《离散数学 II》实验报告

课程简介

课程名称：离散数学Ⅱ

授课对象：计算机科学与技术 本科

教材版本：《离散数学教程》北京大学出版社

学时：52　　其中：理论：44　实验：8

授课教师：马慧　　电子邮箱：mahui@ouc.edu.cn

实验一(必做，基本实验，2学时)

实验题目：可图化、可简单图化、连通性的判别及图的矩阵表达

实验目的：

1、掌握可图化的定义及判断方法；

2、掌握可简单图化的定义及判断方法；

3、掌握连通图的判断方法；

4、掌握图的矩阵表达。

实验要求：

1、给定非负整数序列（例如：(4,2,2,2,2)）。

2、判断此非负整数序列是否是可图化的，是否是可简单图化的。

3、请利用 Havel 定理（定理3）输出判断的过程与结果

4、如果可简单图化，请输出其中一个简单图的邻接矩阵，并判断该图是否连通

*说明：要求学生设计的程序不仅对给定非负整数序列得出正确结果，还要对教师测试数据集得出正确结果。

实验内容和实验步骤：

步骤1：判断可图化性

将给定的非负整数序列排序为降序，即 $d1 \geqslant d2 \geqslant \cdots \geqslant dn$

应用 Havel 定理的过程，逐步减少度数，直到得到一个全0序列或发现无法继续（即某个步骤中得到的序列不满足可图化条件）。

步骤2：判断可简单图化性

在判断可图化的过程中，同时检查每个顶点的度数是否超过 n-1。

如果发现某个顶点的度数超过 n-1，则序列不可简单图化。

步骤3：构造简单图的邻接矩阵

如果序列是可简单图化的，根据度数序列构造一个简单图。

从度数最大的顶点开始，依次连接它到度数次大的顶点（或其他符合条件的顶点），直到它的度数减为0。

重复上述过程，直到所有顶点的度数都为0。

在构造过程中记录边的连接情况，形成邻接矩阵。

步骤4：判断图的连通性

检查邻接矩阵，确定是否存在从任意顶点到其他所有顶点的路径。

（使用 dfs 算法）如果存在这样的路径，则图是连通的；否则，图不是连通的。

实验内容和实验步骤

1. 需求分析

任务陈述：

设计一个程序，用于判断给定的非负整数序列是否是可图化的，进一步判断是否是可简单图化的。如果序列是可简单图化的，则输出其中一个简单图的邻接矩阵，并判断该图是否连通。

输入：

一个非负整数序列，例如：(4, 2, 2, 2, 2)。

输出：

是否是可图化的。

是否是可简单图化的。

如果可简单图化，输出其中一个简单图的邻接矩阵。

判断该简单图是否连通。

程序功能：

检查序列是否满足可图化条件（根据 Erdős-Gallai 定理或 Havel-Hakimi 定理）。

如果可图化，进一步检查是否满足简单图条件。

如果可简单图化，生成一个简单图并输出其邻接矩阵。

判断生成的简单图是否连通。

2. 概要设计

数据结构定义：

整数列表：存储输入的非负整数序列。

邻接矩阵：二维数组，用于存储简单图的边。

主程序流程：

输入非负整数序列。

检查序列是否可图化（使用 Havel 定理）。

如果可图化，进一步检查是否可简单图化。

如果可简单图化，生成一个简单图并输出其邻接矩阵。

判断生成的简单图是否连通。

程序模块调用关系：

输入模块：获取非负整数序列。

可图化检查模块：使用 Havel 定理判断序列是否可图化。

简单图检查模块：在可图化的基础上判断序列是否可简单图化。

图生成模块：生成简单图的邻接矩阵。

连通性判断模块：判断生成的简单图是否连通。

算法的时空分析：

时间复杂度：

可图化判断（Havel 定理）：O(n^2)，其中 n 是序列的长度。

图生成（Havel-Hakimi 算法）：O(n^2)，因为需要构建邻接矩阵并调整度数。

连通性判断（DFS）：O(n^2)，因为需要访问每个顶点及其邻接顶点。

空间复杂度：

邻接矩阵：O(n^2)，用于存储简单图的边。

其他数据结构：O(n)，用于存储度数和顶点列表。

实验测试数据、代码及相关结果分析：（由学生填写）

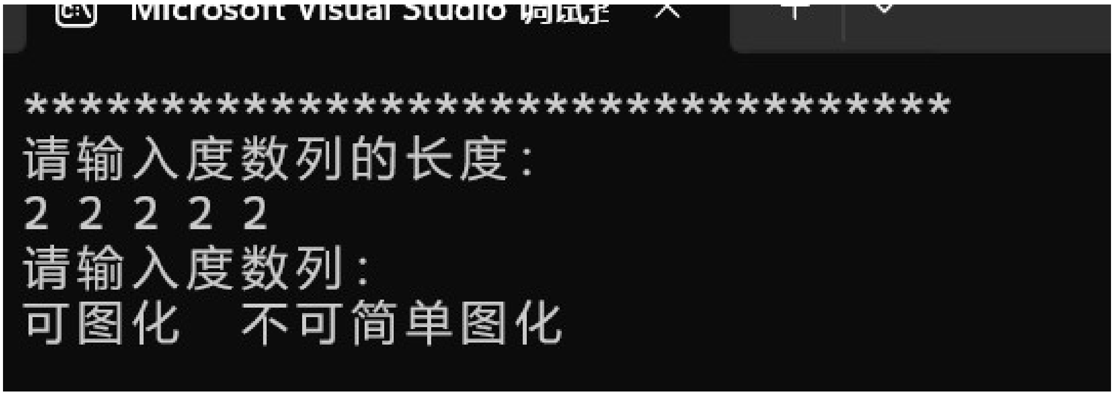
代码

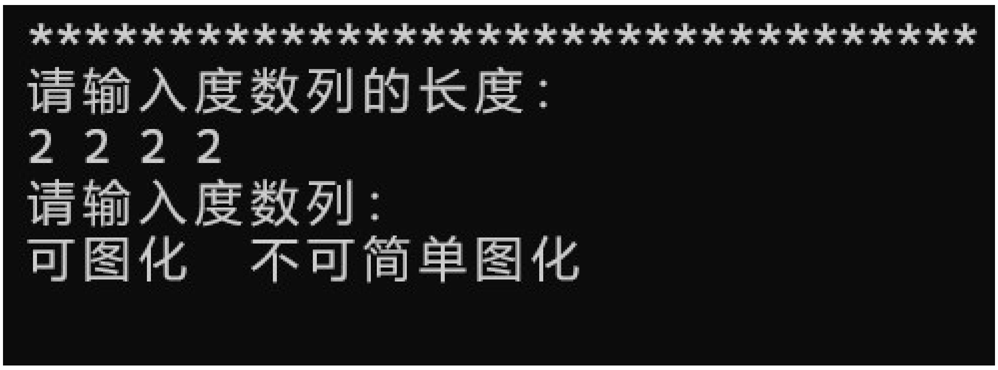
```
#include <iostream>
#include <vector>
#include <algorithm> // 用于 std::sort
#include <cstring>   // 用于 memset
using namespace std;

int sq[1000][1000] = { 0 }; // 初始化邻接矩阵为0
```

```
pair<int, int> d[1000];   // 存储度数和原索引的数组
bool st[1000] = { 0 };      // 状态数组，初始化为 false
int l = 1;                 // 节点的数量

// 返回两个整数中较小的一个
int min(int x, int y) {
    return (x <= y) ? x : y;
}

// 检查度数序列是否可以构成图
bool is_graphable(const vector<int>& degrees) {
    int sum = 0;
    for (int degree : degrees)
        sum += degree; // 计算度数的总和
    return sum % 2 == 0; // 总度数必须是偶数
}

// 检查度数序列是否可以构成简单图
bool is_easy_graphable(const vector<int>& degrees) {
    if (!is_graphable(degrees)) return false; // 先判断是否可以构成图
    int n = degrees.size();
    for (int i = 1; i <= n; i++) {
        int sum2 = 0;
        for (int j = 0; j < i; j++)
            sum2 += degrees[j]; // 计算前 i 个节点的度数之和
        int sum3 = 0;
        for (int k = i; k < n; k++)
            sum3 += min(degrees[k], i); // 计算后续节点中，限制其度数为 i 后的和
        if (sum2 > sum3 + i * (i - 1)) return false; // 判断是否满足条件
    }
    return true; // 可以构成简单图
}

// 深度优先搜索，检查图的连通性
void dfs(int x) {
    if (st[x]) return; //如果该节点已经被访问，直接返回
    st[x] = true; // 标记该节点为已访问
    for (int i = 1; i <= l; i++) {
        if (sq[x][i]) dfs(i); // 如果 x 与 i 有连接，继续访问 i
    }
}

//根据度数序列构建邻接矩阵
void printAdjMartix() {
```

```cpp
    // 将度数按降序排列
    sort(d + 1, d + l + 1, greater<pair<int, int>>());
    for (int i = 1; i <= l; i++) {
        int top = d[i].first; // 当前节点的度数
        if (top == 0) break; // 如果度数为0，跳出循环

        for (int j = 1; j <= top; j++) {
            if (i + j <= l) { // 防止数组越界
                sq[d[i].second][d[i + j].second] = 1; // 创建连接
                sq[d[i + j].second][d[i].second] = 1; // 无向图连接
                d[i + j].first--; // 减少被连接节点的度数
            }
        }

        // 再次按降序排列剩余的度数
        sort(d + i + 1, d + l + 1, greater<pair<int, int>>());
    }

    // 输出邻接矩阵
    printf("    ");
    for (int i = 1; i <= l; i++) {
        printf("V%d ", i); // 打印列头
    }
    cout << endl;
    for (int i = 1; i <= l; i++) {
        printf("V%d ", i); // 打印行头
        for (int j = 1; j <= l; j++) {
            printf(" %d ", sq[i][j]); // 打印邻接矩阵的内容
        }
        cout << endl;
    }
}

int main() {
    cout << "**********************************\n";
    cout << "请输入度数列的长度：\n";
    cin >> l; // 输入节点数量
    memset(sq, 0, sizeof(sq)); // 初始化邻接矩阵
    memset(st, 0, sizeof(st)); // 初始化状态数组

    cout << "请输入度数列：\n";
    for (int i = 1; i <= l; i++) {
        int a;
        cin >> a;
```

```
        d[i] = { a, i }; // 存储度数及其原始索引
    }

    vector<int> degrees(l);
    for (int i = 1; i <= l; i++) {
        degrees[i - 1] = d[i].first; // 填充度数向量
    }

    // 判定是否可以构成图
    if (is_graphable(degrees)) {
        cout << "可图化  ";
    }
    else {
        cout << "不可图化  ";
        return 0; // 如果不可图化则提前退出
    }

    // 判定是否可以构成简单图
    if (is_easy_graphable(degrees)) {
        cout << "可简单图化\n" << endl;
        printAdjMartix(); // 构建邻接矩阵并输出
        int fr = 0;
        for (int i = 1; i <= l; i++) {
            if (!st[i]) {
                fr++; // 统计连通分支数量
                dfs(i); // 访问该组件
            }
        }
        if (fr == 1) {
            printf("\n连通\n");
        }
        else {
            printf("不连通，连通分支数为%d\n", fr);
        }
    }
    else {
        cout << "不可简单图化\n" << endl;
    }

    return 0;
}
```





实验总结：（由学生填写）

在这次实验中，我深刻体会到了理论与实践相结合的重要性。起初，面对非负整数序列的图化判断和简单图生成这些抽象概念，我感到有些迷茫。但通过逐步深入学习，从理解 Havel 定理到掌握 Havel-Hakimi 算法，再到应用 DFS 判断图的连通性，我逐渐将这些理论知识转化为实际操作能力。这个过程不仅增强了我的专业技能，也让我更加明白了学习的真谛——学以致用。

在实验过程中，我还深刻感受到了团队协作的力量。虽然这次实验主要是个人完成，但在遇到难题时，我积极向同学和老师请教，他们的解答和建议让我受益匪浅。这种互帮互助的氛围，让我更加珍惜团队合作的机会，也让我明白了在学术研究中，开放的心态和乐于分享的精神是多么重要。

附录：实验报告的要求

实验报告是反映学生实验效果的最主要的依据，也是学生正确地表达问题、综合问题和发现问题的能力的基本培养手段，因而是非常重要的内容，本课程的实验报告中要包括以下几项内容：

（一） 实验题目；

（二） 实验目的；

（三） 实验要求；

（四） 实验内容和实验步骤；

1．需求分析：陈述程序设计的任务，强调程序要做什么，明确规定：

（1）输入的形式和输入值的范围；

（2）输出的形式；

（3）程序所能实现的功能；

2．概要设计：说明用到的数据结构定义、主程序的流程及各程序模块之间的调用关系。

3．详细设计：提交带注释的源程序或者用伪代码写出每个操作所涉及的算法。

4．调试分析：

（1）调试过程中所遇到的问题及解决方法；

（2）算法的时空分析；

（五） 实验结果：列出对于给定的输入所产生的输出结果。若可能，

测试随输入规模的增长所用算法的实际运行时间的变化。

（六） 实验总结：有关实验过程中的感悟和体会、经验和教训等。