身为省选，乃至NOI的重要算法。

网上讲解的人，真是的。还是自己履履思路好了。

首先是网络流

对于网络流来说，有一个必要的算法就是Ford-Fulkerson方法，之所以为方法，具体原因请见算导。

一开始用Ford-Fulkerson算法的时候，还是用邻接矩阵，但是，现在改了，改用邻接表了。

具体见模板程序以及注释

这里也说几句：

首先注意残留网络的修改，找一条路径上的最小流量处（即瓶颈）是本题的关键，因此，静态链表是要保存父亲节点的。修改残留网络（增广路径）的时候，注意正反都要改。

静态链表存正反的时候，需要注意：请使用0,2,4,6,8,……去存正向边；1,3,5,7,9……去存负向边，0->1,2->3,……

修改的时候，只要异或一下1就好了（e[i]-->e[i^1]）

一并说了吧，有关于最小费用最大流。（最小割什么的只是定理罢了，不要去管它怎么证的），有关最小费用最大流，顾名思义这里肯定是要用到SPFA算法的

SPFA是求最短路径的最好算法，以后不管什么求最短路的题，都直接SPFA就好了，不要想太多，有关SPFA的静态列表实现，也已经在标程模板中了。具体的，参考一下吧。

所以可以在求最大流的同时去求最小费用，也是可以的。这样直接将两个事，变为一件事，何不乐而为之呢？

最小费用的话，在编的时候，没有任何除最大流算法以外的问题，除了一个预处理。

-cost的存在，预处理的时候正反相向的两条边的cost互为相反数，这样就可避免中间算最小费用的时候出现复杂的处理了（省脑子）

下面是有关二分图匹配的问题

所谓二分图，有一道题可以作为经典例题，就是很久之后才A的花店橱窗。

二分图，顾名思义，就是有两个图（这样说可能有点令人费解），还有一个大图（包含这两个图还有超级源点&超级汇点的图）。两个小图之间可以有很多条线连接分别处于两个图中的点，但是小图内部不能有边相连。

那花店橱窗这道题举例子，就是花是一个小图，花瓶是另外一个小图，显然花和花之间连线，这条线的意义啥都不是，同理花瓶。而花与花瓶相连，这条线的的意义就是美学值。

解决二分图匹配，需要用到匈牙利算法，静态链表实现起来丝毫不困难（在有了解决网络流，最短路径的经验之后），于是就有某道题的标程了。

速度编一下：

bool dfs(int u)

{

Int p=first[u];

while(p!=-1)

{

Int v=edge[p].to;

If(!vis[v])

{

Vis[v]=true;

If(match[v]==-1 || dfs(match[v]))

{

Match[v]=u;

Return true;

}

}

P=edge[p].next;

}

Return false;

}

Int main()

{

……

……

Memset(match,-1,sizeof(match));

For(int i=1;i<=cnt;i++)

{

Memset(vis,false,sizeof(vis));

If(dfs(i)) sum++;

}

//所求的sum就是二分图的最大匹配数

}

大概用了3分钟的时间吧，还是在word下编的，Dev-C++肯定会更快。

关于静态链表的数据结构，请见最大流问题或者最短路径。

有些题，看上去像是二分图匹配，实则上是求最大流（最小费用最大流）

比如说花店橱窗，这道题要求的是最大美学价值，怎么办？

设置一个超级源点和超级汇点

超级源点到每个花的流量都是1，费用都是0

每个花瓶到超级汇点的流量都是1，费用都是0

花与花瓶之间的连线，流量是1，费用是美学价值

然后最小费用最大流，得出美学价值。

在这里稍稍注意一下，SPFA算法和Kruskal一样，能求最小，同样也能求最长（后者为最大）所以，也能求最大费用最大流，不过是后话了。

下面就是上面算法中最难的部分——建模

模型的构造是每一个算法中最难的一部分，这可以在DP和图论中得到明显的体现。

所以具体问题，具体分析吧。

就是这样。