**《算法设计与分析》实验报告**

# 实验八 综合练习

|  |  |  |  |
| --- | --- | --- | --- |
| **报告书** | | | |
| 姓名 | 吴宇敖 | 指导教师 |  |
| 学号 | 176001752 | 日 期 | 2020.6.21 |
| 班级 | 17计算机软件一班 |  |  |
| **实验内容** | | | |
| **[本次实验无OJ]**  1）求下图最大流，给出推演过程。    2)**了解什么是旅行商TSP问题。请设计算法求解TSP问题，只需给出算法框架和方法说明。** | | | |
| **实验目的** | | | |
| 1. 了解网络流及相关算法 2. 学习如何使用算法求解问题 | | | |
| **实验过程和步骤** | | | |
| **以下仅供参考，请在报告中删除**  **【实验题目】**  1）求下图最大流，给出推演过程。     1. v1->v2->v5->v7 min{13-5,5-3,9-5}=2 上述节点的连接处加2 2. V1->v2->v4->v5->v7 min{13-7,6-3,6-2,9-7}=2上述节点的连接处加2 3. V1->v2->v4->v7 min{13-9,6-5,4-2}=1上述节点的连接处加1 4. V1->v3->v2 上述节点的连接处减1,v1->v3上述节点的连接处加1 5. V1->v3->v6->v7 min{9-2,5-0,10-1}=5上述节点的连接处加5 6. V1->v3->v4->v6->v7 min{9-7,5-2,4-1,10-6}=1上述节点的连接处加2   最大流为：11+9=20  图示为：(v1,v2)=11 (v2,v5)=5 (v5,v7)=9 (v1,v3)=9 (v3,v2)=0 (v2,v4)=6  (v4,v5)=4 (v4,v7)=3 (v4,v6)=3 (v6,v7)=8 (v3,v6)=5 (v3,v4)=4 (v1,v3)=9  **【实验题目】**  2)**了解什么是旅行商TSP问题。请设计算法求解TSP问题，只需给出算法框架和方法说明。**  **【程序代码及注释】**  **g[N][N]//存储城市之间的距离**  **//核心函数，求出动态规划dp数组**  **void TSP(){**  **//初始化dp[i][0]**  **for(int i = 0 ; i < N ;i++){**  **dp[i][0] = g[i][0];**  **}**  **//求解dp[i][j],先跟新列在更新行**  **for(int j = 1 ; j < M ;j++){**  **for(int i = 0 ; i < N ;i++ ){**  **dp[i][j] = INF;**  **//如果集和j(或状态j)中包含结点i,则不符合条件退出**  **if( ((j >> (i-1)) & 1) == 1){**  **continue;**  **}**  **for(int k = 1 ; k < N ; k++){**  **if( ((j >> (k-1)) & 1) == 0){**  **continue;**  **}**  **if( dp[i][j] > g[i][k] + dp[k][j^(1<<(k-1))]){**  **dp[i][j] = g[i][k] + dp[k][j^(1<<(k-1))];**  **}**  **}**  **}**  **}**    **}**  **//判断结点是否都以访问,不包括0号结点**  **bool isVisited(bool visited[]){**  **for(int i = 1 ; i<N ;i++){**  **if(visited[i] == false){**  **return false;**  **}**  **}**  **return true;**  **}**  **//获取最优路径，保存在path中,根据动态规划公式反向找出最短路径结点**  **void getPath(){**  **//标记访问数组**  **bool visited[N] = {false};**  **//前驱节点编号**  **int pioneer = 0 ,min = INF, S = M - 1,temp ;**  **//把起点结点编号加入容器**  **path.push\_back(0);**    **while(!isVisited(visited)){**  **for(int i=1; i<N;i++){**  **if(visited[i] == false && (S&(1<<(i-1))) != 0){**  **if(min > g[i][pioneer] + dp[i][(S^(1<<(i-1)))]){**  **min = g[i][pioneer] + dp[i][(S^(1<<(i-1)))] ;**  **temp = i;**  **}**  **}**  **}**  **pioneer = temp;**  **path.push\_back(pioneer);**  **visited[pioneer] = true;**  **S = S ^ (1<<(pioneer - 1));**  **min = INF;**  **}**  **【算法分析】**  TSP算法，我采用的是动态规划的求解思路：假设有0、1、2、3个城市，那么从0开始就有以下三种可能：  （1）从0出发，到1，然后再从1出发，经过[2,3]这几个城市，然后回到0，使得花费最少。  （2）从0出发，到2，然后再从2出发，经过[1,3]这几个城市，然后回到0，使得花费最少。  （3）从0出发，到3，然后再从3出发，经过[1,2]这几个城市，然后回到0，使得花费最少。  可以发现，三个小的解决方案的最优解，构成了大的解决方案，所以这个问题具有最优子结构，可以用动态规划来实现。  设置一个二维的动态规划表dp,定义符号{1,2,3}表示经过[1,2,3]这几个城市，然后回到0。设置一个二维数组l保存两个城市之间的距离。  要求三个方案的最小值意味需要求下面的最小值：  ①dp[0][{1,2,3}]=min{l[0][1]+dp[1][{2,3}] ，l[0][2]+dp[2][{1,3}]，l[0][3]+dp[3][{1,2}]}  ②dp[1][{2,3}] = min{ l[1][2]+dp[2][{3}] ，l[1][3]+dp[3][{2}]}  ③dp[2][{3}] = l[2][3]+l[3][0] | | | |