Programación Funcional

Cifrando Mensajes

Parte I: Cifrado César

Una de las técnicas de cifrado más simples es conocida como *El código de César o Cifrado César*. Este es un cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, utilizando este método con *desplazamiento* = 3, se tendría el siguiente cifrado:

mensaje original: introduccion a la programacion
mensaje cifrado: lqwurgxfflrq d od surjudpdflrq

Por simplicidad, cifraremos sólo las letras minúsculas de los mensajes. Además, vamos a eliminar la letra "ñ" del conjunto de letras minúscula y no consideraremos a las vocales con tildes como letras minúsculas, también por simplicidad.

El orden natural del alfabeto será considerando su inicio en la letra "a" (posición 0) y finalizando en la letra "z" (en la posición 25). Por esto podemos mencionar que desplazar 2 posiciones hacia adelante a la letra "a" hace referencia a la letra "c", mientras que desplazar 2 posiciones hacia atrás a la letra "a" hace referencia a la letra "y". Además, los espacios y otras letras no serán cifrados, pero se mantendrán en su posición original.

Función 1

problema esMinuscula (c: Char) : Bool {
    requiere: {True}
    asegura: {**res** es igual a true si y sólo si *c* es una letra minúscula (entre "a" y "z" ambos inclusive, pero sin incluir "ñ" ni vocales con tildes)}
}

Ejemplo:

entrada: c = 'd'
res: True

Función 2

problema letraANatural (c: Char) : Z {
    requiere: {*c* es una letra minúscula}
    asegura: {**res** es igual a un número entre 0 y 25 (ambos inclusive) que indica el orden en el cual la letra se encuentra en el abecedario.}
}

Ejemplo:

entrada: c = 'b'
res: 1

Función 3

problema desplazar (c: Char, n: Z) : Char {
requiere: {True}
asegura: {Si *c* no es minúscula, entonces **res** = *c*}
asegura: {Si *c* es minúscula y $n \geq 0$, entonces **res** es igual a la letra minúscula que esté *n* posiciones más adelante en el abecedario (rotando en caso de llegar al final)}
asegura: {Si *c* es minúscula y $n \leq 0$, entonces **res** es igual a la letra minúscula que esté *n* posiciones más atrás en el abecedario (rotando en caso de llegar al principio)}
}

Ejemplo:

entrada: c = 'a', n = 3
res: 'd'

Función 4

problema cifrar (s: String, n: Z) : String {
requiere: {$n \geq 0$}
asegura: {La longitud de **res** es igual a la longitud de *s*}
asegura: {Si **s[i]** es minúscula, entonces **res[i]** es la letra correspondiente a desplazar *n* veces hacia adelante en el alfabeto a **s[i]** (para cualquier $i \in Z$ que cumpla $0 \leq i < |s|$)}
asegura: {Si **s[i]** no es minúscula, entonces **res[i]** = **s[i]** (para cualquier $i \in Z$ que cumpla $0 \leq i < |s|$)}
}

Ejemplo:

entrada: s = "computacion", n = 3
res: "frpsxwdflrq"

Función 5

problema descifrar (s: String, n: Z) : String {
requiere: {$n \geq 0$}
asegura: {La longitud de **res** es igual a la longitud de *s*}
asegura: {Si **s[i]** es minúscula, entonces **res[i]** es la letra correspondiente a desplazar *n* veces más atrás en el alfabeto a **s[i]** (para cualquier $i \in Z$ que cumpla $0 \leq i < |s|$)}
asegura: {Si **s[i]** no es minúscula, entonces **res[i]** = **s[i]** (para cualquier $i \in Z$ que cumpla $0 \leq i < |s|$)}
}

Ejemplo:

entrada: "frpsxwdflrq", n = 3
res: "computacion"

Función 6

problema cifrarLista (ls: *seq*⟨String⟩) : *seq*⟨String⟩ {
requiere: {True}
asegura: {La longitud de **res** es igual a la longitud de *ls*}
asegura: {Para todo $i \in Z$, $0 \leq i < |ls|$, **res**[*i*] = *cifrar*(*ls*[*i*], *i*)}
}

Ejemplo:

entrada: ls = ["compu", "labo", "intro"]
res: ["compu", "mbcp", "kpvtq"]

Función 7

problema frecuencia (s: String) : *seq*⟨R⟩ {
requiere: {True}
asegura: {La longitud de **res** es 26}
asegura: {Cada elemento de **res** tiene un valor entre 0 y 100, ambos inclusive}
asegura: {Si no hay letras minúsculas en *s*, todas las posiciones de **res** tienen el valor 0.}
asegura: {En caso contrario, la primera posición de **res** contiene el porcentaje de "a"s (respecto de la cantidad total de letras minúscula en *s*), la segunda el porcentaje de "b"s (respecto de la cantidad total de letras minúscula en *s*), y así sucesivamente hasta la última posición de **res** que contiene el valor porcentual de letras "z" (respecto de la cantidad total de letras minúscula en *s*) que hay en *s*}
}

Ejemplo:

entrada: s = "taller"
res: [16.666668,0.0,0.0,0.0,16.666668,0.0,0.0,0.0,0.0,0.0,0.0,33.333336,
0.0,0.0,0.0,0.0,0.0,16.666668,0.0,16.666668,0.0,0.0,0.0,0.0,0.0,0.0]

Función 8

problema cifradoMasFrecuente (s: String, n: Z) : (Char × R) {
requiere: {*s* contiene al menos una letra minúscula}
asegura: {Considerando el resultado de cifrar *s* con desplazamiento *n*, **res** contiene como primer componente alguna de las letras minúscula con mayor frecuencia, y como segundo componente la frecuencia de dicha letra.}
}

Ejemplo:

entrada: "taller", n = 3
res: ('o', 33.333336)

Función 9

problema esDescifrado (s1: String, s2:String) : Bool {
requiere: {True}
asegura: {**res** = *true* si y sólo si *s*2 = *cifrar*(*s*1*, n*), para algún n}
}

Ejemplo:

entrada: s1 = "taller", s2 = "compu"
res: False

Función 10

problema todosLosDescifrados (ls: *seq*⟨String⟩) : *seq*⟨(String × String)⟩ {
requiere: {No hay elementos repetidos en *ls*}
asegura: {No hay elementos repetidos en **res**}
asegura: {**res** contiene todos los pares (*s*1*, s*2) tales que *s*1 = *descifrar*(*s*2*, n*) donde *n* es un número cuyo modulo 26 es distinto de 0, y tanto *s1* como *s2* pertenecen a *ls*}
asegura: {**res** solamente contiene pares (*s*1*, s*2) tales que *s*1 = *descifrar*(*s*2*, n*) donde *n* es un número cuyo modulo 26 es distinto de 0, y tanto *s1* como *s2* pertenecen a *ls*}
}

Ejemplo:

entrada: ls = ["compu", "frpsx", "mywza"]
res: [("compu", "frpsx"), ("frpsx", "compu")]

Parte II: Código Vigenere

En este cifrado, una letra minúscula se sustituye por otra letra minúscula, pero desplazada un cierto número de posiciones. Este número de desplazamiento se determina mediante una *clave*, que es una palabra o frase repetida *tantas veces como sea necesario para cifrar el mensaje completo*. A diferencia de otros cifrados, como el Cifrado César, el cifrado Vigenere utiliza diferentes desplazamientos de acuerdo con las letras de la clave, lo que lo hace más resistente al análisis criptográfico. Sólo vamos a considerar cifrar las letras minúsculas de los mensajes. Los espacios y otras letras se mantendrán en el mensaje, pero sin cifrarlas. Por ejemplo, para cifrar el mensaje "introducción a la programación" con la clave "compu", primero debe expandirse la clave a la longitud del mensaje, y luego desplazar cada letra del mensaje original por la letra de la clave correspondiente:

mensaje original: introduccion a la programacion
clave: compu

clave expandida: compucompucompucompucompucompu

mensaje cifrado: kbfgifiorcqb p no elqudpgcqudh

Función 11

problema expandirClave (clave: String, n: Z) : String {
requiere: {$n > 0$ y $|clave| > 0$}
requiere: {Todos los caracteres de *clave* son letras minúsculas}
asegura: {La longitud de **res** es igual a n}
asegura: {Para cualquier $i \in Z$ tal que $0 \leq i < |\textbf{res}|$, **res**[i] = *clave*[i mod $|clave|$] }
}

Ejemplo:

entrada: clave = "compu", n = 8
res: "compucom"

Función 12

problema cifrarVigenere (s: String, clave: String) : String {
requiere: {Todos los caracteres de *clave* son letras minúsculas}
requiere: {$|clave| > 0$}
asegura: {La longitud de **res** es igual a la longitud de s}
asegura: {Para cualquier $i \in Z$ tal que $0 \leq i < |\textbf{res}|$, **res**[i] = *desplazar*(*s*[*i*], *n*), donde *n* es igual al número que representa la letra minúscula en la posición i de la clave expandida}
}

Ejemplo:

entrada: s = "computacion", clave = "ip"
res: "kdueciirqdv"

Función 13

problema descifrarVigenere (s: String, clave: String) : String {
requiere: {Todos los caracteres de *clave* son letras minúsculas}

requiere: {|*clave*| > 0}
asegura: {La longitud de **res** es igual a la longitud de *s*}
asegura: {Para cualquier *i* tal que 0 ≤ *i* < |**res**|, **res**[i] = *desplazar*(*s*[*i*], −*n*), donde *n* es igual al número que representa la letra minúscula *expandirClave*[*i*]}

}

Ejemplo:

entrada: s = "kdueciirqdv", clave = "ip"
res: "computacion"

Función 14

problema peorCifrado (s: String, claves: *seq*⟨*String*⟩) : String {
requiere: {Todos los caracteres de *s* son letras minúsculas}
requiere: {La secuencia *claves* tiene al menos un elemento}
requiere: {Todos los elementos de *claves* tienen longitud mayor a 0, y todos sus caracteres son letras minúsculas}
asegura: {**res** pertenece a la secuencia *claves*}
asegura: {De todos los elementos de *claves*, **res** es alguno de los que deja el mensaje *s* a menor *distancia* de su cifrado Vigenere}

}

La *distancia* entre dos secuencias *s*1 y *s*2 de igual longitud se calcula como

$$\sum_{i=0}^{|S1|-1} |LetraANatural(s1[i]) - letraANatural(s2[i])|$$

Ejemplo:

entrada: s = "computacion", claves = ["ip", "asdef", "ksy"]
res: "asdef"

Función 15

problema combinacionesVigenere (msjs: *seq*⟨String⟩, claves: *seq*⟨String⟩, cifrado: String) : *seq*⟨(String × String)⟩ {
requiere: {Las longitudes de *msjs* y *claves* son iguales}
requiere: {Todos los elementos de *claves* tienen longitud mayor a 0, y todos sus caracteres son letras minúsculas}
requiere: {*msjs* no tiene elementos repetidos}
requiere: {*claves* no tiene elementos repetidos}
asegura: {La longitud de **res** es igual a la cantidad de todas las posibles combinaciones de *msjs* y *claves* que cifrados son iguales al parámetro *cifrado*}
asegura: {**res** contiene todas las posibles combinaciones de *msjs* y *claves* que cifrados son iguales al parámetro *cifrado*}

}

Ejemplo:

entrada: msjs = ["hola", "mundo"], claves = ["a", "b"], cifrado = "ipmb"
res: [("hola", "b")]