Resolución del problema spaghetti (ACM-ICPC problema 10151)

Objetivo: El principal objetivo de este proyecto es la resolución del problema número 10151 dentro de los programas de entrenamiento del Concurso de Programación ACM-ICPC. Dicho problema consiste en determinar si dos programas escritos en Fortran son equivalentes, es decir si para cualquier entrada ejecutan exactamente la misma secuencia de sentencias.

Introducción

El problema que hemos resuelto, parte de la base que en los primeros lenguajes de programación como Fortran usan sentencias *goto* incondicionales y condicionales, en lugar de bucles como *for* y *while*. Por tanto, debemos determinar si dos programas escritos 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 a su vez 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:

- **The programs are equivalent**: Si los programas de la pareja son equivalentes.
- **The programs are not equivalent**: Si los programas de la pareja no son equivalentes.

Métodos

Algoritmo de resolución del problema

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 podríamos tener en cuenta las distintas “bifurcaciones” que se generan en la ejecución de un programa cada vez que se alcanza un *goto* condicional, puesto que debemos comprobar todos los posibles caminos que existen.

Implementación del algoritmo en C++

El lenguaje de programación que utilizamos fue C++ y concretamente utilizamos el estándar C++11. La implementación del algoritmo se llevó a cabo en varias fases:

- 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.
- 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.

Estructuras de datos utilizadas

Las estructuras de datos que hemos utilizado son las siguientes:

- **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 el programa tendría que ir, cambiando el flujo de ejecución.
- **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.

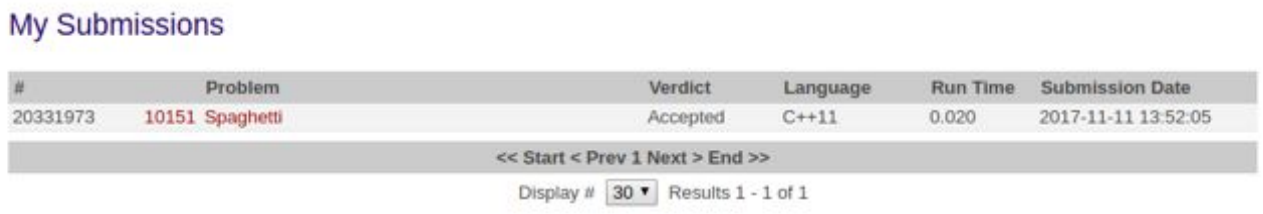
Funciones utilizadas

- **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.

Resultados

El sistema de entrenamiento del concurso ACM-ICPC tiene una base de datos en la que se debe subir el resultado de la implementación en un fichero *bundle.cpp*. Una vez el coordinador del grupo se hubo registrado, procedimos a subir el fichero correspondiente. Automáticamente, la página nos devolvió una tabla con los resultados de nuestro problema:


My Submissions

#	Problem	Verdict	Language	Run Time	Submission Date
20331973	10151 Spaghetti	Accepted	C++11	0.020	2017-11-11 13:52:05

<< Start < Prev 1 Next > End >>

Display # 30 Results 1 - 1 of 1

Figura 1: resultado de la base de datos tras subir el problema.

Como podemos ver, el sistema aceptó la solución de nuestro problema, dando como veredicto **Accepted**, y con un tiempo de ejecución de **0.020**.

Además, como el tiempo de ejecución de nuestro programa fue muy bajo, nos otorgó la segunda posición en el ránking global de intentos que ha tenido la base de datos para este problema concreto.

Total Submissions	Users that tried it	Users that solved it
383	98	33

Your best accepted try				
Ranking	Submission	Run Time	Language	Submission Date
2	20331973	0.020	C++11	2017-11-11 13:52:05

Top 20

Ranking	Submission	User	Run Time	Language	Submission Date
1	20122204	Anton Gefvert	0.000	C++11	2017-10-05 23:06:30
2	20331973	Carlos	0.020	C++11	2017-11-11 13:52:05
3	1739217	ACM_Competition_Test_Acct	0.027	ANSI C	2003-07-22 10:52:48
4	9388652	Jialin Ouyang	0.036	C++	2011-10-19 18:34:05
5	6068951	try	0.040	C++	2007-11-22 14:43:13
6	8341165	Krzysztof Stencel	0.040	C++	2010-10-22 17:58:14
7	8378142	Fredrik Svensson	0.044	C++	2010-11-06 11:56:54
8	1592211	Harry H	0.045	C++	2003-05-16 15:54:46
9	18328578	Chi-Yung Tse	0.050	C++11	2016-11-08 14:25:33
10	5909482	Josh Bao	0.060	C++	2007-09-10 06:59:02
11	2830765	joachim wulff	0.066	ANSI C	2004-09-06 10:19:34
12	9633869	Brian Fry	0.068	C++	2012-01-10 20:56:55
13	791334	Jimmy Mårdell	0.080	ANSI C	2002-03-20 18:31:42
14	8131395	Robert Gerbicz	0.084	C++	2010-07-28 18:39:37
15	6964040	Mark Greve	0.100	C++	2009-02-23 15:17:59
16	2770672	Per Austrin	0.113	C++	2004-08-16 17:46:43
17	5832261	Thiện Chí Nguyễn	0.189	C++	2007-08-15 02:32:05
18	6342626	Konstantin Lopyrev	0.190	C++	2008-03-29 10:23:02
19	4561537	Bal4u	0.191	ANSI C	2006-05-09 14:55:50

Figura 2: Nuestra solución aparece en el top de mejores soluciones (puesto 2)

Conclusiones

Podemos concluir que la solución que hemos propuesto para resolver el problema es por una parte satisfactoria, puesto que ha conseguido pasar todas las pruebas a las que se le ha sometido, y por otra parte podemos determinar que se trata de una solución eficiente, ya que ha obtenido un muy buen tiempo de resolución en la base de datos del problema. Situándose en el segundo puesto del ránking global.

La eficiencia de nuestra solución no solo depende de la optimización del algoritmo que hemos utilizado, sino también del lenguaje y el estándar elegido. C++11 ofrece muchas ventajas de eficiencia que dan muy buenos resultados, acelerando la ejecución del programa.

A la hora de implementar la solución, la principal conclusión extraída es que muchas veces no es necesario buscar la solución más eficiente en la primera aproximación. A veces, es mucho mejor intentar dividir el problema e ir solucionando pequeñas porciones del mismo para finalmente dar lugar a una solución final mucho más adecuada.

Bibliografía

- **Concurso de programación ACM-ICPC:** https://icpc.baylor.edu
- **Problema 10151 (Spaghetti):** https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=13&page=show_problem&problem=1092
- **C++ Reference:** http://www.cplusplus.com/reference/