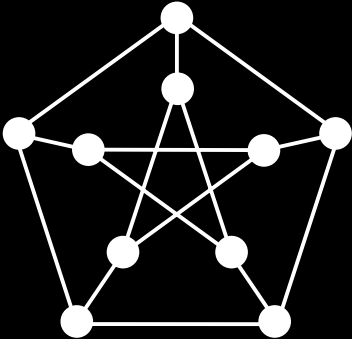


Gianluca Della Vedova

- Large-Scale Graph Algorithms
- Ufficio U14-2041
- <https://www.unimib.it/gianluca-della-vedova>
- gianluca.dellavedova@unimib.it
- <https://github.com/gdv/large-scale-graph-algorithms>
- Everything at <https://elearning.unimib.it/>

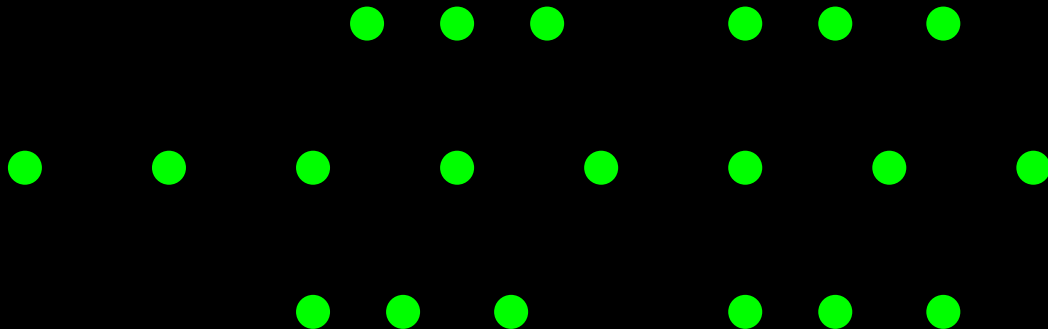
Example



Notation

- number of vertices: n
- number of edges/arcs: m

Better representation



Almost a path

- Compact representation

Breadth-first visit

Data: graph G , vertex $root$

$Q \leftarrow$ a queue;

label $root$ as explored;

$Q.enqueue(root)$;

while $Q \neq \emptyset$ **do**

$v \leftarrow Q.dequeue()$;

foreach $edge(v, w)$ **do**

if w is not labeled as explored **then**

 label w as explored;

$Q.enqueue(w)$

Depth-first visit

Data: graph G , vertex $root$

$S \leftarrow$ a stack;

$S.push(root)$;

while $S \neq \emptyset$ **do**

$v \leftarrow S.pop()$;

if v is not labeled as explored **then**

 label v as explored;

foreach edge (v, w) **do**

$S.push(w)$

Dijkstra's algorithm

Data: graph G , vertex $source$

$Q \leftarrow$ a queue;

foreach *vertex* v **do**

$dist[v] \leftarrow \infty$;

$Q.enqueue(v)$

$dist[source] \leftarrow 0$;

while $Q \neq \emptyset$ **do**

$u \leftarrow$ vertex in Q minimizing $dist[u]$;

$Q.dequeue(u)$;

foreach *neighbor* v of u still in Q **do**

$alt \leftarrow dist[u] + Graph.Edges(u, v)$;

if $alt < dist[v]$ **then**

$dist[v] \leftarrow alt$;

$prev[v] \leftarrow u$;

return $dist[], prev[]$;

Dijkstra's algorithm — priority queue

Data: graph G , vertex $source$

$Q \leftarrow$ a priority queue;

foreach *vertex* v **do**

$dist[v] \leftarrow \infty$;

$Q.add_with_priority(v, dist[v])$

$dist[source] \leftarrow 0$;

while $Q \neq \emptyset$ **do**

$u \leftarrow Q.extract_min$;

foreach *neighbor* v of u still in Q **do**

$alt \leftarrow dist[u] + Graph.Edges(u, v)$;

if $alt < dist[v]$ **then**

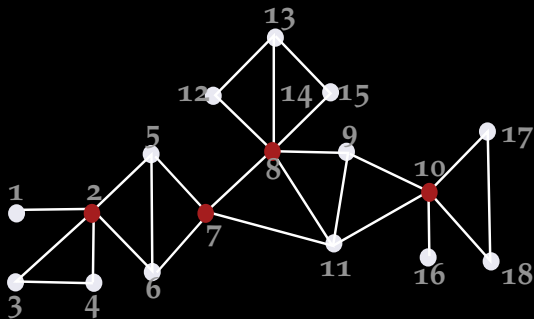
$dist[v] \leftarrow alt$;

$prev[v] \leftarrow u$;

$Q.decrease_priority(v, alt)$

return $dist[], prev[]$;

Biconnected components



Find articulation points

Data: connected graph G , vertex $root$

$S \leftarrow$ a stack;

$S.push((root, nil));$

$d \leftarrow -1;$

while $S \neq \emptyset$ **do**

$d \leftarrow d + 1;$

$(v, p) \leftarrow S.peek();$

if v is not explored **then**

 label v as explored; $parent(v) \leftarrow p$; $depth[v] \leftarrow d$;

$lowpoint[v] = depth[v]$;

foreach edge (v, w) **do**

$S.push((w, v));$

else

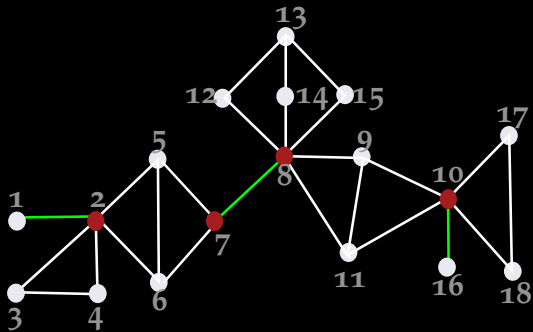
$lowpoint[v] =$

$\min \{ depth[v], \min_{w \in N(v), w \neq p} \{ depth[w] \}, \min_{w \in N(v), parent(w)=v} \{ lowpoint[w] \} \}$

$d \leftarrow d - 1;$

$v \leftarrow S.pop();$

2-edge connected components



Find bridges

Data: connected graph G , vertex $root$

$S \leftarrow$ a stack, $S.push(root, nil)$;

$x \leftarrow -1$;

while $S \neq \emptyset$ **do**

$(v, p) \leftarrow S.peek()$;

if v is not explored **then**

$x \leftarrow x + 1$;

 label v as explored; $P(v) \leftarrow p$; $num[v] \leftarrow x$;

foreach edge (v, w) **do**

$S.push((w, v))$;

else

$nd[v] = 1 + \sum_{w \in N(v), P(w)=v} nd[w]$;

$l[v] =$

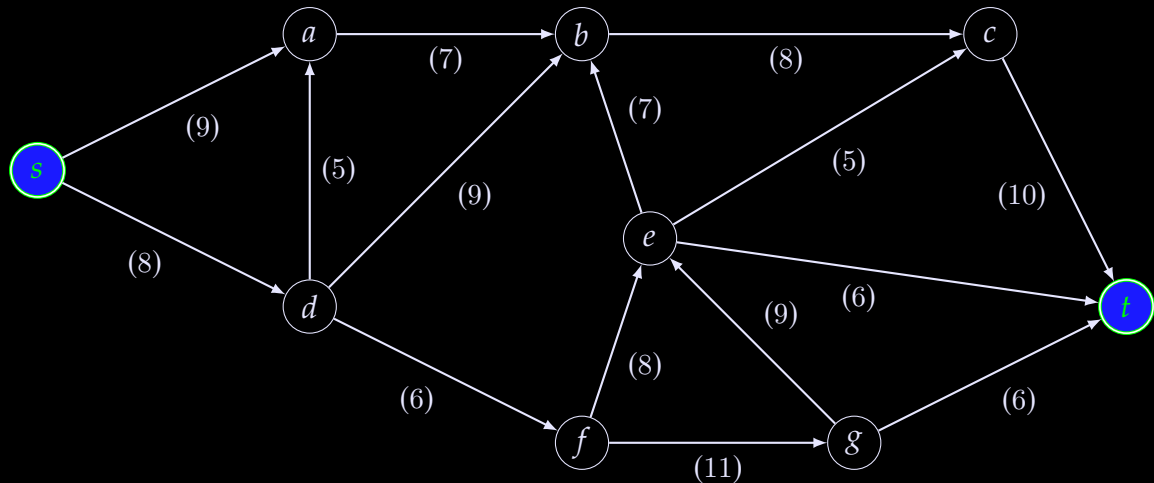
$\min \{ num[v], \min_{w \in N(v), w \neq p, P(w) \neq v} \{ num[w] \}, \min_{w \in children(v)} \{ l[w] \} \}$;

$h[v] =$

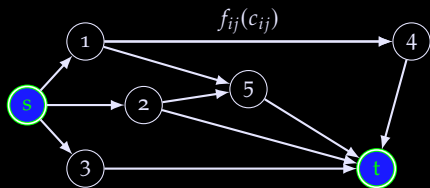
$\max \{ num[v], \max_{w \in N(v), w \neq p, P(w) \neq v} \{ num[w] \}, \max_{w \in children(v)} \{ h[w] \} \}$;

$v \leftarrow S.pop()$;

Max flow



Max flow

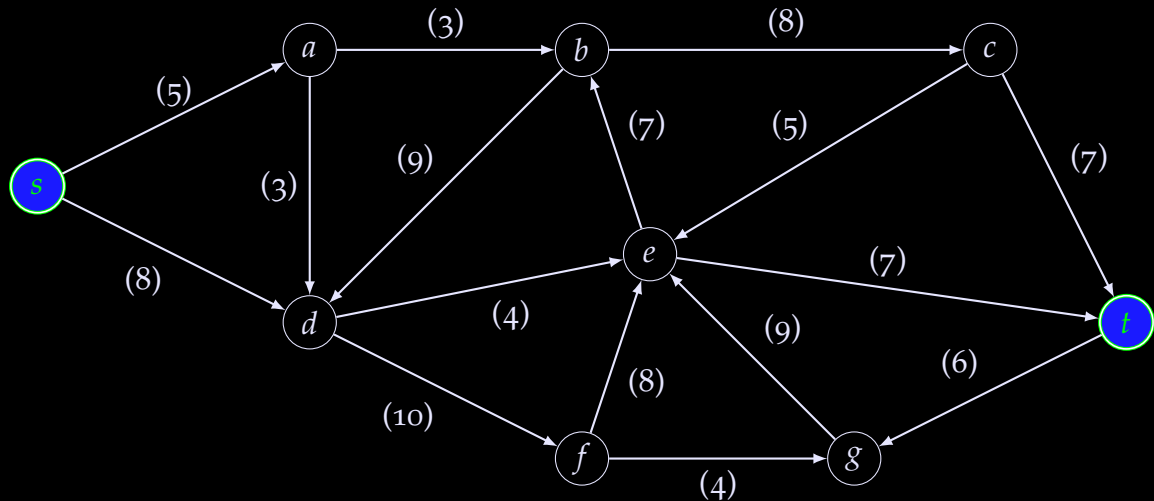


- c_{ij} : capacity of the arc (i, j)
- f_{ij} : flow through the arc (i, j)
- v_i : flow imbalance of the vertex i (< 0 incoming, > 0 outgoing)

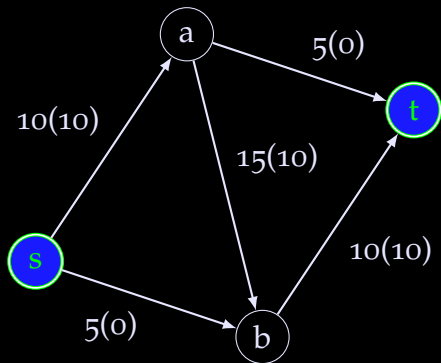
max F

$$\text{s.t. } \sum_{j:(i,j) \in E} f_{ij} - \sum_{k:(i,k) \in E} f_{ki} = \begin{cases} F & \text{if } i = s \\ -F & \text{if } i = t \\ 0 & \text{if } i \text{ otherwise} \end{cases} \quad \forall i \in V$$
$$0 \leq f_{ij} \leq c_{ij} \quad \forall (i, j) \in E$$

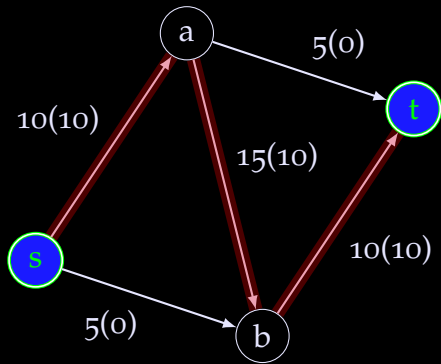
Example



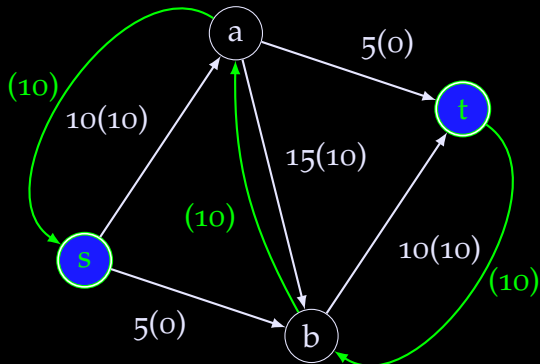
Residual network



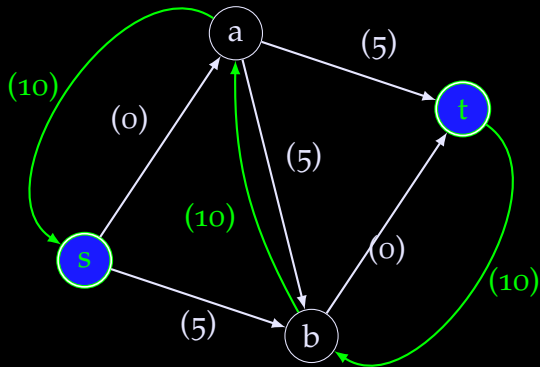
Residual network



Residual network



Residual network



Cut

Let $G = \langle V, E \rangle$ be a directed graph and let $S \subseteq V$. Then:

- $(S, \bar{S}) = E \cap (S \times \bar{S})$ is a forward cut,
- $(\bar{S}, S) = E \cap (\bar{S} \times S)$ is a backward cut,
- $E \cap ((S \times \bar{S}) \cup (\bar{S} \times S))$ is a cut.

Flow — Cut

Lemma 1

Let $G = \langle V, E \rangle$ be a directed graph and let (S, \bar{S}) be a bipartition of V , with $s \in S$, $t \notin S$. Let f be an (s, t) -flow with total flow F . Then

$$F = \sum_{e \in (S, \bar{S})} f(e) - \sum_{e \in (\bar{S}, S)} f(e)$$

Lemma 2

Let $G = \langle V, E \rangle$ be a directed graph and let (S, \bar{S}) be a bipartition of V , with $s \in S$, $t \notin S$. Let f be an (s, t) -flow with total flow F . Then

$$F \leq \sum_{e \in (S, \bar{S})} c(e)$$

Max flow – Min cut theorem

Let f be a flow of a graph $G = (V, E)$. Then the following three conditions are equivalent:

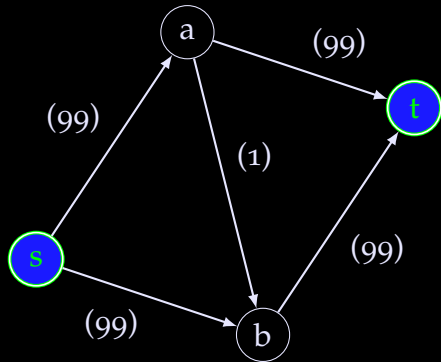
- 1 f is a maximum flow
- 2 the residual graph has no **augmenting** path
- 3 there is a cut (S, T) of G such that $c(S, T) = |f|$

Ford–Fulkerson

- 1 Find an augmenting path
- 2 Use it!

Ford–Fulkerson

- 1 Find an augmenting path
- 2 Use it!

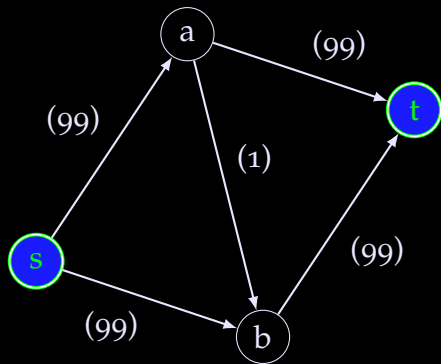


Ford–Fulkerson

- 1 Find an augmenting path
- 2 Use it!

Edmonds–Karp

BFS to find the augmenting path



Figures

- David Eppstein, Public Domain,
<https://commons.wikimedia.org/w/index.php?curid=10261635>

License

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (<https://creativecommons.org/licenses/by-sa/4.0/>).

