原创 图论模板

2019-10-17 11:01:12 _-Y-_-Y-_ 阅读数 4 文章标签: ACM 更多

最短路：SPAF

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 1000000
#define ll long long
#define INF 0x3f3f3f3f
struct node{
  int u,v,w;
  int next;
}edge[maxn];
int head[maxn];
ll dis[maxn];
int cnt,n,t;
void init(){
  memset(head,-1,sizeof head);
  memset(dis,INF,sizeof dis);
  cnt=0;
}
void add(int u, int v, int w, node edge[], int  &cnt, int *head){
  edge[cnt].u=u;
  edge[cnt].v=v;
  edge[cnt].w=w;
  edge[cnt].next=head[u];
  head[u]=cnt++;
}
void spfa(int st, ll *dis, int *head, node edge[]){
  dis[st]=0;
  queue<int> q;
  q.push(st);
  while(!q.empty()){
    int u=q.front();
    q.pop();
    for(int i = head[u]; ~i; i = edge[i].next ){
      int v=edge[i].v;
      if(dis[v] > dis[u] + (ll)edge[i].w){
        dis[v] = dis[u] + (ll)edge[i].w;
        q.push(v);
      }
    }
  }
  return ;
}
int main(){
  while(~scanf("%d %d", &t, &n)){
    init();
    int u,v,w;
    for(int i=0;i<t;i++){
      scanf("%d %d %d", &u, &v, &w);
      add(u, v, w, edge, cnt, head);
      add(v, u, w, edge, cnt, head);
    }
    spfa(1, dis, head, edge);
    cout<<dis[n]<<endl;
  }
  return 0;
}
```

最短路 ： 迪杰斯特拉（谜之优化，备用）

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <queue>
#include <cstdio>
#define INF 0x3f3f3f3f
#define maxn 1000000
using namespace std;
int n,t;
int d[maxn];//用于记录起点s到v的最短路
bool color[maxn];//用来记录访问状态
vector< pair<int,int> >mp[maxn];
void init(){
  memset(d,INF,sizeof(d));
  memset(color,0,sizeof(color));
  for(int i=0;i<=n;i++){
    mp[i].clear();
  }
}
void dijkstra(int st){
  priority_queue< pair<int,int> > p;//有限队列存存最小节点（默认优先级较大）
  d[st]=0;
  p.push(make_pair(0,st));
  while(!p.empty()){
    pair<int,int> f = p.top();
    p.pop();
    int u=f.second;
    color[u]=1;//取出最小值，若不是最短路则忽略
    if(d[u] < f.first*(-1)) continue;
    for(int j=0; j<mp[u].size(); j++){
      int v=mp[u][j].first;
      if(color[v]==1) continue;
      if(d[v]>d[u]+mp[u][j].second){
        d[v]=d[u]+mp[u][j].second;
        p.push(make_pair(d[v]*(-1),v));//  priority_queue（默认优先级较大）所以要*-1;
      }
    }
  }
}
int main(){
  while(~scanf("%d %d", &t, &n)){
    init();
    int u,v,w;
    for(int i=0;i<t;i++){
      scanf("%d %d %d", &u, &v, &w);
      mp[u].push_back(make_pair(v,w));
      mp[v].push_back(make_pair(u,w));
    }
    dijkstra(1);
    printf("%d\n", d[n]);
  }
  return 0;
}
```

多元最短路：Floyed-Warshall算法

```
#include <cstdio>
#include <algorithm>
#include <iostream>
using namespace std;
int mp[1005][1005];
int main(){
  int n,m,x;
scanf("%d %d %d", &n, &m, &x);
  int u,v,k;
  for(int i=0;i<=n;i++){
    for(int j=0;j<=n;j++){
      mp[i][j]=0x3f3f3f;
    }
  }
```

```
  for(int i=0;i<=n;i++){
    mp[i][i]=0;
  }
  for(int i=0;i<m;i++){
    scanf("%d %d %d", &u, &v, &k);
    mp[u][v]=k;
  }
  for(int k=1;k<=n;k++){
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        mp[i][j]=min(mp[i][k]+mp[k][j],mp[i][j]);
      }
    }
  }
  int ans=0;
  for(int i=1;i<=n;i++){
    ans=max(mp[i][x]+mp[x][i],ans);
  }
  printf("%d\n", ans);
  return 0;
}
```

无向图的最小生成树：Kruskal算法

```
#include <iostream>
#include <algorithm>
using namespace std;
#define maxn 100000
struct node{
  int u,v,w;
}edge[maxn];
int fa[maxn];
int cnt;
int n,m;
void init(){
  cnt=0;
}
void add(int u,int v,int w){
  edge[u].u=u;
  edge[u].v=v;
  edge[u].w=w;
}
bool cmp(node a,node b){
  return a.w<b.w;
}
int root(int x){
  return fa[x]==x?x:fa[x]=root(fa[x]);
}
void unite(int x,int y){
  x=root(x);
  y=root(y);
  if(x!=y) fa[y]=x;
}
bool ailke(int x,int y){
  return root(x)==root(y);
}
int Kruskal(){
  sort(edge,edge+m,cmp);
  for(int i=0;i<=n;i++) fa[i]=i;
  int ans=0,num=0;
  for(int i=0;i<m;i++){
    if(!ailke(edge[i].u,edge[i].v)){
      unite(edge[i].u,edge[i].v);
      ans+=edge[i].w;
      num++;
    }
    if(num==n-1) return ans;
  }
  return 0;
}
```

```
int main(){
  while(cin>>n&&n){
    init();
    cin>>m;
    for(int i=0;i<m;i++){
      cin>>edge[i].u>>edge[i].v>>edge[i].w;
    }
    cout<<Kruskal()<<endl;
  }
}
```

无向图的最小生成树：Prim算法（备用）

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>
#define maxn 110
#define INF 0x3f3f3f3f
using namespace std;
int n,t;
int m[maxn][maxn];
void init(){
  memset(m,INF,sizeof m);
}
int prim(){
  int ans=0;
  int min=INF;
  int d[maxn],color[maxn],b[maxn];
  for(int i=0;i<n;i++){
    color[i]=0;//记录改点是否遍历过
    d[i]=m[0][i];//记录初始每个顶点最短路径为到顶点的距离
    b[i]=0;//记录他的上一个节点
  }
  color[0]=1;
  for(int i=1;i<n;i++){
    min=INF;
    int u=-1;
    for(int j=0;j<n;j++){
      if(!color[j]&&d[j]<min){
        min=d[j];
        u=j;
      }
    }
    color[u]=1;
    //cout<<u+1<<" "<<b[u]+1<<endl;
    ans+=m[u][b[u]];
    for(int j=1;j<n;j++){
      if(!color[j]&&m[u][j]<d[j]){
        d[j]=m[u][j];
        b[j]=u;
      }
    }
  }
  return ans;
}
int main(){
  while(cin>>n&&n){
    cin>>t;
    init();
    int u,v,w;
    for(int i=0;i<t;i++){
      cin>>u>>v>>w;
      u--;
      v--;
      m[u][v]=min(m[u][v],w);
      m[v][u]=min(m[v][u],w);
    }
    cout<<prim()<<endl;
  }
```

```
  return 0;
}
```

有向图的最小生成树：最小树形图

```
/*
Command Network
POJ - 3164
https://vjudge.net/problem/POJ-3164
题意：在有向图中找出最小生成树（最小树形图）
解法：朱,刘算法
算法思路:
1：确定一个根
2：找到除根外每一个点的最小入边，若这些边构成了环，则缩环成点，并将环内的每一个点的其他入边都减去环内的入边
3：重复步骤2直到没有环出现（构成了树）。
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
using namespace std;
#define maxn 10000
#define INF 0x3f3f3f3f
int X[maxn],Y[maxn];
double IN[maxn];
int PRE[maxn];
int ID[maxn];
int VIS[maxn];
int n,m,cnt;
struct node{
  int u,v;
  double w;
}edge[maxn];
double dtc(int a,int b){
  return sqrt((double)((X[a]-X[b])*(X[a]-X[b])+(Y[a]-Y[b])*(Y[a]-Y[b])));
}
void add(int u,int v){
  edge[cnt].u=u;
  edge[cnt].v=v;
  edge[cnt++].w=dtc(u,v);
}
double zhuliu(int root){
  double ans=0;
  while(1){
    for(int i=1;i<=n;i++){
      IN[i]=INF;
    }
    for(int i=0;i<cnt;i++){
      if(edge[i].v==edge[i].u) continue;
      else if(IN[edge[i].v]>edge[i].w){
        PRE[edge[i].v]=edge[i].u;//记录每个点的最小入边
        IN[edge[i].v]=edge[i].w;//记录每个点的最小入边的边权
      }
    }
    for(int i=1;i<=n;i++){
      if(i!=root&&IN[i]==INF) return -1;
    }
    int num=0;
    memset(ID,0,sizeof ID);
    memset(VIS,0,sizeof VIS);
    //ID数组的初始化要与cnt记录的环数相匹配，这里ID用0初始化，则cnt从1开始记录，也可以用-1初始化，让cnt从0开始记录
    //初始化要注意，因为这里的节点是从1开始编号的，所以VIS数组中不会出现0，可以用0来初始化，若节点是从0开始编号，则VIS中会有0，不能用0初始化
    IN[root]=0;
    for(int i=1;i<=n;i++){
      ans+=IN[i];
      int v=i;
      while(!ID[v]&&v!=root&&VIS[v]!=i){
        VIS[v]=i;
        v=PRE[v];
      }
```

```
      if(v!=root&&!ID[v]){
        ID[v]=++num;
        for(int j=PRE[v];j!=v;j=PRE[j]){
          ID[j]=num;
        }
      }
    }
    if(!num) break;//没有环, 树成立
    for(int i=1;i<=n;i++){
      if(!ID[i]){
        ID[i]=++num;
      }
    }
    for(int i=0;i<m;i++){
      int x=edge[i].u,y=edge[i].v;
      edge[i].u=ID[x];
      edge[i].v=ID[y];
      if(ID[x]!=ID[y]){
        edge[i].w-=IN[y];
      }
    }
    n=num;
    root=ID[root];//维护新的点数目和根
  }
  return ans;
}
int main(){
  int u,v;
  while(~scanf("%d %d", &n, &m)){
    cnt=0;
    for(int i=1;i<=n;i++){
      scanf("%d%d", &X[i], &Y[i]);
    }
    for(int i=0;i<m;i++){
      scanf("%d%d", &u, &v);
      if(u!=v) add(u,v);
    }
    double ans=zhuliu(1);
    if (ans == -1) printf("poor snoopy\n");
    else printf("%.2f\n", ans);
  }
  return 0;
}
```

最大流：Dinic算法

```
/*
Dining
POJ - 3281
https://cn.vjudge.net/problem/POJ-3281
输入：
            第一行输入三个整数N, F, D
            接下来n行，每行先输入两个整数 Fi 和 Di，分别表示编号为 i 的牛喜欢的食物和饮料的数量，
            接下来的Fi个整数表示第i头牛喜欢的食物的编号，最后Di个整数表示第i头牛喜欢的饮料的编号。
输出：
            输出同时得到喜欢的食物和饮料的牛的数量的最大值。
样例输入：
        4 3 3
        2 2 1 2 3 1
        2 2 2 3 1 2
        2 2 1 3 1 2
        2 1 1 3 3
样例输出：
        3
解法：
        dinic算法
*/
#include <iostream>
#include <cstdio>
#include <algorithm>
```

```cpp
#include <cstring>
#include <queue>
using namespace std;
#define maxn 1000000
#define INF 0x3f3f3f3f
int n,m,k;
int a,b,c;
int B,E;
int dis[maxn];
struct node{
    int u,v,w;
    int next;
}edge[maxn];
int head[maxn];
int cnt;
void init(){
    memset(head,-1,sizeof head);
    cnt=0;
}
void add(int u,int v){
    edge[cnt].u=u;
    edge[cnt].v=v;
    edge[cnt].w=1;
    edge[cnt].next=head[u];
    head[u]=cnt++;

    edge[cnt].u=v;
    edge[cnt].v=u;
    edge[cnt].w=0;
    edge[cnt].next=head[v];
    head[v]=cnt++;
}
bool bfs(){
    memset(dis,-1,sizeof dis);
    queue<int> q;
    dis[B]=0;
    q.push(B);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];~i;i=edge[i].next){
            int v=edge[i].v;
            if(dis[v]==-1&&edge[i].w>0){
                dis[v]=dis[u]+1;
                q.push(v);
            }
        }
    }
    return dis[E]!=-1;
}
int dfs(int u,int w){
    int ans=0;
    if(u==E) return w;
    for(int i=head[u];~i;i=edge[i].next){
        if(edge[i].w>0&&dis[edge[i].v]==dis[u]+1){
            int num=min(edge[i].w,w-ans);
            num=dfs(edge[i].v,num);
            ans+=num;
            edge[i].w-=num;
            edge[i^1].w+=num;
        }
    }
    if(!ans) dis[u]=-2;
    return ans;
}
int dinic(){
    int ans=0;
    int num;
    while(bfs()){
        while(num=dfs(B,INF)) ans+=num;
    }
```

```
    return ans;
}
int main(){
    cin>>n>>m>>k;
    init();
    B=0;
    E=n+n+m+k+1;
    for(int i=1;i<=m;i++){
        add(B,n+i);
    }
    for(int i=1;i<=k;i++){
        add(n+m+i,E);
    }
    for(int i=1;i<=n;i++){
        add(i,n+m+k+i);
        cin>>a>>b;
        while(a--){
            cin>>c;
            add(n+c,i);
        }
        while(b--){
            cin>>c;
            add(n+m+k+i,n+m+c);
        }
    }
    printf("%d\n", dinic());
    return 0;
}
```

最大流：EK算法（备用）

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 1000
#define INF 0x3f3f3f
int cap[maxn][maxn];
int f[maxn][maxn];
int pre[maxn],p[maxn];
int k,m,n;
int EK(int s,int t){
  int v,u;
  int ans=0;
  queue<int> q;
  memset(f,0,sizeof(f));
  while(1){
    memset(p,0,sizeof(p));
    p[s]=INF;
    q.push(s);
    while(!q.empty()){
      u=q.front();
      q.pop();
      for(v=1 ;v<=n ;v++){
        if(!p[v]&&cap[u][v]>f[u][v]){
          pre[v]=u;
          q.push(v);
          p[v]=min(p[u],cap[u][v]-f[u][v]);
        }
      }
    }
    if(p[t]==0) break;
    for(u=t;u!=s;u=pre[u]){
      f[pre[u]][u] += p[t];
      f[u][pre[u]] -= p[t];
    }
    ans+=p[t];
  }
  return ans;
}
int main(){
  int a,b,c;
```

```
  while(~scanf("%d %d", &k, &n)){
    memset(cap,0,sizeof(cap));
    for(int i=0;i<k;i++){
      scanf("%d %d %d", &a, &b, &c);
      cap[a][b]+=c;
    }
    printf("%d\n", EK(1,n));
  }
  return 0;
}
```

最小费用流最大流：MCMF_SPAF

```
/*
【模板】最小费用最大流
P3381
https://www.luogu.org/problem/P3381
题面：
        如题，给出一个网络图，以及其源点和汇点，
        每条边已知其最大流量和单位流量费用，
        求出其网络最大流和在最大流情况下的最小费用。
输入：
        第一行包含四个正整数N、M、S、T，分别表示点的个数、
        有向边的个数、源点序号、汇点序号。
        接下来M行每行包含四个正整数ui、vi、wi、fi，
        表示第i条有向边从ui出发，到达vi，边权为wi（即该边最大流量为wi），
        单位流量的费用为fi。
输出：
        一行，包含两个整数，依次为最大流量和在最大流量情况下的最小费用。
样例输入：
            4 5 4 3
            4 2 30 2
            4 3 20 3
            2 3 20 1
            2 1 30 9
            1 3 40 5
样例输出：
            50 280
说明：
            第一条流为4-->3，流量为20，费用为3*20=60。
            第二条流为4-->2-->3，流量为20，费用为（2+1）*20=60。
            第三条流为4-->2-->1-->3，流量为10，费用为（2+9+5）*10=160。
            故最大流量为50，在此状况下最小费用为60+60+160=280。
            故输出50 280。
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 1000000
#define INF 0x3f3f3f3f
int n,m;
struct node{
    int u,v,w,c;
    int next;
}edge[maxn];
int head[maxn];
int per[maxn];
int dis[maxn];
int q[maxn];
int cnt;
int s,t,ans;
void init(){
    memset(head,-1,sizeof head);
    cnt=0;
}
void add(int u,int v,int w,int c){
    edge[cnt].u=u;
    edge[cnt].v=v;
    edge[cnt].w=w;
    edge[cnt].c=c;
    edge[cnt].next=head[u];
```

```
    head[u]=cnt++;

    edge[cnt].u=v;
    edge[cnt].v=u;
    edge[cnt].w=-w;
    edge[cnt].c=0;
    edge[cnt].next=head[v];
    head[v]=cnt++;
}
bool spfa(){
    memset(per,-1,sizeof per);
    memset(dis,INF,sizeof dis);
    dis[s]=0;
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];~i;i=edge[i].next){
            int v=edge[i].v;
            if(edge[i].c>0&&dis[u]+edge[i].w<dis[v]){
                dis[v]=dis[u]+edge[i].w;
                per[v]=i;
                q.push(v);
            }
        }
    }
    return dis[t]!=INF;
}
void MCMF(){
    int flow=0;
    int ans=0;
    while(spfa()){
        int u=t;
        int mini=INF;
        while(u!=s){
            if(edge[per[u]].c<mini){
                mini=edge[per[u]].c;
            }
            u=edge[per[u]].u;
        }
        flow+=mini;
        u=t;
        while(u!=s){
            edge[per[u]].c-=mini;
            edge[per[u]^1].c+=mini;
            u=edge[per[u]].u;
        }
        ans+=mini*dis[t];//mini 单位流量的代价，dis 表示流量
    }
    printf("%d %d\n" ,flow ,ans);
}
int main(){
    init();
    cin>>n>>m>>s>>t;
    int a,b,c,d;
    for(int i=0;i<m;i++){
        cin>>a>>b>>c>>d;
        add(a,b,d,c);
    }
    MCMF();
    return 0;
}
```

次小生成树：kruskal算法

```
#include<bits/stdc++.h>
using namespace std;
const int L=1e5+7;
const int inf=0x3f3f3f3f;
```

```
const int maxn=1000+7;
int father[maxn],n,m,num[maxn],nPos;//父节点（并查集），点数，边数，最小生成树点集 当前访问方位
struct node{
    int s,y,w;
}edge[L];//左端点，右端点，权值
void init(){//初始化并查集
    for(int i=0;i<=n;i++)
        father[i]=i;
}
int root(int x){//并查集，构造父节点
    return father[x]==x?x:father[x]=root(father[x]);
}
void unite(int x,int y){//并查集，合并两个联通图
    x=root(x);
    y=root(y);
    if(x!=y)
        father[y]=x;
}
int alike(int x,int y){//并查集，判断是否为同一连通图
    return root(x)==root(y);
}
int secondTree(int pos)//次小生成树
{
    init(); //初始化
    int sum=0,cnt=0;
    for(int i=0;i<m;i++)//对于删去边后的图进行最小生成树运算
    {
        if(cnt==n-1)
            break;
        if(i==pos)
            continue;
        if(!alike(edge[i].s,edge[i].y)){
            unite(edge[i].s,edge[i].y);
            sum+=edge[i].w;
            cnt++;
        }
    }
    return cnt!=n-1?-1:sum;//判断删除边后是否能构成最小生成树
}
int cmp(node a,node b){
    return a.w<b.w;
}
int kruskal(){//最小生成树
    init();
    sort(edge,edge+m,cmp);//对边进行权值排序
    int sum=0,cnt=0;
    for(int i=0;i<m;i++)//每次选择最小且未访问过的一条边
    {
        if(cnt==n-1)
            break;
        if(!alike(edge[i].s,edge[i].y)){
            unite(edge[i].s,edge[i].y);
            sum+=edge[i].w;
            cnt++;
            num[++nPos]=i;
        }
    }
    return cnt!=n-1?-1:sum;//判断边是否大于等于n-1，否则输出-1
}
void read(){//读入数据
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++)
        scanf("%d%d%d",&edge[i].s,&edge[i].y,&edge[i].w);
}
void solve(){//解决方案
    int Min=inf;
    nPos=0;
    int mst=kruskal();//最小生成树值
    if(mst==-1) {//没有最小生成树即输出-1
        printf("-1\n");
        return;
```

```
    }
    for(int i=1;i<=nPos;i++){//对最小生成树的每条边进行遍历，选择删边后的最小值
        int secmst=secondTree(num[i]);
        if(secmst!=-1)//若没有次小生成树输出-1
            Min=min(Min,secmst);
        }
    if(Min!=inf&&Min!=mst)
        printf("%d\n",Min);//输出结果
    else
        printf("-1\n");
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        read();//读入数据
        solve();//解决方案
    }
    return 0;
}
```

LCA：暴力求解

```
/*
DFS建树 + 普通搜索
思路：
建树：找到根节点，
然后记录每个节点的父亲节点
和该节点的的深度，
搜索：从两个节点开始向上找，
一直找的两个节点重合为止
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 10010
vector<int> tree[maxn];
int fa[maxn];
int deep[maxn];
bool find_root[maxn];
int root,n,q;
void dfs(int x){
    for(int i=0;i<tree[x].size();i++){
        int y=tree[x][i];
        deep[y]=deep[x]+1;
        fa[y]=x;
        dfs(y);
    }
}
void init(){
    for(int i=1;i<=n;i++){
        if(!find_root[i]){
            root=i;
            break;
        }
    }
    deep[root]=1;
    fa[root]=root;
    dfs(root);
}
int lca(int x,int y){
    while(deep[x]>deep[y]) x=fa[x];
    while(deep[x]<deep[y]) y=fa[y];
    while(x!=y){
        x=fa[x];
        y=fa[y];
    }
    return x;
}
int main(){
    scanf("%d %d", &n, &q);
```

```
    memset(find_root,false,sizeof find_root);
    int x,y;
    for(int i=1;i<n;i++){
        scanf("%d %d", &x, &y);
        tree[x].push_back(y);
        find_root[y]=true;
    }
    init();
    while(q--){
        scanf("%d %d", &u, &v);
        printf("%d\n", lca(u,v));
    }
    return 0;
}
```

LCA：倍增优化

```
/*
DFS+倍增优化
与暴力求解对比:
不必一步一步的向上寻找父亲节点，可以跳着寻找
思路:
倍增思想：任何数字都可以转换为二进制，
那么我存每一个节点的二的幂次位祖先，
是不是想要那个节点的第多少祖先可以很快查询出来
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 10010
vector<int> tree[maxn];
int anc[maxn][25];
int fa[maxn];
int deep[maxn];
bool find_root[maxn];
int root,n,q;
void dfs(int x){
    anc[x][0]=fa[x];
    for(int i=1;i<=22;i++){
        anc[x][i]=anc[anc[x][i-1]][i-1];//体现倍增
    }
    for(int i=0;i<tree[x].size();i++){
            int y=tree[x][i];
            fa[y]=x;
            deep[y]=deep[x]+1;
            dfs(y);
    }
}
void init(){
    for(int i=1;i<=n;i++){
        if(!find_root[i]){
            root=i;
            break;
        }
    }
    deep[root]=1;
    fa[root]=root;
    dfs(root);
}
int lca(int x,int y) {
    if (deep[x]<deep[y]) swap(x,y);
    for (int i=22;i>=0;i--) {
        if (deep[y]<=deep[anc[x][i]]) {
            x=anc[x][i];
        }
    }
    if (x==y) return x;
    for (int i=22;i>=0;i--){
        if (anc[x][i]!=anc[y][i]) {
            x=anc[x][i];
            y=anc[y][i];
```

```
        }
    }
    return anc[x][0];
}
int main(){
    scanf("%d %d", &n, &q);
    memset(find_root,false,sizeof find_root);
    int u,v;
    for(int i=1;i<n;i++){
        scanf("%d %d", &u, &v);
        tree[u].push_back(v);
        find_root[v]=true;
    }
    init();
    while(q--){
        scanf("%d %d", &u, &v);
        printf("%d\n", lca(u,v));
    }
    return 0;
}
```

LCA：欧拉序+ST表

```
/*
欧拉序+ST表
最近公共祖先是在该序列上两个点第一次出现的区间内深度最小的那个点
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 10010
vector<int> tree[maxn];
struct node{
    int deep;
    int m;
}a[maxn<<2],num[maxn<<2][25];//a存的是欧拉序，num是ST表
int first[maxn];//存每个节点第一次出现的cnt
int deep[maxn];//深度
bool find_root[maxn];
int root,n,q;
int cnt;
node calc(node a,node b){
    if(a.deep<b.deep) return a;
    return b;
}
void dfs(int x){
    first[x]=cnt;
    if(tree[x].size()==0){
        a[cnt].m=x;
        a[cnt++].deep=deep[x];
    }
    for(int i=0;i<tree[x].size();i++){
        int y=tree[x][i];
        deep[y]=deep[x]+1;
        a[cnt].m=x;
        a[cnt++].deep=deep[x];
        dfs(y);
    }
}
void ST(){
    int l=int(log((double)cnt)/log(2.0));
    for(int j=1;j<=l;j++){
        for(int i=1;i+(1<<(j-1))-1<=cnt;i++){
            num[i][j]=calc(num[i][j-1], num[i+(1<<(j-1))][j-1]);
        }
    }
}
void init(){
    for(int i=1;i<=n;i++){
        if(!find_root[i]){
            root=i;
```

```
            break;
        }
    }
    cnt=1;
    deep[root]=1;
    dfs(root);
    cnt--;
    for(int i=1;i<=cnt;i++){
        num[i][0]=a[i];
    }
    ST();
}
int rmq(int l,int r){
    l=first[l];
    r=first[r];
    if(l>r) swap(l,r);
    int k=int(log((double)(r-l+1))/log(2.0));
    return calc(num[l][k],num[r-(1<<k)+1][k]).m;
}
int main(){
    int x,y;
    scanf("%d %d", &n, &q);
    memset(find_root,false,sizeof find_root);
    for(int i=1;i<n;i++){
        scanf("%d %d", &x, &y);
        tree[x].push_back(y);
        find_root[y]=true;
    }
    init();
    while(q--){
        scanf("%d %d", &x, &y);
        printf("%d\n", rmq(x,y));
    }
    return 0;
}
```

Targan算法：割点

```
/*
样例输入：
            12 14
            1 2
            2 3
            3 4
            4 5
            5 6
            6 7
            7 8
            8 9
            8 10
            10 2
            9 4
            5 11
            11 12
            6 8
样例输出：
            11
            5
            2
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 1000100
struct node{
    int u,v;
    int next;
}edge[maxn];
int dfn[maxn],low[maxn];
int head[maxn];
int cnt,total;
```

```
void init(){
    memset(head,-1,sizeof head);
    cnt=total=0;
}
void add(int u,int v){
    edge[cnt].u=u;
    edge[cnt].v=v;
    edge[cnt].next=head[u];
    head[u]=cnt++;
}
void tarjan(int u,int fa){
    dfn[u]=low[u]=++total;
    int k=0;
    for(int i=head[u];~i;i=edge[i].next){
        int v=edge[i].v;
        if(!dfn[v]){
            k++;
            tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if((u!=fa&&dfn[u]<=low[v])||(u==fa&&k>1)){
                cout<<u<<endl;
            }
        }else if(fa!=v){
            low[u]=min(low[u],dfn[v]);
        }
    }
}
int n,m;
int main(){
    int u,v;
    scanf("%d %d", &n, &m);
    init();
    for(int i=0;i<m;i++){
        scanf("%d %d", &u, &v);
        add(u,v);
        add(v,u);
    }
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i,i);
    for(int i=1;i<=n;i++) cout<<i<<" "<<dfn[i]<<" "<<low[i]<<endl;
    return 0;
}
```

Tarjan算法：割边

```
/*
样例输入：
            12 14
            1 2
            2 3
            3 4
            4 5
            5 6
            6 7
            7 8
            8 9
            8 10
            10 2
            9 4
            5 11
            11 12
            6 8
样例输出：
            11 12
            5 11
            1 2
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 100010
struct node{
```

```
    int u,v;
    int next;
}edge[maxn];
int dfn[maxn],low[maxn];
int head[maxn];
int cnt,total;
void init(){
    memset(head,-1,sizeof head);
    cnt=0;
    total=0;
}
void add(int u,int v){
    edge[cnt].u=u;
    edge[cnt].v=v;
    edge[cnt].next=head[u];
    head[u]=cnt++;
}
void tarjan(int u,int fa){
    dfn[u]=low[u]=++total;
    for(int i=head[u];~i;i=edge[i].next){
        int v=edge[i].v;
        if(!dfn[v]){
            tarjan(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]){
                cout<<u<<" "<<v<<endl;
            }
        }else if(fa!=v){
            low[u]=min(low[u],dfn[v]);
        }
    }
}
int n,m;
int main(){
    int u,v;
    scanf("%d %d", &n, &m);
    init();
    for(int i=0;i<m;i++){
        scanf("%d %d", &u, &v);
        add(u,v);
        add(v,u);
    }
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i,i);
    return 0;
}
```

Tarjan算法：强连通分量

```
/*
样例输入:
6 8
1 2
1 3
3 4
3 5
4 1
2 4
5 6
4 6
样例输出:
6
5
2 4 3 1
*/
#include <bits/stdc++.h>
using namespace std;
#define maxn 100010
struct node{
    int u,v;
    int next;
```

```
}edge[maxn];
int dfn[maxn],low[maxn];
int head[maxn];
int cnt,total;
stack<int> st;
void init(){
    while(!st.empty()) st.pop();
    memset(head,-1,sizeof head);
    cnt=0;
    total=0;
}
void add(int u,int v){
    edge[cnt].u=u;
    edge[cnt].v=v;
    edge[cnt].next=head[u];
    head[u]=cnt++;
}
void tarjan(int u){
    dfn[u]=low[u]=++total;
    st.push(u);
    for(int i=head[u];~i;i=edge[i].next){
        int v=edge[i].v;
        if(!dfn[v]){
            tarjan(v);
            low[u]=min(low[u],low[v]);//回溯
        }else {
            low[u]=min(low[u],dfn[v]);//更新，记录v可以不通过u而到达u的祖先节点
        }
    }
    if(low[u]==dfn[u]){
            while(1){
                    int t=st.top();
          st.pop();
                    cout<<t<<" ";
          if(t==u) break;
            }
            cout<<endl;
    }
    return ;
}
int n,m;
int main(){
    int u,v;
    scanf("%d %d", &n, &m);
    init();
    for(int i=0;i<m;i++){
        scanf("%d %d", &u, &v);
        add(u,v);
    }
    for(int i=1;i<=n;i++) if(!dfn[i]) tarjan(i);
    return 0;
}
```

BFS

```
/*
Robot
POJ - 1376
https://cn.vjudge.net/problem/POJ-1376
题意:
      求机器人从起点到终点的一条用时最短的路径，机器人可以执行如下步骤
      向机器人面对的方向走1,2,3步
      向左转，向右转
      每个步骤耗时为1
样例输入:
     9 10
     0 0 0 0 0 0 1 0 0 0
     0 0 0 0 0 0 0 0 1 0
     0 0 0 1 0 0 0 0 0 0
     0 0 1 0 0 0 0 0 0 0
```

```
      0 0 0 0 0 0 1 0 0 0
      0 0 0 0 0 1 0 0 0 0
      0 0 0 1 1 0 0 0 0 0
      0 0 0 0 0 0 0 0 0 0
      1 0 0 0 0 0 0 0 1 0
      7 2 2 7 south
      0 0
样例输出:
      12
思路:
      这道题的坑
      只能向左转向右转，不能向后转
      若第一步走不了，第二步和第三步一定走不了
      机器人很胖，不能走边界
      障碍物是障碍物，路线是路线
      有可能起始位置有障碍物
      有可能终点位置有障碍物
      若没有路径一定要跳出来
      障碍物不可以走，但走过的路还是可以走的
      有可能起始位置就是终点位置
*/
#include <cstdio>
#include <cstring>
using namespace std;
int bx,by,ex,ey,bs,n,m;
char bf[10];
int mp[100][100][4];
int ans;
void bl(int x,int y,int z){
    if(z==0){//向这个方向前进1,2,3步，若第一步有障碍，则1,2,3走不了
        if(x-1>=0&&mp[x-1][y][z]==0)                    mp[x-1][y][z]=ans+1;
        if(x-2>=0&&mp[x-2][y][z]==0&&mp[x-1][y][z]!=-1) mp[x-2][y][z]=ans+1;
        if(x-3>=0&&mp[x-3][y][z]==0&&mp[x-1][y][z]!=-1) mp[x-3][y][z]=ans+1;
    }
    if(z==2){
        if(x+1<=n&&mp[x+1][y][z]==0)                    mp[x+1][y][z]=ans+1;
        if(x+2<=n&&mp[x+2][y][z]==0&&mp[x+1][y][z]!=-1) mp[x+2][y][z]=ans+1;
        if(x+3<=n&&mp[x+3][y][z]==0&&mp[x+1][y][z]!=-1) mp[x+3][y][z]=ans+1;
    }
    if(z==1){
        if(y-1>=0&&mp[x][y-1][z]==0)                    mp[x][y-1][z]=ans+1;
        if(y-2>=0&&mp[x][y-2][z]==0&&mp[x][y-1][z]!=-1) mp[x][y-2][z]=ans+1;
        if(y-3>=0&&mp[x][y-3][z]==0&&mp[x][y-1][z]!=-1) mp[x][y-3][z]=ans+1;
    }
    if(z==3){
        if(y+1<=m&&mp[x][y+1][z]==0)                    mp[x][y+1][z]=ans+1;
        if(y+2<=m&&mp[x][y+2][z]==0&&mp[x][y+1][z]!=-1) mp[x][y+2][z]=ans+1;
        if(y+3<=m&&mp[x][y+3][z]==0&&mp[x][y+1][z]!=-1) mp[x][y+3][z]=ans+1;
    }
    if(mp[x][y][(z+1+4)%4]==0) mp[x][y][(z+1+4)%4]=ans+1;//转向
    if(mp[x][y][(z-1+4)%4]==0) mp[x][y][(z-1+4)%4]=ans+1;
}
int bfs(){
    for(int i=0;i<4;i++){//判断是否找到终点
        if(mp[ex][ey][i]!=0){
            return mp[ex][ey][i];
        }
    }
    int a=0;
    ans++;
    for(int i=0;i<=n;i++){//找下一步所有可能的情况并记录最小步数
        for(int j=0;j<=m;j++){
            for(int k=0;k<4;k++){
                if(ans==mp[i][j][k]){
                    bl(i,j,k);
                    a=1;
                }
            }
        }
    }
    if(a==0)//从起点遍历完都没有找到终点
```

```
    return 0;
    bfs();//这是一个循环，不断的找下步可能的情况，直到找到终点或者遍历完
}
int main() {
    int a;
    while(~scanf("%d %d", &n, &m)&&(n||m)){
        memset(mp,0,sizeof(mp));
        ans=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                scanf("%d", &a);
                if(a==0) for(int k=0;k<4;k++){//这里是将点转化为路线
                    if(mp[i][j][k]==0)     mp[i][j][k]=0;
                    if(mp[i+1][j][k]==0)   mp[i+1][j][k]=0;
                    if(mp[i+1][j+1][k]==0) mp[i+1][j+1][k]=0;
                    if(mp[i][j+1][k]==0)   mp[i][j+1][k]=0;
                }
                else     for(int k=0;k<4;k++) mp[i][j][k]=-1,mp[i+1][j][k]=-1,mp[i+1][j+1][k]=-1,mp[i][j+1][k]=-1;
            }
        }
        scanf("%d %d %d %d %s", &bx, &by, &ex, &ey, bf);
        for(int i=0;i<=m;i++){//处理边界，机器人太胖，不能走边界
            for(int k=0;k<4;k++){
                mp[0][i][k]=-1;
                mp[n][i][k]=-1;
            }
        }
        for(int i=0;i<=n;i++){
            for(int k=0;k<4;k++){
                mp[i][0][k]=-1;
                mp[i][m][k]=-1;
            }
        }
        if(bf[0]=='n') bs=0;//初始方向
        if(bf[0]=='s') bs=2;
        if(bf[0]=='w') bs=1;
        if(bf[0]=='e') bs=3;
        mp[bx][by][bs]=1;
        if(mp[ex][ey][0]==-1||mp[bx][by][0]==-1)//有可能开始或终点处有障碍物
        printf("-1\n");
        else
        printf("%d\n", bfs()-1);
    }
    return 0;
}
```

DFS

```
/*
Oil Deposits
HDU - 1241
https://cn.vjudge.net/problem/HDU-1241
输入
            输入可能有多个矩形区域（即可能有多组测试）。
            每个矩形区域的起始行包含 n 和 m，表示行和列的数量，1<=n,m<=100，
            如果 m =0 表示输入的结束，接下来是n行，每行m个字符。
            每个字符对应一个小方格，并且要么是 '*'，代表没有油，要么是 '@'，表示有油。
输出
            对于每一个矩形区域，输出油藏的数量。
            两个小方格是相邻的，当且仅当他们水平或者垂直或者对角线相邻（即8个方向）。
样例输入：
            1 1
            *
            3 5
            *@*@*
            **@**
            *@*@*
            1 8
            @@****@*
```

```
            5 5
            ****@
            *@@*@
            *@**@
            @@@*@
            @@**@
            0 0
样例输出:
            0
            1
            2
            2
*/
#include <bits/stdc++.h>
using namespace std;
char mp[105][105];
int n,m;
int dfs(int x,int y){
    if(mp[x][y]=='@'&&x<n&&x>=0&&y<m&&y>=0){//判断边界
        mp[x][y]='*';
        dfs(x+1,y),dfs(x-1,y),dfs(x,y+1),dfs(x,y-1);//向八个方向搜索
        dfs(x+1,y+1),dfs(x-1,y-1),dfs(x-1,y+1),dfs(x+1,y-1);
        return 1;
    }
    return 0;
}
int main() {
    while(~scanf("%d %d", &n, &m)&&(m||n)){
        memset(mp,0,sizeof(mp));
        for(int i=0;i<n;i++){
                scanf("%s", mp[i]);
        }
        int ans=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                ans+=dfs(i,j);
            }
        }
        printf("%d\n", ans);
    }
    return 0;
}

```

文章最后发布于: 2019-10-17 11:0