# 项目报告

## 一、项目内容（对项目要完成的内容进行描述）

1.代码实现Prim实现#4（基于堆）

2.代码实现 Kruskal实现#2（基于UNION-FIND）

3.设计实验，针对多组相同实例，比较真实运行时间

4.提交实验报告

## 二、项目实施方案（包括求解思路，数据结构设计，算法描述）

1.Prim实现思路：

使用edge存储每条边的信息（权重，起始点，终止点），node存储所有顶点，used\_node维护树上结点，used\_edge维护树上结点的所有边，将used\_edge按权重建堆，每次提取堆中的最小值，即heappop。将提取出来的edge存储于MST[ ]中，每次提取完最小edge重新整理堆，直到所有顶点都在树上。

2.Kruskal实现思路：

编写find\_father函数：找到当前顶点所在集合的leader

编写union函数：即合并两个集合

首先使用归并排序将所有边按权重升序排序，初始时所有的顶点各自为一个集合。之后按照排序顺序，如果该边的两个顶点的leader不是同一个，便使用union合并两个集合，并且将该边存储在MST上。

## 三、实验方案（介绍实验目的，使用的方法，用例的生成，如何对比等）

实验目的：体会python使用方法，体会Prim和Kruskal算法的实现方法和算法优劣

使用的方法：根据算法实现描述，建立堆使用python自带的库heapq，而union-find算法自己编写

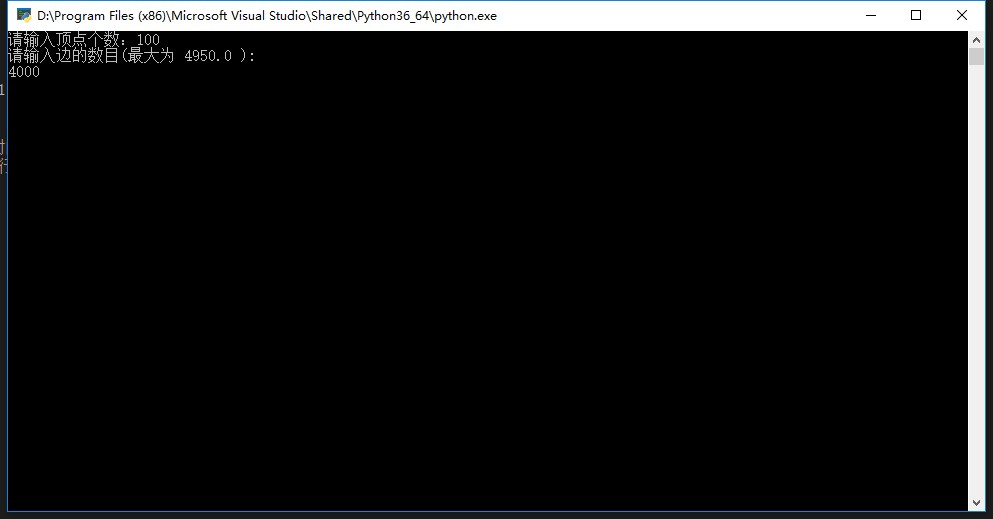
用例生成：根据输入的顶点数、边数生成随机图

如何对比：使用Prim和Kruskal分别求最小MST，比较所耗费的时间以及是否准确

## 四、实验结果分析（展示实验结果，对结果进行分析，说明实验结果背后的原因）

实验结果：

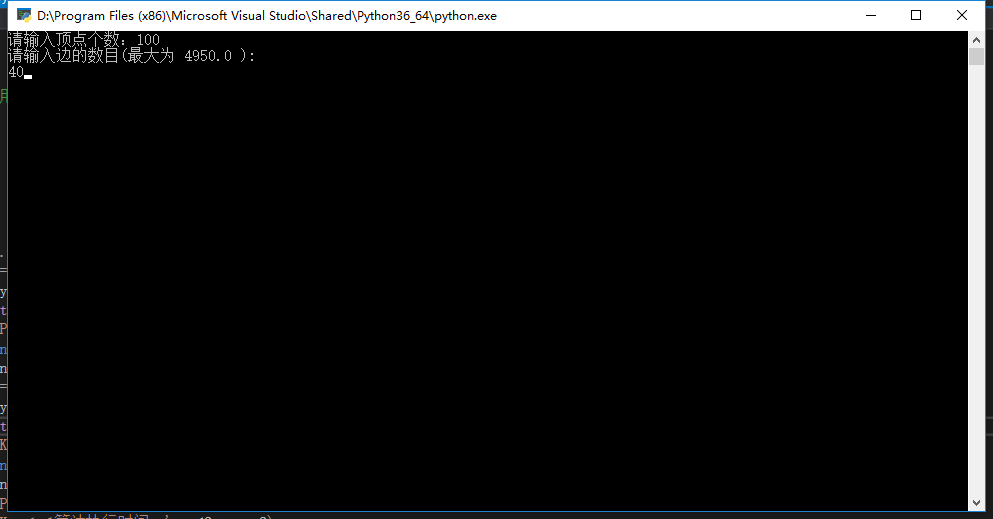
1.输入100个顶点，4000条边（密集图）



两算法时间比较为：



2.输入100个顶点，40条边（疏散图）



算法执行时间为：



可以看到Kruskal执行时间/Prim执行时间明显降低，由于UNION-FIND为手写代码，代码效率要低于heapq库中的堆的代码效率，所以不论我如何设置参数，Kruksal用于要慢于Prim

但比较两算法之间的倍数关系，可以大致得到在疏密图Kruskal运行时间较短，密集图Prim运行时间较短的结论