**《算法设计与分析》**

**课程实验报告**



**学院：计算机学院（国家示范性软件学院）**

**班级：2019211308**

**姓名：曾世茂(50%) 顾天阳(50%)**

**学号：2019211532 2019211539**

目录

[旅行商问题 3](#_Toc32360)

[一、问题描述 3](#_Toc16292)

[二、 算法设计 3](#_Toc32069)

[1． 回溯法 3](#_Toc29592)

[（1） 整体设计思路 3](#_Toc10602)

[（2） 全局变量说明 4](#_Toc2625)

[（3） 主要函数设计............. 4](#_Toc15044)

[2． 分支界限法 5](#_Toc25175)

[（1） 整体算法设计思路 5](#_Toc27068)

[（2） 全局变量设计 6](#_Toc7397)

[（3） 主要算法设计 6](#_Toc7248)

[三、 运行结果 8](#_Toc13450)

[四、算法效率 9](#_Toc17623)

[图的m着色问题 9](#_Toc29618)

[一、问题描述 9](#_Toc18387)

[二、算法设计 9](#_Toc27017)

[三、运行结果 11](#_Toc10636)

[四、算法效率 12](#_Toc15300)

旅行商问题

一、问题描述

参照教科书，编程实现回溯法、分支限界法，求解旅行商问题，并对比2个算法对同一规模问题的运行时间

参照图1、图2，针对指定起始城市，计算最短旅行路径

1） 图1 15个基站图，起始城市结点20

2） 图2 20个基站图，起始城市结点20

1. 算法设计
   1. 回溯法
      1. 整体设计思路

旅行售货员问题的解空间是一颗排列树。回溯法即为遍历n个节点的全排列找出其中最短的路径。

在递归算法Backtrack中，当i=n时，说明已经走完了所有的节点。若最后一个节点到开始节点之间存在路径，则说明找到了一个回路。若该回路长度小于当前最优路径长度，则将该路径保存下来。

若i<n,则找出所有未访问过的节点。若上一个节点到该节点之间存在道路且加上该路径后总长度小于最优总长度，则将该节点添加到当前路径中，并进入下一层的回溯，否则剪去该子树。结束下一层回溯后应将路径进行恢复。

* + 1. 全局变量说明

1. vector<**int**> \*nodeMap;      //节点名对应表，将题目中不规律节点顺序映射为从0开始
2. **double** \*\*dis;              //两点间的距离
3. **int** nodeNum;               //节点数量
4. **int** \*visit;                //是否访问过某节点
5. **double** currentLength;      //当前长度
6. **double** bestLength;         //当前最短回路长度
7. vector<**int**> road;          //当前路径
8. vector<**int**> bestRoad;      //当前最短路径
9. **int** start;                 //开始节点
10. **long** **int** visitNodeNum = 0; //访问节点数量
    * 1. 主要函数设计  
          backTrace:
11. **void** backTrace(**int** n)
12. {
13. visitNodeNum++;
14. **if** (n >= nodeNum)
15. {
16. //这里还要写回去的
17. **int** lastNode = road.back();
18. **if** (dis[start][lastNode] == -1)
19. {
20. **return**; //没有路回去了
21. }
22. currentLength += dis[start][lastNode];
23. **if** (currentLength <= bestLength)
24. {
25. bestLength = currentLength;
26. bestRoad = road;
27. }
28. currentLength -= dis[start][lastNode];
29. **return**;
30. }
31. **else**
32. {
33. **int** lastNode = road.back();//得到上一个访问的点
34. **for** (**int** i = 0; i < nodeNum; i++)
35. {
36. **if** (visit[i] == 1 || dis[lastNode][i] == -1 || currentLength + dis[lastNode][i] >= bestLength)//已经被访问过或没有到达该点的路径或长度已经超出了当前最短距离
37. {
38. **continue**;
39. }
40. road.push\_back(i);
41. visit[i] = 1;
42. currentLength += dis[lastNode][i];
43. backTrace(n + 1);
44. currentLength -= dis[lastNode][i];//回溯
45. visit[i] = 0;
46. road.erase(road.end() - 1);
47. }
48. }
49. }
    1. 分支界限法
       1. 整体算法设计思路

使用优先队列储存活结点，以该活结点拓展出来的路径的下界作为优先队列的排序依据。活结点的路径长度下界为当前路径的长度家上剩余节点的最小出度之和。每次从优先队列中取出一个节点进行拓展，若拓展节点的路径下界已经大于最优路径长度，则说明该节点不可能产生最短路径，则减去该子树，否则则添加进优先队列中。当访问到树的叶子节点的时候，说明在当前的所有活结点中，没有一个活结点的路径下界小于该叶子节点的路径长度，故该叶子节点的路径即为最优路径，即可退出循环。得益于此思路，分支界限法不必访问完所有的节点，第一个访问到的叶子节点即为所求，故访问的节点数量大大降低。

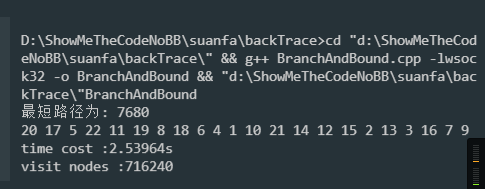
* + 1. 全局变量设计

1. priority\_queue<MinHeapNode \*, vector<MinHeapNode \*>, Operater> H;
2. vector<**int**> \*nodeMap;  //将题目中的节点编号映射成从0开始
3. **double** \*\*dis;          //两点间的距离
4. **double** \*minOut;        //每个节点的最小出度
5. **int** nodeNum;           //节点数量
6. **double** bestLength;     //当前最优路径长度
7. vector<**int**> bestRoad;  //最优路径
8. **int** start;             //起始点
9. **long** visitNodeNum = 0; //已访问的节点数量
   * 1. 主要算法设计
10. **while** (H.size() > 0)
11. {
12. visitNodeNum++;
13. MinHeapNode \*currentNode = H.top();
14. H.pop();
15. **if** (currentNode->currentRoad.size() == nodeNum - 1)
16. {//还剩余一个节点以及回去的道路，故路径已经确定
17. **int** unvisit = 0;
18. **for** (; unvisit < nodeNum; unvisit++)
19. {
20. **if** (currentNode->visist[unvisit] == 0)
21. {
22. **break**;
23. }
24. } //找到尚未访问的点
25. **if** (dis[currentNode->getLastVisit()][unvisit] != -1 &&
26. dis[unvisit][start] != -1 &&
27. currentNode->currentCost + dis[currentNode->getLastVisit()][unvisit] + dis[unvisit][start] < bestLength)
28. {
29. bestLength = currentNode->currentCost + dis[currentNode->getLastVisit()][unvisit] + dis[unvisit][start];
30. currentNode->currentCost = bestLength;
31. currentNode->lowerstCost = bestLength;
32. currentNode->visist[unvisit] = 1;
33. currentNode->currentRoad.push\_back(unvisit);
34. H.push(currentNode);
35. }
36. **else**//该节点无法产生最优路径
37. {
38. **delete** currentNode;
39. }
40. }
41. **else** **if** (currentNode->currentRoad.size() == nodeNum)
42. { //已经找到最优路径
43. bestRoad = currentNode->currentRoad;
44. **delete** currentNode;
45. **break**;
46. }
47. **else**
48. {
49. **for** (**int** i = 0; i < nodeNum; i++)
50. {
51. **if** (currentNode->visist[i] == 1 || dis[currentNode->getLastVisit()][i] == -1)
52. {//已访问过或到该节点无可行路径
53. **continue**;
54. }
55. **else**
56. {   //得到新的节点
57. MinHeapNode \*nextNode = currentNode->getNext(i);
58. **if** (nextNode->lowerstCost < bestLength)//有望望产生最优路径
59. {
60. H.push(nextNode);
61. }
62. **else**
63. {
64. **delete** nextNode;
65. }
66. }
67. }
68. **delete** currentNode;
69. }
70. }
71. 运行结果

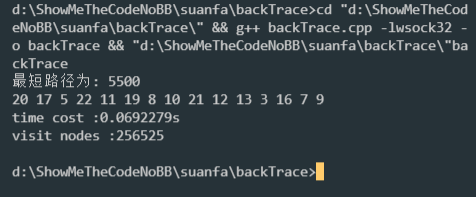
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 问题 | 求解算法 | 最短回路 | 路径长度 | 搜索节点数 | 运行时间 |
| 15个节点 | 回溯 | 20 17 5 22 11 19 8 10 21 12 13 3 16 7 9 | 5500 | 256525 | 0.067863s |
| 分支界限 | 20 17 5 22 11 19 8 10 21 12 13 3 16 7 9 | 5500 | 20928 | 0.0453709s |
| 20个节点 | 回溯 | 20 17 5 22 11 19 18 8 1 10 21 14 12 15 2 13 3 16 7 9 | 6978 | 76106093 | 24.5887s |
| 分支界限 | 20 17 5 22 11 19 18 8 1 10 21 14 12 15 2 13 3 16 7 9 | 6978 | 535816 | 1.77675s |
| 22个节点 | 回溯 | 20 17 5 22 11 19 8 18 6 4 1 10 21 14 12 15 2 13 3 16 7 9 | 7680 | 485699103 | 169.733s |
| 分支界限 | 20 17 5 22 11 19 8 18 6 4 1 10 21 14 12 15 2 13 3 16 7 9 | 7680 | 716240 | 2.56559s |

运行截图：

分支界限法：22个节点



回溯法：15个节点



四、算法效率

1. 回溯法

回溯法实际上搜索了排列树上的所有可能的道路，即所有节点的全排列，故算法的时间复杂度为O(n!)。如15个节点中14！为87,178,291,200，而利用剪枝只访问到了其中的256525，可见剪枝的效果是非常显著的。但当n稍大一些后仍然无法运行。空间复杂度除去递归需要需要的空间外只需储存两点间的距离，故空间复杂度为O(n²)

1. 分支界限法

分支界限法剪枝效率取决于预估下界的与真实下界的差距，差距越小剪枝越有效。在最坏情况下，分支界限法仍然需要搜索几乎整棵树，故最坏情况下时间复杂度仍为O(n!),空间复杂度最坏情况下需要储存所有可能的路径节点，故空间复杂度为O(n!)

图的m着色问题

一、问题描述

给定无向连通图G和m种不同的颜色。用这些颜色为图G的各顶点着色，每个顶点着一种颜色。是否有一种着色法，使G中每条边的2个顶点着有不同颜色。这个问题是图的m着色判定问题。若一个图最少需要m种颜色才能使图中每条边连接的2个顶点着不同颜色，则称这个数m为该图的色数。求一个图的色数m的问题称为图的m可着色优化问题。

实验基于昆明LTE网络，从中分别选取22个和42个基站进行图的抽象，进而对基站抽象成的图进行着色。

二、算法设计

根据回溯法的递归描述框架Backtrack设计图的m着色算法。用图的邻接矩阵a表示无向连通图G=(V,E)。若(i,j)属于图G=(V,E)的边集E，则a[i][j]=1，否则a[i][j]=0。整数1，2,…,m用来表示m种不同颜色。顶点i所着的颜色用x[i]表示。数组x[1:n]是问题的解向量。问题的解空间可表示为一棵高度为n+1的完全m叉树。解空间树的第i层中每个结点都有m个儿子，每个儿子相应于x[i]的m个可能的着色之一。第n+1层结点均为叶结点。

其中，Backtrack(i)搜索解空间中第i层子树。类Color的数据成员记录解空间中结点信息，以减少传给Backtrack的参数。sum记录当前已经找到的可m着色方案数。

在算法Backtrack中，当i>n时，算法搜索至叶结点，得到新的m着色方案，当前找到的可m着色方案数sum增1。

当i<=n时，当前扩展结点Z是解空间中的内部结点。该结点有m个儿子结点x[i]。对当前扩展结点Z的每个儿子结点，由函数Ok()检查其可行性，并以深度优先的方式递归地对可行子树搜索，或剪去不可行子树。

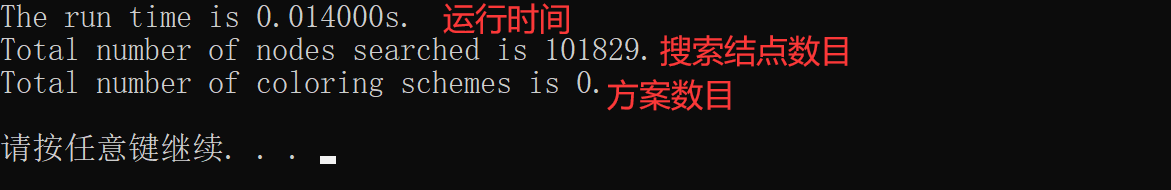
图m可着色问题的回溯算法可描述如下：

1. **class** Color
2. {
3. **friend** **int** mColoring(**int**, **int**, **int** \*\*);
5. **private**:
6. **bool** Ok(**int** k);
7. **void** Backtrack(**int** t);
9. **int** n,    //图的顶点数
10. m,    //可用颜色数
11. \*\*a,  //图的邻接矩阵
12. \*x;   //当前已找到的可m着色方案
13. **long** sum; //当前已找到的可m着色方案数
14. };
16. **bool** Color::Ok(**int** k) //检查颜色可用性
17. {
18. **for** (**int** j = 1; j <= n; j++)
19. **if** ((a[k][j] == 1) && (x[j] == x[k]))
20. **return** **false**;
21. **return** **true**;
22. }
24. **void** Color::Backtrack(**int** t)
25. {
26. sum\_nodes++;
27. **if** (t > n)
28. {
29. sum++;
30. // for (int i = 1; i <= n; i++)
31. //     printf("%d ", x[i]);
32. // printf("\n");
33. }
34. **else**
35. {
36. **for** (**int** i = 1; i <= m; i++)
37. {
38. x[t] = i;
39. **if** (Ok(t))
40. Backtrack(t + 1);
41. x[t] = 0;
42. }
43. }
44. }

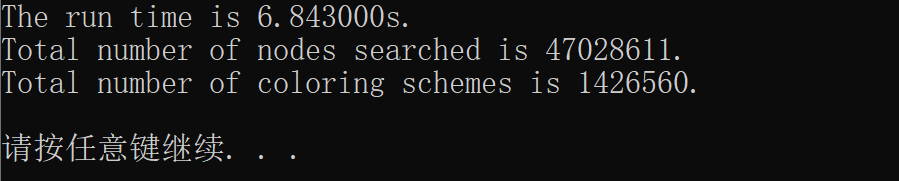
三、运行结果

1、22个基站

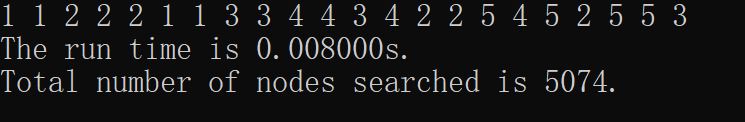
为了找到尽可能少的着色数，令m从小到大进行测试，当m=4时，程序运行结果如下：



这表示当有22个基站时，最小着色数大于4，于是令m=5，再次运行，程序运行结果如下：



进而针对m=5找到一种可行方案如下：

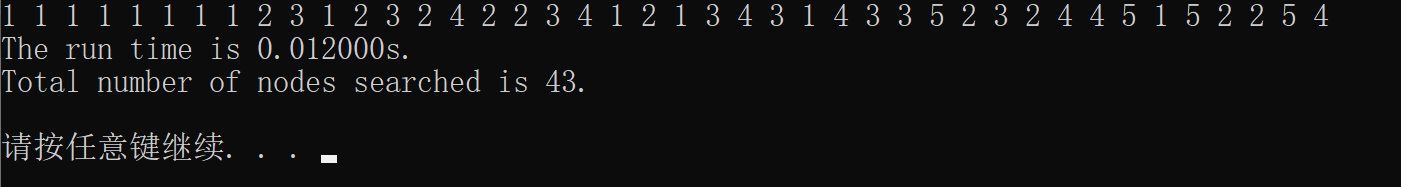


列出如下统计结果：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 颜色总数m | 搜索过的结点数L | 程序运行时间T | 着色方案数目sum | 一种可行方案x | 搜索一种可行方案的结点数L’ | 搜索一种可行方案的运行时间T’ |
| 5 | 47028611 | 6.843s | 1426560 | 1,1,2,2,2,1,1,3,3,4,4,3,4,2,2,5,4,5,2,5,5,3 | 5074 | 0.008s |

2、42个基站

由于n=42相对较大，加之时间复杂度是指数级别的，从而使得程序运行时间骤增，已经无法在可观时间内获取着色方案总数，这里针对特定的m值找出对应的可行方案及其相关信息，运行结果如下：



统计结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 颜色总数m | 一种可行方案x | 搜索一种可行方案的结点数L | 搜索一种可行方案的运行时间T |
| 5 | 1 1 1 1 1 1 1 1 2 3 1 2 3 2 4 2 2 3 4 1 2 1 3 4 3 1 4 3 3 5 2 3 2 4 4 5 1 5 2 2 5 4 | 43 | 0.012 |
| 10 | 1 1 1 1 1 1 1 1 2 3 1 2 3 2 4 2 2 3 4 1 2 1 3 4 3 1 4 3 3 5 2 3 2 4 4 5 1 5 2 2 5 4 | 43 | 0.012 |
| … | … | … | … |

四、算法效率

图m可着色问题的回溯算法的计算时间上界可以通过计算解空间树中内结点个数来估计。图m可着色问题的解空间树中内结点个数是。对于每个内结点，在最坏情况下，用Ok()函数检查当前扩展结点的每个儿子所对应的颜色的可用性需耗时O(mn)。因此，回溯法总的时间耗费是。