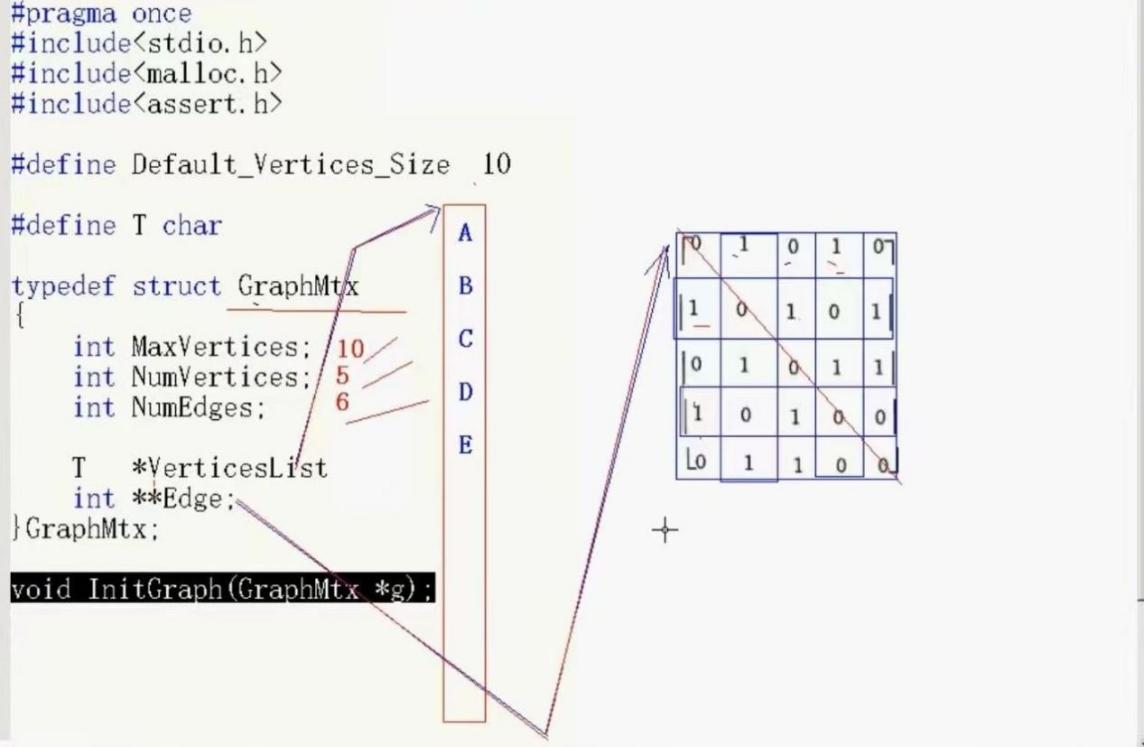
**图之邻接矩阵表示法实现**

无向图的表示矩阵是对称的

有向图的表示矩阵不一定对称

这里我们使用无向图（不带权）来进行实现



#include <iostream>

#include<malloc.h>

#include <assert.h>

using namespace std;

#define Default\_Vertex\_Size 10

#define T char

typedef struct GraphMtx {

int MaxVertices; //容量（可以不填满）

int NumVertices; //真实顶点数

int NumEdges; //真实边数

T\* verticeList; //定义指向顶点列表的指针

int\*\* Edge; //边不是0就是1，因此是int，又因为是一个二维列表，所以就用二级指针来指向整个二维列表

}GraphMtx;

void InitGraph(GraphMtx &g) {

g.MaxVertices = Default\_Vertex\_Size;

g.NumEdges = g.NumVertices = 0;

//对顶点列表的开辟

g.verticeList = (T\*)malloc(sizeof(T) \* (g.MaxVertices));

assert(g.verticeList != NULL);

//对边二维列表的开辟(之前没有遇到过，第一次学习)

g.Edge = (int\*\*)malloc(sizeof(int\*) \* g.MaxVertices);

assert(g.Edge != NULL);

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

g.Edge[i] = (int\*)malloc(sizeof(int) \* g.MaxVertices);

}

//将二维列表里的每一个数据都初始化为0

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

for (int j = 0; j < g.MaxVertices; j++) {

g.Edge[i][j] = 0;

}

}

}

void InsertVertex(GraphMtx&g, T v) {

if (g.NumVertices >= g.MaxVertices) { //先看看满不满

return;

}

g.verticeList[g.NumVertices++] = v;

}

void ShowGraph(GraphMtx& g) {

cout << " ";

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

cout << g.verticeList[i] << " ";

}

cout << endl;

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

cout << g.verticeList[i] << " ";

for (int j = 0; j < g.NumVertices; j++) {

cout << g.Edge[i][j] << " ";

}

cout << endl;

}

}

int GetVertexPos(GraphMtx& g,T v) {

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

if (g.verticeList[i] == v) {

return i;

}

}

return -1;

}

void InsertEdge(GraphMtx& g, T v1, T v2) {

int p1 = GetVertexPos(g, v1);

int p2 = GetVertexPos(g, v2);

if (p1 == -1 || p2 == -1) {

return;

}

if (g.Edge[p1][p2] != 0) {

return;

}

g.Edge[p1][p2] = g.Edge[p2][p1] = 1;

g.NumEdges++;

}

void RemoveEdge(GraphMtx& g, T v1, T v2) {

int p1 = GetVertexPos(g, v1);

int p2 = GetVertexPos(g, v2);

if (p1 == -1 || p2 == -1) {

return;

}

if (g.Edge[p1][p2] ==0) {

return;

}

g.Edge[p1][p2] = g.Edge[p2][p1] = 0;

g.NumEdges--;

}

//比较复杂，既要删除顶点，也要删除相关行和列

void RemoveVertex(GraphMtx& g, T v) {

//下面是追求美观不追求效率的方法

//效率高的方法是用最后一行或最后一列进行覆盖

//先删除顶点

int p = GetVertexPos(g, v);

if (p == -1) {

return;

}

for (int i = p; i < g.NumVertices-1; i++) {

g.verticeList[i] = g.verticeList[i + 1];

}

//统计被删除的边数

int num = 0;

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

if (g.Edge[p][i] != 0) {

num++;

}

}

//下面删除顶点所在行和所在列删除

//先删除行

for (int i = p; i < g.NumVertices - 1; i++) {

for (int j = 0; j < g.NumVertices; j++) {

g.Edge[i][j] = g.Edge[i + 1][j];

}

}

//再删除列

for (int i = p; i < g.NumVertices-1; i++) {

for (int j = 0; j < g.NumVertices; j++) {

g.Edge[j][i] = g.Edge[j][i+1];

}

}

g.NumVertices--;

g.NumEdges -= num;

}

void DestroyGraph(GraphMtx& g) {

free(g.verticeList);

g.verticeList = nullptr;

//释放二维列表要多一步，即释放两次

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

free(g.Edge[i]);

}

free(g.Edge);

g.Edge = nullptr;

g.MaxVertices = g.NumEdges = g.NumVertices = 0;

}

//找到第一个邻接顶点

//顺序是按照我们自己插入的顺序来看的

int GetFirstNeighbor(GraphMtx& g, T v) {

int p= GetVertexPos(g, v);

if (p == -1) {

return -1;

}

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

if(g.Edge[p][i]==1){

return i;

}

}

return -1;

}

int GetNextNeighbor(GraphMtx& g, T v, T w) {

int pv = GetVertexPos(g, v);

int pw = GetVertexPos(g, w);

if (pv== -1||pw==-1) {

return -1;

}

for (int i = pw+1; i < g.NumVertices; i++) {

if (g.Edge[pv][i] == 1) {

return i;

}

}

return -1;

}

测试函数：

int main() {

GraphMtx gm;

InitGraph(gm);

InsertVertex(gm, 'A');

InsertVertex(gm, 'B');

InsertVertex(gm, 'C');

InsertVertex(gm, 'D');

InsertVertex(gm, 'E');

ShowGraph(gm);

cout << endl;

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

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

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

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

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

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

ShowGraph(gm);

cout << endl;

int p=GetFirstNeighbor(gm, 'A');

cout << p << endl;

int p2= GetNextNeighbor(gm, 'A','B');

cout << p2 << endl;

RemoveEdge(gm, 'B', 'C');

ShowGraph(gm);

cout << endl;

RemoveVertex(gm, 'C');

ShowGraph(gm);

cout << endl;

DestroyGraph(gm);

}