研究生算法课课堂笔记

上课日期：2015.12.07　　第(1)节课

组长：段祎纯　　组员：崔治丞

组员：　　组员：

注意：请提交 Word 格式文档。

一．内容概要

1. Openjudge 上给出的两个模板的使用说明
2. 12 月 10 号模拟考试注意事项

二．详细内容

1. 网络流模板的使用说明

Dinic 算法，使用链式前向星，之前课上给出的是基于邻接矩阵的算法，而由于考试题的数据规模，使用邻接矩阵会超时。

对于模板，不需要知道实现即可使用，接口简单。

```
#define N (100+2)
#define M (N*N+4*N)
```

（1）N,M 都放在前面定义。N 是点数，考虑到新加入的源点和汇点，需要在原有基础上增加 2 个点， M 是边数。若 N 和 M 的使用散落在程序中，使用起来会很麻烦。所以统一定义在最前面，为程序中总的点数和边数，一定要设计好。

以 pigs 为例，最多 100 个顾客，再考虑源点和汇点 ，一共需要（100 + 2）=102 个点。对于边，最坏情况全联通 N * N ，超级源点到每个 custom 都有边，custom 到超级汇点都有边，还有反向边。所以最多需要 N * N + 4 * N 条边。只要 N 不太大，内存不超即可。

（2）若使用时出现 Runtime Error，不要认为模板错误，先检查 N, M 的定义是否正确。

（3）#define 中的()不要删掉

在进行网络流计算时，可能会遇到边的 Capacity 很大，计算得到的最大流 flow 超过了 INT_MAX，则可以将每条边的 Capacity 设置成 long long 类型，不过这样做使得每条边的存储容量增加了一倍。

解决方法是，每条边的 Capacity 仍然设为 int 类型，每次 dfs 调用的返回值也是 int，但最后 dinic 函数进行累加和返回的是 long long 类型

```
struct edge {
  int v, cap, next;
};
int dfs(int u, int t, int bn);

LL dinic(int s, int t) {
    LL max_flow = 0;
    ……
    return max_flow;
}
```

调用 dfs 求出所有的增广路径，是递归实现的，但不用考虑堆栈溢出的问题。

问题：如果 dfs 返回的最大流只能是 int 类型，但是当前层次图中所有的增广路径的流量和超过了 int 类型，会有问题吗？

答案：不会有问题。 正常情况下，dfs 调用一次后，层次图就变得不连通了。如果限制为 INT_MAX，那么一次 dfs 调用后，当前层次图可能仍然是连通的，此时 while 循环中会多调用几遍 dfs，效率上也许会有损失，但是正确性不会有问题。

相当于启动时，从源点开始，有一个虚拟边的 Capacity 为 INT_MAX，则图中的流量不会超过 INT_MAX。

其他可行方案：将每条边的 Capacity 除以 1000 或更多，相当于换用一种计量单位来表示，除非严格区分 Capacity 为极小值和极大值（比如 1 和 30 亿）。

解决某些问题时，某些边权重需要设为无穷大。比如 Pigs 中两个 customers 先后访问同一个猪圈，他们之间的边的 capacity 为无穷大，用 INT_MAX 即可。一种模拟无穷大的方式是找到图中某一个割，不要求是最小割，比如源点为 A，其他顶点都是 B，用这样的一个 cut 的容量来模拟无穷大。因为最大流的流量等于最小割的容量，不会超过这个割的容量，所以可以用这个 cut 的 capacity 为表示 infinite。但考试的时候**不要**这么写。有一道题，源点到其它点的 capacity 之和超过了 MAX_INT，C 不会报警，溢出后会变成负数再累加成较小的正数。所以还是要用 INT_MAX 来表示无穷大。

如 Project selection，各 projects 之间有依赖关系，每个 project 有个 profit，可正可负，找闭合子图。存在依赖的边的权重如果用 INT_MAX 表示无穷大，从理论上说有可能最小割把这条边切掉（这条边的代价比其它几条边之和要小）。考试中不用担心这一点，实际工作中如果使用这个模板要注意。

设计模板时存在 trade off，尽量没用 c++，基本用 c 写出，没封装

```
struct edge {
  int v, cap, next;
};
int head[N], level[N], cur[N];
int num_of_edges;
```

使用链式前向星，u 和 v 的边，u 不用记录，保存在 u 的链表中，v 为目标顶点，capacity 为容量。用数组的下标模拟连边，head 为链表的表头，level 为层次，cur 记录当前扫描到哪条边，对扫描进行优化，避免重复扫描，num_of_edges 为边的数量。

```
for (int i=cur[u]; i>=0; i=e[i].next)
```

这个代码设计的效率是比较高的。除非使用 preflow 的方法才有可能效率更高，并且只有当规模很大的稠密图时才有区别，因为 preflow 的时间复杂度可以达到 n^3

dinic_init 在使用代码之前进行初始化，每个测试用例使用前都要初始化

add_edge 一次性增加双向边，网络流代码都要求同时加两个方向的边

若有平行边，不用特殊处理，遇到一条就加一次，效率上有些小损失，但是保证正确。若 u、v 本来就有双向边，当成两个单向边，分别加也没问题。

这里要求两个边的容量之和不能超过 MAX_INT，否则增广过程中边的容量可能出现 overflow。

遇到 RE，看下标定义够不够大。另外，**assert 也**可能会造成 RE

```
assert(c1>=0 && c2>=0 && c1+c2>=0); // check for overflow
```

一对顶点之间如果一个方向的容量为 MAX_INT，反方向必须是 0，否则一个流过来，反向边的容量增加就 overflow 了。理论上说，两个方向的 capacity 都是无穷大的情况需要特殊处理，但是考试中没有这种情况。

print_graph 为 debug 程序：每行是图中顶点，括号外是相连的点的序号，括号里是 capacity

upstream 函数：s 为源点，n 为节点总数目，计算从源点可达的顶点数目，不需要知道汇点的序号。这个函数在 pig 题中用不到。它的作用是：在得到最大流后，计算顶点数最少的最小割：在残留网络中从源点开始遍历，能到达的点的个数。不包括源点本身。

对于多个测试用例，每个都要初始化，要在循环里面调用 dinic_init。

考虑鲁棒性，顶点序号是从 1 到 n，或者是从 0 到 n – 1 都能够正确运行，只要总顶点数（用于确定数组最大合法下标）正确就行。若顶点序号是从 1 到 n，可以增加源点为 0，汇点为 n + 1；若顶点序号是从 0 到 n – 1，可以增加源点为 n，汇点为 n + 1，两种方法一共都是 n + 2 个顶点。如果第 0 个顶点不用也可以，反正没有出入边，算法中不会有影响。

代码中需要传入源点 s 和汇点 t 的序号。

1. dinic_init
2. add_edge
3. dinic

Pigs 中，

```
if (npigs > 0) {
    add_edge(i, t, npigs, 0);
}
```

if (npig > 0)要不要都可以：增加两个方向都是 0 的边，也不会出错。

最大匹配模板

可以不用，使用最大流也可以解决，增加源点、汇点。

使用匈牙利算法，和课上讲的版本一样，只不过使用 global 变量来简化参数的传递。

init 每组测试数据都要调用, n1 是左边的点的数量，n2 是右边的

```
void init(int n1, int n2);
```

鲁棒性，下标从 0 或者 1 都可以，数组多分配了一个，多出来的顶点不影响正确性，因

为反正没有边相连。

dfs 做最大匹配
优化：先做贪心，再做匈牙利算法

使用方法：（奶牛问题）多个用例，循环里层做初始化，u、v 有连边就调用 add_edge，然后调用 hungarian 函数。若要知道具体匹配，L 数组和 R 数组即是。

模拟考试注意事项：

这次的机考，要自带草稿纸，与正式考试区别是正式考试是不准带草稿纸的。会统一发两张草稿纸，考试结束后要收回，文具都是要自己带。还有一个区别，模拟考试的教室分配在教学网上，具体座位随机安排，考场上宣布。而正式考试的教室分配不会提前公布。

若没有正式办选课手续，不要来模拟考试，机房位置有限。机房机器坏了的话，可临时换。Codeblock 配置有问题，没办法统一配置，助教会发邮件讲如何配置 codeblock

模拟考试只有 2 个小时，一共 5 道题。
1、2：热身题（并查集、递归、拓扑、优先队列）
3、4：网络流相关
5 ：动态规划（有一定复杂度）

期末考试，至少 6 道题（可能 7 道题）
题目分配与模拟考试类似，与模拟考试和作业不会有重复
热身题 2 道
网络流 2-3 道
动态规划 2 道
知识点不同，题目中英文都有

还有 2 次上机练习，没去过的同学建议去。