浅谈二分图最大权匹配

徐翊轩

江苏省常州高级中学

July 30, 2020

Contents

问题定义

二分图：二分图是能够分成左右两边的无向图，记其左侧点集为 L，右侧点集为 R，则对于边集中的每一条边 $(u_i, v_i, w_i) \in E$，应当满足 $u_i \in L, v_i \in R$，其中 w_i 表示边 i 的边权。

问题定义

二分图：二分图是能够分成左右两边的无向图，记其左侧点集为 L，右侧点集为 R，则对于边集中的每一条边 $(u_i, v_i, w_i) \in E$，应当满足 $u_i \in L, v_i \in R$，其中 w_i 表示边 i 的边权。

二分图最大权匹配：选择一个边集 E 的子集 S，满足 S 中的任意两条边没有公共点，并且最大化 S 中每条边的边权之和，我们称 S 为二分图的最大权匹配。

问题定义

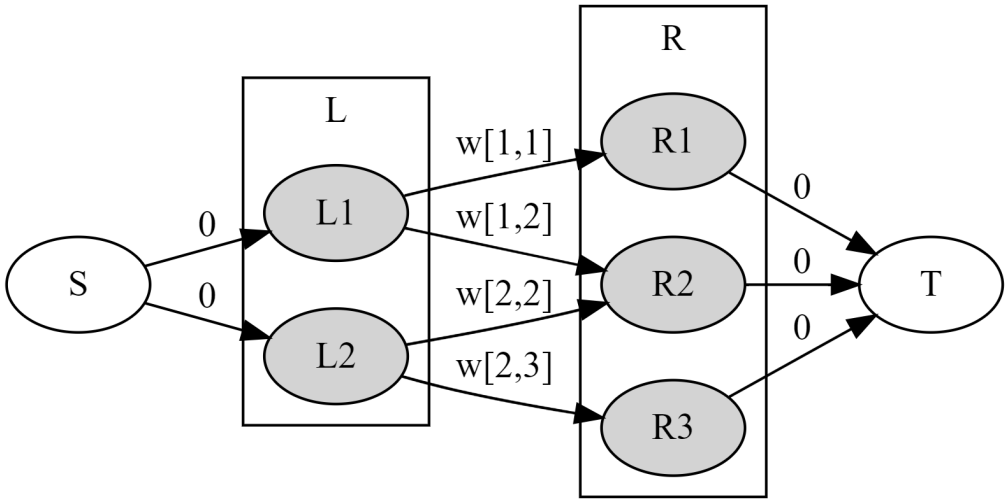
二分图：二分图是能够分成左右两边的无向图，记其左侧点集为 L，右侧点集为 R，则对于边集中的每一条边 $(u_i, v_i, w_i) \in E$，应当满足 $u_i \in L, v_i \in R$，其中 w_i 表示边 i 的边权。

二分图最大权匹配：选择一个边集 E 的子集 S，满足 S 中的任意两条边没有公共点，并且最大化 S 中每条边的边权之和，我们称 S 为二分图的最大权匹配。

二分图最大权完美匹配：不失一般性地，我们令 $|L| \leq |R|$，那么，选择一个边集 E 的子集 S，满足 $|S| = |L|$，且 S 中的任意两条边没有公共点，并且在此基础上最大化 S 中每条边的边权之和，我们称 S 为二分图的最大权完美匹配。

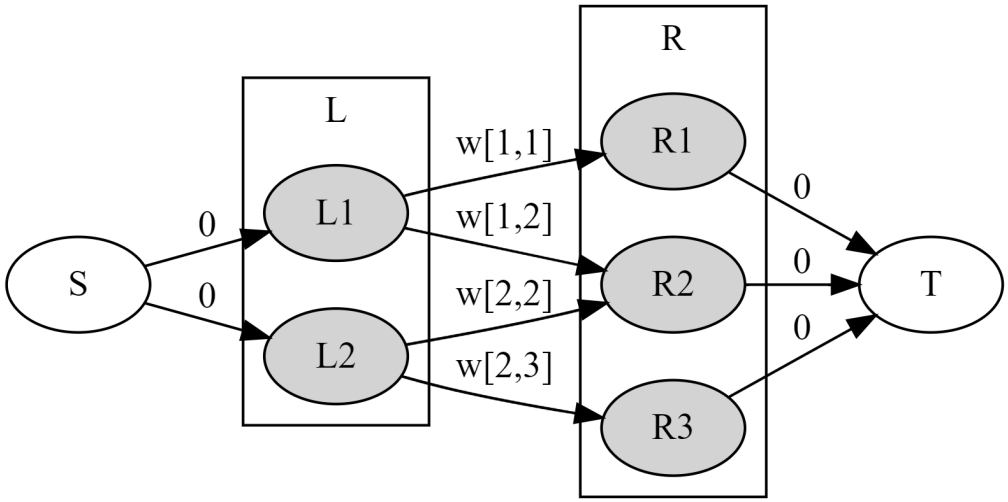
SPFA 费用流

我们可以用最大费用流来解决二分图最大权匹配问题，如下图：



SPFA 费用流

我们可以用最大费用流来解决二分图最大权匹配问题，如下图：



但是，常见的费用流算法是基于 SPFA 的单路增广算法，其最坏情况下时间复杂度为 $O(F \times N \times M)$，其中 F 表示最大流的大小，N, M 分别表示图的点数和边数。因此，令两侧的点数为 N，在解决二分图最大权匹配问题时，其最坏时间复杂度为 $O(N^4)$。

Dijkstra 费用流

除了常见的 SPFA 费用流外，也存在一种在最初进行一次 SPFA 算法后，将图转化为非负权图，从而使用 Dijkstra 算法代替 SPFA 算法进行单路增广，达到降低最坏情况下时间复杂度的费用流算法，其最坏情况下时间复杂度为 $O(N \times M + F \times M \log M)$ 。

Dijkstra 费用流

除了常见的 SPFA 费用流外，也存在一种在最初进行一次 SPFA 算法后，将图转化为非负权图，从而使用 Dijkstra 算法代替 SPFA 算法进行单路增广，达到降低最坏情况下时间复杂度的费用流算法，其最坏情况下时间复杂度为 $O(N \times M + F \times M \log M)$ 。

虽然最坏情况下的时间复杂度更优，但实际应用中，由于 SPFA 算法的复杂度上界难以达到，而费用残量网络中的边权也始终都在变化，Dijkstra 费用流的实际运行效率往往不如普通的 SPFA 费用流。

Dijkstra 费用流

除了常见的 SPFA 费用流外，也存在一种在最初进行一次 SPFA 算法后，将图转化为非负权图，从而使用 Dijkstra 算法代替 SPFA 算法进行单路增广，达到降低最坏情况下时间复杂度的费用流算法，其最坏情况下时间复杂度为 $O(N \times M + F \times M \log M)$ 。

虽然最坏情况下的时间复杂度更优，但实际应用中，由于 SPFA 算法的复杂度上界难以达到，而费用残量网络中的边权也始终都在变化，Dijkstra 费用流的实际运行效率往往不如普通的 SPFA 费用流。

不过值得一提的是，在处理 $M = O(N^2)$ 的图上的问题时，我们可以**去掉 Dijkstra 算法的堆优化**，从而得到 $O(F \times N^2 + N^3)$ 的时间复杂度，那么，在解决二分图最大权匹配问题时，其最坏时间复杂度就降为了 $O(N^3)$ ，是要优于 SPFA 费用流的。

算法流程

Dijkstra 费用流算法的主要思想在于维护满足最短路性质的数组 h_i，即保证 $h_s = 0$，且对于图中的任意一条边 (u, v, w)，均有 $h_u + w \geq h_v$，也即 $h_u + w - h_v \geq 0$。

算法流程

Dijkstra 费用流算法的主要思想在于维护满足最短路性质的数组 h_i，即保证 $h_s = 0$，且对于图中的任意一条边 (u, v, w)，均有 $h_u + w \geq h_v$，也即 $h_u + w - h_v \geq 0$。

那么，若令每条边的新权 $w' = h_u + w - h_v$，我们便得到了一张非负权图，并且，不难发现原图中任意一条 s 到 t 的，长度为 $dist$ 路径，在新图中的长度均为 $dist - h_t$。

算法流程

Dijkstra 费用流算法的主要思想在于维护满足最短路性质的数组 h_i，即保证 $h_s = 0$，且对于图中的任意一条边 (u, v, w)，均有 $h_u + w \geq h_v$，也即 $h_u + w - h_v \geq 0$。

那么，若令每条边的新权 $w' = h_u + w - h_v$，我们便得到了一张非负权图，并且，不难发现原图中任意一条 s 到 t 的，长度为 $dist$ 路径，在新图中的长度均为 $dist - h_t$。

因此，对于确定的 t，最小化 $dist - h_t$，相当于最小化了 $dist$，从而在新图上的最短路所经过的边集就是原图上的最短路所经过的边集。

算法流程

Dijkstra 费用流算法的主要思想在于维护满足最短路性质的数组 h_i，即保证 $h_s = 0$，且对于图中的任意一条边 (u, v, w)，均有 $h_u + w \geq h_v$，也即 $h_u + w - h_v \geq 0$。

那么，若令每条边的新权 $w' = h_u + w - h_v$，我们便得到了一张非负权图，并且，不难发现原图中任意一条 s 到 t 的，长度为 $dist$ 路径，在新图中的长度均为 $dist - h_t$。

因此，对于确定的 t，最小化 $dist - h_t$，相当于最小化了 $dist$，从而在新图上的最短路所经过的边集就是原图上的最短路所经过的边集。

初始时，h_i 显然可以设置为用 SPFA 算法得出的最短路。那么，我们只需要在完成一次增广，并加入所增广的路径的反向边后，得到另一组可行的 h_i 就可以了。

算法流程

令 $dist'_i$ 表示加入反向边前，新图上的最短路数组。

算法流程

令 $dist'_i$ 表示加入反向边前，新图上的最短路数组。
则**对于最短路上的每一条边** (u, v, w) ，均有

$$dist'_u + h_u + w - h_v = dist'_v$$

算法流程

令 $dist'_i$ 表示加入反向边前，新图上的最短路数组。
则**对于最短路上的每一条边** (u, v, w)，均有

$$dist'_u + h_u + w - h_v = dist'_v$$

我们希望得到新的 h'_i，满足对于边 (u, v, w)，有

$$h'_u + w - h'_v \geq 0$$

算法流程

令 $dist'_i$ 表示加入反向边前，新图上的最短路数组。
则**对于最短路上的每一条边** (u, v, w) ，均有

$$dist'_u + h_u + w - h_v = dist'_v$$

我们希望得到新的 h'_i ，满足对于边 (u, v, w) ，有

$$h'_u + w - h'_v \geq 0$$

并且，对于其反向边 $(v, u, -w)$ ，有

$$h'_v - w - h'_u \geq 0$$

算法流程

令 $dist'_i$ 表示加入反向边前，新图上的最短路数组。
则**对于最短路上的每一条边** (u, v, w) ，均有

$$dist'_u + h_u + w - h_v = dist'_v$$

我们希望得到新的 h'_i ，满足对于边 (u, v, w) ，有

$$h'_u + w - h'_v \geq 0$$

并且，对于其反向边 $(v, u, -w)$ ，有

$$h'_v - w - h'_u \geq 0$$

从而 $h'_u + w - h'_v = 0$

算法流程

又因为

$$(dist'_u + h_u) + w - (dist'_v + h_v) = 0$$

算法流程

又因为

$$(dist'_u + h_u) + w - (dist'_v + h_v) = 0$$

可以对所有点，令

$$h'_i = dist'_i + h_i$$

算法流程

又因为

$$(dist'_u + h_u) + w - (dist'_v + h_v) = 0$$

可以对所有点，令

$$h'_i = dist'_i + h_i$$

同时，**对于不在最短路上的边** (u, v, w) ，由于

$$dist'_u + h_u + w - h_v \geq dist'_v$$

算法流程

又因为

$$(dist'_u + h_u) + w - (dist'_v + h_v) = 0$$

可以对所有点，令

$$h'_i = dist'_i + h_i$$

同时，**对于不在最短路上的边** (u, v, w) ，由于

$$dist'_u + h_u + w - h_v \geq dist'_v$$

有

$$(dist'_u + h_u) + w - (dist'_v + h_v) \geq 0$$

算法流程

即对于图中任意的一条边 (u, v, w)，均有

$$h'_u + w - h'_v \geq 0$$

算法流程

即对于图中任意的一条边 (u, v, w)，均有

$$h'_u + w - h'_v \geq 0$$

最后，注意到 $dist'_i$ 即为原图中的最短路 $dist_i$ 减去 h_i，而 $h'_i = dist'_i + h_i$，恰好就是原图中的最短路 $dist_i$。因此 h_i 的取值范围也始终在原图中最短路的取值范围内，而不会超出可存储的范围。

运行效率

以下是朱震霆学长在 UOJ80 的提交。

ID	题目	提交者	结果	用时	内存	语言
#247887	#80. 二分图最大权匹配	whzzt	100	9815ms	6792kb	C++11

运行效率

以下是朱震霆学长在 UOJ80 的提交。

ID	题目	提交者	结果	用时	内存	语言
#247887	#80. 二分图最大权匹配	whzzt	100	9815ms	6792kb	C++11

学长本人在 ACM 区域赛时曾提到，在 UOJ 上采用费用流通过的代码少之又少，即使是这份代码也是经过了硬核的常数优化。

```
#include<bits/stdc++.h>
#pragma GCC optimize("Ofast,no-stack-protector,-funroll-loops")
#pragma GCC target("avx,sse4")
#define gc() getchar()
```

运行效率

以下是朱震霆学长在 UOJ80 的提交。

ID	题目	提交者	结果	用时	内存	语言
#247887	#80. 二分图最大权匹配	whzzt	100	9815ms	6792kb	C++11

学长本人在 ACM 区域赛时曾提到，在 UOJ 上采用费用流通过的代码少之又少，即使是这份代码也是经过了硬核的常数优化。

```
#include<bits/stdc++.h>
#pragma GCC optimize("Ofast,no-stack-protector,-funroll-loops")
#pragma GCC target("avx,sse4")
#define gc() getchar()
```

可见，虽然 Dijkstra 费用流的时间复杂度是 $O(N^3)$，但是，想要在 $1s$ 内通过 $N = 400$ 的数据范围还是相当勉强的。更不用说连复杂度都是 $O(N^4)$ 的 SPFA 费用流了。

算法简介

Kuhn-Munkres 算法，简称 KM 算法，是专门用作解决**二分图最大权匹配**，或**二分图最大权完美匹配**问题的图论算法。

算法简介

Kuhn-Munkres 算法，简称 KM 算法，是专门用作解决**二分图最大权匹配**，或**二分图最大权完美匹配**问题的图论算法。

令非负权**完全二分图**左侧点数为 N_L ，右侧点数为 N_R ，且 $N_L \leq N_R$ ，KM 算法求得的是该二分图的一个**完美匹配**，即左侧的所有点均在一条匹配边上的一组匹配，其时间复杂度为 $O(N_L^2 \times N_R)$ 。

算法简介

Kuhn-Munkres 算法，简称 KM 算法，是专门用作解决**二分图最大权匹配**，或**二分图最大权完美匹配**问题的图论算法。

令非负权**完全二分图**左侧点数为 N_L ，右侧点数为 N_R ，且 $N_L \leq N_R$ ，KM 算法求得的是该二分图的一个**完美匹配**，即左侧的所有点均在一条匹配边上的一组匹配，其时间复杂度为 $O(N_L^2 \times N_R)$ 。

注意到虽然我们要求图是完全二分图，实际应用中，对于非完全二分图，我们可以按照需要在没有边相连的点之间连上权为 0 ，或 $-\infty$ 的边，再将所有边权增加 ∞ ，从而适应 KM 算法的使用条件。

算法简介

Kuhn-Munkres 算法，简称 KM 算法，是专门用作解决**二分图最大权匹配**，或**二分图最大权完美匹配**问题的图论算法。

令非负权**完全二分图**左侧点数为 N_L ，右侧点数为 N_R ，且 $N_L \leq N_R$ ，KM 算法求得的是该二分图的一个**完美匹配**，即左侧的所有点均在一条匹配边上的一组匹配，其时间复杂度为 $O(N_L^2 \times N_R)$ 。

注意到虽然我们要求图是完全二分图，实际应用中，对于非完全二分图，我们可以按照需要在没有边相连的点之间连上权为 0 ，或 $-\infty$ 的边，再将所有边权增加 ∞ ，从而适应 KM 算法的使用条件。

在 KM 算法的求解过程中，我们还可以同时解决**二分图最小顶标和**问题，并得到一组可行解。

算法简介

Kuhn-Munkres 算法，简称 KM 算法，是专门用作解决**二分图最大权匹配**，或**二分图最大权完美匹配**问题的图论算法。

令非负权**完全二分图**左侧点数为 N_L ，右侧点数为 N_R ，且 $N_L \leq N_R$ ，KM 算法求得的是该二分图的一个**完美匹配**，即左侧的所有点均在一条匹配边上的一组匹配，其时间复杂度为 $O(N_L^2 \times N_R)$ 。

注意到虽然我们要求图是完全二分图，实际应用中，对于非完全二分图，我们可以按照需要在没有边相连的点之间连上权为 0 ，或 $-\infty$ 的边，再将所有边权增加 ∞ ，从而适应 KM 算法的使用条件。

在 KM 算法的求解过程中，我们还可以同时解决**二分图最小顶标和**问题，并得到一组可行解。

相比于用最大费用流解决二分图最大权匹配问题，KM 算法有着明显的常数和代码复杂度的优势。并且，对传统的 KM 算法稍加改进，我们可以让 KM 算法像费用流一样，求出每一个大小的匹配的最优解，从而在功能上完全包含费用流算法。

对偶原理

KM 算法是基于线性规划的对偶原理的。

对偶原理

KM 算法是基于线性规划的对偶原理的。
令二分图左侧点集为 L，右侧点集为 R，边集为 E。
则二分图最大权匹配问题可以描述为如下线性规划：

$$\begin{array}{l|ll} \mathit{Maximize} & \sum_{(u,v)\in E} w_{u,v}x_{u,v} & \\ \mathit{Satisfy} & \sum_{v\in R} x_{u,v} \le 1 & (u \in L) \\ & \sum_{u\in L} x_{u,v} \le 1 & (v \in R) \\ & x_{u,v} \in \{0,1\} & \end{array}$$

对偶原理

KM 算法是基于线性规划的对偶原理的。
令二分图左侧点集为 L，右侧点集为 R，边集为 E。
则二分图最大权匹配问题可以描述为如下线性规划：

$$\begin{array}{l|ll} Maximize & \sum_{(u,v)\in E} w_{u,v}x_{u,v} & \\ Satisfy & \sum_{v\in R} x_{u,v} \le 1 & (u \in L) \\ & \sum_{u\in L} x_{u,v} \le 1 & (v \in R) \\ & x_{u,v} \in \{0,1\} & \end{array}$$

由于该线性规划矩阵的特殊性，可以证明，一定存在一组整数最优解，从而可以将取值范围为整数的限制 $x_{u,v} \in \{0,1\}$ 改为 $x_{u,v} \ge 0$，此时，答案不会发生改变。

对偶原理

考虑该线性规划的对偶线性规划，即：

$$\begin{array}{l|l} \mathit{Minimize} & \sum_{u\in L} y_u + \sum_{v\in R} y_v \\ \mathit{Satisfy} & y_u + y_v \geq w_{u,v} \qquad ((u,v)\in E) \\ & y_u \geq 0 \qquad (u \in L) \\ & y_v \geq 0 \qquad (v \in R) \end{array}$$

对偶原理

考虑该线性规划的对偶线性规划，即：

$$\begin{array}{l|l} \mathit{Minimize} & \sum_{u\in L} y_u + \sum_{v\in R} y_v \\ \mathit{Satisfy} & y_u + y_v \geq w_{u,v} \qquad ((u,v)\in E) \\ & y_u \geq 0 \qquad (u \in L) \\ & y_v \geq 0 \qquad (v \in R) \end{array}$$

由线性规划的对偶原理，这两个问题的答案在数值上是相等的。

对偶原理

考虑该线性规划的对偶线性规划，即：

$$\begin{array}{l|l} \mathit{Minimize} & \sum_{u\in L} y_u + \sum_{v\in R} y_v \\ \mathit{Satisfy} & y_u + y_v \geq w_{u,v} \qquad ((u,v)\in E) \\ & y_u \geq 0 \qquad (u \in L) \\ & y_v \geq 0 \qquad (v \in R) \end{array}$$

由线性规划的对偶原理，这两个问题的答案在数值上是相等的。

考虑如下描述新产生的问题：为二分图上每一个节点设置一个非负的顶标，要求每条边两侧点的顶标之和不小于边权，并且最小化所有点的顶标和。

对偶原理

考虑该线性规划的对偶线性规划，即：

$$\begin{array}{l|l} \mathit{Minimize} & \sum_{u\in L} y_u + \sum_{v\in R} y_v \\ \mathit{Satisfy} & y_u + y_v \geq w_{u,v} \qquad ((u,v)\in E) \\ & y_u \geq 0 \qquad (u \in L) \\ & y_v \geq 0 \qquad (v \in R) \end{array}$$

由线性规划的对偶原理，这两个问题的答案在数值上是相等的。

考虑如下描述新产生的问题：为二分图上每一个节点设置一个非负的顶标，要求每条边两侧点的顶标之和不小于边权，并且最小化所有点的顶标和。

这个新问题被称为**二分图最小顶标和**问题。

算法思路

令二分图左侧点的顶标为 x_i，右侧点的顶标为 y_i。

算法思路

令二分图左侧点的顶标为 x_i，右侧点的顶标为 y_i。

我们称一组顶标是合法的，当且仅当其满足了每条边两侧点的顶标之和不小于边权。对于一组合法的顶标，若边 (u, v, w) 满足 $x_u + y_v = w$，则称该边为一条**等边**。

算法思路

令二分图左侧点的顶标为 x_i，右侧点的顶标为 y_i。

我们称一组顶标是合法的，当且仅当其满足了每条边两侧点的顶标之和不小于边权。对于一组合法的顶标，若边 (u, v, w) 满足 $x_u + y_v = w$，则称该边为一条**等边**。

KM 算法是基于对一组初始时合法的顶标进行调整的算法，由对偶线性规划的性质，**二分图中任意一组合法的匹配的权值和一定不大于任意一组合法的顶标和**，从而若能找到一组由等边构成的完美匹配，我们就找到了两个问题的一个公共解，因此也是两个问题各自的最优解。

算法思路

令二分图左侧点的顶标为 x_i，右侧点的顶标为 y_i。

我们称一组顶标是合法的，当且仅当其满足了每条边两侧点的顶标之和不小于边权。对于一组合法的顶标，若边 (u, v, w) 满足 $x_u + y_v = w$，则称该边为一条**等边**。

KM 算法是基于对一组初始时合法的顶标进行调整的算法，由对偶线性规划的性质，**二分图中任意一组合法的匹配的权值和一定不大于任意一组合法的顶标和**，从而若能找到一组由等边构成的完美匹配，我们就找到了两个问题的一个公共解，因此也是两个问题各自的最优解。

因此，我们希望找到一条由等边构成的增广路。

算法思路

考虑从二分图左侧尚未匹配的任意一点 s 开始进行搜索。

算法思路

考虑从二分图左侧尚未匹配的任意一点 s 开始进行搜索。

若点 s 所在的等边连通块内存在一条增广路，则可以增广，使得匹配大小增加 1 。否则，我们必须调整顶标，以产生新的等边。

算法思路

考虑从二分图左侧尚未匹配的任意一点 s 开始进行搜索。

若点 s 所在的等边连通块内存在一条增广路，则可以增广，使得匹配大小增加 1 。否则，我们必须调整顶标，以产生新的等边。

令 s 所在的等边连通块中的点集为 C ，记

$$\delta = Min\{x_i + y_j - w_{i,j}\} \quad (i \in L, i \in C, j \in R, j \notin C)$$

算法思路

考虑从二分图左侧尚未匹配的任意一点 s 开始进行搜索。

若点 s 所在的等边连通块内存在一条增广路，则可以增广，使得匹配大小增加 1 。否则，我们必须调整顶标，以产生新的等边。

令 s 所在的等边连通块中的点集为 C ，记

$$\delta = Min\{x_i + y_j - w_{i,j}\} \quad (i \in L, i \in C, j \in R, j \notin C)$$

令所有 C 中左侧的点的顶标减少 δ ， C 中右侧的点的顶标增加 δ ，即可在保证顶标合法的情况下至少增加一条等边。可以发现，随着等边连通块 C 的扩大，我们一定可以通过这样的方法找到一组由等边构成的完美匹配，因此 KM 算法是正确的。

算法流程

令初始时的顶标为 $x_i = Max\{w_{i,j}\}\ (j \in R), y_i = 0$ 。

算法流程

令初始时的顶标为 $x_i = Max\{w_{i,j}\}\ (j \in R), y_i = 0$ 。

每次从一个尚未匹配的 $s \in L$ 出发，进行增广，维护 $visx_i, visy_i$ 表示各点是否在当前的等边连通块内。

算法流程

令初始时的顶标为 $x_i = Max\{w_{i,j}\}\ (j \in R), y_i = 0$ 。

每次从一个尚未匹配的 $s \in L$ 出发，进行增广，维护 $visx_i, visy_i$ 表示各点是否在当前的等边连通块内。

对于 $visy_i = false$ 的右侧点，维护 $slack_i$ ，满足

$$slack_i = Min\{x_j + y_i - w_{j,i}\} \quad (j \in L, visx_j = true)$$

算法流程

令初始时的顶标为 $x_i = Max\{w_{i,j}\}\ (j \in R), y_i = 0$ 。

每次从一个尚未匹配的 $s \in L$ 出发，进行增广，维护 $visx_i, visy_i$ 表示各点是否在当前的等边连通块内。

对于 $visy_i = false$ 的右侧点，维护 $slack_i$ ，满足

$$slack_i = Min\{x_j + y_i - w_{j,i}\} \quad (j \in L, visx_j = true)$$

每次改变顶标时，找到

$$\delta = Min\{slack_i\} \quad (i \in R, visy_i = false)$$

算法流程

令初始时的顶标为 $x_i = Max\{w_{i,j}\}\ (j \in R), y_i = 0$ 。

每次从一个尚未匹配的 $s \in L$ 出发，进行增广，维护 $visx_i, visy_i$ 表示各点是否在当前的等边连通块内。

对于 $visy_i = false$ 的右侧点，维护 $slack_i$ ，满足

$$slack_i = Min\{x_j + y_i - w_{j,i}\} \quad (j \in L, visx_j = true)$$

每次改变顶标时，找到

$$\delta = Min\{slack_i\} \quad (i \in R, visy_i = false)$$

令左侧连通块内的点顶标 $-\delta$ ，右侧连通块内的点顶标 $+\delta$ ，右侧不在连通块内的点的 $slack_i$ 减少 δ 。

算法流程

改变顶标后，找到右侧 $slack$ 变为 0 的点，访问它们。

算法流程

改变顶标后，找到右侧 $slack$ 变为 0 的点，访问它们。

对于访问到的点 pos，若 pos 不存在匹配点，则找到了一条由等边构成的增广路；若 pos 存在匹配点 $match_{pos}$，访问 $match_{pos}$，并更新右侧不在连通块内的点的 $slack$。

算法流程

改变顶标后，找到右侧 $slack$ 变为 0 的点，访问它们。

对于访问到的点 pos，若 pos 不存在匹配点，则找到了一条由等边构成的增广路；若 pos 存在匹配点 $match_{pos}$，访问 $match_{pos}$，并更新右侧不在连通块内的点的 $slack$。

不难发现，单次增广的最坏时间复杂度为 $O(N_L \times N_R)$。

算法流程

改变顶标后，找到右侧 $slack$ 变为 0 的点，访问它们。

对于访问到的点 pos，若 pos 不存在匹配点，则找到了一条由等边构成的增广路；若 pos 存在匹配点 $match_{pos}$，访问 $match_{pos}$，并更新右侧不在连通块内的点的 $slack$。

不难发现，单次增广的最坏时间复杂度为 $O(N_L \times N_R)$。

因此 KM 算法的总时间复杂度为 $O(N_L^2 \times N_R)$。

算法拓展

在介绍 KM 算法时，我们提到，对传统的 KM 算法稍加改进，我们可以让 KM 算法像费用流一样，求出每一个大小的匹配的最优解。

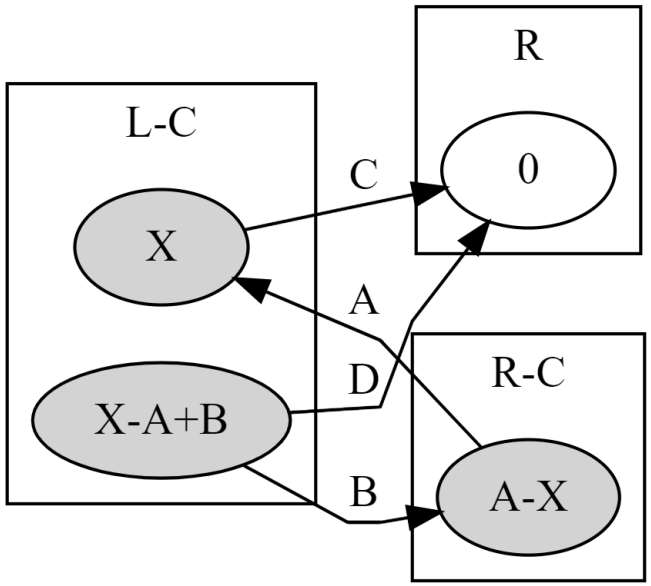
算法拓展

在介绍 KM 算法时，我们提到，对传统的 KM 算法稍加改进，我们可以让 KM 算法像费用流一样，求出每一个大小的匹配的最优解。

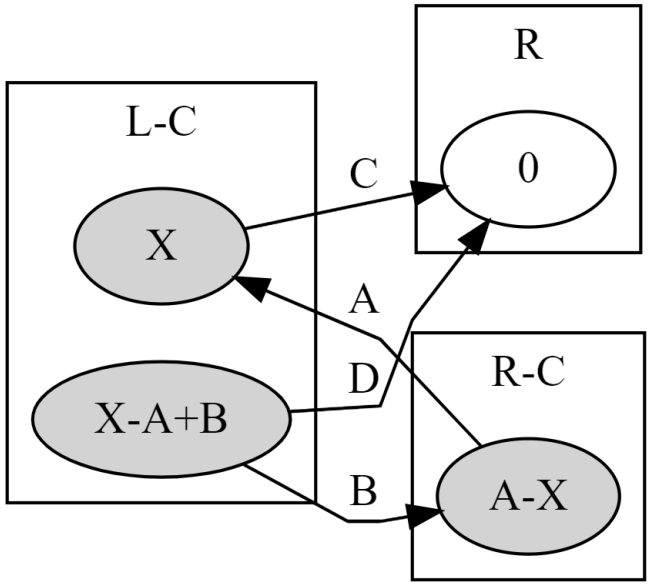
实际上，改进的方式十分简单，只需要统一将初始时左侧点的顶标设为 $+\infty$，并在每次增广时由从一个点开始增广改为从所有没有访问过的左侧点开始增广即可。

算法拓展

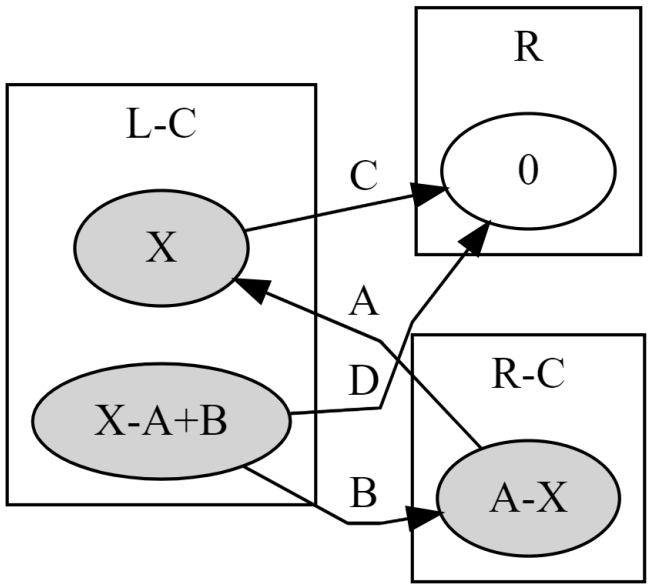
考虑这样做的正确性，如下图，左上方和右下方的两个点已经在第一次增广中匹配，考虑从左下方的点出发寻找增广路，首先访问了已经匹配的两个点，则各个点的顶标如图所示：



算法拓展



算法拓展



可以发现，此时，对于没有匹配的右上角的点来说，左侧两个点的顶标相差恰好是两条增广路的对称差，即边 A, B 的边权对增广路长度的贡献。再考虑计算 $slack$ 时减去的 $w_{i,j}$，也即图中的 C, D，可以发现，最小化 $slack$ 的过程就相当于找到一条最长的增广路。

算法拓展

因此，与费用流算法一样，在稍作修改后，KM 算法同样可以重复地找到最长的增广路，从而求出每一种边数的最大匹配。

算法拓展

因此，与费用流算法一样，在稍作修改后，KM 算法同样可以重复地找到最长的增广路，从而求出每一种边数的最大匹配。

由此，我们用 KM 算法在 $O(N^3)$ 的时间内解决了二分图最大权匹配问题，并且在功能上完全包含了费用流算法。

运行效率

以下是笔者在 UOJ80 的提交。

ID	题目	提交者	结果	用时	内存	语言
#382561	#80. 二分图最大权匹配	xuyixuan	100	1452ms	1992kb	C++11

运行效率

以下是笔者在 UOJ80 的提交。

ID	题目	提交者	结果	用时	内存	语言
#382561	#80. 二分图最大权匹配	xuyixuan	100	1452ms	1992kb	C++11

可见，在没有经过任何常数优化的情况下，KM 算法的运行效率也相当可观。事实上，由于我们很有可能在较早的时候就找到一条增广路，并且计算 $slack$ 数组的复杂度实际上与两边在 C 中的点数均有关，KM 算法的 $O(N^3)$ 是相当不满的。

ICPC Shanghai 2019 Blood Pressure Game

给定大小为 n 的数列 a 和大小为 m 的集合 b，求出把 b 中的元素中的 k 个添加到 a 中相邻的两个元素中间，或数组 a 的两侧，在每个位置（相邻 a 元素的之间或两侧）最多只能插入一个 b 中元素的条件下，使得相邻数之差的绝对值的和尽可能大。

ICPC Shanghai 2019 Blood Pressure Game

给定大小为 n 的数列 a 和大小为 m 的集合 b，求出把 b 中的元素中的 k 个添加到 a 中相邻的两个元素中间，或数组 a 的两侧，在每个位置（相邻 a 元素的之间或两侧）最多只能插入一个 b 中元素的条件下，使得相邻数之差的绝对值的和尽可能大。

求出 $k = 1, 2, 3, \ldots, m$ 时的最优解，并输出 $k = m$ 时的方案。

ICPC Shanghai 2019 Blood Pressure Game

给定大小为 n 的数列 a 和大小为 m 的集合 b，求出把 b 中的元素中的 k 个添加到 a 中相邻的两个元素中间，或数组 a 的两侧，在每个位置（相邻 a 元素的之间或两侧）最多只能插入一个 b 中元素的条件下，使得相邻数之差的绝对值的和尽可能大。

求出 $k = 1, 2, 3, \ldots, m$ 时的最优解，并输出 $k = m$ 时的方案。

$n, m \leq 600$

题目解法

出题人的 Intended Solution 是优化建图的费用流。

题目解法

出题人的 Intended Solution 是优化建图的费用流。

通过加入中介点，以及按照一定顺序加边，可以建出一张点数约为 $4n$，边数约为 $10n$ 的网络流图，再用费用流算法解决就可以了。

题目解法

出题人的 Intended Solution 是优化建图的费用流。

通过加入中介点，以及按照一定顺序加边，可以建出一张点数约为 $4n$，边数约为 $10n$ 的网络流图，再用费用流算法解决就可以了。

若采用 Dijkstra 费用流实现，时间复杂度为 $O(n^2 \log n)$。

题目解法

出题人的 Intended Solution 是优化建图的费用流。

通过加入中介点，以及按照一定顺序加边，可以建出一张点数约为 $4n$，边数约为 $10n$ 的网络流图，再用费用流算法解决就可以了。

若采用 Dijkstra 费用流实现，时间复杂度为 $O(n^2 \log n)$。

但实际上，这个解法的常数大得可怕，并且许多队伍当场建出的图可能包含更多的节点和边，加之优化建图过后的费用流想要输出方案的细节又很繁琐，采用 Intended Solution 的队伍都没有当场在时限内通过这道题。

题目解法

出题人的 Intended Solution 是优化建图的费用流。

通过加入中介点，以及按照一定顺序加边，可以建出一张点数约为 $4n$，边数约为 $10n$ 的网络流图，再用费用流算法解决就可以了。

若采用 Dijkstra 费用流实现，时间复杂度为 $O(n^2 \log n)$。

但实际上，这个解法的常数大得可怕，并且许多队伍当场建出的图可能包含更多的节点和边，加之优化建图过后的费用流想要输出方案的细节又很繁琐，采用 Intended Solution 的队伍都没有当场在时限内通过这道题。

而如果了解了本文中提及的，可以求得每一个流量的最优解的 KM 算法，那么本题便是对该算法的一个直接应用，得到的解法时间复杂度为 $O(n^3)$，常数很小，可以通过。

当场结果

这道题是这场区域赛的 M 题，排行榜如下：

RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Observers *太D了 杭州学军中学	10 1215	285 2 tries	42 1 try	261 1 try	18 1 try	83 1 try	120 2 tries		30 1 try	187 1 try	78 3 tries	11 2 tries		
2	*杭州二中1队 浙江省杭州第二中学	10 1222	102 2 tries	18 3 tries	279 1 try	73 1 try	95 1 try	133 1 try		29 2 tries	237 2 tries	110 3 tries	6 1 try	3 tries	
3	*谪妄 江苏省常州高级中学	10 1489	268 3 tries	20 1 try	274 5 tries	34 1 try	58 3 tries	125 1 try		28 1 try		113 4 tries	10 1 try		299 3 tries
4	*被D爆了 杭州学军中学	9 1062	208 1 try	16 1 try		29 1 try	56 3 tries	141 2 tries		69 1 try	285 4 tries	92 3 tries	6 1 try		5 tries
5	Participants 不朽之夜 上海交通大学	9 1167	220 4 tries	29 1 try		18 2 tries	67 1 try	153 3 tries		47 1 try	155 1 try	295 3 tries	23 1 try		
6	三演一鸽 清华大学	9 1175	272 1 try	40 2 tries	3 tries	86 1 try	56 2 tries	160 1 try		34 2 tries		129 1 try	11 2 tries		267 3 tries
7	德先生 北京大学	8 899		31 4 tries		78 1 try	66 1 try	121 1 try	1 try	35 1 try	250 2 tries	188 3 tries	10 1 try		
8	Calabash! 南京大学	8 932		21 1 try		32 1 try	73 3 tries	186 1 try	211 2 tries	64 1 try		188 5 tries	17 1 try		

当场结果

这道题是这场区域赛的 M 题，排行榜如下：

RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Observers *太D了 杭州学军中学	10 1215	285 2 tries	42 1 try	261 1 try	18 1 try	83 1 try	120 2 tries		30 1 try	187 1 try	78 3 tries	11 2 tries		
2	*杭州二中1队 浙江省杭州第二中学	10 1222	102 2 tries	18 3 tries	279 1 try	73 1 try	95 1 try	133 1 try		29 2 tries	237 2 tries	110 3 tries	6 1 try	3 tries	
3	*谵妄 江苏省常州高级中学	10 1489	268 3 tries	20 1 try	274 5 tries	34 1 try	58 3 tries	125 1 try		28 1 try		113 4 tries	10 1 try		299 3 tries
4	*被D爆了 杭州学军中学	9 1062	208 1 try	16 1 try		29 1 try	56 3 tries	141 2 tries		69 1 try	285 4 tries	92 3 tries	6 1 try		5 tries
5	Participants 不朽之夜 上海交通大学	9 1167	220 4 tries	29 1 try		18 2 tries	67 1 try	153 3 tries		47 1 try	155 1 try	295 3 tries	23 1 try		
6	三演一鸽 清华大学	9 1175	272 1 try	40 2 tries	3 tries	86 1 try	56 2 tries	160 1 try		34 2 tries		129 1 try	11 2 tries		267 3 tries
7	德先生 北京大学	8 899		31 4 tries		78 1 try	66 1 try	121 1 try	1 try	35 1 try	250 2 tries	188 3 tries	10 1 try		
8	Calabash! 南京大学	8 932		21 1 try		32 1 try	73 3 tries	186 1 try	211 2 tries	64 1 try		188 5 tries	17 1 try		

当场通过的两支队伍程序的时间复杂度均为 $O(n^3)$，其中清华大学“三演一鸽”队采用的便是本文所介绍的 KM 算法，而笔者所在的“谵妄”队采用的是动态规划。可见，即使在高水平的 ACM 队伍中，真正会运用 KM 算法的选手仍然不多。

Allocation

这是笔者今年出在 NOI 模拟赛中的一道题目。

Allocation

这是笔者今年出在 NOI 模拟赛中的一道题目。

给定 $N \times N$ 的非负权完全二分图，其中左侧第 i 个点与右侧第 j 个点之间的边边权为 $a_{i,j}$，你需要回答 Q 个询问，每次询问给定一个正整数 C，求出下列线性规划的结果：

$$\begin{array}{l|l} \mathit{Minimize} & \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i + z \\ \mathit{Satisfy} & x_i + y_i + \frac{z}{C} \geq a_{i,j} \ (1 \leq i, j \leq N) \\ & x_i, y_j, z \geq 0 \ (1 \leq i, j \leq N) \end{array}$$

Allocation

这是笔者今年出在 NOI 模拟赛中的一道题目。

给定 $N \times N$ 的非负权完全二分图，其中左侧第 i 个点与右侧第 j 个点之间的边边权为 $a_{i,j}$，你需要回答 Q 个询问，每次询问给定一个正整数 C，求出下列线性规划的结果：

$$\begin{array}{l|l} \mathit{Minimize} & \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i + z \\ \mathit{Satisfy} & x_i + y_i + \frac{z}{C} \geq a_{i,j} \ (1 \leq i, j \leq N) \\ & x_i, y_j, z \geq 0 \ (1 \leq i, j \leq N) \end{array}$$

$N \leq 500, Q \leq 5 \times 10^3, a_{i,j} \leq 10^9$

题目解法

令 $z' = \frac{z}{C}$，则以上线性规划可以写为：

$$\begin{array}{l|l} Minimize & \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i + z' \times C \\ Satisfy & x_i + y_i + z' \geq a_{i,j} \ (1 \leq i, j \leq N) \\ & x_i, y_j, z' \geq 0 \ (1 \leq i, j \leq N) \end{array}$$

题目解法

令 $z' = \frac{z}{C}$，则以上线性规划可以写为：

$$\begin{array}{l|l} Minimize & \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i + z' \times C \\ Satisfy & x_i + y_i + z' \geq a_{i,j}\ (1 \leq i, j \leq N) \\ & x_i, y_j, z' \geq 0\ (1 \leq i, j \leq N) \end{array}$$

考虑其对偶线性规划，即：

$$\begin{array}{l|l} Maximize & \sum_{i=1}^{N} \sum_{j=1}^{N} a_{i,j} x_{i,j} \\ Satisfy & \sum_{j=1}^{N} x_{i,j} \leq 1\ (1 \leq i \leq N) \\ & \sum_{i=1}^{N} x_{i,j} \leq 1\ (1 \leq j \leq N) \\ & \sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j} \leq C \\ & x_{i,j} \geq 0\ (1 \leq i, j \leq N) \end{array}$$

题目解法

令 $z' = \frac{z}{C}$，则以上线性规划可以写为：

$$\begin{array}{l|l} Minimize & \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i + z' \times C \\ Satisfy & x_i + y_i + z' \geq a_{i,j}\ (1 \leq i, j \leq N) \\ & x_i, y_j, z' \geq 0\ (1 \leq i, j \leq N) \end{array}$$

考虑其对偶线性规划，即：

$$\begin{array}{l|l} Maximize & \sum_{i=1}^{N} \sum_{j=1}^{N} a_{i,j} x_{i,j} \\ Satisfy & \sum_{j=1}^{N} x_{i,j} \leq 1\ (1 \leq i \leq N) \\ & \sum_{i=1}^{N} x_{i,j} \leq 1\ (1 \leq j \leq N) \\ & \sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j} \leq C \\ & x_{i,j} \geq 0\ (1 \leq i, j \leq N) \end{array}$$

可以发现，这恰好是二分图限制匹配总数为 C 的最大权匹配问题，采用 KM 算法解决即可，总时间复杂度为 $O(N^3 + Q)$。

当场结果

这道题是这场模拟赛的第 2 题，排行榜如下：

排名	名称	总分	anticipate	allocation	accompany
1	std	300	100	100	100
2	tester - sczsunnuozhou	277	100	77	100
3	FZ-ChenYu	254	77	77	100
4	cdqz-yyf=gg	182	5	77	100
5	cdqz-wyp	177	100	77	0
6	FZ-YBW	175	40	65	70
7	cdqz-zzh	154	22	77	55
8	sczsunruofan	150	77	3	70
9	sczchenghuarui	142	72	0	70
9	zsjz-dh	142	77	25	40
11	FZ-LvXianMeng	137	77	25	35
12	sczpcf	135	62	3	70

当场结果

这道题是这场模拟赛的第 2 题，排行榜如下：

排名	名称	总分	anticipate	allocation	accompany
1	std	300	100	100	100
2	tester - sczsunnuozhou	277	100	77	100
3	FZ-ChenYu	254	77	77	100
4	cdqz-yyf=gg	182	5	77	100
5	cdqz-wyp	177	100	77	0
6	FZ-YBW	175	40	65	70
7	cdqz-zzh	154	22	77	55
8	sczsunruofan	150	77	3	70
9	sczchenghuarui	142	72	0	70
9	zsjz-dh	142	77	25	40
11	FZ-LvXianMeng	137	77	25	35
12	sczpcf	135	62	3	70

当场转化得到最大权匹配问题，并且用费用流解决的选手都得到了 65 或 77 分。但却没有选手采用 KM 算法得到满分。

当场结果

这道题是这场模拟赛的第 2 题，排行榜如下：

排名	名称	总分	anticipate	allocation	accompany
1	std	300	100	100	100
2	tester - sczsunnuozhou	277	100	77	100
3	FZ-ChenYu	254	77	77	100
4	cdqz-yyf=gg	182	5	77	100
5	cdqz-wyp	177	100	77	0
6	FZ-YBW	175	40	65	70
7	cdqz-zzh	154	22	77	55
8	sczsunruofan	150	77	3	70
9	sczchenghuarui	142	72	0	70
9	zsjz-dh	142	77	25	40
11	FZ-LvXianMeng	137	77	25	35
12	sczpcf	135	62	3	70

当场转化得到最大权匹配问题，并且用费用流解决的选手都得到了 65 或 77 分。但却没有选手采用 KM 算法得到满分。

可见，在当今 OI 界，真正会运用 KM 算法的选手同样不多。

总结

本文研究了二分图最大权匹配问题的一些解法，并对它们的时间复杂度和实际运行效率进行了对比。

总结

本文研究了二分图最大权匹配问题的一些解法，并对它们的时间复杂度和实际运行效率进行了对比。

常见的 SPFA 费用流算法的时间复杂度是 $O(N^4)$ 的，并且，即使采用 Dijkstra 费用流的技巧将复杂度优化至 $O(N^3)$，其实际运行效率仍然不高。

总结

本文研究了二分图最大权匹配问题的一些解法，并对它们的时间复杂度和实际运行效率进行了对比。

常见的 SPFA 费用流算法的时间复杂度是 $O(N^4)$ 的，并且，即使采用 Dijkstra 费用流的技巧将复杂度优化至 $O(N^3)$，其实际运行效率仍然不高。

相比之下，KM 算法在同样具有 $O(N^3)$ 的时间复杂度的情况下，有着明显简单的代码复杂度，以及显著优异的实际运行速度。并且，在对传统的 KM 算法进行修改后，KM 算法也可以像费用流算法一样，重复地找到每一条最长增广路，从而在功能上完全代替费用流算法。

总结

本文研究了二分图最大权匹配问题的一些解法，并对它们的时间复杂度和实际运行效率进行了对比。

常见的 SPFA 费用流算法的时间复杂度是 $O(N^4)$ 的，并且，即使采用 Dijkstra 费用流的技巧将复杂度优化至 $O(N^3)$，其实际运行效率仍然不高。

相比之下，KM 算法在同样具有 $O(N^3)$ 的时间复杂度的情况下，有着明显简单的代码复杂度，以及显著优异的实际运行速度。并且，在对传统的 KM 算法进行修改后，KM 算法也可以像费用流算法一样，重复地找到每一条最长增广路，从而在功能上完全代替费用流算法。

在 ACM 竞赛中，不能求出每一条最长增广路的 KM 算法已经相当的普及，但是在 OI 界，尚有很多选手，乃至很多高水平选手对 KM 算法不够了解，希望笔者的本次讨论能够教会更多原本并不了解 KM 算法的选手，进一步在 OI 界普及 KM 算法。

感谢

感谢 CCF 提供学习和交流的平台。

感谢

感谢 CCF 提供学习和交流的平台。
感谢杨天祺、朱震霆学长对本文的启发和帮助。

感谢

感谢 CCF 提供学习和交流的平台。
感谢杨天祺、朱震霆学长对本文的启发和帮助。
感谢集训队教练高闻远对我的帮助与指导。

感谢

感谢 CCF 提供学习和交流的平台。
感谢杨天祺、朱震霆学长对本文的启发和帮助。
感谢集训队教练高闻远对我的帮助与指导。
感谢大家的聆听。

关于课件的发放

本次讲课的课件，会连同所讲的第二道例题和数据和标程一起发布在 UOJ 和 LOJ 用户群中，以供大家在课后进行学习。