Algorithms and Computability

Lecture 3
Maximum Flow

SW6 spring 2025

Simonas Šaltenis

AALBORG UNIVERSITY
DENMARK

Maximum flow

- Main goals of the lecture:
  - *to understand how flow networks and maximum flow problem can be* ***formalized****;*
  - *to understand the* ***Ford-Fulkerson*** *method and to be able to prove that it works correctly;*
  - *to understand the* ***Edmonds-Karp*** *algorithm and the intuition behind the analysis of its worst-case running time;*
  - *To understand how* ***linear programming*** *can be used to solve different variants of network flow problems*
  - *to be able to apply the Ford-Fulkerson method to solve the* ***maximum-bipartite-matching*** *problem.*

Flow networks

- *What if the weights in a graph are maximum capacities of some flow of material*?
  - Pipe network to transport fluid (e.g., water, oil)
    - Edges – pipes, vertices – junctions of pipes
  - Data communication network
    - Edges – network connections of different capacity, vertices – routers (do not produce or consume data just move it)
  - Concepts (informally):
    - ***Source*** vertex *s* (where material is produced)
    - ***Sink*** vertex *t* (where material is consumed)
    - For all other vertices – what goes in must go out
    - *Goal: maximum rate of material flow from source to sink*

Formalization

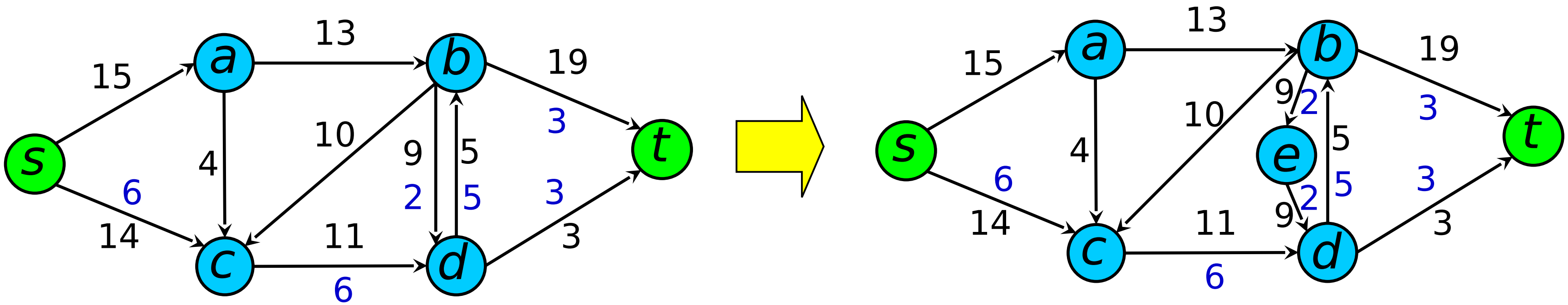
- *How do we formalize flows?*
  - Graph $G = (V, E)$ – a **flow network**
    - Directed, each edge has **capacity** $c(u,v) \geq 0$
    - Two special vertices: ***source*** s, and ***sink*** t
    - For any other vertex v, there is a path $s \rightarrow \ldots \rightarrow v \rightarrow \ldots \rightarrow t$
  - **Flow** – a function $f : V \times V \rightarrow R$
    - *Capacity constraint*: For all $u, v \in V$: $0 \leq f(u,v) \leq c(u,v)$
    - *Flow conservation:* For all $u \in V - \{s, t\}$:

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$$

$$f(V,u) = f(u,V)$$

Antiparallel edges

- To simplify the discussion we do not allow both (*u*,*v*) and (*v*,*u*) together in the graph.
- Easy to eliminate such *antiparallel* edges by introducing artificial vertices.



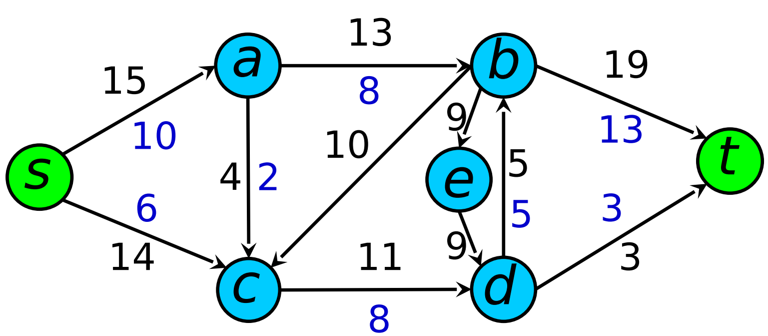
Maximum flow

- *What do we want to maximize?*
  - ***Value*** of the flow *f*:

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) = f(s,V) - f(V,s)$$

  - Equivalently:

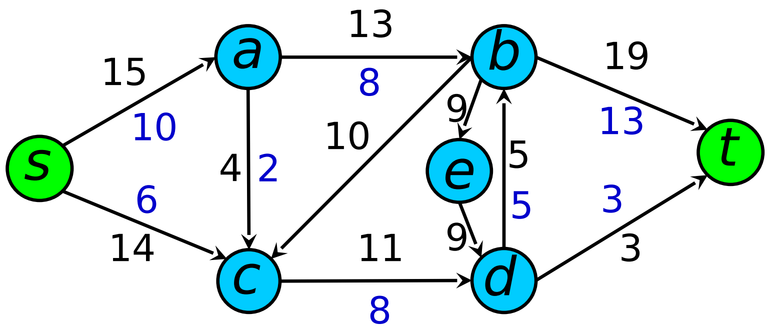
$$|f| = \sum_{v \in V} f(v,t) - \sum_{v \in V} f(t,v) = f(V,t) - f(t,V)$$



- *The goal: to find a flow of maximum value!*

Augmenting path

- *Straightforward idea for an algorithm*:
  - If $|f| \geq 0$, i.e., we have some flow (or none),…
  - …and can find a path *p* from *s* to *t* (***augmenting path***), such that there is $a > 0$, and for each edge (u,v) in *p* we can add *a* units of flow: $f(u,v) + a \leq c(u,v)$
  - Then just do it, to get a better flow!
  - *Augmenting path in this graph*?



The Ford-Fulkerson method

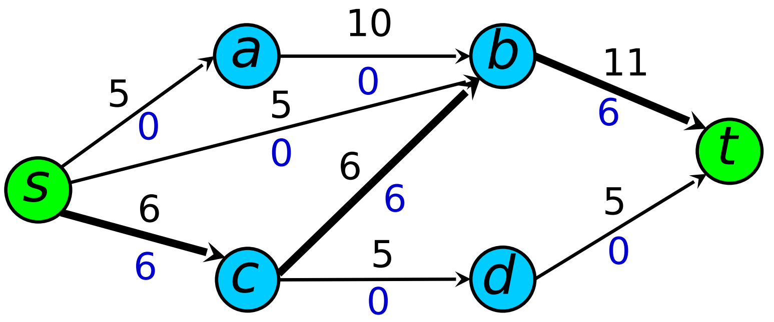
- Sketch of the method:

```
Ford-Fulkerson(G,s,t)
01 initialize flow f to 0 everywhere
02 while there is an augmenting path p do
03    augment flow f along p
04 return f
```

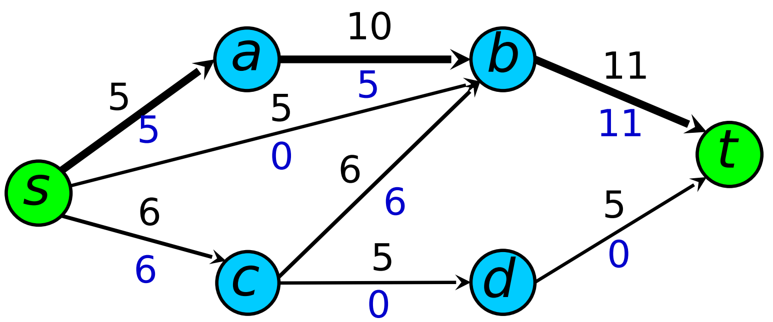
- *How do we find augmenting path?*
- *How much additional flow can we send through that path?*
- *Does the algorithm always find the maximum flow?*

Changing our mind...

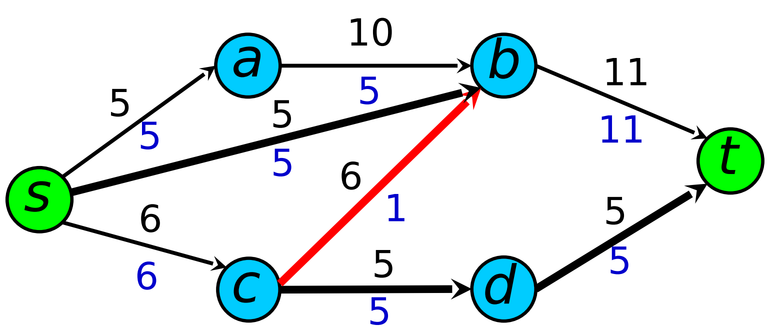
- *Need to be careful how we define the augmenting path*:
  - Let’s try sending flow:
    - ***scbt*** : 6 units



    - ***sabt*** : 5 units



    - *Can we do more?*
    - *Yes!:* ***sbcdt*** : 5 units!



Residual network

- *How do we find an augmenting path?*
  - It is any path in a ***residual network***:
    - Residual capacities:

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

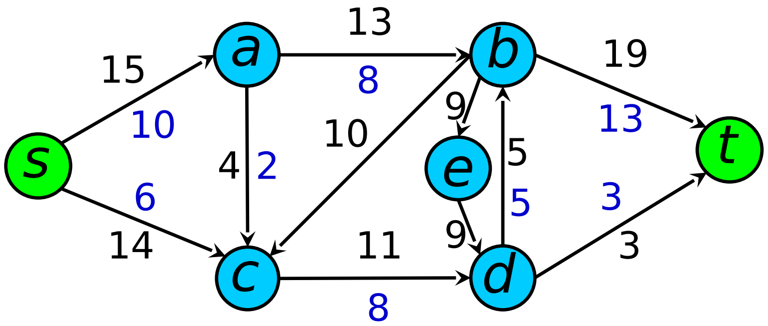
Remember: no antiparallel edges!

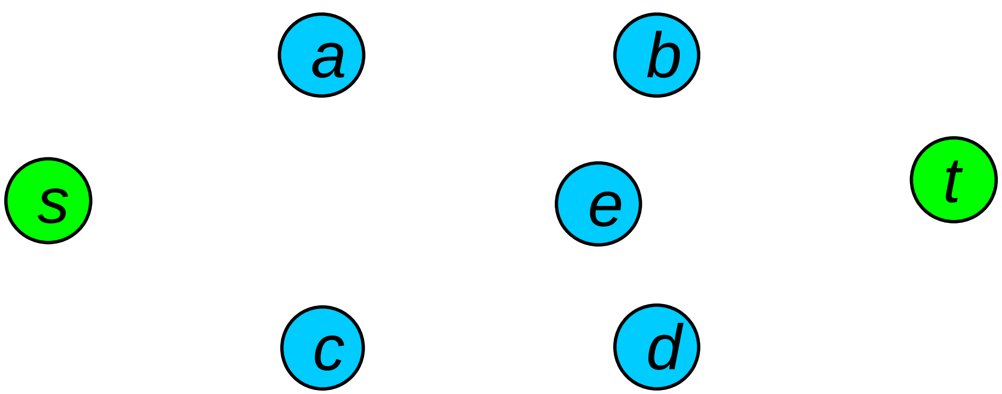
    - Residual network: $G_f=(V, E_f)$, where

      $E_f = \{(u,v) \in V \times V: c_f(u,v) > 0\}$
    - Observation – edges in E_f are either edges in E or their reversals: $|E_f| \leq 2|E|$

Compute residual network

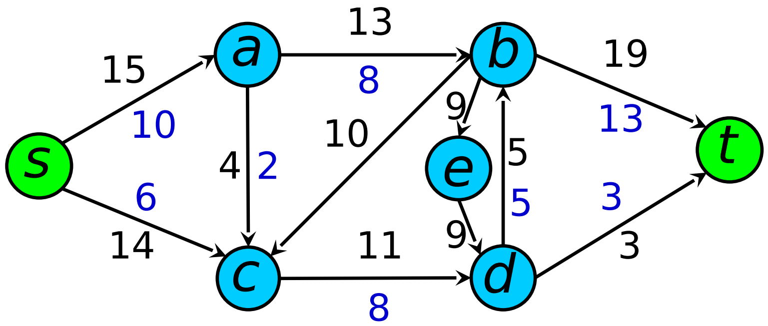
- *Compute residual network*:

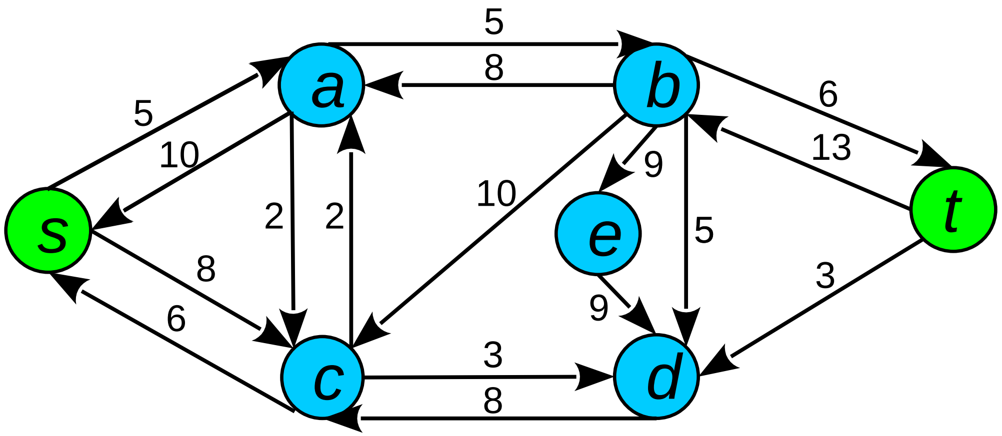




Compute residual network

- *Compute residual network*:





Residual capacity of a path

- *How much additional flow can we send through an augmenting path*?
  - ***Residual capacity*** of a path p from s to t in G_f:

    $c_f(p) = \min\{c_f(u,v): (u,v) \text{ is in } p\}$
  - Doing augmentation: for all (u,v) in p, we just add this $c_f(p)$ to $f(u,v)$, if $(u,v) \in E$, else (if $(v,u) \in E$) subtract it from $f(v,u)$.
  - Resulting flow is a valid flow with a larger value, more specifically $|f| + c_f(p)$. Why?
    - **Validity**: *capacity constraint* on each edge + *flow conservation* at each vertex
    - **Value increase**: let's look at s or t.
  -

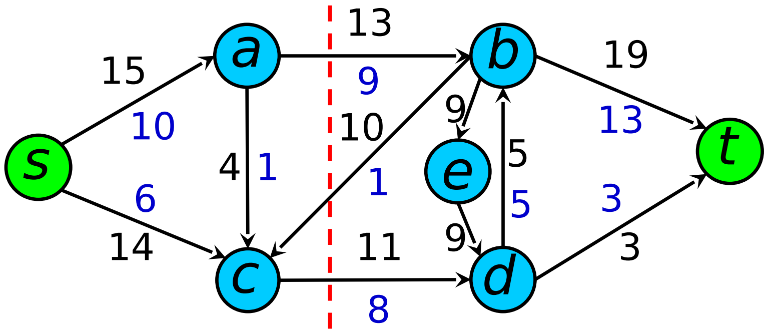
The Ford-Fulkerson method

```
Ford-Fulkerson(G,s,t)
01 for each edge (u,v) ∈ G.E do
02    f(u,v) ← 0
03 while there exists a simple path p from s to t in
   residual network G_f do
04    c_f = min{c_f(u,v): (u,v) ∈ p}
05    for each edge (u,v) ∈ p do
06       if (u,v) ∈ G.E then f(u,v) ← f(u,v) + c_f
07       else f(v,u) ← f(v,u) - c_f
08 return f
```

- The algorithms based on this method differ in how they choose *p* in step 03.

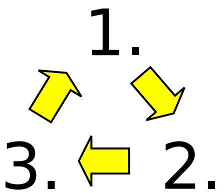
Cuts

- *Does it always find the maximum flow?*
  - A ***cut*** is a partition of V into S and $T = V - S$, such that $s \in S$ *and* $t \in T$
  - The ***net flow*** ($f(S,T)$) through the cut is the sum of flows $f(u,v)$ minus the sum of flows $f(v,u)$, where $u \in S$ *and* $v \in T$
  - The ***capacity*** ($c(S,T)$) of the cut is the sum of capacities $c(u,v)$, where $u \in S$ *and* $v \in T$
  - $|f| = f(S,T) \leq c(S,T)$
    - *Why?*



Correctness of Ford-Fulkerson

- ***Max-flow min-cut theorem***:
  - If f is the flow in G, the following conditions are equivalent:
    - 1. f is a maximum flow in G
    - 2. The residual network G_f contains no augmenting paths
    - 3. $|f| = c(S,T)$ for some cut (S,T) of G



- We have to prove three parts:
- 1. => 2. and 3. => 1. are rather obvious
- For 2. => 3., let's examine the cut:
- $(S, V - S)$, where S = $\{v \in V$: *there exists a path from s to v in* $G_f\}$
- From this we have 1. $\Leftrightarrow$ 2., which means that the Ford-Fulkerson method always correctly finds a maximum flow

Worst-case running time

- *What is the worst-case running time of this method?*
  - Let's assume integer flows.
  - Each augmentation increases the value of the flow by some positive amount.
  - Augmentation can be done in $O(E)$.
  - Total *worst-case* running time $O(E|f^*|)$, where f^* is the max-flow found by the algorithm.
  - *Can we run into this worst-case*?
  - *Lesson: how an augmenting path is chosen is very important!*

Edmonds-Karp algorithm

- *Take* ***shortest path*** *(in terms of the number of edges) as an augmenting path – Edmonds-Karp algorithm*
  - *How do we find such a shortest path?*
  - Running time $O(VE^2)$, because the number of augmentations is $O(VE)$
  - To prove this we need to prove that:
    - *The length of the shortest path does not decrease*
    - *Each edge can become* ***critical*** *at most ~ $V/2$ times.* Edge (u,v) on an augmenting path p is critical if it has the minimum residual capacity in the path: $c_f(u,v) = c_f(p)$

Non-decreasing shortest paths

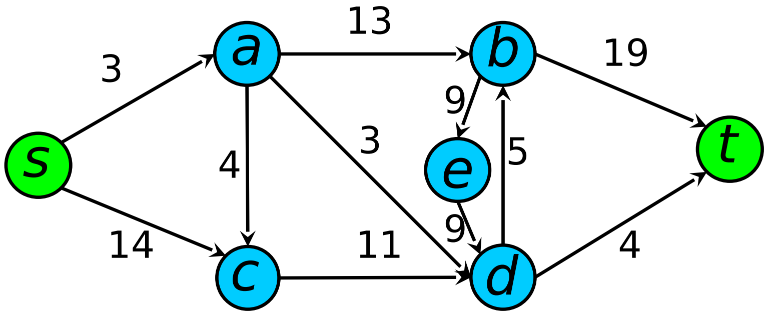
- *Why does the length of a shortest path from s to any v does not decrease?*
  - Observation: Augmentation may add some edges to residual network or remove some.
    - Removed edges obviously only increase the length of the shortest path
  - Only the added edges (“shortcuts”) may potentially decrease the length of a shortest path.
  - *Where do these edges are added*?
    - Opposite to some edge on the augmenting path, which is the shortest path!

Number of augmentations

- *Why each edge can become critical at most ~V/*2 *times*?
  - Scenario for edge (u,v):
    - Critical the first time: (u,v) on an augmenting path
    - Disappears from the residual network
    - Reappears on the residual network: (v,u) has to be on an augmenting path
    - We can show that in-between these events the distance from s to u increased by at least 2.
    - This can happen at most V/ 2 times
- We have proven that the running time of Edmonds-Karp is $O(VE^2)$.

Example of Edmonds-Karp

- *Run the Edmonds-Karp algorithm on the following graph:*



Linear programming and flow

- Many problems can be formulated as linear programs
  - Shortest path
  - Maximum flow
  - Other variants of it: e.g. *minimum-cost-flow problem*

$$\text{maximize} \quad \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs}$$

$$\text{subject to}$$

$$f_{uv} \leqslant c(u,v) \qquad \text{for each } u,v \in V$$

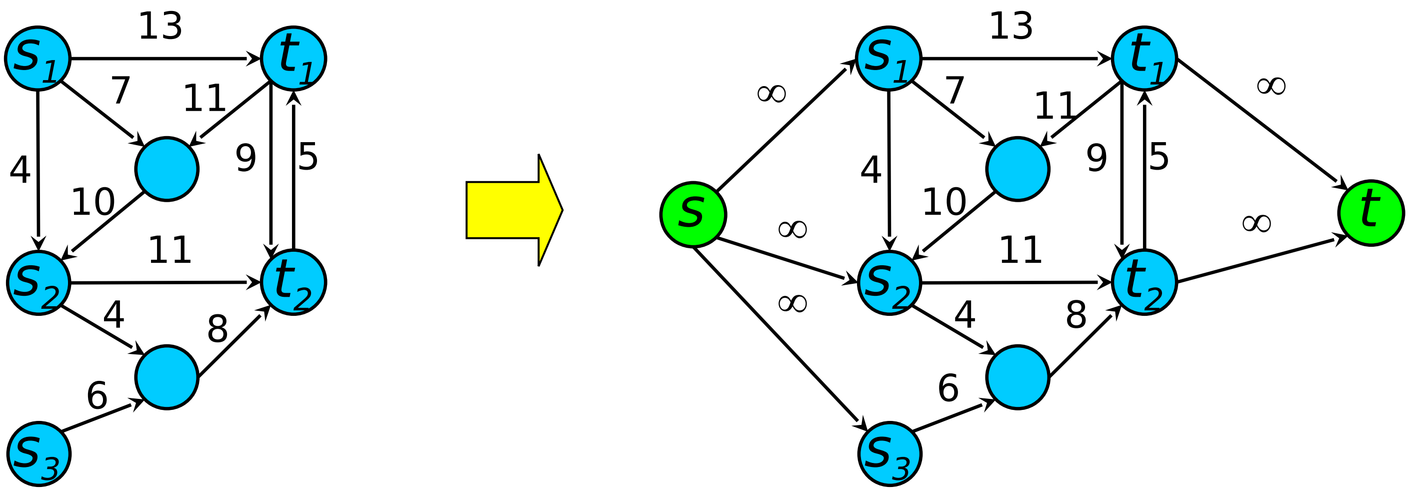
$$\sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} \quad \text{for each } u \in V - \{s,t\}$$

$$t_{uv} \geqslant 0 \qquad \text{for each } u,v \in V$$

  - Remember *duality*?
    - Here, informally: ***Minimize*** the sum of capacities on a cut, subject to it being a cut. ~= max-flow min-cut theorem!

Multiple sources or sinks

- *What if we have more sources or sinks?*
  - Augment the graph to make it with one source and one sink!



Application of max-flow

- *Maximum bipartite matching problem*
  - ***Matching*** in a graph is a subset M of edges such that each vertex has at most one edge of M incident on it. It puts vertices in pairs.
  - We look for *maximum* matching in a ***bipartite*** graph, where $V = L \cup R$, L and R are disjoint and all edges go between L and R
  - Dating agency example:
    - L – women, R – men.
    - An edge between vertices: they have a chance to be "compatible" (can be matched)
    - Do as many matches between "compatible" persons as possible

Maximum bipartite matching

- *How can we reformulate this problem to become a max-flow problem?*

- *What is the running time of the algorithm if we use the Ford-Fulkerson method?*