Analysis of Algorithms

V. Adamchik

Lecture 8

CSCI 570

University of Southern California

Network Flow (2 lectures)

Reading: chapter 7.1 – 7.4

Violation of Academic Integrity

Honor Code Pledge: "I affirm that I have not used any unauthorized materials in completing this exam and have neither given assistance to others nor received assistance from others."

There are significant consequences for violating academic integrity.

If you feel that you violate the pledge you signed, I want you to step forward and contact me directly by the end of this week.

The Network Flow Problem

Solve by reduction

Our fourth major algorithm design technique
(greedy, divide-and-conquer, and dynamic programming).

input → input(NF)

solve(NF)

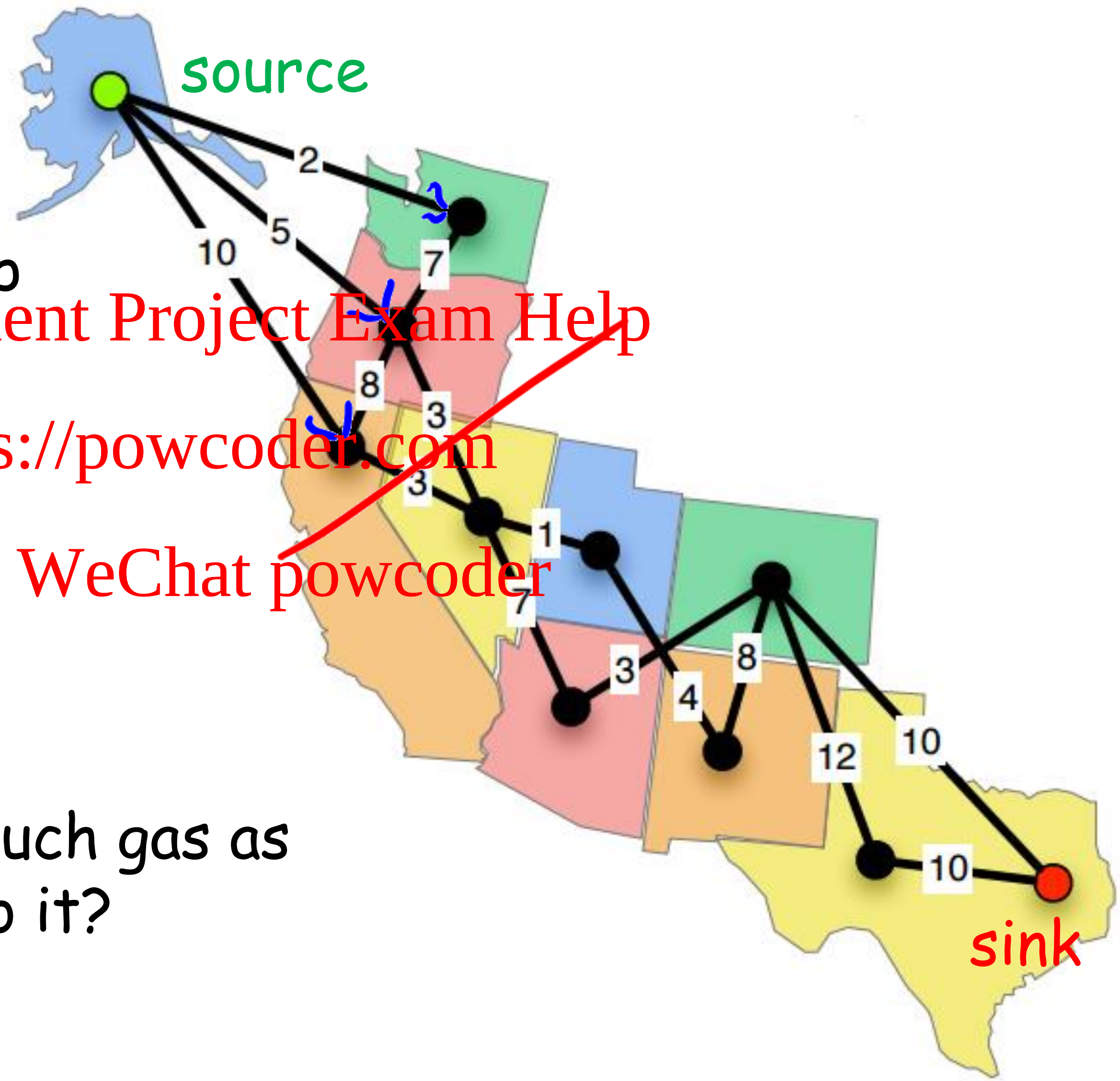
output ← output(NF)

Plan:

The Ford-Fulkerson algorithm

Max-Flow Min-Cut Theorem

The Flow Problem

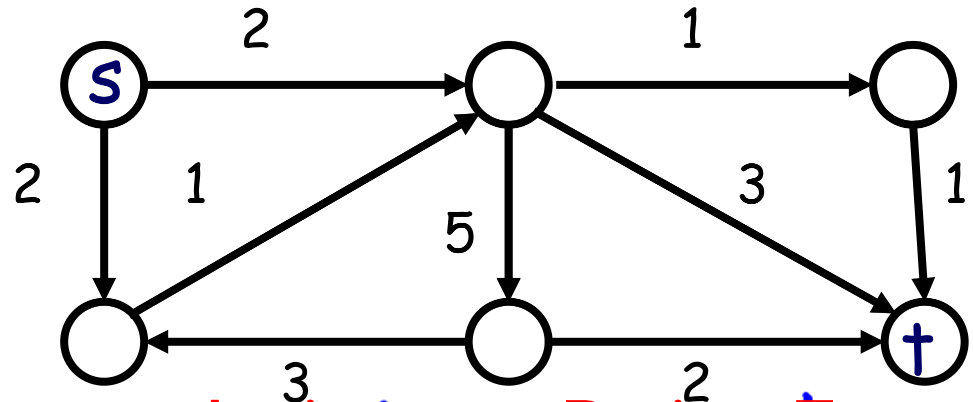


Suppose you want to ship natural gas from Alaska to Texas.

Pipes have capacities.

The goal is to send as much gas as possible. How can you do it?

The Max-Flow Problem



$c > 0$

$NF = (V, E, s, t, c)$

$c(e) > 0, \; e \in E$

$c(e) = 0, \; e \notin E$

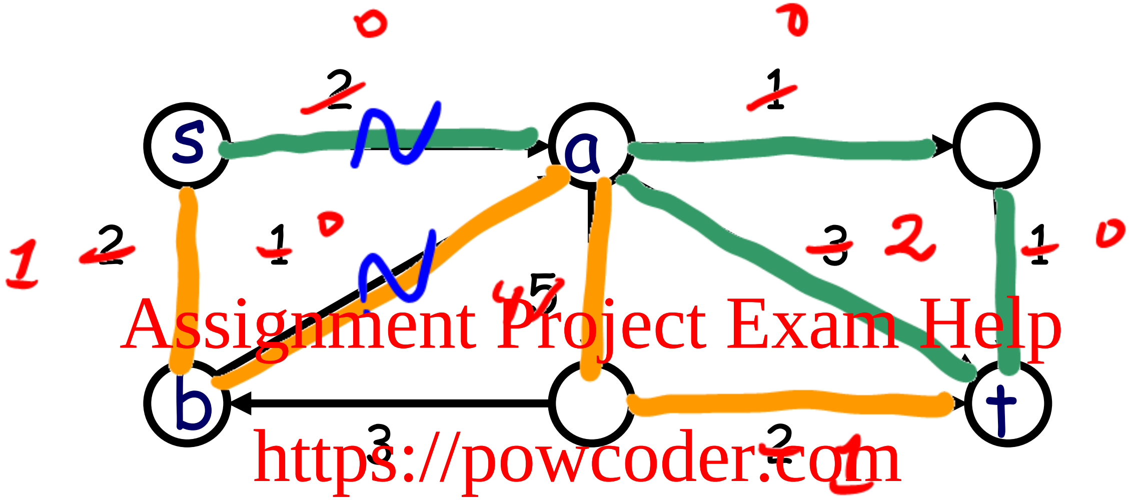
<u>Def.</u> A flow $f(e): E \to \mathbb{R}$

1. $0 \le f(e) \le c(e)$, capacity law

2. $\sum_{\text{into } v} f(e) = \sum_{\text{out } v} f(e)$, conservation law

$v \neq \{s, t\}$

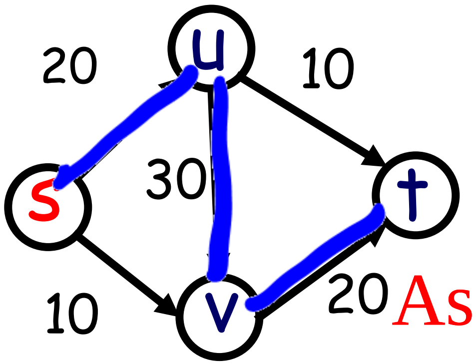
The MAX Flow Problem

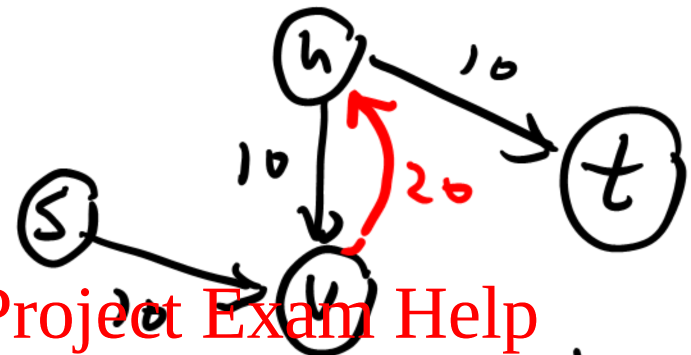


The max-flow here is 3.

How can you see that the flow is really max?

Greedy Approach: push the max





Push 20 via s-u-v-t

is it max?

1) Push 10 s-u-t

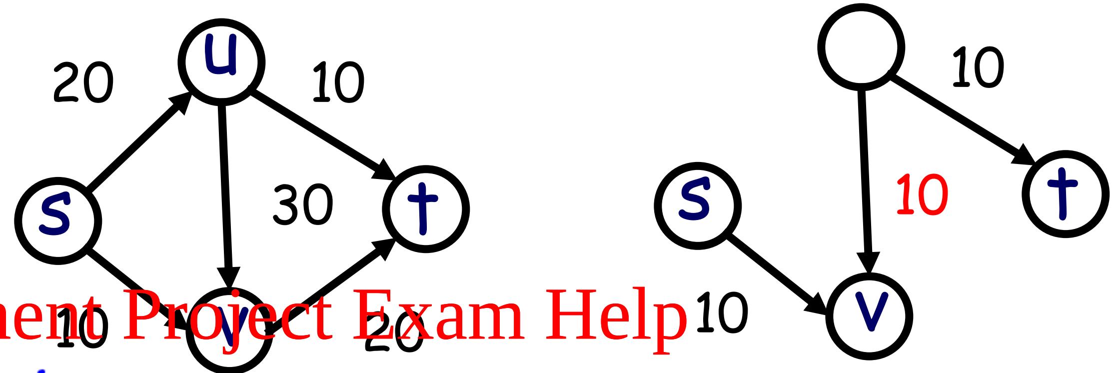
2) Push 10 via s-v-t

3) Push 10 via s-u-v-t

max-flow = 30

Canceling Flow

Push 20 via s-u-v-t



To increase flow by

1) find an s-t path with unused capacity

2) find a cancelable flow

modify our graph G by adding new edges

Residual Graph $\boxed{G_f}$

$G = (V, E)$

$G_f = (V, E_f)$

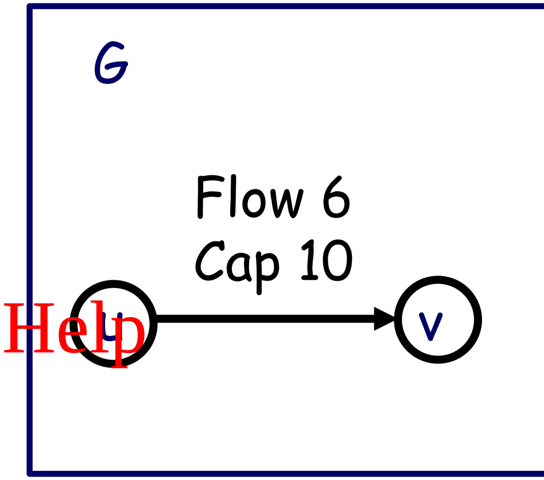
E_f consists of

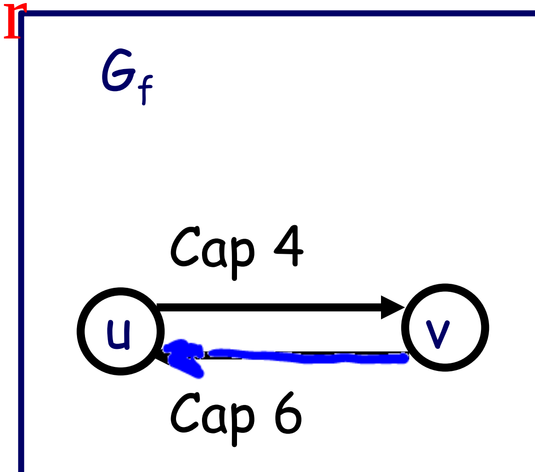
1) forward edges, $e \in E$

   $c_f = c(e) - f(e) > 0$

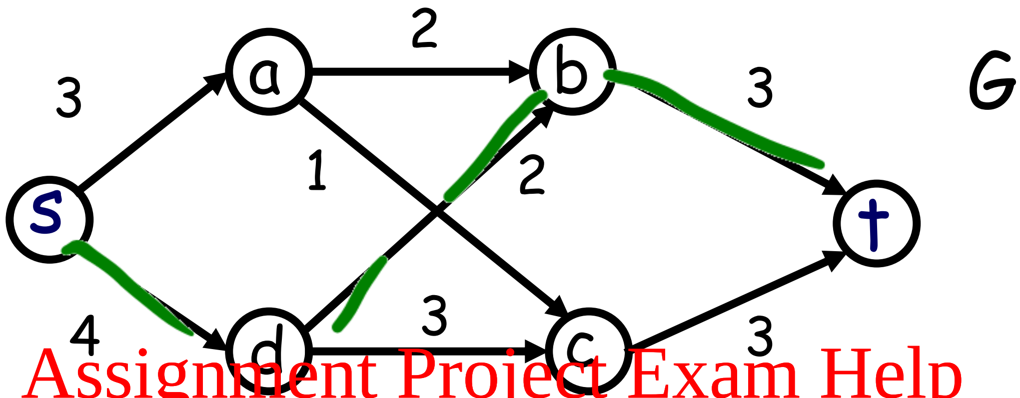
2) backward edges, $e \notin E$

   $c_f(e) = f(e) \geq 0$

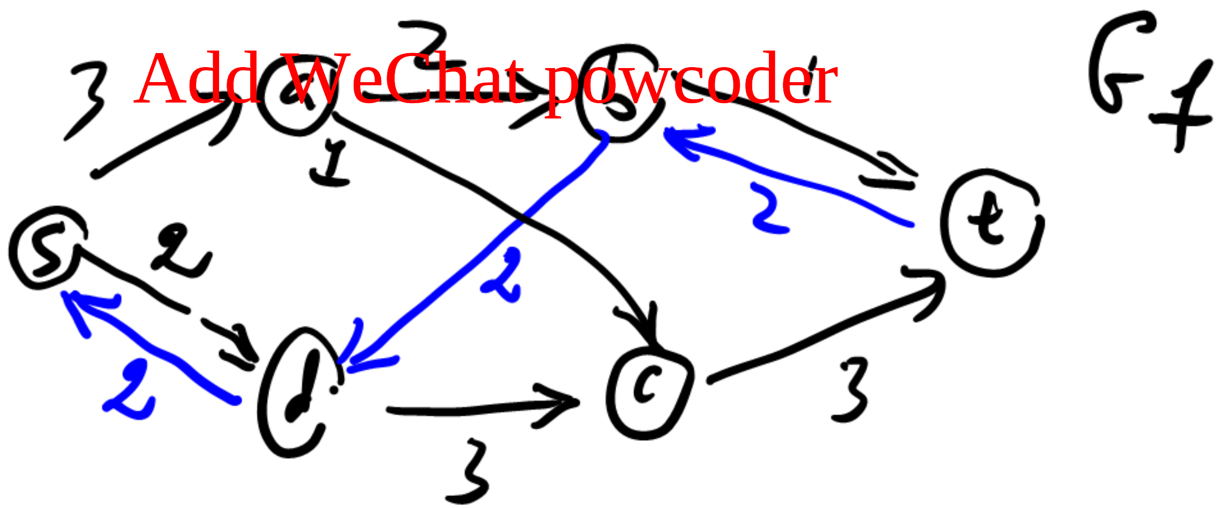




Example: residual graph



Push 2 along s-d-b-t and draw the residual graph



Augmenting Path = Path in G_f

traversal

Let P be an $\boxed{\text{s–t path}}$ in the residual graph G_f.

Let bottleneck(P) be the smallest capacity in G_f on any edge of P.

If bottleneck(P) > 0 then we can increase the flow by sending bottleneck(P) along the path P.

augment(f, P):
b = bottleneck(P)
for each e = (u,v) ∈ P:
    if e is a forward edge:
        decrease $c_f(e)$ by b //add some flow
    else:
        increase capacity by b //erase some flow

The Ford-Fulkerson Algorithm

FF

Algorithm. Given (G, s, t, c)

start with $f(u,v)=0$ and $G_f = G$.

while exists an augmenting path in G_f

find bottleneck

augment the flow along this path

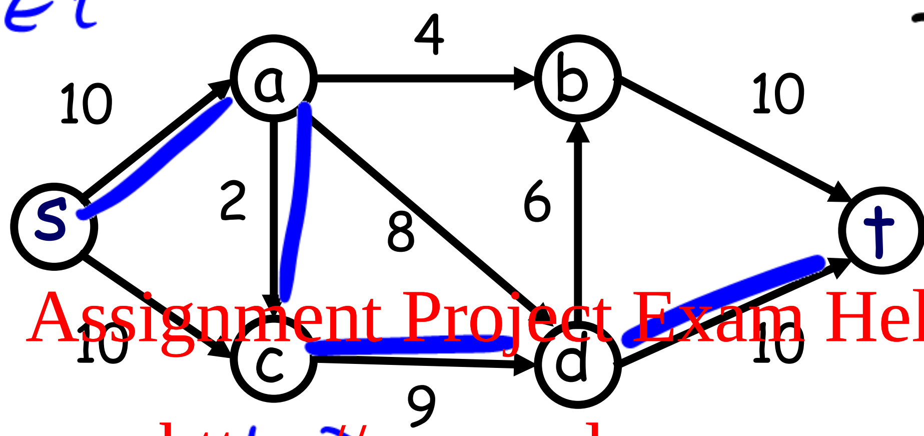
update the residual graph G_f

Example

$G_f = G$

$f(e) = 0,\ e \in E$

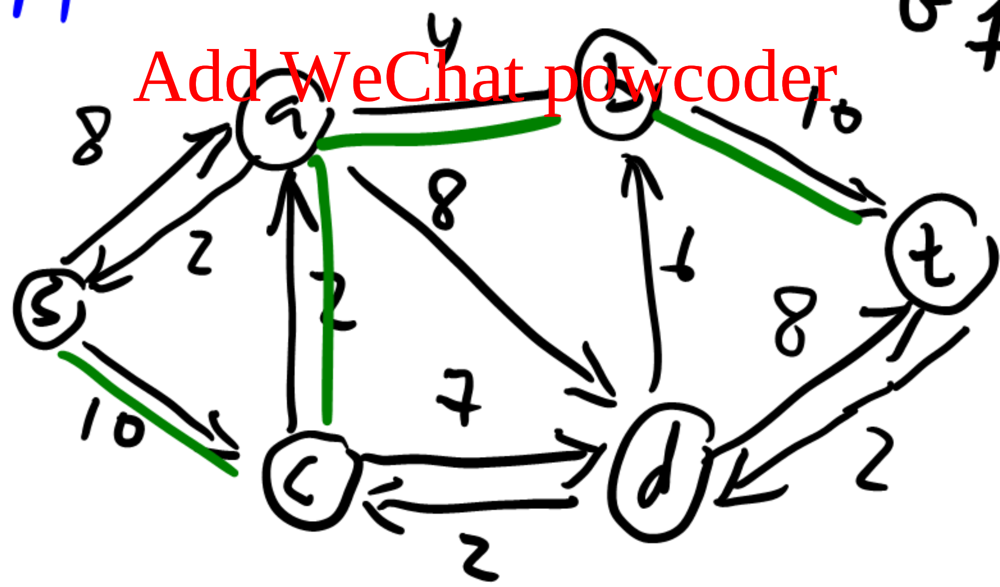
$G = G_f$



$\boxed{\text{traversal}}$

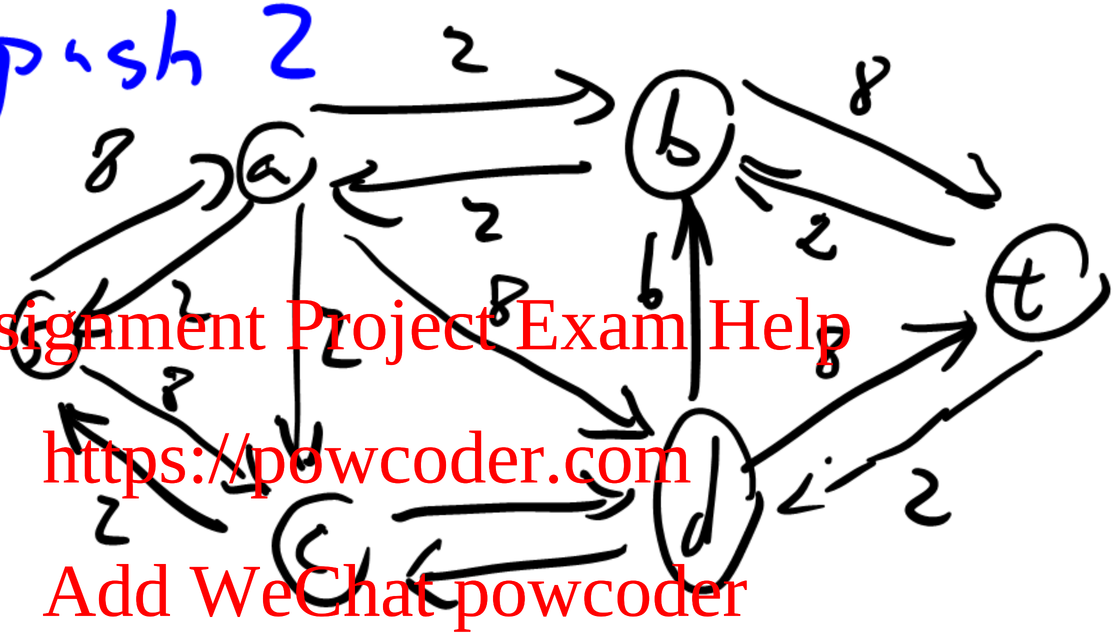
Path s-a-c-d-t , push 2

G_f



Example

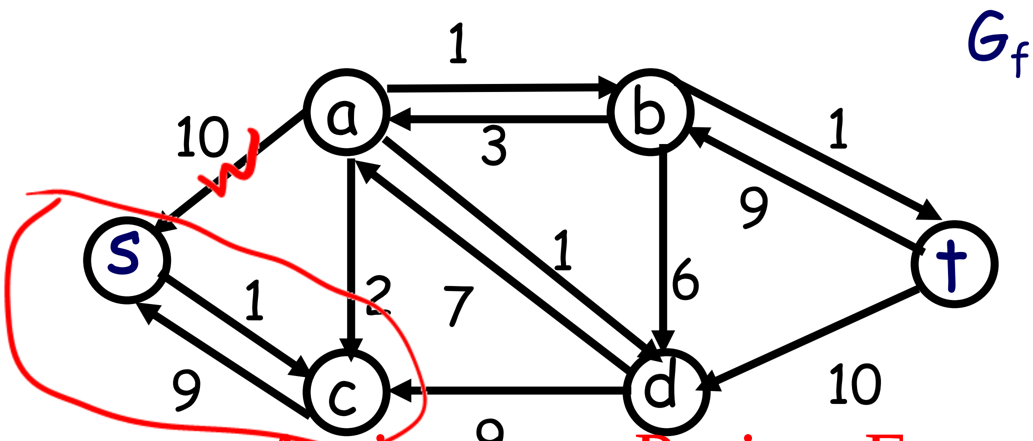
Path s-c-a-b-t, push 2



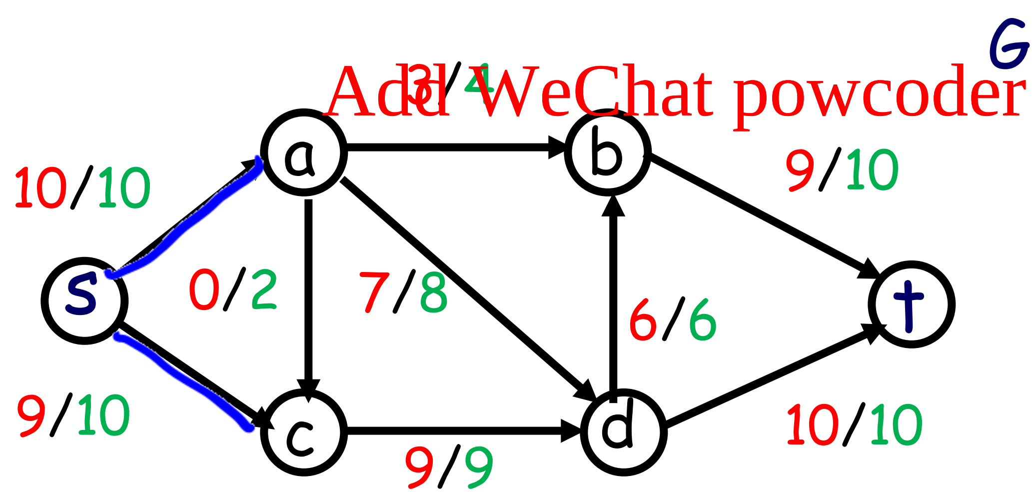
~~Path s-c-d-b-~~t

Example

Path s-c-d-a-b-t. Do it yourself.



In graph G edges are with flow/cap notation



max-flow = 19

The Ford-Fulkerson Algorithm

Runtime Complexity

integers

<u>Algorithm</u>. Given $(G, s, t, \boxed{c \in \mathbb{N}^+})$

start with $f(u,v)=0$ and $G_f = G$. $\rightarrow O(E+V)$

while exists an augmenting path in G_f

steps

find bottleneck $\rightarrow O(V)$

augment the flow along this path

update the residual graph

each step is linear

$\#$ of steps $\leq |f| = \sum f(e)$

number

e out of s

$O(|f| \cdot (V+E))$

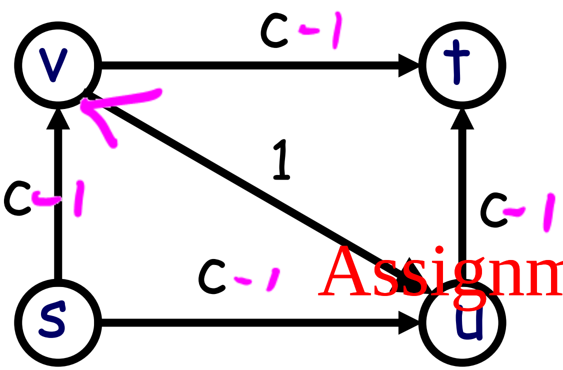
is it polynomial?

NO

The worst-case

$\boxed{O(|f|\,(E+V))}$

$c = 10^9$



$O(|f| \cdot E)$

$E = O(V^2)$

1. s-v-u-t, f=1
2. s-u-v-t, f=2

continue...

iterations = 2·C

$c \in \mathbb{N}^+$

Proof of Correctness

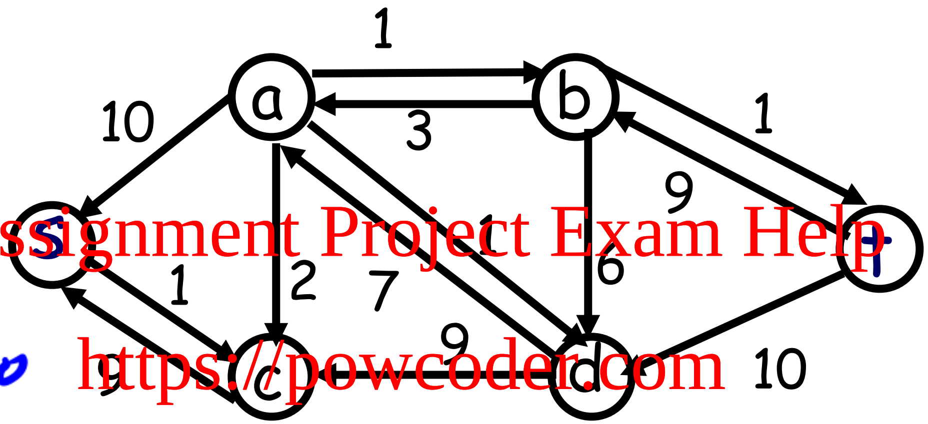
How do we know the algorithm terminate

How do we know the flow is $\boxed{\text{max}}$imum?

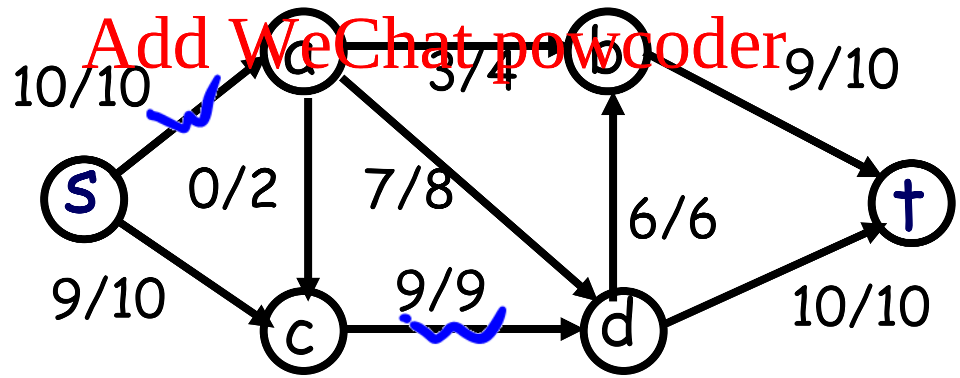
$c_f(e) = c(e) - f(e)$ G_f

$2 - 2 = 0$

$2.0886/ - 1.555 \neq 0$



G

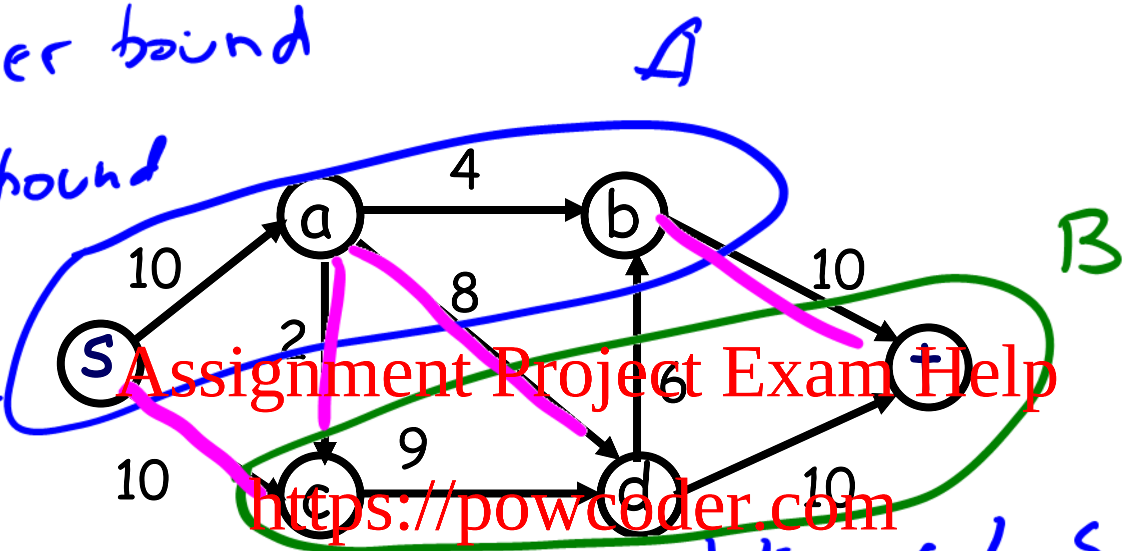


Cuts and Cut Capacity $cap(A, B)$

$f \le$ upper bound

if $f =$ upper bound then

f is max



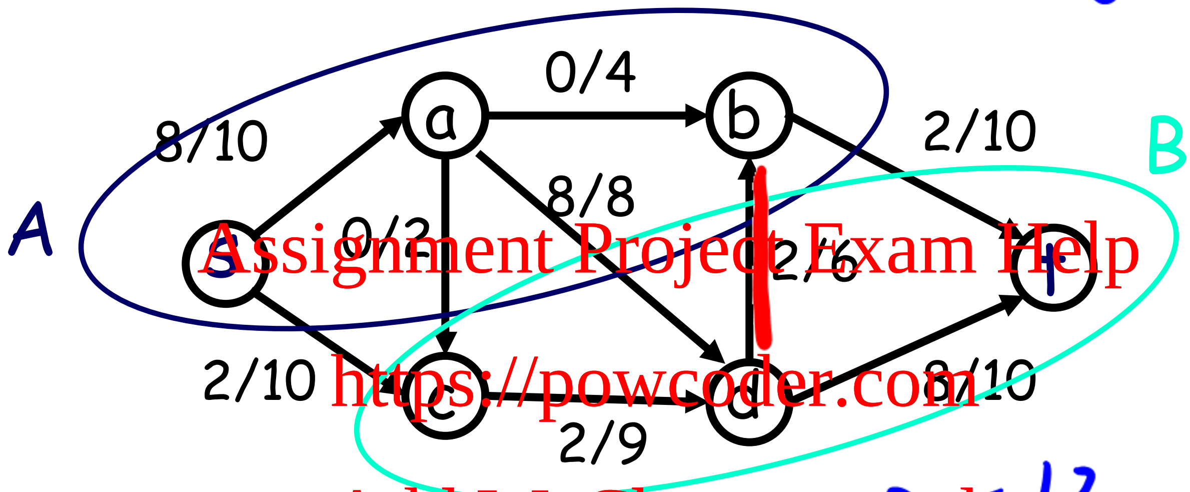
Def. A cut is a vertex partition, s.t. $s \in A$, $t \in B$

Def. $cap(A,B) = \sum_{\text{out of } A} c(e) = 10 + 2 + 8 + 10 = 30$

Cuts and Flows

Consider a graph with some flow and cut

G



The flow-out of A is ___ 2 + 0 + 8 + 2 = 12

The flow-in to A is ___ 2

The flow across (A,B) is ___ 12 - 2 = 10

What is a flow value |f| in this graph? 8 + 2 = 10

Lemma 1

$s \in A,\ t \in B$

For any flow f and any (A,B) cut

$$|f| = \sum_{v} f(s,v) = \sum_{u \in A, v \in B} f(u,v) - \sum_{u \in A, v \in B} f(v,u)$$

Proof.

$$|f| = \sum_{\text{out } s} f(e) \stackrel{?}{=} \sum_{\text{out } s} f(e) - \sum_{\text{to } s} f(e) = $$

by conservation law

any A

$$= \sum_{v \in A} \left[\sum_{\text{out } v} f(e) - \sum_{\text{to } v} f(e) \right] = 0, \text{ for all } v \in A,\ v \neq s$$

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

■

Lemma 2 !!!

For any flow f and any (A,B) cut $|f| \leq cap(A,B)$.

Proof.

$$|f| = \sum_{\text{out } A} f(e) - \underbrace{\sum_{\text{to } A} f(e)}_{\text{negative}} \leq \sum_{\text{out } A} f(e) \leq \sum_{\text{out } A} c(e) = cap(A,B)$$

capacity law

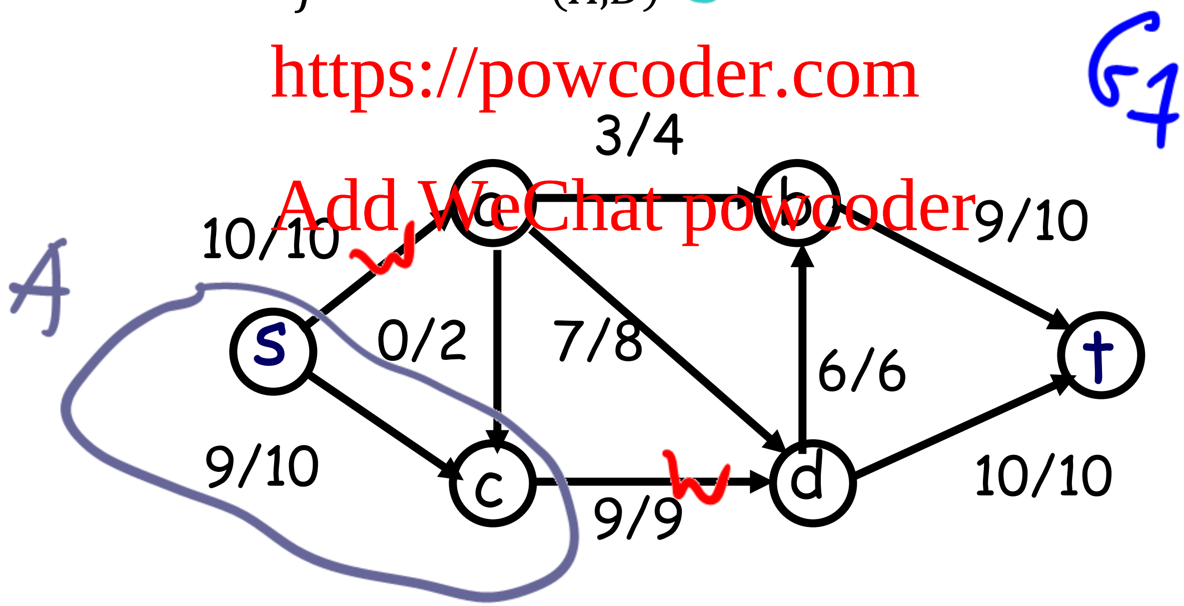
$= 0$

$f = c$

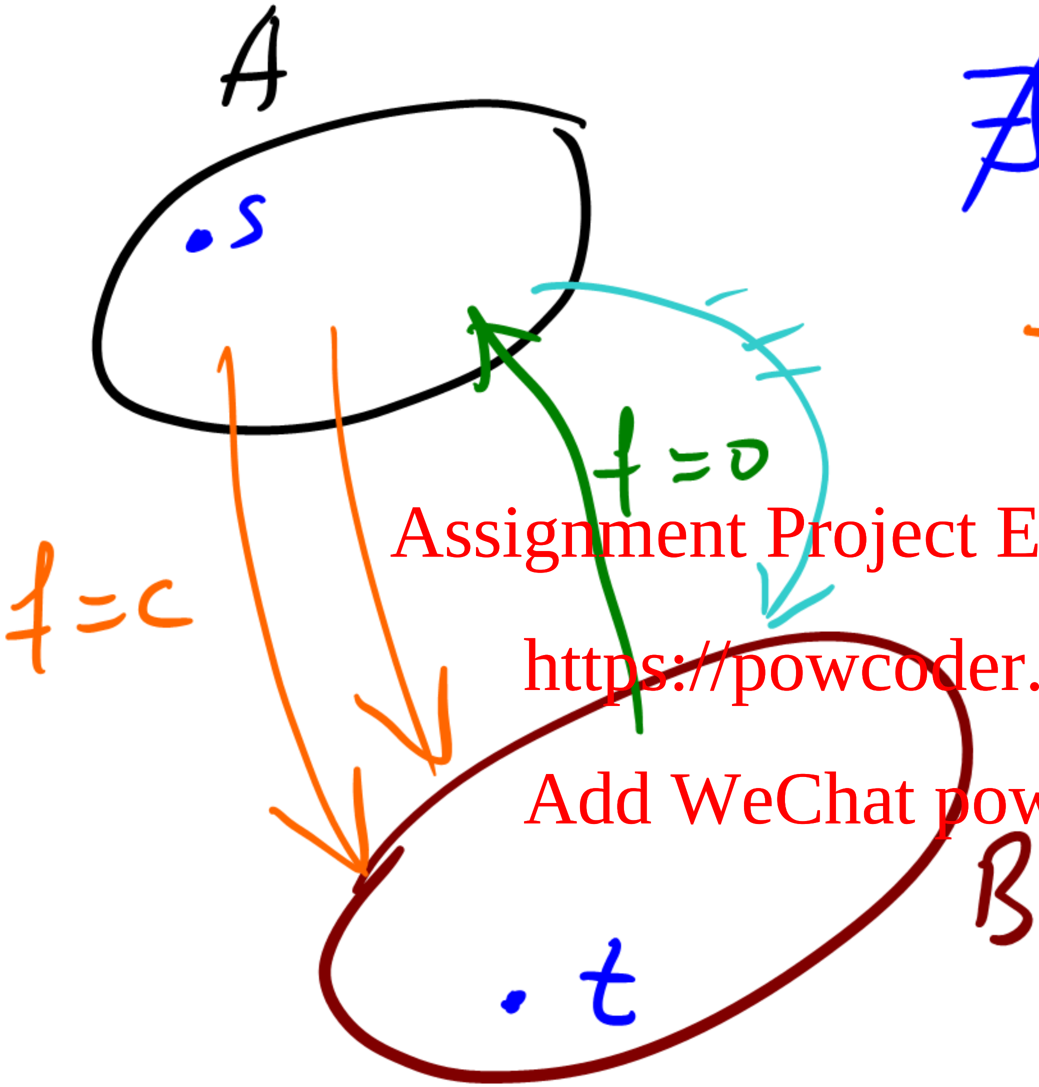
Max-flow Theorem

<u>Theorem</u>. *The Ford-Fulkerson algorithm outputs the maximum flow.*

$$\max_{f} |f| = \min_{(A,B)} cap(A,B)$$



Where is a min-cut ?



$\nexists$ s-t path

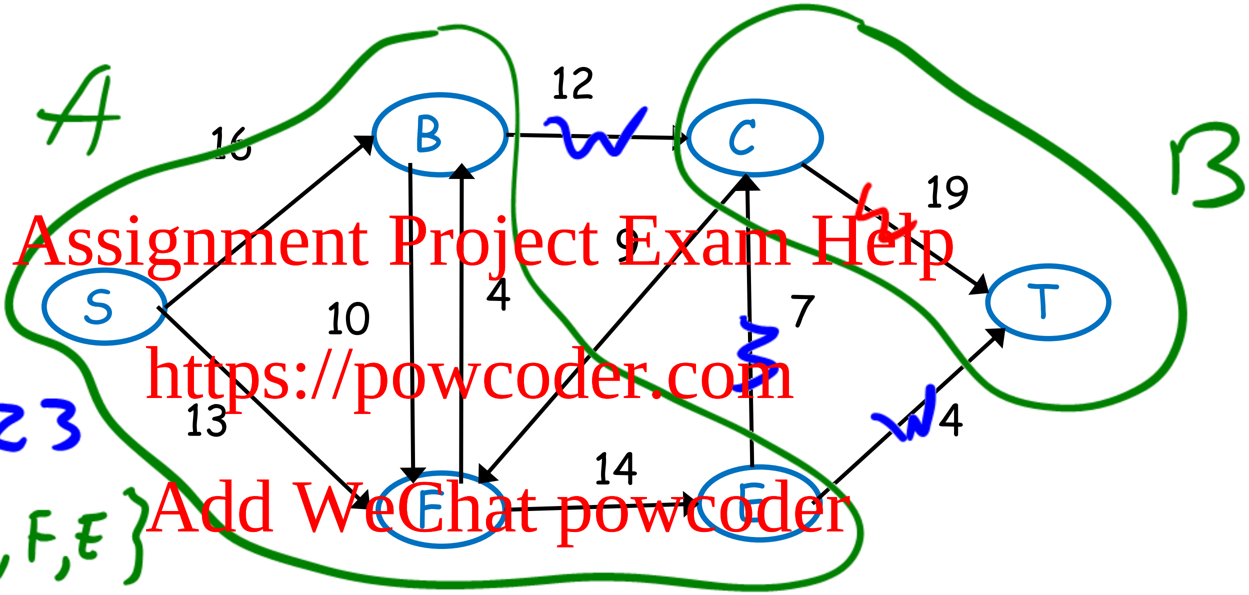
1. $f = c$
2. $f \neq 0$ it follows that it has an s-t path contradiction

p. 116

Discussion Problem 1

Run the Ford-Fulkerson algorithm on the following network:

SBCT : 12
SFET : 4
SFECT : 7
max-flow = 23



$A = \{S, B, F, E\}$
$B = \{T, C\}$

How do you find a min-cut ?

1. Run FF
2. Run BFS in G_f from S

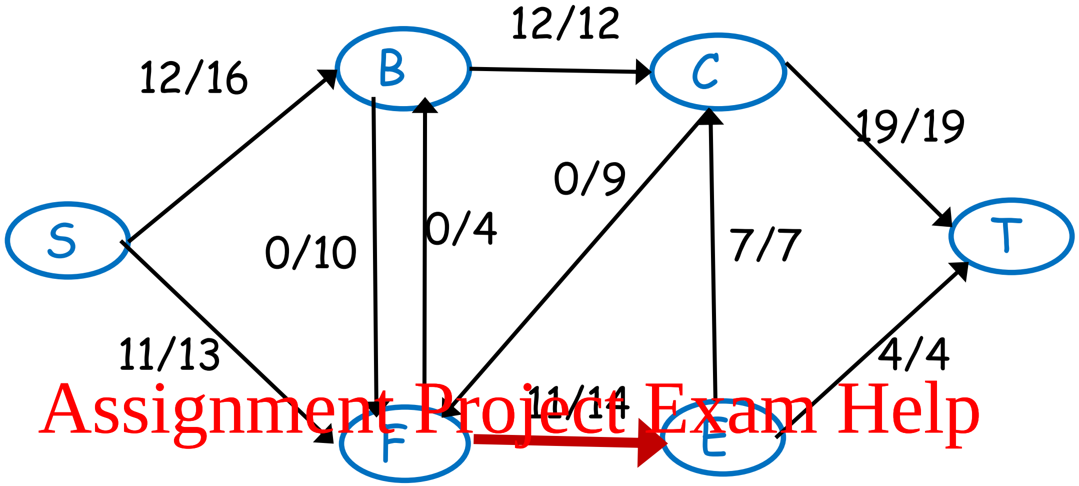
Is a min-cut unique?

$A = \{S, B, C, E, F\}$
$B = \{T\}$

Discussion Problem 2

You have successfully computed a maximum s-t flow for a network $G = (V, E)$ with positive integer edge capacities. Your boss now gives you another network G' that is identical to G except that the capacity of exactly one edge is decreased by one. You are also explicitly given the edge whose capacity was changed. Describe how you can compute a maximum flow for G' in linear time.

easy case

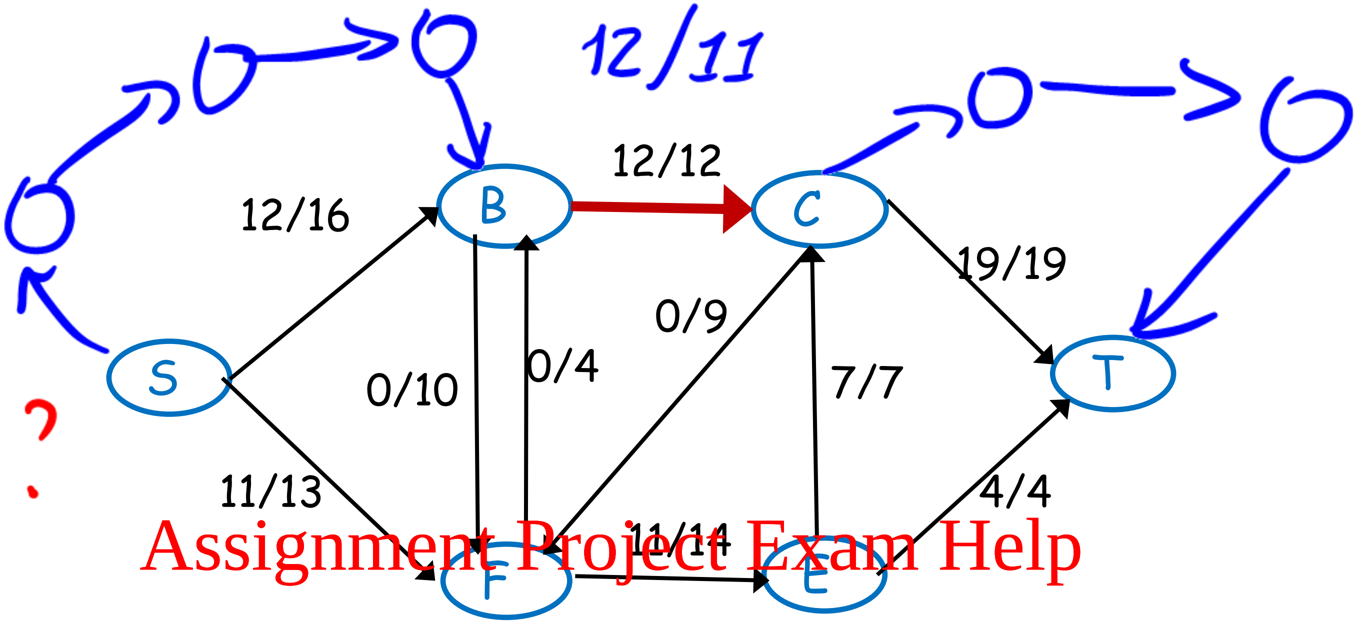


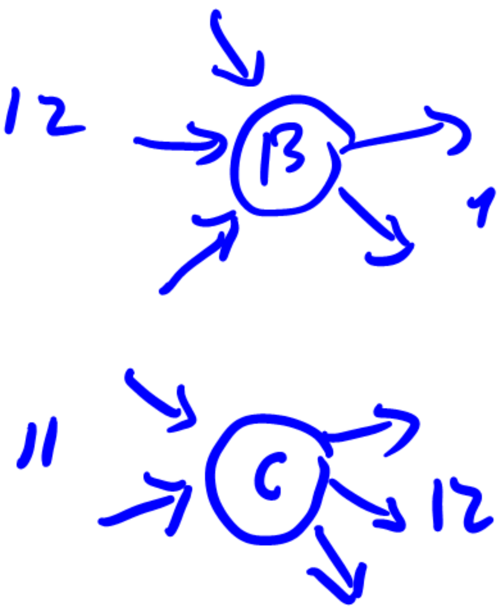
flow/cap

11/13

it is OK

is f^* max flow?



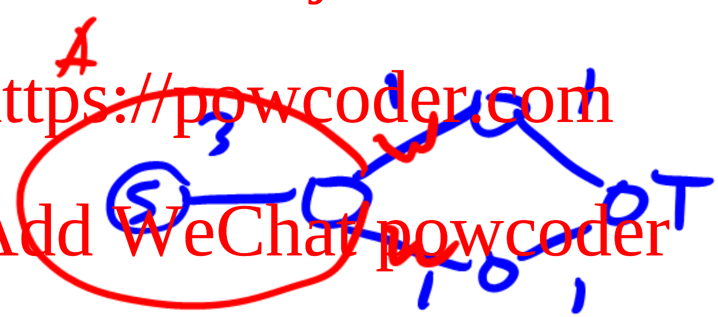


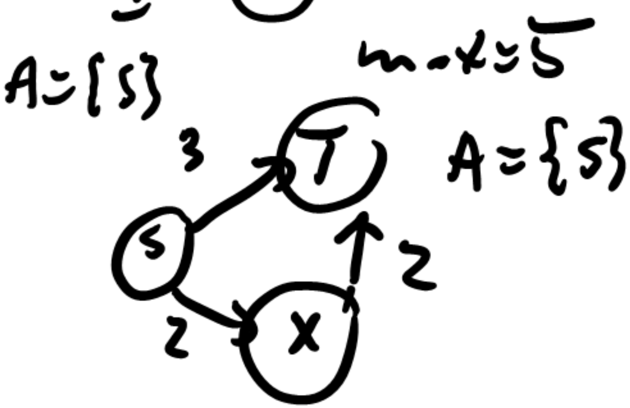
1. Find S-B path with non-zero flow
2. $f^* = f - 1$ on the path
3. Find C-T path with no zero flow
4. $f^* = f - 1$
5. Find S-T path if it exists.

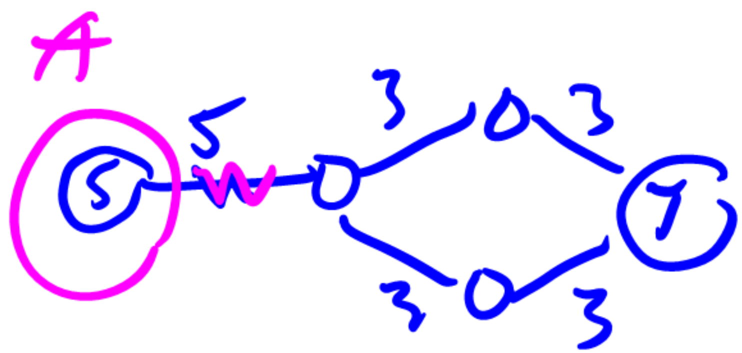
Discussion Problem 3

If we add the same positive number to the capacity of every directed edge, then the minimum cut (but not its value) remains unchanged. If it is true, prove it, otherwise provide a counterexample.

FALSE



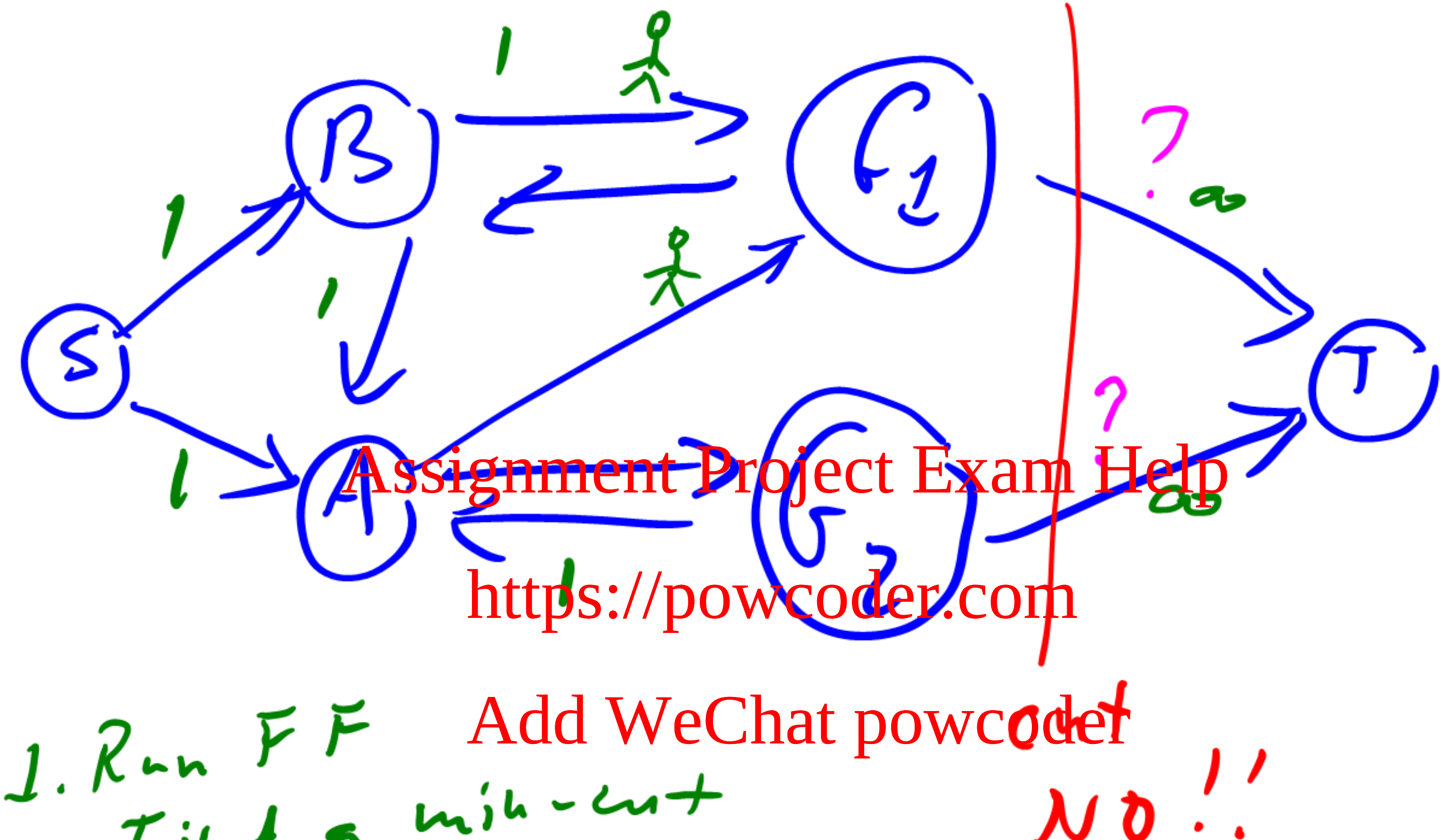




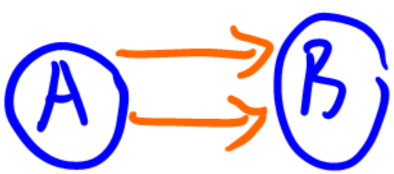
Discussion Problem 4

$NF = (V, E, s, t, c)$

In a daring burglary, someone attempted to steal all the candy bars from the CS department. Luckily, he was quickly detected, and now, the course staff and students will have to keep him from escaping from campus. In order to do so, they can be deployed to monitor strategic routes. Compute the minimum number of students/staff needed and show the monitored routes.



1. Run FF
2. Find a min-cut



ch. 7.3

Reduction

Formally, to reduce a problem Y to a problem X (we write $\boxed{Y \leq_p X}$) we want a function f that maps Y to X such that:

- f is a polynomial time computable

- $\forall$*instance* $y \in Y$ is solvable if and only if $f(y) \in X$ is solvable.

input(Y) $\xrightarrow{f}$ input(NF)

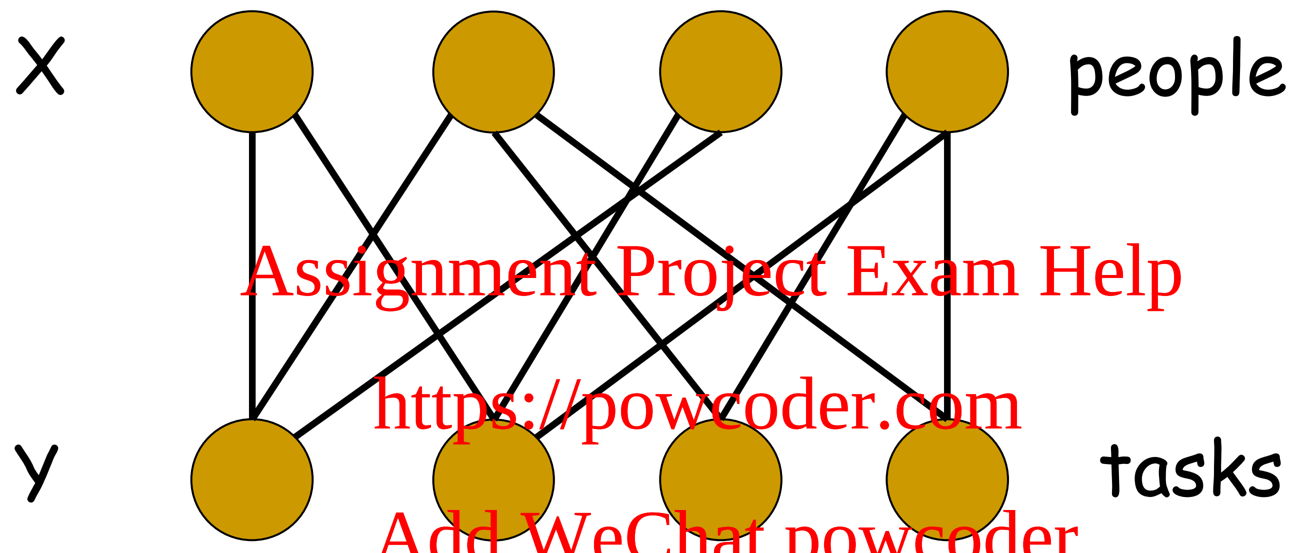
Run(NF-Algo)

output(Y) $\xleftarrow{f}$ output(NF)

Solving by reduction to NF

1. Describe how to construct a flow network
2. Make a claim. Something like "this problem has a feasible solution if and only if the max flow is ..."
3. Prove the above claim in both directions

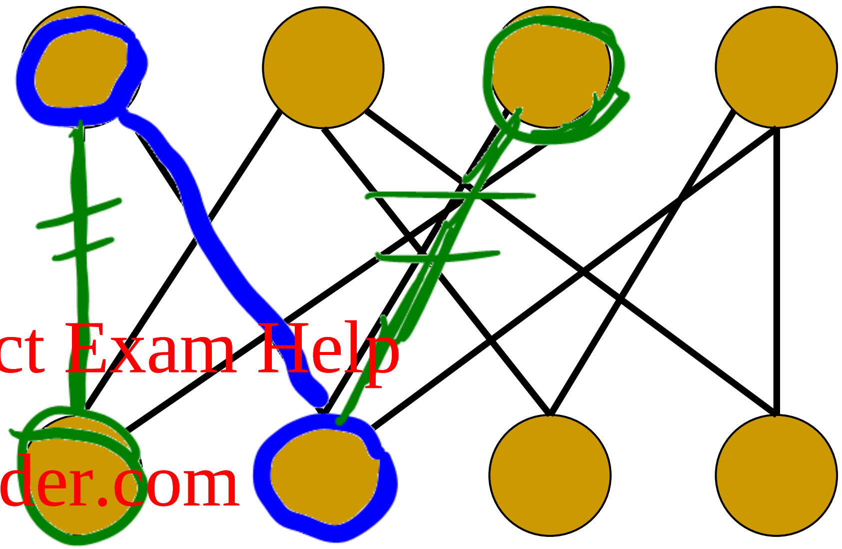
Bipartite Graph



A graph is bipartite if the vertices can be partitioned into two disjoint (also called independent) sets X and Y such that all edges go only between X and Y (no edges go from X to X or from Y to Y). Often writes G = (X, Y, E).

Bipartite Matching

Definition. A subset of edges is a matching if no two edges have a common vertex (mutually disjoint).



Definition. A maximum matching is a matching with the largest possible number of edges

Goal. Find a maximum matching in G.

Solving by Reduction

Given an instance of bipartite matching.

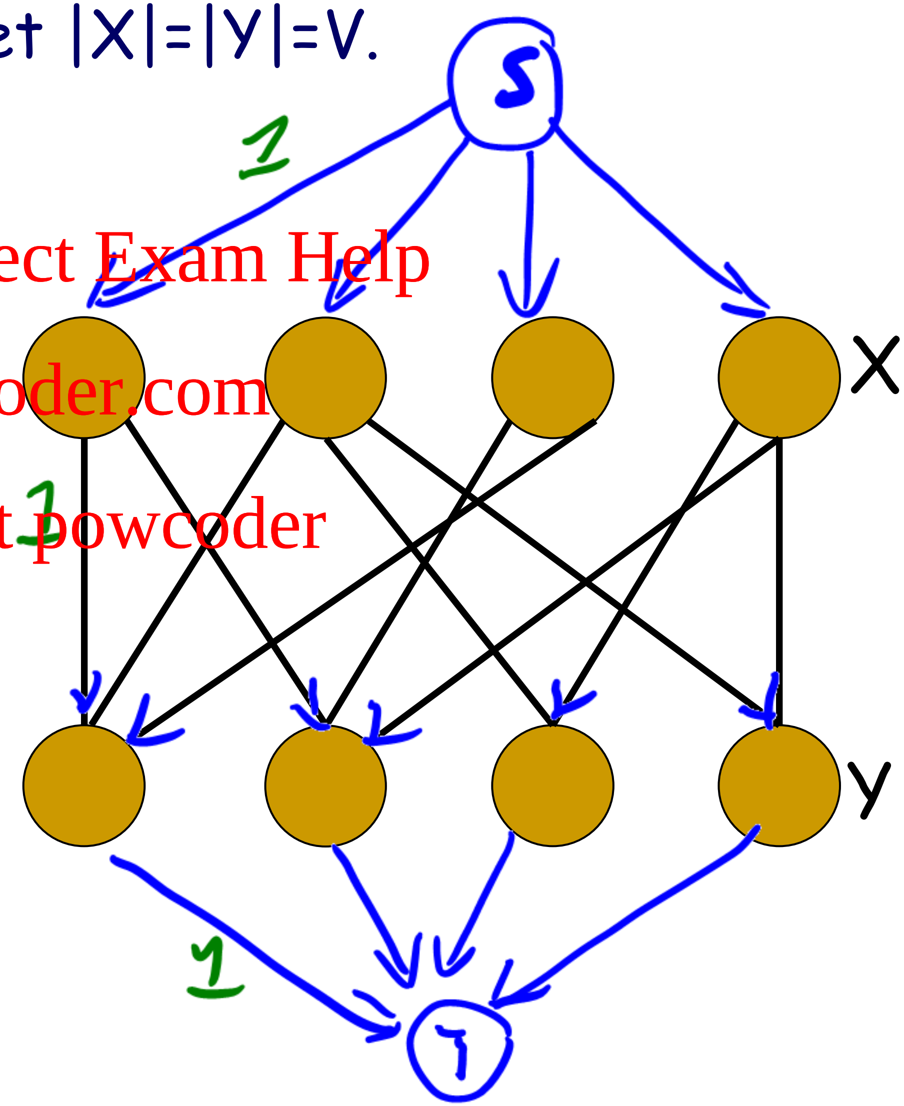
Create an instance of network flow.

The solution to the network flow problem can easily be used to find the solution to the bipartite matching.

Reducing Bipartite Matching to Network Flow

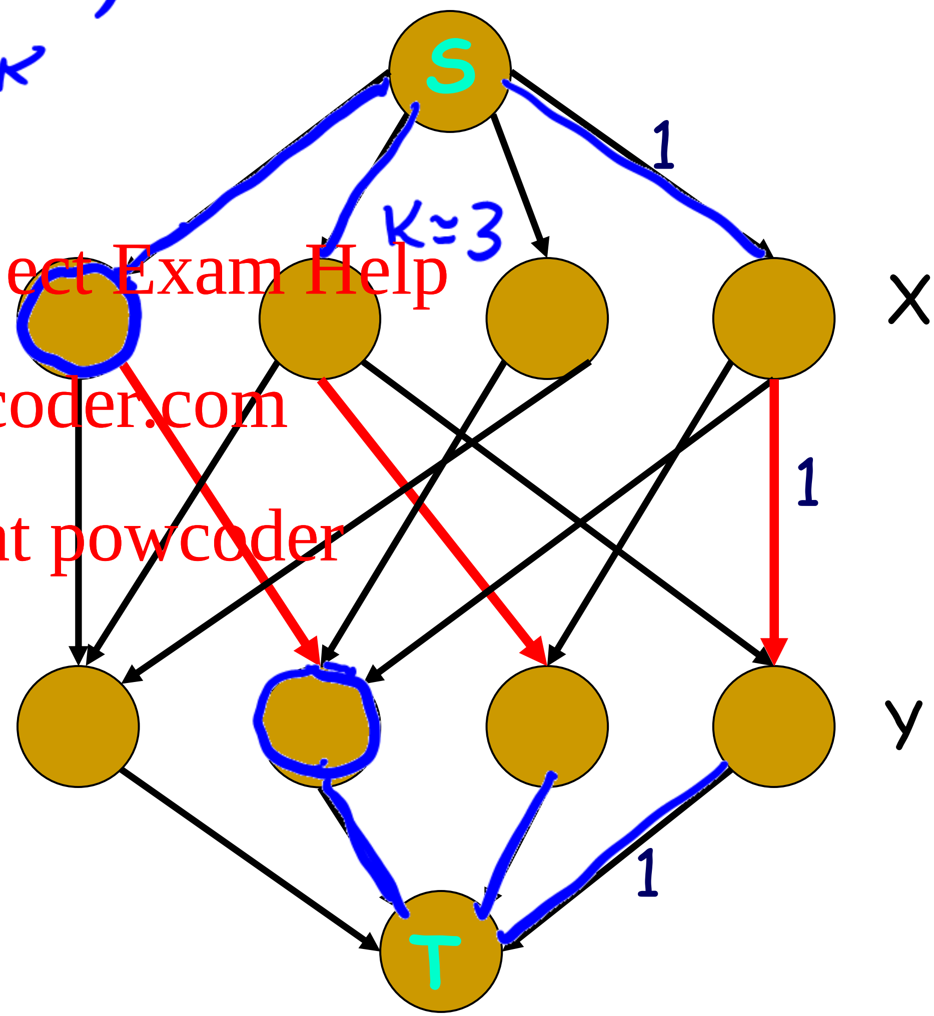
Given bipartite $G = (X, Y, E)$. Let $|X|=|Y|=V$.

Claim.
max-matching
=
max-flow



Max matching $\Rightarrow$ = Max flow

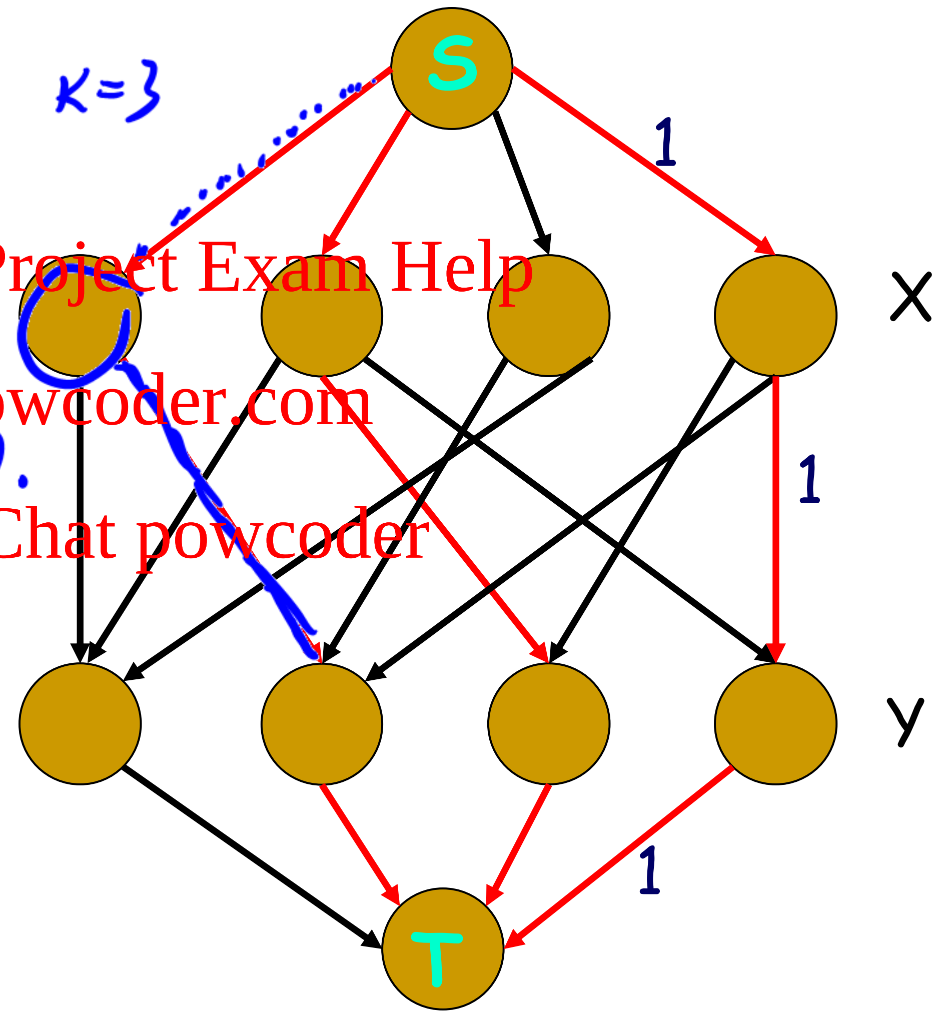
$\Rightarrow$ if $\exists$ a matching of k edges,
then $\exists$ a flow of value k



Max matching ⇐ Max flow

∃ a flow of value k, then ∃ a matching of size k.

k=3



To find a matching just follow the flow.

Runtime Complexity

Given bipartite $G = (X, Y, E)$. , $|X|=|Y|=V$.

?

$NF = (V_1, E_1)$

$V_1 = 2\cdot V + 2, \quad E_1 = E + 2\cdot V, \quad |f| = V$

$FF: O(|f|(V_1+E_1)) = O(|f|\cdot E_1) =$

$= O(V\cdot(E+2V)) = O(V\cdot E)$

is it polynomial?

Yes

$E = O(V^2)$

Discussion Problem 5

DIY

We're asked to help the captain of the USC tennis team to arrange a series of matches against UCLA's team. Both teams have n players; the tennis rating of the i-th member of USC's team is a_i and the tennis rating for the k-th member of UCLA's team is b_k. We would like to set up a competition in which each person plays one match against a player from the opposite school. Our goal is to make as many matches as possible in which the USC player has a higher tennis rating than his or her opponent. Give an algorithm to decide which matches to arrange to achieve this objective.

Player	Rating	Team
A	10	Trojans
B	5	Trojans
C	15	Trojans
D	20	Trojans
E	7	Bruins
F	14	Bruins
G	16	Bruins
H	19	Bruins

How to improve the efficiency of the Ford-Fulkerson Algorithm?

FF: run DFS

Edmonds-Karp: run BFS

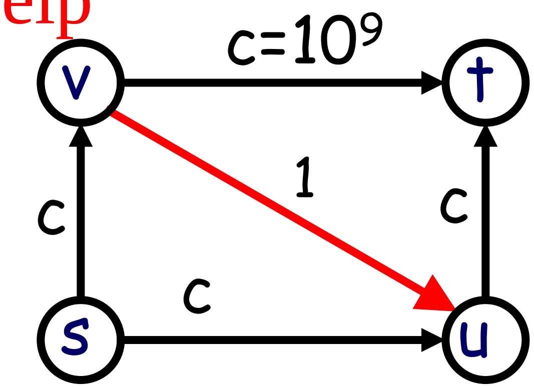
s-v-t

s-u-t

Runtime: $O(V \cdot E^2)$

polynomial!!

$O(|f|\ (E+V))$



Edmonds-Karp algorithm

Algorithm. Given (G, s, t, c)

1) Start with |f|=0, so f(e)=0
2) Find a shortest augmenting path in G_f
3) Augment flow along this path
4) Repeat until there is no an s-t path in G_f

BFS

Theorem.

The runtime complexity of the algorithm is $O(V E^2)$.

prove it!!

(without proof)

Runtime history

n = V, m = E,
U = |f|

14 years

year	discoverer(s)	bound	
1951	Dantzig [11]	$O(n^2 m U)$	
1956	Ford & Fulkerson [17]	O(m U)	
1970	Dinitz [13] Edmonds & Karp [15]	O(n m²)	shortest path
1970	Dinitz [13]	$O(n^2 m)$	
1972	Edmonds & Karp [15] Dinitz [14]	$O(m^2 \log U)$	capacity scaling
1973	Dinitz [14] Gabow [19]	$O(nm \log U)$	
1974	Karzanov [36]	$O(n^3)$	preflow-push
1977	Cherkassky [9]	$O(n^2 m^{1/2})$	
1980	Galil & Naamad [20]	$O(nm \log^2 n)$	
1983	Sleator & Tarjan [46]	$O(nm \log n)$	splay tree
1986	Goldberg & Tarjan [26]	$O(nm \log(n^2/m))$	preflow-push
1987	Ahuja & Orlin [2]	$O(nm + n^2 \log U)$	
1987	Ahuja et al. [3]	$O(nm \log(n\sqrt{\log U}/m))$	
1989	Cheriyan & Hagerup [7]	$E(nm + n^2 \log^2 n)$	
1990	Cheriyan et al. [8]	$O(n^3/\log n)$	
1990	Alon [4]	$O(nm + n^{8/3} \log n)$	
1992	King et al. [37]	$O(nm + n^{2+\epsilon})$	
1993	Phillips & Westbrook [44]	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$	
1994	King et al. [38]	$O(nm \log_{m/(n \log n)} n)$	
1997	Goldberg & Rao [24]	$O(\min(n^{2/3}, m^{1/2}) m \log(n^2/m) \log U)$	
2013	Orlin	O(m n)	

Discussion Problem 6

A company has n locations in city *A* and plans to move some of them (or all) to another city *B*. The i-th location costs a_i per year if it is in the city *A* and b_i per year if it is in the city *B*. The company also needs to pay an extra cost, $c_{ij} > 0$, per year for traveling between locations i and j. We assume that $c_{ij} = c_{ji}$. Design an efficient algorithm to decide which company locations in city *A* should be moved to city *B* in order to minimize the total annual cost.

Discussion Problem 7

We say that two paths are vertex-disjoint if they do not share any vertices (except s and t).

Given a directed graph $G = (V, E)$ with two distinguished nodes s, t. Design an algorithm to find the maximum number of vertex-disjoint s-t paths in G.

Discussion Problem 8

There are n students in a class. We want to choose a subset of k students as a committee. There has to be m_1 number of freshmen, m_2 number of sophomores, m_3 number of juniors, and m_4 number of seniors in the committee. Each student is from one of k departments, where $k = m_1 + m_2 + m_3 + m_4$. Exactly one student from each department has to be chosen for the committee. We are given a list of students, their home departments, and their class (freshman, sophomore, junior, senior). Describe an efficient algorithm based on network flow techniques to select who should be on the committee such that the above constraints are all satisfied.