1 实验题目

问题描述：

给 n 个学生 1, 2, 3, … , n，m 个教授 1, 2, 3, … , m，以及 t 个时间段 1, 2, 3,…, t。每个学生 u 会在某个时间点 w 想见某个教授 v，教授 v 会根据自己在 w 时间点是否空闲决定是否见这个学生。问最多可以有多少次见面发生。

约束：

1. 一个教授不可以在相同时间内见两个或两个以上的学生
2. 一个学生不可以在同一时间见两个教授

输入可能有多组数据，以 EOF 结尾，每组数据格式如下：

第一行四个正整数 P、N、M、T。其中 P 表示有几次会面等待安排，N、M、T 含义和题目相同。接下来 P 行，其中第 j 行表示一次待安排的会面，格式如下：正整数 u[j]、v[j]、 w[j]，表示学生 u[j]在时间点 w[j]想见教授 v[j]。

输出格式如下：

对于每组数据，输出可安排的最大会面数。

数据范围：

1 <= P <= 10000 1 <= N <= 100 1 <= M <= 20 1 <= T <= 30

2 设计思想

经过一定的转化可以将其转化为二分图匹配问题，把每个学生的每个时间和每个教授的每个时间都看作一个节点，若有见面需求则将其连线。所有学生的点可以在集合 A 中，所有教授的点可以在集合 B 中，这样集合 A 与集合 B 就构成了二分图。此时可以使用解决二分图匹配的算法解决此问题。

我们也可以使用网络流算法，将 A 集合中的点连到源点 s 上，将 B 集合中的点连接到汇点 t 上，把 A 集合和 B 集合中的边改为有向边（从 A 指向 B）, 所有边的权值都设为一，在构造出的这个图上跑最大流算法即可得到结果。

将输入转化为合理的图是解决该问题的第一步，由于最大流算法涉及到 BFS, DFS 的搜索，构造残差图又涉及到增删边，所以使用邻接矩阵加邻接表（双向数组链表实现）的方法来实现图，这样遍历一个点的所有邻居为 O(deg(v)) ,访问一条边是否存在，增删新边均为
O(1)。效率较高。

最大流算法基于 FF 算法实现，分别使用 Edmond-Karp 最大流算法（即 SPFF）和 Dinic 最大流算法跑该问题，对比结果可以观察两个算法的效率。

3 测试结果

编译使用宏来分开编译两种算法：
使用 SPFF 最大流算法: g++ -std=c++11 main.cpp –DXJSPFF 使用 Dinic 最大流算法: g++ -std=c++11 main.cpp –DXJDINIC

输入文件为班级提供的标准测试样例，按照 in, in-small, in-big 的问题规模分别陈述测试结果（由于输入输出较为巨大，无法给出具体样例，可以在 https://github.com/EAirPeter/2017FallAlgo/tree/master/exp1 查看输入文件以及标准输出）：
自己的输出与班级标准输出一致（使用 diff 检查无不同），但运行时间有所不同：

in:	SPFF 算法： 0ms	Dinic 算法： 0ms
In-small:	SPFF 算法：50ms	Dinic 算法：55ms
In-big:	SPFF 算法: 18720ms	Dinic 算法:3156ms
In-big:（O2 优化）	SPFF 算法：5600ms	Dinic 算法：2847ms

由于 in-big 输入较大（十万条数据），所以开启 O2 优化。在 In-big 样例运行时，有 2300ms 左右的时间都用来读取输入，所以实际的最大流算法还是比较理想的。也可以很明显看出，Dinic 算法在较密集图上效率吊打 SPFF 算法（节省了五倍的时间），
但是 SPFF 算法也是 Dinic 算法的基石，并且在稀疏图上二者效率区别不大。

4 总结与体会

此次实验实现了两种最大流算法，加深了对最大流算法执行过程的体会，对比算法执行时间也加深了对渐进时间复杂度的体会. 在将输入转化为图的过程中第一次使用邻接表+邻接矩阵混合表示的方法, 感觉此方法效率不错, 但是处理双向链表十分容易出错, 需要写一个模板供以后使用.