Práctica 1: Megabot 3000 (parte 1)

Programación 2

Curso 2019-2020

Esta práctica consiste en el desarrollo de un robot conversacional o *chatbot*, capaz de dialogar con el usuario para proporcionarle la mejor respuesta posible a las consultas que formule. Los conceptos necesarios para desarrollar esta práctica se trabajan en el *Tema 1* y *Tema 2* de teoría.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 13 de marzo**, hasta las **23:59**
- Debes entregar un único fichero llamado `prac1.cc` con el código de todas las funciones

Código de honor

Si se detecta copia (total o parcial) en tu práctica, tendrás un **0** en la entrega y se informará a la dirección de la Escuela Politécnica Superior para que adopte medidas disciplinarias

Está bien discutir con tus compañeros posibles soluciones a las prácticas
Está bien apuntarte a una academia si sirve para obligarte a estudiar y hacer las prácticas

Está mal copiar código de otros compañeros para resolver tus problemas
Está mal apuntarte a una academia para que te hagan las prácticas

Si necesitas ayuda acude a tu profesor/a
No copies

Normas generales

- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Se puede acceder a él de dos maneras:
  - Página principal del DLSI (`https://www.dlsi.ua.es`), opción "ENTREGA DE PRÁCTICAS"
  - Directamente en la dirección `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud
  - No se admitirán entregas fuera de plazo
- Tu práctica debe poder ser compilada sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas

- Si tu práctica no se puede compilar su calificación será 0
- El 70 % de la nota de la práctica dependerá de la corrección automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado. El otro 30 % dependerá de la revisión manual del código que haga tu profesor de prácticas, por lo que debes también ajustarte a la guía de estilo de la asignatura
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

prac1.cc

```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las transparencias de presentación de la asignatura (*Tema 0*)

1. Descripción de la práctica

Tras una absurda discusión sobre cuál es la mejor asignatura del Grado en Ingeniería Informática de la UA (absurda porque, claramente, la mejor es *Programación 2*), todos tus compañeros te han dado de lado. Para suplir este vacío, has decidido construirte un amigo virtual que dé respuesta a todas tus dudas, alguien que esté siempre disponible y nunca se enfade contigo.

Tu objetivo en esta práctica es construir una primera versión de este robot conversacional que sea capaz de, dada una consulta, proporcionar la respuesta más adecuada en función de la base de conocimientos que almacene en un momento dado. Después de barajar distintos nombres para tu creación, que no terminan de sonarte bien (*Palexa*, *Chiri*, *Tortana*, *Bubel*...), decides bautizarlo como *Megabot 3000*.

2. Detalles de implementación

En el Moodle de la asignatura se publicarán varios ficheros que necesitarás para la correcta realización de la práctica:

- `prac1.cc`. Este fichero contiene un esqueleto de programa sobre el que realizar tu práctica. Descárgalo y añade tu código en él. Este fichero contiene la siguiente información:
  - Un array `greetings` con la lista de posibles saludos iniciales del *chatbot* (ver Sección 6.2)
  - Un tipo enumerado `Error` que contiene todas las posibles clases de error que se pueden dar en esta práctica (por ejemplo, `ERR_OPTION`)
  - Una función `error` que se encarga de mostrar el correspondiente mensaje de error por pantalla en función del parámetro que se le pase. Por ejemplo, cuando la función recibe el parámetro `ERR_OPTION`, mostrará por pantalla el mensaje "`ERROR: wrong menu option`"
  - El prototipo de la función `cleanString` que se describe más abajo
  - Una función `showMainMenu` y otra `showTrainMenu` que muestran por pantalla el menú principal y el menú de entrenamiento respectivamente
  - Los prototipos de las funciones que se deben implementar obligatoriamente para gestionar todas las opciones de los menús, tal y como se explica en la Sección 6. Además de éstas, deberás incluir otras funciones para que tu código sea más sencillo y fácil de leer e interpretar por un humano (más concretamente, por el profesor que tiene que corregir tu práctica)
  - Una función `main` que implementa la gestión del menú principal y llama a las funciones correspondientes dependiendo de la opción elegida por el usuario

- `converter.o`. Fichero en código objeto (compilado para máquinas Linux de 64 bits) que contiene una función `cleanString` que recibe como parámetro una cadena de texto y devuelve esa misma cadena sin acentos, ni diéresis, ni eñes (consulta la Sección 6.1.3 para más información)
- `autocorrector-prac1.tgz`. Contiene los ficheros del autocorrector para probar la práctica con algunas pruebas de entrada. La corrección automática de la práctica se realizará con un corrector similar, con estas pruebas y otras más definidas por el profesorado de la asignatura
- `prac1`. Fichero ejecutable de la práctica (compilado para máquinas Linux de 64 bits) desarrollado por el profesorado de la asignatura, para que puedas probarlo con las entradas que quieras y ver la salida correcta esperada

Ten en cuenta, además, las siguientes observaciones:

- Para poder utilizar en tu práctica la función `cleanString` del fichero `converter.o`, tendrás que compilar tu código de la siguiente manera:

Terminal
```
$ g++ -Wall -g converter.o prac1.cc -o prac1
```

- En la corrección de la práctica se introducirán siempre datos del tipo correcto, aunque con valores que pueden ser incorrectos. Por ejemplo, si se pide el identificador de una frase de ejemplo, se probará siempre con un valor entero, que podría ser `-1237` o `0`, pero nunca se introducirá un valor de otro tipo (carácter, `string`, número real, ...)
- El resto de decisiones en la implementación de la práctica quedan a tu criterio, pero ten en cuenta que el código fuente de la práctica será revisado por tu profesor de prácticas siguiendo la guía de estilo publicada en la web de la asignatura. Parte de la nota de la práctica depende de dicha revisión

3. Funcionamiento del programa

Megabot 3000 contará con una base de conocimiento compuesta de una serie de intenciones. Cada intención tendrá un conjunto de frases de ejemplo y una respuesta asociada. Por ejemplo, podemos crear una intención que nos permita dar respuesta cuando un usuario pregunta por las fechas de exámenes de Programación 2. Algunas frases de ejemplo que podríamos asociar a esta intención serían: "¿Cuándo es el examen de Programación 2?", "¿Qué día me examino de P2?" y "Dime cuándo es el examen de P2". La respuesta para esa intención podría ser "El 3 de junio de 2020". Se podrían definir intenciones para que el *chatbot* responda a cosas tan variadas como dónde está el despacho de un profesor, cuál es el río más largo del mundo o qué distancia hay entre Alicante y París. Cada intención tendrá su conjunto de frases de ejemplo y su respuesta asociada.

Cuando un usuario realice una consulta al sistema, el programa deberá compararla con todas las frases de ejemplo que tenga almacenadas, localizando aquella que más se parezca. Una vez localizada ésta, se mirará a qué intención pertenece y, a continuación, se devolverá la respuesta asociada a dicha intención.

4. Componentes

El programa a desarrollar debe poder gestionar tres elementos fundamentales: las *intenciones* que es capaz de identificar y responder el robot, las *frases de ejemplo* que se asocian a dichas intenciones y, por último, el propio *chatbot* que almacenará toda la base de conocimiento y los diferentes parámetros de configuración del programa. Los siguientes apartados describen la estructura de cada uno de estos componentes en detalle.

4.1. Example

Una estructura de tipo `Example` almacenará la información relativa a una *frase de ejemplo* asociada a una intención que tenga el *chatbot* en su base de conocimiento. Cada frase de ejemplo contendrá un identificador único (`id`) que será asignado por el programa de manera automática (la primera frase tendrá el `id 1`, la segunda el 2, etc.). Además, contendrá la cadena de texto con la propia frase (`text`) y un vector que almacenará las distintas palabras (`tokens`) que la componen. Esta información se almacenará en un registro (`struct`) del siguiente tipo:

```
struct Example{
  int id;
  string text;
  vector<string> tokens;
};
```

- En esta práctica llamamos *token* a cualquier cadena de caracteres entre dos espacios en blanco. Por ejemplo, la cadena "la casa verde" contendría tres *tokens*: "la", "casa" y "verde"

4.2. Intent

Una *intención* (`Intent`) identifica una necesidad de información específica del usuario del sistema. Conocer los horarios de prácticas de Programación 2 o saber la capital de Kirguistán, serían dos ejemplos de intenciones. Una intención se compone de tres elementos. El primero de ellos es el nombre de la intención (`name`) que permite identificarla de manera única. El segundo es el conjunto de frases de ejemplo asociadas a esa intención (`examples`). Por último, tenemos la respuesta a esa intención (`response`). Esta información se almacenará en un registro del siguiente tipo:

```
struct Intent{
  string name;
  vector<Example> examples;
  string response;
};
```

Por ejemplo, si queremos almacenar la intención de conocer el horario de prácticas de Programación 2, podemos crear un `Intent` y darle a `name` el valor `horario p2`. Algunos ejemplos de frases (se almacenarían en el vector `examples`) que podríamos asociar con este `Intent` serían: "¿Cuándo tengo prácticas de Programación 2?", "¿Cuál es mi grupo de prácticas de P2?" o "Quiero mi horario de prácticas de Programación 2". Finalmente, la respuesta esperada, que se almacenaría en `response`, podría ser "Los lunes de 11:00 a 13:00".

4.3. Chatbot

Esta estructura almacena toda la base de conocimiento del *chatbot*, formada por la lista de intenciones almacenadas en el sistema (`intents`) y por la información de configuración del mismo: el identificador que se asignará a la siguiente frase de ejemplo que se cree (`nextId`), el umbral de similitud para determinar cuándo el sistema es capaz de responder o no a una consulta (`threshold`) y la medida se similitud empleada (`similarity`). Toda esta información se almacenará en un registro con el siguiente formato:

```
struct Chatbot{
  int nextId;
  float threshold;
  char similarity[3];
  vector<Intent> intents;
};
```

El campo `similarity` almacenará una cadena de dos caracteres que identificará al algoritmo empleado. En esta primera práctica el único valor posible será `jc`, correspondiente al coeficiente de Jaccard.

5. Menú

Al ejecutar la práctica, se mostrará de inicio por pantalla el *menú principal* del programa, quedando a la espera de que el usuario elija una opción:

Terminal

```
1- Train
2- Test
3- Report
q- Quit
Option:
```

Las opciones válidas que puede introducir el usuario son los números del `1` al `3` y la letra `q`. Si la opción elegida no es ninguna de éstas (por ejemplo, un `4`, una `a` o un `;`), se emitirá el error `ERR_OPTION` llamando a la función `error` con dicho parámetro. Cuando el usuario elige una opción correcta, se debe ejecutar el código asociado a esa opción. Al finalizar, volverá a mostrar el menú principal y a pedir otra opción, hasta que el usuario decida salir del programa utilizando la opción `q`.

6. Opciones

En los siguientes apartados se describe el funcionamiento que deberá tener cada una de las opciones que se muestran en el menú principal del programa.

6.1. Train

Esta opción permitirá incrementar la base de conocimiento del *chatbot* para que aprenda nuevas intenciones, con sus frases de ejemplo y respuestas asociadas, usando para ello la función `train`. Ésta se activa cuando el usuario elige la opción `1` del menú principal. Al elegir esta opción se mostrará por pantalla un nuevo menú, que de ahora en adelante llamaremos *menú de entrenamiento*, con todas las opciones para gestionar la base de conocimiento, quedando a la espera de que el usuario introduzca una opción:

Terminal

```
1- Add intent
2- Delete intent
3- Add example
4- Delete example
5- Add response
b- Back to main menu
Option:
```

Las opciones válidas que puede introducir el usuario son los números del `1` al `5` y la letra `b`. Al igual que con el menú principal, si la opción elegida no es ninguna de éstas se emitirá el error `ERR_OPTION`, llamando a la función `error` con dicho parámetro y mostrando de nuevo el menú de entrenamiento. Cuando el usuario elige una opción correcta, se debe ejecutar el código asociado a esa opción. Al finalizar, volverá a mostrar este menú de entrenamiento y a pedir otra opción, hasta que el usuario decida volver al menú principal utilizando la opción `b`.

En las siguientes subsecciones se describe el funcionamiento que deberá de tener cada una de las opciones que se muestran en el menú de entrenamiento.

6.1.1. Add intent

Esta opción permite la creación de un nuevo `Intent`, haciendo uso para ello de la función `addIntent`. Ésta se activa cuando el usuario elige la opción `1` del menú de entrenamiento. En primer lugar se le pedirá al usuario que introduzca el nombre del nuevo `Intent` con el siguiente mensaje:

Terminal
```
Intent name:
```

Si ya existe una intención con ese nombre, se mostrará el error `ERR_INTENT` y se volverá al menú de entrenamiento. Si no existe, se creará un nuevo `Intent` con ese nombre (de momento sin frases de ejemplo ni respuesta asociada) y se añadirá al final del vector `intents` del *chatbot*.

6.1.2. Delete intent

Esta opción permite eliminar un `Intent` existente, haciendo uso para ello de la función `deleteIntent`. Ésta se activa cuando el usuario elige la opción `2` del menú de entrenamiento. En primer lugar se pedirá que introduzca el nombre del `Intent` que se desea eliminar con el siguiente mensaje:

Terminal
```
Intent name:
```

Si el nombre introducido por el usuario no existe, se mostrará el error `ERR_INTENT` y se volverá al menú de entrenamiento. Si el nombre es correcto, se pedirá al usaurio confirmación con el siguiente mensaje:

Terminal
```
Confirm [Y/N]?:
```

Si la respuesta es `Y` o `y`, se borrará el `Intent` y se volverá al menú de entrenamiento. Si la respuesta es `N` o `n`, se volverá al menú de entrenamiento sin hacer nada. Si la respuesta no es ninguna de las anteriores, se mostrará de nuevo el mensaje anterior solicitando la confirmación.

6.1.3. Add example

Esta opción permite añadir frases de ejemplo al final del vector `examples` de un `Intent` existente, haciendo uso para ello de la función `addExample`. Ésta se activa cuando el usuario elige la opción `3` del menú de entrenamiento. En primer lugar se le pedirá al usuario que introduzca el nombre del `Intent` al que se desean añadir ejemplos:

Terminal
```
Intent name:
```

De nuevo, si el nombre introducido por el usuario no existe se mostrará el error `ERR_INTENT` y se volverá al menú de entrenamiento. Si el nombre es correcto, se le pedirá al usuario que introduzca una nueva frase de ejemplo con el siguiente mensaje:

Terminal

```
New example:
```

Se deberá de crear un nuevo `Example` y almacenar el texto introducido por el usuario en el campo `text`. A cada nuevo ejemplo que se cree se le asignará un identificador (`id`) nuevo, que se corresponderá con el valor que haya almacenado en el campo `nextId` del registro `Chatbot`. El primer `Example` tendrá el valor `1` y se deberá incrementar el valor de `nextId` cada vez que se cree uno nuevo.

Para facilitar el posterior cálculo de similitud entre consultas y frases de ejemplo, en el vector `tokens` se almacenará cada una de las palabras de la frase de ejemplo por separado. Antes de almacenar las palabras en este vector, se llamará a la función `cleanString` (proporcionada por el profesorado en el fichero `prac1.cc`) pasando como parámetro la cadena almacenada en el campo `text`. Esta función devolverá esa misma cadena tras eliminar los acentos, las diéresis y las eñes. Esto se hace para evitar problemas con la codificación interna de estos caracteres, que haría que algunas de las funciones sugeridas más abajo no funcionaran correctamente. De la cadena devuelta habrá que eliminar todos aquellos caracteres que no sean alfanuméricos (letras o números), como es el caso de los signos de puntuación. Además, se pasará todo el texto a minúsculas y se eliminará el carácter 's' de final de palabra. Este proceso de normalización del texto va a permitir que palabras como `Casa`, `casa` y `casas` se conviertan a la misma cadena `casa`, facilitando de esta manera la comparación entre cadenas.

Al terminar de almacenar el ejemplo, se volverá a pedir al usuario que introduzca un nuevo ejemplo para ese mismo `Intent`, mostrando de nuevo el mensaje anterior. Esto se repetirá hasta que el usuario escriba `q`. En ese momento, dejarán de leerse frases de ejemplo y se volverá al menú de entrenamiento.

- Ten en cuenta que el usuario podría meter más de un espacio en blanco para separar entre sí dos palabras en la frase de ejemplo
- Para identificar si un carácter es alfanumérico puedes utilizar la función `isalnum` de la librería `cctype`, que recibe un único carácter y devuelve `true` si el carácter es letra o número, o `false` en caso contrario
- Para pasar a minúsculas puedes usar la función `tolower`, también de la librería `cctype`, que recibe un carácter y devuelve su equivalente en minúsculas. Si no es una letra mayúscula simplemente devolverá el mismo carácter
- Los identificadores de ejemplo (`id`) son únicos y no se pueden repetir aunque el ejemplo pertenezca a un `Intent` diferente
- Nunca se deben almacenar palabras vacías en el vector `tokens`
- Si el usuario escribiera una cadena totalmente vacía, o al aplicar el proceso de normalización de texto quedara vacía, no se almacenará el ejemplo y por tanto no deberá incrementarse el valor de `nextId`
- Cuando se elimina un signo de puntuación, no se debe introducir un espacio en blanco en su lugar. Por ejemplo, si tenemos la cadena "`hola,mundo`", al eliminar el signo de puntuación quedará como una única palabra "`holamundo`"

6.1.4. Delete example

Esta opción permite eliminar un ejemplo de la lista `examples` de un `Intent`, haciendo uso para ello de la función `deleteExample`. Ésta se activa cuando el usuario elige la opción `4` del menú de entrenamiento. En primer lugar se le pedirá al usuario que introduzca el `id` del ejemplo a eliminar, mostrando el siguiente mensaje por pantalla:

```
Terminal
Example id:
```

Si existe una frase de ejemplo con ese `id` en algún `Intent` del vector `intents` del *chatbot*, se eliminará de la lista `examples` del `Intent` y se volverá al menú de entrenamiento. Si no existe ningun ejemplo con ese identificador, se lanzará el error `ERR_EXAMPLE` y se volverá al menú de entrenamiento sin hacer nada.

- Cuando se elimina un ejemplo, no es necesario reordenar los valores de `id` de los ejemplos restantes, ni modificar el valor `nextId` del registro `Chatbot`

6.1.5. Add response

Esta opción permite añadir la respuesta (`response`) asociada a un `Intent`, haciendo uso para ello de la función `addResponse`. Ésta se activa cuando el usuario elige la opción `5` del menú de entrenamiento. En primer lugar se le pedirá al usuario que introduzca el nombre del `Intent` al que se quiere asociar la respuesta:

```
Terminal
Intent name:
```

De nuevo, si el nombre introducido por el usuario no existe se mostrará el error `ERR_INTENT` y se volverá al menú de entrenamiento. Si el nombre es correcto, se le pedirá al usuario que introduzca el texto de la respuesta con el siguiente mensaje:

```
Terminal
New response:
```

Si el `Intent` seleccionado ya tenía respuesta, ésta se sobreescribirá sin mostrar ningún tipo de mensaje adicional. Tras completar la operación se volverá al menú de entrenamiento.

6.2. Test

Esta opción permitirá probar el *chatbot* y verlo en funcionamiento, usando para ello la función `test`. Ésta se activa cuando el usuario elige la opción `2` del menú principal. En primer lugar se mostrará por pantalla un mensaje de bienvenida precedido de los caracteres >>, como se muestra en este ejemplo:

```
Terminal
>> ¿Qué puedo hacer hoy por ti?
<<
```

Los caracteres << indican que el *chatbot* está a la espera de que el usuario escriba su consulta, a continuación en esa misma línea. Estos caracteres se mostrarán cada vez que sea el turno de que el usuario escriba. Por otra parte, todos los mensajes del *chatbot* (saludos o respuestas) irán precedidos de los caracteres >>.

En cuanto a los posibles mensajes de bienvenida, éstos están almacenados en el array `greetings` que se proporciona en el fichero `prac1.cc`. Estos mensajes se mostrarán aleatoriamente, de manera que cada vez que ejecutemos `test` el mensaje de bienvenida puede ser diferente. Para conseguir este objetivo, deberás incluir las siguientes instrucciones en tu código:

```
position = rand() % KSIZE;
cout << ">> " << greetings[position] << endl;
```

- Es necesario incluir la librería `cstdlib` para poder usar la función `rand`
- La constante `KSIZE` representa el número de frases de bienvenida diferentes que hay y viene ya definida en el fichero `prac1.cc` proporcionado
- La función `rand` genera números aleatorios entre `0` y `RAND_MAX` (este valor depende de la versión de la librería). Al hacer la operación `rand() % KSIZE` estamos limitando el resultado a valores entre `0` y `KSIZE-1`. Este valor nos sirve en el código anterior para seleccionar la posición concreta del array `greetings` que queremos mostrar
- La función `srand` permite establecer una semilla que determina los valores aleatorios que irá devolviendo la función `rand`. Al asignarle un valor concreto garantizamos que la secuencia de números aleatorios que se va generando será la misma para todo el mundo (esto es imprescindible para realizar la corrección automática de la práctica). En la función `main` del fichero `prac1.cc` proporcionado se ha incluido una llamada a esta función. ¡¡No cambies su valor!!

A continuación vemos un ejemplo de diálogo con el usuario, en el que el *chatbot* da la bienvenida, el usuario pregunta por su horario y el programa le responde, quedando a la espera de una nueva consulta:

Terminal

```
>> ¡Hola! Soy Megabot 3000. ¿En qué puedo ayudarte?
<< Me gustaría saber mi horario de prácticas de Programación 2
>> Las prácticas son los martes de 9:00 a 11:00
<<
```

El diálogo terminará cuando el usuario escriba `q`, volviendo a continuación al menú principal.

6.2.1. Cálculo de la similitud entre consultas y frases de ejemplo

La parte más importante del *chatbot* es su capacidad para identificar la intención del usuario, es decir, saber qué nos está pidiendo para ofrecerle la mejor respuesta posible. En nuestro caso, cada `Intent` almacenado tiene un conjunto de frases de ejemplo (formas diferentes de expresar una intención) y una respuesta. El algoritmo para determinar la mejor respuesta para una consulta del usuario será el siguiente:

1. Procesar la consulta del usuario igual que se hace con las frases de ejemplo almacenadas en cada `Intent`: eliminar los acentos, diéresis y eñes usando `cleanString`, mantener sólo los caracteres alfanuméricos, pasar a minúsculas, eliminar el carácter 's' de final de palabra y extraer los *tokens*
2. Comparar los tokens de la consulta del usuario con los de las frases de ejemplo de cada `Intent` que tengamos almacenado, utilizando para ello una medida de similitud entre cadenas (en nuestro caso el coeficiente de Jaccard, explicado en la siguiente sección)
3. Mirar a qué `Intent` pertenece la frase de ejemplo almacenada que mayor valor de similitud tenga con la consulta del usuario, devolviendo como respuesta aquella que haya almacenada en el campo `response` de ese `Intent`

- En caso de que dos frases de ejemplo, pertenecientes a distintas intenciones, tengan el mismo valor de similitud, nos quedaremos con el `Intent` que aparezca primero en el vector `intents` del *chatbot*

6.2.2. Coeficiente de Jaccard

El *coeficiente de Jaccard* mide el grado de similitud que existe entre dos conjuntos, independientemente del tipo de elementos que lo componen. En nuestro caso, los dos conjuntos a comparar son la lista de *tokens* de la consulta del usuario y de las frases de ejemplo asociadas a cada `Intent`.

Dadas dos cadenas de caracteres `A` y `B`, el coeficiente de Jaccard $J(A,B)$ vendrá dado por la siguiente fórmula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Es decir, se divide el número de palabras que hay en común entre ambas cadenas ($|A \cap B|$) por el número de palabras diferentes que hay en total ($|A \cup B|$). El valor resultante estará siempre comprendido entre `0` y `1`. Cuanto más se aproxime el valor a `1`, mayor será la similitud entre ambas cadenas.

Por ejemplo, dados estos tres conjuntos de palabras:

```
A: cuál es mi horario
B: qué horario tengo
C: quiero saber mi horario actual
```

Si queremos obtener la similitud del conjunto `A` con cada uno de los otros dos, tendremos que calcular los coeficientes $J(A,B)$ y $J(A,C)$:

$$J(A,B) = 1/(4+3-1) = 0,1667$$
$$J(A,C) = 2/(4+5-2) = 0,2857$$

En este ejemplo, el conjunto `C` es el que más se parece a `A` de los dos, ya que $J(A,C) > J(A,B)$.

- Para hacer correctamente el cálculo del coeficiente de Jaccard, en el conjunto de palabras que se extrae de cada cadena no pueden haber elementos repetidos. Es decir, si el conjunto de palabras almacenadas en `tokens` es “es”, “de”, “noche”, “o”, “es”, “de”, “dia”, tendríamos que quedarnos únicamente con las palabras “es”, “de”, “noche”, “o”, “dia” a la hora de hacer el cálculo del coeficiente de Jaccard, sin tener en cuenta las repeticiones de las palabras “es” y “de”

6.2.3. Consultas sin respuesta

Para que nuestro sistema sea capaz de determinar cuándo tiene conocimientos para contestar a una consulta y cuándo no, vamos a establecer un umbral de similitud que se almacenará en el campo `threshold` del registro `Chatbot`.

Inicialmente le vamos a asignar el valor `0.25`. De esta manera, si el valor más alto de similitud (según el coeficiente de Jaccard) obtenido para todas las frases de ejemplo almacenadas en el sistema está por debajo de ese valor, asumiremos que nuestro *chatbot* no ha conseguido encontrar un ejemplo entre todos los `Intents` almacenados que se parezca lo suficiente a la consulta del usuario. En ese caso, habrá que llamar a la función `error` con el parámetro `ERR_RESPONSE` para mostrar el mensaje correspondiente.

6.3. Report

Esta opción mostrará por pantalla un resumen de la información almacenada en el *chatbot*:

- Algoritmo usado para calcular la similitud (campo `similarity` del registro `Chatbot`). En esta práctica el único valor posible es `Jaccard`
- Umbral de corte, almacenado en el campo `threshold` del registro `Chatbot`
- Información de las intenciones almacenadas: nombre, respuesta y frases de ejemplo asociadas
- Para cada frase de ejemplo hay que mostrar la lista de palabras almacenadas en el vector `tokens`, incluyendo cada una de estas palabras entre los caracteres < y >

- Número total de intenciones almacenadas en el *chatbot*
- Número total de frases de ejemplo almacenadas entre todas las intenciones
- Número medio de frases de ejemplo por intención

Para mostrar esta información se usará la función `report`. Ésta se activa cuando el usuario elige la opción `3` del menú principal.

A continuación se muestra un ejemplo de salida por pantalla para un *chatbot* que tiene dos intenciones almacenadas (llamadas `vacaciones navidad` y `cordial`), una con tres frases de ejemplo asociadas (con `id` igual a `1`, `3` y `4`) y la otra con dos (con `id` igual a `2` y `6`):

Terminal

```
Similarity: Jaccard
Threshold: 0.25
Intent: vacaciones navidad
Response: El 24 de diciembre
Example 1: ¿Cuándo empiezan las vacaciones de Navidad?
Tokens 1: <cuando> <empiezan> <la> <vacacione> <de> <navidad>
Example 3: Quiero saber cuándo paramos en Navidad
Tokens 3: <quiero> <saber> <cuando> <paramo> <en> <navidad>
Example 4: Dime la fecha de inicio de las vacaciones navideñas
Tokens 4: <dime> <la> <fecha> <de> <inicio> <de> <la> <vacacione> <navidena>
Intent: cordial
Response: No tan bien como tú
Example 2: ¿Qué tal estás, Megabot?
Tokens 2: <que> <tal> <esta> <megabot>
Example 6: ¿Estás bien?
Tokens 6: <esta> <bien>
Total intents: 2
Total examples: 5
Examples per intent: 2.5
```