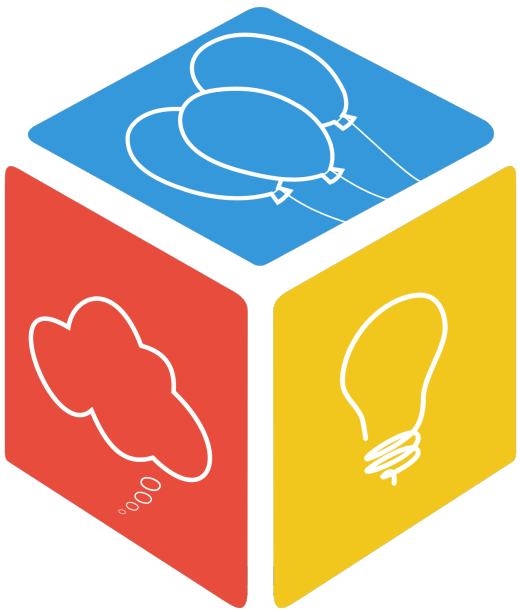
Editorial Maratón de Programación UFPS 2017



Programación Competitiva
UFPS

Universidad Francisco de Paula Santander

Junio 10 de 2017

Maratón Interna UFPS

La maratón interna UFPS 2017 contó la participación de 16 equipos de la Universidad Francisco de Paula Santander y contó con 12 problemas inéditos:

- A - Problem with a ridiculously long name but with a ridiculously short description
- B - Beautiful Triad
- C - Chris' Challenge
- D - The Domino Game
- E - Seven Segments Display
- F - The Robot's Grid

Maratón Interna UFPS

- G - Geonosis
- H - How many inversions?
- I - My password is a palindromic prime number
- J - Juliana and the stairs
- K - Polygonal Park
- L - The redundant manuscript

Maratón Abierta UFPS

La maratón abierta UFPS 2017, ejecutada a traves de la Red de Programación Competitiva RPC y en la cual participaron 209 equipos de más de 5 paises de latinoamerica contó con 12 problemas inéditos:

- A - Antivirus
- B - Beautiful Triad
- C - AirCraft: Monster
- D - Drought in Nlogonia
- E - My password is a palindromic prime number
- F - Macarons in la Fête

Maratón Abierta UFPS

- G - Geonosis
- H - How many inversions?
- I - Quidditch Match
- J - The Robot's Grid
- K - Polygonal Park
- L - Problem with a ridiculously long name but with a ridiculously short description

1. Problem with a ridiculously long name but with a ridiculously short description

Table: Basta mirar los primeros valores

n	$(66^n)\%100$	n	$(66^n)\%100$	n	$(66^n)\%100$
0	1	7	56	14	36
1	66	8	96	15	76
2	56	9	36	16	16
3	96	10	76	17	56
4	36	11	16	18	96
5	76	12	56	19	36
6	16	13	96	20	76

2. Beautiful Triad

Dados N y K, la respuesta es la cantidad de triadas que se pueden formar conteniendo solo números entre 0 y N, cuyas restas sean menores a K.

2. Beautiful Triad

Cuando $K = 0$, solo se pueden formar triadas que contengan 3 números iguales. Por tanto, si $K = 0$, entonces la respuesta es igual a $N + 1$.

2. Beautiful Triad

Cuando $K = 0$, solo se pueden formar triadas que contengan 3 números iguales. Por tanto, si $K = 0$, entonces $f(N, 0) = N + 1$.

Cuando $K = 1$, por cada par de números consecutivos se pueden formar 6 triadas. Así que cuando $K = 1$ entonces
$f(N, 1) = f(N, 0) + (6 * N)$

2. Beautiful Triad

Cuando $K = 0$, solo se pueden formar triadas que contengan 3 números iguales. Por tanto, si $K = 0$, entonces $f(N, 0) = N + 1$.

Cuando $K = 1$, por cada par de números consecutivos se pueden formar 6 triadas. Así que cuando $K = 1$ entonces $f(N, 1) = f(N, 0) + (6 * N)$

Cuando $K = 2$, se tiene que $f(N, 2) = f(N, 1) + (6K * (N - (K - 1)))$

2. Beautiful Triad

Generalizando, $f(N, K) = f(N, K-1) + (6 * K * (N - (K-1)))$

2. Beautiful Triad

Generalizando, $f(N, K) = f(N, K-1) + (6 * K * (N - (K-1)))$

Al expandir esta fórmula y resolverla, se puede llegar a una expresión sencilla que no depende de $f(N, K-1)$.

3. Chris' Challenge

En este problema debemos agrupar números enteros en conjuntos, luego debemos decir cuantos conjuntos diferentes se formaron y cual es el tamaño del conjunto más grande.

3. Chris' Challenge

En este problema debemos agrupar números enteros en conjuntos, luego debemos decir cuantos conjuntos diferentes se formaron y cual es el tamaño del conjunto más grande.

El proceso descrito en el enunciado es el siguiente:

- Tomar cada número del rango como un conjunto unitario.
- Tomar cada par de números.
- Calcular la suma de los divisores de cada uno de los números.
- Verificar si la diferencia entre la suma de los divisores de ambos números es menor o igual a un valor k.
- Si esto se cumple, unir los conjuntos a los que pertenecen estos dos números.

3. Chris' Challenge

Para solucionar este problema hay dos tips que podemos tener en cuenta:

3. Chris' Challenge

Para solucionar este problema hay dos tips que podemos tener en cuenta:

- Precalcular la suma de los divisores de todos los números hasta 10^4

3. Chris' Challenge

Para solucionar este problema hay dos tips que podemos tener en cuenta:

- Precalcular la suma de los divisores de todos los números hasta 10^4
- La union de conjuntos puede ser implementada de manera eficiente con la estructura de datos *Disjoint Set - Union Find*.

4. The Domino Game

- Según la definición, un juego esta cerrado cuando las dos esquinas del juego son iguales ($b = c$), y todas las fichas que contengan la pinta de las esquinas ya han sido ubicadas previamente en el juego.

4. The Domino Game

- Según la definición, un juego esta cerrado cuando las dos esquinas del juego son iguales ($b = c$), y todas las fichas que contengan la pinta de las esquinas ya han sido ubicadas previamente en el juego.
- Se debe llevar a binario el número a que representa el juego, y verificar si los bits de las 7 fichas de la pinta b se encuentran activos. De ser así, el juego está cerrado.

5. Seven Segments Display

Solución: Implementación sencilla.

- Acumular cuantos leds funcionales hay de cada posición.
- La cantidad de displays de siete segmentos que se puede formar es igual a la mínima cantidad de leds funcionales en una posición.

6. The Robot's Grid

Una aproximación a este problema podría ser un BFS que cuente los caminos posibles. Sin embargo, el caso de 25 x 25 tiene 32247603683100 caminos, por lo que esta solución será TLE.

6. The Robot's Grid

Una aproximación a este problema podría ser un BFS que cuente los caminos posibles. Sin embargo, el caso de 25 x 25 tiene 32247603683100 caminos, por lo que esta solución será TLE.

Una observación importante es que cada vez que el robot gira, no puede volver a esa fila o columna, ni a una mas "externa" a ella, por lo cual su movimiento siempre será en forma de caracol hacia el centro. Por tanto, en cada giro surge un nuevo problema del mismo tipo, pero mas pequeño.

6. The Robot's Grid

Solución 1: Programación Dinámica

- Siempre que haya una sola fila o columna, hay un solo movimiento posible (De frente hasta la última casilla). Por tanto $f(1, C) = f(R, 1) = 1$

6. The Robot's Grid

Solución 1: Programación Dinámica

- Siempre que haya una sola fila o columna, hay un solo movimiento posible (De frente hasta la última casilla). Por tanto $f(1, C) = f(R, 1) = 1$
- En caso contrario, supongamos que el robot va ascendiendo por la primera columna. El puede girar en cualquier fila, por lo cual debemos considerar todas las posibilidades desde 1 hasta R. Al hacer esto, no puede volver a la columna en la que estaba, y por tanto cada posibilidad de 1 a R, debe considerarse con C - 1.

6. The Robot's Grid

Solución 1: Programación Dinámica

- Podriamos verificar si el movimiento es horizontal o vertical en cada paso, y realizar el paso anterior con filas y columnas por aparte. Sin embargo, es mas facil "girar" la figura completa en cada giro del robot, de forma que el siempre ascienda por la primera columna.

6. The Robot's Grid

Solución 1: Programación Dinámica

- Podriamos verificar si el movimiento es horizontal o vertical en cada paso, y realizar el paso anterior con filas y columnas por aparte. Sin embargo, es mas facil "girar" la figura completa en cada giro del robot, de forma que el siempre ascienda por la primera columna.
- Para esto, basta con invertir las filas y las columnas en la recursión. Es decir: $f(R, C) = \sum_{i=1}^{R} f(C-1, i)$

6. The Robot's Grid

Solución 1: Programación Dinámica

- Podriamos verificar si el movimiento es horizontal o vertical en cada paso, y realizar el paso anterior con filas y columnas por aparte. Sin embargo, es mas facil "girar" la figura completa en cada giro del robot, de forma que el siempre ascienda por la primera columna.
- Para esto, basta con invertir las filas y las columnas en la recursión. Es decir: $f(R, C) = \sum_{i=1}^{R} f(C-1, i)$
- Es fundamental utilizar memorización para no repetir problemas superpuestos.

6. The Robot's Grid

Solución 2: Triangulo de Pascal

- Si bien no se explicará a profundidad, cabe notar que la solución de este problema tiene una relación directa con el triangulo de Pascal.

7. Geonosis

En este problema debemos calcular el poder que necesita la nave Rogue Two para poder destruir todas las torres que componen Geonosis en el orden que el capitan quiere.

El poder para destruir una torre se encuentra dado por la suma de la energia mínima necesaria para enviar mensajes entre cada par de torres de la fortaleza, es decir

$$\sum_{v,u,u!=v} d(u,v)$$

donde $d(u,v)$ es la energia mínima para enviar un mensaje de la torre u a la torre v.

7. Geonosis

Solución: APSP, Floyd-Warshall

7. Geonosis

Solución: APSP, Floyd-Warshall

Para lograr tener un ACC en este ejercicio, es necesario entender bien como funciona el algoritmo de Floyd Warshall.

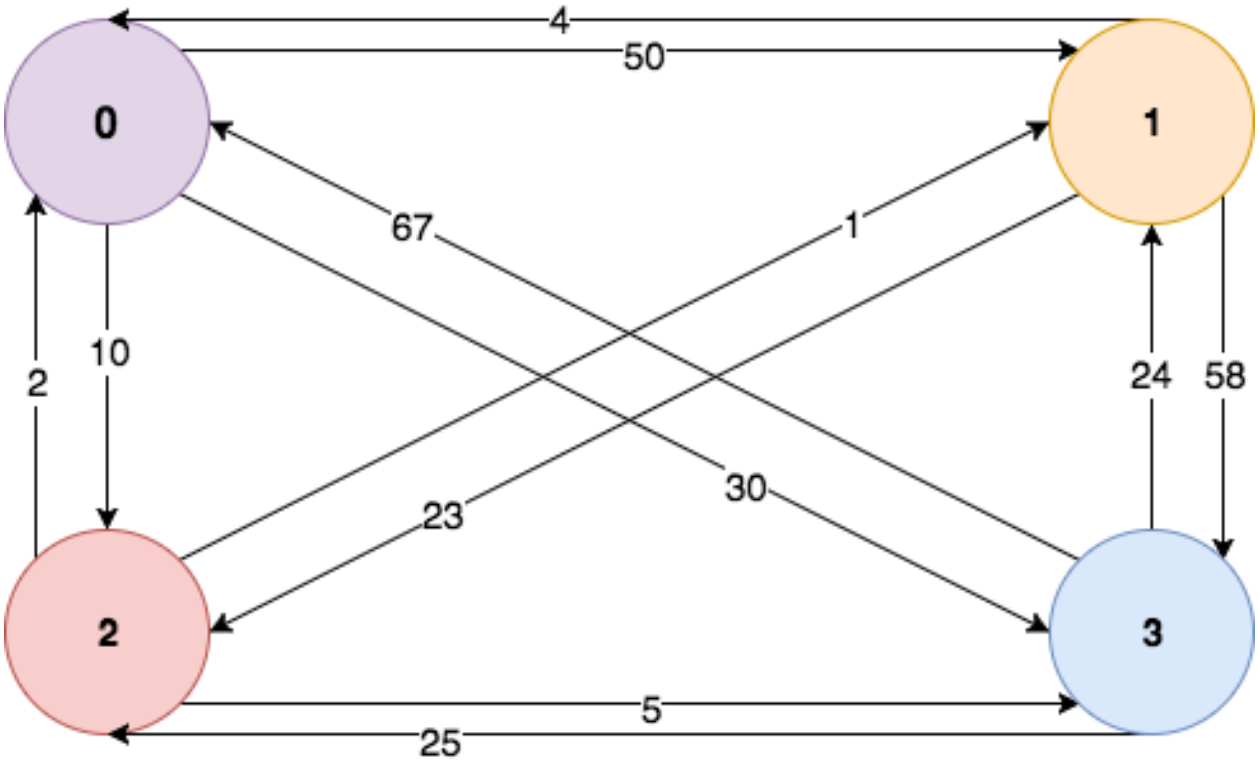
```
1  void floydWarshall(){
2      int k, i ,j;
3
4       for( k = 0; k < n; k++ ){
5           for( i = 0; i < n; i++ ){
6               for( j = 0; j < n; j++ ){
7                   ady[i][j] = min( ady[i][j], ( ady[i][k] +
       ady[k][j] ) );
8               }
9           }
10      }
11 }
```

7. Geonosis

- El ciclo externo del Floyd-Warshall k, controla el nodo pivot del grafo.
- Cada que se elige un nodo pivot, se calculan todas las rutas pasando a traves de ese nodo.
- Los ciclos internos calculan y actualizan las rutas entre cada par de nodos $i - j$, *pasando por el nodo pivot* k.
- No importa el orden en el que se elija el nodo pivot, dado que en cada iteración se recalculan todas las rutas al final terminaremos con todas las rutas mínimas.

7. Geonosis

Veamos un ejemplo:



El orden en el que se van a eliminar las torres es: 0 - 3 - 1 - 2

7. Geonosis

- Para destruir la torre 0 va a necesitar 157 de energia. (Suma de la ruta mínima entre cada par de nodos 0,1,2,3)
- Para destruir la torre 3 va a necesitar 106 de energia. (Suma de la ruta mínima entre cada par de nodos 1,2,3 sin tomar en cuenta a 0)
- Para destruir la torre 1 va a necesitar 24 de energia. (Suma de la ruta mínima entre los nodos 1, 2 sin tomar en cuenta a 0 y a 3)
- Para destruir la torre 2 va a necesitar 0 de energia. (Rutas del nodo 2, sin tener en cuenta a 0, 3 y 1)
- Esto nos da como resultado 287 de Energia para destruir todas las torres de Geonosis.

7. Geonosis

¿Cómo hacer esto con 1 solo Floyd Warshall?

7. Geonosis

¿Cómo hacer esto con 1 solo Floyd Warshall?

Elijamos los nodos pivot en el orden inverso al que se desean destruir.

Para el ejemplo, el orden en el que añadiremos los nodos pivot será 2 - 1 - 3 - 0.

7. Geonosis

- Cuando el pivot es 2 tenemos las rutas mínimas sin pasar por 1, 3 o 0.
- Cuando añadimos el pivot 1 tenemos las rutas mínimas sin contar los nodos 3 o 0.
- Al añadir el pivot 3, tenemos las rutas mínimas sin contar con el nodo 0.
- Por último añadimos el pivot 0 y tenemos las rutas mínimas pasando por todos los nodos.

7. Geonosis

Para obtener la respuesta, en una variable vamos a ir acumulando el valor de la ruta entre $i - j$ si, tanto i como j ya han sido tomados como pivot.

- Cuando el pivot es 2 solo se acumula la ruta $2 - 2$.
- Cuando el pivot es 1 se acumulan las rutas $1 - 1$, $1 - 2$, $2 - 1$, $2 - 2$.
- Cuando el pivot es 3 se acumulan las rutas $1 - 1$, $1 - 2$, $1 - 3$, $2 - 1$, $2 - 2$, $2 - 3$, $3 - 1$, $3 - 2$, $3 - 3$.
- Por último cuando añadimos el pivot 0 se acumulan todas las rutas.

8. How many inversions?

El ejercicio incluye un pseudocódigo con la solución. Esta solución es correcta, y solo hace falta convertirla a un lenguaje aceptado. Se recomienda modificar el funcionamiento para no enviar el arreglo como parámetro como lo indica el pseudocódigo, sino mantenerlo como variable global a la cual se pueda acceder desde el interior de los métodos.

9. My password is a palindromic prime number

Se realiza la división de la forma tradicional como se haría a mano con dividendo, divisor cociente y residuo. El primer residuo siempre será uno. Seguir iterando (es decir, multiplicando el residuo por 10, tomandolo como nuevo divisor y seguir el proceso) hasta el fin del ciclo.

Como se garantiza que el resultado es un decimal periódico puro, en algún punto el residuo debe ser igual al primer residuo obtenido, y allí termina el ciclo. Como el dividendo es 1, y el divisor siempre es mayor, el primer residuo obtenido siempre será 1.

10. Juliana and the stairs

- Nos interesa saber si Juliana ha caido en todos los escalones en el rango, o si le faltan algunos. Por tanto nos interesan 3 datos: los valores menor y mayor del rango, y la cantidad de escalones en los que ha caído (no es la misma cantidad de caidas, pues pudo caer varias veces en el mismo escalón).

10. Juliana and the stairs

- Nos interesa saber si Juliana ha caido en todos los escalones en el rango, o si le faltan algunos. Por tanto nos interesan 3 datos: los valores menor y mayor del rango, y la cantidad de escalones en los que ha caído (no es la misma cantidad de caidas, pues pudo caer varias veces en el mismo escalón).
- Si la cantidad de elementos en el rango (mayor - menor + 1) es igual a la cantidad de escalones en los que ha caido, la respuesta es "TRUE". En caso contrario, es la resta entre ellos.

10. Juliana and the stairs

- Nos interesa saber si Juliana ha caido en todos los escalones en el rango, o si le faltan algunos. Por tanto nos interesan 3 datos: los valores menor y mayor del rango, y la cantidad de escalones en los que ha caído (no es la misma cantidad de caidas, pues pudo caer varias veces en el mismo escalón).
- Si la cantidad de elementos en el rango (mayor - menor + 1) es igual a la cantidad de escalones en los que ha caido, la respuesta es "TRUE". En caso contrario, es la resta entre ellos.
- Se recomienda utilizar una estructura de tipo set (TreeSet), pues no admite repetidos y ordena inmediatamente.

11. Polygonal Park

En primer lugar, se necesita calcular las dimensiones del rectangulo. Para esto podemos sumar el largo de todas las figuras entre el primer y segundo corner (incluyendolos) para obtener el ancho, y todas las figuras entre el segundo y tercer corner (incluyendolos) para hallar el alto. Con estos dos datos, ya sabemos el área del rectángulo completo.

11. Polygonal Park

En primer lugar, se necesita calcular las dimensiones del rectangulo. Para esto podemos sumar el largo de todas las figuras entre el primer y segundo corner (incluyendolos) para obtener el ancho, y todas las figuras entre el segundo y tercer corner (incluyendolos) para hallar el alto. Con estos dos datos, ya sabemos el área del rectángulo completo.

Luego de esto, debemos restar el area de cada casa. Si la casa es cuadrada, su area es K^2. Si es triangular, su area es $(\sqrt{3} * k)/4$.

12. The redundant manuscript

IMPORTANTE: Si se hacen concatenaciones, usar una estructura eficiente con las concatenaciones (En java, StringBuilder en lugar de String). Si se hacen impresiones directamente, usar un método de salida eficiente.

12. The redundant manuscript

IMPORTANTE: Si se hacen concatenaciones, usar una estructura eficiente con las concatenaciones (En java, StringBuilder en lugar de String). Si se hacen impresiones directamente, usar un método de salida eficiente.

Las palabras con 3 o menos letras se agregan a la salida sin mayor procesamiento. Las palabras mayores, se almacenan en alguna estructura que permita optimamente saber si han sido agregadas previamente, y que no permita repetidos (Set o treeset recomendado).

13. Antivirus

Solución: Estructuras de Datos, Fenwick Tree, Segment Tree.

13. Antivirus

Solución: Estructuras de Datos, Fenwick Tree, Segment Tree.

Para poder solucionar este problema de manera eficiente es necesario contar con una estructura de datos que nos permita ejecutar consultas y actualizaciones de manera optima.

Usando estas estructuras de Datos se puede almacenar:

- La cantidad de registros que existen en un rango.
- La cantidad máxima de registros que se pueden infectar en un archivo, cumpliendo con la manera de propagación del virus hacia arriba.
- La cantidad máxima de registros que se pueden infectar en un archivo, cumpliendo con la manera de propagación del virus hacia abajo.

14. Drought in Nlogonia

Si bien hay una solución posible con Programación Dinámica, la solución mas eficiente de este problema es con el uso de colas.

14. Drought in Nlogonia

Si bien hay una solución posible con Programación Dinámica, la solución mas eficiente de este problema es con el uso de colas.

14. Drought in Nlogonia

Si bien hay una solución posible con Programación Dinámica, la solución mas eficiente de este problema es con el uso de pilas.

Por cada muro leído, se eliminan muros del tope de la pila mientras sean menores al muro actual, y sean diferentes al muro mas alto almacenada actualmente.

14. Drought in Nlogonia

Si bien hay una solución posible con Programación Dinámica, la solución mas eficiente de este problema es con el uso de pilas.

Por cada muro leído, se eliminan muros del tope de la pila mientras sean menores al muro actual, y sean diferentes al muro mas alto almacenada actualmente.

Al eliminar muros, es necesario actualizar la medida del ancho del nuevo compartimento como la suma de los compartimentos que lo conformaban.

14. Drought in Nlogonia

Al llegar al último muro, ya se han eliminado todos los muros que quedarian sumergidos, dejando solo los compartimientos que aseguran el máximo, y que deben llenarse a tope.

Cada compartimiento debe llenarse entonces hasta la pared mas pequeña de sus extremos.

15. AirCraft: Monster

Problema subset sum en 3 dimensiones. Este problema es un NP-Completo.

Como hay máximo 30 misiones, es posible solucionar el problema con un algoritmo $O(2^{N/2}N)$.

15. AirCraft: Monster

Se deben dividir por la mitad las misiones, y en cada mitad formar todas las combinaciones posibles. Si alguna combinación coincide con la respuesta, inmediatamente podemos indicar que es posible.

15. AirCraft: Monster

Se deben dividir por la mitad las misiones, y en cada mitad formar todas las combinaciones posibles. Si alguna combinación coincide con la respuesta, inmediatamente podemos indicar que es posible.

Al finalizar, tendremos dos listas de combinaciones. Es importante ordenar estas listas bajo un criterio (Por ejemplo, ordenar por X, en X iguales ordenar por A, y en A iguales ordenar por D).

15. AirCraft: Monster

Se deben dividir por la mitad las misiones, y en cada mitad formar todas las combinaciones posibles. Si alguna combinación coincide con la respuesta, inmediatamente podemos indicar que es posible.

Al finalizar, tendremos dos listas de combinaciones. Es importante ordenar estas listas bajo un criterio (Por ejemplo, ordenar por X, en X iguales ordenar por A, y en A iguales ordenar por D).

Sumamos la combinación mas alta de la primera lista, con la combinación mas baja de la segunda. Si la suma es igual al esperado, podemos imprimir posible. Si es menor, pasamos a la siguiente combinación de la segunda lista. Si es mas mayor, pasamos al anterior de la primera.

15. AirCraft: Monster

Se repite este proceso hasta encontrar las combinaciones que cumplan la condición, o hasta llegar al final de una de las dos listas (en cuyo caso, es imposible formar la combinación).

16. Macarons in la Fête

- Pierre hace 1 bandeja de macarrones de cada sabor, para un total de n bandejas.
- Cada bandeja tiene m macarrones.
- Cada bandeja es de 1 solo tipo *Traditional* o *Glazed*.
- Un cliente es feliz si Pierre hace una bandeja de al menos 1 de sus preferencias.
- Cada cliente tiene una lista de preferencias en donde máximo 1 macaron es de tipo glazed.
- Pierre quiere hacer a todos sus clientes felices y **minimizar** la cantidad de bandejas de tipo glazed, ¿Es posible lograr esto?
- Si es así, ¿cuales bandejas deben ser de tipo Glazed?

16. Macarons in la Fête

Solución: Greedy

- Evaluar cliente por cliente.
- Si la única posible preferencia de un cliente es un *Glazed Macaron*, la bandeja de ese sabor se hace *Glazed*.
- Cuando un sabor se hace *Glazed*, se descartan todas las preferencias *Traditional* de ese sabor en los demás clientes.
- Si hay un cliente cuyas preferencias son solo *Traditional Macarons* y ya todos esos sabores han sido hechos de tipo *Glazed*, la solución es imposible.
- Si todos los clientes estan satisfechos, tenemos una solución valida.

16. Macarons in la Fête

Solución: Greedy

Cuando se elige una bandeja de tipo *Glazed* es porque no hay otra opción posible, esto garantiza que la repuesta obtenida tiene el mínimo número de bandejas de tipo *Glazed*.

17. Quidditch Match

Originalmente, este es un problema típico de diagramas de Voronoi (Algoritmo de Fortune).

Consiste en conformar las áreas que encierran a cada jugador, y sumar las áreas de los jugadores de cada equipo, dando como resultado el área mas grande.

Sin embargo, dado que los casos son pequeños (Como máximo habrán 10 jugadores de cada equipo, para un total de 20 y el campo no puede ser muy grande), es posible lograr accepted con otras soluciones (como Montecarlo).