算法 LAB4 实验报告

姓名：侯家珍　　日期：2019/11/22
学号：517030910253　　分数：

1. 简介/介绍/引言

利用**模拟退火算法**解决 TSP 旅行商问题。

1) 实验内容

本次实验内容为采用所学的模拟退火算法求解对称和非对称 TSP 问题。

2) 实验要求

• 本次实验有两组输入，输入格式不同，距离计算方式不同。

• 输入 1:输入 1 同 LAB3,即依然采用上次 LAB3 的 data 文件，该文件每行分三列，分别为城市编号、地理 x 坐标、地理 Y 坐标。该输入是一一个对称的 TSP 问题，即边 02 和边 2U1 长度相等，即图是一个无向图。需注意城市坐标比较特殊，共有 48 个城市，对于两个城市 i 和 j,其距离采用下述方法计算:

```
xd = x[i] - x[j];
yd = y[i] - y[j];
rij = sqrt( (xd*xd + yd*yd) / 10.0 );
tij = nint( rij ); /* nint(x) = (int) (x+0.5); */
if (tij<rij) dij = tij + 1;
else dij = tij;
```

•输入 2:输入 2 请采用 lab4.dat,这是一个共 358"城市”的非对称 TSP 问题，对应的图是有向图，边 0102 和边 V2U1 长度可能不相等。文件 lab4.dat 给出了矩阵形式的距离数据，共 358 行 358 列，数字之间以水平制表符 tab 隔开，矩阵的第 i 行 j 列的数值表示的是城市 i 和 j 之间的距离(如果数值为 0,就认为距离/权重为 0)。

• 由于两个输入文件的数据格式不同，程序可以分开编写，或者用选项控制计算输入文件 1 还是输入文件 2。城市的数目可能对读取数据文件有用，可以将其作为参数由控制台输入，或者直接写死在程序中亦可，这个不重要。

• 请为程序的主计算代码加入高精度计时函数，记录程序的运行时间。

• 撰写实验报告时，请加入相应的讨论环节(重要，请认真写)，讨论动态规划、近似算法、本节算法解决 TSP 问题时的各自特点。因为我们甚至采用了相同的数据文件。

2. 基本原理/实现方法

(1) 实验分析

模拟退火算法是爬山贪心算法的改进，以一定的概率接受较差的解从而可以跳出局部最优的值。其物理退火过程由以下三部分组成：

- ✓ 加温过程
- ✓ 等温过程
- ✓ 冷却过程

其中加温过程对应算法设定的初始温度，在我的算法中我设置为了 200000 摄氏度，等温过程对应算法的 Metropolis 抽样过程，冷却过程对应控制参数的下降。这里能量的变化就是目标函数，要得到的最优解就是能量最低状态。

Metropolis 准则是 SA 算法收敛于全局最优解的关键所在，Metropolis 准则以一定的概率接受恶化解，这样就使得算法可以跳离局部最优解的陷阱。

(2) 算法分析

模拟退火算法解决 TSP 问题实现的步骤如下：

1）初始化：取温度 T0 足够大，令 T = T0，取一条任意的城市路径 S1，确定每个 T 时的迭代次数，即 Metropolis 链长 L。

2）对当前温度 T 和 k∈[1,L],重复步骤 3 到 6。

3）对当前解 S1 随机产生一个扰动(这里采用简单的随机交换两个城市的方法)得到一个新解 S2.

4）计算 S2 相对于 S1 的增量 df = f（S2） - f（S1），这里的 f（）函数即为路径的长度。

5）若 df < 0,接受 S2 作为新的当前路径，即 S1 = S2；否则 S2 的接受概率为 exp（-df/T）,即随机产生（0,1）上的均匀分布的随机数 rand，若 exp（-df/T）>rand,则接受 S2 作为新的当前解，S1 = S2；否则保留当前解。

6）如果满足最终的终止条件，Stop，则输出当前解 S1 作为最优解，结束程

序。我的算法中的终止条件 Stop 是设定结束温度。否则按衰减函数衰减 T 后返回 2）

(3) 伪代码分析

SIMULATED-ANNEALING()

设定初温 T = T0，终止温度 T_stop，随机产生初始状态 s = s0，令 k = 0

while(T>T_stop) do

for i <-- 1 to L //迭代次数

sj = create_new(s) //产生新状态 sj

if min{1, exp(-(Csj-Cs)/ Tk)} ≥ random[0, 1]

then

s = sj

Tk+1 = update(tk) //降温

return 算法搜索结果

(4) 核心代码分析与展示

第一种输入的距离函数：

```
void distance()//初始化距离矩阵
{
    int i, j,t=0, m, n;
    int x1, y1, x2, y2;
    FILE* fp;
    fp = fopen("lab3.dat", "r");

    while (!feof(fp)) {
        fscanf(fp, "%d", &tmp[t]);
        t++;
    }
    for (m = 0; m < 48; m++) {
        i = tmp[m * 3];
        x1 = tmp[m * 3 + 1];
        y1 = tmp[m * 3 + 2];
        for (n = 0; n < 48; n++) {
            j = tmp[n * 3];
            x2 = tmp[n * 3 + 1];
            y2 = tmp[n * 3 + 2];
            int xd = x1 - x2;
            int yd = y1 - y2;
```

```
            double rj = sqrt(((xd * xd) + (yd * yd)) / 10.0);
            int  tj = (int)(rj + 0.5);
            if (tj < rj)
                citymap[i][j] = tj + 1;
            else
                citymap[i][j] = tj;
        }
    }
    for (int k = 0; k < 49; k++)  citymap[k][k] = 0;

    fclose(fp);
}
```

第二种输入的距离函数：

```
void distance()//初始化距离矩阵
{
    int i, j;
    FILE* fp;
    fp = fopen("lab4.dat", "r");
    for (i = 1; i < 359; i++) {
        for (j = 1; j < 359; j++) {
            fscanf(fp, "%d", &citymap[i][j]);
        }
    }
    fclose(fp);
    }
```

初始化函数，初始化一条新路径，这里直接按照 1,2,3……顺序进行初始化。

```
void init()
{
    for (int i = 0; i < 48; i++)
     city_list[i] = i + 1;  // 初始化一个解
 }
```

创建一条新路径，采用随机交换两个城市的顺序的方法。

```
void create_new()
{
    int pos1,pos2;
    double r1 = ((double)rand()) /(RAND_MAX + 1.0);
    double r2 = ((double)rand()) / (RAND_MAX + 1.0);
    pos1 = (int)(48 * r1);
    pos2 = (int)(48 * r2); //随机产生两个位置
    int temp = city_list[pos1];
    city_list[pos1] = city_list[pos2];
```

```
    city_list[pos2] = temp;    // 交换两个点
        }
```

然后是退火算法的核心部分，令初始温度为 200000，最终温度为 1e-10，退火系数为 0.99，每次的迭代次数为 1000。

```
while (T > T_end) // 当温度低于结束温度时，退火结束
        {
         for (int i = 0; i < L; i++)
             {
              memcpy(city_list_copy, city_list, 48 * sizeof(int)); // 复制数组
              create_new(); // 产生新解
              f1 = path_len(city_list_copy);//计算路径长度
              f2 = path_len(city_list);
              df = f2 - f1;
             // Metropolis 准则
                if (df >= 0)
                 {
                 r = ((double)rand()) / (RAND_MAX);
                 //产生 0 到 1 的随机数
                 if (exp(-df / T) <= r) // 保留原来的解
                     {
                     memcpy(city_list, city_list_copy, 48 * sizeof(int));
                               }
                           }
                     }
               T *= q; // 降温
          count++;
                 }
```

(5) 结果展示：

对于 48 个城市的 TSP 问题，采用退火近似算法的时候，结果如下（退火算法的结果不同次数下会有不同的最优解，在我跑的次数中差值 48 个城市的在 3000 左右，358 个城市在 300 左右）：

```
运行时间5.991588秒
采用模拟退火算法，初始温度T0=200000.00,降温系数q=0.99,每个温度迭代1000次,共降温3506次，得到的TSP最优路径为:
30->28->36->7->18->44->31->38->9->8->1->40->11->23->3->22->16->41->34->29->2->26->4->35->45->10->24->42->5->48->39->32->
21->25->14->13->47->20->12->15->33->46->6->37->19->27->17->43
最优路径长度为:11019.000000
```

而在 lab3 中我们采用近似算法时，得到的结果如下：

```
接下来会打印48个州的TSP问题近似算法得出的一条路线
该结果选定第一个城市作为开始节点
路径为：
1->8->9->38->31->44->18->7->28->6->30->37->19->27->17->43->36->40->15->12->11->23->14->25->13->39->32->48->5->29->2->42->10->24->26->4->35->45->34->41->47->21->33->20->46->22->3->16->1
运行时间0.011334秒

总的路径长度为13926
```

在 lab2 的动态规划中，因为 29 个城市的都没有跑出来，所以只能拿 20 个城市的进行比较，

```
输入你想读入的城市文件名（有7,20,29三种选择）：
20
运行时间1.828874秒
最短路径的长度是：7260
具体路径是：0->7->8->14->17->6->5->18->16->19->11->10->12->13->4->9->3->1->2->15->0
总子问题数为：94633623
非重叠子问题数为：13531439
```

我们可以看出在数据量较小的情况下，动态规划和退火算法都是可以在较短的时间内得到一个最优解的，但是当随着数据量的上升，即使用了状态压缩的方法，也会因为开启的 dp 数组过大而存在内存耗尽的结果，从而无法得到最短路径。所以动态规划的方法不适合数据很多的情况。不过当数据量比较大时，对时间有一定要求而有不需要得到一个最优解，我们可以用近似算法很快的得到一个近似解，从图里我们可以看到近似算法得到解的时间在毫秒量级，而退火算法却要花费几秒的时间。不过退火算法在输入数据很多的情况下求最优解有很好的应用，能够在可以接受的时间范围内得到一个最优解，如下图是 358 个城市的 TSP 问题的输出结果（初始温度这里变为 100000，也可以趋于稳定）：

```
运行时间31.409534秒
采用模拟退火算法，初始温度T0=100000.00,降温系数q=0.99,每个温度迭代1000次,共降温3437次，得到的TSP最优路径为:
86->269->42->44->13->52->132->144->220->197->83->320->51->341->275->230->310->19->215->340->277->343->286->157->31->111->100->49->29->274->240->22->301->155->272->246->267->152->112->102->162->153->271->330->80->241->34->322->261->254->140->107->211->185->65->94->249->137->325->41->89->218->202->169->222->200->262->331->258->18->14->4->323->190->192->309->76->213->299->104->188->206->5->126->345->289->256->174->338->263->333->324->337->270->37->30->239->205->183->273->234->79->64->329->252->154->66->229->72->315->130->186->210->180->219->53->21->122->121->128->99->68->147->38->327->326->351->307->303->201->255->319->227->179->266->280->339->17->170->349->110->335->191->298->133->105->212->181->166->344->314->160->346->194->78->352->305->235->268->116->336->318->232->149->283->302->251->61->165->321->96->226->295->292->287->125->6->88->84->264->139->114->148->81->1->358->97->265->342->216->355->278->46->26->39->156->98->209->195->74->63->120->189->118->250->158->199->73->308->242->207->193->243->208->159->150->182->115->198->221->47->171->167->131->350->196->77->2->70->71->328->106->176->173->109->311->35->10->136->40->36->43->27->113->312->224->163->134->313->332->143->353->260->357->217->135->67->15->69->23->172->25->223->317->225->60->45->259->168->3->58->253->293->236->294->11->28->214->141->32->178->297->142->238->356->9->16->82->146->296->95->175->57->7->285->248->103->92->91->306->138->123->245->354->288->145->124->233->334->237->54->93->276->281->127->12->247->161->85->33->228->316->291->164->8->304->282->151->300->284->348->129->203->50->20->59->244->187->108->290->257->279->62->204->75->117->90->231->347->56->87->48->24->55->101->119->184->177
最优路径长度为:1408.000000
```

综合这三次实验，可以看出三种算法各自的优缺点，我们可以在不同的情况下选择不同的算法，在对时间要求比较高但不需要太过精确的解的话，可以选取近似算法进行求解。如果数据量较小求优化解的话，动态规划是可以值得考虑的算法，当数据量特别

大的时候，退火算法的优越性就会很明显了。

3. 实验感想

这是最后一次实验啦，这次的实验如果搞清楚退火算法的整个流程写起来思路要比之前的动态规划更清晰！感谢助教和老师在这 12 周当中的帮助和付出！

祝老师和助教万事胜意！也祝算法原理课越办越好！