模板：

1. Isap
2. 费用流（注释部分有动态加边的解释）
3. 无源汇有上下界可行流
4. 有源汇有上下界可行流
5. 有源汇有上下界最大流
6. 有源汇有下界最小费用可行流

isap

struct edge

{

int from,to,cap,flow,next;

};

int inf=1e7;

struct isap

{

int n,m,s,t;

int np,first[maxn],num[maxn],d[maxn],cur[maxn],fa[maxn];

bool vis[maxn];

edge e[maxm<<1];

int other(int p)

{

return p&1?p+1:p-1;

}

void addedge(int u,int v,int w)

{

e[++np]=(edge){u,v,w,0,first[u]};

first[u]=np;

e[++np]=(edge){v,u,0,0,first[v]};

first[v]=np;

}

void bfs()

{

memset(vis,0,sizeof(vis));

//memset(d,-1,sizeof(d));

memset(num,0,sizeof(num));

for(int i=1;i<=n;i++)d[i]=n;

queue<int>q;

q.push(t);

d[t]=0;

num[0]=1;

vis[t]=1;

while(!q.empty())

{

int k=q.front();q.pop();

for(int p=first[k];p!=0;p=e[p].next)

{

int j=e[p].to;

int p2=other(p);

if(!vis[j]&&e[p2].cap>e[p2].flow)

{

vis[j]=1;

d[j]=d[k]+1;

num[d[j]]++;

q.push(j);

}

}

}

}

int augment()

{

int x=t,a=inf;

while(x!=s)

{

a=min(a,e[fa[x]].cap-e[fa[x]].flow);

x=e[fa[x]].from;

}

x=t;

while(x!=s)

{

e[fa[x]].flow+=a;

e[other(fa[x])].flow-=a;

x=e[fa[x]].from;

}

return a;

}

int maxflow(int s,int t)

{

this->s=s;

this->t=t;

int flow=0;

bfs();

//memset(num,0,sizeof(num));

//for(int i=1;i<=n;i++)num[d[i]]++;

int x=s;

memcpy(cur,first,sizeof(cur[0])\*(n+1));

while(d[s]<n)

{

if(x==t)

{

flow+=augment();

x=s;

}

int ok=0;

for(int p=cur[x];p!=0;p=e[p].next)

{

int j=e[p].to;

if(e[p].cap>e[p].flow&&d[x]==d[j]+1)

{

ok=1;

cur[x]=p;

fa[j]=p;

x=j;

break;

}

}

if(!ok)

{

int md=n-1;

for(int p=first[x];p!=0;p=e[p].next)

{

int j=e[p].to;

if(e[p].cap>e[p].flow)md=min(md,d[j]);

}

num[d[x]]--;

if(num[d[x]]==0)return flow;

d[x]=md+1;

num[d[x]]++;

cur[x]=first[x];

if(x!=s)x=e[fa[x]].from;

}

}

return flow;

}

}e;

int main()

{

int ai,bi,ci;

scanf("%d%d",&e.m,&e.n); //e.m边数，e.n点数

for(int i=1;i<=e.m;i++)

{

scanf("%d%d%d",&ai,&bi,&ci);

e.addedge(ai,bi,ci);

}

printf("%d\n",e.maxflow(1,e.n));

return 0;

}

费用流

struct edge

{

int from,to,next,cap,flow,cost;

};

struct myek

{

edge e[maxm<<1];

int n,m,s,t,np;

int first[maxn],inq[maxn],d[maxn],a[maxn],fa[maxn];

void addedge(int u,int v,int cap,int cost)

{

e[++np]=(edge){u,v,first[u],cap,0,cost};

first[u]=np;

e[++np]=(edge){v,u,first[v],0,0,-cost};

first[v]=np;

}

int other(int x)

{

return x&1?x+1:x-1;

}

int spfa()

{

for(int i=1;i<=n;i++)d[i]=inf;

for(int i=1;i<=n;i++)a[i]=0;

queue<int>q;

q.push(s);

d[s]=0;

inq[s]=1;

a[s]=finf;

while(!q.empty())

{

int k=q.front();q.pop();

inq[k]=0;

for(int p=first[k];p!=0;p=e[p].next)

{

int j=e[p].to;

if(e[p].cap>e[p].flow&&d[j]>d[k]+e[p].cost)

{

d[j]=d[k]+e[p].cost;

fa[j]=p;

a[j]=min(a[k],e[p].cap-e[p].flow);

if(!inq[j])

{

inq[j]=1;

q.push(j);

}

}

}

}

return a[t];

}

void update(int p)//若不会超时，则不用动态加边，通常动态加边加的边都有一定规律。

int maxflow(int s,int t)

{

this->s=s;

this->t=t;

int flow=0,cost=0;

int delt=0;

while(1)

{

delt=spfa();

if(delt==0)break;

flow+=delt;

cost+=delt\*d[t];

int x=t;

while(x!=s)

{

//动态加边的位置，即手动考虑某些边一定是在一些边满流后才会走，那么满流后再加入

//update(fa[x]);

e[fa[x]].flow+=delt;

e[other(fa[x])].flow-=delt;

x=e[fa[x]].from;

}

}

return cost;

}

}e;

有上下界的网络流：

模板依然用isap

无源汇可行流：

int n,m,s,t;

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

s=n+1;t=n+2;

//有无源汇的可行流其实就是先默认下界，对于不守恒的点

//这里有一个值得思考的点，若流入>流出，到底是s向其连边还是其向t连边呢？

//应为s向其连边，因为我们是要让它多流出来平衡，而网络流算法是流量守恒的

//那么显然我们应流入一些虚的流量让其流出已达到平衡，若流出>流入显然同理

for(int ii=1;ii<=m;ii++)

{

scanf("%d%d%d%d",&ai,&bi,&ci,&di);

e.addedge(ai,bi,di-ci);

a[ai]-=ci; a[bi]+=ci;

}

int sum=0;

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

{

if(a[i]>0)

e.addedge(s,i,a[i]),sum+=a[i];

else

e.addedge(i,t,-a[i]);

}

e.n=t;

if(e.maxflow(s,t)==sum)printf("Yes\n");

else printf("No\n");

有源汇可行流：

//有源汇其实就是让到t的流量流回s来变成无源汇，即t向s连一条边即可。

//但其实这个模型是有问题的，假想该图不连通，但每个不连通图中都可以跑无源汇

//那么这种方法就会判出来成立，但想必没有这种特殊的情况，因为毕竟网络流是为

//实际情况服务的，而且这种情况符合网络流的定义。。。。。。

e.addedge(n,1,2e9);

有源汇有上下界最大流：

e.addedge(n,1,inf);

//有源汇的最大流就是先跑可行流，然后删t到s的路，此时是流量守恒的，

//此时就可以跑s到t的最大流了，因为虚ss和虚tt没有办法退流，

//故此时直接跑最大流再加上之前的就可以了。

if(e.maxflow(s,t)==sum)

{

int id=e.np-1;

long long flow=e.e[id].flow;

e.e[id].flow=e.e[id].cap=e.e[e.other(id)].flow=e.e[e.other(id)].cap=0;

flow+=e.maxflow(1,n);

cout<<flow<<'\n';

}

else printf("0\n");

有源汇有下界最小费用可行流

费用流模板同前

连法同有上下界的可行流直接连即可，然后跑即可

网络流：

四个定理：1.最小割==最大流

2.DAG上最小路径覆盖==原来的点数 - 新建二分图的最大匹配

3.二分图上的最大独立集==点数-最大匹配数

4.二分图上的最小覆盖集==最大匹配

最大权闭合子图：

S连正权点，负权点连t，当选x就必须选y时x就向y连一条边，最后正权点的和-最大流即可。

上下界：即让其流量平衡即可。有源汇：t向s连一条inf的边。

最大流：删除t向s连的边，然后s跑最大流即可。

最小流：删除s向t连的边，然后t向s跑最大流即可。

上下界最小费可行流：同理，不过直接跑最小费即可。

存在最小割的并：残量网络跑tarjan，若belong不同即是。

最小割的交：且一个属于s，一个属于t。

二分图匹配输出字典序最小的解：暴搜，枚举（目前有两种：从小到大和从大到小）都可以，都是直接暴力枚举然后看能否有合法的匹配。

快速退流：对于一条原图上的边，将它的容量和流量改为0，然后u向s跑最大流，然后v向t跑最大流即可。

线段树优化建图，可持久化优化建图：

即正常建图，把一些公共的边开成线段树上的这样可以减少连边，切记持久化后旧版本要向新版本连边