**http://www.cnblogs.com/liu-runda/p/6262832.html**

有上下界的网络流的核心是”**调整**”,我们通过一个初始的未必可行的流调整出一个可行流,还可以从可行的未必最大/最小的流调整出最大/最小流.

另一个常用技巧是**有源汇的流和无源汇的流(循环流)的转换**.除了无源汇可行流的求解,其他有源汇的上下界网络流都要用到这个技巧.

1. **无源汇有上下界可行流**(也就是循环流)

模型:一个网络,求出一个流,使得每条边的流量必须>=Li且<=Hi,每个点必须满足总流入量=总流出量(流量守恒)(这个流的特点是循环往复,无始无终).

这个算法是有上下界网络流算法的基础,只要深刻理解这个算法其他算法也就水到渠成,因此我用大篇幅力图将这个算法的思想和细节阐述清楚.

可行流算法的核心是将一个不满足流量守恒的初始流调整成满足流量守恒的流.

流量守恒,即每个点的总流入量=总流出量

如果存在一个可行流,那么一定满足每条边的流量都大于等于流量的下限.因此我们可以令每条边的流量等于流量下限,得到一个初始流,然后建出这个流的残量网络.(即:每条边的流量等于这条边的流量上限与流量下限之差)**这个初始流不一定满足流量守恒,因此最终的可行流一定是在这个初始流的基础上增大了一些边的流量使得所有点满足流量守恒.**

因此我们考虑在残量网络上求出一个另不满足流量守恒的附加流,使得这个附加流和我们的初始流合并之后满足流量守恒.即:

如果某个点在所有边流量等于下界的初始流中满足流量守恒,那么这个点在附加流中也满足流量守恒,

如果某个点在初始流中的流入量比流出量多x,那么这个点在附加流中的流出量比流入量多x.

如果某个点在初始流中的流入量比流出量少x,那么这个点在附加流中的流出量比流入量少x.

         可以认为附加流中一条从u到v的边上的一个流量代表将原图中u到v的流量增大1

X的数值可以枚举x的所有连边求出.比较方便的写法是开一个数组A[],A[i]表示i在初始流中的流入量-流出量的值,那么A[i]的正负表示流入量和流出量的大小关系,下面就用A[i]表示初始流中i的流入量-流出量

但是dinic算法能够求的是满足流量守恒的有源汇最大流,不能在原网络上直接求一个这样的无源汇且不满足流量守恒的附加流.注意到附加流是在原网络上不满足流量守恒的,这启发我们添加一些原网络之外的边和点,用这些边和点实现”原网络上流量不守恒”的限制.

具体地,如果一个点i在原网络上的附加流中需要满足流入量>流出量(初始流中流入量<流出量,A[i]<0),那么我们需要给多的流入量找一个去处,因此我们**建一条从i出发**流量=-A[i]的边.如果A[i]>0,也就是我们需要让附加流中的流出量>流入量,我们需要让多的流出量有一个来路,因此我们**建一条指向i的**流量=A[i]的边.

当然,我们所新建的从i出发的边也要有个去处,指向i的边也要有个来路,因此我们新建一个虚拟源点ss和一个虚拟汇点tt(双写字母是为了和有源汇网络流中的源点s汇点t相区分).新建的指向i的边都从ss出发,从i出发的边都指向tt.一个点要么有一条边指向tt,要么有一条边来自ss,

指向tt的边的总流量上限一定等于ss流出的边的总流量上限,因为每一条边对两个点的A[i]贡献一正一负大小相等,所以全部点的A[i]之和等于0,即小于0的A[i]之和的绝对值=大于0的A[i]之和的绝对值.

如果我们能找到一个流满足新加的边都满流,这个流在原图上的部分就是我们需要的附加流(根据我们的建图方式,“新加的边都满流”和”附加流合并上初始流得到流量平衡的流”是等价的约束条件).

那么怎样找出一个新加的边都满流的流呢?可以发现假如存在这样的方案,这样的流一定是我们所建出的图的ss-tt最大流,所以跑ss到tt的最大流即可.如果最大流的大小等于ss出发的所有边的流量上限之和(此时指向tt的边也一定满流,因为这两部分边的流量上限之和相等).

最后,每条边在可行流中的流量=容量下界+附加流中它的流量(即跑完dinic之后所加反向边的权值).

代码(ZOJ2314 Reactor Cooling)

[复制代码](javascript:void(0);)

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn=300,maxm=100000;

struct edge{

int to,next,w,num;

}lst[maxm];int len=0,first[maxn],\_first[maxn];

void addedge(int a,int b,int w,int num){

lst[len].num=num;

lst[len].to=b;lst[len].next=first[a];lst[len].w=w;first[a]=len++;

lst[len].num=num;

lst[len].to=a;lst[len].next=first[b];lst[len].w=0;first[b]=len++;

}

int vis[maxn],dis[maxn],q[maxn],head,tail,s,t,T;

bool bfs(){

vis[s]=++T;dis[s]=1;head=tail=0;q[tail++]=s;

while(head!=tail){

int x=q[head++];

for(int pt=first[x];pt!=-1;pt=lst[pt].next){

if(lst[pt].w&&vis[lst[pt].to]!=T){

vis[lst[pt].to]=T;dis[lst[pt].to]=dis[x]+1;q[tail++]=lst[pt].to;

}

}

}

if(vis[t]==T)memcpy(\_first,first,sizeof(first));

return vis[t]==T;

}

int dfs(int x,int lim){

if(x==t){

return lim;

}

int flow=0,a;

for(int pt=\_first[x];pt!=-1;pt=lst[pt].next){

\_first[x]=pt;

if(lst[pt].w&&dis[lst[pt].to]==dis[x]+1&&(a=dfs(lst[pt].to,min(lst[pt].w,lim-flow)))){

lst[pt].w-=a;lst[pt^1].w+=a;flow+=a;

if(flow==lim)return flow;

}

}

return flow;

}

int dinic(){

int ans=0,x;

while(bfs())

while(x=dfs(s,0x7f7f7f7f))ans+=x;

return ans;

}

int low[maxm],ans[maxm];

int totflow[maxn],n,m;

void work(){

memset(totflow,0,sizeof(totflow));

memset(first,-1,sizeof(first));len=0;

scanf("%d%d",&n,&m);

int u,v,b;

s=0;t=n+1;

for(int i=1;i<=m;++i){

scanf("%d%d%d%d",&u,&v,&low[i],&b);

addedge(u,v,b-low[i],i);totflow[u]-=low[i];totflow[v]+=low[i];

}

int sum=0;

for(int i=1;i<=n;++i){

if(totflow[i]<0){

addedge(i,t,-totflow[i],0);

}else{

sum+=totflow[i];

addedge(s,i,totflow[i],0);

}

}

if(dinic()==sum){

puts("YES");

for(int i=1;i<=n;++i){

for(int pt=first[i];pt!=-1;pt=lst[pt].next){

if(lst[pt].num==0||pt%2==0)continue;

ans[lst[pt].num]=lst[pt].w+low[lst[pt].num];

}

}

for(int i=1;i<=m;++i)printf("%d\n",ans[i]);

}else puts("NO");

}

int main(){

int tests;scanf("%d",&tests);

while(tests--){

work();if(tests)printf("\n");

}

return 0;

}

[复制代码](javascript:void(0);)

　　2. 有源汇有上下界可行流

模型:现在的网络有一个**源点s和汇点t**,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.

源点和汇点不满足流量守恒,这让我们很难办,因此我们想办法把问题转化成容易处理的每个点都满足流量守恒的无源汇情况.

为了使源汇点满足流量守恒,我们需要有边流入源点s,有边流出汇点t.注意到源点s的流出量等于汇点t的流入量,我们就可以从汇点t向源点s连一条下界为0上界为无穷大的边,相当于把从源点s流出的流量再流回来.在这样的图中套用上面的算法求出一个可行的循环流,拆掉从汇点t到源点s的边就得到一个可行的有源汇流.

这里有一个小问题:最后得到的可行的有源汇流的流量是多少?

可以发现,循环流中一定满足s流出的总流量=流入s的总流量,假定原图中没有边流入s,那么s流出的流量就是t到s的无穷边的流量,也就是s-t可行流的流量.因此我们最后看一下t到s的无穷边的流量(即dinic跑完之后反向边的权值)即可知道原图中有源汇可行流的流量.

代码:这个可行流算法在有源汇有上下界最大流/最小流中都会用到,可以看下面两个算法的代码

3.有源汇有上下界最大流

模型:现在的网络有一个源点s和汇点t,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.在这些前提下要求**总流量最大**.

首先套用上面的算法求出一个有源汇有上下界可行流.此时的流不一定最大.

接下来在残量网络上跑s-t最大流即可.

最终的最大流流量=可行流流量(即t到s的无穷边上跑出的流量)+新增广出的s-t流量

问题:会不会增广的时候使得一些边不满足流量下限?

不会.因为我们一开始建的图就是把大小等于流量下限的流量拿出去之后的残量网络,这些流量根本没有在图中出现.

代码:ZOJ 3229 Shoot The Bullet~~东方文花帖~~ (由于ZOJ的评测插件似乎挂了,并不知道对不对,请谨慎取用)

[复制代码](javascript:void(0);)

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn=2005,maxm=100005;

const int inf=0x7f7f7f7f;

struct edge{

int to,next,w,num;

}lst[maxm];int len=0,first[maxn],\_first[maxn];

void addedge(int a,int b,int w,int num){

lst[len].num=num;

lst[len].to=b;lst[len].next=first[a];lst[len].w=w;first[a]=len++;

lst[len].num=num;

lst[len].to=a;lst[len].next=first[b];lst[len].w=0;first[b]=len++;

}

int q[maxn],vis[maxn],dis[maxn],T,s,t,head,tail,ss,tt;

bool bfs(){

head=tail=0;vis[s]=++T;q[tail++]=s;

while(head!=tail){

int x=q[head++];

for(int pt=first[x];pt!=-1;pt=lst[pt].next){

if(lst[pt].w&&vis[lst[pt].to]!=T){

vis[lst[pt].to]=T;dis[lst[pt].to]=dis[x]+1;q[tail++]=lst[pt].to;

}

}

}

if(vis[t]==T)memcpy(\_first,first,sizeof(first));

return vis[t]==T;

}

int dfs(int x,int lim){

if(x==t)return lim;

int flow=0,a;

for(int pt=\_first[x];pt!=-1;pt=lst[pt].next){

\_first[x]=pt;

if(lst[pt].w&&dis[lst[pt].to]==dis[x]+1&&(a=dfs(lst[pt].to,min(lst[pt].w,lim-flow)))){

lst[pt].w-=a;lst[pt^1].w+=a;flow+=a;

if(flow==lim)return flow;

}

}

return flow;

}

int dinic(){

int ans=0,x;

while(bfs())

while(x=dfs(s,inf))ans+=x;

return ans;

}

int totflow[maxn];

void Add(int a,int b,int lo,int hi,int num){

totflow[a]-=lo;totflow[b]+=lo;

addedge(a,b,hi-lo,num);

}

int low[maxm],ans[maxm];

int n,m,tot;

void bound\_flow(){

int sum=0;

for(int i=s;i<=t;++i){

if(totflow[i]<0){

addedge(i,tt,-totflow[i],0);

}else{

sum+=totflow[i];

addedge(ss,i,totflow[i],0);

}

}

addedge(t,s,0x7f7f7f7f,0);

int tmps=s,tmpt=t;

s=ss;t=tt;

if(dinic()==sum){

for(int pt=first[ss];pt!=-1;pt=lst[pt].next){

lst[pt].w=lst[pt^1].w=0;

}

for(int pt=first[tt];pt!=-1;pt=lst[pt].next){

lst[pt].w=lst[pt^1].w=0;

}

int flow0=lst[len-1].w;

lst[len-1].w=lst[len-2].w=0;

s=tmps;t=tmpt;

printf("%d\n",flow0+dinic());

for(int i=1;i<=m;++i){

for(int pt=first[i+n];pt!=-1;pt=lst[pt].next){

if(lst[pt].num!=0){

ans[lst[pt].num]=lst[pt].w+low[lst[pt].num];

}

}

}

for(int i=1;i<=tot;++i)printf("%d\n",ans[i]);

}else{

printf("-1\n");

}

}

void work(){

s=0;t=n+m+1;

ss=n+m+2;tt=n+m+3;

memset(first,-1,sizeof(first));len=0;

memset(totflow,0,sizeof(totflow));

int x,y;

for(int i=1;i<=m;++i){

scanf("%d",&x);

Add(n+i,t,x,inf,0);

}

int l,h;

tot=0;

for(int i=1;i<=n;++i){

scanf("%d%d",&x,&y);

Add(s,i,0,y,0);

for(int j=1;j<=x;++j){

++tot;

scanf("%d%d%d",&y,&l,&h);

Add(i,n+y+1,l,h,tot);low[tot]=l;

}

}

bound\_flow();printf("\n");

}

int main(){

while(scanf("%d%d",&n,&m)!=EOF)work();

return 0;

}

[复制代码](javascript:void(0);)

4.有源汇有上下界最小流

模型:现在的网络有一个源点s和汇点t,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.在这些前提下要求**总流量最小**.

依然是先跑出一个有源汇可行流.这时候的流也不一定是最小的.假如我们能在残量网络上找到一条s-t的路径使得去掉这条路径上的流量之后仍然满足流量下限,我们就可以得到一个更小的流.好像我们并没有什么算法可以”找到尽可能多的能够去除流量的路径”

这时候需要我们再理解一下dinic的反向边.反向边的流量增加等价于正向边的的流量减少.因此我们在残量网络上找出t到s的流就相当于减小了s到t的流,因此我们在跑出可行流的残量网络上跑t-s最大流,用可行流的大小减去这一次t-s最大流的大小就是最小流的大小.(t-s最大流其实是尽量缩减s-t方向的流).

问题:会不会使流量缩减到不满足流量下限?

不会.和有源汇有上下限的最大流一样,我们之前从每条边上拿出了大小等于流量下限的流量构成初始流,这些流量不在我们建出的图中.最极端的情况是缩减到所有边的流量等于流量下限,不会更小了.

代码:bzoj2502 清理雪道

[复制代码](javascript:void(0);)

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn=205,maxm=100005;

struct edge{

int to,next,w;

}lst[maxm];int len=0,first[maxn],\_first[maxn];

void addedge(int a,int b,int w){//printf("Add %d %d\n",a,b);

lst[len].to=b;lst[len].next=first[a];lst[len].w=w;first[a]=len++;

lst[len].to=a;lst[len].next=first[b];lst[len].w=0;first[b]=len++;

}

int q[maxn],vis[maxn],dis[maxn],head,tail,s,t,T,ss,tt;

bool bfs(){

head=tail=0;vis[s]=++T;dis[s]=1;q[tail++]=s;

while(head!=tail){

int x=q[head++];

for(int pt=first[x];pt!=-1;pt=lst[pt].next){

if(lst[pt].w&&vis[lst[pt].to]!=T){

vis[lst[pt].to]=T;dis[lst[pt].to]=dis[x]+1;q[tail++]=lst[pt].to;

}

}

}

if(vis[t]==T)memcpy(\_first,first,sizeof(first));

return vis[t]==T;

}

int dfs(int x,int lim){

if(x==t)return lim;

int flow=0,a;

for(int pt=\_first[x];pt!=-1;pt=lst[pt].next){

\_first[x]=pt;

if(lst[pt].w&&dis[lst[pt].to]==dis[x]+1&&(a=dfs(lst[pt].to,min(lst[pt].w,lim-flow)))){

lst[pt].w-=a;lst[pt^1].w+=a;flow+=a;

if(flow==lim)return flow;

}

}

return flow;

}

int dinic(){

int ans=0,x;

while(bfs()){

while(x=dfs(s,0x7f7f7f7f))ans+=x;

}

return ans;

}

int totflow[maxn];

void del(int x){

for(int pt=first[x];pt!=-1;pt=lst[pt].next)lst[pt].w=lst[pt^1].w=0;

}

int main(){

int n;scanf("%d",&n);

int x,y;

memset(first,-1,sizeof(first));

for(int i=1;i<=n;++i){

scanf("%d",&x);

for(int j=1;j<=x;++j){

scanf("%d",&y);

totflow[i]--;totflow[y]++;

addedge(i,y,0x7f7f7f7f);

}

}

s=0;t=n+1;ss=n+2,tt=n+3;

for(int i=1;i<=n;++i){

addedge(s,i,0x7f7f7f7f);

addedge(i,t,0x7f7f7f7f);

}

for(int i=1;i<=n;++i){

if(totflow[i]<0){

addedge(i,tt,-totflow[i]);

}else{

addedge(ss,i,totflow[i]);

}

}

addedge(t,s,0x7f7f7f7f);

int tmps=s,tmpt=t;

s=ss;t=tt;

dinic();

int flow0=lst[len-1].w;

lst[len-1].w=lst[len-2].w=0;

del(ss);del(tt);

s=tmpt;t=tmps;

printf("%d\n",flow0-dinic());

return 0;

}

[复制代码](javascript:void(0);)