# 4基于邻接表结构的图实现

## 4.1 问题描述

一个图由顶点集V和边集E组成，每条边是一个点对（v，w），其中v，w属于V，如果点对是有序的，则图称为有向图，反之称为无向图。邻接表是对于每一个顶点，都有一个表保存所有邻接的顶点的一种图的表示方法。本实验要求封装一个基于邻接表结构的图ADT模块，提供图ADT基本的、常见的13种操作，并且为该图ADT提供文件的I/O方法。通过实验达到加深对图的概念、基本运算的理解、熟练掌握图的逻辑结构与物理结构的关系、以邻接表作为物理结构和熟练掌握图基本运算的实现。

## 4.2 模块设计

### 4.2.1 总体架构

根据实验的要求，给出整个模块的总体架构：

模块包括一个图实现文件（.cpp）

模块的宏定义define、结构定义、结构声明、函数指针以及相应的别名typedef、函数声明被包括在实现文件的首部

模块所声明的函数的函数体以及文件操作实现函数在实现文件的尾部

模块的ADT操作实现包括在实现文件中部

### 4.2.2 数据结构设计

数据结构的定义

根据实验的要求以及图ADT的定义，给出基于C++语言实现的图ADT的结构定义和别名：

typedef struct Graph {

struct ElemType \*elemtype;

int number;

} Graph, \*Graphptr;

其中elemtypes是一个ElemType结构类型的指针，指向顶点数组的首结点。number是一个整型，用来存储图中顶点数目

然后给出顶点ElemType的结构定义和别名：

typedef struct ElemType {

char id[MAXCHARLENGTH];

char name[MAXCHARLENGTH];

char birth[MAXCHARLENGTH];

ElemNext \*next\_elem;

} ElemType;

其中，id、name以及birth都是一个长度为MAXCHARLENGTH的字符类型数组，用于保存学生的信息。next\_elem是一个ElemNext类型的指针，用于指向顶点的邻接顶点。顶点保存的便是一个学生的联系方式。

最后给出邻接点的结构定义和别名：

typedef struct ElemNext {

ElemNext \*next\_elem;

int current;

} ElemNext;

其中，next\_elem是一个ElemNext类型的指针，用于指向顶点的邻接顶点。current是一个整型，用于保存邻接顶点的数组索引。

为了便于确定函数的返回状态，采用了如下的typedef：

typedef int Status;

Status是一个整型，用于表示函数的执行状态。

宏定义

#define MAXCHARLENGTH 20 表示每个字符数组的最大长度

#define OK 1 表示函数执行目标功能

#define FALSE 0 表示函数未执行目标功能

#define MAX 100 表示循环队列的最大空间

### 4.2.3 ADT操作设计

根据实验的要求以及图ADT的定义，该图ADT应包括如下的操作：

CreateGraph（创建图），DestroyGraph（销毁图），LocateVex（返回顶点位置信息），GetVex（得到顶点信息），PutVex（设置顶点信息），FirstAdjVex（求顶点的第一邻接点），NextAdjVex（求顶点相对某顶点的下一邻接点），InsertVex（插入顶点），DeleteVex（删除顶点），InsertArc（插入弧），DeleteArc（删除弧），DFSTraverse（深度优先遍历），BFSTraverse（广度优先遍历）。

为了模块的扩展性，DFSTraverse内部函数所用到的访问函数VisitFunc由外部提供，使用函数指针作为参数，在C++语言中的定义如下：

Status (\*VisitFunc)(ElemType \*e);

基于以上的定义，给出所有ADT操作在C++语言中对应得函数原型：

Status CreateVex(ElemType \*\*pElemType,Graphptr \*G);

Status CreateArc(ElemNext \*\*pElemNext,int number,ElemType \*elemType);

Status CreateGraph(Graphptr \*G,ElemType \*V,ElemNext \*VR);

Status DestroyGraph(Graphptr \*G);

Status LocateVex(Graphptr G, ElemType example);

Status GetVex(Graphptr G, ElemType example);

Status PutVex(Graphptr \*G, ElemType example, ElemType value);

Status FirstAdjVex(Graphptr G, ElemType example);

Status NextAdjVex(Graphptr G, ElemType example, ElemType w);

Status InsertVex(Graphptr \*G, ElemType value);

Status DeleteVex(Graphptr \*G, ElemType example);

Status InsertArc(Graphptr \*G, ElemType example, ElemType example\_to);

Status DeleteArc(Graphptr \*G, ElemType example, ElemType example\_to);

void DFS(Graphptr G, int v);

Status DFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e));

Status BFSTraverse(Graphptr G, Status(\*Visit)(ElemType \*e));

Status VisitElem(ElemType \*e);

### 4.2.4 文件存储设计

根据实验的要求以及图ADT的数据结构，该图ADT应包括如下的文件操作：

WriteToFile（将图写入文件）、ReadFromFile（从文件中读取图）

根据ADT操作设计中的定义，给出文件存储操作在C++语言中对应的函数声明：

Status WriteToFile(Graphptr G);

Status ReadFromFile(Graphptr \*G);

### 4.2.5 配套演示系统

配套演示系统采用文本菜单界面，该系统每次从输入流接受一个数字，并且根据数字的不同执行不同的操作。

## 4.3 模块实现

### 4.3.1 开发环境

开发本系统时的操作环境为Windows 10

编程环境

IDE：Clion

Toolchains：

Environment：MinGW version：5.0

C Compiler：mingw32-gcc

C++ Compiler：mingw32-g++

### 4.3.2 ADT操作实现

CreateVex：

该操作接受一个ElemType \*指针pElemType和一个Graphptr 指针，用来创建一个顶点集

该操作首先接受用户创建顶点的数目，并检查用户输入的数目是否合法。然后根据用户所要输入顶点的数目为所有的顶点分配一块连续的空间，并将顶点的数目保存在图中

该操作依次读取用户输入的顶点信息并且判断当前输入顶点在顶点集中是否已经存在。然后根据是否存在执行相应的结果

该操作的时间复杂度O(n3)，空间复杂度O(n)。

CreateArc：

该操作接受ElemNext \*指针\*pElemNext、一个整型 number和一个ElemType指针elemType

该操作首先根据用户顶点的数目创建一块连续的空间，然后每个顶点的索引对应空间中next\_elem指向其顶点的邻接顶点，每当为顶点创建邻接关系时，存储邻接顶点的空间都是单独分配，然后再用头插法插入到邻接链表中。创建邻接关系时，首先确定该邻接关系是否已经存在。然后根据是否存在执行相应的结果

该操作的时间复杂度以及空间复杂度需要根据创建关系的数目来确定。

CreateGraph：

该操作接收一个Graphptr 指针G、一个ElemType 指针V和一个ElemNext 指针VR，用所有的参数来构成一张图。

该操作将图中的指针elemtype指向顶点数组的首顶点，然后再将每个顶点中指针next\_elem指向其对应的邻接链表的首结点，然后创建图成功

该操作的时间复杂度O(1)，空间复杂度O(1)。

DestroyGraph：

该操作接收一个Graphptr指针G，将指针所指向的图销毁。

该操作首先检查指针以及图是否存在，非空且存在的话则会先按照循环依次释放所有的弧头结点所动态分配的空间，再循环依次释放所有顶点所动态分配的空间。最后将图的数目设置为0

该操作的时间复杂度为O(n+e)，空间复杂度为O(1)。

LocateVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则打印顶点相应的序号，否则告知用户顶点不存在

该操作的时间复杂度为O(n)，空间复杂度为O(1)。

GetVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则打印顶点所有的信息，否则告知用户顶点不存在

该操作的时间复杂服为O(n)，空间复杂度为O(1)。

PutVex：

该操作接收一个Graphptr指针G、一个ElemType结构example和一个ElemType结构value

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则打印顶点所有的信息并且将修改后的顶点信息赋给顶点，然后打印出来

该操作的时间复杂服为O(n)，空间复杂度为O(1)。

FirstAdjVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则检查改顶点的第一个邻接顶点是否存在，若存在，则打印第一个邻接顶点的信息

该操作的时间复杂度是O(n)，空间复杂度是O(1)。

NextAdjVex：

该操作接收一个Graphptr指针G、一个ElemType结构example和一个ElemType结构example\_to

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，如果存在顶点与传入的顶点相同，则遍历顶点的所有邻接顶点，然后检查用户输入的邻接顶点在顶点的所有邻接顶点是否存在，若存在，则检查该邻接顶点是否存在后继邻接顶点；若存在，则打印后继邻接顶点的信息

该操作的时间复杂度是O(n+e)，空间复杂度是O(1)。

InsertVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，检查是否用户输入的顶点在图中已经存在，若不存在，创建一块连续的空间，空间的大小比原顶点数组所占的空间多一个顶点的空间，然后将原顶点数组的所有信息(包括每个顶点的邻接顶点的指针)复制到新的顶点数组中，最后将新顶点插入到新顶点数组的尾部

该操作的时间复杂度是O(n)，空间复杂度是O(n)。

DeleteVex：

该操作接收一个Graphptr指针G和一个ElemType结构example

该操作首先检查图是否存在，存在的话则循环遍历图的所有顶点，检查是否用户输入的顶点在图中已经存在，若存在，创建一块连续的空间，空间的大小比原顶点数组所占的空间少一个顶点的空间，然后删除每个顶点中所有指向被删除顶点有关的弧关系并且将每个顶点的所有邻接顶点中索引值大于被删除顶点的索引值减1，接着将原顶点数组除被删除顶点以外的所有顶点的所有信息(包括每个顶点的邻接顶点的指针)复制到新的顶点数组中

该操作的时间复杂度是O(n)，空间复杂度是O(n)。

InsertArc：

该操作接收一个Graphptr 指针G,、ElemType结构example和一个ElemType结构example\_to

该操作首先判断图是否为空，然后遍历所有顶点，检查是否存在用户输入的弧尾顶点和弧头顶点；若都存在，遍历弧尾结点的所有邻接顶点，检查是否存在邻接顶点与弧头顶点相同；若不存在，则用头插法将弧头顶点插入到弧尾顶点的邻接顶点中

该操作的时间复杂度是O(n+e)，空间复杂度是O(1)。

DeleteArc：

该操作接收一个Graphptr 指针G、ElemType结构example和一个ElemType结构example\_to

该操作首先判断图是否为空，然后遍历所有顶点，检查是否存在用户输入的弧尾顶点和弧头顶点；若都存在，遍历弧尾结点的所有邻接顶点，检查是否存在邻接顶点与弧头顶点相同；若存在，则将弧头顶点从弧尾顶点的邻接顶点中删除

该操作的时间复杂度是O(n+e)，空间复杂度是O(1)。

DFS：

该操作接收一个Graphptr 指针G和一个整型v

该操作首先判断索引值为v的顶点是否访问过；若没有访问过，则改变顶点的访问状态，并且访问索引值为v的顶点，然后对其邻接顶点调用递归

该操作的时间复杂度是O(n+e)，空间复杂度是O(n)。

DFSTraverse：

该操作接收一个Graphptr 指针G和一个Visit函数指针，深度优先遍历全图

该操作首先判断图是否为空，如果存在且非空则首先访问标志数组visited并置0，然后循环遍历该图的全部顶点，对没有访问过的顶点调用DFS函数遍历全图

该操作的时间复杂度是O(n)，空间复杂度是O(1)。

BFSTraverse：

该操作接收一个Graphptr 指针G和一个Visit函数指针，广度优先遍历全图

该操作首先判断图是否为空，如果存在且非空则首先访问标志数组visited并置0，然后初始化一个队列并且循环遍历图的所有顶点，在每一次循环中先判断该顶点是否被访问过，若没有被访问过则访问并将其入队，然后以队列不为空为条件循环，首先将元素出队，遍历其所有邻接顶点，若未被访问则访问并且将其入队，依此循环。

该操作的时间复杂度是O(n+e)，空间复杂度是O(n)。

### 4.3.3 文件存储实现

WriteToFile：

该操作接收一个Graphptr指针G实现将图存储进文件的功能

该操作首先检查图是否存在以及文件是否能够被打开／创建，如果存在且文件可以正常打开，则依次将图的顶点数，每个顶点的信息以及它所有邻接顶点信息写入文件中，最后关闭文件

该操作的时间复杂度是O(n+e)，空间复杂度是O(n+e)。

ReadFromFile：

该操作接收一个Graphptr指针G实现从文件读取图的功能

该操作首先检查图是否存在以及文件是否能够被打开，如果图不存在且文件可以正常打开，则创建图并且依次将文件中存储的顶点数，每个顶点的信息以及它所有邻接顶点信息写入图中，最后关闭文件

该操作的时间复杂度是O(n+e)，空间复杂度是O(n+e)。

### 4.3.4 配套演示系统实现

外部函数实现

VisitElem：

该操作接收一个ElemType结构指针，将指针指向的顶点信息输出。

在该实验的演示系统中，该操作将ElemType数据结构的三个成员变量id，name，birth的值依次打印输出。

演示功能实现

该系统采用简单文本界面，实现的过程相对而言简单——首先输出系统菜单，然后从输入流中读入一个数字，在根据数字所对应的功能执行相应的代码即可。

## 4.4 模块测试

### 4.4.1 配套演示系统操作介绍

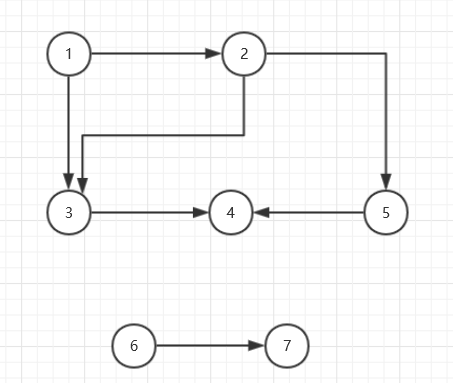
演示系统的界面如下图所示：

该演示系统操作简单，输入对应的数字并按下回车即可，输入1-17表明运行对应的功能，且相应操作的结果会被输出到屏幕上，输入0退出系统。

### 4.4.2 测试方案

测试数据应兼顾包括有效数据无效数据两个方面，有效数据旨在验证系统功能的正确性，而无效甚至是错误的数据旨在验证系统的鲁棒性。根据以上的要求，拟定了以下一个测试方案：

1. 在未创建图的情况下执行销毁图，插入顶点、深度优先遍历
2. 创建一个有7个顶点的有向图，有7条边

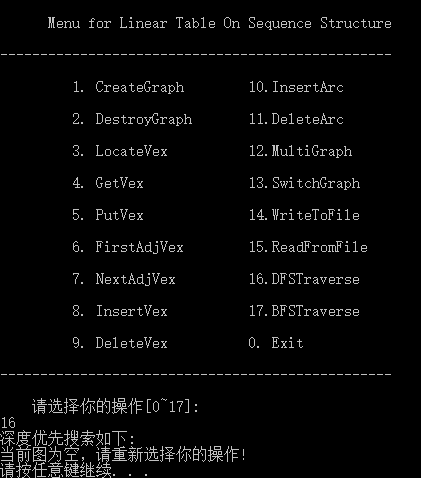


1. 确定ID为5的顶点的位置信息（比较是否正确）
2. 确定ID为2的第一和第二邻接顶点
3. 取得ID为1的顶点的值
4. 修改ID为1的顶点的值
5. 插入一个新的顶点8和一条新的弧（8，6）
6. 广度优先遍历该图
7. 保存该图
8. 关闭并重新打开后读取该图
9. 深度优先遍历该图
10. 删除下标为6的顶点
11. 广度优先遍历该图

测试创建图的时候输入为：1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6 7 7 7 1 1 2 3 \* 2 3 5 \* 3 4 \* 5 4 \* 6 7 \* #

### 4.4.3 测试结果

1. 在未创建图的情况下执行销毁图、插入顶点、深度优先遍历



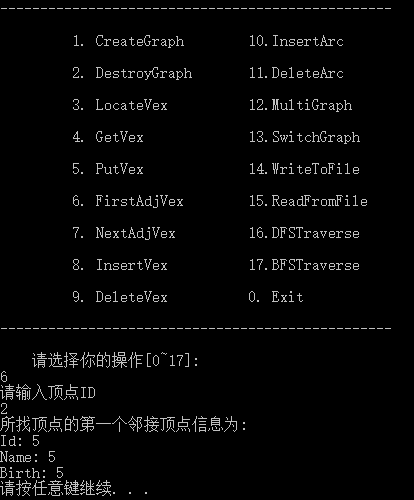
1. 创建一个有7个顶点的有向图，有7条边



1. 确定ID为5的顶点的位置信息（比较是否正确）

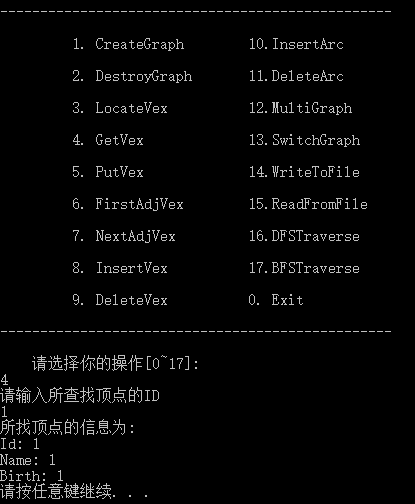


1. 确定ID为2的第一和第二邻接顶点

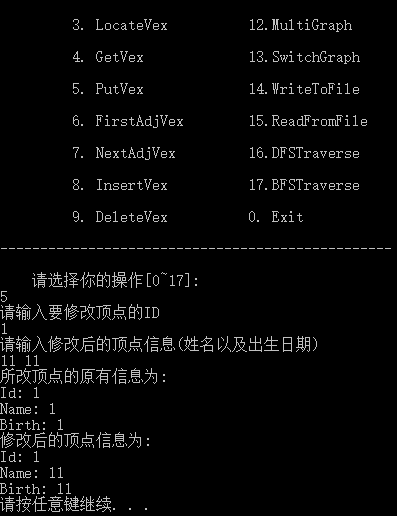


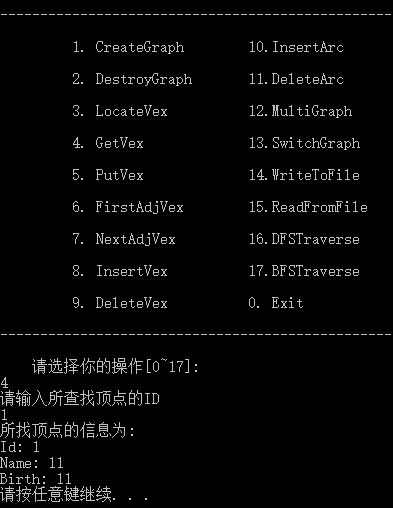


1. 取得ID为1的顶点的值

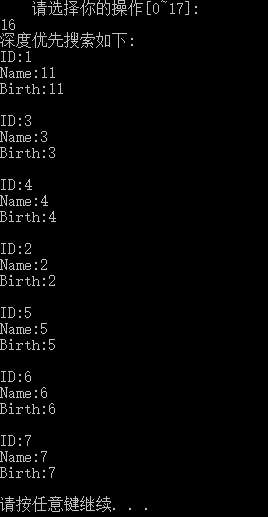


1. 修改ID为1的顶点的值



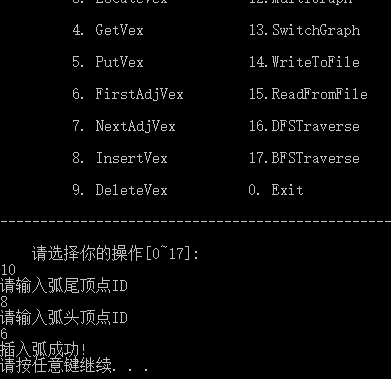


1. 深度优先遍历该图

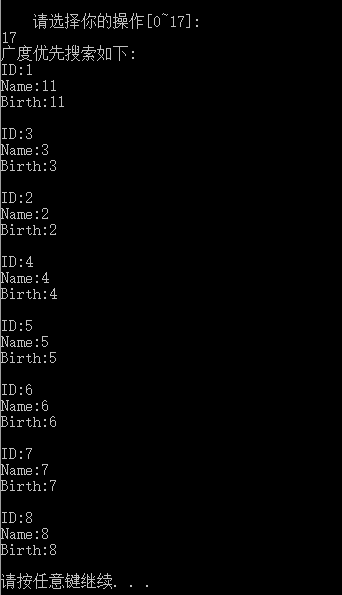


1. 插入一个新的顶点8和一条新的弧（8，6）

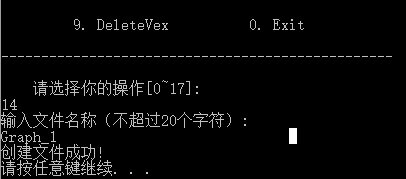




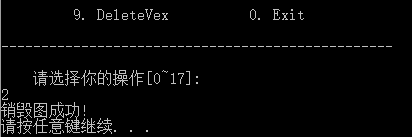
1. 广度优先遍历该图

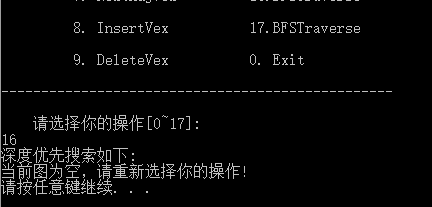


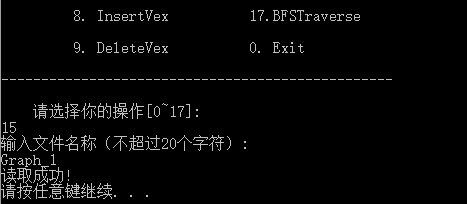
1. 保存该图



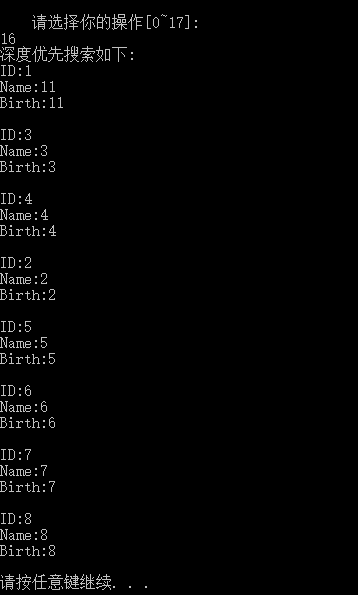
1. 销毁图后先深度优先遍历，然后重新读取该图







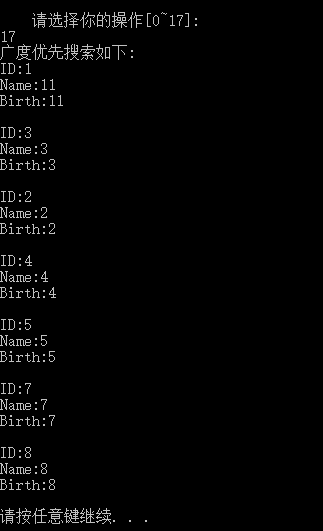
1. 深度优先遍历该图



1. 删除下标为6的顶点



1. 广度优先遍历该图



## 4.5 实验小结

通过本次实验，我加深了对于基于邻接表的图的理解,熟悉了如何用邻接表表示图，并且对图的各种操作加深了印象，更好地理解了图这种数据结构。

通过了本次实验，我收获颇丰，但是也仍然存在一些不足。在实验的过程中也有许多没有考虑到的地方，例如：在删除顶点的时候，一开始只考虑到删除所有与顶点有关的弧关系，但没有考虑到部分邻接顶点的索引值也要相应地改变，所以导致了程序出现了异常。