Instituto Tecnológico de Tijuana

Nombre de Facultad

Ingeniería Informática

Proyecto / Tarea / Practica:

Práctica 4

Materia:

Datos Masivos

Facilitador:

Jose Christian

Alumnos:

Erik Saul Rivera Reyes

Brayan Baltazar Moreno

Fecha:

Tijuana Baja California a 09 de 03 2022

Code

```
//PRACTICA 4
C:\Spark\spark-3.2.1-bin-hadoop3.2\bin\spark-shell

//Fibonacci por medio de recursion basica

def fib(n: Int): BigInt = n match {
  case 0 => 0
  case 1 => 1
  case _ => fib(n-2) + fib(n-1)
}

(0 to 10).foreach(n => print(fib(n) + " "))
fib(50)

//Fibnacci usando el metodo "tail"

def fibTR(num: Int): BigInt = {
  @scala.annotation.tailrec
  def fibFcn(n: Int, acc1: BigInt, acc2: BigInt): BigInt = n match {
    case 0 => acc1
    case 1 => acc2
    case _ => fibFcn(n - 1, acc2, acc1 + acc2)
  }

  fibFcn(num, 0, 1)
}
fibTR(90)

//Fibonacci usando un mapa mutable
def memoize[K, V](f: K => V): K => V = {
  val cache = scala.collection.mutable.Map.empty[K, V]
  k => cache.getOrElseUpdate(k, f(k))
}
val fibM: Int => BigInt = memoize(n => n match {
  case 0 => 0
  case 1 => 1
  case _ => fibM(n-2) + fibM(n-1)
})
fibM(90)
```

1-Por este método se realiza una secuencia de Fibonacci de manera básica y directa usando lo que es la recursión

```
scala> def fib(n: Int): BigInt = n match {
     |   case 0 => 0
     |   case 1 => 1
     |   case _ => fib(n-2) + fib(n-1)
     | }
fib: (n: Int)BigInt

scala>

scala> (0 to 10).foreach(n => print(fib(n) + " "))
0 1 1 2 3 5 8 13 21 34 55
scala> fib(50)
```

2-Por este método se realiza una secuencia de Fibonacci pero con una herramienta llamada “tail” la cual nos despliega los valores de una forma diferente además de que los resultados se muestran mucho mas rápido que en la forma básica

```
scala> def fibTR(num: Int): BigInt = {
     |   @scala.annotation.tailrec
     |   def fibFcn(n: Int, acc1: BigInt, acc2: BigInt): BigInt = n match {
     |     case 0 => acc1
     |     case 1 => acc2
     |     case _ => fibFcn(n - 1, acc2, acc1 + acc2)
     |   }
     |
     |   fibFcn(num, 0, 1)
     | }
fibTR: (num: Int)BigInt

scala> fibTR(90)
res0: BigInt = 2880067194370816120

scala>
```

3-Por este método se realiza la secuencia Fibonacci pero con el uso de los mapas mutables los cuales nos permite ser mas efectivos a la hora de desplegar los valores asi de tener todas las ventajas que tienen los mapas mutables en general

```
scala> def memoize[K, V](f: K => V): K => V = {
     |   val cache = scala.collection.mutable.Map.empty[K, V]
     |   k => cache.getOrElseUpdate(k, f(k))
     | }
memoize: [K, V](f: K => V)K => V

scala> val fibM: Int => BigInt = memoize(n => n match {
     |   case 0 => 0
     |   case 1 => 1
     |   case _ => fibM(n-2) + fibM(n-1)
     | })
fibM: Int => BigInt = $Lambda$1774/0x0000000801426100@65018381

scala> fibM(90)
res1: BigInt = 2880067194370816120

scala> _
```