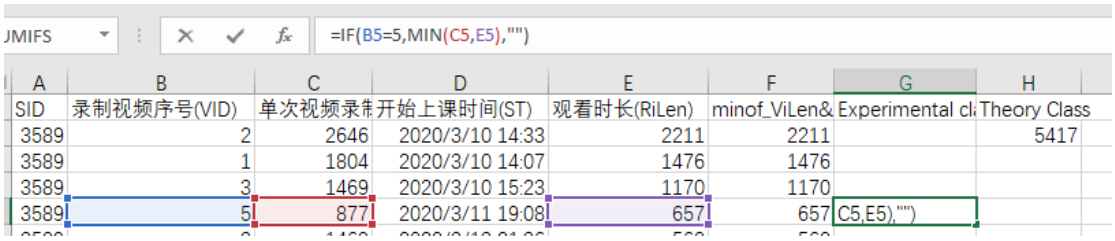
1、 根据所提供 xls 表，计算每个学生每节网课的有效上课时间是多少。

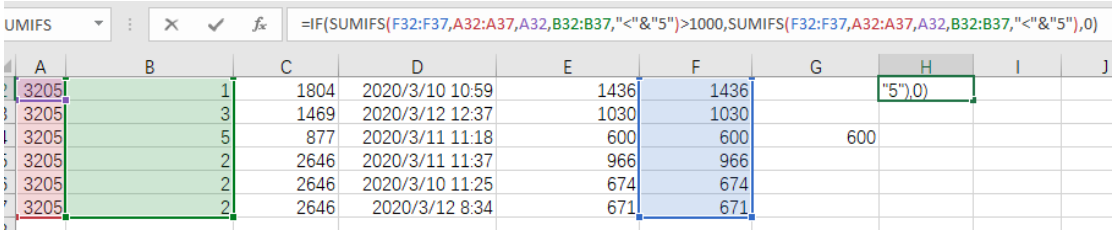
答：

	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。

2、 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)))
            )
        )
    )
)
```

3、 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) {
```

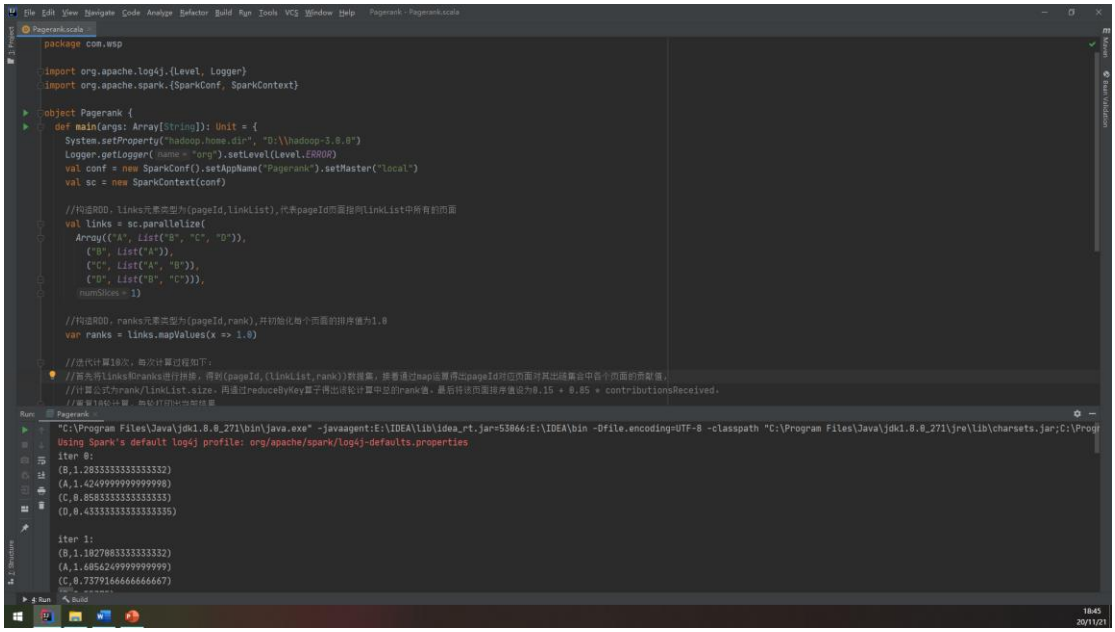
```
      val contributions = links.join(ranks).flatMap {
        case (pageId, (linkList, rank)) =>
          linkList.map(link => (link, rank / linkList.size))
      }
      ranks = contributions
        .reduceByKey((a, b) => a + b)
        .mapValues(rank => 0.15 + 0.85 * rank)
      print("iter " + i + ":\n")
      ranks.collect().foreach(println)
      println()
    }
  }
}
```

```
iter 0:
(B,1.2833333333333332)
(A,1.4249999999999998)
(C,0.8583333333333333)
(D,0.43333333333333335)

iter 1:
(B,1.1027083333333332)
(A,1.6056249999999999)
(C,0.7379166666666667)
(D,0.55375)

iter 2:
(B,1.1538854166666666)
(A,1.4009166666666664)
(C,0.8402708333333333)
(D,0.6049270833333332)

iter 3:
(B,1.161135503472222)
(A,1.487917708333333)
(C,0.8040203993055554)
(D,0.5469263888888888)

iter 4:
(B,1.1457290690104163)
(A,1.4786738476562495)
(C,0.8040203993055554)
(D,0.5715766840277777)
```

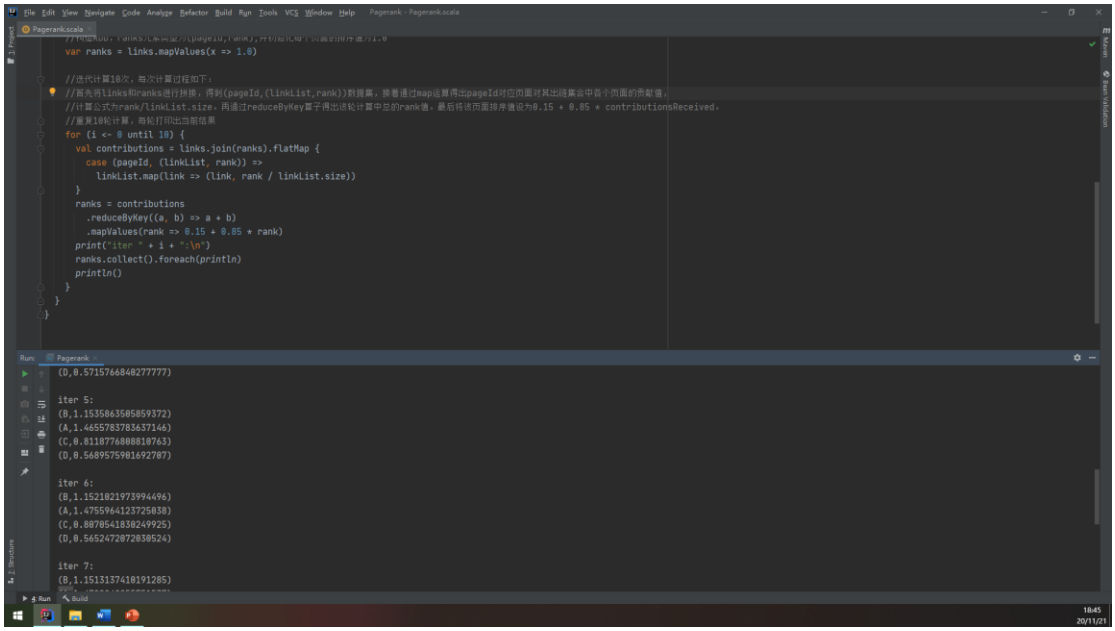
前 5 次迭代

```
iter 5:
(B,1.1535863505859372)
(A,1.4655783783637146)
(C,0.8118776808810763)
(D,0.5689575901692707)

iter 6:
(B,1.1521021973994496)
(A,1.4755964123725038)
(C,0.8070541830249925)
(D,0.5652472072030524)

iter 7:
(B,1.1513137410191285)
(A,1.4722848955751537)
(C,0.8083157132335067)
(D,0.5680856501722094)

iter 8:
(B,1.152117966527056)
(A,1.4721508579904996)
(C,0.8085837884028159)
(D,0.5671473870796269)

iter 9:
(B,1.151795159344013)
(A,1.4729483816191942)
(C,0.8081470492728162)
(D,0.5671094097639748)
```

后 5 次迭代


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(name = "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"))),
numSlices = 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。
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -javaagent:E:\IDEA\lib\idea_rt.jar=53866:E:\IDEA\bin -Dfile.encoding=UTF-8 -classpath "C:\Program Files\Java\jdk1.8.0_271\jre\lib\charsets.jar;C:\Progr
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
iter 0:
(B,1.2833333333333332)
(A,1.4249999999999998)
(C,0.8583333333333333)
(D,0.4333333333333335)
iter 1:
(B,1.1827083333333332)
(A,1.6056249999999999)
(C,0.7379166666666667)

环境演示


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()
(D,0.5715766848277777)
iter 5:
(B,1.1535863505859372)
(A,1.4655783783637146)
(C,0.8110776808810763)
(D,0.5689575901692707)
iter 6:
(B,1.1521821973994496)
(A,1.4755964123725038)
(C,0.8070541830249925)
(D,0.5652472072030524)
iter 7:
(B,1.1513137418191285)

环境演示