课程小项目3：

[问题描述]

给定一个图，设计一个程序，找出一条从某一顶点A到另一顶点B边数最少的一条路径。

[输入]

图的顶点个数N，图中顶点之间的边的关系及要找的路径的起点A和终点B。

[输出]

若A到B无路径，则输出“There is no path”，否则输出A到B路径上各顶点。

[存储结构]

图采用邻接矩阵或邻接表的方式存储。

**项目3实验报告**

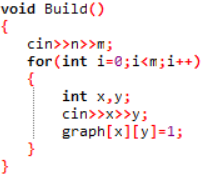
1. **实验内容简介**

根据已给信息建立一个有向图的邻接矩阵，找出一条从某一顶点a到另一顶点b边数最少的一条路径。

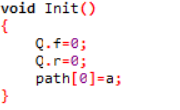
输入：第一行输入两个数n、m，n表示图中有n个点，m表示接下来的m行每行输入两个数x、y，表示有一条从x指向y的有向边。最后输入a、b，表示询问一条从某一顶点a到另一顶点b边数最少的一条路径。

输出：若a到b无路径，则输出“There is no path”，否则输出a到b路径上各顶点（各顶点之间用空格分隔）。

1. **程序设计思路**
   * + 1. 建立邻接矩阵：这个有向图用一个n\*n大小的二维矩阵graph[][]来存储，初始化所有元素为0。当有一条从x指向y的有向边时，graph[x][y]赋值为1。
       2. 寻路：采用广搜的方法，从a开始依次访问与a邻接的点v1、v2、……、vk，若没有到达b，则继续访问与v1的邻接的点v11、v12、……、v1k，……。如此下去，直到访问到b。用这样的方法，最先到达b的路径一定是边数最少的路径。
       3. 记录：采用队列来记录被访问过的点。每次访问与队列头部邻接的点，然后删去队列头部的点。如果队列为空队列，则说明a到b无路径。在每次访问点的过程中，把当前点的序号作为与其邻接且未被访问的点的前驱点记录下来以便于输出时进行从b到a的回推。
2. **算法分析与设计**
   * + 1. 建立邻接矩阵的函数Build()

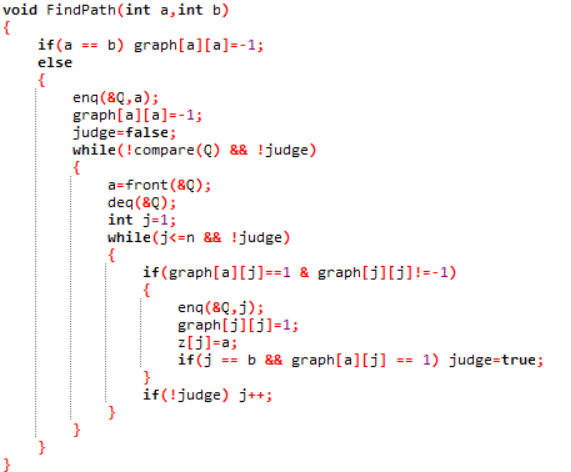


* + - 1. 初始化函数Init()



将队列的头和尾初始化为0，将记录路径的path[]的起点设置为a。

* + - 1. 寻路函数FindPath()



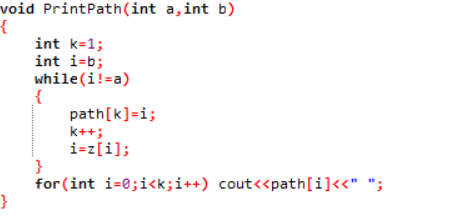
enq(q \*Q,int x)函数的核心代码为Q->q[Q->r]=x，作用是将队列的尾设置为起点a。

front(q \*Q)函数的核心代码为return Q->q[Q->f]，作用是返回队列头的序号。

deq(q \*Q)函数的核心代码为Q->f++，作用是将队列的头后移一个单位。

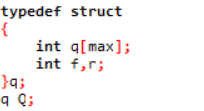
compare(q Q)通过判断Q.f与Q.r是否相等来判断队列是否为空。

* + - 1. 打印路径的函数PrintPath()



利用z[]回溯得到path[]并顺序打印path[]中的序号。

1. **数据结构设计**
   * + 1. 队列结构体



一维数组q[]存储点，f、r分别模拟一个队列的头和尾。

* + - 1. 二维数组graph[][]存储邻接矩阵

用来存储邻接矩阵。

* + - 1. 一维数组path[]、z[]

前者用来记录路径上的点的序号，后者用来记录当前点的序号，作为与其邻接且未被访问的点的前驱点。

1. **系统实现**

开始

Build()

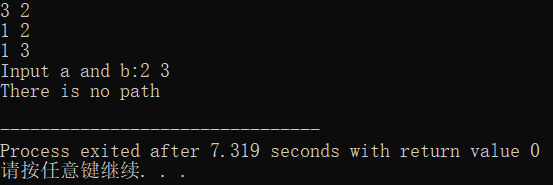
Input a and b

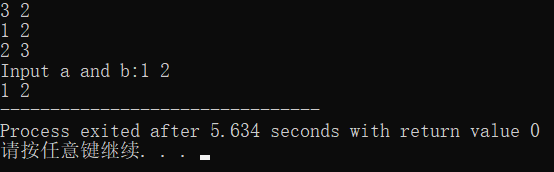
FindPath()

PrintPath()

Init()

1. **测试结果**





**附件：源代码**

#include<bits/stdc++.h>

using namespace std;

#define max 100

typedef struct

{

    int q[max];

    int f,r;

}q;

q Q;

int graph[max][max],z[max],path[max];;

int n,m,x,y,a,b;

bool judge;

void enq(q \*Q,int x)

{

    Q->q[Q->r]=x;

    if(Q->r == max-1) Q->r=0;

    else Q->r++;

    if(Q->r == Q->f) cout<<"Error!"<<endl;

}

int front(q \*Q)

{

    if(Q->r == Q->f) cout<<"Error!"<<endl;

    else return Q->q[Q->f];

}

void deq(q \*Q)

{

    if(Q->r == Q->f) cout<<"Error!"<<endl;

    else

    {

        if(Q->f == max-1) Q->f=0;

        else Q->f++;

    }

}

bool compare(q Q)

{

    if(Q.f == Q.r) return true;

    else return false;

}

void Build()

{

   cin>>n>>m;

   for(int i=0;i<m;i++)

   {

       int x,y;

       cin>>x>>y;

       graph[x][y]=1;

   }

}

void Init()

{

    Q.f=0;

    Q.r=0;

    path[0]=a;

}

void FindPath(int a,int b)

{

    if(a == b) graph[a][a]=-1;

    else

    {

        enq(&Q,a);

        graph[a][a]=-1;

        judge=false;

        while(!compare(Q) && !judge)

        {

            a=front(&Q);

            deq(&Q);

            int j=1;

            while(j<=n && !judge)

            {

                if(graph[a][j]==1 & graph[j][j]!=-1)

                {

                    enq(&Q,j);

                    graph[j][j]=1;

                    z[j]=a;

                    if(j == b && graph[a][j] == 1) judge=true;

                }

                if(!judge) j++;

            }

        }

    }

}

void PrintPath(int a,int b)

{

    int k=1;

    int i=b;

    while(i!=a)

    {

        path[k]=i;

        k++;

        i=z[i];

    }

    for(int i=0;i<k;i++) cout<<path[i]<<" ";

}

int main()

{

    Build();

    cout<<"Input a and b:";

    cin>>a>>b;

    Init();

    FindPath(a,b);

    if(judge) PrintPath(a,b);

    else cout<<"There is no path"<<endl;

    return 0;

}