图的遍历调研

一、一笔画问题

1. 概述：

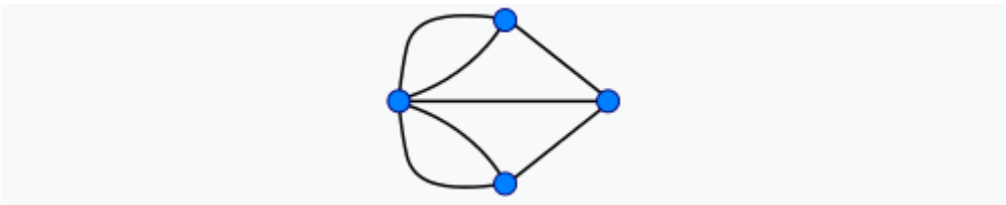
图论起源于 18 世纪，1736 年瑞士数学家欧拉（Euler）发表了图论的第一篇论文“哥尼斯堡七桥问题”。在当时的哥尼斯堡城有一条横贯全市的普雷格尔河，河中的两个岛与两岸用七座桥连结起来。当时那里的居民热衷于一个难题：有游人怎样不重复地走遍七桥，最后回到出发点。[1]

为了解决这个问题，欧拉用 A,B,C,D 4 个字母代替陆地，作为 4 个顶点，将联结两块陆地的桥用相应的线段表示，于是哥尼斯堡七桥问题就变成了图中，是否存在经过每条边一次且仅一次，经过所有的顶点的回路问题了。欧拉在论文中指出，这样的回路是不存在的。

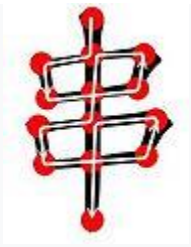
什么是欧拉路径？欧拉路径就是一条能够不重不漏地经过图上的每一条边的路径，即小学奥数中的一笔画问题。而若这条路径的起点和终点相同，则将这条路径称为欧拉回路。

数学家欧拉找到一笔画的规律是：

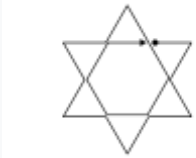
- 凡是由偶点组成的连通图，一定可以一笔画成。画时可以把任一偶点为起点，最后一定能以这个点为终点画完此图。
- 凡是只有两个奇点的连通图（其余都为偶点），一定可以一笔画成。画时必须把一个奇点为起点，另一个奇点终点。
- 其他情况的图都不能一笔画出。（有偶数个奇点除以二可以算出此图至少需几笔画成。）



图一：无法一笔画



图二：尽管按照中文书写习惯“串”字不止一笔，但它可以一笔写成。



图三：六角星

图四

2. 求解算法思想

查找欧拉路径的算法有 Fluery 算法和 Hierholzer 算法。[2]

Hierholzer 算法流程：

1.对于无向图，判断度数为奇数的点的个数，若为 0，则设任意一点为起点，若为 2，则从这 2 个点中任取一个作为起点；对于有向图，判断入度和出度不同的点的个数，若为 0，则设任意一点为起点，若为 2，则设入度比出度小 1 的点为起点，另一点为终点。具体起点的选择要视题目要求而定。

2.从起点开始进行递归：对于当前节点 x，扫描与 x 相连的所有边，当扫描到一条(x,y)时，删除该边，并递归 y。扫描完所有边后，将 x 加入答案队列。

3.倒序输出答案队列。（因为这里是倒序输出，我们可以用栈来存储答案，当然用双端队列也可以）

解析：

从起点开始，每一次执行递归函数，相当于模拟一笔画的过程。递归的边界显然就是路径的终点，对于一个有欧拉路径的图，此时图上的所有边都已被删除，自然就不能继续递归。由于存储答案是在遍历以后进行的，答案存储也就是倒序的，因此要倒序输出答案。

Fleury 算法流程：

1. 任取 G 中一个点 x

2. 选择一条从点 x 出发的边 i（若 i 可以不为桥则不应将 i 选为桥），设 i 连向点 y，删除 i，然后：

1）i 不为桥，走到点 y；

2）i 迫不得已必须为桥，走到点 y 并删去点 x（因为删去 i 后，点 x 将成为孤立点）。

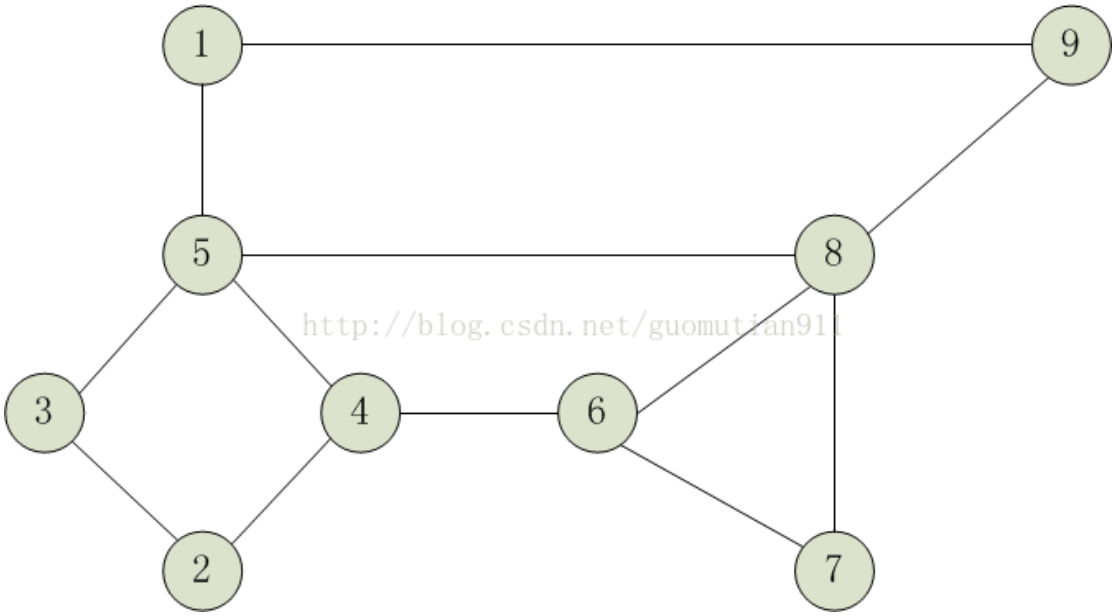
3. 步骤 2 无法执行时停止算法

当算法停止时，依次经过的点所得到的简单回路为图 G 的一条欧拉回路。

3. 举例说明求解过程

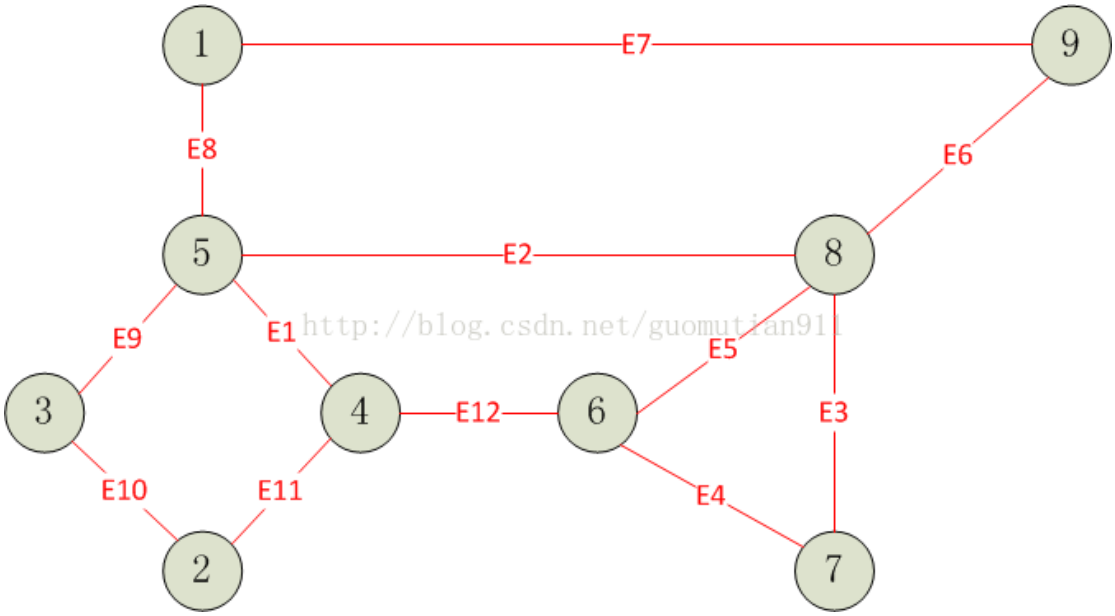
图 2 为连通图 G，现利用 Fleury 算法求它的欧拉通路。

（注意区分：欧拉通路、欧拉回路）



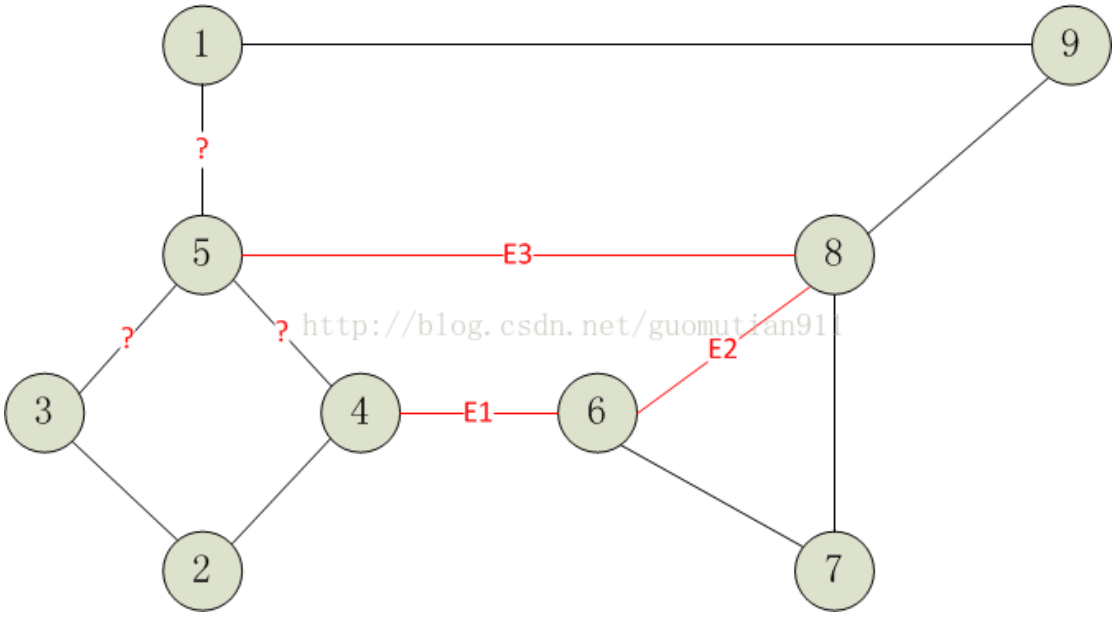
其中一种欧拉通路如下：4 5 8 7 6 8 9 1 5 3 2 4 6，

其搜索路径如下图所示：



现在让我们来分析算法实现过程：

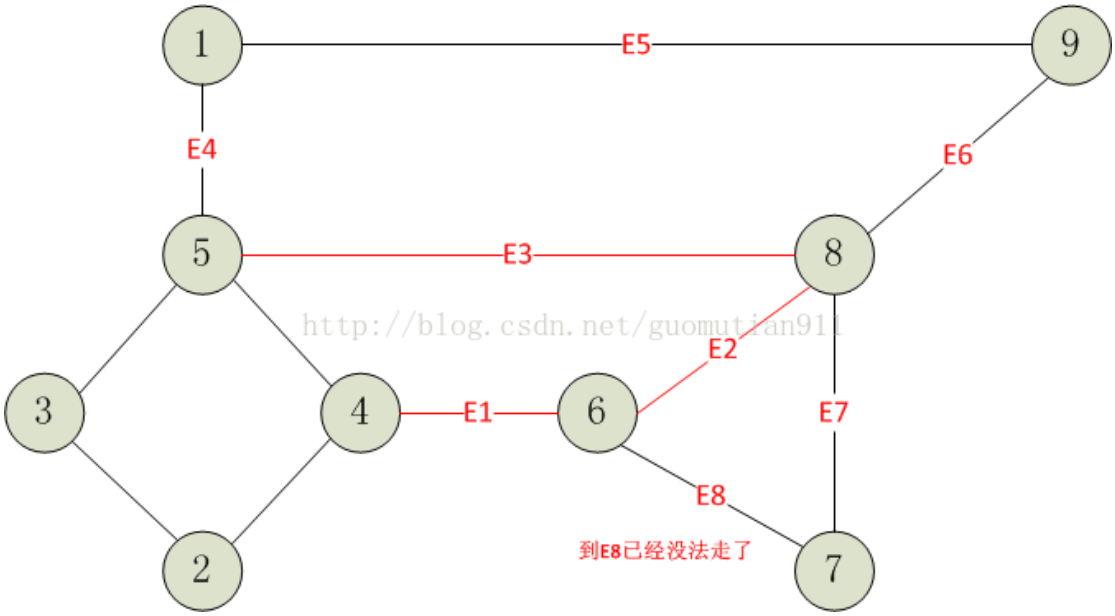
假设我们这样走：4，6，8，5，此时在 5 处有三种选择（3，4，1），那么哪种能走通哪种走不通呢？答案是（3，4）通，1 不通。为什么呢？来看下图。



分析：

因为（5~1）之间的边是除去已走过边（E（G）-｛E1（4~6），E2（6~8），E3（8~5）｝）图 G 的一个桥，所谓桥即去掉该边后，剩下的所有顶点将不能够连通，即无法构成连通图。

而选择（5~3）和（5~4）则满足定义中第二条（b）中的要求。当然当（5~3）和（5~4）都不存在，即定义中所说“除非无别的边可供选择时”，此时就就可以选择（5~1），其他情况下一定要优先选择非桥的边，否则就可能出现无法走通的情况。也就是说该搜索方法无法构成欧拉通路。如下图是选择（5~1）的后果：



而（5~3）和（5~4）则可以顺利完成欧拉图通路的搜索。

4. 算法具体步骤

Hierholzer 算法

```
#include<iostream>
#include<stack>
using namespace std;
const int N=500;
int n,tot,c=N,jp[N],cnt[N],edge[N][N];
char a,b;
stack<int> q;
void dfs(int now)
{
    for(int i=1;i<=N;i++)
        if(edge[now][i]==1)
```

```
            {
                edge[now][i]--,edge[i][now]--;
                dfs(i);
            }
        q.push(now);//加入答案队列
}//算法过程
int main()
{
        cin>>n;
        for(int i=1;i<=n;i++)
        {
            cin>>a>>b;
            c=min(c,a);
            c=min(c,b);
            edge[a][b]++，edge[b][a]++;
            cnt[a]++;
            cnt[b]++;//统计每个节点的度数
        }
        for(int i=1;i<=N;i++)
            if(cnt[i]%2==1)
                jp[tot++]=i;//找出度数为奇数的节点
        if(tot!=2 && tot)
        {
            cout<<"No Solution";
            return 0;
        }//若该图没有欧拉路径则判误
        int stat;
        if(tot)
            stat=min(jp[0],jp[1]);
        else
```

```
            stat=c;//找出起点
        dfs(stat);
        while(!q.empty())
        {
            char ct=q.top();
            cout<<ct;
            q.pop();
        }//倒序输出
        return 0;
}
```

Fleury 算法

```
public void euler(int[][] graph,Stack s,int current,int start){
        int num_edges;    //graph 中的边数
        boolean flag = false;    //是否还有与 x 关联的边
        s.push(current);
        for(int i = start; i < graph.length ; i++){
            //从 start 开始搜索是否有边
            if(graph[i][x] > 0){ //i 与 current 有边
                flag = true;
                graph[i][current] = 0; graph[current][i] = 0; //删除边
                euler(graph,s,i,0);    //从 i 开始搜索
                break;
            }
        }

        if(flag){ //如果没有变与当前节点 current 相连
            s.pop();
            int m = s.peek();
```

```
        G[m][current] = G[current][m] = 1; //恢复边
        int new_start = current + 1;
        if(s.size() == num_edges){ //完成回路
            return;
        }else{
            euler(graph,s,m,new_start);
        }
    }
```
[3]

5. 性能分析

可以证明,当算法停止时所得的简单回路 *Wm=v0e1v1e2….emvm(vm=v0)*为 *G* 中的一条欧拉回路，复杂度为 *O(e*e)*。

二、哈密尔顿问题

1. 问题概述：

设有一个图，若其上存在一个圈，这个圈包含该图上的每一节点，则称该圈为哈密顿圈，这个图称为哈密顿图.哈密顿圈问题可叙述为：什么样的图为哈密顿图，或者说判断一个图是哈密顿图的行之有效的充分必要条件是什么.目前这一问题尚未解决.

关于哈密顿圈的研究，最早是欧拉(L.Euler)研究骑士(相当于中国象棋中的马)从棋盘上的某一位置出发，走遍所有可能的位置且每个位置只通过一次后，回到原来的位置，而哈密顿(W.R.Hamilton)研究环游世界的游戏是在欧拉之后近一个世纪，然而却由此开始引起人们对于这个问题的注意.

实际上，哈密顿的游戏就是，在一个正十二面体上，从一个顶点开始沿着棱走遍所有的十二个顶点回到原地，使得每一顶点只通过一次.就是在十二面体相应的图上求一个哈密顿圈.若一个图上存在一条路，这条路包含该图上的所有节点，则称该图为可迹图，称这样的路为哈密顿路，若对于一个图上的每一节点，存在一个以该节点为始点的哈密顿路，则称该图为齐次可迹图，一个图被称为哈密顿连通图，若对于这图上任何两节点，存在一条哈密顿路连结这两节点，称一

个图为亚哈密顿图，若从这图上去掉无论哪一节点，所得的图都是哈密顿图，称一个图为亚可迹图，若从这图上去掉无论哪一节点，所得的图都是可迹图。[4]

2. 求解算法思想

在 Dirac 定理的前提下构造哈密顿回路[5]

过程:

1:任意找两个相邻的节点 S 和 T,在其基础上扩展出一条尽量长的没有重复结点的路径.即如果 S 与结点 v 相邻,而且 v 不在路径 S -> T 上,则可以把该路径变成 v -> S -> T,然后 v 成为新的 S.从 S 和 T 分别向两头扩展,直到无法继续扩展为止,即所有与 S 或 T 相邻的节点都在路径 S -> T 上.

2:若 S 与 T 相邻,则路径 S -> T 形成了一个回路.

3:若 S 与 T 不相邻,可以构造出来一个回路.设路径 S -> T 上有 k+2 个节点,依次为 S, v1, v2, ..., vk, T.可以证明存在节点 vi(i 属于[1, k]),满足 vi 与 T 相邻,且 vi+1 与 S 相邻.找到这个节点 vi,把原路径变成 S -> vi -> T -> vi+1 -> S,即形成了一个回路.

4:到此为止,已经构造出来了一个没有重复节点的的回路,如果其长度为 N,则哈密顿回路就找到了.如果回路的长度小于 N,由于整个图是连通的,所以在该回路上,一定存在一点与回路之外的点相邻.那么从该点处把回路断开,就变回了一条路径,同时还可以将与之相邻的点加入路径.再按照步骤 1 的方法尽量扩展路径,则一定有新的节点被加进来.接着回到路径 2.

N(N>=2)阶竞赛图构造哈密顿通路

N 阶竞赛图:含有 N 个顶点的有向图，且每对顶点之间都有一条边。对于 N 阶竞赛图一定存在哈密顿通路。

数学归纳法证明竞赛图在 n >= 2 时必存在哈密顿路：

(1)n = 2 时结论显然成立；

(2)假设 n = k 时,结论也成立,哈密顿路为 V1, V2, V3, ..., Vk;

设当 n = k+1 时,第 k + 1 个节点为 V(k+1),考虑到 V(k+1)与 Vi(1<=i<=k)的连通情况,可以分为以下两种情况。

1:Vk 与 V(k+1)两点之间的弧为<Vk, V(k+1)>,则可构造哈密顿路径 V1, V2,…, Vk, V(k+1).

2:Vk 与 V(k+1)两点之间的弧为<V(k+1),Vk>,则从后往前寻找第一个出现的 Vi(i=k-1,i>=1,--i),满足 Vi 与 V(k+1)之间的弧为<Vi,V(k+1)>,则构造哈密顿路径 V1, V2, …, Vi, V(k+1), V(i+1), …, V(k).若没找到满足条件的 Vi,则说明对于所有的 Vi(1<=i<=k)到 V(k+1)的弧为<V(k+1),V(i)>,则构造哈密顿路径 V(k+1), V1, V2, …, Vk.

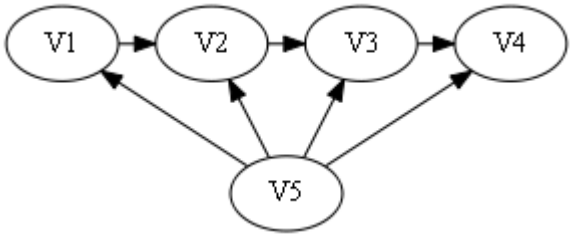
证毕。

竞赛图构造哈密顿路时的算法同以上证明过程。

3. 举例说明求解过程

用图来说明:[6]

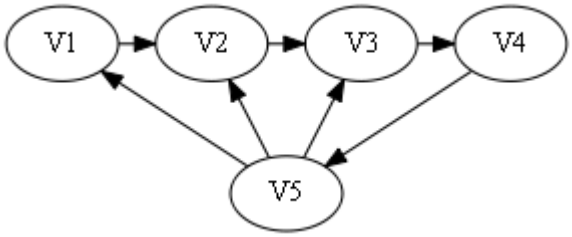
假设此时已经存在路径 V1 -> V2 -> V3 -> V4,这四个点与 V5 的连通情况有 16 种,给定由 0/1 组成的四个数,第 i 个数为 0 代表存在弧<V5,Vi>,反之为 1,表示存在弧<Vi,V5>



sign[]={0, 0, 0, 0}.

很显然属于第二种情况,从后往前寻找不到 1,即且不存在弧<Vi, V5>.

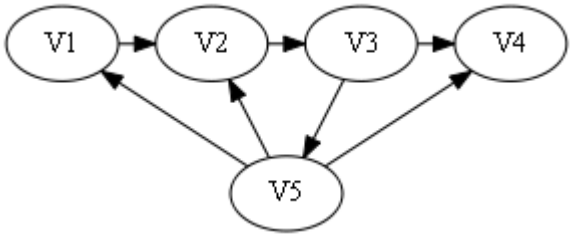
则构造哈密顿路:V5 -> V1 -> V2 -> V3 -> V4.



sign[]={0, 0, 0, 1}.

属于第一种情况,最后一个数字为 1,即代表存在弧<Vi, V5>且 i=4(最后一个点)

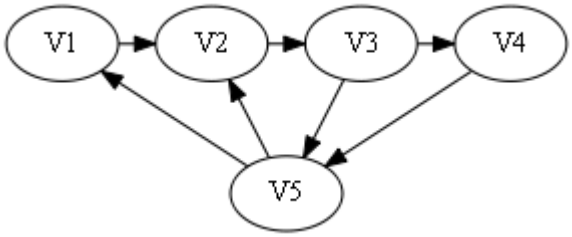
则构造哈密顿路: V1 -> V2 -> V3 -> V4 -> V5.



sign[]={0, 0, 1, 0}.

属于第二种情况,从后往前找到 1 出现的第一个位置为 3.

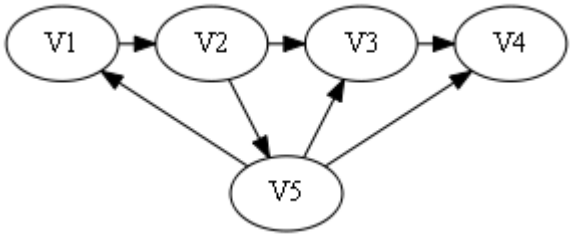
构造哈密顿路: V1 -> V2 -> V3 -> V5 -> V4.



sign[]={0, 0, 1, 1}.

属于第一种情况,最后一个数字为 1,即代表存在弧<Vi, V5>且 i=4(最后一个点)

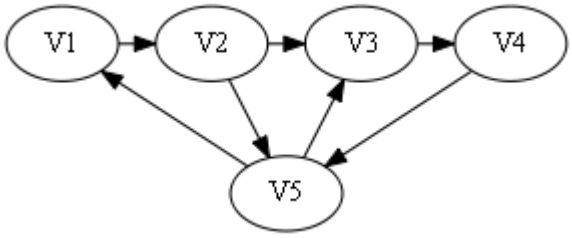
则构造哈密顿路: V1 -> V2 -> V3 -> V4 -> V5.



sign[]={0, 1, 0, 0}.

属于第二种情况,从后往前找到 1 出现的第一个位置为 2.

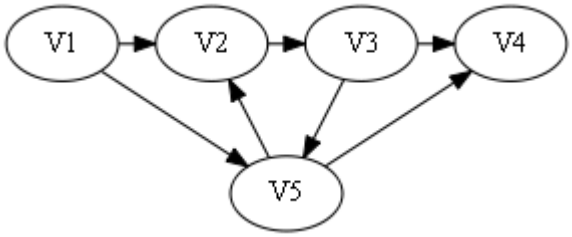
构造哈密顿路: V1 -> V2 -> V5 -> V3-> V4.



sign[]={0, 1, 0, 1}.

属于第一种情况,最后一个数字为 1,即代表存在弧<Vi, V5>且 i=4(最后一个点)

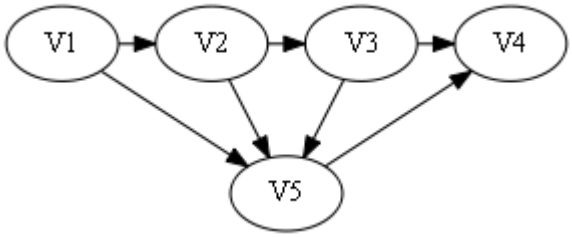
则构造哈密顿路:V1 -> V2 -> V3 -> V4 -> V5.(就不举末尾为 1 的栗子了~~)



sign[]={1, 0, 1, 0}.

属于第二种情况,从后往前找到 1 出现的第一个位置为 3.

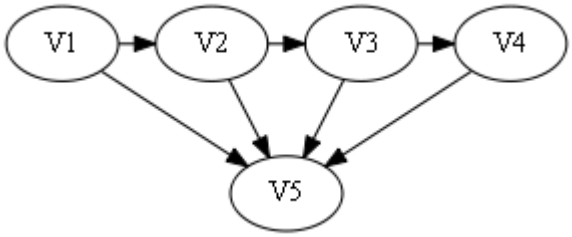
构造哈密顿路: V1 -> V2 -> V3 -> V5-> V4.



sign[]={1, 1, 1, 0}.

属于第二种情况,从后往前找到 1 出现的第一个位置为 3.

构造哈密顿路: V1 -> V2 -> V3 -> V5-> V4.



4. 算法具体步骤

设 s 为哈密顿回路的起始点,t 为哈密顿回路中终点 s 之前的点.ans[]为最终的哈密顿回路.倒置的意思指的是将数组对应的区间中数字的排列顺序方向.

1:初始化,令 s = 1,t 为 s 的任意一个邻接点.

2:如果 ans[]中元素的个数小于 n,则从 t 开始向外扩展,如果有可扩展点 v,放入 ans[]的尾部,并且 t=v,并继续扩展,如无法扩展进入步骤 3.

3:将当前得到的 ans[]倒置,s 和 t 互换,从 t 开始向外扩展,如果有可扩展点 v,放入 ans[]尾部,并且 t=v,并继续扩展.如无法扩展进入步骤 4.

4:如果当前 s 和 t 相邻,进入步骤 5.否则,遍历 ans[],寻找点 ans[i],使得 ans[i]与 t 相连并且 ans[i + 1]与 s 相连,将从 ans[i + 1]到 t 部分的 ans[]倒置,t=ans[i + 1],进如步骤 5.

5:如果当前 ans[]中元素的个数等于 n,算法结束,ans[]中保存了哈密顿回路(可看情况是否加入点 s).否则,如果 s 与 t 连通,但是 ans[]中的元素的个数小于 n,则遍历 ans[],寻找点 ans[i],使得 ans[i]与 ans[]外的一点(j)相连,则令 s=ans[i - 1],t = j,将 ans[]中 s 到 ans[i - 1]部分的 ans[]倒置,将 ans[]中的 ans[i]到 t 的部分倒置,将点 j 加入到 ans[]的尾部,转步骤 2.[7]

5. 性能分析

时间复杂度:

如果说每次到步骤 5 算一轮的话,那么由于每一轮当中至少有一个节点被加入到路径 S -> T 中,所以总的轮数肯定不超过 n 轮,所以时间复杂度为 O(n^2).空间上由于边数非常多,所以采用邻接矩阵来存储比较适合.

三、中国邮递员问题

1. 问题概述：

中国邮递员问题是邮递员在某一地区的信件投递路程问题。邮递员每天从邮局出发，走遍该地区所有街道再返回邮局，问题是他应如何安排送信的路线可以使所走的总路程最短。这个问题由中国学者管梅谷在 1960 年首先提出，并给出了解法——“奇偶点图上作业法”，被国际上统称为“中国邮递员问题”。用图论的语言

描述，给定一个连通图 G，每边 e 有非负权)，要求一条回路经过每条边至少一次，且满足总权最小。

这就是中国邮递员问题(Chinese Postman Problem)，简称 CPP 问题，其命名是因为中国数学家管梅谷在 1962 年首先提出了这个问题。[8]

2. 求解算法思想

奇偶点图上作业法：

1960 年我国管梅谷发表于数学学报上的论文“奇偶点图上作业法”，是针对于中国邮递员问题的最早论文，将关于“一笔画”问题的一些已知结果与物资调拨中的图上作业法的基本思想相结合，得到了 CPP 问题中添边策略的一种方法。[9]

①生成初始可行方案：

若图中有奇点，则把它配成对，每一对奇点之间必有一条链，把这条链的所有边作为重复边加到图中去，新图中必无奇点。便给出了第一个可行方案。

②调整可行方案：

使重复边总长度下降.当边(w , v) (w,v)(*w*,*v*)上有两条或两条以上的重复边时，从中去掉偶数条，得到一个总长度较小的方案。于是，有：

1)在最优方案中，图的每条边上最多有一条重复边。

2)在最优方案中，图中每个圈上的重复边的总权不大于该圈总权的一半。

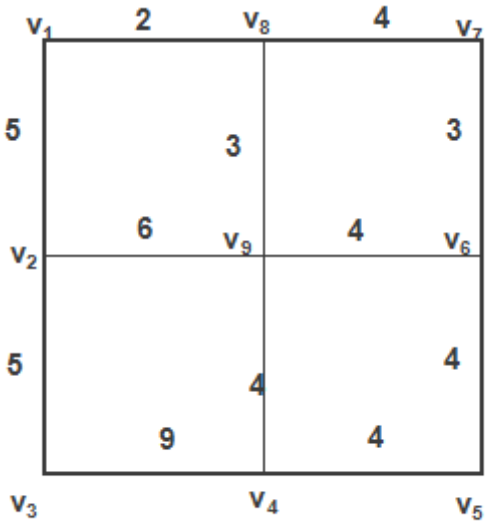
③判断最优方案的标准：

一个最优方案一定是满足上述 1)和 2)的可行方案。反之，一个可行方案若满足上述 1)和 2)，则这个可行方案一定是最优方案。

根据判断标准，对给定的可行方案，检查它是否满足上述条件 1)和 2)。若满足，所得方案即为最优方案；若不满足，则对方案进行调整，直至上述条件 1)和 2)均得到满足时为止。

3. 举例说明求解过程

快递员从 V1 出发给 V2、V3、V4、V5、V6、V7、V8、V9 派发快递，求派完所有回到原出发点的最短路径？如下图所示：[10]



题目分析

在分析这道题目之前，我们在想这样的一个问题

图 1 和图 2 当中哪一个图满足从图中任何一点出发，途径每条边最终还能回到原点？

图 1 无奇点不需要走重复边；图 2 有奇点需要走重复边。

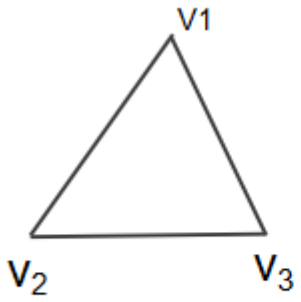


图1

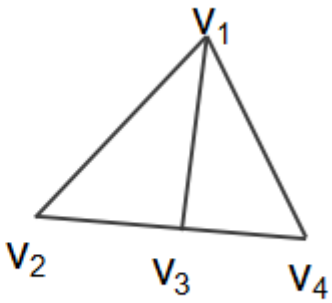


图2

从上面的结果可以让我们得到一些灵感。

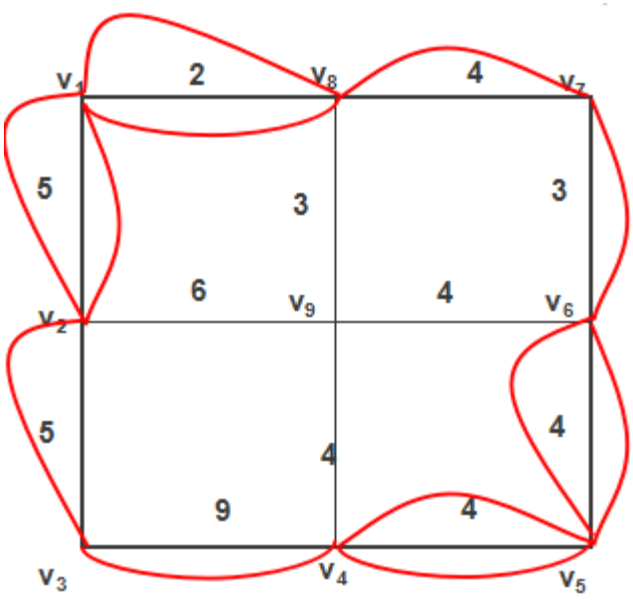
那么一个图应该满足什么条件才能达到上面要求呢？

满足各个结点的度为偶数！在一个多重边的连通图中，从某个顶点出发，经过不同的线路，又回到原出发点，这样的线路必是欧拉图。

定理 1：连通的多重图 G 是欧拉图，当且仅当 G 中无奇点。

欧拉圈：若存在一个简单圈，过每边一次，且仅一次，则称为欧拉圈。

1、化奇图为偶图，结果如下：



注意：不是将跨越的奇点连接起来！

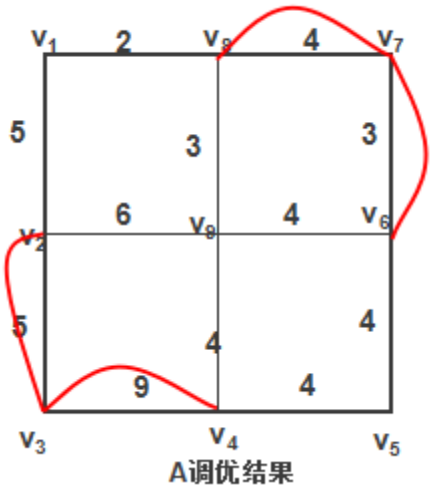
2、但是上面这样的连接结果是最优方案？如果不是最优方案，那么什么样的指标才算是最优的方案？应该如果去找到这样的最优方案？

最后对化奇图为偶图进行下面两个约束条件调优：

A、在最优方案中，图的每一边最多有一条重复边

B、最优方案中，图中的每一个圈（环）重复边的总权值不大于该圈总权值的一半

下面图是经过 A 约束条件的调优结果：



A调优结果

B 约束条件调优算法执行的步骤：

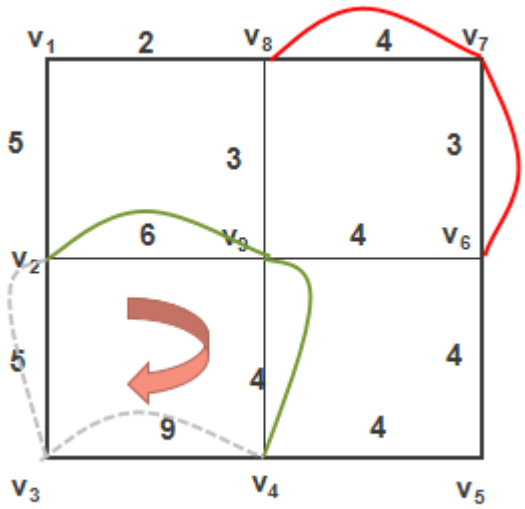
1、找出图中的一个圈，既一条不存在重复路径的回路

2、计算圈中的重复边的总和 D，计算圈的所有边的总和 S

3、如果 D>S/2，则把这个圈的重复边全部变成单边，单边全部变成重复边

4、重复以上的 1,2,3 直到不存在 D > S/2 的情况

最优在 A 的调优结果基础上面，B 调优结果如下：



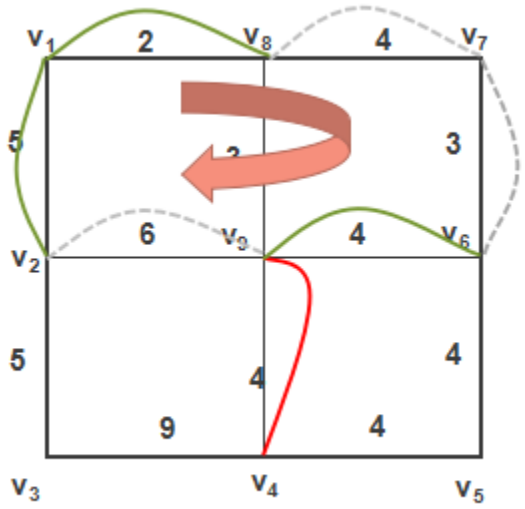
B调优结果

圈（v2,v3,v4,v9,v2）的总长度为 24，但圈上重复边总权为 14，大于该圈总长度的一半，因此可以做一次调整，以[v2,v9]，[v9,v4]上的重复边代[v2,v3]，[v3,v4]上的重复边，使重复边总长度下降为 17。

但是上面的图还存在可以调优的情况：

圈（v1,v2, v9, v6,v7, v8,v1）的总长度为 24，但圈上重复边总权为 13，大于该圈总长度的一半，因此可以做一次调整，使重复边总长度下降为 15。结果如下

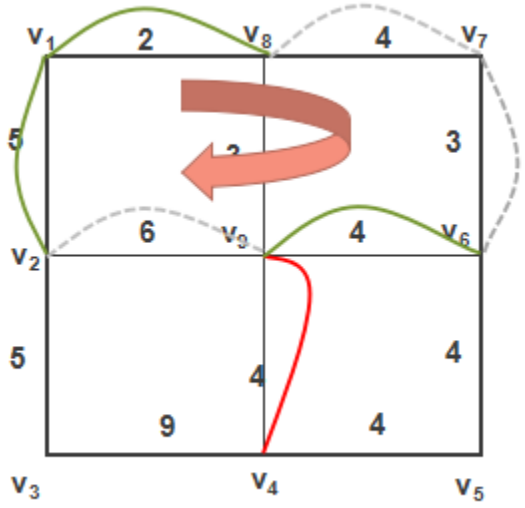
图



B调优结果

最终结果

最优方案：v1-v2-v9-v8-v1-v8-v7-v6-v5-v4-v9-v6-v9-v4-v3-v2-v1（任意一个欧拉圈即可）



B调优结果

4. 算法具体步骤

1）*floyd* 算法

```
//邻接矩阵结构
typedef struct
{
    VertexType vers[MAXVEX];       //顶点表
    EdgeType arc[MAXVEX][MAXVEX];       //邻接矩阵，可看作边表
    int numVertexes, numEdges;       //图中当前的顶点数和边数
}MGraph;
//使用弗洛伊德核心算法，三层循环求解
    for (k = 0; k < G.numVertexes;k++)
    {
        for (i = 0; i < G.numVertexes;i++)
        {
            for (j = 0; j < G.numVertexes;j++)
            {
                if ((*dist)[i][j]>((*dist)[i][k]+(*dist)[k][j])&&i!=j)       //i!=j 使不
更新中间自己到自己的数据和路径
                {
                    //将权值和更新，路径也变为中转点
                    (*dist)[i][j] = (*dist)[i][k] + (*dist)[k][j];
                    (*path)[i][j] = (*path)[i][k];
                }
            }
        }
    }
```

2）最小权匹配算法

```
function [linked,F,PA]=minweightmatch(E)
%-------------------------
% E  为图的邻接矩阵
%-------------------------
```

```
% linked 为相互配对的奇点
% F 为新增的路径权值
% PA 为配对奇点之间的最短路径
%-------------------------
% 找出所有奇点
L=length(E);
JD=mod(sum(E>0 & E<inf),2);
p=find(JD==1);
% 弗洛伊德算法求奇点间最短路
[D,path]=floydf(E);
% 构建奇点间二分图
BinV=inf*ones(length(p),length(p));
for i=1:length(p)
for j=1:length(p)
if i~=j
BinV(i,j)=D(p(i),p(j));
end
end
end
% 令无穷大权值为 9999
BinV(BinV==inf)=9999;
LL=length(BinV);
solution=[];
% 整数规划求解最小权匹配
LL=length(BinV);
Aeq=[];
for i=1:LL
Aeq=[Aeq;zeros(1,(i-1)*LL) ones(1,LL) zeros(1,(LL-i)*LL)];
end
for i=1:LL
```

```
temp=[zeros(i-1,LL);ones(1,LL);zeros(LL-i,LL)];
temp=reshape(temp,[1,LL*LL]);
Aeq=[Aeq;temp];
end
beq=ones(2*LL,1);
for i=1:LL
for j=i+1:LL
tt=zeros(LL,LL);
tt(i,j)=1;tt(j,i)=-1;
tt=reshape(tt,[1,LL*LL]);
Aeq=[Aeq;tt];
beq=[beq;0];
end
end
lb=zeros(LL*LL,1);ub=ones(LL*LL,1);
options = optimoptions('intlinprog');
options.TolInteger=0.001;
[x,fval]=intlinprog(BinV(:),LL*LL,[],[],Aeq,beq,lb,ub,options);
x=reshape(x,[LL,LL]);
for i=1:LL
for j=i+1:LL
if x(i,j)==1;
solution=[solution;p(i) p(j)];
end
end
end
kp=x;kp(x~=0 & x~=1)=0;
kkp=find(sum(kp)==0);
p=p(kkp);
for i=1:2:length(p)
```

```
solution=[solution;p(i) p(i+1)] ;
end
% 构造配对奇点矩阵
linked=solution;
% 新增路径总权值
F=fval/2;
% 配对奇点最短路
[m,n]=size(linked);
PA(1).path=[];
for i=1:m
temp=[linked(i,1)];
while(temp(end)~=linked(i,2))
temp=[temp path(temp(end),linked(i,2))];
end
PA(i).path=temp;
end
```
[11]

5. 性能分析

Floyd 算法适用于 APSP(All Pairs Shortest Paths，多源最短路径)，是一种动态规划算法，稠密图效果最佳，边权可正可负。

此算法简单有效，由于三重循环结构紧凑，对于稠密图，效率要高于执行|V|次 Dijkstra 算法，也要高于执行|V|次 SPFA 算法。

优点：容易理解，可以算出任意两个节点之间的最短距离，代码编写简单。

缺点：时间复杂度比较高 O(n*3)，不适合计算大量数据。[12]

四、旅行推销员问题

1. 问题概述：

旅行推销员问题（英语：Travelling salesman problem,TSP）是这样一个问题：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市

的最短回路。它是组合优化中的一个 NP 难问题，在运筹学和理论计算机科学中非常重要。

最早的旅行商问题的数学规划是由 Dantzig（1959）等人提出，并且是在最优化领域中进行了深入研究。许多优化方法都用它作为一个测试基准。尽管问题在计算上很困难，但已经有了大量的启发式算法和精确方法来求解数量上万的实例，并且能将误差控制在 1%内。[13]

2. 求解算法思想

对原问题进行分析，TSP 的一个解可表述为一个循环排列：[14]

Π= (Π_1，Π_2，Π_3… Π_n），即

$\Pi_1 \rightarrow \Pi_2 \rightarrow \ldots \rightarrow \Pi_n \rightarrow \Pi_1$

有（n-1）！/2 种不同方案，若使用穷举法，当 n 很大时计算量是不可接受的。旅行商问题综合了一大类组合优化问题的典型特征，属于 NP 难题，不能在多项式时间内进行检验。若使用动态规划的方法时间复杂性和空间复杂性都保持为 n 的指数函数。

本次实验利用模拟退火算法（Simulated Annealing）求解 TSP 问题。模拟退火算法最早由 N.Metropolis 等人于 1953 年提出，基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。该算法从某一较高初温出发，伴随温度参数的不断下降，结合概率突跳特性在解空间随机寻找全局最优解。

退火是将固体加热到足够高的温度，使分子呈随机排列态，然后逐步降温冷却，最后分子以低能状态排列，得到稳定状态的固体。退火的过程有：

（1）加温过程：增强粒子运动，消除系统原本可能存在的非均匀态；

（2）等温过程：对于与环境换热而温度不变的封闭系统，系统状态的自发变化总是朝向自由能减少的方向进行，当自由能达到最小时，系统平衡；

（3）冷却过程：使粒子热运动减弱并逐渐趋于有序，系统能量逐渐下降，从而得到低能的晶体结构。

其中，固体在恒温下达到热平衡的过程采用 Metropolis 方法进行模拟：

温度恒定为 T 时，当前状态 i 转为新状态 j，如果 j 状态的能量小于 i，则接受状态 j 为当前状态；否则，如果概率 $p=\exp\{-(E_j-E_i)/(k*T)\}$大于[0,1)区间的随机数，则仍接受状态 j 为当前状态；若不成立则保留状态 i 为当前状态。

温度变化时，由于 $p=\exp\{-(E_j-E_i)/(k*T)\}$，因此在高温下，可接受当前状态能量差较大的新状态；在低温下，只接受与当前状态能量差较小的新状态。

退火过程由冷却进度表(Cooling Schedule)控制，包括控制参数的初值 t 及其衰减因子 Δt、每个 t 值时的迭代次数 L 和停止条件 S。

我们使用模拟退火算法实现 TSP 的参数选取如下：

控制参数初值：$t_0 = 2000$

停止准则：连续 2 个 Mapkob 链中对路径无任何变动（优化或恶化的）时即停止算法运行。

冷却进度表中的控制参数 t 的衰减函数：$\alpha(t)=0.98 * t$

Mapkob 链长：定长 20000

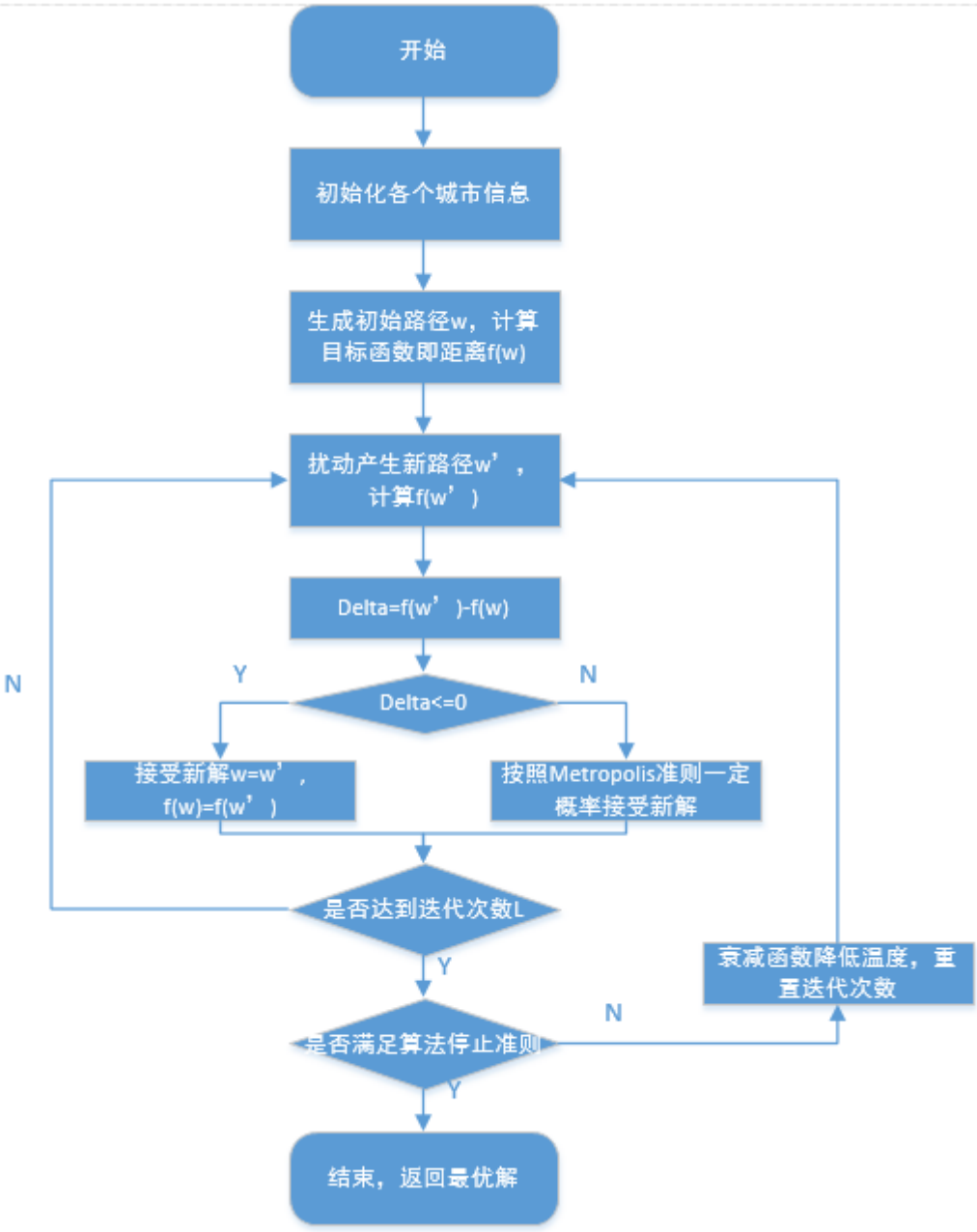
新解的产生：随机交换两个序号。任选序号 u 和 v (u<v)，将 u 和 v 的顺序交换。

（$\Pi_1 \ldots \Pi_{u-1}\Pi_u\Pi_{u+1} \ldots \Pi_{v-1}\Pi_v\Pi_{v+1} \ldots \Pi_n$）

变为

（$\Pi_1 \ldots \Pi_{u-1}\Pi_v\Pi_{u+1} \ldots \Pi_{v-1}\Pi_u\Pi_{v+1} \ldots \Pi_n$）

3. 举例说明求解过程



程序输入：

```
5
    0    8    2    5    8
    8    0   12   14    4
    2   12    0    8   10
    5   14    8    0   18
    8    4   10   18    0
```

程序输出 1：

```
Please input the number of cities:
5
     0     8     2     5     8
     8     0    12    14     4
     2    12     0     8    10
     5    14     8     0    18
     8     4    10    18     0
The best path is:
0 -> 3 -> 1 -> 4 -> 2 -> 0
The distance of the best path is: 35
请按任意键继续. . .
```

程序输出 2：

```
Please input the number of cities:
5
     0     8     2     5     8
     8     0    12    14     4
     2    12     0     8    10
     5    14     8     0    18
     8     4    10    18     0
The best path is:
1 -> 4 -> 2 -> 3 -> 0 -> 1
The distance of the best path is: 35
请按任意键继续. . .
```

程序输出 3：

```
Please input the number of cities:
5
     0     8     2     5     8
     8     0    12    14     4
     2    12     0     8    10
     5    14     8     0    18
     8     4    10    18     0
The best path is:
3 -> 2 -> 4 -> 1 -> 0 -> 3
The distance of the best path is: 35
请按任意键继续. . .
```

4. 算法具体步骤

核心伪代码[15]

```
Begin:
INITIALIZE; {  初始化 i0= Π1 …Πn, t0=2000, L= 20000 }
randomize; {  初始化随机数种子}
Repeat
bChange:=0;
for l:=1 to L do
begin:
GENERATE; {  随机交换两个序号产生新的路径}
CALCULATE(df); {  计算 df= f(j)-f(i)  的值}
if ACCEPT(t, df) then {  根据 Metropolis 准则判断是否接受新的路径}
begin
f: = f + df; {  计算已接受的路径的长度}
bChange:= 1;
end;
end;
t:=t * 0.9; {  衰减函数 α(t)=0.98 * t }
if (bChange= 0) then s:=s+1 else s:=0;
until s = 2 {  停止准则为连续两个 Mapkob 链路径无变化}
End;
```

源码

```
#include <iostream>
#include <time.h>
#include <math.h>
using namespace std;

#define MAX_CITY_NUM 100
```

```
#define T0 2000
#define T 1e-5
#define ALPHA 0.98
#define L 20000

struct path{
	int route[MAX_CITY_NUM];
	double dis;
};

double w[MAX_CITY_NUM][MAX_CITY_NUM];
int num;
double s = 0;
path p0;

void Init_City()
{
	cout << "Please input the number of cities:" << endl;
	cin >> num;
	for (int i = 0; i < num; ++i)
		for (int j = 0; j < num; ++j)
			cin >> w[i][j];
}

void Init_path()
{
	p0.dis = 0;
	for (int i = 0; i < num; ++i)
	{
		p0.route[i] = i;
```

```
            if (i != num - 1)
                p0.dis += w[i][i + 1];
        }
        p0.dis += w[num - 1][0];
}

path generate(path p)
{
        int x = 0, y = 0;
        while (x == y)
        {
            x = (int)(num * (rand() / (RAND_MAX + 1.0)));
            y = (int)(num * (rand() / (RAND_MAX + 1.0)));
        }
        path gen = p;
        int tmp;
        tmp = gen.route[x];
        gen.route[x] = gen.route[y];
        gen.route[y] = tmp;

        gen.dis = 0;
        for (int i = 0; i < num - 1; ++i)
            gen.dis += w[gen.route[i]][gen.route[i + 1]];
        gen.dis += w[gen.route[num - 1]][gen.route[0]];
        return gen;
}

void TSP_SA()
{
        double t = T0;
```

```
srand(time(NULL));
path cur = p0;
path next = p0;
int bChange;
while (t > T)
{
	bChange = 0;
	for (int i = 0; i < L; ++i)
	{
		next = generate(cur);
		double df = next.dis - cur.dis;
		if (df <= 0)
		{
			cur = next;
			bChange = 1;
		}
		else
		{
			double rndp = rand() / (RAND_MAX + 1.0);
			double eps = exp(-df / t);
			if (eps > rndp && eps < 1)
			{
				cur = next;
				bChange = 1;
			}
		}
	}

	if (cur.dis < p0.dis)
		p0 = cur;
```

```
            t *= ALPHA;
            if (!bChange)
                ++s;
            else
                s = 0;

            if (s == 2)
                break;
        }
}

int main()
{
        Init_City();
        Init_path();
        TSP_SA();
        cout << "The best path is:" << endl;
        for (int i = 0; i < num; ++i)
                cout << p0.route[i] << " -> ";
        cout << p0.route[0] << endl;
        cout << "The distance of the best path is: " << p0.dis << endl;
        system("pause");
        return 0;
}
```

5. 性能分析

本次实验采用模拟退火算法解决 TSP 问题，该算法具有超越算法本身的启发式意义。模拟退火算法将自然物理过程中的固体退火过程与组合优化问题进行类比，成功地将退火过程迁移用于组合优化问题，不仅应用范围广、使用灵活，而且初

始化条件少。但是收敛速度较慢，运行时间较长。模拟退火算法是在状态空间搜索近似最优解的过程，由于搜索过程中会以一定概率接受更差的结果，所以能够跳出局部最优状态范围。

参考资料

1. 许卓群.数据结构与算法[M].高等教育出版社,2004.
2. 彭军、向毅.数据结构与算法[M].人民邮电出版社,2013.
3. Budd T.经典数据结构（Java 语言版）[M].清华大学出版社,2005.
4. 王晓东.算法设计与分析[M].清华大学出版社,2003.5.
5. 严蔚敏.数据结构 C 语言版[M].清华大学出版社,2007.
6. 王广芳.数据结构算法与应用-C++语言描述[M].机械工业出版社,2006.
7. 严蔚敏等.数据结构题集(C 语言版)[M].清华大学出版社,1999.2.
8. 严太山，郭观七，李文彬.关于中国邮递员问题的一种解法[J].湖南理工学院学报：自然科学版，2015（1）：81-83.
9. Frank M.Carrano.数据结构与 C++高级教程[M].清华大学出版社,2004.
10. 南淑萍.关于数据结构中的图论[J].湖北科技学院学报，2014（10）：13-14.
11. 贾丹，周军.基于 C 语言的图论[J].辽宁工业大学学报：社会科学版，2015（2）：132-134
12. 高贤强，化希耀，陈立平.引入计算思维的《数据结构》教学改革研究[J].现代计算机：专业版，2015（7）：16-19.
13. Robert Sedgewick,Kevin Wayne.算法(第 4 版)[M].人民邮电出版社,2012.
14. 张爱平,赖欣.JSP 调用 JavaBean 实现 Web 数据库访问[J].计算机时代,2007,(01).
15. 韩世芬.现代计算机[J].科技资讯,2006,(17).