Algorithms for Programming Contests - Week 05

Prof. Dr. Javier Esparza,
Vincent Fischer,
Jakob Schulz,
conpra@model.cit.tum.de

11. November 2025

Flow Network

Definition (Flow network)

A *flow network* is a tuple (V, E, c, s, t), where

- (V, E) is a directed graph
- $c : E \to \mathbb{R}_{\geq 0}$ is the *capacity function*
- $s \in V$ is a designated vertex called *source*
- $t \in V$ is a designated vertex called *target* or *sink*

W.l.o.g., we will only consider flow networks without antiparallel edges, i.e. if $(u, v) \in E$, then $(v, u) \notin E$.

Flow Network

Definition (Flow)

For a given flow network (V, E, c, s, t), let $f : E \to \mathbb{R}_{\geq 0}$. We define

- the *outflow* at $v \in V$ as $\text{out}_f(v) := \sum_{u \in vE} f(v, u)$
- the *inflow* at $v \in V$ as $\text{in}_f(v) := \sum_{u \in Ev} f(u, v)$, where $Ev := \{u \in V \mid (u, v) \in E\}$

f is called *flow* if

$$\forall (u, v) \in E: \quad 0 \leq f(u, v) \leq c(u, v) \tag{1}$$

$$\forall u \in V \setminus \{s, t\}: \quad \text{out}_f(u) = \text{in}_f(u) \tag{2}$$

Remark: (1) is called *capacity constraint*, while (2) is called *flow conservation*. A function f satisfying only (1) is called a *pre-flow*.

Maximum Flow Problem

Definition (Flow value)

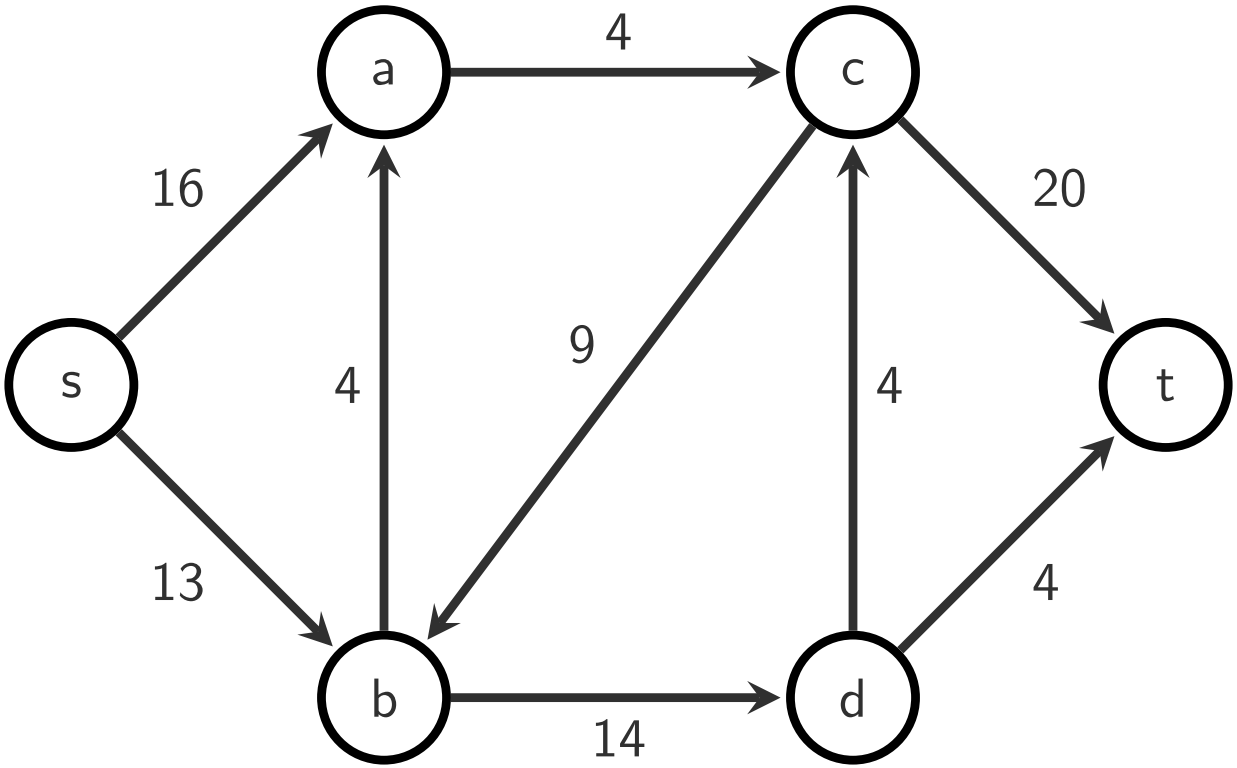
The *value* $|f|$ of a flow f is defined as

$$|f| = \text{out}_f(s) - \text{in}_f(s)$$

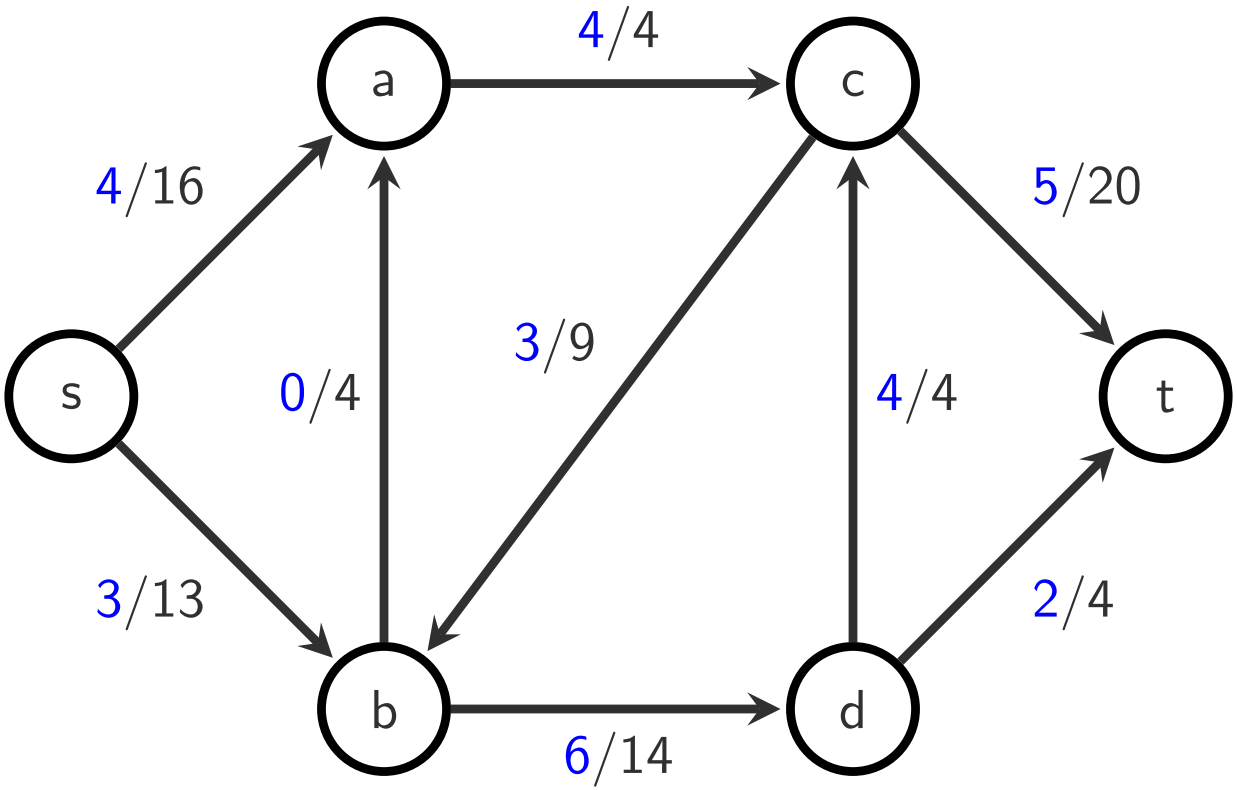
Definition (Maximum Flow Problem)

For a given flow network (V, E, c, s, t), what is a flow f with maximal value $|f|$ over all flows?

Example: Flow network



Example: Flow network with flow



$$|f| = 7$$

Maximum Flow Problem

Let (V, E, c, s, t) be a flow network.

Lemma

There exists a maximum flow.

Maximum Flow: Existence

Let (V, E, c, s, t) be a flow network.

Lemma

There exists a maximum flow.

Proof.

Note that the functions $E \to \mathbb{R}$ form a finite-dimensional vector-space. The set A of all $f : E \to \mathbb{R}$ satisfying (1) is closed, as $A = \prod_{e \in E} [0, c(e)]$. The set B of all $f : E \to \mathbb{R}$ satisfying (2) is closed, as for each $u \in V \setminus \{s, t\}$, the function $\phi_u : f \mapsto \text{out}_f(u) - \text{in}_f(u)$ is continuos, and $B = \bigcap_{u \in V \setminus \{s,t\}} \phi_u^{-1}(\{0\})$.
Therefore, the set $F = A \cap B$ of all flows $E \to \mathbb{R}$ is closed. Moreover, it is bounded (since $F \subseteq A$ and A is bounded by $\max_{e \in E} c(e)$), and hence (since E is finite) compact. Now, the function $\psi : f \mapsto |f|$ is continuous, and hence $\psi(F)$ is compact as well. Since $0 \in F$ (i.e. the constant 0-function), we have $\psi(F) \neq \emptyset$, and hence a maximum is attained.

Maximum Flow: Remarks

Let (V, E, c, s, t) be a flow network.

Lemma

The set F of all flows $E \to \mathbb{R}$ is convex, i.e. for all $f_1, f_2 \in F$ and $\alpha \in [0, 1]$, the function defined by $e \mapsto \alpha f_1(e) + (1 - \alpha) f_2(e)$ is again a flow. It has value $\alpha |f_1| + (1 - \alpha) |f_2|$.

Note: In particular, this implies that there exists a flow with any value between 0 and the maximum flow value.

Lemma

If c is integral, i.e. $c(e) \in \mathbb{Z}$ for all $e \in E$, then there exists a maximum flow that is integral.

Note: this not only means that the flow *value* is integral, but also *all* $f(e)$!

Reductions to maximum flow

Hints

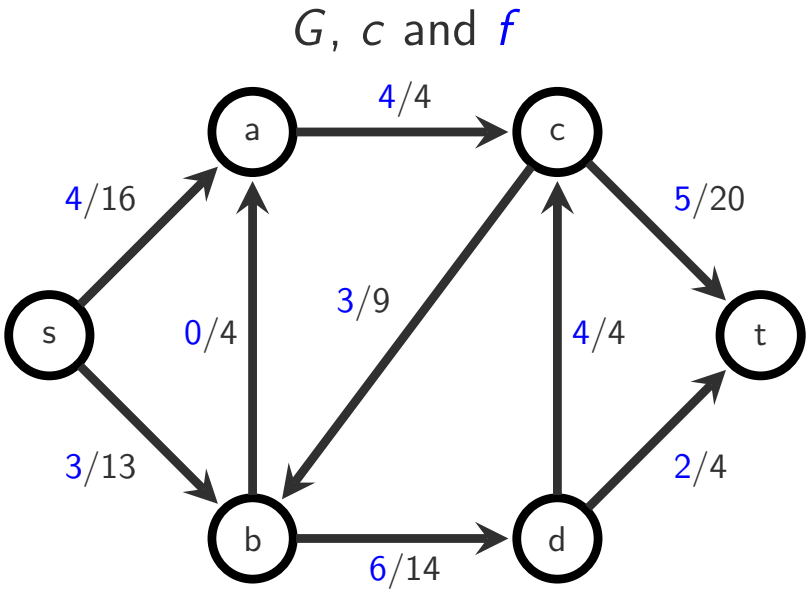
- Minimum flow: send no flow at all.
- Multiple sources/sinks: add super-source/sink and edges with infinite capacity to other sources/sinks. ("infinite" can be replaced by any bound on $|f|$ such as $\text{out}_c(s)$)
- Sources/sinks with supply constraints (e.g. "source can supply at most 5"): add super-source/sink with edges to sources/sinks with corresponding capacity.
- Vertex capacities: Split up vertex with edge of that capacity in between.
- Antiparallel edges: Insert vertex in between one edge.
- Undirected edges: Convert to two antiparallel directed edges.

Maximum Flow: Towards First Algorithm

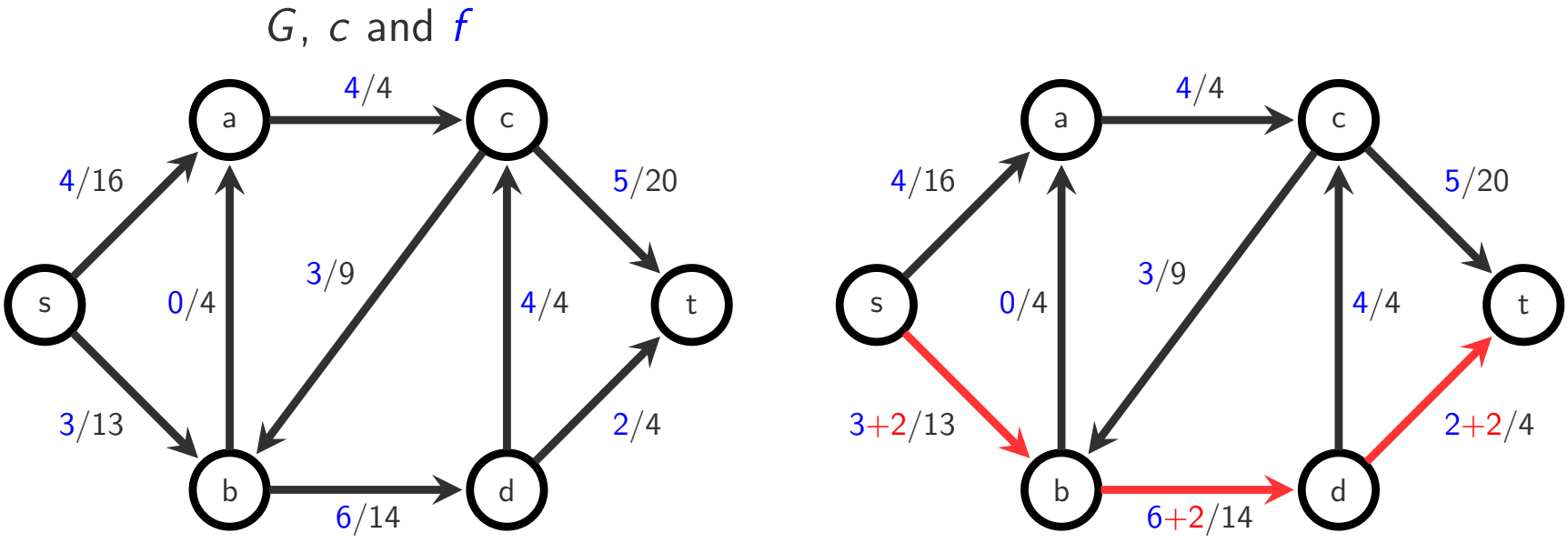
First, we look at the *Ford-Fulkerson-Algorithm*. The idea:

- Start with the 0 flow, i.e. $f(e) = 0$ for all e.
- Iteratively improve f by finding a path along which the flow can be increased ("augmenting path").

Example: Augmenting path

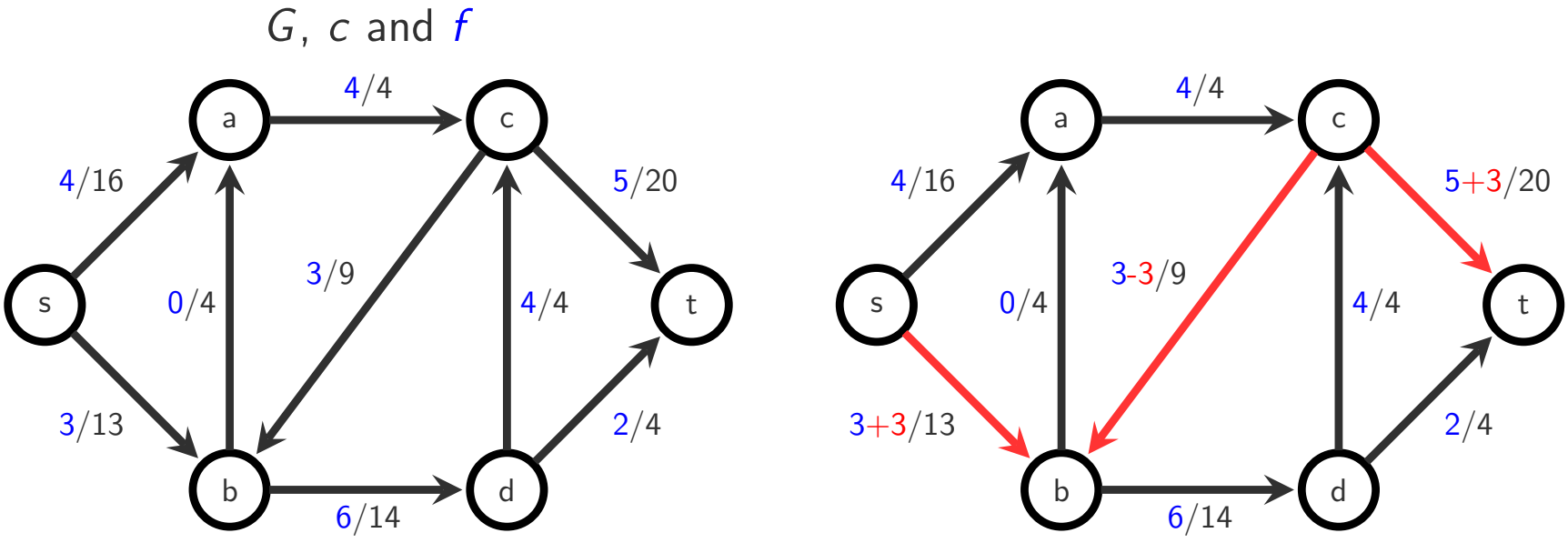


Example: Augmenting path



"Augmenting path" p: can increase flow by 2

Example: Augmenting path



"Augmenting path" p: can increase flow by 3

Augmenting Paths

Definition (Residual capacity and residual network)

For a given flow network G and a flow f, and a pair of vertices $u, v \in V$, the *residual capacity* $c_f(u, v)$ is defined by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

The *residual network* of G induced by f is $G_f = (V, E_f)$, where

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

Augmenting Paths

Definition (Augmenting Path)

Given a flow network G and a flow f, an *augmenting path* p is a simple path in the residual network G_f from s to t.
The *residual capacity* of an augmenting path p is given by

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

The flow f_p of an augmenting path p in G_f is defined as

$$f_p(u, v) = \begin{cases} c_f(p) & \text{if } (u, v) \text{ is on } p \\ 0 & \text{otherwise} \end{cases}$$

Augmenting flow

Definition (Augmenting flows)

If f is a flow in a flow network G and f' is a flow in the corresponding residual network G_f, then the *augmentation* $f \uparrow f'$ of f by f' is a flow in G defined as

$$(f \uparrow f')(u, v) = f(u, v) + f'(u, v) - f'(v, u) \qquad \text{for } (u, v) \in E$$

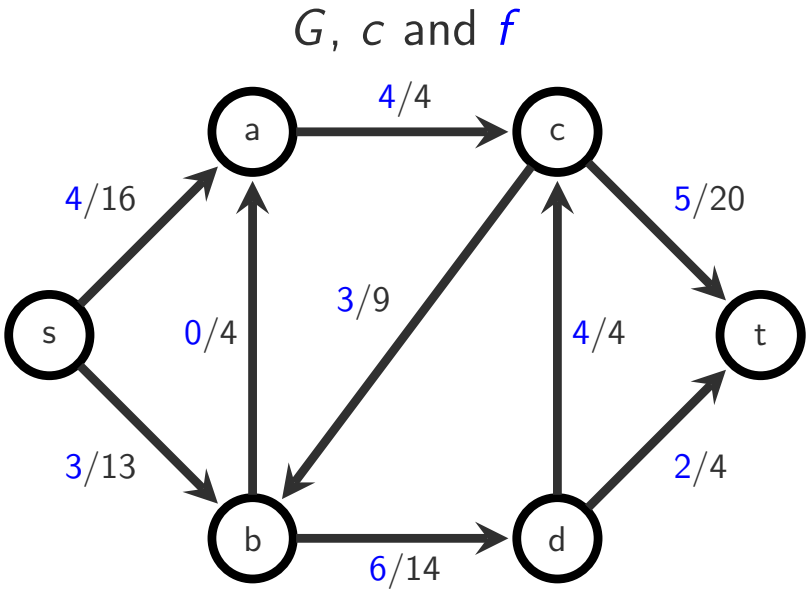
where $f'(e) := 0$ for $e \notin E_f$.

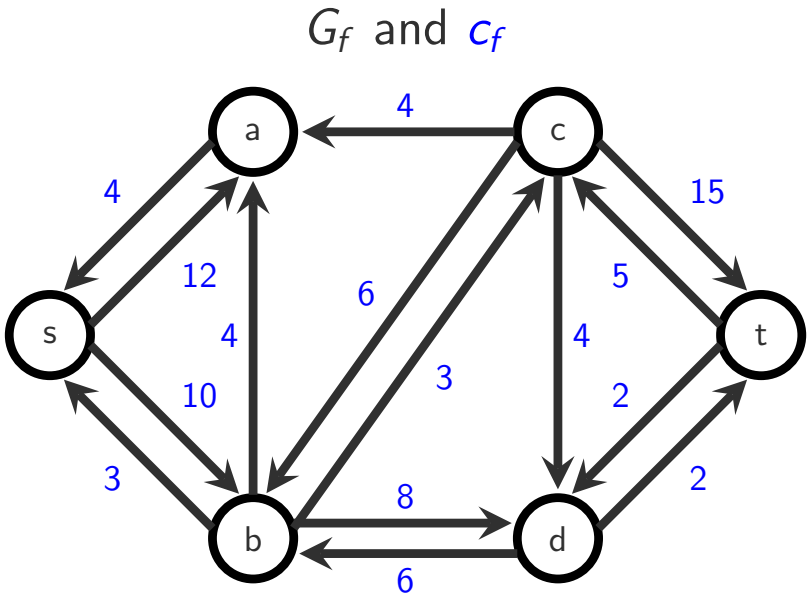
Lemma

If f is a flow in a flow network G and f' is a flow in the corresponding residual network G_f, then $f \uparrow f'$ is also a flow in G and

$$|f \uparrow f'| = |f| + |f'|$$

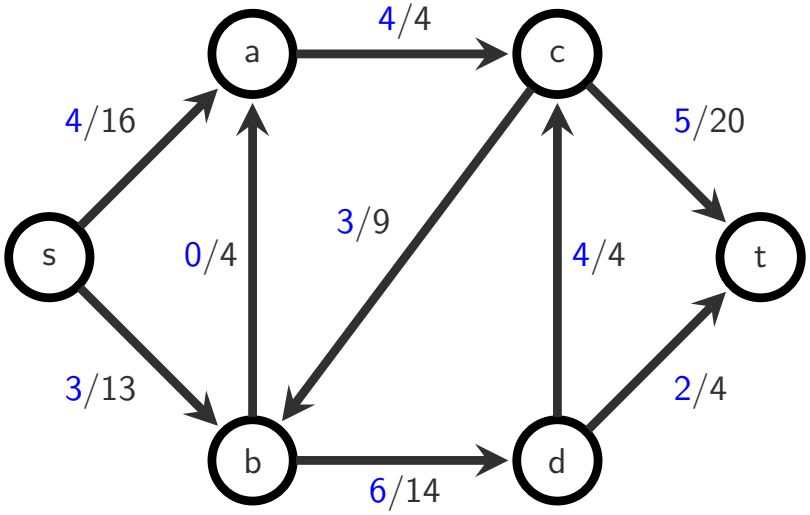
Example: Residual network



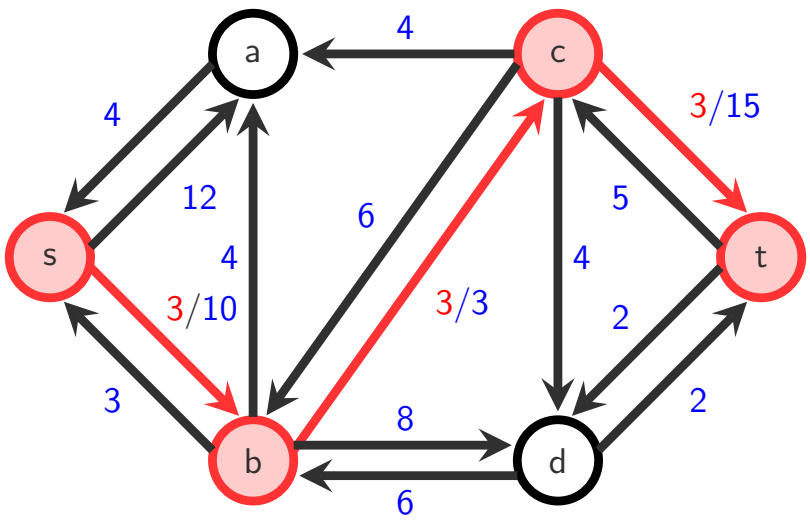


Example: Augmenting path

G, c and f



G_f, c_f, p and f_p

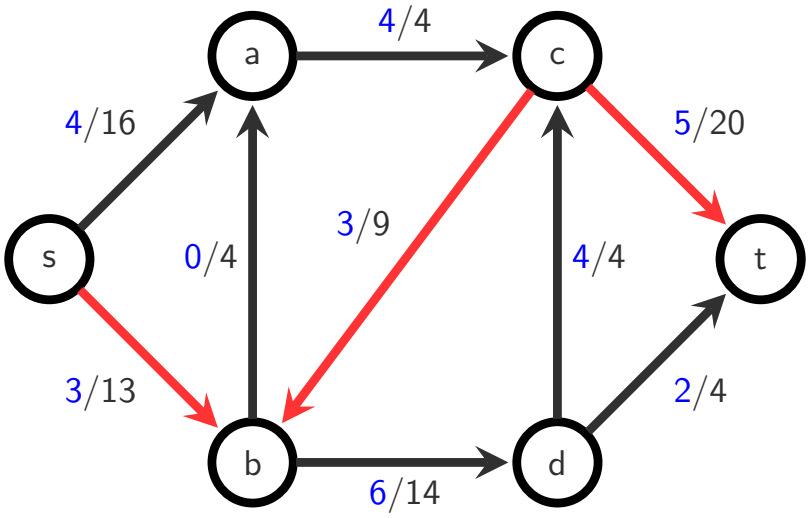


$$p = \text{sbct}$$

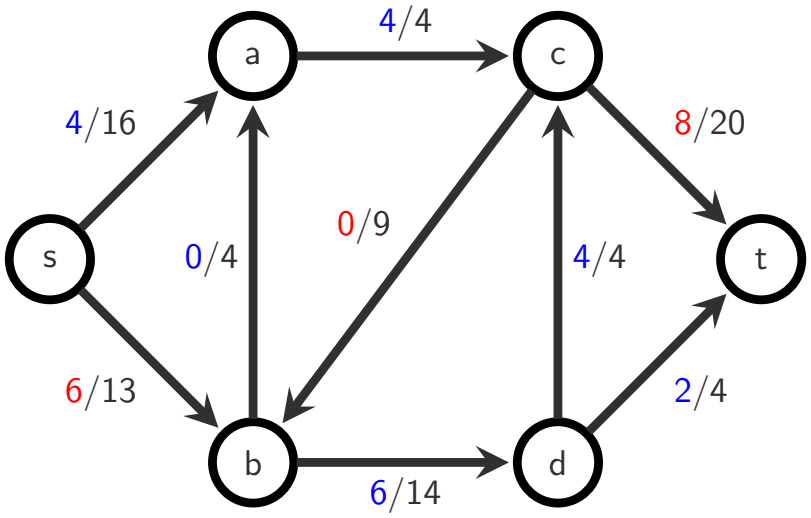
$$c_f(p) = 3$$

Example: Augmenting flow

G, c, f and p



G, c and $f' = f \uparrow f_p$



$$p = \text{sbct}$$

$$c_f(p) = 3$$

Augmenting path algorithm (Ford-Fulkerson algorithm)

Algorithm 1 Ford-Fulkerson algorithm

▷ Initialize flow to 0 ($f \leftarrow 0$)
for $(u, v) \in E$ **do**
    $f(u, v) \leftarrow 0$
end for
while there exists a path p from s to t in the residual network G_f **do**
    ▷ Augment f by f_p ($f \leftarrow f \uparrow f_p$)
    $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is on } p\}$
    for each edge (u, v) in p **do**
        if $(u, v) \in E$ **then**
            $f(u, v) \leftarrow f(u, v) + c_f(p)$
        else
            $f(v, u) \leftarrow f(v, u) - c_f(p)$
        end if
    end for
end while

Max-flow min-cut

Definition (Cut)

For a given flow network G with source s and target t, a *cut* $C = (S, T)$ is a partititon of V into two subsets S and T such that $s \in S$ and $t \in T$. The *capacity* $c(S, T)$ of a cut (S, T) is defined as

$$c(S, T) = \sum_{(u,v) \in (S \times T) \cap E} c(u, v)$$

If a flow f is given, the *net flow* across the cut (S, T) is defined as

$$f(S, T) = \sum_{(u,v) \in (S \times T) \cap E} f(u, v) - \sum_{(u,v) \in (T \times S) \cap E} f(u, v)$$

Lemma

Let $G = (V, E, c, s, t)$ be a flow network and f be a flow in G. Moreover, let (S, T) be a cut. Then $|f| = f(S, T) \leq c(S, T)$.

Lemma

Let $G = (V, E, c, s, t)$ be a flow network and f be a flow in G. TFAE:

1. *f is a maximum flow*
2. *There are no augmenting paths*
3. *There exists a cut (S, T) s.t. $|f| = c(S, T)$*

Corollary

If the Ford-Fulkerson Algorithm terminates, the result is correct.

Corollary (Max-flow min-cut theorem)

The maximum value $|f|$ over all flows f is equal to the minimum capacity $c(S, T)$ over all cuts (S, T).

Running time

Running time (and even termination!) depends on the choice of the augmenting paths and how they are found:

- Choosing any path (with DFS):
  - If capacities can be irrational, algorithm might not terminate.
  - If capacities are integral: Each iteration increases flow by at least 1. Hence, at most U iterations, where U is the value of the maximum flow. Complexity $\mathcal{O}(|E|\,U)$.
- Shortest path by number of edges (with BFS): Edmonds-Karp algorithm. Complexity $\mathcal{O}(|V|\,|E|^2)$ (at most $\frac{|V||E|}{2}$ iterations).
- "Widest paths", i.e. with maximal residual capacity (with modified Dijkstra): Complexity $\mathcal{O}((|E| + |V| \log |V|)\,|E| \log U)$ (at most $\mathcal{O}(|E| \log U)$ iterations) when capacities are integral

Better:

- Dinic's algorithm using "blocking flows". Complexity $\mathcal{O}(|V|^2\,|E|)$. (Karzanov's variant: $\mathcal{O}(|V|^3)$)
- "Push-Relabel Algorithm" in $\mathcal{O}(|V|\,|E| \log \frac{|V|^2}{|E|})$ resp. $\mathcal{O}(|V|^3)$

Dinic's algorithm: Trivia

Complete abstract of paper by Shimon Even (1976):

Abstract: Recently A.V. Karzanov improved Dinic's algorithm to run in time $O(n^3)$ for networks of n vertices. For the benefit of those who do not read Russian, the Dinic-Karzanov algorithm is explained and proved.

In addition to being the best algorithm known for network flow, this algorithm is unique in that it does not use path augmentation.

Later in the paper:

I do not read Russian and could not read the more detailed available material on the work of Dinic and Karzanov [5]. The short descriptions in Soviet Math. Dokl. arehard to read because they are translations and lack details. I have rediscovered Dinic's algorithm with J. Hopcroft in 1972 and have reconstructed Karzanov's result with the help of A. Itai. The proofs are my own and I do not know if they differ from those of Dinic and Karzanov. In any case I alone am responsible for any mistakes that may be in my exposition.

Dinic's algorithm: Trivia

Dinitz (2006), Dinitz' Algorithm: The Original Version and Even's Version

> *The reader may be aware of the so called "Dinic's algorithm" [...]. [...] Shimon Even and [...] Alon Itai [...] were very curious and intrigued by the two new network flow algorithms: mine and that of Alexander Karzanov [...]. It was very difficult for them to decipher these two papers (each compressed into four pages, to meet the page restriction of the [...] journal Doklady). [...] Even and Itai understood both papers, except for the layered network maintenance issue. [...] "Dinic's algorithm" was a great success and gained a place in the annals of the computer science community. Hardly anyone was aware that the algorithm, taught in many universities since then, is not the original version [...]. [...] Also, its name was rendered incorrectly as [dinik] instead of [dinits]. After more than 15 years [...] I finally explained the original version of my algorithm [...] to Shimon Even.*

Blocking flow

Definition (Level graph)

Given a residual network $G_f = (V, E_f)$, let $d_{G_f}(s, v)$ be the length of the shortest path from s to v in G_f (by number of edges).
The *level graph* of G_f is the graph $G_L = (V, E_L, c_L)$, where

$$E_L = \{(u, v) \in E_f : d_{G_f}(s, v) = d_{G_f}(s, u) + 1\}$$

$$c_L(u, v) = \begin{cases} c_f(u, v) & \text{if } (u, v) \in E_L \\ 0 & \text{otherwise} \end{cases}$$

Definition (Blocking flow)

A *blocking flow* in the level graph G_L is a flow f such that every path from s to t in G_L contains a saturated edge, i.e., an edge (u, v) with $f(u, v) = c_L(u, v)$.

Dinic's algorithm

Algorithm 2 Dinic's algorithm

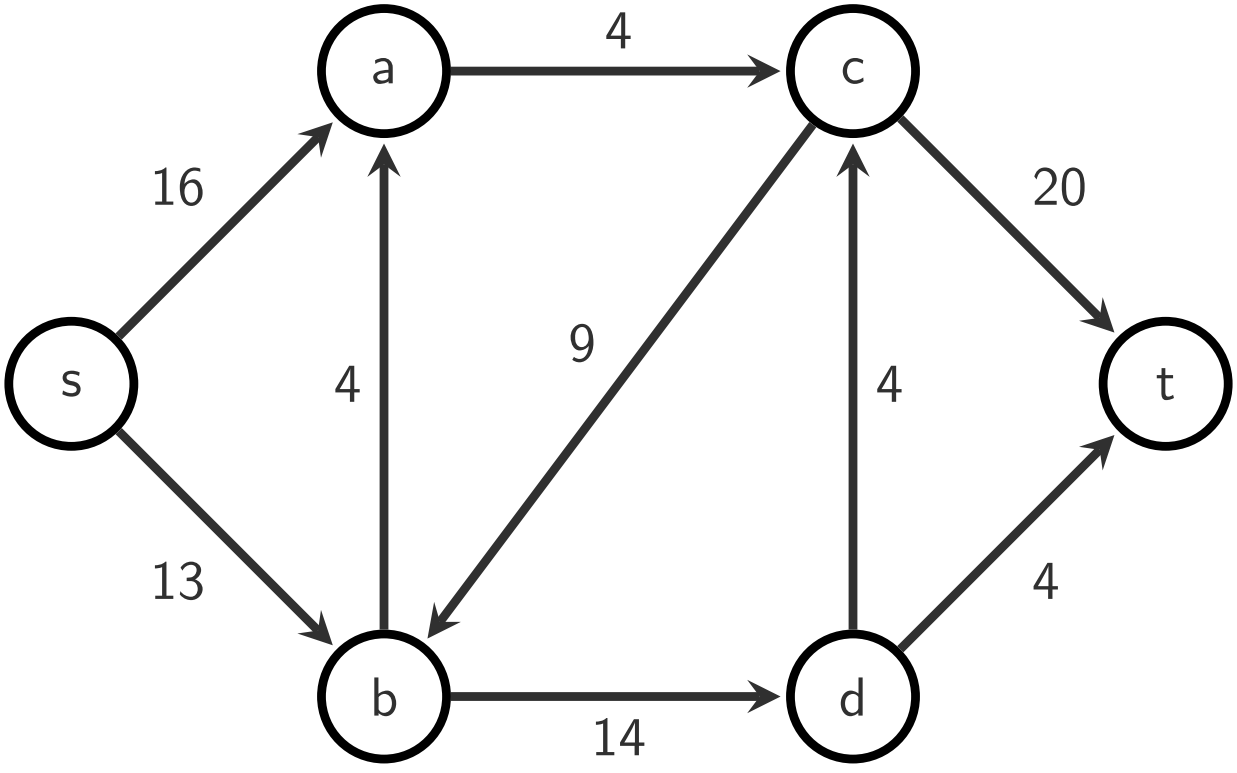
$f \leftarrow 0$
while there exists a blocking flow f' in G_L with $|f'| > 0$ **do**
    $f \leftarrow f \uparrow f'$
end while

Finding a blocking flow

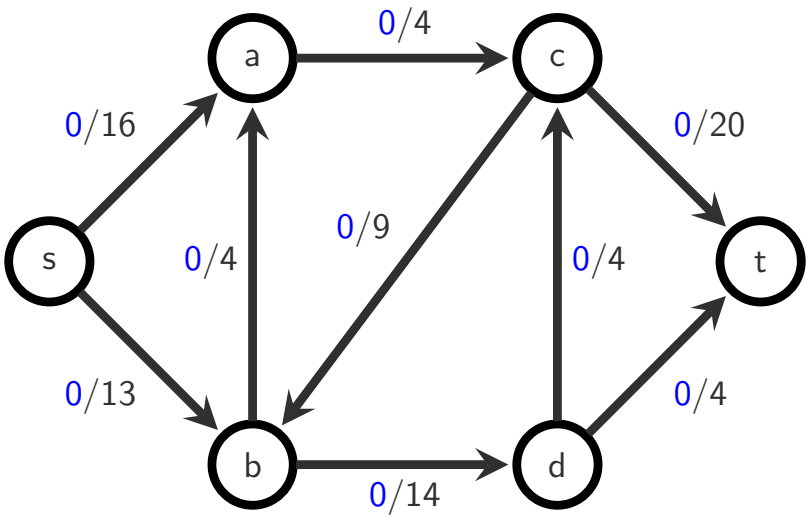
Algorithm 3 Finding blocking flows via DFS

$f' \leftarrow 0$; $p \leftarrow s$; $u \leftarrow s$
while $u \neq t$ **do**
  while there is an edge $(u, v) \in E_L$ with $f'(u, v) < c_L(u, v)$ **do**
    $p \leftarrow pv$
    $u \leftarrow v$
  end while
  if $u = t$ **then**
    $f' \leftarrow f' \uparrow f_p$; $p \leftarrow s$; $u \leftarrow s$
  else if $u = s$ **then**
    return f'
  else
    let (v, w) be the last edge on p; delete w from p
    delete (v, w) from E_L; $u \leftarrow v$
  end if
end while

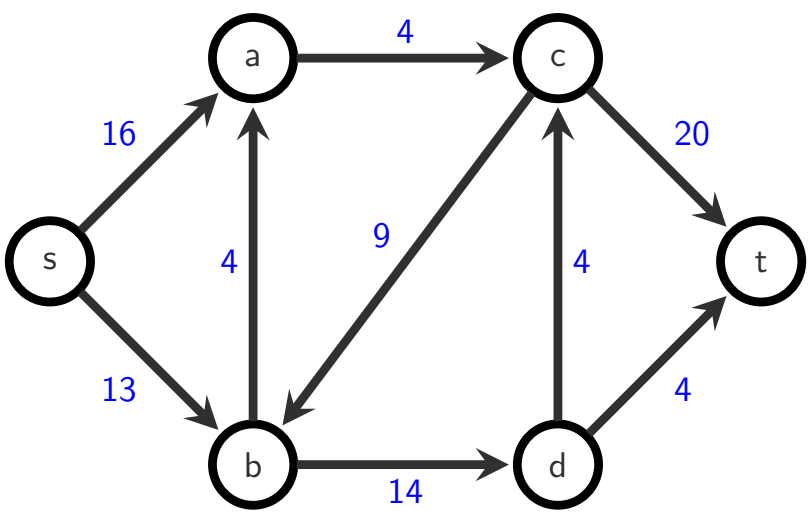
Dinic's algorithm (example)



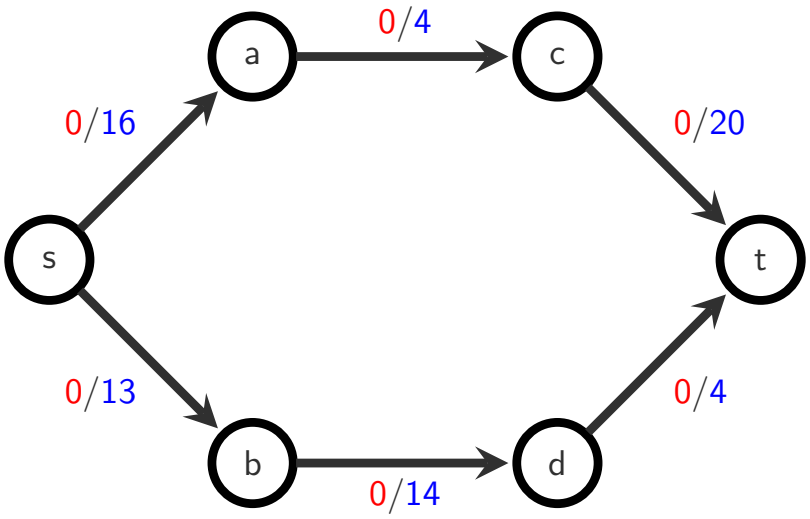
G, c and f

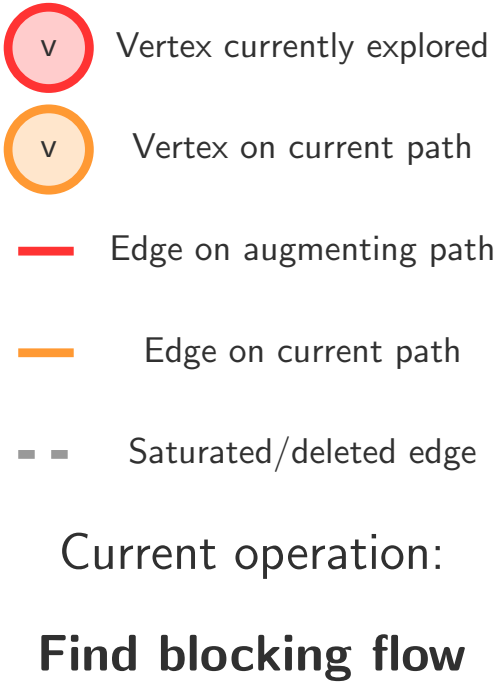


G_f and c_f

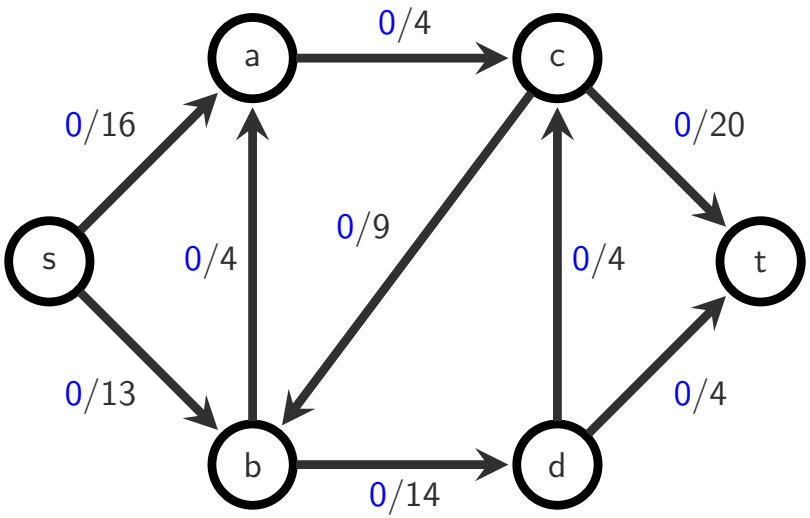


G_L, c_L and f'

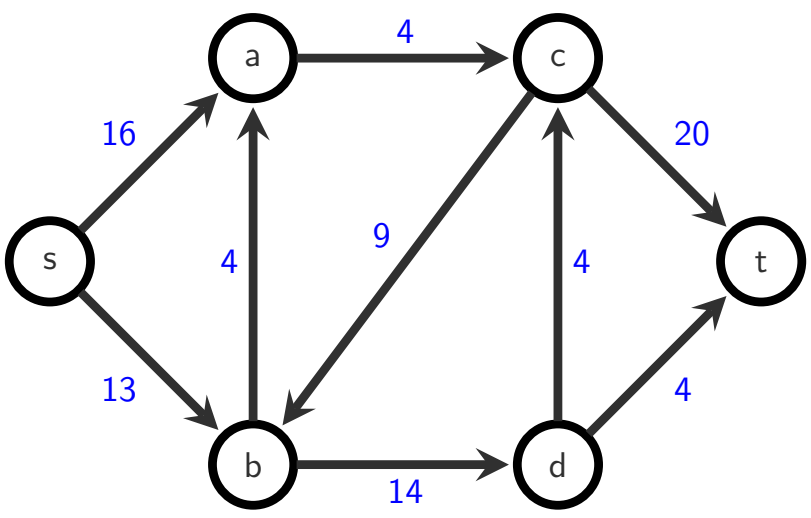




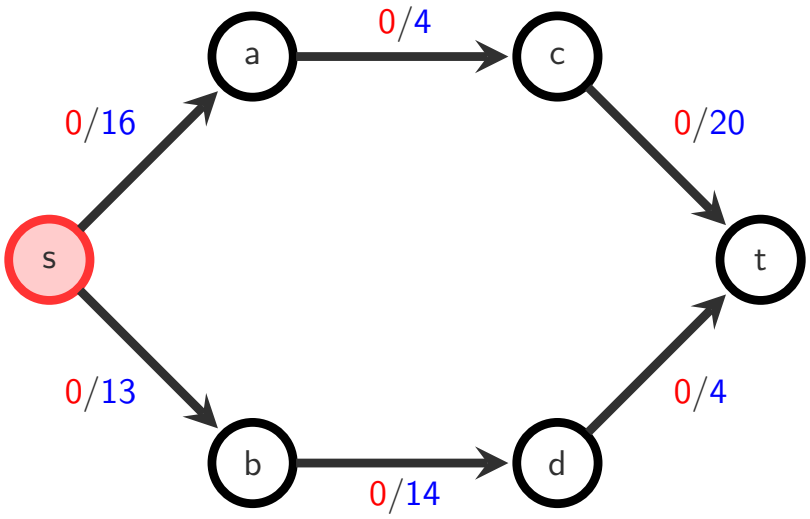
G, c and f



G_f and c_f



G_L, c_L and f'



v Vertex currently explored

v Vertex on current path

Edge on augmenting path

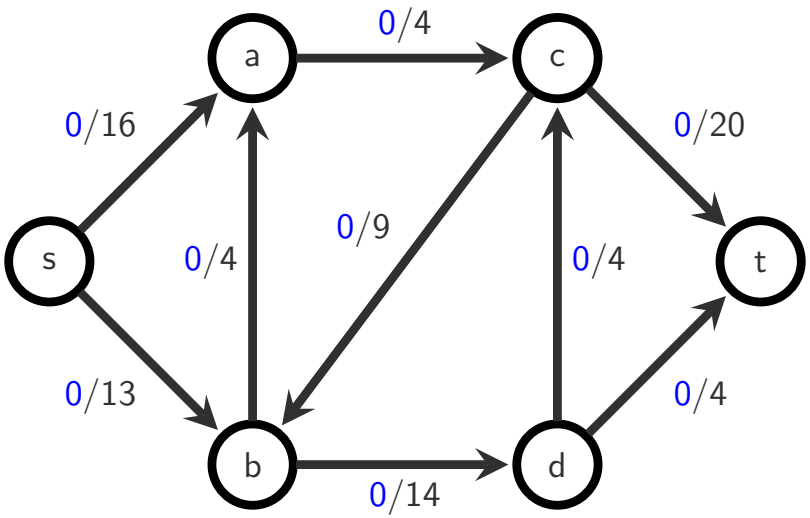
Edge on current path

Saturated/deleted edge

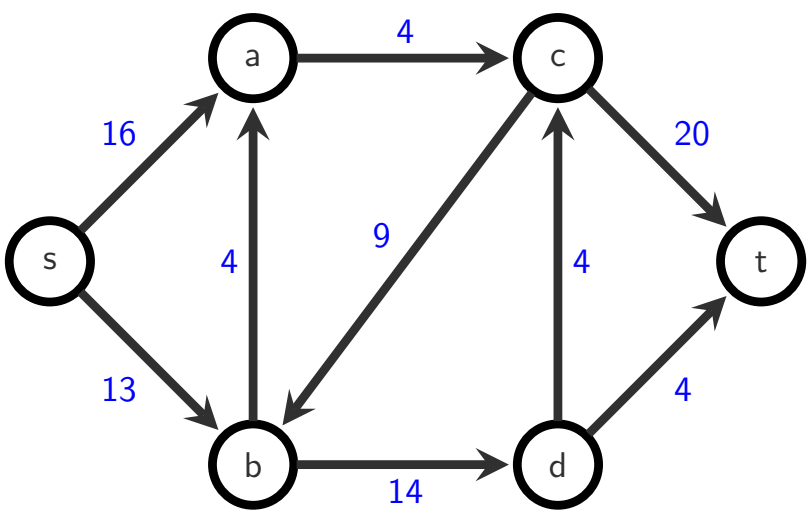
Current operation:

Find augmenting path

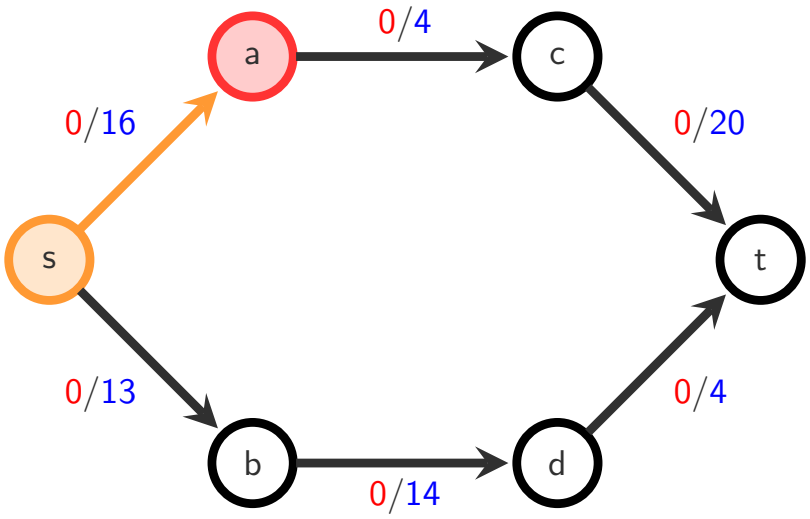
G, c and f

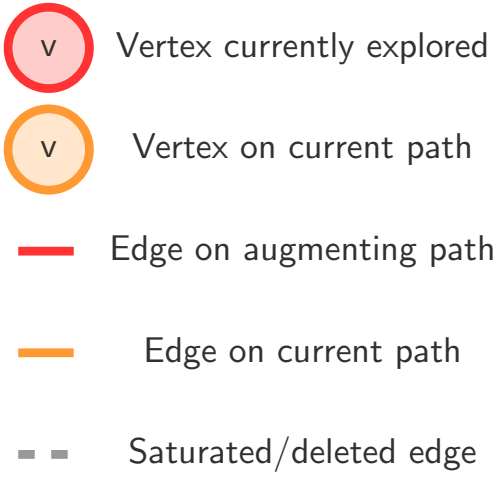


G_f and c_f



G_L, c_L and f'

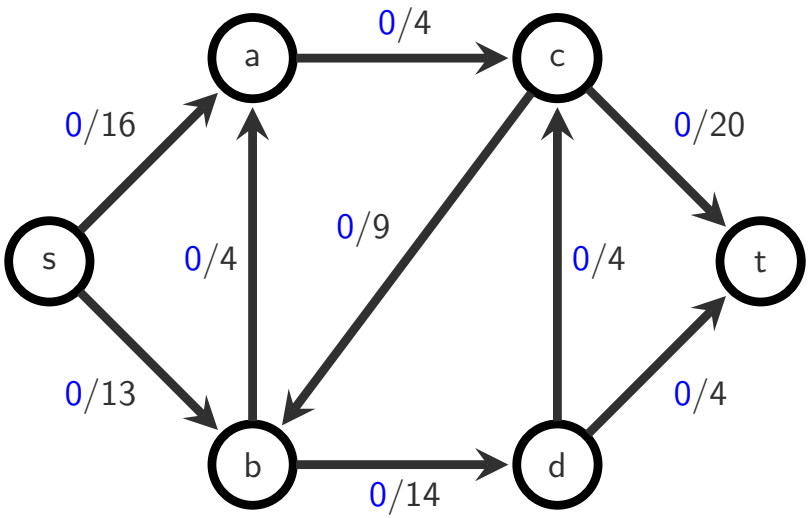




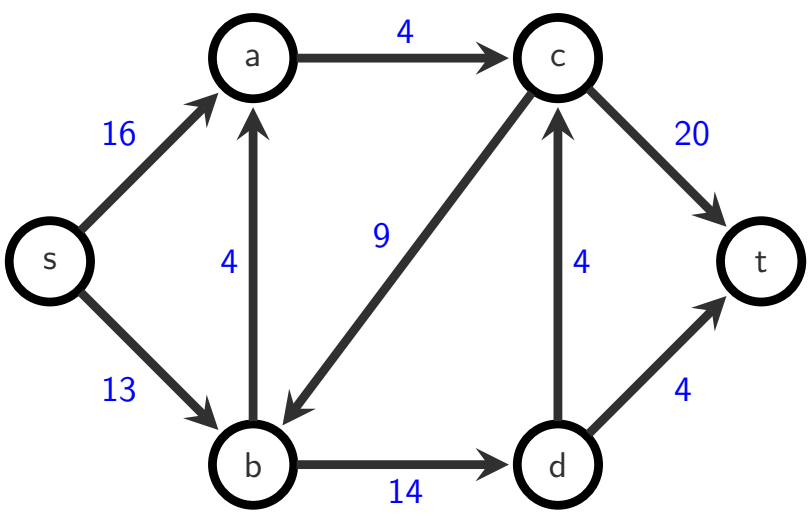
Current operation:

Find augmenting path

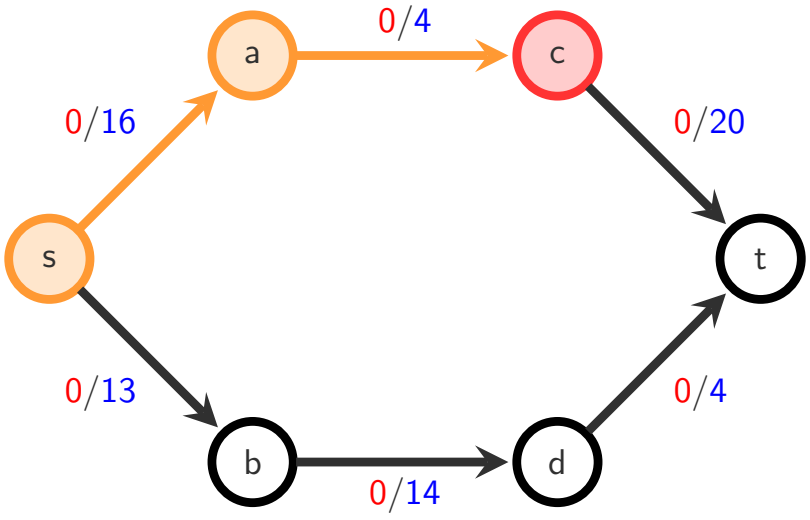
G, c and f

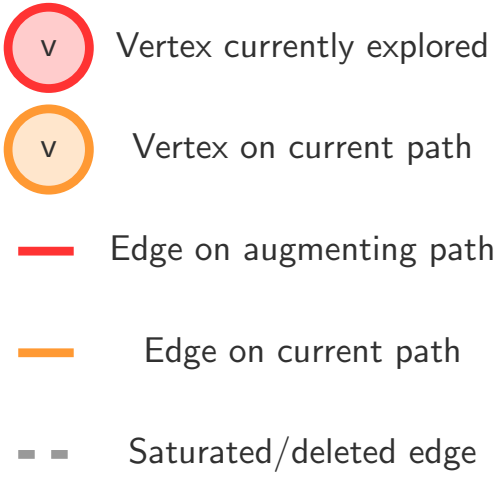


G_f and c_f



G_L, c_L and f'

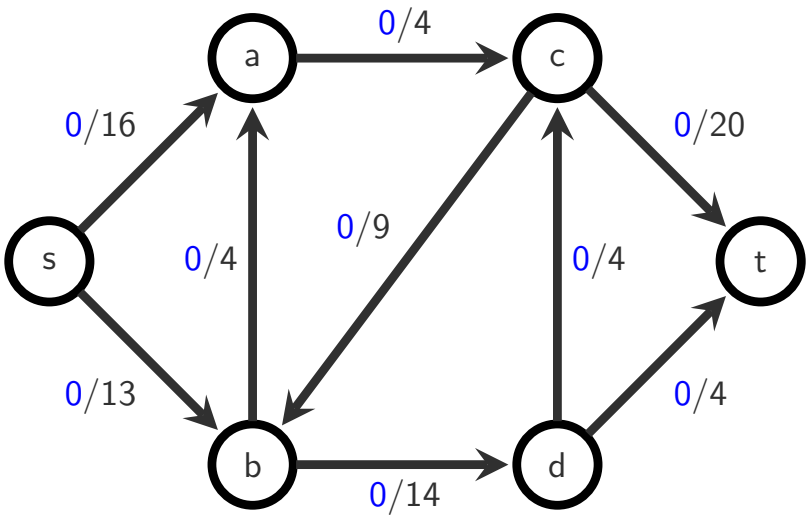




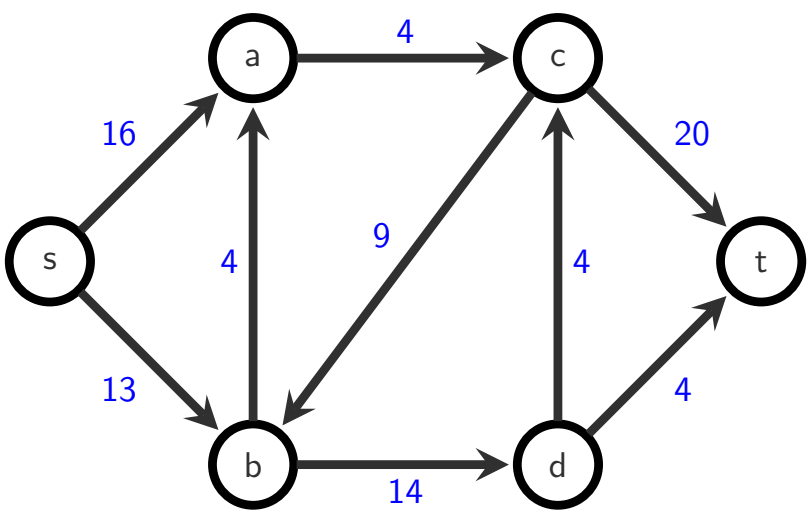
Current operation:

Find augmenting path

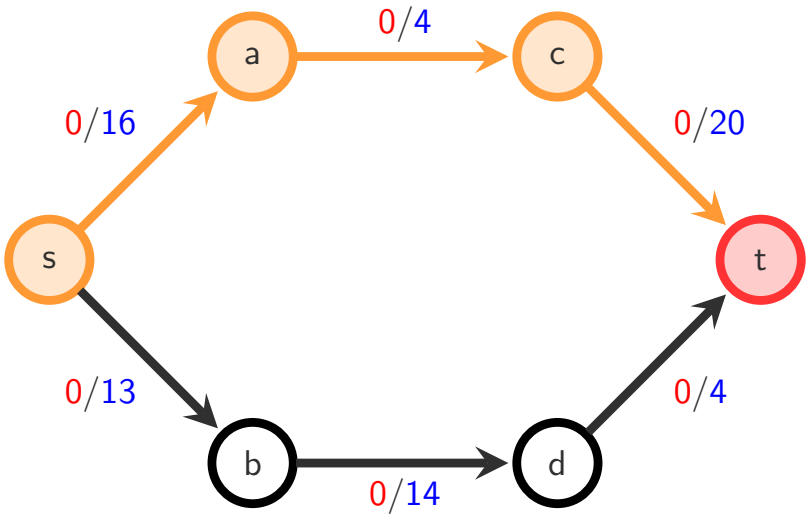
G, c and f

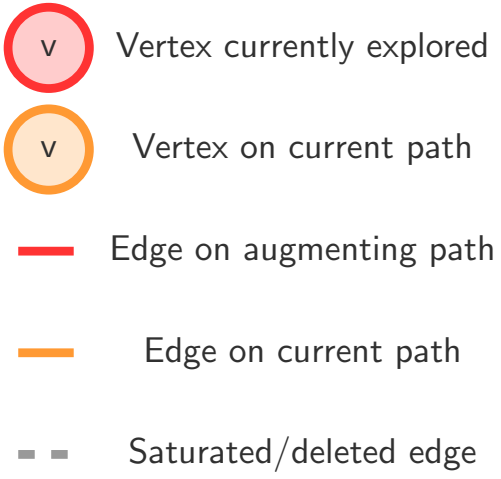


G_f and c_f



G_L, c_L and f'

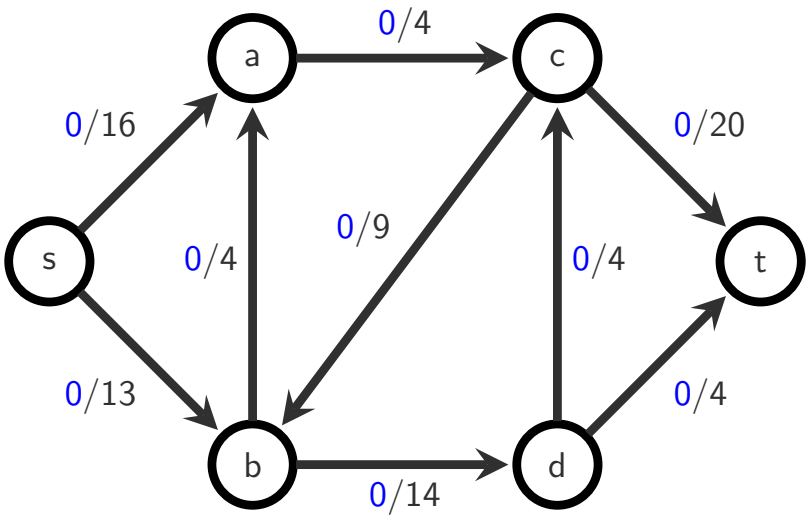




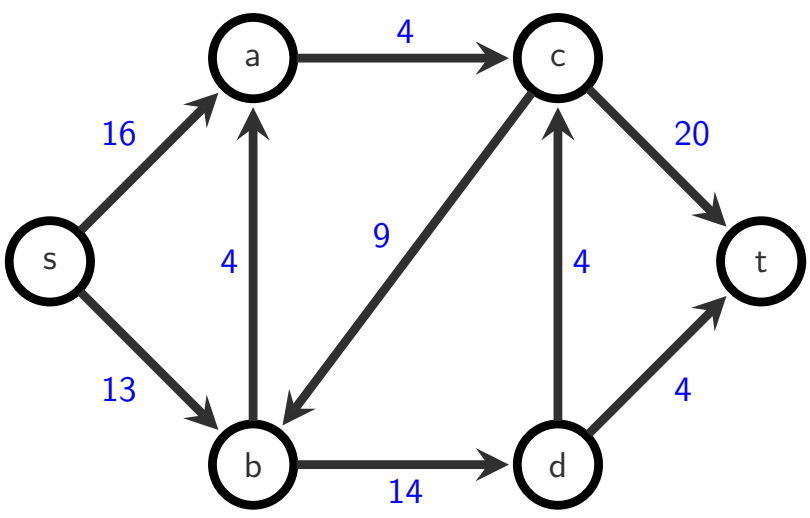
Current operation:

Find augmenting path

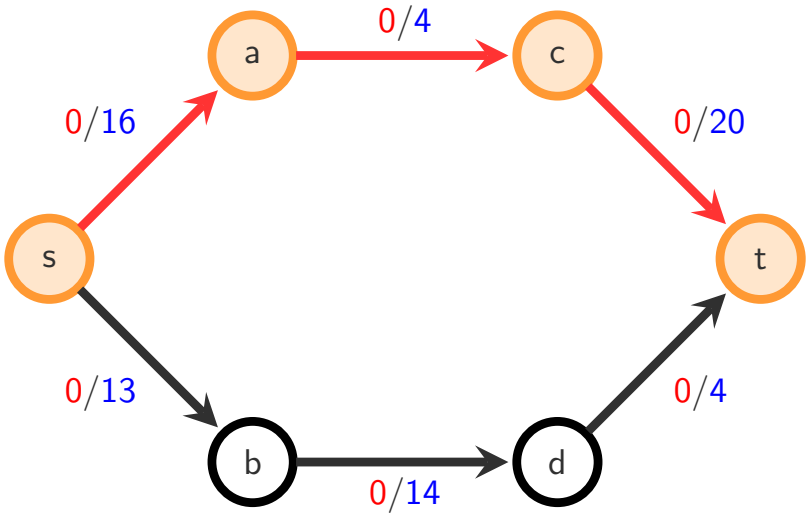
G, c and f

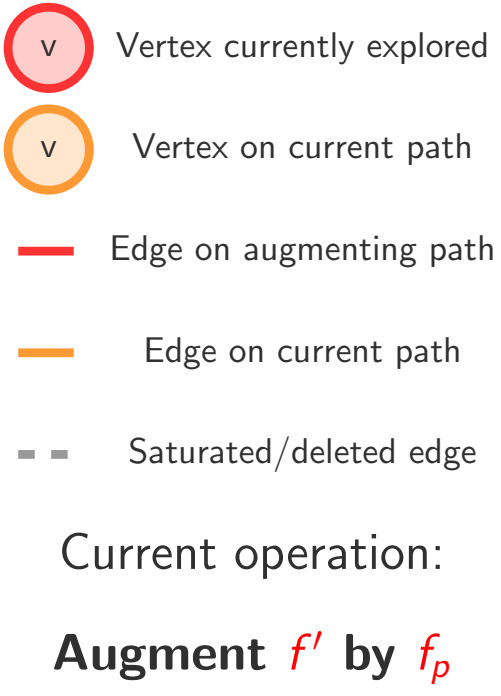


G_f and c_f

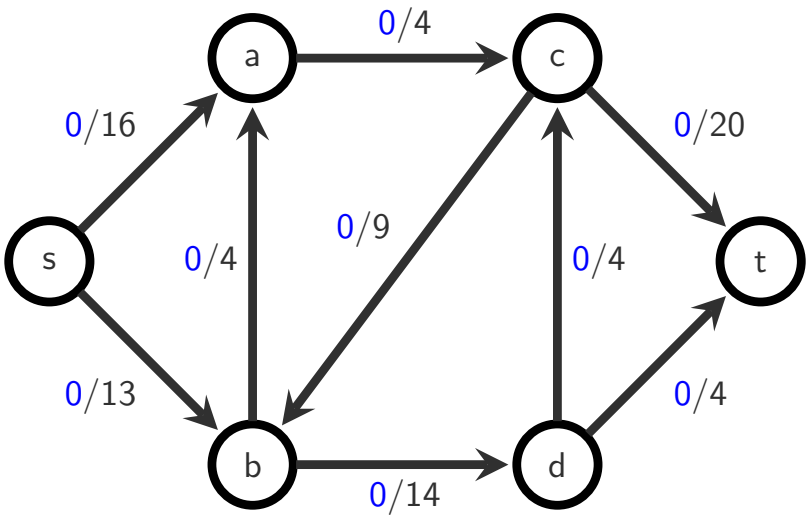


G_L, c_L and f'

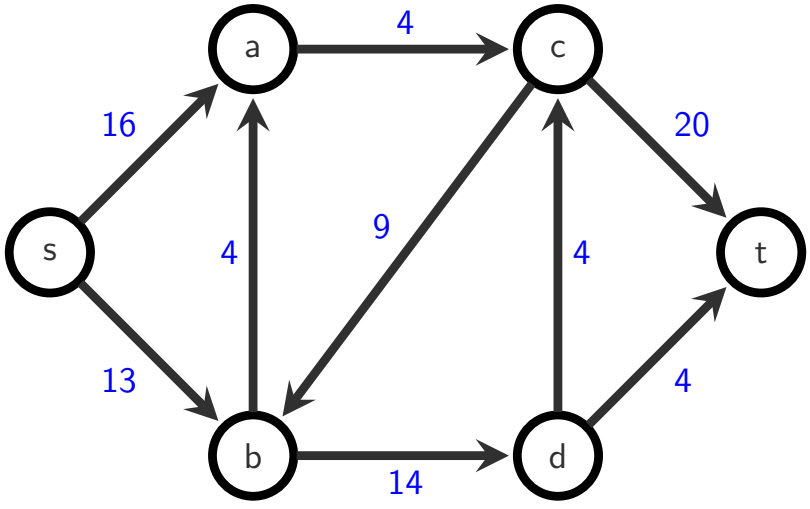




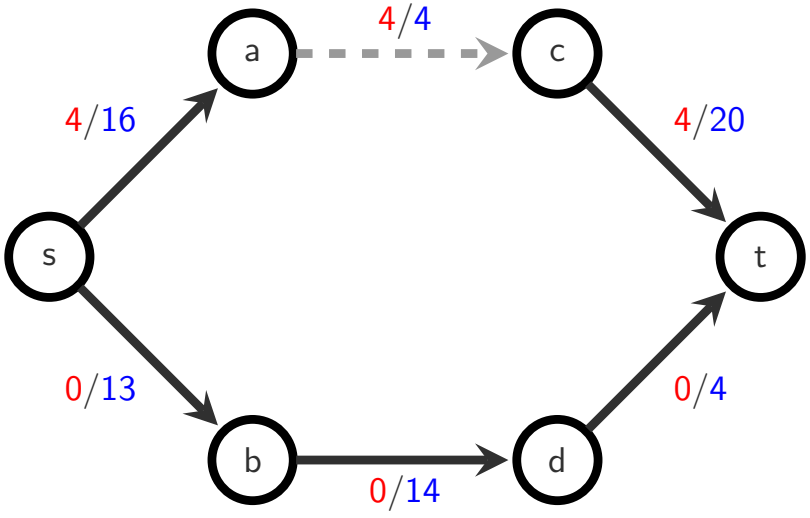
G, c and f



G_f and c_f



G_L, c_L and f'



v Vertex currently explored

v Vertex on current path

Edge on augmenting path

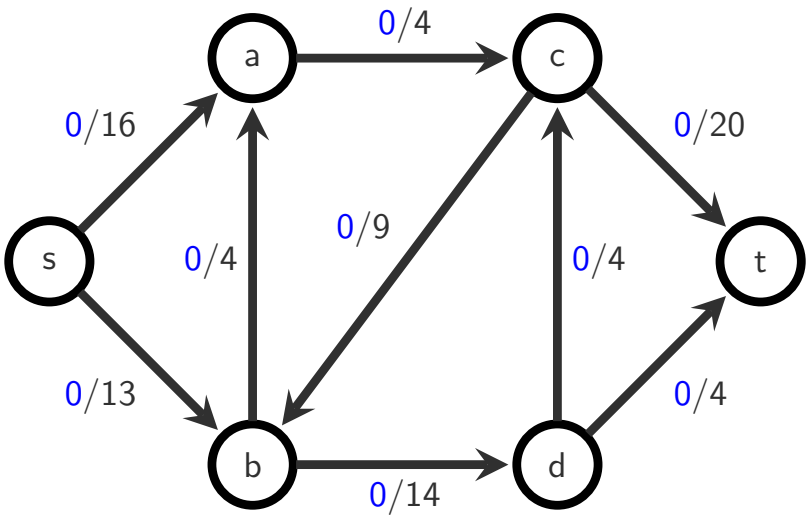
Edge on current path

Saturated/deleted edge

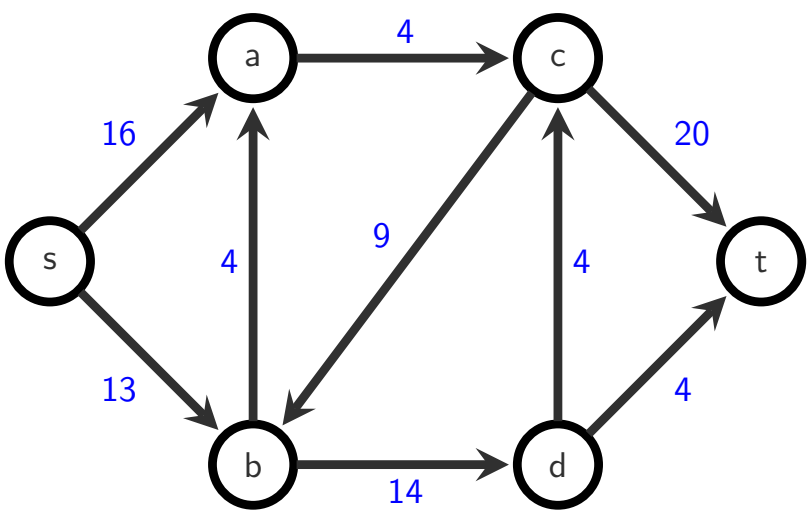
Current operation:

Augment f' by f_p

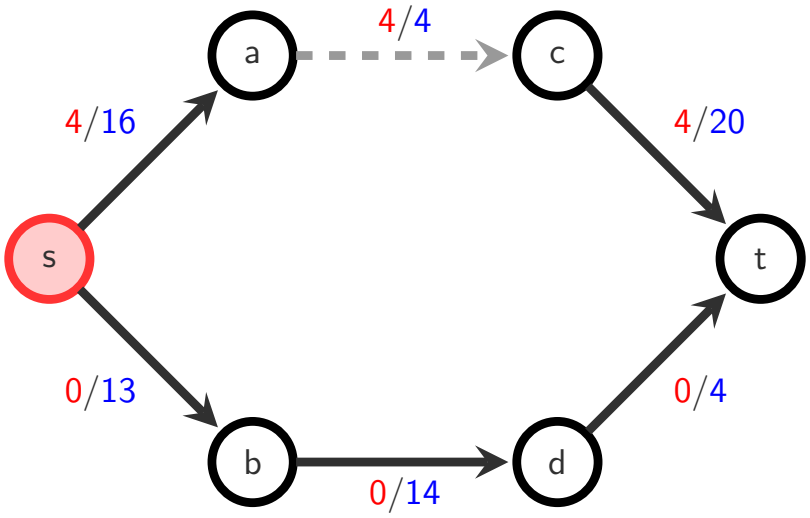
G, c and f

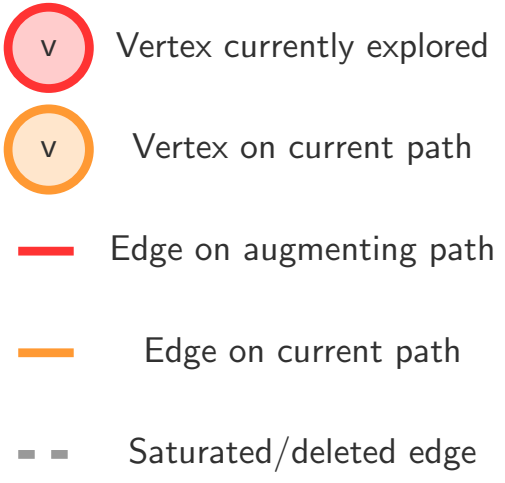


G_f and c_f



G_L, c_L and f'

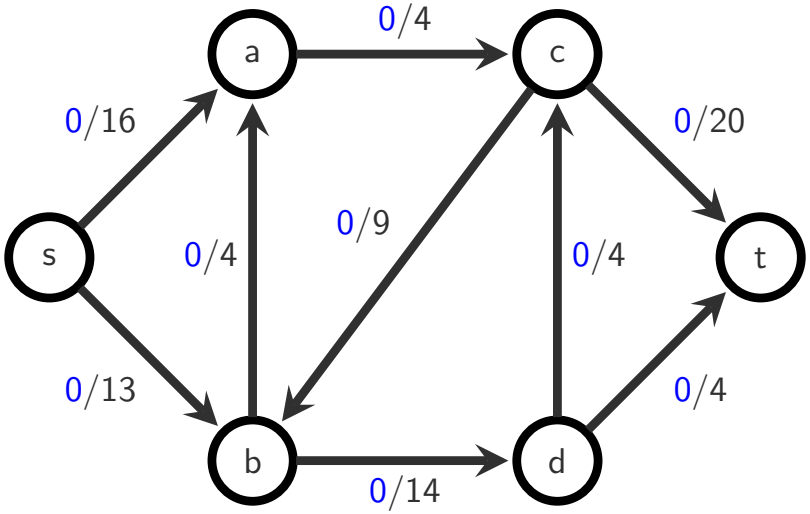




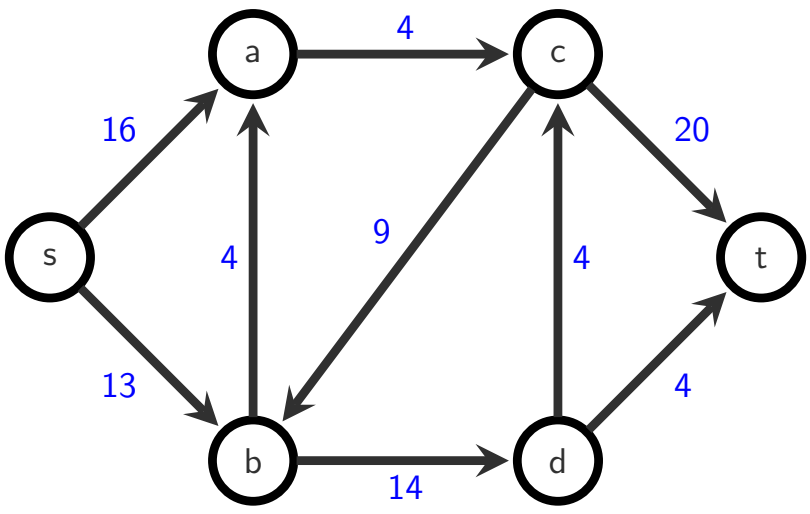
Current operation:

Find augmenting path

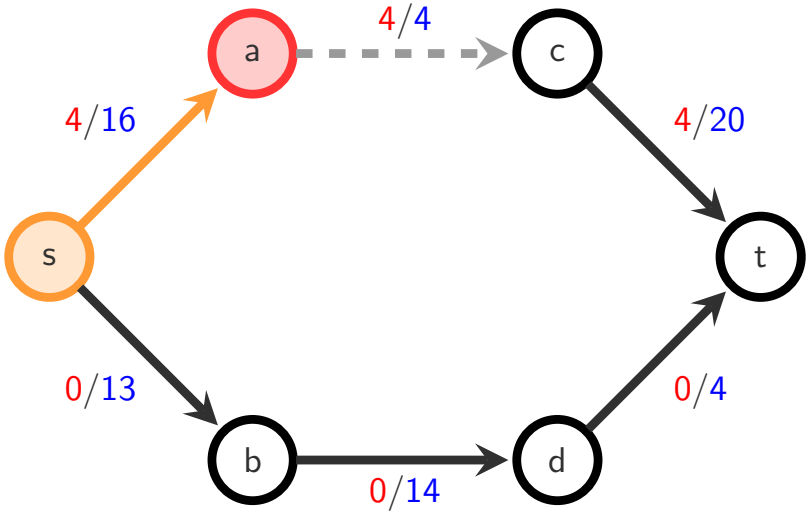
G, c and f

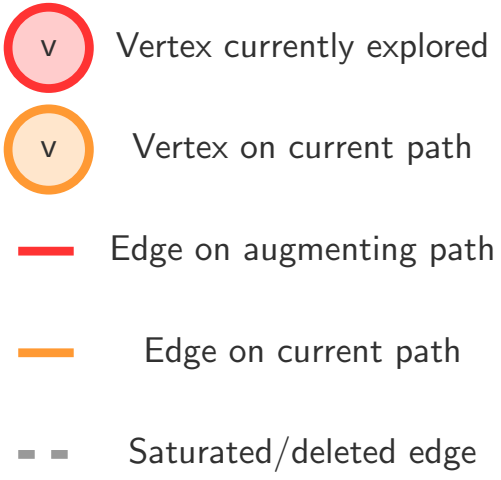


G_f and c_f



G_L, c_L and f'

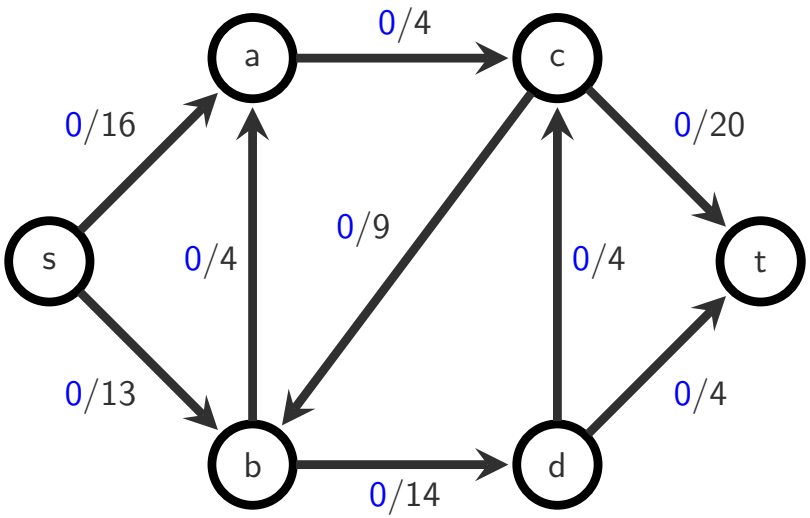




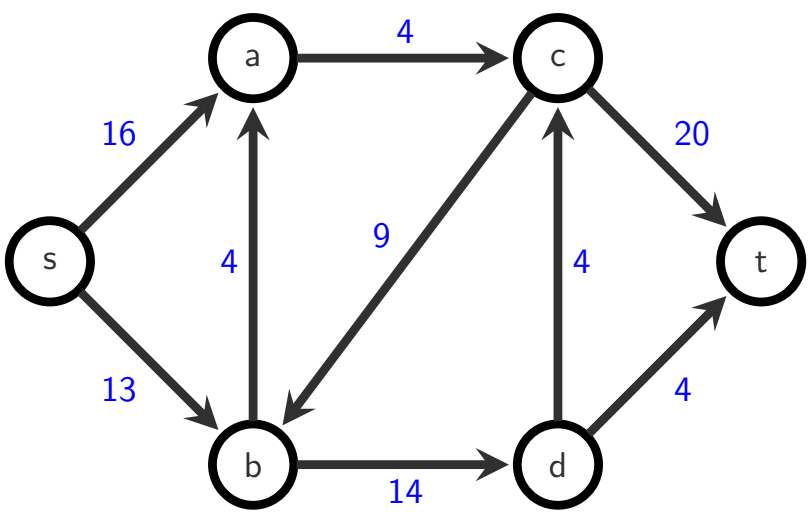
Current operation:

Find augmenting path

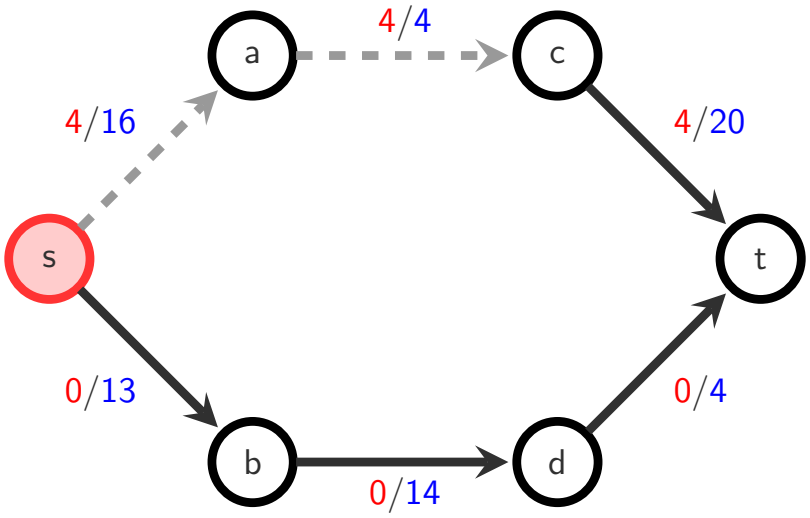
G, c and f

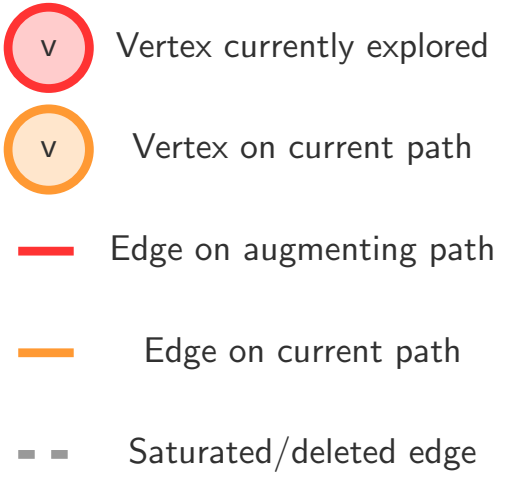


G_f and c_f



G_L, c_L and f'

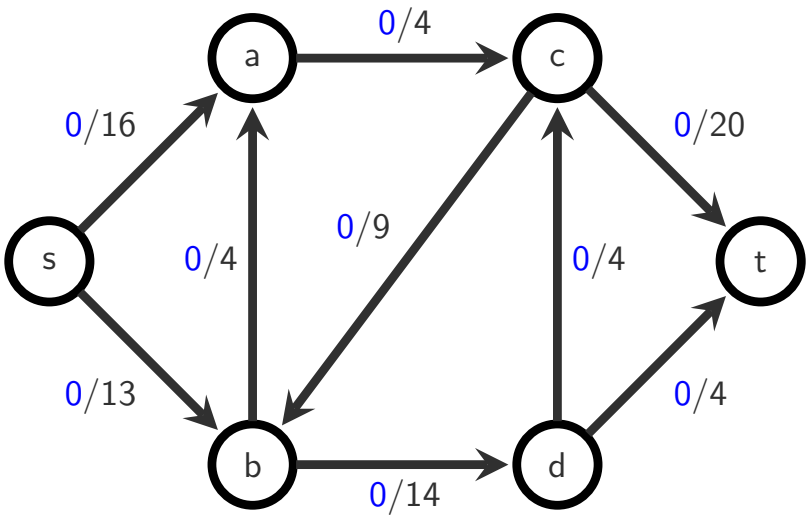




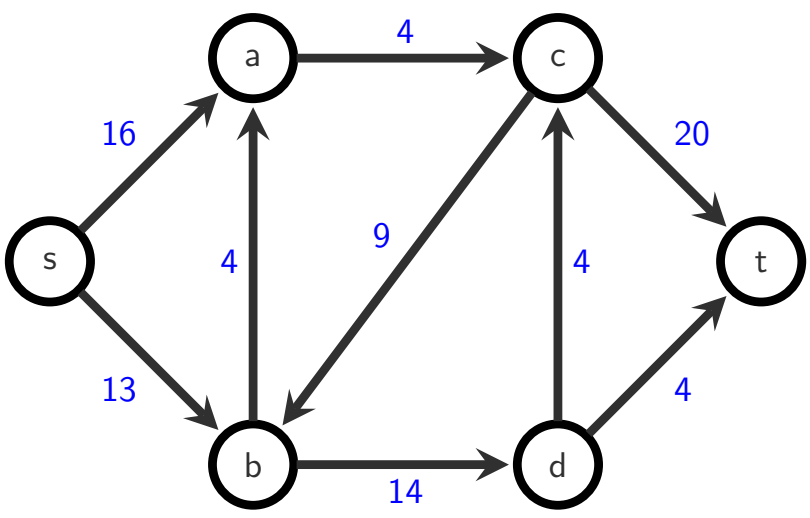
Current operation:

Find augmenting path

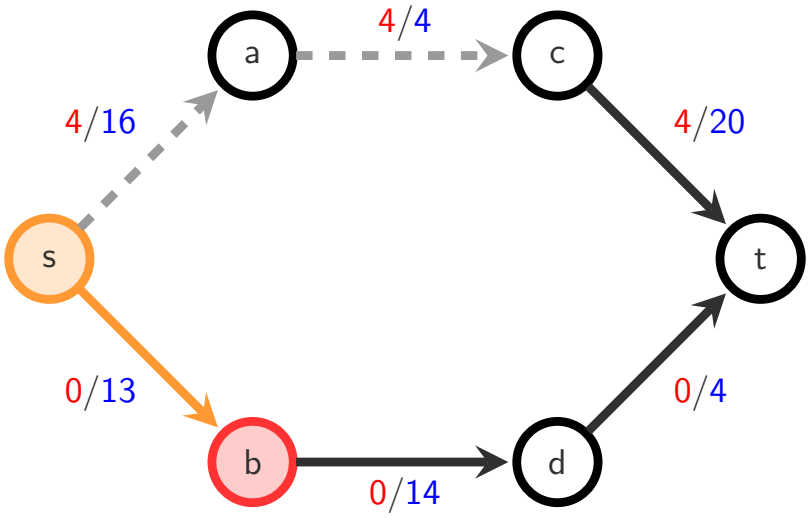
G, c and f

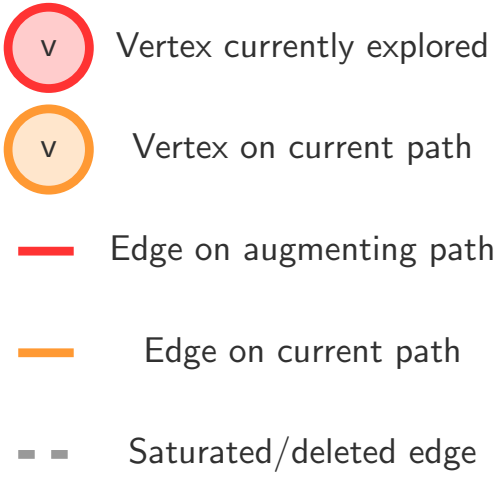


G_f and c_f



G_L, c_L and f'

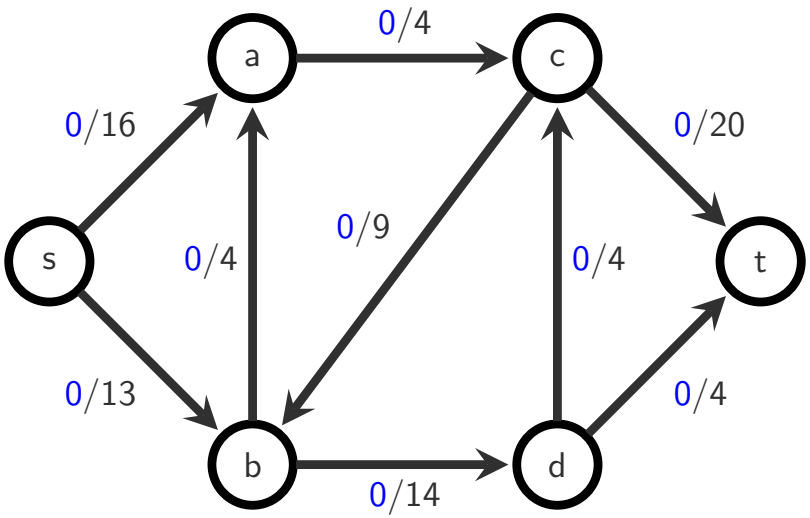




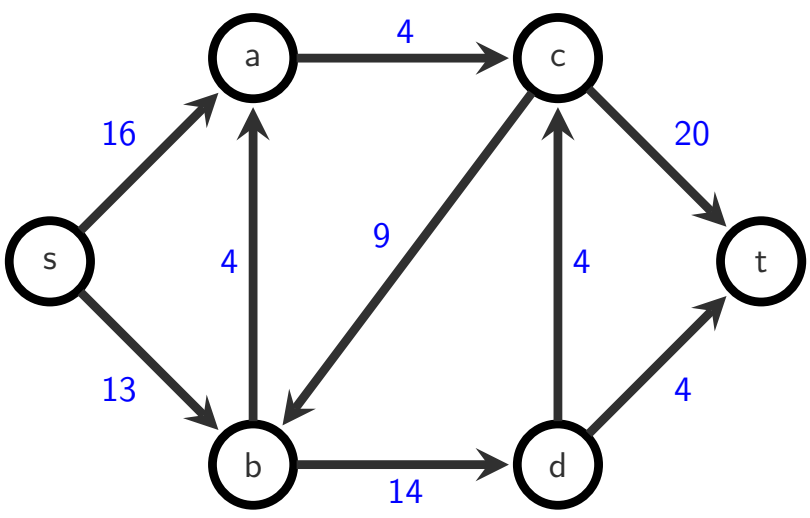
Current operation:

Find augmenting path

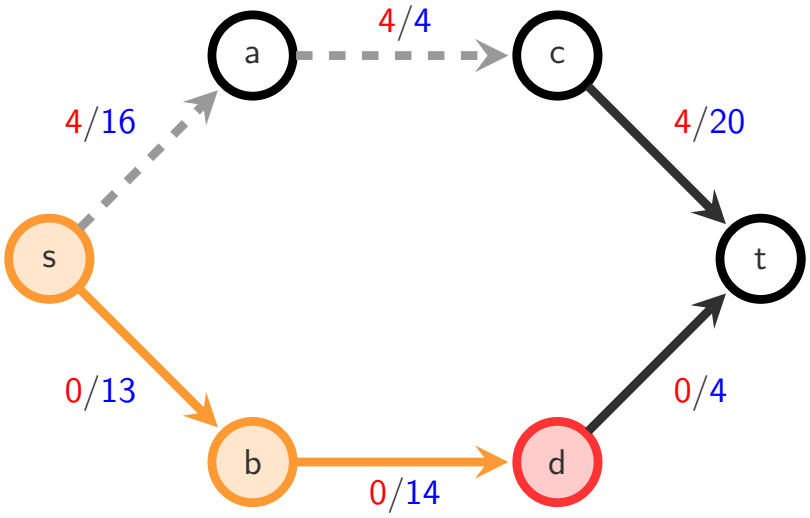
G, c and f

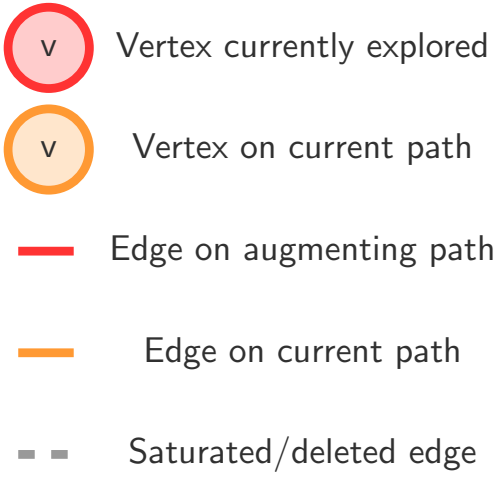


G_f and c_f



G_L, c_L and f'

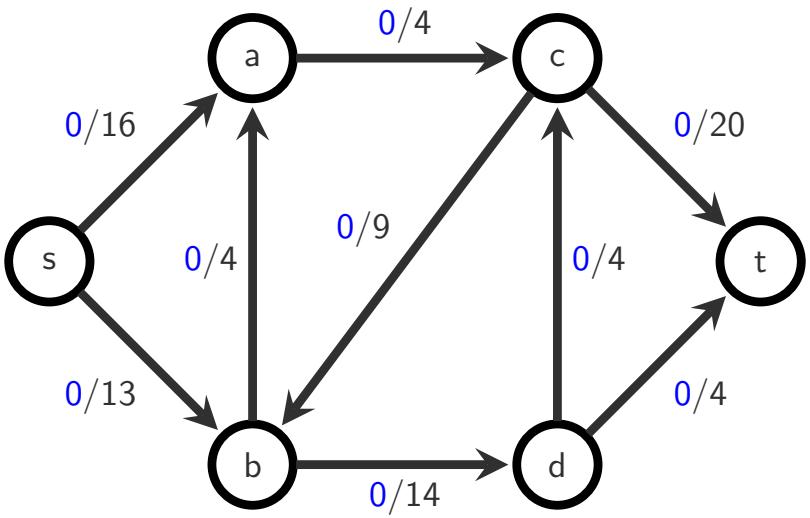




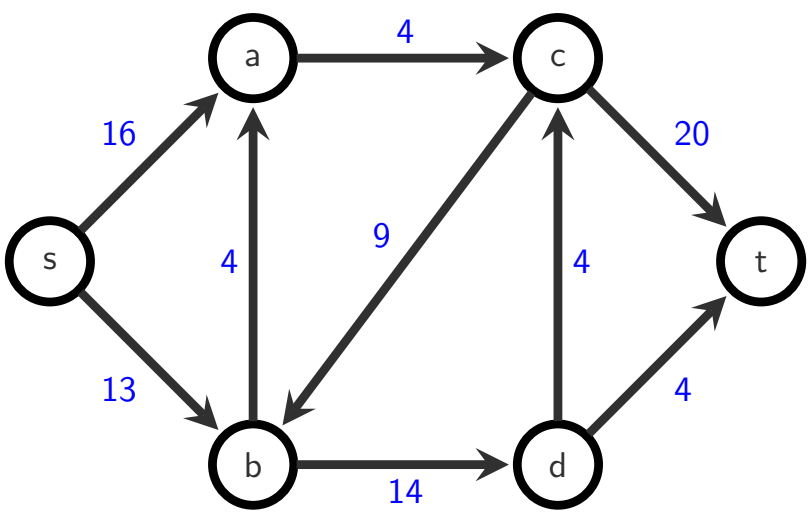
Current operation:

Find augmenting path

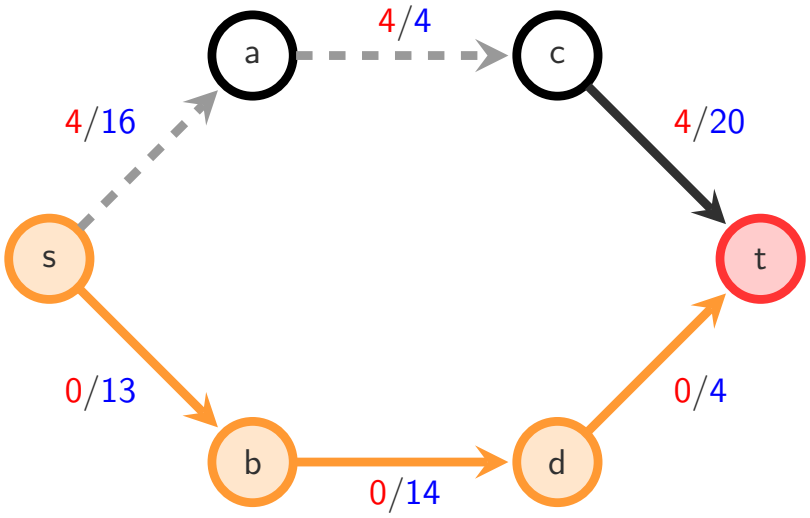
G, c and f

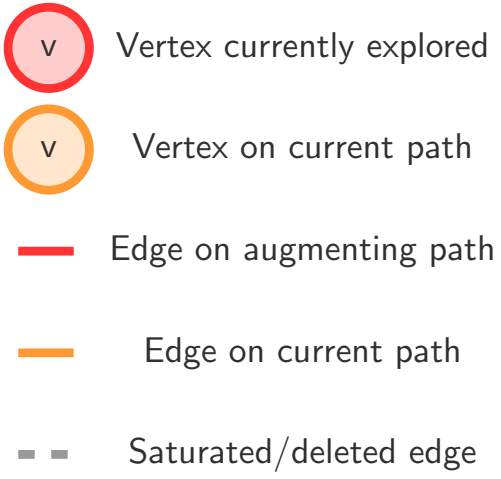


G_f and c_f



G_L, c_L and f'

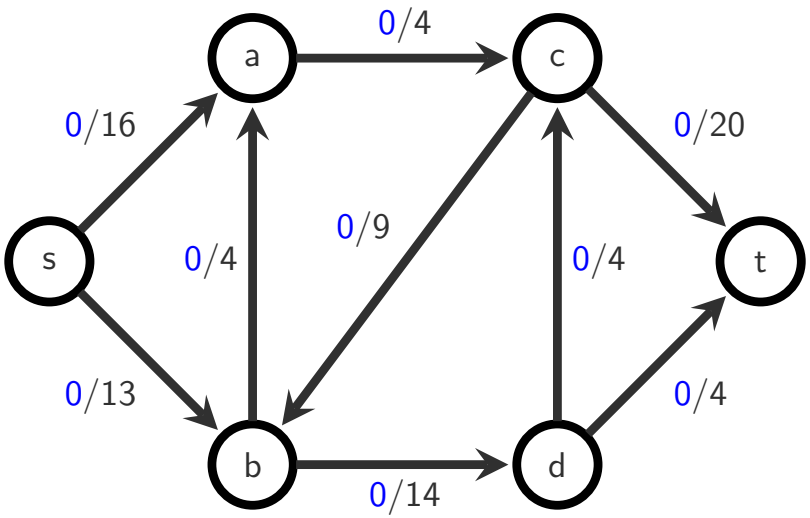




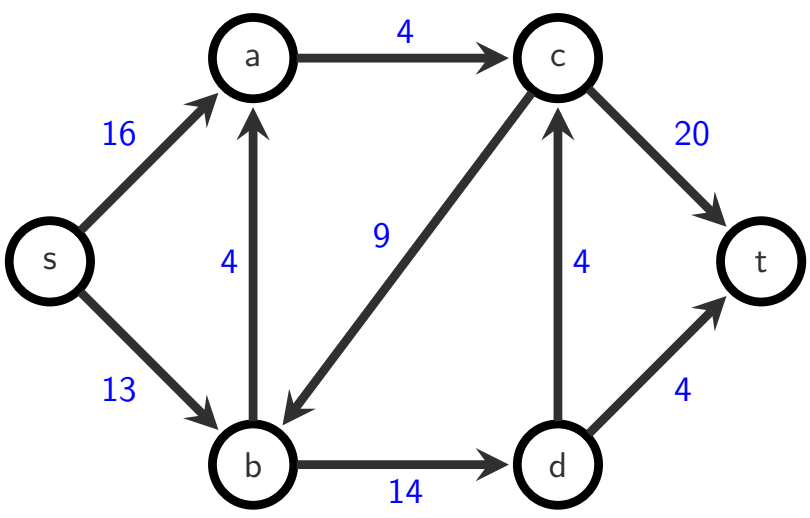
Current operation:

Find augmenting path

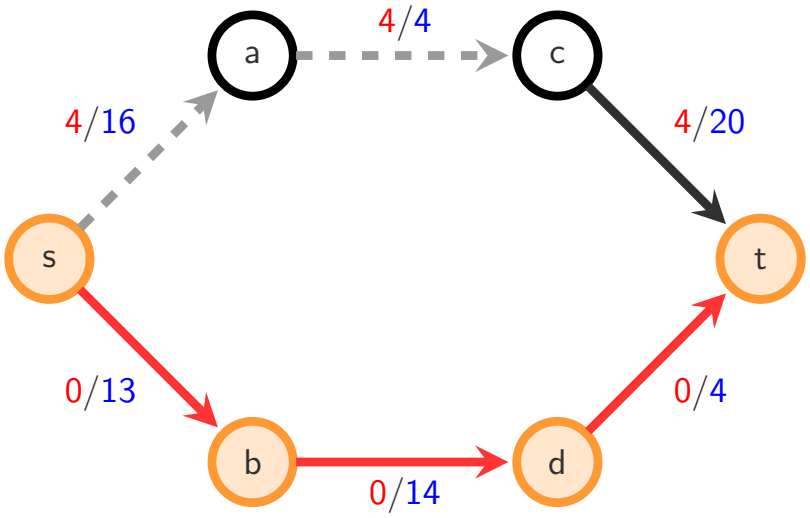
G, c and f

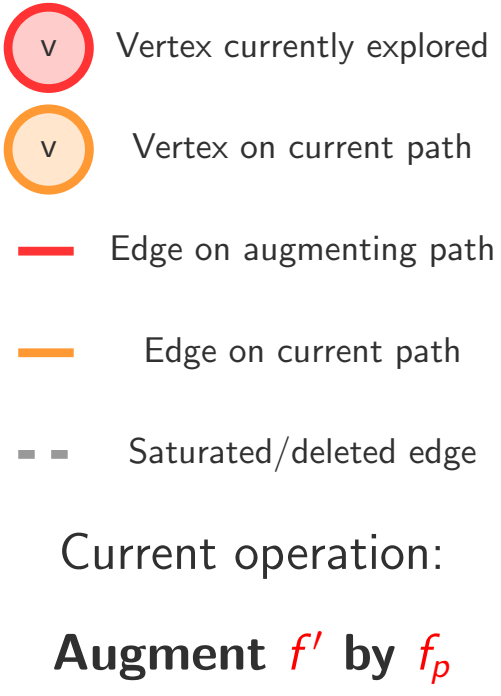


G_f and c_f

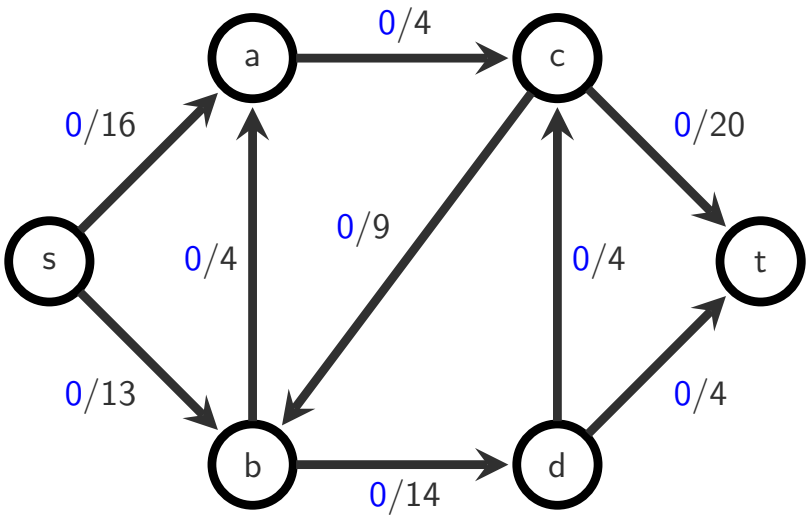


G_L, c_L and f'

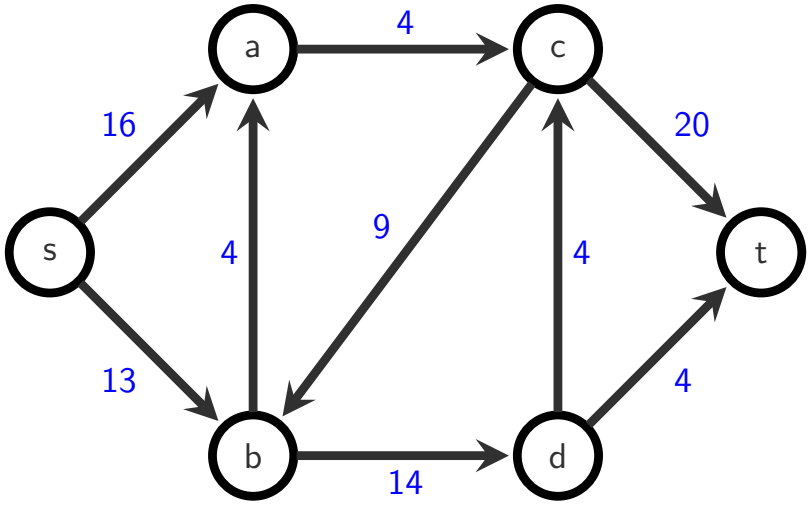




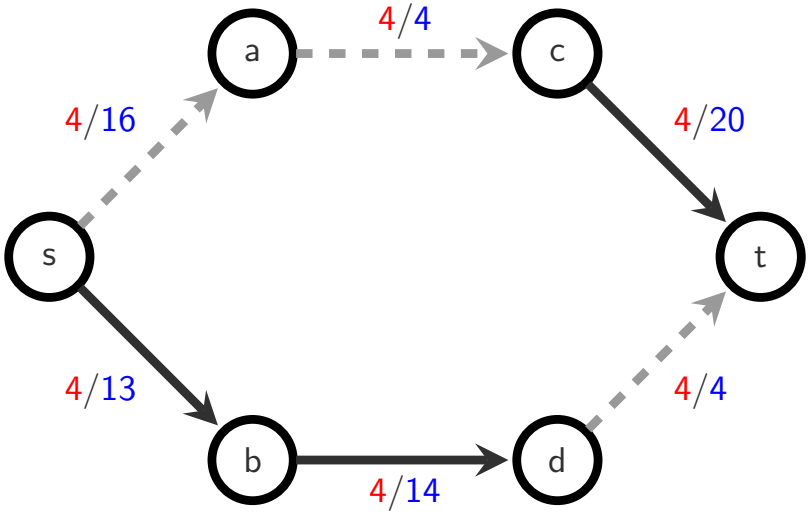
G, c and f

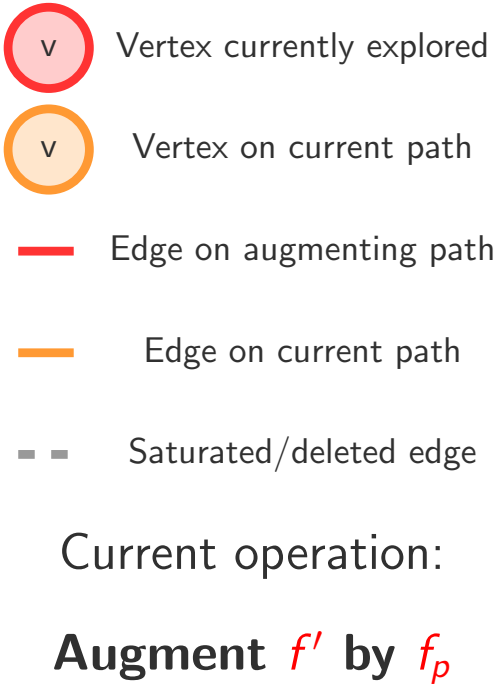


G_f and c_f

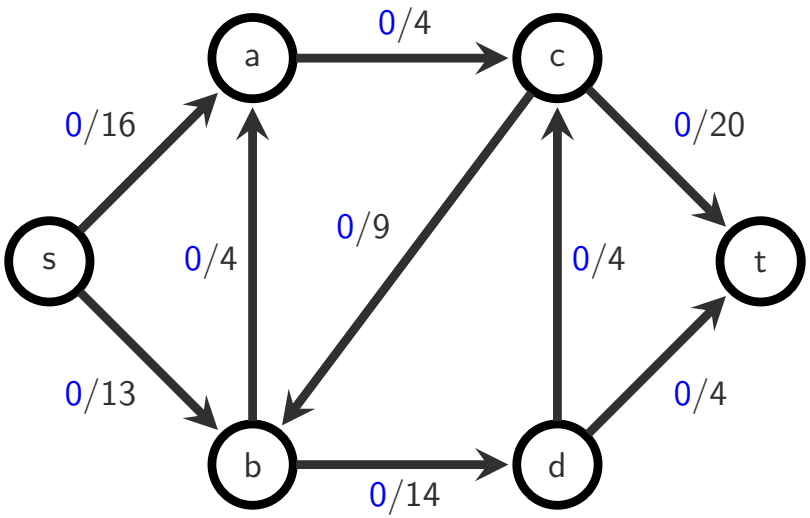


G_L, c_L and f'

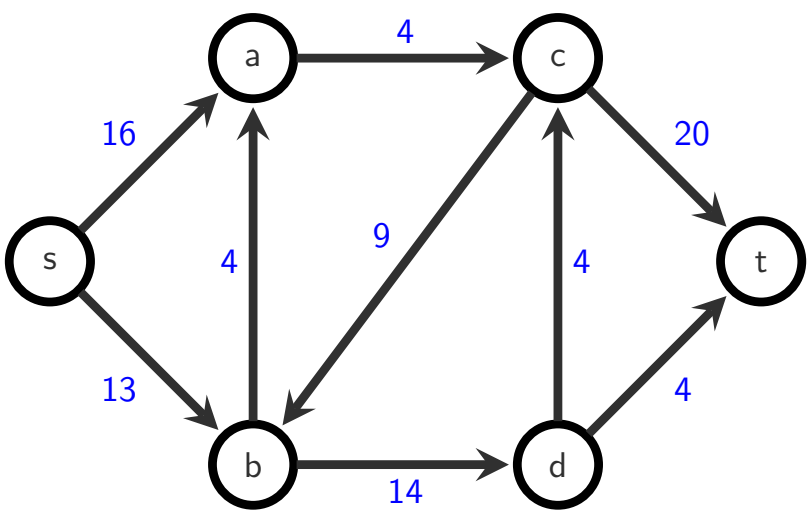




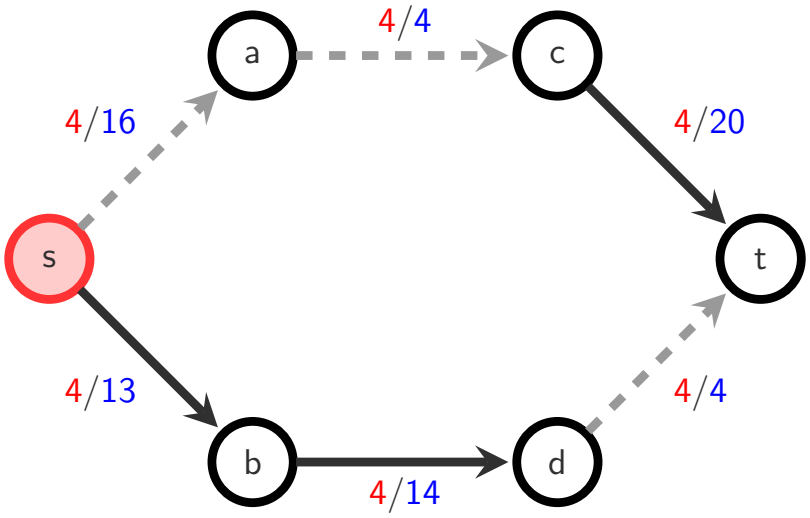
G, c and f

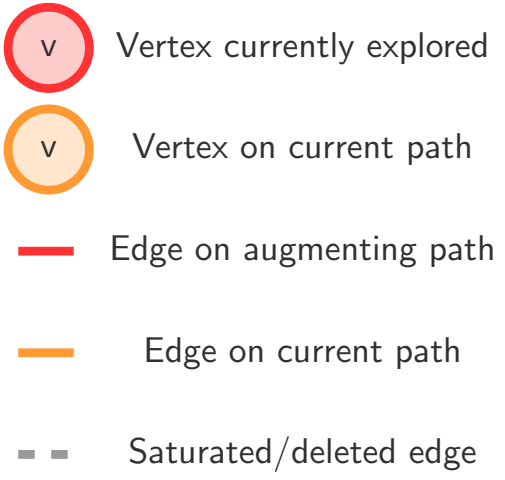


G_f and c_f



G_L, c_L and f'

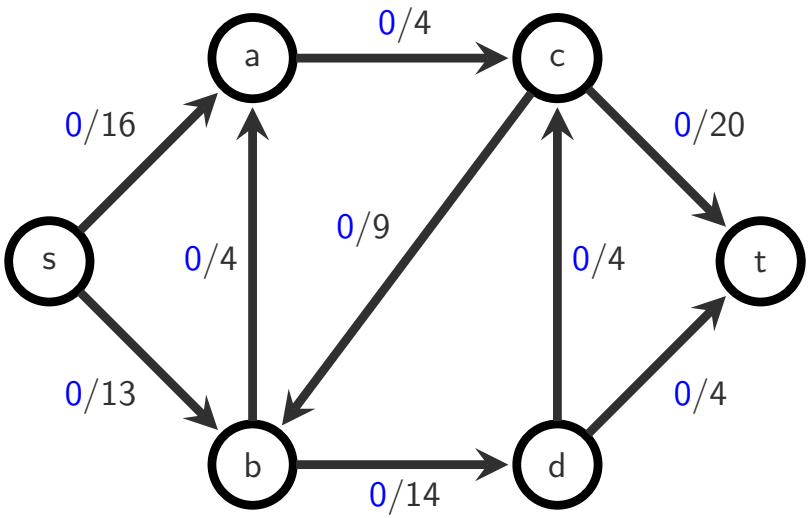




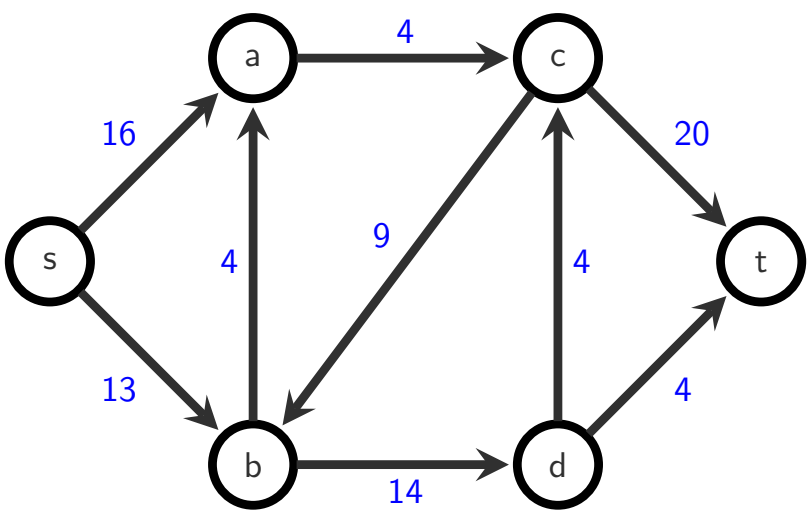
Current operation:

Find augmenting path

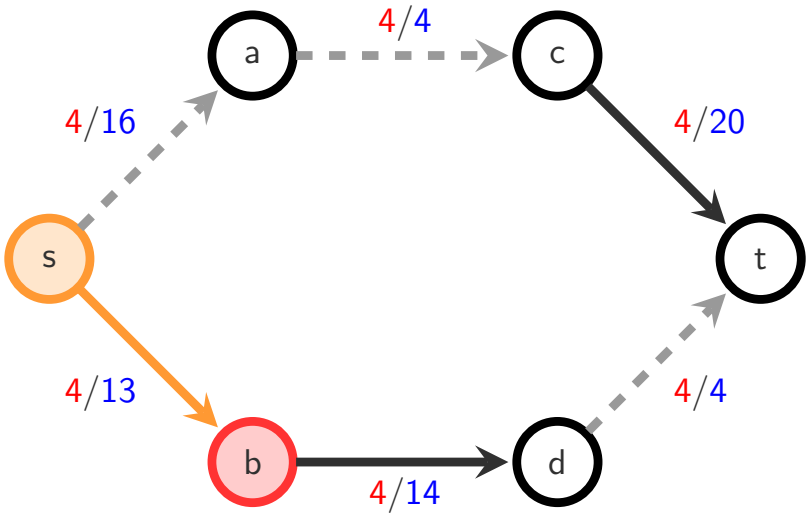
G, c and f

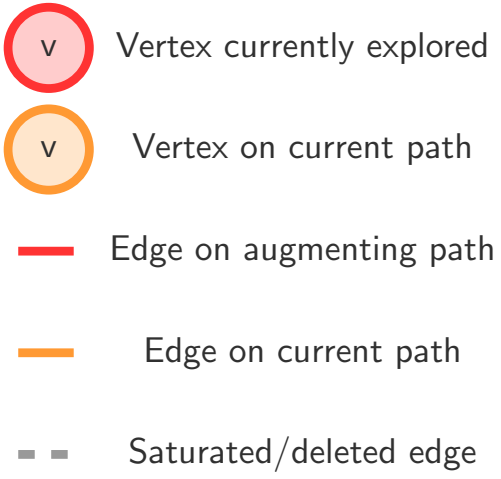


G_f and c_f



G_L, c_L and f'

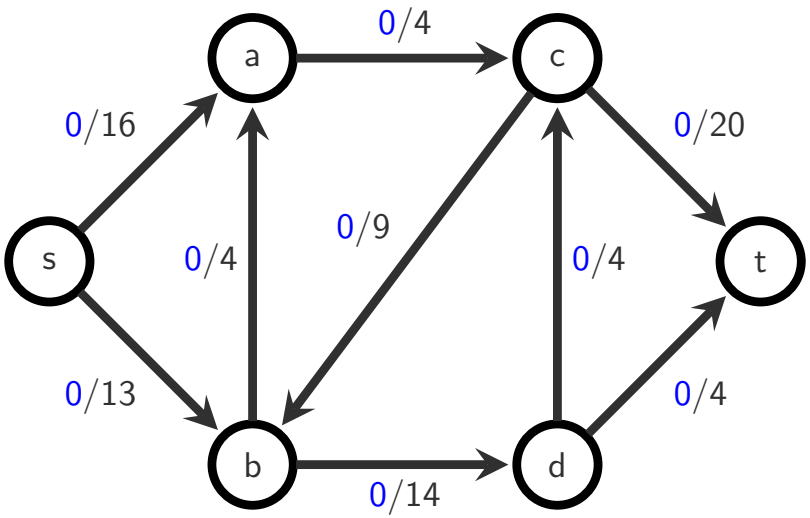




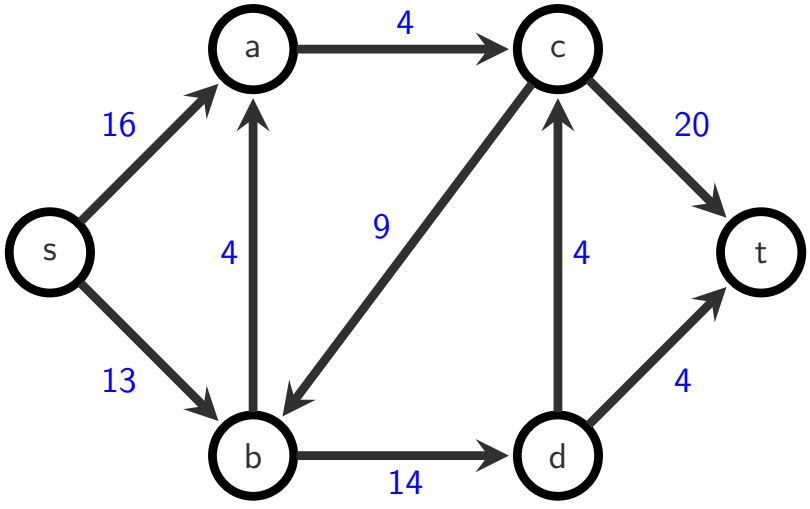
Current operation:

Find augmenting path

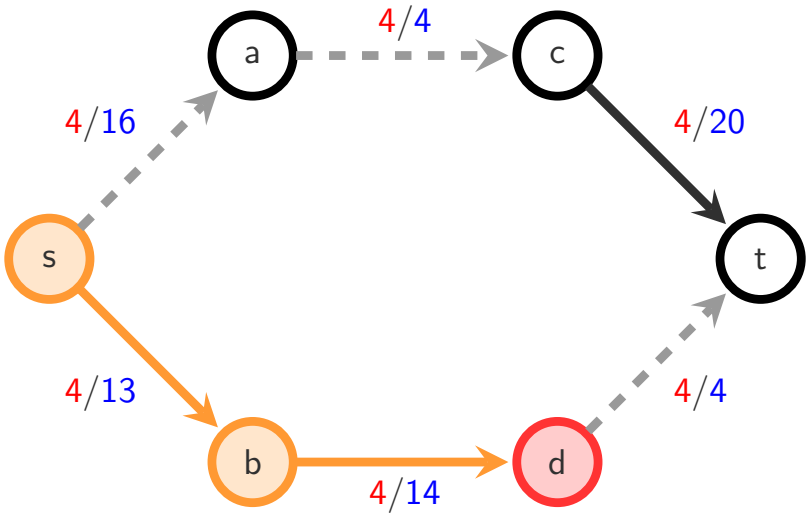
G, c and f

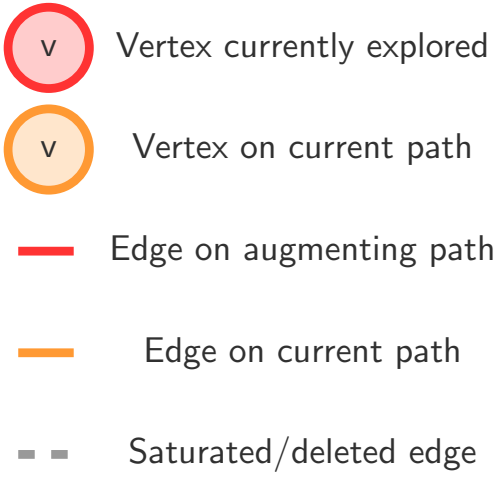


G_f and c_f



G_L, c_L and f'

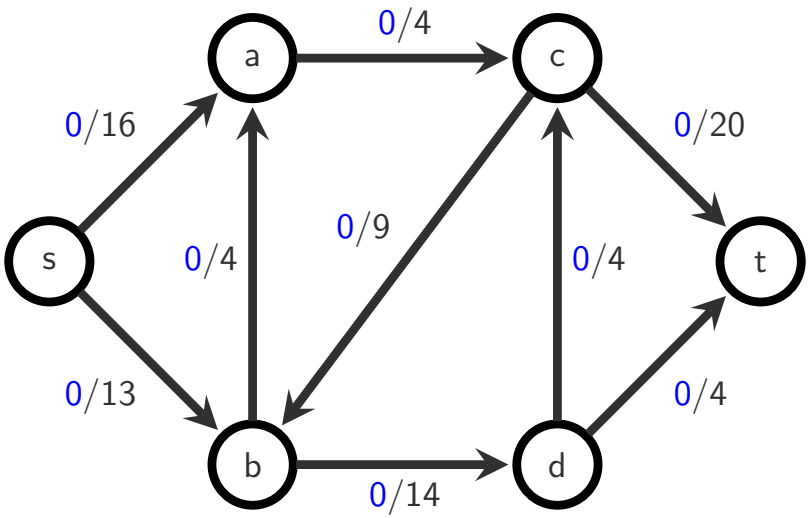




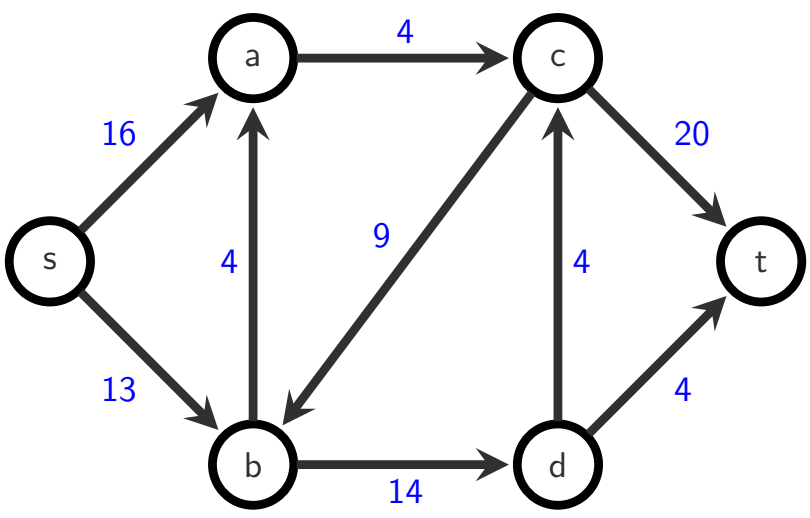
Current operation:

Find augmenting path

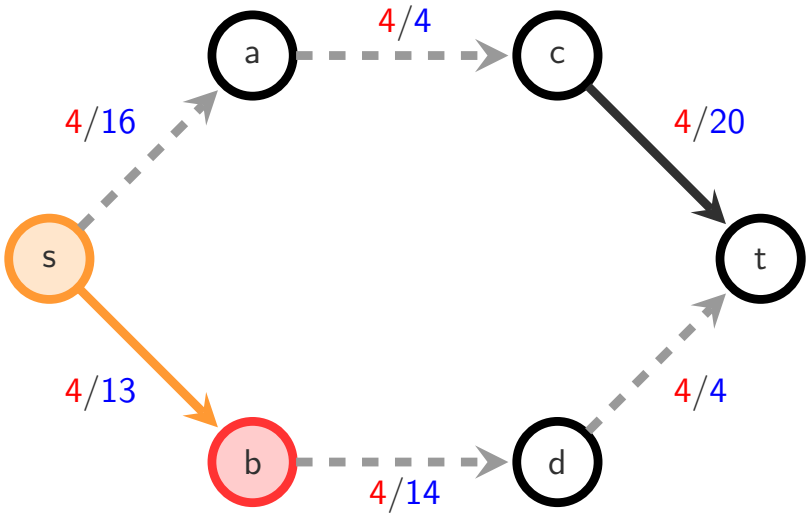
G, c and f

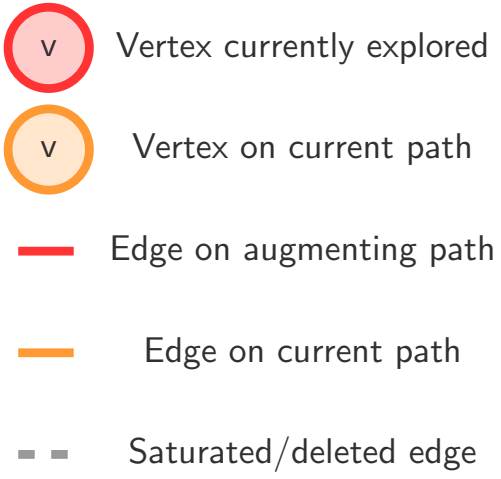


G_f and c_f



G_L, c_L and f'

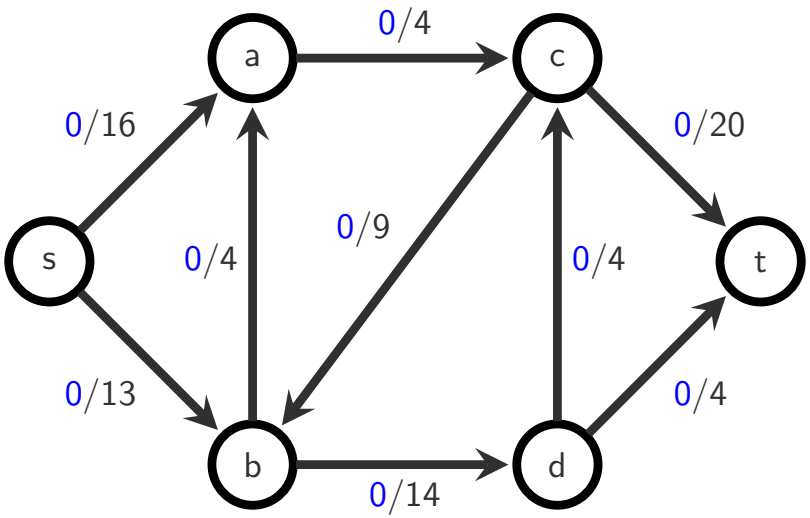




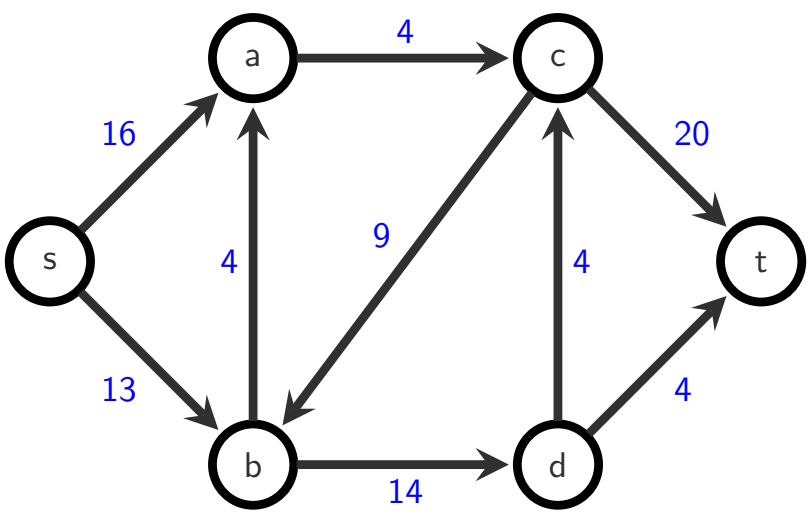
Current operation:

Find augmenting path

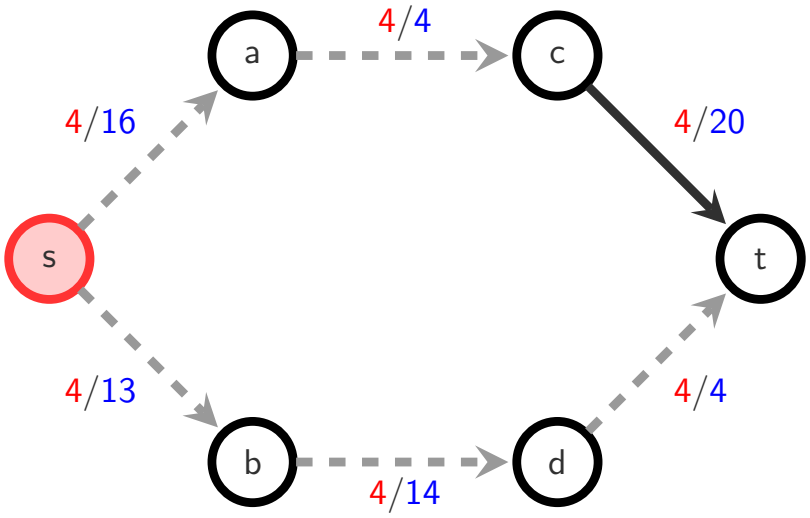
G, c and f

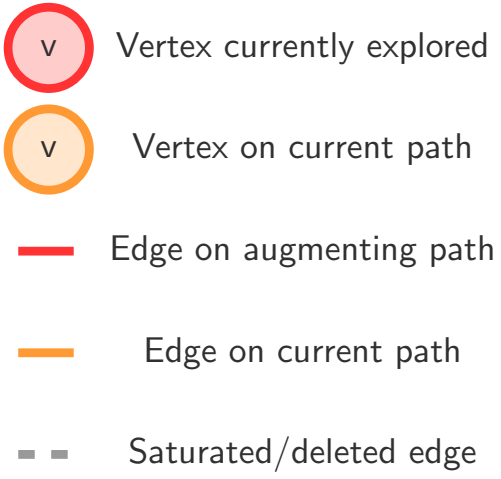


G_f and c_f



G_L, c_L and f'

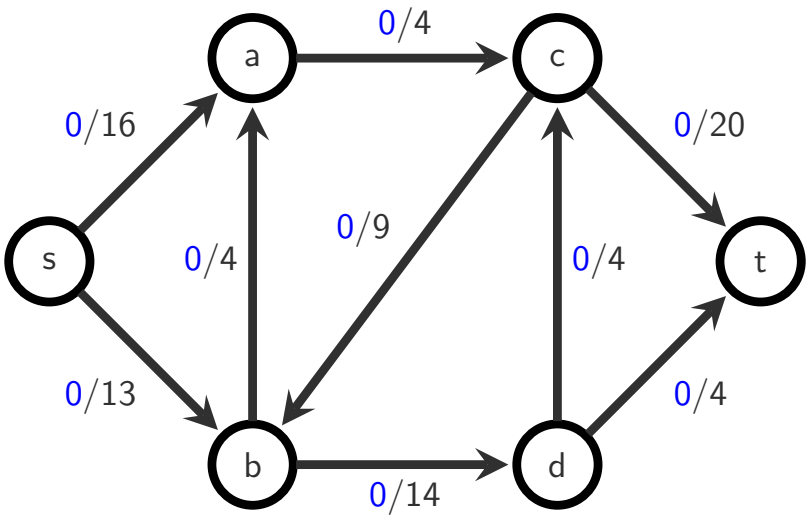




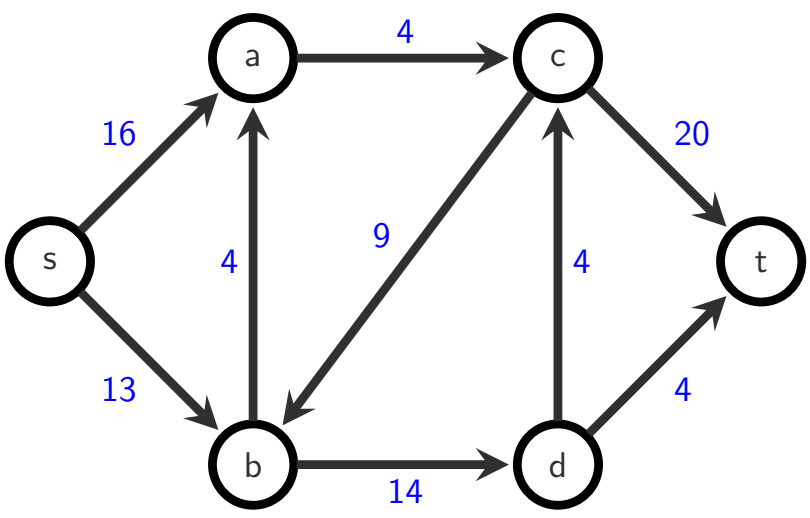
Current operation:

Find augmenting path

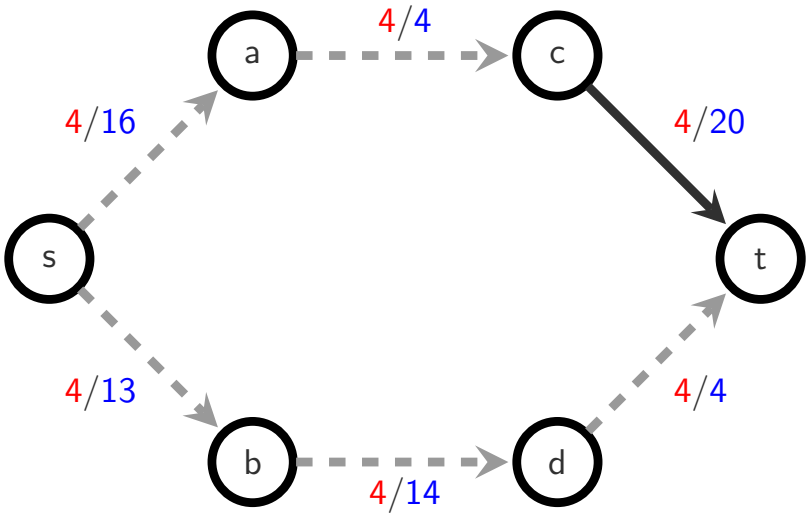
G, c and f

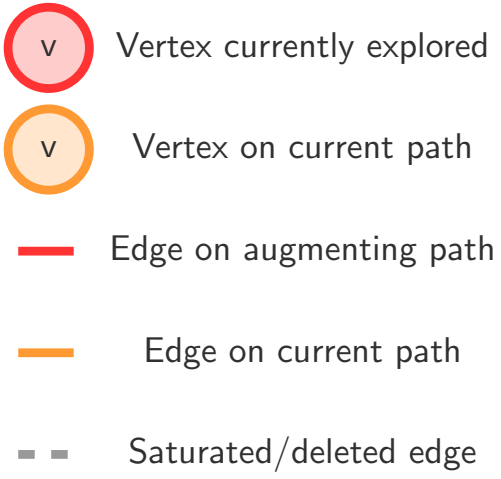


G_f and c_f



G_L, c_L and f'

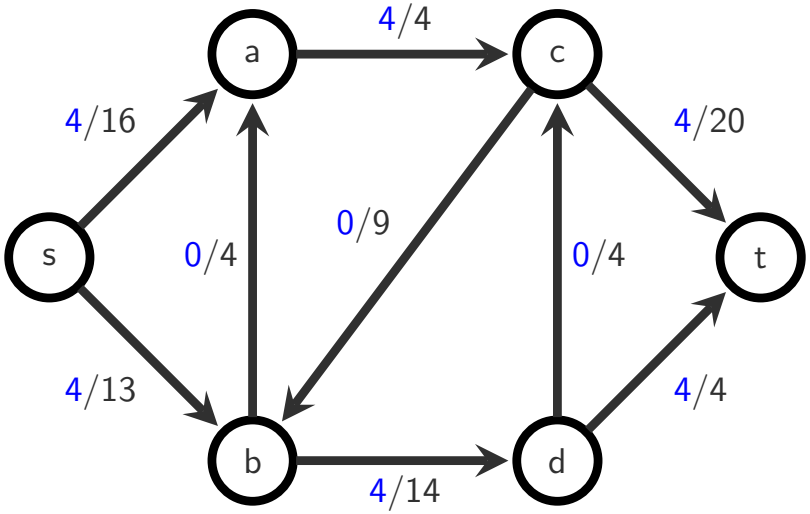




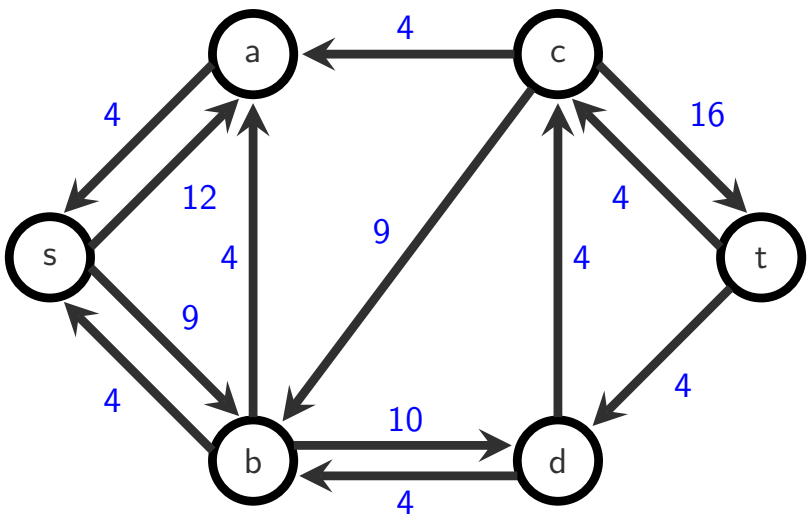
Current operation:

Augment f by f'

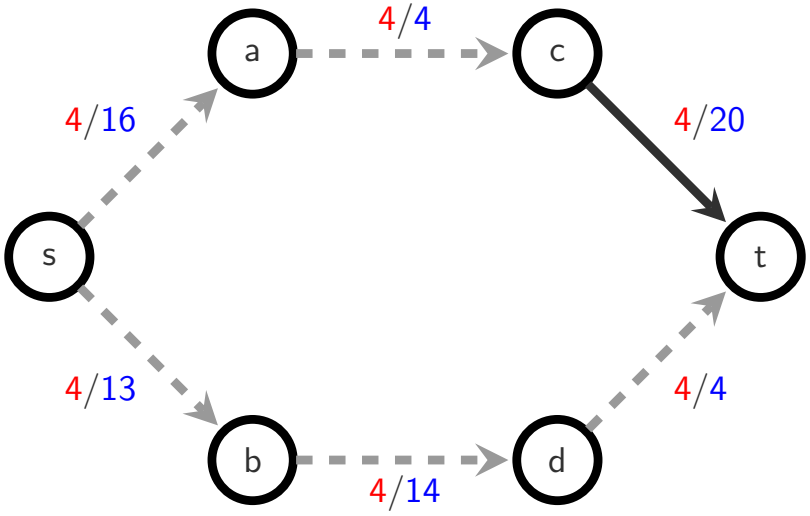
G, c and f

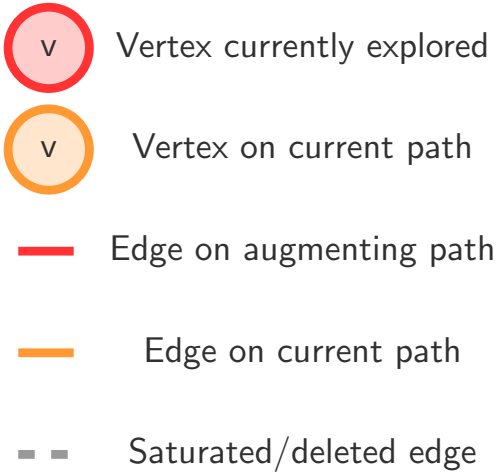


G_f and c_f



G_L, c_L and f'

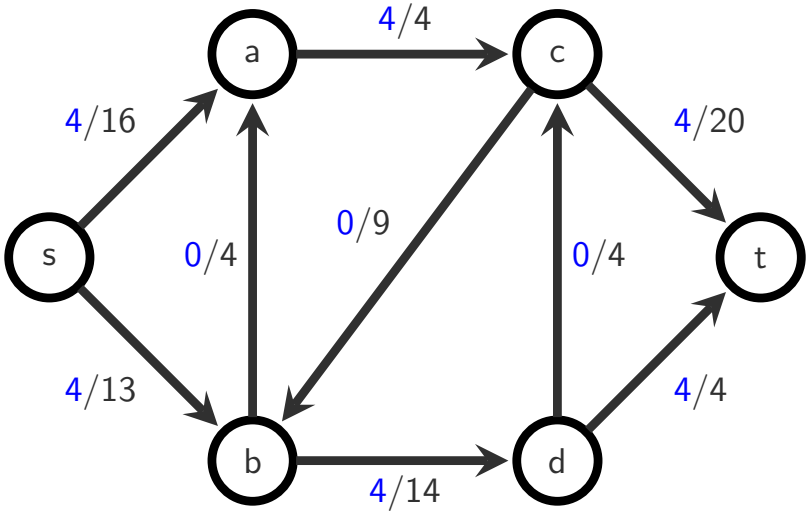




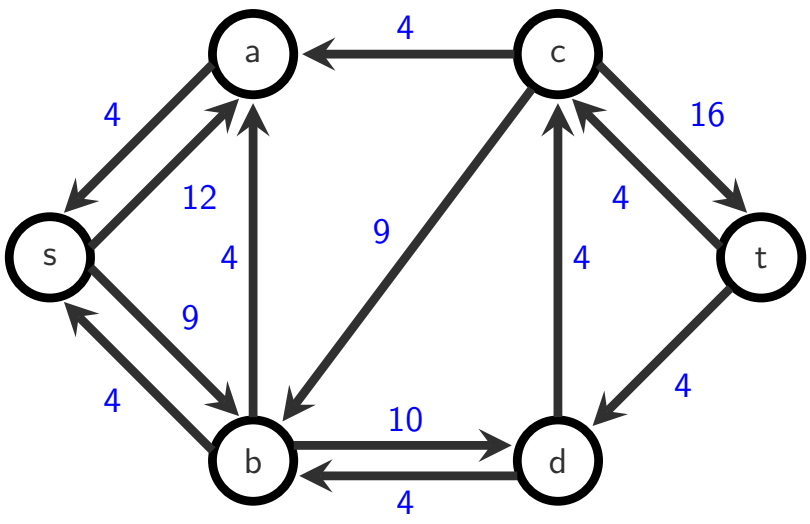
Current operation:

Augment f by f'

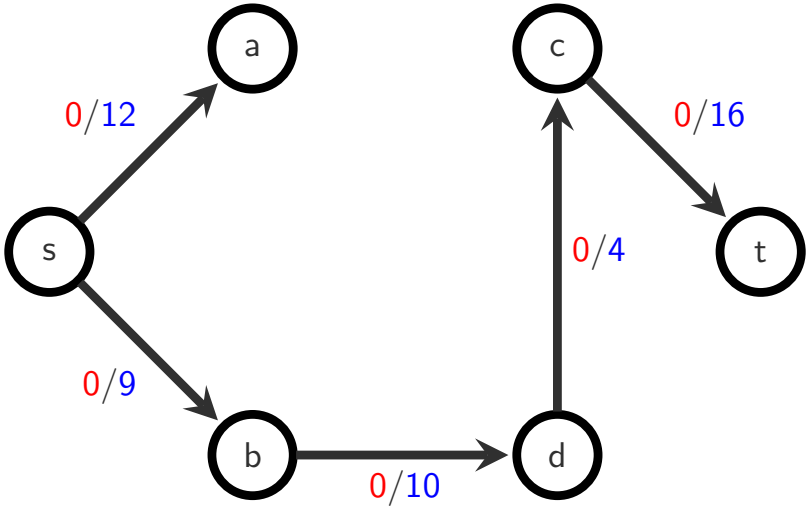
G, c and f

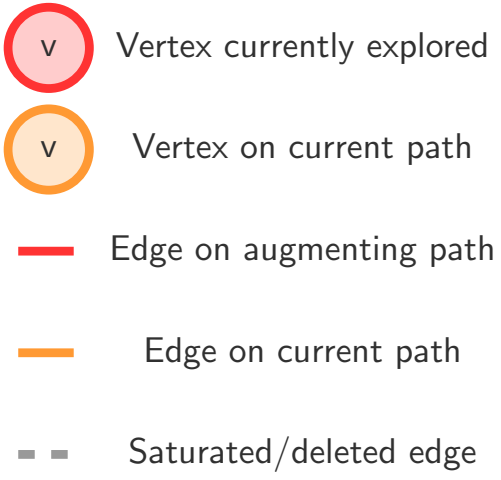


G_f and c_f



G_L, c_L and f'

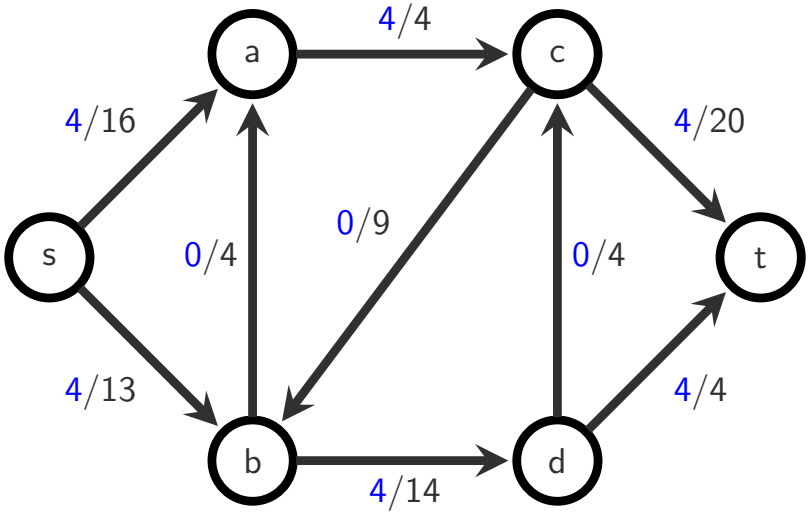




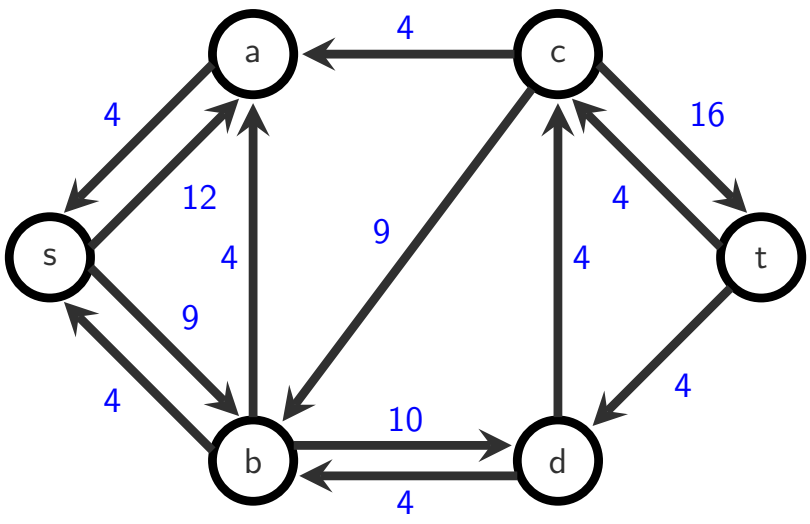
Current operation:

Find blocking flow

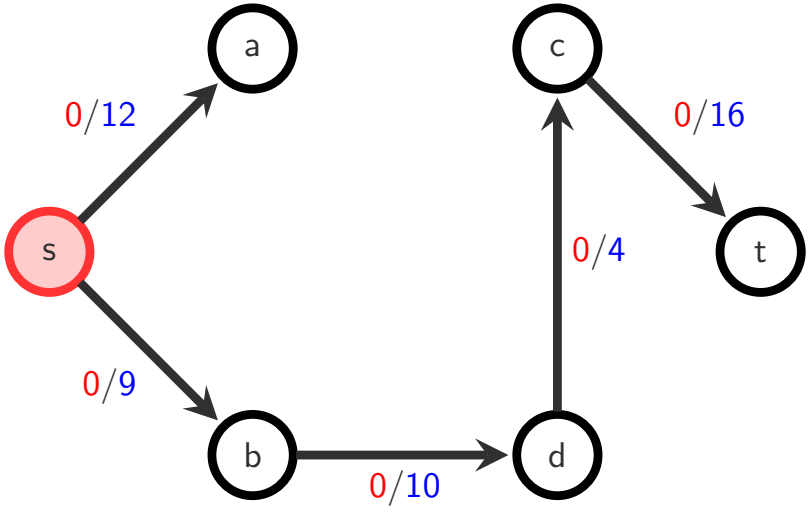
G, c and f

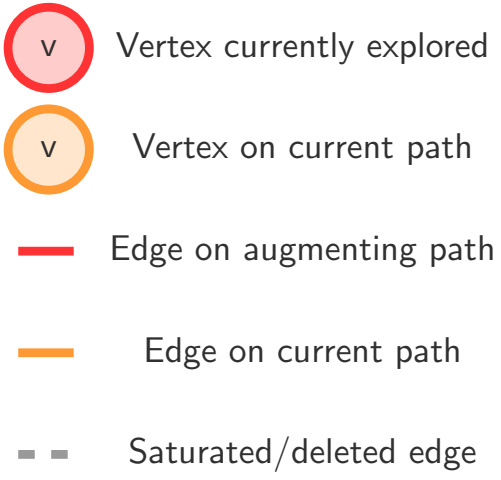


G_f and c_f



G_L, c_L and f'

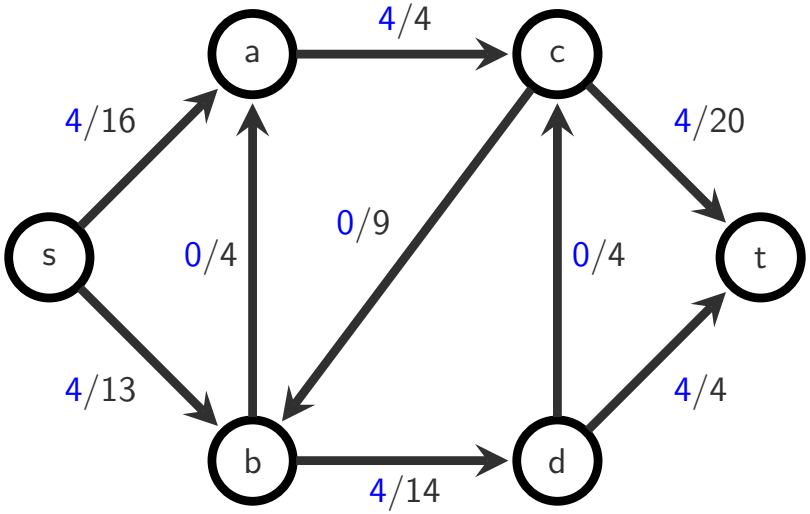




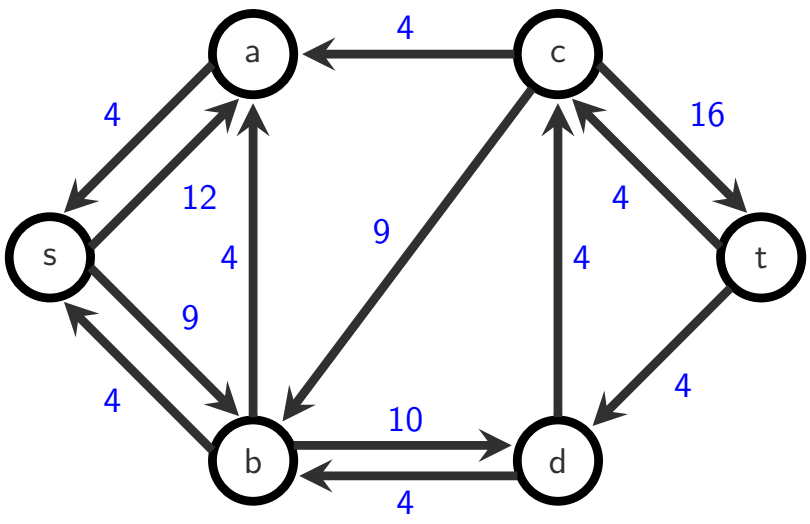
Current operation:

Find augmenting path

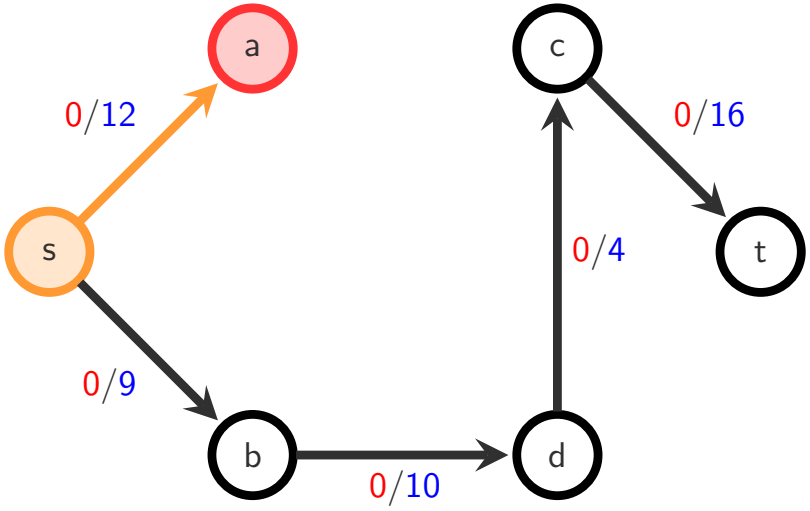
G, c and f

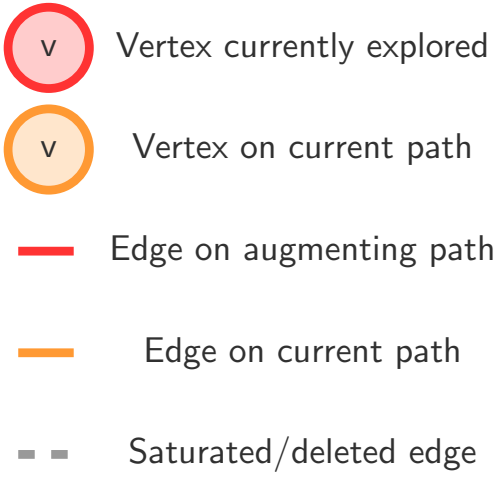


G_f and c_f



G_L, c_L and f'

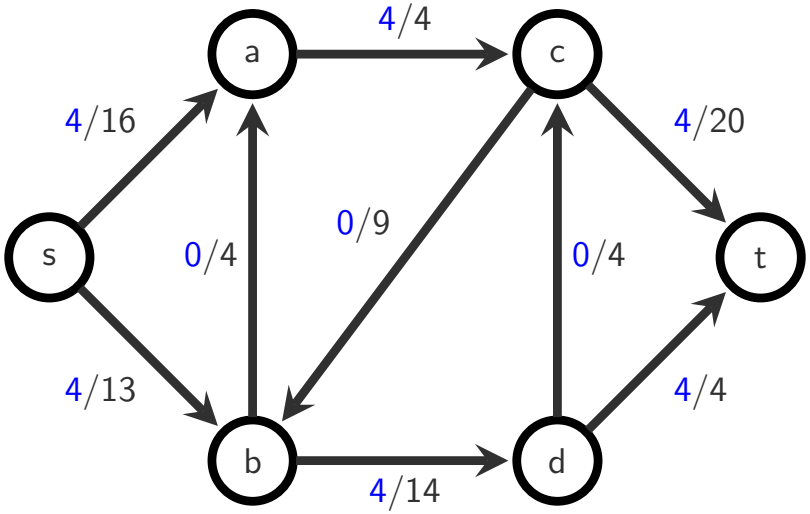




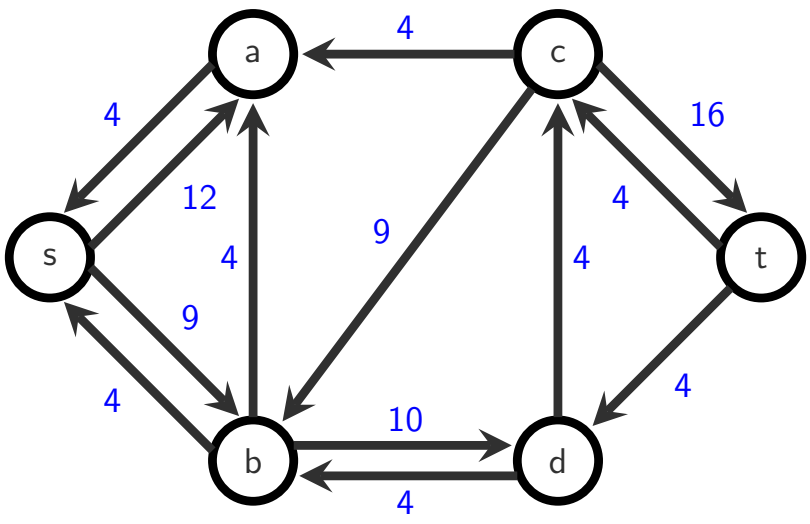
Current operation:

Find augmenting path

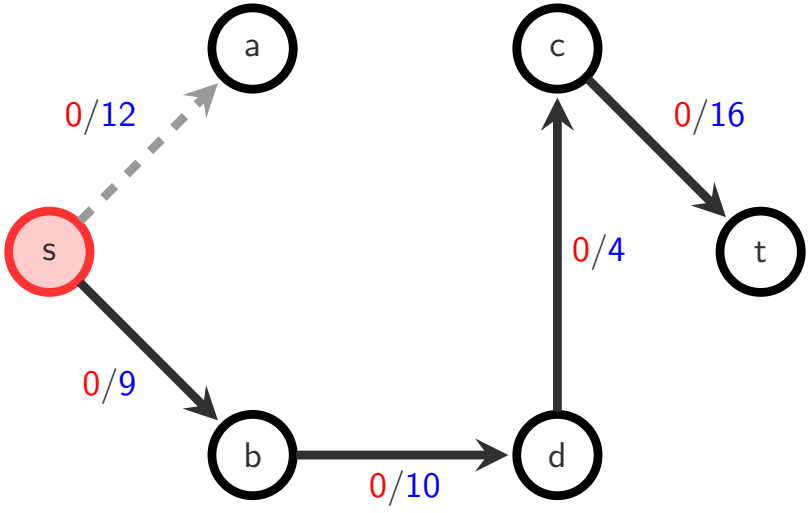
G, c and f

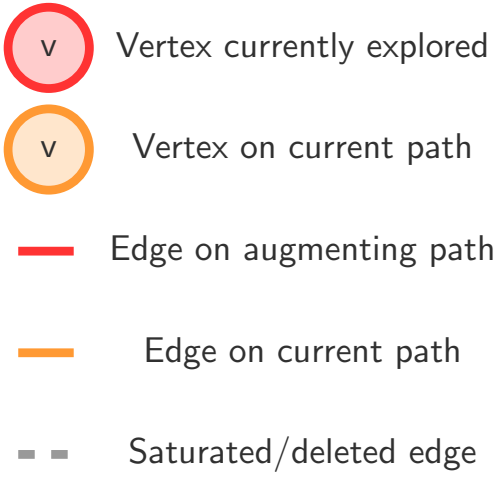


G_f and c_f



G_L, c_L and f'

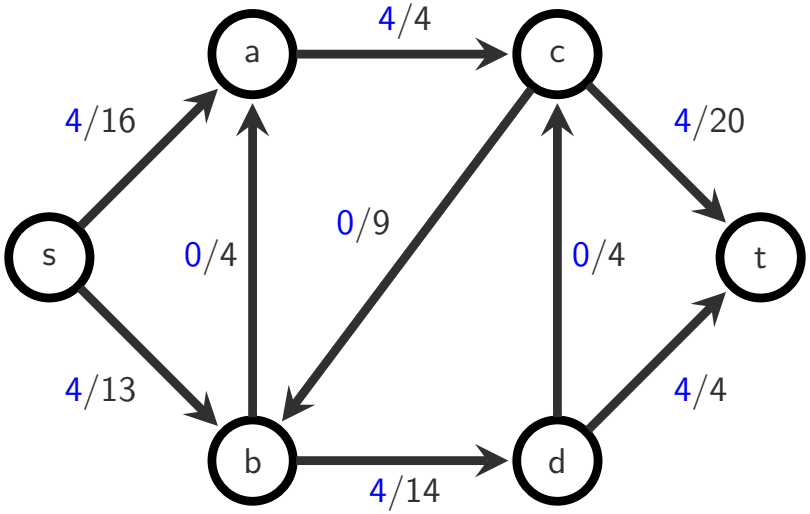




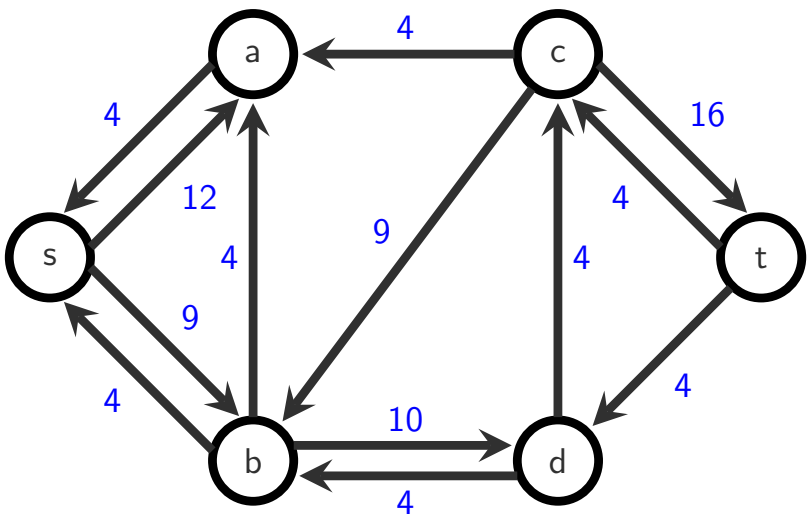
Current operation:

Find augmenting path

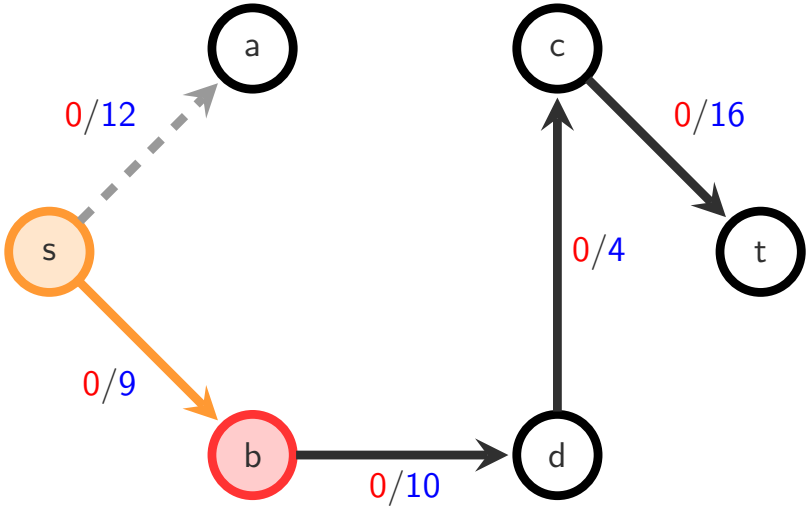
G, c and f

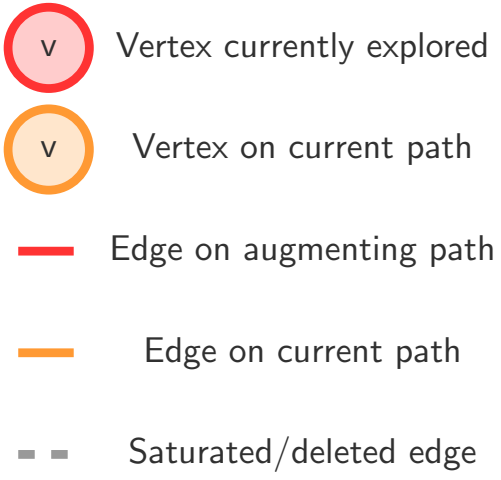


G_f and c_f



G_L, c_L and f'

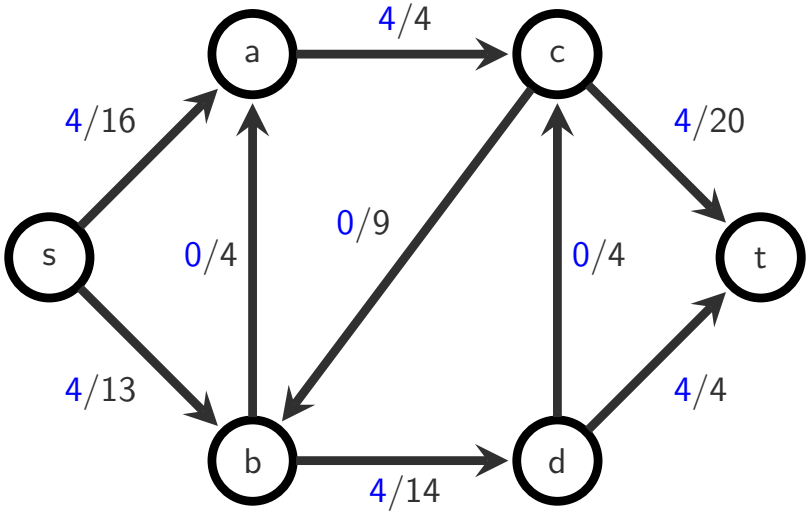




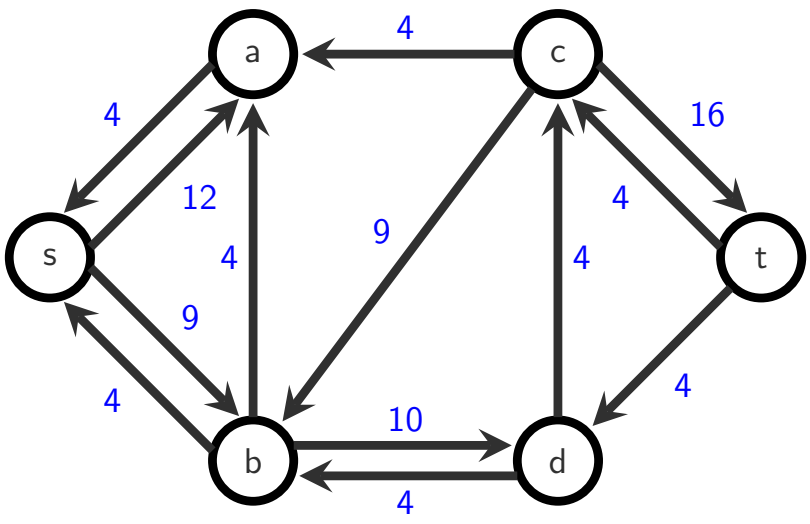
Current operation:

Find augmenting path

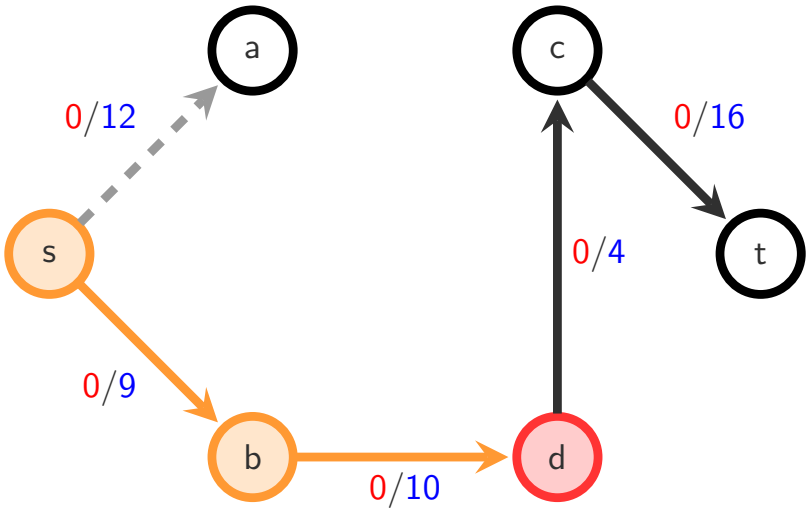
G, c and f

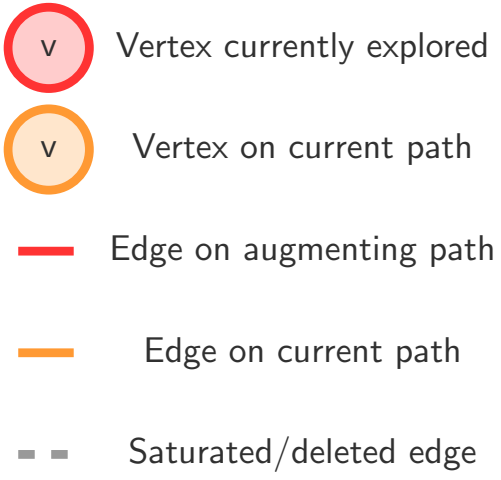


G_f and c_f



G_L, c_L and f'

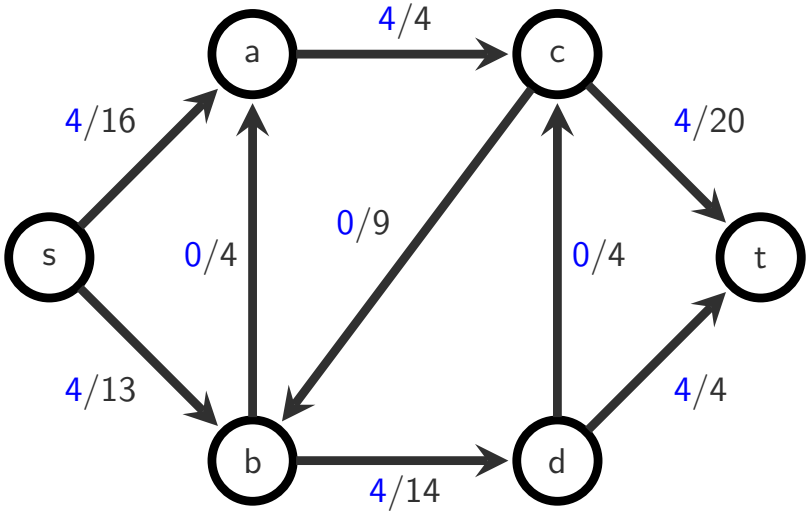




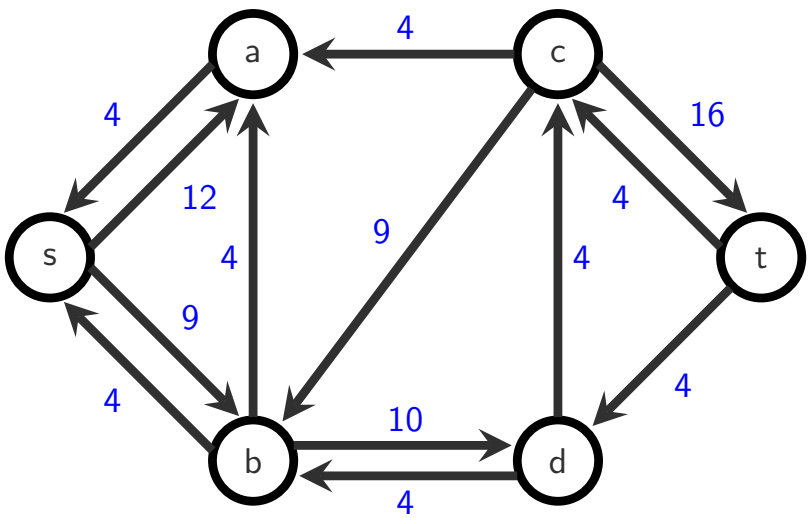
Current operation:

Find augmenting path

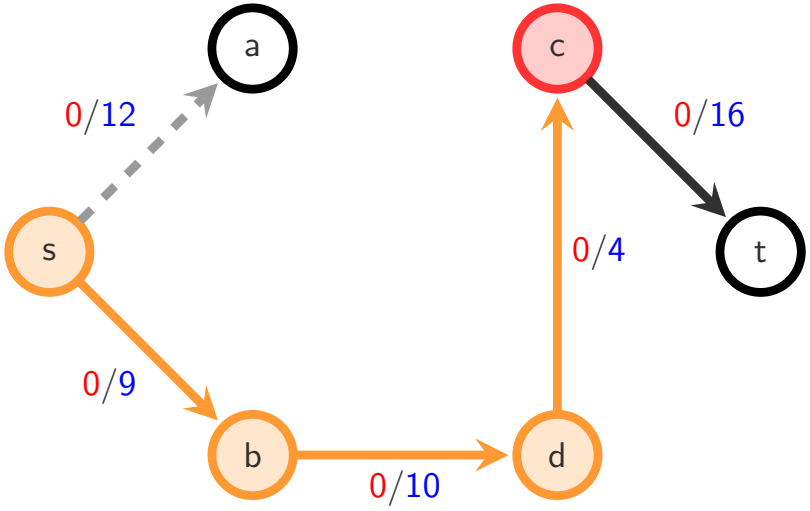
G, c and f

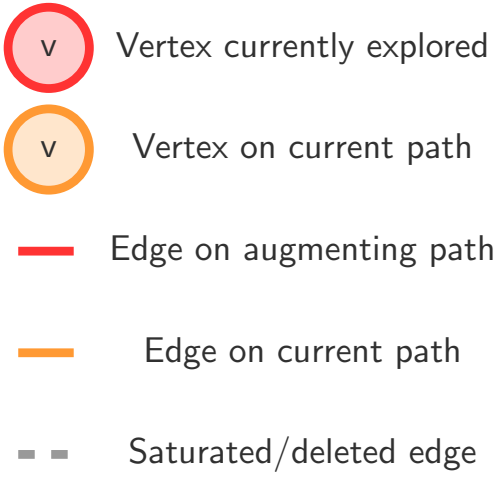


G_f and c_f



G_L, c_L and f'

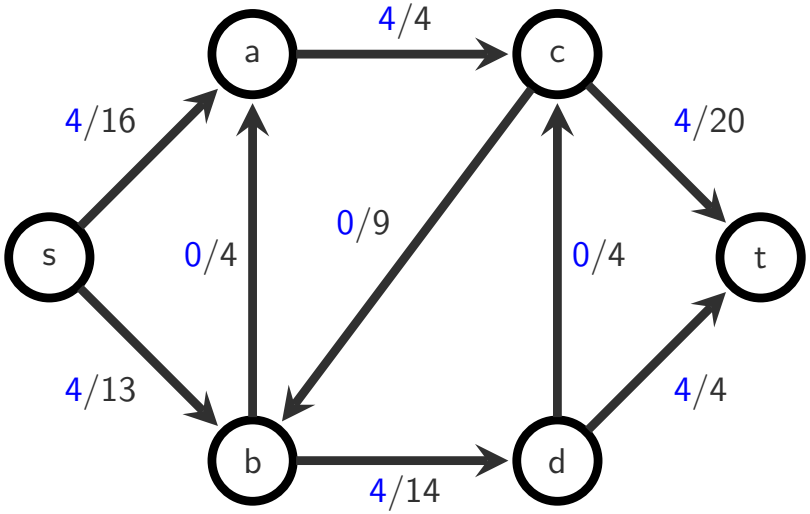




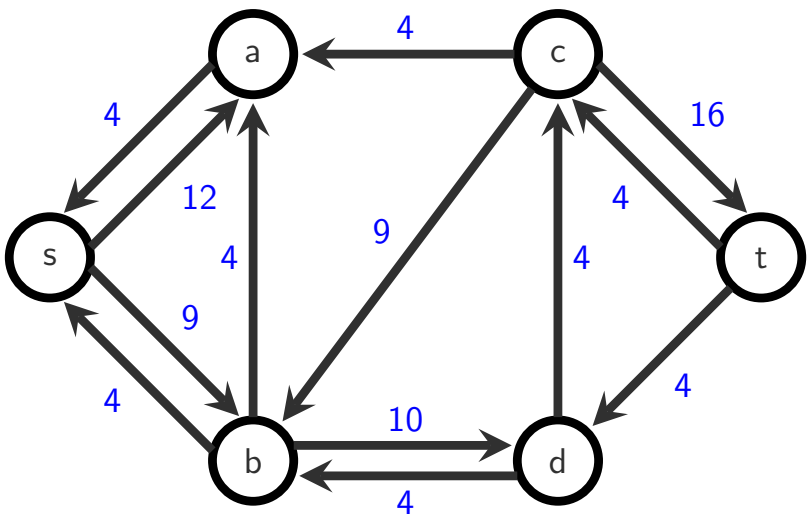
Current operation:

Find augmenting path

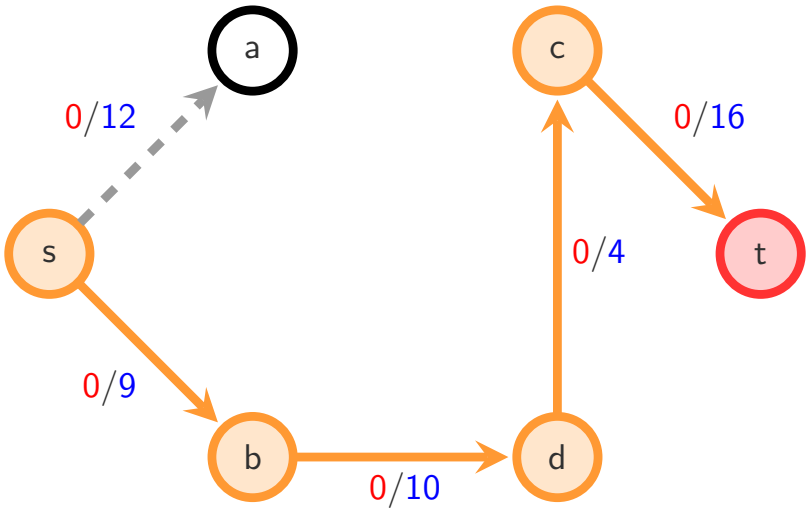
G, c and f

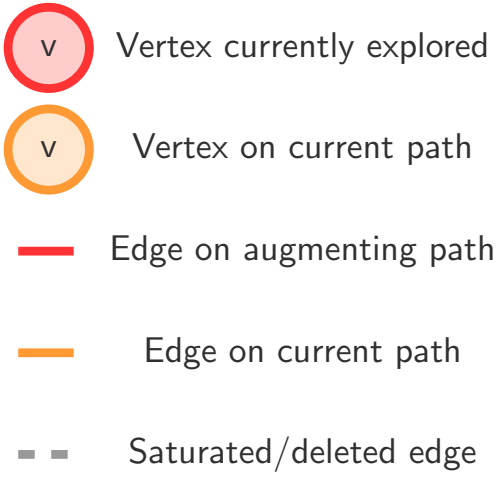


G_f and c_f



G_L, c_L and f'

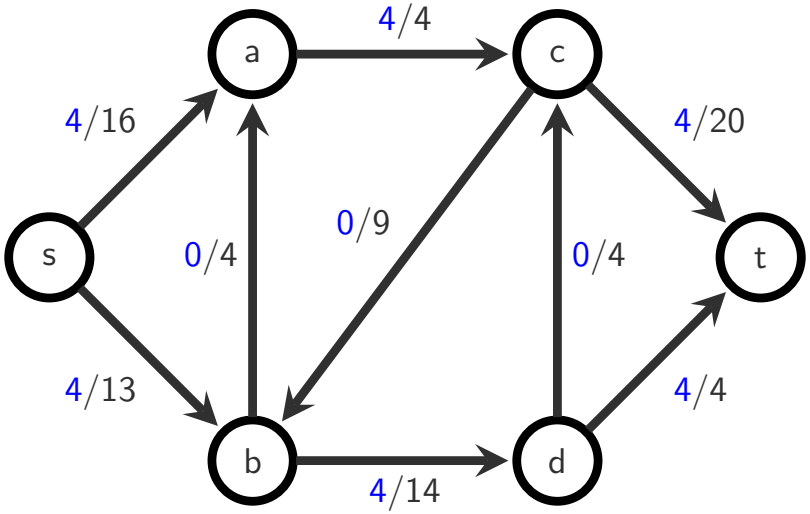




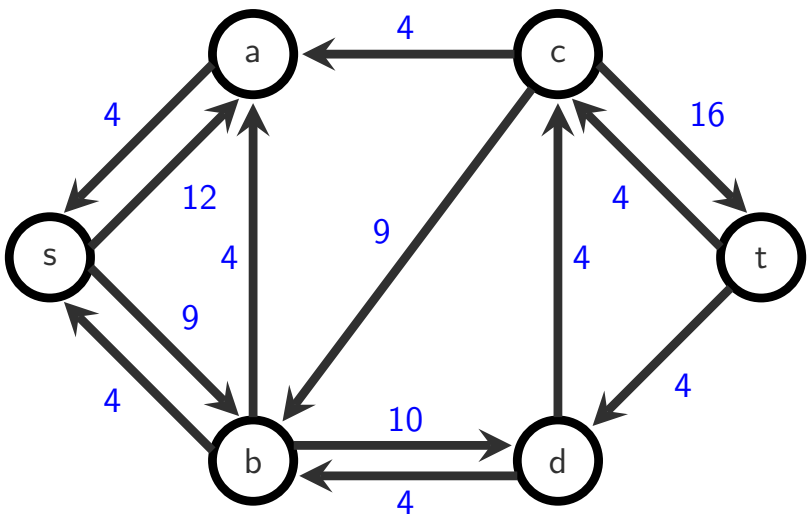
Current operation:

Find augmenting path

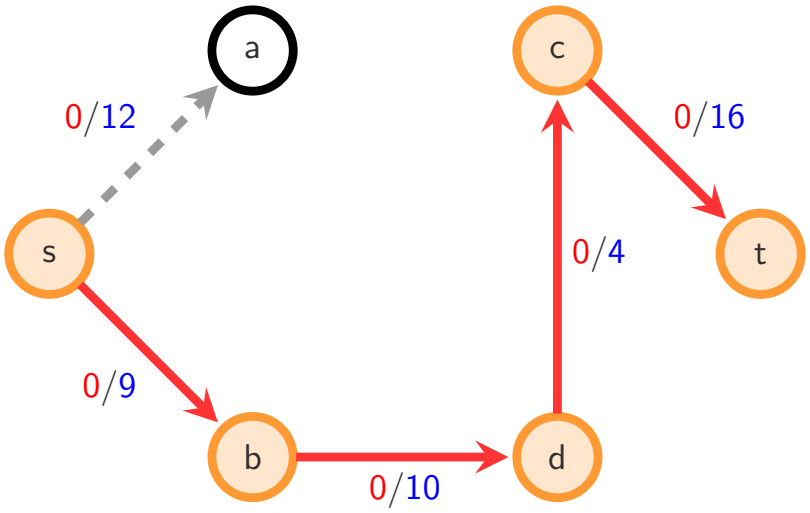
G, c and f

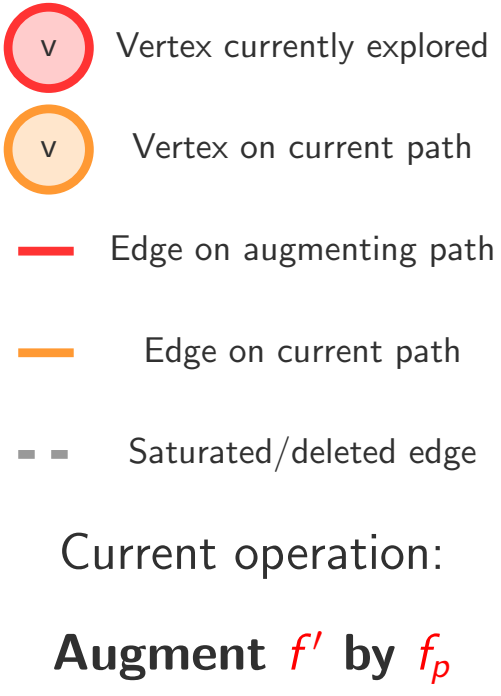


G_f and c_f

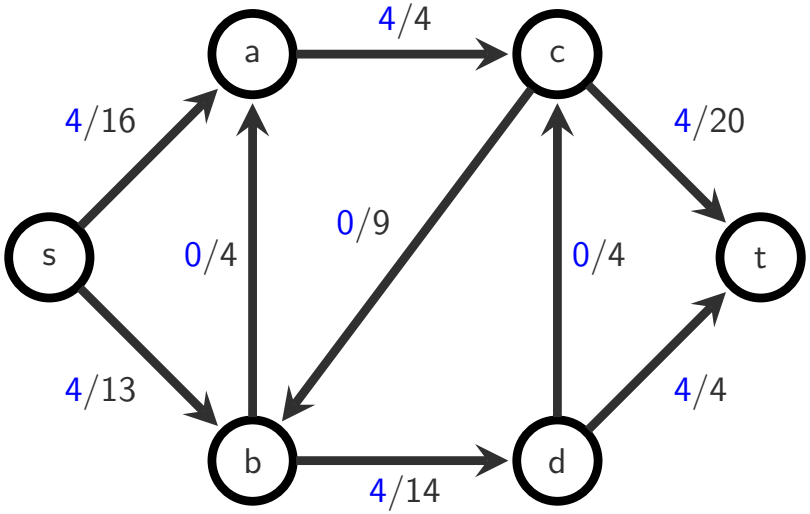


G_L, c_L and f'

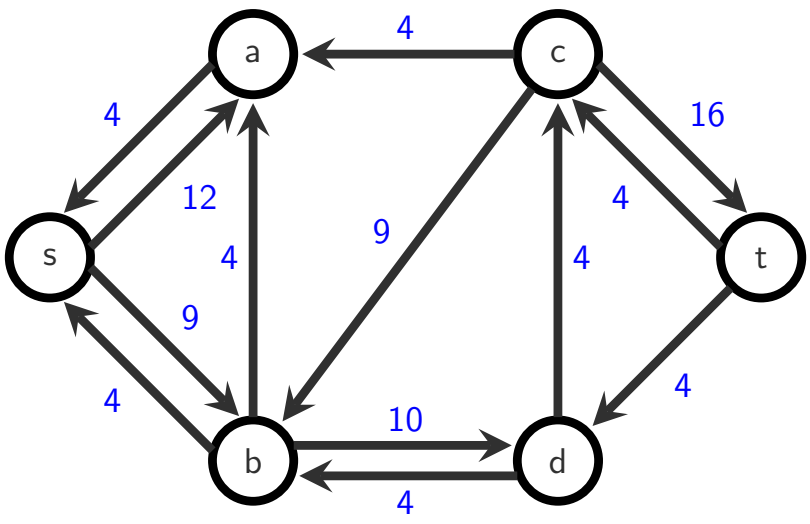




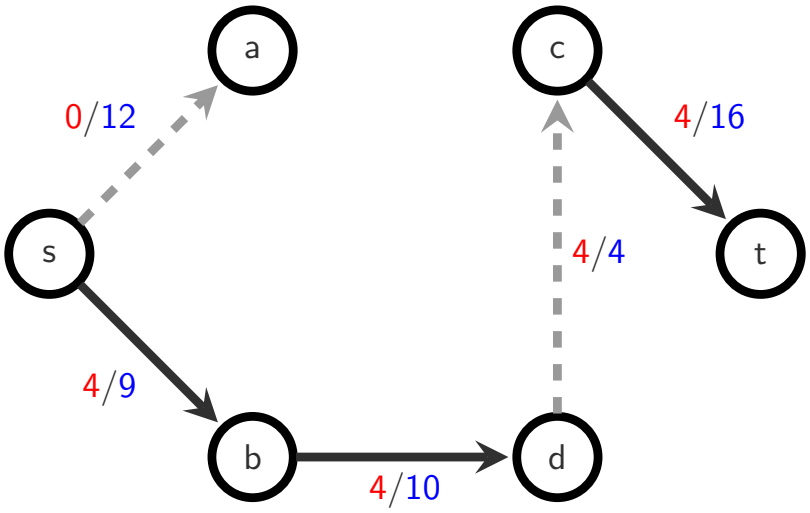
G, c and f



G_f and c_f



G_L, c_L and f'



v Vertex currently explored

v Vertex on current path

Edge on augmenting path

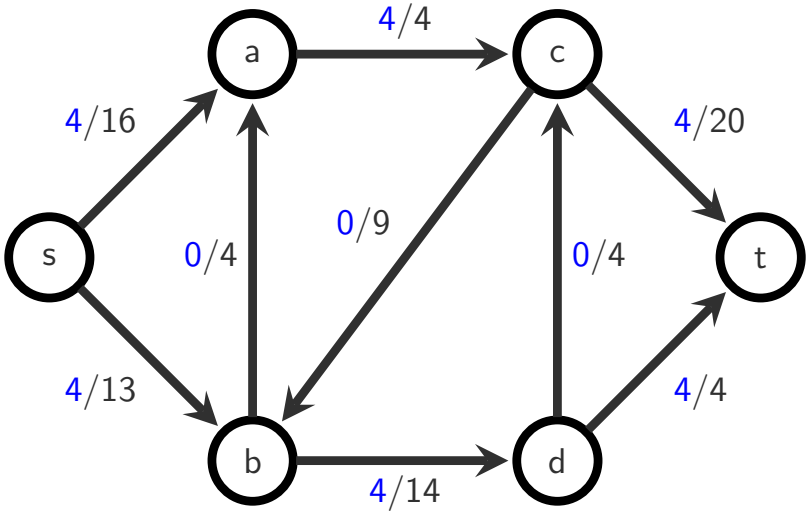
Edge on current path

Saturated/deleted edge

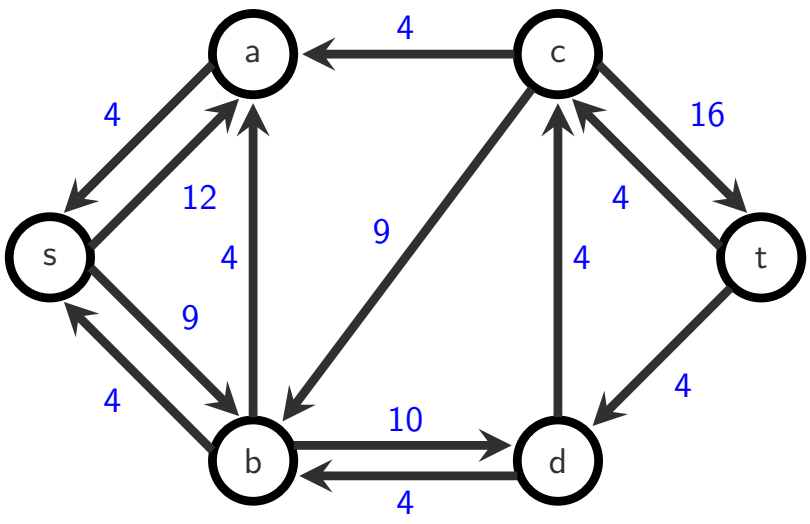
Current operation:

Augment f' by f_p

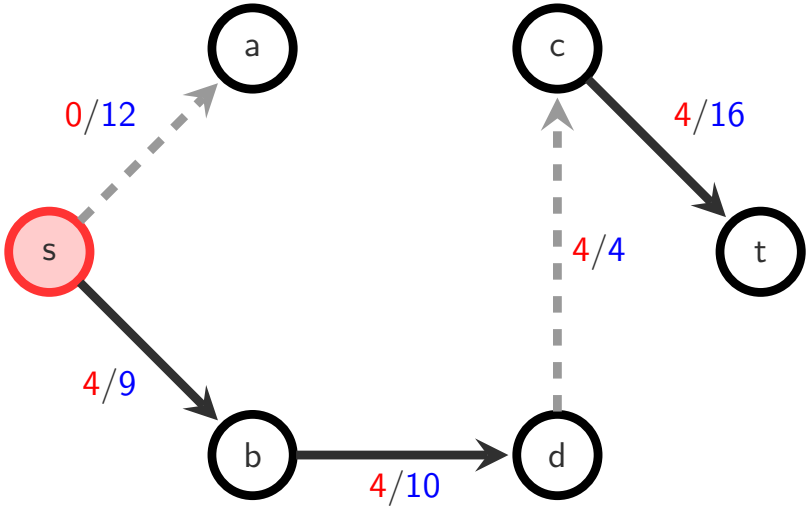
G, c and f

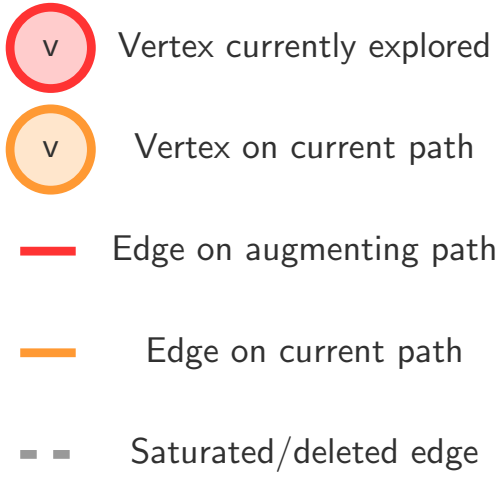


G_f and c_f



G_L, c_L and f'

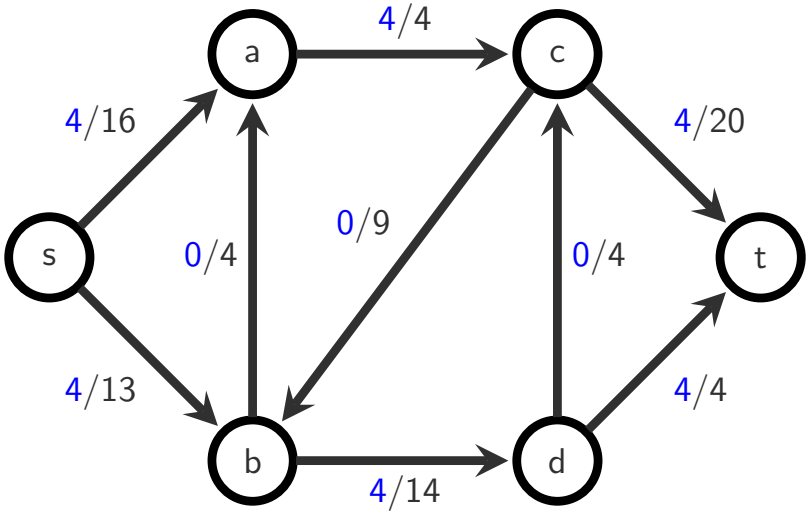




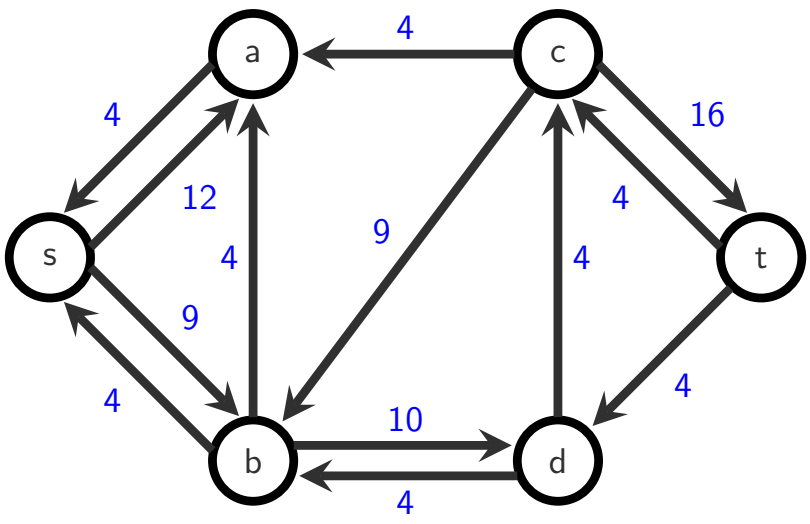
Current operation:

Find augmenting path

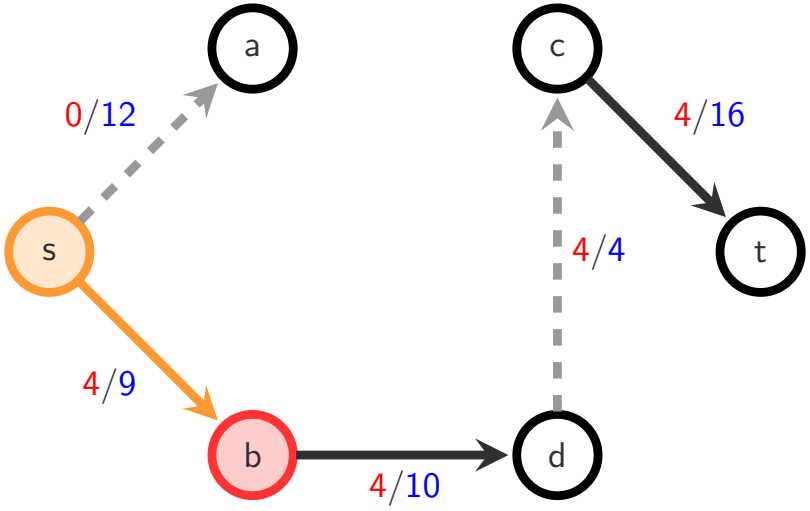
G, c and f

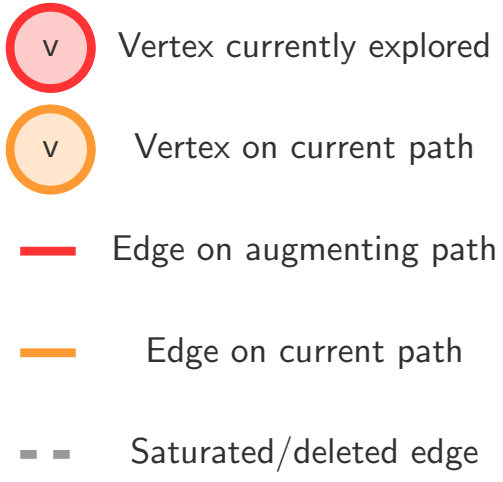


G_f and c_f



G_L, c_L and f'

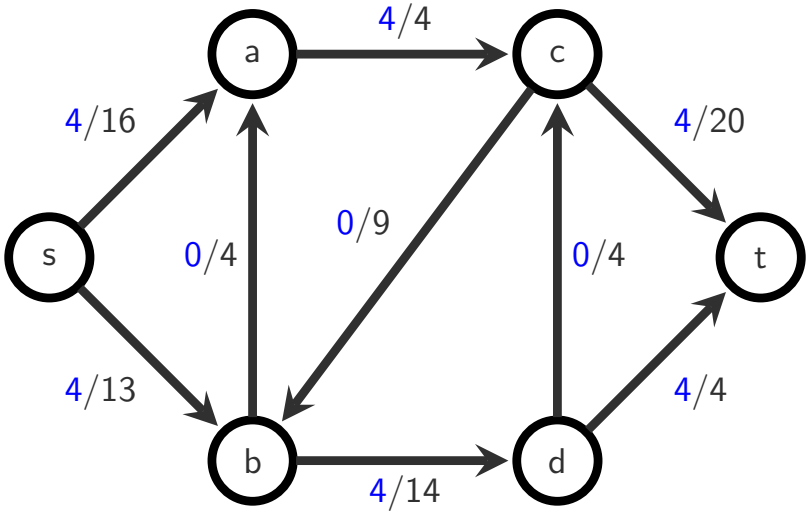




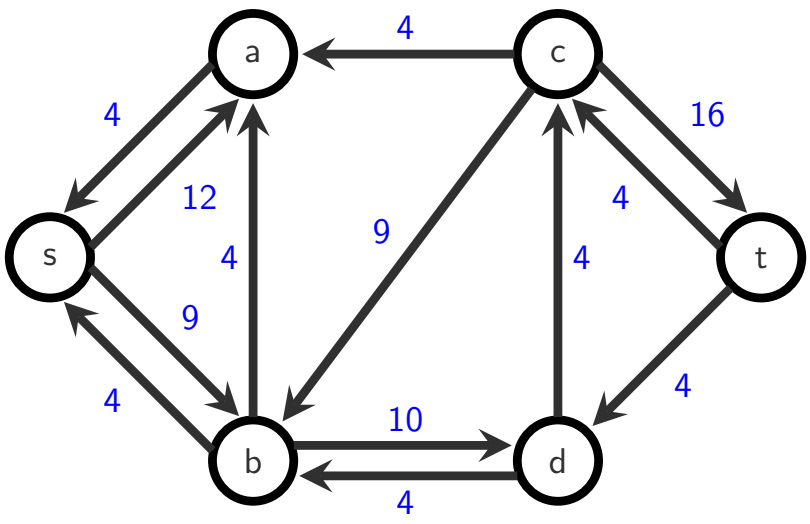
Current operation:

Find augmenting path

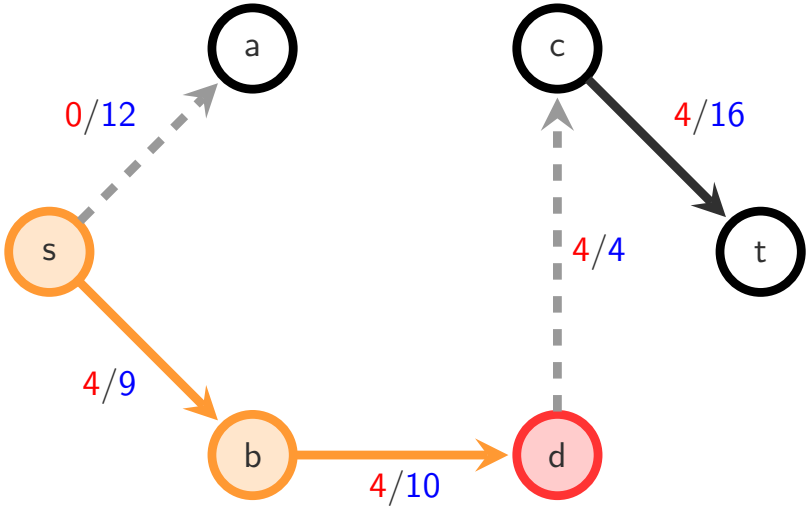
G, c and f

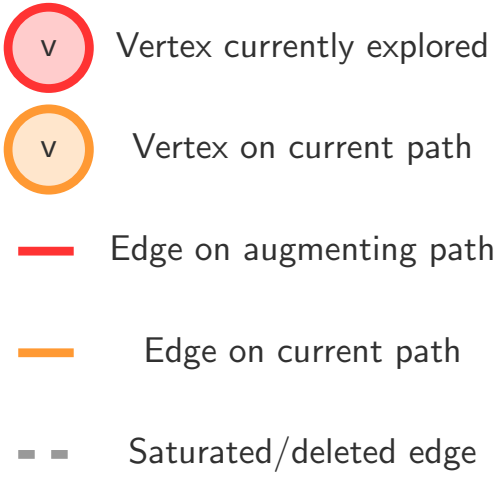


G_f and c_f



G_L, c_L and f'

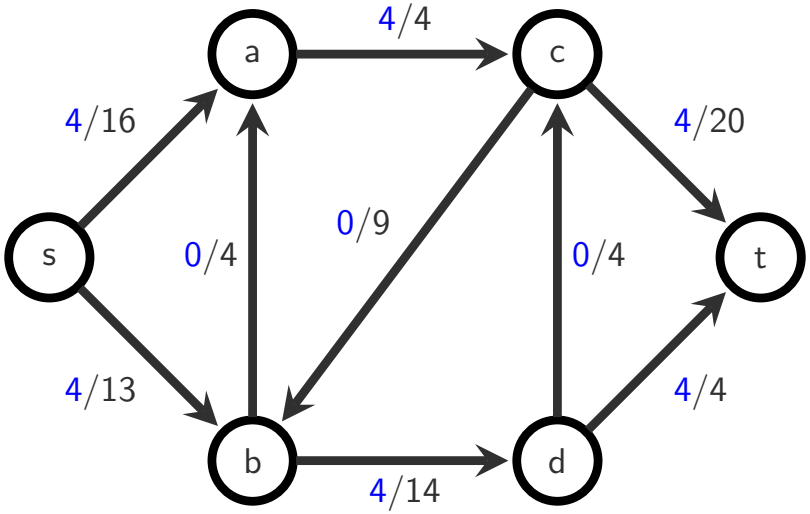




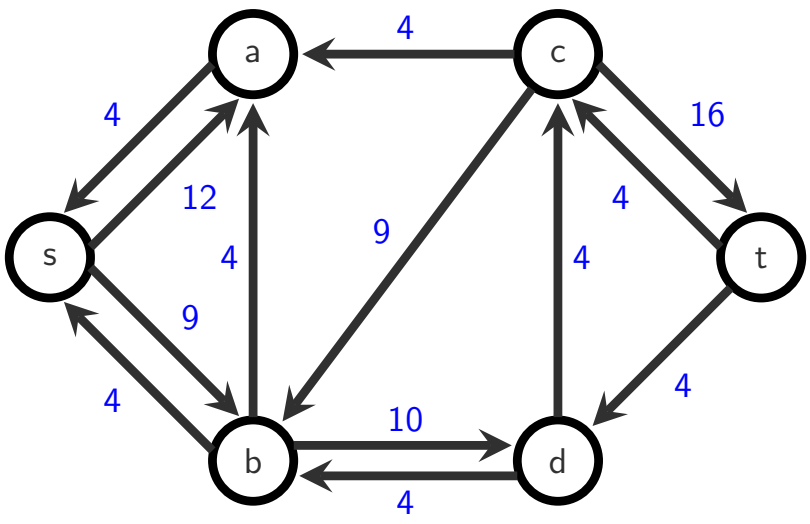
Current operation:

Find augmenting path

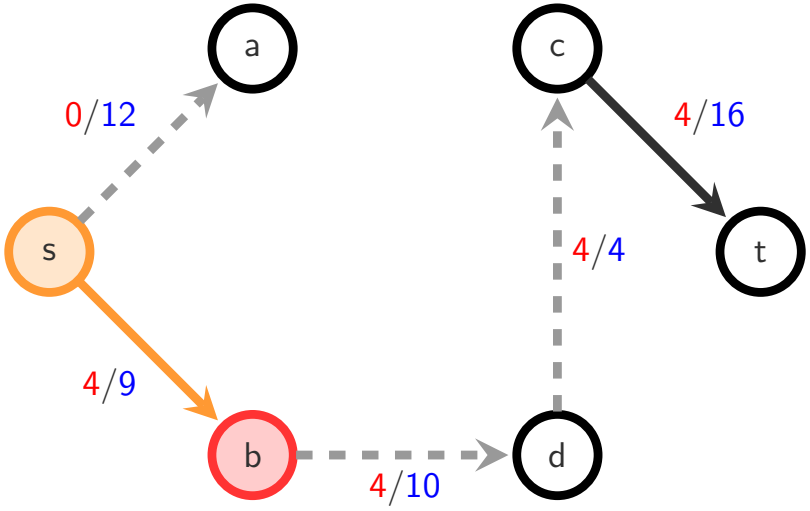
G, c and f

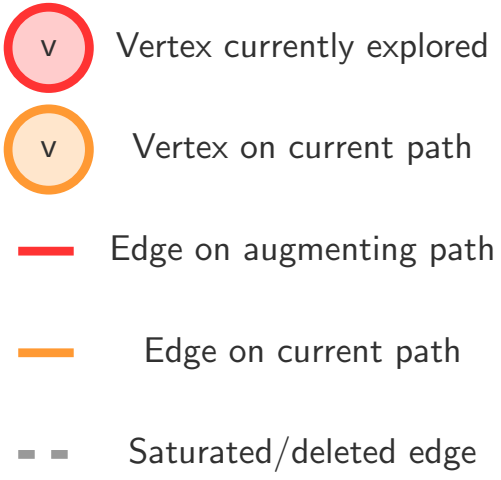


G_f and c_f



G_L, c_L and f'

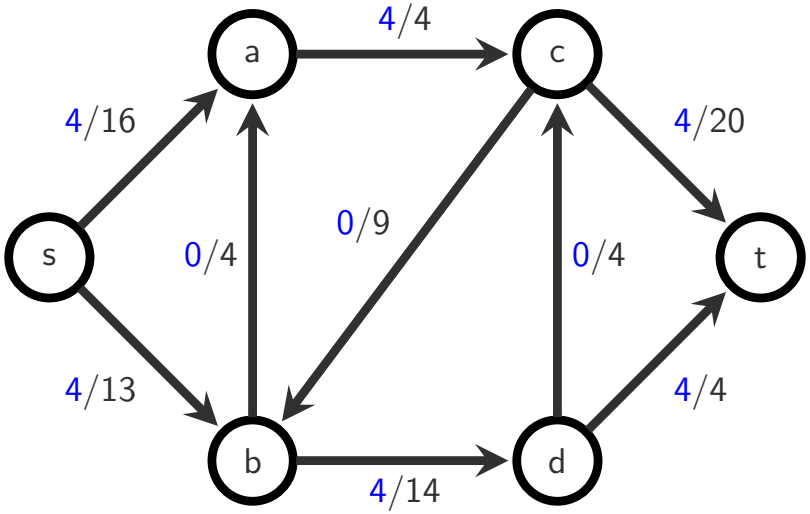




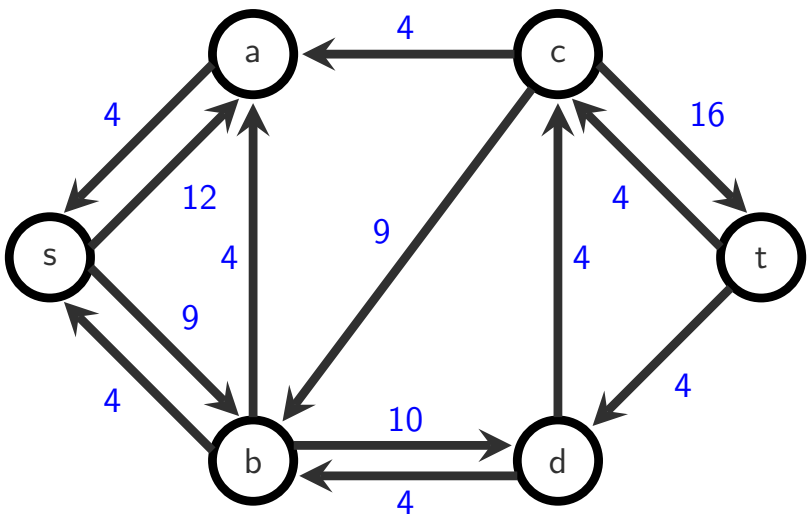
Current operation:

Find augmenting path

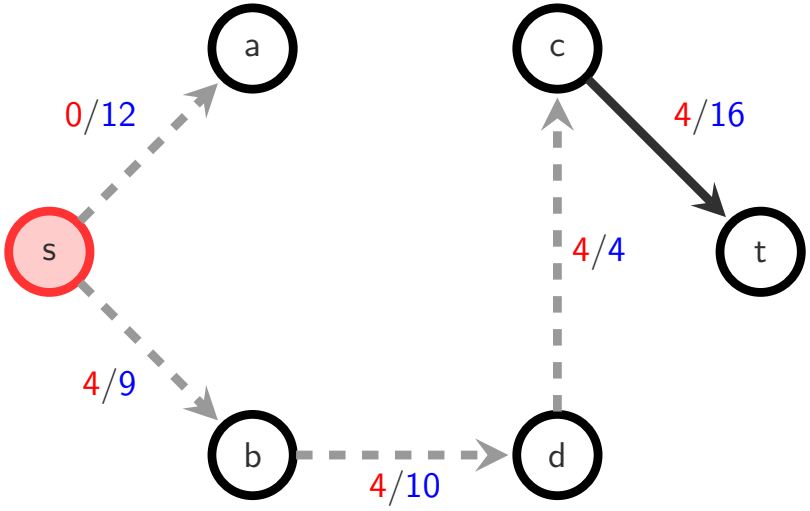
G, c and f

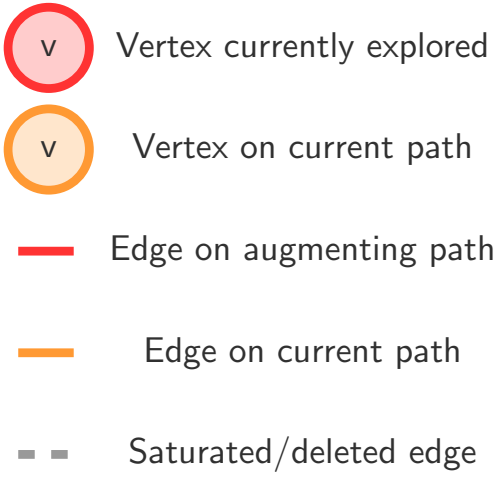


G_f and c_f



G_L, c_L and f'

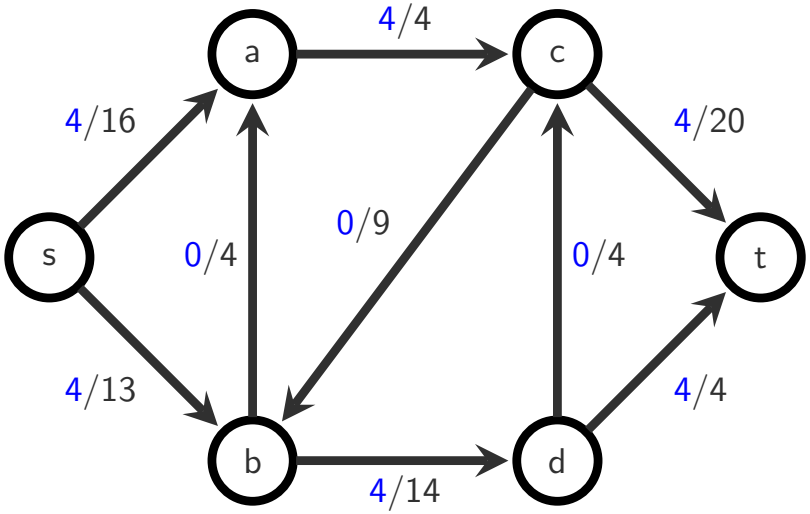




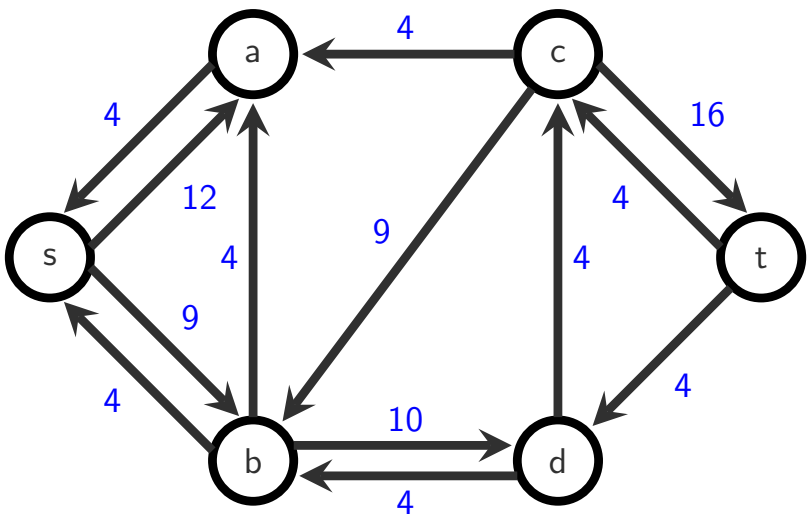
Current operation:

Find augmenting path

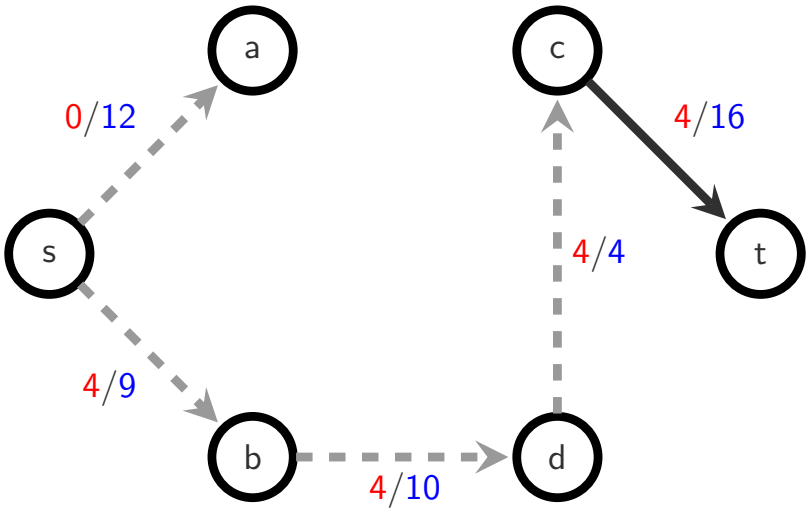
G, c and f



G_f and c_f



G_L, c_L and f'



v Vertex currently explored

v Vertex on current path

Edge on augmenting path

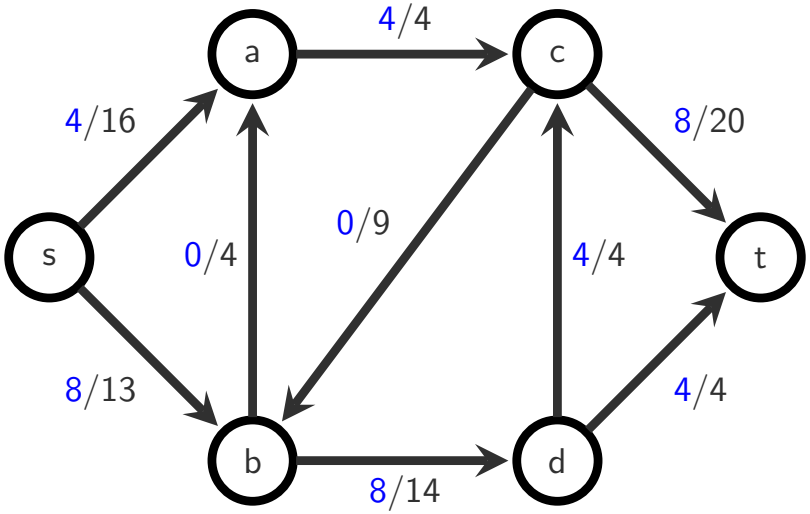
Edge on current path

Saturated/deleted edge

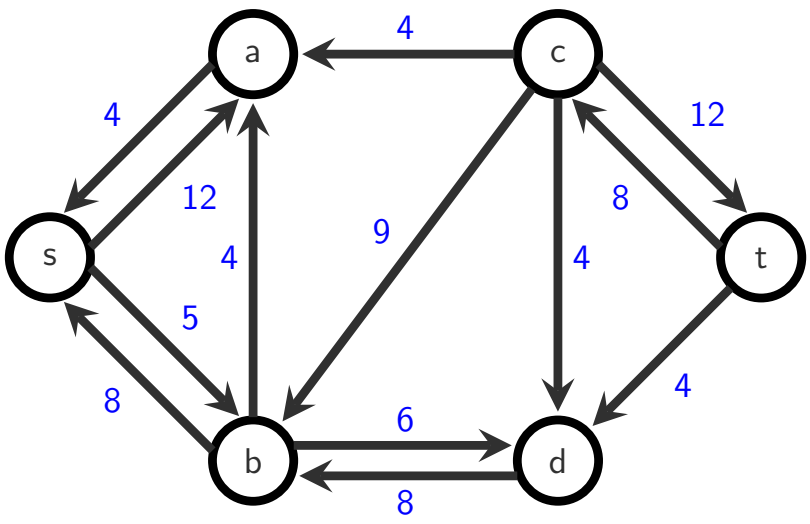
Current operation:

Augment f by f'

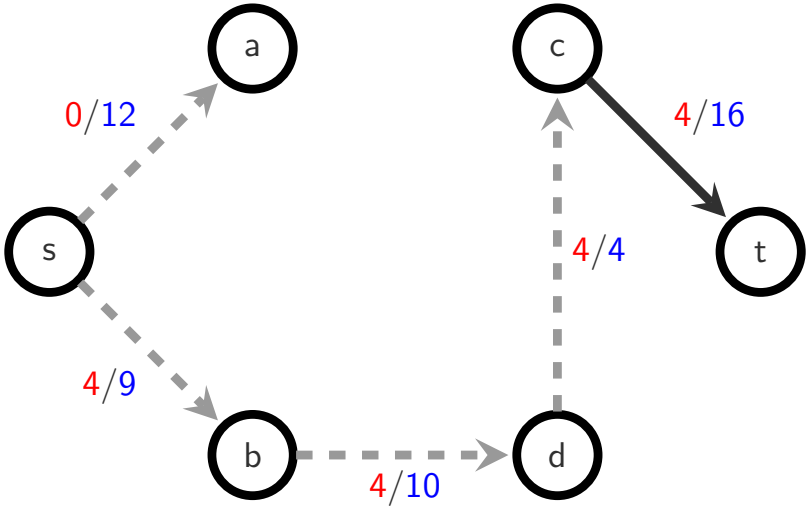
G, c and f

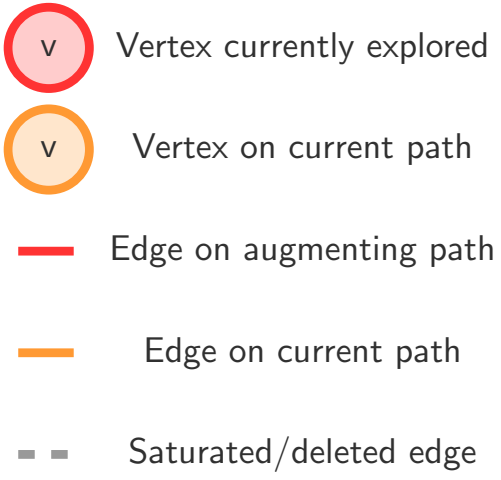


G_f and c_f



G_L, c_L and f'

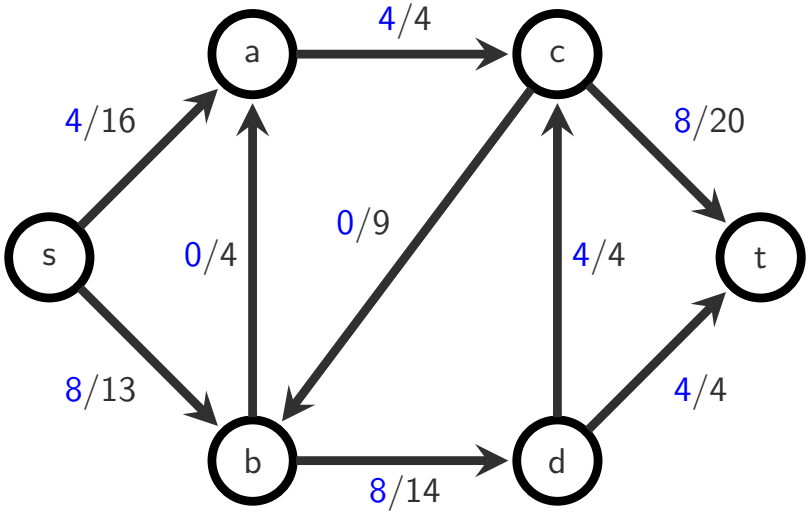




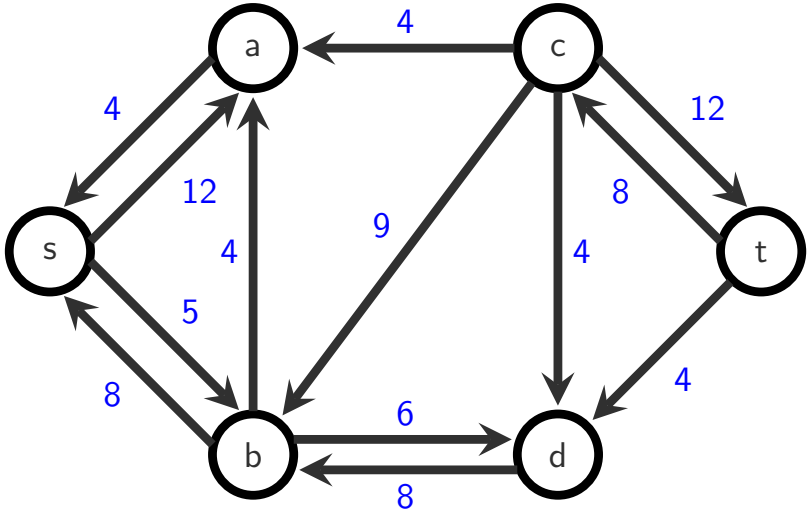
Current operation:

Augment f by f'

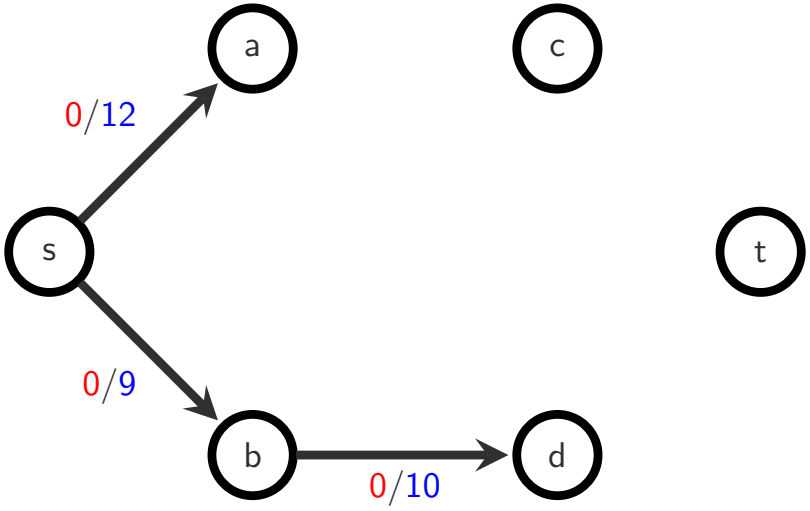
G, c and f

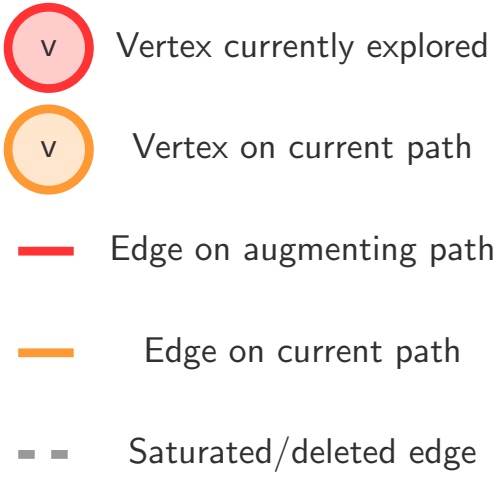


G_f and c_f



G_L, c_L and f'

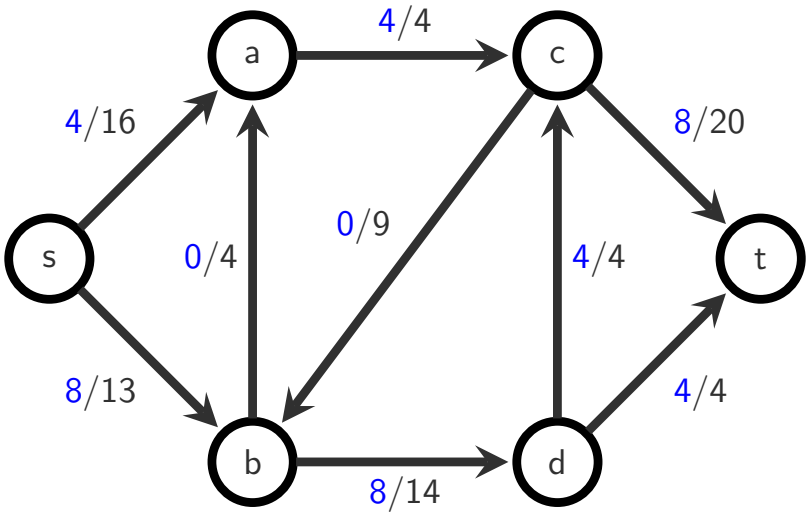




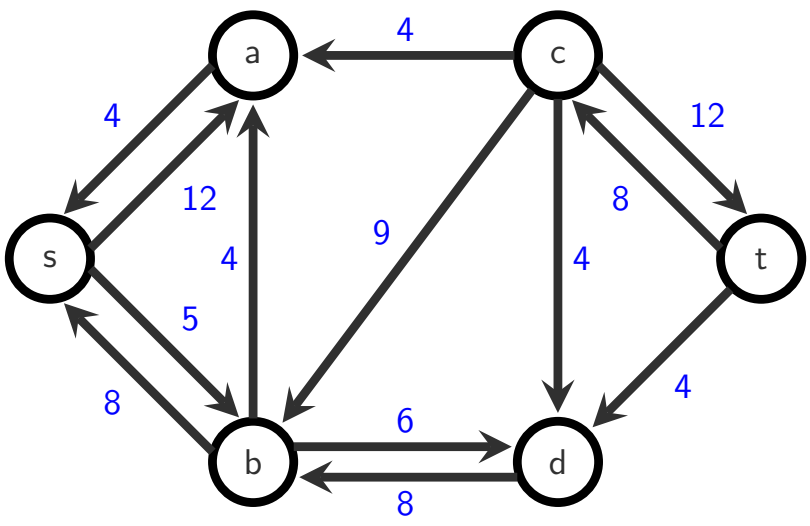
Current operation:

Find blocking flow

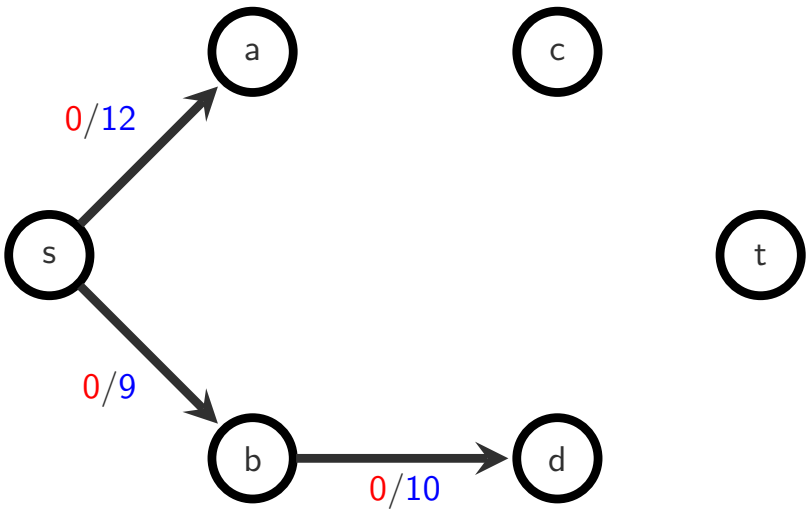
G, c and f

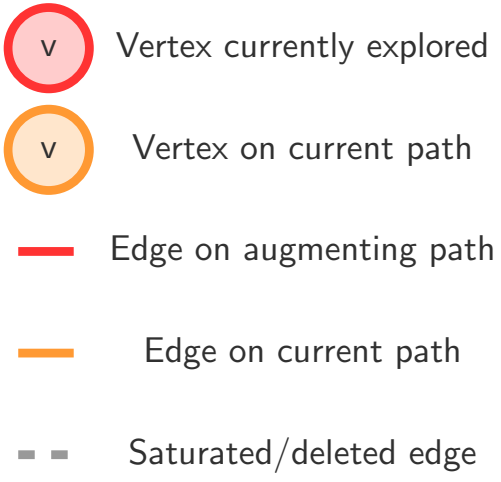


G_f and c_f



G_L, c_L and f'





No path from s to t:

Maximum flow found

Dinic's Algorithm: Analysis

- One can show: in each iteration, the blocking flow consists of augmenting paths of the same length, and this length increases each iteration.
- Hence: $\mathcal{O}(|V|)$ iterations.
- Each iteration (finding blocking flow) in $\mathcal{O}(|V|\,|E|)$
- Total: $\mathcal{O}(|V|^2\,|E|)$

Finding the blocking flow can also be done in $\mathcal{O}(|V|^2)$ using Karzanov's Variant for a total running time of $\mathcal{O}(|V|^3)$.

Dinic's Algorithm: Karzanov's Variant

Definition (Preflow)

Let (V, E, c, s, t) be a flow network. A function $f : E \to \mathbb{R}_{\geq 0}$ is called *preflow* if

$$\forall (u, v) \in E: \quad 0 \leq f(u, v) \leq c(u, v) \tag{3}$$

$$\forall u \in V \setminus \{s\}: \quad \text{in}_f(u) \geq \text{out}_f(u) \tag{4}$$

Definition (Excess)

Let (V, E, c, s, t) be a flow network and $f : E \to \mathbb{R}_{\geq 0}$. The *excess* of f at $v \in V$ is defined as

$$\text{ex}_f(v) := \text{in}_f(v) - \text{out}_f(v).$$

v is called *active* or *overflowing* if $\text{ex}_f(v) > 0$.

Dinic's Algorithm: Karzanov's Variant

Karzanov's idea: maintain a preflow f' and store the excess of each node, along with a stack that stores the last increases of the excess.

Algorithm 4 Finding blocking flows: Karzanov's Variant

```
1:  (v_1, v_2, ..., v_n) ← TopologicalOrder(G_L)
2:  for each v ∈ V \ {s, t} do
3:      ex[v] ← 0
4:      p[v] ← EmptyStack()
5:      frozen[v] ← false
6:  end for
7:  f' ← 0
8:  for each v ∈ sE_L do
9:      e ← (s, v)
10:     f'(e) ← c_L(e)
11:     if v ≠ t then
12:         ex[v] ← c_L(e)
13:         p[v].push((e, c_L(e)))
14:     end if
15: end for
16: while there is an active vertex do
17:     for i = 1, 2, ..., n do
18:         if v_i is active then
19:             for each w ∈ v_i E_L do e ← (v_i, w)
20:                 if f'(e) < c_L(e) and not frozen[w] then
21:                     δ ← min{c_L(e) − f'(e), ex[v_i]}
22:                     f'(e) ← f'(e) + δ
23:                     ex[v_i] ← ex[v_i] − δ
24:                     ex[w] ← ex[w] + δ
25:                     p[w].push((e, δ))
26:                 end if
27:             end for
28:         end if
29:     end for
30:     Let i be maximal s.t. v_i is active
31:     while v_i is active do
32:         (e, δ) ← p[v_i].pop()
33:         δ' ← min{δ, ex[v_i]}
34:         f'(e) ← f'(e) − δ'
35:         ex[v_i] ← ex[v_i] − δ'
36:         ex[v] ← ex[v] + δ' where (v, v_i) = e
37:     end while
38:     frozen[v_i] ← true
39: end while
```

Dinic-Karzanov-Algorithm: Remarks

- Finds blocking flow in $\mathcal{O}(|V|^2)$
- But: quite complicated
- Better: Use Push-Relabel-Algorithms!

Push-Relabel Algorithms

The class of *push-relabel* algorithms for maximum flow work by maintaining a *preflow* and pushing it along edges, while (re-)labeling vertices to determine where flow can be pushed.

Push-Relabel-Algorithms: Height labels

Definition (Height function)

For a given flow network G and a flow f, a function $h\colon V \to \mathbb{N}$ is a *height function* (or *distance labeling*) if

$$\begin{aligned} h(s) &= |V| \\ h(t) &= 0 && \text{and} \\ h(u) &\leq h(v) + 1 && \text{for every residual edge } (u,v) \in E_f. \end{aligned}$$

An edge $(u,v) \in E_f$ is called *admissible* if $h(u) = h(v) + 1$.

Note: If h is a height function, the distance of v to t in G_f is at least $h(v)$.

Push and relabel operations

Algorithm 5 Push operation

▷ Applies to $(u, v) \in E_f$ when u is overflowing and $h(u) = h(v) + 1$
$\delta \leftarrow \min(\text{ex}[u], c_f(u, v))$
if $(u, v) \in E$ **then**
    $f(u, v) \leftarrow f(u, v) + \delta$
else
    $f(v, u) \leftarrow f(v, u) - \delta$
end if
$\text{ex}[u] \leftarrow \text{ex}[u] - \delta$
$\text{ex}[v] \leftarrow \text{ex}[v] + \delta$

Algorithm 6 Relabel operation

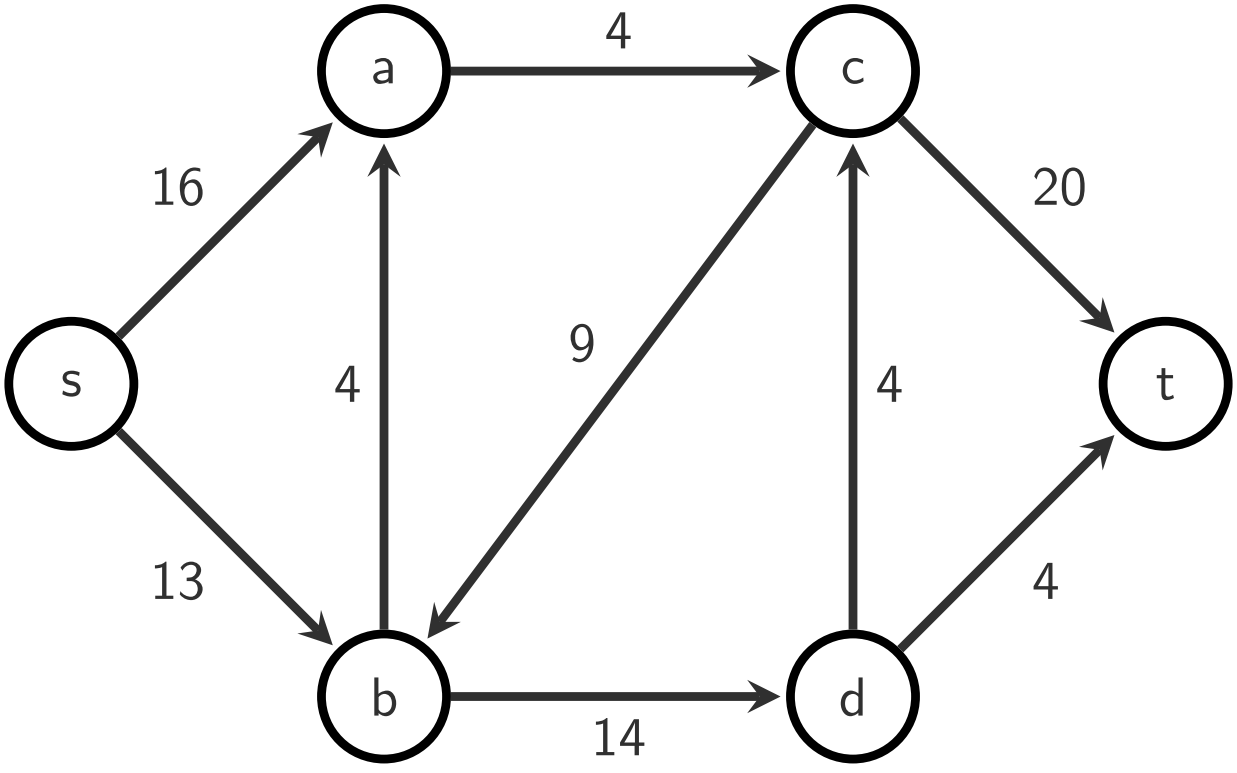
▷ Applies to u when u is overflowing and $h(u) \leq h(v)$ for all $(u, v) \in E_f$
$h(u) \leftarrow 1 + \min\{h(v) : (u, v) \in E_f\}$

Push-Relabel-Algorithm (Goldberg-Tarjan algorithm)

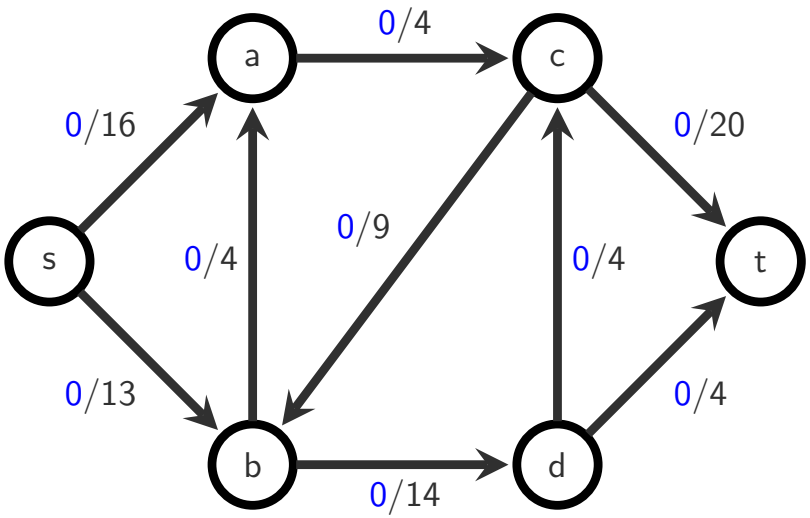
Algorithm 7 Push-relabel algorithm

for each vertex $v \in V$ **do**
    $h(v) \leftarrow 0$; $\text{ex}[v] \leftarrow 0$
end for
for $(u, v) \in E$ **do**
    $f(u, v) \leftarrow 0$
end for
$h(s) \leftarrow |V|$
for each vertex $v \in sE$ **do**
    $f(s, v) \leftarrow c(s, v)$
    $\text{ex}[v] \leftarrow \text{ex}[v] + c(s, v)$
    $\text{ex}[s] \leftarrow \text{ex}[s] - c(s, v)$
end for
while there is an applicable push or relabel operation **do**
    select an applicable push or relabel operation and perform it
end while

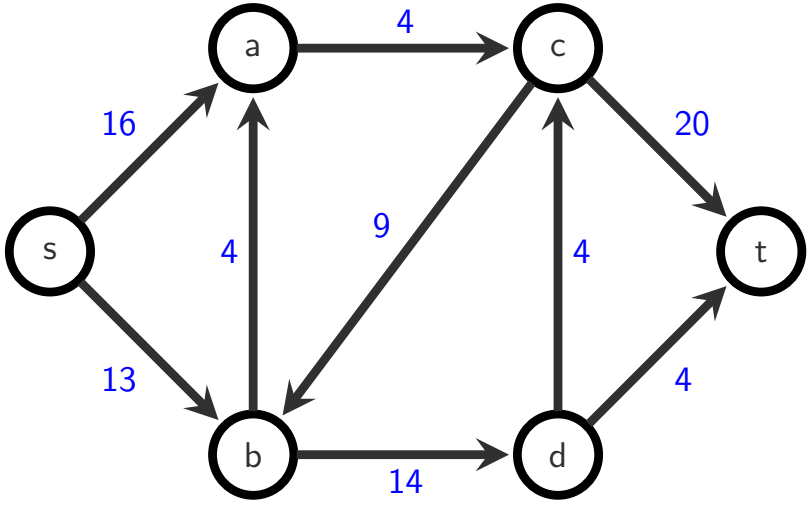
Push-relabel algorithm (example)



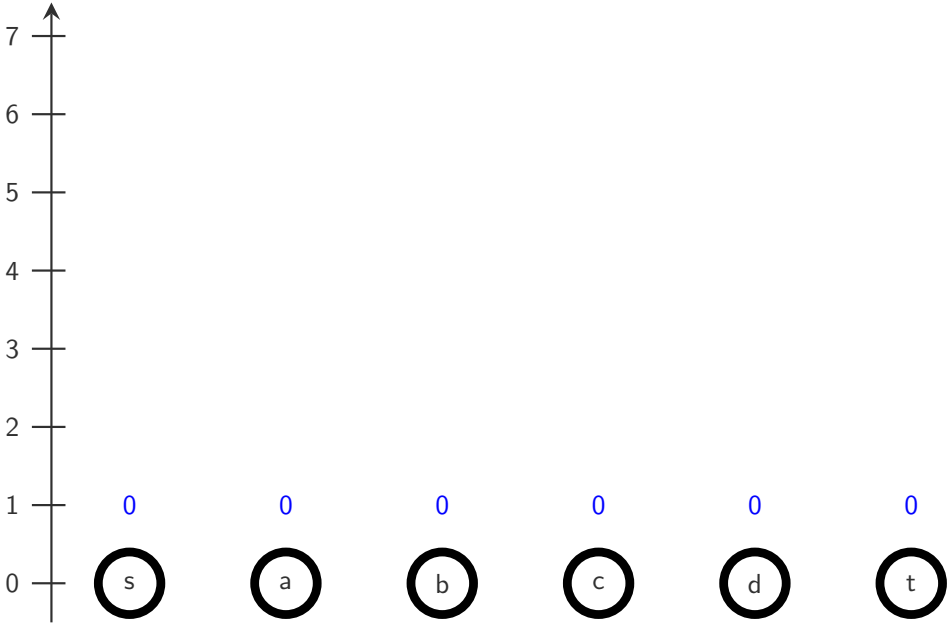
G, c and f

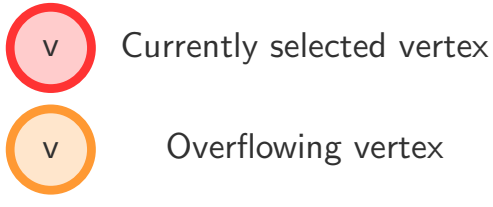


G_f and c_f



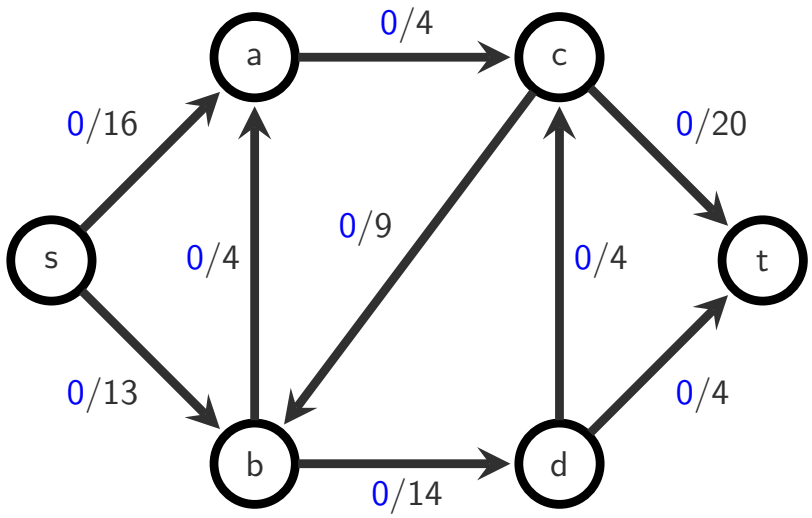
$h(v)$ and $e(v)$



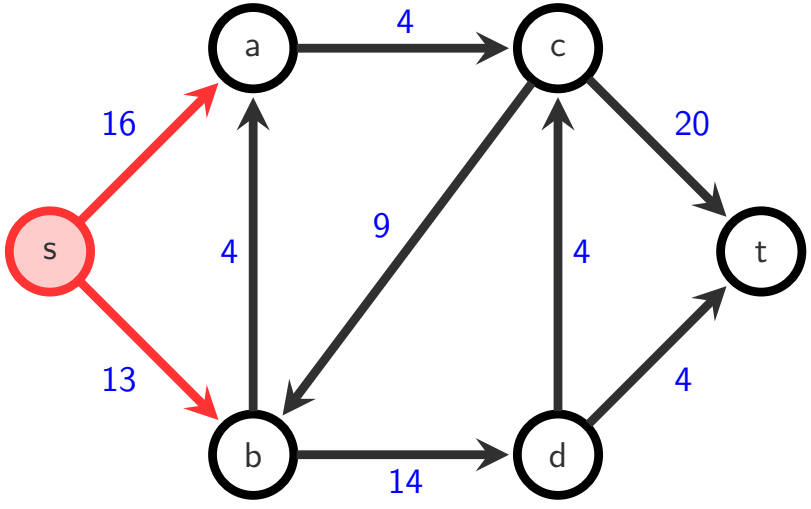


Current operation:

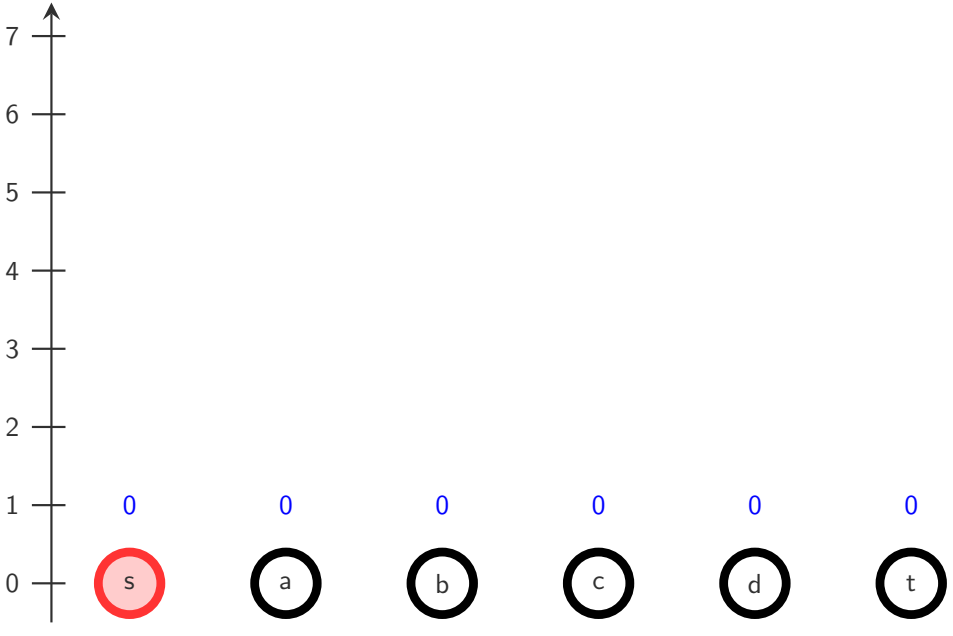
G, c and f

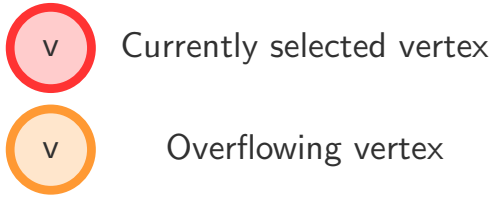


G_f and c_f



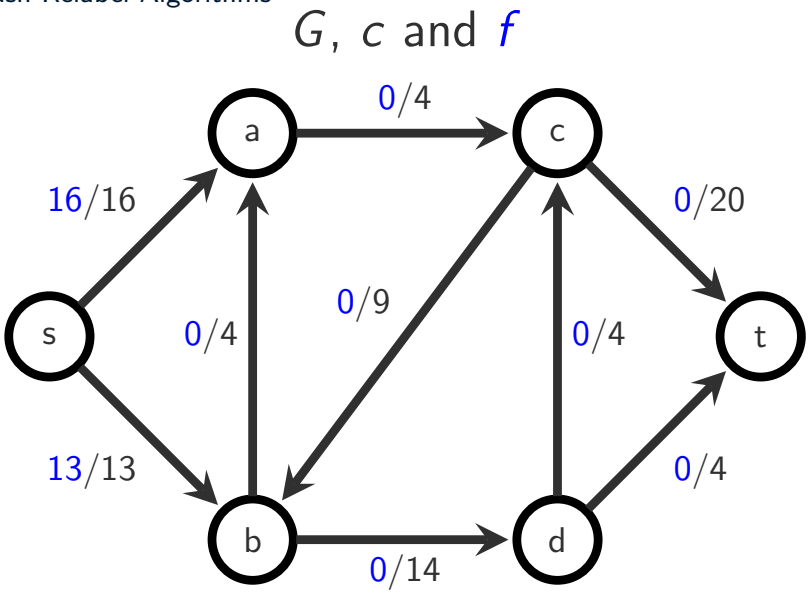
$h(v)$ and $e(v)$

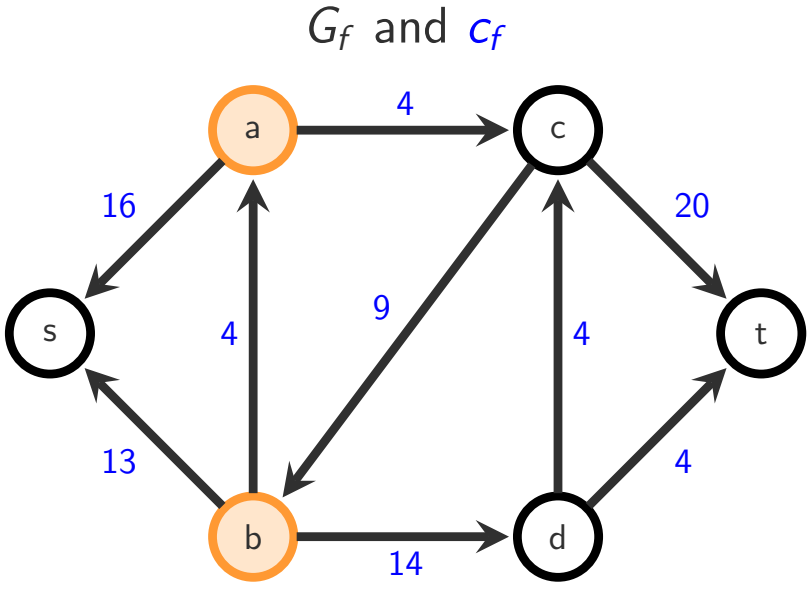


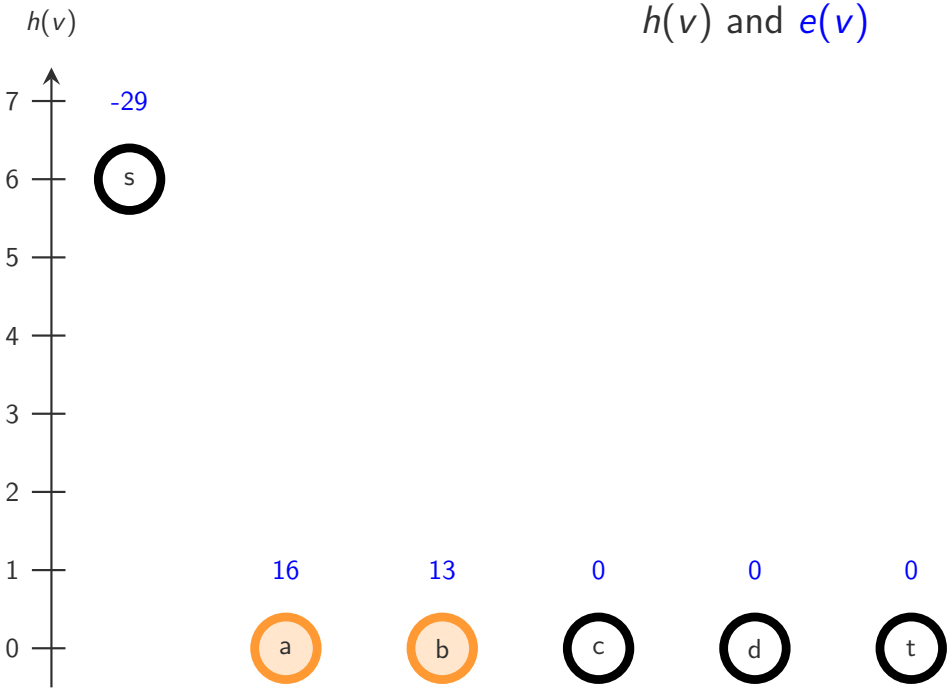


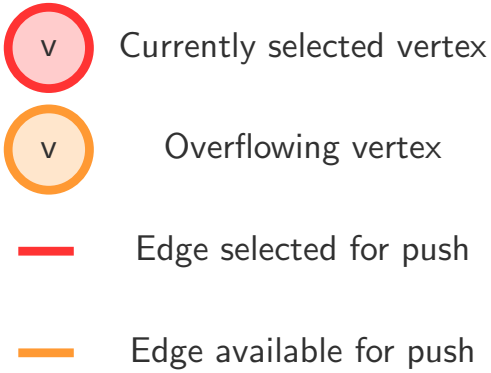
Current operation:

Initialize preflow

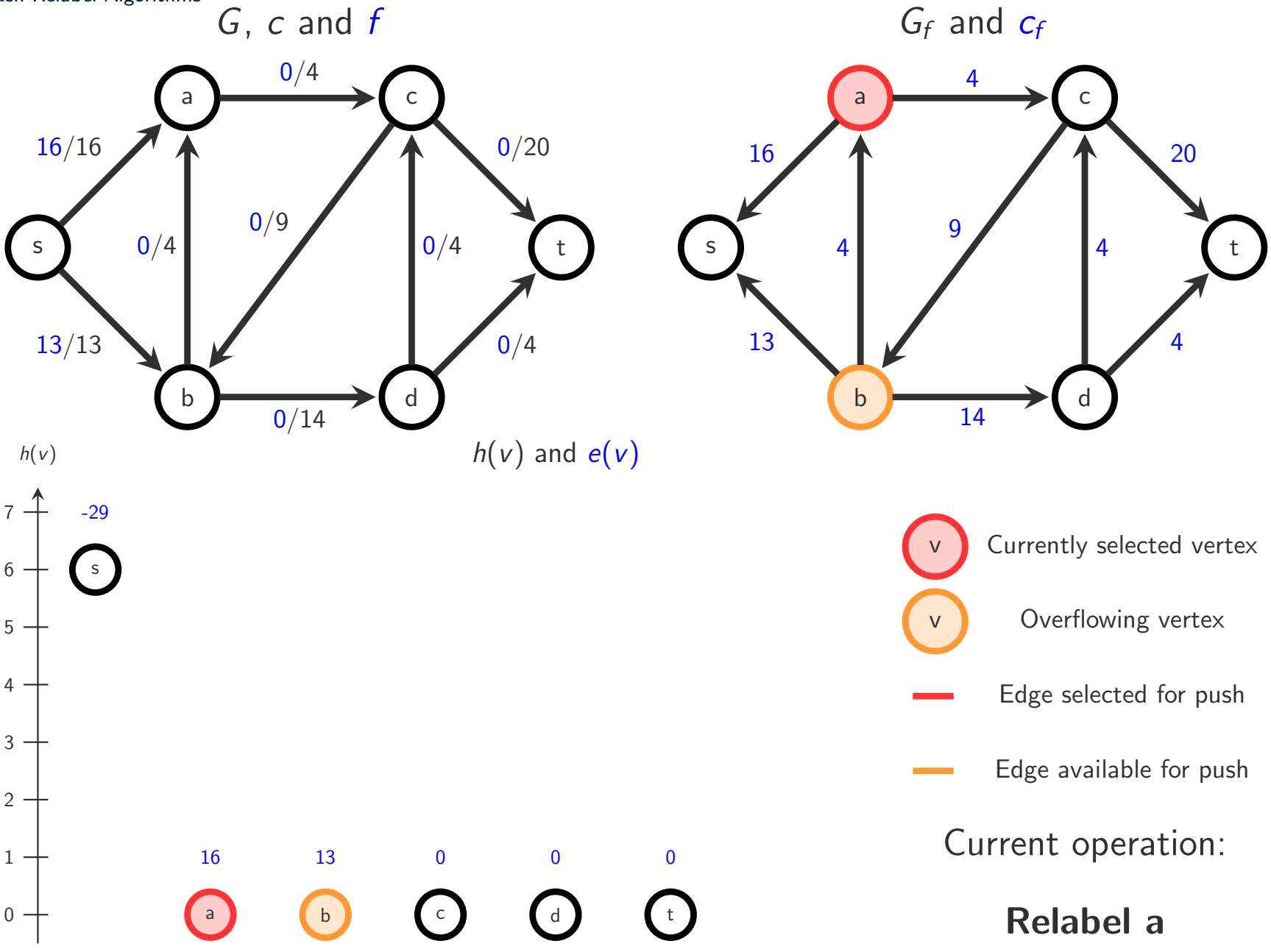




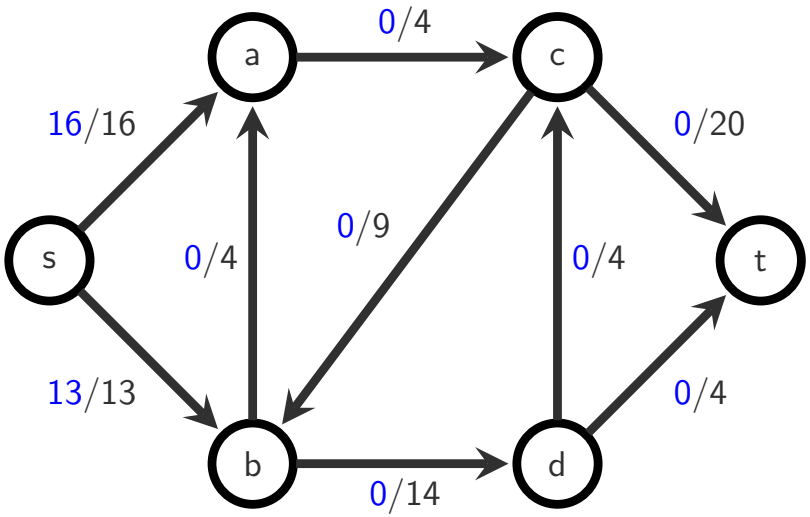




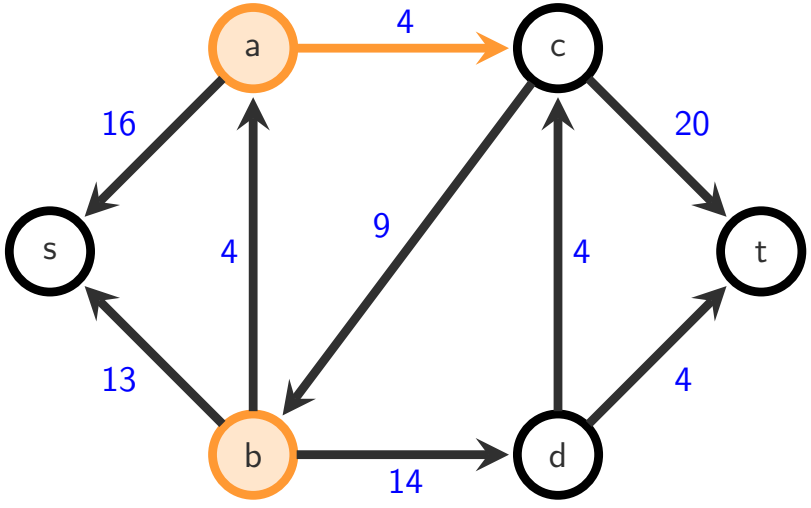
Current operation:


G, c and f
0/4
16/16
0/20
0/9
0/4
0/4
13/13
0/4
0/14
a
c
s
t
b
d
G_f and c_f
4
16
20
4
9
4
13
4
14
h(v) and e(v)
h(v)
7
6
5
4
3
2
1
0
-29
16
13
0
0
0
Currently selected vertex
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Relabel a

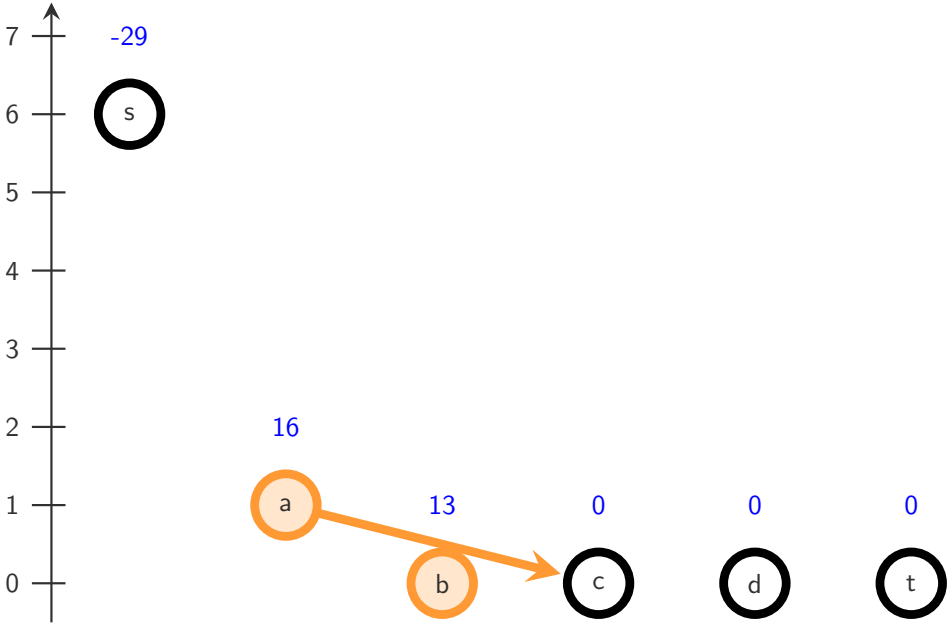
G, c and f

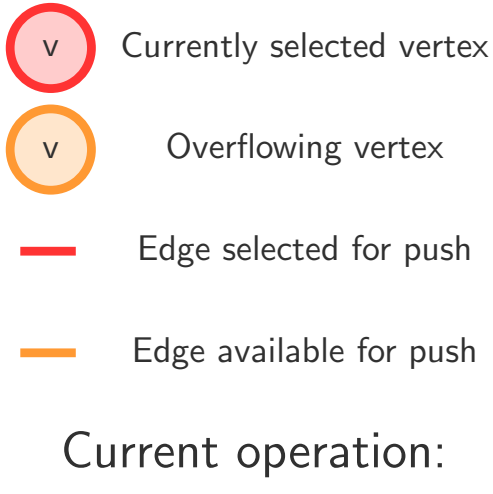


G_f and c_f

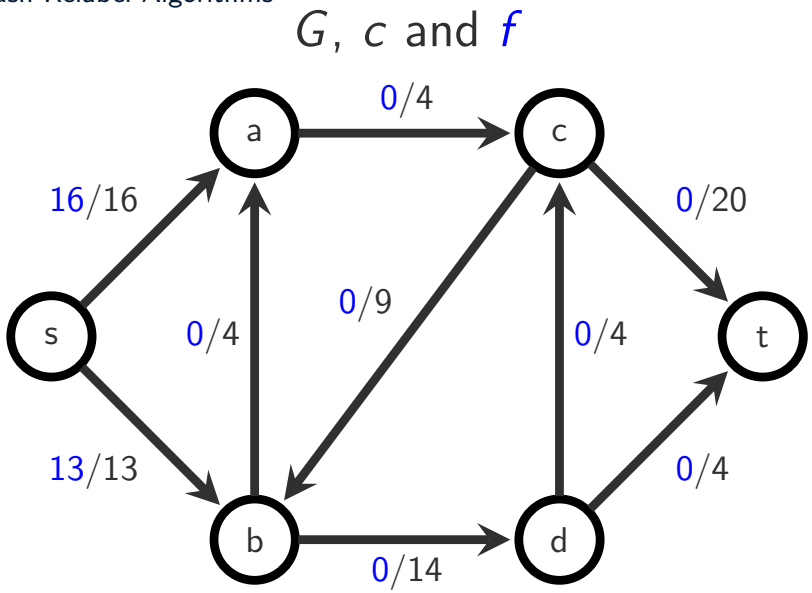


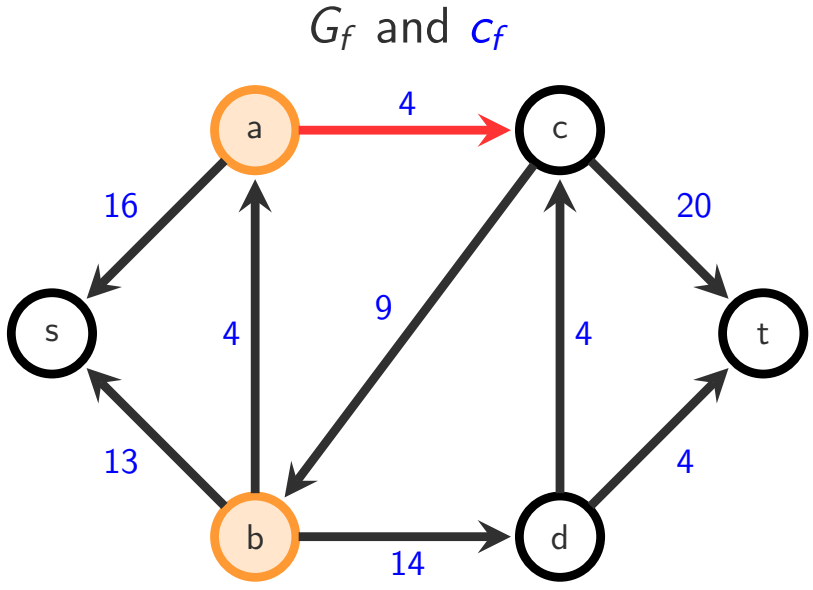
$h(v)$ and $e(v)$

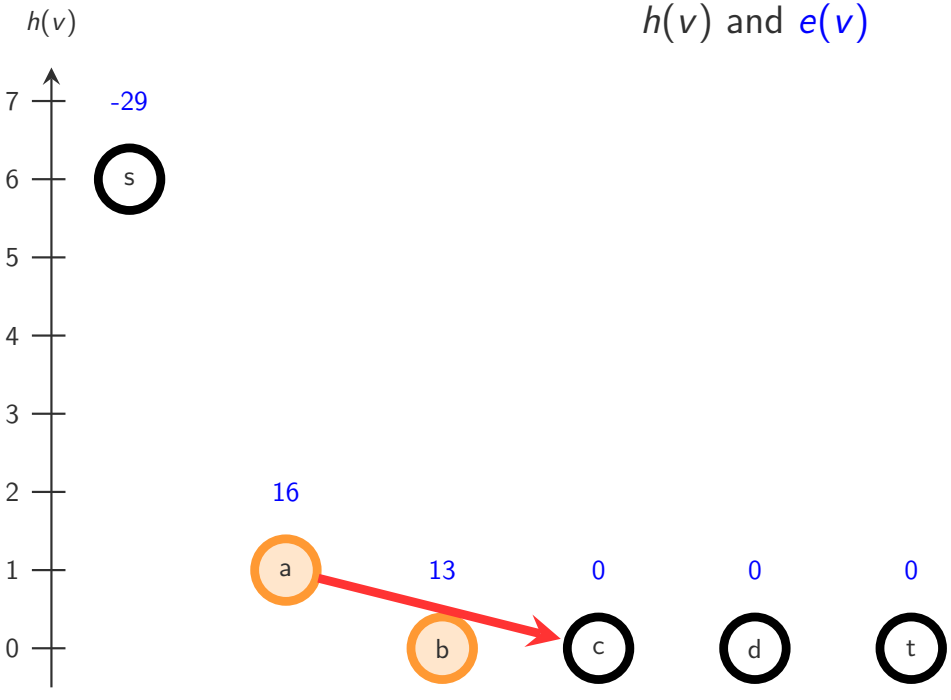


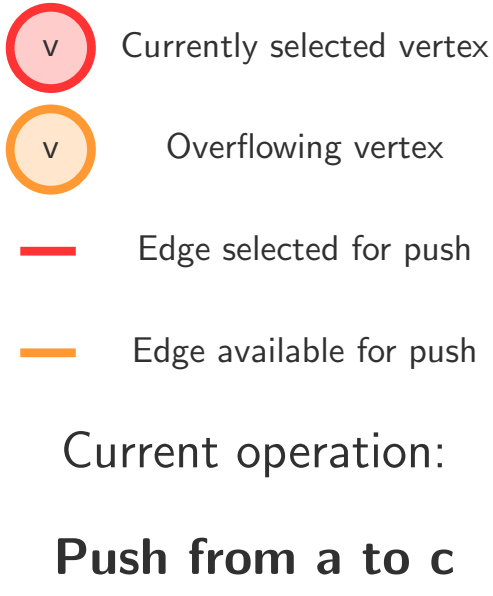


Current operation:

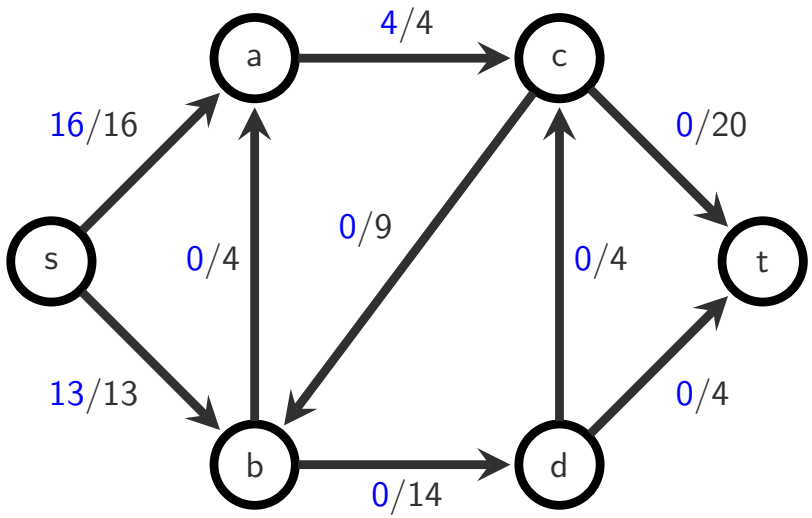

G, c and f
a
c
s
t
b
d
0/4
16/16
0/20
0/9
0/4
0/4
13/13
0/4
0/14


G_f and c_f
a
c
s
t
b
d
4
16
20
4
9
4
13
4
14

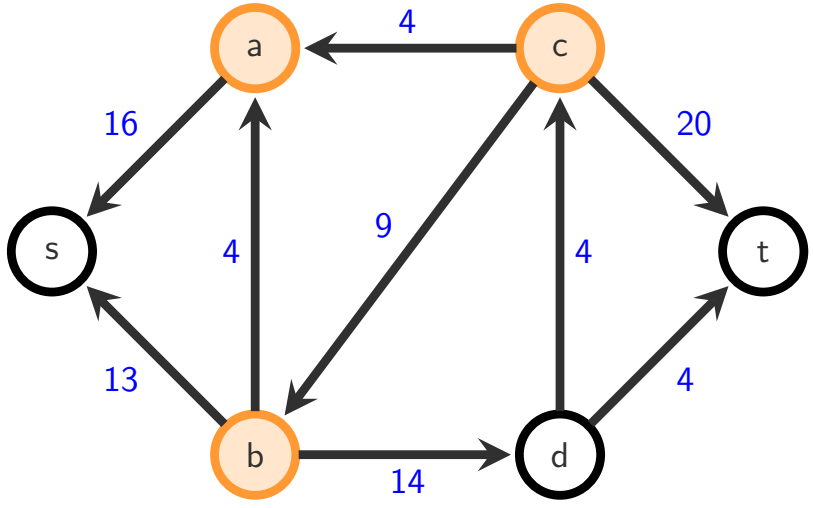

$h(v)$
$h(v)$ and $e(v)$
7
6
5
4
3
2
1
0
-29
s
16
a
13
b
0
c
0
d
0
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Push from a to c

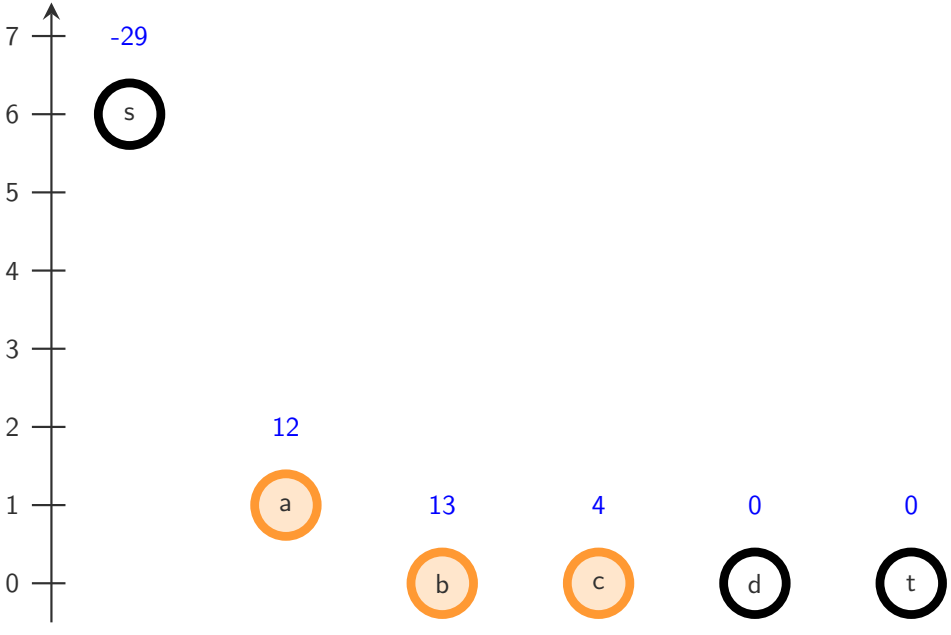
G, c and f

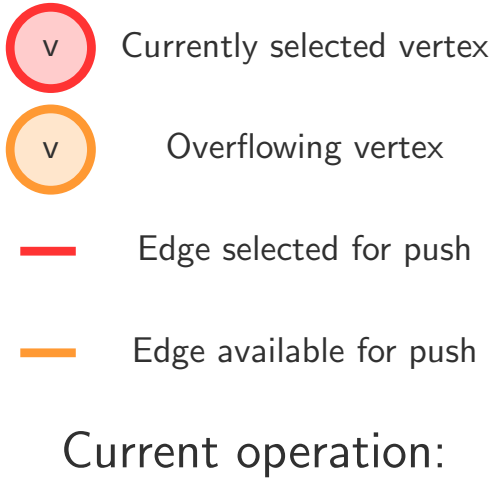


G_f and c_f

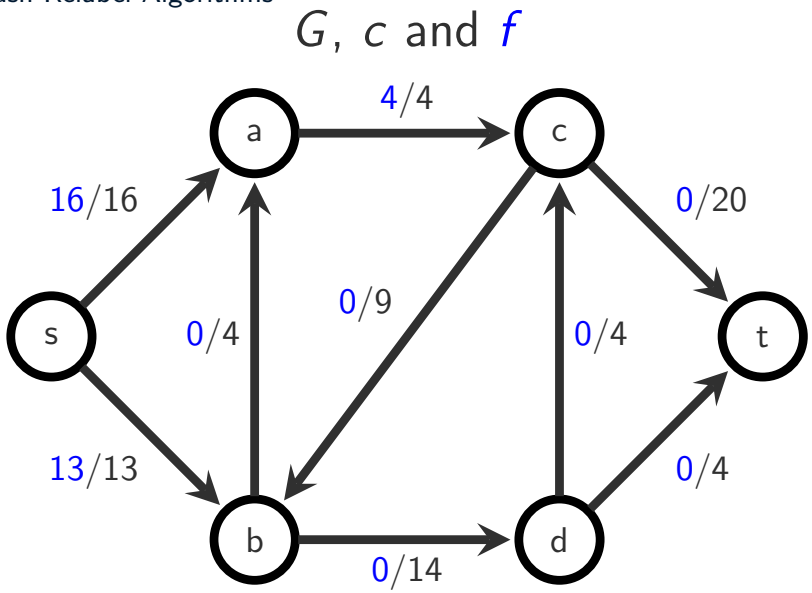


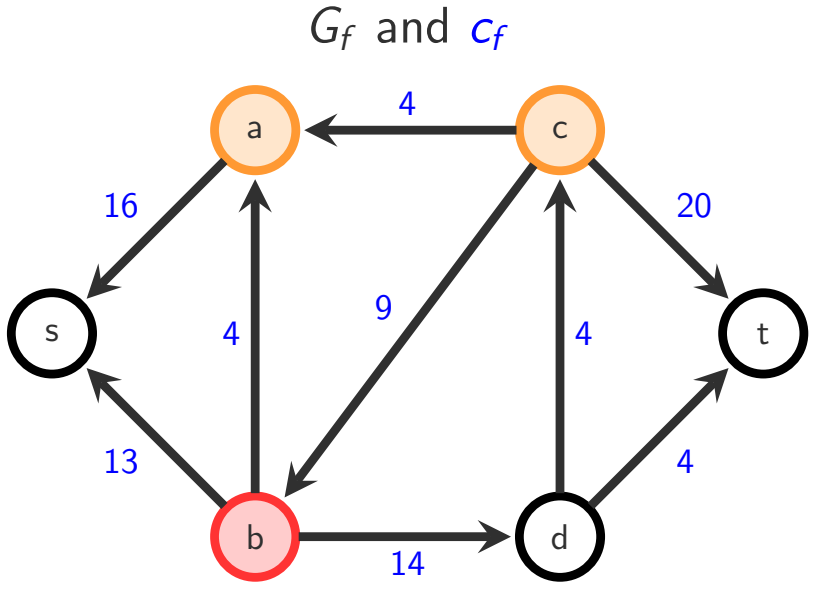
$h(v)$ and $e(v)$

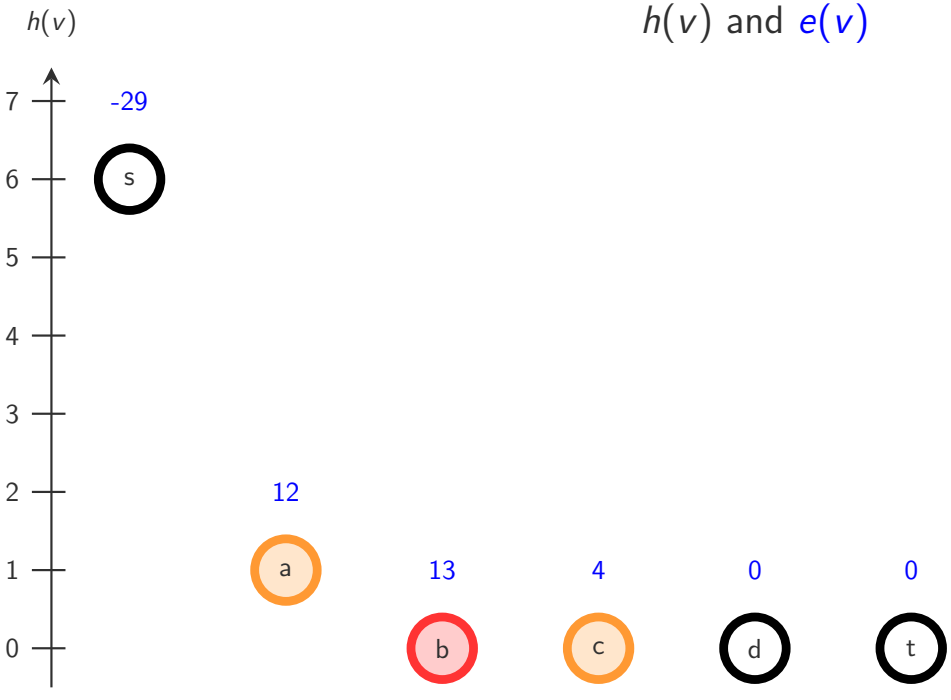


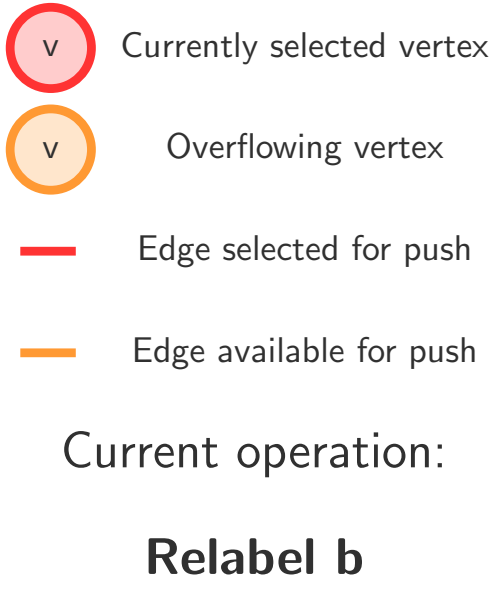


Current operation:

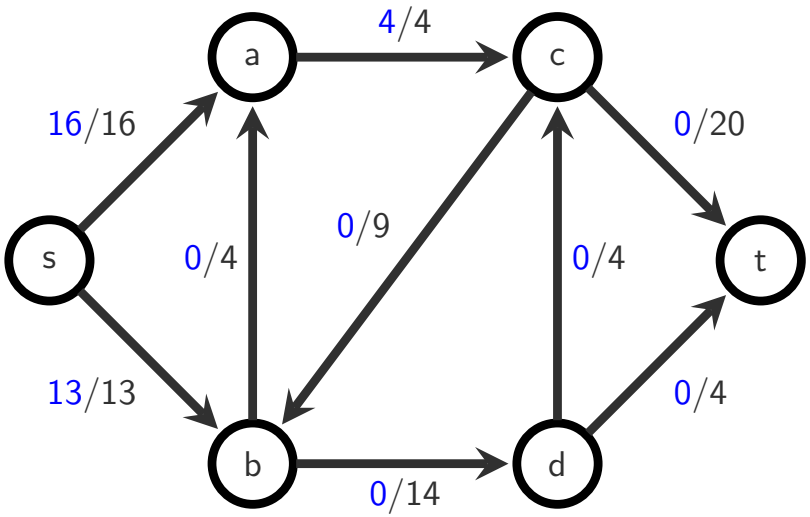

G, c and f
a
c
s
t
b
d
4/4
16/16
0/20
0/9
0/4
0/4
13/13
0/4
0/14


G_f and c_f
a
c
s
t
b
d
4
16
20
9
4
4
13
4
14

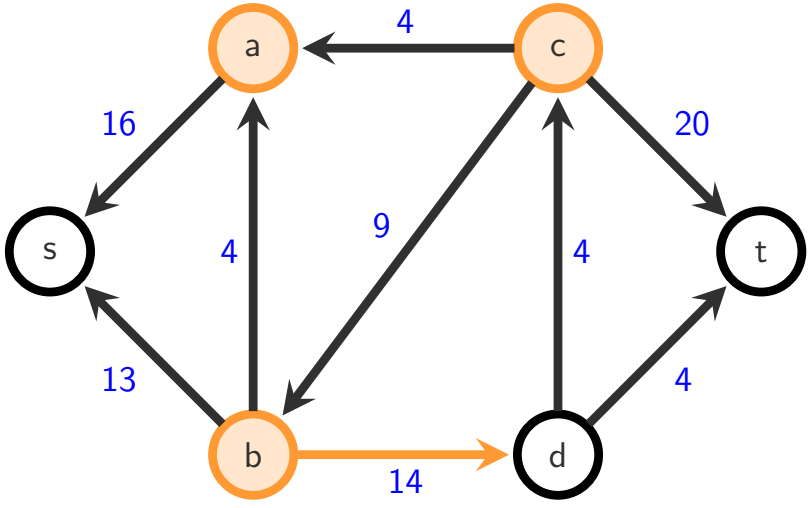

$h(v)$
$h(v)$ and $e(v)$
7
6
5
4
3
2
1
0
-29
s
12
a
13
b
4
c
0
d
0
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Relabel b

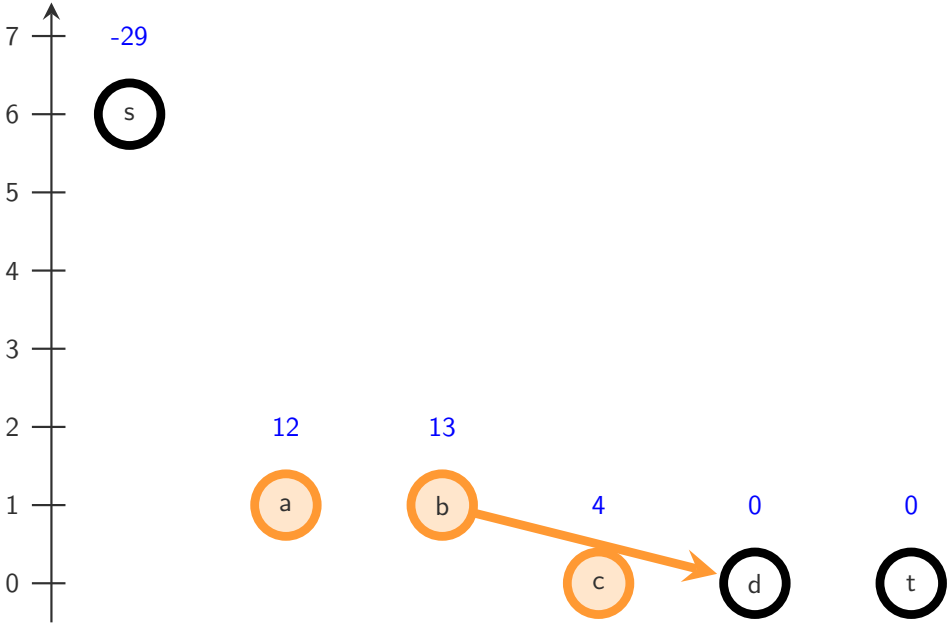
G, c and f

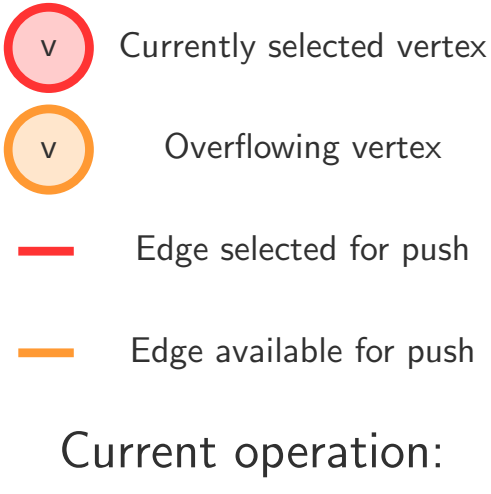


G_f and c_f

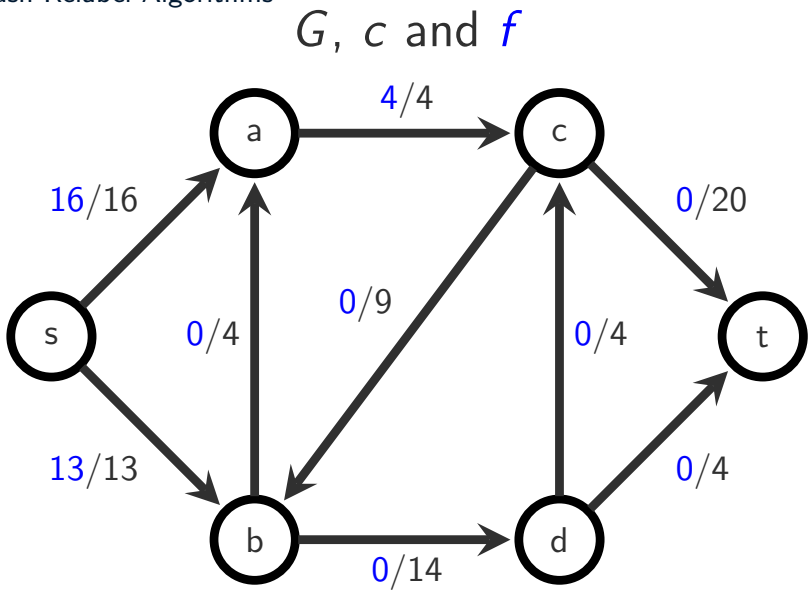


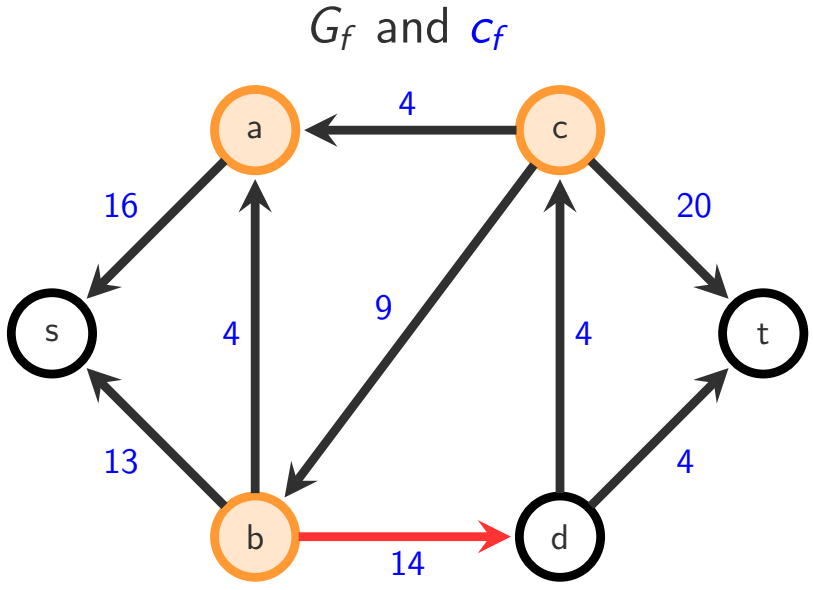
$h(v)$ and $e(v)$

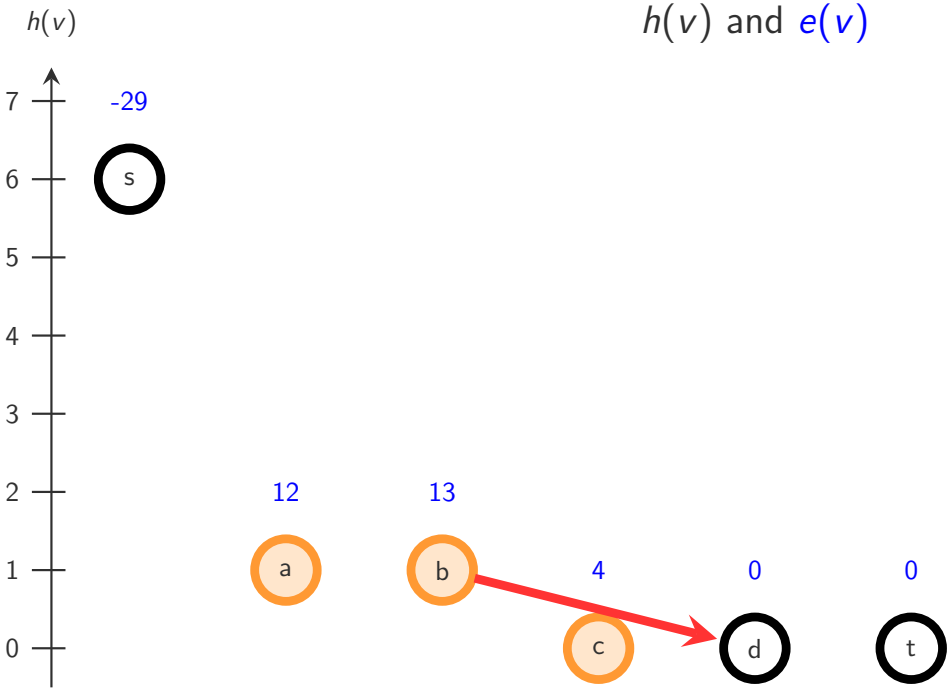




Current operation:







v Currently selected vertex

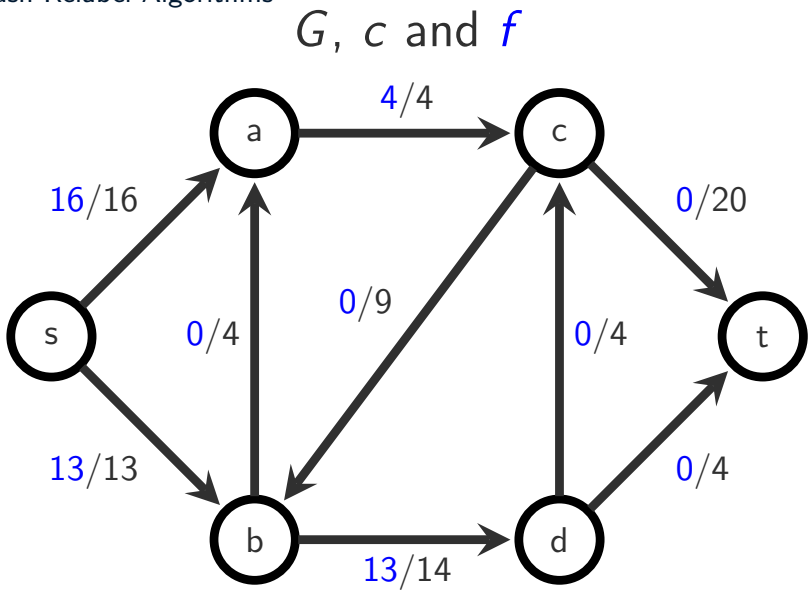
v Overflowing vertex

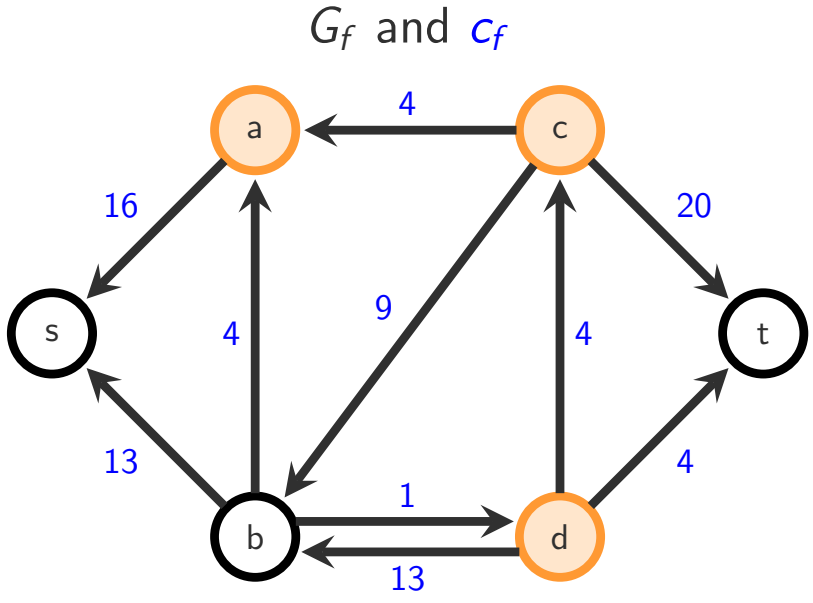
Edge selected for push

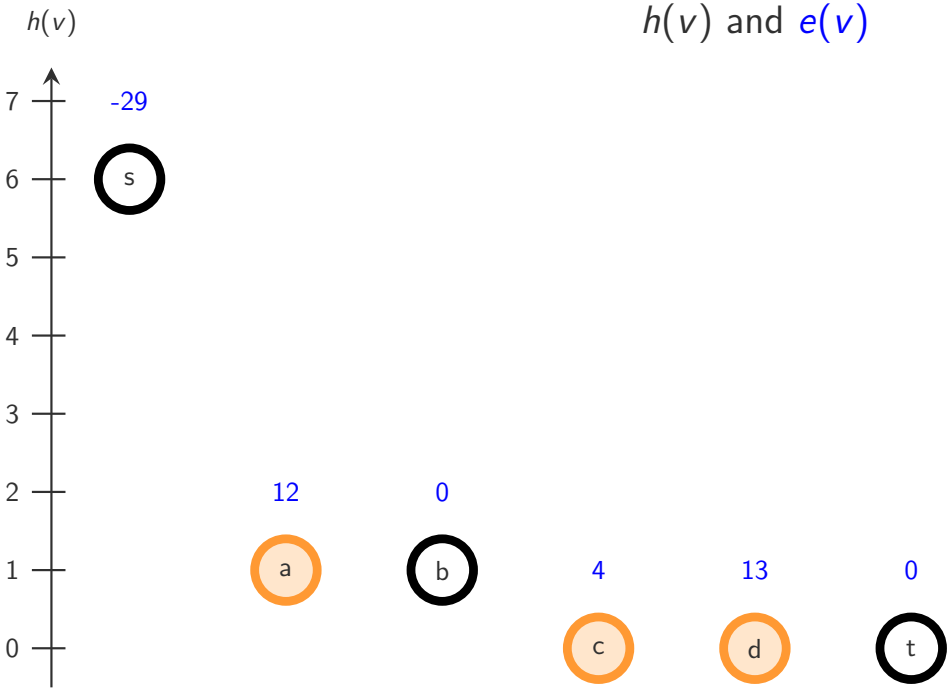
Edge available for push

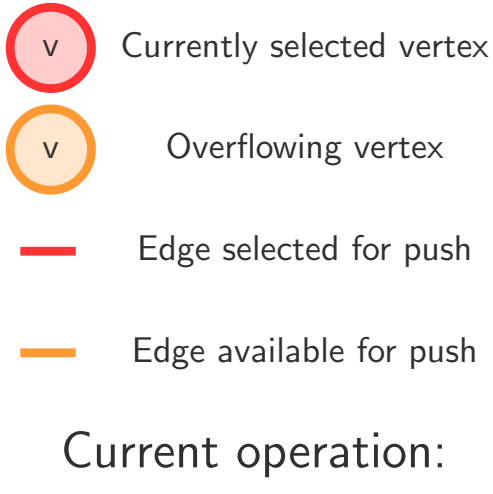
Current operation:

Push from b to d



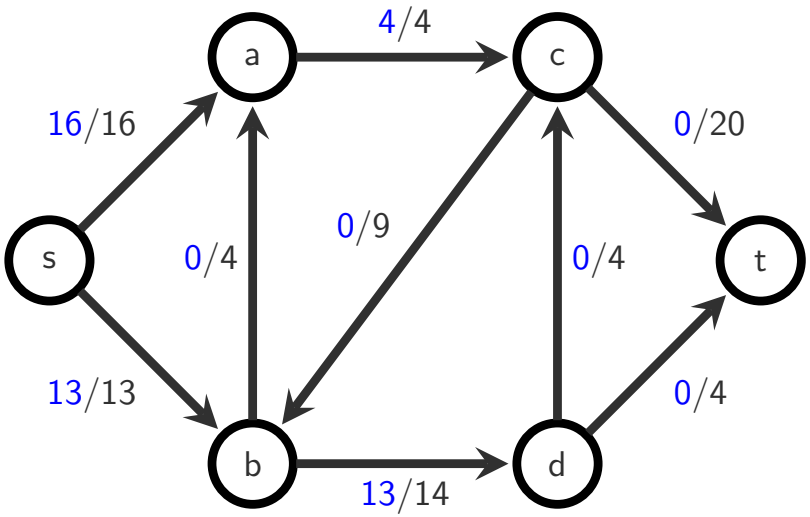




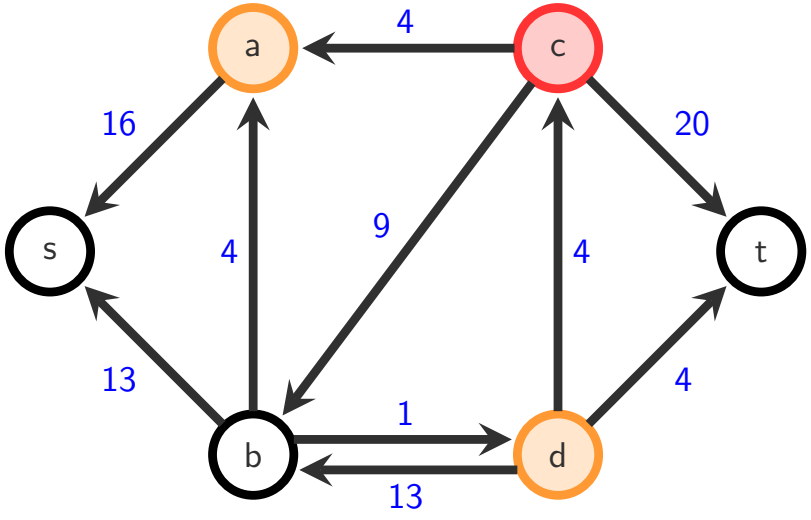


Current operation:

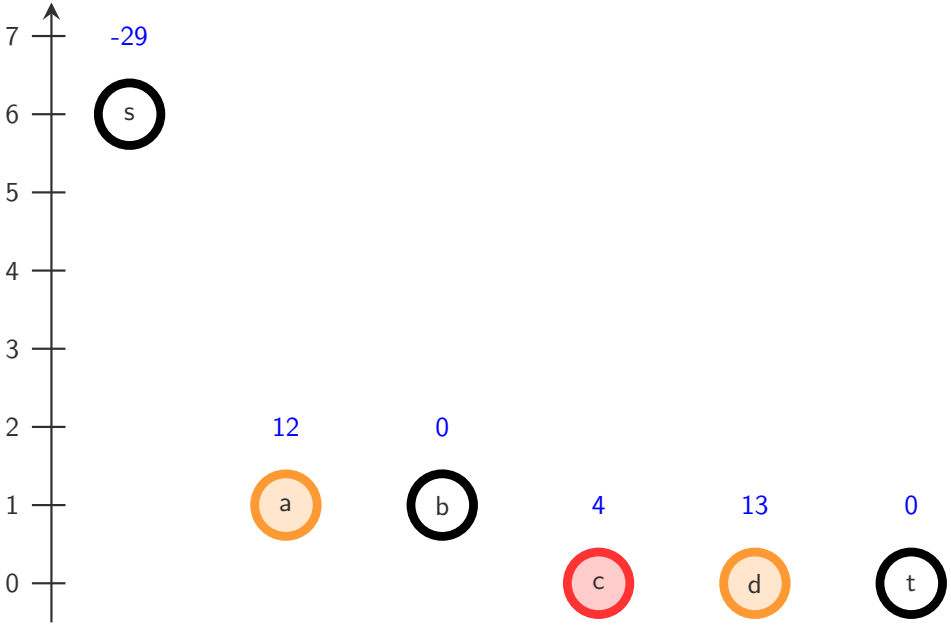
G, c and f



G_f and c_f



$h(v)$ and $e(v)$



v Currently selected vertex

v Overflowing vertex

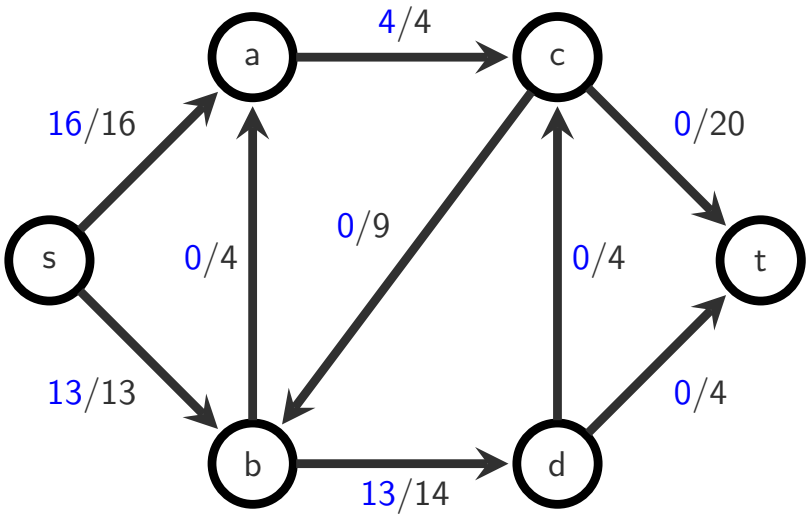
Edge selected for push

Edge available for push

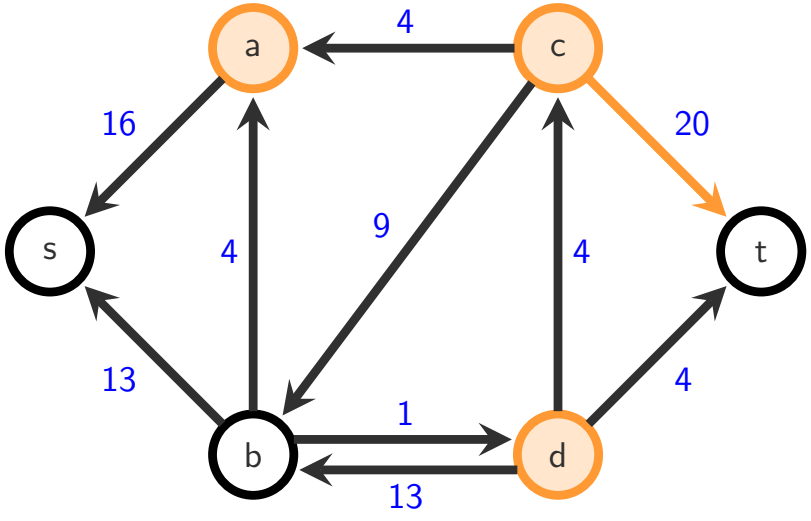
Current operation:

Relabel c

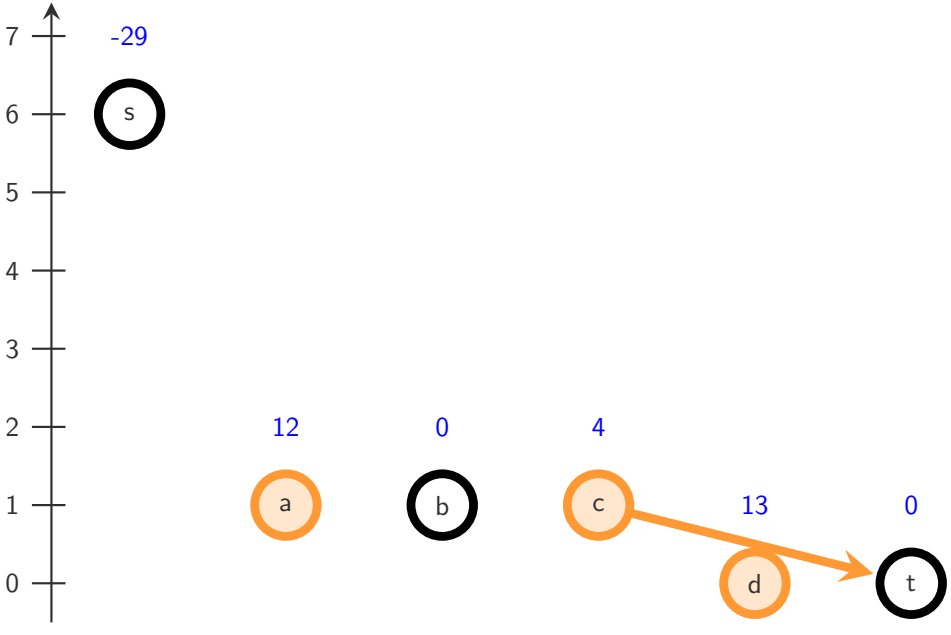
G, c and f

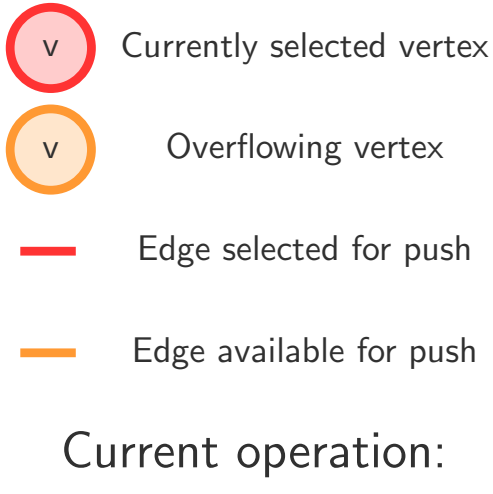


G_f and c_f

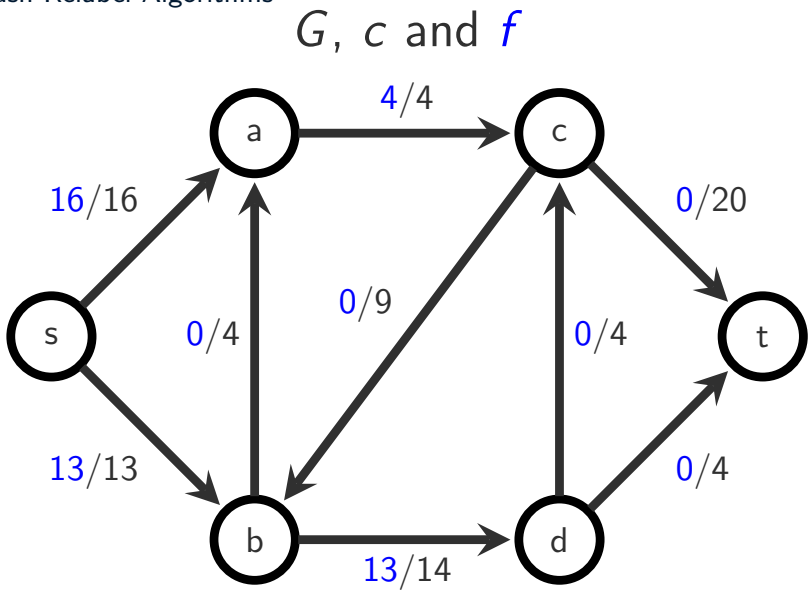


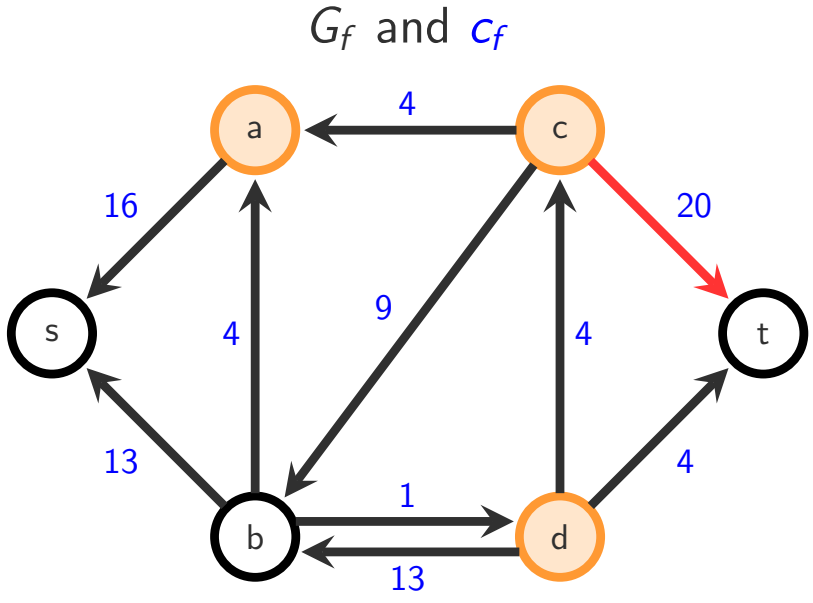
$h(v)$ and $e(v)$

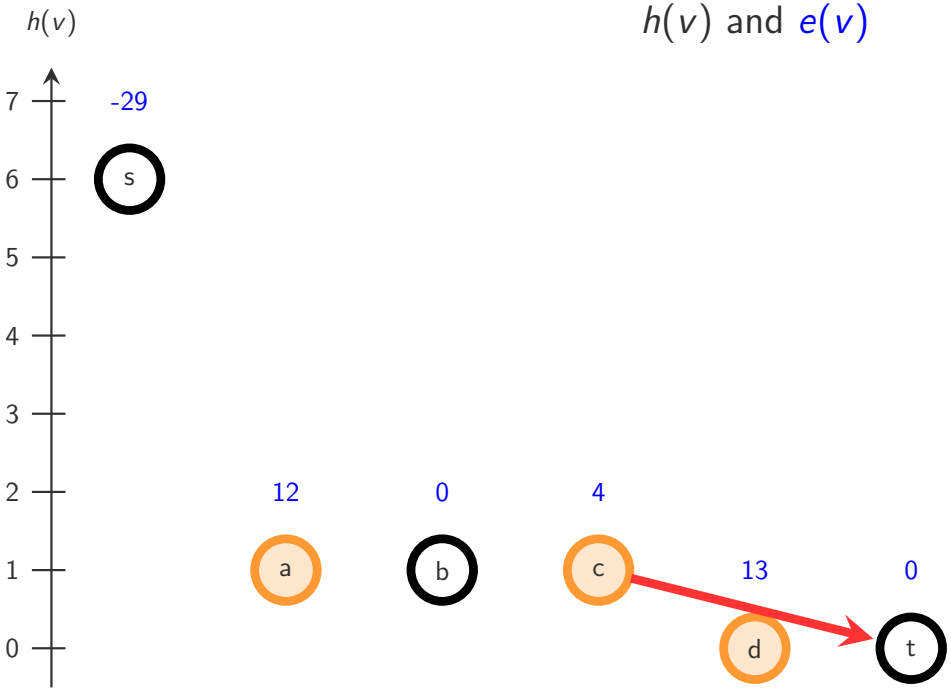




Current operation:







v Currently selected vertex

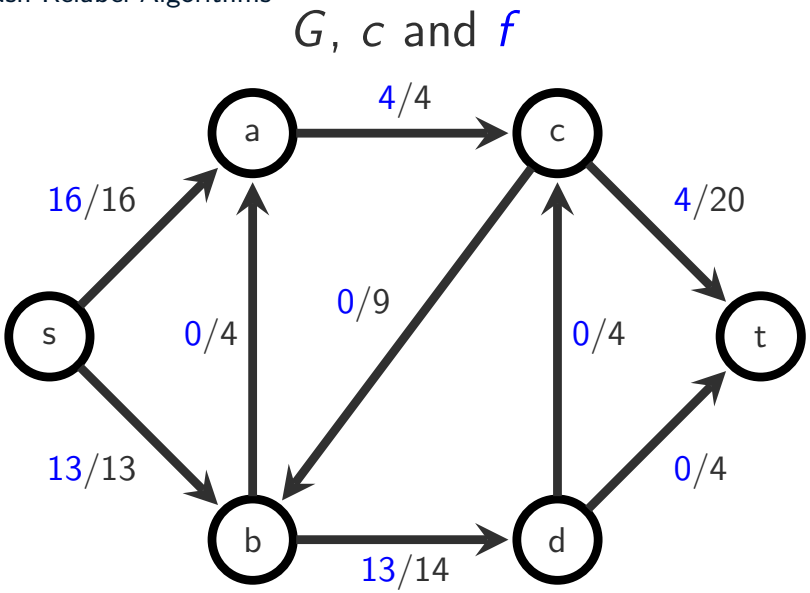
v Overflowing vertex

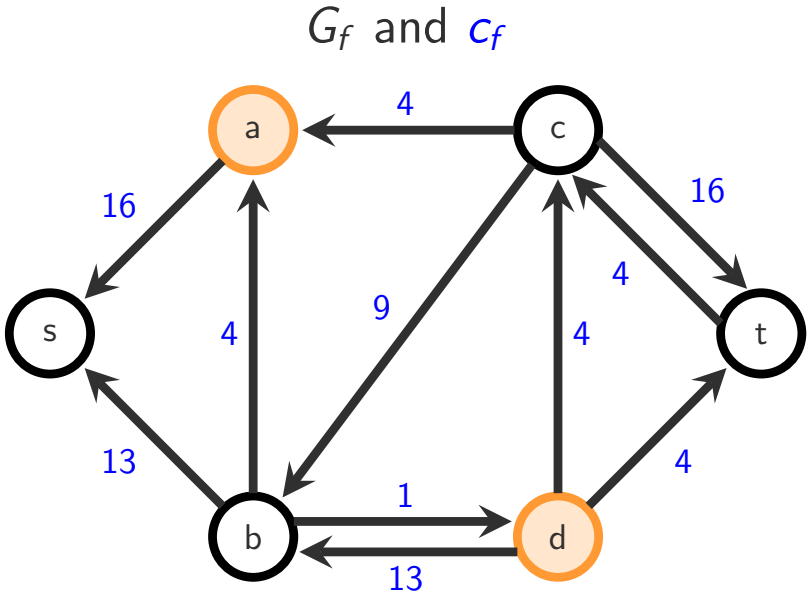
Edge selected for push

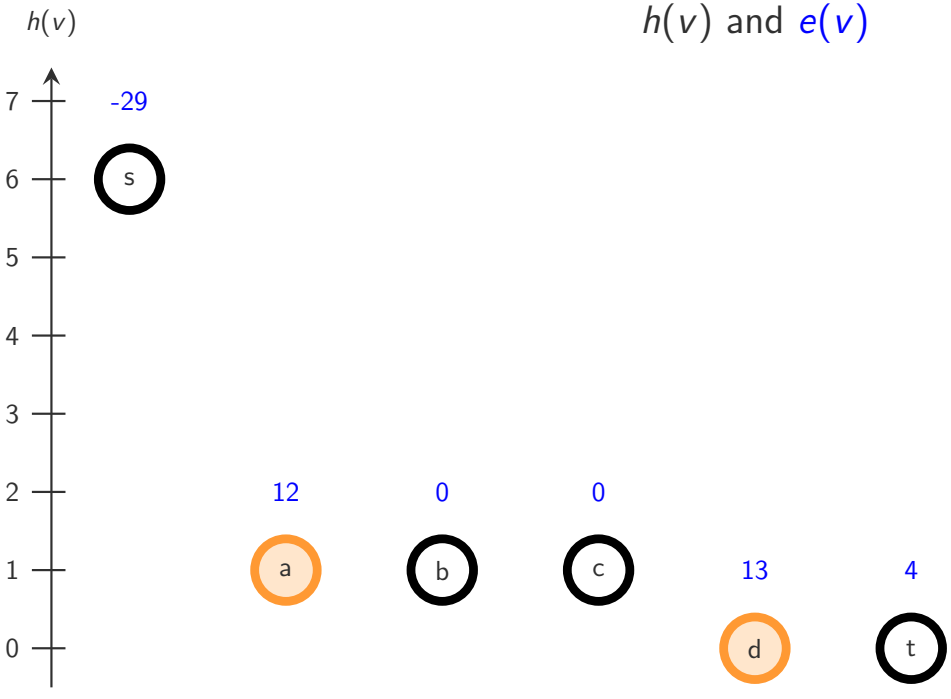
Edge available for push

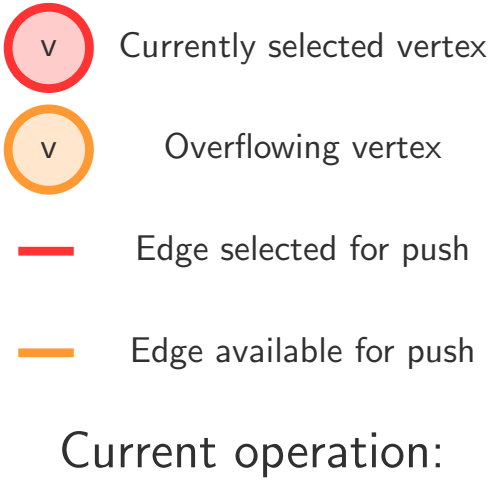
Current operation:

Push from c to t



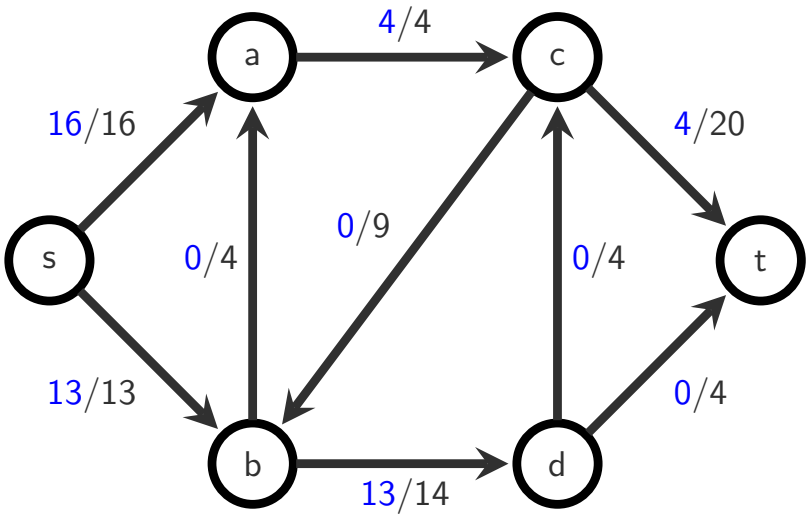




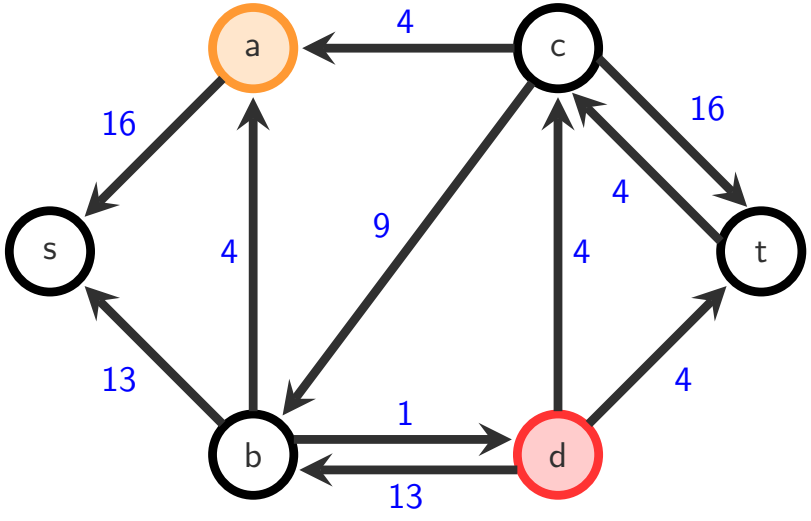


Current operation:

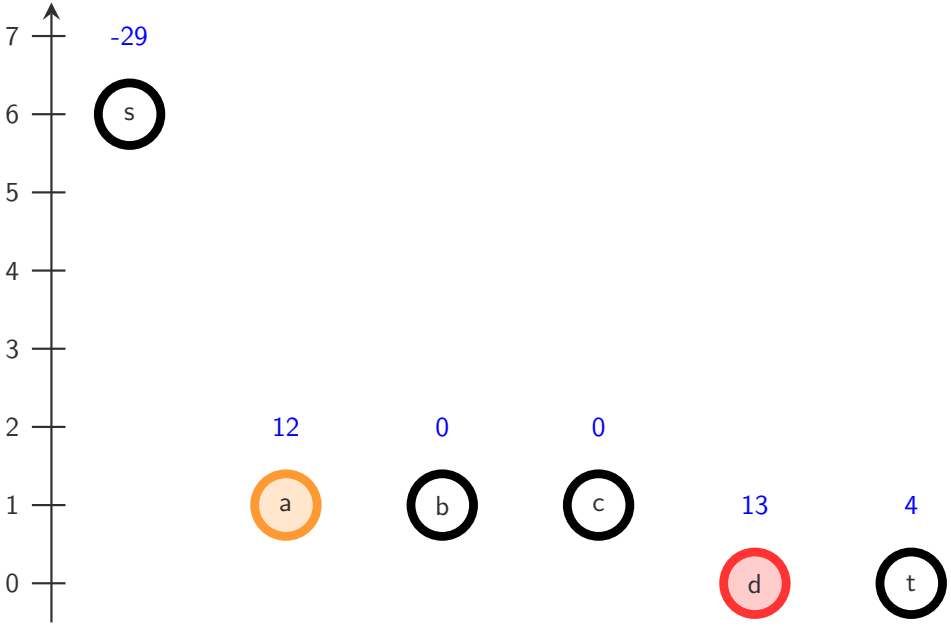
G, c and f

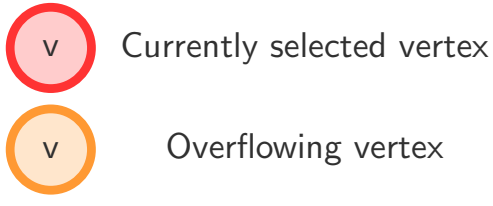


G_f and c_f



$h(v)$ and $e(v)$

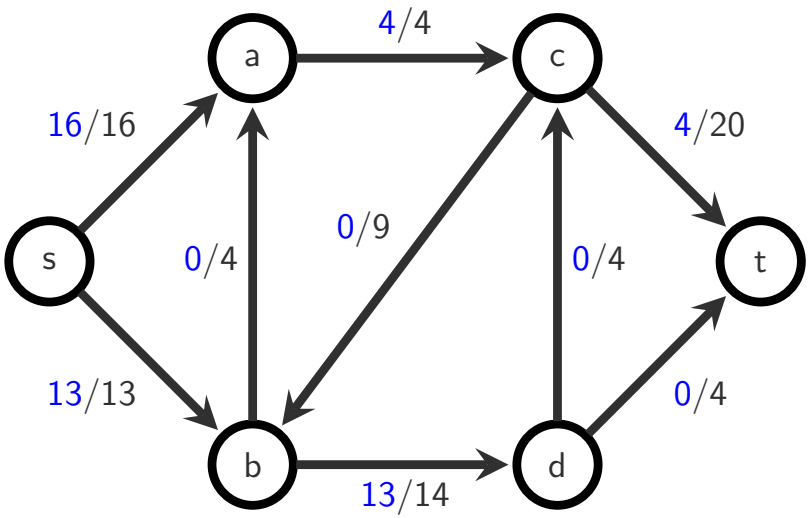




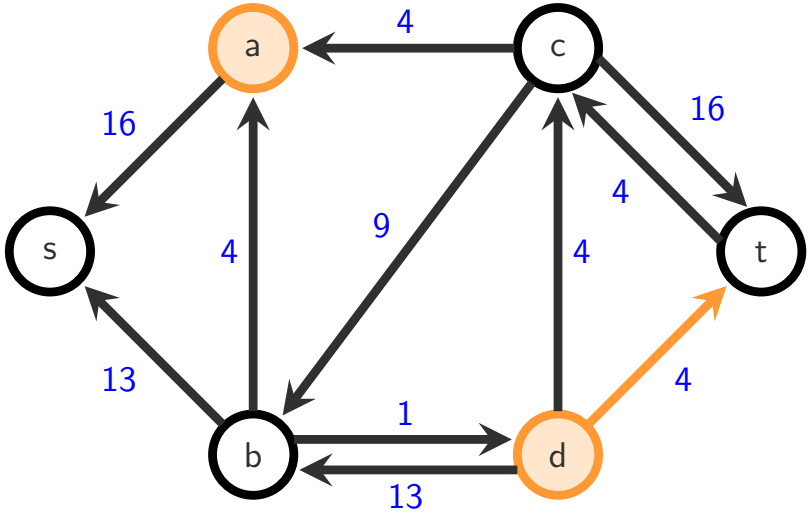
Current operation:

Relabel d

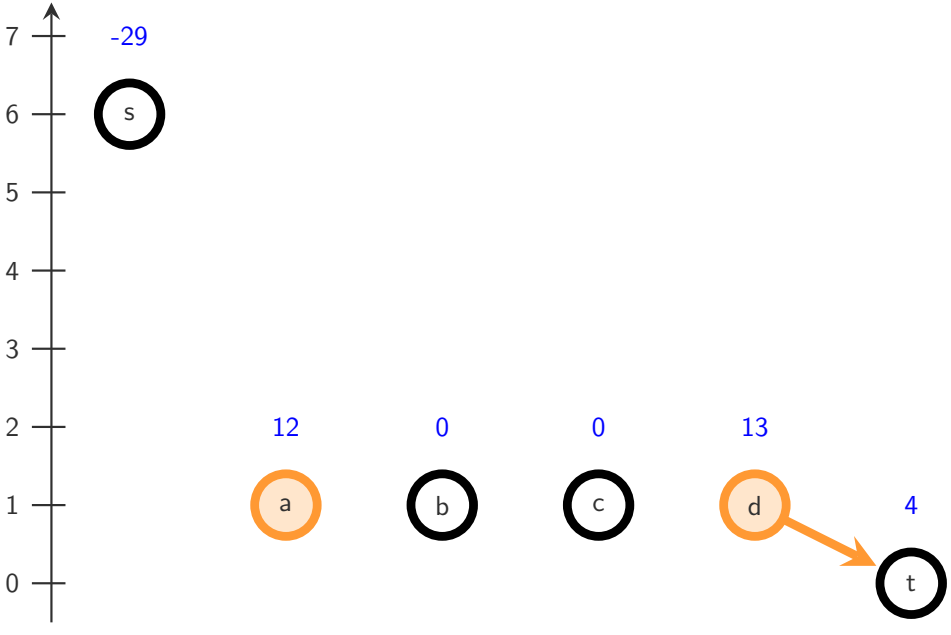
G, c and f

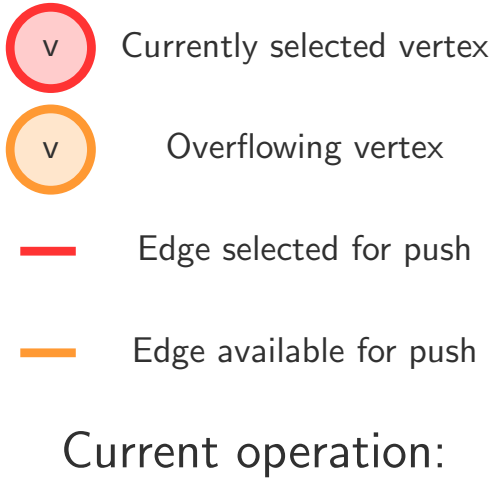


G_f and c_f



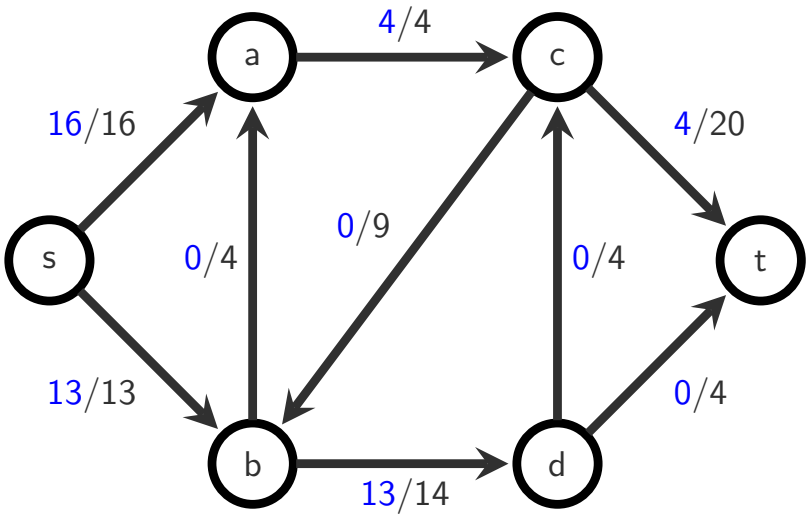
$h(v)$ and $e(v)$



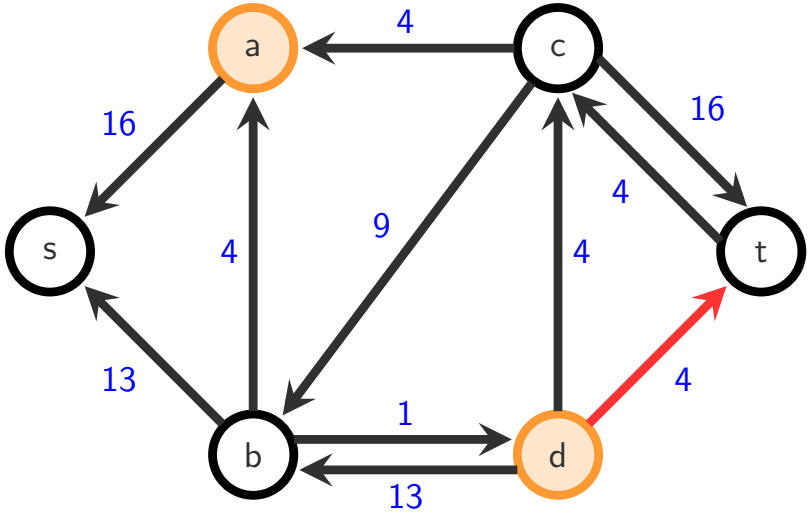


Current operation:

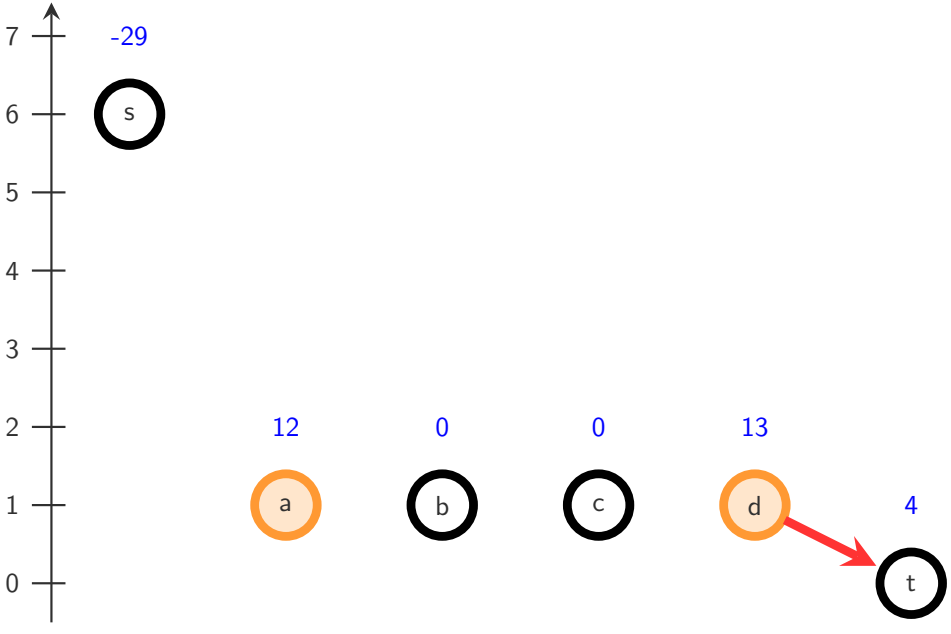
G, c and f

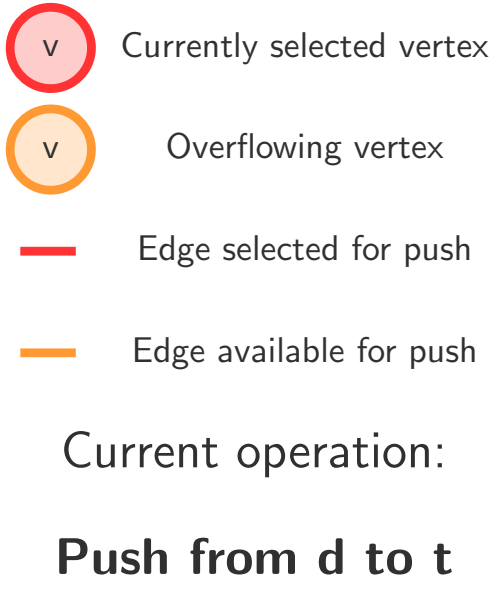


G_f and c_f

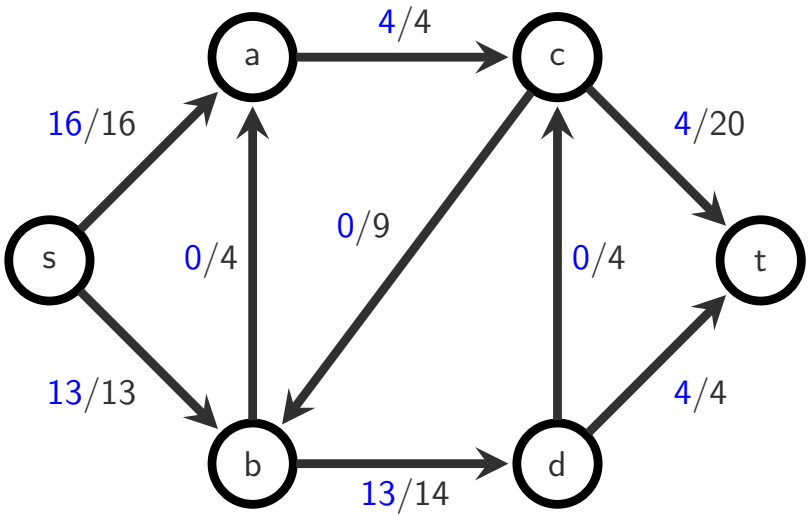


$h(v)$ and $e(v)$

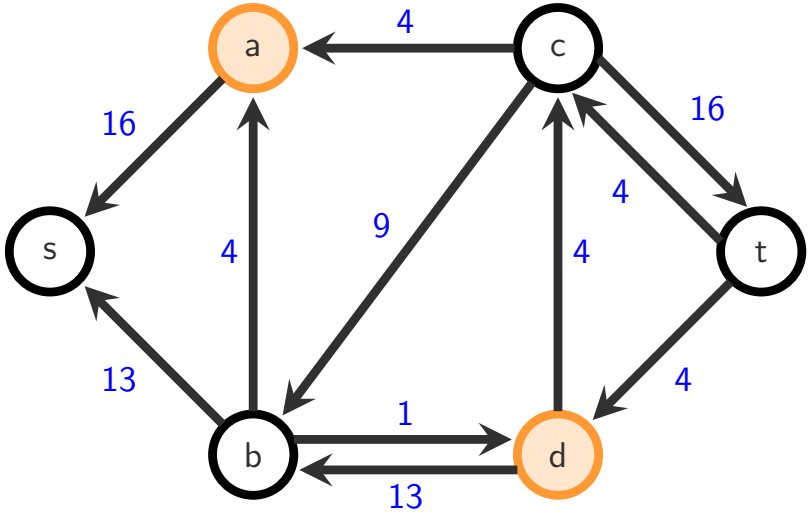




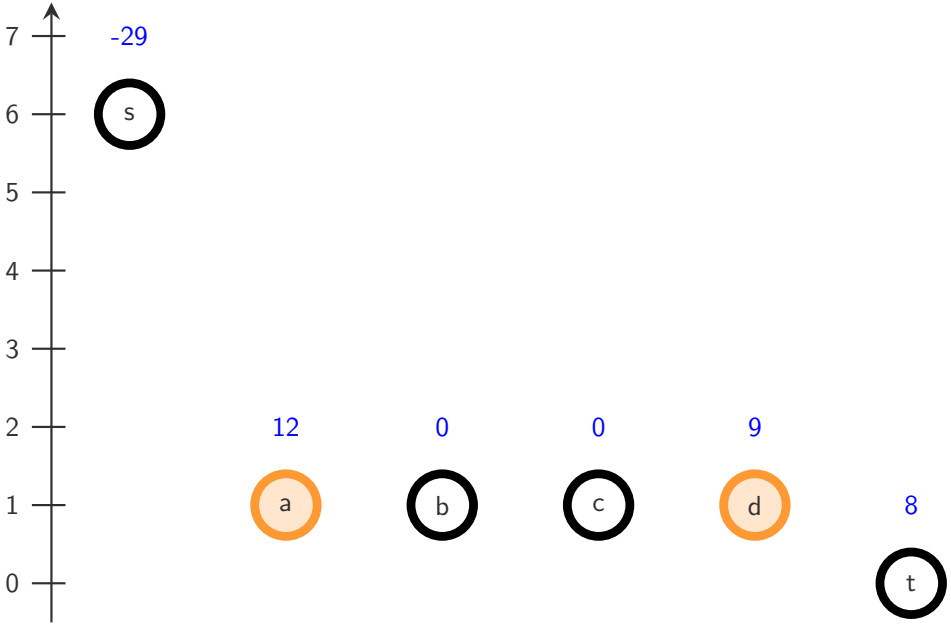
G, c and f

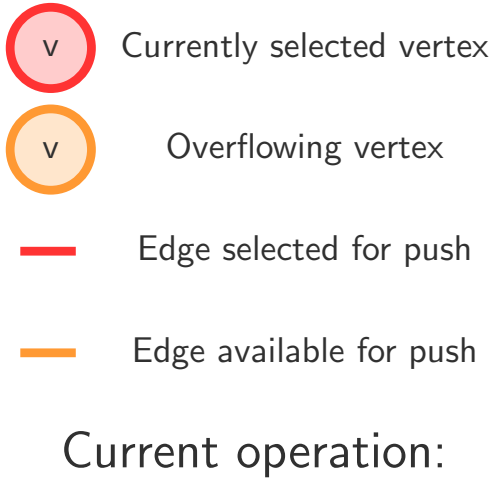


G_f and c_f



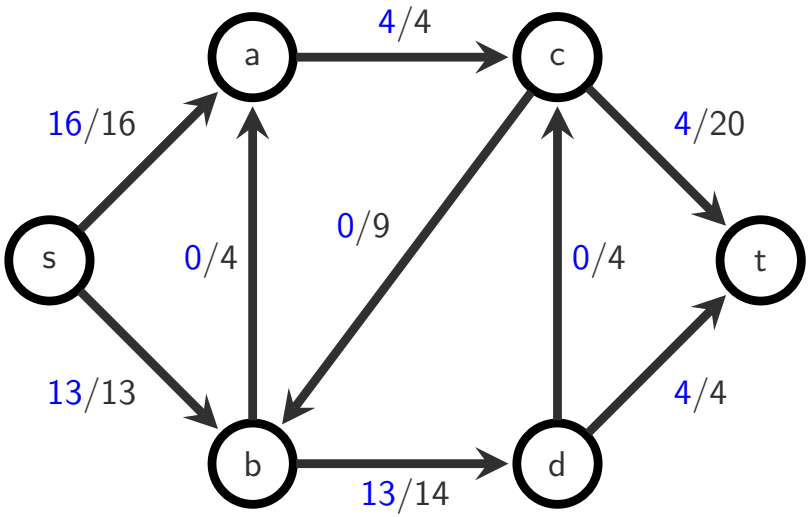
$h(v)$ and $e(v)$



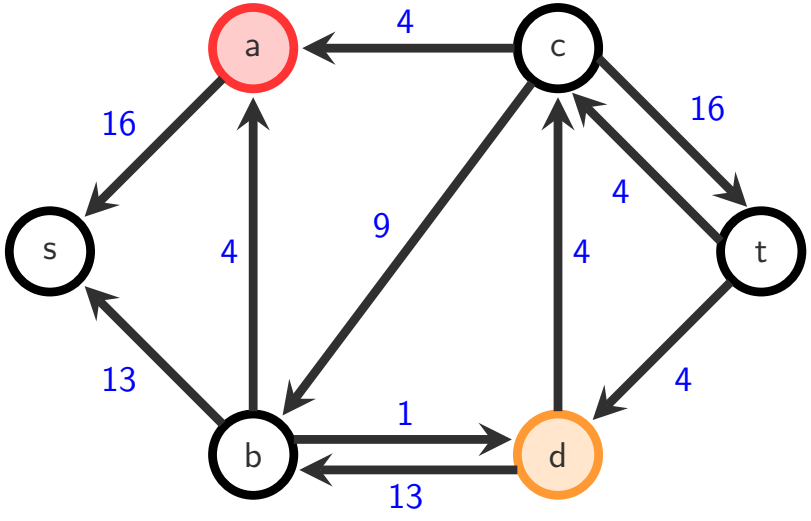


Current operation:

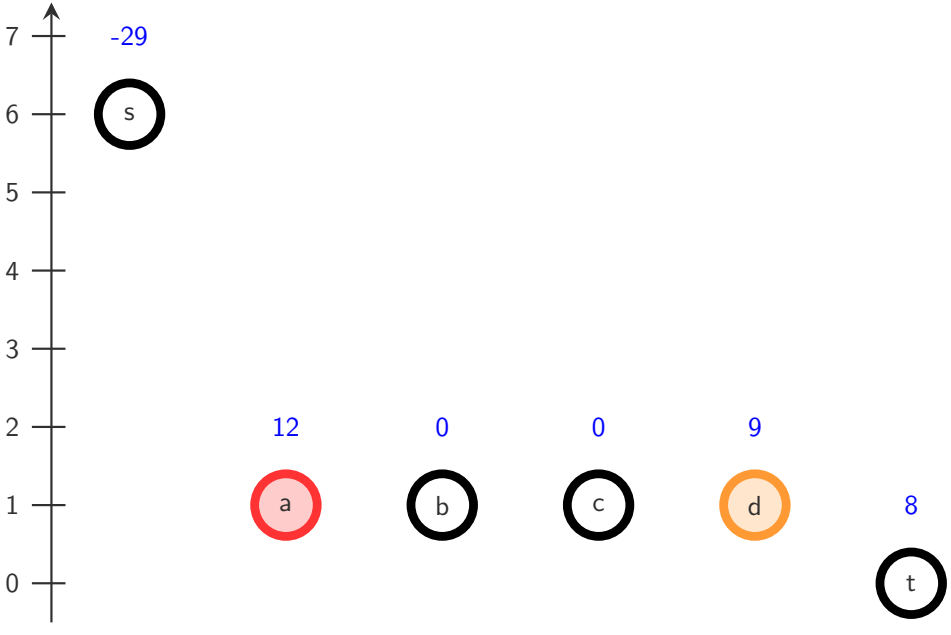
G, c and f

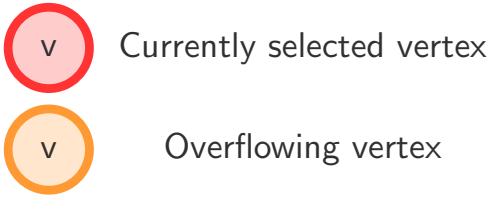


G_f and c_f



$h(v)$ and $e(v)$

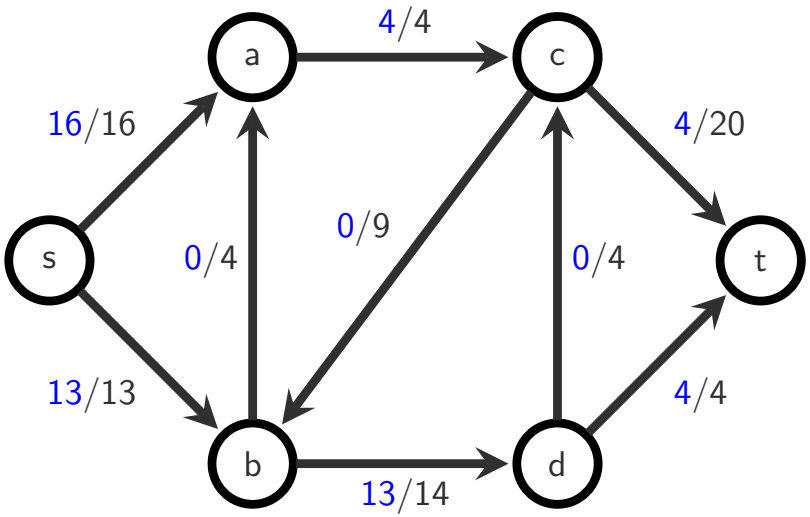




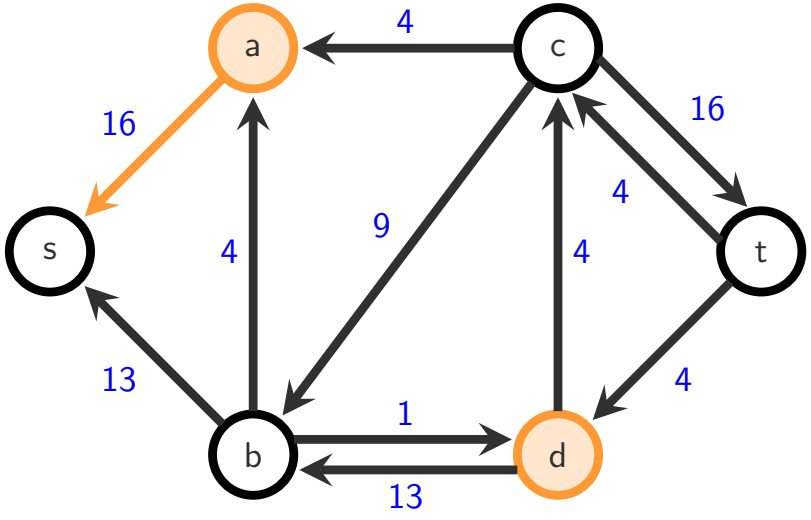
Current operation:

Relabel a

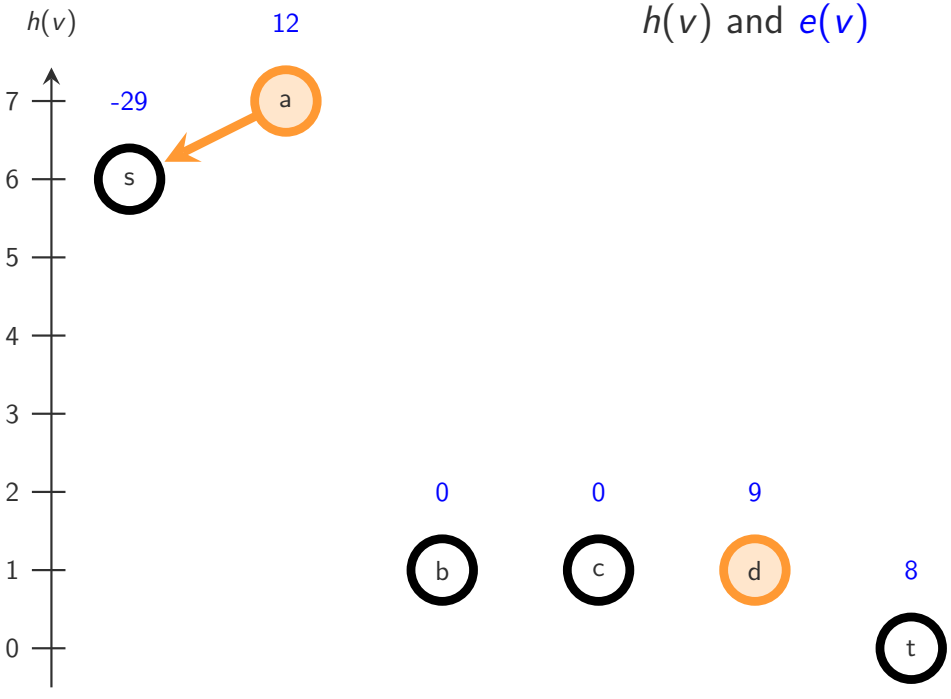
G, c and f

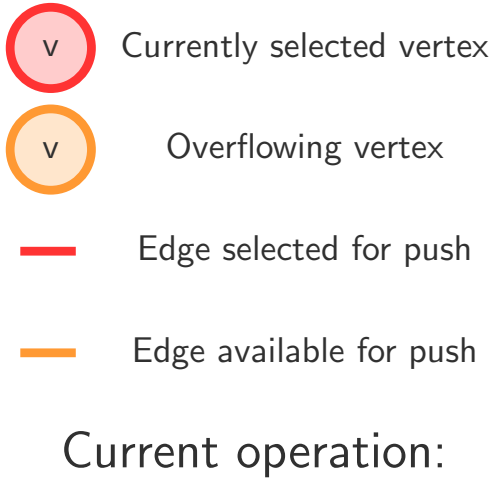


G_f and c_f



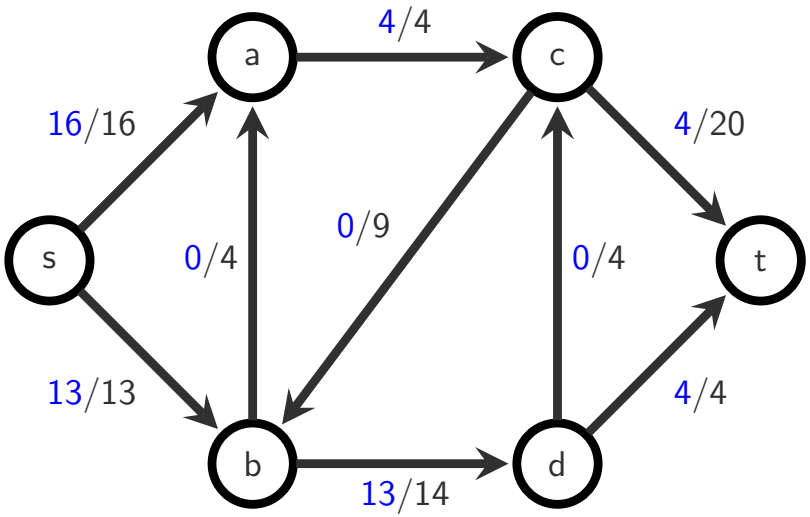
$h(v)$ and $e(v)$



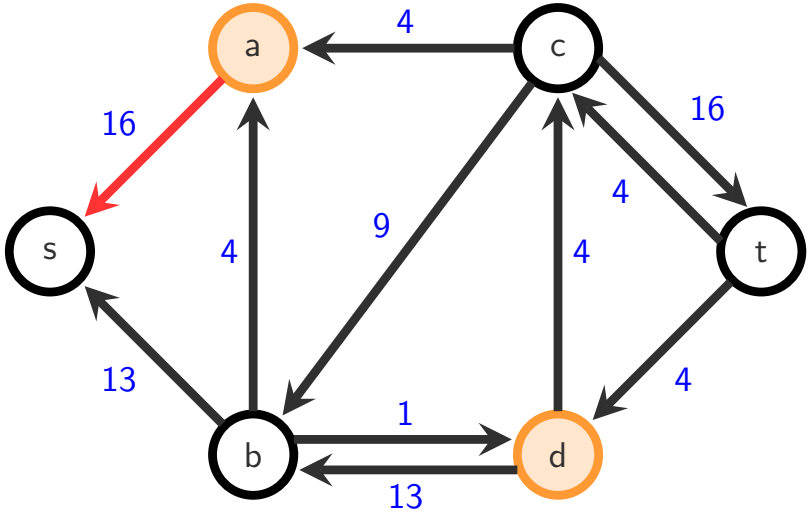


Current operation:

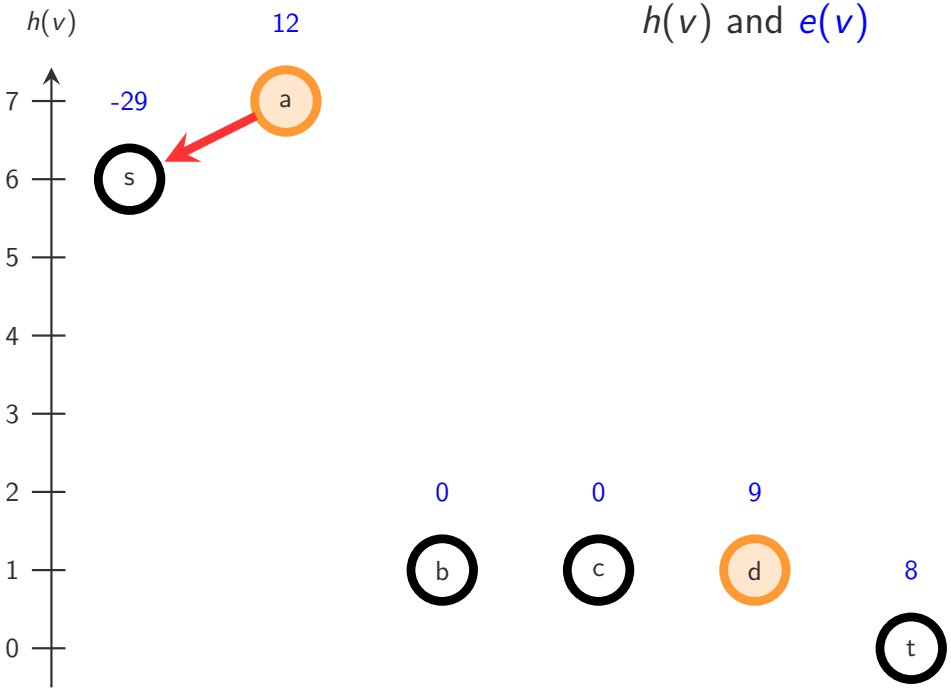
G, c and f



G_f and c_f



$h(v)$ and $e(v)$



v Currently selected vertex

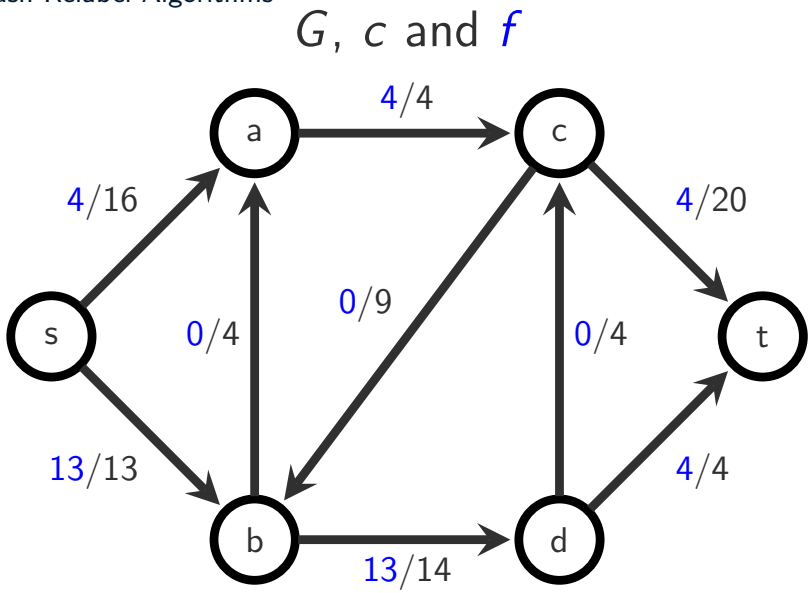
v Overflowing vertex

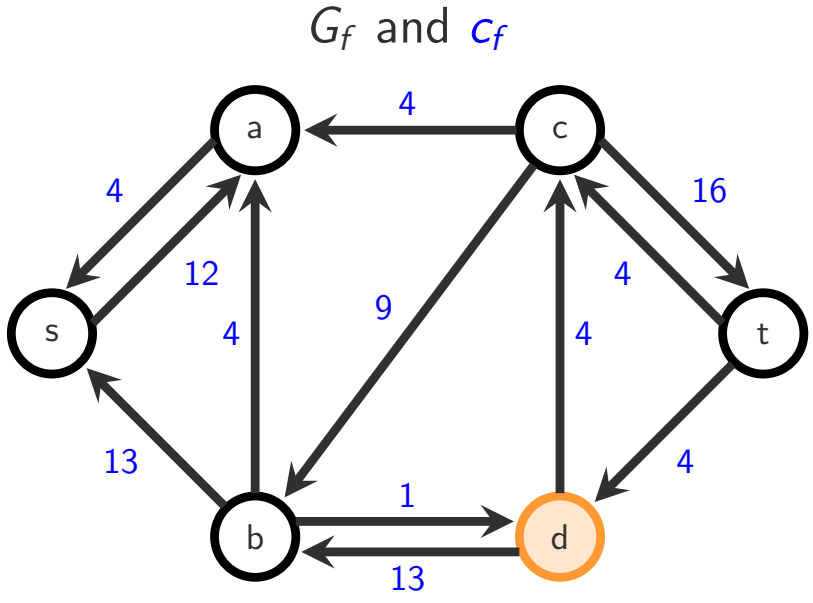
Edge selected for push

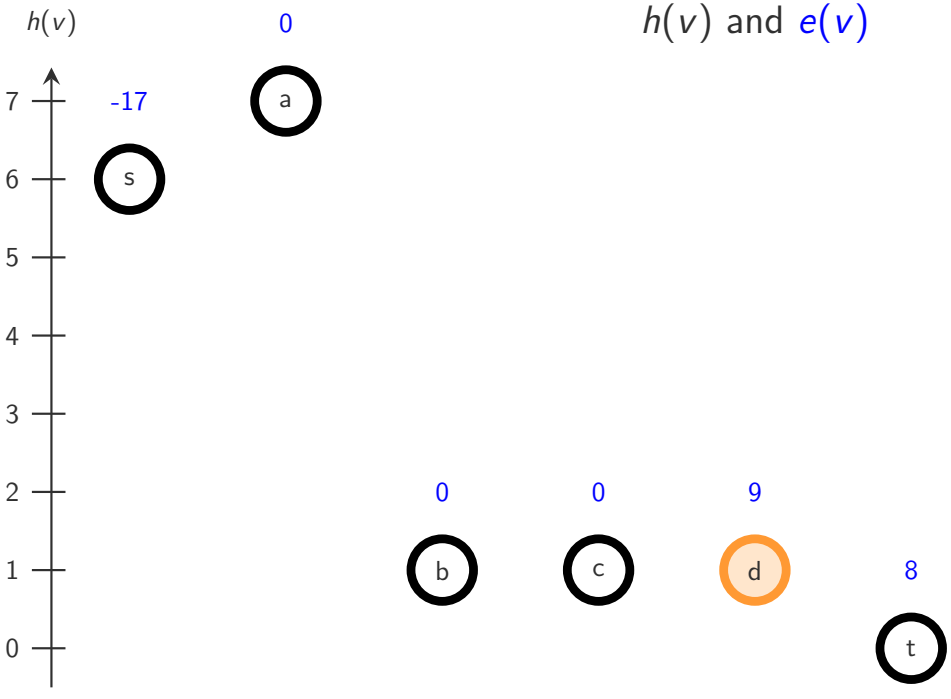
Edge available for push

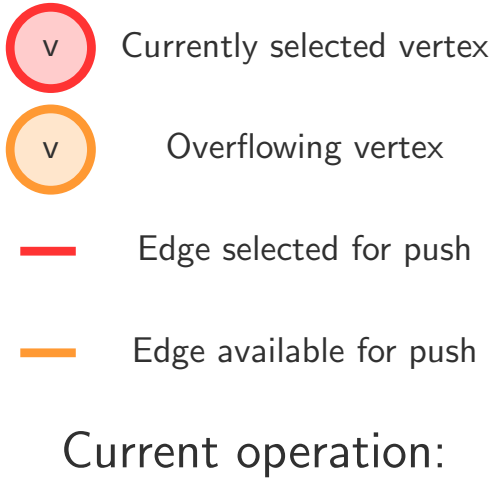
Current operation:

Push from a to s

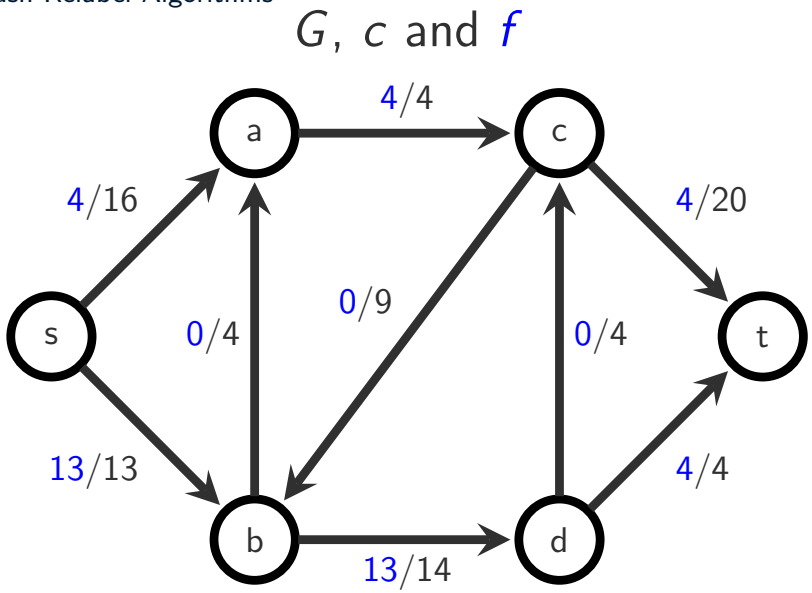


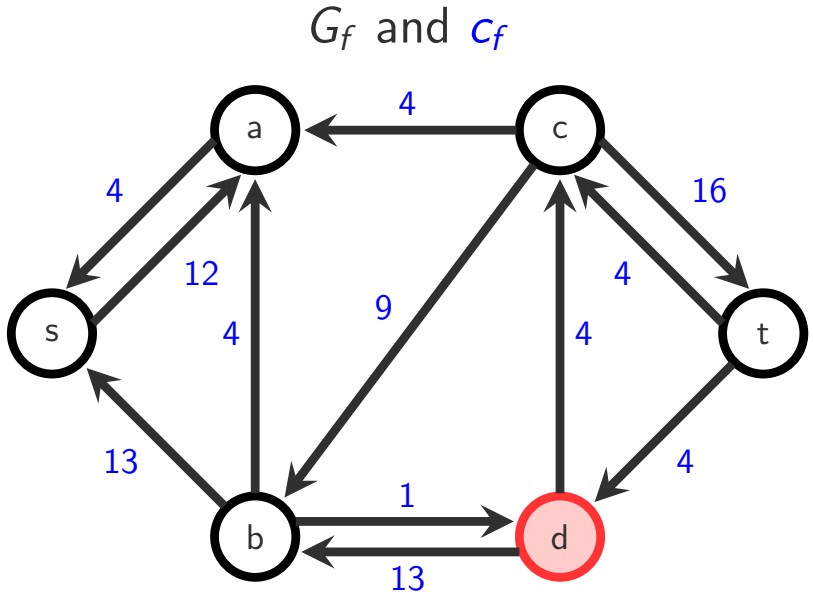


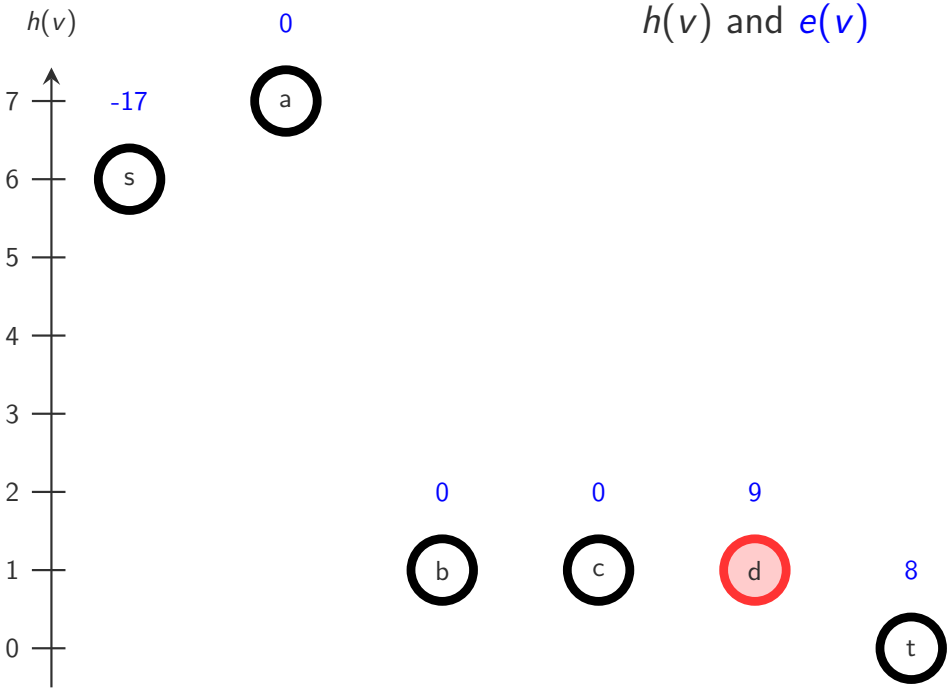


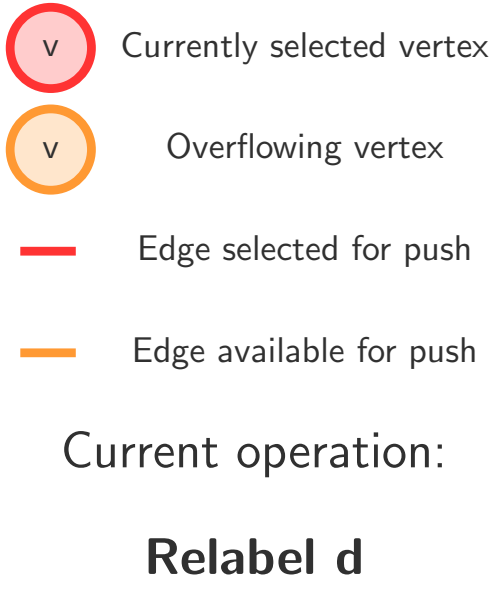


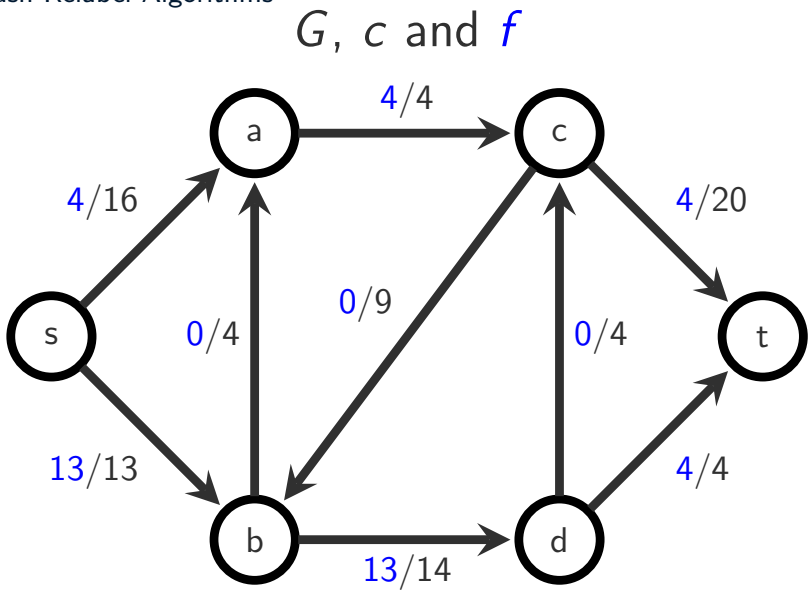
Current operation:

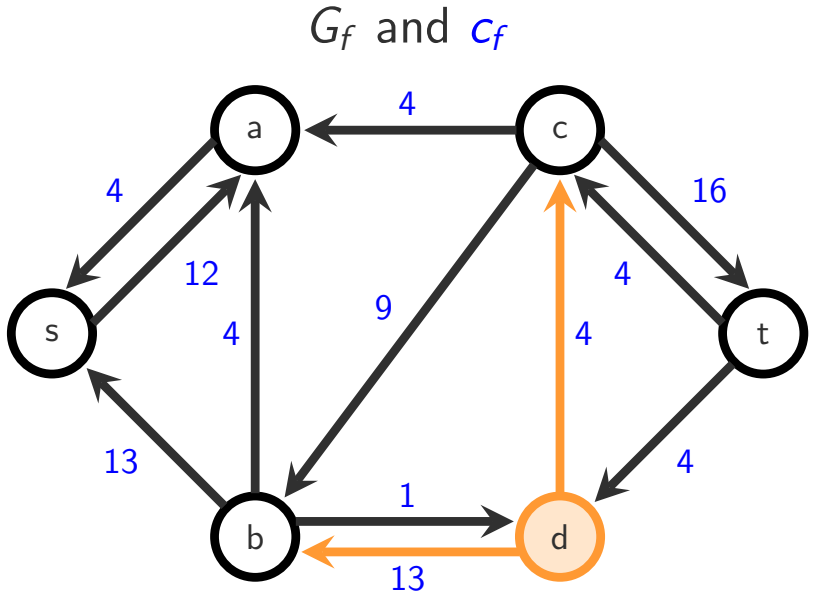

G, c and f
a
c
s
b
d
t
4/4
4/16
4/20
0/4
0/9
0/4
13/13
4/4
13/14

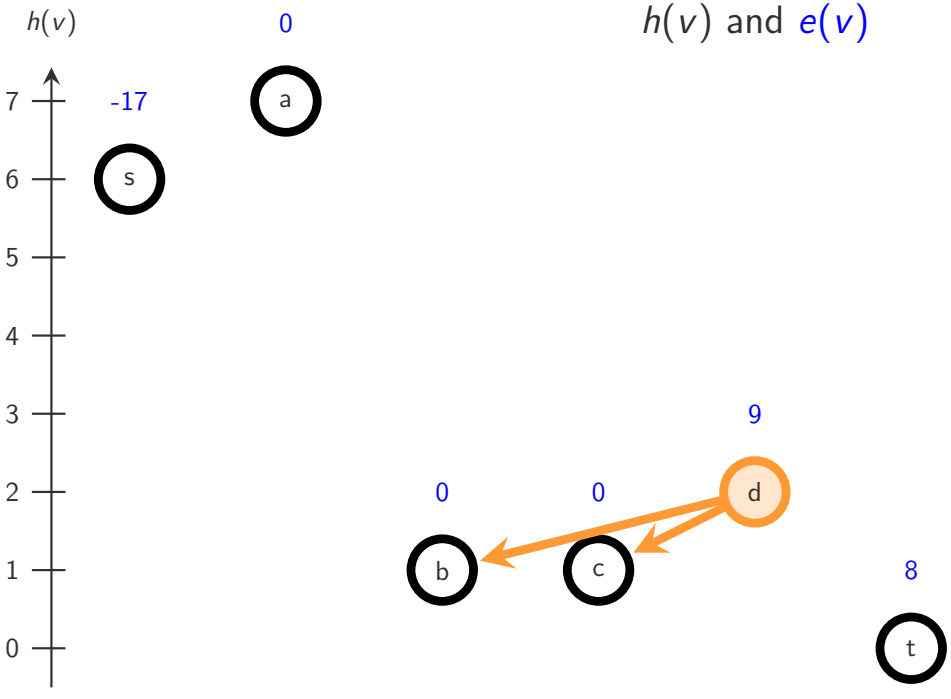

G_f and c_f
a
c
s
b
d
t
4
4
16
12
4
9
4
4
13
4
1
13

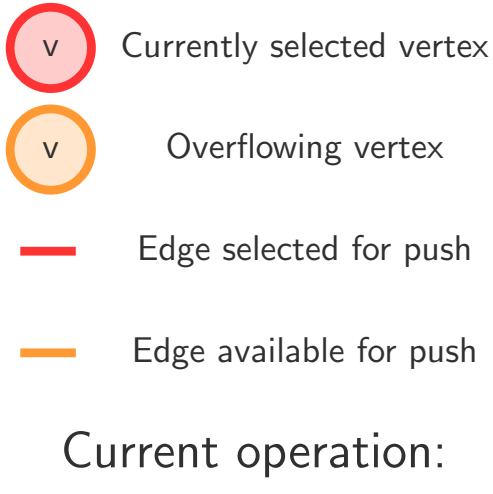

$h(v)$
$h(v)$ and $e(v)$
7
6
5
4
3
2
1
0
-17
0
0
0
9
8
s
a
b
c
d
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Relabel d



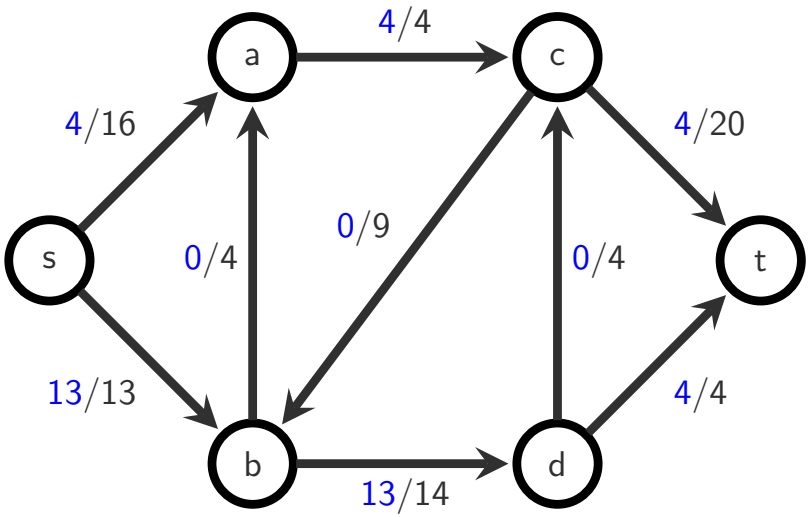




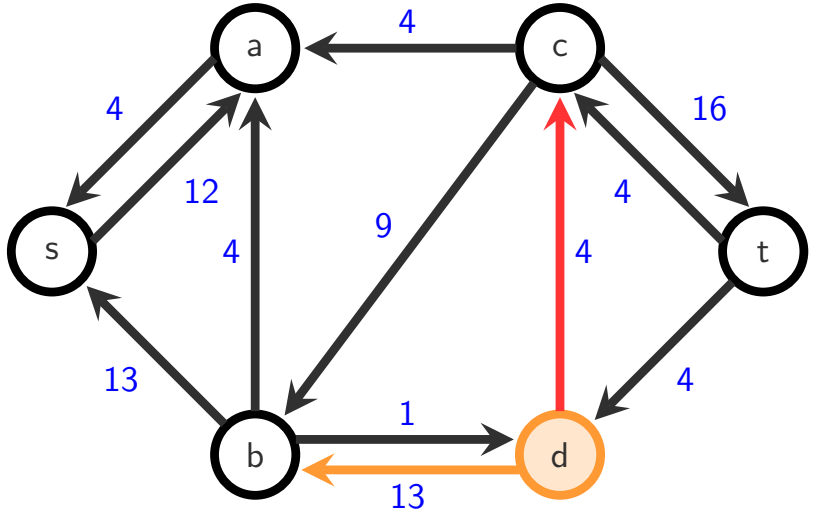


Current operation:

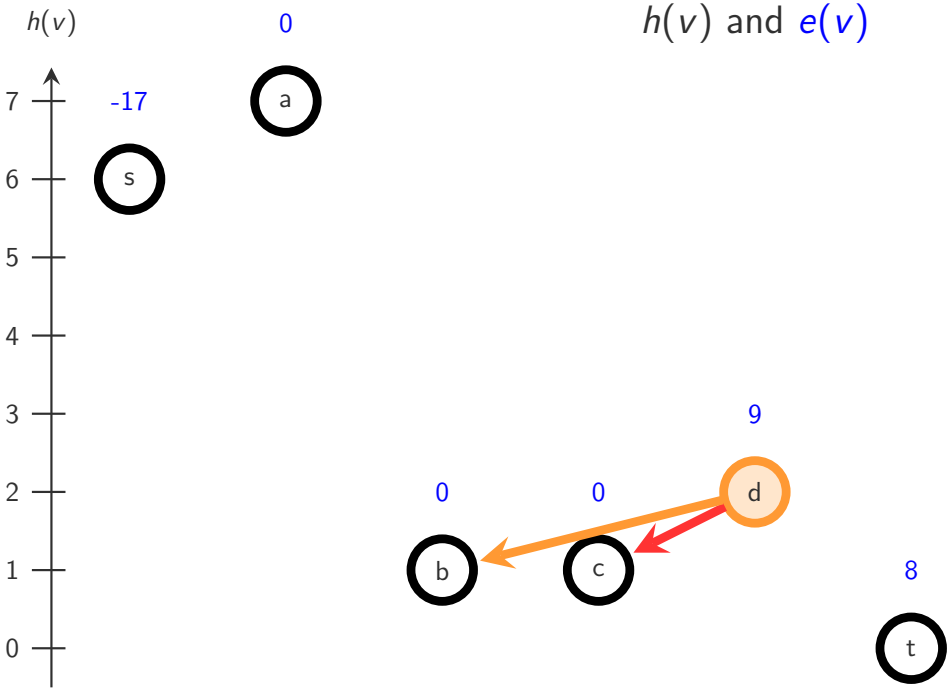
G, c and f

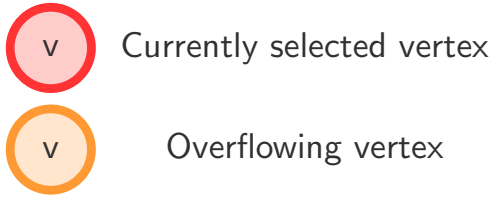


G_f and c_f



$h(v)$ and $e(v)$

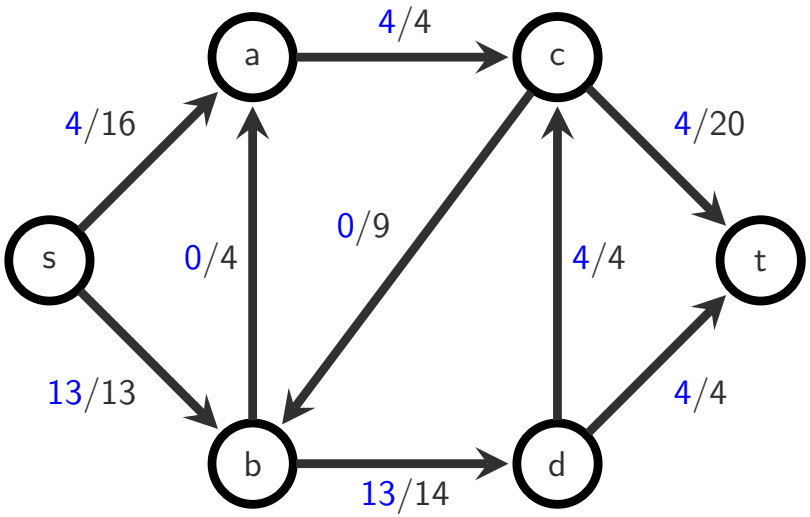




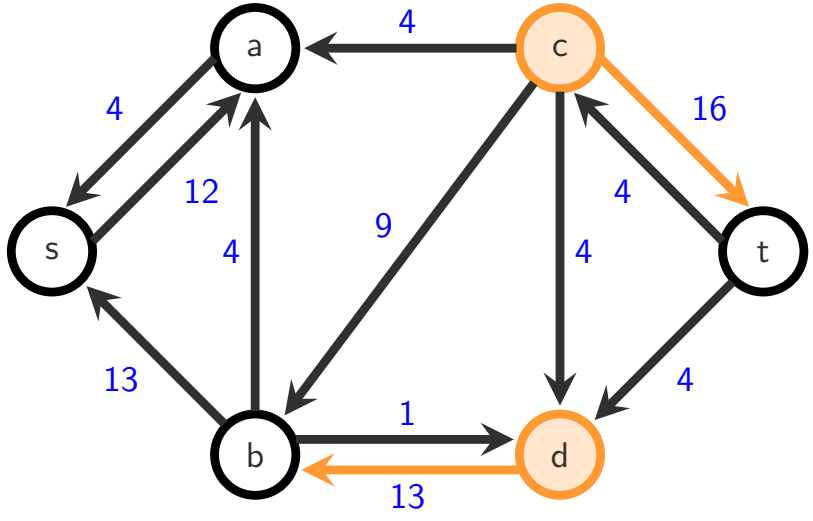
Current operation:

Push from d to c

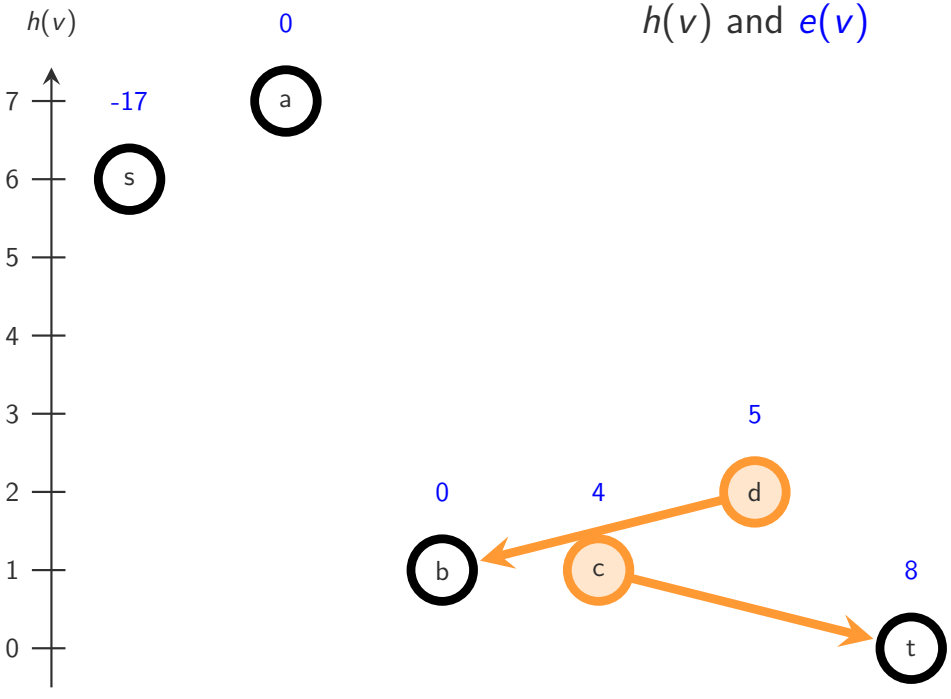
G, c and f

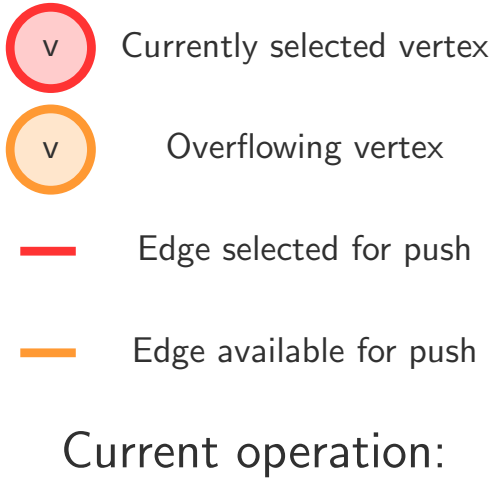


G_f and c_f



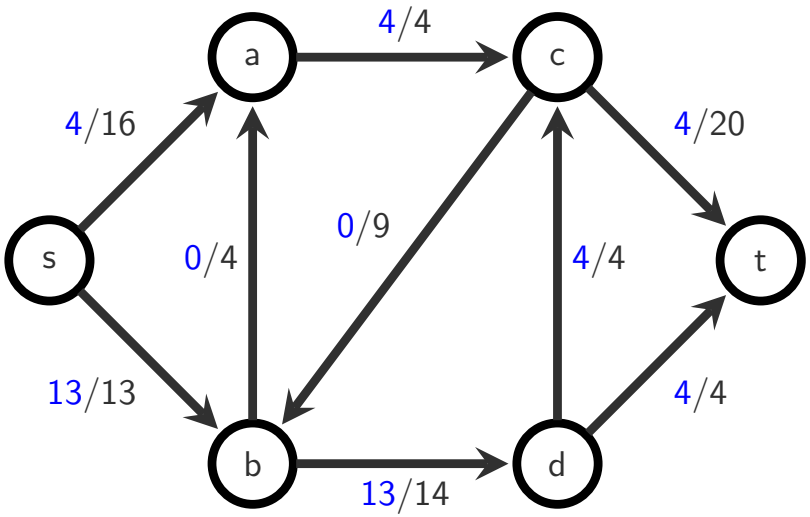
$h(v)$ and $e(v)$



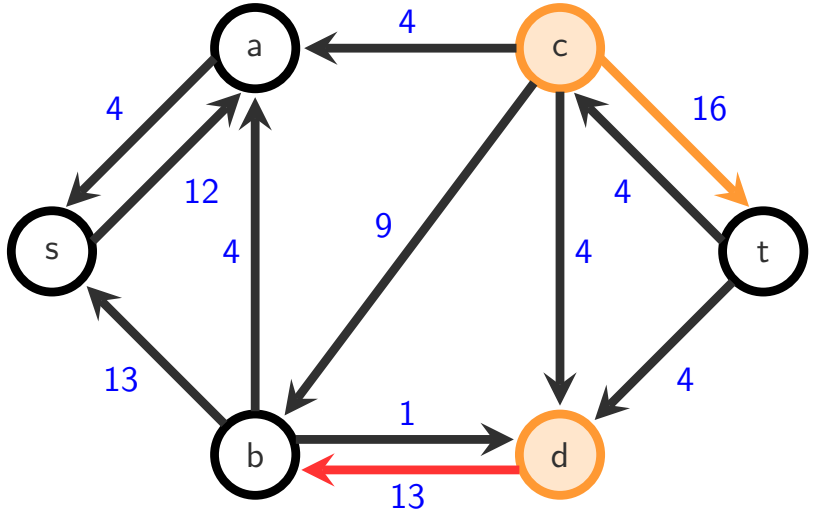


Current operation:

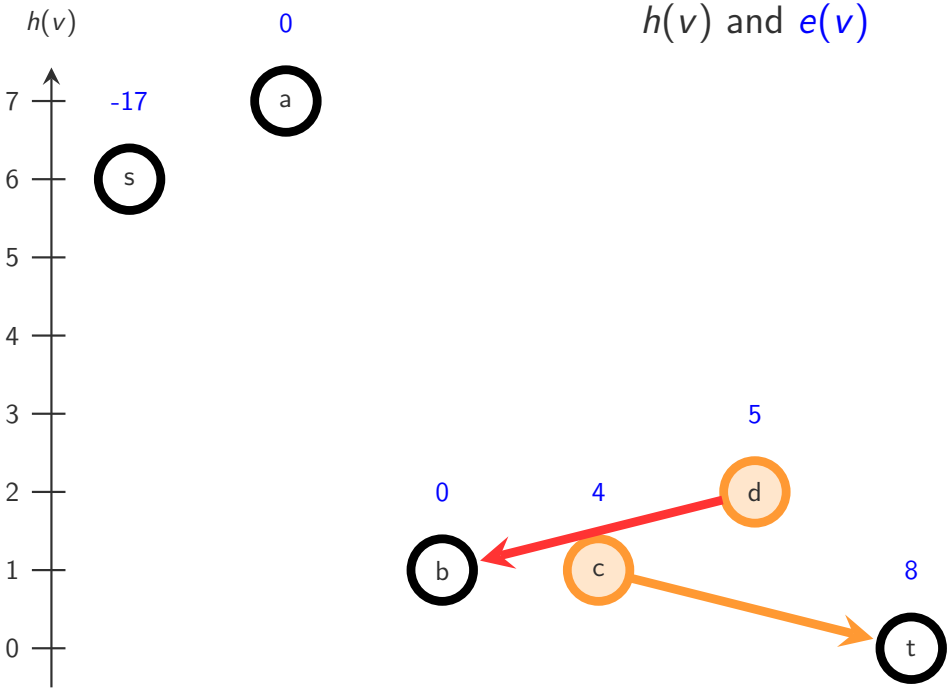
G, c and f



G_f and c_f



$h(v)$ and $e(v)$



v Currently selected vertex

v Overflowing vertex

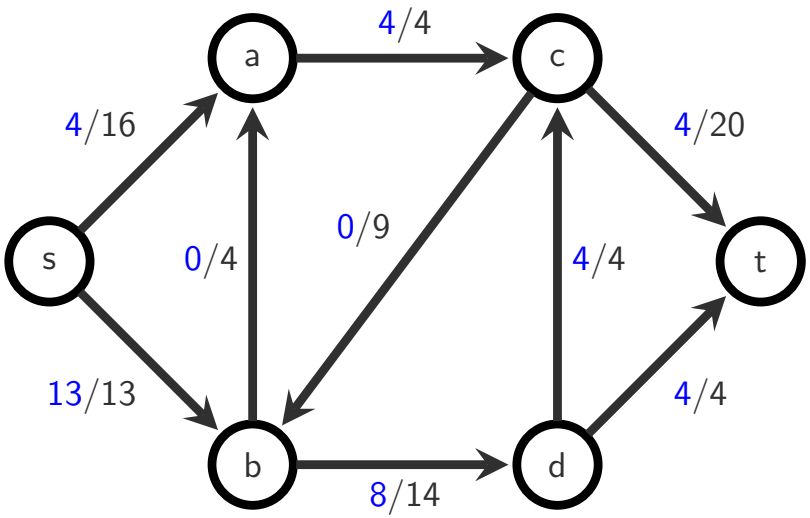
Edge selected for push

Edge available for push

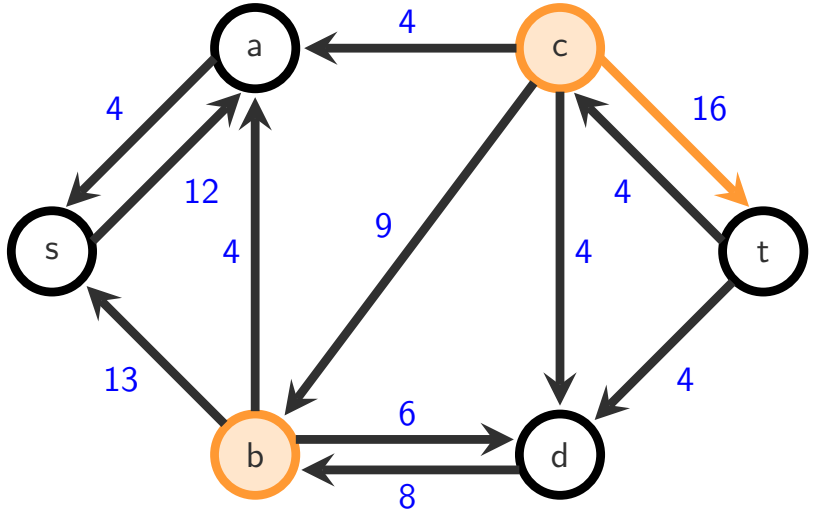
Current operation:

Push from d to b

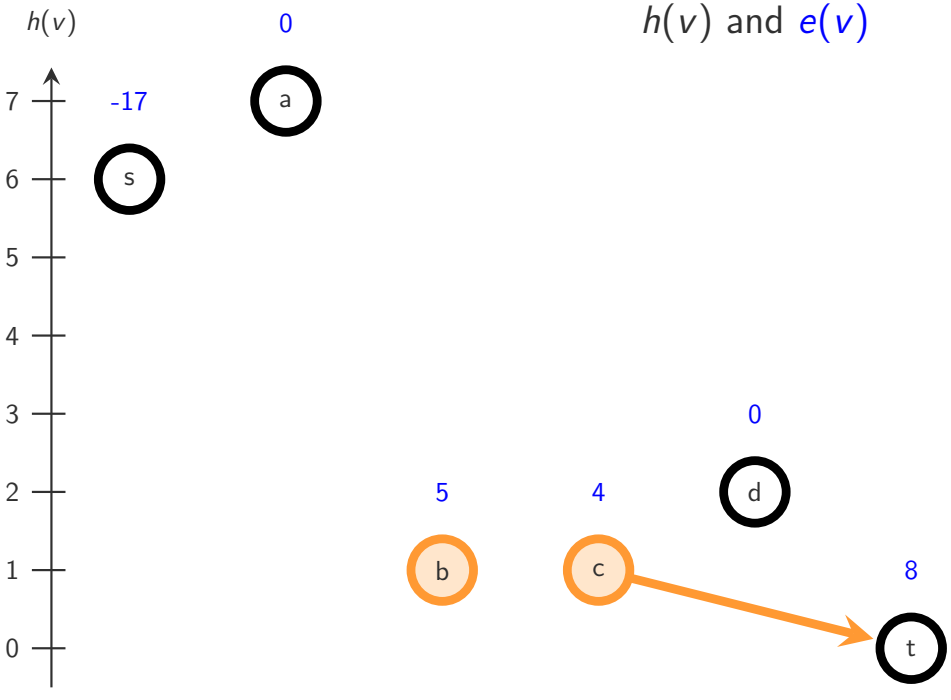
G, c and f

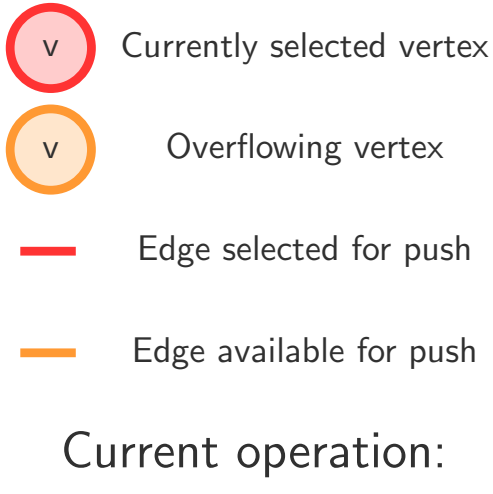


G_f and c_f



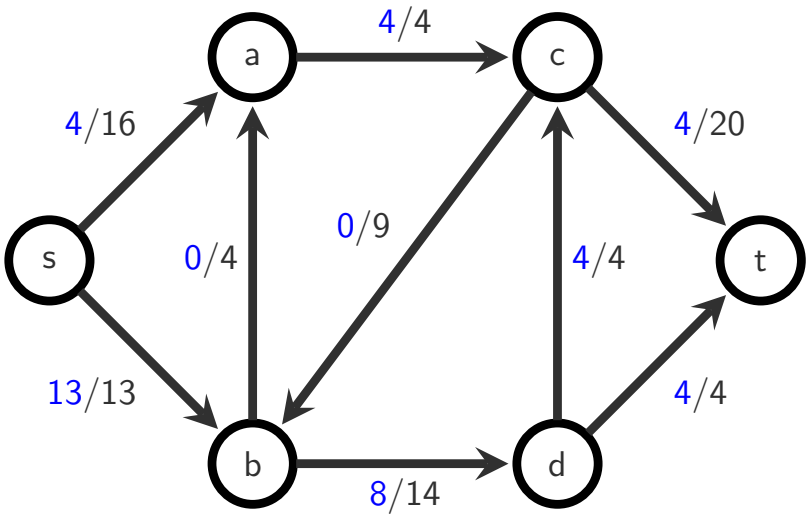
$h(v)$ and $e(v)$



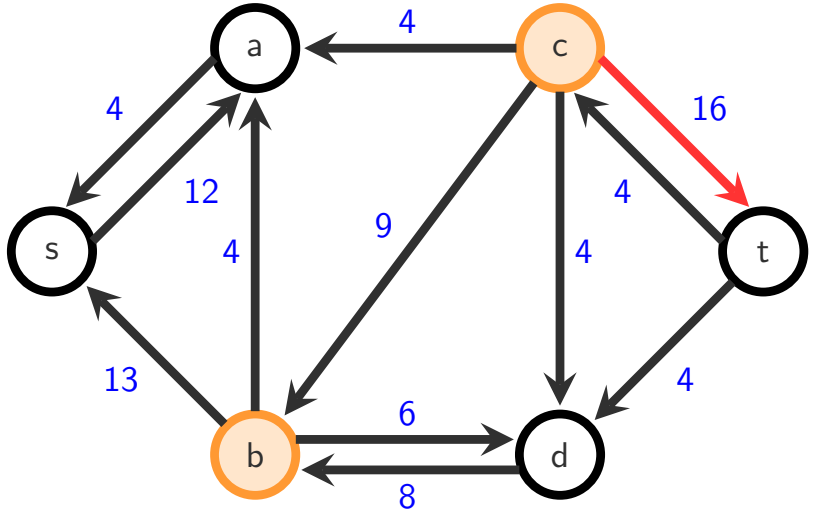


Current operation:

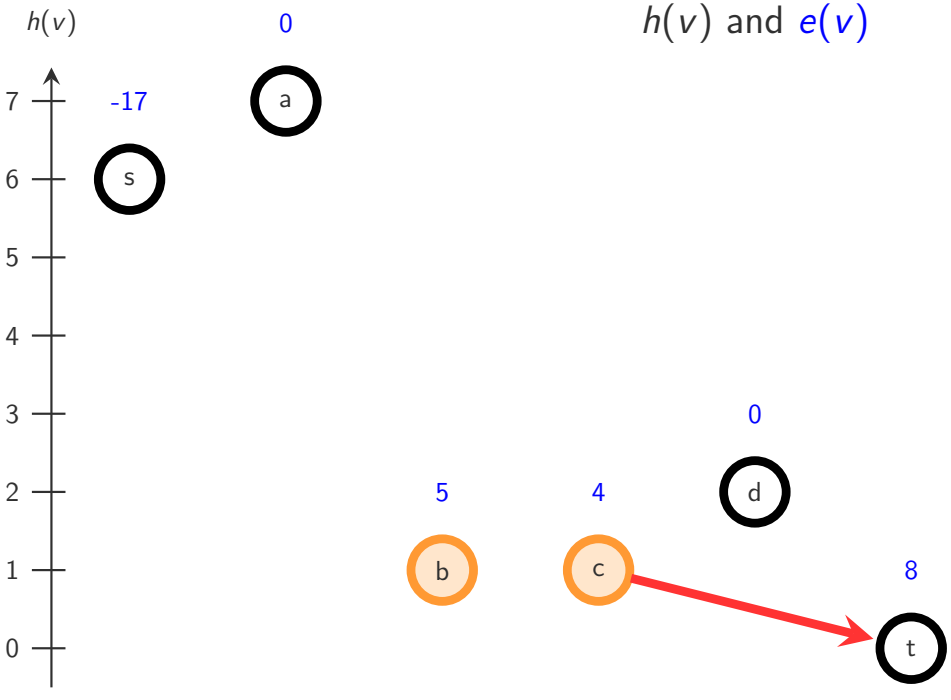
G, c and f

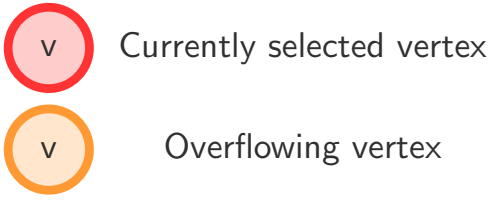


G_f and c_f



$h(v)$ and $e(v)$

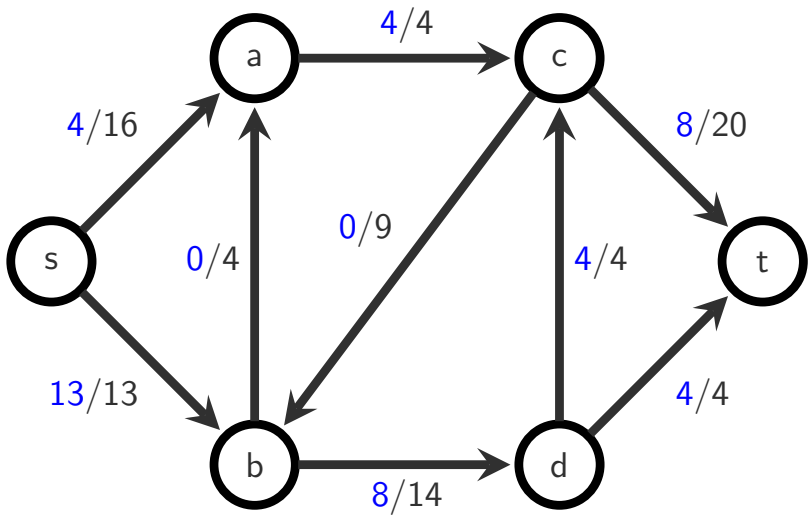




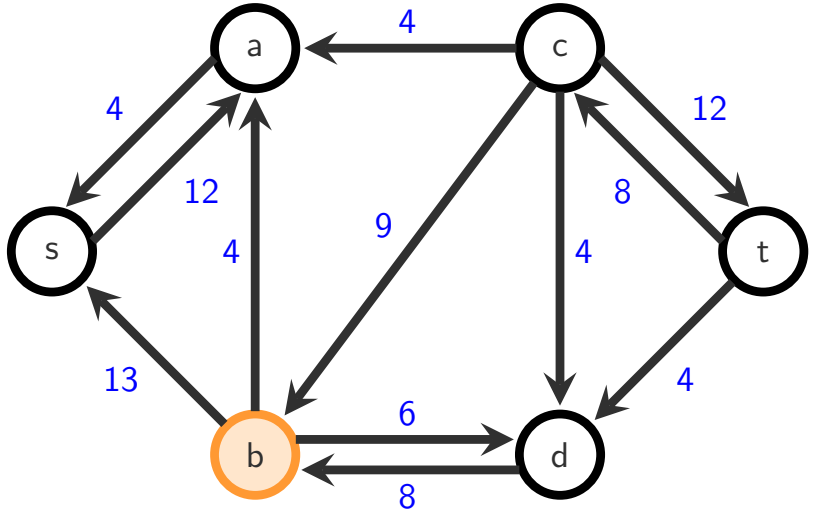
Current operation:

Push from c to t

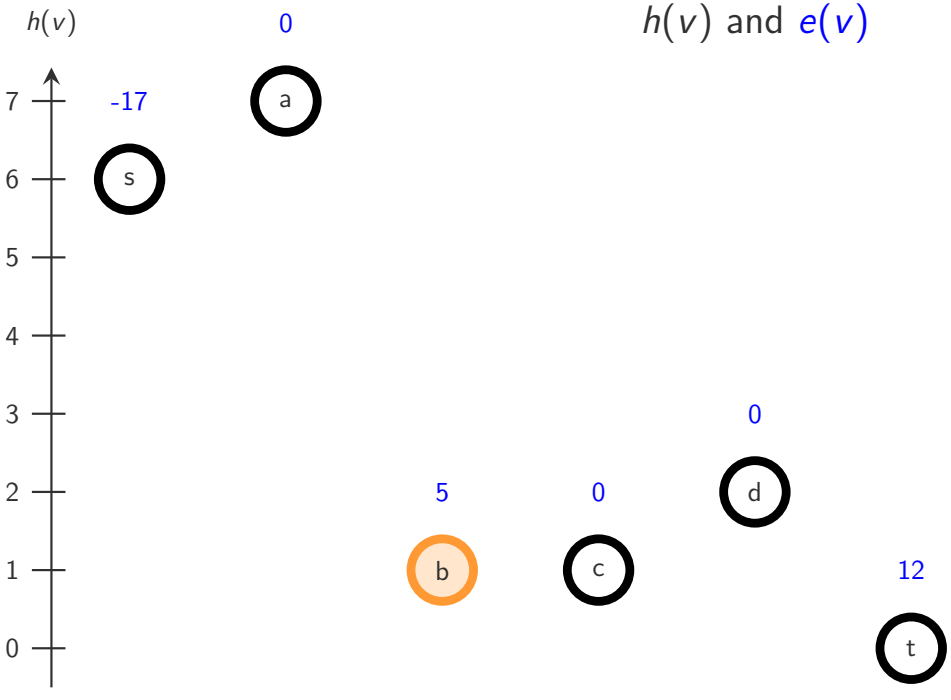
G, c and f

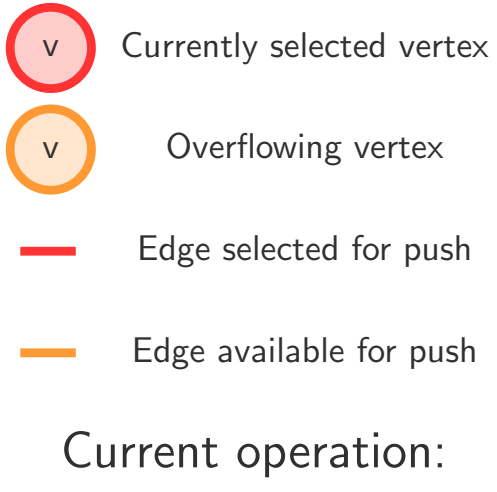


G_f and c_f



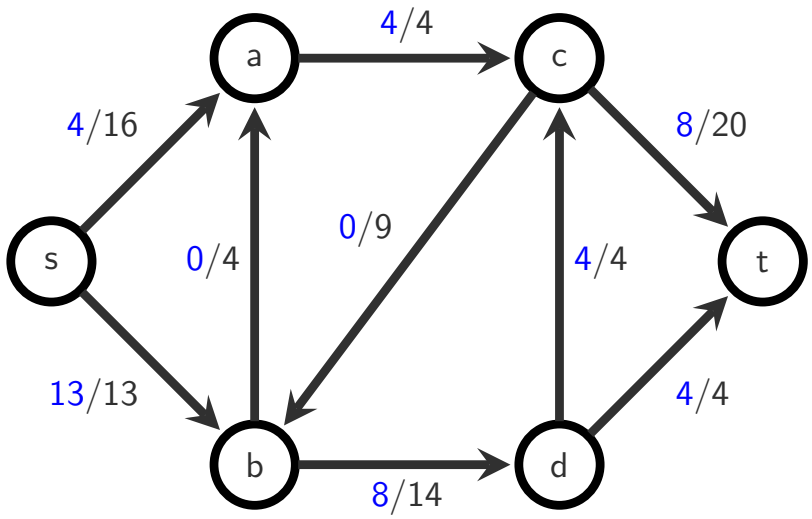
$h(v)$ and $e(v)$



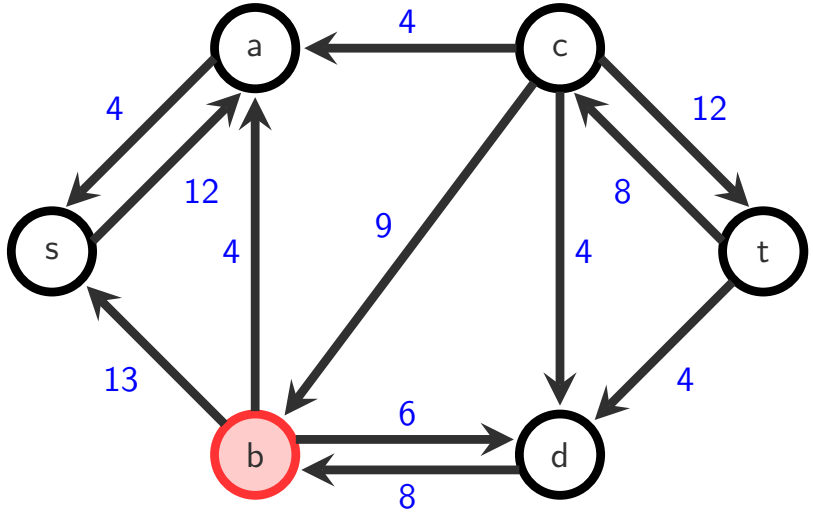


Current operation:

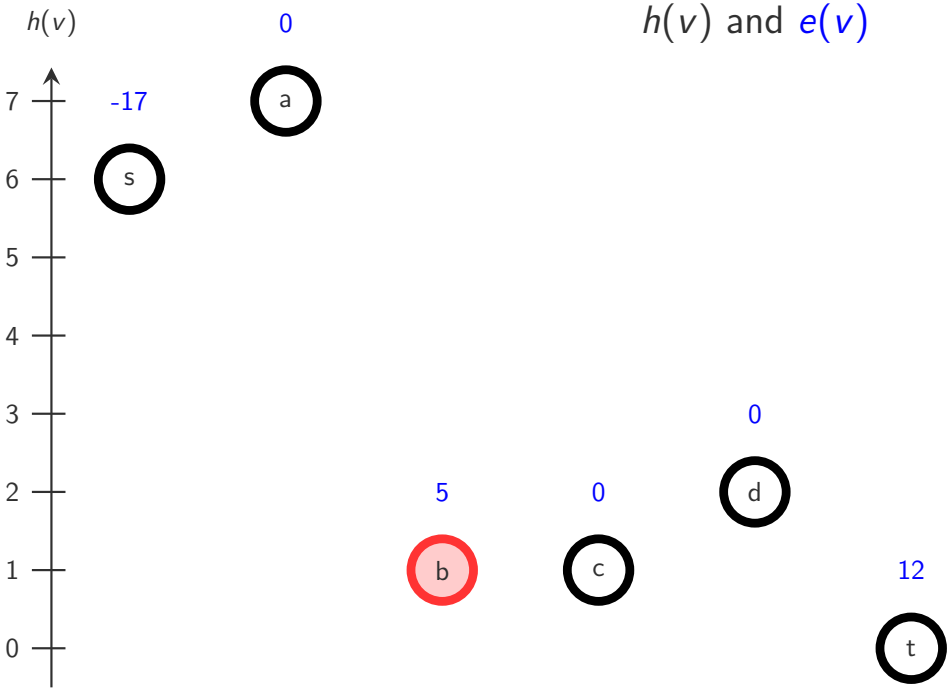
G, c and f

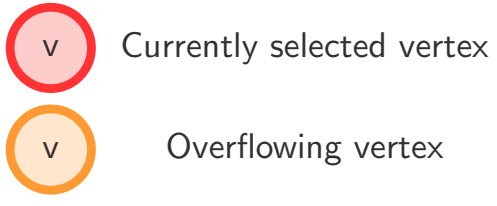


G_f and c_f



$h(v)$ and $e(v)$

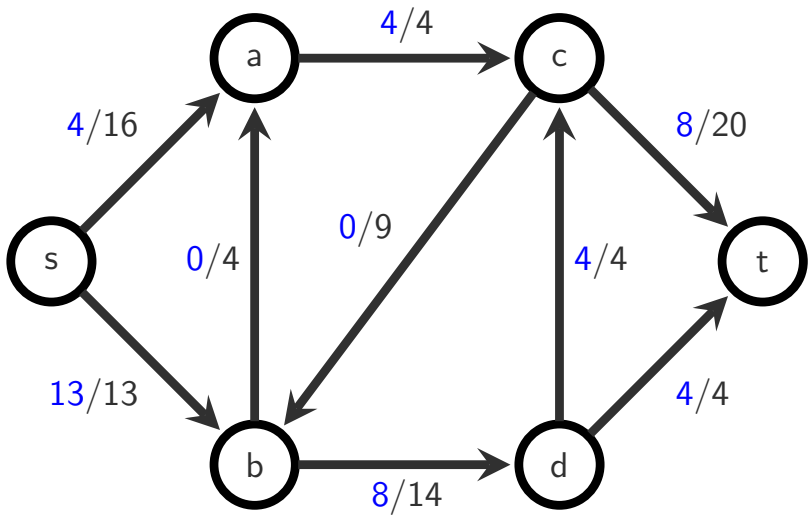




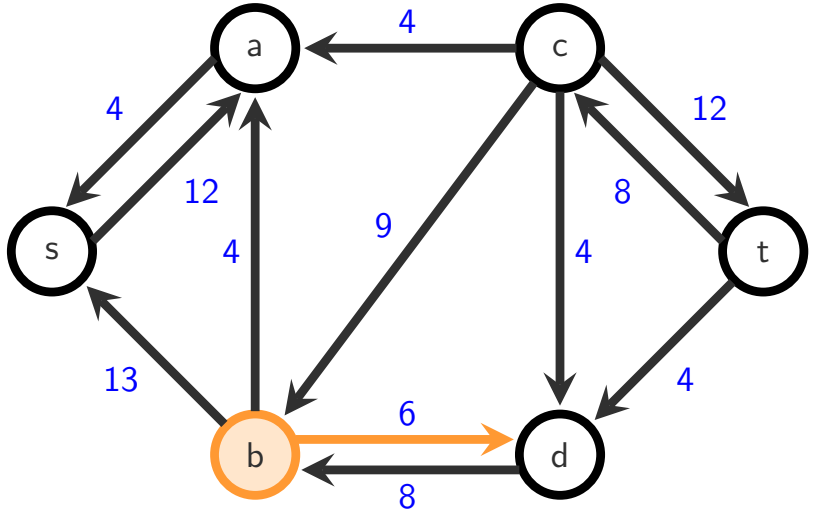
Current operation:

Relabel b

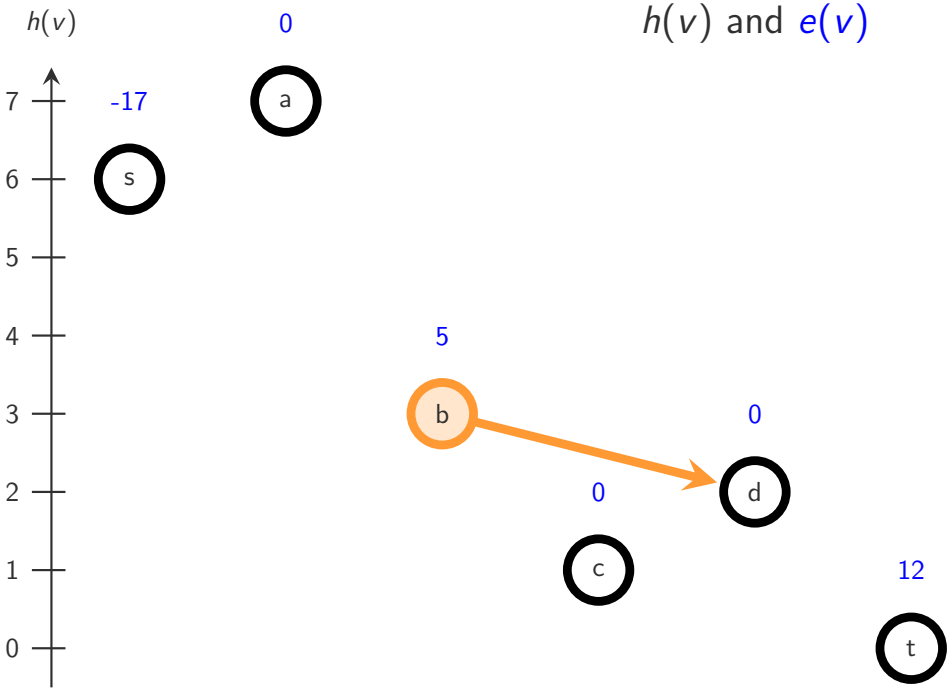
G, c and f

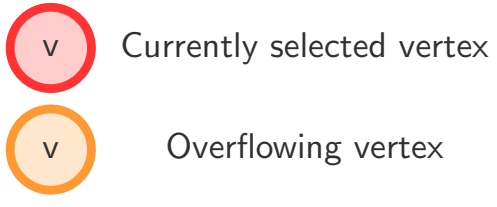


G_f and c_f

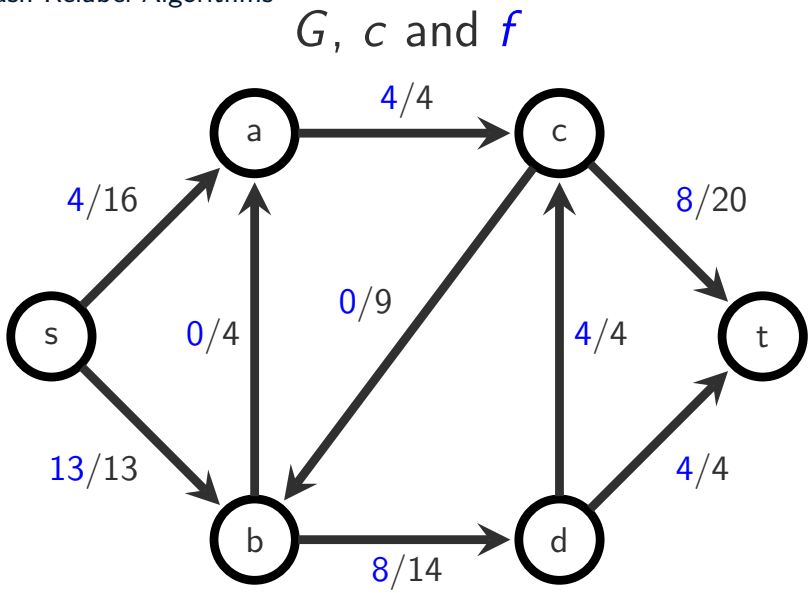


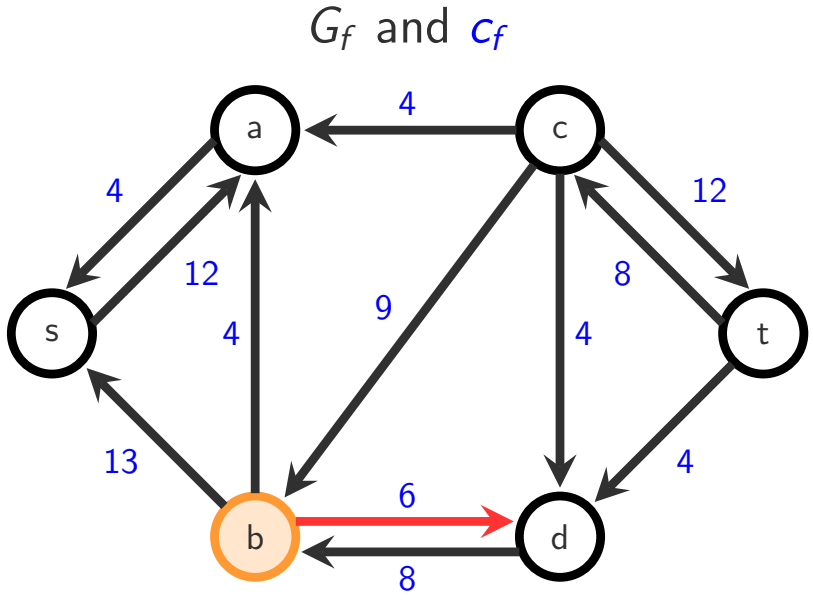
$h(v)$ and $e(v)$

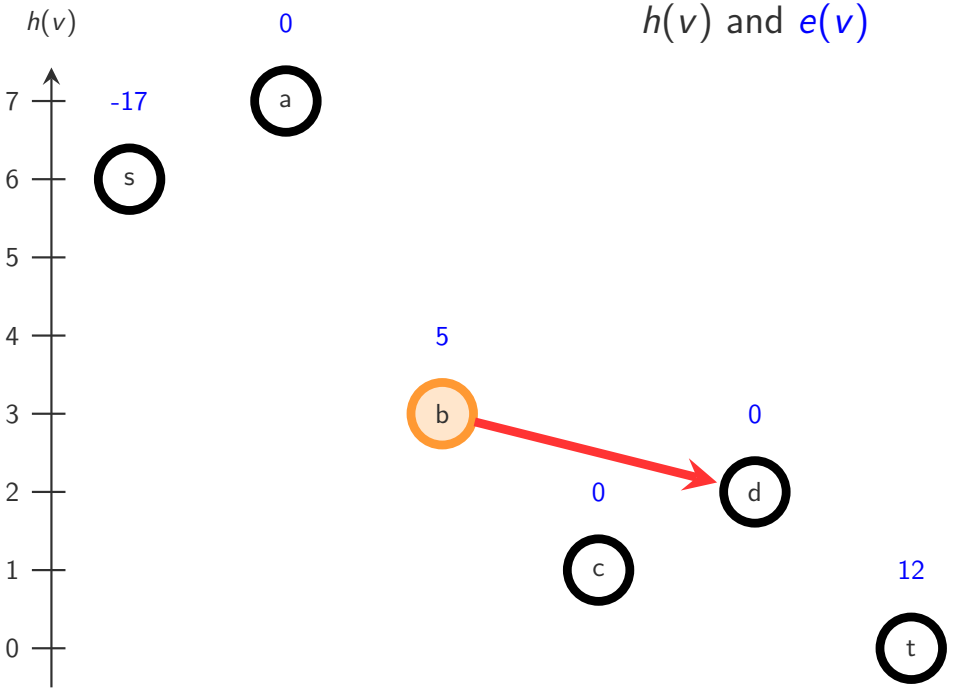


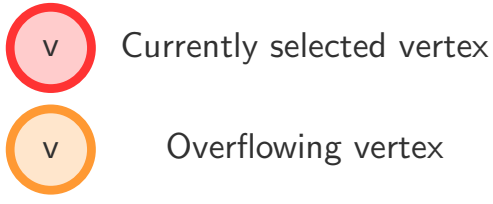


Current operation:



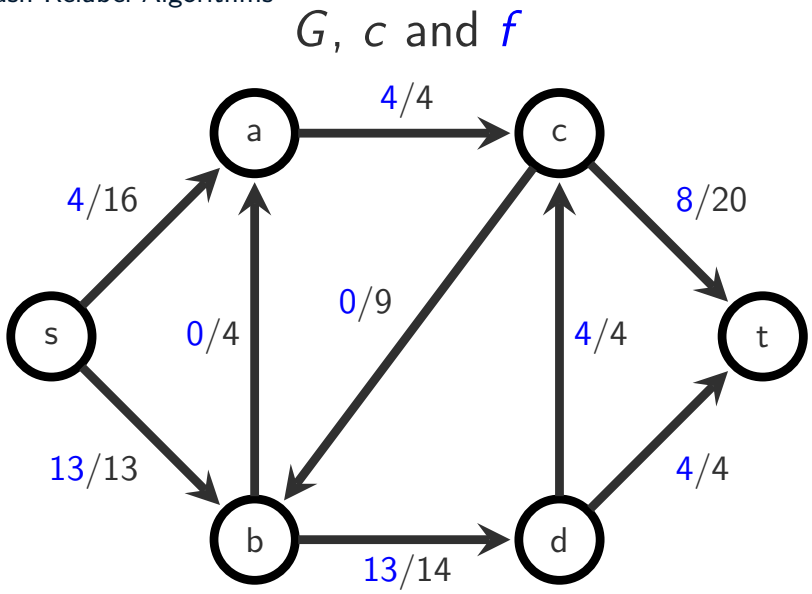


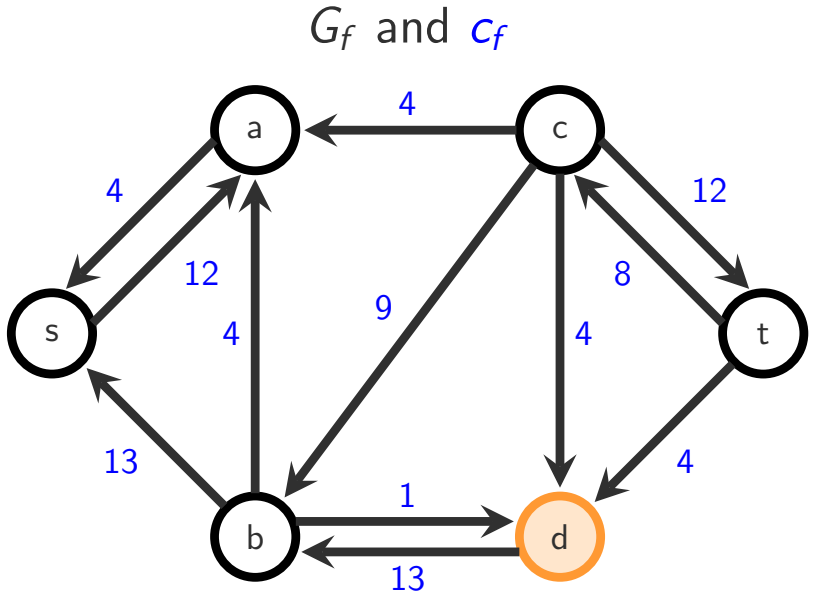


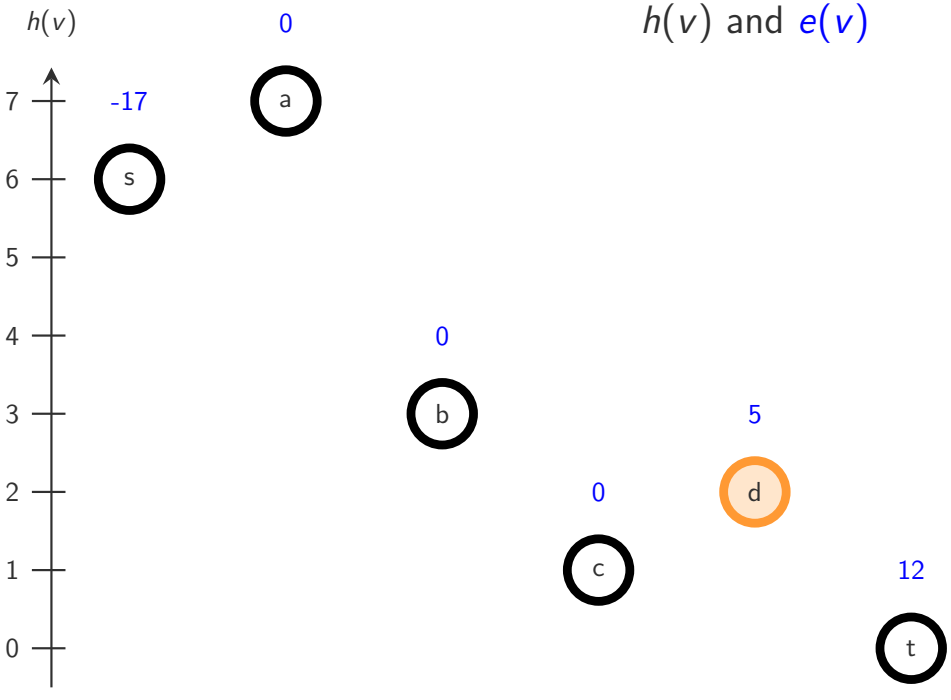


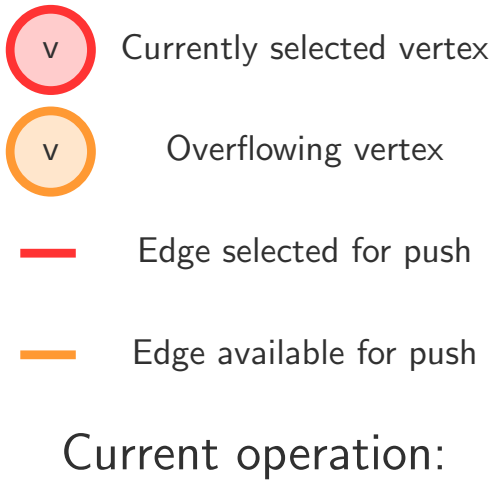
Current operation:

Push from b to d

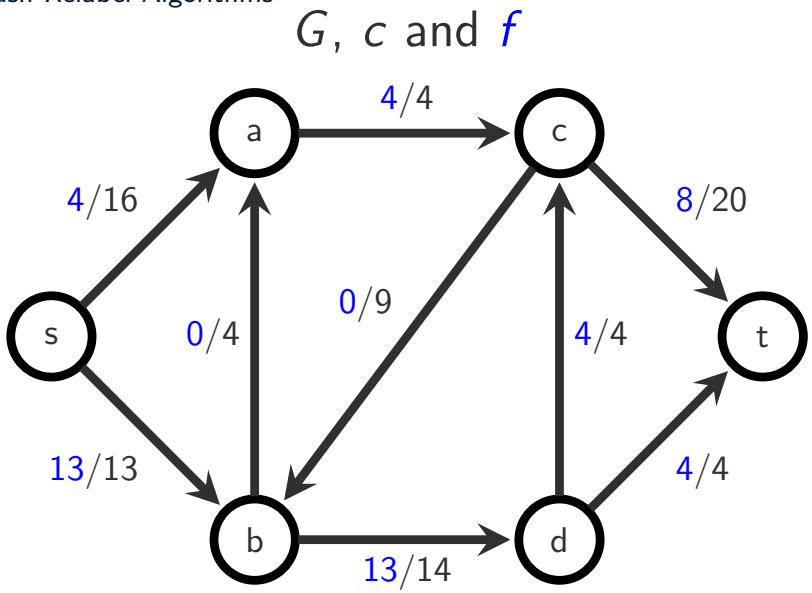


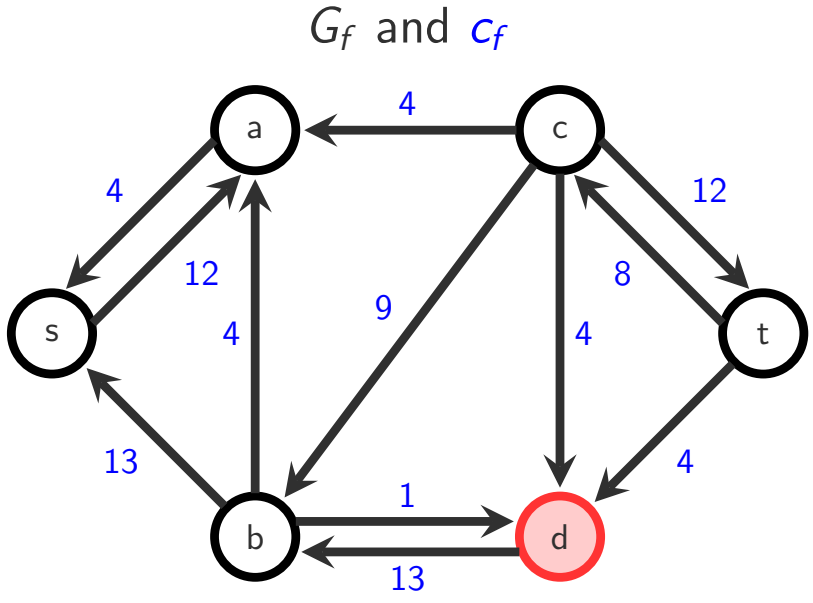


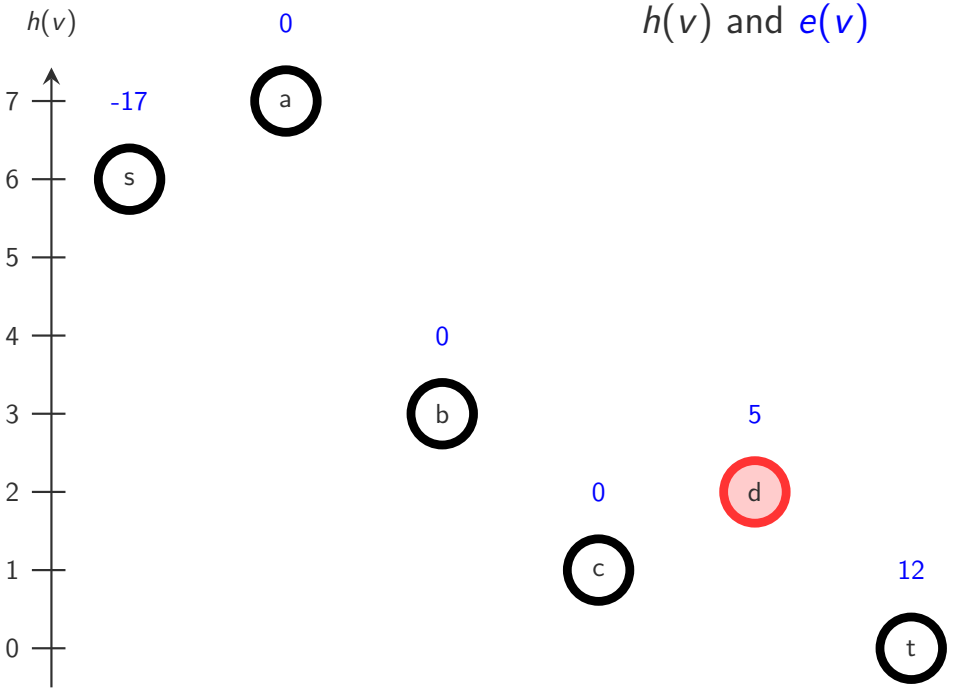


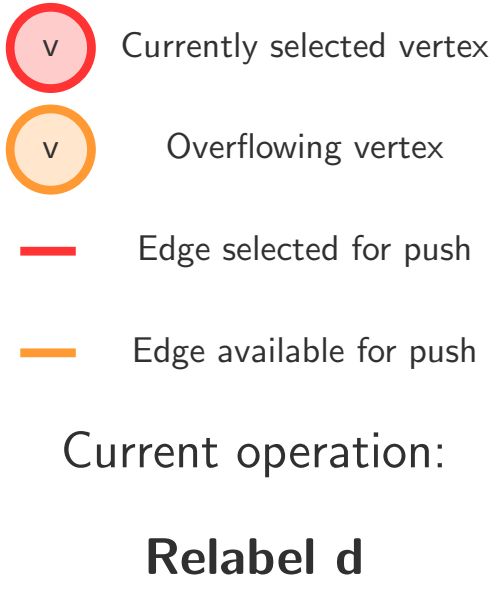


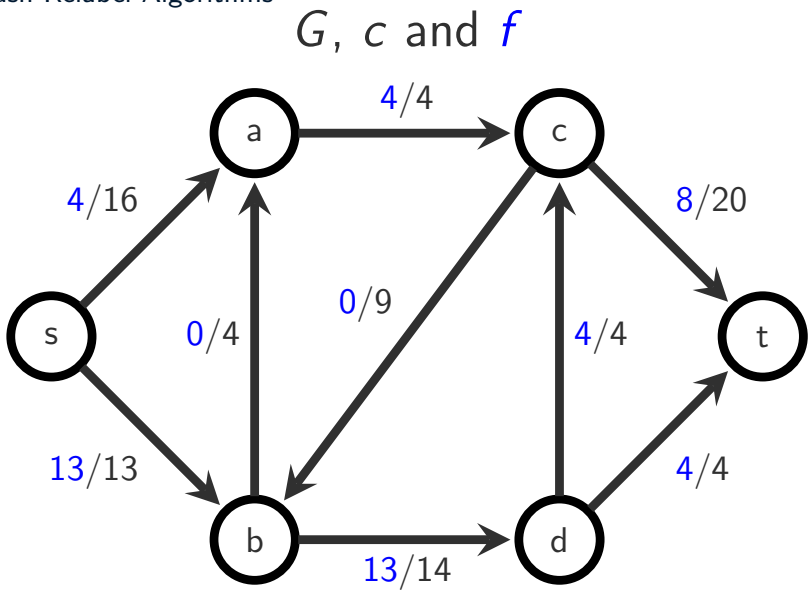
Current operation:

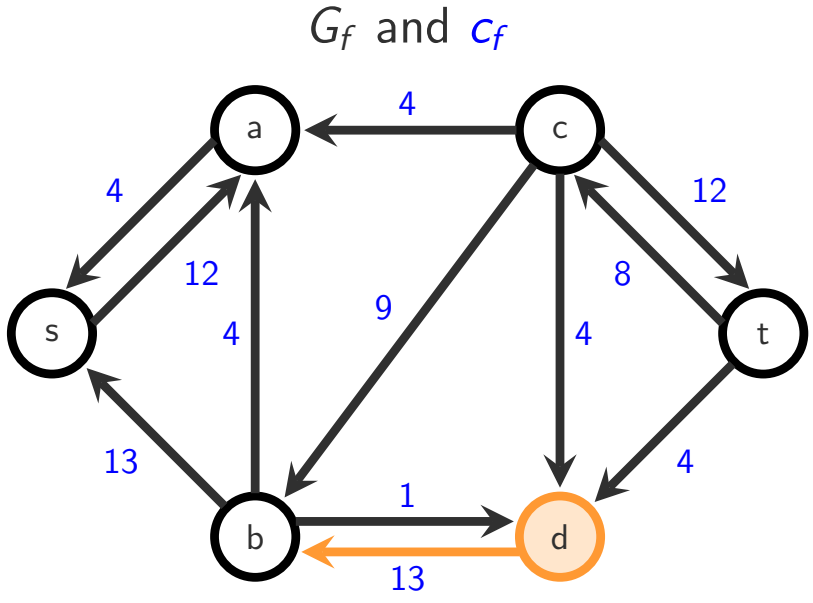

G, c and f
a
c
s
b
d
t
4/4
4/16
8/20
0/4
0/9
4/4
13/13
4/4
13/14

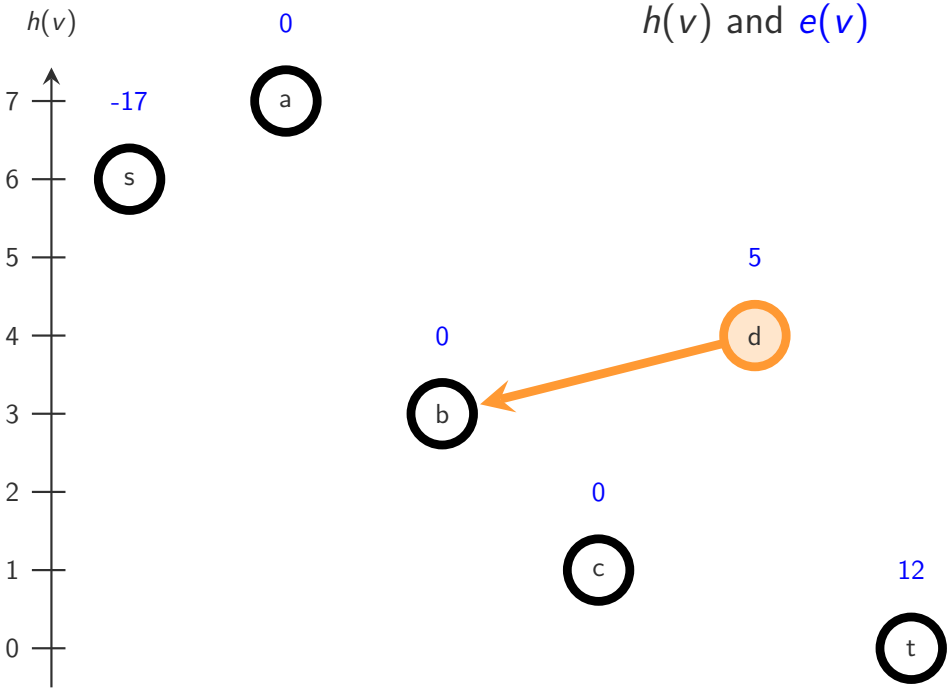

G_f and c_f
a
c
s
b
d
t
4
4
12
12
9
8
4
13
1
4
13

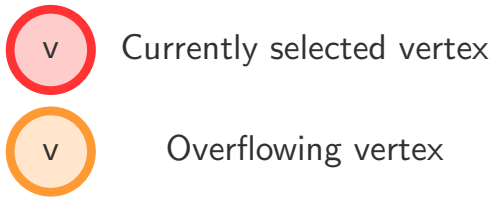

$h(v)$
$h(v)$ and $e(v)$
7
6
5
4
3
2
1
0
-17
0
0
5
0
12
s
a
b
d
c
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Relabel d

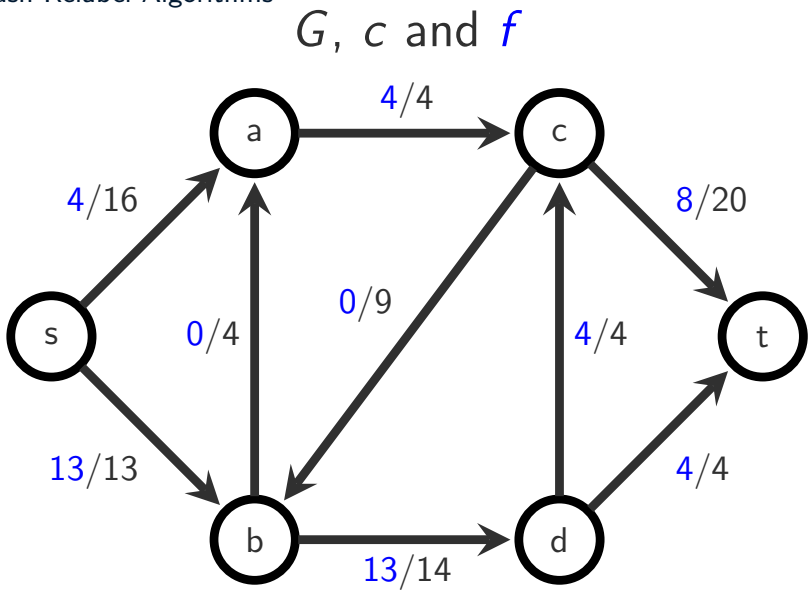


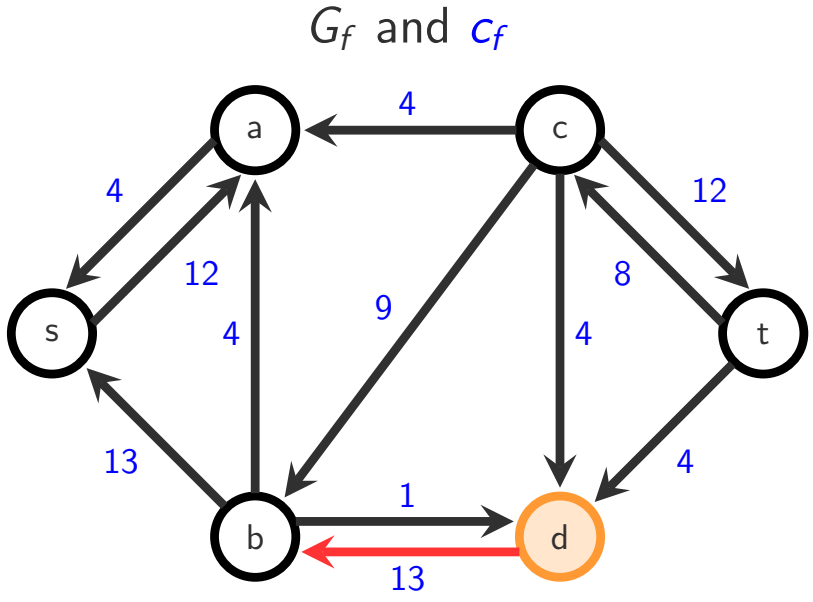


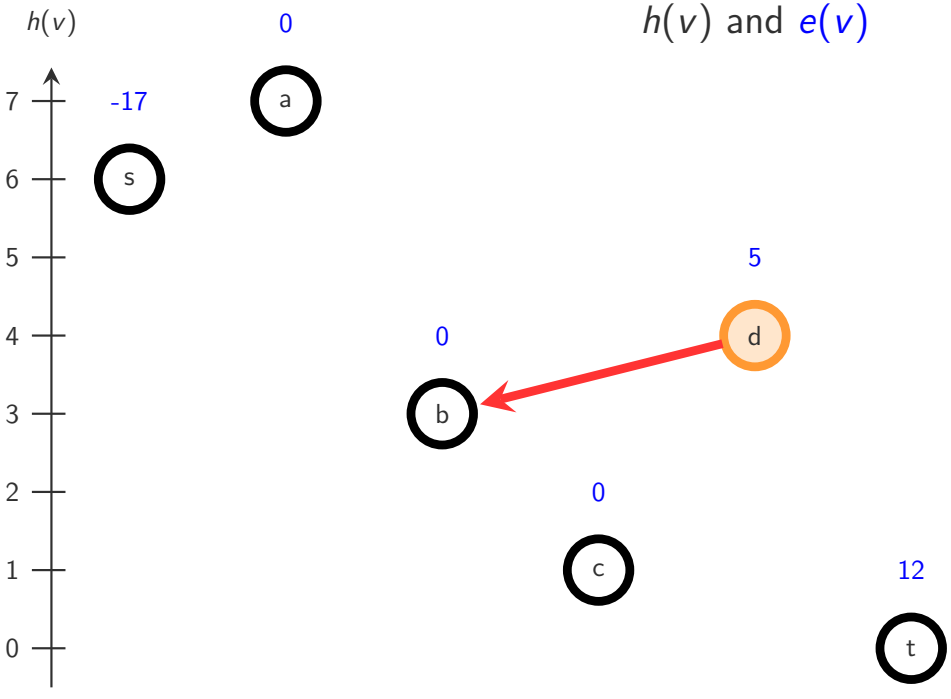


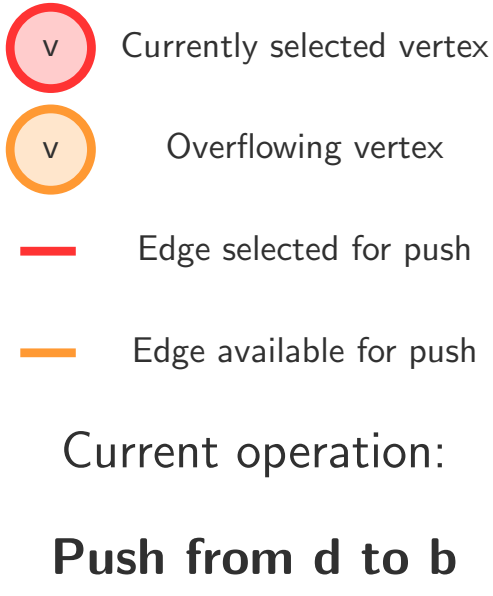


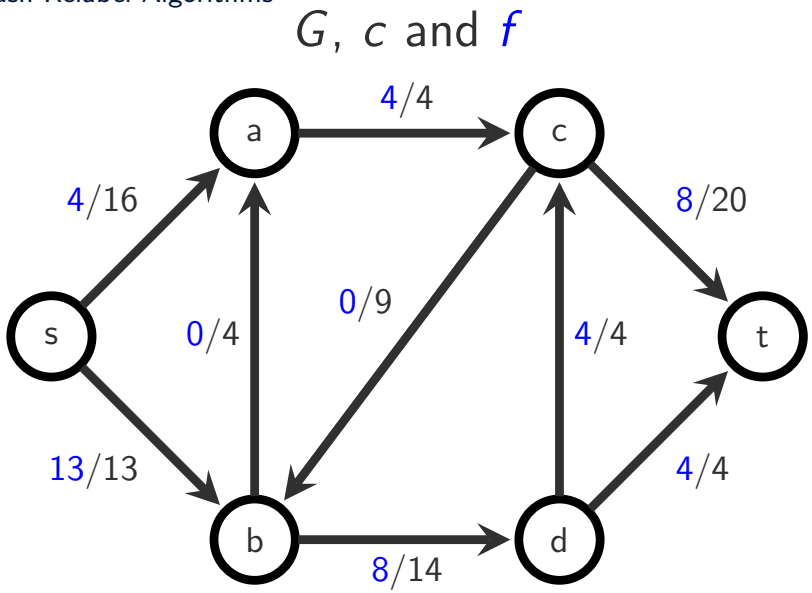
Current operation:

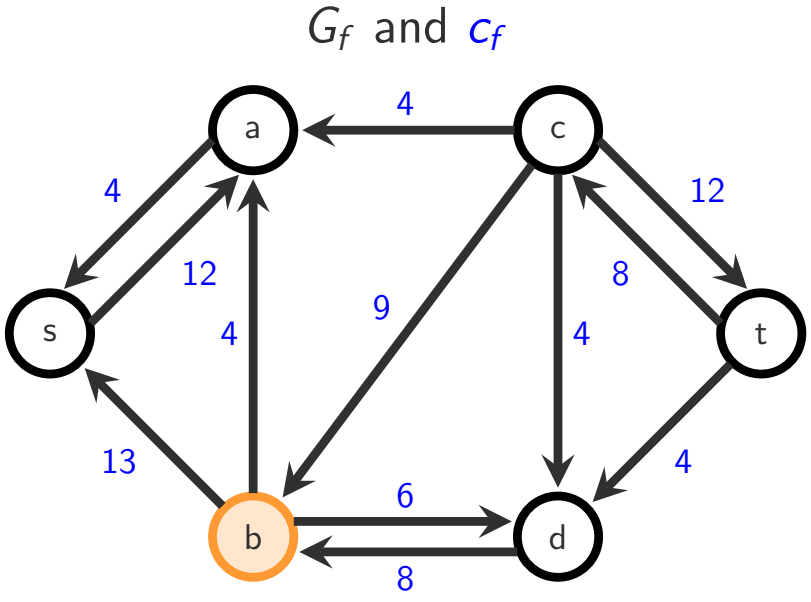

G, c and f
a
c
s
b
d
t
4/4
4/16
8/20
0/4
0/9
4/4
13/13
4/4
13/14

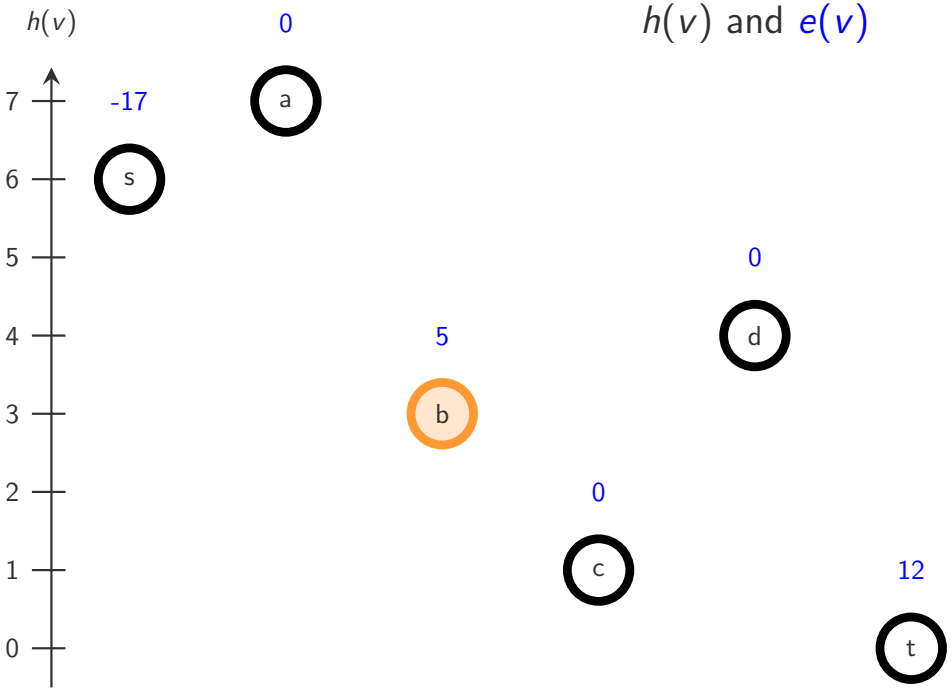

G_f and c_f
a
c
s
b
d
t
4
4
12
12
9
8
4
13
4
1
13

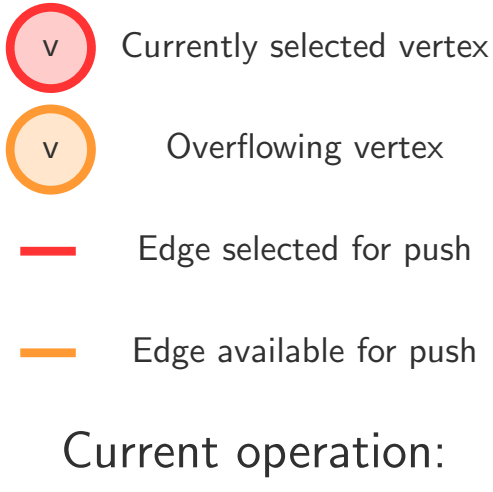

h(v)
h(v) and e(v)
0
7
6
5
4
3
2
1
0
-17
a
s
5
d
0
b
0
c
12
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Push from d to b

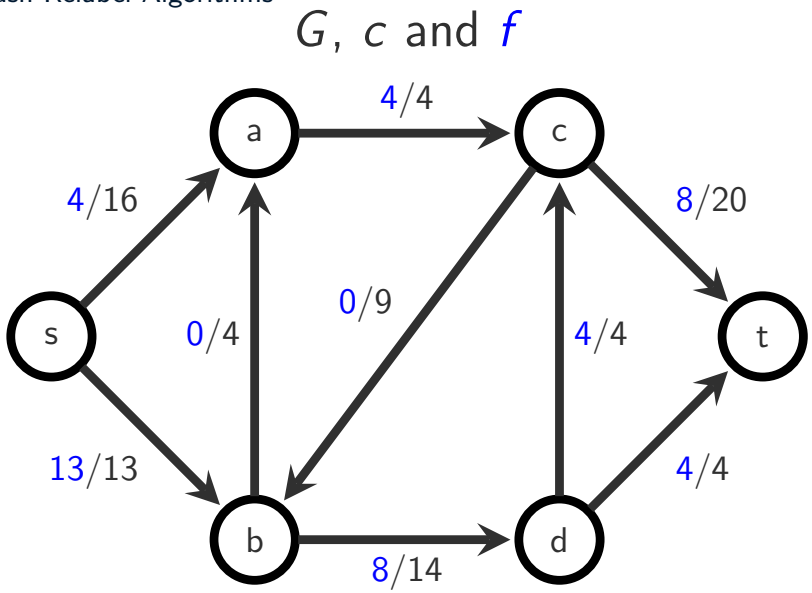


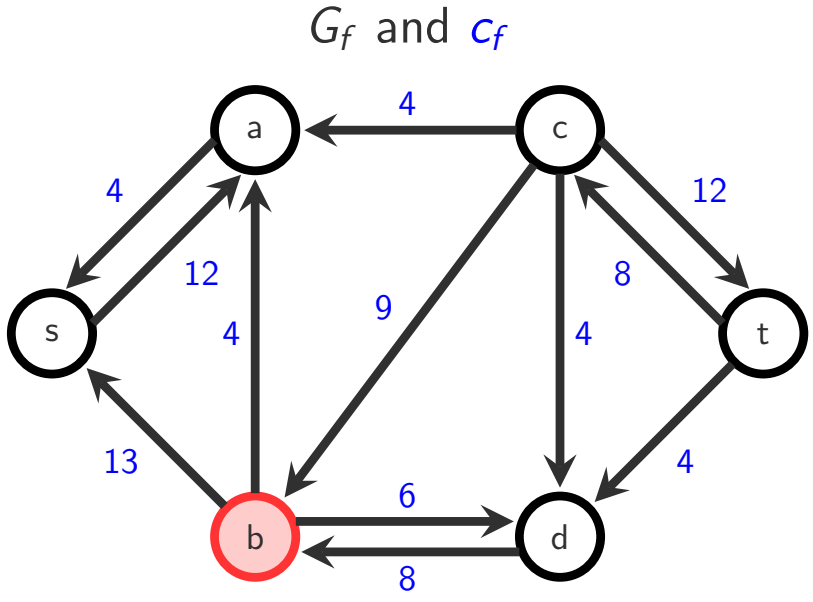


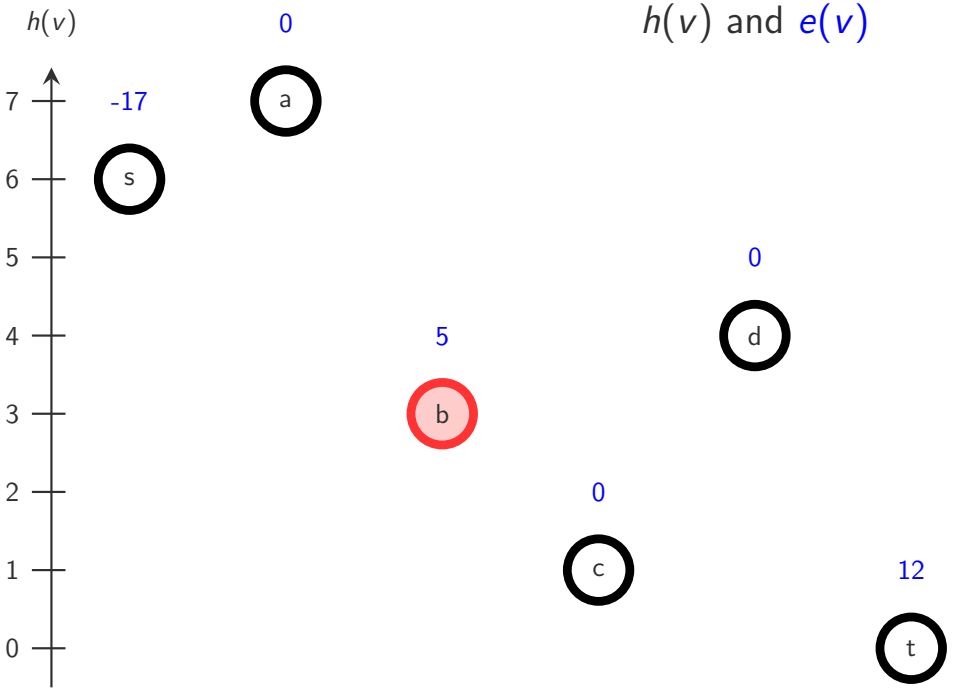


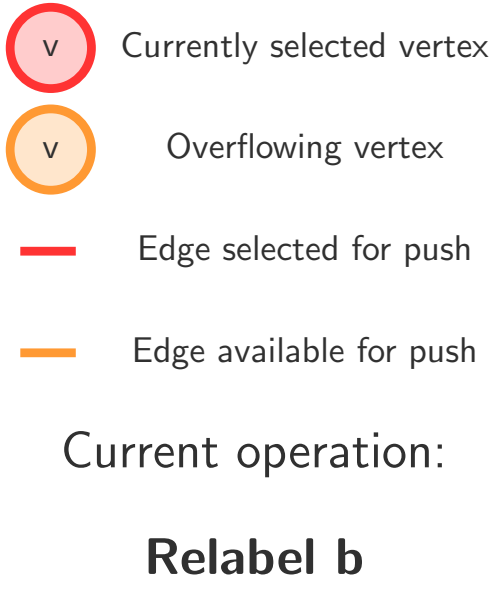


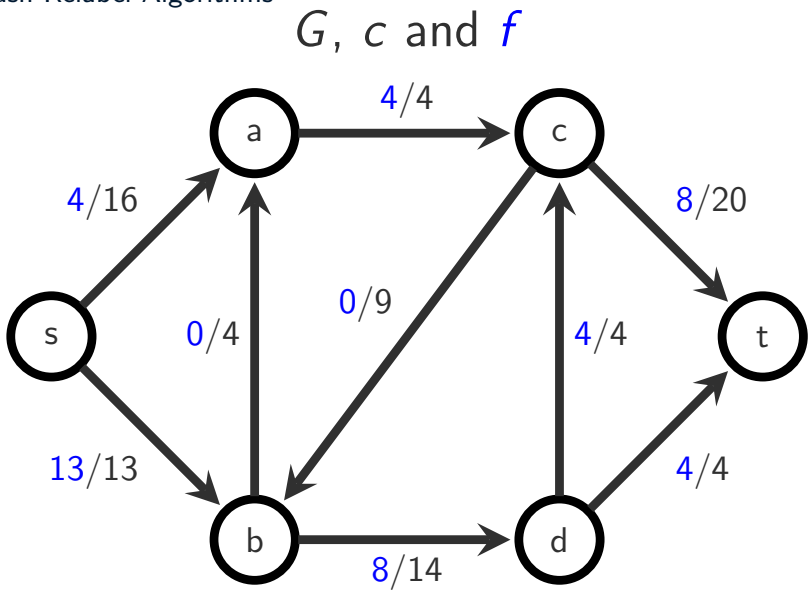
Current operation:

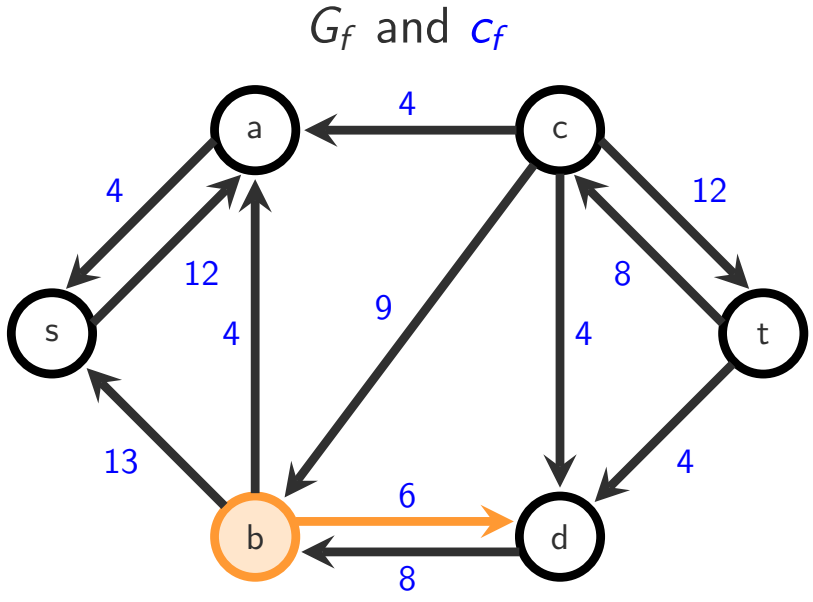

G, c and f
a
c
s
b
d
t
4/4
4/16
8/20
0/4
0/9
4/4
13/13
4/4
8/14

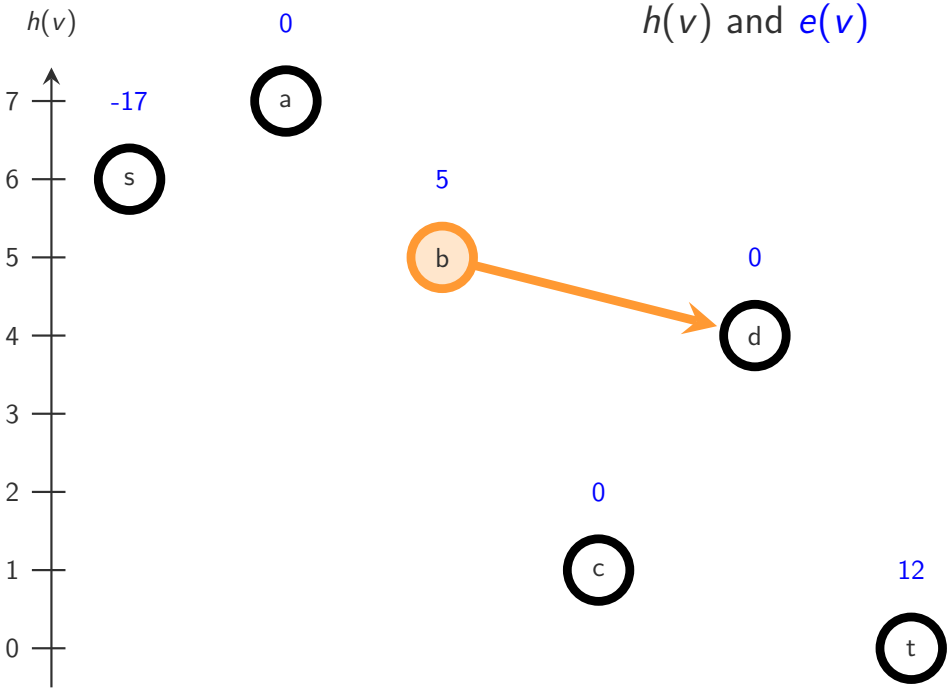

G_f and c_f
a
c
s
b
d
t
4
4
12
12
9
8
4
13
6
4
8

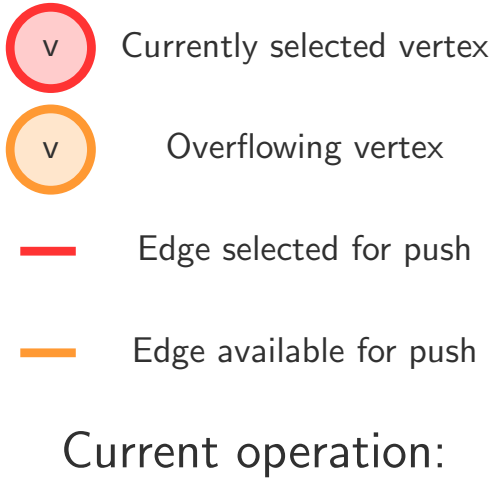

$h(v)$
$h(v)$ and $e(v)$
7
6
5
4
3
2
1
0
-17
0
0
5
0
12
s
a
d
b
c
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Relabel b

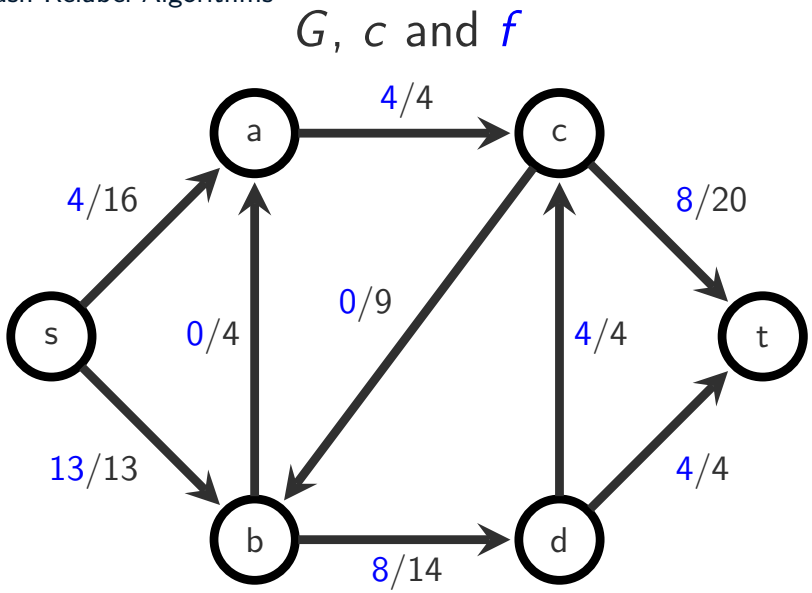


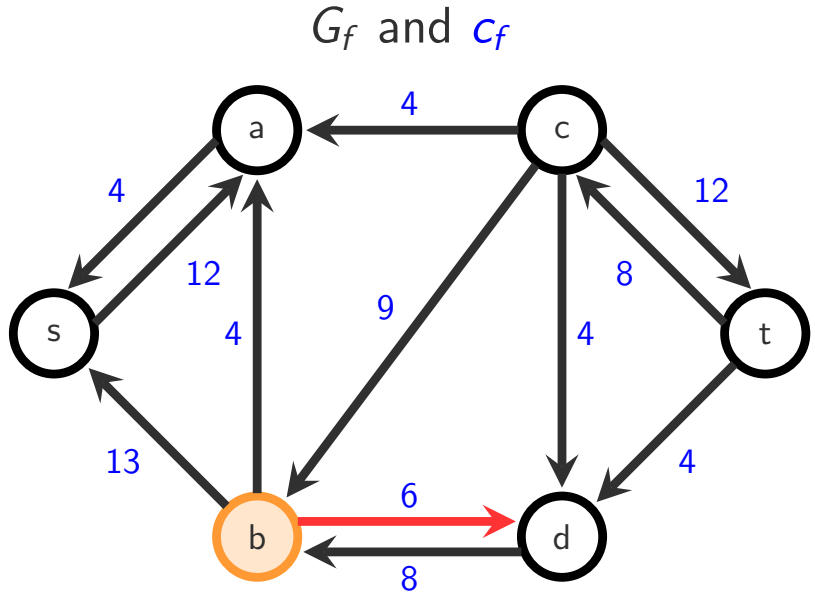


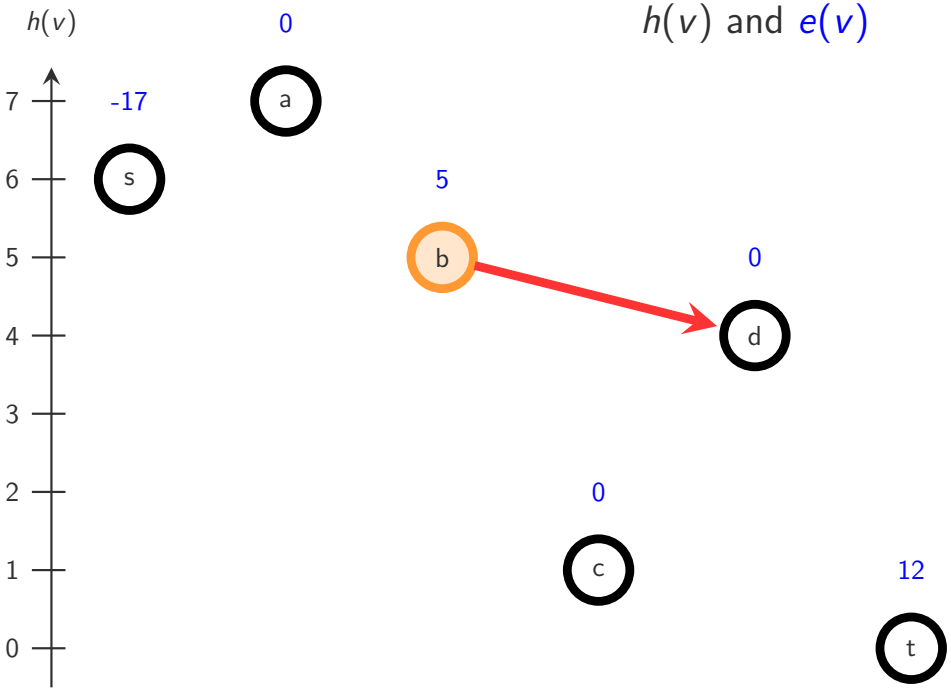




Current operation:







v Currently selected vertex

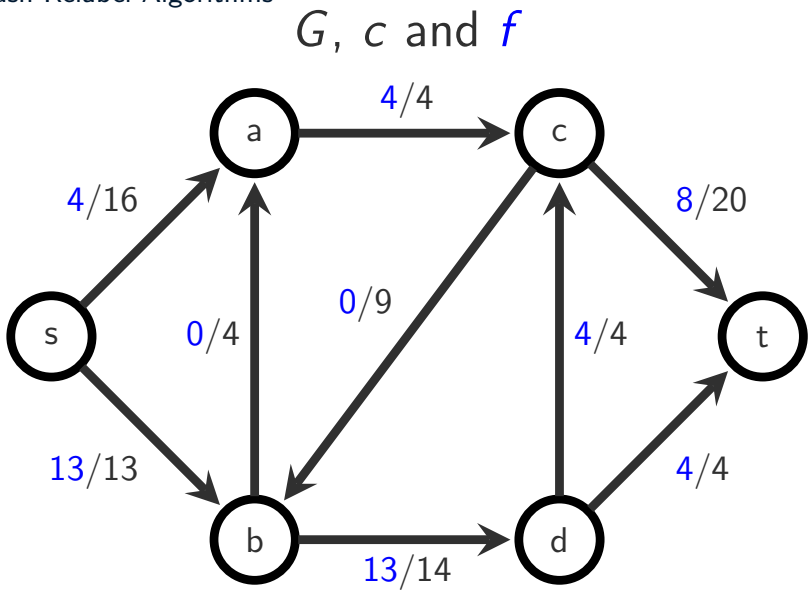
v Overflowing vertex

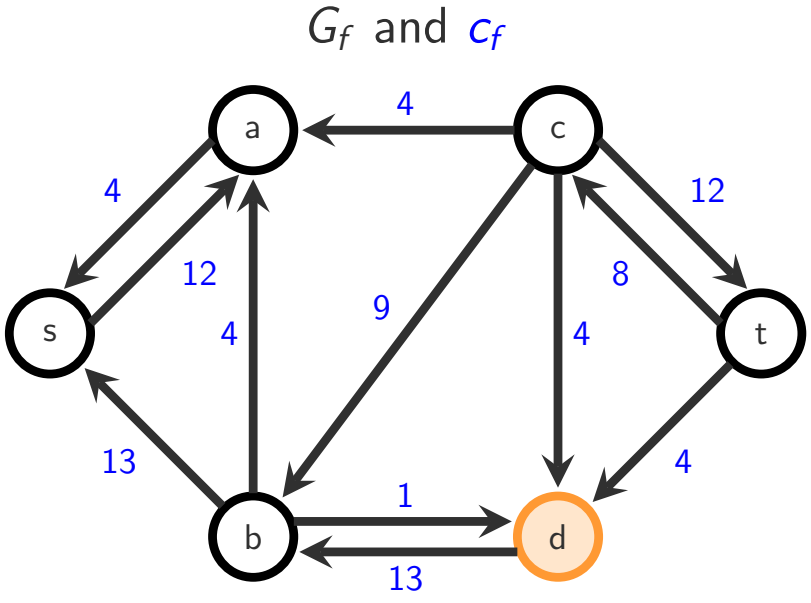
Edge selected for push

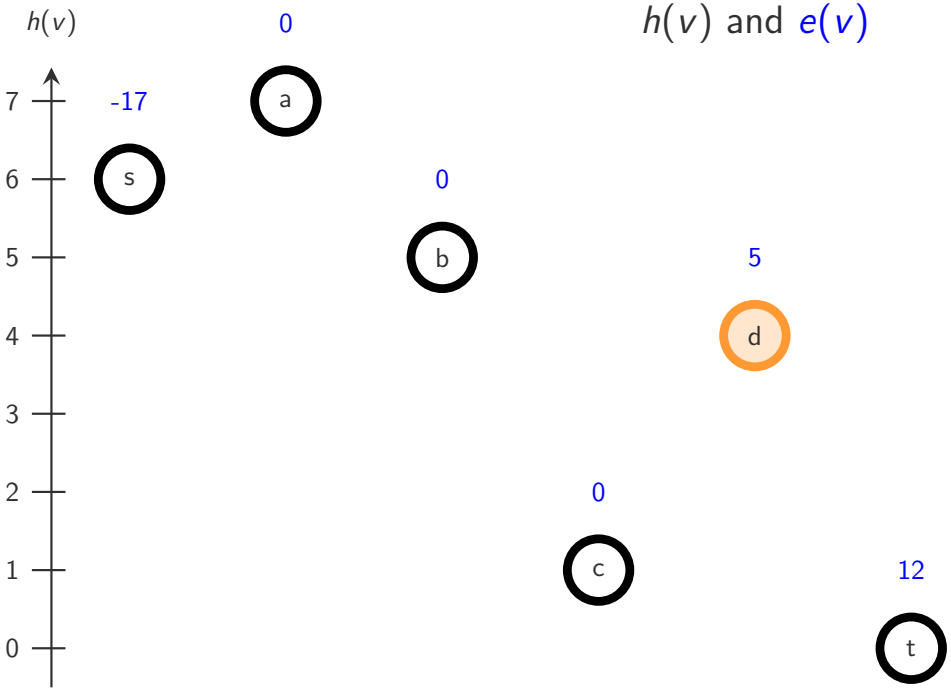
Edge available for push

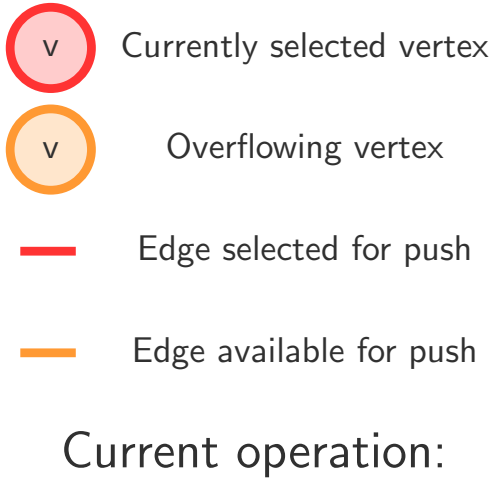
Current operation:

Push from b to d

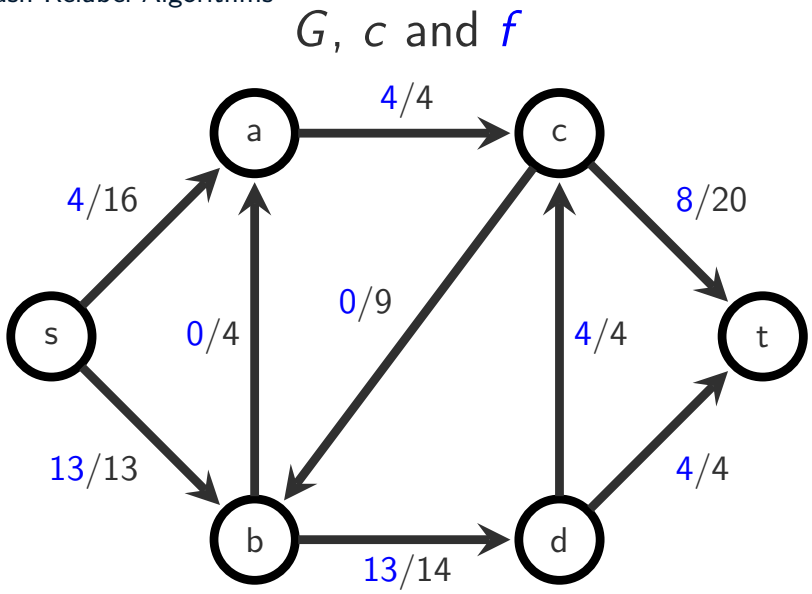


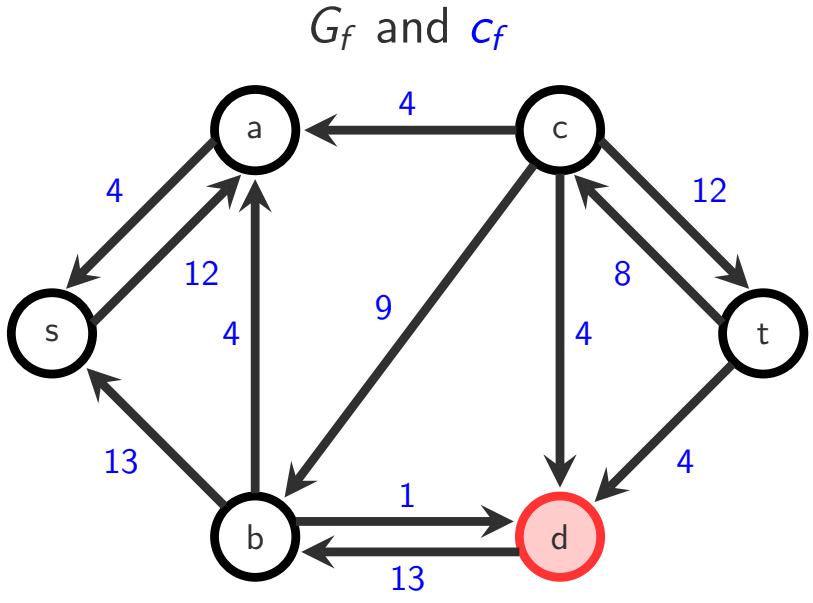


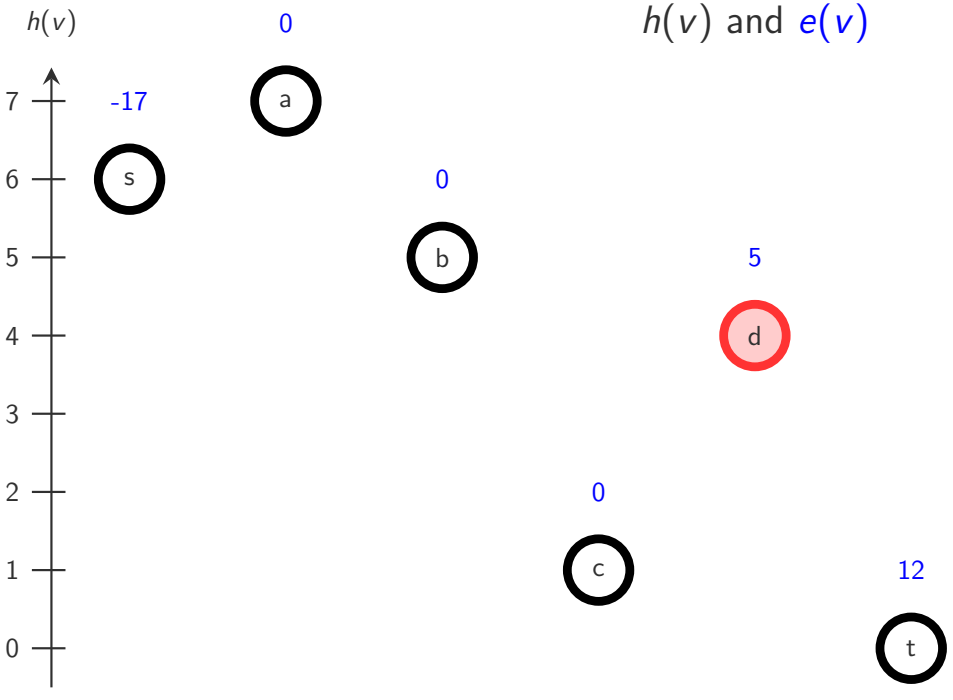


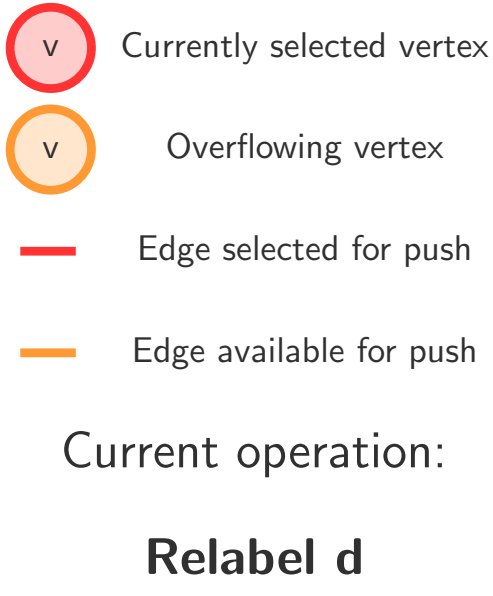


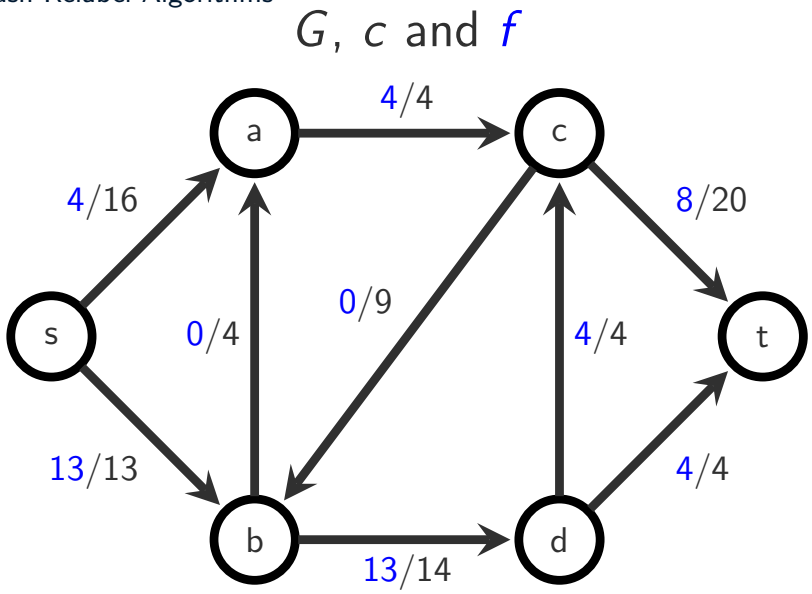
Current operation:

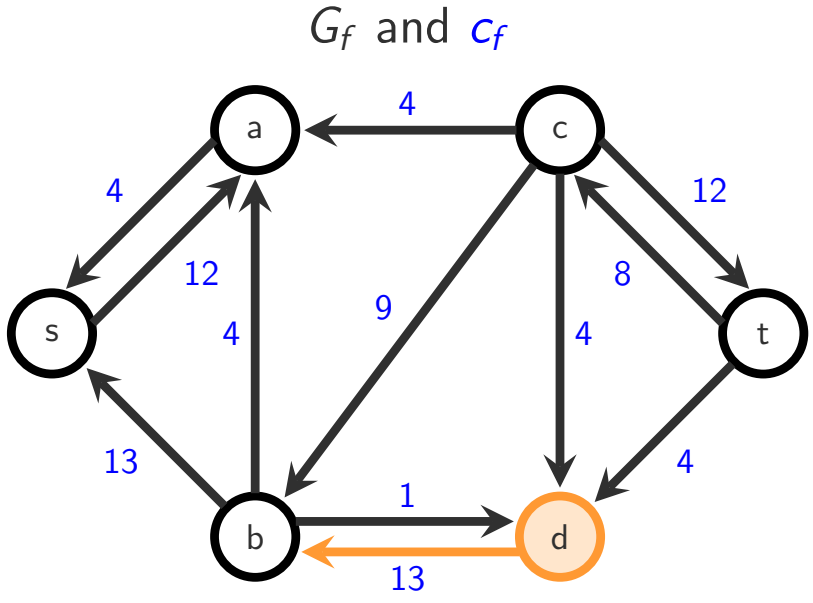

G, c and f
a
c
s
b
d
t
4/4
4/16
8/20
0/4
0/9
4/4
13/13
4/4
13/14

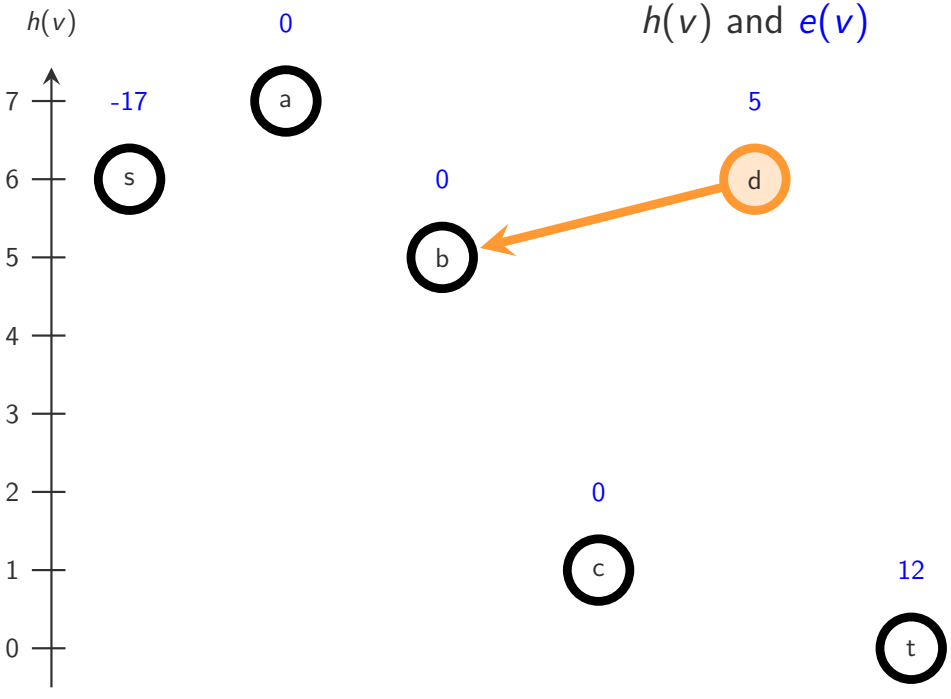

G_f and c_f
a
c
s
b
d
t
4
4
12
12
9
8
4
13
1
4
13

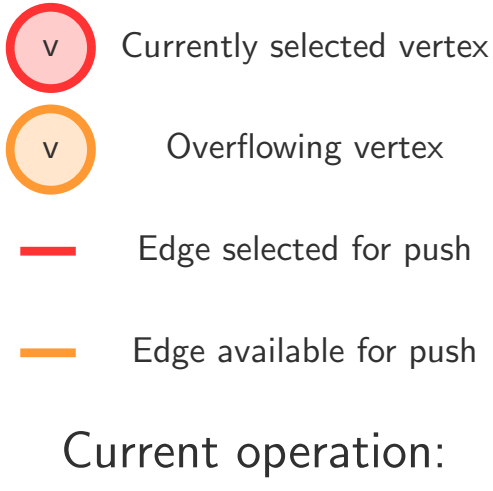

$h(v)$
$h(v)$ and $e(v)$
7
6
5
4
3
2
1
0
-17
0
0
5
0
12
s
a
b
d
c
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
Current operation:
Relabel d



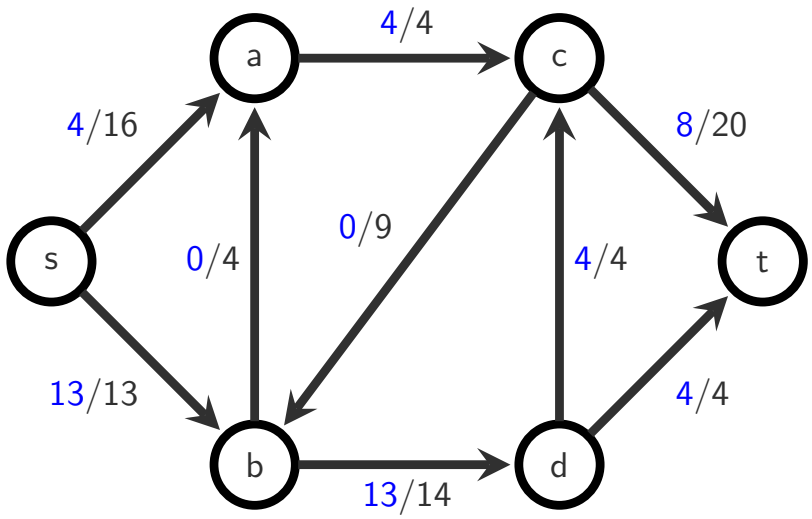




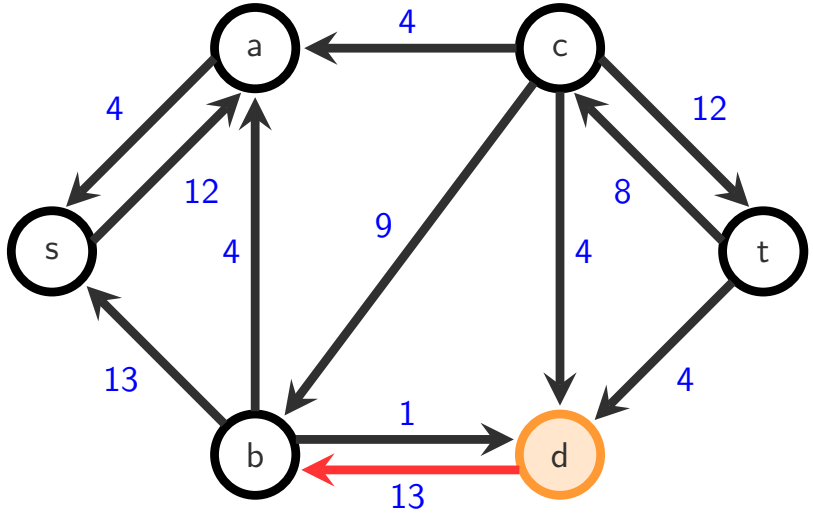


Current operation:

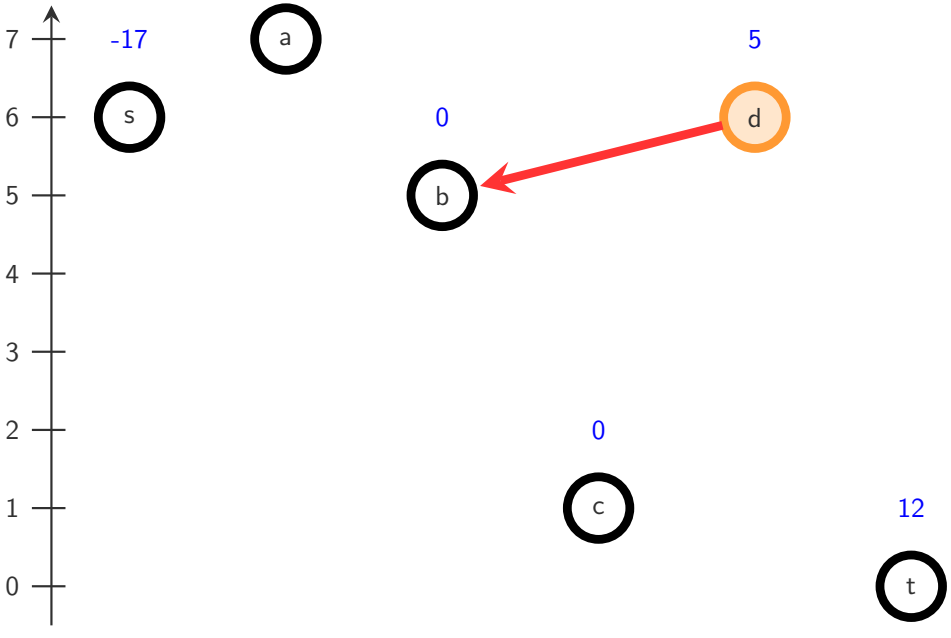
G, c and f

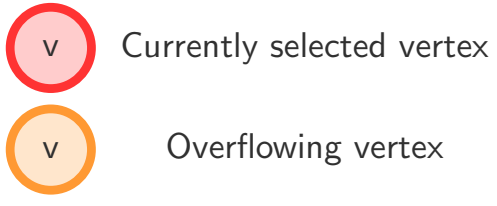


G_f and c_f



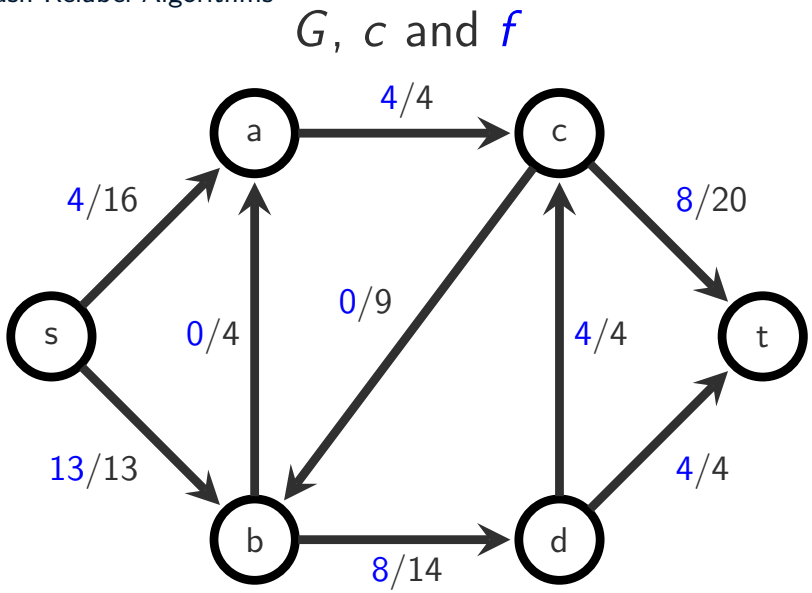
$h(v)$ and $e(v)$

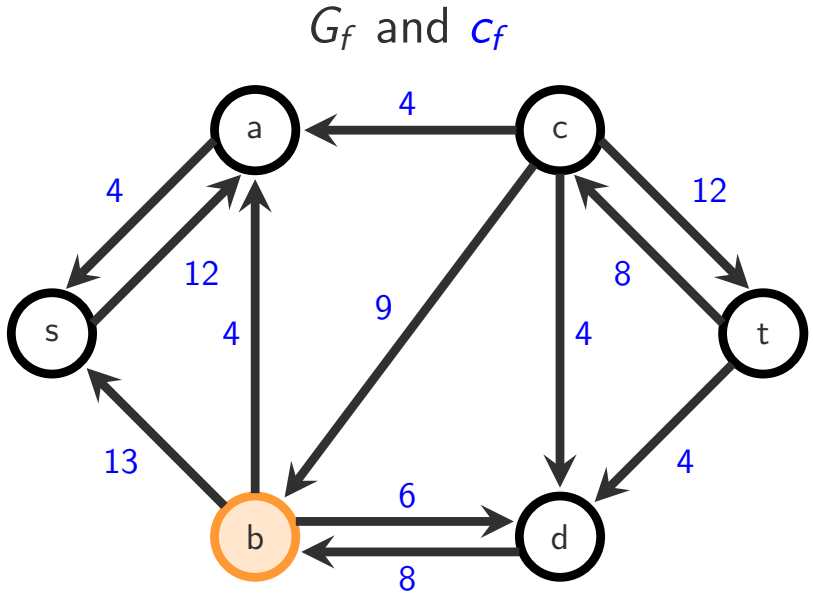


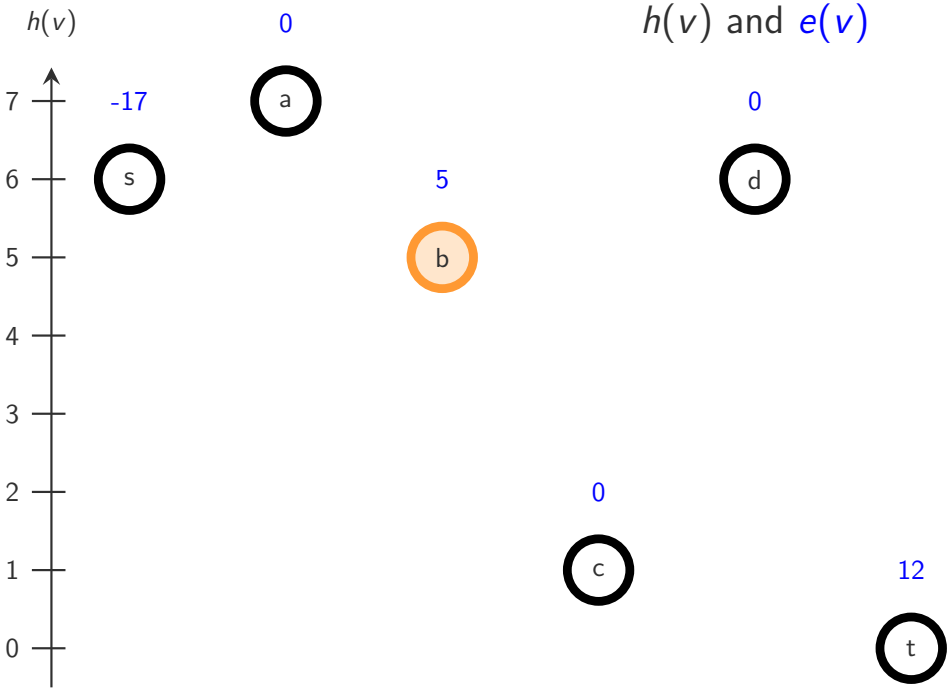


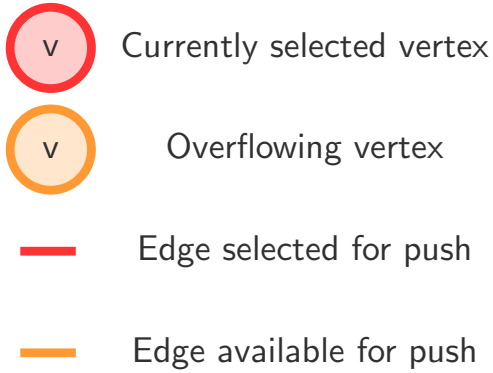
Current operation:

Push from d to b



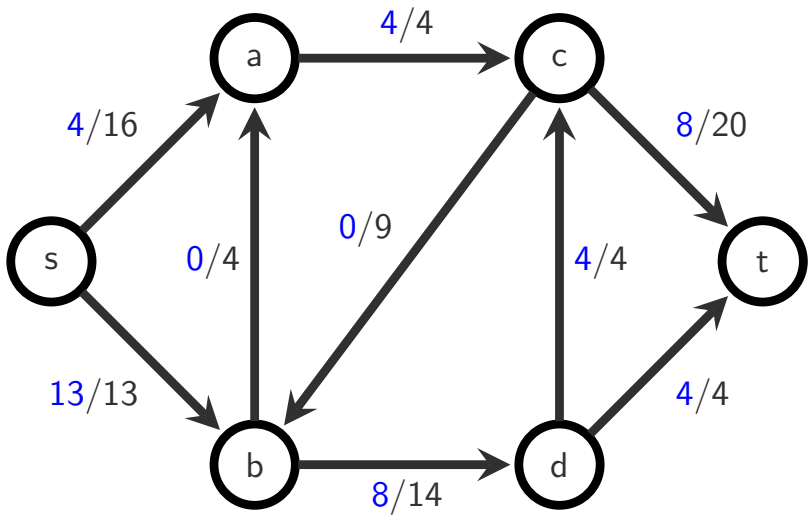




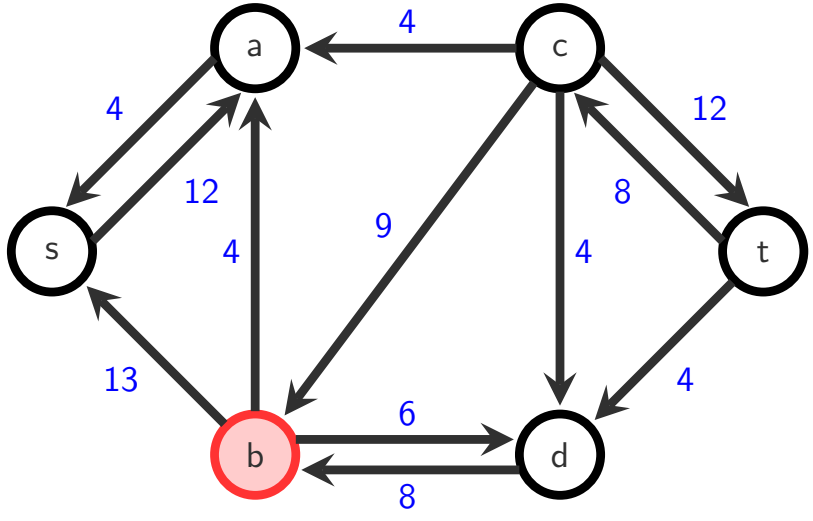


Current operation:

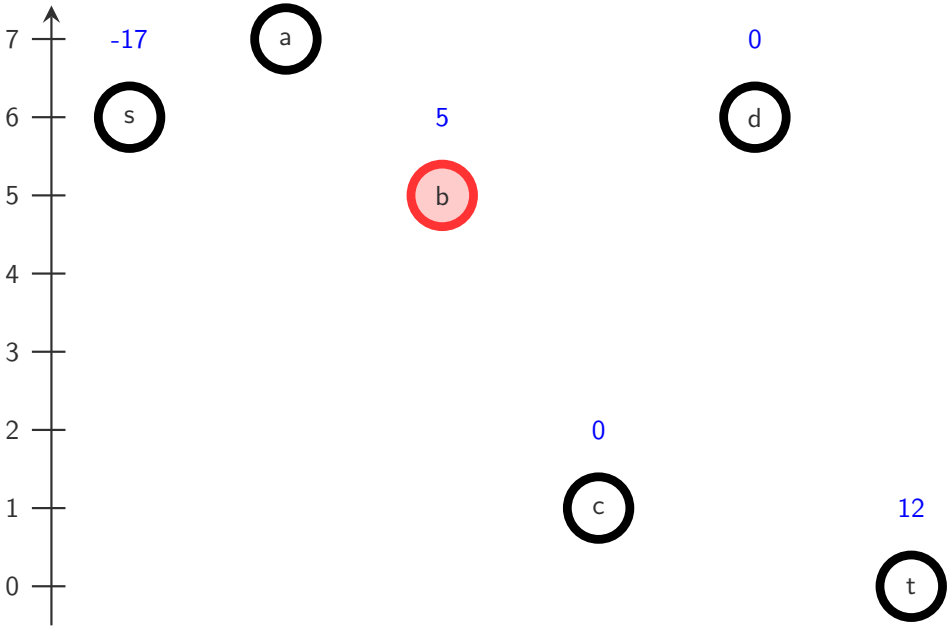
G, c and f

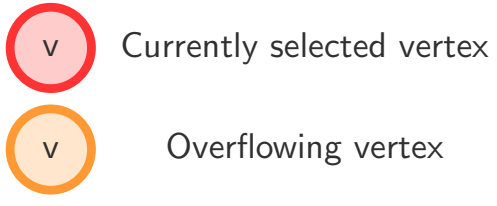


G_f and c_f



$h(v)$ and $e(v)$

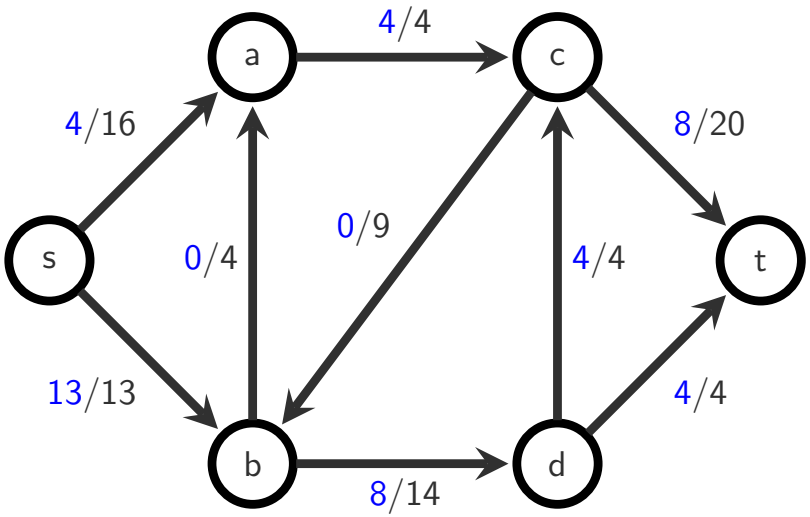




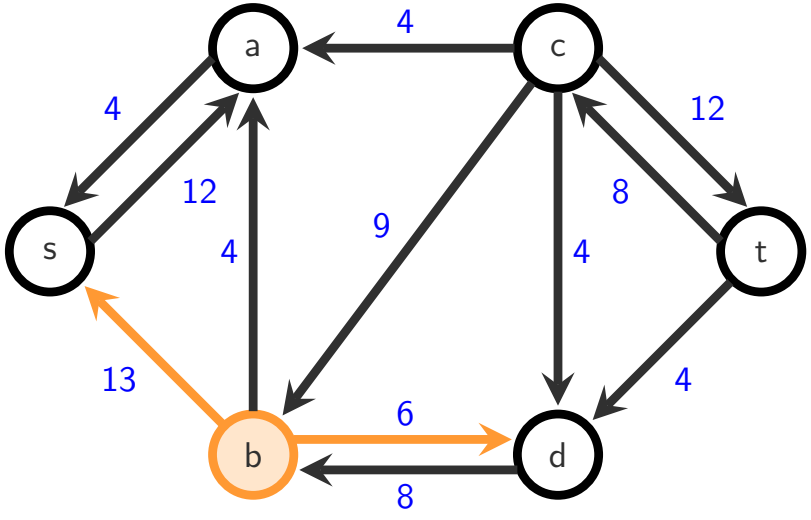
Current operation:

Relabel b

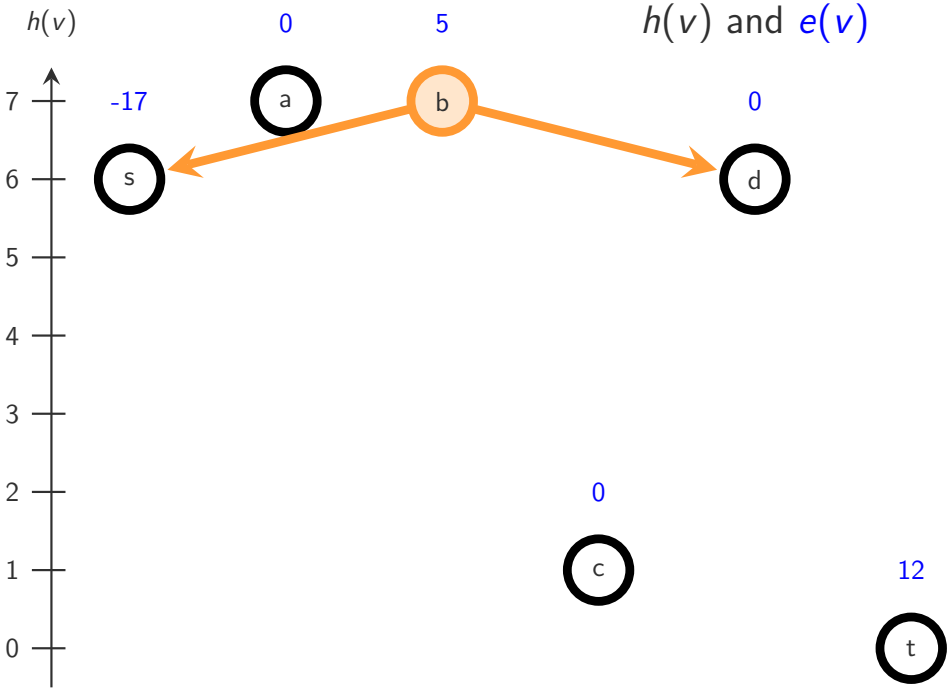
G, c and f



G_f and c_f



$h(v)$ and $e(v)$



v Currently selected vertex

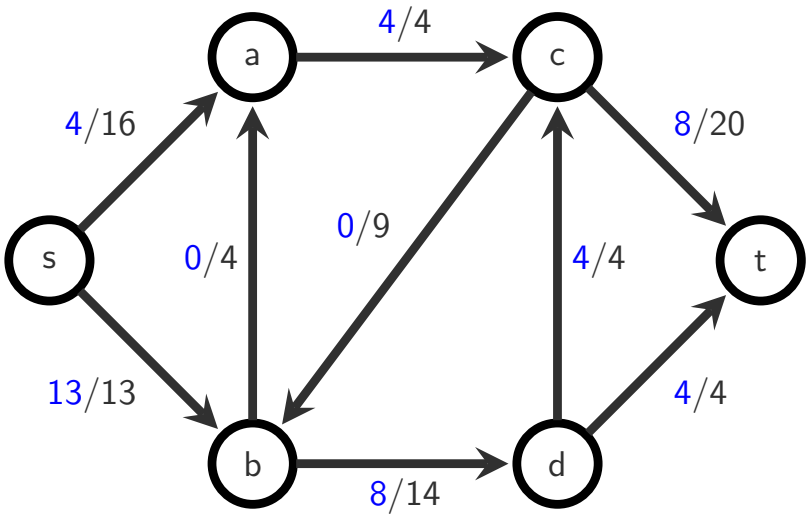
v Overflowing vertex

Edge selected for push

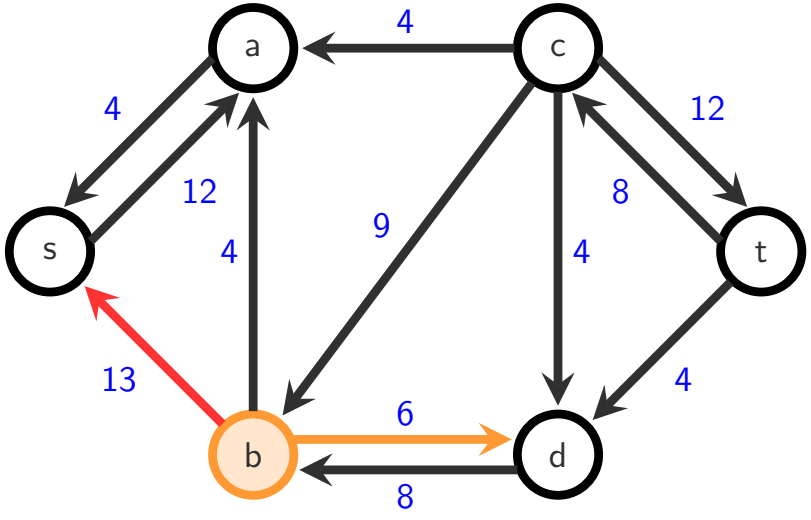
Edge available for push

Current operation:

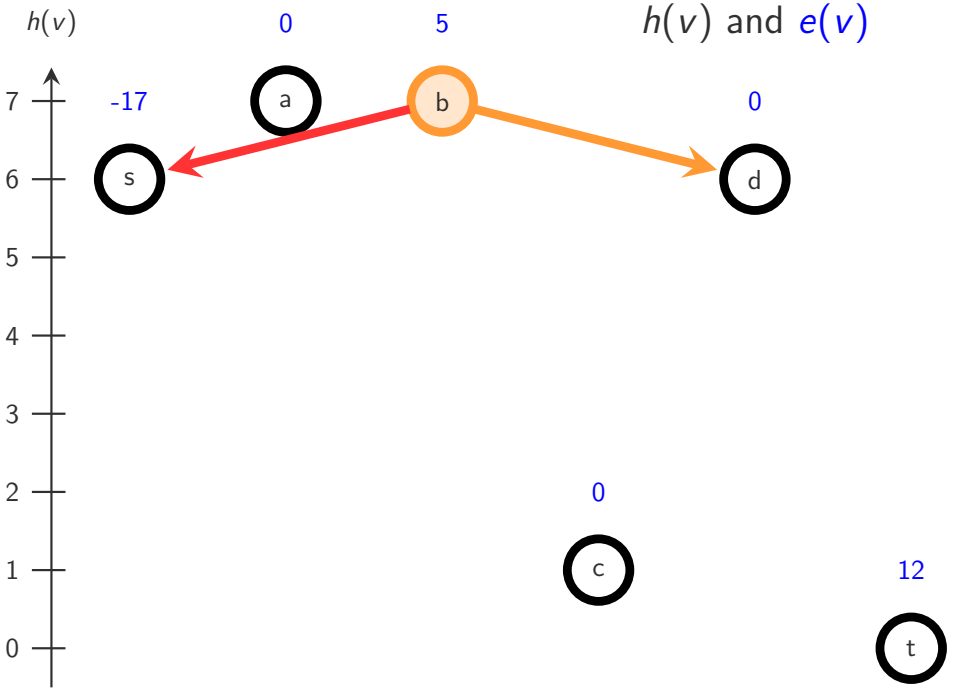
G, c and f



G_f and c_f



$h(v)$ and $e(v)$



v Currently selected vertex

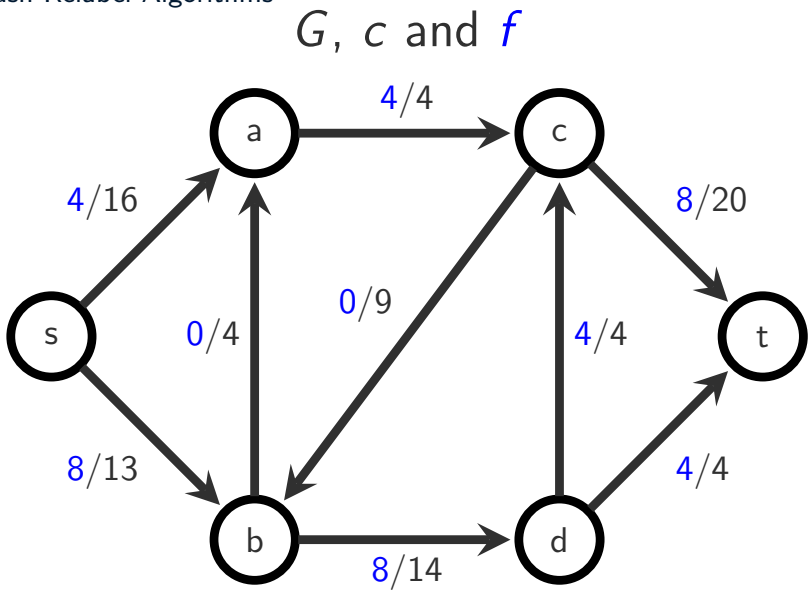
v Overflowing vertex

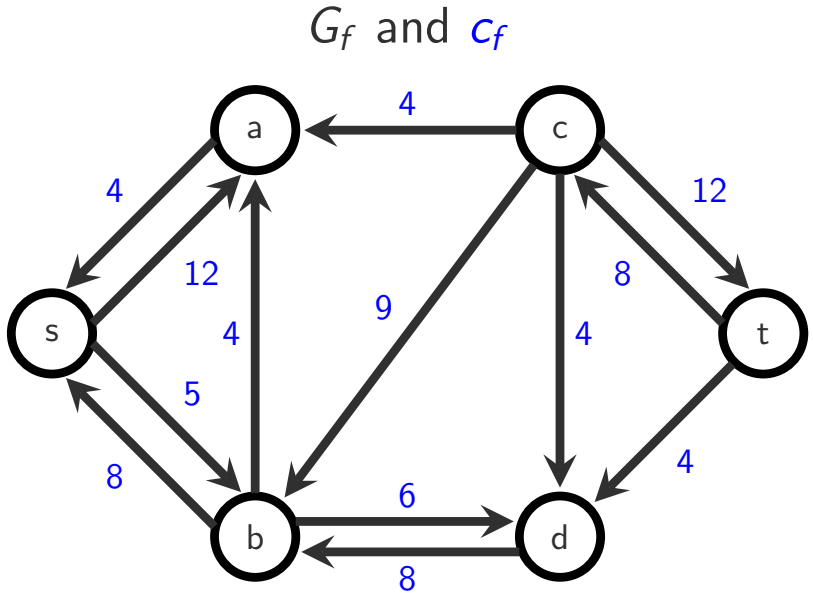
Edge selected for push

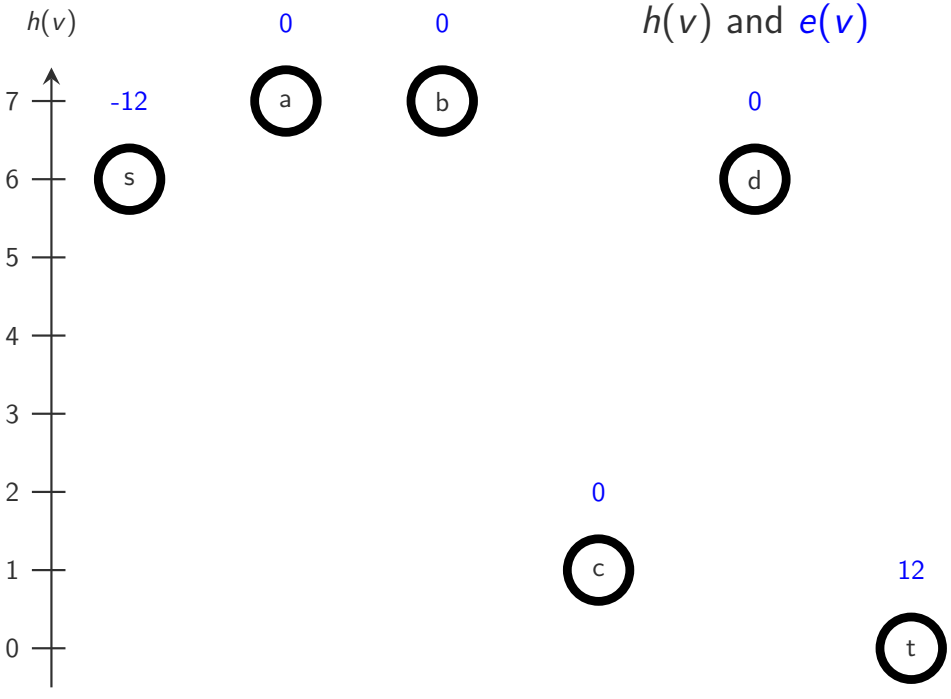
Edge available for push

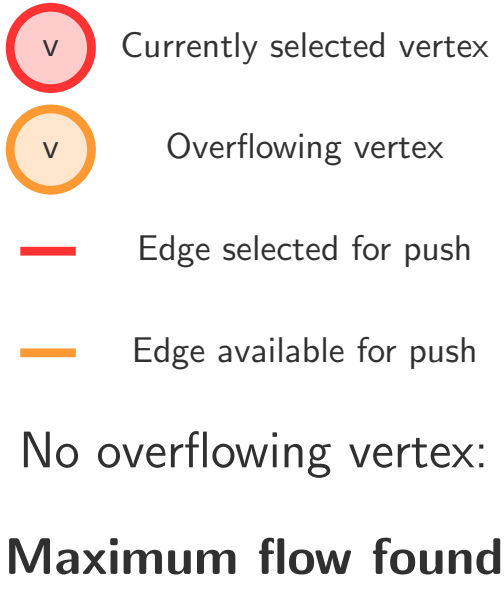
Current operation:

Push from b to s


G, c and f
a
c
s
t
b
d
4/4
4/16
8/20
0/4
0/9
4/4
8/13
4/4
8/14


G_f and c_f
a
c
s
t
b
d
4
4
12
12
4
9
8
4
5
8
6
4
8


$h(v)$
$h(v)$ and $e(v)$
0
0
-12
a
b
0
s
d
7
6
5
4
3
2
1
0
0
12
c
t


v
Currently selected vertex
v
Overflowing vertex
Edge selected for push
Edge available for push
No overflowing vertex:
Maximum flow found

Push-Relabel-Algorithm: Correctness

During execution,

- f is always a preflow
- h is always a height function
- every relabel operation on u strictly increases $h(u)$
- s is reachable from any active vertex in G_f
- if w is reachable from v in G_f with distance k, then $h(v) \leq h(w) + k$
- t is not reachable from s in G_f.

The last property implies that after execution (when there is no more overflowing vertex), we obtain a flow that has no augmenting path, i.e. a maximum flow.

Push-Relabel-Algorithm: Analysis

One can show:

- $\mathcal{O}(|V|^2)$ relabel operations
- $\mathcal{O}(|V|\,|E|)$ "saturating" push operations (where afterwards $f(e) = c(e)$)
- $\mathcal{O}(|V|^2\,|E|)$ "non-saturating" push operations. This can be improved (see next slide).

Push-Relabel-Algorithm: Analysis

Keep list of overflowing vertices in appropriate data structure and update accordingly after each operation!
Order for choosing next overflowing vertex?

- Any order (e.g. with stack): Goldberg-Tarjan algorithm, $\mathcal{O}(|V|^2 |E|)$.
- FIFO (with a queue): $\mathcal{O}(|V|^3)$.
- Highest label (with buckets): $\mathcal{O}(|V|^2 \sqrt{|E|})$.

Note: Having $\mathcal{O}(|V|^2 \sqrt{|E|})$ push-operations is not enough for a total complexity of $\mathcal{O}(|V|^2 \sqrt{|E|})$. Need amortized constant-time for each push operation!

Push-Relabel-Algorithm: Achieving $\mathcal{O}(|V|^2\sqrt{|E|})$

- Note: any active vertex v can reach s in G_f using at most $n-1$ edges, so $h(v) \leq h(s) + n - 1 = 2n - 1$.
  Hence, can store active vertices in $2n-1$ buckets $V_0, V_1, \ldots, V_{2n-1}$, where V_i contains vertices of height i (implement e.g. using array of doubly-linked lists).
  Maintain index i as highest index to non-empty bucket V_i
  - Initialize with $i = 0$
  - Increase during relabel
  - After push, if $V_i = \emptyset$: decrease i until $V_i \neq \emptyset$
- Moreover, for each $v \in V$, store admissible edges leaving v in doubly-linked list A_v and update during push/relabel operations.

Heuristics for the Push-Relabel-Algorithm

Two-phase algorithm

- In first phase, only push/relabel vertices with $h(v) < |V|$.
- Does not compute complete flow, but value of maximum flow at t.
- Remaining excess flow may be pushed back to s in second phase.

Initial labeling heuristic

- Compute initial heights as minimal distance to t by backwards BFS, computing $h(v) \leftarrow d_G(v, t)$.
- Avoids unnecessary initial relabeling operations.
- Can also compute labeling for second phase with $h(v) \leftarrow d_{G_f}(v, s) + |V|$.

Heuristics for the Push-Relabel-Algorithm

Gap heuristic

- After each relabeling, check if there is a height k with $0 < k < |V|$ such that there is no vertex v with $h(v) = k$ (keep a count array).
- If yes, all vertices u with $k < h(u) < |V|$ are disconnected from t in G_f and can be disregarded (set $h(u) \leftarrow |V|$).
- One of the most efficient heuristics, crucial for improving the performance.

Further reading

Several improved algorithms available:

- Orlin: Max flows in $\mathcal{O}(nm)$ time, or better, 2013: $\mathcal{O}(|V|\,|E|)$
- Sidford and Lee: Path-Finding Methods for Linear Programming, 2014: $\mathcal{O}(|E|\sqrt{|V|}(\log|V|)^{\mathcal{O}(1)}(\log U)^2)$

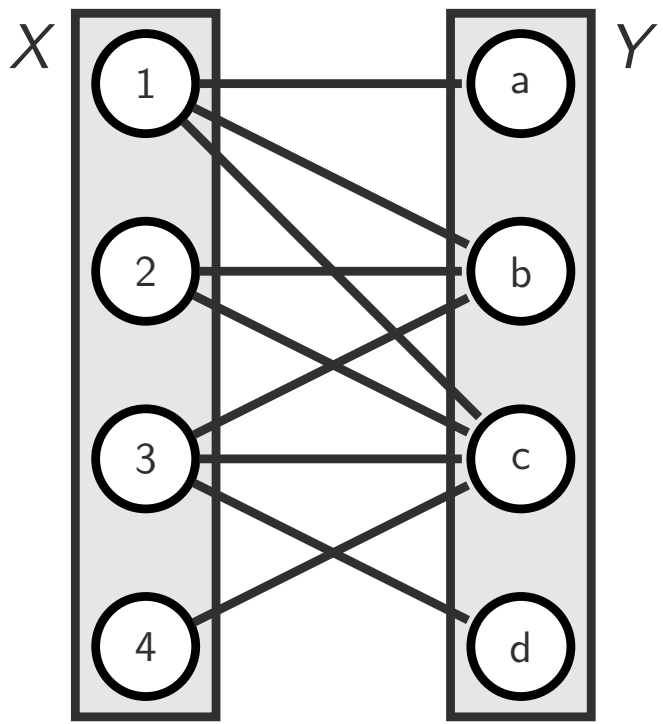
Additional literature on flow problems and algorithms:

- T. H. Cormen et al.: Introduction to Algorithms. MIT press, 2009.
- R. Ahuja, T. Magnanti and J. B. Orlin: Network Flows: Theory, Algorithms and Applications. Prentice Hall, 1993.
- B. Korte, J. Vygen: Combinatorial Optimization: Theory and Algorithms. Springer, 2012.

Application: Bipartite Matching

Definition (Bipartite Matching / Maximum Matching Problem)

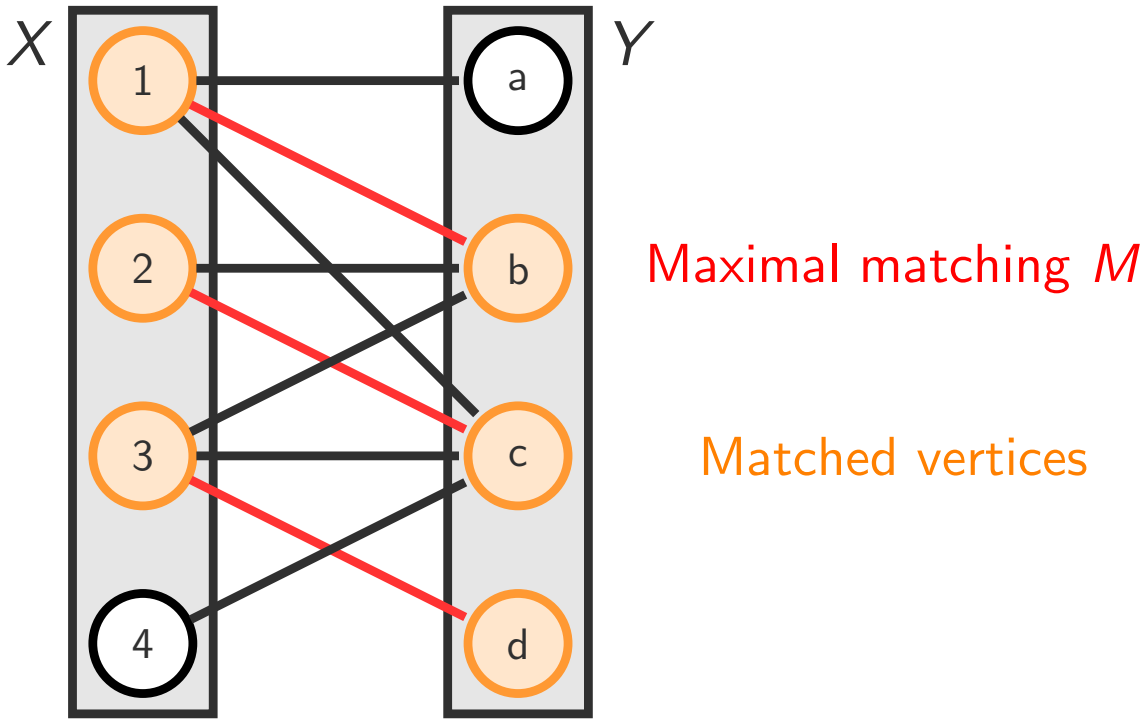
Given two disjoint sets of vertices X and Y and a set of edges $E \subseteq X \times Y$, a *matching* $M \subseteq E$ is a subset of edges such that each node of $X \cup Y$ appears in at most one edge of M. The *maximum matching problem* is finding a matching M such that $|M|$ is maximal.



Application: Bipartite Matching

Definition (Bipartite Matching / Maximum Matching Problem)

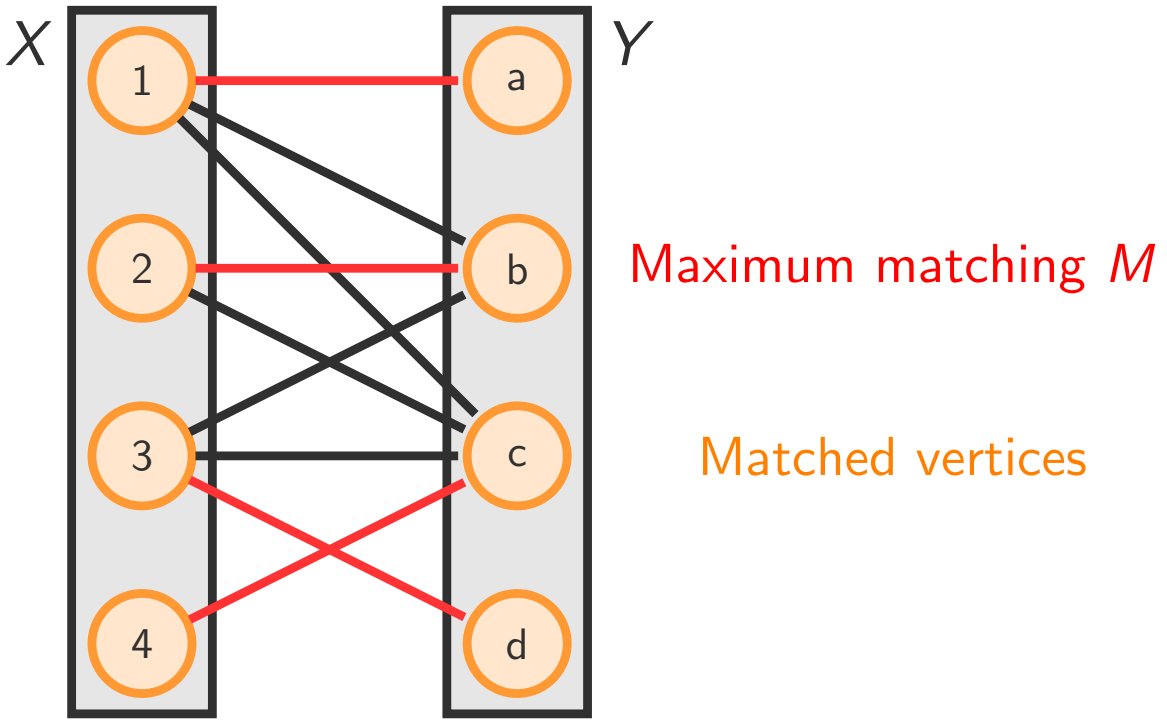
Given two disjoint sets of vertices X and Y and a set of edges $E \subseteq X \times Y$, a *matching* $M \subseteq E$ is a subset of edges such that each node of $X \cup Y$ appears in at most one edge of M. The *maximum matching problem* is finding a matching M such that $|M|$ is maximal.



Application: Bipartite Matching

Definition (Bipartite Matching / Maximum Matching Problem)

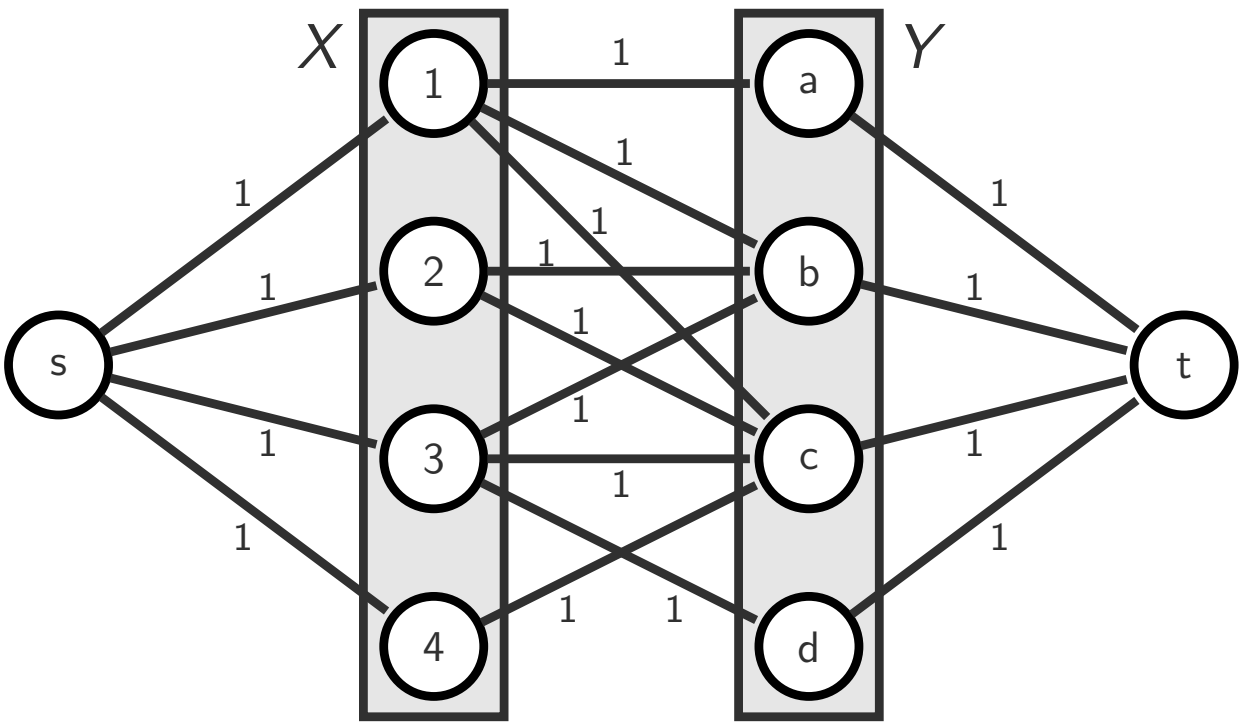
Given two disjoint sets of vertices X and Y and a set of edges $E \subseteq X \times Y$, a *matching* $M \subseteq E$ is a subset of edges such that each node of $X \cup Y$ appears in at most one edge of M. The *maximum matching problem* is finding a matching M such that $|M|$ is maximal.



Application: Bipartite Matching

Matching problem as a flow problem

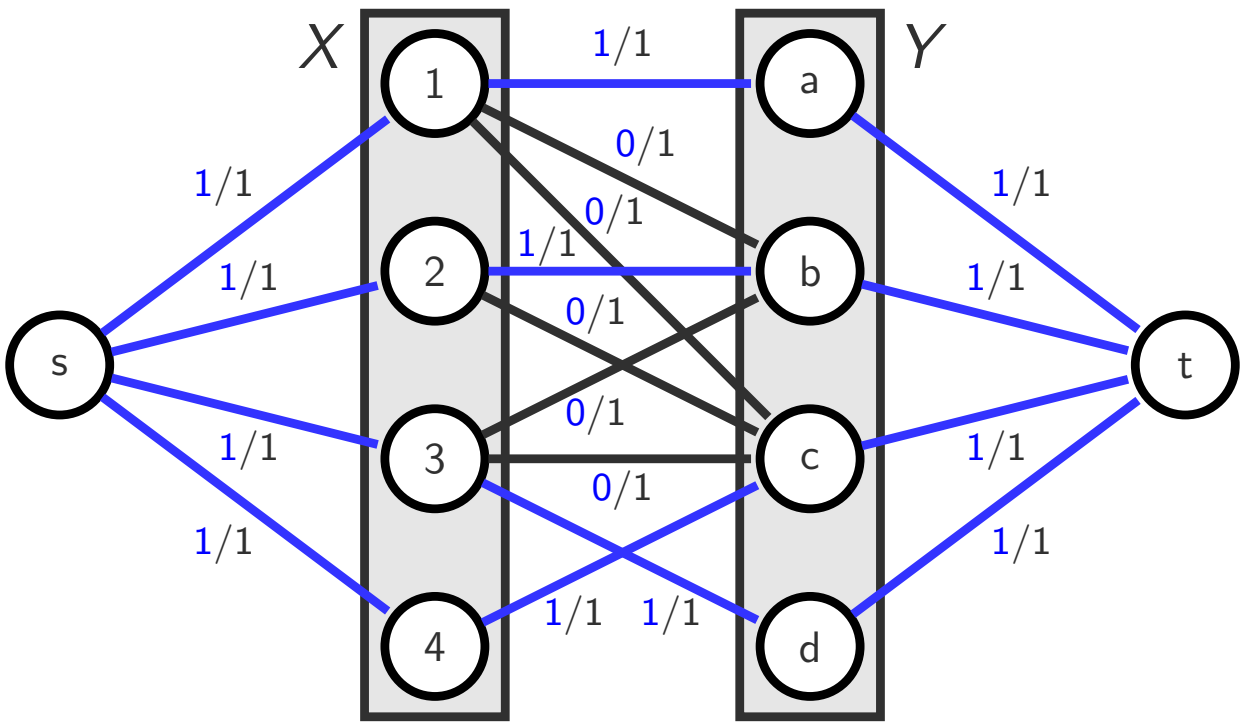
Given a bipartite matching problem, construct flow network $G = (V, E')$ with $V = \{s, t\} \cup X \cup Y$ and $E' = E \cup (\{s\} \times X) \cup (Y \times \{t\})$ and $c(e) = 1$ for all $e \in E'$. Then the value of the maximum flow is equal to the size of the maximum matching.



Application: Bipartite Matching

Matching problem as a flow problem

Given a bipartite matching problem, construct flow network $G = (V, E')$ with $V = \{s, t\} \cup X \cup Y$ and $E' = E \cup (\{s\} \times X) \cup (Y \times \{t\})$ and $c(e) = 1$ for all $e \in E'$. Then the value of the maximum flow is equal to the size of the maximum matching.



Choosing the Algorithm

Which Algorithm to choose?

- In general, Push-Relabel-Algorithm is best
- Better asymptotic complexity not always decisive — check the constraints
- Implement improvements as needed