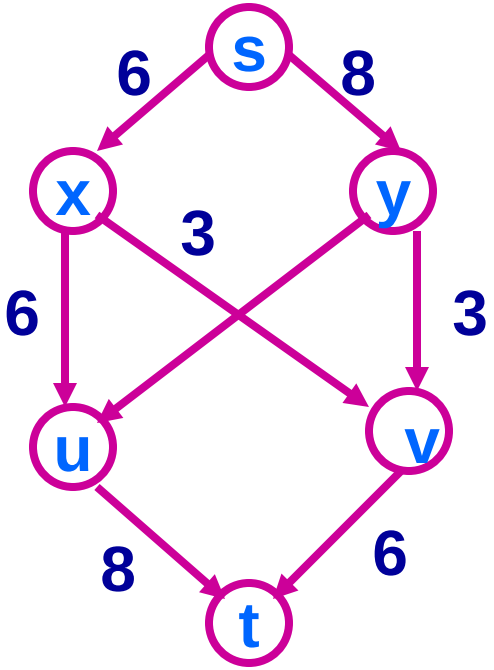
Maximum Flow

Huang Shell Ying



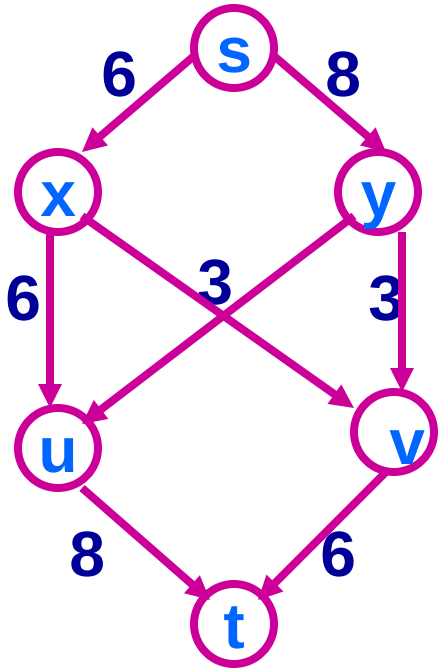
Introduction to Algorithms. **Cormen, T.H., C.E. Leiserson. R.L. Rivest, Chapter 27**

Network Flow and Ford-Fulkerson Method

(Sections 27.1 & 27.2 of Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest)

- A **flow network** is a weighted, directed graph G = (V, E) with two specially marked nodes, the ***source*** s and the ***sink*** t.

- Each edge has a nonnegative ***capacity*** c(u,v) ≥ 0.



- If (u,v) ∉ E, assume c(u,v) = 0

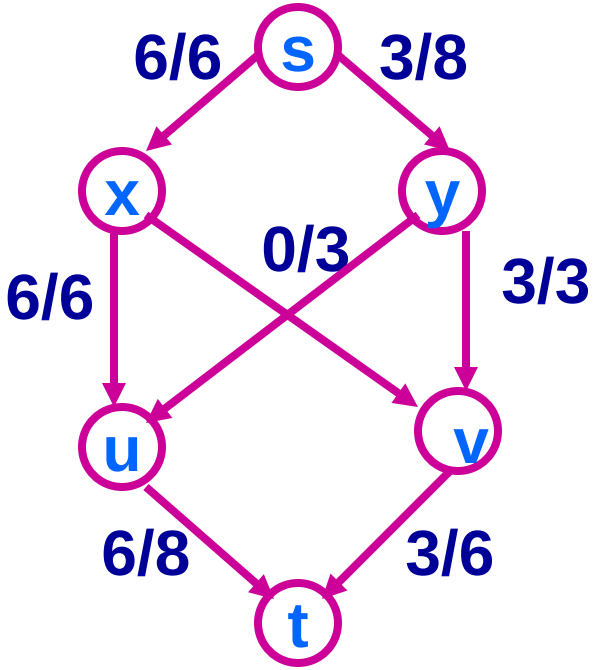
- **Assume that every vertex lies on some path from s to t.**
- **The flow network models a material flow problem where the source produces the material at some steady rate and the sink consumes the material at the same rate.**
- **The material can be water in pipes, parts through assembly lines, current through electrical networks, information through communication networks and so on.**
- **Each edge in the flow network is a conduit for the material. Each conduit has a stated capacity, which is the maximum rate the material can flow through the conduit.**

- **A flow in G is a real-valued function f : V x V → R that satisfies**

  (i) (capacity constraint) f(u,v) ≤ c(u,v) for each edge (u,v);

  (ii) the total flow into the sink t equals the total flow out of the source s, i.e.

$$\sum_{(s,v)\in E} f(s,v) = \sum_{(u,t)\in E} f(u,t)$$



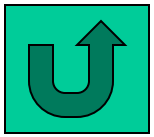
(iii) (flow conservation) for any intermediate vertex x , the total into x equals the total flow out of x, i.e

$$\sum_{(u,x)\in E} f(u,x) = \sum_{(x,v)\in E} f(x,v)$$

(iv) (Skew summetry) For all u, v ∈ V, we have

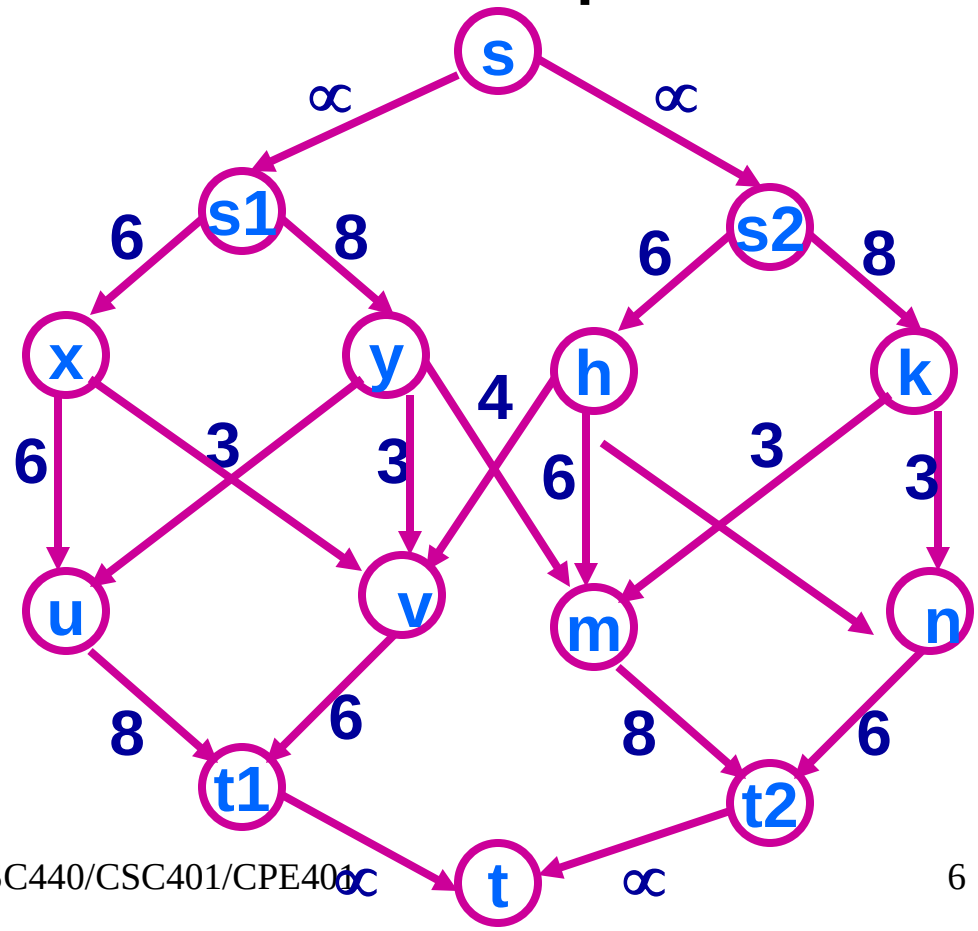
f(u,v) = - f(v,u)

- **The value of a flow is defined as $|f| = \sum_{v\in V} f(s,v)$ i.e. the total net flow out of the source.**



In the maximum-flow problem, we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value from s to t.

- **A maximum-flow problem may have several sources and sinks, rather than just one of each.**
- **The problem of determining a maximum flow in a network with multiple sources and multiple sinks can be reduced to an ordinary maximum flow problem. A supersource and a supersink are added to the network.**

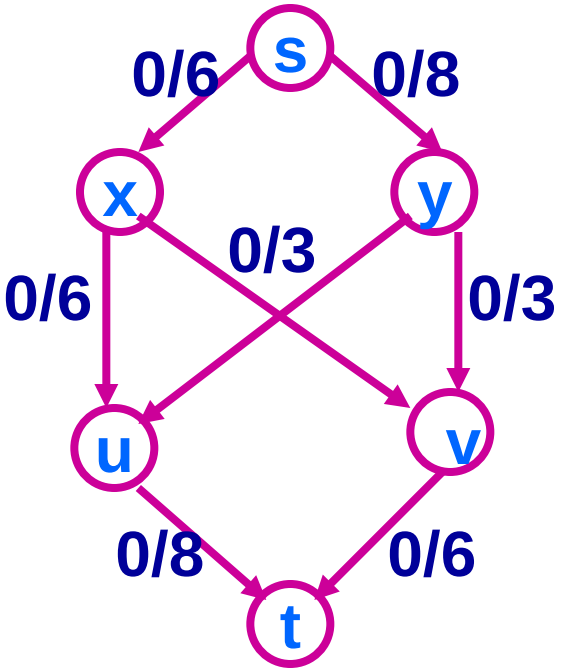


The Ford-Fulkerson Method

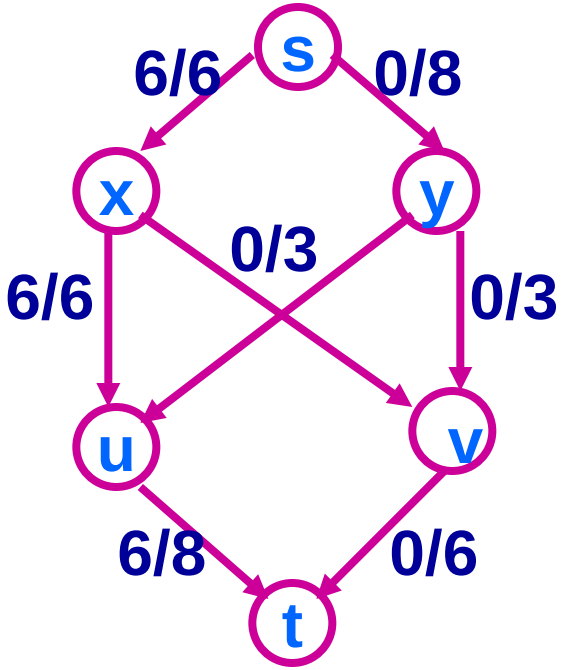
```
Ford-Fulkerson-Method(G, s, t)
    initialize flow f to 0;
    while (there exists an augmenting path p)
            augment flow f along p
    return f
```

- The method is iterative
- Start with f(u,v) = 0 for all u, v $\in$ V
- At each iteration, we increase the flow value by finding an "augmenting path", which is simply a path from s to t along which we can push more flow, and then augmenting the flow along this path.

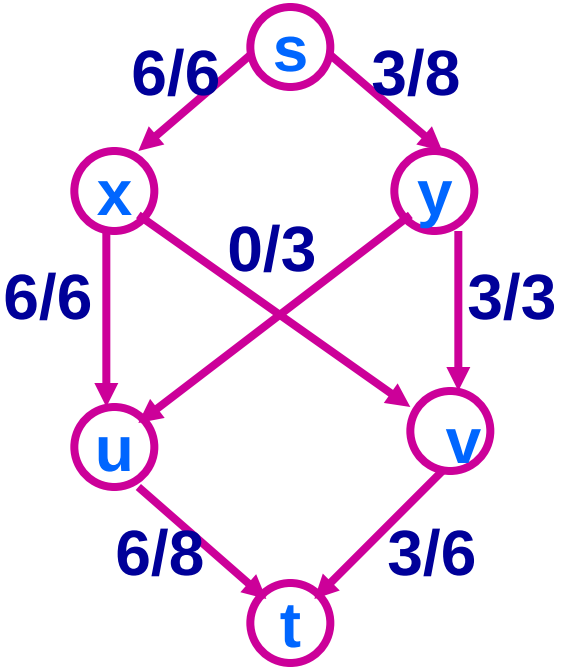
- **Repeat this process until no augmenting path can be found**
- **Upon termination, a maximum flow is found**



After initialization



Iteration 1



Iteration 2

Residual Networks

- **Intuitively, given a flow network and a flow, the residual network consists of edges that can admit more net flow**
- **The amount of additional flow we can push from a vertex u to a vertex v before exceeding the capacity c(u,v) is the residual capacity of (u,v), given by**

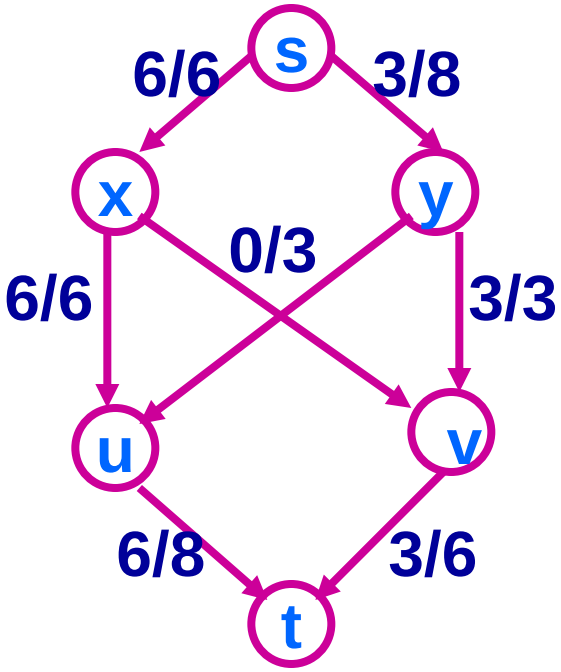
$$c_f(u,v) = c(u, v) - f(u, v)$$

- **Given a flow network G = (V,E) and a flow f, the residual network of G induced by f is $G_f = (V, E_f)$, where**

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

Each (u,v) in E_f is a residual edge.

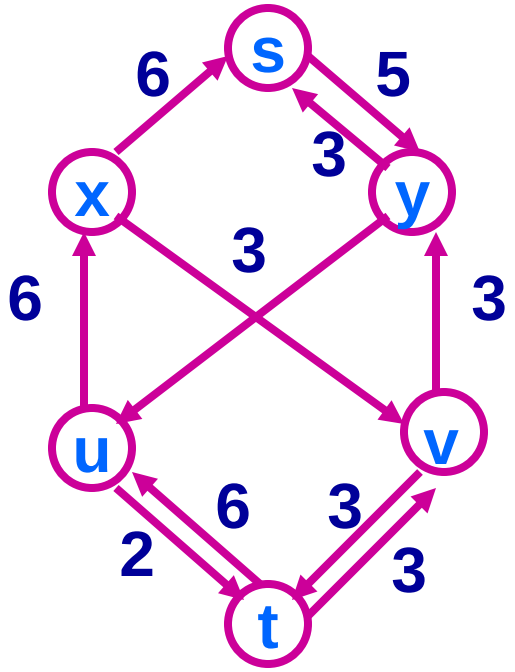
- **Note that (u,v) may be a residual edge even if it is not an edge in E. That is, in general, E_f is not a subset of E.**



The original flow

Slide 40

Slide 41

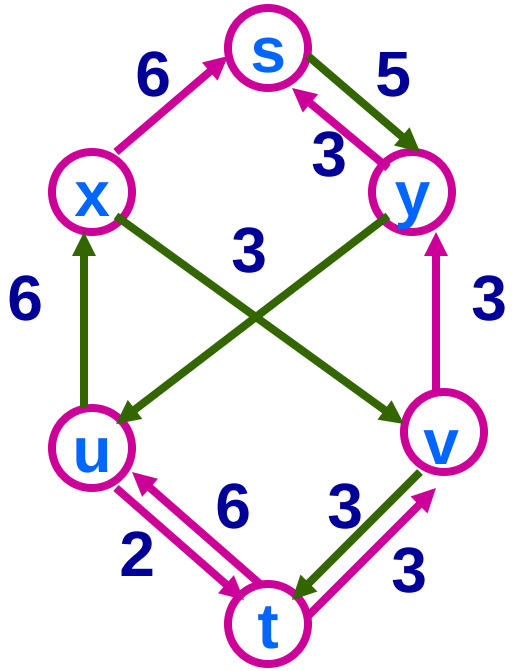


Its residual network

Augmenting Paths

- Given a flow network G = (V,E) and a flow f, an augmenting path p is a simple path from s to t in the residual network G_f .
- The maximum amount of net flow that we can push along the edges of an augmenting path p is the residual capacity of p, given by

$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is on } p\}$$

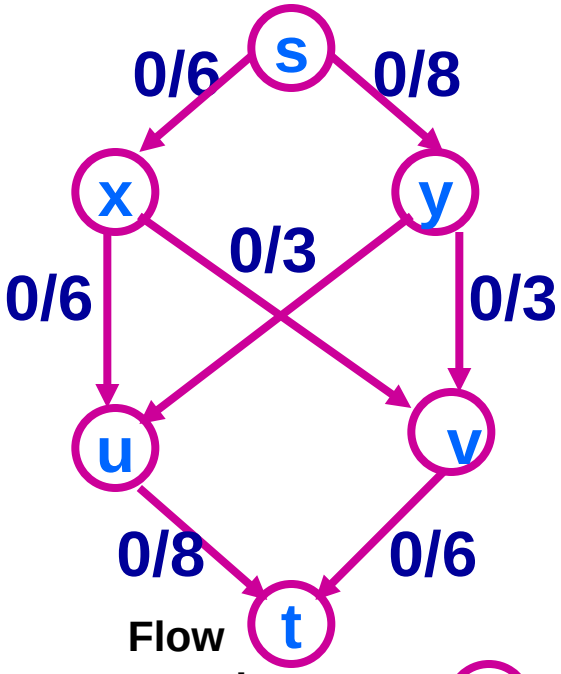


Expand the Ford-Fulkerson Method earlier on, we get:

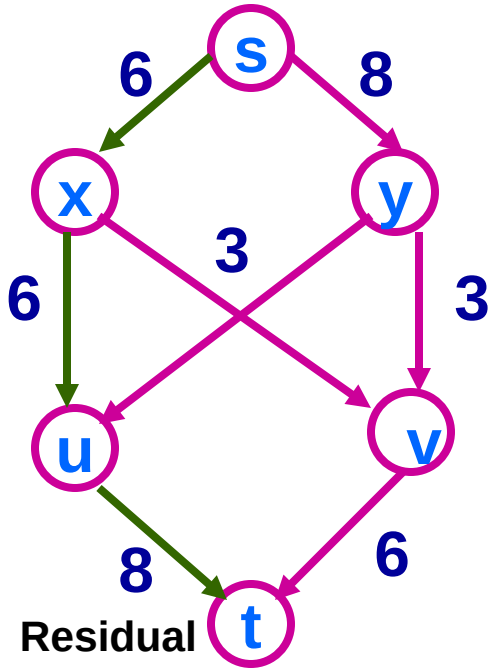
```
Ford-Fulkerson(G, s, t)
   for each edge∈ E          // G = (V,E)
         {  f[u][v] = 0;  f[v][u] = 0; }
   while (there exists an augmenting path p from
                s to t in the residual network Gf) {
      flowIncr = min{cf(u,v) : (u,v) in p}
      for each edge (u,v) in p {
            f[u][v] = f[u][v] + flowIncr;
            f[v][u] = -f[u][v]; }
      }
```

O(E) each iteration

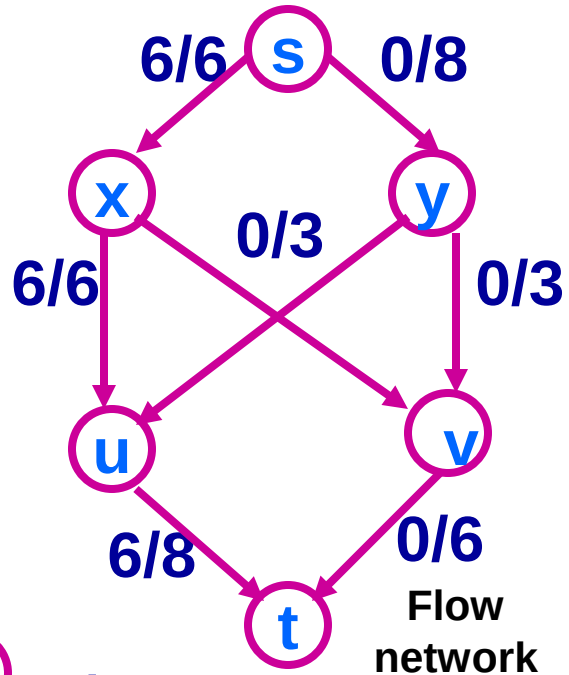
O(|f*|) iterations



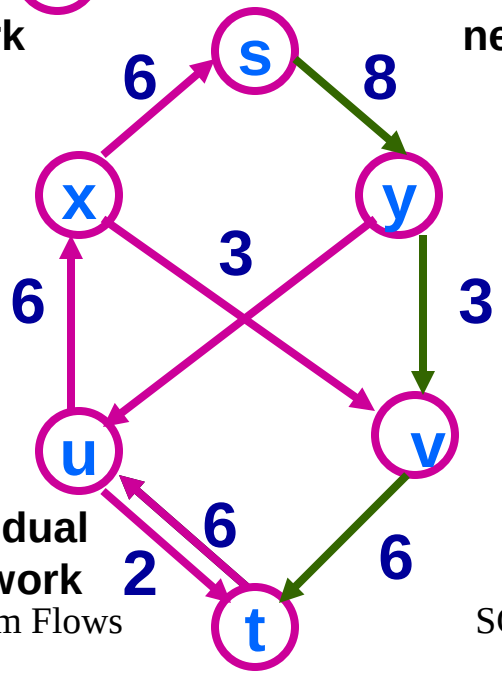
Flow network



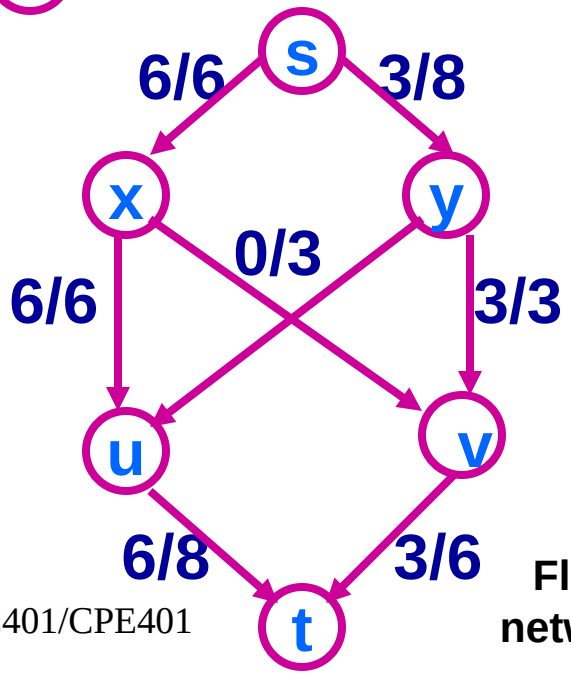
Residual network



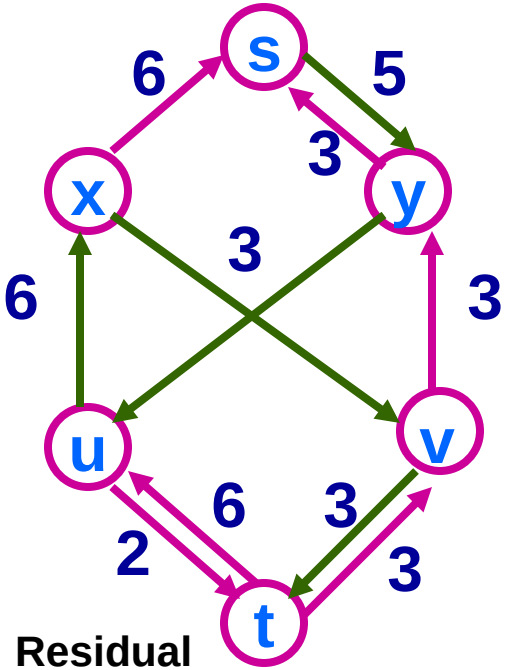
Flow network

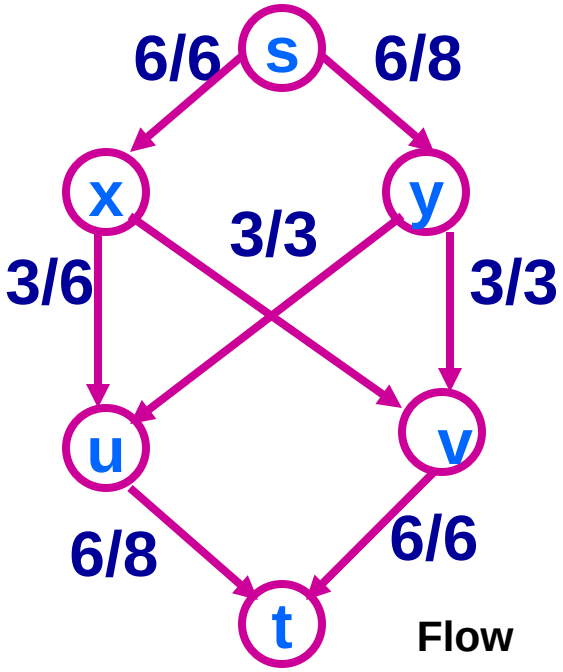


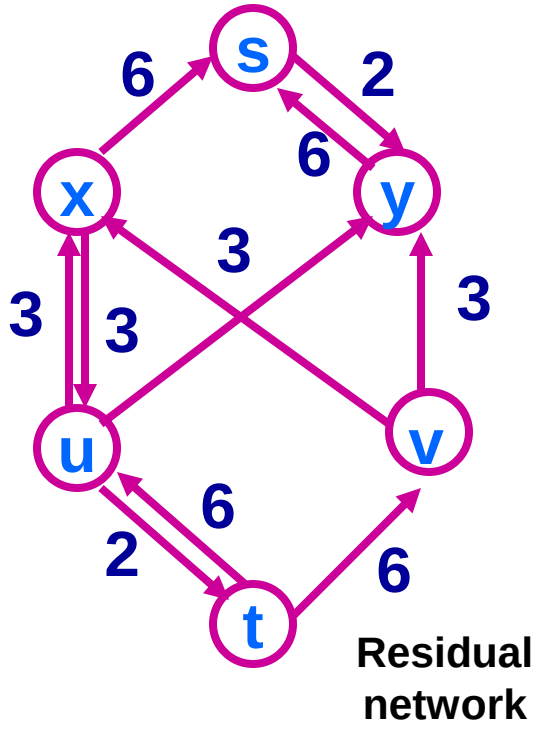
Residual network



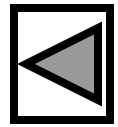
Flow network



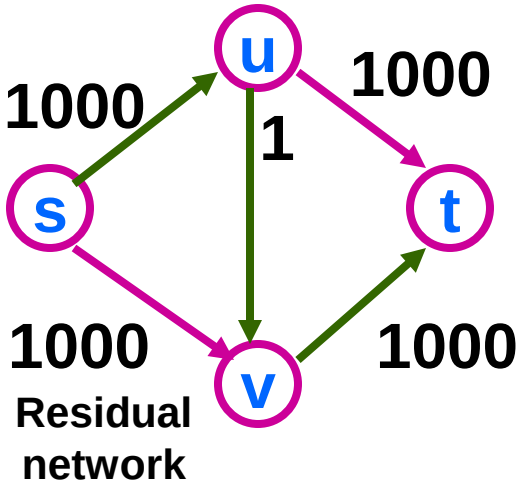


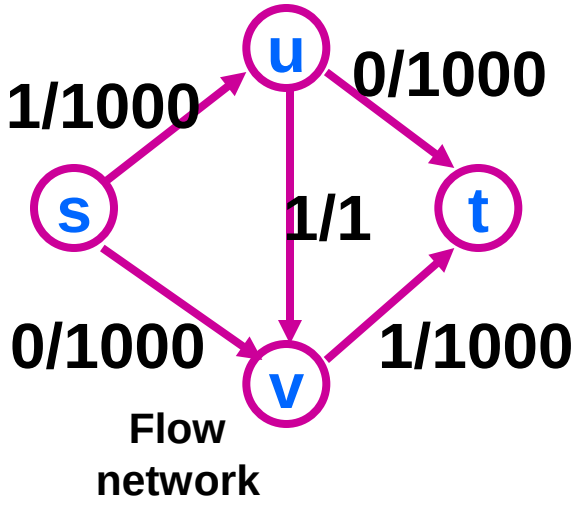


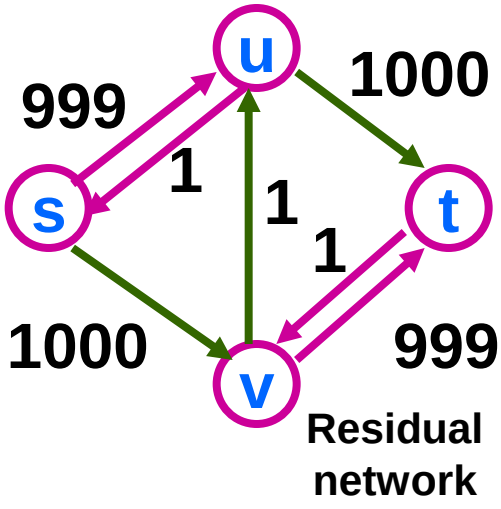
No more augmenting path
Max flow = 12

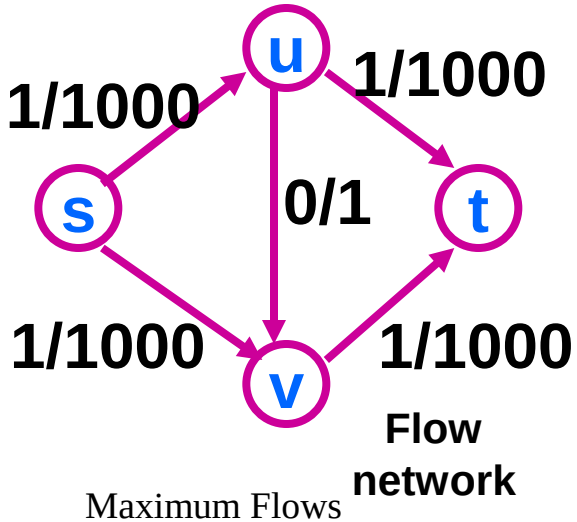


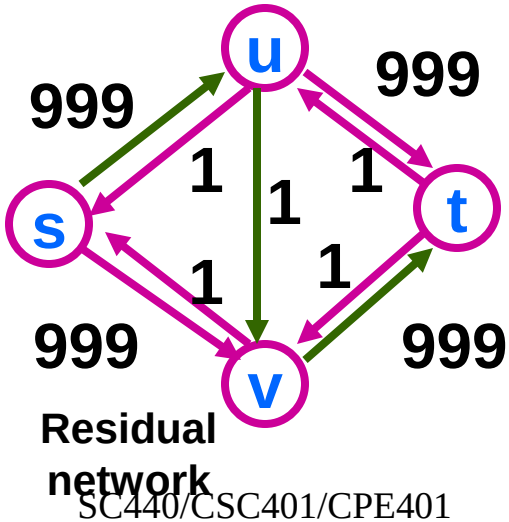
- **A straightforward implementation of Ford-fulkerson runs in time O(E|f*|). f* is the maximum flow. Example:**











We need to find 2000 augmenting paths before finding the maximum flow

- **The bound of O(E|f*|) can be improved if we use breadth-first-search to find the augmenting path p which is a *shortest* path from s to t in the residual network, where each edge has unit distance (weight). The Ford-Fulkerson method so implemented is called the *Edmonds-Karp algorithm*.**
- **Theorem If the Edmonds-Karp algorithm is run on a flow network G = (V,E) with source s and sink t, then the total number of flow augmentations performed by the algorithm is at most O(VE).**
- **The total running time of the Edmonds-Karp algorithm is $O(VE^2)$.**

Exercise

For the network in Figure Q1, find the maximum flow from source *s* to sink *t* by the Ford-Fulkerson method:

1. **each augmenting path is obtained by DFS on the residual network**
2. **each augmenting path is obtained by BFS on the residual network.**

Assume the neighbours of a vertex are explored in alphabetical order in the search for an augmenting path. Clearly show the revised flow network, the residual network and how each augmenting path is chosen.

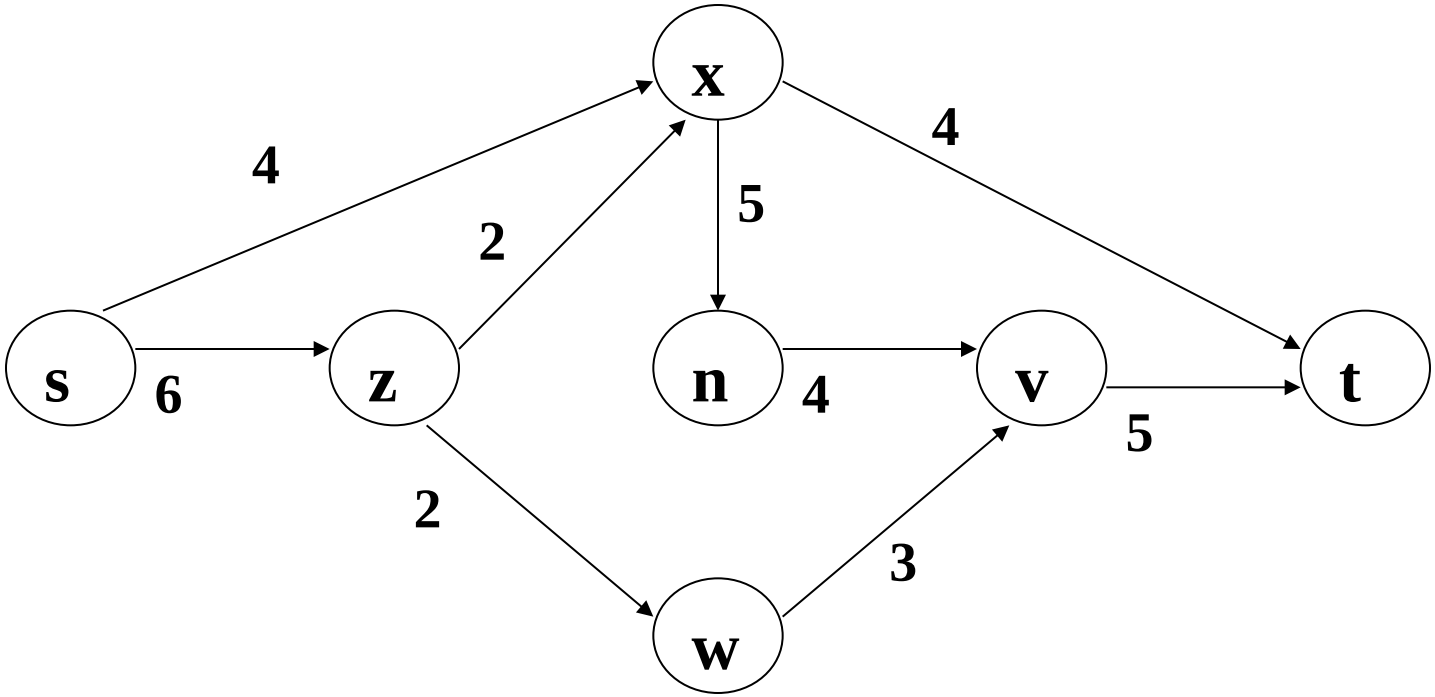
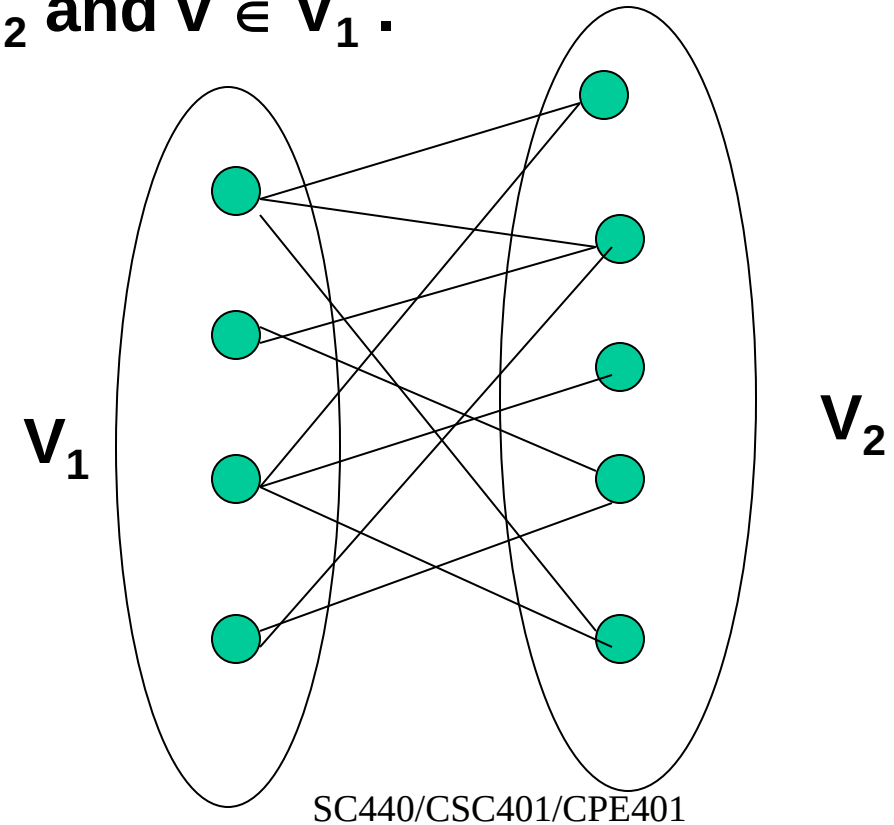


Figure Q1

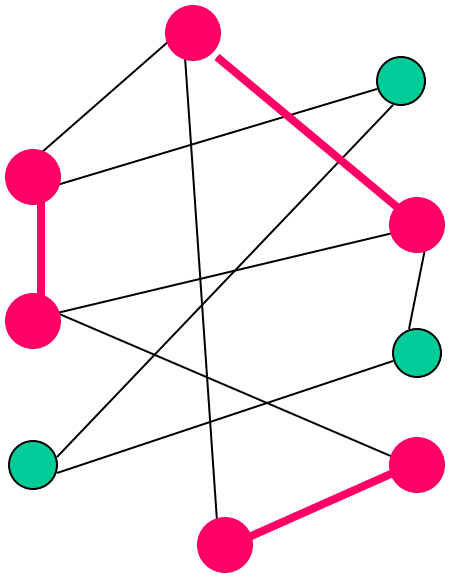
Maximum bipartite matching

- A <u>bipartite</u> graph is an undirected graph G = (V,E) in which V can be partitioned into two sets V_1 and V_2 such that $(u,v) \in E$ implies $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$.



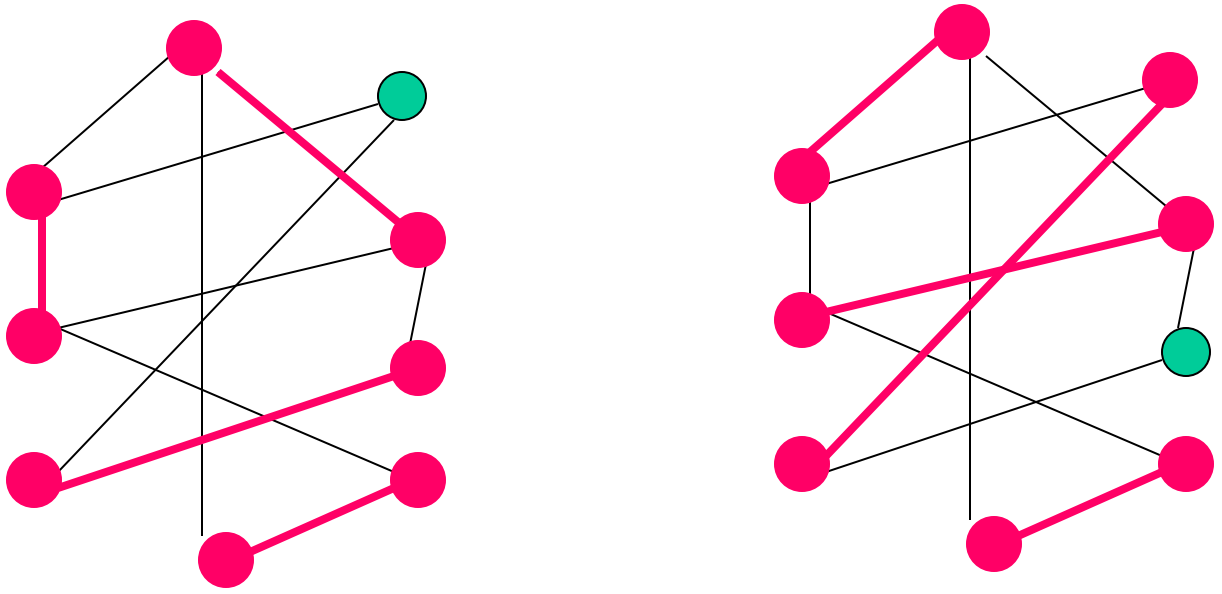
- **Given an undirected graph G = (V,E), a <u>matching</u> is a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is incident on v.**

A vertex v is <u>matched</u> by matching M if there is an edge in M that is incident on v. Otherwise v is unmatched.



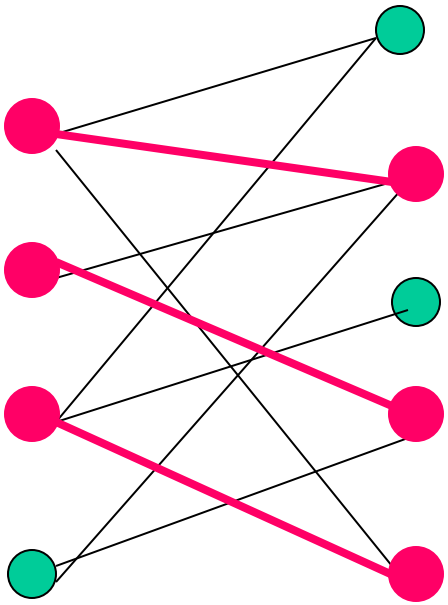
G = (V, E)

- **A <u>maximum matching</u> is a matching of maximum cardinality, that is, a matching M such that for any matching M', we have $|M| \geq |M'|$.**

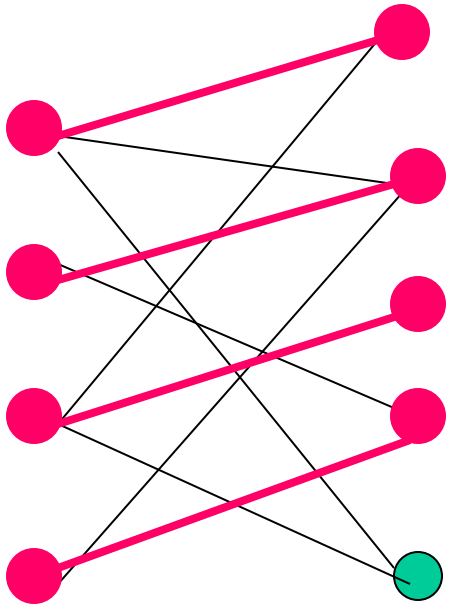


Two possible maximum matchings for G

- **Our problem is to find a maximum matching in a bipartite graph.**

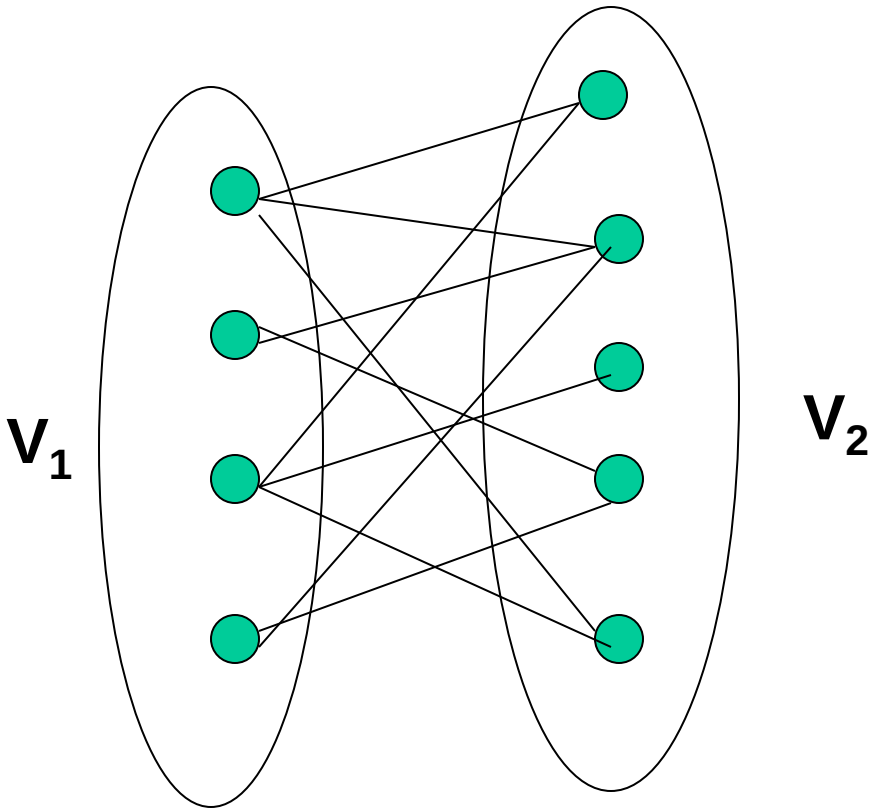


A matching, but not a maximum matching



A maximum matching

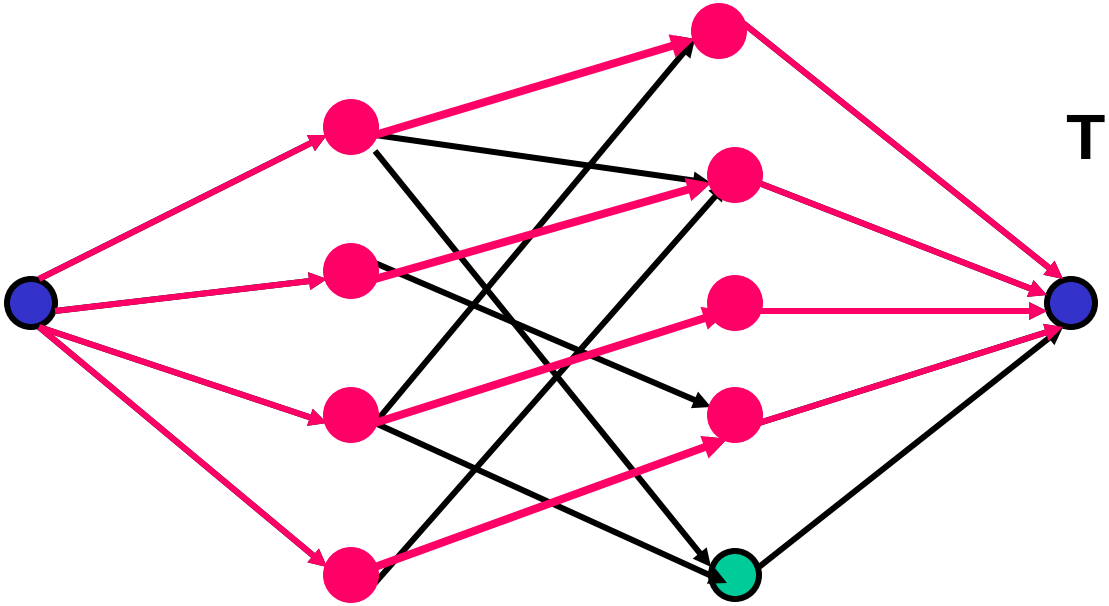
- **Applications of the maximum matching in a bipartite graph include allocating operators to machines such that maximum number of machines are operated by operators simultaneously.**



V_1 is the set of operators and V_2 is the set of machines

The edges represents which machines each operator is capable of operating

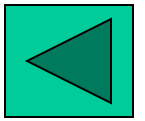
- **We can use the Ford-Fulkerson method to find a maximum matching in an undirected bipartite graph G = (V, E).**
- **First construct a flow network G' = (V', E'). Each edge has unit capacity.**



The graph G'

- **Then find the maximum flow f in G'. $|M| = |f|$.**

Feasible Flow Problem



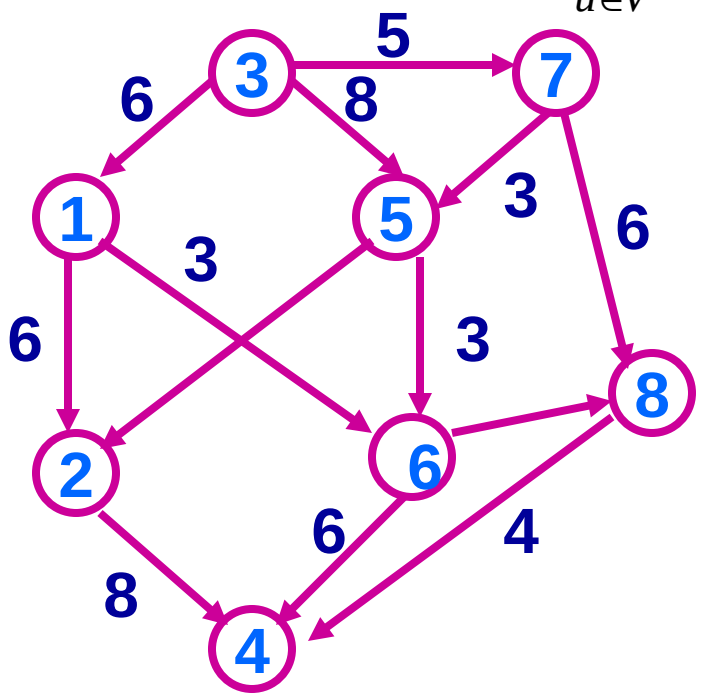
The problem:

Suppose merchandise is available at some seaports and is desired by other ports. We know the stock of merchandise available at the ports, the amount required at the other ports, and the maximum quantity of merchandise that can be shipped on a particular sea route. We wish to know whether we can satisfy all the demands by using the available supplies.

The constraint: for u $\in$ V,

$$\sum_{(u,x)\in E} f(u,x) - \sum_{(y,u)\in E} f(y,u) = b(u)$$

- **b(u) > 0 : u is a sea port with some merchandise;**
- **b(u) < 0 : u is a sea port that needs some merchandise;**
- **b(u) = 0 : u is a transit sea port.**
- **We assume that** $\sum_{u \in V} b(u) = 0$

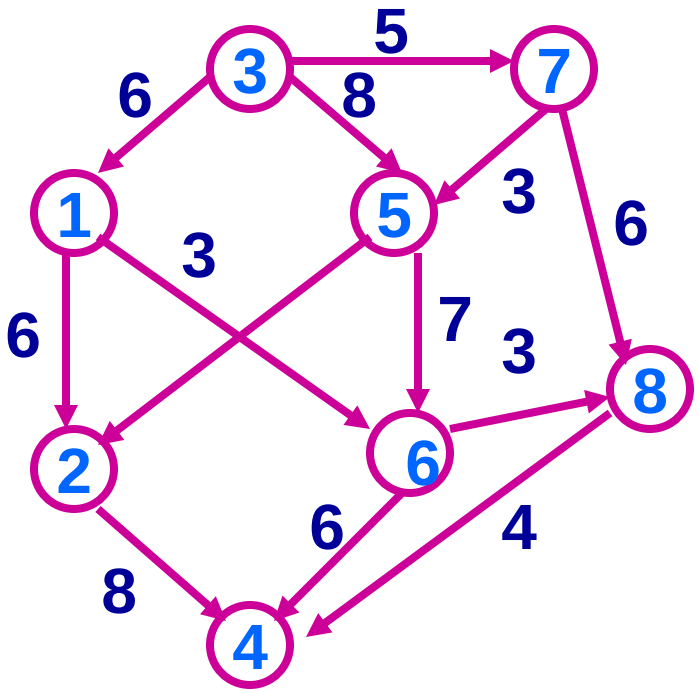


For example:
b(7) = 6,
b(3) = 18,
b(8) = - 12
b(4) = -12
Other b(u) = 0

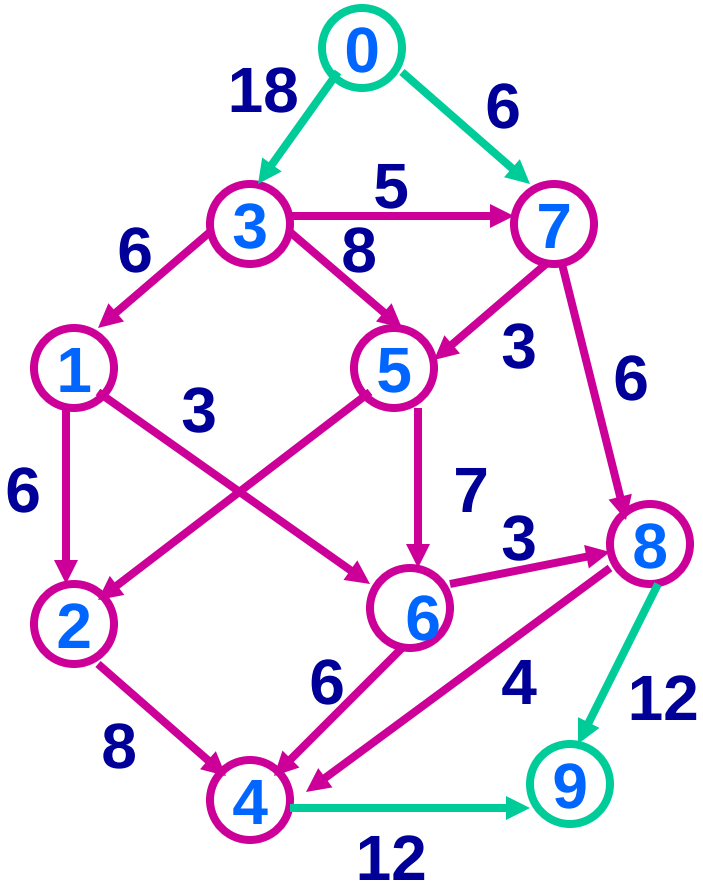
Transform into a Maximum Flow Problem

- **Define an augmented network:**
  - **Add two nodes, a source node and a sink node**
  - **For each node u with b(u) > 0, add an arc from source to u with capacity b(u)**
  - **For each node u with b(u) < 0, add an arc from u to sink with capacity –b(u)**
- **This is the transformed network.**
- **We solve a maximum flow problem from the source to the sink in the transformed network.**
- **If the maximum flow saturates all source and sink arcs, the original Feasible Flow Problem has a feasible solution; otherwise, it is infeasible.**

- **Example**



b(7) = 6, b(3) = 18,
b(8) = - 12 b(4) = -12
Other b(u) = 0



The transformed network

Verification that the algorithm works

- **We are to show that the original network contains a feasible flow if and only if the transformed network contains a flow that saturates all source and sink links.**
- **If the original network has a feasible flow x, the same flow with x_{si} = b(i)for each source arc (s, i) and x_{it} = -b(i) for each sink arc (i, t) is a maximum flow in the transformed network.**
- **If x is a maximum flow in the transformed network, this flow in the original network satisfies the constraints in the feasible flow problem.**

Scheduling on Uniform Parallel Machines

- **We consider the problem of scheduling of a set J of jobs on M uniform parallel machines.**
- **Each job $j \in J$ has a processing requirement p_j (the number of machine days required to complete the job),**
- **a release date r_j (representing the beginning of the day by which the job becomes available for processing) and**
- **a due date $d_j \geq r_j + p_j$(representing the beginning of the day by which the job must be completed).**
- **We allow preemptions on machines.**
- **The problem is to determine a feasible schedule that completes all jobs before their due dates or show no such schedule is possible.**

Formulation of the feasible scheduling problem as a maximum flow problem

Consider the problem given here.

No. of machines M = 3

Job (j)	**1**	**2**	**3**	**4**
Processing time(p_j)	**1.5**	**1.25**	**2.1**	**3.6**
Release time (r_j)	**3**	**1**	**3**	**5**
Due date (d_j)	**5**	**4**	**7**	**9**

- **First rank all the release and due dates, r_j and d_j for all j, in ascending order. We get**

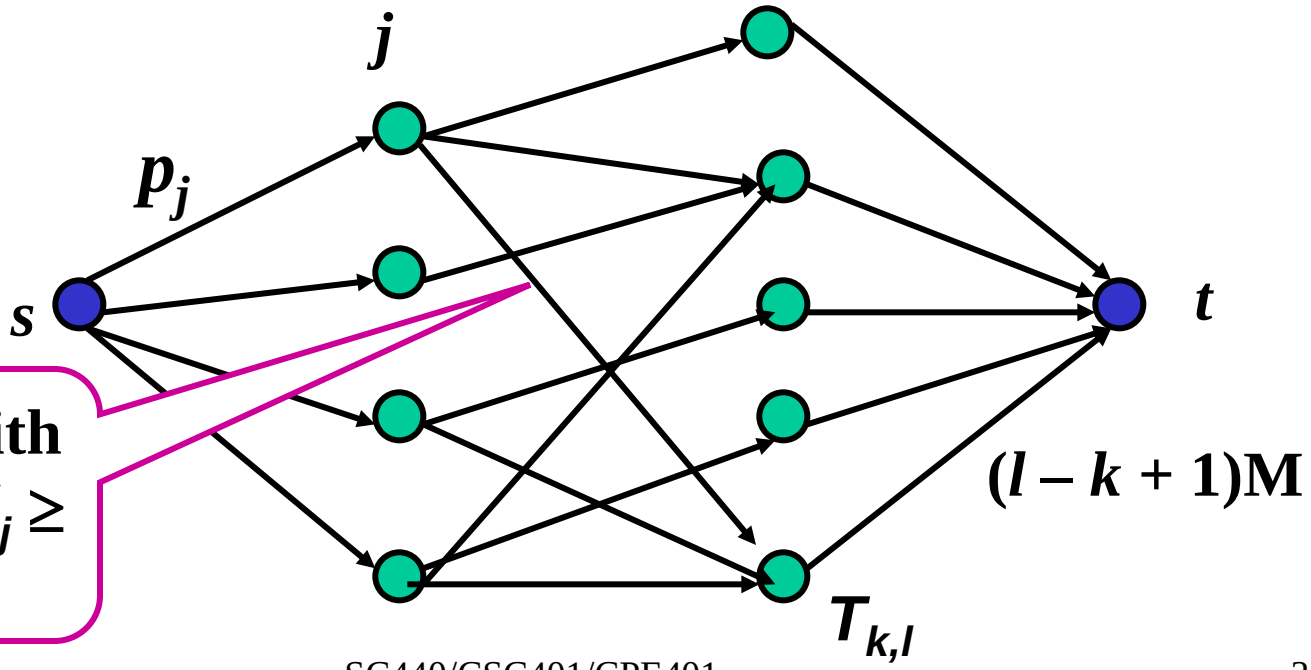
  1, 3, 4, 5, 7, 9

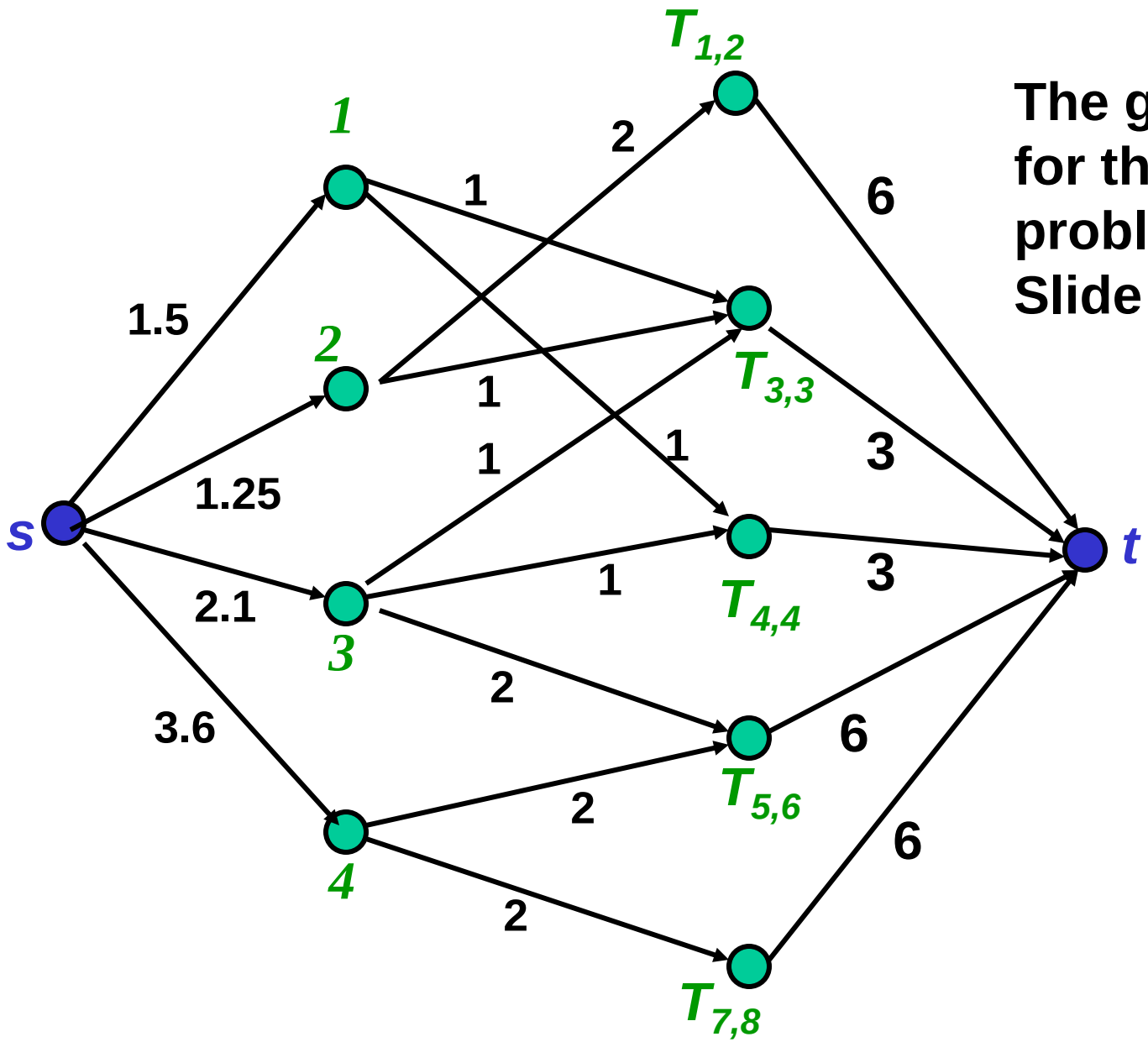
- **Next determine the mutually disjoint intervals of dates. Let $T_{k,l}$ denote the interval that starts at the beginning of date k and ends at the beginning of date $l + 1$. We have 5 intervals**

  $$T_{1,2}\ ,\ T_{3,3}\ ,\ T_{4,4}\ ,\ T_{5,6}\ ,\ T_{7,8}$$

- **Notice that we can process all jobs j with $r_j \leq k$ and $d_j \geq l + 1$ in the interval $T_{k,l}$.**
- **We formulate the scheduling problem as a maximum flow problem on a bipartite network G.**
- **We introduce a source node s, a sink node t, a node representing each job j and a node representing each interval $T_{k,l}$.**

- **The capacity of the edge from source *s* to each j is p_j .**
- **The capacity of the edge from each $T_{k,l}$ to sink *t* is $(l - k + 1)M$.**
- **Connect each job *j* to each $T_{k,l}$ if $r_j \le k$ *and* $d_j \ge l + 1$ with edge capacity $l - k + 1$.**





The graph G for the problem in Slide 31

- **The scheduling problem has a feasible schedule if and only if the maximum flow value in G equals the sum of all p_j, in other words, the flow on every edge (s, j) is p_j.**
- **The validity of this formulation can be established by showing a one-to-one correspondence between feasible schedules and the flows of value $\sum_{j \in J} p_j$ from the source to the sink.**

Exercises

1. **Formulate the following scheduling problem on uniform parallel machines as a maximum flow problem, assuming 2 machines are available each day. You do not need to solve them.**

Job (j)	**1**	**2**	**3**	**4**
Processing time in days(p_j)	**2.5**	**3.1**	**5.0**	**1.8**
Release time (r_j)	**1**	**2**	**1**	**2**
Due date (d_j)	**4**	**7**	**7**	**5**

2. Several families go out to dinner together. To increase their social interaction, they would like to sit at tables so that no two members of the same family are at the same table. Show how to formulate finding a seating arrangement that meets this objective as a maximum flow problem. Assume that the dinner contingent has *p* families and that the *i*th family has *a*(*i*) members. Also assume that *q* tables are available and that the *j*th table has a seating capacity of *b*(*j*).

3. The figure below shows the network for a feasible flow problem. Transform the 2 instances of the problem into 2 maximum flow problems. You do not need to solve them.

Problem instance 1:

b(7) = 6, b(4) = - 3

b(1) = 5, b(2) = -8

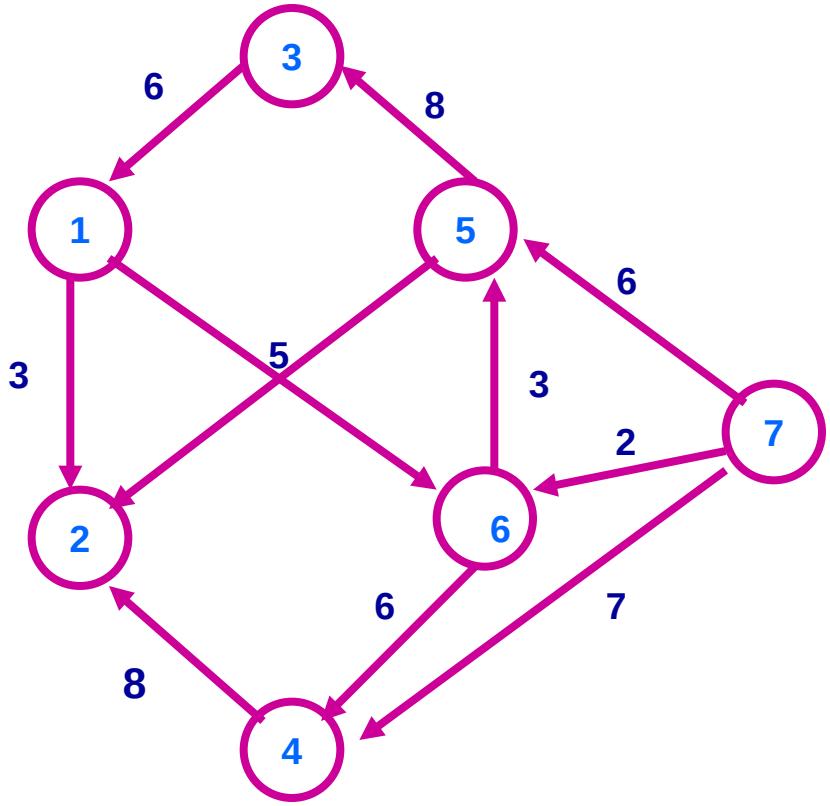
Other b(u) = 0

Problem instance 2:

b(5) = 6,

b(7) = 8, b(2) = -14

Other b(u) = 0



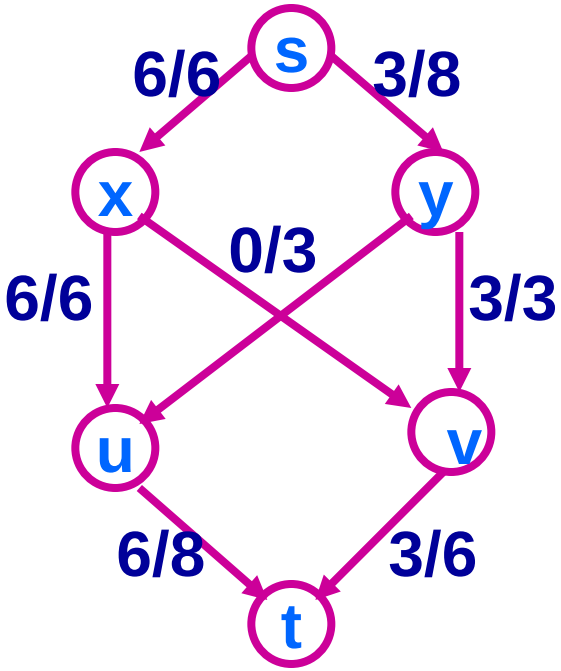
	s	x	y	u	v	t
s	0	6	8	0	0	0
x	0	0	0	6	3	0
y	0	0	0	3	3	0
u	0	0	0	0	0	8
v	0	0	0	0	0	6
t	0	0	0	0	0	0

The capacity

	s	x	y	u	v	t
s		6	3			
x				6		
y					3	
u						6
v						3
t						

The current flow

f(u,v) = - f(v,u)



	s	x	y	u	v	t
s						
x						
y						
u						
v						
t						

$f(u,v) = - f(v,u)$

$c_f(u,v) = c(u, v) - f(u, v)$

The residual capacity