线性规划的简单应用和实现

浙江省杭州二中　　李宇骞

摘要

线性规划在实际生活中应用非常广泛，已经创造了无数的财富。但是它在竞赛中的应用很少。然而，我相信它的潜力很大，所以在这里向大家简单地介绍了线性规划的一些应用，以及如何实现解线性规划，以抛砖引玉，希望线性规划能够在竞赛中如同网络流一样熠熠生辉。

本文主要分三部分，第一部分简单地介绍了线性规划，给出了其定义；第二部分给出了一些简单的应用，以及一个线性规划的经典应用——多物网络流；第三部分是用单纯形（Simplex）算法实现解线性规划。

由于对大多数竞赛选手而言，写一个线性规划的程序比构造一个模型更为恐怖（虽然难度可能不及），并且单纯形法不是多项式级别的，不实践很难知道它的速度到底怎么样，所以本文着重于第三部分，较详细地描述了一些实现的细节，以及简单的证明，并且对单纯形法的运行速度做了一些实验，还与专业的数学软件 MATLAB 和 LINDO 做了对比，从一定程度上说明了单纯形法的速度是卓越的。同时，200 行左右的程序可以让大家不必那么担心编程的复杂度，某些情况下，100 行左右的程序就足够了。

关键字

线性规划（Linear programming）　　单纯形法（Simplex）
多物网络流（Multicommodity flow）

引言

“随著强有力的算法的发展与应用,线性规划能解决的问题也越来越来多。在历史上，没有哪种数学方法可以像线性规划那样，直接为人类创造如此巨额的财富，并对历史的进程发生如此直接的影响。”

孙捷，这位曾经执教于清华大学的美国华盛顿大学博士如此评价线性规划。他还举了这样一个实例：

在波斯湾战争期间，美国军方利用线性规划，有效地解决了部队给养和武器调运问题，对促进战争的胜利，起了关键的作用。难怪人们说，因为使用炸药，第一次世界大战可说是「化学的战争」；因为使用原子弹，第二次世界大战可说是「物理的战争」；因为使用线性规划，波斯湾战争可称为「数学的战争」。

线性规划在实际生活当中的威力已毋庸质疑，但是在信息学竞赛中，他的光芒还没有闪

耀在我们的眼前，让我们通过学习和了解，去渐渐感受它的光彩。

正文

第一部分　简介与定义

我们会遇到很多这样的问题：他们需要使目标最大化或者最小化；他们通常面临资源或者其它方面上的限制，或者必须在某些方面进行取舍而不能兼顾。如果这些问题的目标可以表示成一个线性的函数，它们的限制或者取舍可以表示成一些线性的等式或者不等式，那么我们就可以将这些问题描述成线性规划的问题。

首先来看一个实例：

假如你要竞选市长。要当上市长，你必须有 5 万的城市居民的投票、10 万郊区居民的投票以及 2.5 万农村居民的投票。

你有以下四种方案使你获得更多的投票：

1.建设道路

2.加强枪支管制

3.发放农业津贴

4.减免油税

并且，你对上述四种方案进行了评估，得到了在某一方案上开支的钱和某一区域内选民票数的变化的关系，如下表。表格中的某一项表示在对应方案上每开支 1 万元，对应区域中选票增加的数量，以千张为单位。

	城市	郊区	农村
建设道路	－2	5	3
枪支管制	8	2	－5
农业津贴	0	0	10
减免油税	10	0	－2

比如第一行第一列-2 代表在建设道路上每增加 1 万元的支出，会减少 2 千人的城市居民选票；第一行第二列 5 表示在建设道路上每增加 1 万元支出，会增加 5 千人的郊区居民选票；第二行第三列表示在枪支管制上每增加 1 万元支出，会减少 5 千人的农村居民选票。

你要用最少的支出来获得足够的选票当上市长，假设初始时，你的投票数都为 0。

这个问题的目标是要求开支最小，它的限制是选票必须达到最低限制，取舍关系是为了增加一个区域的居民投票而在一个项目上投资，可能会造成其它区域的居民投票减少。当然，这个问题还有一些潜在的限制，比如支出不能为负。

下面，我们把它描述成一个线性规划问题：

假设 4 种项目的支出分别为 x_1、x_2、x_3、x_4 万元，

目标：最小化 $x_1+x_2+x_3+x_4$（总支出最小）

限制：

$-2x_1+8x_2+0x_3+10x_4 >= 50$（城市居民）

$5x_1+2x_2+0x_3+0x_4 >= 100$（郊区居民）

$3x_1-5x_2+10x_3-2x_4 >= 25$（农村居民）

$x_1, x_2, x_3, x_4 >= 0$（开支不可能为负）

那么到底什么是线性规划呢？我们来看定义。

线性规划：在满足一些线性等式或者不等式的条件下，最优化一个线性函数。

线性函数：给定一些实数：$a_1, a_2, \ldots, a_n$ 和一些变量 $x_1, x_2, \ldots, x_n$，这些变量的线性函数 f 是 $f(x_1, x_2, \ldots, x_n) = a_1x_1+a_2x_2+\ldots+a_nx_n=\sum_{j=1}^{n} a_j x_j$

线性等式或者不等式：如果 $f(x_1, x_2, \ldots, x_n)$ 是一个线性函数，那么 $f(x_1, x_2, \ldots, x_n) = b$ 是一个线性等式，$f(x_1, x_2, \ldots, x_n) <= b$ 或者 $f(x_1, x_2, \ldots, x_n) >= b$ 是一个线性不等式。这些东西统称为线性约束。

线性规划的解：一个向量（$y_1, y_2, \ldots, y_n$），使得当 $x_i=y_i$ 时目标函数最优且满足条件。

线性规划的可行解：一个向量（$y_1, y_2, \ldots, y_n$），使得当 $x_i=y_i$ 时满足条件，但目标函数不一定最优。

线性规划的最优值：在满足条件的前提下目标函数的最优值。

一般情况下，我们可以把一个线性规划的问题写成如下形式：

最小化或者最大化 $f(x_1, x_2, \ldots, x_n)$

满足：

$p_i(x_1, x_2, \ldots, x_n) <= a_i$

$q_i(x_1, x_2, \ldots, x_n) = b_i$

$r_i(x_1, x_2, \ldots, x_n) >= c_i$

其中 f 是目标函数，p_i 是限制中所有的小于等于的线性不等式，q_i 是限制中所有的线性等式，r_i 是限制中所有的大于等于的线性不等式。

比如：

最大化 $2x_1-3x_2+3x_3$

满足：

$x_1+x_2-x_3 = 7$

$x_1-2x_2+2x_3 <= 4$

最小化 $2x_1+7x_2$

满足:

$x_1 = 7$

$3x_1+x_2 >= 24.5$

$x_2 >= 3$

$x_3 <= -1$

都是线性规划。

注意，线性规划中的系数不要求是整数或者有理数，可以是任何实数，并且线性规划的

最优解($y_1, y_2, \dots, y_n$)中的 y 也不一定要是整数或者有理数。如果 y 限定成整数，那么问题是 NPC 的，也就是现在为止还没有有效(多项式)解法。

第二部分　简单的应用

下面主要讲了三个简单的应用，前两个分别是相对比较简单的资源优化配置和最佳物资供给，最后一个是相对复杂的多物网络流。

一、　资源优化配置

有 m 种资源和 n 个项目，每个资源都是有限的，设它们的上限为 $b_j(1 <= j <= m)$。假设第 i 个项目做出 x_i 的成果量，可以获得 c_i*x_i 的收益，同时会消耗第 j 种资源 $a_{ij}*x_i$。求最大收益。

比如下面的这个问题就是一个资源优化配置问题：

某工厂现在分别有钢材、木材、塑料 b_1、b_2、b_3 吨，工厂可以生产 4 种产品，第 i 种产品每生产一吨可以获得 c_i 万的收益，但是要耗费 a_{i1} 吨钢材，a_{i2} 吨木材以及 a_{i3} 吨塑料。求工厂的最大收益。

我们将上述问题描述成线性规划的问题：

假设 4 种产品产量分别为 x_1、x_2、x_3、x_4 吨

最大化 $c_1x_1+c_2x_2+c_3x_3+c_4x_4$

满足:

$a_{11}x_1+a_{21}x_2+a_{31}x_3+a_{41}x_4 <= b_1$

$a_{12}x_1+a_{22}x_2+a_{32}x_3+a_{42}x_4 <= b_2$

$a_{13}x_1+a_{23}x_2+a_{33}x_3+a_{43}x_4 <= b_3$

$x_1, x_2, x_3, x_4 >= 0$

上述式子可以简写成：

最大化 $\sum_{i=1}^{4} c_i x_i$

满足：

$$\sum_{i=1}^{4} a_{ij} x_i <= b_j (1 <= j <= 3)$$

$x_i >= 0(1 <= i <= 4)$

下面是一般的资源优化配置问题的线性规划描述：

最大化 $\sum_{i=1}^{n} c_i x_i$

满足：

$$\sum_{i=1}^{n} a_{ij} x_i <= b_j (1 <= j <= m)$$

$x_i >= 0(1 <= i <= n)$

二、 最佳物资供给

有 m 种需求要满足，假设第 j 种需求至少要 b_j 的量才能满足。现在有 n 种物资，其中每单位第 i 种物资会提供给第 j 种需求 a_{ij} 的量。假设第 i 种物资供给了 x_i 单位，要支付费用 c_i*x_i 。在满足所有需求的前提下，使总费用最小。

下面的问题就是一个最佳物资供给问题：

一个人一天至少要摄入 b_1 克糖类、b_2 克脂肪、b_3 克蛋白质以及 b_4 克维生素。米饭的价格为每克 c_1 元，每克米饭会提供 a_{11} 克糖类、a_{12} 克脂肪、a_{13} 克蛋白质以及 a_{14} 克的维生素；蔬菜每克 c_2 元，每克蔬菜回提供 a_{21} 克糖类、a_{22} 克脂肪、a_{23} 克蛋白质以及 a_{24} 克维生素；肉类每克 c_3 元，每克肉类提供 a_{31} 克糖类、a_{32} 克脂肪、a_{33} 克蛋白质以及 a_{34} 克维生素。请问至少要多少钱才能满足人一天的营养需求？

上述问题可以描述为下面的线性规划问题：

假设人一天吃米饭 x_1 克、蔬菜 x_2 克、肉类 x_3 克

最小化 $c_1x_1+c_2x_2+c_3x_3$

满足：

$a_{11}x_1+a_{21}x_2+a_{31}x_3 >= b_1$

$a_{12}x_1+a_{22}x_2+a_{32}x_3 >= b_2$

$a_{13}x_1+a_{23}x_2+a_{33}x_3 >= b_3$

$a_{14}x_1+a_{24}x_2+a_{34}x_3 >= b_4$

$x_1, x_2, x_3, x_4 >= 0$

一般的最佳物资供给问题可以描述为如下的线性规划问题：

最小化 $\sum_{i=1}^{n} c_i x_i$

满足：

$$\sum_{i=1}^{n} a_{ij} x_i >= b_j (1 <= j <= m)$$

$x_i >= 0 (1 <= i <= n)$

三、 多物网络流

看到这个标题，很多人都会联想到一般的网络流：有着简单、高效的解法，并且用途广泛。一些经典的网络流模型常常巧妙得令人瞠目结舌。多物网络流虽然和网络流有很多的相同之处，但是它至今为止唯一的有效解法只有线性规划。然而，我相信它的用途绝不比网络流逊色，应该更强大，并且它的一些模型的构造将会比普通的网络流更令人吃惊，因为它不仅包含了只有一个源点和汇点的网络流，还可以应对有多个源点和汇点的网络流。

那么到底什么是多物网络流呢？其实上面已经略有提及。

多物网络流基本上和一般的网络流一致，唯一的区别就是多物网络流有 k 个源点和汇点，k 可能大于 1。假设第 i 个源点为 s_i，第 i 个汇点为 t_i （1 <= i <= k）。多物网络流的问题就是要求一个满足 s_i 到 t_i 的流量都为 f_i 的可行流。

下面是具体的定义：

多物网络流：有一个 V 个点、E 条边的有向图，假设第 e 条边从 u_e 到 v_e，并且拥有非负的流量限制 c_e。给出 k 个物品，第 i 个物品用(s_i, t_i, d_i)表示。这里，s_i 是第 i 个物品的源点，t_i 是第 i 个物品的汇点，d_i 是该物品要求的从 s_i 到 t_i 的流量。我们定义一个函数 f_i，$f_i(u_e,v_e)$

表示物品 i 在边 e 上的流量。这个函数满足流量守恒定律（除了源点 s_i 和汇点 t_i 之外，流进一个点的流量等于流出一个点的流量）。最后，n 个物品在一条边上的流量和不超过这条边的容量。我们需要确定一组满足这些条件的函数 f_i。

如果上面的定义不够直观，我们来看下面的这个实例：

某公司要用铁路运送 k 种物品，分别从城市 s_i 到 t_i，每个物品每天要送出 d_i。给出城市之间每天铁路的流量限制。假设物品可以任意地成若干份，从而可以分别从不同的线路走。求一个可行的运送方案。

用线性规划来描述多物网络流：

最小化：0

满足：

$$\sum_{i=1}^{k} f_i(u_e,v_e) <= c_e(1 <= e <= E)$$

$$\sum_{p:v_p=v} f_i(u_p,v_p) = \sum_{q:u_q=v} f_i(u_q,v_q)(1 <= i <= k, 1 <= v <= V \text{且} v \text{不等于} s_i \text{或} t_i)$$

$$\sum_{e:u_e=s_i} f_i(u_e,v_e) = d_i(1 <= i <= k)$$

$$f_i(u_e,v_e) >= 0(1 <= i <= k, 1 <= e <= E)$$

这里最小化 0 的意思就是只求一个满足条件的可行解，也就是目标函数中所有的系数都是 0。

（注：这里的描述和《算法导论》上不一样，主要考虑到可能有重边的情况）

在一般的多物网络流基础上，如果每条边还有一个花费，假设第 e 条边的单位花费为 $Cost_e$，那么在这条边上的花费就等于这条边上的流量和乘以 $Cost_e$，整个网络流的费用就是所有边的费用和。我们现在的问题不仅是要求一个可行流，还要求一个费用最小的可行流。上述问题就是一个最小费用的多物网络流问题。它的条件和多物网络流完全一样，只是目标函数不同：多物网络流最小化 0，多物最小费用流最小化 $\sum_{e=1}^{E} Cost_e \sum_{i=1}^{k} f_i(u_e,v_e)$。

接下来有一道改编自 NWERC2006-D 的多物最小费用流的例题：

在一个地图上，某铁路公司有 4 项目。第 i 个项目要建造一条从城市 s_i 到 t_i 的铁路。现在有一些铁路段可以供公司选择建造，每一段都是从一个城市到另一个城市的，且每一段铁路的造价是已知的。由于项目之间是独立的，每一段铁路只能被一个项目所拥有、建造，造好以后也只能被此项目所使用。求完成 4 个项目的最小费用。

我们将城市作为点，铁路段作为边进行构图，每条边的容量都是 1，单位费用为建造该铁路段的费用，k 等于 4，s_i 和 t_i 没有变，d_i 都是 1。

如果仔细看上面的这个模型，你会发现一个问题：够出来的图中，一个物品在一条边上的流必须是 1 或者 0，不能是其他的实数，但是多物网络流中一个物品在一条边上的流量可以是任意的实数。那么我们这样做求出来的答案是不是错的呢？很幸运，答案是对的，因为我们只要求最小费用，而不需要给出具体的方案。也就是说，我们的求出的目标函数的值一定等于最小的费用，但是 $f_i(u_e,v_e)$可能是一个非 0 或 1 的数。

为什么呢？

首先，我们构造的模型所求出来的目标函数的最小值用一定小于等于问题中的最小费用。因为具体的问题比我们构造的模型更严格——某物品在一条边上的流必须是 0 或者 1。也就是说，具体问题的解应用到我们构造的模型中一定满足条件，但是我们构造模型的解应用到具体的问题中可能不满足条件。

因此，如果有一个具体问题的解的费用等于我们构造出模型的最小的目标函数值，那么这个费用一定是具体问题的最小费用。换句话说，如果有一个方案能满足具体问题的条件，且费用就等于我们构造出的模型的最小的目标函数值，那么我们构造出的目标函数值就是具体问题的答案。

那么我们如何保证对于最小的目标函数值，能够构造出满足具体问题条件的方案呢？这个就要看第三部分中将要提到的单纯形（Simplex）算法的具体实现：单纯形算法对于我们构造的模型，给出的解一定是一个整数解。而由于边的流量限制，单纯形算法给出的解中一个物品在一条边上的流量不是 0 就是 1。所以单纯形算法给出的解刚好就是我们要找的方案。所以我们构造出的这个模型不管怎么样，解出来的目标函数值一定是最小费用，如果用单纯形法来解，还能给出一个具体的方案。

第三部分　单纯形算法的实现

下面一共有六个部分，其中前五个部分讲述单纯形算法，第一部分是线性规划的标准形式(Standard form)与松弛(Slack form)形式，第二部分是转轴（Pivot）操作，第三部分是单纯形算法的主程序，第四部分是初始化，第五部分是一些细节的实现和优化，以及一道简单的练习题。最后的第六部分是测试部分，记录了我写的程序和 LINDO 以及 MATLAB 之间的一些测试与实验。

一、　线性规划的标准形式(Standard form)与松弛形式(Slack form)

为了方便，我们希望将线性规划的形式更加统一，以便使以后的工作量更小一些。要将一个线性规划转化成另一种形式，首先要保证这两个形式是等价的。所以，我们先来看等价的定义：

假设两个线性规划分别是 A 和 B。如果两个线性规划都是要最大化目标函数，或者最小化目标函数，那么它们等价的充要条件是：对于 A 的任何一个可行解，B 也有一个可行解，使得 A 和 B 的目标函数值一样；对于 B 的任何一个可行解，A 也有一个可行解，使得 A 和 B 的目标函数值一样。此时 A 的最优值等于 B 的最优值。如果一个是最大化目标函数，一个是最小化目标函数，那么他们等价的充要条件是：对于 A 的任何一个可行解，B 也有一个可行解，使得 A 的目标函数值是 B 的相反数；对于 B 的任何一个可行解，A 也有一个可行解，使得 A 的目标函数值是 B 的相反数。此时，A 的最优值等于（-1*B 的最优值）。

大家可以简单地理解为如果 B 是由 A 经过一些可以接受的变换得到的，那么 A 和 B 是等价的。这里可以接受的变换包括一些不等式的变换（左右同乘以-1，移项等）；等式的带换；给目标函数乘以-1，并且改最大化为最小化或者改最小化为最大化。

接下来，我们来看标准形式的定义：

目标函数是最大化的，所有的线性约束都是小于等于的不等式，所有的变量都有非负的限制。

我们可以这样写一个线性规划的标准形式：

最大化 $f(x_1, x_2, \ldots, x_n)$

满足：

$x_i >= 0(1 <= i <= n)$

$p_i(x_1, x_2, \ldots, x_n) <= a_i(1 <= i <= m)$

然后，我们来看如何将一个不是标准形式的线性规划变为标准形式。

一个线性规划会有哪些地方不符合标准形式呢？

1、 目标函数是最小化的

2、 约束中有线形等式

3、 约束中有大于等于的线形不等式

4、 有些变量没有非负的限制

我们来各个击破。

1、如果线性规划 A 是最小化 $\sum_{i=1}^{n} c_i x_i$ ，我们建立一个新的线性规划 B，B 的约束和 A 完全一样，目标是最大化 $\sum_{i=1}^{n} -c_i x_i$ 。那么 A 和 B 显然是等价的。

2、如果线性规划 A 中有一个条件是 $q_i(x_1, x_2, \ldots, x_n) = b_i$，那么我们建立一个新的线性规划 B，其它都和 A 一样，只有 $q_i(x_1, x_2, \ldots, x_n) = b_i$ 换成了 $q_i(x_1, x_2, \ldots, x_n) <= b_i$ 和 $q_i(x_1, x_2, \ldots, x_n) >= b_i$。那么显然 A 和 B 是等价的。

3、如果线性规划 A 中有一个条件是 $\sum_{i=1}^{n} a_i x_i >= b$ ，那么我们建立一个线性规划 B，其它都和 A 一样，只有 $\sum_{i=1}^{n} a_i x_i >= b$ 换成了 $\sum_{i=1}^{n} -a_i x_i <= -b$ 。那么显然 A 和 B 是等价的。

4、如果线性规划 A 中的一个变量 x_i 没有非负的限制，那么我们建立一个新的线性规划 B，其它都和 A 一样，只有所有的 x_i 都换成了两个新的变量的差——$(x_{i1}-x_{i2})$，且加上了 $x_{i1} >= 0$、$x_{i2} >= 0$ 的条件。那么 A 和 B 也是等价的，因为 A 中任何一个可行解$(x_1, x_2, \ldots, x_i, \ldots, x_n)$，如果 $x_i >= 0$，那么直接令 $x_{i1} = x_i, x_{i2} = 0$；如果 $x_i < 0$，那么令 $x_{i1} = 0, x_{i2} = -x_i$ 即可。而 B 中的任何一个可行解显然可以简单的转化为 A 中的一个可行解，使得目标函数值一样。

那么我们怎么更方便得来表示标准形式呢？只需要 1 个矩阵和两个向量。

第一个向量 c，有 n 个元素，表示目标函数为 $\sum_{i=1}^{n} c_i x_i$ 。第二个向量为 b，有 m 个元素。然后是矩阵 a，大小为 n*m。a 和 b 共同表示了所有的约束，即有 m 个不等式，第 i 个为 $\sum_{j=1}^{m} a_{ij} x_j <= b_i$ 。

在单纯形算法中，为了方便，通常把标准形式转换成松弛形式。不必担心，标准形式到松弛形式的转换非常方便：

我们新加入 m 个变量，假设为 x_{n+1} 到 x_{n+m}。其中 $x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j$ 。现在，线性规划

可以表示成：

最大化：$f(x_1, x_2, \ldots, x_n)$

满足：

$$x_{n+i} = b_i - \sum_{j=1}^{m} a_{ij} x_j$$

$x_i >= 0$（$1 <= i <= n+m$）

我们再来看松弛形式的定义和表示。

一个松弛形式可以用一个大小为 m 的整数集合 B 和一个大小为 n 的整数集合 N，以及一个大小为(n+m)*(n+m)的实数矩阵 A，两个大小为（n+m）的实数向量 c 和 b 还有一个实数 v 表示。代表如下这个线性规划：

最大化：$\sum_{i \in N} c_i x_i$ +v

满足：

$$x_i = b_i - \sum_{j \in N} A_{ij} x_j (i \in B)$$

$x_i >= 0(1 <= i <= n+m)$

总的说，松弛形式可以用一个六元组（N, B, A, b, c, v）表示。

对于线性规划

最大化：$f(x_1, x_2, \ldots, x_n)$

满足：

$$x_{n+i} = b_i - \sum_{j=1}^{m} a_{ij} x_j$$

$x_i >= 0$（$1 <= i <= n+m$）

来说，v=0，N 包含 1 到 n 的整数，B 包含 n+1 到 n+m 的整数，$A_{n+i,\ j}=a_{ij}$。可以看出，N 是所有等式中右侧变量下标的集合，同样也是目标函数中所有变量下标的集合；而 B 是所有等式左侧的变量的下标集合。我们通常叫下标在 N 中的变量为非基（Nonbasic）变量，而叫下标在 B 中的变量为基（Basic）变量。

在算法进行的过程中，N 和 B 是在变化的，一个 B 中的元素会和 N 中的一个元素调换，也就是一个左边的变量到了右边，而一个右边的变量到了左边，同时改变 A、b、c、v，使得新的形式和原来的是等价的。这就是我们接下来要说到的转轴（Pivot）操作。

对于一个 b 中元素都是非负数的松弛形式，都有一个可行解，那就是令所有的非基变量为 0，此时目标函数值就是 v，基变量 x_i 就等于 b_i。我们将通过有限次转轴操作，得到一个这样的松弛形式，使得该可行解就是线性规划的解，v 就是线性规划的最优值。

二、 转轴操作(Pivot)

转轴操作需要两个参数，分别是 l 和 e，表示 B 中的 l 和 N 中的 e 进行了调换。

在原来的松弛形式中，肯定有一个等式：$x_l = b_l - \sum_{j \in N} A_{lj} x_j$。$x_l$ 和 x_e 左右调换以后等式就变为 $A_{le} x_e = b_l - \sum_{j \in N 且 j \neq e} A_{lj} x_j - x_l$，也就是 $x_e = b_l / A_{le} - \sum_{j \in N 且 j \neq e} (A_{lj} / A_{le} * x_j) - x_l / A_{le}$。再将该的等式带入到目标函数以及其它的等式中，就达到了调换的目标。

上面就是转轴操作的大概过程，下面是具体的伪代码：

```
pivot(l, e)
    tb[e] = b[l]/A[l][e];
    tA[e][l] = 1/A[l][e];
    for（i 属于 N 且 i 不等于 e）
        ta[e][i] = A[l][i]/A[l][e];
    //上述部分得到了我们需要的以 xe 为左侧变量的等式
    for(i 属于 B 且 i 不等于 l)
        tb[i] = b[i]-A[i][e]*tb[e];
        For(j 属于 N 且 j 不等于 e)
            tA[i][j] = A[i][j]-A[i][e]*tA[e][j];
    //上述部分将等式带入了其它的等式当中
    tv = v+tb[e]*c[e];
    tc[l] = -tA[e][l]*c[e];
    for(i 属于 N 且 i 不等于 e)
        tc[i] = c[i]-c[e]*tA[e][i];
    //上述部分将等式带入了目标函数
    v = tv;
    N = N 除去 e 加上 l;
    B = B 除去 l 加上 e;
    A = tA;
    b = tb;
    c = tc;
```

三、 主程序

单纯形算法的主程序分两块，分别是初始化，最优化。初始化过程对于当前一个任意的松弛形式，得到一个 b 都为非负数的松弛形式，或者得到该线性规划无解。我们将稍后来讲这个过程。最优化过程对于当前一个 b 都为非负数的松弛形式，得到一个 v 就是最优值且 b 都是非负数的松弛形式，或者得到该线性规划的解为无穷大。伪代码如下：

```
simplex
    init;
    opt;
```

那么下面，我们将主要来讲 opt 这个过程（这里将 opt 过程看成一个整体，而非直接展开来写，是因为在 init 过程中，我们还将用到 opt，为了避免同一个东西写两次，我们将它看成一个整体）。

opt 过程大致上就是不停选取一个合适的 e，然后再选取一个合适的 l，进行 pivot 操作，直到我们肯定线性规划的最优值为无穷大或者已经得到了解。

我们先来看 opt 过程的伪代码，然后再来仔细分析。

```
opt
    while (true) do
        if (所有的 i 属于 N 都有 c[i] <= 0)
            return  已经找到了解；
        else  选取一个 e 使得 c[e] > 0;
        delta =  无穷大;
        for(i 属于 B)
            if (A[i][e] > 0 and delta > b[i]/A[i][e])
```

```
            delta = b[i]/A[i][e];
            l = i;
        if (delta ==  无穷大) return  最优值无穷大;
        else pivot(l, e);
```

首先，opt 过程当中，初始时 b 全部都是非负数，且由 l 的选择可得任何一次 pivot 操作后，b 仍然全部都是非负数。为什么 l 的选择保证了这点呢？因为我们在选择 l 时，保证 b[l]/A[l][e]是最小的。只要你再仔细看一下 pivot 操作中如何生成新的松弛形式的 b，你将很快能理解这点。所以证明略去。

因为任何一步中，b 都是非负数，所以所有非基（Nonbasic）变量都为 0 显然都是一个可行解。而取这个可行解的时候，目标函数就是 v。在 opt 过程中，v 不会变小，因为我们选取的 e 满足 c[e] > 0，且 x_e 原本等于 0，不可能再变小。事实上它在绝大多数 pivot 操作后都会增大，当然 v 也有可能不变。这是因为我们在保证 b 不会出现负数的情况下让 x_e 增加得最多。正因为这点，单纯形法可以解决多物最小费用流的那道例题。因为每次流量都会增加到最大，那么就像网络流的增广算法一样，整数容量的边上不可能增加出非整数的流量。那么有没有可能 v 一直不变，造成程序死循环呢？这种事情发生的概率极小，并且只要稍加小心就能避免——选取 e 和 l 的时候遵循字典序优先，也就是 e 小的优先，如果 e 一样，l 小的优先。这就是 Bland's 法则。

我们判断无穷大的时候，是因为有一个非基变量 x_i，使得 c[i]大于 0，且对于任何 j 属于 B，都有 A[j][i]小于 0。此时，我们让其它非基变量都为 0，x_i 为无穷大的时候，所有的基变量显然是满足非负条件的（此时对于任何 j 属于 B，有 $x_j = b_j - A[j][i]*x_i$），此时目标函数无穷大。所以无穷大的判断是正确的。

那么当对于所有的 i 属于 N，都有 c[i] <= 0 时，v 为什么就是最忧值呢？严格的证明很麻烦，但是大家可以很方便地理解这一事实：首先，一但现在的非基变量的值都确定了，那么基变量的值也就定了，也就是说，一个线性规划的可行解可以由现在所有非基变量的值确定；因为变量都有非负限制，现在非基变量都为 0，所以它们只能变大，如果它们其中一些变大一点的话，由于目标函数中这些变量的系数是小于等于 0 的，v 肯定不会变大。所以 v 就是最优值。

最后我们来看一下时间复杂度。用了 Bland's 法则以后，我们的程序可以避免死循环，也就是一种松弛形式最多只出现一次，这里一种松弛形式可以由 N 中的元素确定（因为一旦 N 给定，B、A、a、b、c 都是可以唯一确定的），所以松弛形式最多只有 C_{n+m}^{n} 个，所以复杂度就是 pivot 的复杂度乘以 C_{n+m}^{n}。这里给出的只是上限，事实上它的时间复杂度很低。

四、 初始化

如果原本 b 就都是非负数，那么直接返回原来的形式就可以了，否则，为了得到一个 b 都大于等于 0 的初始形式，我们要构造一个新的线性规划——辅助问题。

假设原来的松弛形式中的条件是：

$$x_i = b_i - \sum_{j \in N} A_{ij} x_j (i \in B)$$

那么我们在辅助问题中，条件是：

$$x_i = b_i - \sum_{j \in N} A_{ij} x_j + x_0 (i \in B)$$

目标是最大化-x_0。

也就是辅助问题中的 N 比原问题的 N 增加了一个 0，B 不变，b 不变，对于所有的 i 属

于 B，都有 A_{i0} = -1，其它的 A 不变(此时的 A 大小其实是（n+m+1）*(n+m+1)，下标从 0 到 n+m)，c 中只有 c_0 = -1，其它都为 0。

由于 x_0 有非负限制，所以此线性规划的最大值不会超过 0。如果目标函数最大值为 0，也就是 x_0=0，那么 $x_i = b_i - \sum_{j \in N} A_{ij} x_j + x_0 (i \in B)$ 就是 $x_i = b_i - \sum_{j \in N} A_{ij} x_j (i \in B)$，和原来的条件一样。此时的松弛形式必然满足所有的 b 都大于等于 0。同时，如果最大值小于 0，就意味着条件是无法满足的（$b_i - \sum_{j \in N} a_{ij} x_j (i \in B)$不可能全部为非负数），那么原来的线性规划必然是无解的。所以，我们只要把这个线性规划给解出来就可以得到满足条件的松弛形式或者得到无解。

但是我们现在仍然有同样的问题，初始时 b 会有负数。幸运的是，我们现在有很简单的办法可以得到一个松弛形式，使得 b 中没有负数：

假设 b 中最小的为 b_l，那么进行一次 pivot(l, 0)，b 中就没有负数了。这个结果很显然，这里就不细讲了。

接下来，我们只要调用 opt 过程就解出了该线性规划，此时如果辅助问题的最优值小于 0，那么直接返回无解；如果最优值等于 0，那么直接令原来的松弛形式 N 等于当前辅助问题的 N，B 等于当前的 B，A 等于当前的 A，b 等于当前的 b。这是因为辅助问题的限制条件和原问题是一样的。不过我们还要把 x_0 去掉。如果此时 0 属于 B，那么任意选一个 e 属于 N，进行一次 pivot(0，e)操作。因为 $x_0 = b_0 = 0$，所以这次操作对 b 其实没有什么影响，我们的目的是让 x_0 成为非基变量。此时，因为 x_0=0，我们可以把 0 从 N 中除去（注意，之所以不在 B 中除取是因为 B 中每一个变量都代表着一个约束，一个等式，如果把 B 中的 0 去掉，就以为着少了一个条件限制，导致答案出错），然后 A 的大小再变回(n+m)*(n+m)，其实就是把下标中有 0 的都扔掉。现在剩下来的唯一问题就是调整原松弛形式的 c——目标函数。对于原问题的 c 中所有的变量 x_i，如果 i 属于当前的 B，那么我们将 $x_i = b_i - \sum_{j \in N} A_{ij} x_j$ 带入。如此，目标函数也和原来等价了。这样，我们就完成了初始化的任务：产生一个 b 都为非负数的等价的松弛形式。

下面是伪代码：

```
init
    找到 b 中最小的元素 b[l];
    if (b[l] >= 0) return;//原问题已经满足 b 都大于等于 0
    origC = c;//记录下原来的目标函数，以便还原
    N 加上 0。
    for(i 属于 B)
        A[i][0] = -1;
    for(i 属于 N)
        c[i] = 0;
    c[0] = -1;
    //辅助问题已经构造完毕
    pivot(l, 0);//辅助问题的 b 现在都为非负数了
    opt;//把辅助问题解出来
    if (v < 0) return  无解；//辅助问题的最优值小于 0
    把 0 除去；//注意 0 在 B 中的情况
    c = origC;
```

```
for(i 满足 c[i] > 0 且 i 属于 B)
    v = v+c[i]*b[i];
    for(j 属于 N)
        c[j] = c[j]-A[i][j]*c[i];
    c[i] = 0;
//c 已经被还原
return 已经成功初始化;
```

五、 细节与优化

首先，由于单纯形算法涉及实数运算，所以精度误差是无法避免的。因此在实现当中，一些 0 免不了要用极小量 eps 代替，从而避免由极小的精度误差而导致很荒谬的答案。

当然，我们最关心的还是它的速度。单纯形算法的主要操作是 pivot，一个单纯形算法所调用的 pivot 操作越少，那么程序显然就越快。但是怎么才能让 pivot 操作更少呢？一个最简单的想法就是在 opt 过程中，对 l 和 e 的选择加以优化。其中最简单的，就是贪心，选取一组 e,l 使得 v 增加得最大，如果 v 增加得一样大，再遵循 Bland's 法则。这里要注意的是，增量的初始最大值不能定义为 0，因为增量为 0 也是要进行 pivot 操作的，而不能判断为已经到达了最优值。事实上这点简单的优化对速度提高不少，可以看第六部分的测试——加贪心优化和不加贪心优化的对比。

下面，我们来看一道来自 UVA 的题目，编号是 10498。

题目大意：某人要买套餐给 m 个人吃，每个人吃的套餐必须都是一样的。套餐由 n 种食品组成，每种食品的单位价格是已知的。由于口味不同，每个人每得到一单位的某种食品获得的满足度可能不一样，这些也是已知的。每个人满足度是有上限的，上限已知。求在不超过任何人的满足度上限的条件下，此人最多能花多少钱。（某种食品可以买实数单位）

输入为 n，m，接下来 n 个数字，表示每单位食品的价格，然后是 m 行，每行 n+1 个数字，第 i 行前 n 个数字代表第 i 个人得到一单位某种食品所获得的满足度，最后一个数字代表满足度上限。

输出一行，为最高价格，上取整。

这是一道非常显然的线性规划题目，给出的就是标准形式，非常方便。并且，该题的初始的 b 就全部都是非负的，所以连 init 过程都不用写。这道题的整个程序只有 100 行多一点，具体的程序可以参考附录。

六、 测试

下面是我的程序和数学软件 LINDO 和 MATLAB7.0 在解线性规划上的对比。其中 LINDO 用的也是单纯形法，MATLAB7.0 用的是内点法。在开始之前，我先简单介绍一下线性规划另外两种算法，它们都是多项式级别的：

椭球法（Ellipsoid）：理论分析出来是多项式级别的算法，但是实际效果不及单纯形法。一篇文章里这样说到：许多人拿出越来越大的问题，让椭球算法去解。奇怪的是，在所有的计算实验中，椭球法都败在单纯形法手下。所谓姜还是老的辣，在理论上「坏」的单纯形法，在实际上表现，远胜於理论上「好」的椭球法。

内点法(Interior-point)：据说理论上是多项式级别的，实际效果也比单纯形法好。但可惜的是，我没有将其和单纯形法做过有效的对比：MATLAB 用的就是这个算法，但是当我想用 MATLAB 和单纯形法一较高下的时候，MATLAB 却可能因为大数据读入的问题害得我电脑很慢。

这里测试所用的数据都是用 gen.cpp（见附录）程序随机生成的，虽然数据不是很强，

但是我相信体现程序的速度还是没有问题的。由于在这里只是大致体现单纯形算法的速度，所以精度有限，误差在 1 秒以内。

下面是我的程序与 LINDO 和 MATLAB 在一些比较小的数据下的对比。由于 MATLAB 在（100，100）这个规模的数据的时候已经很慢，所以更大的数据就没有让 MATLAB 跑。这里我的程序用的是没有加贪心优化的版本。

变量个数 n / 限制个数 m	50	100
50	我的程序 瞬间 MATLAB7.0 瞬间 LINDO 瞬间	我的程序 瞬间 MATLAB7.0 17 秒 LINDO 瞬间
100	我的程序 瞬间 MATLAB7.0 17 秒 LINDO 瞬间	我的程序 一闪 MATLAB7.0 30 秒 LINDO 一闪

下面是更大的数据时，我的普通程序与加了贪心优化的程序和 LINDO 的对比

变量个数 n / 限制个数 m	200	300
200	普通程序： 8 秒，pivot 操作 5396 次 加贪心优化的程序： 1 秒，pivot 操作 336 次 LINDO： 1 秒，迭代次数 558	普通程序： 11 秒，pivot 操作 5310 次 加贪心优化的程序： 1 秒，pivot 操作 349 次 LINDO： 1 秒，迭代次数 578
300	普通程序： 28 秒，pivot 操作 7864 次 加贪心优化的程序： 2 秒，pivot 操作 485 次 LINDO： 2 秒，迭代次数 819	普通程序： 105 秒，pivot 操作 18243 次 加贪心优化的程序： 7 秒，pivot 操作 947 次 LINDO： 7 秒，迭代次数 1369

上面表格中，之所以普通程序和加贪心优化的程序的运行时间比不等于 pivot 的操作次数比是因为加贪心优化的程序每一次 pivot 操作之前比普通程序要多干一件事情——选取（l,e）使得增量最大。

下面是极限数据时加贪心优化的程序和 LINDO 的对比

变量个数 n / 限制个数 m	400	500

400	加贪心优化的程序： 23 秒，pivot 操作 1368 次 LINDO： 17 秒，迭代次数 2487	加贪心优化的程序： 22 秒，pivot 操作 1078 次 LINDO： 11 秒，迭代次数 1770
500	加贪心优化的程序： 36 秒，pivot 操作 1521 次 LINDO： 24 秒，迭代次数 2686	加贪心优化的程序： 53 秒，pivot 操作 1837 次 LINDO： 59 秒，迭代次数 4022

从上表当中，可以看出加贪心优化的程序在操作次数上较少，但是总时间上经常不及 LINDO，我想这很有可能是因为 LINDO 在另一方面的优化很强——pivot 过程本身运行一次的速度。这里我没有仔细研究过，只是随便看了些资料，大概和逆矩阵有些关系，这个就留给大家自己探索。

总结

线性规划解决的问题通常要求一个满足一些线性约束的可行解或者某个目标函数的最优值，这个目标函数必须是线性的，比如资源优化配置、最佳物资供给以及多物网络流。线性规划可以用单纯形法来解，虽然它的时间复杂度不是多项式级别的，但是它的实际效果很好，对于 100 个变量，100 个约束的线性规划也几乎瞬间出解。单纯形法中，对于 l、e 的选择加以贪心的优化，可以得到非常好的优化效果，使速度有明显提升。还有很多题目中，初始化是没有必要进行的，此时单纯形法的整个程序非常短，很容易实现。

线性规划在竞赛中的应用很少，但是毫无疑问它在实际中的应用是很广泛的。因此，我在这里向大家粗略地介绍了线性规划，希望大家有所了解。但是由于缺乏竞赛中的经典例题，线性规划的竞赛实用性和技巧性无法与其它内容相媲美。希望大家以后多思考，能够使得线性规划像网络流一样，在竞赛中熠熠生辉。

感谢

（还在增加当中……）

参考文献

《算法导论》（《Introduction to Algorithms》）
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein 著
《管理前沿文库 管理线性规划》

张惠恩著

附录

单纯形法源程序：Lp.cpp

加入贪心优化后的源程序：LP_greedy.cpp

UVA10498 源程序：10498.cpp

数据生成程序：Gen.cpp

将输入文件转换为 Matlab 可识别的命令的程序：Translator.cpp

将输入文件转换为 Lindo 可识别的文件的程序：Translator.cpp

将一般的形式转换为标准形式的程序：ToStandard.cpp

所有的程序源文件以及应用程序都在文件夹 LinearProgramming 中，请查看 Readme.txt 以了解更具体的概况。

UVA10498 Happiness 英文原题：
（台湾地震，暂时拿不到题目）