Andres Felipe Chaparro - 2266252

Informe

INTERSECCION

CONJUNTOS DIFUSOS:

Descripción de ejercicio: Esta parte del ejercicio consiste en la implementacion de la funcion de interseccion de los conjuntos difusos, tomando dos conjuntos como base y generando un nuevo conjunto que contiene los elementos con su minimo grado de pertenencia. Utilizando un enfoque algoritmico para manejar listas que contienen pares de elementos ademas de sus grados de depertenencia y resolviendo el problema mediante recursion y transformacion iterativa en Scala.

Interseccion de Conjuntos Difusos

Descripción de la Función:

intersectSets: La funcion toma dos conjuntos difusos que estan representados como listas de tuplas(elemento, grado de pertenencia) y asi calculaa la interseccion. Se basa en seleccionar el menor grado de pernetencia para los elementos en comun entre ambos conjuntos.

Ejemplos y Proceso de Ejecución:

Ejemplo intersectSets (Recursiva):

Código de Ejemplo:

```
val setA1 = Map(1 -> 0.2, 2 -> 0.5, 3 -> 0.6)
val setB1 = Map(3 -> 0.6, 4 -> 0.7, 1 -> 0.8)
val result1 = SetOperations.intersect(setA1, setB1)
```

Proceso de Ejecución:

Debido a que el conjunto A contiene los elementos 1, 2, 3 y el conjunto B contiene 3,4 y 1 el resultado sera la interseccion de los elementos en comun con el menor grado de pertenencia.

```
Interseccion 1: Map(1 -> 0.2, 3 -> 0.6)
```

Pila de llamadas:

Cuando la funcion recursiva inicia, hace una comparacion de los primeros elementos de las listas y va avanzando mientras almacena el minimo grado de pertenencia cuando hay elementos en comun.

2. Informe de Corrección

1. Argumentación sobre la Corrección

intersectSets (Recursiva):

El principio de esta funcion recursiva es el comparar cada uno de los elementos de los conjuntos y aplicar la operacion de interseccion definida con el minimo grado de pertenencia. La correccion de la funcion se argumenta mostrando que, para los conjuntos recursivamente definidos, el algoritmo compara de una forma correcta los grados de pertenencia y aplica la logica esperada. En el caso base, cuando un conjunto es vacio, la funcion retorna un conjunto vacio, lo que es coherente con el concepto de interseccion.

2. Casos de Prueba:

Se realizaron diversos tipos de casos de prueba bajo diferentes configuraciones de conjuntos difusos para asegurar que la funcion devuelve los resultados esperados:

Prueba 2:

```
val setA2 = Map.empty[Int, Double]
val setB2 = Map(1 -> 0.3, 2 -> 0.4)
val result2 = SetOperations.intersect(setA2, setB2)
println(s"Interseccion 2 (conjunto A vacio): $result2")
```

Resultado:

```
Interseccion 2 (conjunto A vacio): Map()
```

Prueba 3:

```
val setA3 = Map(1 -> 0.5, 2 -> 0.7)
val setB3 = Map(3 -> 0.6, 4 -> 0.8)
val result3 = SetOperations.intersect(setA3, setB3)
println(s"Interseccion 3 (sin elementos comunes): $result3")
```

Resultado:

```
Interseccion 3 (sin elementos comunes): Map()
```

Prueba 4:

```
val setA4 = Map(1 -> 0.0, 2 -> 0.8, 3 -> 0.0)
val setB4 = Map(2 -> 0.5, 3 -> 0.0, 4 -> 0.9)
val result4 = SetOperations.intersect(setA4, setB4)
println(s"Interseccion 4 (grados de pertenencia cero): $result4")
```

Resultado:

```
Interseccion 4 (grados de pertenencia cero): Map(2 -> 0.5, 3 -> 0.0)
```

Prueba 5:

```
val setA5 = Map(1 -> 0.5, 2 -> 0.7, 3 -> 0.9)
val result5 = SetOperations.intersect(setA5, setA5)
println(s"Interseccion 5 (conjunto consigo mismo): $result5")
```

Resultado:

```
Interseccion 5 (conjunto consigo mismo): Map(1 -> 0.5, 2 -> 0.7, 3 -> 0.9)
```

Prueba 6:

```
val setA6 = Map(1 -> 0.5, 2 -> 0.1, 3 -> 0.6)
val setB6 = Map(3 -> 0.3, 2 -> 0.7, 1 -> 0.9)
val result6 = SetOperations.intersect(setA6, setB6)
println(s"Interseccion 6: $result6")
```

Resultado:

```
Interseccion 6: Map(1 -> 0.5, 2 -> 0.1, 3 -> 0.3)
```

3. Conclusiones

- **Reflexión sobre el Proceso**: Los procesos de implementacion y prueba de la funcion de interseccion de conjuntos difusos mostro, tanto en terminos de recursion como en su rendimiento que la solucion es la adecuada. Ademas, los casos de prueba mostraron consistencia de resultados, mostrandonos que la funcion si se comporta de forma predecible.
- **Desafíos y Aprendizajes**: Se tuvieron multiples desafios al momento de desarrollar estos problemas pero uno de los desafios mas importantes era asegurarnos que los grados de pertenencia a la hora de hacer las comparaciones se manipularan de una forma adecuada y precisa para obtener el resultado correcto. La implementacion de una solucion recursiva fue clave para poder manejar conjuntos de cualquier tamaño
- **Aplicaciones y Relevancia**: El concepto de interseccion tiene una gran relevancia en sistemas que necesitan tomar desiciones basadas en conjuntos de valores imprecisos, como en la inteligencia artificial o en problemas de toma de decisiones. La implementacion realizada es muy versatil y puede extenderse facilmente a otros tipos de operaciones sobre los conjuntos difusos.