Soluciones Concurso

11/4

Con “C” de Concurso: Problemas que empiezan por la letra “C”

¡Esperamos que os hayan gustado los problemas!

Por: Raúl Martín, Iván Martín, Jakub Jan Luzcyn, Isaac Lozano

Escuela Técnica Superior de Ingeniería Informática (URJC)

@urjc_cp @etsii_urjc

Estadísticas de los problemas

*antes del Freeze

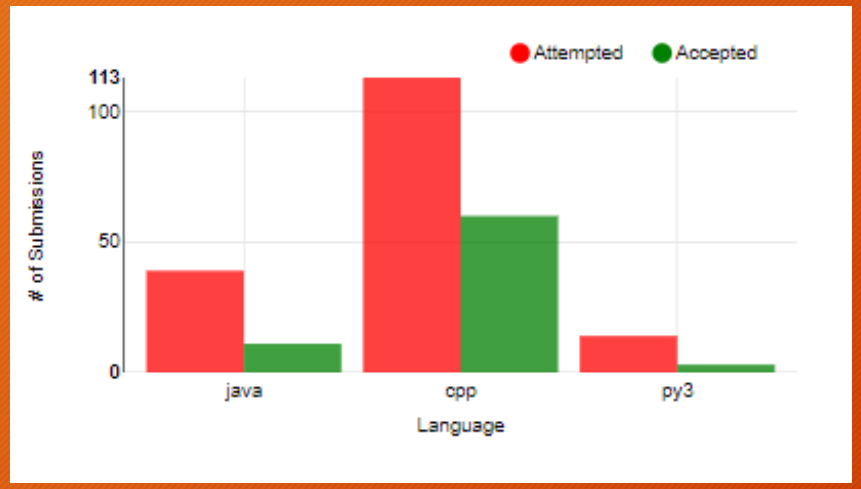
Problema	Primer AC No URJC	Primer AC URJC
A: Carrera en Pekín Express	David López (UCM) [14’]	Jorge Sendarrubias (URJC) [47’]
B: Coleccionando Cartas	?	?
C: Comprendiendo el Código	Alberto Maurel (UCM) [77’]	-
D: Comunidades en Cuarentena	?	-
E: Cajas de Juegos Desordenados	Alberto Maurel (UCM) [19’]	-
F: Cero Virus	Eduardo Santos (UPV) [11’]	ShuXiang Gao (URJC) [27’]
G: Capturando Humanos	?	-
H: Compras Extremas	Jinqing Cai (UCM) [9’]	Francisco Tórtola (URJC) [8’]

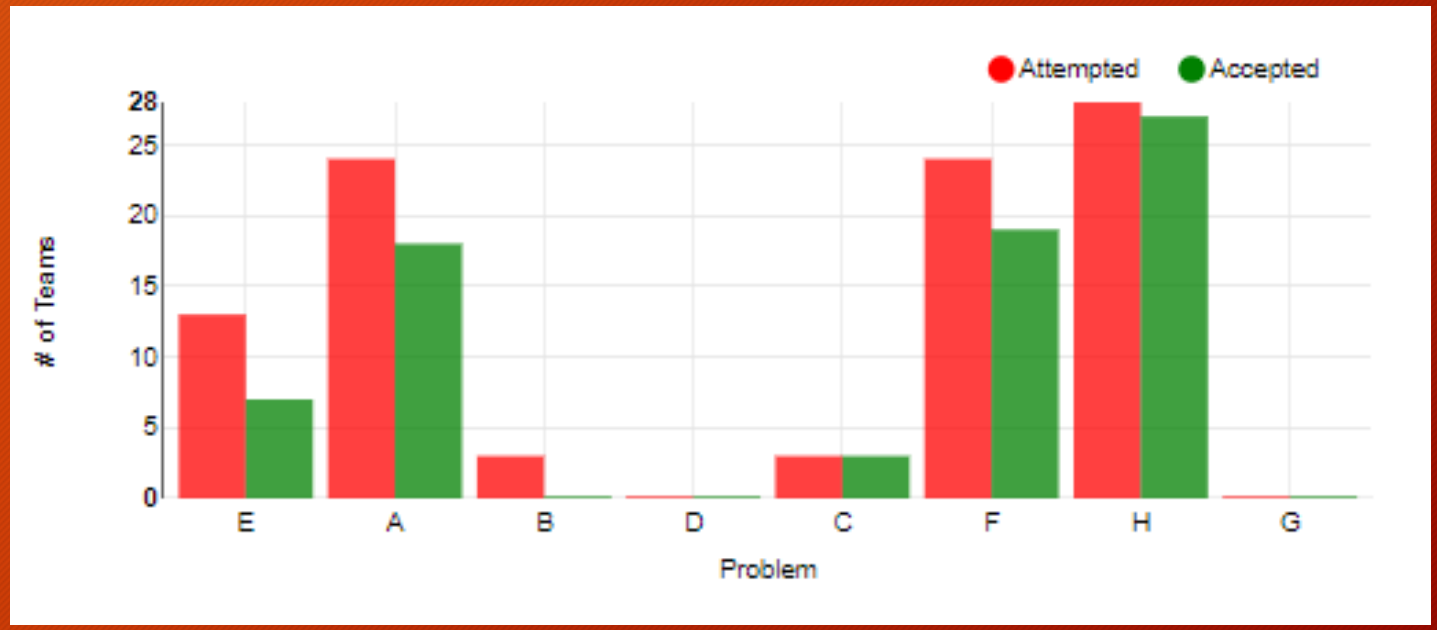
Se anotan con ‘?’ aquellos problemas con envíos durante el freeze

Estadísticas de los problemas

Problema	Categorías	Casos de prueba
A: Carrera en *Pekín Express*	Grafo ponderado, Camino Mínimo (Dijkstra)	18 (15,0 MB)
B: Coleccionando Cartas	Cola de Prioridad	19 (10,3 MB)
C: Comprendiendo el código	Bucles, Ad-Hoc, Time-Waster	21 (2,5 MB)
D: Comunidades en Cuarentena	Árboles, Recorrido en Profundidad (DFS)	14 (2,3 MB)
E: Cajas de Juegos Desordenados	Pensar, Listas Circulares	8 (26,2 MB)
F: Cero Virus	Grafos, Union-Find	4 (3,2 MB)
G: Capturando Humanos	Grafo dirigido, Camino Mínimo (Dijkstra), Greedy, Grafo Inverso	10 (35,2 MB)
H: Compras Extremas	Bucles, Ad-Hoc	6 (73 KB)

Estadísticas de los problemas





Estadísticas antes del Freeze

Problema H

Compras Extremas

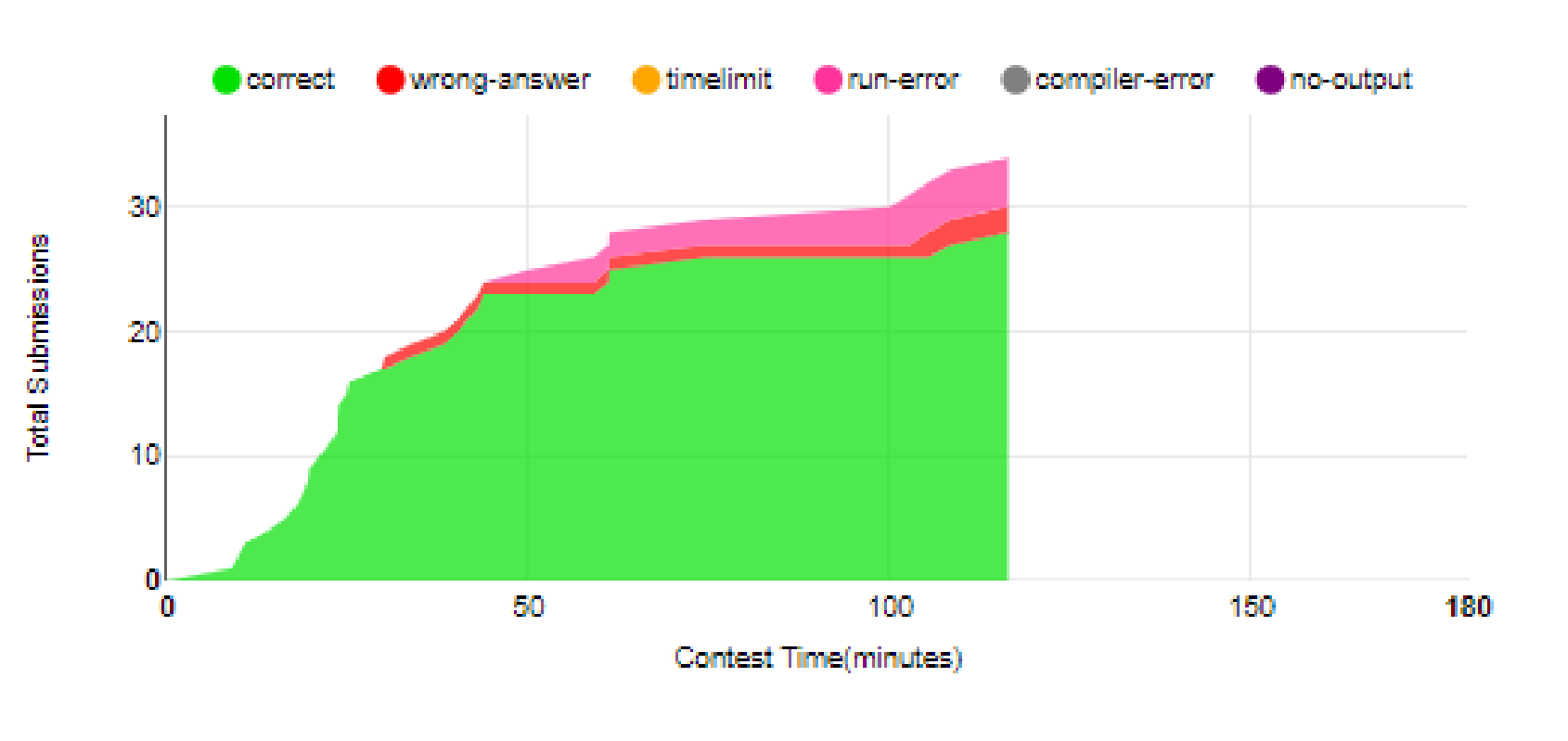
Autor: Isaac Lozano

Primer AC no URJC: Jinqing Cai (UCM) [9’]

Primer AC URJC: Francisco Tórtola (URJC) [8’]

Compras Extremas

H



Compras Extremas

H

¡El problema fácil del concurso!

Necesitamos saber quién es el que ha hecho la compra más divertida.

Para ello, solo tenemos que leer uno a uno los elementos que ha comprado cada persona y sumarlos para saber como de divertida ha sido su compra.

Finalmente, imprimimos aquel que haya tenido la compra más divertida.

Problema C

Comprendiendo el Código

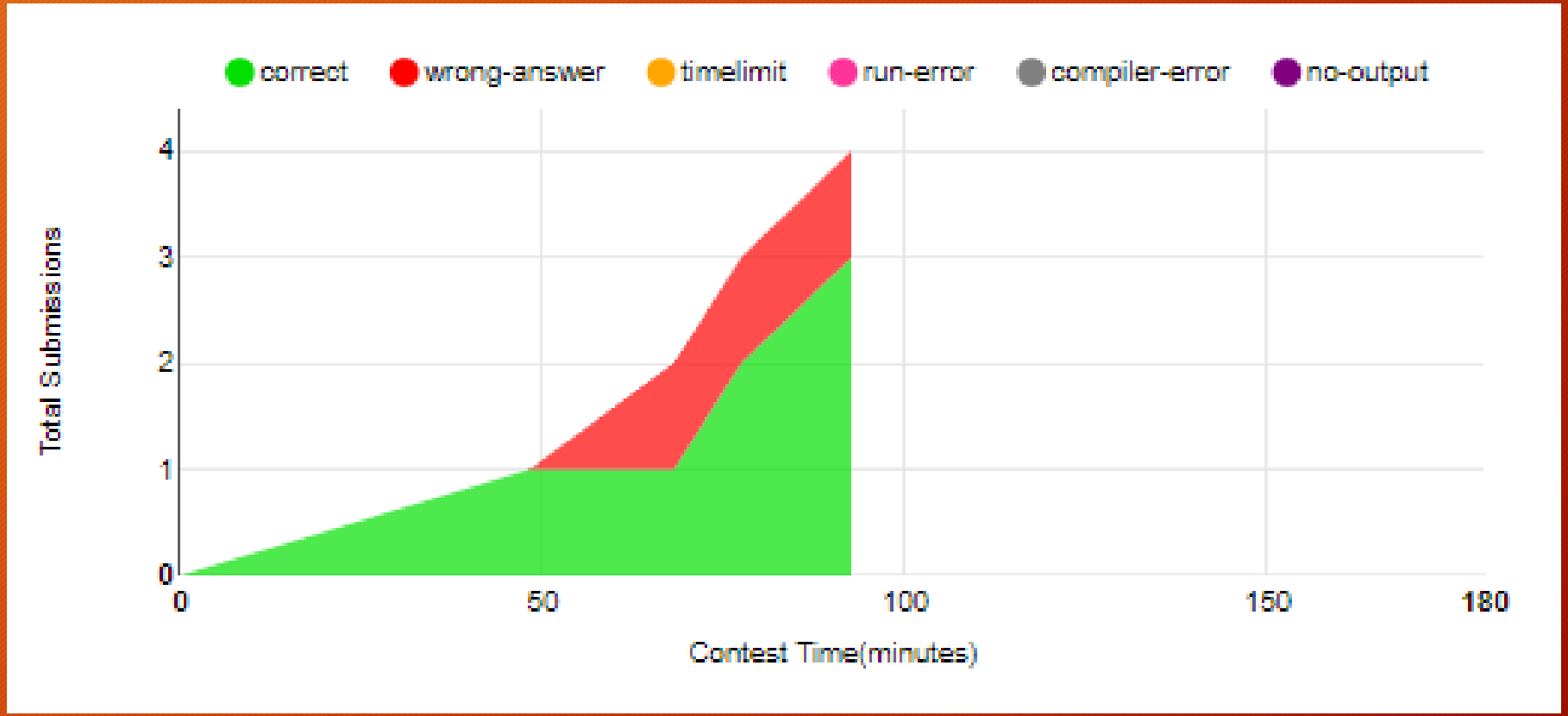
Autor: Jakub Jan Luczyn

Primer AC no URJC: Alberto Maurel (UCM) [77’]

Primer AC URJC: N/A

Comprendiendo el Código

C



Comprendiendo el Código

C

En este problema se pide que consigamos leer el código de una forma muy específica:

- Siempre se debe hacer de izquierda a derecha
- Debemos pasar por todos los pares verticales
- No podemos pasar por el mismo elemento dos veces

Es un problema algo laborioso de programar, pero es muy sencillo una vez se entiende.

Comprendiendo el Código

C

Como podemos empezar por arriba o por abajo, esto nos deja tan solo dos posibilidades para realizar la lectura:

Comenzar por arriba

Comenzar por abajo

Para conseguir la solución, aunque sea un poco laborioso, tan solo tenemos que calcular ambas e imprimir el mínimo. Un error común es calcular primero el camino horizontal y luego el vertical. ¡Error! También hay que calcular las transiciones de las esquinas del camino.

Problema F

Cero Virus

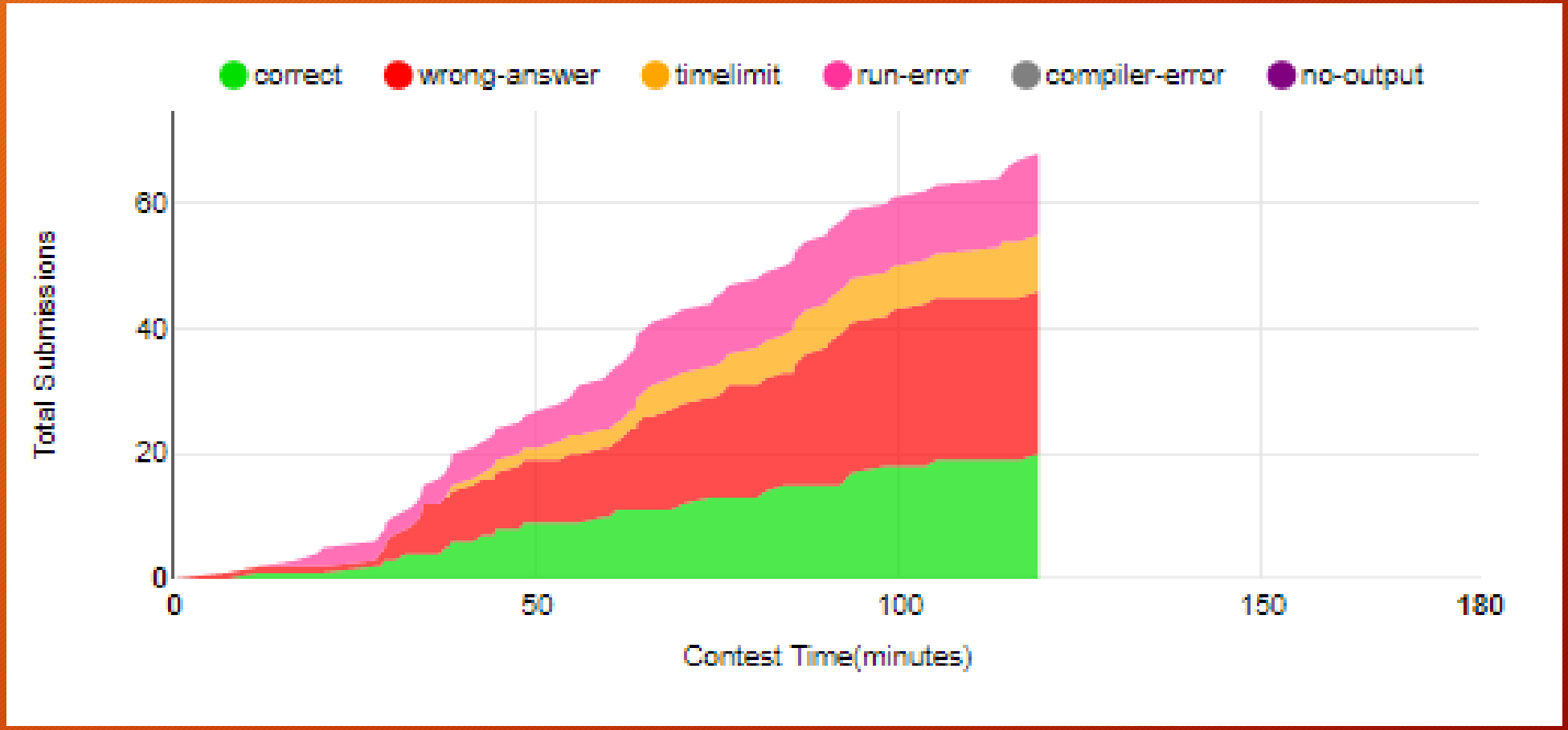
Autor: Raúl Martín Santamaría

Primer AC no URJC: Eduardo Santos (UPV) [11’]

Primer AC URJC: ShuXiang Gao (URJC) [27’]

Cero Virus

F


correct
wrong-answer
timelimit
run-error
compiler-error
no-output
Total Submissions
60
40
20
0
0
50
100
150
180
Contest Time(minutes)

Cero Virus

F

Problema típico de Union-Find.

Se van descubriendo conexiones entre personas y personas infectadas según vamos leyendo la entrada.

Exploraciones del grafo propagando infección --> TLE

Dos soluciones:

- Array de infectados, donde marcamos la raíz de la componente como infectada

- Nodo imaginario I, cuando un grupo se sabe que esta infectado se une a dicho grupo I.

Cero Virus

F

Errores más comunes:

- La persona 0 era válida
- No utilizar DisjointSets (UnionFind)
- No propagar la infección al añadir nuevas conexiones (Al hacer unión el nuevo grupo esta infectado si cualquiera de los dos antiguos lo esta)

Problema B

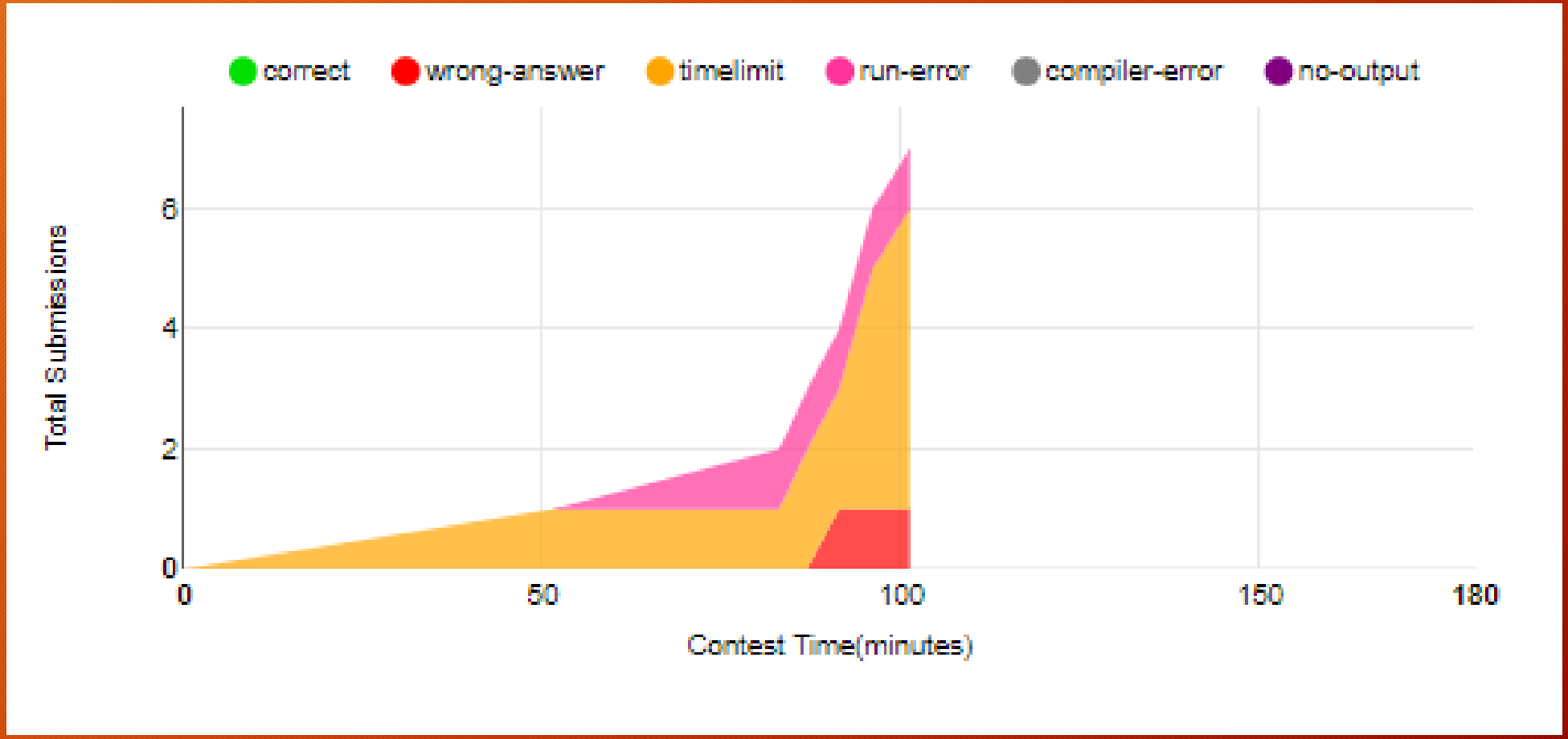
Coleccionando Cartas

Autor: Iván Martín de San Lázaro

Sin ACs antes del Freeze

Coleccionando Cartas

B



Coleccionando Cartas

B

Manolo quiere completar su colección de cartas, pero no a cualquier precio. Lo importante es fijarnos en que cada mes comprará siempre las cartas más baratas, y parará de comprar cartas cuando no tenga más dinero.

Sin embargo, todos los meses llegarán precios nuevos de cartas, y los precios anteriores también hay que mantenerlos por si el mes siguiente pudiéramos comprar más cartas.

Por tanto, necesitamos una estructura que:

- Me diga qué carta hay más barata cada momento.
- Pueda añadir nuevas cartas y las reordene por precio.

Coleccionando Cartas

B

¡¡Cola de Prioridad!!

Eso es lo que estamos buscando. Metemos todas las cartas con su precio inicial (Y las fluctuaciones del primer mes) en la cola de prioridad, y continuamos mes por mes hasta completar la colección o gastar el dinero del último mes.

Coleccionando Cartas

B

Cuando cojamos un elemento de la cola de prioridad pueden pasar 3 cosas:

- No tenemos la carta y sí dinero para comprarla: Siguiendo el razonamiento de Manolo, la compramos seguro (Eliminandola de la cola de prioridad)
- Ya tenemos la carta: La eliminamos de la cola y continuamos
- No tener la carta, ni dinero para comprarla: Todas las demás cartas (Por la cola de prioridad) van a costar lo mismo o más = Continuamos con el siguiente mes (¡Y no guardamos el dinero del mes anterior! => WA)

Coleccionando Cartas

B

Notas:

Cuidado con la descripción de la salida. En este caso, se pedía que, si una colección no se completaba, se imprimieran los números de colección de las cartas que nos faltaban en orden **ascendente**

También se indicaba que en caso de que dos cartas tuvieran el mismo precio, se cogería primero la de menor índice

(Estas dos cosas han causado WA)

Problema E

Juegos Desordenados

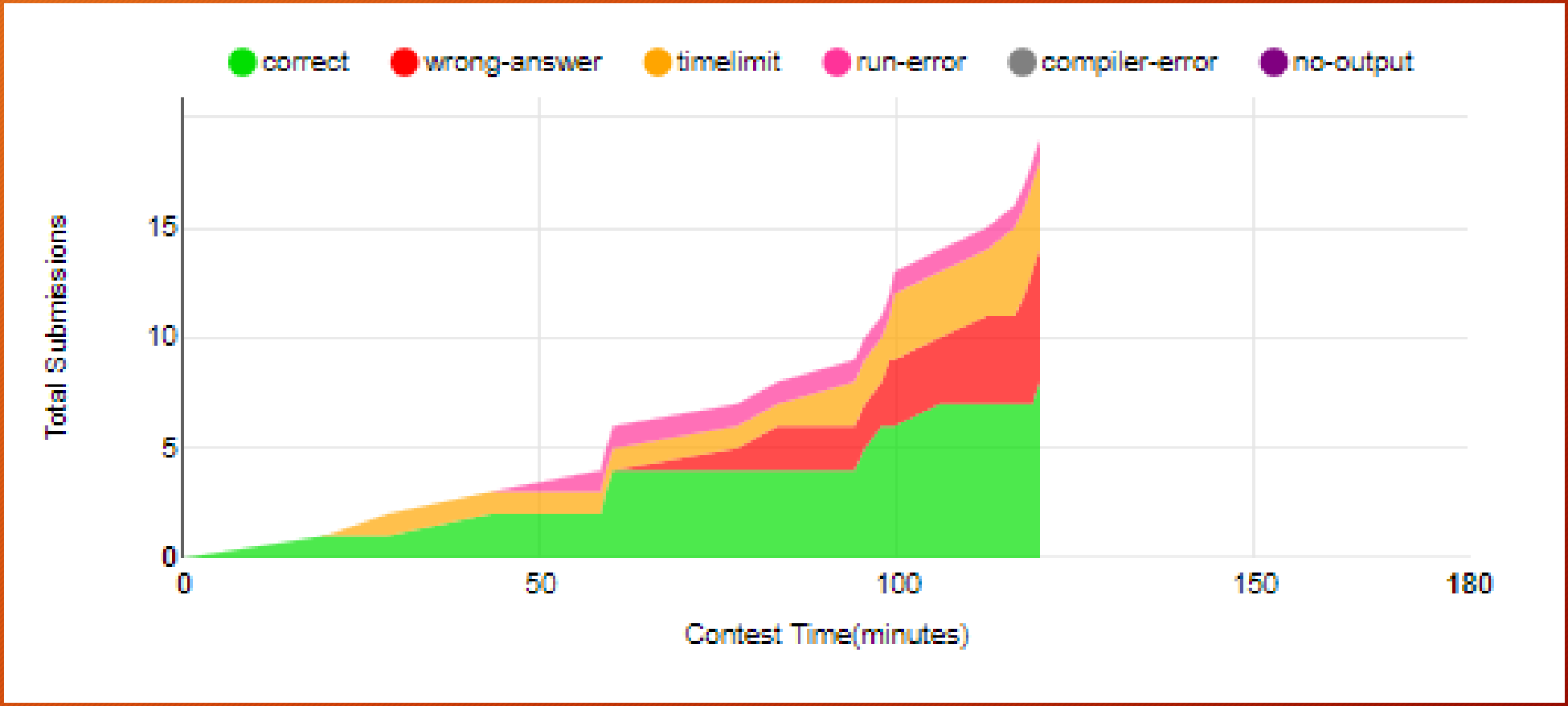
Autor: Iván Martín de San Lázaro

Primer AC no URJC: Alberto Maurel (UCM) [19’]

Primer AC URJC: N/A

Juegos Desordenados

E



Juegos Desordenados

E

Miremos la *estrategia* que sigue David: Caja 0 contiene Juego 1, así que abre caja 1, encontrando juego 3, abriendo caja 3… Así hasta encontrar juego 0. Esto se modela como una (O, mejor dicho, varias, según el caso) listas circulares.

El objetivo es mirar si para cada juego (de 0 a J) podemos encontrar el disco ese juego, comenzando en la caja J. Si todos los juegos los podemos encontrar abriendo como mucho P cajas, imprimimos “SI” y si no, “NO”

Veamos un ejemplo, ya que dibujar el problema ayuda mucho a entenderlo:

Juegos Desordenados

E

Veamos el siguiente ejemplo, buscando el juego 0:

0 -> 1 -> 3 -> 6 -> 4 -> 0

En total, hemos abierto 5 cajas.

Juegos Desordenados

E

0 requiere 5 cajas…

¿Y el juego 2?

2 -> 5 -> 7 -> 2

El juego 2 requiere abrir 3 cajas.

Continuando así para todos los juegos…

Juegos Desordenados

E

Analizando todos los casos, encontramos:

0->1->3->6->4->0	5
1->3->6->4->0->1	5
2->5->7->2	3
3->6->4->0->1->3	5
4->0->1->3->6->4	5
5->7->2->5	3
6->4->0->1->3->6	5
7->2->5->7	3

Juegos Desordenados

E

Para cada juego, tendremos que realizar siempre exactamente tantos pasos como el tamaño de la lista circular donde esté contenido…

Comprobar los N juegos mirando el tamaño de la lista en la que están contenidos es una posible solución (Que, aunque muy justa, entra en tiempo)

Juegos Desordenados

E

¡Pero hay una solución más fácil aún!

En ningún momento nos piden **qué** juego es con el que nos podrían quitar la consola. Solo debemos saber si existe o no algún caso así.

No hace falta mirar caso por caso. Nos basta con mirar la longitud de la lista más grande y compararla con la paciencia de la madre de David.

Si la paciencia es mayor o igual que esa longitud, estamos salvados. En caso contrario, hay uno o más juegos que podríamos no encontrar (De hecho, si la lista más grande tiene tamaño N, hay exactamente N juegos que no podremos encontrar)

Problema A

Carrera en *Pekín Express*

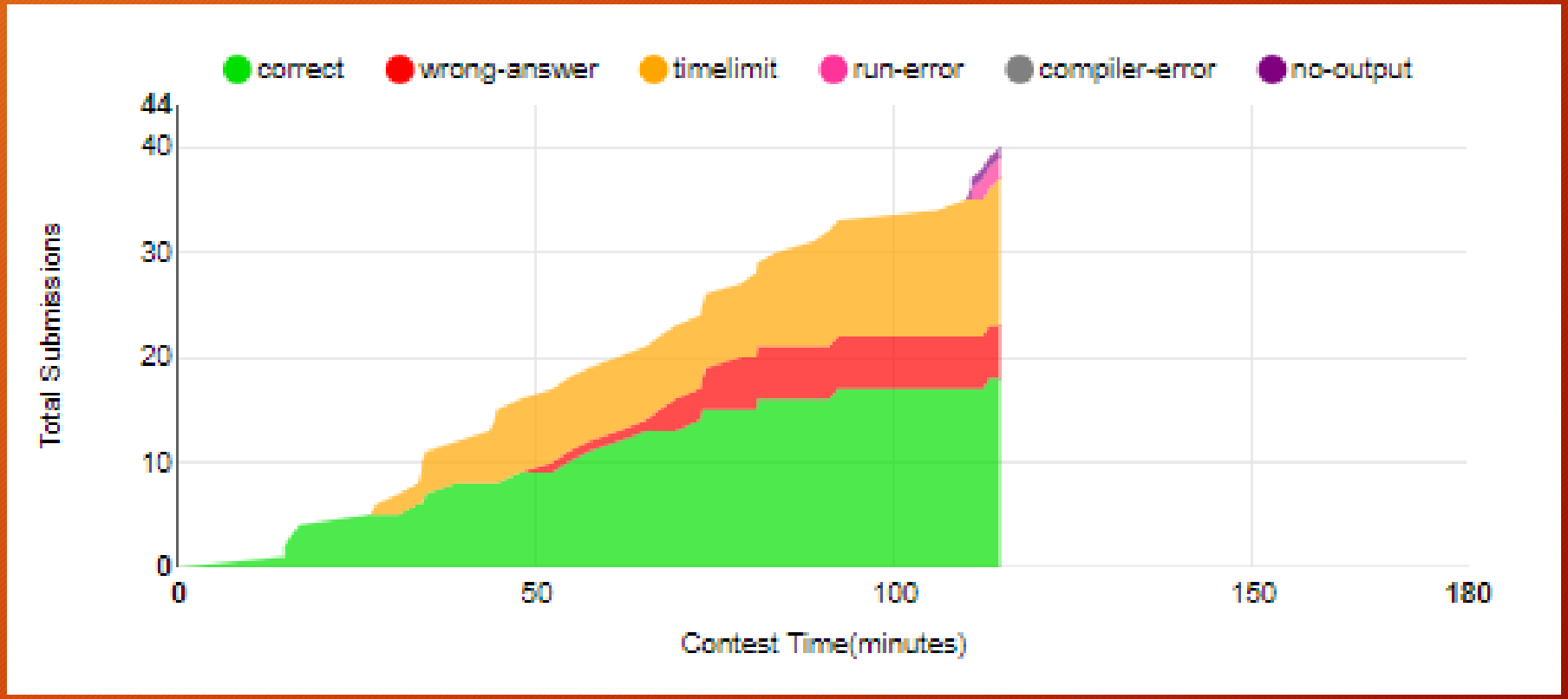
Autor: Iván Martín de San Lázaro

Primer AC no URJC: David López (UCM) [14’]

Primer AC URJC: Jorge Sendarrubias (URJC) [47’]

Carrera en *Pekín Express*

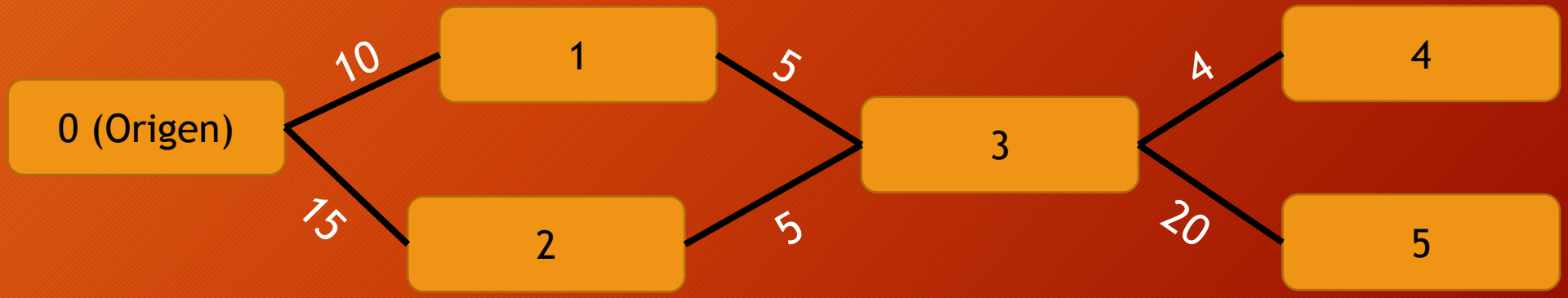
A



Carrera en *Pekín Express*

A

El punto clave del problema es la última frase “Podemos asumir que cada equipo ha cogido la mejor ruta posible a su destino”. Como la ciudad es un grafo (ponderado) y tenemos que conseguir desde el mismo punto de origen, la mejor ruta a muchos puntos de la ciudad… Usaremos el algoritmo de Dijkstra



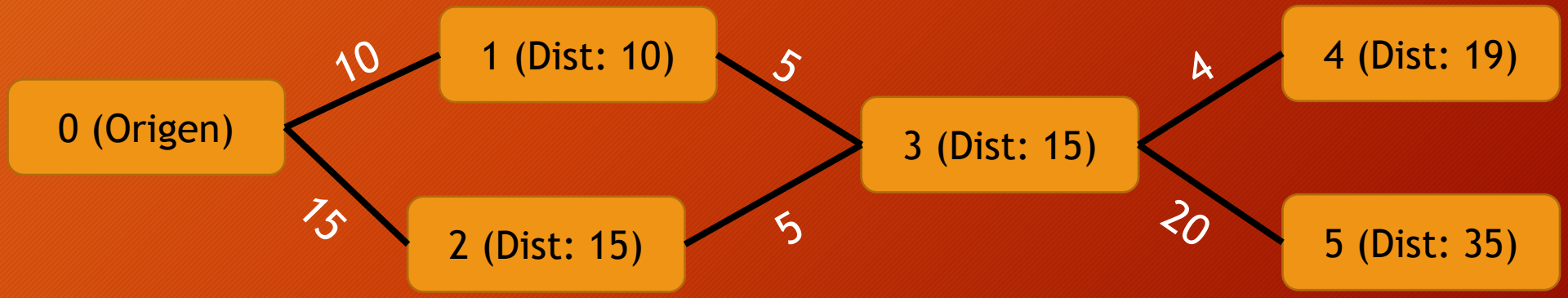
Grafo del ejemplo

Carrera en *Pekín Express*

A

En este caso, la ruta mínima desde 0 al resto de puntos es la de la imagen.

Como mínimo, cada equipo tardará este tiempo en llegar al punto de destino que le haya tocado.

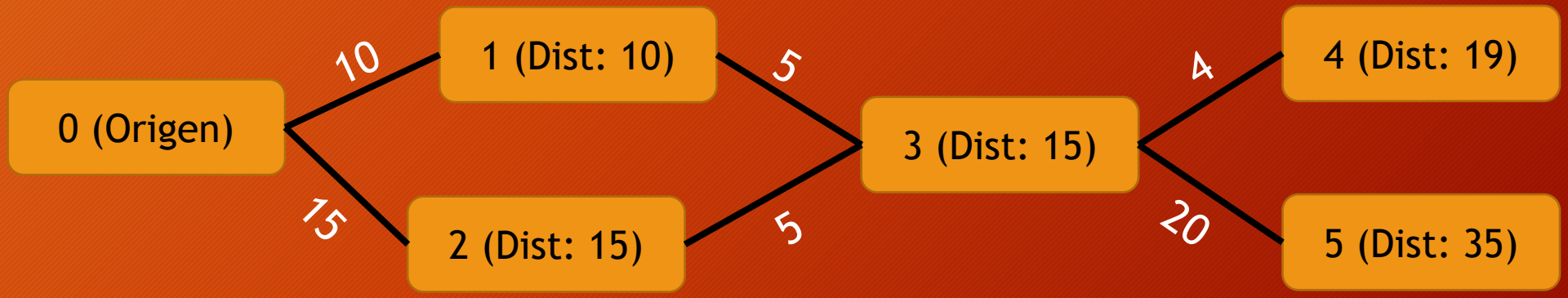


Grafo del ejemplo

Carrera en *Pekín Express*

A

Pero no basta solo con eso, ya que algunos equipos salen con penalización. La solución consiste en, para cada equipo, sumar su penalización a la distancia del punto que tienen que llegar. Ese es el tiempo exacto que tardarán. Ordenando esos valores (Teniendo en cuenta que, si llegan a la vez, se imprime primero el de menor índice), solucionamos el problema.



Grafo del ejemplo

Problema D

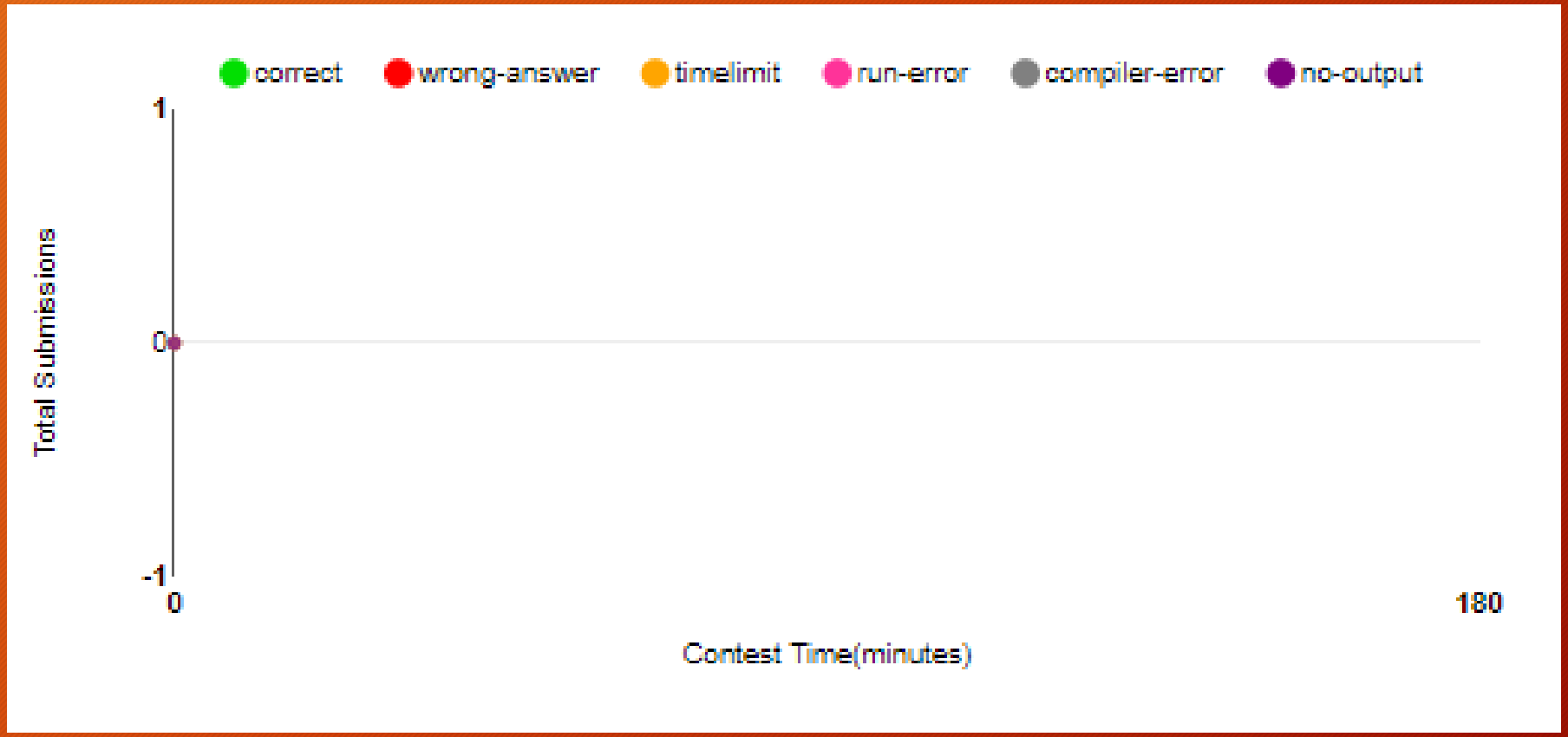
Comunidades en Cuarentena

Autor: Jakub Jan Luczyn

Sin envíos antes del Freeze

Comunidades en Cuarentena

D



Comunidades en Cuarentena

D

Lo primero a tener en cuenta es que no es necesario probar todos los posibles tamaños de comunidad, ya que los divisores del número son los únicos que aseguran una división exacta en partes iguales.

Aquí es importante saber que en los primeros 100.000 números (cota superior del tamaño de nuestro árbol) cabe esperar aproximadamente 100 divisores como máximo

(https://en.m.wikipedia.org/wiki/Highly_composite_number)

Comunidades en Cuarentena

D

Una vez establecido eso, se puede deducir que es necesario un método en O(n) para comprobar cada divisor.

Para ello se puede usar una búsqueda en profundidad. Empezando en un nodo cualquiera se hará una llamada recursiva a todos sus vecinos (excluido el padre) que devolverá "el número de elementos sin agrupar" en esa rama. Las llamadas a las hojas devolverán 1 y eso se irá propagando hacia el origen de la llamada. Cada nodo debe sumar los valores que recibe, sumar 1 a ese valor (él mismo) y comprobar si es posible crear un grupo de nodos.

Comunidades en Cuarentena

D

Hay 3 casos posibles:

- La suma es inferior al tamaño de grupo que se busca. En ese caso simplemente se propaga hacia arriba el valor para que lo intente agrupar el padre.

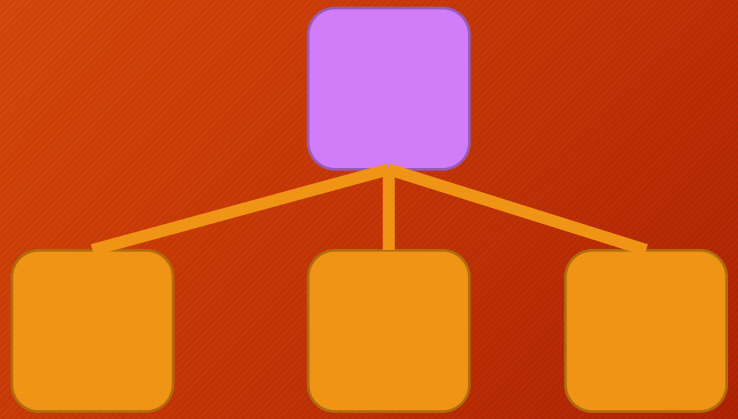
- La suma es igual al límite. Este nodo forma un grupo con los nodos sin asociar por abajo y devuelve 0, ya que su padre ya no necesita agruparse con ellos.

- El valor es superior al límite. En este caso hay que devolver un error para indicar que es imposible realizar la agrupación, ya que juntarse con alguno de los hijos dejaría "descolgada" otra rama. Por como se propagan los valores de retorno es imposible que un solo vecino devuelva un valor que haga superar el límite.

Comunidades en Cuarentena

D

Para el caso en el que el valor es superior al límite, la única posibilidad es que, para los hijos, no haya forma de agruparse entre ellos y todos apunten en última instancia al padre. Por ejemplo, en este árbol, no podemos juntar las comunidades de 2 en 2, ya que el padre (En morado) recibirá un 3, que indica que hay 3 hijos suyos sin emparejar, y que estos 3 hijos solo se pueden emparejar con el padre, lo cual es imposible:



Problema G

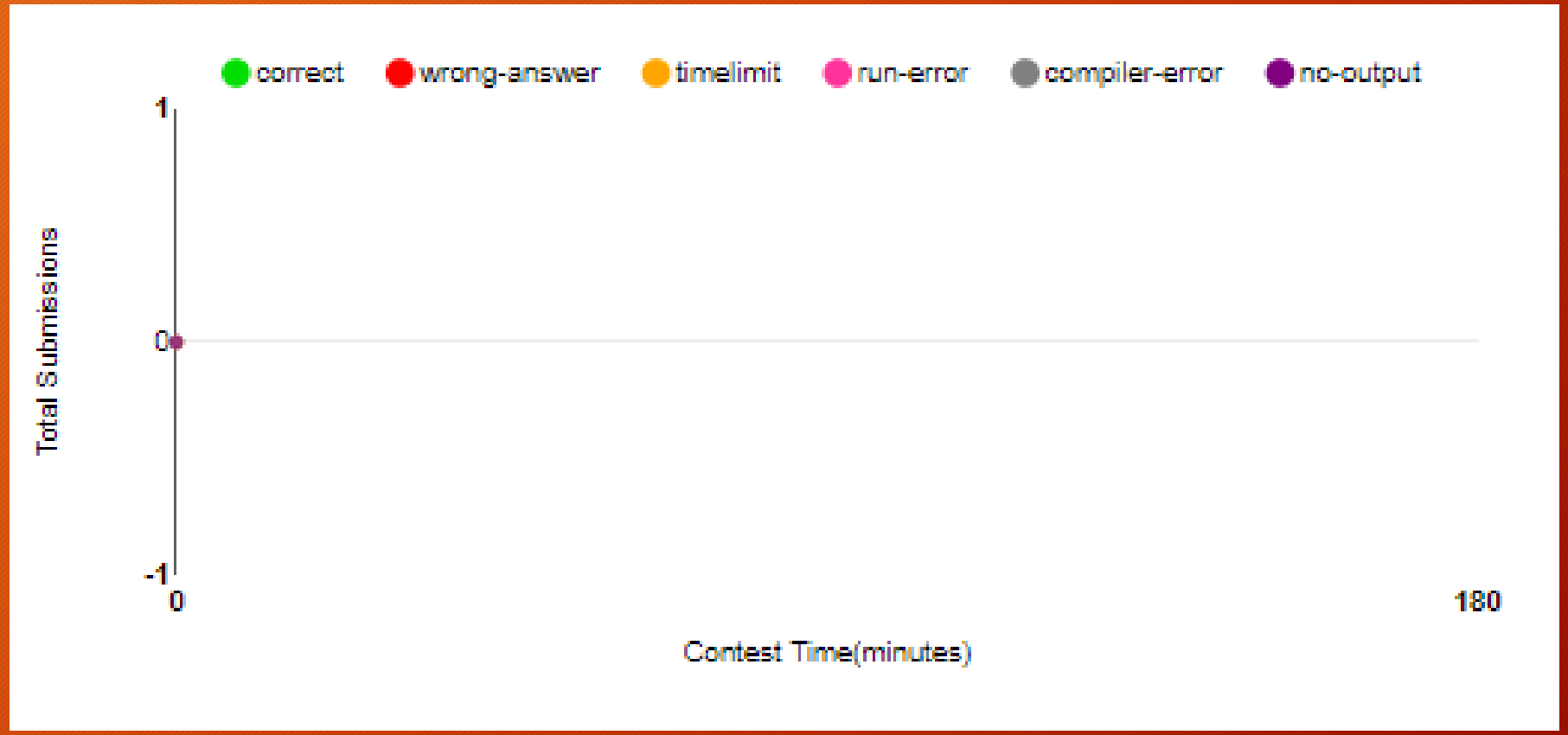
Capturando Humanos

Autores: Jakub Jan Luczyn & Iván Martín de San Lázaro

Sin envíos antes del Freeze

Capturando Humanos

G



Capturando Humanos

G

La solución a este problema es modelarlo, paso por paso. Lo importante está en fijarse que debemos calcular el tiempo mínimo del meteorito a cada ciudad, y posteriormente de cada ciudad a la lanzadera.

Ambas cosas se pueden hacer lanzando un Dijkstra, con la peculiaridad de que, para calcular el tiempo mínimo de todos los puntos a la lanzadera, lanzaremos un Dijkstra con el grafo inverso, origen Lanzadera.

Para conseguir el grafo inverso, simplemente invertimos el origen y el destino de todas las aristas que componen el grafo.

Capturando Humanos

G

En adición de llevar la suma de tiempos a la hora de calcular cada uno de los dos Dijkstra, debemos llevar también la cuenta de cuantas personas hemos abducido, continuando el Dijkstra si encontramos un tiempo igual al anterior, pero con un número mayor de personas abducidas.

Con el vector de resultados de Dijkstra, ordenamos por tiempo de llegada cada nave, y recorremos dicho vector hasta sumar tantas personas como nos piden. El tiempo de llegada de la última nave que complete el número de personas que nos piden es la solución.