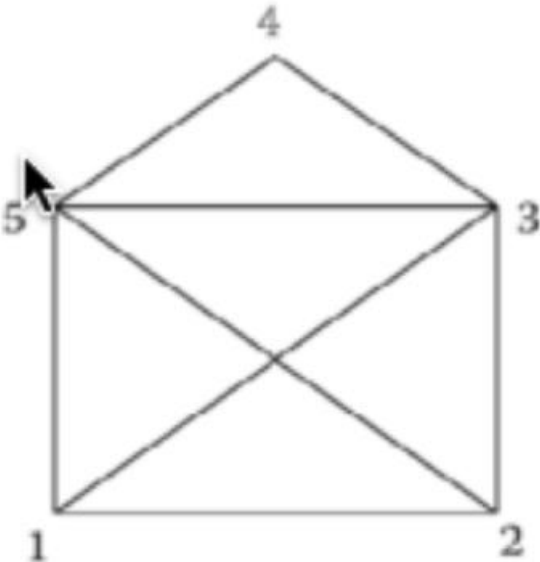


《数据结构》上机报告

2019 年 12 月 1 日

姓名：李佳庚 学号：1852409 班级：计算机1班 得分：_____

实验题目	图	
问题描述	<p>圣诞节马上到了，我们用一笔画画出圣诞老人的房子吧。现在的问题是，一共有多少种画法呢？</p> <p>请你写一个程序，从下图所示房子的左下角（数字1）开始，按照节点递增顺序，输出所有可以一笔画完的顺序，要求一条边不能画两次。</p> 	
基本要求	<p>(1) 程序要添加适当的注释，程序的书写要采用 缩进格式 。</p> <p>(2) 程序要具在一定的 健壮性，即当输入数据非法时， 程序也能适当地做出反应，如 插入删除时指定的位置不对 等等。</p> <p>(3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。</p> <p>(4) 根据实验报告模板详细书写实验报告，在实验报告中给出主要算法的复杂度分析。</p>	
	已完成基本内容（序号）：	1, 2, 3, 4, 5

选做要求	<ol style="list-style-type: none"> 1. 利用离散数学知识，从数学层面解决一笔画问题。 2. 利用程序语言，提出解决一笔画问题的方法。 3. 考虑一笔画问题的规律，并提出能否一笔画在离散数学方面和计算机程序设计语言方面的决定因素。
数据结构设计	<div data-bbox="284 481 799 600" data-label="Text"> <pre> 3 4 #define NODENUM_MAX 100 5 // 邻接矩阵 6 int map[NODENUM_MAX][NODENUM_MAX]; 7 </pre> </div> <p>很简单，这样就够了。</p> <p>本题名为一笔画问题，作为图的实验报告题目。不难理解，这道题与图论有关。 根据一笔画题目的描述： 请你写一个程序，从下图所示房子的左下角（数字1）开始，按照节点递增顺序，输出所有可以一笔画完的顺序，要求一条边不能画两次。↵</p> <p>轻松可以得知：本题要求的是从1号位置开始出发，走遍图中所有的边。但对于是否需要回到原本的1号位置并没有具体的要求。</p> <p>我们可以知道：</p> <ul style="list-style-type: none"> 一、本题一笔画所得路径应为通路，但也可能为回路(其实不可能)。 二、本题一笔画要求经过所有的路径。（当然这个条件下也会经过所有的顶点） <p>所以，不难得知：本题就是一个以固定起点寻找欧拉通路的题目。</p> <p>回想“戈尼斯堡七桥问题”，欧拉以简单图的方式形象地刻画了戈尼斯堡七座桥之间的关系。欧拉是利用数学知识，将实际问题抽象成了数学问题。而我们可以利用计算机的程序语言，将原本已经抽象过的图继续抽象，变为邻接矩阵/邻接表。</p> <p>其实这个不断抽象，利用更高层面的角度思考问题，就是我们高级语言在处理复杂问题时需要考虑的东西，也是不可缺少的一个步骤。</p> <p>在这道题上，只需要邻接矩阵就可以解决。 所以数据结构也就是一个二维数组罢了。 如果有更多的节点需要考虑，那么不妨将map设置成指针，然后进行动态内存申请。</p>
功能(函数)说明	<p>我认识的一位大佬，在我向他吐槽“不想写有关图的作业，因为觉得太麻烦的时候”，告诉我：“图和树不是差不多吗？”</p> <p>也就是这句话让我感觉到了，处理图的问题时，一些在处理有关树的问题上能够有很好效果的方法，或者是就是出身于树概念的算法得到了很好的应用的原因。 比如BFS，比如DFS。</p> <p>说起DFS，这个算法就是在计算机层面上解决欧拉通路问题的最容易理解的算法。（下为代码）</p>

```

void DFS(ALGraph G, int v) {
    ArcNode *p;
    cout << G.vertices[v].data;
    visited[v] = 1;
    p = G.vertices[v].firstarc;
    while (p) {
        if (!visited[p->adjvex]) {
            cout << ' ';
            DFS(G, p->adjvex);
        }
        p = p->arcToNextNode;
    }
}

void DFSTraverse(ALGraph G) {
    for (int v = 0; v < G.ddnum; ++v)
        visited[v] = 0;
    for (int v = 0; v < G.ddnum; ++v) {
        if (!visited[v]) {
            cout << ' {';
            DFS(G, v);
            cout << ' }';
        }
    }
}

```

我们不妨回顾一下这个问题：

从 1 号位置开始，经过所有的边，所有的顶点。

那么我们不妨就将 1 号位置看作一棵树的根节点。

我们需要寻找的欧拉通路就是这个以 1 号位置为根节点的树的一条分支而已。只不过这条分支满足一些条件——一共有 9 层。比如：1-2-5-3-4-5-1-3-2。并且这 9 层每两层之间就是一条唯一的没有在之前被走过的边。

于是我们还需要加一个 visited 数组：

```

#define LEN_MAX 5000
bool visited[LEN_MAX]

```

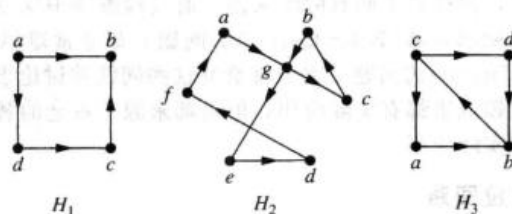
由此，我们不难看出，对于一条分支的深入到最低层的遍历，这不就是 DFS 吗。

但是这样我觉得可能没有什么效率。毕竟有些很复杂的图，节点很多，边也很多。所以如果使用 DFS 进行检测，那么很可能用了很长的时间也跑不出结果。

所以我们有理由在数学层面上对其进行更笼统但是更有用处的分析。

（下图自离散数学及其应用）

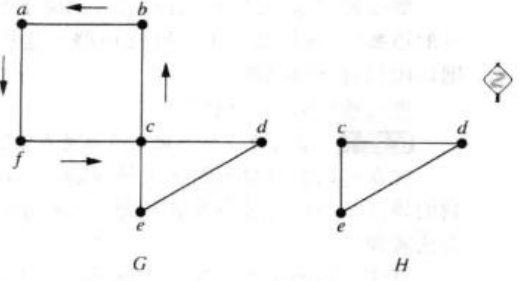
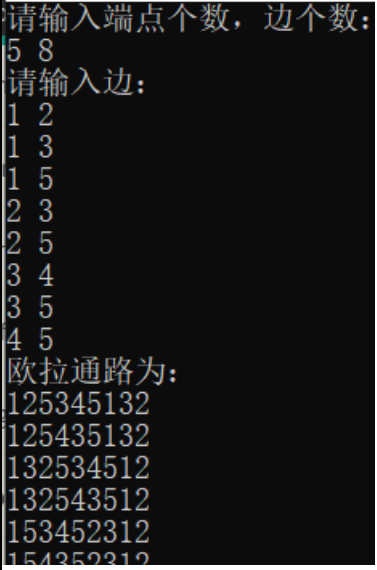
例 2 在图 4 中，哪些有向图有欧拉回路？在没有欧拉回路的那些图中，哪些具有欧拉通路？



其实多看几个图就很轻松得知：判断多重图是否有欧拉回路和欧拉通路，存在着比较简单的标准，如果结点个数为 n ，那么只需要 n 的时间就可以完成对欧拉通路是否存在的检测。

（可惜的是不清楚具体到底是那一条路）

我们想想，如果一个连通图具有欧拉通路，一个节点的入度如果是偶数的话，那么也就是说，经过以它为端点的所有的边的话，如果以这个节点为起点，那么有可能这个节点也为终点。

	<p>但是如果像本题一样，以 1 号位置为起点，由于 1 号位置连接着 2, 3, 4 三个节点，所以如果以其为起点，必定不可能以 1 号位置为终点。即使回到了一号节点啊，也会马上前往下一个节点。</p> <p>就像书中所说的：</p> <p>假设 G 是连通多重图且 G 的每一个顶点的度都是偶数。一条边一条边地构造从 G 的任意顶点 a 开始的简单回路。设 $x_0 = a$。首先任意地选择一条关联 a 的边 $\{x_0, x_1\}$，因为 G 是连通的，所以这是可行的。通过一条一条地增加边来继续构造，建立尽量长的简单通路 $\{x_0, x_1\}, \{x_1, x_2\}, \dots, \{x_{n-1}, x_n\}$，直到不能再向这条通路中增加边。当我们到达一个顶点，并且在通路中已包含所有与该顶点相关联的边时，就会出现这种情况。例如，在图 5 的图 G 中，从 a 开始且连续地选择边 $\{a, f\}$、$\{f, c\}$、$\{c, b\}$ 和 $\{b, a\}$。</p>  <p>这样通路必定会结束。 如果所有边都用完了，说明通路已经构造好了。</p>
开发环境	Win10 Microsoft Visual Studio Community 2017 15.9.3 Debug x86
调试分析	<p>(运行结果截图)</p> 
心得体会	<p>(对整个实验过程做出总结，对重要的算法做出性能分析。)</p> <p>DFS 属实很慢。 本身这个算法的时间复杂度，并不由算法本身决定。 我使用邻接矩阵表示，而使用邻接矩阵时，DFS 的时间复杂度为 $O(N)$。</p> <p>这道题对我最大的影响就是让我体会到了 DFS 和 BFS（虽然这道题没用到）的强大。 很多看似与遍历算法完全无关的问题，其实都可以用这两个算法得出。</p>

	<p>比如，地图两点之间最短距离问题。那就是 BFS 利用树层序遍历的思想得出的。 （当然 DFS 也可以做到想没头苍蝇一样乱走） 比如，八皇后问题，马踏棋盘问题……</p> <p>生活中也是，DFS 和 BFS 的可以应用在方方面面。</p> <p>另外，建立图的思想我觉得很重要。 说起图，那和邻接表和邻接矩阵脱不开联系。 邻接矩阵在各个方面都有很好的作用。</p> <p>邻接矩阵不是可以表示两个节点之间的关系吗。 既然与关系又有了联系，那么肯定可以应用到关系型数据库当中。 于是数据库设计当中的寻找 Primary Key，在不包含联合主键的情况下，就有了更加普遍的算法。 既然与关系有关，那么邻接矩阵也可以求解偏序关系有关的问题。既然可以解决偏序关系，那么和词法解析或许也可以找到联系。</p> <p>生活当中，图的拓扑排序可以为我解决很多方面的问题。图的邻接矩阵表示衍生出来的分析法可以让我在处理问题的决策上更加具有智慧……</p> <p>我觉得图真的挺有意思的。图和其他数据结构一样，是一种思想。不过在我看来，图是一种更有韵味、更有能力的数据结构。</p>
--	--