Clasificatorio AdaByron URJC 2023

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - ¿Cuántas rondas?	Adhoc, Bucles y Matemáticas.
B - Los Problemas de la Tecnología	Mapas, Long Long.
C - Jesters's League	Mapas, Arrays, Condicionales.
D - ¡Al rico café!	BFS, Grafos.
E - Bonito Dibujito	Recursividad.

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - ¿Cuántas rondas?	2	8.00 KB
B - Los Problemas de la Tecnología	25	7.13 MB
C - Jesters's League	33	26.00 MB
D - ¡Al rico café!	63	1.71 MB
E - Bonito Dibujito	7	5.14 MB
- **Total**	130	39.98 MB (+-)

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - ¿Cuántas rondas?	LongLongLovers	10
B - Los Problemas de la Tecnología	Teamto de Verano	6
C - Jesters's League	Teamto de Verano	22
D - ¡Al rico café!	MIIDAS AC?	16
E - Bonito Dibujito	¿?	¿?

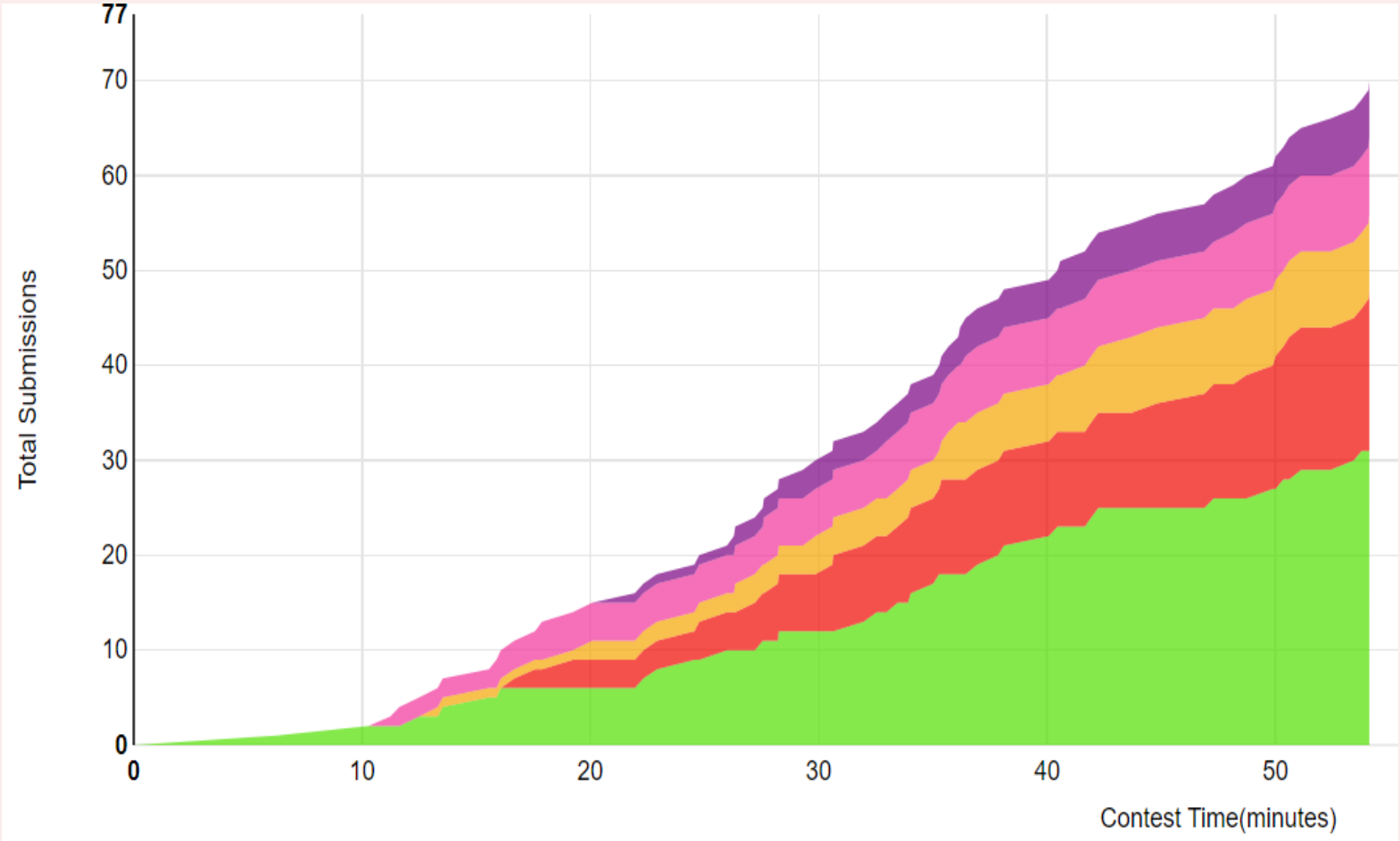
*Antes de congelar el marcador.

Estadísticas*

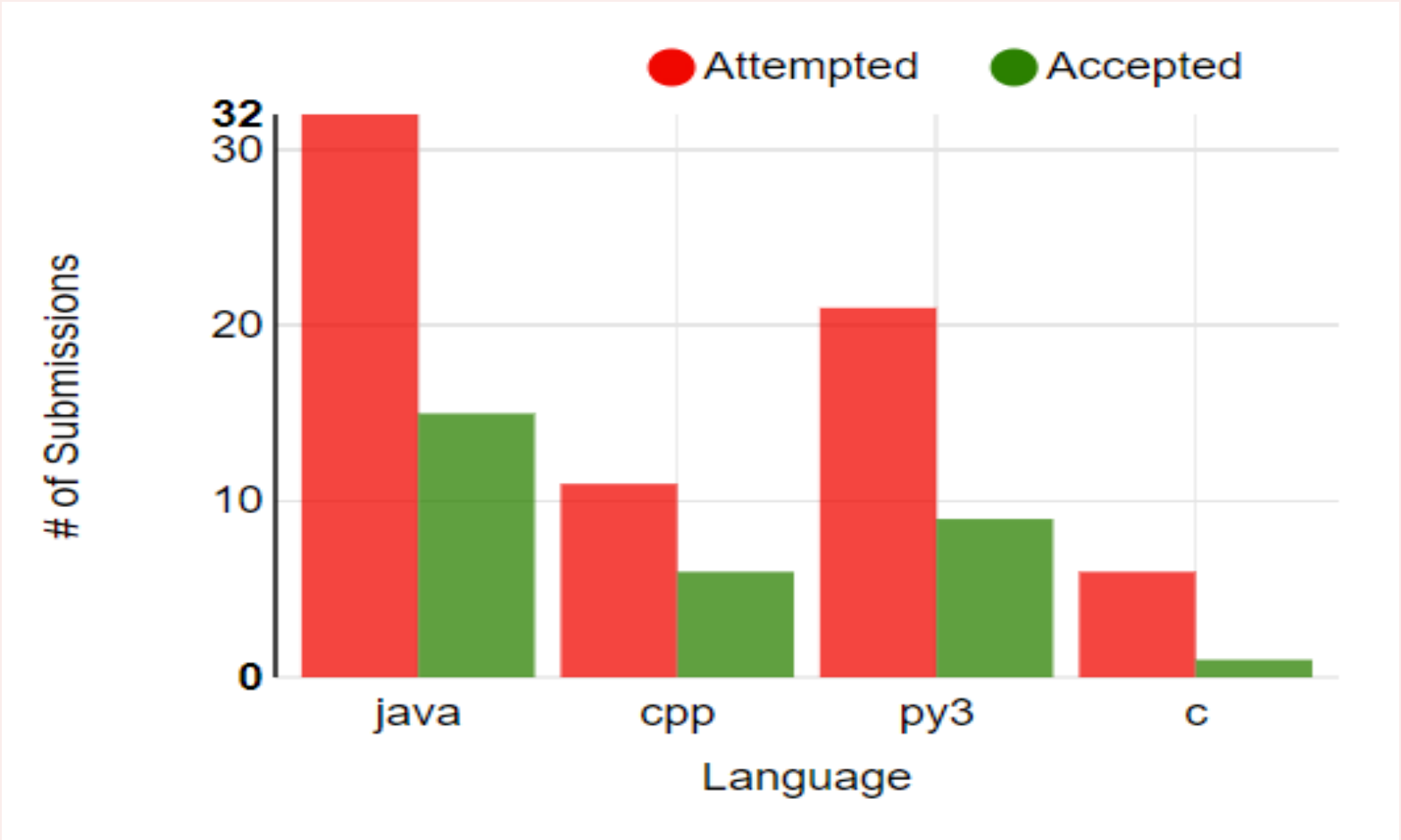
Problema	Válidos	Envíos	% éxito
A - ¿Cuántas rondas?	9	12	75 %
B - Los Problemas de la Tecnología	7	33	21 %
C - Jesters's League	6	10	60 %
D - ¡Al rico café!	7	10	70 %
E - Bonito Dibujito	0	0	- %

*Antes de congelar el marcador.

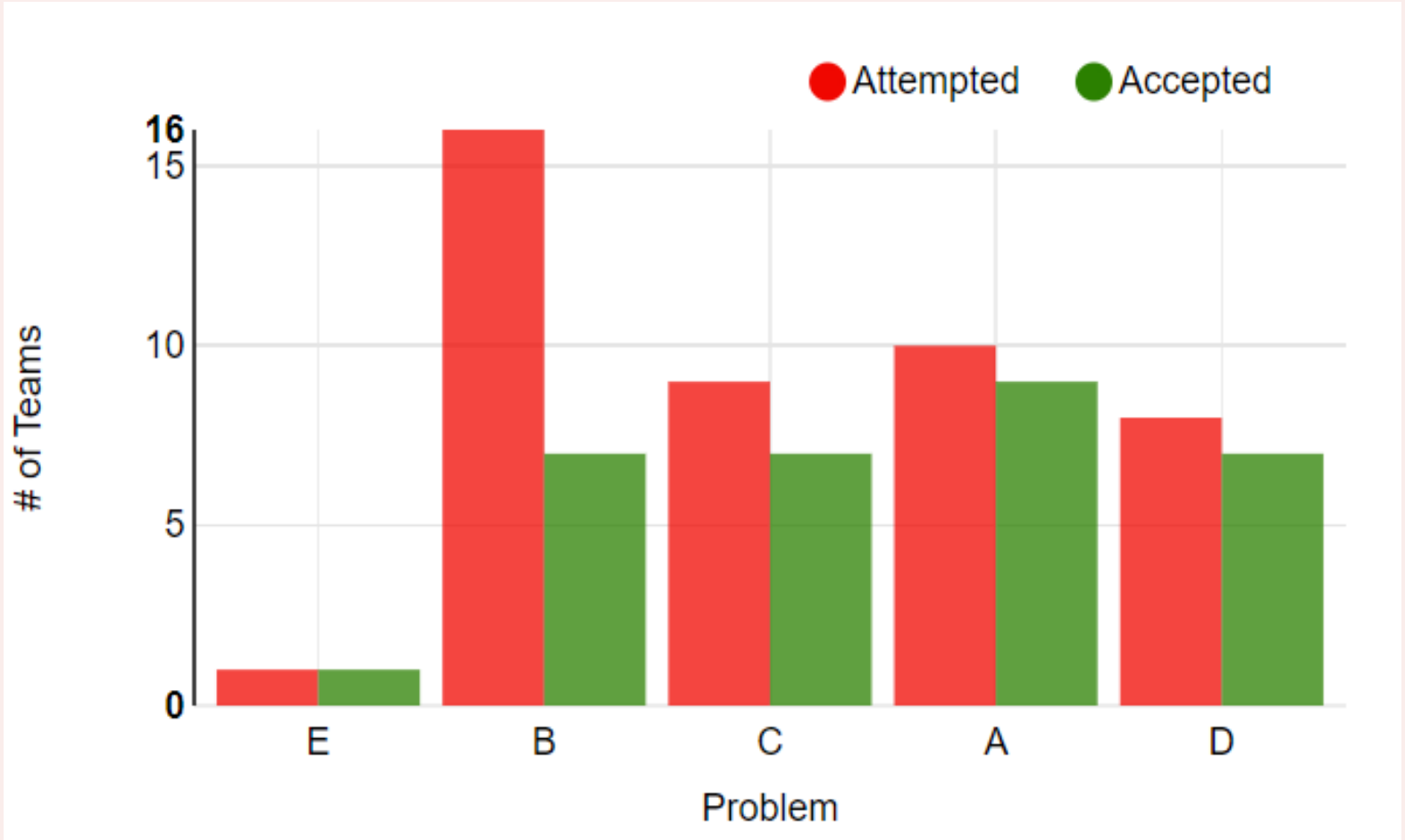
Estadísticas varias



Estadísticas varias



Estadísticas varias



A. ¿Cuántas rondas?

Envíos	Válidos	% éxito
9	12	75 %

A. ¿Cuántas rondas?

Dado un número de participantes, ¿cuántas rondas dan un ganador único en el mejor caso?
¿Cuántas veces podemos dividir entre dos hasta obtener 1 equipo?

$$R = log_2 n$$

A. ¿Cuántas rondas?

El algoritmo quedaría como:

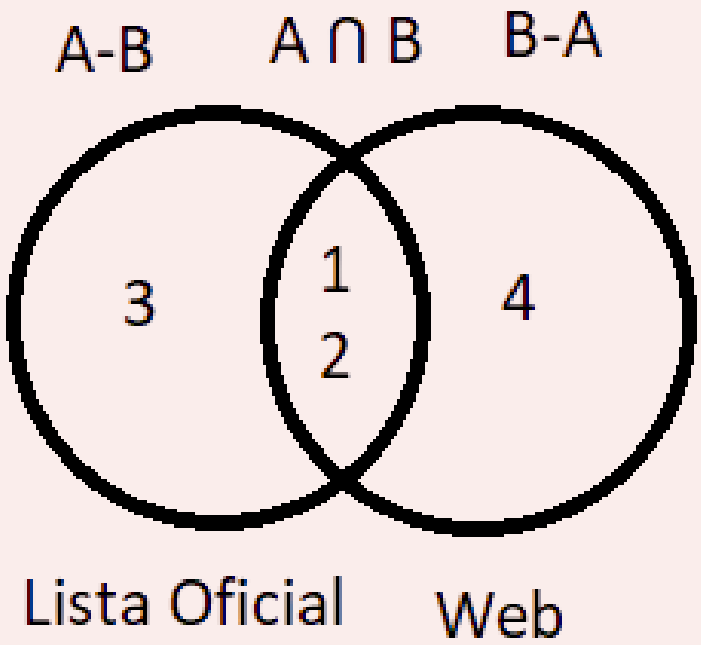
```
int res(int num){
    int cnt = 0;
    while (num >>= 1) cnt++;
    return cnt;
}

int cases; scanf("%d",&cases);
while(cases--){
    int n; scanf("%d",&n);
    printf("%d ",res(n));
}
```

B. Los problemas de la tecnología

Envíos	Válidos	% éxito
7	33	21 %

B. Los problemas de la tecnología

Dado dos listados de números A y B representado mediante conjuntos, devuelve la diferencia de A menos B ($A \setminus B$).



B. Los problemas de la tecnología

Errores comunes.

- No usar Long Long Int -> WA
- No devolver elementos ordenados -> WA
- Mostrar los elementos de B que no están en A (3 1 2 3 1 2 4 imprimir 3 4) -> WA

Posibles ideas...

Aproximación	Tiempo	Espacio	Veredicto
Array y recorrer por cada premio en web	$O(P + P^2)$	$O(P)$	TLE
2 Arrays ordenados y recorrer	$O(2 \cdot (P + PlogP) + P)$	$O(P + P)$	AC
Mapa sumando y restando	$O(P + P + P)$	$O(P)$	AC
Set añadiendo y eliminando	$O(P + P + P)$	$O(P)$	AC

Siendo P el número de premios.

B. Los problemas de la tecnología

El algoritmo quedaría como:

```
int P
long long int P_i
set<long long int> S
read P
for i in P
    read P_i
    S.insert(P_i);
for i in P
    read P_i
    S.erase(P_i);
bool flag = true,correct=true
for s in S
    if flag imprimir s
    else imprimir " " + s
    correct=false;
    flag=false;
if correct
    imprimir "OK"
```

D. ¡Al rico café!

Envíos	Válidos	% éxito
7	10	70 %

D. ¡Al rico café!

El enunciado nos habla de un sorteo especial de café en el que los participantes tienen que etiquetar a un número de amigos.

La novedad de este sorteo es que tenemos muchos ganadores. Una persona puede recibir el premio si ha sido nombrado ganador, o también si ha sido etiquetado por alguien que ha recibido el premio.

En la entrada del premio nos indican el número usuarios que han participado, y la lista de qué usuarios han sido etiquetados por qué otros usuarios.

Tareas a realizar:

- Analizar la entrada y almacenar la información en un grafo: los usuarios son los nodos, y las etiquetas serán las aristas del grafo.
- Partiendo del nodo ganador (última entrada del problema) ver cuántos usuarios (nodos) serían ganadores del premio.
- Comprobar si la lista de ganadores es igual que la lista de usuarios que han participado.

D. ¡Al rico café!

Este problema se podría solucionar utilizando el algoritmo BFS, *Breath First Search* o DFS, *Depth-first search*.

El algoritmo quedaría como:

```
def bfs(edges, to_analize):
    while len(to_analize) > 0:
        source = to_analize[0]
        to_analize.remove(source)
        visited.append(source)

        amigos = edges[source]
        for amigo in amigos:
            if amigo not in visited and amigo not in to_analize:
                to_analize.append(amigo)

    if len(visited) < len(edges):
        print("Alguien se queda sin premio")
    else:
        print("Todos ganadores")
```

E. Bonito Dibujito

Envíos	Válidos	% éxito
0	0	- %

E. Bonito Dibujito

Dada un nivel de recurrencia o profundidad, dibuja el fractal generado como un String o char[][].

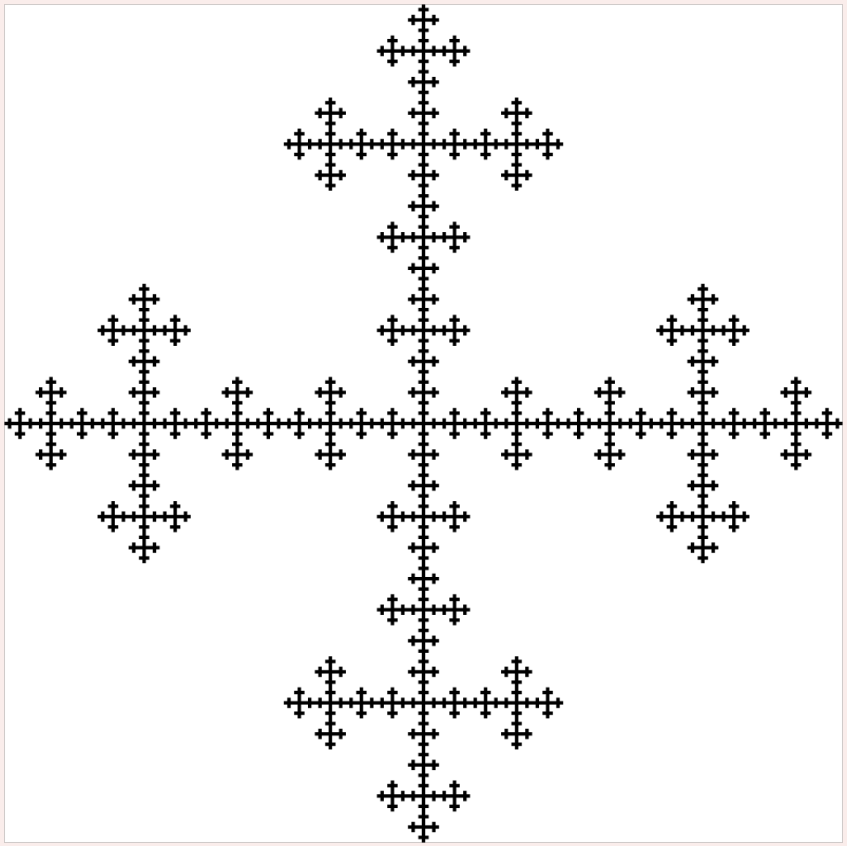


Figura: Ejemplo de fractal

E. Bonito Dibujito

La clave es darse cuenta de cual es la recurrencia, en este caso, al dividir dividir el cuadrado en 3x3, se repite en el cuadrado central y los que solapan con el. Dada un nivel de recurrencia o profundidad, dibuja el fractal generado como un String o char[][].

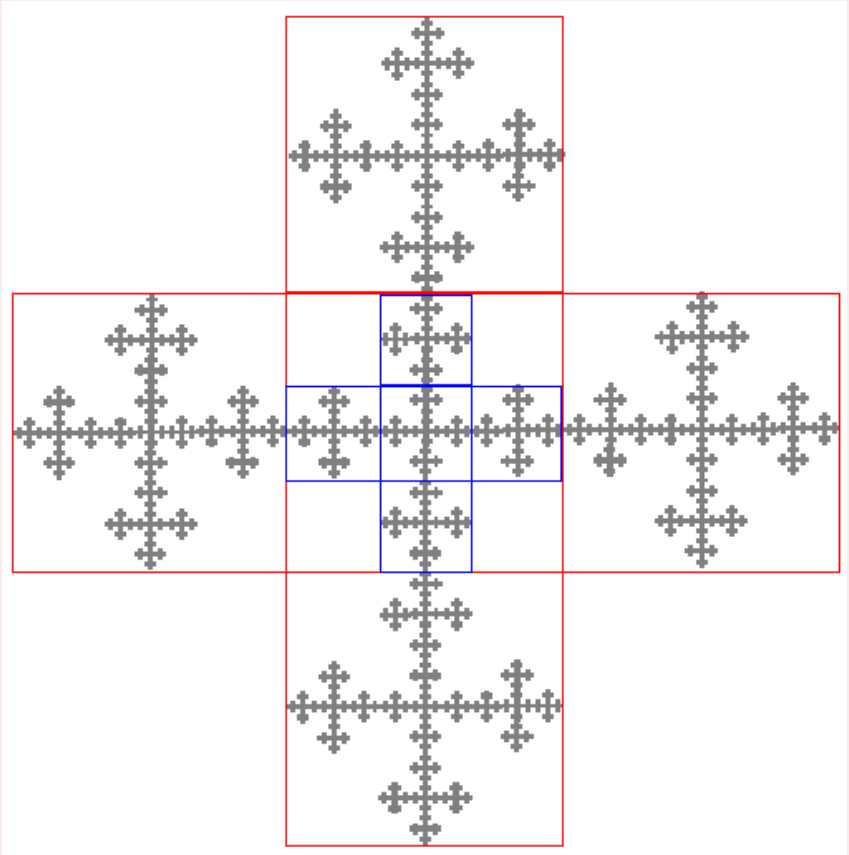


Figura: Recurrencia del fractal

E. Bonito Dibujito

Una vez detectada la recurrencia, el problema es fácil de resolver.

- Solución mala: Generarlo a mano
- Solución correcta: ¡Recursividad!

E. Bonito Dibujito

```
void r(char[][] m, int size, int x, int y) {
    if (size >= 3) {
        int ns = size / 3; // newSize
        r(m, ns, x + ns,     y);
        r(m, ns, x,          y + ns);
        r(m, ns, x + ns,     y + ns);
        r(m, ns, x + 2 * ns, y + newSize);
        r(m, ns, x + ns, y + 2 * ns);
    } else if (size == 1) {
        // Base case: paint current tile
        m[x][y] = 'X';
    } else { /* Impossible, throw */ }
}
```

E. Bonito Dibujito

```
void main() {
    int n = readInt();
    int width = pow(3, n);
    char[][] m = new char[width][width];
    fill(m, ' ');
    r(m, width, 0, 0);
    print(m);
}
```

C. Jesters's League

Envíos	Válidos	% éxito
6	10	60 %

C. Jesters's League

El equipo de Piqué, quiere un programa que le ayude a gestionar los equipos de su "Jesters's League". Concretamente, desean adquirir un s*software* que controle que un jugador no juegue con más de un equipo a lo largo de la liga.
Tareas

- Identificar el número de líneas que será necesario procesar.
- Extraer de cada línea de la entrada el equipo y el jugador.
- Almacenar el equipo en el que ha jugado cada uno de los jugadores.
- Para la jornada a detectar errores, comprobar si algún jugador ha jugado en un equipo distinto al que se indica.

C. Jesters's League

Ideas algorítmicas buenas y malas

- Usar una estructura como un mapa o diccionario ☺.
- Utilizar una lista ☹. Acceder a la información es poco eficiente.
- Utilizar como clave los equipos y almacenar los jugadores ☹. Determinar si un jugador a jugado en algún equipo requiere recorrer todos los equipos.
- Utilizar como clave el jugador y asignarle el equipo que ha jugado ☺. Solo puede jugar en un equipo y no hay errores.

C. Jesters's League

El algoritmo quedaría como:

```
def solve_jester():
    j, e = read_first_line()
    players = {}
    for _ in range(j * e):
        team, player = read_team_player()
        players[player] = team
    flag = False
    for _ in range(e):
        team, player = read_team_player()
        if player in players and players[player] != team:
            print(player)
            flag = True
    if not flag:
        print("correcto")
```