CSCI 3104: Algorithms

Lecture 17: Ford–Fulkerson Algorithm, Bipartite Matching Intro to Dynamic Programming

Rachel Cox

Department of Computer Science

What will we learn today?

- ❑ Intro to Dynamic Programming
- ❑ Ford-Fulkerson Algorithm
- ❑ Maximum Bipartite Matching

Intro to Algorithms, CLRS:
Sections 26.1, 26.2, 26.3, 15.1

Ford_Fulkerson

FORD-FULKERSON-METHOD(G, s, t)

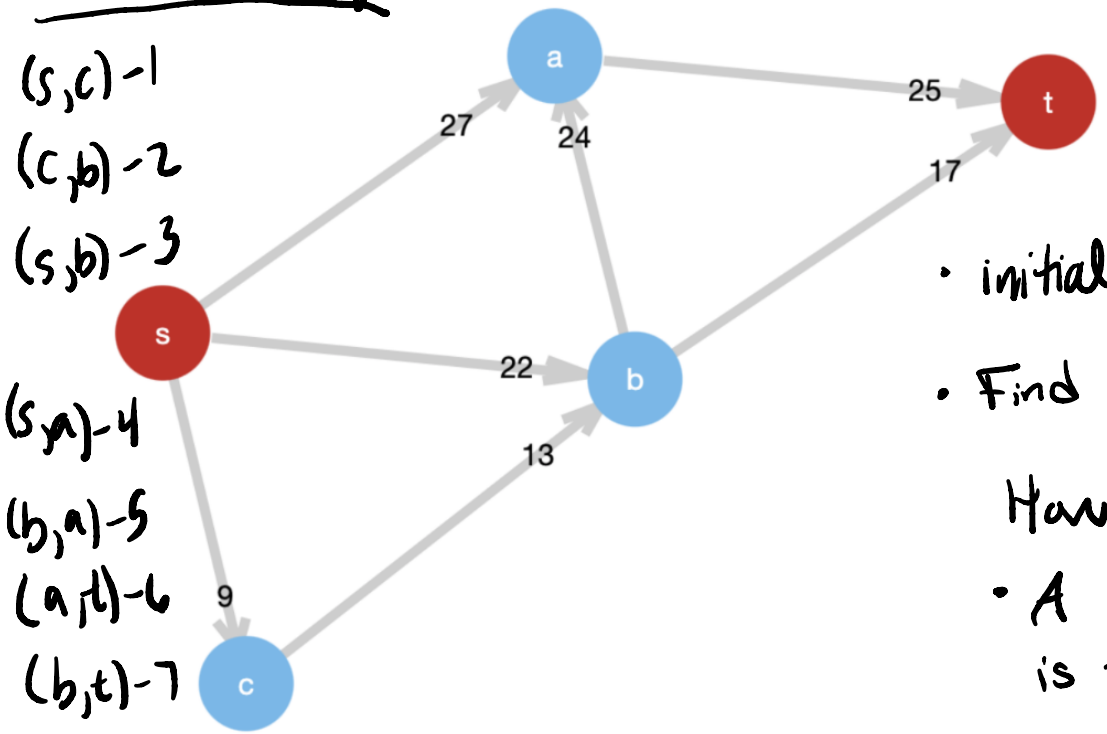
1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f

- Initialize $f(e) = 0 \quad \forall e \in G$
  While there is an s-t path in G_f
  Let P be a simple s-t path in G_f
  $f' = augment(P, f)$
  Set $f = f'$
  Set $G_f = G_f$, from augment
  Return f

while loop exits once we can no longer find a simple path.

number the edges

(s,c)-1
(c,b)-2
(s,b)-3
(s,a)-4
(b,a)-5
(a,t)-6
(b,t)-7



- initialize flow = 0 $f = [0, 0, 0, 0, 0, 0, 0]$
- Find an initial path from source → target.

How to find? BFS, DFS

- A common way to choose an augmenting path is to pick the path with least number of edges in G_f. Suppose our first path: $s \to a \to t$

Ford_Fulkerson

FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f

· 1ST iteration of while loop

S→a→t

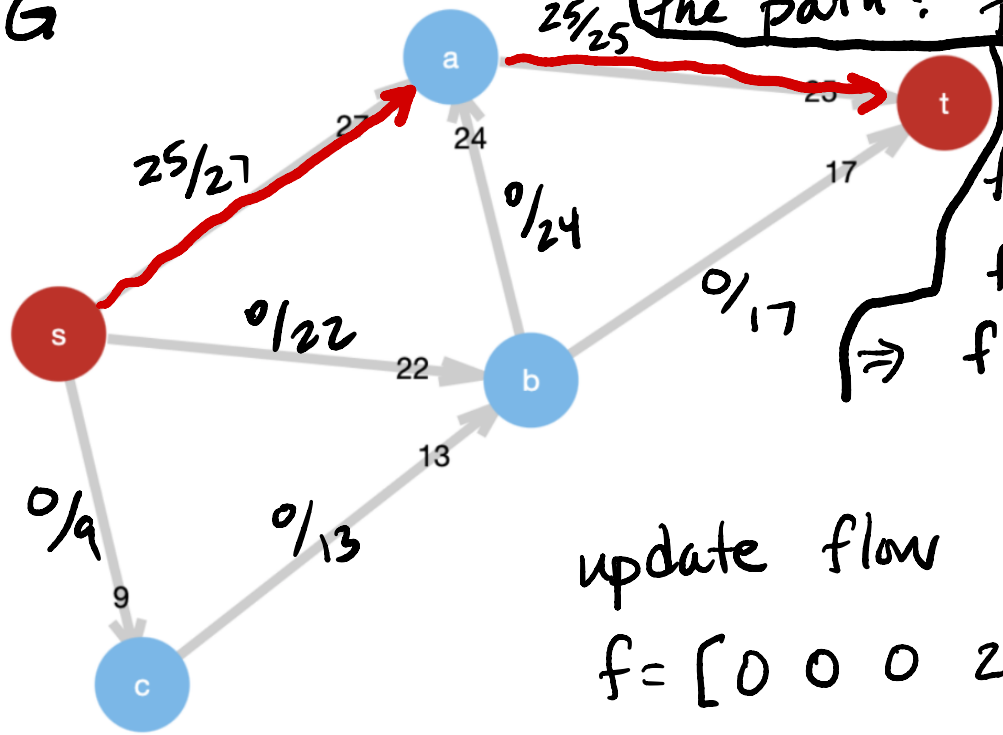
· How do you compute the flow on the path? Take minimum capacity of edges.

flow$_{(s,a)}$ = 27

flow$_{(a,t)}$ = 25

⇒ f = min(25, 27) = 25

G

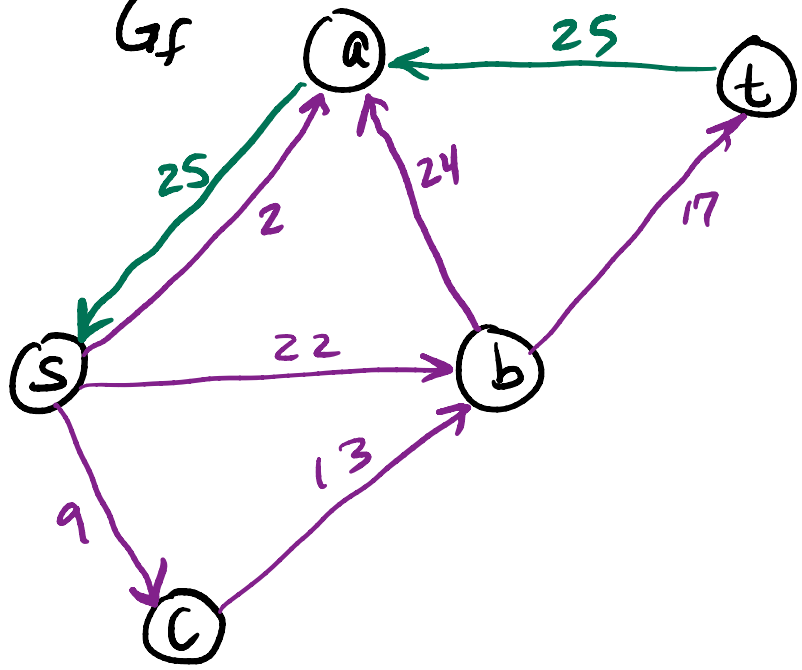


update flow list:

f = [0 0 0 25 0 25 0]

Initialize $f(e) = 0 \quad \forall e \in G$
While there is an s-t path in G_f
Let P be a simple s-t path in G_f
$f' = augment(P, f)$
Set $f = f'$
Set $G_f = G_f$, from augment
Return f

G_f



Next path: s→b→t

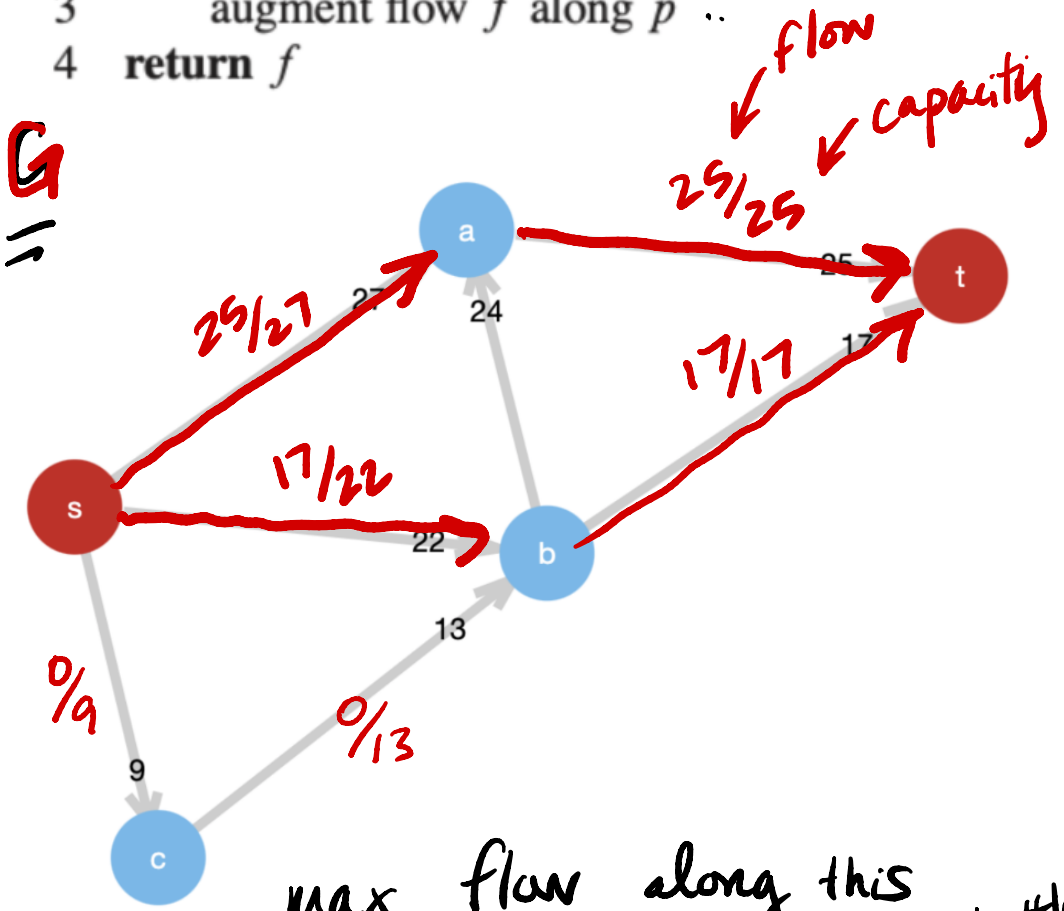
Ford–Fulkerson

FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f

2nd iteration

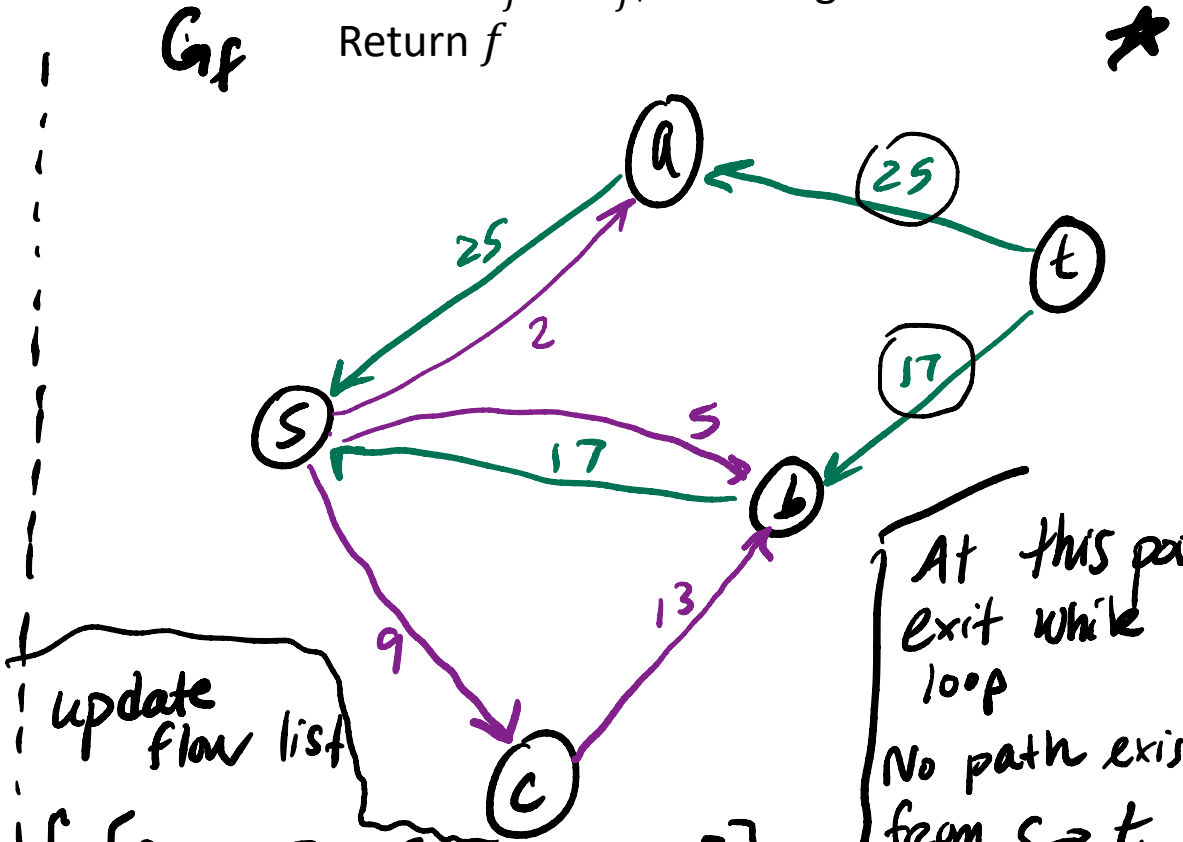
Initialize $f(e) = 0 \quad \forall e \in G$
While there is an s-t path in G_f
Let P be a simple s-t path in G_f
$f' = augment(P, f)$
Set $f = f'$
Set $G_f = G_f$, from augment
Return f



S→b→t

max flow along this path = min (22, 17)

"bottleneck capacity"



update flow list

f=[0 0 17 25 0 25 17]

At this point exit while loop

No path exists from s→t

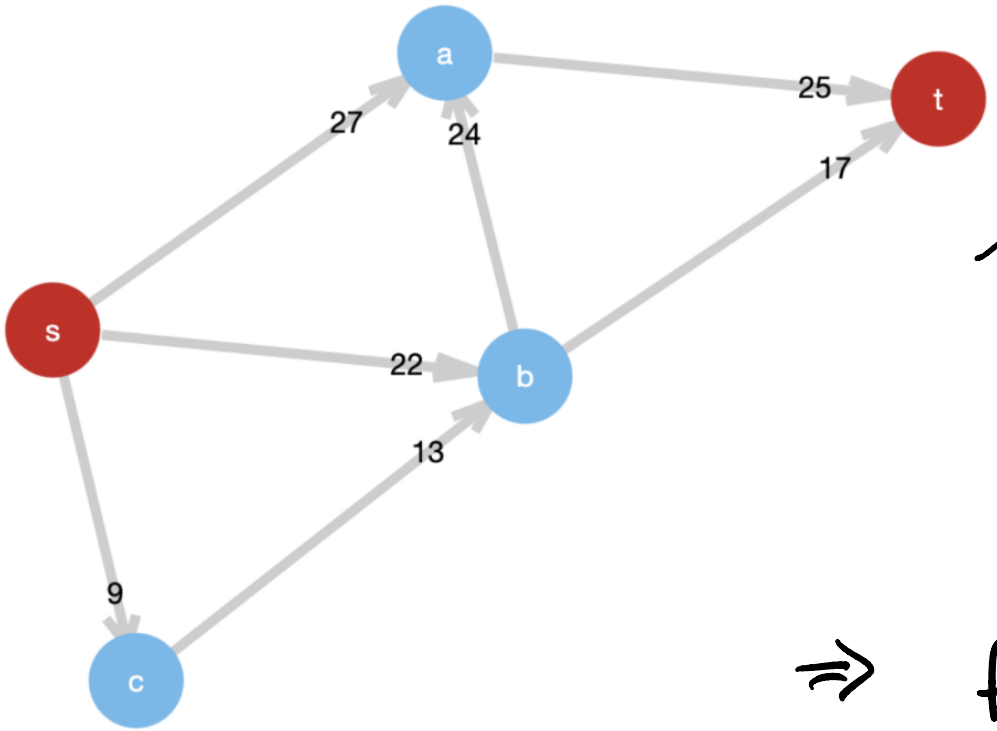
Ford_Fulkerson

FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f

Initialize $f(e) = 0 \quad \forall e \in G$
While there is an s-t path in G_f
Let P be a simple s-t path in G_f
$f' = augment(P, f)$
Set $f = f'$
Set $G_f = G_f$, from augment
Return f

G



Return flow

The max flow = sum of edges out of t in G_f ← "residual network"

$\Rightarrow \quad f = 25 + 17 = 42$

Ford–Fulkerson

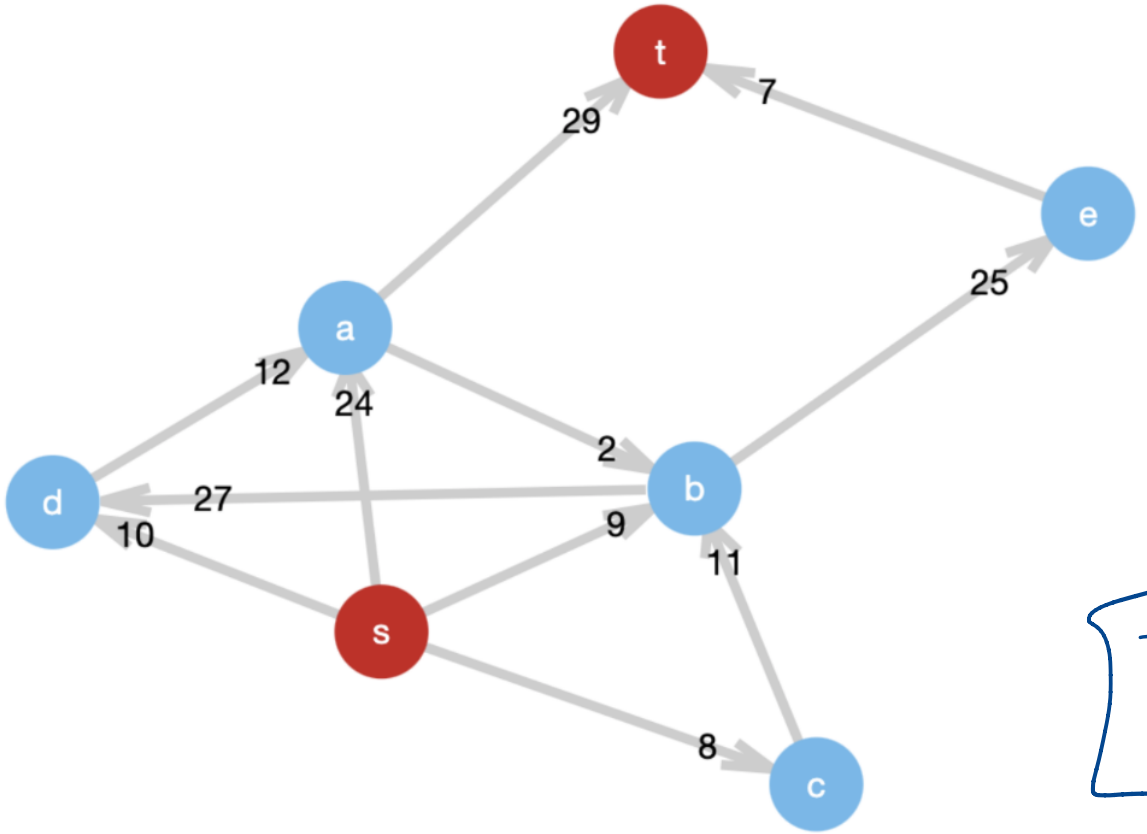
Sidenote: Runtime of F-F

- Note: every time through while loop, we increase flow.

→ worst case is that we increase flow 1 unit at a time ⇒ while loop executes = magnitude of max flow $= |f^*|$

FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f



→ with every execution of while loop, we find a path. if there are $|E|$ edges in graph, then we may need to search every edge, every time (worst case)

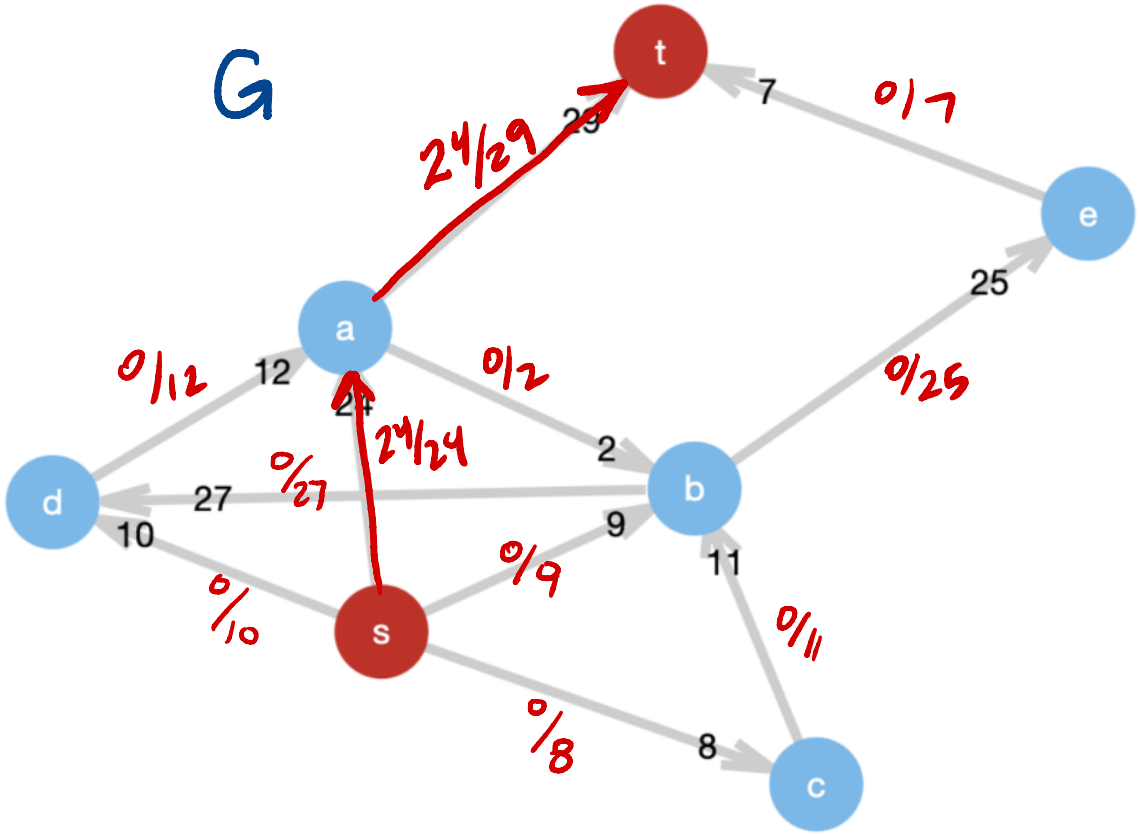
$$\Rightarrow O(|f^*||E|)$$

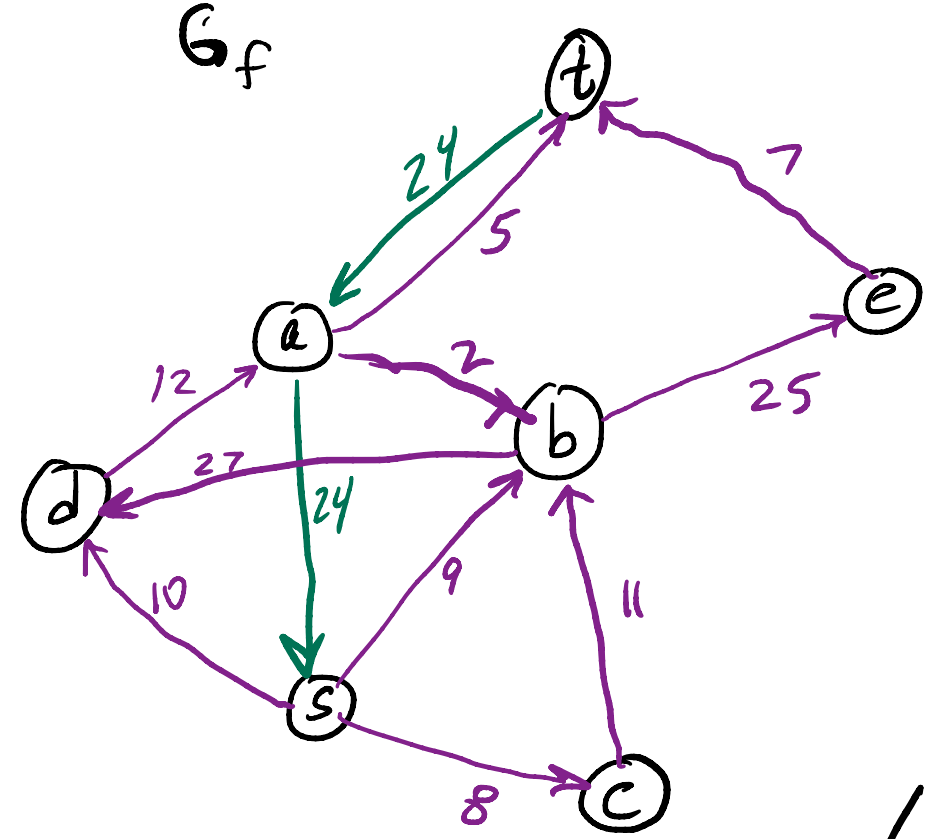
Ford–Fulkerson

FORD-FULKERSON-METHOD(G, s, t) First iteration

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f

initial path: $s \to a \to t$





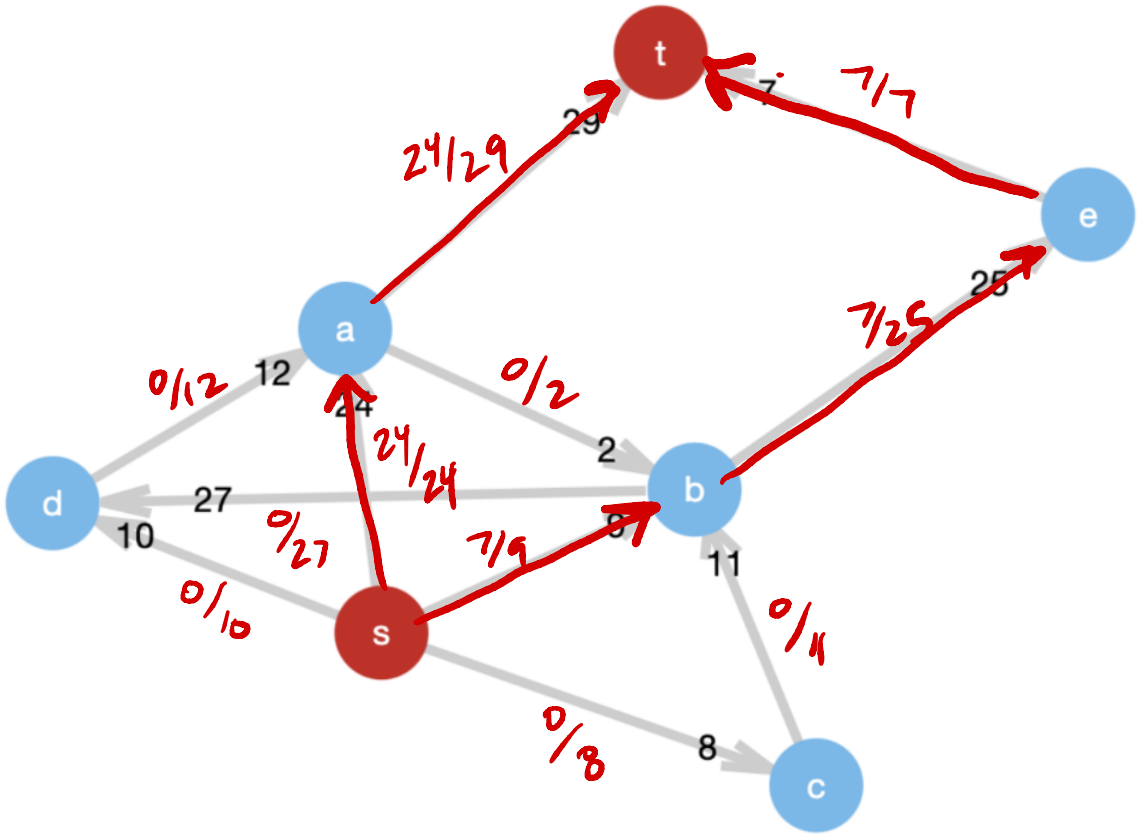
Next path $s \to b \to e \to t$ (7 units of flow)

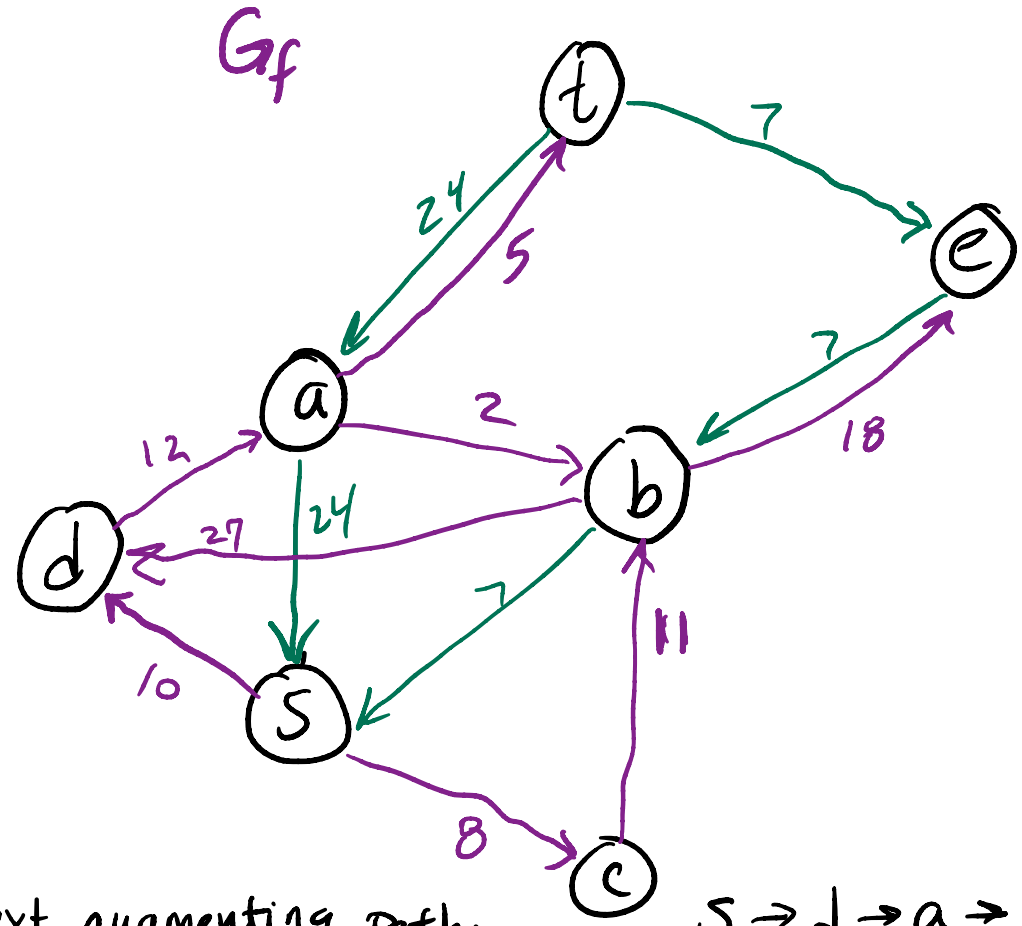
Ford–Fulkerson

2nd iteration

FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f





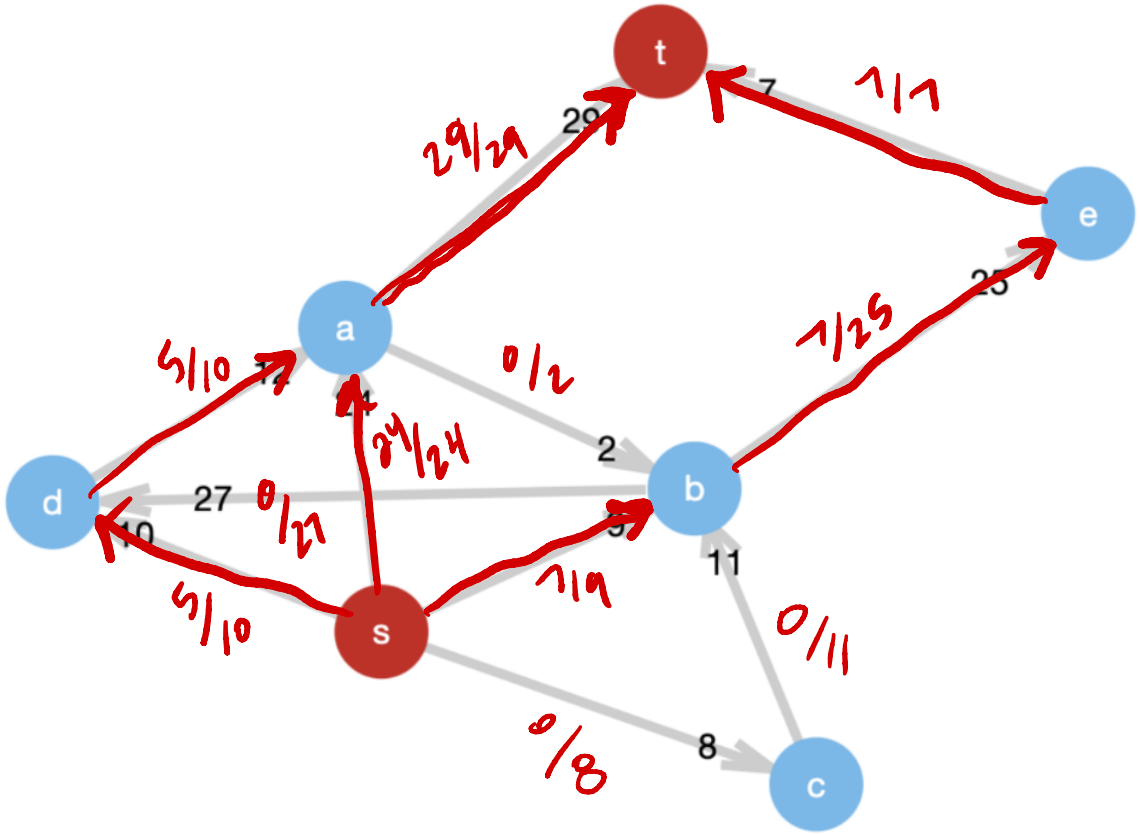
next augmenting path: $s \to d \to a \to t$

Ford–Fulkerson

3rd iteration

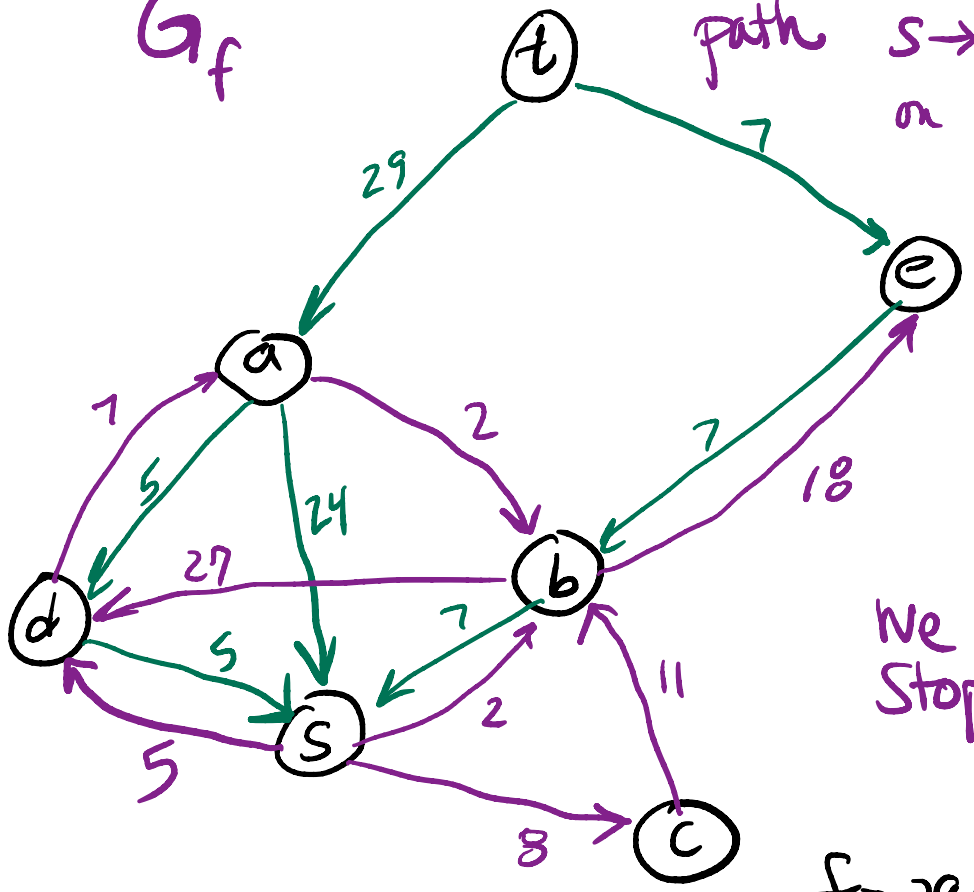
FORD-FULKERSON-METHOD(G, s, t)

1 initialize flow f to 0
2 **while** there exists an augmenting path p in the residual network G_f
3 augment flow f along p
4 **return** f



At this point, we see there is no longer a simple path $s \to t$ on G_f.

G_f



We Stop.

$f = 29 + 7 = 36$

Max–Flow Min–Cut

- How do we know that we have a max flow?
- What is the flow capacity at the source? ← gives an upper bound on the max flow.

Recall: $(s, t) - cut$

- Divide the vertices into two sets A and B, such that $s \in A, t \in B$. All flow must cross between A and B somewhere.

Max–Flow Min–Cut

Max-Flow Mini-Cut Theorem - In every flow network, the max $s - t$ flow is equal to the minimum capacity of an $s - t$ cut.

- applications to network connectivity/availability
- Road networks
- ...

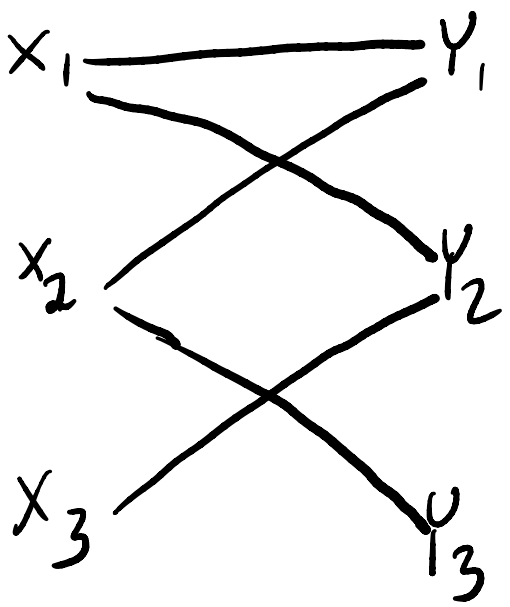
etc.

Bipartite Matching

$V = (x_1, x_2, x_3, y_1, y_2, y_3)$

A graph $G = (V, E)$ is ***bipartite*** if its vertex set V is partitioned into sets X and Y such that every edge has one end in X and one end in Y.

example



- This is bipartite because
  - → no x_i connects to another x_j
  - → no y_i connects to y_j

Bipartite Matching

Source Paper

- left set of "nodes" are different bee species
- Right set of nodes represent different types of plants.

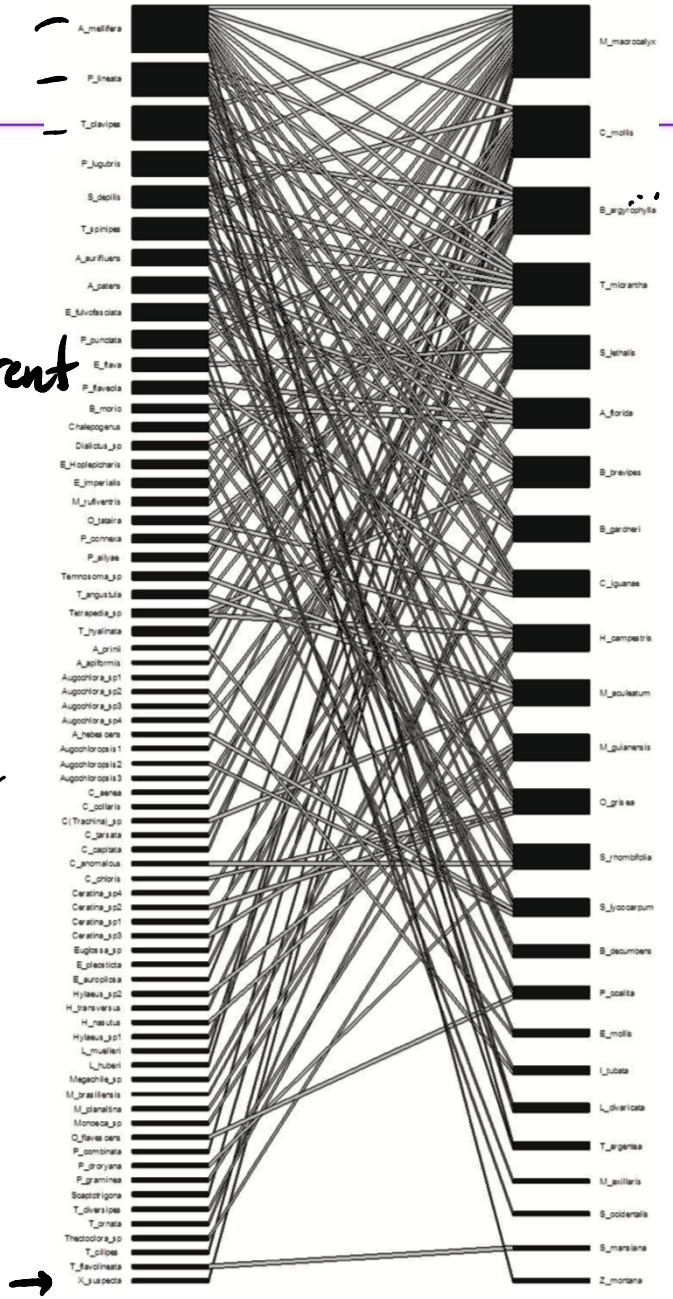


Fig 2. Bipartite graph of bee-plant interaction's network sampled in the Semideciduous Forest of "Fazenda São José", Uberlândia-MG, in 2010 and 2011. Bees are represented on the left side of the graph and plants on the right side.

Table 2. Values of the Betwenness Centrality Index for species of the network of bee-plant interaction sampled in the Semidecidous Forest of "Fazenda São José", Uberlândia-MG, in 2010 and 2011.

Bee species	Index	Plant	Index
Apis mellifera	0.305	*Merremia macrocalyx*	0.319
Paratrigona lineata	0.108	*Coccoloba mollis*	0.152
Tetragona clavipes	0.105	*Banisteriopsis argyrophylla*	0.174
Paratetrapedia cf. *lugubris*	0.097	*Trema micrantha*	0.107
Scaptotrigona aff. *depilis*	0.040	*Serjania lethalis*	0.100
Trigona spinipes	0.053	*Arrabidaea florida*	0.084
Augochloropsis cf. *aurifluens*	0.016	*Bauhinia brevipes*	0.101
Augochloropsis cf. *patens*	0.049	*Bidens gardneri*	0.058
Exomalopsis fulvofasciata	0.039	*Celtis iguanae*	0.035
Paratetrapedia punctata	0.079	*Heteropterys* cf. *campestris*	0.068
Epicharis flava	0.028	*Machaerium aculeatum*	0.056
Paratetrapedia cf. *flaveola*	0.010	*Matayba guianensis*	0.083
Bombus morio	0.010	*Oxalis grisea*	0.052
Chalepogenus sp.	0.008	*Sida rhombifolia*	0.052
Dialictus sp.	0.002	*Solanum lycocarpum*	0.045
E. (*Hoplepicharis*) affinis	0.012	*Brachiaria decumbens*	0.009
Euglossa imperialis	0.041	*Prestonia coalita*	0.022
Melipona rufiventris	0.013	*Elephantopus* cf. *mollis*	0.021
Oxytrigona cf. *tataira*	0.0003	*Ipomoea tubata*	0.021
Paratetrapedia connexa	0.015	*Luehea divaricata*	0.003
Partamona ailyae	0.013	*Terminalia argentea*	0.001
Temnosoma sp.	0.009	*Memora axillaris*	0.000
Tetragonisca angustula	0.006	*Senna ocidentalis*	0.000
Tetrapedia sp.	0.018	*Serjania mansiana*	0.000
Trigona hyalinata	0.011	*Zeyheria montana*	0.000

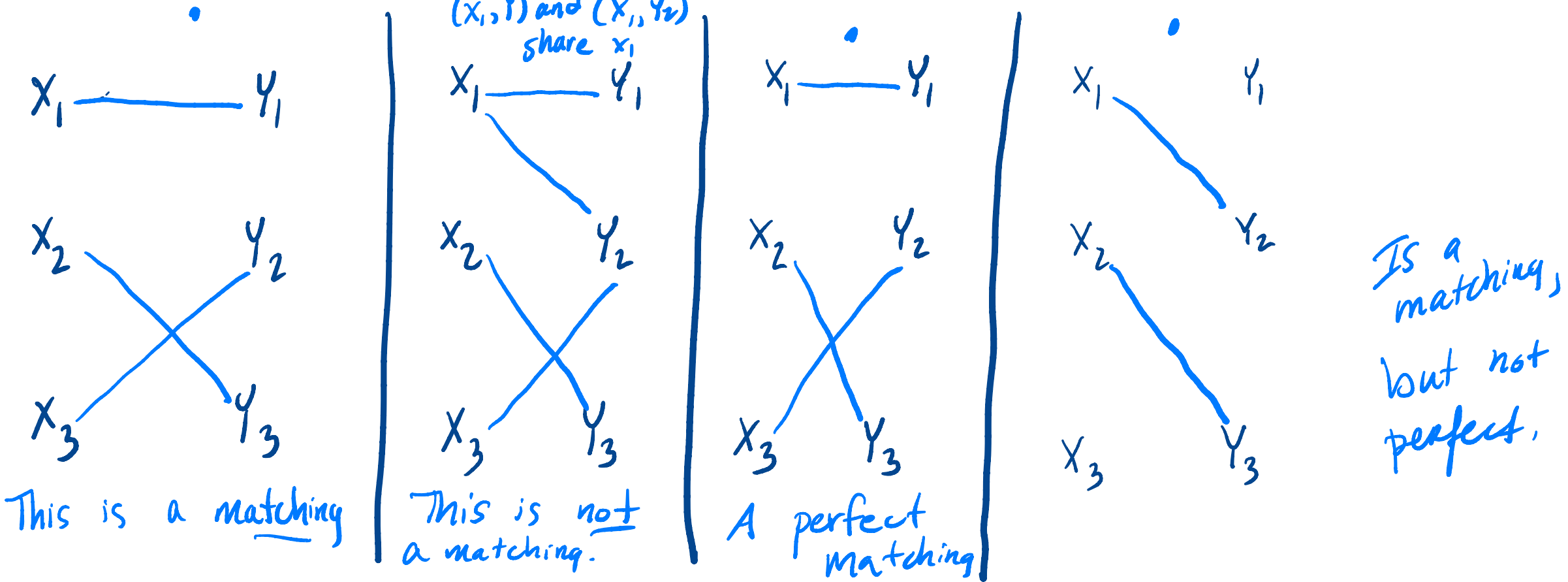
Note: Bee species that were not presented in the table obtained index equal to zero.

Bipartite Matching

A ***matching*** in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ where every vertex appears in at most one edge of M.

In other words, a matching is a subset of edges where no two edges share an endpoint.

M is a ***perfect matching*** if every V appears in exactly one edge of M.

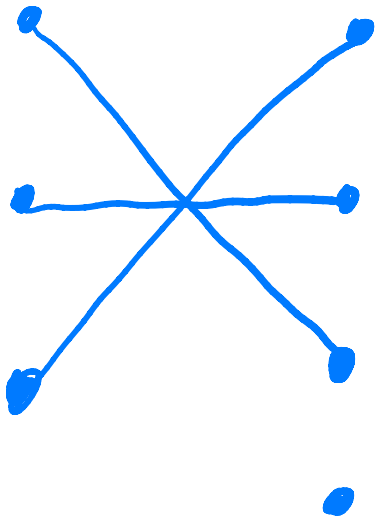


Bipartite Matching

Maximum Size Matching

If $|X|$ and $|Y| = n$, then a perfect matching of $|M|$ is n.

- Perfect matching may not exist, how do we maximize X and Y included in matching?



Bipartite Matching

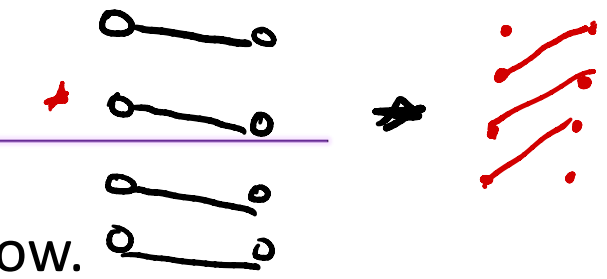
Idea: Convert Bipartite Matching to Max Flow

Given an undirected bipartite graph, what do we do to change it to a max flow graph?

1. Create a Directed Graph. Add direction to edges.
2. Add edge weights. Typically assign unit edge weight
3. Add source and sink. Add directed edges here to.

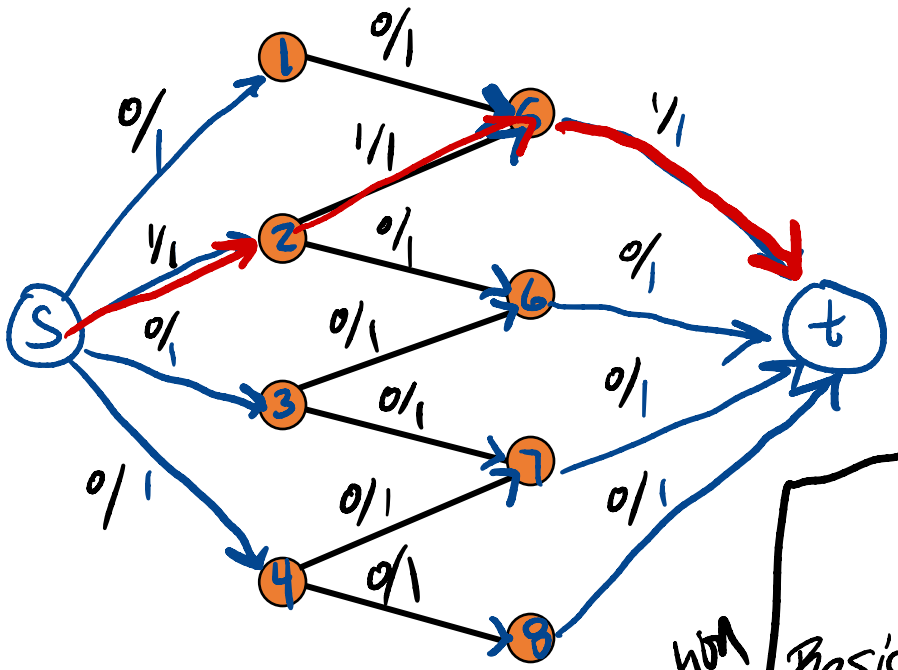
- Choose a simple path, push $f(e) = 1$ through the path.
- Run Ford-Fulkerson.
- The max flow will be a max bipartite matching!

Bipartite Matching

Note: 

Example: Find the maximum bipartite matching given the graph below.

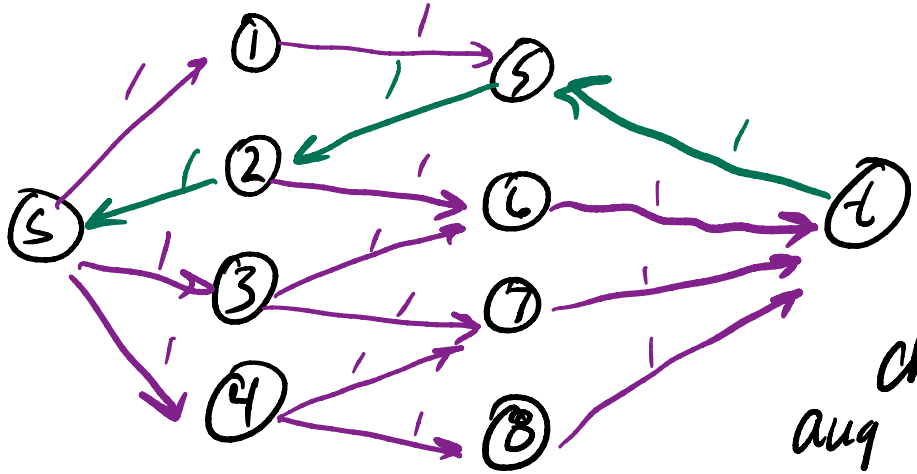
Step 1 Convert to a max flow
- Add directed edges
- Add (s), (t)
- add unit weights



Step 2
Start with a path.

$s \to 2 \to 5 \to t$

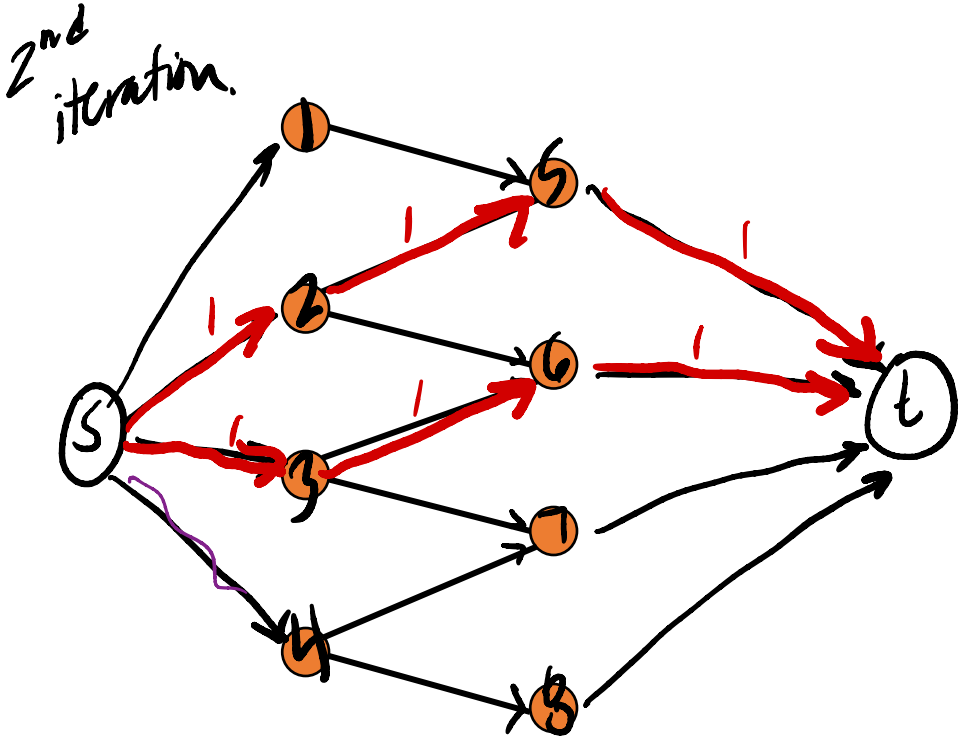
Iteration 1 → Residual graph:

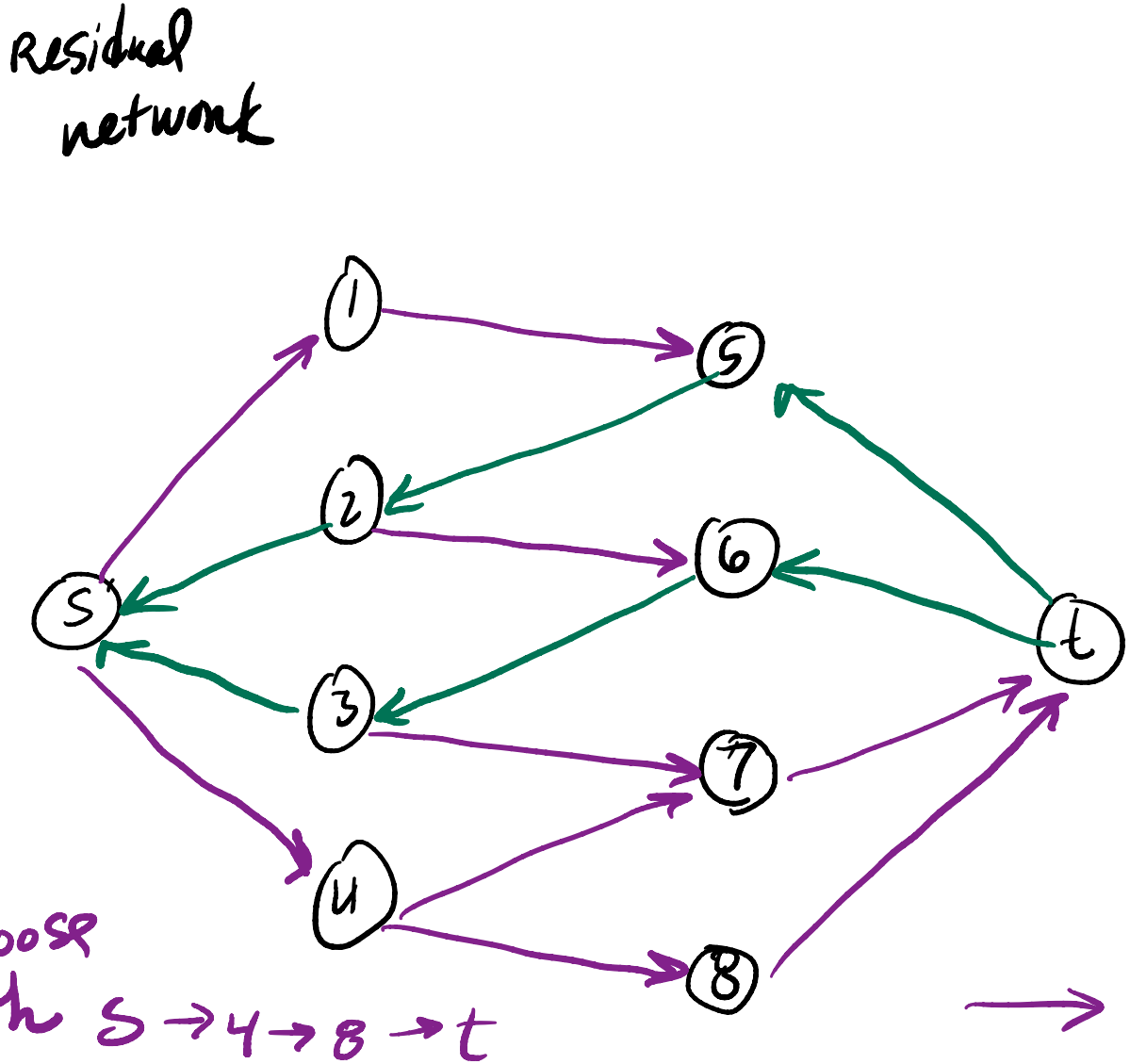


Choose new aug path:
$s \to 3 \to 6 \to t$

Bipartite Matching

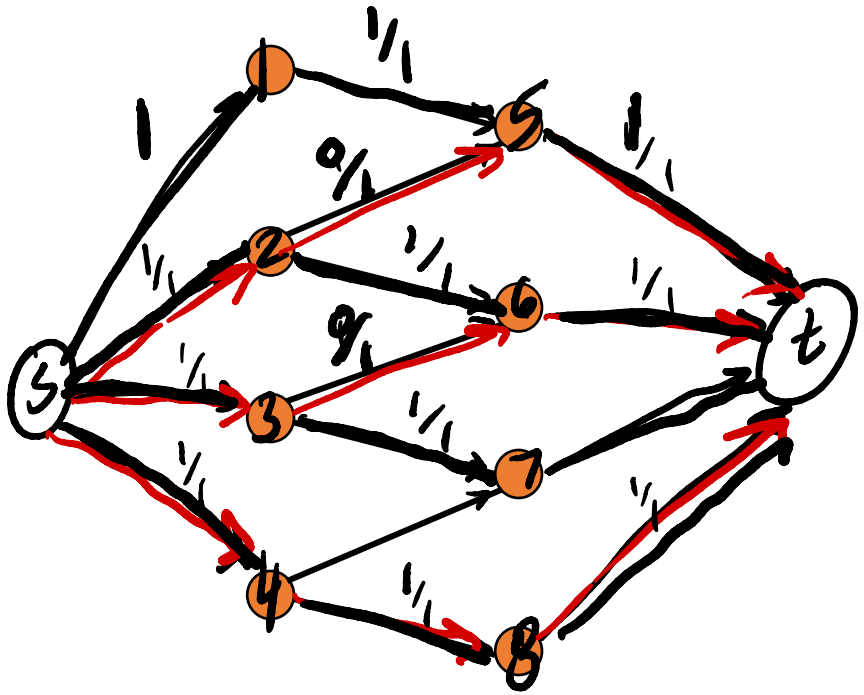
Example: Find the maximum bipartite matching given the graph below.





Bipartite Matching

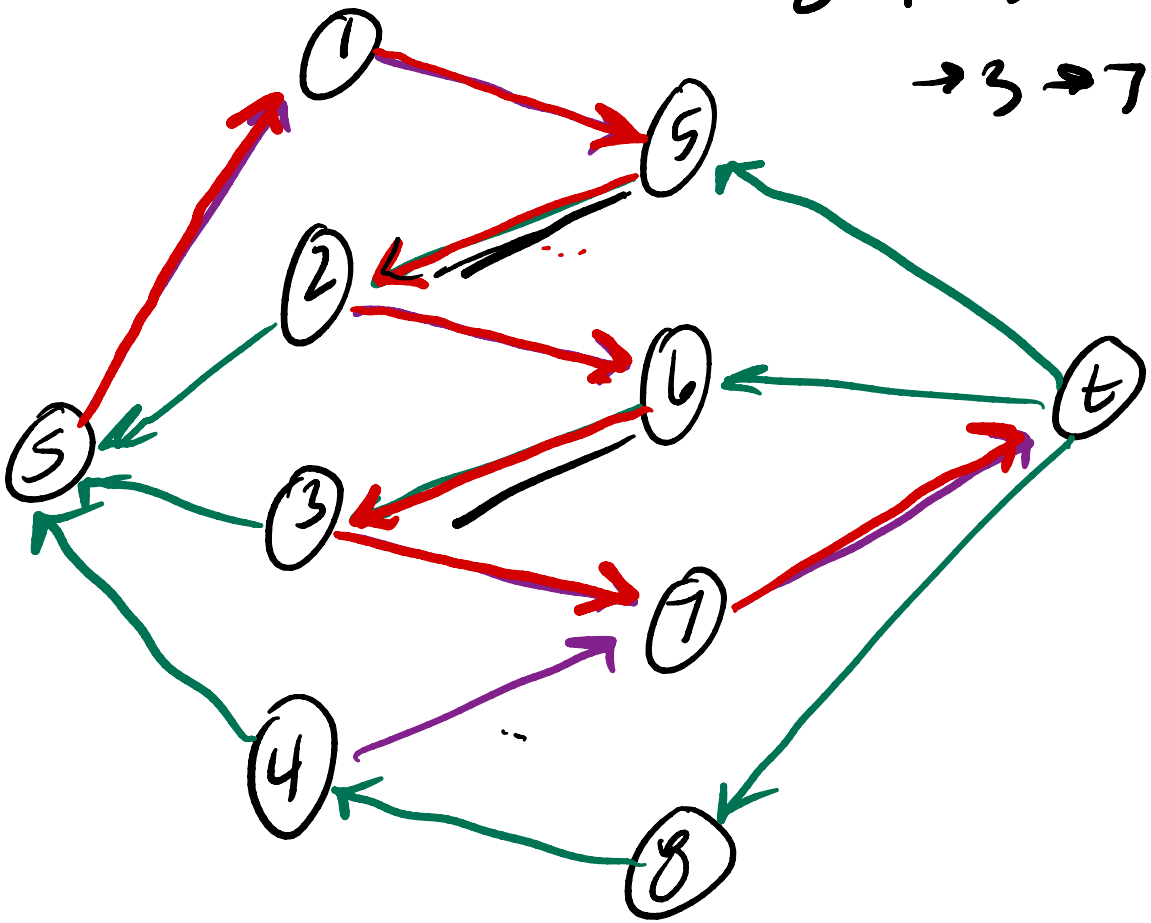
Example: Find the maximum bipartite matching given the graph below.



G_f

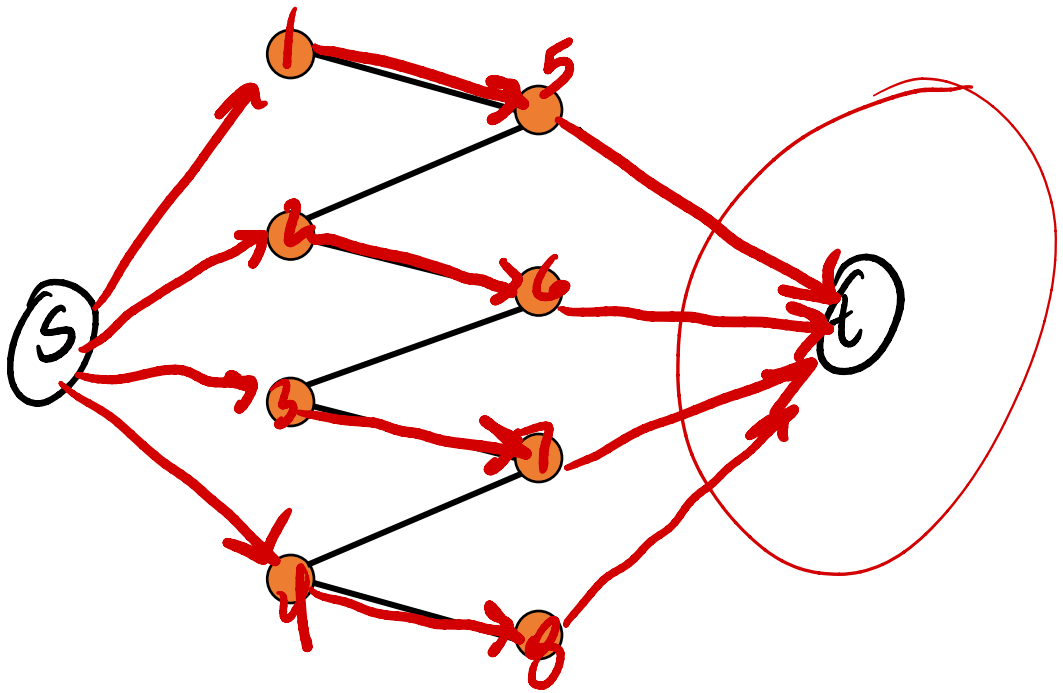
Next path?

s→1→5→2→6
→3→7→t



Bipartite Matching

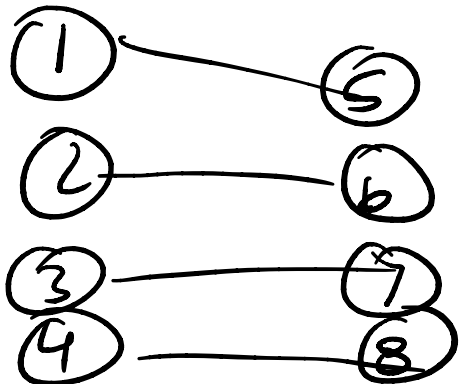
Example: Find the maximum bipartite matching given the graph below.



The max flow = 4

This equals to the maximum number of matchings!

corresponds to



Intro to Dynamic Programming

Question: What is Dynamic Programming in a nutshell?

- Algorithmic Paradigm that breaks a problem into subproblems and store the results of subproblems to avoid re-computation.

Properties of Problems that can be Solved with Dynamic Programming

- Overlapping subproblems
- Optimal substructure

Intro to Dynamic Programming

Example: The Fibonacci sequence

Intro to Dynamic Programming

When should you NOT use Dynamic Programming?

- Not useful for problems that don’t have overlapping subproblems with repeat computations.