Especificación matemática ejercicios 71-80

José Simón Ramos Sandoval

Universidad Nacional de Colombia, Programación de computadores 2021-1s

1 CADENAS

Ejercicio 71
Desarrollar un algoritmo que reciba como entrada un caracter y de como salida el número de ocurrencias de dicho caracter en una cadena de caracteres.

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conoce un caracter y una cadena de caracteres

1.2 Objetos desconocidos:

- Se desconoce el número de veces que aparece dicho caracter en la cadena de caracteres

1.3 Relación entre objetos:

- Cada vez que el caracter aparezca en la cadena, un contador aumentará

2. Diseño y prueba conceptual

Entradas: 2 variables: Un caracter x ($x \in \mathbb{ASCII}$) y una cadena de caracteres ($cad \in \mathbb{ASCII}^*$)

Salidas: 1 variable: Número de ocurrencias del caracter en la cadena $y \in \mathbb{N}$

Relación:

$$\begin{array}{rcl} ocurrencia : \mathbb{ASCII} \times \mathbb{ASCII}^* & \rightarrow & \mathbb{N} \\ ocurrencia(x, cad) & \rightarrow & \begin{cases} y \text{ tal que} y = y + 1 & si \quad \exists_{i=0}^{|cad|-1} cad_i = x \\ y & \text{En otro caso} \end{cases} \end{array}$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_71_SimonRamos.py»

Ejercicio 72
Desarrollar un algoritmo que reciba como entrada dos cadenas y determine si la primera es subcadena de la segunda. (No se deben usar operaciones de subcadenas propias del lenguaje de programación).

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conocen dos cadenas de caracteres
- Se sabe que una cadena dada es subcadena de otra si todos los caracteres de la primera se encuentran en orden en la segunda

1.2 Objetos desconocidos:

- Se desconoce si una cadena dada es subcadena de otra

1.3 Relación entre objetos:

- Si una cadena dada es subcadena de otra, todos los caracteres de la primera cadena se encuentran en orden en la segunda
- SI primero creamos una función que extraiga fragmentos de una cadena, podemos hacer una segunda función que compare la primera cadena con fragmentos de la segunda del tamaño de la primera. SI alguno de los fragmentos es igual, será subcadena, de lo contrario, no lo será
- Lo anterior se puede plantear de forma recursiva, saber si una cadena es subcadena de otra es fácil si un fragmento del tamaño de la segunda cadena es igual a esta segunda cadena, si no es el último fragmento, comprobar si es algún fragmento en la longitud de la cadena -1. Si en algún punto la longitud de la segunda cadena es mayor que los fragmentos que se pueden comparar, entonces no es subcadena

2. Diseño y prueba conceptual

Función que saca los fragmentos:

Entradas: 3 variables: Una cadena de caracteres a la cual se le sacarán los fragmentos ($cad \in \mathbb{ASCII}^*$); una posición de inicio ($ini \in \mathbb{N}$) y una posición de fin ($fin \in \mathbb{N}$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} fragmento : \mathbb{ASCII}^* \times \mathbb{N} \times \mathbb{N} & \rightarrow & \mathbb{ASCII}^* \\ fragmento(cad, ini, fin) & \rightarrow & C \text{ tal que } \forall_{i=ini}^{fin-1} cad_i = C_i \end{array}$$

Función que determina si es subcadena

Entradas: 3 variables: Una cadena de caracteres sub que se desconoce si es subcadena ($sub \in \mathbb{ASCII}^*$); una cadena de caracteres cad que se determinará si contiene a una subcadena ($cad \in \mathbb{ASCII}^*$). número de caracteres de la cadena cad ($x \in \mathbb{N}$)

Salidas: 1 variable: 1 valor booleano (True si la primera es subcadena de la segunda, False en otro caso)

Relación:

$$subcadena : \mathbb{ASCII} \times \mathbb{ASCII}^* \times \mathbb{N} \to \mathbb{B}$$

$$subcadena(sub, cad, n) \to \begin{cases} False & si \quad x < |sub| \\ fragmento(cad, x - |sub|, x) = sub \vee subcadena(sub, cad, x-1) & si \quad \text{En otro caso} \end{cases}$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_72_SimonRamos.py»

Ejercicio 73:

Desarrollar un algoritmo que reciba dos cadenas de caracteres y determine si la primera está incluida en la segunda. Se dice que una cadena está incluida en otra, si todos los caracteres (con repeticiones) de la cadena están en la segunda cadena sin tener en cuenta el orden de los caracteres.

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conocen dos cadenas de caracteres

1.2 Objetos desconocidos:

- Se desconoce si una cadena dada está incluida en otra

1.3 Relación entre objetos:

- Si una cadena dada esta incluida en otra, entonces las repeticiones de los caracteres en la primera cadena son menores o iguales a las repeticiones de los caracteres en la segunda

2. Diseño y prueba conceptual

Función que saca las repeticiones de un caracter en una cadena

Para esto utilizamos la función especificada en el ejercicio 71

Entradas: 2 variables: Un caracter x ($x \in \mathbb{ASCII}$) y una cadena de caracteres ($cad \in \mathbb{ASCII}^*$)

Salidas: 1 variable: Número de ocurrencias del caracter en la cadena $y \in \mathbb{N}$

Relación:

$$ocurrencia : \mathbb{ASCII} \times \mathbb{ASCII}^* \to \mathbb{N}$$

$$ocurrencia(x, cad) \to \begin{cases} y \text{ tal que} y = y + 1 & si \quad \exists_{i=0}^{|cad|-1} cad_i = x \\ y & \text{En otro caso} \end{cases}$$

Función que determina si una cadena está contenida en otra:

Entradas: 2 variables:: Dos cadenas de caracteres ($cad1 \in \mathbb{ASCII}^*$) y ($cad2 \in \mathbb{ASCII}^*$)

Salidas: 1 variable: Un valor booleano (True si está contenida, False en otro caso)

Relación:

$$\begin{array}{rcl} contenido : \mathbb{ASCII}^* \times \mathbb{ASCII}^* & \rightarrow & \mathbb{B} \\ contenido(cad1, cad2) & \rightarrow & \begin{cases} True & si \quad \forall_{i=0}^{|cad1|-1}(ocurrencia(cad1_i, cad1) \leq ocurrencia(cad1_i, cad2) \\ False & \text{En otro caso} \end{cases} \end{array}$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_73_SimonRamos.py»

Ejercicio 74

Desarrollar un algoritmo que invierta una cadena de caracteres.

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conoce una cadena de caracteres

1.2 Objetos desconocidos:

- Se desconoce la cadena invertida de la cadena dada

1.3 Relación entre objetos:

- Al invertir una cadena el último elemento de esta, pasará a ser el primero, esto se puede definir de forma recursiva si paso el caracter n de la cadena y lo concateno con los otros n-1 elementos, y después paso el elemento n-1 de la cadena. Cuando solo me quede un elemento, se pasará solo este elemento

2. Diseño y prueba conceptual

Función que invierte

Entradas: 2 variables: 1 cadena de caracteres: ($cad \in \mathbb{ASCII}^*$) y la cantidad (n) de caracteres de la cadena($n \in \mathbb{N}$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} invierte : \mathbb{ASCII}^* \times \mathbb{N} & \rightarrow & \mathbb{ASCII}^* \\ invierte(cad, n) & \rightarrow & \begin{cases} cad_0 & si \quad x = 1 \\ cad_{n-1} + invierte(cad, n-1) & \text{En otro caso} \end{cases} \end{array}$$

Función que integra todo:

Entradas: 1 cadena de caracteres: ($cad \in \mathbb{ASCII}^*$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} final : \mathbb{ASCII}^* & \rightarrow & \mathbb{ASCII}^* \\ final(cad) & \rightarrow & invierte(cad, |cad|) \end{array}$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_74_SimonRamos.py»

Ejercicio 75

Desarrollar un algoritmo que determine si una cadena de caracteres es palíndrome. Una cadena se dice palíndrome si al invertirla es igual a ella misma.

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conoce una cadena de caracteres

1.2 Objetos desconocidos:

- Se desconoce si la cadena dada es palíndrome o no

1.3 Relación entre objetos:

- Para determinar si una cadena es palíndrome en primer lugar se deben tener todos los caracteres como minúsculas, esto debido a que una palabra, por ejemplo "*Ala*", es palíndrome aunque el primer caracter sea mayúscula. Así que para trabajar con facilidad y determinar ese tipo de casos se deben tener todos los caracteres como minúsculas
- Una vez que se determina que todos los caracteres son minúsculas, una cadena es palíndrome si el último caracter y el primero son iguales. Se hace la misma comparación con el penúltimo caracter y el segundo, así hasta que se llegue al caracter de la mitad. Si para todos los caracteres se cumple lo anterior, la cadena será palíndrome.

2. Diseño y prueba conceptual

Función que convierte a todos los caracteres en minúsculas

Entradas: Una cadena de caracteres de tamaaño n $cad \in \mathbb{ASCII}$

Salidas: Una cadena de caracteres en la que no hay letras mayúsculas $minus \in \mathbb{ASCII}^*$

Relación:

$$convierte(\mathbb{ASCII}^n) \rightarrow \mathbb{ASCII}^n$$

$$convierte(cad) \rightarrow minus \text{ tal que } \forall_{i=0}^{n-1} \begin{cases} minus_i = cad_i - "A" + "a" & si \quad "A" \leq cad_i \leq "Z" \\ minus_i = cad_i & \text{En otro caso} \end{cases}$$

Función que determina si es palindrome o no:

Entradas: Una cadena de caracteres de tamaño n $cadena \in \mathbb{ASCII}^n$ en donde todos los caracteres son minúsculas

Salidas: Un booleano que determina si la cadena es palíndrome o no $v \in \mathbb{B}$

Relación:

$$
\begin{array}{rcl}
palindrome1(\mathbb{ASCII}^n) & \rightarrow & \mathbb{B} \\
palindrome1(cad) & \rightarrow & \begin{cases} True & si \quad \forall_{i=n,i=i-1}^{\lfloor \frac{n}{2} \rfloor} cad_{-i} = cad_{i-1} \\ False & \text{En otro caso} \end{cases}
\end{array}
$$

Función que integra todo:

Entradas: Una cadena de caracteres de tamaño n $cadena \in \mathbb{ASCII}^n$

Salidas: Un booleano que determina si la cadena es palíndrome o no $v \in \mathbb{B}$

Relación:

$$
\begin{array}{rcl}
palindrome(\mathbb{ASCII}^n) & \rightarrow & \mathbb{B} \\
palindrome(cad) & \rightarrow & palindrome1(convierte(cad))
\end{array}
$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_75_SimonRamos.py»

Ejercicio 76

Desarrollar un algoritmo que determina si una cadena de caracteres es frase palíndrome. Una cadena se dice frase palíndrome si la cadena al eliminarle los espacios es palíndrome.

1. Análisis y especificación:

1.1 Objetos conocidos:

- Se conoce una cadena de caracteres

1.2 Objetos desconocidos:

- Se desconoce si la cadena de caracteres es una frase palíndrome o no

1.3 Relación entre los objetos:

- La cadena dada será frase palíndrome si al eliminar los espacios entre las palabras tenemos una cadena palíndrome. Por lo que es posible implementar una función que haga esta separación y posteriormente volver a llamar las funciones especificadas en el punto anterior

2. Diseño y prueba conceptual

Función que elimina los espacios:

Entradas: Una cadena de caracteres de tamaño n $cad \in \mathbb{ASCII}^n$

Salidas: Una cadena de caracteres que no contendrá espacios $s \in \mathbb{ASCII}^*$

Relación:

$$
\begin{array}{rcl}
espacios : ASCII^n & \rightarrow & \mathbb{ASCII}^* \\
espacios(cad) & \rightarrow & s \text{ tal que si } \forall_{i=0}^{n-1} cad_i \neq \text{" "} \text{ entonces } s_i = cad_i
\end{array}
$$

Una vez se tiene la cadena sin espacios, se pueden aplicar las funciones del punto anterior para determinar si es palíndrome

Función que convierte a todos los caracteres en minúsculas

> **Entradas:** Una cadena de caracteres de tamaaño n $cad \in \mathbb{ASCII}^n$la cual no tiene espacios
>
> **Salidas:** Una cadena de caracteres en la que no hay letras mayúsculas $minus \in \mathbb{ASCII}^*$
>
> **Relación:**

$$\begin{array}{rcl} convierte(\mathbb{ASCII}^n) & \rightarrow & \mathbb{ASCII}^n \\ convierte(cad) & \rightarrow & minus \text{ tal que } \forall_{i=0}^{n-1} \begin{cases} minus_i = cad_i - "A" + "a" & si \quad "A" \leq cad_i \leq "Z" \\ minus_i = cad_i & \text{En otro caso} \end{cases} \end{array}$$

Función que determina si es palindrome o no:

> **Entradas:** Una cadena de caracteres de tamaño n $cadena \in \mathbb{ASCII}^n$ en donde todos los caracteres son minúsculas y no tiene espacios
>
> **Salidas:** Un booleano que determina si la cadena es palíndrome o no $v \in \mathbb{B}$
>
> **Relación:**

$$\begin{array}{rcl} palindrome1(\mathbb{ASCII}^n) & \rightarrow & \mathbb{B} \\ palindrome1(cad) & \rightarrow & \begin{cases} True & si \quad \forall_{i=n, i=i-1}^{\lfloor \frac{n}{2} \rfloor} cad_{-i} = cad_{i-1} \\ False & \text{En otro caso} \end{cases} \end{array}$$

Función que integra todo:

> **Entradas:** Una cadena de caracteres de tamaño n $cadena \in \mathbb{ASCII}^n$
>
> **Salidas:** Un booleano que determina si la cadena es palíndrome o no $v \in \mathbb{B}$
>
> **Relación:**

$$\begin{array}{rcl} palindrome(\mathbb{ASCII}^n) & \rightarrow & \mathbb{B} \\ palindrome(cad) & \rightarrow & palindrome1(convierte(espacios(cad))) \end{array}$$

3. Codificación

> El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_76_SimonRamos.py»

Ejercicio 77

Desarrollar un algoritmo que realice el corrimiento circular a izquierda de una cadena de caracteres. El corrimiento circular a izquierda es pasar el primer carácter de una cadena como último carácter de la misma.

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conoce una cadena de caracteres

1.2 Objetos desconocidos:

- Se desconoce la cadena con el corrimiento circular a izquierda

1.3 Relación entre objetos:

- Ya que si colocamos un subínice negativo, la cadena cuenta las posiciones de derecha a izquierda (Por ejemplo, si tenemos como subíndice a -1 la cadena retorna el último elemento) podemos utilizar subínices negativos para devolver del último al primero.
- El problema se soluciona si puedo devolver el segundo elemento y concatenarle el resto de elementos, si devuelvo el elemento 2 de la cadena será el mismo proceso, al llegar a la última posición solo se retornará el primer elemento.

2. Diseño y prueba conceptual
Función que hace el corrimiento parcial:

Entradas: 2 variables: Una cadena de caracteres ($cad \in \mathbb{ASCII}^*$)
y la cantidad de caracteres (n) de la cadena ($n \in \mathbb{N}$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} corriparcial : \mathbb{ASCII}^* \times \mathbb{N} & \rightarrow & \mathbb{ASCII}^* \\ corriparcial(cad, n) & \rightarrow & \begin{cases} cad_0 & si \quad n = 1 \\ cad_{-n+1} + corrparcial(cad, n-1) & \text{En otro caso} \end{cases} \end{array}$$

Función que hace el corrimiento a izquierda:

Entradas: Una cadena de caracteres ($cad \in \mathbb{ASCII}^*$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} corrimiento : \mathbb{ASCII}^* & \rightarrow & \mathbb{ASCII}^* \\ corrimiento(cad) & \rightarrow & corriparcial(cad, |cad|) \end{array}$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_77_SimonRamos.py»

Ejercicio 78
Desarrollar un algoritmo que realice el corrimiento circular a derecha de una cadena de caracteres. El corrimiento circular a derecha de una cadena es poner el último carácter de la cadena como primer carácter de la misma.
1. Análisis y especificación
1.1 Objetos conocidos:

- Se conoce una cadena de caracteres

1.2 Objetos desconocidos:

- Se desconoce la cadena con el corrimiento circular a derecha

1.3 Relación entre objetos:

- Ya que si colocamos un subínice negativo, la cadena cuenta las posiciones de derecha a izquierda (Por ejemplo, si tenemos como subíndice a -1 la cadena retorna el último elemento) podemos utilizar subínices negativos para devolver del último al primero.
- El problema se soluciona si puedo devolver el elemento n-2 y concatenarle el resto de elementos, si devuelvo el elemento n-3 de la cadena será el mismo proceso, al llegar a la posición cero, se retornará el último elemento (o el elemento n_{-1}).

2. Diseño y prueba conceptual
Función que hace el corrimiento parcial:

Entradas: 2 variables: Una cadena de caracteres ($cad \in \mathbb{ASCII}^*$) y la cantidad de caracteres (n) de la cadena ($n \in \mathbb{N}$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} corriparcial : \mathbb{ASCII}^* \times \mathbb{N} & \rightarrow & \mathbb{ASCII}^* \\ corriparcial(cad, n) & \rightarrow & \begin{cases} cad_{-1} & si \quad n = 1 \\ corrparcial(cad, n-1) + cad_{n-2} & \text{En otro caso} \end{cases} \end{array}$$

Función que hace el corrimiento a derecha:

Entradas: Una cadena de caracteres ($cad \in \mathbb{ASCII}^*$)

Salidas: 1 cadena de caracteres

Relación:

$$\begin{array}{rcl} corrimiento : \mathbb{ASCII}^* & \rightarrow & \mathbb{ASCII}^* \\ corrimiento(cad) & \rightarrow & corriparcial(cad, |cad|) \end{array}$$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_78_SimonRamos.py»

Ejercicio 79
Desarrollar un algoritmo que codifique una cadena de caracteres mediante una cadena de correspondencias de caracteres dada. La cadena de correspondencias tiene como el primer caracter el caracter equivalente para el caracter 'a', el segundo caracter para la 'b'y así sucesivamente hasta la 'z'. No se tiene traducción para las mayúsculas ni para la 'ñ'.
1. Análisis y especificación
1.1 Objetos conocidos:

- Se conocen una cadena de caracteres a codificar y una cadena de correspondencia.
- El caracter de la primera psoicón de la cadena de correspondencia se codifica como la “a” , la segunda como la “b” y así sucesivamente.

1.2 Objetos desconocidos:

- Se desconoce la cadena final después de aplicarle la codificación con la cadena de correspondencia

1.3 Relación entre los objetos:

- El primer caracter de la cadena de correspondencia se codifica como una “a”, el segundo como una “b” y así sucesivamente. Por lo anterior, podemos decir que tomamos un caracter de la cadena a codificar, le restamos el código ASCII de la letra “a” y el resultado será la posición del caracter correspondiente en la cadena de correspondencia. Si la cadena de correspondencia no abarca a todos los caracteres del alfabeto, entonces los caracteres que no tienen correspondencia se mantendrán igual
- Además, no hay correspondencia para las letras mayúsculas o para la ñ

2. Diseño y prueba conceptual

Entradas: Dos cadenas de caracteres: Una cadena a codificar de tamaño n ($cad \in \mathbb{ASCII}^n$) y una cadena de correspondencia de tamaño m ($cor \in \mathbb{ASCII}^m$)

Salidas: Una cadena de caracteres ya codificada del mismo tamaño n ($c \in \mathbb{ASCII}^m$)

Relación:

$$codifica : \mathbb{ASCII}^n \times \mathbb{ASCII}^m \rightarrow \mathbb{ASCII}^n$$

$$codifica(cad, cor) \rightarrow c \text{ tal que } \forall_{i=0}^{n-1} \begin{cases} c_i = cor_k & si \quad "a" \leq cad_i \leq "z" \wedge k < m \\ c_i = cad_i & si \quad \text{En otro caso} \end{cases}$$

Donde $n =$ la cantidad de caracteres de cad, $m =$ cantidad de caracteres de cor y $k = cad_i - "a"$

$codifica : \mathbb{ASCII}^n \times \mathbb{ASCII}^m \to \mathbb{ASCII}^n$

$codifica(cad, cor) \to c$ tal que $\forall_{i=0}^{n-1} \begin{cases} c_i = cor_{cad_i - "a"} & si \quad "a" \leq cad_i \leq "z" \wedge cad_i - "a" < m \\ c_i = cad_i & si \quad \text{En otro caso} \end{cases}$

Donde n es la cantidad de caracteres de cad y m la cantidad de caracteres de cor

3. Codificación

> El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_79_SimonRamos.py»

Ejercicio 80

Desarrollar un algoritmo que decodifique una cadena de caracteres mediante una cadena de correspondencias de caracteres dada. La cadena de correspondencias tiene como el primer caracter el caracter equivalente para el caracter 'a', el segundo caracter para la 'b'y así sucesivamente hasta la 'z'. No se tiene traducción para las mayúsculas ni para la 'ñ'.

1. Análisis y especificación

1.1 Objetos conocidos:

- Se conocen dos cadenas de caracteres: Una de ellas es la cadena decodificada $dec \in \mathbb{ASCII}^n$y la otra es la cadena de correspondencia $cor \in \mathbb{ASCII}^m$
- El primer caracter de la cadena se corresponde con la letra "a", el segundo con la letra "b" y así sucesivamente hasta la "z"

1.2 Objetos desconocidos:

- Se desconoce la cadena resultante al decodificar la cadena decodificada en base a la cadena de correspondencia

1.3 Relación entre los objetos:

- La posición del caracter en la cadena de correspondencia mas el código de la letra "a" se corresponde con el código ASCII de la letra a la que corresponde ese caracter. Si la cadena de correspondencia no contiene los caracteres suficientes para abarcar a todo el alfabeto de la a-z sin contar a la ñ, los caracteres que no puedan ser decodificados se mantendrán igual. No hay decodificación para las mayúsculas y la ñ. $(cor_i + "a")$
- Así que en primer lugar es necesario conocer la posición del caracter de la cadena decodificada en la cadena de codificación, a esta psoción sumarle el código de la "a" y asó obtendremos el código de la cadena de correspondencia

2. Diseño y prueba conceptual

Función que detecta la posición del caracter

Entradas: Dos variables: Un caracter ($cr \in \mathbb{ASCII}$), una cadena de tamaño m ($cor \in \mathbb{ASCII}^m$) en donde se buscará en qué posición contiene al caracter cr

Salidas: Un natural $p \in \mathbb{N}$que equivale a la posicón del caracter cr en con

Relación:

$$\begin{aligned} posicion : \mathbb{ASCII} \times \mathbb{ASCII}^m &\rightarrow & \mathbb{N} \\ posicion(cr, cor) &\rightarrow & \begin{cases} p & si \quad \exists_{p=0}^{m-1} cor_p = cr \\ m & \text{En otro caso} \end{cases} \end{aligned}$$

Donde m es la cantidad de caracteres de cor

Función que decodifica:

Entradas: Dos cadenas de caracteres: Una cadena a decodificar de tamaño n ($dec \in \mathbb{ASCII}^n$) y otra cadena de correspondencia de tamaño m ($cor \in \mathbb{ASCII}^m$)

Salidas: Una cadena de caracteres de tamaño n ($c \in \mathbb{ASCII}^n$)

Relación:

$$decodifica : \mathbb{ASCII}^n \times \mathbb{ASCII}^m \rightarrow \mathbb{ASCII}^n$$

$$decodifica(dec, cor) \rightarrow c \text{ tal que } \forall_{i=0}^{n-1} \begin{cases} c_i = posicion(dec_i, cor) + "a" & si \quad "a" \leq dec_i \leq "z" \wedge psocion(dec_i, cor) \neq m \\ c_i = dec_i & \text{En otro caso} \end{cases}$$

Donde m es la cantidad de caracteres de cor

$$decodifica : \mathbb{ASCII}^n \times \mathbb{ASCII}^m \rightarrow \mathbb{ASCII}^n$$

$$decodifica(dec, cor) \rightarrow c \text{ tal que } \forall_{i=0}^{n-1} \begin{cases} c_i = k + "a" & si \quad "a" \leq dec_i \leq "z" \wedge k \neq m \\ c_i = dec_i & \text{En otro caso} \end{cases}$$

Donde m es el número de caracteres de con y $k = posicion(dec_i, cor)$

3. Codificación

El código donde se resuelve el ejercicio se encuentra en el archivo «Ejercicio_80_SimonRamos.py»