Discussion 10

CSCI 570

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

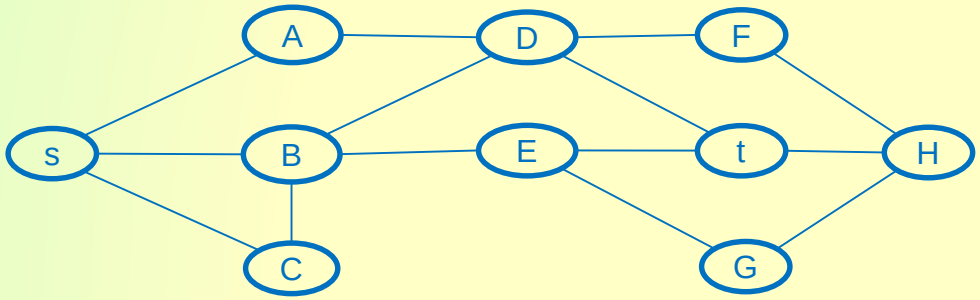
DISCUSSION 10

Outline

- Problems with Solutions

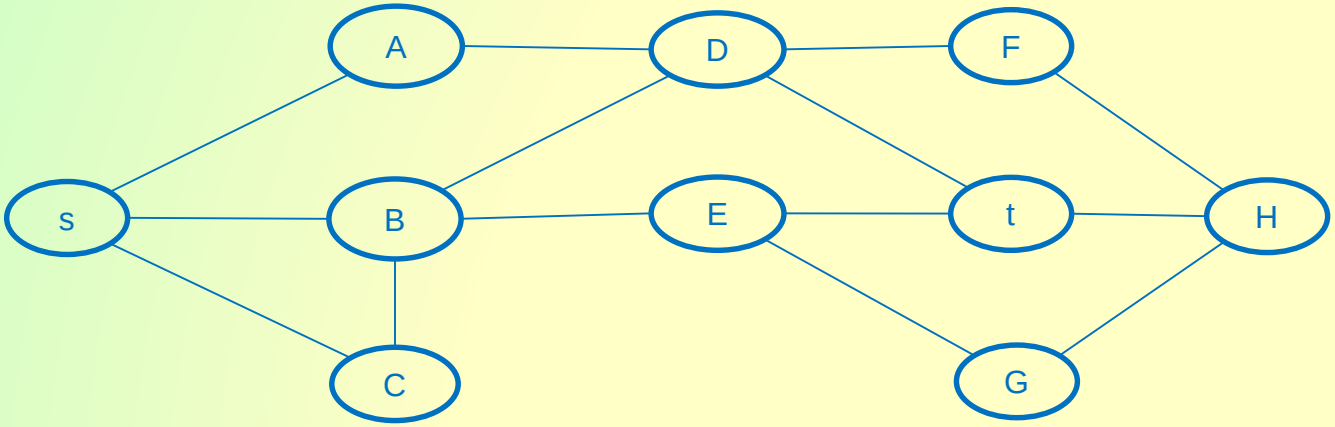
Problem 1

- Professor Jones has determined that *x* priceless artifacts (which belong in a museum) are located in a labyrinth. The labyrinth can be thought of as a graph, with each edge representing a path and each node an intersection of paths. All of the artifacts are in the same treasure room, which is located at one of the intersections. However, the artifacts are extremely burdensome, so Jones can only carry one artifact at a time. There is only one entrance to the labyrinth, which is also a node in the graph. The entrance serves as the only exit as well. All the paths (i.e. edges) are protected by human-eating vines, which will be woken up after someone passes the path, so Jones can only go through each path (i.e. edge) once.
- To obtain a single artifact, Jones would have to find a tour which starts at the entrance, reaches the artifact and leads back again, without getting Jones captured by the vines. If he tries to get another artifact, he must find another tour which does not use the edges he's already traversed. Given the map of the labyrinth and the location of the treasure, give a polynomial-time algorithm that determines how many artifacts Jones can obtain (and how to do so). Justify your algorithm.



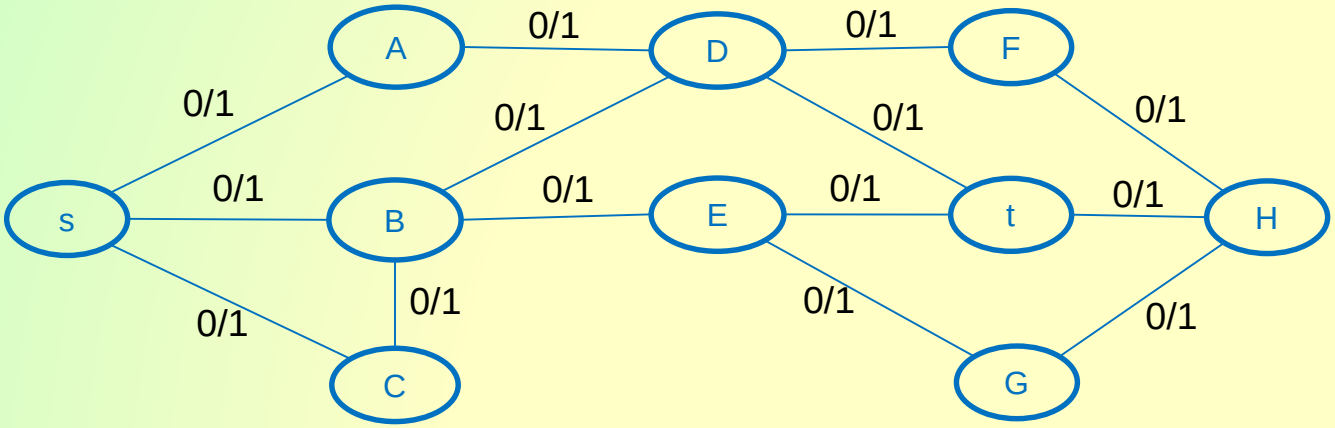
Solution 1

- Assume the entrance/exit is at node s, and the treasure room is at node t
- The total number of artifacts we can obtain is half the number of paths between s and t since we need to go both ways between s and t (since we have an undirected graph)
  - So we need to find how many edge-disjoint paths p exist between s and t
  - The number of artifacts we can obtain is floor(p/2)
- There is a chance that floor(p/2) > x, so the actual answer for the number of artifacts we can obtain is min(floor(p/2), x)
- Consider the following graph



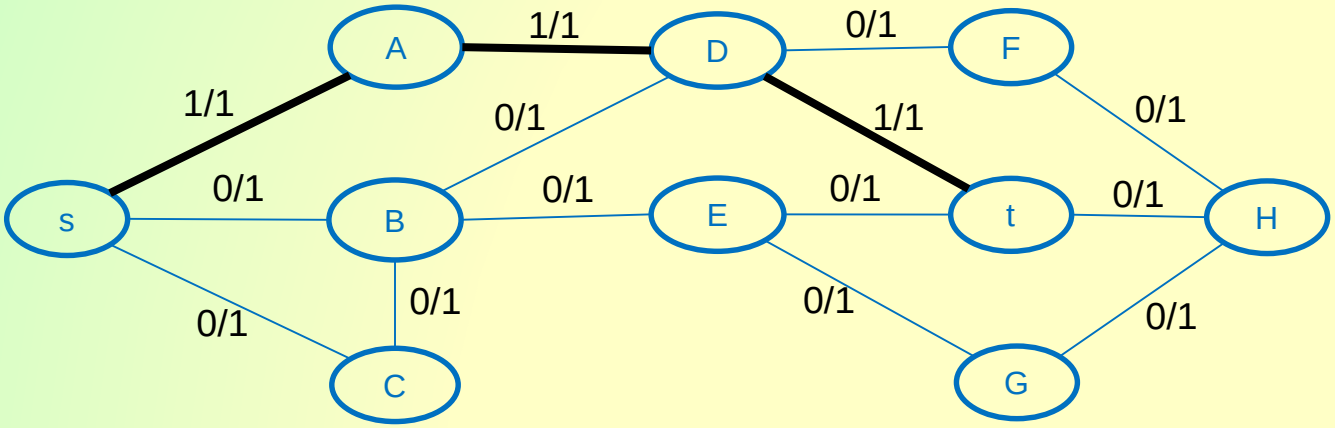
Solution 1

- To find the edge-disjoint paths, set the capacity on all of the edges to 1 and run a max-flow algorithm



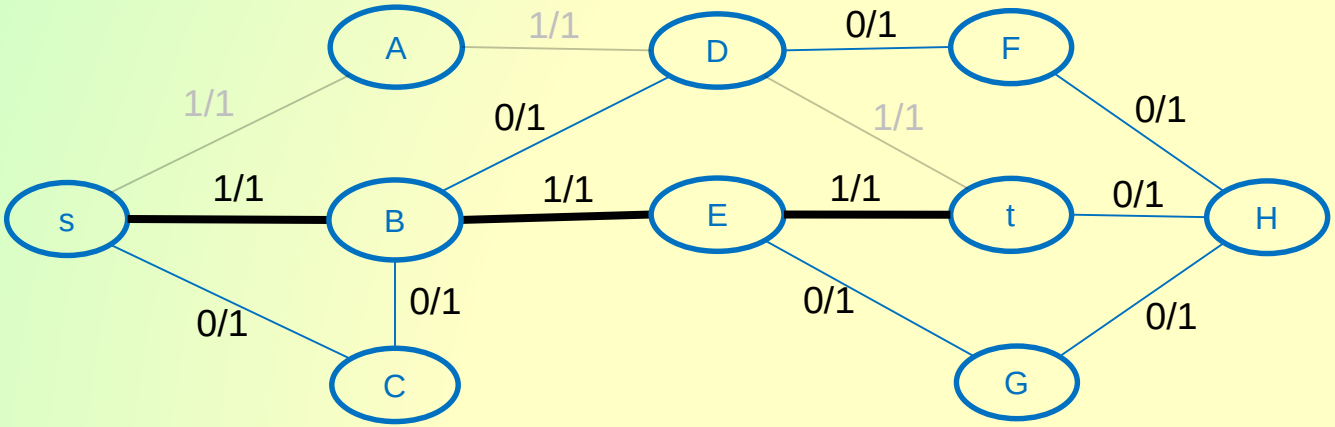
Solution 1

- To find the edge-disjoint paths, set the capacity on all of the edges to 1 and run a max-flow algorithm
  - s-A-D-t with flow=1



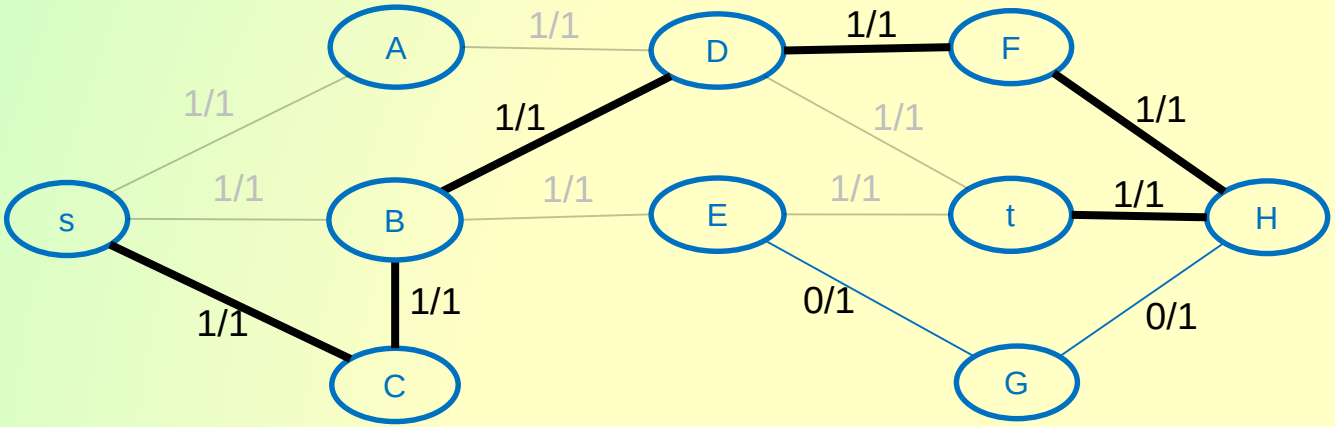
Solution 1

- To find the edge-disjoint paths, set the capacity on all of the edges to 1 and run a max-flow algorithm
  - s-A-D-t with flow=1
  - s-B-E-t with flow=1



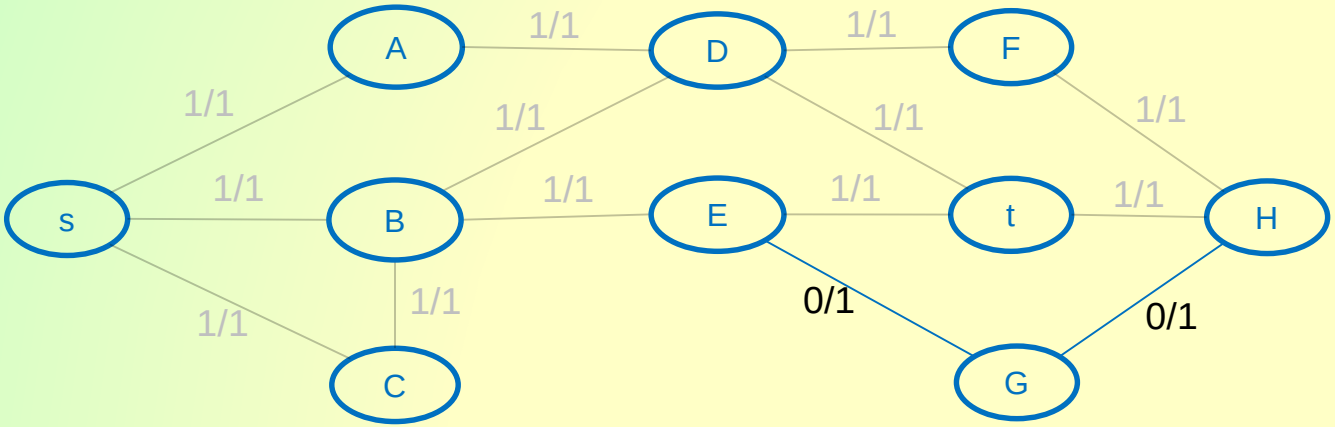
Solution 1

- To find the edge-disjoint paths, set the capacity on all of the edges to 1 and run a max-flow algorithm
  - s-A-D-t with flow=1
  - s-B-E-t with flow=1
  - s-C-B-D-F-H-t with flow=1



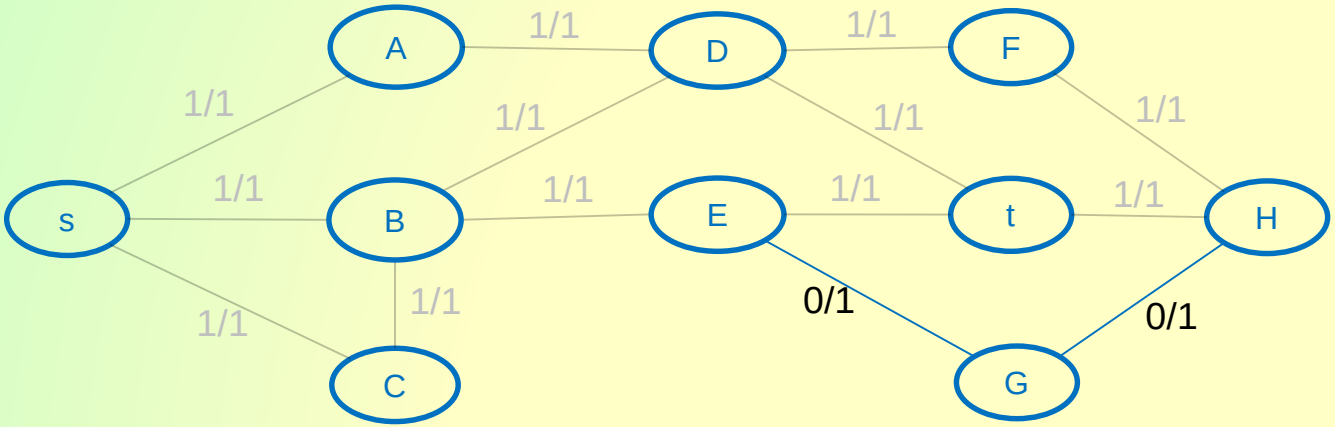
Solution 1

- To find the edge-disjoint paths, set the capacity on all of the edges to 1 and run a max-flow algorithm
  - s-A-D-t with flow=1
  - s-B-E-t with flow=1
  - s-C-B-D-F-H-t with flow=1
  - No other paths exist with any remaining capacity from s to t
    - We already knew this would be the case based on the value of the min-cut, which was 3



Solution 1

- Now we know the number of edge-disjoint paths p from s to t is 3
- The total number of artifacts we can obtain is floor(p/2) = 1 (assuming that value is larger than the number of artifacts at location t)



Problem 2

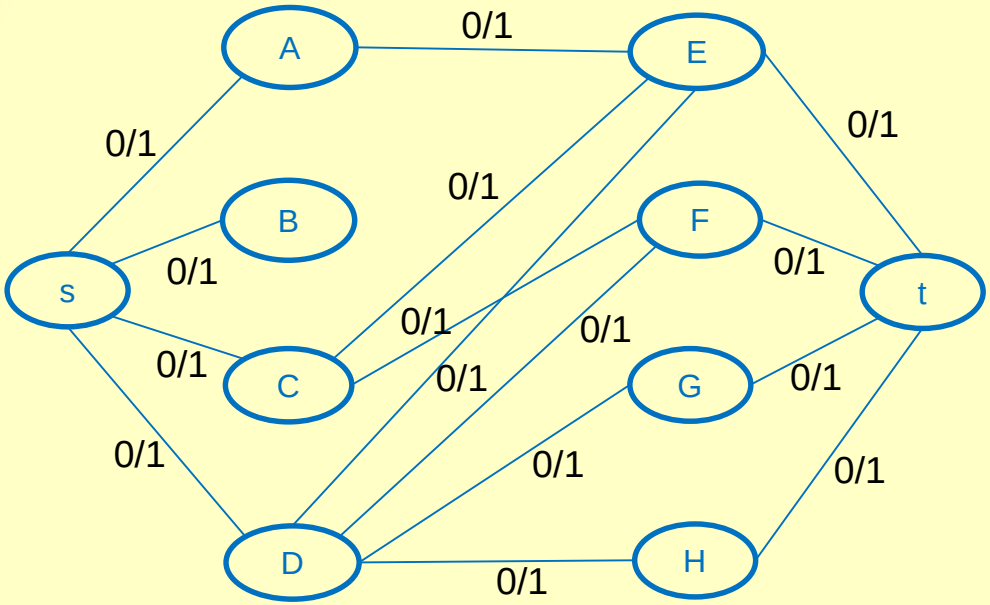
- We're asked to help the captain of the USC tennis team arrange a series of matches against UCLA's team. Both teams have *n* players. The tennis rating (a positive number, where a higher number can be interpreted to mean a better player) of the *i*th member of USC's team is t_i and the tennis rating 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.
- Because we get to select who plays against whom, our goal is to make sure that in *as many matches as possible*, the USC player has a higher tennis rating than his or her opponent. Use network flow to give a polynomial-time algorithm to decide which matches to arrange to achieve this objective.

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

Solution 2

- Create a bipartite graph with *n* vertices for Trojan players, and *n* vertices for Bruins players
- Add an edge from each Trojan vertex to each Bruin vertex where the Trojan player has a higher rating than the Bruin player
  - A can play against E
  - B can play against no one
  - C can play against E, F
  - D can play against E, F, G, H
- Create a source node and connect it to each Trojan
- Create a sink node and connect it to each Bruin
- Give each edge a capacity of one

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

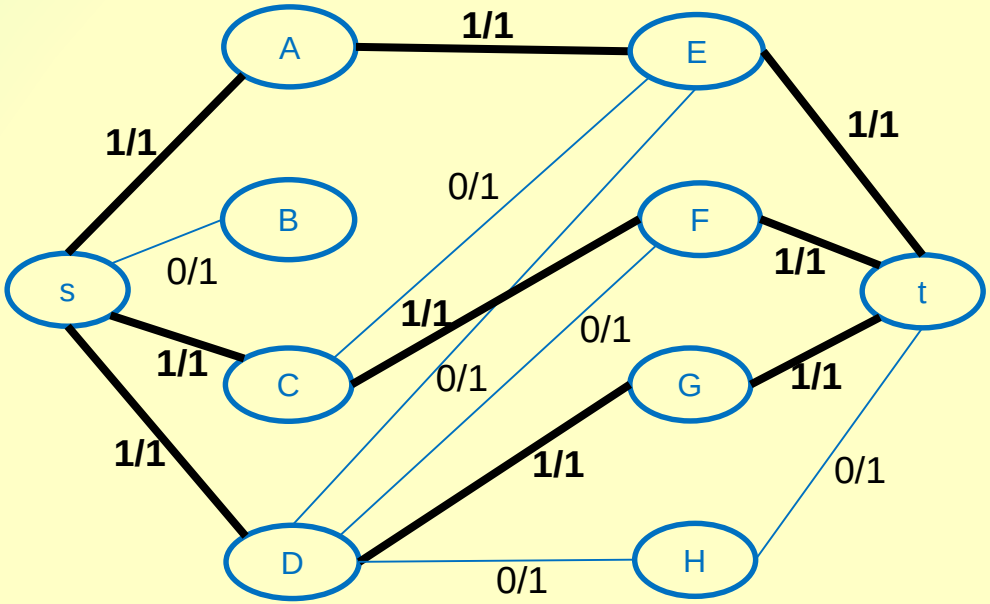


Solution 2

- Run any polynomial-time version of network flow (actually, even the basic Ford-Fulkerson would be poly time here, because *C* is *O(n)* for this graph)
- The edges with flow are the matches that will be set up
  - Due to conservation of flow, each player is involved in at most one match (as each player node has either one total incoming or one total outgoing capacity)
  - Unmatched players can be paired in any way desired (or randomly)

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

Trojan Player	Bruin Player
A (10)	E (7)
B (5)	H (19)
C (15)	F (14)
D (20)	G (16)



Problem 3

- Suppose I am teaching a large class and have many TAs. I want a total of *k* office hours to be held each week, and I have a set of *k* hour-long time intervals $i_1, i_2, \ldots i_k$ in which the TA office hours room is available. I have collected from each TA a list of which subset of the time intervals he or she can hold office hours. I also know, for each TA *j*, the maximum M_j hours he or she can hold office hours, and I have for each a minimum m_j hours he or she should be required to hold. Lastly, I have some value *H* for the total number of hours I'd like held during the week.
- Design a polynomial-time algorithm that takes as input an instance of this problem (the time slots, the TA schedules, the sets of M_j and m_j values, and the value *H*) and determines if there is a valid way to schedule the TA's office hours to respect these constraints.

TA	TA Availability	Minimum hours	Maximum hours
A	4-5, 5-6	1	2
B	5-6, 6-7	1	2
C	5-6, 6-7, 7-8	1	3
D	7-8, 8-9	1	2

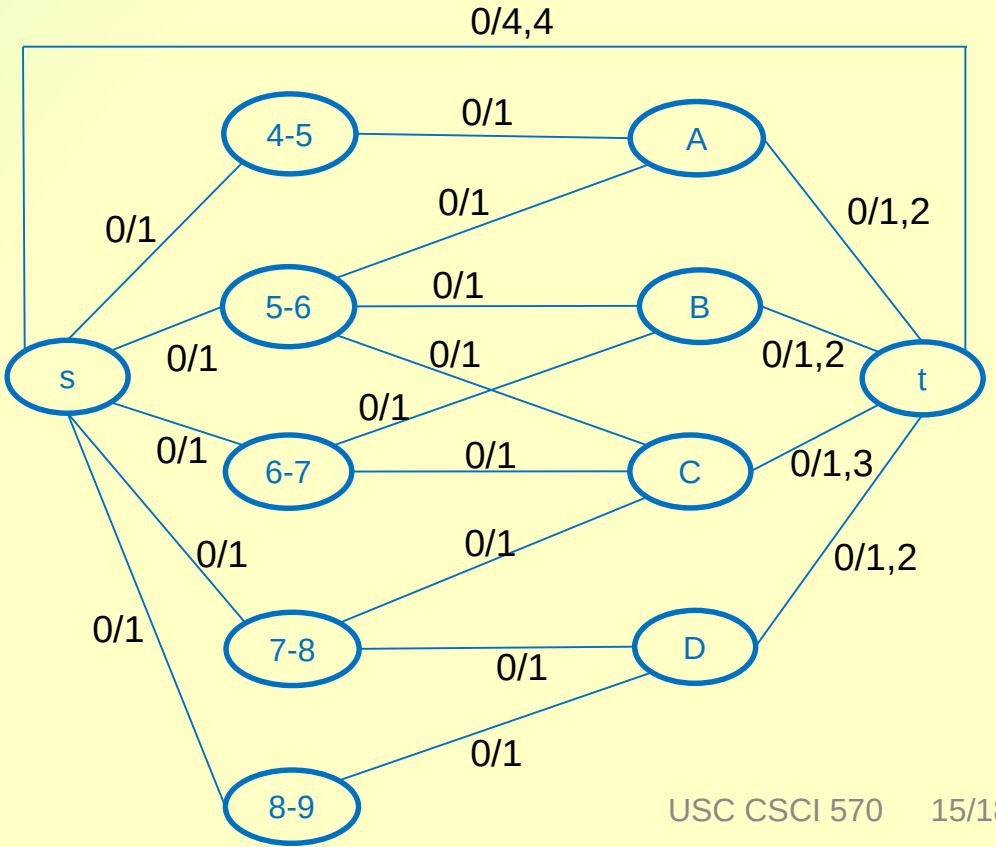
TA office is available from 4-5, 5-6, 6-7, 7-8, 8-9
Total number of hours I would like is 4

Solution 3

- This is a bipartite matching problem
- Create a node for each of the k time intervals and each TA
- Add an edge between a time interval node and TA with a capacity of 1 if the TA can cover it
- Create a source node connected to each time interval node with a capacity of 1
- Create a sink node connected from each TA with a minimum capacity and maximum capacity based on their available hours
- Connect the sink to the source with minimum and maximum capacity of H

TA	TA Availability	Minimum hours	Maximum hours
A	4-5, 5-6	1	2
B	5-6, 6-7	1	2
C	5-6, 6-7, 7-8	1	3
D	7-8, 8-9	1	2

TA office available 4-5, 5-6, 6-7, 7-8, 8-9
Total number of hours I would like is 4

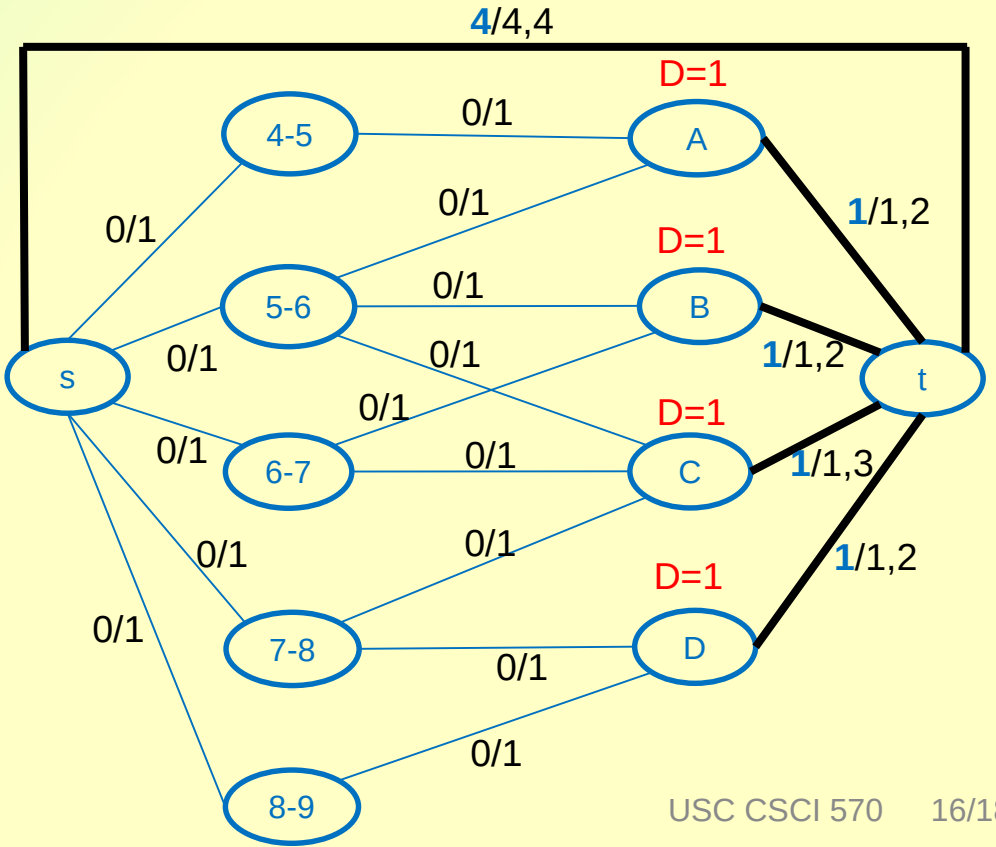


Solution 3

- Send flow of 1 along A-t, B-t, C-t, and D-t since that is the minimum capacity for those edges
- That would then force a flow of 4 along t-s, which is what we wanted
- This puts a demand D=1 on nodes A, B, C, and D

TA	TA Availability	Minimum hours	Maximum hours
A	4-5, 5-6	1	2
B	5-6, 6-7	1	2
C	5-6, 6-7, 7-8	1	3
D	7-8, 8-9	1	2

TA office available 4-5, 5-6, 6-7, 7-8, 8-9
Total number of hours I would like is 4

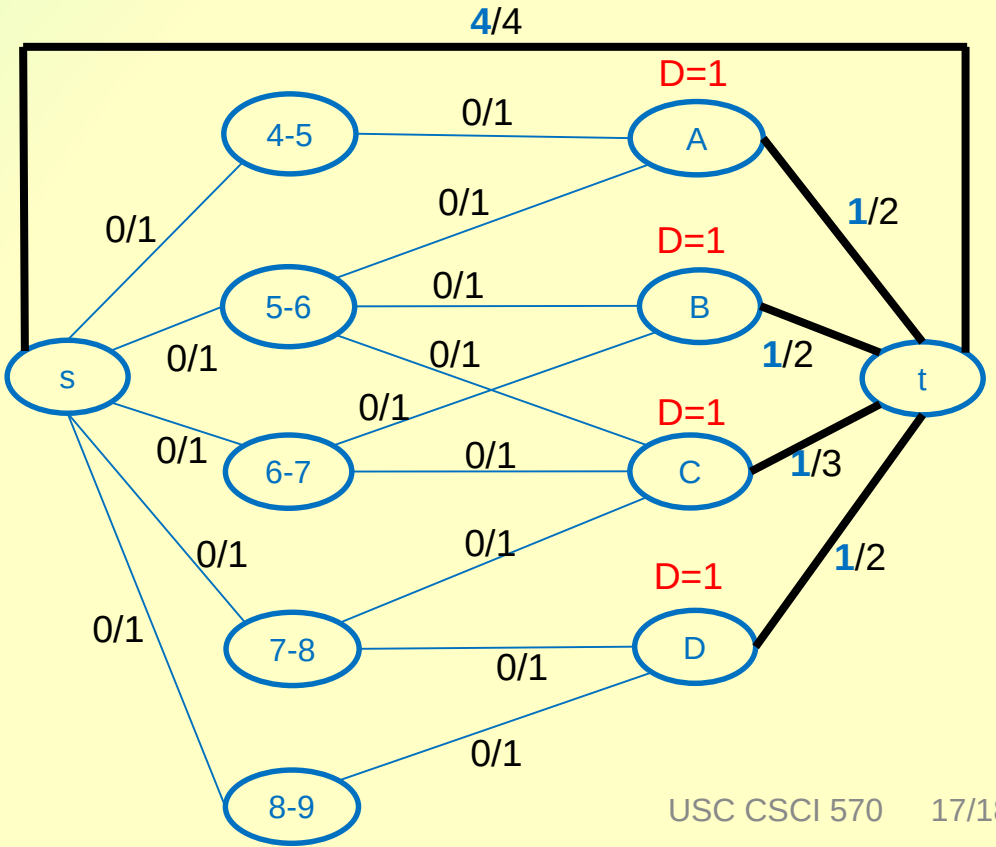


Solution 3

- We can now remove the minimum capacities and model this graph as a max flow problem
- Use any max-flow algorithm to solve this problem with the catch that there is not an infinite flow originating from the source but just a flow of finite value=4
- As long as the flow satisfies the demands, the max-flow will provide a solution to the problem

TA	TA Availability	Minimum hours	Maximum hours
A	4-5, 5-6	1	2
B	5-6, 6-7	1	2
C	5-6, 6-7, 7-8	1	3
D	7-8, 8-9	1	2

TA office available 4-5, 5-6, 6-7, 7-8, 8-9
Total number of hours I would like is 4



Solution 3

- Call a circulation algorithm to determine the hours the TAs will have office hours
  - › If there is a valid circulation, you have found the matching of TAs to intervals

TA	TA Availability	Minimum hours	Maximum hours	Hours
A	4-5, 5-6	1	2	4-5
B	5-6, 6-7	1	2	5-6
C	5-6, 6-7, 7-8	1	3	6-7
D	7-8, 8-9	1	2	8-9

TA office available 4-5, 5-6, 6-7, 7-8, 8-9
Total number of hours I would like is 4