**关于指针、链表串联结点的一些问题**

在创建十字链表中，碰到几个问题。

1. **不能对空指针进行取值的操作**

struct ArcNode \*now;

now = G.node[a-1].firstout;

while(NULL != now)

{

now = now->tlink;

}

**now->tlink = next;** (这句话会导致内存错误Segmentation fault，因为上面退出while循环的条件就是now为空指针。所以退出循环后，此时的now是NULL，对NULL取->tlink肯定会报错啊！所以应该直接对指针now赋值)

now = next;(next是一个ArcNode指针，是新增的一个结点)

1. **链表中结点的串联要注意指针的操作**

错误示例：

struct ArcNode \*now;

now = G.node[a-1].firstout;

while(NULL != now)

{

now = now->tlink;

}

now = next;(next是一个ArcNode指针，是新增的一个结点)

**错误分析：**这种更新链表的方式，到最后，每个顶点的firstout都只会指向第一次输入的那根弧的弧结点，然后不会再指向后面的新增的弧结点。如果这种赋值，虽然当退出while循环后，now的值变成了最后那个结点的tlink，但是 "now=next;" 这句话又改变了now指针的指向，让它指向了新增的结点next，而不是起到给now指向的tlink指针赋值的作用！！！！！ 所以这里如果要给now指向的那个tlink指针赋值(即让这个链表的各结点串联起来)，就必须是写成： "now->tlink=×××;"，这样才是赋值啊！！！**综上所述：**while的循环条件只能是now->tlink是否为空，这样退出循环时，now指向的就是这个链表中最后那个结点，然后对它取->tlink进行赋值就能成功把这些弧结点串联起来了！！！！

**正确示例：**

struct ArcNode \*now;

now = G.node[a-1].firstout;

while(NULL != now->tlink)

{

now = now->tlink;

}

Now->tlink = next;

**附页(十字链表代码)：**

//---------------十字链表---------------

//有向图、有向网专有链表结构

//优点:解决了邻接表入度难以计算的问题

//这个代码编写过程中遇到2大问题：

// 1)当一个指针指向空指针NULL时，对它进行取值操作会出现内存错误

// 2)在向链表添加结点时，对指针指向的内容的操作不对，要让结点串联起来，应该是用指针p指向那个结点，并通过指针p改变那个结点里包含的指针的指向(即指针的值)，

// 而不是改变指针p的值，这样根本起不到串联结点的作用，没有改变指针指向的那个结点里的内容，只是单纯的在修改指针变量p的指向。

#include <iostream>

#include <string>

#include <stdlib.h>

using namespace std;

typedef enum {DG, DN}GraphKind;

typedef string vextype; //图中顶点的信息

typedef struct ArcNode{ //弧结点

int tailvex; //弧尾顶点的数组下标(记得-1)

int headvex; //弧头顶点的数组下标(记得-1)

struct ArcNode \*hlink; //指向含有相同弧头的下一个弧结点

struct ArcNode \*tlink; //指向含有相同弧尾的下一个弧结点

int info; //有向网时代表弧的权值

}ArcNode;

typedef struct VNode{ //顶点结点

vextype data; //顶点的信息

ArcNode \*firstin; //以该顶点作为第一个弧头的弧结点

ArcNode \*firstout; //以该顶点作为第一个弧尾的弧结点

}VNode;

typedef struct MGraph{

int vexnum; //图的顶点个数

int arcnum; //图的弧的个数

int kind; //图的种类

VNode \*node; //图的顶点结点数组，顺序存储的

}MGraph;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*尾插法(顺序)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//建立十字链表时采用的是尾插法，即每个新的弧结点都插在之前弧结点的后面

void CreateGraph(MGraph &G, int kind)

{

int i,a,b;

G.kind = kind;

cout << "输入图中顶点个数:";

cin >> G.vexnum;

cout << "输入图中弧的个数:";

cin >> G.arcnum;

// G.node = (VNode \*)malloc(G.vexnum \* sizeof(VNode));

G.node = new VNode[G.vexnum];

if(NULL == G.node) {cout << "new G.node error\n"; exit(1);}

cout << "输入各顶点信息(例如V1,V2,V3...):\n";

for(i = 0; i < G.vexnum; ++i)

{

cin >> G.node[i].data;

G.node[i].firstin = NULL;

G.node[i].firstout = NULL;

}

cout << "输入每条弧的两个顶点(头 尾，例如1 2，代表顶点1为弧尾、顶点2为弧头):\n";

for(i = 0; i < G.arcnum; ++i)

{

cin >> a;

cin >> b;

ArcNode \*next = new ArcNode;

next->tailvex = a-1; //弧尾下标赋值

next->headvex = b-1; //弧头下标赋值

if(G.kind == DN)

{

cout << "输入弧尾为" << a << "弧头为" << b << "的弧的权值:";

cin >> next->info;

}

else

{

next->info = -1;

}

next->hlink = NULL;

next->tlink = NULL;

//记录顶点a的出度信息

if(NULL == G.node[a-1].firstout) //该位置的顶点的出度还为空时，直接让该顶点的fisrstout指针指向新的表结点

{

G.node[a-1].firstout = next;

}

else

{ //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*这里就是尾插法的关键代码\*\*\*\*\*\*\*\*↓↓↓\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ArcNode \*now;

now = G.node[a-1].firstout; //找含有同样顶点作为弧尾的弧结点，一直找到最后一个这样的弧结点，然后在它后面添加新增的这个含有相同弧尾的弧结点

// cout << "更新出度信息，now->headvex:" << now->headvex << "now->tailvex:" << now->tailvex << endl;

//法1：1

//法1：2 while(NULL != now) //找到最后一个表结点

//法1：3 {

//法1：4 cout << "出度:" << now->tailvex << " " << now->headvex << endl;

//法1：5 now = now->tlink; /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*注!!!!!!!!!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//法1：6 //对NULL进行取值的操作往往会造成Segmentation fault,比如now已经等于NULL了，还取now->tlink肯定错！！

//法1：7 } /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//法1：8

//法1：9 ××这句话是错的×× now->tlink = next; //更新最后一个表结点,这行好像总是会报错"Segmentation fault"。。。。。。。。。

//法1：10 //因为此时的now等于最后那个弧结点的tlink，其值为NULL。上面while循环的退出条件就是now等于NULL时！！！！！

//法1：11 now = next; //更新最后一个表结点

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//方法1这种更新链表的方式，到最后，每个顶点的firstout都只会指向第一次输入的那根弧的弧结点，然后不会再指向后面的新增的弧结点。

//归根到底的原因就是： "now = next;" 这句话！！！这句话只是把新增弧结点的地址赋给了now这个ArcNode指针，而不是赋给now指向的那个tlink指针！！！！

//如果用法1这种赋值，虽然当退出第7行的while循环后，now的值变成了最后那个结点的tlink，但是第11行的 "now=next;" 这句话又改变了now指针的指向，让它指向了新增的结点next，

//而不是起到给now指向的tlink指针赋值的作用！！！！！ 所以这里如果要给now指向的那个tlink指针赋值(即让这个链表的各结点串联起来)，就必须是写成： "now->tlink=×××;"，这样才是赋值啊！！！

//综上所述：while的循环条件只能是now->tlink是否为空，这样退出循环时，now指向的就是这个链表中最后那个结点，然后对它取->tlink进行赋值就能成功把这些弧结点串联起来了！！！！

//更新链表法2：

while(now->tlink)

now = now->tlink;

now->tlink = next;

}

//记录顶点b的入度信息

if(NULL == G.node[b-1].firstin) //该位置的顶点的入度还为空时，直接让该顶点的fisrstin指针指向新的表结点

{

G.node[b-1].firstin = next;

}

else

{

ArcNode \*now = G.node[b-1].firstin; //找含有同样顶点作为弧头的弧结点，一直找到最后一个这样的弧结点，然后在它后面添加新增的这个含有相同弧头的弧结点

// cout << "更新入度信息，now->headvex:" << now->headvex << "now->tailvex:" << now->tailvex << endl;

/\*法1： while(NULL != now) //找到最后一个表结点

{

now = now->hlink;

}

// now->hlink = next; //更新最后一个表结点,不能写成now->hlink！！！！因为此时的now已经是最后那个弧结点的hlink了，其值为NULL

now = next;\*/

//法2：

while(now->hlink)

now = now->hlink;

now->hlink = next;

}

}//for

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*头插法(逆序)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//建立十字链表时采用的是头插法，即每个新的弧结点都插在之前弧结点的前面

void CreateGraph\_Inverted(MGraph &G, int kind)

{

int i,a,b;

G.kind = kind;

cout << "输入图中顶点个数:";

cin >> G.vexnum;

cout << "输入图中弧的个数:";

cin >> G.arcnum;

// G.node = (VNode \*)malloc(G.vexnum \* sizeof(VNode));

G.node = new VNode[G.vexnum];

if(NULL == G.node) {cout << "new G.node error\n"; exit(1);}

cout << "输入各顶点信息(例如V1,V2,V3...):\n";

for(i = 0; i < G.vexnum; ++i)

{

cin >> G.node[i].data;

G.node[i].firstin = NULL;

G.node[i].firstout = NULL;

}

cout << "输入每条弧的两个顶点(头 尾，例如1 2，代表顶点1为弧尾、顶点2为弧头):\n";

for(i = 0; i < G.arcnum; ++i)

{

cin >> a;

cin >> b;

ArcNode \*next = new ArcNode;

next->tailvex = a-1; //弧尾下标

next->headvex = b-1; //弧头下标 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*头插法(逆序)中的关键步骤\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

next->hlink = G.node[b-1].firstin; //将新增的表结点放入顶点结点的firstin，此时这个结点的hlink指向的就是原来顶点结点指向的firstin(即：第一个以该顶点结点为弧头的弧结点)

next->tlink = G.node[a-1].firstout;//将新增的表结点放入顶点结点的firstout，此时这个结点的tlink指向的就是原来顶点结点指向的firstout(即：第一个以该顶点结点为弧尾的弧结点)

if(G.kind == DN)

{

cout << "输入弧尾为" << a << "弧头为" << b << "的弧的权值:";

cin >> next->info;

}

else

{

next->info = -1;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*头插法(逆序)中的关键步骤\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

G.node[a-1].firstout = next; //最后让顶点结点a的firstout指向此时新增的那个弧结点

G.node[b-1].firstin = next; //最后让顶点结点a的firstin指向此时新增的那个弧结点

}

}

void print(const MGraph &G)

{

int i;

ArcNode \*now;

cout << "\n各顶点出度信息(即:以该顶点为弧尾时，弧的弧头顶点下标-1):\n";

for(i = 0; i < G.vexnum; ++i)

{

now = G.node[i].firstout;

cout << "(" << i << ") " << G.node[i].data << ": ";

while(NULL != now)

{

if(G.kind == DG)

cout << now->headvex << " ";

else

cout << now->headvex << "(" << now->info << ") ";

now = now->tlink;

}

cout << endl;

}

cout << "\n各顶点入度信息(即:以该顶点为弧头时，弧的弧尾顶点下标-1):\n";

for(i = 0; i < G.vexnum; ++i)

{

now = G.node[i].firstin;

cout << "(" << i << ") " << G.node[i].data << ": ";

while(NULL != now)

{

if(G.kind == DG)

cout << now->tailvex << " ";

else

cout << now->tailvex << "(" << now->info << ") ";

now = now->hlink;

}

cout << endl;

}

cout << endl;

}

void clear(MGraph &G)

{

int i;

ArcNode \*p, \*q;

for(i = 0; i < G.vexnum; ++i) //为了避免对已delete的结点再次delete导致错误，故只delete以每个顶点为入度的所有弧结点，这样就可以保证一次性释放所有动态分配的弧结点

{

p = G.node[i].firstin;

while(NULL != p)

{

q = p->hlink;

delete p;

p = q;

}//while

}//for

delete [] G.node;

}

int main()

{

MGraph G;

cout << "创建有向图:\n";

CreateGraph(G, DG); //尾插法(顺序插入)

// CreateGraph\_Inverted(G, DG); //头插法(逆序插入)

print(G);

clear(G);

cout << "创建有向网:\n";

CreateGraph(G, DN); //尾插法(顺序插入)

// CreateGraph\_Inverted(G, DN); //头插法(逆序插入)

print(G);

clear(G);

return 0;

}