Soluciones oficiales del concurso nacional de la Olimpiada de Informática 2010

Por César Cepeda (Secretario Académico del COMI) cesar@auronix.com

El concurso nacional del Olimpiada de Informática 2010 es el primer año en el que se aplica una modalidad de 4 problemas por examen para ambos días de competencia. La principal motivante para dicho cambio es la gran disparidad de nivel que existe entre los diferentes estados participantes. Esta disparidad hace muy difícil el poder presentar a todos los concursantes con problemas retadores para su respectivo nivel. Al aumentar a 4 el número de problemas por examen, se tiene un poco más de libertad en cuanto a la dificultad de los mismos, pudiendo tender a un esquema en el que se tenga un problema fácil que sea accesible para el 100% de los concursantes, un problema medio accesible a los competidores de bronce, un problema difícil accesible a los competidores de plata y un problema muy difícil el cual es accesible únicamente a los mejores 10 competidores de cada año.

Este documento contiene información sobre cómo se resuelve cada uno de los 8 problemas presentados en el concurso, además de información pertinente para cada uno de ellos. Se complementa con los archivos que contienen el texto final del examen que se presentó durante ambos días de concurso, los archivos con códigos de soluciones y los archivos con los casos de prueba de cada uno de los problemas.

Los códigos de solución, tanto para Karel como para lenguaje de alto nivel, están escritos en Pascal. La razón por la que escribo en Pascal es porque me parece un lenguaje más legible que las otras opciones y un programador de java, C o C++ no debe tener ningún problema para entenderlos, lo cual no necesariamente sucede en el sentido inverso.

Día de competencia 1. Examen de Karel

Karel es un simulador que se utiliza en la Olimpiada de Informática desde hace varios años. Cuenta con las estructuras principales de un lenguaje de programación de alto nivel como procedimientos, ciclos y bifurcaciones, pero carece de un heap de variables, funciones, estructuras, objetos y operadores aritméticos entre otros.

El objetivo de iniciar a usar Karel es por su gran poder didáctico para iniciar a la gente en la programación. En últimos años se ha tenido cada vez más la preocupación sobre el "sobre-uso" de Karel por parte de los estados. En el aspecto de que muchos estados dedican un tiempo excesivo a Karel y poco tiempo a los lenguajes de alto nivel, lo cual no debería ser así pensando que Karel es únicamente una introducción y como tal, debe eventualmente superarse para pasar a temas más complejos. Sobre este punto, creo que en efecto es un error en el que se puede caer. Por lo cual, antes de exponer las soluciones al examen de Karel, expongo los puntos para los que consideró que Karel es útil, los cuales, habiendo sido dominados por un alumno, deben ser dejados atrás para pasar a programación con lenguajes de alto nivel.

- **Solución de un problema en vez de una instancia de problema**: Una situación común con alumnos a los que se les presenta este tipo de problemas por primera vez es el confundir la solución de un problema con la solución de la instancia de ejemplo. Esto es, el alumno, la primera vez, piensa que tiene que resolver el ejemplo expuesto y tarda en comprender que lo que se le pide es una solución general que pueda resolver cualquier ejemplo que se ajuste a las características definidas. El primer uso de Karel debe ser por tanto, que el alumno entienda la diferencia entre una solución genérica y la solución de un caso específico.
- **Capacidad de poner sus ideas en código**: Las computadoras son máquinas con la capacidad de ejecutar instrucciones bien definidas en un orden establecido. Para resolver un problema utilizando computadoras es necesario que el usuario sea capaz de dar estas órdenes en una manera no ambigua. La segunda aplicación de Karel debe ser que el alumno sea capaz de pensar una idea y plasmarla en código con las instrucciones que tiene disponibles. **IMPORTANTE: Esto rara vez sucede, los alumnos suelen tender a codificar antes de tener una idea clara de lo que desean hacer. Karel es el momento para FORZARLOS a no adquirir ese mal hábito obligándolos a tener una idea bien estructurada antes de iniciar a codificarla.**
- **Estructura:** Al ser un lenguaje con varias limitaciones, en Karel, la estructura de un programa suele ser mucho más importante que en su contraparte de alto nivel. Es importante que el alumno aprenda que cosas conviene separar en funciones y como estructurar su programa para evitar el tener que escribir códigos de cientos de líneas evitando así la incidencia de errores tipográficos y de lógica.
- **Recursión:** Karel es un lenguaje que permite la recursión y muchos de los problemas que se plantean en concursos nacionales requieren en fuerte medida de ella. La recursión por su misma naturaleza, suele ser complicada de visualizar mentalmente, algunas veces es difícil incluso de diagramar. Karel es una excelente herramienta para que el alumno comprenda lo que es la recursión, las implicaciones y usos de un stack recursivo y logre tener una mejor visualización mental de sus aplicaciones.

Una vez que un alumno tiene un dominio real de los puntos anteriores, hay poca ganancia, salvo la práctica de un lenguaje muy específico, en seguir utilizando Karel. Es el momento de pasar a un lenguaje de alto nivel y a problemas de complejidad matemática más alta.

Problema 1. Rombo

Rombo es el problema fácil de Karel. Fue clasificado así por las siguientes razones:

- El texto del problema explica lo que hay que hacer, entendiendo el texto no es necesario analizar o deducir nada más antes de poder resolver el problema.
- La solución no requiere de recursión.
- El texto del problema asegura que el número de zumbadores en la mochila siempre es el exacto, por lo que no es necesario realizar ninguna comparación o validación compleja, simplemente acomodar los zumbadores como se pide.

Este problema califica si el alumno es capaza de producir una solución genérica para un problema planteado y si es capaz de pasar a código sus ideas.

Solución

El problema pide que se vayan rellenando de zumbadores las esquinas de un cuadrado, además asegura que el número de zumbadores con los que se cuenta es tal que siempre alcanzarán de manera exacta para hacer el rellenado simétrico en las cuatro esquinas.

La manera más sencilla de hacerlo es simplemente ir llenando las esquinas de manera progresiva hasta quedarse sin zumbadores. Para evitar tener que dividir en cuatro los zumbadores que tienes, basta con ir haciendo lo mismo esquina por esquina, de ese modo aseguras que las cuatro esquinas se van llenando igual.

La solución oficial rellena las esquinas llenando una "escalera" de zumbadores en cada esquina, para esto se basa en dos procedimientos.

- avanzaHastaEsquina: Hace que Karel avance de la posición donde esta hasta encontrar una pared o un zumbador. Si encuentra pared significa que la esquina está vacía y hay que comenzar a llenarla, si encuentra zumbador significa que la esquina ya tiene algunos zumbadores en cuyo caso, regresa una posición para iniciar a llenar la escalera. Algo importante a notar aquí es que si al regresar una posición se coloca en una casilla que tiene zumbador, entonces lo toma para que el procedimiento de llenar la escalera funcione correctamente.
- avanzaEscalon: Este procedimiento va haciendo una "escalera" de zumbadores y se repite mientras Karel quede en una posición donde no hay zumbador. El procedimiento verifica no chocar contra paredes y también verifica si al avanzar se "choca" contra los zumbadores de la siguiente esquina, en cualquier caso, cuando detecta que ya no puede avanzar, coloca a Karel en la posición adecuada para que Karel pueda seguir avanzando hasta la siguiente esquina.

El programa simplemente avanzaHastaEsquina y avanzaEscalon mientras tenga zumbadores en la mochila.

Problema 2. Almohada

Almohada es un problema que busca evaluar la capacidad del alumno para utilizar la recursión. Aunque seguramente habrá alguna solución que no requiera de la recursión, las soluciones que se vienen a la mente son todas recursivas.

Hay varias formas de resolver el problema, la mayoría de ellas utilizan la variable recursiva de Karel para saber cuántas fibras duras hay en la almohada. Para mostrar una forma poco usual de utilizar la recursión en Karel, la solución oficial no utiliza variables, sino la salida del stack recursivo para determinar la firmeza de la almohada.

La solución oficial realiza los siguientes pasos para resolver el problema:

- Recoge todos los zumbadores del mundo
- Los acomoda todos del lado izquierdo de la almohada
- Una vez acomodados avanza recursivamente mientras existan fibras duras en la almohada.
- Cuando se corta la recursión porque ya no hay más fibras duras, utiliza la salida de la recursión para continuar avanzando un número de pasos igual al número de fibras duras.
- Si llega al final antes de terminar la recursión, la almohada es dura, si termina la recursión y está exactamente en el final, entonces la almohada es perfecta, por último, si termina la recursión y no está en el final, entonces la almohada es suave.

De nuevo la solución oficial trata de hacer el máximo uso posible de la programación estructurada separando en funciones todo aquello que se utilice más de una vez en el programa.

Problema 3. Rebelde

Rebelde es un problema que evalúa si el alumno es capaz hacer usos complejos de la recursividad en Karel demandándole que implemente la búsqueda de un camino. Este es un problema que difícilmente se puede resolver si no se tiene algo de experiencia en Karel. Además el problema requiere de tener cuidado en la implementación para guardar el camino que llevas recorrido y no volver a pasar por él.

Este es el problema con implementación más larga del examen y en el que se tiene que tener más cuidado. Tradicionalmente el lenguaje Pascal de Karel no soporta la doble recursión, esto es, no soporta que dos funciones se llamen recursivamente una a la otra. Sin embargo el lenguaje Java de Karel si lo soporta. Aunque la doble recursión en sí misma no representa una ventaja computacional, si representa una ventaja en codificación. La siguiente versión de Karel, la cual estará disponible al público para finales del año 2010 soporta doble recursión en Pascal mediante la palabra clave *define-prototipo-instruccion*. En las soluciones oficiales hay dos archivos de solución para este problema, una utilizando la gramática actual de Karel y la otra utilizando la nueva funcionalidad. El ahorro en código gracias a la doble recursión es de un 31%.

La solución oficial hace lo siguiente:

- Recoge los zumbadores en la posición inicial y memoriza el número.
- Avanza hacia la primera casilla del mapa y comienza a buscar la salida del mapa pasando siempre a la función *buscaSalida* el número de cambios que todavía se tienen que hacer.
- En la función *buscaSalida*, si se llega a un salida y ya no es necesario hacer cambios, el programa termina.
- Si no es salida, para cada posición
    - La marca con 5 zumbadores para que si vuelve a llegar a ella no la tome en cuenta para el camino.
    - Ejecuta la función que se indica en la instrucción llamando recursivamente a *buscaSalida*
    - Posterior a ejecutar la instrucción de la posición, si aún hay cambios restantes, aplica los tres cambios posibles, dependiendo de la instrucción y llama recursivamente a *buscaSalida* indicándole que el número de cambios restantes ha disminuido en uno.
    - Al finalizar la recursión, si aún no ha encontrado la salida, regresa el número original a la posición.

Problema 4. Recuerda

Recuerda es el problema del día uno que buscaba que el alumno sea capaz de tener ideas creativas. El problema no requería ningún conocimiento especial, simplemente requería de una idea creativa. Este problema fue catalogado por el COMI y ratificado por los resultados como el problema difícil de la olimpiada, lo que demuestra que muchos alumnos carecen de la capacidad de enfrentarse a problemas novedosos que son radicalmente distintos a lo que anteriormente han visto.

El problema pide que Karel recuerde la posición donde inició pero no es posible marcarla de ninguna forma, ya que cualquier número de zumbadores que pongas puede ser un valor válido de instrucción en el mundo.

El método para resolver un problema como este radica en analizar qué es lo que se necesita realmente. Una forma de saber si volviste al lugar de inicio es contar cuantos pasos has avanzado en la dirección horizontal y cuantos has avanzado en la dirección vertical. Al contar los pasos, digamos en la dirección horizontal puedes contabilizar con un +1 cuando avances hacia el este y un -1 cuando avances hacia el oeste. Si en algún momento, después del inicio el avance en el la dirección horizontal y el avance en la dirección vertical vuelve a ser 0, en ese momento estarás de nuevo en la posición original.

Ahora… ¿Cómo se pueden llevar dos variables en Karel? Una forma sencilla es estableciendo un sistema numérico posicional como el que usamos todos los días para contar. En dicho sistema, los avances en la dirección horizontal se llevan en una posición y los avances en la dirección vertical se llevan en otra posición. Lo importante de un método así es que el valor de una posición nunca sobre escriba a la siguiente. Cómo el mundo de Karel tiene una dimensión máxima de 100x100 entonces podemos llevar el avance en la dirección horizontal en las unidades y el avance en la dirección vertical en las centenas. Iniciamos en 0 (0 unidades, 0 centenas) y si en algún momento volvemos a llegar a cero, habremos regresado a la posición original.

El dejar que Karel llevará un zumbador en la mochila al inicio del mundo fue hecho por dos razones, la primera es porque el COMI tenía varias soluciones parcialmente correctas que hacían uso de dicho zumbador y la segunda fue para confundir a los alumnos que hicieran un análisis pobre de lo que se requería para resolver el problema.

Día de competencia 2. Lenguajes de alto nivel

Los lenguajes de alto nivel, debido a las herramientas con las que cuentan permiten jugar con un rango más amplio de complejidades, no sólo entre problemas, sino en un mismo problema suelen existir varias soluciones correctas que difieren en cuanto a su complejidad computacional o a la cantidad de memoria que requieren.

Esta olimpiada decidimos no poner ningún problema cuya solución fuera trivial en el aspecto de ser puramente implementativa, esto es, en ninguno de los cuatro problemas bastaba sólo con entender lo que se pedía e implementarlo para obtener 100 puntos. Sin embargo, escogimos problemas los cuales permitían soluciones correctas con complejidades computacionales altas que si bien no obtenían el 100% de los puntos, si obtenían entre 40% y 60% dependiendo del problema. De modo que era factible obtener un score de hasta 200 puntos con ideas muy simples.

Para cada uno de los problemas del día 2 hay al menos dos implementaciones, una que da puntos parciales y la otra que da el 100% de los puntos.

Problema 1. Vueltas

Este es el problema más sencillo del día 2, y fue en el que más alumnos obtuvieron puntos. Como se esperaba, hubo una buena cantidad de alumnos con 100 puntos y una cantidad similar de alumnos con 50 puntos (eran los

obtenidos por la solución parcial). Fueron muy pocos los alumnos que obtuvieron un puntaje distinto de 100, 50 ó 0 y en esos casos se debió principalmente a errores al momento de implementar.

El problema pide que se volteé un tablero que puede ser de tamaño máximo 1000 x 1000 al cual se le puede dar un máximo de 50,000 vueltas.

Solución parcial: 50 puntos

El tablero es representable en memoria, ya que 1000x1000 son 1,000,000 de enteros los cuales, tomando enteros de 4 bytes representan un gasto total de 4MB de memoria. El límite para el problema era de 64MB, por lo que estamos muy por abajo.

La solución parcial es la que representa el tablero en memoria y va volteando el tablero conforme se le indica en el archivo de entrada. El problema con esta solución es que puede llegar a ser muy lenta. Analizando la complejidad de la misma podemos ver que en cada vuelta del tablero volteamos todas las posiciones del tablero, es decir, potencialmente 1,000,000 de asignaciones a memoria. Si repetimos esto 50,000 tenemos que nuestro programa puede realizar potencialmente hasta 50,000,000,000 de operaciones. Suponiendo que una máquina moderna puede realizar algunas decenas de millones de operaciones en un segundo, esta solución podría tardar del orden de 500,000 segundos en terminar (¡Alrededor de una semana!)

Sin embargo, en la mitad de los casos el número de vueltas no era tal que una implementación que fuera simulando todas las vueltas corre en tiempo.

Solución óptima: 100 puntos

Para la solución óptima, el alumno tiene que darse cuenta de dos cosas:

- Dar dos vueltas seguidas en un sentido (vertical u horizontal) es lo mismo que no haber dado ninguna vuelta. Esto es algo sencillo de ver, el alumno que lo note podrá darse cuenta de que en realidad sólo importa si el número de vueltas en un sentido es par (indica que no hay que dar ninguna vuelta) o impar (indica que basta dar una vuelta para llegar al estado final).
- El orden en de las vueltas es conmutativo: La primera propiedad es sencilla de ver, sin embargo convencerse de que no importa si en medio de vueltas verticales metes una horizontal por ejemplo, es algo un poco menos obvio. Menos aún el hecho de que puedes tomar todas las órdenes de vueltas, ordenarlas en el orden que quieras, aplicarlas y al final el resultado será el mismo. Una forma de verlo es darse cuenta es observar que las vueltas verticales afectan la fila final de un número, pero nunca su columna, mientras que las horizontales afectan sólo la columna, pero no la fila. Experimentando un poco, en un tiempo muy corto puede verse que el orden de las vueltas no tiene efecto sobre el resultado final.

Por tanto la solución óptima lee el tablero en memoria, cuenta el número de vueltas verticales y horizontales, si el número de vueltas verticales es impar, aplica sólo una vuelta vertical. Si el número de vueltas horizontales es impar aplica sólo una vuelta horizontal. Posteriormente escribe el estado final de la tabla.

Problema 2. Saltos

Vueltas es un problema cuya solución óptima requiere de cierto análisis matemático. Las matemáticas que se requieren son muy básicas, aritmética y nociones de teoría de números. El problema buscaba evaluar que tanta capacidad tenía el alumno de analizar matemáticamente el planteamiento de un problema.

La idea más simple para resolver el problema es guardar en memoria el tablero, ir marcando las posiciones por las que ya se pasó, simular los saltos y cuando se regrese a una posición previamente visitada, dar como respuesta el número de saltos que se han dado. Sin embargo el tablero es muy grande (10millones x 10millones) y almacenarlo en memoria para ir marcando las casillas visitadas requiere de 100TB de RAM.

El alumno debía ser capaz de darse cuenta de que la memoria requerida era excesiva y buscar una forma alterna para saber si había caído en una casilla previamente visitada.

En muchos problemas, no sólo de informática, sino de la vida. El camino más corto hacia la respuesta es plantearse la pregunta correcta. Las preguntas que debían haberse formulado en la mente del alumno debían ser las siguientes (en el orden que las escribo):

- ¿Es cierto que en algún salto voy a regresar a una casilla visitada? La respuesta afirmativa a esta pregunta se obtiene dándose cuenta de que el número de casillas en el tablero es finita, sin embargo el número de saltos que se pueden dar no, eventualmente el número de saltos excederá el número de casillas en el tablero, por lo que forzosamente se tiene que volver a caer en una casilla que ya se visitó previamente.
- OK, ya vi que en algún momento visitaré de nuevo una casilla. ¿Puedo visitar cualquier casilla de las que ya visité? La motivante de esta pregunta debe ser el hecho de darse cuenta de que para saber si ya visitaste una casilla tienes que de alguna forma mantenerla en memoria, ya que el número de casillas es, como dijimos, 100 millones de millones, esto no es posible. Antes de ponerse a almacenar, vale la pena preguntarse si en efecto es necesario recordar todas las casillas por las que se ha pasado.
    - Dado que el salto está definido, estando en cualquier casilla del tablero, sólo se puede pasar a otra casilla específica, es decir si estás en la casilla a_0 y saltas, forzosamente caerás en la casilla a_1, no hay posibilidad de cambiar la casilla en donde vas a caer ya que el salto siempre es el mismo. Más aún, si estas en la casilla a_1 el único lugar de donde podías venir era de a_0.
    - De lo anterior puede verse que si en algún momento caes en una casilla que y habías visitado, entonces hay dos opciones:
        - La casilla de donde venías también ya había sido visitada, o
        - La casilla en la que caíste es la casilla de donde empezaste.
    - Por lo tanto, la primera casilla que repetirás será siempre la primera casilla. No es necesario almacenar todas las casillas por las que has pasado, basta únicamente con fijarse si vuelves a la casilla original.

Solución parcial: 50 puntos

La solución parcial utiliza lo deducido en los puntos anteriores para ir simulando los saltos y checando si volvió a la casilla original. La razón por la que esta solución no obtiene el 100% de los puntos es porque hay tableros en los que el número de saltos es de muchos millones y la simulación tarda más tiempo del permitido.

Solución óptima: 100puntos

Un buen hábito cuando se tiene una solución es preguntarse cómo se va a desempeñar dicha solución en casos extremos. Haciendo este ejercicio es fácil encontrar ejemplos en los que el número de saltos a dar es de varios millones. Un segundo no alcanza para ejecutar muchos millones de sumas, de modo que se debe buscar una solución que haga menos cálculos.

Dado que simular es muy tardado, debe existir una forma de calcular de manera fácil de calcular en cuantos saltos se vuelve al inicio. Como siempre es necesario ir analizando las propiedades del problema.

- La primera propiedad a notar es que el salto está dividido en dos partes, la vertical y la horizontal y ambas son independientes una de la otra. Si es posible saber cada cuando el salto vertical nos lleva de regreso a la fila de origen y cada cuando el salto horizontal nos lleva a la columna de origen. Entonces sabemos que cada múltiplo de estos dos números nos llevara a la fila y columna de origen. Por lo tanto teniendo esos números basta con buscar el mínimo común múltiplo de ambos para tener la solución.

La pregunta entonces es: ¿Cada cuánto, tomando por ejemplo el salto horizontal, volvemos a la columna de origen?

- Para volver a caer en la columna de origen (pensando en los saltos horizontales), necesitamos que los saltos nos desplacen un número de columnas que sea un múltiplo de la longitud del tablero, es decir, si el tablero tuviera 1000 columnas de ancho, entonces volveremos a la columna original si avanzamos 1000 columnas, 2000 columnas, 3000 o cualquier otro múltiplo del ancho del tablero.
- Lo que hay que buscar entonces es el mínimo común múltiplo entre ambos números, la longitud del tablero y la longitud del salto y dividirlo entre la longitud del salto.

Hasta aquí ya sabemos en cuantos saltos se vuelve a la columna original y a la fila original, también sabemos que el mínimo común múltiplo de ambos será la cantidad de saltos necesaria para volver a la casilla original. El único punto que queda pendiente es ¿Cómo se calcula el mínimo común múltiplo de dos números?

Una forma es de nuevo ir incrementado cada número por sí mismo (a manera de irlo multiplicando por 1, por 2, por 3, etc.) hasta que ambos lleguen al mismo número. Ese será el mínimo común múltiplo de ambos. Aunque este algoritmo es correcto, de nuevo, puede llegar a ser lento, ya que puede implicar un gran número de sumas antes de encontrarse.

Si recordamos nuestra definición de primaria de un mínimo común múltiplo sabremos que se encuentra al factorizar en primos ambos números, tomar la intersección de primos entre ambos y multiplicar, la intersección por los primos no comunes en ambos números. Factorizar en primos es otra cosa que puede tomar mucho tiempo, para empezar, porque necesitamos tener una lista de todos los primos a probar. Sin embargo, es útil notar que la intersección de primos entre ambos números es además lo que se conoce como máximo común divisor de ambos. Si podemos calcular la intersección de primos de ambos números, podemos obtener los primos no comunes simplemente dividiendo el número original entre el máximo común divisor. En otras palabras:

Mínimo común múltiplo(a,b)=a * b / máximo común divisor(a,b)

Obtener el máximo común divisor de ambos números es algo computacionalmente más sencillo. La forma más común es utilizar el algoritmo de Euclides, conocido desde hace más de 2000 años. El algoritmo se basa en la siguiente característica:

- Sean ***a*** y ***b*** dos números enteros donde ***a*** < ***b***. Al dividir ***b***/***a*** se obtiene un cociente ***q*** y un residuo ***r***.
- El fundamento del algoritmo de Euclides se basa en que el máximo común divisor de ***a*** y ***b*** es el mismo que el máximo común divisor entre ***a*** y ***r***.

Demostrar que el fundamento del algoritmo de Euclides es cierto es relativamente sencillo y para no extenderme demasiado en este problema queda como tarea para el lector. Como pista sugiero un método visual (con varitas) donde el problema se traduce en ver ¿Cuál es la varita más grande que cabe de manera exacta dentro de las otras dos varitas? Y tratar de ver por qué es necesario que el mínimo común divisor de ***a*** y de ***b*** divida también de manera exacta a ***r***.

Como última nota para este problema, es necesario observar que en algunas multiplicaciones un entero de 32 bits puede desbordarse, de modo que es necesario utilizar un tipo de datos de 64 bits para asegurar que el problema se resuelve correctamente en el 100% de los casos.

Problema 3. Espías

Espías es un problema que evalúa si el alumno es capaz de modelar el problema planteado como una representación útil computacionalmente. En este caso, un grafo dirigido. No es necesario que el alumno sepa lo que es un grafo dirigido, simplemente que sea capaz de modelar el problema como tal para poder darse cuenta de las propiedades del mismo.

Además, puede ser útil (no necesario) el conocer una de las estructuras de datos más sencillas que existen, la pila. El uso de la pila sirve únicamente para disminuir el factor de complejidad computacional de la solución, sin embargo se pueden obtener los 100 puntos sin necesidad de utilizar una pila.

Solución parcial: 40 puntos

Al igual que con los problemas anteriores hay una solución parcial que obtenía 40 puntos la cual es muy sencilla de implementar. Para obtenerla basta con simular el camino de mentes que lee el lector iniciando desde cada uno de los espías y almacenar el mejor para entregarlo posteriormente como resultado.

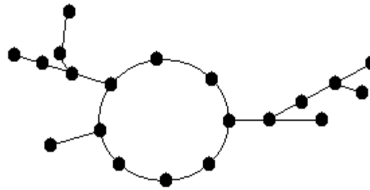
Solución óptima: 100 puntos

El problema es claramente un grafo dirigido donde cada espía es un vértice y cada relación de espías una arista que va de que espía al espiado. De cada vértice sale una y sólo una arista, de modo que el grafo tiene ***E*** aristas y ***E*** vértices.

Para resolver el problema, el alumno debe darse cuenta de que el hecho de que cada espía espíe a otro da como resultado que cualquier secuencia de mentes leídas termina siempre en un ciclo, es decir, iniciando la lectura desde cualquier espía, podemos seguir leyendo mentes hasta que regresemos a un espía cuya mente ya habíamos leído. No sólo eso, como cada espía espía a otro, es imposible detenernos antes de que eso pase.

También es importante ver que una vez que se entra a un ciclo, es imposible salir de él, y que iniciar la lectura desde cualquiera de los espías del ciclo nos da como resultado exactamente el mismo número de mentes leídas, el número de espías en el ciclo.

Lo anterior nos deja el problema como un conjunto de grafos dirigidos disjuntos, en los que se tiene un ciclo con varias ramas que entran al ciclo (ver figura). El máximo para cada uno de estos grafos será el número de espías en el ciclo mas el número de espías en la rama más larga. El máximo total para el problema será el máximo de los máximos de cada grafo.



Hay varias formas de buscar los ciclos en un grafo de este tipo. Lo importante al implementar la solución es notar que en la solución parcial se está visitando cada ciclo muchas veces, lo que hace que la complejidad del algoritmo potencialmente sea E^2**,** la idea tras la solución óptima es contar cada ciclo sólo una vez, cuando ya se detectó un ciclo una vez, sabemos que la cantidad de mentes que puede leer cualquier espía de ese ciclo es igual al número de espías en el ciclo por lo que no es necesario volver a revisar ninguno de los espías de ese ciclo. Si además, para los espías en las ramas vamos guardando su distancia al ciclo, o la suma de su distancia y la longitud del ciclo, entonces cada que lleguemos a un espía que ya habíamos visto previamente no es necesario seguir adelante, ya que sabemos cuántas mentes hay hacia adelante. De esta forma revisamos a cada espía sólo una vez y nuestro algoritmo se vuelve de complejidad proporcional a E.

Cuando llegamos a un espía nuevo, no sabemos cuántos espías hay adelante, de modo que la solución óptima utiliza una pila para recordar los espías por los que pasó y al llegar al final puede ir sacando a los espías en el orden en el que los visitó y actualizando el máximo de mentes que puede leer cada uno de ellos.

Problema 4. Antenas

Antenas es un problema en dónde se esperaba una solución de aproximación o alguna heurística que buscara optimizar lo más posible un resultado. Antenas es una variante de un problema **NP-completo** conocido *mínimum cardinality set cover* que se formula de la siguiente manera:

- Dado un universo U de n elementos y una colección de subconjutos de U, $S = \{S_1, S_2, \ldots, S_k\}$, encuentra una subcolección de cardinalidad mínima S que contenga a todos los elementos de U.

Es decir, dados n elementos y una colección de conjuntos de esos elementos, encuentra cual es el mínimo número de conjuntos que debes tomar para que los n elementos aparezcan en al menos uno de los conjuntos.

En el caso de antenas, se pueden tomar las n casillas no cubiertas de la ciudad como los elementos de U y todos los lugares posibles para poner una antena como los conjuntos de S donde cada conjunto incluye a los

elementos que alcanza según su cobertura. Si se puede resolver el *mínimum cardinality set cover* entonces tendremos una solución óptima para antenas.

Antenas es un caso particular, ya que la cardinalidad de los ***S*** conjuntos es fija y más aún, la distancia entre los elementos de un mismo conjunto S_i está limitada por una constante. Estas características hacen que obtener un buen factor de aproximación sea más sencillo que en el caso genérico del problema. De hecho la variante en la que la cardinalidad de cada conjunto en ***S*** está limitada por una constante ***k*** se conoce como *mínimum k-set cover* y cae dentro de la familia de los problemas **APX**, esto es, se puede demostrar que hay un algoritmo ejecutable en tiempo polinomial para encontrar una aproximación a la solución óptima cuyo factor está limitado por una constante. Hay una gran información en internet sobre cualquiera de estos temas si el lector desea ahondar en cualquiera de ellos.

No esperamos que ningún alumno tenga conocimientos profundos de complejidad computacional, problemas **NP**, problemas **APX**, algoritmos de aproximación o cuestiones similares en la fase nacional de la Olimpiada. Por esta razón se decidió optar por poner instancias pequeñas del problema y calificar a los alumnos comparando su resultado con los resultados de otros alumnos y con una heurística del propio comité. A continuación expongo varias ideas sencillas y los resultados aproximados que cada una hubiera obtenido.

Heurística 1. Pon la antena en el primer lugar que puedas.

Esta heurística es la más simple. Lee el mapa de la ciudad, coloca las antenas iniciales, y posteriormente ve recorriendo la ciudad cuadro por cuadro, si encuentras un cuadro vacío, pon ahí una antena y marca como cubiertos los cuadros vacios que caen dentro de la cobertura de la antena.

Esta solución hubiera obtenido 37 puntos.

Heurística 2. Pon la esquina superior de la antena en el primer lugar que puedas

Queda claro que la heurística uno desperdicia bastante área de cobertura de cada antena, ya que estamos recorriendo la ciudad de arriba abajo, de izquierda a derecha, cuando ponemos la antena centrada en el primer cuadro vacío estamos desperdiciando al menos la mitad superior del área de cobertura de la antena y en muchos casos la mitad izquierda de la mitad inferior, es decir estamos utilizando en la mayoría de las veces solo la cuarta parte de la cobertura de la antena.

La segunda heurística entonces es, cada que se encuentre un cuadro vacío, se coloca una antena de modo que la esquina superior izquierda del área de cobertura de la antena quede en el cuadro vacío.

Esta simple heurística hubiera obtenido 90 puntos.

Heurística 3. Busca la mejor columna para colocar la antena

Optimizando aún más la idea de la primera heurística tenemos que al haber puesto la esquina superior izquierda del área de cobertura de la antena en el primer cuadro vacío, dejamos de desperdiciar el área superior de la cobertura, sin embargo, es posible que dejemos un cuadro vacío que estuviera en el cuadrante inferior izquierdo sin cubrir obligándonos a poner una antena más posteriormente.

La tercera heurística busca el primer cuadro vacío, una vez que lo encuentra en vez de poner la esquina superior izquierda del área de cobertura en ese cuadro, prueba poner la antena en todas las columnas posibles, es decir

pone la esquina superior izquierda del área en el cuadro y luego va “recorriendo” la antena hacia la izquierda hasta que la esquina superior derecha de la antena está en el cuadro. Para cada posible colocación guarda cuantos cuadros vacíos cubrió y al final se queda con la posición que cubre más cuadros y deja la antena más hacia la derecha.

Esta heurística hubiera obtenido 99.31 puntos.

Esta heurística fue la que se utilizó como heurística del comité.

Existen muchas otras líneas para seguir en cuanto a heurísticas para este problema, también existen métodos para asegurar el factor de aproximación que siguen siendo polinomiales. La idea principal de este problema era que los alumnos *perdieran el miedo* a probar ideas. Agrego dos heurísticas más que continúan la optimización de la idea original. Queda como ejercicio para el lector buscar ideas distintas de aproximación al problema.

Heurística 4. ¿Porqué quedarse a la derecha?

En la heurística 3 buscamos la mejor posición para colocar la antena y de entre las mejores posiciones que nos daban el mismo máximo tomamos la que estaba más a la derecha. La pregunta entonces es ¿Realmente es lo mejor quedarse a la derecha? ¿Qué sucede si nos quedamos a la izquierda?

Si se modifica el programa para qué deje la antena lo más a la izquierda posible, se ejecuta con algunos casos de prueba aleatorios y se comparan los resultados con los obtenidos cuando se deja la antena a la derecha, se observa que los resultados varían. Hay casos en los que se obtiene un mejor resultado dejándola a la derecha y casos en los que se obtiene un mejor resultado dejándola a la izquierda.

La heurística 4 prueba ambas opciones. Resuelve el problema dejando las antenas lo más a la derecha posible y guarda el resultado. Posteriormente lo resuelve dejando las antenas lo más a la izquierda. Compara ambos resultados y entrega el mejor de los dos.

Esta heurística hubiera obtenido 99.65 puntos y hubiera hecho bajar el puntaje del resto de las soluciones en 6 casos, ya que para estos encuentra una solución mejor que todas las anteriormente encontradas.

Heurística 5. ¿Porqué quedarse en algún lado en específico?

Cuando se está haciendo una heurística de aproximación, es muy importante exprimir al máximo todas las propiedades observadas del problema. Ya hemos visto que hay casos para los que es mejor quedarse a la derecha y casos para los que es mejor quedarse a la izquierda. Pero, porque quedarse siempre del mismo lado, dentro de la misma instancia de un problema debe haber situaciones en las que la mejor opción era la derecha y situaciones en las que la mejor opción era la izquierda.

La heurística 5 extiende la heurística 4 (es importante notar como hemos ido extendiendo la idea inicial de modo que cada nueva heurística nos asegura un resultado, al menos tan bueno como el anterior) calculando, además del caso con todas las antenas a la izquierda o todas a la derecha, varios casos en los que antena por antena se decide de manera aleatoria si se deja a la izquierda o a la derecha.

Las reglas de las olimpiadas internacionales requieren que un programa sea determinista, es decir, que para una cierta entrada, entregue siempre la misma salida. Por lo anterior, cuando se desea utilizar algoritmos aleatorios

es necesario inicializar la semilla a un número constante, de modo que la secuencia de números aleatorios sea siempre la misma. Evaluando la complejidad de la solución y por motivos del bicentenario de México, decidí correr la heurística 5 con semillas iniciales que van desde el 1810 hasta el 2010 ☺.

Esta heurística hubiera obtenido 99.7 puntos, sólo es derrotada en 2 casos, en ambos por la diferencia mínima y encuentra soluciones mejores a todas las conocidas en 12 de los 25 casos de prueba, en algunos de ellos por un margen amplio.

Conclusiones

Creo que el examen nacional de la OMI 2010 fue un examen muy bien equilibrado en lo que se refiere a grado de dificultad y temas vistos. El examen ofrecía una amplia variedad que hacía que todos los alumnos, desde los mejor preparados hasta los que llevaban poca preparación encontrarán algo accesible a su capacidad y algo retador.

La modalidad de 4 problemas en vez de 3 resulto ser buena, aunque no hubo calificaciones perfectas en ninguno de los días, los mejores estudiantes obtuvieron puntajes totales superiores a los 600 puntos, que significan puntajes promedio mayores a 300 puntos por día.

Considero que hace mucha falta en la preparación de los alumnos en lenguajes de alto nivel, ya que aunque la complejidad de los problemas no es mucho más alta que en Karel, los resultados son en general mucho más bajos. Exhorto pues a los entrenadores estatales a que dediquen más tiempo a la preparación en este rubro.