6-15

typedef char vextype

typedef struct node{

int adjvex;

struct node \* next;

} edgenode;

typedef struct{

vextype vertex;

edgenode \* link;

}vexnode;

vexnode ga[n];

void CreatAdjlist(Vexnode ga[])

{

int i, j, k;

edgenode \* s;

for(i = 0; i < N; ++i)

{

ga[i].vertex = getchar();

ga[i].link = NULL;

}

for(k = 0; k < E; ++k)

{

scanf("%d%d", &i, &j);

s = malloc(sizeof(edgenode));

s -> adjvex = j;

s -> next = ga[i].link;

ga[i].link = s;

}

}

6-16

深度

#define N 8

typedef struct node

{

int adjvex;

struct node \* next;

}edgenode;

typedef struct

{

char vertex;

edgenode \* link;

}vexnode;

int visited[N];

int exist\_path\_DFS(vexnode ga[], int i, int j) // 以邻接表为存储结构，判断两个结点之间是否有路径，有则返回1，否则返回0

{

edgenode \* p;

if(ga[i].link -> adjvex == j) 填空

return 1;

else

{

visited[i] = 1; //标记vi已被访问

p = ga[i].link; //访问vi的第一个邻接点 填空

while(p != NULL) //依次访问vi的邻接点 填空

{

if(visited[p -> adjvex] == 0 && exist\_path\_DFS(ga, p -> adjvex, j) == 1) //vi的邻接点若未被访问过，且该邻接点到vj存在路径 填空

return 1;

p = p -> next;

}

}

return 0;

}

广度

int visited[MaxSize] = {0};

int BFS(ALGraph G, int i, int j)

{

InitQueue(Q);

EnQueue(Q, i);

while(!isEmpty(Q))

{

DeQueue(Q, u);

visited[u] = 1;

if(u == j)

return 1;

for(int p = FirstNeighbor(G, u); p; p = NextNeighbor(G, u, p))

{

if(p == j)

return 1;

if(!visited[p])

{

EnQueue(Q, p);

visited[p] = 1;

}

}

}

return 0;

}

6-17

d = -1;

void FindPath(Graph \* G, int u, int v, int path[], int d)

{

int w, i;

ArcNode \* p;

d++;

path[d] = u;

visited[u] = 1;

if(u == v)

printf(path[]);

p = G -> adjlist[u].firstarc;

while(p != NULL)

{

w = p -> adjvex;

if(visited[w] == 0)

FindPath(G, w, v, path, d);

p = p -> nextarc;

}

visited[u] = 0;

}

6-18

bool visited[Max\_Vertex\_Num];

void DFSTraverse(Graph G)

{

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

{

visited[v] = FALSE;

}

time = 0;

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

{

if(!visited[v])

DFS(G, v);

}

}

void DFS(Graph G, int v)

{

visited[v] = TRUE;

Visit(v);

for(w = FirstNeighbor(G,v); w >= 0; w = NextNeighbor(G, v, w))

{

if(!visited[w])

{

DFS(G, w);

}

}

time = time + 1;

finishTime[v] = time;

}

6-19

#include<iostream>

#include<cstdio>

using namespace std;

bool f[105][105],a[105]; //f邻接矩阵，a给走过的顶点做标记

int m,n,ans;

void ooo(int s)

{

a[s]=1; //标记

for(int i=1;i<=n;i++)

{

if(!a[i]&&f[s][i]) //当两点连通并且此点没有被访问过

{

m++; //累计总数

a[i]=1; //标记

ooo(i); //DFS

}

}

}

int main()

{

int x,y;

scanf("%d%d%d",&n,&x,&y);

while(x&&y) //末尾是0 0时结束输入

{

f[x][y]=1; f[y][x]=1; //无向图要双向处理

scanf("%d%d",&x,&y);

}

for(int i=1;i<=n;i++)

{

m=0; //m记录当前顶点连通的所有顶点数

if(!a[i]) //当此点没有被访问过

{

ooo(i);

ans=max(ans,m); //判断最大数

}

}

printf("%d",ans+1); //+1是因为连通的顶点数还要加上它本身的一个点

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int mp[100][100];

int visit[100];

void dfs(int x,int n)

{

int i;

visit[x]=1;

for(i=1;i<=n;i++)

{

if(!visit[i]&&mp[x][i])

{

dfs(i,n);

}

}

}

int main()

{

int T,n,m,u,v,i,count;

scanf("%d",&T);

while(T--)

{ count=0;

scanf("%d%d",&n,&m);

memset(mp,0,sizeof(mp));

memset(visit,0,sizeof(visit));

for(i=0;i<=m-1;i++)

{

scanf("%d%d",&u,&v);

mp[u][v]=mp[v][u]=1;

}

for(i=1;i<=n;i++)

{

if(!visit[i])

{

dfs(i,n);

count++;//每有一个新的未被标记的点就有了一个新的分支

}

}

printf("%d\n",count);

}

return 0;

}

typedef struct Arcnode{

int idx;

struct Arcnode \*nextarc;

}ArcNode, \*ptrArc;

typedef int GElemType;

typedef struct{

GElemType vex;

ptrArc firstArc;

}VexType;

typedef struct{

VexType \*vexs; //需要进行动态分配，采用固定数组会导致堆栈溢出

int vexnum;

int arcnum;

int GKind;

}AdjList;

Status InitAdjList(AdjList \*G)

{

int i;

G->vexs=(VexType \*)malloc(MAX\_VEX\_NUM\*sizeof(VexType));

for(i=0;i<MAX\_VEX\_NUM;i++)

G->vexs[i].firstArc=NULL;

G->vexnum=0;

G->arcnum=0;

G->GKind=0;

return OK;

}

//创建邻接表

Status CreateAdjList(AdjList \*G, int n, int \*\*edges, int edgesSize)

{

int i;

ptrArc p;

InitAdjList(G);

for(i=0;i<n;i++)

G->vexs[i].vex=i;

G->vexnum=n;

for(i=0;i<edgesSize;i++)

{

p=(ptrArc)malloc(sizeof(ArcNode));

if(!p)

return ERROR;

p->idx=edges[i][1];

p->nextarc = G->vexs[edges[i][0]].firstArc;

G->vexs[edges[i][0]].firstArc=p;

p=(ptrArc)malloc(sizeof(ArcNode));

if(!p)

return ERROR;

p->idx=edges[i][0];

p->nextarc=G->vexs[edges[i][1]].firstArc;

G->vexs[edges[i][1]].firstArc=p;

G->arcnum++;

}

return OK;

}

//连通图的深度优先遍历

void DFS(AdjList G, int visited[], int v0, int batchNo)

{

ptrArc p;

int w;

visited[v0]=batchNo;

p=G.vexs[v0].firstArc;

while(p)

{

w=p->idx;

if(!visited[w])

{

DFS(G,visited, w,batchNo);

}

p=p->nextarc;

}

}

//非连通图的深度优先遍历

int DFSGraphTraverse(AdjList G)

{

int visited[MAX\_VEX\_NUM]={0};

int i;

int count=0;

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

{

if(!visited[G.vexs[i].vex]){

count++;

DFS(G,visited, G.vexs[i].vex, count);

}

}

return count;

}

int countComponents(int n, int\*\* edges, int edgesSize, int\* edgesColSize){

AdjList G;

CreateAdjList(&G, n, edges, edgesSize);

return DFSGraphTraverse(G);

}