CSE 417
Network Flows (pt 2)
Modeling with Max Flow

UNIVERSITY *of* WASHINGTON

Reminders

- **HW6 is due on Friday**
  - start early
  - may take time to figure out the sub-structure

Review of last lecture

> Defined the maximum flow problem
  - find the feasible flow of maximum value
  - flow is feasible if it satisfies edge capacity and node balance constraints

> Described the Ford-Fulkerson algorithm
  - starts with a feasible flow (all zeros) and improves it (by augmentations)
  - non-greedy: augmentations can *undo* flow added by previous augmentations
  - essentially optimal if max capacity on edges into t is O(1)
  - have not yet proven it correct...

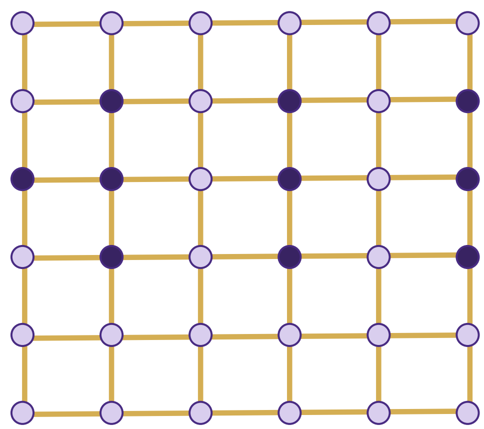
> Many, many other algorithms...

Outline for Today

- Escape Problem
- Covering with Dominos
- Token Placing
- Processor Scheduling
- Airline Scheduling

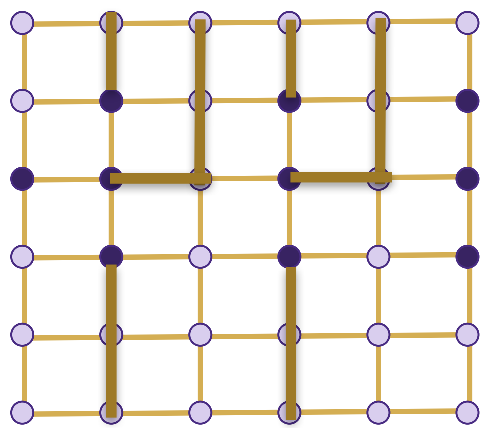
Escape Problem

> **Problem**: Given a set of points $(x_1, y_1), \ldots, (x_n, y_m)$ on an n x n grid, determine whether there exists a set of paths along grid lines from each of the points to the boundary that *do not intersect*



Escape Problem Example

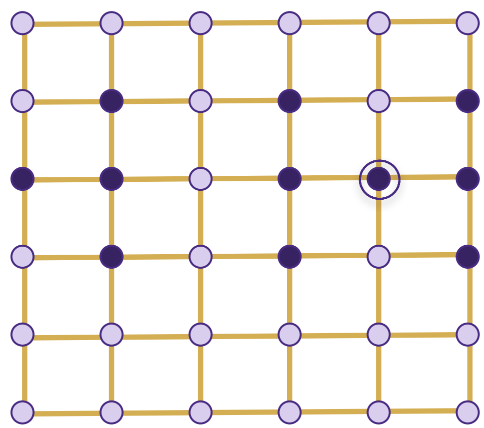
This example has a solution:



from “Introduction to Algorithms”

Escape Problem Example 2

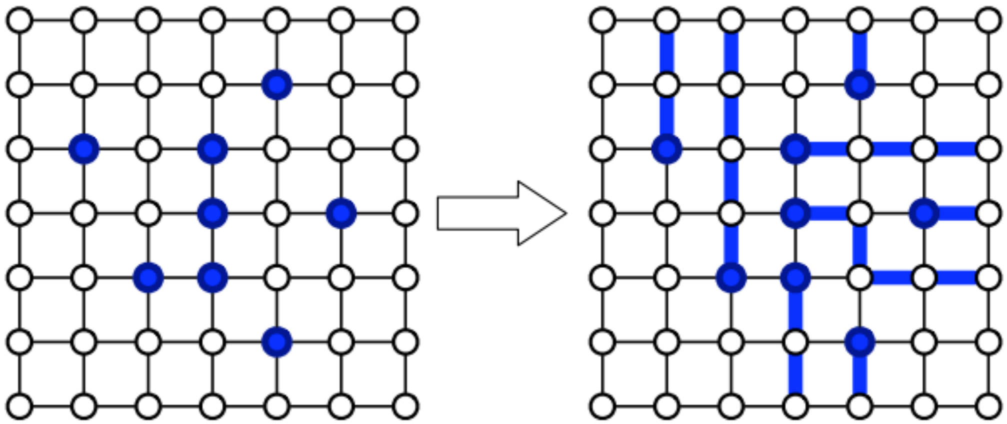
This example does not:



from “Introduction to Algorithms”

Escape Problem Example 3

Another with a solution:



from http://jeffe.cs.illinois.edu/teaching/algorithms/notes/24-maxflowapps.pdf

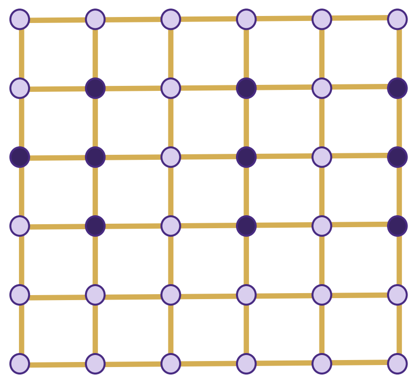
Robber Problem

- **Problem**: Bank robbers are planning to rob a number of banks around the city at exactly the same time. They will be pursued by the police as they flee. Find *escape routes* for the robbers that do not use any of the same roads or intersections.
  - (one robber doesn't want to run into the police chasing another robber)
- Other assumptions:
  - city map is given as a graph with intersections as nodes and roads as edges
  - can assume all the banks are at the corners of intersections

Robber Problem

> Generalization of the previous problem...

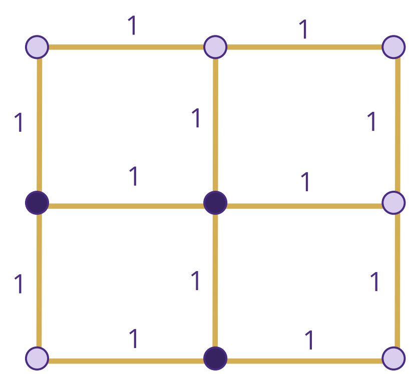
> Here's what it might like look like if the streets are a grid:



> (from an ACM programming contest ~1997)

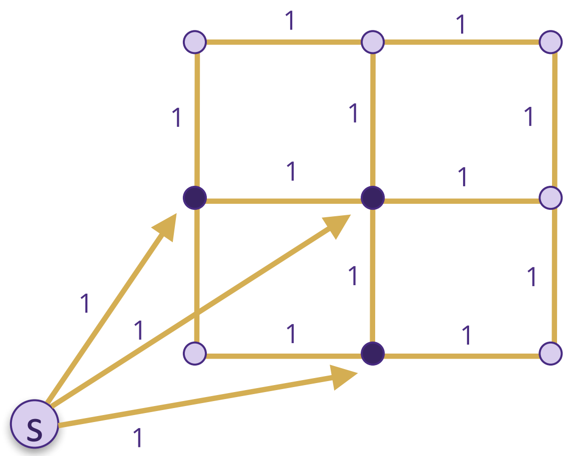
Robber Problem

> Solve by modeling with max flow:
  - roads become edges (in both direction) with capacity 1



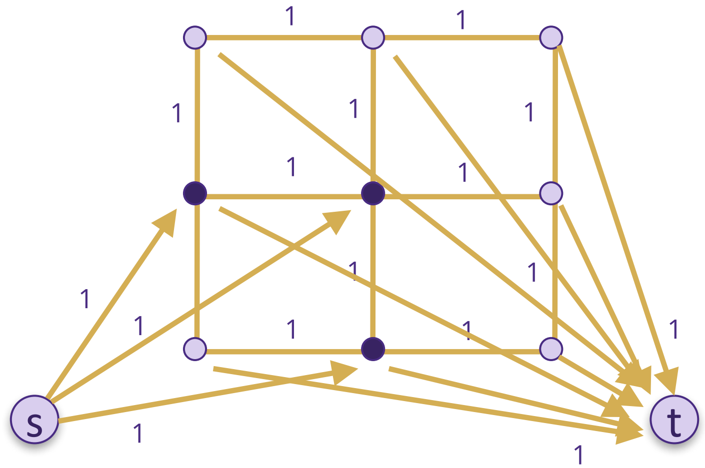
Robber Problem

> Solve by modeling with max flow:
  - roads become edges (in both direction) with capacity 1
  - source node has edges to each bank with capacity 1



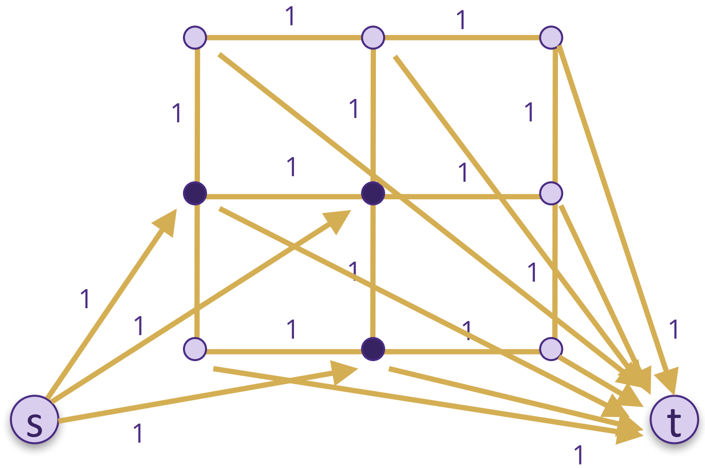
Robber Problem

- Solve by modeling with max flow:
  - roads become edges (in both direction) with capacity 1
  - source node has edges to each bank with capacity 1
  - edges from boundary nodes to the sink with capacity 1

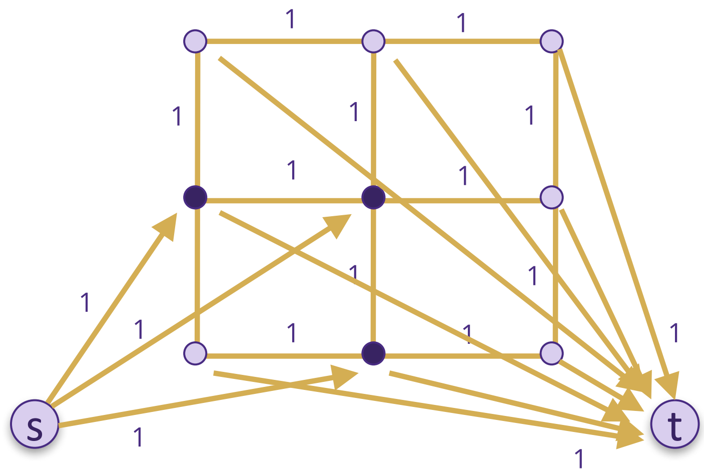


Robber Problem

- Solve by modeling with max flow
- Has a solution iff the max flow equals the number of robbers



Robber Problem

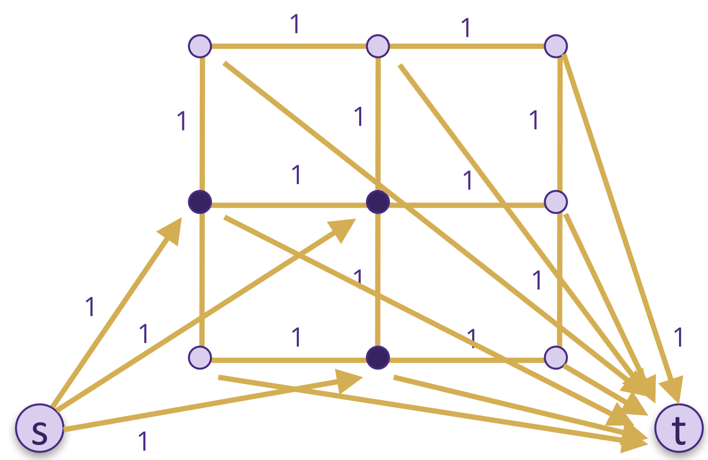


> Solve by modeling with max flow

> Has a solution iff the max flow equals the number of robbers:
  - can assume flow is {0, 1} on each edge — (by F-F)
  - flow on each edge (s,u) gives a path from u to the boundary…
    > flow into u on (s,u) leaves on some edge (u,v)
    > flow into v on (u,v) leaves on some edge (v,w)
    > can only stop with an edge (z,t), and z is a boundary node by construction
  - two paths using the same edge would violate edge capacity

(not as obvious as it sounds…)

Robber Problem



> Solve by modeling with max flow

> Has a solution iff the max flow equals the number of robbers

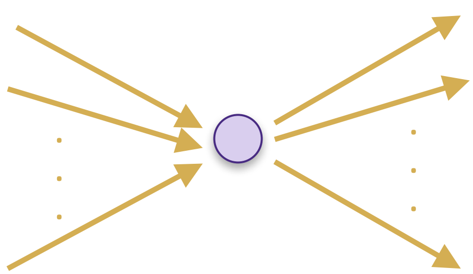
> **Q**: What's missing?

> **A**: Two paths could use the same intersection!

- as long as the paths enter and leave via different <u>edges</u>, it would be allowed
- somehow need to put a capacity on <u>nodes</u> also

Node Capacity Constraints

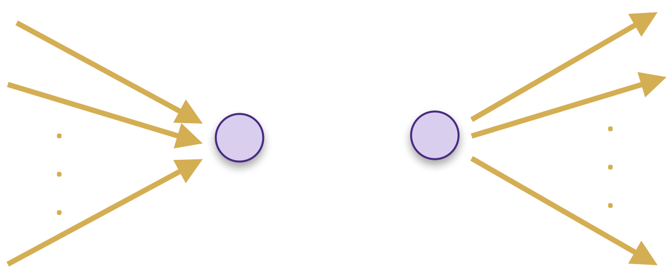
- These are easy to add to any max flow problem
- Consider any node…
  - it has some number of incoming edges and outgoing edges



Node Capacity Constraints

> These are easy to add to any max flow problem

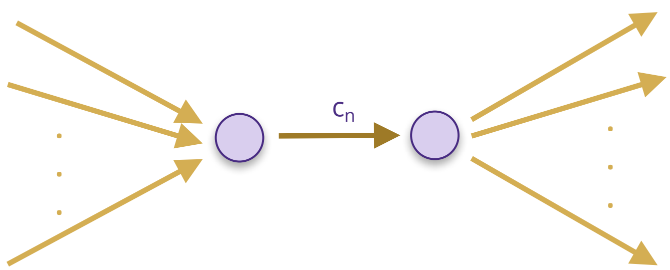
> Consider any node…
  – split it into two parts
  – one part for incoming edges and one for outgoing edges



Node Capacity Constraints

> These are easy to add to any max flow problem

> Consider any node…
  - split it into two parts
  - one part for incoming edges and one for outgoing edges
  - add edge between them with capacity for the node



Node Capacity Constraints

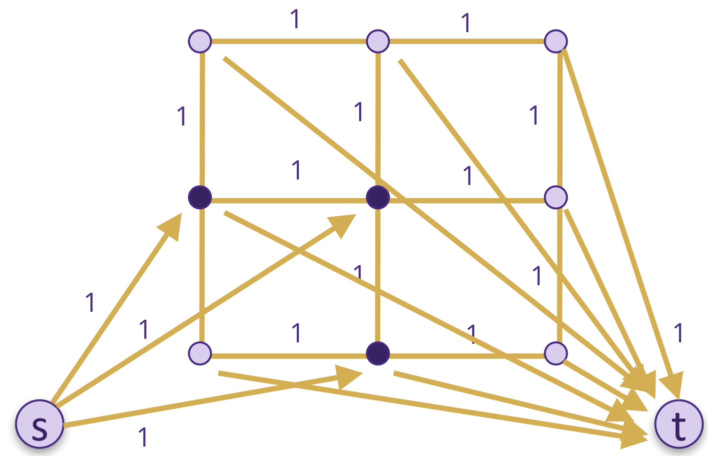
> These are easy to add to any max flow problem

> Consider any node…
  - split it into two parts
  - one part for incoming edges and one for outgoing edges
  - add edge between them with capacity for the node

> All flow through the node now goes through this internal edge

> That allows us to limit the total flow using the node
  - node balance constraint is preserved…
  - flow balance constraint on the two nodes tell us:
    flow into the first node = flow along internal edge = flow out of the second node

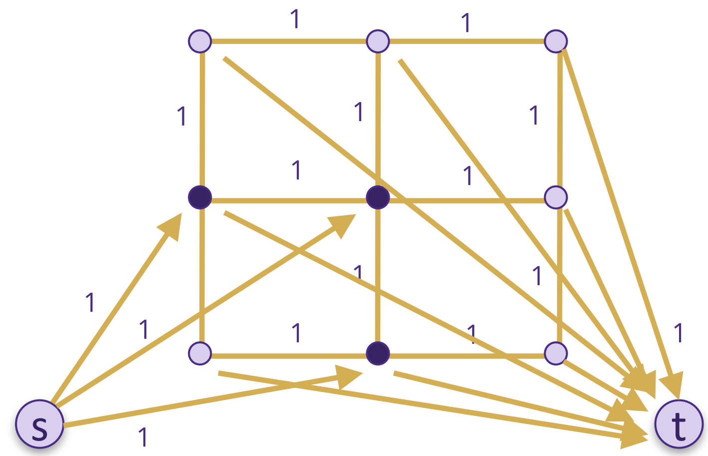
Robber Problem



> Solve by modeling with max flow
  – previous construction
  – plus node capacities of 1

> Has a solution iff the max flow equals the number of robbers
  – already proved that paths must be edge-disjoint
  – node capacity means paths must be node-disjoint as well

Robber Problem



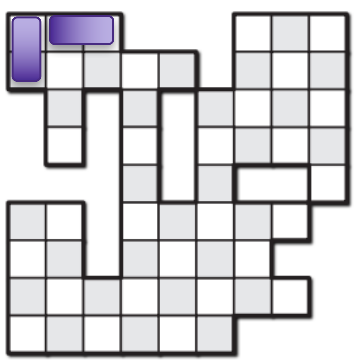
- > Solve by modeling with max flow
  - – previous construction
  - – plus node capacities of 1
- > Has a solution iff the max flow equals the number of robbers
  - – we have shown that this properly models the robber problem:
    - > every solution to the robber problem corresponds to a flow of value #robbers
    - > every 0/1 flow of value #robbers encodes escape paths for all robbers
- > Ford-Fulkerson runs in O(nm) time since U = 1

Outline for Today

> Escape Problem
> Covering with Dominos
> Token Placing
> Processor Scheduling
> Airline Scheduling

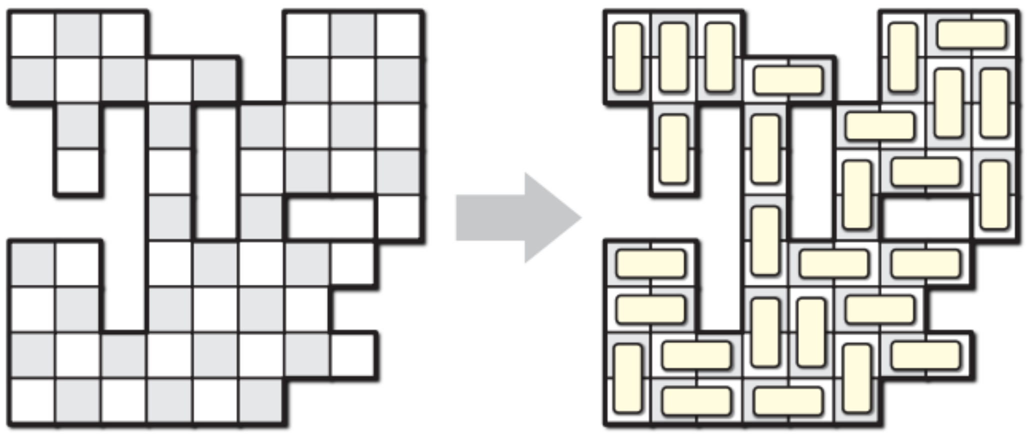
Covering with Dominos

- **Problem**: Given a checker board with some squares *deleted*, find a way to cover the board with dominos.
  - each domino covers two adjacent squares (vertical or horizontal)



Covering with Dominos Example

> **Problem**: Given a checker board with some squares *deleted*, find a way to cover the board with dominos.



from http://jeffe.cs.illinois.edu/teaching/algorithms/notes/24-maxflowapps.pdf

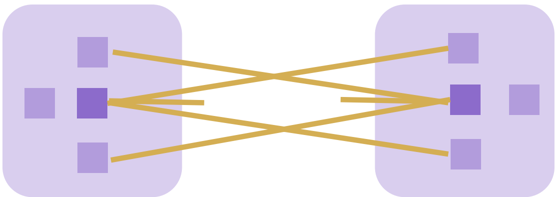
Covering with Dominos

> **Observation**: This looks like a matching problem
>   – each domino connects a *pair* of adjacent squares

> Unfortunately, it looks like a <u>general</u> matching problem
>   – graph has a node for each square and edges between (≤ 4) adjacent ones
>   – as noted before, the problem is harder than max flow on a general graph

Covering with Dominos: False Start

- **Observation**: This looks like a general matching problem
- Could try putting every square on both the left & right side
  - allow matching squares on left and right only to adjacent ones (not selves)



- BUT the matchings would not always be solutions...
  - e.g., square 1 on the left might be matched with square i, but square 1 on the right might be matched with square j ≠ i

Covering with Dominos

> **Observation**: This looks like a general matching problem

> In fact, this is a *bipartite* matching problem!

> **Q**: What are the two parts?

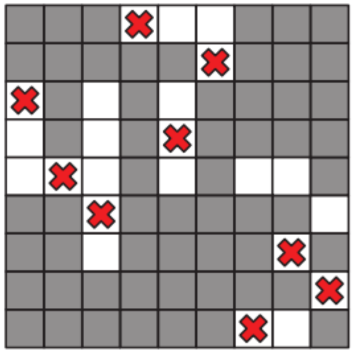
> **A**: The dark squares and the light squares
  - each domino touches *exactly one of each*

Outline for Today

- Escape Problem
- Covering with Dominos
- Token Placing
- Processor Scheduling
- Airline Scheduling

Token Placing

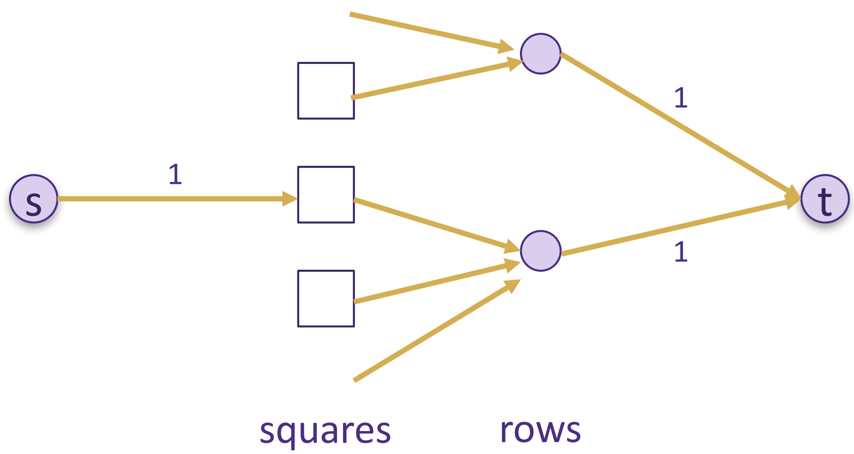
> **Problem**: Given a checker board with some squares deleted, find a set of locations to place tokens such that there is *exactly* one token in every row and one in every column.



from http://jeffe.cs.illinois.edu/teaching/algorithms/notes/24-maxflowapps.pdf

Token Placing: False Start

- First thought: what the heck even is this?
- Second thought: turn row / col restrictions into flow constraints?



Token Placing: False Start

- First thought: what the heck even is this?

- Second thought: turn row / col restrictions into flow constraints?
  - can use that to ensure only one in each row OR in each column
    - (a useful idea we can use elsewhere)
  - BUT how can we do both at once?
    - lose track of what column it came from when we flow into the row nodes
    - could put 2 units of flow into a square, one for row & one for col,
      but there is no guarantee that the solution uses 2 units
  - doesn't seem to work…

Token Placing

- **Hint**: this is a bipartite matching problem

- **Q**: Matching what and what?
- **A**: Between rows and columns
  - (row, col) pair = square on the board
  - white squares show which (row, col) pairs are allowed
  - matching because each row & col can only be used once
  - has a solution if there is a matching that uses all n rows & cols
    - the matching says on what squares you place tokens

Outline for Today

> Escape Problem
> Covering with Dominos
> Token Placing
> Processor Scheduling
> Airline Scheduling

Processor Scheduling

> **Problem**: Given n programs & m (single-core) processors along with:
> – times *after which* the programs can be started
> – times *by which* the programs must be completed
> – total processing time to complete the program
>
> **Find a schedule** for running the programs on processors that meets the deadlines.

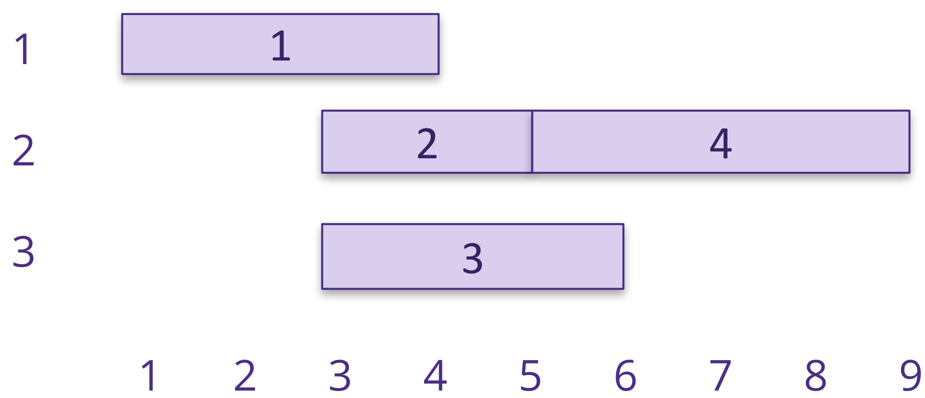
> Note that programs can be stopped, restarted, and moved between processors with no penalty.

Processor Scheduling: Example

program	start	end	req time
1	1	5	3
2	3	5	2
3	3	7	4
4	5	9	4

Find a schedule using 3 processors.

Processor Scheduling: Example



program	start	end	req time
1	1	5	3
2	3	5	2
3	3	7	4
4	5	9	4

Processor Scheduling

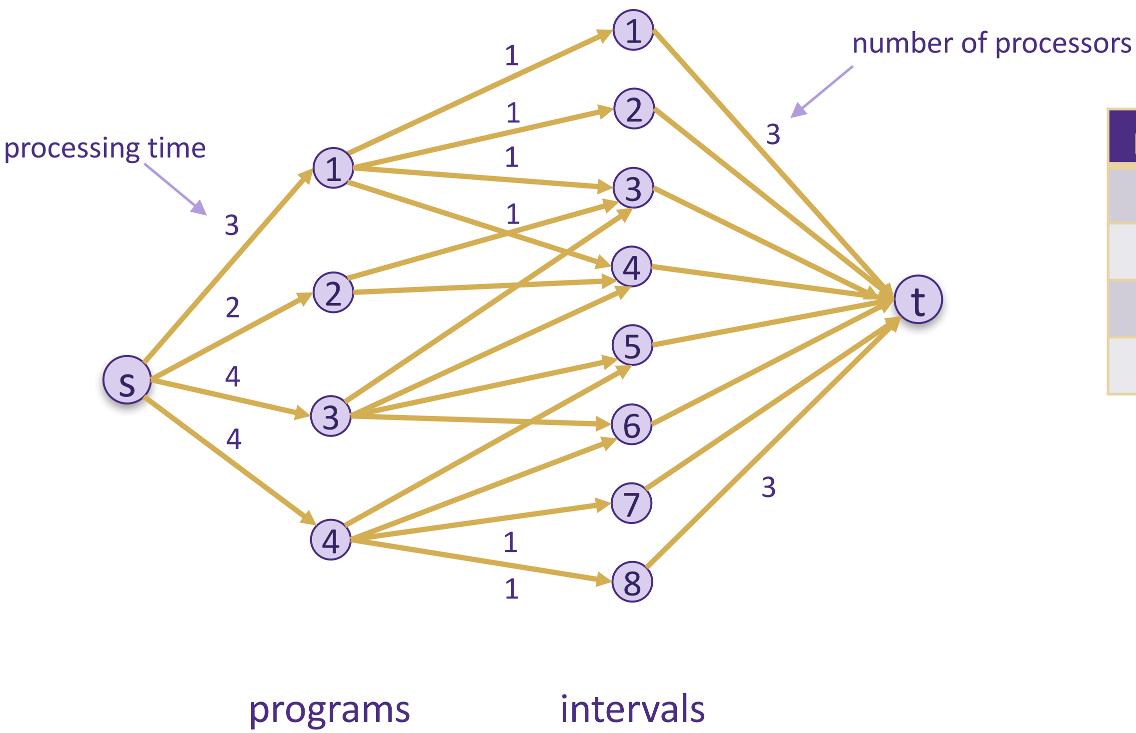
> Note that programs can be stopped, restarted, and moved between processors with no penalty.

> If the program can be started at time s and must finish by time e and requires total processing time p, we need to find an assignment of the program to p 1-second intervals in [s, e].
>   – i.e, this is essentially a matching problem
>   – units of processing for programs need to be matched to intervals of available time on the processors
>   – in this case, p units of time matched to intervals in [s, e]

Processor Scheduling

- Can make this more efficient...
  - as described, we have np nodes on the left and mT nodes on the right, where T = last finish time – earliest start time
- Can have only one node for each program and 1-second interval
  - allow a program requiring p units of time to be assigned to p intervals
  - allow each interval to be assigned m different programs (for m processors)

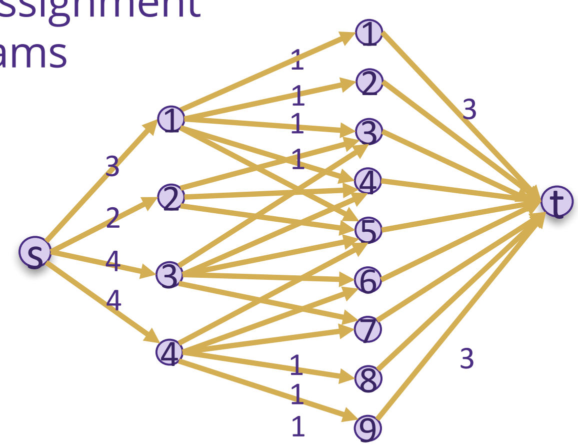
Processor Scheduling: Example



program	start	end	req time
1	1	5	3
2	3	5	2
3	3	7	4
4	5	9	4

Processor Scheduling

- Max flow of value equal to sum of processing times gives an assignment of each program to a set of *distinct* times such that the total assigned to each time is at most m
  - can choose program ~> processor assignment arbitrarily since we can move programs around with no penalty



Processor Scheduling

- Can make this more efficient...
- Can have only one node for each program and 1-second interval
  - allow a program requiring p units of time to be assigned to p intervals
  - allow each interval to be assigned m different programs (for m processors)
- Still pseudo-polynomial due to use of 1-second intervals
  - need T (= last finish time – earliest start time) intervals
  - instead, break into intervals containing no program start or end
  - time is completely fungible within each of these intervals

Outline for Today

> Escape Problem
> Covering with Dominos
> Token Placing
> Processor Scheduling
> Airline Scheduling

Airline Scheduling

- **Problem**: Given a collection of n flights with departure times (s_j) and arrival times (e_j), determine whether there it is possible to schedule all of the flights using only m crews (pilots, etc.)

- Can easily generalize this to require a certain amount of preparation time ($t_{i,j}$) between particular pairs of flights
  - pilots and attendants might need breaks
  - they might also need to transit from one city to another

Airline Scheduling: Example

> Suppose we have these **three** flights and **two** crews:

flight	start	end
1	0	2
2	3	4
3	3	4

delay	1	2	3
1		2	1
2	2		3
3	1	3	

Airline Scheduling: Example

flight	start	end
1	0	2
2	3	4
3	3	4

delay	1	2	3
1		2	1
2	2		3
3	1	3	

- Flights 2 & 3 cannot be serviced by 1 crew
  - they are flying at the same time
- Flights 1 & 2 cannot be serviced by 1 crew :
  - 2 needs to start 1 hour after 1 but takes 2 hours to prepare
- Flights 1 & 3 can be serviced by 1 crew

Airline Scheduling: Example

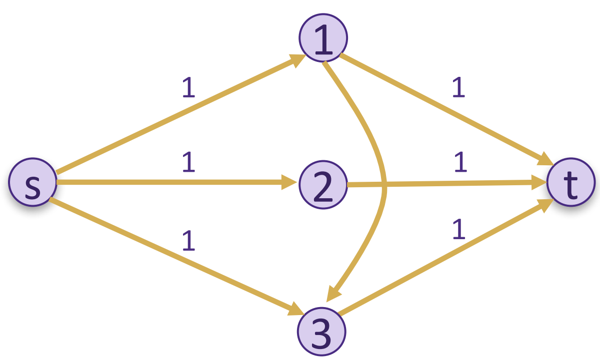
flight	start	end
1	0	2
2	3	4
3	3	4

delay	1	2	3
1		2	1
2	2		3
3	1	3	

> Optimal schedule:
  - crew 1 runs 1 and then 3
  - crew 2 runs 2

Airline Scheduling

> Model single crew as a network flow where flow of 1 unit describes a schedule for one crew:

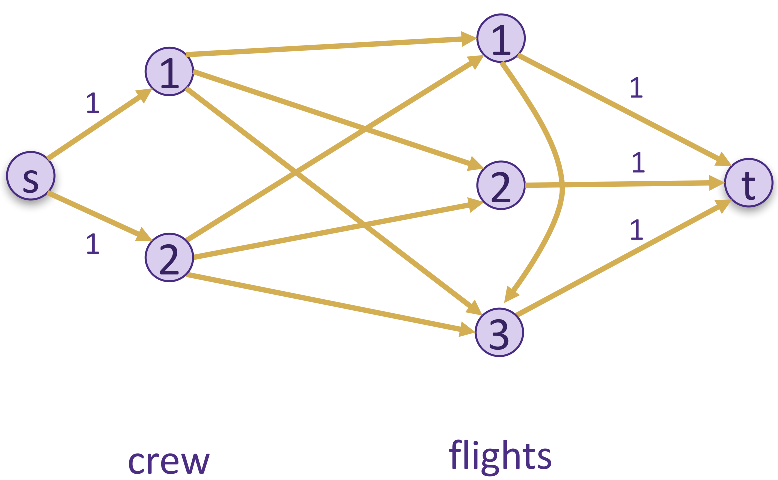


Arrow from i to j if j can be servied after i (i.e., if $s_j \geq e_i + t_{i,j}$)

Paths from s to t are in 1-to-1 correspondence with valid schedules for one crew. (I.e., really a path problem so far.)

Airline Scheduling: False Start

> Identify each flow with an individual crew....



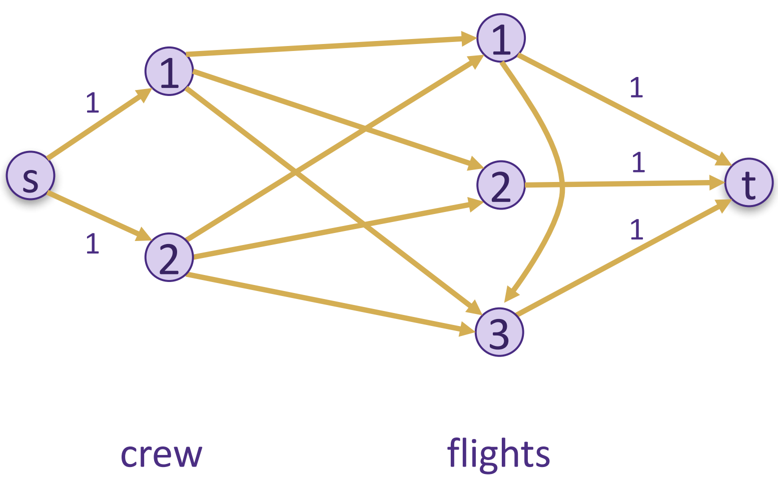
Flow of value of m schedules all crews

Q: What is wrong with this?

A: Doesn't tell us whether all flights are actually scheduled!

Airline Scheduling: False Start

> Identify each flow with an individual crew....



Need to ensure that some crew's path goes through every node

Saw how to set upper bounds on flow through a node, but what we really want here are *lower bounds* (of 1) on the flow

We will see how to support lower bounds **next lecture**...

Airline Scheduling v2

> **Problem**: Given a collection of n flights with departure times (s_j) and arrival times (e_j), determine the minimum number of crews needed to service all of the flights.

> **Q**: How do we solve this?

> **A**: Binary search
  - answer is between 0 and n (inclusive)
  - previous algorithm says if we need more or fewer crews