离散数学课程实验

问题描述

本次报告拟解决的问题:

设有n个城市，城市之间均有道路，道路的长度均大于或等于0，可能是∞（对应关联的城市之间无交通线）。一个旅行商从某个城市出发，要经过每个城市一次且仅一次，最后回到出发的城市，求最短路线长度。 （TSP最短路线长度）

算法描述

本次实验采用最小生成树算法。

算法流程：

1.Prim算法求得最小生成树

2.找到欧拉回路，抄近路的方式获得哈密顿回路

3.按照哈密顿回路顶点计算回路长度

数据来源

实验数据来自网站TSPLIB

(http://comopt.ifi.uni-heidelberg.de/software/)，

本次实验用到的数据编号为：

gr17 , gr21 , gr48 , gr120 , fri26 .

数据的结构：

以上编号的数据均为矩阵下三角存储，直接存储无向网的邻接矩阵信息。实验前先将源文件处理为纯数据，实验过程中直接读取处理后的文件。

代码实现

编程语言：C/C++;

运行环境：Visual Studio Community 2017

关键代码摘要说明:

1.图的存储结构：

程序采用邻接矩阵存储地图信息，并使用动态分配空间的方式创建邻接矩阵，以便减少内存消耗。

//----------------------- Graph 组成元素定义-----------------------

#define INFINITY\_MY -1 //用在网中，表示两点不相邻

//--------顶点集(顺序表)------

typedef int VexElemType; //顶点类型，定义为整型，为了简便以1 2 3……代表各个顶点

//也可以定义为其它类型，存储TSP问题中的城市名称等

struct VertexSet //顶点集 整型数组 存储名称

{

VexElemType \*elem;

int maxsize;

};

//-------邻接矩阵(二维的顺序表)------

typedef int AdjMatElem; //采用整型的距离

typedef int\*\* Mat; //非结构体的矩阵类型，用于计算过程

struct AdjMat //邻接矩阵

{

AdjMatElem \*\*elem;

int maxsize; // AdjMat 为方阵

};

2.关键函数原型：

//求最小生成树,结果存储一维向量 prim 中

Status Prim(GraphM G, int\* &prim);

//将最小生成树结果转存在矩阵 treemat 中

Status GetTreeMat(GraphM G, int \*prim, int\*\* &treemat);

//得到哈密顿回路

Status GetHCircuit\_1(int\*\* tree, int N, int p, int\* &Hcircuit);

//得到另一条哈密顿回路

Status GetHCircuit\_2(int\*\* tree, int N, int p, int\* &Hcircuit);

//计算回路长度

int CountHCicuit(GraphM G, int\* &Hcircuit);

3.最小生成树算法：

采用Prim算法，将运行结果（int\* adjvex）存储在一维数组中带回主函数。

//执行 N-1 次可以得到结果

int p;

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

{

p = FindMinCost(lowcost, finished, N); //找到最小值对应的顶点

finished[p] = true; //赋值为真

ChangeLowCost(p, mat[p], adjvex, lowcost, finished, N); //修改最低花费,同时标明路径来源

}

4.哈密顿回路算法：

此处采用近似算法，通过深度遍历求得一条哈密顿回路，调转深度遍历的子结点访问顺序得到另一条回路。利用栈循环实现，以便提高效率。

while (!StackEmpty\_t(S))

{

Pop\_t(S, p);

Hcircuit[i++] = p;

for (int q = LastAdjVex(tree, N, p); q > 0; q = PriorAdjVex(tree, N, p, q))

{

if (!visited[q])

{

visited[q] = true;

Push\_t(S, q);

}

}

}

5.源程序：

 

6.实验数据





实验结果表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 顶点数 | 最优解 | 计算解 | 运行时间 | 误差 |
| 17 | 2085 | 2277 | 0.001s | 9.21% |
| 21 | 2707 | 3175 | 0.001s | 17.3% |
| 26 | 937 | 1073 | 0.001s | 14.5% |
| 48 | 5046 | 6476 | 0.002s | 28.3% |
| 120 | 6942 | 9593 | 0.015s | 38.1% |

实验结果分析

本次实验在顶点数为120及以下时均可快速得到结果，但是随着顶点数的增加误差越来越大。

对于运行时间：

在顶点数26及以下时，运行时间<=1ms。

在顶点数为48时，达到2ms，增幅较缓。

在顶点数由48增加到120时，用时增加到15ms，增幅变大。

总体来讲，实验数据范围内均可快速得到近似答案。

对于误差：

在顶点数为17以下时，推测误差大约在10%以下。

在顶点数为17~30时，推测误差大约在10%~20%之间。

在顶点数大于120时，误差可能高于40%。

总体来讲，算法误差较大。

实验误差分析

1.最小生成树算法适用与两边之和大于第三边的情况，实验数据并不能保证满足算法适用条件。

2.在求哈密顿回路时，使用了近似算法，利用深度遍历的过程取得欧拉回路进而得到哈密顿回路，导致并不能取得所有的哈密顿回路情况，此处也可以作为进一步优化算法的出发点。

总评

本次实验：

算法执行速度可以接受，但随着问题规模的增大，误差增幅太大，不能得到满意的答案

本次实验采用近似算法求解，重要的误差来源主要有两个：

1.不一定满足生成树算法使用条件。

2.哈密顿回路不能取到所有的情况。