AdaByron 2021

Estadísticas y Soluciones

accenture idealista

ThoughtWorks® autentia atsistemas URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	**Categoría**
A - Dickie y las subsecuencias	Coordinate compression, Programación Dinámica, RSQ, BIT
B - Byron y las cervezas	RMQ, Segment-Trees
C - Bar BBTOtra	Floyd-Warshall, Backtracking
D - Lluvia de documentos	Set, Mapas, Arrays
E - Flamingo airlines	Pilas, Ad-Hoc
F - Las cartas Pukemon	Pensar, Logaritmos, Divisiones
G - El alineador	Arrays, Set, Mapas, Time Waster
H - Nos vamos de viaje	Dijkstra, Programación dinámica
I - First Dates	Doble búsqueda binaria, Matrices
J - La Agenda del recuerdo	Trie
K - Coleccionando comics	Cola de Prioridad, Pilas, Greedy
L - Fiesta madrileña	Grafos, Puntos de articulación, Componentes conexas
M - La fortaleza	Geometría, Convex-Hull

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - Dickie y las subsecuencias	22	30 B
B - Byron y las cervezas	27	45.2MB
C - Bar BBTOtra	12	20 KB
D - Lluvia de documentos	8	1.6 MB
E - Flamingo airlines	11	42.9 MB
F - Las cartas Pokémon	4	6.94 MB
G - El alineador	8	1.93 KB
H - Nos vamos de viaje	32	5.54 MB
I - First Dates	17	34.4 MB
J - La Agenda del recuerdo	7	6.8MB
K - Coleccionando comics	15	10.8 MB
L - Fiesta madrileña	9	584 KB
M - La fortaleza	6	3.3MB
- **Total**	178	160MB (+-)

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - Dickie y las subsecuencias	Solo Verdis Oscuros (UCM)	172'
B - Byron y las cervezas	TempleOS Devs (UCM)	18'
C - Bar BBTOtra	Solo Verdis Oscuros (UCM)	149'
D - Lluvia de documentos	Solo Verdis Oscuros (UCM)	4'
E - Flamingo airlines	- C+++ (UCM)	12'
F - Las cartas Pokémon	Solo Verdis Oscuros (UCM)	25'
G - El alineador	UwUntu (UAM)	48'
H - Nos vamos de viaje	-	-
I - First Dates	Echo (UAM)	33'
J - La Agenda del recuerdo	Los Greedioides... (UCM)	107'
K - Coleccionando comics	Solo verdis Oscuros (UCM)	39'
L - Fiesta madrileña	Echo (UAM)	110'
M - La fortaleza	-	-

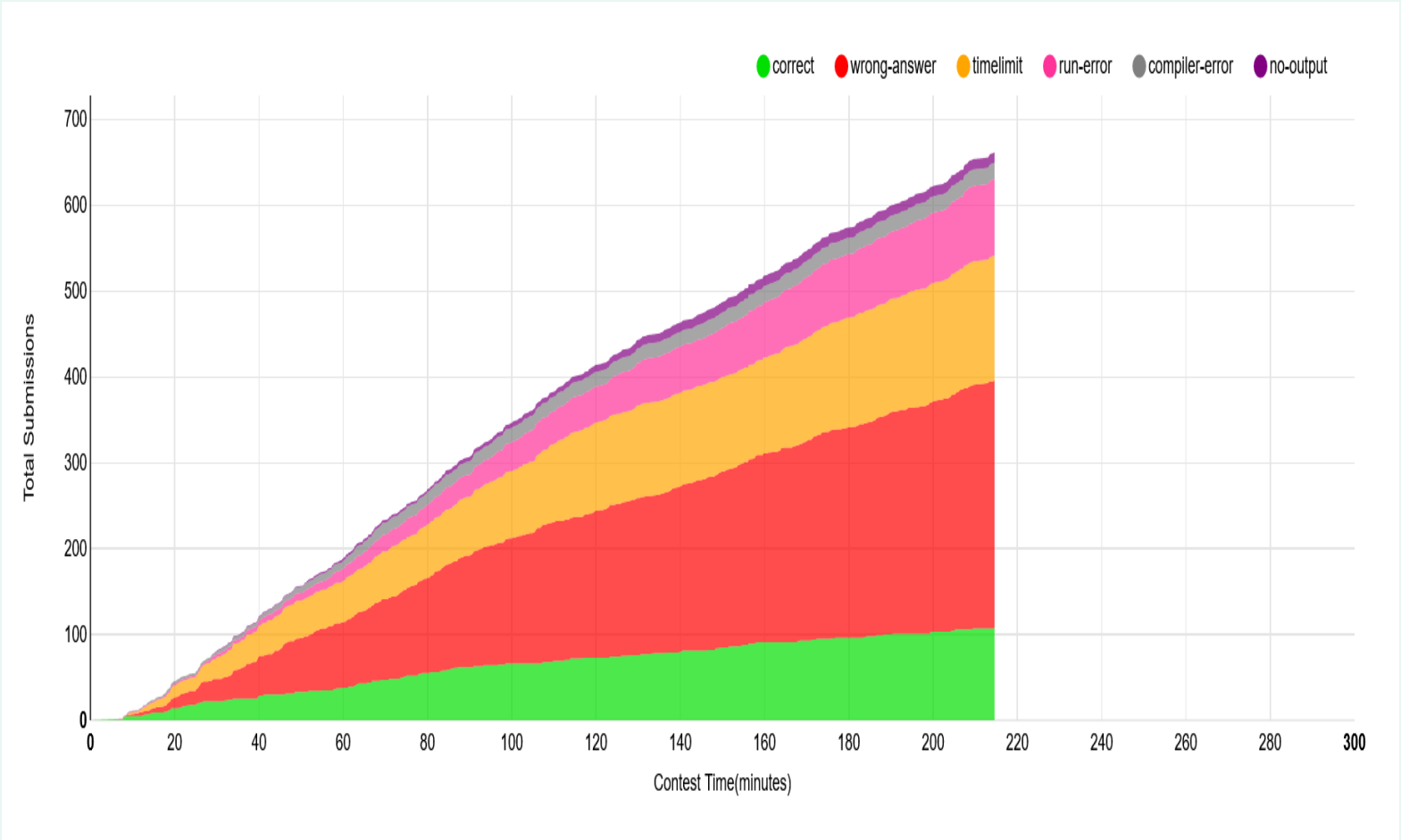
*Antes de congelar el marcador.

Estadísticas*

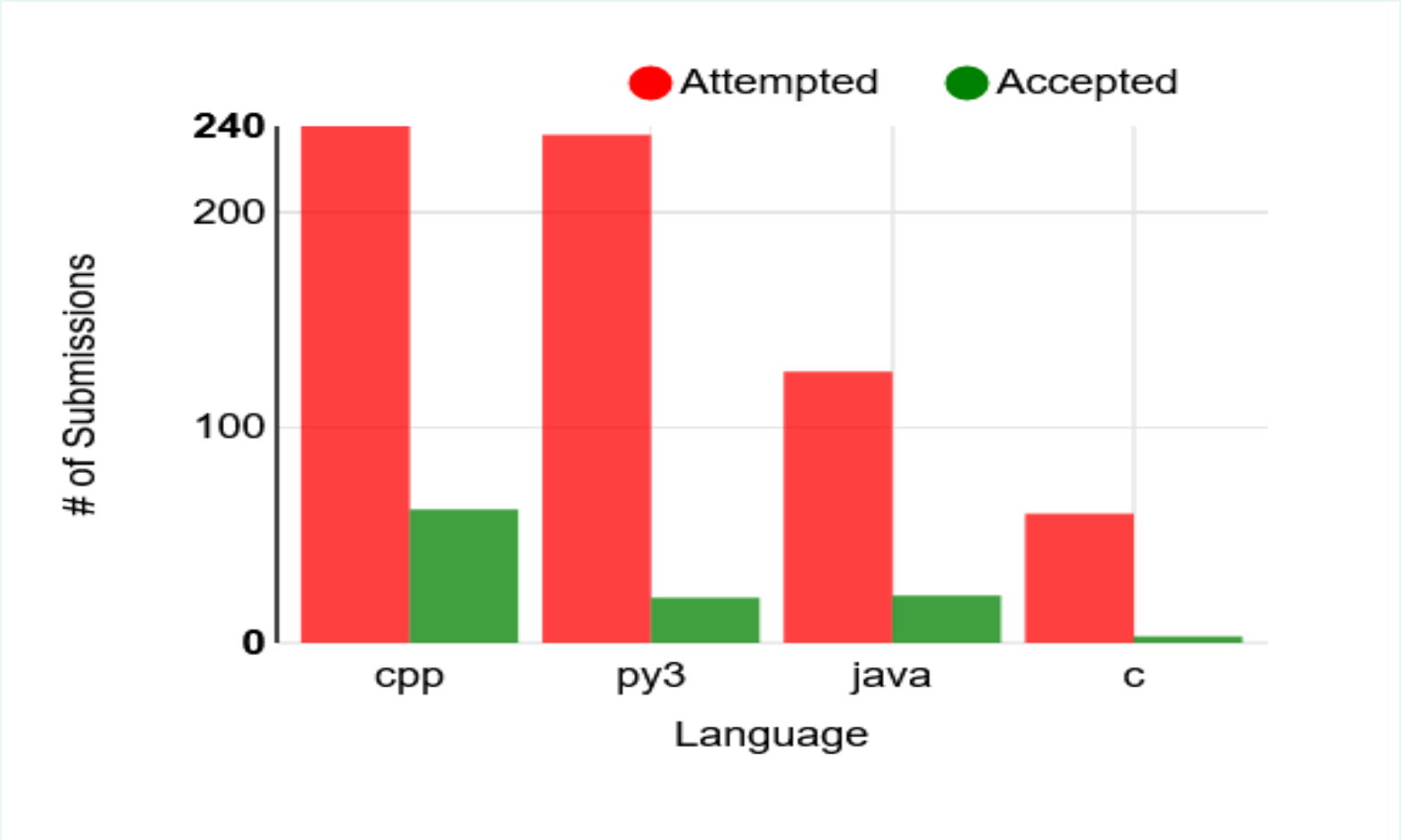
Problema	Envíos	Válidos	% éxito
A - Dickie y las subsecuencias	17	2	11 %
B - Byron y las cervezas	118	8	6,7 %
C - Bar BBTOtra	11	2	18 %
D - Lluvia de documentos	95	26	37 %
E - Flamingo airlines	69	10	14,4 %
F - Las cartas Pokémon	80	15	18,6 %
G - El alineador	30	8	26 %
H - Nos vamos de viaje	0	0	0 %
I - First Dates	83	16	19,3 %
J - La Agenda del recuerdo	70	2	3 %
K - Coleccionando comics	49	9	18,3 %
L - Fiesta madrileña	3	3	100 %
M - La fortaleza	2	0	0 %

*Antes de congelar el marcador.

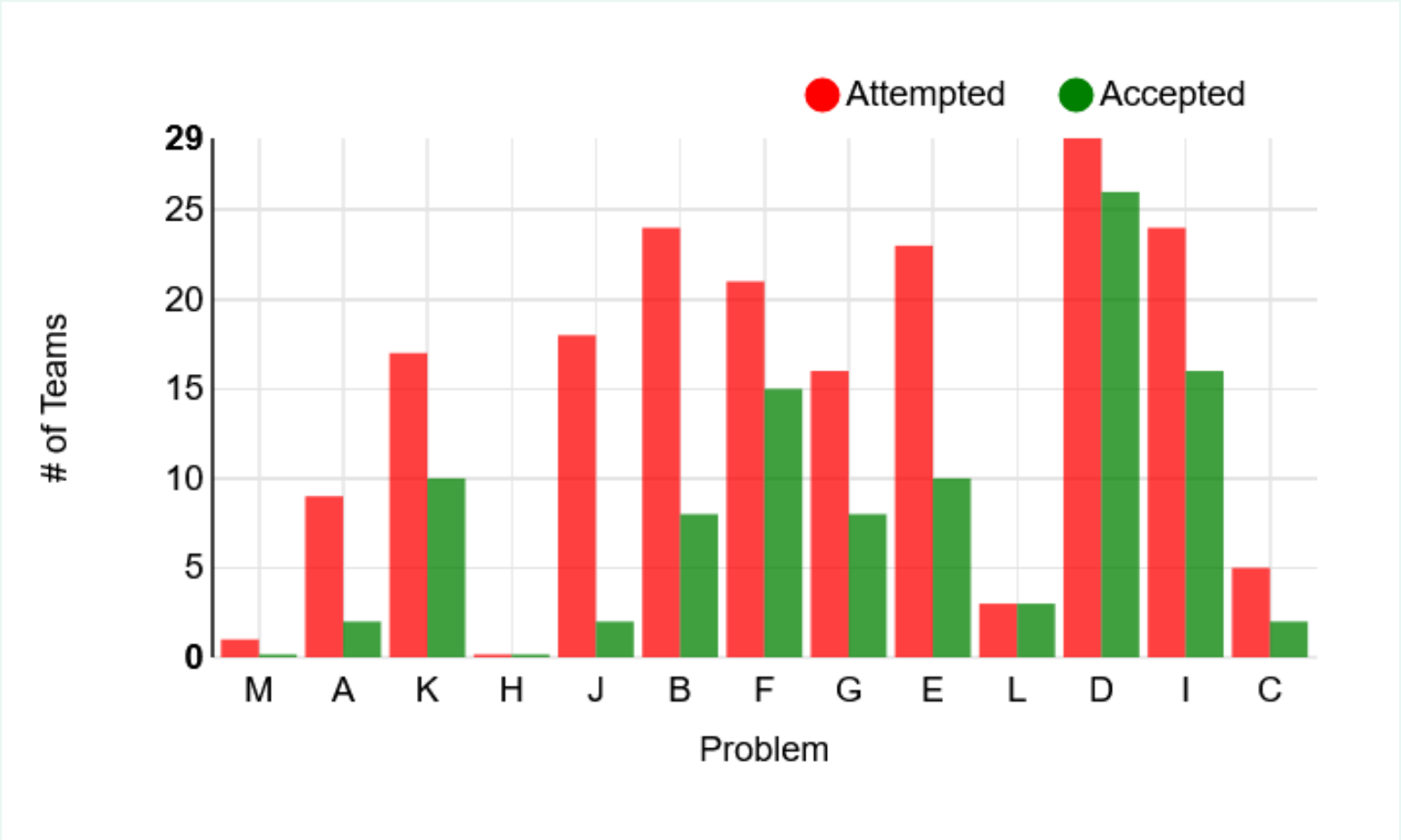
Estadísticas varias



Estadísticas varias



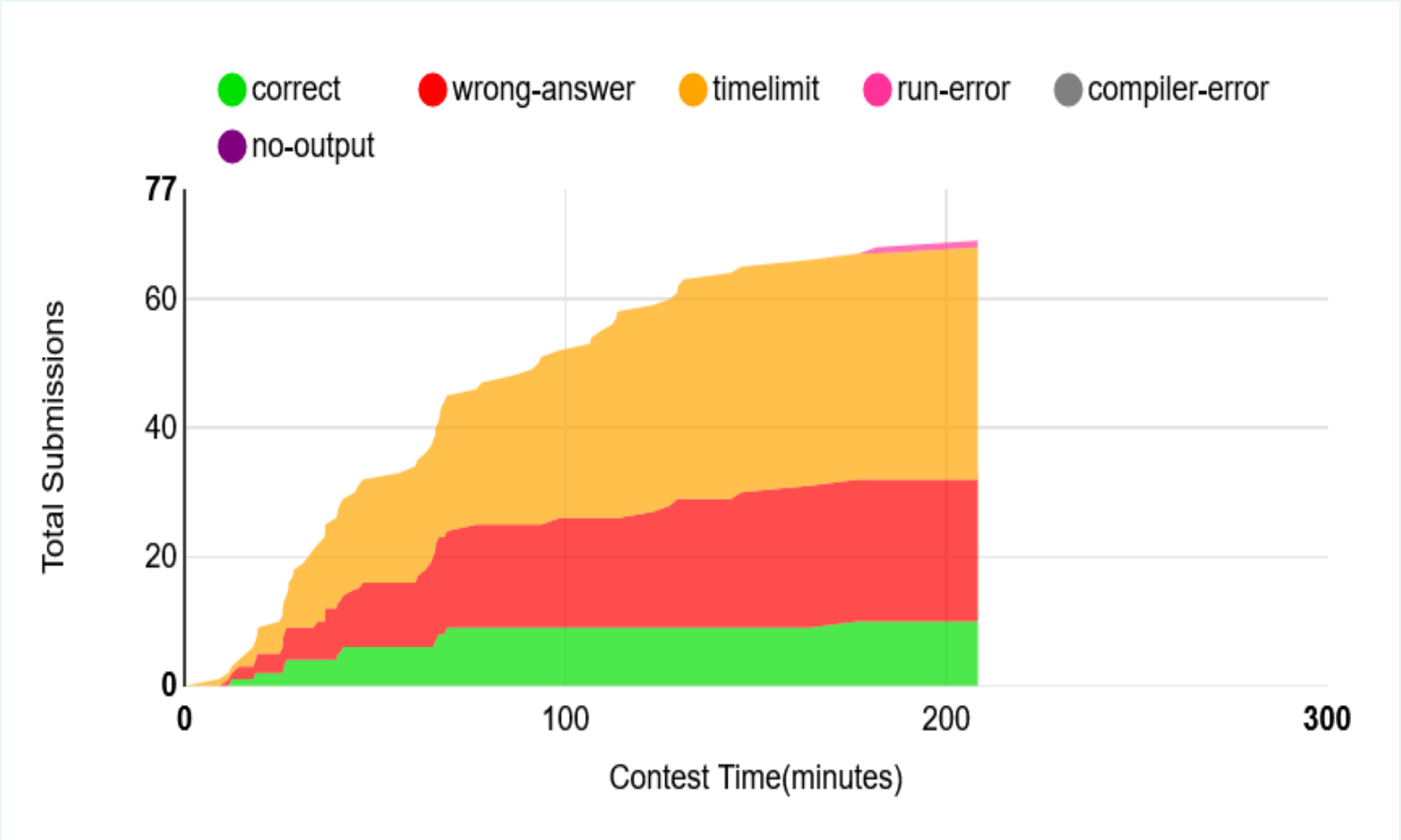
Estadísticas varias



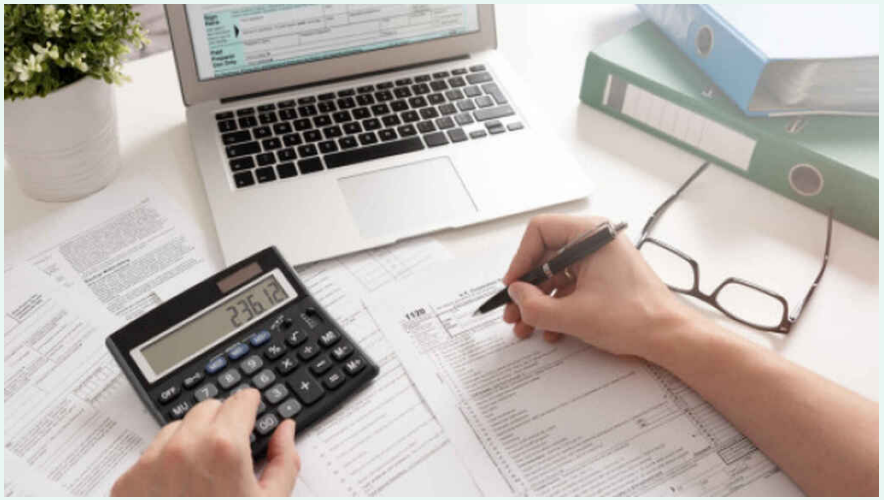
D. Lluvia de documentos

Envíos	Válidos	% éxito
95	26	27.37 %

D. Lluvia de documentos



D. Lluvia de documentos



En este problema se trataba de detectar **índices** duplicados en la lista de elementos de entrada, sin tener en cuenta si existían nombres duplicados o no. Para resolver este problema se pueden seguir dos enfoques:

- Utilizar una estructura de datos de conjunto (Set) y leer la entrada almacenando los índices y obviando los nombres
- Utilizar un array de String de $N+1$ posiciones (es importante que tenga $N+1$, dado que los índices de la entrada comienzan en 1) o bien utilizar uno de N posiciones, restando 1 a cada índice de la entrada

D. Lluvia de documentos

Si optamos por la segunda opción, debemos inicializar el array con un String vacío (“”) o con los identificadores que se van leyendo.

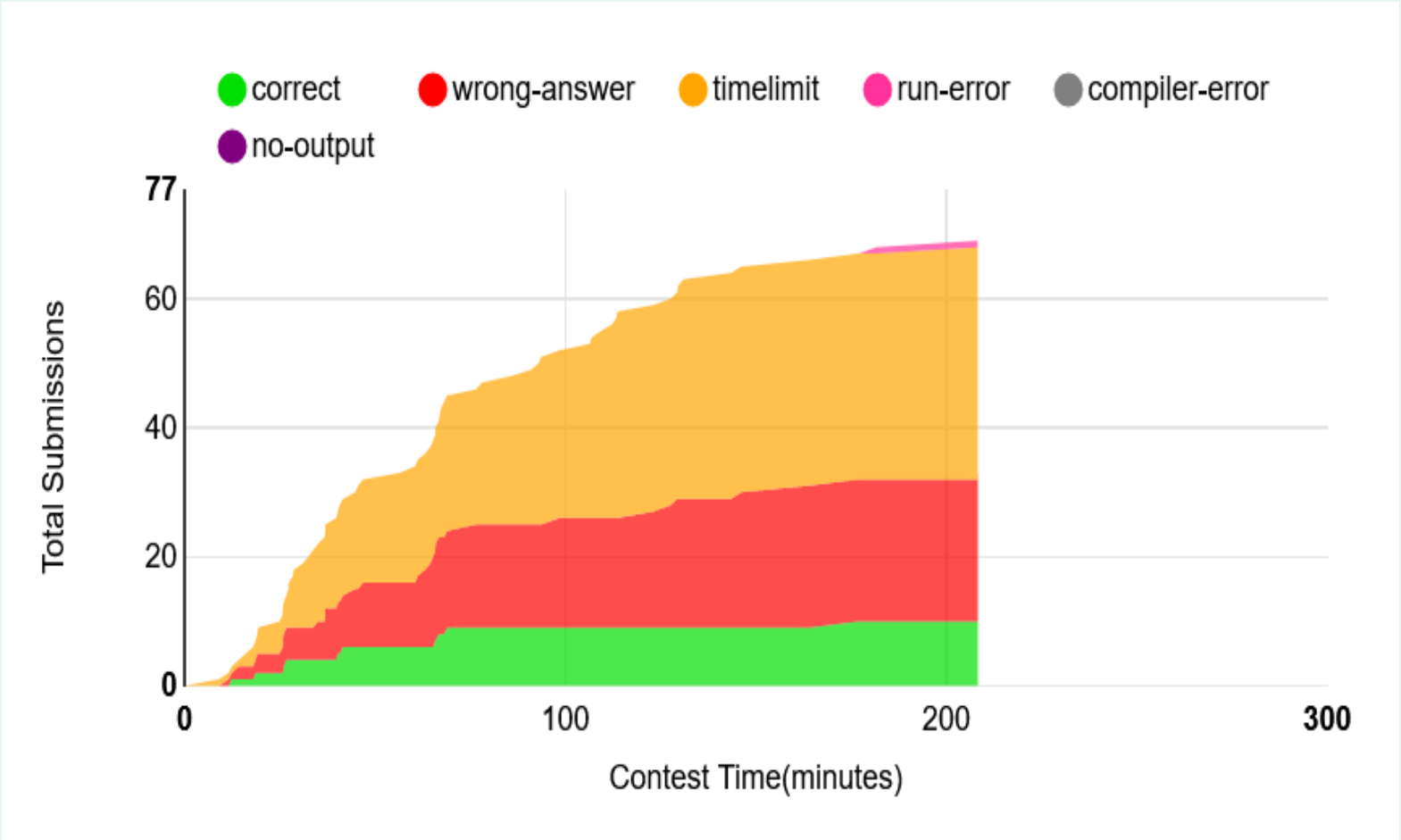
D. Lluvia de documentos

Lo que no se debe intentar es almacenar los identificadores y posteriormente comprobar la no-duplicidad. Esto supone una solución en $O(n^2)$, cuando puede resolverse en $O(n)$.

E. Flamingo Airlines

Envíos	Válidos	% éxito
69	10	14 %

E. Flamingo Airlines



E. Flamingo Airlines

Estamos en el aeropuerto viendo a flamencos embarcar en un avión... y liarla cada vez que embarca un flamenco de menor edad. Sabiendo el orden de la cola y la edad de cada flamenco en ella, ¿Cómo podemos calcular con cuánto retraso saldrá el avión?

E. Flamingo Airlines

Algunas ideas iniciales que no funcionan...

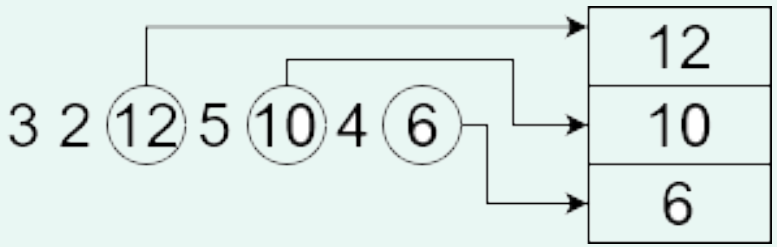
1. Primera idea: Cada vez que embarque un flamenco, recorrer toda la cola buscando el mayor elemento... TLE. Esta idea se ejecuta en $O(n^2)$. ¡Podemos hacerlo mucho mejor!
2. Segunda idea: Recorrer la cola buscando el mayor elemento, y sólo buscar de nuevo cuando el mayor flamenco embarque... Más rápido, pero no supera el TLE: Sigue siendo $O(n^2)$

E. Flamingo Airlines

Idea clave: Siempre será el mayor flamenco de la cola el que más escándalo montará, hasta que embarque.
Pensemos en el caso base: El último flamenco nunca montará escándalo cuando embarque. Sin embargo, podrá montarlo si el flamenco que tenga delante es menor que él.
Para resolver este problema, necesitaremos una pila. Podemos recorrer la cola del aeropuerto en sentido inverso, comparando cada vez la cabeza de la pila con el elemento actual. Si es mayor (o igual), este será el flamenco que más escándalo montará, por lo que podemos introducirlo a la pila.

E. Flamingo Airlines

Gracias a ello conseguiremos una pila como la siguiente:



Cuando ahora pasemos a recorrer la cola en el orden inicial, embarcando a los flamencos, podremos saber fácilmente cuál es el flamenco que más escándalo montará: Aquel que esté a la cabeza de la pila. Eso sí, si nos lo encontramos, tendremos que eliminarlo de la pila, pues habrá embarcado, y ya estará sentado en el avión. Esta aproximación resuelve el problema en complejidad lineal: $O(n)$

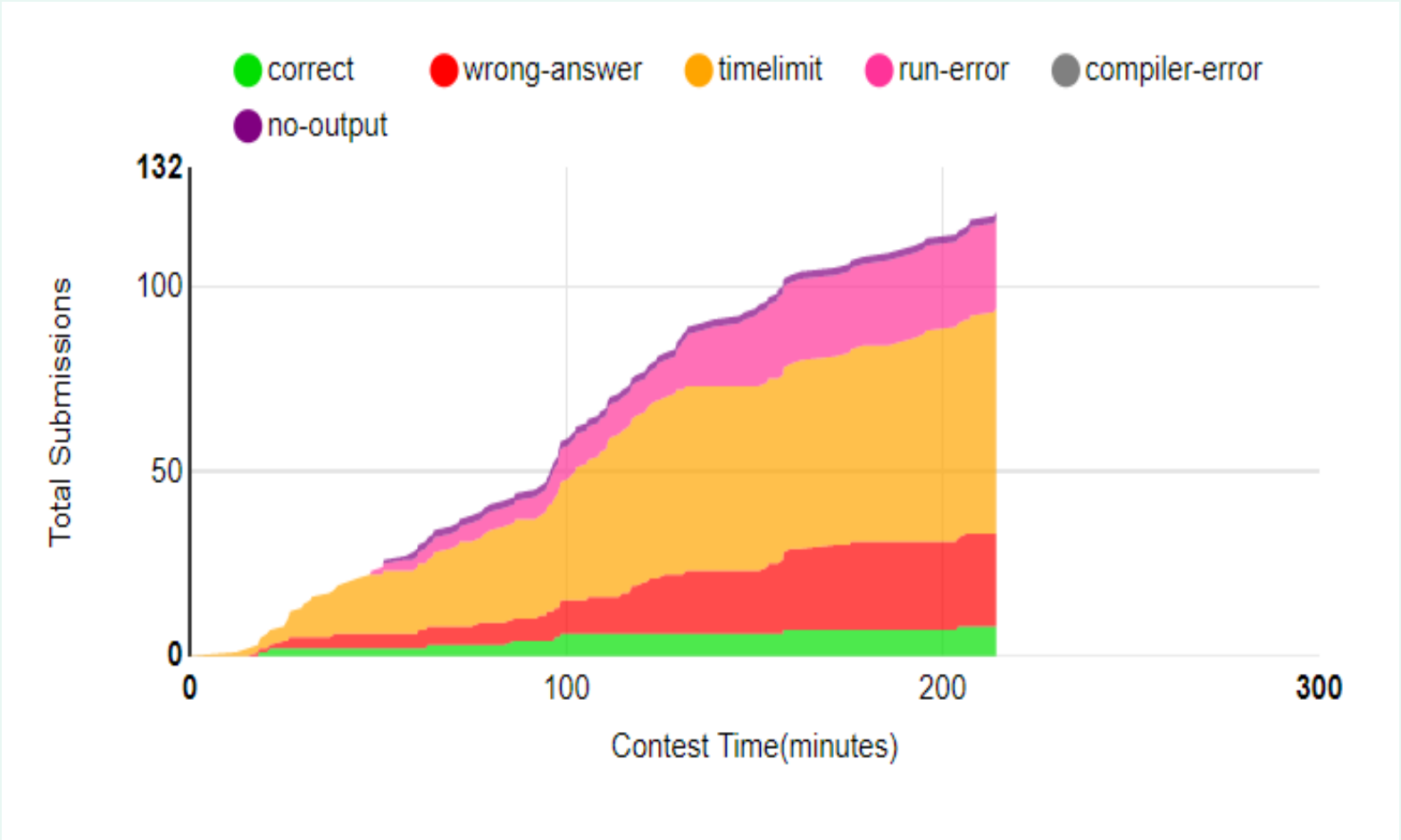
E. Flamingo Airlines

Un último apunte: Es muy importante fijarse en los límites, la espera puede ser... ¡casi eterna! Hasta 10^6 flamencos de edad 10^6 pueden generar una espera de hasta 10^{12}, que es mayor de 2^{32}. Usuarios de Java y C++, era necesario usar un tipo de dato que permitiera números más grandes (long en java y long long int en C++). ¡A volar!

B. Byron y las cervezas

Envíos	Válidos	% éxito
118	8	6.77 %

B. Byron y las cervezas



B. Byron y las cervezas

- Este es el típico problema de Range Minimum Query (RMQ)
- Tomar en cuenta que el resultado podia ser long long
- Solución $O(N^2)$ no funcionaría ya que los rangos de las pueden ser muy grandes
- Debemos utilizar alguna estructura de datos auxiliar para responder las queries. Algunas alternativas son: Segment Tree, Sparse Table, variante del Binary Indexed Tree.

B. Byron y las cervezas

Complejidad en tiempo para responder cada RMQ ($O(RMQ)$):

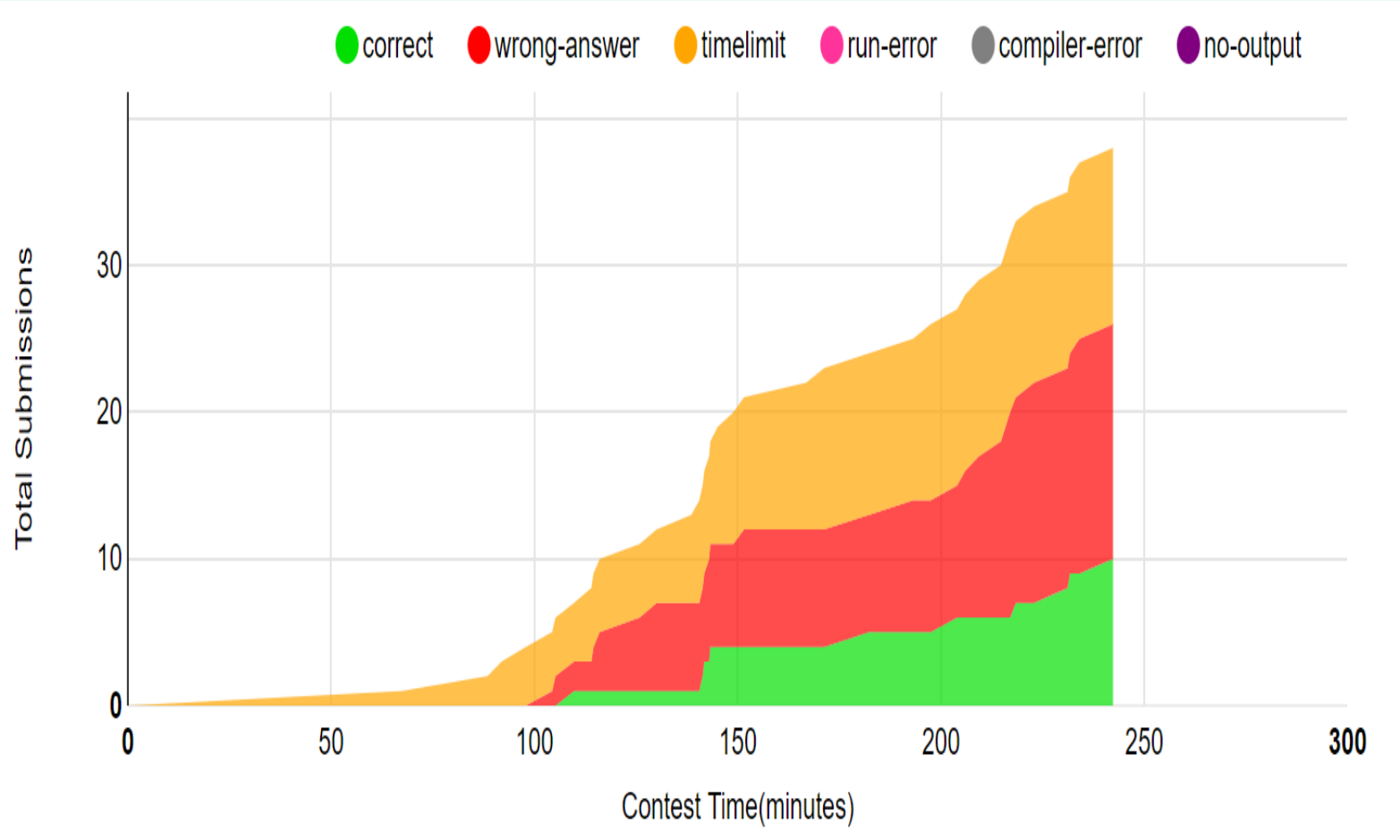
- Segment Tree: $O(log(N))$
- Sparse Table: $O(1)$
- Binary Indexed Tree: $O(log(N))$

Por lo tanto, la complejidad en tiempo de la solución esperada es: $O(N + Q \times O(RMQ))$

F. Las cartas Pukemon

Envíos	Válidos	% éxito
80	15	18,6 %

F. Las cartas Pukemon



F. Las cartas Pukemon

Tenemos que encontrar las cartas, dada cualquier posición. Si nos colocamos en el centro, en cada paso reducimos a la mitad el espacio de búsqueda –> Búsqueda Binaria
Otra forma de verlo: ¿Cuántos pasos tenemos que dar de búsqueda binaria en el peor caso?

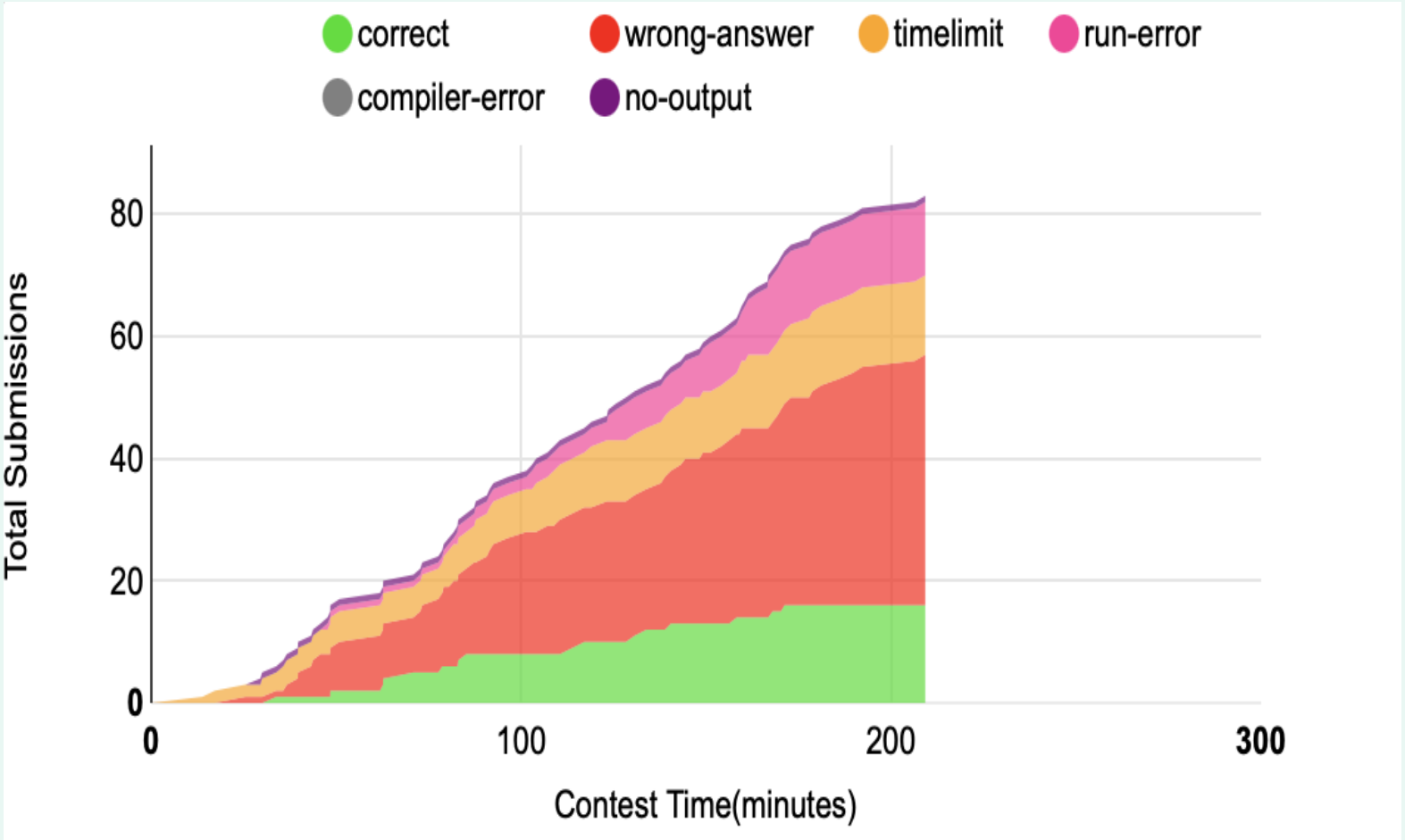
F. Las cartas Pukemon

- Solución 1: $ceiling(log_2(max(X, Y, Z) + 1)$
- Solución 2: $M = 2^{(battery)}$; $checkifmax(X, Y, Z) < M$
- Solución 3:
  - $M = max(X, Y, Z)$
  - $while\ M! = 0$
  - $counter++$

I. First Dates

Envíos	Válidos	% éxito
82	16	19,28 %

I. First Dates



I. First Dates

Contexto:

- El objetivo del problema es conseguir que cierto número de solteros encuentren el amor.
- Para ello, tenemos una sala cuadrada con habitaciones donde:
  - Hay 1 soltero en cada habitación y cada uno tiene un identificador.
  - Hay solteros sin habitación asignada que serán emparejados con los más adecuados de las habitaciones.
- A la hora de crear parejas, cada soltero sin habitación debe buscar su media naranja comparando sus identificadores.

Complejidad esperada: $O(\log N)$

I. First Dates

Solución 1: Doble búsqueda binaria.

- Modelado: Cada habitación es una celda de una matriz de $N \times M$ qué está ordenada.

18	33	37	47	54
57	59	66	74	75
136	164	166	170	175
187	190	205	216	217
222	227	232	241	247

- Fila: Buscar en la primera columna la posición en la que el elemento es menor al buscado y donde el elemento de la siguiente fila es mayor.
- Columna: Buscar en la fila seleccionada el número más cercano al deseado.

I. First Dates

Solución 2: Una única búsqueda binaria.

- Modelado: Guardar la matriz en una lista ordenada.

18	33	37	47	54

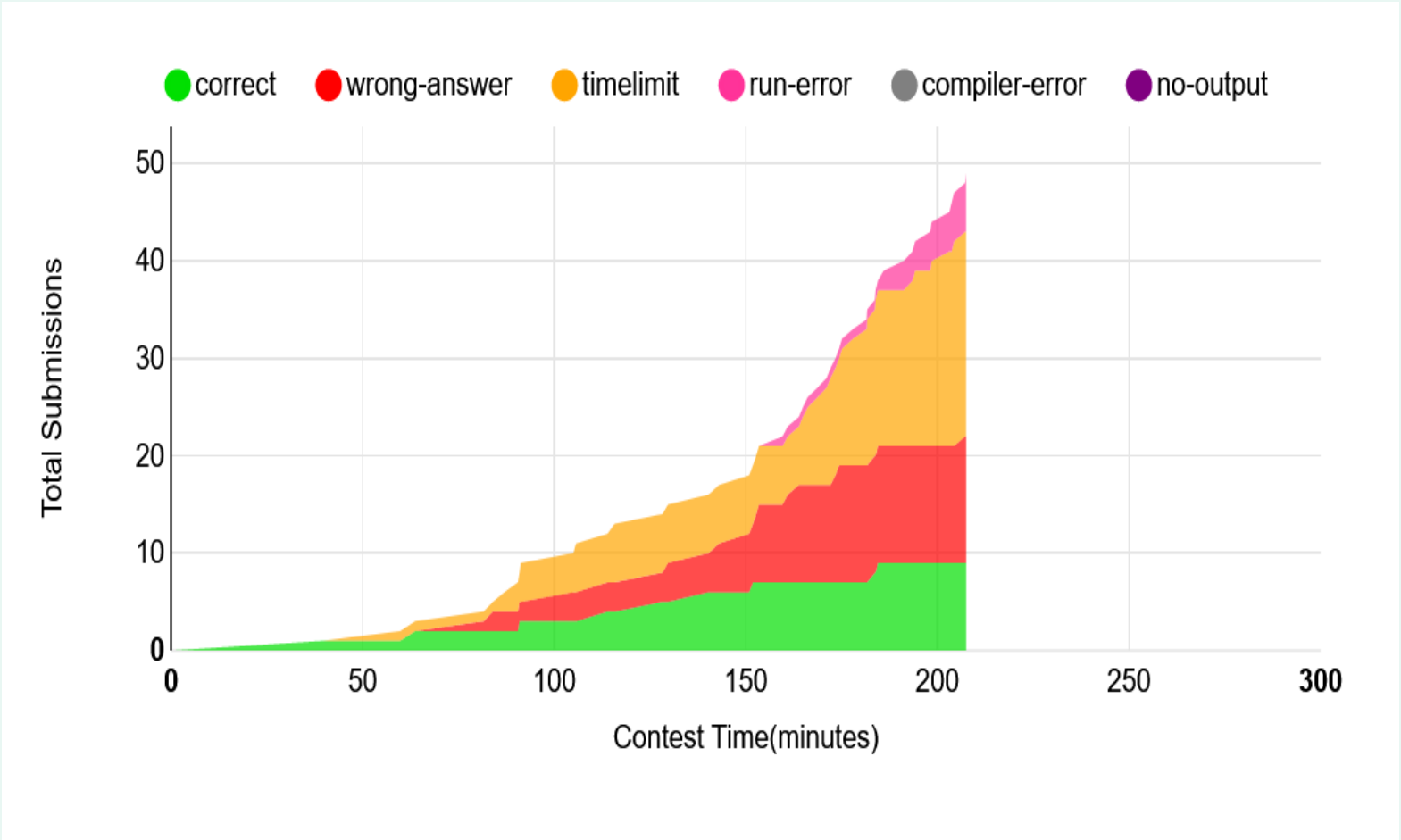
- Hacer una búsqueda binaria sobre la lista.
- Para sacar la posición de la habitación elegida se usa la fórmula:

$$Fila = pos/M$$
$$Columna = pos\,\%M$$

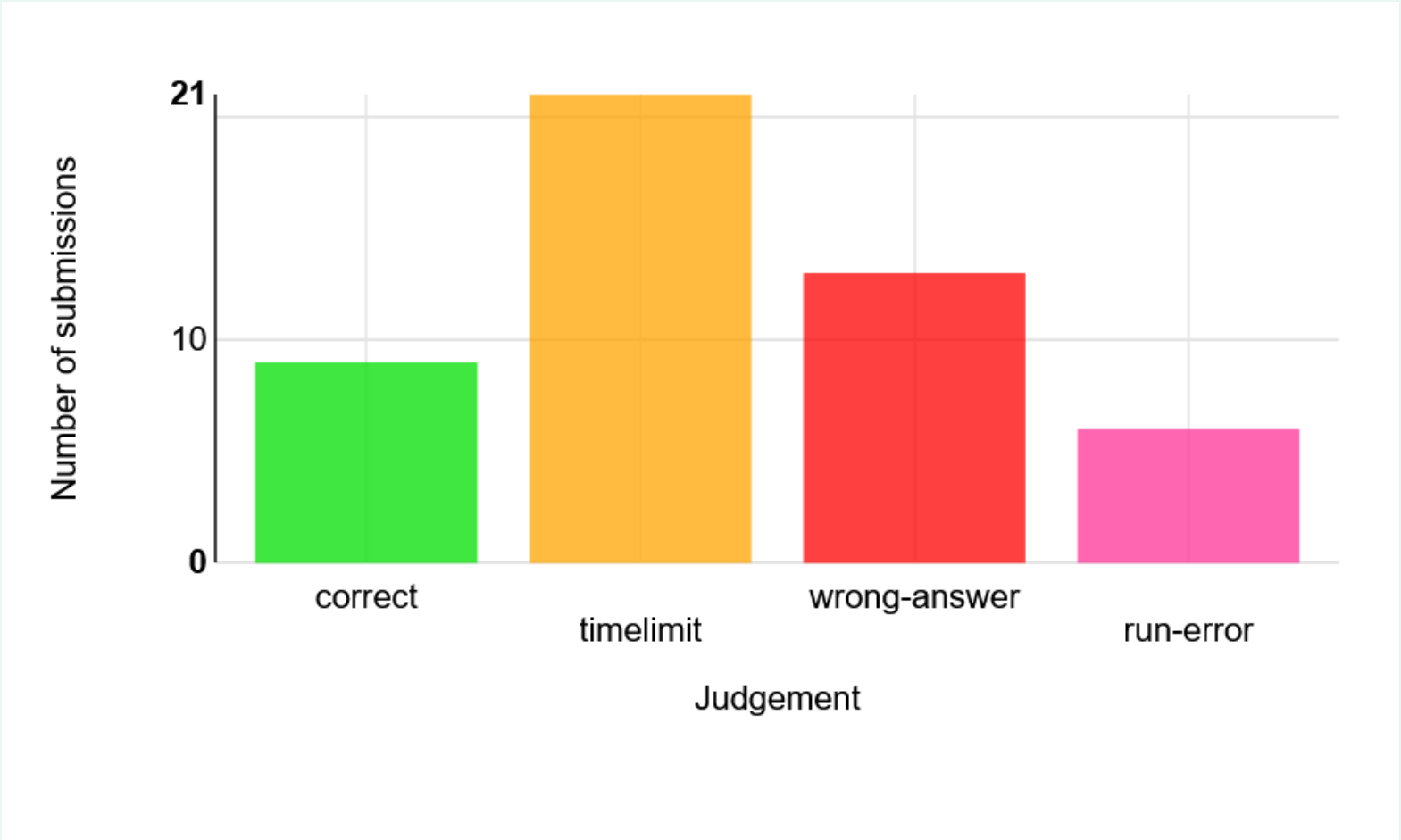
K. Coleccionando cómics

Envíos	Válidos	% éxito
49	9	18.36

K. Coleccionando cómics

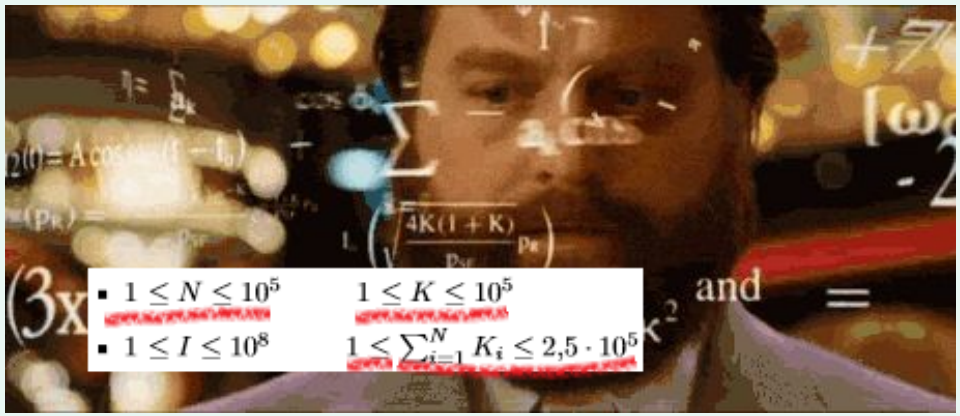


K. Coleccionando cómics



K. Coleccionando cómics

Después de recopilar todos los problemas del concurso nos hemos dado cuenta de que nos faltaba un problema voraz; y así es como nació este enunciado.
Se pide averiguar en qué momento se cogerá el cómic más valioso de la tienda.
La idea es simular el comportamiento de los clientes y contar cuál de ellos se llevará el ejemplar bueno.
Entonces hay que usar un bucle y en cada iteración buscar el menor valor entre las cimas de todas las pilas con otro bucle. ¿O no? Esta solución con dos bucles anidados tiene pinta de $O(N^2)$...



K. Coleccionando cómics

Además, la complejidad de $O(N^2)$ no se calcula

$$Npilas * tamao(pila)$$

sino como

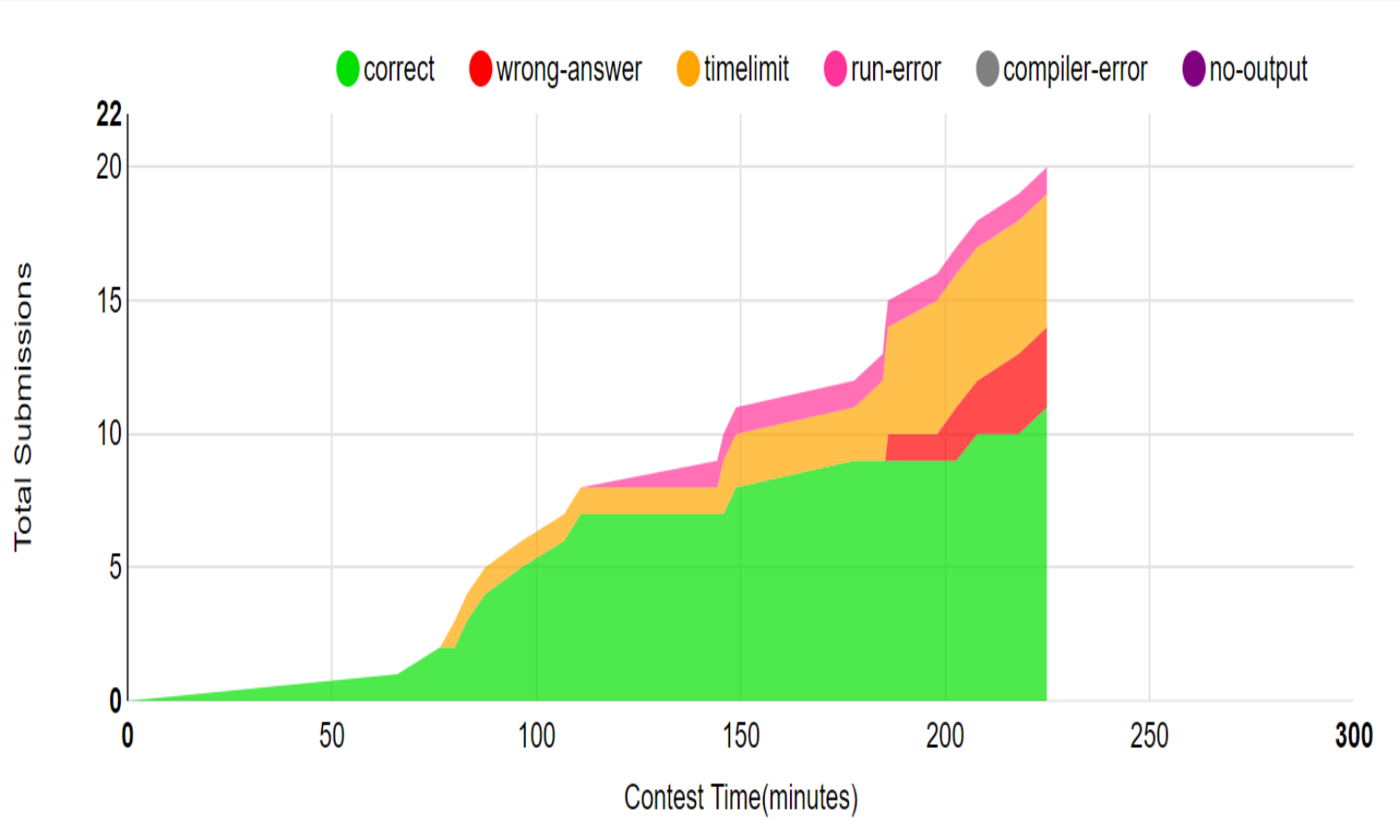
$$Ncmics * Npilas$$

La manera de arreglar esto es sustituir el bucle interno por una cola de prioridad para guardar las cimas de las pilas en todo momento. Cada vez que un cliente se lleva un cómic simplemente se elimina el primer elemento de la cola de prioridad y se inserta en ella el siguente cómic de la misma pila que el comic que se ha eliminado.
Para esto hay que usar un objeto (idComic,idPila) en la cola de prioridad o llevar la cuenta de manera externa (con un mapa) que cómic pertenece a que pila
Con esto la complejidad se reduce a $O(N * logP)$

G. El alineador

Envíos	Válidos	% éxito
30	8	26 %

G. El alineador



G. El alineador

El fútbol es un deporte muy bonito... pero complicado. El entrenador es aquella persona que se encarga de planificar los entrenamientos y organizar a sus jugadores el día de un partido mediante un esquema de juego o alineación. Generalmente, una alineación se compone de 4 tipos de jugadores: porteros, defensas, mediocentros y delanteros. Cada jugador es identificado por el nombre y el número que se muestra en su camiseta.

G. El alineador

En este problema se pide comprobar si un listado de jugadores es válido para jugar un partido con un esquema de juego específico.
Por ejemplo, si el esquema de juego establece que jugaran 4 defensas, en el listado de jugadores deberá haber exactamente 4 jugadores cuya posición sea defensa.
Si la alineación es correcta se imprime en un formato determinado. Si es incorrecta, se mostrará:"Error en la alineación".

```
5 1 1 0 1
4 1 9
9
Castolo
DL
5
Bautista
MC
3                                   Error en la alineacion
Facu
DF
1
Van de Vij
```

G. El alineador

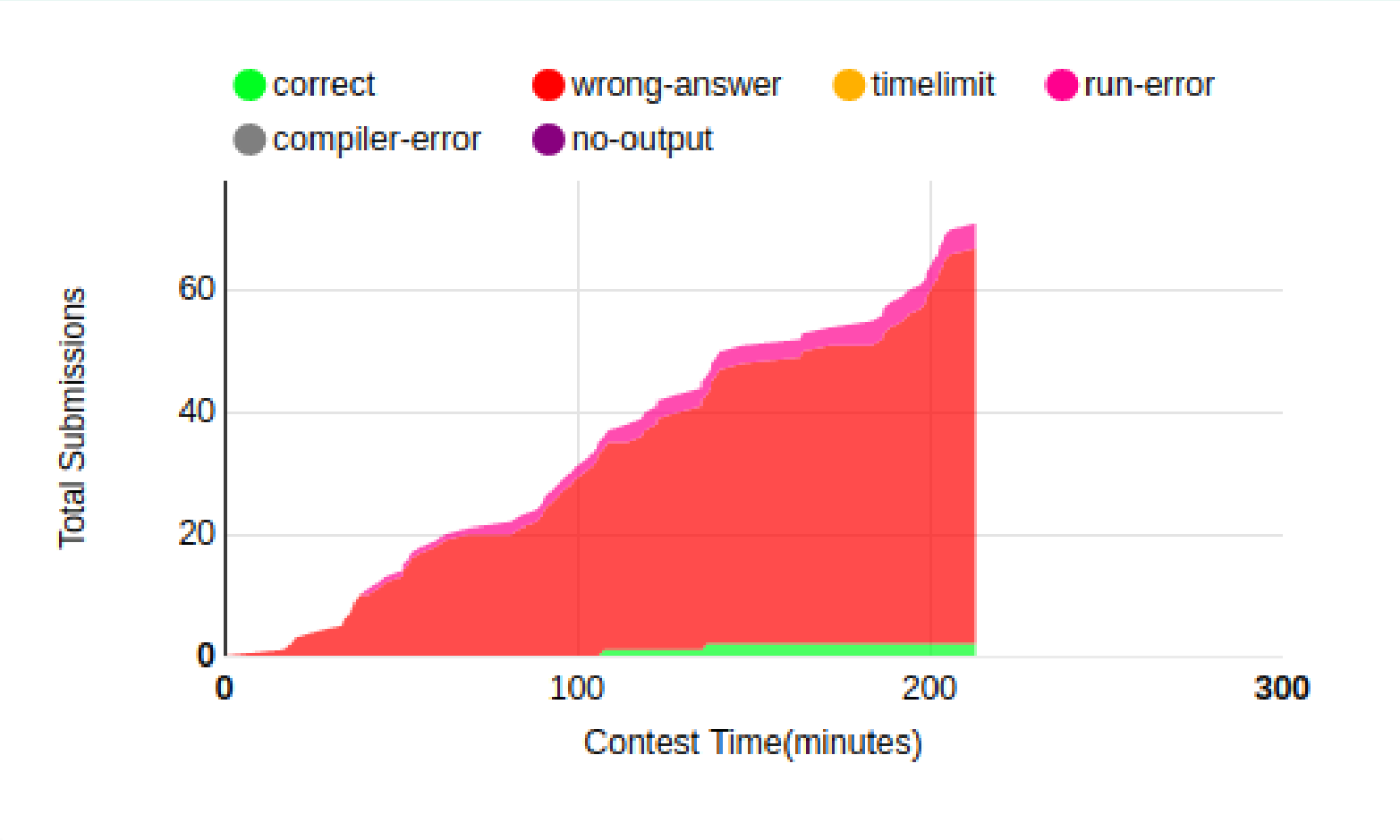
Ideas clave:

- Utilizar variables enteras como contadores del número de jugadores por posición.
- Utilizar una estructura similar a un *set* para almacenar los dorsales de los jugadores que estarán en la alineación.
- Por cada jugador:
  1. Se comprueba si su dorsal está en el set ($\mathcal{O}(1)$)).
  2. Si está, se comprueba que no se haya superado el número máximo de jugadores para su posición (P, DF, MC, DL).
  3. Si se ha superado el número de jugadores, el programa finaliza.
  4. Si no se han superado, se almacena el nombre del jugador en una variable que permita imprimir el resultado si la alineación es correcta.

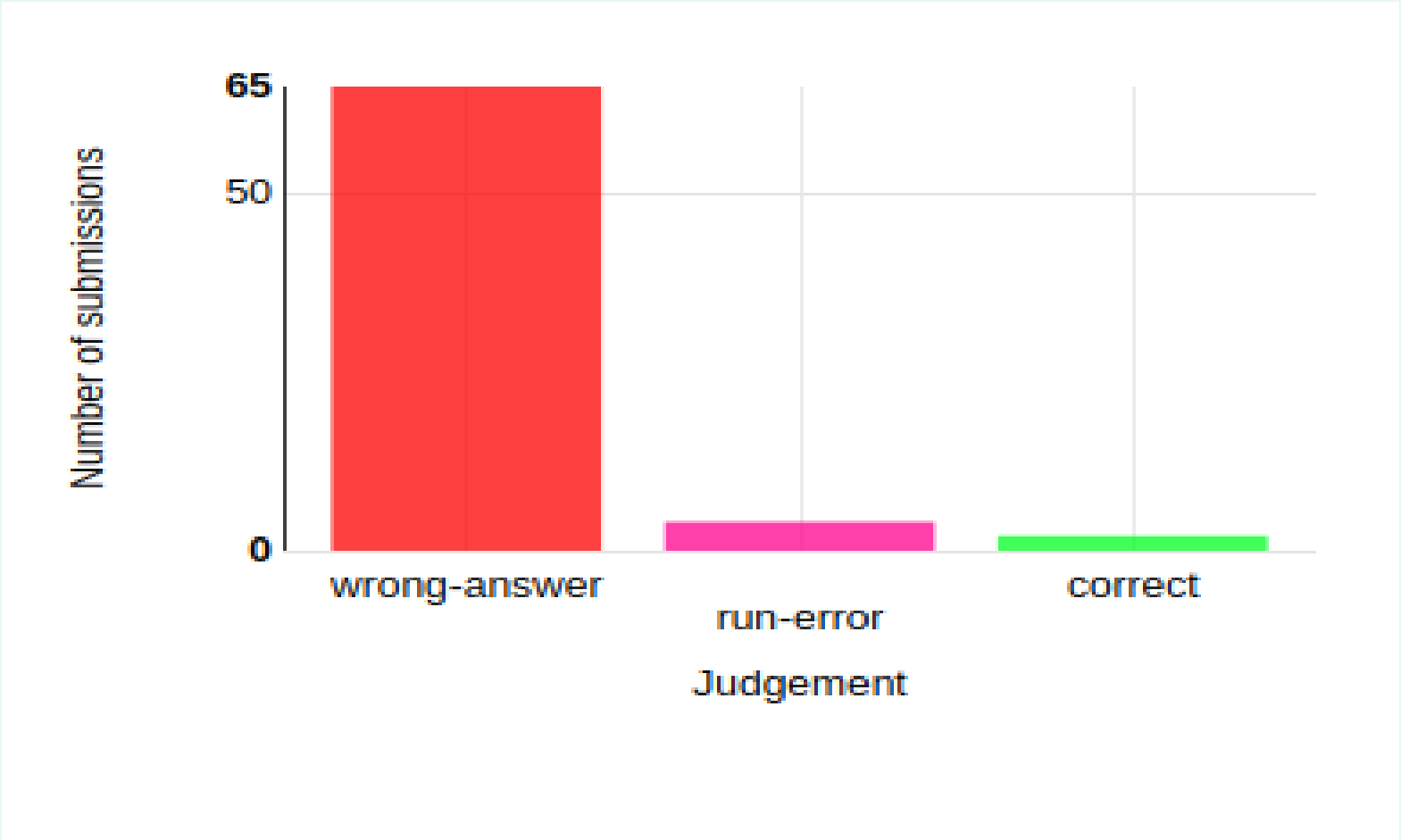
J. La agenda del recuerdo

Envíos	Válidos	% éxito
68	2	2.85 %

J. La agenda del recuerdo



J. La agenda del recuerdo



J. La agenda del recuerdo

- La idea principal del problema es: Dadas una lista de palabras y una lista de prefijos, encontrar todas las palabras que contengan un prefijo

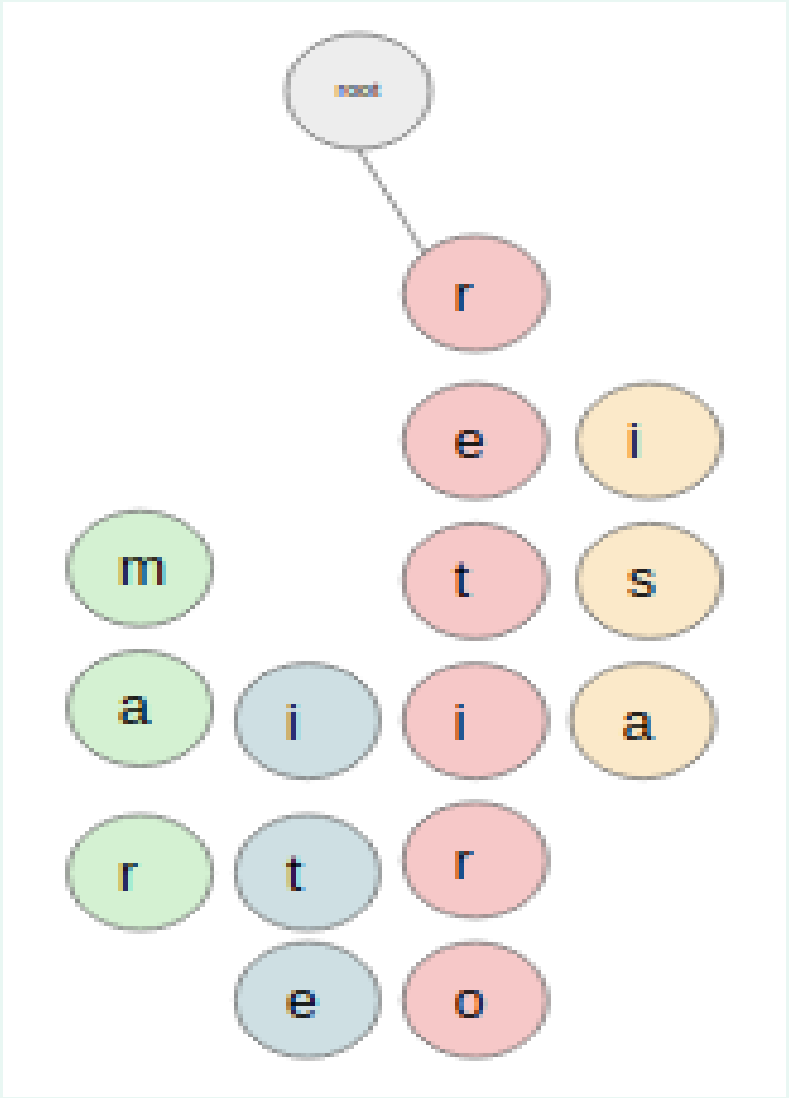
J. La agenda del recuerdo

- Primera idea, por cada query (prefijo), buscamos todas las palabras que coincidan, después de eso, ordenamos los resultados y, ¡voilá!
- Esta solución no es óptima y tiene complejidad O(NQ), siendo N y Q tan altos (> 20000), ¡TLE!

J. La agenda del recuerdo

- Mejor idea, utilizamos un Trie
- En el Trie, la raíz tendrá 26 potenciales hijos, uno por cada letra. Luego, por cada letra, otros 26 potenciales hijos y así sucesivamente
- Cada nodo guarda una referencia a las palabras que tienen ese prefijo, de tal forma, que podemos observar el resultado de cualquier query rápidamente
- El peor caso es tener Q queries que busquen los mismos N resultados. Sin embargo, este caso no se dará nunca debido a las propias restricciones del problema

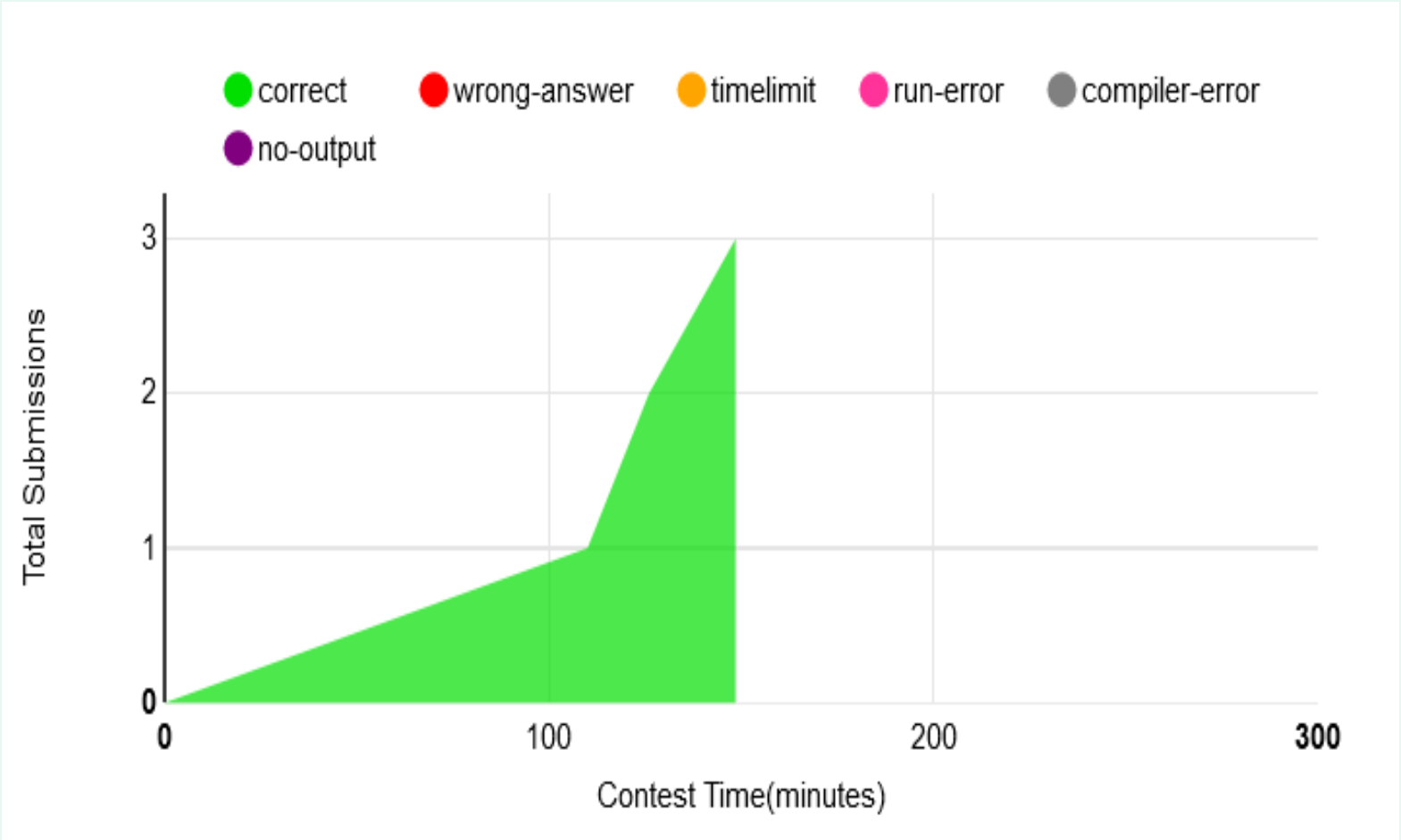
J. La agenda del recuerdo



L. Fiesta Madrileña

Envíos	Válidos	% éxito
3	3	100

L. Fiesta Madrileña



L. Fiesta Madrileña

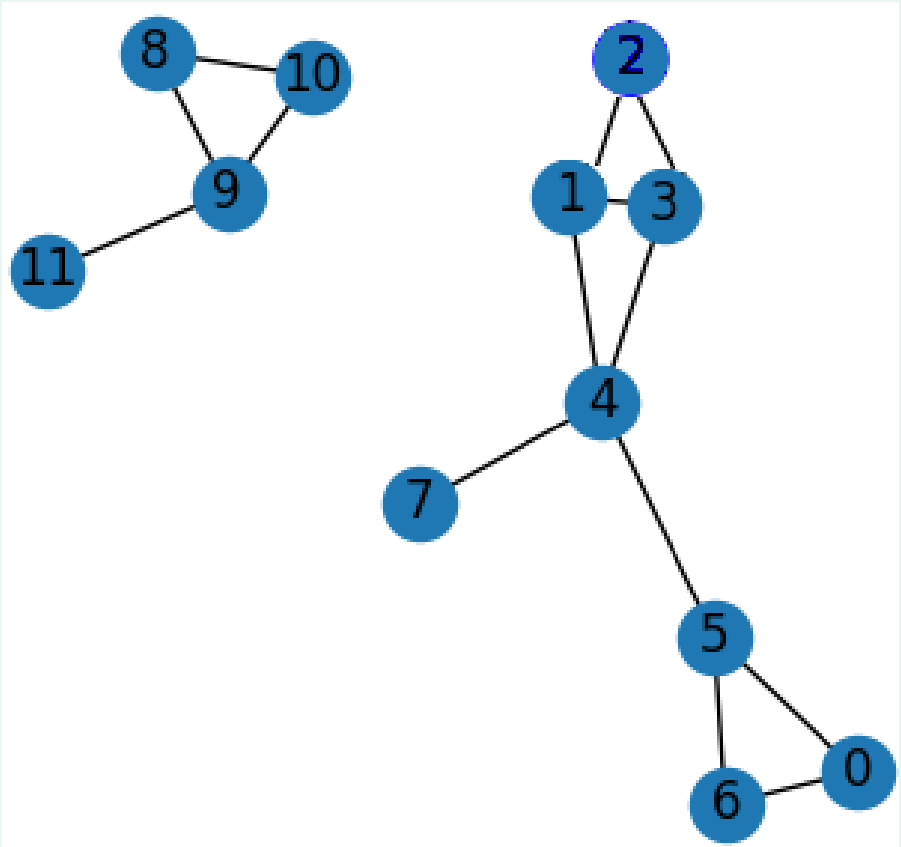
En el enunciado se describe la situación actual donde jóvenes de diferentes países de Europa vienen a Madrid donde las restricciones por COVID-19 son menos restrictivas y pueden organizar fiestas que no son del todo legales.

La Policía Nacional pregunta a los turísticas por las personas con las que van a pasar el fin de semana y vuestra labor es la siguiente:

1. Determinar el número de fiestas que se van a organizar.
2. Determinar el nombre de los turistas que se deben controlar para desarticular cada fiesta.

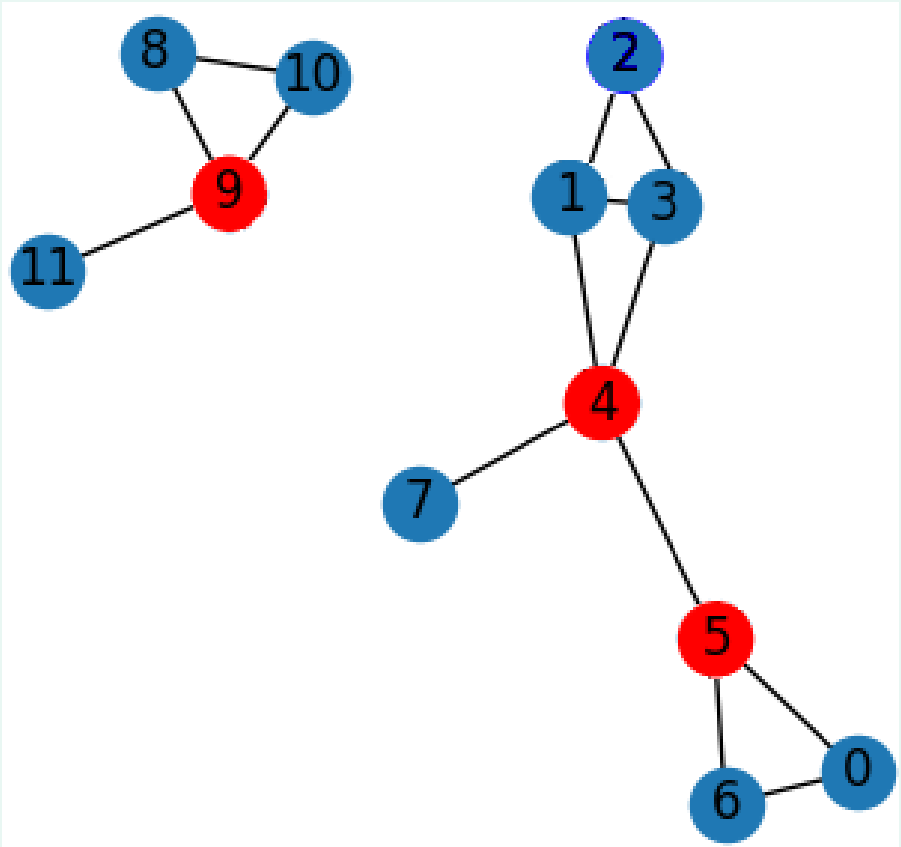
L. Fiesta Madrileña

Si vemos el caso de prueba:



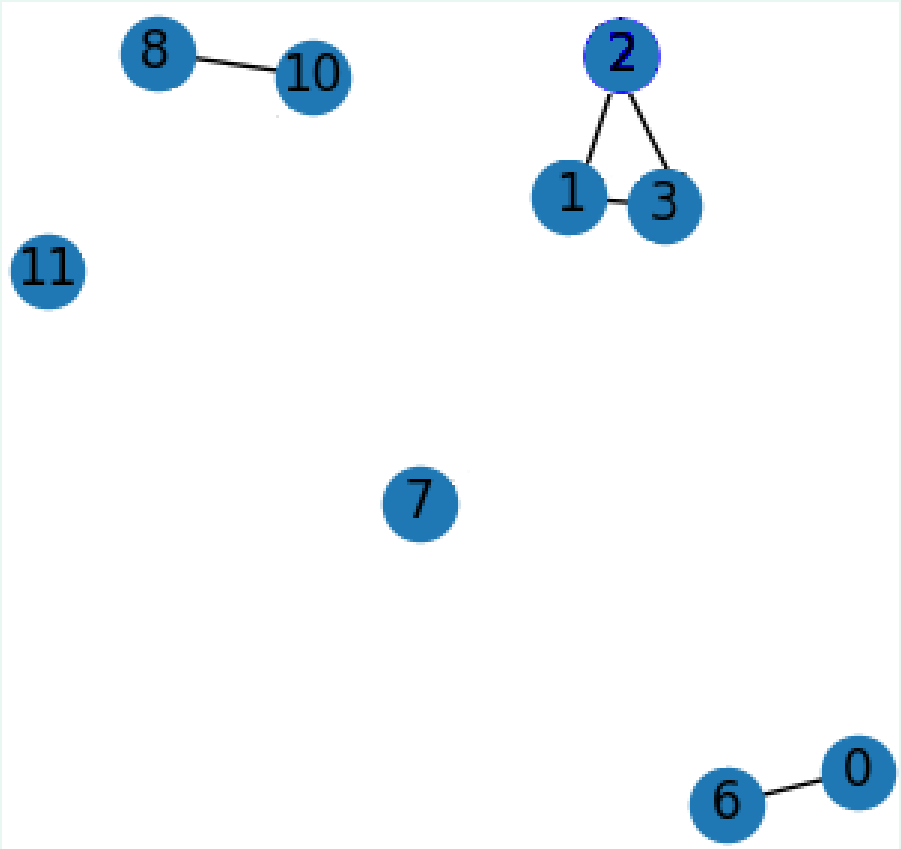
L. Fiesta Madrileña

Si vemos el caso de prueba:



L. Fiesta Madrileña

Si vemos el caso de prueba:



L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.

L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.
   - Cálculo de las componentes conexas con BFS o DFS.

L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.
   - Cálculo de las componentes conexas con BFS o DFS.
2. Determinar el nombre de los turistas que se deben controlar para desarticular cada fiesta.

L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.
   - Cálculo de las componentes conexas con BFS o DFS.
2. Determinar el nombre de los turistas que se deben controlar para desarticular cada fiesta.
   - Extraer los puntos de articulación.

L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.
   - Cálculo de las componentes conexas con BFS o DFS.
2. Determinar el nombre de los turistas que se deben controlar para desarticular cada fiesta.
   - Extraer los puntos de articulación.

Para identificar los puntos de articulación, podríamos pensar en varios enfoques:

L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.
   - Cálculo de las componentes conexas con BFS o DFS.
2. Determinar el nombre de los turistas que se deben controlar para desarticular cada fiesta.
   - Extraer los puntos de articulación.

Para identificar los puntos de articulación, podríamos pensar en varios enfoques:

1. (BIEN, PERO NO EN TIEMPO) Calcular el número de componentes conexas como si los nodos no existieran. $\mathcal{O}(N^2)$.

L. Fiesta Madrileña

Lo que se os pide es:

1. Determinar el número de fiestas que se van a organizar.
   - Cálculo de las componentes conexas con BFS o DFS.
2. Determinar el nombre de los turistas que se deben controlar para desarticular cada fiesta.
   - Extraer los puntos de articulación.

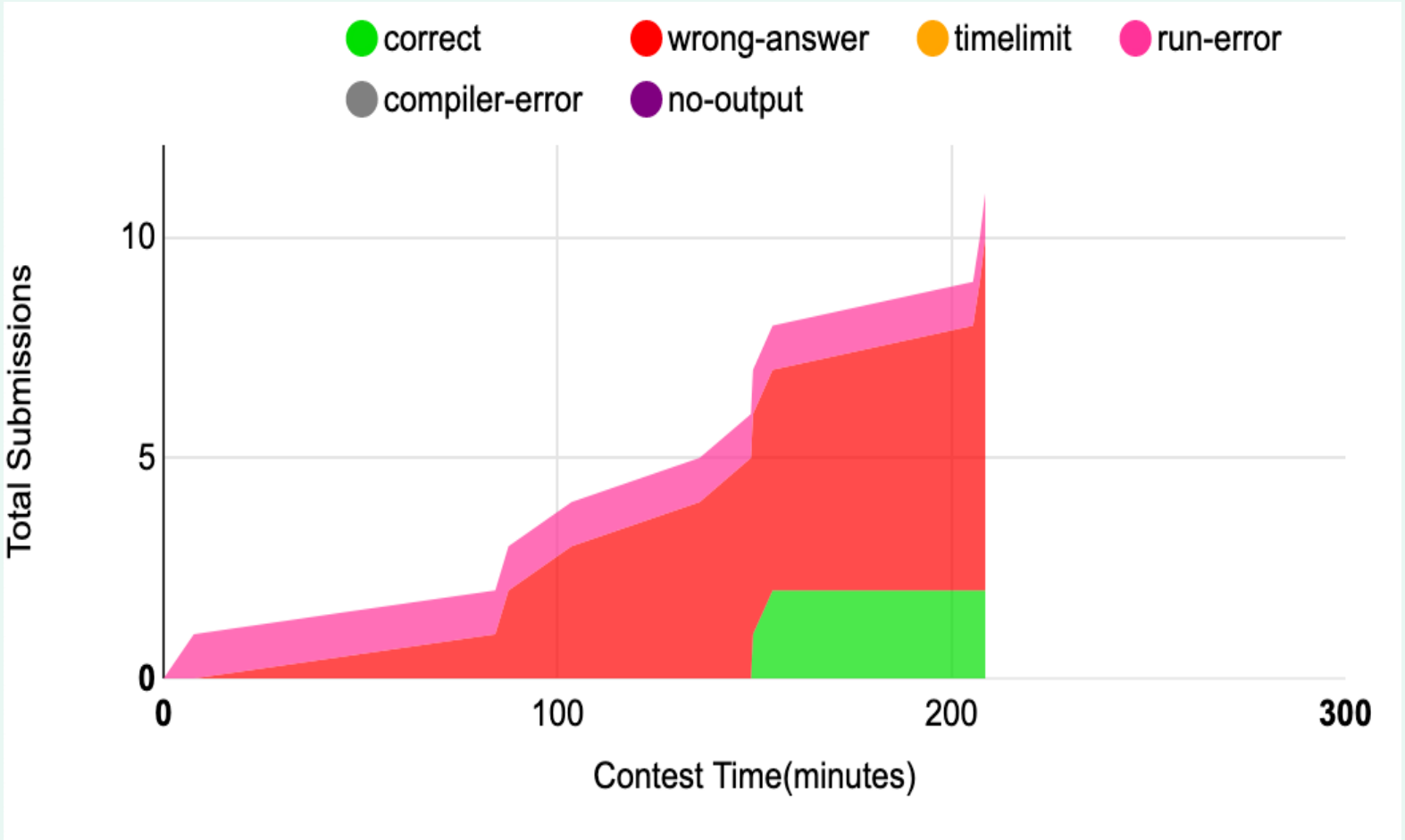
Para identificar los puntos de articulación, podríamos pensar en varios enfoques:

1. (BIEN, PERO NO EN TIEMPO) Calcular el número de componentes conexas como si los nodos no existieran. $\mathcal{O}(N^2)$.
2. Algoritmo de Tarjan: un nodo es un punto de articulación si:
   - Es la raíz del árbol DFS y tiene, al menos, dos hijos.
   - No es la raíz y tiene un hijo, v, tal que no hay ningún vértice en el subarbol con raíz v que tiene una arista de vuelta a uno de los ancestros de u.

C. Bar BBTOtra

Envíos	Válidos	% éxito
11	2	18,18 %

C. Bar BBTOtra



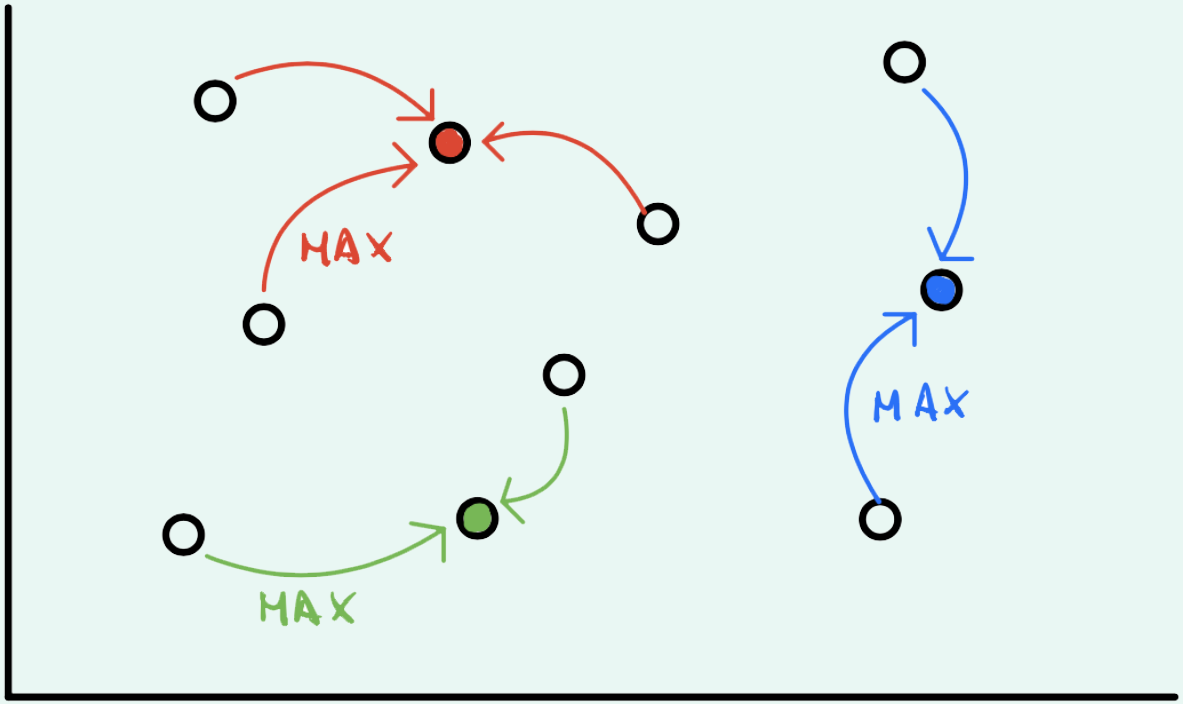
C. Bar BBTOtra

- Queremos abrir franquicias cerca de estudiantes.
- Cada estudiante irá a su bar más cercano.
- En los bares hay aforo ilimitado.
- Queremos que el estudiante más lejano de su bar esté lo más cerca posible.

C. Bar BBTOtra

- Variante del p-center problem $\rightarrow$ Problema de optimización combinatoria.
- Criterio voraz no garantiza optimalidad.
- Solución: backtracking para probar todas las posibles combinaciones.
- Cualquier solución con p elementos no repetidos es factible
- El orden no es importante en las soluciones.
  - $\{1,2,3\} = \{2,3,1\}$
- **ERROR COMÚN:** La distancia proporcionada **NO** tiene por qué ser la mínima.
  - **Solución:** Floyd-Warshall previo a BT.

C. Bar BBTOtra

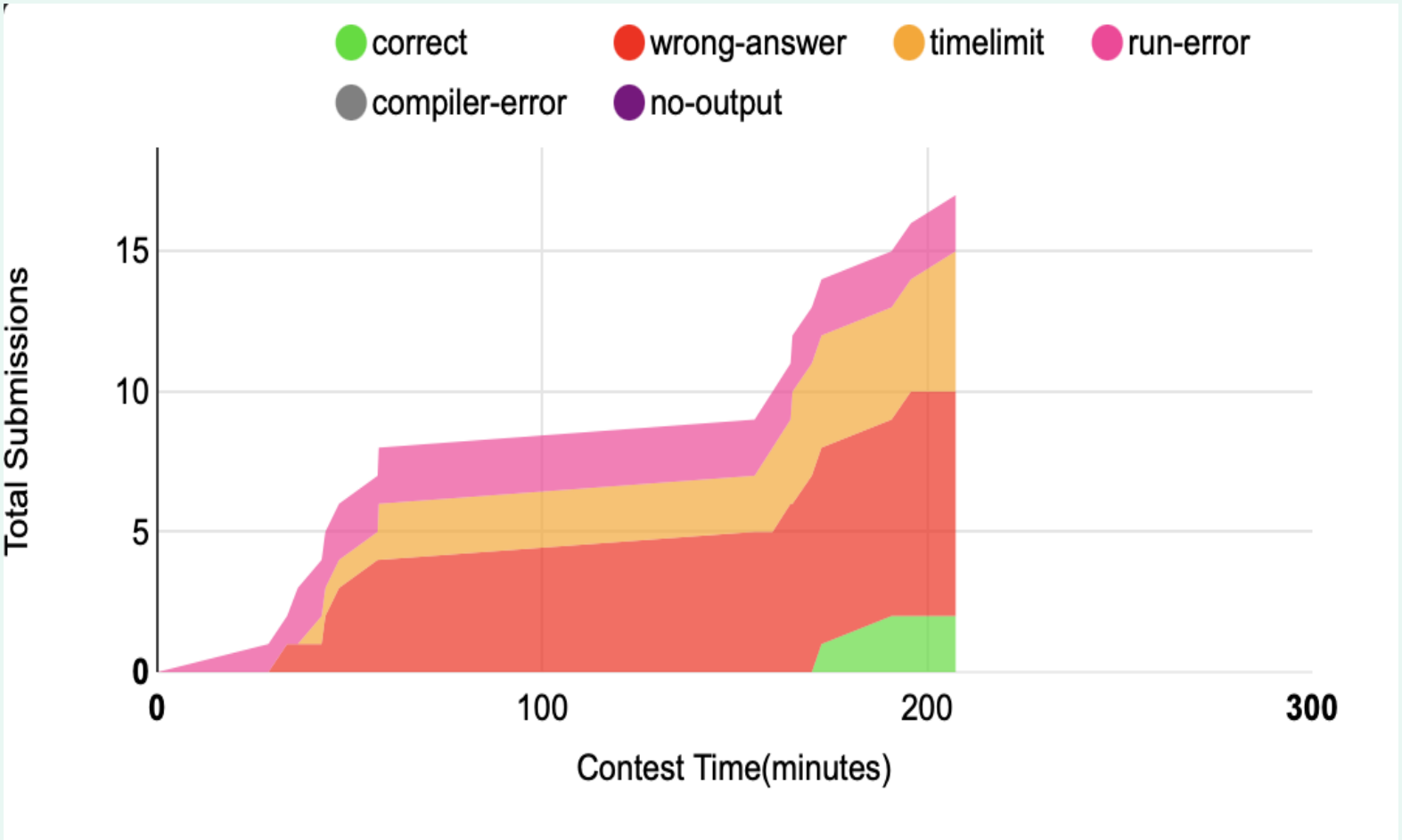


$$f(S) = \max_{i \in (N \setminus S)} \left\{ \min_{j \in S} d_{ij} \right\}$$

A. Dickie y las subsecuencias

Envíos	Válidos	% éxito
17	2	1.76 %

A. Dickie y las subsecuencias



A. Dickie y las subsecuencias

Aunque el enunciado es sencillo de entender, resolver el problema puede que no sea tan sencillo.

- Similar a la solución del Longest Increasing Subsequence utilizando estructuras auxiliares para responder Range Sum Queries
- Observar que el K, es pequeño, su valor máximo es 10
- La idea es utilizar un array 2D de tamaño $K \times N$. La entrada (i, j) contiene cuantas subecuencias de tamaño i, culminan en j. Recordar que el arreglo tiene elementos unicos y $N < 10^5$. Este array 2D representa K Binary Indexed Trees de tamaño N.
- Cada valor de array a_i puede ser mayor que 10^5 por lo tanto, debemos transformar estos numeros a otros que esten en el rango $[1, N]$. Esto se puede hacer copiando el array, ordenando la copia del array y utilizando un map. Llamemos $newVal$ al map: $newVal[copy[i]] = i$
- Luego, el arreglo con valores de $[1, N]$ se podría generar de esta manera: $a[i] = newVal[a[i]]$

A. Dickie y las subsecuencias

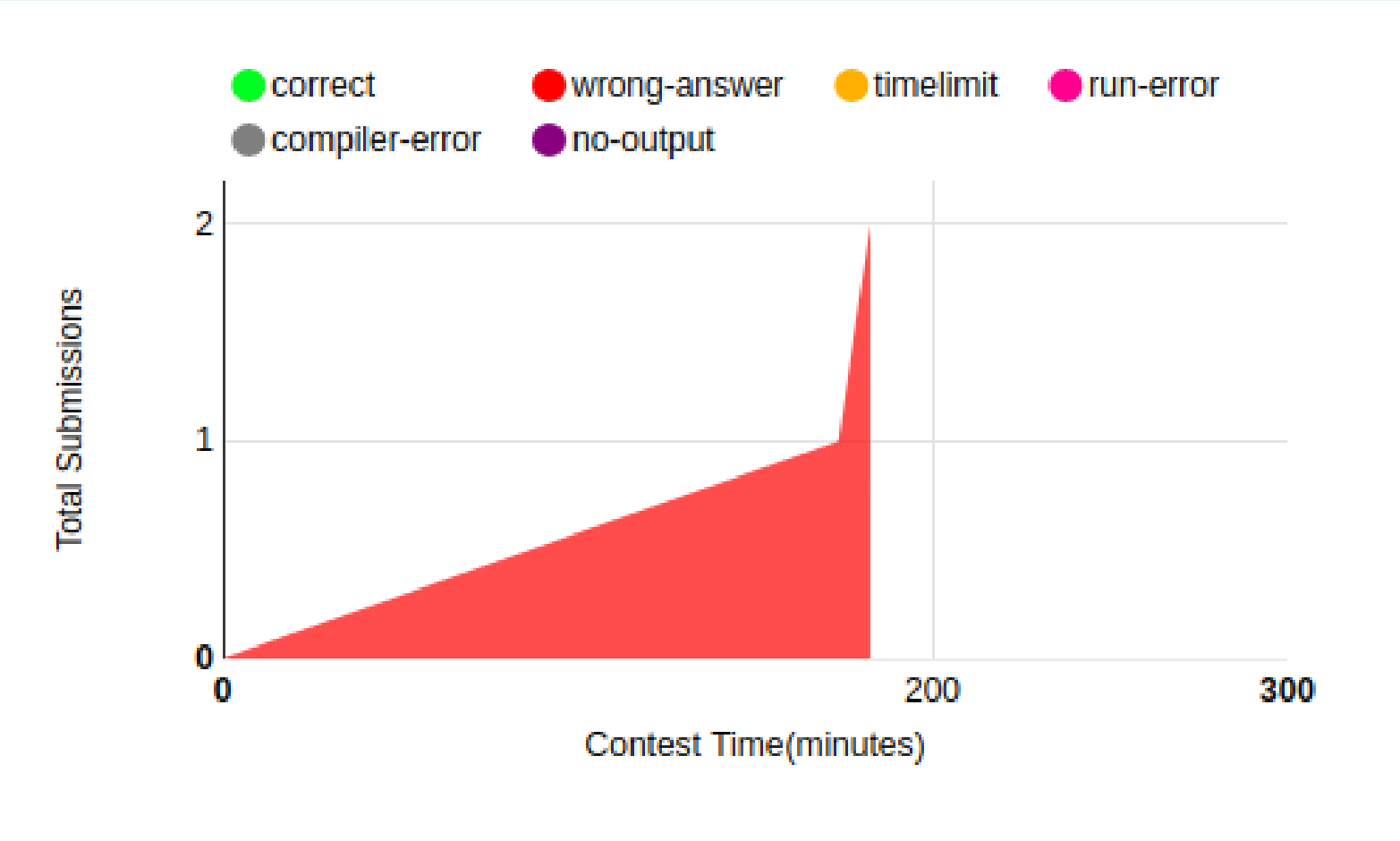
Tomando en cuenta lo anterior, podemos calcular el resultado de la siguiente manera:

```
long long ans = 0;
for(int i = 0; i < n; i++) {
    update(a[i], 1LL, 1);
    for(int j = 2; j <= k; j++) {
        long long s = sum(a[i]-1, j-1);
        update(a[i], s, j);
    }
}
ans = sum(n, k);
```

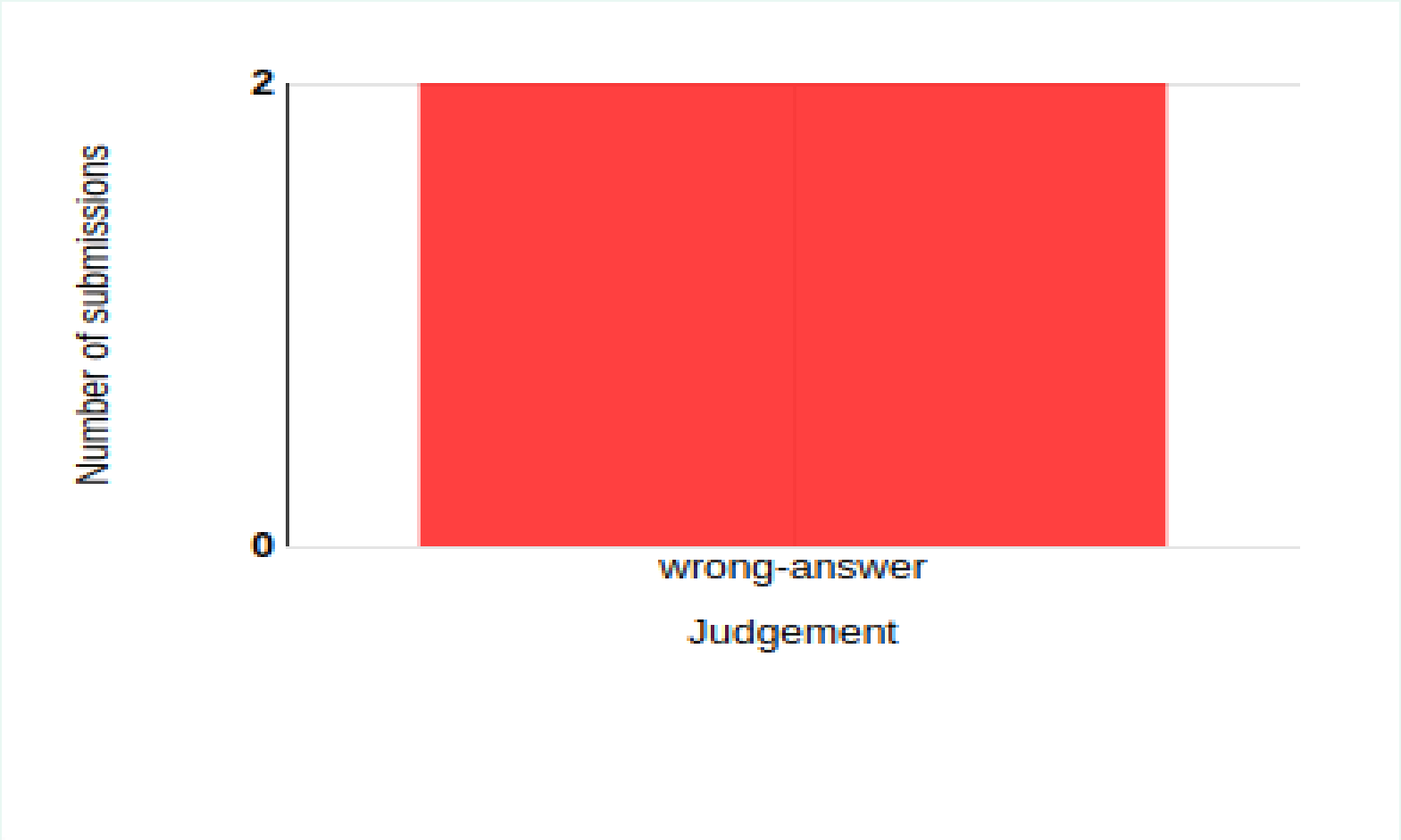
M. La fortaleza

Envíos	Válidos	% éxito
0	0	0 %

M. La fortaleza

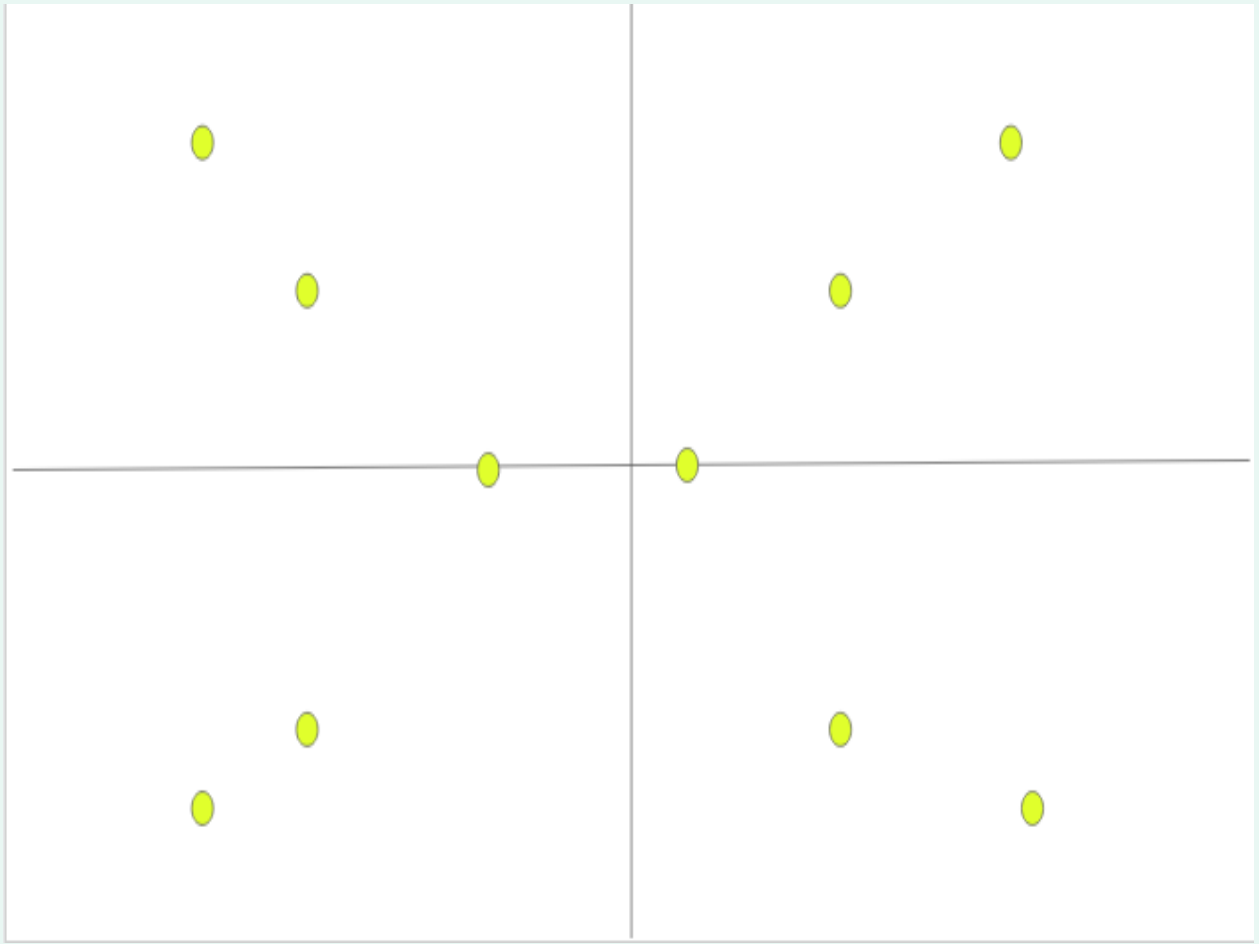


M. La fortaleza



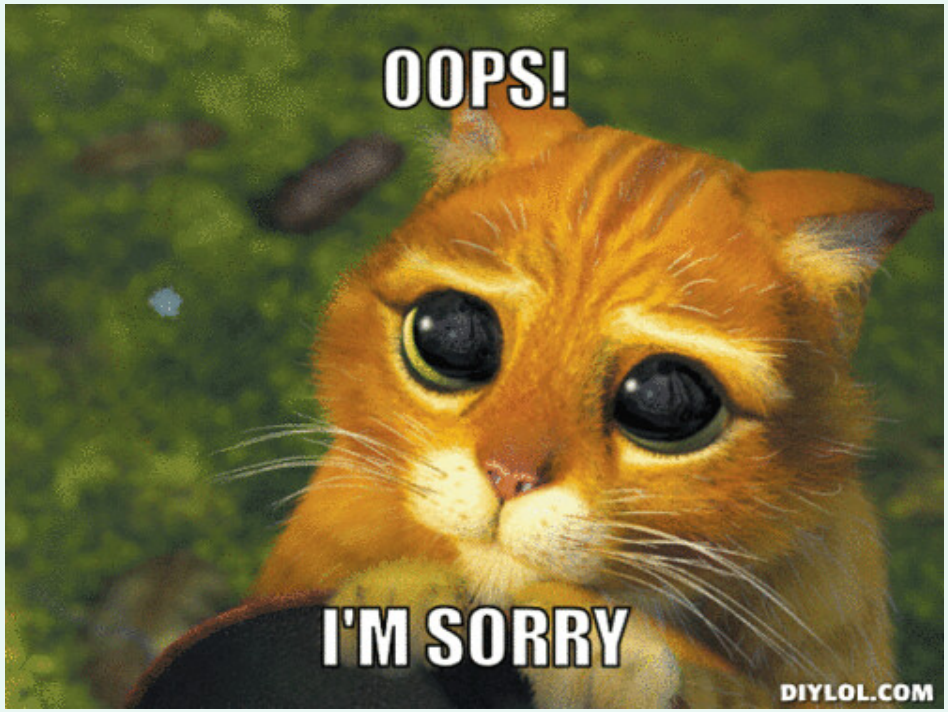
M. La fortaleza

- Espartaco quería construir fortalezas “concéntricas” tomando en consideración los puntos posibles en el mapa, de la forma:



M. La fortaleza

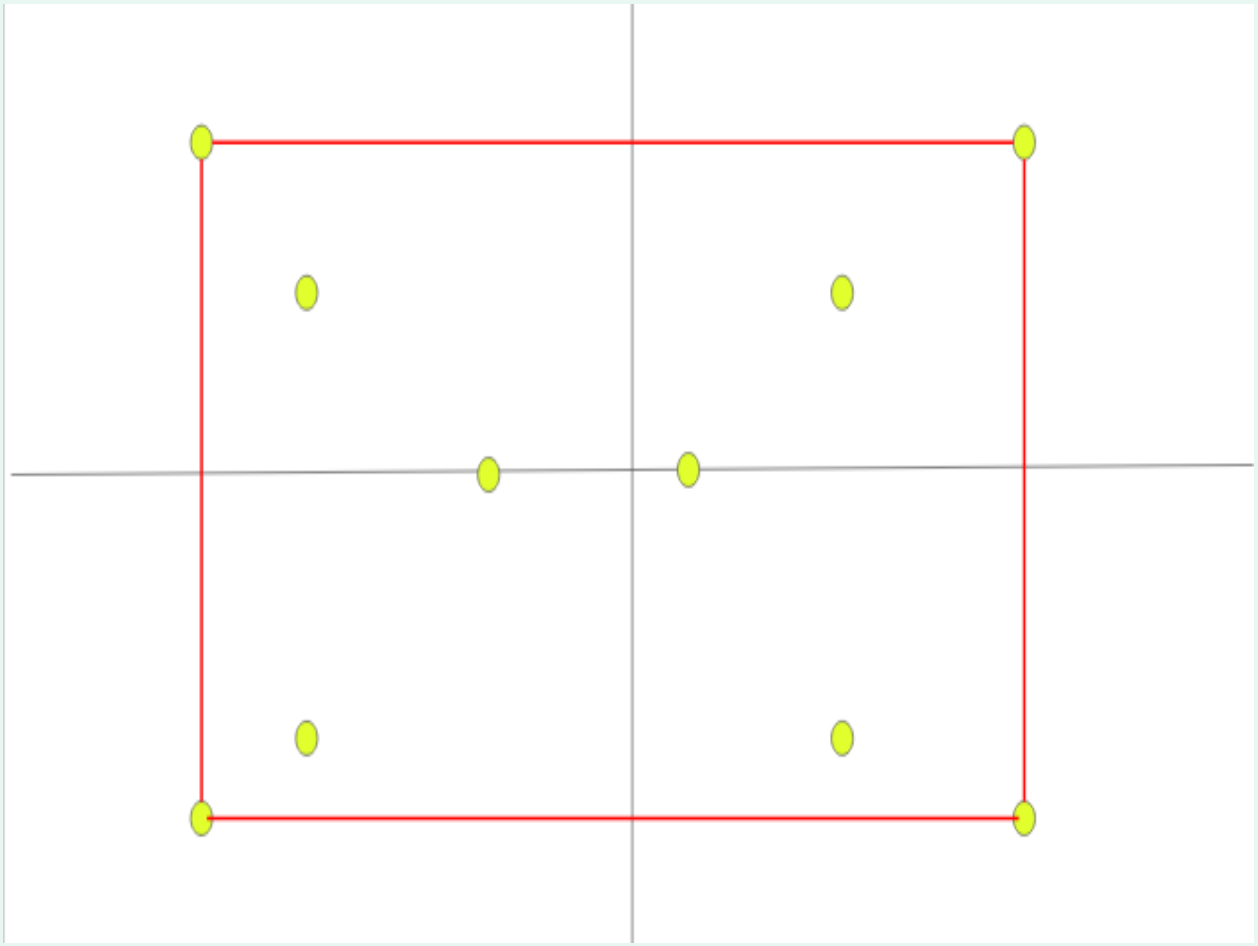
- La definición de concéntrico estaba errónea, ya que un polígono, a pesar de poder circunscribirse y tener un centro, podía no compartir este mismo centro con otro polígono contenido, por lo tanto, la definición correcta era de polígono envolvente.



M. La fortaleza

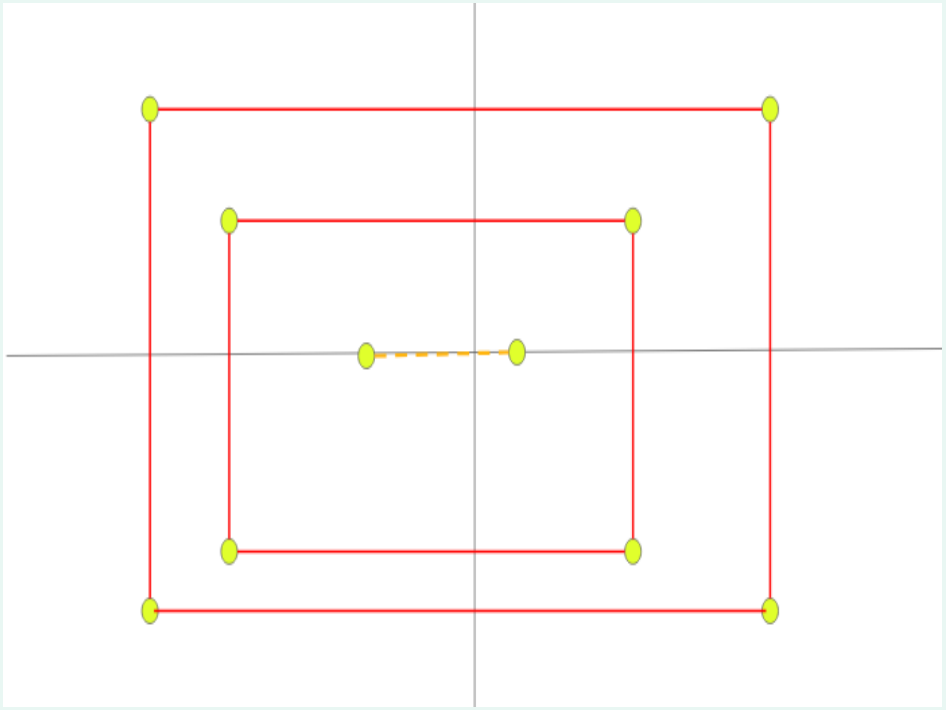
- Teníamos que tener en cuenta que para hacer polígonos envolventes desde el orígen, tambien equivale a hacer polígonos envolventes desde afuera hacia adentro. El algoritmo de cápsula convexa es ideal para esta tarea
- El algoritmo de cápsula convexa sobre X retorna el subconjunto de puntos que contienen todos los posibles segmentos dentro de X

M. La fortaleza



M. La fortaleza

- Finalmente, teníamos que repetir esta operación descartando puntos anteriormente considerados
- Teniendo en cuenta que Espartaco siempre quería una fortaleza (i.e. Un polígono de área > 0)



H. Nos vamos de viaje

Envíos	Válidos	% éxito
0	0	0 %

H. Nos vamos de viaje

Resumen del enunciado:

- Determinar la distancia mínima desde el punto de partida S hasta el punto de llegada F (equivalente al coste que debe pagar)
- Establecer si es posible pagar el coste de manera exacta, con el mínimo número de monedas y en caso de varias soluciones, se escribirá aquella que utilice más billetes de los tipos que aparecen antes en la entrada.

H. Nos vamos de viaje

Aproximaciones incorrectas

- BFS para determinar distancia mínima -> TLE
- BackTracking para establecer el conjunto de monedas -> TLE

Aproximaciones correctas

- Dijkstra para determinar distancia mínima sobre matriz -> OKAY
- BFS para crear lista de adyacencia + Dijkstra sobre lista de adyacencia -> OKAY
- Programación dinámica sobre resultado de Dijkstra -> OKAY

H. Nos vamos de viaje

¿Cómo dar la solución óptima encontrada en un DP? ¡Muy fácil si se tiene el DP listo!
En el libro, Competitive Programming 3 en la página 102 y 103 se explica esto.

- Almacenamos la información del predecesor en cada estado. Si hay más de un predecesor óptimo y tenemos que dar salida a todas las soluciones óptimas, se almacena esos predecesores en una lista. Una vez que tenemos el estado final óptimo, backtracking desde el estado final óptimo y seguir la(s) transición(es) óptima(s) registrada(s) en cada estado hasta que lleguemos a uno de los casos base.
- Utilizando el código DP top down sería misma estructura excepto que utiliza los valores almacenados en la tabla que memoriza para reconstruir la solución.

H. Nos vamos de viaje

Paso 1

```
int DP(int moneda, int restante){
    ...
}

void print_DP(int moneda, int restante) {
    //Funcion que devuelve void
    ...
}
```

H. Nos vamos de viaje

Paso 2

```
int DP(int moneda, int restante){
    if(restante==0)
        return 0; //Caso base 1
    if(restante<0 || moneda>=casos)
        return INF; //Caso base 2
    ...
}

void print_DP(int moneda, int restante) {
    //Funcion que devuelve void
    if(restante==0 || restante<0 || moneda>=casos)
        return; //Mismos casos bases
    ...
}
```

H. Nos vamos de viaje

```
int DP(int moneda, int restante){
    if(restante==0)
        return 0; //Caso base 1
    if(restante<0 || moneda>=casos)
        return INF; //Caso base 2
    int &best = memo[moneda][restante];
    //Memorizar este estado
    if(best!=-1)
        return best;
        //No volver a calcular si se tiene el valor
    int check = INF;
    for(int i=0; i<=cantidad[moneda];i++){
        int resValue = restante-coins[moneda]*i;
        int value = DP(moneda+1,resValue)+i;
        check = min(check,value);
    }
    return best=check;
}
```

H. Nos vamos de viaje

```
void print_DP(int moneda, int restante) {
    //Funcion que devuelve void
    if(restante==0 || restante<0 || moneda>=casos)
        return; //Mismos casos bases
    for(int i=cantidad[moneda]; i>=0;i--) {
        int resValue = restante-coins[moneda]*i;
        int value = DP(moneda+1,resValue)+i;
        if (value == memo[moneda][restante]) {
           //Que modelo da la solucion optima?
            print_DP(moneda+1,resValue);
            //Imprimela
            for(int j=0; j<i;j++) sol[moneda]--;
            break;
            //No visitar otros estados
        }
    }
}
```