# 二分图

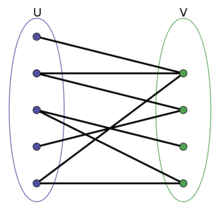
## 概念

二分图是一类特殊性质的图

来自百度百科的概念：

[二分图](https://baike.baidu.com/item/%E4%BA%8C%E5%88%86%E5%9B%BE)又称作二部图，是图论中的一种[特殊模型](https://baike.baidu.com/item/%E7%89%B9%E6%AE%8A%E6%A8%A1%E5%9E%8B)。 设G=(V,E)是一个无向图，如果顶点V可分割为两个互不相交的子集(A,B)，并且图中的每条边（i，j）所关联的两个顶点i和j分别属于这两个不同的顶点集(i in A,j in B)，则称图G为一个二分图。

二分图的边一般都是有无向的。但是在关于二分图算法中可以当成有向的用



## 判定

给定一个图，用染色法，即从其中一个顶点开始，将跟它相邻的点染成与其不同的颜色，如果邻接的点以及被染成了相同颜色的，则说明不是二分图，用dfs遍历即可。

这里有一些问题和细节：

* 给出的图可能有多个联通分量，因此dfs时候应该从每个点为起点，分别作dfs,把color数组作为有没有访问过得标记。
* 图的总点数是1不是二分图，点数必须大于二，但是在dfs过程中，遇到孤立点应该返回 true是二分图 ，因为大于1的孤立点组成的图是二分图
* 不需要考虑重边的情况，多个边也不影响结果，且不影响dfs代码
* 不需要考虑自环的情况，有自环也是二分图，且不影响dfs代码

题目：hihocoder 1121

## 存储：

* 正常的链式前向星，邻接表。必要时用数组记录点的所属。
* 改造链式前向星，把边集数组的前驱head开成2个，分别记录由集合1出发的边和

集合2出发的边。

* 改造邻接表，开两个vector，分别记录由集合1出发的边和集合2出发的边。

## 最大匹配和完美匹配：

**二分图匹配：**是指寻找二分图子图，满足子图任意两条边没有公共顶点。

所有满足这个条件的子图都叫做二分图匹配图。

**二分图最大匹配：**所有匹配中，边数最多的那个图叫最大匹配

**二分图完美匹配：**所有匹配中，匹配图点数等于原本二分图点数(所有顶点都是匹配点)，显然完美匹配一定是最大匹配，但是反过来不一定，而且所有二分图都存在最大匹配，但是不一定存在完美匹配。

求最大匹配数量有以下几种策略

**二分图最大匹配数：**最大匹配数指的不是有多少种匹配，而是最大匹配图的边数。因为二分图匹配指的就是匹配边

### ****转换成网络流：****

二分图有两组点，分别建立两个点S和T，他们各连接一组点，都用权值为1的边连接，原本边的权值是INF(无穷大)，并且在这里要求每条边都是严格单向由左边集合到右边集合的，运行一次网络最大流。算出的最大流就是最大匹配数.时间复杂度等同于最大流算法的时间复杂度。

最大流算法参见另一片文章



#### ****编码细节：****

* 无论用哪个最大流算法，直接带就好，推荐用链式前向星实现
* 注意输入的二分图要转换成网流落图，网络流图在链式前向星下，偶数边存正边，奇数边存放虚拟边(权值是0的边)
* 注意数据范围，网流落图边数比二分图多，尤其是洛谷那道题好几百万

### ****匈牙利算法：****

这篇博客讲的易懂<https://blog.csdn.net/cillyb/article/details/55511666>

思路引用这个博客：说穿了，就是你从二分图中找出一条路径来，让路径的起点和终点都是还没有匹配过的点，并且路径经过的连线是一条没被匹配、一条已经匹配过，再下一条又没匹配这样交替地出现。找到这样的路径后，显然路径里没被匹配的连线比已经匹配了的连线多一条，于是修改匹配图，把路径里所有匹配过的连线去掉匹配关系，把没有匹配的连线变成匹配的，这样匹配数就比原来多1个。不断执行上述操作，直到找不到这样的路径为止。

匈牙利算法时间复杂度是V\*E(邻接表/链式前向星)，邻接矩阵在稀疏图较慢，匈牙利算法求最大匹配数,是一个不断改变匹配点而修改匹配图的过程，。

为了方便说，点集合分为x集和y集

#### dfs实现:

用vis数组记录y集的某个点有木有访问过(dfs里每次循环遍历的也是y集)，用sym[i]记录和当前y集的i相连的点 在x集的编号。sym数组是匈牙利算法比较重要的结构，它相当于存储了当前匹配图。

dfs要以x集每个点为起点，各进行一次，对于每次都执行这样的操作：

每次递归当前点所有能到的点，但是看思路说明，每次都是从x集出发的，因此就用了sym数组找某个y集节点的前驱(或者叫父节点也可以)，每次dfs去搜索next的前驱，因为next必定在y集上，now以及sym[next]必定在x集上

以x集每个点为起点，各进行一次dfs原因在于实现：这不断改变匹配点而修改匹配图的过程，也就是每个dfs，如果返回true说明修改成功，匹配数变多。

#### bfs实现:

类似的思路,用dfs的知识即可。

#### 编码细节:

* 不论dfs还是bfs,按理说要以x集所有点为起点，各进行搜索，要每次搜索前把vis标记数组置0，但实际可以用个巧妙方法优化去掉这个操作。可以把vis设置为int数组，初始是0,每次搜索标记方式是把vis[now]+1,而不是直接等于1，然后设置一个time表示次数，根据vis和time的值来判断有没有访问过。
* 如果要找到具体的最大匹配图边集，通过遍历sym数组就可以了
* 二分图匹配的hungary算法的图应该是单向图。从X集合到Y集合，一般点数不多，可用邻接矩阵存贮，比较方便

## ****Hopcroft-Karp算法:****

在匈牙利算法上进行优化，时间复杂度达到E\*V^0.5

## 带权二分图的最佳匹配：

带权二分图就是普通二分图基础上,每条边给个权值

如果G为加权二分图,则权值和最大的完美匹配称为最佳匹配.,最佳匹配存在需要满足条件,首先要是带权值二分图，其次这个图G要存在完美匹配。

### 转换成最小费用最大流：

二分图和网络流是有联系的，说过普通二分图最大匹配可以转换成最大流，那么带权二分图可以转换最小费用最大流的模型，方法类似，也是增加起点终点，构建的网络流图边的费用cost就是二分图的边权，加进去的与起点终点相连的边cost是0，构建的网络流图边的流量val是INF,但与起点终点两连的边流量是1，就像不带权二分图的网络流解法那样，实际是求最大费用最大流。

构建好网络流图跑一边费用流算法，但是费用流时间复杂度比较高。所以这么做有些地方超时，还是用KM算法比较好

### KM算法：

是最大流的朴素dfs增光算法和匈牙利算法的结合，时间复杂度V^3,比费用流低，但写不好容易写成V\*4。这里没有算进边因为一般都是满图，边数是V\*(V-1)/2

KM算法求出的是最佳匹配，最佳匹配是完美匹配，如果原本图不存在完美匹配，要给图添加权值是0的边。是二分图满边，这样就存在完美匹配了

#### 步骤:

1. 首先选择定点较少的集当做x集，初始时对x集的每一个顶点设置顶标lx，顶标的值为该点关联的最大边的权值，y集的顶点顶标ly为0。
2. 对于x集中的每个顶点，在相等子图中利用匈牙利算法找一条增广路径，如果没有找到，则修改顶标，扩大相等子图，继续找增广路径。当每个点都找到增广路径时(能进行KM算法的图,都存在完美匹配)，此时意味着每个点都在匹配中，即找到了二分图的完备匹配。该完备匹配即为二分图的最佳匹配。

什么是相等子图呢？因为每个顶点有一个顶标，如果我们选择边权等于两端点的顶标之和的边，它们组成的图称为相等子图。

* KM算法和匈牙利算法类似的地方是，也要按照x集每个点为起点搜索
* KM算法是在匈牙利基础上进行的，每次调用改进的匈牙利算法的dfs(或bfs),用两个数组visx和visy标记x集和y集，分别在dfs中对它们标记，作用是是在dfs结束后，去改变走过的点标顶值,x集减少，y集增加。
* KM算法的搜索和匈牙利算法不同的是，KM算法只按照lx+ly==edge.val的边进行搜索，所以很可能找不到一条路径，因此对于每个点为起点的搜索都要反复进行，直到搜到为止
* dfs过程要找最小的差值lx+ly-edge.val，这个值用来更新lx和ly,最小差值反而使得完全匹配边权和最大
* 怎么输出最大总边权值，开两个数组xtoy和ytox，在dfs时更新的，最后就反应了完全匹配图的点映射关系，用这个在邻接矩阵中找边，求出总边权和就好。

## 二分图的性质和名词解释：

最大匹配：寻找二分图子图，满足子图任意两条边没有公共顶点，最大匹配是边数最多的匹配。最大匹配数就是匹配的边数

最小点覆盖：选取最少的点，使任意一条边至少有一个端点被选择，最小点覆盖数就是选取的点数

最小边覆盖：选取最少的边，使之覆盖了图中所有顶点，且任何一个顶点有且只有一条边与之关联，最小边覆盖数就是覆盖的边数。

最大独立子集：一个最大的点的集合，该集合内的任意两点没有边相连，最大独立子集数就是集合的点数。

最小路径覆盖：最小路径覆盖和最小边覆盖不同，不要求给的图是二分图，而是要求是N x N的有向图，不能有环，然后根据原图构造二分图，构造方法是将点一分为二，如，i分为i1和i2然后如果i和j有边，那么就在i1和j2之间连一条边。由此构成二分图。然后最小路径覆盖 = n-m，n为原图的点的个数，m为新造二分图的最大匹配

最大团: 一个最大的点的集合，该集合内的任意两点都有边相连，最大团数是集合的点数。

* 最大匹配数 = 最小点覆盖数（这是 Konig 定理）
* 最大独立集S 与 最小点覆盖集T 互补
* 最小边覆盖数 =最大独立子集数=顶点数 - 最大匹配数
* 最大独立子集=二分图补图的 最大团