AdaByron Regional Madrid 2022

Estadísticas y Soluciones

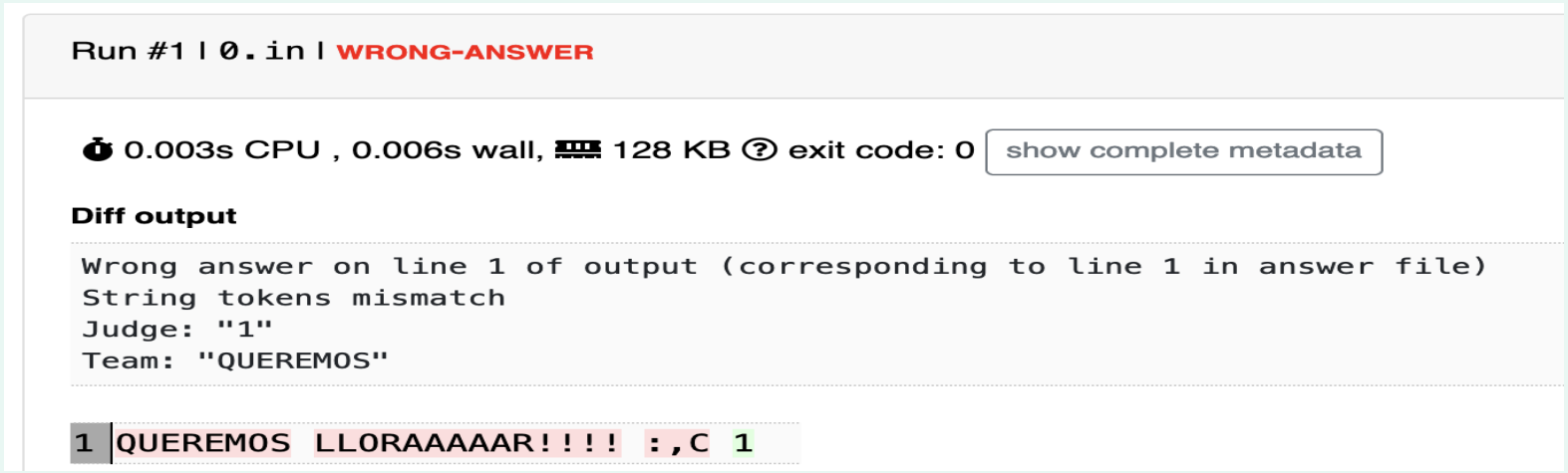
UNIVERSIDAD
POLITÉCNICA
DE MADRID

ETSI SISTEMAS
INFORMÁTICOS

¿Qué problema os ha gustado más?

Estadísticas frikis

- Número de clarifications - 117
- Número de envíos - 778
- Número de envíos por los jueces - 213
- Número de commits - 125
- Número de “sin comentarios” - 36
- ¿Bonsais salvados? - 23472783742374236
- Problemas con long long fallados - MUCHOS
- Códigos corregidos a mano - El juez se encarga :-)



Clasificación de los problemas

Problema	Categoría
A - Ayudando a los camareros	Programación Dinámica
B - ¡¡¡Explosiones cuadradas!!!	Geometría, algebra lineal
C - Robo perfecto en Villa Rejilla	Geometría, algebra lineal
D - El juego del pañuelo	Cardinalidad máxima, grafo bipartido, máximo flujo
E - Wordle: el juego	Bucles, arrays
F - La conquista del espacio	Grafos, backtracking
G - El hacker al que le sobraba el dinero	Programación dinámica
H - Guerra intergaláctica	Grafos, puntos de articulación
I - Bajo Presión	Estructuras
J - El juego de la piedra	Módulo, pensar
K - ¡A Fortnitear!	Voraz
L - Los primos traviesos	Sieve + Busqueda binaria
M - Jakub y los cuadrados	Matemáticas square-free

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - Ayudando a los camareros	2	4 KB
B - ¡¡¡Explosiones cuadradas!!!	2	400B
C - Robo perfecto en Villa Rejilla	3	200B
D - El juego del pañuelo	8	4.5 MB
E - Wordle: el juego	17	115 KB
F - La conquista del espacio	8	2KB
G - El hacker al que le sobraba el dinero	22	23.8 MB
H - Guerra intergaláctica	19	14.1 MB
I - Bajo Presión	9	1.11 MB
J - El juego de la piedra	5	89 KB
K - ¡A Fortnitear!	10	110 KB
L - Los primos traviesos	11	10 KB
M - Jakub y los cuadrados	31	15 KB
- **Total**	147	35 MB (+-)

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - Ayudando a los camareros	C intenta otra vez	1:03
B - ¡¡¡Explosiones cuadradas!!!	Los braquistócronos	1:14
C - Robo perfecto en Villa Rejilla	–	–
D - El juego del pañuelo	_(/)_/	0:49
E - Wordle: el juego	_(/)_/	0:11
F - La conquista del espacio	–	–
G - El hacker al que le sobraba el dinero	C intenta otra vez	0:03
H - Guerra intergaláctica	Echo	0:41
I - Bajo Presión	TheProgrammers	0:31
J - El juego de la piedra	C-ANSInos	0:11
K - ¡A Fortnitear!	–	–
L - Los primos traviesos	_(/)_/	2:19
M - Jakub y los cuadrados	–	–

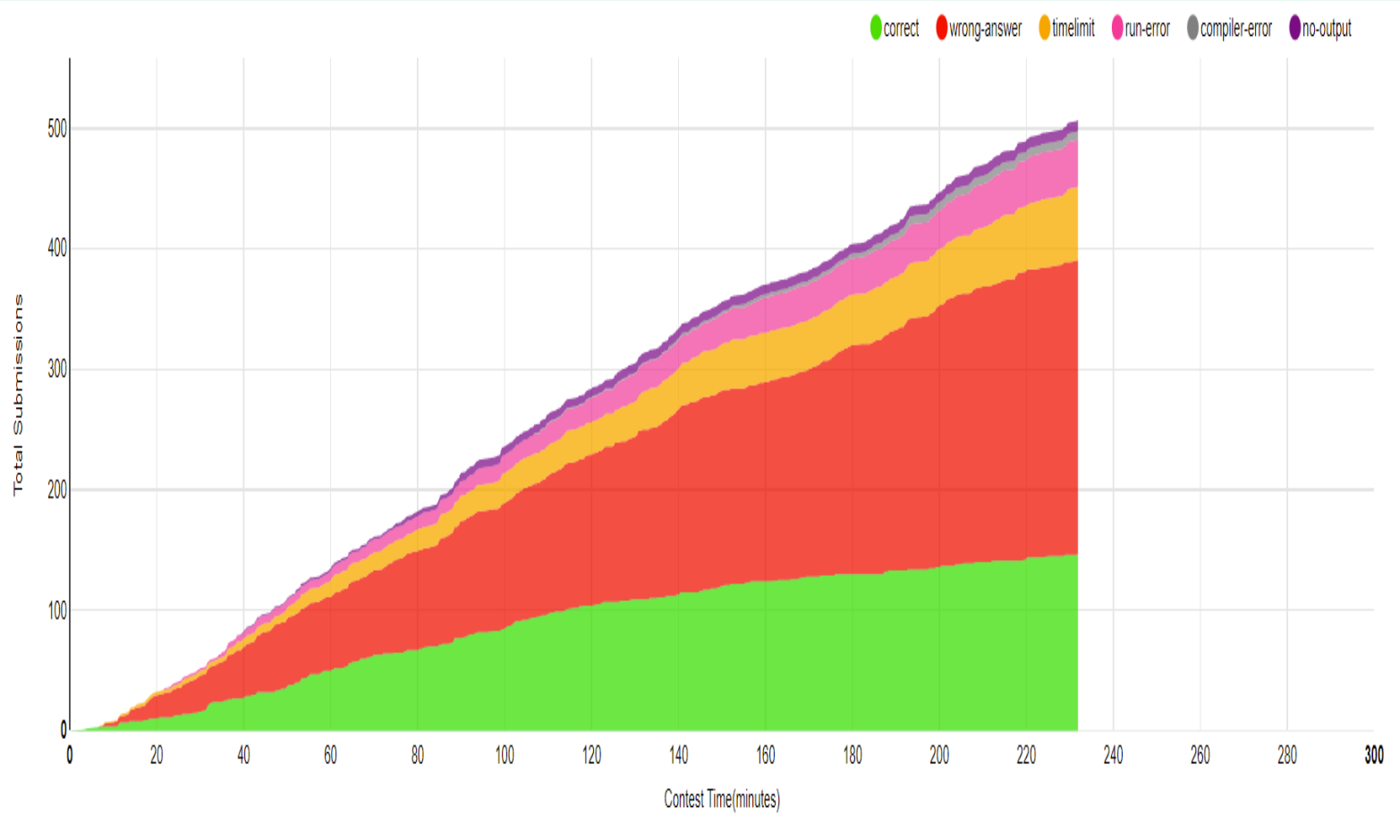
*Antes de congelar el marcador.

Estadísticas*

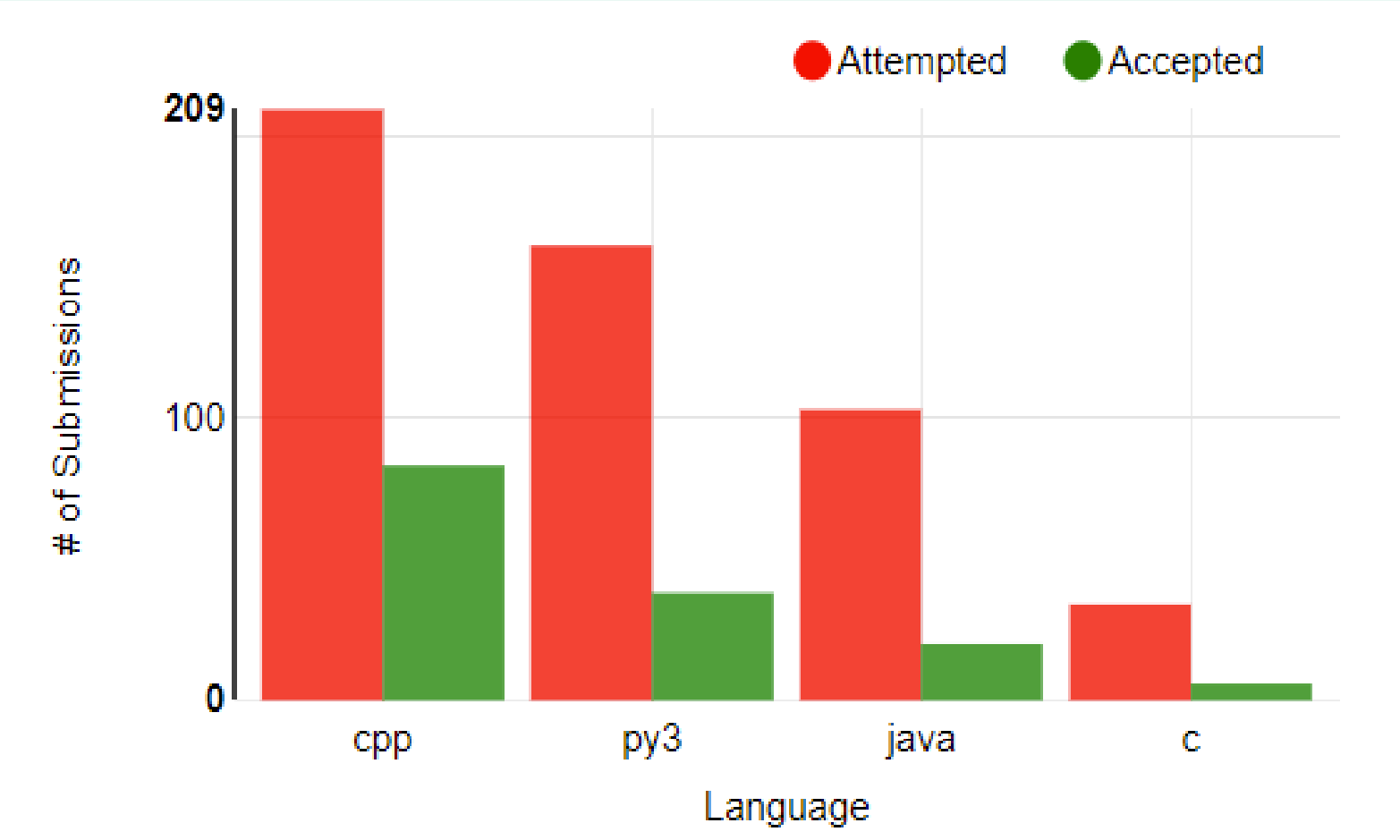
Problema	Envíos	Válidos	% éxito
A - Ayudando a los camareros	29	14	48.27 %
B - ¡¡¡Explosiones cuadradas!!!	1	1	100 %
C - Robo perfecto en Villa Rejilla	0	0	0 %
D - El juego del pañuelo	26	4	15.38 %
E - Wordle: el juego	81	35	43.20 %
F - La conquista del espacio	1	0	0 %
G - El hacker al que le sobraba el dinero	143	15	10.49 %
H - Guerra intergaláctica	20	4	20.00 %
I - Bajo Presión	59	28	47.45 %
J - El juego de la piedra	79	34	43.03 %
K - ¡A Fortnitear!	0	0	0 %
L - Los primos traviesos	15	5	33.3 %
M - Jakub y los cuadrados	9	0	0 %

*Antes de congelar el marcador.

Estadísticas varias



Estadísticas varias



Estadísticas varias

Run #1 | 0.in | WRONG-ANSWER

0.003s CPU , 0.006s wall, 128 KB exit code: 0 show complete metadata

Diff output

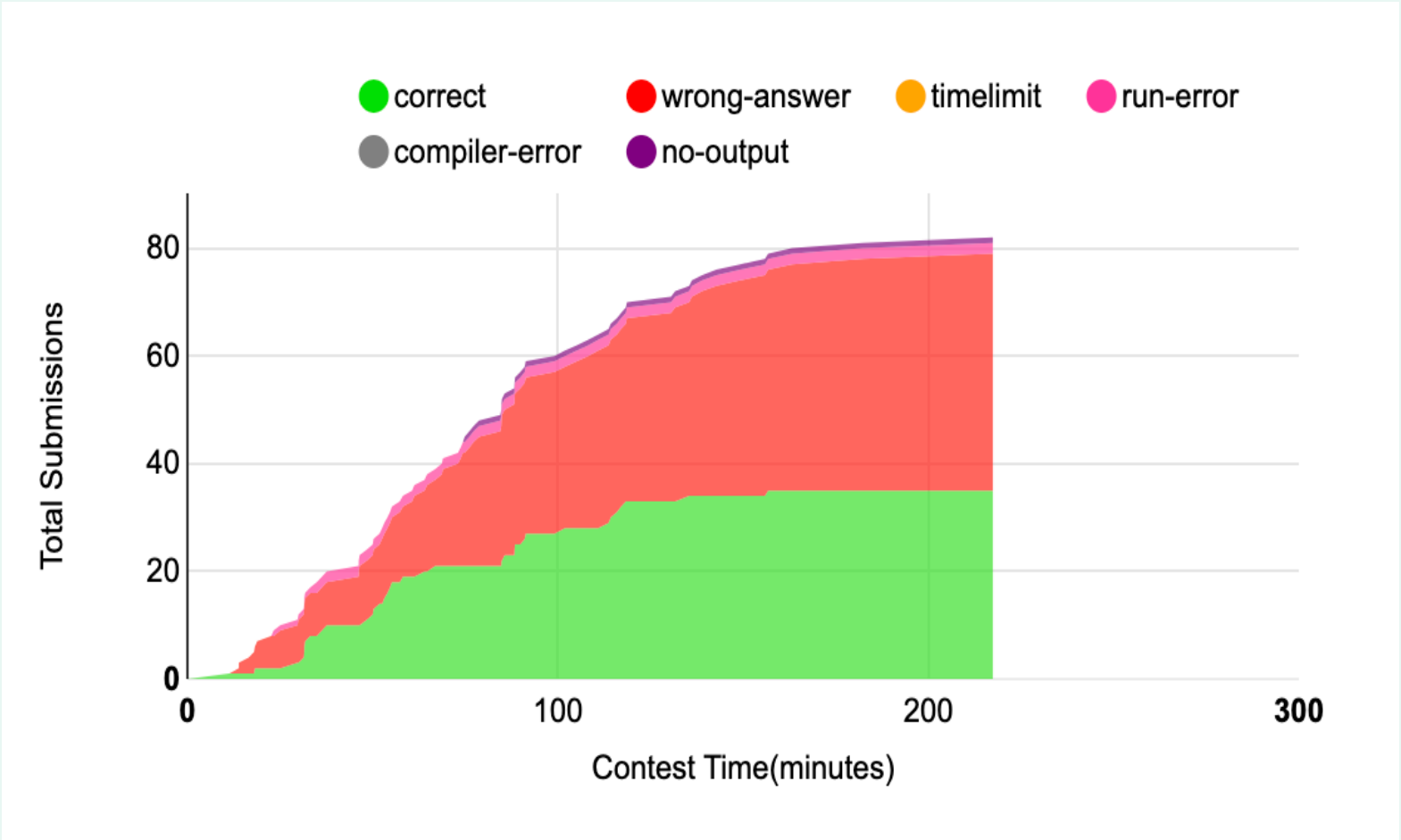
```
Wrong answer on line 1 of output (corresponding to line 1 in answer file)
String tokens mismatch
Judge: "1"
Team: "QUEREMOS"
```

```
1 QUEREMOS LLORAAAAAR!!!! :,C 1
```

E. Wordle

Envíos	Válidos	% éxito
81	35	43.20 %

E. Wordle


correct
wrong-answer
timelimit
run-error
compiler-error
no-output
Total Submissions
80
60
40
20
0
0
100
200
300
Contest Time(minutes)

E. Wordle

En este problema se simula una partida del famoso juego Wordle. Concretamente, se simula un número n de partidas. Para cada partida se debe leer, en primer lugar, la palabra a adivinar y, a continuación, se podrá leer un mínimo de 1 palabra (si lo palabra a adivinar se encuentra en el primer inento) y un máximo de 6 palabras (si la palabra a adivinar no se adivina). Además, todas las palabras tienen un tamaño fijo de 5 caracteres.
Se pide que, tras procesar las n partidas se muestre el porcentaje de partidas ganadas y el porcentaje de partidas ganadas con 1, 2, 3, 5 y 6 intentos.

E. Wordle

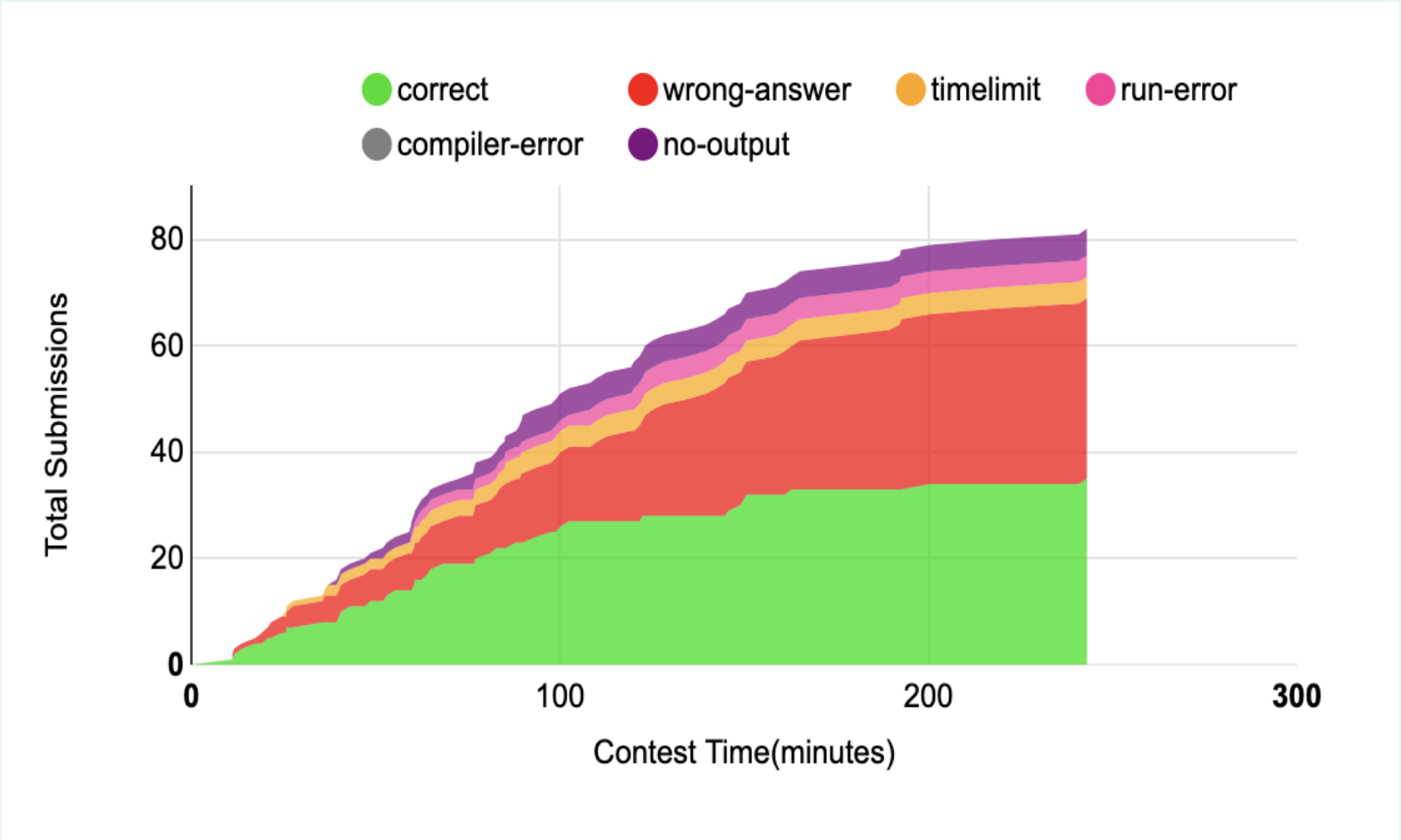
Tomando en cuenta lo anterior, podemos calcular el resultado de la siguiente manera:

```
num_partidas = input()
partidas_ganadas = 0
max_intentos = 6
intento = [0,0,0,0,0,0]
for(num_partidas) {
    palabra_oculta = input()
    for(i=0 hasta max_intentos) {
        prediccion = input()
        if prediccion == palabra_oculta{
            intento[i]++
            partidas_ganadas++
            break
        }
    }
}
mostrar_resultados...
```

J. El Juego de la piedra

Envíos	Válidos	% éxito
79	34	43.03 %

J. El Juego de la piedra



J. El Juego de la piedra

En este ejercicio nos explican las reglas de un juego clásico, varios jugadores cogen piedras de cualquierda de los montones y el que coja la última piedra del último montón pierde. Se trata de un juego determinista, y el objetivo es calcular que jugador pierde, sabiendo el orden de juego, y que como máximo pueden saltar su turno 3 veces.

J. El Juego de la piedra

Se trata de un problema simple, en el que simplemente hay que tener en cuenta varios aspectos:

- No es posible simular el juego: debido a que el número de piedras y de montones es elevado: $1 \leq N \leq 2^{30}$, $1 \leq M \leq 2^{30}$.
- Cuidado con el *overflow* al multiplicar: $2^{30} * 2^{30} > 2^{32} (MAXINT)$.
- Cuidado al restar -1 después del módulo, RTE en Java, WA en C, AC en Python.
- Se puede ignorar el aspecto de saltar turnos, dado que si un jugador va a perder, y pasa su turno, el resto de jugadores van a saltar su turno también para no perder. Es decir, el jugador que primero tiene que saltar su turno, es el que pierde.

J. El Juego de la piedra

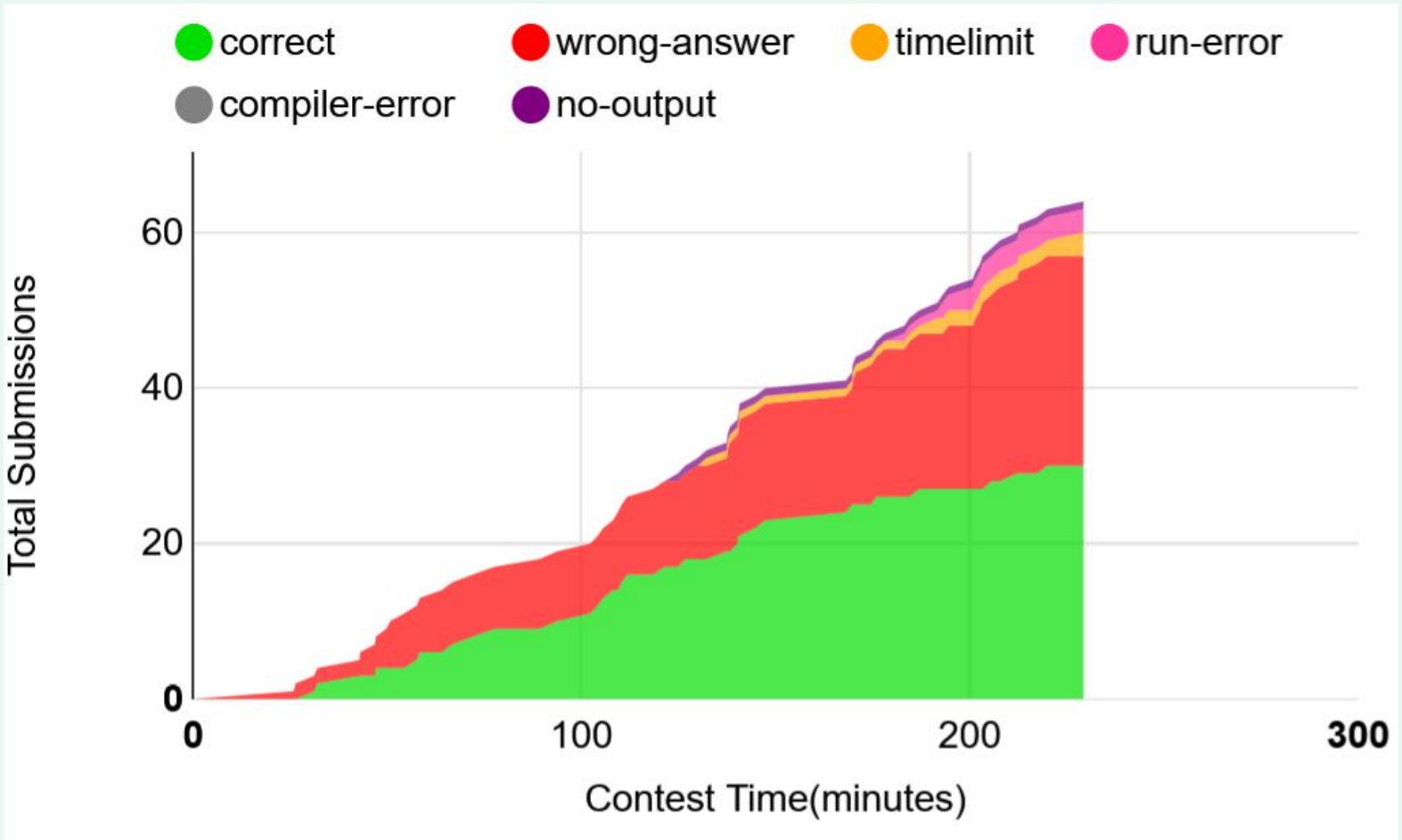
Tomando en cuenta lo anterior, podemos calcular el resultado de la siguiente manera:

```
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long n = sc.nextInt(), m = sc.nextInt();
        String[] names = new String[sc.nextInt()];
        for (int i = 0; i < names.length; i++) {
            names[i] = sc.next();
        }

        int target = (int)((n * m - 1) % names.length);
        System.out.println(names[target]);
    }
```

I. Bajo Presion

Envíos	Válidos	% éxito
59	28	47.45 %

I. Bajo Presion



I. Bajo Presion

En este ejercicio nos cuentan que un grupo de personas han sido secuestradas en Israel por una célula radical.

Conocemos:

- El número de secuestradores dentro de la célula ($0 \leq R_i \leq R$).
- El nivel de estrés que cada uno de los radicales son capaces de soportar ($2 \leq E_{R_i} \leq 20000$).

  Saben que cuando el estrés de un secuestrador es igual o superior a dicho nivel, . . .

Además, sabemos que a dichos radicales le van a asignar un número determinado de tareas. Y en la entrada nos dan:

- El nivel de estrés que cada tarea le va a aportar a cada radical.
- A qué radical se le asigna cada tarea.

El programa debe indicar qué radicales son los que no van a soportar la presión, o `Tenemos un problema` en el caso de que no haya ningún radical que llegue a ese nivel de presión.

I. Bajo Presion

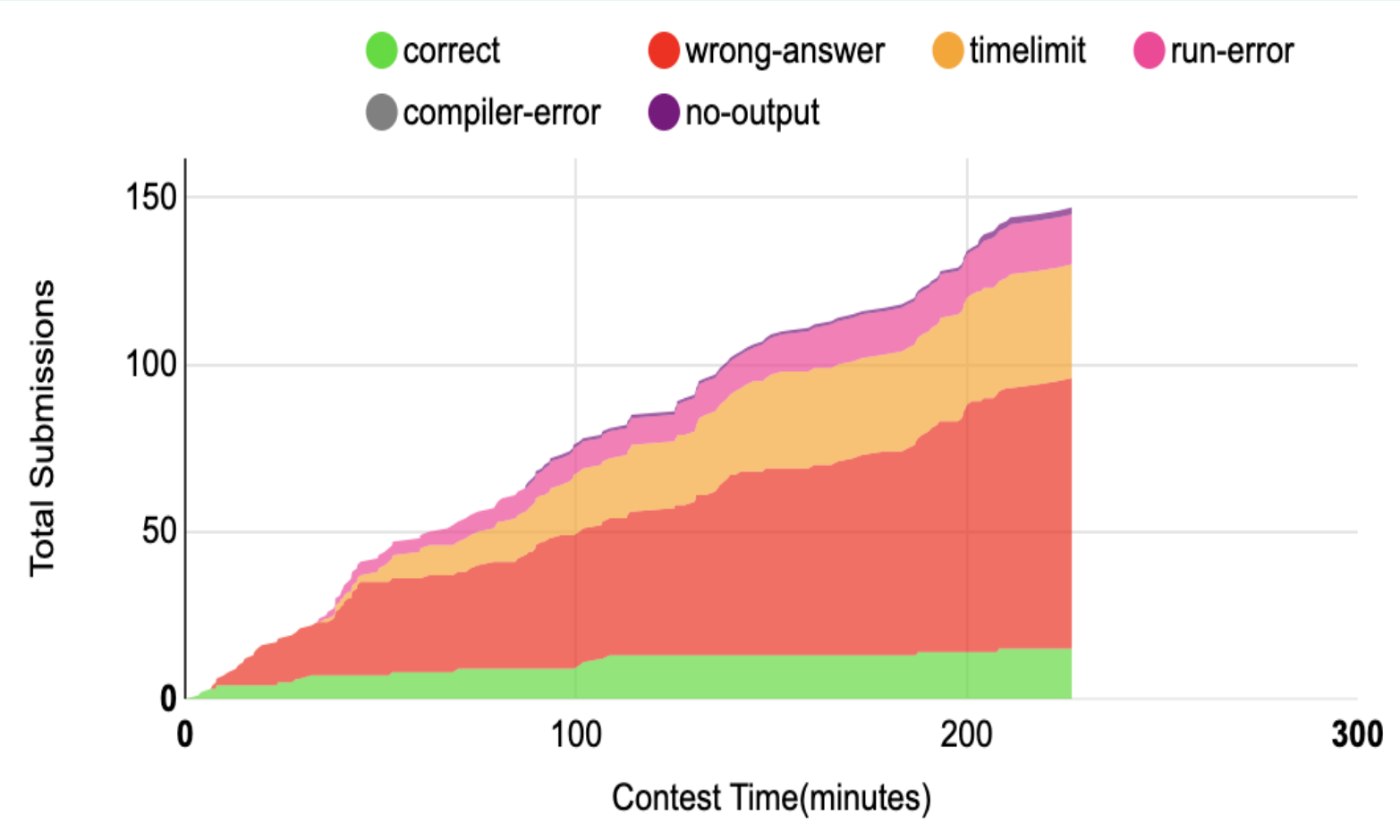
El enfoque más sencillo es el siguiente:

1. Crear un array de R elementos e inicializarlo a 0 (tensión actual).
2. Tenemos otro array con el nivel de estrés de cada una de las tareas.
3. Un tercer array con los niveles máximo que permite soportar cada radical.
4. Por cada tarea que se asigna:
   - Sumar al radical al que se le asigna la tarea el nivel de estrés de dicha tarea.
   - Si la tensión de dicho radical es mayor o igual que la tensión límite:
     - Imprimir por pantalla su nombre `Radical X`
     - En el array de *tesión actual* darle el valor de -1 para saber que dicho radical ya ha sido identificado.

G. El hacker al que le sobraba el dinero

Envíos	Válidos	% éxito
143	15	10.49 %

G. El hacker al que le sobraba el dinero



G. El hacker al que le sobraba el dinero

El problema consistía en encontrar la subsecuencia máxima dentro de la secuencia de transacciones.

Más allá de la complejidad del propio problema, si atendemos a los límites observamos que los valores de las transacciones pueden ser muy altos, por lo que había que tener cuidado con los tipos de datos a utilizar (ejem, Python).

El problema puede abordarse:

- Usando programación dinámica
- Usando divide y vencerás

G. El hacker al que le sobraba el dinero

Fallos comunes que hemos visto...

- No usar long long (el peor caso, la suma de todos los elementos positivos da overflow)
- No contemplar un caso con todos los valores negativos, cuyo resultado sería 0.

G. El hacker al que le sobraba el dinero

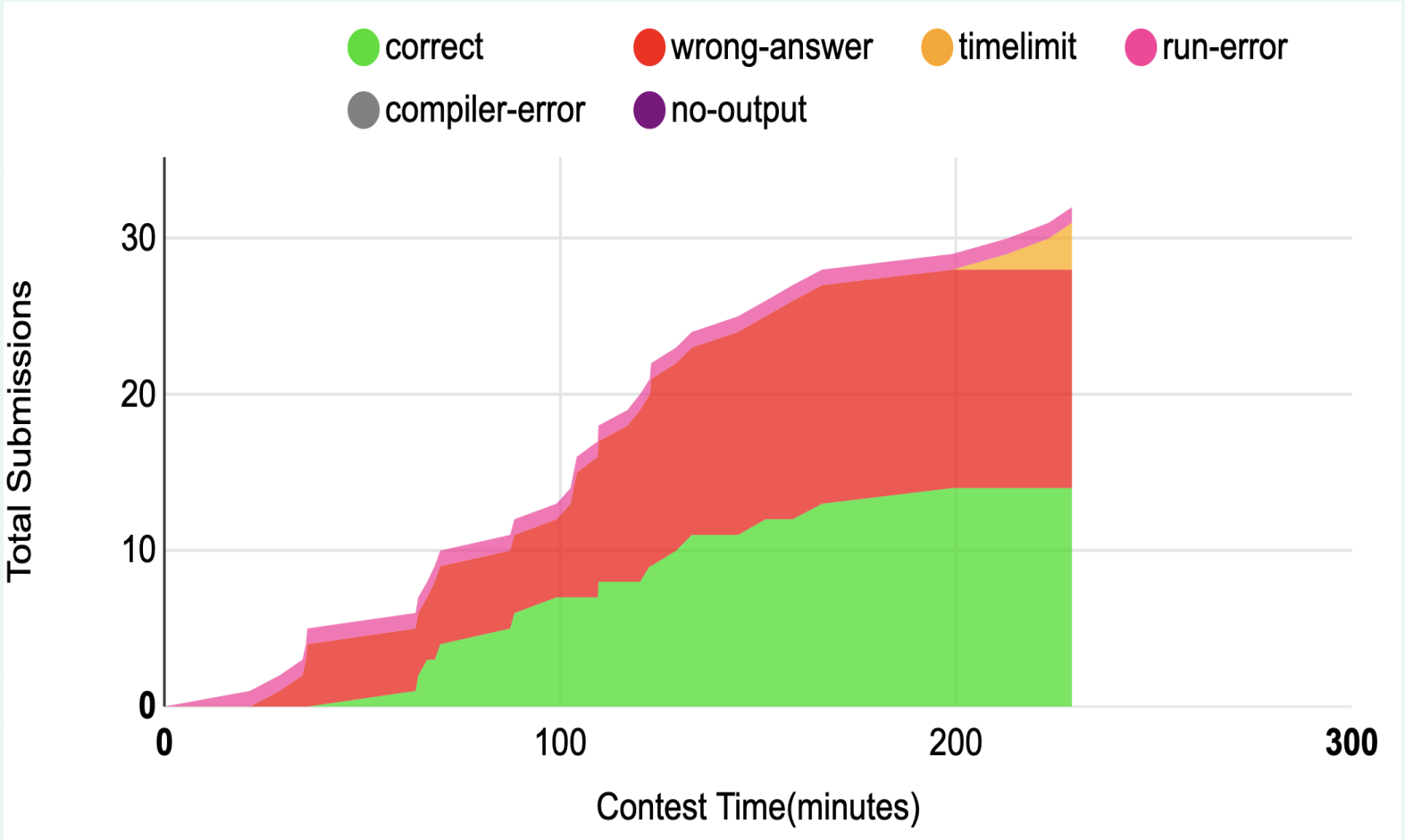
Mantenemos una lista donde cada valor de la posición i es la suma máxima encontrada hasta dicha posición. Luego devolvemos el mayor valor de esa lista:

```
current_sum = [0 for _ in transactions]
max_sum = 0
for i, transaction in enumerate(transactions):
    if i == 0:
        current_sum[i] = max(current_sum[i], transactions[i])
    else:
        current_sum[i] = max(current_sum[i], current_sum[i - 1] + trans

    max_sum = max(max_sum, current_sum[i])
```

A. Ayudando a los camareros

Envíos	Válidos	% éxito
29	14	48.27 %

A. Ayudando a los camareros



A. Ayudando a los camareros

Dado el número de empleados que han ido a comer al restaurante y el precio de la comida por persona, ¿cuál es el número mínimo de cheques con el que podemos pagar?

Ten en cuenta que necesitamos pagar la cantidad exacta de dinero que cuesta la comida. Afortunadamente, el precio de la comida es siempre un entero y siempre tendremos disponibles cheques de valor de 1 €, por lo que siempre será posible pagar la comida, además, de todos los cheques tenemos una cantidad infinita.

A. Ayudando a los camareros

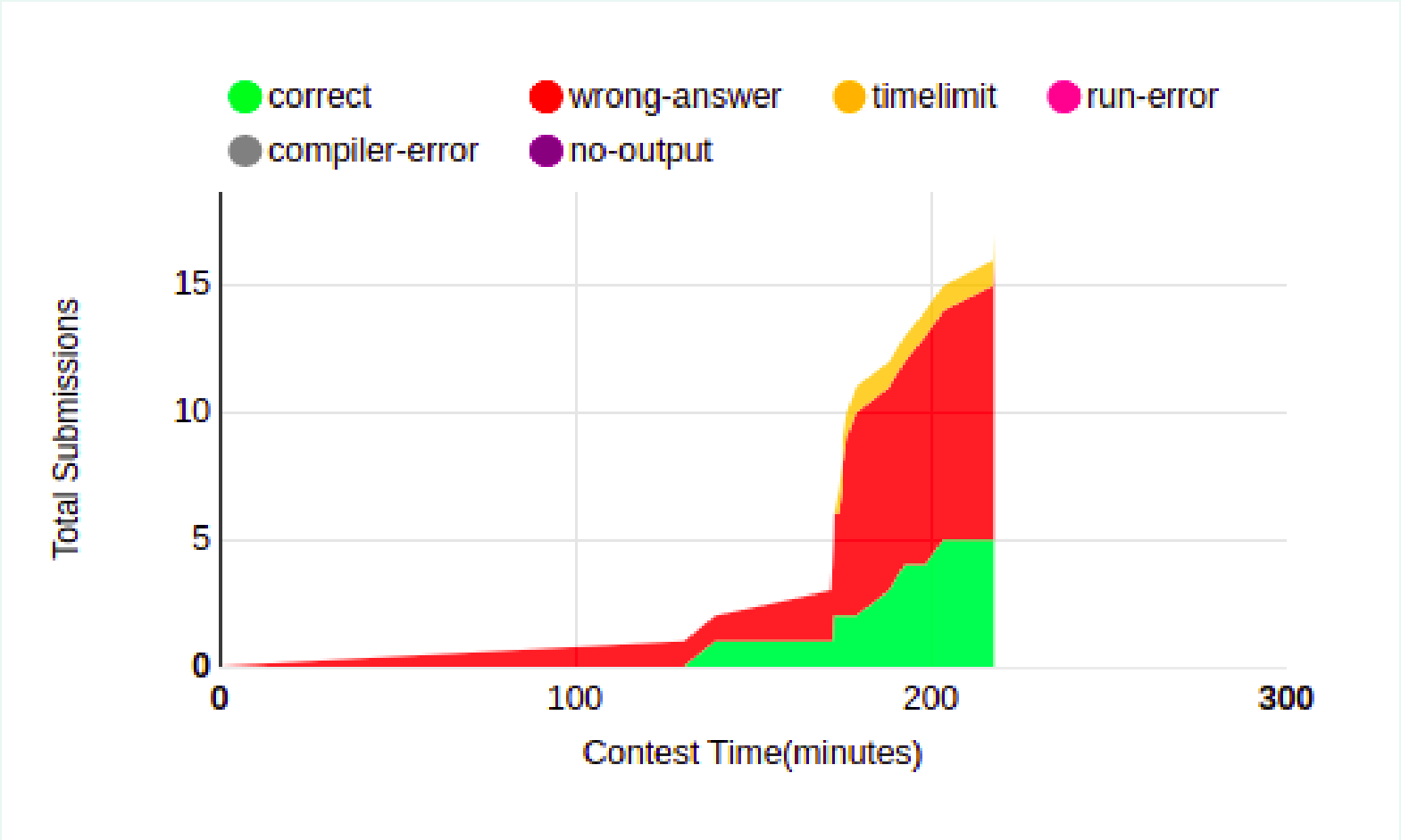
Tomando en cuenta lo anterior, podemos calcular el resultado de la siguiente manera:

```
def minCheques(cheques, n_valores, coste):
    tabla = [0 for _ in range(coste + 1)]
    tabla[0] = 0
    for i in range(1, coste + 1):
        tabla[i] = sys.maxsize

    # Cheques minimos para todos los costes
    for i in range(1, coste + 1):

        # Cheques menores de i
        for j in range(n_valores):
            if cheques[j] <= i:
                sub_res = tabla[i - cheques[j]]
                if sub_res != sys.maxsize and sub_res + 1 < tabla[i]:
                    tabla[i] = sub_res + 1
    if tabla[coste] == sys.maxsize:
        return -1
    return tabla[coste]
```

L. Primos traviesos

Envíos	Válidos	% éxito
15	5	33.3 %

L. Primos traviesos



L. Primos traviesos

Nos pide imprimir el mínimo número primo tal que:
$\sum_{i=0}^{n} min(P_j, N_i) >= k$
Teniendo en cuenta que:

- La casa más alta iba a tener 1,000,000 de unidades
- Habrían como mucho 100,000 casas
- K podía llegar a ser 1,000,000,000

L. Primos traviesos

Posibles ideas:

- Un bucle desde 0 hasta N y luego otro de 0 hasta K probando todas las posibles alturas. O(NK). TLE
- Un bucle desde 0 hasta N con otro anidado desde 0 hasta el máximo de todas las alturas de N_i. O(NC). TLE
- Manejar sumatorias de todas las casas con todas las alturas $10^5 * 10^6 = 10^{11}$. Si usabas un int para esto. WA

L. Primos traviesos

Mejor idea:

- A partir de cierto número primo en adelante, siempre superaremos K. Entonces consideramos una función f que nos diga si superamos el umbral o no. Tal que f(i) = 0 si no se supera el umbral con el i-ésimo número primo y f(i) = 1 en el caso contrario.
- Precalculamos todos los números primos desde 2 hasta 1,000,000
- Calculamos hasta 1,000,000 porque la máxima altura del edificio siempre será el máximo número primo que imprimiremos.
- Si aún con esto no fuere posible. Imprimimos *IMPOSIBLE*

L. Primos traviesos

Mejor idea:

- Aún así, con bucles anidados esto da TLE
- Podemos utilizar búsqueda binaria para marginar partes del problema que no nos sirvan
- para un i cualquiera, si f(i) = 1, significa que siempre será 1 para cualquier j tal que j>i
- para un i cualquiera, si f(i) = 0, significa que siempre será 0 para cualquier j tal que j<i
- Podemos iterar este enfoque usando *búsqueda binaria*. Convirtiendo el problema en O(NLgM). M siendo el tamaño de los números primos conseguidos desde 0 hasta 1M

L. Primos traviesos

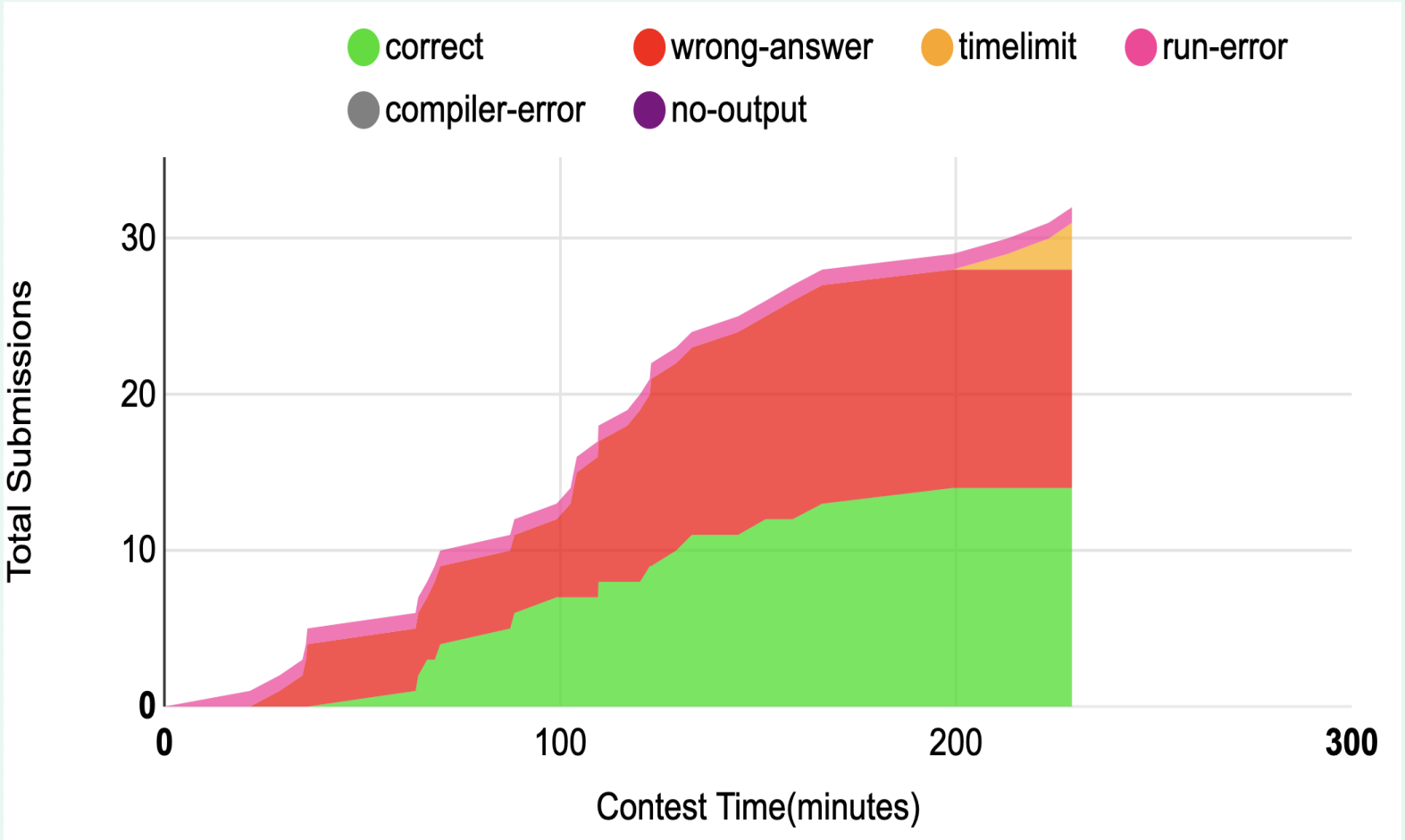
Otra forma

- Ordenar las casas de menor a mayor
- Hacer un bucle desde 0 hasta la altura máxima de las casas
- Si la altura excede a la casa en la posición k, sumar 1 a k (pasar de casa)
- Sumar al area total el número de casas menos k por cada iteración
- Si no hay nada más que sumar, guardar la altura A conseguida
- Finalmente, a partir de ese A buscar el siguiente primo tal que $P_j > A$ e imprimirlo
- Esta solución es O(MLgN) siendo M la altura máxima y N la cantidad de casas.

D. El juego del pañuelo

Envíos	Válidos	% éxito
26	4	15.38 %

D. El juego del pañuelo



D. El juego del pañuelo

Dado una serie de equipos, determinar sabiendo que jugador gana a que oponente/s la mejor distribucción para obtener el mejor resultado.
Problemas encontrados...

- Representar mal el grafo -> WA
- Usar programación dinámica -> TLE

D. El juego del pañuelo

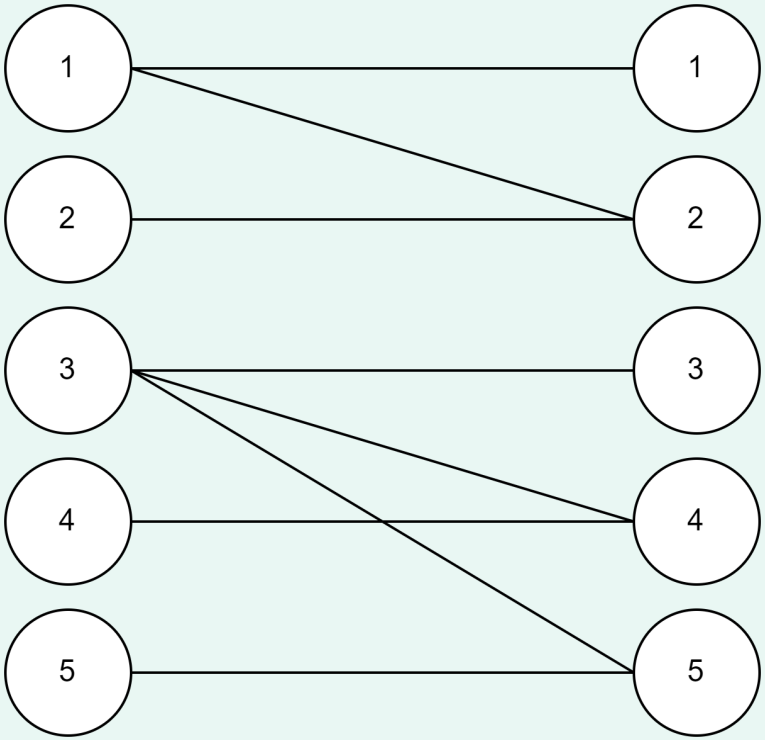


Figura: Representación de los dos equipos en un grafo bipartito

D. El juego del pañuelo

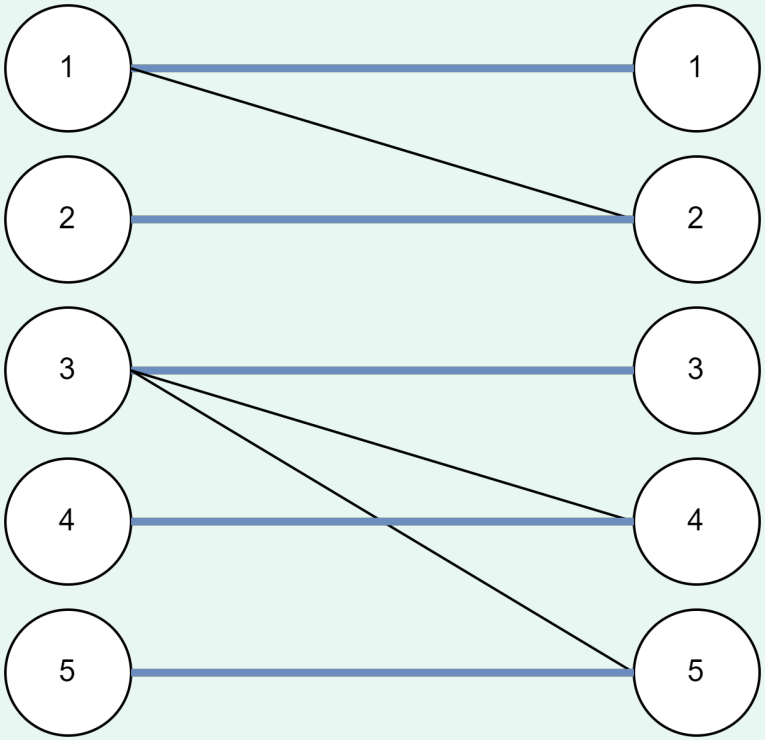


Figura: Selección de oponentes según rivales a los que se les gana

D. El juego del pañuelo

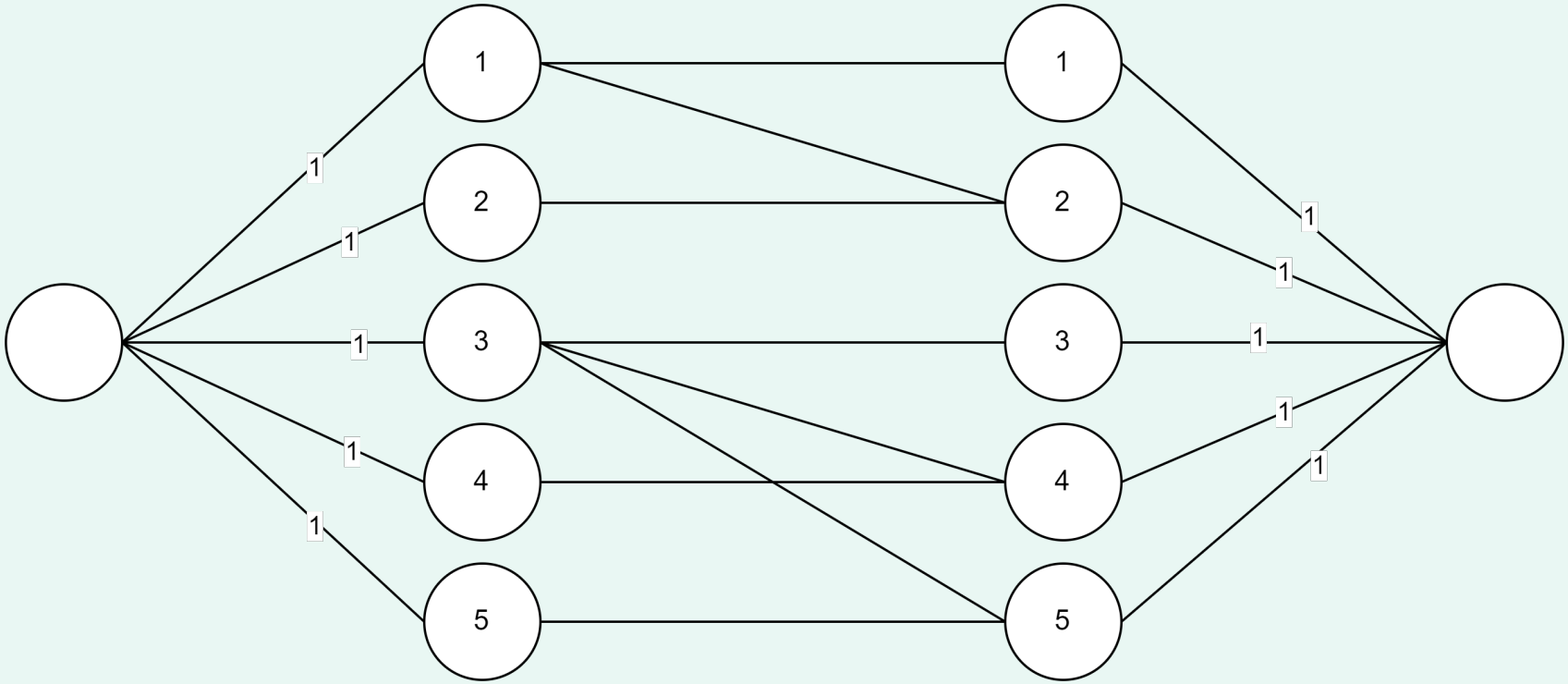


Figura: Representación en grafo de máximo flujo con peso

D. El juego del pañuelo

¿Cómo resolverlo?

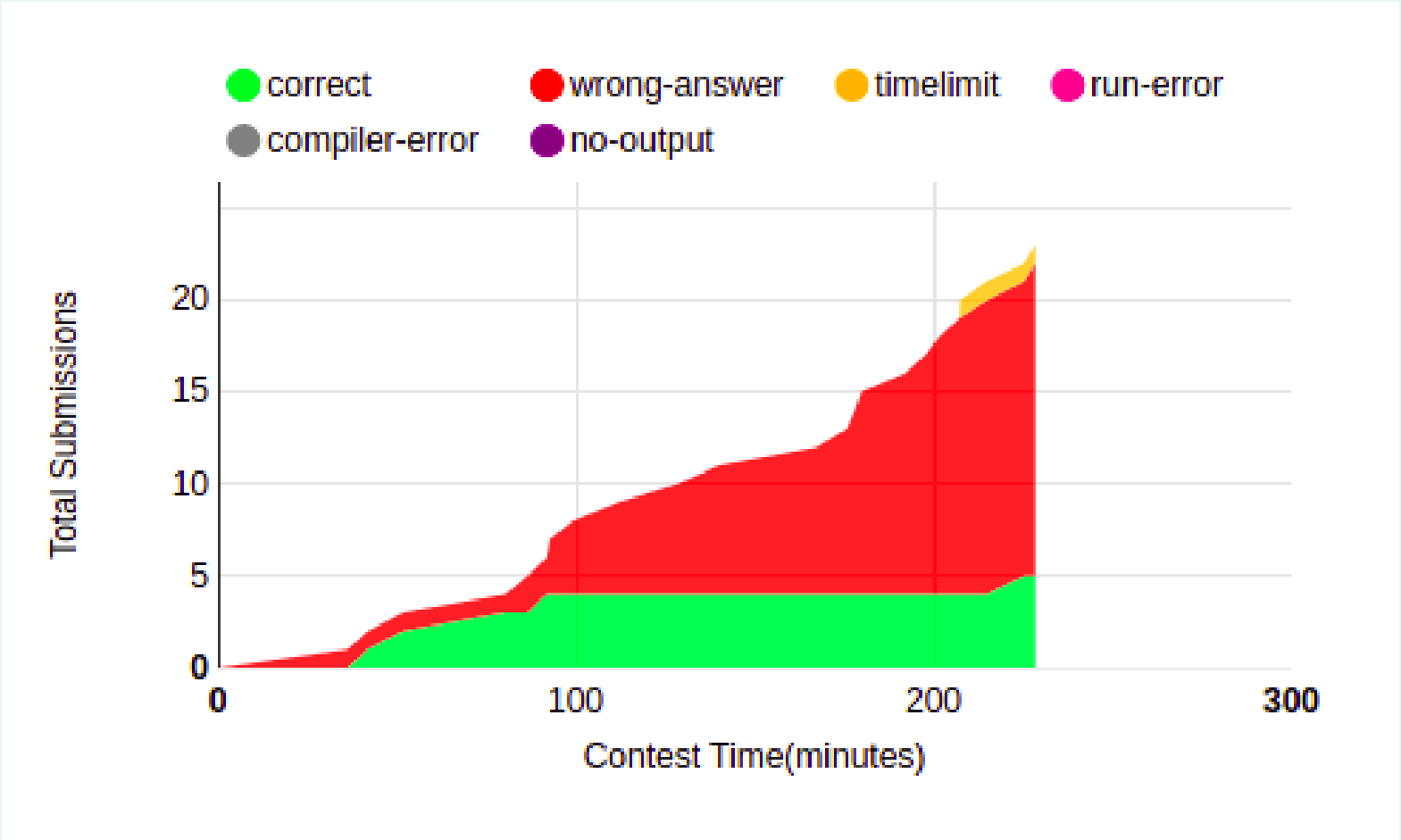
- Hopcroft Karp $O(\sqrt{V} \cdot E)$ *
- Dinic $O(V^2 E)$ *
- Push Relabel $O(V^3)$ *
- Max Bipartite Maching $O(VE^2)$ *
- Ford Fulkerson $O(VE^2)$ *
- Kuhn's Algorithm $O(VE)$ *

*https://isaaclo97.github.io/resources/codes/Hopcroft-Karp.html
https://www.geeksforgeeks.org/
hopcroft-karp-algorithm-for-maximum-matching-set-1-introduction/
*https://isaaclo97.github.io/resources/codes/Dinic.html
https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/
*https://isaaclo97.github.io/resources/codes/PushRelabel.html
https://www.geeksforgeeks.org/
push-relabel-algorithm-set-1-introduction-and-illustration/
*https://isaaclo97.github.io/resources/codes/MaxBipartiteMaching.html
*https://www.geeksforgeeks.org/maximum-bipartite-matching/
*https://cp-algorithms.com/graph/kuhn_maximum_bipartite_matching.html

H. Guerra Intergaláctica

Envíos	Válidos	% éxito
20	4	20.00 %

H. Guerra Intergaláctica

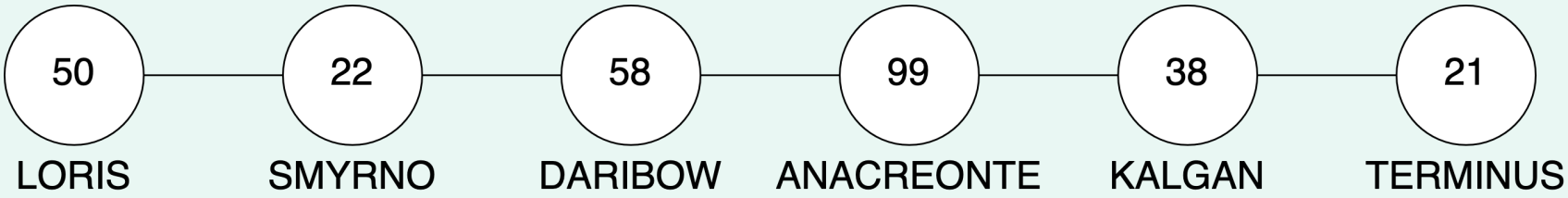


H. Guerra Intergaláctica

En este problema se nos pedía calcular el coste que tendría perder los planetas estratégicos en una guerra intergaláctica. Modelando el mapa de la galaxia como un grafo, podemos ver que la pregunta puede reformularse como sigue: Dado un grafo $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de aristas, encuentra los vértices que, si son eliminados del grafo, lo desconectan generando diferentes componentes conexas. Dicho de otro modo, debemos encontrar los nodos críticos o *puntos de articulación* del grafo.

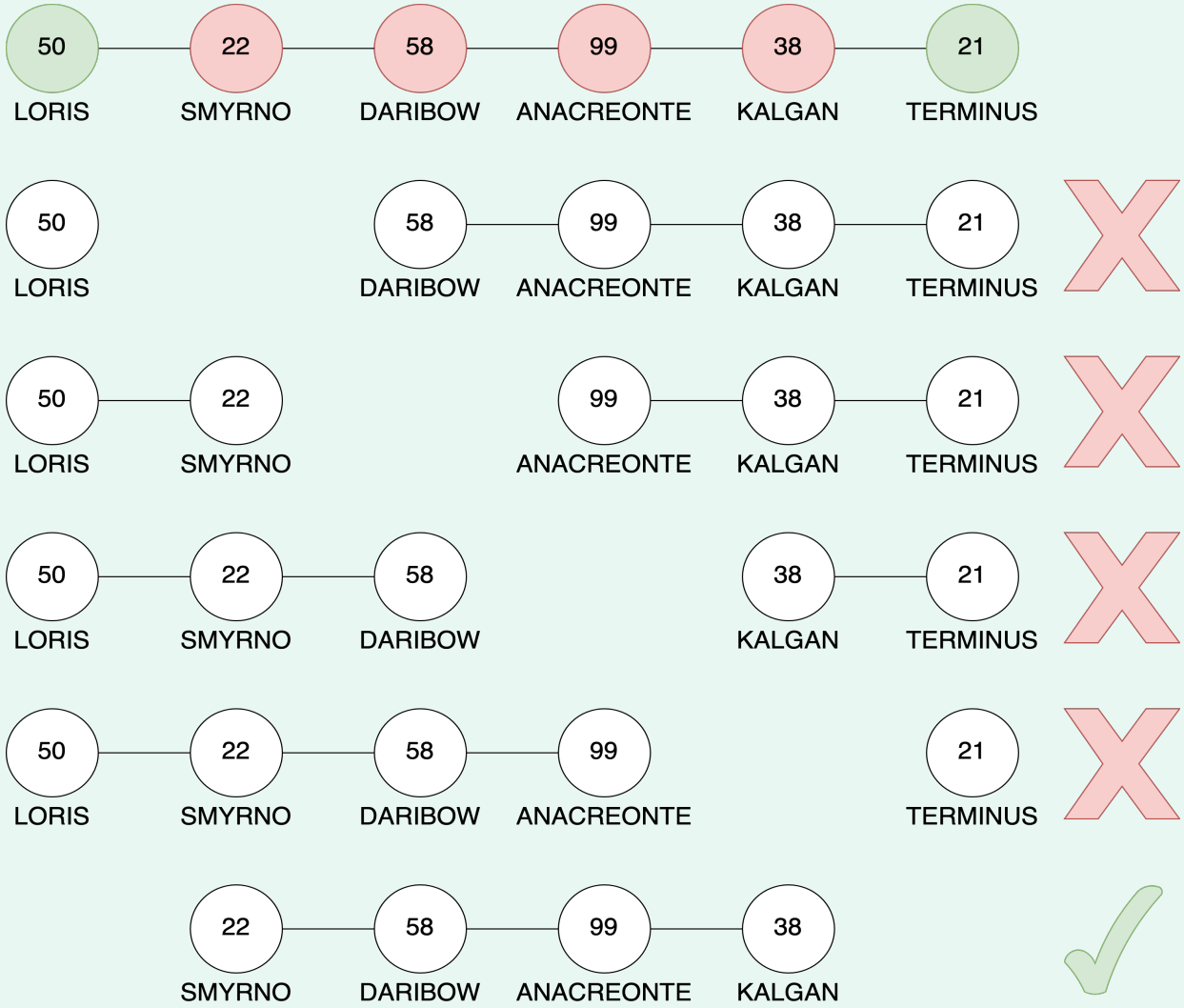
H. Guerra Intergaláctica

En el ejemplo de la entrada, tenemos el siguiente grafo:



H. Guerra Intergaláctica

Podemos ver fácilmente como si perdemos SMYRNO, DARIBOW, ANACREONTE o KALGAN, el grafo queda dividido en diferentes componentes conexas. sin embargo, perder LORIS o TERMINUS no implica una desconexión de la red.



H. Guerra Intergaláctica

Para encontrar los puntos de articulación de un grafo, debemos hacer lo siguiente:

- Eliminar un nodo v del grafo
- Evaluar si el grafo sigue conformando una sola componente conexa o está dividido en varias (BFS o DFS)
- Añadir de nuevo v al grafo

Este proceso es demasiado complejo ($O(V * (V + E))$, por lo que esta aproximación se saldría de los límites de tiempo.

H. Guerra Intergaláctica

Una mejor aproximación basada en DFS sería la siguiente:

- Lanzamos DFS sobre el grafo, generando el árbol del recorrido en profundidad
- Consideramos que un vértice u es un punto de articulación si cumple una de estas dos condiciones:
  1. u es la raíz del árbol y tiene **al menos** dos hijos
  2. u no es la raíz del árbol y tiene un hijo v tal que ningún vértice del subárbol con raíz en v tiene una arista hacia ningún ancestro de u

El primer caso es trivial, pero el segundo no lo es tanto.

H. Guerra Intergaláctica

Debemos añadir diferentes estructuras de datos a nuestro recorrido DFS para mantener la información necesaria para encontrar los puntos de articulación (algoritmo de Tarjan):

- Una lista de padres *parent[]*, donde *parent[u]* guarda el padre de u en el árbol
- Una lista *disc[]* que almacena el momento en el que se descubren los nodos en el recorrido, donde *disc[u]* almacena el momento en el que se descubre u
- Una lista *low[]*, que almacena el vértice con el mínimo momento de descubrimiento que puede ser alcanzado desde el subárbol con raíz en u, es decir, *low[u] = min(low[u], low[v])*, para todo v adyacente a u no visitado*.

En resumen: si u no es raíz del árbol y el valor de *low[v]* de uno de sus hijos es mayor que el valor de *desc[u]*, entonces u es un punto de articulación, porque habrá una arista hacia un ancestro de u desde el subárbol con raíz en v

*si el nodo v ya ha sido visitado y no es el padre de u, entonces *low[u] = min(low[u], disc[u])*

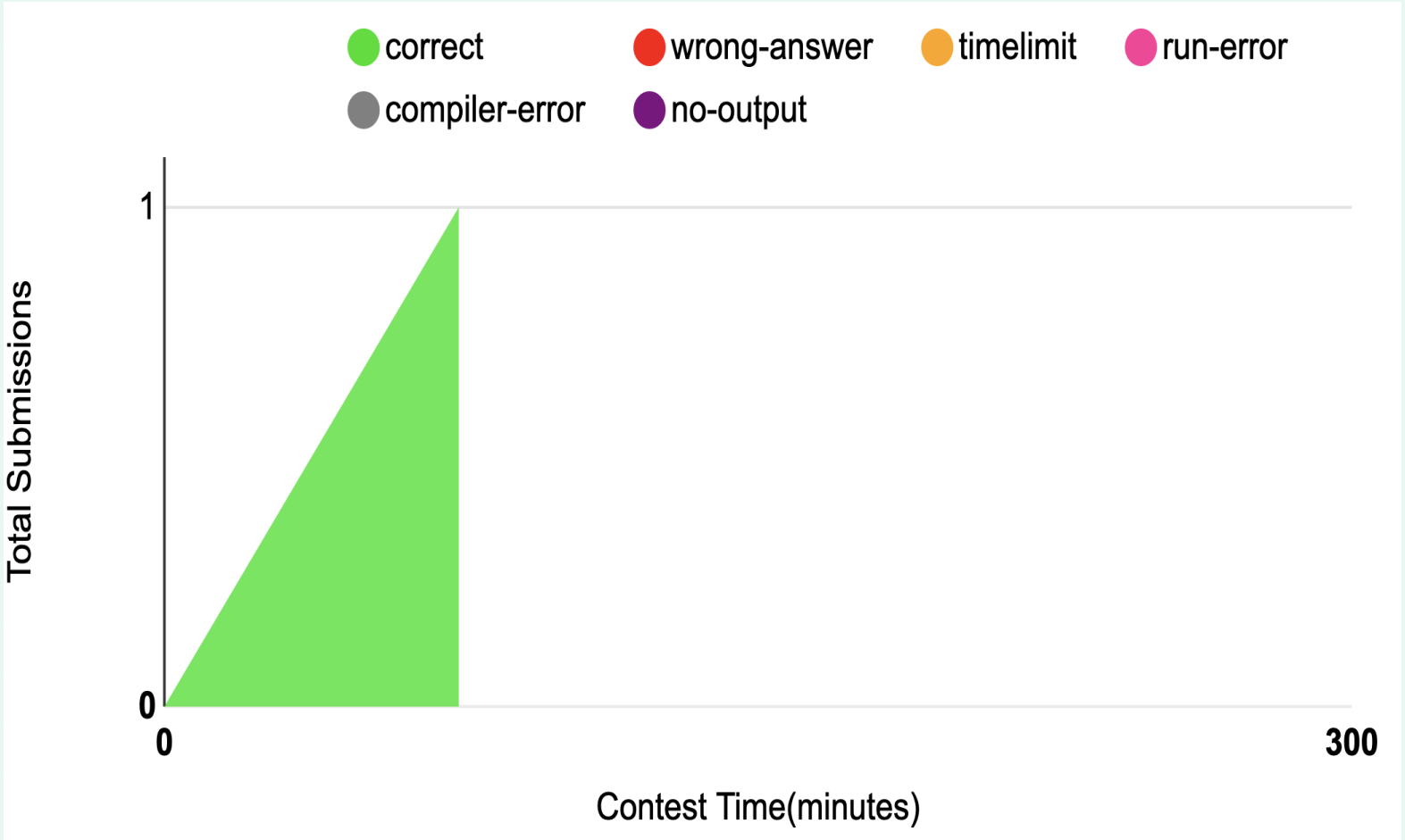
H. Guerra Intergaláctica

De esta forma, encontrando los puntos de articulación del grafo y sumando sus costes asociados, podemos encontrar el coste de perder todos los planetas estratégicos. Manteniendo el orden de entrada, podemos imprimir aquellos nodos que son puntos de articulación (para ello, ayuda un mapa que mantenga la relación nombre_de_la_ciudad $\rightarrow$ número_de_nodo).

B. ¡¡¡Explosiones cuadradas!!!

Envíos	Válidos	% éxito
1	1	100 %

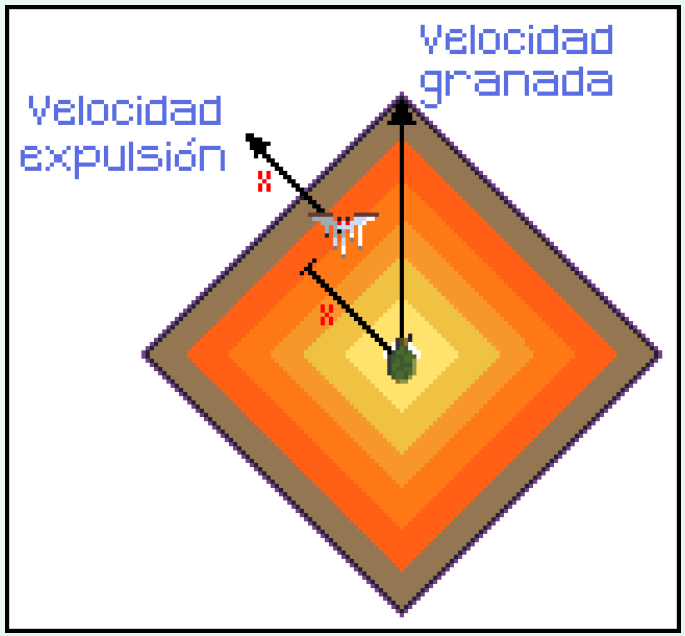
B. ¡¡¡Explosiones cuadradas!!!



B. ¡¡¡Explosiones cuadradas!!!

Este problema consistía en calcular la velocidad a la que sale despedido un enemigo tras la explosión de una granada (siguiendo unas reglas especiales). Para ello se proporciona la posición y velocidad direccional de la granada, así como la posición del enemigo.
Observaciones importantes:

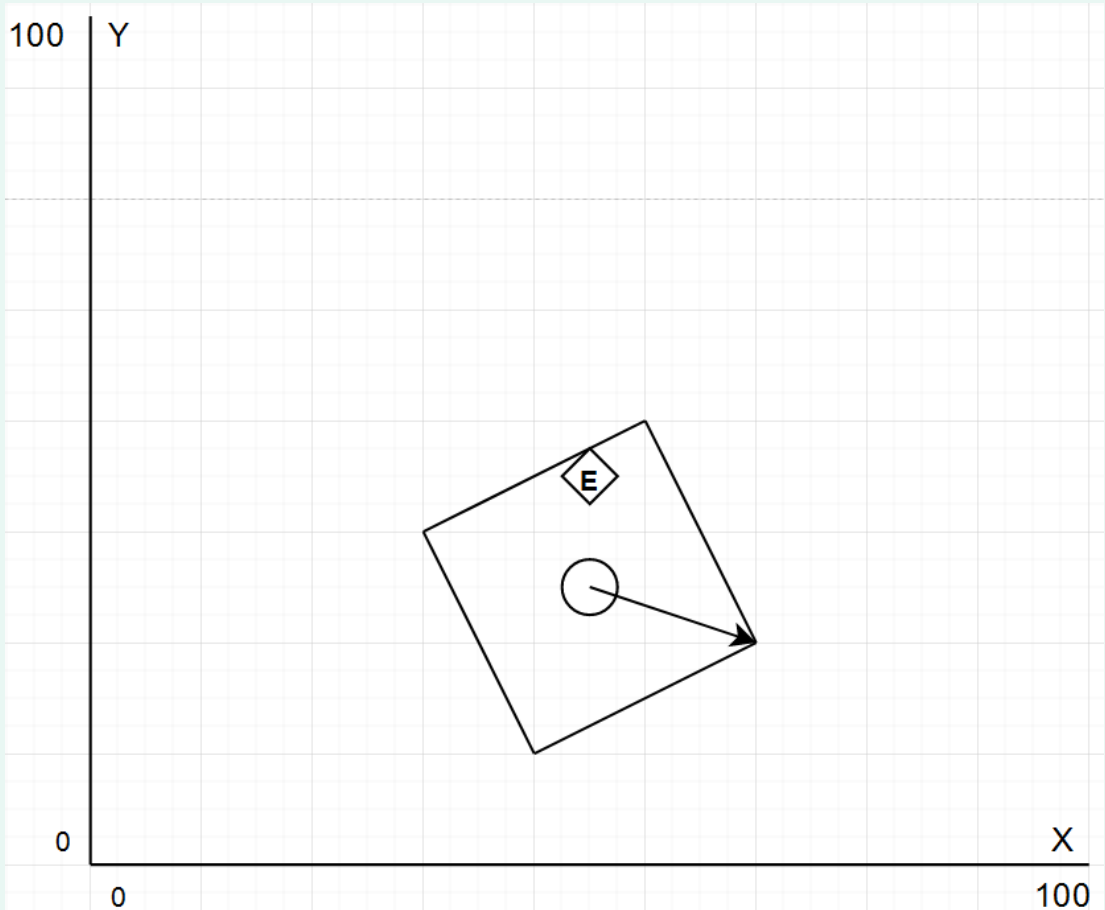
- La **explosión es cuadrada**, dependiendo del vector de velocidad de la granada.
- La velocidad del **empuje** a calcular de la granada será **hacia la arista más próxima** del cuadrado de explosión. Esta será igual a la proyección de la granada en esa dirección.
- **No** se tendrán en cuenta **entradas con soluciones múltiples** (ej: el enemigo se encuentra justo en una diagonal del cuadrado).



B. ¡¡¡Explosiones cuadradas!!!

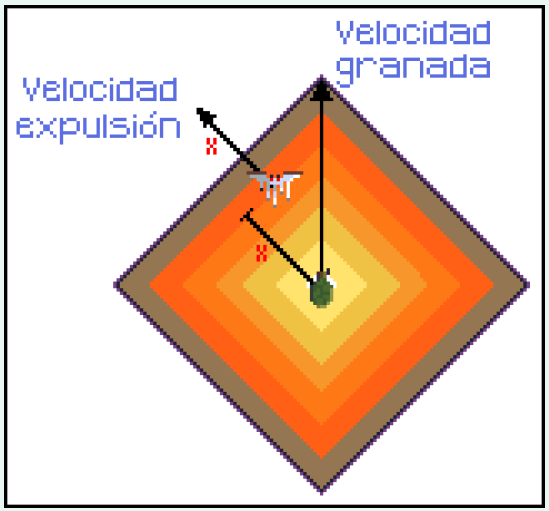
Tomando en cuenta lo anterior, las ideas principales son las de detectar si el enemigo está dentro del cuadrado formado por los vectores ortogonales que parten de la granada y son formadas por su velocidad.

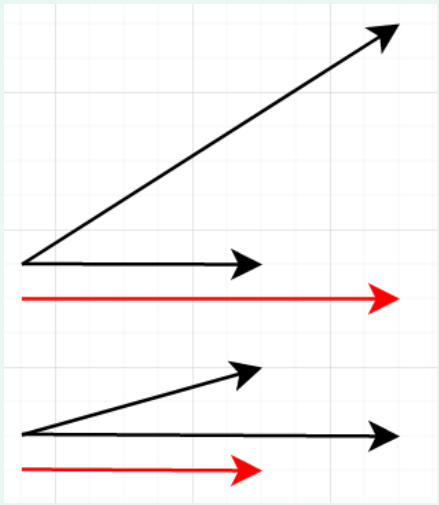
- Ya sea tratando de ver en qué sub-triángulo se encuentra el enemigo.
- Ya sea calculando en qué lado de la reta de la arista se encuentra el enemigo.



B. ¡¡¡Explosiones cuadradas!!!

Lo que queda claro es que, tras saber **si está dentro el enemigo, tendremos que calcular la proyección** de su posición hacia la arista más cercana.
Recordamos que la fórmula de la proyección de un vector es la siguiente (dando como resultado el vector rojo que vemos en la imagen).

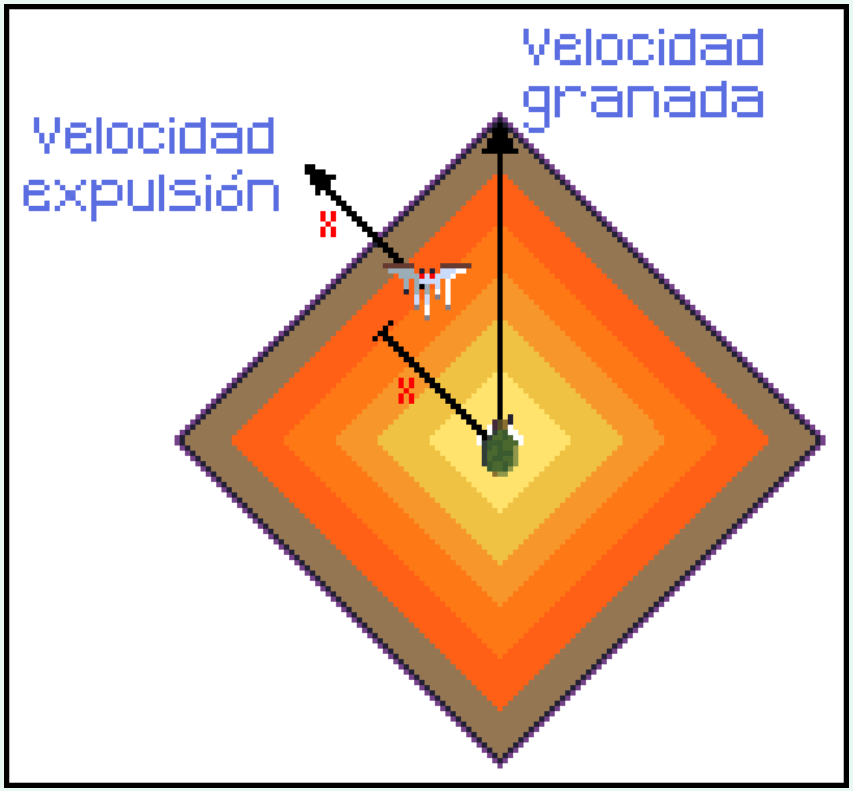


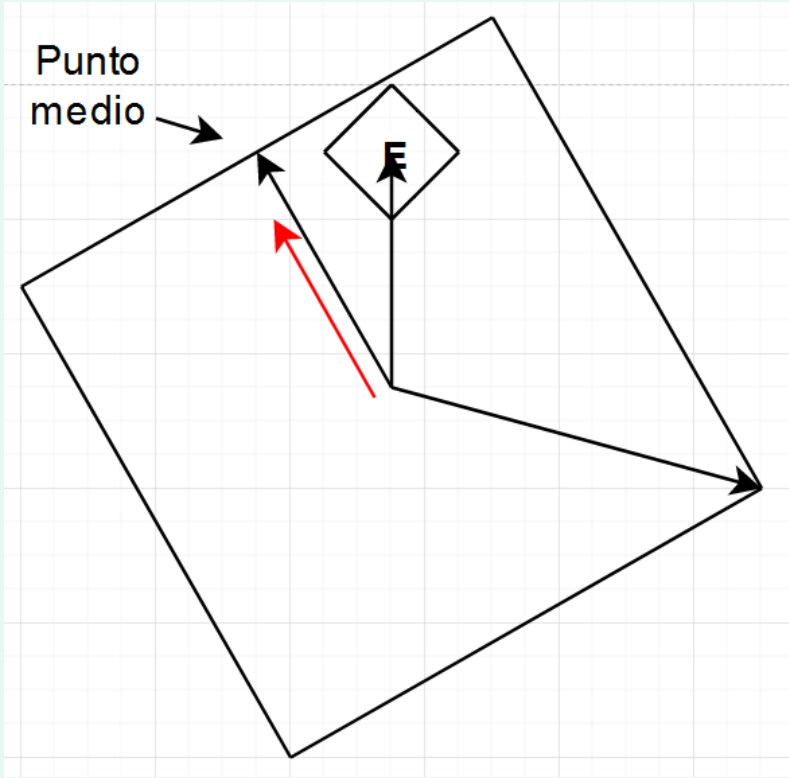


$$\vec{proyec} = (\vec{a} \cdot \vec{b} / \vec{b} \cdot \vec{b}) * \vec{b}$$
$$\vec{proyec} = (\vec{a} \cdot \vec{b} / |\vec{b}|^2) * \vec{b}$$

B. ¡¡¡Explosiones cuadradas!!!

Sabiendo esto, podemos calcular la **proyección sobre el punto medio de cada arista** como se ve a continuación.



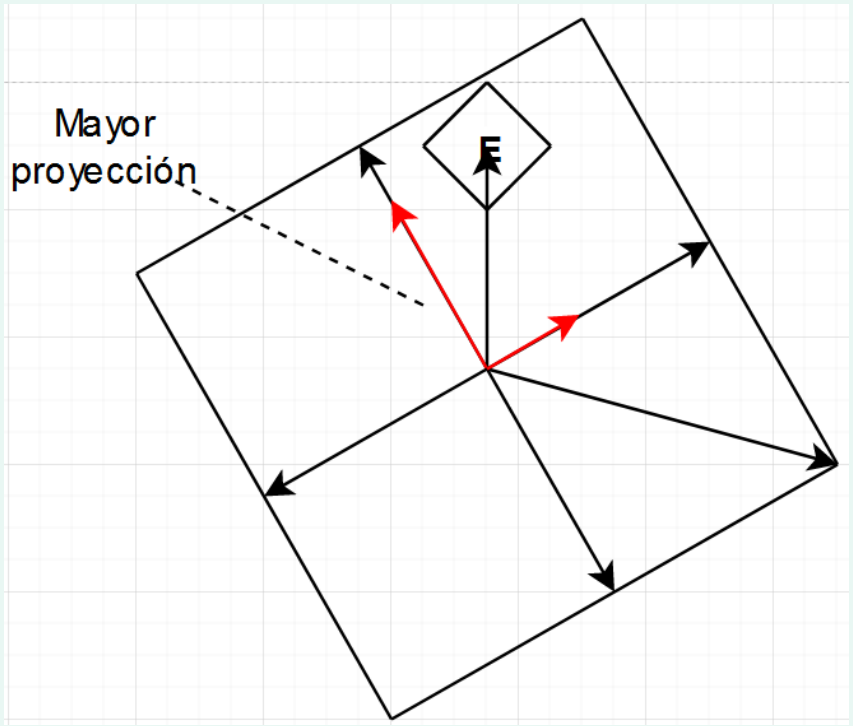


En este caso $\vec{a}$ será el vector granada-enemigo y $\vec{b}$ será el vector granada-arista.

B. ¡¡¡Explosiones cuadradas!!!

Pero para saber cuál es la buena, es necesario hallar las proyecciones con todos los puntos medios (o al menos dos).

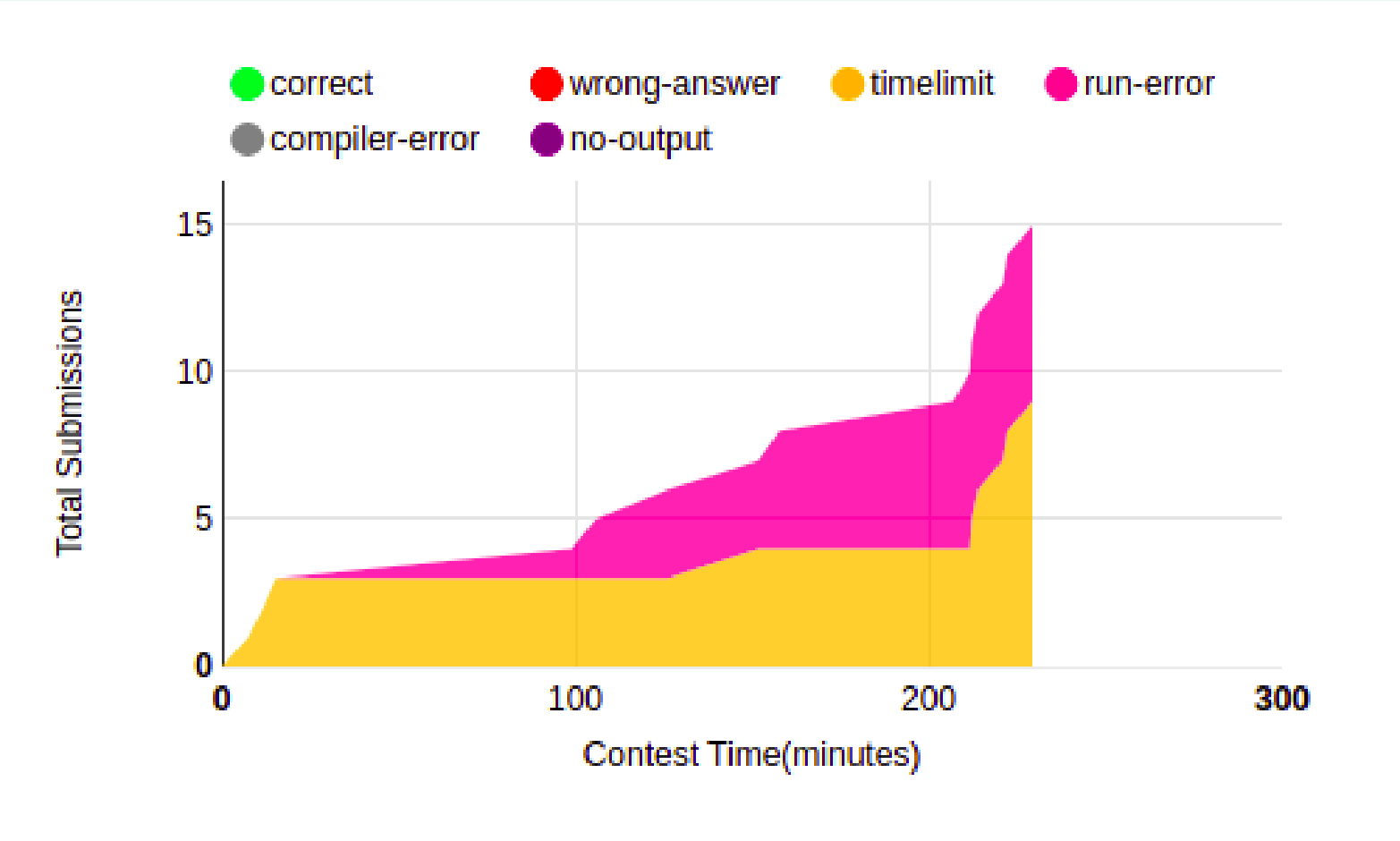
- **El enemigo estará dentro del cuadrado sí**: el módulo de la mayor proyección de las cuatro es menor que el módulo de la granada a cualquiera de los centros de arista.
- La velocidad de repulsión **será**: la **proyección con el mayor módulo**.
- Si queremos saber cuál de los **lados** es el **ganador**, ya que varias proyecciones siempre se solapan, basta con evaluar: $\vec{a} \cdot \vec{b} > 0$



M. Jakub y los Cuadrados

Envíos	Válidos	% éxito
9	0	0 %

M. Jakub y los Cuadrados



M. Jakub y los Cuadrados

Problema: Contar los numeros *square-free* menores o iguales a N Idea de la solucion: Contar los numeros que **no** son *square-free* y aplicar el principio de inclusion-exclusion.

Para calcular la cantidad de numeros *square-free* hasta N existen varias maneras. Se puede simplemente iterar hasta N y validar si el numero actual es *square-free* o no (descomponiendolo en factores primos). Este metodo no entra en tiempo ya que el N es muy grande.

Otra manera un poco mas astuta seria contar todos los numeros que son divisibles por $p_i^x * p_j^y ... \forall p \in P$. Donde P es el conjunto de numeros primos menores o iguales a la raiz de N. Haciendo uso de la tecnica de inclusion-exclusion. Sin embargo este metodo si se realiza de manera recursiva dara RTE ya que se agotaran los recursos de la pila. En caso de hacerse iterativamente, el TLE estara asegurado tambien.

La solucion esperada para este problema se basa en calcular la **Funcion de Mobius** ($\mu(x)$ hasta $\sqrt{N}$ y luego iterar hasta $\sqrt{(N)}$ y acumular en una variable lo siguiente:

$$ans = ans + \mu(x) * -1 * \frac{n}{x^2}$$

Esto nos dara el numero total de numeros que **no** son *square-free*. Luego solo queda restar esa cantidad del n.

M. Jakub y los Cuadrados

```
// global scope
#define pb push_back
#define forn(i, n) for(int i = 0; i < (int)(n); ++i)
#define for1(i, n) for(int i = 1; i < (int)(n); ++i)
typedef long long ll;
void fastIO() {
    cin.sync_with_stdio(false);
    cin.tie(0);
}
const int MAXN = 1e7+5;
ll n, m; // sizes
bool isPrime[MAXN];
int mu[MAXN];
vector<ll> primes;
```

M. Jakub y los Cuadrados

```
void sieve() {
    memset(isPrime, 1, sizeof(isPrime));
    memset(mu, 0, sizeof(mu));
    mu[1] = 1;
    for(int x=2;x<MAXN;x++) {
        if(isPrime[x]) {
            primes.pb(x);
            mu[x] = -1;
        }
        int sz = primes.size()
        for(int j=0; j<sz && primes[j]*x<MAXN ;j++){
         isPrime[x*primes[j]] = 0;
         if (x%primes[j] == 0){
        break;
    }
            mu[x*primes[j]] = (-1)*mu[x];
        }
}
}
```

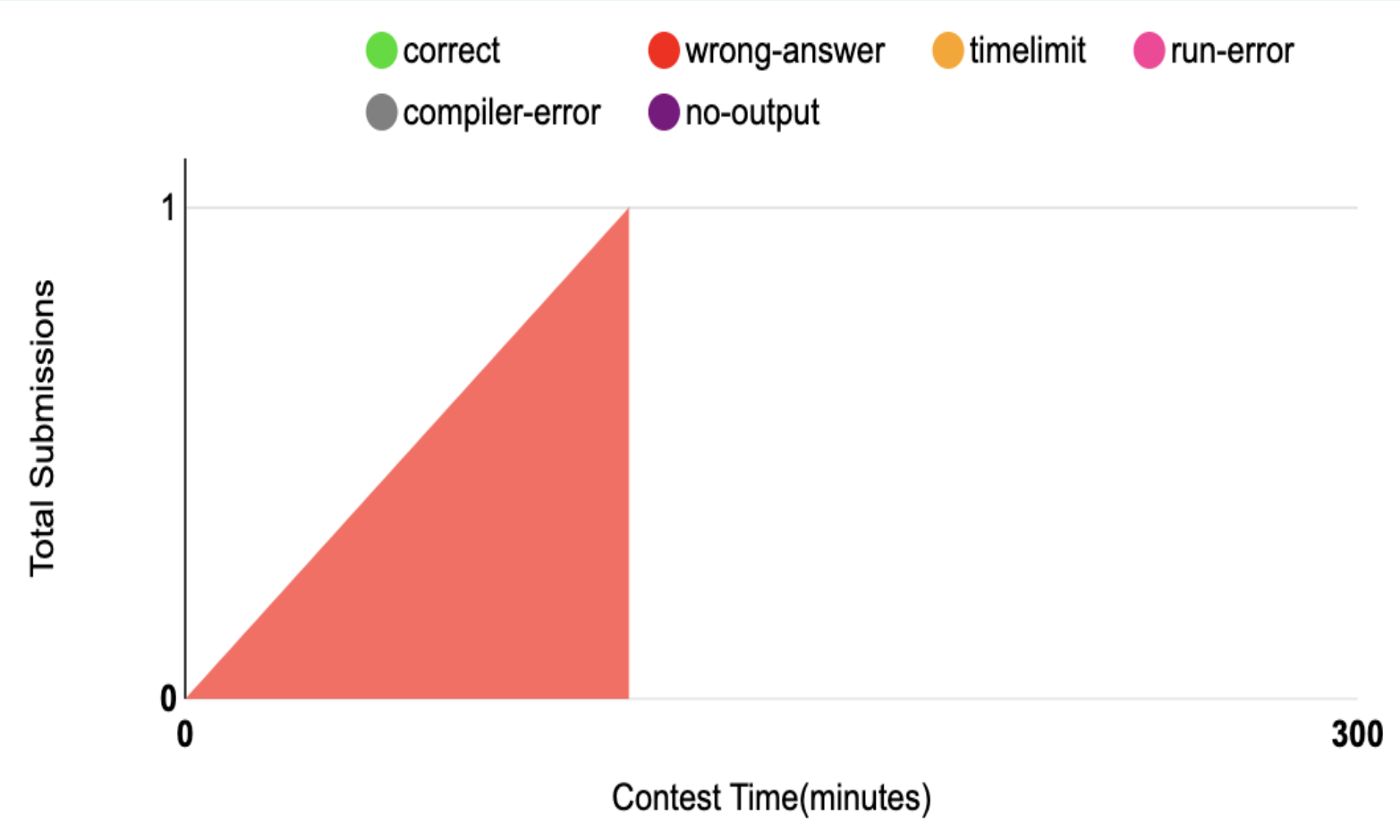
M. Jakub y los Cuadrados

```
void solve() {
    cin >> n;
    ll ans = 0;
    m = primes.size();
    for(ll i = 2; i*i <= n; i++) {
        if (mu[i])
            ans += (mu[i]*-1) * (n/(i*i));
    }

    cout << n - ans << endl;
}

int main() {
    fastIO();
    sieve();
    int t;
    cin >> t;
    while(t--) {
        solve();
    }
    return 0;
```

F. La conquista del espacio

Envíos	Válidos	% éxito
1	0	0 %

F. La conquista del espacio



F. La conquista del espacio

La solución de este problema tiene tres puntos claves a valorar:

- Representación del mapa estelar mediante un grafo ponderado en una matriz de adyacencia.
- Analizar todas las posibles soluciones mediante un algoritmo de *backtracking*.
- Realizar la búquesda de soluciones empezando por el planeta 1 para obtener la mejor ruta lexícográficamente.

F. La conquista del espacio

Para determinar la secuencia de planetas que se deben visitar para lograr visitar el mayor número de planetas diferentes con el menor número de viajes interplanetarios, debemos ser capaces de comparar dos rutas:

```
function es_mejor (ruta_a, rupa_b):
  return (num_panetas(ruta_a) > num_planetas(ruta_b)) OR
    (num_panetas(ruta_a) == num_planetas(ruta_b)
      AND len(ruta_a < len(ruta_b))
```

F. La conquista del espacio

Tomando en cuenta lo anterior, podemos calcular el resultado de la siguiente manera:

```
function explora (he, mapa, ruta, fuel):
  planeta = pop(ruta)
  fuel = max(2000, fuel + he[planeta] * 0.10); he[planeta] = 0
  mejor_ruta = ruta

  for(p = 0; p < NUM_PLANETAS; p++):
    if(p != planeta):
      if(mapa[planeta][p]!=-1 AND mapa[planeta][p]<=fuel):
        candidato = ruta
        candidato.add(p)

        candidato = es_mejor(he, mapa, candidato, fuel-mapa[planeta][p]

        if(es_mejor(candidato, mejor_ruta)): mejor_ruta = candidato

  return mejor_ruta

solucion = explora(HELIO, MAPA, [0], 2000)
```

C. Robo perfecto en Villa Rejilla

Envíos	Válidos	% éxito
0	0	0 %

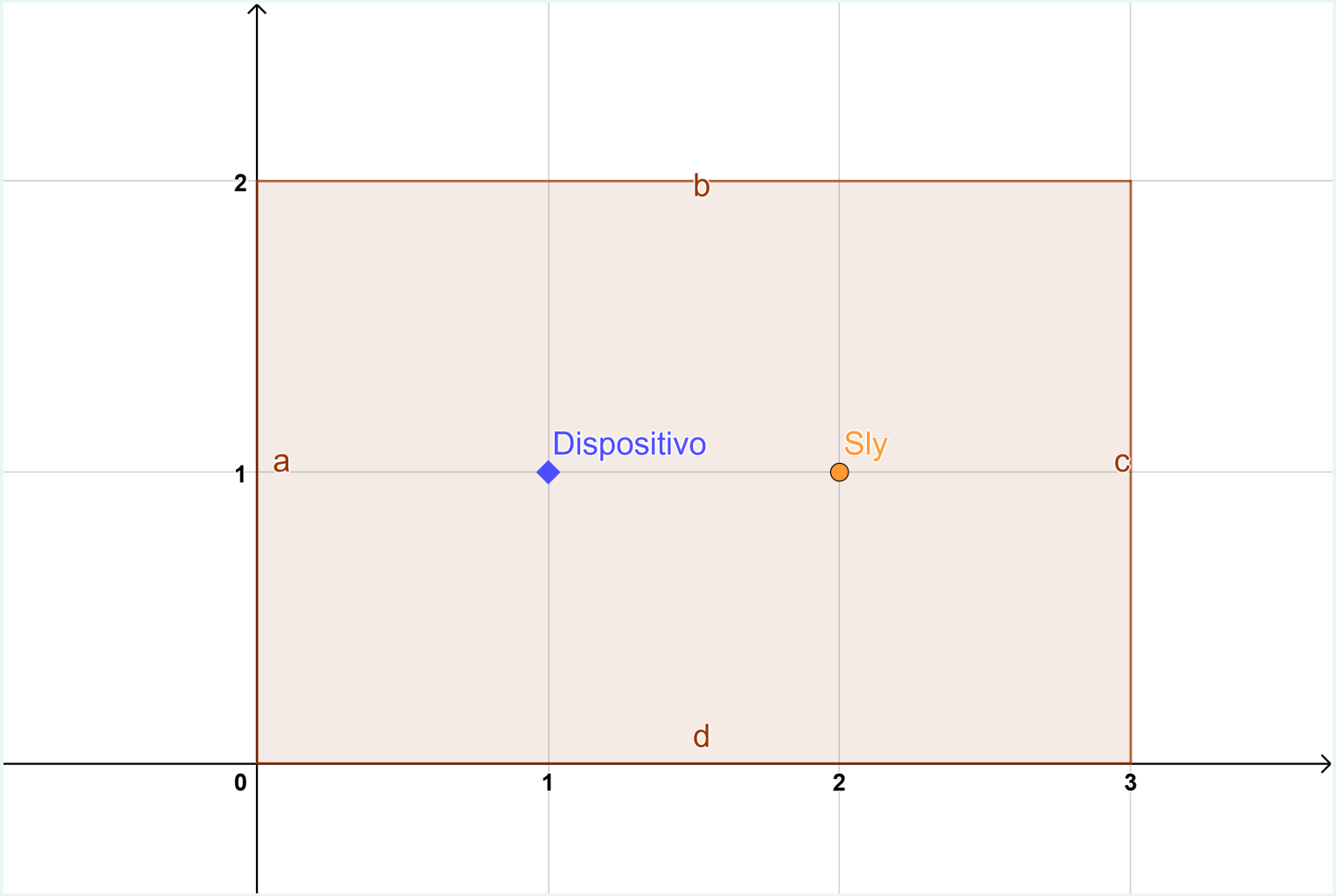
C. Robo perfecto en Villa Rejilla

Este problema consistía en calcular desde cuántos ángulos era posible detectar a nuestro protagonista sabiendo la posición tanto de éste como del detector dentro de una habitación con espejos.
Observaciones importantes:

- Detectar a Sly se puede interpretar como disparar un láser (con alcance limitado) desde el dispositivo.
- En lugar de *disparar* rayos y ver si impactan a Sly, es mejor analizar qué posiciones (reales y reflejadas) tienen tanto Sly como el dispositivo.
- Objetos detrás de otros objetos no son visibles por el dispositivo. Esto aplica tanto para Sly como para el dispositivo.

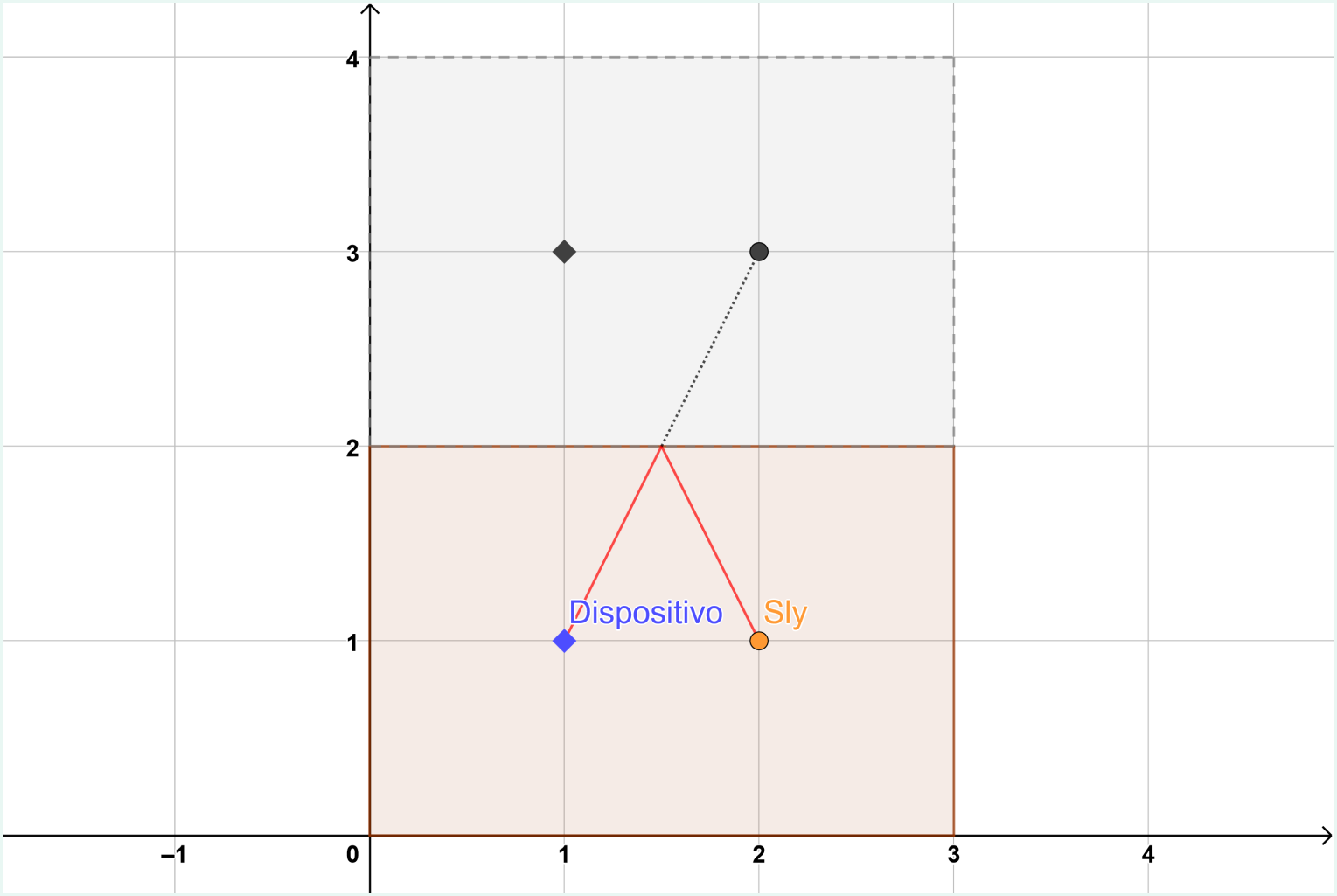
C. Robo perfecto en Villa Rejilla

Veamos cómo resolver el problema paso a paso basándonos en el caso de ejemplo:



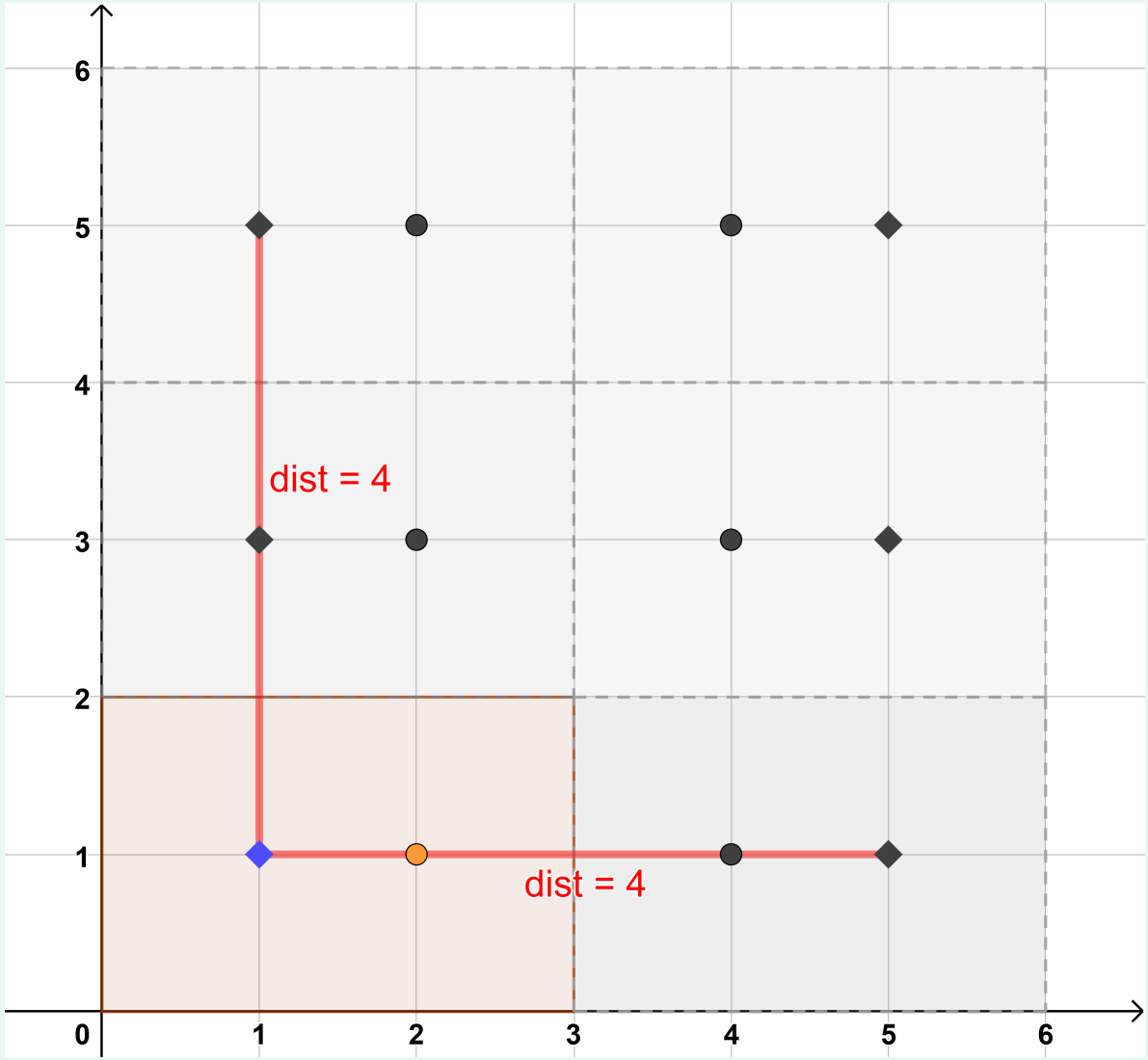
C. Robo perfecto en Villa Rejilla

En lugar de calcular rebotes en las paredes, podemos extender la sala para calcular las posiciones reflejadas:



C. Robo perfecto en Villa Rejilla

La idea es calcular las posiciones reflejadas del dispositivo y Sly para el primer cuadrante, teniendo en cuenta el alcance:



C. Robo perfecto en Villa Rejilla

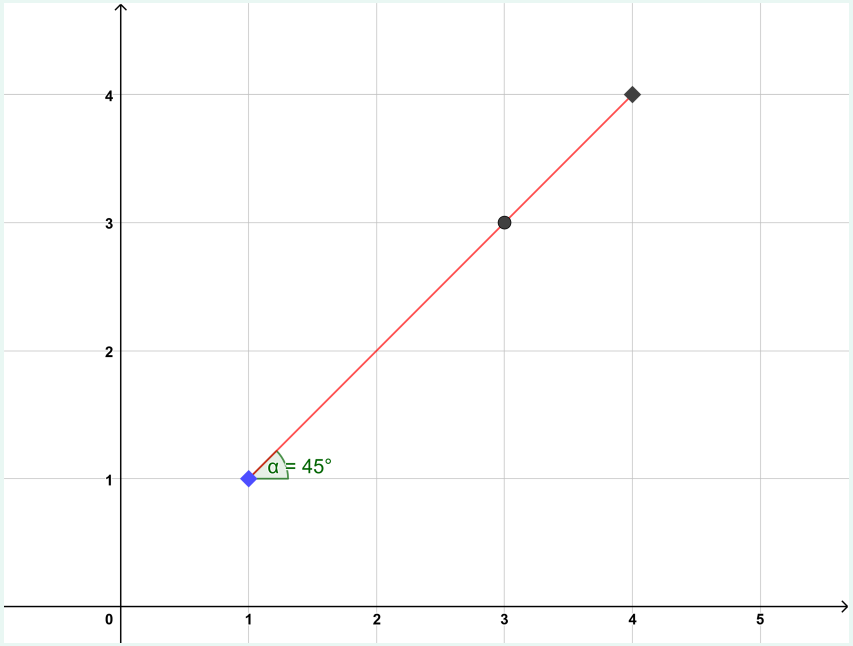
Las posiciones del resto de cuadrantes son muy fáciles de calcular:

- Segundo cuadrante: $[-1 \cdot x, y]$
- Tercer cuadrante: $[-1 \cdot x, -1 \cdot y]$
- Cuarto cuadrante: $[x, -1 \cdot y]$

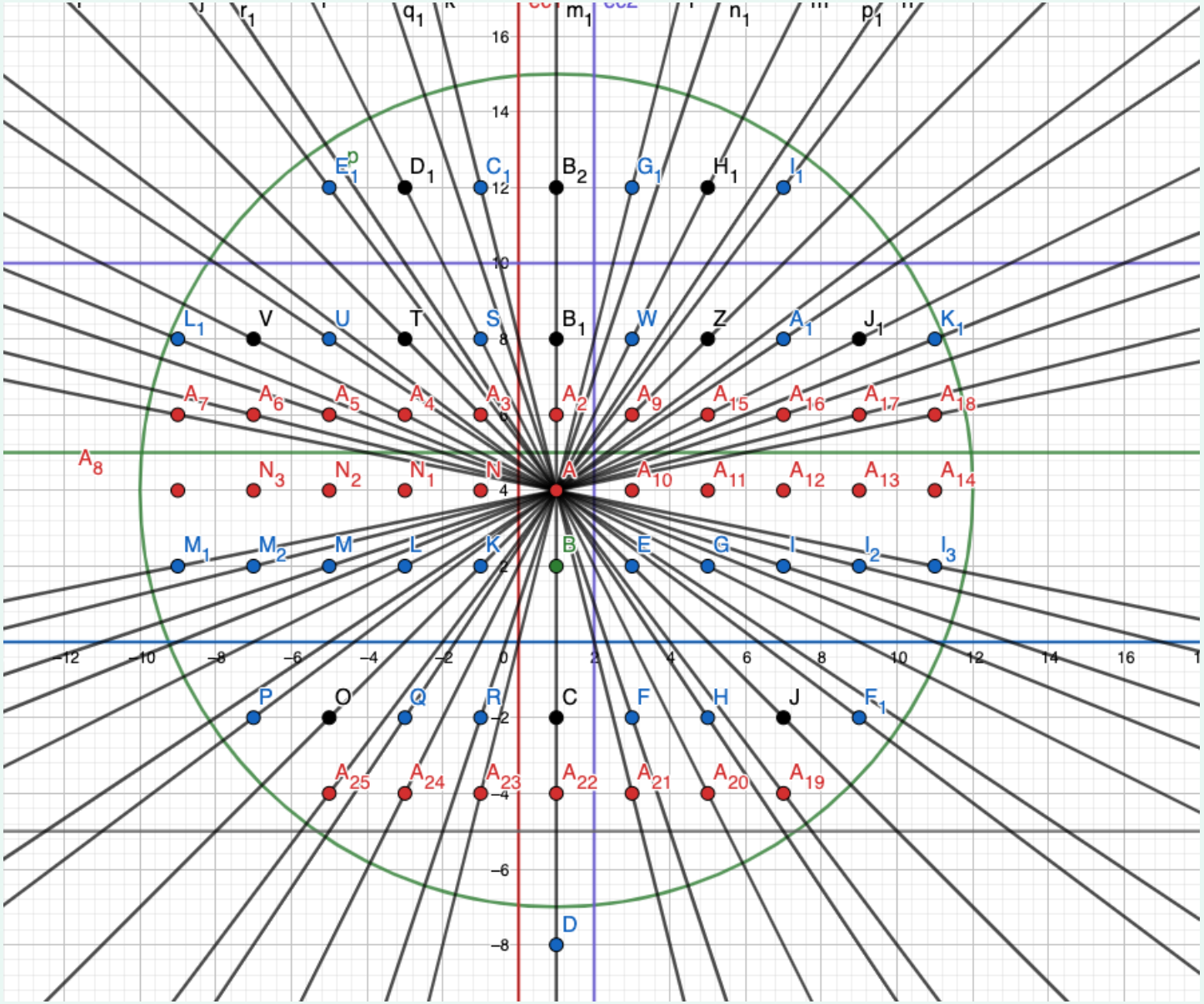
Filtraremos las posiciones que superen el alcance máximo del dispositivo calculando la distancia euclídea desde el dispositivo al resto de posiciones:

$$distancia = \sqrt{(x - x_d)^2 + (y - y_d)^2}$$

Por último, para eliminar aquellas posiciones obstaculizadas por otros objetos basta con calcular los ángulos desde el dispositivo al resto de posiciones con $atan2$



C. Robo perfecto en Villa Rejilla



K. ¡A Fortnitear!

Envíos	Válidos	% éxito
0	0	0 %

K. ¡A Fortnitear!

K. ¡A Fortnitear!

K. ¡A Fortnitear!

- Encontrar la configuración óptima del inventario de Elmorenus.
- Mochila con partición $\rightarrow$ voraz.
- **¡OJO!**: a Elmorenus no le gustan algunos tipos de armas.
  - Si la siguiente arma no le gusta, basta con pasar a la siguiente.
- Esquema voraz básico.
- Si no ordenas antes de elegir, $O(N^2) \rightarrow TLE$.
- Ordenando, $O(n \log n) \rightarrow AC$.