1 附录 D 基于邻接表图实现的源程序

/* Gragh on LinkTable Sturcture */

main.cpp

```
#include "def.h"
#include "SingleGraph.h"
#include "Manager.h"

int main() {
    int state;
    int op;
    int index;
    int key, key1, key2;
    char filename[30];
    VertexType v;
    Manager M;
    M.MenuDisp();
    cout << "enter your command:" << endl;
    cin >> op;
    while (op){
        switch(op){
            case 1: // 创建新表
            state = M.NewGraph();
            if (state == OK) {
                cout << "create successfully. The structure is:" <<
                endl;
                M.DispStruc();
            }
            else cout << "wrong data set!" << endl;
            break;
            case 2: // 删除一张图
            if ((index = M.GetCommand1()) != -1){
                M.DelGraph(index);
                cout << "delete successfully! The structure is:" <<
                endl;
                M.DispStruc();
                break;
            }
            case 3:
            M.DispStruc();
```

```
break;
case 4: // 创建图
if ((index = M.GetCommand2()) != -1){
    VertexType V[MAX_VERTEX_NUM];
    KeyType VR[100][2];
    ReadData(V, VR);
    state = CreateGraph(M.elem[index], V, VR);
    M.IfDataSetError(state);
}
break;
case 5: // 销毁图
if ((index = M.GetCommand1()) != -1){
    DestroyGraph(M.elem[index]);
    cout << "successfully." << endl;
}
break;
case 6: // 定位节点
if ((index = M.GetCommand1()) != -1){
    cout << "enter the key:" << endl;
    cin >> key;
    state = LocateVex(M.elem[index], key);
    if (state == -1){
        cout << "can't find vertex!" << endl;
    }
    else{
        cout << "the index and name is: " << state << "," \
        << M.elem[index].vertices[state].data.others <<
        endl;
    }
}
break;
case 7: // 修改节点值
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
    M.GetVertexValue(v);
    state = PutVex(M.elem[index], key, v);
    M.IfDataSetError(state);
}
break;
case 8: // 获取第一个邻接点
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
    state = FirstAdjVex(M.elem[index], key);
    if (state == -1){
        cout << "without firstadjvex!" << endl;
    }
    else{
        cout << "the first adjvex is: " <<
        M.elem[index].vertices[state].data.key << "," \
        << M.elem[index].vertices[state].data.others <<
        endl;
    }
}
break;
case 9: // 返回v的邻接点w的下一节点的位序
if ((index = M.GetCommand1()) != -1){
    cout << "node v: ";
    M.GetKey(key1);
    cout << "node w: ";
    M.GetKey(key2);
    state = NextAdjVex(M.elem[index], key1, key2);
    if (state == -1){
        cout << "without firstadjvex!" << endl;
    }
    else{
        cout << "the next adjvex is:" <<
        M.elem[index].vertices[state].data.key << "," \
        << M.elem[index].vertices[state].data.others <<
        endl;
    }
}
break;
case 10: // 插入顶点
if ((index = M.GetCommand1()) != -1){
    M.GetVertexValue(v);
    state = InsertVex(M.elem[index], v);
    M.IfDataSetError(state);
}
break;
case 11: // 删除节点
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
    state = DeleteVex(M.elem[index], key);
    M.IfDataSetError(state);
}
break;
case 12: // 插入弧
if ((index = M.GetCommand1()) != -1){
    cout << "node v: ";
    M.GetKey(key1);
    cout << "node w: ";
    M.GetKey(key2);
    state = InsertArc(M.elem[index], key1, key2);
    M.IfDataSetError(state);
}
break;
case 13: // 删除弧
if ((index = M.GetCommand1()) != -1){
    cout << "node v: ";
    M.GetKey(key1);
    cout << "node w: ";
    M.GetKey(key2);
    state = DeleteArc(M.elem[index], key1, key2);
    M.IfDataSetError(state);
}
break;
case 14: // DFS
if ((index = M.GetCommand1()) != -1){
    DFSTraverse(M.elem[index], visit);
}
break;
case 15: // BFS
if ((index = M.GetCommand1()) != -1){
    BFSTraverse(M.elem[index], visit);
}
break;
case 16: // 保存文件
if ((index = M.GetCommand1()) != -1){
    cout << "enter the filename:" << endl;
    cin >> filename;
    state = SaveGraph(M.elem[index], filename);
    M.IfIoError(state);
}
```

```
break;
case 17: // 加载文件
if ((index = M.GetCommand2()) != -1){
    cout << "enter the filename:" << endl;
    cin >> filename;
    state = LoadGraph(M.elem[index], filename);
    M.IfIoError(state);
}
break;
case 18: // 距离小于k的顶点集合
if ((index = M.GetCommand1()) != -1){
    M.GetKey(key);
    cout << "enter k:" <<endl;
    int k;
    cin >> k;
    vector<int> LessThanK =
    VerticesSetLessThanK(M.elem[index], key, k);
    cout << "vertices are:" << endl;
    for (int i : LessThanK){
        cout << M.elem[index].vertices[i].data.key << "," \
        << M.elem[index].vertices[i].data.others << "  ";
    }
}
break;
case 19: // 最短距离
if ((index = M.GetCommand1()) != -1){
    cout << "enter v:";
    M.GetKey(key1);
    cout << "enter w:";
    M.GetKey(key2);
    cout << "dist is: " <<
    ShortestPathLength(M.elem[index], key1, key2);
}
break;
case 20: // 连通分量数
if ((index = M.GetCommand1()) != -1){
    cout << "the num is:" <<
    ConnectedComponentsNums(M.elem[index]);
}
break;
default:
```

```
            cout << "wrong command!" << endl;
            break;

        }
        cout << endl;
        cout << endl;
        M.MenuDisp();
        cout << "enter your command:" << endl;
        cin >> op;

    }
    cout << "bye." << endl;

    return 0;
}
```

def.h

```
#pragma once
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <string>
#include <map>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAX_VERTEX_NUM 20
typedef int status;
typedef int KeyType;
typedef enum {DG,DN,UDG,UDN} GraphKind;
typedef struct {
    KeyType key;
    char others[20];
} VertexType; //顶点类型定义


typedef struct ArcNode { //表结点类型定义
    int adjvex; //顶点位置编号
    struct ArcNode *nextarc; //下一个表结点指针
    ArcNode() : nextarc(nullptr){}
} ArcNode;
typedef struct VNode{    //头结点及其数组类型定义
    VertexType data;  //顶点信息
    ArcNode *firstarc;  //指向第一条弧
    VNode() : firstarc(nullptr){}
} VNode,AdjList[MAX_VERTEX_NUM];
typedef struct LGragh{ //邻接表的类型定义
    string name;
    AdjList vertices;  //头结点数组
    int vexnum,arcnum;  //顶点数、弧数
    GraphKind kind; //图的类型
    LGragh() : vexnum(0), arcnum(0), kind(UDG), name("unnamed"){}
} ALGraph;
```

SingleGraph.h

```
#pragma once
#include "def.h"

// 读取数据
void ReadData(VertexType V[],KeyType VR[][2]);

// 创建图
status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]);
int find_index(const ALGraph &G, int key); //
查找key对应顶点的下标-------------通用函数
status AddVertex(ALGraph &G, int key1, int key2); // 添加边

// 销毁图
status DestroyGraph(ALGraph &G);

// 根据u在图G中查找顶点，查找成功返回位序，否则返回-1;
int LocateVex(const ALGraph &G,KeyType u);
```

```
//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK;
//如果查找失败或关键字不唯一，返回ERROR
status PutVex(ALGraph &G,KeyType u,VertexType value);

//根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1
int FirstAdjVex(const ALGraph &G,KeyType u);

//根据u在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位
序，查找失败返回-1
int NextAdjVex(const ALGraph &G,KeyType v,KeyType w);

//在图G中插入顶点v，成功返回OK,否则返回ERROR
status InsertVex(ALGraph &G,VertexType v);

//在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR
status DeleteVex(ALGraph &G,KeyType v);
int GetIndex(const ALGraph &G, KeyType v); //
和find_index用法相同-------------通用

//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
status InsertArc(ALGraph &G,KeyType v,KeyType w);
int IfConflict(const ALGraph &G, KeyType v, KeyType w); // 防止同一条弧
被插两次

//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR
status DeleteArc(ALGraph &G,KeyType v,KeyType w);

// dfs
status DFSTraverse(ALGraph &G,void (*visit)(VertexType));
void dfs(const ALGraph &G, int index, void (*visit)(VertexType), int
ifvisit[]);

//bfs
status BFSTraverse(ALGraph &G,void (*visit)(VertexType));
void visit(VertexType v); //-------------------------------------------
通用

//保存文件
status SaveGraph(ALGraph G, char FileName[]);

//加载文件
```

```
status LoadGraph(ALGraph &G, char FileName[]);

//与v距离小于k的顶点
vector<int> VerticesSetLessThanK(const ALGraph &G, KeyType v,int k);
vector<int> GetDist(const ALGraph &G, KeyType v); // 辅助函数: 返回各顶
点和v的距离----------通用

// v和w之间的最短距离
int ShortestPathLength(const ALGraph &G, KeyType v, KeyType w);

// 返回连通分量个数
int ConnectedComponentsNums(const ALGraph &G);
void dfs0(const ALGraph &G, int index, vector<int> &ifvisit); // 辅助函
数
```

SingleGraph.cpp

```
#include "def.h"
#include "SingleGraph.h"

void ReadData(VertexType V[],KeyType VR[][2]){
    // 获取V数组元素
    int key;
    char others[30];
    int i = 0;
    cout << "enter the key-value pair end with \"-1 nil\"" << endl;
    cin >> key >> others;
    while (key != -1){
        V[i].key = key;
        strcpy(V[i].others, others);
        i++;
        cin >> key >> others;
    }
    V[i].key = key;
    strcpy(V[i].others, others);
    cout << "vertex read successfully. Enter the edge end with \"-1
      -1\"" << endl;
    // 获取VR数组元素
    int key1, key2;
    i = 0;
    cin >> key1 >> key2;
    while (key1 != -1){
        VR[i][0] = key1;
        VR[i][1] = key2;
        i++;
        cin >> key1 >> key2;
    }
    VR[i][0] = key1;
    VR[i][1] = key2;
    cout << "read successfully!" << endl;
}


status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])
{
    //判断keyの唯一性
    int hash[100] = {0};
    int i = 0;
    while (V[i].key != -1){
        if (hash[V[i].key] == 1){
            return ERROR;
        }
        hash[V[i].key] = 1;
        i++;
    }
    //图的操作
    // 添加顶点
    i = 0;
    G.vexnum = 0;
    G.arcnum = 0;
    G.kind = UDG;
    while (i < 20 && V[i].key != -1){
        G.vertices[i].data = V[i];
        G.vertices[i].firstarc = nullptr;
        G.vexnum++;
        i++;
    }
    if (i == 20 && V[i].key != -1) return ERROR; // 加入的顶点数不能溢出
    // 添加边
    i = 0;
    while (VR[i][0] != -1){
        int state = AddVertex(G, VR[i][0], VR[i][1]);
        if (state == ERROR) return ERROR;
        i++;
    }
    return OK;
}

status AddVertex(ALGraph &G, int key1, int key2) {
    int index1 = find_index(G, key1);
    if (index1 == -1) return ERROR;
    int index2 = find_index(G, key2);
    if (index2 == -1) return ERROR;

    VNode &Head1 = G.vertices[index1];
    ArcNode *NewNode1 = (ArcNode *) malloc(sizeof(struct ArcNode));
    NewNode1->adjvex = index2;
    NewNode1->nextarc = Head1.firstarc;
    Head1.firstarc = NewNode1;

    VNode &Head2 = G.vertices[index2];
    ArcNode *NewNode2 = (ArcNode *) malloc(sizeof(struct ArcNode));
    NewNode2->adjvex = index1;
    NewNode2->nextarc = Head2.firstarc;
    Head2.firstarc = NewNode2;

    G.arcnum++;
    return OK;
}

int find_index(const ALGraph &G, int key){
    for (int i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == key) return i;
    }
    return -1;
}



status DestroyGraph(ALGraph &G){
    for (int i = 0; i < G.vexnum; i++){
        ArcNode* p = G.vertices[i].firstarc;
        ArcNode* pre;
        while (p){
            pre = p;
            p = p->nextarc;
            free(pre);
        }
        G.vertices[i].firstarc = nullptr;
    }
    G.vexnum = 0;
    G.arcnum = 0;
    return OK;
}


int LocateVex(const ALGraph &G,KeyType u)
{
    for (int i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == u){
            return i;
        }
    }
    return -1;
}


status PutVex(ALGraph &G,KeyType u,VertexType value)
{
    int hash[100] = {0};
    int i;
    for (i = 0; i < G.vexnum; i++){
        hash[G.vertices[i].data.key] = 1;
    }
    if (hash[u] == 0 || u != value.key && hash[value.key] == 1) return
    ERROR;
    for (i = 0; i < G.vexnum; i++){
        if (u == G.vertices[i].data.key){
            G.vertices[i].data = value;
        }
    }
    return OK;
}


int FirstAdjVex(const ALGraph &G,KeyType u)
```

```
{
    int i;
    for (i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == u) break;
    }
    if (i == G.vexnum) return -1;
    if (G.vertices[i].firstarc->adjvex) return
    G.vertices[i].firstarc->adjvex;
    else return -1;
}


int NextAdjVex(const ALGraph &G,KeyType v,KeyType w)
{
    int i;
    for (i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == v) break;
    }
    if (i == G.vexnum) return -1;
    ArcNode* p = G.vertices[i].firstarc;
    while (p){
        if (G.vertices[p->adjvex].data.key == w){
            if (p->nextarc) return p->nextarc->adjvex;
            else return -1;
        }
        p = p->nextarc;
    }
    return -1;
}


status InsertVex(ALGraph &G,VertexType v)
{
    int hash[100] = {0};
    int i;
    for (i = 0; i < G.vexnum; i++){
        hash[G.vertices[i].data.key] = 1;
    }
    if (hash[v.key] == 1) return ERROR;
    if (G.vexnum == 20) return ERROR;
    G.vertices[G.vexnum++].data = v;
    G.vertices[G.vexnum].firstarc = nullptr;
    return OK;
}


status DeleteVex(ALGraph &G,KeyType v)
{
    int index = GetIndex(G, v);
    if (index == -1) return ERROR;
    if (G.vexnum == 1) return ERROR;
    // 删除节点
    ArcNode* p = G.vertices[index].firstarc;
    // 防止p本来就是null
    if (p != nullptr){
        ArcNode* next = p->nextarc;
        while (next){
            free(p);
            p = next;
            next = next->nextarc;
        }
        free(p);
        G.vertices[index].firstarc = nullptr;
    }
    for (int i = index; i < G.vexnum-1; i++){
        G.vertices[i] = G.vertices[i+1];
    }
    G.vexnum--;
    // 删除边
    for (int i = 0; i < G.vexnum; i++){
        p = G.vertices[i].firstarc;
        // 这里分类稍微麻烦，要看看第一个节点是否就是被删的节点
        if (p && p->adjvex == index) {
            G.vertices[i].firstarc = p->nextarc;
            G.arcnum--;
            free(p);
        }
        else{
            while (p && p->nextarc){
                if (p->nextarc->adjvex == index){
                    ArcNode* bin = p->nextarc;
                    p->nextarc = p->nextarc->nextarc;
                    free(bin);
                    G.arcnum--;
                    break;
                }
                p = p->nextarc;
            }
        }
    }
    // 调整删除节点后adjvex的值
    for (int i = 0; i < G.vexnum; i++){
        p = G.vertices[i].firstarc;
        while(p){
            if (p->adjvex > index) p->adjvex--;
            p = p->nextarc;
        }
    }
    return OK;
}

int GetIndex(const ALGraph &G, KeyType v) {
    for (int i = 0; i < G.vexnum; i++) {
        if (G.vertices[i].data.key == v) return i;
    }
    return -1;
}


status InsertArc(ALGraph &G,KeyType v,KeyType w)
{
    int index1 = GetIndex(G, v);
    int index2 = GetIndex(G, w);
    if (index1 == -1 || index2 == -1) return ERROR;
    if (IfConflict(G, v, w)) return ERROR;

    ArcNode* p = G.vertices[index1].firstarc;
    ArcNode* newnode = (ArcNode*)malloc(sizeof(struct ArcNode));
    newnode->adjvex = index2;
    newnode->nextarc = p;
    G.vertices[index1].firstarc = newnode;
    G.arcnum++;
    ArcNode* p0 = G.vertices[index2].firstarc;
    ArcNode* newnode1 = (ArcNode*)malloc(sizeof(struct ArcNode));
    newnode1->adjvex = index1;
    newnode1->nextarc = p0;
    G.vertices[index2].firstarc = newnode1;

    return OK;
}

int IfConflict(const ALGraph &G, KeyType v, KeyType w){
    for (int i = 0; i < G.vexnum; i++){
        if (G.vertices[i].data.key == v){
            ArcNode* p = G.vertices[i].firstarc;
            int target = GetIndex(G, w);
            while (p){
                if (p->adjvex == target) return 1;
                p = p->nextarc;
            }
        }
    }
    return 0;
}


status DeleteArc(ALGraph &G,KeyType v,KeyType w)
{
    int index1 = GetIndex(G, v);
    int index2 = GetIndex(G, w);
    if (index1 == -1 || index2 == -1) return ERROR;
    int flag = 0;
    // 删弧
    ArcNode* p1 = G.vertices[index1].firstarc;
    if (p1 && p1->adjvex == index2) {
        flag = 1;
        G.vertices[index1].firstarc = p1->nextarc;
        G.arcnum--; // 弧计数在这里减去就行，下面不用
        free(p1);
    }
    else{
        while (p1->nextarc){
            if (p1->nextarc->adjvex == index2){
                flag = 1; // 同样的，这里找到弧设置一下就行，下面不用
                ArcNode* bin = p1->nextarc;
                p1->nextarc = p1->nextarc->nextarc;
                free(bin);
                G.arcnum--;
                break;
            }
            p1 = p1->nextarc;
        }
    }
    // 对称情况
    ArcNode* p2 = G.vertices[index2].firstarc;
    if (p2 && p2->adjvex == index1) {
        G.vertices[index2].firstarc = p2->nextarc;
        //G.arcnum--;
        free(p2);
    }
    else{
        while (p2->nextarc){
            if (p2->nextarc->adjvex == index1){
                ArcNode* bin = p2->nextarc;
                p2->nextarc = p2->nextarc->nextarc;
                free(bin);
                //G.arcnum--;
                break;
            }
            p2 = p2->nextarc;
        }
    }

    if (flag == 0) return ERROR;
    return OK;
}


status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
{
    int ifvisit[20] = {0};
    for (int i = 0; i < G.vexnum; i++){
        dfs(G, i, visit, ifvisit);
    }
    return OK;
}

void dfs(const ALGraph &G, int index, void (*visit)(VertexType), int 
ifvisit[]){
    if (ifvisit[index] == 1) return;
    visit(G.vertices[index].data);
    ifvisit[index] = 1;
    ArcNode* p = G.vertices[index].firstarc;
    while (p){
        dfs(G, p->adjvex, visit, ifvisit);
        p = p->nextarc;
    }
}


status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
{
    int q[100];
    int ifvisit[100];
    for (int i = 0; i < G.vexnum; i++){
        ifvisit[i] = 0;
    }
    int head = 0, tail = 0;
    for (int i = 0; i < G.vexnum; i++){
        if (ifvisit[i] == 0){
            q[tail++] = i;
            ifvisit[i] = 1;
        }
        while (head != tail){
            VNode cur = G.vertices[q[head++]];
            visit(cur.data);
            ArcNode* p = cur.firstarc;
            while (p){
                if (ifvisit[p->adjvex] == 0){
                    q[tail++] = p->adjvex;
                    ifvisit[p->adjvex] = 1;
                }
                p = p->nextarc;
            }
        }
    }
    return OK;
}

void visit(VertexType v)
{
    printf(" %d %s",v.key,v.others);
}


status SaveGraph(ALGraph G, char FileName[])
{
    FILE* fp = fopen(FileName, "w");
    if (!fp) return ERROR;
    for (int i = 0; i < G.vexnum; i++){
        fprintf(fp, "%d %s\n", G.vertices[i].data.key,
        %%G.vertices[i].data.others);
    }
    char last[30] = "nil";
    fprintf(fp, "%d %s\n", -1, last);
    for (int i = 0; i < G.vexnum; i++){
        ArcNode* p = G.vertices[i].firstarc;
        while (p){
            fprintf(fp, "%d %d\n", G.vertices[i].data.key,
            %%G.vertices[p->adjvex].data.key);
            p = p->nextarc;
        }
    }
    fprintf(fp, "%d %d\n", -1, -1);
    fclose(fp);
    return OK;
}


status LoadGraph(ALGraph &G, char FileName[])
{
    FILE* fp = fopen(FileName, "r");
    if (!fp) return ERROR;
    int key;
    char others[30];
    fscanf(fp, "%d %s", &key, others);
    while (key != -1){
        G.vertices[G.vexnum].data.key = key;
        strcpy(G.vertices[G.vexnum].data.others, others);
        G.vexnum++;
        fscanf(fp, "%d %s", &key, others);
    }
    KeyType key1, key2;
    fscanf(fp, "%d %d", &key1, &key2);
    while (key1 != -1){
        InsertArc(G, key1, key2);
        fscanf(fp, "%d %d", &key1, &key2);
    }
    fclose(fp);
    return OK;
}


vector<int> VerticesSetLessThanK(const ALGraph &G, KeyType v, int k){
    vector<int> dist = GetDist(G, v);
    // 统计距离小于k的节点
    vector<int> LessThanK;
    for (int i = 0; i < G.vexnum; i++){
        if (dist[i] < k){
            LessThanK.push_back(i);
        }
    }
    return LessThanK;
}

vector<int> GetDist(const ALGraph &G, KeyType v){
    vector<int> ifvisited(G.vexnum, 0);
    vector<int> dist(G.vexnum, INT16_MAX);
    queue<int> q;
    int cur_size;
    int index;
    int cur_dist = 0;
    //bfs过程中记录距离
    int index0 = GetIndex(G, v);
    dist[index0] = 0;
    ifvisited[index0] = 0;
    q.push(index0);
    cur_size = q.size();
    while (!q.empty()){
        for (int i = 0; i < cur_size; i++){
            index = q.front();
            q.pop();
            dist[index] = cur_dist;
            // 加入与该点邻接的点
            ArcNode *p = G.vertices[index].firstarc;
            while (p){
                if (ifvisited[p->adjvex] == 0){
                    q.push(p->adjvex);
                    ifvisited[p->adjvex] = 1;
                }
                p = p->nextarc;
            }
        }
        cur_dist++;
        cur_size = q.size();
    }
    return dist;
}


int ShortestPathLength(const ALGraph &G, KeyType v, KeyType w){
    vector<int> dist = GetDist(G, v);
    return dist[GetIndex(G, w)];
}


int ConnectedComponentsNums(const ALGraph &G){
    vector<int> ifvisited(G.vexnum, 0);
    int num = 0;
    for (int i = 0; i < G.vexnum; i++){
        if (ifvisited[i] == 0){
            num++;
            dfs0(G, i, ifvisited);
        }
    }
    return num;
}
```

```
void dfs0(const ALGraph &G, int index, vector<int> &ifvisit){
    if (ifvisit[index] == 1) return;
    ifvisit[index] = 1;
    ArcNode* p = G.vertices[index].firstarc;
    while (p){
        dfs0(G, p->adjvex, ifvisit);
        p = p->nextarc;
    }
}
```

Manager.h

```
#include "def.h"
#include "SingleGraph.h"
#pragma once

class Manager {
    private:

    public:
    ALGraph elem[10];
    int length;
    int initial_size;
    map<string, int> name_index; // 图名索引

    Manager();

    //展示菜单
    void MenuDisp();

    // 交互判空函数，为空报错
    int GetCommand1();

    // 判不空交互函数，不为空报错
    int GetCommand2();

    // 数据集错误检查
    void IfDataSetError(int state);

    // 判断io错误
    void IfIoError(int state);

    // 获取key
    void GetKey(int &key);

    // 获取一个节点值
    void GetVertexValue(VertexType &v);

    // // 定位节点位序检查，找不到报错
    // void IfFind(int state);

    //创建一张图
    status NewGraph();

    //删除一张图
    status DelGraph(int index);

    //展示树形结构
    void DispStruc();
};
```

Manager.cpp

```
#include "Manager.h"

Manager::Manager() : length(0), initial_size(10){}

void Manager::MenuDisp() {
    cout << "                  Menu" << endl;
    cout << "--------------------------------------------" << endl;
    cout << "  1. NewGraph                  2. DelGraph" << endl;
    cout << "  3. DispStruc                            " << endl;
    cout << "  4. CreateGraph               5. DestroyGraph" << endl;
    cout << "  6. LocateVex                 7. PutVex" << endl;
    cout << "  8. FirstAdjVex               9. NextAdjVex" << endl;
    cout << "  10. InsertVex                11. DeleteVex" << endl;
    cout << "  12. InsertArc                13. DeleteArc" << endl;
    cout << "  14. DFS                      15. BFS" << endl;
    cout << "  16. Save                     17. Load" << endl;
    cout << "  18. VerticesSetLessThanK    19. ShortestPathLength" <<
    endl;
    cout << "  20. ConnectedComponentsNums  0. quit" << endl;
    cout << "--------------------------------------------" << endl;
}
```

```
int Manager::GetCommand1(){
    string name;
    cout << "enter the name of the graph to be operated:" << endl;
    cin >> name;
    map<string, int>::iterator it = this->name_index.find(name);
    if (it == this->name_index.end()) {
        cout << "can't find the graph!" << endl;
        return -1;
    }
    else if (this->elem[it->second].vexnum == 0){
        cout << "empty graph!" << endl;
        return -1;
    }
    else return it->second;
}

int Manager::GetCommand2(){
    string name;
    cout << "enter the name:" << endl;
    cin >> name;
    map<string, int>::iterator it = this->name_index.find(name);
    if (it == this->name_index.end()) {
        cout << "can't find the graph!" << endl;
        return -1;
    }
    else if (this->elem[it->second].vexnum != 0){
        cout << "existed graph!" << endl;
        return -1;
    }
    else return it->second;
}

void Manager::IfDataSetError(int state) {
    if (state == ERROR){
        cout << "data set error!" << endl;
    }
    else{
        cout << "successfully." << endl;
    }
```

```
}

void Manager::IfIoError(int state) {
    if (state == ERROR){
        cout << "IO ERROR!" << endl;
    }
    else{
        cout << "successfully." << endl;
    }
}

void Manager::GetKey(int &key) {
    cout << "enter the key:" << endl;
    cin >> key;
}

void Manager::GetVertexValue(VertexType &v) {
    cout << "enter the key-value pair:" << endl;
    cin >> v.key >> v.others;
}
//void Manager::IfFind(int state) {
    // if (state == -1){
        // cout << "can't find vertex!" << endl;
        // }
    // else{
        // cout << ""
        // }
    //}

status Manager::NewGraph() {
    VertexType V[MAX_VERTEX_NUM];
    KeyType VR[100][2];
    cout << "enter a name for this graph:" << endl;
    cin >> this->elem[this->length].name;
    // 读取数据
    ReadData(V, VR);
    // 创建图
    int state = CreateGraph(this->elem[this->length], V, VR);
    if (state == OK) {
        this->name_index.insert(pair<string
        ,int>(this->elem[this->length].name, this->length));
        this->length++;
    }
    return state;
}

status Manager::DelGraph(int index) {
    return DestroyGraph(this->elem[index]);
}

void Manager::DispStruc() {
    cout << "|-Manager" << endl;
    for (int i = 0; i < this->length; i++){
        if (this->elem[i].vexnum == 0){
            cout << "    |-null" << endl;
        }
        else {
            cout << "    |-" << this->elem[i].name << endl;
        }
    }
}
```