二分图匹配算法总结

(author:Junhang huang)

二分图最大匹配的匈牙利算法：

二分图是这样一个图，它的顶点可以分类两个集合 X 和 Y，所有的边关联在两个顶点中，恰好一个属于集合 X，另一个属于集合 Y 。

最大匹配： 图中包含边数最多的匹配称为图的最大匹配。

完美匹配： 如果所有点都在匹配边上，称这个最大匹配是完美匹配。

最小覆盖： 最小覆盖要求用最少的点（X 集合或 Y 集合的都行）让每条边都至少和其中一个点关联。可以证明：最少的点（即覆盖数）＝最大匹配数

最小路径覆盖：用尽量少的不相交简单路径覆盖有向无环图 G 的所有结点。解决此类问题可以建立一个二分图模型。把所有顶点 i 拆成两个：X 结点集中的 i 和 Y 结点集中的 i',如果有边 i->j，则在二分图中引入边 i->j'，设二分图最大匹配为 m,则结果就是 n-m。

最大独立集问题： 在 N 个点的图 G 中选出 m 个点，使这 m 个点两两之间没有边．求 m 最大值．

如果图 G 满足二分图条件，则可以用二分图匹配来做．最大独立集点数 = N - 最大匹配数

二分图最大匹配问题的匈牙利算法

算法的思路是不停的找增广轨,并增加匹配的个数,增广轨顾名思义是指一条可以使匹配数变多的路径,在匹配问题中,增广轨的表现形式是一条"交错轨",也就是说这条由图的边组成的路径,它的第一条边是目前还没有参与匹配的,第二条边参与了匹配,第三条边没有..最后一条边没有参与匹配,并且始点和终点还没有被选择过.这样交错进行,显然他有奇数条边.那么对于这样一条路径,我们可以将第一条边改为已匹配,第二条边改为未匹配...以此类推.也就是将所有的边进行"反色",容易发现这样修改以后,匹配仍然是合法的,但是匹配数增加了一对.另外,单独的一条连接两个未匹配点的边显然也是交错轨.可以证明,当不能再找到增广轨时,就得到了一个最大匹配.这也就是匈牙利算法的思路。

一、二分图最大匹配

二分图最大匹配的经典匈牙利算法是由 Edmonds 在 1965 年提出的，算法的核心就是根据一个初始匹配不停的找增广路，直到没有增广路为止。

匈牙利算法的本质实际上和基于增广路特性的最大流算法还是相似的，只需要注意两点：

（一）每个 X 节点都最多做一次增广路的起点；

（二）如果一个 Y 节点已经匹配了，那么增广路到这儿的时候唯一的路径是走到 Y 节点的匹配点（可以回忆最大流算法中的后向边，这个时候后向边是可以增流的）。

找增广路的时候既可以采用 dfs 也可以采用 bfs，两者都可以保证 O(nm)的复杂度，因为每找一条增广路的复杂度是 O(m)，而最多增广 n 次，dfs 在实际实现中更加简短。

核心代码：

```
void init()
{
    int a,b,c,i,j;
    memset(link,0xff,sizeof(link));                    //初始化为-1
    memset(mat,0,sizeof(mat));
    memset(mk,0,sizeof(mk));
    for(i=0;i<n;i++)
    {
        scanf("%d: (%d)",&a,&b);

        for(j=0;j<b;j++)
        {
            scanf("%d",&c);
            mat[i][c]=true;
        }
    }
}

bool find(int r)                    // 返回是否找到增广路(递归,时间复杂度 O(n*n))
{
    int j;
    for(j=0;j<n;j++)
    {
        if(!mk[j]&&mat[r][j])
        {
```

```
                mk[j]=1;
                if(link[j]==-1||find(link[j]))
                {
                    link[j]=r;                                  //保存当前匹配节点
                    return true;
                }
            }
        }
        return false;
}
```

二、Hopcroft-Karp 算法

SRbGa 很早就介绍过这个算法，它可以做到 O(sqrt(n)*e)的时间复杂度，并且在实际使用中效果不错而且算法本身并不复杂。

Hopcroft-Karp 算法是 Hopcroft 和 Karp 在 1972 年提出的，该算法的主要思想是在每次增广的时候不是找一条增广路而是同时找几条不相交的最短增广路，形成极大增广路集，随后可以沿着这几条增广路同时进行增广。

可以证明在寻找增广路集的每一个阶段所寻找到的最短增广路都具有相等的长度，并且随着算法的进行最短增广路的长度是越来越长的，更进一步的分析可以证明最多只需要增广 ceil(sqrt(n))次就可以得到最大匹配（证明在这里略去）。

因此现在的主要难度就是在 O(e)的时间复杂度内找到极大最短增广路集，思路并不复杂，首先从所有 X 的未盖点进行 BFS，BFS 之后对每个 X 节点和 Y 节点维护距离标号，如果 Y 节点是未盖点那么就找到了一条最短增广路，BFS 完之后就找到了最短增广路集，随后可以直接用 DFS 对所有允许弧(dist[y]=dist[x]+1，可以参见高流推进 HLPP 的实现)进行类似于匈牙利中寻找增广路的操作，这样就可以做到 O(m)的复杂度。

实现起来也并不复杂，对于两边各 50000 个点，200000 条边的二分图最大匹配可以在 1s 内出解，效果很好：）

KM 算法是通过给每个顶点一个标号（叫做顶标）来把求最大权匹配的问题转化为求完备匹配的问题的。设顶点 Xi 的顶标为 A[i]，顶点 Yi 的顶标为 B [i]，顶点 Xi 与 Yj 之间的边权为 w[i,j]。在算法执行过程中的任一时刻，对于任一条边(i,j)，A[i]+B[j]>=w[i,j]始终 成立。KM 算法的正确性基于以下定理：

若由二分图中所有满足 A[i]+B[j]=w[i,j]的边(i,j)构成的子图（称做相等子图）有完备匹配，那么这个完备匹配就是二分图的最大权匹配。

这个定理是显然的。因为对于二分图的任意一个匹配，如果它包含于相等子图，那么它的边权和等于所有顶点的顶标和；如果它有的边不包含于相等子图，那么它的边权和小于所

有顶点的顶标和。所以相等子图的完备匹配一定是二分图的最大权匹配。

初始时为了使 A[i]+B[j]>=w[i,j]恒成立，令 A[i]为所有与顶点 Xi 关联的边的最大权，B[j]=0。如果当前的相等子图没有完备匹配，就按下面的方法修改顶标以使扩大相等子图，直到相等子图具有完备匹配为止。

我们求当前相等子图的完备匹配失败了，是因为对于某个 X 顶点，我们找不到一条从它出发的交错路。这时我们获得了一棵交错树，它的叶子结点全部是 X 顶点。现在我们把交错树中 X 顶点的顶标全都减小某个值 d，Y 顶点的顶标全都增加同一个值 d，那么我们会发现：

两端都在交错树中的边(i,j)，A[i]+B[j]的值没有变化。也就是说，它原来属于相等子图，现在仍属于相等子图。

两端都不在交错树中的边(i,j)，A[i]和 B[j]都没有变化。也就是说，它原来属于（或不属于）相等子图，现在仍属于（或不属于）相等子图。

X 端不在交错树中，Y 端在交错树中的边(i,j)，它的 A[i]+B[j]的值有所增大。它原来不属于相等子图，现在仍不属于相等子图。

X 端在交错树中，Y 端不在交错树中的边(i,j)，它的 A[i]+B[j]的值有所减小。也就说，它原来不属于相等子图，现在可能进入了相等子图，因而使相等子图得到了扩大。

现在的问题就是求 d 值了。为了使 A[i]+B[j]>=w[i,j]始终成立，且至少有一条边进入相等子图，d 应该等于 min{A[i]+B[j]-w[i,j]|Xi 在交错树中，Yi 不在交错树中}。

以上就是 KM 算法的基本思路。但是朴素的实现方法，时间复杂度为 O(n4)——需要找 O(n)次增广路，每次增广最多需要修改 O(n)次顶 标，每次修改顶标时由于要枚举边来求 d 值，复杂度为 O(n2)。实际上 KM 算法的复杂度是可以做到 O(n3)的。我们给每个 Y 顶点一个“松弛量”函数 slack，每次开始找增广路时初始化为无穷大。在寻找增广路的过程中，检查边(i,j)时，如果它不在相等子图中，则让 slack[j]变成原值与 A [i]+B[j]-w[i,j]的较小值。这样，在修改顶标时，取所有不在交错树中的 Y 顶点的 slack 值中的最小值作为 d 值即可。但还要注意一点：修改 顶标后，要把所有的 slack 值都减去 d。

//核心代码：

```
bool path(int u)                                        //寻找增广路
{
    int v;
    sx[u] = true;
    for (v = 0; v < M; v++)
    {
        if ((!sy[v]) && lx[u] + ly[v] == weight[u][v])
        {
            sy[v] = true;
            if (match[v] == -1 || path(match[v]))
            {
                match[v] = u;
                return true;
            }
        }
    }
    return false;
```

```
}


int KM_BestMatch(bool signal)
{
    int i, j, Maxcnt, k, ans = 0, dx;
    if (!signal)                          //求最小权值对边权，进行取反操作
    {
        for (i = 0; i < N; i++)
        {
            for (j = 0; j < M; j++)
            {
                weight[i][j] = -weight[i][j];
            }
        }
    }
    for (i = 0; i < M; i++)
    {
        ly[i] = 0;                        //初始化 y 的标号为 0
    }
    for (i = 0; i < N; i++)               //初始化 x 的标号为和 x 相连的边的最大权值
    {
        lx[i] = -NIL;
        for (j = 0; j < M; j++)
        {
            if (lx[i] < weight[i][j])     //获取最大边权值
            {
                lx[i] = weight[i][j];
            }
        }
    }

    memset(match, -1, sizeof(match));     //初始化匹配节点标记
    for (k = 0; k < N; k++)
    {
        while (1)
        {
            memset(sx, 0, sizeof(sx));    //标记 x 的节点是否被访问过
            memset(sy, 0, sizeof(sy));    //标记 y 的节点是否被访问过
            if (path(k))                  //如果找到增广路进入下个节点
            {
                break;
            }
            int dx = NIL;
```

```
            for (i = 0; i < N; i++)
            {
                if (sx[i])                          //如果 sx[i]输入子图
                {
                    for (j = 0; j < M; j++)
                    {
                        if (!sy[j] && (dx > lx[i] + ly[j] - weight[i][j]))     //sy[i]不属于子图
                        {
                            dx = lx[i] + ly[j] - weight[i][j];                  //去到值 dx
                        }
                    }
                }
            }
        //定理：如果 sx[i]输入相等子图，sy[i]不属于相等子图，如果 lx[i] + ly[j]的值减小
        //说明他们原来不属于子图现在进入子图了。
            for (i = 0; i < N; i++)
            {
                if (sx[i])
                {
                    lx[i] -= dx;                                                //lx[i] - dx
                }
            }
            for (i = 0; i < M; i++)
            {
                if (sy[i])
                {
                    ly[i] += dx;                                                //ly[i] + dx
                }
            }
        }
    }
    ans = 0;
    for (i = 0; i < M; i++)
    {
        if (match[i] != -1)                         //一直 wa 的地方遍历到边界的时候要小心
        ans += weight[match[i]][i];
    }
    if (!signal)
    {
        ans = -ans;
    }
```

三、二分图最优匹配

二分图最优匹配的经典算法是由 Kuhn 和 Munkres 独立提出的 KM 算法，值得一提的是最初的 KM 算法是在 1955 年和 1957 年提出的，因此当时的 KM 算法是以矩阵为基础的，随着匈牙利算法被 Edmonds 提出之后，现有的 KM 算法利用匈牙利树可以得到更漂亮的实现。

KM 算法中的基本概念是可行顶标(feasible vertex labeling)，它是节点的实函数并且对于任意弧(x,y)满足 l(x)+l(y)≥w(x,y)，此外一个概念是相等子图，它是 G 的一个生成子图，但是只包含满足 l(xi)+l(yj)=w(xi,yj)的所有弧(xi,yj)。

有定理：如果相等子图有完美匹配，那么该匹配是最大权匹配，证明非常直观也非常简单，反设其他匹配是最优匹配，它的权必然比相等子图的完美匹配的权要小。

KM 算法主要就是控制了怎样修改可行顶标的策略使得最终可以达到一个完美匹配，首先任意设置可行顶标（如每个 X 节点的可行顶标设为它出发的所有弧的最大权，Y 节点的可行顶标设为 0），然后在相等子图中寻找增广路，找到增广路就沿着增广路增广。

而如果没有找到增广路呢，那么就考虑所有现在在匈牙利树中的 X 节点（记为 S 集合），所有现在在匈牙利树中的 Y 节点（记为 T 集合），考察所有一段在 S 集合，一段在 not T 集合中的弧，取

delta = min {l(xi)+l(yj)-w(xi,yj),xi ∈ S, yj ∈ not T}

明显的，当我们把所有 S 集合中的 l(xi)减少 delta 之后，一定会有至少一条属于(S,not T)的边进入相等子图，进而可以继续扩展匈牙利树，为了保证原来属于(S,T)的边不退出相等子图，把所有在 T 集合中的点的可行顶标增加 delta。

随后匈牙利树继续扩展，如果新加入匈牙利树的 Y 节点是未盖点，那么找到增广路，否则把该节点的对应的 X 匹配点加入匈牙利树继续尝试增广。

复杂度分析：由于在不扩大匹配的情况下每次匈牙利树做如上调整之后至少增加一个元素，因此最多执行 n 次就可以找到一条增广路，最多需要找 n 条增广路，故最多执行 n^2 次修改顶标的操作，而每次修改顶标需要扫描所有弧，这样修改顶标的复杂度就是 O(n^2)的，总的复杂度是 O(n^4)的。

事实上我现在看到的几个版本的实现都是这样实现的，但是实际效果还不错，因为这个界通常很难达到。

对于 not T 的每个元素 yj，定义松弛变量 slack(yj) = min{l(xi)+l(yj)-w(xi,yj),xi ∈ S}，很明显的每次的 delta=min{slack(yj),yj∈ not T}，每次增广之后用 O(n^2)的时间计算所有点

的初始 slack，由于生长匈牙利树的时候每条弧的顶标增量相同，因此修改每个 slack 需要常数时间（注意在修改顶标后和把已盖 Y 节点对应的 X 节点加入匈牙利树的时候是需要修改 slack 的）。这样修改所有 slack 值时间是 O(n)的，每次增广后最多修改 n 次顶标，那么修改顶标的总时间降为 O(n^2)，n 次增广的总时间复杂度降为 O(n^3)。事实上我这样实现之后对于大部分的数据可以比 O(n^4)的算法快一倍左右。

四、二分图的相关性质

本部分内容主要来自于 SRbGa 的黑书，因为比较简单，仅作提示性叙述。

(1) 二分图的最大匹配数等于最小覆盖数，即求最少的点使得每条边都至少和其中的一个点相关联，很显然直接取最大匹配的一段节点即可。

(2) 二分图的独立数等于顶点数减去最大匹配数，很显然的把最大匹配两端的点都从顶点集中去掉这个时候剩余的点是独立集，这是|V|-2*|M|，同时必然可以从每条匹配边的两端取一个点加入独立集并且保持其独立集性质。

(3) DAG 的最小路径覆盖，将每个点拆点后作最大匹配，结果为 n-m，求具体路径的时候顺着匹配边走就可以，匹配边 i→j',j→k',k→l'....构成一条有向路径。

【最优完备匹配】

对于二分图的每条边都有一个权（非负），要求一种完备匹配方案，使得所有匹配边的权和最大，记做最优完备匹配。（特殊的，当所有边的权为 1 时，就是最大完备匹配问题）

KM 算法：（全称是 Kuhn-Munkras，是这两个人在 1957 年提出的，有趣的是，匈牙利算法是在 1965 年提出的）

为每个点设立一个顶标 Li，先不要去管它的意义。设 vi,j 为(i,j)边的权，如果可以求得一个完备匹配，使得每条匹配边 vi,j=Li+Lj，其余边 vi,j≤Li+Lj。此时的解就是最优的，因为匹配边的权和=ΣLi，其余任意解的权和都不可能比这个大。

定理：二分图中所有 vi,j=Li+Lj 的边构成一个子图 G，用匈牙利算法求 G 中的最大匹配，如果该匹配是完备匹配，则是最优完备匹配。

问题是，现在连 Li 的意义还不清楚。其实，我们现在要求的就是 L 的值，使得在该 L 值下达到最优完备匹配。

L 初始化：

Li=max{wi,j}(i∈x,j∈y)

Lj=0

建立子图 G，用匈牙利算法求 G 的最大匹配，如果在某点 i (i∈x)找不到增广轨，则得不到完备匹配。

此时需要对 L 做一些调整：设 S 为寻找从 i 出发的增广轨时访问的 x 中的点的集合，T 为访问的 y 中的点的集合。找到一个改进量 dx，dx=min{Li+Lj-wi,j}(i∈S,j 不∈T)

Li=Li-dx (i∈S)

Li=Li+dx (i∈T)

重复以上过程，不断的调整 L，直到求出完备匹配为止。

从调整过程中可以看出：每次调整后新子图中在包含原子图中所有的边的基础上添加了一些新边。每次调整后∑Li 会减少 dx，由于每次 dx 取最小，所以保证了解的最优性。

复杂度分析：

设 n 为点数，m 为边数，从每个点出发寻找增广轨的复杂度是 O(m)，如果找不到增广轨，对 L 做调整的复杂度也是 O(m)，而一次调整或者找到一条增广轨，或者将两个连通分量合成一个，而这两种情况最多都只进行 O(n)次，所以总的复杂度是 O(nm)。

扩展：根据 KM 算法的实质，可以求出使得所有匹配边的权和最小的匹配方案。

L 初始化：

Li=min{wi,j}(i∈x,j∈y)

Lj=0

dx=min{wi,j-Li-Lj}(i∈S,j 不∈T)

Li=Li+dx (i∈S)

Li=Li-dx (i∈T)

【最优匹配】

与最优完备匹配很相似，但不必以完备匹配为前提。

只要对 KM 算法作一些修改就可以了：

将原图转换成完全二分图（m=|x||y|），添加原图中不存在的边，并且设该边的权值为 0。

匈牙利算法：

初始时最大匹配为空

```
while 找得到增广路径
    do 把增广路径加入到最大匹配中去
```

可见和最大流算法是一样的。但是这里的增广路径就有它一定的特殊性，下面我来分析一下。
（注：匈牙利算法虽然根本上是最大流算法，但是它不需要建网络模型，所以图中不再需要源点和汇点，仅仅是一个二分图。每条边也不需要有方向。）

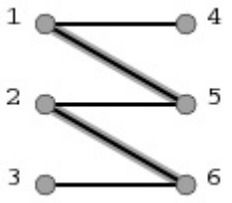


图 1

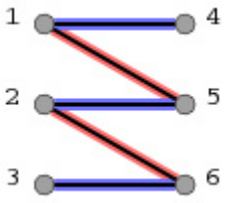


图 2

图 1 是我给出的二分图中的一个匹配：［1，5］和［2，6］。图 2 就是在这个匹配的基础上找到的一条增广路径：3->6->2->5->1->4。过程由 3->6，判断 6 是否被访问过，如果是就再继续查找增广路，否则就停止，因为 2->6 那 6 就是被 2 访问过那，下一个点就是访问 2，然后再判断 2 是否存在除 2->6 的匹配，如果存在就再继续找下去，就是这样不断的找交错轨，一直找到最后的一个点是没有被访问过的。

我们借由它来描述一下二分图中的增广路径的性质：

(1)有奇数条边。
(2)起点在二分图的左半边，终点在右半边。
(3)路径上的点一定是一个在左半边，一个在右半边，交替出现。（其实二分图的性质就决定了这一点，因为二分图同一边的点之间没有边相连，不要忘记哦。）
(4)整条路径上没有重复的点。

(5)起点和终点都是目前还没有配对的点，而其它所有点都是已经配好对的。（如图 1、图 2 所示，［1，5］和［2，6］在图 1 中是两对已经配好对的点；而起点 3 和终点 4 目前还没有与其它点配对。）
(6)路径上的所有第奇数条边都不在原匹配中，所有第偶数条边都出现在原匹配中。（如图 1、图 2 所示，原有的匹配是［1，5］和［2，6］，这两条配匹的边在图 2 给出的增广路径中分边是第 2 和第 4 条边。而增广路径的第 1、3、5 条边都没有出现在图 1 给出的匹配中。）
(7)最后，也是最重要的一条，把增广路径上的所有第奇数条边加入到原匹配中去，并把增广路径中的所有第偶数条边从原匹配中删除（这个操作称为增广路径的取反），则新的匹配数就比原匹配数增加了 1 个。（如图 2 所示，新的匹配就是所有蓝色的边，而所有红色的边则从原匹配中删除。则新的匹配数为 3。）

不难想通，在最初始时，还没有任何匹配时，图 1 中的两条灰色的边本身也是增广路径。因此在这张二分图中寻找最大配匹的过程可能如下：

(1)找到增广路径 1->5，把它取反，则匹配数增加到 1。
(2)找到增广路径 2->6，把它取反，则匹配数增加到 2。
(3)找到增广路径 3->6->2->5->1->4，把它取反，则匹配数增加到 3。
(4)再也找不到增广路径，结束。

当然，这只是一种可能的流程。也可能有别的找增广路径的顺序，或者找到不同的增广路径，最终的匹配方案也可能不一样。但是最大匹配数一定都是相同的。

对于增广路径还可以用一个递归的方法来描述。这个描述不一定最准确，但是它揭示了寻找增广路径的一般方法：
“从点 A 出发的增广路径”一定首先连向一个在原匹配中没有与点 A 配对的点 B。如果点 B 在原匹配中没有与任何点配对，则它就是这条增广路径的终点；反之，如果点 B 已与点 C 配对，那么这条增广路径就是从 A 到 B，再从 B 到 C，再加上“从点 C 出发的增广路径”。并且，这条从 C 出发的增广路径中不能与前半部分的增广路径有重复的点。

比如图 2 中，我们要寻找一条从 3 出发的增广路径，要做以下 3 步：
(1)首先从 3 出发，它能连到的点只有 6，而 6 在图 1 中已经与 2 配对，所以目前的增广路径就是 3->6->2 再加上从 2 出发的增广路径。
(2)从 2 出发，它能连到的不与前半部分路径重复的点只有 5，而且 5 确实在原匹配中没有与 2 配对。所以从 2 连到 5。但 5 在图 1 中已经与 1 配对，所以目前的增广路径为 3->6->2->5->1 再加上从 1 出发的增广路径。
(3)从 1 出发，能连到的不与自己配对并且不与前半部分路径重复的点只有 4。因为 4 在图 1 中没有与任何点配对，所以它就是终点。所以最终的增广路径是 3->6->2->5->1->4。

但是严格地说，以上过程中从 2 出发的增广路径（2->5->1->4）和从 1 出发的增广路径（1->4）并不是真正的增广路径。因为它们不符合前面讲过的增广路径的第 5 条性质，它们的起点都是已经配过对的点。我们在这里称它们为“增广路径”

只是为了方便说明整个搜寻的过程。而这两条路径本身只能算是两个不为外界所知的子过程的返回结果。
显然，从上面的例子可以看出，搜寻增广路径的方法就是 DFS，可以写成一个递归函数。当然，用 BFS 也完全可以实现。

至此，理论基础部份讲完了。但是要完成匈牙利算法，还需要一个重要的定理：

如果从一个点 A 出发，没有找到增广路径，那么无论再从别的点出发找到多少增广路径来改变现在的匹配，从 A 出发都永远找不到增广路径。

要用文字来证明这个定理很繁，话很难说，要么我还得多画一张图，我在此就省了。其实你自己画几个图，试图举两个反例，这个定理不难想通的。（给个提示。如果你试图举个反例来说明在找到了别的增广路径并改变了现有的匹配后，从 A 出发就能找到增广路径。那么，在这种情况下，肯定在找到别的增广路径之前，就能从 A 出发找到增广路径。这就与假设矛盾了。）
有了这个定理，匈牙利算法就成形了。如下：

初始时最大匹配为空

`for` 二分图左半边的每个点 `i`

`do` 从点 `i` 出发寻找增广路径。如果找到，则把它取反（即增加了总了匹配数）。

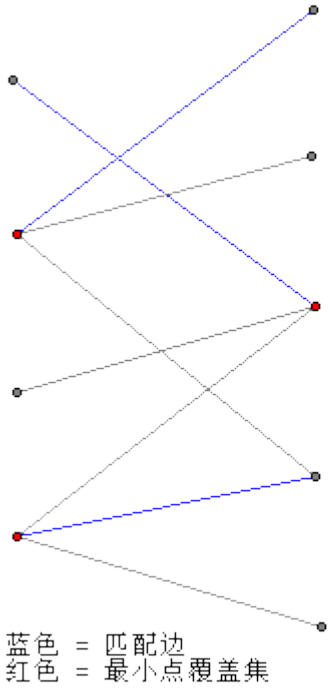
如果二分图的左半边一共有 n 个点，那么最多找 n 条增广路径。如果图中共有 m 条边，那么每找一条增广路径（DFS 或 BFS）时最多把所有边遍历一遍，所花时间也就是 m。所以总的时间大概就是 O（n * m）。

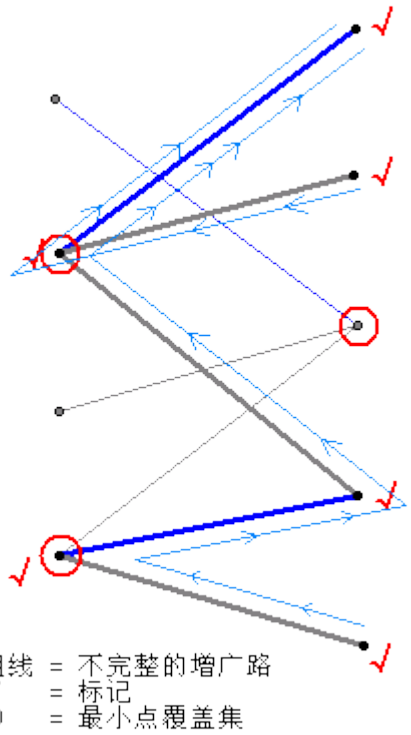
二分图最大匹配的 König 定理及其证明

本文将是这一系列里最短的一篇，因为我只打算把 König 定理证了，其它的废话一概没有。以下五个问题我可能会在以后的文章里说，如果你现在很想知道的话，网上去找找答案：

1. 什么是二分图；
2. 什么是二分图的匹配；
3. 什么是匈牙利算法；(http://www.matrix67.com/blog/article.asp?id=41)
4. König 定理证到了有什么用；
5. 为什么 o 上面有两个点。

König 定理是一个二分图中很重要的定理，它的意思是，一个二分图中的最大匹配数等于这个图中的最小点覆盖数。如果你还不知道什么是最小点覆盖，我也在这里说一下：假如选了一个点就相当于覆盖了以它为端点的所有边，你需要选择最少的点来覆盖所有的边。比如，下面这个图中的最大匹配和最小点覆盖已分别用蓝色和红色标注。它们都等于 3。这个定理相信大多数人都知道，但是网络上给出的证明并不多见。有一些网上常见的“证明”明显是错误的。因此，我在这里写一下这个定理的证明，希望对大家有所帮助。





假如我们已经通过匈牙利算法求出了最大匹配（假设它等于 M），下面给出的方法可以告诉我们，选哪 M 个点可以覆盖所有的边。

匈牙利算法需要我们从右边的某个没有匹配的点，走出一条使得“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”的路（交错轨，增广路）。但是，现在我们已经找到了最大匹配，已经不存在这样的路了。换句话说，我们能寻找到很多可能的增广路，但最后都以找不到“终点是还没有匹配过的点”而失败。我们给所有这样的点打上记号：从右边的所有没有匹配过的点出发，按照增广路的“交替出现”的要求可以走到的所有点（最后走出的路径是很多条不完整的增广路）。那么这些点组成了最小覆盖点集：右边所有没有打上记号的点，加上左边已经有记号的点。看图，右图中展示了两条这样的路径，标记了一共 6 个点（用 “√”表示）。那么，用红色圈起来的三个点就是我们的最小覆盖点集。

首先，为什么这样得到的点集点的个数恰好有 M 个呢？答案很简单，因为每个点都是某个匹配边的其中一个端点。如果右边的哪个点是没有匹配过的，那么它早就当成起点被标记了；如果左边的哪个点是没有匹配过的，那就走不到它那里去（否则就找到了一条完整的增广路）。而一个匹配边又不可能左端点是标记了的，同时右端点是没标记的（不然的话右边的点就可以经过这条边到达了）。因此，最后我们圈起来的点与匹配边一一对应。

其次，为什么这样得到的点集可以覆盖所有的边呢？答案同样简单。不可能存在某一条边，它的左端点是没有标记的，而右端点是有标记的。原因如下：如果这条边不属于我们的匹配边，那么左端点就可以通过这条边到达（从而得到标记）；如果这条边属于我们的匹配边，

那么右端点不可能是一条路径的起点，于是它的标记只能是从这条边的左端点过来的（想想匹配的定义），左端点就应该有标记。

最后，为什么这是最小的点覆盖集呢？这当然是最小的，不可能有比 M 还小的点覆盖集了，因为要覆盖这 M 条匹配边至少就需要 M 个点（再次回到匹配的定义）。

证完了。

个人推荐题目：

Hdu 1068 (Girls and Boys) （基础）
http://acm.hdu.edu.cn/showproblem.php?pid=1068
题意：二分图匹配
解法：匈牙利算法找增广路

Hdu 1083 (Courses) （基础）
http://acm.hdu.edu.cn/showproblem.php?pid=1083
题意：二分图最大匹配
解法：匈牙利算法

Hdu 2813 (One fihgt one) (基础)
http://acm.hdu.edu.cn/showproblem.php?pid=2813
题意：赤裸裸的最优匹配问题
解法：简单 KM 算法

Poj 3041 (Asteroids) (中等)

http://acm.pku.edu.cn/JudgeOnline/problem?id=3041
题意：最大匹配问题（这题比较注重分析能力）
解法：匈牙利算法

网络推荐题目：

POJ 1486 - Sorting Slides(中等)
http://acm.pku.edu.cn/JudgeOnline/problem?id=1486
题意：二分图的必须边
解法：需正真理解最大匹配算法，详见http://hi.baidu.com/kevin0602/blog/item/1d5be63b5bec9bec14cecb44.html

POJ 1904 - King's Quest(中等，好题)
http://acm.pku.edu.cn/JudgeOnline/problem?id=1904
题意：求二分图所有可能的匹配边
解法：虽然最终不是用匹配算法，但需要理解匹配的思想转换成强连通分量问题。

POJ 2060 -Taxi Cab Scheme(基础)
http://acm.pku.edu.cn/JudgeOnline/problem?id=2060
题意：最小路径覆盖

POJ 2594 -Treasure Exploration(中等)
http://acm.pku.edu.cn/JudgeOnline/problem?id=2594
题意：可相交最小路径覆盖
解法：先传递闭包转化下

POJ 3041 - Asteroids(基础)
http://acm.pku.edu.cn/JudgeOnline/problem?id=3041
POJ 2226 - Muddy Fields(基础)
http://acm.pku.edu.cn/JudgeOnline/problem?id=2226
题意：行列的覆盖
解法：最小点集覆盖 = 最大匹配

POJ 2195 - Going Home(基础)
http://acm.pku.edu.cn/JudgeOnline/problem?id=2195
题意：最小权值匹配
解法：KM算法

POJ 2400 - Supervisor, Supervisee(中等)
http://acm.pku.edu.cn/JudgeOnline/problem?id=2400
题意：输出所有最小权匹配
解法：KM, 然后回溯解，汗，输入的两个矩阵居然是反过来的

POJ 2516 -Minimum Cost(中等)
http://acm.pku.edu.cn/JudgeOnline/problem?id=2516
题意：最小权值匹配或最小费用流
解法：拆点 + KM算法(只有正确的才能过)，费用流(ms错的可能也能过)

POJ 3686 - The Windy's(较难)
http://acm.pku.edu.cn/JudgeOnline/problem?id=3686
题意：最小权值匹配
解法：拆点,然后尽管用KM算法去水吧，数据其实弱得不得了 O(50＊50＊2500) -> 16ms
相
关：http://hi.baidu.com/kevin0602/blog/item/2829dc01d7143b087bec2c97.html

SPOJ 412 - K-path cover(较难)
https://www.spoj.pl/problems/COVER/
题意：略
解法：很牛叉的一道匹配
相关：http://hi.baidu.com/roba/blog/item/c842fdfac10d24dcb48f31d7.html

SGU 206. Roads(较难)
http://acm.sgu.ru/problem.php?contest=0&problem=206
解法：经典题目，也可以使用spoj 412 那题的优化