

## 家园

### Homeland

#### 题意及算法分析

仔细分析题意，我们不难看出本题具有一个流网络应有的各个要素：

- 源——地球：0
- 汇——月球：-1
- 中间点——太空站：1~n
- 边——往返于各个太空站之间的太空船
- 边的容量——太空船最多可容纳的人数。

所不同的是，本题还多出一个时间因素，并且是给定流量，要求最短时间。

我们面临着难题：一般的流网络无法表示两点间不同时刻的不同容量和流量。

因此，就需要对流网络进行改造：拆点——

如果最短时间为  $T$ ，我们就把每个点  $V_i$ （包括源和汇）拆成  $V_{i,0}$ 、 $V_{i,1}$ 、……、 $V_{i,T}$  这  $T+1$  个点：这样，太空船在  $t-1$  时刻从  $V_i$  出发( $t \leq T$ )， $t$  时刻到达  $V_j$  的动作就可以用边  $V_{i,t-1} \rightarrow V_{j,t}$  来表示，如果  $C(V_{i,t-1}, V_{j,t})=a$ ， $f(V_{i,t-1}, V_{j,t})=b$ ，那么就表示太空船可以容纳  $a$  个人，而此时正载有  $b$  个人；又因为每个太空站可以容纳无限的人停留，所以对于每个  $V_{i,t-1}$ ，都应有一条容量不限的边指向  $V_{i,t}$ 。

这样改造之后，源就是  $V_{0,0}$  点，汇就是  $V_{-1,t}$  点，从源到汇的流量就表示到时刻  $T$  为止，从地球到月球最多可以输送的人数。

我们从小到大地尝试每个  $T$  值，第一个能够把  $k$  个人全部送达月球的时刻就是最优解。

在算法的实现上，我们采用宽度搜索找增广路径的算法（Edmonds-Karp 算法）来求最大流，并且，每当  $T$  值加 1 时，我们只需在原有流量的基础上进行增广，而无须重新求解。

算法的空间复杂度为  $O(T \times n^2)$ 。由于  $T$  最大可达到 1000 以上（目前找到的最大值为 1159），所以在 TP7 下只保存得下流量，容量网络需要在求解过程中重建（好在并不太麻烦）。

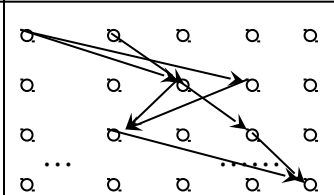
算法的时间复杂度为  $O((n \times T)^3)$ ：求解的全过程中找不到增广路径的宽度搜索次数为  $O(T)$ （因为一旦找不到增广路径， $T$  值就会加 1），找到增广路径的为  $O(|V| \times |E|)$ ，一次宽度搜索的复杂度为  $O(|E|)$ ，把它们综合起来，并代入  $|E|=(m+n) \times T$ —— $m$  表示  $m$  艘太空船某一时刻必然正从一个太空站驶往另一个， $n$  表示  $V_{i,t-1}$  指向  $V_{i,t}$  的边——和  $|V|=n \times T$ ，即得

$O(T^3 \times n \times (m^2 + n^2))$ ，从题目所给范围 ( $n \leq 20, m \leq 13$ ) 看，我们也可以把  $m$  视为  $O(n)$ ，所以时间复杂度可简写为  $O((n \times T)^3)$ 。

一个猜想

在构造测试数据的过程中，我发现最优解  $T$  随着人数  $k$  的增加而有规律地增加，即当  $k$  增加一个特定的值  $d$  时， $T$  增加一个特定的值  $t$ ，写成表达式即： $T = t \times \left\lfloor \frac{k+a}{d} \right\rfloor + b$  ( $t > 0, d > 0, a \in [0, d), b$  无限制)，其中  $t、d、a、b$  由流网络的结构和容量决定。

例如右图所示的数据，图中给出了对应的改造后的流网络（略去了边上的容量和所有  $V_{i,t-1}$  指向  $V_{i,t}$  的边），数据对应的答案列于右表，从中可以看出这个流网络对应的  $t=4、d=6、a=5、b=-1$ ，即  $T = 4 \times \left\lfloor \frac{k+5}{6} \right\rfloor - 1$ 。

4 3 2 (n=4,m=3,k=2)	0 1 2 3 -1		Answer : kT1~637~12713~ 181119~2415... ...5035
1 4 0 2 1 -1 t: 0			
5 4 1 2 3 -1 1			
5 3 0 3 1 2			
3			

上面只是我在实践中发现的一点现象，但限于理论水平不够，还无法解释或是证明这一点。不过，如果它成立的话，我们至少可以先解  $k$  值较小的情况（用时很少），求出  $t、d、a、b$  的值，再计算出  $k$  值很大时的解，或许我们仅根据输入的数据就可以直接计算了。

```
源程序及注释:
{$A+ ,B- ,D+ ,E- ,F- ,G+ ,I+ ,L+ ,N+ ,O- ,P- ,Q- ,R- ,S- ,T+ ,V- ,X+ }
{$M 16384,0,655360}
program Escape;
const
  Infns='Homeland.in';
  Outfns='Homeland.out';
  LimitM=13;
  LimitN=20;
  LimitK=50;
  LimitTime=1300; { 时间最大值 }
```

```

    LimitNode=500;    { 用于求最大流的宽度搜索算法 }
    LLimitDanger=LimitNode-50;
type
    TMap=array[0..LimitN,-1..LimitN] of byte;
    TTimeMap=array[0..LimitTime] of ^TMap;
    TStationBool=array[-1..LimitN] of boolean;
    TShip=record
        content,moor:byte;
        mooring:array[1..LimitN+2] of shortint;
    end;
    TSpaceShip=array[1..LimitM] of TShip;
    TNode=record
        delta,station:shortint;
        level:word;
    end;
    TLinearNode=array[1..LimitNode] of TNode;
    TStationInt=array[0..LimitTime,-1..LimitN] of shortint;
var
    Ship      :TSpaceShip;      { 太空船 }
    Flow      :TTimeMap;        { 记录每一时刻每条边的流量 }
    Father    :TStationInt;      { 求最大流时用于记录每点在增广路径上的父结点 }
    OpenList  :^TLinearNode;     { 求最大流时记录宽度搜索的待扩展结点 }
    n,m,k    :byte;
    pre,now,
    head,tail :word;            { 待扩展队列的头、尾指针 }

procedure Init; { 读入 }
var
    f:Text;
    i,j:byte;
begin
    assign(f,Infs); reset(f);
    readln(f,n,m,k);
    for i:=1 to m do
        with Ship[i] do
            begin
                read(f,content,moor);
                for j:=1 to moor do { 读入太空船信息 }
                    read(f,mooring[j]);
                readln(f);
            end;
        close(f);
    end;

procedure Out(totaltime:word); { 输出 }
var
    f:text;
begin

```

```

    assign(f,Outfns);rewrite(f);
    writeln(f,totaltime);
    close(F);halt;
end;

function NoAnswer:boolean;    { 判断无解情况 }
var
    i,j:byte;
    flag:boolean;
    Touch:TStationBool;
begin
    fillchar(Touch,sizeof(Touch),false);
    Touch[0]:=true;
    repeat
        flag:=true;
        for i:=1 to m do
            with Ship[i] do
                for j:=1 to moor do
                    if Touch[mooring[j]] and (not Touch[mooring[j mod moor + 1]])
                        then begin
                            Touch[mooring[j mod moor + 1]]:=true;
                            flag:=false;
                        end;
                until flag;
            NoAnswer:=not Touch[-1];
        end;
end;

function min(a,b:byte):byte;    { 求 a,b 中的较小者 }
begin
    if a<=b then min:=a
        else min:=b;
end;

procedure IncFlow(delta:shortint;level:word);
{ 根据已找到的一条增广路径进行增广: delta 为增加的流量 }
var
    i,fa:shortint;
begin
    i:=-1;  dec(k,delta);    { 剩余的 k 人中又有 delta 人得到安排 }
    repeat
        fa:=Father[level,i];
        if fa>=0
            then begin { 对正向弧进行增广 }
                inc(Flow[level-1]^[fa,i],delta);
                i:=fa;  dec(level);
            end
        else begin { 对反向弧进行增广 }
                dec(Flow[level]^[i,-fa],delta);

```

```

        i:=-fa; inc(level);
    end;
until i=0;
end;

```

```

function Open:boolean; { 扩展结点 head, open=true 表示未找到增广路径, 需要继续扩展结点 }
var
    Map:array[-1..LimitN] of shortint;
    i,which,t:shortint;
    nextlevel:word;
begin
    with OpenList^[head] do
        begin
            fillchar(Map,sizeof(Map),0);
            Map[station]:=LimitK;
            for i:=1 to m do { 求得从(level,station)点出发的所有弧的容量 }
                with Ship[i] do
                    begin
                        which:=level mod moor +1;
                        if mooring[which]=station
                            then inc(Map[mooring[which mod moor +1]],content);
                    end;
            if Map[-1]>Flow[level]^[station,-1] then { 能够增广 }
                begin
                    Father[level+1,-1]:=station;
                    IncFlow(min(delta,Map[-1]-Flow[level]^[station,-1]),level+1);
                    Open:=false; exit;
                end;
            Open:=true;
            nextlevel:=level+1;
            if level<pre then
                for i:=0 to n do { 找正向弧 }
                    if (Father[nextlevel,i]=100) and (Map[i]>Flow[level]^[station,i]) then
                        begin
                            Father[nextlevel,i]:=station;
                            inc(tail);
                            OpenList^[tail].delta:=min(delta,Map[i]-Flow[level]^[station,i]);
                            OpenList^[tail].level:=nextlevel;
                            OpenList^[tail].station:=i;
                        end;
            if level=0 then exit;
            nextlevel:=level-1;
            for i:=1 to n do { 找反向弧 }
                if (Father[nextlevel,i]=100) and (Flow[nextlevel]^[i,station]>0) then
                    begin
                        Father[nextlevel,i]:=-station;
                        inc(tail);
                        OpenList^[tail].delta:=min(delta,Flow[nextlevel]^[i,station]);
                    end;
        end;
    end;

```

```

        OpenList^[tail].level:=nextlevel;
        OpenList^[tail].station:=i;
    end;
end;
end;

procedure SearchWay; { 宽度搜索 }
var
    i,j:word;
    flag:boolean;
begin
    repeat
        fillchar(Father,sizeof(Father),100);
        head:=1; tail:=1;
        with OpenList^[1] do { 从源出发 }
            begin
                delta:=k; level:=0; station:=0;
            end;
        flag:=true;
        while (head<=tail) and flag do
            begin
                flag:=Open; inc(head);
                if tail>LimitDanger then { 队列平移: 充分利用队列空间 }
                    begin
                        move(OpenList^[head],OpenList^[1],(tail-head+1)*sizeof(TNode));
                        dec(tail,head-1); head:=1;
                    end;
            end;
        until flag or (k=0); { 找不到增广路径或是已找到最优时间 }
    end;

begin { 主程序 }
    Init;
    if NoAnswer then Out(0);
    New(OpenList);
    for now:=1 to LimitTime do { 累试每个时刻 }
        begin
            pre:=now-1;
            new(Flow[pre]);
            fillchar(Flow[pre]^,sizeof(Flow[pre]^),0); { 为流量赋初值 }
            SearchWay;
            if k=0 then Out(now);
        end;
    end.
end.

```