Week 2 - Friday

COMP 4500

Last time

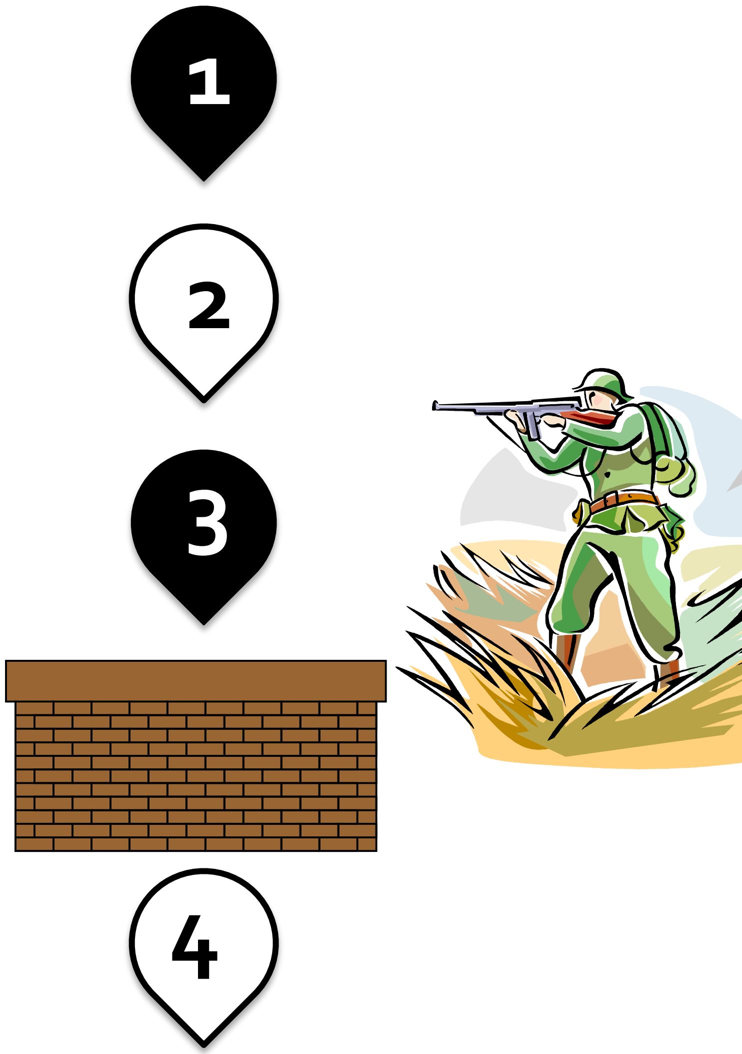
- What did we talk about last time?
- More proof techniques
- Asymptotic orders of growth
- Started stable marriage

Questions?

Assignment 1

Logical warmup

- Four men are standing in front of a firing-squad
- #1 and #3 are wearing black hats
- #2 and #4 are wearing white hats
- They are all facing the same direction with a wall between #3 and #4
- Thus,
  - #1 sees #2 and #3
  - #2 sees #3
  - #3 and #4 see no one
- The men are told that two white hats and two black hats are being worn
- The men can go if one man says what color hat he's wearing
- No talking is allowed, with the exception of a man announcing what color hat he's wearing.
- Are they set free? If so, how?



Stable Marriage

Imagine *n* men and *n* women

- All 2***n*** people want to get married
- All of them are *willing* to marry any of the n members of the opposite gender
- Each woman has ranked all ***n*** men in order of preference
- Each man has ranked all ***n*** women in order of preference
- We want to match them up so that the marriages are **stable**

Stability

- Consider two marriages:
  - Anna and Bob
  - Caitlin and Dan
- This pair of marriages is unstable if
  - Anna likes Dan more than Bob **and** Dan likes Anna more than Caitlin **or**
  - Caitlin likes Bob more than Dan **and** Bob likes Caitlin more than Anna
- We want to arrange all ***n*** marriages such that none are unstable

Gale-Shapley Pseudocode

- While there is man m who is free and hasn't proposed to every woman
  - Choose any such man m
  - Let w be the highest-ranked woman in m's preferences that m hasn't proposed to yet
  - If w is free then
    - (m, w) become engaged
  - Else w is engaged to some man called m'
    - If w prefers m' to m
      - m remains free
    - Else
      - (m,w) become engaged
      - m' becomes free
- Return the set of engaged pairs

Observations

- Once a woman is engaged, she'll stay engaged
  - Maybe her engagement will change to a man she likes more, but she will never become free again
- The sequence of women that a particular man proposes to will get lower and lower on his preference list

Progress

- We want to **bound** the time that an algorithm takes
- Sometimes that means coming up with some kind of **indirect** measurement of the operations
- We can define $\boldsymbol{P}(t)$, the **progress** at time t, as the set of unique proposals of m to w on the t^{th} iteration of the algorithm
- Note that on every iteration, a unique proposal (m, w) happens, so the size of $\boldsymbol{P}(t + 1)$ is always one more than $\boldsymbol{P}(t)$

Running time

- The algorithm runs at most n^2 iterations of the While loop.
- **Proof:**
  - No men will propose after they have proposed to all the women.
  - There are a maximum of n^2 ways for any man to propose to any woman.
  - At each iteration, the progress increases.
  - Thus, it's impossible for the algorithm to run more than n^2 iterations.

  ∎

If m is free, there is a woman he hasn't proposed to

- **Proof by contradiction:**
  - Suppose that m is free but has already proposed to every woman.
  - We have already established that a woman can never become unengaged once she's been proposed to.
  - Since m has proposed to all women, they're all engaged.
  - But then there would be n women who are engaged to n different men.
  - Since m is one of those n men, he must not be free, which is a contradiction.

■

Everyone is matched when the algorithm terminates

- **Proof by contradiction:**
  - Suppose that there is at least one man *m* who is unmatched at the end of the algorithm.
  - He must have proposed to every woman or the While loop would not have terminated.
  - However, this contradicts the previous proof that any free man must have a woman he hasn't proposed to.

■

The algorithm gives a stable matching

- **Proof by contradiction:**
  - Suppose that the matching is not stable.
  - Thus, there are pairs (*m*, *w*) and (*m*', *w*') such that *m* prefers *w*' and *w*' prefers *m*.
  - It must be the case that *m*'s last proposal was to *w*.
  - Case 1: *m* never proposed to *w*'
    - Since *m* proposed to women in descending order of preference, he must prefer *w* more than *w*', a contradiction.
  - Case 2: *m* did propose to *w*'
    - If so, *w*' preferred some later proposer *m*'' to *m*.
    - But for *w*' to end up with *m*', *m*' = *m*'' or *m*' is someone she preferred even more than *m*'', and thus more than *m*, a contradiction.
  - Since all cases lead to contradictions, the matching must be stable.

■

Five Representative Problems

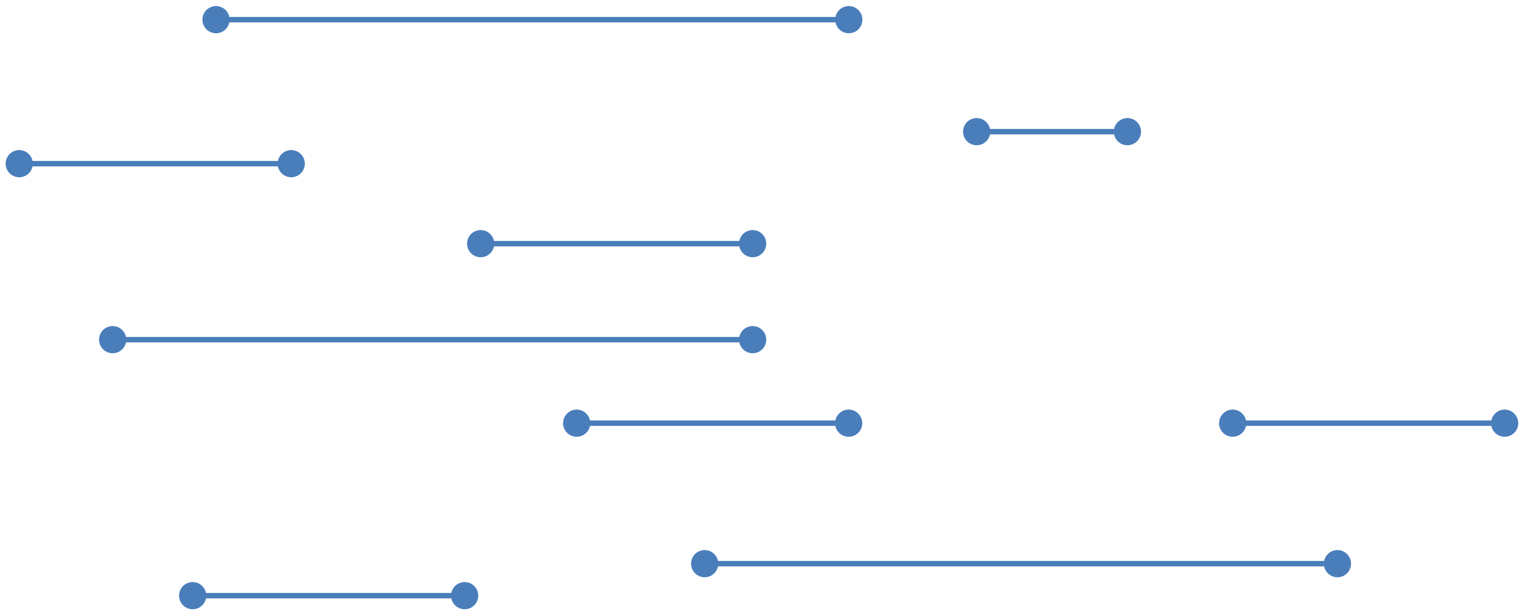
Interval scheduling

- 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.

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



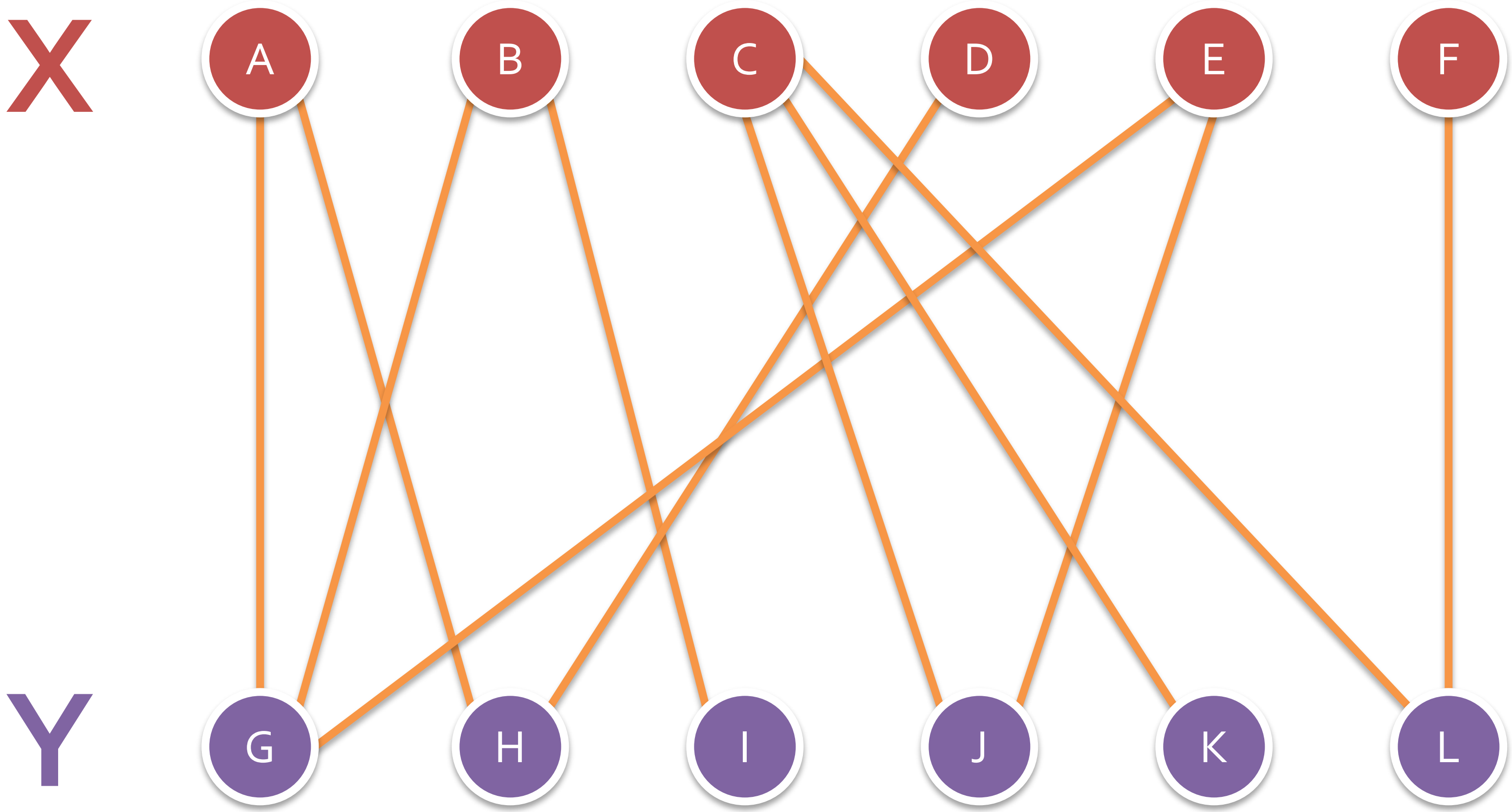
Weighted interval scheduling

- The **weighted interval scheduling** problem extends interval scheduling by attaching a weight (usually a real number) to each request
- Now the goal is not to maximize the **number** of requests served but the total **weight**
- Our greedy approach is worthless, since some high value requests might be tossed out
- We could try all possible subsets of requests, but there are **exponential** of those
- **Dynamic programming** will allow us to save parts of optimal answers and combine them efficiently

Bipartite graphs

- A bipartite graph is one whose nodes can be divided into two disjoint sets X and Y
- There can be edges between set X and set Y
- There are no edges inside set X or set Y
- A graph is bipartite if and only if it contains no odd cycles

Bipartite graph



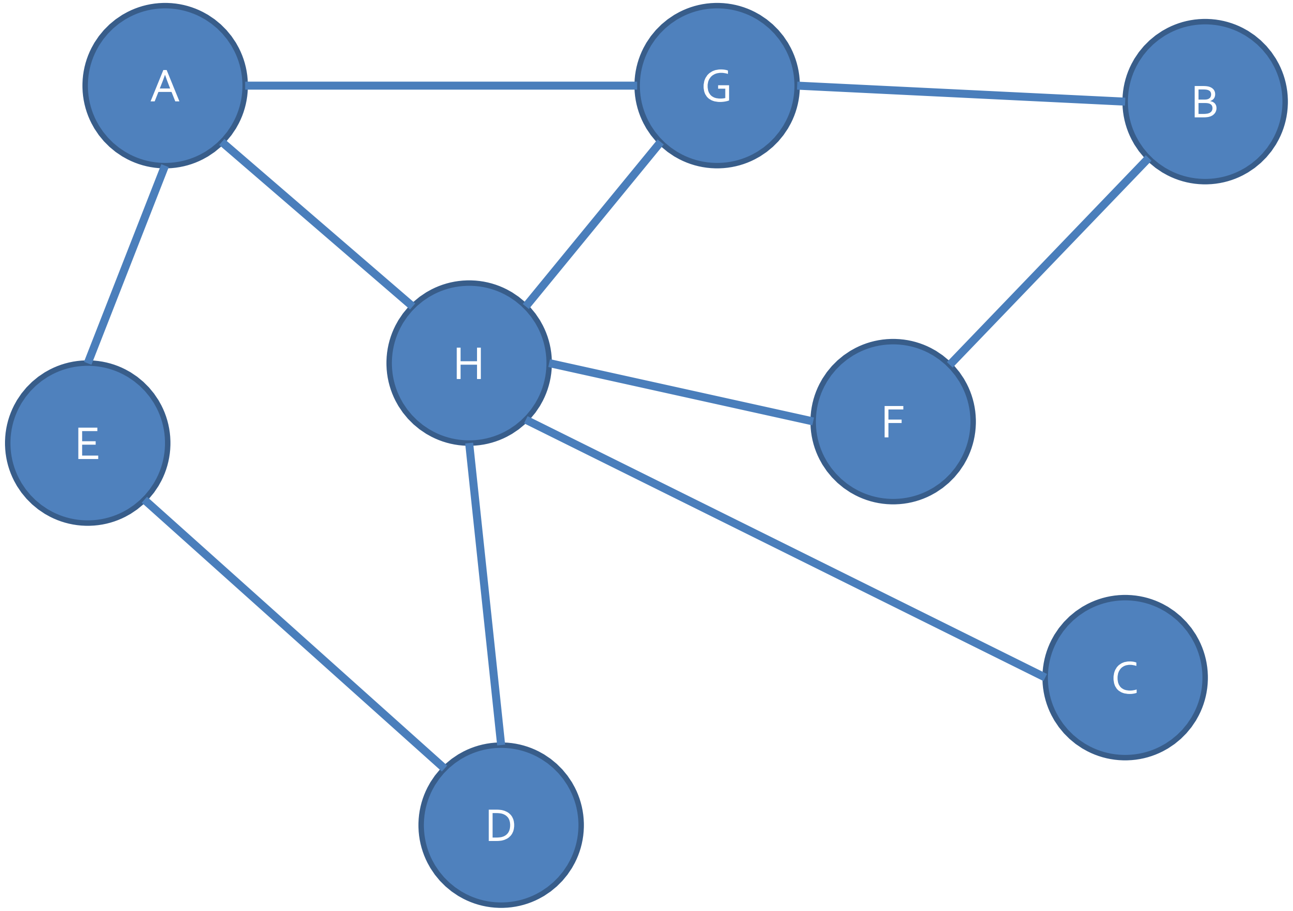
Bipartite matching

- A **perfect matching** is when every node in set X and every node in set Y is matched
- It is not always possible to have a perfect matching
- We can still try to find a **maximum matching** in which as many nodes are matched up as possible
- Our algorithm will use the idea of an augmenting path, which is useful in many network flow problems
- This technique is neither greedy nor dynamic programming

Independent set

- Independent set is another graph problem
- Given an undirected graph, find the largest set of nodes that are not connected to each other
- Doesn't sound too bad, right?
- Practical application:
  - Nodes represent friends of yours
  - An edge between those two nodes means they hate each other
  - What's the largest group of friends you could invite to a party if you don't want any to hate each other?

Independent set example



NP-complete

- Independent set is NP-complete
- That means:
  - The best algorithm we know is exponential (try all subsets of vertices)
  - All other NP-complete problems can be turned into it
  - Even all polynomial time problems can be turned into it (though it's not always easy to see how)

Solving interval scheduling with independent set

- Take your interval scheduling problem and make all the requests nodes
- If the nodes overlap, put an edge between them
- Then, run your independent set algorithm
- Magically, you'll get exactly those nodes corresponding to the largest set of non-overlapping requests

Solving bipartite matching with independent set

- A little confusing!
- Make a new graph where there's a node corresponding to every edge from the bipartite graph
- Now, connect every node in the new graph to every other node (which was an edge) that shared endpoints in the original graph
- Running an independent set algorithm will now pick the largest set of nodes (which were edges) such that none of the nodes are connected
- Thus, only edges in the original graph will be selected if they don't share endpoints

Competitive facility location

- Imagine that you have a graph where nodes represent locations
- There are edges between locations that are "too close" to both have coffee shops
- Each node has a value associated with it, giving how much coffee you can sell
- What if there are two companies that each take turns picking a location to build their next coffee shop?
- What algorithm should either company follow to guarantee the most value? Or to guarantee at least a certain amount of value?

PSPACE-complete

- The competitive facility location problem is PSPACE-complete
  - Problems that can be solved using only polynomial space and unbounded time
- It is believed to be even harder than NP-complete
- Even though coffee chains don't play games like this, PSPACE-complete problems include generalizations of:
  - Almost every board game
  - Game theory problems
  - Serious AI problems

Three-sentence summary of an efficient solution to Stable Marriage

Implementing Stable Marriage

Arrays

- An array is a random access list data structure available in many programming languages
- An array of length n has the following properties:
  - Retrieving the i^{th} element in the list takes $O(1)$ time
  - Checking to see if an element appears in an unordered array takes $O(n)$ time
  - Checking to see if an element appears in a sorted array takes $O(\log n)$
  - Adding or removing elements can take $O(n)$ time to move elements over or resize the array

Linked lists

- A linked list is a sequential access list data structure available in most programming languages
- A list has the following properties:
  - Retrieving the $\boldsymbol{i}^{th}$ element in the list takes O(***i***) time
  - Checking to see if an element appears may always take O(***n***) time
  - Adding or removing elements from the beginning and end of the linked list usually takes O(1) time

Gale-Shapley Pseudocode

- While there is man *m* who is free and hasn't proposed to every woman
  - Choose any such man *m*
  - Let *w* be the highest-ranked woman in *m*'s preferences that *m* hasn't proposed to yet
  - If *w* is free then
    - (*m*, *w*) become engaged
  - Else *w* is engaged to some man called *m*'
    - If *w* prefers *m′* to *m*
      - *m* remains free
    - Else
      - (*m*,*w*) become engaged
      - *m*' becomes free
- Return the set of engaged pairs

Steps in the loop

- Each iteration of the loop, we need to do four things efficiently:
  1. Find a free man ***m***
  2. Find the highest-ranked woman ***w*** that ***m*** hasn't proposed to
  3. See if ***w*** is currently engaged and, if so, her current partner
  4. For a woman ***w***, decide whether she prefers ***m*** or ***m'***

Finding a free man and his next proposal

- If we keep a linked list of free men, we can find a free man in constant time
- Each man has a list (presumably stored as an array) of his preferences
- We only need to keep the index of the next woman he should propose to
- Thus, we can keep all of the indexes for all of these men in a single array and increment the appropriate index whenever a man proposes, in constant time

Finding a woman's partner and her preferences

- We can keep a separate array that lists which man each woman is engaged to
  - Most languages provide **`null`** or a similar value to represent no current partner
- Before the algorithm, we can create an ***n*** x ***n*** array of ranks called **`ranking`**, where **`ranking`**[*w*][*m*] gives the ***w***'s ranking of ***m***
- With this array, we can look up ***w***'s ranking of ***m*** and ***m'*** in constant time

Total running time

- Before, we proved that we needed a maximum of n^2 iterations of the While loop to solve the Stable Marriage problem
- We just demonstrated that we can do $\Theta(n^2)$ work before the loop and then do constant work inside each iteration
- Thus, the total work is $\Theta(n^2) + \Theta(n^2)$, which is $\Theta(n^2)$

Upcoming

Next time…

- Common running times
- Worked exercises
- Proofs by induction

Reminders

- Read section 2.4
- Work on Assignment 1
  - Due next Friday by midnight