带花树

带花树是解决一般图的最大匹配的算法。

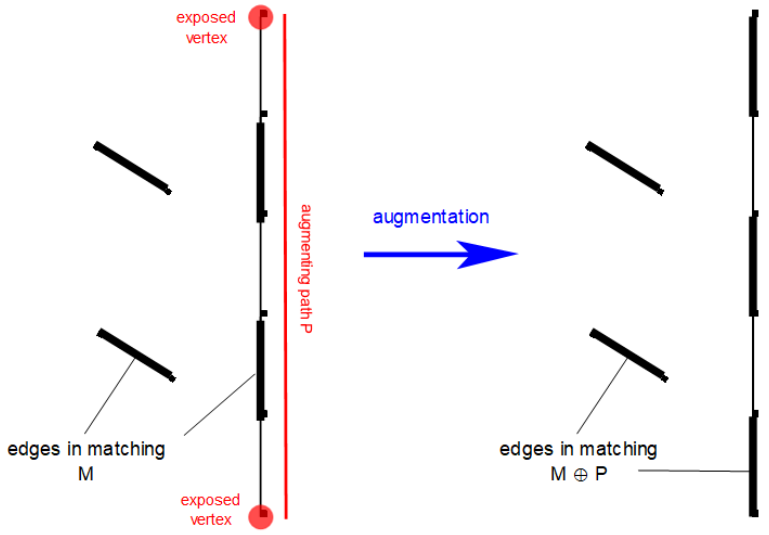
增广路

由于涉及到最大匹配，我们这里对增广路进行一个简短的介绍：

1.如果在一个 M 匹配的图 G 中，有一个点 v 是孤立的指没有与其匹配的点。一条在G中的路径如果其边在M中交替出现，则称为交错路$alternating path$一条增广路$augmenting path$$P$是一条开始并结束于不相同的孤立点的交错路。

2.在增广路中不是匹配的边比是匹配的边多，因此增广路的边条数为奇数。

3.一个匹配在增广路上的增广即为异或操作$M1=M xor P $



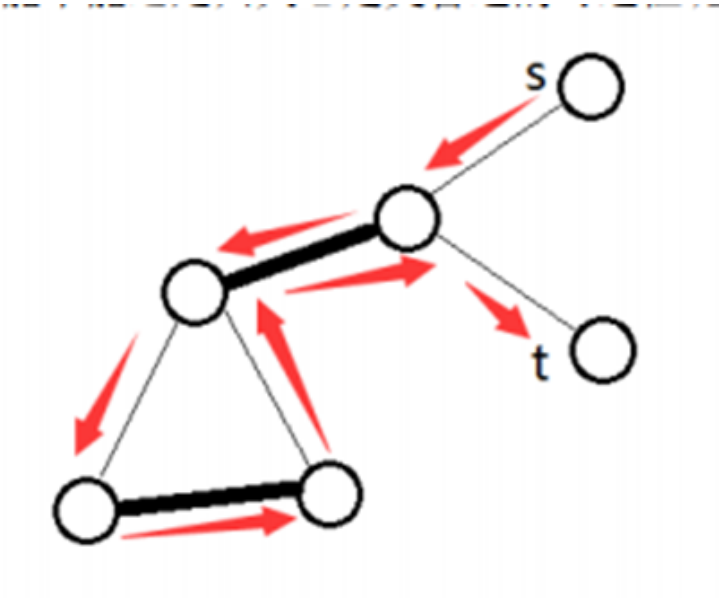
一般图中增广路与二分图的区别

考虑下我们之前在二分图中找增广的过程：

1.我们寻找增广路的时候，会将路径上的点黑白染色，匹配只存在于黑点白点之间。

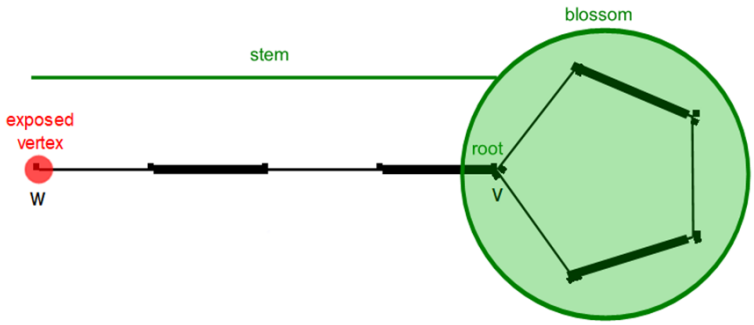
2.如果没有环或是只有偶环（二分图中只有偶环），那么每个点的颜色是确定的。

3.但是如果出现了奇环，那么点的颜色不再确定，因为奇环顺时针走一圈和逆时针走一圈的染色结果是不同的。(如下图所示，模拟一次增广的染色，我们可以明显发现进入奇环的那个点的染色结果出现冲突)



花与收缩的定义

在一个 M 匹配的图 G 中，一朵花 B 是一个图 G 中的包含 $2k+1$ 条边的奇环，其中 k 条边在 M 中，并存在从环上任意一个点 v (花根)到一个孤立点 w 的交错路（花茎）。



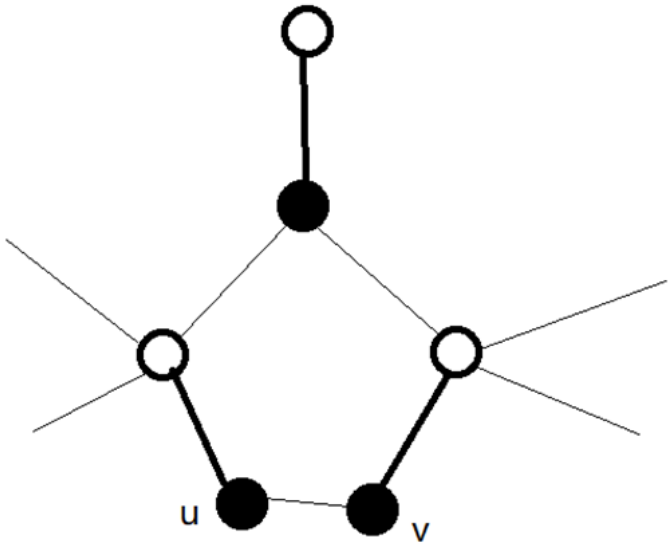
如何找到一朵花

1.从一个孤立的点 w 开始遍历图。

2.从孤立点 w 开始遍历，标记 w 为 o 型点□$out of M$□□

3.交替地用 i 和 o 标记结点，保证无两个相邻节点有标记。

4.如果找到两个相邻结点含有标记 o □那么我们就找到了一个奇环和一朵花。

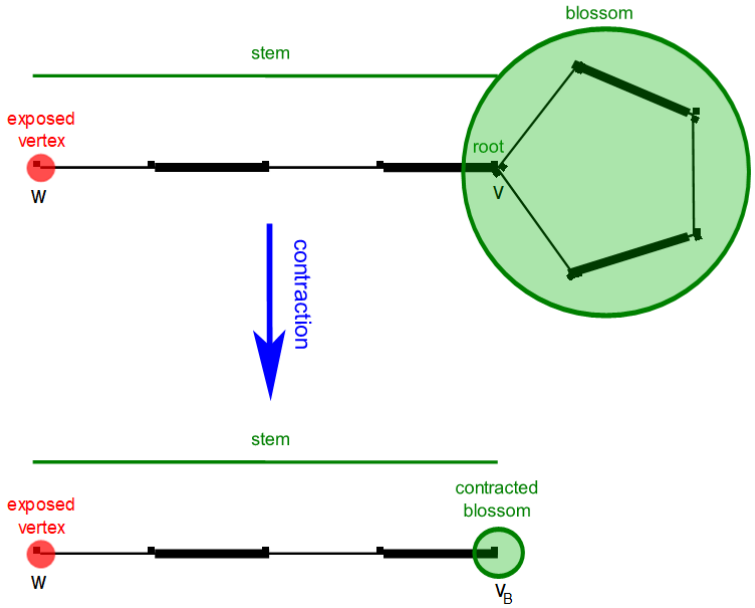


收缩的定义

定义收缩图 G' 是图 G 将所有花 B 缩为点后的图。

定义收缩匹配 M' 是在 G' 中的匹配 M

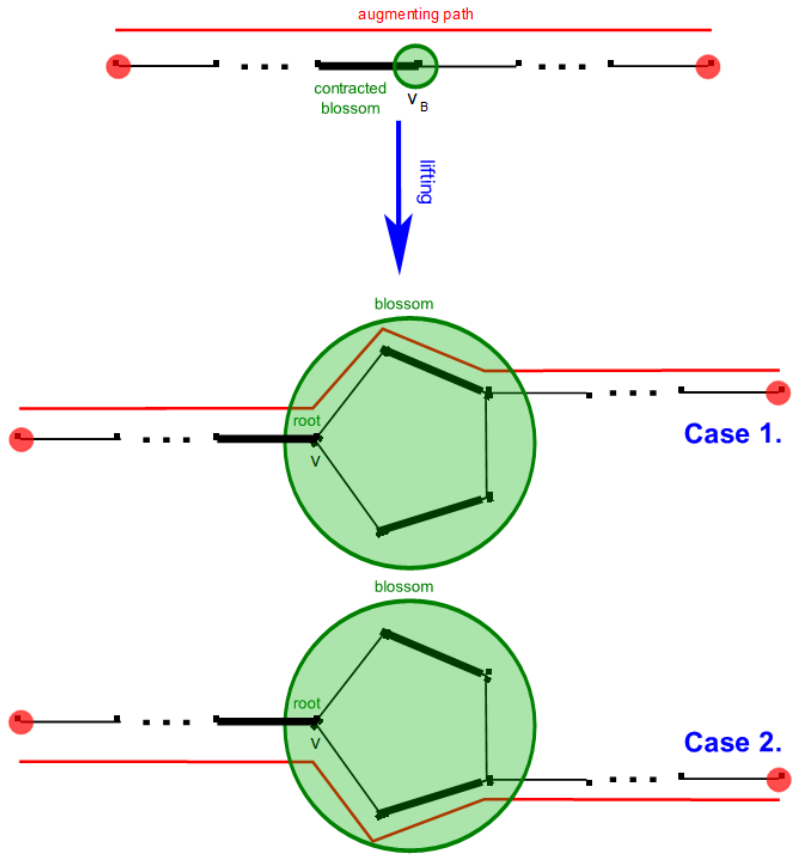
G' 含有 M' 的增广路当且仅当 G 含有 M 的增广路，且任何 M' 在图 G' 的增广路 P' 都可以通过展开收缩的花还原G中的匹配 M 因此如果存在任意一条增广路 P' 通过收缩的花 V_B 都可以找到合适的增广路通过花 B 因此我们可以将一朵花缩成一个点。

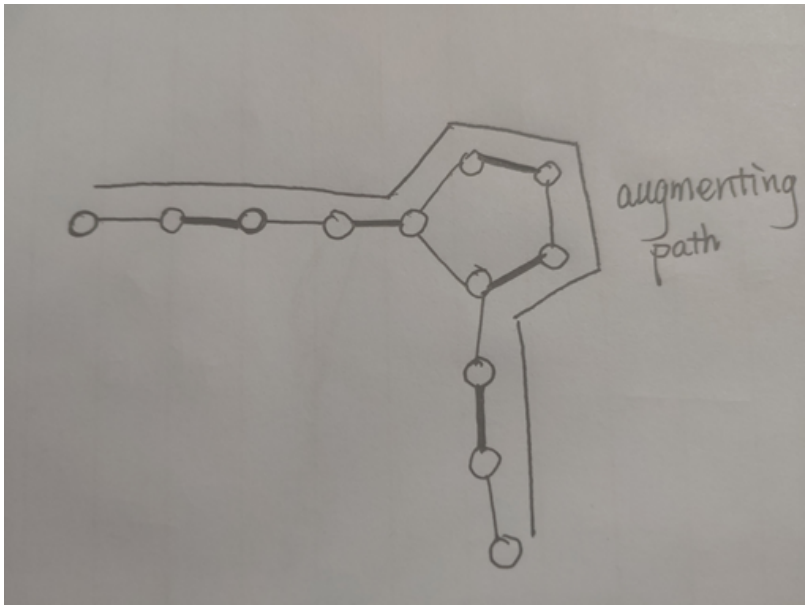


收缩的正确性证明

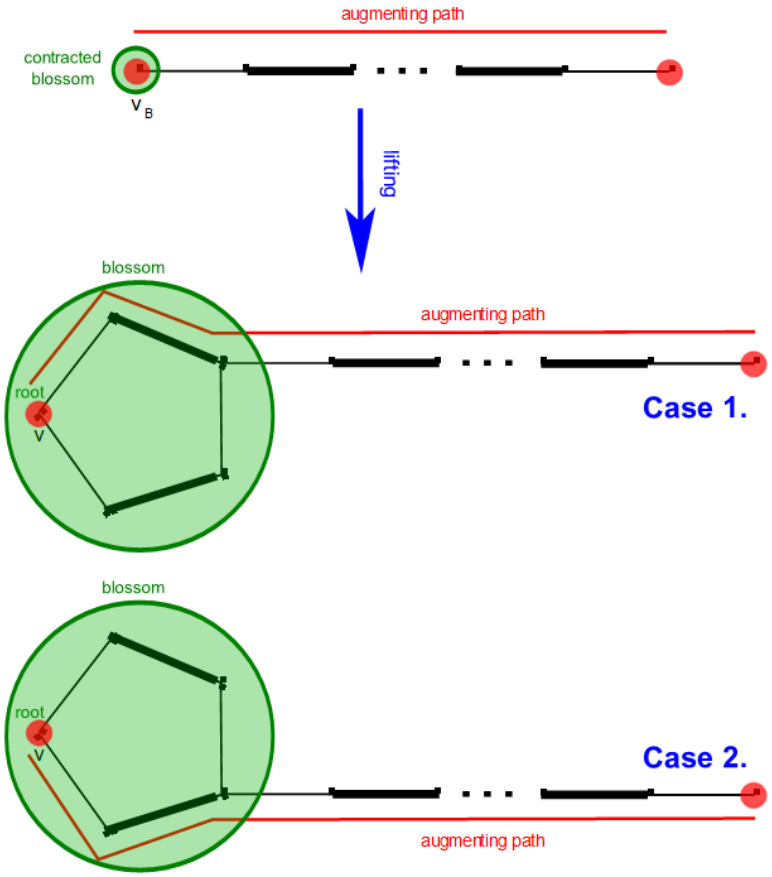
如以下几图所示，如果 P' 在图 G' 中经过 $u \rightarrow V_B \rightarrow w$,这条增广路可以在 G 中配替换为 $u \rightarrow (u'\rightarrow \ldots\rightarrow w') \rightarrow w$, 其中 u' 与 w' 在花 B 中。路径 $u' \rightarrow w'$ 的选择需保证构成的新增广路仍然是交替的。（其中 u' 即在花

B 中也在匹配 M 中（花根□□$w' \rightarrow w$ 是一条增广路）





若 P' 在 V_B 结束，这条增广路可以在 G 中被替换为 $u \rightarrow (u' \rightarrow \ldots \rightarrow v')$□其中 u' 与 v' 在花 B 中。路径 $u' \rightarrow v'$ 的选择需保证构成的新增广路仍然是交替的。（其中 v' 是孤立的（花根□□$u \rightarrow u'$ 是一条增广路）



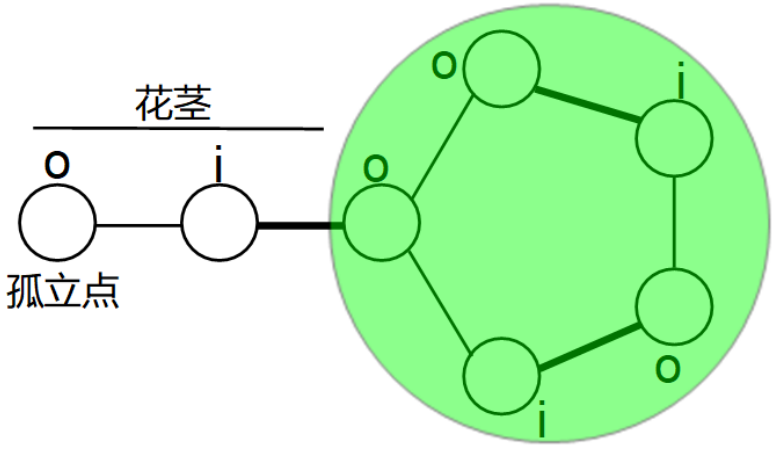
继续观察可以发现，整个奇环的匹配状态只与顶点的匹配状态有关，如果在后来的某一次寻找时奇环上的匹配被改变了，那么顶点的颜色唯一决定了整个环的匹配边是如何走的。（花上的任意一个点都可作为o型点出边）

细节方面的一些问题

花根的匹配关系与花

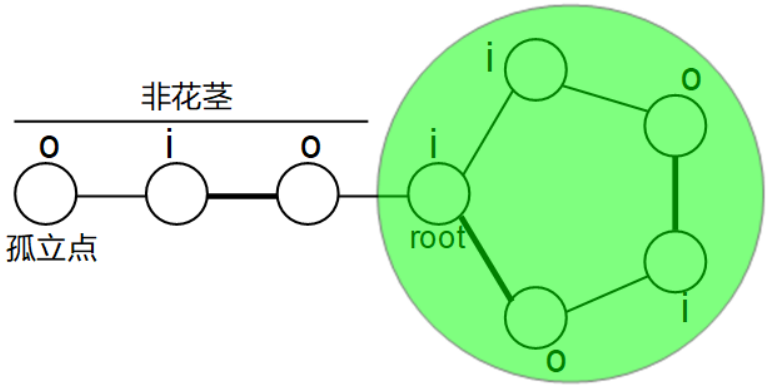
为什么我们仅仅将花根与环外匹配的环称为花？为什么花根与环内匹配不能称为一朵花？

因为带花树缩花的原因是为了防止寻找增广路的时候绕环一圈后重新进入花茎而发生干扰。如果花茎是这样的：



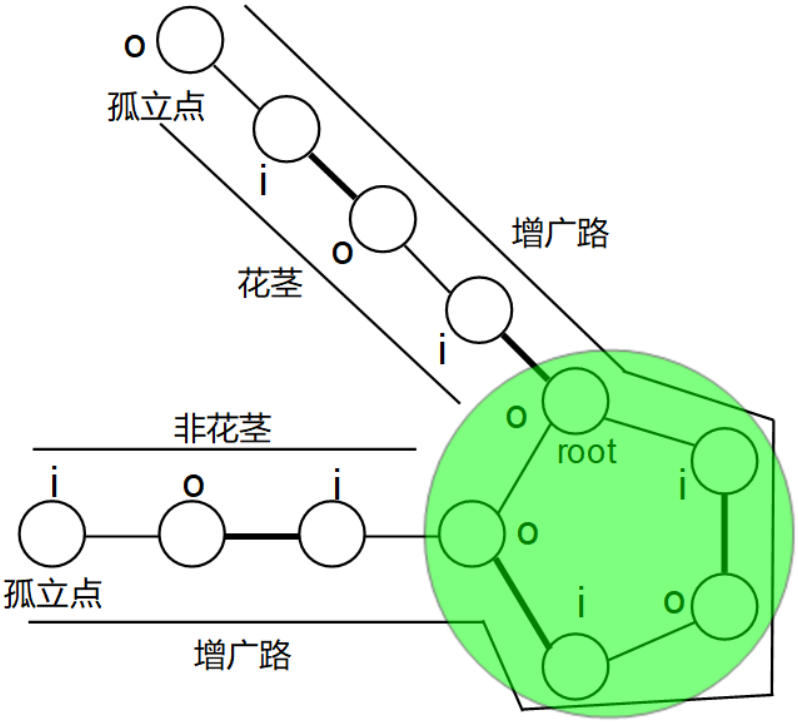
此时o标记的点相邻，因为队列中的点均为o标记点，所以可能会与花茎发生干扰。

如果花茎是这样的：



i标记点相邻，显然不会与花茎发生干扰~~甚至有可能直接增广成功子~~。虽然该环是个奇环，但显然当前的交错路不会是花茎，当前花茎相连的点也不是花根。

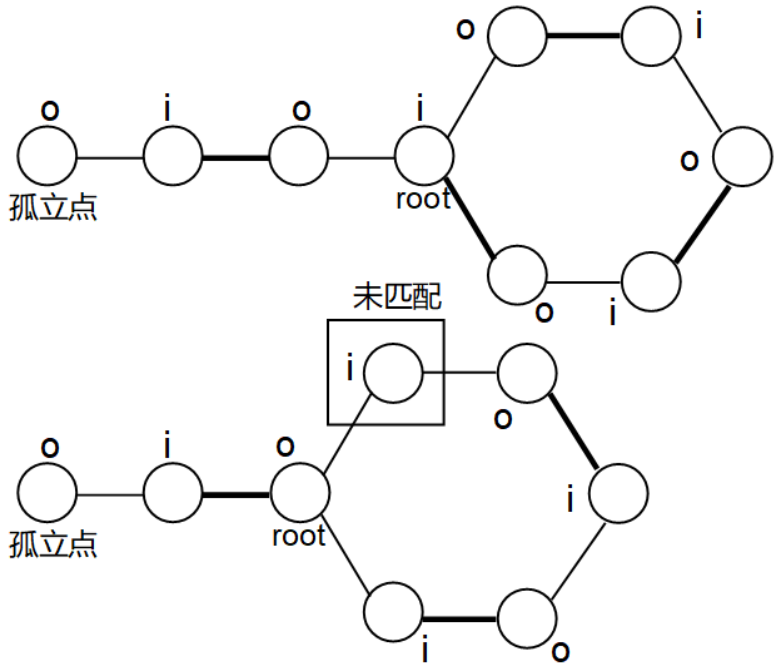
如果我们想让上图成为一朵花，那么应该是下图的形式：



偶环处理

为什么我们不对偶环进行处理？

因为如果是偶环，那么从任意一个点开始进行标记，标记是不会发生冲突的（偶环是二分图）。故不会对花茎产生干扰，限制其匹配即可。



匹配方案

如何输出匹配方案？

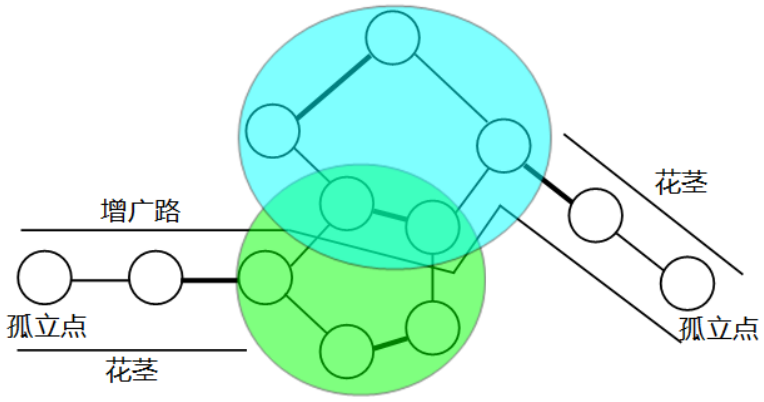
我们不进行显式缩点，记录每一个点所属于的花，每一次寻找到增广路后就可以直接沿着增广路维护信息

多花嵌套

不进行显式缩点，如果有多朵花嵌套会对算法过程有影响吗？

答案是不会！

我们只需考虑多个花嵌套时对单条增广路的影响：



如图~~反正我没找到反例~~，我们发现，无论花如何嵌套，我们始终可以找出沿着花边走的一条合法交错路。（在上一部分已经说明过可以找出一朵花的合法路径，因此也可以推广到嵌套）

寻找增广路

结点信息

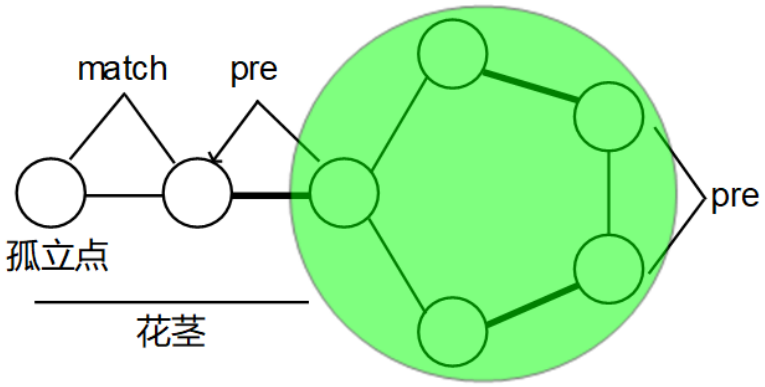
对于每个结点我们记录：

1.$type:$ 这个点的类型，0 为 o 型点，1 为 i 型点。

2.$match:$ 这个点所匹配的点。注意 $match$ 是双向指针，如果 $match[u]=v$,则 $match[v]=u$

3.$pre:$ 这个点在交错路中相邻但不和当前点构成匹配的点，若$match[u]$ 与其他点匹配了，则需要将$match[u]$ 置为 $pre[u]$注意pre 在非花的增广路中是单向的，由在交错路中的后继指向其前驱，在花中 pre 指针是双向的。

4.$father:$ 这个点所属的花，若不属于任何花则置为本身。为了方便输出方案，我们不进行显式缩点。因此需要引入这个指针。



增广过程

我们一次从每一个孤立点开始宽搜，并依次用i和o标记图上的点:

1.若寻找到一个未被标记的未匹配点：将其标记为i型点，找到一条增广路，更新并维护该增广路的信息，完成增广。

2.若寻找到一个未被标记的点，但其已经被匹配，将其标记为i型点，并将其匹配的点标记为o型点，加入队列。

3.若寻找到一个已经被标记的i型点，说明此时构成偶环，直接无视。

4.若寻找到一个已经被标记的o型点，说明此时构成奇环且构成花，在当前扩展出的交错路上找到其公共祖先lca,lca此时为花根，沿着两侧的花边爬到花根，将路径上的结点father指针更新，标记全部更新为o并将pre指针变为双向指针。

指针定义

为什么链的交错路上的pre是单向指针，而花上的pre指针式双向指针？

因为在链的交错路上增广方向确定，只需要单向指针即可，而花上我们不确定会从什么方向增广，故需要双向指针。

例题

由于这个算法使用较少，用到也基本上是模板，这里给出一道模板题。UOJ#79 一般图最大匹配

代码实现

===

```
#include<algorithm>
#include<stack>
#include<ctime>
#include<cstring>
#include<cmath>
#include<iostream>
#include<iomanip>
#include<cstdio>
#include<queue>
#include<vector>
using namespace std;
inline int read(){
    int num=0,f=1;char x=getchar();
    while(x<'0'||x>'9'){if(x=='-')f=-1;x=getchar();}
    while(x>='0'&&x<='9'){num=num*10+x-'0';x=getchar();}
    return num*f;
}
const int maxn=505;
int n,m,ti;
vector<int> l[maxn];
int match[maxn],fa[maxn],pre[maxn];
int ty[maxn],vst[maxn];
int getf(int x){
    return (x==fa[x])?x:fa[x]=getf(fa[x]);
}
int LCA(int x,int y){//找到花根
    ++ti;
    x=getf(x);y=getf(y);
    while(vst[x]!=ti){
        if(x){
            vst[x]=ti;
            x=getf(pre[match[x]]);
        }
        swap(x,y);
    }
    return x;
}
```

```
queue<int> Q;
void blossom(int x,int y,int lca){//从两个冲突点开始，将所有的i型点加入可增广的队列中，即开花操作。
    while(getf(x)!=lca){
        pre[x]=y;
        y=match[x];
        if(ty[y]==1){
            ty[y]=0;
            Q.push(y);
        }
        if(getf(x)==x)fa[x]=lca;
        if(getf(y)==y)fa[y]=lca;
        x=pre[y];
    }
}
int aug(int s){
    for(int i=1;i<=n;++i)fa[i]=i,ty[i]=-1;
    while(!Q.empty())Q.pop();
    ty[s]=0;Q.push(s);
    while(!Q.empty()){
        int nw=Q.front();Q.pop();
        for(int nxt:l[nw]){
            if(ty[nxt]==-1){
                pre[nxt]=nw;
                ty[nxt]=1;
                if(!match[nxt]){//找到增广路直接增广
                    for(int to=nxt,from=nw;to;from=pre[to]){
                        match[to]=from;
                        swap(match[from],to);
                    }
                    return true;
                }
                ty[match[nxt]]=0;
                Q.push(match[nxt]);
            }else if(ty[nxt]==0&&getf(nw)!=getf(nxt)){//发现冲突且没有在一朵花中
                int lca=LCA(nw,nxt);
                blossom(nw,nxt,lca);
                blossom(nxt,nw,lca);
            }
        }
    }
    return false;
}
int main(){
    n=read();m=read();
    for(int i=1,x,y;i<=m;++i){
        x=read();y=read();
        l[x].push_back(y);
        l[y].push_back(x);
    }
```

```
    int ans=0;
    for(int i=1;i<=n;++i)
        if(!match[i])ans+=aug(i);
    printf("%d\n",ans);
    for(int i=1;i<=n;++i)printf("%d ",match[i]);
    return 0;
}
```