ADS: Algorithmen und Datenstrukturen 2

Teil 7

Prof. Dr. Gerhard Heyer

Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

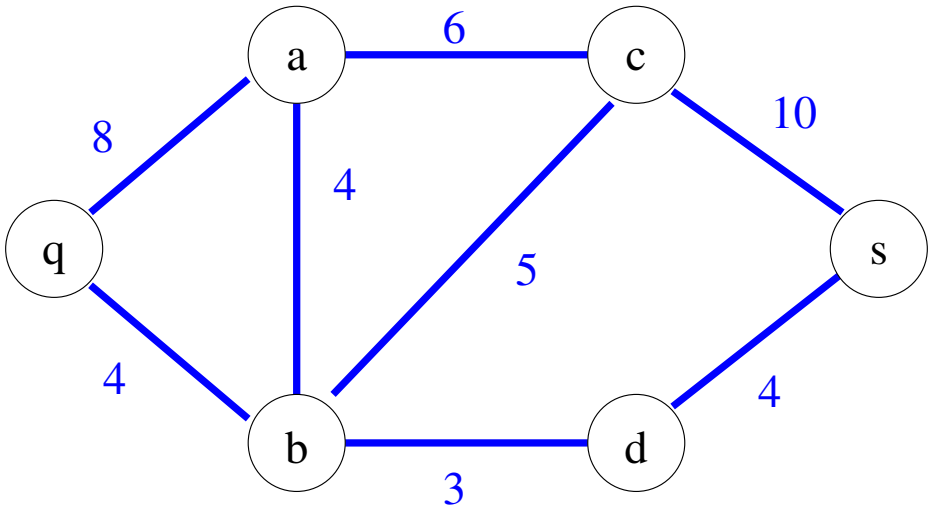
23. Mai 2018

[Letzte Aktualisierung: 27/06/2018, 13:08]

Flüsse in Netzwerken I

Anwendungsprobleme:

- Wieviel Autos können durch ein Straßennetz fahren?
- Wieviel Abwasser fasst ein Kanalnetz?
- Wieviel Strom kann durch ein Leitungsnetz fließen?

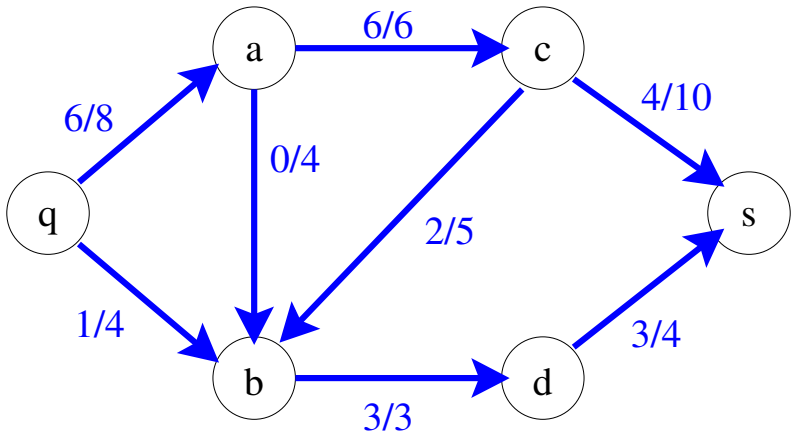


Kantenwert: Kapazität $c(e)$

Flüsse in Netzwerken II

- Ein (Fluss-) Netzwerk ist ein gerichteter Graph $G = (V, E, c)$ mit ausgezeichneten Knoten q (Quelle) und s (Senke), sowie einer Kapazitätsfunktion $c :\rightarrow \mathbb{R}^+$.
- Ein Fluss für das Netzwerk ist eine Funktion $f : E \rightarrow \mathbb{R}_0^+$, so dass gilt:
  - Kapazitätsbeschränkung: $\forall e \in E : 0 \leq f(e) \leq c(e)$.
  - Symmetriebedingung: $f(u, v) = -f(v, u)$
  - Flusserhaltung/Kirchhoffsches Gesetz: $\forall v \in V \setminus \{q, s\} : \sum_{(v',v)\in E} f(v', v) = \sum_{(v,v')\in E} f(v, v')$
  - Der Wert von f, $w(f)$, ist die Summe der Flusswerte der die Quelle q verlassenden Kanten: $\sum_{(q,v)\in E} f(q, v)$

Flüsse in Netzwerken III



Kantenbeschriftung: Fluss $f(e)$ / Kapazität $c(e)$

Gesucht: Fluss mit maximalem Wert

- begrenzt durch Summe der aus q wegführenden bzw. in s eingehenden Kapazitäten
- jeder weitere "Schnitt" durch den Graphen, der q und s trennt, begrenzt maximalen Fluss

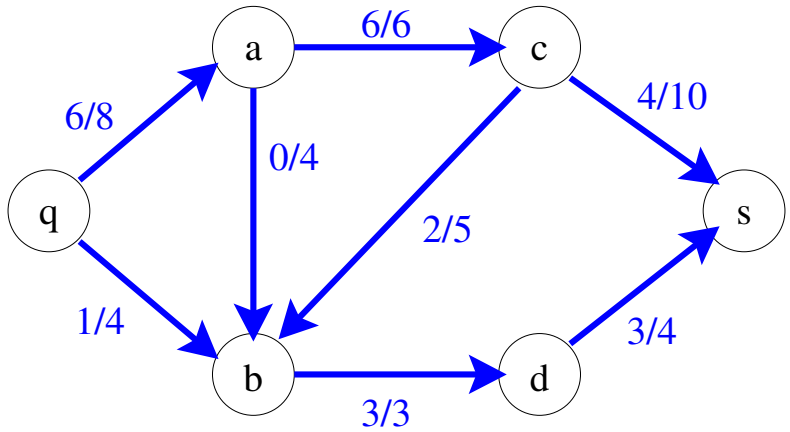
Restgraph

Sei f ein zulässiger Fluss für $G = (V, E, c)$. Sei
$E' = \{(v, w) | (v, w) \in E \vee (w, v) \in E\}$

- Wir definieren die Restkapazität einer Kante $e = (v, w) \in E'$ als

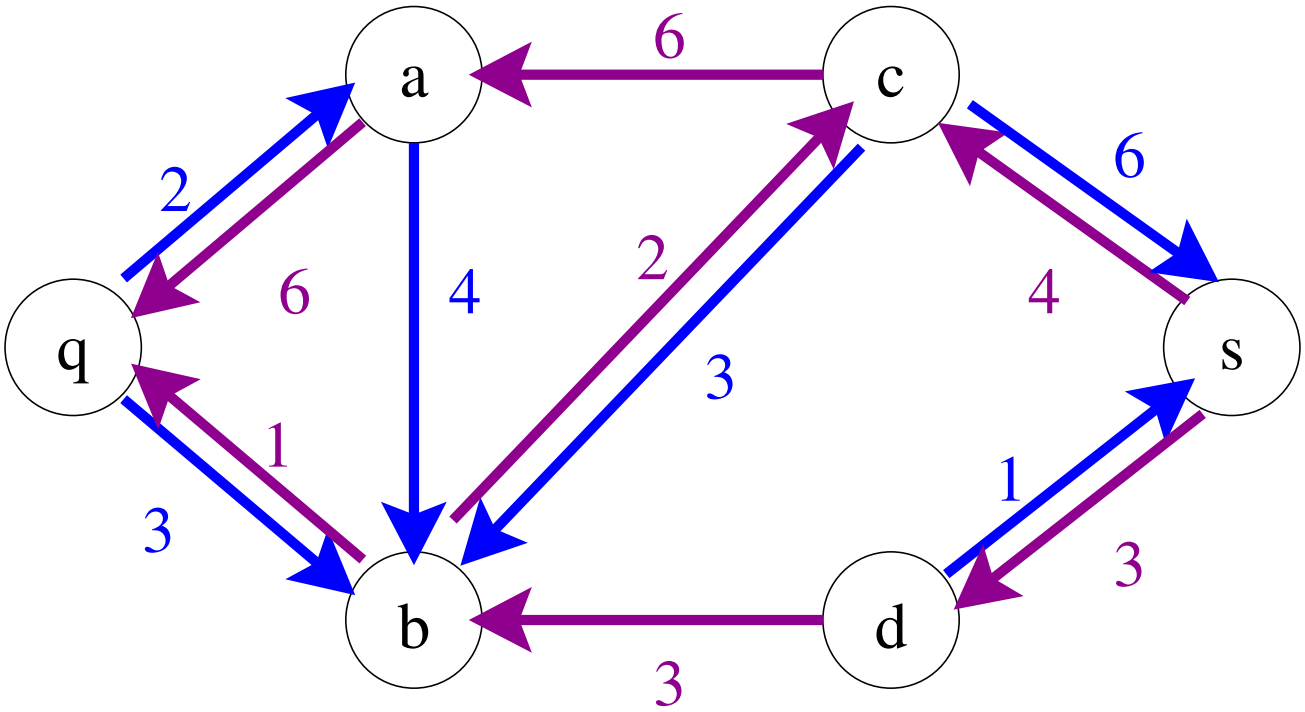
$$rest(e) := \begin{cases} c(e) - f(e) & \text{falls } e \in E \\ f(\bar{e}) & \text{falls } \bar{e} \in E \end{cases}$$

  wobei $\bar{e} = (w, v)$ die Rückwärtskante zu e bezeichne.
- Der Restgraph G_f von f (bzgl. G) besteht aus den Kanten $e \in E'$, für die $rest(e) > 0$



$f(e)/c(e)$

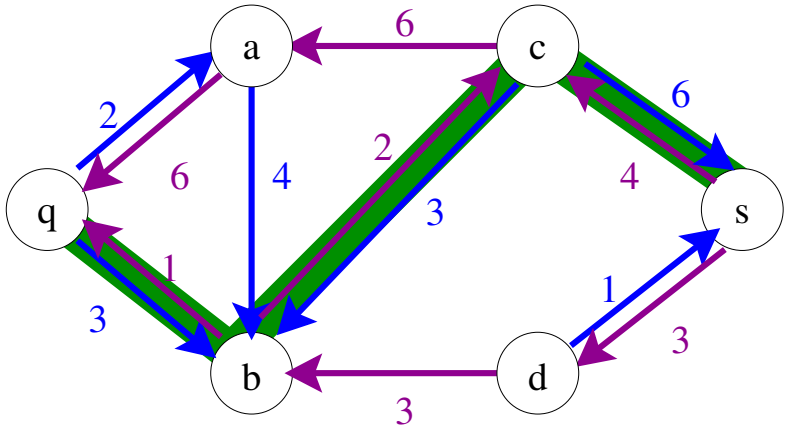
Restgraph



$rest(e)$

Erweiterungspfad

- Jeder gerichtete Pfad p von q nach s im Restgraphen heißt Erweiterungspfad.

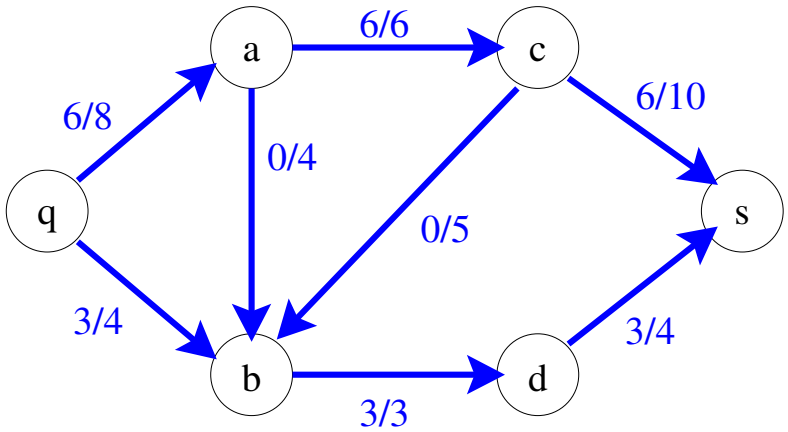


Ein *Erweiterungspfad* ist ein einfacher (zyklenfreier) Pfad p in G_f von q nach s.
Die *Restkapazität* von p ist $c_f(p) = min\{(c_f(u,v)|(u,v) \text{ liegt auf } p\}$

Erweiterungspfad

Beobachtung:
Sei G ein Netzwerk, c eine Kapazität und f ein Fluss. Sei G_f das zugehörige Restnetzwerk und c_f die zugehörige Restkapazität. Sei ferner g ein zulässiger Fluss auf G_f. Dann gilt: $(f + g)$ ist ein zulässiger Fluss auf G mit $|(f + g)| = |f| + |g|$

- Optimierung des Flusses für G:
  - entlang eines zunehmenden Weges p
  - Verbesserung um minimales $rest(e)$ aller e auf dem Weg p



Ford-Fulkerson-Algorithmus

Optimierung eines bestehenden Flusses f in einem Netzwerk G mit Kapazität c:

- bestimme Restnetzwerk G_f
- suche Erweiterungspfad
- bestimme $c_f(p)$
- erhöhe den Fluss um Minimum der verfügbaren Restkapazität der einzelnen Kanten des Erweiterungspfades

```
Solange es einen zunehmenden Weg p in Restgraph G gibt
   r = min {rest(e) | e liegt auf p}
   Fuer alle e = (v,w) auf p tue
      Falls e in E dann
         f(e) = f(e) + r
      Sonst f(w,v) = f(w,v) - r
```

Ford-Fulkerson-Algorithmus — Laufzeit

- $O(|E|)$ für Konstruktion des Restgraphen
- $O(|E|)$ für Konstruktion eines Erweiterungspfades: Graphtraversierung und Bestimmung des Flusses entlang des Wegs
- in jedem Schritt verbessert sich der Fluss um $r = \min\{rest(e) | e \text{ liegt auf } p\}$

Für *ganzzahlige* Kapazitäten: $O(E) \times \max f$ (jeder Durchlauf erhöht den Fluss um mindestens eins, also gibt es bis zu $|f^*|$ Durchläufe
(f* maximaler Fluss)

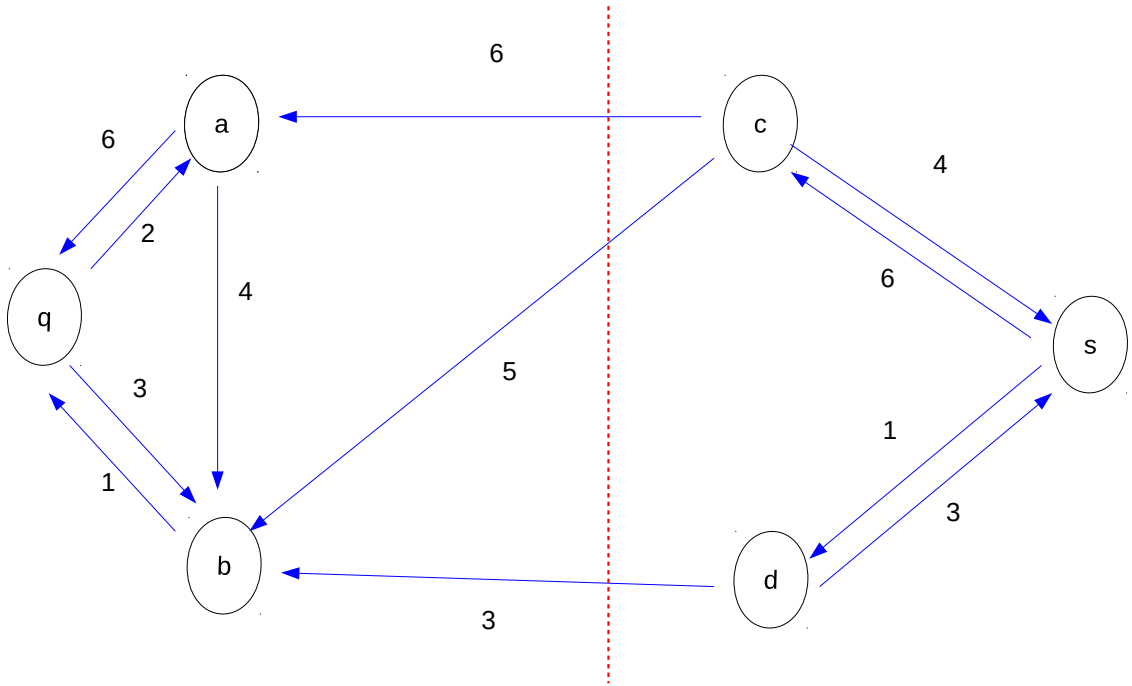
$\Rightarrow$ Für *allgemeine* Kapazitäten: Konvergenz nicht garantiert!

Min-Cut

Wenn ein Erweiterungspfad gefunden werden kann, bringt der neue Fluss $(f + g)$ eine Verbesserung.

Aber kann immer ein Erweiterungspfad gefunden werden, wenn der bisherige Fluss f nicht maximal ist?

Beobachtung: Im Graph mit optimalem Fluss existiert im Restgraph kein Weg von Quelle zu Senke!



Min-Cut-Max-Flow-Theorem

Es sei V ein Graph bestehend aus der Menge der Kanten $e = (u, v)$ mit $u \in A$ und $v \in B$. Wir definieren *Schnitt* (A, B) mit $A \cap B = \emptyset$, $A \neq \emptyset$, $B \neq \emptyset$, and $A \cup B = V$,
Die Kapazität von (A, B) is $c(A, B) = \sum_{e \in (A,B)} c(e)$.

Theorem (Min-Cut-Max-Flow-Theorem): Sei f ein zulässiger Fluss für G. Dann sind folgende Aussagen äquivalent:

1. f ist maximaler Fluss in G.
2. Der Restgraph von f enthält keinen zunehmenden Weg.
3. $w(f) = c(A, B)$ für einen Schnitt (A, B) von G.

Min-Cut-Max-Flow-Theorem – Beweis

(1) Sei (A, B) ein Schnitt mit $q \in A$ und $s \in B$. Flusserhaltung impliziert, dass $\sum_{e \in (A,B)} f(e) = w(f)$. Dies gilt für jeden Schnitt, der q und s trennt. Da $f(e) \leq c(e)$ erfüllt sein muss, gilt ausserdem $w(f) \leq c(A, B)$ für alle Schnitte die q und s trennen.

(2) ((1) $\Rightarrow$ (2)) Sei f ein maximaler Fluss. Würde der zugehörige Restgraph G_f einen zunehmenden Weg enthalten enthalten, könnte der Fluss weiter erhöht werden. Also gibt es keinen zunehmenden Weg.

(3) ((2) $\Rightarrow$ (3)) Sei A die Menge der Knoten, die in G_f von q aus erreicht werden können, und sei $B = V \setminus A$. Also gilt $q \in A$ und insbesonders $s \in B$. Zudem ist (A, B) ein Schnitt, der q und s trennt.
Es gibt keinen Weg in G_f durch Kanten von $c(A, B)$. (Wenn $e = (u, v) \in (A, B)$ existieren würde, wäre ja nicht nur u von q erreichbar sondern auch v, was der Definition von B widersprechen würde.) Daher gilt für die Restkapaziät entlang jeder Kante von $e \in c(A, B)$, dass $rest(e) = c(e) - f(e) = 0$ verschwindet. Also ist $f(e) = c(e)$ für alle $e \in c(A, B)$.
Wegen (1) gilt $w(f) = \sum_{e \in (A,B)} f(e) = \sum_{e \in (A,B)} c(e) = c(A, B)$.

(4)((3) $\Rightarrow$ (1)) Für alle zulässigen Flüsse f und Schnitte (A, B), die q und s trennen, gilt nach (1) $w(f) \leq c(A, B)$. Wenn also $w(f) = c(A, B)$ ist, dann ist $w(f)$ notwendigerweise maximal.

Push & Relabel - Goldberg-Tarjan 1988

Lokale Berechnung eines max flow von Quelle s zu Senke t
(Abweichend vom Vorherigen werden im nachfolgenden Beispiel s (source) und t (target) für Quelle und Senke verwendet.

Pre-Fluss: Verallgemeinerung der Fluss-Begriffs so, dass der Zufluss in einen Knoten mindestens so gross wie der Abfluss ist.
Der **Exzess** $ex(x)$ an einem Knoten, $x \neq s, t$ ist der Unterschied von Zufluss und Abfluss. $ex(x) \geq 0$ in einem Pre-Fluss.

Idee: Beginne mit dem Ausfluss an der Quelle s und schiebe den Exzess schrittweise in Richtung Senke t.

Problem: "Buchhaltung", damit immer ein Gefälle gewährleistet ist und Fluss nicht im Kreis verschoben wird.
Distanz- oder Höhenfunktion $h(x)$ mit $h(s) = |V|$, $h(t) = 0$, und $h(u) \geq h(v) + 1$ für jede Kante (u, v) entlang der Fluss noch verschoben werden kann.

Wie gehabt: Restgraph mit $rest(u, v) = c(u, v) - f(u, v)$ für $e \in E$ und $rest(v, u) = f(u, v)$ für "Rückwärtskanten"

Push & Relabel

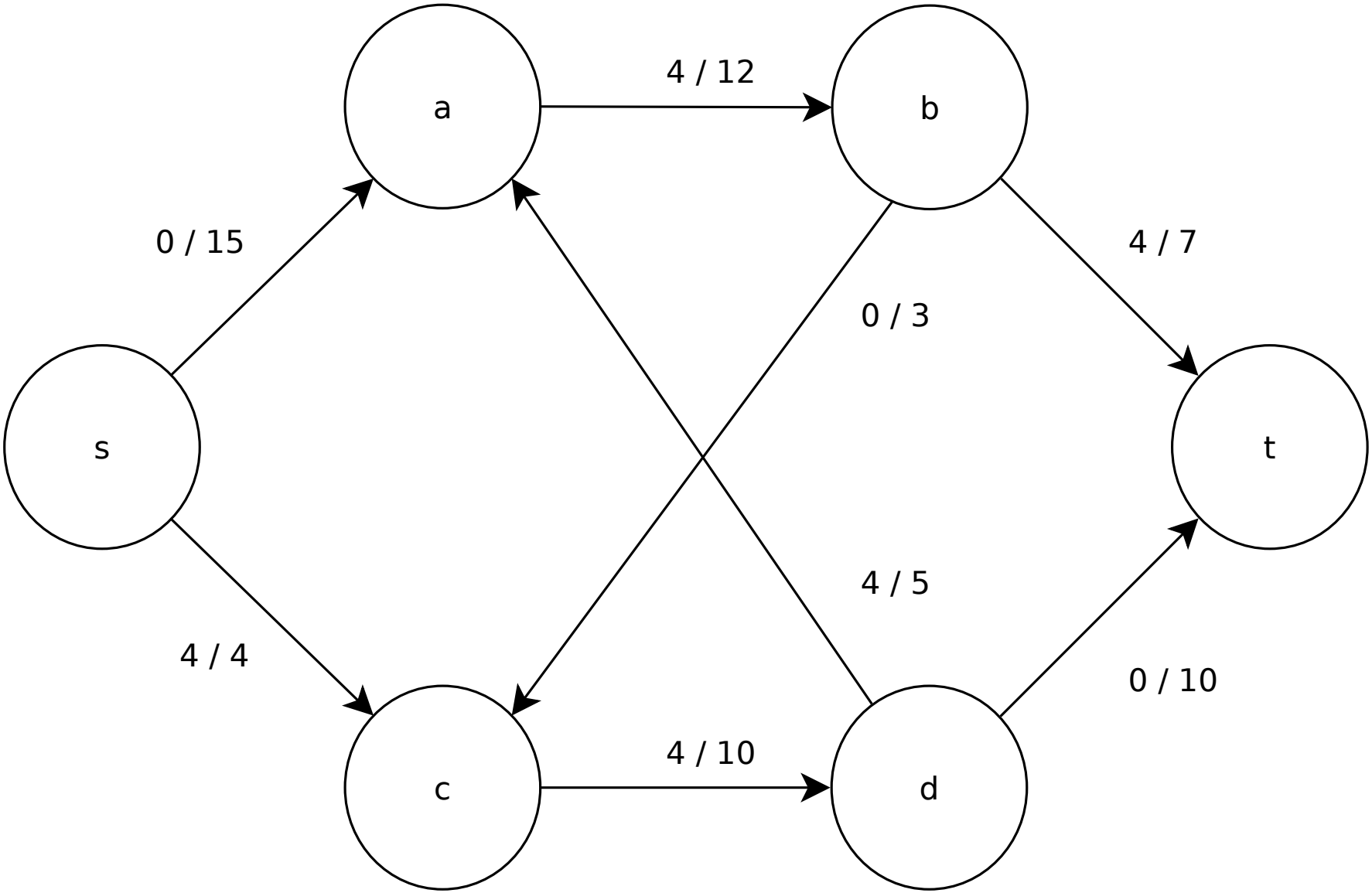
1. Initialisiere Graph
   - ex, Restgraph, h (wobei die Bedingung h(u)¡=h(v)+1 nicht für en initialen Push gilt)
2. Bestimme aktiven Knoten und erlaubte Kanten
   - erhöhe h (relabel), falls erforderlich
3. Push Fluss von aktivem Knoten über erlaubte Kanten
   - passe ex an (aus Pre-Fluss)
   - passe Rerstgraph an
4. Iteriere (2) und (3) bis ex(v) = 0 für alle $v \in V$

Push & Relabel

Erlaubte Kante: $rest(u, v) > 0$ und $h(u) = h(v) + 1$
Aktiver Knoten: $ex(x) > 0$

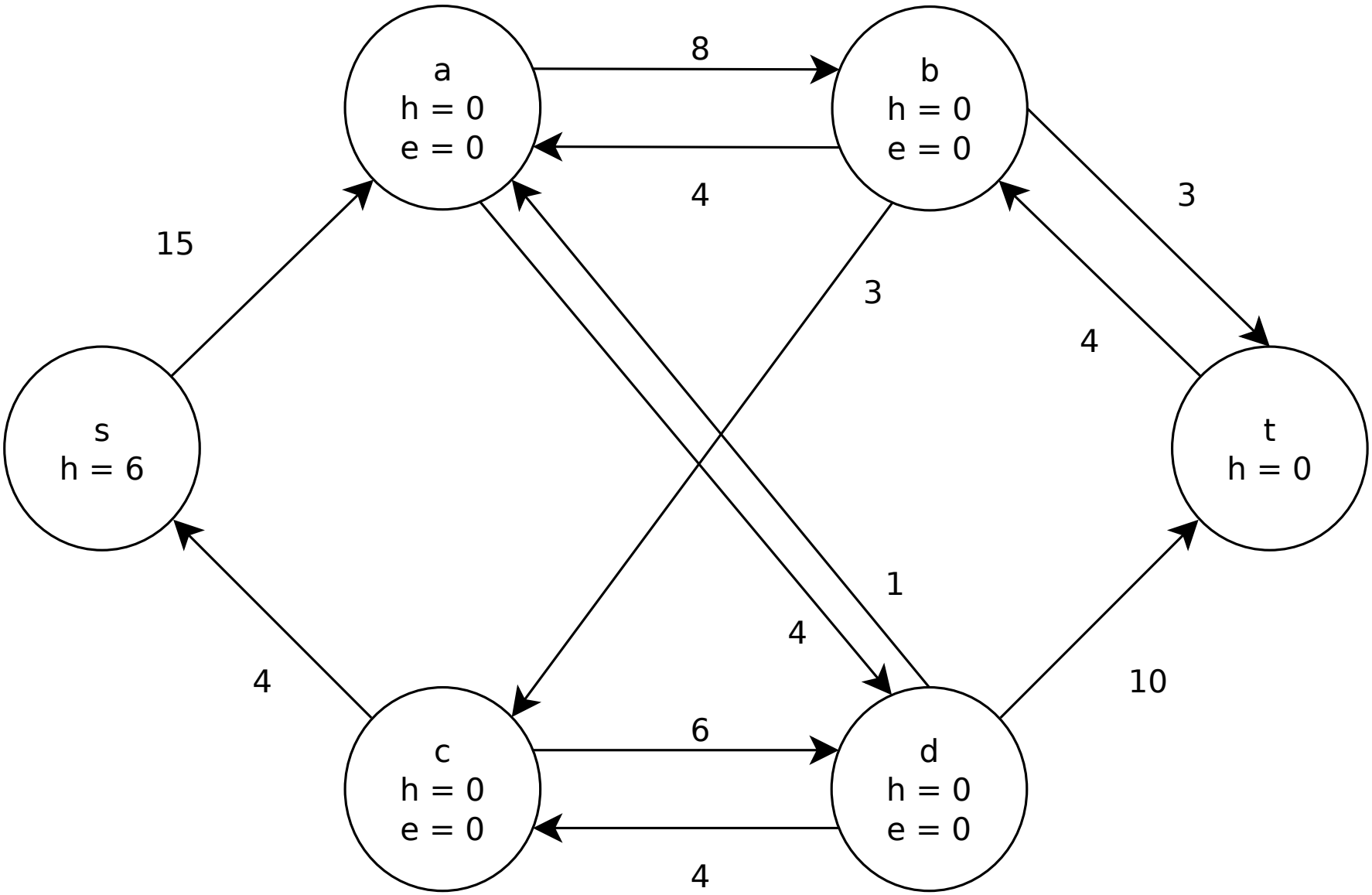
- **Push-Schritt** entlang einer erlaubten Kante $e = (u, v)$ aus einem aktiven Knoten:
  $f(e) \leftarrow f(e) + \min\{ex(u), rest(e)\}$ wenn $e \in E(G)$
  Sonst is e in Restgraphen die Rück-Kante einer Kante $e' \in E(G)$. In diesem Fall setze
  $f(e') \leftarrow f(e') - \min\{ex(u), rest(e)\}$.
- **Relabel-Schritt** an einem aktiven Knoten u ohne erlaubte Kante:
  $h(u) \leftarrow 1 + \min\{h(v) | (u, v) \in E(G_f)\}$
  Die Kanten zu den Nachbarn v mit minimalen Wert von h werden dadurch aktiviert.

Push and Relabel: Schritte



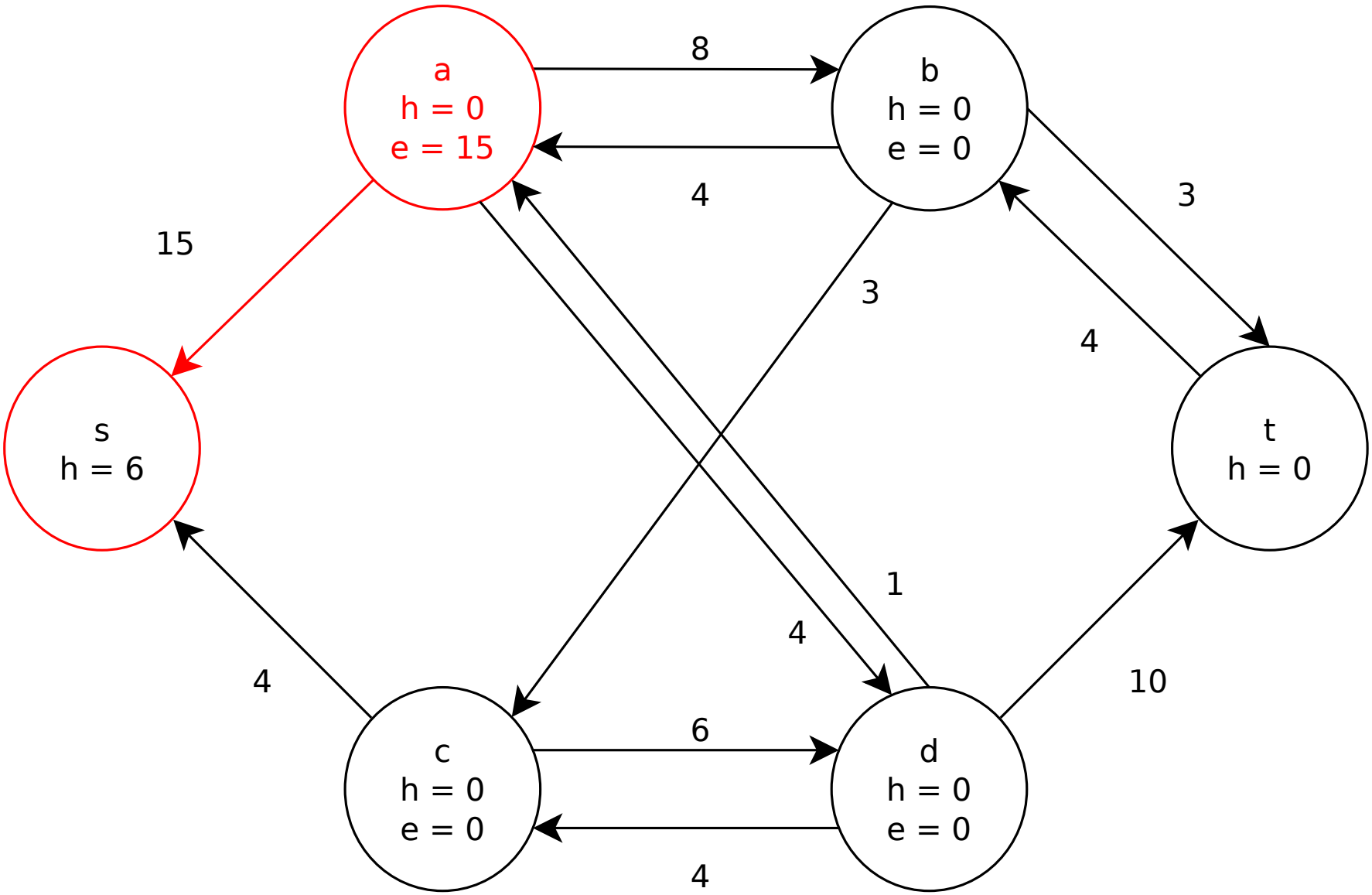
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



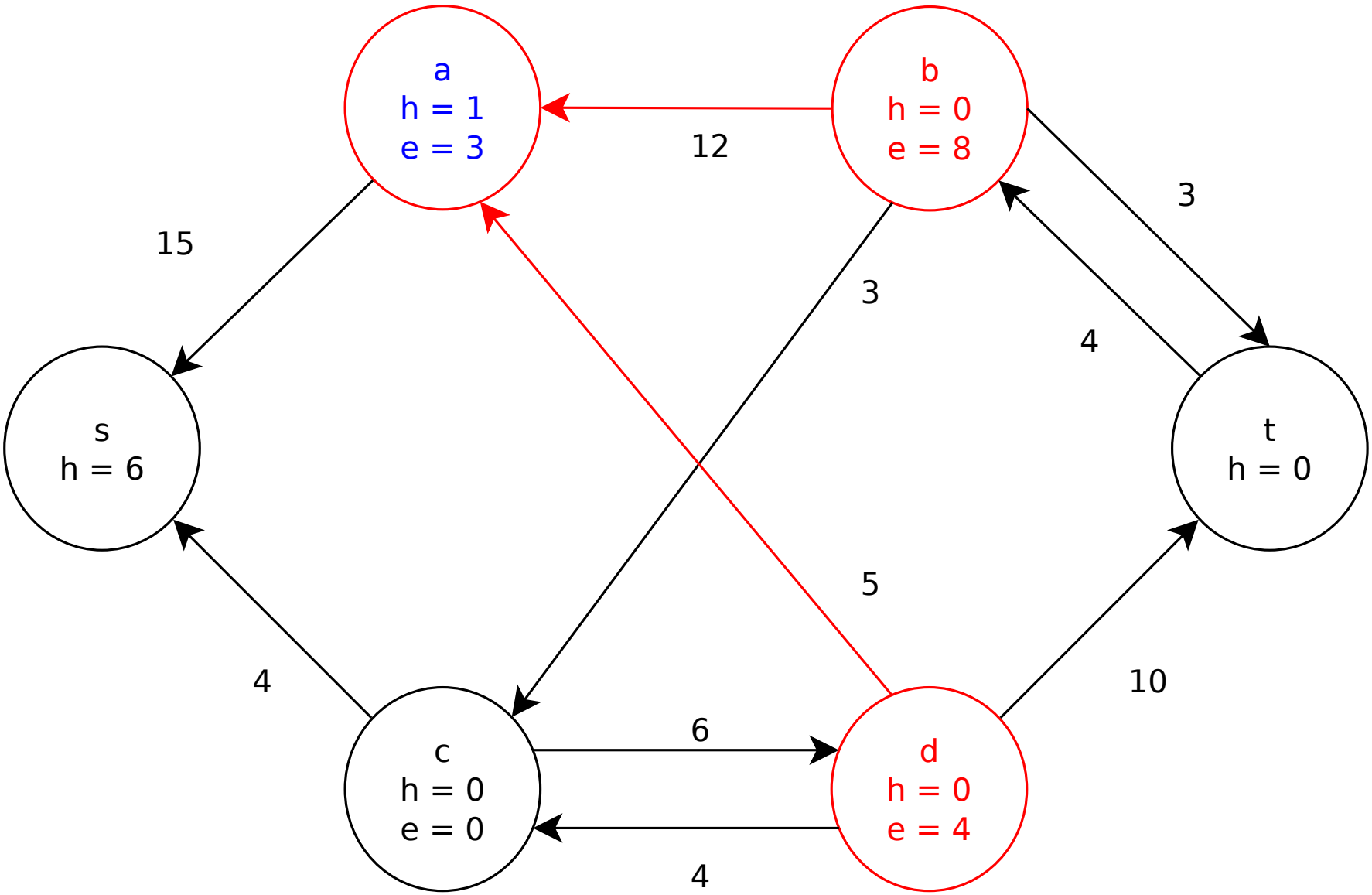
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



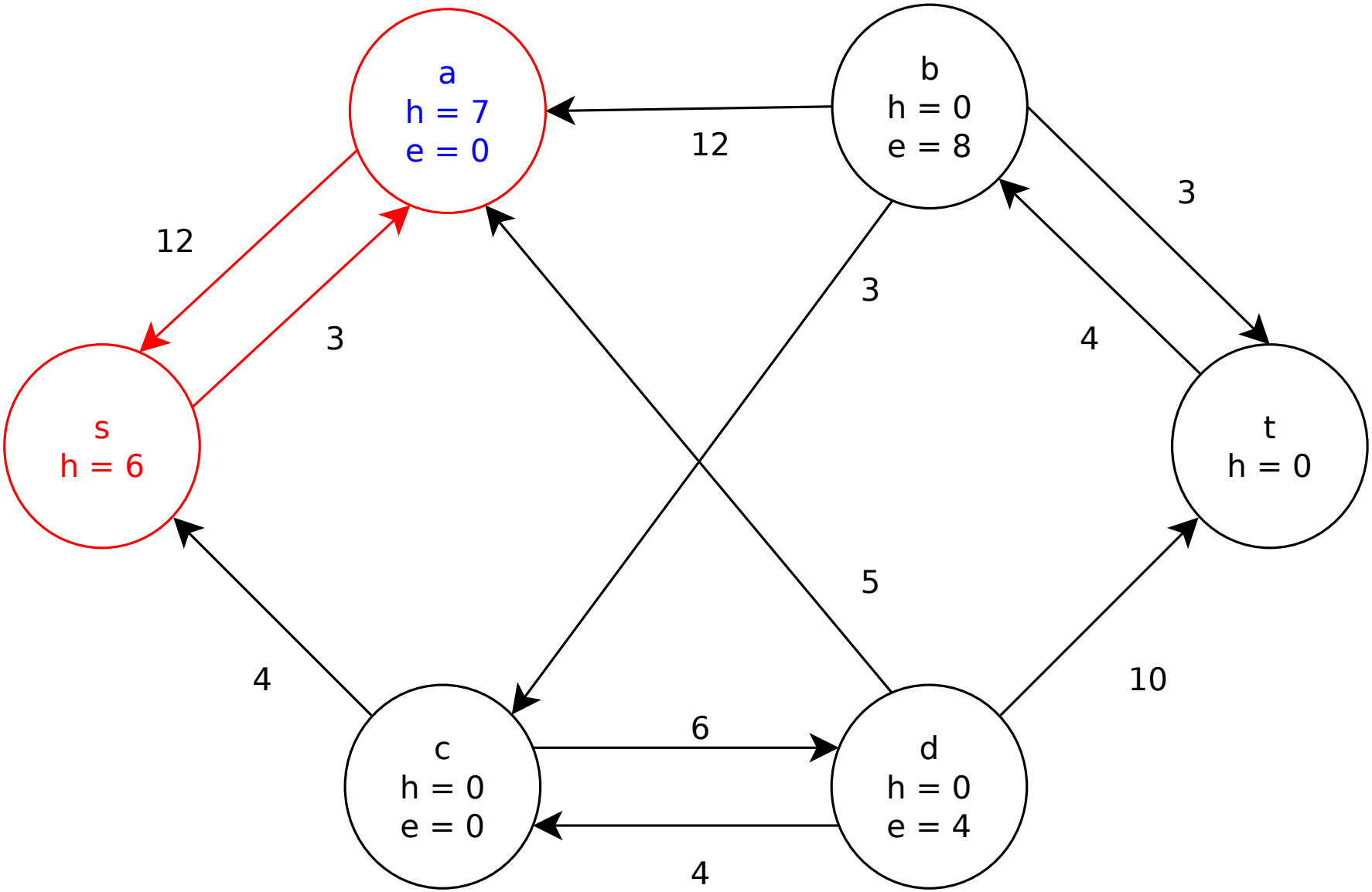
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



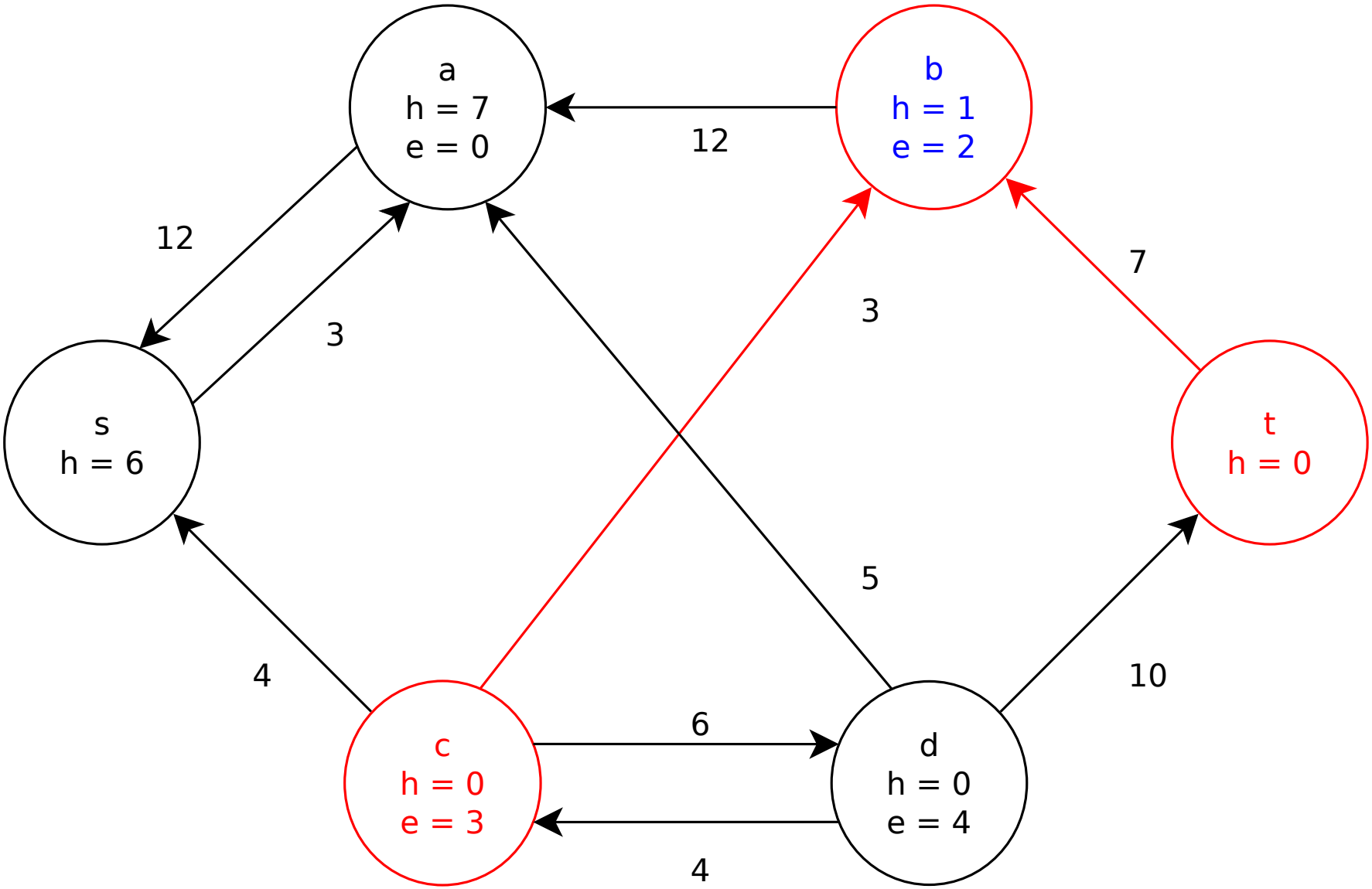
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



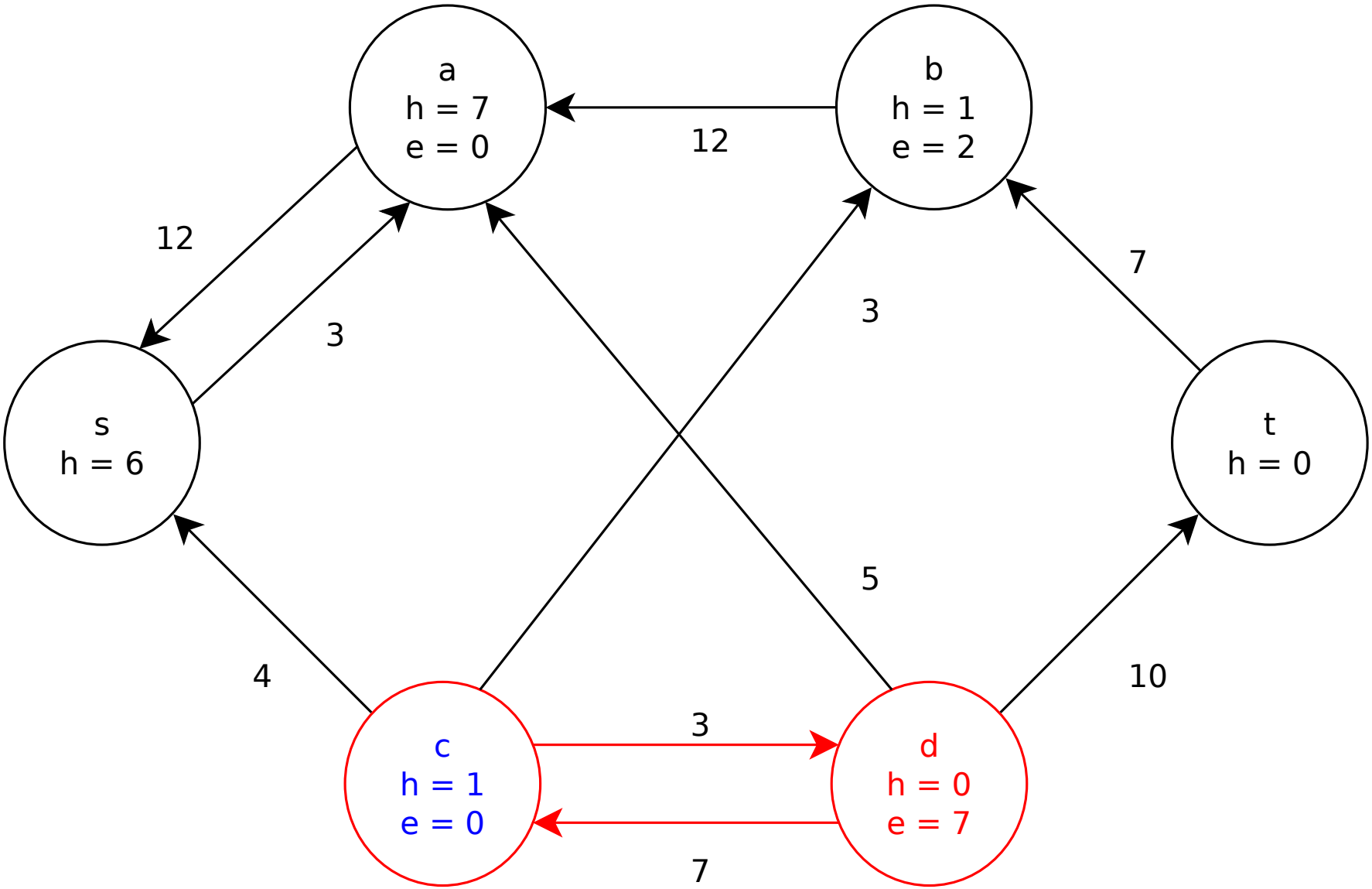
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



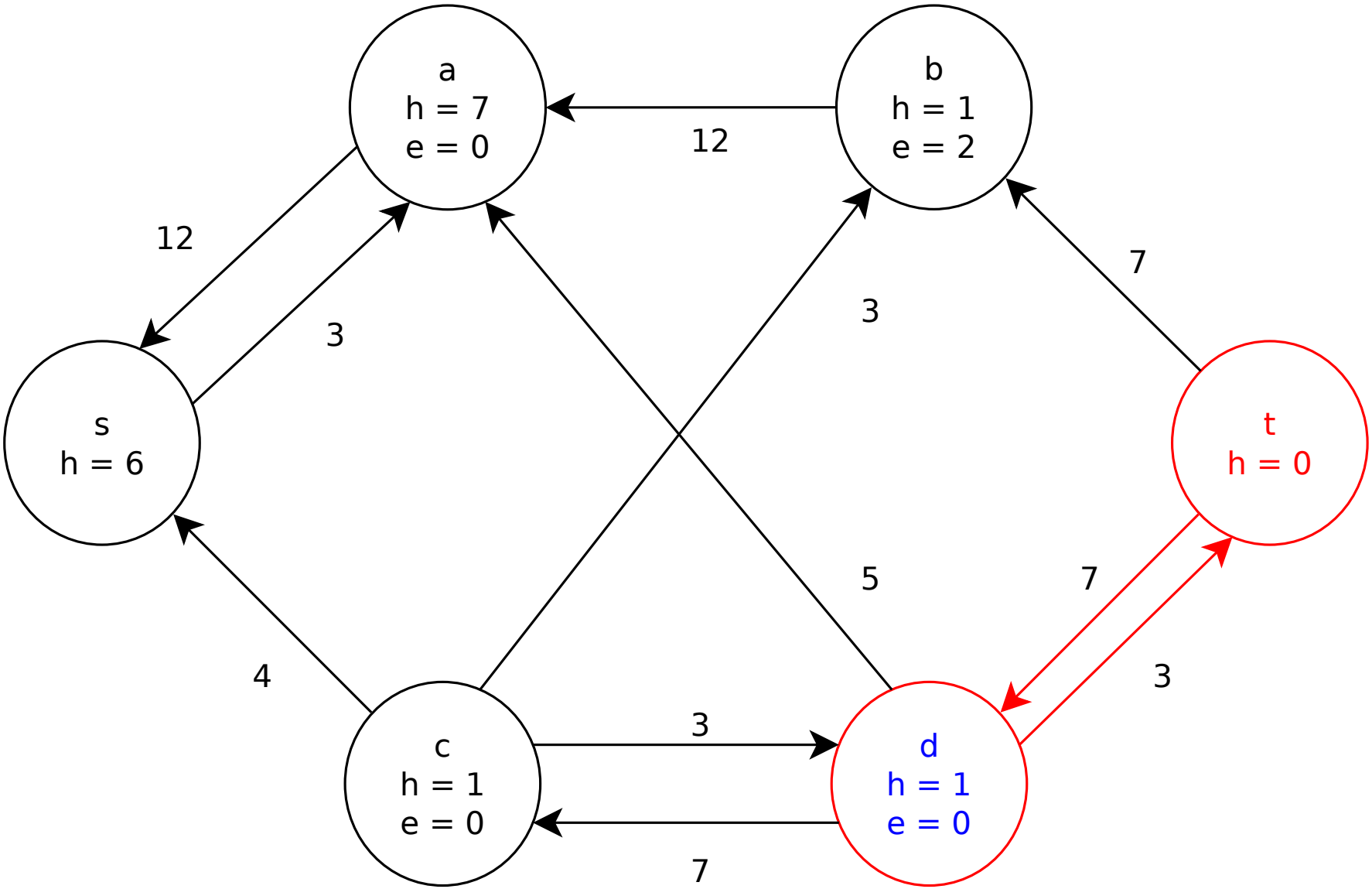
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



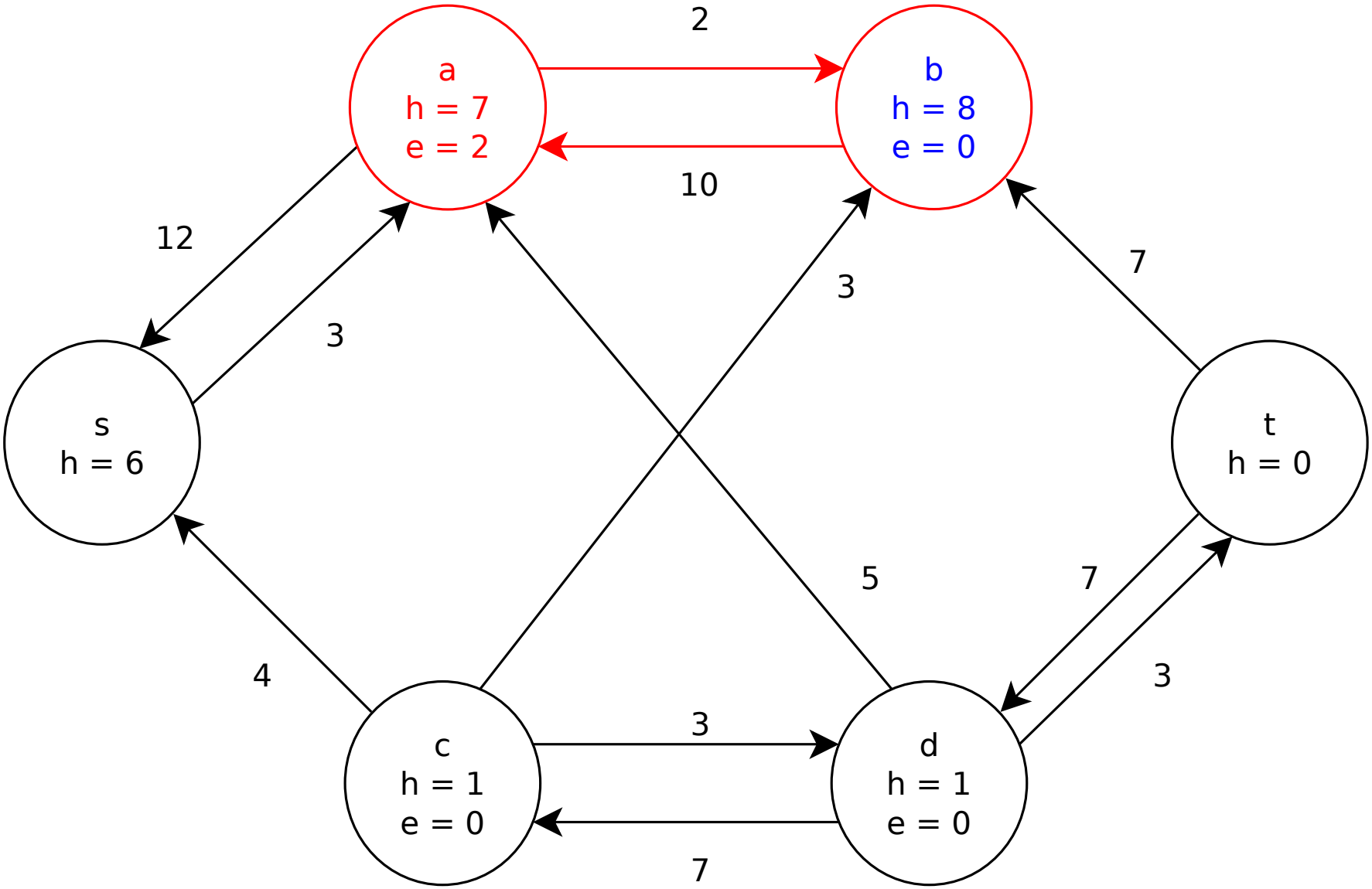
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



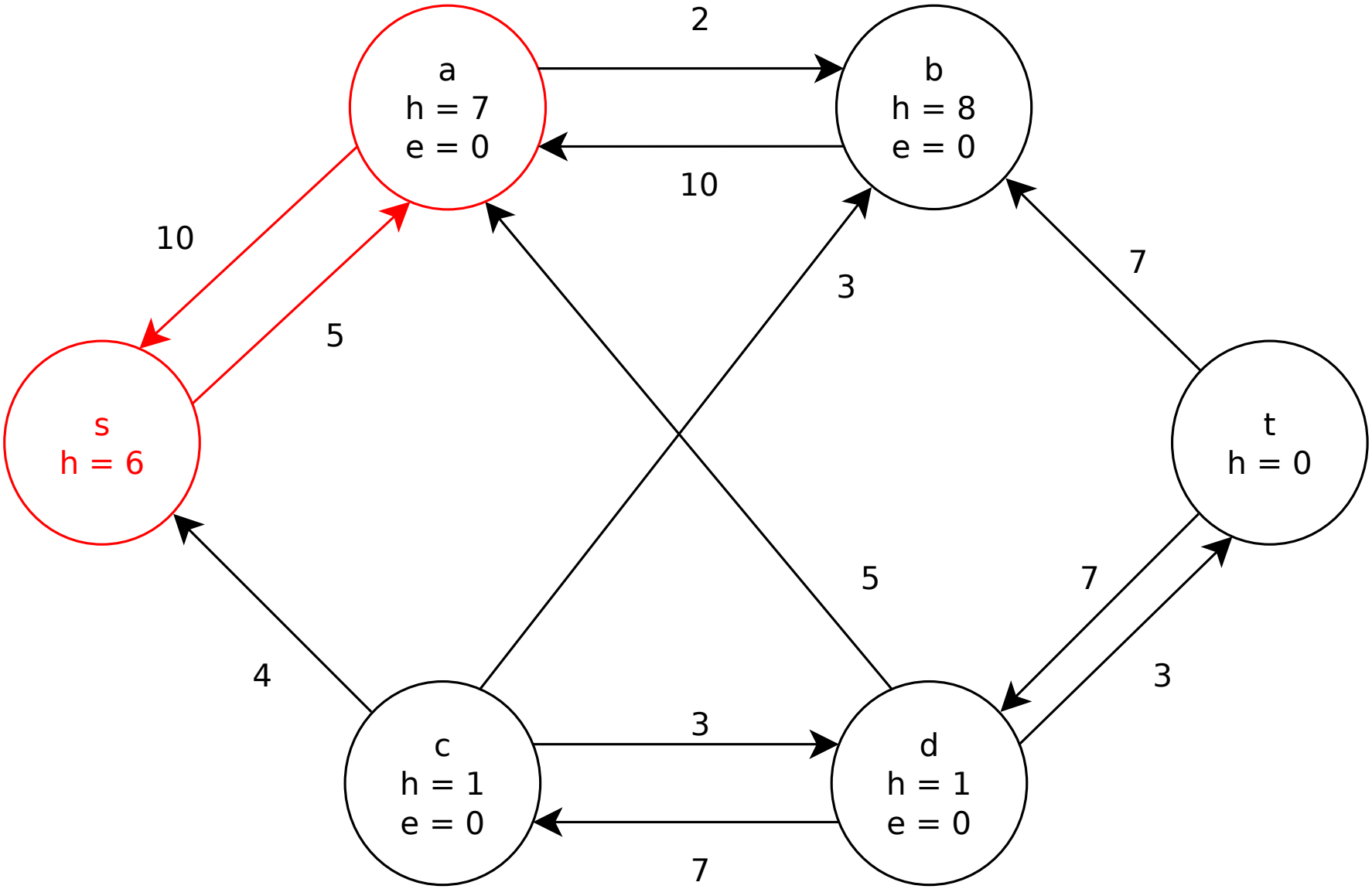
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



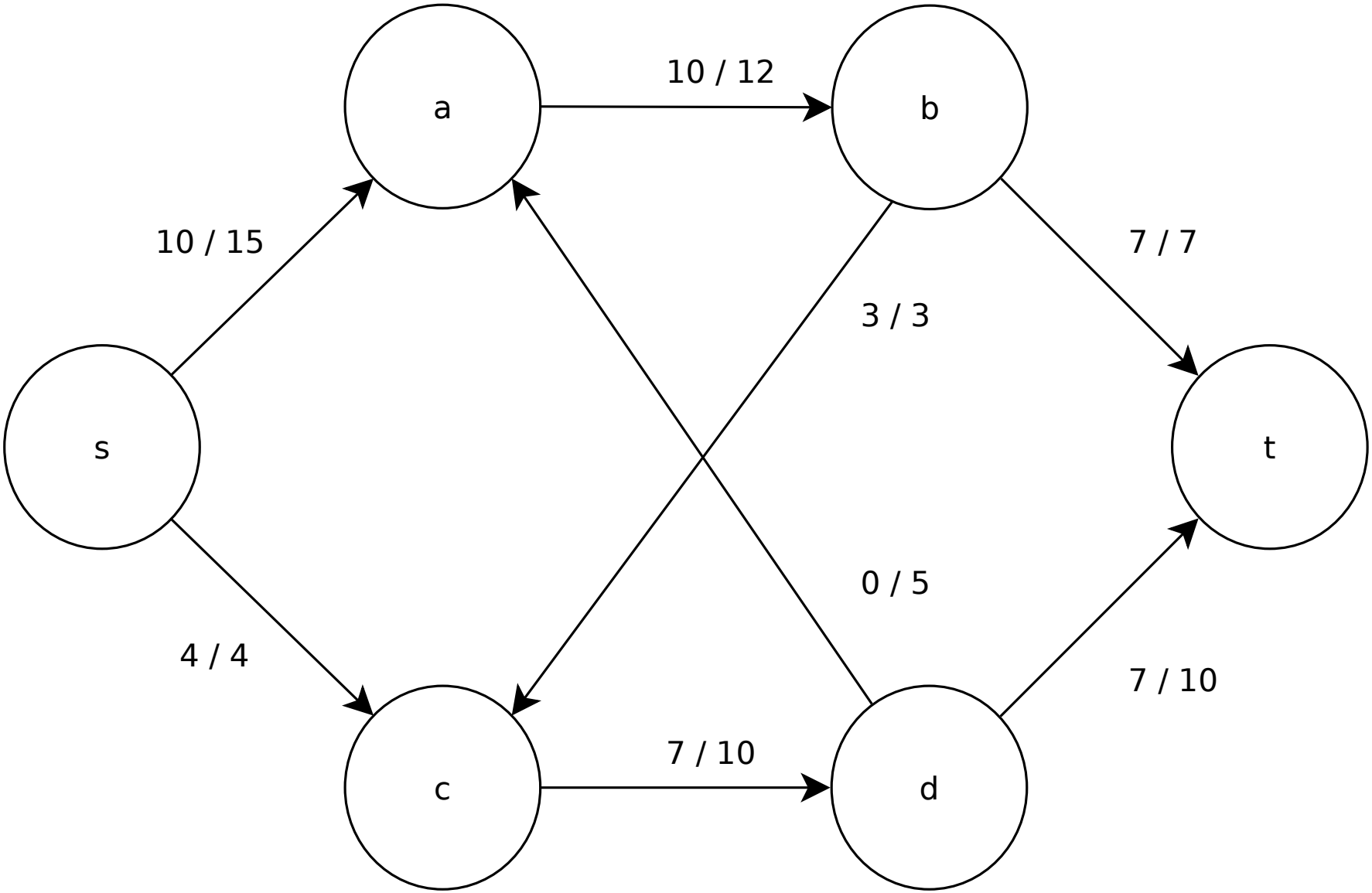
[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



[By Kamac124 - Own work, CC BY-SA 4.0]

Push and Relabel: Schritte



[By Kamac124 - Own work, CC BY-SA 4.0]

Warum funktioniert das?

- Jeder **push** Schritt entlang (u, v) erhält die **Pre-Fluss-Eigenschaft**. Dabei kann Restkapazität auf der Kante (v, u) erzeugt werden und diese in den Restgraphen aufgenommen werden. Für diese gilt $h(v) \leq h(u) + 1$
- Jeder **relabel** Schritt erhält die **Ordnungsbedingung** für h.
  Wenn $h(u)$ keine erlaubte Kante hat, dann gilt $h(u) \leq h(v)$. Nach dem **relabel** Schritt ist daher $h'(u) > h(u)$. Für einlaufende Kanten (p, u) gilt also $h(p) \leq h(u) + 1 < h'(u) + 1$. Für die auslaufenden Kanten gilt $h'(u) \leq h(v) + 1$ per Konstruktion. Da h nur am Knoten u verändert wird, erfüllt h nach dem **relabel** Schritt die Ordnungsbedingung

Jeder **push** und jeder **relabel** Schritt erhät die Pre-Fluss und die Ordnungsbedingung

Im Restgraphen gibt es hier **keinen** Pfad von s nach t: Entlang eines solchen Pfades ändert sich h höchstens um 1 entlang jeder Kante. Da $h(s) = |V|$ und $h(t) = 0$, müsste ein erweiternder Pfad mindestens $|V|$ Kanten enthalten. Es gibt aber in einem Graphen keine Pfade, die länger als $|V| - 1$ sind.

Korrektheits- und Komplexitätsbetrachtung

Wenn der Algorithmus terminiert, sind weder aktive Knoten noch erlaubte Kanten übrig.
Keine aktiven Knoten impliziert, dass der Exzess an jedem Knoten verschwindet und der Pre-Fluss f ist daher ein Fluss.
Im Restgraphen gibt es keinen Pfad von s nach t, also muss der Fluss (auf grund des min-cut-max-flow Theorems) maximal sein.
Da in jedem **push** Schritt der Exzess reduziert wird, kann das nur endlich oft gemacht werden.
Es sind $O(|E|)$ Kanten zu betrachten.
In jedem **relabel** Schritt wird ein $h(u)$ erhöht. Da $h(u) \leq f(v)$ entlang erlaubter Kanten ist, kann $h(u)$ nicht grösser als $2|V|$ werden, d.h., es können nicht mehr als $O(|V|)^2$ **relabel** Schritte auftreten.
Daher terminiert der push-relabel Algorithmus nach $O(|V|^2|E|)$ Schritten.

Verbesserte Auswahl eines aktiven Knotens: Discharging

Idee: bearbeite einen aktiven Knoten bis er inaktiv wird.

- verwende **push** Schritte bis entweder der Knoten inaktiv ist oder alle erlaubten Kanten saturiert sind, i.e., die gesamte Restkapazitaet entlang der erlaubten Kanten aufgebraucht ist.
- mache einen **relabel** Schritt um neue erlaubte Kanten zu finden.
- wiederhole, bis der Knoten inaktiv ist.

Ein solcher Schritt heisst **discharging**.
Zusammenfassung:

- Starte mit dem trivialen Pre-Fluss definiert durch die Kapazitaet der auslaufenden Kanten von s, $f(s,u) = c(s,u)$ und $f(x,y) = 0$ sonst. Konstruiere den Restgraphen G_f.
- Solange es einen Aktiven Knoten gibt: **discharge** ihn.

Die Performanz des Algorithmus verbessert sich, wenn zunaechst die aktiven Knoten mit dem grössten Wert von h **discharged** werden.

Matching

Matching (Zuordnung) M für ungerichteten Graphen $G = (V, E)$ ist eine Teilmenge der Kanten, so dass jeder Knoten in V in höchstens einer Kante vorkommt

- $|M|$ = Größe der Zuordnung
- Perfektes Matching: kein Knoten bleibt "allein" (unmatched), d.h. jeder Knoten ist in einer Kante von M vertreten

Matching M ist maximal, wenn es kein Matching M' gibt mit $|M| < |M'|$

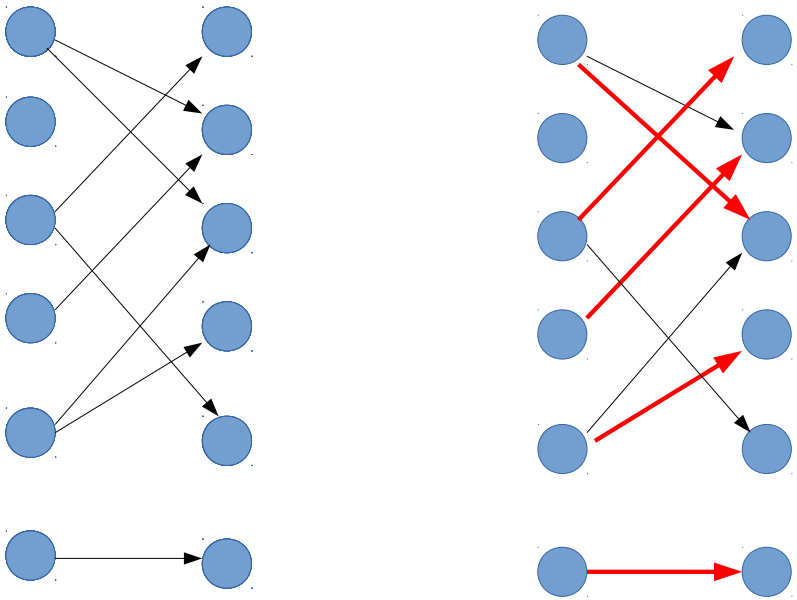
Ein bipartiter Graph ist ein Graph, dessen Knotenmenge V in zwei disjunkte Teilmengen V_1 und V_2 aufgeteilt ist, und dessen Kanten jeweils einen Knoten aus V_1 mit einem aus V_2 verbinden (also keine Kanten innerhalb von V_1 oder V_2).

Maximales Bipartites Matching

Beispiel (leicht verallgemeinerbar):

- Eine Gruppe von M Personen bewirbt sich auf N Jobangebote
- Jede Person hat eine Prioritätenliste von präferierten Jobs
- Jeder Arbeitgeber hat eine Proritätenliste von präferierten Kandidaten
- Wieviele Personen können maximal auf die verfügbaren Jobs verteilt werdeen?

Maximales Bipartites Matching - Beispiel



Verteilung von Bewerbern auf Jobangebote

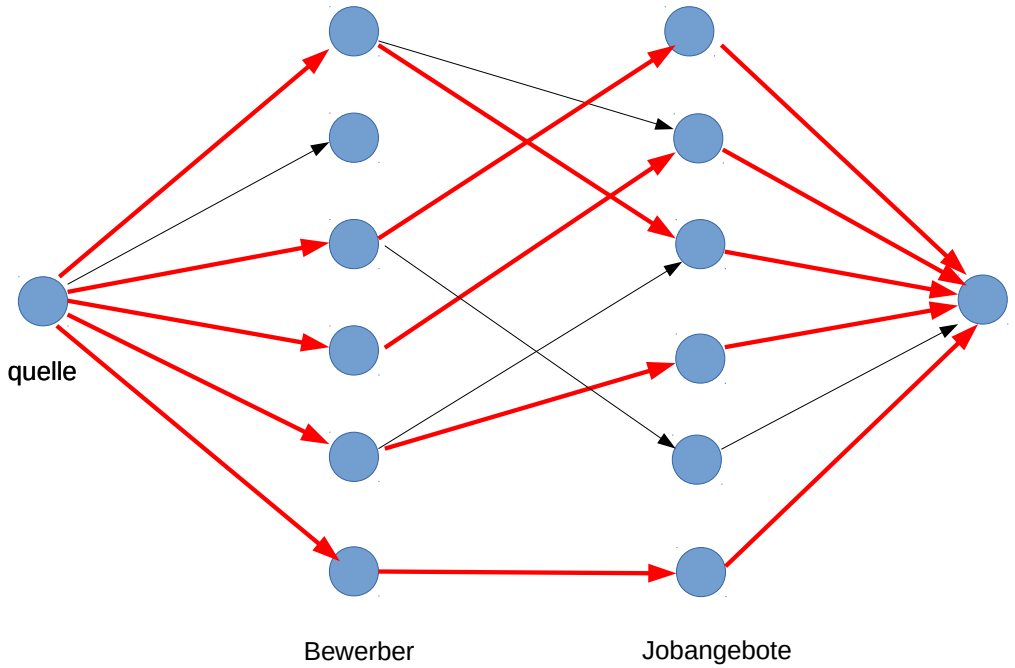
Bipartites Matching und maximaler Fluss

Maximales Matching kann auf maximalen Fluss zurückgeführt werden:

- Quelle und Senke hinzufügen.
- Kanten von V_1 nach V_2 richten.
- Jeder Knoten in V_1 erhält eingehende Kante von der Quelle.
- Jeder Knoten in V_2 erhält ausgehende Kante zur Senke.
- Alle Kanten erhalten Kapazität $c(e) = 1$

$\rightarrow$ Anwendung des Ford-Fulkerson-Algorithmus

Maximales Bipartites Matching - Beispiel



Verteilung von Bewerbern auf Jobangebote