个人资料

时间囊

Qing　微博

加好友　发纸条

写留言　加关注

博客等级：12
博客积分：375
博客访问：18,642
关注人气：26
获赠金笔：0
赠出金笔：0
荣誉徽章：

相关博文

floodfill算法
sanny

下午指数很可能再次大幅上攻
短线猎手博客

权重股低迷不振，创业板迭创新
旭初之道

张佳宁落入凡间的
摄影师韩心璐

“双重悖论”：经济快速增长与
中信出版社

淘气天尊：反抽撤退，后市还破
淘氣天尊

香港经济日报：大陆人懒得去了
陈思进

为什么得到具德上师的摄受非常

正文　字体大小：大 中 小

最大流模板【EdmondsKarp算法，简称EK算法，O(m^2n)】

(2011-05-26 23:28:50)

转 载 ▼

标签： it　分类： 图论

因为是初学教程，所以我会尽量避免繁杂的数学公式和证明。也尽量给出了较为完整的代码。本文的目标群体是网络流的初学者，尤其是看了各种NB的教程也没看懂怎么求最大流的小盆友们。本文的目的是，解释基本的网络流模型，最基础的最大流求法，即bfs找增广路法，也就是EK法，全名是Edmond-Karp，其实我倒是觉得记一下算法的全名和来历可以不时的拿出来装一装。

比如说这个，EK算法首先由俄罗斯科学家Dinic在1970年提出，没错，就是dinic算法的创始人，实际上他提出的也正是dinic算法，在EK的基础上加入了层次优化，这个我们以后再说，1972年Jack Edmonds和Richard Karp发表了没有层次优化的EK算法。但实际上他们是比1790年更早的时候就独立弄出来了。

你看，研究一下历史也是很有趣的。

扯远了，首先来看一下基本的网络流最大流模型。

有n个点，有m条有向边，有一个点很特殊，只出不进，叫做源点，通常规定为1号点。另一个点也很特殊，只进不出，叫做汇点，通常规定为n号点。每条有向边上有两个量，容量和流量，从i到j的容量通常用c[I,j]表示,流量则通常是f[I,j]。通常可以把这些边想象成道路，流量就是这条道路的车流量，容量就是道路可承受的最大的车流量。很显然的，流量<=容量。而对于每个不是源点和汇点的点来说，可以类比的想象成没有存储功能的货物的中转站，所有”进入”他们的流量和等于所有从他本身”出去”的流量。

把源点比作工厂的话，问题就是求从工厂最大可以发出多少货物，是不至于超过道路的容量限制，也就是，最大流。

比如这个图。每条边旁边的数字表示它的容量。

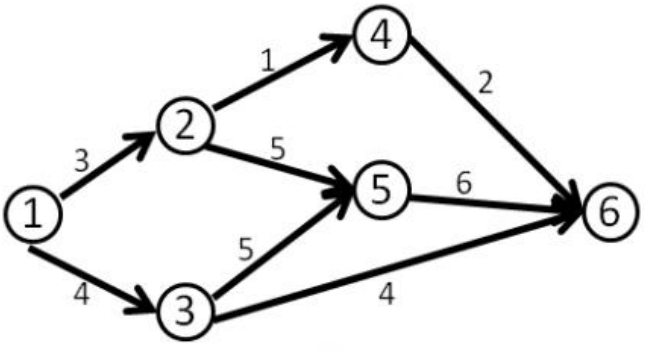

图 1

下面我们来考虑如何求最大流。

首先，假如所有边上的流量都没有超过容量(不大于容量)，那么就把这一组流量，或者说，这个流，称为一个可行流。一个最简单的例子就是，零流，即所有的流量都是0的流。

我们就从这个零流开始考虑，假如有这么一条路，这条路从源点开始一直一段一段的连到了汇点，并且，这条路上的每一段都满足流量<容量，注意，是严格的<,而不是<=。那么，我们一定能找到这条路上的每一段的(容量-流量)的值当中的最小值delta。我们把这条路上每一段的流量都加上这个delta，一定可以保证这个流依然是可行流，这是显然的。

这样我们就得到了一个更大的流，他的流量是之前的流量+delta，而这条路就叫做增广路。

我们不断地从起点开始寻找增广路，每次都对其进行增广，直到源点和汇点不连通，也就是找不到增广路为止。当找不到增广路的时候，当前的流量就是最大流，这个结论非常重要。

寻找增广路的时候我们可以简单的从源点开始做bfs，并不断修改这条路上的delta量，直到找到源点或者找不到增广路。

这里要先补充一点，在程序实现的时候，我们通常只是用一个c数组来记录容量，而不记录流量，当流量+1的时候，我们可以通过容量-1来实现，以方便程序的实现。

Bfs过程的半伪代码：下面另给一个C++版的模板

```
int BFS()
{
    int i,j,k,v,u;
    memset(pre,-1,sizeof(pre));
    for(i=1;i<=n;++i)flow[i]=max_int;
    queue<int>que;
    pre[start]=0;
    que.push(start);
    while(!que.empty())
    {
        v=que.front();
        que.pop();
        for(i=1;i<=n;++i)
        {
            u=i;
            if(u==start||pre[u]!=-1||map[v][u]==0)continue;
            pre[u]=v;
            flow[u]=MIN(flow[v],map[v][u]);
            que.push(u);
        }
    }
    if(flow[end]==max_int)return -1;
    return flow[end];
}
```

但事实上并没有这么简单，上面所说的增广路还不完整，比如说下面这个网络流模型。

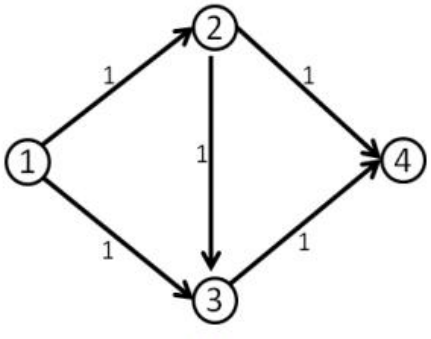

图 2

推荐博文

“深圳机场撞人事件”不是一个人

2550mAh电

日本雾霾之战：民众与政府的博弈

探秘保加利亚“新娘集市”：少女

别光顾着把枪口对准柴静

震撼！俄西伯利亚又现4个神秘大

美国总统专机升级 奢

中国制造的甲午战败

南非猎豹冒死捕杀豪猪被扎满嘴刺

美国产妇流行吃自己的胎盘(图)

谁看过这篇博文

粪里有毒	0分钟前
Endlessde…	3月3日
浮虚_	2月28日
char256	2月26日
yxr童鞋	2月9日
CJoier_金坤	2月1日
a897383520	1月27日
爱it的小…	1月12日
弥可可	1月8日
超级臭宝妈	1月4日
yinshuire…	1月2日
南工ACM校…	12月24日

我们第一次找到了1-2-3-4这条增广路，这条路上的delta值显然是1。于是我们修改后得到了下面这个流。（图中的数字是容量）

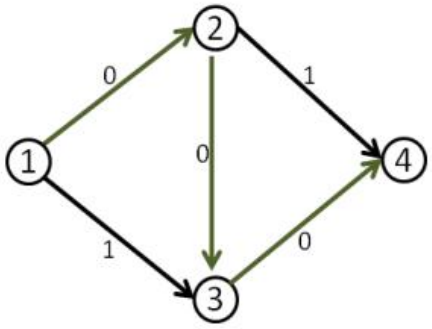

图 3

这时候(1,2)和(3,4)边上的流量都等于容量了，我们再也找不到其他的增广路了，当前的流量是1。

但这个答案明显不是最大流，因为我们可以同时走1-2-4和1-3-4，这样可以得到流量为2的流。

那么我们刚刚的算法问题在哪里呢？问题就在于我们没有给程序一个”后悔”的机会，应该有一个不走(2-3-4)而改走(2-4)的机制。那么如何解决这个问题呢？回溯搜索吗？那么我们的效率就上升到指数级了。

而这个算法神奇的利用了一个叫做反向边的概念来解决这个问题。即每条边(I,j)都有一条反向边(j,i)，反向边也同样有它的容量。

我们直接来看它是如何解决的：

在第一次找到增广路之后，在把路上每一段的容量减少delta的同时，也把每一段上的反方向的容量增加delta。即在Dec(c[x,y],delta)的同时，inc(c[y,x],delta)

我们来看刚才的例子，在找到1-2-3-4这条增广路之后，把容量修改成如下

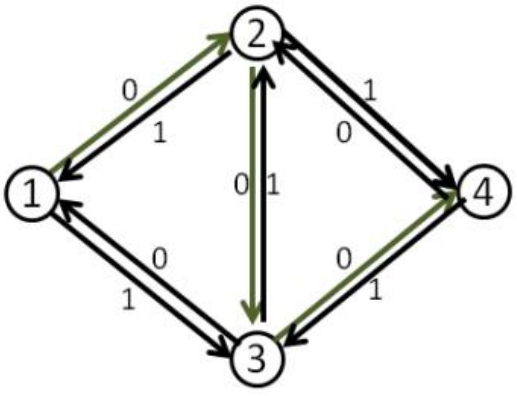

图 4

这时再找增广路的时候，就会找到1-3-2-4这条可增广量，即delta值为1的可增广路。将这条路增广之后，得到了最大流2。

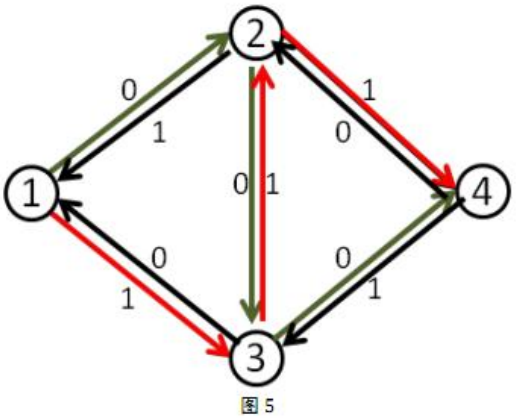

图 5

那么，这么做为什么会是对的呢？我来通俗的解释一下吧。

事实上，当我们第二次的增广路走3-2这条反向边的时候，就相当于把2-3这条正向边已经是用了的流量给”退”了回去，不走2-3这条路，而改走从2点出发的其他的路也就是2-4。（有人问如果这里没有2-4怎么办，这时假如没有2-4这条路的话，最终这条增广路也不会存在，因为他根本不能走到汇点）同时本来在3-4上的流量由1-3-4这条路来”接管”。而最终2-3这条路正向流量1，反向流量1，等于没有流量。

这就是这个算法的精华部分，利用反向边，使程序有了一个后悔和改正的机会。而这个算法和我刚才给出的代码相比只多了一句话而已。

```
#include<iostream>
#include<queue>
using namespace std;
const int maxn=205;
const int inf=0x7fffffff;
int r[maxn][maxn]; //残留网络，初始化为原图
bool visit[maxn];
int pre[maxn];
int m,n;
bool bfs(int s,int t)  //寻找一条从s到t的增广路，若找到返回true
{
    int p;
    queue<int > q;
    memset(pre,-1,sizeof(pre));
    memset(visit,false,sizeof(visit));
    pre[s]=s;
    visit[s]=true;
    q.push(s);
    while(!q.empty())
    {
        p=q.front();
        q.pop();
        for(int i=1;i<=n;i++)
        {
            if(r[p][i]>0&&!visit[i])
            {
                pre[i]=p;
                visit[i]=true;
                if(i==t) return true;
                q.push(i);
            }
        }
    }
    return false;
}
int EdmondsKarp(int s,int t)
{
   int flow=0,d,i;
   while(bfs(s,t))
   {
```

```
        d=inf;
        for(i=t;i!=s;i=pre[i])
            d=d<r[pre[i]][i]? d:r[pre[i]][i];
        for(i=t;i!=s;i=pre[i])
        {
            r[pre[i]][i]-=d;
            r[i][pre[i]]+=d;
        }
        flow+=d;
    }
    return flow;
}

int main()
{
    while(scanf("%d%d",&m,&n)!=EOF)
    {
        int u,v,w;
        memset(r,0,sizeof(r));///
        for(int i=0;i<m;i++)
        {
            scanf("%d%d%d",&u,&v,&w);
            r[u][v]+=w;
        }
        printf("%d\n",EdmondsKarp(1,n));
    }
    return 0;
}
```


前一篇：操作系统实验四、进程同步实验

后一篇：最大流模板【Ford-Fulkerson方法，简称FF方法】

2014-7-1 15:33 回复(0)

发评论

粪里有毒：您还未开通博客，点击一秒开通。

分享到微博 评论并转载此博文 匿名评论

验证码： 请点击后输入验证码 收听验证码

发评论

〈 前一篇
操作系统实验四、进程同步实验

后一篇 〉
最大流模板【Ford-Fulkerson方法，简称FF方法】