Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 2

Matemáticas Discretas para la Computación

Autor: Lukas Pavez
RUT: 19.401.577-1
Profesor: Pablo Barcelo B.
Auxiliares: Arniel Labrada D.
Ilana Mergudich T.
Ayudantes: Jaime Salas T.
Javier Marinković
Joaquín Romero
Pablo Paredes Haz
Fecha de entrega: 13/07/2018

1. P1

1.1. Parte Teórica

Dada la palabra str, el numero de permutaciones palíndromas se puede calcular de la siguiente forma:
Primero, se contarán las repeticiones de todos los caracteres que conforman la palabra, si las repeticiones de dos o más caracteres de la palabra es un número impar, entonces la palabra tiene 0 permutaciones palíndromas (puede haber a lo más un carácter un numero impar de veces, si pasa esto, se toma una de las repeticiones y se pone al medio de la palabra al formar la permutación palíndroma).

Luego, siendo n es el numero de caracteres de la palabra, se deben contar la cantidad de permutaciones de n/2 si n es par, (n-1)/2 si n es impar. Esto es porque la mitad de una palabra palíndroma es el reflejo de la otra mitad, por lo que solo se deben contar las permutaciones de una de las mitades, ya que si se cuentan las permutaciones de n entonces se estarían contando casos repetidos, por lo tanto el número de permutaciones, contando caracteres repetidos n_t seria igual a $piso(n/2)!$.

Ahora, el número de formas de contar las permutaciones repetidas es, según la regla del producto, la multiplicación de todas las formas de permutar los caracteres repetidos, por lo que el número de permutaciones repetidas $n_r = \Pi(\frac{rep(n_i)}{2})!$, donde $rep(n_i)$ es el numero de repeticiones del carácter n_i, y esta dividido por 2 ya que se están contando las permutaciones de la mitad de los caracteres de n.

Finalmente, por regla del producto, el número total de permutaciones (contando las repetidas), n_t, sería igual a la multiplicación entre el número de permutaciones que no se repiten n_p y el numero de permutaciones repetidas n_r, por lo tanto, el numero de permutaciones sin repeticiones $n_p = \frac{n_t}{n_r}$.

1.2. Código

El lenguaje utilizado es Python 2.7, código adjunto en sección Anexos

El código desarrollado hace lo siguiente:
Parte guardando los caracteres diferentes de la palabra en una lista letters, y el número de repeticiones de cada caracter en el diccionario reps, donde la llave es el caracter y el número de repeticiones es el valor. Luego se cuenta si hay más de un caracter repetido impar veces, si hay 2 o mas entonces retorna 0. Se continúa calculando el numerador de la fórmula obtenida en la parte teórica, donde fac(x) es el factorial de x, y después calcula el denominador, que parte siendo 1 y luego se va multiplicando por el factorial de las repeticiones de cada caracter, dividido por 2. Finalmente se retorna la división entre el numerador y denominador calculados.

2. P2

2.1. a

Supongamos que el jugador 1 escoge una secuencia binaria wd, donde w es una secuencia binaria y d es un dígito.
Lo que el jugador 2 querría elegir para ganar es la secuencia w, ya que así se aseguraría de ganar antes del primero, pero las secuencias deben ser del mismo largo, por lo que elige sw, con s un dígito, por lo que para ganar necesita que aparezca el dígito s antes que w, mientras que para que el primer jugador gane debe salir al principio wd, o que salga el contrario de s, después w y después d, por lo que el segundo jugador tiene más posibilidades de ganar.

2.2. b

Como se tiene una secuencia s de largo 3, lo que se debe retornar es una nueva secuencia donde el primer elemento es la negación del segundo elemento de s y el segundo y tercer elementos son el primero y segundo de s respectivamente.

El lenguaje utilizado es Python 2.7, código adjunto en sección Anexos

El código desarrollado hace lo siguiente:

Se crea la secuencia calculando la negación del segundo elemento de s, y se le agrega los primeros 2 elementos de s, y luego se retorna la secuencia creada.

El calculo de la negación es el siguiente: si el caracter recibido es 's', entonces se retorna 'c', y si se recibe 'c' se retorna 's'.

3. Anexos

3.1. P1

```python
#!/usr/bin/env python

def fac(n):
    if n==0:
        return 1
    else:
        return n*fac(n-1)

def count_letters_and_reps(s):
    letters = []
    reps = {}
    for st in s:
        if not st in letters:
            letters.append(st)

    for let in letters:
        reps[let] = s.count(let)

    return letters, reps

def p1(s):
    l = len(s)

    letters, reps = count_letters_and_reps(s)

    delete = []
    aux = 1
    for let in letters:
        if reps[let]%2!=0:
            aux -= 1

    if aux<0: return 0
    num = fac(int(l/2))
    den = 1
    for let in letters:
        den *= fac(int(reps[let]/2))
    return num/den


if __name__ == '__main__':
    print "--------------------------------------------------------------------------\n"
```

```
    s = raw_input("ingrese input: ")
    print '\n' + str(p1(s))

    #print p1("aabbccaaddc")
    print "\n-------------------------------------------------------------------------------"
```

3.2. P2

```
#!/usr/bin/env python

def neg(s):
    return 's' if (s == 'c') else 'c'

# If the first player choice is s[0]-s[1]-s[2]
# the second player must choose (not-s[1])-s[0]-s[1]
def p2(s):
    return neg(s[1]) + s[0] + s[1]


if __name__ == '__main__':
    print "-------------------------------------------------------------------------------\n"
    options = ["c","s"]
    s = raw_input("ingrese input (ej: csc): ")

    if len(s) != 3:
            print "\ninput incorrecto, debe ser secuencia de largo 3"
    else:
            aux = True
            for st in s:
                    if not st in options:
                            aux = False
            if not aux:
                    print "\ninput incorrecto, la secuencia solo debe contener caracteres 's' o 'c'"
            else:
                    print '\n' + p2(s)


    print "\n-------------------------------------------------------------------------------"
```