Week 4 - Monday

COMP 4500

Last time

- What did we talk about last time?
- Non-recursive DFS
- Running time for BFS and DFS
- Determining bipartiteness
- Connectivity in directed graphs
- Started topological sorting

Questions?

Assignment 2

Formatting assignments

- Assignments must be **typewritten**
  - That means not handwritten, no matter how you digitize it
- I give some leeway for mathematical content, but things like superscripting and subscripting really matter for math
  - $10n$ is **not** the same as 10^n
- I don't grade grammar or spelling explicitly, but if your sentences are too confusing to understand, it might cost you points

Tips for Word

- Assignment 1 had more mathematical notation than most assignments will, but more is still coming
- If you're typing out math with exponents, subscripts, or fractions, I highly recommend using the **Equation** tool on the **Insert** tab
  - It's not as easy to use as LaTeX (but you can actually use LaTeX commands and have them automatically converted)
- A cheat for quick math is to use **Ctrl-+** for superscript and **Ctrl-=** for subscript
- Please use · instead of * for multiplication

Tips for LaTeX

- Use **`\cdot`** for · instead of **`*`**
- Superscripts and subscripts need braces for multiple characters
  - **`$10^n$`** gives 10^n
  - **`$n^100$`** gives n^100, use **`$n^{100}$`** for n^{100}
- Regular quotes and single quotes are smart quotes for the right side of a quoted expression
- Use one or two backticks (`) for smart quotes on the left:
  - **`"goats"`** is rendered as ”goats”
  - **`` ``goats'' ``** is rendered as “goats”
- Common mathematical functions have their own commands
  - Use **`\log`** for log and **`\sin`** for sin

Logical warmup

- An epidemic has struck the Island of Knights and Knaves
  - Sick Knights always lie
  - Sick Knaves always tell the truth
  - Healthy Knights and Knaves are unchanged
- During the epidemic, a Nintendo Switch was stolen
- There are only three possible suspects: Jacob, Karl, and Louie
- They are good friends and know which one actually stole the Switch
- Here is part of the trial's transcript:
  - **Judge (to Jacob):** What do you know about the theft?
  - **Jacob:** The thief is a Knave
  - **Judge:** Is he healthy or sick?
  - **Jacob:** He is healthy
  - **Judge(to Karl):** What do you know about Jacob?
  - **Karl:** Jacob is a Knave.
  - **Judge:** Healthy or sick?
  - **Karl:** Jacob is sick.
- The judge thought a while and then asked Louie if he was the thief. Based on his yes or no answer, the judge decided who stole the Switch.
- Who was the thief?

Quiz review

- Use a proof by induction to prove the following claim.

$$1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}, n \in \mathbb{Z}, n \geq 1$$

Topological Sort

Directed acyclic graph

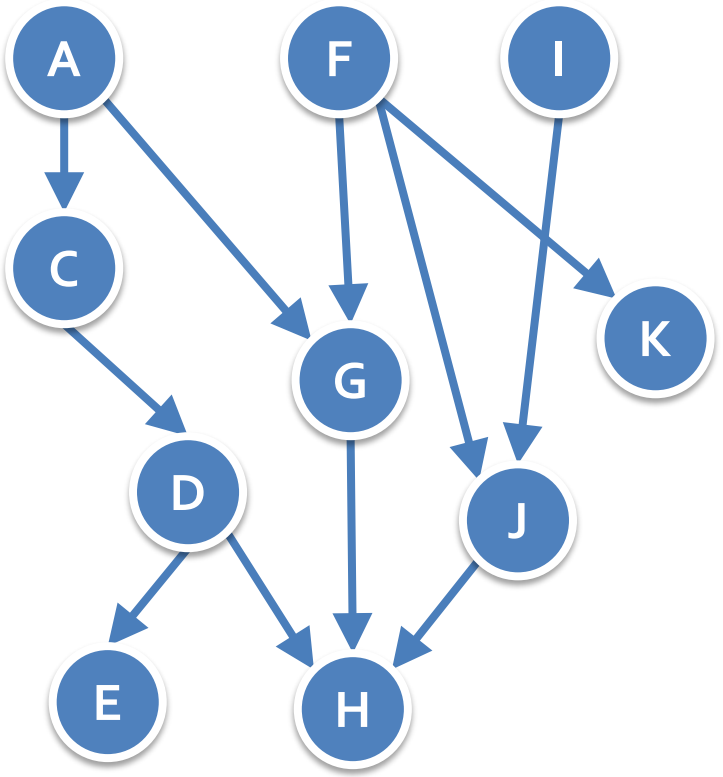
- A directed acyclic graph (DAG) is a directed graph without cycles in it
- These can be used to represent dependencies between tasks
- An edge flows from the task that must be completed first to a task that must come after
- A cycle in such a graph would mean there was a circular dependency
- By running topological sort, we discover if a directed graph has a cycle, as a side benefit

Topological sort

- A **topological sort** gives an ordering of the tasks such that all tasks are completed in dependency ordering
- In other words, no task is attempted before its prerequisite tasks have been done
- There are usually multiple legal topological sorts for a given DAG

Topological sort

- Give a topological sort for the following DAG:



- A F I C G K D J E H

Topological sort algorithm

- Create list ***L***
- Add all nodes with no incoming edges into set ***S***
- While ***S*** is not empty
  - Remove a node ***u*** from ***S***
  - Add ***u*** to ***L***
  - For each node ***v*** with an edge ***e*** from ***u*** to ***v***
    - Remove edge ***e*** from the graph
    - If ***v*** has no other incoming edges, add ***v*** to ***S***
- If the graph still has edges
  - Print "Error! Graph has a cycle"
- Otherwise
  - Return ***L***

Three-Sentence Summary of Greedy Algorithms, the Interval Scheduling Problem, and Scheduling to Minimize Lateness

Greedy Algorithms

Greedy algorithms

- Greedy algorithms always take the next step that looks best locally
  - Many problems do not have this property
- Sometimes this is referred to as **optimal substructure**
  - An optimal solution can be built by combining optimal solutions to smaller problems
- The book proves that a greedy approach is optimal in two different ways:
  - The greedy algorithm stays ahead
  - An exchange argument

Interval scheduling

- In the interval scheduling problem, some resource (a phone, a motorcycle, a toilet) can only be used by one person at a time
- People make requests to use the resource for a specific time interval [s, f]
- The goal is to schedule as many uses as possible
- There's no preference based on who or when the resource is used

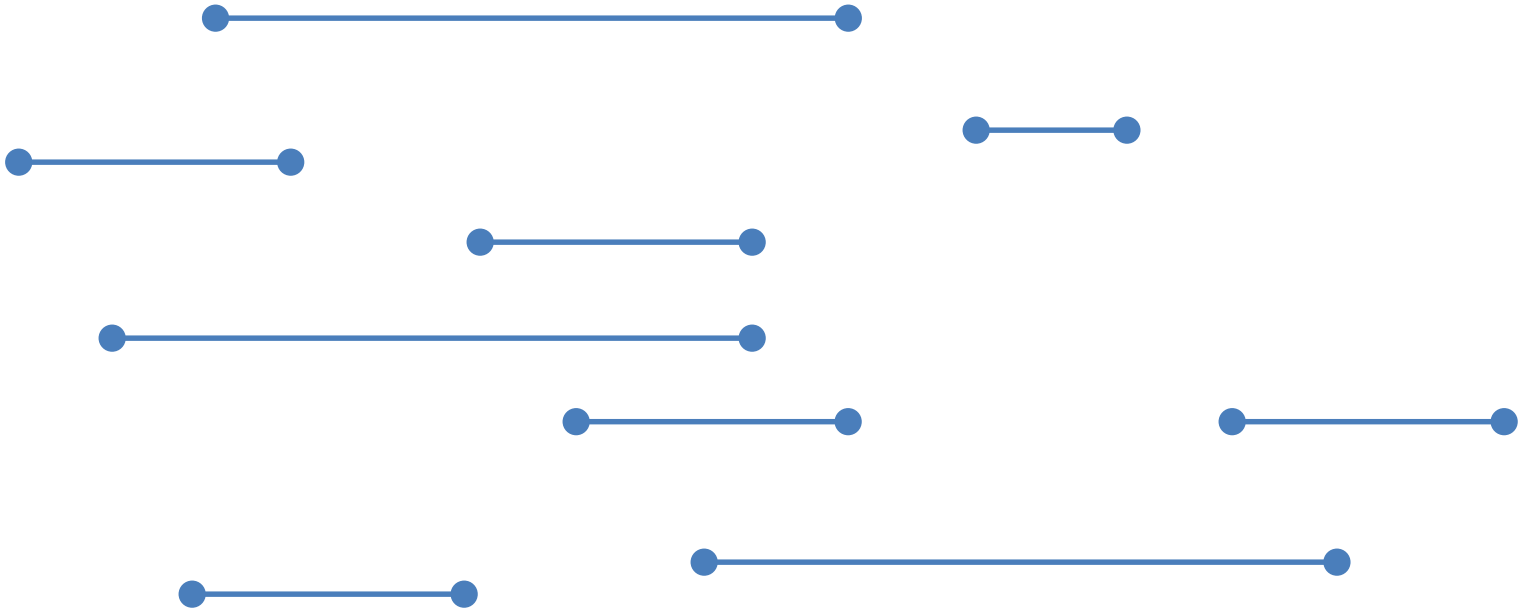
Designing the algorithm

- We (magically) know it's going to be greedy
- Which interval do we select next?
  - The one that starts earliest?
    - No.
  - The shortest?
    - Better, but still no.
  - The interval that overlaps with the fewest other intervals?
    - Still no.
- A choice that leads to an optimal algorithm is choosing the interval that **finishes** first

Interval scheduling algorithm

- Interval scheduling can be done with a greedy algorithm
- While there are still requests that are not in the compatible set
  - Find the request *r* that ends earliest
  - Add it to the compatible set
  - Remove all requests *q* that overlap with *r*
- Return the compatible set

Interval scheduling example



Proving optimality

- First of all, it's clear that our algorithm returns a compatible set of requests, let's call it ***A***
- Imagine some optimal solution ***O***
- If we can show that $|A| = |O|$, we are done
- We want to show that our algorithm **stays ahead** of (does no worse than) the algorithm that builds ***O***
- Let $i_1, i_2, \ldots, i_k$ be the requests in ***A***, in the order added
- Let $j_1, j_2, \ldots, j_m$ be the requests in ***O***, in left to right order
- For any request ***r***, let ***s***(***r***) be its starting time and ***f***(***r***) be its finishing time

For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- Proof by induction:
  - Basis case: $r = 1$
    - Our algorithm starts with i_1 being the request with the smallest finishing time. j_1 cannot have a smaller one.
  - Induction step: Assume $f(i_r) \leq f(j_r)$ for $r = x$, where $x \geq 1$
    - Consider $f(i_{x+1})$. Because intervals in O are ordered left to right, $f(j_x) \leq s(j_{x+1})$. By the induction hypothesis, $f(i_x) \leq f(j_x)$. Thus, $f(i_x) \leq s(j_{x+1})$. Since we always select a request with the smallest finish time, and j_{x+1} is a legal request to select, it must be the case that $f(i_{x+1}) \leq f(j_{x+1})$. ■

The greedy algorithm returns an optimal set *A*

- Proof by contradiction:
  - Suppose that ***A*** is not optimal. Then, ***O*** must have more requests. In other words, $m > k$. By the proof on the prior slide, $f(i_k) \leq f(j_k)$. Since $m > k$, there is at least one request j_{k+1} in ***O***. To be legal, j_{k+1} starts after j_k ends, and thus also after i_k ends. If we remove all of the requests that are not compatible with $i_1, i_2, \ldots i_k, j_{k+1}$ must still be available. But our greedy algorithm stopped with request i_k, when it's supposed to stop when there are no more requests left to consider: contradiction.

Running time

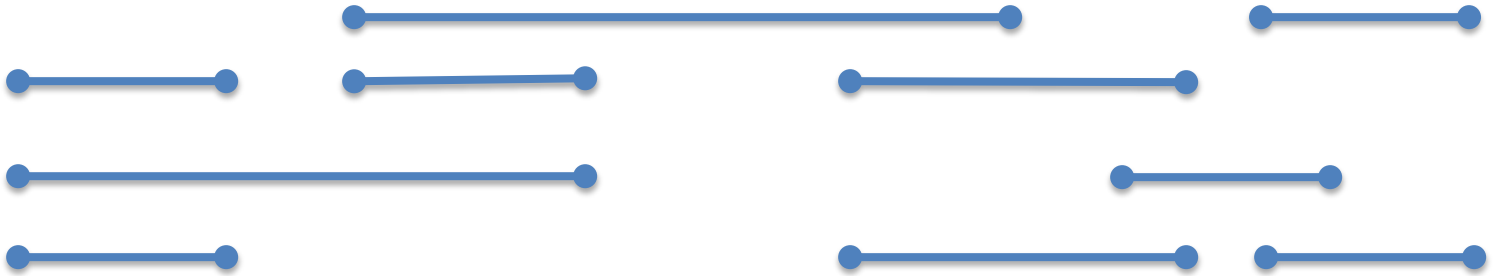
- First, we sort the n requests in order of finishing time
  - The best comparison-based sort takes $O(n \log n)$
- We scan through the n sorted requests again and make an array S of length n such that $S[i]$ contains the starting value of i, $s(i)$
  - $O(n)$ time
- Our algorithm selects the first interval in our list sorted on finishing time. We then move through array S until we find the first interval j such that $s(j) \geq$ the finishing time selected. We add it. We continue the process until we have moved through the entire array S.
  - $O(n)$ time
- Total time: $O(n \log n)$

Scheduling all intervals

- What if all intervals need to be scheduled?
  - Example: Lectures are intervals and resources are classrooms
  - Example: Roasting pigs are intervals and resources are fire pits
- The problem is to find the minimum number of resources needed to satisfy all intervals
- The book calls this problem **interval partitioning**

Interval partitioning: visualization

- Intervals to schedule (arranged unhelpfully):



- Intervals to schedule (arranged helpfully):

- The depth ***d*** of a set of intervals is the maximum number that pass a single point on the time-line

Interval partitioning algorithm

- Sort intervals by their start times
- Let $I_1, I_2, \ldots, I_n$ be the ordered intervals
- For $j = 1, 2, 3, \ldots, n$
  - For each interval I_i that precedes I_j in sorted order and overlaps it
    - Exclude the label of I_i from consideration for I_j
  - If there is any label from $\{1, 2, \ldots, d\}$ that has not been excluded
    - Assign that label to I_j
  - Else
    - Leave I_j unlabeled

Interval partitioning correctness

- **Claim:** In our algorithm, every interval will be assigned a label, and no two overlapping intervals will receive the same label.
- **Proof:** Consider interval I_j, and suppose there are t intervals earlier in the sorted order that overlap it. These t intervals with I_j form $t + 1$ intervals that pass over a common point on the time-line. Thus, $t + 1 \leq d$ and $t \leq d - 1$. Thus, there must be at least one label left to be assigned to I_j.

Proof continued

- To show that no two overlapping intervals are assigned the same label, consider two intervals I and I' that overlap. Suppose that I precedes I' in the sorted order. When I' is considered by the algorithm, I is in the set of intervals whose labels are excluded. ■

Upcoming

Next time…

- Finish interval partitioning
- Scheduling to minimize lateness
- Shortest path
- Minimum spanning tree

Reminders

- **Work on Assignment 2**
  - **Due Friday before midnight**
- Read sections 4.4 and 4.5
- Exam 1 is next Monday
  - Review will be on Friday