26.1-1 证明：在一个流网络中，将一条边分解为两条边所得到的是一个等价的网络。更形式化地说，假定流网络 G 包含边 (u, v)，我们以如下方式创建一个新的流网络 G'：创建一个新结点 x，用新的边 (u, x) 和 (x, v) 来替换原来的边 (u, v)，并设置 $c(u, x)=c(x, v)=c(u, v)$。证明：G' 中的一个最大流与 G 中的一个最大流具有相同的值。

证明:

形式化命题：下证对每个 G=(V,E)中的流，可以构造流 G'=(V',E')，使得在 G 中有对应同值的流，这个结果在 G 中最大流时应同样成立。让 f 为 G 中一个流，通过构造， $V' = V \cup \{x\}$并且$E' = (E - \{(u,v)\}) \cup \{(u,x),(x,v)\}$，以下式构造 G'中的 f'

$$f'(y,z) = \begin{cases} f(y,z) & if \ \ (y,z) \neq (u,x) and \ \ (y,z) \neq (x,v) \\ f(u,v) & if \ \ (y,z) = (u,x) \quad or \quad (y,z) = (x,v) \end{cases}$$

也就是说，根据题意，应有 f'与 f 等价，其中 f(u,v)通过顶点 x，x 只有一个输入流且来自 u，x 只有一个输出流且去往 v。

证明上述命题：

1.先证 f'满足容量限制，显然对于 E'在 V'-{u,v,x}中的每个顶点满足容量限制。对于边(u,x)和(x,v)，由于$f(u,x) = f(u,v) \leq c(u,v) = c(u,x)$和$f(x,v) = f(u,v) \leq c(u,v) = c(x,v)$，故其同样满足容量限制。

2.再证明流量守恒，对于 u，假设$u \notin \{s,t\}$，故有

$$\begin{aligned}\sum_{y \in V'} f'(u,y) &= \sum_{y \in V' - \{x\}} f'(u,y) + f'(u,x) \\ &= \sum_{y \in V - \{v\}} f(u,y) + f(u,v) \\ &= \sum_{y \in V} f(u,y) \\ &= \sum_{y \in V} f(y,u) \\ &= \sum_{y \in V} f'(y,u)\end{aligned}$$

对顶点 v，由对称性，也可以证明流量守恒特性。

对于顶点 x，

$$\begin{aligned}\sum_{y \in V'} f'(y,x) &= f'(u,x) \\ &= f'(x,v) \\ &= \sum_{y \in V'} f'(x,y)\end{aligned}$$

故 f'是 G'的一个合法流。

再证相同条件下的流的值等价。若 s 不在{u,v}内，由于命题中构造中保证 s 的输入边和输出边相同，故最大流会维持不变。

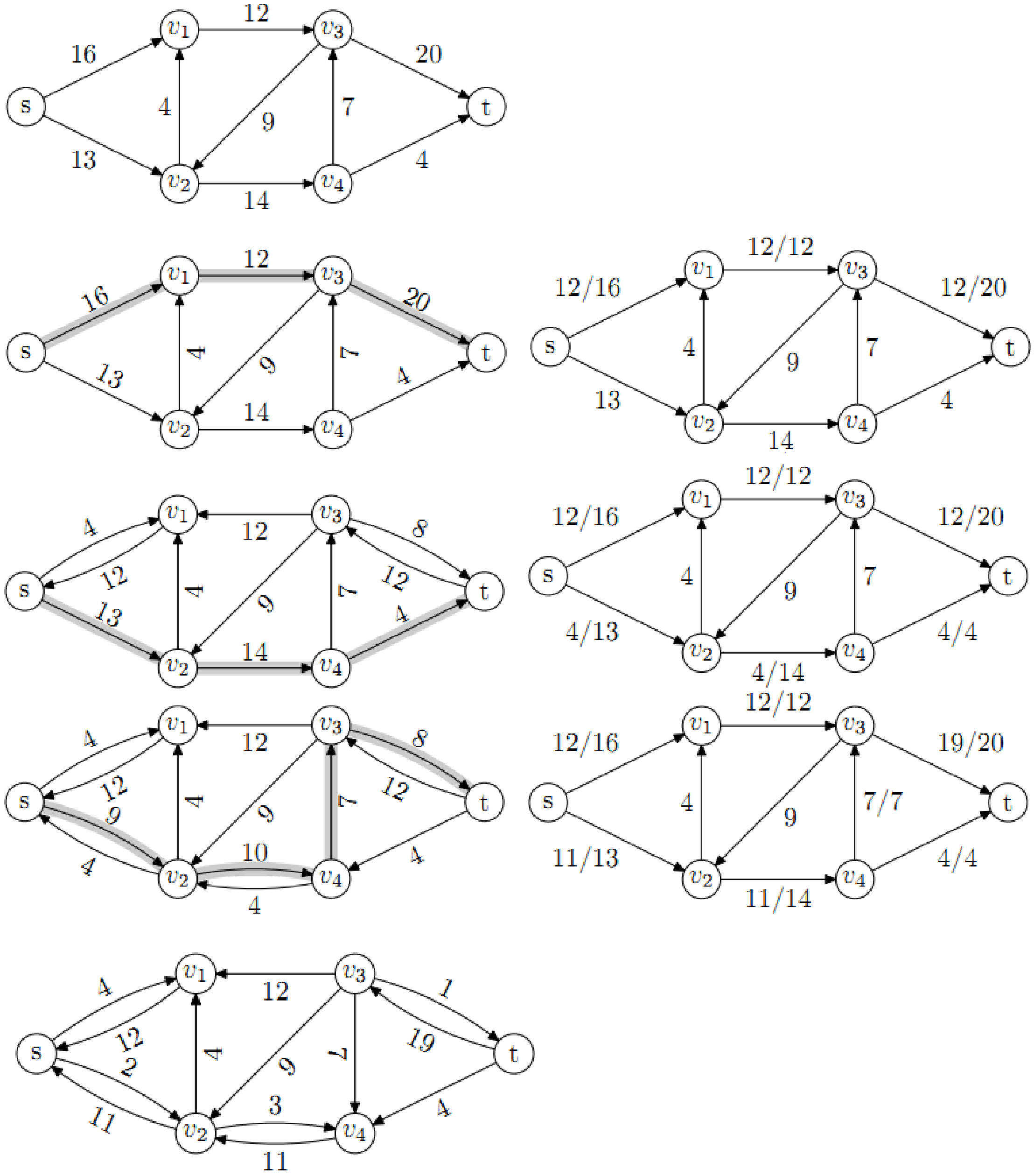
若 s=u，则有

$$\begin{aligned}|f'| &= \sum_{y \in V'} f'(u,y) \quad - \sum_{y \in V'} f'(y,u) \\ &= \sum_{y \in V' - \{x\}} f'(u,y) \quad - \sum_{y \in V'} f'(y,u) + f'(u,x) \\ &= \sum_{y \in V - \{v\}} f(u,y) \quad - \sum_{y \in V} f(y,u) + f(u,v)\end{aligned}$$

$$= \sum_{y\in V} f(u,y) \quad - \sum_{y\in V} f(y,u)$$
$$= |f|$$

当 s=v 时，由对称性同样可证，故 f'是 G'中的流且满足|f'|=|f|。

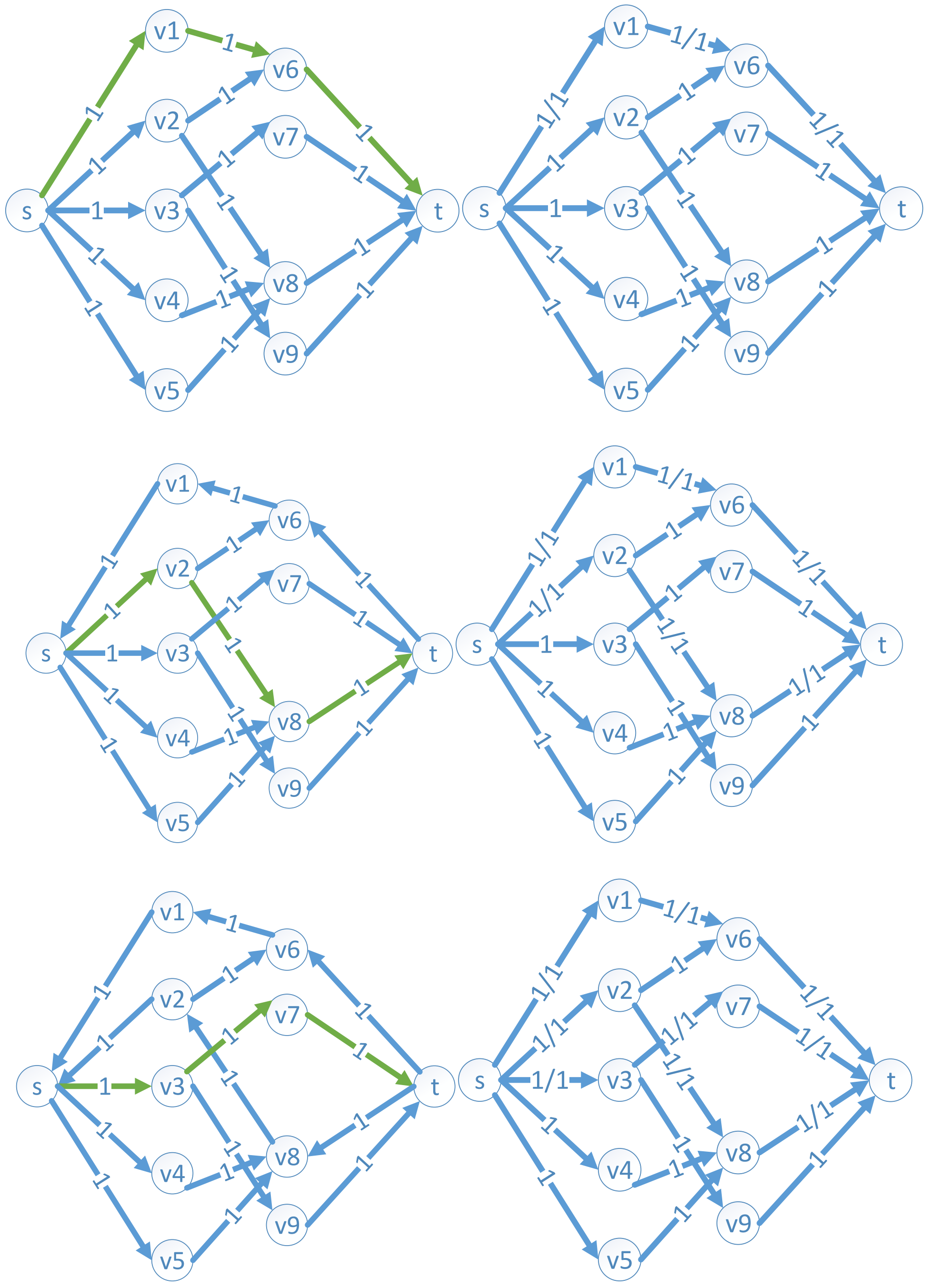
26.2-3 在图 26-1(a)所示的流网络上演示 Edmonds-Karp 算法的执行过程。



最大流为 12+4+7=23

26.3-1 在图 26-8(c)上运行 Ford-Fulkerson 算法，给出每次流量递增后的残存网络。将集合 L 中的结点从上至下编号 1～5，集合 R 中的结点从上至下编号 6～9。对于每次迭代，选择字典次序最小的增广路径。

共三次执行，每次执行时的残存网络及对应的流网络如下图：



三次执行的增广路径依次为 sv1v6t、sv2v8t、sv3v7t。

8.7 分派问题一般陈述如下：给 n 个人分派 n 件工作，把工作 j 分配给第 i 个人的成本为 COST(i,j)。设计一个回溯算法，在给每个人分派一件不同工作的情况下使得总成本最小。

解：

给每个人、每件工作均按 1,2,…,n 的顺序编号,从而问题的解可用 n 元组 X[1..n] 表示问题的解，X[i]是第 i 个人完成的工作的编号。X 是 n 个数的排列，总共有 n!个元组。

记 totalcost 为目前获得的最小总成本，初始值 totalcost=+∞

curcost 是计算过程中已经发生的成本。

检索：从第一个人开始，按序搜索。对当前正在考虑的第 k 个人，在前面 k-1 个人完成相应工作的基础上,看第 k 个人能做剩下的哪些工作,相应成本是多少,然后继续搜索第 k+1 人…等其他人的工作，构造深度优先搜索过程。

剪枝条件：对当前正在考虑的第 k 个人，若当前考虑完成作业 j，1)j 作业以前没人做，2）若有 curcost+cost(k,j)>totalcost，即当前部分解成本已经大于已知的最好成本，则剪枝，否则深度优先搜索。

```
backtrack_job(k)
    for j← 1 to n do
        if J(j) =0 then //目前还没人做的作业。对之前已经有人做的作业，就不再考虑了
            if curcost +cost(k,j)< totalcost then //如果 curcost+cost(k,j)>
                                                  //totalcost，剪枝，不再继续 k
                                                  //以后的搜索
                curcost = curcost +cost(k,j);
                X(k)=j  //构造部分解
                J(j)=1;  //设置 j 作业被选用状态
                if k=n then  //如果已经获得问题的一个解，判断是否更优
                    if curcost<totalcost then
                        Y = X;  //保存当前最好解
                        totalcost = curcost
                    endif
                else backtrack_job(k+1) //否则，继续搜索，此时 curcost 等做了修正
                X(k)=0  //回到本层，恢复初始状态，以便于搜索其他作业
                J(j)=0
                curcost = curcost - cost(k,j)
            Endif
         endif
      repeat
end
procedure Job()  //总控程序
    global totalcost = +∞;
    global curcost = 0;
    X←0
    Y←0 //记录最优解
    Backtrack_job(1);//backtrack_job 退出后，Y 就是最优解，totalcost 是相应
```

```
                //的代价
    print Y, totalcost
end
```

8.8 设 W=(5,7,10,12,15,18,20)和 M=35,使用过程 SUMOFSUB 找出 W 中使得和数等于 M 的全部子集并画出所生成的部分状态空间树。