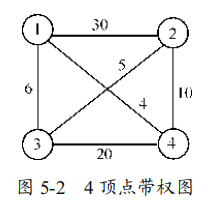
**实验报告5**

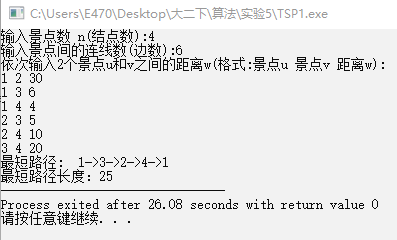
1.请用回溯法和分支限界法求对称的旅行商问题（TSP 问题）。

测试数据为教材图 5-2中的4顶点带权图。



①回溯法

结果：



代码：

#include <iostream>

#include <algorithm>

using namespace std;

const int NoEdge=1e4;//无边标记-无穷大

const int N=10;

class Traveling{

public:

void TSP(int \*\*,int);

void print();

private:

void Backtrack(int);

int n, //图G的顶点数

\*x, //当前解

\*bestx, //当前最优解

\*\*g, //图G的邻接矩阵

l, //当前费用

bestl; //当前最优值

};

void Traveling::TSP(int \*\*g1,int n1){//初始化

n=n1;

x=new int [n+1];

bestx=new int [n+1];

for(int i=0;i<=n1;i++){

x[i]=i;

bestx[i]=0;

}

g=g1;

l=0;

bestl=NoEdge;

Backtrack(2); //搜索x[2:n]的全排列

}

//核心函数算 法复杂度为：O(n!)

void Traveling::Backtrack(int i){

if(i>n){ //排列结束

if(g[x[n]][1]!=NoEdge && (l+g[x[n]][1]<bestl)) {

for(int j=1;j<=n;j++) bestx[j]=x[j];

bestl=l+g[x[n]][1];

}

}else{

for(int j=i;j<=n;j++){//可否进入x[j]子树,可则搜索子树

if(g[x[i-1]][x[j]]!=NoEdge && (l+g[x[i-1]][x[j]]<bestl)){

swap(x[i],x[j]);

l+=g[x[i-1]][x[i]];

Backtrack(i+1);

l-=g[x[i-1]][x[i]];

swap(x[i],x[j]);

}

}

}

}

void Traveling::print(){//打印路径

cout<<"最短路径: ";

for(int i=1;i<=n;i++)

cout<<bestx[i]<<"->";

cout<<"1"<<endl;//回到起点结束

cout<<"最短路径长度："<<bestl;

}

int main(){

int u,v,w,n,m;//u,v：城市 w：u和v城市间的路长

cout<<"输入景点数 n(结点数):";

cin>>n;

cout<<"输入景点间的连线数(边数):";

cin>>m;

cout<<"依次输入2个景点u和v之间的距离w(格式:景点u 景点v 距离w):"<<endl;

int \*\*g;

g=new int \*[N];//初始一个N行N列的矩阵

for(int i=0;i<N;i++){

g[i]=new int[N];

}//注意二维指针的初始化！！！

for(int i=1;i<=n;i++)//初始化G图

for(int j=i;j<=n;j++)

g[i][j]=g[j][i]=NoEdge;//表示路径不可达

for(int i=1;i<=m;i++){//更新G图

cin>>u>>v>>w;

g[u][v]=g[v][u]=w;

}

Traveling Y;

Y.TSP(g,n);

Y.print();

return 0;

}

体会：

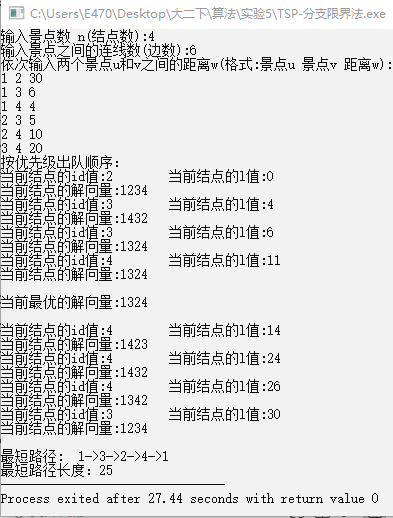
时间复杂度为：O(n!) 空间复杂度：O(n)

回溯法是按照深度优先搜索策略，根据隐约束(约束函数和限界函数)，也称剪枝函数，在解空间中搜索问题最优解。当发现当前结点不满足求解条件时，就回溯，尝试其他的路径。回溯法是一种“能进则进，进不了则换，换不了则退”的搜索方法。

回溯法解题的关键是设计有效的显约束(对解分量的取值范围的限定)和隐约束。

②分支限界法

结果：



代码：

#include <iostream>

#include <algorithm>//swap(a,b)

#include <queue>

using namespace std;

const int NoEdge=1e4;//无边标记-无穷大

const int N=10;

class Node{ //定义结点,记录当前结点的解信息

public:

int l, //当前已走过的路径长度

id, //景点序号

\*x; //记录当前路径

Node() {}

Node(int l1,int id1){ l=l1; id=id1; }

};

class Traveling{

public:

void BBTSP(int \*\*,int);

void print();

private:

void Bfs();

int n, //图G的顶点数

\*\*g, //图G的邻接矩阵

\*bestx, //当前最优解

bestl; //当前最优值

};

//定义队列的优先级.以l为优先级,l值越小,越优先

bool operator<(const Node &a, const Node &b){

return a.l>b.l;

}

void Traveling::BBTSP(int \*\*g1,int n1){//初始化

n=n1;

g=g1;

bestl=NoEdge;

bestx=new int [n+1];

for(int i=0;i<=n1;i++)

bestx[i]=0;

Bfs();

}

void Traveling::print(){//打印路径

cout<<endl;

cout<<"最短路径: ";

for(int i=1;i<=n;i++)

cout<<bestx[i]<<"->";

cout<<"1"<<endl;//回到起点结束

cout<<"最短路径长度："<<bestl;

}

//Bfs为优先队列式分支限界法搜索 核心函数 算法复杂度为：O(n!)

void Traveling::Bfs(){

int t; //当前处理的景点序号t

Node livenode,newnode;//当前扩展结点 新结点

priority\_queue<Node> q; //创建一个优先队列,优先级为已经走过的路径长度l,l值越小，越优先

newnode=Node(0,2);//创建根节点

newnode.x=new int [n+1];//绝对不能少！！！

for(int i=1;i<=n;i++)

newnode.x[i]=i;//初时化根结点的解向量

q.push(newnode);//根结点加入优先队列

cout<<"按优先级出队顺序："<<endl;//用于调试

while(!q.empty()){

livenode=q.top();//取出队头元素作为当前扩展结点livenode

q.pop(); //队头元素出队

//用于调试

cout<<"当前结点的id值:"<<livenode.id<<" 当前结点的l值:"<<livenode.l<<endl;

cout<<"当前结点的解向量:";

for(int i=1; i<=n; i++)

cout<<livenode.x[i];

cout<<endl;

t=livenode.id;//当前处理的景点序号

//搜到倒数第2个结点时不需要往下搜索

if(t==n){ //立即判断是否更新最优解

if(g[livenode.x[n-1]][livenode.x[n]]!=NoEdge && g[livenode.x[n]][1]!=NoEdge &&

(livenode.l+g[livenode.x[n-1]][livenode.x[n]]+g[livenode.x[n]][1]<bestl)){

bestl=livenode.l+g[livenode.x[n-1]][livenode.x[n]]+g[livenode.x[n]][1];

cout<<endl;

cout<<"当前最优的解向量:";

for(int i=1;i<=n;i++){

bestx[i]=livenode.x[i];

cout<<bestx[i];

}cout<<endl;cout<<endl;

}

continue;

}

//判断当前结点是否满足限界条件,不满足则不再扩展

if(livenode.l>=bestl) continue;

//扩展-没有到达叶子结点

for(int j=t; j<=n; j++){//搜索扩展结点的所有分支

if(g[livenode.x[t-1]][livenode.x[j]]!=NoEdge){//如果x[t-1]景点与x[j]景点有边相连

int l=livenode.l+g[livenode.x[t-1]][livenode.x[j]];

if(l<bestl){//有可能得到更短的路线

newnode=Node(l,t+1);

newnode.x=new int [n+1];//绝对不能少！！！

for(int i=1;i<=n;i++)

newnode.x[i]=livenode.x[i];//复制父结点的解向量

swap(newnode.x[t], newnode.x[j]);//交换x[t]、x[j]的值

q.push(newnode);//新结点入队

}

}

}

}

}

int main(){

int u,v,w,n,m;//u,v：城市 w：u和v城市间的路长

cout<<"输入景点数 n(结点数):";

cin>>n;

cout<<"输入景点之间的连线数(边数):";

cin>>m;

cout<<"依次输入两个景点u和v之间的距离w(格式:景点u 景点v 距离w):"<<endl;

int \*\*g;

g=new int \*[N];//初始一个N行N列的矩阵

for(int i=0;i<N;i++){

g[i]=new int[N];

}//注意二维指针的初始化！！！

for(int i=1;i<=n;i++)//初始化G图

for(int j=i;j<=n;j++)

g[i][j]=g[j][i]=NoEdge;//表示路径不可达

for(int i=1;i<=m;i++){//更新G图

cin>>u>>v>>w;

g[u][v]=g[v][u]=w;

}

Traveling Y;

Y.BBTSP(g,n);

Y.print();

return 0;

}

体会：

时间复杂度为：O(n!) 空间复杂度：O(n\*n!)

优先队列式分支限界法，以广度优先方式搜索问题的解空间树，按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

回溯法与分支限界法的区别.

求解目标不同：

1) 回溯法是找出解空间树中满足约束条件的所有解；

2) 分支限界法则是找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解。

搜索方式不同：

1) 回溯法以深度优先的方式搜索解空间树；

2) 分支限界法则以广度优先或以最小耗费优先的方式搜索解空间树。

2. 以下问题为著名的TSP CHN144 问题（数据在QQ群共享中），

请查询相关参考文献资料，对该问题的求解方法，优缺点和结果进行综述。

选择中国144个城市解决TSP问题，命名为CHN144。TSP问题可以简单描述为给出一条遍历给定的若干城市中所有城市的最短路径。

求解方法：

一般可分为精确算法和启发式算法。

优缺点：

精确算法在理论上能保证求得最优解，但是计算时间呈指数复杂度，难以解决大规模问题。

启发式算法可分为独立于问题的算法和问题相关算法。独立于问题的算法如遗传算法等，虽然理论上能够求得最优解，但收敛缓慢，对于大规模问题计算时间较长。问题相关算法在求解时效率非常高，但此类算法国语以来问题的特殊性，算法常常陷于局部优解。

结果：

在城市规模较少时，可以达到目前的最优结果，算法收敛速度较快，平均结果和收敛率还可以接受。但城市规模达到一定程度时，已经无法求得满意解了。这些算法应该结合其他算法在这方面的长处加以改进。

3. 以百度、高德、腾讯等为首的互联网企业，在地图和导航方面进行了很多的研究，但是其路径规划算法几乎都是基于时间和路程或费用的路径规划算法，其对于社会和环境所带来的影响并不在考虑之列。然而，像丹麦等国家的研究机构，已经开始支持基于环保和节能的路径规划算法（*Eco-Routing*）：例如，<http://people.cs.aau.dk/~byang/reduction.htm> ，或者基于安全的路径选择问题： Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. *Risk-Aware Path Selection with Time-Varying, Uncertain Travel Costs—A Time Series Approach*.*The VLDB Journal 27(2)*: 179-200 (2018). 请你调研一下这方面的相关情况，撰写一个小报告（一般建议 1500字以内）。

可参考：<https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4358928>

基于环保和节能的路径规划算法研究：

车辆路径问题是组合优化和运筹学领域研究的热点间題之一，主要研究满足约束条件的最优车辆使用方案以及最优的车辆路径方案，它基于一般车辆路径问题的框架，研究满足物流配送经营和运作需要的各种车辆路径问題，并构建具有高质量和高鲁棒性问题的求解算法，对于提高经营管理水平和降低运作成本具有重要的理论意义和现实价值。在已有的文献中，研究车辆路径问题的目标大多是车辆行驶里程最短、运输总费用最低、车辆按一定时间到达、使用的车辆数最少等。随着能源的日趋短缺和环境压力的不断增大，全社会的节能、环保意识逐渐加强，节能减排成为了物流配送车辆路线优化的新突破。

参考论文资料，相关算法如下。

从节能减排啲角度重新审视车辆路径问題，以节能减排为目标，寻找给所有顾客提供服务燃油消耗最少的路线。通过分析车辆运输中可以节能減排的环节，确定装载量和运输距离为基于节能減排的车辆路线模型主要的直接影响因素，定义基于节能減排的车辆路线问题为己知汽车空载时行驶每公里的燃油消耗量和增加单位负载行驶每公里的燃油捎耗量，寻找给所有顾客提供服务路径最短并且燃油消耗最少的路线。建立基于节能减排的几个车辆路径模型，包括有无能力约束的、闭合式和开放式的以及多任务的VRPRFC模型，开发设计其优化算法，并用MATLAB编程实现求解。以具休实例作为实证分析的材料，结果显示算法的可行性，得到了满足各约束条件的路径。通过比较VRPRFC模型与VRPMTD模型的不同，分析节能减排对车辆路径安排的影响及影响程度， 为运输、物流和配送管理等领域中最优车辆路径方案的规划与设计提供了借鉴和参考。VRPRFC是一个复杂的路径优化问题，影响因素很多，不同的角度可以有不同的路径策略，因此模型尚待完善，对于大规模求解算法也有待进一步深入研究。

减少使用生态路线的车辆路径问题：

减少运输产生的温室气体（GHG）排放对于应对全球变暖和气候变化至关重要。例如，欧盟承诺到2020年将温室气体排放量减少到1990年水平的20％。在欧盟，交通运输的排放量占温室气体总排放量的近四分之一。生态路由使驾驶员能够使用最环保的路线，是一种减少车辆排放的简单而有效的方法。

参考论文资料，相关算法是究如何使用大型运输数据（包括GPS（全球定位系统）数据，CAN（控制器局域网）总线数据和3D激光扫描数据）来实现生态路由。特别是，我们通过以下三个主要步骤来实现生态路由：量化车辆环境影响，为道路网络分配生态权重以及开发新颖的生态路由算法。