Trabajo Práctico Final: Hackeando a Evil Corp.

Estructuras de Datos
Tecnicatura en Programación Informática
Universidad Nacional de Quilmes

1. Introducción

Mr. Robot[1] lo ha logrado. Finalmente pudo acceder a los servidores de Evil Corp. y obtener la base de clientes asociados a la corporación. Pero como todo listado de clientes que se precie de serlo, la clave de cada cliente está encriptada. Sin embargo Mr. Robot es un hacker extremadamente competente: descubrió que la encriptación usa MD5 y conoce la longitud y el conjunto de caracteres (el vocabulario) que componen las claves. Con esta información es posible intentar un ataque por *fuerza bruta*: para toda clave encriptada x del listado, generar utilizando el vocabulario todas las claves posibles de un largo dado, encriptarlas utilizando MD5, y comprobar si el resultado de la encriptación coincide con x. Desafortunadamente Mr. Robot sabe mucho de firewalls, troyanos, y de vulnerabilidades de Linux; pero poco de algoritmos y estructuras de datos. Debemos ayudar a Mr. Robot a implementar el ataque por fuerza bruta y terminar con Evil Corp. para siempre.

Para implementar la solución tenemos que ser capaces de generar todas las palabras posibles para un vocabulario y longitud dados. Lo haremos utilizando un árbol general (cada nodo puede tener múltiples hijos) donde por cada nivel se enumeran todas las palabras posibles con longitud igual a la profundidad del nivel. A su vez, cada nodo agrega un elemento del vocabulario al valor del nodo padre. La figura 1 muestra el árbol para el vocabulario $\{a, b, c\}$ y longitud dos.

La raíz (nivel cero) contiene todas las palabras posibles de largo cero, esto es, sólo el string vacío (denotado en el diagrama con el símbolo $\bullet$). En el nivel uno están todas las palabras posibles de largo uno (a, b, y c). Finalmente, el nivel dos lista todas las palabras posibles de largo dos (aa, ab, ac, ba, bb, bc, ca, cb, y cc). Podemos ver que el último nivel del árbol contiene las palabras buscadas.

2. Ejercicios

La resolución del trabajo práctico comprende los siguientes ejercicios a resolver en el lenguaje de programación C++:

[1] Si no viste Mr. Robot, ¡mirala! Es la serie más realista sobre hackers que anda dando vueltas.

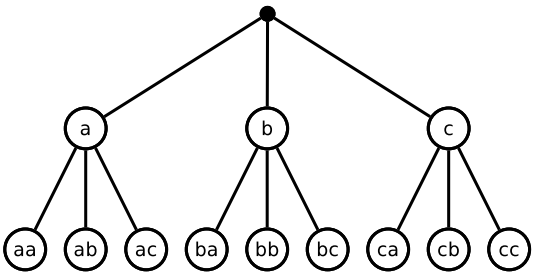

Figura 1: Palabras posibles de largo dos para el vocabulario $\{a, b, c\}$

Ejercicio 1. Primero vamos a implementar el tipo abstracto árbol general `GenTree` en los archivos `gentree.h` y `gentree.cpp`, definiendo (si las hay) las precondiciones para cada operación. Los árboles generales contendrán valores de tipo `string` en los nodos, e implementarán las siguientes operaciones:

- `GenTree leaf(string s)`, que crea una hoja con un string dado.
- `bool isLeaf(GenTree t)`, que retorna si el árbol general `t` es o no una hoja.
- `string value(GenTree t)`, que retorna el string almacenado en la raíz del árbol general `t`.
- `ArrayList children(GenTree t)`, que retorna la lista de hijos del nodo raíz del árbol general `t`.
- `void addChild(GenTree& t, GenTree child)`, que agrega el árbol general `child` como último hijo de la raíz del árbol general `t`.
- `void destroyTree(GenTree& t)`, que libera la memoria ocupada por todos los nodos del árbol general árbol general `t`.

Nota: Esto es simplemente un árbol general sin ninguna restricción en la cantidad de hijos. El TP luego va a usar este tipo abstracto para formar árboles con ciertas características.

Ejercicio 2. Completar todas las funciones sobre árboles generales del archivo `test_tree.cpp` y generar casos de prueba para validar su correcto funcionamiento.

Ejercicio 3. Completar en el archivo `main.cpp` las siguientes funciones:

1. `gentree generate(string vocab[], int vocabSize, int len)`, que dado un array con caracteres determinando un vocabulario, un tamaño de alfabeto y la longitud de las palabras a generar, retorna todas las palabras posibles.
2. `List allPasswords(GenTree t)`, que dado un árbol general retorna en una lista con todas las palabras del último nivel.

Ayuda 1: Construir el árbol desde la raíz hacia las hojas. La idea es definir un procedimiento recursivo que dado un nodo n en el nivel i, genere sus hijos de la siguiente manera. Para cada elemento v del vocabulario:

- Computar un valor s igual a la concatenación del valor de n y v.
- Crear un nuevo nodo n_s con el valor s obtenido en el paso anterior.
- Agregar a n_s como hijo de n.
- Generar recursivamente los hijos de n_s en el nivel $i + 1$.

El procedimiento comienza con el nodo raíz, con valor igual a string vacío y en el nivel cero. La recursión finaliza cuando el nivel del nodo a procesar es igual al largo máximo de las palabras a generar.

Ayuda 2: Para acumular los valores de las hojas en una lista, debemos implementar un recorrido sobre el árbol que dado un nodo n:

- Si n es una hoja, agrega su valor al vector acumulando los resultados.
- En caso contrario recorrer recursivamente cada subárbol de n.

Ayuda 3: Considerá implementar la recursión en funciones auxiliares.

3. Entrega

3.1. Archivos a entregar

Junto a este enunciado proveemos los siguientes archivos:

- `array_list.h` y `array_list.cpp`, que implementan `ArrayList` y no deben ser modificados.
- `linked_list.h` y `linked_list.cpp`, que implementan `List` y no deben ser modificados.
- `gentree.h` y `gentree.cpp`, que deben ser completados con la implementación del tipo abstracto `GenTree` como define el Ejercicio 1.
- `test_tree.h` y `test_tree.cpp`, que deben ser completados con la implementación de diversas funciones sobre `GenTree`, como define el Ejercicio 2.
- `main.cpp`, en donde *sólo* debemos completar `generate` y `all_passwords` como define el Ejercicio 3. El resto del código de `main.cpp` se encarga de invocar a dichas funciones sobre datos de prueba, reportando si el programa retorna los resultados esperados.
- `md5.h`, que implementa MD5 hashing y no debe ser modificado.

Importante: No debes crear ningún otro archivo para la entrega de este TP.

3.2. Forma de entrega

La entrega consiste en enviar un mail a la dirección de la cátedra[2] con asunto `Entrega <nombre> <apellido> ED 201s2`. Por ejemplo, mi asunto sería `Entrega Federico Sawady ED 2018s2`. En dicho mail debe adjuntarse un archivo zip con una carpeta que contenga los archivos `.h`, `.cpp` y `.cbp` del TP, de modo que al ser ejecutado, el programa reporte a todos los casos de prueba como exitosos. El archivo debe llamarse `tp_<inicial-apellido>.zip`. Por ejemplo, mi entrega sería `tp_fsawady.zip`.

Importante: Cualquier entrega que no cumpla con estos requisitos será ignorada, y el trabajo será considerado *no entregado*.

4. Pautas de evaluación

Evaluaremos fundamentalmente los siguientes aspectos:

- Defensa del trabajo. Tendrás que ser capaz de demostrar que entendés tu trabajo. Está bien buscar ideas en internet y/o consultar con tus compañeros, pero tenés que *demostrar* que entendés lo que estás entregando. Luego de la entrega se tomará un ejercicio adicional, trivial, para demostrar que entendiste de qué trata tu código. Este ejercicio adicional es tan importante como el TP. No es posible aprobar el TP sin realizar correctamente dicho ejercicio. Entonces, procurá entender tu código.
- El proyecto debe compilar bien. Si esto no sucede, el trabajo se considera desaprobado. Consejo: **compilá todo el tiempo tu código**. No avances con otras cuestiones hasta que no hayas logrado esto, y pedí ayuda en lo que respecta al lenguaje y las herramientas que usamos.
- La implementación correcta del TP. Recordá que podés pedir ayuda a los profesores y ayudantes. Pero dicho esto, la implementación debe **ejecutar correctamente**, y, de no hacerlo, te damos el tiempo de la defensa del TP para que puedas corregirlo. Si no llegás a completar en el lapso estipulado una implementación correcta, el trabajo se considera desaprobado.
- El estilo de la materia. Que tu implementación compile y funcione correctamente no es garantía de que hayas utilizado los conceptos vistos en la materia. Es muy importante que, por ejemplo, respetes barreras de abstracción, indiques los invariantes de representación, precondiciones, etc. Un trabajo con graves faltas en este sentido no estará aprobado con buena nota, e incluso puede llegar a estar desaprobado, si es que no cumple con ninguna de las consignas vistas durante la materia.

5. Mientras Tanto, en el Mundo Real ...

Este trabajo es puramente pedagógico y no refleja como se administran claves o se crackean sistemas en el mundo real. Por ejemplo:

[2] `tpi-doc-ed@listas.unq.edu.ar`

- Nadie encripta claves usando MD5, no es apto para dicha tarea. En la vida real MD5 se usa para verificar que un archivo no fue adulterado. Muchos sitios dan a conocer el MD5 *hash* de los archivos disponibles para bajar. Luego de obtener un archivo podemos calcular su MD5 hash y compararlo con el del sitio para saber si tenemos el contenido correcto.
- Nadie encriptaría claves sin "ponerles sal"[3]
- No hace falta desperdiciar memoria usando un árbol general para computar las palabras posibles a partir de un vocabulario dado. La generación de palabras es fácil de hacer sin recurrir a una estructura de datos auxiliar.
- Los ataques por fuerza bruta se realizan usando un diccionario de claves conocidas que son usualmente empleadas por usuarios incautos. Por eso la mayoría de los sitios obligan a usar caracteres numéricos y símbolos en la claves. El uso de sal criptográfica complica los ataques con diccionarios.

6. Consejos

- **No procastinar:** si bien el código C++ a escribir es poco, entender el enunciado y pensar cómo resolver cada ejercicio requerirá atención y tiempo. ¡No esperes a último momento!
- **Preguntar, preguntar, preguntar:** ni bien comiences con el trabajo es esperable que tengas dudas (por ejemplo, acerca de la generación del árbol en el Ejercicio 3). No dudes en consultar con la cátedra, es parte de nuestra tarea como docentes ayudarte a que puedas completar el trabajo.
- **No demorar las pruebas:** no esperes a tener terminado el trabajo completo para probar tu implementación. Las pruebas deben ser continuas: probar cada operación al ser finalizada, y cada módulo al final de cada ejercicio. Mientras antes descubras tus errores, mejor.

— ¡Mucha suerte! —

[3] https://en.wikipedia.org/wiki/Salt_(cryptography)