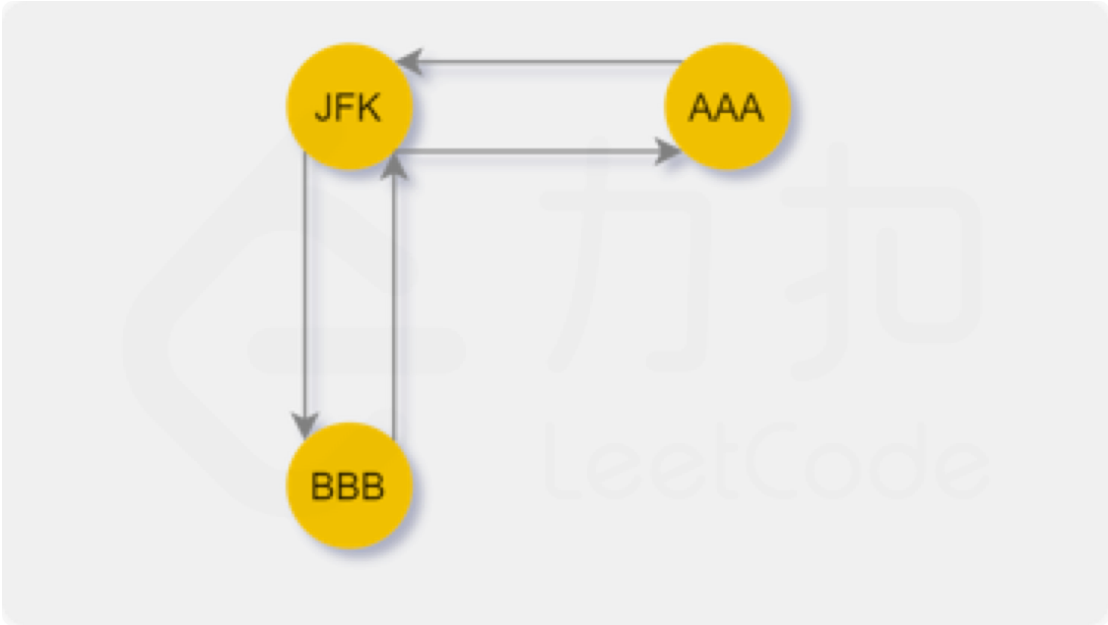
其他项目 **欧拉回路/通路问题**

路径图的深度优先搜索

引言

1、上图说明什么是欧拉路径



> 这种「一笔画」问题与欧拉图或者半欧拉图有着紧密的联系，下面给出定义：
>
> 通过图中所有边恰好一次且行遍所有顶点的通路称为欧拉通路。
>
> 通过图中所有边恰好一次且行遍所有顶点的回路称为欧拉回路。
>
> 具有欧拉回路的无向图称为欧拉图。
>
> 具有欧拉通路但不具有欧拉回路的无向图称为半欧拉图（所以做路径分析，尽量需要采集数据成为有向边，这样逻辑才明确）。
>
> 因为本题保证至少存在一种合理的路径（可能不存在最优解，AAA，BBB可能都可以作为优先解，如果看成买东西，可能AAA，BBB都可以作为优先购买，不存在那个产品更重要），也就告诉了我们，这张图是一个欧拉图或者半欧拉图。我们只需要输出这条欧拉通路的路径即可。

2、题目大背景是在诸多机票中做出排序，找到最合理的排序路径（如图就是JFK飞BBB，BBB再飞JFK，JFK飞AAA，AAA再飞JFK，显然在成环路径中，环路的中心点是需要平台化的，例如JFK应该是一个重要的中转点）

3、昨天项目内有一道SQL题目，但是需要用算法解决：查询两次曝光成功点击事件间隔中

曝光未点击的数量的最大值（推荐效率问题，也是用户路径问题，如果用上题目思考，就是流量引入BBB和AAA，甚至CCC，DDD都未能成功引发曝光，只有从新回到JFK才引发正常曝光，只要AAA，BBB，CCC，DDD相互交换都不成为成功曝光，只有倒入回JFK才成功）

题解

对于2问题中，找到成本最低的环路（用户消费决策路径、或者产品需要设计的最低成本路径）需要用到欧拉回路/通路问题，需要考虑Hierholzer 算法或者DFS最小堆回朔算法（倒序插入，这是个比较暴力的方式）Hierholzer 算法看就py和C++最简单

```
class Solution:
    def findItinerary(self, tickets: List[List[str]]) -> List[str]:
        def dfs(curr: str):
            while vec[curr]:
                tmp = heapq.heappop(vec[curr])
                dfs(tmp)
            stack.append(curr)
        vec = collections.defaultdict(list)
        for depart, arrive in tickets:
            vec[depart].append(arrive)
        for key in vec:
            heapq.heapify(vec[key])
        stack = list()
        dfs("JFK")
        return stack[::-1]
```

链接：leetcode-cn.com/problems/reconstruct-itinerary/solution/zhong-xin-an-pai-xing-cheng-by-leetcode-solution/

对于3问题：目前的做法是从SQL中提取原始的所有数据，然后一次做遍历算法（排序）

进化

考虑大数据有海量数据，而都在服务器计算显然不适合，损耗很大，我们就构想，能否把类似问题3的数据在客户端完成计算，结果上传服务器。数据结构形式可以是如下的
list（Array）

[JFK，[AAA,CCC]],[JFKm[DDD,AAA,CCC]],[[AAA,CCC],JFK,[DDD]],[[AAA,CCC,BBB]]这

样上报数据，实际用的行为是JFK - AAA - CCC，中断操作后继续 - JFK - DDD - AAA -CCC 再中断操作，每次中断操作就是上报时间点，可能包括JFK，也可能不包括，如果需要针对JFK切割且中断上报，则是[JFK，[AAA,CCC]],[JFK,[DDD,AAA,CCC,AAA,CCC]],[JFK,[DDD,AAA,CCC,BBB]]借助客户端存储建立一个可排序栈即可完成,服务器改用SQL或者读取数据排序皆可比较简单形式完成，这样比纯在服务器端计算工作要更明确，对数据治理也更清晰

目前工程中埋点我们都采用sdk利用本地存储机制建立栈，用户行为依次入栈，然后再条件配置下，出栈，倒排，拆解，发送数据，且数据可能是单个有向边，或者连续有向边

总结

1、首要问题就是拆出关键路径分组客户端排序，把大数据的的深度探索问题和栈回朔问题转移到客户端来完成，形成有向数据边，并且成链条解决逆序插入排序的问题，简化算法

2、找到环路中的闭合环点为业务平台汇聚点（例如对于购买来说，所有业务都会汇聚到支付，不论你先买什么后买什么，都是要汇入支付环节的）这是路径规划的核心：寻找业务价值环节中最有价值的环节

3、大数据计算现在并非纯SQL为王的天下，在算法函数开窗函数加入以后，最终看很可能回到算法解题思路（足够大内存就可以了），所以写SQL以后可能慢慢就是解决存储的问题，而通过SQL提取数据，开工程算法用内存解决以后再写回SQL也许会成为主流