天津大学

程序设计综合实践课程报告

图论实验

学生姓名 陈秋澄

学院名称 智能与计算学部

专　　业 大类

学　　号 **3022244290**

1. dfs

1.1 题目分析

题目要求使用 dfs 算法搜索去做，而 DFS 是图论里面的一种搜索算法。他可以由一个根节点出发，遍历所有的子节点，进而把图中所有的可以构成树的集合都搜索一遍，达到全局搜索的目的。在这里可以使用邻接矩阵的方法求，使用二维数组标记出那些节点是相连的，然后从 1 开始遍历，如果这一个点和前一个点相连，而且没有被计算过则保存这一个点，最后输出得到的序列。

1.2 题目代码

```
#include <iostream>
#include <string.h>
#include <stdio.h>
using namespace std;
int s[1000][1000];            //邻接矩阵
int a[1000];                  //将 dfs 遍历序列储存在里面
int x[1000];                  //相当于 visited 数组
int j;
void dfs(int t,int n)         //设计递归
{
    a[j++]=t;
    x[t]=1;
    for(int i=1;i<=n;i++)
    {
        if(s[t][i]==1&&x[i]!=1)
            dfs(i,n);
    }
}
int main()
{
    int m,n;
    while(cin>>m>>n)
    {
        memset(x,0,sizeof(x));
        memset(x,0,sizeof(x));  //初始化
        int y,z;
```

```
        for(int i=0;i<n;i++)
        {
            cin>>y>>z;
            s[y][z]=s[z][y]=1;
        }
        dfs(1,m);
        for(int i=0;i<m-1;i++)
        cout<<a[i]<<" ";
    cout<<a[m-1]<<endl;
    }

    return 0;
}
```

2. bfs

2.1 题目分析

bfs 理论上与 dfs 原理相同，首先进行全局控制：比如我们通过变量 i 来控制我们遍历的行数，这样就能逐一击破了。

而有关初始点，我们知道坐标点需要从最左侧的 0 开始遍历，只要找到不是 0 的数就代表有链接点了。但下一个链接，坐标得从第 N 个开始遍历了，（因为之前的已经遍历过了），这里的 N 是变量 j。

2.2 题目代码

```
#include <bits/stdc++.h>
using namespace std;
#define N 15
int n,e,x,y;
int q[15],hh,tt=1;
bool g[N][N];
bool k[N];                      //建立 bool 函数
void bfs(){
    int l,flag=true,t;
    while(hh<tt){
        k[q[hh]]=true;
        l=hh;
        flag=false;
        t=q[hh];
        cout<<q[hh]<<" ";       //输出答案
        hh++;
        for(int i=1;i<=n;i++){
            if(!k[i]&&g[t][i]){
                k[i]=true;
                q[tt++]=i;
                flag=true;
            }
        }

    }
    return;
}
```

```
int main(){
    cin>>n>>e;
    for(int i=0;i<e;i++){
        cin>>x>>y;
        g[y][x]=g[x][y]=1;  //简化代码，意为 g[y][x]=(g[x][y]=1);
    }
    q[0]=1;
    k[1]=true;
    bfs();                  //调用 bfs 函数
    return 0;
}
```

3. 蜜罐

3.1 题目分析

经分析，本题为最小生成树问题，可用 Prim 算法实现。

Prim 算法每次循环都将一个蓝点 u 变为白点，并且此蓝点 u 与白点相连的最小边权 min[u]还是当前所有蓝点中最小的。这样相当于向生成树中添加了 n-1 次最小的边，最后得到的一定是最小生成树。

我们在脑海里构建这样一个图：

蓝点和虚线代表未进入最小生成树的点、边；白点和实线代表已进入最小生成树的点、边。初始时所有点都是蓝点，min[1]=0,min[2、3、4、5]=∞。权值之和 MST=0。

第一次循环自然是找到 min[1]=0 最小的蓝点 1。将 1 变为白点，接着枚举与 1 相连的所有蓝点 2、3、4，修改它们与白点相连的最小边权。min[2]=w[1][2]=2; min[3]=w[1][3]=4;min[4]=w[1][4]=7；

第二次循环是找到 min[2]最小的蓝点 2。将 2 变为白点，接着枚举与 2 相连的所有蓝点 3、5，修改它们与白点相连的最小边权。min[3]=w[2][3]=1;min[5]=w[2][5]=2;以此类推，求出答案。

3.2 题目代码

```
// prim 算法求最小生成树
#include <cstdio>
#include <string>
#include <cstring>
#include <iostream>
#include <algorithm>
#define INF 0x3f3f3f3f
using namespace std;
const int maxn = 505;
int a[maxn][maxn];
int vis[maxn],dist[maxn];
int n,m;
int u,v,w;
long long sum = 0;
int prim(int pos) {
```

```
    dist[pos] = 0;
    //一共有 n 个点,就需要 遍历 n 次,每次寻找一个权值最小的点,记录其下标
    for(int i = 1; i <= n; i ++) {
        int cur = -1;
        for(int j = 1; j <= n; j ++) {
            if(!vis[j] && (cur == -1 || dist[j] < dist[cur])) {
                cur = j;
            }
        }
                                // 这里需要提前终止
        if(dist[cur] >= INF) return INF;
        sum += dist[cur];
        vis[cur] = 1;
        for(int k = 1; k <= n; k ++) {
                                // 只更新还没有找到的最小权值
            if(!vis[k]) dist[k] = min(dist[k],a[cur][k]);
        }
    }
    return sum;
}
int main() {
    int t;
    cin>>t;
    while(t--){
        cin>>n>>m;
        memset(a,0x3f,sizeof(a));
        memset(dist,0x3f,sizeof(dist));
        for(int i = 1; i <= m; i ++) {
            cin>>u>>v>>w;
            a[u][v] = min(a[u][v],w);
            a[v][u] = min(a[v][u],w);
        }
        int value = prim(1);
        if(value >= INF) puts("impossible");
        else cout<<sum<<endl;      //输出
    }
    return 0;
}
```

4. 村村通

4.1 题目分析

城镇之间构成了一个又一个的集合，而对于集合的每一个元素（城镇）又没有特殊限定，那我们直接用并查集解决：

每建一条公路，就是把两城镇所在的集合合并。

显然，对于 n 个不同的集合，想要把它们连起来，至少需要连 n-1 条线，那我们求出一共有多少个集合。

把集合数减去 1，输出即可。

但是如何求集合的数量呢？

每个集合都有一个“祖宗”，“祖宗”的序号不会超过 1000，通过桶排序即可解决，即每遇到一个城镇，就把它的“祖宗”对应的下标变为 1，我们可以通过路径压缩使同一个集合的城镇拥有同一个祖宗，那最后遍历桶，看看有多少被标为 1 的元素即可。

4.2 题目代码

```
#include<iostream>
#include<bits/stdc++.h>
#include<cstdio>
#include<cstring>
using namespace std;
const int maxn=1001;            //定义常量 maxn 作为数组大小
int a[maxn];                    //并查集 a
bool ok[maxn];                  //桶
int cz(int x){                  //并查集查找函数
    if(a[x]==x) return x;
    else return a[x]=cz(a[x]);   //路径压缩
}
void hb(int x,int y){           //合并函数
    int x1=cz(x),y1=cz(y);
    a[x1]=a[y1];
}
int main(){
    int n,m,x,y,ans;
```

```
    while(1){
        cin>>n;                 //先读一个数据
        if(n==0) break;         //是 0，停止读入
        cin>>m;                 //不是 0，继续
        ans=0;                  //集合数量
        for(int i=1;i<=n;++i){
            a[i]=i;             //并查集初始化
        }
        for(int i=1;i<=m;++i){
            cin>>x>>y;          //把 x,y 两个城镇连起来
            hb(x,y);            //就是合并 x,y 所在的集合
        }
        for(int i=1;i<=n;++i){
            ok[cz(i)]=1;       //入桶
        }
        for(int i=1;i<=n;++i){
            if(ok[i]) ans++;  //被标记过，代表着一个集合
        }
        cout<<ans-1<<endl;    //输出答案
        memset(ok,0,sizeof(ok));  //清空桶
    }
    return 0;
}
```

5. 一个人的旅行

5.1 题目分析

经分析，此题是一道明显的最短路问题，可以用 **dijkstra** 和 **spfa** 等解决 。

一般的做法较容易想到，就是求出所有出发的站到所有终点站的最短路径中的最小值 ，这样就重复多次调用 **dijkstra** 或 **spfa**， 但如果运用一些技巧就可大大优化，题目中 **a , b** 均是大于 **1** 的，所以可以在设一个点作为草儿的家的位置且该点的序号为 **0** 。

只要把该点与所有始发站之间均建立一条边且距离为 **0** ，那么只要以点 **0** 为源点 调用一次 **dijkstra** 或 **spfa** 就可以了。我用的是 **spfa**，出现错误的原因可能是有的目的地是孤立的点（在这里指草儿无法到达的点 ， 即前面未出现过的点）。

5.2 题目代码

```
#include<bits/stdc++.h>
using namespace std ;
const int MAXN = 1005 ;
const int INF = 0x7fffffff ;
int vis[MAXN] ;          // 标记数组，确认城市是否出现过
struct Node
{
    int adj ;
    int dist ;
    Node *next ;
}* vert[MAXN];
//Node * vert[MAXN] ;
queue <int> q ;
int m  , st , dt ;
int dest[MAXN] ;
int ss[MAXN] ;           // 记录出发站的城市数目
int dd[MAXN] ;           // 记录终点站的城市数目
int dis[MAXN] ;
int inq[MAXN] ;
int sumc ;               // 记录出现的不同的城市数目
void spfa(int v0)
```

```
{
    Node * p ;
    int i ;
    for(i = 0 ; i <= sumc ; i ++)
    {
        dis[dest[i]] = INF ;
    }
    dis[0] = 0 ;
    while (!q.empty())
    {
        q.pop() ;
    }
    q.push(v0) ;
    inq[v0] ++ ;
    while (!q.empty())
    {
        int tmp = q.front() ;
        q.pop() ;
        inq[tmp] -- ;
        p = vert[tmp] ;
        while (p != NULL)
        {
            int td = p -> dist ;
            int tadj = p -> adj ;
            if(td + dis[tmp] < dis[tadj])
            {
               dis[tadj] = td + dis[tmp] ;
               if(inq[tadj] == 0)
               {
                   inq[tadj] ++ ;
                   q.push(tadj) ;
               }
            }
            p = p -> next ;
        }
    }
}
void dele()                              // 删除邻接表
{
    Node * p ;
    int i ;
    for(i = 0 ; i <= sumc ; i ++)
    {
        if(i == 0)
```

```
        p = vert[0] ;
        else
        p = vert[dest[i]] ;
        while (p != NULL)
        {
            vert[dest[i]] = p -> next ;
            delete p ;
            p = vert[dest[i]] ;
        }
    }
}
int main()
{
    while(cin>>m>>st>>dt)
    //while (scanf("%d%d%d" , &m , &st , &dt) != EOF)
    {
        memset(vis , 0 , sizeof(vis)) ;
        memset(vert , 0 , sizeof(vert)) ;
        memset(dest , 0 , sizeof(dest)) ;
        memset(dis , 0 , sizeof(dis)) ;
        memset(inq , 0 , sizeof(inq)) ;
        int i ;
        sumc = 0 ;
        Node * p ;
        for(i = 0 ; i < m ; i ++)
        {
            int a , b , w ;
            cin >> a >> b >> w ;
            if(!vis[a])
            {
                vis[a] = 1 ;
                sumc ++ ;
                dest[sumc] = a ;
            }
            if(!vis[b])
            {
                vis[b] = 1 ;
                sumc ++ ;
                dest[sumc] = b ;
            }
            p = new Node ;
            p -> adj = b ;
            p -> dist = w ;
            p -> next = vert[a] ;
```

```
            vert[a] = p ;

            p = new Node ;
            p -> adj = a ;
            p -> dist = w ;
            p -> next = vert[b] ;
            vert[b] = p ;
        }
        for( i = 0 ; i < st ; i ++)
        {

            cin>>ss[i];
            if(!vis[ss[i]])          // 这里也不要忘记判断
            {
                vis[ss[i]] = 1 ;
                sumc ++ ;
                dest[sumc] = ss[i] ;
            }
            p = new Node ;
            p -> adj = ss[i] ;
            p -> dist = 0 ;
            p -> next = vert[0] ;
            vert[0] = p ;

            p = new Node ;
            p -> adj = 0 ;
            p -> dist = 0 ;
            p -> next = vert[ss[i]] ;
            vert[ss[i]] = p ;
        }
        for( i = 0 ; i < dt ; i ++)
        {

            cin>>dd[i];
            if(!vis[dd[i]])       // 这里也不要忘记判断，终点站的城市可能第
                                  //一次出现
            {
                vis[dd[i]] = 1 ;
                sumc ++ ;
                dest[sumc] = dd[i] ;
            }
        }
        spfa(0) ;
        int min = INF ;
```

```
        for( i = 0 ; i < dt ; i ++)
        {
            if(min > dis[dd[i]])
            {
                min = dis[dd[i]] ;
            }
        }

        cout<<min<<endl;
        dele() ;
    }
    return 0 ;
}
```

6. 文化之旅

6.1 题目分析

本题我选择用 vector，所以加了许多 stl（优先队列，set）。

因为大多数情况下使用 priority_queue 实现的 Dijstra 算法是比基于邻接矩阵的 Dijstra 算法快的，所以没有使用邻接矩阵。

6.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
struct edge{
    int from,to,dis;
};
struct data{
    int d,u;
    set <int> r;       //原来想用 bool 存储，然而总是出问题
    bool operator < (const data &tmp) const{
        return d>tmp.d;
    }                  //自定义数值小的先出列
};
set <int> r1;
int a[102][102],c[102],d[102],k;
vector <edge> ed;     //存边
vector <int> g[102];  //存边的编号
priority_queue <data> h;
void pc(data q,int x) //标记当前使者会被哪些文化排斥
{
    q.r.insert(x);
    for(int j=1;j<=k;j++)
        if(a[j][x])
            q.r.insert(j);
}

int main()
{
    memset(d,127/3,sizeof(d));
    const int inf=d[0];
    int n,m,s,t,i,j,q1,q2,q3,q4=-1;
```

```
    cin>>n>>k>>m>>s>>t;
    for(i=1;i<=n;i++) cin>>c[i];
    for(i=1;i<=k;i++)
        for(j=1;j<=k;j++)
            cin>>a[i][j];
    for(i=1;i<=m;i++)
    {
        cin>>q1>>q2>>q3;
        ed.push_back((edge){q1,q2,q3});
        g[q1].push_back(++q4);
        ed.push_back((edge){q2,q1,q3});
        g[q2].push_back(++q4);
    }                                       //无向图所以要存两次
    d[s]=0;
    r1.insert(c[s]);
    for(j=1;j<=k;j++)
        if(a[j][c[s]])
            r1.insert(j);
    h.push((data){0,s,r1});
    while(!h.empty())
    {
        data x=h.top(); h.pop();
        int u=x.u,len=g[u].size();
        if(x.d!=d[u]) continue;
        for(i=0;i<len;i++)
        {
            edge &e=ed[g[u][i]];
            data p=x;
            if(!p.r.count(c[e.to]))      //不被排斥
                if(d[e.to]>d[u]+e.dis)   //可以走并且这种走法更优
                {
                    d[e.to]=d[u]+e.dis;
                    pc(p,c[e.to]);       //标记
                    h.push((data){d[e.to],e.to,p.r});   //入队
                }
        }
    }
    if(d[t]==inf) cout<<"-1"<<endl;
                                        //判断使者能否到达终点
    else cout<<d[t]<<endl;
    return 0;
}
```

7. 公交线路

7.1 题目分析

这类题统称最短路问题，其核心思想就是对于 A,B 点，能否找到 C 点使 A 到 C 的距离加上 C 到 B 的距离要小于 A 直接到 B 的距离。

对 A B C 三个点进行枚举：

首先假设 C 点为 1 ，A 点为 1， B 点为 1 。计算 1 点到 1 点的距离是否大于 1 点到 1 点加上 1 点到 1 点的距离，这显然是没有意义的，所以直接跳过。

接下来是 C 为 1， A 为 1 ，B 为 2，计算 1 点到 2 点的距离是否大于 1 点到 1 点加上 1 点到 2 点的距离，这同样是没有意义的，跳过。

跳过一些无意义的部分，我们直接来到第一个有意义的点：

C 为 1 ，A 为 2 ，B 为 3，计算 2 点到 3 点的距离是否大于 2 点到 1 点加上 1 点到 3 点的距离：

从邻接表或图可知，2 到 1 的距离是 3，1 到 3 的距离是 5,而 2 到 3 的距离为 inf，即无限远，所以 2 到 3 是可以以 1 点为中转站到达，当前的最短距离是 5+3=8，所以此时我们就可以更改邻接表上的数据 arr[2][3]=8;arr[3][2]=8。

接下来是 C=1，A=2，B=4；2 到 4 为 7，2 到 1 为 3，1 到 4 为 inf，所以这次不会产生变化。

以此类推，就能计算出各个点之间两两之间的最短距离。

另外，在最短路计算中常用 0x3f3f3f3f 来指代最大值或无限远，常用 memset(a,0x3f,sizeof a)来对整个初始数组赋上无限远的初值。

7.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1005;
int n, m, s, t, tot;
int map0[N][N];
int main() {
    cin>>n>>m>>s>>t;
```

```
    memset(map0, 0x3f, sizeof (map0));
    for (int i = 0; i <= n; i++) {
        map0[i][i] = 0;
    }
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin>>u>>v>>w;
        map0[u][v] = min(map0[u][v], w); //两个点可能存在多条道路，取最小
        map0[v][u] = min(map0[v][u], w);
    }
    for (int k = 1; k <= n; k++) {         //枚举所有的 C 点
        for (int i = 1; i <= n; i++) {     //枚举所有 A 点
            for (int j = 1; j <= n; j++) { //枚举所有 B 点
                if (map0[i][j] > map0[i][k] + map0[k][j]) {
                    map0[i][j] = map0[i][k] + map0[k][j];
                }
            }
        }
    }
    if (map0[s][t] == 0x3f3f3f3f) {
        cout<<"-1"<<endl;
    } else {
        cout<<map0[s][t]<<endl;
    }
}
```

8. 弗洛伊德

8.1 题目分析

使用 Floyd 算法求解即可。

思想如下：

第一步，遍历所有结点

第二步，每次以当前遍历的结点 k 作为中介点，查找是否存在最短路径

即下面的代码：

```
if (dis[i][k] != INF && dis[k][j] != INF && dis[i][k] + dis[k][j] < dis[i][j])
{
    dis[i][j] = dis[i][k] + dis[k][j];
}
```

第三步，遍历结束后，dis 中就存放的是任意两点之间的最短路径。

8.2 题目代码

```
#include <bits/stdc++.h>
using namespace std;
const int floyd_max_num = 200;
const int INF = 1000000000;
int dis[floyd_max_num][floyd_max_num];
void floyd(int n)                  //使用 Floyd 算法求解
{
    for (int k = 0; k < n; k++)
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (dis[i][k] != INF && dis[k][j] != INF && dis[i][k] +
dis[k][j] < dis[i][j])
                {
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
```

```
        }
    }
}
int main()
{
    int n;                          //表示共有 n 个结点
    cin >> n;
                                    //初始化距离矩阵
    fill(dis[0], dis[0] + floyd_max_num * floyd_max_num, INF);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> dis[i][j];
            if (dis[i][j] == 0 && i != j)
            {
                dis[i][j] = INF;
            }
        }
    }
    floyd(n);
    for (int i = 0; i < n; i++)  //输出答案
    {
        for (int j = 0; j < n; j++)
        {
            if (dis[i][j] == INF)
            {
                cout << -1 << " ";
            }
            else
            {
                cout << dis[i][j] << " ";
            }
        }
        cout << endl;
    }
    return 0;
}
```

9. 奖学金(reward)

9.1 题目分析

拓扑排序，要求我们对先遍历后处理的思想理解的要深刻。

另外，从下往上遍历时通过传参改变父节点值，尽可能减少空间和时间的开销。

9.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
int n,m,x,y;
const int maxn=10001;
int r[maxn];
int c[maxn];
int a[maxn][301];
int ans[maxn];
int main()
{
    while(cin>>n>>m)
    {
        memset(r,0,sizeof(r));          //标记入度
        memset(c,0,sizeof(c));          //边
        memset(a,0,sizeof(a));          //标记作用
        memset(ans,0,sizeof(ans));     //也是标记作用
        int tot=0;                     //计数 ，计算出度的数
        int money=0;//钱
        int k=0;//增量
        int t;//每次的度数为 0 的数
        while(m--)
        {
            cin>>x>>y;                  //y 到 x 的边
            r[x]++;                    //入度加 1；
            c[y]++;                    //链接边+1
            a[y][c[y]]=x;              //y 代表这个结点练的边，c[y]标记这是 y
//的第几个边，连到 x
        }
        while (tot<n)                 //没节点有全部删完 tot 代表已删的节点
        {
            t=0;
```

```
            for (int i=1;i<=n;i++)
              if (!r[i])       //如果 i 节点入度为 0，那么存储这个节点并删除
              {
                tot++;
                t++;           //0 的度数,也就是比上一层多的人数
                ans[t]=i;
                r[i]=maxn;    //删除结点
                money+=100;
              }
            if (!t)
                {
                    cout<<"impossible"<<endl;

                    break;
                }
            money+=k*t;       //对于每一个层次补齐差价，k 为每个人的差价
            k++;//加差价
            for (int i=1;i<=t;i++)            //点的遍历
            {
                for (int j=1;j<=c[ans[i]];j++)//点对应边的遍历
                {
                    r[a[ans[i]][j]]--;//入度减一，下次判断如果是 0 就出度
                }
            }
        }
        if(t)
         cout<<money<<endl;
    }
    return 0;
}
```