课程报告

王昱龙 郭宇辰 周可玉 梁子豪

一、问题描述：

使用 PW 数据集中的 mashup.csv 文件构建一个无向图，该无向图中的点表示 api，边表示两个 api 之间曾经有过合作构建 mashup，两个边之间的权重根据两个 api 之间的合作次数进行函数映射表示兼容性。api.csv 文件表明每个 api 有一到多个关键词可以描述。

目标是使用算法，对于输入特定的关键词，可以返回 n 组 api，每组 api 表示的点在无向图中是连通的，n 组 api 采用按照权重的降序排列。

同时还要使用命中率指标和时间开销指标对算法的有效性和效率进行评价。

接着在原先方法的基础上进行改进，再次使用两个指标进行评价，观察其有效性和效率是否发生改善。

二、设计思路：

数据处理：

1. 首先根据 mashup 表和 api 表生成新的 apiTable，里面是 mashup 中出现过的 api 并给它们整数编号。
2. 根据 apiTable 中 api 到 key 的映射关系得到新的表 keyToAPIs，是 key 到 api 的映射，同时也对 key 进行编号。
3. 根据 mashup 得到一张 corporation 表，里面反映了不同 api 之间的合作次数。
4. 最后将 mashup 中的 api 和 key 全都对应成其编号以便进行测试。

代码实现部分：

1.输入一组编号表示的 key（数字和 key 的对应关系见 singleKeyToAPIs.csv）
2.对每个 key 从 singleKeyToAPIs 里找到一组 api，
3.递归执行 k 层循环，遍历 api 的组合：每次循环需要重新初始化 dp 和 routes，pow 不变。
4.将每个 api 组合作为输入的 k 个关键节点，执行算法体
5.将每个组合运算的结果：权值和路径按照权值大小保存在优先队列中
6.设置 topS 数量 s=10，然后输出前十条路径，也就是推荐 api。
7.命中率检测：需要将 mashup 中添加一列 numAPIs 用数字表示的 apis。
8.随机从每行的 numAPIs 抽出几个关键词作为输入，作 total = 100 次输入，统计 mashup 中
9.实际的 api 在前十条数字路径中被包括的次数 count，count/total 就是命中率。

三、算法设计思路：

首先将参考代码通过增加 HashSet<Integer>数组改进为可以显示出具体路径的形式：

```
import java.util.*;
public class referenceCode {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(), m = sc.nextInt(), k = sc.nextInt();
```

```
        long[][] pow = new long[n][n]; // 用于存储节点之间的权重，表达无向图
        long[][] dp = new long[n][1 << k]; // 存储状态压缩 DP 的结果
        @SuppressWarnings("unchecked")
        HashSet<Integer>[][] routes = new HashSet[n][1 << k];// 创建一个二维数组，每个元素都是一个 HashSet<Integer>，
用来存储路径上的结点
        // 初始化 routes 数组，为每个元素创建一个新的 HashSet
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < 1<<k ; j++) {
                routes[i][j] = new HashSet<>();
            }
        }
        // 初始化 pow 和 dp 数组
        for (int i = 0; i < n; i++) {
            Arrays.fill(pow[i], Integer.MAX_VALUE); // 初始化距离为最大值
            pow[i][i] = 0; // 节点到自身的距离为 0
            Arrays.fill(dp[i], Integer.MAX_VALUE); // 初始化状态 DP 为最大值
        }
        // 读取边的信息并记录节点之间的最小距离
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt() - 1, v = sc.nextInt() - 1, w = sc.nextInt();
            // 采用邻接矩阵存储结果
            pow[u][v] = Math.min(pow[u][v], w);
            pow[v][u] = Math.min(pow[v][u], w);
        }
        //读取所有关键节点
        int y = -1;// 用来记录最后一个关键点
        // 标记关键节点，并初始化状态关键节点的 DP 值和 routes 为结点本身
        for (int i = 0; i < k; i++) {
            int x = sc.nextInt() - 1;
            dp[x][1 << i] = 0; // 以输入的关键点为根，二进制中对应关键节点位置为 1 的 dp 总权重为 0
            routes[x][1 << i].add(x+1);
            y = x; // 记录最后一个关键节点
        }
        // 使用状态压缩 DP 求解最小路径和
        // 遍历从单个节点开始，直到满足所有关键节点都连同的所有可能状态
        for (int s = 1; s < 1 << k; s++) {
            // 遍历以节点 i 为根节点，状态为 s 时的情况
            for (int i = 0; i < n; i++) {
                // 遍历 s 二进制的所有子集，执行 i 的度数大于 1 时的操作
                // 更新 routes 里的路径
                for (int t = s & (s - 1); t > 0; t = (t - 1) & s) {
                    //dp[i][s] = Math.min(dp[i][s], dp[i][t] + dp[i][s ^ t]);
                    if(dp[i][t] + dp[i][s ^ t]<dp[i][s]){
                        dp[i][s]=dp[i][t] + dp[i][s ^ t];
```

```
                    routes[i][s].addAll(routes[i][t]);
                    routes[i][s].addAll(routes[i][s ^ t]);
                }
            }
        }
        // 处理状态为 s 的情况下最短路径
        deal(s, n, dp, pow,routes);
    }
    System.out.print("斯坦纳树的权重为：");
    System.out.println(dp[y][(1 << k) - 1]); // 输出结果
    System.out.println("路径矩阵为：");
    for (int j = 0; j < 1<<k ; j++) {
        System.out.println("第"+j+"列路径");
        for (int i = 0; i < n; i++) {
            System.out.print(routes[i][j]+"  ");
        }
        System.out.println();
    }
    System.out.print("斯坦纳树的路径为：");
    System.out.println(routes[y][(1 << k) - 1]);
}
// 处理状态 s 下的最短路径情况
private static void deal(int s, int n, long[][] dp, long[][] pow, HashSet<Integer>[][] routes) {
    // 建立优先队列，按照 dp 值升序排列
    PriorityQueue<long[]> pq = new PriorityQueue<>(Comparator.comparingLong(o -> o[1]));
    boolean[] vis = new boolean[n];
    for (int i = 0; i < n; i++) {
        if (dp[i][s] != Integer.MAX_VALUE) {
            pq.add(new long[]{i, dp[i][s]});
        }
    }
    // 使用 Dijkstra 算法求解最短路径，即求解以每个节点为顶点，满足状态 s（包含对应的点）的权重和的最小值
    while (!pq.isEmpty()) {
        long[] tmp = pq.poll();//删除队列首元素并保存在 tmp 中
        // 如果该行表示的点已经访问，则跳过
        if (vis[(int) tmp[0]]) {
            continue;
        }
        //否则进行访问并设置为已访问
        vis[(int) tmp[0]] = true;
        //遍历所有与 tmp 对应点直接连接的点
        for (int i = 0; i < n; i++) {
            // 跳过所有不与 tmp 对应点相连的点和该点本身
            if (i == tmp[0] || pow[(int) tmp[0]][i] == Integer.MAX_VALUE) {
```

```
                    continue;
                }
                if (tmp[1] + pow[(int) tmp[0]][i] < dp[i][s]) {
                    //如果遍历到的点的 dp 值可以更新为更小的由 tmp 的 dp 值+两点间权重就更新其 dp 和 routes
                    dp[i][s] = (tmp[1] + pow[(int) tmp[0]][i]);
                    routes[i][s].clear();
                    routes[i][s].addAll(routes[(int)tmp[0]][s]);
                    routes[i][s].add(i+1);
                    pq.add(new long[]{i, dp[i][s]});
                }
            }
        }
    }
}
```

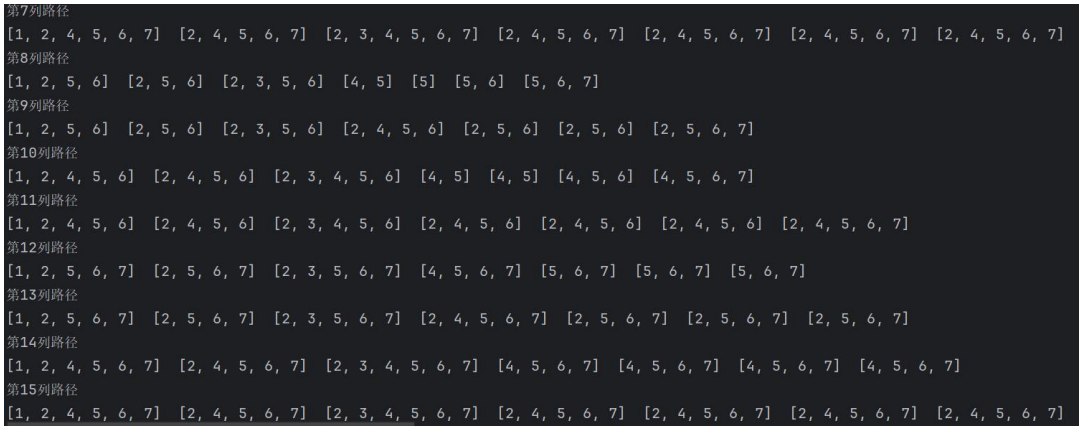
以输入

7 7 4
1 2 3
2 3 2
4 3 9
2 6 2
4 5 3
6 5 2
7 6 4
2 4 7 5

为例：

运行结果为：

```
斯坦纳树的权重为：11
斯坦纳树的路径为：[2, 4, 5, 6, 7]
路径矩阵为：
第0列路径
[]  []  []  []  []  []  []
第1列路径
[1, 2]  [2]  [2, 3]  [2, 4, 5, 6]  [2, 5, 6]  [2, 6]  [2, 6, 7]
第2列路径
[1, 2, 4, 5, 6]  [2, 4, 5, 6]  [3, 4]  [4]  [4, 5]  [4, 5, 6]  [4, 5, 6, 7]
第3列路径
[1, 2, 4, 5, 6]  [2, 4, 5, 6]  [2, 3, 4, 5, 6]  [2, 4, 5, 6]  [2, 4, 5, 6]  [2, 4, 5, 6]  [2, 4, 5, 6, 7]
第4列路径
[1, 2, 6, 7]  [2, 6, 7]  [2, 3, 6, 7]  [4, 5, 6, 7]  [5, 6, 7]  [6, 7]  [7]
第5列路径
[1, 2, 6, 7]  [2, 6, 7]  [2, 3, 6, 7]  [2, 4, 5, 6, 7]  [2, 5, 6, 7]  [2, 6, 7]  [2, 6, 7]
第6列路径
[1, 2, 4, 5, 6, 7]  [2, 4, 5, 6, 7]  [2, 3, 4, 5, 6, 7]  [4, 5, 6, 7]  [4, 5, 6, 7]  [4, 5, 6, 7]  [4, 5, 6, 7]
```



然后处理数据，对 api，key 编号以及统计 api 合作次数，还有获取 key 到 api 的映射关系，将优化后的算法一部分作为初始化内容，一部分嵌入循环体，对多种 api 组合执行算法计算并将路径和权重结果存入优先队列。

完成数据处理之后对算法进行结构重组：

首先输入读入关键词，然后利用关键词在 KeyToAPIs.csv 中找到每个关键词对应的 API 组合，一共 k 组。

然后利用 corporation.csv 初始化 pow 矩阵，且到自身的权重为 0，pow[u][v]的的值是合作次数经过 weight 函数计算之后的值，不相连的两点权重为 Integer.MAX_VALUE。接着创建 double 的二维数组存储 dp 值，以及 HashSet<Integer>的二维数组存储节点。

接下来利用递归迭代的方法每次从 k 组对应的 API 组合选出一个进行组合作为关键节点执行循环体也就是算法，并且将执行结果的最小斯坦纳树的路径结点和权重作为一个 Pair 存放在按照 Pair 的权重升序的优先队列中。

最后输出优先队列的前五条记录作为 top10 推荐 api。

循环体包括以下内容：初始化 dp 数组为最大整数以及 以输入的关键点为根，二进制中对应关键节点位置为 1 的 dp 总权重为 0；初始化 routes 数组关键节点的 hashSet 中存放关键结点本身。使用状态压缩 DP 求解最小路径和，使用 Dijkstra 算法求解最短路径，将 routes[y][(1 << k) - 1]，dp[y][(1 << k) - 1]作为 Pair 存放在优先队列中。

四、功能模块源代码结构介绍：

一共 7 个部分

1.初始化部分

利用 corporation.csv 初始化 pow 矩阵，且到自身的权重为 0，pow[u][v]的的值是合作次数经过 weight 函数计算之后的值，不相连的两点权重为 Integer.MAX_VALUE。接着创建 double 的二维数组存储 dp 值，以及 HashSet<Integer>的二维数组存储节点。

2.调用循环体

将 k 个属性转换成 n1*n2*...*nK 个 hashSet 数组（使用 hashSet 是为了应对不同属性被同一个 api 包括的情况，hashSet 可以去除重复），利用递归迭代的方法每次从 k 组对应的 API 组合选出一个进行组合作为关键节点执行循环体也就是算法，并且将执行结果的最小斯坦纳树的路径结点和权重作为一个 Pair 存放在按照 Pair 的权重升序的优先队列中。

3.输出结果

输出优先队列的前五条记录作为 top5 推荐 api。

4.循环体的函数

初始化 dp 数组为最大整数以及 以输入的关键点为根，二进制中对应关键节点位置为 1 的 dp 总权重为 0；初始化 routes 数组关键节点的 hashSet 中存放关键结点本身。对每种状态 s，遍历 i 进行状态转移同时更新 route 记录结点，调用 deal 函数，将 routes[y][(1 << k) - 1]，dp[y][(1 << k) - 1]作为 Pair 存放在优先队列中。

5.deal 函数

通过 Dijkstra 算法用更新状态 s 列，以及更新 dp 值。

6.weight 函数

作为权值计算方案，这里使用了取倒数。

7.Pair 类

将 HashSet<Integer>和 Double 组合以便存入优先队列。

```
import java.util.*;
import com.opencsv.CSVReader;
import com.opencsv.CSVReaderBuilder;
import com.opencsv.exceptions.CsvValidationException;
import java.io.FileReader;
import java.io.IOException;

public class method1 {
    public static void main(String[] args) {
//下面是初始化的内容：
        String csvFilePath = "C:\\JavaProjects\\KeysToAPIs\\src\\singleKeyToAPIs.csv";
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入关键词个数：");
        int k = sc.nextInt();
        int [] keys = new int[k];//关键词数组
        LinkedHashSet<Integer>[] apiGroups = new LinkedHashSet[k];//与关键词数组对应的 Hashset
        for (int i = 0; i < k; i++) {
            apiGroups[i] = new LinkedHashSet<>();
        }
        System.out.println("请输入用整数表示的关键词：");
        for(int i=0;i<k;i++) {
            keys[i] = sc.nextInt();//读取 keys[i]
            //初始化与 keys[i]关键词对应的 Hashset
            int rowsToSkip = keys[i];//跳过 keys[i]行
            try (CSVReader reader = new CSVReaderBuilder(new FileReader(csvFilePath)).build()) {
                // 跳过 keys[i]行
                for (int j = 0; j < rowsToSkip; j++) {
                    reader.readNext();
                }
                // 读取后续行的数据
                String[] row = reader.readNext();
                // 添加元素到 apiGroups
```

```
            for (int ix = 2; ix < row.length; ix++) {
                int value = Integer.parseInt(row[ix]);
                apiGroups[i].add(value);
            }
        } catch (IOException | CsvValidationException e) {
            throw new RuntimeException(e);
        }
    }
    //输出 apiGroups
    System.out.println("apiGroups 打印查看");
    for(int p=0;p<k;p++){
        System.out.println("关键词"+keys[p]+"对应的 api 组为：");
        Iterator<Integer> iterator1 = apiGroups[p].iterator();
        while(iterator1.hasNext()){
            System.out.println(iterator1.next());
        }
    }
    //数量定义
    //Scanner sc = new Scanner(System.in);
    int n = 552;//结点个数
    double[][] pow = new double[n][n]; // 用于存储节点之间的权重，表达无向图
    // 初始化 pow 数组
    for (int i = 0; i < n; i++) {
        Arrays.fill(pow[i], Integer.MAX_VALUE); // 初始化距离为最大值
        pow[i][i] = 0; // 节点到自身的距离为 0
    }
    String csvFilePath2 = "C:\\JavaProjects\\KeysToAPIs\\src\\corporation.csv";
    //合作次数的条数
    int m = 1457;//边的个数
    try (CSVReader reader = new CSVReaderBuilder(new FileReader(csvFilePath2)).build()) {
        reader.readNext();//跳过表头
        // 初始化 pow 数组
        for (int i = 0; i < m; i++) {
            String[] row = reader.readNext();
            int u = Integer.parseInt(row[0])-1,v=Integer.parseInt(row[1])-1;
            double w= weight(Integer.parseInt(row[2]));
            // 采用邻接矩阵存储结果
            pow[u][v] = Math.min(pow[u][v], w);
            pow[v][u] = Math.min(pow[v][u], w);
        }
    } catch (IOException | CsvValidationException e) {
        throw new RuntimeException(e);
    }
    //创建 dp 数组
```

```
        double[][] dp = new double[n][1 << k]; // 存储状态压缩 DP 的结果
        //创建 routes 数组
        @SuppressWarnings("unchecked")
        HashSet<Integer>[][] routes = new HashSet[n][1 << k];// 创建一个二维数组，每个元素都是一个 HashSet<Integer>，
用来存储路径上的结点
//上面是初始化的内容：

//下面是调用循环体的内容：
        //定义优先队列，存放路径方案和对应权重，按照权重升序排列
        PriorityQueue<Pair> priorityQueue = new PriorityQueue<>(Comparator.comparingDouble(Pair::getSecond));
        // 用于存储组合的结果
        LinkedHashSet<Integer> currentCombination = new LinkedHashSet<>();
        // 开始递归遍历
        recursiveLoop(apiGroups, 0,currentCombination,n,k,pow,dp,routes,priorityQueue);
//上面是调用循环体的内容：

//下面是输出结果的内容：
        //定义输出的 topn 数量 S
        int S = 10;
        int count = 0;
        System.out.println("权重最小的前"+S+"组 api 推荐");
        System.out.println("格式为[api1,api2,...,apin]：权重");
        while (!priorityQueue.isEmpty() && count < S) {
            Pair pair = priorityQueue.poll();
            System.out.println(pair.getFirst() + " : " + pair.getSecond());
            count++;
        }
//上面是输出结果的内容：
    }
//下面是循环体的内容：
    private static void recursiveLoop(LinkedHashSet<Integer>[] apiGroups, int
currentIndex,LinkedHashSet<Integer> currentCombination,int n,int k,double[][] pow,double[][]
dp,HashSet<Integer>[][] routes,PriorityQueue<Pair> priorityQueue) {
        // 如果当前索引已经超过数组的长度，说明已经完成了一种组合，输出结果
        if (currentIndex == apiGroups.length) {
            //输出此时的组合情况：
            System.out.println("此时的关键结点为：");
            Iterator<Integer> iterator0 = currentCombination.iterator();
            while(iterator0.hasNext()){
                System.out.println(iterator0.next());
            }
            // 初始化 dp 数组
            for (int i = 0; i < n; i++) {
                Arrays.fill(dp[i], Integer.MAX_VALUE); // 初始化状态 DP 为最大值
```

```
            }
            // 初始化 routes 数组，为每个元素创建一个新的 HashSet
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < 1 << k; j++) {
                    routes[i][j] = new HashSet<>();
                }
            }
            //读取所有关键节点，初始化 dp 数组和 routes 数组
            int y = -1;// 用来记录最后一个关键点
            // 标记关键节点，并初始化状态关键节点的 DP 值和 routes 为结点本身
            Iterator<Integer> iterator = currentCombination.iterator();
            for (int i = 0; i < k; i++) {
                int x = iterator.next() - 1;
                dp[x][1 << i] = 0; // 以输入的关键点为根，二进制中对应关键节点位置为 1 的 dp 总权重为 0
                routes[x][1 << i].add(x + 1);
                y = x; // 记录最后一个关键节点
            }
            // 使用状态压缩 DP 求解最小路径和
            // 遍历从单个节点开始，直到满足所有关键节点都连同的所有可能状态
            for (int s = 1; s < 1 << k; s++) {
                // 遍历以节点 i 为根节点，状态为 s 时的情况
                for (int i = 0; i < n; i++) {
                    // 遍历 s 二进制的所有子集，执行 i 的度数大于 1 时的操作
                    // 更新 routes 里的路径
                    for (int t = s & (s - 1); t > 0; t = (t - 1) & s) {
                        //dp[i][s] = Math.min(dp[i][s], dp[i][t] + dp[i][s ^ t]);
                        if (dp[i][t] + dp[i][s ^ t] < dp[i][s]) {
                            dp[i][s] = dp[i][t] + dp[i][s ^ t];
                            routes[i][s].addAll(routes[i][t]);
                            routes[i][s].addAll(routes[i][s ^ t]);
                        }
                    }
                }
                // 处理状态为 s 的情况下最短路径
                deal(s, n, dp, pow, routes);
            }
            System.out.print("斯坦纳树的权重为：");
            System.out.println(dp[y][(1 << k) - 1]); // 输出结果
            System.out.println("路径矩阵为：");
            for (int j = 0; j < 1<<k ; j++) {
                System.out.println("第" + j + "列路径");
                for (int i = 0; i < n; i++) {
                    System.out.print(routes[i][j] + "  ");
                }
```

```
                System.out.println();
            }
            System.out.print("斯坦纳树的路径为：");
            System.out.println(routes[y][(1 << k) - 1]);
            priorityQueue.add(new Pair(routes[y][(1 << k) - 1], dp[y][(1 << k) - 1]));
            return;
        }
        // 遍历当前 LinkedHashSet 中的元素，递归地处理下一个 LinkedHashSet
        for (Integer number : apiGroups[currentIndex]) {
            // 添加当前元素到当前组合
            currentCombination.add(number);
            // 递归处理下一个 LinkedHashSet
            recursiveLoop(apiGroups, currentIndex + 1, currentCombination, n, k, pow, dp, routes, priorityQueue);
            // 回溯，移除当前元素，以便尝试其他组合方式
            currentCombination.remove(number);
        }
    }
//上面是循环体的内容：
//下面是 deal 函数：
    private static void deal(int s, int n, double[][] dp, double[][] pow, HashSet<Integer>[][] routes) {
        // 建立优先队列，按照 dp 值升序排列
        PriorityQueue<double[]> pq = new PriorityQueue<>(Comparator.comparingDouble(o -> o[1]));
        boolean[] vis = new boolean[n];
        for (int i = 0; i < n; i++) {
            if (dp[i][s] != Integer.MAX_VALUE) {
                pq.add(new double[]{i, dp[i][s]});
            }
        }
        // 使用 Dijkstra 算法求解最短路径，即求解以每个节点为顶点，满足状态 s（包含对应的点）的权重和的最小值
        while (!pq.isEmpty()) {
            double[] tmp = pq.poll();//删除队列首元素并保存在 tmp 中
            // 如果该行表示的点已经访问，则跳过
            if (vis[(int) tmp[0]]) {
                continue;
            }
            //否则进行访问并设置为已访问
            vis[(int) tmp[0]] = true;
            //遍历所有与 tmp 对应点直接连接的点
            for (int i = 0; i < n; i++) {
                // 跳过所有不与 tmp 对应点相连的点和该点本身
                if (i == tmp[0] || pow[(int) tmp[0]][i] == Integer.MAX_VALUE) {
                    continue;
                }
                if (tmp[1] + pow[(int) tmp[0]][i] < dp[i][s]) {
```

```
                    //如果遍历到的点的 dp 值可以更新为更小的由 tmp 的 dp 值+两点间权重就更新其 dp 和 routes
                    dp[i][s] = (tmp[1] + pow[(int) tmp[0]][i]);
                    routes[i][s].clear();
                    routes[i][s].addAll(routes[(int)tmp[0]][s]);
                    routes[i][s].add(i+1);
                    pq.add(new double[]{i, dp[i][s]});
                }
            }
        }
    }
//上面是 deal 函数：
//下面是 weight 函数：
    private static double weight(int x){
        return  1/(double)x;
    }
//上面是 weight 函数：
}
//下面是 Pair 类
class Pair {
    private HashSet<Integer> first;
    private double second;
    public Pair(HashSet<Integer> first, double second) {
        this.first = new HashSet<>(first);  // 使用 HashSet 的拷贝以确保独立性
        this.second = second;
    }
    public HashSet<Integer> getFirst() {
        return new HashSet<>(first);  // 返回 HashSet 的拷贝以确保独立性
    }
    public double getSecond() {
        return second;
    }
}
//上面是 Pair 类
```

随机输入关键词进行测试运行结果部分截图：

```
第1列路径
[192, 1, 2, 522, 541, 127] [192, 2, 522, 541, 127] [192, 3, 277, 541, 127] [] [] [] [192, 7, 187] [192, 148, 8, 541, 127] [192, 9, 541,
第2列路径
[1, 2, 470, 472, 522, 541] [2, 470, 472, 522, 541] [3, 277, 470, 472] [] [] [] [209, 470, 7, 472, 187] [148, 470, 472, 8] [470, 472, 9,
第3列路径
[192, 1, 2, 470, 472, 522, 541, 127] [192, 2, 470, 472, 522, 541, 127] [192, 3, 277, 470, 472, 541, 127] [] [] [] [192, 209, 470, 7, 472,
```

可以发现路径矩阵中存在较多的空值路径，通过对 mashup.csv 的 SQL 分析，发现其中大部分都是孤立点，即与其它 api 合作次数为 0 的点，所以通过随机输入进行测试，很大概率出现路径为空的情况，因为当两点距离为整数最大值时会跳过对 dp 和 routes 的更新。

以下是 SQL 分析语句和分析结果：

```
SELECT count(*) as num1 from mashup

SELECT count(*) as num2 from mashup
where rapis not like '%,%'
```

num1	num2
5954	3185

53.5%的 api 都是独立出现的，所以当输入的关键词越多，出现空路径的概率 P 就越大，且通过数学分析可以得知 $P=1-(\frac{1}{2})^k$，当 k=3 时，P 的大小就为 87.5%。所以必须通过 mashup 数据集中的关键词组合进行测试。

六、测试方案：

以上代码的 k 需要手动输入，不方便测试，于是考虑通过自动读取来进行多组测试，对以上代码进行修改使其读取改进后的 mashup2.csv 文件(所有 api 和 key 都用其对应的编号表示)中的全部 key 关键词作为一次输入，通过比较前 10 条输出路径中的 api 编号和 mashup2.csv 中 key 对应的 api 编号是否吻合统计 Hit 次数，并且计算命中次数与 key 输入次数的比值作为 Hit Rate。

此外在代码首位添加时间记录函数

```
//毫秒为为单位
long startTime=System.currentTimeMillis(); //获取开始时间
long endTime=System.currentTimeMillis(); //获取结束时间
System.out.println("时间开销为： "+(endTime-startTime)+"ms");
//纳秒为为单位
 long startTime=System.nanoTime();  //获取开始时间
 long endTime=System.nanoTime(); //获取结束时间
System.out.println("时间开销为： "+(endTime-startTime)+"ns");
```

记录程序运行时间。

代码修改方案：

通过读取 mashup2.csv 的第三列整数 keyNums 作为输入的 key，第二列整数 apiNums 作为保存的正确结果，每次读取一列 keyNums，解析出个数 k，数字分解到数组中，然后作为输入，最后将输出的 top10 记录的路径分别分解开与保存的正确结果作比对，查看是否有一致的，一旦有一组路径和正确结果一致就将命中次数 count 加 1，所有的 keynums 输入完之后，用 count/keyNums 组数得出命中率。

```
import java.util.*;
import com.opencsv.CSVReader;
import com.opencsv.CSVReaderBuilder;
import com.opencsv.exceptions.CsvValidationException;
import java.io.FileReader;
import java.io.IOException;
```

```java
public class method1ForTest {
    public static void main(String[] args) {
//下面是初始化的内容：
        String csvFilePath = "C:\\JavaProjects\\KeysToAPIs\\src\\singleKeyToAPIs.csv";
        Scanner sc = new Scanner(System.in);
        //数量定义
        //Scanner sc = new Scanner(System.in);
        int n = 552;//结点个数
        double[][] pow = new double[n][n]; // 用于存储节点之间的权重，表达无向图
        // 初始化 pow 数组
        for (int i = 0; i < n; i++) {
            Arrays.fill(pow[i], Integer.MAX_VALUE); // 初始化距离为最大值
            pow[i][i] = 0; // 节点到自身的距离为 0
        }
        //开始计时
        long startTime=System.currentTimeMillis();
        String csvFilePath2 = "C:\\JavaProjects\\KeysToAPIs\\src\\corporation.csv";
        //合作次数的条数
        int m = 1457;//边的个数
        try (CSVReader reader = new CSVReaderBuilder(new FileReader(csvFilePath2)).build()) {
            reader.readNext();//跳过表头
            // 初始化 pow 数组
            for (int i = 0; i < m; i++) {
                String[] row = reader.readNext();
                int u = Integer.parseInt(row[0])-1,v=Integer.parseInt(row[1])-1;
                double w= weight(Integer.parseInt(row[2]));
                // 采用邻接矩阵存储结果
                pow[u][v] = Math.min(pow[u][v], w);
                pow[v][u] = Math.min(pow[v][u], w);
            }
        } catch (IOException | CsvValidationException e) {
            throw new RuntimeException(e);
        }
        //大循环即将开始
        //定义命中次数
        int hitTimes=0;
        int circulation=5954;
        String csvFilePath1 = "C:\\JavaProjects\\KeysToAPIs\\src\\mashup2.csv";
        try (CSVReader reader1 = new CSVReaderBuilder(new FileReader(csvFilePath1)).build()) {
            reader1.readNext();//跳过表头
            //定义大循环次数
            //int circulation=5954;
            for(int ci = 0;ci<circulation;ci++){
```

```
                String[] row = reader1.readNext();
                //解析第 2 列 apis 作为 answer 数组
                String[] answersAsString = row[1].split(",");  // 使用逗号分割字符串
                int[] answers = new int[answersAsString.length];  // 创建整数数组，大小为分割后的字符串数组长度
                for (int i = 0; i < answersAsString.length; i++) {
                    answers[i] = Integer.parseInt(answersAsString[i]);
                }
                //解析第 3 列 keys 作为输入的数组
                String[] keysAsString = row[2].split(",");  // 使用逗号分割字符串
                int[] keys = new int[keysAsString.length];  // 创建整数数组，大小为分割后的字符串数组长度
                for (int i = 0; i < keysAsString.length; i++) {
                    keys[i] = Integer.parseInt(keysAsString[i]);
                }
                int k = keysAsString.length;
                LinkedHashSet<Integer>[] apiGroups = new LinkedHashSet[k];//与关键词数组对应的 Hashset
                //读取 keys[i]
                for(int i=0;i<k;i++) {
                    apiGroups[i] = new LinkedHashSet<>();
                    //初始化与 keys[i]关键词对应的 Hashset
                    int rowsToSkip = keys[i];//跳过 keys[i]行
                    try (CSVReader reader = new CSVReaderBuilder(new FileReader(csvFilePath)).build()) {
                        // 跳过 keys[i]行
                        for (int j = 0; j < rowsToSkip; j++) {
                            reader.readNext();
                        }
                        // 读取后续行的数据
                        String[] row2 = reader.readNext();
                        // 添加元素到 apiGroups
                        for (int ix = 2; ix < row2.length; ix++) {
                            int value = Integer.parseInt(row2[ix]);
                            apiGroups[i].add(value);
                        }
                    } catch (IOException | CsvValidationException e) {
                        throw new RuntimeException(e);
                    }
                }
                //输出 apiGroups
//                System.out.println("apiGroups 打印查看");
//                for(int p=0;p<k;p++){
//                    System.out.println("关键词"+keys[p]+"对应的 api 组为：");
//                    Iterator<Integer> iterator1 = apiGroups[p].iterator();
//                    while(iterator1.hasNext()){
//                        System.out.println(iterator1.next());
//                    }
```

```
//                }
                //创建 dp 数组
                double[][] dp = new double[n][1 << k]; // 存储状态压缩 DP 的结果
                //创建 routes 数组
                @SuppressWarnings("unchecked")
                HashSet<Integer>[][] routes = new HashSet[n][1 << k];// 创建一个二维数组，每个元素都是一个
HashSet<Integer>,用来存储路径上的结点
//上面是初始化的内容：
//下面是调用循环体的内容：
                //定义优先队列，存放路径方案和对应权重，按照权重升序排列
                PriorityQueue<Pair> priorityQueue = new
PriorityQueue<>(Comparator.comparingDouble(Pair::getSecond));
                // 用于存储组合的结果
                LinkedHashSet<Integer> currentCombination = new LinkedHashSet<>();
                // 开始递归遍历
                recursiveLoop(apiGroups, 0,currentCombination,n,k,pow,dp,routes,priorityQueue);
//上面是调用循环体的内容：
                //判断是否发生命中
                //定义输出的 topn 数量 S
                int S = 10;
                int count = 0;
                System.out.println("权重最小的前"+S+"组 api 推荐");
                System.out.println("格式为[api1,api2,...,apin]：权重");
                while (!priorityQueue.isEmpty() && count < S) {
                    Pair pair = priorityQueue.poll();
                    HashSet<Integer> hashSet =pair.getFirst();
                    // 创建整数数组，大小为 HashSet 的大小
                    Integer[] array = new Integer[hashSet.size()];
                    // 将 HashSet 中的元素复制到数组中
                    hashSet.toArray(array);
                    // 将 Integer 数组转换为 int 数组
                    int[] intArray = Arrays.stream(array).mapToInt(Integer::intValue).toArray();
                    //命中的定义：每一个 intArray 元素都与 answers 数组中某一个元素相同。
                    int num = 0;
                    for (int i : intArray) {
                        for (int j : answers) {
                            if (i == j) {
                                num++;
                                break;  // 找到匹配元素后跳出内层循环
                            }
                        }
                    }
                    //所有的都一致表示命中，增加命中次数,不再继续比较优先队列其他元素
                    if(num==hashSet.size()){
```

```
                        hitTimes++;
                        break;
                    }
                    count++;
                }
                //上面判断是否发生命中
            }
        } catch (IOException | CsvValidationException e) {
            throw new RuntimeException(e);
        }
        //结束计时
        long endTime=System.currentTimeMillis();
//下面是计算时间开销的内容
        System.out.println("时间开销为："+(endTime-startTime)+"ms");
//上面是计算时间开销的内容
//下面是计算命中率的内容
        System.out.println("命中率为："+(double)hitTimes/(double)circulation);
//上面是计算命中率的内容
    }
    //下面是循环体的内容：
    private static void recursiveLoop(LinkedHashSet<Integer>[] apiGroups, int
currentIndex,LinkedHashSet<Integer> currentCombination,int n,int k,double[][] pow,double[][]
dp,HashSet<Integer>[][] routes,PriorityQueue<Pair> priorityQueue) {
        // 如果当前索引已经超过数组的长度，说明已经完成了一种组合，输出结果
        if (currentIndex == apiGroups.length) {
            //输出此时的组合情况：
            System.out.println("此时的关键结点为：");
            Iterator<Integer> iterator0 = currentCombination.iterator();
            while(iterator0.hasNext()){
                System.out.println(iterator0.next());
            }
            // 初始化 dp 数组
            for (int i = 0; i < n; i++) {
                Arrays.fill(dp[i], Integer.MAX_VALUE); // 初始化状态 DP 为最大值
            }
            // 初始化 routes 数组，为每个元素创建一个新的 HashSet
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < 1 << k; j++) {
                    routes[i][j] = new HashSet<>();
                }
            }
            //读取所有关键节点，初始化 dp 数组和 routes 数组
            int y = -1;// 用来记录最后一个关键点
            // 标记关键节点，并初始化状态关键节点的 DP 值和 routes 为结点本身
```

```
        Iterator<Integer> iterator = currentCombination.iterator();
        for (int i = 0; i < k; i++) {
            int x = iterator.next() - 1;
            dp[x][1 << i] = 0; // 以输入的关键点为根，二进制中对应关键节点位置为 1 的 dp 总权重为 0
            routes[x][1 << i].add(x + 1);
            y = x; // 记录最后一个关键节点
        }
        // 使用状态压缩 DP 求解最小路径和
        // 遍历从单个节点开始，直到满足所有关键节点都连同的所有可能状态
        for (int s = 1; s < 1 << k; s++) {
            // 遍历以节点 i 为根节点，状态为 s 时的情况
            for (int i = 0; i < n; i++) {
                // 遍历 s 二进制的所有子集，执行 i 的度数大于 1 时的操作
                // 更新 routes 里的路径
                for (int t = s & (s - 1); t > 0; t = (t - 1) & s) {
                    //dp[i][s] = Math.min(dp[i][s], dp[i][t] + dp[i][s ^ t]);
                    if (dp[i][t] + dp[i][s ^ t] < dp[i][s]) {
                        dp[i][s] = dp[i][t] + dp[i][s ^ t];
                        routes[i][s].addAll(routes[i][t]);
                        routes[i][s].addAll(routes[i][s ^ t]);
                    }
                }
            }
            // 处理状态为 s 的情况下最短路径
            deal(s, n, dp, pow, routes);
        }
        System.out.print("斯坦纳树的权重为：");
        System.out.println(dp[y][(1 << k) - 1]); // 输出结果
        System.out.println("路径矩阵为：");
        for (int j = 0; j < 1<<k ; j++) {
            System.out.println("第" + j + "列路径");
            for (int i = 0; i < n; i++) {
                System.out.print(routes[i][j] + "  ");
            }
            System.out.println();
        }
        System.out.print("斯坦纳树的路径为：");
        System.out.println(routes[y][(1 << k) - 1]);
        priorityQueue.add(new Pair(routes[y][(1 << k) - 1], dp[y][(1 << k) - 1]));
        return;
    }
    // 遍历当前 LinkedHashSet 中的元素，递归地处理下一个 LinkedHashSet
    for (Integer number : apiGroups[currentIndex]) {
        // 添加当前元素到当前组合
```

```
            currentCombination.add(number);
            // 递归处理下一个 LinkedHashSet
            recursiveLoop(apiGroups, currentIndex + 1, currentCombination, n, k, pow, dp, routes, priorityQueue);
            // 回溯，移除当前元素，以便尝试其他组合方式
            currentCombination.remove(number);
        }
    }
//上面是循环体的内容：
    //下面是 deal 函数：
    private static void deal(int s, int n, double[][] dp, double[][] pow, HashSet<Integer>[][] routes) {
        // 建立优先队列，按照 dp 值升序排列
        PriorityQueue<double[]> pq = new PriorityQueue<>(Comparator.comparingDouble(o -> o[1]));
        boolean[] vis = new boolean[n];
        for (int i = 0; i < n; i++) {
            if (dp[i][s] != Integer.MAX_VALUE) {
                pq.add(new double[]{i, dp[i][s]});
            }
        }
        // 使用 Dijkstra 算法求解最短路径，即求解以每个节点为顶点，满足状态 s（包含对应的点）的权重和的最小值
        while (!pq.isEmpty()) {
            double[] tmp = pq.poll();//删除队列首元素并保存在 tmp 中
            // 如果该行表示的点已经访问，则跳过
            if (vis[(int) tmp[0]]) {
                continue;
            }
            //否则进行访问并设置为已访问
            vis[(int) tmp[0]] = true;
            //遍历所有与 tmp 对应点直接连接的点
            for (int i = 0; i < n; i++) {
                // 跳过所有不与 tmp 对应点相连的点和该点本身
                if (i == tmp[0] || pow[(int) tmp[0]][i] == Integer.MAX_VALUE) {
                    continue;
                }
                if (tmp[1] + pow[(int) tmp[0]][i] < dp[i][s]) {
                    //如果遍历到的点的 dp 值可以更新为更小的由 tmp 的 dp 值+两点间权重就更新其 dp 和 routes
                    dp[i][s] = (tmp[1] + pow[(int) tmp[0]][i]);
                    routes[i][s].clear();
                    routes[i][s].addAll(routes[(int)tmp[0]][s]);
                    routes[i][s].add(i+1);
                    pq.add(new double[]{i, dp[i][s]});
                }
            }
        }
    }
```

```
//上面是 deal 函数：
    //下面是 weight 函数：
    private static double weight(int x){
        return  100/(double)x;
    }
//上面是 weight 函数：
}
//下面是 Pair 类
class Pair {
    private HashSet<Integer> first;
    private double second;
    public Pair(HashSet<Integer> first, double second) {
        this.first = new HashSet<>(first);  // 使用 HashSet 的拷贝以确保独立性
        this.second = second;
    }
    public HashSet<Integer> getFirst() {
        return new HashSet<>(first);  // 返回 HashSet 的拷贝以确保独立性
    }
    public double getSecond() {
        return second;
    }
}
//上面是 Pair 类
```

七、改进的算法设计思路：

保持代码的基本结构，仍然为 7 个模板。

1. 在**命中率**指标上通过改进 weight 函数，使其从 1/x 变为-x+100，保持权重与合作次数负相关的基本关系基础上提升了两个性质：①使其值为较大的整数，不易出现因为机器级浮点数的限制导致当小数较小时变为 0。②weight 函数关于 x 的导数为一个常数-1，而不是与 x 相关的变量，这表明 x 的大小不会改变其在负相关性上反向增长的速率，从而提高了准确率。

2.在**时间开销**指标上通过使用 K-CAR 算法进行优化：提高运算速率（待实现）。

八、测试方案：

代码基本结构保持不变，在算法是线上，weight 函数设置上进行调整。

```
    private static double weight(int x){
        return  100-x;
    }
```

测试整个大循环的运行时间作为算法的时间开销；

```
System.out.println("时间开销为："+(endTime-startTime)+"ms");
```

以及命中次数除以整个测试的输入次数作为命中率。

```
  System.out.println("命中率为："+(double)hitTimes/(double)circulation);
```

十、实验结果与分析

代码整体逻辑框架和中间测试都是正确的，最后运行时没有出现正确的结果很有可能是因为表处理的环节出现某一步的错误，这些表的处理是一环套一环的，中间某一个表的处理不合规，遗漏或者没有重新编号都会导致索引异常，此外原始数据的 api 到 key 的对应关系和 mashup 中 api 与 key 的关系的一致性没有得到检验，也是会导致即使过程全部正确也无法得到正确结果的原因。

十三、参考文献：

[1] 王范．关键词驱动的兼容 APIs 推荐系统[D]. 曲阜师范大学,2021.DOI:10.27267/d.cnki.gqfsu.2021.001444

[2] 石雨轩．基于组斯坦纳树的知识图谱搜索算法研究[D]. 南京大学,2021.DOI:10.27235/d.cnki.gnjiu.2021.002007

[3] https://www.yuque.com/chuweijie/kb/yxzbpz7ssrmgw2wr?singleDoc#

附录 A

A.1 过滤 mashup 中没有重新过的 api 得到新的 api 表

```
import org.apache.commons.csv.*;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;
public class CSVReaderWriter {
```

```java
    public static void main(String[] args) {
        readCSV();
        //writeCSV();
    }
    public static void readCSV() {
        // 读取 mashup 表中数据为 list
        List<Mashup> mashups = new ArrayList<>();
        try {
            FileReader reader = new FileReader("D:/my/mashup.csv");
            Iterable<CSVRecord> records = CSVFormat.DEFAULT.withFirstRecordAsHeader().parse(reader);
            for (CSVRecord record : records) {
                // System.out.println(String.format("{}_{}_{}",
record.get("Name"),record.get("rapis"),record.get("keys")));
                String name = record.get("Name");
                String rapis = record.get("rapis");
                String keys = record.get("keys");
                Mashup mashup = new Mashup(name, rapis, keys);
                mashups.add(mashup);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        // 读取 apitable 表中数据为 list
        List<Apitable> apitables = new ArrayList<>();
        try {
            FileReader reader = new FileReader("D:/my/apitable2.csv");
            Iterable<CSVRecord> records = CSVFormat.DEFAULT.withFirstRecordAsHeader().parse(reader);
            for (CSVRecord record : records) {
                System.out.println(String.format("{}_{}_{}",
record.get("num"),record.get("api"),record.get("keys2")));
                String num = record.get("num");
                String api = record.get("api");
                String keys2 = record.get("keys2");
                Apitable apitable = new Apitable(num, api, keys2);
                apitables.add(apitable);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        // 筛选 apitables 里 api 在 mashup 里的
        Set<String> mashupApiSets = mashups.stream().map(Mashup::getRapis).collect(Collectors.toSet());
        List<Apitable> filteredApitables = apitables.stream()
                .filter(apitable -> mashupApiSets.contains(apitable.getApi()))
                .collect(Collectors.toList());
```

```
        //写入文件 new_filtered_apitables.csv
        try {
            FileWriter writer = new FileWriter("D:/my/new_filtered_apitables.csv");
            CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT.withHeader("num", "api","keys2"));
            for (int i = 0;i<filteredApitables.size();i++) {
                // 从 1 开始编号
                csvPrinter.printRecord(i+1,
filteredApitables.get(i).getApi(),filteredApitables.get(i).getKeys2());
            }
            csvPrinter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // 任务 2
        List<Apitable> apitable2 = new ArrayList<>();
        try {
            FileReader reader = new FileReader("D:/my/apitable2.csv");
            Iterable<CSVRecord> records = CSVFormat.DEFAULT.withFirstRecordAsHeader().parse(reader);
            for (CSVRecord record : records) {
                // System.out.println(String.format("{}_{}_{}",
record.get("num"),record.get("api"),record.get("keys2")));
                String num = record.get("num");
                String api = record.get("api");
                String keys2 = record.get("keys2");
                Apitable apitable = new Apitable(num, api, keys2);
                apitable2.add(apitable);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        //将 key 转换成 num
        Map<String, String> groupedApitables = apitable2.stream()
                .collect(Collectors.groupingBy(Apitable::getKeys2,
                        Collectors.mapping(Apitable::getNum,
                        Collectors.joining(","))));
        List<KeyToApi> keyToApis = groupedApitables.entrySet().stream()
                .map(entry -> new KeyToApi(entry.getKey(), entry.getValue()))
                .collect(Collectors.toList());
        //写入文件 new_keytoapi.csv
        try {
            FileWriter writer = new FileWriter("D:/my/new_keytoapi.csv");
            CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT.withHeader("num", "keys2"));
            for (int i = 0;i<keyToApis.size();i++) {
                csvPrinter.printRecord(keyToApis.get(i).getNums(),keyToApis.get(i).getKeys2());
```

```
            }
            csvPrinter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void writeCSV() {
        try {
            FileWriter writer = new FileWriter("yourfile.csv");
            CSVPrinter csvPrinter = new CSVPrinter(writer, CSVFormat.DEFAULT.withHeader("ColumnOne",
"ColumnTwo"));
            csvPrinter.printRecord("Data1", "Data2");
            csvPrinter.printRecord("Data3", "Data4");
            csvPrinter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

A.2 构建关键词到 api 映射的 java 文件

```
import com.opencsv.CSVReader;
import com.opencsv.CSVWriter;
import com.opencsv.exceptions.CsvException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
public class KeyToAPIConverter {
    public static void main(String[] args) {
        // 文件路径
        String inputFilePath = "filtered_apitable2.csv";
        String outputFilePath = "singleKeytoAPIs.csv";
```

```
        try {
            // 读取原始 CSV 文件
            CSVReader reader = new CSVReader(new FileReader(inputFilePath));
            List<String[]> originalRows = reader.readAll();
            // 处理原始数据，构建关键词到 API 的映射
            Map<String, List<String>> keyToAPIsMap = new HashMap<>();
            for (String[] row : originalRows.subList(1, originalRows.size())) {
                String[] keys = row[2].split(",");
                for (String key : keys) {
                    keyToAPIsMap.computeIfAbsent(key, k -> new ArrayList<>()).add(row[0]);
                }
            }
            // 创建新的 CSV 文件并写入数据
            CSVWriter writer = new CSVWriter(new FileWriter(outputFilePath));
            List<String[]> newRows = new ArrayList<>();
            // 添加标题行
            newRows.add(new String[]{"keyNum", "singleKey", "apis"});
            // 遍历关键词到 API 的映射，生成新的行
            int keyNum = 1;
            for (Map.Entry<String, List<String>> entry : keyToAPIsMap.entrySet()) {
                String singleKey = entry.getKey();
                String apis = String.join(",", entry.getValue());
                newRows.add(new String[]{String.valueOf(keyNum), singleKey, apis});
                keyNum++;
            }
            // 写入数据到新的 CSV 文件
            writer.writeAll(newRows);
            writer.close();
            System.out.println("生成成功：" + outputFilePath);
        } catch (IOException | CsvException e) {
            e.printStackTrace();
        }
    }
}
```

A.3　统计不同 api 之间合作次数的 java 文件

```
import java.io.*;
import java.util.*;

public class ApiCorporation {
    public static void main(String[] args) {
```

```
        String mashupFilePath = "C:\\JavaProjects\\KeysToAPIs\\src\\mashup.csv";
        String filteredApiTablePath = "C:\\JavaProjects\\KeysToAPIs\\src\\filtered_apitable2.csv";
        String corporationFilePath = "C:\\JavaProjects\\KeysToAPIs\\src\\corporation.csv";
        // 读取 mashup 表和 filtered_apitable2 表
        List<String[]> mashupData = readCSV(mashupFilePath);
        List<String[]> apiTableData = readCSV(filteredApiTablePath);
        // 创建合作次数的映射表
        Map<String, Integer> corporationMap = createCorporationMap(mashupData);
        // 创建新的 corporation 表
        List<String[]> corporationTable = createCorporationTable(apiTableData, corporationMap);
        // 将 corporation 表写入文件
        writeCSV(corporationFilePath, corporationTable);
    }
    // 读取 CSV 文件
    private static List<String[]> readCSV(String filePath) {
        List<String[]> data = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] row = line.split(",");
                data.add(row);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return data;
    }
    // 创建合作次数的映射表
    private static Map<String, Integer> createCorporationMap(List<String[]> mashupData) {
        Map<String, Integer> corporationMap = new HashMap<>();
        for (String[] row : mashupData) {
            Set<String> apiSet = new HashSet<>();
            // 将 mashup 表中的 api 组合添加到集合中
            for (int i = 1; i < row.length; i++) {
                apiSet.add(row[i]);
            }
            // 计算并更新合作次数
            for (String api1 : apiSet) {
                for (String api2 : apiSet) {
                    if (!api1.equals(api2)) {
                        String key = api1 + "," + api2;
                        int count = corporationMap.getOrDefault(key, 0);
                        corporationMap.put(key, count + 1);
                    }
```

```
                }
            }
        }
        return corporationMap;
    }
    // 创建新的 corporation 表
    private static List<String[]> createCorporationTable(List<String[]> apiTableData, Map<String, Integer>
corporationMap) {
        List<String[]> corporationTable = new ArrayList<>();
        // 添加表头
        corporationTable.add(new String[]{"apinum1", "apinum2", "coNum"});
        for (int i = 0; i < apiTableData.size(); i++) {
            String[] row1 = apiTableData.get(i);
            for (int j = i + 1; j < apiTableData.size(); j++) {
                String[] row2 = apiTableData.get(j);
                String api1 = row1[1];
                String api2 = row2[1];
                String key = api1 + "," + api2;
                if (corporationMap.containsKey(key)) {
                    int coNum = corporationMap.get(key);
                    String[] newRow = {row1[0], row2[0], String.valueOf(coNum)};
                    corporationTable.add(newRow);
                }
            }
        }
        return corporationTable;
    }
    // 写入 CSV 文件
    private static void writeCSV(String filePath, List<String[]> data) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            for (String[] row : data) {
                writer.write(String.join(",", row));
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

A.4 将 mashup 中的 api 和 key 转换成对应编号

```
import com.opencsv.CSVReader;
```

```java
import com.opencsv.CSVWriter;
import com.opencsv.exceptions.CsvException;

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;
public class MashupConverter {
    public static void main(String[] args) {
        String mashupFilePath = "C:\\JavaProjects\\KeysToAPIs\\src\\mashup.csv";
        String filteredApiTablePath = "C:\\JavaProjects\\KeysToAPIs\\src\\filtered_apitable2.csv";
        String singleKeyToAPIsPath = "C:\\JavaProjects\\KeysToAPIs\\src\\singleKeyToAPIs.csv";
        String mashup2FilePath = "C:\\JavaProjects\\KeysToAPIs\\src\\mashup2.csv";
        try {
            // 读取 filtered_apitable2.csv 构建 api 名字和编号的映射
            Map<String, String> apiNameToNumberMap = readMapping(filteredApiTablePath);
            // 读取 singleKeytoAPIs.csv 构建 key 名字和编号的映射
            Map<String, String> keyNameToNumberMap = readMapping(singleKeyToAPIsPath);
            // 处理 mashup.csv 文件，将 api 名字和 key 名字转换为编号
            processMashupFile(mashupFilePath, apiNameToNumberMap, keyNameToNumberMap, mashup2FilePath);
            System.out.println("转换成功：" + mashup2FilePath);
        } catch (IOException | CsvException e) {
            e.printStackTrace();
        }
    }
    private static Map<String, String> readMapping(String filePath) throws IOException {
        Map<String, String> mapping = new HashMap<>();
        try (CSVReader reader = new CSVReader(new FileReader(filePath))) {
            List<String[]> rows = reader.readAll();
            for (String[] row : rows.subList(1, rows.size())) {
                mapping.put(row[1], row[0]);
            }
        } catch (CsvException e) {
            throw new RuntimeException(e);
        }
        return mapping;
    }
    private static void processMashupFile(String inputFilePath, Map<String, String> apiMapping,
                                          Map<String, String> keyMapping, String outputFilePath) throws
IOException, CsvException {
        try (CSVReader reader = new CSVReader(new FileReader(inputFilePath));
             CSVWriter writer = new CSVWriter(new FileWriter(outputFilePath))) {
            List<String[]> rows = reader.readAll();
            List<String[]> newRows = new LinkedList<>(rows.subList(0, 1));
```

```
            for (String[] row : new ArrayList<>(rows.subList(1, rows.size()))) {
                String[] newRow = new String[row.length];
                newRow[0] = row[0]; // 保持第一列不变
// 转换第二列的 api 名字为编号
                String[] apiNames = row[1].split(",");
                StringBuilder apiNumbers = new StringBuilder();
                for (String apiName : apiNames) {
                    if (apiMapping.containsKey(apiName)) {
                        apiNumbers.append(apiMapping.get(apiName)).append(",");
                    }
                }
                newRow[1] = apiNumbers.toString();
// 转换第三列的 key 名字为编号
                String[] keyNames = row[2].split(",");
                StringBuilder keyNumbers = new StringBuilder();
                for (String keyName : keyNames) {
                    if (keyMapping.containsKey(keyName)) {
                        keyNumbers.append(keyMapping.get(keyName)).append(",");
                    }
                }
                newRow[2] = keyNumbers.toString();
                newRows.add(newRow);
            }
            // 写入新的 CSV 文件
            writer.writeAll(newRows);
        }
    }
}
```