Introducción

Redigonda Maximiliano

Índice

Introducción

Problema

Rescatando el tesoro - Enunciado

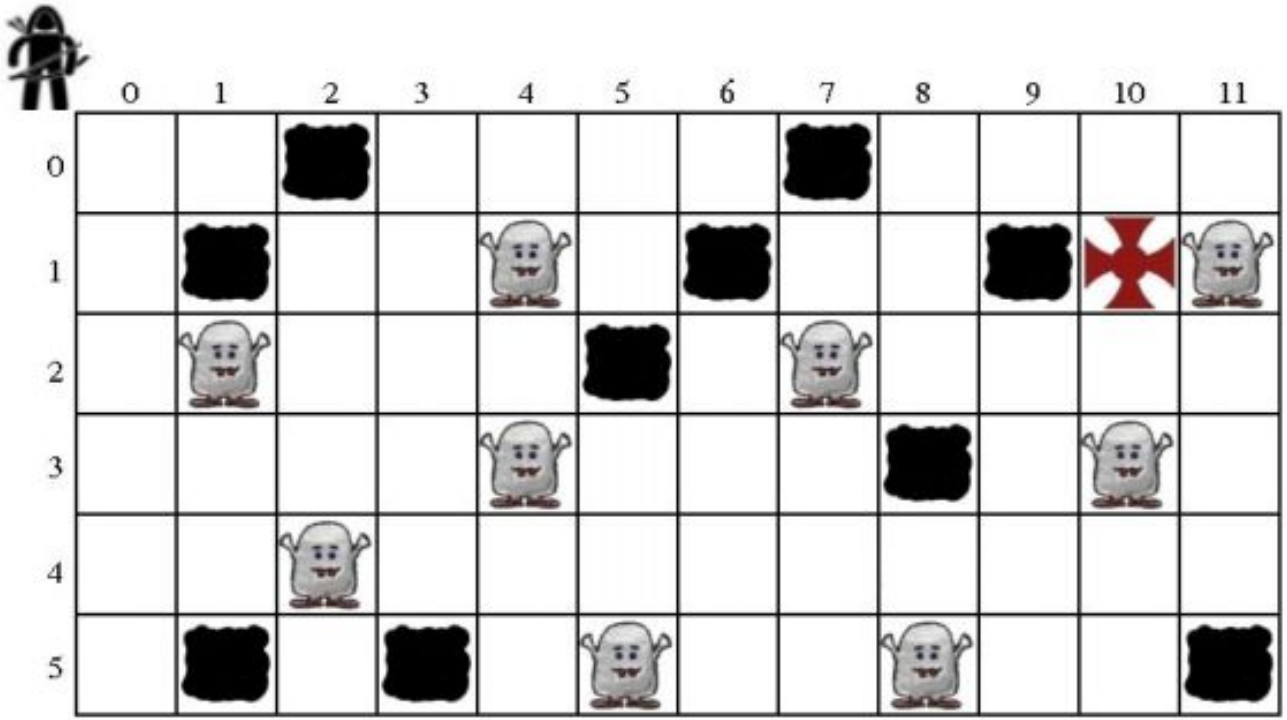
Un arquero fue enviado a rescatar un tesoro de una cueva.

La cueva consta de N x M habitaciones, dispuestas en forma de grilla. Hay puertas entre habitaciones adyacentes (que comparten un lado). Nuestro arquero comienza en la casilla (0, 0).

Cada habitación puede tener un pozo, un monstruo, o un tesoro (el tesoro se encuentra en exactamente una casilla).

El arquero no puede entrar en habitaciones con pozos, y cuenta con F flechas anestésicas que pueden dormir a monstruos durante tiempo indeterminado.

Rescatando el tesoro - Enunciado



Rescatando el tesoro - Enunciado

Nuestro arquero se puede mover entre casillas consecutivas (que comparten un borde). No puede moverse en diagonal, no puede salir del tablero, no puede ir a una casilla con un pozo, y solo puede ir a una casilla con un monstruo si le quedan flechas, en ese caso, la usa y su cantidad de flechas se reduce en 1.

El objetivo es responder si es posible que nuestro arquero logre llegar al tesoro, y en caso de que sea posible, debemos responder el tamaño del camino más corto con el que puede hacerlo.

Rescatando el tesoro - Datos de entrada

La primera línea tiene los 3 números N, M y F, las dimensiones de la cueva, y la cantidad de flechas que nuestro arquero posee.

Las siguientes N líneas contienen una cadena de M caracteres con el siguiente significado:

- M -- Monstruo.
- # -- Pozo.
- T -- Tesoro
- . -- Vacío.

Rescatando el tesoro - Entrada de ejemplo

```
6 12 2

..#....#....

.#..M.#..#TM

.M...#.M....

....M...#.M.

..M.........

.#.#.M..M..#
```

Rescatando el tesoro - Datos de salida

La primer línea de la salida debe contener la palabra SI o NO, dependiendo de si es posible que el arquero llegue al tesoro.

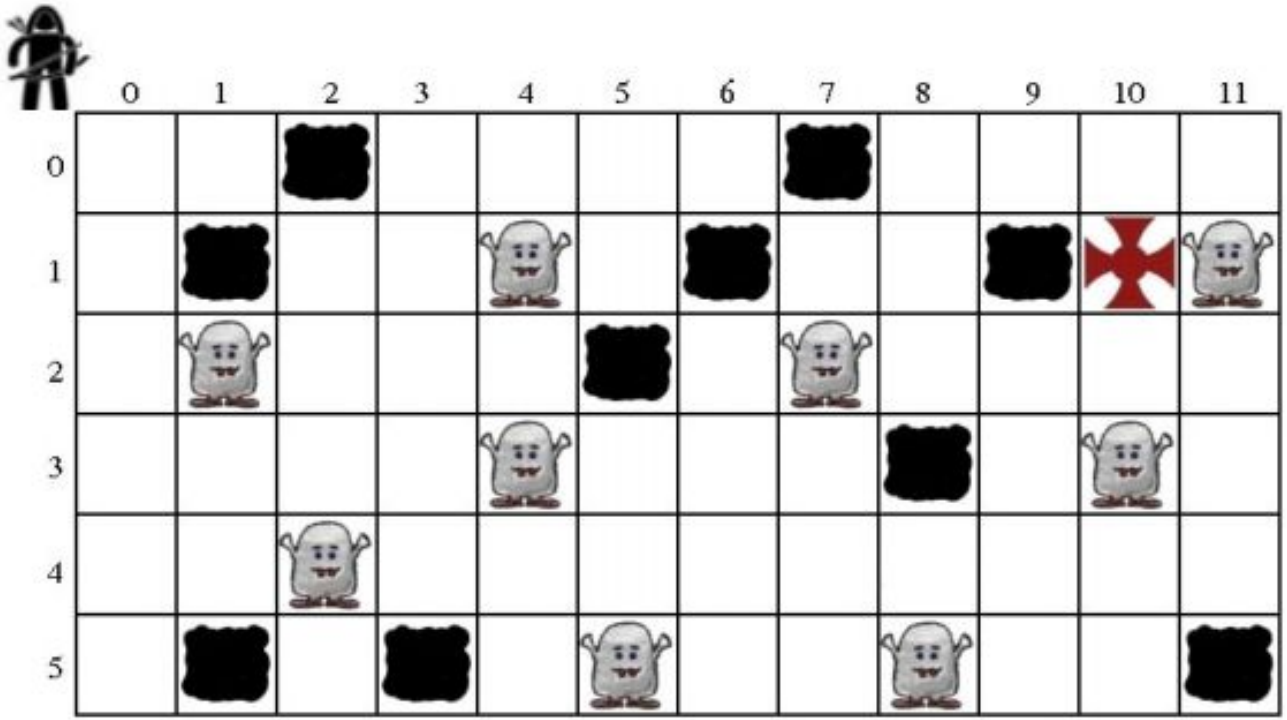
En caso de que la respuesta sea SI, la segunda línea debe contener un número entero indicando la mínima cantidad de habitaciones por las que el arquero debe pasar.

Rescatando el tesoro - Salida de ejemplo

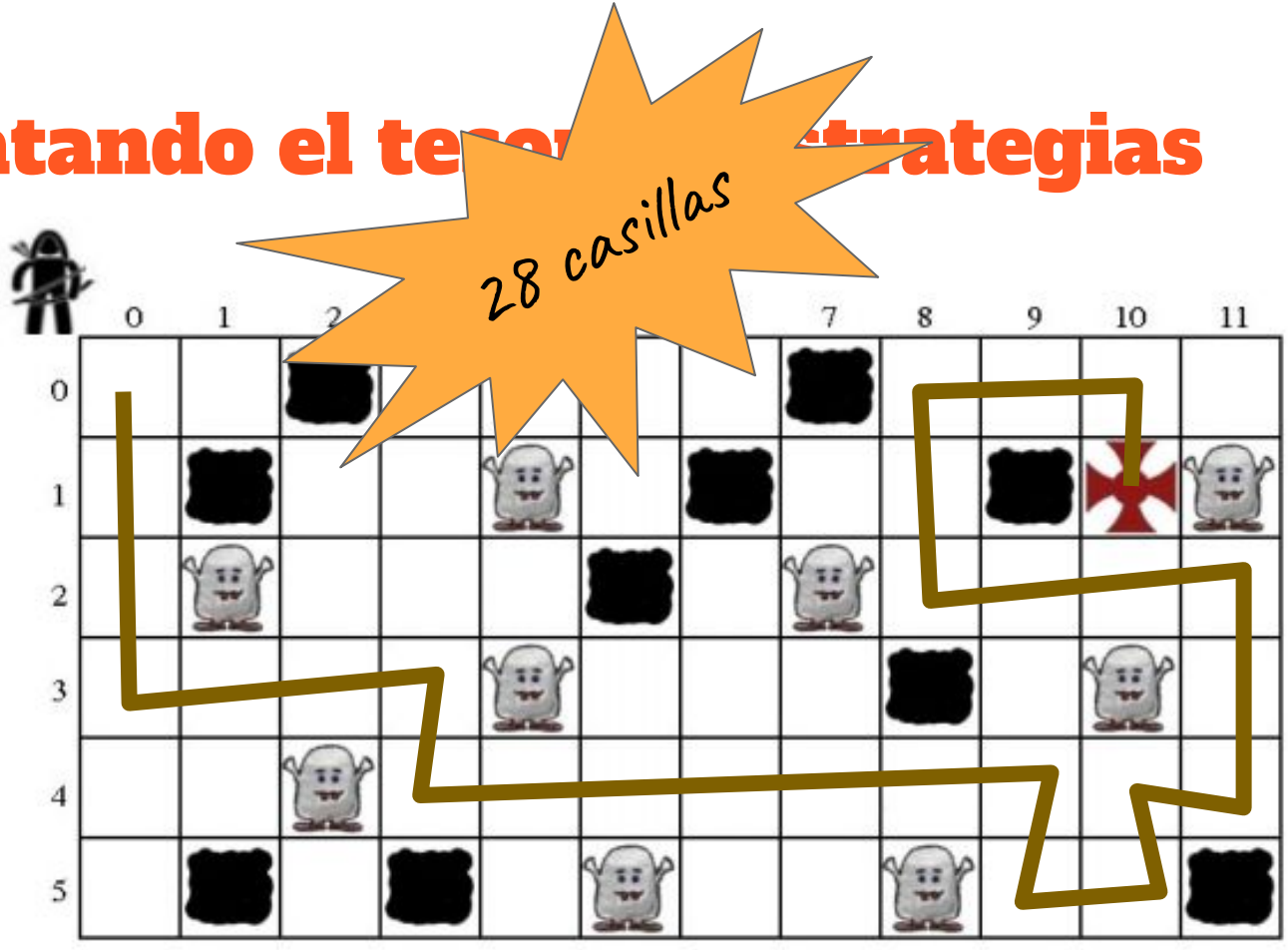
```
SI

16
```

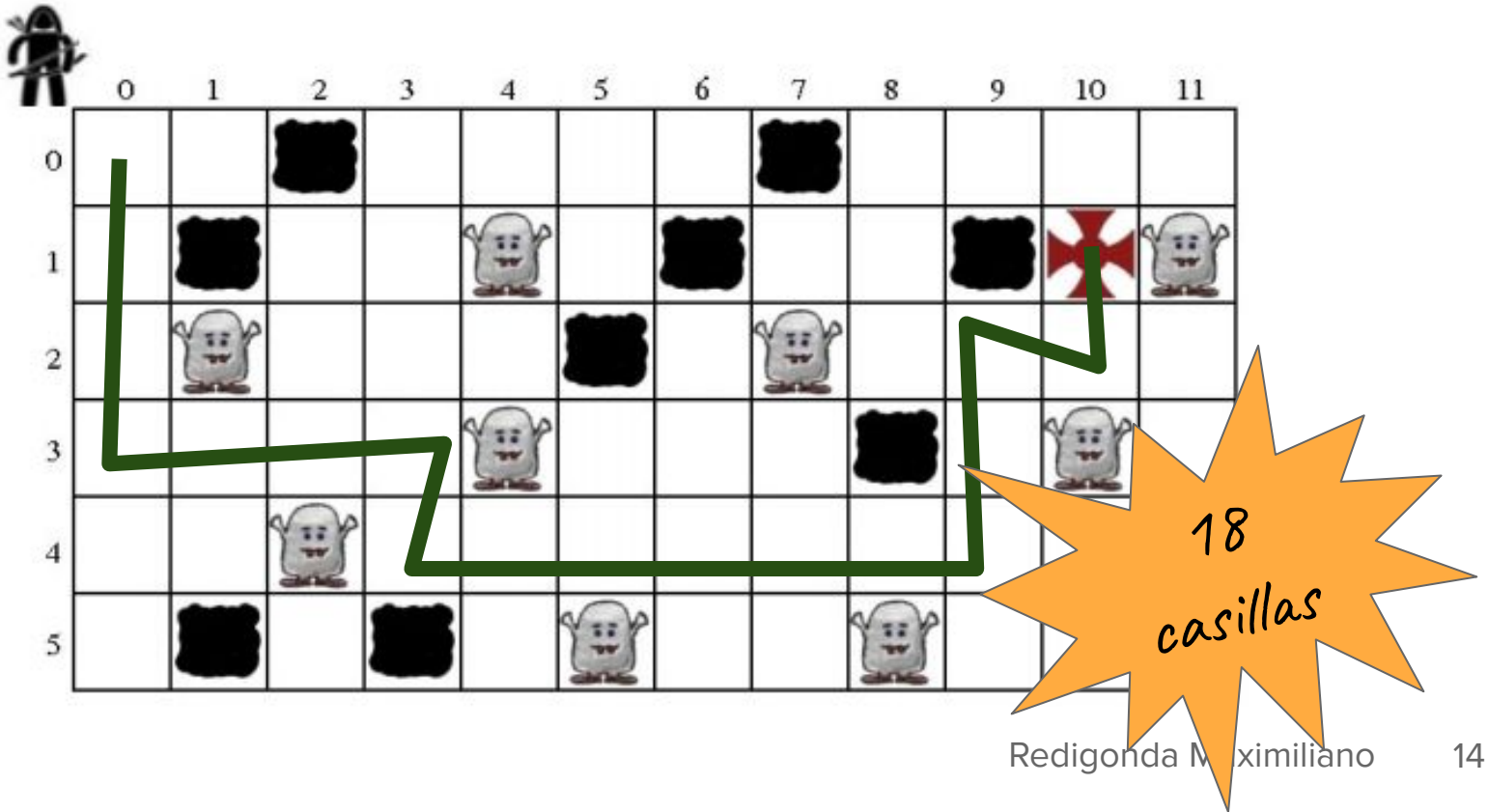
Rescatando el tesoro - Estrategias



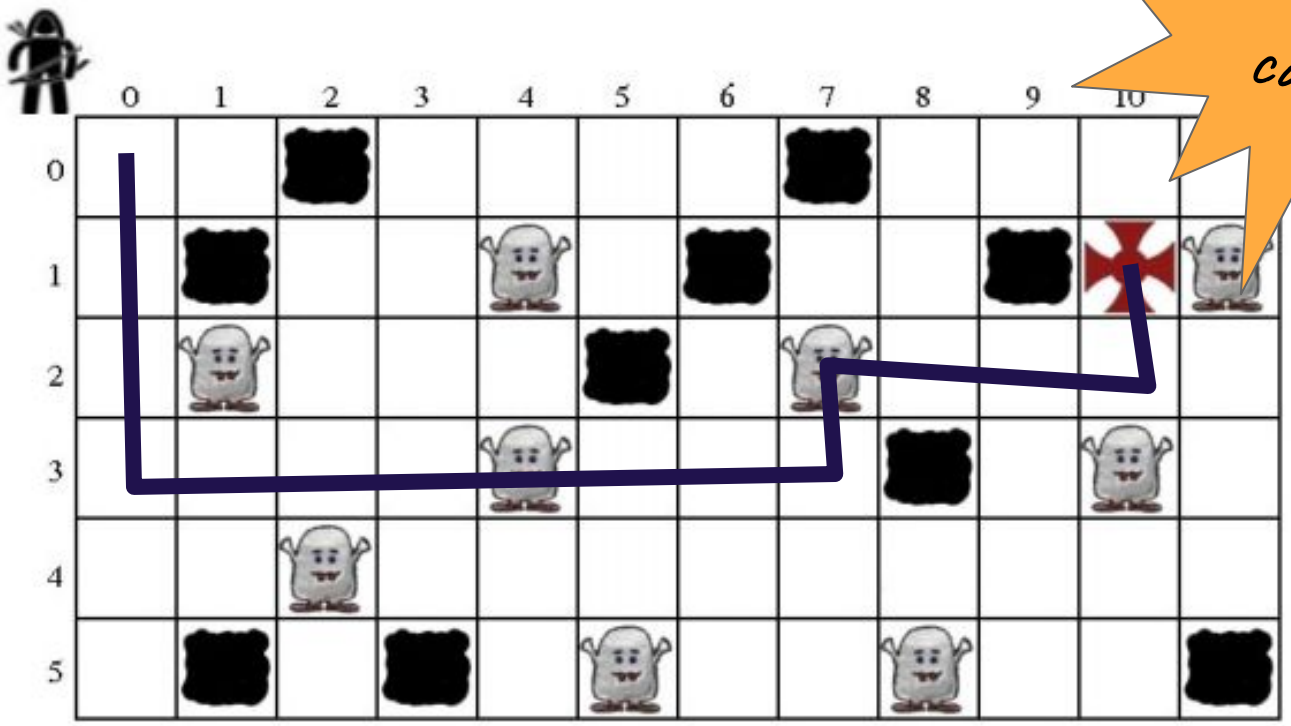
Rescatando el te[illegible]trategias



Rescatando el tesoro - Estrategias



Rescatando el tesoro - Estrategias



16 casillas

Problemas

Nuestro código tiene que ser correcto y suficientemente eficiente para todos los posibles casos que cumplan las restricciones ($1 \leq N \leq 100$, $1 \leq M \leq 100$, $0 \leq F \leq 20$, no hay pozos ni monstruos en la casilla (0, 0), etc.).

No se puede probar a ciencia cierta si tu código es correcto y rápido para todas las instancias posibles. Se utilizan cientos de casos de prueba, y se comparan los resultados de tu programa con uno que se conoce que es correcto, o se utiliza un checker.

Tipos de Competencias

Tipos de Competencias

Presenciales:

- IOI (International Olympiad in Informatics)
- ACM ICPC (International Collegiate Programming Contest)

Regulares:

- Codeforces
- AtCoder
- TopCoder

Anuales:

- Google Code Jam
- Facebook Hacker Cup

Olimpiada Informática

Para estudiantes de escuelas secundarias. Individual. Anual.

Cada país envía ≤4 competidores, y cada país tiene distintos criterios para seleccionarlos.

2 días de competencia, 5 horas cada una, 3 problemas a resolver.

Problemas con puntaje parcial.

Sin acceso a tabla de posiciones en vivo.

Sin material de referencia disponible.

ACM ICPC

Para estudiantes de universidades. Equipos de 3 personas. Anual.

Dividido en regiones, cada región envía una cierta cantidad de equipos dependiente de la cantidad de universidades de esa región.

1 día de competencia, 5 horas, entre 7 y 14 problemas.

Sin puntaje parcial, pero con penalidad por envíos tardíos o incorrectos.

Con acceso a la tabla de posiciones en vivo (yay!)

Con material de referencia disponible (yay!)

Codeforces

Comunidad (no /r/competitive_programming, no discord).

Competencias regulares, individuales, y cortas (generalmente 2 horas).

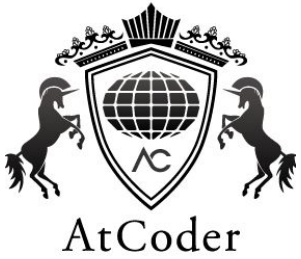
Problemas ordenados por dificultad general aproximada.

Sistema de rating más “oficial”. Divisiones.

Gym para simulacros virtuales de competencias que ya ocurrieron.

Editorial y discusión luego de cada competencia.

AtCoder



AtCoder Beginner/Regular/Grand Contests.

Competencias regulares, individuales, y cortas (cerca de 2 horas).

Problemas ordenados por dificultad general aproximada.

Enunciados generalmente cortos, problemas de “calidad”.

Sistema de rating interesante.

Final presencial, pocos participantes seleccionados.

TopCoder

Pionero en competencias de programación. Algo de comunidad.

Competencias regulares, individuales, y cortas (cerca de 2 horas).

Problemas ordenados por dificultad general aproximada.

Problemas de “calidad”, un poco “más matemáticos”.

Sistema de rating. Divisiones.

Final presencial, pocos participantes seleccionados.

Google Code Jam / Facebook Hacker Cup

Todas las rondas son online, excepto las finales. Anuales.

Solo pasan ≤25 personas de todo el mundo a las finales, sin restricciones de edad o de participaciones.

Regalan remeras (polos? camisetas?) a los mejores X participantes.

ICPC

ICPC

Aquella que merece nuestra atención :3

Subregional: utilizado por universidades para enviar equipos al regional.

Regional: entre 3 y 4 cupos* para disputar entre Argentina, Bolivia, Chile, Paraguay, Perú, y Uruguay.

ICPC

No más de 5 participaciones en la Regional, ni más de 2 participaciones en la Final Mundial por participante.

No más de 1 equipo por universidad pasa a la Final Mundial.

Cupo para el mejor equipo entre los países que no clasificaron.

... existen otras reglas extrañas como la de los “restos”.

ICPC - Reglas de la competencia

5 horas, 10-14 problemas, 3 participantes por equipo, 1 computadora.
C/C++, Java o Python.

En el Regional se puede llevar cualquier cantidad de material de referencia, en la final mundial sólo 25 páginas.

Cada problema vale 1 punto, y se desempata por penalidad.

Cada problema **resuelto correctamente** aporta como penalidad la cantidad de minutos en el que fue aceptado, más 20 puntos por cada envío incorrecto antes de ser aceptado (se puede intentar infinitas veces).

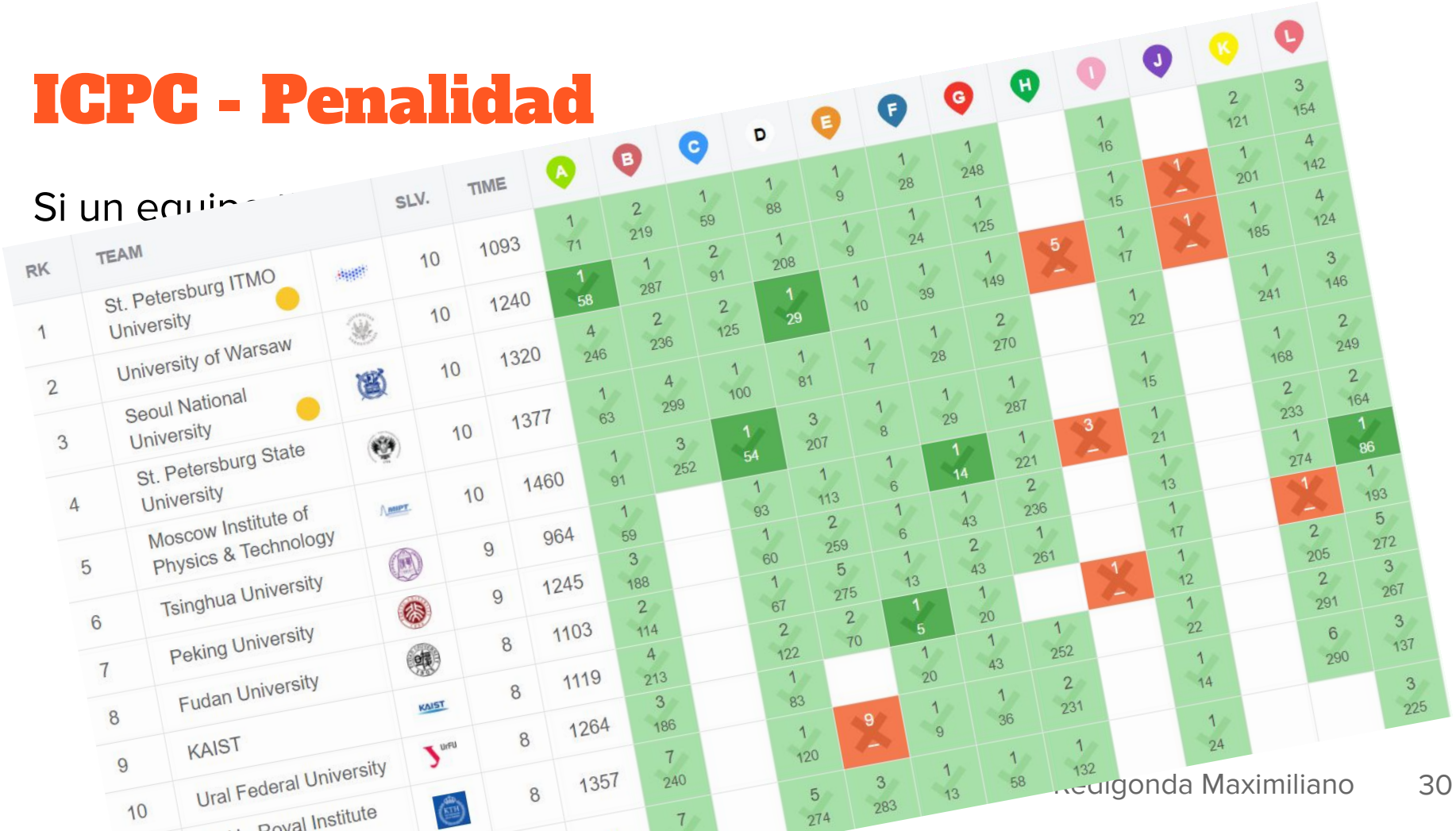
ICPC - Penalidad

Si un equipo tiene los envíos: A(47), B(55), B(59), C(120), C(270), B(287), B(289), B(290), B(292), B(294), B(297), H(299).

El equipo obtiene 3 puntos y 47 + 2*20 + 287 + 299 = 673 de penalidad.

ICPC - Penalidad

Si un equip



RK	TEAM	SLV.	TIME	A	B	C	D	E	F	G	H	I	J	K	L
1	St. Petersburg ITMO University	10	1093	1 71	2 219	1 59	1 88	1 9	1 28	1 248		1 16		2 121	3 154
2	University of Warsaw	10	1240	1 58	1 287	2 91	1 208	1 9	1 24	1 125		1 15	1 _	1 201	4 142
3	Seoul National University	10	1320	4 246	2 236	2 125	1 29	1 10	1 39	1 149	5 _	1 17	1 _	1 185	4 124
4	St. Petersburg State University	10	1377	1 63	4 299	1 100	1 81	1 7	1 28	2 270		1 22		1 241	3 146
5	Moscow Institute of Physics & Technology	10	1460	1 91	3 252	1 54	3 207	1 8	1 29	1 287		1 15		1 168	2 249
6	Tsinghua University	9	964	1 59			1 93	1 113	1 6	1 14	1 221	3 _	1 21	2 233	2 164
7	Peking University	9	1245	3 188			1 60	2 259	1 6	1 43	2 236		1 13	1 274	1 86
8	Fudan University	8	1103	2 114			1 67	5 275	1 13	2 43	1 261		1 17	1 _	1 193
9	KAIST	8	1119	4 213			2 122	2 70	1 5	1 20		1 _	1 12	2 205	5 272
10	Ural Federal University	8	1264	3 186			1 83		1 20	1 43	1 252		1 22	2 291	3 267
	Royal Institute	8	1357	7 240			1 120	9 _	1 9	1 36	2 231		1 14	6 290	3 137
		7					5 274	3 283	1 13	1 58	1 132		1 24		3 225

ICPC - Posibles respuestas del juez

ACCEPTED (AC) -- Problema aceptado, equipo suma 1 punto.

Compilation Error (CE) -- El código fuente tiene errores de compilación.

Runtime Error (RE) -- El programa se comporto indebidamente y tuvo que ser destruido (quizás accedió a un arreglo fuera de los límites!).

Time Limit Exceeded (TLE) -- Quizás era correcto, pero tardó demasiado.

Wrong Answer (WA) -- Corrió correctamente, pero la respuesta no fue correcta.

(Hay otros como PE, Filename Mismatch, etc.)

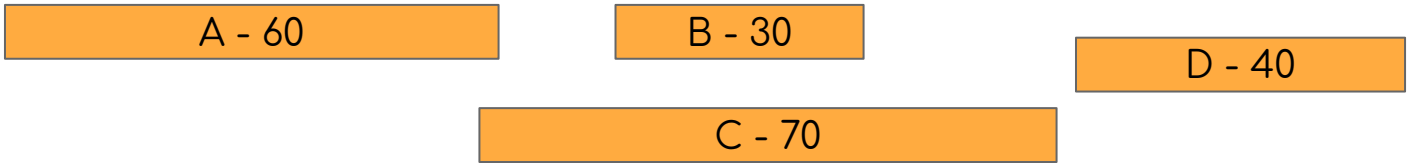
Cómo competir?

Modelo de competencias

Modelo matemático para competencias

Tenemos $7 \leq N \leq 14$ problemas, cada uno tarda en resolverse T_1, T_2, ..., T_N.

En qué orden debemos enviar los problemas para maximizar nuestro puntaje, y minimizar nuestra penalidad?



Si resolvemos en el orden ABCD:

60 + 90 + 160 + 200 = 510

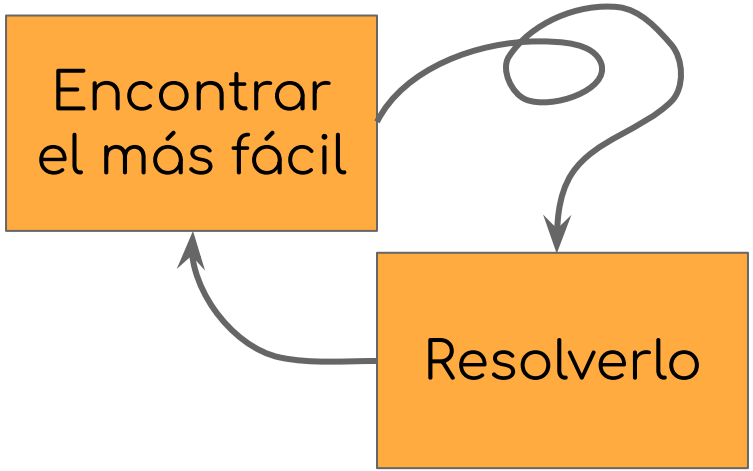
Eso resulta igual a:

60 + (60 + 30) + (60 + 30 + 70) + (60 + 30 + 70 + 40)

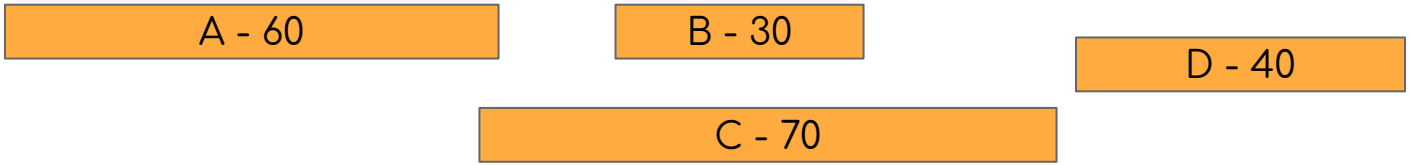
Modelo matemático para competencias

Conviene resolver el más corto primero!

La noción de “conviene” se puede probar matemáticamente.



Modelo matemático para competencias



Si resolvemos en el orden BDAC:

30 + 70 + 130 + 200 = 430

Eso resulta igual a:

30 + (30 + 40) + (30 + 40 + 60) + (30 + 40 + 60 + 70)

Técnicas generales

Técnica “Tablita”

Conocen “Los Simpsons”?

Apenas comienza la competencia, se crea una tabla con los nombres de los problemas.

Cuando un integrante quiere leer un problema, lo dice en voz alta, y su desarrollo en ese problema debe ser resumido en la “tablita”.

En momentos más avanzados de la competencia, sabremos con quien hablar por cada problema.

Técnica “Tablita”

A	B	C	D
E	F	G	H
I	J	K	L

Técnica “Tablita”

A: dar la permutación lexicográficamente mínima de una string. $N \leq 10^5$.	B	C	D: Greedy (o Flujo)! ordenar los monstruos por fuerza?
E: HARD	F: trivial	G: fuerza bruta fácil	H: DP en árboles.
I	J	K:	L

Técnica “Tiempo de Paz”

Conocen el Age of Empires?

Para dar tiempo a los imperios a desarrollarse, se establece un “tiempo de paz”.

En competencias, los primeros problemas a ser resueltos serán generalmente los más sencillos, los integrantes del equipo no se hablan (mucho) hasta que no queden problemas “fáciles”.

Técnica “Tiempo de Paz”

Mejor paralelización (se evita que dos o más estén en el mismo problema “fácil”). Al comienzo la competencia es “individual”.

Se evita duplicar el costo de leer, entender y pensar un problema “fácil”.

Costo de oportunidad: es el valor de la mejor elección que no fue tomada.

Si dos o más trabajan en un mismo problema, están sacrificando tiempo que puede ser utilizado para avanzar con otros problemas (que todavía no son imposibles).

Técnica “Buen Support”

Conocen a Braum?

Situación: nuestro compañero está luchando un problema desde hace bastante tiempo, y nos pide/le ofrecemos ayuda.

Peor: “contame el problema y lo que pensaste.”

Mejor: *lee el problema*, *piensa el problema superficialmente*, “dale, contame lo que pensaste”, *verifica que ambos entendieron lo mismo del problema*, *cuestiona todo*.

Técnica “Buen Support”

Incluso si tu compañero aún no te ha explicado su solución, leer su código puede ser muy útil.

Mientras más experiencia se tenga participando en equipo, más fácil será interpretar los códigos de tus compañeros y recordarles sus errores comunes (quizás siempre olvida usar **long long** en algunas situaciones?).

Los “hacks” de Codeforces/Topcoder pueden ser buenos para leer código e intentar encontrar casos bordes para los problemas.

Técnica “Enviar - Imprimir - Salir”

Conocen a Leonidas?

En muchas competencias podemos imprimir nuestros códigos.

Después de enviar un problema, no quedarse esperando el resultado, sino imprimir el código y dejar a tus compañeros progresar (la PC es el cuello de botella).

(me quedé sin memes)

Técnica “Codear en papel”

Situación frecuente: un competidor pensó el problema, está convencido de que tiene todas las ideas y de que puede empezar a programar enseguida, le dan la PC, y se pone a decidir cosas como: “hago una función para esto?”, “uso pares o me creo mi propia estructura”, etc.

Solución: escribir en papel, código válido*, ya que esta forma nos vemos obligados a pensar en todas las decisiones de diseño.

Además, removemos tiempo de uso de computadora ya que ahora solo debemos transcribir, y no programar.

Cómo entrenar?

Analizando ratings

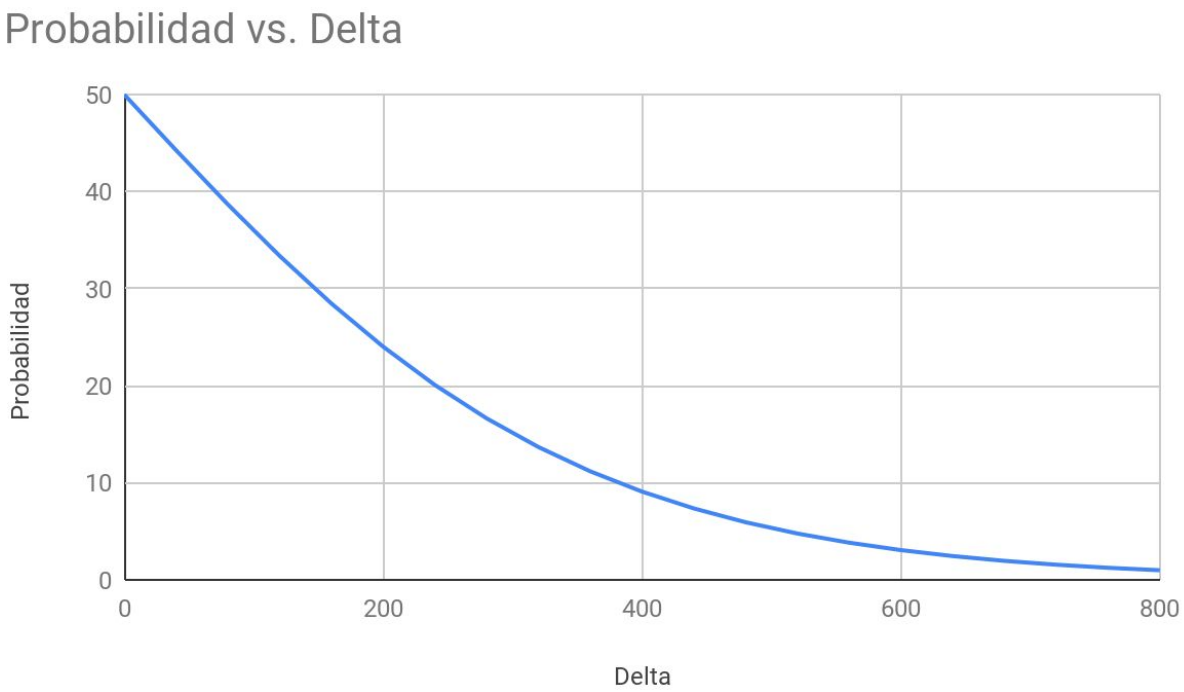
Rating en Codeforces

Casi todas las páginas que proveen competencias regulares (Codeforces, AtCoder, TopCoder) tienen algún tipo de sistema de rating.

La idea consiste en asociar un número (usualmente entre 0 y 4000) que sirva para medir qué tan bueno uno es compitiendo. Este puntaje se ajusta después de cada competencia.

Para un par específico de competidores, si la diferencia de rating es D, la probabilidad de que gane el de menor rating es de: $$\frac{1}{1+10^{\frac{D}{400}}}$$

Rating en Codeforces - Ejemplos



Delta	Probabilidad
0	50%
50	43%
100	36%
150	30%
200	24%
250	19%
300	15%
350	12%

Rating en Codeforces - Problemas

Utilizando la misma fórmula, podemos también asignarle un rating a cada problema, y aproximar la probabilidad de resolverlo durante la competencia.

Junto con el sistema de etiquetas (greedy, dp, data structures, etc.) de Codeforces, esto nos ayudará a poder entrenar

Existen algunos problemas de los cuales tenemos que estar al tanto. Si un problema es fácil, pero la mayoría de los competidores se enfocaron en otros problemas y no lo resolvieron, el problema terminará con un rating alto.

Rating en Codeforces - Funciona?

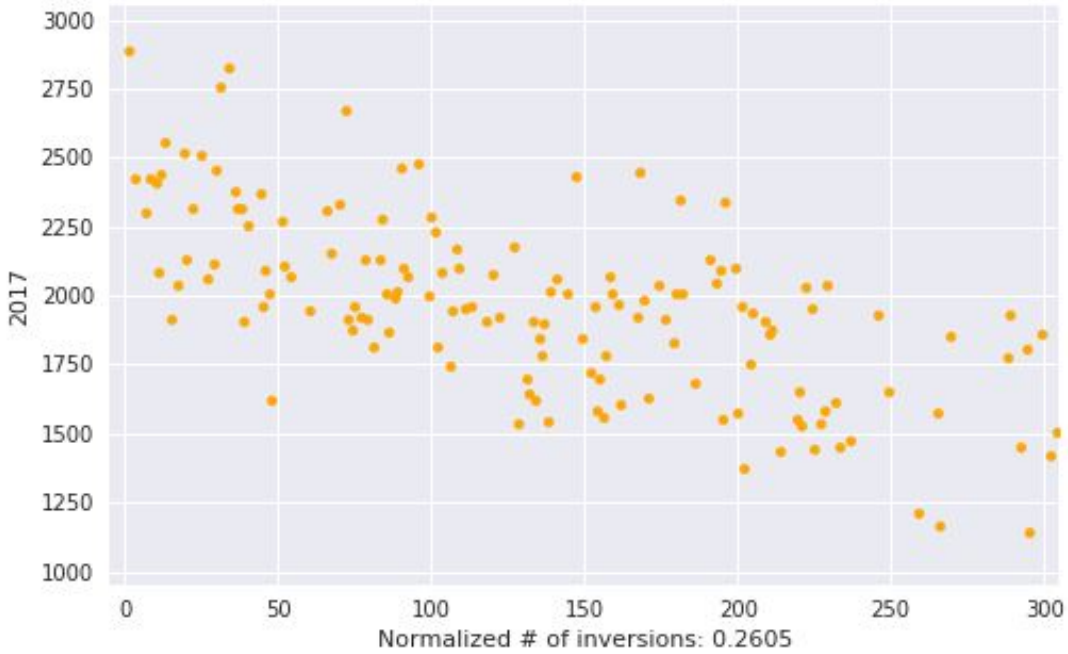
El sistema funciona decentemente dentro de Codeforces.

Qué pasa si intentamos utilizar ese mismo rating para otra competencia, como la IOI?

Tenemos ciertas diferencias importantes:

- Duración (2h vs 5h)
- Cantidad de problemas (5/6 vs 3)
- Posibilidad de ver tabla de posiciones
- Material de referencia, temario, etc.

Rating en Codeforces - IOI



Si bien existe correlación, no podemos decir mucho de un competidor con 1900 puntos de rating.

En ICPC tenemos una diferencia aún mayor: la competencia es en equipos.

Analizando ratings

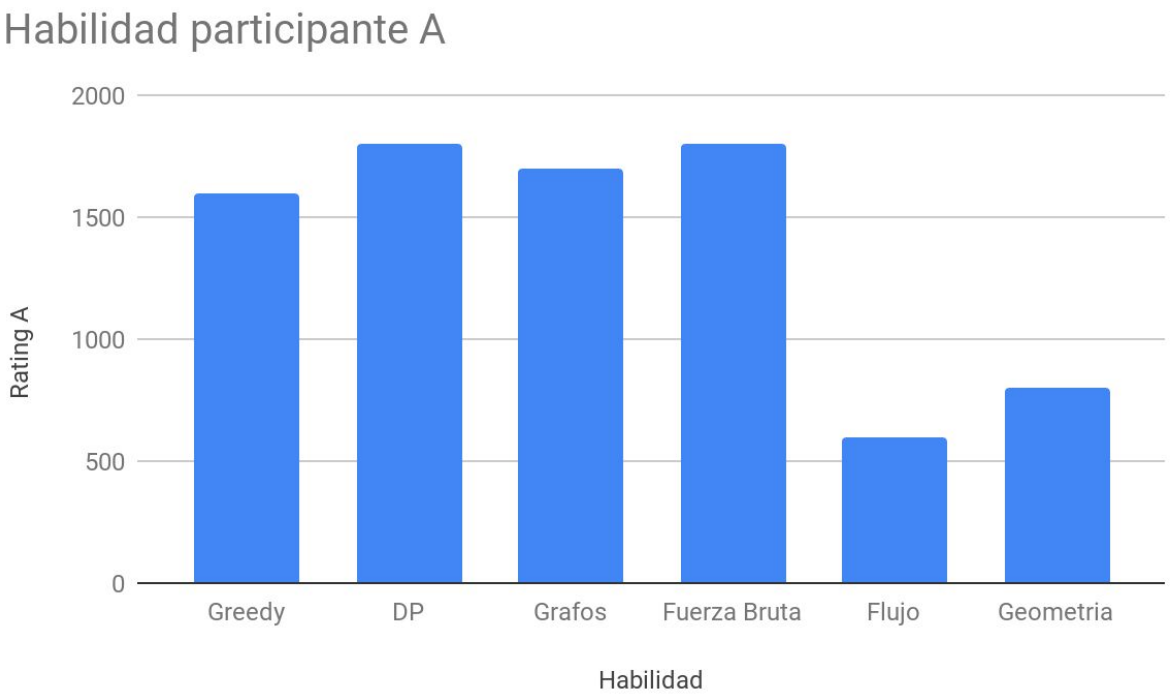
Nosotros podemos tener una mejor perspectiva de nuestras habilidades.

Si nos enfocamos en entrenar un tema específico, podríamos ser capaces de resolver problemas con rating 2200 con mucha probabilidad, a pesar de que nuestro rating nos diga que somos 1600.

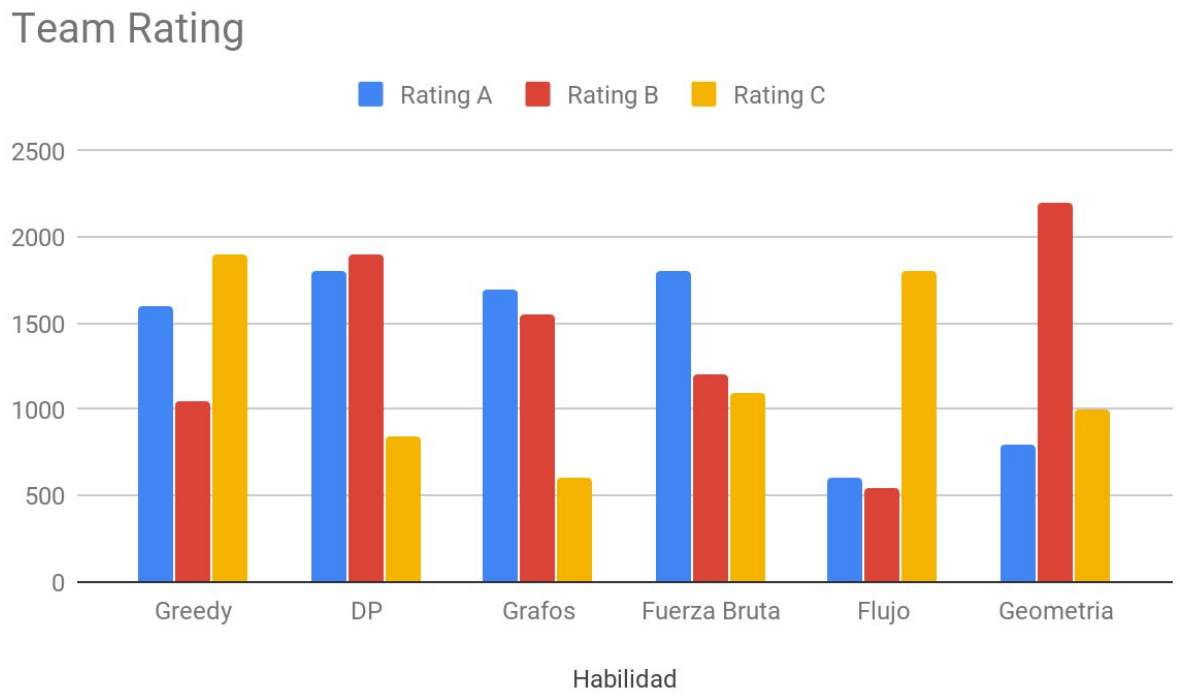
Nuestro rating sería cercano a 0 no sabemos programar, sin importar si tuvimos todas las observaciones necesarias para resolver el problema!

Podemos intentar descomponer el rating de alguien en su habilidad con cada tema general.

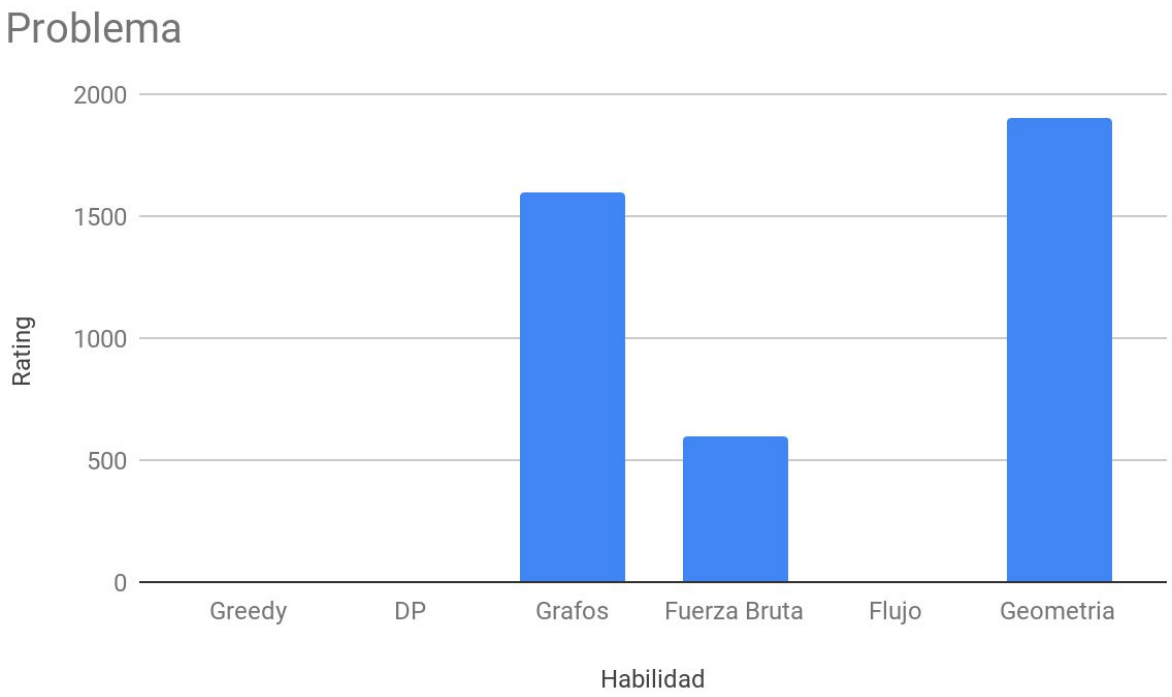
Analizando ratings - Ejemplo



Analizando ratings - Equipo ejemplo



Analizando ratings - Ejemplo problema



Analizando ratings

Si un integrante del equipo puede resolver un problema, asumimos que el equipo en su totalidad puede resolver ese problema. Esto motiva a que el rating de un equipo en cada tema sea al menos el máximo de los ratings de sus integrantes en ese tema.

Si un problema requiere grandes habilidades de estructuras de datos, y grandes habilidades de grafos, y tenemos integrantes del equipo especializados en esos temas, el problema podrá ser resuelto, provisto que haya **buena comunicación**.

Analizando ratings

Además, si dos integrantes son buenos en un tema, si unen fuerzas, podrían resolver problemas de ese tema más rápido (paralelizando descubrir algunas observaciones, y posiblemente paralelizando código).

Conclusión #1: conviene tener cierto grado de separación entre los temas que cada integrante entrena (si todos enfocan en grafos y DP no resulta tan óptimo como que se dividan grafos, DP y geometría).

Conclusión #2: siempre sirve aprender más, no importa si tu compañero es “el encargado” de aprender de tema X, saber algo más de X siempre puede ayudar.

Técnicas generales

Recursos

Un recurso es algo que te puede enseñar algo (un libro, un tutorial, un blog, un video, un amigo, un profesor).

Para competencias de programación, tenemos algunos recursos casi obligatorios:

- Competitive Programming 3 (Steven y Felix Halim)
- Guide to Competitive Programming (Antti Laaksonen)
- Blogs en Codeforces/Topcoder/etc

Upsolving

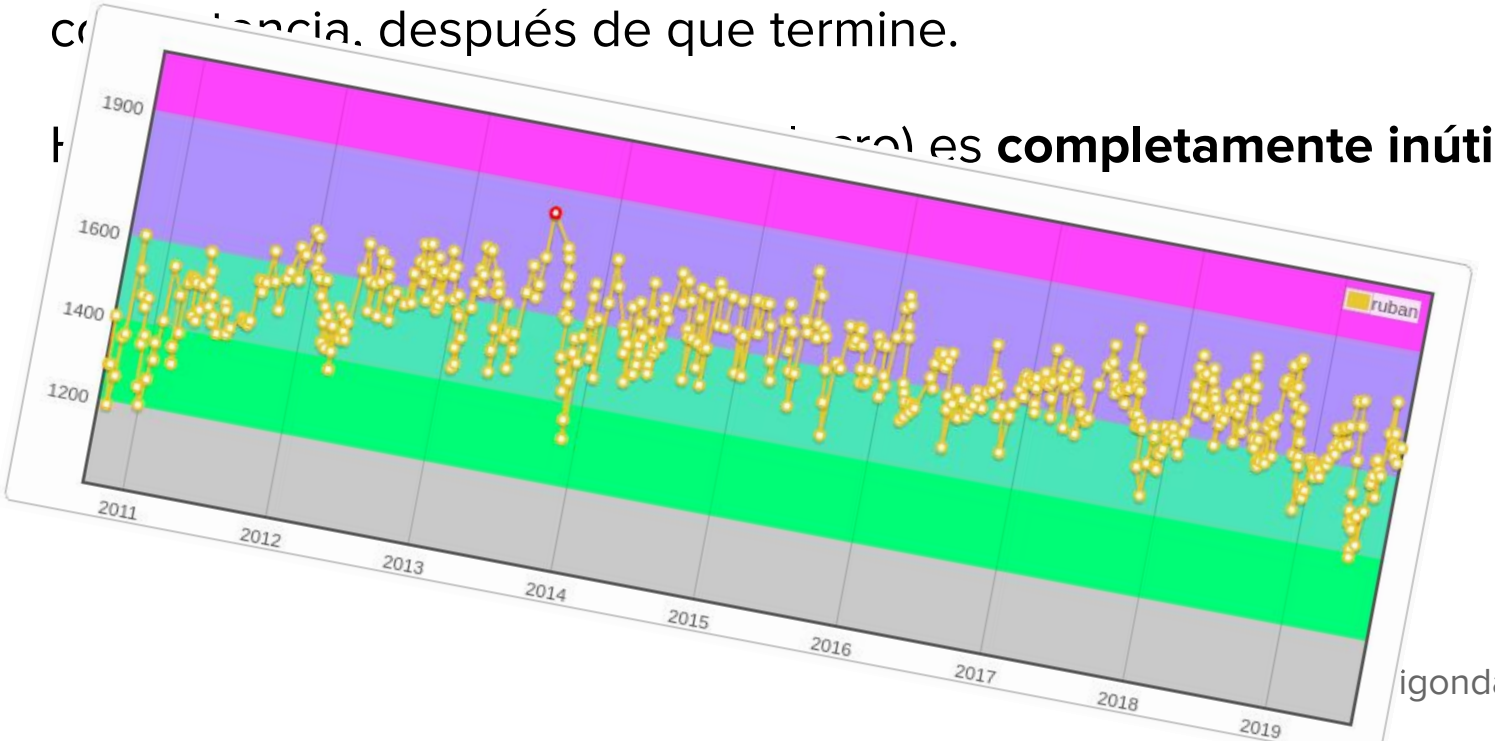
Upsolving es resolver un problema que no pudiste resolver durante la competencia, después de que termine.

Hacer una competencia (o simulacro) es **completamente inútil*** sin upsolving.

Upsolving

Upsolving es resolver un problema que no pudiste resolver durante la co[illegible]ncia, después de que termine.

H[illegible]ero) es **completamente inútil*** sin



Upsolving

En competencias/simulacros de ICPC, resolver los siguientes 2 problemas.

En competencias/simulacros individuales, resolver al menos el siguiente problema.

Resolverlos pronto, de ser posible, antes del siguiente simulacro/competencia.

Aprender lo suficiente para ser capaz de resolverlo (y a problemas parecidos) en el futuro. Si hay alguna técnica que no se conocía, buscar problemas que usen esa técnica.

Upsolving

La práctica no produce perfección, produce permanencia.

Leer editoriales y códigos para cada problema para asegurarnos que la técnica que estamos haciendo permanente es la adecuada.

Ejemplo: si resolvimos algo con compresión de componentes fuertemente conexas, pero existen soluciones con menos líneas y más sencillas usando DFS, aprender cómo se usó para poder utilizarlo en próximas ocasiones.

Tablita

Conocen “Los Simpsons”?

Para mantener la información de las simulaciones/competencias que se realizaron, utilizamos una tablita.

Tablita - Ejemplo

Fecha	Nombre de la competencia	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Score	Off Contest
1/2/2018	NEERC 13	S	S		N	U	S	O	S	S	S	S				7 11	1
1/5/2018	2018 Jiaozuo Regional	S	U	S	S	S	S		U	S						6 12	0
1/8/2018	XIX Open Cup Siberia	S	U		S	S					U	S				4 12	0

Entrenar

Si conocemos las etiquetas de un problema, tenemos demasiada información (que no va a estar en una competencia de verdad).

Si no conocemos las etiquetas de un problema, corremos el riesgo de enfrentarnos a problemas que requieran habilidades diferentes a las que queremos mejorar.

En simulacros/competencias de equipo entrenamos resolver problemas sin saber las etiquetas.

En entrenamientos individuales podemos entrenar sabiendo las etiquetas de los problemas.

Entrenar

En un simulacro/competencia en equipo suele haber pocos problemas “interesantes” (pero se entrena la dinámica del equipo).

Al entrenar debemos enfocarnos en maximizar la cantidad de problemas “interesantes” que pensamos.

Nivel 0: trivial, fácil, moderado.
Nivel 1: difícil, pero usando técnicas ya adquiridas.
Nivel 2: difícil, usando técnicas no adquiridas.
Nivel 3: difícil, imposible*.

Entrenar

En un simulacro/competencia en equipo suele haber pocos problemas “interesantes” (pero se entrena la dinámica del equipo).

Al entrenar debemos enfocarnos en maximizar la cantidad de problemas “interesantes” que pensamos.

Nivel 0: trivial, fácil, moderado.
Nivel 1: difícil, pero usando técnicas ya adquiridas.
Nivel 2: difícil, usando técnicas no adquiridas.
Nivel 3: difícil, imposible*.

Entrenar

Aprovechar datos como rating del problema (por ahora solo en Codeforces), cantidad de gente que resolvió el problema, o letra asignada (en competencias online) para estimar la dificultad del problema.

Usar cronómetro para practicar problemas fuera de competencias (guardar el tiempo que tardaste en resolver el problema junto con un pequeño resumen de la solución en una hoja de cálculo puede ser muy útil).

Al hacer simulacros de competencias en equipo, imaginar que es una final mundial.

Dudas?