网络流

网络是指一个有向图 $G=(V,E)$，有两个特殊节点：源点 S 和汇点 T。每条有向边 $(x,y)\in E$ 都有一个权值 $c(x,y)$，称作边的容量。若 $(x,y)\notin E$，则 $c(x,y)=0$

用 $f(x,y) (f=flow)$ 表示边 (x,y) 上的流量， $c(x,y)-f(x,y)$ 成为边的剩余容量。通常用 $f(x,y)/c(x,y)$ 的形式标记边上的流量与容量。

可行流满足的条件：

1. 容量限制：$f(x,y)\le c(x,y)$
2. 流量守恒 ：该定义由此公式给出： $$\sum_{(u,x)\in E}f(u,x)=\sum_{(x,v)\in E}f(x,v),x\ne S,y\ne T$$ 这里 S,T 分别为起点（源点）和终点（汇点）。 其中 $\sum_{(S,v)\in E}f(S,v)$ 称为整个网络的流量。

一些比较重要的概念

最大流：从源点流向汇点的最大流量。

增广路：一条从源点到汇点所有边剩余容量大于 0 的路径。

残量网*络*：由网络中所有节点和剩余容量大于 0 的边构成的子图这里的边包括有向边和反向边。

最大流

定*义*：令 $G=(V,E)$ 是一个有源汇点的网络，我们希望在 G 上指定合适流 f，以最大化整个网络流量 $|f|$，这一问题称为最大流问题(Maximum Flow Problem)。

如何计算这个值？

我们可以使用一类叫做*Ford-Fulkerson*增广的方法去计算。该方法运用贪心思想寻找增广路并更新求解最大流。

我们思考一个过程，首先每次找一条可行流（即一条从源点到汇点的流量大于0的路）。那么思考贪心，每次选择完都可以得到一个较优解。那如果此方法不是最优的怎么办？我们不妨参考反悔贪心的思路，给流量一个反悔的选择（世上真有反悔药）。具体方法就是在原边 (u,v) 上新建一个反向边 (v,u)。初始设置此边流量为 0 。那么每次找到一条增广路时，我们将正向边减去的流量放到反向边中去。由流量守恒可知，该方法应该是正确的。

EK算法

我们可以选择一个比较自然的方案在图上进行BFS搜索。

具体过程如下：

1.如果在 G_f 上可以从 S 到 T，则找到一条增广路。 2.对于增广路 p，我们计算出 p 经过边的剩余容量的最小值 $\Delta=min_{(u,v)\in p}c_f(u,v)$。我们给每条边都去掉 Δ，对反向边加上 Δ。 3.因为我们修改了流量，所以我们得到新的 G_f。

具体方法参考下发文件的 `EK_algorithm.cpp`

Dinic算法（Dinner算法bushi）

EK算法是每条边进行一次修改，但是效率不一定优秀，那么我们尝试优化一下这个算法。

先前EK实际上是对网络做了分层，但是分完层后只修改一条边，这样效率并没有发挥极致。那么我们可以考虑到 T，每次找到一条到 T 的路线后，显然我们最大的增广路就是将改变阻塞后的路，那么，我们可以尝试找到一条该路径后将整个网络阻塞。于是达到第二步优化，分层后用DFS，处理分层网络，一次性阻塞所有可行边即可。

这里需要加挺多优化才能达到正确的复杂度，所以参见代码 `Dinic_algorithm`

然后思考一下数组 `now[]` 的作用。

最小费用最大流

最小费用最大流是在达到最大流的前提下最小化费用。那么就有点像最短路算法了。那么思考EK算法的过程。将此处与SPFA算法对比，发现两者很相似，本质上都是一个广搜过程。于是我们可以稍加改进EK将其用来求最小费用最大流。

然后，你在网络流领域的基础已经学完了，可以做绝大部分的网络流题目了。

但是，正如图所示：

网络流最难的不是算法而是建图，这就是为何网络流问题评级这么高的问题。

这就要看大家的刷题经验了，那么快去刷题罢！