**五 一 数 学 建 模 竞 赛**

****

**题 目： 徐州潘安湖风景区游览路线设计**

**关键词：游览路线 图论 Dijkstra算法 蚁群算法 DFS 多线程互斥 正态分布 TSP**

**摘 要：**

随着徐州潘安湖湿地公园从徐州最大的、塌陷最严重的采煤塌陷区转型为国家级水利风景区后，徐州旅游综合收入节节高升，本题选取潘安湖景区的部分景点提出关于徐州潘安湖风景区游览路线的问题。本文基于蚁群算法得出距离最短的路线，再以生成符合正态分布的随机数与多线程互斥的方法得出游览总时间最长的路线，算法的可行度与可操作性高，在现实生活中具有重要的实际指导意义。

针对问题一，需要解决游览所有景点至少1次的距离最短的路线问题，利用蚁群算法模拟真实蚂蚁寻路的过程，通过信息素的更新来使路径的选取更加符合题意，由于题目描述为每个景点经过至少一次，在代码中体现为蚂蚁小概率会选择已去的景点，接着很容易由编程得出最短路径，并按照表3的格式填写。

针对问题二，在问题一的基础上，规定游览起止时间，解决游览总时间最长的游览路线问题，假设所有时间仅仅分配为走路时间和游玩时间，其本质和第一题一样，仍旧是哈密顿最短路径问题，引入时间的正态分布随机生成机制以及约束条件，通过扩大蚁群规模和增加迭代数目，易求得结果，并按照表4的格式填写。

针对问题三，在问题二的基础上，增加为三个旅游团，一个多团体争夺资源的问题，我们将其简化为多线程互斥问题。通过定义互斥量数组，在蚂蚁对象选择下个节点时对互斥量进行访问，从而确认是否能选择该节点。

针对问题四，在问题三的基础上，限制了步行速度的要求，我们假设这些团体的行进速度分布曲线满足正态分布，设计相应算法生成随机速度，其数学期望为上下限的平均值，使其整体速度的平均值不超过上下限的平均值。

针对问题五，在问题四的基础上，增加了两个不确定因素，对于情况二，只需要随机增加出发时间的随机生成值,然后实现方法同问题四。而对于情况二，则其情况更为复杂，设计扰动函数，即在一定条件下打乱序列，具体情况比如

1、每次寻路上小概率加上意外时间(时长随机生成且超过极限值时打乱序列)。

2、对散客的来访假设其来访满足正态分布，则随机在高峰段打乱序列。

1. 问题重述

1.1问题背景

2017年12月12日至13日，习近平总书记在徐州调研贾汪区潘安采煤塌陷区整治工程时强调，塌陷区要坚持走符合国情的转型发展之路，打造绿水青山，并把绿水青山变成金山银山。位于徐州市贾汪区西南部、紧邻马庄的潘安湖湿地公园原来就是徐州最大的、塌陷最严重的采煤塌陷区。2010年，江苏省单体投资最大的一宗土地整理项目在潘安湖启动，2016年，贾汪被列为“国家全域旅游示范区”首批创建单位，全区旅游综合收入节节高升。本题选取潘安湖景区的部分景点，要求完成徐州潘安湖风景区游览路线设计问题。

1.2问题概述

相关的规定与假设为：

（1）任意两个景点之间的最短步行距离如表1给出。

（2）第二问、第三问假设步行速度。

（3）游客在景区停留的时间由“景点之间的步行时间”、“景点游览时间（即在景点内游玩的时间）” 和“在景区外的等待时间”三部分组成，其他时间忽略不计，游览时间必须符合表2的要求。

问题一：从景石出发，步行游览①-⑦景点，建立数学模型，找出从景石出发，到达⑦湿地商业街，并且经过①—⑥所有景点至少1次的距离最短的路线，计算该路线的长度，并将相关结果填入表格3。（注：在每个景点不用停留。）

问题二：如果某游客12:00从景石出发，要求他17:00前到达湿地商业街，17:30离开湿地商业街(注：根据表2的要求在湿地商业街游览时间至少为30分钟)。建立数学模型，为该游客设计一条能游览完全部景点(景点①—⑦)且游览总时间最长的游览路线（假设在各个景点没有等待时间），并完成表4的填写。

问题三：如果有3个旅游团，与问题二的条件相同，并且每个景点（湿地商业街除外）同时只能容纳1个旅游团游览，按照时间顺序后到达的旅游团，需要等待先到达的旅游团游览结束之后才能开始游览。建立数学模型，为三个旅游团分别设计一条能游览完全部7个景点且游览总时间最长的游览路线，并完成表5的填写。

问题四：假设3个旅游团的步行速度可以在1km/h到3km/h之间调节，但是总的平均步行速度不能超过2km/h，其他条件与问题三相同，建立数学模型，为三个旅游团分别设计一条能游览完全部7个景点且游览总时间长，总的等待时间短的游览路线，并完成表6的填写。

问题五：在现实中，考虑如下两个不确定性因素：

(1)不同旅游团从景石出发的时间具有不确定性，例如，多个旅游团在不同的时间从景石出发开始游览，在此情况下到达湿地商业街的时间可以顺延。

(2)每个景点的等待时间也存在不确定性因素，例如，旅游设施短时间的维护和清理，或者受到散客客流的影响。

考虑上述两个不确定性因素，其它条件与问题4相同，建立数学模型，为多个旅游团分别设计一条能游览完全部7个景点且游览总时间长，总的等待时间短的游览路线。

1. **问题分析**

本题研究的是旅游路线问题，问题一至五为四种递进情况下合理规划路线的问题，本文的解题思路也是沿着这样的思想一步一步展开的，即通过简单到复杂的情况逐步展开讨论。

2.1问题一的分析

针对问题一，我们发现除了个别特殊情况，这个问题即寻找最短路径的哈密顿通路。根据题意，通过蚁群算法模型模拟游客寻路的过程，我们通过在大量迭代过程中，“信息素”这一变量的更新来完成对函数的收敛，(蚁群算法的移动规则：蚂蚁朝信息素最多的方向移动，当周围没有信息素指引时，会按照原来运动方向惯性移动。而且会记住最近走过的点，防止原地转圈。)而这一更新过程使得信息素与(DBQ(我们设定的信息素总量)/总路程)正相关，使算法函数向总路程最短的方向收敛。且根据“每个节点至少到达一次”的表述，我们将选择去过的节点的概率设定为0.01，即存在但极小。

2.2问题二的分析

针对问题二，在问题一的基础上设计惩罚函数，引入随机生成时长满足正态分布的时间量，实现对路径的约束。根据题意我们发现只有两个节点存在时间约束，且其他节点都有足够的调整空间，即其题目的本质和第一题相似，仍是一个寻找最短路径的哈密顿通路的TSP问题。

2.3问题三的分析

针对问题三，我们发现这是一个多团体争夺资源的问题，我们将其简化为多线程互斥问题。通过定义互斥量数组，在蚂蚁寻路，选择下个节点时对互斥量进行访问，根据返回结果确认是否能选择该节点。当不能选择时，做二元判断，即选择等待或选择跳过。这个判断我们通过0-1概率生成选择。通过线程通讯获得其线程中对象的逃出时间并写入其进入时间。运行程序循环打印出路径列表即可。

2.4问题四的分析

针对问题四，根据题意则是在上一问的基础上，团体的速度出现波动，根据假设，这些团体的行进速度分布曲线满足正态分布，设计相应算法生成随机速度，取合理方差，且其数学期望为上下限的平均值，使其整体速度的平均值不超过上下限的平均值，即设计生成速度的随机函数。将之前两个问题中使用打表列出的移动时间用路程/随机速度取代，运行程序循环打印出路径列表即可。

2.5问题五的分析

针对问题五，由题意可知其随机性更大。针对情况一来说，只需要随机增加出发时间的随机生成值,(查阅资料可得，故宫作为中国游客流量最大的景点，其极限日流量为六万，按十人组团来算，其旅游团数不超过6000，数量级不大，其出发时间接近平均分布)然后实现方法同问题四。而针对情况二，则其情况更为复杂，设计扰动函数，具体情况比如①每次寻路上小概率加上意外时间(时长随机生成且超过极限值时进入扰动函数)。②对散客的来访假设其来访满足正态分布，则随机在高峰段进入扰动函数。最后运行程序循环打印出路径列表即可。

1. **模型假设**
2. 假设游客游览途中不因为任何事情的干扰而终止游览。
3. 假设前四题的景点都不存在游客过多禁止游览，设备维护等问题。
4. 假设前四题游客不因为特殊情况中途休息。
5. 假设前三题步行速度是匀速的。
6. **符号说明**

|  |  |
| --- | --- |
| 符号 | **意义** |
| C | 景点的数量 |
| **D** | 两两景点间的距离 |
| **L** | 最短路径 |
|  | 景点之间最短步行距离矩阵 |
| **S** | 游客出发的景点 |
| **E** | 游客终止的景点 |
|  | 景点之间最短步行距离所花的时间矩阵 |
|  | 游客进入景点的时间 |
|  | 游客在景点花费的时间 |
|  | 游客选择等待的几率 |
|  | 游客移动的速度 |
| **Textra** | 游客在游览过程中由于意外而耽误的时间 |
|  | 散客来访的概率 |
| N | 概率密度函数 |
|  | 位置参数，尺度参数 |
|  | 调用扰动函数的概率 |

1. **模型的建立与求解**

5.1问题一模型的建立与求解

5.1.1建模过程

我们首先由Dijkstra算法已经知道了景点之间最短步行距离矩阵，再通过蚁群算法模型模拟游客寻路的过程，并通过大量迭代与更新，搜索过程使用分布式计算方式，使算法函数向总路程最短的方向收敛，最后算法将得到经过①—⑥所有景点至少1次的距离最短的路线与路程。（表格见附录）

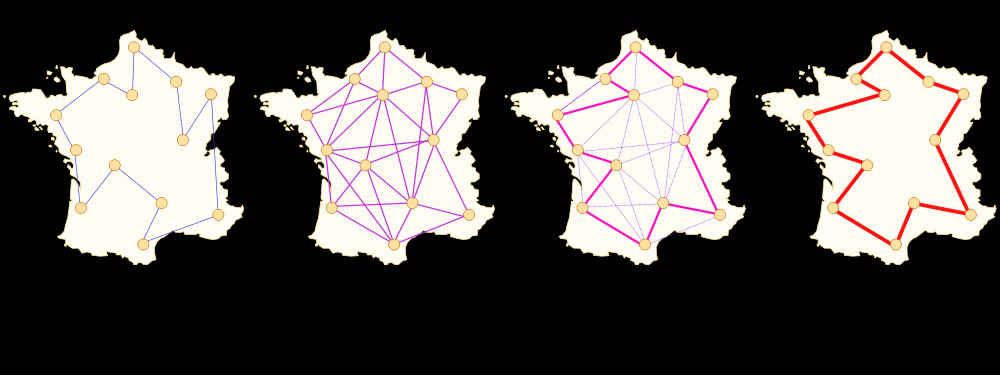


图1 蚁群算法的收敛过程

蚁群算法解决TSP问题的基本步骤包括初始化相关参数（比如蚁群规模，信息素重要程度因子，启发程度重要程度因子，信息素会发因子，信息素释放总量，最大迭代次数，迭代次数初值），构建解空间，更新信息素，判断是否终止。[1]

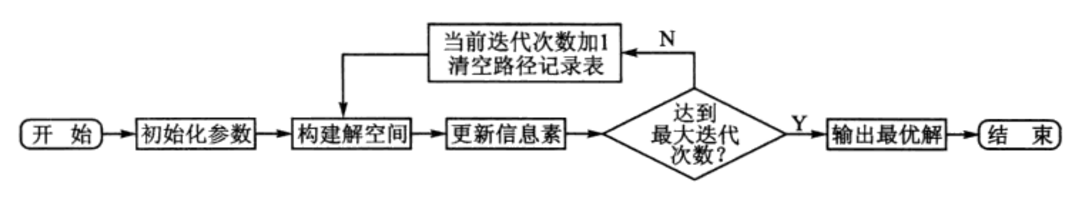


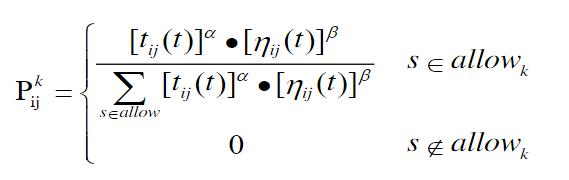
图2 蚁群算法图解

蚁群算法的2个核心步骤是路径构建和信息素更新。[2]



图3 现实中的蚁群

①随机概率是按照下列公式进行随机计算的：



其中：

i,j分别为起点和终点

nij =1/dij为能见度，是两点i,j路距离的倒数

为时间t时由i到j的信息素强度

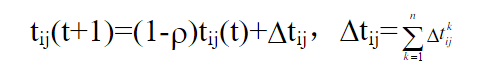
allowedk为尚未访问过的节点集合

为两常数，分别是信息素和和能见度的加权值

②信息素更新：

初始化信息浓度

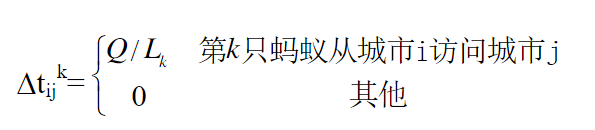
每一轮过后，问题空间中的所有路径上的信息素都会发生蒸发,然后，所有的蚂蚁根据自己构建的路径长度在它们本轮经过的边上释放信息素，公式如下：



其中：

表示蚂蚁k在景点i与景点j的连接路径上，释放的信息素浓度。

表示所有蚂蚁在景点i与景点j的连接路径上，释放的信息素浓度。



5.1.2数学模型

目标函数：

约束条件：

5.1.3模型求解

step1：建立蚂蚁对象，初始化包括节点数目、蚁群规模、迭代次数、信息素残留因子、启发因子(信息素的重要程度)、期望因子(节点间距离的重要程度)等参数。

step2：迭代开始，在迭代过程中，由于信息素的更新其结果会收敛。

step3：循环打印迭代列表，最终打印最短路径序列和最短路径长度。

**5.2问题二模型的建立与求解**

5.2.1建模过程

本题约束了游客游览总时间最长的条件，因此是在问题一的基础上设计惩罚函数，约束的实现是通过引入随机生成时长满足正态分布的时间量。

正态分布（英语：normal distribution）又名高斯分布（英语：Gaussian distribution），是一个非常常见的连续概率分布。正态分布在统计学上十分重要，经常用在自然和社会科学来代表一个不明的随机变量。

正态分布的数学期望值或期望值μ等于位置参数，决定了分布的位置；其方差 σ^2的开平方或标准差σ 等于尺度参数，决定了分布的幅度。

正态分布的概率密度函数曲线呈钟形，因此人们又经常称之为钟形曲线（类似于寺庙里的大钟，因此得名）。我们通常所说的标准正态分布是位置参数μ=0，尺度参数 σ^2 = 1的正态分布（见图中红色曲线）

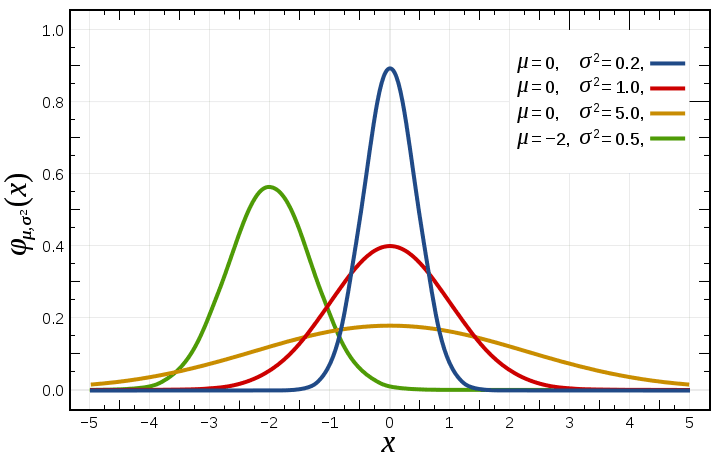


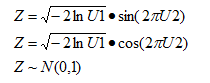
图4 正态分布（高斯分布）曲线

同时我们可知有两个节点存在时间约束，其它节点都有足够的调整时间，我们通过约束条件由算法实现了建模过程（结果见附录）。

5.2.2数学模型

如下是我们使用生成随机数的方法：

高斯分布随机数产生方法——Box-Muller算法



Y=u+(Z×sigma)  
u，sigma分别为期望和方差

目标函数:

约束条件:

5.2.3模型求解

step1：建立蚂蚁对象，初始化参数。

step2：引入互斥量数组和线程通信模块。

step3：引入等待函数(包括对是否选择等待的判断以及选择等待时获取占用互斥量线程的离开时间)。

step4：迭代开始，结果收敛。

step5：打印最优解路径序列、路径长度和参考时间长度5.3问题三模型的建立与求解。

5．3问题四模型的建立与求解

5.3.1建模过程

我们在第二问的模型基础上引入多线程模型，通过创建互斥量数组来保证每个景点都只能被一个线程(即一个团体)占用。

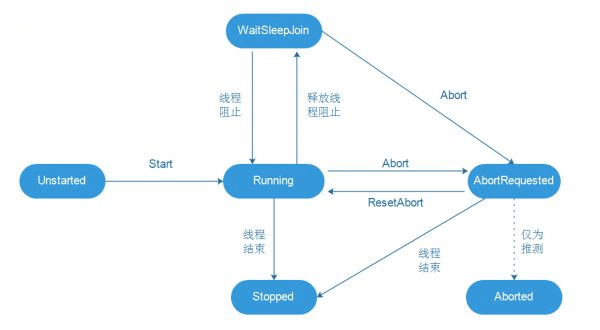


图5 单个线程的状态转换

由于遇到已被占用的互斥量，游客存在一定几率进行等待，这个几率让其随机产生，且在等待函数中要有对占用互斥量的他线程进行访问，即线程通信。

这一过程是对团体不能进入被别团体占据的景点的仿真，理论上只要蚁群规模

足够大且迭代次数够多，则必然能算出最优解。

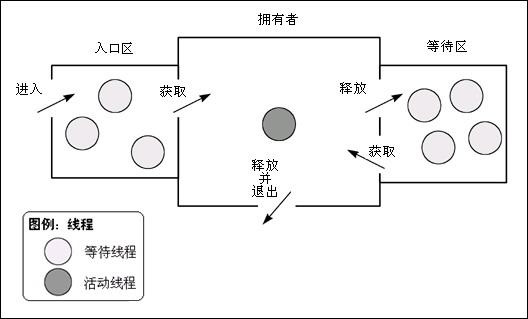


图6 多线程互斥图解

5.3.2数学模型

目标函数：

约束条件：

5.3.3模型求解

step1：建立蚂蚁对象，初始化参数。

step2：引入互斥量数组和线程通信模块。

step3：引入等待函数(包括对是否选择等待的判断以及选择等待时获取占用互斥量线程的离开时间)。

step4：迭代开始，结果收敛。

step5：打印最优解路径序列、路径长度和参考时间长度。

5.4问题四模型的建立与求解

5.4.1建模过程

在问题三的基础上，本题使旅游团的步行速度可以在1km/h到3km/h之间，因此速度是不确定性的，但是总的平均速度不能超过2km/h, 因此是在问题三之上增加一个随机速度的约束条件，这些团体的行进速度分布曲线满足正态分布，设计相应算法生成随机速度，取合理方差，且其数学期望为上下限的平均值，使其整体速度的平均值不超过上下限的平均值，即设计生成速度的随机函数。将之前两个问题中使用打表列出的移动时间用路程/随机速度取代，运行程序即可得到结果（结果见附录）。

5.4.2数学模型

目标函数：

约束条件：

5.4.3模型求解

step1：建立蚂蚁对象，初始化参数。

step2：引入行进速度生成函数。

step3：创建多线程，迭代开始，结果收敛。

step5：打印最优解路径序列、路径长度和参考时间长度

5.5问题五模型的建立与求解

5.5.1建模过程

我们在上一问的模型基础上加入扰动因素，对题设条件进一步仿真。

5.5.2数学模型

目标函数：

约束条件：

5.5.3模型求解

step1：建立蚂蚁对象，初始化参数。

step2：引入随机增加出发函数和扰动函数。

step3：创建多线程，迭代开始，扰动函数被随机调用，结果收敛。

step5：打印最优解路径序列、路径长度和参考时间长度。

1. **误差分析**

6.1 问题一

6.1.1 误差分析

由于是现代优化算法，蚁群算法给出的最优解是概率性的，可能会由于收敛函数设计的不到位而产生误差  
  
6.1.2 结果分析

我们将迭代结果加上约束打印出来，从而验证其结果的收敛。  
  
6.1.3 模型检验

由于题目中的节点数量为个位数，其全排列解空间也并非不能使用暴力算法的出结果。根据DFS算法得出的序列和总路程和根据蚁群算法得出的结果完全吻合，说明本模型正确合理，是可行的。  
  
  
6.2 问题二、问题三和问题四

6.2.1 误差分析

1.每增加一个随机函数，意味着我们需要更大的蚂蚁群体和迭代数才能确保函数的收敛是有效的。否则就会产生误差。

2.我们没有讨论其时间约束是应该设定成整型还是浮点数，这意味着我们对到达时间是否有近似行为，考虑到现实生活我们认为02:30:01和02:30:59同属一分钟，所以我们对结果近似化。在这一过程中可能发生了误差。

3.对于问题三和问题四，多线程的引入会使程序运行的开销大大增加，可能会出现硬件上的异常，产生误差。  
  
6.2.2 结果分析

通过打印中间结果可以判断函数的收敛性。且问题三的结论是问题四结论的极限（理论最优解）  
  
6.2.3 模型检验

随机生成序列计算出冲突次数和路径总长，和模型返回结果进行比对，由此说明模型的有效性。

6.3 问题五

6.3.1 误差分析

1．扰动函数让程序运行结果的随机程度再次增强，对于扰动函数的设计可能和现实有较大的差距。

2．多线程与随机程度的再次增强，会使得程序运行的开销指数级增加，出现异常的几率更高，被抛出的数据无法为函数收敛起到帮助，即提高了函数收敛的难度。也意味着最后的结果可能是局部最优解而不是全局最优解。  
  
6.3.2 结果分析

蚁群规模的增大和迭代次数的增加理论上抵消了随机程度增大对函数收敛的影响，打印中间结果显示函数收敛。  
  
6.3.3 模型检验

将随机生成的各种扰动因素和真实情况进行比对，来验证模型的可靠性。

1. **模型评价与改进**

7.1 问题一、二模型的评价

7.1.1 模型的优点

1.Dijkstra经典算法是一种遍历算法，每次能保证100%搜索到最短路径。

2.针对DFS，作为传统优化算法能在小范围内保证结果的准确性，适用于对模型的检验。

3.对于蚁群算法，实现了随机算法与局部搜索算法相结合，是启发式算法的改进，可以用来解决一些尚未找到有效算法的问题。蚁群算法可在后期快速收敛。

4.在问题二中，引入截断正态分布随机数，使算法更具有统计意义。

7.1.2 模型的缺点

1.原始Dijkstra算法在存储图形数据和运算时，基于网络的权矩阵，需要根据其节点与距离之间的关系，形成关联矩阵、邻接矩阵与距离矩阵，需要定义N×N的二维数组来存储数据，其中N为网络的节点数，当网络的节点数较大时，将占用大量的计算机内存。它的搜索效率非常低，时间开销较大。

2.蚁群算法在程序中随机数过多时，容易增大解空间的范围，从而使得收敛的速度大大降低，也使得结果可能出现更多的误差。

7.1.3 模型的改进

1.在原始Dijkstra算法的基础上，减小算法中成功搜索的搜索范围以尽快到达目标节点，并优化其存储结构。

2.对参数进行系统化调优，使其效率更高。

7.2 问题三、四模型的评价

7.2.1 模型的优点

1.蚁群算法，实现了随机算法与局部搜索算法相结合，是启发式算法的改进，可以用来解决一些尚未找到有效算法的问题，节省时空效率。

2.蚁群算法采用正反馈机制，使得搜索过程不断收敛，最终逼近最优解。

3.蚁群算法是一种基于种群的进化算法，具有本质并行性，易于实现并行。

4.搜索过程采用分布式计算方式，多个个体同时进行并行计算，大大提高了算法的计算能力和运行效率。

4.多线程实现互斥模拟，对题设进行仿真。

7.2.2 模型的缺点

1.如果α和β设置不当，导致求解速度很慢且所得解质量较差。

2.基本蚁群算法计算量大，求解所需时间较长。

3.多线程开销较大。

7.2.3 模型的改进

1.蚁群算法中的α和β的选取往往是通过经验来取得的，因而可以利用其他算法对蚁群系统参数α和β进行训练。

2.针对蚁群算法初始信息素匮乏的缺点，采用其他算法生成初始信息素分布，利用蚁群算法求精确解，从而提高时间效率和求解精度。亦可以在搜索过程中融入确定性局部搜索算法。

3.对于多线程，可以分别设置各个任务的优先级以优化性能。

7.3 问题五模型的评价

7.3.1 模型的优点

1.扰动函数增加随机性，使结果更具有统计意义和启发性。

2.建立的模型能与实际紧密联系，可操作性强，使模型更贴近实际，通用

性、推广性较强。

7.3.2 模型的缺点

1. 建模过程创新不足，约束条件可能不全面，知识面有待拓展。

2. 公式较复杂，计算较困难。

3. 时间段分得不够精细，比较粗糙。

4. 多线程与随机程度的再次增强，会使得程序运行的开销指数级增加，出现异常的几率更高，被抛出的数据无法为函数收敛起到帮助，即提高了函数收敛的难度。也意味着最后的结果可能是局部最优解而不是全局最优解。

7.3.3 模型的改进

1.对约束条件进行深入加强，对α、β进行训练。

2.对时间进行更细的划分。

1. **参考文献**

[1] M. Dorigo, V. Maniezzo, et A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics--Part B , volume 26, numéro 1, pages 29-41, 1996.

[2] 数据之巅，【C#代码实战】群蚁算法理论与实践全攻略——旅行商等路径优化问题的新方法，<https://www.cnblogs.com/asxinyu/p/Path_Optimization_Tsp_Problem_Ant_System_CSharp.html>，20150914。

1. 附件

9.1结果表格

**表3 第一问结果表格**

|  |  |  |
| --- | --- | --- |
| 出发景点 | 到达景点 | 步行距离(米) |
| 景石 | 森林小剧场 | 210 |
| 森林小剧场 | 儿童戏水场 | 265 |
| 儿童戏水场 | 游客服务中心 | 285 |
| 游客服务中心 | 阳光草坪 | 380 |
| 阳光草坪 | 儿童科普体验区 | 230 |
| 儿童科普体验区 | 湿地博物馆 | 260 |
| 湿地博物馆 | 湿地商业街 | 190 |
| 总步行距离(最短路线距离) | | 1820 |
| 最短路线(请用①~⑥序号标出) | | 景石→③→⑤→①→②→④→⑥→⑦ |
| 备注：可以根据需求增减本表格的行数 | | |

**表4 第二问结果表格(游览时间最长的路线信息)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 景点名称 | 达到时间点 | 游览时间  (停留时间,单位分钟) | 离开时间点 |
| 1 | 景石 | **12:00** | 0 | **12:00** |
| 2 | 阳光草坪 | 12:10 | 53 | **13:03** |
| 3 | 儿童科普体验区 | 13:09 | 36 | **13:45** |
| 4 | 森林小剧场 | 14:00 | 30 | **14:30** |
| 5 | 儿童戏水场 | 14:38 | 46 | **15:24** |
| 6 | 游客服务中心 | 15:33 | 23 | **15:56** |
| 7 | 湿地博物馆 | 16:02 | 42 | **16:44** |
| 8 | 湿地商业街 | 16:50 | 40 | **17:30** |
| **总的游览时间： 270分钟** | | | | |
| **总的步行时间： 60分钟** | | | | |

**表5 第三问结果表格**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **第一旅游团** | | | **第二旅游团** | | | **第三旅游团** | | |
|  | 达到  时间点 | 游览时间(停留时间,单位分钟) | 离开  时间点 | 达到  时间点 | 游览时间(停留时间,单位分钟) | 离开  时间点 | 达到  时间点 | 游览时间(停留时间,单位分钟) | 离开  时间点 |
| 景石 | **12:00** | **0** | **12:00** | **12:00** | **0** | **12:00** | **12:00** | **0** | **12:00** |
| 游客服务中心 | **15:08** | **10** | **15:18** | **12:09** | **30** | **12:39** | **15:48** | **30** | **16:18** |
| 阳光草坪 | **12:10** | **40** | **12:50** | **14:15** | **36** | **14:51** | **14:51** | **46** | **15:37** |
| 森林小剧场 | **13:58** | **30** | **14:30** | **13:30** | **30** | **14:00** | **13:00** | **30** | **13:30** |
| 儿童科普体验区 | **12:57** | **47** | **13:44** | **14:58** | **48** | **15:46** | **13:44** | **60** | **14:44** |
| 儿童戏水场 | **14:38** | **22** | **15:00** | **12:47** | **30** | **13:22** | **12:14** | **38** | **12:52** |
| 湿地博物馆 | **15:24** | **30** | **15:54** | **15:54** | **30** | **16:24** | **16:24** | **30** | **16:54** |
| 湿地商业街 | **16:00** | **90** | **17:30** | **16:30** | **60** | **17:30** | **17:00** | **30** | **17:30** |
| **总步行时间** | **59分钟** | | | **61分钟** | | | **66分钟** | | |
| **总游览时间** | **269分钟** | | | **64分钟** | | | **264分钟** | | |
| **总等待时间** | **2分钟** | | | **5分钟** | | | **0分钟** | | |

**表6 第四问结果表格**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **第一旅游团** | | | | **第二旅游团** | | | | **第三旅游团** | | | |
|  | 达到  时间点 | 游览时间(停留时间,单位分钟) | 离开  时间点 | 达到下一个景点的步行速度 | 达到  时间点 | 游览时间(停留时间,单位分钟) | 离开  时间点 | 达到下一个景点的步行速度 | 达到  时间点 | 游览时间(停留时间,单位分钟) | 离开  时间点 | 达到下一个景点的步行速度 |
| 景石 | **12:00** | **0** | **12:00** | **2** | **12:00** | **0** | **12:00** | **2** | **12:00** | **0** | **12:00** | **2** |
| 游客服务中心 | **15:08** | **10** | **15:18** | **2** | **12:09** | **30** | **12:39** | **2** | **15:48** | **30** | **16:18** | **2** |
| 阳光草坪 | **12:10** | **40** | **12:50** | **2** | **14:15** | **36** | **14:51** | **2** | **14:51** | **46** | **15:37** | **2** |
| 森林小剧场 | **13:58** | **30** | **14:30** | **2** | **13:30** | **30** | **14:00** | **2** | **13:00** | **30** | **13:30** | **2** |
| 儿童科普体验区 | **12:57** | **47** | **13:44** | **2** | **14:58** | **48** | **15:46** | **2** | **13:44** | **60** | **14:44** | **2** |
| 儿童戏水场 | **14:38** | **22** | **15:00** | **2** | **12:47** | **30** | **13:22** | **2** | **12:14** | **38** | **12:52** | **2** |
| 湿地博物馆 | **15:24** | **30** | **15:54** | **2** | **15:54** | **30** | **16:24** | **2** | **16:24** | **30** | **16:54** | **2** |
| 湿地商业街 | **16:00** | **90** | **17:30** | **2** | **16:30** | **60** | **17:30** | **2** | **17:00** | **30** | **17:30** | **2** |
| **总步行时间** | **59分钟** | | | | **61分钟** | | | | **66分钟** | | | |
| **总游览时间** | **269分钟** | | | | **64分钟** | | | | **264分钟** | | | |
| **总等待时间** | **2分钟** | | | | **5分钟** | | | | **0分钟** | | | |
| **平均速度** | 2 | | | | 2 | | | | 2 | | | |

9.2 实现代码

9.2.1 问题一

Dijkstra算法：求景点之间最短距离

#-\*- coding: utf-8 -\*-

import json

nodes = ('0', '1', '2', '3', '4', '5', '6','7')

distances = {

'0': {'1': 300, '2': 360, '3': 210, '4': 590, '5': 475, '6': 500, '7': 690},

'1': {'0': 300, '2': 380, '3': 270, '4': 230, '5': 285, '6': 200, '7': 390},

'2': {'0': 360, '1': 380, '3': 510, '4': 230, '5': 765, '6': 580, '7': 760},

'3': {'0': 210, '1': 270, '2': 510, '4': 470, '5': 265, '6': 450, '7': 640},

'4': {'0': 590, '1': 230, '2': 230, '3': 470, '5': 515, '6': 260, '7': 450},

'5': {'0': 475, '1': 285, '2': 765, '3': 265, '4': 515, '6': 460, '7': 650},

'6': {'0': 500, '1': 200, '2': 580, '3': 450, '4': 260, '5': 460, '7': 190},

'7': {'0': 690, '1': 390, '2': 760, '3': 640, '4': 450, '5': 650, '6': 190},

}

unvisited = {node: None for node in nodes} #把None作为无穷大使用

visited = {}#用来记录已经松弛过的数组

#current = '0' #要找B点到其他点的距离

currentDistance = 0

order = []

#unvisited[current] = currentDistance#B到B的距离记为0

def DistaceInOneNode(current, visited = visited, unvisited = unvisited, nodes = nodes, currentDistance = currentDistance, distances = distances):

unvisited[current] = currentDistance

while True:

for neighbour, distance in distances[current].items():

if neighbour not in unvisited: continue#被访问过了，跳出本次循环

newDistance = currentDistance + distance#新的距离

if unvisited[neighbour] is None or unvisited[neighbour] > newDistance:#如果两个点之间的距离之前是无穷大或者新距离小于原来的距离

unvisited[neighbour] = newDistance#更新距离

visited[current] = currentDistance#这个点已经松弛过，记录

del unvisited[current]#从未访问过的字典中将这个点删除

if not unvisited: break#如果所有点都松弛过，跳出此次循环

candidates = [node for node in unvisited.items() if node[1]]#找出目前还有哪些点未松弛过

current, currentDistance = sorted(candidates, key = lambda x: x[1])[0]#找出目前可以用来松弛的点

order.append(current)

return visited

print

rlt = DistaceInOneNode('')

rlt\_key = sorted(rlt.items(), key=lambda d:d[0])

#print(rlt\_key)

file = open('data.txt', 'a')

file.write(str(rlt\_key))

file.close()

dfs算法：求解最短路径

#include <stdio.h>

#include <string.h>

#include <string>

#include <iostream>

using namespace std;

const int N = 9; //景点数量 + 1

int a[N][N] = { //距离邻接矩阵

{0},

{0,0,300,360,210,590,475,500,690},

{0,300,0,380,270,230,285,200,390},

{0,360,380,0,510,230,765,580,770},

{0,210,270,510,0,470,265,450,640},

{0,590,230,230,470,0,515,260,450},

{0,475,285,765,265,515,0,460,650},

{0,500,200,580,450,260,460,0,190},

{0,690,390,760,640,450,650,190,0}};

//地点名

string sc[] = {"景石", "游客服务中心", "阳光草坪", "森林小剧场", "儿童科普体验区", "儿童戏水场", "湿地博物馆", "湿地商业街"};

int visited[N] = {0}; //初始全为0

int c1[99]; //ans

int c2[99] = {0, 1}; //temp

int Min = 99999999; //minimum

int cnt = 1; //index for Array

bool check() //判断是否所有点都访问过

{

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

{

if(!visited[i])

return false;

}

return true;

}

void dfs(int cur, int sum)

{

if (sum > Min) return; // 如果之前走过的大于最短路，没必要再尝试了

c2[cnt] = cur; //存储路径到临时数组c2

if (cur == 8 && check())

{

printf("[过程量]%d\n", sum);

if (sum < Min) //若找到更优解，更新结果，将临时数组赋给结果数组

{

Min = sum;

memcpy(c1, c2, sizeof(c2));

}

return;

}

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

{

if (a[cur][i] > 0 && visited[i] == 0)

{

cnt ++;

visited[i] = 1;

dfs(i, sum + a[cur][i]);

cnt --;

visited[i] = 0;

}

}

}

int main()

{

visited[1] = 1;

dfs(1, 0);

printf("Min: %d\n最短路径：", Min);

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

{

if(!c1[i]) break;

printf("%d ", c1[i]-1);

}

printf("\n\n 表格3 第一问结果\n\n");

printf("出发景点\t到达景点\t步行距离（米）\n");

int sum = 0;

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

{

if(!c1[i + 1]) break;

sum += a[c1[i]][c1[i + 1]];

if(sc[c1[i] - 1] == "景石")

cout<<sc[c1[i] - 1]<<"\t\t"<<sc[c1[i + 1] - 1]<<"\t"<<a[c1[i]][c1[i + 1]]<<endl;

else cout<<sc[c1[i] - 1]<<"\t"<<sc[c1[i + 1] - 1]<<"\t"<<a[c1[i]][c1[i + 1]]<<endl;

}

printf("\n总路程 = %d\n", sum);

return 0;

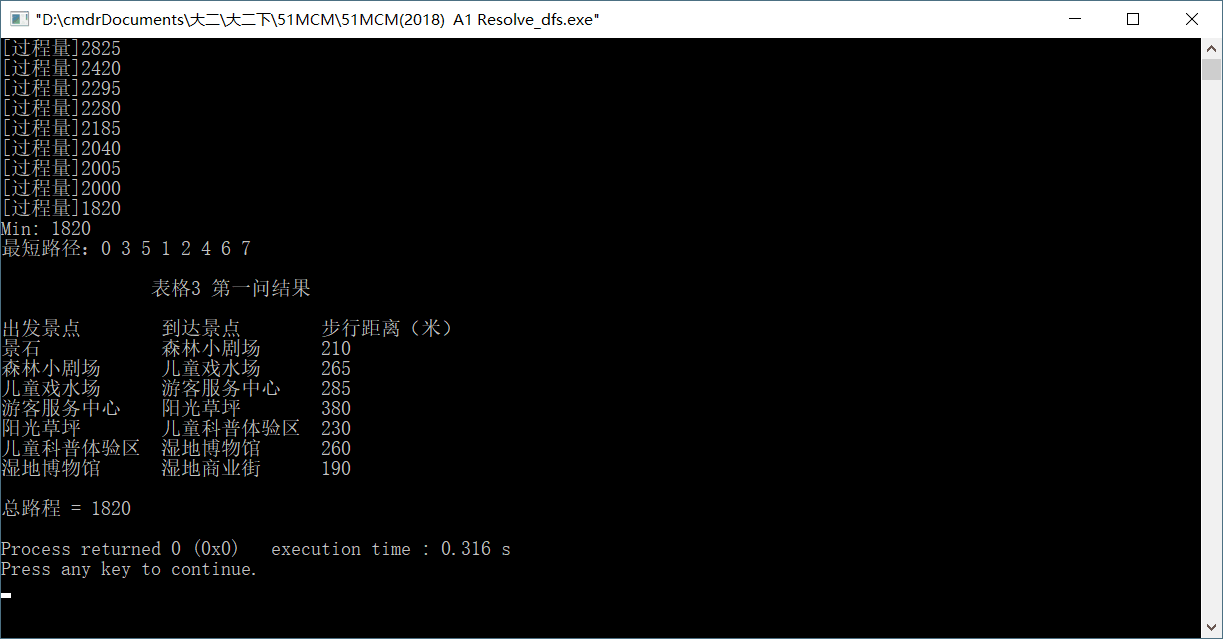
}

图7 问题一运行结果

蚁群算法：求解最短路径

#include<iostream>

#include<math.h>

#include<time.h>

#include<string.h>

using namespace std;

const double ALPHA = 2.0; //启发因子，信息素的重要程度

const double BETA = 3.0; //期望因子，景点间距离的重要程度

const double ROU = 0.5; //信息素残留参数

//0 3 5 1 2 4 6 7

const int N\_ANT\_COUNT = 100; //蚂蚁数量

const int N\_IT\_COUNT = 3000; //迭代次数

const int N\_CITY\_COUNT = 8; //景点数量

const double DBQ = 100.0; //总的信息素

const double DB\_MAX = 10e9; //一个标志数，10的9次方

double g\_Trial[N\_CITY\_COUNT][N\_CITY\_COUNT];

//两两景点间信息素，就是环境信息素

double g\_Distance[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{ 0, 300, 360, 210, 530, 475, 500, 690 },

{ 300, 0, 380, 270, 230, 285, 200, 390 },

{ 360, 380, 0, 510, 230, 665, 490, 680 },

{ 210, 270, 510, 0, 470, 265, 450, 640 },

{ 530, 230, 230, 470, 0, 515, 260, 450 },

{ 475, 285, 665, 265, 515, 0, 460, 650 },

{ 500, 200, 490, 450, 260, 460, 0, 190 },

{ 690, 390, 680, 640, 450, 650, 190, 0 }

};//两两景点间距离

double bestLength = DB\_MAX;

//返回指定范围内的随机整数

int rnd(int nLow, int nUpper)

{

return nLow + (nUpper - nLow) \* rand() / (RAND\_MAX + 1);

}

//返回指定范围内的随机浮点数

double rnd(double dbLow, double dbUpper)

{

double dbTemp = rand() / ((double)RAND\_MAX + 1.0);

return dbLow + dbTemp \* (dbUpper - dbLow);

}

//返回浮点数四舍五入取整后的浮点数

double ROUND(double dbA)

{

return(double)((int)(dbA + 0.5));

}

//定义蚂蚁类

class CAnt

{

public:

CAnt(void);

~CAnt(void);

public:

int m\_nPath[N\_CITY\_COUNT]; //蚂蚁走的路径

double m\_dbPathLength; //蚂蚁走过的路径长度

int m\_nAllowedCity[N\_CITY\_COUNT]; //没去过的景点

int m\_nCurCityNo; //当前所在景点编号

int m\_nMovedCityCount; //已经去过的景点数量

public:

int ChooseNextCity(); //选择下一个景点

void Init();//初始化

void Move();//蚂蚁在景点间移动

void Search(); //搜索路径

void CalPathLength(); //计算蚂蚁走过的路径长度

void CalPathLengthTest();//测试用

};

//构造函数

CAnt::CAnt(void)

{

}

//析构函数

CAnt::~CAnt(void)

{

}

//初始化函数，蚂蚁搜索前调用

void CAnt::Init()

{

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

m\_nAllowedCity[i] = 1; //设置全部景点为没有去过

m\_nPath[i] = 0; //蚂蚁走的路径全部设置为0

}

//蚂蚁走过的路径长度设置为0

m\_dbPathLength = 0.0;

//选择一个出发景点

m\_nCurCityNo = 0;

//m\_nCurCityNo = rnd(0,N\_CITY\_COUNT);

//把出发景点保存入路径数组中

m\_nPath[0] = m\_nCurCityNo;

//标识出发景点为已经去过了

m\_nAllowedCity[m\_nCurCityNo] = 0;

//已经去过的景点数量设置为1

m\_nMovedCityCount = 1;

}

//选择下一个景点

//返回值 为景点编号

int CAnt::ChooseNextCity()

{

int nSelectedCity = -1; //返回结果，先暂时把其设置为-1

//==============================================================================

//计算当前景点和没去过的景点之间的信息素总和

double dbTotal = 0.0;

double prob[N\_CITY\_COUNT]; //保存各个景点被选中的概率

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

prob[i] = pow(g\_Trial[m\_nCurCityNo][i], ALPHA)\*pow(1.0 / g\_Distance[m\_nCurCityNo][i], BETA);

//该景点和当前景点间的信息素

dbTotal = dbTotal + prob[i]; //累加信息素，得到总和

}

else //如果景点去过了，则其被选中的概率值为0

{

prob[i] = 0.0;

}

}

//==============================================================================

//进行轮盘选择

double dbTemp = 0.0;

if (dbTotal> 0.0) //总的信息素值大于0

{

dbTemp = rnd(0.0, dbTotal); //取一个随机数

for (int i = 0; i<N\_CITY\_COUNT; i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

dbTemp = dbTemp - prob[i]; //这个操作相当于转动轮盘

if (dbTemp < 0.0) //轮盘停止转动，记下景点编号，直接跳出循环

{

nSelectedCity = i;

break;

}

}

}

}

//==============================================================================

//如果景点间的信息素非常小 ( 小到比double能够表示的最小的数字还要小 )

//那么由于浮点运算的误差原因，上面计算的概率总和可能为0

//会出现经过上述操作，没有景点被选择出来

//出现这种情况，就把第一个没去过的景点作为返回结果

if (nSelectedCity == -1)

{

for (int i = 0; i<N\_CITY\_COUNT; i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

nSelectedCity = i;

break;

}

}

}

//==============================================================================

//返回结果，就是景点的编号

return nSelectedCity;

}

//蚂蚁在景点间移动

void CAnt::Move()

{

int nCityNo = ChooseNextCity(); //选择下一个景点

m\_nPath[m\_nMovedCityCount] = nCityNo; //保存蚂蚁走的路径

m\_nAllowedCity[nCityNo] = 0;//把这个景点设置成已经去过了

m\_nCurCityNo = nCityNo; //改变当前所在景点为选择的景点

m\_nMovedCityCount++; //已经去过的景点数量加1

}

//蚂蚁进行搜索一次

void CAnt::Search()

{

Init();//蚂蚁搜索前，先初始化

//如果蚂蚁去过的景点数量小于景点数量，就继续移动

while (m\_nMovedCityCount < N\_CITY\_COUNT)

{

Move();

}

//完成搜索后计算走过的路径长度

CalPathLength();

//printf("%.0lf\n", m\_dbPathLength);

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLength()

{

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for (int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i - 1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

}

//加上从最后景点返回出发景点的距离

/\*

n = m\_nPath[0];

m\_dbPathLength = m\_dbPathLength + g\_Distance[m][n];

\*/

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLengthTest()

{

//

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for (int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i - 1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

cout << n << " " << m << " " << " " << g\_Distance[n][m] << " " << m\_dbPathLength << endl;

}

//加上从最后景点返回出发景点的距离

/\*

n = m\_nPath[0];

m\_dbPathLength = m\_dbPathLength + g\_Distance[m][n];

\*/

}

//tsp类

class CTsp

{

public:

CTsp(void);

~CTsp(void);

public:

CAnt m\_cAntAry[N\_ANT\_COUNT]; //蚂蚁数组

CAnt m\_cBestAnt; //定义一个蚂蚁变量，用来保存搜索过程中的最优结果

//该蚂蚁不参与搜索，只是用来保存最优结果

public:

//初始化数据

void InitData();

//开始搜索

void Search();

//更新环境信息素

void UpdateTrial();

};

//构造函数

CTsp::CTsp(void)

{

}

CTsp::~CTsp(void)

{

}

//初始化数据

void CTsp::InitData()

{

//先把最优蚂蚁的路径长度设置成一个很大的值

m\_cBestAnt.m\_dbPathLength = DB\_MAX;

//计算两两景点间距离

//这个是一开始就写好的

//初始化环境信息素，先把景点间的信息素设置成一样

//这里设置成1.0，设置成多少对结果影响不是太大，对算法收敛速度有些影响

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

for (int j = 0; j<N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = 1.0;

}

}

}

//更新环境信息素

void CTsp::UpdateTrial()

{

//临时数组，保存各只蚂蚁在两两景点间新留下的信息素

double dbTempAry[N\_CITY\_COUNT][N\_CITY\_COUNT];

memset(dbTempAry, 0, sizeof(dbTempAry)); //先全部设置为0

//计算新增加的信息素,保存到临时数组里

int m = 0;

int n = 0;

for (int i = 0; i < N\_ANT\_COUNT; i++) //计算每只蚂蚁留下的信息素

{

for (int j = 1; j < N\_CITY\_COUNT; j++)

{

m = m\_cAntAry[i].m\_nPath[j];

n = m\_cAntAry[i].m\_nPath[j - 1];

dbTempAry[n][m] = dbTempAry[n][m] + DBQ / m\_cAntAry[i].m\_dbPathLength;

dbTempAry[m][n] = dbTempAry[n][m];

}

//最后景点和开始景点之间的信息素

n = m\_cAntAry[i].m\_nPath[0];

dbTempAry[n][m] = dbTempAry[n][m] + DBQ / m\_cAntAry[i].m\_dbPathLength;

dbTempAry[m][n] = dbTempAry[n][m];

}

//==================================================================

//更新环境信息素

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

for (int j = 0; j < N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = g\_Trial[i][j] \* ROU + dbTempAry[i][j]; //最新的环境信息素 = 留存的信息素 + 新留下的信息素

}

}

}

void CTsp::Search()

{

char cBuf[256]; //打印信息用

//在迭代次数内进行循环

for (int i = 0; i < N\_IT\_COUNT; i++)

{

//每只蚂蚁搜索一遍

for (int j = 0; j < N\_ANT\_COUNT; j++)

{

m\_cAntAry[j].Search();

//cout<<m\_cAntAry[j].m\_dbPathLength<<endl;

//查看变量，打断点

}

//保存最佳结果

for (int j = 0; j < N\_ANT\_COUNT; j++)

{

if (m\_cAntAry[j].m\_dbPathLength < bestLength)

{

m\_cBestAnt = m\_cAntAry[j];

for (int k = 0; k<N\_CITY\_COUNT; k++)

{

sprintf(cBuf, "%d ", m\_cAntAry[j].m\_nPath[k]);

printf(cBuf);

}

cout << endl;

bestLength = m\_cAntAry[j].m\_dbPathLength;

m\_cAntAry[j].CalPathLengthTest();

//cout<<"\n"<<bestLength<<endl;

}

}

//更新环境信息素

UpdateTrial();

}

int main()

{

//用当前时间点初始化随机种子，防止每次运行的结果都相同

time\_t tm;

time(&tm);

unsigned int nSeed = (unsigned int)tm;

srand(nSeed);

//开始搜索

CTsp tsp;

//g\_distanc

tsp.InitData(); //初始化

tsp.Search(); //开始搜索

//输出结果

printf("\nThe best tour is :\n");

char cBuf[128];

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

sprintf(cBuf, "%d ", tsp.m\_cBestAnt.m\_nPath[i]);

if (i % 20 == 0)

{

printf("\n");

}

printf(cBuf);

}

printf("\n\nPress any key to exit!");

getchar();

return 0;

}

9.2.2 问题二

对问题一中的蚁群算法代码进行改动，增加时间约束。

#include<iostream>

#include<math.h>

#include<time.h>

#include<string.h>

#include <windows.h>

#include <process.h>

#include <fstream>

#include <stdlib.h>

#include <cstdio>

#include <random>

#include<set>

using namespace std;

const double ALPHA = 1.0; //启发因子，信息素的重要程度

const double BETA = 2.0; //期望因子，景点间距离的重要程度

const double ROU = 0.5; //信息素残留参数

//0 3 5 1 2 4 6 7

const int N\_ANT\_COUNT = 100; //蚂蚁数量

const int N\_IT\_COUNT = 1000; //迭代次数

const int N\_CITY\_COUNT = 8; //景点数量

const double DBQ = 200.0; //总的信息素

const double DB\_MAX = 10e9; //一个标志数，10的9次方

double g\_Trial[N\_CITY\_COUNT][N\_CITY\_COUNT];

//两两景点间信息素，就是环境信息素

double g\_Distance[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{ 0, 300, 360, 210, 530, 475, 500, 690},

{ 300, 0, 380, 270, 230, 285, 200, 390},

{ 360, 380, 0, 510, 230, 665, 490, 680},

{ 210, 270, 510, 0, 470, 265, 450, 640},

{ 530, 230, 230, 470, 0, 515, 260, 450},

{ 475, 285, 665, 265, 515, 0, 460, 650},

{ 500, 200, 490, 450, 260, 460, 0, 190},

{ 690, 390, 680, 640, 450, 650, 190, 0}

};//两两景点间距离

double g\_Walk[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{0,9,10.8,6.3,15.9,14.25,15,20.7},

{9,0,11.4,8.1,6.9,8.55,6,11.7},

{10.8,11.4,0,15.3,6.9,19.95,14.7,20.4},

{6.3,8.1,15.3,0,14.1,7.95,13.5,19.2},

{15.9,6.9,6.9,14.1,0,15.45,7.8,13.5},

{14.25,8.55,19.95,7.95,15.45,0,13.8,19.5},

{15,6,14.7,13.5,7.8,13.8,0,},

{20.7,11.7,20.4,19.2,13.5,19.5,5.7,0}

};

double bestLength = DB\_MAX;

//检查节点是否都被访问

bool check(int\* group)

{

int sizeofgroup = sizeof(group);

int flag = 0;

for(int i=0; i<sizeofgroup-1; i++)

{

if(group[i]==0)

{

flag+=1;

}

}

if(flag == sizeofgroup-1)return true;

else return false;

}

//返回指定范围内的随机整数

int rnd(int nLow, int nUpper)

{

return nLow + (nUpper - nLow) \* rand() / (RAND\_MAX + 1);

}

//返回指定范围内的随机浮点数

double rnd(double dbLow, double dbUpper)

{

double dbTemp = rand() / ((double)RAND\_MAX + 1.0);

return dbLow + dbTemp \* (dbUpper - dbLow);

}

//返回浮点数四舍五入取整后的浮点数

double ROUND(double dbA)

{

return(double)((int)(dbA+0.5));

}

random\_device rd;

mt19937 gen(rd());

//生成一个符合截断正态分布的随机数

double normal\_rdn(double lowwer, double upper, double v) //下限，上限，方差

{

double e = (lowwer + upper) / 2;

normal\_distribution<double> normal(e, v);

double ans = (double)normal(gen);

if(ans >= lowwer && ans <= upper) return ans;

else return (double)normal\_rdn(lowwer, upper, v);

}

//森林小剧场，dt = 30，且开放时间为整除30的数

//因为到湿地商业街还需时间，故t <= 240

//编号 3

bool theater(int t)

{

if(t % 30 || t > 240)

return false;

return true;

}

//游客服务中心，10 <= dt <= 30，且t\_end <= 240

//编号 1

bool s\_center(int t, int dt)

{

if(t + dt > 240)

return false;

return true;

}

//delta time Array：在每个景点停留时间上下限

double dtAry[N\_CITY\_COUNT][2] = {

{0, 0},//0 source, ignore

{10, 30},//1

{20, 60},//2

{30, 30},//3,constant int

{30, 60},//4

{20, 60},//5

{30, 60},//6

{30, 30}//7 destination, ignore it in problem 2

};

//返回一个介于上限和下限之间的rnd

double randTime(int CITY\_NO)

{

int a = dtAry[CITY\_NO][0], b =dtAry[CITY\_NO][1];

return normal\_rdn(b, a, 1);

}

//定义蚂蚁类

class CAnt

{

public:

CAnt(void);

~CAnt(void);

public:

int m\_nPath[N\_CITY\_COUNT]; //蚂蚁走的路径

double m\_dbPathLength; //蚂蚁走过的路径长度

double m\_dbPathTime;//蚂蚁用的时间

int m\_nAllowedCity[N\_CITY\_COUNT]; //没去过的景点

int m\_nCurCityNo; //当前所在景点编号

int m\_nMovedCityCount; //已经去过的景点数量

double m\_nTimeCost[N\_CITY\_COUNT];//每个节点进入的时间

double m\_nTimeEachCost[N\_CITY\_COUNT]; //= {0};//每个节点花的时间

public:

int ChooseNextCity(); //选择下一个景点

void Init();//初始化

void Move();//蚂蚁在景点间移动

void Search(); //搜索路径

void CalPathLength(); //计算蚂蚁走过的路径长度

void CalPathLengthTest();//测试用

void CalPathTime();//计算蚂蚁用的时间

void CalPathTimeTest();//test

};

//构造函数

CAnt::CAnt(void)

{

}

//析构函数

CAnt::~CAnt(void)

{

}

//初始化函数，蚂蚁搜索前调用

void CAnt::Init()

{

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m\_nAllowedCity[i] = 1; //设置全部景点为没有去过

m\_nPath[i] = 0; //蚂蚁走的路径全部设置为0

}

//蚂蚁走过的路径长度设置为0

m\_dbPathLength = 0.0;

//选择一个出发景点

m\_nCurCityNo = 0;

//m\_nCurCityNo = rnd(0,N\_CITY\_COUNT);

//把出发景点保存入路径数组中

m\_nPath[0] = m\_nCurCityNo;

//标识出发景点为已经去过了

m\_nAllowedCity[m\_nCurCityNo] = 0;

//每个节点所要花费的时间，初始点设为0

m\_nTimeEachCost[m\_nCurCityNo] = 0;

//已经去过的景点数量设置为1

m\_nMovedCityCount = 1;

//把出发时间定为0

m\_nTimeCost[m\_nCurCityNo] = 0;

}

//选择下一个景点

//返回值 为景点编号

int CAnt::ChooseNextCity()

{

int nSelectedCity=-1; //返回结果，先暂时把其设置为-1

//==============================================================================

//计算当前景点和没去过的景点之间的信息素总和

double dbTotal=0.0;

double prob[N\_CITY\_COUNT]; //保存各个景点被选中的概率

double temp\_enteringTime[N\_CITY\_COUNT] = {0};

double TimeHereCost = randTime(m\_nCurCityNo);

m\_nTimeEachCost[m\_nCurCityNo] = TimeHereCost;

for (int i = 0; i < N\_CITY\_COUNT;i++)

{

temp\_enteringTime[i] = m\_nTimeCost[m\_nCurCityNo] + g\_Walk[m\_nCurCityNo][i] + TimeHereCost;

//对景点号进行确认，如果不符合条件直接概率调零

if(i == 1)

{

//我们需要临时时间

if(!s\_center(temp\_enteringTime[i],randTime(i)))

{

prob[i] = 0;

continue;

}

}

if(i == 3)

{

if(!theater(temp\_enteringTime[i]))

{

prob[i]=0;

continue;

}

}

if (m\_nAllowedCity[i] == 1) //景点没去过

{

prob[i]=pow(g\_Trial[m\_nCurCityNo][i],ALPHA)\*pow(1.0/g\_Distance[m\_nCurCityNo][i],BETA);

//该景点和当前景点间的信息素

dbTotal=dbTotal+prob[i]; //累加信息素，得到总和

}

else //如果景点去过了，则其被选中的概率值为0

{

prob[i]=0.01;

}

}

//==============================================================================

//进行轮盘选择

double dbTemp=0.0;

if (dbTotal> 0.0) //总的信息素值大于0

{

dbTemp=rnd(0.0,dbTotal); //取一个随机数

for (int i=0;i<N\_CITY\_COUNT;i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

dbTemp=dbTemp-prob[i]; //这个操作相当于转动轮盘

if (dbTemp < 0.0&& i < N\_CITY\_COUNT-1) //轮盘停止转动，记下景点编号，直接跳出循环

{

nSelectedCity=i;

//发现不是终点则可以选

break;

}

else if(dbTemp < 0.0&& i == N\_CITY\_COUNT-1)

{

if(check(m\_nAllowedCity)){nSelectedCity=i;break;}

//确认是否全部都选过了

else break;

}

}

}

}

if(nSelectedCity == -1)

{

for (int i=0;i<N\_CITY\_COUNT;i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

//对i进行判断

nSelectedCity=i;

break;

}

}

}

//返回结果，就是景点的编号

//记录进入时间

m\_nTimeCost[nSelectedCity] = temp\_enteringTime[nSelectedCity];

return nSelectedCity;

}

//蚂蚁在景点间移动

void CAnt::Move()

{

int nCityNo=ChooseNextCity(); //选择下一个景点

m\_nPath[m\_nMovedCityCount]=nCityNo; //保存蚂蚁走的路

m\_nAllowedCity[nCityNo]=0;//把这个景点设置成已经去过了

m\_nCurCityNo=nCityNo; //改变当前所在景点为选择的景点

m\_nMovedCityCount++; //已经去过的景点数量加1

}

//蚂蚁进行搜索一次

void CAnt::Search()

{

Init();//蚂蚁搜索前，先初始化

//如果蚂蚁去过的景点数量小于景点数量，就继续移动

while(m\_nMovedCityCount < N\_CITY\_COUNT)

{

Move();

}

//完成搜索后计算走过的路径长度

CalPathLength();

CalPathTime();

//printf("%.0lf\n", m\_dbPathLength);

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLength()

{

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for(int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i-1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

}

//加上从最后景点返回出发景点的距离

/\*

n = m\_nPath[0];

m\_dbPathLength = m\_dbPathLength + g\_Distance[m][n];

\*/

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLengthTest()

{

//

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for(int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i-1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

cout<<n<<" "<<m<<" "<<" "<<g\_Distance[n][m]<<" "<<m\_dbPathLength<<endl;

}

}

//计算蚂蚁游玩的时间

void CAnt::CalPathTime()

{

m\_dbPathTime = 0.0; //先置0

int m = 0;

int n = 0;

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m = m\_nTimeEachCost[i];

m\_dbPathTime = m\_dbPathTime + m;

}

}

//计算蚂蚁游玩的时间

void CAnt::CalPathTimeTest()

{

m\_dbPathTime = 0.0; //先置0

int m = 0;

int n = 0;

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m = m\_nTimeEachCost[i];

m\_dbPathTime = m\_dbPathTime + m;

}

cout<<m\_dbPathTime<<endl;

}

//tsp类

class CTsp

{

public:

CTsp(void);

~CTsp(void);

public:

CAnt m\_cAntAry[N\_ANT\_COUNT]; //蚂蚁数组

CAnt m\_cBestAnt; //定义一个蚂蚁变量，用来保存搜索过程中的最优结果

//该蚂蚁不参与搜索，只是用来保存最优结果

public:

//初始化数据

void InitData();

//开始搜索

void Search();

//更新环境信息素

void UpdateTrial();

};

//构造函数

CTsp::CTsp(void)

{

}

CTsp::~CTsp(void)

{

}

//初始化数据

void CTsp::InitData()

{

//先把最优蚂蚁的路径长度设置成一个很大的值

m\_cBestAnt.m\_dbPathLength = DB\_MAX;

//计算两两景点间距离

//初始化环境信息素，先把景点间的信息素设置成一样

//这里设置成1.0，设置成多少对结果影响不是太大，对算法收敛速度有些影响

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

for(int j = 0; j<N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = 1.0;

}

}

}

//更新环境信息素

void CTsp::UpdateTrial()

{

//临时数组，保存各只蚂蚁在两两景点间新留下的信息素

double dbTempAry[N\_CITY\_COUNT][N\_CITY\_COUNT];

memset(dbTempAry, 0, sizeof(dbTempAry)); //先全部设置为0

//计算新增加的信息素,保存到临时数组里

int m = 0;

int n = 0;

for(int i = 0; i < N\_ANT\_COUNT; i++) //计算每只蚂蚁留下的信息素

{

for (int j = 1; j < N\_CITY\_COUNT; j++)

{

m = m\_cAntAry[i].m\_nPath[j];

n = m\_cAntAry[i].m\_nPath[j - 1];

dbTempAry[n][m] = dbTempAry[n][m]+DBQ/m\_cAntAry[i].m\_dbPathLength;

//要对时间点进行惩罚

dbTempAry[m][n] = dbTempAry[n][m];

}

//最后景点和开始景点之间的信息素

n = m\_cAntAry[i].m\_nPath[0];

dbTempAry[n][m] = dbTempAry[n][m]+ DBQ/m\_cAntAry[i].m\_dbPathLength;

dbTempAry[m][n] = dbTempAry[n][m];

}

//更新环境信息素

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

for (int j = 0; j < N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = g\_Trial[i][j] \* ROU + dbTempAry[i][j]; //最新的环境信息素 =留存的信息素 + 新留下的信息素

}

}

}

void CTsp::Search()

{

char cBuf[256]; //打印信息用

//在迭代次数内进行循环

for(int i = 0; i < N\_IT\_COUNT; i++)

{

//每只蚂蚁搜索一遍

for (int j = 0; j < N\_ANT\_COUNT;j++)

{

m\_cAntAry[j].Search();

}

//保存最佳结果

for (int j = 0; j < N\_ANT\_COUNT; j++)

{

if(m\_cAntAry[j].m\_nPath[N\_CITY\_COUNT-1] != 7

||m\_cAntAry[j].m\_nTimeCost[N\_CITY\_COUNT-1] >= 300)continue;

if ((330-(m\_cAntAry[j].m\_dbPathLength)\*30) > (330-(m\_cBestAnt.m\_dbPathLength)\*30))

{

m\_cBestAnt = m\_cAntAry[j];

for(int k = 0; k<N\_CITY\_COUNT;k++)

{

sprintf(cBuf, "%d ", m\_cAntAry[j].m\_nPath[k]);

printf(cBuf);

}

cout<<endl;

bestLength = m\_cAntAry[j].m\_dbPathLength;

m\_cAntAry[j].CalPathLengthTest();

m\_cAntAry[j].CalPathTimeTest();

}

}

UpdateTrial();

}

}

int main()

{

//用当前时间点初始化随机种子，防止每次运行的结果都相同

time\_t tm;

time(&tm);

unsigned int nSeed = (unsigned int)tm;

srand(nSeed);

//开始搜索

CTsp tsp;

//g\_distanc

tsp.InitData(); //初始化

tsp.Search(); //开始搜索

//输出结果

printf("\nThe best tour is :\n");

char cBuf[128];

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

sprintf(cBuf, "%d ", tsp.m\_cBestAnt.m\_nPath[i]);

if (i % 20 == 0)

{

printf("\n");

}

printf(cBuf);

if(i>0)

{

cout<<tsp.m\_cBestAnt.m\_nTimeEachCost[tsp.m\_cBestAnt.m\_nPath[i]] <<" "<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<endl;

}

else

{

cout<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<" "<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<endl;

}

}

return 0;

}

9.2.3 问题三

对问题二中的蚁群算法代码进行改动，增加多进程互斥。

#include<iostream>

#include<math.h>

#include<time.h>

#include<string.h>

#include <stdio.h>

#include <iostream>

#include<windows.h>

#include<process.h>

#include <unistd.h>

#include <pthread.h>

#include <fstream>

#include <stdlib.h>

#include <cstdio>

#include <random>

#include<set>

using namespace std;

const double ALPHA = 1.0; //启发因子，信息素的重要程度

const double BETA = 2.0; //期望因子，景点间距离的重要程度

const double ROU = 0.5; //信息素残留参数

//0 3 5 1 2 4 6 7

const int N\_ANT\_COUNT = 100; //蚂蚁数量

const int N\_IT\_COUNT = 1000; //迭代次数

const int N\_CITY\_COUNT = 8; //景点数量

const double DBQ = 200.0; //总的信息素

const double DB\_MAX = 10e9; //一个标志数，10的9次方

int mutex[N\_ANT\_COUNT] = {0};//互斥量数组生成

//即同一个地点不能进入，先设置为均可进入

double g\_Trial[N\_CITY\_COUNT][N\_CITY\_COUNT];

//两两景点间信息素，就是环境信息素

double g\_Distance[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{ 0, 300, 360, 210, 530, 475, 500, 690},

{ 300, 0, 380, 270, 230, 285, 200, 390},

{ 360, 380, 0, 510, 230, 665, 490, 680},

{ 210, 270, 510, 0, 470, 265, 450, 640},

{ 530, 230, 230, 470, 0, 515, 260, 450},

{ 475, 285, 665, 265, 515, 0, 460, 650},

{ 500, 200, 490, 450, 260, 460, 0, 190},

{ 690, 390, 680, 640, 450, 650, 190, 0}

};//两两景点间距离

double g\_Walk[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{0,9,10.8,6.3,15.9,14.25,15,20.7},

{9,0,11.4,8.1,6.9,8.55,6,11.7},

{10.8,11.4,0,15.3,6.9,19.95,14.7,20.4},

{6.3,8.1,15.3,0,14.1,7.95,13.5,19.2},

{15.9,6.9,6.9,14.1,0,15.45,7.8,13.5},

{14.25,8.55,19.95,7.95,15.45,0,13.8,19.5},

{15,6,14.7,13.5,7.8,13.8,0,5.7},

{20.7,11.7,20.4,19.2,13.5,19.5,5.7,0}

};//两两景点间步行时间（v=2km/h）

double bestLength = DB\_MAX;

//检查节点是否都被访问

bool check(int\* group)

{

int sizeofgroup = sizeof(group);

int flag = 0;

for(int i=0; i<sizeofgroup-1; i++)

{

if(group[i]==0)

{

flag+=1;

}

}

if(flag == sizeofgroup-1)return true;

else return false;

}

//返回指定范围内的随机整数

int rnd(int nLow, int nUpper)

{

return nLow + (nUpper - nLow) \* rand() / (RAND\_MAX + 1);

}

//返回指定范围内的随机浮点数

double rnd(double dbLow, double dbUpper)

{

double dbTemp = rand() / ((double)RAND\_MAX + 1.0);

return dbLow + dbTemp \* (dbUpper - dbLow);

}

//返回浮点数四舍五入取整后的浮点数

double ROUND(double dbA)

{

return(double)((int)(dbA+0.5));

}

random\_device rd;

mt19937 gen(rd());

//生成一个符合截断正态分布的随机数

double normal\_rdn(double lowwer, double upper, double v) //下限，上限，方差

{

double e = (lowwer + upper) / 2;

normal\_distribution<double> normal(e, v);

double ans = (double)normal(gen);

if(ans >= lowwer && ans <= upper) return ans;

else return (double)normal\_rdn(lowwer, upper, v);

}

//森林小剧场，dt = 30，且开放时间为整除30的数

//因为到湿地商业街还需时间，故t <= 240

//编号 3

bool theater(int t)

{

if(t % 30 || t > 240)

return false;

return true;

}

//游客服务中心，10 <= dt <= 30，且t\_end <= 240

//编号 1

bool s\_center(int t, int dt)

{

if(t + dt > 240)

return false;

return true;

}

//delta time Array：在每个景点停留时间上下限

double dtAry[N\_CITY\_COUNT][2] = {

{0, 0},//0 source, ignore

{10, 30},//1

{20, 60},//2

{30, 30},//3,constant int

{30, 60},//4

{20, 60},//5

{30, 60},//6

{30, 30}//7 destination, ignore it in problem 2

};

//返回一个介于上限和下限之间的rnd

double randTime(int CITY\_NO)

{

int a = dtAry[CITY\_NO][0], b =dtAry[CITY\_NO][1];

return rnd(a, b);

}

//定义蚂蚁类

class CAnt

{

public:

CAnt(void);

~CAnt(void);

public:

int m\_nPath[N\_CITY\_COUNT]; //蚂蚁走的路径

double m\_dbPathLength; //蚂蚁走过的路径长度

double m\_dbPathTime;//蚂蚁用的时间

int m\_nAllowedCity[N\_CITY\_COUNT]; //没去过的景点

int m\_nCurCityNo; //当前所在景点编号

int m\_nMovedCityCount; //已经去过的景点数量

double m\_nTimeCost[N\_CITY\_COUNT];//每个节点进入的时间

double m\_nTimeEachCost[N\_CITY\_COUNT]; //= {0};//每个节点花的时间

public:

int ChooseNextCity(); //选择下一个景点

void Init();//初始化

void Move();//蚂蚁在景点间移动

void Search(); //搜索路径

void CalPathLength(); //计算蚂蚁走过的路径长度

void CalPathLengthTest();//测试用

void CalPathTime();//计算蚂蚁用的时间

void CalPathTimeTest();//test

//void Lanch();

};

//构造函数

CAnt::CAnt(void)

{

}

//析构函数

CAnt::~CAnt(void)

{

}

//初始化函数，蚂蚁搜索前调用

void CAnt::Init()

{

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m\_nAllowedCity[i] = 1; //设置全部景点为没有去过

m\_nPath[i] = 0; //蚂蚁走的路径全部设置为0

}

//蚂蚁走过的路径长度设置为0

m\_dbPathLength = 0.0;

//选择一个出发景点

m\_nCurCityNo = 0;

//m\_nCurCityNo = rnd(0,N\_CITY\_COUNT);

//把出发景点保存入路径数组中

m\_nPath[0] = m\_nCurCityNo;

//标识出发景点为已经去过了

m\_nAllowedCity[m\_nCurCityNo] = 0;

//每个节点所要花费的时间，初始点设为0

m\_nTimeEachCost[m\_nCurCityNo] = 0;

//已经去过的景点数量设置为1

m\_nMovedCityCount = 1;

//把出发时间定为0

m\_nTimeCost[m\_nCurCityNo] = 0;

}

//选择下一个景点

//返回值 为景点编号

int CAnt::ChooseNextCity()

{

int nSelectedCity=-1; //返回结果，先暂时把其设置为-1

//计算当前景点和没去过的景点之间的信息素总和

double dbTotal=0.0;

double prob[N\_CITY\_COUNT]; //保存各个景点被选中的概率

double temp\_enteringTime[N\_CITY\_COUNT] = {0};

double TimeHereCost = randTime(m\_nCurCityNo);

m\_nTimeEachCost[m\_nCurCityNo] = TimeHereCost;

for (int i = 0; i < N\_CITY\_COUNT;i++)

{

if(mutex[i] == 1)continue;

else mutex[i] == 1; //判断是否可以进入

temp\_enteringTime[i] = m\_nTimeCost[m\_nCurCityNo] + g\_Walk[m\_nCurCityNo][i] + TimeHereCost;

//对景点号进行确认，如果不符合条件直接概率调零

if(i == 1)

{

//我们需要临时时间

if(!s\_center(temp\_enteringTime[i],randTime(i)))

{

prob[i] = 0;

continue;

}

}

if(i == 3)

{

if(!theater(temp\_enteringTime[i]))

{

prob[i]=0;

continue;

}

}

if (m\_nAllowedCity[i] == 1) //景点没去过

{

prob[i]=pow(g\_Trial[m\_nCurCityNo][i],ALPHA)\*pow(1.0/g\_Distance[m\_nCurCityNo][i],BETA);

//该景点和当前景点间的信息素

dbTotal=dbTotal+prob[i]; //累加信息素，得到总和

}

else //如果景点去过了，则其被选中的概率值为0

{

prob[i]=0.01;

}

}

//进行轮盘选择

double dbTemp=0.0;

if (dbTotal> 0.0) //总的信息素值大于0

{

dbTemp=rnd(0.0,dbTotal); //取一个随机数

for (int i=0;i<N\_CITY\_COUNT;i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

dbTemp=dbTemp-prob[i]; //这个操作相当于转动轮盘

if (dbTemp < 0.0&& i < N\_CITY\_COUNT-1) //轮盘停止转动，记下景点编号，直接跳出循环

{

nSelectedCity=i;

//发现不是终点则可以选

break;

}

else if(dbTemp < 0.0&& i == N\_CITY\_COUNT-1)

{

if(check(m\_nAllowedCity)){nSelectedCity=i;break;}

//确认是否全部都选过了

else break;

}

}

}

}

if(nSelectedCity == -1)

{

for (int i=0;i<N\_CITY\_COUNT;i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

//对i进行判断

nSelectedCity=i;

break;

}

}

}

mutex[m\_nCurCityNo] = 0;//释放已走过的节点

//返回结果，就是景点的编号

//记录进入时间

m\_nTimeCost[nSelectedCity] = temp\_enteringTime[nSelectedCity];

return nSelectedCity;

}

//蚂蚁在景点间移动

void CAnt::Move()

{

int nCityNo=ChooseNextCity(); //选择下一个景点

m\_nPath[m\_nMovedCityCount]=nCityNo; //保存蚂蚁走的路

m\_nAllowedCity[nCityNo]=0;//把这个景点设置成已经去过了

m\_nCurCityNo=nCityNo; //改变当前所在景点为选择的景点

m\_nMovedCityCount++; //已经去过的景点数量加1

}

//蚂蚁进行搜索一次

void CAnt::Search()

{

Init();//蚂蚁搜索前，先初始化

//如果蚂蚁去过的景点数量小于景点数量，就继续移动

while(m\_nMovedCityCount < N\_CITY\_COUNT)

{

Move();

}

//完成搜索后计算走过的路径长度

CalPathLength();

CalPathTime();

//printf("%.0lf\n", m\_dbPathLength);

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLength()

{

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for(int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i-1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

}

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLengthTest()

{

//

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for(int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i-1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

cout<<n<<" "<<m<<" "<<" "<<g\_Distance[n][m]<<" "<<m\_dbPathLength<<endl;

}

}

//计算蚂蚁游玩的时间

void CAnt::CalPathTime()

{

m\_dbPathTime = 0.0; //先置0

int m = 0;

int n = 0;

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m = m\_nTimeEachCost[i];

m\_dbPathTime = m\_dbPathTime + m;

}

//计算蚂蚁游玩的时间

void CAnt::CalPathTimeTest()

{

m\_dbPathTime = 0.0; //先置0

int m = 0;

int n = 0;

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m = m\_nTimeEachCost[i];

m\_dbPathTime = m\_dbPathTime + m;

}

cout<<m\_dbPathTime<<endl;

}

//tsp类

class CTsp

{

public:

CTsp(void);

~CTsp(void);

public:

CAnt m\_cAntAry[N\_ANT\_COUNT][3];

public:

//初始化数据

void InitData();

//开始搜索

static void Search();

//更新环境信息素

void UpdateTrial();

void lanch();

};

//构造函数

CTsp::CTsp(void)

{

}

CTsp::~CTsp(void)

{

}

//初始化数据

void CTsp::InitData()

{

//先把最优蚂蚁的路径长度设置成一个很大的值

m\_cBestAnt.m\_dbPathLength = DB\_MAX;

//计算两两景点间距离

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

for(int j = 0; j<N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = 1.0;

}

}

}

//更新环境信息素

void CTsp::UpdateTrial()

{

//临时数组，保存各只蚂蚁在两两景点间新留下的信息素

double dbTempAry[N\_CITY\_COUNT][N\_CITY\_COUNT];

memset(dbTempAry, 0, sizeof(dbTempAry)); //先全部设置为0

//计算新增加的信息素,保存到临时数组里

int m = 0;

int n = 0;

for(int i = 0; i < N\_ANT\_COUNT; i++) //计算每只蚂蚁留下的信息素

{

for (int j = 1; j < N\_CITY\_COUNT; j++)

{

m = m\_cAntAry[i].m\_nPath[j];

n = m\_cAntAry[i].m\_nPath[j - 1];

dbTempAry[n][m] = dbTempAry[n][m]+DBQ/m\_cAntAry[i].m\_dbPathLength;

//要对时间点进行惩罚

dbTempAry[m][n] = dbTempAry[n][m];

}

//最后景点和开始景点之间的信息素

n = m\_cAntAry[i].m\_nPath[0];

dbTempAry[n][m] = dbTempAry[n][m]+ DBQ/m\_cAntAry[i].m\_dbPathLength;

dbTempAry[m][n] = dbTempAry[n][m];

}

//更新环境信息素

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

for (int j = 0; j < N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = g\_Trial[i][j] \* ROU + dbTempAry[i][j]; //最新的环境信息素 =留存的信息素 + 新留下的信息素

}

}

}

void CTsp::Search()

{

char cBuf[256]; //打印信息用

char dBuf[256];

char eBuf[256];

//现在我们有三个信息要打印了

pthread\_t \*tid1;

pthread\_t \*tid2;

pthread\_t \*tid3;

//多线程指针

//在迭代次数内进行循环

for(int i = 0; i < N\_IT\_COUNT; i++)

{

//每只蚂蚁搜索一遍

for (int j = 0; j < N\_ANT\_COUNT;j++)

{

err1 = phread\_create(&tid1, NULL, m\_cAntAry[j][0].Search, NULL);

err2 = phread\_create(&tid2, NULL, m\_cAntAry[j][1].Search, NULL);

err3 = phread\_create(&tid3, NULL, m\_cAntAry[j][2].Search, NULL);

if(err1 != 0)

{

printf("can'tcreate thread: %s/n", strerror(err));

break;

}

if(err2 != 0)

{

printf("can'tcreate thread: %s/n", strerror(err));

break;

}

if(err3 != 0)

{

printf("can'tcreate thread: %s/n", strerror(err));

break;

}

//这里要让多个团体同时进行

//cout<<m\_cAntAry[j].m\_dbPathLength<<endl;

//查看变量，打断点

}

//保存最佳结果

for (int j = 0; j < N\_ANT\_COUNT; j++)

{

if(m\_cAntAry[j].m\_nPath[N\_CITY\_COUNT-1] != 7

||m\_cAntAry[j].m\_nTimeCost[N\_CITY\_COUNT-1] >= 300)continue;

int m\_dWholeTime = (330-(m\_cAntAry[j][0].m\_dbPathLength)\*30) +(330-(m\_cAntAry[j][1].m\_dbPathLength)\*30) + (330-(m\_cAntAry[j][2].m\_dbPathLength)\*30);

int m\_dBestTime = (330-(m\_BestAnt[j][0].m\_dbPathLength)\*30) + (330-(m\_BestAnt[j][1].m\_dbPathLength)\*30) + (330-(m\_BestAnt[j][2].m\_dbPathLength)\*30);

if (m\_dWholeTime > m\_dBestTime)

{

m\_cBestAnt = &m\_cAntAry[j];

//指向头地址

for(int k = 0; k<N\_CITY\_COUNT;k++)

{

sprintf(cBuf, "%d ", m\_cAntAry[j][0].m\_nPath[k]);

sprintf(dBuf, "%d ", m\_cAntAry[j][1].m\_nPath[k]);

sprintf(eBuf, "%d ", m\_cAntAry[j][0].m\_nPath[k]);

printf(cBuf);

printf(dBuf);

printf(eBuf);

}

}

}

//更新环境信息素

UpdateTrial();

}

}

DWORD WINAPI TspThread(LPVOID lpParameter)

{

//用当前时间点初始化随机种子，防止每次运行的结果都相同

time\_t tm;

time(&tm);

unsigned int nSeed = (unsigned int)tm;

srand(nSeed);

//开始搜索

CTsp tsp;

//g\_distanc

tsp.InitData(); //初始化

tsp.Search(); //开始搜索

//输出结果

printf("\nThe best tour is :\n");

char cBuf[128];

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

sprintf(cBuf, "%d ", tsp.m\_cBestAnt.m\_nPath[i]);

if (i % 20 == 0)

{

printf("\n");

}

printf(cBuf);

if(i>0)

{

cout<<tsp.m\_cBestAnt.m\_nTimeEachCost[tsp.m\_cBestAnt.m\_nPath[i]] <<" "<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<endl;

}

else

{

cout<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<" "<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<endl;

}

}

return 0;

}

int main()

{

HANDLE Thread1;

Thread1 = CreateThread(NULL, 0, TspThread, NULL, 0, NULL);

CloseHandle(Thread1);

while(1);

return 0;

}

9.2.4 问题四

对问题三中的蚁群算法代码进行改动，增加扰动函数。

#include<iostream>

#include<math.h>

#include<time.h>

#include<string.h>

#include <stdio.h>

#include <iostream>

#include<windows.h>

#include<process.h>

#include <unistd.h>

#include <pthread.h>

#include <fstream>

#include <stdlib.h>

#include <cstdio>

#include <random>

#include<set>

using namespace std;

const double ALPHA = 1.0; //启发因子，信息素的重要程度

const double BETA = 2.0; //期望因子，景点间距离的重要程度

const double ROU = 0.5; //信息素残留参数

//0 3 5 1 2 4 6 7

const int N\_ANT\_COUNT = 100; //蚂蚁数量

const int N\_IT\_COUNT = 1000; //迭代次数

const int N\_CITY\_COUNT = 8; //景点数量

const double DBQ = 200.0; //总的信息素

const double DB\_MAX = 10e9; //一个标志数，10的9次方

int mutex[N\_ANT\_COUNT] = {0};//互斥量数组生成

//即同一个地点不能进入，先设置为均可进入

double g\_Trial[N\_CITY\_COUNT][N\_CITY\_COUNT];

//两两景点间信息素

double g\_Distance[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{ 0, 300, 360, 210, 530, 475, 500, 690},

{ 300, 0, 380, 270, 230, 285, 200, 390},

{ 360, 380, 0, 510, 230, 665, 490, 680},

{ 210, 270, 510, 0, 470, 265, 450, 640},

{ 530, 230, 230, 470, 0, 515, 260, 450},

{ 475, 285, 665, 265, 515, 0, 460, 650},

{ 500, 200, 490, 450, 260, 460, 0, 190},

{ 690, 390, 680, 640, 450, 650, 190, 0}

};//两两景点间距离

/\*

double g\_Walk[N\_CITY\_COUNT][N\_CITY\_COUNT] = {

{0,9,10.8,6.3,15.9,14.25,15,20.7},

{9,0,11.4,8.1,6.9,8.55,6,11.7},

{10.8,11.4,0,15.3,6.9,19.95,14.7,20.4},

{6.3,8.1,15.3,0,14.1,7.95,13.5,19.2},

{15.9,6.9,6.9,14.1,0,15.45,7.8,13.5},

{14.25,8.55,19.95,7.95,15.45,0,13.8,19.5},

{15,6,14.7,13.5,7.8,13.8,0,5.7},

{20.7,11.7,20.4,19.2,13.5,19.5,5.7,0}

};

\*/

//由于速度变化，路径上的时间不能用打表实现

double bestLength = DB\_MAX;

const double SPEED\_VARIANCE = 1;//速度随机分布的方差

//检查节点是否都被访问

bool check(int\* group)

{

int sizeofgroup = sizeof(group);

int flag = 0;

for(int i=0; i<sizeofgroup-1; i++)

{

if(group[i]==0)

{

flag+=1;

}

}

if(flag == sizeofgroup-1)return true;

else return false;

}

//返回指定范围内的随机整数

int rnd(int nLow, int nUpper)

{

return nLow + (nUpper - nLow) \* rand() / (RAND\_MAX + 1);

}

//返回指定范围内的随机浮点数

double rnd(double dbLow, double dbUpper)

{

double dbTemp = rand() / ((double)RAND\_MAX + 1.0);

return dbLow + dbTemp \* (dbUpper - dbLow);

}

//返回浮点数四舍五入取整后的浮点数

double ROUND(double dbA)

{

return(double)((int)(dbA+0.5));

}

random\_device rd;

mt19937 gen(rd());

//生成一个符合截断正态分布的随机数

double normal\_rdn(double lowwer, double upper, double v) //下限，上限，方差

{

double e = (lowwer + upper) / 2;

normal\_distribution<double> normal(e, v);

double ans = (double)normal(gen);

if(ans >= lowwer && ans <= upper) return ans;

else return (double)normal\_rdn(lowwer, upper, v);

}

const double PI = 3.14159;

int xx = 0; //这里定义x一个全局变量并且初始付值0，这个的功用将会在子函数UNIFORM中得以体现

void UNIFORM(double \*p)

{

int i, a;

double f;

for (i = 0; i<2; i++, xx = xx + 689)

{

a = rand() + xx; //加上689是因为系统产生随机数的更换频率远远不及程序调用函数的时间

a = a%1000;

f = (double)a;

f = f / 1000.0;

\*p = f;

p++;

}

}

//这个函数产生的正态分布随机数更为neat

//用于产生速度

double Speedproducer(double lowwer, double upper, double D)//D为方差

{

int i, j;

double A, B, C, E, r;

double uni[2];

double \*p;

srand((unsigned)time(NULL)); //随机数种子采用系统时钟

UNIFORM(&uni[0]); //调用UNIFORM函数产生2个均匀分布的随机数并存入数组nui[2]

A = sqrt((-2)\*log(uni[0]));

B = 2 \* PI\*uni[1];

C = A\*cos(B);

E = (lowwer + upper) / 2;//数学期望为均值，换言之，其均值可以调控

r = E + C \* D; //E,D分别是期望和方差

if(r >= lowwer && r <= upper) return r;

else return BoxMuller(lowwer, upper, D);

}

double speed()

{

return Speedproducer(1, 3, SPEED\_VARIANCE)

//返回符合期望的时间

}

//森林小剧场，dt = 30，且开放时间为整除30的数

//因为到湿地商业街还需时间，故t <= 240

//编号 3

bool theater(int t)

{

if(t % 30 || t > 240)

return false;

return true;

}

//游客服务中心，10 <= dt <= 30，且t\_end <= 240

//编号 1

bool s\_center(int t, int dt)

{

if(t + dt > 240)

return false;

return true;

}

//delta time Array：在每个景点停留时间上下限

double dtAry[N\_CITY\_COUNT][2] = {

{0, 0},//0 source, ignore

{10, 30},//1

{20, 60},//2

{30, 30},//3,constant int

{30, 60},//4

{20, 60},//5

{30, 60},//6

{30, 30}//7 destination, ignore it in problem 2

};

//返回一个介于上限和下限之间的rnd

double randTime(int CITY\_NO)

{

int a = dtAry[CITY\_NO][0], b =dtAry[CITY\_NO][1];

return rnd(a, b);

}

//定义蚂蚁类

class CAnt

{

public:

CAnt(void);

~CAnt(void);

public:

int m\_nPath[N\_CITY\_COUNT]; //蚂蚁走的路径

double m\_dbPathLength; //蚂蚁走过的路径长度

double m\_dbPathTime;//蚂蚁用的时间

int m\_nAllowedCity[N\_CITY\_COUNT]; //没去过的景点

int m\_nCurCityNo; //当前所在景点编号

int m\_nMovedCityCount; //已经去过的景点数量

double m\_nTimeCost[N\_CITY\_COUNT];//每个节点进入的时间

double m\_nTimeEachCost[N\_CITY\_COUNT]; //= {0};//每个节点花的时间

public:

int ChooseNextCity(); //选择下一个景点

void Init();//初始化

void Move();//蚂蚁在景点间移动

void Search(); //搜索路径

void CalPathLength(); //计算蚂蚁走过的路径长度

void CalPathLengthTest();//测试用

void CalPathTime();//计算蚂蚁用的时间

void CalPathTimeTest();//test

//void Lanch();

};

//构造函数

CAnt::CAnt(void)

{

}

//析构函数

CAnt::~CAnt(void)

{

}

//初始化函数，蚂蚁搜索前调用

void CAnt::Init()

{

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m\_nAllowedCity[i] = 1; //设置全部景点为没有去过

m\_nPath[i] = 0; //蚂蚁走的路径全部设置为0

}

//蚂蚁走过的路径长度设置为0

m\_dbPathLength = 0.0;

//选择一个出发景点

m\_nCurCityNo = 0;

//m\_nCurCityNo = rnd(0,N\_CITY\_COUNT);

//把出发景点保存入路径数组中

m\_nPath[0] = m\_nCurCityNo;

//标识出发景点为已经去过了

m\_nAllowedCity[m\_nCurCityNo] = 0;

//每个节点所要花费的时间，初始点设为0

m\_nTimeEachCost[m\_nCurCityNo] = 0;

//已经去过的景点数量设置为1

m\_nMovedCityCount = 1;

//把出发时间定为0

m\_nTimeCost[m\_nCurCityNo] = 0;

}

//选择下一个景点

//返回值 为景点编号

int CAnt::ChooseNextCity()

{

int nSelectedCity=-1; //返回结果，先暂时把其设置为-1

//计算当前景点和没去过的景点之间的信息素总和

double dbTotal=0.0;

double prob[N\_CITY\_COUNT]; //保存各个景点被选中的概率

double temp\_enteringTime[N\_CITY\_COUNT] = {0};

double TimeHereCost = randTime(m\_nCurCityNo);

m\_nTimeEachCost[m\_nCurCityNo] = TimeHereCost;

for (int i = 0; i < N\_CITY\_COUNT;i++)

{

if(mutex[i] == 1)continue;

else mutex[i] == 1; //判断是否可以进入

temp\_enteringTime[i] = m\_nTimeCost[m\_nCurCityNo] + g\_Distance[m\_nCurCityNo][i]/speed() + TimeHereCost;

//这里路上的时间不再使用g\_Walk确认，而是用路程除以时间来表示

//对景点号进行确认，如果不符合条件直接概率调零

if(i == 1)

{

//我们需要临时时间

if(!s\_center(temp\_enteringTime[i],randTime(i)))

{

prob[i] = 0;

continue;

}

}

if(i == 3)

{

if(!theater(temp\_enteringTime[i]))

{

prob[i]=0;

continue;

}

}

if (m\_nAllowedCity[i] == 1) //景点没去过

{

prob[i]=pow(g\_Trial[m\_nCurCityNo][i],ALPHA)\*pow(1.0/g\_Distance[m\_nCurCityNo][i],BETA);

//该景点和当前景点间的信息素

dbTotal=dbTotal+prob[i]; //累加信息素，得到总和

}

else //如果景点去过了，则其被选中的概率值为0

{

prob[i]=0.01;

}

}

//进行轮盘选择

double dbTemp=0.0;

if (dbTotal> 0.0) //总的信息素值大于0

{

dbTemp=rnd(0.0,dbTotal); //取一个随机数

for (int i=0;i<N\_CITY\_COUNT;i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

dbTemp=dbTemp-prob[i]; //这个操作相当于转动轮盘

if (dbTemp < 0.0&& i < N\_CITY\_COUNT-1) //轮盘停止转动，记下景点编号，直接跳出循环

{

nSelectedCity=i;

//发现不是终点则可以选

break;

}

else if(dbTemp < 0.0&& i == N\_CITY\_COUNT-1)

{

if(check(m\_nAllowedCity)){nSelectedCity=i;break;}

//确认是否全部都选过了

else break;

}

}

}

}

//如果景点间的信息素非常小 ( 小到比double能够表示的最小的数字还要小 )

//那么由于浮点运算的误差原因，上面计算的概率总和可能为0

//会出现经过上述操作，没有景点被选择出来

//出现这种情况，就把第一个没去过的景点作为返回结果

if(nSelectedCity == -1)

{

for (int i=0;i<N\_CITY\_COUNT;i++)

{

if (m\_nAllowedCity[i] == 1) //景点没去过

{

//对i进行判断

nSelectedCity=i;

break;

}

}

}

mutex[m\_nCurCityNo] = 0;//释放已走过的节点

//返回结果，就是景点的编号

//记录进入时间

m\_nTimeCost[nSelectedCity] = temp\_enteringTime[nSelectedCity];

return nSelectedCity;

}

//蚂蚁在景点间移动

void CAnt::Move()

{

int nCityNo=ChooseNextCity(); //选择下一个景点

m\_nPath[m\_nMovedCityCount]=nCityNo; //保存蚂蚁走的路

m\_nAllowedCity[nCityNo]=0;//把这个景点设置成已经去过了

m\_nCurCityNo=nCityNo; //改变当前所在景点为选择的景点

m\_nMovedCityCount++; //已经去过的景点数量加1

}

//蚂蚁进行搜索一次

void CAnt::Search()

{

Init();//蚂蚁搜索前，先初始化

//如果蚂蚁去过的景点数量小于景点数量，就继续移动

while(m\_nMovedCityCount < N\_CITY\_COUNT)

{

Move();

}

//完成搜索后计算走过的路径长度

CalPathLength();

CalPathTime();

//printf("%.0lf\n", m\_dbPathLength);

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLength()

{

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for(int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i-1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

}

//加上从最后景点返回出发景点的距离

/\*

n = m\_nPath[0];

m\_dbPathLength = m\_dbPathLength + g\_Distance[m][n];

\*/

}

//计算蚂蚁走过的路径长度

void CAnt::CalPathLengthTest()

{

//

m\_dbPathLength = 0.0; //先把路径长度置0

int m = 0;

int n = 0;

for(int i = 1; i < N\_CITY\_COUNT; i++)

{

m = m\_nPath[i];

n = m\_nPath[i-1];

m\_dbPathLength = m\_dbPathLength + g\_Distance[n][m];

cout<<n<<" "<<m<<" "<<" "<<g\_Distance[n][m]<<" "<<m\_dbPathLength<<endl;

}

}

//计算蚂蚁游玩的时间

void CAnt::CalPathTime()

{

m\_dbPathTime = 0.0; //先置0

int m = 0;

int n = 0;

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m = m\_nTimeEachCost[i];

m\_dbPathTime = m\_dbPathTime + m;

}

}

//计算蚂蚁游玩的时间

void CAnt::CalPathTimeTest()

{

m\_dbPathTime = 0.0; //先置0

int m = 0;

int n = 0;

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

m = m\_nTimeEachCost[i];

m\_dbPathTime = m\_dbPathTime + m;

}

cout<<m\_dbPathTime<<endl;

}

//tsp类

class CTsp

{

public:

CTsp(void);

~CTsp(void);

public:

CAnt m\_cAntAry[N\_ANT\_COUNT][3]; //蚂蚁数组

CAnt\* m\_cBestAnt; //定义一个蚂蚁变量，用来保存搜索过程中的最优结果

//该蚂蚁不参与搜索，只是用来保存最优结果

//static pthread\_mutex\_t spot\_mutex[N\_CITY\_COUNT];

public:

//初始化数据

void InitData();

//开始搜索

static void Search();

//更新环境信息素

void UpdateTrial();

void lanch();

};

//构造函数

CTsp::CTsp(void)

{

}

CTsp::~CTsp(void)

{

}

//初始化数据

void CTsp::InitData()

{

//先把最优蚂蚁的路径长度设置成一个很大的值

m\_cBestAnt.m\_dbPathLength = DB\_MAX;

//计算两两景点间距离

//这个是一开始就写好的

//初始化环境信息素，先把景点间的信息素设置成一样

//这里设置成1.0，设置成多少对结果影响不是太大，对算法收敛速度有些影响

for(int i = 0; i < N\_CITY\_COUNT; i++)

{

for(int j = 0; j<N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = 1.0;

}

}

}

//更新环境信息素

void CTsp::UpdateTrial()

{

//临时数组，保存各只蚂蚁在两两景点间新留下的信息素

double dbTempAry[N\_CITY\_COUNT][N\_CITY\_COUNT];

memset(dbTempAry, 0, sizeof(dbTempAry)); //先全部设置为0

//计算新增加的信息素,保存到临时数组里

int m = 0;

int n = 0;

for(int i = 0; i < N\_ANT\_COUNT; i++) //计算每只蚂蚁留下的信息素

{

for (int j = 1; j < N\_CITY\_COUNT; j++)

{

m = m\_cAntAry[i].m\_nPath[j];

n = m\_cAntAry[i].m\_nPath[j - 1];

dbTempAry[n][m] = dbTempAry[n][m]+DBQ/m\_cAntAry[i].m\_dbPathLength;

//要对时间点进行惩罚

dbTempAry[m][n] = dbTempAry[n][m];

}

//最后景点和开始景点之间的信息素

n = m\_cAntAry[i].m\_nPath[0];

dbTempAry[n][m] = dbTempAry[n][m]+ DBQ/m\_cAntAry[i].m\_dbPathLength;

dbTempAry[m][n] = dbTempAry[n][m];

}

//更新环境信息素

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

for (int j = 0; j < N\_CITY\_COUNT; j++)

{

g\_Trial[i][j] = g\_Trial[i][j] \* ROU + dbTempAry[i][j]; //最新的环境信息素 =留存的信息素 + 新留下的信息素

}

}

}

void CTsp::Search()

{

char cBuf[256]; //打印信息用

char dBuf[256];

char eBuf[256];

//现在我们有三个信息要打印了

pthread\_t \*tid1;

pthread\_t \*tid2;

pthread\_t \*tid3;

//多线程指针

//在迭代次数内进行循环

for(int i = 0; i < N\_IT\_COUNT; i++)

{

//每只蚂蚁搜索一遍

for (int j = 0; j < N\_ANT\_COUNT;j++)

{

err1 = phread\_create(&tid1, NULL, m\_cAntAry[j][0].Search, NULL);

err2 = phread\_create(&tid2, NULL, m\_cAntAry[j][1].Search, NULL);

err3 = phread\_create(&tid3, NULL, m\_cAntAry[j][2].Search, NULL);

if(err1 != 0)

{

printf("can'tcreate thread: %s/n", strerror(err));

break;

}

if(err2 != 0)

{

printf("can'tcreate thread: %s/n", strerror(err));

break;

}

if(err3 != 0)

{

printf("can'tcreate thread: %s/n", strerror(err));

break;

}

//这里要让多个团体同时进行

//cout<<m\_cAntAry[j].m\_dbPathLength<<endl;

//查看变量，打断点

}

//保存最佳结果

for (int j = 0; j < N\_ANT\_COUNT; j++)

{

if(m\_cAntAry[j].m\_nPath[N\_CITY\_COUNT-1] != 7

||m\_cAntAry[j].m\_nTimeCost[N\_CITY\_COUNT-1] >= 300)continue;

int m\_dWholeTime = (330-(m\_cAntAry[j][0].m\_dbPathLength)\*30) +(330-(m\_cAntAry[j][1].m\_dbPathLength)\*30) + (330-(m\_cAntAry[j][2].m\_dbPathLength)\*30);

int m\_dBestTime = (330-(m\_BestAnt[j][0].m\_dbPathLength)\*30) + (330-(m\_BestAnt[j][1].m\_dbPathLength)\*30) + (330-(m\_BestAnt[j][2].m\_dbPathLength)\*30);

if (m\_dWholeTime > m\_dBestTime)

{

m\_cBestAnt = &m\_cAntAry[j];

//指向头地址

for(int k = 0; k<N\_CITY\_COUNT;k++)

{

sprintf(cBuf, "%d ", m\_cAntAry[j][0].m\_nPath[k]);

sprintf(dBuf, "%d ", m\_cAntAry[j][1].m\_nPath[k]);

sprintf(eBuf, "%d ", m\_cAntAry[j][0].m\_nPath[k]);

printf(cBuf);

printf(dBuf);

printf(eBuf);

}

}

}

}

//更新环境信息素

UpdateTrial();

}

}

DWORD WINAPI TspThread(LPVOID lpParameter)

{

//用当前时间点初始化随机种子，防止每次运行的结果都相同

time\_t tm;

time(&tm);

unsigned int nSeed = (unsigned int)tm;

srand(nSeed);

//开始搜索

CTsp tsp;

//g\_distanc

tsp.InitData(); //初始化

tsp.Search(); //开始搜索

//输出结果

printf("\nThe best tour is :\n");

char cBuf[128];

for (int i = 0; i < N\_CITY\_COUNT; i++)

{

sprintf(cBuf, "%d ", tsp.m\_cBestAnt.m\_nPath[i]);

if (i % 20 == 0)

{

printf("\n");

}

printf(cBuf);

if(i>0)

{

cout<<tsp.m\_cBestAnt.m\_nTimeEachCost[tsp.m\_cBestAnt.m\_nPath[i]] <<" "<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<endl;

}

else

{

cout<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<" "<<tsp.m\_cBestAnt.m\_nTimeCost[tsp.m\_cBestAnt.m\_nPath[i]]<<endl;

}

}

return 0;

}

int main()

{

HANDLE Thread1;

Thread1 = CreateThread(NULL, 0, TspThread, NULL, 0, NULL);

//此处为单线程

//为改为多线程对返回数据进行再次清洗提供方便

CloseHandle(Thread1);

while(1);

return 0;

}