图结构研讨

第四次讨论课

经典算法设计技术研讨

目录

CONTENTS

哈密尔顿问题

1．问题概述

问题引入

火星运河悖论

X国的人造火星卫星发现了火星上的运河水道还有20个城市遗址，如图1所示。每个城市用一个拉丁字母来代表。最南边的T是火星的“南极城”。

X国《天地指南报》刊登了如下悬赏100万美元征解的题目：从某个火星城出发，沿运河水路而行，每个城必须经过而且只经过一次，并且所经过的城市的代表字母恰好能拼写成一句话。问，是否有这样的途径?如果有，请把它画出来 [2] 。

《天地指南报》编辑很快收到5万多封读者来信，都回答“不可能存在这样的途径”(There is no possible way)。

问题引入

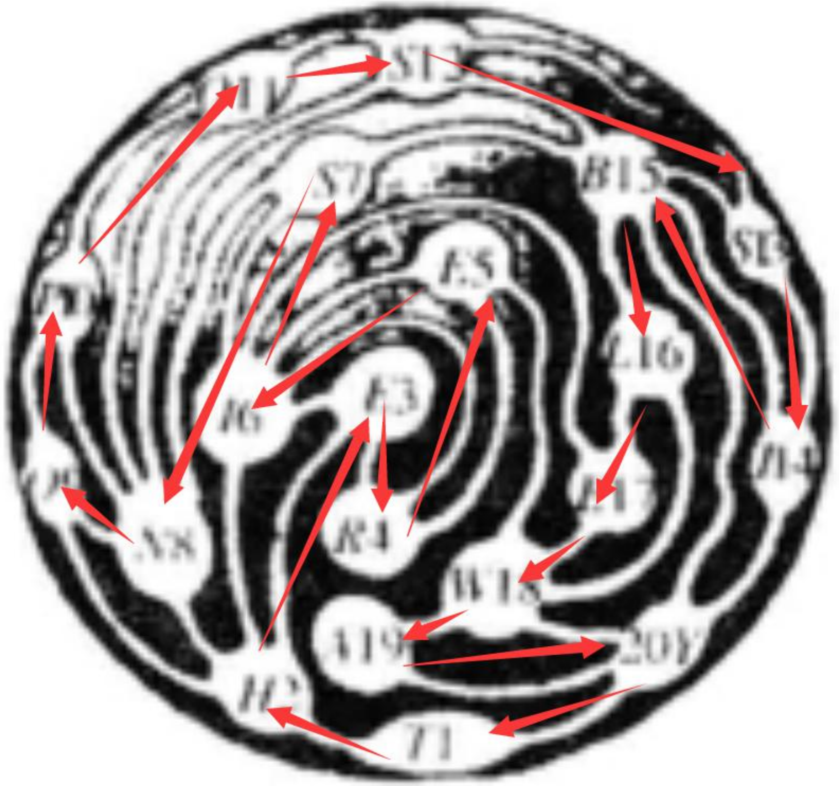
读者的答案是“正确”的：

图1中已用1～20标出了这个途径——从“南极城”T出发到达终点y。由一路上所经过的城市的代表字母，就拼写成了“There is no possible way”，其含义是“不可能存在这样的途径”；

但是，这句话的意思就是有这样的途径，即这样的途径是存在的。

从字面意思来看的话，“不可能存在这样的途径”(There is no possibleway)表示不存在题目中要求的途径；而事实上又存在这样的途径，这就导致了自相矛盾的局面。这就是所谓的“火星运河悖论”。

读者回答的“不可能存在这样的途径”，是这个图1上的一条“哈密顿轨道”。如果从y再走到“南极城”T，则从“南极城”出发又回到了“南极城”，这又是一个“哈密顿圈”。这样，图1就是“哈密顿图”。



夸夸我，我真的画出来了

问题引入

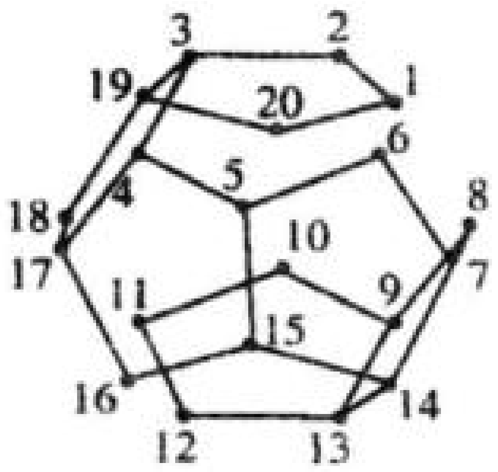
1859年，英国数学家、物理学家威廉·罗恩·哈密顿(1805～1865)，提出了以下“周游世界游戏”，

据说公布在当地市场上：用图2那样的一个正12面体的20个顶点来表示地球上的20个城市，如何才能从某个城市出发，沿着各条棱走正好只经过每个城市一次，最后返回到出发地点?

哈密顿的问题，被他简化为图3的左或右所示的“棋盘”平面图形问题。哈密顿还自豪地用棋盘做了形象明了的说明：“12面遨游，单身周游列国游戏。本玩具是钦命的爱尔兰天文学博士、爵士威廉·罗恩·哈密顿的发明。宴席上，作为即兴表演，稀奇无比。”

因为是皇帝钦命的天文学博士发明的玩具，大家都十分感兴趣。于是，当时英伦三岛掀起了一股“单身周游列国”热 [2] 。

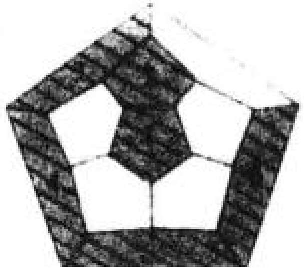
最后，这个问题已由哈密顿本人解决。他的答案如图4所示，与前面的“不可能存在这样的途径”本质上相同。图4中从1→20→1，就形成了一个哈密顿圈，即哈密顿图。已经证明，除此之外采用别的本质不同的方式，是不能按要求周游世界的。



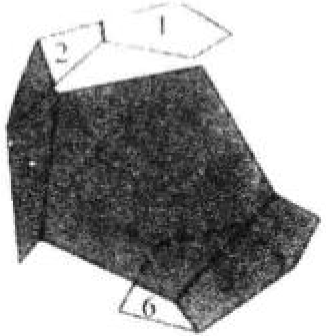
问题引入

不只外国人钟情“单身周游列国”，中国著名数学家苏步青(1902-2003)也是“爱好者”之一。

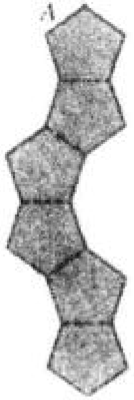
苏步青从图3右边“周游世界棋盘”里的12个大大小小的五边形中，挑出了6个(图5中画有斜线的那6个)，这6个五边形在原正12面体中的位置如图6所示。再把图6所示的6个五边形“摊平”，就得到图7那样的有20个顶点的20边形。



哈密顿和苏步青把一个12面体“压扁”、变形后再进行研究的方法，值得我们深思。



“周游世界棋盘”问题是哈密顿问题的特例，其对象后来也扩展为一般的m×n棋盘上走马步的问题。用电子计算机研究之后，目前的成果有：



对任意奇数的m，n，m×n，棋盘上不存在马的哈密顿同路；

国际象棋8 x 8的棋盘上至少存在10条哈密顿回路；

中国象棋9×10的棋盘上至少存在300条哈密顿回路。

这些问题，包括中国数学工作者在内的许多学者仍在不断地寻找解决方法。

问题意义

要判定一个图是否具有哈密顿圈的问题，是图论中著名的难题之一。除个别情形以外，迄今为止还没有找到一个图是否具有哈密顿圈的**必要而且充分**的条件。

哈密顿圈问题引出了诸如**货郎问题、邮递员问题**等类似的问题。比如，货郎问题就是货郎必须到每个村庄售货，怎样走才能使路程最短?当然，这个问题因为还要求“路程最短”，比哈密顿圈问题难度更大，以致用现代电子计算机来解决都很复杂。

这类问题的研究，促进了最优化方法、图论等问题的研究，促进了运筹学、拓扑学等学科的发展。

哈密尔顿（1805~1865），爱尔兰数学家、物理学家，对四元数有很大的贡献

1、问题概述及相关术语解释

问题背景

1859年，爱尔兰数学家哈密尔顿（Hamilton）提出下列周游世界的游戏：

在正十二面体的二十个顶点上依次标记伦敦、巴黎、莫斯科等世界著名大城市，正十二面体的棱表示连接这些城市的路线。

试问能否在图中做一次旅行，从顶点到顶点，沿着边行走，经过每个城市恰好一次之后再回到出发点。

在无向图G=<V,E>中，

哈密尔顿路径（Hamiltonian path）
遍历G中每个顶点一次且仅一次的路径，（不一定形成回路，也被称作“哈密尔顿通路”）

哈密尔顿回路（Hamiltonian cycle）
遍历G中每个顶点一次且仅一次的回路

哈密尔顿图（Hamiltonian graph，Traceable graph）
具有哈密尔顿回路的图

半哈密顿图
具有哈密顿通路但不具有哈密顿回路的图

哈密尔顿（1805~1865），爱尔兰数学家、物理学家，对四元数有很大的贡献

2、同类问题的比较

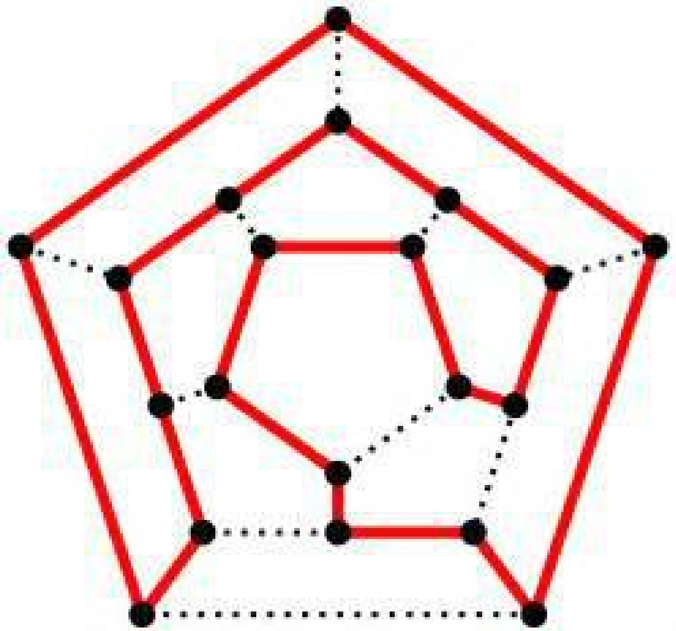
18世纪初普鲁士的哥尼斯堡，有一条河穿过，河上有两个小岛，有七座桥把两个岛与河岸联系起来（如概述图）。

有个人提出一个问题：一个步行者怎样才能不重复、不遗漏地一次走完七座桥，最后回到出发点。后来大数学家欧拉把它转化成一个几何问题——一笔画问题。

他不仅解决了此问题，且给出了连通图可以一笔画的充要条件是：**奇点的数目不是0个就是2个**（连到一点的数目如是奇数条，就称为奇点，如果是偶数条就称为偶点，要想一笔画成，必须中间点均是偶点，也就是有来路必有另一条去路，奇点只可能在两端，因此任何图能一笔画成，**奇点要么没有要么在两端**）

哥尼斯堡七桥问题

哥尼斯堡七桥问题
寻找一条遍历图中所有边的简单路径，
哈密尔顿的周游世界问题
寻找一条遍历图中所有点的基本路径，


哈密尔顿的周游世界问题

2 . 算法思想

2.1 必要条件与充分条件

1.必要非充分条件：

若G=(V,E) 是一个哈密顿图，则对于V的每一个非空子集S，均有W(G - S) ≤|S|。其中|S|是S中的顶点数，W(G - S)表示图G擦去属于S中的顶点后，剩下子图的连通分支的个数。

2.充分非必要条件：

设G=(V,E)是一个无向简单图，|V|=n. n≥3. 若对于任意的两个顶点u，v∈ V，d(u)+d(v) ≥n，那么, G是哈密尔顿图。此条件由美国图论数学家奥勒在1960年给出。

2.2 P与NP问题

1.P问题：

P问题是具有多项式算法的判定问题。这里的P代表Polynomial。P问题就是可以有一个确定型图灵机在多项式时间内解决的问题。即那些存在O（n），O（nk），O（nlogn）等多项式时间复杂度解法的问题。比如排序问题、最小生成树、单源最短路径。直观的讲，我们将P问题视为可以较快解决的问题。

2.NP问题：

与之相对应的，还有N P类，NP类是具有多项式时间验证机的语言类，其中验证机的定义如下：语言A的验证机是一个算法，其中A = { w ∣ 对 某 个 字 符 串 c , V 接 受 < w , c > }

验证机是什么呢，其实是一个算法，假设代号为V，用于验证语言A，则：
A={w|对某个字符串c，V接受<w,c>}注意是“某个”这里的w你可以看做是“判定问题的输入”，比如一张无向图，而c可以看做是其中某个路径，只要w,c输入给V这个算法，返回true，就ok，至于c有N!个还是很少，跟A是w的集合没啥关系，如果V是多项式的，则A属于NP。

2.2 P与NP问题

3.多项式时间可验证性：

因为只根据w的长度来度量验证机的时间，所以多项式时间验证机在w的长度的多项式时间内运行。如果语言A有一个多项式时间验证机，我们就称它是多项式时间可验证的。

4.成员与成员资格证书：

验证机利用额外的信息（即上述定义中的符号 c ）来验证字符串 w 是 A 的成员。该信息称为 A的成员资格证书．或证明。注意，对于多项式验证机，证书具有多项式的长度（ w 的长度），因为这是该验证机在它的时间界限内所能访问的全部信息长度。

5.N P

N P 的意思就是非确定型多项式时间，这也是使用非确定型多项式时间图灵机的一个特征。一个非常重要的定理就是：一个语言在 NP 中，当且仅当它能被某个非确定型多项式时间图灵机判定。

2.3 判定方法

2.3.1 基本必要条件

设图G = < V , E >是哈密顿图,则对于v vv的任意一个非空子集S,
若以 |S|表示S中元素的数目,G – S表示G中删除了S中的点以及这些点所关联的边后得到的子图,
则W(G−S)≤|S|成立.其中W(G−S)是G−S中联通分支数。

2.3.2 Dirac定理(充分条件)

设一个无向图中有N个顶点,若所有顶点的度数大于等于N/2,则哈密顿回路一定存在.(N/2指的是⌈N/2⌉,向上取整)

2.3.3 竞赛图(哈密顿通路)

N(N>=2)阶竞赛图一点存在哈密顿通路.

3 . 求解过程及性能分析

方法1. DFS搜索求哈密尔顿回路

方法1：DFS搜索求哈密尔顿回路

```
using namespace std;
int n,m;
int u,v;
int g[N][N];
int vis[N],appear[N];
int ans[N],num[N];
int length;
int x;
int main(){
    memset(vis,0,sizeof(vis));
    memset(appear,0,sizeof(appear));
    cin>>n>>m; //读入点数与边数
    for(int i=1;i<=m;i++){
        cin>>u>>v; //读入两点
        g[u][++num[v]]=v;//记录u-v的边
        g[v][++num[v]]=u;//记录v-u的边
    }
    for(x=1;x<=n;x++) //枚举每一个点，将其作为起点来尝试访问
    {
        if(!appear[x])//如果点x不在之前曾经被访问过的图里
        {
            length=0;//记录答案的长度
            dfs(0,x);
        }
    }
    return 0;
}
```

方法1：DFS搜索求哈密尔顿回路

```
void dfs(int last,int i)//last表示上次访问的点
{
    vis[i]=1;//标记为已经访问过
    appear[i]=1;//标记为已在一张图中出现过
    ans[length++]=i;//记录答案
    for(int j=1;j<=num[i];j++)
    {
        if(g[i][j]==x&&g[i][j]!=last)//回到起点构成哈密顿环
        {
            ans[++length]=g[i][j];//存储答案
            for(int i=1;i<=length-1;i++) //找到了一个环，输出ans
                cout<<ans[i]<<' ';
            cout<<ans[length]<<endl;
            length--;//长度-1
            break;
        }
        if(!vis[g[i][j]])//遍历与i相关联的所有未访问的点。
            dfs(i,g[i][j]);
    }
    length--;
    vis[i]=0;//回溯
}
```

方法2. Dirac 定理下构造无向图的哈密顿回路

方法2：Dirac 定理下构造无向图的哈密顿回路

2.哈密尔顿回路的判定：Dirac 定理

设一无向图有 n 个顶点，u、v 为图中任意不相邻的两点，deg(x) 代表 x 的度数

若 $deg(u)+deg(v)\geqslant n$ ，则图中存在哈密尔顿回路

推论：对于 $n\geqslant 3$ 的无向图，若其任一一点 u 的度数 $deg(u)\geqslant \frac{n}{2}$ ，则图中存在哈密尔顿回路

方法2：Dirac 定理下构造无向图的哈密顿回路

1、思维过程

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

若 S 与 T 相邻，则路径 S -> T 形成了一个回路

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.

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

方法2：Dirac 定理下构造无向图的哈密顿回路

2.伪代码

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

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]，进入步骤 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

方法2：Dirac 定理下构造无向图的哈密顿回路

3.时间复杂度

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)

空间复杂度
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

方法2：Dirac 定理下构造无向图的哈密顿回路

4.代码-初始化及辅助函数

```
#include<bits/stdc++.h>
#define INF 0x3f3f3f3f
]#define N 1001
#define E 1e-6
#define LL long long
using namespace std;
bool G[N][N];
bool vis[N];
int ans[N];
void Reverse(int arv[N],int s,int t){//将数组anv从下标s到t的部分的顺序反向
    int temp;
    while(s<t){
        swap(arv[s],arv[t]);
        s++;
        t--;
    }
}
```

```
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    n*=2;
    for(int i=0;i<=n;i++){
        for(int j=0;j<=n;j++){
            if(i==j){
                G[i][j]=false;
                G[j][i]=false;
            }
            else{
                G[i][j]=true;
                G[j][i]=true;
            }
        }
    }
    int ansi=0;
    memset(ans, 0, sizeof(ans));
    for(int i=1;i<=m;i++){
        int x,y;
        scanf("%d%d",&x,&y);
        G[y][x]=false;
        G[x][y]=false;
    }
    Hamilton(n);
    for(int i=0;i<n;i++)
        printf("%d ", ans[i]);
    printf("\n");
    return 0;
}
```

方法2：Dirac 定理下构造无向图的哈密顿回路

```
void Hamilton(int n){
    int t;
    int s=1;//初始化取s为1号点
    for(int i=1;i<=n;i++)
        if(G[s][i]){
            t=i;//取任意邻接与s的点为t
            break;
        }
    memset(vis,false,sizeof(vis));
    vis[s]=true;
    vis[t]=true;
    ans[0]=s;
    ans[1]=t;
    int ansi=2;
    while(true){
        //从t向外扩展
        while(true){
            int i;
            for(i=1;i<=n;i++){
                if(G[t][i] && !vis[i]){
                    ans[ansi++]=i;
                    vis[i]=true;
                    t=i;
                    break;
                }
            }
            if(i>n)
                break;
        }
        //将当前得到的序列倒置
        Reverse(ans,0,ansi-1);
        //s和t互换
        swap(s,t);
```

```
while(true){//从t继续扩展,相当于在原来的序列上从s向外扩展
    int i;
    for(i=1;i<=n;i++){
        if(G[t][i] && !vis[i]){
            ans[ansi++]=i;
            vis[i]=true;
            t=i;
            break;
        }
    }
    if(i>n)
        break;
}
```

```
        //如果s和t不相邻,进行调整
        if(!G[s][t]){
            //取序列中的一点i,使得ans[i]与t相连,并且ans[i+1]与s相连
            int i;
            for(i=1;i<ansi-2;i++)
                if(G[ans[i]][t]&&G[s][ans[i+1]])
                    break;
            i++;
            t=ans[i];
            Reverse(ans,i,ansi-1);//将从ans[i+1]到 t 部分的ans[]倒置
        }//此时s和t相连
        //如果当前序列包含n个元素,算法结束
        if(ansi==n)
            return;
        //当前序列中元素的个数小于n,寻找点ans[i],使得ans[i]与ans[]外的一个点相连
        int i,j;
        for(j=1;j<=n;j++){
            if(vis[j])
                continue;
            for(i=1;i<ansi-2;i++)
                if(G[ans[i]][j])
                    break;
            if(G[ans[i]][j])
                break;
        }
        s=ans[i-1];
        t=j;//将新找到的点j赋给t
        Reverse(ans,0,i-1);//将ans[]中s到ans[i-1]的部分倒置
        Reverse(ans,i,ansi-1);//将ans[]中ans[i]到t的部分倒置
        ans[ansi++]=j;//将点j加入到ans[]尾部
        vis[j]=true;
    }
}
```

角度3. N 阶竞赛图下构造有向图的哈密顿通路

方法3：N 阶竞赛图下构造有向图的哈密顿通路

竞赛图

每对顶点之间都有一条边相连的有向图，
n 个顶点的竞赛图称为 n 阶竞赛图。

可以证明：
含有N个顶点的有向图，且每对顶点之间都有一条边的图，一定存在哈密顿通路
（证明过程在下一页）

方法3：N 阶竞赛图下构造有向图的哈密顿通路

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.

方法3：N 阶竞赛图下构造有向图的哈密顿通路

```
int ans[105];
int map[105][105];
void Insert(int arv[], int &len, int index, int key){
    if(index>len)
        index=len;
    len++;
    for(int i=len-1; i>=0; i--){
        if(i!=index && i)
            arv[i]=arv[i-1];
        else{
            arv[i]=key;
            return;
        }
    }
}
void Hamilton(int n){
    int ansi = 1;
    ans[ansi++] = 1;
    for(int i=2; i<=n; i++){//第一种情况,直接把当前点添加到序列末尾
        if(map[i][ans[ansi-1]]==1)
            ans[ansi++]=i;
        else{
            int flag=0;
            //当前序列从后往前找到第一个满足条件的点j,使得存在<Vj,Vi>且<Vi,Vj+1>.
            for(int j=ansi-2; j>0; j--){
                if(map[i][ans[j]]==1){//找到后把该点插入到序列的第j+1个点前.
                    flag=1;
                    Insert(ans,ansi,j+1,i);
                    break;
                }
            }
            if(!flag)//否则说明所有点都邻接自点i,则把该点直接插入到序列首端.
                Insert(ans,ansi,1,i);
        }
    }
}
```

```
int main(){
    int n,m;
    scanf("%d", &n);
    m=n*(n-1)/2;
    for(int i=0;i<m;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        if(u<v)
            map[v][u]=1;
    }
    Hamilton(n);
    for(int i=1;i<=n;i++)
        printf(i==1? "%d":" %d",ans[i]);
    printf("\n");
    return 0;
}
```

角度4. 状压DP求最短Hamilton

方法4：状压DP求最短Hamilton

给定一张 n 个点的带权无向图，点从 $0 \sim n-1$ 标号，求起点 0 到终点 $n-1$ 的最短 Hamilton 路径。

Hamilton 路径的定义是从 0 到 $n-1$ 不重不漏地经过每个点恰好一次。

输入格式

第一行输入整数 n。

接下来 n 行每行 n 个整数，其中第 i 行第 j 个整数表示点 i 到 j 的距离（记为 $a[i,j]$）。

对于任意的 x,y,z，数据保证 $a[x,x]=0$，$a[x,y]=a[y,x]$ 并且 $a[x,y]+a[y,z] \geq a[x,z]$。

输出格式

输出一个整数，表示最短 Hamilton 路径的长度。

数据范围

$1 \leq n \leq 20$
$0 \leq a[i,j] \leq 10^7$

输入样例：

```
5
0 2 4 5 1
2 0 6 5 3
4 6 0 8 3
5 5 8 0 5
1 3 3 5 0
```

输出样例：

```
18
```

方法4：状压DP求最短Hamilton

1.本题思路

假设:一共有七个点,用0,1,2,3,4,5,6来表示,那么先假设终点就是5,在这里我们再假设还没有走到5这个点,且走到的终点是4,那么有以下六种情况:

first: 0–>1–>2–>3–>4 距离:21

second: 0–>1–>3–>2–>4 距离:23

third: 0–>2–>1–>3–>4 距离:17

fourth: 0–>2–>3–>1–>4 距离:20

fifth: 0–>3–>1–>2–>4 距离:15

sixth: 0–>3–>2–>1–>4 距离:18

如果此时你是一个商人你会走怎样的路径?显而易见,会走第五种情况对吧?因为每段路程的终点都是4,且每种方案的可供选择的点是0~4,而商人寻求的是走到5这个点的最短距离,而4到5的走法只有一种,所以我们选择第五种方案，可寻找到走到5这个点儿之前,且终点是4的方案的最短距离,此时0~5的最短距离为(15+4走到5的距离).(假设4–>5=8)

方法4：状压DP求最短Hamilton

同理:假设还没有走到5这个点儿,且走到的终点是3,那么有一下六种情况:
first: 0–>1–>2–>4–>3 距离:27
second: 0–>1–>4–>2–>3 距离:22
third: 0–>2–>1–>4–>3 距离:19
fourth: 0–>2–>4–>1–>3 距离:24
fifth: 0–>4–>1–>2–>3 距离:26
sixth: 0–>4–>2–>1–>3 距离:17

此时我们可以果断的做出决定:走第六种方案!!!,而此时0~5的最短距离为(17+3走到5的距离)(假设3–>5=5)

在以上两大类情况之后我们可以得出当走到5时:
1.以4为终点的情况的最短距离是:15+8=23;
2.以3为终点的情况的最短距离是:17+5=22;
经过深思熟虑之后,商人决定走以3为终点的最短距离,此时更新最短距离为:22。

当然以此类推还会有以1为终点和以2为终点的情况,此时我们可以进行以上操作不断更新到5这个点的最短距离,最终可以得到走到5这个点儿的最短距离,然后再返回最初的假设,再依次假设1,2,3,4是终点,最后再不断更新,最终可以得出我们想要的答案

方法4：状压DP求最短Hamilton

一、状态表示： `dp[i][j]` 表示从顶点0到顶点j，且经过的顶点的集合为i的所有路径。
`dp[i][j]` 的值表示这些所有路径的和的最小值。

i 就是一个状态表示，用二进制数表示，假设i=10011，根据下标，低位到高位分别是0到4，即经过了顶点0、1、4。

二、状态计算： 根据 `dp[i][j]` 的含义，如果集合i的顶点里不包含终点j和起点0，那么这个值就没有含义，规定为无穷大。

> 例如，假如集合 i 等于**11** $_{(2)}$，则表示该状态经过顶点0和顶点1，即直接求出0->1的距离即可。
> 如果集合i表示的状态是**100** $_{(2)}$，那就表示该路径经过的顶点不过起点0，可直接判断为不存在。

因此求 `dp[i][j]` 的值，需要在状态i表示的集合经过起点0的情况下，即 `i&1!=0`，去掉集合i中的顶点j，得到集合t，即 `t=i-{j}`，然后在集合t中寻找新的终点k，这时有 `dp[t][k]`，还需要顶点k到顶点j的代价 `g[k][j]`，这时只需要取 `min(dp[i][j], dp[t][k]+g[k][j]);` 就好了。

二进制i表示的十进制数肯定是要大于二进制t表示的十进制数的，求后面的状态 `dp[i][j]` 时要用到前面的状态 `dp[t][k]`，所以按照状态更新的拓扑序，**应先枚举状态，再枚举到达的点。**

即，对于二维表dp，**应该按行更新，先求出每种路径状态i下可到达各个终点j的距离。**

> 举个例子，按照我们的逻辑，在计算 `f[101][1]` （101是二进制下的数，点的编号从000开始）的时候，我们中间会用到 `f[001][2] + g[2][1]` 来更新该状态。 不会存在 **f[111][1]** 这样数据，求到达顶点1时，顶点1已经从集合i中去掉了
> **如果先枚举到达的点的话**，我们会先计算 `f[101][1]`，再计算 `f[001][2]`。那么我们在用 `f[001][2]` 更新 `f[101][1]` 的时候，由于 `f[001][2]` 还没计算过，所以还是正无穷，那么更新的 `f[101][1]` 的值就是错误的。

最终结果：就是经过所有顶点，即状态111……11，且到达的顶点是n-1，即 `dp[(1<<n)-1][n-1]`。

方法4：状压DP求最短Hamilton

三、初始化问题

`dp[0][j]`，状态i为0，表示集合里不包括任何顶点，即 0->j 一个顶点也不经过，显然不可能，初始化为正无穷。`dp[0][j]=INF;`

`dp[i][0]`，说明经过集合i的顶点后到达顶点0，

1. 若i为1，只有顶点0被选中，说明集合i中只有顶点0，那么可以初始化为1；`dp[1][0]=1;`
2. 若i大于0，说明在经过一系列顶点后还要到达顶点0，显然不可能，初始化为正无穷。`dp[i][0]=INF`

状态i等于二进制数1说明只经过顶点0，终点也为顶点为0，0->0的距离为0，且只能更新`dp[1][0]`这一个距离，

> **几个特殊情况：**
> 假如路径的状态i是2^k时，即100……0，必不包括顶点0，所以到任何顶点的距离都不会更新。
> 假如路径的状态i是(2^k)+1时，即100……01，只包含顶点0和顶点k，那么只会更新0->k的直接距离，0为终点，k为起点，不经过其它顶点。

方法4：状压DP求最短Hamilton

2.DP分析:

用二进制来表示要走的所以情况的路径,这里用i来代替

例如走0,1,2,4这三个点,则表示为:10111;

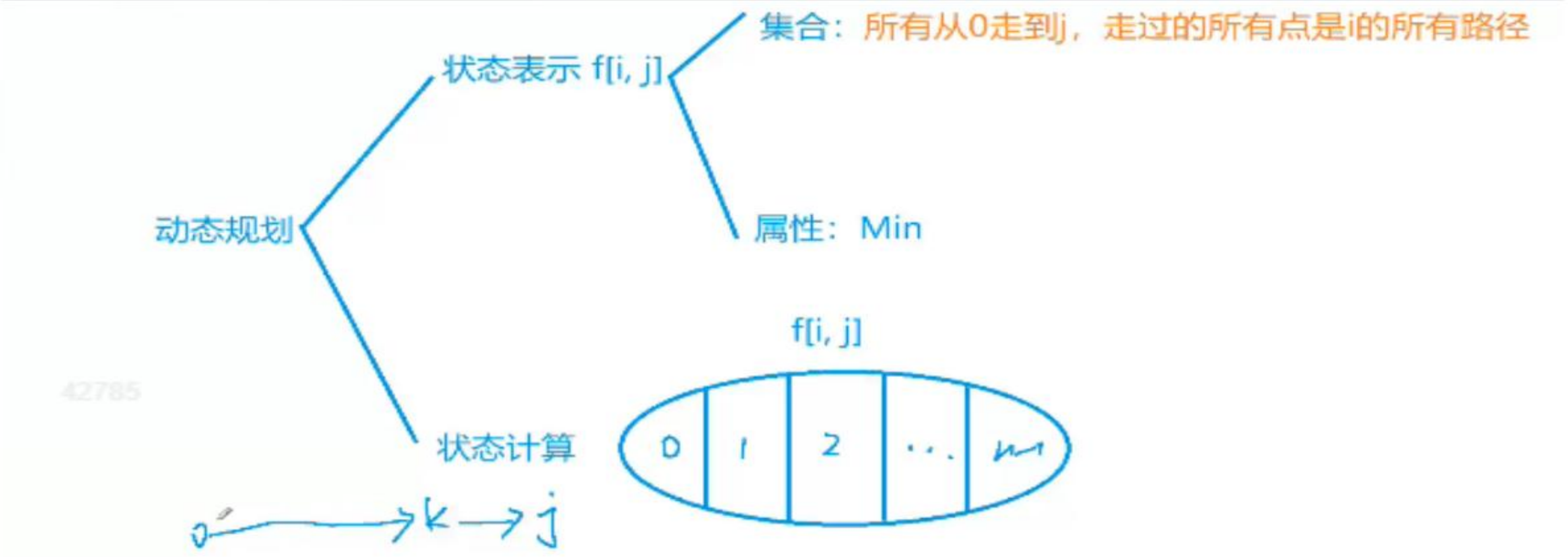
走0,2,3这三个点:1101;

状态表示:f[i][j];

集合:所有从0走到j,走过的所有点的情况是i的所有路径

属性:MIN

状态计算:如1中分析一致,0–>······–>k–>j中k的所有情况



状态转移方程:f[i][j]=min(f[i][j],f[i-(1<<j)][k]+w[k][j])

方法4：状压DP求最短Hamilton

```
#include<iostream>
#include<cstring>
#include<algorithm>

using namespace std;

const int N=20,M=1<<N;

int f[M][N],w[N][N];//w表示的是无权图

int main()
{
    int n;
    cin>>n;

    for(int i=0;i<n;i++)
     for(int j=0;j<n;j++)
      cin>>w[i][j];

    memset(f,0x3f,sizeof(f));//因为要求最小值，所以初始化为无穷大
    f[1][0]=0;//因为零是起点,所以f[1][0]=0;

    for(int i=0;i<1<<n;i++)//i表示所有的情况
     for(int j=0;j<n;j++)//j表示走到哪一个点
      if(i>>j&1)
       for(int k=0;k<n;k++)//k表示走到j这个点之前,以k为终点的最短距离
        if(i>>k&1)
         f[i][j]=min(f[i][j],f[i-(1<<j)][k]+w[k][j]);//更新最短距离

    cout<<f[(1<<n)-1][n-1]<<endl;//表示所有点都走过了,且终点是n-1的最短距离
    //位运算的优先级低于'+'-'所以有必要的情况下要打括号
    return 0;
}
```

参考资料

CSDN博主「目義」https://blog.csdn.net/qq_45667304/article/details/121390309

知乎-冒泡https://www.zhihu.com/question/67578069/answer/254894874

https://www.luogu.com.cn/blog/123-day/ha-mi-dun-wen-ti

https://www.acwing.com/solution/content/18533/

THANKS!