ALGORITMOS Y ESTRUCTURAS DE DATOS II GRADO INGENIERÍA INFORMÁTICA,CURSO 16/17 PRÁCTICA DE DIVIDE Y VENCERÁS Y ANÁLISIS DE ALGORITMOS.

Memoria de la actividad:

Nombre: Ricardo Ramírez García
Grupo y Subgrupo: 2.2
Email: ricardo.ramirezg@um.es

Nombre: Pablo Salinas Rodríguez
Grupo y Subgrupo: 2.2
Email: pablo.salinasr@um.es

1. (hasta 1,5 puntos) Pseudocódigo y explicación del algoritmo, justificando las decisiones de diseño, las estructuras de datos y las funciones básicas del esquema algorítmico.

esConsonante(char c) : bool

```
if (c ==’a’ OR c == ‘e’ OR c == ‘i’ OR c == ‘o’ OR c == ‘u’) then return false
else return true
```

esConsonante :
Esta función devuelve verdadero o falso en función de si el carácter que se le pasa como parámetro es consonante o vocal respectivamente.

esVC(String p) : bool

```
if(!esConsonante(a[0]) AND esConsonante(a[1])) then return true
else return false
```

esVC :
Esta función comprueba si silaba que se le pasa como parámetro está formada por una vocal – consonante, si es así devuelve true, en caso contrario false.

esCV(String p) : bool

```
if ((esConsonante(a[0])) AND (!esConsonante(a[1]))) then return true
else return false
```

esCV :
Esta función comprueba si la silaba que se le pasa como parametro está formada por una consonante – vocal, si es así devuelve true, en caso contrario false.

max(String cad1, String cad2) : String

```
if(cad1.size() >= cad2.size()) return cad1;
sino return cad2;
```

max(String cad1, String cad2, String cad3) : String

```
if((cad1.size() >= cad2.size()) && (cad1.size() >= cad3.size())) return cad1;
sino if((cad2.size() > cad1.size())&&(cad2.size() > cad3.size())) return cad2;
sino return cad3;
```

max :
Esta función está sobrecargada ya que se puede llamar con dos o tres palabras, una vez que se le pasan las cadenas como parámetros devuelve la cadena de mayor longitud.

maxConsVocal(string cadena,int tamano) : String

```
if tamano <= CasoBase then
      return iterativo(cadena)
end
      cad1 = generarSubcadenaDer(cadena)
      cad2 = generarSubcadenaIzq(cadena)
      return  combinar(cadena,maxConsVocal(cad1,(tamano/2)),maxConsVocal(cad2,(tamano-(tamano/2))))
```

maxConsVocal :
Consta de dos partes, el caso base que se resuelve aplicando la función iterativa y el caso recursivo que consta de dos llamadas recursivas, una con la mitad derecha de la cadena y otra con la mitad izquierda, el resultado de las dos llamadas iterativas se le pasa a la función combinar junto a la cadena entera que se le ha pasado como parámetro a la función actual, esta función se encarga de combinar los resultados parciales para conseguir el resultado, finalmente se devuelve el resultado del combinar.

iterativo(string cadena) : String

```
cadenaActual = ""
cadenaMaxima = ""
buscando = 1

i = 0
while NOT esConsonante(cadena[i]) AND NOT finCadena(cadena)  do  i = i + 1
if NOT finCadena(cadena) then
       buscando = 1
       cadenaActual = cadena[i]

       for j = i + 1 to cadena.length() do
              if buscandoConsonantes(buscando) then
                     if esConsonante(cadena[j]) then
                            cadenaActual = cadenaActual + cadena[j]
                            buscando = 1
                     else cadenaActual = ""
              else if buscandoVocales(buscando) then
                     if !esConsonante(cadena[j])  then
                            cadenaActual = cadenaActual + cadena[j];
                            actualizarCadenaMaxima(cadenaActual);
                            buscando = 0
                     sino
                            cadenaActual = cadena[j]        buscando = 1
                     end
              end
       end

              return cadenaMaxima

else    return ""
```

iterativo:
Es una función que devuelve la subcadena más larga que contenga un número par de consonantes seguidas de vocal consecutivas. Primero buscamos la primera consonante, si no hay consonantes devolver directamente la cadena vacía, una vez encontrada esa consonante tenemos una condición boolean que nos ayuda a analizar en qué estado nos encontramos , es decir ; si estamos buscando una vocal o una consonante y dentro de estos casos nos encontramos ante diferentes situaciones que analizamos, al mismo tiempo que vamos alternando entre consonantes y vocales , actualizamos la cadenaActual(Válida) y sólo en el caso en el que el carácter actual sea vocal y la cadenaActual sea de mayor longitud que la cadenaMáxima(Válida) se actualiza la cadenaMáxima.

```
combinar(String cadena, String izq, String der) : String
//n = cadena.size()
calcularSilabaCentral(cadena)
if cadena.size() <= 7 then return cadenaMaximaValida(cadena)
else if esVC(silabaCentral) then
            i:= 1
            while i <= n-(n/2) AND esCV(siguienteSilabaDer) do
                  SolucionDer = solucionDer + siguienteSilabaDer
                  i = i + 2
            end
            j:= 1
            while (n/2)-j-2 >=0 AND esCV(siguienteSilabaIzq) do
                  solucionIzq = solucionIzq + siguienteSilabaIzq
                  j = j + 2
            end
            return max(solucionDer + silabaCentral + solucionIzq, izq, der)
      else
            calcularSilabaCentral(cadena)
            if not esValida(Silaba Central) then return max(der, izq)
            y := 2
            mientras y <= n-(n/2) AND esCV(siguienteSilabaDer) do
                  SolucionDer = solucionDer + siguienteSilabaDer
                  y = y + 2
            end

            z := 2
            mientras (n/2)-2-z >=0 AND esCV(siguienteSilabaIzq) do
                  solucionIzq = siguienteSilabaIzq + solucionIzq
                  z = z + 2
            end
            return max(solucionIzq + cadenaCentral + solucionDer, der, izq)
      end
      return max(der, izq)
end
```

Combinar:
Esta función se encarga de combinar las tres cadenas pasadas como parámetro (cadena, MaxDer, MaxIzq), buscando la cadena más larga valida que se pueda formar en el centro de la cadena completa, por donde se ha partido la cadena, y comprobar cual es el máximo de los tres, la cadena central, la máxima que se ha obtenido de la mitad derecha o la cadena máxima que se ha obtenido de la mitad izquierda. Para calcular la cadena central se tratan todos los casos posibles comprobando si la cadena central esVC o esCV y recorriendo silaba a silaba por ambos lados de la cadena añadiendo silabas validas a la solución hasta que se añada toda la cadena o hasta que no haya más silabas validas que se puedan añadir.

2. (hasta 1,5 puntos) Estudio teórico del tiempo de ejecución del algoritmo (tm, tM y tp) y obtención de conclusiones acerca de los órdenes.

esConsonante:
Para comprobar si es consonante se ejecuta una comprobación por lo tanto se ejecuta en **tiempo constante.**

esVC:
Para comprobar si es VC se hace una comprobación por lo tanto se ejecuta en **tiempo constante.**

esCV:
Para comprobar si es CV se hace una comprobación por lo tanto se ejecuta en **tiempo constante.**

max:
Para comprobar si una cadena es máxima entre dos o tres cadenas se deben hacer una o dos comprobaciones respectivamente que son de **tiempo constante,** suponiendo que calcular el tamaño de la cadena es tiempo constante se hace una comprobación por lo tanto se ejecuta en **tiempo constante.**

maxConsVocal:
Esta es la función recursiva en la cual el caso base hace una llamada a la función iterativo que es de orden lineal.
Función recursiva:

$$\boldsymbol{t(n)} = \begin{cases} \boldsymbol{n}\,, \boldsymbol{si\ n \leq caso\ base} \\ \boldsymbol{2 * t\left(\frac{n}{2}\right) + 1}\,, \boldsymbol{en\ otro\ caso} \end{cases}$$

Se suma uno para simplificar el cálculo ya que las instrucciones son de orden lineal.

$$t(n) - 2 * t\left(\frac{n}{2}\right) = 1 \quad \text{*Cambio de variable n } = 2^k;\ \text{k } = \log_2 n$$

$$(x-2)(x-1) = 0$$

$$t(n) = c1 * 2^{\log_2 x} + c2 * 1^{\log_2 x}$$

$$t(n) = c1 * x^{\log_2 2} + c2 = c1 * x + c2$$

$$t(n) \in \mathrm{O(n)}$$

Como el caso recursivo y el caso base tarda lo mismo se puede decir que la función es de orden lineal.

iterativo:
En el mejor caso -> el bucle no va a encontrar ninguna consonante por lo que va a llegar al final de la cadena directamente y va a devolver que no ha encontrado nada sin hacer ningún análisis:
Se ejecutan 6 instrucciones para inicializar las variables y entrar, y una instrucción en cada iteración del bucle.

$$t_m(n) = 6 + \sum_{i=1}^{n} 1 = 6 + n$$

$$t_m(n) \in \omega(n)$$

En el peor caso -> la primera letra sería una consonante y entraría en el análisis de casos tanto si es consonante como si es vocal el número de instrucciones que se ejecutan es el mismo.

$$t_M(n) = 6 + \sum_{i=2}^{n} 5 \sim 6 + 5 * n$$

$$t_M(n) \in O(n)$$

El tiempo prometido ya que $t_m(n) \in \omega(n)$ y $t_M(n) \in O(n)$ se puede decir que $\Theta(\mathrm{f}) = \mathrm{O(f)} \cap \Omega(\mathrm{f}) =$ el $t_p \in \Theta(\mathrm{n})$

Combinar:
Mejor caso -> cadena en la que la silaba central sea vocal – vocal o consonante – consonante que devuelve el resultado de una llamada a la función max que por lo tanto es constante.

$$t_m(n) \in \omega(1)$$

El peor caso -> sería que tuviera recorrer toda la cadena en busca de silabas validas, para que se dé este caso la cadena entera debería ser válida:

$$t_M(n) = \sum_{i=1}^{n/4} 1 + \sum_{j=0}^{n/4} 1 = \frac{n}{4} + \frac{n}{4} = {}^{n}/_{2} \qquad t_M \in O(n)$$

1El tiempo promedio -> se calcula multiplicando cada instrucción por la probabilidad de que se ejecuten, esta probabilidad es la de que el bucle continúe y en cada iteración es menor.

Probabilidad de que una silaba sea consonante – vocal:

$$\frac{5}{26} * \frac{21}{26} = 0{,}1553$$

Como tienen que ir seguidas en cada iteración la probabilidad irá elevada al número de silabas, es decir, el número de la iteración:

$$t_p(n) = \sum_{i=1}^{n/2} 1 * (0{,}1553^i) + \sum_{j=1}^{n/2} 1 * (0{,}1553^j)$$

Se comporta como una geométrica con la diferencia de que la base es menor que 1 por lo tanto decrece.

$$t_p(n) = 2 * \sum_{i=1}^{n/2} 1 * (0{,}1553^i)$$

$$t_p(n) = 2 * \left(\frac{0{,}1553^{\left(\frac{n}{2}\right)+1} - 1}{0{,}1553 - 1} \right)$$

Nos dimos cuenta de que la probabilidad cuando los valores de n eran muy pequeños y que el número de instrucciones no variaba mucho, así que decidimos hacer el límite para calcular hasta cuantas instrucciones se podían calcular como mucho.

$$\lim_{n \to \infty} 2 * \left(\frac{0{,}1553^{\left(\frac{n}{2}\right)+1} - 1}{0{,}1553 - 1} \right) \sim 2{,}36$$

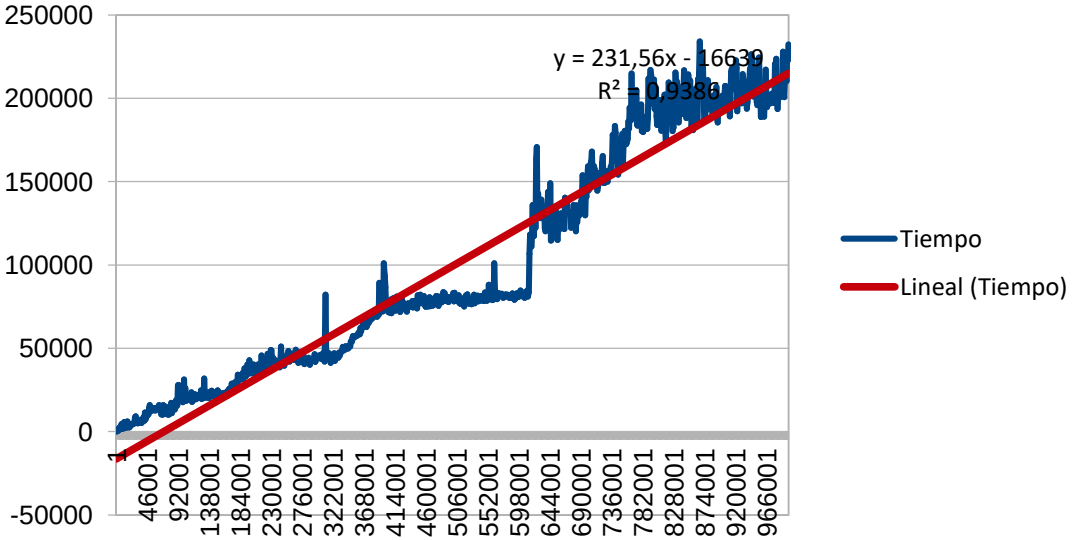
Por lo tanto hemos llegado a la conclusión de que en el tiempo promedio el combinar es **constante**.

3. (hasta 1,5 puntos) Programación del algoritmo. El programa debe ir documentado, con explicación de qué es cada variable, qué realiza cada función y su correspondencia con las funciones básicas del esquema algorítmico correspondiente.

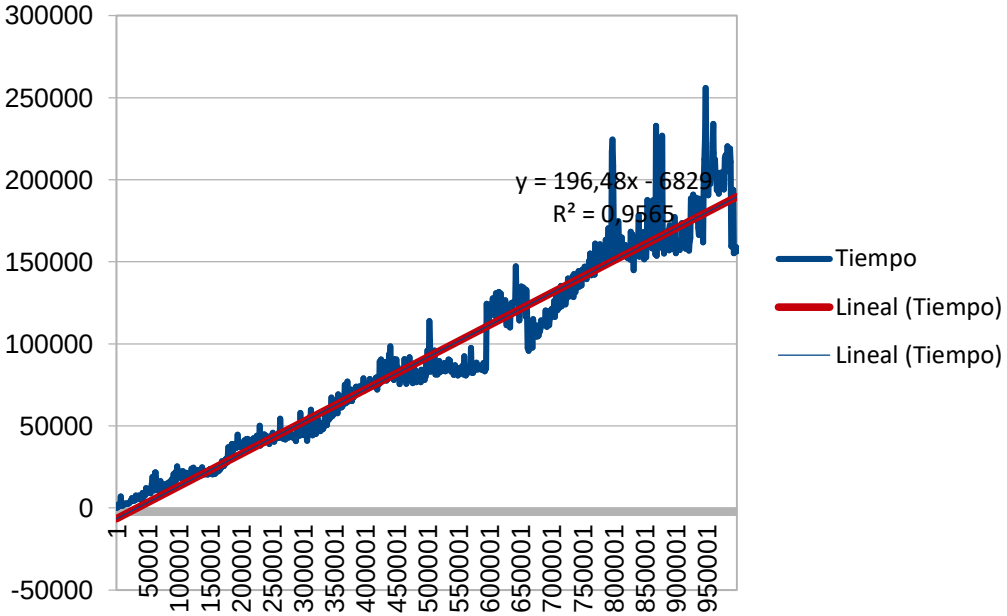
4. (hasta 1 punto) Validación del algoritmo, justificando los experimentos realizados para asegurar que el algoritmo funciona correctamente, y en su caso programas utilizados para la validación.

   Para validar el algoritmo hicimos un estudio con todos los casos posibles, sobre todo con los que podían dar más fallos a la hora la función combinar, más tarde hicimos un función que generaba cadenas de n caracteres, generábamos cadenas muy grandes para hacer las pruebas, y utilizábamos el resultado de la función iterativa para contrastar el resultado con el de la función recursiva.

5. (hasta 1,5 puntos) Estudio experimental del tiempo de ejecución para distintos tamaños de problema. Habrá que experimentar con tamaños suficientemente grandes para obtener resultados significativos. En el caso de resolver la opción a) habrá que estudiar cómo influye en el tiempo de ejecución hacer 2 o 3 subdivisiones.

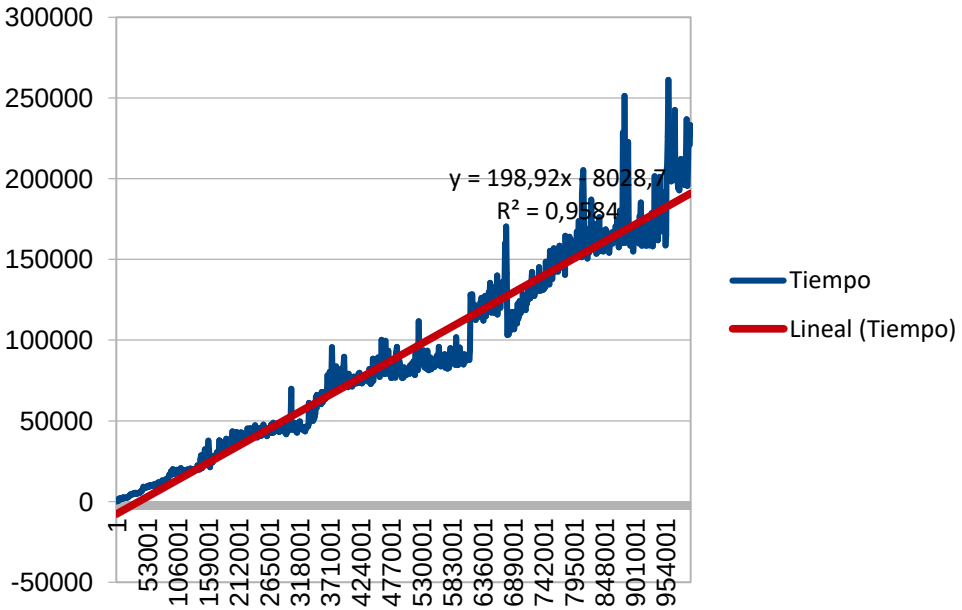


Caso base 5:
Las diversas variaciones respecto a la línea de regresión creemos que son debidas a los distintos niveles de cache, pero en general la gráfica muestra que el algoritmo se ejecuta en tiempo lineal.
Eq: 231,56x-1,6639



Caso base 1:
En general la gráfica muestra que el algoritmo se ejecuta en tiempo lineal.
Eq: 196,48x-6829

Caso base 2:
En general la gráfica muestra que el algoritmo se ejecuta en tiempo lineal.
Eq: 198,92x-8028,7



6. (hasta 1 punto) Contraste del estudio teórico y el experimental, buscando justificación a las discrepancias entre los dos estudios.

   Las tres ecuaciones que ajustan las gráficas para distintos casos bases (1, 5, 10) corresponden al orden de n, todas las gráficas pertenecen al orden de n por que no influye ya que la función a la que se llama en el caso base (iterativo) y la parte recursiva pertenecen a n según el estudio teórico. Por lo tanto los resultados son congruentes.

   Para comprobar que la probabilidad de la ejecución de instrucciones en el combinar era correcta probamos para casos pequeños y casos muy grandes y nos salían resultados acordes ya que con una cadena de 20000 salían 6 letras de solución y para 60000 también y así para varios casos que cada vez la probabilidad de que haya más silabas validas es menor.