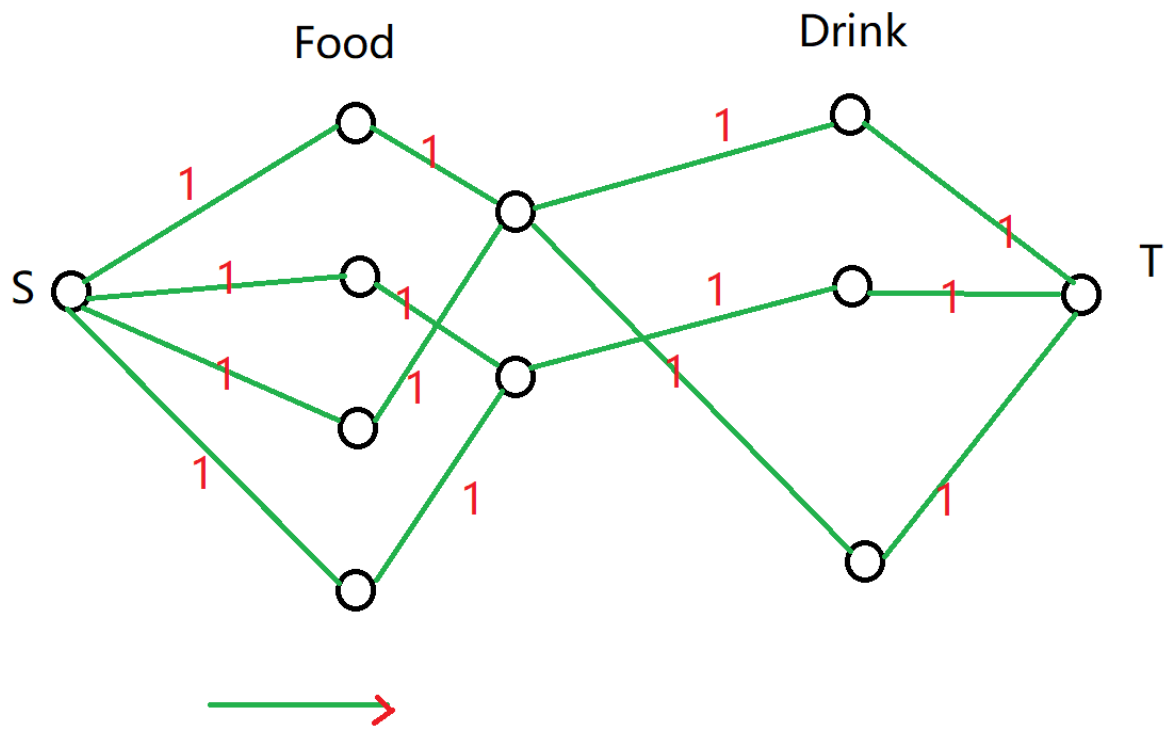
网络流__8 拆点

餐饮奶牛

> n头奶牛，第i头有`F[i]`种喜欢的食物和`D[i]`种喜欢的饮料。现在有提供了`F`种食物和`D`种饮料，求能够分配的有吃有喝的最大奶牛数量

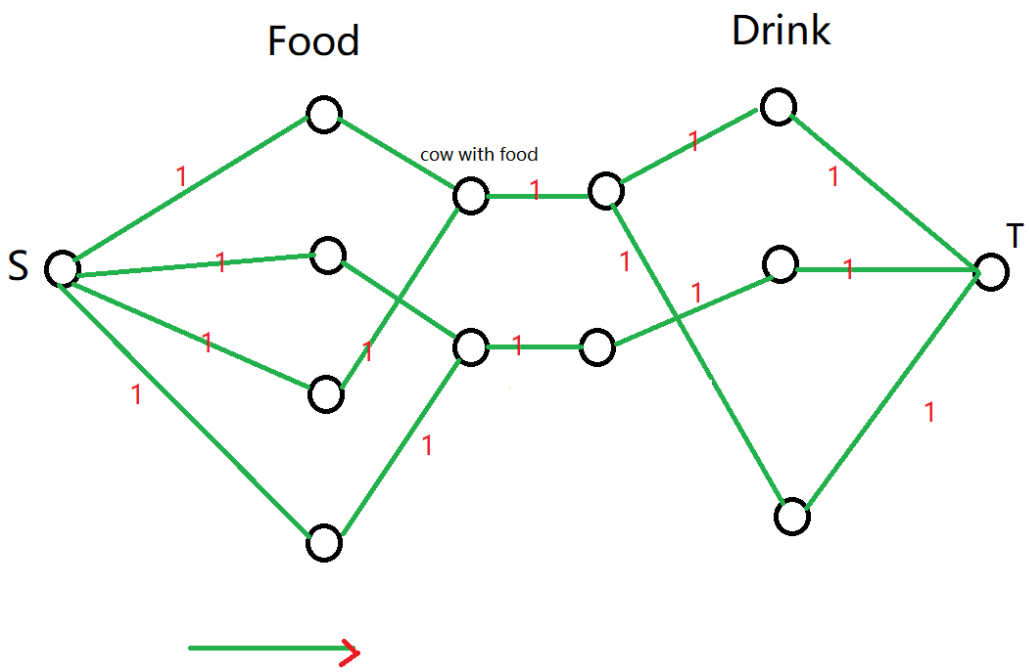
看到这道题目，第一想法是这样子建图：



但是这样子建图的问题会出现：对于原图任一可行流不一定能对应一种分配方式，因为同一个点可以流入多个流量并流出多个流量，与每个奶牛只一个食物一个饮料矛盾。因此我们要使用拆点

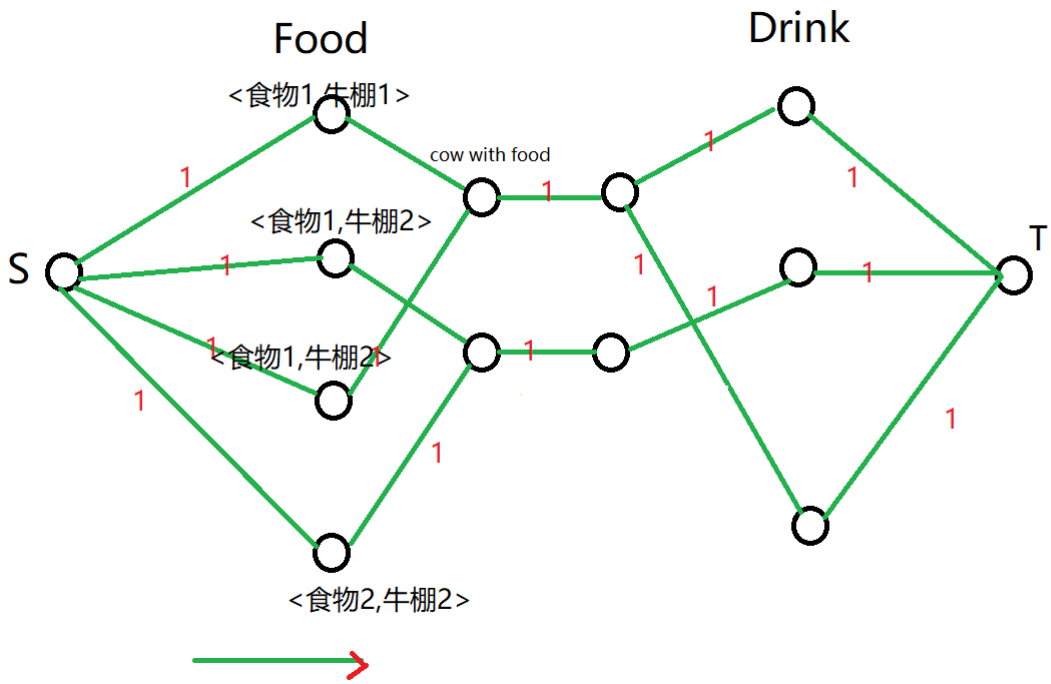
> 拆点： 将一个点拆成多个点来表示当前点的不同状态

本题的拆点方式如下：



将cow拆成两个点，两点以一条容量为1的边相连，能够限制从而不出现上述情况，这样保证了一头牛只会匹配一个食物和饮料。

在y总下面的视频我曾经评论：假如这头牛有三个或者更多的需求的话该怎么做呢？显然，当时我的想法确实是一种方法，那就是`将其中两个需求通过组合变成一个新点`。例如本题中假如有两种食物，三种饮料，还有两种牛的饮食牛棚的话，我们可以将牛棚和食物通过组合的方式构造出新的四个点，然后再将对应的点进行连接，建图就完成了~



下面是代码：

```
const int N = (110+110+2*110+2)*2, M = (N + 110*110*4)*2;
const int INF = 1e8;
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];
struct Cow{
    int f, d, F[N], D[N];
```

```
}cow[N];

int get(int num,int p){
    if(p==0) return num; //food
    else if(p==1) return F + num; //cow head
    else if(p==2) return F + n + num; //cow tail
    return F + 2 * n + num; //drink
}

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 ++;
}

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

```
    n = read(), F = read(), D = read();
    S = 0, T = n*2 + F + D + 1;

    rep(i,1,F) add(S,get(i,0),1); //S -> food = 1
    rep(i,1,n){
        cow[i].f = read(), cow[i].d = read();
        rep(j,1,cow[i].f) cow[i].F[j] = read();
        rep(j,1,cow[i].d) cow[i].D[j] = read();
    }
    rep(i,1,n) rep(j,1,cow[i].f) add(get(cow[i].F[j],0), get(i,1),1); //food ->
cow = 1
    rep(i,1,n) add(get(i,1),get(i,2),1); //cowhead -> cowtail = 1
    rep(i,1,n) rep(j,1,cow[i].d) add(get(i,2), get(cow[i].D[j],3),1); //cowtail
-> drink = 1
    rep(i,1,D) add(get(i,3),T,1); //drink -> T = 1

    print(dinic());
    return 0;
}
```

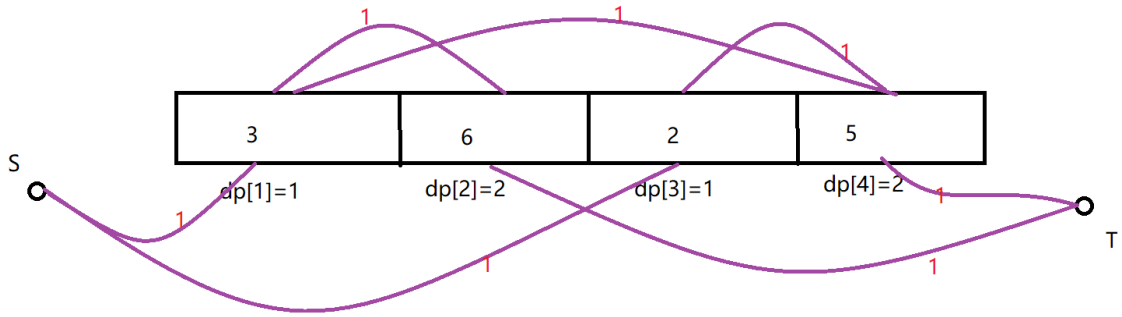
最长递增子序列问题

> 给定一个正整数序列`x1,x2,x3...`，求
>
> 1.其最长递增(非严格)子序列的长度`s`。
>
> 2.计算从给定的序列种可取出多少个长度为s的递增子序列。
>
> 3.如果允许多次使用`x1`和`xn`，求最多可取出多少个长度为s的递增子序列

第一问的话很好求解，直接DP即可，状态转移方程为：

$$dp[i] = max(dp[i], a[i] >= a[j]?dp[j] + 1 : 1)$$

巧妙的是建图：



[1]从源点 S 向所有`dp[i]=1`连一条容量为1的边

[2]从所有`dp[i]=s`向汇点 T 连一条容量为1的边

[3]由于一个点只能被使用一次，因此顺理成章我们需要拆点，即对于每个点向拆出来的点连一条边

[4]对于`j<i`若有`dp[j]==dp[i]-1`则从j向i连一条边，表示可以组成一个单调不减子序列

这样，从S->T的最大流便是可拆个数

对于第三小问只需要：

- 将源点流出和流入汇点的边全改为正无穷即可

- 将所有`dp[i]=1`和`dp[i]=s`的点与拆出来的点之间的边改为正无穷即可

代码：

```
const int INF = 1e8;
const int N = 510*2+512,M = 258000;
int n,m,S,T,k,s,ans=0;
int e[M],ne[M],f[M],h[N],idx=0;
int dp[510],a[510],q[N],cur[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 ++;
}

int work() {
    int mx = 0;
    rep(i,1,n) {
        dp[i] = 1;
        for(int j=1;j<i;j++)
            if(a[i] >= a[j]) dp[i] = max(dp[i],dp[j]+1);
        mx = max(mx,dp[i]);
    }
    return mx;
}

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;
}
//================================
int main(){
    memset(h,-1,sizeof h);
    n = read(); rep(i,1,n) a[i] = read();
    S = 0, T = n * 2 + 1;
    s = work(); print(s);
    rep(i,1,n){
        add(i,n+i,1); //由于限制f[i]拆成i和n+i两个点
        if(dp[i] == 1) add(S,i,1);  //S向f[i]=1连一条边
        if(dp[i] == s) add(i+n,T,1);
        rep(j,1,i) if(a[j] <= a[i] && dp[j] + 1 == dp[i]) add(j+n,i,1);
    }

    if(s==1) {print(n);print(n); exit(0);}  //特判
    print(ans=dinic());

    for(int i=0;i<idx;i+=2){
        int a = e[i^1], b = e[i];
        if(a == S && b == 1) f[i] = INF;
        else if(a == 1 && b == 1+n) f[i] = INF;
        else if(a == n+n && b == T) f[i] = INF;
        else if(a == n && b == 2*n) f[i] = INF;
    }
    print(ans+=dinic());

    return 0;
}
```