Maximum Flow Algorithms

Network Algorithms

Georgia Kaouri

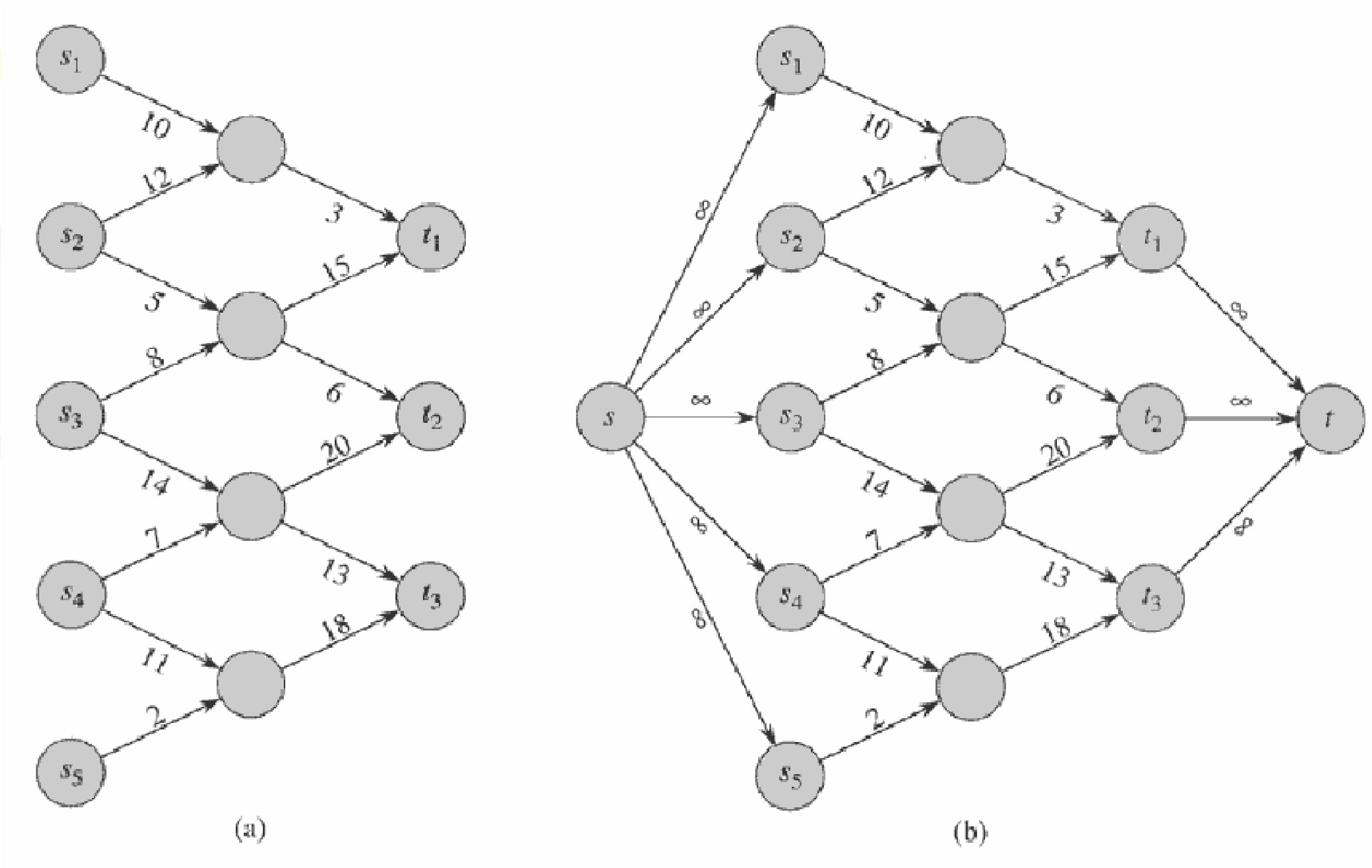
Contents

- Applications, special cases
- Flow networks
- Ford Fulkerson algorithm
- Preflow - push algorithms
- Lift - to - front algorithm

Applications

- Material flows
- Model liquids flowing, current through electrical networks, information through communication networks, etc
- Maximum matching in bipartite graphs

Special case: multiple source, multiple sink maximum flow - problem



Problem Definition

Input: A connected, directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a non negative capacity $c(u, v) \geq 0$. There is a node $s \in V$ (source), s.t. for all $u_i \in V$ $c(u_i, s)=0$ and $t \in V$ (target), s.t. for all $u_i \in V$ $c(t, u_i)=0$.

Output: A maximum flow from s to t.

Flow

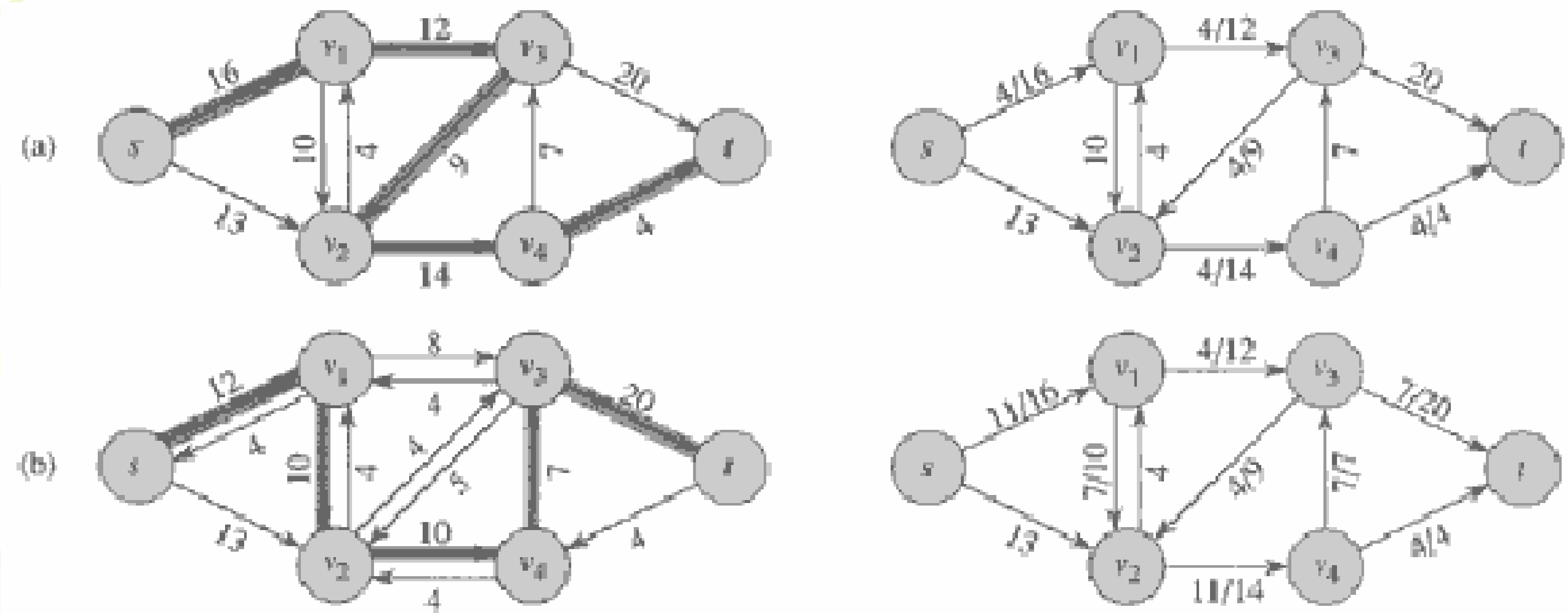
A flow in G is a real valued function $f: V \times V \rightarrow R$ that satisfies the following properties:

Capacity constraint: For all u, v ∈ V, we require $f(u, v) \leq c(u, v)$

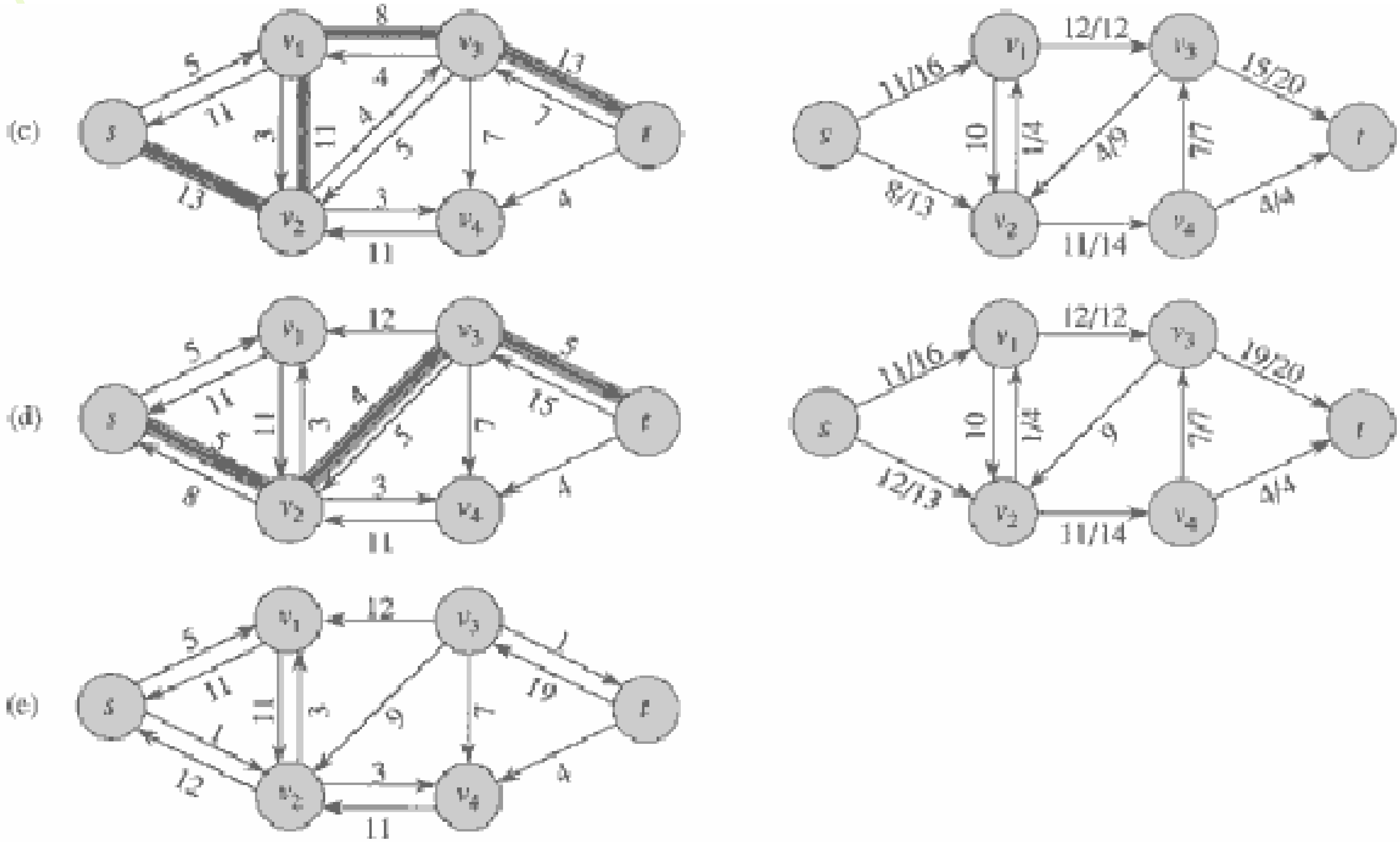
Skew symmetry: For all u, v ∈ V, we require $f(u, v) = -f(v, u)$

Flow conservation: For all u ∈ V-{s, t}, we require $\sum_{u \in V} f(u, v) = 0$

Ford - Fulkerson method (G, s, t)



Ford - Fulkerson method (G, s, t)



Edmonds - Karp algorithm

Modify the Ford - Fulkerson algorithm so that the augmenting path is a shortest path from s to t in the residual network.

Ford - Fulkerson: $O(E |f^*|)$

Edmonds - Karp: $O(VE^2)$

Preflow - Push algorithm

- Preflow – push algorithms work on one vertex at a time and its neighbors.
- Flow conservation property is not maintained.
- A preflow is maintained.

Preflow is a function $f:V\times V \rightarrow R$ that satisfies skew symmetry, capacity constraints and the following relaxation of flow conservation: $f(V, u) \geq 0$ for all vertices $u \in V-s$.

Excess flow into u is the net flow into a vertex given by $e(u) = f(u, V)$. If $e(u) > 0$, vertex u is overblowing.

Intuition

- Edges are like water pipes
- Nodes are joints
- Distance is like height from the ground.
  - Destination is at the ground.
- Initially source is at the highest level and sends water to all adjacent nodes.
- Whenever a node has accumulated water, it sends (**pushes**) water to nodes at lower label.
- So water moves towards the destination
- Sometimes water gets locally trapped as all neighboring nodes are at a greater height.
- Then, the node label is raised (**relabelling**).

Heights

Let $G = (V, E)$ be a flow network with source s and target t and let f be a preflow in G. A function $h: V \rightarrow N$ is a height function if

$h(s) = |V|$, $h(t) = 0$ and $h(u) \leq h(v) + 1$ for every residual edge $(u, v) \in E_f$.

It follows that if for two vertices $u, v \in V$ $h(u) > h(v) + 1$, then (u, v) is not in the residual graph.

Basic Operations: Push

If u is overflowing ($e(u) > 0$), $c_f(u, v) > 0$ and $h(u) = h(v) + 1$, we can PUSH

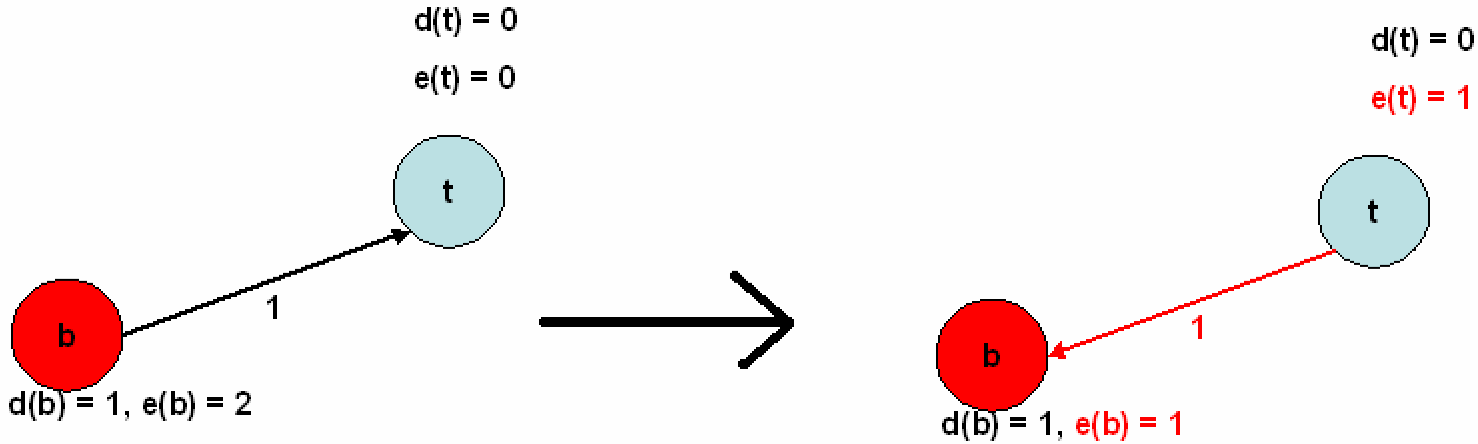
$d_f(u, v) = \min\{e(u), c_f(u, v)\}$ units of flow.

Then we have to update:

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

We call edge (u, v) an admissible edge.

Example



Code for Push operation

PUSH(u, v)

▷ **Applies when:** u is overflowing, $c_f(u, v) > 0$, and $h[u] = h[v] + 1$.
▷ **Action:** Push $d_f(u, v) = \min(e[u], c_f(u, v))$ units of flow from u to v.
$d_f(u, v) \leftarrow \min(e[u], c_f(u, v))$
$f[u, v] \leftarrow f[u, v] + d_f(u, v)$
$f[v, u] \leftarrow -f[u, v]$
$e[u] \leftarrow e[u] - d_f(u, v)$
$e[v] \leftarrow e[v] + d_f(u, v)$

Definitions

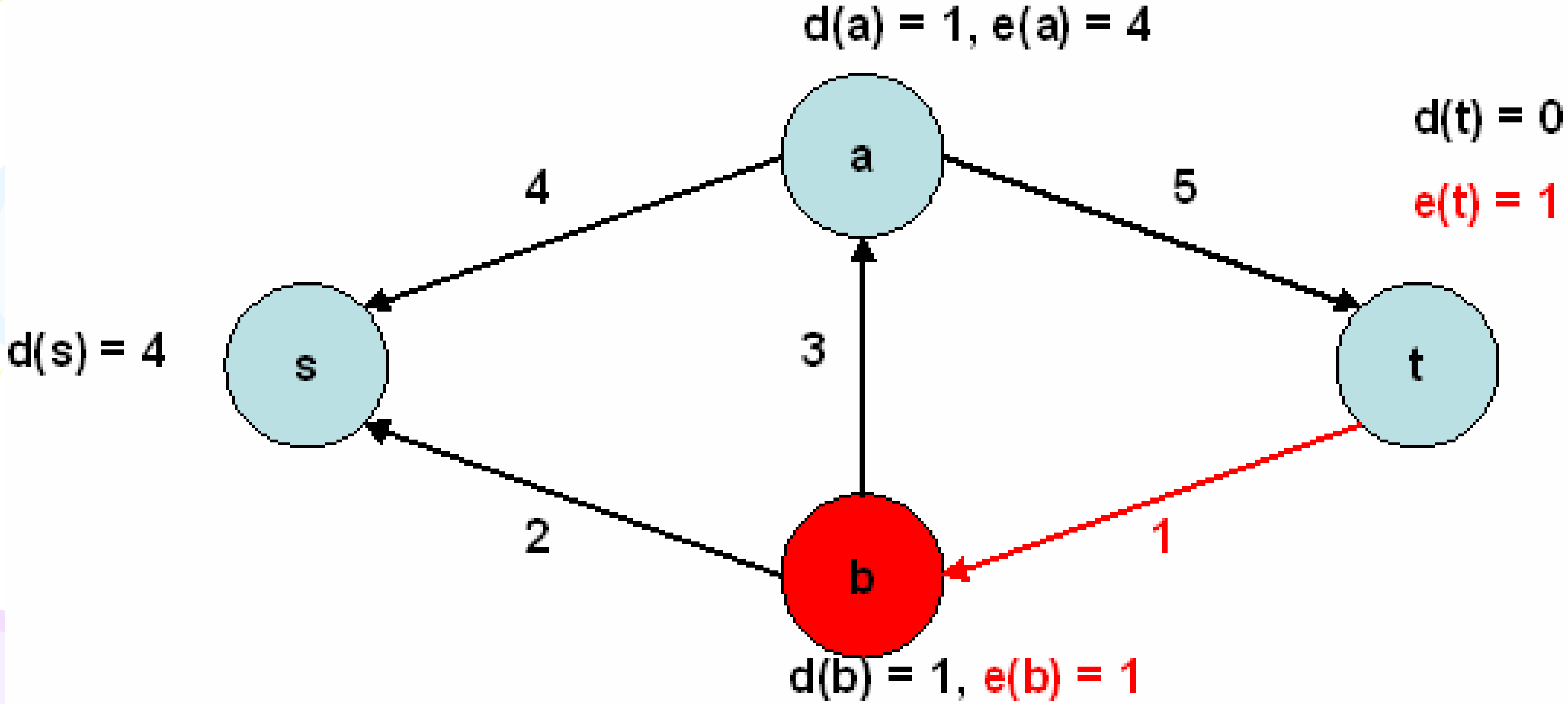
- The operation PUSH(u, v) is called a push from u to v.
- When we operate PUSH(u, v) and $c_f(u, v) = 0$ afterwards, we call edge (u, v) a saturated edge and the push from u to v a saturated push.
- Otherwise, it is an unsaturated push.

Basic Operation: Lift

If u is overflowing ($e(u) > 0$) and for all $v \in V$ $(u, v) \in E_f$ we cannot push any more units of flow to vertices neighbor to u, so we need to LIFT u (relabel).

As a result we have to update the height of u.

Example



Code for Lift (Relabel) operation

RELABEL(u)

▷ **Applies when:** u is overflowing and for all $v \in V$ such that $(u, v) \in E_f$,
we have $h[u] \le h[v]$.

▷ **Action:** Increase the height of u.

$h[u] \leftarrow 1 + \min\{h[v] : (u, v) \in E_f\}$

Generic - Preflow - Push (G)

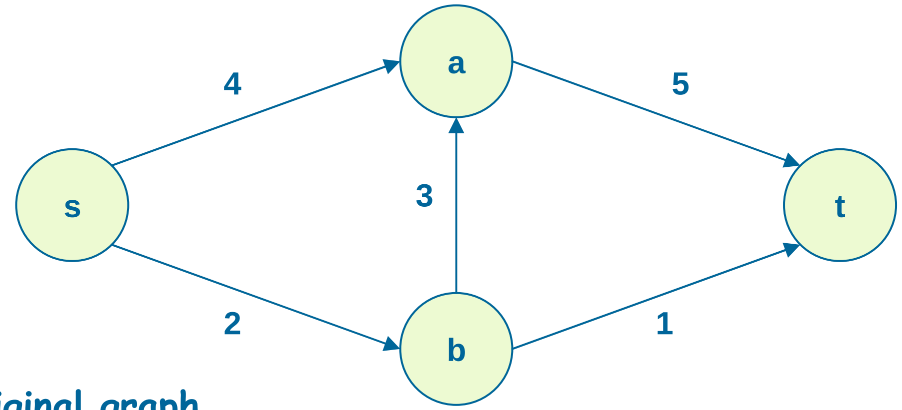
1. Initialization
   1. Set $f = 0$
   2. Compute distance labels $h(i)$ for all nodes i
   3. $f_{sj} = c_{sj}$ for all $(s, j) \in E$
   4. Set $h(s) = n$

2. while there exists an applicable push or lift operation

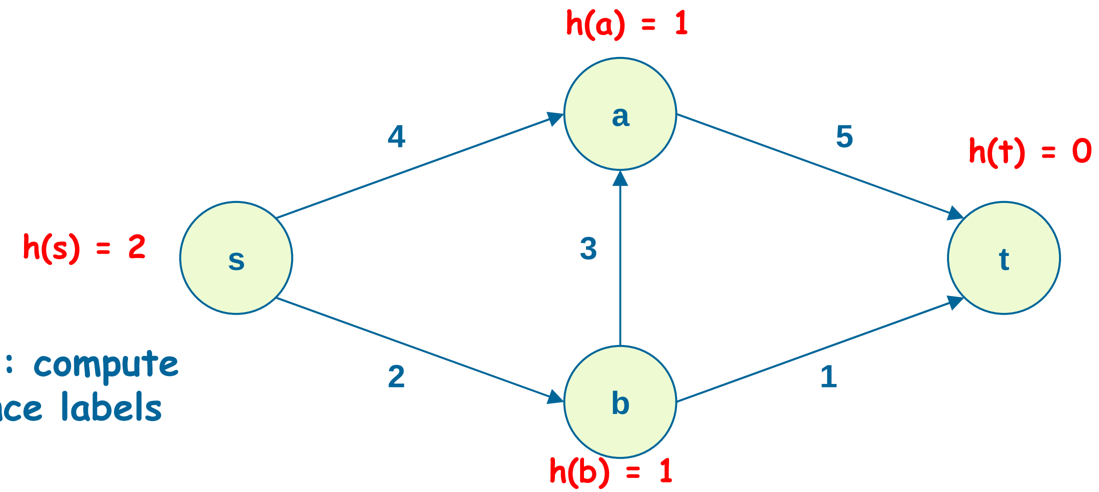
   do select an applicable push or lift operation and perform it

Preflow Push example: initialize

1. Initialization
   1. Set f = 0
   2. Compute distance labels h(i) for all nodes i



Original graph



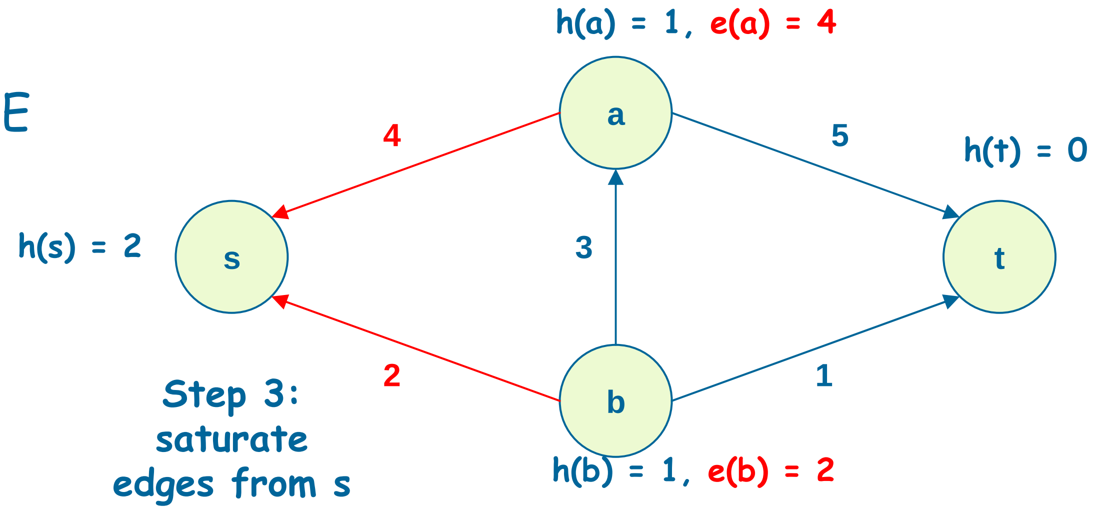
Step 2: compute distance labels

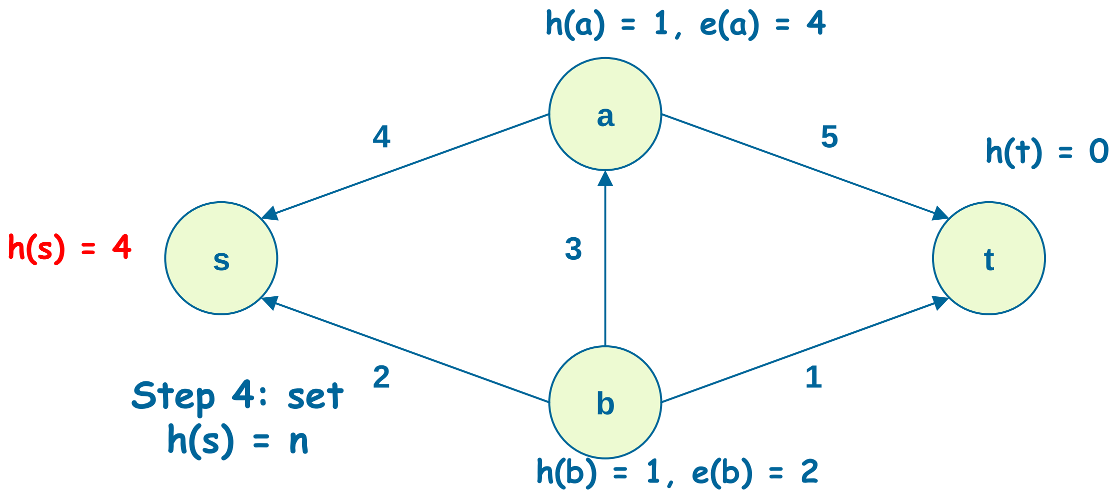
Preflow Push example: initialize

1. Initialization

3. $f_{sj} = C_{sj}$ for all $(s, j) \in E$

4. Set $h(s) = n$





Preflow Push example: while-loop

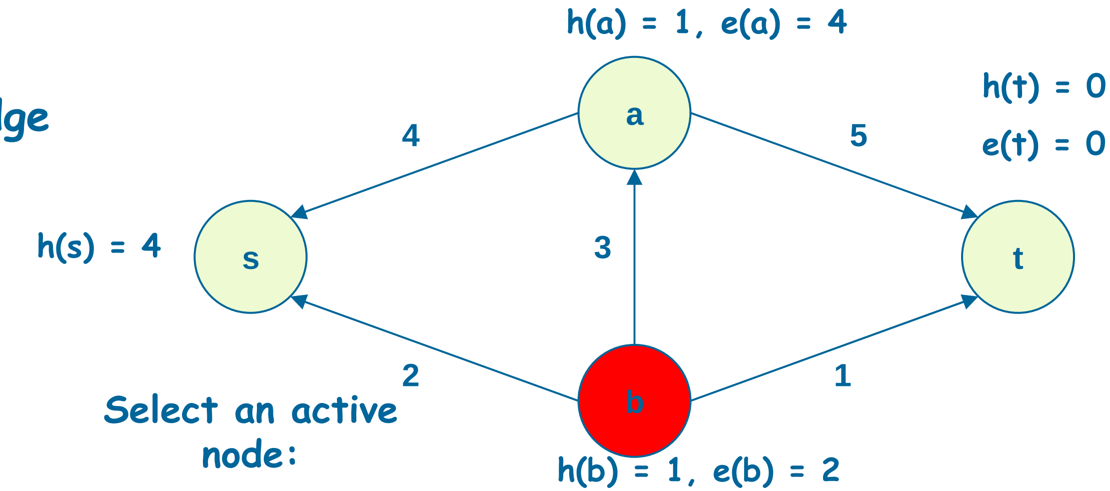
2. Push-Relabel (node i)

if there is an admissible edge (i,j)

push min(e(i), c_f(i,j)) on edge (i,j)

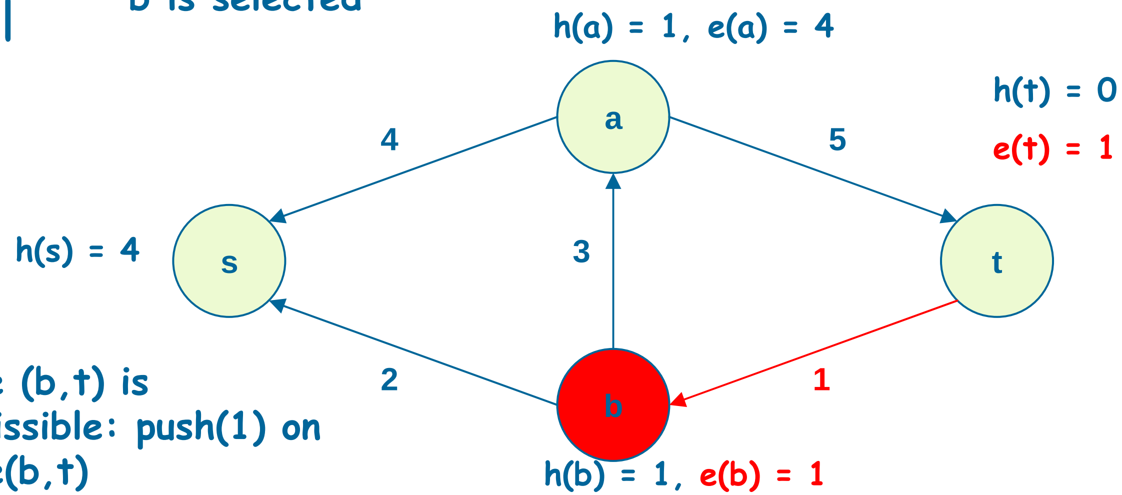
else

h(i) = 1 + min { h(j) | c_f(i,j) > 0 }



Select an active node:

b is selected



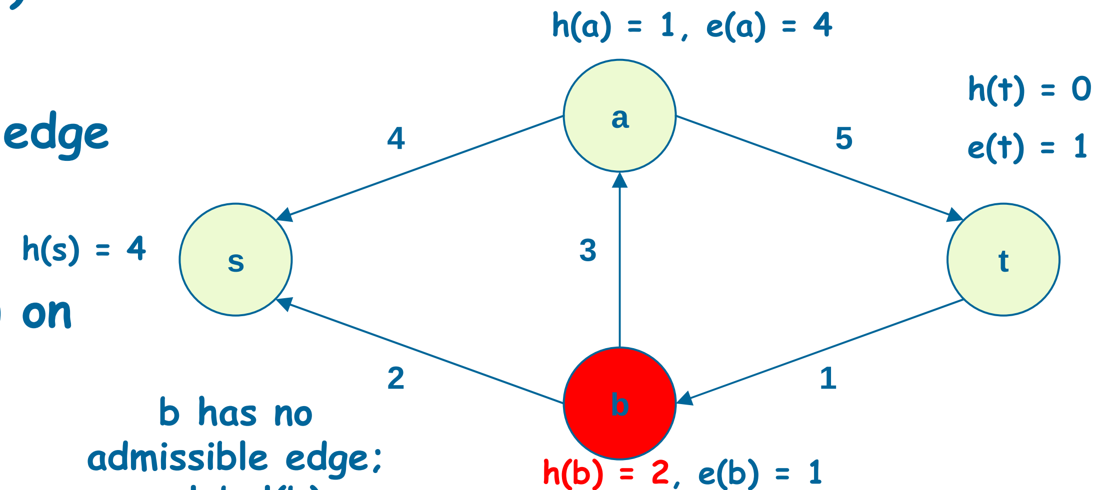
edge (b,t) is admissible: push(1) on edge(b,t)

Preflow Push example: while-loop

2. Push-Relabel(node i)

if there is an admissible edge (i, j)

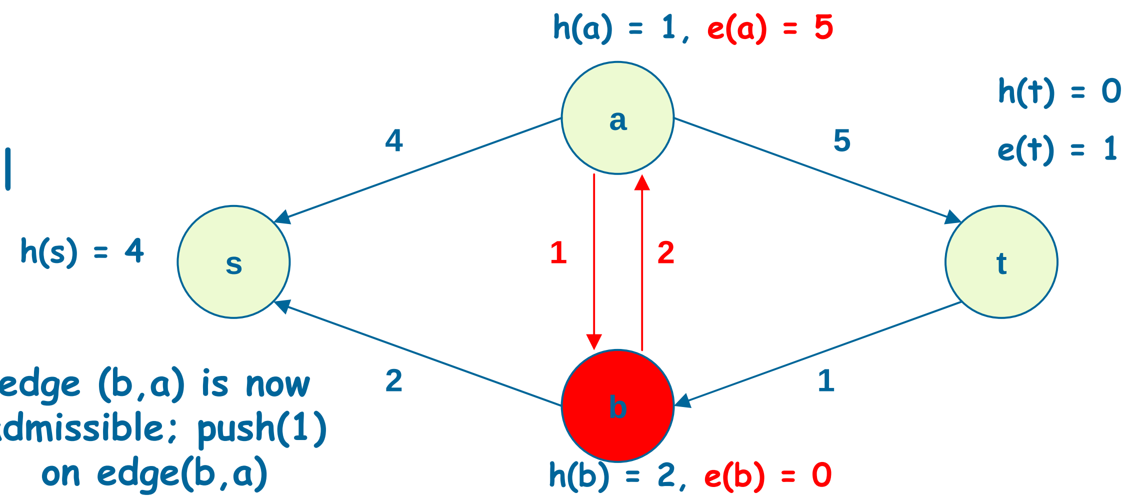
push min($e(i)$, $c_f(i,j)$) on edge (i,j)



b has no admissible edge; relabel(b)

else

$h(i) = 1 + \min \{ h(j) \mid c_f(i,j) > 0 \}$



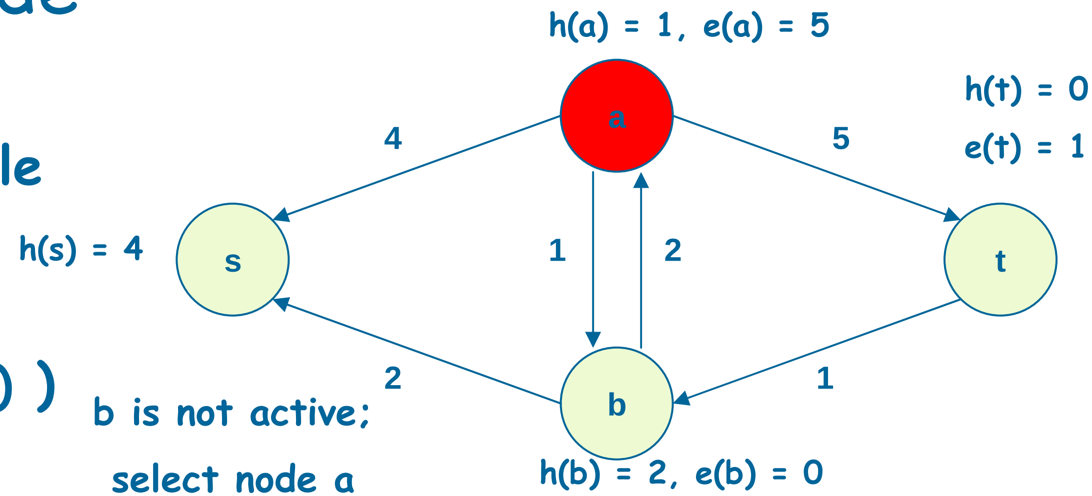
edge (b,a) is now admissible; push(1) on edge(b,a)

Preflow Push example: while-loop

2. Push-Relabel (node i)

if there is an admissible edge (i, j)

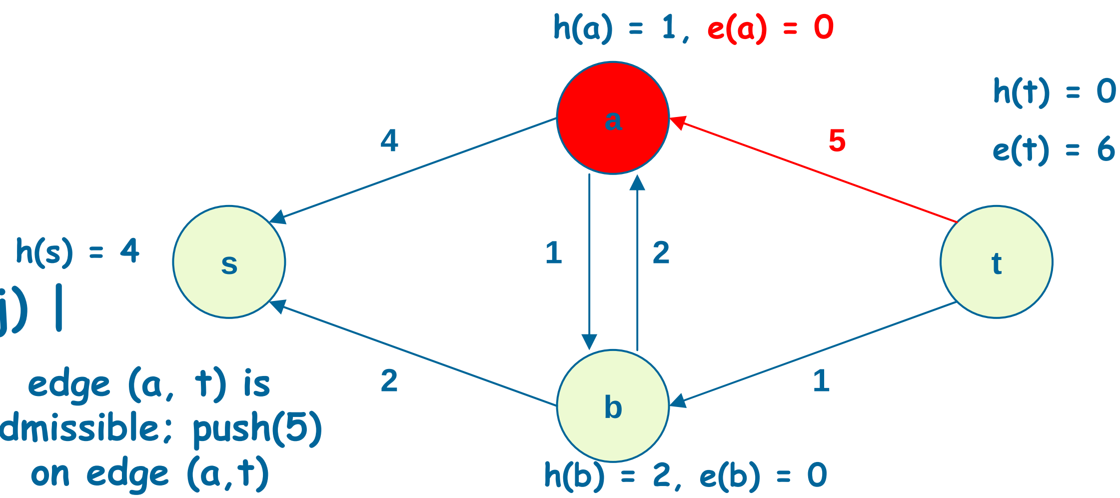
push min($e(i)$, $c_f(i,j)$) on edge (i,j)



b is not active; select node a

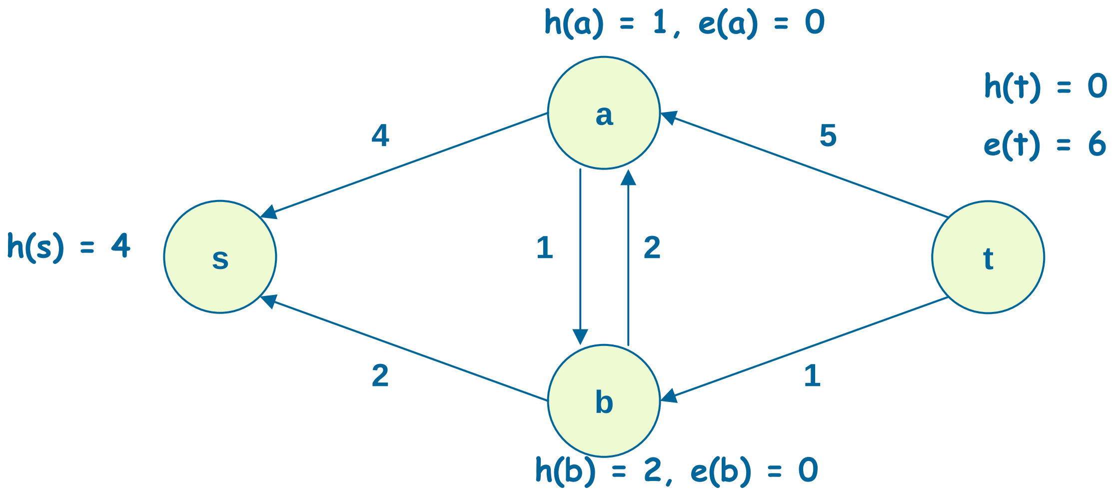
else

$h(i) = 1 + \min \{ h(j) \mid c_f(i,j) > 0 \}$



edge (a, t) is admissible; push(5) on edge (a,t)

Preflow Push example: while-loop



There are no more active nodes; the algorithm drops out of the while-loop

Correctness

- During the execution of the Preflow Push algorithm, height h(u) never decreases. Moreover whenever a lift operation is applied to u, h(u) increases.
- During the execution of the Preflow Push algorithm, the attribute h is maintained as a height function.
  - Induction to the number of basic operations performed: Initially OK.

    After Lift(u): For nodes v: $(u, v) \in E_f$ OK. For nodes w: $(w, u) \in E_f$, before Lift $h(w) \leq h(u) + 1 \rightarrow h(w) < h(u) + 1$ afterwards.

    After Push(u,v): Edge (u, v)is either added to E_f ($h(v) = h(u)-1$) or (u, v) is removed from E_f (the constraint is removed).

Correctness

- When the algorithm terminates, there is no path from s to t in the residual graph.
  - Assume a path $u_0, u_1, \ldots, u_k$ from s to t with k edges. The minimum distance between two edges must be at least 1 and wlog the path is simple, so $k < |V|$. h is a height function, so

    $h(u_i) \leq h(u_i+1) + 1$ for $i=0, \ldots, k-1$

    Combining the inequalities we obtain $h(s) \leq h(t) + k$. However $h(t) = 0$, so $h(s) \leq k < |V|$, contradiction.

Correctness

- If the algorithm terminates then the preflow f is maximum blow for G.

  If the algorithm terminates each vertex in V-{s, t} has 0 excess, so there are no overflowing vertices. h is a height function, there is no path from s to t and by max flow min cut theorem f is a maximum flow.

Termination: Bound the operations it performs

- Lift operations: $(2|V|-1)(|V|-2) < 2|V|^2$
- Saturated pushes: $2|V||E|$
- Unsaturated pushes: $2|V|^2(|V|+|E|)$

- Generic Preflow push algorithm: $O(V^2E)$

Lift – to – front algorithm

- By choosing carefully the order of the operations and managing the data structure carefully we can solve the maximum flow problem faster.

- Lift – to – front algorithm: list of vertices, list of neighbors to a vertex, discharging vertices (perform all allowed push and lift operations).

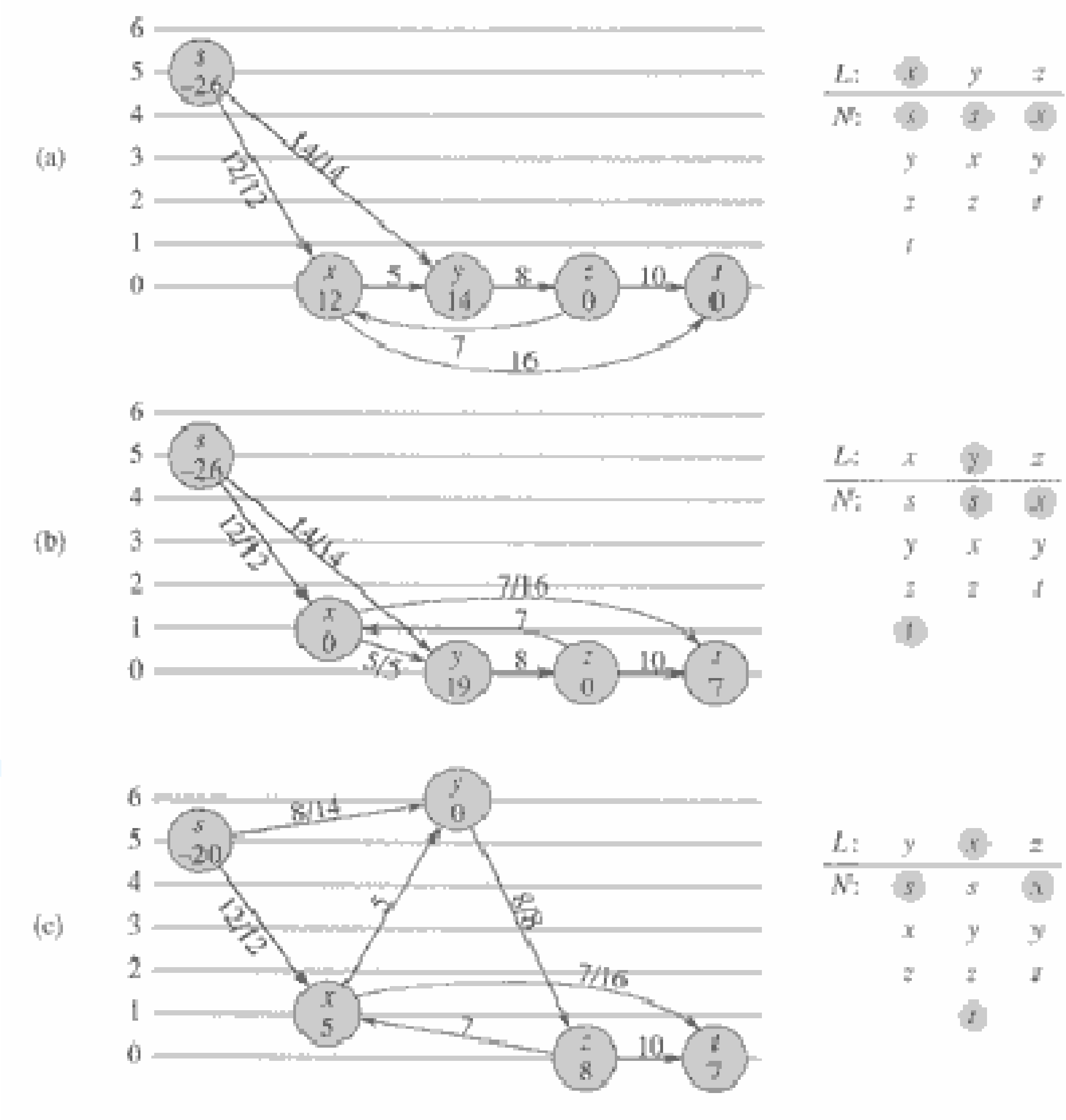
Discharge(u)

```
DISCHARGE(u)
while e[u] > 0
    do v ← current[u]
       if v = NIL
          then RELABEL(u)
               current[u] ← head[N[u]]
       elseif c_f(u, v) > 0 and h[u] = h[v] + 1
          then PUSH(u, v)
       else current[u] ← next-neighbor[v]
```

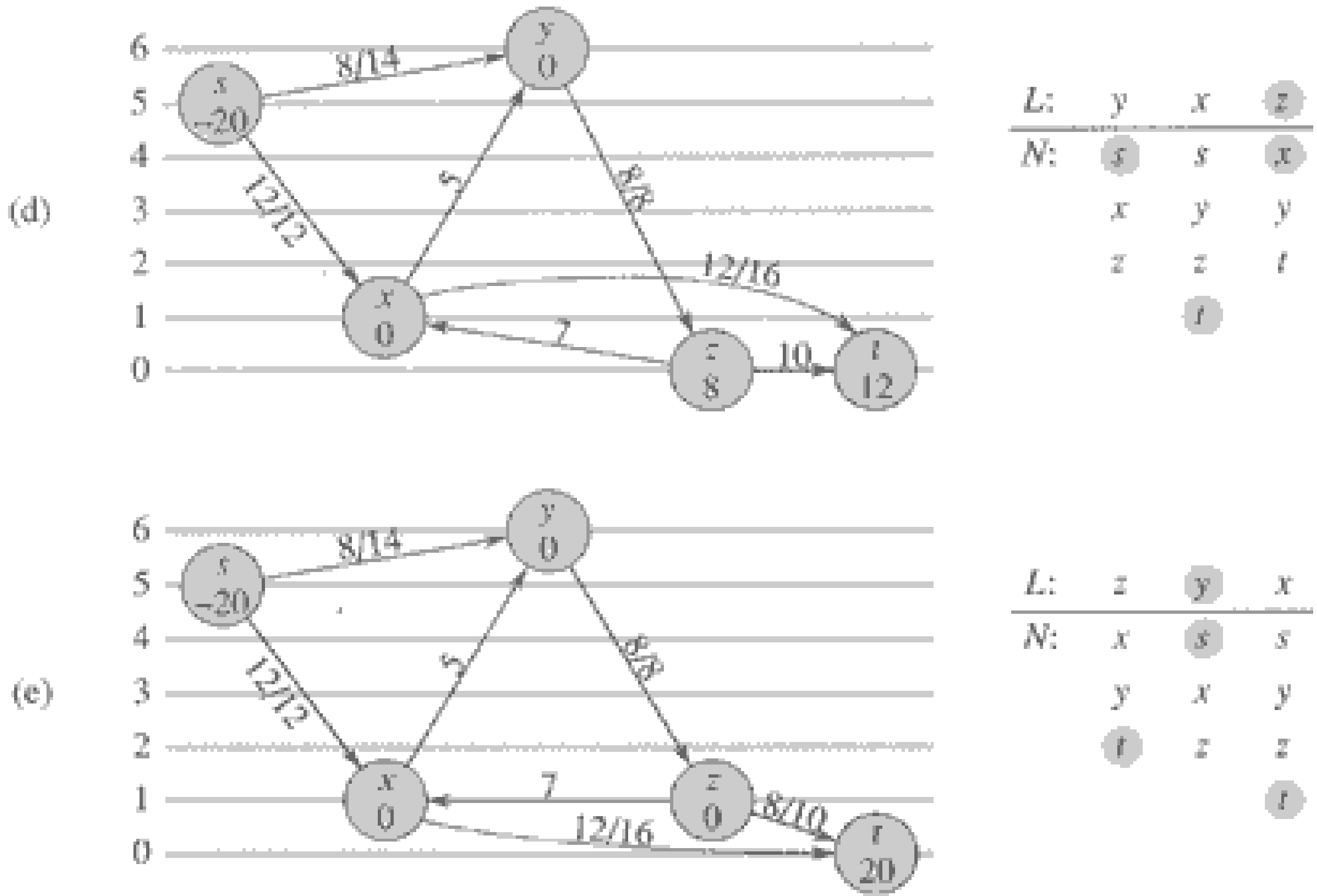
Lift – to – front algorithm

```
RELABEL-TO-FRONT(G, s, t)
INITIALIZE-PREFLOW(G, s)
L ← V[G] − {s, t}, in any order
for each vertex u ∈ V[G] − {s, t}
        do current[u] ← head[N[u]]
u ← head[L]
while u ≠ NIL
        do old-height ← h[u]
           DISCHARGE(u)
           if h[u] > old-height
              then move u to the front of list L
           u ← next[u]
```

Example



Example



(d)
6
5
4
3
2
1
0
s
−20
y
0
x
0
z
8
t
12
8/14
12/12
5
8/8
12/16
7
10
L: y x z
N: s s x
x y y
z z t
t
(e)
6
5
4
3
2
1
0
s
−20
y
0
x
0
z
0
t
20
8/14
12/12
5
8/8
7
12/16
8/10
L: z y x
N: x s s
y x y
t z z
t

Running time of Lift – to – front algorithm: $O(V^3)$

- There are $O(V^2)$ phases (because of the $O(V^2)$ lift operations). Each phase has at most $|V|$ calls of the Discharge operation. If Discharge does not perform a lift operation, the length of the list L is less than $|V|$. If it does, the next call of Discharge is in the next phase. So the while loop performs at most $O(V^3)$ operations.
- Discharge:
  - Lift operations: $O(V^2)$
  - current(u) update: $O(VE)$
  - Push operation: $O(VE)$ saturating pushes, $O(V^3)$ unsaturating pushes