



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

程序设计挑战式课程设计

极限挑战

挑战，不是为着征服自然，而是为着突破自我，超越自我
生命有极限，思想无极限，高度有极限，境界无极限

作业名称:	基于模拟退火算法的“旅行商问题”解决策略
学 院:	微电子学院
班 级:	DL062445
学 号:	2024303798
姓 名:	颜语恩
团队组成:	颜语恩

西北工业大学

2024 年 12 月 13 日

请填写以下十项内容，将表格按页对齐（插入空行），勿删除任何部分。

1、问题与背景（描述程序所要解决的问题或应用背景）

旅行商问题（Travelling Salesman Problem, TSP）是组合优化领域的经典问题之一。问题的核心是：给定若干城市及这些城市之间的距离（或成本），一名旅行商需要从某个城市出发，访问每个城市一次且仅一次，然后返回出发城市。目标是找到一条总距离（或总成本）最小的旅行路线。

然而，由于 TSP 是一个经典的 NP-hard 问题，这意味着目前没有任何已知的多项式时间算法可以在所有情况下求解其最优解。换言之，如果你想求出一个 NP-hard 问题最精确的解，有且仅有将所有可能结果全部暴力枚举出来再进行逐个比较这一种方法。

但是，当可能的选择逐渐增加，暴力枚举所需的时空成本会呈爆炸式指数上升，在实际问题中逐渐失去其应用价值。例如，当城市数量为 5 时，需要计算的路径数量仅为 $5! = 120$ 条。然而，随着城市数量的增加，达到 10 个时，路径数量将激增至 $10! = 3,628,800$ 条；如果进一步增加到 20 个，路径数量将达到 $20!$ 约等于 2.43×10^{18} 条，这意味着即使使用每秒能够处理 10^{12} 次计算的超级计算机，也需要数百万年才能完成所有路径的枚举。因此，在面对大规模问题时，暴力枚举往往不再是一个可行的解决方案，而需要借助更高效的算法来应对。

为了解决这个问题，在数学家们的实践中，他们找到了一些能够在有限时间范围内最大程度地逼近客观精确解的算法，为这类复杂的优化问题提供了高效的解决方案。而本作所采用的**模拟退火（SA）**就是其中最为经典的算法之一。

模拟退火算法来源于固体退火原理，是一种基于概率的算法。将固体加温

至充分高的温度，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，分子和原子越不稳定。而徐徐冷却时粒子渐趋有序，能量减少，原子越稳定。在冷却（降温）过程中，固体在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

模拟退火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中随机寻找目标函数的全局最优解，即在局部最优解能概率性地跳出并最终趋于全局最优。模拟退火算法是通过赋予搜索过程一种时变且最终趋于零的概率突跳性，从而可有效避免陷入局部极小并最终趋于全局最优的串行结构的优化算法。

TSP 问题的解决意义重大。除了传统的物流运输，它也出现在芯片设计、机器人路径规划、无人机航线规划等现代科技领域。而模拟退火等优化算法的意义就在于能够以较低的计算成本获得高质量的近似解，这使其成为解决 TSP 等实际问题的有效工具，成为理论计算与实际应用的桥梁，深刻影响着多个行业的效率提升和创新进程。

2、开发工具（列出所使用的开发工具和第 3 方开发库）

Visual Studio 2022

3、主要功能（详细说明程序的功能）

用户从键盘输入若干城市及这些城市的经纬度坐标，从指定起点城市出发，访问每个城市一次且仅一次，然后返回起点城市。通过程序的算法，规划出一条在相当程度上逼近客观最小总路程的旅行路线。

4、设计内容（详细描述解决问题的原理和方法、算法、数据结构等）

(1) 在 `main` 函数中实现模拟退火以逼近最佳解的过程。

模拟退火算法包含两个部分即 Metropolis 算法和退火过程, 分别对应内循环和外循环。

外循环就是退火过程, 将固体达到较高的温度 (初始温度 T_0), 然后按照降温系数 α 使温度按照一定的比例下降, 当达到终止温度 T_{\min} 时, 冷却结束, 即退火过程结束。

Metropolis 算法是内循环, 即在每次温度下, 迭代 L 次, 寻找在该温度下能量的最小值 (即最优解)。在该温度下, 整个迭代过程中温度不发生变化, 能量发生变化。下图中所示即为在同一温度下迭代时固体能量发生的变化。

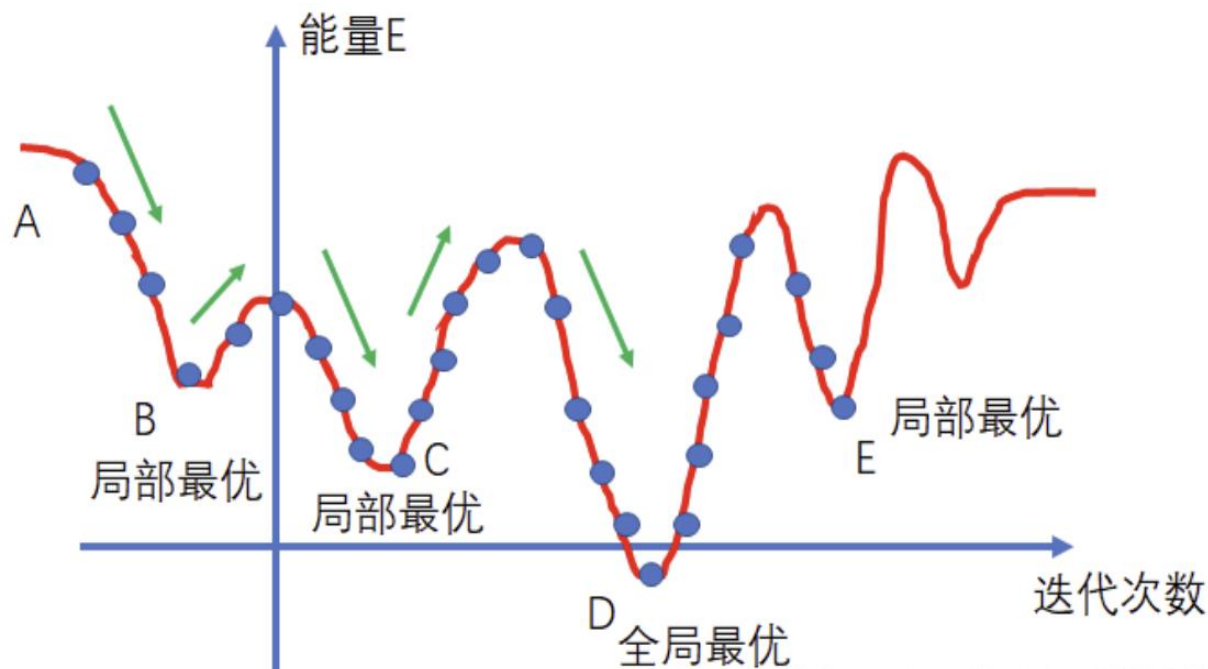


图 1 相同温度条件下的能量迭代过程示意

Metropolis 算法针对的问题是如何在局部最优解的情况下让其跳出来（如图中 B、C、E 为局部最优），是退火的基础。假设前一个状态为 X_n ，当系统依据某一指标使状态变为 X_{n+1} 时，相应的，当前的能量就由 E_n 变为 E_{n+1} 。则定义一个接受概率 P ，使系统能量由 E_n 变为 E_{n+1} 的概率为 P 。Metropolis 准则规定的 P 的计算公式如下：

$$P = \begin{cases} 1, E(n+1) < E(n) \\ e^{-\frac{E(n+1)-E(n)}{T}}, E(n+1) \geq E(n) \end{cases}$$

从上式我们可以看到，如果能量减小了，那么这种转移就被接受（概率为 1），如果能量增大了，就说明系统偏离全局最优值位置更远了，此时算法不会立刻将其抛弃，而是进行概率操作：首先在区间 $[0,1]$ 产生一个均匀分布的随机数 ϵ ，如果 $\epsilon < P$ ，则此种转移接受，否则拒绝转移。继而往复循环。其中能量的变化量 dE 和当前温度 T 进行决定接受概率 P 的大小，所以这个值是动态的。但它满足三条基本准则：[1]在固定温度下，接受使能量下降的候选解的概率要大于使能量上升的候选解概率；[2]随着温度的下降，接受使能量上升的解的概率要逐渐减小；[3]当温度趋于零时，只能接受能量下降的解。当我们赋予模拟退火算法一个 TSP 的实际情景，则每一个状态下的能量实际指的就是当前路线的总长。

在退火过程中有几个需要人为控制的参数：[1]初始温度 T_0 。应选的足够高，使所有转移状态都被接受。初始温度越高，获得高质量的解的概率越大，耗费的时间越长。本作选取 1000。[2]退火速率 α 。最简单的下降方式是指数式下降： $T(n) = \alpha T(n), n=1,2,3,\dots$ 其中 α 是小于 1 的正数，一般取值为 0.8 到 0.99 之间，使得对每一温度都有足够的转移尝试，指数式下降的收敛速度比较慢。本作选取 0.999。[3]终止温度 T_{\min} 。温度下降到终止温度时退火完成。本作选取 1×10^{-8} 。

综上所述，我们可以为模拟退火算法的主函数流程画一个示意图。如下：

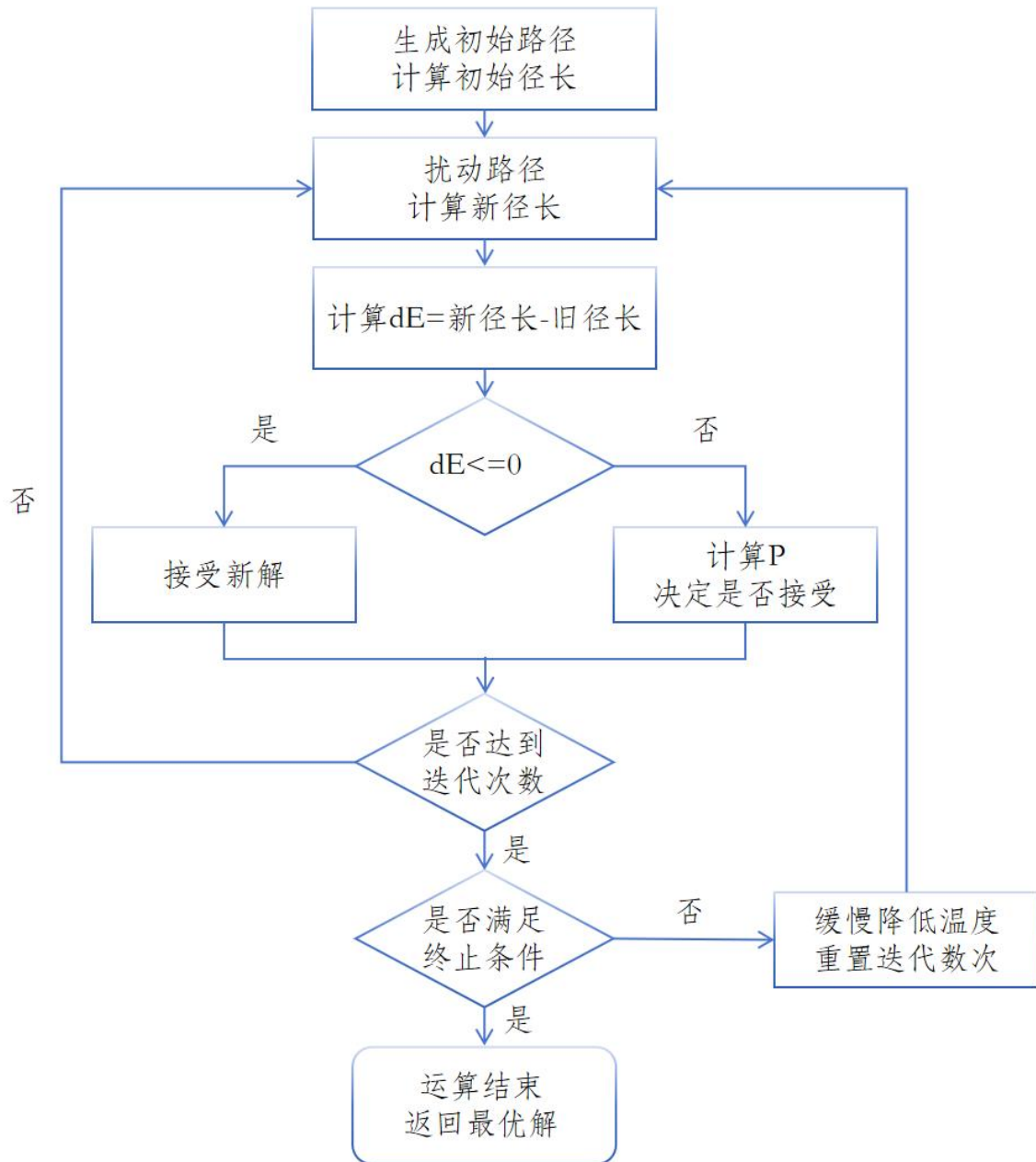


图 2 TSP 中模拟退火算法的实现过程示意

(2) 获得初始解的算法：内嵌 Fisher-Yates 洗牌算法的蒙特卡洛算法

蒙特卡洛算法（Monte Carlo）也称统计模拟法、统计实验法，是把概率现象作为研究对象的数值模拟方法，是按抽样调查法求取统计值推定未知特性的计算方法。该方法通过构造一个和系统相似的概率模型，在数字计算机上进行随机试验来模拟系统的随机特性，故适用于对离散系统进行仿真实验，特别适用于一些解析法难以求解甚至无法求解的问题。

蒙特卡洛的理论基础是概率统计理论，其实现方法是随机抽样、统计实验，用抽样后的样本发生的频率来估计概率，所以它求得的是近似解，而不是精确解，随着样本数的增多，近似解越接近精确解。蒙特卡洛方法本身不是优化算法，与模拟退火算法等优化算法有着本质区别。

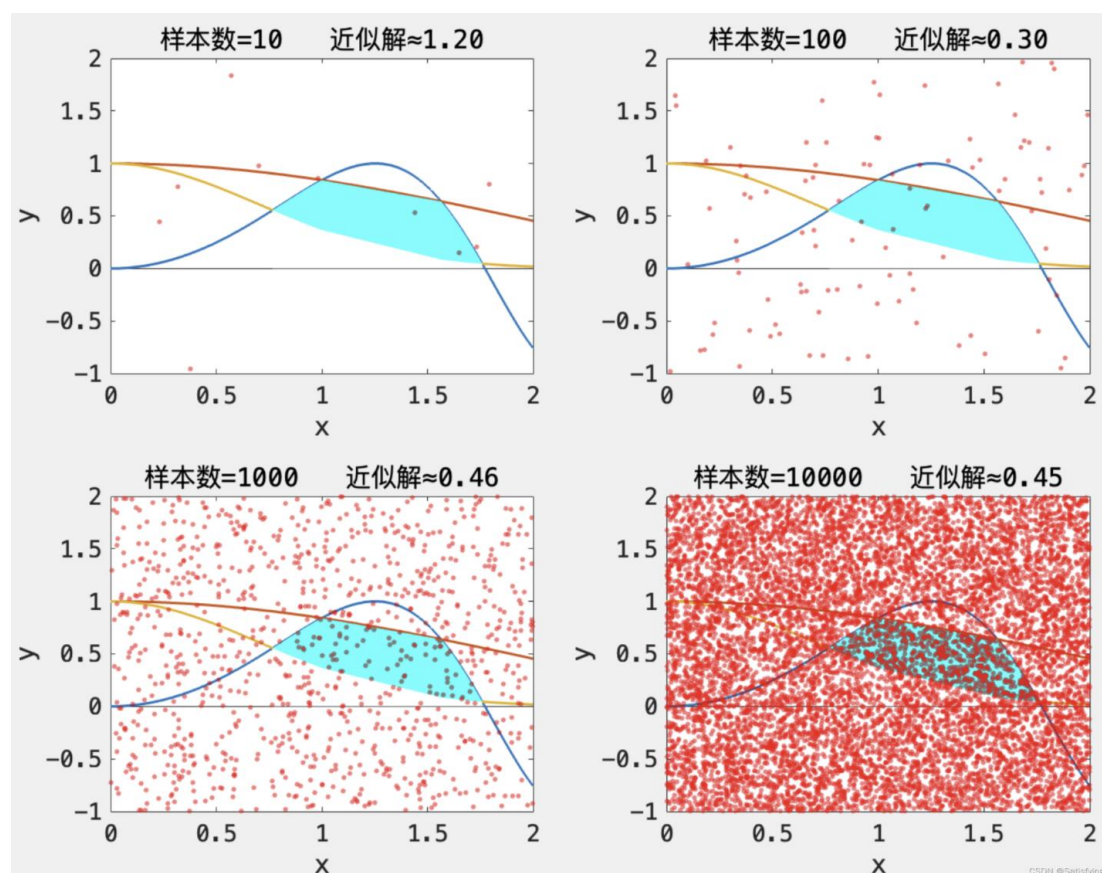


图 3 蒙特卡洛算法示意

本作使用蒙特卡洛算法来生成一个较为接近精确解的初始解，可以有效降低模拟退火过程的时间空间成本，让其迭代过程更多地集中在针对某一大方向的细微优化上。

本作中蒙特卡洛算法的实现过程中还采用了 Fisher-Yates 洗牌算法用于将城市途经顺序随机打乱。Fisher-Yates 洗牌算法是一种高效且公平的随机排列算法，它的时间复杂度是 $O(n)$ ，并且能够确保每种排列出现的概率相等。其基本步骤为：从数组的最后一个元素开始，逐步向前遍历，对于索引为 i 的元素，从 0 到 i 范围内随机选取一个索引 j 并交换索引 i 和索引 j 处的元素。一直重复上述步骤直到第一个元素。

(3) 扰动路径产生新解的方法：交换法和逆序法的加权结合

在每个温度下，我们需要对上一个温度传递下来的路径进行多次扰动以求出当前温度的全局最优解，这意味着我们需要有两种不同的需求。

第一，是对当前路径的细微扰动，目的是在当前大方向基础上进行深入优化以求当前大方向的最优解。该需求对应的实现方式就是交换法，即在路径上随机选取两点将其顺序交换。这种方法对路径的扰动较小，可以保持选定大方向不变。

第二，是对路径的较大幅度扰动，目的是跳出局部最优以寻求全局最优。该需求对应的实现方式就是逆序法，即在路径上选定两点，将该两点及两点间的所有点的顺序全部逆置。这种方法对路径的扰动较大，可以跳出当前大方向而跳入另一个大方向。

由于从全局角度观察，我们对精细优化的需求必定大于我们对大方向探索的需求，因此本作对于这两种方法的使用进行了 8:2 加权处理。经过多个数据集的测试，该加权比重较为理想可靠。

(4) 每个温度的迭代过程中对新解集的动态存储：链表的使用

由于模拟退火算法的优化特性，迭代次数 L 越大，模拟出的解越接近真实的精确解。因此， L 的值一般而言都较大，新解集数据量难以确定。此时，如果使用数组存储新解集，一方面，数组的大小是固定的，一旦分配了内存，大小无法动态调整；另一方面，数组需要一次性分配足够大的连续内存空间，可能受到内存限制，而如果数组定义过大而实际使用较少，也会造成内存浪费。

而与之相应的，链表的节点是动态分配的，可以根据需要随时插入或删除节点，而不需要预先分配固定大小的内存；链表的节点可以将大规模数据分段分散在内存的任何位置，通过指针连接。因此，本作中创建了链表来动态存储每个迭代过程中的新解集，一定程度上解决了上述使用数组的问题。

(5) 计算地球上两点之间的直线距离：Haversine 公式

由于现实中路况复杂多变，因此本作中两座城市之间的距离近似为地球上两点间的直线距离。其计算依据 Haversine 公式，灵感来源于 NOJ 第 8 题“计算地球上两点之间的距离”。

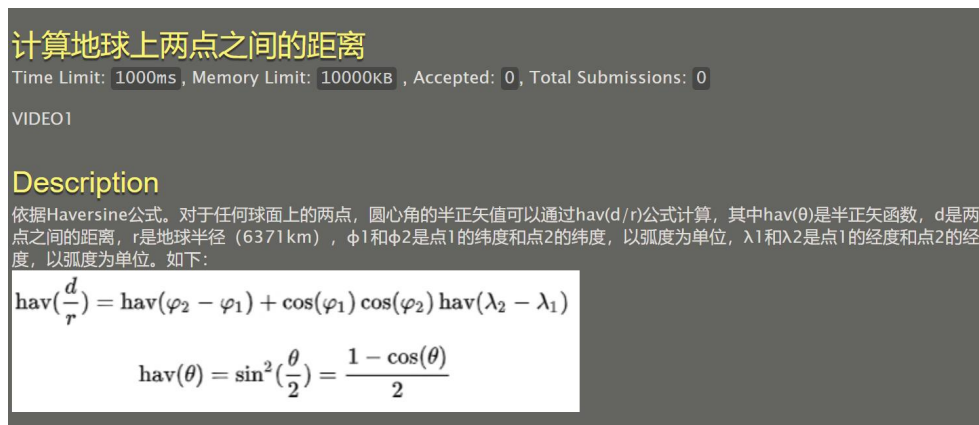


图 4 NOJ “计算地球上两点之间的距离”

(6) 简易加载动画

由于优化规划过程所需时间一般较长，故本作还在程序运行结果出来之前使用<thread>和<chrono>库设计了一个独立线程，显示加载动画。

5、程序文件与工程名称（标出程序中所有文件名、工程名称及其说明）

工程名称：Major_Project_1

作业程序文件：main.cpp

6、函数模块（程序中各个函数的原型声明及其说明）

- (1) struct CITY 城市结构体，包含城市索引、城市名称和城市经纬度。
- (2) void loading_animation() 函数，显示加载动画。
- (3) struct Node 链表的结点，其数据段包含一条路线以及该路线的总长度。

- (4) void insert_node(Node*& head, double distance, int* city_order, int size) 函数，插入一个新结点到链表的尾部，如果链表为空，则创建一个头结点。
- (5) double toRadians(double degree) 函数，将一个角度值转换为弧度值。
- (6) double hav(double x) 函数， double hav_distance(CITY city1, CITY city2) 函数，根据 Haversine 公式计算两座城市之间的距离。
- (7) double path_length(int* city_order, int num, CITY* city_passed) 函数，计算一整条路线的长度。
- (8) int* monte_carlo(int samples, int n, CITY* city_passed) 函数，根据蒙特卡洛算法生成初始解，返回记录了初始解路线的数组。
- (9) void dis_change(int* city_order, int n) 函数，交换法扰动路径。
- (10) void dis_reverse(int* city_order, int n) 函数，逆序法扰动路径。
- (11) void disturb(int* city_order, int n) 函数，加权两种方法，对路径进行扰动。
- (12) bool metropolis(double dE, double T) 函数，依据 Metropolis 准则，对是否接受新解进行决策。
- (13) int main() 主函数，对模拟退火算法的主体核心部分进行实现，并完成最终结果的输出。

7、使用说明（运行程序的小型说明书）

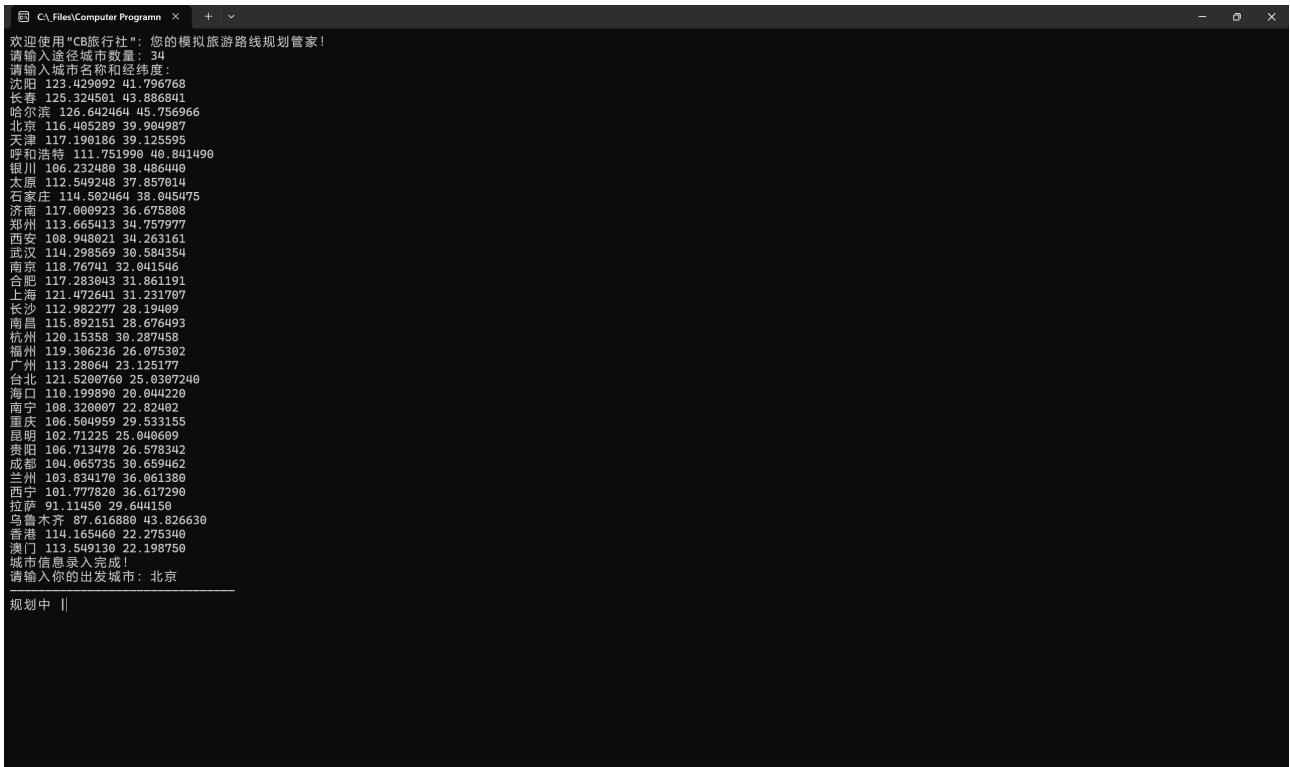
- (1) 运行 Major_Project_1.exe，等待 3 秒。
- (2) 首先输入城市数目，然后在接下来的每一行输入一条城市信息，包括城市的名称、城市的经度、城市的纬度，每两个数据之间用空格间隔。最后，输入你的出发城市，即最后输出路线的起点和终点。
- (3) 耐心等待程序规划完成。

8、程序开发总结（简要叙述编写本作业的收获与思考）

本程序的设计灵感来自于数学建模中的模拟退火算法，我使用 C++ 代码复现了这一经典算法、经典问题。在这次程序的设计过程中，我从模拟退火的主干出发，在具体逐步实现的过程中碰到了许许多多各种各样的问题，由此延伸出了很多主干以外的分支。比如，如何选取初始解？如何对路径进行扰动？ T_0 、 α 、 T_{\min} 、 L 的值应该选取多少合适？如何求解两座城市之间的距离？面对这些问题，我也尝试了不同渠道、不同来源的解决方法，并进行了不厌其烦的优化与完善，最终给自己交了一份满意的答卷。

通过这一次程序设计，不仅提高了我的代码编写能力，加深了我对不同算法、不同数据结构的认识，而且将我的编程思维与数学建模思想进行了深层次的结合，对我未来的发展起了一定的促进作用。诚然，只有将理论联系实际、勇于投身实践，才能逐步提升自己的知识水平，不断打磨自己的问题解决能力和创新创造能力，实现学科融合，实现思维深化。

9、运行截图（附上程序运行的截图画面，至少有 1 幅，截图越翔实得分越高）



```

C:\Files\Computer Program > .\
欢迎使用"CB旅行社": 您的模拟旅游路线规划管家!
请输入途径城市数量: 34
请输入城市名称和经纬度:
沈阳 123.429092 41.796768
长春 125.324501 43.886841
哈尔滨 126.642464 45.756966
北京 116.405289 39.904987
天津 117.190186 39.125595
呼和浩特 111.751990 40.841490
银川 106.232480 38.486440
太原 112.549248 37.857814
石家庄 114.502464 38.045475
济南 117.000923 36.675808
郑州 113.665413 34.757977
西安 108.948021 34.263161
武汉 114.298569 30.584354
南京 118.76741 32.041546
合肥 117.283043 31.861191
上海 121.472641 31.231707
长沙 112.982277 28.19409
南昌 115.892151 28.676493
杭州 120.15358 30.287458
福州 119.306236 26.075302
广州 113.28064 23.125177
台北 121.520076 25.030724
海口 110.199890 20.044220
南宁 108.320007 22.82402
重庆 106.504959 29.533155
昆明 102.71225 25.040609
贵阳 106.713478 26.578342
成都 104.065735 30.659462
兰州 103.834170 36.061380
西宁 101.777820 36.617290
拉萨 91.11450 29.644150
乌鲁木齐 87.616880 43.826630
香港 114.165460 22.275340
澳门 113.549130 22.198750
城市信息录入完成!
请输入你的出发城市: 北京

规划中 ||
  
```



```

C:\Files\Computer Program > .\
欢迎使用"CB旅行社": 您的模拟旅游路线规划管家!
请输入途径城市数量: 34
请输入城市名称和经纬度:
沈阳 123.429092 41.796768
长春 125.324501 43.886841
哈尔滨 126.642464 45.756966
北京 116.405289 39.904987
天津 117.190186 39.125595
呼和浩特 111.751990 40.841490
银川 106.232480 38.486440
太原 112.549248 37.857814
石家庄 114.502464 38.045475
济南 117.000923 36.675808
郑州 113.665413 34.757977
西安 108.948021 34.263161
武汉 114.298569 30.584354
南京 118.76741 32.041546
合肥 117.283043 31.861191
上海 121.472641 31.231707
长沙 112.982277 28.19409
南昌 115.892151 28.676493
杭州 120.15358 30.287458
福州 119.306236 26.075302
广州 113.28064 23.125177
台北 121.520076 25.030724
海口 110.199890 20.044220
南宁 108.320007 22.82402
重庆 106.504959 29.533155
昆明 102.71225 25.040609
贵阳 106.713478 26.578342
成都 104.065735 30.659462
兰州 103.834170 36.061380
西宁 101.777820 36.617290
拉萨 91.11450 29.644150
乌鲁木齐 87.616880 43.826630
香港 114.165460 22.275340
澳门 113.549130 22.198750
城市信息录入完成!
请输入你的出发城市: 北京

规划完成!

最短旅游路线:
北京 -> 天津 -> 济南 -> 沈阳 -> 长春 -> 哈尔滨 -> 上海 -> 台北 -> 福州 -> 杭州 -> 南京 -> 南昌 -> 合肥 -> 南昌 -> 武汉 -> 长沙 -> 香港 -> 澳门 -> 广州 -> 海口 -> 南宁 -> 贵阳 ->
重庆 -> 成都 -> 昆明 -> 乌鲁木齐 -> 拉萨 -> 西宁 -> 兰州 -> 银川 -> 西安 -> 呼和浩特 -> 太原 -> 郑州 -> 石家庄 -> 北京
最短路径长度: 11340.69km
请按任意键继续... |
  
```

10、源程序（附上程序源代码，若是多个文件，标出文件名）

//大作业 1——使用模拟退火算法解决 TSP 问题

//关键点：模拟退火算法、蒙特卡洛算法、Fisher-Yates 洗牌算法、链表

//制作者：2024303798 颜语恩 DL062445

/* ———— 头文件包含以及初始准备 ———— */

#include <iostream>

#include <cmath>

#include <random>

#include <ctime>

#include <iomanip>

#include <thread>

#include <chrono>

using namespace std;

#define M_PI 3.14159265358979323846

#define EARTH_R 6371.4

//定义城市结构体

struct CITY {

int id;

char name[20];

double x;

double y;

};

//加载动画

bool loading = false;

void loading_animation() {

const char spinner[] = { '|', '/', '-', '\\' }; // 动画帧

int index = 0; // 动画帧索引

while (loading) { // 在任务未完成前持续输出动画

cout << "\r 规划中 " << spinner[index] << flush; // 使用回车符覆盖行

index = (index + 1) % 4; // 循环动画帧

this_thread::sleep_for(chrono::milliseconds(100)); // 控制动画速度

}

cout << "\r 规划完成!" << endl; // 输出完成状态

}

```
/* ———— 链表的使用 ———— */
```

```
// 定义链表节点结构
```

```
struct Node {
    double distance;           // 路径长度
    int* city_order;           // 路径城市顺序
    Node* next;                // 指向下一个节点
    Node(double dist, int* order, int size) : distance(dist), next(nullptr) {
        city_order = new int[size];
        for (int i = 0; i < size; i++) {
            city_order[i] = order[i];
        }
    }
    ~Node() {
        delete[] city_order; // 释放路径内存
    }
};
```

```
// 插入节点到链表尾部
```

```
void insert_node(Node*& head, double distance, int* city_order, int size) {
    Node* new_node = new Node(distance, city_order, size);
    if (!head) { // 链表为空
        head = new_node; // 新节点为头节点
    }
    else {
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = new_node;
    }
}
```

```
/* ———— 距离的计算 ———— */
```

```
// 依据 Haversine 公式计算地球上两点之间的距离
```

```
double hav(double x)
{
    double result;
    result = (1 - cos(x)) / 2;
    return result;
}

double toRadians(double degree)
```



```

{
    return degree * (M_PI / 180.0);
}
double hav_distance(CITY city1, CITY city2)
{
    double lat1 = toRadians(city1.x);
    double lat2 = toRadians(city2.x);
    double lon1 = toRadians(city1.y);
    double lon2 = toRadians(city2.y);
    double dlat = lat2 - lat1;
    double dlon = lon2 - lon1;
    double a = hav(dlat) + cos(lat1) * cos(lat2) * hav(dlon);
    double c = 2 * asin(sqrt(a));
    return EARTH_R * c;
}

//计算一整条路径的长度
double path_length(int* city_order, int num, CITY* city_passed) { //num 包括尾部城市, city_order 的最大下标为 num-1
    double distance = 0;
    for (int i = 0; i < num-1 ; i++) {
        distance += hav_distance(city_passed[city_order[i]],
city_passed[city_order[i + 1]]);
    }
    return distance;
}

/* ———— 蒙特卡洛算法 ———— */

int* monte_carlo(int samples, int n, CITY* city_passed) { //n 不包括尾部城市
    double min_distance = 1e9;
    int* path = new int[n+1];
    //循环 100 次, 每次生成一个随机解, 记录最短路径长度
    for (int i = 0; i < samples; i++) {
        int* city_order = new int[n+1];
        for (int j = 0; j < n; j++) {
            city_order[j] = j;
        }
        //使用 Fisher-Yates 洗牌算法打乱城市顺序
        for (int j = n-1 ; j > 0; j--) {
            int k = rand() % (j + 1);
            int temp = city_order[j];
            city_order[j] = city_order[k];
        }
    }
}

```

```

        city_order[k] = temp;
    }
    city_order[n] = city_order[0];
    //计算路径长
    double distance = path_length(city_order, n+1, city_passed);
    if (distance < min_distance) {
        min_distance = distance;
        for (int j = 0; j <= n; j++) {
            path[j] = city_order[j];
        }
    }
    delete[] city_order;
}
return path;
}

```

/* ———— 模拟退火算法相关函数 ———— */

//扰动路径产生新解——交换法（影响较小）

```

void dis_change(int* city_order, int n) {
    int i = rand() % n;
    int j = rand() % n;
    int temp = city_order[i];
    city_order[i] = city_order[j];
    city_order[j] = temp;
    city_order[n] = city_order[0];
}

```

//扰动路径产生新解——逆转法（影响较大）

```

void dis_reverse(int* city_order, int n) {
    int i = rand() % n;
    int j = rand() % n;
    if (i > j) {
        int temp = i;
        i = j;
        j = temp;
    }
    while (i < j) {
        int temp = city_order[i];
        city_order[i] = city_order[j];
        city_order[j] = temp;
        i++;
        j--;
    }
}

```

```

    city_order[n] = city_order[0];
}
//随机选择扰动方法
void disturb(int* city_order, int n) {
    int r = rand() % 10;
    if (r < 8) {
        dis_change(city_order, n);
    }
    else {
        dis_reverse(city_order, n);
    }
}

//Metropolis 准则
bool metropolis(double dE, double T) {
    if (dE < 0) {
        return true;
    }
    else {
        double p = exp(-dE / T);
        double r = (double)rand() / RAND_MAX;
        if (r < p) {
            return true;
        }
        else {
            return false;
        }
    }
}

/* ———— 主函数：模拟退火算法 ———— */

int main() {
    srand(static_cast<unsigned int>(time(0)));
    int n;
    cout << "欢迎使用\"CB 旅行社\": 您的模拟旅游路线规划管家! " << endl;
    this_thread::sleep_for(chrono::seconds(2));
    cout << "请输入途径城市数量: ";
    cin >> n;

    CITY* city_passed = new CITY[n + 1];
    cout << "请输入城市名称和经纬度: " << endl;
    for (int i = 0; i < n; i++) {

```

```

        city_passed[i].id = i;
        cin >> city_passed[i].name >> city_passed[i].x >> city_passed[i].y;
    }
    city_passed[n] = city_passed[0];
    cout << "城市信息录入完成！" << endl;
    cout << "请输入你的出发城市：";
    char start_city[20];
    cin >> start_city;

    cout << "-----" <<
endl;
    thread loading_thread(loading_animation); //开启加载动画线程
    loading = true;

    //使用蒙特卡洛算法生成初始解
    const int samples = 100; //蒙特卡洛算法的样本数
    int* initial_path = monte_carlo(samples, n, city_passed);
    double distance = path_length(initial_path, n + 1, city_passed); //初始解路径
    长度

    double T = 1000; //初始温度
    double T_min = 1e-8; //终止温度
    double alpha = 0.999; //降温速率
    int L = 100; //每个温度下的迭代次数
    int* city_order = new int[n + 1];
    for (int i = 0; i <= n; i++) {
        city_order[i] = initial_path[i];
    }
    delete[] initial_path;
    initial_path = nullptr;

    //模拟退火过程
    while (T > T_min) {

        //创建链表存储当前温度下所有新解
        Node* head = nullptr;
        //迭代L次
        int k = 0;
        for (int i = 0; i < L; i++) {
            int* city_order_temp = new int[n + 1];
            for (int j = 0; j <= n; j++) {
                city_order_temp[j] = city_order[j];
            }
            disturb(city_order_temp, n);
            double new_distance = path_length(city_order_temp, n + 1, city_passed);

```

```

        double dE = new_distance - distance;
        if (metropolis(dE, T)) {
            insert_node(head, new_distance, city_order_temp, n + 1);
            k++; //最终 k 为当前温度下的新解数量
        }
        delete[] city_order_temp;
    }

    //遍历链表，记录当前最优路径及其路径长度
    if (k > 0) {
        double min_distance = 1e9;
        Node* temp = head; //从头节点开始
        while (temp) { //指针不为空
            if (temp->distance < min_distance) {
                min_distance = temp->distance;
                for (int i = 0; i <= n; i++) {
                    city_order[i] = temp->city_order[i];
                } //更新最优路径
            }
            temp = temp->next;
        }
        distance = min_distance; //更新最短路径长度
    }
    city_order[n] = city_order[0];
    //至此，当前温度下的最优路径已经找到

    //释放链表内存
    Node* temp = head;
    while (temp) {
        Node* next = temp->next;
        delete temp;
        temp = next;
    }
    head = nullptr;

    //降温
    T *= alpha;

}

//输出最佳路径以及最短路径长
loading = false;
loading_thread.join();
cout << "-----" <<
endl;

```

```

cout << "最短旅游路线: " << endl;
//找到出发城市的位置
int start = 0;
for (int i = 0; i <= n; i++) {
    if (strcmp(city_passed[city_order[i]].name, start_city) == 0) {
        start = i;
        break;
    }
}
for (int i = start; i <= n; i++) {
    cout << city_passed[city_order[i]].name << " -> ";
}
for (int i = 0; i < start; i++) {
    cout << city_passed[city_order[i]].name << " -> ";
}
cout << city_passed[city_order[start]].name << endl;
cout << "最短路径长度: " << fixed << setprecision(2) << distance << "km" << endl;

delete[] city_passed;
system("pause");
return 0;
}

```