INFORME DEL TALLER 4

MAYRA ALEJANDRA SANCHEZ SALINAS (2040506)

FUNDAMENTOS DE PROGRAMACIÓN FUNCIONAL Y CONCURRENTE

JUAN FRANCISCO DIAZ FRIAS

UNIVERSIDAD DEL VALLE
SANTIAGO DE CALI, VALLE DEL CAUCA
2022

1. **Anagramas**

- **Representación**

```
type Palabra = String
```

Una palabra será una cadena sin espacios en blanco ni signos de puntuación, ni acentos, formada con las letras del alfabeto en minúsculas

```
type Frase = List[Palabra]
```

Representació de que una frase contiene palabras

Una lista de ocurrencias de una frase o de una palabra, es una lista de parejas (car, num) ordenada por car, donde num es un entero positivo que indica cuántas veces aparece el carácter car en la frase o en la palabra

```
type Ocurrencias = List [(Char, Int)]
```

```
val diccionario: List[Palabra]
```

El diccionario de las palabras con respecto a las cuales se harán los anagramas, se representará por medio de una lista de palabras

```
List ["moco", "casa", "como", "lapiz", "sos", "ocasos", "martillo", "yo", "jamon", "pensar", "mayra", "pensar", "jugar", "trabajar"]
```

Lista de palabras para tenerlas en cuenta en el diccionario: "moco", "casa", "como", "lapiz", "sos", "ocasos", "martillo", "yo", "jamon", "pensar", "mayra", "pensar", "jugar", "trabajar"

- **Calculando listas de ocurrencias**

```
def lOcPal(p: Palabra): Ocurrencias = {
  val asociacion = (p.toLowerCase groupBy identity) map{case (w,ws)=>(w,ws.length)}
  (SortedMap[Char,Int]() ++ asociacion).toList
}
```

La función lOcpal lo que hace es recibir una palabra y devolver una lista de ocurrencia asociada a esas palabras

```
def locFrase (f: Frase): Ocurrencias = { lOcPal(f.mkString)}
```

Lo mismo que la función anterior, lOcFrase recibe una frase y devuelve las ocurrencias de la frase en una lista

- **Calculando los anagramas de una palabra**

```
lazy val diccionarioPorOcurrencias: Map [Ocurrencias, List[Palabra]] ={
diccionario groupBy (palabra => lOcPal(palabra))
}
```

Lo que hace esta función es agrupar la lista de palabras y sus ocurrencias y agregarla al diccionario

```
def anagramasDePalabras (pal: Palabra): List[Palabra]={
  diccionarioPorOcurrencias(lOcPal(pal))
}
```

Lo que hace esta función es recibir una palabra y devolver la lista de sus anagramas usando el valor de diccionarioPorOcurrencias

- **Calculando los subconjuntos de un conjunto con repeticiones**

```
def combinaciones (lOcurrencias: Ocurrencias): List[Ocurrencias] = {
  (lOcurrencias foldRight List[Ocurrencias](Nil)) { case ((a, b), c) => {
    c ++ (for {combinar <- c; n <- 1 ≤ to ≤ b} yield (a, n) :: combinar)
  }
  }
}
```

La función combinaciones recibe una lista de ocurrencias, y devuelve la lista de todas las sublistas de ocurrencias de la lista original (incluyendo la sublista sin caracteres)

- **Calculando los anagramas de una frase**

```
def complemento (lOc: Ocurrencias , slOc: Ocurrencias): Ocurrencias = {
  def complemento1(lOc1: Ocurrencias, slOc2: Ocurrencias): Ocurrencias = slOc2 match {
    case Nil => lOc1
    case slOc :: slOcss =>
      val (lOcChar, lOcInt) = lOc1.unzip
      val (slOcChar, slOcInt) = slOc2.unzip
      val index = lOcChar.indexOf(slOcChar.head)
      val lOcNew: List[(Char, Int)] = lOcChar.zip(lOcInt.updated(index, lOcInt(index) - slOcInt.head))
      complemento1(lOcNew, slOcss)
  }
  complemento1(lOc, slOc) flatMap reemplazar
}

def reemplazar (elem: (Char, Int)): List[(Char, Int)] = elem match {
  case (char, 0) => Nil
  case (char, int) => List((char, int))}
```

Esta función complemento lo que hace es restar la lista de ocurrencias `slOc` de la lista de ocurrencias `lOc`.

La condición previa es que la lista de ocurrencias `slOc` sea un subconjunto de la lista de ocurrencias `lOc` -- cualquier carácter que aparezca en `slOc` debe aparece

en `lOc`, y su frecuencia en `slOc` debe ser menor o igual que su frecuencia en `lOc`.

El valor resultante es una ocurrencia, lo que significa que está ordenado y no tiene entradas cero.

```
def anagramasDeFrase(frase: Frase): List[Frase] = {
  def iteracionFor(lOcurrencias: Ocurrencias): List[Frase] = {
    if (lOcurrencias.isEmpty) List(Nil)
    else for {
      combinacion <- combinaciones( lOcurrencias )
      palabra <- diccionarioPorOcurrencias getOrElse (combinacion, Nil)
      frase <- iteracionFor( complemento(lOcurrencias,lOcPal(palabra)) )
      if !combinacion.isEmpty
    } yield palabra :: frase
  }
  iteracionFor( locFrase(frase) )
}
```

la función anagramasDeFrase, que recibe una frase, y devuelve la lista de frases que son anagramas de la frase original

- **INFORME DE USO DE COLECCIONES Y EXPRESIONES FOR**

FUNCION	¿SE USÓ TÉCNICA?
lOcPal	Si
lOcFrase	Si
anagramaDePalabras	Si
combinaciones	Si
complemento	Si
anagramaDeFrase	Si

Las colecciones permiten que no sea necesario eliminar explícitamente.y las expresiones for son muy útiles porque permiten realizar consultas típicas de bases de datos, además, construyen una lista con los resultados de cada iteración.

- **INFORME DE CORRECCIÓN**
  ARGUMENTACIÓN SOBRE LA CORRECCIÓN

FUNCION	ARGUMENTACIÓN
lOcPal	Sea p un string <-> se analiza la frecuencia de cada letra de la palabra
lOcFrase	Sea f una frase que contiene palabras de tipo string <-> se analiza la frecuencia de cada letra de las palabras de la frase
anagramaDePalabras	Sea p una palabra se analiza la ocurrencia de ella y buscar si en el diccionario existe una palabra que cumpla con la misma ocurrencia de cada letra
combinaciones	Sea x cantidad de lOcurrencias que se agreguen a la lista se combinan entre ellas
complemento	Sea lOc una la ocurrencia de una palabra y slOc una parte de la misma ocurrencia de esa palabra se devuelve lo que hace falta en slOc para llegar a cumplir lOc
anagramaDeFrase	Sea f una frase se analiza la ocurrencia de ella y buscar si en el diccionario existen palabras que conformen la misma frase y organizarla de diferentes formas

CASOS DE PRUEBA

```
/**
 * Palabras
 */
val palabra =lOcPal("japones")

combinaciones(palabra)
anagramasDePalabras( pal= "japones")
```

Para la primera palabra se escogió japonés

- val palabra: Anagramas.Ocurrencias = List((a,1), (e,1), (j,1), (n,1), (o,1), (p,1), (s,1))

  Como se puede observar en la palabra japones la palabra tiene una sola letra diferente por lo que la frecuencia en cada letra es 1

- val res0: List[Anagramas.Ocurrencias] = List(List(), List((s,1)), List((p,1)), List((p,1), (s,1)), List((o,1)), List((o,1), (s,1)), List((o,1), (p,1)), List((o,1), (p,1), (s,1)), List((n,1)), List((n,1), (s,1)), List((n,1), (p,1)), List((n,1), (p,1), (s,1)), List((n,1), (o,1)), List((n,1), (o,1), (s,1)), List((n,1), (o,1), (p,1)), List((n,1), (o,1), (p,1), (s,1)), List((j,1)), List((j,1), (s,1)), List((j,1), (p,1)), List((j,1), (p,1), (s,1)), List((j,1), (o,1)), List((j,1), (o,1), (s,1)), List((j,1), (o,1), (p,1)), List((j,1), (o,1), (p,1), (s,1)), List((j,1), (n,1)), List((j,1), (n,1), (s,1)), List((j,1), (n,1), (p,1)), List((j,1), (n,1), (p,1), (s,1)), List((j,1), (n,1), (o,1)), List((j,1), (n,1), (o,1), (s,1)), List((j,1), (n,1), (o,1), (p,1)), List((j,1), (n,1), (o,1), (p,1), (s,1))...

- val res1: List[Anagramas.Palabra] = List(japones, esponja)

  El anagrama de japonés es esponja

```
val palabra2 =lOcPal("delira")

combinaciones(palabra2)
anagramasDePalabras( pal= "delira")
```

La siguiente palabra fue delira

- val palabra2: Anagramas.Ocurrencias = List((a,1), (d,1), (e,1), (i,1), (l,1), (r,1))

Como se puede observar en la palabra delira la palabra tiene una sola letra diferente por lo que la frecuencia en cada letra es 1

- val res2: List[Anagramas.Ocurrencias] = List(List(), List((r,1)), List((l,1)), List((l,1), (r,1)), List((i,1)), List((i,1), (r,1)), List((i,1), (l,1)), List((i,1), (l,1), (r,1)), List((e,1)), List((e,1), (r,1)), List((e,1), (l,1)), List((e,1), (l,1), (r,1)), List((e,1), (i,1)), List((e,1), (i,1), (r,1)), List((e,1), (i,1), (l,1)), List((e,1), (i,1), (l,1), (r,1)), List((d,1)), List((d,1), (r,1)), List((d,1), (l,1)), List((d,1), (l,1), (r,1)), List((d,1), (i,1)), List((d,1), (i,1), (r,1)), List((d,1), (i,1), (l,1)), List((d,1), (i,1), (l,1), (r,1)), List((d,1), (e,1)), List((d,1), (e,1), (r,1)), List((d,1), (e,1), (l,1)), List((d,1), (e,1), (l,1), (r,1)), List((d,1), (e,1), (i,1)), List((d,1), (e,1), (i,1), (r,1)), List((d,1), (e,1), (i,1), (l,1)), List((d,1), (e,1), (i,1), (l,1), (r,1))...

- val res3: List[Anagramas.Palabra] = List(delira, lidera)

  El anagrama de delira es lidera

```
val palabra3 =lOcPal("matar")

combinaciones(palabra3)
anagramasDePalabras( pal= "matar")
```

La tercer palabra que escogí fue matar

- val palabra3: Anagramas.Ocurrencias = List((a,2), (m,1), (r,1), (t,1))

  Como se puede observar en la palabra matar la palabra tiene una sola letra que se repite más de 1 vez y es la letra a y en la respuesta la frecuencia de cada letra es correcta o sea 2 y 1 en los sobrantes

- val res4: List[Anagramas.Ocurrencias] = List(List(), List((t,1)), List((r,1)), List((r,1), (t,1)), List((m,1)), List((m,1), (t,1)), List((m,1), (r,1)), List((m,1), (r,1), (t,1)), List((a,1)), List((a,2)), List((a,1), (t,1)), List((a,2), (t,1)), List((a,1), (r,1)), List((a,2), (r,1)), List((a,1), (r,1), (t,1)), List((a,2), (r,1), (t,1)), List((a,1), (m,1)), List((a,2), (m,1)), List((a,1), (m,1), (t,1)), List((a,2), (m,1), (t,1)), List((a,1), (m,1), (r,1)), List((a,2), (m,1), (r,1)), List((a,1), (m,1), (r,1), (t,1)), List((a,2), (m,1), (r,1), (t,1)))

- val res5: List[Anagramas.Palabra] = List(matar, marta)

  El anagrama de matar es marta

```
val palabra4 =lOcPal("sopa")

combinaciones(palabra4)
anagramasDePalabras( pal= "sopa")
```

La siguiente palabra es sopa

- val palabra4: Anagramas.Ocurrencias = List((a,1), (o,1), (p,1), (s,1))

  Como se puede observar en la palabra sopa la palabra tiene una sola letra diferente por lo que la frecuencia en cada letra es 1

- val res6: List[Anagramas.Ocurrencias] = List(List(), List((s,1)), List((p,1)), List((p,1), (s,1)), List((o,1)), List((o,1), (s,1)), List((o,1), (p,1)), List((o,1), (p,1), (s,1)), List((a,1)), List((a,1), (s,1)), List((a,1), (p,1)), List((a,1), (p,1), (s,1)), List((a,1), (o,1)), List((a,1), (o,1), (s,1)), List((a,1), (o,1), (p,1)), List((a,1), (o,1), (p,1), (s,1)))

- val res7: List[Anagramas.Palabra] = List(sopa, paso)

  El anagrama de sopa es paso

```
val palabra5 =lOcPal("agranda")

combinaciones(palabra5)
anagramasDePalabras( pal= "agranda")
```

Por último la palabra es agranda

- val palabra5: Anagramas.Ocurrencias = List((a,3), (d,1), (g,1), (n,1), (r,1))

  Como se puede observar en la palabra agranda la palabra tiene una sola letra que se repite más de 1 vez y es la letra a y en la respuesta la frecuencia de cada letra es correcta

- val res8: List[Anagramas.Ocurrencias] = List(List(), List((r,1)), List((n,1)), List((n,1), (r,1)), List((g,1)), List((g,1), (r,1)), List((g,1), (n,1)), List((g,1), (n,1), (r,1)), List((d,1)), List((d,1), (r,1)), List((d,1), (n,1)), List((d,1), (n,1), (r,1)), List((d,1), (g,1)), List((d,1), (g,1), (r,1)), List((d,1),

(g,1), (n,1)), List((d,1), (g,1), (n,1), (r,1)), List((a,1)), List((a,2)), List((a,3)), List((a,1), (r,1)), List((a,2), (r,1)), List((a,3), (r,1)), List((a,1), (n,1)), List((a,2), (n,1)), List((a,3), (n,1)), List((a,1), (n,1), (r,1)), List((a,2), (n,1), (r,1)), List((a,3), (n,1), (r,1)), List((a,1), (g,1)), List((a,2), (g,1)), List((a,3), (g,1)), List((a,1), (g,1), (r,1)), List((a,2), (g,1), (r,1)), List((a,3), (g,1), (r,1)), List((a,1), (g,1), (n,1)), List((a,2), (g,1), (n,...

- val res9: List[Anagramas.Palabra] = List(agranda, granada)

  El anagrama de agranda es granada

```
//complementoS
complemento(lOcPal("japones"), lOcPal("es"))
complemento(lOcPal("delira"), lOcPal("ira"))
complemento(lOcPal("matar"), lOcPal("tar"))
complemento(lOcPal("sopa"), lOcPal("a"))
complemento(lOcPal("agranda"), lOcPal("da"))
```

- val res10: Anagramas.Ocurrencias = List((a,1), (j,1), (n,1), (o,1), (p,1))

  El complemento de “es” son a, j, n, o y la p
- val res11: Anagramas.Ocurrencias = List((d,1), (e,1), (l,1))

  El complemento de “ira” son d, e y l

- val res12: Anagramas.Ocurrencias = List((a,1), (m,1))

  El complemento de “tar” son a y m

- val res13: Anagramas.Ocurrencias = List((o,1), (p,1), (s,1))

  El complemento de “a” son o, p y s

- val res14: Anagramas.Ocurrencias = List((a,2), (g,1), (n,1), (r,1))

  El complemento de “da” son a, g, n y r

Ahora para las frase se escogieron las siguientes:

```
/**
 * Frases
 */

val frase= List ( "cosas","como","yo")

locFrase(frase)
anagramasDeFrase(frase)
combinaciones(locFrase(frase))
```

La frase que se escogió fue la siguiente: cosas, como, yo

- val res15: Anagramas.Ocurrencias = List((a,1), (c,2), (m,1), (o,4), (s,2), (y,1))

  Las frecuencias en estas frases aumentan más que en la frecuencia de palabras por ejemplo en esta frase la letra c y s se repiten 2 veces y la o se repite 4 veces

- Los posibles anagramas de la frese son los siguientes: val res16: List[Anagramas.Frase] = List(List(yo, como, cosas), List(yo, cosas, como), List(como, yo, cosas), List(como, cosas, yo), List(cosas, yo, como), List(cosas, como, yo))

- val res17: List[Anagramas.Ocurrencias] = List(List(), List((y,1)), List((s,1)), List((s,2)), List((s,1), (y,1)), List((s,2), (y,1)), List((o,1)), List((o,2)), List((o,3)), List((o,4)), List((o,1), (y,1)), List((o,2), (y,1)), List((o,3), (y,1)), List((o,4), (y,1)), List((o,1), (s,1)), List((o,2), (s,1)), List((o,3), (s,1)), List((o,4), (s,1)), List((o,1), (s,2)), List((o,2), (s,2)), List((o,3), (s,2)), List((o,4), (s,2)), List((o,1), (s,1), (y,1)), List((o,2), (s,1), (y,1)), List((o,3), (s,1), (y,1)), List((o,4), (s,1), (y,1)), List((o,1), (s,2), (y,1)), List((o,2), (s,2), (y,1)), List((o,3), (s,2), (y,1)), List((o,4), (s,2), (y,1)), List((m,1)), List((m,1), (y,1)), List((m,1), (s,1)), List((m,1), (s,2)), List((m,1), (s,1), (y,1)), List((m,1), (s,2), (y,1)), List((m,1), (o,1)), List((m,1...

```
val frase2= List ( "yo","amo","jugar")

locFrase(frase2)
anagramasDeFrase(frase2)
combinaciones(locFrase(frase2))
```

La frase que se escogió fue la siguiente: yo, amo, jugar

- val res18: Anagramas.Ocurrencias = List((a,2), (g,1), (j,1), (m,1), (o,2), (r,1), (u,1), (y,1))

  En esta frase la letra a y la letra o son las únicas que se repiten 2 veces en toda la frase

- Las siguientes frases son los anagramas que se pueden formas de acuerdo a yo amo jugar val res19: List[Anagramas.Frase] = List(List(yo, amo, jugar), List(yo, jugar, amo), List(amo, yo, jugar), List(amo, jugar, yo), List(jugar, yo, amo), List(jugar, amo, yo))

- val res20: List[Anagramas.Ocurrencias] = List(List(), List((y,1)), List((u,1)), List((u,1), (y,1)), List((r,1)), List((r,1), (y,1)), List((r,1), (u,1)), List((r,1), (u,1), (y,1)), List((o,1)), List((o,2)), List((o,1), (y,1)), List((o,2), (y,1)), List((o,1), (u,1)), List((o,2), (u,1)), List((o,1), (u,1), (y,1)), List((o,2), (u,1), (y,1)), List((o,1), (r,1)), List((o,2), (r,1)), List((o,1), (r,1), (y,1)), List((o,2), (r,1), (y,1)), List((o,1), (r,1), (u,1)), List((o,2), (r,1), (u,1)), List((o,1), (r,1), (u,1), (y,1)), List((o,2), (r,1), (u,1), (y,1)), List((m,1)), List((m,1), (y,1)), List((m,1), (u,1)), List((m,1), (u,1), (y,1)), List((m,1), (r,1)), List((m,1), (r,1), (y,1)), List((m,1), (r,1), (u,1)), List((m,1), (r,1), (u,1), (y,1)), List((m,1), (o,1)), List((m,1), (o,2)), List((m,1), (...

```
val frase3= List ( "me","gusta","salir")

locFrase(frase3)
anagramasDeFrase(frase3)
combinaciones(locFrase(frase3))
```

La frase que se escogió fue la siguiente: me, gusta, salir

- val res21: Anagramas.Ocurrencias = List((a,2), (e,1), (g,1), (i,1), (l,1), (m,1), (r,1), (s,2), (t,1), (u,1))

  La frecuencia de la letras de esta frase son únicamente la y la u que se repite 2 veces, el resto de las letras se repite 1 vez

- Estas son las frase formadas del anagrama de la frase: val res22: List[Anagramas.Frase] = List(List(me, salir, gusta), List(me, gusta, salir), List(salir, me, gusta), List(salir, gusta, me), List(gusta, me, salir), List(gusta, salir, me))

- val res23: List[Anagramas.Ocurrencias] = List(List(), List((u,1)), List((t,1)), List((t,1), (u,1)), List((s,1)), List((s,2)), List((s,1), (u,1)), List((s,2), (u,1)), List((s,1), (t,1)), List((s,2), (t,1)), List((s,1), (t,1), (u,1)), List((s,2), (t,1), (u,1)), List((r,1)), List((r,1), (u,1)), List((r,1), (t,1)), List((r,1), (t,1), (u,1)), List((r,1), (s,1)), List((r,1), (s,2)), List((r,1), (s,1), (u,1)), List((r,1), (s,2), (u,1)), List((r,1), (s,1), (t,1)), List((r,1), (s,2), (t,1)), List((r,1), (s,1), (t,1), (u,1)), List((r,1), (s,2), (t,1), (u,1)), List((m,1)), List((m,1), (u,1)), List((m,1), (t,1)), List((m,1), (t,1), (u,1)), List((m,1), (s,1)), List((m,1), (s,2)), List((m,1), (s,1), (u,1)), List((m,1), (s,2), (u,1)), List((m,1), (s,1), (t,1)), List((m,1), (s,2), (t,1)), List((m,1), (...

```
val frase4= List ( "amo","la","programacion")

locFrase(frase4)
anagramasDeFrase(frase4)
combinaciones(locFrase(frase4))
```

La frase que se escogió fue la siguiente: amo, la, programación

- val res24: Anagramas.Ocurrencias = List((a,4), (c,1), (g,1), (i,1), (l,1), (m,2), (n,1), (o,3), (p,1), (r,2))

  La letra en esta frase con más frecuencia es la a y despues con 3 viene la o y finalmente la m y la r con 2 veces en la frase

- Estos son los anagramas según la frase: val res25: List[Anagramas.Frase] = List(List(amo, la, programacion), List(amo, programacion, la), List(la, amo, programacion), List(la, programacion, amo), List(programacion, amo, la), List(programacion, la, amo))

- val res26: List[Anagramas.Ocurrencias] = List(List(), List((r,1)), List((r,2)), List((p,1)), List((p,1), (r,1)), List((p,1), (r,2)), List((o,1)), List((o,2)), List((o,3)), List((o,1), (r,1)), List((o,2), (r,1)), List((o,3), (r,1)), List((o,1), (r,2)), List((o,2), (r,2)), List((o,3), (r,2)), List((o,1), (p,1)), List((o,2), (p,1)), List((o,3), (p,1)), List((o,1), (p,1), (r,1)), List((o,2), (p,1), (r,1)), List((o,3), (p,1), (r,1)), List((o,1), (p,1), (r,2)), List((o,2), (p,1), (r,2)), List((o,3), (p,1), (r,2)), List((n,1)), List((n,1), (r,1)), List((n,1), (r,2)), List((n,1), (p,1)), List((n,1), (p,1), (r,1)), List((n,1), (p,1), (r,2)), List((n,1), (o,1)), List((n,1), (o,2)), List((n,1), (o,3)), List((n,1), (o,1), (r,1)), List((n,1), (o,2), (r,1)), List((n,1), (o,3), (r,1)), List((n,1), (o,...

```
val frase5= List ( "quiero","viajar","mucho")

locFrase(frase5)
anagramasDeFrase(frase5)
combinaciones(locFrase(frase5))
```

La frase que se escogió fue la siguiente: quiero, viajar mucho

- val res27: Anagramas.Ocurrencias = List((a,2), (c,1), (e,1), (h,1), (i,2), (j,1), (m,1), (o,2), (q,1), (r,2), (u,2), (v,1))

  La frecuencia de cada letra en este caso es hasta máximo 2 en: a, i, o, r y u

- Estos son los anagramas según la frase: val res28: List[Anagramas.Frase] = List(List(quiero, mucho, viajar), List(quiero, viajar, mucho), List(mucho, quiero, viajar), List(mucho, viajar, quiero), List(viajar, quiero, mucho), List(viajar, mucho, quiero))

- val res29: List[Anagramas.Ocurrencias] = List(List(), List((v,1)), List((u,1)), List((u,2)), List((u,1), (v,1)), List((u,2), (v,1)), List((r,1)), List((r,2)), List((r,1), (v,1)), List((r,2), (v,1)), List((r,1), (u,1)), List((r,2), (u,1)), List((r,1), (u,2)), List((r,2), (u,2)), List((r,1), (u,1), (v,1)), List((r,2), (u,1), (v,1)), List((r,1), (u,2), (v,1)), List((r,2), (u,2), (v,1)), List((q,1)), List((q,1), (v,1)), List((q,1), (u,1)), List((q,1), (u,2)), List((q,1), (u,1), (v,1)), List((q,1), (u,2), (v,1)), List((q,1), (r,1)), List((q,1), (r,2)), List((q,1), (r,1), (v,1)), List((q,1), (r,2), (v,1)), List((q,1), (r,1), (u,1)), List((q,1), (r,2), (u,1)), List((q,1), (r,1), (u,2)), List((q,1), (r,2), (u,2)), List((q,1), (r,1), (u,1), (v,1)), List((q,1), (r,2), (u,1), (v,1)), List((q,1), (...