 2). En total hay 9 casos y algunos de ellos son equivalentes. Hacía falta analizar cada caso por separado.

Solución en Python

```
t = int(input())
for test in range(t):
    n, m = map(int, input().split())
    if n == 3:
        print(m)
    elif n == 2:
        if m%3 == 0:
            print(2*m//3)
        elif m%3 == 1:
            print(2*(m//3) + 2)
        else:
            print(2*(m//3) + 4)
    else:
        if m%3 == 0:
            print(m//3)
```

```
        elif m%3 == 1:
            print(m//3 + 3)
        else:
            print(m//3 + 2)
```

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

Solución en C++

```
#include <iostream>

using namespace std;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n, m;
        cin >> n >> m;
        int res = 0;
        if (n == 1) {
                res += m/3;
                if (m%3 == 1) res += 3;
                else if (m%3 == 2) res += 2;
        }
        else if (n == 2) {
                res += 2*(m/3);
                res += (m%3)*2;
        }
        else {
                res = m;
        }
        cout << res << endl;
  }
}
```

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

Concurso de programación

Arturo y Benito compiten en un concurso de programación que consta de *N* problemas. En cada problema consiguen una serie de puntos. Para cada problema, el ganador del problema es quien consigue más puntos. Además, nunca hay empates porque se da la peculiaridad de que Arturo siempre consigue un número par de puntos y Benito un número impar. Sabemos los *N* puntos que ha conseguido en los problemas (a_i) Arturo y los *N* puntos que ha conseguido Benito (b_i), pero no sabemos a qué problema corresponde cada puntuación, así que no es posible determinar en cuántos problemas han ganado cada uno. Vuestro objetivo es determinar cuál es el máximo número de problemas que puede haber ganado Arturo.

Estrategia de la solución

Una observación fácil es que si existe una solución en la que Benito pierde *k* partidas, existe otra solución en la que Benito pierde en las *k* partidas donde consigue menos puntos, y puede que en algunas más. Con esto podemos ver que tenemos que valorar sólo este tipo de soluciones.
Ordenaremos las puntuaciones de ambos jugadores de menor a mayor, para cada puntuación de Benito buscaremos la primera puntuación de Arturo que no hayamos usado y que sea mayor que la puntuación de Benito. Si no existe, es que ya hemos encontrado el máximo número de partidas que puede ganar Arturo.

Solución en Python

```python
t = int(input())
for test in range(t):
    n = int(input())
    a = list(map(int, input().split()))
    b = list(map(int, input().split()))
    a = sorted(a)
    b = sorted(b)
    i, j, cnt = 0, 0, 0
    while i < n and j < n:
        if a[i] > b[j]:
            cnt += 1
            j += 1
        i += 1
    print(cnt)
```

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

Solución en C++

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

using namespace std;
```

```cpp
typedef vector<int> vi;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;

        vi A(n), B(n);
        for (int i = 0; i < n; ++i) cin >> A[i];
        for (int i = 0; i < n; ++i) cin >> B[i];
        sort(A.begin(), A.end());
        sort(B.begin(), B.end());

        int ind = 0;
        for (int i = 0; i < n; ++i)
        if (A[i] > B[ind]) ++ind;

        cout << ind << endl;
  }
}
```

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

Pintando Piedras

Pedro tiene *n* piedras formando una fila y quiere pintarlas con *c* colores distintos de manera que dos piedras consecutivas no sean del mismo color.

Ha pintado algunas de las piedras y ahora se pregunta de cuántas formas puede pintar las piedras que le faltan para conseguir su objetivo.

Estrategia de la solución

En este problema se necesitaba programación dinàmica para conseguir más de 11 puntos, si no sabéis lo que es, leed los manuales primero.

Primero, si en algún momento había dos piedras consecutivas pintadas del mismo color, la respuesta era 0.
Ahora, nos miramos los segmentos de 0 consecutivos.
Si un segmento tiene k ceros consecutivos y están al inicio o al final, el número de soluciones para este segmento, independientemente del color de la piedra consecutiva al segmento, es *(c-1)^k*. Si el segmento de 0 consecutivos es todo (las *n* piedras) la solución es *c*(c-1)^(n-1)*. Estas fórmulas las podéis demostrar fácilmente si pintáis las piedras progresivamente.
Ahora tenemos dos opciones, las piedras adyacentes al segmento son del mismo color o son de diferente color. Llamaremos *D(k)* el número de soluciones de un segmento de *k* ceros con las piedras adyacentes de distinto color, y *M(k)* si son del mismo color.
Entonces tenemos:
D(1) = c-2
M(1) = c-1
*D(k) = (c-2)*D(k-1) + M(k-1) si k > 1*
*M(k) = (c-1)*D(k-1) si k > 1*
Podemos precalcular estos valores y obtener la solución.

Solución en Python

```
import sys
sys.setrecursionlimit(30000)

M = int(1e9+7)
dp, a, n, c = [None] * 4

def mpow(a, n):
    if n == 0:
        return 1
    temp = mpow(a*a % M, n//2)
    if n % 2 == 0:
        return temp
    return temp * a % M

def f(k):
```

```
    if k == 1:
        return c-2
    if dp[k] == -1:
        dp[k] = g(k-1) + (c-2)*f(k-1)
    return dp[k]

def g(k):
    if k == 1:
        return c-1
    return (c-1)*f(k-1)

def solve(l, r):
    ans = 1
    s = r-l+1
    if s == 1:
        if l == 0 or l == n-1:
            ans = c-1
        else:
            ans = c-1 if a[l-1] == a[l+1] else c-2
    else:
        if l == 0 and r == n-1:
            ans = c * mpow(c-1, s-1)
        elif l == 0 or r == n-1:
            ans = mpow(c-1, s)
        else:
            ans = g(s) if a[l-1] == a[r+1] else f(s)
    return ans % M

t = int(input())
for test in range(t):
    dp = [-1] * 10001
    n, c = map(int, input().split())
    a = list(map(int, input().split()))
    invalid = False
    for i in range(1, n):
        if a[i] != 0 and a[i-1] == a[i]:
            invalid = True
    if invalid:
        print(0)
    elif n == 1:
        if a[0] == 0:
            print(c)
        else:
            print(1)
    elif c == 1:
        if n == 1:
            print(1)
        else:
            print(0)
    else:
        ans = 1
        first = -1
        for i in range(n):
            if a[i] == 0:
```

```
            if first == -1:
                first = i
        elif first != -1:
            ans = ans * solve(first, i-1) % M
            first = -1 # Reset
    if first != -1:
        ans = ans * solve(first, n-1) % M
    print(ans)
```

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

Solución en C++

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

using namespace std;

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

const ll mod = 1e9 + 7;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        ll c;
        cin >> n >> c;
        vi V(n);
        for (int i = 0; i < n; ++i) cin >> V[i];
        ll res = 1;
        int ind1 = 0;
        int ind2 = n-1;
        while (ind1 < ind2 and min(V[ind1], V[ind2]) == 0) {
                if (V[ind1] == 0) ++ind1;
                else --ind2;
                res = (res*(c-1))%mod;
        }
        if (ind1 == ind2) {
                if (V[ind1] == 0) res = (res*c)%mod;
                cout << res << endl;
                continue;
        }
        vi D(n);
        vi M(n);
        M[0] = 0;
        D[0] = 1;
        M[1] = max(0ll, c-1);
        D[1] = max(0ll, c-2);
        for (int i = 2; i < n; ++i) {
                M[i] = (max(0ll, c-1)*D[i-1])%mod;
```

```
            D[i] = (max(0ll, c-2)*D[i-1] + M[i-1])%mod;
        }
        int ind3 = ind1 + 1;
        while (ind3 <= ind2) {
            while (V[ind3] == 0) ++ind3;
            if (V[ind1] == V[ind3]) res = (res*M[ind3-ind1-1])%mod;
            else res = (res*D[ind3-ind1-1])%mod;
            ind1 = ind3;
            ++ind3;
        }
        cout << res << endl;
    }
}
```

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

La ruta de los 12 lagos

La ruta de los 12 lagos es una pequeña ruta de montaña en Francia, muy cercana a la frontera con Andorra. Consiste en una ruta circular para hacer a pie o en bicicleta donde a lo largo del recorrido se pueden ver 12 lagos.

Laura quería hacer esta ruta pero se ha equivocado de carretera y ha acabado en la ruta de los *n* lagos. Esta ruta es muy parecida a la de los 12 lagos pero mucho más larga. Hay *n* lagos y entre el *i*-ésimo lago y el *i+1*-ésimo hay un camino de c_i kilómetros. Entre el *n*-ésimo lago y el primero hay un camino de c_n kilómetros.

Como esta ruta es demasiado larga existen diferentes líneas de autobús, cada línea va de un lago l_1 a un lago l_2 por el camino más corto (según la ruta circular). Hay una línea de bus para cada par de lagos y cada línea va en los dos sentidos.

Al haberse confundido, a Laura se le han quitado las ganas de andar. Conserva intactas, eso sí, sus ganas de ver la zona y hacer fotografías por lo que ha decidido aprovechar las líneas de autobuses para disfrutar del paisaje. Quiere planificar una ruta que tenga las siguientes condiciones:

- Puede salir de cualquier lago, pero tiene que acabar en el mismo lago.
- Como máximo puede coger cada línea de bus una vez en cada uno de los dos sentidos.
- Quiere recorrer el máximo número de kilómetros.

Con estas condiciones, ¿cuál es el máximo número de kilómetros que puede recorrer Laura sin caminar?

Estrategia de la solución

La observación principal del problema era ver que lo que realmente nos pedían era la suma de las distancias mínimas entre cada par de lagos multiplicada por 2.

Una vez tenemos esto conseguir 54 puntos era relativamente sencillo.

Para conseguir los 100 puntos hacía falta una solución más eficiente:
Si nos miramos los índices de los lagos módulo *n*, tenemos que para cada lago *i*, hay otro lago *l_i*, donde el camino *i* → *i+1* → *i+2* → ... → *l_i -1* → l_i es el más corto para ir de *i* a *l_i*, pero que par *l_i +1* el camino más corto es i → *i-1* → *i-2* → ... → *l_i* + *2* → *l_i+1*. Este *l_i* es único (o puede haber 2 si el camino entre i y *l_i* es la mitad del total) y se cumple que si *l_(i+1)* >= *l_i*. Entonces si calculamos el primer l podemos ir calculando los siguientes aumentando este hasta que nos pasemos. Esta solución en total es O(n) si podemos calcular las distancias entre dos lagos en tiempo constante.

Solución en Python

```
t = int(input())
```

```
for test in range(t):
	n = int(input())
	a = list(map(int, input().split()))
	v = [0 for i in range(2*n+1)]
	for i in range(1, 2*n+1):
		v[i] = v[i-1] + a[(i-1)%n]
	suma = v[n]
	res = 0
	acum = 0
	ind2 = 0
	for ind1 in range(n):
		if ind2 <= ind1:
			ind2 = ind1
			acum = 0
		else:
			km = ind2 - ind1 + 1
			kv = v[ind1] - v[ind1-1]
			acum -= kv*km
		while 2*(v[ind2+1]-v[ind1]) <= suma:
			ind2 += 1
			acum += v[ind2] - v[ind1]
		res += 2*acum
		if 2*(v[ind2]-v[ind1]) == suma:
			res -= v[ind2] - v[ind1];
	print(res)
```

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

Solución en C++

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

using namespace std;

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

int main() {
  int t;
  cin >> t;
  while (t--) {
	int n;
	cin >> n;
	vi A(n);
	for (int i = 0; i < n; ++i) cin >> A[i];
	vi V(2*n+1,0);
	for (int i = 1; i <= 2*n; ++i) {
		V[i] = V[i-1] + A[(i-1)%n];
	}
	ll suma = V[n];
	ll res = 0;
	ll acum = 0;
	int ind2 = 0;
```

```
        for (int ind1 = 0; ind1 < n; ++ind1) {
                if (ind2 <= ind1) {
                        ind2 = ind1;
                        acum = 0;
                }
                else {
                        ll km = ind2 - ind1 + 1;
                        ll kv = V[ind1] - V[ind1-1];
                        acum -= kv*km;
                }
                while (2*(V[ind2+1] - V[ind1]) <= suma) {
                        ++ind2;
                        acum += V[ind2] - V[ind1];
                }
                res += 2*acum;
                if (2*(V[ind2] - V[ind1]) == suma) res -= V[ind2] - V[ind1];
        }
        cout << res << endl;
  }
}
```

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

Dirigiendo las carreteras de Grafolandia

En Grafolandia hay *n* ciudades conectadas por *m* carreteras bidireccionales, de forma que se puede llegar desde cualquier ciudad a cualquier otra pasando por una o más carreteras. Para reducir el tránsito, el gobierno ha decidido hacer que *k* carreteras pasen a ser de una sola dirección, pero quieren que desde cada una de las ciudades se pueda seguir yendo a todas las otras.

Ayuda a los gobernantes grafolandeses a saber si esto es posible, y en el caso de que sea posible di qué carreteras seleccionar y en qué dirección tiene que ir cada carretera.

Estrategia de la solución

En uno de los entrenamientos se habló de las aristas puente, en este problema se puede ver que las únicas aristas que no se pueden dirigir son las aristas puente. Entonces si tenemos demasiadas aristas puente la respuesta es NO.
En caso de que la respuesta sea SI, recorremos el grafo con un dfs, cada vez que visitamos una arista por primera vez, y no sea puente, la dirigimos del nodo actual a su vecino. Se puede demostrar que este algoritmo funciona.

Solución en Python

```
import sys
sys.setrecursionlimit(200000)

num = None
minnum = None
cont = None
acum = None
G = None
A = None
R = None

def vecino(x, b):
    global num, minnum, cont, acum, G, A, R
    if A[b][0] == x:
        return A[b][1]
    return A[b][0]

def actu(x, b):
    global num, minnum, cont, acum, G, A, R
    acum += 1
    if A[b][0] == x:
        R[b] = 1
    else:
        R[b] = -1

def dfs(x, p = -1):
    global num, minnum, cont, acum, G, A, R
    num[x] = cont
```

```
    minnum[x] = cont
    cont += 1
    for ID in G[x]:
        y = vecino(x, ID)
        if y == p:
            continue
        if num[y] == -1:
            dfs(y, x)
            minnum[x] = min(minnum[x], minnum[y])
            if minnum[y] <= num[x]:
                actu(x, ID)
        else:
            minnum[x] = min(minnum[x], num[y])
            if num[y] < num[x]:
                actu(x, ID)

t = int(input())
for test in range(t):
    n, m, k = map(int, input().split())
    num = [-1 for i in range(n)]
    minnum = [-1 for i in range(n)]
    cont = 0
    acum = 0
    G = []
    for i in range(n):
        G.append([])
    A = [None for i in range(m)]
    R = [0 for i in range(m)]
    for i in range(m):
        u, v = map(int, input().split())
        A[i] = (u, v)
        G[u].append(i)
        G[v].append(i)
    dfs(0)
    if acum < k:
        print("NO")
        continue
    print("SI")
    for i in range(m):
        if k == 0:
            break
        if R[i] == 1:
            print(A[i][0], A[i][1])
        if R[i] == -1:
            print(A[i][1], A[i][0])
        if R[i] != 0:
            k -= 1
```

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

Solución en C++

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

```cpp
using namespace std;

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

vi num, minnum;
int cont;
int acum;
vvi G;
vpi A;
vi R;

inline int vecino(int x, int b) {
  if (A[b].first == x) return A[b].second;
  return A[b].first;
}

inline void actu(int x, int b) {
  ++acum;
  if (A[b].first == x) R[b] = 1;
  else R[b] = -1;
}

void dfs(int x, int p = -1) {
  num[x] = minnum[x] = cont++;
  for (auto id : G[x]) {
        int y = vecino(x, id);
        if (y == p) continue;
        if (num[y] == -1) {
                dfs(y, x);
                minnum[x] = min(minnum[x], minnum[y]);
                if (minnum[y] <= num[x]) actu(x, id);
        }
        else {
                minnum[x] = min(minnum[x], num[y]);
                if (num[y] < num[x]) actu(x, id);
        }
  }
}

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n, m, k;
        cin >> n >> m >> k;
        num = minnum = vi(n, -1);
        cont = acum = 0;
        G = vvi(n);
        A = vpi(m);
        R = vi(m, 0);
```

```
        for (int i = 0; i < m; ++i) {
                cin >> A[i].first >> A[i].second;
                G[A[i].first].push_back(i);
                G[A[i].second].push_back(i);
        }
        dfs(0);
        if (acum < k) {
                cout << "NO" << endl;
                continue;
        }
        cout << "SI" << endl;
        for (int i = 0; i < m; ++i) {
                if (!k) break;
                if (R[i] == 1) cout << A[i].first << ' ' << A[i].second << endl;
                if (R[i] == -1) cout << A[i].second << ' ' << A[i].first << endl;
                if (R[i]) --k;
        }
  }
}
```

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