**算法分析与设计实验报告**

**第 四 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李平凡 | 学号 | 201907040102 | | 班级 | 计科1905 |
| 时间 | 6.2 | 地点 | 软件大楼 | | | |
| 实验名称 | 分支限界法求解tsp问题 | | | | | |
| 实验目的 | 通过搜索资料，要求掌握一些基本问题和典型例题的处理方式 | | | | | |
| 实验原理 | 分支限界法采用的是队列和优先队列两种方式来求解对应问题，tsp显然两种方法均可求解，但采用优先队列的分支限界法时间开销上会明显优于队列式分支限界法，同时我们用bound函数来作为优先级的一个评判标准，因为对这个问题的求解也一定程度上类似于排列树框架求解模式，所以也可以认为是回溯法来求解tsp问题，那么bound函数就是限界条件来进行剪枝操作当然也可以认为是优先队列，那么bound函数就是优先级，我们每次都是根据bound优先级来进行下一个城市的寻找和处理。 | | | | | |
| 实验步骤 | ①首先输入一个整数代表一共多少个城市，接着输入n\*n的矩阵来表示城市之间的距离。  ②利用bound函数来定义优先级，专门求解路径下界，定义优先级为当前路径花费加上到当前点最短路径花费，加上剩下点的到其他点的最短路径花费。利用此优先级来选择下一个城市进行相应的处理。  ③ 当到达最后一个叶子节点（优先队列分支限界法有点类似于排列树）也就意味着有了最终的最优解。 | | | | | |
| 关键代码 | int bound( int t)  {  int min1=max ,min2=max, temmin=0; //这里min与min2初始化一下  for(int i=t; i<=num; i++)  {  //这里求min1  if( g[x[t-1]][x[i]] != -1 && g[x[t-1]][x[i]] < min1)  {  min1 =g[x[t-1]][x[i]]; //这里求要去的第t城市。选择当前到t路径最短的  }    //这里求min2  for(int j =1; j<=num; j++) //贪心的思想，求剩下的路径的下界  {  if( g[x[i]][x[j]]!=-1 && g[x[i]][x[j]]!=0 && g[x[i]][x[j]] < min2)  {  min2 = g[x[i]][x[j]];  }  }  temmin += min2; //这里是剩下点的最短出出路径的和的累加  }  return cl+ min1 + temmin;  }  void prim(int t)  {  //如果到叶节点，则表示所有城市都走过了，这个时候就保存一下最短路径以及相应的城市顺序 ，当然是满足条件就是比之前的bext还要小的路径的长度，每次取最优  if(t>num)  {  if(g[x[num]][1]!=-1 && cl + g[x[num]][1] <best) //记得还有最后回到原点 ，这里必须是有到原点的一条路径  {  for(int i =1; i<=num; i++)  {  bestx[i] =x[i]; //将走过的顺序放入best数组中  }  best = cl + g[x[num]][1]; //保存加上回到原点的的路径  }  }    else{  for(int j =t; j<=num; j++)  {  if( g[x[t-1]][x[j]] !=-1 && (bound(t)<best)) //满足条件的选择要选的城市的编号  {  swap(x[j],x[t]);  cl += g[x[t-1]][x[t]]; //注意，这里不是g[x[t-1]x[j]]，j改成t，因为上面的j与t的位置的数换了。    prim(t+1);    //回溯，恢复走过的。  cl -= g[x[t-1]][x[t]];  swap(x[j],x[t]);  }  }  }  } | | | | | |
| 测试结果 | 通过随机矩阵c文件生成10个城市的各个路径开销，然后在求解tsp问题程序中求得结果。    由于数据规模的限制，所以不可能会解决100甚至1000级别以上的问题，除非拥有足够多的时间。当达到15规模的时候我的电脑已经计算不出来了。最大计算量仅仅能计算到14个城市    但唯一以后的地方就在于自己用的c++里面自带的时间函数14级别和10级别的时间相差甚少，因此怀疑15级别的时候已经超过了计算机承受的最大运算量，可能和底层的关系有关，说明并不是只要有足够的时间就能计算出一个问题的结果。 | | | | | |
| 实验心得 | 紧接着上面实验结果的推理，我们可以大致估算一下数据规模，15个城市共有15！的排列可能性，然后算上矩阵开销，15\*15，每次处理一个城市bound函数会调用一次，它会遍历矩阵每个点来得到相应的优先级，所以时间开销是（15+15\*14+...+15!）\*15\*15左边一项代表所有的点(左边的点并不一定全部都是需要处理，被处理的时候不一定都需要n\*n)，可以理解为排列树点，右边一项15\*15代表的是矩阵规模，因为bound函数需要几乎遍历矩阵上所有路径开销。最终时间开销至少O(n!\*n^2) | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**