

欧拉图

本页面将简要介绍欧拉图的概念、实现和应用。



定义

本文中仅讨论有限图。

在图论中，**欧拉路径 (Eulerian path)** 是经过图中每条边恰好一次的路径，**欧拉回路 (Eulerian circuit)** 是经过图中每条边恰好一次的回路。如果一个图中存在欧拉回路，则这个图被称为**欧拉图 (Eulerian graph)**；如果一个图中不存在欧拉回路但是存在欧拉路径，则这个图被称为**半欧拉图 (semi-Eulerian graph)**。



Warning



此处定义中虽然使用「路径」一词，但严格说来此处使用的概念应该是「迹 (trail)」。

欧拉路径与欧拉回路仅能使用每条边恰好一次，但并没有对经过顶点的情况进行限制。

性质

以下我们假设所讨论的图 G 中不存在孤立顶点。该假设不失一般性，因为对于存在孤立顶点的图 G ，以下性质对从 G 中删除孤立顶点后得到的图 G' 仍然成立。

对于连通图 G ，以下三个性质是互相等价的：

1. G 是欧拉图；
2. G 中所有顶点的度数都是偶数（对于有向图，每个顶点的入度等于出度）；
3. G 可被分解为若干条不共边回路的并。

以下我们对等价性进行证明。

若一个图 G 是欧拉图，那么 G 中所有顶点的度数都是偶数：考虑从任意顶点开始沿着欧拉回路走一圈，则每个点 v 的度数等于离开点 v 的次数加到达点 v 的次数。又由于行动的轨迹是一个回路，则对于每个点 v ，离开该点的次数等于到达该点的次数。这也就是说，每个点的度数都形如 $2k$ ，即偶数。特别地，对于有向图，根据相同的证明过程，每个顶点的入度等于出度。

若一个图 G 中所有顶点的度数都是偶数（或入度与出度相等），则它可被分解为若干条不共边回路的不交并：考虑从任意顶点 u 开始，选择任意出边 (u, v) ，走向对应的相邻顶点 v 并删除 (u, v) ，直到返回最初开始的顶点 u 。可以证明该过程必定会最终回到 u ：每当到达一个新的顶点 $v \neq u$ 时，根据上一条性质，该顶点剩余的度数为奇数，也就是说必定存在一条出边，该过程不

会在点 v 终止。（换句话说，该过程会且仅会在回到点 u 时停止。）又因图 G 中的边数是有限的，该过程必定会在有限步内停止，则最终必然可以返回 u 并得到一条回路。注意到在前述证明中我们仅使用了点度数均为偶数的性质，且在找到并删除一条回路后剩下部分的图仍然满足该性质，我们可以不断重复该过程直到剩下的图为空图，从而将 G 拆分为若干条不共边的回路。更进一步地，每条回路都可以被从其多次经过的顶点处分解成若干简单环的不交并，所以上述性质中的简单回路亦可被替换为简单环。

若一个连通图 G 可被分解为若干条不共边回路的不交并，则 G 是欧拉图：对于一组不共边回路，每次从中选出两条有共同顶点的回路并将其合并为一条，重复该过程直到不存在有共同顶点的两条回路。可以证明该过程结束时剩下的回路唯一。对于任意两条不共边回路 P_1, P_2 ，若 P_1 与 P_2 共点，则可以在共点处直接进行合并；否则，任取 P_1 上的点 v_1 与 P_2 上的点 v_2 ，根据 G 的连通性，存在连接 v_1 和 v_2 的路径 e_1, e_2, \dots, e_k ，其中的每条边 e_i 都被一个回路 C_i 包含，且 P_1 与 C_1 ， C_i 与 C_{i+1} ， C_k 与 P_2 均存在共点（或者 $C_i = C_{i+1}$ ，此情况不影响证明）。此情况下， P_1 与 P_2 可以通过 C_1, \dots, C_k 进行合并。也就是说，任意两条回路都可以进行合并，最后剩下的回路必定唯一，且组成该回路的边集是所有不共边回路的并即 $E(G)$ ，该回路为 G 上的欧拉回路， G 为欧拉图。

以上的性质同时也构成了欧拉图的判断条件。具体地说，一个图是欧拉图当且仅当非零度顶点互相（强）连通，且顶点的度数都是偶数（或入度与出度相等）。

对于半欧拉图，其性质与欧拉图相似：一个半欧拉图具有恰好两个奇度数的顶点，且这两个顶点就是欧拉路径的两个端点。通过在这两个点连接起来，可以将半欧拉图转化为欧拉图。通过删除欧拉图中的任意一条边，可以得到一个半欧拉图。由此可以导出半欧拉图的判别法：一个图是半欧拉图当且仅当非零度顶点互相（强）连通，且奇度数顶点恰好有两个。对于有向图，第二个条件为恰存在两个顶点 u, v ，其中 $\deg^+(u) - \deg^-(u) = 1, \deg^+(v) - \deg^-(v) = -1$ ，且其余顶点的入度等于出度。

欧拉回路/欧拉路径的构造

此处我们介绍最常用的 Hierholzer 算法，该算法的核心思想为利用上述欧拉图性质中的第三点，即欧拉图可以被拆解为若干条不共边回路的不交并。可以注意到，在上述证明中其实已经提到了完整可行的将不共边回路合并为欧拉回路的操作，且在使用合适的数据结构储存时（如使用链表的头指针结构储存环）实现并不困难。

算法的具体流程为先从图中找到一条回路作为当前回路，每次从当前回路中选取剩余度数不为零的点，从该点出发找到一条新的简单回路，并将该简单回路与前回路合并，重复该过程直到当前回路中的所有点均无剩余度数，此时的当前回路即为欧拉回路。

该算法同样适用于有向图。对于半欧拉图，可以从图中找到一条连接两个奇度数点的路径作为当前路径，每次选取度数非零的点寻找简单回路并将其与当前路径合并，最后得到欧拉路径。

实现

Hierholzer 算法的伪代码如下：

```

1  Input. The edges of the graph  $e$ , where each element in  $e$  is  $(u, v)$ 
2  Output. The vertex of the Euler Road of the input graph.
3  Method.
4  Function Hierholzer ( $v$ )
5       $circle \leftarrow$  Find a Circle in  $e$  Begin with  $v$ 
6      if  $circle = \emptyset$ 
7          return  $v$ 
8       $e \leftarrow e - circle$ 
9      for each  $v \in circle$ 
10          $v \leftarrow$  Hierholzer( $v$ )
11     return  $circle$ 
12 Endfunction
13 return Hierholzer(any vertex)

```

时间复杂度分析

Hierholzer 算法的时间复杂度为 $O(|E| + |V|)$ 。

注意到在前述正确性分析中，在欧拉图或半欧拉图上寻找简单回路（或半欧拉图的初始路径）的过程是 **无需回溯** 的，只要沿着剩下的边一直走就必定可以发现所求的回路或路径，且 **每条边仅会被访问一次**。为了利用这一性质，在实现上应采取类链表的方式储存图中的边，如邻接表或链式前向星，以便每条边在被访问过后即刻删除之。如果采用朴素的邻接矩阵进行储存，则每次寻边耗时 $O(|V|)$ ，总复杂度为 $O(|V||E|)$ 。



Note

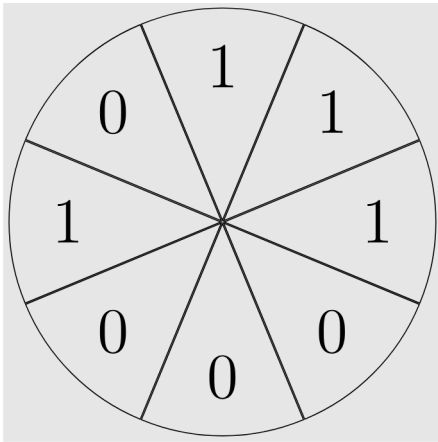
事实上，该算法的准确复杂度应为 $O(|E|)$ 而非 $O(|V| + |E|)$ ，这是因为该算法的实现方式可以采取依赖于边而不依赖于点的方法，通过维护剩余边的总链表来进行下一步回路的寻找。

如果需要输出字典序最小的欧拉路或欧拉回路，则需要将边排序，时间复杂度为 $\Theta(|E| \log |E|)$ 或 $\Theta(|E|)$ （使用计数排序或者基数排序）。

应用

有向欧拉图可用于计算机译码。

设有 m 个字母，希望构造一个有 m^n 个扇形的圆盘，每个圆盘上放一个字母，使得圆盘上每连续 n 位对应长为 n 的符号串。转动一周（ m^n 次）后得到由 m 个字母产生的长度为 n 的 m^n 个各不相同的符号串。



构造如下有向欧拉图：

设 $S = \{a_1, a_2, \dots, a_m\}$ ，构造 $D = \langle V, E \rangle$ ，如下：

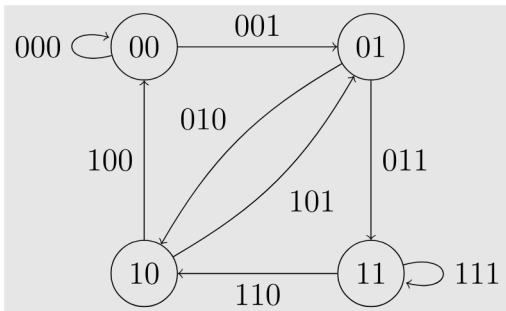
$$V = \{a_{i_1}a_{i_2} \cdots a_{i_{n-1}} \mid a_i \in S, 1 \leq i \leq n-1\}$$

$$E = \{a_{j_1}a_{j_2} \cdots a_{j_{n-1}} \mid a_j \in S, 1 \leq j \leq n\}$$

规定 D 中顶点与边的关联关系如下：

顶点 $a_{i_1}a_{i_2} \cdots a_{i_{n-1}}$ 引出 m 条边： $a_{i_1}a_{i_2} \cdots a_{i_{n-1}}a_r, r = 1, 2, \dots, m$ 。

边 $a_{j_1}a_{j_2} \cdots a_{j_{n-1}}$ 引入顶点 $a_{j_2}a_{j_3} \cdots a_{j_n}$ 。



这样的 D 是连通的，且每个顶点入度等于出度（均等于 m ），所以 D 是有向欧拉图。

任求 D 中一条欧拉回路 C ，取 C 中各边的最后一个字母，按各边在 C 中的顺序排成圆形放在圆盘上即可。

例题

洛谷 P2731 骑马修栅栏



给定一张有 500 个顶点的无向图，求这张图的一条欧拉路或欧拉回路。如果有多组解，输出最小的那一组。

在本题中，欧拉路或欧拉回路不需要经过所有顶点。

边的数量 m 满足 $1 \leq m \leq 1024$ 。

解题思路



本题为 Hierholzer 算法的直接应用。

保存答案可以使用 `std::stack<int>`，因为如果找的不是回路的话必须将那一部分放在最后。

注意，不能使用邻接矩阵存图，否则时间复杂度会退化为 $\Theta(nm)$ 。由于需要将边排序，建议使用前向星或者 `std::vector` 存图。示例代码使用 `std::vector`。

示例代码

```
1  #include <algorithm>
2  #include <iostream>
3  #include <stack>
4  #include <vector>
5  using namespace std;
6
7  struct edge {
8      int to;
9      bool exists;
10     int revref;
11
12     bool operator<(const edge& b) const { return to < b.to; }
13 };
14
15 vector<edge> beg[505];
16 int cnt[505];
17
18 constexpr int dn = 500;
19 stack<int> ans;
20
21 void Hierholzer(int x) { // 关键函数
22     for (int& i = cnt[x]; i < (int)beg[x].size();) {
23         if (beg[x][i].exists) {
24             edge e = beg[x][i];
25             beg[x][i].exists = beg[e.to][e.revref].exists = false;
26             ++i;
27             Hierholzer(e.to);
28         } else {
29             ++i;
30         }
31     }
32     ans.push(x);
33 }
34
35 int deg[505];
36 int reftop[505];
37
38 int main() {
39     cin.tie(nullptr)->sync_with_stdio(false);
40     for (int i = 1; i <= dn; ++i) {
41         beg[i].reserve(1050); // vector 用 reserve 避免动态分配空间，
42         加快速度
43     }
44
45     int m;
46     cin >> m;
47     for (int i = 1; i <= m; ++i) {
48         int a, b;
49         cin >> a >> b;
```

```

50     beg[a].push_back(edge{b, true, 0});
51     beg[b].push_back(edge{a, true, 0});
52     ++deg[a];
53     ++deg[b];
54 }
55
56 for (int i = 1; i <= dn; ++i) {
57     if (!beg[i].empty()) {
58         sort(beg[i].begin(), beg[i].end()); // 为了要按字典序贪心，
59 必须排序
60     }
61 }
62
63 for (int i = 1; i <= dn; ++i) {
64     for (int j = 0; j < (int)beg[i].size(); ++j) {
65         beg[i][j].revref = reftop[beg[i][j].to]++;
66     }
67 }
68
69 int bv = 0;
70 for (int i = 1; i <= dn; ++i) {
71     if (!deg[bv] && deg[i]) {
72         bv = i;
73     } else if (!(deg[bv] & 1) && (deg[i] & 1)) {
74         bv = i;
75     }
76 }
77
78 Hierholzer(bv);
79
80 while (!ans.empty()) {
81     cout << ans.top() << '\n';
82     ans.pop();
83 }
}


```


习题

- [SGU 101 Domino](#)
- [POJ 1780 Code](#)
- [洛谷 P1127 词链](#)
- [洛谷 P1333 瑞瑞的木棍](#)
- [洛谷 P1341 无序字母对](#)
- [洛谷 P6066 \[USACO05JAN\]Watchcow S](#)
- [洛谷 P6628 \[省选联考 2020 B 卷\] 丁香之路](#)

- [洛谷 P3520 \[POI 2011\] SMI-Garbage](#)

 本页面最近更新：2025/8/11 19:57:48，[更新历史](#)

 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

 本页面贡献者：[orzAtalod](#), [Ir1d](#), [NachtgeistW](#), [StudyingFather](#), [H-J-Granger](#), [Tiphereth-A](#), [countercurrent-time](#), [Enter-tainer](#), [CCXXI](#), [Early0v0](#), [mgt](#), [AngelKitty](#), [cjsoft](#), [diauweb](#), [ezoixx130](#), [GekkaSaori](#), [Haohu Shen](#), [Konano](#), [leoleoasd](#), [LovelyBuggies](#), [lychees](#), [Makkiy](#), [Marcythm](#), [minghu6](#), [P-Y-Y](#), [PotassiumWings](#), [SamZhangQingChuan](#), [sshwy](#), [Suyun514](#), [weiyong1024](#), [aofall](#), [Chrogeek](#), [CoelacanthusHex](#), [GavinZhengOI](#), [Gesrua](#), [Great-designer](#), [Henry-ZHR](#), [iamtwz](#), [kenlig](#), [kxccc](#), [lowtune](#), [mcendu](#), [Menci](#), [Peanut-Tang](#), [Persdre](#), [PsephurusGladius](#), [shuzhouliu](#), [SukkaW](#), [zhu-yifang](#), [zryi2003](#)

© 本页面的全部内容在 [CC BY-SA 4.0](#) 和 [SATA](#) 协议之条款下提供，附加条款亦可能应用