计算机体系结构课程实验报告

赵悦棋 517021910158

1. 实验环境

a) 处理器：2.3GHz Dual-Core Intel i5-7360U
b) 内存：8GB 2133MHz LPDDR3
c) 操作系统：macOS 10.15.5
d) 监测工具：NULL
e) JDK：1.8.0_251
f) Scala：2.12.11
g) Spark：2.4.6
h) SBT：1.13.12

2. 环境搭建

初次搭建实验环境选择的是 Windows 平台，使用了 Win10 2004，但是在根据教程搭建环境的过程中遇到了诸多问题，比如：教程的时效性不强，本着用新不用旧的心态我一开始配置了最新版的 Scala 2.13，但后来发现即使是最新的 Spark 3.0 也只支持到 Scala 2.12，首次回滚；回滚完之后根据教程初始化了项目，但是在国内使用 Maven 拉取依赖速度非常慢，于是配置了 OpenWrt 的路由器做透明代理；拉取完依赖注意到 IDEA 自带的 Scala Archetype 已经很老了，自动生成的 pom 文件中的仓库地址直接访问会得到 404，于是照着教程覆盖了 pom 文件；为了尽快搭建出环境来此时我已经回滚 Scala 到和教程中一模一样的版本，并使用了教程中的 pom 文件，然后又注意到 pom 文件中使用 Scala 版本变量来拉取依赖，但是由于 Spark Core 同时兼容 Scala 2.10 和 2.11，所以依赖在仓库中的名称已经更新为 2.11 结尾，导致拉取依赖又出错，再次修改；花了很多时间才真的让项目导入完毕，但写了 demo 程序发现根本跑不起来……

在尝试了网络上的各种教程后我仍无法使用 Maven 在 Windows 平台搭建环境，于是转投了 macOS 平台，并改用 SBT 作为包管理工具，有了先前配置的路由器，通过 brew 安装 SBT 十分顺利，并且没有奇怪的依赖错误，最终我在 macOS 平台完成了实验环境的搭建，为了测试环境运行正常我根据教程中的代码写了 WordCount 来初上手 Spark 和 Scala，运行情况良好，我终于可以正式开始实验了。

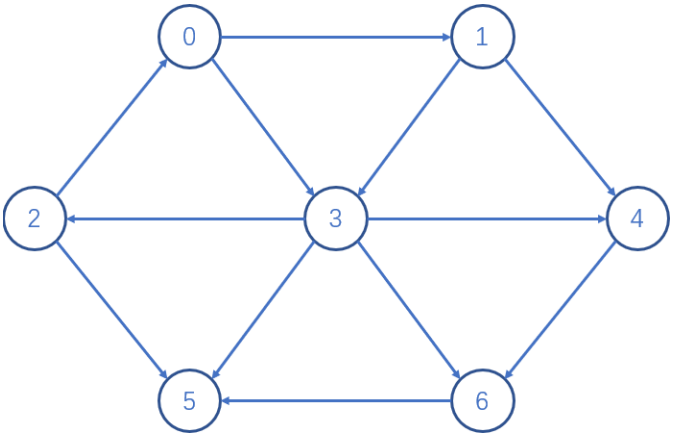
Spark 在运行过程中会有大量的 INFO 输出，为了在撰写报告的时候能截到比较美观的输出，需要在工程文件中手动引入 Sprak 默认的配置文件，并将输出从 INFO 修改为 ERROR，这样可以隐藏所有的运行信息仅保留我的代码输出。

```
# Set everything to be logged to the console
log4j.rootCategory=ERROR, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
```

3. 实验一

单源最短路径问题指的是给定一个含权有向图，并指定其中一个顶点为源，计算从源到其他所有顶点的最短路径长度的问题。根据 GraphX 文档中 Graph Builders 板块的

指导，我为下图写了一个可以被 GraphLoader.edgeListFile 加载的文件（data/Graph.txt）。



由于 GraphLoader.edgeListFile 规定了格式只能是源点和终点的组合，我无法在使用它加载图的条件下为每一条边增加权重，因此所有的边的权重都是默认的 1。通过打印 iptGraph 的内容，验证了自定义的图已经被正确地加载。

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home/bin/java ...
[info]<--------Edge Information------> [info]<-------Vertex Information----->
Edge(0,1,1)                           (4,1)
Edge(0,3,1)                           (0,1)
Edge(1,3,1)                           (6,1)
Edge(1,4,1)                           (2,1)
Edge(2,0,1)                           (1,1)
Edge(2,5,1)                           (3,1)
Edge(3,2,1)                           (5,1)
Edge(3,4,1)
Edge(3,5,1)
Edge(3,6,1)
Edge(4,6,1)
Edge(6,5,1)
```

接下来就要解决 SSSP 问题，解 SSSP 用到了 Dijkstra 算法的思想。在原图的基础上生成一个新的图，每个顶点的权重用来记录从源点到该顶点的最短路径，因此在初始化阶段除了源点外其余所有顶点的权重均设置为 INF（取 Int 的最大值）。然后利用 Pregel 开始求解，核心思路就是对于每一个顶点，如果从我出发到另一个顶点的路径长度比它本身记录的路径长度还要短，那么就更新那个顶点的属性，直到图的状态不再变化为止。

Pregel 的格式为 Pregel(initialMsg)(vprog, sendMsg, mergeMsg)。其中 initialMsg 部分为初始化消息，这个消息会发给每一个顶点用于初始化属性，因此也为 INF；vprog 根据收到的消息更新属性，根据核心思路，此处取最小值；sendMsg 为消息发送函数，该函数的运行参数是一个代表边的上下文，同样的根据核心思路可以写出条件语句，但是注意由于迭代并不是从源点开始，且 INF 并不是真的无穷大而是整形的最大值，因此对于当前权重为 INF 的顶点必需跳过，否则加上边的权重之后会溢出变为负边，导致最短路径可以到无穷小的错误；mergeMsg 是收到多条消息时的合并逻辑，不难想到收到多条消息时应选择其中值最小的接收。最终输出如上图所示，笔头验证无误，详见 SSSP.scala 中的代码和注释。

```
[info]<---------Shortest Path-------->
(4,2)
(0,0)
(6,2)
(2,2)
(1,1)
(3,1)
(5,2)
```

4. 实验二

内容

5. 实验三

内容

6. 问答题

内容