# Mapreduce – Duplikate finden und auflisten

from mrsim import mr s imu l a t o r

d e f map ( key , v a l ) :

# r e s 􀀀 L i s t

r e s = [ ]

# ( key , v a l ) 􀀀 Tupel , d e r d e r L i s t e h i n z u g e f u e g t wi rd

r e s . append ( ( key , v a l ) )

r e turn r e s

d e f r e d u c e ( key , v a l ) :

r e s = [ ]

i f ( l e n ( v a l ) > 1 ) :

# remove d u p l i c a t e s ( o r i g i n a l o r d e r i s n o t p r e s e r v e d )

v a l = l i s t ( s e t ( v a l ) )

# r e s . append ( ( key , v a l ) )

r e s . append ( ( s t r ( key ) + ’ : ’ + s t r ( v a l ) ) )

r e turn r e s

i n p u t s = [ ( 1 2 3 , ’Name1 ’ ) , ( 1 2 3 , ’Name2 ’ ) , ( 4 5 6 , ’Name1 ’ ) ,

( 3 4 5 , ’Name3 ’ ) , ( 4 5 6 , ’Name2 ’ ) , ( 1 2 3 , ’Name1 ’ ) ]

r e s = mr s imu l a t o r ( i n p u t s , map , r e d u c e )

p r i n t ( r e s )

# Python Suchen

a) Merge Sort:

#merge f u n c t i o n

d e f merge ( a r r 1 , a r r 2 ) :

r e s u l t = [ ]

whi l e l e n ( a r r 1 ) != 0 and l e n ( a r r 2 ) != 0 :

i f a r r 1 [ 0 ] < a r r 2 [ 0 ] :

r e s u l t . append ( a r r 1 [ 0 ] )

a r r 1 . remove ( a r r 1 [ 0 ] )

e l s e :

r e s u l t . append ( a r r 2 [ 0 ] )

a r r 2 . remove ( a r r 2 [ 0 ] )

i f l e n ( a r r 1 ) == 0 :

r e s u l t += a r r 2

e l s e :

r e s u l t += a r r 1

r e turn r e s u l t

#Code f o r merge s o r t

d e f me r g e s o r t ( a r r ) :

i f l e n ( a r r ) <= 1 :

r e turn a r r

middl e = l e n ( a r r ) / / 2

l e f t = me r g e s o r t ( a r r [ : middl e ] )

r i g h t = me r g e s o r t ( a r r [ middl e : ] )

r e turn merge ( l e f t , r i g h t )

# t e s t

p r i n t ( me r g e s o r t ( [ 3 , 6 , 8 , 1 0 , 1 , 2 , 1 ] ) )

1

b) Depth-first Search:

d e f d e e pSe a r c h ( graph , s t a r t , d e s t i n a t i o n , v i s i t e dNo d e s = [ ] ) :

i f s t a r t == d e s t i n a t i o n :

r e turn True

i f gr aph [ s t a r t ] :

f o r node i n f i l t e r ( lambda x : x n o t i n v i s i t e dNo d e s ,

gr aph [ s t a r t ] ) :

v i s i t e dNo d e s . append ( node )

i f d e e pSe a r c h ( graph , node , d e s t i n a t i o n , v i s i t e dNo d e s ) :

r e turn True

e l s e :

v i s i t e dNo d e s . append ( ’Node ’+ s t r ( node )+

’ i s f i n i s h e d , no p a t h from ’+ s t r ( node )+

’ t o ’+ s t r ( d e s t i n a t i o n ) )

r e turn F a l s e

a d j L i s t = f 0 : [ 1 , 2 ] , 1 : [ 2 , 3 ] , 2 : [ 4 ] , 3 : [ 4 , 5 ] , 4 : [ ] , 5 : [ ] g

v i s i t e dNo d e s = [ ]

p r i n t ( d e e pSe a r c h ( a d j L i s t , 0 , 5 , v i s i t e dNo d e s ) )

p r i n t ( v i s i t e dNo d e s )

c) Breadth-first Search:

import queue a s q

d e f b r e i t e n s u c h e ( adj , s t a r t , d e s t i n a t i o n , v i s i t e dNo d e s = [ ] ) :

queue = q . Queue ( )

queue . p u t ( s t a r t )

whi l e queue . q s i z e ( ) > 0 :

c u r r e n tNo d e = queue . g e t ( )

v i s i t e dNo d e s . append ( c u r r e n tNo d e )

f o r s u c c e s s o r i n a d j [ c u r r e n tNo d e ] :

i f s u c c e s s o r i n v i s i t e dNo d e s :

v i s i t e dNo d e s . append ( ’Node ’ + s t r ( s u c c e s s o r ) +

’was a l r e a d y v i s i t e d e a r l i e r . ’ )

c ont inue

e l i f s u c c e s s o r == d e s t i n a t i o n :

v i s i t e dNo d e s . append ( s u c c e s s o r )

r e turn True

e l s e :

queue . p u t ( s u c c e s s o r )

r e turn F a l s e

a d j L i s t = f 0 : [ 1 , 2 ] , 1 : [ 2 , 3 ] , 2 : [ 4 ] , 3 : [ 4 , 5 ] , 4 : [ ] , 5 : [ ] g

v i s i t e dNo d e s = [ ]

p r i n t ( b r e i t e n s u c h e ( a d j L i s t , 0 , 5 , v i s i t e dNo d e s ) )

p r i n t ( v i s i t e dNo d e s )

# Spark

from o p e r a t o r import add

from p y s p a r k l i n g import Co n t e x t

s c = Co n t e x t ( )

a) Find 25 suppliers with the lowest account balance.

t o p 2 5 = (

s c . t e x t F i l e ( ’ s u p p l i e r . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( row [ 1 ] , f l o a t ( row [ 5 ] ) ) )

. s o r tBy ( lambda row : row [ 1 ] )

. t a k e ( 2 5 )

)

# more e f f i c i e n t on PySpark ( wi t h o u t g l o b a l s o r t i n g , which c a u s e s s u b s t a n t i a l

# s h u f f l e / r e p a r t i t i o n i n g ) , e x a c t l y t h e same a s above s o l u t i o n on P y s p a r k l i n g ,

# which implements t o p ( num, key ) a s s o r tBy ( key , a s c e n d i n g = F a l s e ) . t a k e ( num)

t o p 2 5 = (

s c . t e x t F i l e ( ’ s u p p l i e r . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( row [ 1 ] , f l o a t ( row [ 5 ] ) ) )

. t o p ( 2 5 , key=lambda row : 􀀀row [ 1 ] )

# or . t a k eOr d e r e d ( 2 5 , key=lambda row : row [ 1 ] ) # only a v a i l a b l e on PySpark

)

p r i n t ( t o p 2 5 )

b) How many suppliers have a positive account balance?

num pos ba l = (

s c . t e x t F i l e ( ’ s u p p l i e r . t b l ’ )

.map ( lambda l i n e : f l o a t ( l i n e . s p l i t ( ’ j ’ ) [ 5 ] ) )

. f i l t e r ( lambda a c c t b a l : a c c t b a l > 0)

. c o u n t ( )

)

p r i n t ( num po s ba l )

c) Find out all brands produced by the same manufacturer and calculate the items number and the

total sales price for each brand of each manufacturer.

b r a n d mf g r c o u n t = (

s c . t e x t F i l e ( ’ p a r t . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( ( row [ 2 ] , row [ 3 ] ) , 1 ) )

. reduceByKey ( add )

)

br and mfgr sum = (

s c . t e x t F i l e ( ’ p a r t . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( ( row [ 2 ] , row [ 3 ] ) , f l o a t ( row [ 7 ] ) ) )

. reduceByKey ( add )

)

1

r e s u l t = b r a n d mf g r c o u n t . j o i n ( br and mfgr sum ) . s o r tBy ( lambda x : x [ 0 ] ) . c o l l e c t ( )

p r i n t ( r e s u l t )

# s h o r t e r s o l u t i o n

d e f a d d t u p l e s e l eme n twi s e (\_ a r g s ) :

r e turn t u p l e ( sum( x ) f o r x i n z i p (\_ a r g s ) )

r e s u l t = (

s c . t e x t F i l e ( ’ p a r t . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( ( row [ 2 ] , row [ 3 ] ) , ( 1 , f l o a t ( row [ 7 ] ) ) ) )

. reduceByKey ( a d d t u p l e s e l eme n twi s e )

)

p r i n t ( r e s u l t . s o r tBy ( lambda x : x [ 0 ] ) . c o l l e c t ( ) )

d) How many items have 3 words in their name?

num name l ength 3 = (

s c . t e x t F i l e ( ’ p a r t . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : l e n ( row [ 1 ] . s p l i t ( ) ) )

. f i l t e r ( lambda l e n g t h : l e n g t h == 3)

. c o u n t ( )

)

p r i n t ( num name l ength 3 )

e) How many different items does each supplier have?

s u p p l i e r = (

s c . t e x t F i l e ( ’ s u p p l i e r . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( row [ 0 ] , ( row [ 1 ] , row [ 2 ] , row [ 3 ] , row [ 4 ] , row [ 5 ] , row [ 6 ] ) ) )

)

p a r t s u p p l i e r s = (

s c . t e x t F i l e ( ’ p a r t s u p p . t b l ’ )

. map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

. map ( lambda row : ( row [ 1 ] , ( row [ 0 ] , row [ 2 ] , row [ 3 ] , row [ 4 ] ) ) )

)

s u p p l i e r p s r i g h t = (

s u p p l i e r . r i g h tOu t e r J o i n ( p a r t s u p p l i e r s )

.map ( lambda x : ( x [ 1 ] [ 1 ] [ 0 ] , ( x [ 1 ] [ 0 ] , x [ 0 ] , x [ 1 ] [ 1 ] [ 1 : ] ) ) )

)

p a r t = (

s c . t e x t F i l e ( ’ p a r t . t b l ’ )

.map ( lambda l i n e : l i n e . s p l i t ( ’ j ’ ) )

.map ( lambda row : ( row [ 0 ] , ( row [ 1 ] , row [ 2 ] , row [ 3 ] , row [ 4 ] , row [ 5 ] , row [ 6 ] , row [ 7 ] , row [ 8 ] ) ) )

)

s u p p l i e r p a r t = (

s u p p l i e r p s r i g h t . r i g h tOu t e r J o i n ( p a r t )

.map ( lambda x : ( x [ 1 ] [ 0 ] [ 0 ] [ 0 ] , 1 ) )

. reduceByKey ( add )

)

p r i n t ( s u p p l i e r p a r t . c o l l e c t ( ) )

2

Note: The is .join on Pysparkling, but it works incorrectly. The correct result for our query e)

can be obtained using .rightOuterJoin, but in the general case that won’t work.

It is possible to express and answer all 5 queries with Spark’s RDDs. For the queries c) and e),

however, the result is very elaborate and unattractive. DataFrame concept from Spark offers a

better approach.