旅行商实验报告

PB17050948 陈文博

实验内容

1.结构体定义

节点结构体：

```
  typedef struct{
     int**G;//邻接矩阵
     int *s;//边
     int *v;//顶点
     int *path;//路径
     int num;//城市数目
     int distance;//距离
}Graph;
```

结构体定义在TSP.h文件中

2. 函数定义

程序TSP.cpp中的函数包括：

void Create_Graph(Graph&G,int n) void Simple_Greedy(Graph&G) void Enumerate(Graph&G) void Recursive(Graph&G) void A_star(Graph&G)

（1）void Create_Graph(Graph&G,int n)

1. 函数功能：初始化图
2. 函数入口：图结构体G、城市数量n
3. 算法流程：初始化一个n*n的矩阵表示无向连通图以及相关的边、点等信息
4. 算法分析：时间复杂度O（n^2），空间复杂度O（1）

（2）void Enumerate(Graph&G)

1. 函数功能：枚举算法求解TSP
2. 函数入口：图G
3. 算法流程：递归生成G的所有n个无重复节点路径（全排列），记录每次得到的最短距离以及其路径
4. 算法分析：时间复杂度O（n！），空间复杂度O（1）

（3） void Recursive(Graph&G)

1. 函数功能：递归算法求解TSP
2. 函数入口：图G

3. 算法流程：选取起点0和点1作为初始状态，将2加入集合，此时序列0120与0210代价相同，取0120（随意）。加入3时，判断G[0][3]+G[0][1]-G[0][1]与G[1][3]+G[3][2]-G[1][2]与G[2][3]+G[3][0]-G[2][0]的大小关系，选择最小的两点插入，以此类推直到所有的点填入
4. 算法分析：时间复杂度O（n^2），空间复杂度O（1）

（4）void Simple_Greedy(Graph&G)

1. 函数功能：简单贪婪算法求解TSP问题
2. 函数入口：图G
3. 算法流程：每次选取最新结点所连通的未选择的距离最短的点填入路径数组
4. 算法分析：时间复杂度O（n），空间复杂度O（1）

（5） void A_star(Graph&G) (未完成)

1. 函数功能：A*算法求解TSP
2. 函数入口：文件中的总比特数
3. 算法流程：改进简单贪婪算法，选取代价函数h(n)=f(n)+g(n)，其中f(n)表示已走过的总距离，g(n)=min*(num-depth)表示预计离终点最近的距离（min表示到目前为止的最短路径，num表示总城市数量，depth表示搜索深度即已选定的结点数量）对G进行广度优先搜索，用队列存放每次获得的结点以及之前的结点，按它们的f(n)进行排序，并每次优先选取f(n)最小的结点进入最短路径数组
4. 算法分析：时间复杂度O（n^2），空间复杂度O（n）

3. 实验结果

城市数为5：

```
0     8     187     226     159
8     0     89     159     79
187     89     0     229     16
226     159     229     0     70
159     79     16     70     0

Enumerate:
Path: 0 → 1 → 2 → 4 → 3 → 0
Distance: 409
Simple_Greedy:
Path: 0 → 1 → 4 → 2 → 3 → 0
Distance: 558
Recursive:
Path: 0 → 1 → 2 → 4 → 3 → 0
Distance: 409
```

城市数为9：

```
0     286     315     509     252     294     201     337     549
286     0     136     332     400     40     703     179     580
315     136     0     5     314     434     587     313     593
509     332     5     0     780     544     553     407     212
```

```
252     400     314     780     0     430     669     110     765
294     40     434     544     430     0     463     87     269
201     703     587     553     669     463     0     161     30
337     179     313     407     110     87     161     0     255
549     580     593     212     765     269     30     255     0
Enumerate:
Path: 0 → 4 → 7 → 5 → 1 → 2 → 3 → 8 → 6 → 0
Distance: 1073
Simple_Greedy:
Path: 0 → 6 → 8 → 3 → 2 → 1 → 5 → 7 → 4 → 0
Distance: 1073
Recursive:
Path: 0 → 5 → 4 → 1 → 7 → 2 → 8 → 3 → 6 → 0
Distance: 3175
```

城市数为15:

```
0     1105     813     374     605     847     1269     660     446     1283     732
  720
1105     0     591     1287     592     835     1331     599     1190     411     394
  564
813     591     0     483     302     726     534     1305     810     311     569
1293
374     1287     483     0     273     106     665     646     710     1383     474
1241
605     592     302     273     0     389     188     404     980     650     250
131
847     835     726     106     389     0     1357     13     730     1106     295
995
1269     1331     534     665     188     1357     0     229     650     1296     954
  1183
660     599     1305     646     404     13     229     0     1032     195     1365
161
446     1190     810     710     980     730     650     1032     0     1359     198
  138
1283     411     311     1383     650     1106     1296     195     1359     0     583
    715
732     394     569     474     250     295     954     1365     198     583     0
847
720     564     1293     1241     131     995     1183     161     138     715     847
    0
Enumerate:
Path: 0 → 3 → 5 → 7 → 11 → 4 → 6 → 2 → 9 → 1 → 10 → 8 → 0
Distance: 3267
Simple_Greedy:
Path: 0 → 3 → 5 → 7 → 11 → 4 → 6 → 2 → 9 → 1 → 10 → 8 → 0
Distance: 3267
Recursive:
Path: 0 → 1 → 9 → 2 → 4 → 6 → 5 → 8 → 11 → 7 → 10 → 3 → 0
Distance: 6916
```

讨论与总结

枚举算法可以有效的找到全局最优解，但因为其复杂度为O(n!)，当城市数量大时极其耗时，简单贪婪算法可以快速找到局部最优解，但不能保证找到的解是全局最优解。递归法每次加入新元素都是考虑当前最佳状态，故也只能找到局部最优解。A*算法是简单贪婪算法加上启发式搜索的改进，理论上可以较快的找到比简单贪婪更具有全局最优性的解，由于时间关系未能完整实现。此外，现代很多智能优化算法也同样可以用于TSP问题，诸如遗传算法、模拟退火算法、粒子群优化等等，稍作了解。