Concurso Final 2024/2025

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - Microondas	DP, mochila.
B - Bonsai	RMQ, Segment Tree
C - Anchas escaleras	DP, contar caminos.
D - Dijkstraídos	Camino mínimo en grafos.
E - Derrapando en la mediana	Ordenación/Ad hoc.
G - Buscando bollos	Ad hoc

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - Microondas	10	88 KB
B - Bonsai	6	90 MB
C - Anchas escaleras	12	300 B
D - Dijkstraídos	8	172 KB
E - Derrapando en la mediana	8	2 MB
G - Buscando bollos	7	6 MB
- **Total**	36	(+-) 102 MB

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - Microondas	r.martinsa.2024	6
B - Bonsai	r.martinsa.2024	14
C - Anchas escaleras	osama been lagging	7
D - -	Corky Ávila	6
E - -	Corky Ávila	9
F - ¡Saca tú!	Campo obligatorio	3
G - Buscando bollos	Jestein & Jasmoug	5

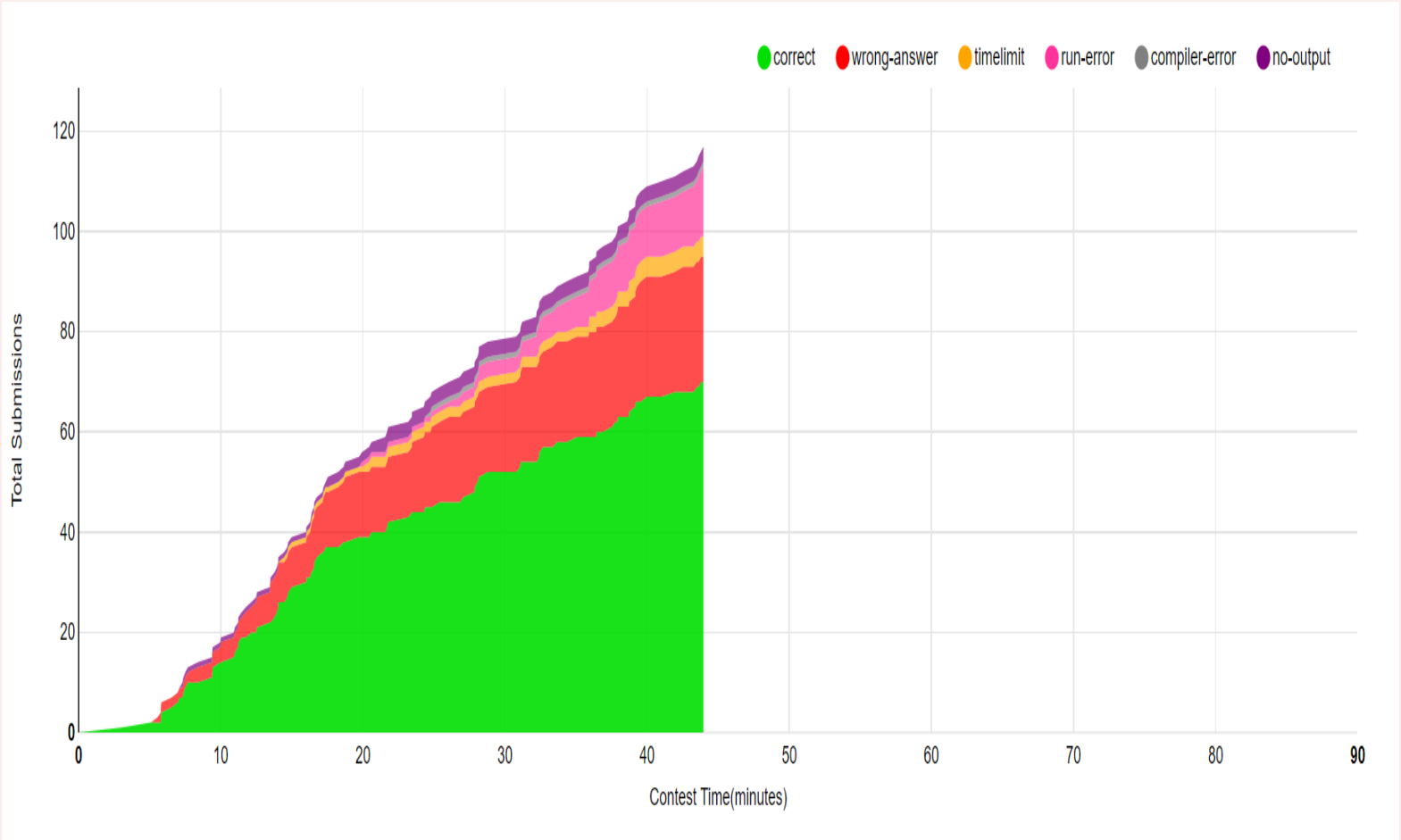
*Antes de congelar el marcador.

Estadísticas*

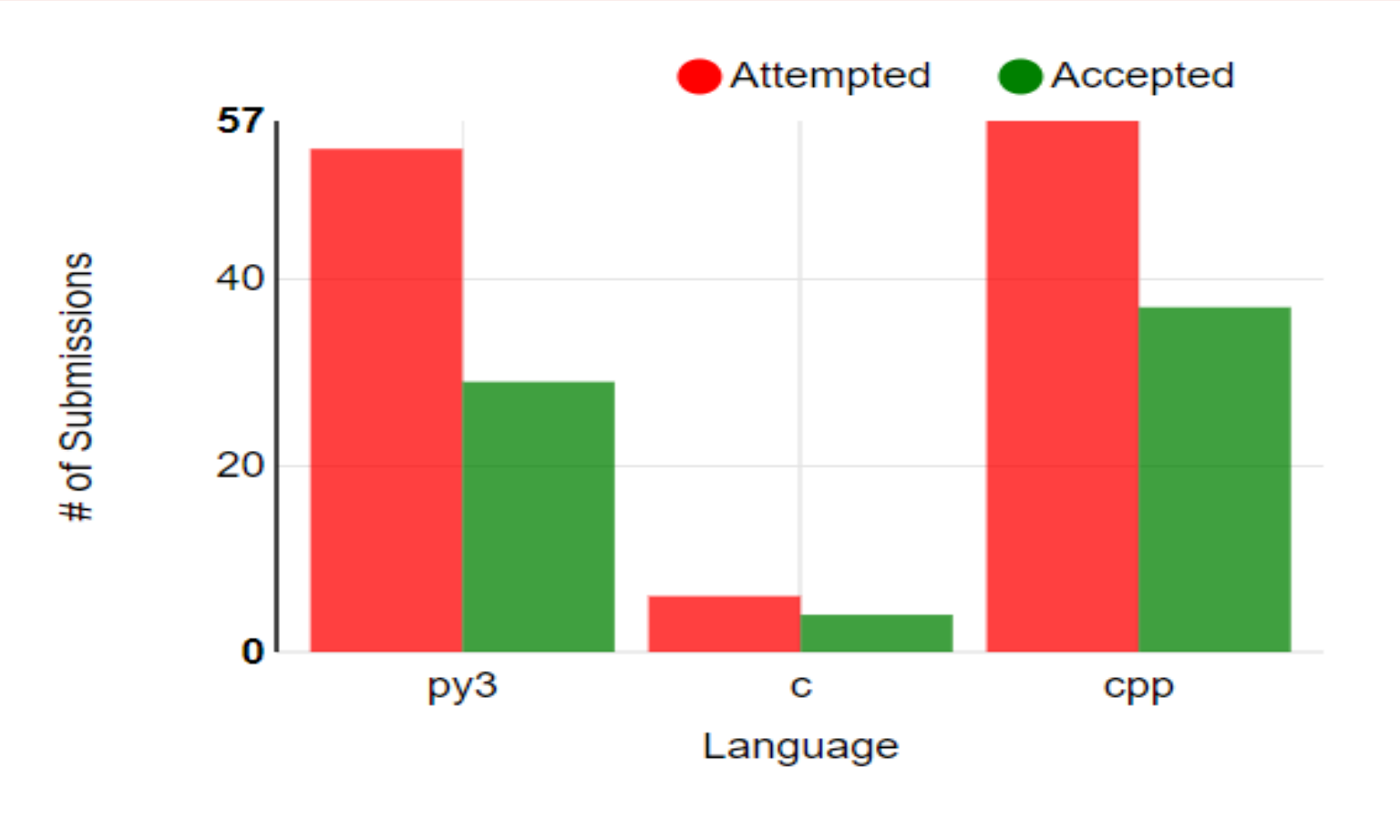
Problema	Válidos	Envíos	% éxito
A - Microondas	9	16	56 %
B - Bonsai	3	10	30 %
C - Anchas escaleras	6	8	75 %
D - Envíos Prioritarios	8	15	53 %
E - Una ronda más	20	23	87 %
F - ¡Saca tú!	21	29	72 %
G - Buscando bollos	2	12	17 %

*Antes de congelar el marcador.

Estadísticas varias



Estadísticas varias



Estadísticas varias

A. Microondas

Envíos	Válidos	% éxito
16	9	56 %

A. Microondas

Tenemos:

- N tuppers
  - t_i tiempo para que se caliente
  - p_i puntuación
- T tiempo máximo de uso del microondas

A. Microondas

MOCHILA

A. Microondas

Ordenar los tuppers en fila. Recorrer la fila y elegir calentar o no calentar cada tupper mientras no nos pasemos del tiempo máximo.

A. Microondas

```
microondas(i, t_act):
    if t_act > T:
        return -infinito
    if i == N:
        return 0
    calentar = microondas(i+1, t_act+t_i) + p_i
    no_calentar = microondas(i+1, t_act)
    return max(calentar, no_calentar)
```

A. Microondas

```
microondas(i, t_act):
    if t_act > T:
        return -infinito
    if i == N:
        return 0
    if  memo[i][t_act]:
        return memo[i][t_act]
    calentar = microondas(i+1, t_act+t_i) + p_i
    no_calentar = microondas(i+1, t_act)
    memo[i][t_act] = max(calentar, no_calentar)
    return memo[i][t_act]
```

A. Microondas

```
leer(T)
leer(N)
leer(tuppers)

inicializar(memo)

max_punt = microondas(0, 0)

imprimir(max_punt)
```

B. Bonsai

Envíos	Válidos	% éxito
10	3	30 %

B. Bonsai

Tenemos una secuencia de números (litros de agua de cada planta):

$$[e_1, e_2, \ldots, e_i, \ldots, e_n]$$

B. Bonsai

Tenemos una secuencia de números (litros de agua de cada planta):

$$[e_1, e_2, \ldots, e_i, \ldots, e_n]$$

Esta secuencia hay que transformarla en (raíces de cada planta):

$$[e_1(e_1 - 1), e_2(e_2 - 1), \ldots, e_i(e_i - 1), \ldots, e_n(e_n - 1)]$$

B. Bonsai

$$[e_1(e_1-1), e_2(e_2-1), \ldots, e_i(e_i-1), \ldots, e_n(e_n-1)]$$

Iterar en esta secuencia de números, desde la posición l a la posición r e ir sumando en cada consulta: **TLE**

B. Bonsai

Utilizar un **array de acumulados** evita este **TLE**.

¿Problema?

B. Bonsai

Utilizar un **array de acumulados** evita este **TLE**.

¿Problema?

Hay consultas donde hay que actualizar valores, luego el array de acumulados pierde su utilidad.

B. Bonsai

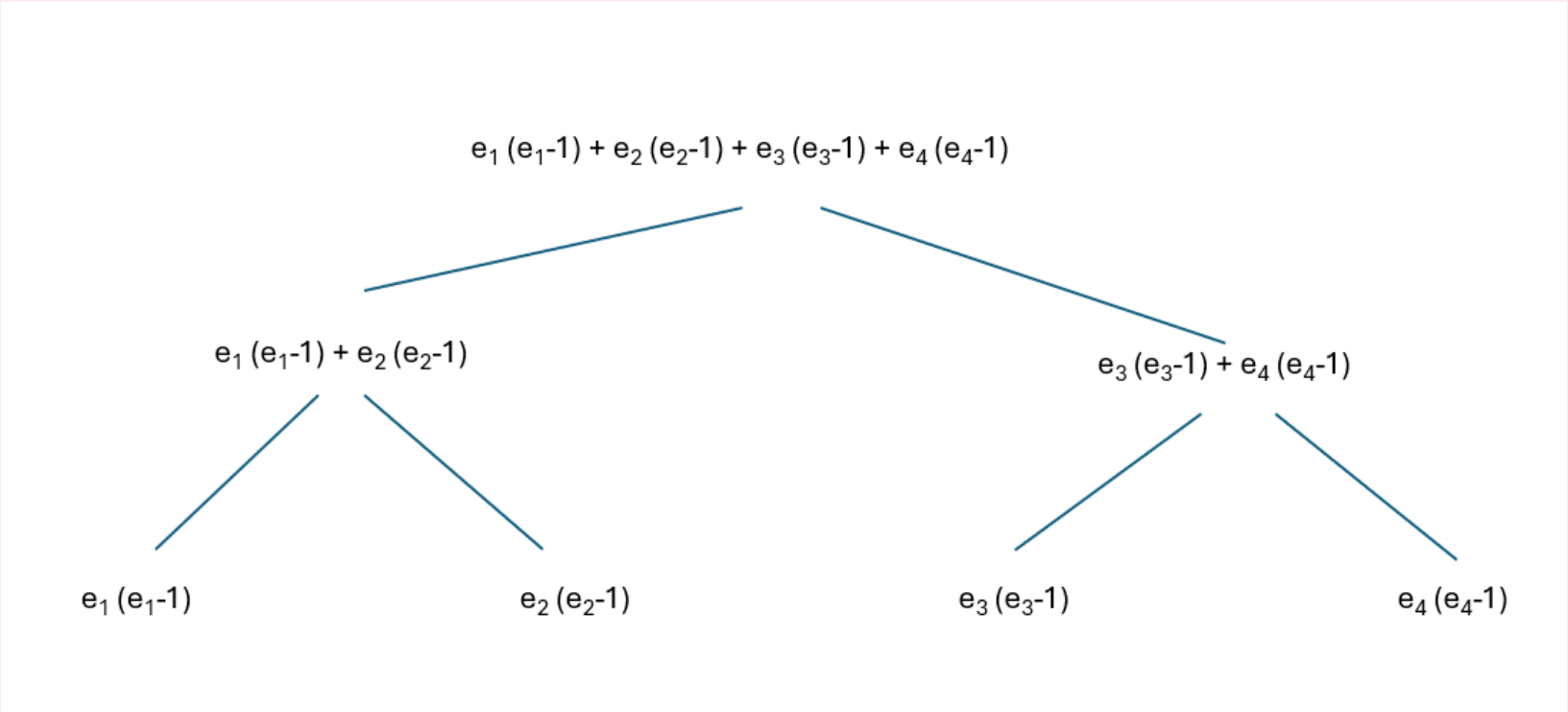
¿Solución?

B. Bonsai

¿Solución?

Segment Tree

B. Bonsai


$e_1\,(e_1\text{-}1) + e_2\,(e_2\text{-}1) + e_3\,(e_3\text{-}1) + e_4\,(e_4\text{-}1)$
$e_1\,(e_1\text{-}1) + e_2\,(e_2\text{-}1)$
$e_3\,(e_3\text{-}1) + e_4\,(e_4\text{-}1)$
$e_1\,(e_1\text{-}1)$
$e_2\,(e_2\text{-}1)$
$e_3\,(e_3\text{-}1)$
$e_4\,(e_4\text{-}1)$

B. Bonsai

Cada consulta tiene una complejidad de:

$$\mathcal{O}(\log(N))$$

Os pasaremos más información sobre esta nueva estructura de datos!

B. Bonsai

```
n, m = map(int, input().split())
array = list(map(int, input().split()))
for i in range(n):
    array[i] = array[i]*array[i]-array[i]
st = SegmentTree(array)
for _ in range(m):
    line = list(map(int, input().split()))
    op = line[0]
    if op == 1:
        ind = line[1]
        num = line[2]
        st.set(ind, num * (num - 1))
    else:
        l = line[1]
        r = line[2]
        print(st.query(l, r))
```

C. Anchas escaleras

Envíos	Válidos	% éxito
8	6	75 %

C. Anchas escaleras

En cada escalón puedes bajar de golpe g_i escalones. Quieres saber cuántas maneras hay de bajar un número E de escaleras.

C. Anchas escaleras

Idea de fuerza bruta:

- En cada escalón bajar g_1, g_2, g_3... escalones y ver si es una manera válida.
- Una manera es válida si no te pasa del último escalón.

CUIDADO CON EL MÓDULO!!

C. Anchas escaleras

```
bajar(i):
    if i<0:
        return 0
    if i==0:
        return 1
    maneras = 0
    for g_i in g:
        maneras = (maneras + bajar(i-g_i))%1000000007
    return maneras
```

C. Anchas escaleras

```
bajar(i):
    if i<0:
        return 0
    if i==0:
        return 1
    if memo[i]:
        return memo[i]
    maneras = 0
    for g_i in g:
        maneras = (maneras + bajar(i-g_i))%1000000007
    memo[i] = maneras
    return memo[i]
```

C. Anchas escaleras

```
leer(E)
leer(G)
leer(g)

inicializar(memo)

maneras = bajar(E)

imprimir(maneras[i])
```

D. Envíos Prioritarios

Envíos	Válidos	% éxito
15	8	53 %

D. Envíos Prioritarios

En el problema llegan paquetes (identificados como “P2W” o “NOR”) y despachos de camiones con capacidad limitada que recogen los paquetes en cola. Cada paquete se identifica según su orden de aparición y, tanto en la cola como en el camión, los “P2W” deben tener prioridad sobre los paquetes “NOR”, respetando siempre el orden de llegada.

D. Envíos Prioritarios

Algoritmo de Dijkstra

```
n = int(input())
prior, desp, idx = deque(), deque(), 0
for _ in range(n):
    a = input().split()
    if a[0] == 'R':
        cap = int(a[1])
        r = []
        while prior and cap:
            r.append(str(prior.popleft()))
            cap -= 1
        while desp and cap:
            r.append(str(desp.popleft()))
            cap -= 1
        print(r)
    elif a[0] == 'NOR':
        desp.append(idx); idx += 1
    elif a[0] == 'P2W':
        prior.append(idx); idx += 1
```

E. Una ronda más

Envíos	Válidos	% éxito
23	20	87 %

E. Una ronda más

En este problema, varios profesores salen a tomar refrescos después de clase y cada uno anota la cantidad consumida y el precio unitario de su bebida, excepto uno que olvida su precio. Se conoce el total de la cuenta y, utilizando los datos de los demás profesores, debemos calcular el precio unitario del refresco del profesor olvidadizo para que la suma total de los consumos coincida con el total dado.

E. Una ronda más

```
while not input:
    N = int(input())
    if N == 0: break
    total = 0
    X = -1
    for i in range(N):
        C = int(input())
        cont += 1
        P = int(input())
        cont += 1
        if P == -1:  X = C
        else:  total = total - C*P
    total += int(input())
    cont +=1
    print(int(total/X))
```

F. ¡Saca tú!

Envíos	Válidos	% éxito
29	21	72 %

F. ¡Saca tú!

En este problema se debe determinar el lado de la mesa desde donde se realiza el próximo saque en un juego de ping-pong, donde el saque se alterna en cada punto jugado. Al inicio, con el marcador 0-0, el saque se hace en el lado derecho y, tras cada punto, se cambia de lado, alternando entre derecha e izquierda.

F. ¡Saca tú!

Por lo tanto, dependiendo de la suma total de puntos anotados por ambos jugadores, si es par el siguiente saque será desde la derecha, y si es impar, desde la izquierda

F. ¡Saca tú!

```
N = int(input())
cont += 1
for i in range(N):
    a,b = map(int, input().strip().split())
    cont += 1
    if (a+b)%2 == 0:
        print("DERECHA")
    else:
        print("IZQUIERDA")
```

G. Buscando bollos

Envíos	Válidos	% éxito
12	2	17 %

G. Buscando bollos

¿Podremos salir del laberinto siguiendo la regla de la **mano izquierda**?

G. Buscando bollos

¿Hace falta simularlo?

G. Buscando bollos

¿Cuándo no podríamos salir de un laberinto con esta técnica?

G. Buscando bollos

```
##############.####
#................#
#.....########...#
#.....#......#...#
#.....#.######...#
#................#
####################
```

G. Buscando bollos

```
###############.###
#..............###
#.....#####...#.#
#.....#...#...###
#.....#####.....#
#................#
##########.#######
```

G. Buscando bollos

Conclusión: puesto que la entrada y la salida de la sala están en la **misma pared** (la pared exterior),

SIEMPRE podremos salir utilizando la regla de la mano izquierda.

G. Buscando bollos

```
int t
for(;t>0;t--)
    int n
    int m
    for(;n>0;n--)
        readLine();
    print("Hay bollos")
```