Lab1_scala

November 19, 2022

[2]:
```
val data = Array.range(1,30)
val rdd = sc.parallelize(data)
rdd.collect() //pobranie elementow ze zbioru
```

```
data = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,␣
 ↪20, 21, 22, 23, 24, 25, 26, 27, 28, 29)
rdd = ParallelCollectionRDD[1] at parallelize at <console>:31
```

[2]:
```
Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29)
```

[6]:
```
rdd.first()
rdd.take(3)
rdd.takeSample(true,10) //nadpisanie wynikow jesli wlozymy do jednej komorki
```

[6]:
```
Array(20, 21, 23, 2, 3, 6, 1, 3, 17, 16)
```

[7]:
```
val rdd2 = rdd.map( x => x*x ) //mapowanie na potegi
rdd2.collect()
```

```
rdd2 = MapPartitionsRDD[5] at map at <console>:27
```

[7]:
```
Array(1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289,
324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841)
```

[9]:
```
val rdd2 = rdd.map( x => List(x,x) ) //mapowanie na liste par
rdd2.collect()
```

```
rdd2 = MapPartitionsRDD[7] at map at <console>:29
```

[9]:
```
Array(List(1, 1), List(2, 2), List(3, 3), List(4, 4), List(5, 5), List(6, 6),
List(7, 7), List(8, 8), List(9, 9), List(10, 10), List(11, 11), List(12, 12),
List(13, 13), List(14, 14), List(15, 15), List(16, 16), List(17, 17), List(18,
18), List(19, 19), List(20, 20), List(21, 21), List(22, 22), List(23, 23),
List(24, 24), List(25, 25), List(26, 26), List(27, 27), List(28, 28), List(29,
29))
```

[10]:
```
val rdd2 = rdd.flatMap( x => List(x,x)) //flatmapa listy par
rdd2.collect()
```

```
rdd2 = MapPartitionsRDD[8] at flatMap at <console>:29
```

[10]:
```
Array(1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12,
12, 13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22,
22, 23, 23, 24, 24, 25, 25, 26, 26, 27, 27, 28, 28, 29, 29)
```

[11]:
```
val value = rdd.reduce( (t1,t2) => t1 + t2) //zwraca skalar - rekursywnie␣
↪powtarzana operacja, ktora doprowadzila do sumowania wszysktich elementow
```

```
value = 435
```

[11]:
```
435
```

[12]:
```
val rdda = sc.parallelize( List( "aa","bb","cc","dd","ee","ff","gg" ) )
val value = rdda.reduce( (t1, t2) => t1+t2)
```

```
rdda = ParallelCollectionRDD[9] at parallelize at <console>:27
value = ddaaccffggbbee
```

[12]:
```
ddaaccffggbbee
```

[13]:
```
rdd.count() //liczba elementow - proste
```

[13]:
```
29
```

[16]:
```
val data1 = Array.range(1,21)
val data2 = Array.range(19,25)
val rdd1 = sc.parallelize(data1)
val rdd2 = sc.parallelize(data2)
val rdd3 = rdd1.union(rdd2) //unia -> suma dwoch data setow
rdd3.collect()
```

```
data1 = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,␣
↪20)
data2 = Array(19, 20, 21, 22, 23, 24)
rdd1 = ParallelCollectionRDD[16] at parallelize at <console>:35
rdd2 = ParallelCollectionRDD[17] at parallelize at <console>:36
rdd3 = UnionRDD[18] at union at <console>:37
```

[16]:
```
Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 19,
20, 21, 22, 23, 24)
```

[17]:
```
val rdd4 = rdd3.distinct() //odfiltrowanie powtarzajacych sie elementow
rdd4.collect()
```

```
rdd4 = MapPartitionsRDD[21] at distinct at <console>:27
```

[17]:
```
Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24)
```

[18]:
```
rdd3.count() //rozmiar
```

[18]:
```
26
```

[19]:
```
rdd4.count() //rozmiar
```

[19]:
```
24
```

[20]:
```
val rdd5 = rdd1.intersection(rdd2) //
rdd5.collect()
```

```
rdd5 = MapPartitionsRDD[27] at intersection at <console>:28
```

[20]:
```
Array(19, 20)
```

[21]:
```
val rdda1 = sc.
 ↪parallelize(List("aa","bb","cc","dd","aa","cc","ee","ff","dd","dd","aa"))
val rdda2 = rdda1.map( k => (k,1)) //ile kluczy wystepuje w datasecie
rdda2.countByKey()
```

```
rdda1 = ParallelCollectionRDD[28] at parallelize at <console>:27
rdda2 = MapPartitionsRDD[29] at map at <console>:28
```

[21]:
```
Map(cc -> 2, aa -> 3, bb -> 1, ee -> 1, ff -> 1, dd -> 3)
```

[22]:
```
val rddr1 = sc.parallelize(List("aa","bb","cc","dd","ee","ff","gg","aa")).map(␣
 ↪k => (k,1))
val rddr2 = sc.parallelize(List("aa","cc","mm","rr","tt")).map( k => (k,1))
rddr1.join(rddr2).collect() //dotyczy krotek wystepujacych w obu zbiorach
```

```
rddr1 = MapPartitionsRDD[33] at map at <console>:27
rddr2 = MapPartitionsRDD[35] at map at <console>:28
```

```
[22]: Array((aa,(1,1)), (aa,(1,1)), (cc,(1,1)))
```

```
[23]: rddr1.leftOuterJoin(rddr2).collect()
```

```
[23]: Array((ee,(1,None)), (aa,(1,Some(1))), (aa,(1,Some(1))), (gg,(1,None)),
      (dd,(1,None)), (ff,(1,None)), (bb,(1,None)), (cc,(1,Some(1))))
```

```
[24]: val data = Seq(("Java", "20000"), ("Python", "100000"), ("Scala", "3000")) //
       ↪lokalna lista sekwencji
      val rddt = sc.parallelize(data) //wysylamy na klaster
      rddt.collect()
```

```
data = List((Java,20000), (Python,100000), (Scala,3000))
rddt = ParallelCollectionRDD[42] at parallelize at <console>:30
```

```
[24]: Array((Java,20000), (Python,100000), (Scala,3000))
```

```
[25]: val dfFromrddt = rddt.toDF()
      dfFromrddt.printSchema()
```

```
root
 |-- _1: string (nullable = true)
 |-- _2: string (nullable = true)


dfFromrddt = [_1: string, _2: string]
```

```
[25]: [_1: string, _2: string]
```

```
[26]: val dfFromrddt = rddt.toDF("language","users_count")
      dfFromrddt.printSchema()
```

```
root
 |-- language: string (nullable = true)
 |-- users_count: string (nullable = true)


dfFromrddt = [language: string, users_count: string]
```

```
[26]: [language: string, users_count: string]
```

```
[27]: val columns = Seq("language","users_count") //sekwencja z etykietami
      val dfFromrddt2 = spark.createDataFrame(rddt).toDF(columns:_*) //obiekt spark␣
       ↪(niezainicjalizowany - ok) | tworzony data frame
```

```
columns = List(language, users_count)
dfFromrddt2 = [language: string, users_count: string]
```

[27]:
```
[language: string, users_count: string]
```

[29]:
```
dfFromrddt2.show() //odpowiednik collect()
```

```
+--------+-----------+
|language|users_count|
+--------+-----------+
|    Java|      20000|
|  Python|     100000|
|   Scala|       3000|
+--------+-----------+
```

[30]:
```
val rdd_csv2 = sc.textFile("../Dane/dane.csv")
rdd_csv2.collect()
```

```
rdd_csv2 = ../Dane/dane.csv MapPartitionsRDD[50] at textFile at <console>:27
```

[30]:
```
Array(Karolina|Kozieł|FIZYKA|1|7, Weronika|Kapłon|FIZYKA|1|5,
Izabela|Snażyk|INFORMATYKA|1|5, Leo|Brockhuis|FIZYKA|2|2,
Alicja|Kawala|FIZYKA|1|7, Bartosz|Piętka|INFORMATYKA|1|3,
Dawid|Pietruch|FIZYKA|1|5, Piotr|Kukiełka|FIZYKA|2|2, Stanisław|Król|FIZYKA|2|2,
Franciszek|Kramarczyk|INFORMATYKA|1|5, Aleksandra|Popiel|FIZMED|2|2,
Kamil|Tomczyk|INFORMATYKA|1|7, Hubert|Mazur|INFORMATYKA|1|5,
Tymoteusz|Kruk|INFORMATYKA|2|2, Robert|Gałat|INFORMATYKA|2|2,
Patryk|Śledź|INFORMATYKA|1|3, Jadwiga|Bizoń|FIZMED|1|3,
Rafał|Tyczyński|FIZMED|1|7, Joanna|Zborowska|FIZMED|1|7,
Rafał|Damian|FIZYKA|1|7, Michał|Piwowarczyk|INFORMATYKA|1|7,
Weronika|Stanek|FIZYKA|1|5, Oskar|Szew…
```

[31]:
```
rdd_csv2.count()
```

[31]:
```
576
```

[32]:
```
val rdd_array = rdd_csv2.map( line => line.split('|')) //tablica tablic
rdd_array.collect()
```

```
rdd_array = MapPartitionsRDD[51] at map at <console>:27
```

[32]:
```
Array(Array(Karolina, Kozieł, FIZYKA, 1, 7), Array(Weronika, Kapłon, FIZYKA, 1,
5), Array(Izabela, Snażyk, INFORMATYKA, 1, 5), Array(Leo, Brockhuis, FIZYKA, 2,
2), Array(Alicja, Kawala, FIZYKA, 1, 7), Array(Bartosz, Piętka, INFORMATYKA, 1,
3), Array(Dawid, Pietruch, FIZYKA, 1, 5), Array(Piotr, Kukiełka, FIZYKA, 2, 2),
Array(Stanisław, Król, FIZYKA, 2, 2), Array(Franciszek, Kramarczyk, INFORMATYKA,
1, 5), Array(Aleksandra, Popiel, FIZMED, 2, 2), Array(Kamil, Tomczyk,
INFORMATYKA, 1, 7), Array(Hubert, Mazur, INFORMATYKA, 1, 5), Array(Tymoteusz,
Kruk, INFORMATYKA, 2, 2), Array(Robert, Gałat, INFORMATYKA, 2, 2), Array(Patryk,
Śledź, INFORMATYKA, 1, 3), Array(Jadwiga…
```

[48]:
```
rdd_array.filter(line => line.contains("Karolina") ).collect() //zbiera karoliny
```

[48]:
```
Array(Array(Karolina, Kozieł, FIZYKA, 1, 7), Array(Karolina, Chmielewska,
FIZMED, 1, 3), Array(Karolina, Foryś, FIZYKA, 2, 2), Array(Karolina, Wójcik,
INFORMATYKA, 1, 7), Array(Karolina, Ciesielska, FIZMED, 2, 2), Array(Karolina,
Zasadzień, FIZMED, 1, 3), Array(Karolina, Mizera, INFORMATYKA, 1, 7),
Array(Karolina, Domijan, FIZYKA, 1, 5), Array(Karolina, Podgórska, FIZYKA, 1,
5), Array(Karolina, Placek, INFORMATYKA, 1, 7), Array(Karolina, Tytko, FIZMED,
1, 5), Array(Karolina, Kozioł, FIZMED, 1, 7))
```

[49]:
```
rdd_csv2.filter(line => line.contains("Karolina") ).collect() //zbiera karoliny
```

[49]:
```
Array(Karolina|Kozieł|FIZYKA|1|7, Karolina|Chmielewska|FIZMED|1|3,
Karolina|Foryś|FIZYKA|2|2, Karolina|Wójcik|INFORMATYKA|1|7,
Karolina|Ciesielska|FIZMED|2|2, Karolina|Zasadzień|FIZMED|1|3,
Karolina|Mizera|INFORMATYKA|1|7, Karolina|Domijan|FIZYKA|1|5,
Karolina|Podgórska|FIZYKA|1|5, Karolina|Placek|INFORMATYKA|1|7,
Karolina|Tytko|FIZMED|1|5, Karolina|Kozioł|FIZMED|1|7)
```

[37]:
```
val rdd_csv = spark.read.option("header","false").csv("../Dane/dane.csv")
rdd_csv.collect()
```

```
lastException = null
rdd_csv = [_c0: string]
```

[37]:
```
Array([Karolina|Kozieł|FIZYKA|1|7], [Weronika|Kapłon|FIZYKA|1|5],
[Izabela|Snażyk|INFORMATYKA|1|5], [Leo|Brockhuis|FIZYKA|2|2],
[Alicja|Kawala|FIZYKA|1|7], [Bartosz|Piętka|INFORMATYKA|1|3],
[Dawid|Pietruch|FIZYKA|1|5], [Piotr|Kukiełka|FIZYKA|2|2],
[Stanisław|Król|FIZYKA|2|2], [Franciszek|Kramarczyk|INFORMATYKA|1|5],
[Aleksandra|Popiel|FIZMED|2|2], [Kamil|Tomczyk|INFORMATYKA|1|7],
[Hubert|Mazur|INFORMATYKA|1|5], [Tymoteusz|Kruk|INFORMATYKA|2|2],
[Robert|Gałat|INFORMATYKA|2|2], [Patryk|Śledź|INFORMATYKA|1|3],
[Jadwiga|Bizoń|FIZMED|1|3], [Rafał|Tyczyński|FIZMED|1|7],
[Joanna|Zborowska|FIZMED|1|7], [Rafał|Damian|FIZYKA|1|7],
[Michał|Piwowarczyk|INFORMATYKA|1|7], [Weronika|Stanek|FIZYKA|1|5], [Os…
```

[38]:
```
rdd_csv.count()
```

```
[38]: 576
```

```
[39]: rdd_csv.show() //obciete rekordy
```

```
+--------------------+
|                 _c0|
+--------------------+
|Karolina|Kozieł|F...|
|Weronika|Kapłon|F...|
|Izabela|Snażyk|IN...|
|Leo|Brockhuis|FIZ...|
|Alicja|Kawala|FIZ...|
|Bartosz|Piętka|IN...|
|Dawid|Pietruch|FI...|
|Piotr|Kukiełka|FI...|
|Stanisław|Król|FI...|
|Franciszek|Kramar...|
|Aleksandra|Popiel...|
|Kamil|Tomczyk|INF...|
|Hubert|Mazur|INFO...|
|Tymoteusz|Kruk|IN...|
|Robert|Gałat|INFO...|
|Patryk|Śledź|INFO...|
|Jadwiga|Bizoń|FIZ...|
|Rafał|Tyczyński|F...|
|Joanna|Zborowska|...|
|Rafał|Damian|FIZY...|
+--------------------+
only showing top 20 rows
```

```
[40]: rdd_csv.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
```

```
[41]: rdd.collect()
```

```
[41]: Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
      22, 23, 24, 25, 26, 27, 28, 29)
```

```
[42]: rdd.saveAsTextFile("../Files/test.txt")
```

```
[43]: val rdd_txt = sc.textFile("../Files/test.txt")
      rdd_txt.collect()
```

```
rdd_txt = ../Files/test.txt MapPartitionsRDD[79] at textFile at <console>:27
```

```
[43]: Array(1, 10, 20, 21, 19, 26, 27, 13, 14, 4, 5, 15, 16, 28, 29, 2, 3, 24, 25, 22,
      23, 6, 7, 11, 12, 17, 18, 8, 9)
```

```
[2]: val data1 = Array.range(1,21)
     val data2 = Array.range(19,25)
     val rdd1 = sc.parallelize(data1)
     val rdd2 = sc.parallelize(data2)
     val rdd3 = rdd1.union(rdd2)
     rdd3.collect()
```

```
data1 = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,␣
 ↪20)
data2 = Array(19, 20, 21, 22, 23, 24)
rdd1 = ParallelCollectionRDD[3] at parallelize at <console>:35
rdd2 = ParallelCollectionRDD[4] at parallelize at <console>:36
rdd3 = UnionRDD[5] at union at <console>:37
```

```
[2]: Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 19,
     20, 21, 22, 23, 24)
```

```
[3]: val rdd4 = rdd3.distinct()
     rdd4.collect()
```

```
rdd4 = MapPartitionsRDD[8] at distinct at <console>:27
```

```
[3]: Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
     22, 23, 24)
```

```
[4]: val rdda1 = sc.
      ↪parallelize(List("aa","bb","cc","dd","aa","cc","ee","ff","dd","dd","aa"))
     val rdda2 = rdda1.map( k => (k,1))
     rdda2.countByKey()
```

```
rdda1 = ParallelCollectionRDD[9] at parallelize at <console>:27
rdda2 = MapPartitionsRDD[10] at map at <console>:28
```

```
[4]: Map(cc -> 2, aa -> 3, bb -> 1, ee -> 1, ff -> 1, dd -> 3)
```

```
[5]: val rddr1 = sc.parallelize(List("aa","bb","cc","dd","ee","ff","gg","aa")).map(␣
      ↪k => (k,1))
     val rddr2 = sc.parallelize(List("aa","cc","mm","rr","tt")).map( k => (k,1))
     rddr1.join(rddr2).collect()
```

```
rddr1 = MapPartitionsRDD[14] at map at <console>:27
rddr2 = MapPartitionsRDD[16] at map at <console>:28
```

[5]:
```
Array((aa,(1,1)), (aa,(1,1)), (cc,(1,1)))
```

[6]:
```
rddr1.leftOuterJoin(rddr2).collect()
```

[6]:
```
Array((ee,(1,None)), (aa,(1,Some(1))), (aa,(1,Some(1))), (gg,(1,None)),
(dd,(1,None)), (ff,(1,None)), (bb,(1,None)), (cc,(1,Some(1))))
```

[7]:
```
rddr1.rightOuterJoin(rddr2).collect()
```

[7]:
```
Array((aa,(Some(1),1)), (aa,(Some(1),1)), (mm,(None,1)), (tt,(None,1)),
(rr,(None,1)), (cc,(Some(1),1)))
```