# 图结构研讨

欧拉路径问题

# 目录/CONTENTS

01

# 欧拉路径问题概述

# 欧 拉 路 径 问 题 概 述

欧拉路径问题源自于柯尼斯堡七桥问题，该问题是：当时东普鲁士柯尼斯堡（今日俄罗斯加里宁格勒）市区跨普列戈利亚河两岸，河中心有两个小岛。小岛与河的两岸有七条桥连接。在所有桥都只能走一遍的前提下，如何才能把这个地方所有的桥都走遍？欧拉在在他1736年发表的论文《柯尼斯堡的七桥》中不仅解决了七桥问题，也提出了一笔画定理（即欧拉路径）。

而欧拉路径问题，本质上就是**判断这个图是否能够遍历完所有的边而没有重复**，特别的，如果这个路径可以回到起点，称为欧拉回路。

抽象成数学问题表示如下：

欧拉路径：在一个图中，由i点出发，将每个边遍历一次最终到达j点的一条路径。

欧拉回路：i=j时的欧拉路径。

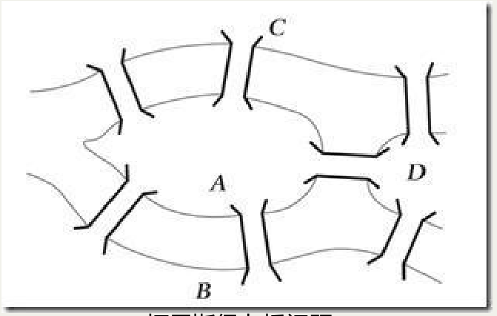
而满足欧拉路径或欧拉回路的充要条件为：

1）连通的无向图 G 有欧拉路径的充要条件是：G 中奇顶点（连接的边数量为奇数的顶点）的数目等于0或者2。

2）连通的无向图 G 是欧拉环（存在欧拉回路）的充要条件是：G 中每个顶点的度都是偶数。

3）连通的有向图 G 有欧拉路径的充要条件：由于起点和终点不同，那么起点的出度显然比它的入度少一，终点的入度显然比它的出度少一,而其他点的入度等于出度。

4）连通的有向图 G 是欧拉环（存在欧拉回路）的充要条件是：所有点的入度等于出度，就存在一条欧拉回路。




柯尼斯堡七桥问题

02

# 算法思想

## 算　法　思　想

**1、Hierholzer 算法（逐步插入回路法）**

在满足欧拉路径性质的子图中,加入一个环仍后可一笔画完成。而实现就是起点开始,DFS走到它所连接的点时删去这条边,顺着继续操作,结束时在一个栈中放入此时点的编号,最后倒序倒序输出即可。

通俗来讲，该算法分为以下步骤：

1）选择任一顶点为起点，遍历所有相邻边。

2）深度搜索，访问相邻顶点。将经过的边都删除。

3）如果当前顶点没有相邻边，则将顶点入栈。

4）栈中的顶点倒序输出，就是从起点出发的欧拉回路。

（当我们顺序地考虑该问题时，我们也许很难解决该问题，因为我们无法判断当前节点的哪一个分支是「死胡同」分支。不妨倒过来思考。我们注意到只有那个入度与出度差为 1 的节点会导致死胡同。而该节点必然是最后一个遍历到的节点。我们可以改变入栈的规则，当我们遍历完一个节点所连的所有节点后，我们才将该节点入栈（即逆序入栈）。对于当前节点而言，从它的每一个非「死胡同」分支出发进行深度优先搜索，都将会搜回到当前节点。而从它的「死胡同」分支出发进行深度优先搜索将不会搜回到当前节点。也就是说当前节点的死胡同分支将会优先于其他非「死胡同」分支入栈。）

（DFS：在搜索过程中，每当访问某个顶点v后，DFS会递归地访问它所有未被访问的相邻顶点。或者先访问v，把所有与v相关联的顶点的边存入栈中。弹出栈顶元素，栈顶元素代表的边所关联的另一个顶点就是下一个要被访问的元素，对该元素重复对v的操作。以此类推，直至栈中所有元素都被处理完毕。简而言之：一条路走到黑，直到无路可走的情况下，才会选择回头，然后重新选择一条路。可参考CSDN上“【算法入门】深度优先搜索(DFS)”这篇文章理解）

## 算　法　思　想

**1、Hierholzer 算法（逐步插入回路法）大概步骤**

```
void dfs(int i) {
    for(int j = 0; j < maxn; j++) {
        if(G[i][j]) {
            G[i][j] = G[j][i] = 0; // 删边
            dfs(j);
        }
    }
    res[n--] = i; //因为是回溯的时候存，所以倒着存
}
```

算　法　思　想

**2、Fluery算法**

首先我们了解一下桥的概念：设无向图G(V, E)为连通图，若边集E1⊆E，在图G中删除E1中所有的边后得到的子图是不连通的，而删除了E1的任一真子集后得到的子图是连通图，则称E1是G的一个割边集。若一条边构成一个割边集，则称该边为割边或桥。即代表这条边一旦被删除，这张图的连通块数量会增加。等价地说，一条边是一座桥当且仅当这条边不在任何环上。一张图可以有零或多座桥。

而Fluery的核心思想是：建立图形，任意选点，把vi的点对应的边设置为ei+1，结束条件是如果此时该点没有关联的边时算法结束，否则可以随意选择哪一条边走，若无别的边提供选择，否则不选择过桥。

大致步骤：

1）任意选取G中的一个点V0,令P0 ＝ V0。

2）假设沿pi = v0e1 v1e2 v2e3 …… viei+1走到了点vi，按照下面的方法从E -｛e1,e2,…,ei｝中选取ei + 1:

a）ei + 1与vi关联。

b）除非无别的边可以选择，否则ei + 1不应该是Gi = G - {e1,e2,…,ei}中的桥。

3）当（2）不能再进行时算法停止。

可以证明的是，当算法停止时，所得到的简单回路pm = v0e1 v1e2 v2……emvm（vm == v0）为G中的一条欧拉回路。

## 算　法　思　想

**2、Fleury算法大概步骤**

```
void dfs(int x){
    s.push(x);
    for(int i=1;i<=n;i++){
        if(map[x][i]>0){
            map[i][x] = map[x][i] = 0;   //删除该边
            dfs(i);
            break;}}}
void Fleury(int v){
    while(!s.empty())
    s.pop();s.push(v);
    while(!s.empty()){
        int b = 0;
        for(int i=1;i<=n;i++){if(map[s.top()][i]>0){
                b = 1;break;}}
        if(b == 0){  //没有相关联的边
            path[cnt++] = s.top();
            s.pop();}
        else{int y = s.top();
            s.pop();
            dfs(y);   //如果存在边的话，就深度遍历
        }}
    cout<<endl;}
```

# 03

## 求解过程

## 求　解　过　程

大致了解欧拉路径问题后，给出其应用问题：

**重新安排行程**

给定一个机票的字符串二维数组 [from, to]，子数组中的两个成员分别表示飞机出发和降落的机场地点，对该行程进行重新规划排序。所有这些机票都属于一个从 JFK（肯尼迪国际机场）出发的先生，所以该行程必须从 JFK 开始。

提示：

如果存在多种有效的行程，请你按字符自然排序返回最小的行程组合。例如，行程 ["JFK", "LGA"] 与 ["JFK", "LGB"] 相比就更小，排序更靠前

所有的机场都用三个大写字母表示（机场代码）。

假定所有机票至少存在一种合理的行程。

所有的机票必须都用一次 且 只能用一次。

输入：[["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

输出：["JFK", "MUC", "LHR", "SFO", "SJC"]

输入：[["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

输出：["JFK","ATL","JFK","SFO","ATL","SFO"]

解释：另一种有效的行程是 ["JFK","SFO","ATL","JFK","ATL","SFO"]。但是它自然排序更大更靠后。

## 求　　解　　过　　程

具体求解思想：对于此题，我们可以理解成给定一个n个点m条边的图，要求从指定的顶点出发，经过所有的边恰好一次（可以理解为给定起点的「一笔画」问题），使得路径的字典序最小。

利用Hierholzer算法给出实现代码

```
class Solution {
public:
    unordered_map<string, priority_queue<string, vector<string>, std::greater<string>>> vec;
    vector<string> stk;
    void dfs(const string& curr) {
        while (vec.count(curr) && vec[curr].size() > 0) {
            string tmp = vec[curr].top();
            vec[curr].pop();
            dfs(move(tmp));
        }
        stk.emplace_back(curr);
    }
    vector<string> findItinerary(vector<vector<string>>& tickets) {
        for (auto& it : tickets) {
            vec[it[0]].emplace(it[1]);
        }
        dfs("JFK");
        reverse(stk.begin(), stk.end());
        return stk;
    }
};
```

04

# 性能分析

## 性 能 分 析

时间复杂度：O(mlogm)，其中 m是边的数量。对于每一条边我们需要O(logm)地删除它，最终的答案序列长度为 m+1，而与n无关。
空间复杂度：O(m)，其中 m是边的数量。我们需要存储每一条边。

题目及题解来源：
作者：LeetCode-Solution
链接：https://leetcode-cn.com/problems/reconstruct-itinerary/solution/zhong-xin-an-pai-xing-cheng-by-leetcode-solution/
来源：力扣（LeetCode）

# THAKNS FOR WATCHING