Lecture 12

Thursday, June 15, 2017 7:51 PM

Longest Common Subsequnce

Given: Two sequences x = x1 x2 … xn and y = y1 y2 … ym, over some alphabet A
Find: Longest common subsequence z = z1 … zk of x and y.

Example: x = TGACTA
y = GTGCATG

LCS z = TGCA, or TGAT, or TGCT.

Step 1: Array of optimal numerical values for sub-problems

$A(i,j) = \text{length of } LCS(x_1 \dots x_i, y_1 \dots y_j)$

$A(n,m)$

Step 2: Recurrence

$A(i,0) = A(0,j) = 0$

$$A(i,j) = \begin{cases} 1 + A(i-1,j-1) & \text{if } x_i = y_j \\ \max\{A(i-1,j), A(i,j-1)\} & \text{if } x_i \neq y_j \end{cases}$$

$x_1 \dots x_i$
$y_1 \dots y_j = C$

$LCS\ z_1 \dots z_k = C$

Step 3: Fill in the array

Step 4: Recover a solution (LCS) from the array by retracing.

Step 4: Recover a solution (LCS) from the array by retracing.

Longest Increasing Subsequence

Given: a sequence of integers a1, a2, …, an
Find: a longest increasing subsequence

Example: 7, 3, 18, 4, 2, 6
has 3, 4, 6 as a LIS

Step 1: Array

$A(i) =$ ~~length of LIS $(a_1, \ldots, a_i)$~~

$A(i) =$ length of LIS of $a_1 \ldots a_n$ that ends with a_i

Final answer $= \max_{1 \le i \le n} \{A(i)\}$

Step 2: Recurrence

$$A(i) = 1 + \max\{A(j) \mid 1 \le j < i \ \& \ a_j < a_i\}$$

$\ldots a_i$

Step 3: Fill in the array

Step 3: Fill in the array

Time : $O(n^2)$

Can be made $O(n \log n)$ with clever data structure.

Step 4: Use the array to find an actual LIS (by retracing).

DP Summary

- "Bottom up" approach, usually using an array of optimal values for sub-problems.
- Efficient recurrence to fill in the array ("Principle of Optimality")
- Can recover not just the optimal values but actual solutions achieving optimal values (by tracing through the array).

So far :

- BFS / DFS : templates for graph algos

- Greedy algorithms : algo type
- Divide & Conquer : algo type
- Dynamic Programming : algo type

Next:

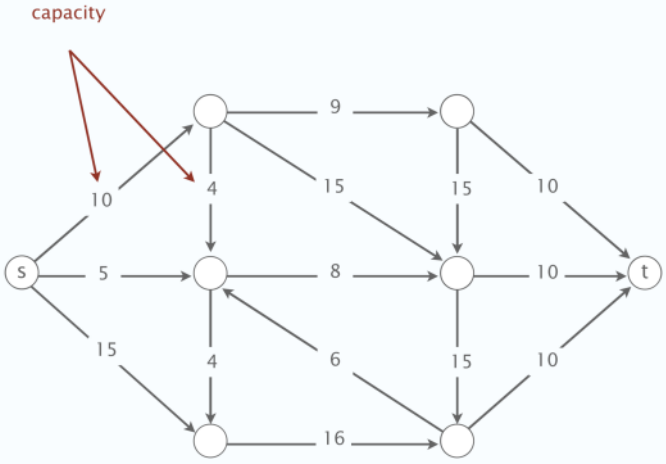
- Network Flow algo : single algo

but can be used to solve many other problems via reductions!

Flow network

- Abstraction for material flowing through the edges.
- Digraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$.
- Nonnegative integer capacity $c(e)$ for each $e \in E$.

no parallel edges
no edge enters s
no edge leaves t



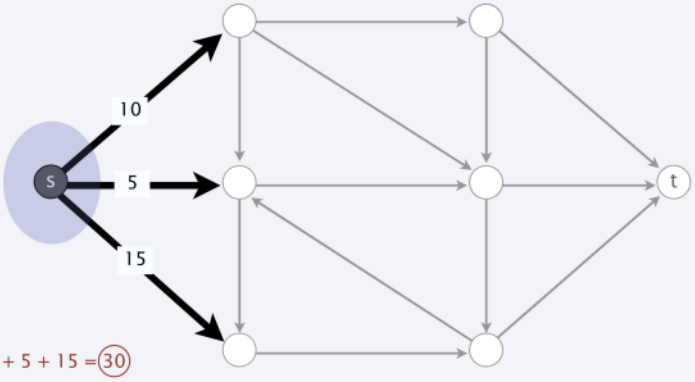
3

Minimum cut problem

Def. A *st*-cut (cut) is a partition (A, B) of the vertices with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



capacity = 10 + 5 + 15 = 30

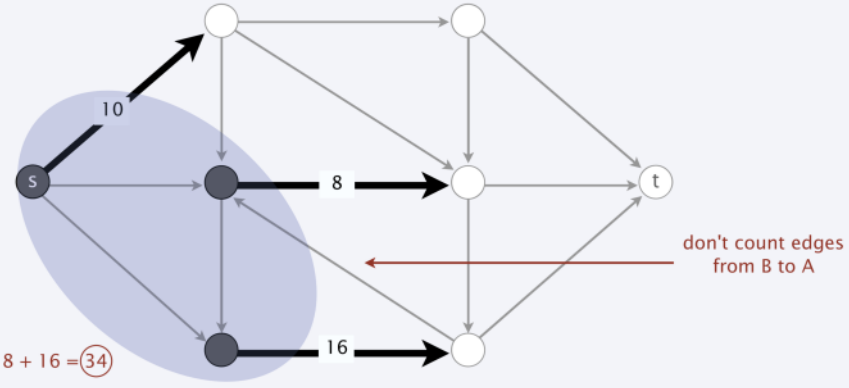
4

Minimum cut problem

Def. A st-cut (cut) is a partition (A, B) of the vertices with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



capacity = 10 + 8 + 16 = 34

5

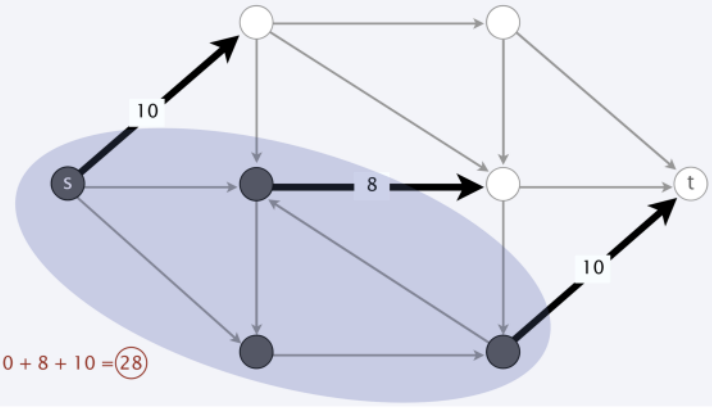
Minimum cut problem

Def. A st-cut (cut) is a partition (A, B) of the vertices with $s \in A$ and $t \in B$.

Def. Its capacity is the sum of the capacities of the edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

Min-cut problem. Find a cut of minimum capacity.

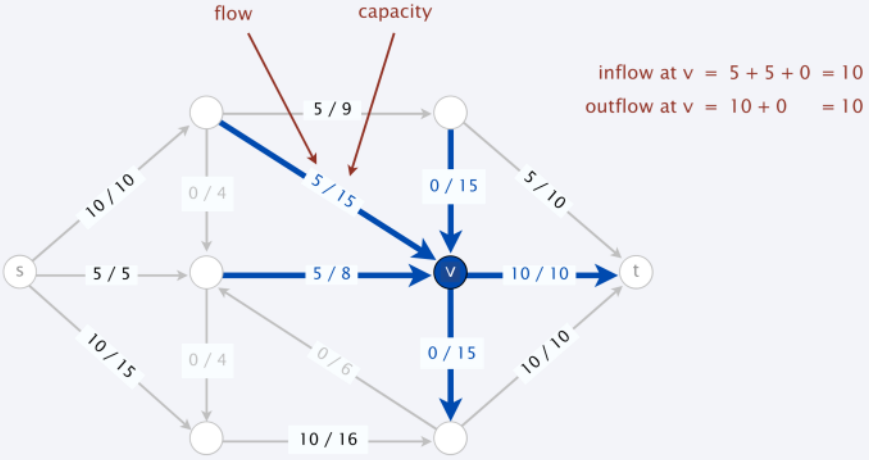


capacity = 10 + 8 + 10 = 28

Maximum flow problem

Def. An st-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

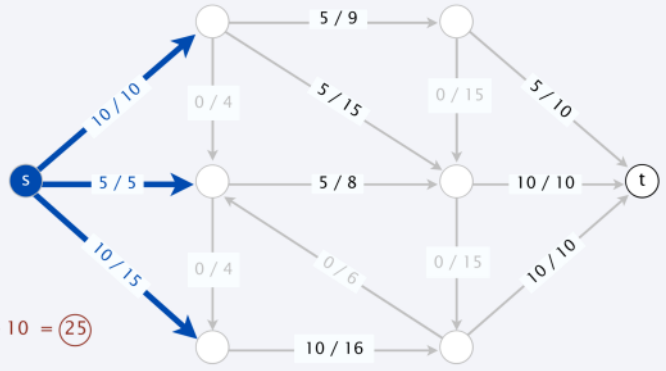


Maximum flow problem

Def. An st-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e)$.



value = 5 + 10 + 10 = 25

8

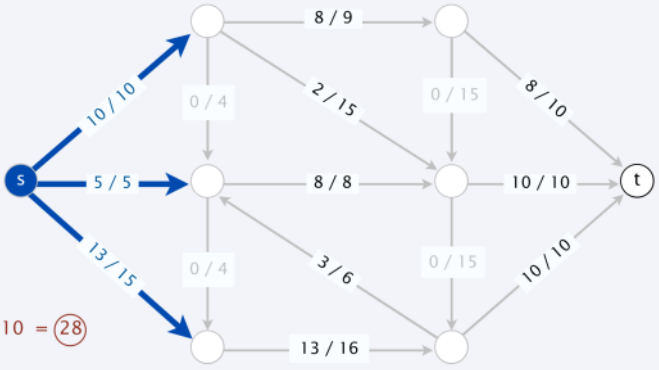
Maximum flow problem

Def. An *st*-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e)$.

Max-flow problem. Find a flow of maximum value.

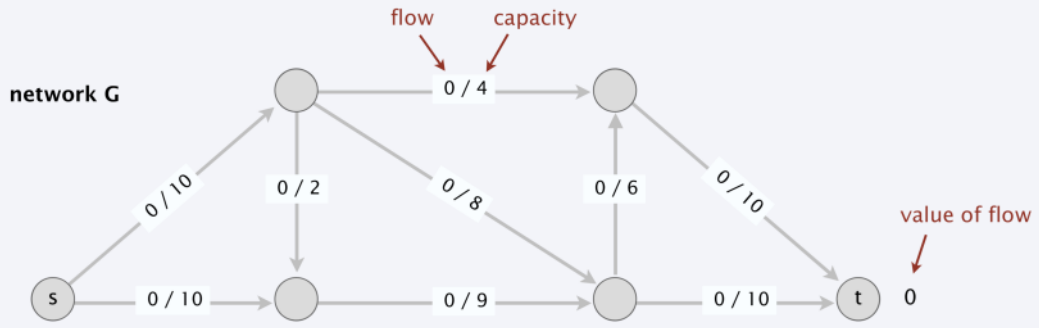


value = 8 + 10 + 10 = 28

Towards a max-flow algorithm

Greedy algorithm.

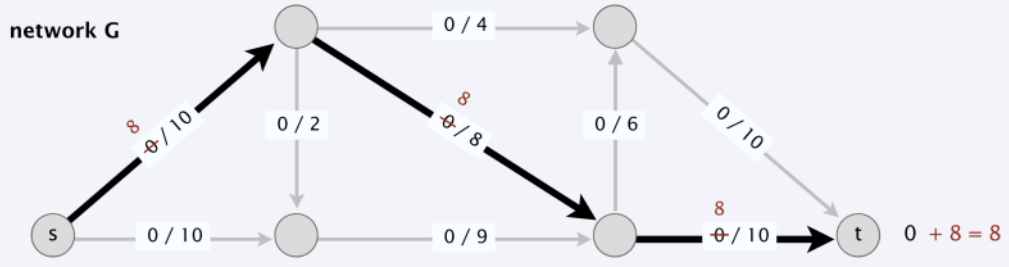
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Towards a max-flow algorithm

Greedy algorithm.

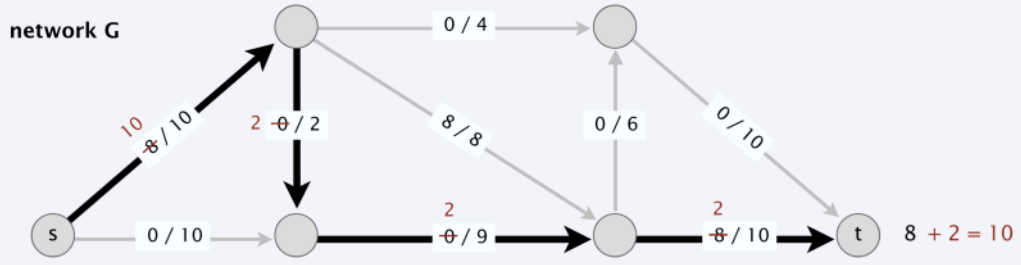
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Towards a max-flow algorithm

Greedy algorithm.

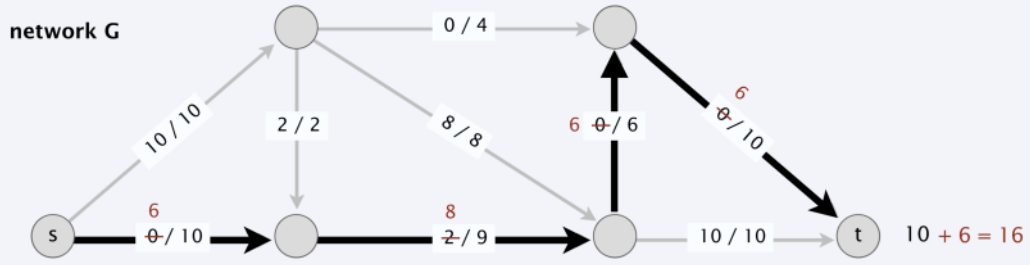
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Towards a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

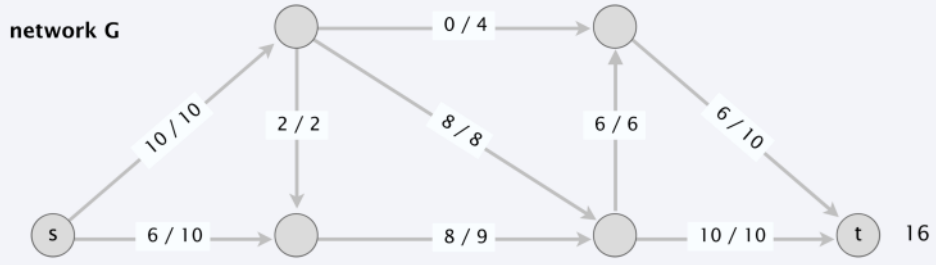


Towards a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

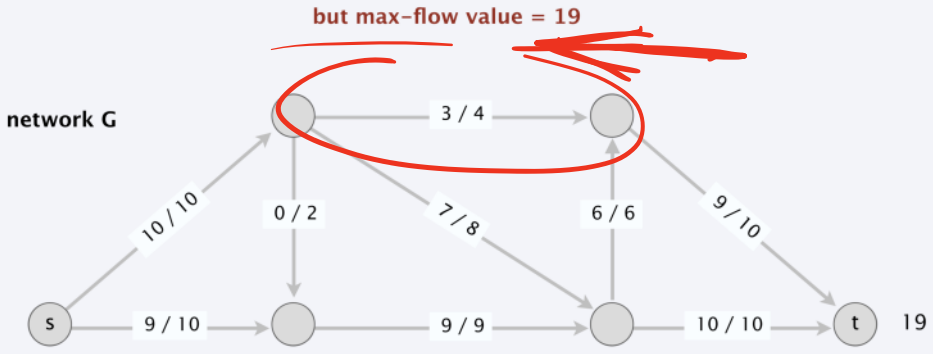
ending flow value = 16



Towards a max-flow algorithm

Greedy algorithm.

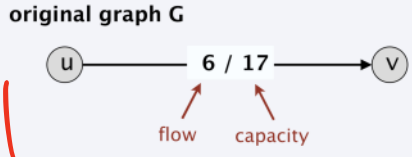
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Residual graph

Original edge: $e = (u, v) \in E$.

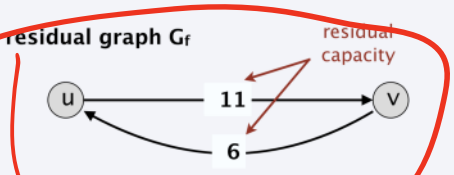
- Flow $f(e)$.
- Capacity $c(e)$.



Residual edge.

- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

where flow on a reverse edge negates flow on a forward edge

17

Augmenting path

Def. An augmenting path is a simple $s \rightarrow t$ path P in the residual graph G_f.

Def. The bottleneck capacity of an augmenting P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f. Then f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

AUGMENT (f, c, P)

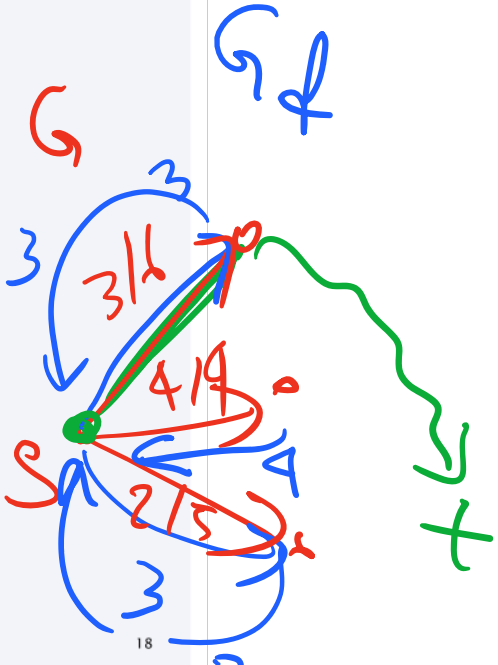
$b \leftarrow$ bottleneck capacity of path P.

FOREACH edge $e \in P$

  IF $(e \in E)$ $f(e) \leftarrow f(e) + b$.

  ELSE $f(e^R) \leftarrow f(e^R) - b$.

RETURN f.

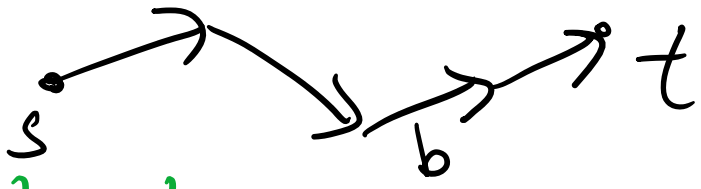


18

(1) Is the new flow legal?

(2) Is the new flow better? Does value increase?

(1) G^f

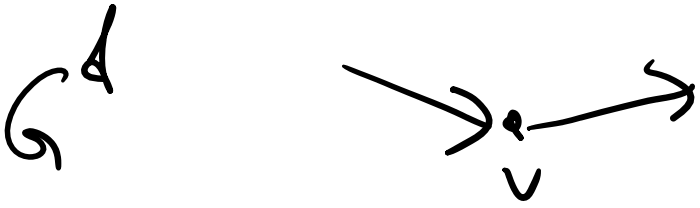


- capacity constraints ✓

$b \geq$ min res. cap.

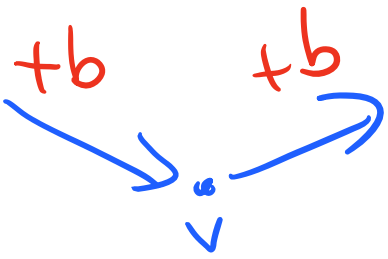
G s ~ ~ . . ~ ~ ~ t

- flow conservation

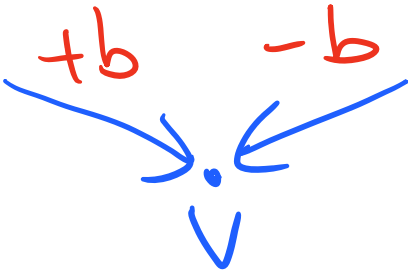


Cases:

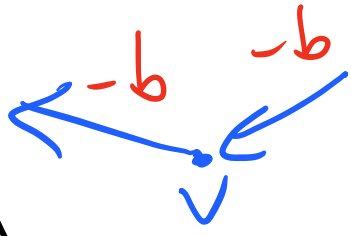
G



both forward edges



1 forward
1 backward

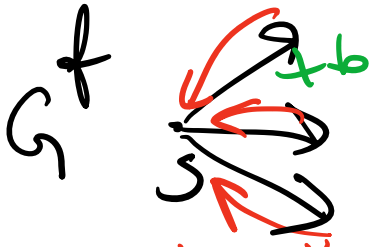


2 backward edges

(2) $val(f') = f_1 + f_2 + f_3$

G

G^f



1st edge on s-t path in G^f is FORWARD edge

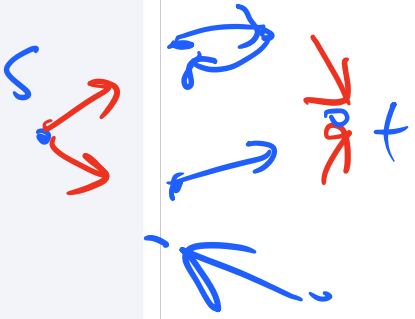
Ford-Fulkerson algorithm

Ford-Fulkerson augmenting path algorithm.

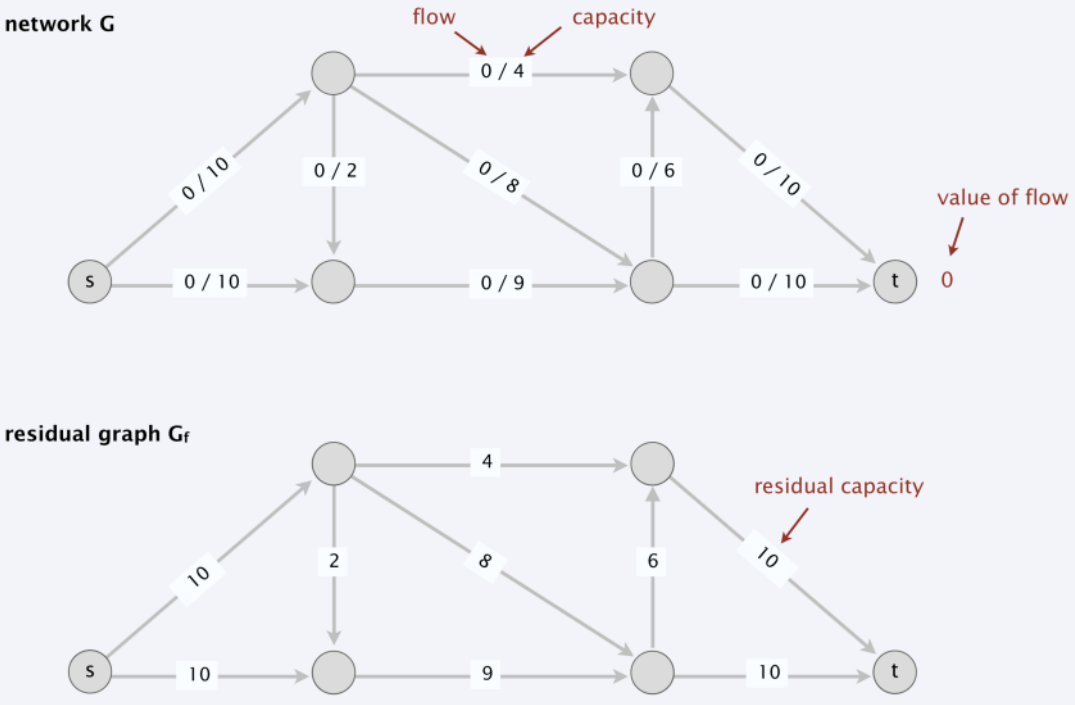
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an augmenting path P in the residual graph G_f.
- Augment flow along path P.
- Repeat until you get stuck.

```
FORD-FULKERSON (G, s, t, c)

FOREACH edge e ∈ E : f(e) ← 0.
   G_f ← residual graph.
   WHILE (there exists an augmenting path P in G_f)
      f ← AUGMENT (f, c, P).
      Update G_f.
   RETURN f.
}
```

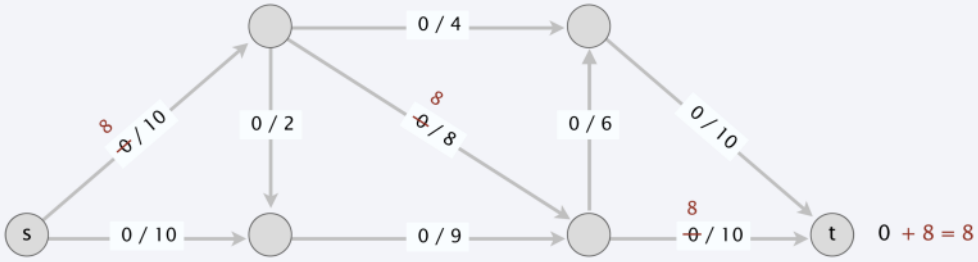


Ford-Fulkerson algorithm demo

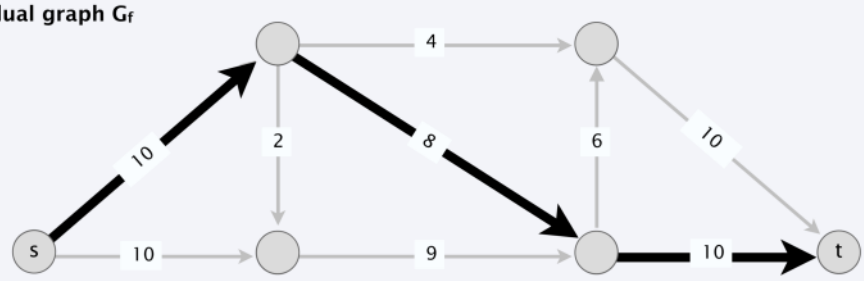


Ford-Fulkerson algorithm demo

network G



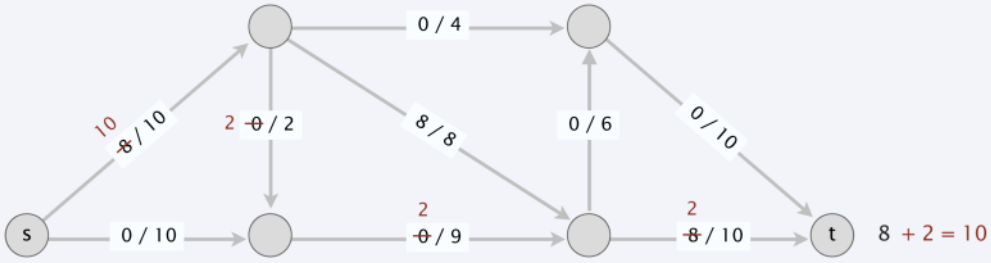
residual graph G_f



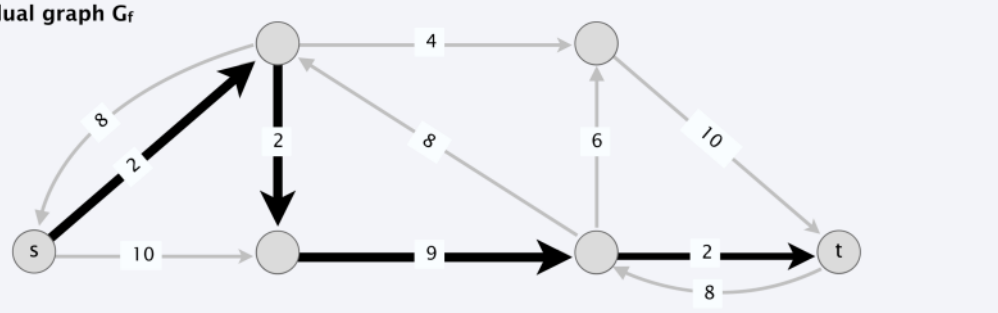
21

Ford-Fulkerson algorithm demo

network G



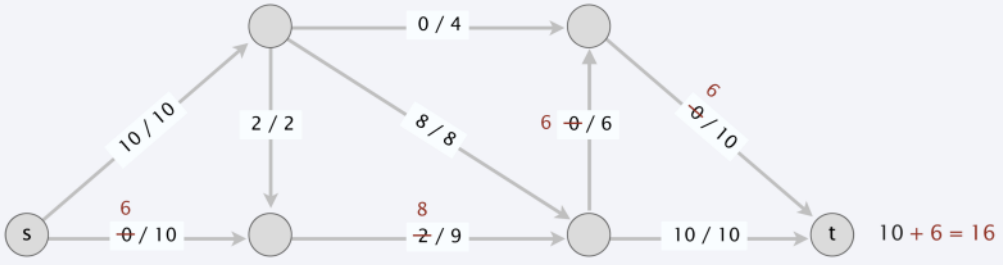
residual graph G_f



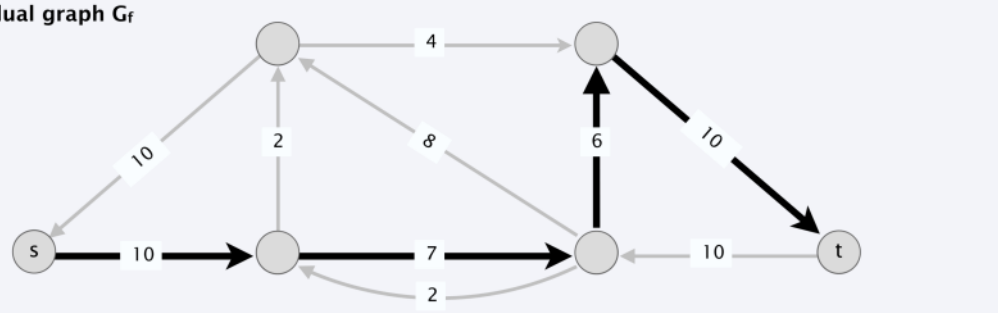
22

Ford-Fulkerson algorithm demo

network G



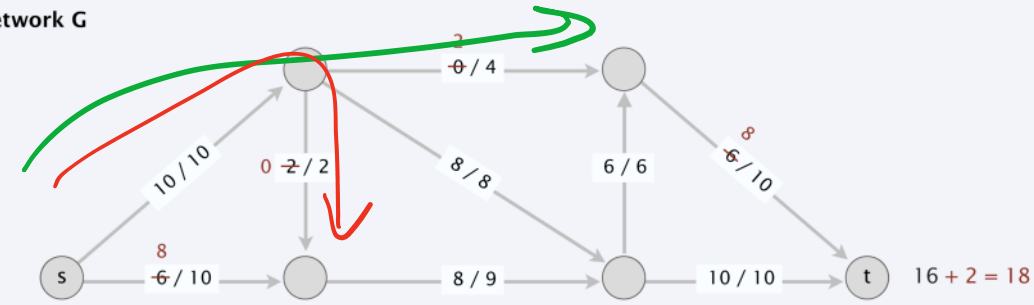
residual graph G_f



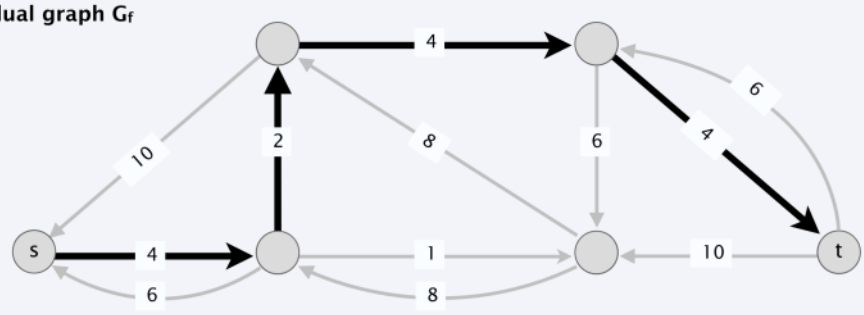
23

Ford-Fulkerson algorithm demo

network G



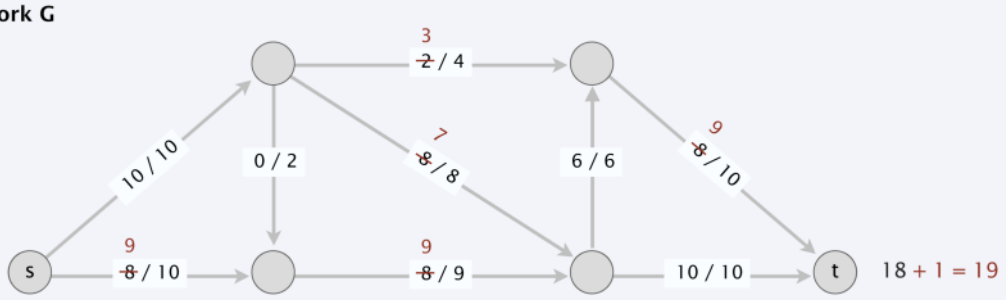
residual graph G_f



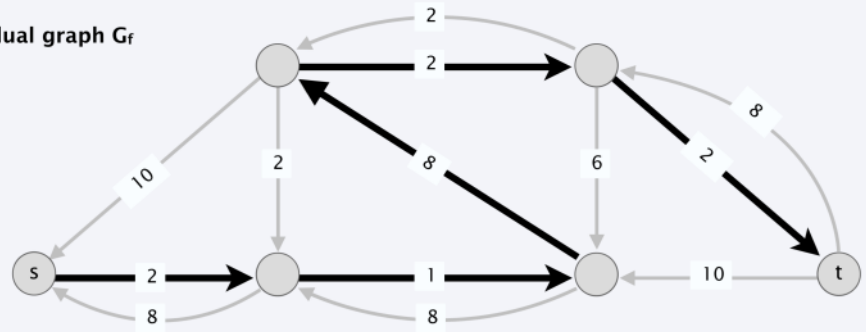
24

Ford-Fulkerson algorithm demo

network G



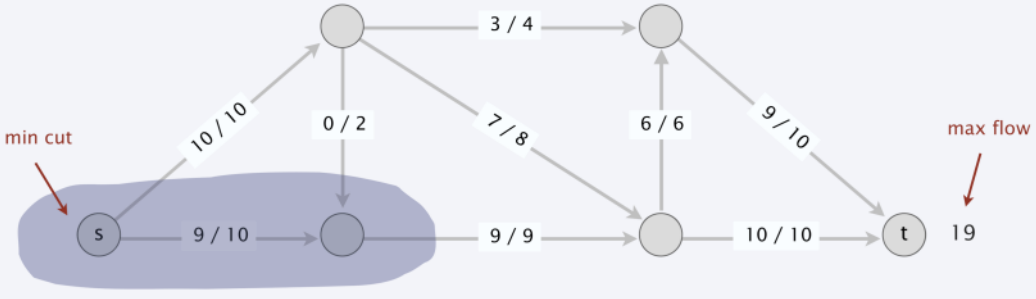
residual graph G_f



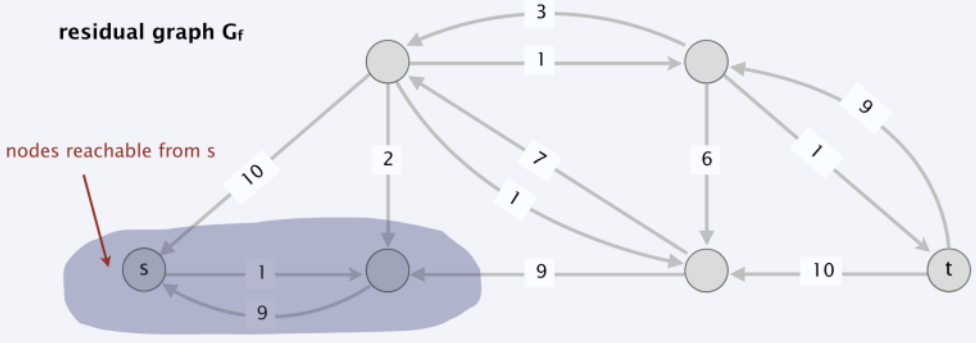
25

Ford-Fulkerson algorithm demo

network G



residual graph G_f

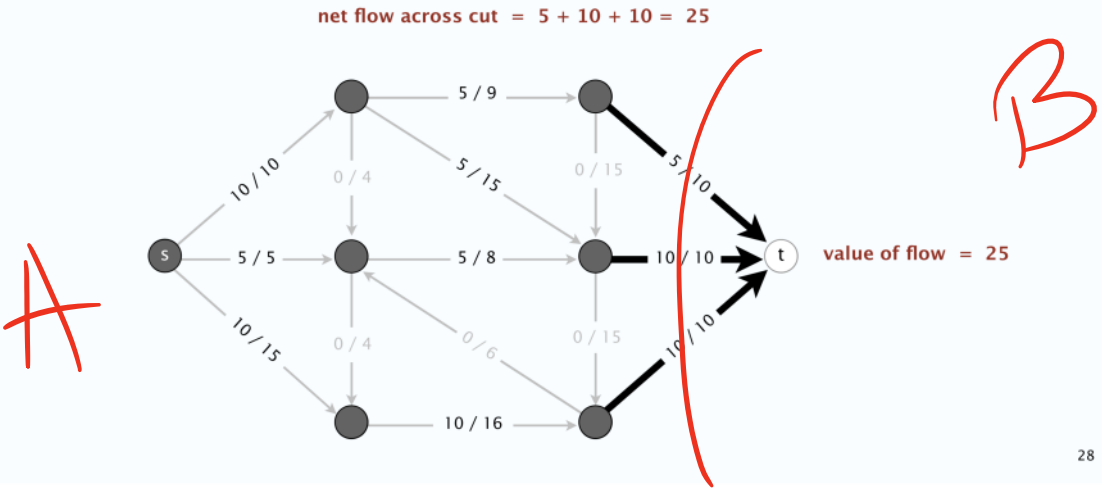


26

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f.

$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



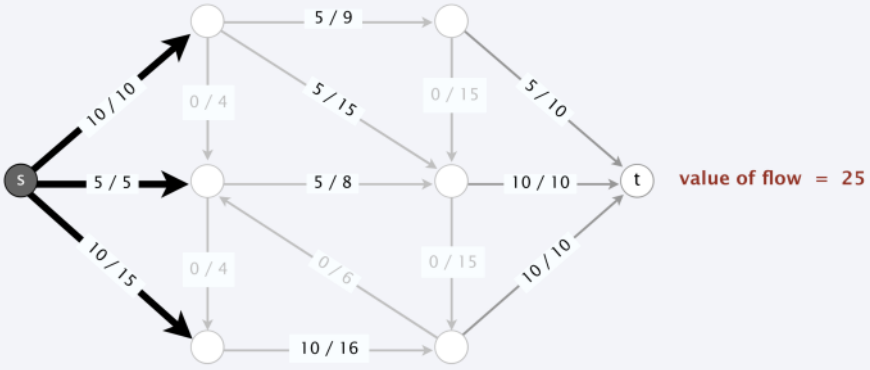
28

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f.

$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$

net flow across cut = 10 + 5 + 10 = 25



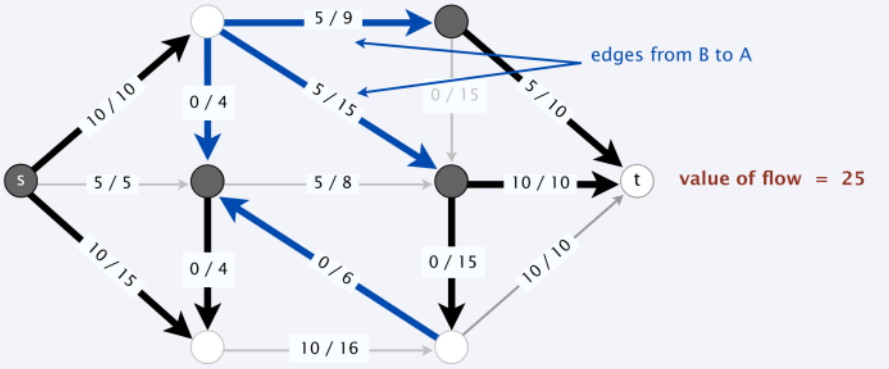
29

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f.

$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$

net flow across cut = (10 + 10 + 5 + 10 + 0 + 0) – (5 + 5 + 0 + 0) = 25



30

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f.

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) = v(f)$$

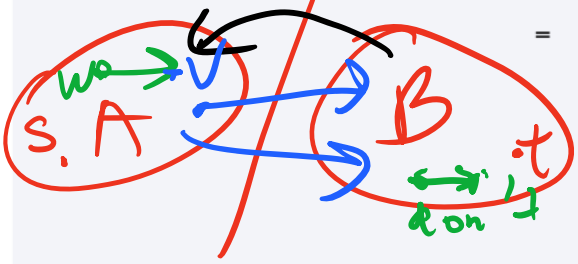
Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

by flow conservation, all terms except v = s are 0

$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \right)$$

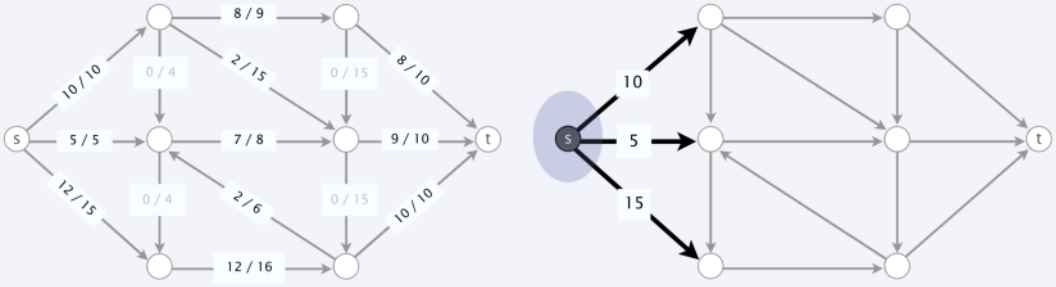
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e). \quad \blacksquare$$



Relationship between flows and cuts

Weak duality. Let f be any flow and (A, B) be any cut. Then, $v(f) \le cap(A, B)$.

Pf.
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$
flow-value lemma
$$\le \sum_{e \text{ out of } A} f(e)$$
$$\le \sum_{e \text{ out of } A} c(e)$$
$$= cap(A, B) \quad \blacksquare$$



value of flow = 27 ≤ capacity of cut = 30

32

Max-flow min-cut theorem

Augmenting path theorem. A flow f is a max-flow iff no augmenting paths.

Max-flow min-cut theorem. Value of the max-flow = capacity of min-cut.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.

ii. f is a max-flow.

iii. There is no augmenting path with respect to f.

[i ⇒ ii]

- Suppose that (A, B) is a cut such that $cap(A, B) = val(f)$.
- Then, for any flow f', $val(f') \le cap(A, B) = val(f)$.
- Thus, f is a max-flow. ▪

weak duality — by assumption

33

Max-flow min-cut theorem

Augmenting path theorem. A flow f is a max-flow iff no augmenting paths.
Max-flow min-cut theorem. Value of the max-flow = capacity of min-cut.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
ii. f is a max-flow.
iii. There is no augmenting path with respect to f.

[ii ⇒ iii] We prove contrapositive: ~iii ⇒ ~ii.

- Suppose that there is an augmenting path with respect to f.
- Can improve flow f by sending flow along this path.
- Thus, f is not a max-flow. ▪

34

Max-flow min-cut theorem

[iii ⇒ i]

- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual graph G_f.
- By definition of cut A, $s \in A$.
- By definition of flow f, $t \notin A$.

$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e)$$

flow-value lemma

$$= \sum_{e \text{ out of } A} c(e)$$

$$= cap(A, B) \quad \blacksquare$$