¡Orden!

La comisión de orden de la OIE está intentando ordenar los mensajes de emoción de los futuros participantes de la OIE 2020. Recordemos que los mensajes están definidos por cuatro enteros no negativos n_o, n_i, n_e, $n_!$, donde son el número de "*O*", el de "*I*", el de "*E*" y el de "*!*" respectivamente. Por ejemplo 3 2 1 2 sería "*OOOIIE!!*" y 0 1 2 0 sería "*IEE*".

Para poder ordenar los diferentes mensajes han definido el siguiente orden, sean m_1 y m_2 dos mensajes, si el número de caracteres de m_1 es menor que el de m_2, m_1 va antes que m_2. Si es mayor va primero . En caso de igualdad se ordenan por orden alfabético ($E < I < O < !$). Es decir, se ordenan por número creciente de caracteres y en caso de igualdad por orden alfabético.

Por ejemplo:

"OOO" < "OIE!"

"OIE!" < "OOOO"

"!" < "EE"

"OO!" < "O!!"

Ahora la comisión se pregunta, dado *n*, cual es el *n*-ésimo mensaje en el orden definido si consideramos todos los mensajes posibles? Recordad que el mensaje tiene que tener como mínimo una letra.

La lista de todos los mensajes empezaría con:

"E","I","O","!","EE",E!","IE","II","I!","OE","OI","OO","O!","!!","EEE","EE!","E!!","IEE","IE!","IIE"...

Estrategia de la solución

Este era el problema más difícil con diferencia, aunque conseguir alguna puntuación parcial era muy fácil (los primeros casos). Vamos a descubrir cómo conseguir los 100 puntos con la solución del autor, aunque también existen otras soluciones no tan eficientes que podían conseguir los 100 puntos.
Antes de empezar a leer la solución es importante estar familiarizado con algunos conceptos básicos de combinatoria:

- PDF corto introductorio: http://www.dmae.upct.es/~mcruiz/Telem06/Teoria/conteo2.pdf
- Página de Wikipedia: https://es.wikipedia.org/wiki/Coeficiente_binomial

Analizando los casos públicos, o haciendo algunos cálculos es fácil ver que ninguna palabra de las que nos piden va a tener más de 70000 letras (es una cota superior muy generosa). El algoritmo va a consistir en varios pasos, en cada paso vamos a determinar una de las

características de la palabra. Primero determinaremos el número de letras, después el número de O's, seguidamente el de I's y finalmente el de E's y por lo tanto el de !'s. En la mayoría de los pasos vamos a utilizar una búsqueda binaria (aquí el enlace al artículo del manual).

Paso 1, número de letras:
Queremos contar cuántas palabras hay con *n* o menos letras, Si queremos contar cuántas palabras hay exactamente con *n* letras podemos ver que tenemos que repartir las letras en O's, I's, E's y !'s. Esto lo podemos calcular con $\binom{n+3}{3}$. Ahora si queremos calcular el número de palabras con *n* letras o menos podemos añadir un quinto carácter, el espacio, y entonces tenemos que repartir las *n* letras en O's, I's, E's, !'s y espacios, el resultado es $\binom{n+4}{4}$, tenemos que tener en cuenta que la palabra vacía, todo espacios, no puede existir. Entonces es $\binom{n+4}{4}$-1. Esta función es creciente, por lo que podemos hacer una búsqueda binaria para calcular el número de letras de nuestra palabra.

Paso 2, número de O's:
Partimos de que nuestra palabra tiene *n* letras, podemos ver que en el orden alfabético primero van a ir todas las palabras que tienen alguna I o E y luego las que solo están formadas por O's y !'s (solamente hay *n*+1 palabras de este tipo). En caso de que se trate del segundo caso, es decir, que la palabra esté entre las *n*+1 últimas de las que tienen *n* letras, podemos encontrar directamente de que palabra se trata y ya hemos terminado. En caso contrario, tenemos que nuestra palabra tiene alguna I o E, por lo tanto primero van a ir todas las palabras que no tienen ninguna O, después las que tienen una, dos, tres, etc. Igual que en el paso 1, queremos calcular cuántas palabras hay con *k* o menos O's, que contengan una E o I. Para esto vamos a contar cuántas palabras tienen más de *k* O's, y lo vamos a restar del total, que ya vimos en el paso 1 que era $\binom{n+3}{3}$. Además también vamos a tener que restar las palabras con *k* o menos O's que no contienen ni I's ni E's, que son exactamente (*k*+1) palabras. Para contar el número de palabras con más de k O's hace falta una observación, las palabras de *n* letras que empiezan con más de *k* O's son exactamente todas las palabras con *n*-(*k*+1) letras añadiendoles *k*+1 O's al principio. Visto esto y utilizando la observación del apartado 1 tenemos que este número es $\binom{n-(k+1)+3}{3}$. Y por lo tanto, el número de palabras de *n* letras que contienen alguna I o E con *k* o menos O's es: $\binom{n+3}{3}$-$\binom{n-(k+1)+3}{3}$-(*k*+1). Esto es creciente y por lo tanto, igual que antes podemos hacer una búsqueda binaria para encontrar el número exacto de O's.

Paso 3, número de I's:
Vamos a considerar que ahora las palabras son solo I E !, y que son de longitud *m* = *n* - *k*. Entonces vemos que este caso es muy parecido al anterior. Las últimas (*m*+1) palabras son las que no contienen ninguna E, el número total de palabras es $\binom{m+2}{2}$, en este caso solo tenemos 3 letras. Y con argumentos parecidos al apartado anterior vemos que el número de palabras con *q* o menos I's es: $\binom{m+2}{2}$-$\binom{m-(q+1)+2}{2}$-(*q*+1). Igual que antes hacemos la

búsqueda binaria en el caso general o encontramos la palabra directamente en caso que esté entre las (*m*+1) últimas de las que tienen *n* letras y *k* O's.

Paso 4, número de E's:
Sea *l* = *m* - *q* = *n* - *k* - *q*. Ahora tenemos *l* opciones, recordemos que hay como mínimo una E, ya que en el caso anterior hemos descartado los que no tienen E's, entonces podemos encontrar directamente el número de E's que necesitamos. A partir de aquí tenemos también el número de !'s y por lo tanto ya hemos terminado.

Solución en C++

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

using namespace std;

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


// Funcion que calcula el maximo comun divisor
ll gcd(ll a, ll b) {
  if (b == 0) return a;
  return gcd(b, a%b);
}

/**
 * En esta funcion calculamos n sobre k
 * Par evitar overflow primero hacemos todas
 * las divisiones y luego multiplicamos.
 *
 * Con las condiciones del problema puede
 * que no haga falta pero siempre es una
 * buena practica cuando trabajamos con numeros
 * cercanos al limite de la capacidad
 *
 * La formula es n*(n-1)*...*(n-k+1)/k*(k-1)*...*1
 */
ll nSk(ll n, ll k) {
  // Guardamos en T, los k factores del numerador
  ll T[k];
  for (int i = 0; i < k; ++i) T[i] = n-i;

  // Recorremos todos los factores del denominador
  // y los vamos dividiendo de los factores que
  // podamos
  for (ll i = 1; i <= k; ++i) {
        // Para cada valor de i recorremos todos
        // los valores de T, hasta que ya hemos utilizado
        // todos los factores primos de i
        int j = 0;
        ll val = i;
        // Cuando val == 1 ya no quedan factores primos
```

```
        // por usar
        while (val > 1) {
        // Tenemos que encontrar el siguiente T[j] que
        // comparta factores primos con val
        while (gcd(val, T[j]) == 1) ++j;

        // Una vez encontrado dividimos val y T[j]
        // entre su maximo comun divisor
        ll g = gcd(val, T[j]);
        T[j] /= g;
        val /= g;
        }
  }

  // Una vez hechas todas las division solo
  // falta multiplicar los factores del numerador
  ll res = 1;
  for (int i = 0; i < k; ++i) res *= T[i];
  return res;
}

// Es una cota superior de la longitud de las palabras
const ll M = 70000;

int main() {
  int t;
  cin >> t;

  while (t--) {
        ll n;
        cin >> n;

        // Paso 1: Encontrar la longitud de la palabra
        ll a0 = 0;
        ll b0 = M;
        while (a0 + 1 < b0) {
        ll c0 = (a0 + b0)/2;
        if (n >= nSk(c0 + 4, 4)) a0 = c0;
        else b0 = c0;
        }
        ll numll = b0;

        // La longitud es b0 = a0 + 1, ahora actualizamos n
        // El valor de n ahora es la posicion de la palabra
        // dentro de las  que tienen exactamente numll caracteres
        n -= (nSk(a0 + 4, 4) - 1);

        // Total indica el numero total de palabras
        // con numll letras
        ll total = nSk(b0 + 3, 3);

        // Paso 2: Encontrar el numero de O's

        // En caso que n este entre las (numll + 1) ultimas
```

```
// posiciones es un caso especial donde no hay
// ni E's ni I's
if (n + numll >= total) {
int kind = n + numll - total;
cout << numll - kind << " 0 0 " << kind << endl;
continue;
}

// Caso general, tenemos que encontrar el numero de O's
ll a1 = -1;
ll b1 = numll;
while (a1 + 1 < b1) {
ll c1 = (a1 + b1)/2;
ll k = total - (nSk(numll - (c1+1) + 3, 3) + (c1+1));
if (k < n) a1 = c1;
else b1 = c1;
}
ll numO = b1;

// Actualizamos el valor de n, ahora es la posicion
// dentro de las que tienen numll letras y numO O's
n -= (total - (nSk(numll - b1 + 3, 3) + b1));

// numll2 es el numero de letras sin determinar
// total2 el numero de palabras con numll letras
// y numO O's
ll numll2 = numll - numO;
ll total2 = nSk(numll2 + 2, 2);


// Paso 3: Encontrar el numero de I's

// En caso que n este entre las (numll2+1) ultimas posiciones
// es un caso especial donde no hay E's
if (n + numll2 >= total2) {
int kind = n + numll2 - total2;
cout << numO << ' ' << numll2 - kind << " 0 " << kind << endl;
continue;
}

//Caso general, tenemos que encontrar el numero de I's
ll a2 = -1;
ll b2 = numll2;
while (a2 + 1 < b2) {
ll c2 = (a2 + b2)/2;
ll k = total2 - (nSk(numll2 - (c2+1) + 2, 2) + (c2+1));
if (k < n) a2 = c2;
else b2 = c2;
}
ll numI = b2;

// n es la posicion entre las palabras que tienen
// numll letras, numO O's y numI I's
// numll3 son las letras que quedan por determinar
```

```
        n -= (total2 - (nSk(numll2 - b2 + 2, 2) + b2));
        ll numll3 = numll2 - numI;

        // Paso 4: Encontrar el numero de E's
        // Se hace directamente con la informacion que
        // tenemos
        ll numE = numll3 - n + 1;

        // Con toda la informacion que tenemos ya conocemos la palabra
        cout << numO << ' ' << numI << ' ' << numE << ' ' << numll - numO - numE - numI <<
endl;

  }
}
```

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