COMP261
Algorithms and Data Structures
2024 Tri 1

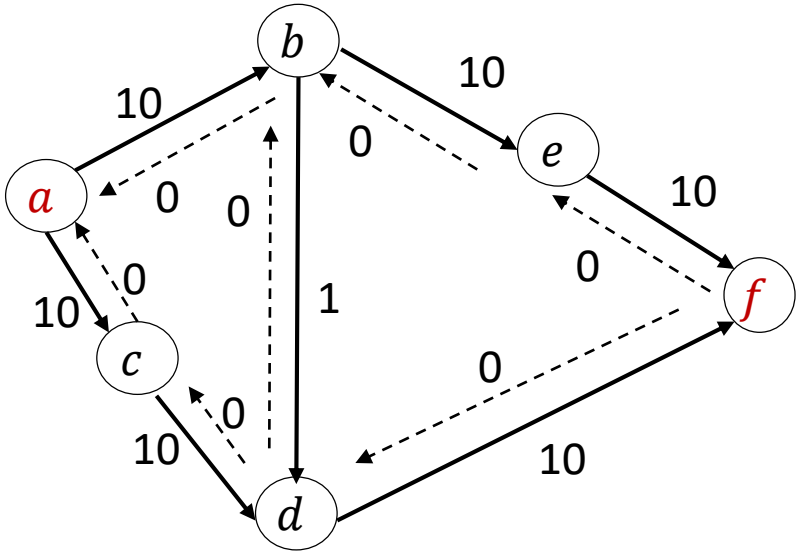
Jyoti Sahni
jyoti.sahni@ecs.vuw.ac.nz
Office Hours (COMP261): AM414, Thursday 10:00 – 12:00

Recap: Ford-Fulkerson method

It is generally called a method rather than an algorithm, as it encompasses several different implementations with different running times.

- Based on 2 important ideas:
  - Residual Graphs (includes reverse edges)
  - Augmentation paths (path in a residual graph)

Recap: Solving Maximum flow problem



Forward edges: for each edge $e = (u, v)$ of G for which $f_e < c_e$, include an edge $e' = (u, v)$ in G_f with capacity $c_e - f_e$

Backward edges (represent decreasing flow): for each edge $e = (u, v)$ of G with $f_e > 0$, include an edge $e' = (v, u)$ in G_f with capacity f_e

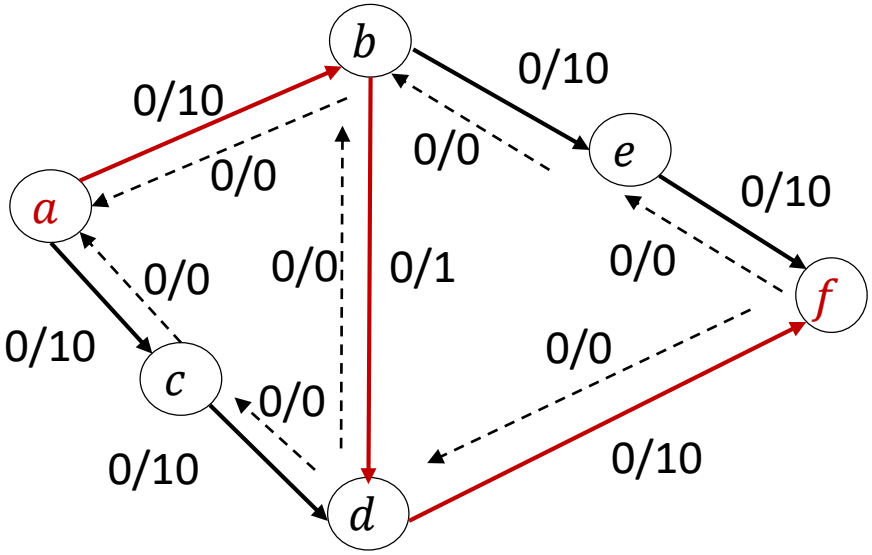
Recap: Solving Maximum flow problem

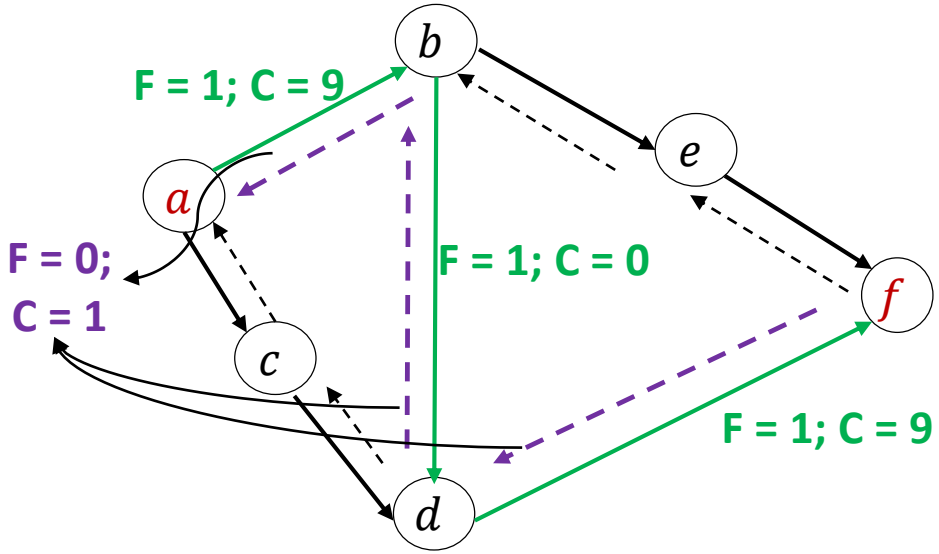
Note : a/b = *flow / capacity left*

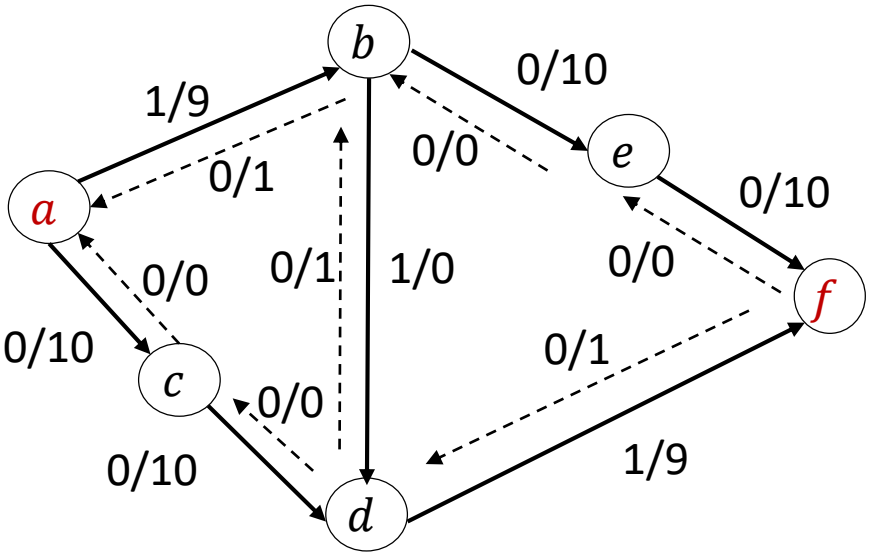
$a \longrightarrow b \longrightarrow d \longrightarrow f$ 1

Forward edges:
capacity $c_e - f_e$

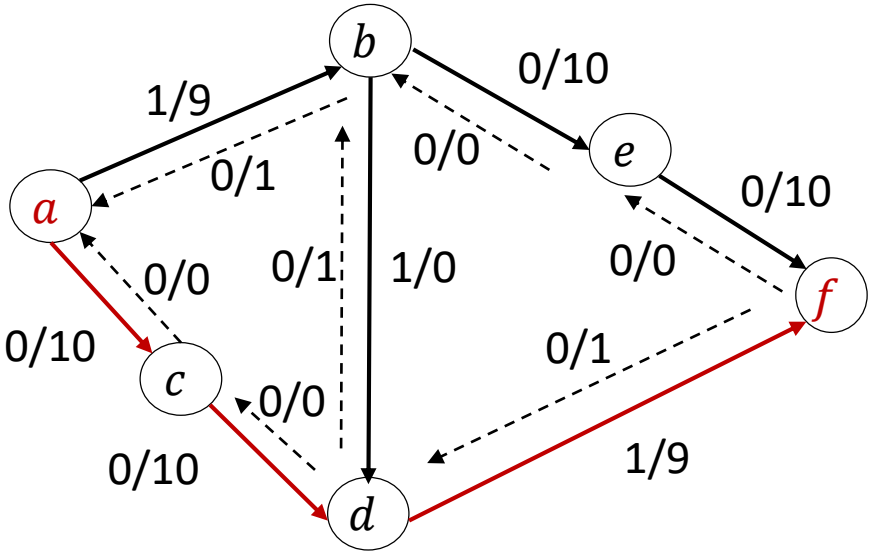
Backward edges (represent decreasing flow):
capacity f_e







Recap: Solving Maximum flow problem



$a \longrightarrow b \longrightarrow d \longrightarrow f$ 1

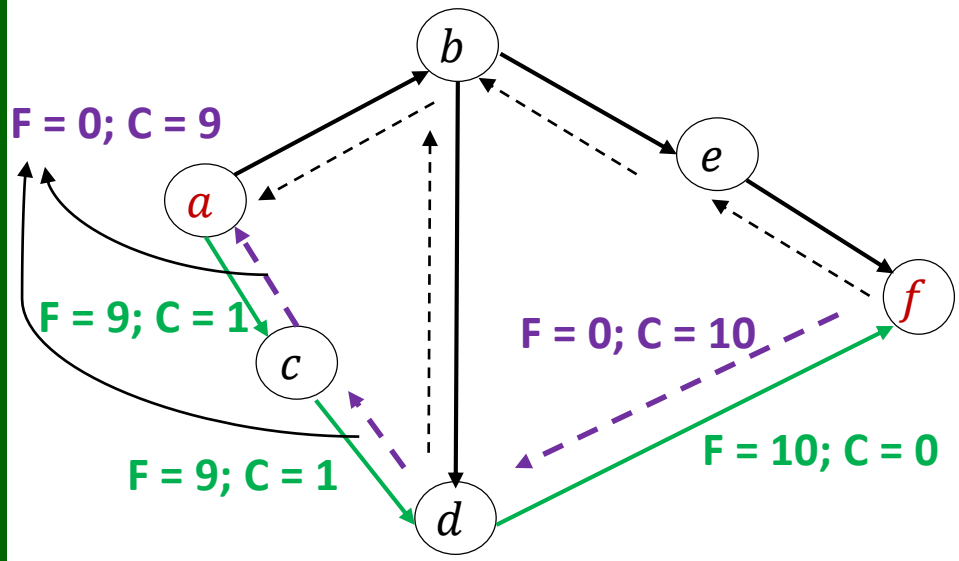
$a \longrightarrow c \longrightarrow d \longrightarrow f$ 9

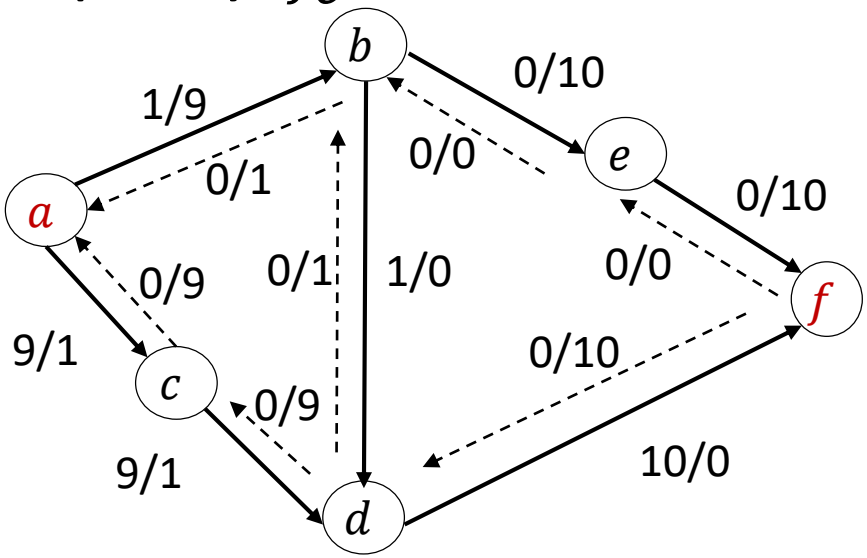
Forward edges:

capacity $c_e - f_e$

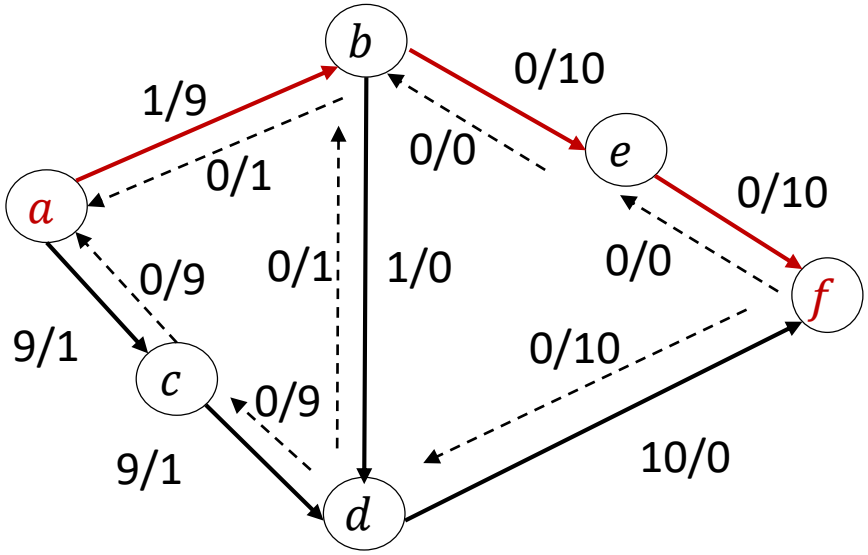
Backward edges (represent decreasing flow):

capacity f_e





Recap: Solving Maximum flow problem



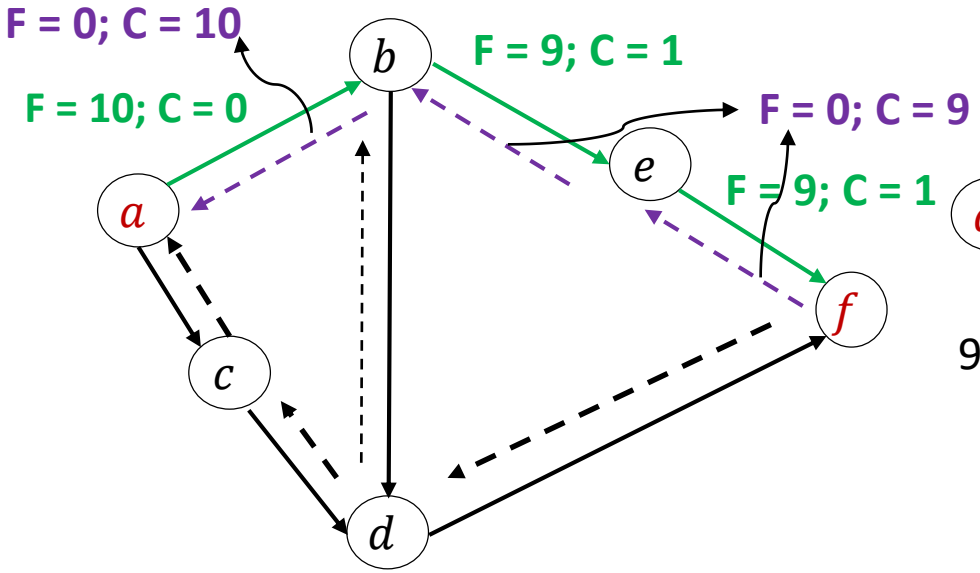
$a \longrightarrow b \longrightarrow d \longrightarrow f$ 1

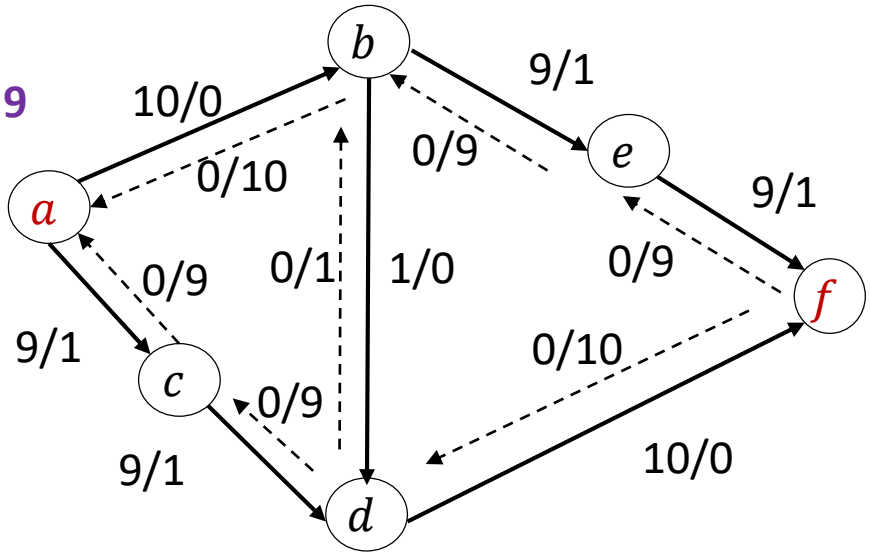
$a \longrightarrow c \longrightarrow d \longrightarrow f$ 9

$a \longrightarrow b \longrightarrow e \longrightarrow f$ 9

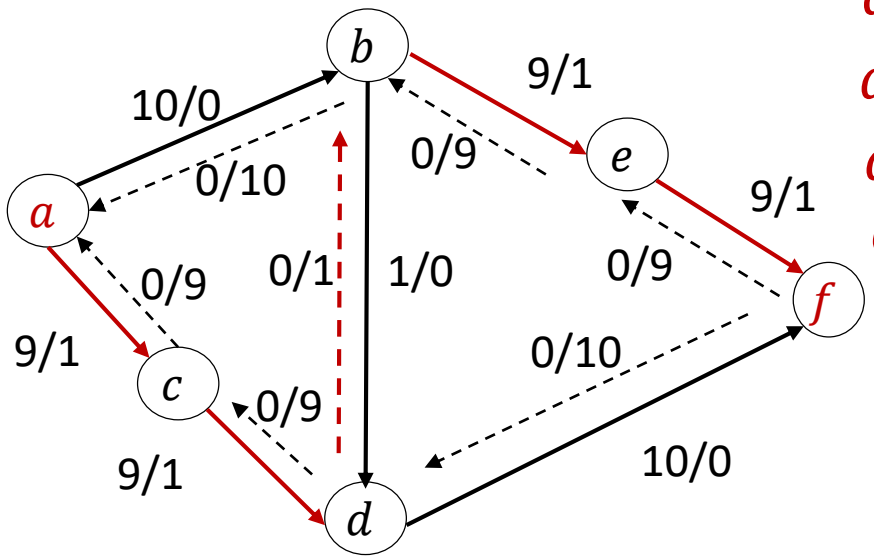
Forward edges:
capacity $c_e - f_e$

Backward edges (represent decreasing flow):
capacity f_e





Recap: Solving Maximum flow problem

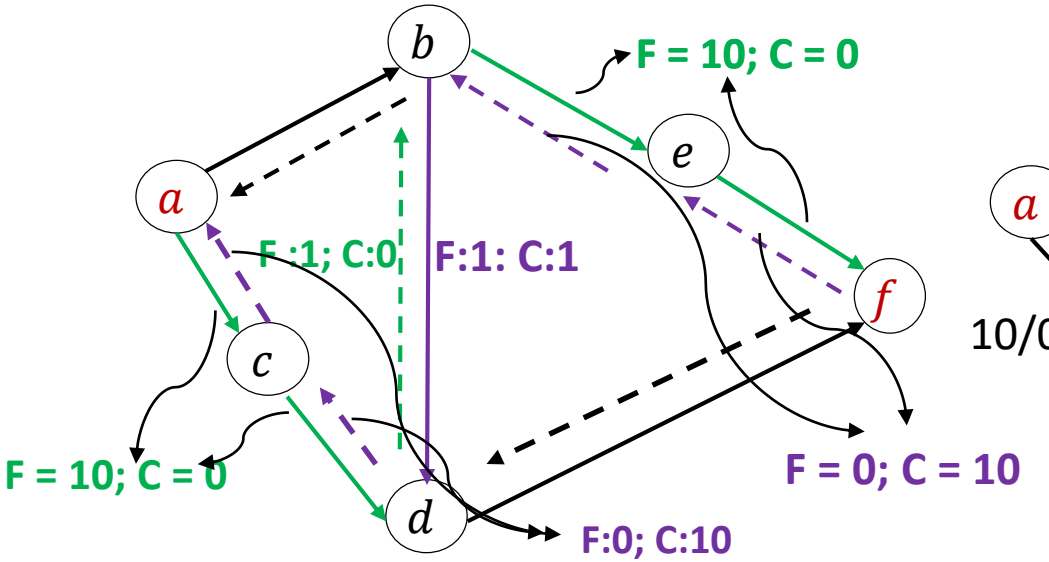


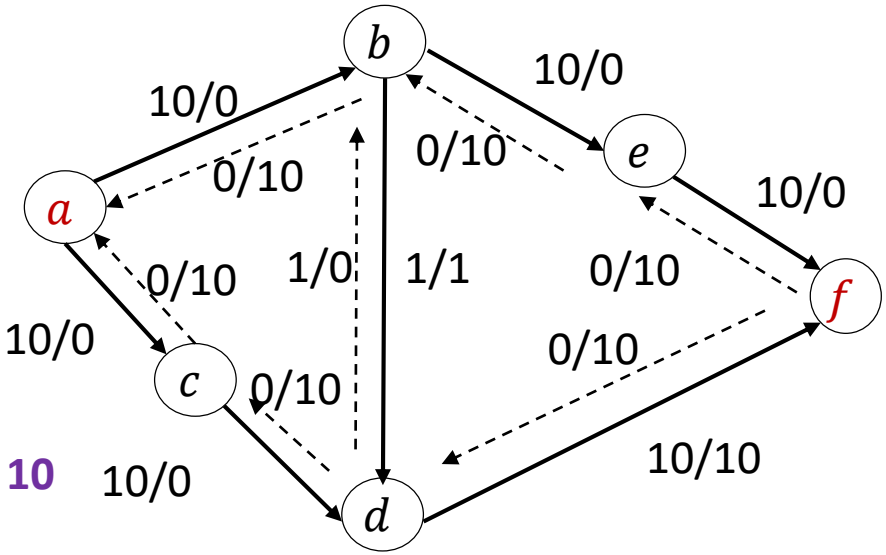
$a \longrightarrow b \longrightarrow d \longrightarrow f$ 1

$a \longrightarrow c \longrightarrow d \longrightarrow f$ 9

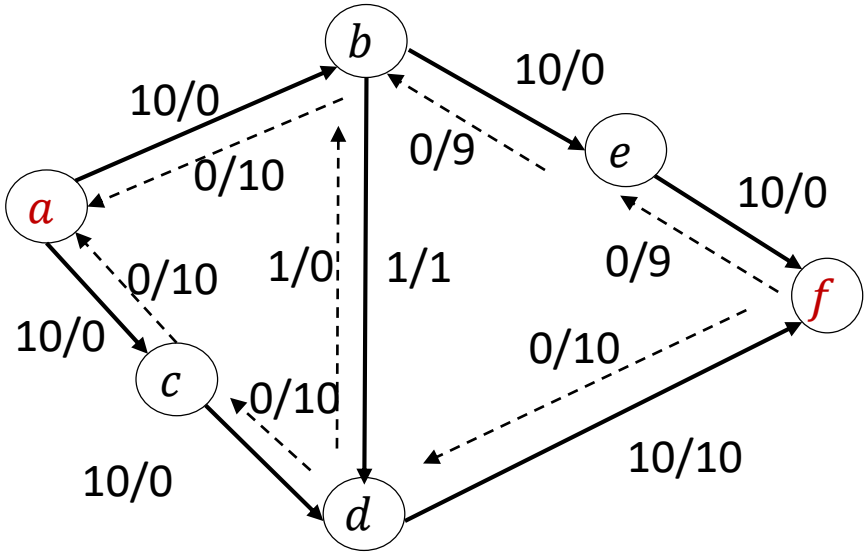
$a \longrightarrow b \longrightarrow e \longrightarrow f$ 9

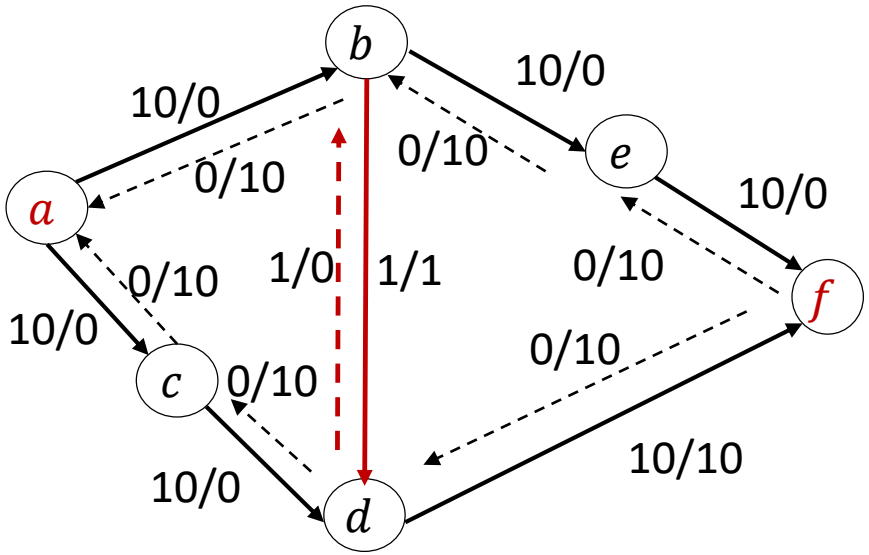
$a \longrightarrow c \longrightarrow d \longrightarrow b \longrightarrow e \longrightarrow f$ 1

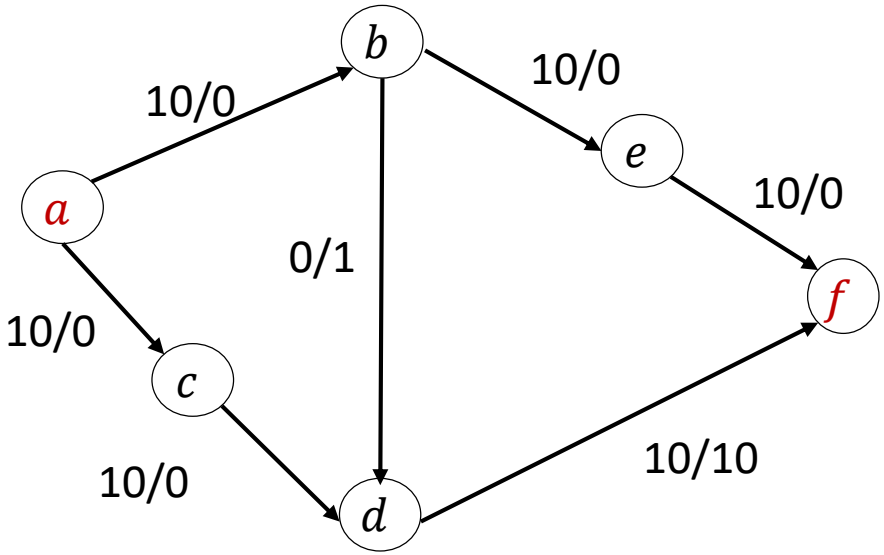




Recap: Solving Maximum flow problem







Overall flow is given by:

$$f_e + g_e - g'_e$$

Recap: Ford-Fulkerson method

- Follows a greedy approach. Iteratively increase the value of flow.

```
FordFulkerson(G, s, t)
Let f_e = 0 for all edges (no flow anywhere)
Initialize Residual Graph RG //  for every forward edge in the
original graph G add a reverse edge with a capacity 0
maxFlow = 0
    Repeat
       Find some P path from s to t in RG such that c_e > 0 for all
       edges in P
           if P exists
              pathFlow = Bottleneck (P, RG)
              maxFlow = maxFlow + pathFlow
              Output (P, pathflow) as an augmentation path
               Update RG
        Endif
    Until the RG has no more augmentation paths.
    Output maxFlow
```

Note: Ford-Fulkerson does not state how to find augmentation paths.

Updating Residual Graph

```
For each edge e = (u, v) ∈ P:
    f(u,v) = f(u,v) + pathFlow //add flow
    c(u,v) = c(u,v) − pathFlow //reduce capacity

    c(v,u) = c(v,u) + pathFlow//increase capacity in reverse edge
EndFor
```

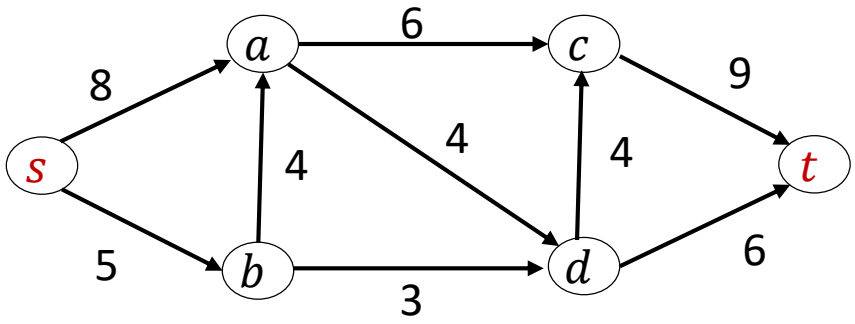
Edmonds-Karp algorithm is an implementation of the Ford-Fulkerson method that uses BFS for finding augmenting paths.

```
BFS(RG, s, t)
Define augPath as ArrayList of edges
q := queue()
q.push(s)
backpointer(v) = null for all v   // backpointer data-structure to hold
                                  //edges that lead to the vertex
while !q.isEmpty()
    cur := q.pull()
    for each outedge e of cur in RG do
      if e.toCity! = s and backpointer(e.toCity) == null and e.cap ≠ 0
            backpointer(e.tocity): = e
            if(backpointer(t)! = null)   // found a path from s to t. Build it now from reverse
                pathEdge = backpointer(t)
                while(pathEdge ! = null)
                      augPath.add(pathEdge)
                       pathEdge = backpointer(pathEdge.fromCity)
                 endWhile
                 Collections.reverse(augPath)
                return augPath
            endIF
            q.push(e.toCity)
      endIf
    endFor
endWhile
return null
```

Implementation – Dry run

Implementing Edmond Karp

- Graph Representation:
  - Adjacency matrix



(Capacity, flow)

3

	s	a	b	c	d	t
s	(0,0)	(8,0)	(5,0)	(0,0)	(0,0)	(0,0)
a	(0,0)	(0,0)	(0,0)	(6,0)	(4,0)	(0,0)
b	(0,0)	(4,0)	(0,0)	(0,0)	(3,0)	(0,0)
c	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(9,0)
d	(0,0)	(0,0)	(0,0)	(4,0)	(0,0)	(6,0)
t	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)

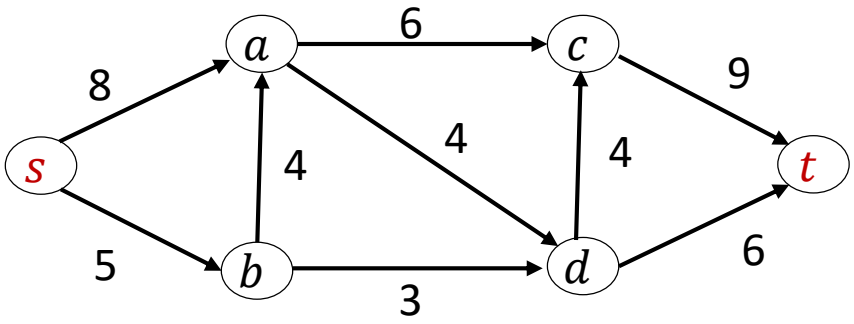
Implementing Edmond Karp

- Graph Representation:
  - Adjacency List

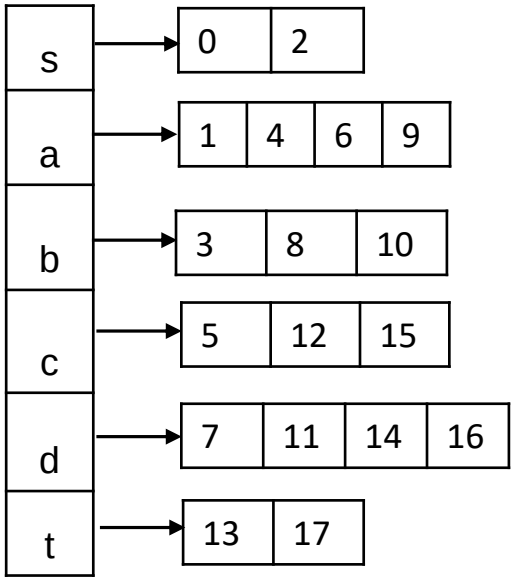
Important to understand this for assignment 3.

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

Edge List (from, to, capacity, flow)



Node list:

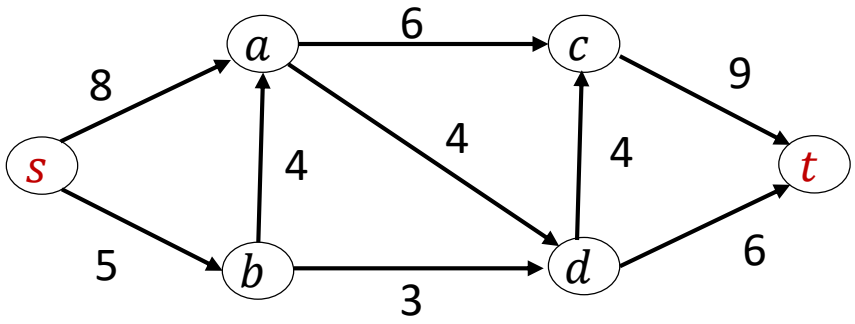


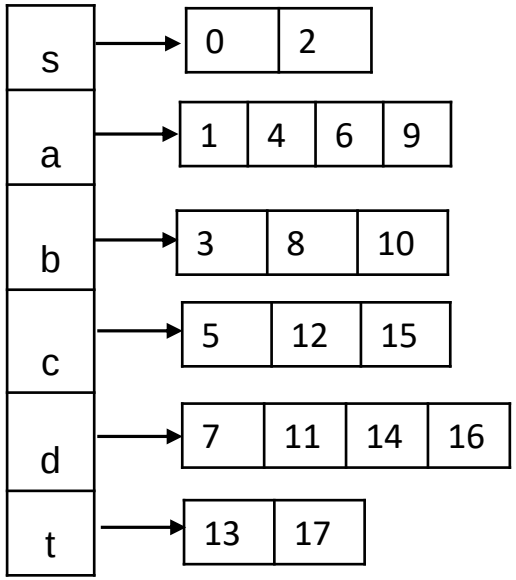
For each node, maintain a list of edge ids originating from the node

Example

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

Reverse edges





Step 1:

a) Initialize residual graph

b) Set maxFlow = 0

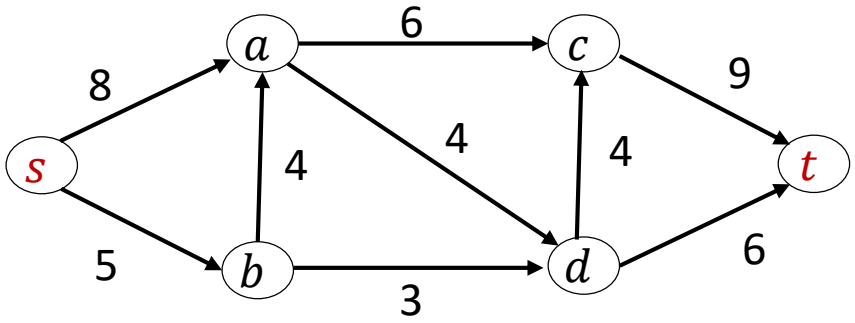
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

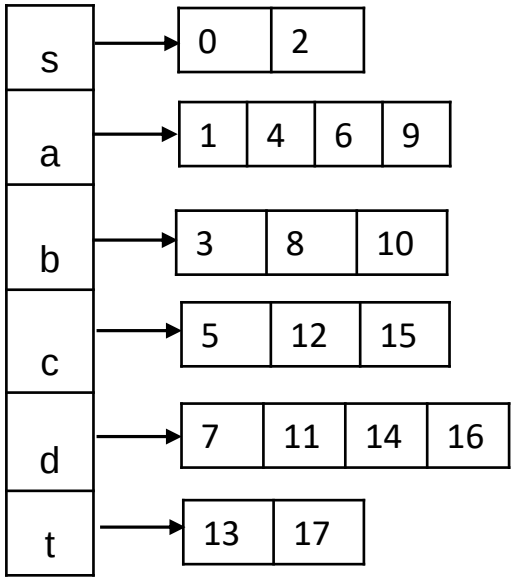
Front

Rear

s

Queue

s	a	b	c	d	t
null	null	null	null	null	null



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

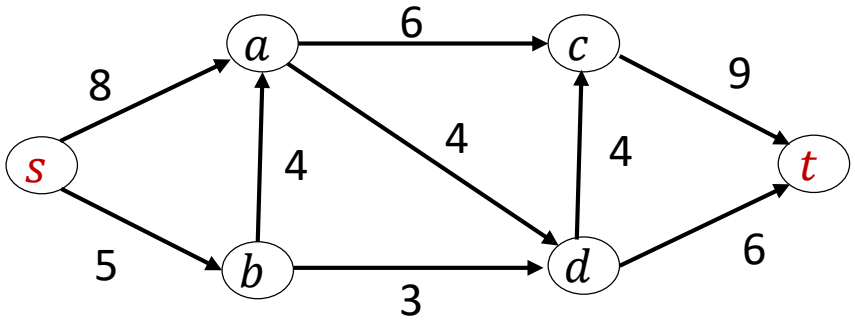
Example

Step 2:

a) find augmentation path

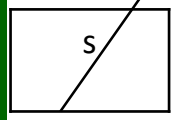
b) add to flow

c) Update residual graph



Stop when no augmentation paths

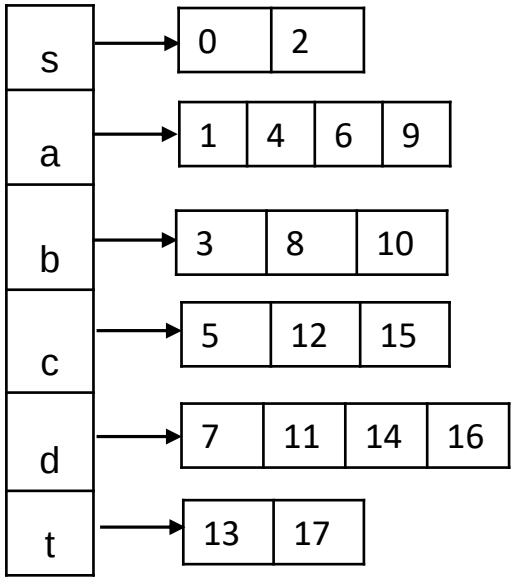
Front



Rear

Queue

s	a	b	c	d	t
null	null	null	null	null	null



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

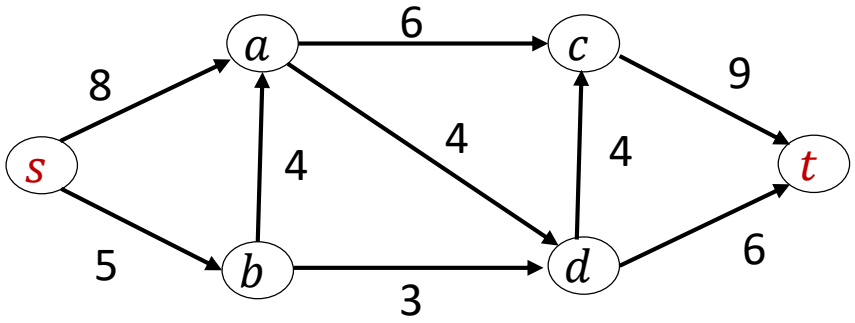
Example

Step 2:

a) find augmentation path

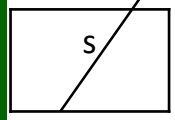
b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front

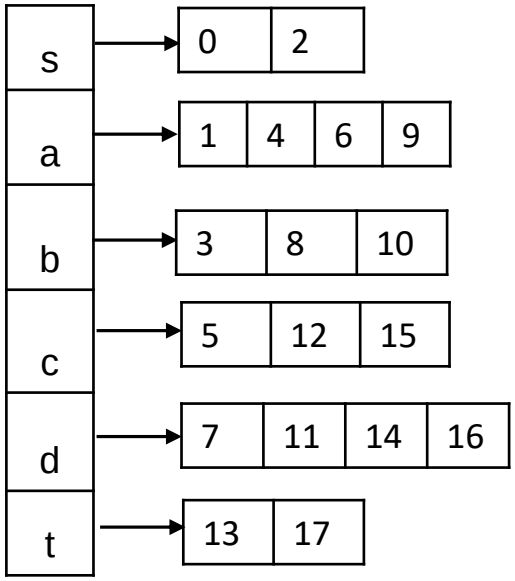


Rear

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

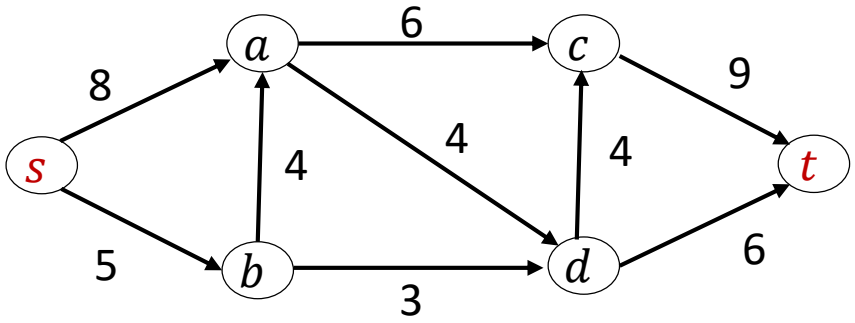
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front

Rear

s	a

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s	0 → 2
a	1 → 4 → 6 → 9
b	3 → 8 → 10
c	5 → 12 → 15
d	7 → 11 → 14 → 16
t	13 → 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

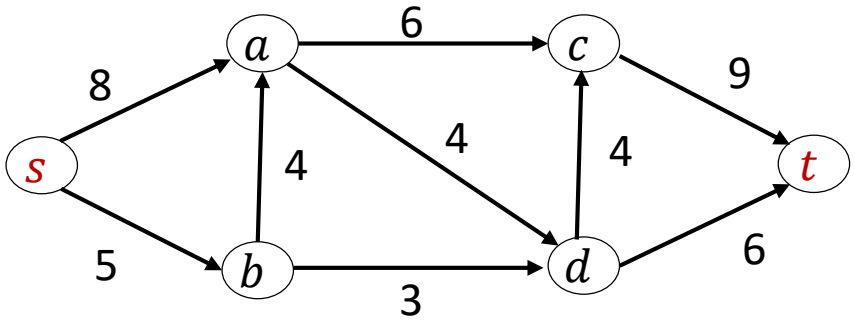
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front

Rear

s	a

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

s	0 → 2
a	1 → 4 → 6 → 9
b	3 → 8 → 10
c	5 → 12 → 15
d	7 → 11 → 14 → 16
t	13 → 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

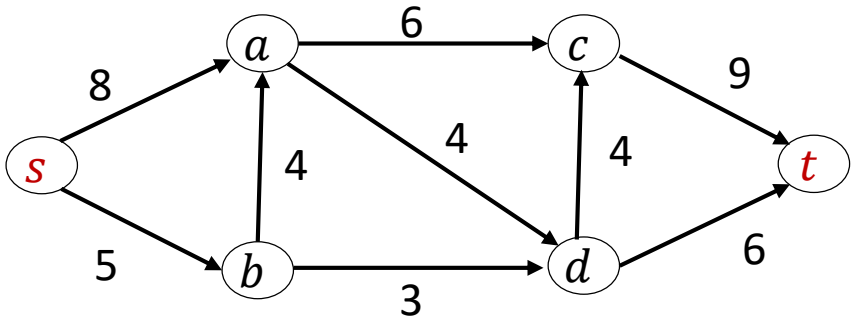
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front

Rear

s	a	b

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

s	0 → 2
a	1 → 4 → 6 → 9
b	3 → 8 → 10
c	5 → 12 → 15
d	7 → 11 → 14 → 16
t	13 → 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

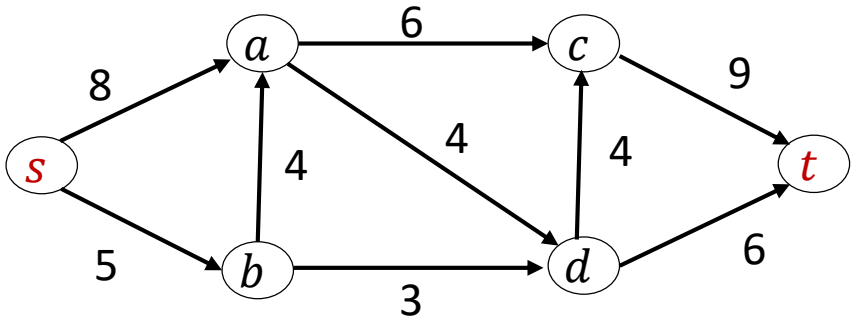
Example

Step 2:

a) find augmentation path

b) add to flow

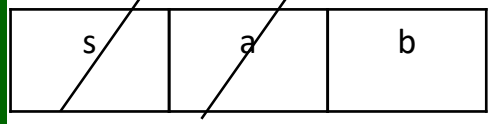
c) Update residual graph



Stop when no augmentation paths

Front

Rear

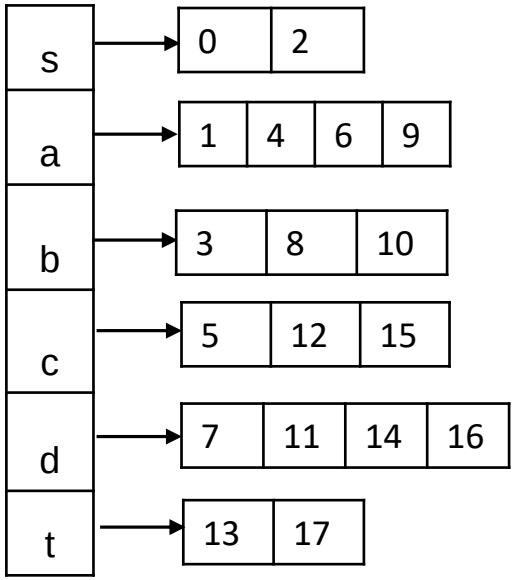


Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

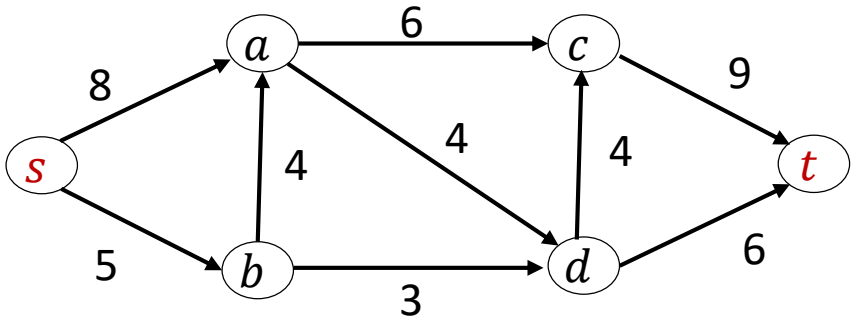
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front

Rear

s	a	b

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

s	0 → 2
a	1 → 4 → 6 → 9
b	3 → 8 → 10
c	5 → 12 → 15
d	7 → 11 → 14 → 16
t	13 → 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

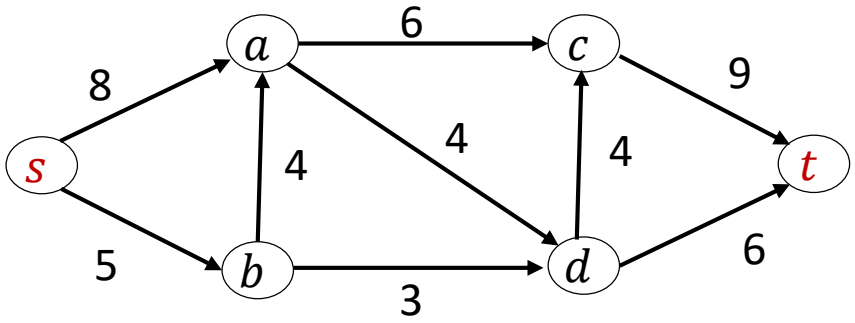
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front Rear

s	a	b	c

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

s	0 → 2
a	1 → 4 → 6 → 9
b	3 → 8 → 10
c	5 → 12 → 15
d	7 → 11 → 14 → 16
t	13 → 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

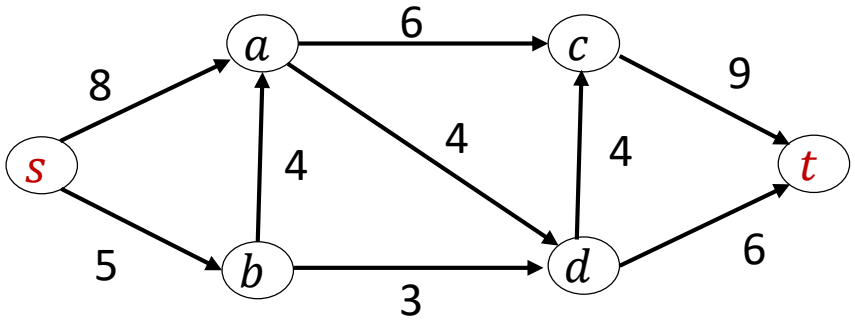
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Front

Rear

s	a	b	c

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

a->d edge[6]

s	0 2
a	1 4 6 9
b	3 8 10
c	5 12 15
d	7 11 14 16
t	13 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

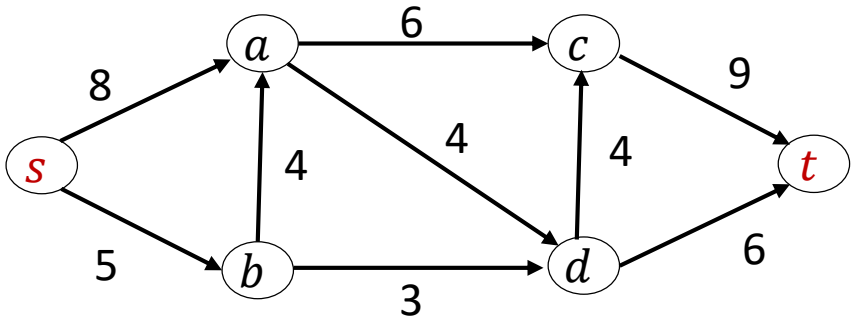
Example

Step 2:

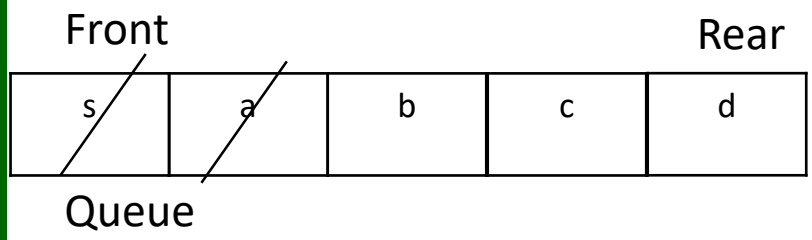
a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths



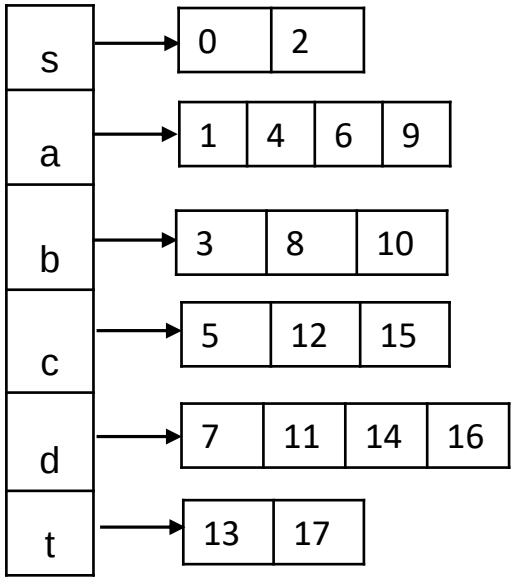
s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

a->d edge[6]



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

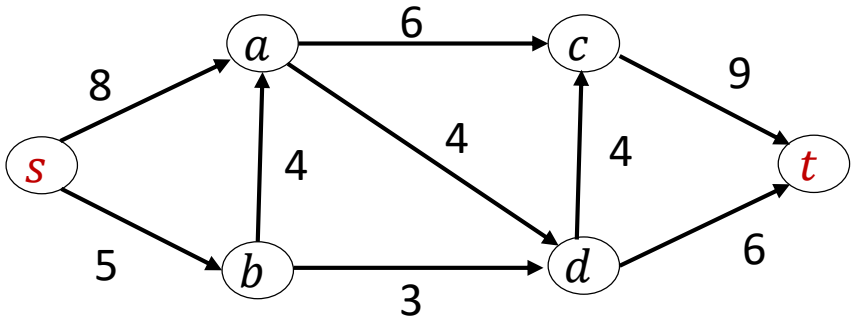
Example

Step 2:

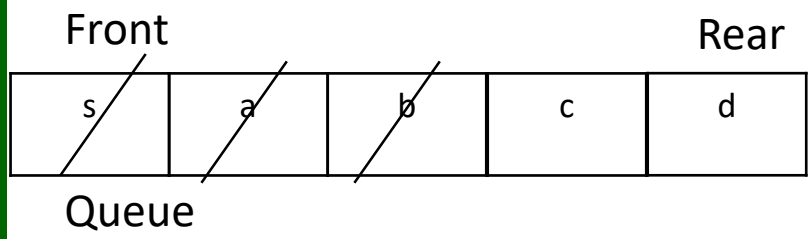
a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths



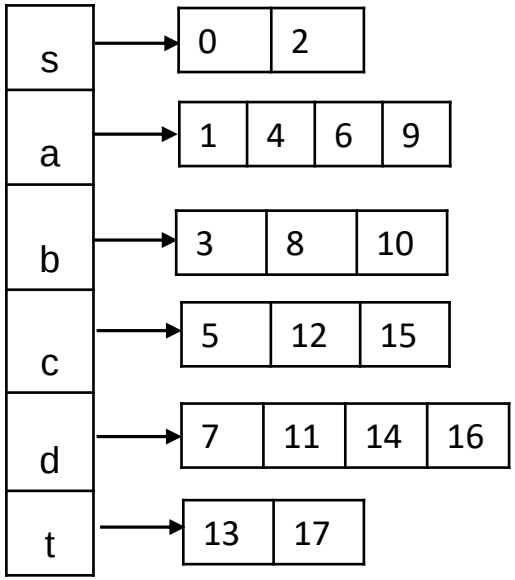
s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

a->d edge[6]



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

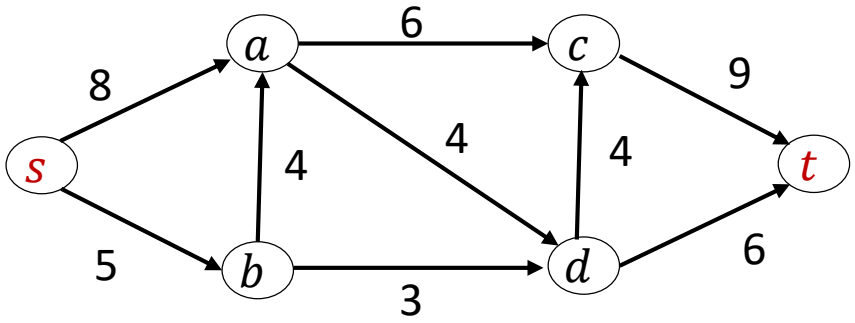
Example

Step 2:

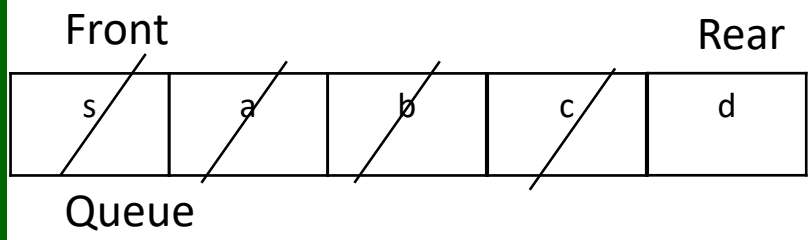
a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths



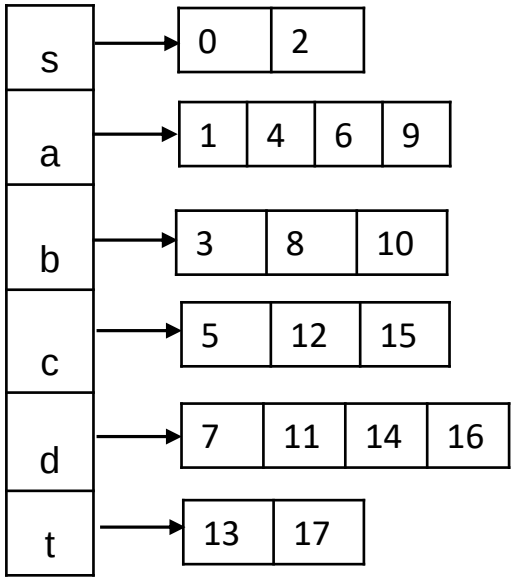
s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

a->d edge[6]



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

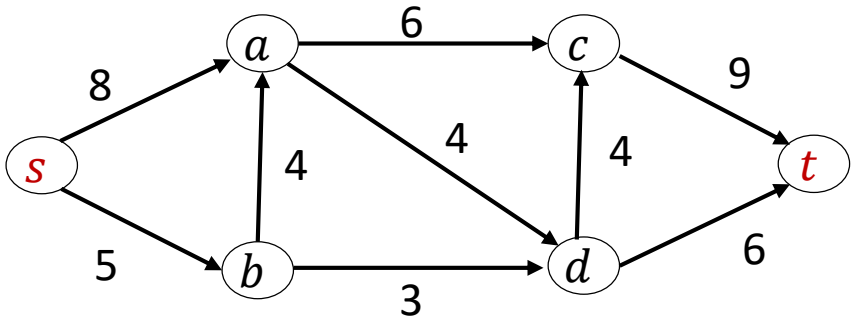
Example

Step 2:

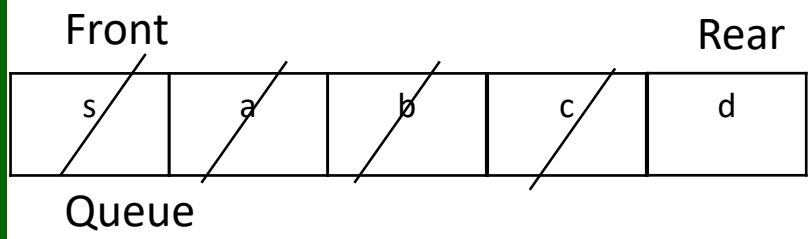
a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths



s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

a->d edge[6]

c->t edge[12]

s	0 2
a	1 4 6 9
b	3 8 10
c	5 12 15
d	7 11 14 16
t	13 17

0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

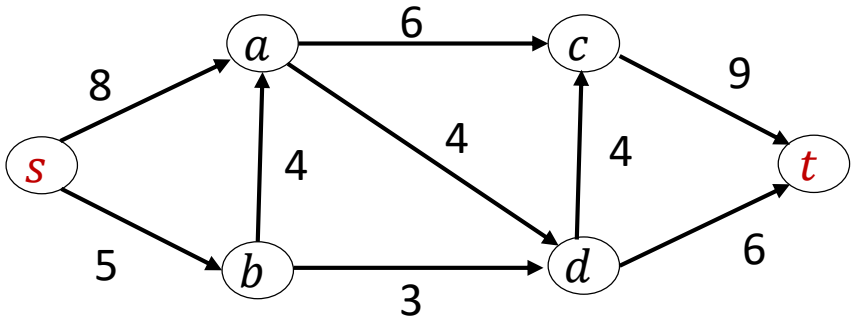
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



0	(s,a,8,0)
1	(a,s,0,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,6,0)
5	(c,a,0,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,9,0)
13	(t,c,0,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

Stop when no augmentation paths

Front Rear

s	a	b	c	d

Queue

s	a	b	c	d	t
null	null	null	null	null	null

s->a edge[0]

s->b edge[2]

a->c edge[4]

a->d edge[6]

c->t edge[12]

s	0	2		
a	1	4	6	9
b	3	8	10	
c	5	12	15	
d	7	11	14	16
t	13	17		

Augmentation path: $[0][4][12]$

$pathFlow: min\{8,6,9\} = 6$

$maxFlow = 6$

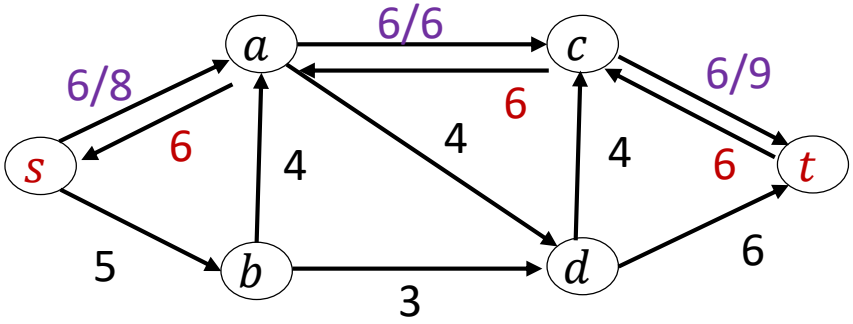
Example

Step 2:

a) find augmentation path

b) add to flow

c) Update residual graph



Stop when no augmentation paths

Array representation:

if $G[u][v]$ is an edge, its reverse edge will be $G[v][u]$

Adjacency list representation:

If Edge array list stores an edge at an even index :i it's reverse edge is at $i + 1$

Otherwise its reverse edge is at index $i - 1$

For each edge in the augmentation path:

- Increase flow by bottleneck
- Decrease capacity by bottleneck
- Increase capacity in the reverse edge by bottleneck

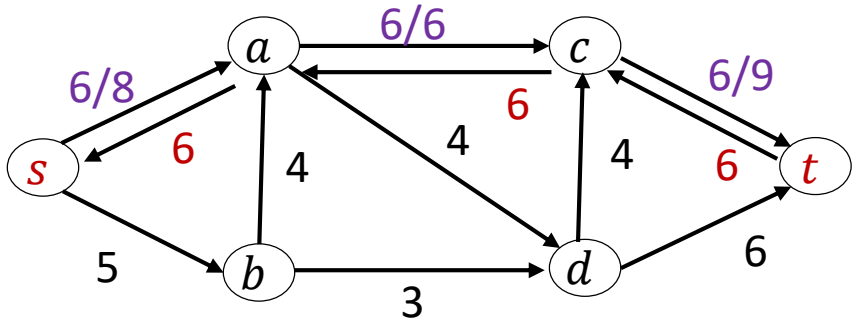
Example

Step 2:

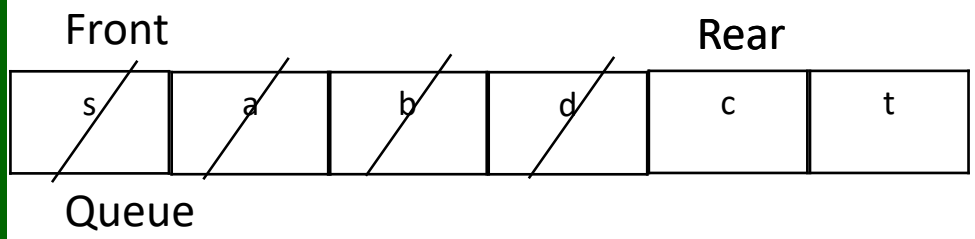
a) find augmentation path

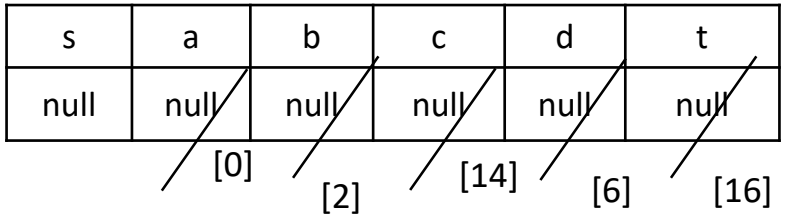
b) add to flow

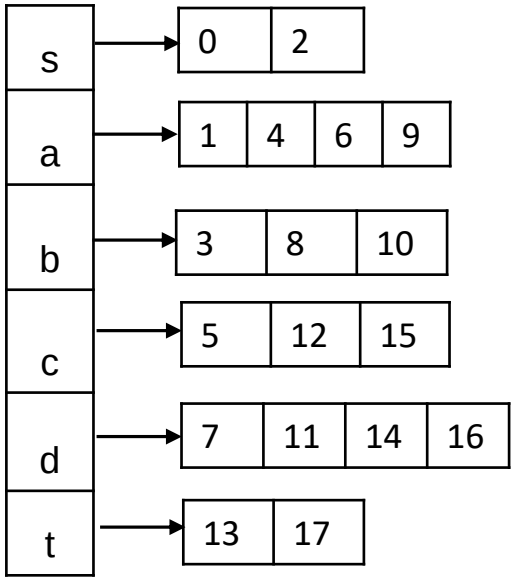
c) Update residual graph



Stop when no augmentation paths







0	(s,a,2,6)
1	(a,s,6,0)
2	(s,b,5,0)
3	(b,s,0,0)
4	(a,c,0,6)
5	(c,a,6,0)
6	(a,d,4,0)
7	(d,a,0,0)
8	(b,a,4,0)
9	(a,b,0,0)
10	(b,d,3,0)
11	(d,b,0,0)
12	(c,t,3,6)
13	(t,c,6,0)
14	(d,c,4,0)
15	(c,d,0,0)
16	(d,t,6,0)
17	(t,d,0,0)

Augmentation path: [0] [6] [16]. $pathFlow$: $\min\{2,4,6\} = 2$

$maxFlow = 6 + 2 = 8$

Example

Repeat until no more augmentation paths are found:

Augmentation path: $s \rightarrow b \rightarrow d \rightarrow t. pathFlow: 3$

Augmentation path: $s \rightarrow b \rightarrow a \rightarrow d \rightarrow t.\ pathFlow: 1$

Augmentation path: $s \rightarrow b \rightarrow a \rightarrow d \rightarrow c \rightarrow t.\ pathFlow: 1$

$Maxflow: 13$

Next Lecture

- Centrality metrics