Palabras abcd

En una cadena de caracteres, una substring de longitud 4 es un conjunto de 4 caracteres consecutivos de la string. Una substring de longitud 4 es buena si y sólo si todos sus caracteres son distintos. Dada una string *s* que tiene solo las letras minúsculas *a, b, c, d,* contad cuantas substrings de longitud 4 buenas tiene.

Estrategia de la solución

Este primer problema era el más sencillo de todos, bastaba con iterar para todos los valores *i* desde 0 hasta *n - 4* y sumar uno a la respuesta para cada *i* tal que los caracteres en las posiciones *i, i+1, i+2, i+3* fuesen distintos. Esta última comprobación se podía realizar de varias maneras (con un if un poco largo o con un vector de booleanos que indicase los caracteres utilizados). Para resolver el primer subcaso no era necesario ni iterar.

Solución en Python

```python
ncases = int(input())

for case in range(ncases):
        n = int(input())
        word = input()

        check = ['a', 'b', 'c', 'd']

        ans = 0
        for i in range(n-3):
                # considero la subpalabra con indices de [i, i+3]
                substr = word[i:i+4]

                # el metodo sorted devuelve la lista de los caracteres ordenados
                substr = sorted(substr)

                # comparo la lista de la subpalabra ordenada con la lista check
                # si son iguales, esta subpalabra tiene los caracteres abcd
                if substr == check:
                        ans += 1

        print(ans)
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

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

using namespace std;

int main() {
    ios::sync_with_stdio(0);
```

```
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        string s;
        cin >> s;
        int ans = 0;
        for (int i = 0; i < n - 3; ++i) {
            vector<bool> letras(4, false);
            // marcamos las letras presentes en la substring actual
            for (int j = i; j <= i + 3; ++j)
                letras[s[j] - ' a '] = true;
            bool todas = true;
            // comprobamos si las 4 letras aparecen en la substring
            for (int j = 0; j < 4; ++j)
                todas = todas && letras[j];
            if (todas)
                ++ans;
        }
        cout << ans << endl;
    }
}
```

Autor de la solución: Jordi Rodríguez (Miembro del comité organizador)

Lanzamiento de pelota

Dean tiene una pelota muy especial con la siguiente propiedad: Cuando Dean la lanza, la pelota bota exactamente *k* veces y justo después del bote número *k*, se desvanece. Además cumple que si Dean la lanza de manera que recorra una distancia de *n* metros antes de hacer el primer bote, la pelota recorre una distancia de $\frac{n}{2}$ metros entre el primer bote y el segundo, de $\frac{n}{3}$ metros entre el segundo bote y el tercero, y así hasta el final. Es decir que antes del último bote recorre una distancia de $\frac{n}{k}$ metros. Dean quiere saber si es posible encontrar un *n* tal que la pelota recorra exactamente *m* metros antes de desvanecerse. Formalmente, encontrad un entero *n* tal que $n + \frac{n}{2} + \frac{n}{3} + ... + \frac{n}{k} = m$ o decid que no existe.

Estrategia de la solución

El segundo problema era una búsqueda binaria. En lugar de buscar si existe un entero *n* tal que $n + \frac{n}{2} + \frac{n}{3} + ... + \frac{n}{k} = m$, buscaremos el mínimo *n* tal que $n + \frac{n}{2} + \frac{n}{3} + ... + \frac{n}{k} \geq m$. Y una vez lo hayamos encontrado, miraremos si se da la igualdad o no. Puesto que la función $n + \frac{n}{2} + \frac{n}{3} + ... + \frac{n}{k}$ es creciente en *n*, podemos realizar la búsqueda binaria. Hay que tener en cuenta un último detalle de implementación: Si calculamos $n + \frac{n}{2} + \frac{n}{3} + ... + \frac{n}{k}$ para valores grandes de *n* y *k*, podemos tener overflow aún usando *long long*. Para evitar esto, podemos hacer que cuando la suma se vuelve mayor que un cierto número (10^{17} por ejemplo), la función que calcula la suma devuelva este número o un número grande.

Solución en Python

```python
# funcion para calcular la suma que pide el enunciado para un n determinado
def sum_floors(n, k):
	ans = 0
	for i in range(1, k+1):
		ans += n//i
	return ans

ncases = int(input())

for case in range(ncases):
	# cojo toda la linea como input
	line = input()

	# la divido en una lista con split
	# al primer elemento lo llamo m y al segundo k
	m, k = line.split(' ')[0:2]
```

```
        # los transformo en ints
        m, k = int(m), int(k)

        # inicio la busqueda binaria
        l, r = 0, m
        while r - l > 1:
                mid = (r+l)//2
                val = sum_floors(mid, k)

                # si val es demasiado pequeno, descarto el segmento de la izquierda
                if val < m: l = mid
                # si es demasiado grande, descarto el de la derecha
                else: r = mid

        # miro si el resultado es exacto
        if sum_floors(r, k) == m:
                print(int(r))
        else:
                print(-1)
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```
#include <iostream>

using namespace std;

long long calc(long long x, int k) {
    long long res = 0;
    for (int i = 1; i <= k; ++i) {
        res += x / i;
        if (res > 1e17)
            return res; // Para evitar overflows
    }
    return res;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        long long c;
        int k;
        cin >> c >> k;
        long long l = 0, r = 1e16, mid;
        while (l + 1 < r) {
            mid = (l + r) / 2;
            if (calc(mid, k) >= c)
                r = mid;
            else
```

```
                l = mid;
        }
        if (calc(r, k) == c)
            cout << r << endl;
        else
            cout << -1 << endl;
    }
}
```

Autor de la solución: Jordi Rodríguez (Miembro del comité organizador)

Números variados

Un número es variado si no tiene dos dígitos iguales consecutivos en su representación en base 10. Dados tres enteros *k*, *l* y *r*, encontrad el *k*-ésimo número variado en el intervalo *[l, r]* o decid que no existe.

Estrategia de la solución

El último problema también era una búsqueda binaria pero un poco más complicada que la del problema 2. Dado un número y un intervalo es difícil calcular el *k*-ésimo número variado del intervalo, pero es más fácil calcular cuántos números variados hay entre 0 y *x*. Supongamos que sabemos hacerlo. Entonces si tenemos *c* números variados entre 0 y *a-1*, encontrar el *k*-ésimo número variado del intervalo *[a, b]* es equivalente a encontrar el *(c+k)*-ésimo número variado del intervalo *[0, b]*. Ahora, con una búsqueda binaria sobre *r*, podemos hallar el mínimo entero *r* tal que haya exactamente *c+k* números variados en el intervalo *[0, r]*.
Veamos ahora cómo resolver la primera parte (encontrar la cantidad de números variados en el intervalo *[0, r]*). Lo haremos con una función recursiva. Si $r \leq 9$ entonces hay *r + 1* números variados en el intervalo *[0, r]* puesto que todos los números del intervalo tienen un único dígito. Para ver como se resuelve el caso $r \geq 10$ haremos un ejemplo. Tomemos *r = 1283*. Con una función que nos diga si un número es variado, contaremos los números variados en el intervalo *[1280, 1283]* y a este número añadiremos la cantidad de números variados en el intervalo *[0, 1279]*. Esta cantidad es exactamente 9 veces la cantidad de números variados en el intervalo *[0, 127]* más 1, porque para cada número variado *v* en el intervalo *[0, 127]* hay 9 en el intervalo *[0, 1279]* (podemos añadir cualquier dígito al final de *v* excepto el último dígito de *v* y el número seguirá siendo variado, así que tenemos 9 opciones). Hay que sumar 1 a esta cantidad porque no estamos contando que al 0, si le añadimos un 0 al final, sigue siendo variado, ya que 00 es en realidad 0.

Solución en Python

```
# devuelve si x es variado
def es_variado(x):
	last = -1
	while x != 0:
		if last == x%10:
			return 0
		last = x%10
		x //= 10
	return 1

# devuelve el número de números variados <= x
def nv_menores_que_x(x):
	# cas base
	if x <= 9:
		return x+1

	# crida recursiva
```

```
        ans = nv_menores_que_x(x//10 - 1)*9 + 1

        # casos delicados
        for i in range(x-x%10, x+1):
                ans += es_variado(i)

        return ans


ncases = int(input())

for case in range(ncases):
        line = input()

        k, l, r = line.split(' ')[0:3]
        k, l, r = int(k), int(l), int(r)

        k += nv_menores_que_x(l-1)

        # inicio la busqueda binaria
        l = 0
        while r-l > 1:
                mid = (r+l)//2
                val = nv_menores_que_x(mid)

                if val < k:
                        l = mid
                else:
                        r = mid

        # compruebo si hay solucion
        if nv_menores_que_x(r) == k:
                print(r)
        else:
                print(-1)
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```
#include <iostream>

using namespace std;

// Devuelve si un numero es variado
bool var(long long x) {
    int cur = -1;
    while (x) {
        if (cur == x % 10)
            return false;
        cur = x % 10;
        x /= 10;
    }
    return true;
```

```
}

long long calc(long long x) {
    if (x <= 9)
        return x + 1;
    long long res = calc(x / 10 - 1) * 9 + 1;
    for (long long i = x - x % 10; i <= x; ++i) {
        res += var(i);
    }
    return res;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    int t;
    cin >> t;
    while (t--) {
        long long a, b, k;
        cin >> k >> a >> b;
        k += calc(a - 1); // Sumamos numeros variados entre 0 y a-1
        long long l = 0, r = b, mid;
        while (l + 1 < r) {
            mid = (l + r) / 2;
            if (calc(mid) >= k)
                r = mid;
            else
                l = mid;
        }
        if (calc(r) < k)
            cout << -1 << endl;
        else
            cout << r << endl;
    }
}
```

Autor de la solución: Jordi Rodríguez (Miembro del comité organizador)