登录/创建账户

条目 讨论 编辑 历史

导航

- 首页
- 社区主页
- 当前事件
- 最近更改
- 随机页面
- 使用帮助
- NOCOW地图
- 新手试练场

搜索

进入 搜索

工具箱

- 链入页面
- 链出更改
- 特殊页面
- 可打印版
- 永久链接

为防止广告，目前nocow只有登录用户能够创建新页面。如要创建页面请先登录/注册（新用户需要等待1个小时才能正常使用该功能）。

Dinic算法

(跳转自Dinic)

算法介绍 [编辑]

层次图 [编辑]

层次图，就是把原图中的点按照到到源的距离分“层”，只保留不同层之间的边的图。

算法流程 [编辑]

1. 根据残量网络计算层次图。
2. 在层次图中使用DFS进行增广直到不存在增广路
3. 重复以上步骤直到无法增广

时间复杂度 [编辑]

$O(n^2 * m)$

代码实现 [编辑]

递归实现1 [编辑]

```
/* 版本已更新 by_Roney*/
/*可以在HDU 3549 Flow Problem 上AC的*/
#include<iostream>
#include<cstdio>
#include<queue>
#include<cstring>
using namespace std;
const int N=20;
const int INF=0x3f3f3f3f;
int s[N][N];//记录图的邻接矩阵
int d[N];//记录图中各点的层次
int n,m;
int min(int a,int b)
{
        return a<b?a:b;
}
bool bfs()
{
        queue<int>Q;
        memset(d,-1,sizeof(d));//此处初始化要特别注意，以上版本的初始化就存在很大问题
        d[1]=0;//如果处理不慎就很容易死循环
        Q.push(1);
        while(!Q.empty()){
                int v=Q.front();Q.pop();
                for(int i=1;i<=n;i++){
                        if(d[i]==-1&&s[v][i]){//此处应是s[v][i]!=0,而不是以上版本中的s[v][i]>0,因
为dfs是可能会走错，这样可以对其进行修正。
                                d[i]=d[v]+1;
                                Q.push(i);
                        }
                }
        }
        return d[n]!=-1;
}
int dfs(int v,int cur_flow)
{
        int dt=cur_flow;
        if(v==n)return cur_flow;
        for(int i=1;i<=n;i++){
                if(s[v][i]>0&&d[v]+1==d[i]){
                        int flow=dfs(i,min(dt,s[v][i]));
                        s[v][i]-=flow;
                        s[i][v]+=flow;
                        dt-=flow;
                }
        }
        return cur_flow-dt;
}
int dinic()
{
        int cur_flow,ans=0;
        while(bfs()){//一次bfs可以找到几条增广路
                while(cur_flow=dfs(1,INF))
                        ans+=cur_flow;
        }
        return ans;
}
int main()
{
        int t,i,cas=0,u,v,w;
        scanf("%d",&t);
        while(t--){
                memset(s,0,sizeof(s));
                scanf("%d %d",&n,&m);
                for(i=1;i<=m;i++){
                        scanf("%d %d %d",&u,&v,&w);
                        if(u==v)continue;
                        s[u][v]+=w;
                }
                printf("Case %d: %d\n",++cas,dinic());
        }
        return 0;
}

===递归实现2===
<source lang="cpp">
//Dinic by fjxmyzwd

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <climits>

using namespace std;

const int maxn = 301, maxm = maxn*maxn;
int n, m, s, t, w, e = 0, l, r;
int head[maxn], head2[maxn], next[maxm], to[maxm], cap[maxm],
    op[maxm], d[maxn], Q[maxn];

inline int fmin(int a, int b) { return a < b ? a : b; }

bool build()
{
    int x, y;
    memset(d, -1, sizeof(d));
    l = 0; r = 1;
    Q[++l] = 1;
    head2[1] = head[1];
    d[1] = 0;
    while(l >= r)
    {
        x = Q[r++];
        for(int i = head2[x]; i; i = next[i])
        {
            y = to[i];
            if(cap[i] && d[y] == -1)
            {
                d[y] = d[x] + 1;
                head2[y] = head[y];
                if(y == n) return true;
                Q[++l] = y;
            }
        }
    }
    return false;
}

int find(int now, int low = INT_MAX)
{
    int ret = 0;
    if(now == n) return low;
    for(int i = head2[now]; i; i = next[i])
    {
        int y = to[i];
        if( (cap[i]) && (d[y] == d[now] + 1) && (ret=find(y,min(low,cap[i]))))
        {
            cap[i] -= ret;
            cap[op[i]] += ret;
            return ret;
        }
    }
    return 0;
}

int main()
{
    memset(next, 0, sizeof(next));
    memset(cap, 0, sizeof(cap));

    freopen("data.in","r",stdin);
    freopen("data.out","w",stdout);
    scanf("%d%d",&m,&n);
    for(int i = 1; i <= m; i++)
    {
        scanf("%d%d%d",&s,&t,&w);
        to[++e] = t; next[e] = head[s]; head[s]=e;
        cap[e]+=w;
        op[e]=++e; to[e]=s; next[e] = head[t]; head[t]=e;
        cap[e]=0;
    }

    int ans = 0, flow = 0;
    while(build())
     while(flow=find(1))
       ans+=flow;
    printf("%d",ans);
    return 0;
}
```

```
 //网络Dicnic（单向） By_zsyzgu
 注意！！
 我的Dicnic是向楼上递归实现2学的，只用了1个月，结果在一场比赛中付出了血的代价。
 （楼上算法易超时的原因是1.正宗Dicnic是每次增广多次，而楼上只有1次；2.楼上find()中，一个增广不通的点可能被访问多次。）

#include<stdio.h>
#include<string.h>
#include<iostream>

using namespace std;

const int maxn=1100;
const int maxm=110000;
const int INF=2000000000;

int n,m;
int S,T;

int tot=0;
int d[maxn];
int first[maxn];
int next[maxm];
int opp[maxm];
int line[maxm];
int value[maxm];
int work[maxn];

int link(int x,int y,int p)
{
    tot++; line[tot]=y; value[tot]=p; opp[tot]=tot+1; next[tot]=first[x]; first[x]=tot;
    tot++; line[tot]=x; value[tot]=0; opp[tot]=tot-1; next[tot]=first[y]; first[y]=tot;
}

void init()
{
     scanf("%d%d",&m,&n);

     S=1,T=n;

     for (int i=1;i<=m;i++)
     {
         int x,y,p; scanf("%d%d%d",&x,&y,&p);

         link(x,y,p);
     }
}

bool build()
{
     int tq=0,q[maxn];
     memset(q,0,sizeof(q));
     memset(d,-1,sizeof(d));

     q[++tq]=S;
     d[S]=0;

     for (int i=1;i<=tq;i++)
     {
         int t=q[i];

         for (int num=first[t];num;num=next[num])
         {
             int nt=line[num];

             if (d[nt]==-1 && value[num]>0)
             {
                d[nt]=d[t]+1;
                q[++tq]=nt;

                if (nt==T) return true;
             }
         }
     }

     return false;
}

int find(int t,int v)
{
    if (t==T || v==0) return v;

    int ans=0;

    for (int &num=work[t];num;num=next[num])
    {
        int nt=line[num],flow;

        if (d[nt]==d[t]+1 && (flow=find(nt,min(v,value[num]))))
        {
           value[num]-=flow;
           value[opp[num]]+=flow;
           v-=flow;
           ans+=flow;
           if (v==0) break;
        }
    }

    return ans;
}

void doit()
{
     int ans=0;

     while (build())
     {
           memcpy(work,first,sizeof(work));
           ans+=find(S,INF);
     }

     printf("%d\n",ans);
}

int main()
{
    init();
    doit();
    return 0;
}
```

注意，此代码貌似有点点问题

pascal

[编辑]

版本1

[编辑]

```
program dinic;

const
 maxp=1000;

type
 lx=array[1..maxp] of longint;

var
 lu:lx;
 a,b:array[1..maxp,0..maxp] of longint;
 d:array[1..maxp] of integer;
 v:array[1..maxp] of boolean;
 dist:array[1..maxp] of longint;
 head,tail:longint;
 t:array[1..maxp] of boolean;
 sum,ans,x,y,s,i,p,j,k,m,n:longint;
 q,c:boolean;

procedure spfa(s:longint);{用spfa进行分层}
var
 i,j,now,sum:longint;
  begin
  fillchar(d,sizeof(d),0);
  fillchar(v,sizeof(v),false);
  for j:= 1 to n do dist[ j ]:=maxlongint;
  dist[s]:=0; v[s]:=true; d[1]:=s;
  head:=1; tail:=1;
  while head<=tail do
    begin
    now:= d[head];
    for i:= 1 to b[now,0] do
      if dist[b[now,i]]>dist[now]+1 then //because a[now,i] is 1!
        begin
        dist[b[now,i]]:= dist[now]+1;
        if not v[b[now,i]] then
          begin
          inc(tail);
          d[tail]:=b[now,i];
          v[b[now,i]]:=true;
        end;
      end;
    v[now]:=false;
    inc(head);
  end;
end;

procedure dfs(x,d:longint);// (dfs找可增广路)
var
 i:longint;
  begin
  lu[d]:=x;
  t[x]:=true;
  if x=n then begin c:=false;s:=d;end;
  for i:=1 to n do if (a[x,i]>0)and c and (not t[i])and (dist[x]<=dist[i]){就这个是进行剪枝的} then dfs(i,d+1);
end;

procedure expand(l:lx;len:longint);{进行增广，和我前面说的费用流几乎一模一样}
var
 i,j,k:longint;
  begin
  k:=maxlongint;
  for i:=1 to len do if k>l[i] then k:=l[i];
  sum:=k;
  for i:=2 to len do
    begin
    dec(a[l[i-1],l[i]],k);
    inc(a[l[i],l[i-1]],k);
   end;
end;

  begin
  read(n,m);
  for i:=1 to m do
    begin
    read(x,y,k);
    a[x,y]:=k;
    inc(b[x,0]);
    b[x,b[x,0]]:=y;
  end;
  c:=false;
  while true do
    begin
    spfa(1);
    for i:=1 to n do t[i]:=false;
    k:=maxlongint;
    c:=true;
    dfs(1,1);
    if c then break;{如果没找到那么就退出}
    expand(lu,s);
    inc(ans,sum);
  end;
  writeln(ans);
end.
```

版本2

[编辑]

```
program poj1273;
//By HT http://www.cnblogs.com/htfy/archive/2012/02/15/2353147.html
Const maxn=400;

Var
 a,flow:array[0..maxn,0..maxn] of longint;
 last,dt:array[0..maxn] of longint;
 i,n,m,st,ed,w,ans,t:longint;

Function min(a,b:longint):longint;
  begin
  if a<b then exit(a) else exit(b);
end;

Procedure spfa;
var
 v:array[0..maxn] of boolean;
 q:array[0..maxn] of longint;
 close,open,x,i:longint;
  begin
  fillchar(dt,sizeof(dt),$7f);
  fillchar(v,sizeof(v),false);
  close:=0;open:=1;
  dt[1]:=1;
  q[1]:=1;
  v[1]:=true;
  while close<>open do
    begin
    close:=close mod n+1;
    x:=q[close];
    for i:=1 to n do
      if (a[x,i]>flow[x,i]) then
        if dt[x]+1<dt[i] then
          begin
          dt[i]:=dt[x]+1;
          last[i]:=x;
          if not v[i] then
            begin
            inc(open);
            q[open]:=i;
            v[i]:=true;
          end;
        end;
  end;
end;

Procedure adddelta;
var
 delta,q,w:longint;
  begin
  q:=abs(last[n]);w:=n;delta:=maxlongint;
  while w<>1 do
    begin
    delta:=min(delta,a[q,w]-flow[q,w]);
    w:=abs(last[w]);
    q:=abs(last[q]);
  end;
  inc(ans,delta);
  q:=abs(last[n]);w:=n;
  while w<>1 do
```

```
    begin
    inc(flow[q,w],delta);
    flow[w,q]:=-flow[q,w];
    w:=abs(last[w]);
    q:=abs(last[q]);
  end;
end;

Procedure netflow;
  begin
  fillchar(flow,sizeof(flow),0);
  fillchar(last,sizeof(last),0);
  spfa;
  while last[n]<>0 do
    begin
    adddelta;
    fillchar(last,sizeof(last),0);
    spfa;
  end;
end;

  begin
while not eof do begin
  readln(m,n);
  fillchar(a,sizeof(a),0);
  for i:=1 to m do
    begin
    readln(st,ed,w);
    inc(a[st,ed],w);
  end;
  ans:=0;
  netflow;
  writeln(ans);
 end;
end.
```

非递归实现 [编辑]

```
/*
Author: Alchemist
Website: http://www.n8lm.cn
以上写的最大流在计算分层网络时都用了一遍宽搜，而这一遍宽搜代价是很大的，下面采用标记法真正实现一次计算分层网络，找增广路顺便更新标记。并且非递归。
*/
typedef pair<int,int> pii;
vector<pii> adj[maxn];
int c[maxn][maxn] = {0};
int maxflow;

void dinic(int s,int t)
{
        int pre[maxn];
        int d[maxn];

        int v, vd, maxf;

        int q[maxn], head, tail;
        bool vis[maxn] = {0};
        head = 1;
        tail = 0;
        q[tail] = t;
        d[t] = 0;
        vis[t] = 1;
        while(tail < head)
        {
                v = q[tail];
                for(int i = 0; i < adj[v].size(); i ++)
                        if(!vis[adj[v][i].first])
                        {
                                vis[adj[v][i].first] = 1;
                                d[adj[v][i].first] = d[v] + 1;
                                q[head ++] = adj[v][i].first;
                        }
                tail ++;
        }

    maxflow = 0;
    maxf = 0x7fffffff;

        v = s;
        while(d[s] < n)
        {
            bool isc = 0;
                for(int i = 0; i < adj[v].size(); i ++)
                {
                        vd = adj[v][i].first;
                        if(c[v][vd] > 0 && d[v] == d[vd] + 1)
                        {
                                maxf = min(c[v][vd], maxf);
                                pre[vd] = v;
                                v = vd;
                                if(v == t)
                                {
                    while(v != s)
                                        {
                                                c[pre[v]][v] -= maxf;
                                                c[v][pre[v]] += maxf;
                                                v = pre[v];
                                        }
                                        maxflow += maxf;
                                        maxf = 0x7fffffff;
                                }
                                isc = 1;
                                break;
                        }
                }
        if(isc)
            continue;
                int mind = 0x7fffffff;
                for(int i = 0; i < adj[v].size(); i ++)
                {
                        vd = adj[v][i].first;
                        if(c[v][vd] > 0)
                                mind = min(mind, d[vd] + 1);
                }

                if(mind < 0x7fffffff)
                {
                        d[v] = mind;
                }
                else
                {
                        d[v] = n;
                        v = pre[v];
                }
        }
}
```

```
/**
 *有向图
 *By: MiYu
 *MyBlog: http://www.baiyun.me
 *一开始看论文学得MPLA， 一直超时，也就是一楼的算法，大家把它作为DINIC的过度算法学习吧
 *下面是自己对MPLA， EK， DINIC 的具体实现，互相学习
 **/

#define MEM(x,y) memset(x,y,sizeof(x))
#define MAXFLOW 9527
const int MN = 16, INF = 0x7ffffff;
int N, M, T, x, y, v, ca = 1;
int graph[MN][MN];
struct Base {
    int g[MN][MN], flow[MN][MN];/*图 和 流网络*/
    int q[MN], tail, top; /*队列*/
    int h[MN]; /*高度*/
    int s, t;  /*起点和终点*/
    int path[MN]; /*增广路径记录*/
    virtual void init () {}
    virtual int run (int ss = 0, int tt = 0, int res = 0) {return res;}
}*base;
/**
 *这个MPLA算法只能放这看看了，可能是自己写挫了
 *效率及其低下，也没有EK代码简单，大概只能作为
 *学习DINIC算法的过度知识来学习，也就是它的层次图思想
 *网上也没找到其他的资料，谁有具体代码实现， 麻烦发一份给我 baiyun@innlab.net
 *无限感激。。。
 **/
struct MPLA : public Base{//最短路径增值算法
    void init () { /*初始化操作*/
        memcpy ( g, graph, sizeof ( graph ) );
        MEM(flow,0);
    }
    bool getHei () {/*计算各点距S的高度值，也就是以s为树跟节点时各节点的深度*/
        tail = top = 0;
        MEM(h,-1);
        q[top++] = s;
        h[s] = 0;
        while ( top != tail ) {
            int u = q[tail++]; if ( tail >= MN ) tail -= MN;
            for ( int i = 1; i <= N; ++ i ) {
                if ( h[i] == -1 && g[u][i] > 0 ) {
                    h[i] = h[u] + 1;
                    q[top++] = i; if ( top >= MN ) top-= MN;
                }
            }
        }
        return h[t] != -1; /*终点高度为0， 则已经没有增广路*/
    }
    int adjNet ( int adj, int *p ) {/*更新残留网络*/
        for ( int i = t; i != s; i = p[i] ) {
            adj = min ( adj, g[p[i]][i] );
        }
        for ( int i = t; i != s; i = p[i] ) {
            g[p[i]][i] -= adj;
            if ( g[p[i]][i] == 0 ) h[i] = N+N;//
            g[i][p[i]] += adj;
        }   return adj;
    }
    int mpla () {
        int ss = s, ee, mi = 0;/*mx记录路径最小流量*/
        MEM(path,0); /*初始化路径*/
        tail = top = 0;
        q[top++] = s;
        while ( tail != top ) {
            ss = q[tail++]; if ( tail >= MN ) tail -= MN;
            if ( ss == t ) {
                mi += adjNet ( INF, path);
                break;
            }
            for ( ee = 1; ee <= N; ++ ee ) {
                if ( h[ss] + 1 == h[ee] ) {/*找一条增广路*/
                    path[ee] = ss;
                    q[top++] = ee; if ( top >= MN ) top-= MN;
                }
            }
        }
        return mi;
    }
    int run ( int ss = 0, int tt = 0, int res = 0 ) {
        s = ss, t = tt;
        while ( getHei () ) {/*对残留网络重新计算高度值*/
            while ( v = mpla () ) res += v;   /*继续搜索增广路经*/
        }
        return res;
    }
}mpla;
/**
 *不得不说EK确实是网络流中很容易理解的算法， 而且实现也很简单
 **/
struct EK : public Base{//EK
    void init () { /*初始化操作*/
        memcpy ( g, graph, sizeof ( graph ) );
        MEM(flow,0);
    }
    bool getPath () {/*搜索增广路径*/
        tail = top = 0;
        MEM(path,0);
        q[top++] = s;
        path[s] = N;
        while ( top != tail ) {
            int u = q[tail++]; if ( tail >= MN ) tail -= MN;
            if ( u == t ) return true; /*找到一条路， 返回*/
            for ( int i = 1; i <= N; ++ i ) {
                if ( path[i] == 0 && g[u][i] > 0 ) {
                    path[i] = u;
                    q[top++] = i; if ( top >= MN ) top-= MN;
                }
            }
        }
        path[s] = 0;
        return path[t] != 0; /*终点不在路径内， 则已经没有增广路*/
    }
    int adjNet ( int adj, int *p ) { /*对残留网络进行更新*/
        for ( int i = t; i != s; i = p[i] ) {
            adj = min ( adj, g[p[i]][i] );
        }
        for ( int i = t; i != s; i = p[i] ) {
            g[p[i]][i] -= adj;
            g[i][p[i]] += adj;/*更新反向流， 保持流量平衡*/
        }   return adj;
    }
    int run ( int ss = 0, int tt = 0, int res = 0 ) { /*执行算法主过程*/
        s = ss, t = tt;
        while ( getPath () ) {/*搜索增广路径*/
            res += adjNet (INF, path);   /*更新残留网络和最大容量*/
        }
        return res;
    }
```

```
}ek;
struct DINIC : public Base{
    void init () { /*初始化操作*/
        memcpy ( g, graph, sizeof ( graph ) );
        MEM(flow,0);
    }
    bool getHei () {/*计算各点距S的高度值，也就是以s为树跟节点时各节点的深度*/
        tail = top = 0;
        MEM(h,-1);
        q[top++] = s;
        h[s] = 0;
        while ( top != tail ) {
            int u = q[tail++]; if ( tail >= MN ) tail -= MN;
            for ( int i = 1; i <= N; ++ i ) {
                if ( h[i] == -1 && g[u][i] > 0 ) {
                    h[i] = h[u] + 1;
                    q[top++] = i; if ( top >= MN ) top-= MN;
                }
            }
        }
        return h[t] != -1; /*终点高度为0， 则已经没有增广路*/
    }
    int adjNet ( int adj, int *p ) { /*找到汇点， 更新残留网络*/
        for ( int i = t; i != s; i = p[i] ) {
            if ( g[p[i]][i] == 0 ) return 0; /*路径不可通*/
            adj = min ( adj, g[p[i]][i] );
        }
        for ( int i = t; i != s; i = p[i] ) {
            g[p[i]][i] -= adj;
            if ( g[p[i]][i] == 0 ) h[i] = MAXFLOW; /*当前边满载， 从层次图脱离*/
            g[i][p[i]] += adj;
        }   return adj;
    }
    int _dinic ( int u, int outFlow, int mi ) {
        if ( h[u] == MAXFLOW ) return 0; /*当前点满载*/
        if ( u == t ) { /*找到汇点， 更新残留网络*/
            return adjNet ( INF, path );
        }
        int res = 0;
        for ( int i = 1; i <= N; ++ i ) {/*搜索层次图，寻找可增广路径*/
            if ( h[u]+1 == h[i] ) { /*可行路径*/
                path[i] = u; /*路径记录*/
                res += dinic ( i, outFlow, min ( mi, g[u][i] ) );
            }
        }
        return res;
    }
    int dinic ( int u, int outFlow, int mi ) {  /*和上面的版本效率相差不大，基本一样*/
        if ( u == t ) { /*找到汇点， 回溯*/
            return mi;
        }
        int res = 0;
        for ( int i = 1; i <= N; ++ i ) {/*搜索层次图，寻找可增广路径*/
            if ( h[u]+1 == h[i] && res < mi && g[u][i]>0 ) { /*可行路径，且满足流量约束*/
                path[i] = u; /*路径记录*/
                int t = dinic ( i, outFlow, min ( mi-res, g[u][i] ) );
                if ( t != 0 ) {
                    res += t;
                    g[u][i] -= t, g[i][u] += t; /*更新反向流*/
                }
            }
        }
        return res;
    }
    int run ( int ss = 0, int tt = 0, int res = 0 ) {
        s = ss, t = tt;
        while ( getHei () ) {/*对剩余图重新计算高度值*/
            MEM(path,0); /*初始化路径*/
            res += dinic ( s, INF, INF ); /*在层次图中搜索可增广路径*/
        }
        return res;
    }
}dinic;
bool outPut ( int res = 0 ) {
    printf ( "Case %d: %d\n", ca ++, res ) ;
    return true;
}
Base *getAlgorithm () {/* 随机抽取算法， ^_^ */
    srand ( time (NULL) );
    int choice = ( rand() ) % 2+1;
    switch ( choice ) {
        case 0: return &mpla;/*这个无限TLE，只能作为一种思想来学习了 */
        case 1: return &ek;
        case 2: return &dinic;
    }
}
bool getInput () {
    MEM(graph,0);
    scanf ( "%d%d", &N, &M );
    for ( int i = 0; i < M; ++ i ) {
        scanf ( "%d%d%d", &x, &y, &v );
        graph[x][y] += v;
    }
    return true;
}
int main ()
{
    DBG
    scanf ( "%d", &T );
    while ( T -- ) {
        getInput ();
        base = getAlgorithm ();
        base->init ();
        outPut ( base->run ( 1, N ) );
    }
    return 0;
}
```

图论及图论算法 [编辑]

图 - 有向图 - 无向图 - 连通图 - 强连通图 - 完全图 - 稀疏图 - 零图 - 树 - 网络

基本遍历算法：宽度优先搜索 - 深度优先搜索 - A* - 并查集求连通分支 - Flood Fill

最短路：Dijkstra - Bellman-Ford（SPFA） - Floyd-Warshall - Johnson算法

最小生成树：Prim - Kruskal

强连通分支：Kosaraju - Gabow - Tarjan

网络流：增广路法（Ford-Fulkerson，Edmonds-Karp，Dinic） - 预流推进 - Relabel-to-front

图匹配 - 二分图匹配：匈牙利算法 - Kuhn-Munkres - Edmonds' Blossom-Contraction

1个分类：图论