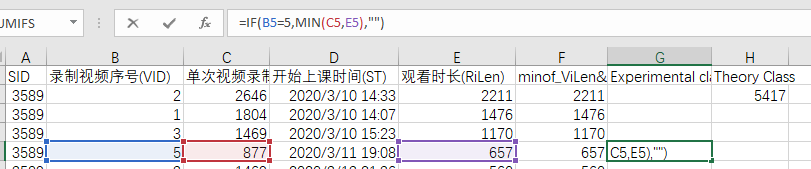
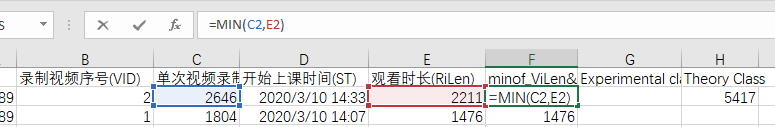
1. **根据所提供xls表，计算每个学生每节网课的有效上课时间是多少。**

答：

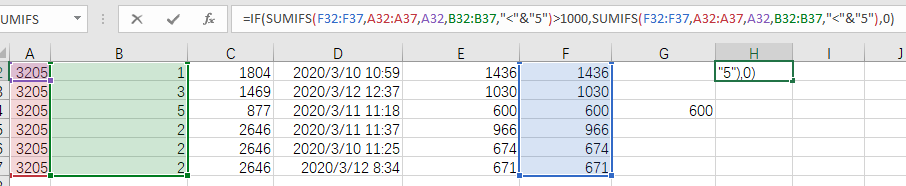
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3589 | 3342 | 3086 | 5002 | 3360 | 3575 | 3088 | 3189 | 3205 |
| 理论课 | 5417 | 1074 | 4811 | 4699 | 5873 | 2721 | 5690 | 0 | 4777 |
| 实验课 | 657 | 0 | 726 | 797 | 821 | 534 | 877 | 732 | 600 |



如图所示是实验课时长计算过程，使用Excel内置IF函数 =IF(B5=5,MIN(C5,E5),"")，判断B列编号是否为5，如果为5代表该视频为实验课视频，然后输出min(C5, E5)，代表视频录制时长和观看时长二者的最小值。



计算理论课时长时，由于也要取视频录制时长和观看时长二者的最小值，因此新创建一列用于保存二者最小者，计算方法为 =MIN(C2,E2)



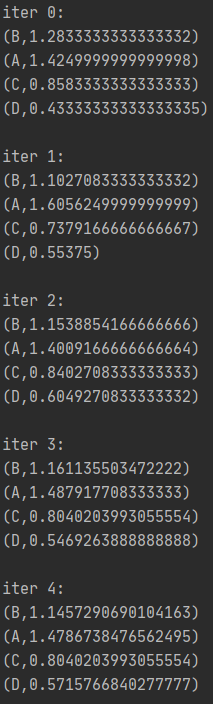
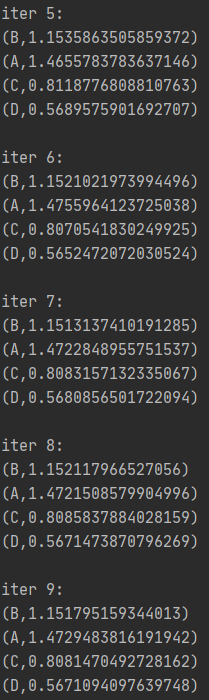
接着，使用IF嵌套SUMIFS函数用于计算理论课总时长，SUMIFS第一个参数用于确定输出的列的范围，第2b和第2b+1个参数用于形成选择条件，其中b代表第几组选择条件，每组中前一个为判定的列的范围，后一个为判定条件。这里有两组判定，=SUMIFS(F32:F37,A32:A37,A32,B32:B37,"<"&"5") 第一组为SID编号，选择A32:A37列，然后判定是否为A32，这样就把同一个SID行选中在一起，第二组为录制视频序号，选择列为B32:B37，判定条件为序号<5，这样就把理论课选中在一起，最后便输出理论课时长总和。但题目中告知如果理论课时长<1000，则判定为0，因此外层使用IF函数，当<1000时，输出0，否则按原值输出。

1. **编写一个Scheme函数，这个函数将给定的链表中的最后一个元素移出来。**

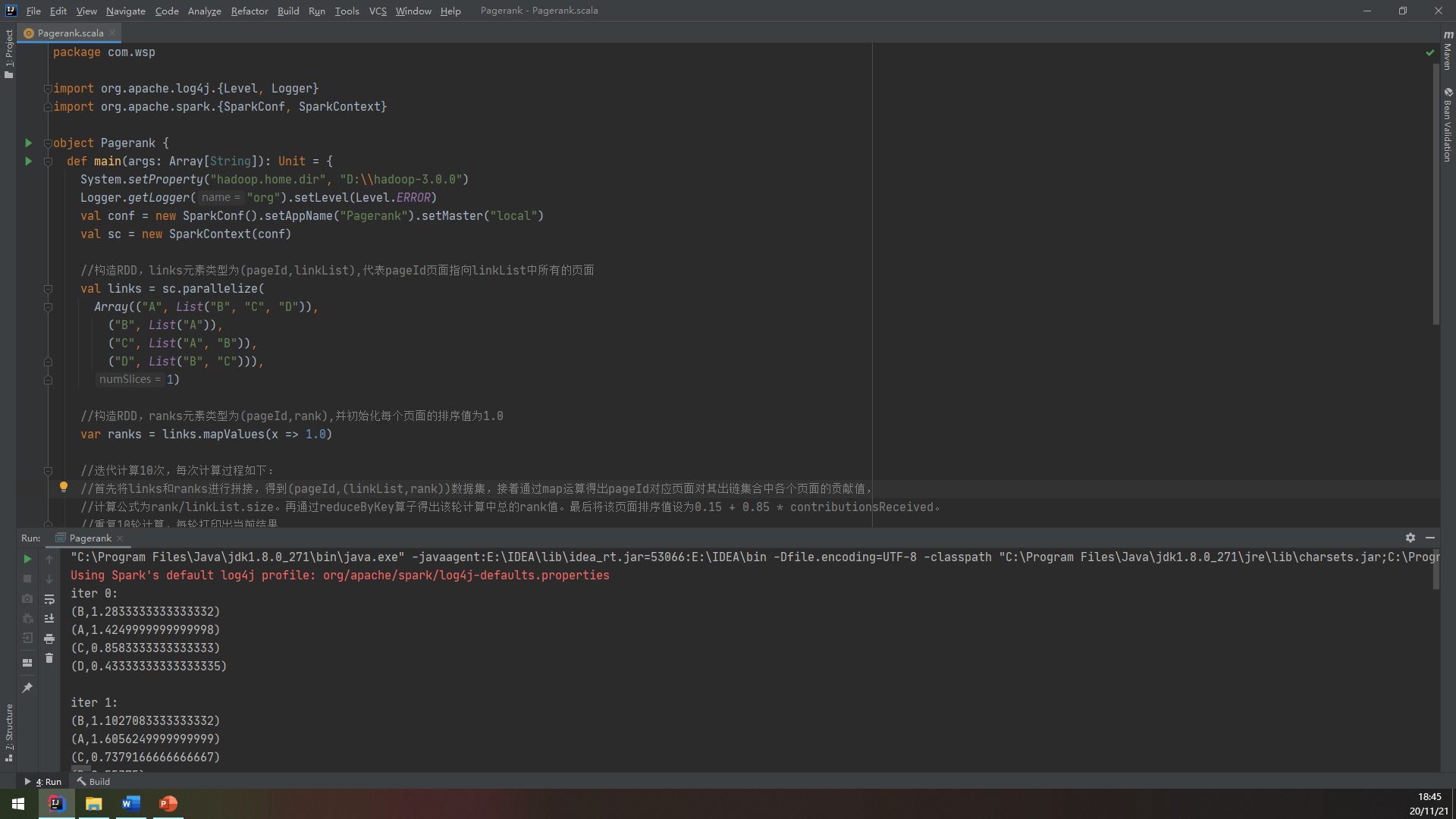
|  |
| --- |
| (DEFINE (tlrm lis)  (COND  (1< length(lis)) NULL)  (ELSE (COND  ((1? length(CDR lis)) CONS(CAR NULL))  (ELSE (tlrm (CDR lis)))  )  )  )  ) |

1. **思考题scala源码，中间注释和计算结果。**

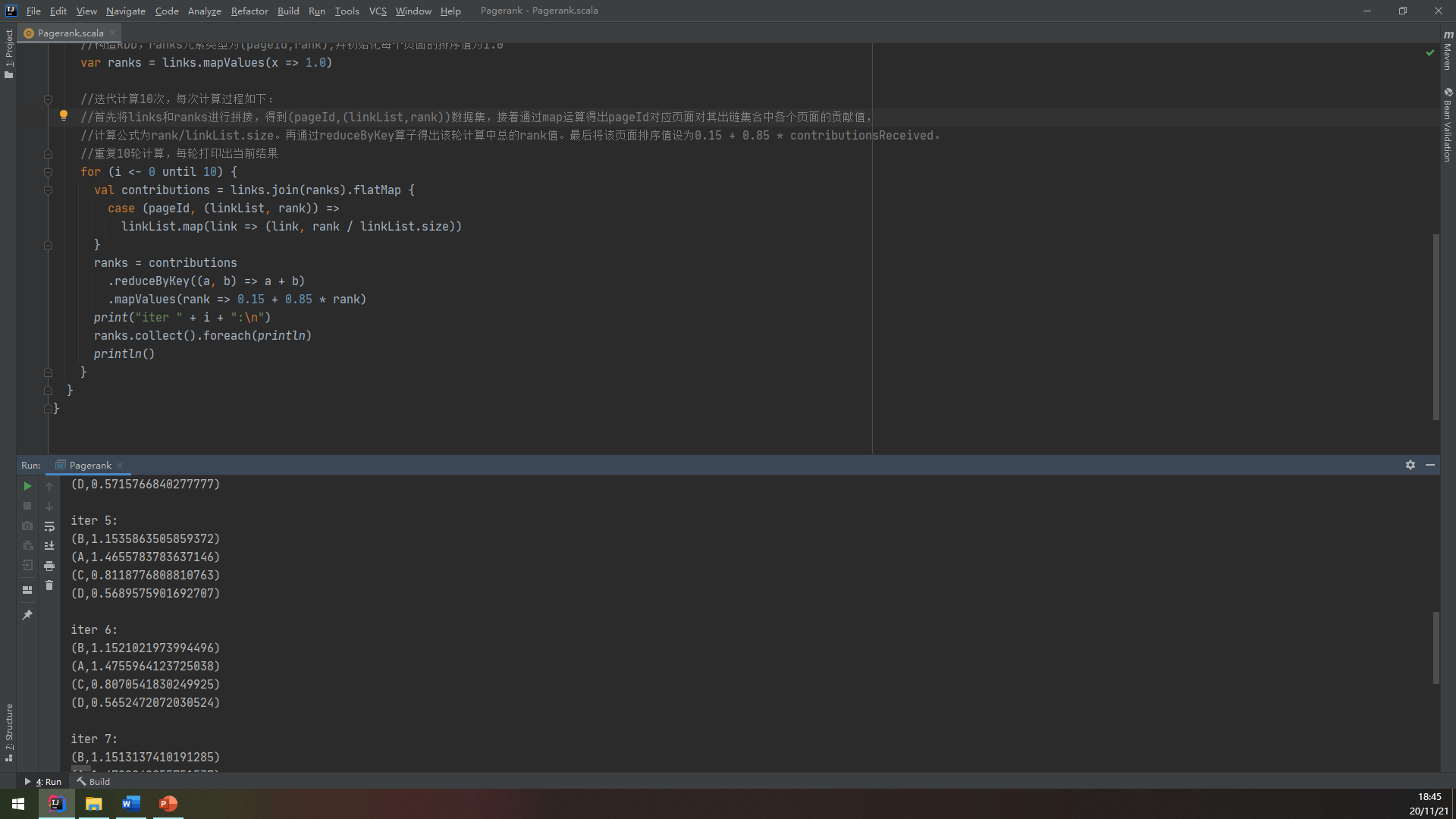
|  |
| --- |
| package com.wsp  import org.apache.log4j.{Level, Logger}  import org.apache.spark.{SparkConf, SparkContext}  object Pagerank {  def main(args: Array[String]): Unit = {  System.setProperty("hadoop.home.dir", "D:\\hadoop-3.0.0")  Logger.getLogger("org").setLevel(Level.ERROR)  val conf = new SparkConf().setAppName("Pagerank").setMaster("local")  val sc = new SparkContext(conf)  //构造RDD，links元素类型为(pageId,linkList),代表pageId页面指向linkList中所有的页面  val links = sc.parallelize(  Array(("A", List("B", "C", "D")),  ("B", List("A")),  ("C", List("A", "B")),  ("D", List("B", "C"))),  1)  //构造RDD，ranks元素类型为(pageId,rank),并初始化每个页面的排序值为1.0  var ranks = links.mapValues(x => 1.0)  //迭代计算10次，每次计算过程如下：  //首先将links和ranks进行拼接，得到(pageId,(linkList,rank))数据集，接着通过map运算得出pageId对应页面对其出链集合中各个页面的贡献值，  //计算公式为rank/linkList.size。再通过reduceByKey算子得出该轮计算中总的rank值。最后将该页面排序值设为0.15 + 0.85 \* contributionsReceived。  //重复10轮计算，每轮打印出当前结果  for (i <- 0 until 10) {  val contributions = links.join(ranks).flatMap {  case (pageId, (linkList, rank)) =>  linkList.map(link => (link, rank / linkList.size))  }  ranks = contributions  .reduceByKey((a, b) => a + b)  .mapValues(rank => 0.15 + 0.85 \* rank)  print("iter " + i + ":\n")  ranks.collect().foreach(println)  println()  }  }  } |

****

前5次迭代 后5次迭代



环境演示



环境演示