网络流_3 最大流之二分图匹配

> 最大流问题一般求集合最优解，对于所有可行解的集合P：
>
> [1]对于流网络的所有可行流的集合，对于可行解中的一个解，有且只有一个可行流与之符合
>
> [2]对于流网络中的任何一个可行流都能对应一个可行解
>
> 则可行解中的最大值等于可行流中的最大流

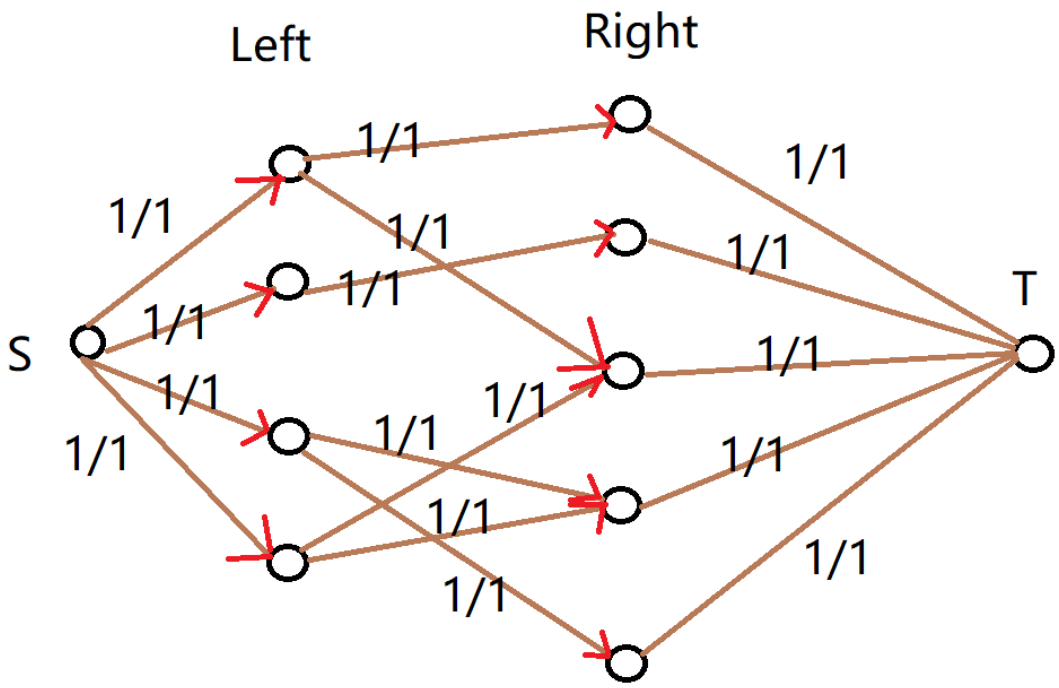
- 匈牙利算法本质上是 EK 算法

对于一个匹配：左集合S和右集合T，求其二分图的最大匹配；

> 一般做法：匈牙利算法，时间复杂度O(n)

网络流做法:

按照网络流的方式建立出源点和汇点



> 从源点S到汇点T，建立从源点S流向左集合S，根据匹配关系从左集合到右集合，从右集合到汇点T的所有容量为 1 的边，对于此流网络的最大流便是最大二分匹配。

网络流24题

1飞行员配对方案问题

> m个外籍和n-m个英国飞行员两两配对，问最大匹配数和匹配方案

- 建图方式：

根据流网络的定义，先建立源点S和汇点T，从源点S向所有外籍飞行员建立一条容量为1的边，从所有外籍飞行员向其可搭档的英国飞行员建立一条容量为1的边，从所有英国飞行员向汇点T建立一条容量为1的边。整个流网络的可行流中的最大流便是最大匹配方案

证明：略（从流量守恒和容量限制来证明）

> 关于最大流一定含有整数型可行流的证明：
>
> 由之前`Dinic`算法的证明可知`Dinic`算法的正确性，由于`Dinic`算法中的一切数据类型都是整型，`Dinic`算法得出的答案包含于最大流集合且整数值可行流集合中存在最大流，因此得证

```
const int INF = 1e8;
int n,m,S,T,x,y;
int e[M],ne[M],f[M],h[N],idx=0;
int q[N],d[N],cur[N];


void add(int a,int b,int c){
    e[idx] = b , f[idx] = c, ne[idx] = h[a] , h[a] = idx ++;
    e[idx] = a , f[idx] = 0, ne[idx] = h[b] , h[b] = idx ++;
}

bool bfs(){
    memset(d,-1,sizeof d);
    int hh = 0 , tt = 0;
    q[0] = S , cur[S] = h[S] , d[S] = 0;

    while(hh <= tt){
        int u = q[hh ++];
        for(int i = h[u];~i;i=ne[i]){
            int j = e[i];
            if(d[j] == -1 && f[i]){
                d[j] = d[u] + 1;
                cur[j] = h[j];
                if( j == T) return true;
                q[++ tt] = j;
            }
        }
    }
    return false;
}

int find(int u,int limit){
    if(u == T) return limit;

    int flow = 0;

    for(int i = cur[u];~ i && flow < limit;i=ne[i]){
        cur[u] = i;
        int ver = e[i];
        if(d[ver] == d[u] + 1 && f[i]){
            int t = find(ver,min(f[i],limit-flow));
            if(!t) d[ver] = -1;
            f[i] -= t , f[i^1] += t , flow += t;
        }
    }
    return flow;
}

int dinic(){
    int ans = 0 , flow = 0;
    while(bfs()) while(flow = find(S,INF)) ans += flow;
```

```
    return ans;
}
//================================
int main(){
    memset(h,-1,sizeof h);

    m = read() , n = read();
    S = 0 , T = n + 1;

    rep(i,1,m) add(S,i,1);
    rep(i,m+1,n) add(i,T,1);

    x = read() , y = read();
    while((~x) && (~y))
    {
        add(x,y,1);
        x = read() , y = read();
    }
    print(dinic());

    for(int i = 0;i < idx;i += 2){
        if(e[i] > m && e[i] <= n && !f[i])
            printf("%d %d\n",e[i^1], e[i]);
    }

    return 0;
}
```

匈牙利算法

> 等价于EK算法,`dfs`一次给每一个飞行员分配搭档，当分配成功时累加答案即可

```
int n,m,a,b,cnt=0;
int e[M],h[N],ne[M],w[M],idx = 0;
int match[N];
bool st[N];

void add(int a,int b){
    e[idx] = b, ne[idx] = h[a] , h[a] = idx ++;
}

bool find(int u){
    for(int i = h[u];~ i;i = ne[i]){
        int j = e[i];
        if(st[j]) continue;
        st[j] = true;
        if(match[j] == 0 || find(match[j])){
            match[j] = u;
            return true;
        }
    }
    return false;
}

signed main(){
    memset(h,-1,sizeof h);
    m = read() , n = read();
```

```
    while(cin >> a >> b , (~a) && (~b))
        add(a,b) , add(b,a);
    rep(i,1,n){
        memset(st,0,sizeof st);
        if(find(i)) cnt ++ ;
    }

    memset(st,0,sizeof st);
    print(cnt/2);

    rep(i,1,n){
        if(match[i] && !st[i] && !st[match[i]]){
            st[i] = true;
            st[match[i]] = true;
            printf("%d %d\n",i,match[i]);
            // cout << i << " " << match[i] << endl;
        }
    }
    return 0;
}
```

2.圆桌问题

> 对于m个单位,第i个单位`r[i]`个人，n张餐桌，第`i`个餐桌容纳`c[i]`个人，要求一张餐桌上不能出现二个人来自同一家公司，问是否有方案能满足所有人都能有座位

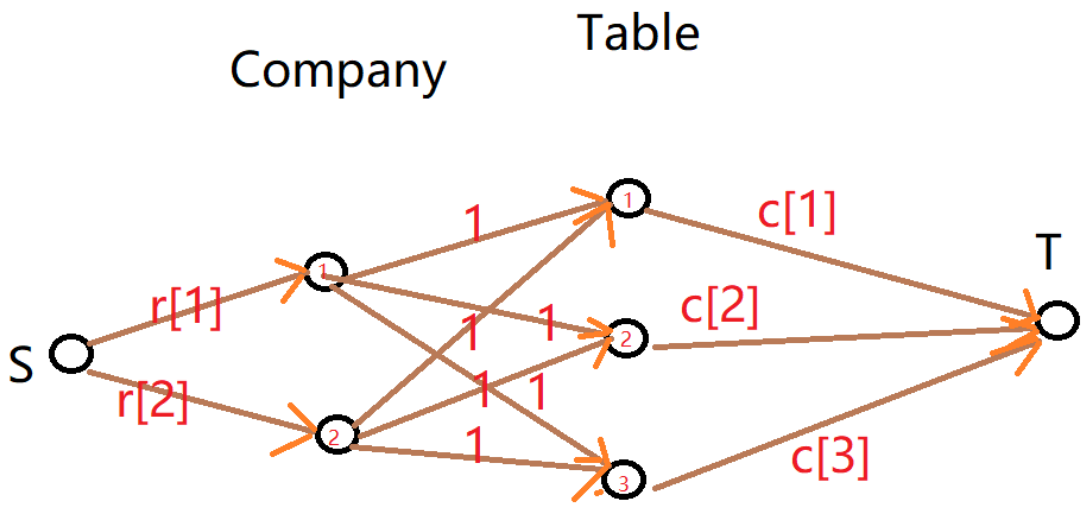
- 建图:

左集合S为每一家公司，右集合D为每一张桌子。建立源点S和汇点T，从源点向S集合的每一家公司连一条容量为`r[i]`的边，从S集合每一家公司向D集合每一张餐桌连一条容量为1的边，从D集合每一张餐桌向汇点T连一条容量为`c[i]`的边。

- 证明:

$$s \rightarrow f \text{可行解对应可行流}$$

左边：每个公司派到对应桌子的人；右边：每张桌子坐的人；



- 容量限制（从实际意义出发）
  - 1.对于中间 每个单位到一张桌子最多一个人 --> 流量 <= 容量 = 1
  - 2.对于左边 流量 = 公司派出人数 <= 公司人数 = 容量

    - 3.对于右边 流量 = 桌子坐的人 <= 桌子容量 = 容量
- 流量守恒
    - 每个公司派出的人坐且只坐一张桌子

$$f \to s$$

显然，构造就是按照这个构造的，证明略

AC代码

判断是否存在方案：先算出源点流出的流量，与`dinic`得出的最大流比较，若小于则不存在方案

```
const int INF = 1e8;
const int N = 510, M = 100010;
int n,m;
int e[M],ne[M],h[N],f[M],idx=0,S,T,c;
int q[N],cur[N],d[N];
bool st[N],vis[N];

void add(int a,int b,int c){
    e[idx] = b , f[idx] = c , ne[idx] = h[a] , h[a] = idx ++;
    e[idx] = a , f[idx] = 0 , ne[idx] = h[b] , h[b] = idx ++;
}

bool bfs(){
    memset(d,-1,sizeof d);
    int hh =  0 , tt = 0;
    q[0] = S, d[S] = 0, cur[S] = h[S];

    while(hh <= tt){
        int u = q[hh ++];
        for(int i = h[u]; ~i ;i=ne[i]){
            int ver = e[i];
            if(d[ver] == -1 && f[i]){
                d[ver] = d[u] + 1;
                cur[ver] = h[ver];
                if(ver == T) return true;
                q[++ tt] = ver;
            }
        }
    }
    return false;
}

int find(int u,int limit){
    if( u == T) return limit;
    int flow = 0;

    for(int i = cur[u]; ~ i && flow < limit; i = ne[i]){
        cur[u] = i;

        int ver = e[i];
        if(d[ver] == d[u] + 1 && f[i]){
            int t = find(ver,min(f[i],limit - flow));
            if(!t) d[ver] = -1;
            f[i] -= t , f[i^1] += t , flow += t;
        }
```

```
    }
    return flow;
}

int dinic(){
    int ans = 0 , flow = 0;
    while(bfs()) while(flow = find(S,INF)) ans += flow;
    return ans;
}
//================================
int main(){
    memset(h,-1,sizeof h);

    int tot = 0;
    m = read() , n = read();
    S = 0 , T = n + m + 1;
    rep(i,1,m) {
        c = read();
        add(S,i,c);
        tot += c;
    }

    rep(i,1,n){
        c = read();
        add(i+m,T,c);
    }

    rep(i,1,m)
    rep(j,1,n)
        add(i,j+m,1);

    if(dinic()!=tot) puts("0");
    else{
        puts("1");
        /*寻找答案*/
        for(int i = 1; i <= m ;i ++ ){
            for(int j=h[i];~j;j=ne[j]){
                if(e[j] > m && e[j] <= n+m && !f[j])
                    printf("%d ",e[j]-m);
            }
            puts("");
        }
    }
    return 0;
}
```

[网络流24题]最小路径覆盖

> 给定有向图G=(V,E)。设P 是G 的一个简单路（顶点不相交）的集合。如果V 中每个顶点恰好在P 的一条路上，则称P是G 的一个路径覆盖。P 中路径可以从V 的任何一个顶点开始，长度也是任意的，特别地，可以为0。G 的最小路径覆盖是G 的所含路径条数最少的路径覆盖。
> 设计一个有效算法求一个有向无环图G 的最小路径覆盖。
> 提示：设$V=1，2，.....，n$，构造网络$G1=(V1,E1)$如下：
>
> $V_1=\{x_0,x_1,...,x_n\}\cup\{y_0,y_1,...,y_n\}$*V$

> $E_1 = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_i) : (i, j) \in E\}$
> 每条边的容量均为1。求网络G1的(x0,y0)最大流。编程任务：对于给定的给定有向无环图G，编程找出G的一个最小路径覆盖。

最小路径覆盖 = 总点数 − 最大匹配数，因此这是一道匹配类问题。像飞行员方案那样，我们把每一个点拆分成左边的匹配点和右边的被匹配点，然后从源点向所有匹配点连容量为1的边，从所有被匹配点向汇点连容量为1的边，如果两个点之间有边，则从一匹配点向另一待匹配点连边。最后建好的网络跑一遍最大流，每一天走过的边对应一条且只属于一条路径，最小路径覆盖即为总点数减去最大流。

那么我们怎么找路径呢？沿用y总上课用的方法，遍历每一条不经过不从源点指向，且不指向汇点的点，每次沿着容量为0(表面被流过了)的边走，开一个数组记录哪些点被走过了。正常dfs解决即可

```
void dfs(int u){
    st[u] = true;
    printf("%d ",u);
    for(int i=h[u];~i;i=ne[i]){
        if(e[i^1] == T || e[i] == S ) continue;
        if(e[i]<=n && st[e[i]]) continue;
        else if(e[i] > n && st[e[i]-n]) continue;
        if(f[i] == 0)
            if(e[i] > n ) dfs(e[i]-n);
            else dfs(e[i]);
    }
}
```

Code:

```
#include<bits/stdc++.h>
using namespace std;
const int N = 510, M = (N+6010)*2,INF=1e8;
#define rep(i,ll,rr) for(int i=ll;i<=rr;++i)
int n,m,F,D,S,T;
int e[M],ne[M],h[N],f[M],idx=0;
int cur[M],q[N],d[N];
void add(int a,int b,int c){
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
}

string node(int x){
    if(x==0) return "S";
    if(x<n) return to_string(x);
    if(x==n) return "11";
    if(x<=2*n)return to_string(x-n);
    return "T";
}

int find(int u,int limit){
    if(u == T) return limit;
    int flow =0 ;

    for(int i=cur[u];~i && flow < limit; i = ne[i]){
        cur[u] = i;
        int ver = e[i];
        if(d[ver] == d[u] + 1 && f[i]){
            int t = find(ver,min(f[i],limit-flow));
```

```
            if(!t) d[ver] = -1;
            f[i] -= t, f[i^1] += t, flow += t;
        }
    }
    return flow;
}

bool bfs(){
    memset(d,-1,sizeof d);
    int hh = 0, tt = 0;
    q[0] = S, cur[S] = h[S], d[S] = 0;

    while(hh <= tt){
        int u = q[hh ++];
        for(int i=h[u];~i;i=ne[i]) {
            int ver = e[i];
            if(d[ver] == -1 && f[i]){
                d[ver] = d[u] + 1;
                cur[ver] = h[ver];
                if(ver == T) return true;
                q[++ tt] = ver;
            }
        }
    }
    return false;
}

int dinic(){
    int ans = 0 ,flow = 0;
    while(bfs()) while(flow = find(S,INF)) ans += flow;
    return ans;
}

bool st[N];

void dfs(int u){
    st[u] = true;
    printf("%d ",u);
    for(int i=h[u];~i;i=ne[i]){
        // if(e[i] <= n) continue;
        if(e[i^1] == T || e[i] == S ) continue;
        if(e[i]<=n && st[e[i]]) continue;
        else if(e[i] > n && st[e[i]-n]) continue;
        if(f[i] == 0) {
            if(e[i] > n ) dfs(e[i]-n);
            else dfs(e[i]);
        }
    }
}

signed main(){
    memset(h,-1,sizeof h);
    scanf("%d%d",&n,&m);
    S=0,T=n*2+1;
    rep(i,1,n) add(S,i,1),add(i+n,T,1);
    rep(i,1,m){
        scanf("%d%d",&F,&D);
        add(F,D+n,1);
```

```
    }
    int ans = dinic();
    for(int i=0;i<idx;i+=2){
        if(e[i^1]==S||e[i]==T) continue;
        if(f[i]==0&&!st[e[i^1]]){
            dfs(e[i^1]); puts("");
        }
    }
    printf("%d\n",n-ans);
    return 0;
}
```