南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

数据结构实验三报告

班　　级 9191062301

学生姓名 孙傲歆

学　　号 919106840333

题　　目 图的遍历

指导教师 练智超

一、实验要求

输入是一个基于邻接表或者邻接矩阵存储的无向图
目标：1）用非递归算法实现深度和广度遍历一个无向图，并输出遍历结果，注意如果该图不连通，可能需要多次遍历 2）用非递归算法实现 Kruskal 算法和 Prim 算法

二、算法思路

1）图的 DFS 遍历

DFS，是一种盲目搜寻法，沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点 v 的所在边都已被探寻过，搜索将回溯到发现节点 v 的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。如果还存在未被发现的节点，则选择其中一个作为源节点并重复以上过程，整个进程反复进行直到所有节点都被访问为止。

2）图的 BFS 遍历

BFS，同样属于一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位置，彻底地搜索整张图，直到找到结果为止。基本过程，BFS 是从根节点开始，沿着树(图)的宽度遍历树(图)的节点。如果所有节点均被访问，则算法中止。一般用队列数据结构来辅助实现 BFS 算法。

3）Kruskal 算法

Kruskal 算法是最小生成树的构造算法。首先构造一个由这 n 个顶点组成，不含任何边的图 T，其中每个顶点自成一个连通分量。该算法不断从图的边集 E 中取出代价最小的一条边（若有多条，则任选其一），若该边的两个顶点来自 T 中不同的连通分量，则讲此边加入 T 中，否则舍弃此边选择下一条代价最小边。以此类推，直到 T 中所有的顶点在同一个连通分量上为止。

4）Prim 算法

Prim 算法也是最小生成树的构造算法。算法始终将顶点集合 V 分成没有元素重叠的两部分，V=U∪(V-U)，T 的初始状态为 $U=\{u_0\}(u_0\in V)$，然后重复执行一下操作:在所有 $u\in U$，$v\in V-U$ 的边中找出一条代价最小的边并入生成树的边集中，同时 v_0并入 U，直至 U=V 为止。此时生成树必有 n-1 条边，则结果 T 就是图的最小生成树。

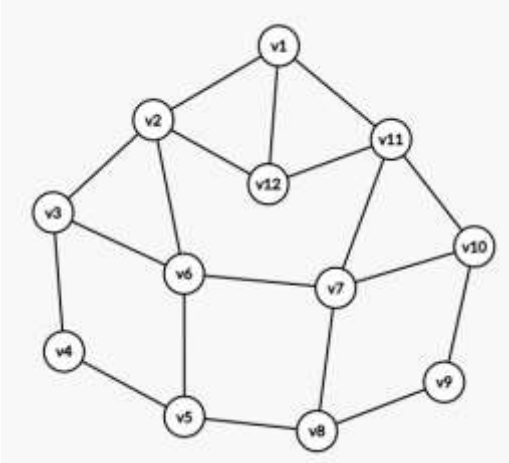
三、代码运行结果

1）DFS 算法和 BFS 算法

输入:图的点数以及图的邻接矩阵
输出:DFS 算法和 BFS 算法访问顶点的顺序
代码输入输出结果:

```
12
0 1 0 0 0 0 0 0 0 0 1 1
1 0 1 0 0 1 0 0 0 0 0 1
0 1 0 1 0 1 0 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0 0 0
0 0 0 1 0 1 0 1 0 0 0 0
0 1 1 0 1 0 1 0 0 0 0 0
0 0 0 0 0 1 0 1 0 1 1 0
0 0 0 0 1 0 1 0 1 0 0 0
0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 1 0 1 0 1 0
1 0 0 0 0 0 1 0 0 1 0 1
1 1 0 0 0 0 0 0 0 0 1 0
dfs遍历结果为:
1 2 3 4 5 6 7 8 9 10 11 12
bfs遍历结果为:
1 2 11 12 3 6 7 10 4 5 8 9
```

输入临近矩阵对应的图:



2）Kruskal 算法

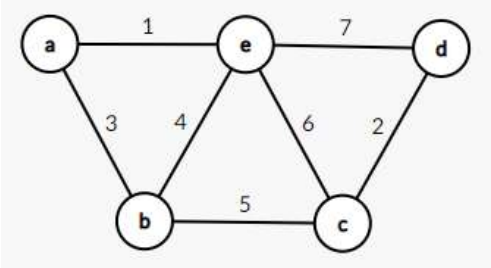
输入:图的边的个数以及图的邻接三元组

输出:最小生成树的边，以及最小生成树的权值

代码输入输出结果:

```
7
0 1 3
0 4 1
1 2 5
1 4 4
2 3 2
2 4 6
3 4 7
A->E
C->D
A->B
B->C
11
```

输入邻接表对应的图:



3）Prim 算法

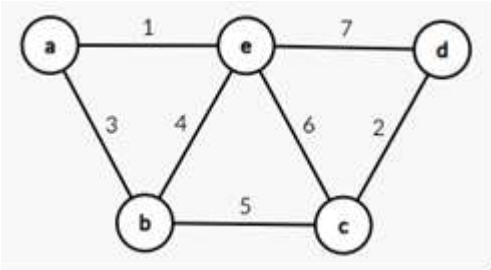
输入:图的点的个数、边的个数以及图的邻接表

输出:最小生成树的边，以及最小生成树的权值

代码输入输出结果:

```
5 7
0 1 3
0 4 1
1 2 5
1 4 4
2 3 2
2 4 6
3 4 7
11
A->B
B->C
C->D
A->E
```

输入邻接表对应的图:



四、代码

1）BFS 和 DFS

```
using namespace std;
#include<bits/stdc++.h>
int n;
int a[100][100];
int visit[100];
int getadj(int x)
{
    for(int i=1;i<=n;i++)
    {
```

```
            if(a[x][i]==1&&visit[i]==0)
                return i;
        }
        return 0;
}
void dfs(int v)
{
        cout<<v<<" ";
        visit[v]=1;
        int k=getadj(v);
        while(k!=0)
        {
            if(visit[k]==0)
                dfs(k);
            k=getadj(v);
        }
}
void bfs(int v)
{
        queue<int>q;
        int k,temp;
        cout<<v<<" ";
        visit[v]=1;
        q.push(v);
        while(!q.empty())
        {
            temp=q.front();
            q.pop();
            k=getadj(temp);
            while(k!=0)
            {
                if(visit[k]==0)
                {
                    cout<<k<<" ";
                    visit[k]=1;
                    q.push(k);
                }
                k=getadj(temp);
            }
        }
}
int main()
{
```

```
    cin>>n;
    memset(a,0,sizeof(a));
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        cin>>a[i][j];
    }

    cout<<"dfs 遍历结果为:"<<endl;
    memset(visit,0,sizeof(visit));
    for(int i=1;i<=n;i++)
    {
        if(visit[i]==0)
            dfs(i);
    }
    cout<<endl;

    cout<<"bfs 遍历结果为:"<<endl;
     memset(visit,0,sizeof(visit));
    for(int i=1;i<=n;i++)
    {
        if(visit[i]==0)
            bfs(i);
    }
    cout<<endl;
}
```

2）Kruskal 算法

```
using namespace std;
#include<bits/stdc++.h>
int n;
struct node
{
    int val;
    int start;
    int end;
};
node v[100];
int father[100];
int cap[100];
bool cmp(node a,node b)
{
    return a.val<b.val?true:false;
}
```

```
void make_set()
{
	for(int i=0;i<n;i++)
	{
		father[i]=i;
		cap[i]=1;
	}
}
int Find(int x)
{
	if(x!=father[x])
	  {
		father[x]=Find(father[x]);
	}
	return father[x];
}
void Union(int x,int y)
{
	x=Find(x);
	y=Find(y);
	if(x==y)
		return;
	if(cap[x]<cap[y])
		father[x]=Find(y);
	else
	{
		if(cap[x]==cap[y])
			cap[x]++;
		father[y]=Find(x);
	}
}
int Kruskal()
{
	int sum = 0;
	make_set();
	for(int i=0;i<n;i++)
	{
		if(Find(v[i].start)!=Find(v[i].end))
		{
			Union(v[i].start,v[i].end);
			cout<<char(v[i].start+65)<<"->"<<char(v[i].end+65)<<endl;
			sum+=v[i].val;
		}
	}
```

```
    return sum;
}
int main()
{
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>v[i].start>>v[i].end>>v[i].val;
    }
    sort(v,v+n,cmp);
    cout<<Kruskal()<<endl;
}
```

3）Prim 算法

```
using namespace std;
#include<bits/stdc++.h>
const int inf=1000000;
int n,m;
int a[100][100];
int lowcost[100];//表示 i 到距离集合最近的距离
int closest[100];//表示 i 与之相连边的顶点序号
int Prim(int s)
{
    for(int i=0;i<n;i++)
    {
        if(i==s)
            lowcost[i]=0;
        else
            lowcost[i]=a[s][i];
        closest[i]=s;
    }
    int minn,pos;
    int sum=0;
    for(int i=0;i<n;i++)
    {
        minn=inf;
        for(int j=0;j<n;j++)//找到集合最小边
        {
            if(lowcost[j]!=0&&lowcost[j]<minn)
            {
                minn=lowcost[j];
                pos=j;
            }
        }
```

```
            if(minn==inf)
                break;
            sum+=minn;
            lowcost[pos]=0;//加入点集合。
            for(int j=0;j<n;j++)//加入新点更新
            {
                if(lowcost[j]!=0&&a[pos][j]<lowcost[j])
                {
                    lowcost[j]=a[pos][j];
                    closest[j]=pos;
                }
            }
        }
        return sum;
}
int main()
{
        int u,v,w;
        cin>>n>>m;
        memset(a,inf,sizeof(a));
        for(int i=0;i<m;i++)
        {
            cin>>u>>v>>w;
            a[u][v]=a[v][u]=w;
        }
        cout<<Prim(0)<<endl;
        int temp;
        for(int i=1;i<n;i++)
        {
            temp=closest[i];
            cout<<char(temp+65)<<"->"<<char(i+65)<<endl;
        }
}
```