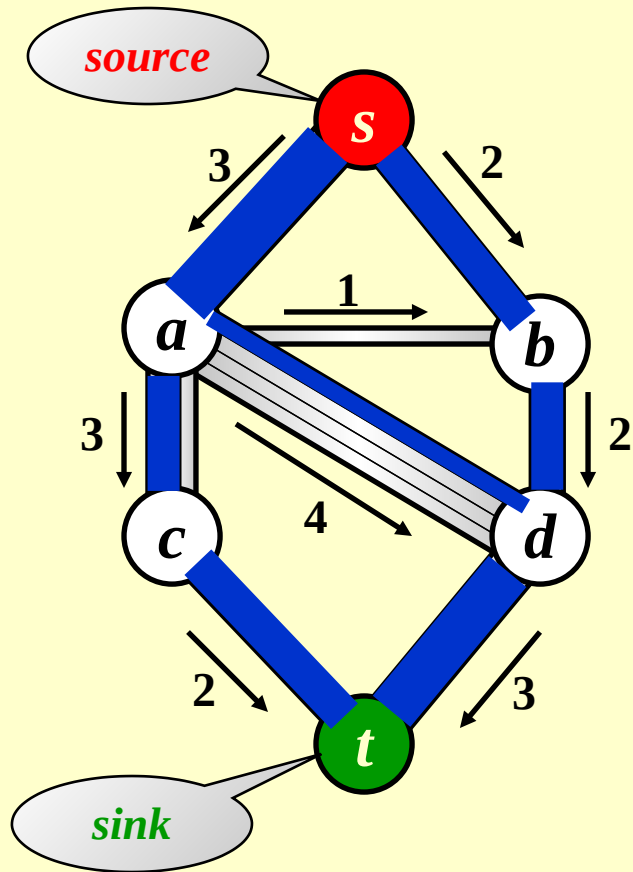
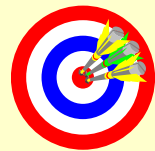


## §4 Network Flow Problems

[ Example ] Consider the following network of pipes:

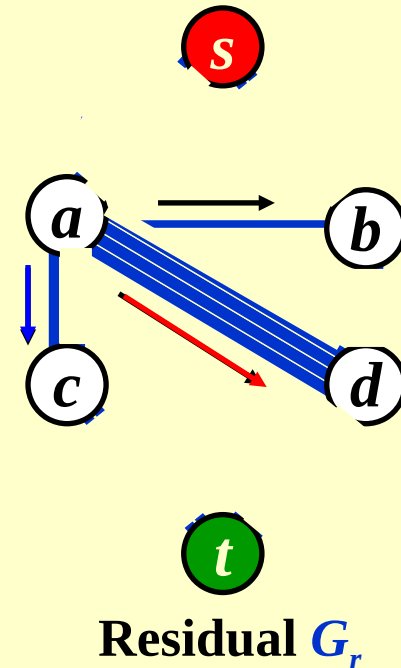
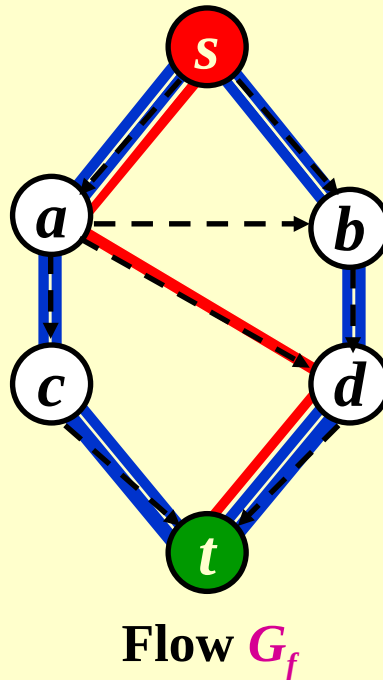
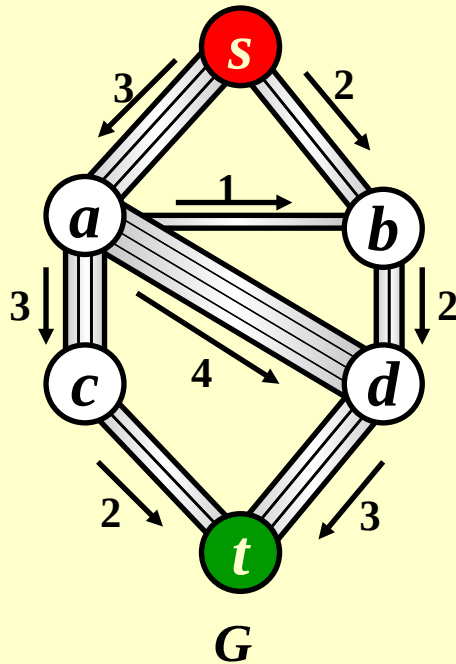


**Note:** Total coming in ( $v$ )  
 $\equiv$  Total going out ( $v$ )  
where  $v \notin \{s, t\}$



Determine the maximum amount of flow that can pass from  $s$  to  $t$ .

# 1. A Simple Algorithm



**Step 1:** Find any path  $s \rightarrow t$  in  $G_r$ ;

**Step 2:** Take the minimum edge on this path as the amount of flow and add to  $G_f$ ;

**Step 3:** Update  $G_r$  and remove the 0 flow edges;

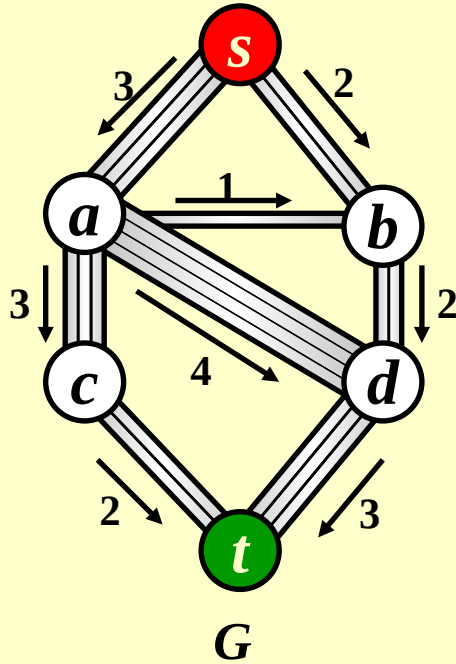
**Step 4:** If (there is a path  $s \rightarrow t$  in  $G_r$ )

Goto **Step 1**;

Else

End.

augmenting path



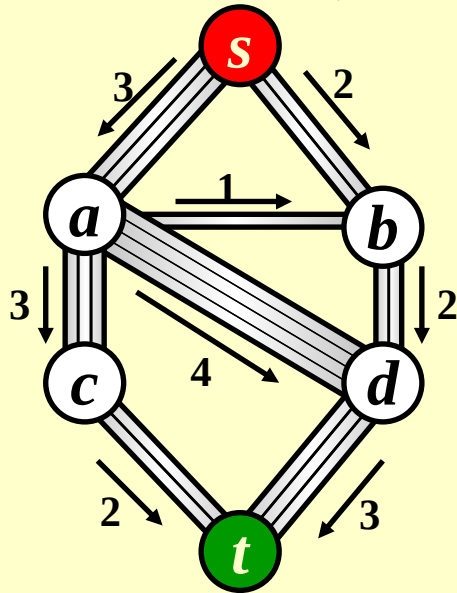
Uh-oh...  
Seems we cannot be  
*greedy* at this point.



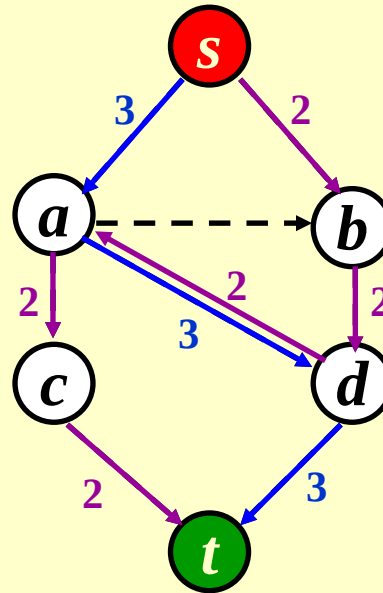
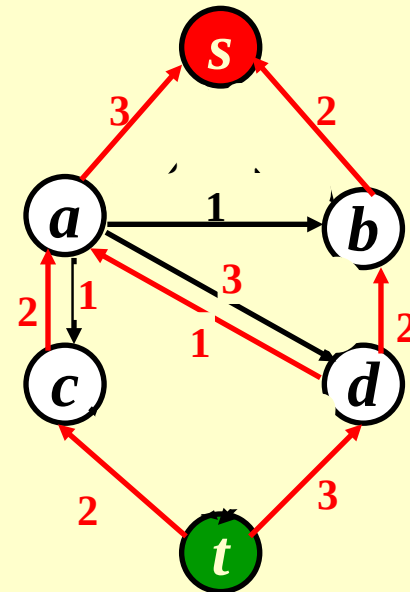
## 2. A Solution – allow the algorithm to **undo** its decisions



For each edge  $(v, w)$  with flow  $f_{v,w}$  in  $G_f$ , add an edge  $(w, v)$  with flow  $f_{v,w}$  in  $G_r$ .



G

Flow  $G_f$ Residual  $G_r$ 

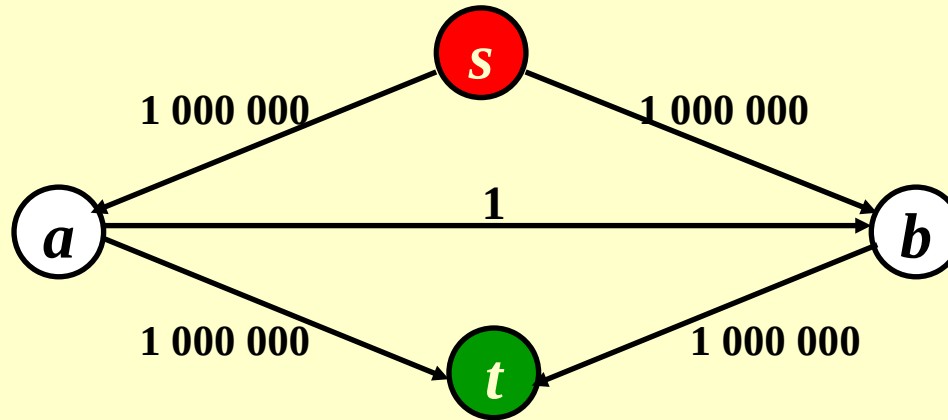
**Proposition** If the edge capabilities are **rational numbers**, this algorithm always terminate with a maximum flow.

**Note:** The algorithm works for  $G$  with ***cycles*** as well.

### 3. Analysis ( If the capacities are all integers )

👉 An augmenting path can be found by an unweighted shortest path algorithm.

$T = O( f \cdot |E| )$  where  $f$  is the maximum flow.



□ Always choose the augmenting path that allows the largest increase in flow. */\* modify Dijkstra's algorithm \*/*

$$\begin{aligned}
 T &= T_{\text{augmentation}} * T_{\text{find a path}} \\
 &= O( |E| \log \text{cap}_{\max} ) * O( |E| \log |V| ) \\
 &= O( |E|^2 \log |V| ) \text{ if } \text{cap}_{\max} \text{ is a small integer.}
 \end{aligned}$$

□ Always choose the augmenting path that has the least number of edges.

$$\begin{aligned}
 T &= T_{\text{augmentation}} * T_{\text{find a path}} \\
 &= O(|E|) * O(|E| \cdot |V|) \text{ /* unweighted shortest path algorithm */} \\
 &= O(|E|^2 |V|)
 \end{aligned}$$

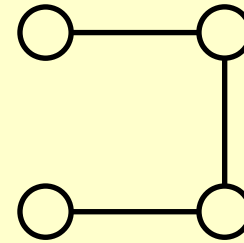
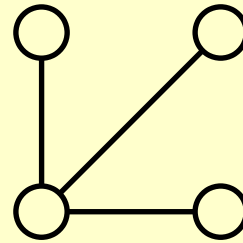
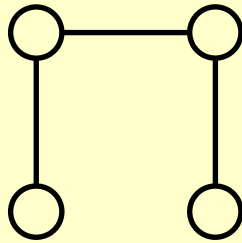
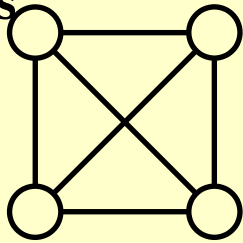
**Note:**

- If every  $v \notin \{s, t\}$  has either a single incoming edge of capacity 1 or a single outgoing edge of capacity 1, then time bound is reduced to  $O(|E| |V|^{1/2})$ .
- The *min-cost flow* problem is to find, among all maximum flows, the one flow of minimum cost provided that each edge has a cost per unit of flow.

## §5 Minimum Spanning Tree

**[ Definition ]** A *spanning tree* of a graph  $G$  is a *tree* which consists of  $V(G)$  and a *subset of  $E(G)$*

**[ Example ]** A complete graph and three of its spanning trees



### Note:

- The minimum spanning tree is a *tree* since it is acyclic -- the number of edges is  $|V| - 1$ .
- It is *minimum* for the total cost of edges is minimized.
- It is *spanning* because it covers every vertex.
- A minimum spanning tree exists iff  $G$  is *connected*.
- Adding a non-tree edge to a spanning tree, we obtain a *cycle*.

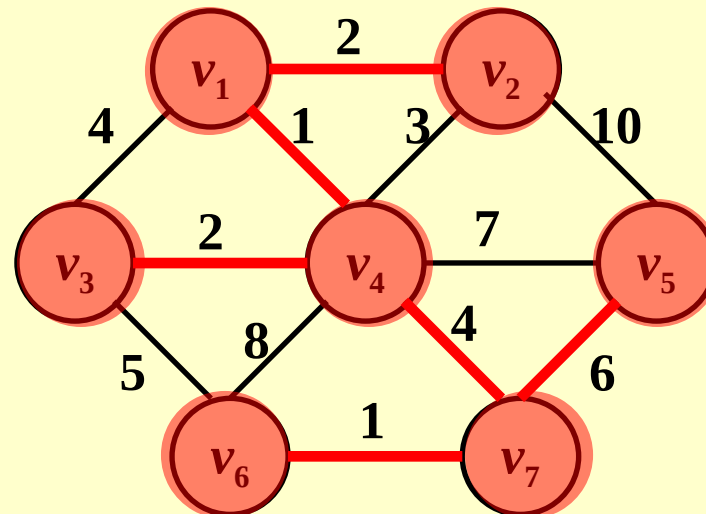


Make the best decision for each stage, under the following constraints :

- (1) we must use only edges within the graph;
- (2) we must use exactly  $|V| - 1$  edges;
- (3) we may not use edges that would produce a cycle.

## 1. *Prim's Algorithm* – grow a tree

*/\* very similar to Dijkstra's algorithm \*/*





## 2. *Kruskal's Algorithm* – maintain a forest

```

void Kruskal ( Graph G )                                 $T = O( |E| \log |E| )$ 
{
    T = { } ;
    while ( T contains less than |V| -1 edges
            && E is not empty ) {
        choose a least cost edge (v, w) from E ;    /* DeleteMin */
        delete (v, w) from E ;
        if ( (v, w) does not create a cycle in T )
            add (v, w) to T ;    /* Union / Find */
        else
            discard (v, w) ;
    }
    if ( T contains fewer than |V| -1 edges )
        Error ( "No spanning tree" ) ;
}

```

A more detailed pseudocode is given by Figure 9.58 on p.321