|  |
| --- |
|  |
| 图论算法 |
|  |

|  |
| --- |
| 兰宇恒 数1601 201686043 |

主程序：

#include<iostream>

#include<stdlib.h>

#include "io.cpp"

#include "cc.cpp"

#include "degree.cpp"

#include "rand.cpp"

#include "spath.cpp"

void main()

{

int v;

cout<<"请输入点的个数:";

cin>>v;

graph g(v);

edge a;

cout<<"请输入连通边的端点，停止输入时取两边为负值:";

a.v=1;a.w=1;

while(a.v>=0||a.w>=0)

{

cin>>a.v>>a.w;

if(a.v>=0&&a.w>=0)

g.insert(a);

}

/\*cout<<"请输入随机生成边的大致个数:";

int e;

cin>>e;

//rande(g,e);

randg(g,e);\*/

io<graph>::show(g);

cout<<"对应点的度数分别为:";

Degree<graph>::print(g);

cout<<endl<<"边数为"<<g.e();

cc<graph> gcc(g);

cout<<endl<<"连通分支个数为"<<gcc.count()<<endl;

/\*cout<<"判断任意两点是否连通,请依次输入两端点:";

int i,j;

cin>>i>>j;

cout<<"是否连通:";

gcc.connect(i,j);

cout<<endl;\*/

cout<<"简单路径搜索,请依次输入两端点:";

int i,j;

cin>>i>>j;

cout<<endl<<"是否连通:";

gcc.connect(i,j);

cout<<endl<<"路径依次为为:";

spath<graph> spathrr(g,i,j);

cout<<spathrr.exists();

cout<<endl;

}

子程序：

1. 判断连通

#include<iostream>

#include<vector>

using namespace std;

template <class graph>

class cc

{

private:

const graph &g;

int ccnt;

vector <int> id;

void ccr(int w)

{

id[w]=ccnt;

typename graph::adjiterator a(g,w);

for (int v=a.beg();!a.end();v=a.nxt())

if(id[v]==-1)

ccr(v);

}

public:

cc(const graph &g):g(g),ccnt(0),id(g.v(),-1)

{

for(int v=0;v<g.v();v++)

if(id[v]==-1)

{ccr(v);ccnt++;}

}

int count() const {return ccnt;}

void connect(int s, int t)

{

if(id[s]==id[t])

cout<<"是";

else cout<<"否";

}

};

1. 计算度数

#include<iostream>

#include<vector>

using namespace std;

template <class graph>

class Degree

{

public:

static int print(const graph &g);

};

template <class graph>

int Degree<graph>::print(const graph &g)

{

for(int v=0;v<g.v();v++)

{

int degree=0;

typename graph::adjiterator a(g,v);

for(int w=a.beg();!a.end();w=a.nxt())

degree ++;

cout<<degree<<" ";

}

return 1;

}

1. 图类

#include<iostream>

#include<iterator>

#include<vector>

using namespace std;

struct edge

{

int v,w;

edge(int v=-1,int w=-1):v(v),w(w) {}

};

class graph

{

int vn,en;

bool digraph;

vector<vector<bool> > adj;

public:

graph(int v,bool digraph=false):

adj(v),vn(v),en(0),digraph(digraph)

{

for(int i=0;i<v;i++)

adj[i].assign(v,false);

}

int v() const{return vn;}

int e() const{return en;}

bool directed() const{return digraph;}

void insert(edge e)

{

int v=e.v,w=e.w;

if(adj[v][w]==false)

en++;

adj[v][w]=true;

if(!digraph)

adj[w][v]=true;

}

void removw(edge e)

{

int v=e.v,w=e.w;

if(adj[v][w]==true)

en--;

adj[v][w]=false;

if(!digraph)

adj[w][v]=false;

}

bool edge(int v,int w) const

{

return adj[v][w];

}

class adjiterator;

friend class adjiterator;

};

class graph::adjiterator

{

const graph &g;

int i,v;

public:

adjiterator(const graph &g,int v):

g(g),v(v),i(-1) {}

int beg()

{

i=-1;return nxt();}

int beg2()

{

i=-1;return nxt2();}

int nxt()

{

for (i++;i<g.v();i++)

if(g.adj[v][i]==true)

return i;

return -1;

}

int nxt2()

{

for (i++;i<g.v();i++)

{

if(g.adj[v][i]==true)

return 1;

else

return 0;

}

}

bool end()

{

return i>=g.v();

}

};

1. 输入输出流

#include<iostream>

using namespace std;

template <class graph>

class io

{

public:

static void show(const graph &);

//static void scanez(graph &);

//static void scan(graph &);

};

template <class graph>

void io<graph>::show(const graph &g)

{

cout<<"对应边相连为:"<<endl;

for(int s=0;s<g.v();s++)

{

cout.width(2);

cout<<s<<":";

typename graph::adjiterator a(g,s);

for (int t=a.beg();!a.end();t=a.nxt())

{

cout.width(2);

cout<<t<<" ";

}

cout<<endl;

}

cout<<"邻接矩阵为:"<<endl;

for(int s2=0;s2<g.v();s2++)

{

typename graph::adjiterator a2(g,s2);

for (int k=a2.beg2();!a2.end();k=a2.nxt2())

{

cout.width(2);

cout<<k<<" ";

}

cout<<endl;

}

}

1. 生成随机图

#include<iostream>

#include "graph.cpp"

using namespace std;

static void rande(graph &g,int e)

{

for(int i=0;i<e;i++)

{

int v=int(g.v()\*rand()/(1+RAND\_MAX));

int w=int(g.v()\*rand()/(1+RAND\_MAX));

g.insert(edge(v,w));

}

}

static void randg(graph &g,double e)

{

double p=2\*e/(g.v()\*(g.v()-1));

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

{

for(int j=0;j<i;j++)

{

if(rand()<p\*RAND\_MAX)

g.insert(edge(i,j));

}

}

}

1. 寻找哈密顿路和欧拉路

#include<iostream>

#include<vector>

using namespace std;

template <class graph>

class spath

{

const graph &g;

vector <bool> visited;

bool found;

bool searchr(int v,int w,int d)

{

if(v==w)

return (d==0);

visited[v]=true;

typename graph::adjiterator a(g,v);

for(int t=a.beg();!a.end();t=a.nxt())

if(!visited[t])

if(searchr(t,w,d-1))

{

cout<<t<<"-";

return true;

}

visited[v]=false;

return false;

}

public:

spath(const graph &g,int v,int w):

g(g),visited(g.v(),false)

{

found=searchr(v,w,g.v()-1);

}

bool exists() const

{

return found;

}

};

1. 最小生成树
2. Prim

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

using namespace std;

struct Edge

{

int u; //边连接的一个顶点编号

int v; //边连接另一个顶点编号

int w; //边的权值

friend bool operator<(const Edge& E1, const Edge& E2)

{

return E1.w < E2.w;

}

};

//创建并查集

void MakeSet(vector<int>& uset, int n)

{

uset.assign(n, 0);

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

uset[i] = i;

}

//查找当前元素所在集合的代表元

int FindSet(vector<int>& uset, int u)

{

int i = u;

while (uset[i] != i) i = uset[i];

return i;

}

void Kruskal(const vector<Edge>& edges, int n)

{

vector<int> uset;

vector<Edge> SpanTree;

int Cost = 0, e1, e2;

MakeSet(uset, n);

for (int i = 0; i < edges.size(); i++) //按权值从小到大的顺序取边

{

e1 = FindSet(uset, edges[i].u);

e2 = FindSet(uset, edges[i].v);

if (e1 != e2) //若当前边连接的两个结点在不同集合中，选取该边并合并这两个集合

{

SpanTree.push\_back(edges[i]);

Cost += edges[i].w;

uset[e1] = e2; //合并当前边连接的两个顶点所在集合

}

}

cout << "Result:\n";

cout << "Cost: " << Cost << endl;

cout << "Edges:\n";

for (int j = 0; j < SpanTree.size(); j++)

cout << SpanTree[j].u << " " << SpanTree[j].v << " " << SpanTree[j].w << endl;

cout << endl;

}

1. Kruskal

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

using namespace std;

struct Edge

{

int u; //边连接的一个顶点编号

int v; //边连接另一个顶点编号

int w; //边的权值

friend bool operator<(const Edge& E1, const Edge& E2)

{

return E1.w < E2.w;

}

};

//创建并查集

void MakeSet(vector<int>& uset, int n)

{

uset.assign(n, 0);

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

uset[i] = i;

}

//查找当前元素所在集合的代表元

int FindSet(vector<int>& uset, int u)

{

int i = u;

while (uset[i] != i) i = uset[i];

return i;

}

void Kruskal(const vector<Edge>& edges, int n)

{

vector<int> uset;

vector<Edge> SpanTree;

int Cost = 0, e1, e2;

MakeSet(uset, n);

for (int i = 0; i < edges.size(); i++) //按权值从小到大的顺序取边

{

e1 = FindSet(uset, edges[i].u);

e2 = FindSet(uset, edges[i].v);

if (e1 != e2) //若当前边连接的两个结点在不同集合中，选取该边并合并这两个集合

{

SpanTree.push\_back(edges[i]);

Cost += edges[i].w;

uset[e1] = e2; //合并当前边连接的两个顶点所在集合

}

}

cout << "Result:\n";

cout << "Cost: " << Cost << endl;

cout << "Edges:\n";

for (int j = 0; j < SpanTree.size(); j++)

cout << SpanTree[j].u << " " << SpanTree[j].v << " " << SpanTree[j].w << endl;

cout << endl;

}

1. 最短路径

#include <iostream>

#include <vector>

const int maxdist = 9999;

using namespace std;

/\*n是总的结点数,v是出发结点,dist是距离,pre前一个结点,d是结点间的权值\*/

void Dijkstra(int n, int v, vector<int> &dist, vector<int> &pre, vector<vector<int>> &d)

{

vector<bool> s(n+1);

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

{

dist[i] = d[v][i];

if (dist[i] < maxdist)

pre[i] = v;

else

pre[i] = 0;

}

dist[v] = 0;

s[v] = true;

for (int i = 2; i <= n;i++)//总的迭代次数

{

int best = v;

int temp = maxdist;

for (int j = 1; j <= n;j++)//找到最小的距离

{

if (!s[j]&&dist[j]<temp)

{

temp = dist[j];

best = j;

}

}

s[best] = true;

for (int j = 1; j <= n;j++)//更新dist和pre

{

if (!s[j] && d[best][j] != maxdist)

{

int newdist = dist[best] + d[best][j];

if (newdist<dist[j])

{

dist[j] = newdist;

pre[j] = best;

}

}

}

}

}

void printpath(vector<int> pre, int init, int fina)

{

int temp=fina;

vector<int> t;

while (temp != init)

{

t.push\_back(temp);

temp = pre[fina];

fina = temp;

}

cout << init << "->";

for (int i = t.size(); i >1;i--)

{

cout << t[i-1] << "->";

}

cout << t[0];

t.clear();

}

图示：

