Laboratorio 2023 de Programación 1

Tarea 2

Información general

Se sugiere leer con mucha atención todo el texto antes de comenzar la tarea. Es muy importante que se respeten todos los requisitos solicitados en esta sección y las siguientes. Si surgen dudas, pedimos que las formulen en el foro correspondiente a la tarea.

Individualidad

Esta tarea se deberá realizar en **grupos de hasta dos estudiantes** (para poder elegir grupo tiene que haber realizado anteriormente el ***Cuestionario 4***). Para todas las tareas rige el Reglamento del Instituto de Computación ante Instancias de No Individualidad en los Laboratorios (lectura obligatoria).

Calendario

Entrega: La entrega de la tarea puede realizarse **hasta las 20:00 del 15 de noviembre**. Los trabajos deberán ser entregados dentro de los plazos establecidos. **No** se aceptarán trabajos fuera de plazo. Para poder entregar tiene que haber realizado anteriormente el ***Cuestionario 5***.

Re-Entrega: Todos los estudiantes que realizaron la entrega pueden hacer modificaciones y realizar una segunda entrega (*re-entrega*). La re-entrega puede realizarse **hasta las 20:00 del 17 de noviembre**.

Forma de entrega

Se debe entregar un **único** archivo de nombre **tarea2.pas** que debe contener **únicamente** el código de los subprogramas pedidos y eventualmente el de subprogramas auxiliares que se necesiten para implementarlos.

Archivos provistos y ejecución de pruebas

El archivo *definiciones.pas* contiene constantes, tipos y procedimientos auxiliares definidos por los docentes. El programa principal para realizar pruebas es provisto por los docentes en el archivo *principal.pas*. No se debe modificar ninguno de estos archivos, dado que no formarán parte de la entrega. Para usar este programa se debe leer y seguir las instrucciones provistas en la sección Cómo ejecutar los casos de prueba de la Segunda Tarea.

Introducción

El **texto predictivo** es una tecnología que se utiliza en aplicaciones de procesamiento de texto y comunicación, como teclados virtuales en dispositivos móviles o aplicaciones de mensajería. Su objetivo es anticipar las palabras o frases que un usuario probablemente querrá escribir a continuación y ofrecer sugerencias para facilitar la escritura.

En este laboratorio implementaremos una versión simplificada de texto predictivo, que luego de escrita una palabra sugiere una lista de posibles palabras siguientes. Por ejemplo, en la Figura 1 para la palabra "Programación" se sugieren: "y", "de" y "que".

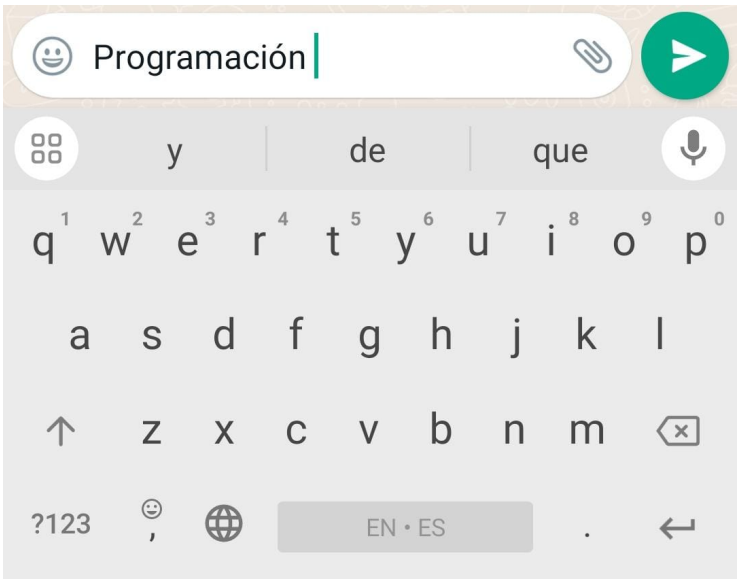

Figure 1: Ejemplo de uso del predictivo de WhatsApp

Las palabras sugeridas son las que aparecen más frecuentemente a continuación de la palabra ingresada, de acuerdo a ciertos textos que el algoritmo de texto predictivo usó para *aprender*.

La tarea consiste en implementar los subprogramas necesarios para implementar las funcionalidades de:

1. entrenamiento del algoritmo para que aprenda las frecuencias a partir de un texto
2. obtención de las palabras sugeridas para continuar una palabra dada.

Constantes y Tipos de Datos

Las constantes y tipos de datos son provistos por los docentes en el archivo *definiciones.pas* y **no deben ser modificados**.

Se definen las siguientes constantes, todas de valor entero mayor que cero:

```
SEMILLA = ...;  { semilla para función de hash }
PASO    = ...;  { paso para función de hash }
MAXHASH = ...;  { cota de la función de hash }
MAXPAL  = ...;  { cota de la palabra }
MAXALTS = ...;  { cota de alternativas }
```

Y los siguientes tipos de datos:

```
{ tipo de los naturales }
Natural = QWord;
```

```
{ arreglo con tope de letras, que representa a una palabra }
Letra   = 'a' .. 'z';
Palabra = record
             cadena : array [1 .. MAXPAL] of Letra;
             tope   : 0 .. MAXPAL
          end;
```

```
{ enumerado para indicar el resultado de una comparación
  entre palabras }
Comparacion = (menor, igual, mayor);
```

```
{ lista de palabras, que representa a un texto }
Texto   = ^NodoPal;
NodoPal = record
             info : Palabra;
             sig  : Texto
          end;
```

```
{ registro para indicar la cantidad de veces que ocurre
  una palabra }
PalabraCant = record
                 pal  : Palabra;
                 cant : integer
              end;
```

```
{ lista de ocurrencias de palabras }
Ocurrencias = ^Nodo;
Nodo        = record
                 palc : PalabraCant;
                 sig  : Ocurrencias
              end;
```

```
{ arreglo indexado por los códigos de hash de las distintas
  palabras y que para cada palabra contiene la lista de
  palabras que suelen aparecer a continuación y la cantidad
  de veces que ocurrieron en los textos de entrenamiento }
Predictor  = array [1 .. MAXHASH] of Ocurrencias;
```

```
{ arreglo con tope para retornar hasta MAXALT alternativas
  de palabras que pueden continuar }
Alternativas = record
                  pals : array [1..MAXALTS] of PalabraCant;
                  tope : 0 .. MAXALTS
               end;
```

La estructura `Predictor` se utiliza de la siguiente manera. Al momento de entrenar el algoritmo con un texto, se debe agregar para cada par de palabras *palabra1 palabra2*, contiguas en el texto, a la *palabra2* como una ocurrencia en la lista de ocurrencias indexada por el código de hash de *palabra1*. Esto es, si por ejemplo el código de hash de "hola" es `8521` y tenemos el predictor de la Figura 2, el algoritmo fue entrenado con textos en los que inmediatamente a continuación de la palabra "hola" aparecieron "que" cinco veces, "y" una vez, "como" diez veces, "quien" dos veces y "te" ocho veces[1].

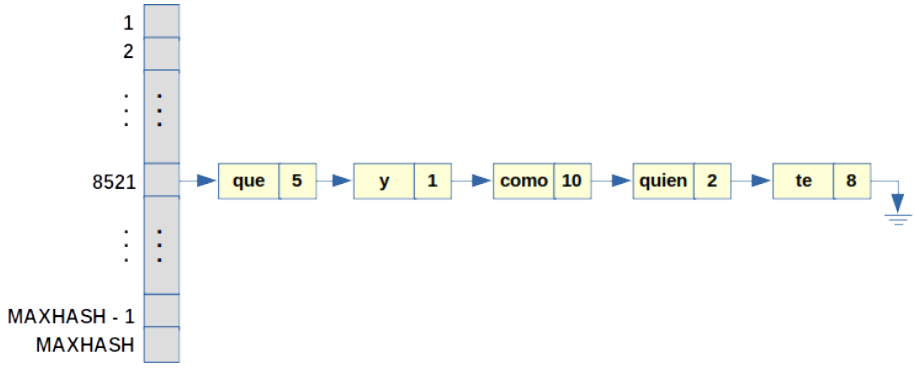

Figure 2: Estructura Predictor

Si, por ejemplo, `MAXALTS` es `3` y solicitamos a ese predictor que nos retorne alternativas, el resultado se cargará en una variable de tipo `Alternativas` con la información que se muestra en la Figura 3.

Ahora, supongamos que para otra palabra solo hay dos posibles alternativas, que son "casa" y "pica", que aparecen 2 y 1 veces respectivamente. Entonces el resultado del predictor para esa palabra sería el de la Figura 4.

[1] Notar que las funciones de hash pueden tener colisiones, pero en esta tarea no las tendremos en cuenta.

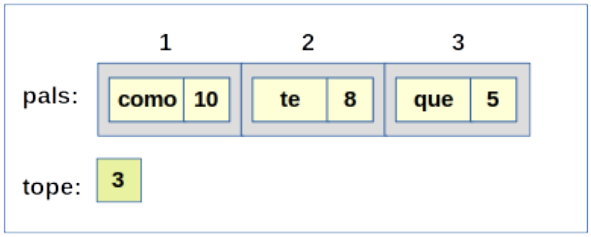

Figure 3: Estructura Alternativas

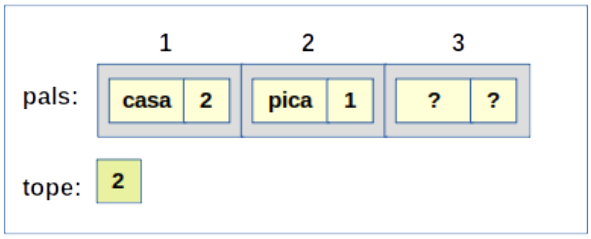

Figure 4: Estructura Alternativas Incompleta

Subprogramas Solicitados

Se deben implementar los siguientes subprogramas:

- Función `hash`. Retorna el código de la función de hash que describimos en la tarea 1 para la palabra `p` usando los parámetros `semilla`, `paso` y `N`.

```
function hash ( semilla, paso, N : Natural
              ; p : Palabra )  :  Natural;
```

- Función `comparaPalabra`. Retorna el resultado de comparar la palabras `p1` y `p2` en base al orden lexicográfico. Si `p1` es menor que `p2` retorna `menor`, si es mayor retorna `mayor` y si son iguales retorna `igual`.

```
function comparaPalabra ( p1, p2 : Palabra ) : Comparacion;
```

- Función `mayorPalabraCant`. Retorna `true` si la cantidad de ocurrencias de `pc1` es mayor que la de `pc2` o si son iguales y la palabra de `pc1` es mayor (en orden lexicográfico) que la de `pc2`. En otro caso retorna `false`.

```
function mayorPalabraCant( pc1, pc2 : PalabraCant ) : boolean;
```

- Procedimiento `agregarOcurrencia`. Agrega una ocurrencia de la palabra `p` en la lista de ocurrencias `pals`. Si la palabra ya pertenece a `pals` entonces incrementa la cantidad de ocurrencias. Si la palabra no pertenece, se inserta al final de la lista (con cantidad 1).

```
procedure agregarOcurrencia ( p : Palabra
                            ; var pals : Ocurrencias );
```

- Procedimiento `inicializarPredictor`. Inicializa el predictor `pred` dejando vacías todas las listas de ocurrencias.

```
procedure inicializarPredictor ( var pred: Predictor );
```

- Procedimiento `entrenarPredictor`. Dada una lista de palabras `txt` que representa a un texto, entrena al predictor `pred` con dicho texto. Esto es: para cada par de palabras *palabra1 palabra2* que aparece en el texto se agrega al lugar del arreglo `pred` indexado por el código de hash de *palabra1* una ocurrencia de *palabra2*. Para agregar la ocurrencia se utiliza el procedimiento `agregarOcurrencia`. El entrenamiento es acumulativo, esto es que si `pred` ya tenía datos de entrenamientos anteriores, se deben mantener.

```
procedure entrenarPredictor ( txt : Texto
                            ; var pred: Predictor );
```

- Procedimiento `insOrdAlternativas`. Inserta `pc` en `alts` conservando su orden, que es de mayor a menor de acuerdo a la relación de orden definida en `mayorPalabraCant`. Se puede considerar que no hay palabras repetidas. Si `alts` tiene `MAXALTS` elementos y todos son mayores que `pc`, entonces éste

no se inserta. Para implementar este procedimiento se sugiere el siguiente algoritmo: insertar `pc` al final de `alts` (si es posible) y luego intercambiarlo con los elementos a su izquierda hasta que llegue a la posición que le corresponde.

```
procedure insOrdAlternativas ( pc : PalabraCant
                             ; var alts: Alternativas );
```

- Procedimiento `obtenerAlternativas`. Retorna en `alts` hasta `MAXALTS` alternativas de palabras que pueden ir a continuación de `p` de acuerdo a la información de ocurrencias que contiene `pred`. Este procedimiento utiliza `insOrdAlternativas` para insertar las palabras en `alts` ordenadas según el número de ocurrencias.

```
procedure obtenerAlternativas ( p : Palabra; pred : Predictor
                              ; var alts: Alternativas );
```

Se pide

Escribir un archivo **tarea2.pas** con todos los subprogramas solicitados. Los encabezados de los subprogramas **deben coincidir exactamente** con los que aparecen en esta letra. Si el estudiante realiza algún cambio se considerará que el subprograma no fue implementado. Si el estudiante lo desea, puede implementar subprogramas auxiliares adicionales (además de los subprogramas pedidos).

Para la corrección, las tareas se compilarán con una versión igual o posterior a **3.0.4 para Linux**. La compilación y la ejecución se realizarán en línea de comandos. El comando de compilación se invocará de la siguiente manera:

```
fpc -Co -Cr -Miso -gl principal.pas
```

`principal.pas` será el programa principal entregado por el equipo docente.

NO se debe compilar con el IDE de Free Pascal.

No está permitido utilizar facilidades de Free Pascal que no forman parte del estándar y no se dan en el curso. Así por ejemplo, no se puede utilizar ninguna de las palabras siguientes: `uses`, `crlscr`, `gotoxy`, `crt`, `readkey`, `longint`, `string`, `break`, `exit`, etcétera.

En esta tarea, como en todos los problemas de este curso, se valorará, además de la lógica correcta, la utilización de un buen estilo de programación de acuerdo a los criterios impartidos en el curso. De esta manera, se hará énfasis en buenas prácticas de programación que lleven a un código legible, bien documentado y mantenible, tales como:

- indentación adecuada,
- utilización correcta y apropiada de las estructuras de control,
- código claro y legible,
- algoritmos razonablemente eficientes,

- utilización de comentarios que documenten y complementen el código,
- utilización de constantes simbólicas,
- nombres mnemotécnicos para variables, constantes, etcétera.

Para resolver la tarea se pueden utilizar todos los conocimientos vistos en el curso.