1. **项目成果总述**

作者队伍最终成绩：总用时**23min**（下载与解压文件用时**17min**，运行代码用时**6min**） 最终成绩并不理想，因为使用校园网在百度网盘下载数据网速并不稳定

作者在网速正常时用大小相同的测试数据进行全过程最快速度为**13min**（下载与解压文件用时7min，运行代码用时6min）

作者作为小组组长，已经将项目实践过程开源到Github：<https://github.com/foiberh/Red-code-annotation-using-Spark>

其中包含小组的全部代码以及自己构建的小数据集

1. **小组方案选择**

作者与组内同学经过短时间的讨论，确定几种可行方案如下：

1. 使用Linux虚拟机进行Spark完全分布式集群的搭建
2. Windows系统下的Spark local模式的搭建
3. 利用docker完成Windows系统下Spark环境的搭建

作者确定这几种方案的原因如下：

1. 在本学期之前课程的学习过程中已经与组内同学完成了Hadoop完全分布式集群的搭建，集群结构为一主四从。而Spark完全分布式的搭建只需在此基础上完成对Spark开发环境的搭建即可
2. 虽然方案一已经可以完成本次项目要求，但由于本次项目的竞速性质，作者考虑到虚拟机可能无法调用电脑的全部性能，导致处理速度较慢
3. 同时考虑到在Windows系统搭建Spark开发环境可能涉及较多环境变量配置的操作，作者也考虑到是否可以利用docker容器进行快速部署
4. **个人参与部分**
5. **作者在测试数据集刚下发后就实现了使用dask的python本地编程实现，在测试数据集上代码运行用时3分钟，我们最终将其作为检验结果正确率的方法**
6. **作者与组内同学首先成功完成了方案一中Linux系统下Spark集群的搭建**，但在进一步尝试完成项目要求时遇到了诸多问题：
   1. 虚拟机存储空间不足
   2. Spark完全分布式集群若想在本地运行，需要将数据储存在所有节点的相同路径下，而且本地运行不太稳定。若将数据先上传至HDFS，由于文件较大，且校园网不太稳定，耗时预估一个小时左右。
7. 此时组内另一位同学完成了Windows系统下Spark集群的搭建，并完成了Scala代码的编写与运行

我们将这种方法也保留为最终的方案之一

1. pyspark

**作者进一步独自实现了在pycharm上pyspark环境的配置，并独自完成了pyspark代码的编写**

期间因为作者第一次接触pyspark库，debug的过程异常艰难

**所以作者与小组内同学自己编写了一个数据集**，数据量只有50行左右，但其中包含了符合项目条件的所有正常与特殊情况

1. **心得体会**

在这个项目中，我收获了许多宝贵的经验和教训，尤其是在团队合作、方案选择以及具体解决代码问题的过程中。

1. 团队合作的重要性

在项目的初期，团队成员之间的有效沟通和协作是确保项目顺利推进的关键。通过与组内同学的讨论，我们迅速确定了几种可行的方案，并分工合作完成了不同环境下的实验。这不仅提高了我们的工作效率，还让我深刻体会到团队合作的重要性。

1. 方案的选择与优化

在项目的方案选择过程中，我们考虑了多种不同的技术方案，包括使用Linux虚拟机进行Spark完全分布式集群的搭建、在Windows系统下的Spark local模式的搭建，以及利用docker完成Windows系统下Spark环境的搭建。最终，我们选择了在Windows系统下使用Spark local模式，并通过pyspark进行代码编写和调试。这个过程让我认识到，合理的技术选型和环境配置能够极大地提升工作效率和项目性能。

1. 代码编写与样例数据测试

**在项目实施过程中，我独自实现了pyspark环境的配置并编写了相应的代码。**由于这是我第一次接触pyspark库，调试过程非常艰难，但也让我学到了很多新的知识和技能。这次经历让我意识到，持续学习和探索新技术是非常重要的，遇到困难时不要轻易放弃，要善于利用各种资源解决问题。

**在使用样例数据测试检测代码可行性以及鲁棒性方面。测试数据集刚下发后，我使用dask的python本地编程实现了快速处理，并选择将其作为检验结果正确率的方法。**这个过程让我认识到，选择合适的工具和方法进行数据处理和测试，能够有效提高工作效率。

**此外，我在代码难以调通的情况下，转变思路，选择与组员先构建一个小而精的测试数据集，以实现更快、更有效的代码检测。**其中包含了符合项目条件的所有正常与特殊情况，这为我们在调试和测试过程中提供了很大的帮助。也是我认为在调试代码过程中无比关键的一步。

1. **附上代码（pySpark）**

from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, lead  
from pyspark.sql.window import Window  
from timeit import default\_timer as timer  
import shutil  
import os  
  
start = timer()  
  
# 目标目录  
output\_dir = "redmark13"  
  
# 删除目标目录中的所有文件  
if os.path.exists(output\_dir):  
 shutil.rmtree(output\_dir)  
  
# 创建 SparkSession 对象，设置为本地模式  
spark = SparkSession.builder \  
 .appName("SuperRedMark") \  
 .master("local[\*]") \  
 .config("spark.executor.memory", "12g") \  
 .config("spark.driver.memory", "12g") \  
 .config("spark.memory.fraction", "0.8") \  
 .config("spark.memory.storageFraction", "0.3") \  
 .getOrCreate()  
  
# 读取基站信息文件  
cdinfo\_df = spark.read.option("header", "false").option("inferSchema", "true").csv(r"G:\BaiduNetdiskDownload\cdinfo.txt")  
cdinfo\_df = cdinfo\_df.withColumnRenamed("\_c0", "station\_id")\  
 .withColumnRenamed("\_c1", "time")\  
 .withColumnRenamed("\_c2", "event")\  
 .withColumnRenamed("\_c3", "phone")  
  
# 将time列转换为字符串类型  
cdinfo\_df = cdinfo\_df.withColumn("time", col("time").cast("string"))  
  
# 读取感染者信息文件  
infected\_df = spark.read.option("header", "false").option("inferSchema", "true").csv(r"G:\BaiduNetdiskDownload\infected.txt")  
infected\_df = infected\_df.withColumnRenamed("\_c0", "infected\_phone")  
  
# 过滤出感染者的基站进出信息  
infected\_cdinfo\_df = cdinfo\_df.join(infected\_df, cdinfo\_df.phone == infected\_df.infected\_phone)  
  
# 窗口函数按基站和手机分组并排序  
window\_spec = Window.partitionBy("phone", "station\_id").orderBy("time")  
  
# 使用窗口函数计算进出基站的时间  
infected\_cdinfo\_df = infected\_cdinfo\_df.withColumn("next\_event", lead("event").over(window\_spec))\  
 .withColumn("next\_time", lead("time").over(window\_spec))  
  
# 计算感染者在每个基站的停留时间  
infected\_cdinfo\_df = infected\_cdinfo\_df.withColumn("infected\_start\_time", col("time"))\  
 .withColumn("infected\_end\_time", col("next\_time"))\  
 .filter((col("event") == 1) & (col("next\_event") == 2))  
  
# 提取感染者的停留区间  
infected\_intervals\_df = infected\_cdinfo\_df.select("station\_id", "infected\_phone", "infected\_start\_time", "infected\_end\_time")  
  
# 窗口函数按基站和手机分组并排序（对所有人的基站进出记录）  
all\_window\_spec = Window.partitionBy("phone", "station\_id").orderBy("time")  
  
# 计算所有人在每个基站的进出时间  
all\_cdinfo\_df = cdinfo\_df.withColumn("next\_event", lead("event").over(all\_window\_spec))\  
 .withColumn("next\_time", lead("time").over(all\_window\_spec))  
  
# 计算所有人在每个基站的停留时间  
all\_cdinfo\_df = all\_cdinfo\_df.withColumn("start\_time", col("time"))\  
 .withColumn("end\_time", col("next\_time"))\  
 .filter((col("event") == 1) & (col("next\_event") == 2))  
  
# 将所有基站进出信息与感染者的停留区间进行比较，找出在相同基站和时间区间有重叠的手机号码  
redmark\_df = all\_cdinfo\_df.join(infected\_intervals\_df, "station\_id")\  
 .filter((col("start\_time") <= col("infected\_end\_time")) &  
 (col("end\_time") >= col("infected\_start\_time")) &  
 (col("phone") != col("infected\_phone")))\  
 .select(col("phone").cast("string")).distinct()  
  
# 创建一个 DataFrame，仅包含感染者手机号  
infected\_phone\_df = infected\_intervals\_df.select(col("infected\_phone").cast("string"))  
  
# 将感染者手机号加入到最终结果中  
redmark\_df\_filtered = redmark\_df.union(infected\_phone\_df).distinct()  
  
# 将phone列转换为数值类型进行排序  
redmark\_df\_sorted = redmark\_df\_filtered.withColumn("phone\_num", col("phone").cast("long")).orderBy("phone\_num")  
  
# 转换回字符串类型并保存结果到文件  
redmark\_df\_sorted = redmark\_df\_sorted.select(col("phone\_num").cast("string").alias("phone")).coalesce(1)  
  
# 计算最终的手机号码数量  
phone\_count = redmark\_df\_sorted.count()  
print(f"最终手机号码数量: {phone\_count}")  
  
# 保存结果到单个文件  
redmark\_df\_sorted.write.mode("overwrite").text("redmark13")  
  
print("共运行时间" + f"{timer() - start}")  
  
# 停止SparkSession  
spark.stop()