Announcements

Announcements

- Sign up for exam time if you haven't yet done so!
- Exam questions will be posted on Sunday
  - Some material from today/Tuesday will be included, so you may not be able to do all of it immediately
- Same procedure as last time
- Any questions?

greedy, DP, network flows (LP)

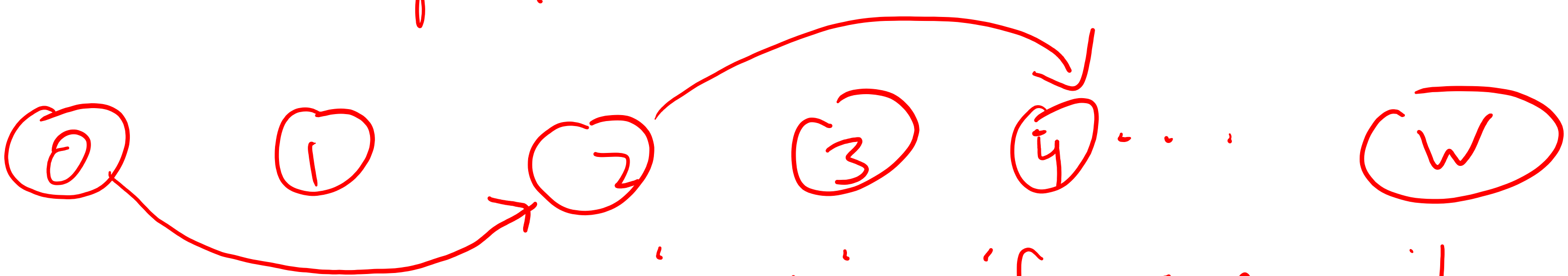
KP w/out rep
All-pairs s.p.

2-dimensional DAG / 2-d array

Dynamic Programming

KP w/repetition

longest path from 0



$i \to j$ if some item had weight $= j - i$, weight of edge = value of item

Knapsack without Repetition

- Same as before, except now you can't have multiple copies of an item.
- Are subproblems from before still useful?

what is max val I can get with a total weight of exactly i?

not useful now, because it doesn't track which items already used

Knapsack without Repetition

- Same as before, except now you can't have multiple copies of an item.
- Are subproblems from before still useful? total weight
- Define $K(w, j)$ = maximum value achievable with capacity of w, ~~using~~ choosing only from items $1, \ldots, j$
- What is the value of $K(w, j)$?

Knapsack without Repetition

- Same as before, except now you can't have multiple copies of an item.
- Are subproblems from before still useful?
- Define *K*(*w*, *j*) = maximum achievable with capacity of *w*, using only items 1, …, *j*
- What is the value of *K*(*w*, *j*)?

choose to include item j

$$K(w,j) = \max\{K(w-w_j, j-1) + v_j, K(w, j-1)\}$$

do not include item j

Knapsack without Repetition

$w_1 = 1$ $K(w,3)$ $K(1,1)$ depends on

$K(w,4)$ $K(w-w_4,3)$ $K(1,0)$ and $K(1-w_1,0)$

Initialize all $K(0,j)=0$ and all $K(w,0)=0$

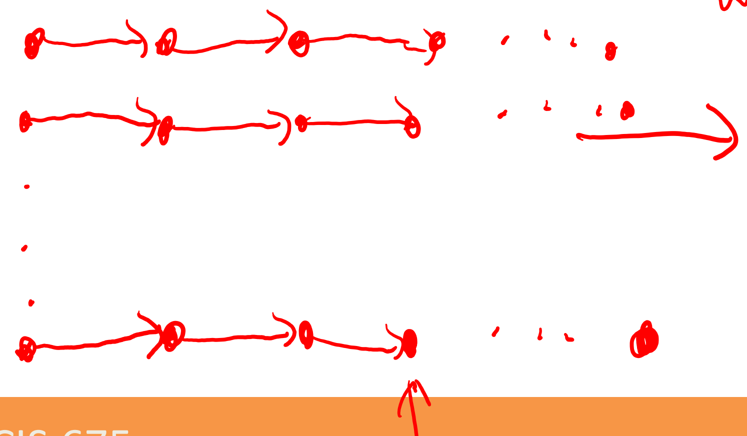
for $j=1$ to n: n iterations

  for $w=1$ to W: W iterations $O(n \cdot W)$

    if $w_j > w$: $K(w,j) = K(w,j-1)$

    else: $K(w,j) = \max\{K(w,j-1), K(w-w_j,j-1)+v_j\}$

return $K(W,n)$

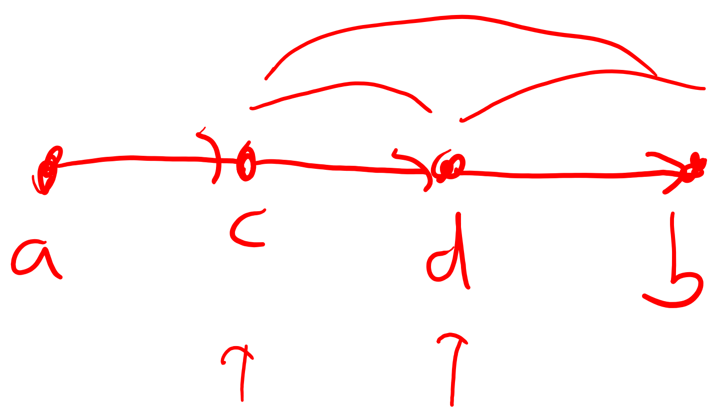


$w\downarrow$ $j\rightarrow$	0	1	2	3	...	N
0	0	0	0	0	...	0
1	0	•	•			
2	0					
⋮	⋮					
W	0					

- Given the adjacency matrix of a weighted, directed graph *G*
- We want to find the shortest path length between *each pair* of nodes

S_{ij} = length of s.p. $i \to j$

- One solution: run Dijkstra's from each node as source
  - Too slow!



All-Pairs Shortest Paths: the Floyd-Warshall Algorithm

- Idea: use a shortest path matrix M to keep track of all-pairs shortest path lengths (APSPLs)
- M^i corresponds to the state of matrix M at the beginning of step i
- Define an *intermediate vertex* on a shortest path as a node that is not the source or destination
- Subproblem: In each ~~step~~ iteration i, we want to find the APSPLs that use only nodes $1, \ldots, i$ as intermediate nodes

All-Pairs Shortest Paths: the Floyd-Warshall Algorithm

- Subproblem: In each step *i*, we want to find the APSPLs that use only nodes *1, …, i*
- What is the initial value of *M*?
- What is the update equation for M^i_{jk}?

M^0 represents cases where we are not allowed to use any int. verts.

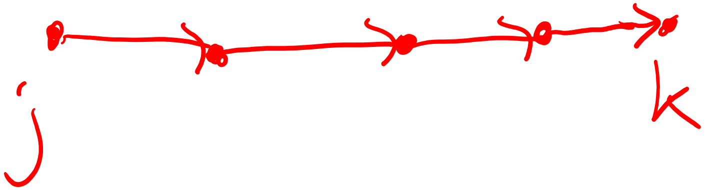
M^0_{ij} = weight of edge (i,j) if such an edge exists, ∞ otherwise

All-Pairs Shortest Paths: the Floyd-Warshall Algorithm

M^x for $x < i$

- What is the update equation for M^i_{jk}?

s.p.l. from $j \to k$ using only nodes from $\{1, \ldots, i\}$ as inter. verts.

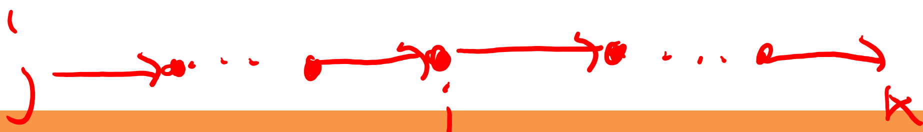


N iterations
each iter is
$O(N^2)$ time,
total
$O(N^3)$

2 possibilities:

1. don't use node i, then $M^i_{jk} = M^{i-1}_{jk}$

2. do use node i. The path looks like

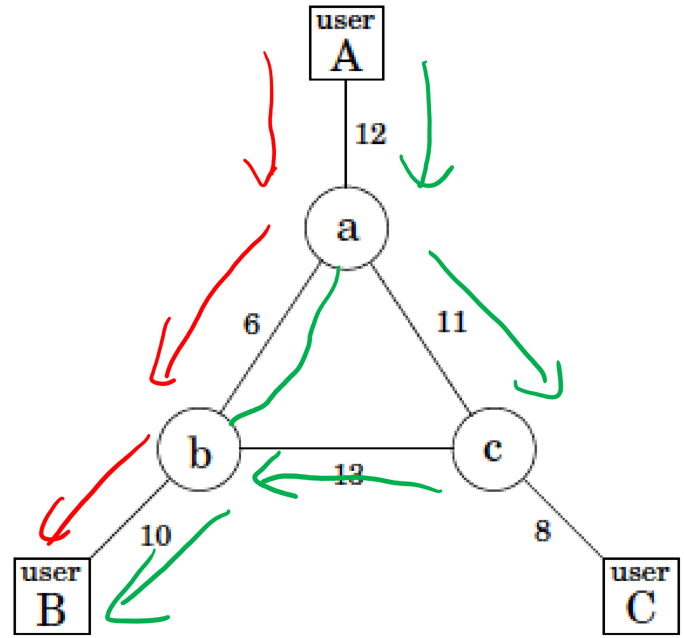


$M^i_{jk} = M^{i-1}_{ji} + M^{i-1}_{ik}$

Network Flows

Allocating Bandwidth

- Need to allocate bandwidth to connect A-B, B-C, and A-C
- A-B pays $3 per unit of bandwidth, B-C pays $2, and A-C pays $4
- Can connect via long path or short path
- Each ~~connection~~ pair of users must have at least 2 units of bandwidth
- Each edge has a maximum capacity



Allocating Bandwidth

- Each connection can use the long path or short path: consider each possibility

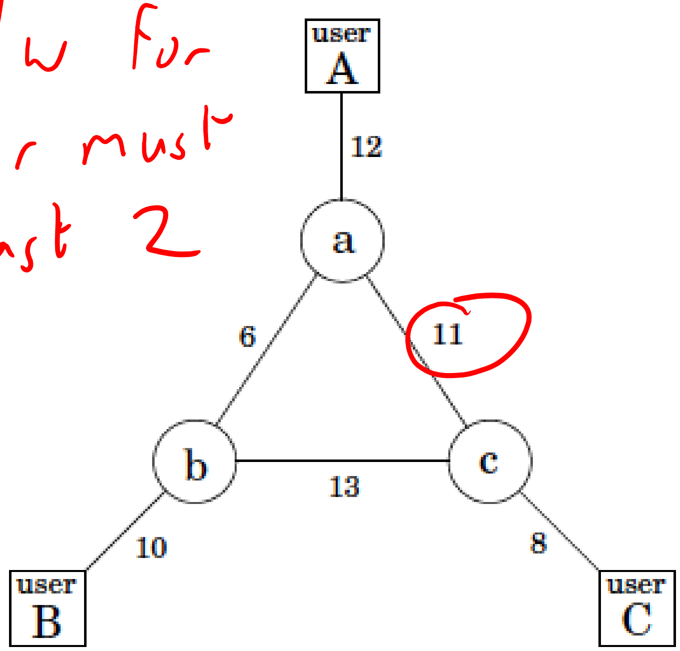
x_{AB} = amount of b/w alloc. to A/B along short path

obj. function

$$\max\ 3x_{AB} + 3x'_{AB} + 2x_{BC} + 2x'_{BC} + 4x_{AC} + 4x'_{AC}$$

$$x_{AB} + x'_{AB} \geq 2$$
$$x_{BC} + x'_{BC} \geq 2$$
$$x_{AC} + x'_{AC} \geq 2$$

total b/w for each pair must be at least 2



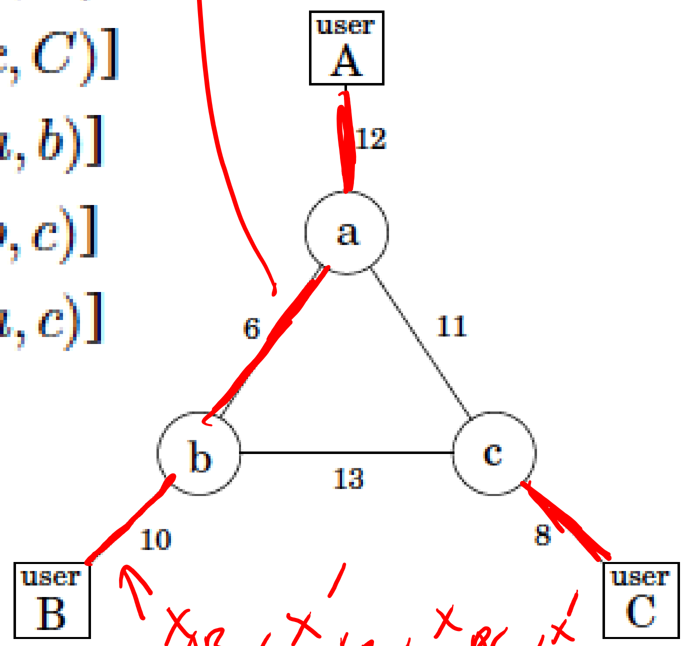
x'_{AB} = amount allocated to A/B along long path

Allocating Bandwidth

- There are bandwidth constraints on each edge

$$
\begin{aligned}
x_{AB} + x'_{AB} + x_{BC} + x'_{BC} &\le 10 && \text{[edge } (b, B)] \\
x_{AB} + x'_{AB} + x_{AC} + x'_{AC} &\le 12 && \text{[edge } (a, A)] \\
x_{BC} + x'_{BC} + x_{AC} + x'_{AC} &\le 8 && \text{[edge } (c, C)] \\
x_{AB} + x'_{BC} + x'_{AC} &\le 6 && \text{[edge } (a, b)] \\
x'_{AB} + x_{BC} + x'_{AC} &\le 13 && \text{[edge } (b, c)] \\
x'_{AB} + x'_{BC} + x_{AC} &\le 11 && \text{[edge } (a, c)]
\end{aligned}
$$

x_{AB}
x'_{AC}
x'_{BC}



$x_{AB}, x'_{AB}, x_{BC}, x'_{BC}$

Allocating Bandwidth: Putting it Together

$$\max \quad 3x_{AB} + 3x'_{AB} + 2x_{BC} + 2x'_{BC} + 4x_{AC} + 4x'_{AC}$$

$$\begin{aligned}
x_{AB} + x'_{AB} + x_{BC} + x'_{BC} &\le 10 \\
x_{AB} + x'_{AB} + x_{AC} + x'_{AC} &\le 12 \\
x_{BC} + x'_{BC} + x_{AC} + x'_{AC} &\le 8 \\
x_{AB} + x'_{BC} + x'_{AC} &\le 6 \\
x'_{AB} + x_{BC} + x'_{AC} &\le 13 \\
x'_{AB} + x'_{BC} + x_{AC} &\le 11 \\
x_{AB} + x'_{AB} &\ge 2 \\
x_{BC} + x'_{BC} &\ge 2 \\
x_{AC} + x'_{AC} &\ge 2 \\
x_{AB}, x'_{AB}, x_{BC}, x'_{BC}, x_{AC}, x'_{AC} &\ge 0
\end{aligned}$$

capacity constraints

Network Flows

- What happens as the number of nodes in the previous problem becomes large?

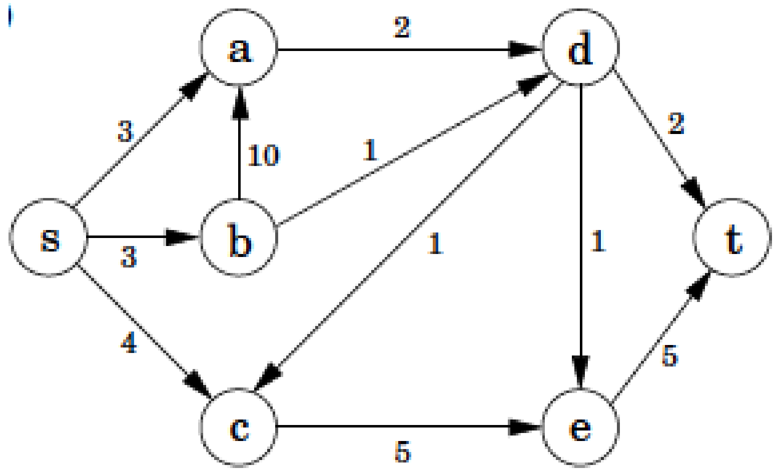
number of constraints ↑

not an efficient solution!

- Better solution: network flow algorithms

Network Flows

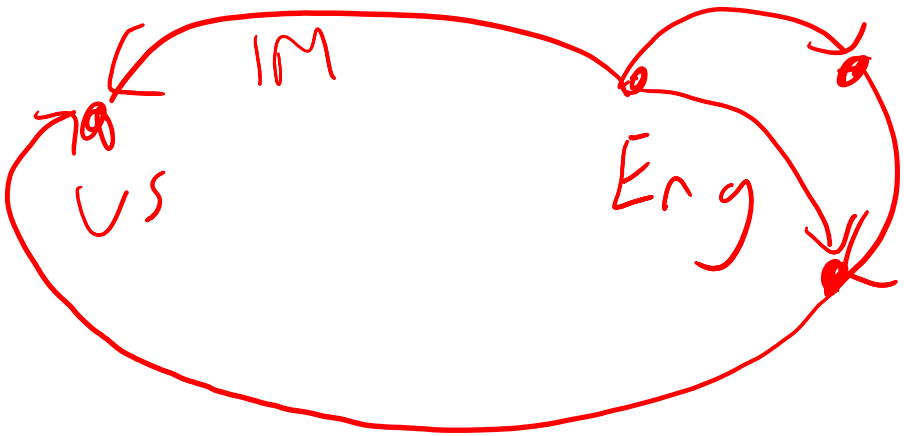
- You are given a network with capacities on the edges

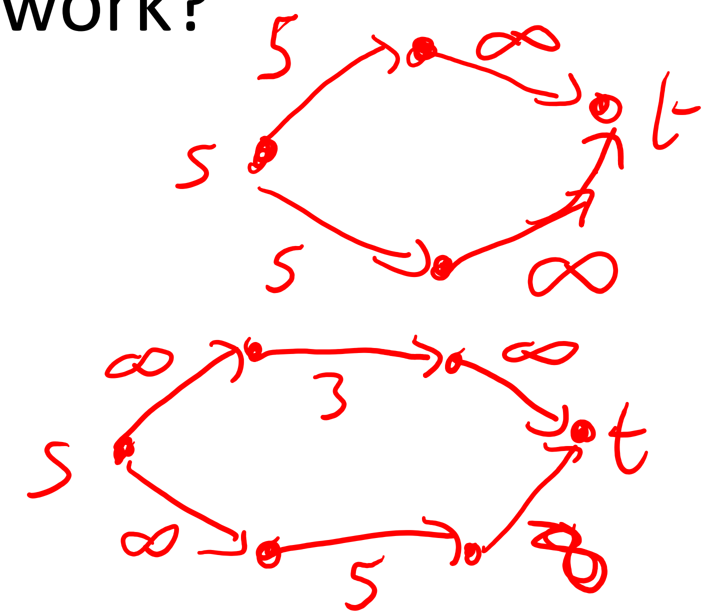


- Goal is to decide how much flow to send along each edge, to maximize total amount from s to t

Examples of Network Flows

- How many products can we ship from England to the US, given the current set of flights?
- How much data can we send from one computer to another, given current network?





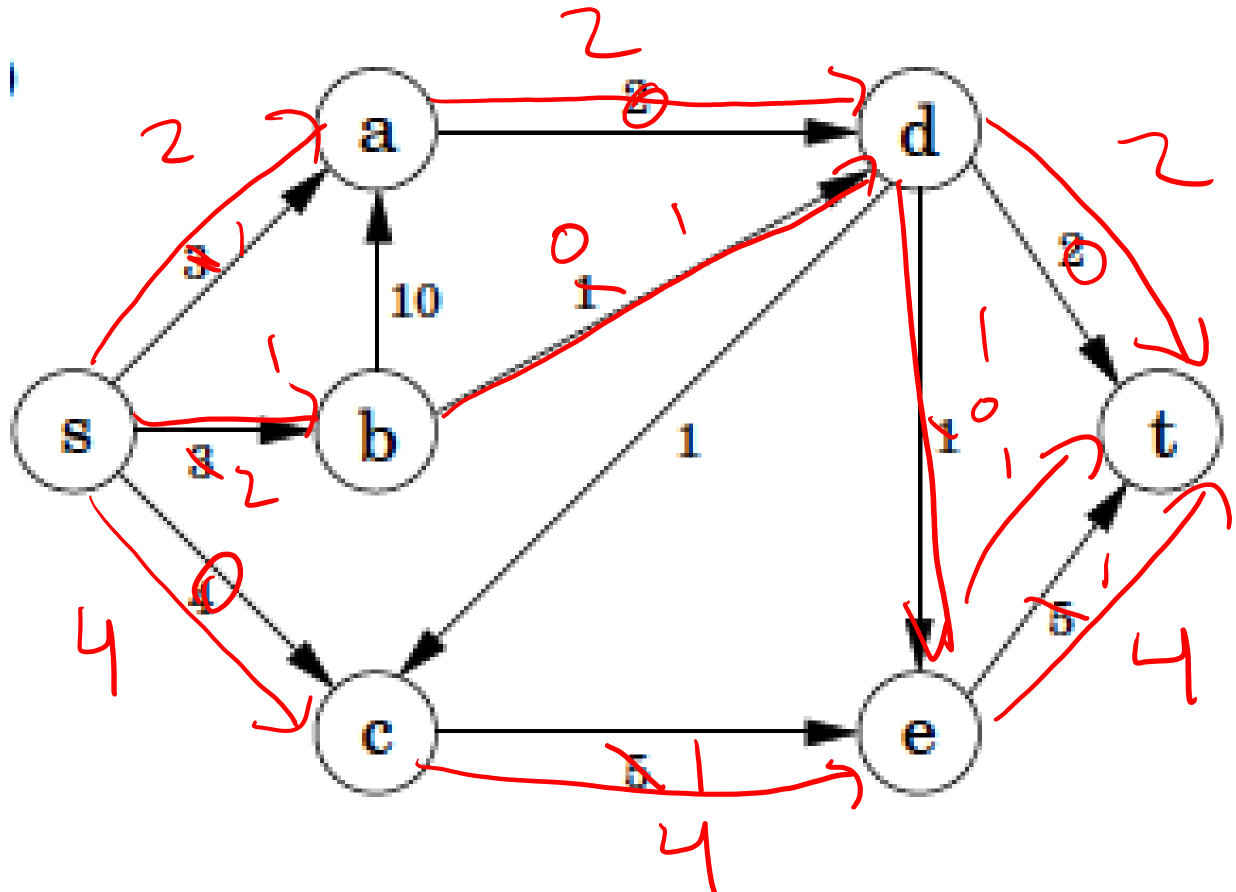
Rules of Network Flows

- The flow sent along an edge cannot exceed the total capacity of that edge constraint
- The total amount of flow entering a node must equal the amount of flow leaving a node, excepting source + target/sink
- This is a linear programming problem!

not efficient to view it like this!

In-Class Exercise

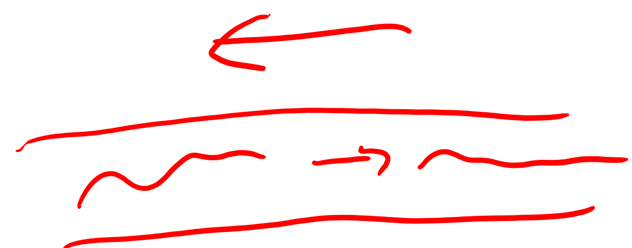
Find the maximum amount of flow from s to t



7

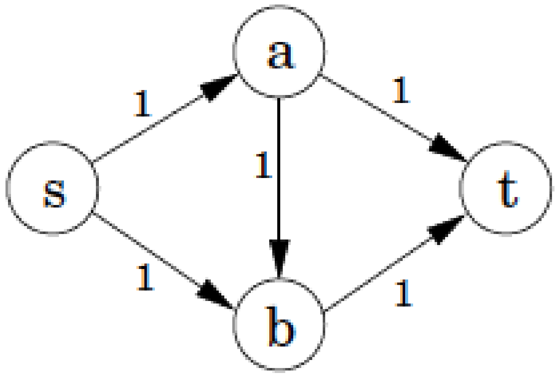
The Ford-Fulkerson Network Flow Algorithm

- Choose a path from S to T – doesn't matter which path (except for efficiency)
- Send as much flow as possible along that path – min capacity of an edge along path
- Rules of path selection:
  - You can use an edge if there is capacity left
  - You can *reverse* flow sent along an edge

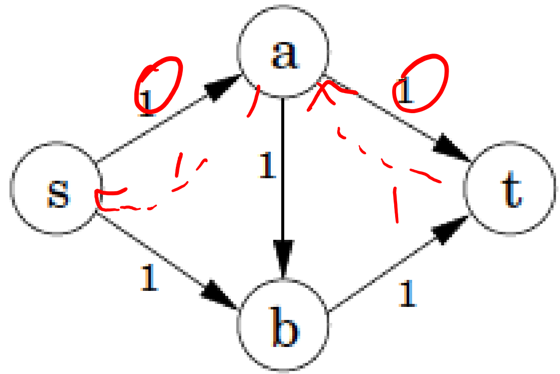


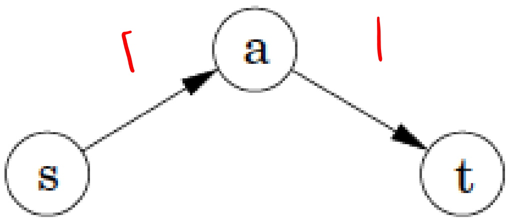
10 gallons

Sketch of Network Flow Algorithm

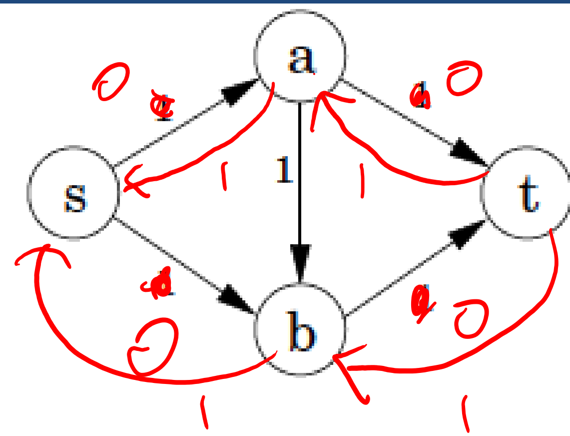


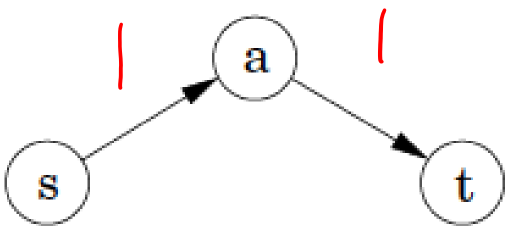
Sketch of Network Flow Algorithm

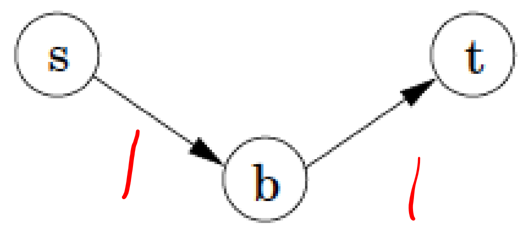




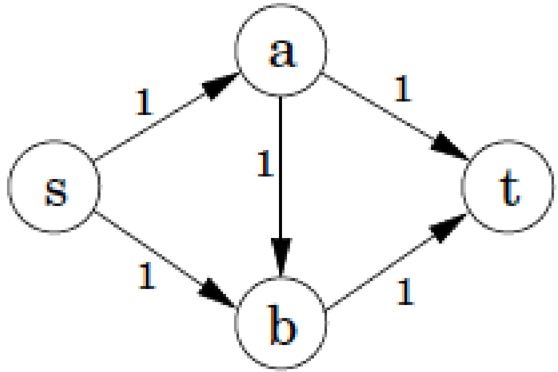
Sketch of Network Flow Algorithm



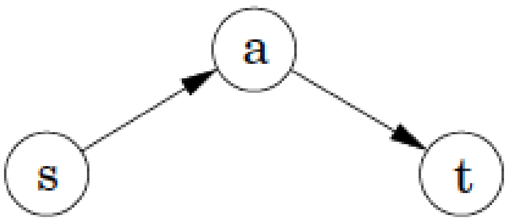


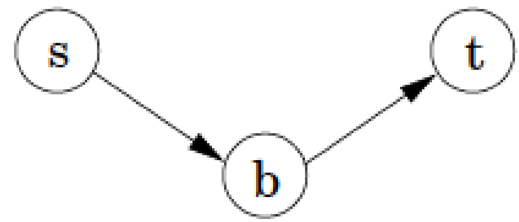


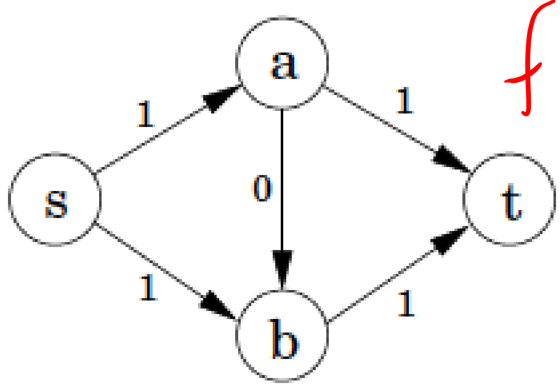
Sketch of Network Flow Algorithm



capacity graph

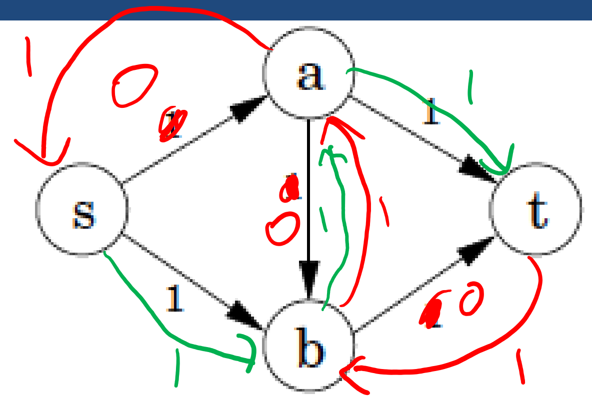


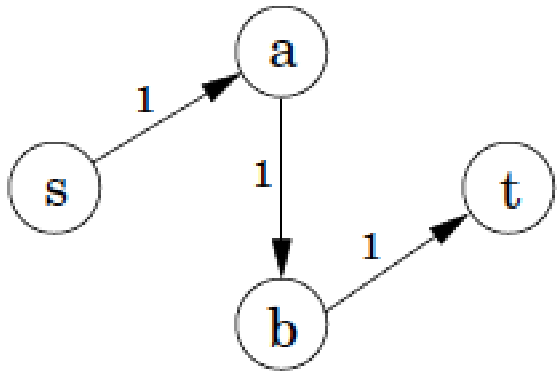




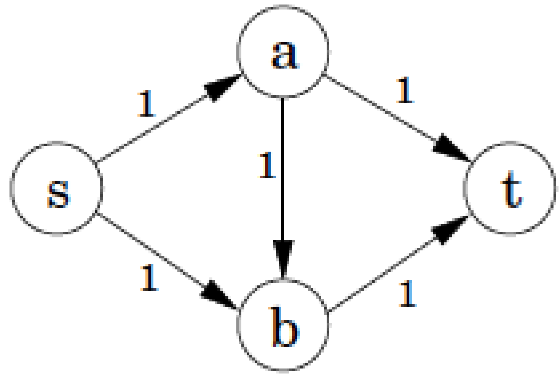
flow graph

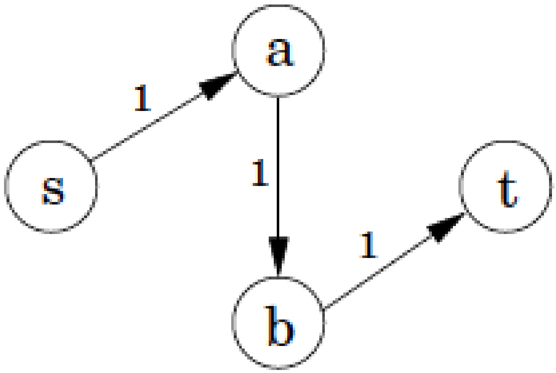
Sketch of Network Flow Algorithm

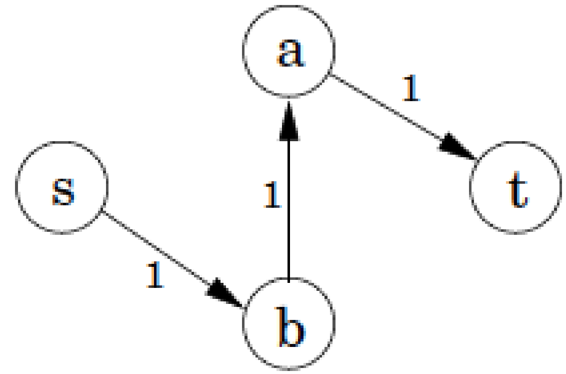




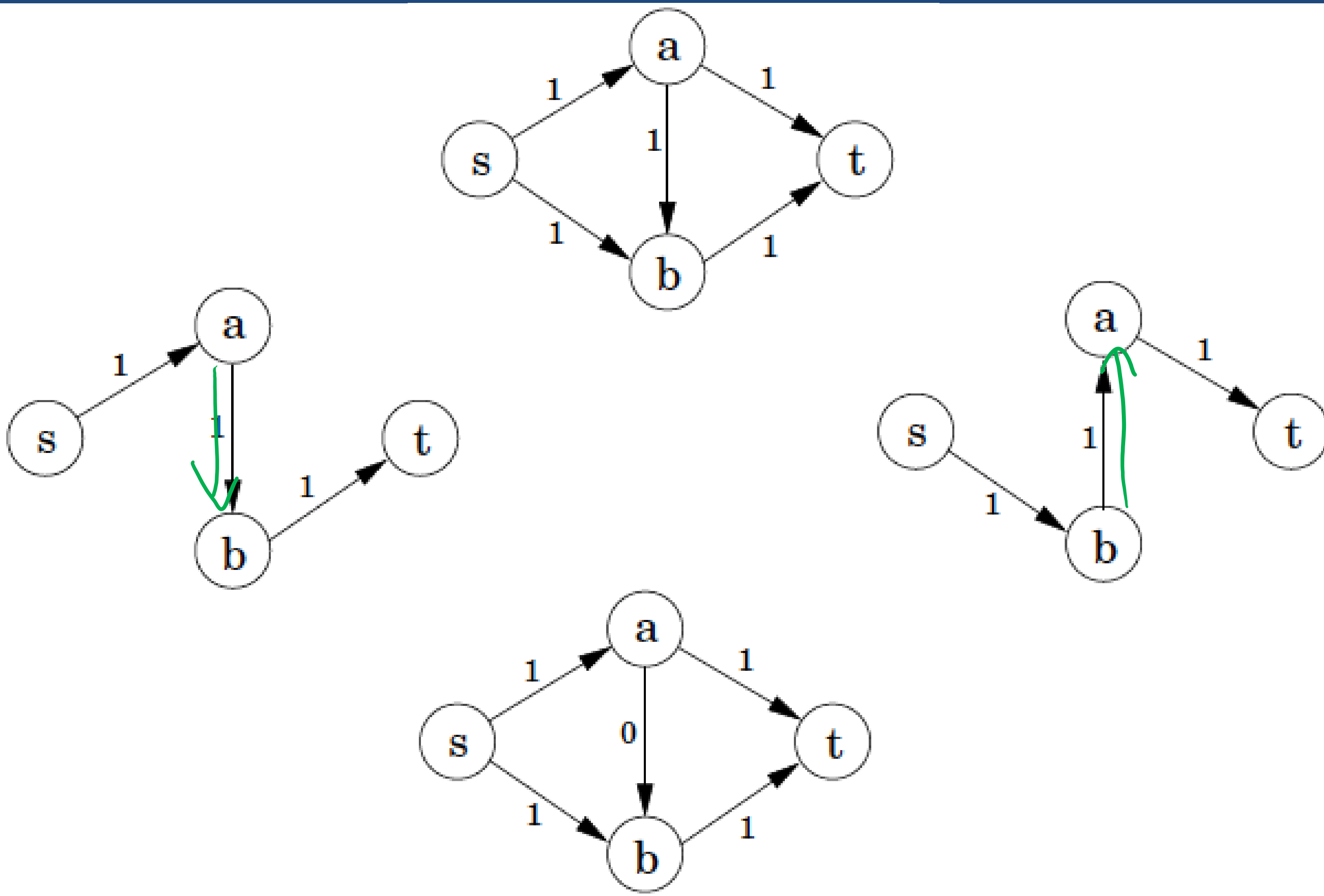
Sketch of Network Flow Algorithm







Sketch of Network Flow Algorithm



Network Flow Example

Find the maximum amount of flow from s to t