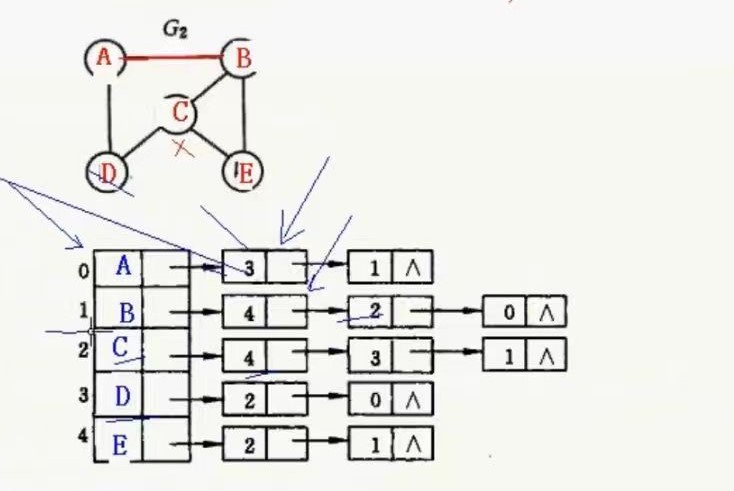
**图之邻接表表示法实现**

****

#include <iostream>

#include<malloc.h>

#include <assert.h>

using namespace std;

#define Default\_Vertex\_Size 10

#define T char

//定义边结点

typedef struct Edge {

int dest; //保存顶点的下标

struct Edge\*link; //指向下一个边结点的指针

}Edge;

//定义顶点结构

typedef struct Vertex {

T data;

struct Edge\*adj;

}Vertex;

//定义邻接表

typedef struct GraphLnk {

int MaxVertices;

int NumVertices;

int NumEdges;

Vertex\* NodeTable; //指向邻接表的指针

}GraphLnk;

void InitGraph(GraphLnk&gl) {

gl.MaxVertices = Default\_Vertex\_Size;

gl.NumVertices = gl.NumEdges = 0;

//创建表结构

gl.NodeTable = (Vertex\*)malloc(sizeof(Vertex\*)\*gl.MaxVertices);

assert(gl.NodeTable != NULL);

//将表结构里的指针赋初始值

for (int i = 0; i < gl.MaxVertices; i++) {

gl.NodeTable[i].adj = NULL;

}

}

void InsertVertex(GraphLnk&gl,T vertex) {

if (gl.NumVertices >= gl.MaxVertices) {

return;

}

gl.NodeTable[gl.NumVertices++].data = vertex;

}

void ShowGraph(GraphLnk&gl) {

Edge \*p;

for (int i = 0; i < gl.NumVertices; i++) {

cout << i << " ";

cout << gl.NodeTable[i].data << "-->";

p = gl.NodeTable[i].adj;

while (p != NULL) {

cout << p->dest << "-->";

p = p->link;

}

cout <<"Nul"<<endl;

}

cout << endl;

}

int GetVertexPos(GraphLnk&gl,T v) {

for (int i = 0; i < gl.NumVertices; i++) {

if (gl.NodeTable[i].data == v) {

return i;

}

}

return -1;

}

void InsertEdge(GraphLnk&gl, T v1, T v2) {

int posv1 = GetVertexPos(gl, v1);

int posv2 = GetVertexPos(gl, v2);

if (v1 == -1 || v2 == -1) {

return;

}

Edge\*s;

//V1-->V2

s = (Edge\*)malloc(sizeof(Edge));

assert(s != nullptr);

s->dest = posv2;

//这里采用头插的方式

s->link = gl.NodeTable[posv1].adj;

gl.NodeTable[posv1].adj = s;

//V2-->V1

s = (Edge\*)malloc(sizeof(Edge));

assert(s != nullptr);

s->dest = posv1;

s->link = gl.NodeTable[posv2].adj;

gl.NodeTable[posv2].adj = s;

gl.NumEdges++;

}

void RemoveEdge(GraphLnk&gl, T v1, T v2) {

int posv1 = GetVertexPos(gl, v1);

int posv2 = GetVertexPos(gl, v2);

if (v1 == -1 || v2 == -1) {

return;

}

//V1-->V2

Edge\*p;

Edge\*q=nullptr;

p = gl.NodeTable[posv1].adj;

while (p != nullptr&&p->dest!=posv2) {

q = p; //q始终指向p的前趋

p = p->link;

}

if (p == nullptr) {

return;

}

if (q == nullptr) { //说明第一个就是我们想要找的结点

gl.NodeTable[posv1].adj = p->link;

}

else {

q->link = p->link;

}

free(p);

//V2-->V1

p = gl.NodeTable[posv2].adj;

q = nullptr;

while (p->dest != posv1) { //p != nullptr可以不用写，因为如果能执行上面的操作，边就一定存在

q = p;

p = p->link;

}

if (p == nullptr) {

return;

}

if (q == nullptr) {

gl.NodeTable[posv2].adj = p->link;

}

else {

q->link = p->link;

}

free(p);

gl.NumEdges--;

}

//删除顶点，很复杂

void RemoveVertex(GraphLnk&gl, T v) {

int posv = GetVertexPos(gl, v);

if (posv == -1) {

return;

}

Edge\*p = gl.NodeTable[posv].adj;

int k;

Edge\*s;

Edge\*t = NULL;

//这里如果p为空，直接不做删除即可

while (p != nullptr) {

k = p->dest;

s = gl.NodeTable[k].adj;

while (s != nullptr&&s->dest != posv) {

t = s; //t为s的前趋结点

s = s->link;

}

if (s != nullptr) {

if (t == nullptr) {

gl.NodeTable[k].adj = s->link;

}

else {

t->link = s->link;

}

free(s);

}

gl.NodeTable[posv].adj = p->link;

free(p);

p = gl.NodeTable[posv].adj;

}

//用最后一个顶点覆盖要删除的顶点

gl.NumVertices--; //先将顶点个数减去1

gl.NodeTable[posv].data = gl.NodeTable[gl.NumVertices].data;

gl.NodeTable[posv].adj = gl.NodeTable[gl.NumVertices].adj;

//更改边的数据

s = gl.NodeTable[posv].adj;

while (s != nullptr) {

k = s->dest;

p = gl.NodeTable[k].adj;

while (p != nullptr) {

if (p->dest == gl.NumVertices) {

p->dest = posv;

break;

}

p = p->link;

}

s = s->link;

}

}

void DestroyGrph(GraphLnk&gl) {

Edge\*p;

for (int i = 0; i < gl.NumVertices; i++) {

p = gl.NodeTable[i].adj;

while (p != nullptr) {

gl.NodeTable[i].adj = p->link;

free(p);

p = gl.NodeTable[i].adj;

}

}

free(gl.NodeTable);

gl.NodeTable = nullptr;

gl.MaxVertices = gl.NumEdges = gl.NumVertices = 0;

}

int GetFirstNeighbor(GraphLnk&gl, T v) {

int posv = GetFirstNeighbor(gl, v);

if (posv == -1) {

return -1;

}

Edge\*p = gl.NodeTable[posv].adj;

if (p != nullptr) {

return p->dest;

}

return -1;

}

int GetNextNeighbor(GraphLnk&gl, T v1, T v2) {

int posv1 = GetVertexPos(gl, v1);

int posv2 = GetVertexPos(gl, v2);

if (v1 == -1 || v2 == -1) {

return -1;

}

Edge\*p = gl.NodeTable[posv1].adj;

while (p != nullptr&&p->dest!=posv2) {

p = p->link;

}

if (p != nullptr && p->link!=nullptr) { //p->link!=nullptr 要保证后续结点不空

return p->link->dest;

}

return -1;

}

int main() {

GraphLnk gl;

InitGraph(gl);

InsertVertex(gl, 'A');

InsertVertex(gl, 'B');

InsertVertex(gl, 'C');

InsertVertex(gl, 'D');

InsertVertex(gl, 'E');

InsertEdge(gl, 'A', 'B');

InsertEdge(gl, 'A', 'D');

InsertEdge(gl, 'B', 'C');

InsertEdge(gl, 'B', 'E');

InsertEdge(gl, 'C', 'D');

InsertEdge(gl, 'C', 'E');

ShowGraph(gl);

//RemoveEdge(gl, 'B', 'C');

RemoveVertex(gl, 'C');

ShowGraph(gl);

return 0;

}