1. Nombre y número del problema a resolver

* Nombre del problema: **Spaghetti**
* Número del problema: **10151**

2. Miembros del equipo

* Carlos Domínguez García (alu0100966589@ull.edu.es) / **Coordinador**
* Cristian Manuel Abrante Dorta (alu0100945850@ull.edu.es)
* Daute Rodríguez Rodríguez (alu0100973914@ull.edu.es)

3. Informe del trabajo realizado

3.1. Descripción del problema

El problema al que nos enfrentamos se llama Spaghetti, se trata del problema número 10151 dentro de los problemas de entrenamiento del Concurso de Programación ACM-ICPC. Consiste en determinar si dos programas en Fortran son equivalentes, es decir, si para cualquier entrada ejecutan exactamente la misma secuencia de sentencias ignorando los goto incondicionales y las posibles etiquetas de las líneas del programa. Con la misma secuencia de sentencias queremos decir sentencias que son textualmente idénticas una vez se han eliminado los espacios en blanco y las etiquetas.

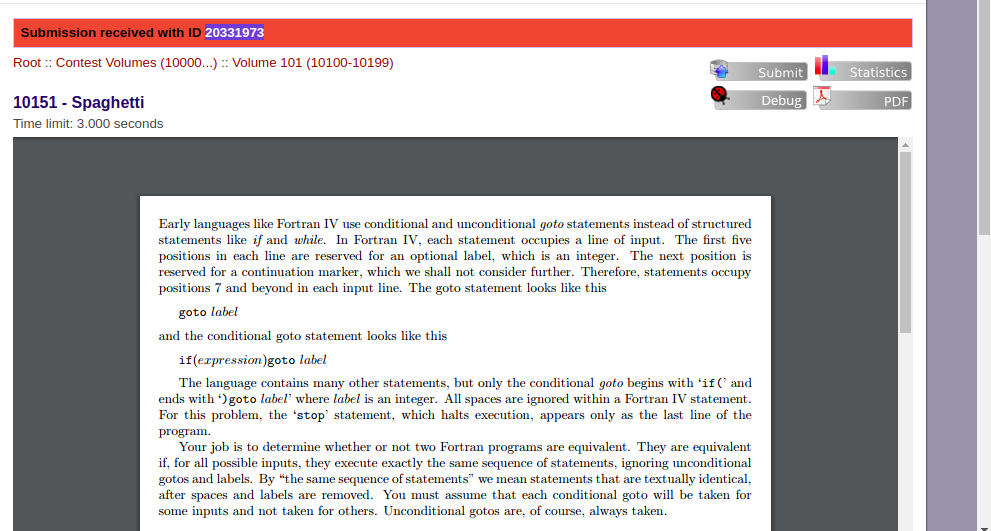
La entrada a la solución presentada consiste en varias parejas de programas separadas por una línea en blanco, cada pareja está separada de sus adyacentes por otra línea en blanco. Nuestra solución debía dar como salida una simple cadena de caracteres por cada una de las parejas de programas de entrada:

o “The programs are equivalent.” -> Si los programas de la pareja son equivalentes.

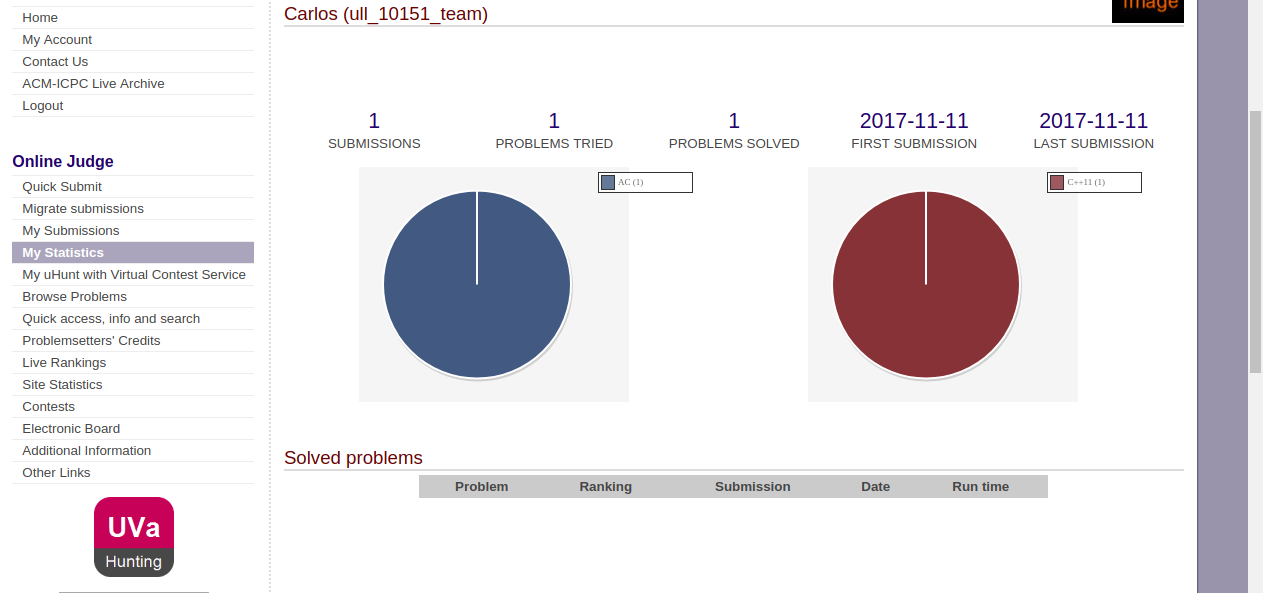
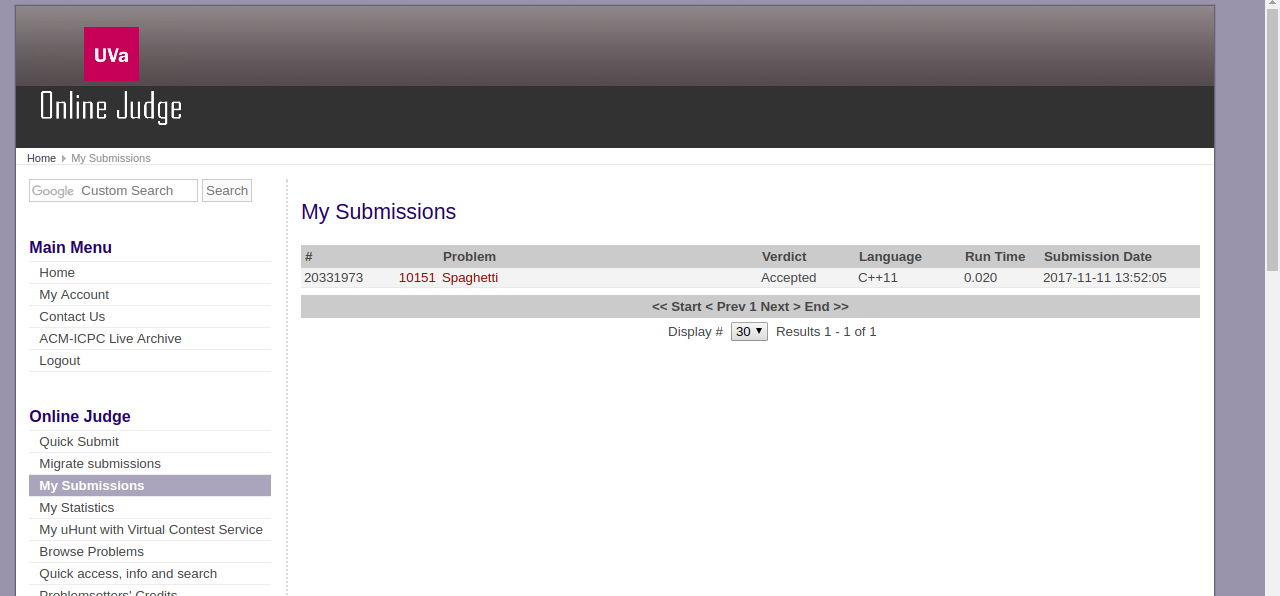
o “The programs are not equivalent” -> Si los programas de la pareja no son equivalentes.

3.2. Tiempo de resolución en la base de datos

Cuando el coordinador del grupo hizo el envío de nuestra solución, apareció un mensaje en el que se nos indicaba el ID de nuestra solución propuesta:



Tras esto el coordinador fue al perfil de la cuenta creada y vio que la propuesta había sido aceptada y el tiempo que había tardado en resolver el problema:



Tras unos minutos se actualizó el ranking y el coordinador pudo ver cómo nuestro equipo aparecía en el top:



La primera y única solución propuesta por nuestro equipo tuvo como tiempo de resolución 0,020 segundos siendo la segunda mejor solución enviada hasta el momento:

3.3. Descripción de la implementación propuesta

Desde que comenzamos a estudiar el problema se nos ocurrió que una posible solución sería representar los programas que teníamos que comparar como grafos, y tras esto, solamente deberíamos recorrerlos de igual manera e ir determinando si los nodos (sentencias de un programa en nuestro problema) que íbamos visitando eran iguales o no. Nos dimos cuenta de que con esta aproximación podíamos tener en cuenta las distintas “bifurcaciones” que se generan en la ejecución de un programa cada vez que se alcanza un goto condicional. Una vez que concluimos que de esta forma podría resolverse el problema propuesto comenzamos a desarrollar el programa:

1. Durante la primera fase de desarrollo creamos las funciones y estructuras de datos necesarias para poder representar cada sentencia como un nodo, cada línea como una etiqueta y un nodo y cada programa como un conjunto de líneas, es decir, las funciones y estructuras de datos encargadas de “preparar” los programas pasados para que puedan ser comparados.

2. En la segunda y última fase de desarrollo implementamos el algoritmo para comparar los programas pasados, y el programa principal que es el encargado de transformar todas las parejas de programas recibidas como entrada para poder compararlos adecuadamente.

3.3.1 Estructuras de datos utilizadas

• Node -> Esta estructura representa una sentencia dentro de un programa, contiene el tipo de sentencia, la sentencia en sí y la etiqueta de la sentencia (en caso de que la tuviese, solo para sentencias goto) de la sentencia a la que representa.

• Line -> La estructura Line está compuesta por un Node y una etiqueta. Representa a las líneas de un programa en Fortran, que pueden o no tener etiqueta asociada, por lo que esta estructura de datos no siempre almacenará una etiqueta.

• Program -> Esta estructura representa un programa de Fortran, contiene un vector con las líneas (Line) que conforman el programa, así como un mapa de equivalencias entre etiquetas y posiciones de dicho vector.

3.3.2 Funciones utilizadas

• parseInt -> Dado un string un inicio y un final, devuelve la conversión de la subcadena [inicio, final] a entero. La utilizamos para obtener las etiquetas de las líneas del programa.

• eraseWhiteSpaces -> Dado un string lo modifica de manera que elimina todos los espacios que hubiera en él. La utilizamos para poder llevar a cabo las comparaciones entre sentencias, recordemos que las sentencias deben ser textualmente iguales tras eliminar los espacios en blanco y las etiquetas.

• interpretIfStament -> Dado un Node, interpreta si la sentencia que almacena es un goto condicional. En caso afirmativo establece el tipo del nodo pasado a conditional.

• interpretGotoStament -> Dado un Node, interpreta si la sentencia que almacena es un goto incondicional. En caso afirmativo establece el tipo del nodo pasado a conditional.

• interpretStopStatement -> Dado un Node, interpreta si la sentencia que almacena es la sentencia stop de fin de programa. En caso afirmativo establece el tipo del nodo pasado a stop.

• interpretOtherStatment -> Dado un Node, establece su tipo a other.

• interpretStatement -> Dado un Node, llama a las 4 funciones anteriores en el orden adecuado para establecer correctamente el tipo del nodo pasado.

• parseLine -> Dada una línea de un programa (string), la transforma a un objeto de la estructura de datos Line. Almacenando su etiqueta si es que la tiene y su sentencia (node), eliminando los espacios en blanco y estableciendo su tipo de manera adecuada.

• loopChecker -> Dado un Program y una línea determinada del mismo, determina si existe un bucle infinito de gotos incondicionales. Se trata de una función auxiliar utilizada en el algoritmo de comparación de programas.

• comparePrograms -> Dados dos Program determina si son equivalentes o no. En esta función se encuentra el algoritmo de comparación de programas en Fortran.

• main -> Se trata de la función principal del programa, la cual se encarga de preparar las parejas de programas pasadas como entrada y de invocar al algoritmo comparador mostrando por pantalla el resultado de la ejecución del mismo.

4. Grafo de confirmaciones

**master** C1 - C2

\

**development** C3 --- C5' - C6 - C7' - C8 - C9 – C11 – C12 – C13

\ / \ / \ \

**test** C4 - C5 C7 C10 C14

\

**bundle** C15

* **C1:** Creando la estructura
* **C2:** Primera versión
* **C3**: Added parseInt and eraseWhiteSpaces functions
* **C4**: Added tests for eraseWhiteSpaces and parseInt. Changed string argument from parse int to be a constant reference. Added .out files to .gitignore
* **C5**: Forgot to add this in the last commit
* **C6**: Added node struct and functions to determine node/statement type
* **C7**: Added tests for interpret statements functions and a new one for parseInt. We have to redo parseInt
* **C8**: Redone parseInt
* **C9**: Added parseLine function
* **C10**: Added some unit tests for parseLine function
* **C11**: Makefile of the project added. Also main function implemented. Added program-function.hpp
* **C12**: First version of the algorithm implemented.
* **C13**: Main function changed
* **C14**: Added 3 couple of program tests more
* **C15**: Created bundle code to submit it. And added screenshots of the submition result :D